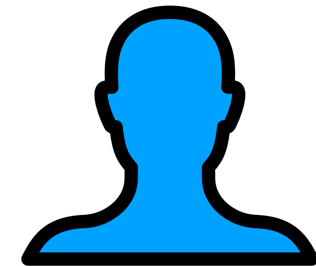
Secure Caches for Compartmentalized Software

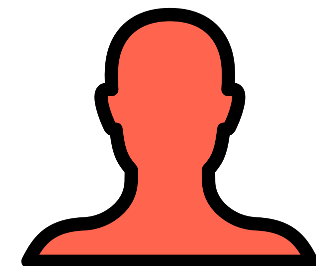
Kerem Arikan
Huaxin Tang, Williams Zhang-Cen,
Yu David Liu, Nael Abu-Ghazaleh, Dmitry Ponomarev

Background

Monolithic Software

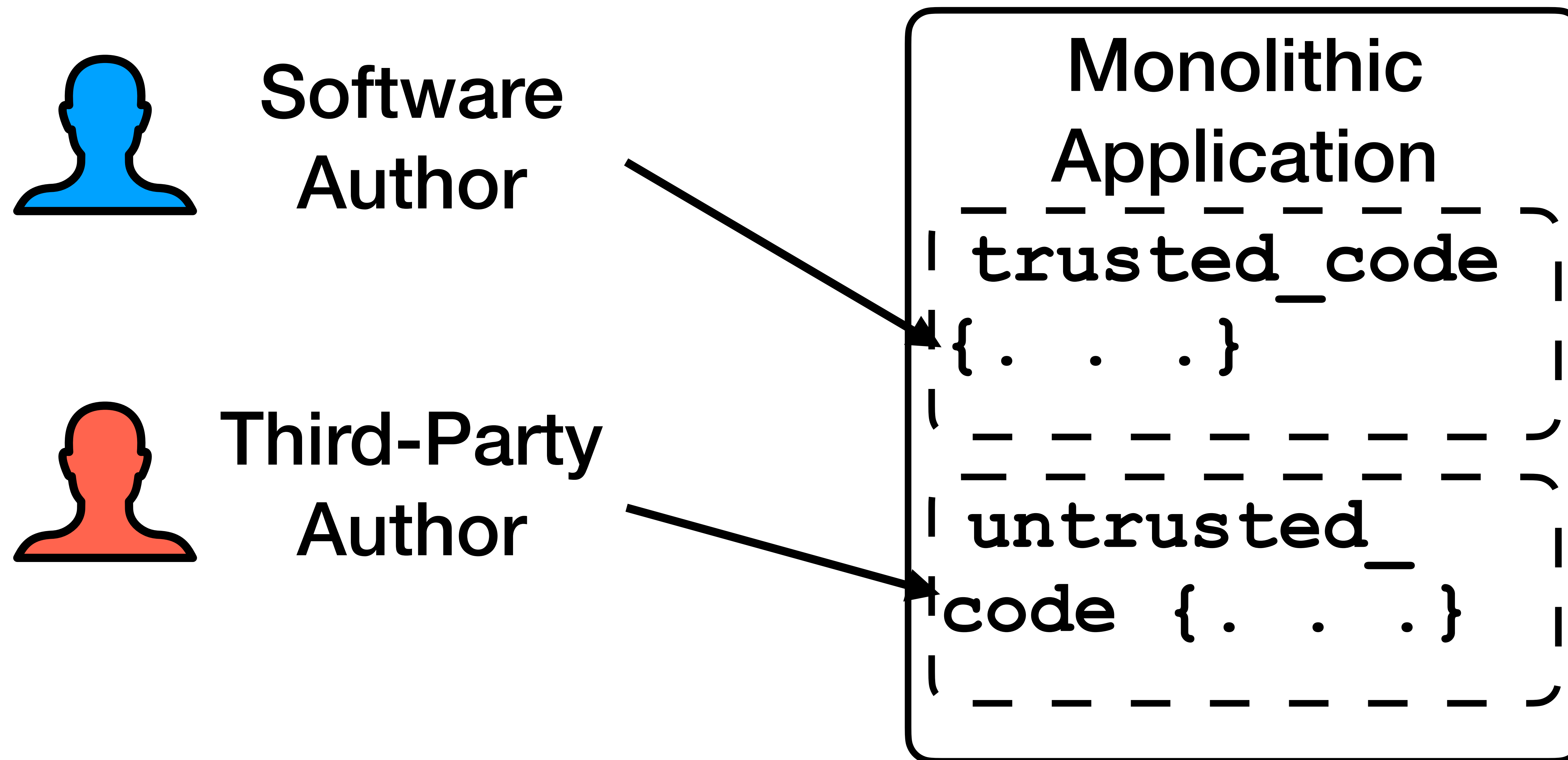


Software Author

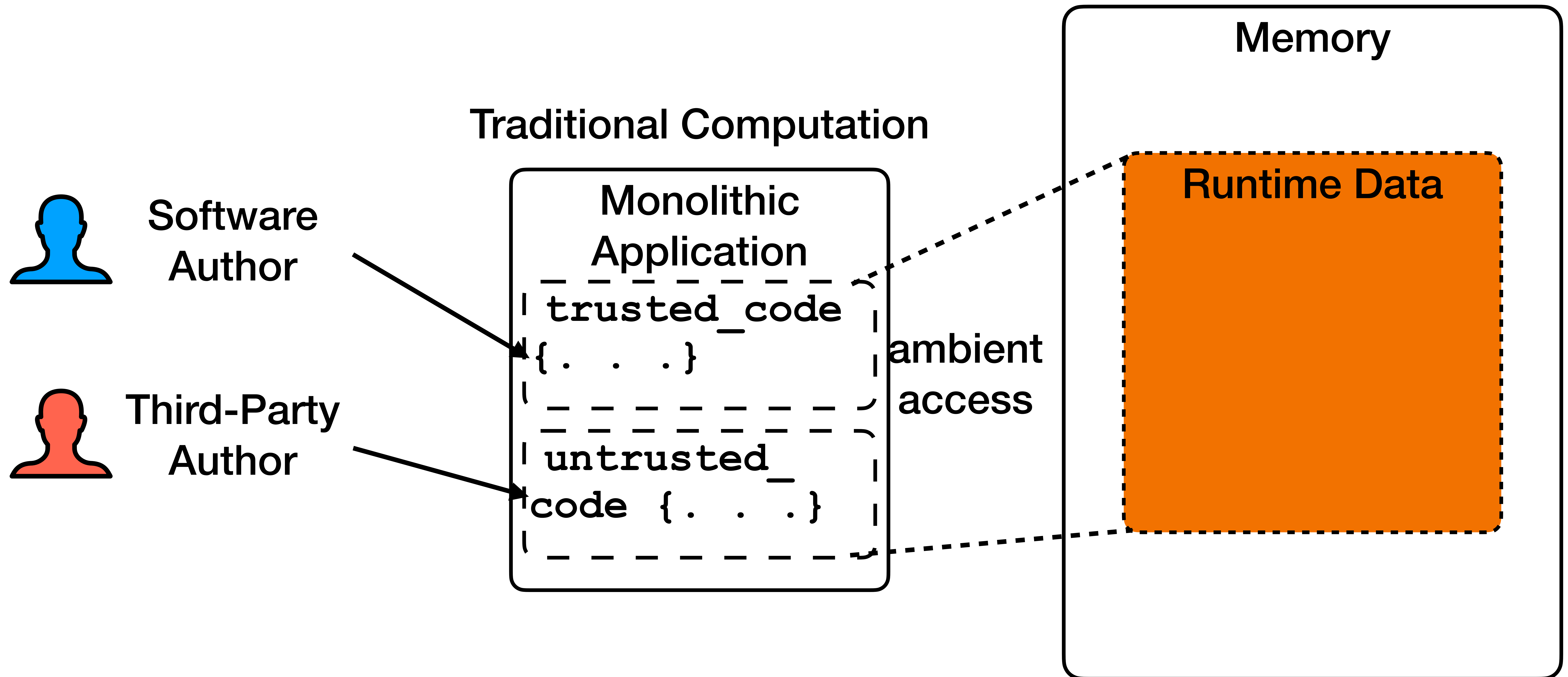


Third-Party Author

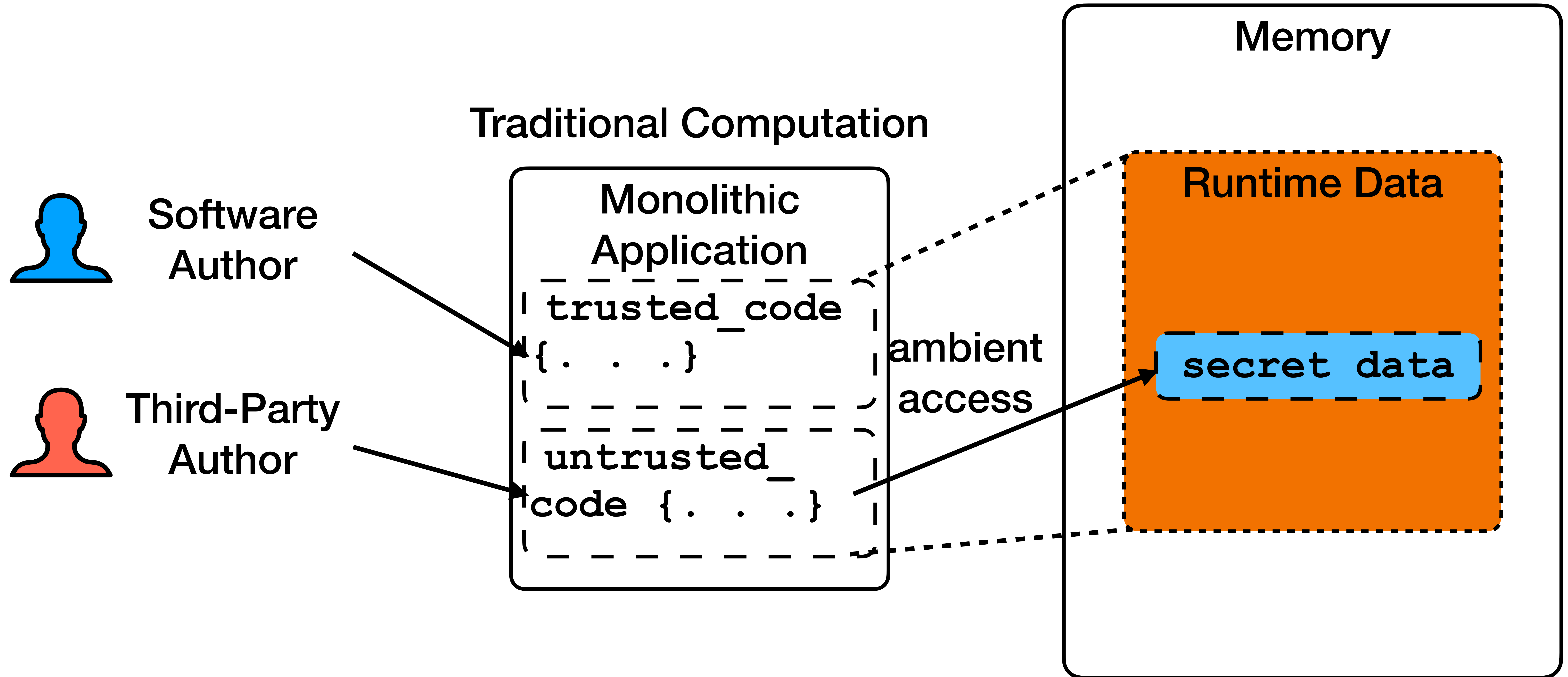
Monolithic Software



Monolithic Software

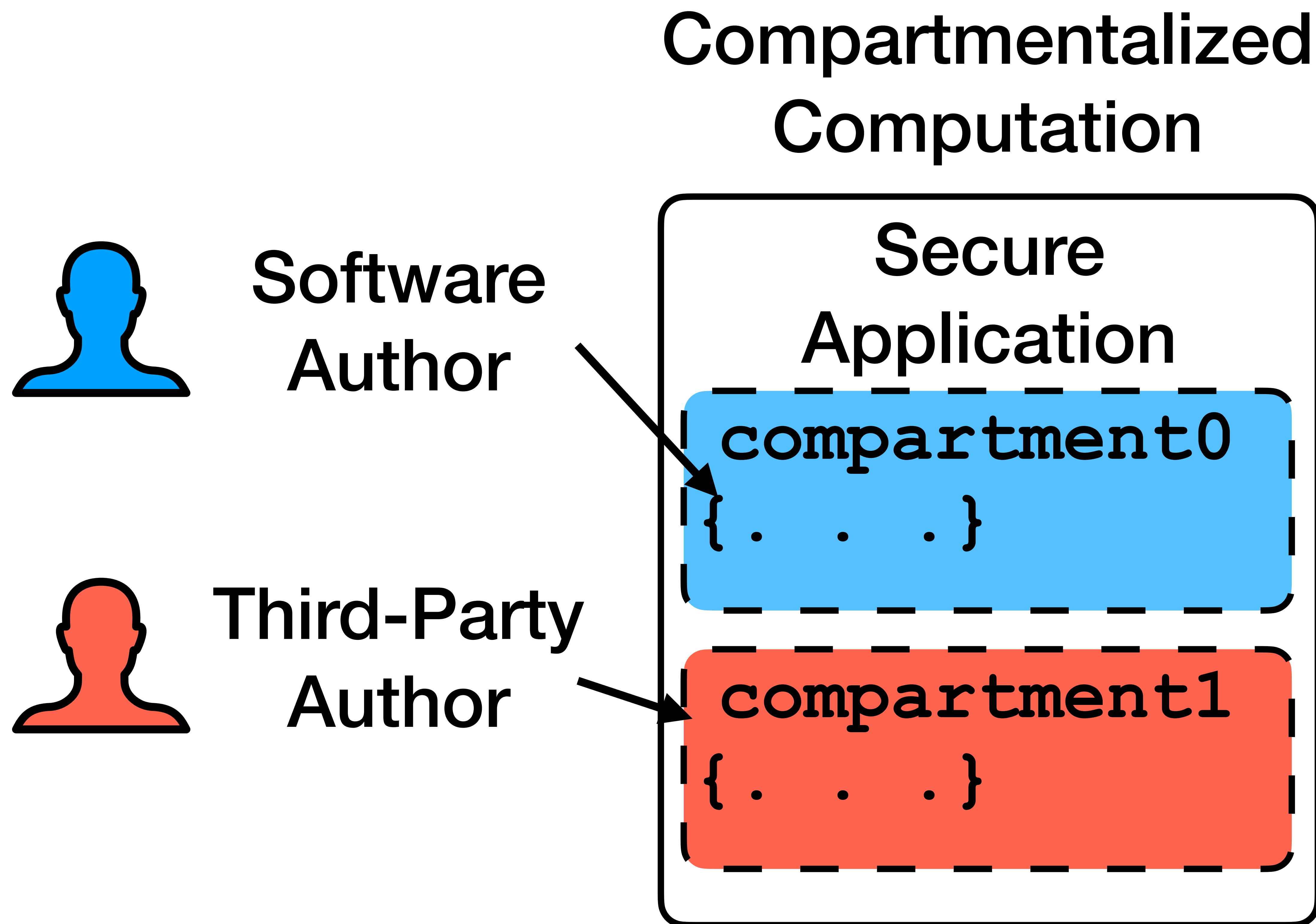


Monolithic Software

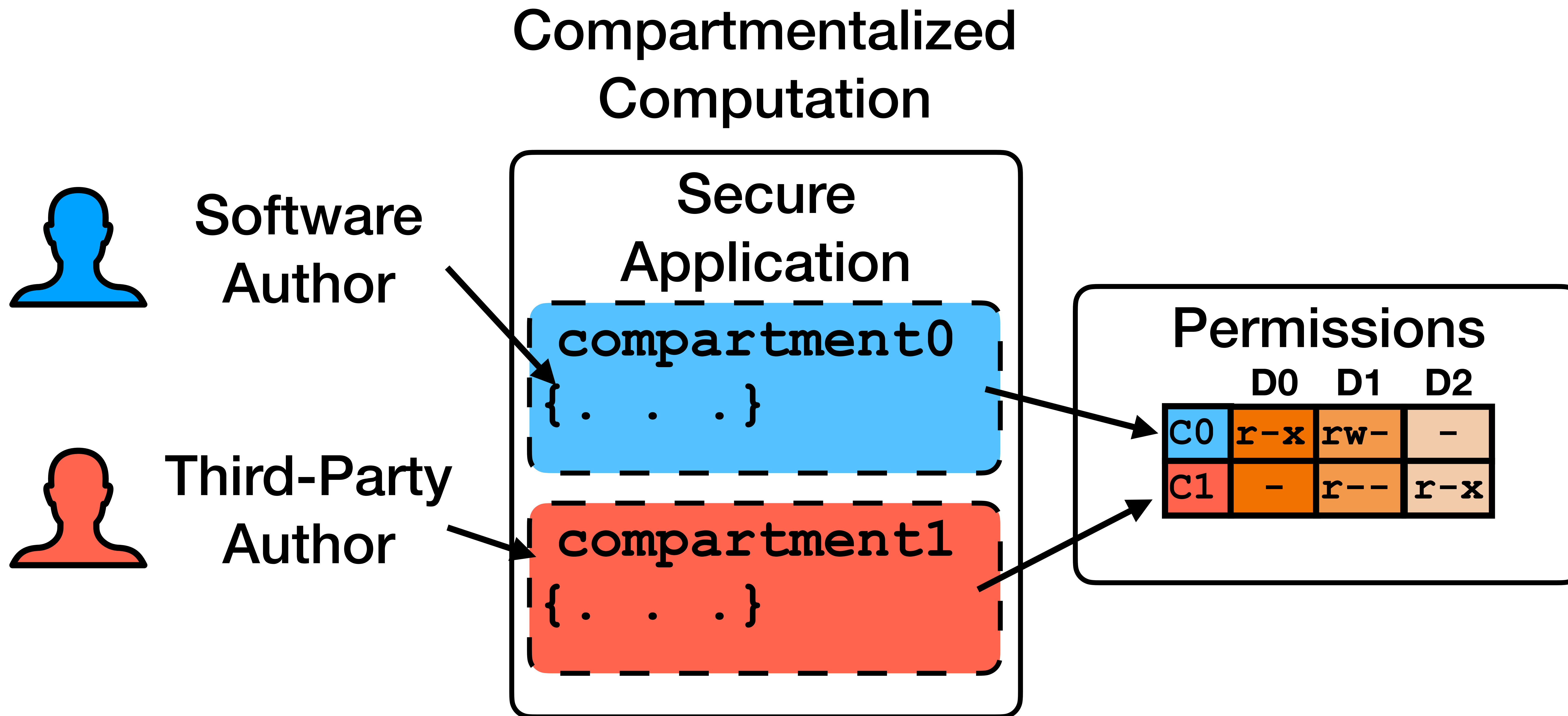


Binghamton University State University of New York
UC Riverside
usenix The Advanced Computing Systems Association

Compartmentalized Software



Compartmentalized Software



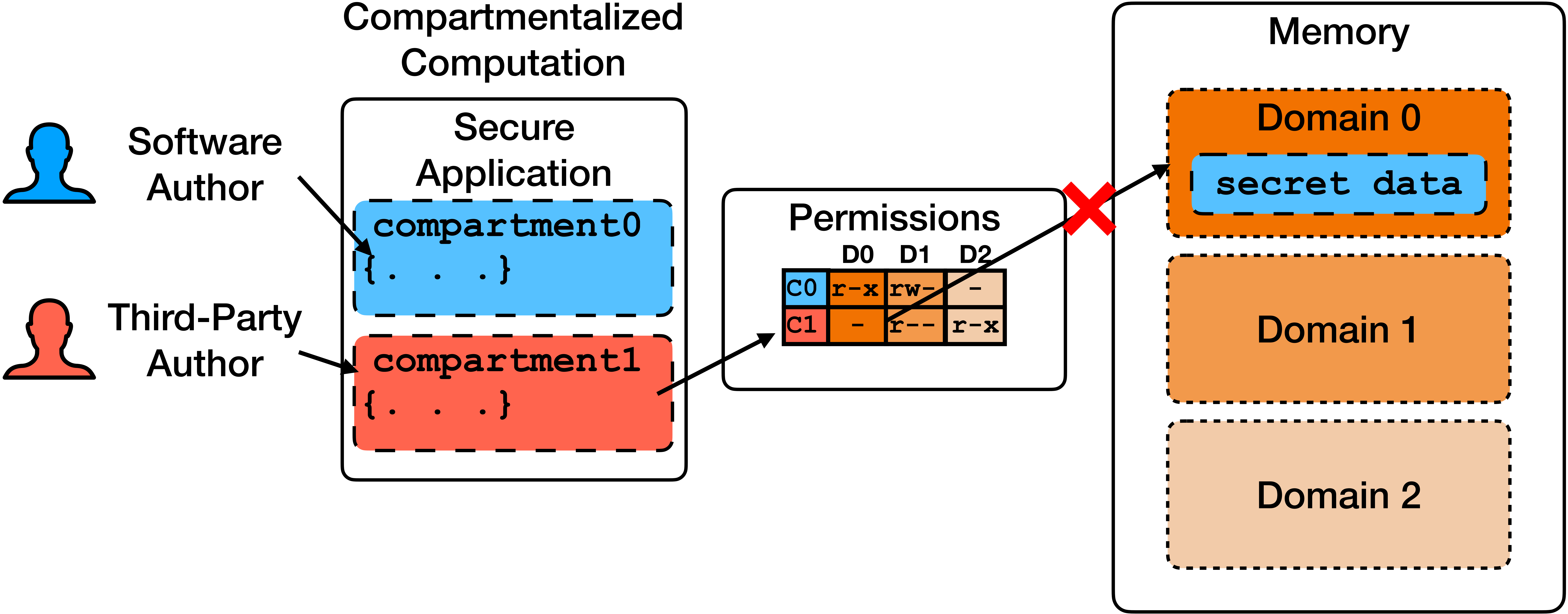
BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Compartmentalized Software

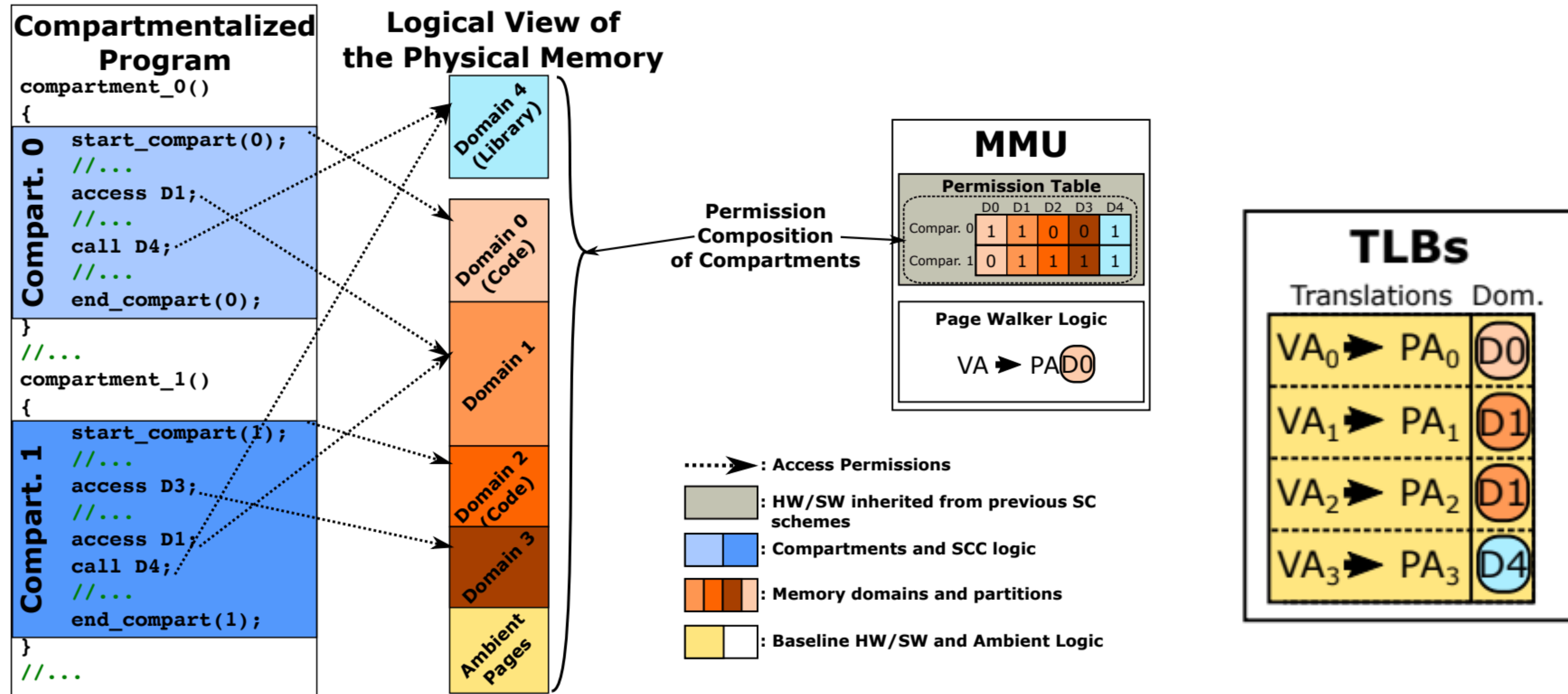
Compartmentalized Software

Compartmentalized Software

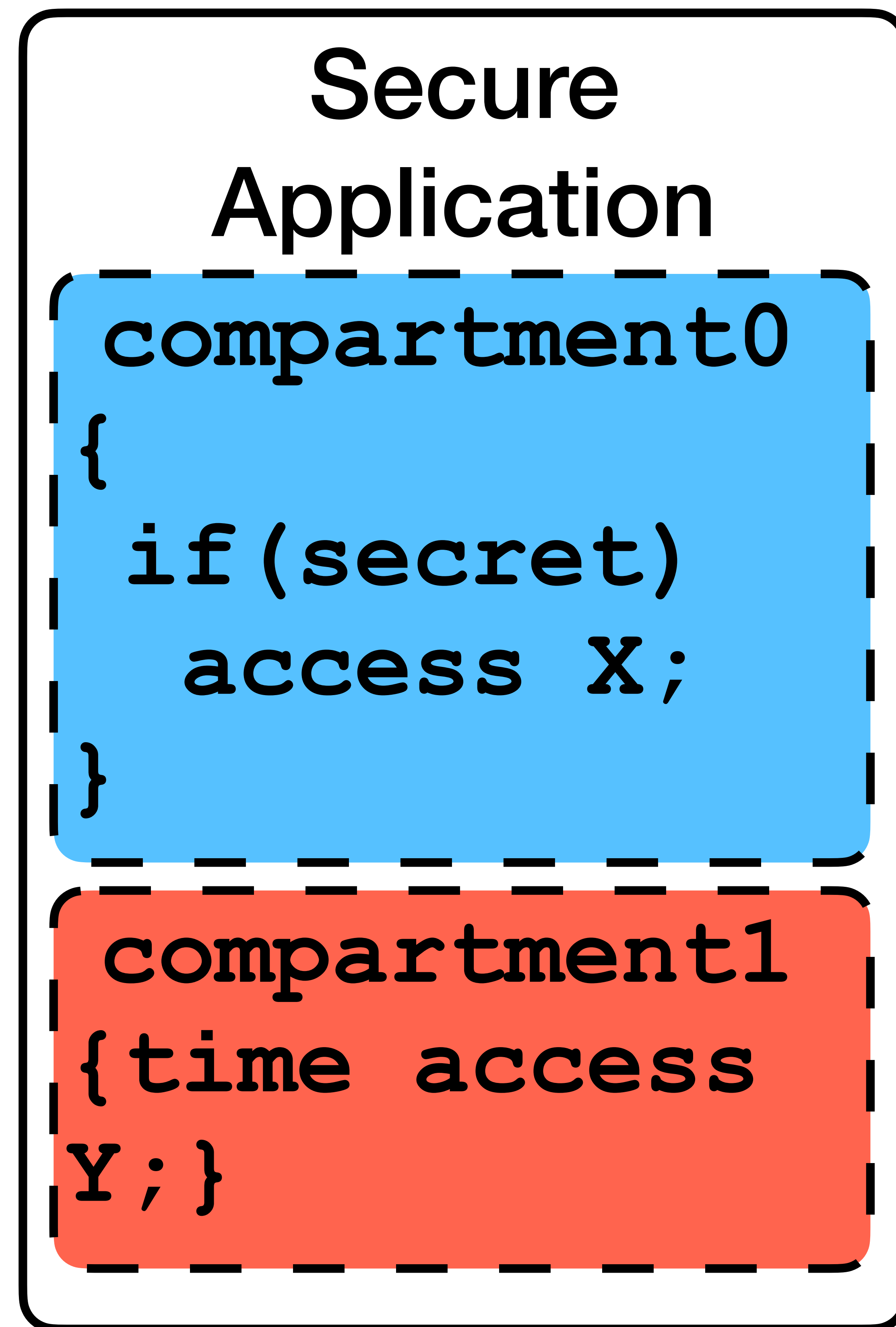
Compartmentalized Software

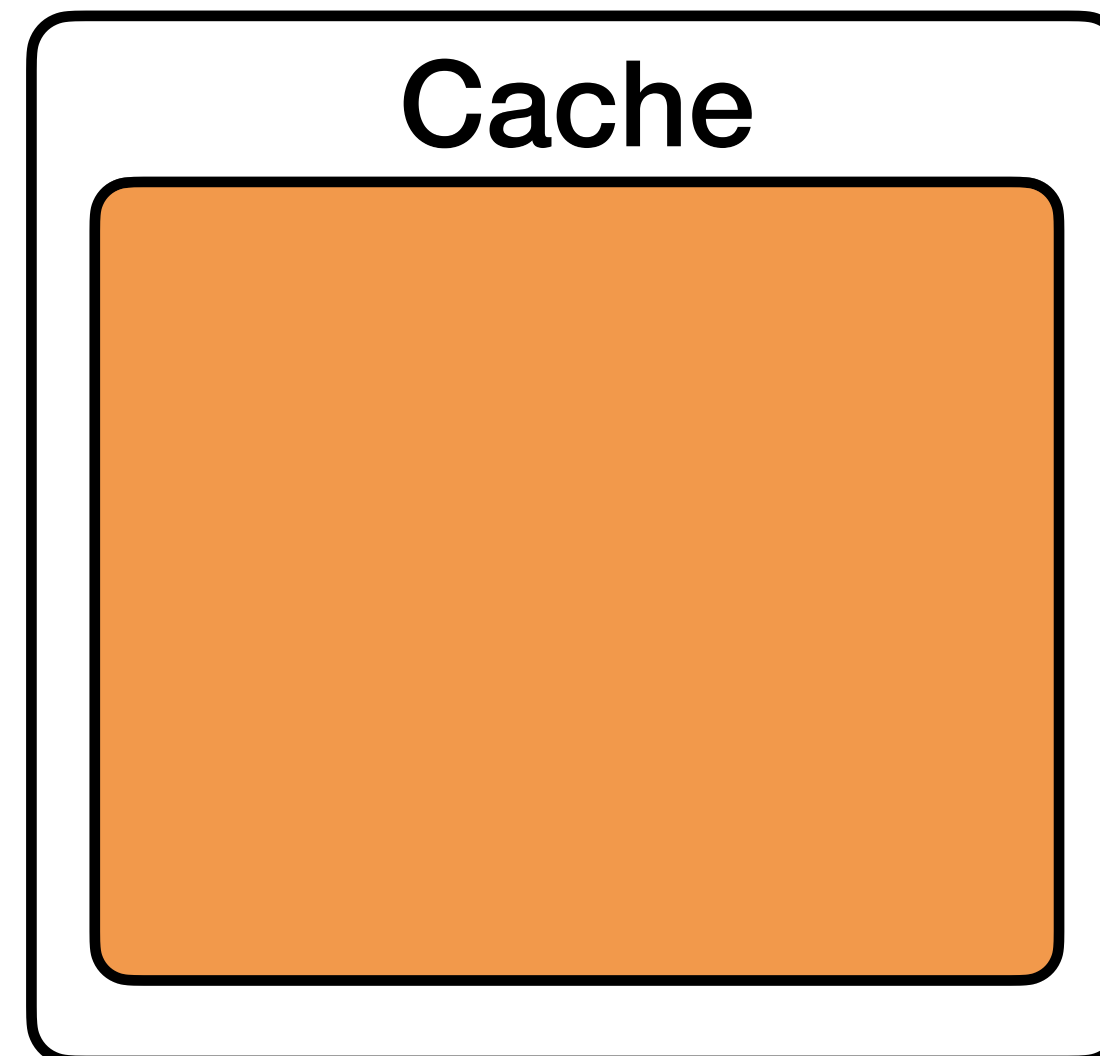


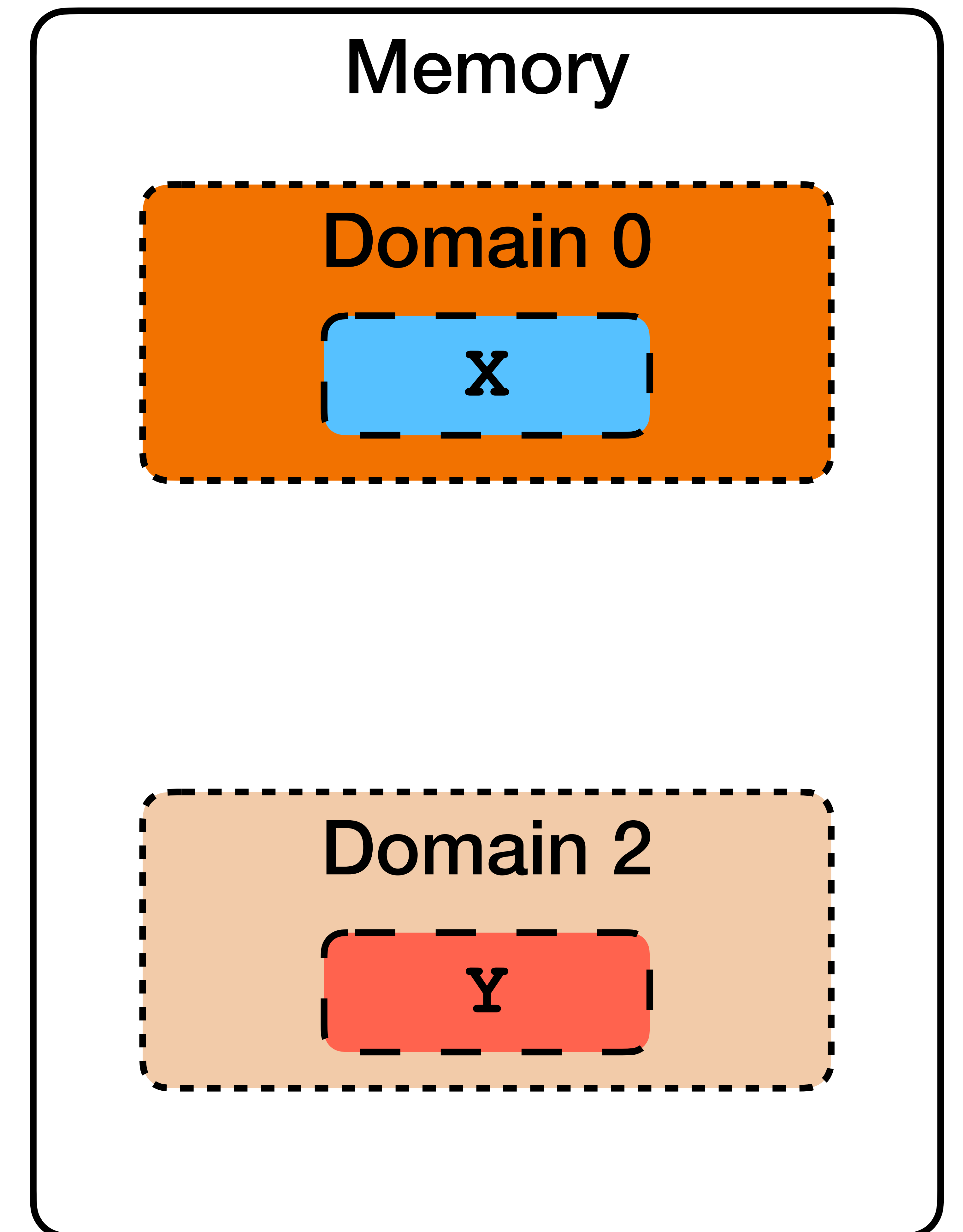
Hardware Side of Domains



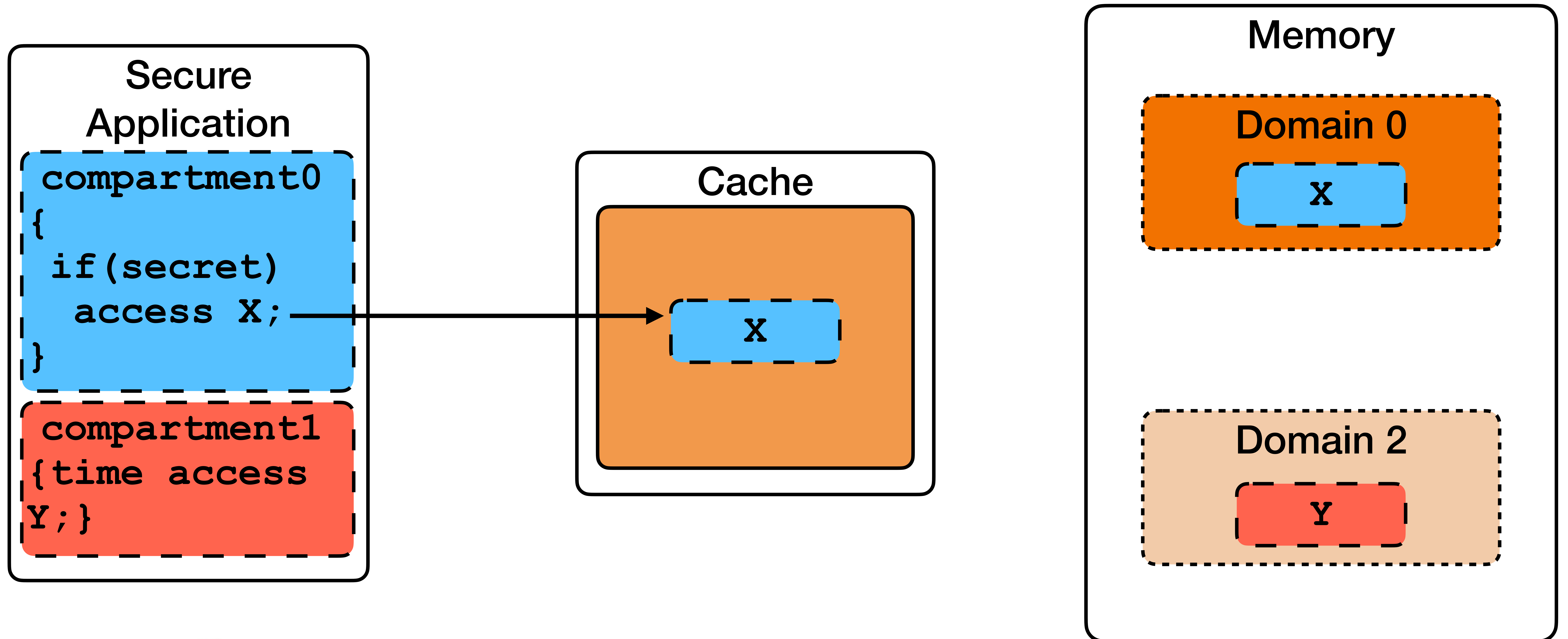
But… Caches Still Leak





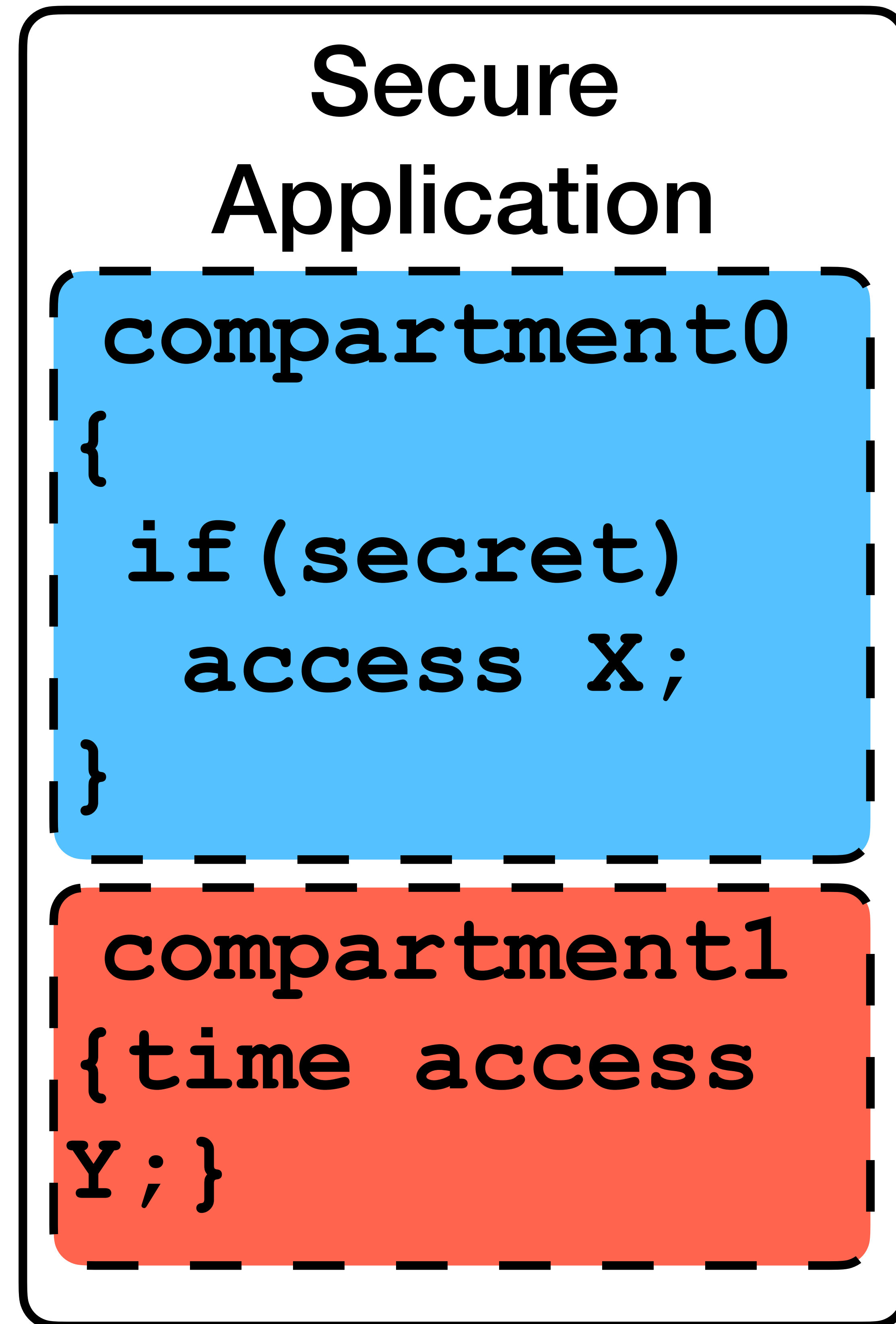


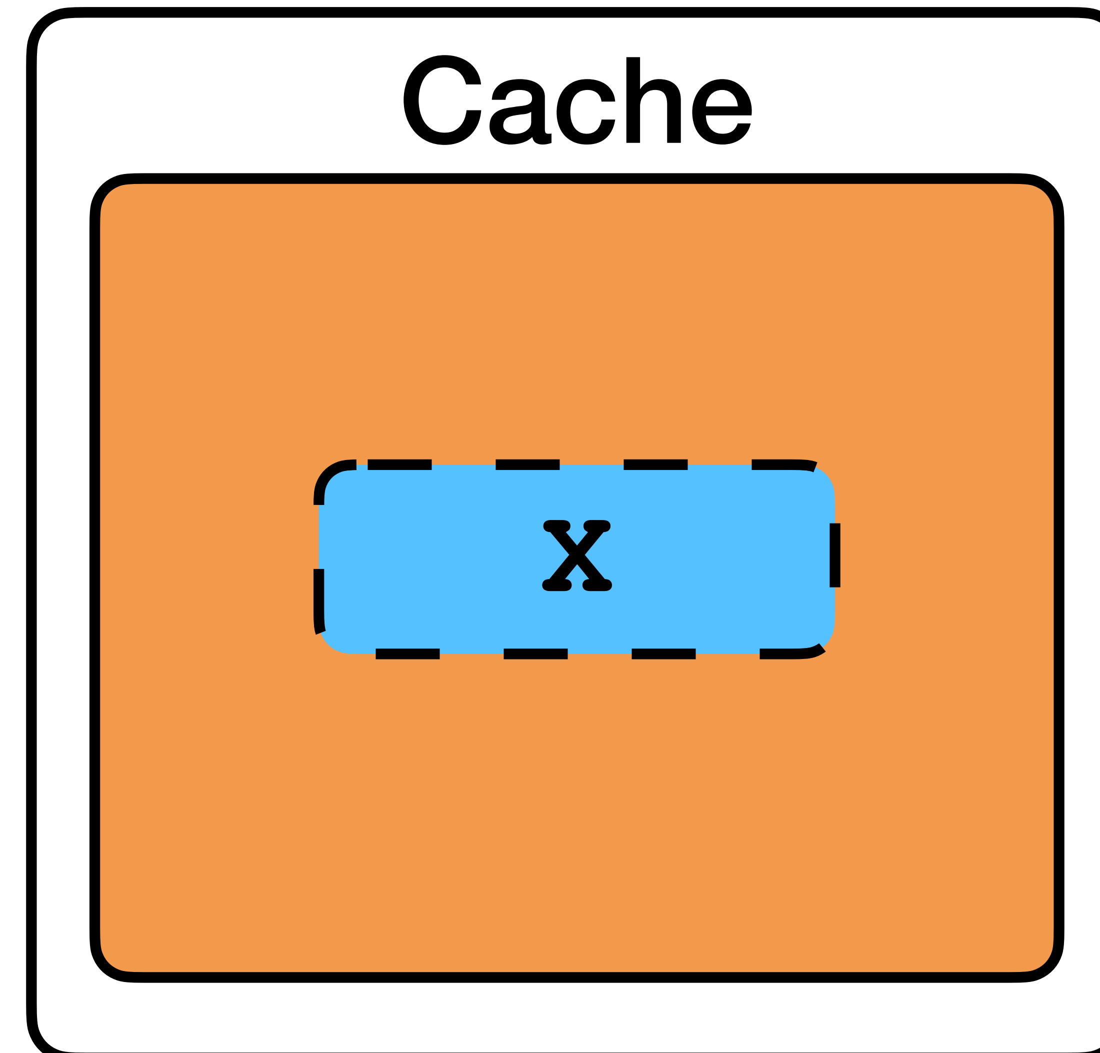
But... Caches Still Leak

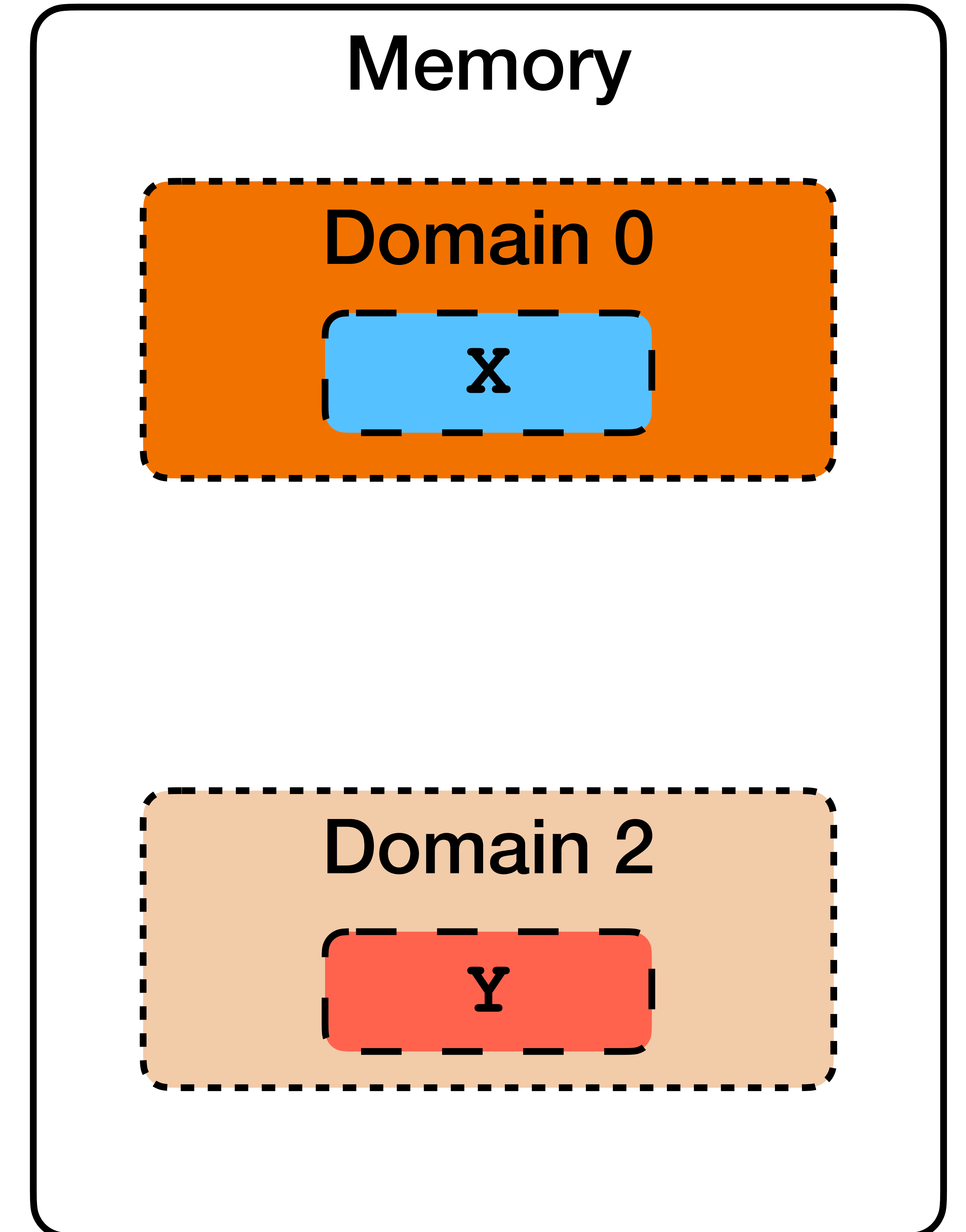


Binghamton University State University of New York
UC Riverside
usenix The Advanced Computing Systems Association

But… Caches Still Leak

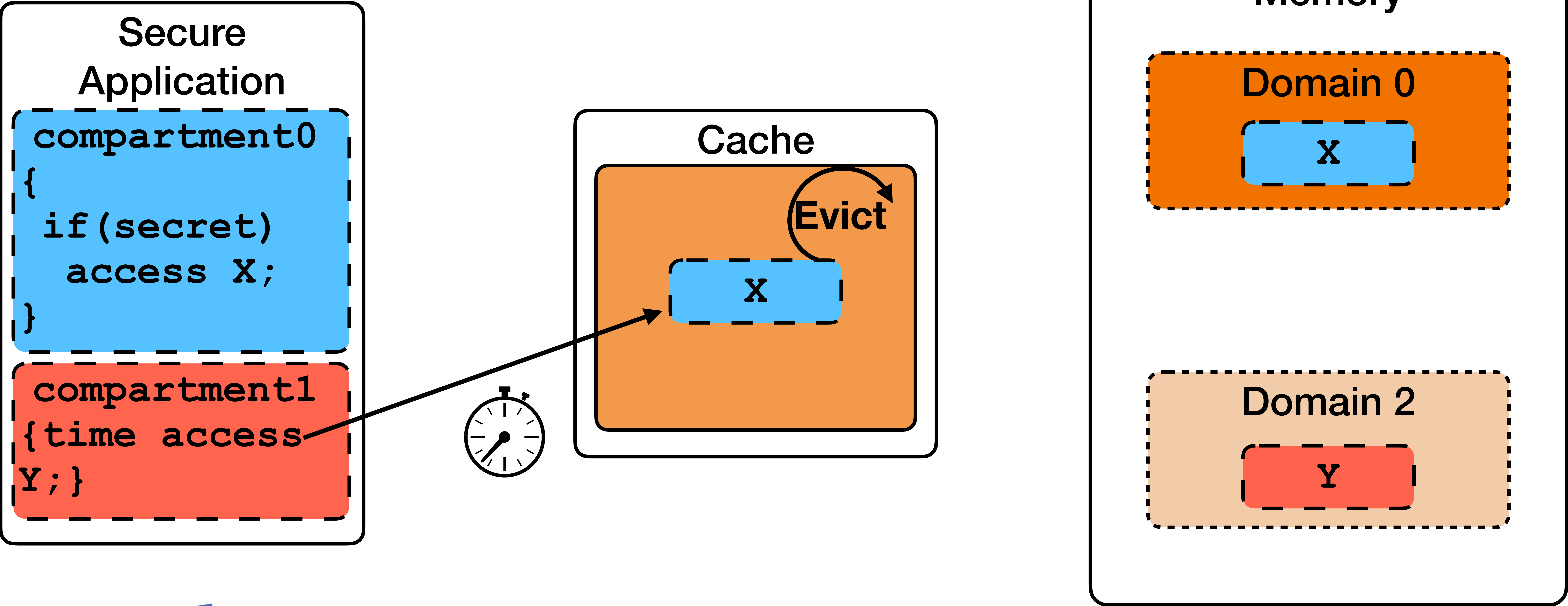






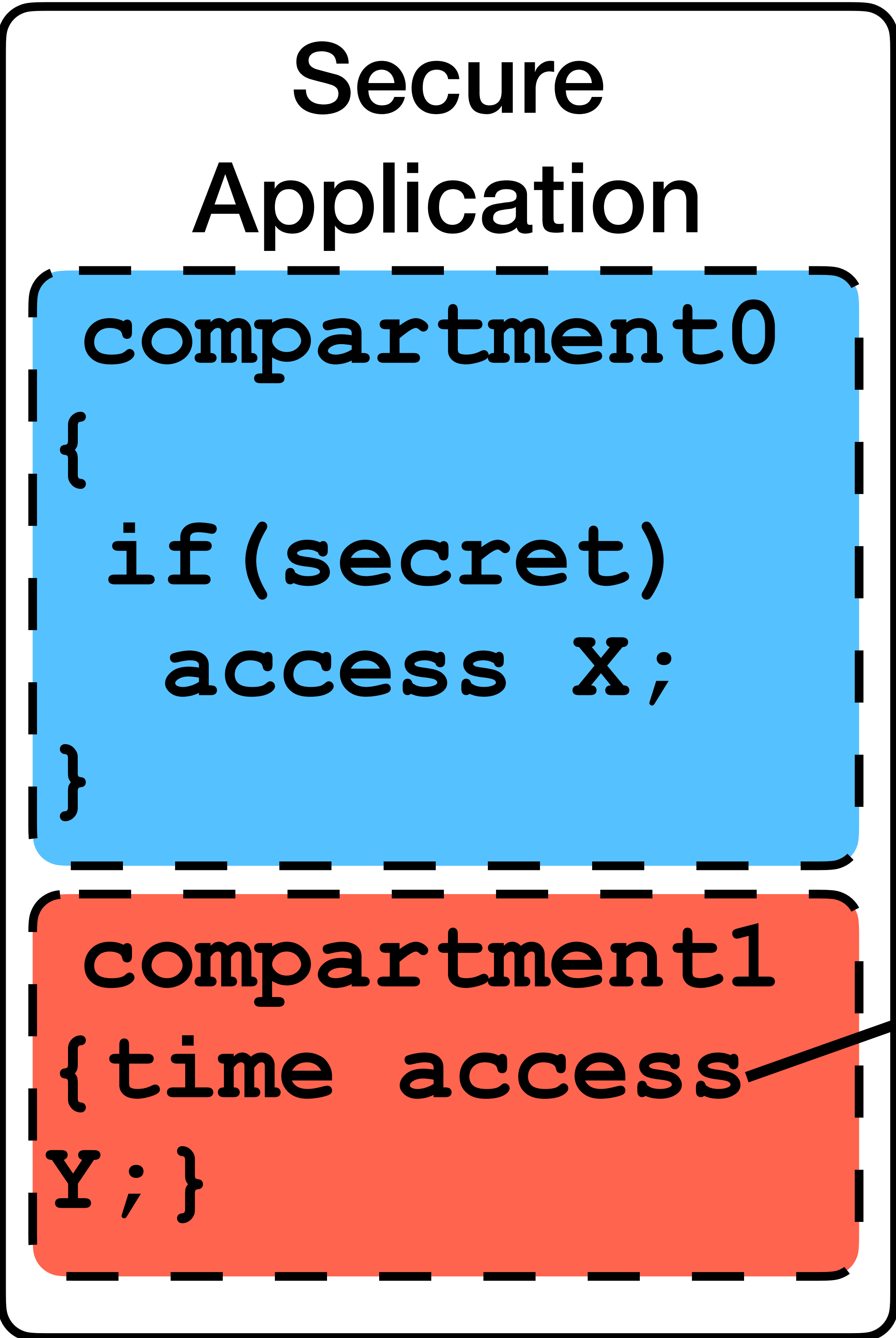
BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

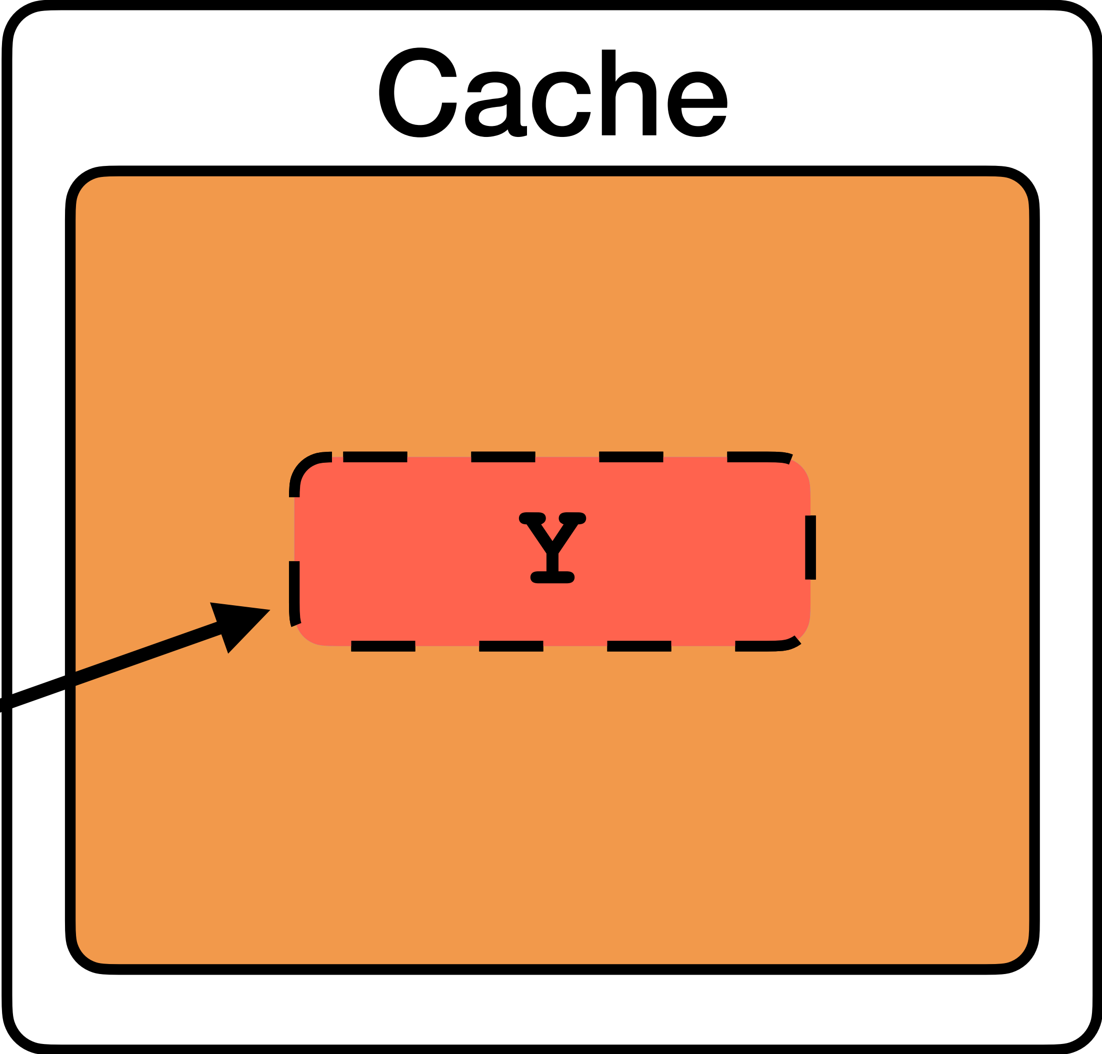
But… Caches Still Leak

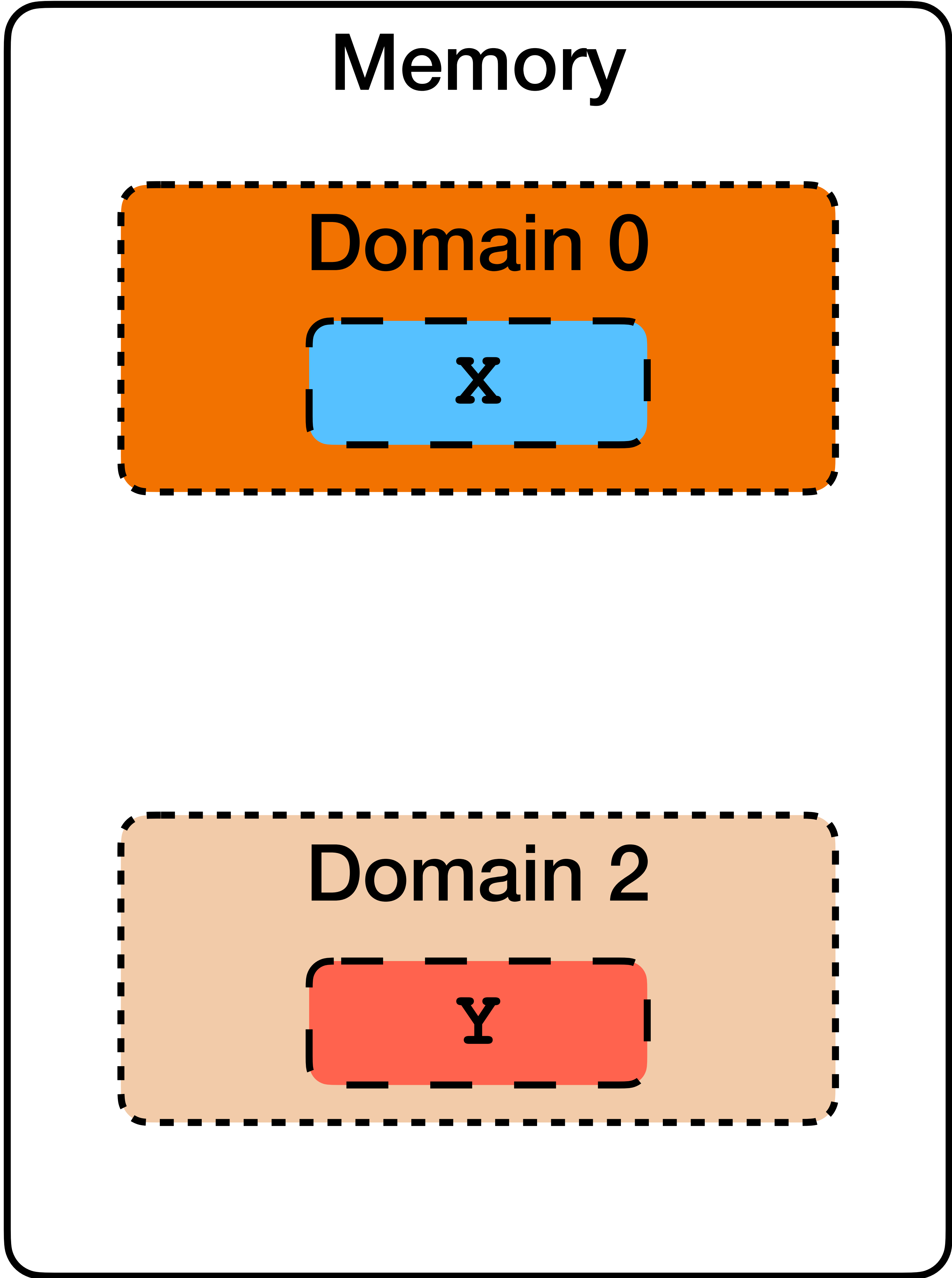


BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

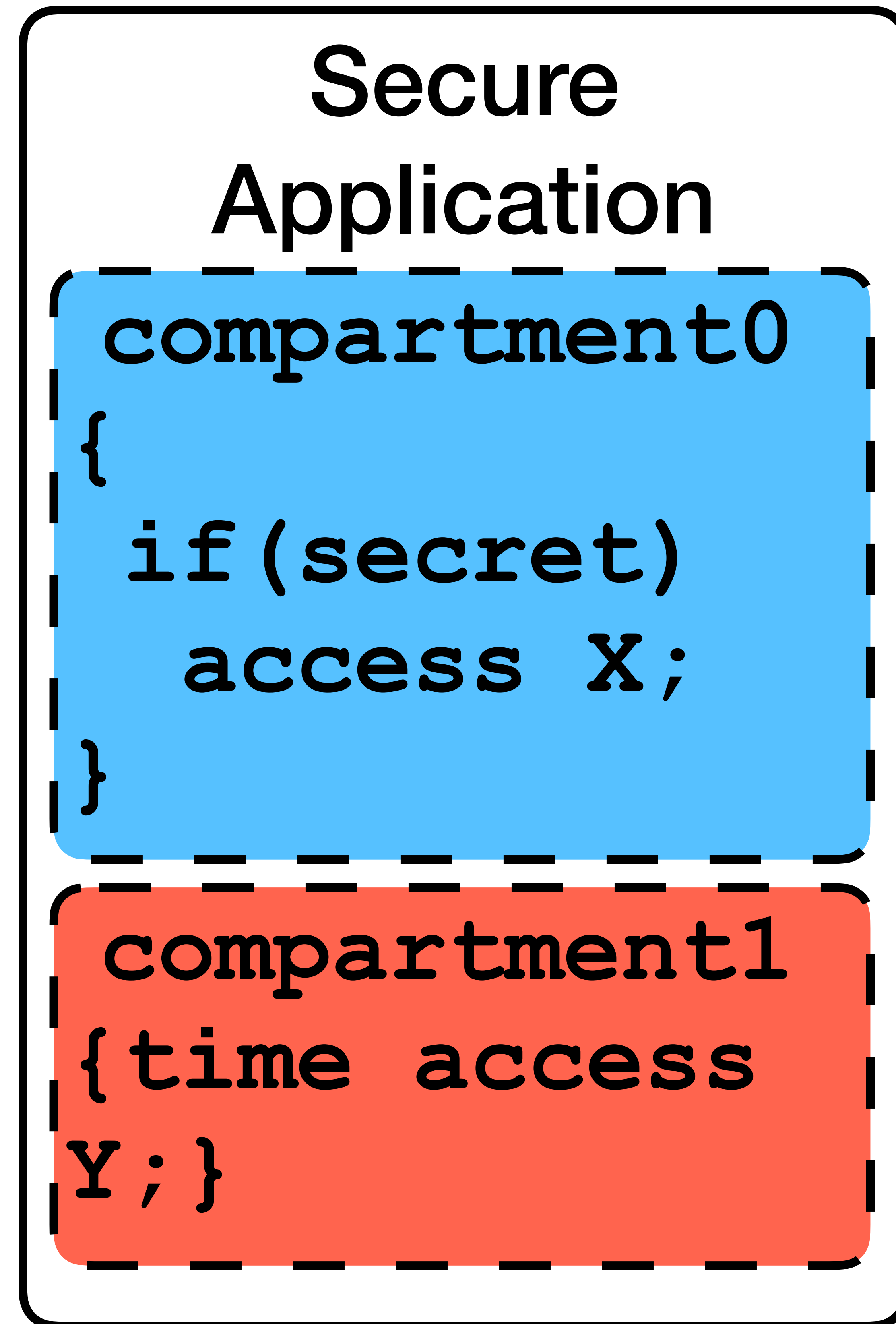
But... Caches Still Leak

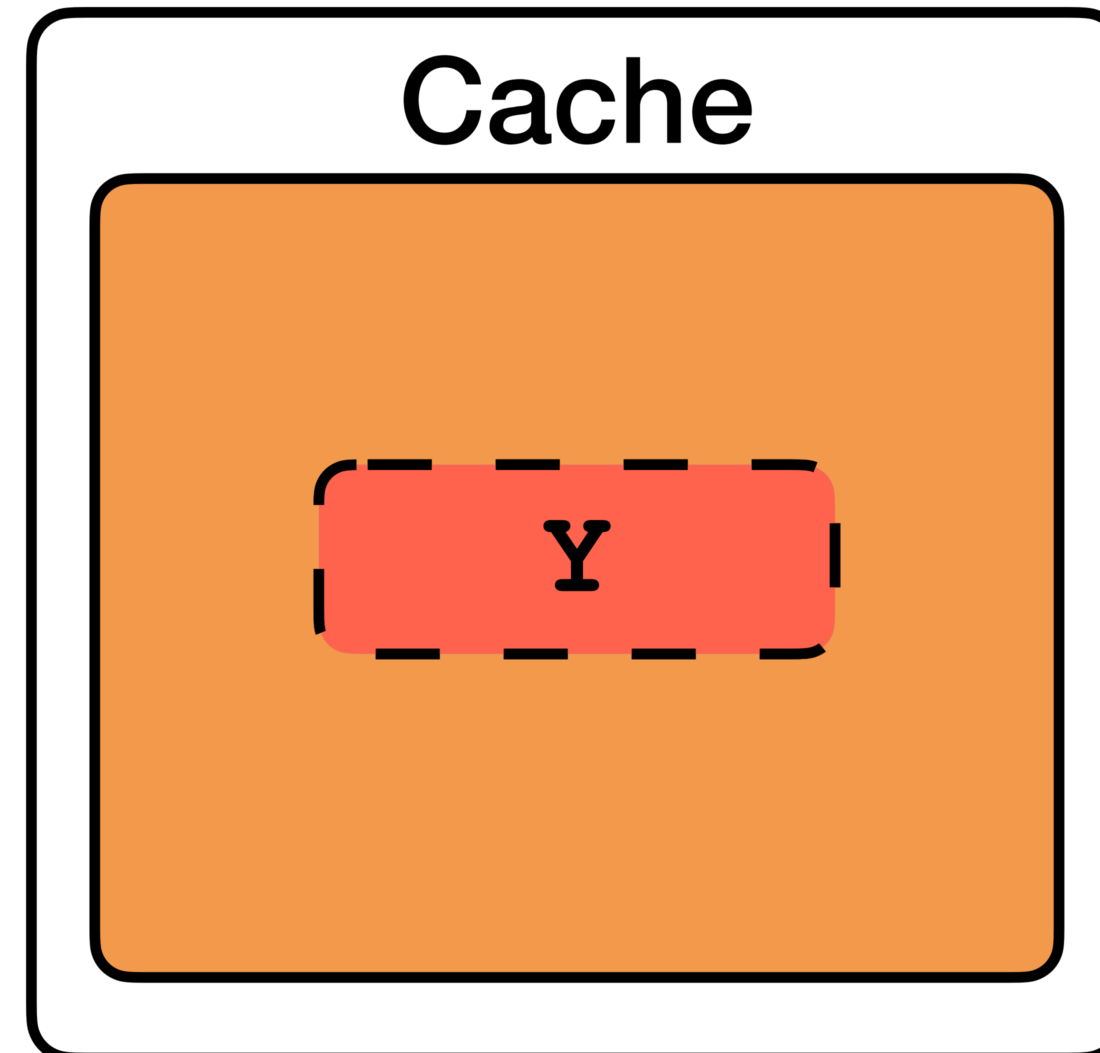


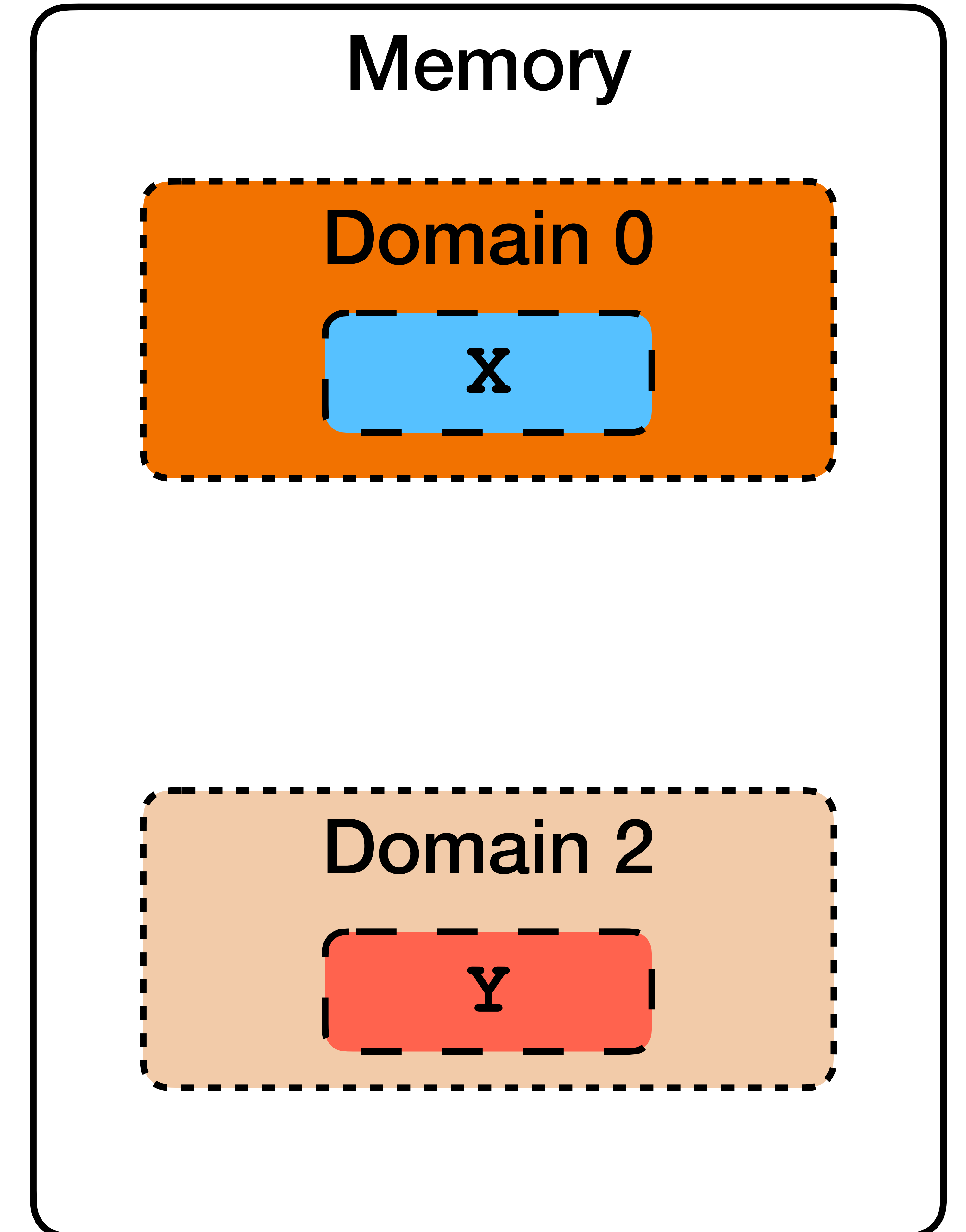




But... Caches Still Leak





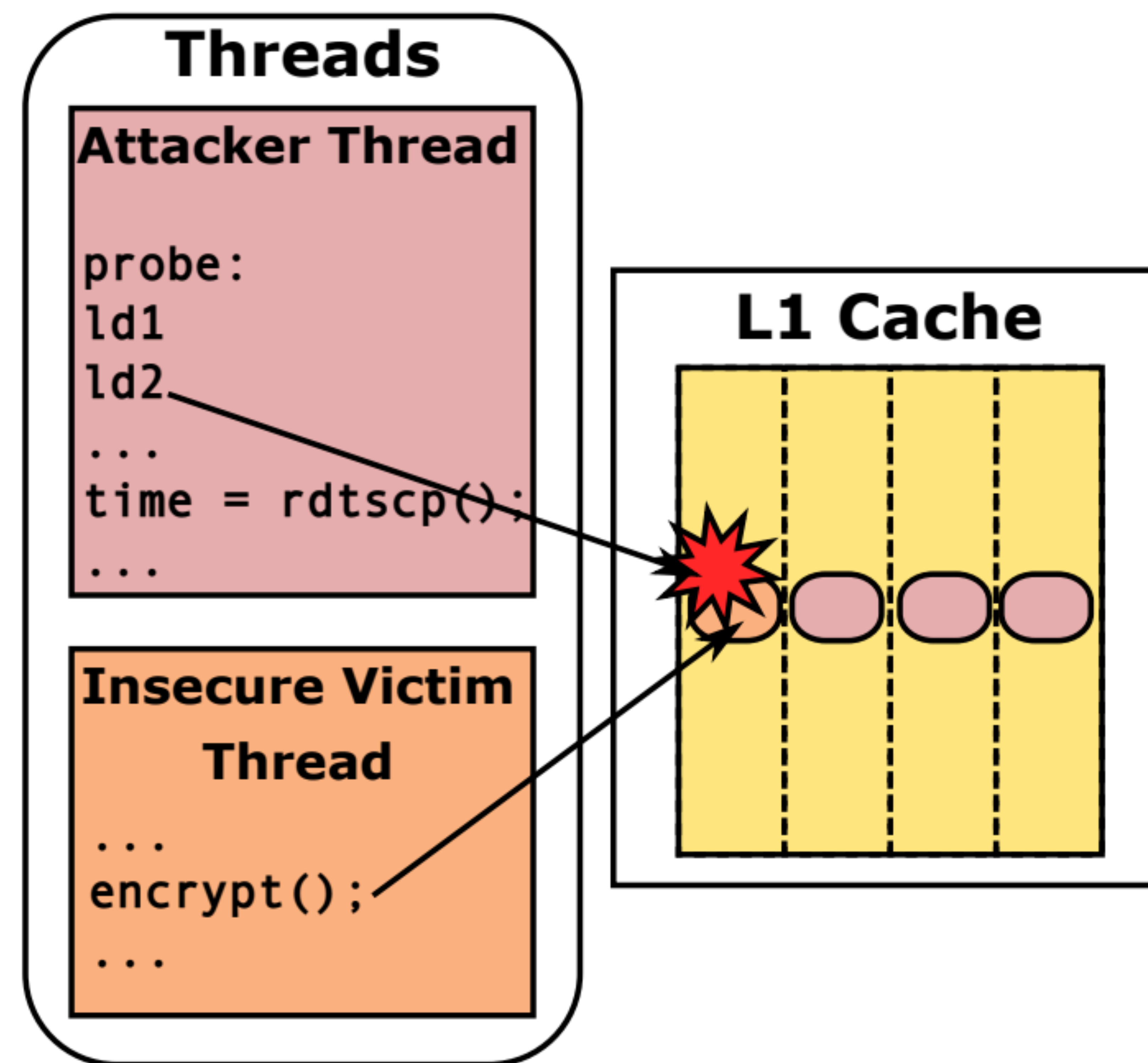


BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK UC RIVERSIDE usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Attacks Has Been Demonstrated

Attacks Has Been Demonstrated (1)

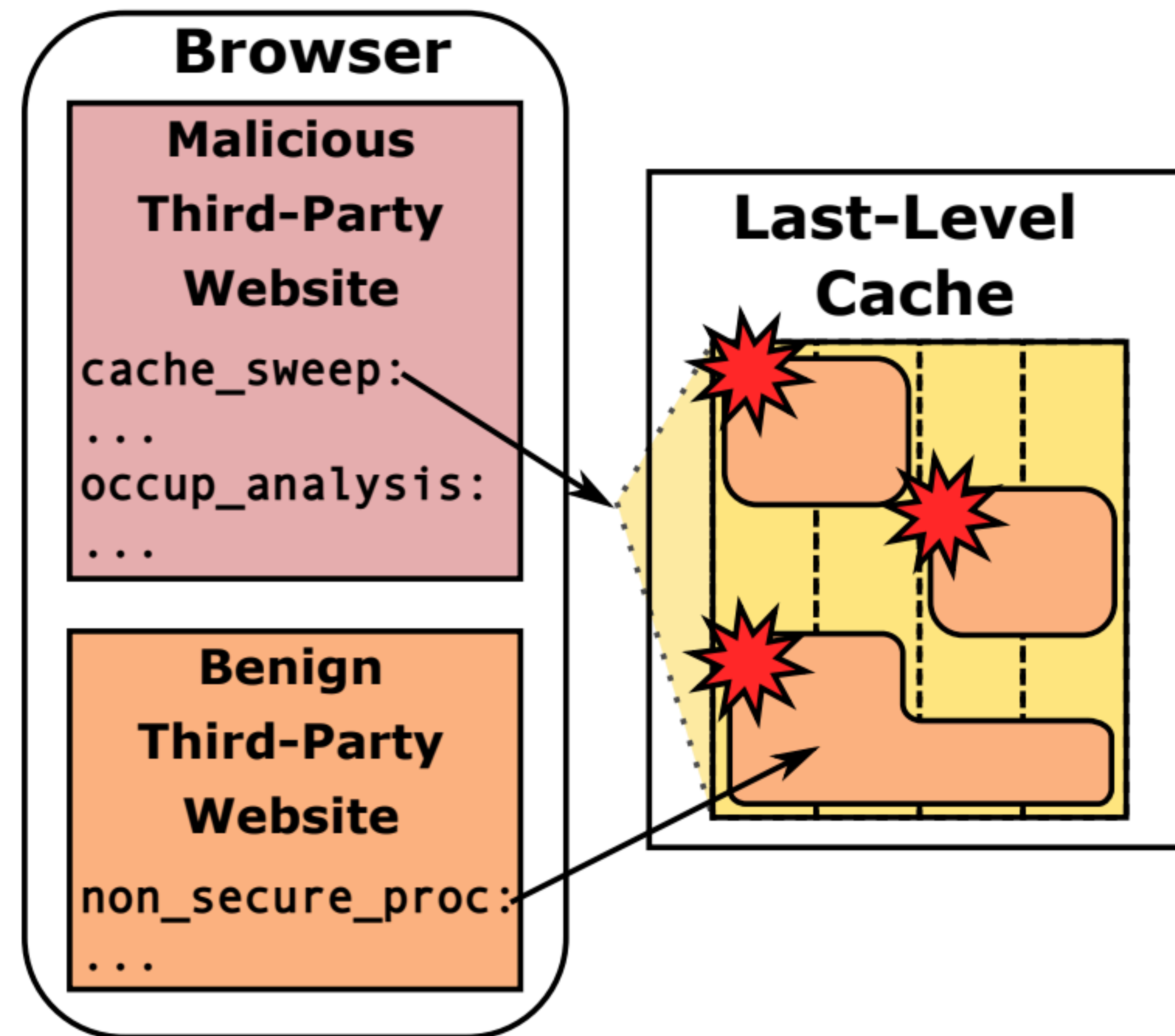
- The attacker and the victim can execute on different hyperthreads (Brasser et al. [1])



[1]: Software Grand Exposure: SGX Cache Attacks Are Practical. In Proceedings of the 11th USENIX Conference on Offensive Technologies, WOOT'17, page 11

Attacks Has Been Demonstrated (2)

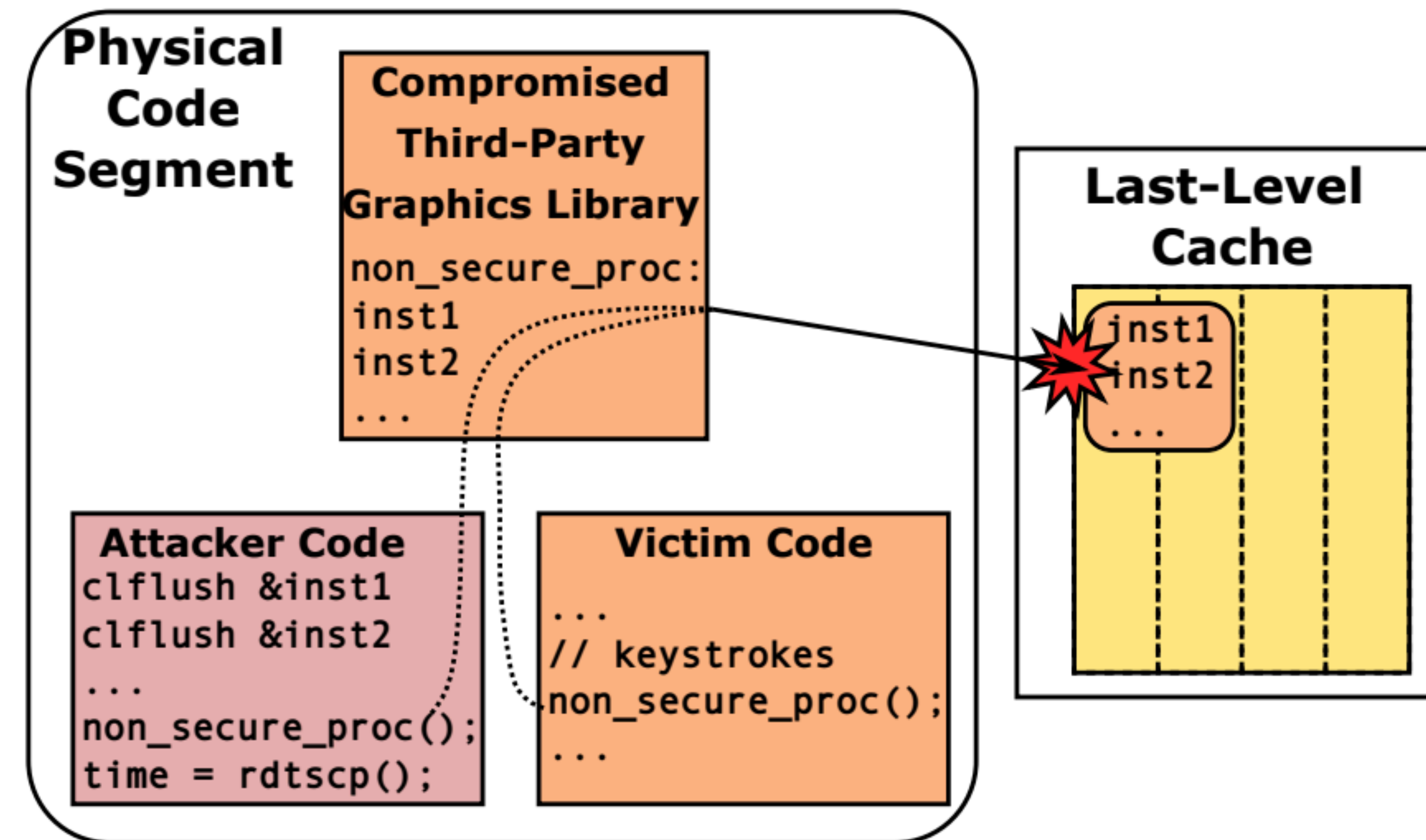
- Browser engines can run malicious website clients (Shusterman et al. [2])



[2]: Robust website fingerprinting through the cache occupancy channel. In USENIX Security Symposium, 2019.

Attacks Has Been Demonstrated (3)

- A shared library can be compromised to collide in the cache (Wang et al. [3])

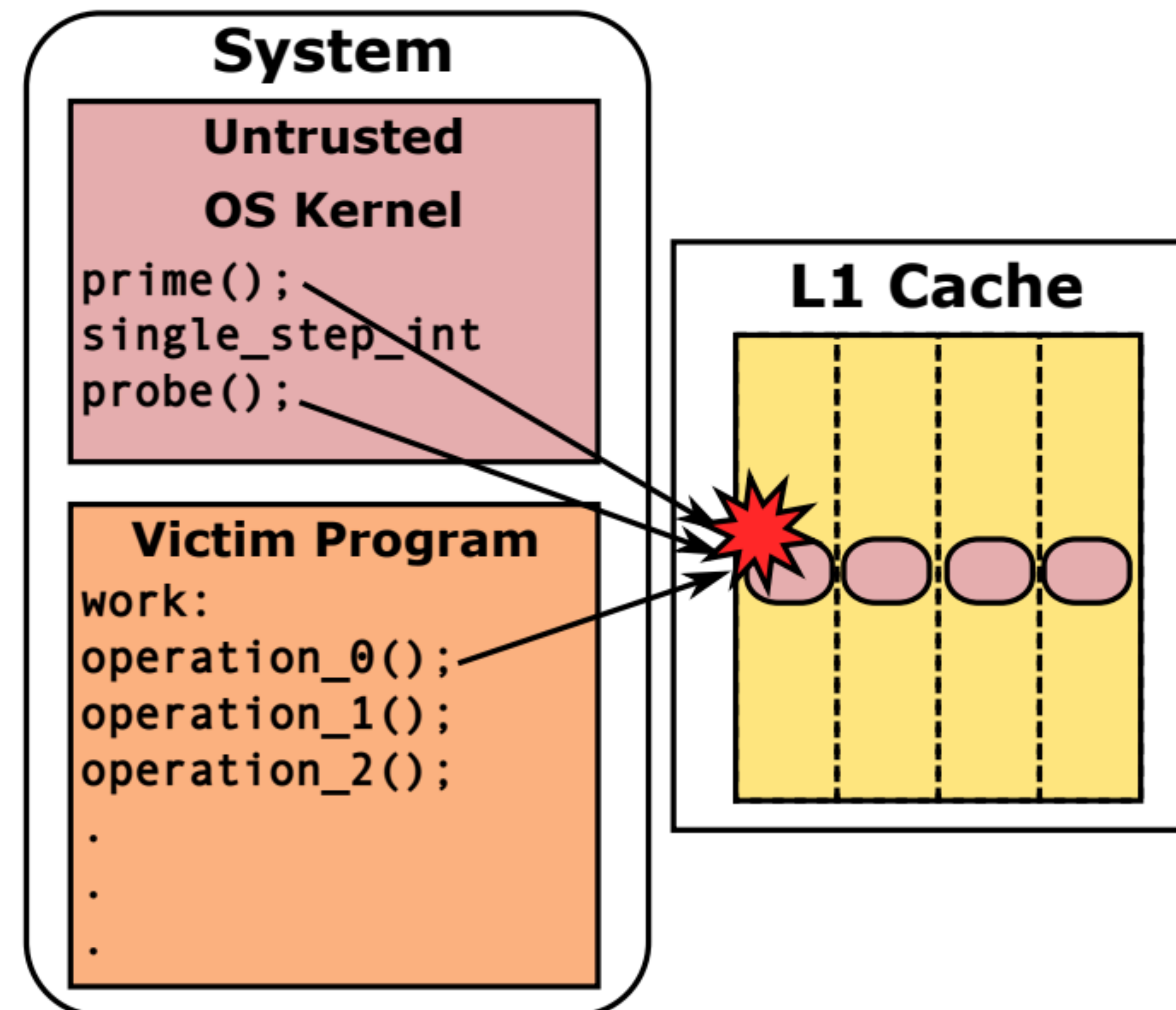


BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK | UC RIVERSIDE | usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

[3]: Software Grand Exposure: SGX Cache Attacks Are Practical. In Proceedings of the 11th USENIX Conference on Offensive Technologies, WOOT'17, page 11

Attacks Has Been Demonstrated (4)

- OS kernel can carry out cache attacks within a process (Hähnel et al. [4])

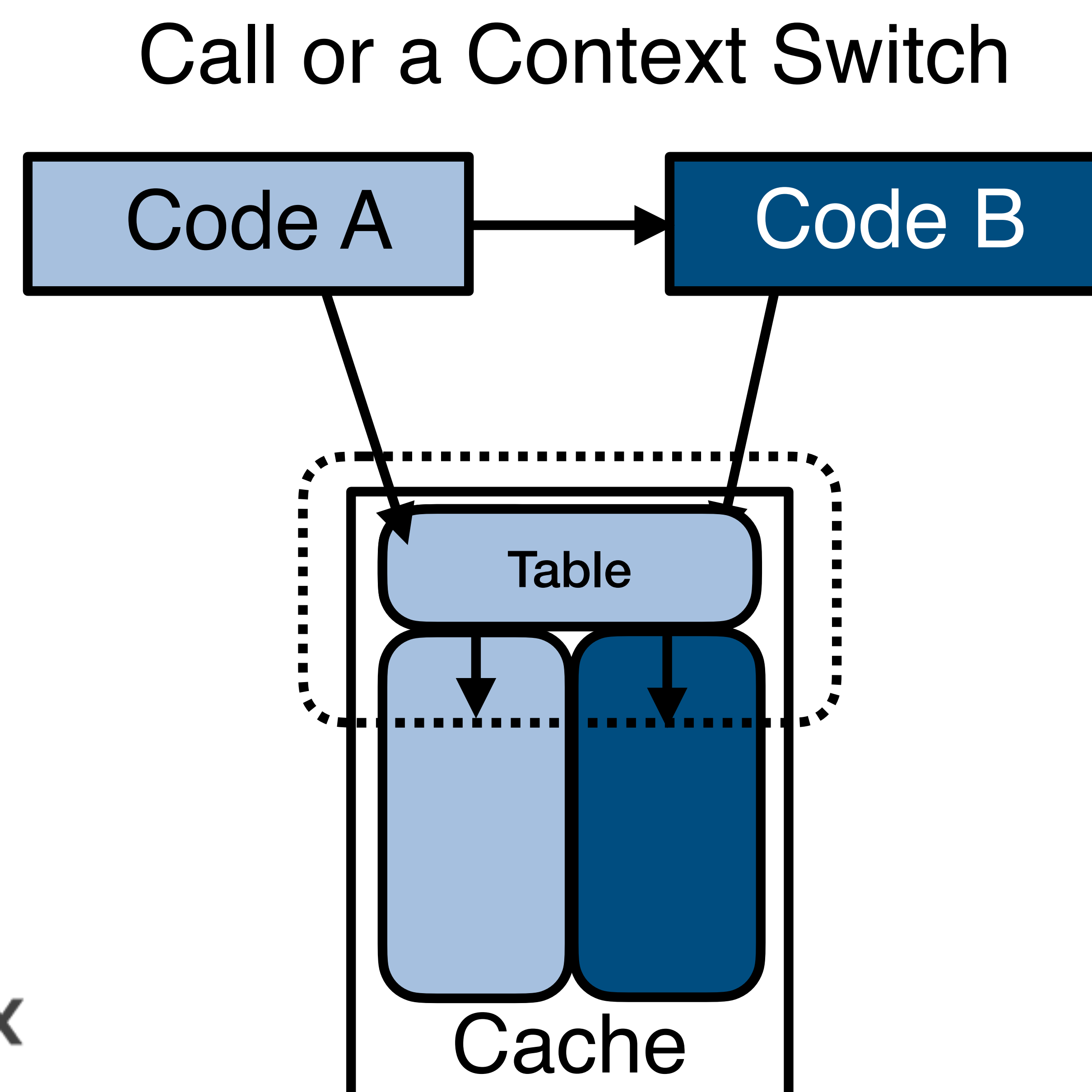


BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

[4]: High-Resolution Side Channels for Untrusted Operating Systems. In 2017 USENIX Annual Technical Conference (USENIX ATC 17), pages 299–312, Santa Clara, CA, July 2017. USENIX Association

Response: Cache Partitioning

- A principled and deterministic approach to stop side-channels.
- However, there hasn't been any effort in implementing partitioning tailored compartmentalization.

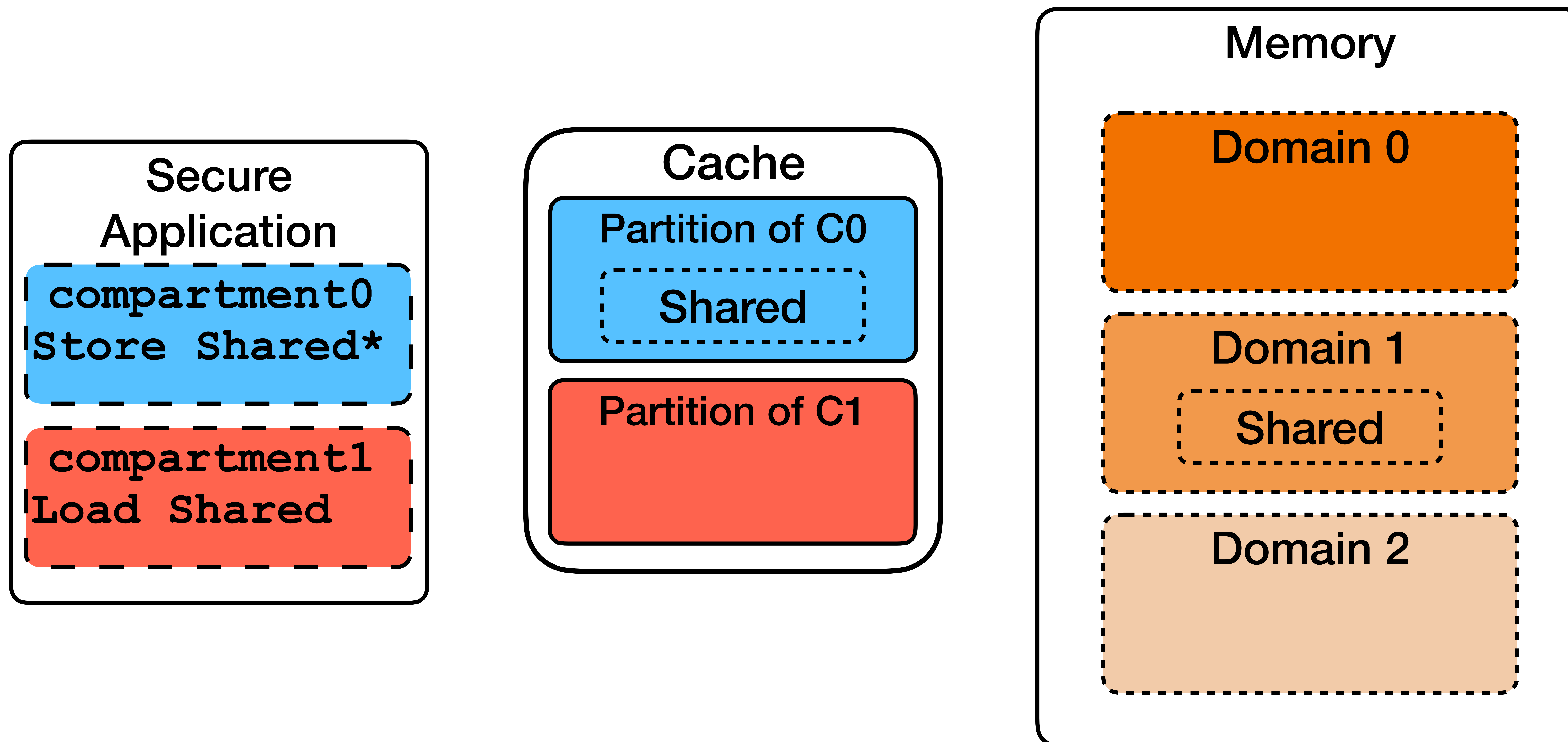


We Propose:
Secure **C**aches for **C**ompartments (SCC)

BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

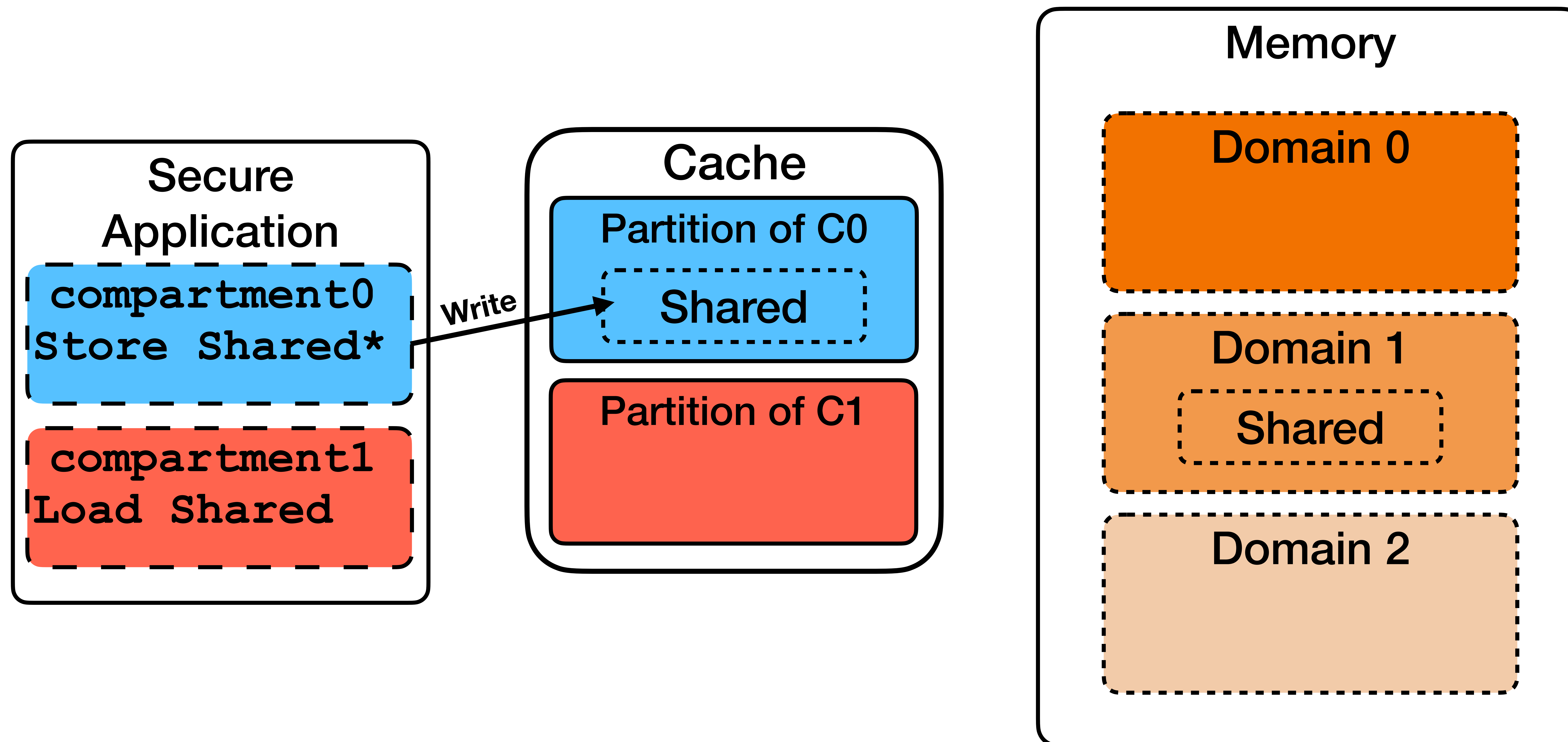
Angle-1: Domain-Oriented Partitioning

The Problem With Code-Oriented Partitioning



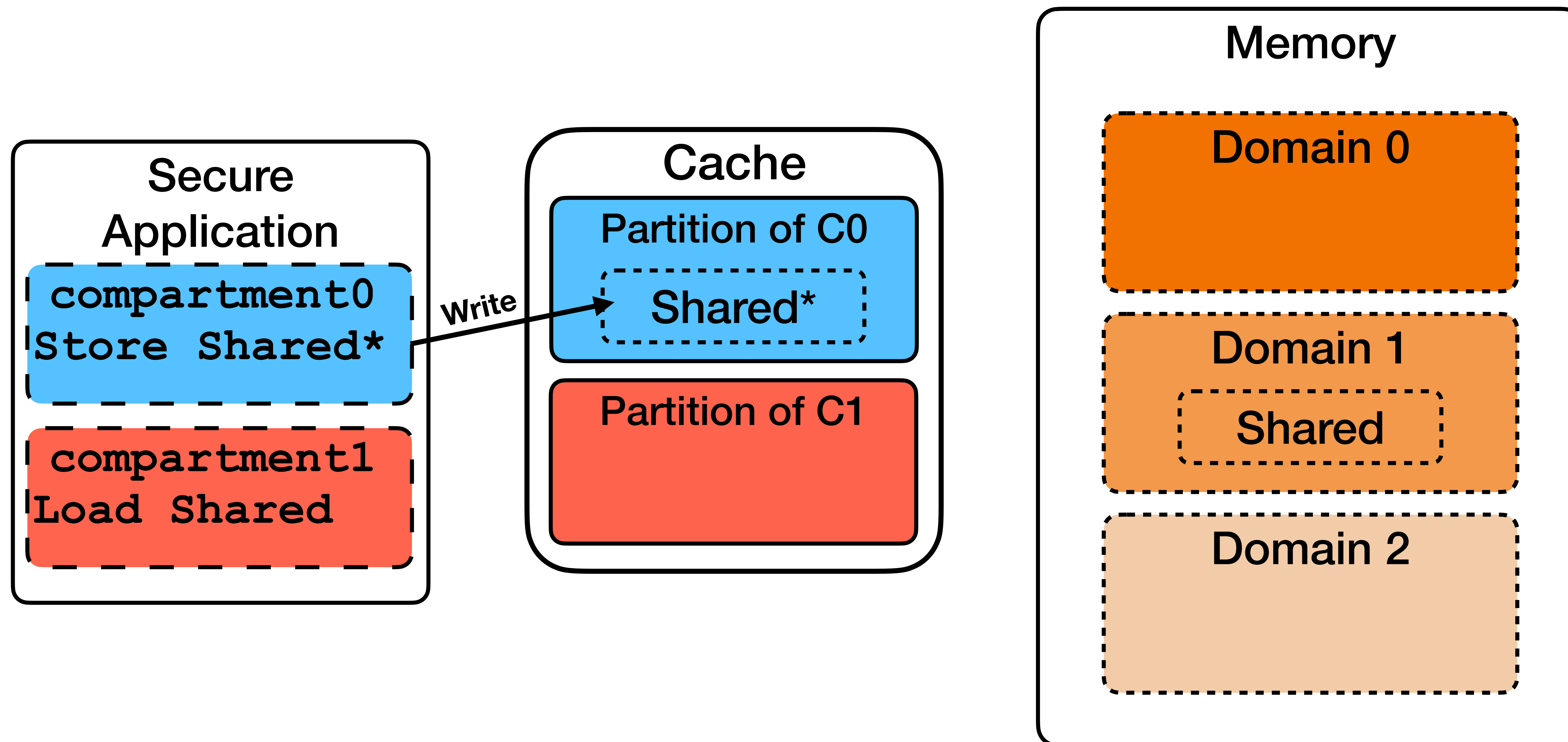
Binghamton University State University of New York
UC Riverside
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

The Problem With Code-Oriented Partitioning

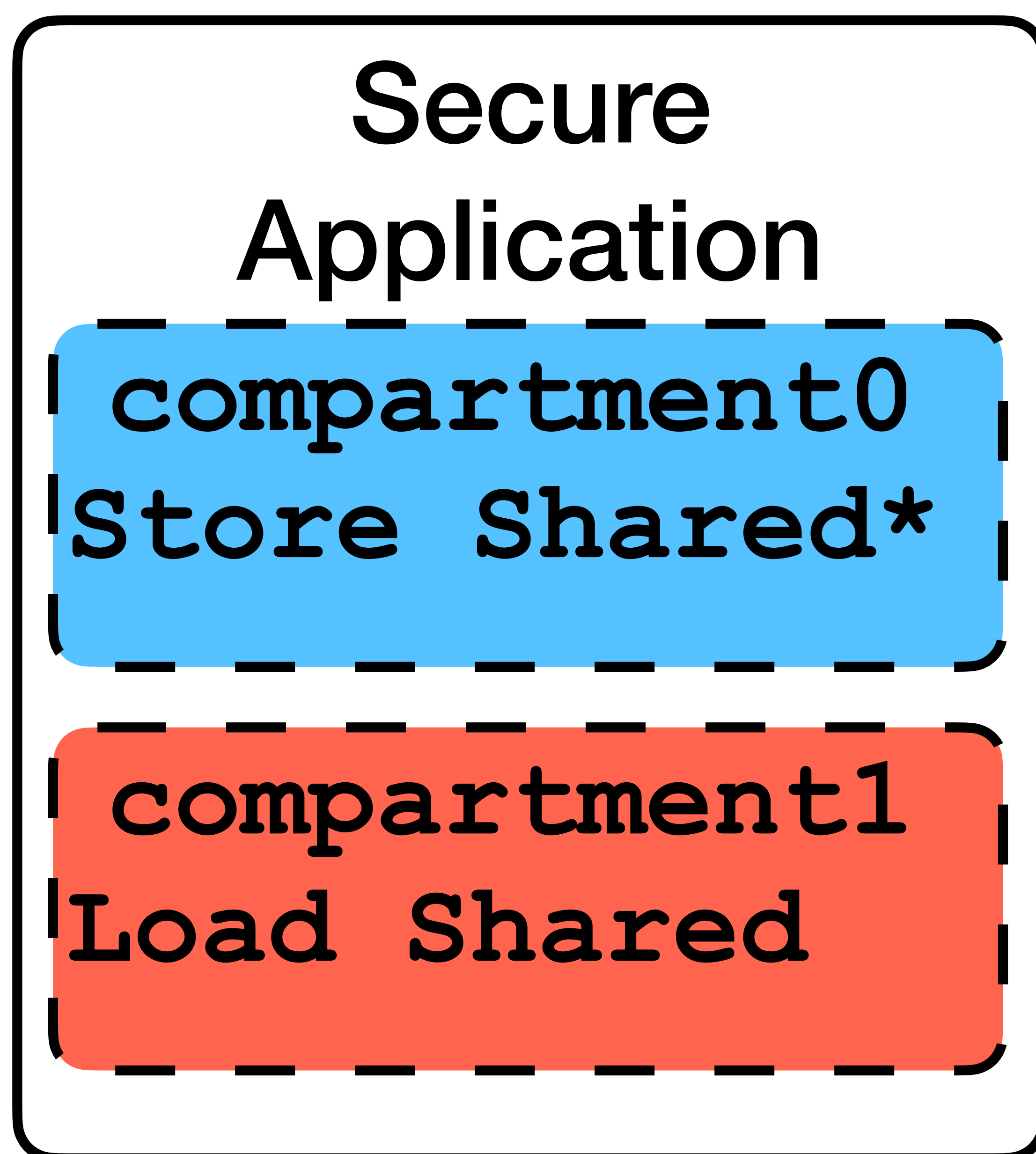


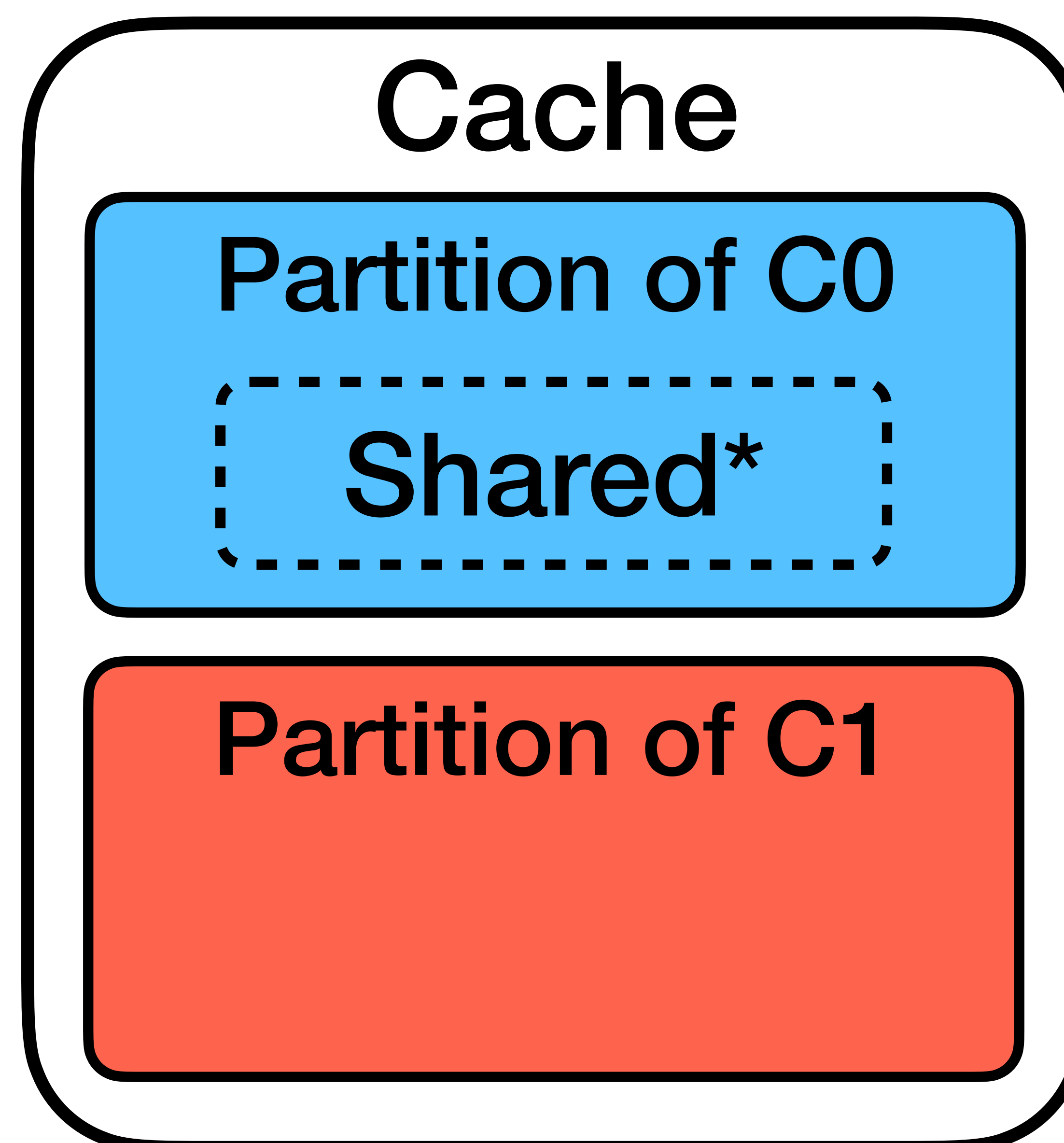
BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

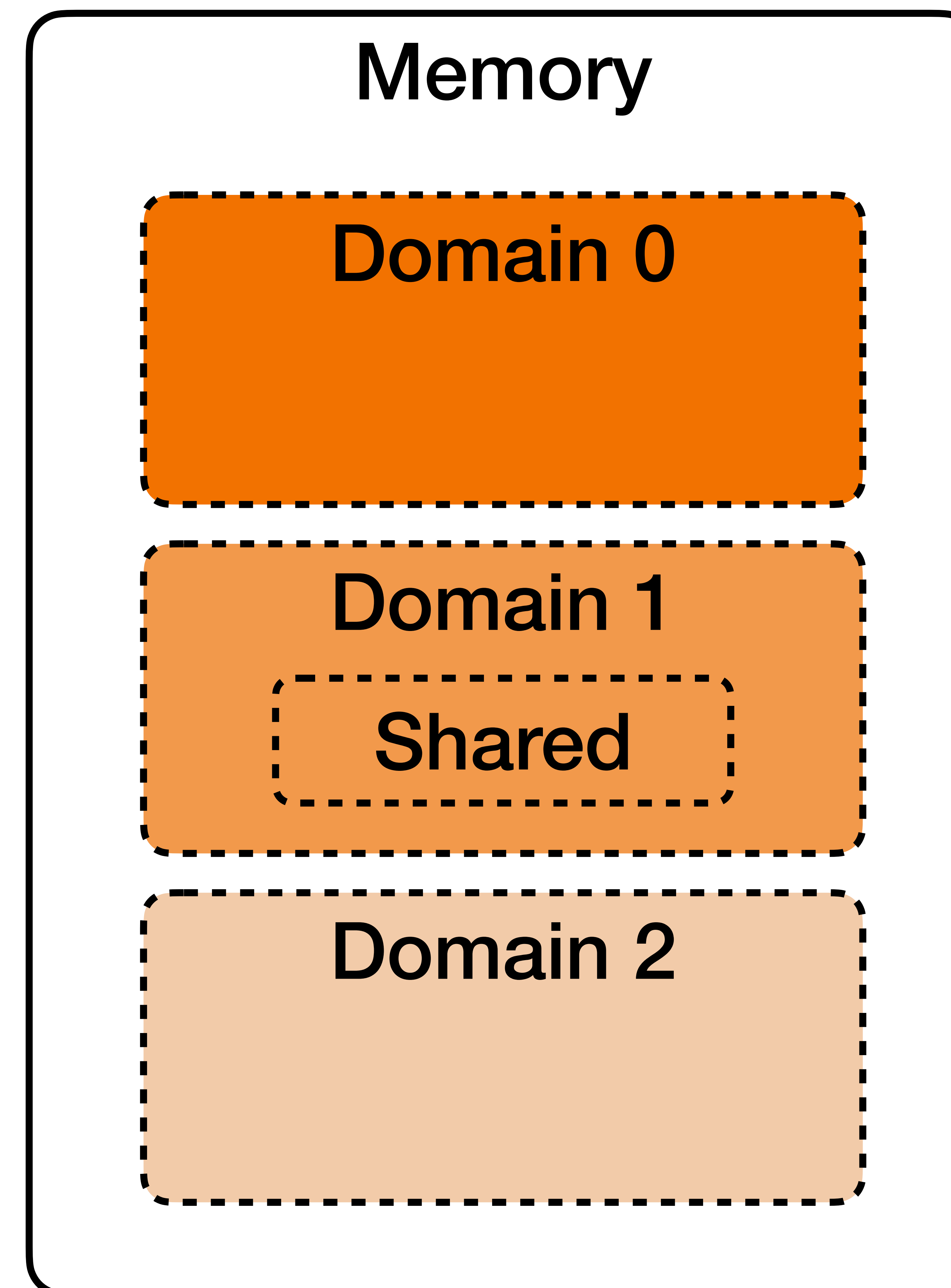
The Problem With Code-Oriented Partitioning



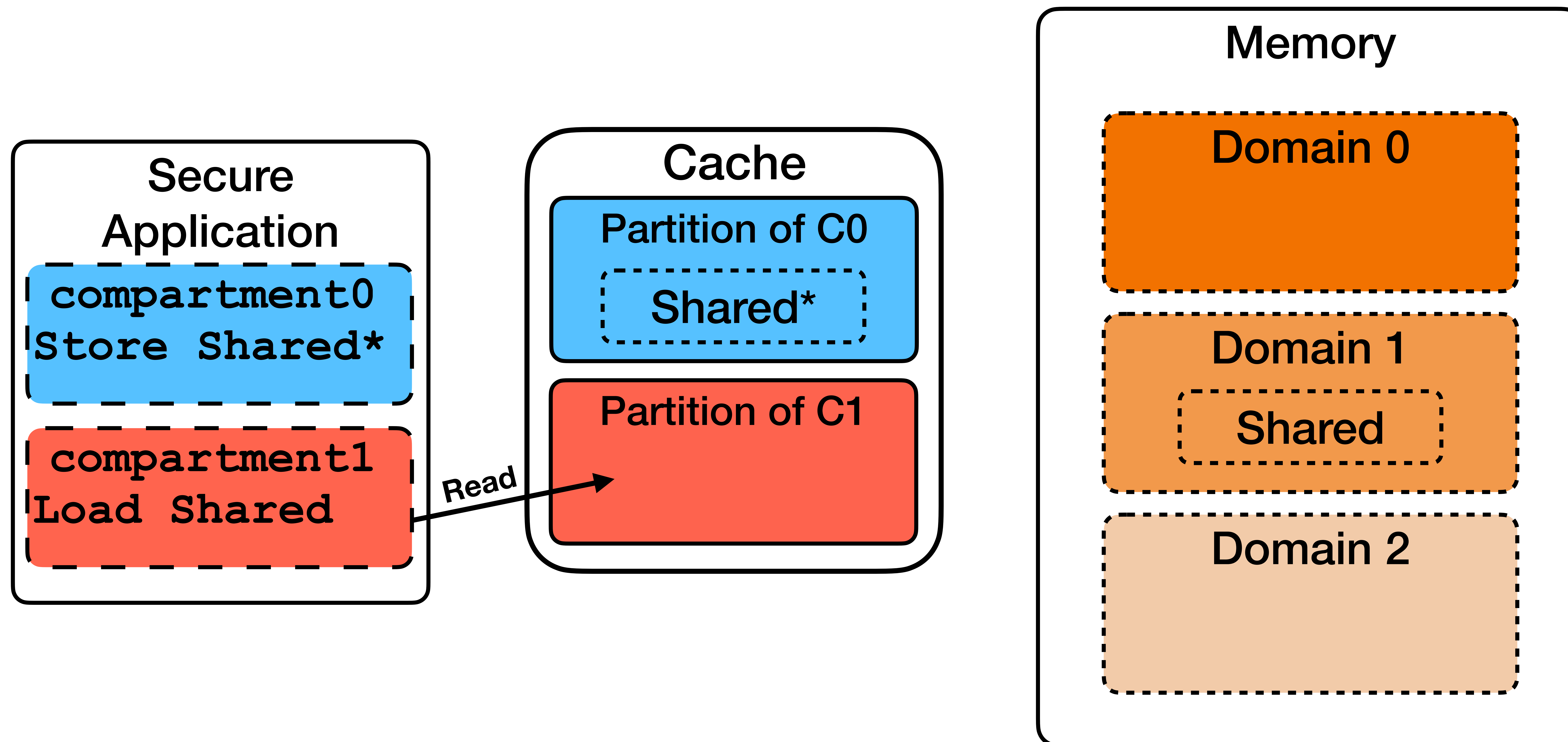
The Problem With Code-Oriented Partitioning





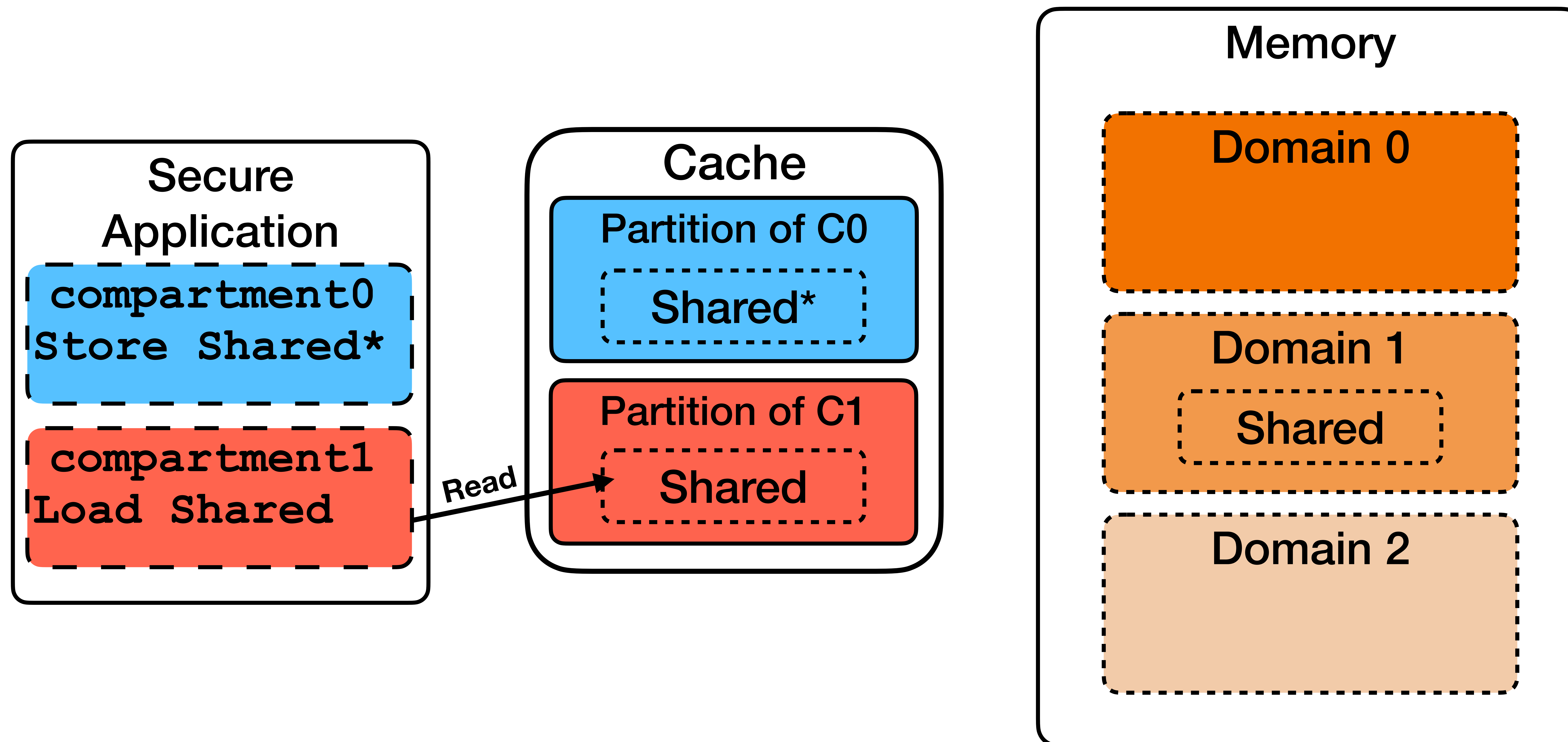


The Problem With Code-Oriented Partitioning



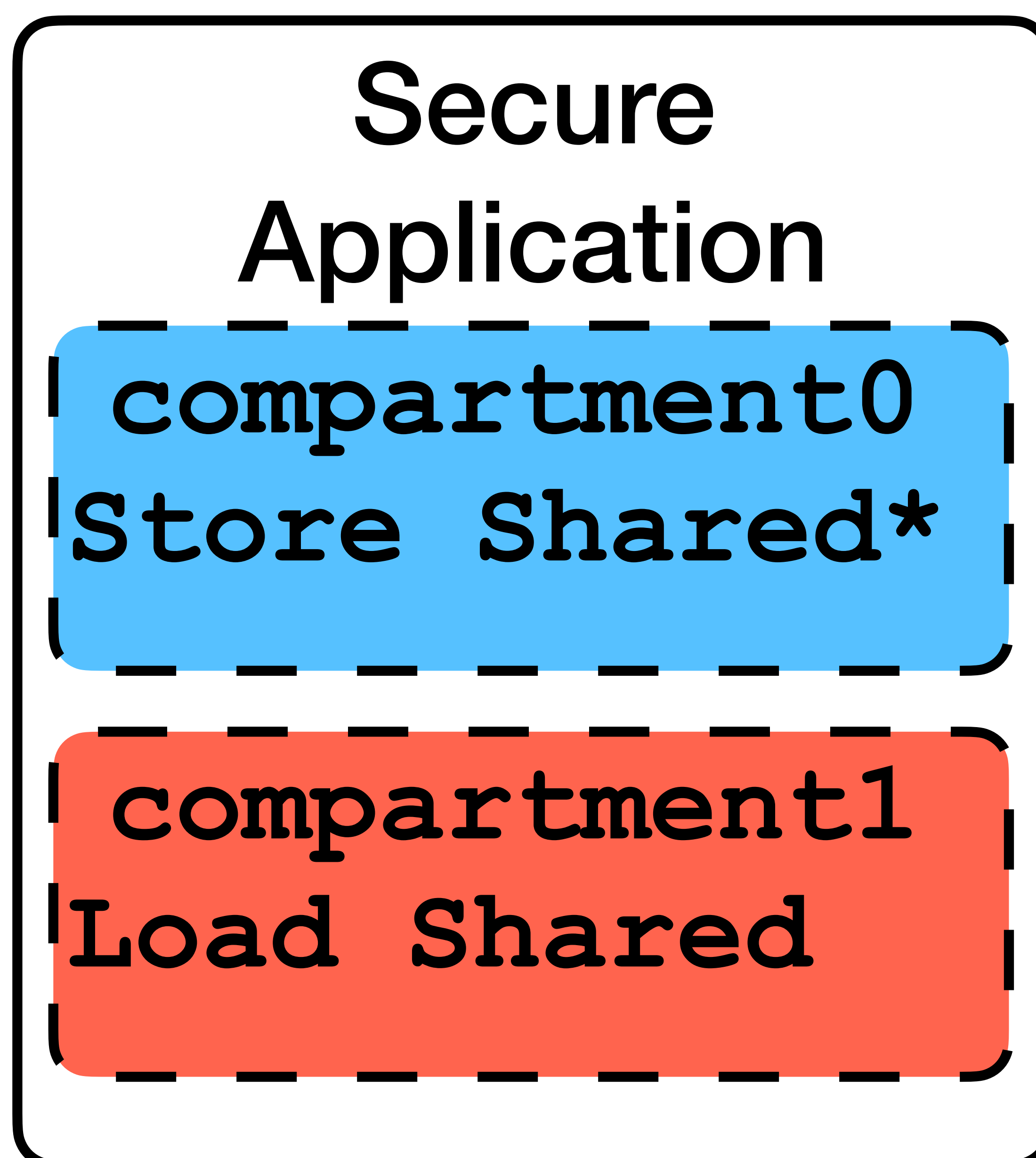
BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

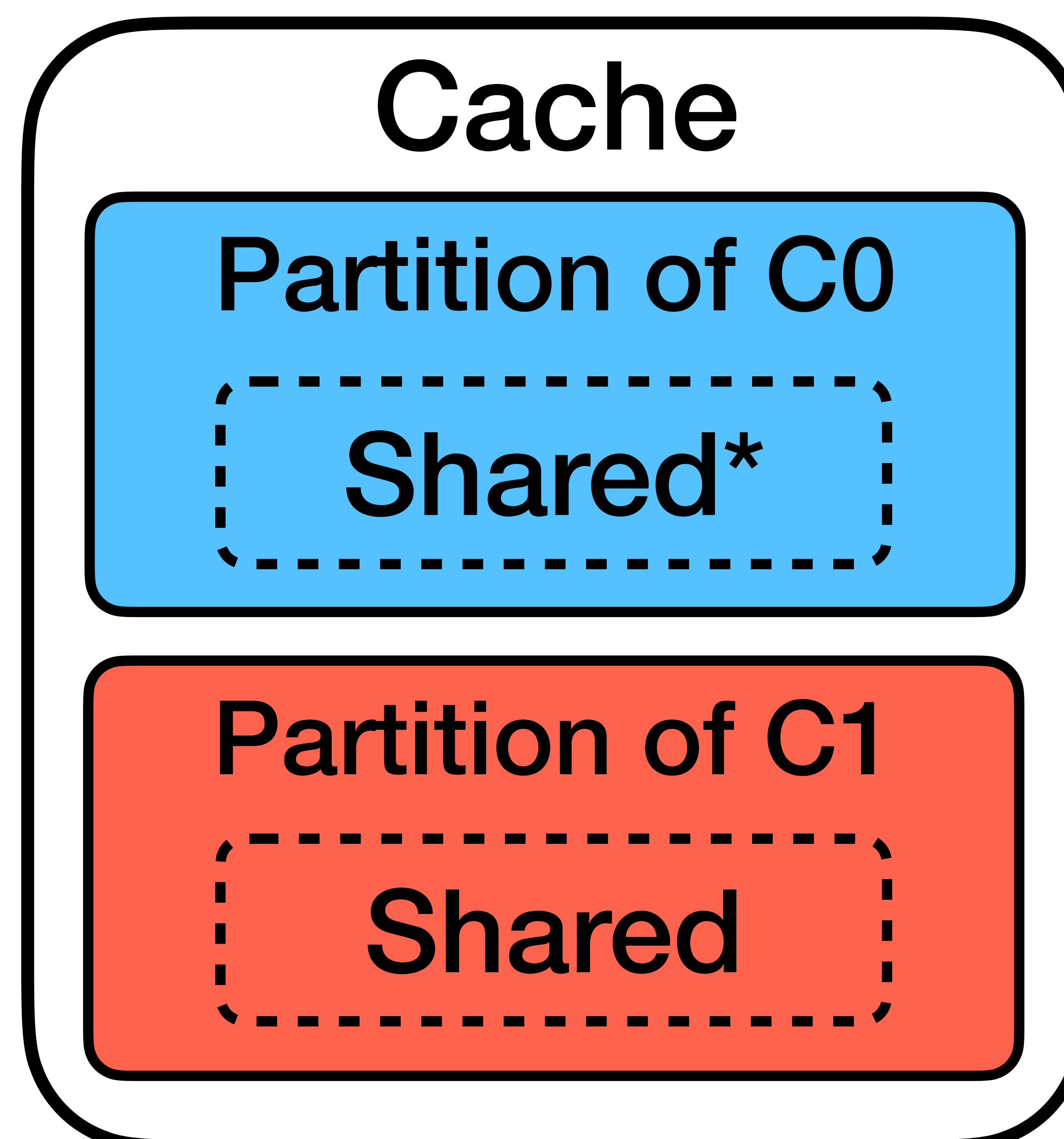
The Problem With Code-Oriented Partitioning

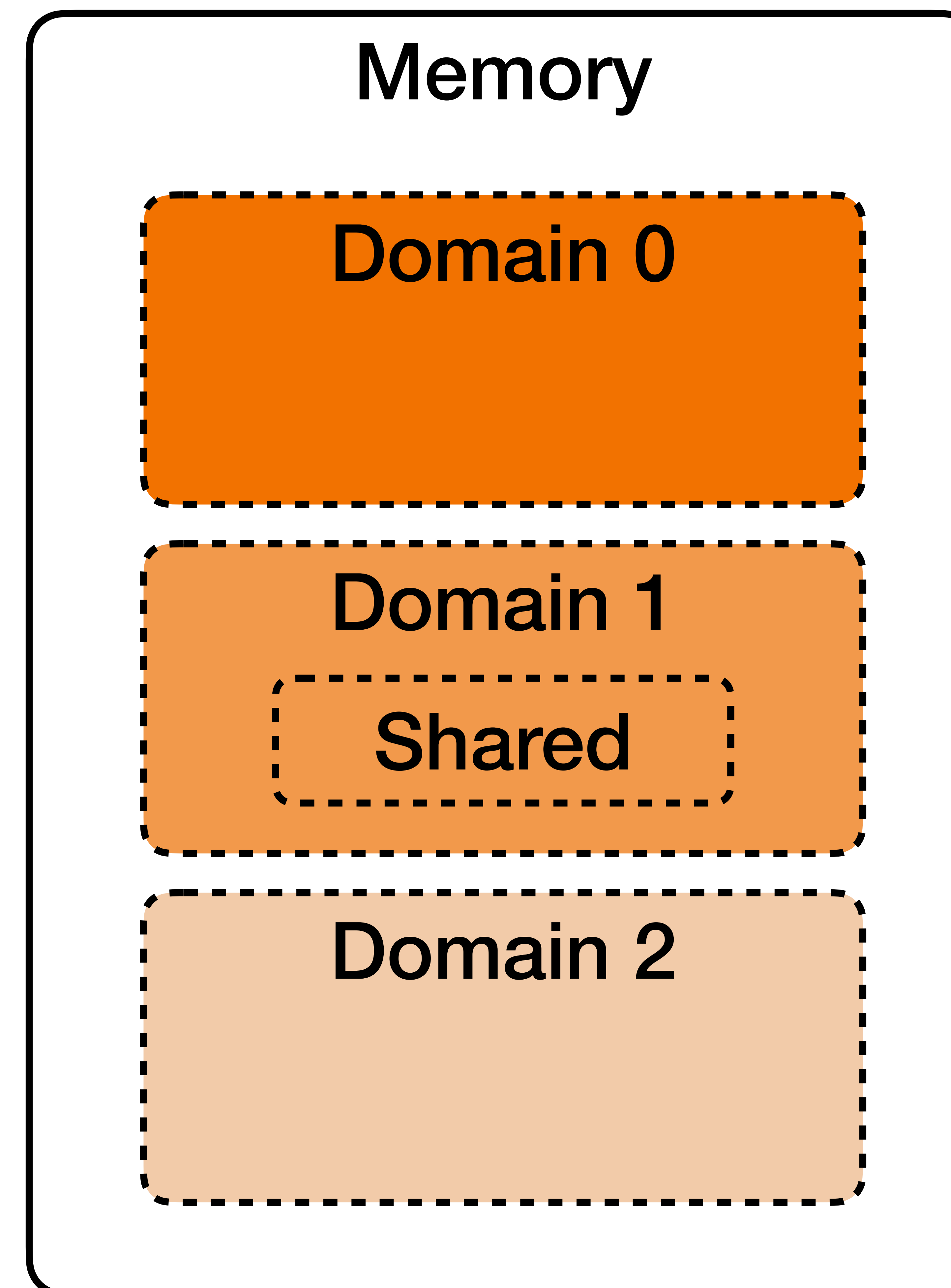


BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

The Problem With Code-Oriented Partitioning

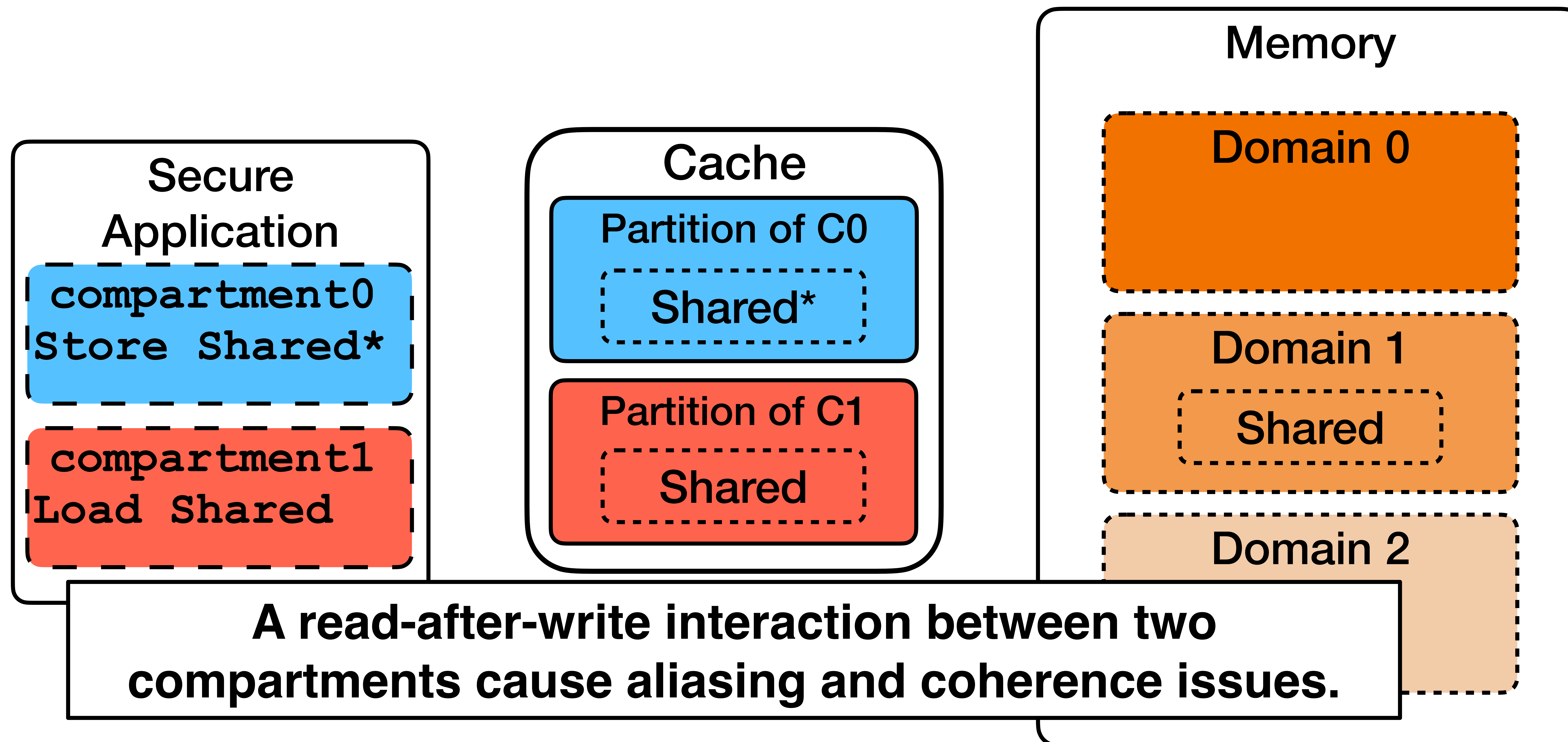






Binghamton University State University of New York | UC Riverside | usenix The Advanced Computing Systems Association

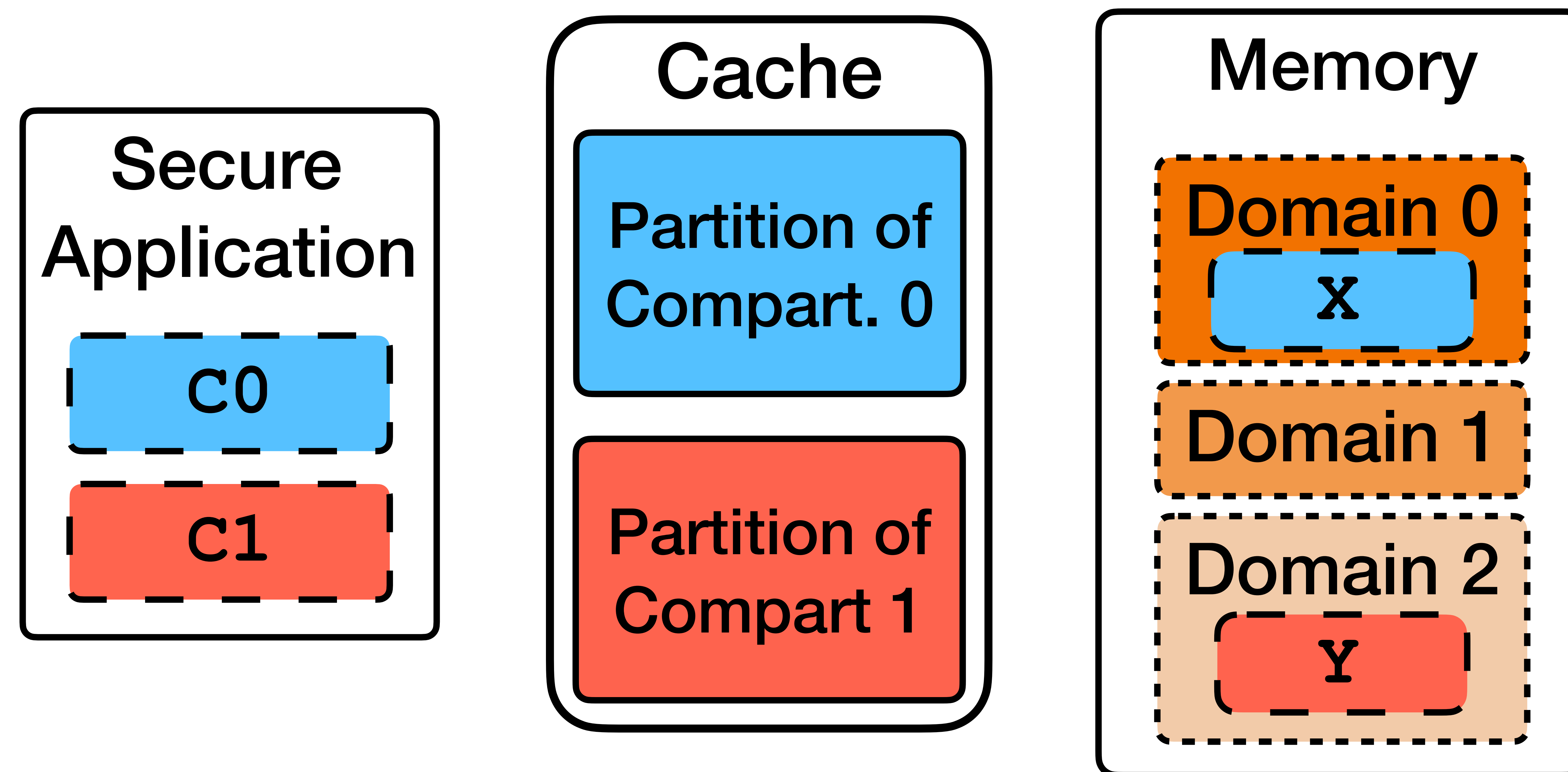
The Problem With Code-Oriented Partitioning



BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK UC RIVERSIDE usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

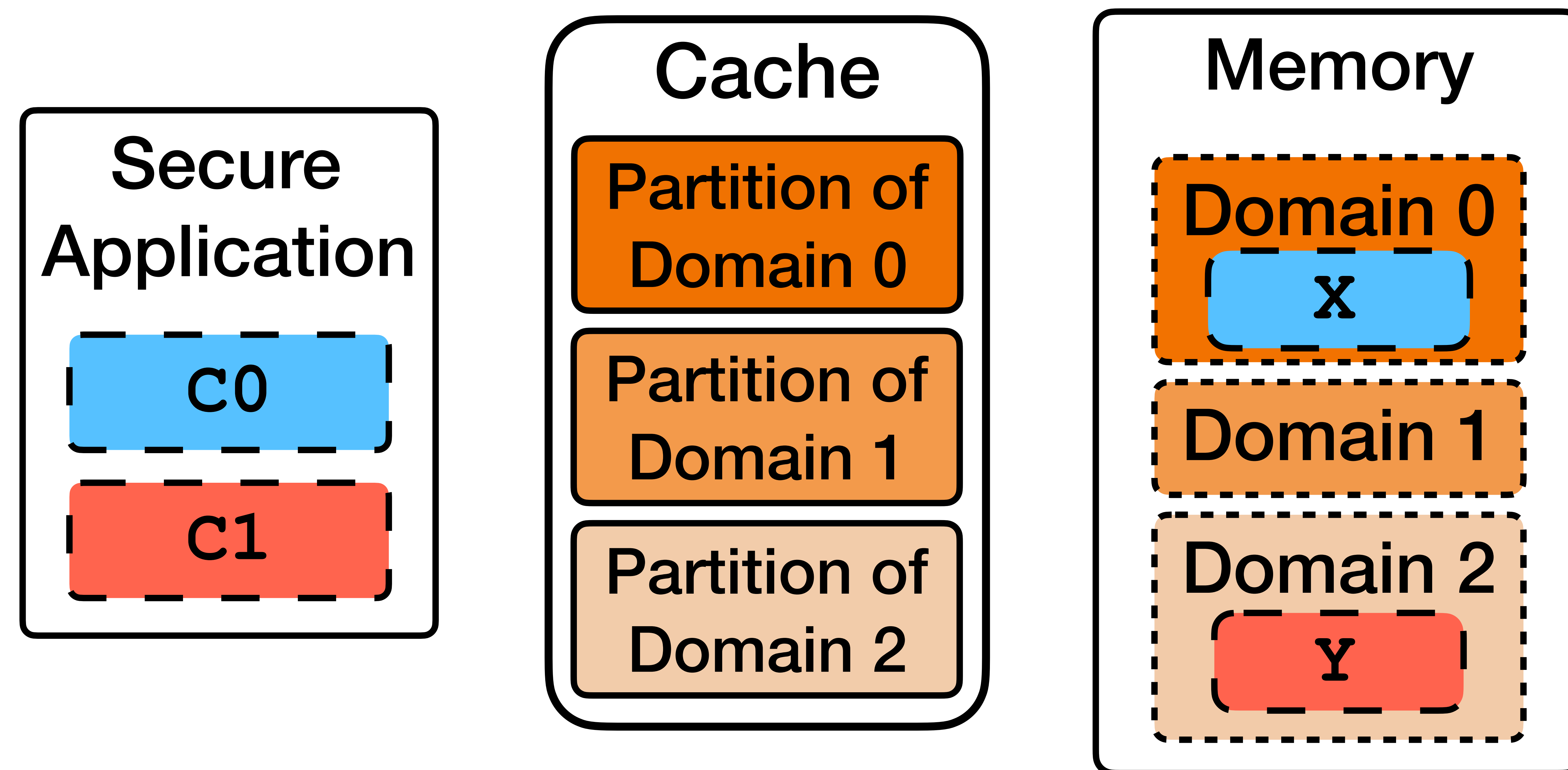
Domain-Oriented Partitioning

- Cache partitions are organized around domains rather than compartments:

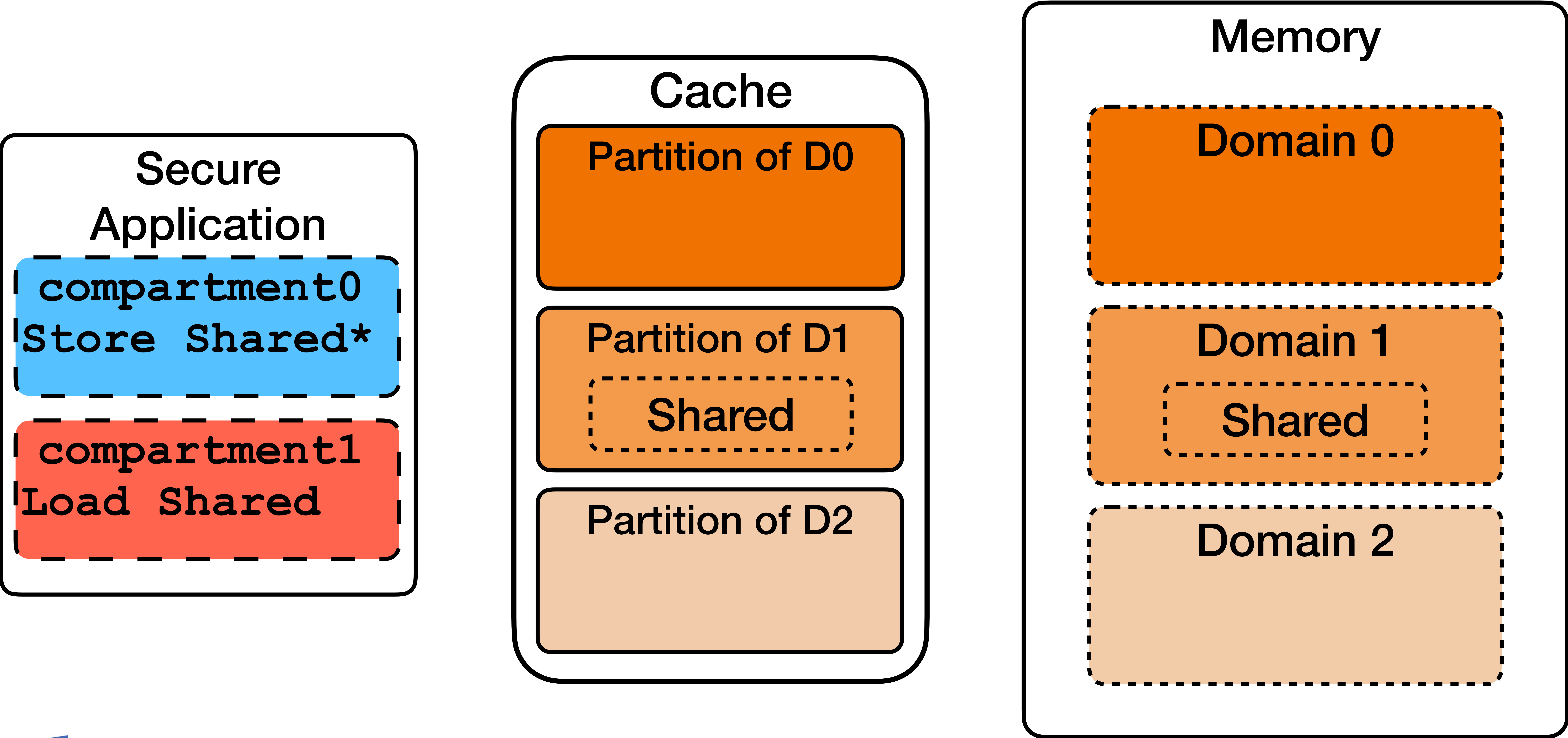


Domain-Oriented Partitioning

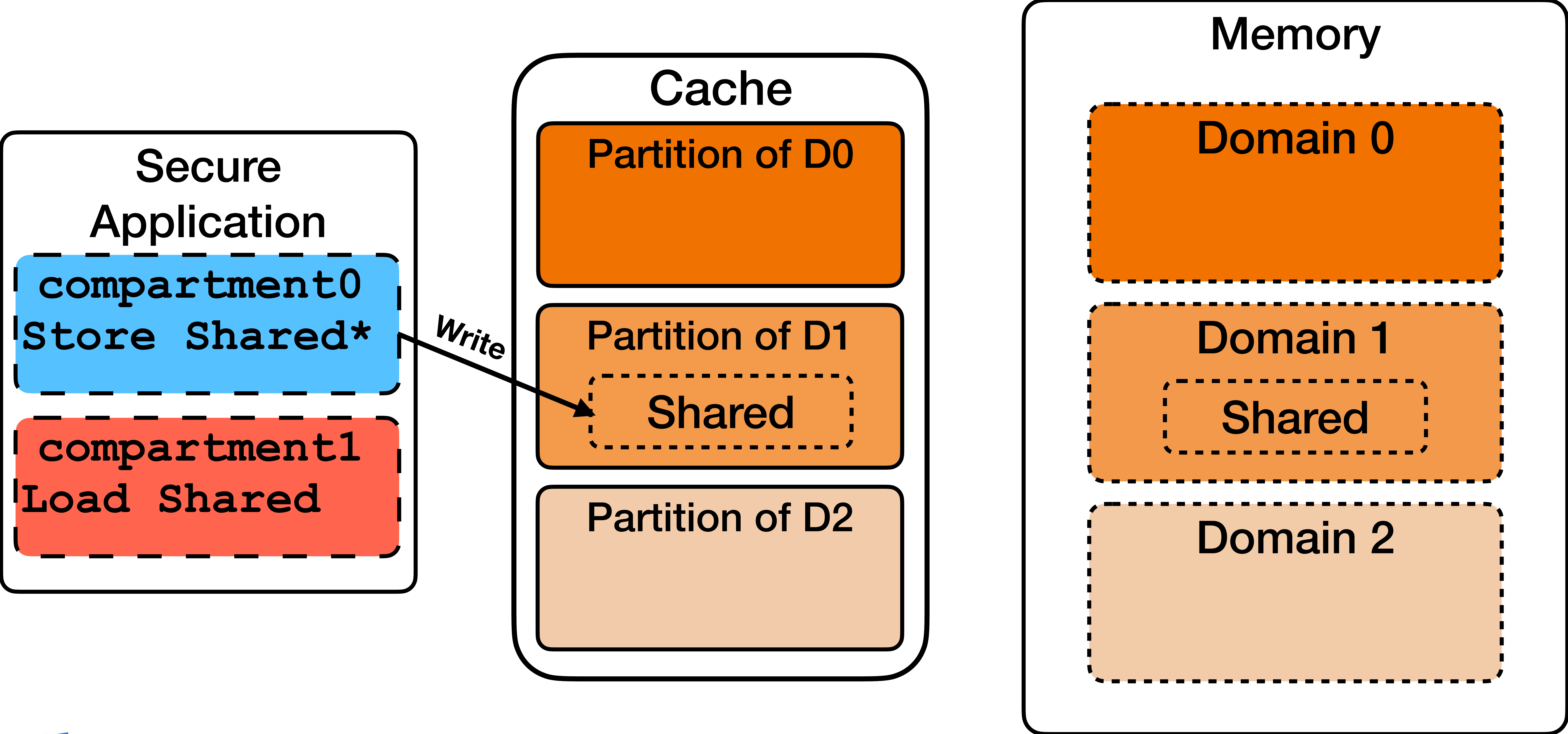
- Cache partitions are organized around domains rather than compartments:



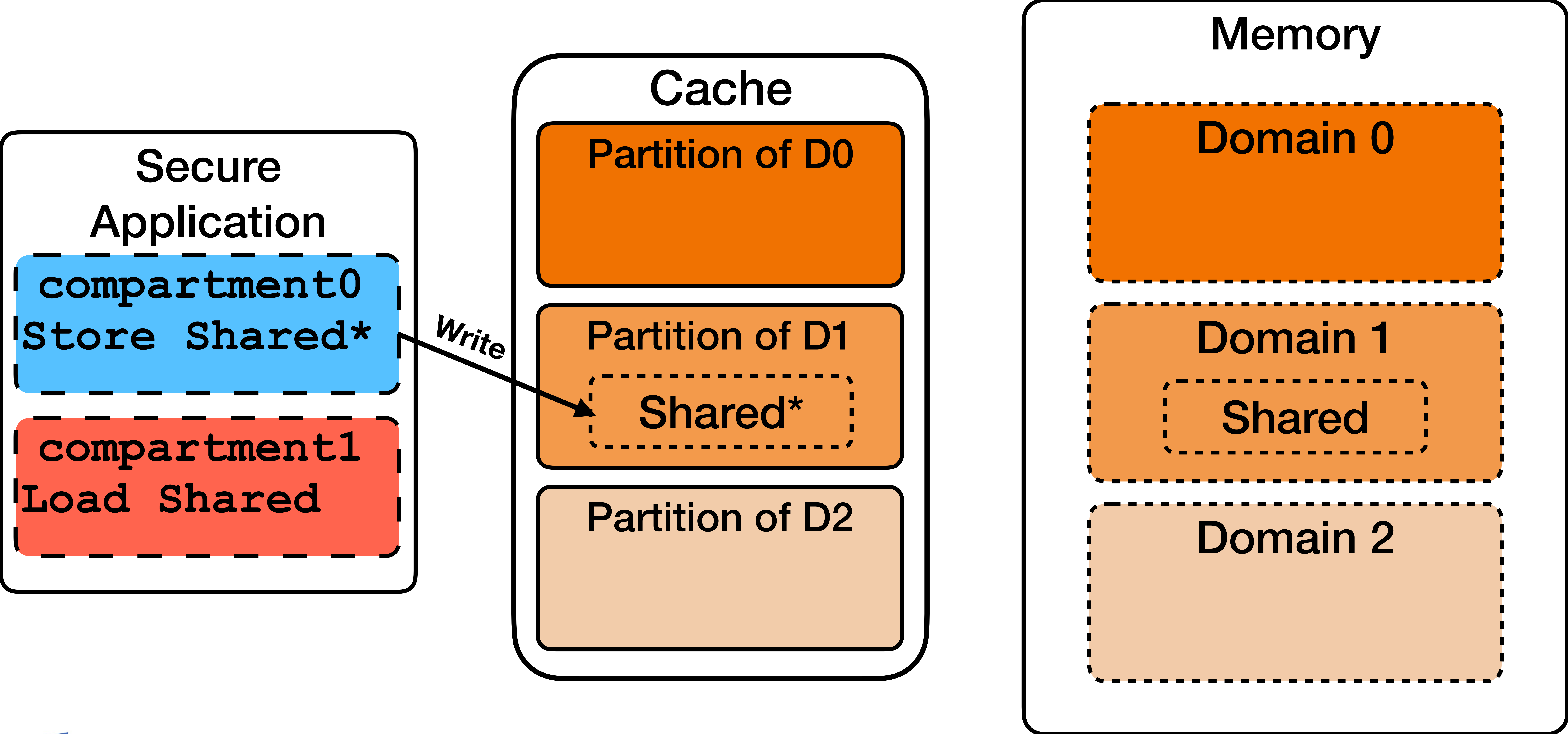
Coherence Under Domain-Oriented Partitioning



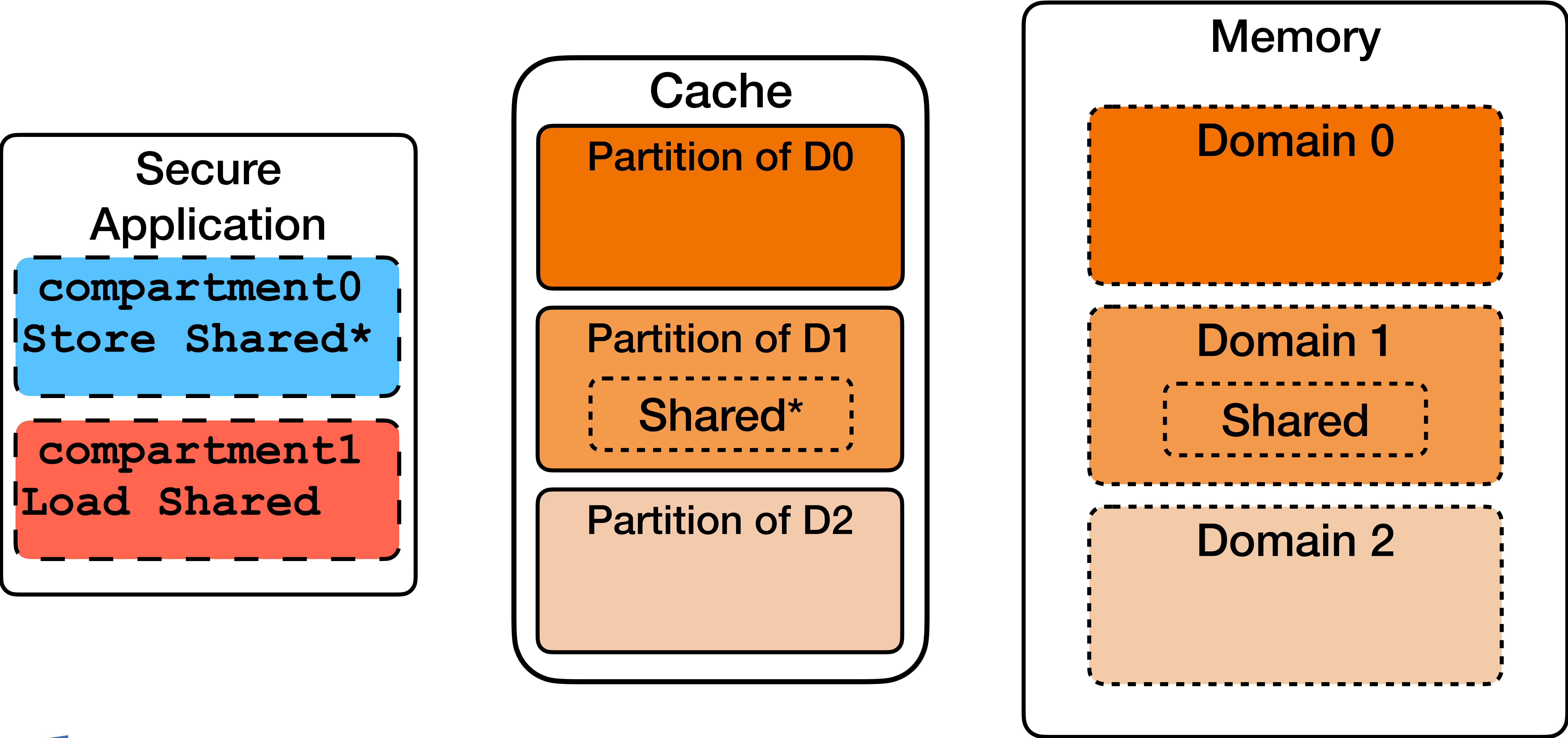
Coherence Under Domain-Oriented Partitioning



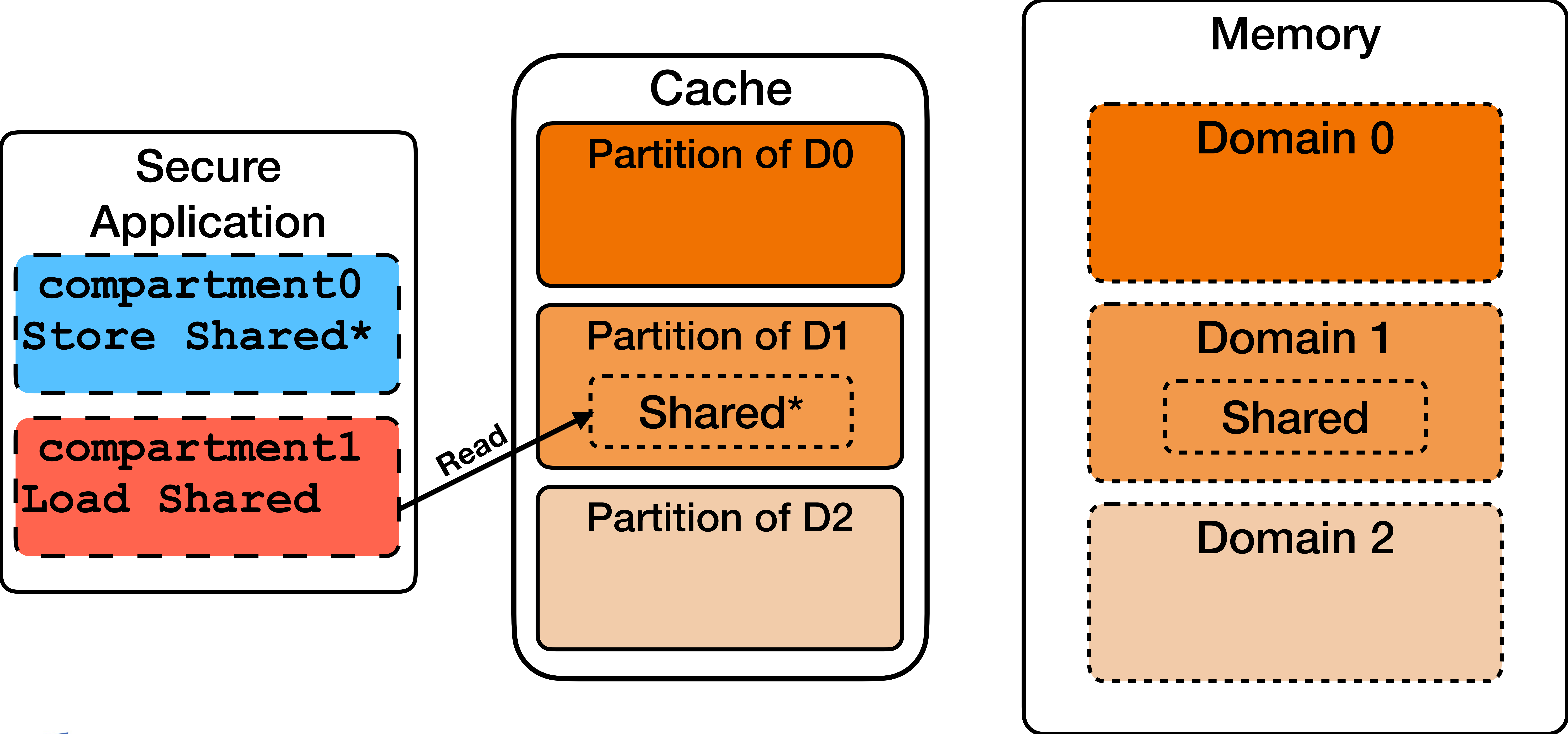
Coherence Under Domain-Oriented Partitioning



Coherence Under Domain-Oriented Partitioning

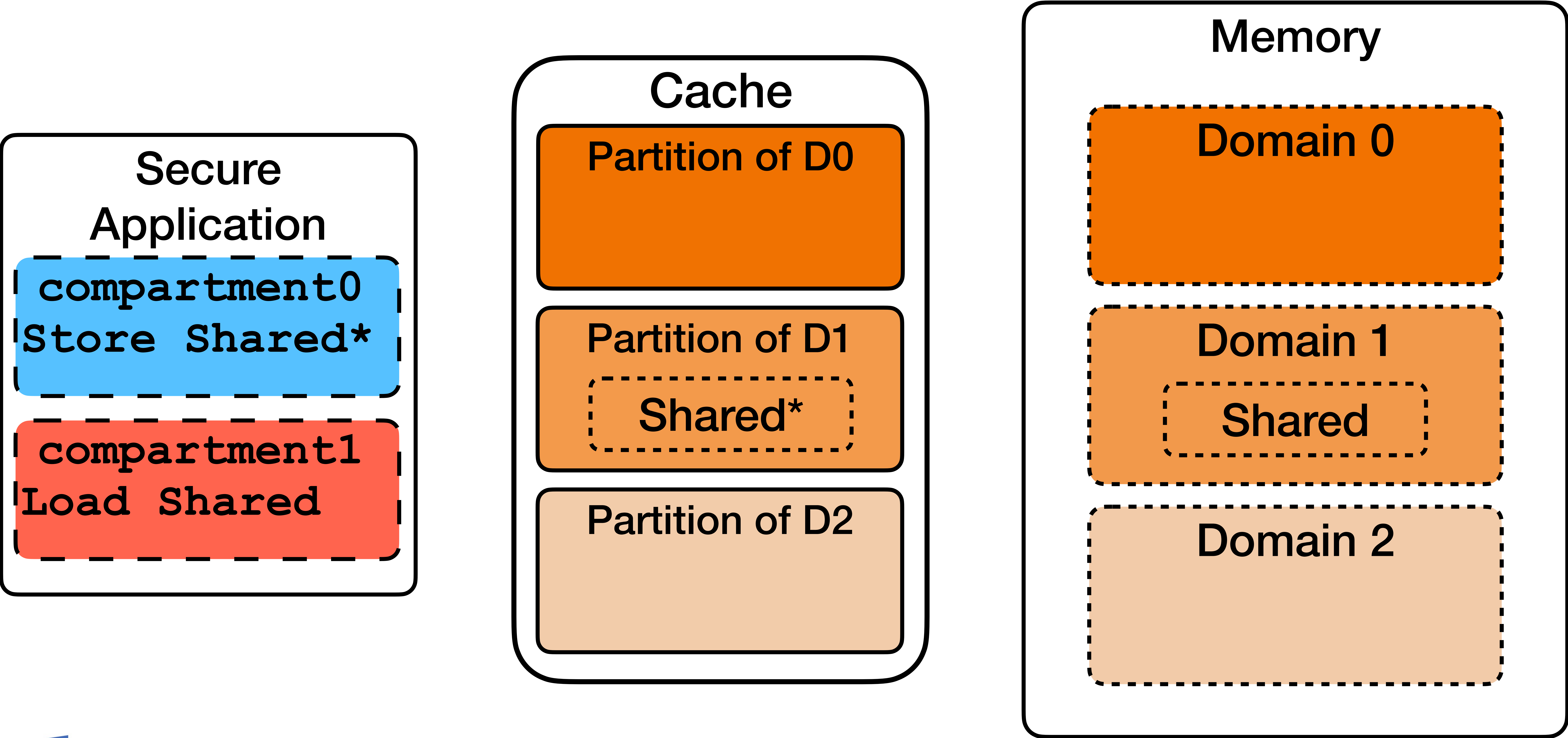


Coherence Under Domain-Oriented Partitioning



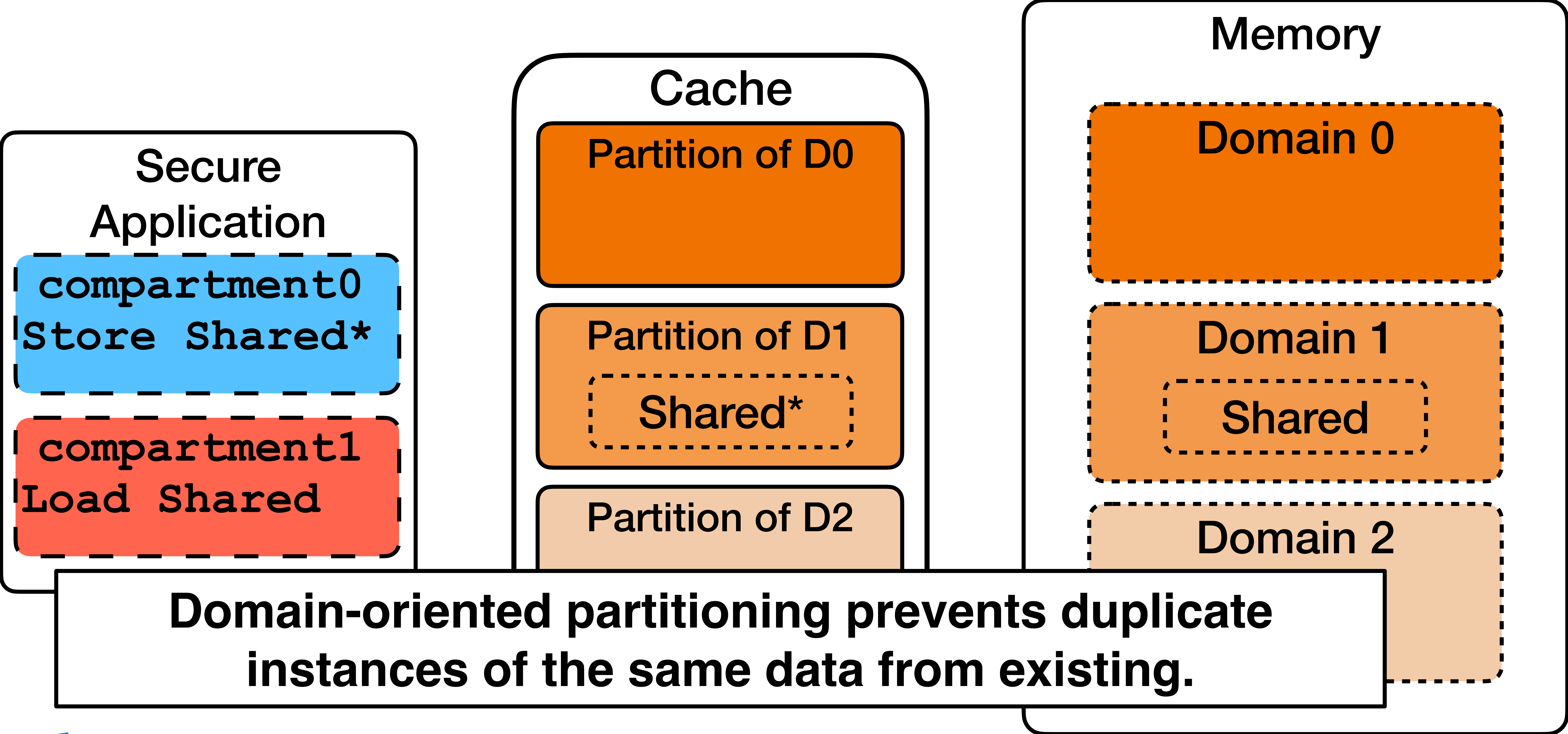
BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Coherence Under Domain-Oriented Partitioning



BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

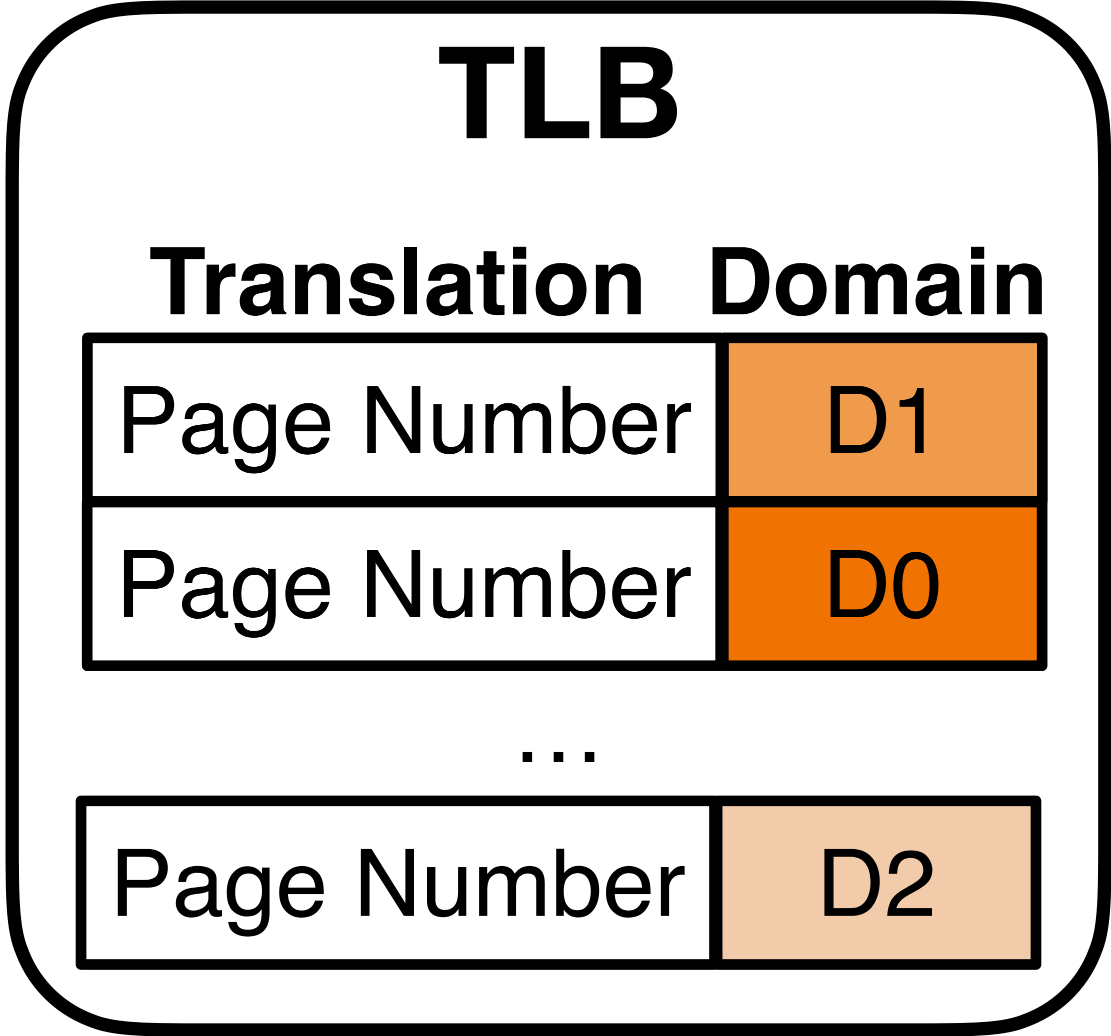
Coherence Under Domain-Oriented Partitioning

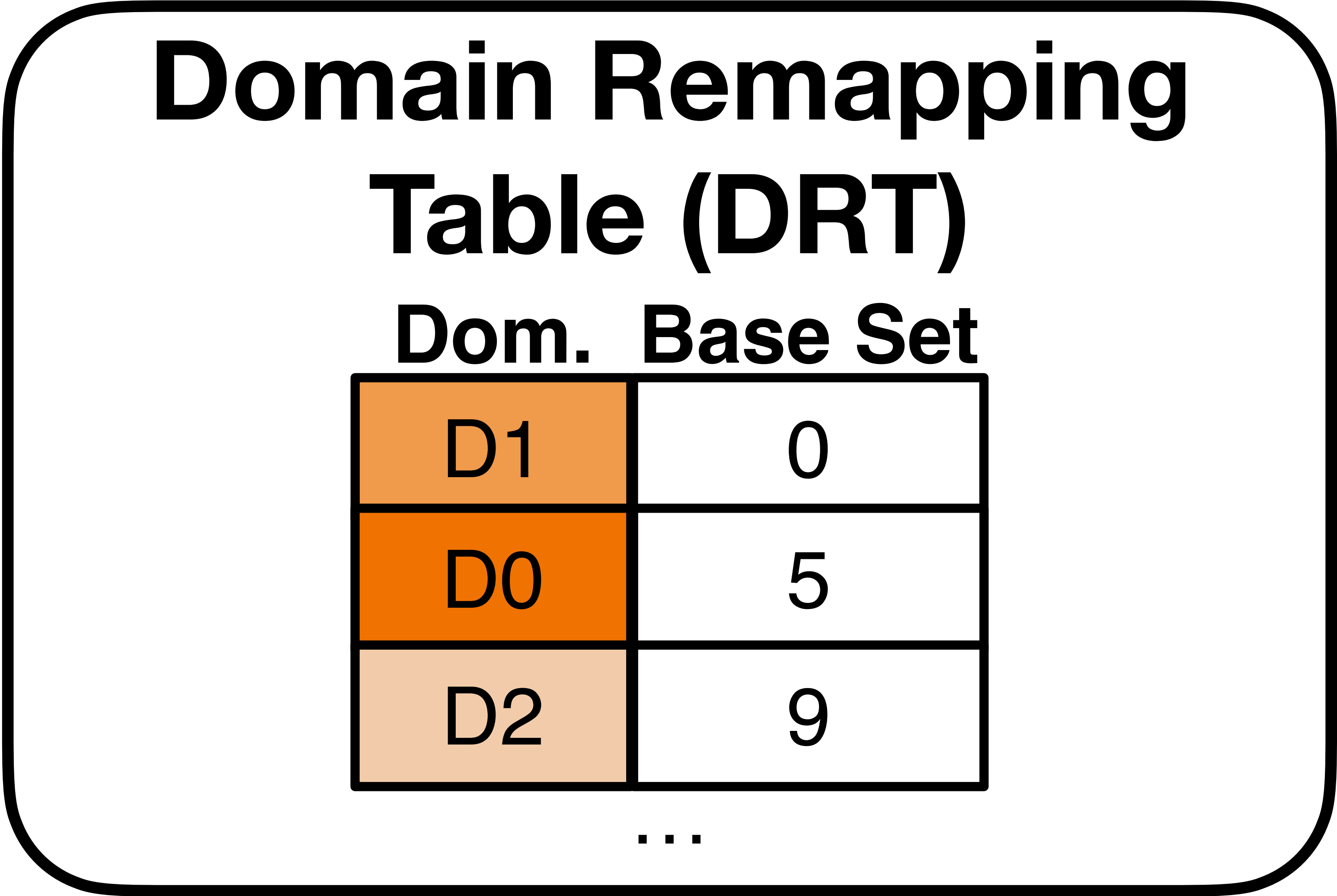


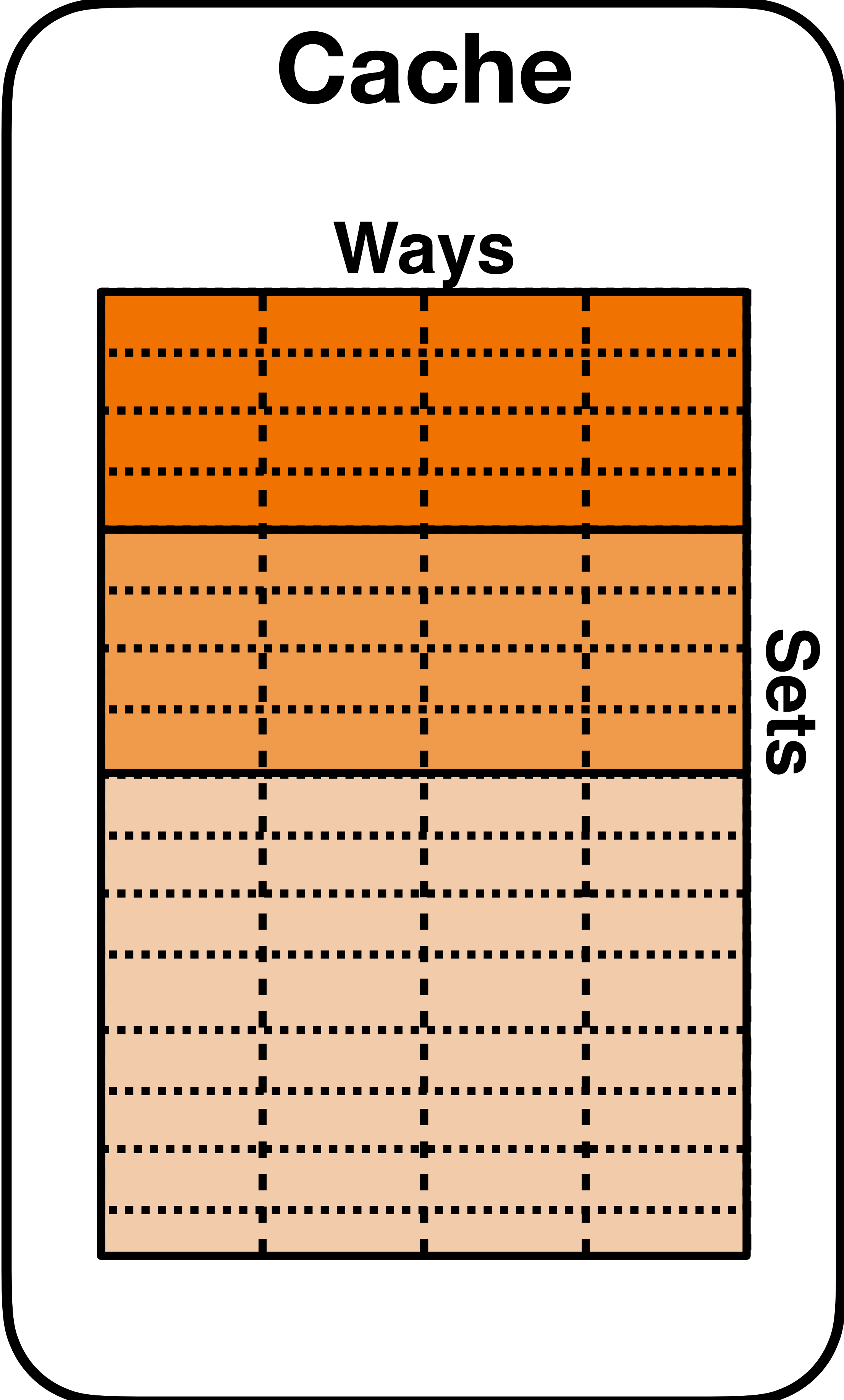
Angle-2: Latency-Aware L1 Cache Partitioning

Access to an **Unoptimized** Partitioned Cache

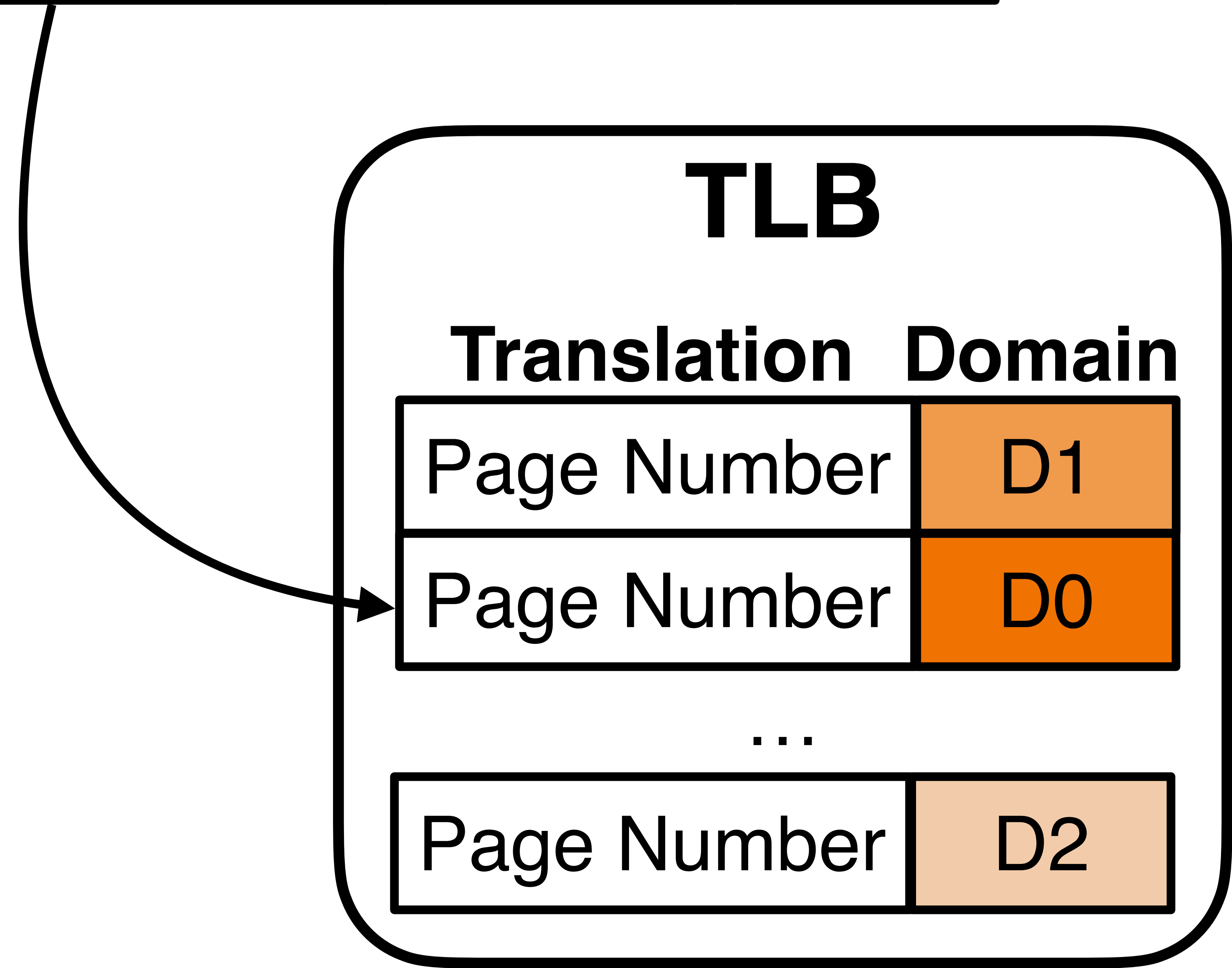
Access to an Unoptimized Partitioned Cache

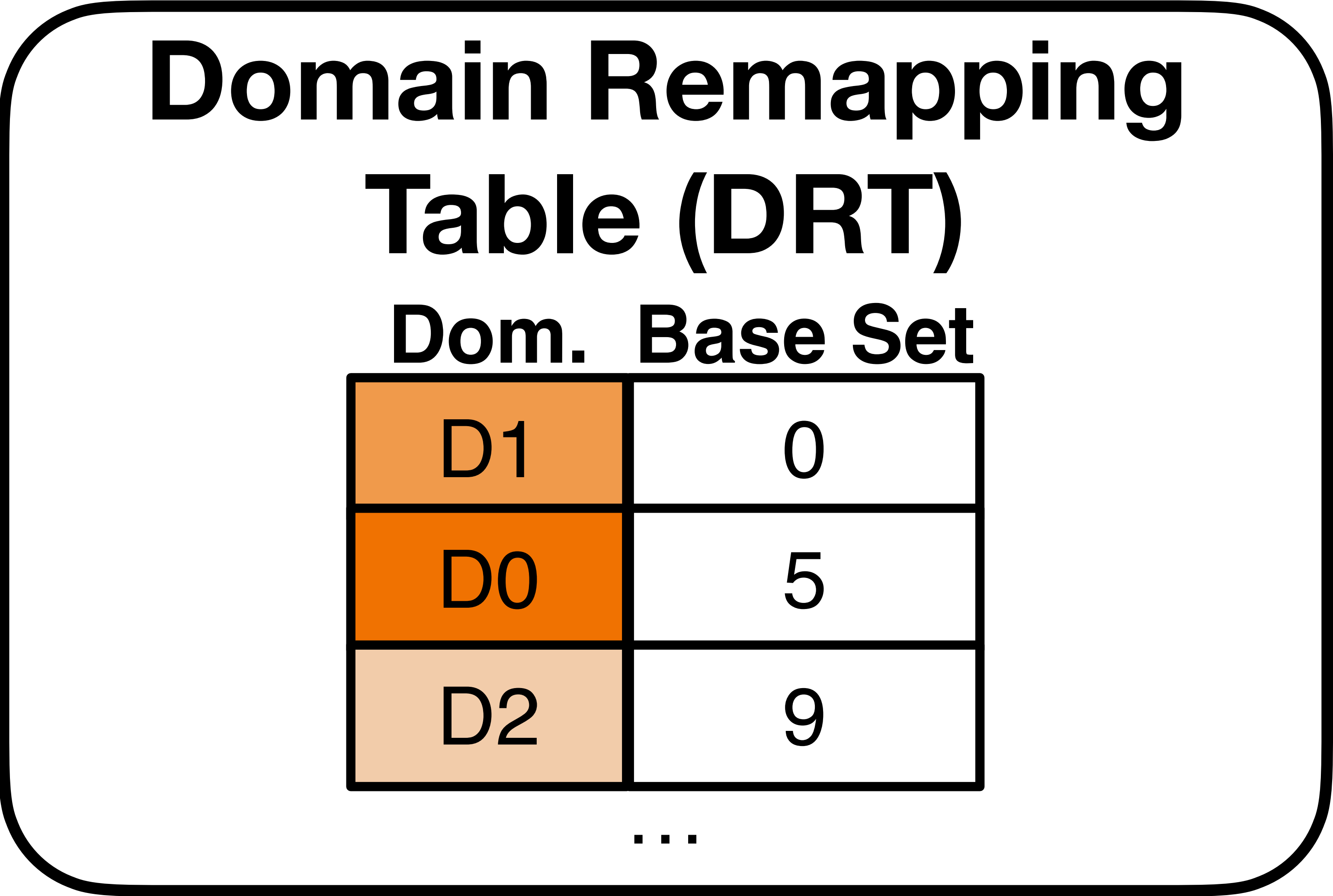


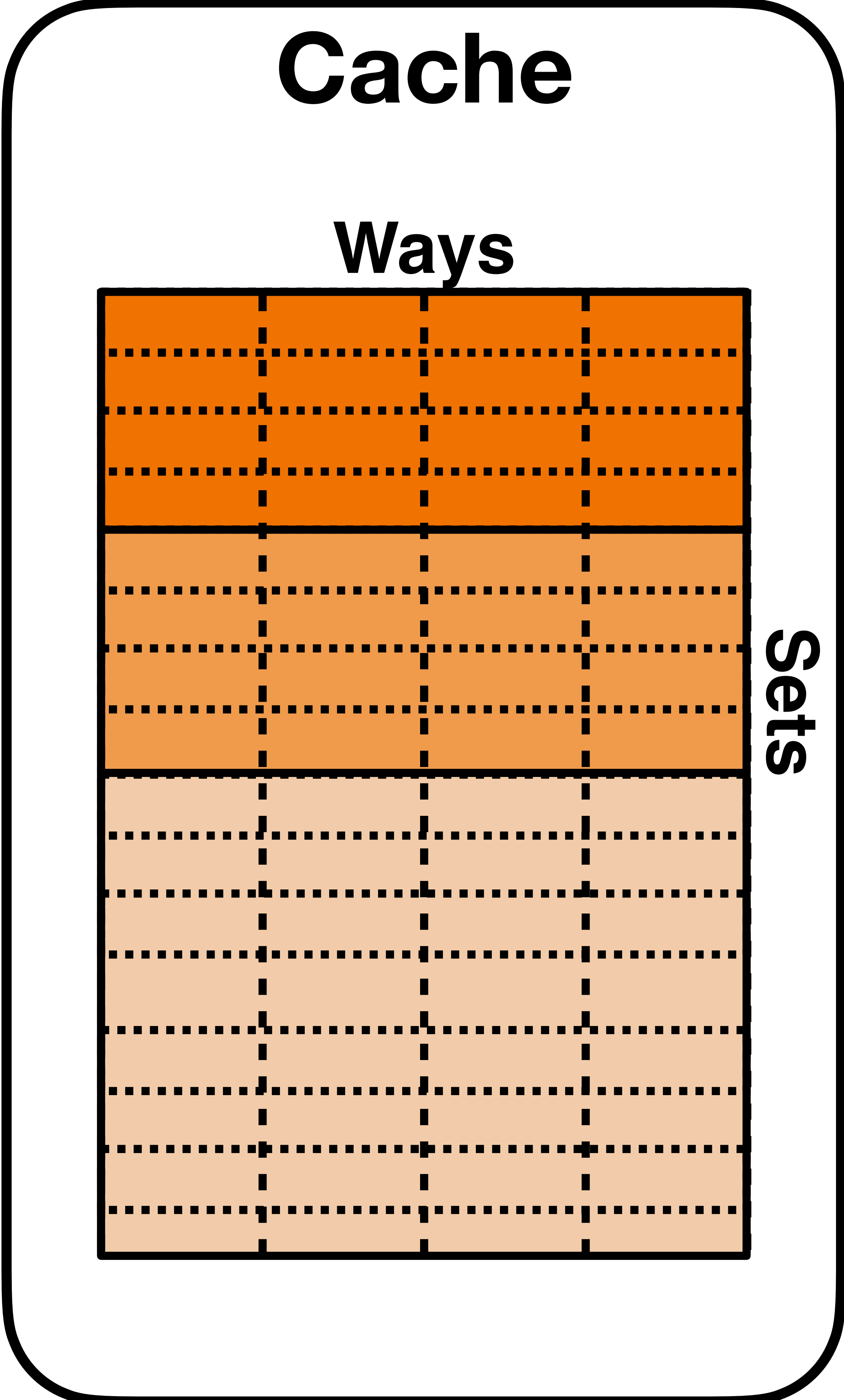




Access to an Unoptimized Partitioned Cache

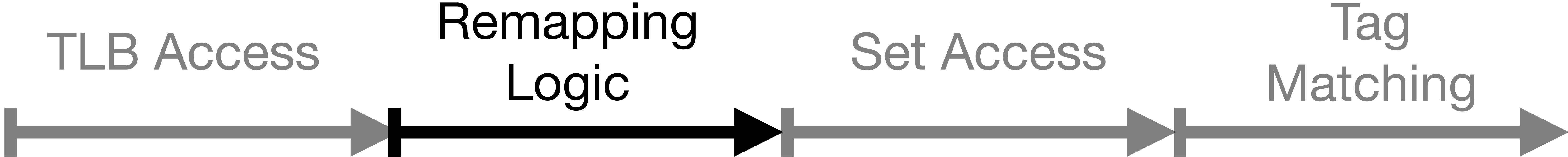


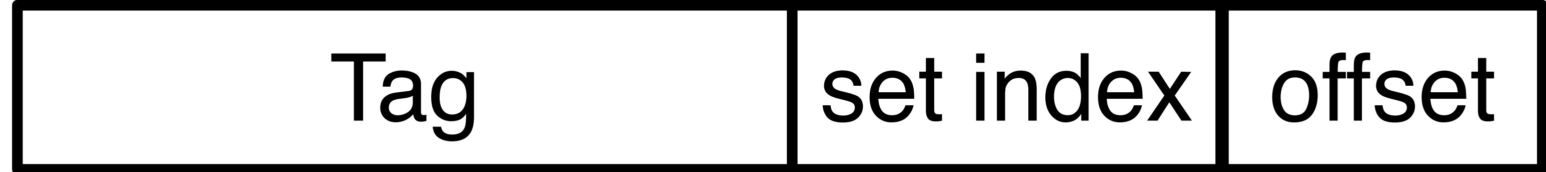




BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Access to an Unoptimized Partitioned Cache



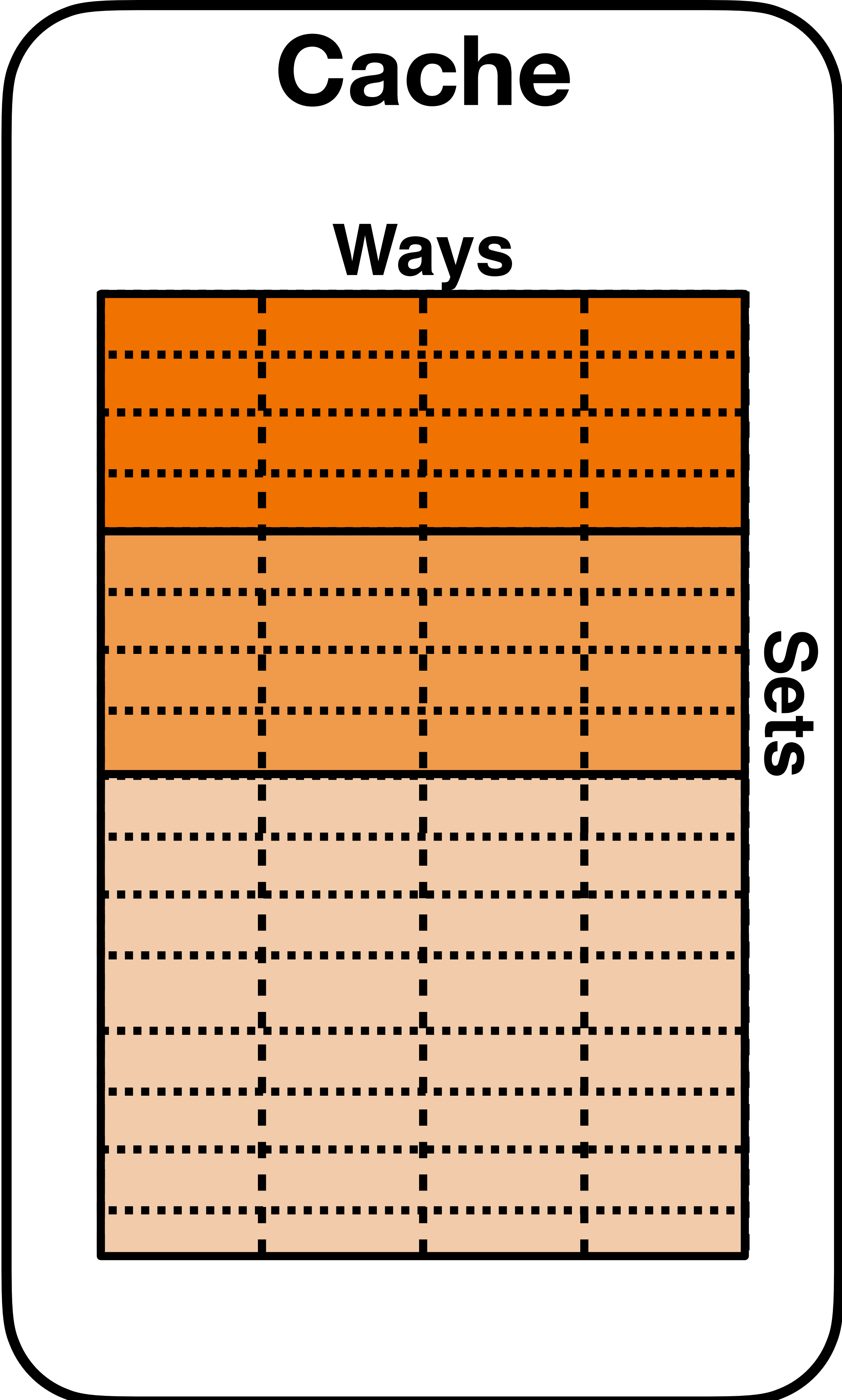


TLB

Translation	Domain
Page Number	D1
Page Number	D0
…	
Page Number	D2

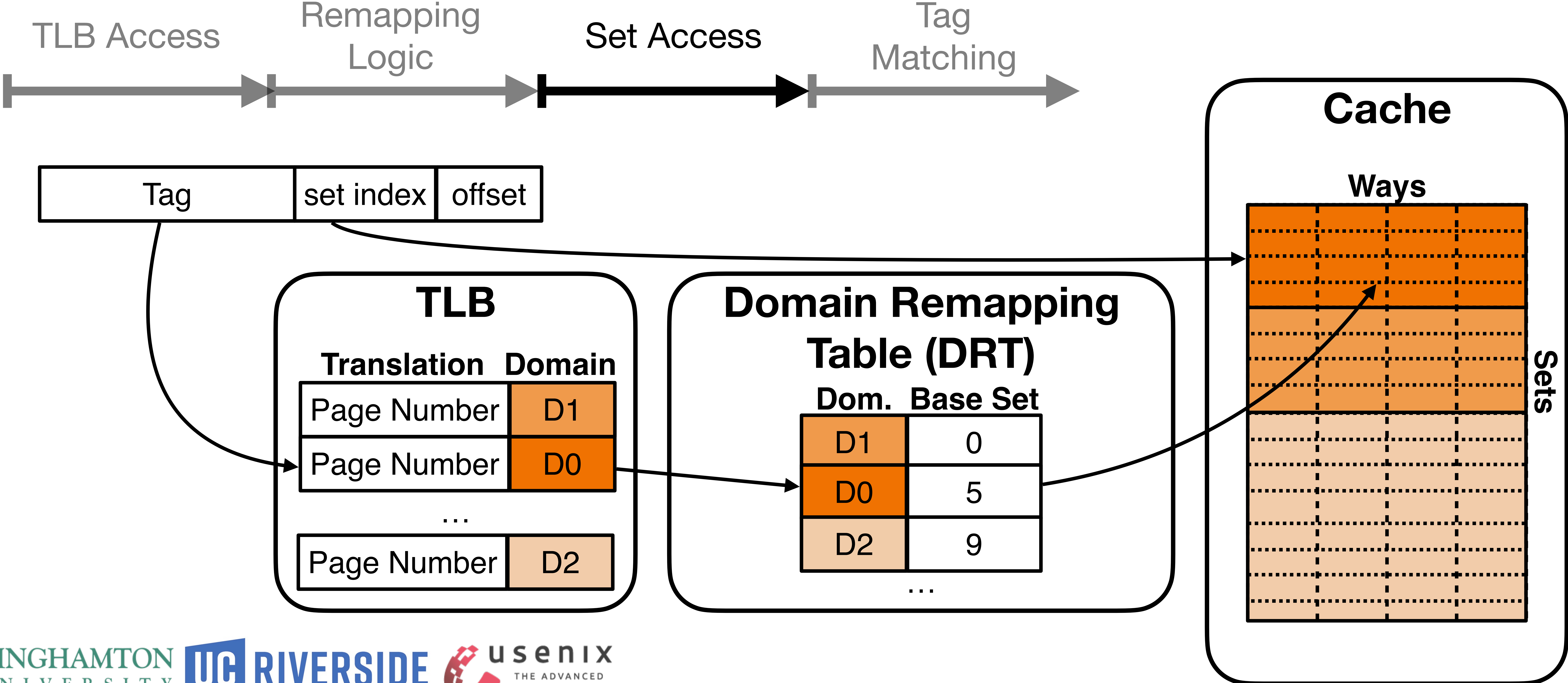
Domain Remapping Table (DRT)

Dom.	Base Set
D1	0
D0	5
D2	9
…	



Binghamton University
State University of New York
UC Riverside
usenix
The Advanced Computing Systems Association

Access to an Unoptimized Partitioned Cache



BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

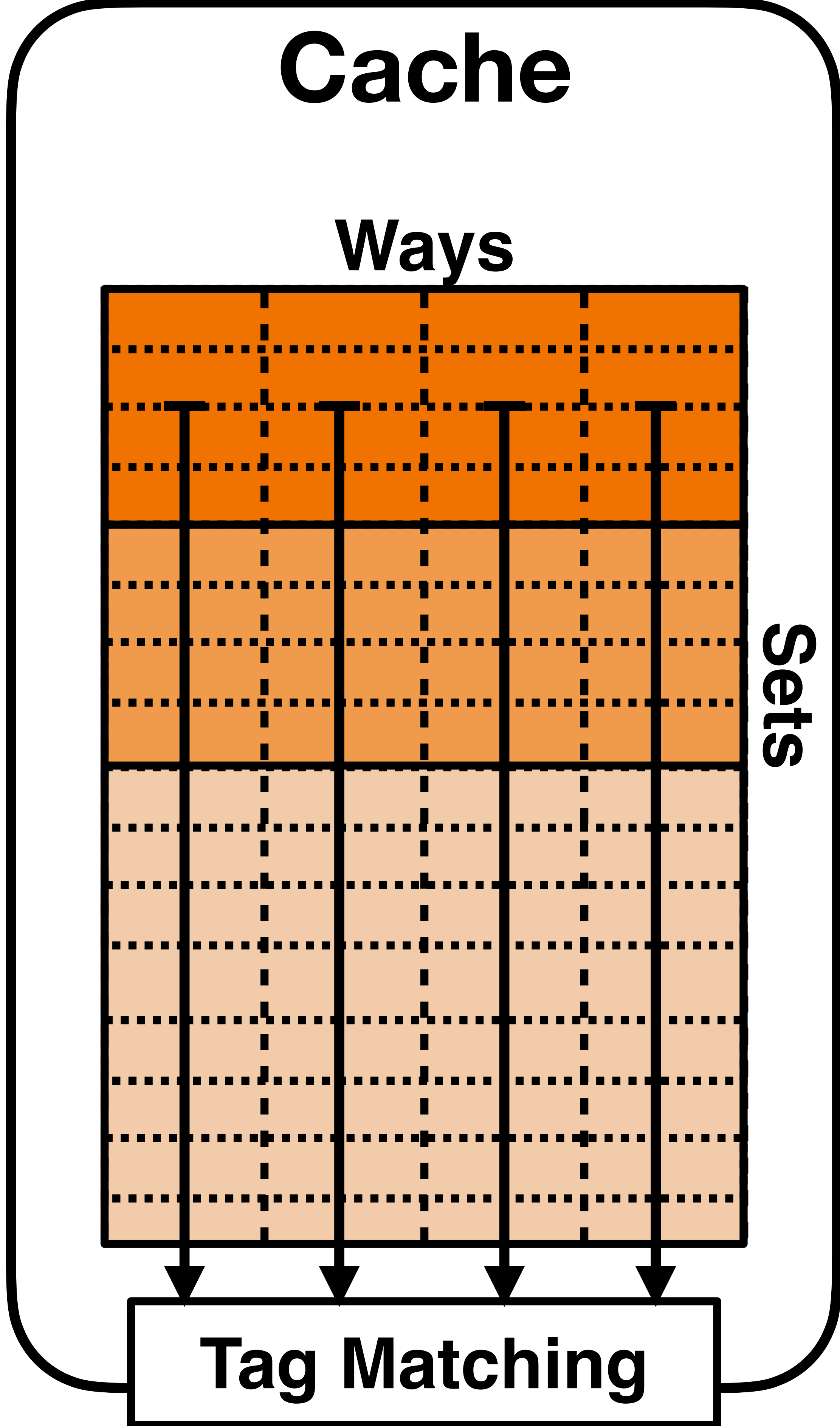
Access to an Unoptimized Partitioned Cache

TLB

Translation	Domain
Page Number	D1
Page Number	D0
...	
Page Number	D2

Domain Remapping Table (DRT)

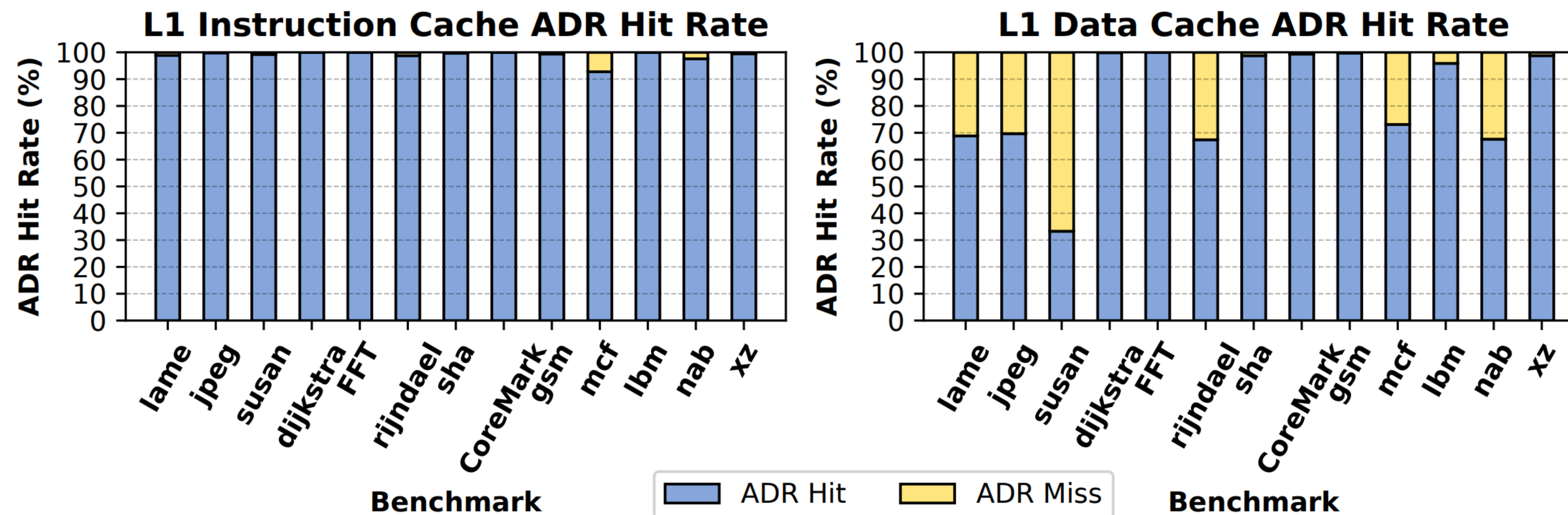
Dom.	Base Set
D1	0
D0	5
D2	9
...	



BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

How to Optimize L1 Cache Accesses?

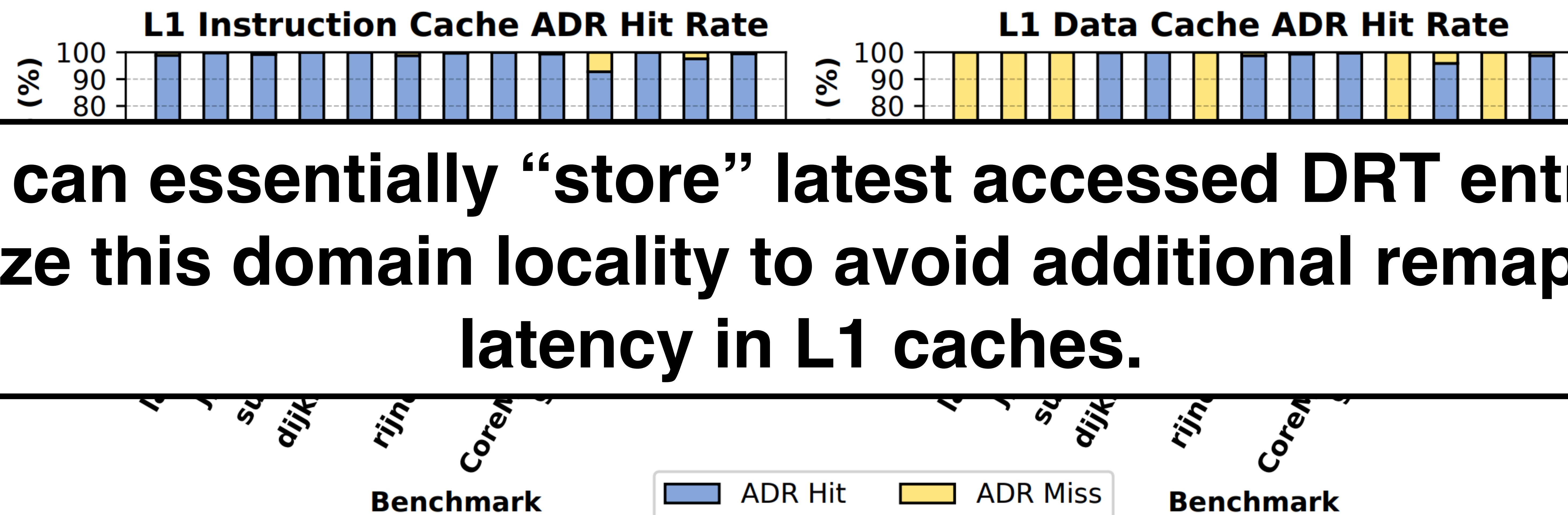
- **Observation:** Compartments tend to access the same domains for extended periods of time.
- SCC can utilize domain access locality to avoid DRT accesses.



How to Optimize L1 Cache Accesses?

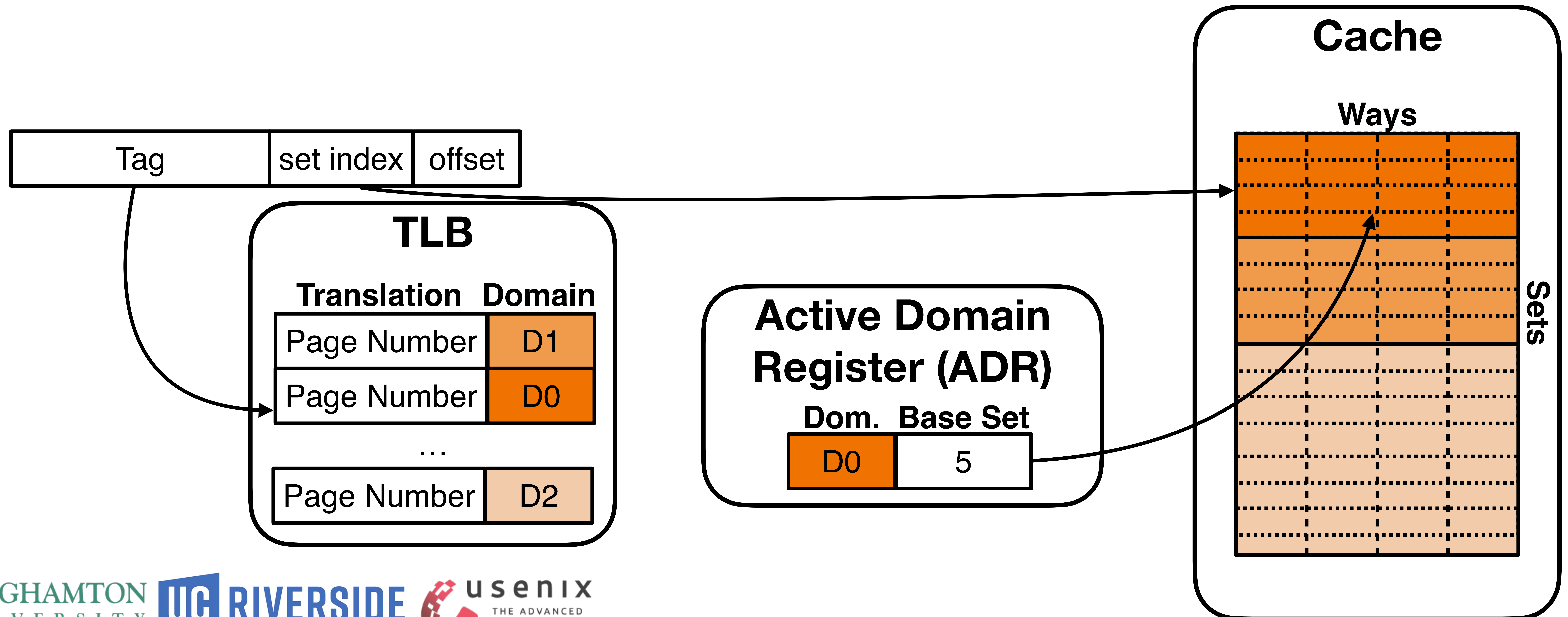
- **Observation:** Compartments tend to access the same domains for extended periods of time.

- SCC can utilize domain access locality to avoid DRT accesses.

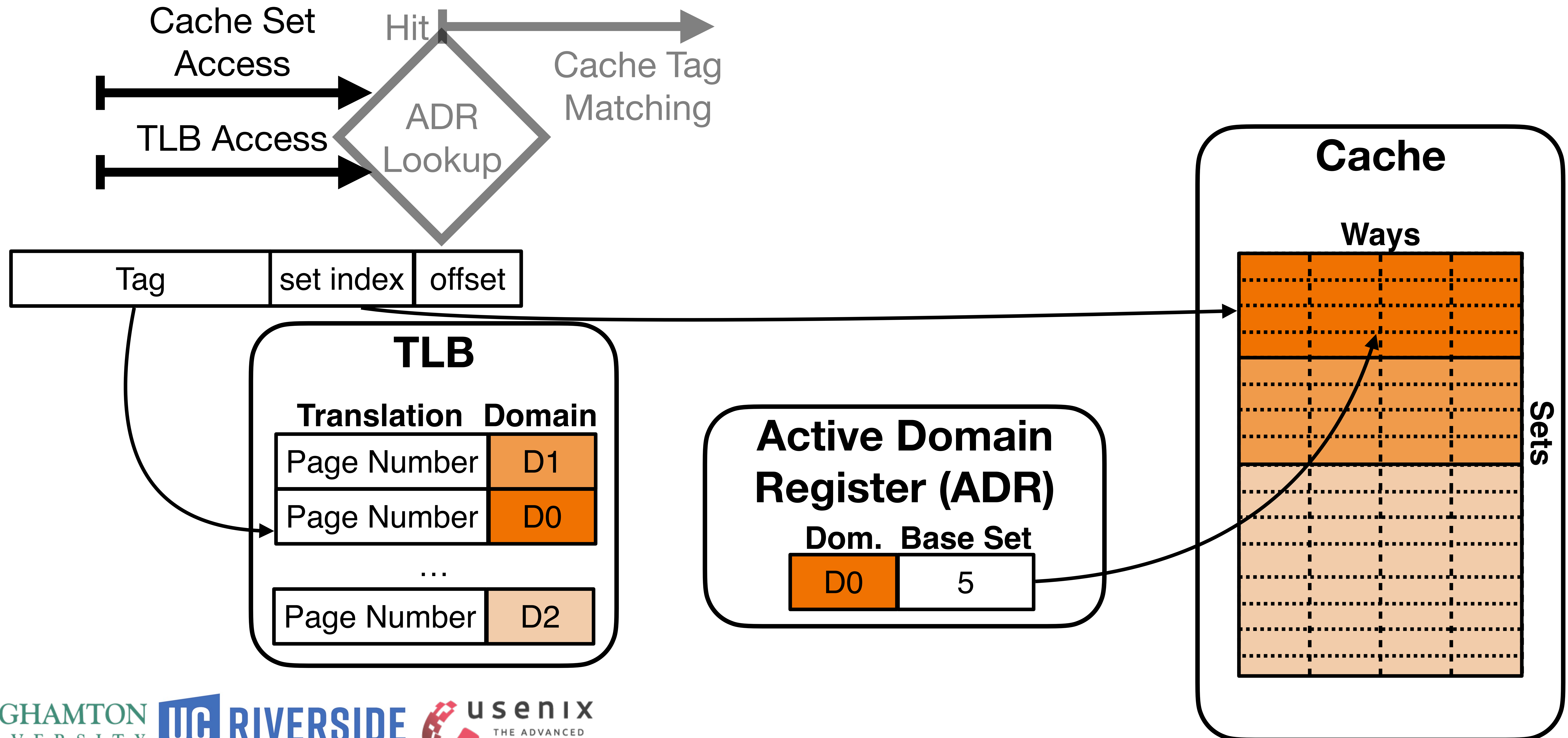


We can essentially "store" latest accessed DRT entry to utilize this domain locality to avoid additional remapping latency in L1 caches.

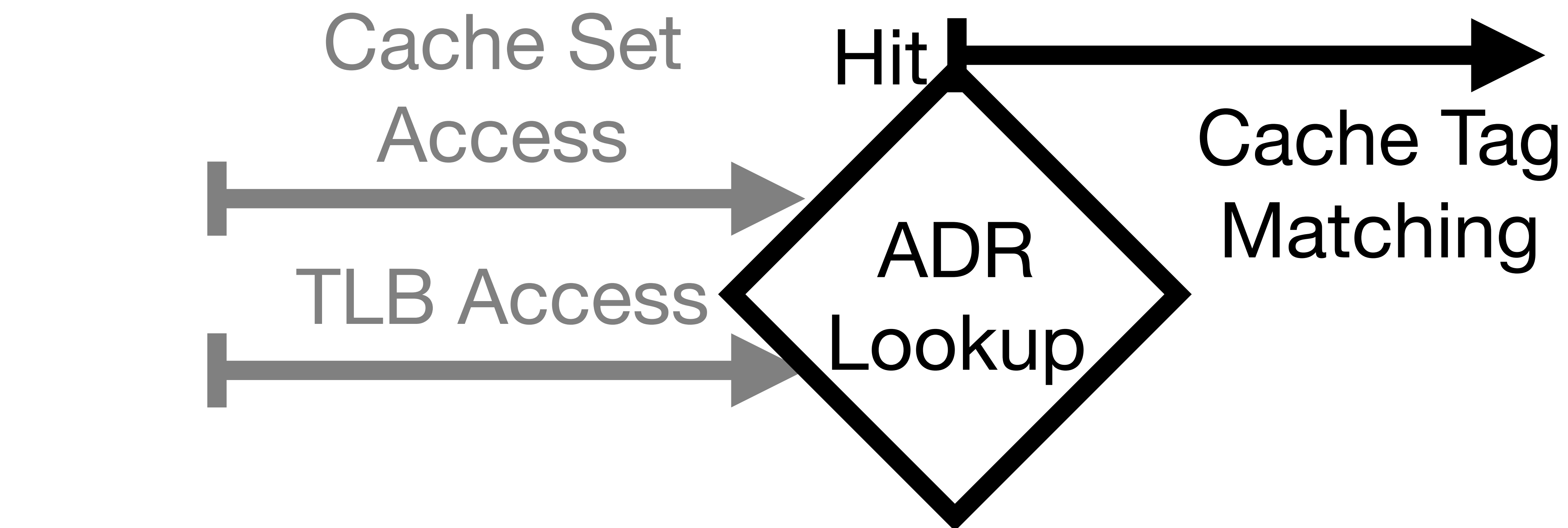
Access to an Optimized Partitioned Cache

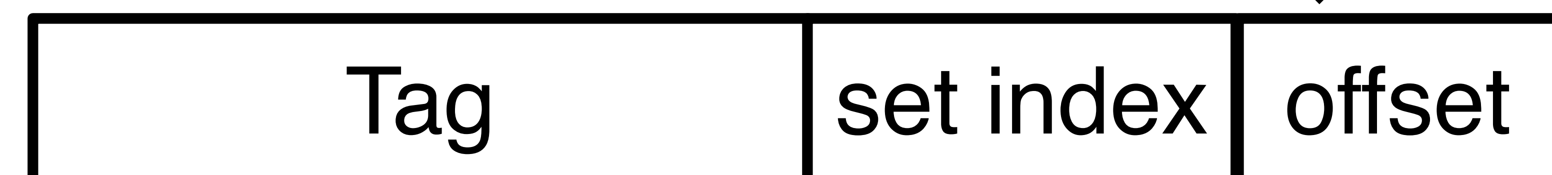


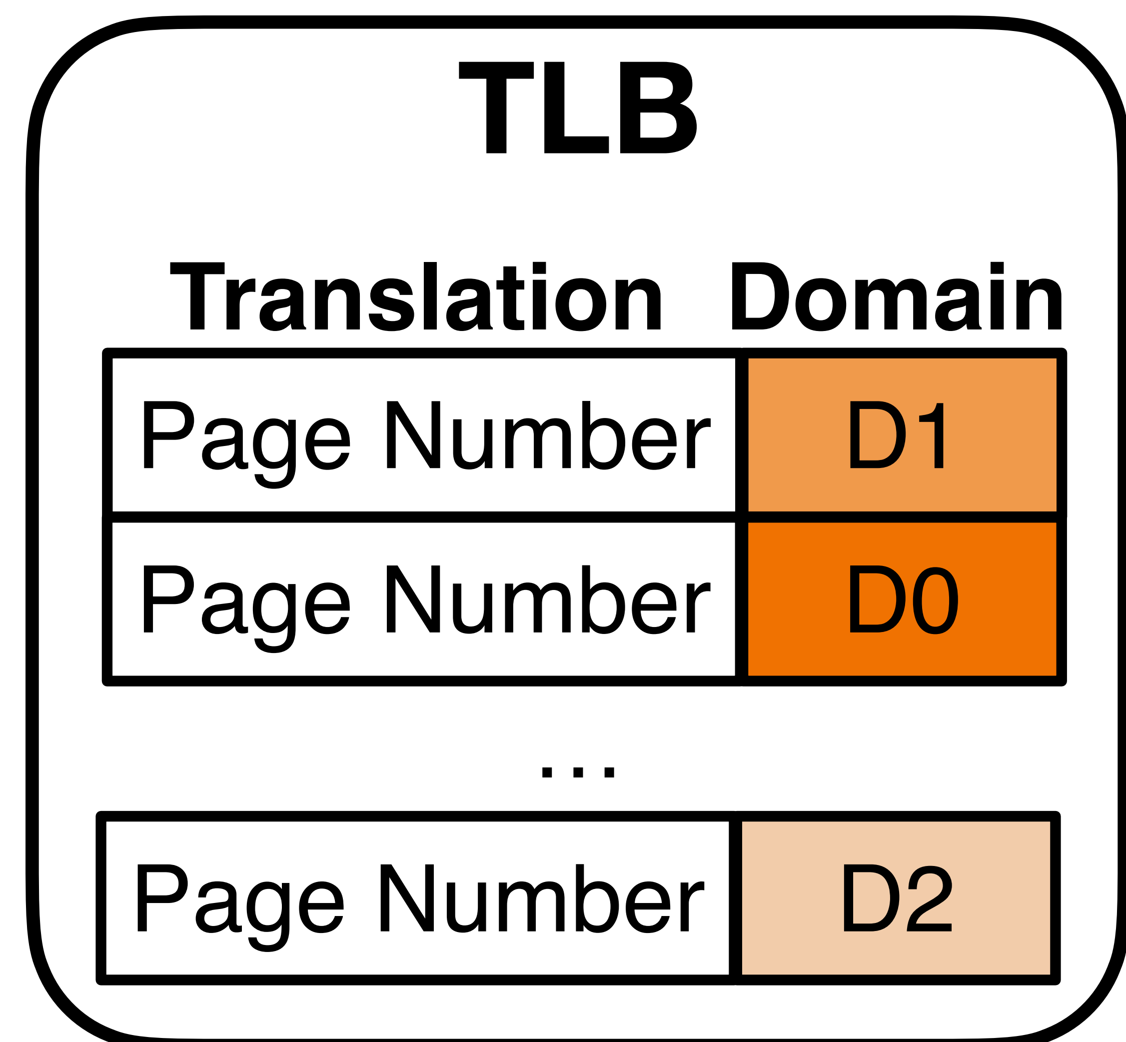
Access to an **Optimized** Partitioned Cache

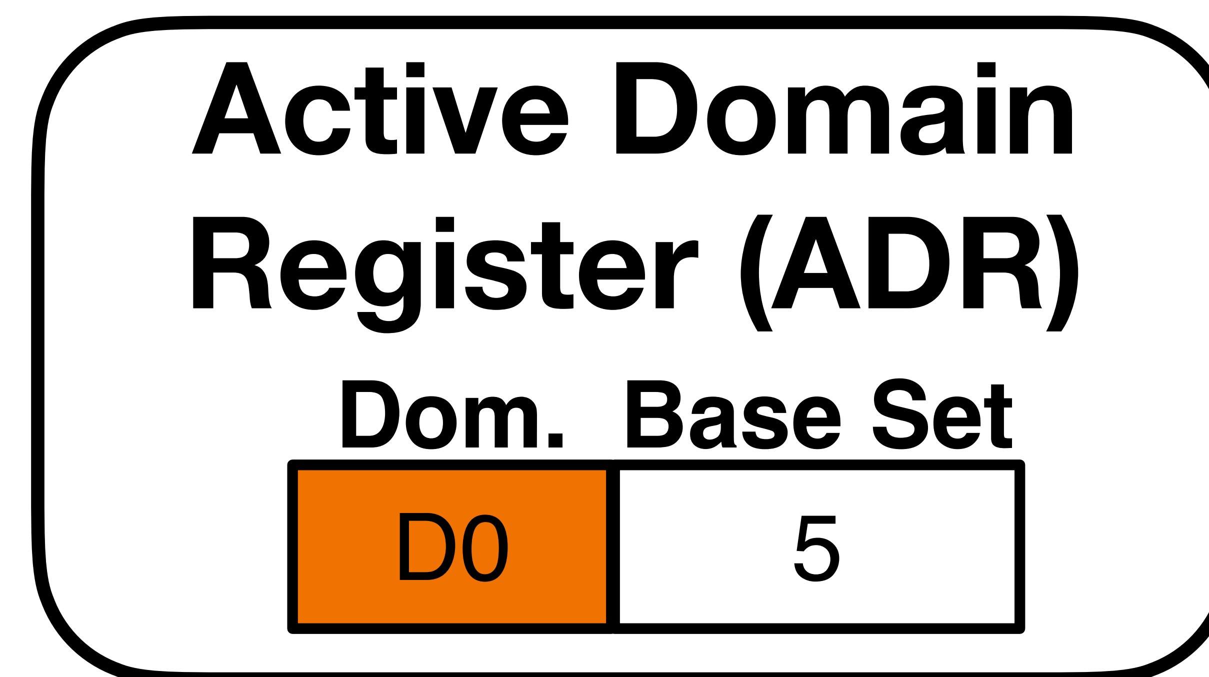


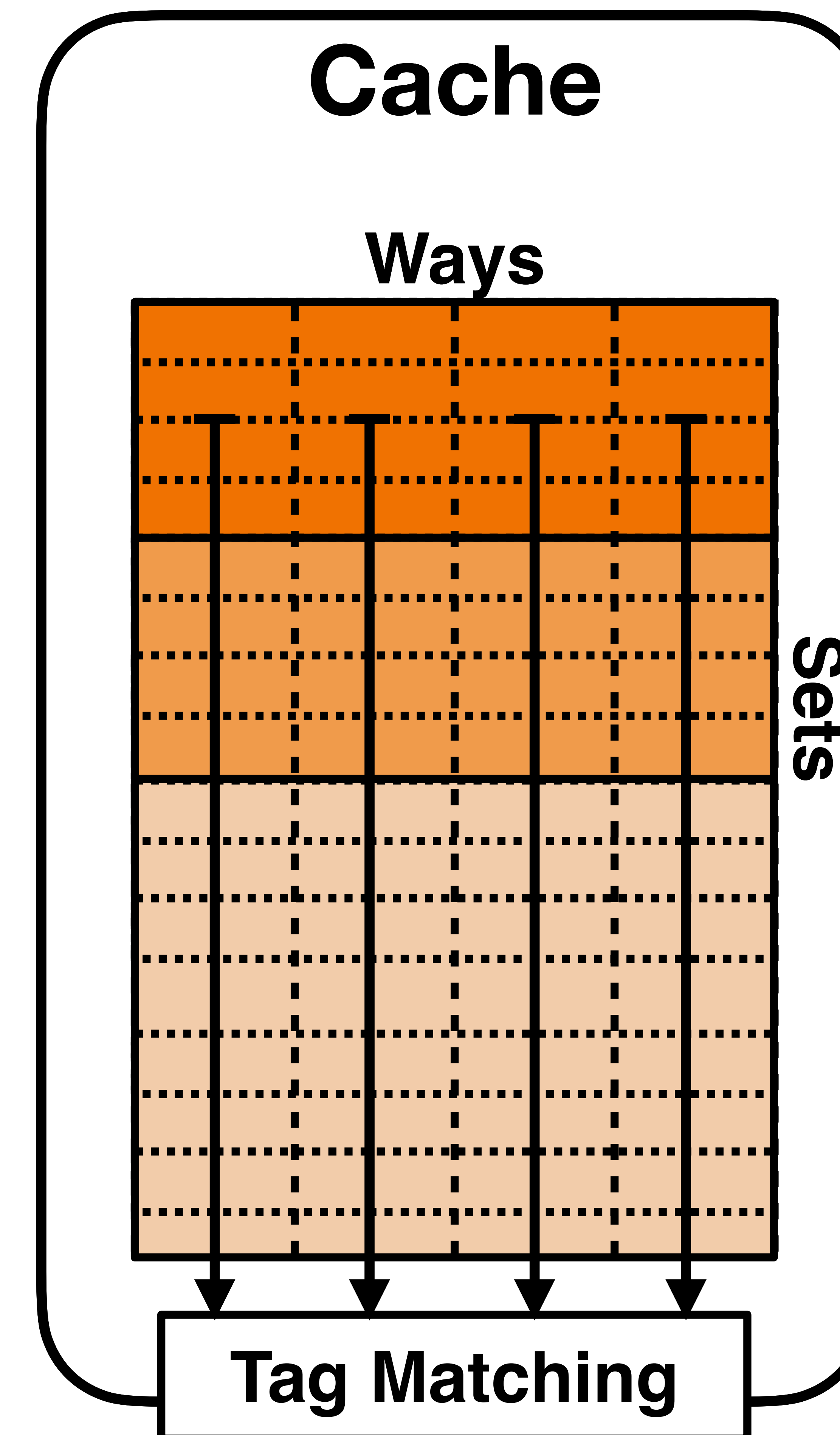
Access to an Optimized Partitioned Cache





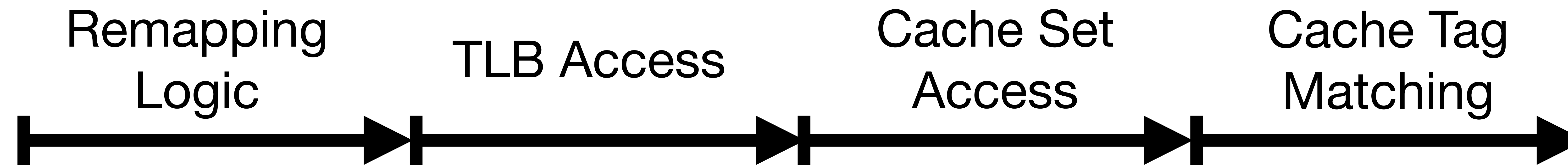




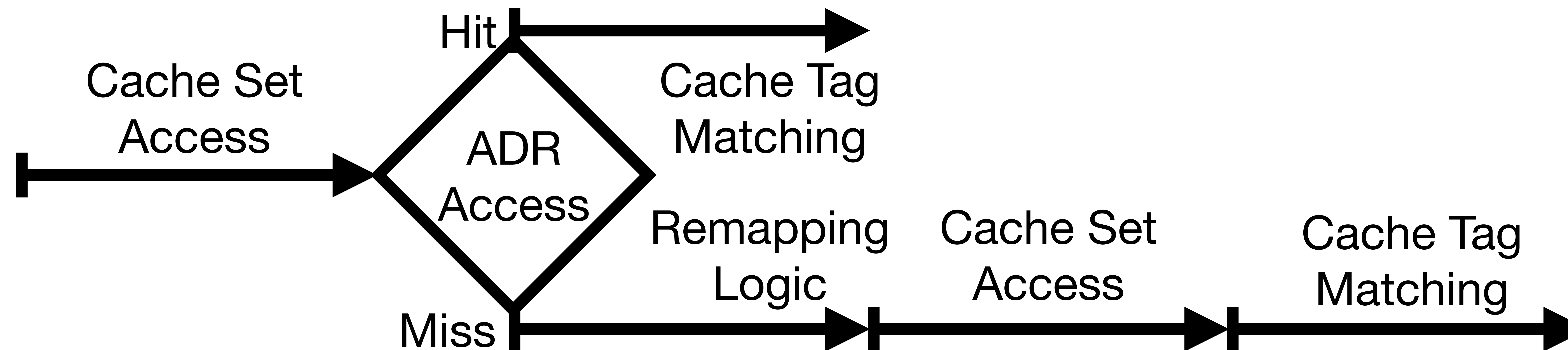


Active Domain Register Summary

- **Unoptimized** Access:



- ADR-**Optimized** Access:

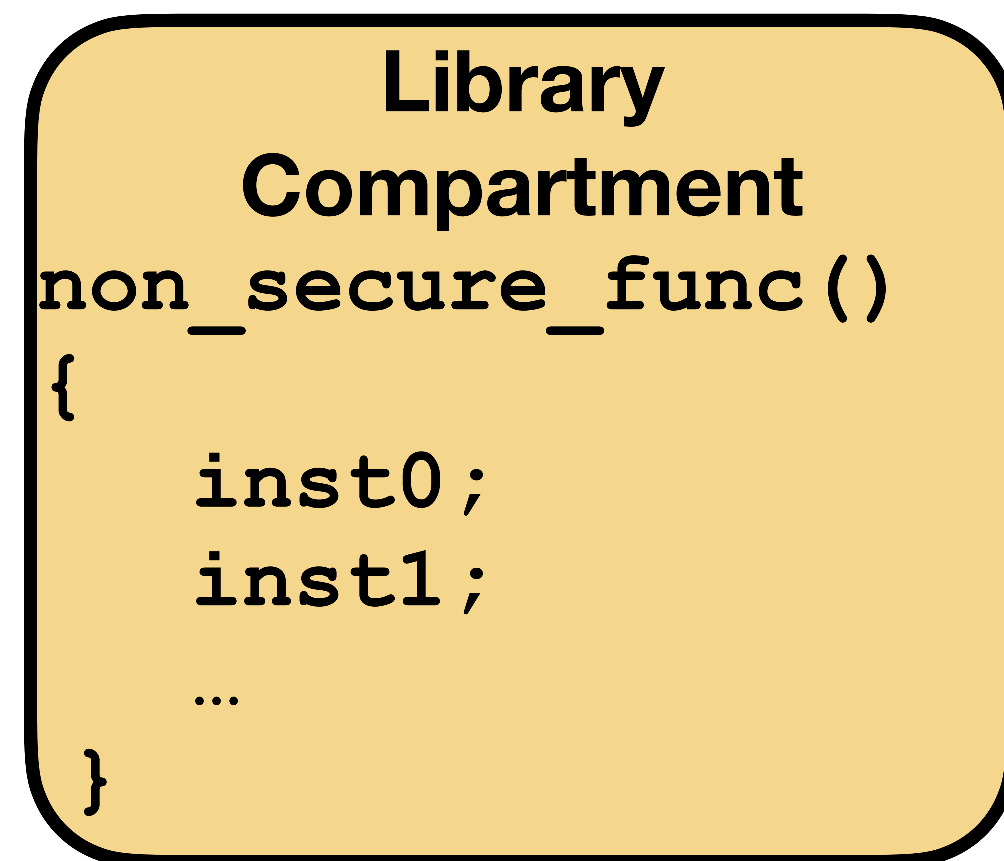


BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Angle-3: Mitigating Shared Library-Based Attacks

Shared Library Side-Channel Attacks

- Accesses to the shared libraries cause leakage:

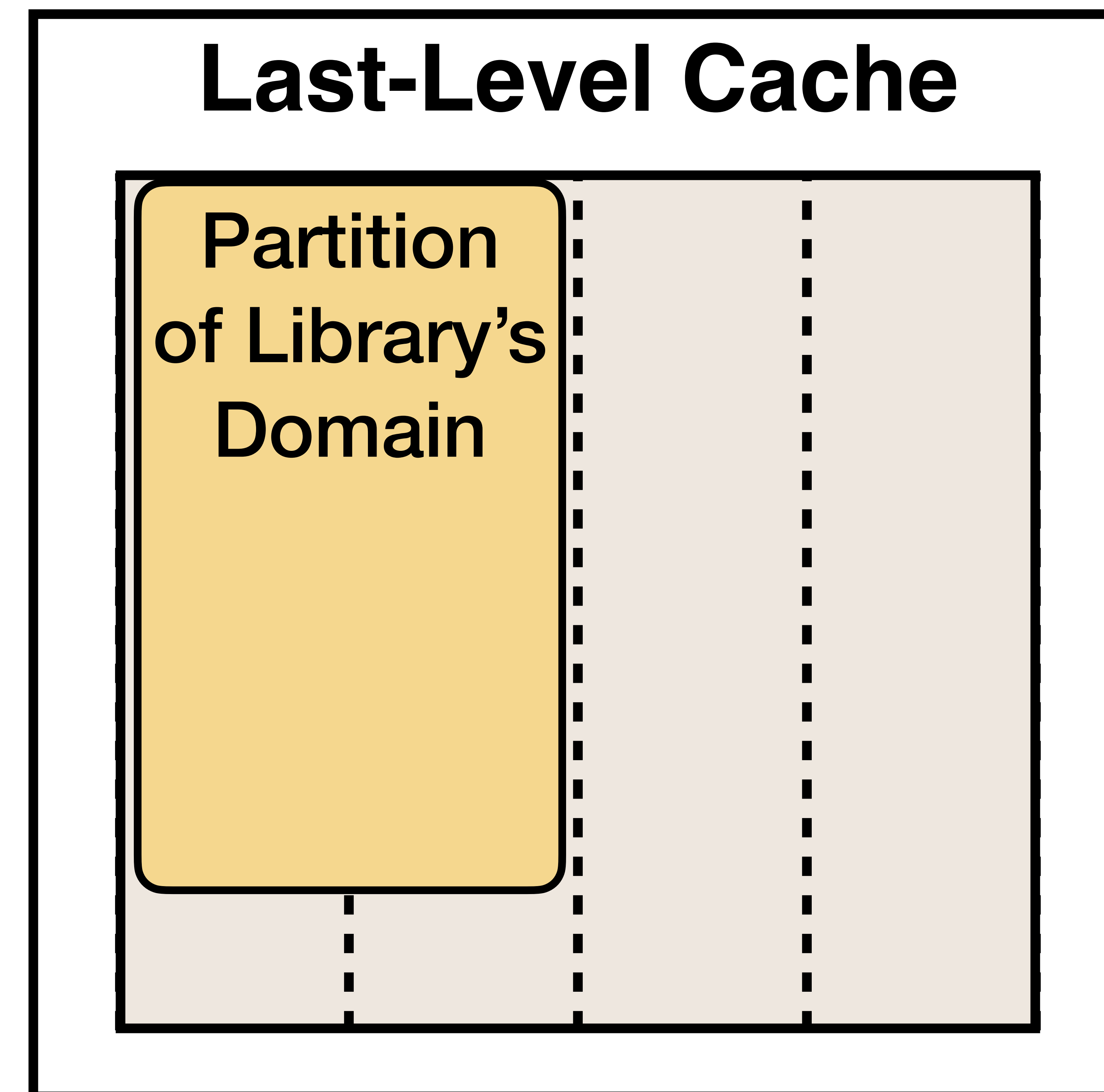


Attacker Compartment

```
clflush &inst0;
clflush &inst1;
…
time1 = rdtscp();
non_secure_func();
time2 = rdtscp();
```

Victim Compartment

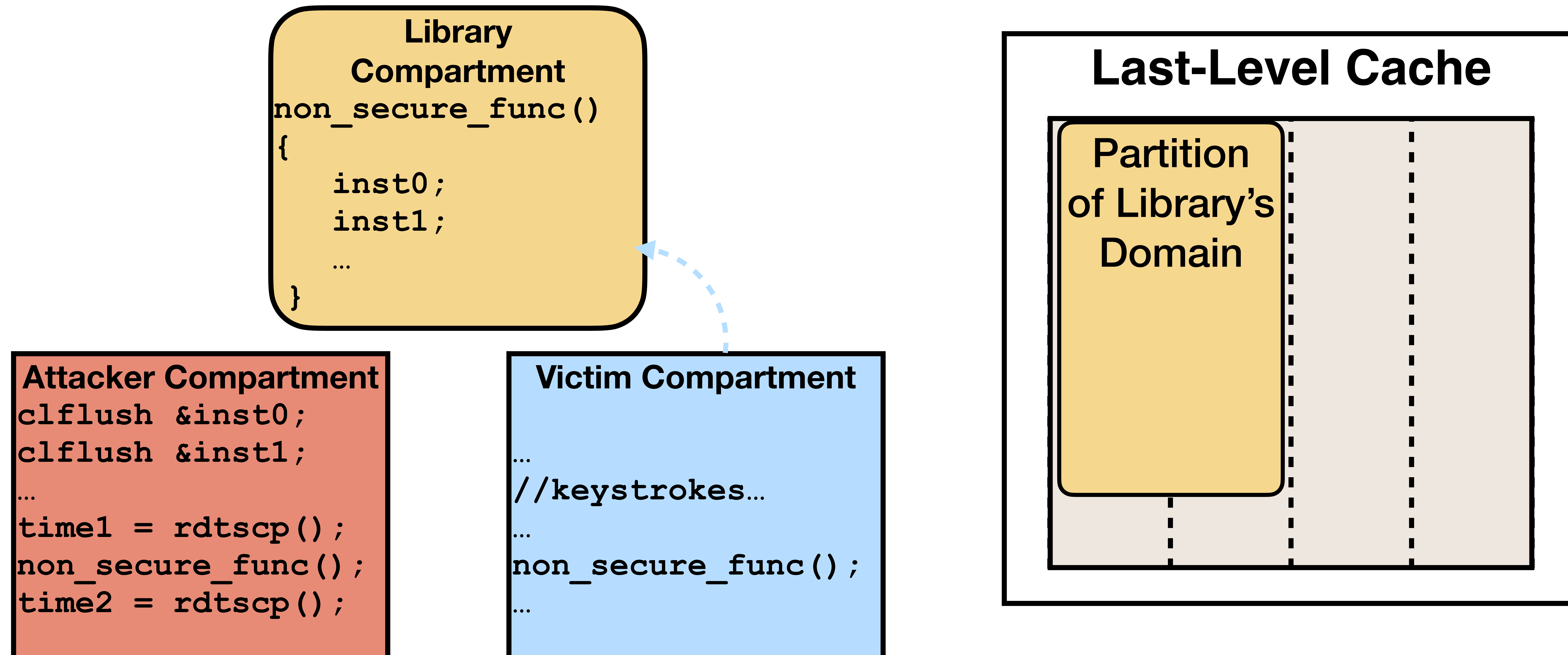
```
…
//keystrokes…
…
non_secure_func();
…
```



BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Shared Library Side-Channel Attacks

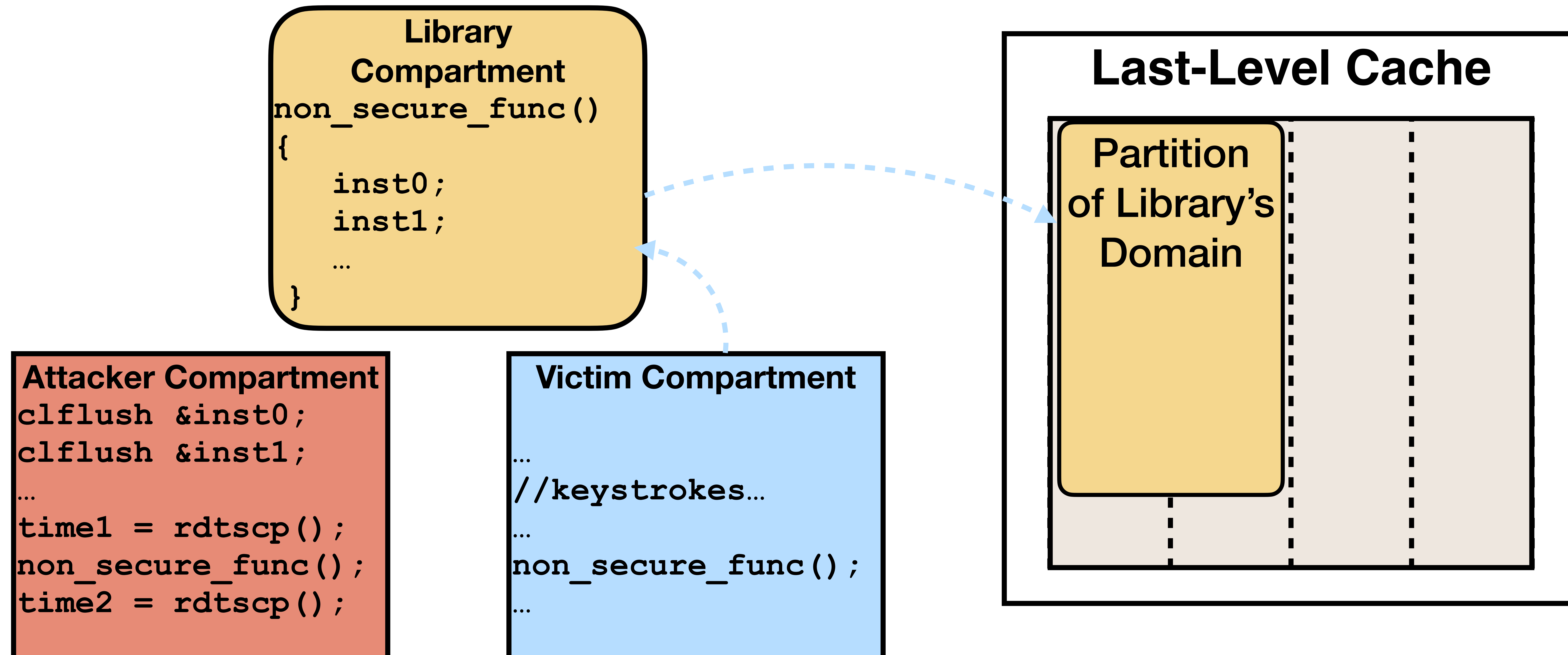
- Accesses to the shared libraries cause leakage:



BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK UC RIVERSIDE usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Shared Library Side-Channel Attacks

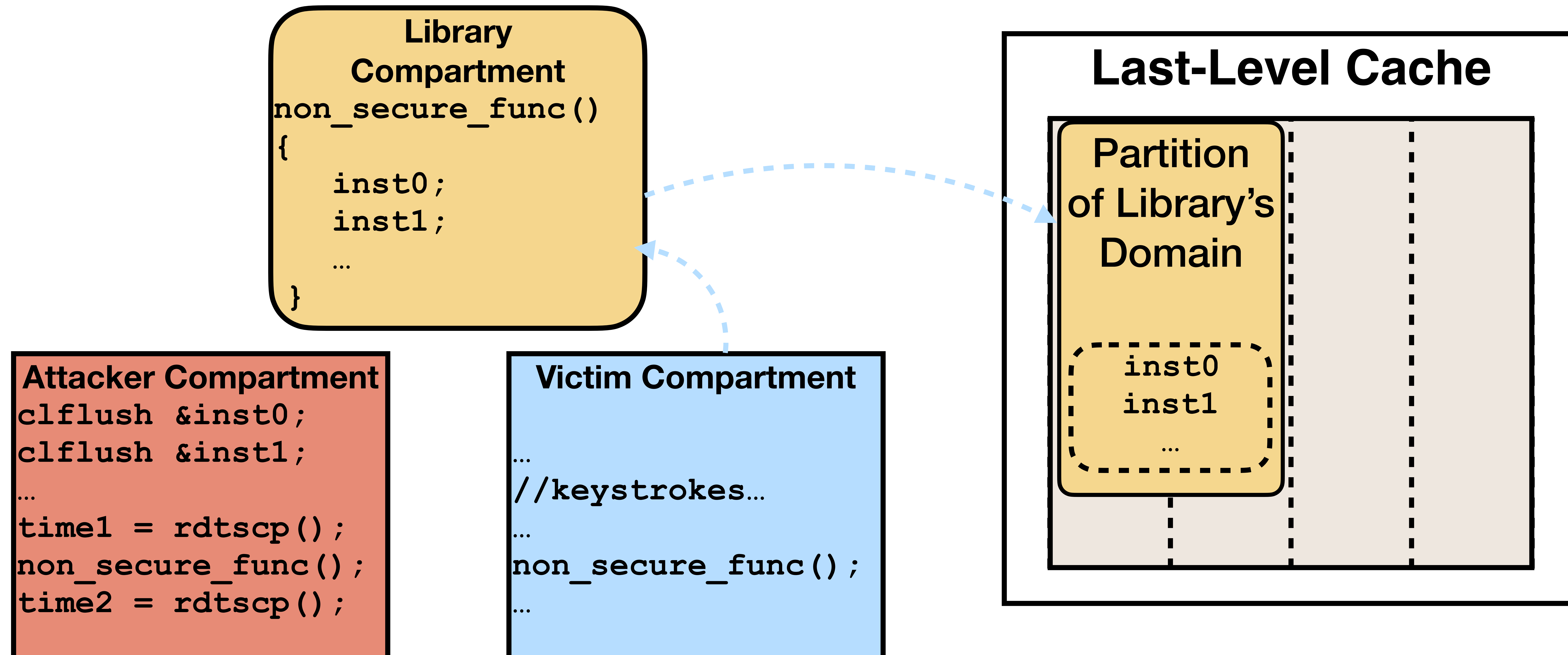
- Accesses to the shared libraries cause leakage:



BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

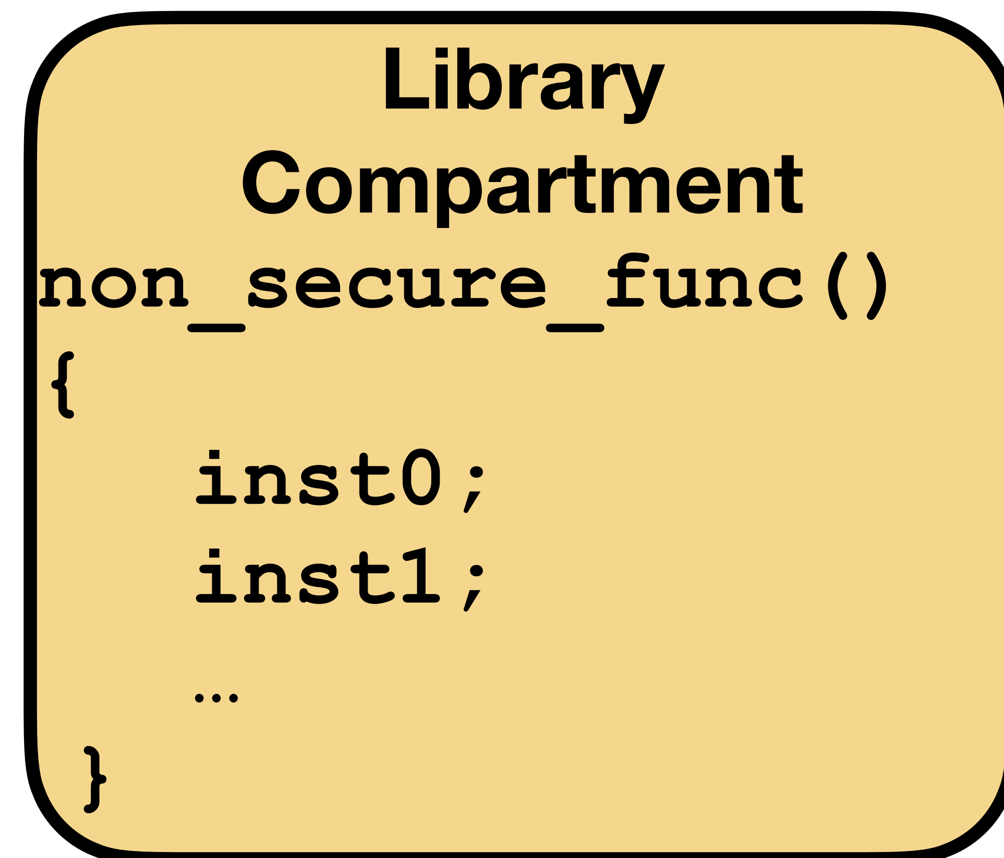
Shared Library Side-Channel Attacks

- Accesses to the shared libraries cause leakage:



Shared Library Side-Channel Attacks

- Accesses to the shared libraries cause leakage:

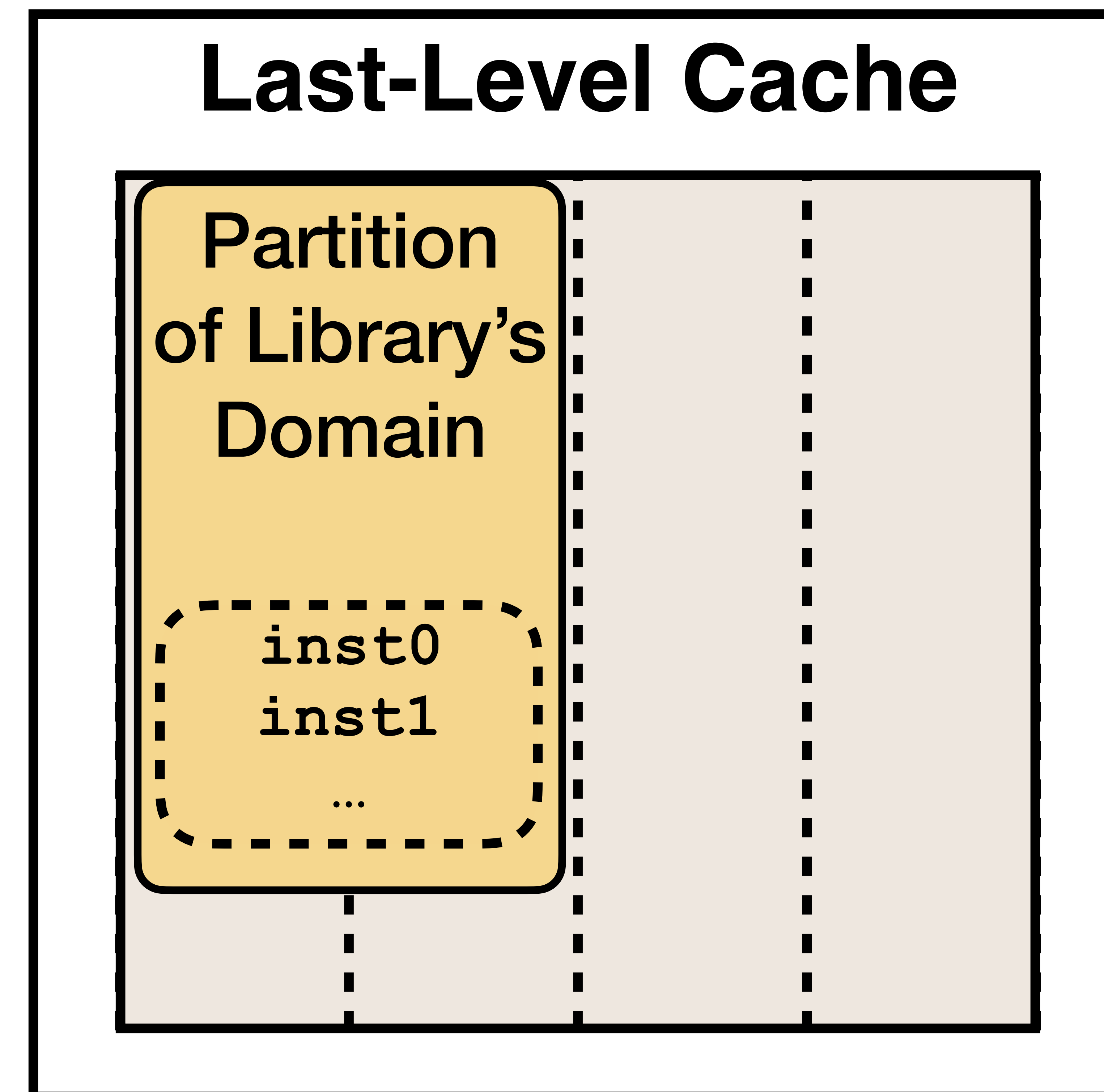


Attacker Compartment

```
clflush &inst0;
clflush &inst1;
…
time1 = rdtscp();
non_secure_func();
time2 = rdtscp();
```

Victim Compartment

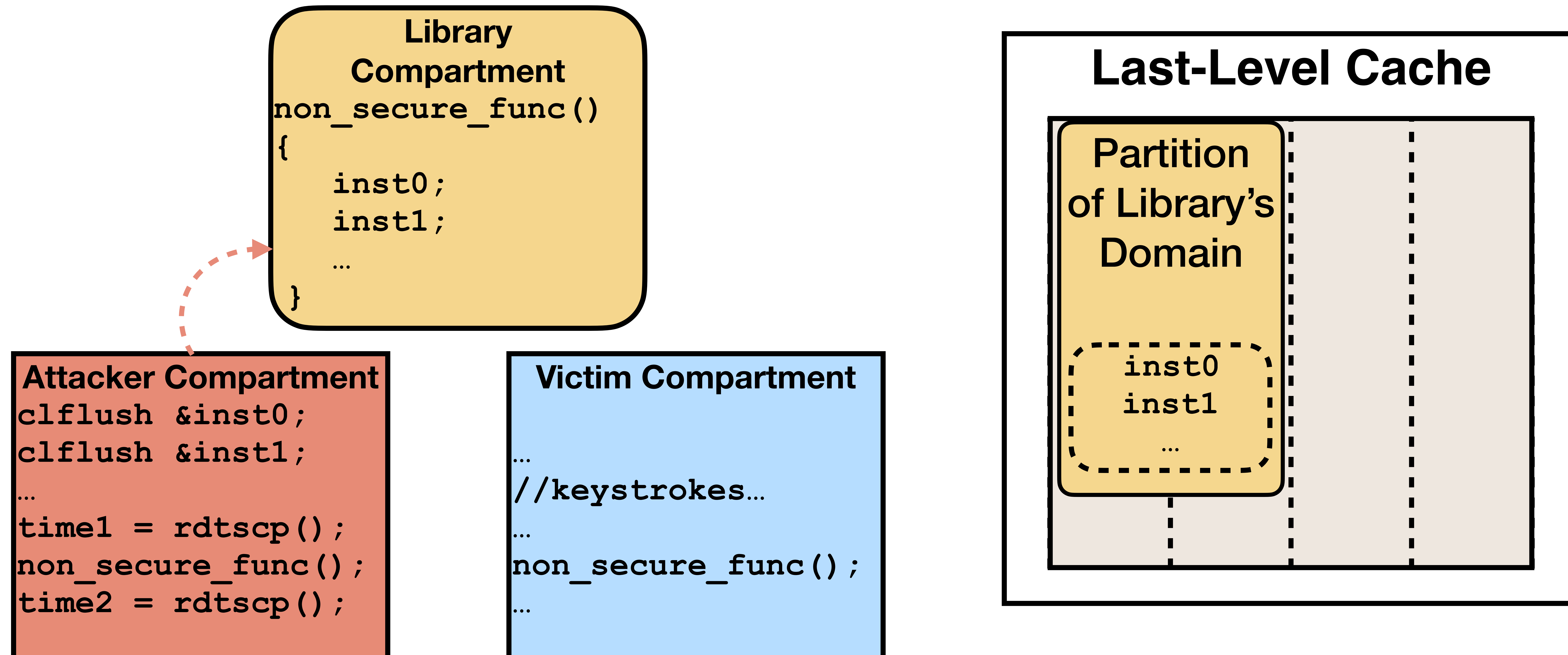
```
…
//keystrokes…
…
non_secure_func();
…
```



BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

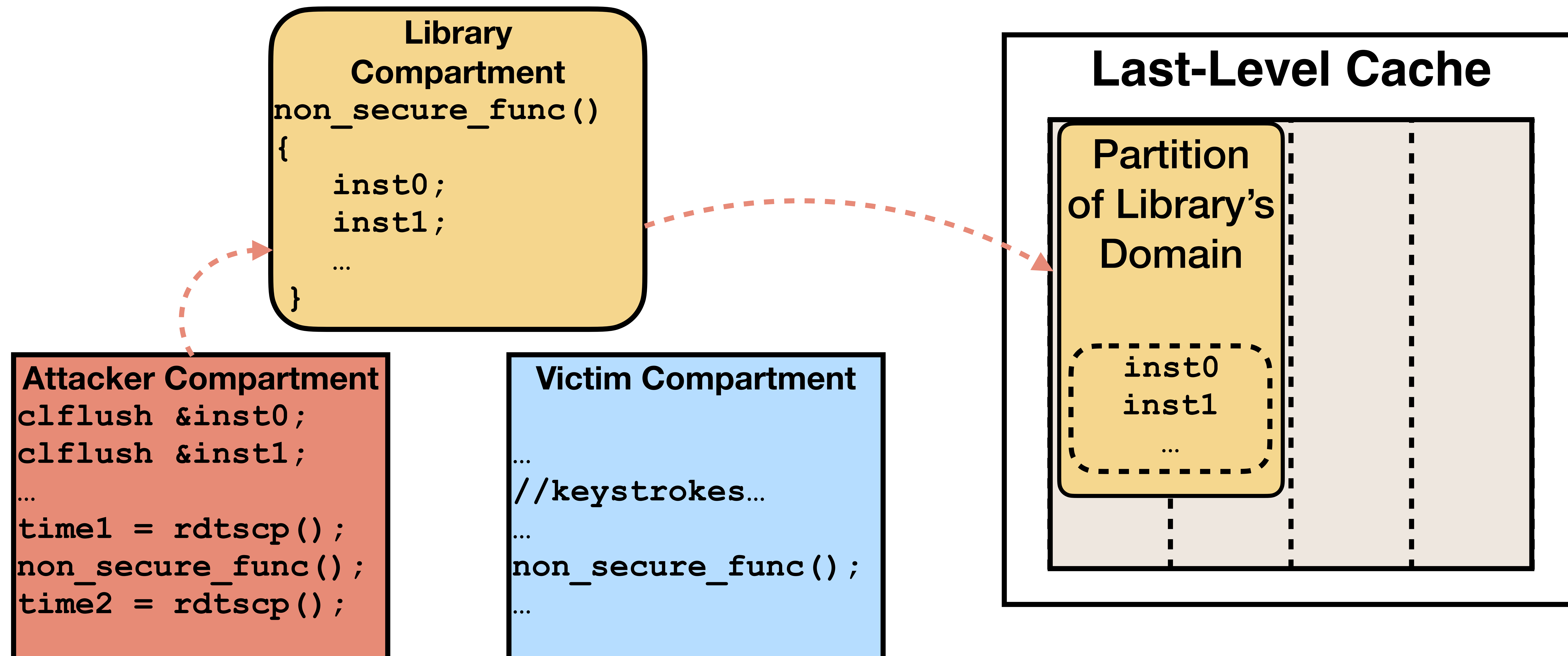
Shared Library Side-Channel Attacks

- Accesses to the shared libraries cause leakage:



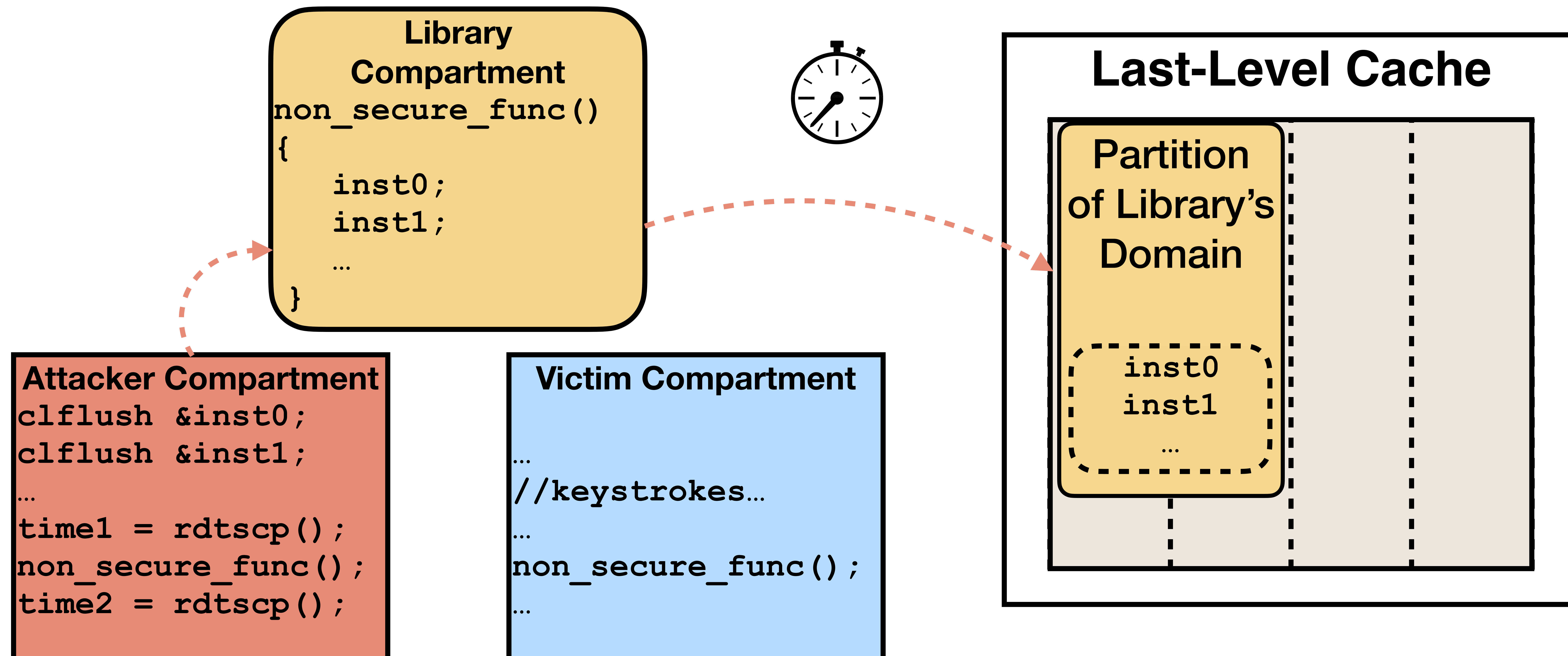
Shared Library Side-Channel Attacks

- Accesses to the shared libraries cause leakage:



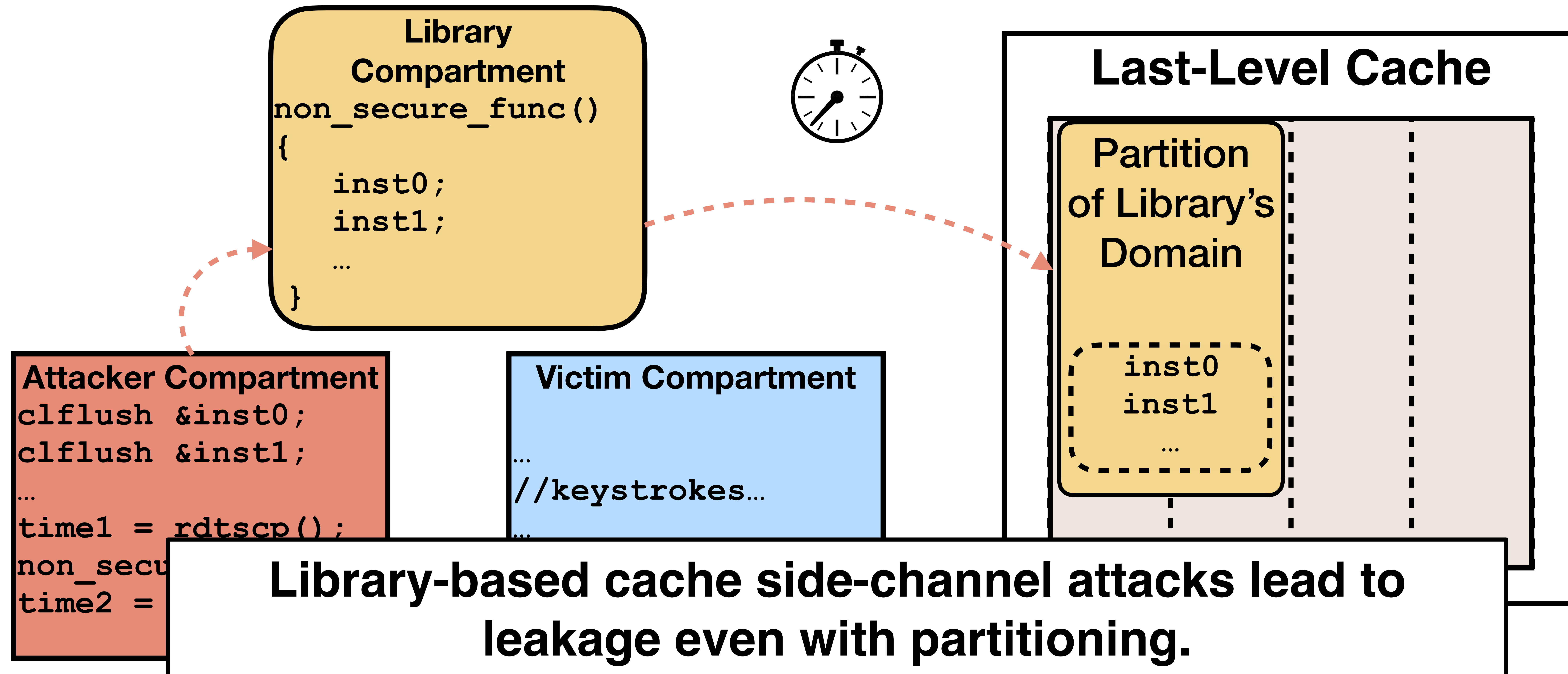
Shared Library Side-Channel Attacks

- Accesses to the shared libraries cause leakage:



Shared Library Side-Channel Attacks

- Accesses to the shared libraries cause leakage:



BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

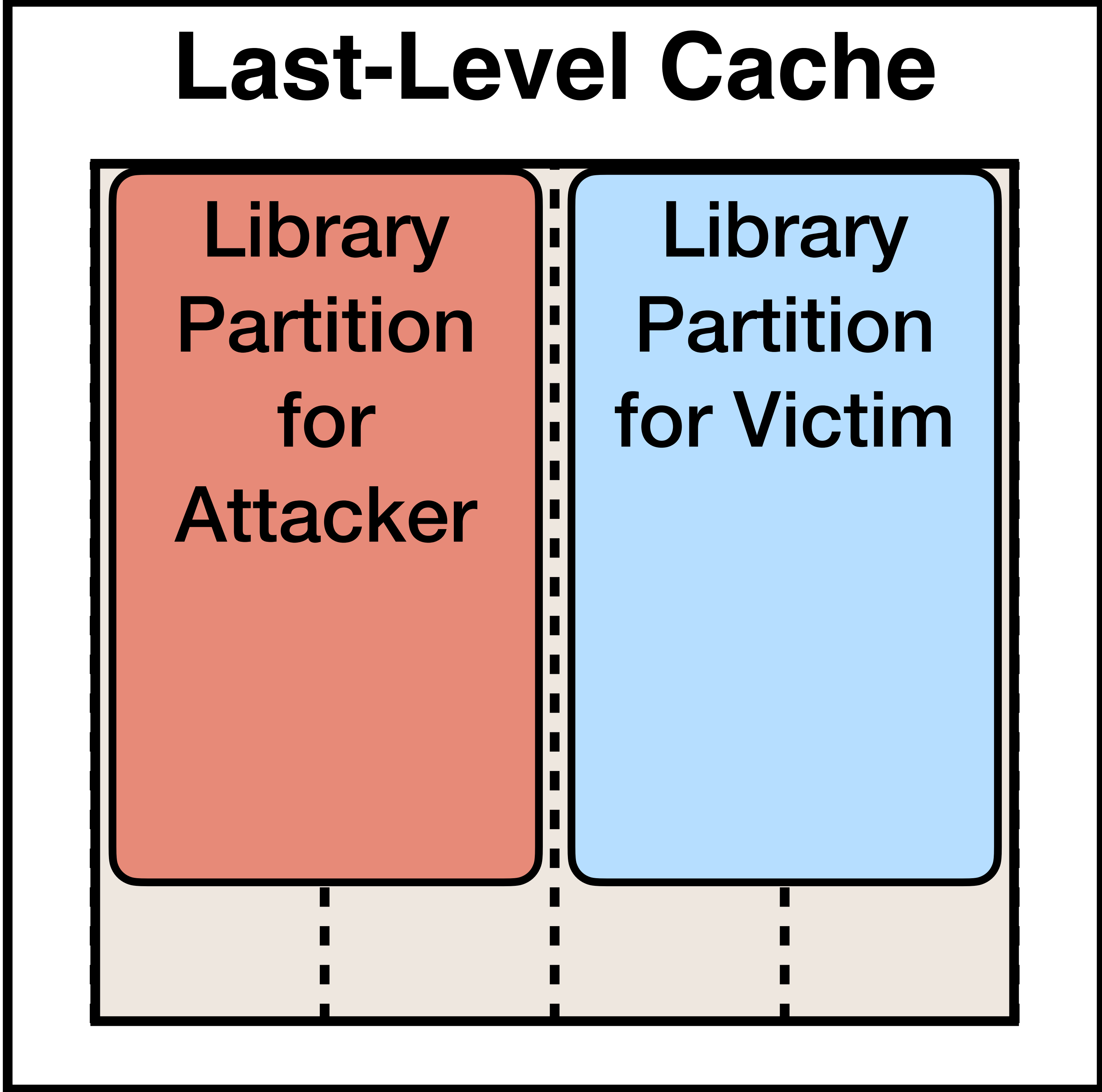
Horizontal Partitioning

- Accesses to the shared libraries cause leakage:

Library Compartment
```
non_secure_func()
{
    inst0;
    inst1;
    …
}
```

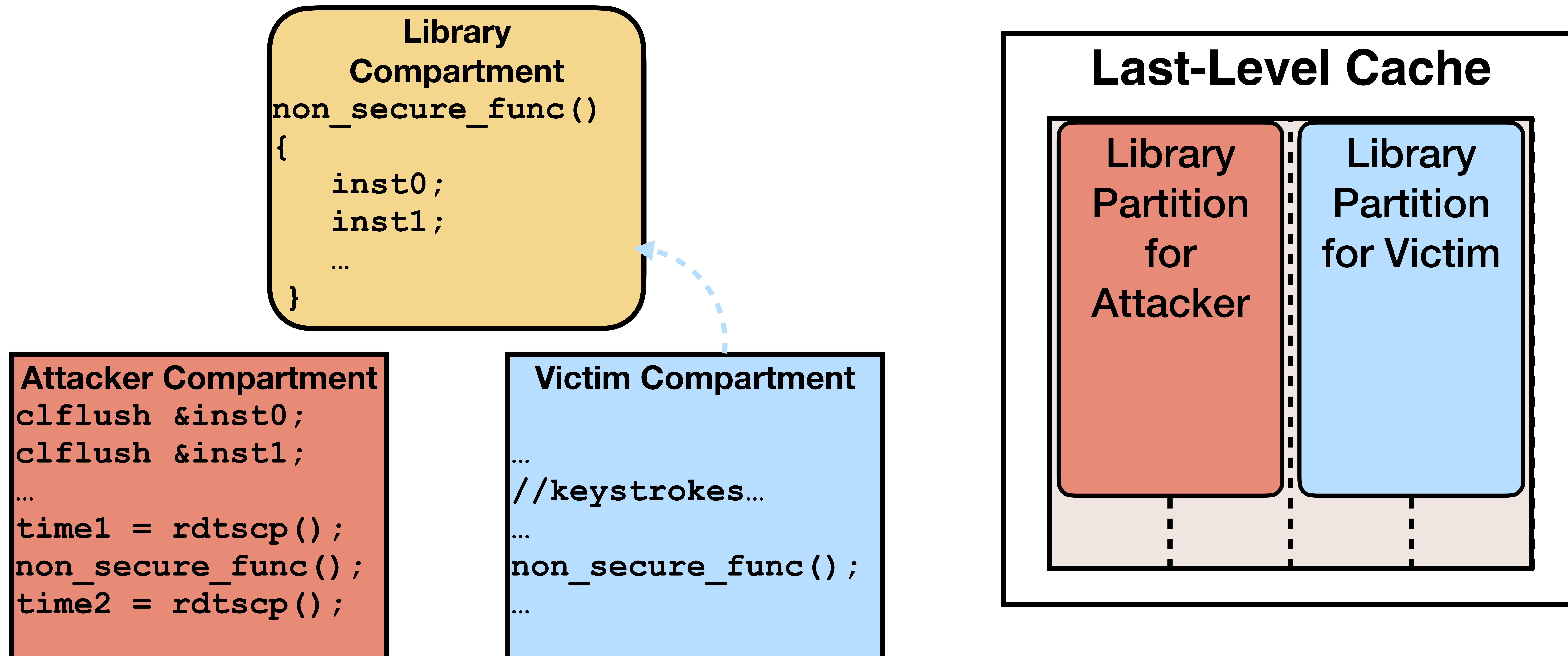
Attacker Compartment
```
clflush &inst0;
clflush &inst1;
…
time1 = rdtscp();
non_secure_func();
time2 = rdtscp();
```

Victim Compartment
```
…
//keystrokes…
…
non_secure_func();
…
```



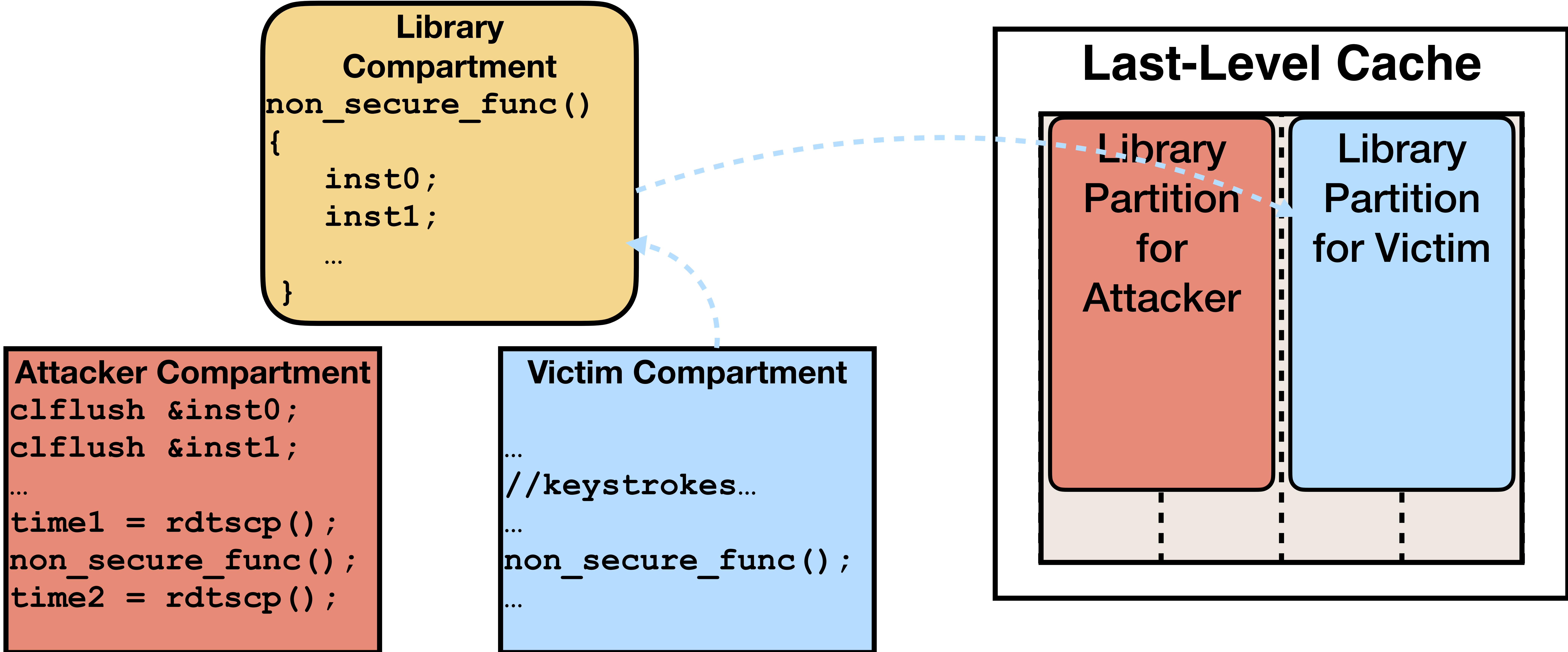
Horizontal Partitioning

- Accesses to the shared libraries cause leakage:



Horizontal Partitioning

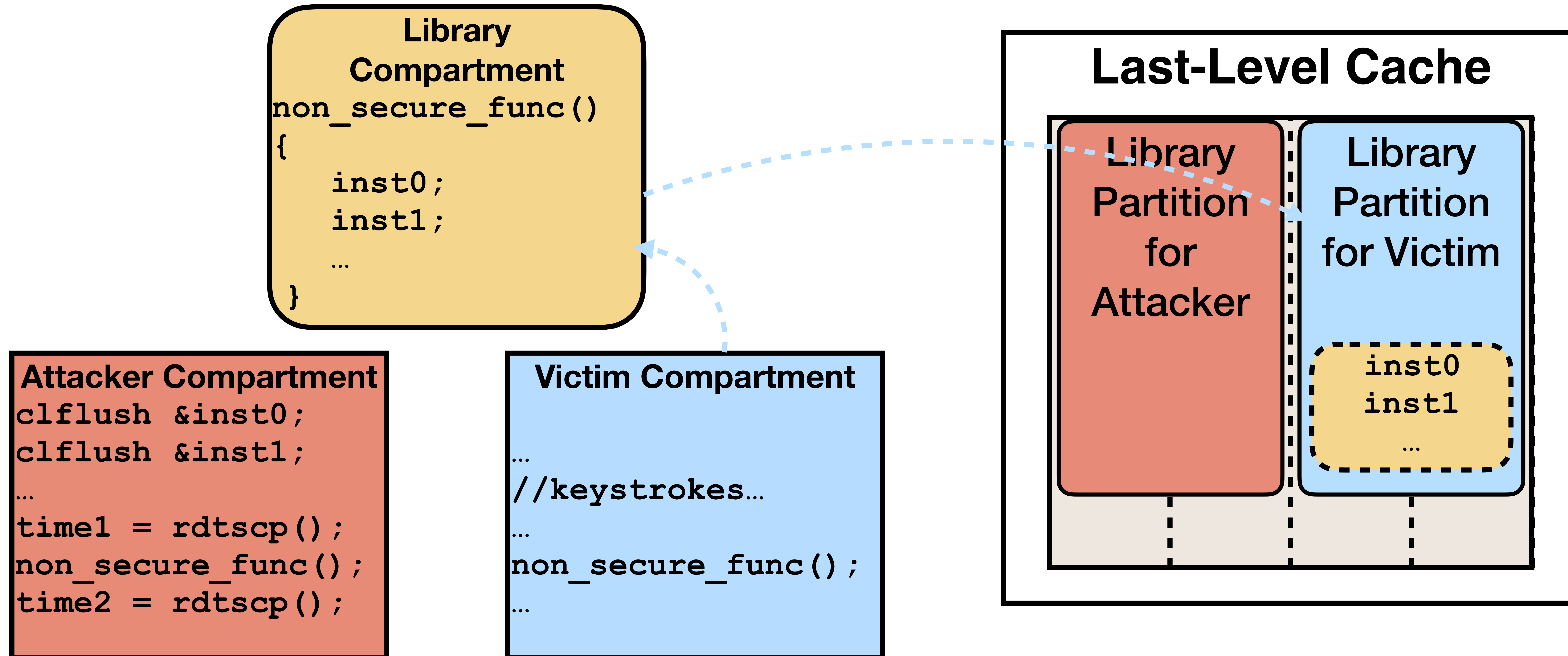
- Accesses to the shared libraries cause leakage:



BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Horizontal Partitioning

- Accesses to the shared libraries cause leakage:



Horizontal Partitioning

- Accesses to the shared libraries cause leakage:

Library Compartment

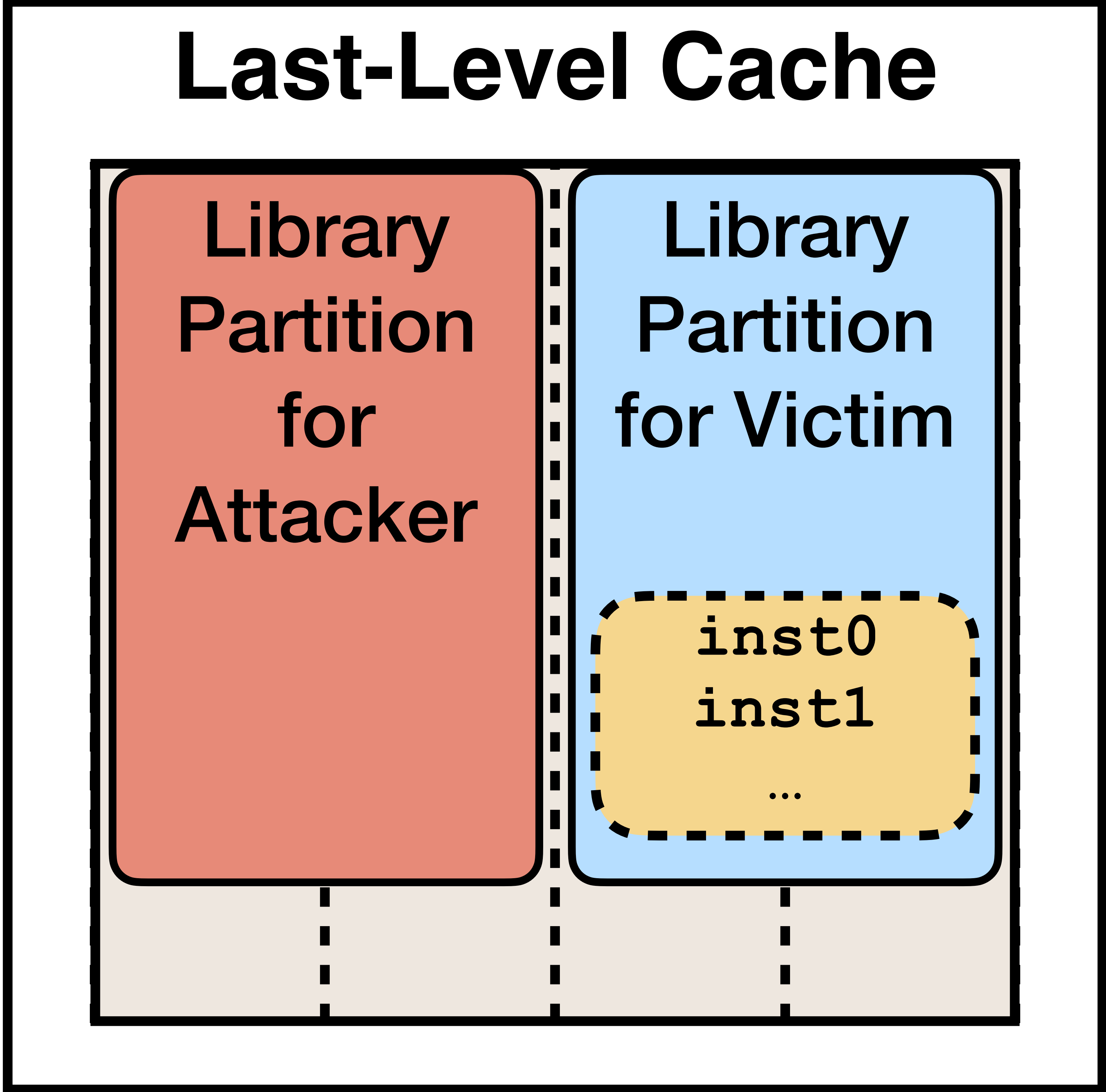
```
non_secure_func()
{
    inst0;
    inst1;
    …
}
```

Attacker Compartment

```
clflush &inst0;
clflush &inst1;
…
time1 = rdtscp();
non_secure_func();
time2 = rdtscp();
```

Victim Compartment

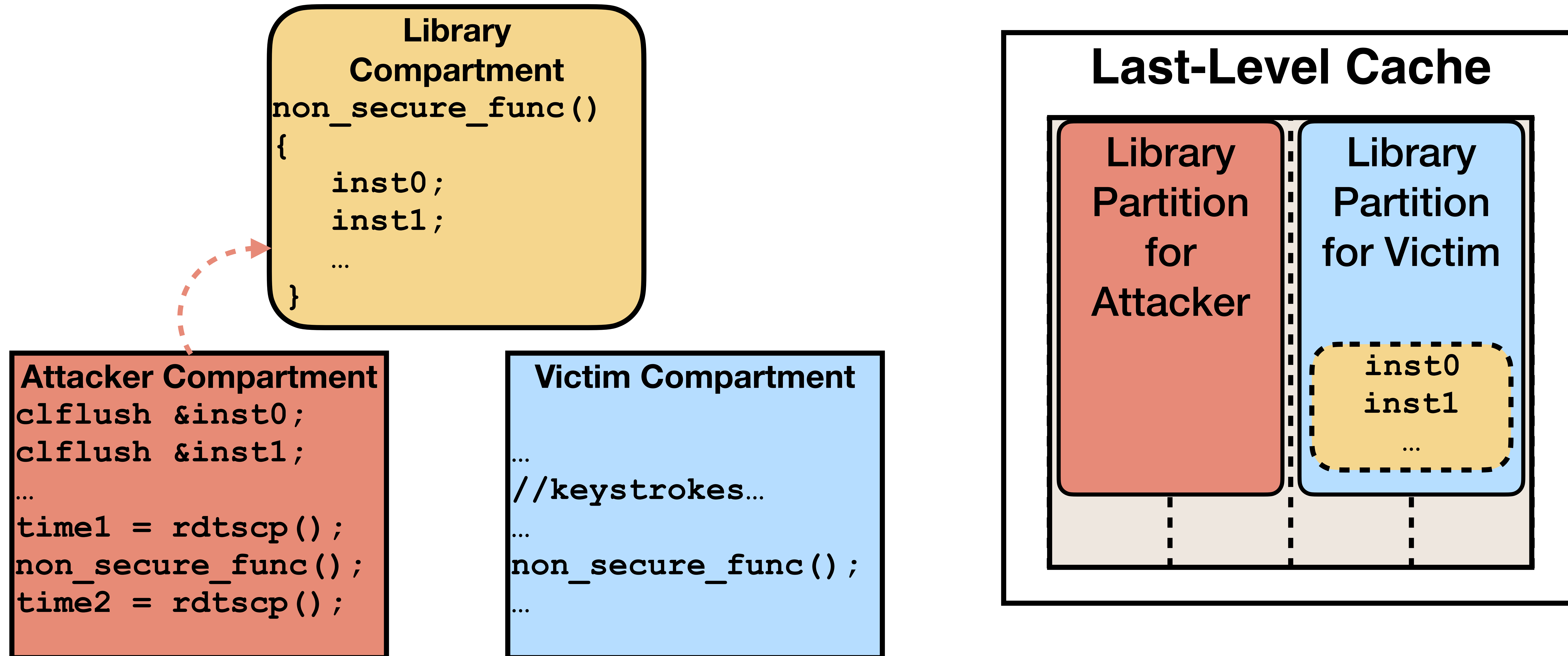
```
…
//keystrokes…
…
non_secure_func();
…
```



Binghamton University State University of New York
UC Riverside
usenix The Advanced Computing Systems Association

Horizontal Partitioning

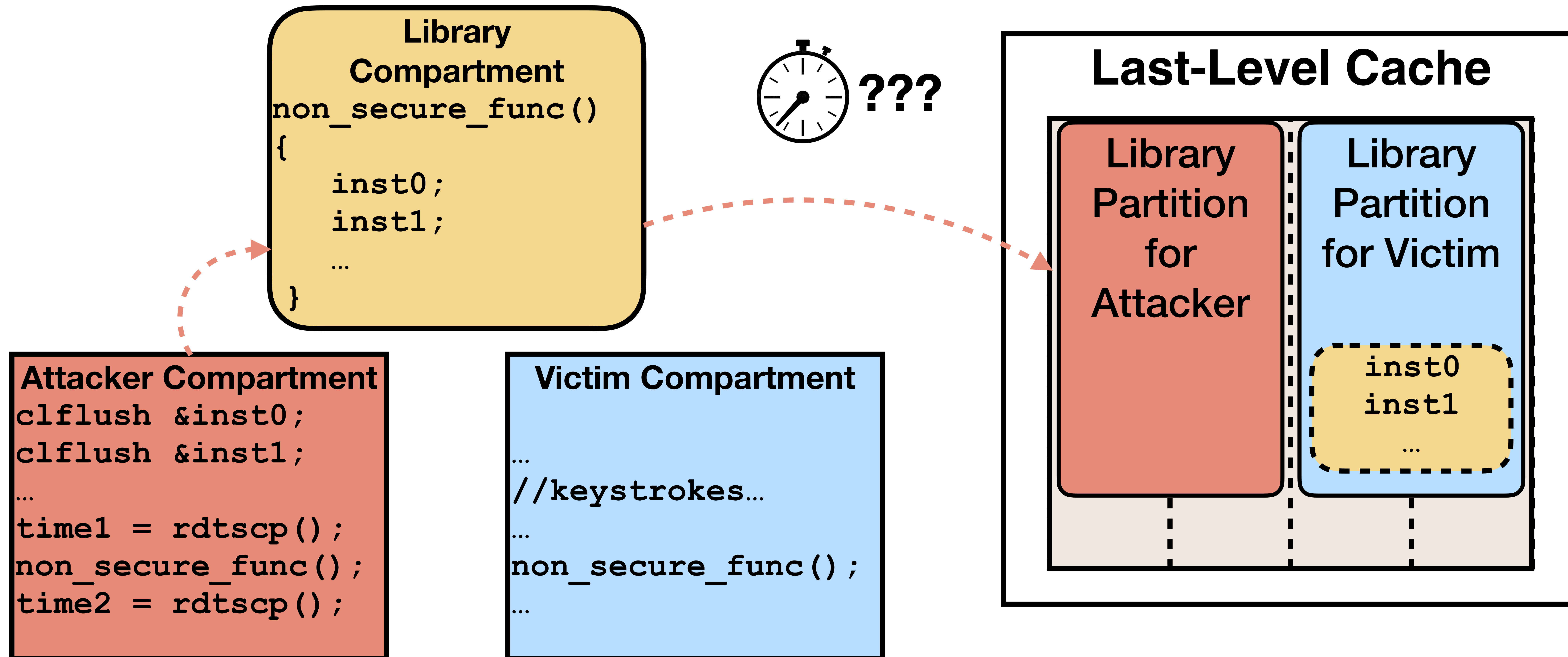
- Accesses to the shared libraries cause leakage:



BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK UC RIVERSIDE usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

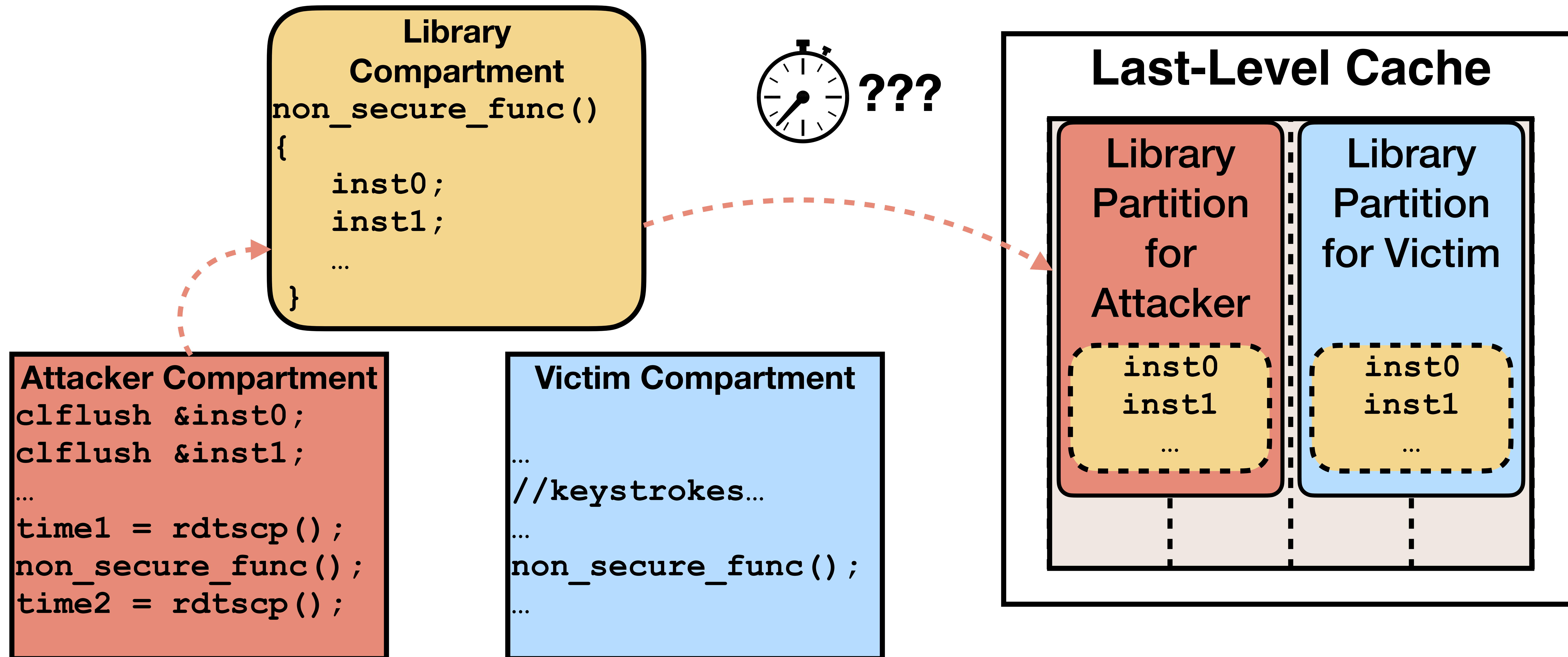
Horizontal Partitioning

- Accesses to the shared libraries cause leakage:



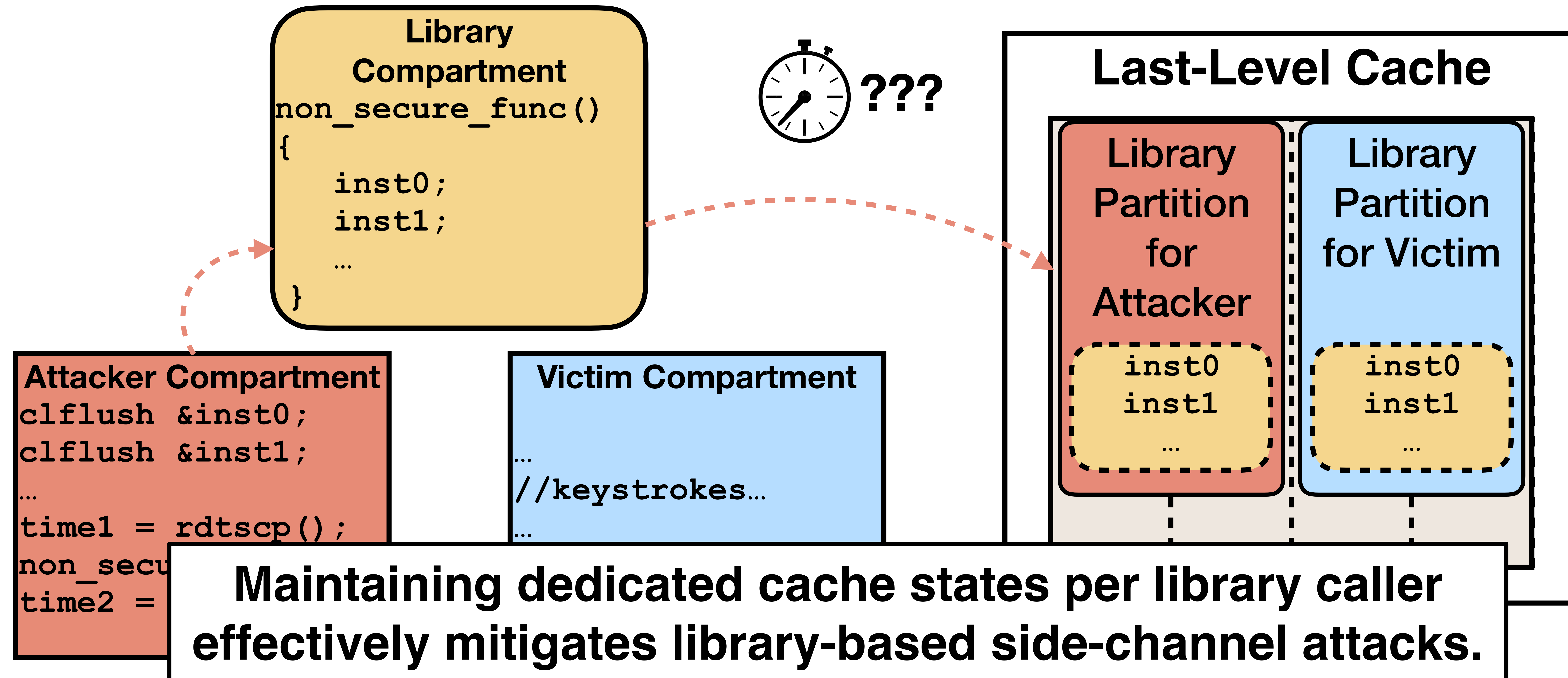
Horizontal Partitioning

- Accesses to the shared libraries cause leakage:



Horizontal Partitioning

- Accesses to the shared libraries cause leakage:



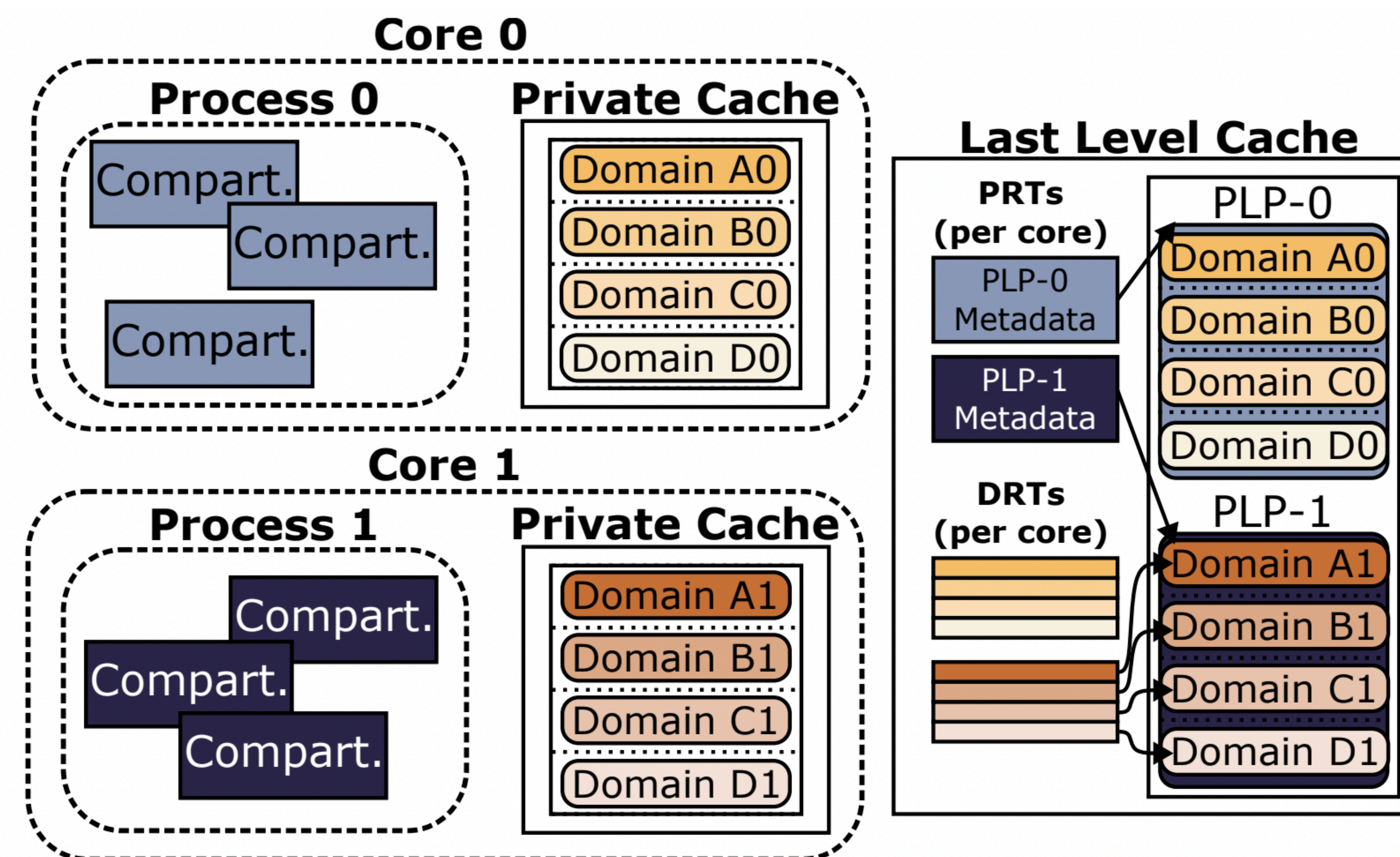
BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Extending **SCC** to Multi-Process Environments

BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

SCC's Multi-Process Setup

- **PRT:** Process Remapping Table
- **PLP:** Process-Level Partition

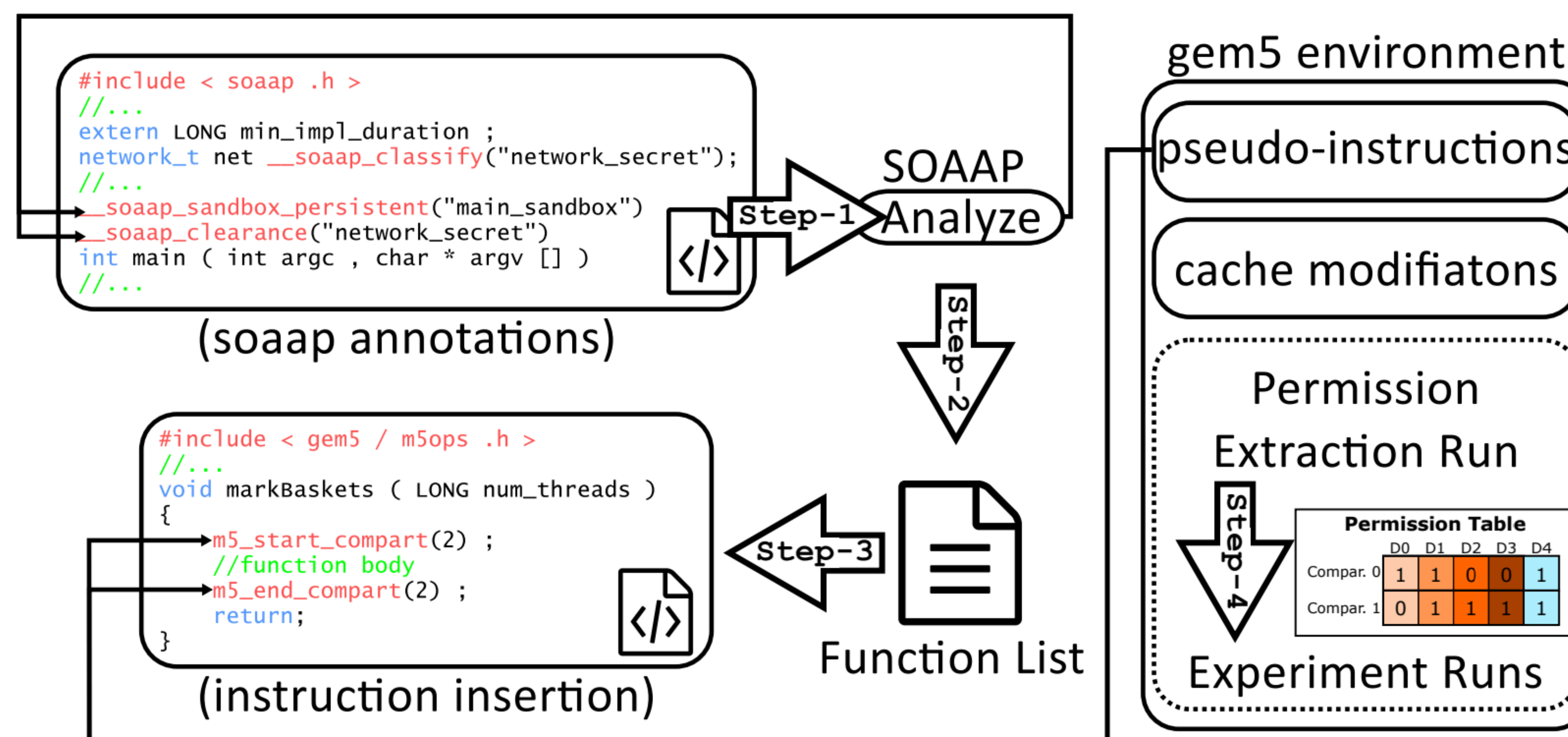


Binghamton University State University of New York
UC Riverside
usenix The Advanced Computing Systems Association

Methodology and Evaluation

Implementation Details

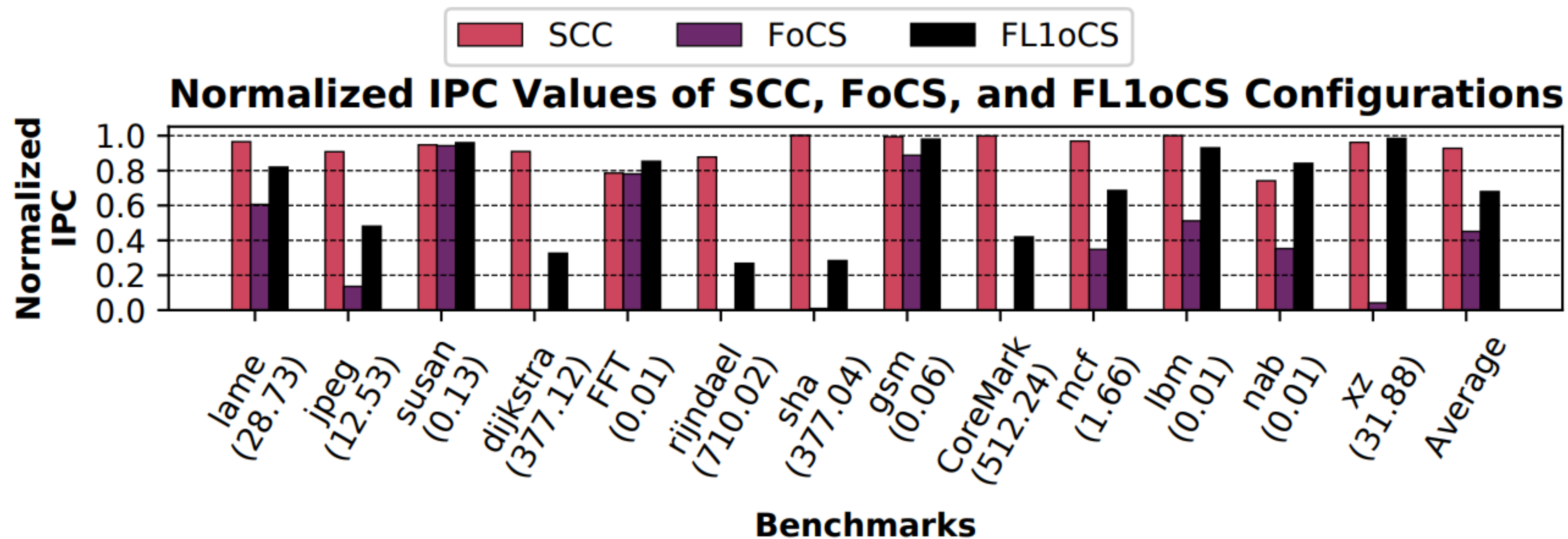
- Implemented in cycle-accurate **gem5 simulator**.
- Used **SOAAP** tool to generate compartments for MiBench and SPEC17 benchmarks.



BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Performance Comparison Against Cache Flushing

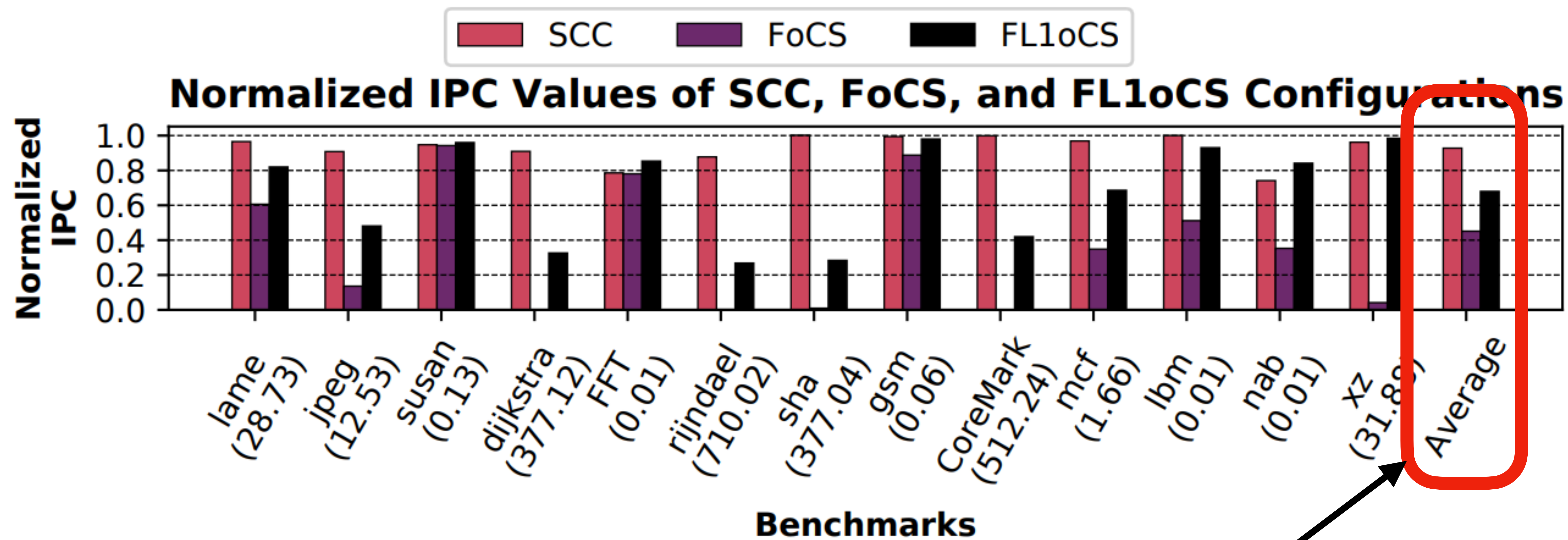
- **FoCS:** Flush on Compartment Switch
- **FL1oCS:** Flush L1 on Compartment Switch (SCC is implemented on other caches)



BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Performance Comparison Against Cache Flushing

- **FoCS:** Flush on Compartment Switch
- **FL1oCS:** Flush L1 on Compartment Switch (SCC is implemented on other caches)

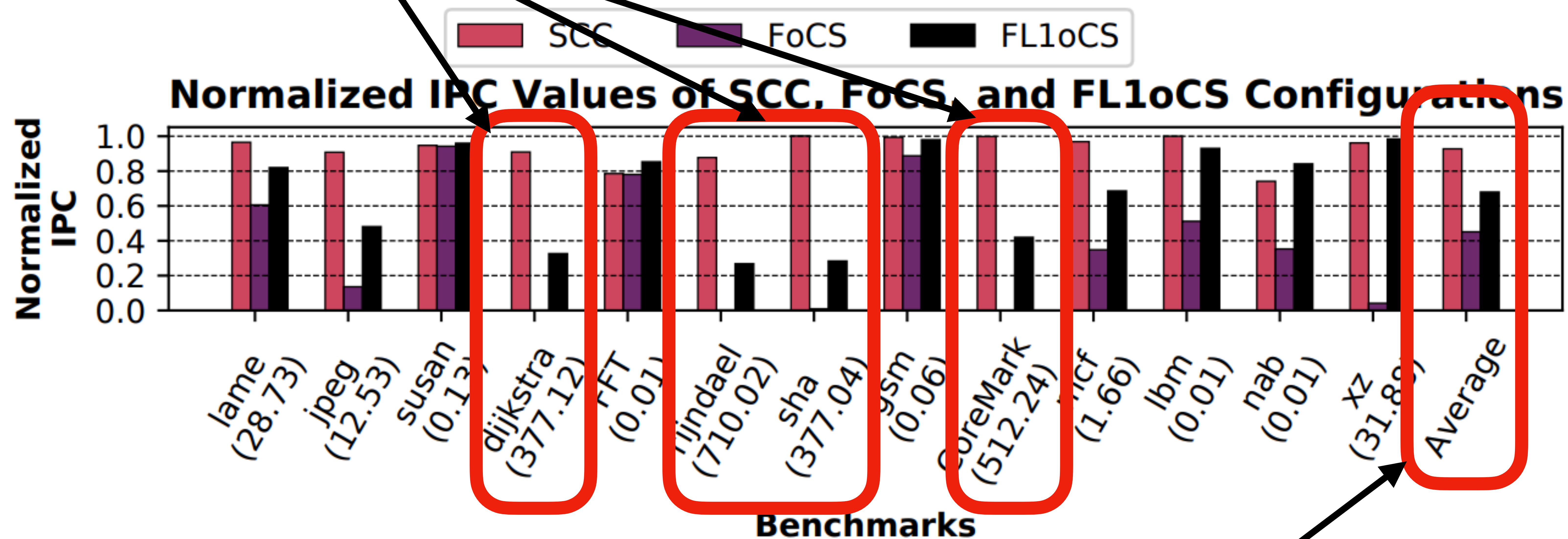


7%, 60%, and 32% performance loss for each configuration respectively.

Performance Comparison Against Cache Flushing

- **FoCS:** Flush on Compartment Switch

Frequent compartment switches cause extreme performance loss (close to 0% of the baseline).



7%, 60%, and 32% performance loss for each configuration respectively.

BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Performance Comparison Against Other Possible Partitioning Approaches

- **Proportional:** Domain partition sizes are proportional to their respective number of allocated pages.
- **Static-X:** X number of partitions are preallocated and uniformly sized.

Performance Comparison Against Other Possible Partitioning Approaches

- **Proportional:** Domain partition sizes are proportional to their respective number of allocated pages.

- **Static-X:** X number of partitions are preallocated and uniformly sized.

SCC outperforms both Proportional and Static configurations.

BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Hardware and Power Overhead

- Hardware area estimations are done with the McPAT tool - a CACTI extension.

Component	Area (mm^2)	Peak Dynamic (W)	RT Dynamic (W)
Baseline Core	16.242 (100%)	36.615 (100%)	36.615 (100%)
ADR	<0.001 (<0.001%)	<0.001 (<0.001%)	<0.001 (+0.002%)
DRT (L1i/d)	<0.001 (+0.002%)	0.003 (+0.009%)	0.009 (+0.027%)
DRT (L2)	0.001 (+0.007%)	0.008 (+0.022%)	0.026 (+0.070%)
DRTs (Total)	0.002 (+0.011%)	0.015 (+0.039%)	0.046 (+0.124%)
Extra Tag Bits	0.101 (+0.62%)	0.212 (+0.29%)	0.113 (+0.33%)
Total	0.103 (+0.63%)	0.224 (+0.61%)	0.155 (+0.42%)

Hardware and Power Overhead

- Hardware area estimations are done with the McPAT tool - a CACTI extension.

Component	Area (mm^2)	Peak Dynamic (W)	RT Dynamic (W)
Baseline Core	16.242 (100%)	36.615 (100%)	36.615 (100%)
ADR	<0.001 (<0.001%)	<0.001 (<0.001%)	<0.001 (+0.002%)
DRT (L1i/d)	<0.001 (+0.002%)	0.003 (+0.009%)	0.009 (+0.027%)
DRT (L2)	0.001 (+0.007%)	0.008 (+0.022%)	0.026 (+0.070%)
DRTs (Total)	0.002 (+0.011%)	0.015 (+0.039%)	0.046 (+0.124%)
Extra Tag Bits	0.101 (+0.62%)	0.212 (+0.29%)	0.113 (+0.33%)
Total	0.103 (+0.63%)	0.224 (+0.61%)	0.155 (+0.42%)

SCC incurs modest area overhead and power consumption.

BINGHAMTON UNIVERSITY STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

Summary

Summary

- Compartmentalized software is critical to limit attack exposure.

Summary

- Compartmentalized software is critical to limit attack exposure.
- Hardware caches can still leak information even with compartments

Summary

- Compartmentalized software is critical to limit attack exposure.
- Hardware caches can still leak information even with compartments
- Need new cache designs to address this leakage

Summary

- Compartmentalized software is critical to limit attack exposure.
- Hardware caches can still leak information even with compartments
- Need new cache designs to address this leakage
- **SCC** is the first step in this direction:

Summary

- Compartmentalized software is critical to limit attack exposure.
- Hardware caches can still leak information even with compartments
- Need new cache designs to address this leakage
- **SCC** is the first step in this direction:
    - Introduces **Domain-Oriented Partitioning**

Summary

- Compartmentalized software is critical to limit attack exposure.
- Hardware caches can still leak information even with compartments
- Need new cache designs to address this leakage
- **SCC** is the first step in this direction:
  - Introduces **Domain-Oriented Partitioning**
  - Implements **latency-aware L1 caches**

Summary

- Compartmentalized software is critical to limit attack exposure.
- Hardware caches can still leak information even with compartments
- Need new cache designs to address this leakage
- **SCC** is the first step in this direction:
  - Introduces **Domain-Oriented Partitioning**
  - Implements **latency-aware L1 caches**
  - **Mitigates library-based side-channel attacks**

Summary

- Compartmentalized software is critical to limit attack exposure.
- Hardware caches can still leak information even with compartments
- Need new cache designs to address this leakage
- **SCC** is the first step in this direction:
  - Introduces **Domain-Oriented Partitioning**
  - Implements **latency-aware L1 caches**
  - **Mitigates library-based side-channel attacks**
- **Evaluated SCC** in gem5 simulator and McPAT:

Summary

- Compartmentalized software is critical to limit attack exposure.
- Hardware caches can still leak information even with compartments
- Need new cache designs to address this leakage
- **SCC** is the first step in this direction:
  - Introduces **Domain-Oriented Partitioning**
  - Implements **latency-aware L1 caches**
  - **Mitigates library-based side-channel attacks**
- **Evaluated SCC** in gem5 simulator and McPAT:
  - Incurs **7%** performance loss and **0.63%** area overhead

More in the Paper…

Secure Caches for Compartmentalized Software

Kerem Arıkan[1], Huaxin Tang[1], Williams Zhang Cen[1],
Yu David Liu[1], Nael Abu-Ghazaleh[2] and Dmitry Ponomarev[1]

[1]*Binghamton University*
[2]*University of California, Riverside*

Abstract

Compartmentalized software systems have been recently proposed in response to security challenges with traditional process-level isolation mechanisms. Compartments provide logical isolation for mutually mistrusting software components, even within the same address space. However, they do not provide side-channel isolation, leaving them vulnerable to side-channel attacks. In this paper, we take on the problem of protecting compartmentalized software from hardware cache side-channel attacks. We consider unique challenges that compartmentalized software poses in terms of securing caches, which include performance implications, efficient and secure data sharing, and avoiding leakage when shared libraries are called by multiple callers. We propose SCC - a framework that addresses these challenges by 1) multi-level cache partitioning including L1 caches with a series of optimizations to minimize performance impact; 2) the concept of domain-oriented partitioning where cache partitions are created per memory domain, instead of per compartment; and 3) creating a separate partition instance of a shared library code for each caller. We formally prove the security of SCC using operational semantics and evaluate its performance using the gem5 simulator on a set of compartmentalized benchmarks.

1 Introduction

Modern programs are becoming increasingly complex software systems that often integrate code developed by independent and mutually untrusting parties. The interactions between code components often take place using insecure interfaces [7,16,26,34,50,53,63,65,72,75,82,84,85,90,97,104]. As an example, consider a browser that incorporates a just-in-time compilation module that compiles and executes a web application which is linked to a cryptographic library storing secret keys. In this case, a malicious web application can potentially compromise secrets held in the cryptographic library, as well as the data of the compilation module itself. Traditional process-centric isolation security models are insufficient to protect systems and applications from such vulnerabilities within the same address space.

In response to these emerging threats, several *in-process compartmentalization* mechanisms have been developed to provide intra-process memory isolation [2, 4, 19, 39, 40, 42, 66, 70]. In the above browser example, in-process isolation can prevent the web application code from accessing memory regions of the cryptographic library or the compilation engine, creating isolated compartments in memory within a single process. In-process compartmentalization solutions come in various forms, including secure enclaves (such as Intel SGX [1, 19]), page-based memory access control schemes [75], or capability-based systems [90]. Regardles
of the implementation and security principles behind the
designs, current proposals are developed around
memory accesses and ensuring computation
fortunately, side-channel attacks throu
resources have remained outside t
tions consider. At the same time, side-
larly those exploiting shared caches, have be
in diverse software environments that benefit from
compartmentalization, including browsers [30, 64, 77],
services [71], virtual machines [32, 44, 102], and graphi
frameworks [86].

In this paper, we propose new cache hierarchies that pr
tect compartmentalized software from side-channel attacks; to the best of our knowledge, this is the first paper that defines and explores this problem. At a high level, our solution augments existing memory protection schemes with *fine-grain cache partitioning*, where in-process compartments can control the data that is isolated from other compartments in the cache across multiple cache levels. Designing efficient and secure partitioned caches supporting compartment isolation requires solving several performance, functionality, and security-related challenges, which we describe next.

First, applying fine-grain partitioning to L1 caches is expensive because L1 caches are accessed frequently, have stringent latency constraints, and are likely to be on the critical path of execution. Previous work leverages flushing the L1 cache on a

BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

UC RIVERSIDE

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

More in the Paper…

Secure Caches for Compartmentalized Software

Kerem Arıkan[1], Huaxin Tang[1], Williams Zhang Cen[1],
Yu David Liu[1], Nael Abu-Ghazaleh[2] and Dmitry Ponomarev[1]

[1]*Binghamton University*
[2]*University of California, Riverside*

Abstract

Compartmentalized software systems have been recently proposed in response to security challenges with traditional process-level isolation mechanisms. Compartments provide logical isolation for mutually mistrusting software components, even within the same address space. However, they do not provide side-channel isolation, leaving them vulnerable to side-channel attacks. In this paper, we take on the problem of protecting compartmentalized software from hardware cache side-channel attacks. We consider unique challenges that compartmentalized software poses in terms of securing caches, which include performance implications, efficient and secure data sharing, and avoiding leakage when shared libraries are called by multiple callers. We propose SCC - a framework that addresses these challenges by 1) multi-level cache partitioning including L1 caches with a series of optimizations to minimize performance impact; 2) the concept of domain-oriented partitioning where cache partitions are created per memory domain, instead of per compartment; and 3) creating a separate partition instance of a shared library code for each caller. We formally prove the security of SCC using operational semantics and evaluate its performance using the gem5 simulator on a set of compartmentalized benchmarks.

1 Introduction

Modern programs are becoming increasingly complex software systems that often integrate code developed by independent and mutually untrusting parties. The interactions between code components often take place using insecure interfaces [7,16,26,34,50,53,63,65,72,75,82,84,85,90,97,104]. As an example, consider a browser that incorporates a just-in-time compilation module that compiles and executes a web application which is linked to a cryptographic library storing secret keys. In this case, a malicious web application can potentially compromise secrets held in the cryptographic library, as well as the data of the compilation module itself. Traditional process-centric isolation security models are insufficient to protect systems and applications from such vulnerabilities within the same address space.

In response to these emerging threats, several *in-process compartmentalization* mechanisms have been developed to provide intra-process memory isolation [2, 4, 19, 39, 40, 42, 66, 70]. In the above browser example, in-process isolation can prevent the web application code from accessing memory regions of the cryptographic library or the compilation engine, creating isolated compartments in memory within a single process. In-process compartmentalization solutions come in various forms, including secure enclaves (such as Intel SGX [1, 19]), page-based memory access control schemes [75], or capability-based systems [90]. Regardles
of the implementation and security principles behind the
designs, current proposals are developed around pr
memory accesses and ensuring computation
fortunately, side-channel attacks throu
resources have remained outside t
tions consider. At the same time, side-c
larly those exploiting shared caches, have be
in diverse software environments that benefit from
compartmentalization, including browsers [30, 64, 77], cl
services [71], virtual machines [32, 44, 102], and graphi
frameworks [86].

In this paper, we propose new cache hierarchies that pr
tect compartmentalized software from side-channel attacks; to the best of our knowledge, this is the first paper that defines and explores this problem. At a high level, our solution augments existing memory protection schemes with *fine-grain cache partitioning*, where in-process compartments can control the data that is isolated from other compartments in the cache across multiple cache levels. Designing efficient and secure partitioned caches supporting compartment isolation requires solving several performance, functionality, and security-related challenges, which we describe next.

First, applying fine-grain partitioning to L1 caches is expensive because L1 caches are accessed frequently, have stringent latency constraints, and are likely to be on the critical path of execution. Previous work leverages flushing the L1 cache on a

- Formal security analysis

BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK

UC RIVERSIDE

usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION

More in the Paper…

Secure Caches for Compartmentalized Software

Kerem Arıkan[1], Huaxin Tang[1], Williams Zhang Cen[1],
Yu David Liu[1], Nael Abu-Ghazaleh[2] and Dmitry Ponomarev[1]

[1] *Binghamton University*
[2] *University of California, Riverside*

Abstract

Compartmentalized software systems have been recently proposed in response to security challenges with traditional process-level isolation mechanisms. Compartments provide logical isolation for mutually mistrusting software components, even within the same address space. However, they do not provide side-channel isolation, leaving them vulnerable to side-channel attacks. In this paper, we take on the problem of protecting compartmentalized software from hardware cache side-channel attacks. We consider unique challenges that compartmentalized software poses in terms of securing caches, which include performance implications, efficient and secure data sharing, and avoiding leakage when shared libraries are called by multiple callers. We propose SCC - a framework that addresses these challenges by 1) multi-level cache partitioning including L1 caches with a series of optimizations to minimize performance impact; 2) the concept of domain-oriented partitioning where cache partitions are created per memory domain, instead of per compartment; and 3) creating a separate partition instance of a shared library code for each caller. We formally prove the security of SCC using operational semantics and evaluate its performance using the gem5 simulator on a set of compartmentalized benchmarks.

1 Introduction

Modern programs are becoming increasingly complex software systems that often integrate code developed by independent and mutually untrusting parties. The interactions between code components often take place using insecure interfaces [7,16,26,34,50,53,63,65,72,75,82,84,85,90,97,104]. As an example, consider a browser that incorporates a just-in-time compilation module that compiles and executes a web application which is linked to a cryptographic library storing secret keys. In this case, a malicious web application can potentially compromise secrets held in the cryptographic library, as well as the data of the compilation module itself. Traditional process-centric isolation security models are insufficient to protect systems and applications from such vulnerabilities within the same address space.

In response to these emerging threats, several *in-process compartmentalization* mechanisms have been developed to provide intra-process memory isolation [2, 4, 19, 39, 40, 42, 66, 70]. In the above browser example, in-process isolation can prevent the web application code from accessing memory regions of the cryptographic library or the compilation engine, creating isolated compartments in memory within a single process. In-process compartmentalization solutions come in various forms, including secure enclaves (such as Intel SGX [1, 19]), page-based memory access control schemes [75], or capability-based systems [90]. Regardles of the implementation and security principles behind the designs, current proposals are developed around pr memory accesses and ensuring computation fortunately, side-channel attacks throu resources have remained outside t tions consider. At the same time, side-c larly those exploiting shared caches, have be in diverse software environments that benefit from compartmentalization, including browsers [30, 64, 77], cl services [71], virtual machines [32, 44, 102], and graph frameworks [86].

In this paper, we propose new cache hierarchies that pr tect compartmentalized software from side-channel attacks; to the best of our knowledge, this is the first paper that defines and explores this problem. At a high level, our solution augments existing memory protection schemes with *fine-grain cache partitioning*, where in-process compartments can control the data that is isolated from other compartments in the cache across multiple cache levels. Designing efficient and secure partitioned caches supporting compartment isolation requires solving several performance, functionality, and security-related challenges, which we describe next.

First, applying fine-grain partitioning to L1 caches is expensive because L1 caches are accessed frequently, have stringent latency constraints, and are likely to be on the critical path of execution. Previous work leverages flushing the L1 cache on a

- Formal security analysis
- Further implementation details

BINGHAMTON UNIVERSITY
STATE UNIVERSITY OF NEW YORK
UC RIVERSIDE
usenix
THE ADVANCED COMPUTING SYSTEMS ASSOCIATION