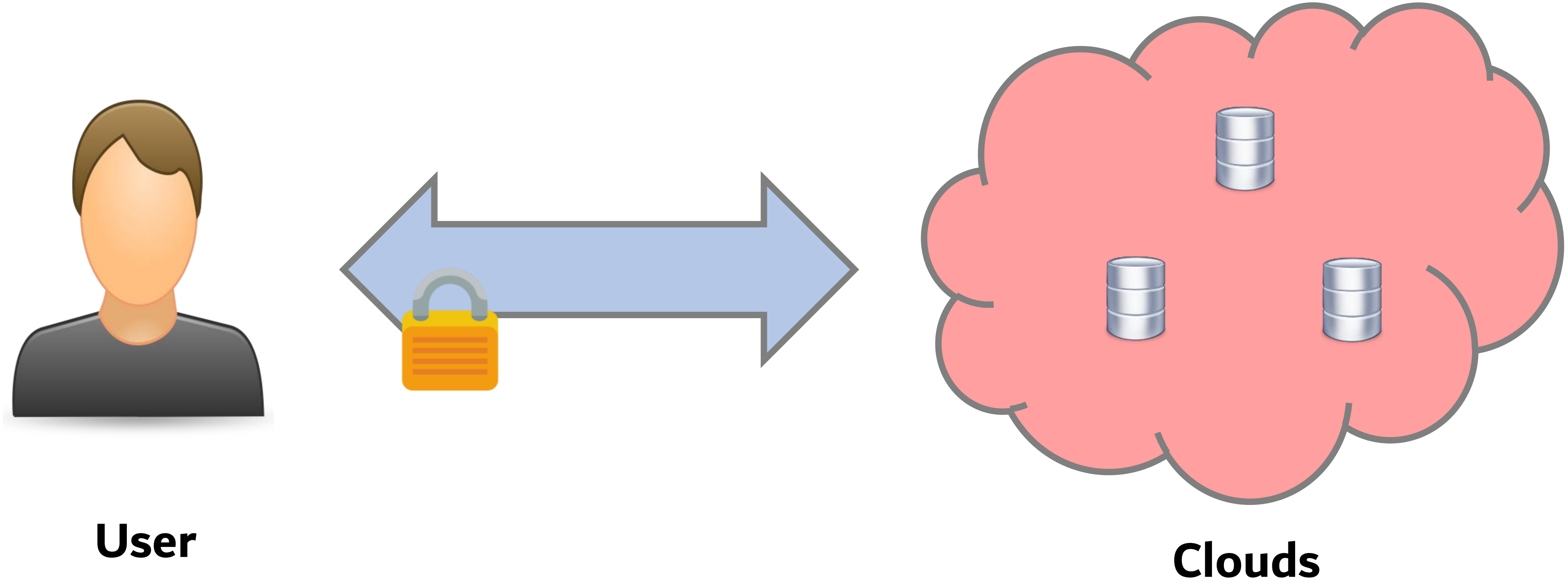
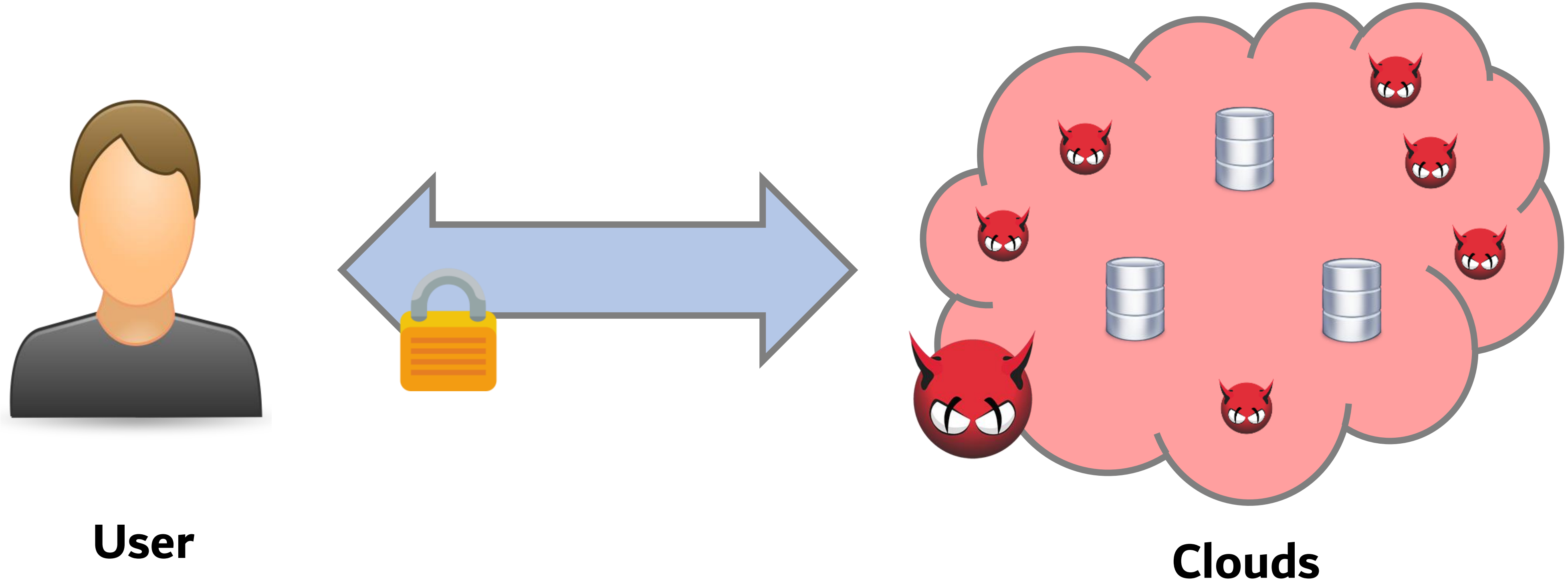
# Obliviate: A Data Oblivious File System for Intel SGX

**Adil Ahmad**
Kyungtae Kim
Muhammad Ihsanulhaq Sarfaraz
Byoungyoung Lee

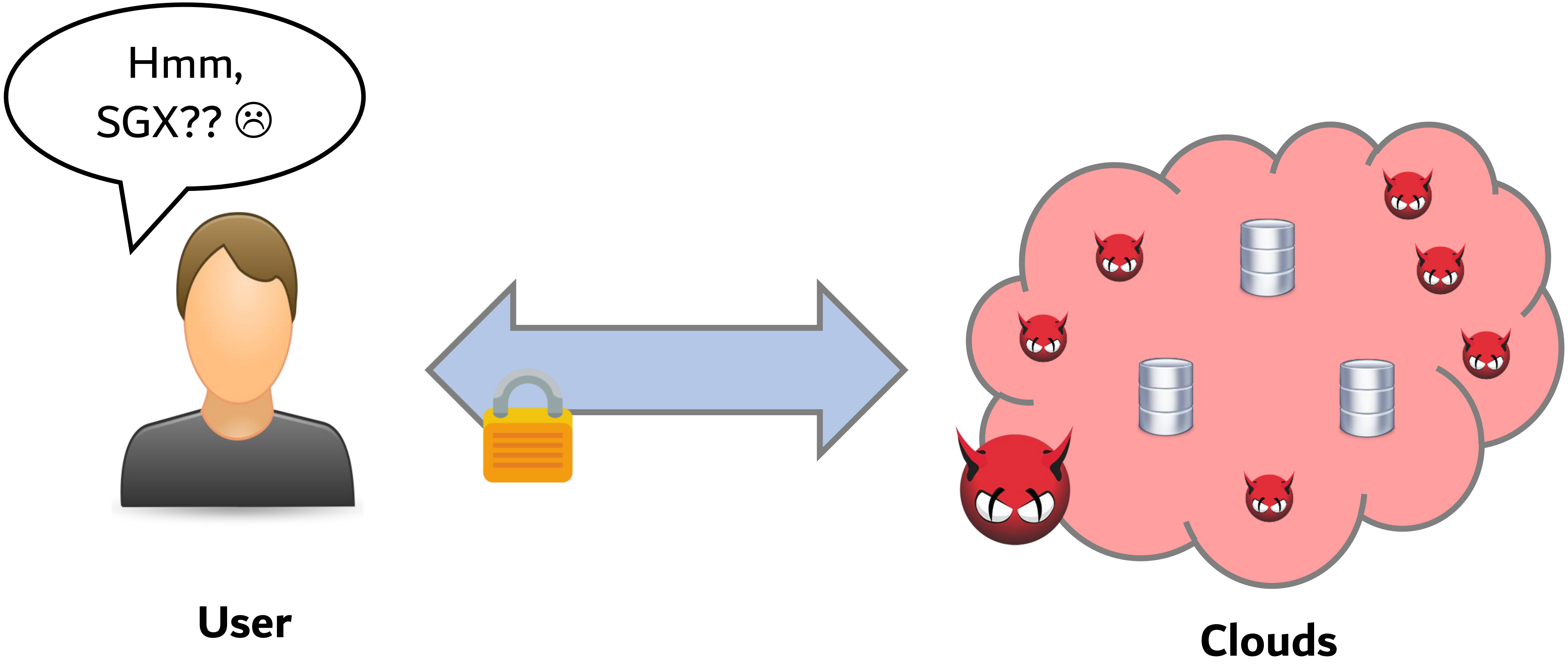
# Clouds? The Ultimate Dream?



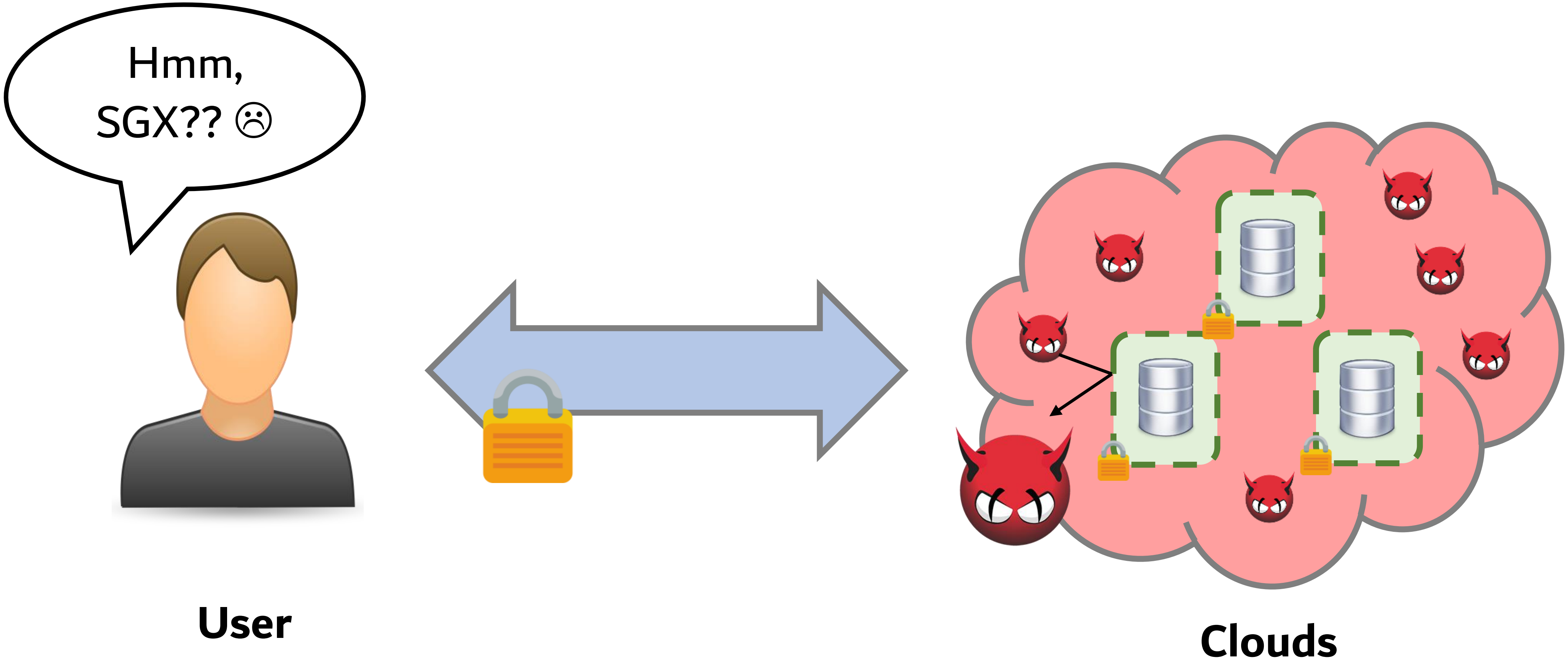
# Clouds? The Ultimate Dream?



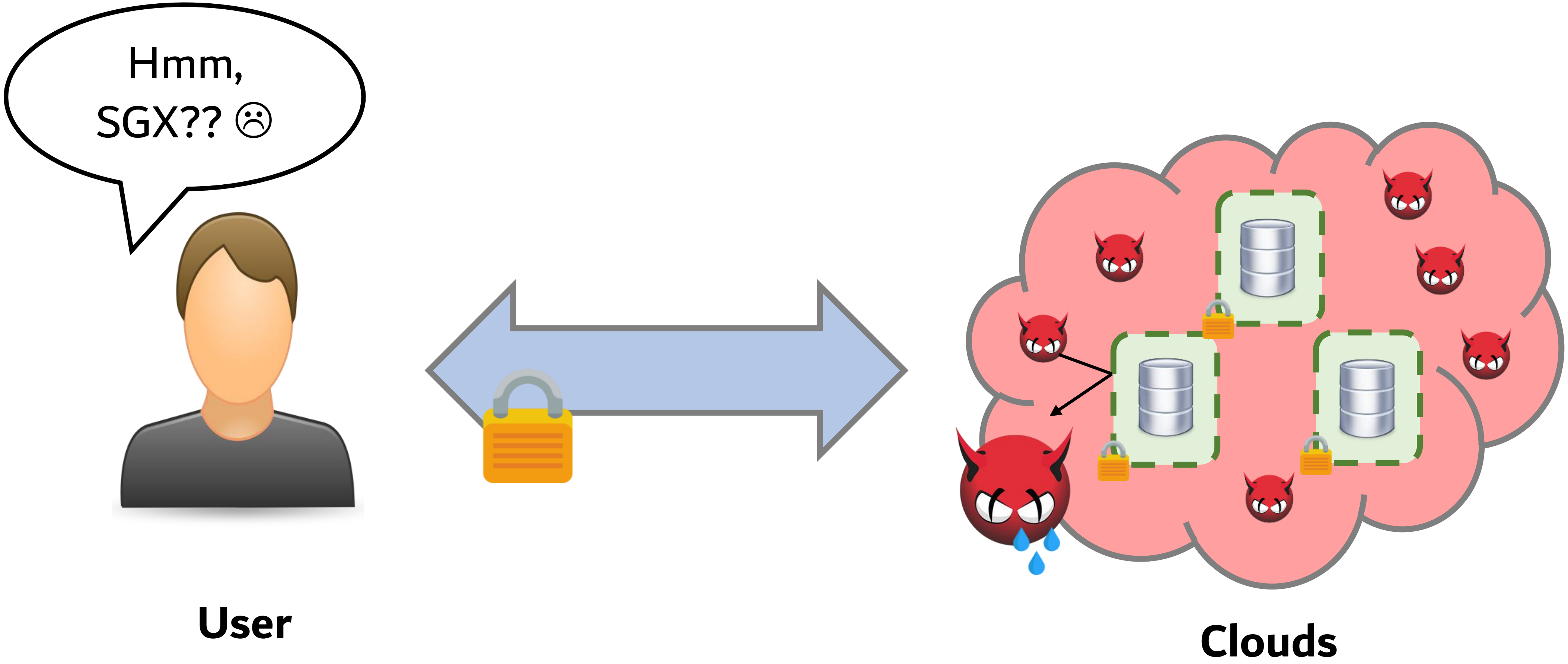
# Clouds? The Ultimate Dream?



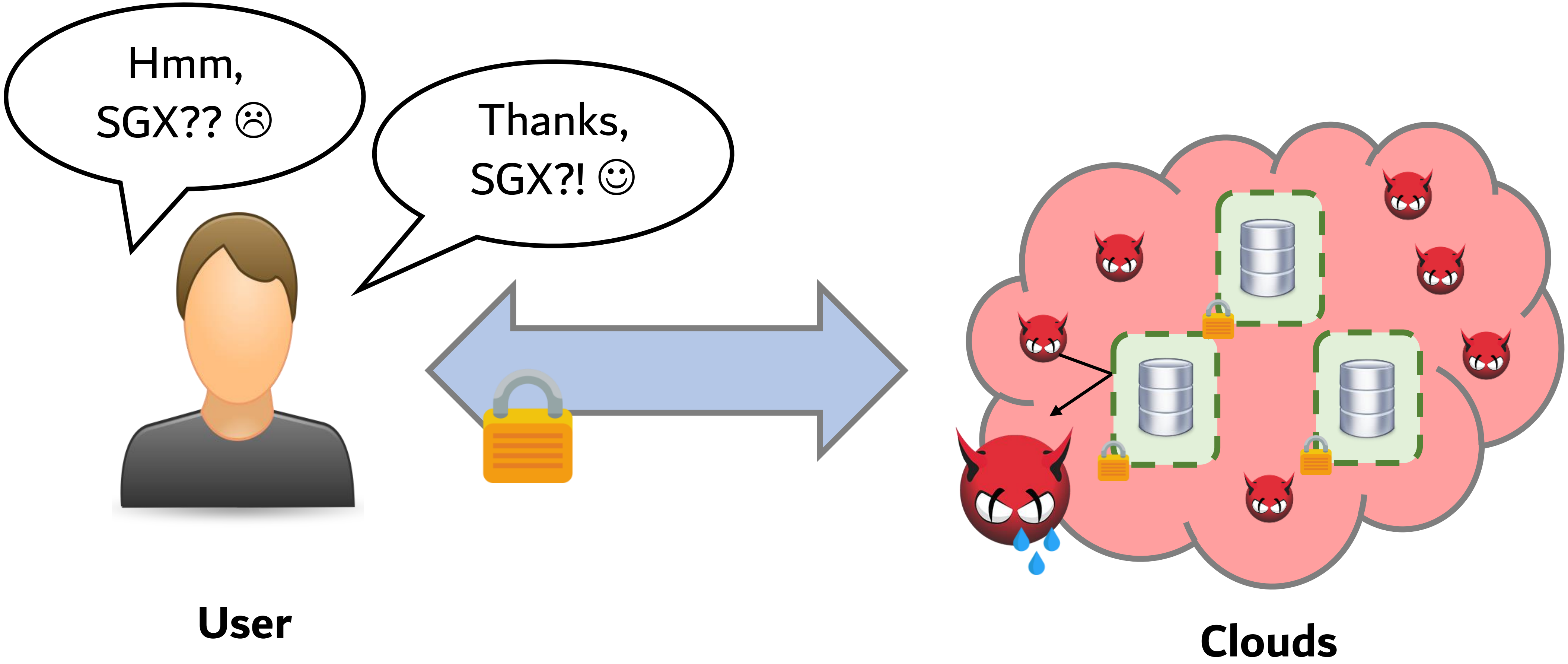
# Clouds? The Ultimate Dream?



# Clouds? The Ultimate Dream?



# Clouds? The Ultimate Dream?



# Clouds? The Ultimate Dream?

Hmm,
SGX?? ☹

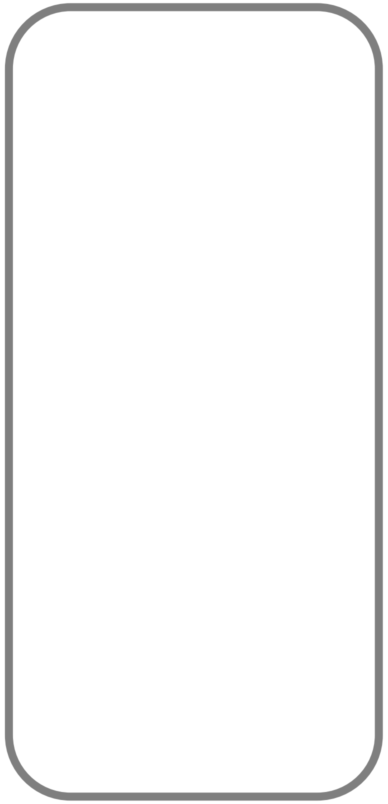
Thanks,

**The real world is a bit more complicated!**

User

Clouds

# The sorcery behind SGX



**Program's Address Space**

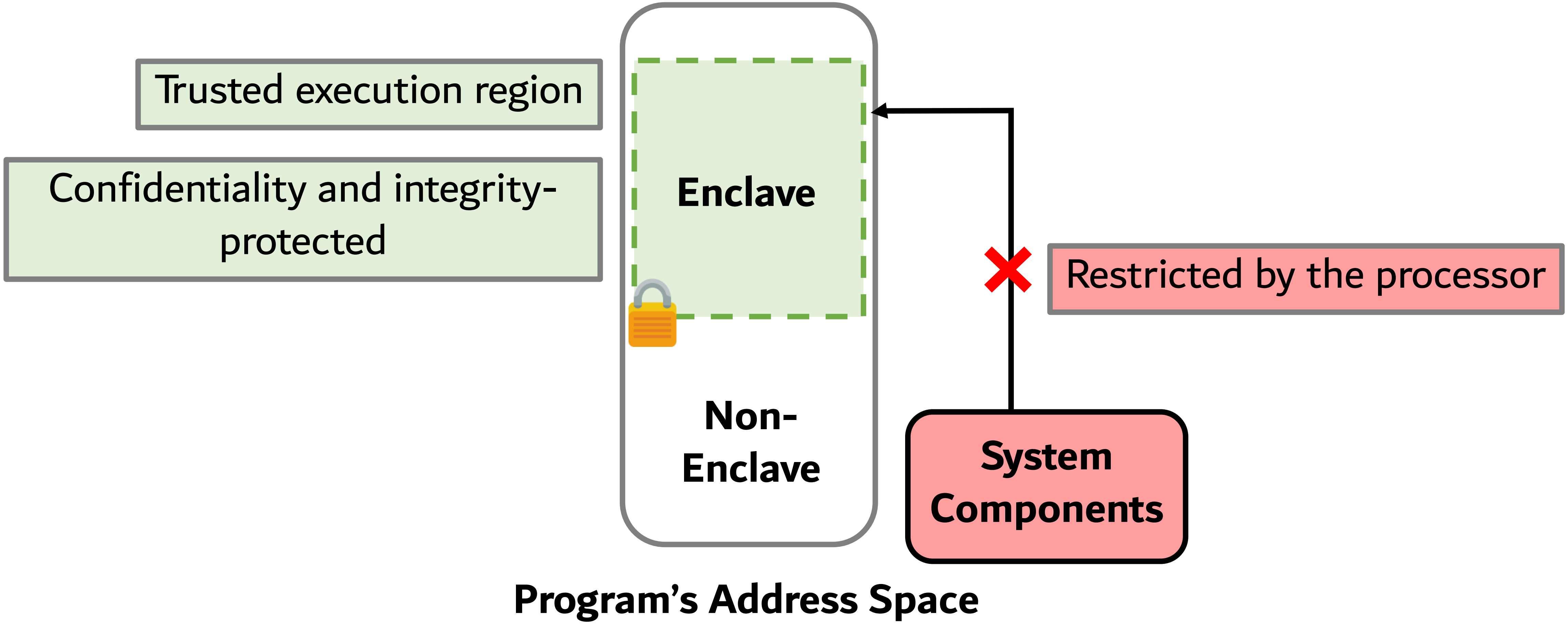
# The sorcery behind SGX

Trusted execution region

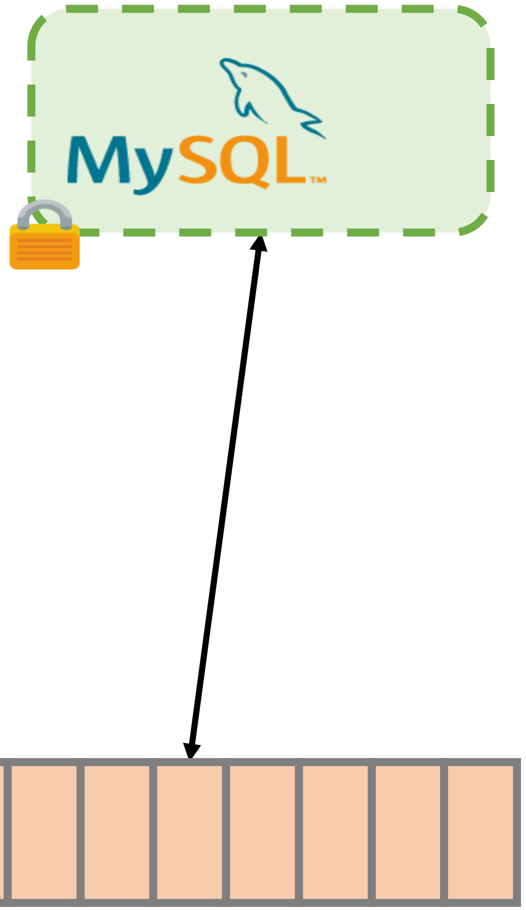
Confidentiality and integrity-protected


**Program's Address Space**

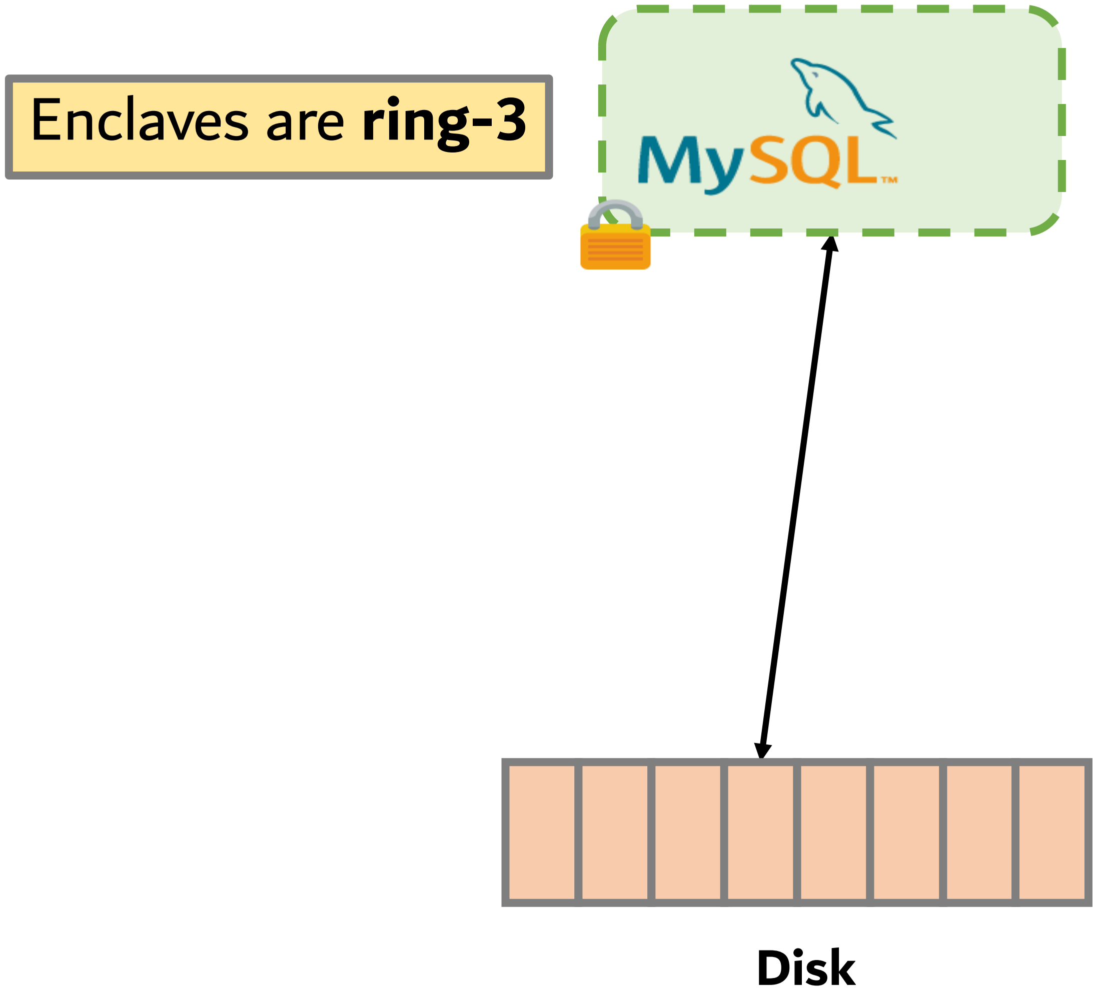
# The sorcery behind SGX



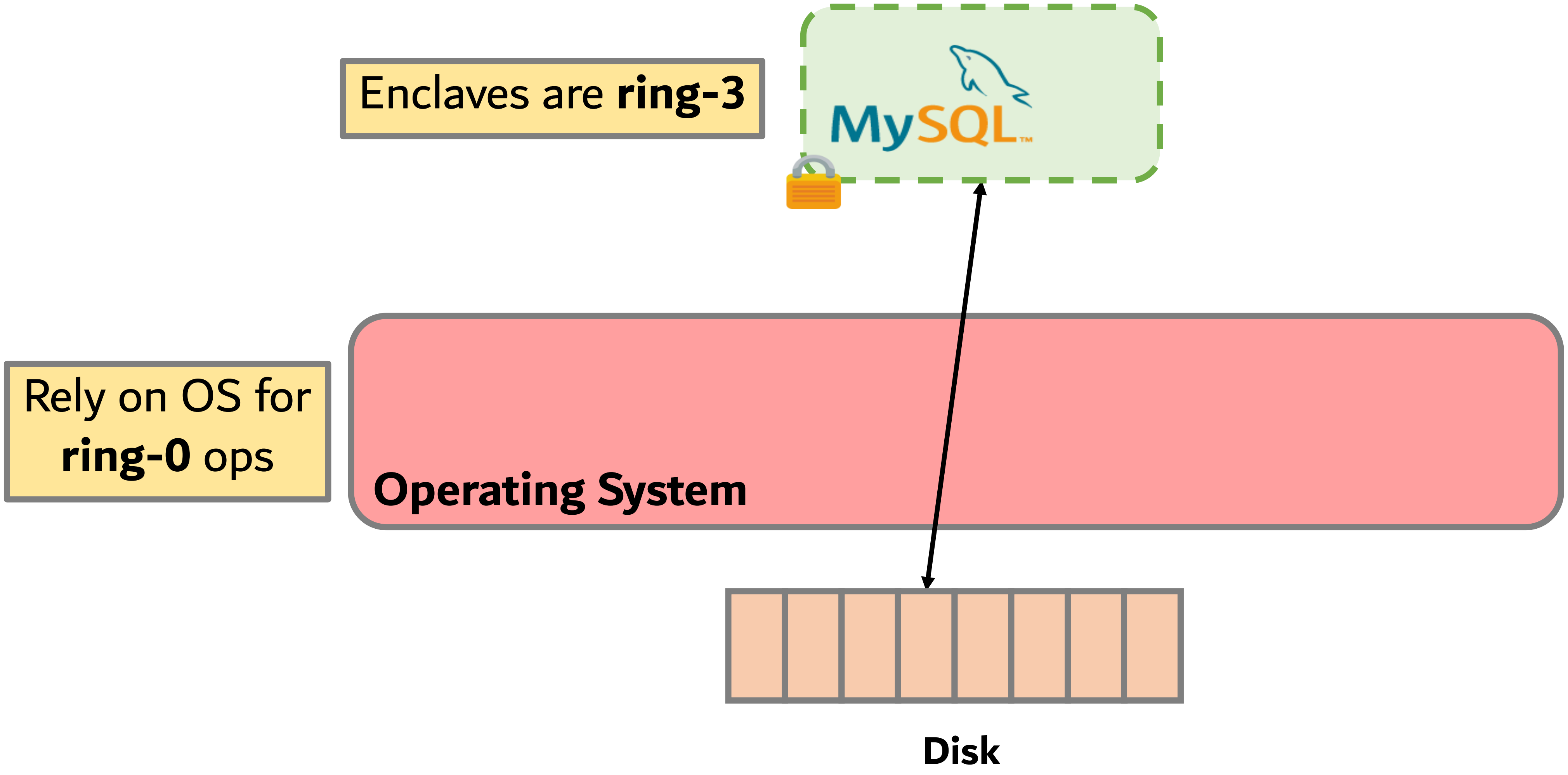
# Possible SGX File Systems



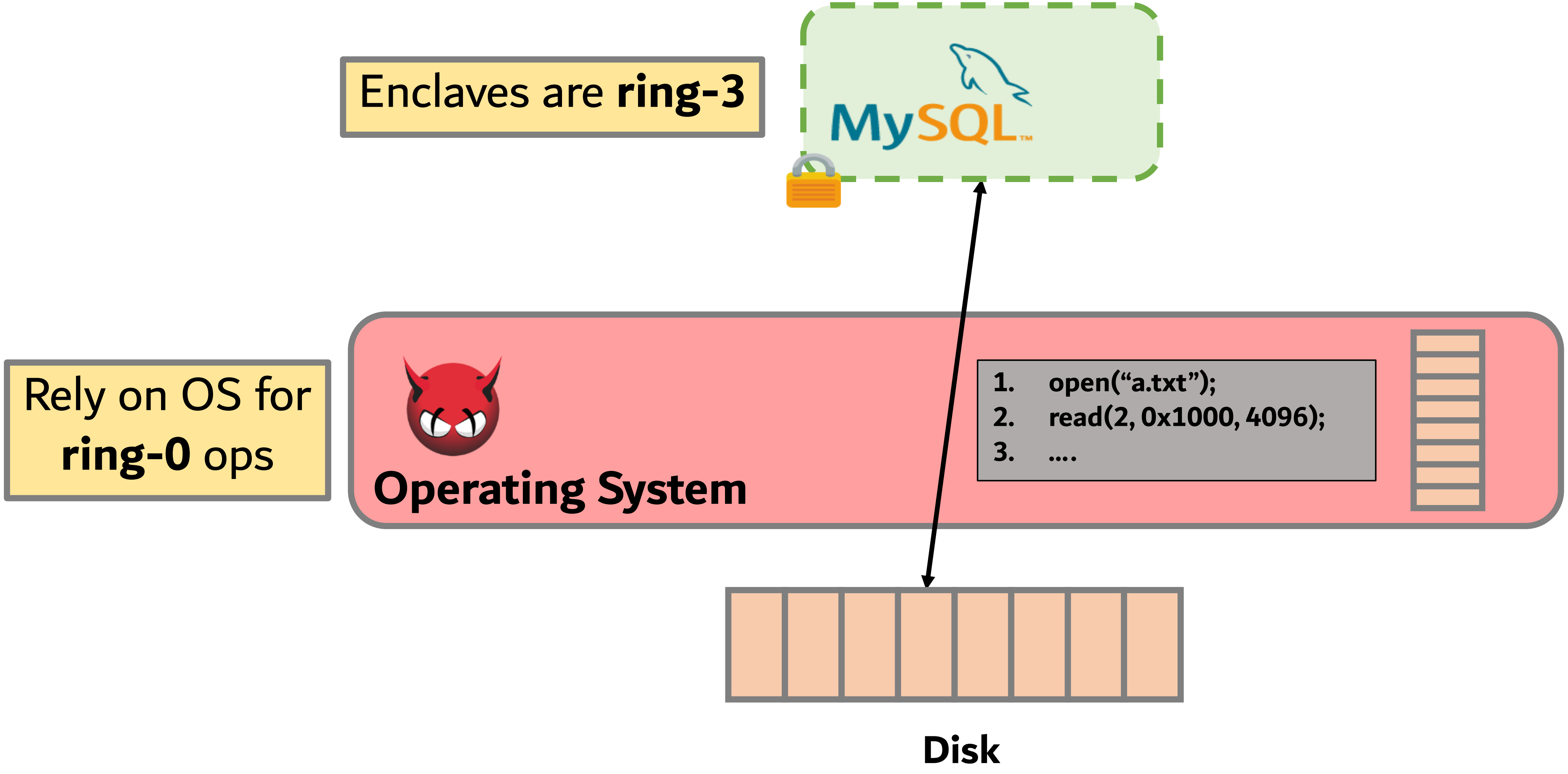
# Possible SGX File Systems



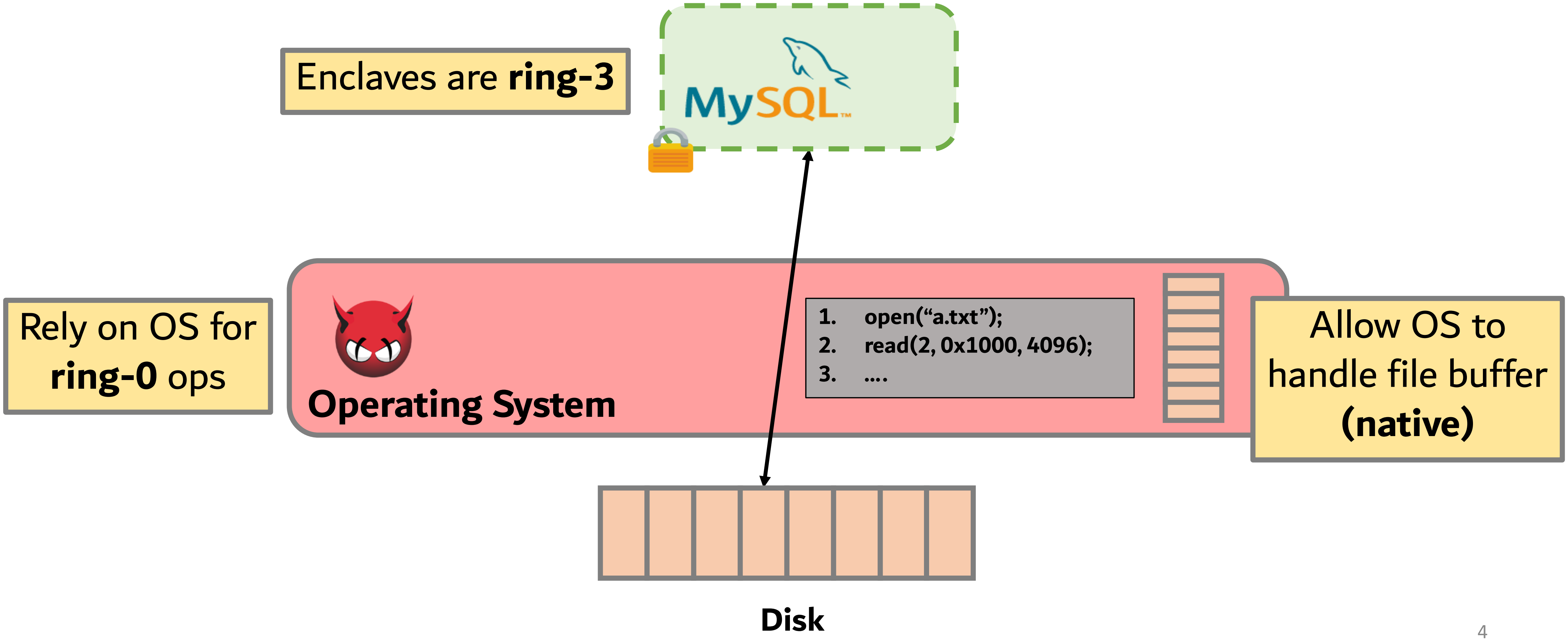
# Possible SGX File Systems



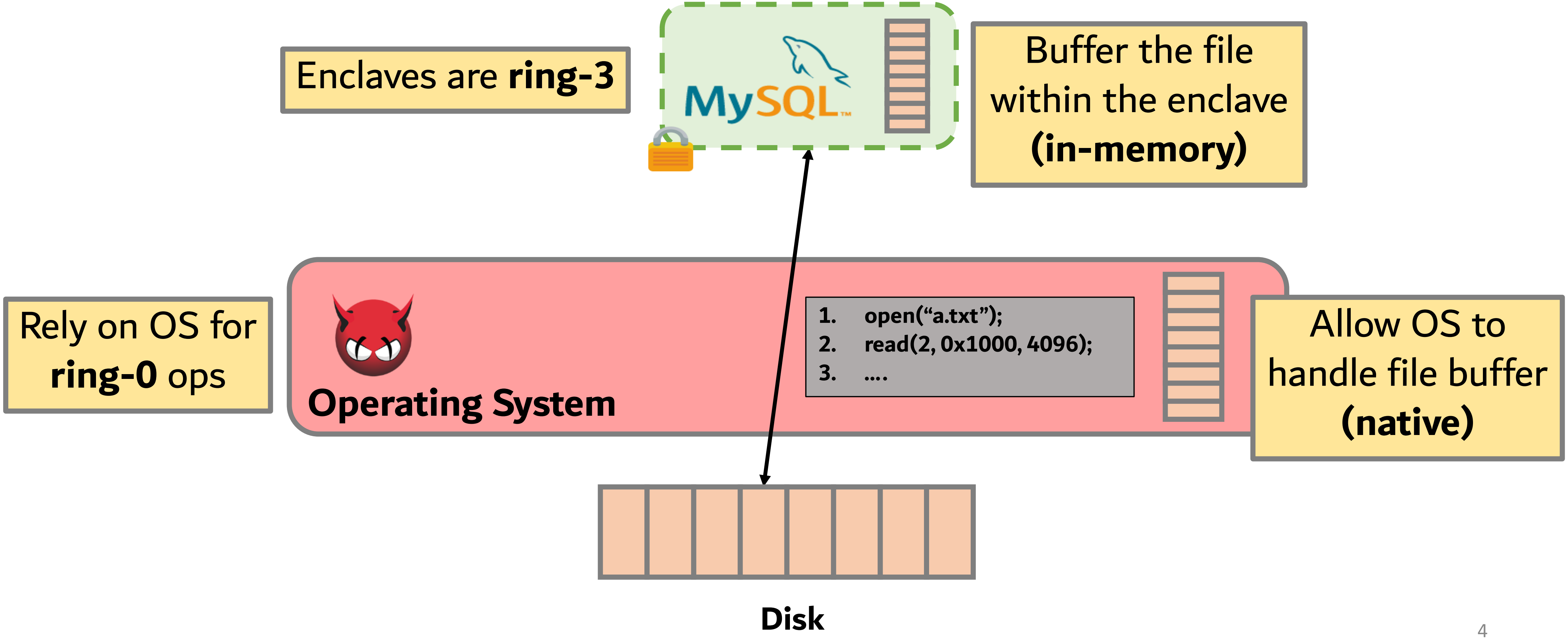
# Possible SGX File Systems



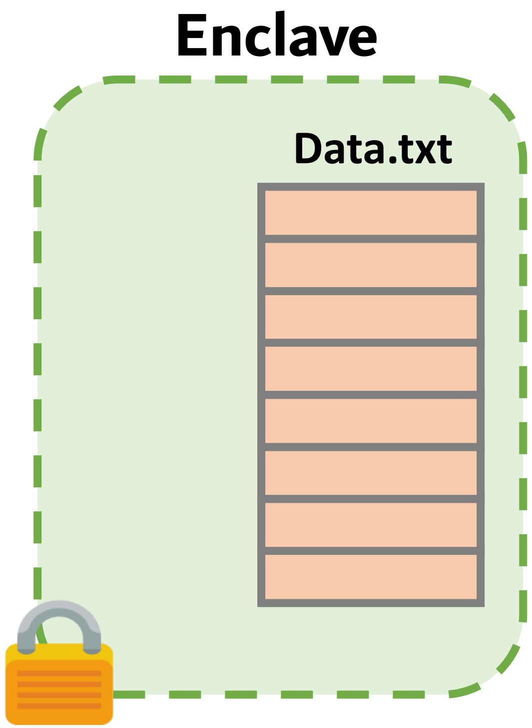
# Possible SGX File Systems



# Possible SGX File Systems



# Side-channel attacks against in-memory FS

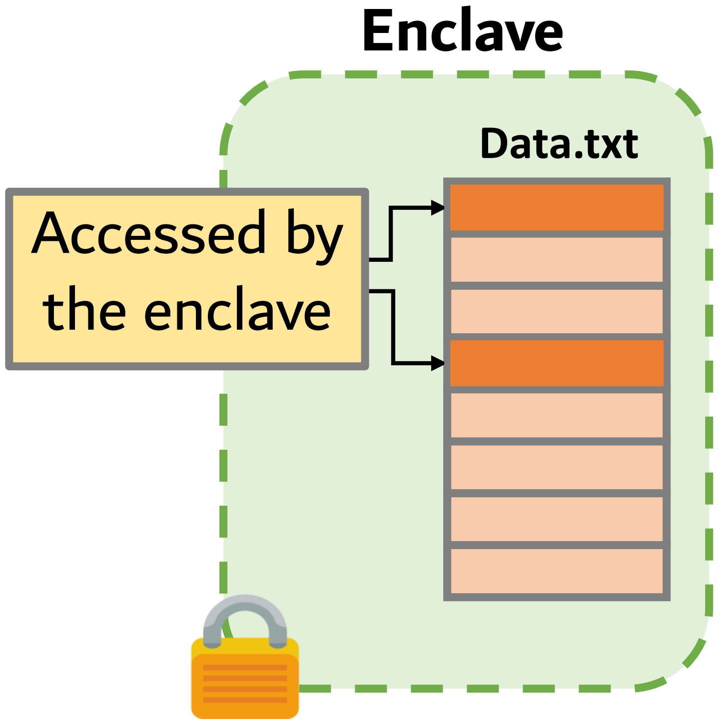




Page table attacks against SGX
[S&P14, SEC17]

Cache attacks against SGX
[DIMVA17, WOOT17, EuroSec17]

# Side-channel attacks against in-memory FS



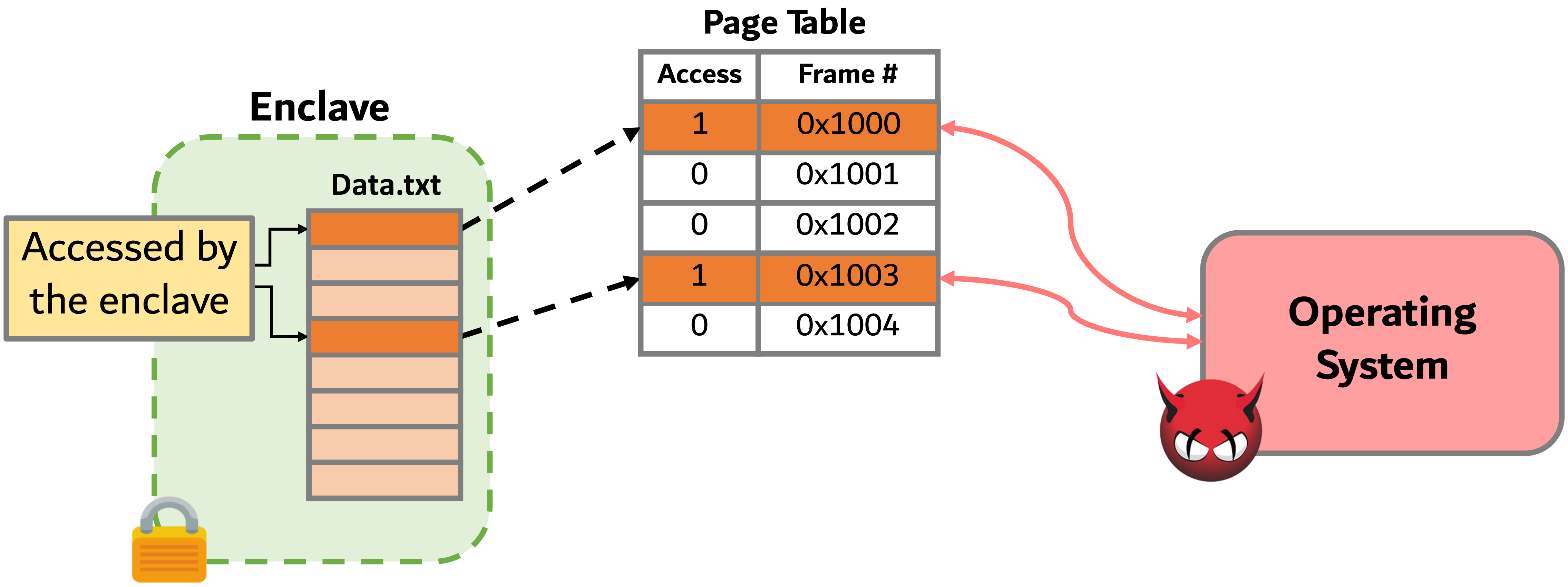

**Page Table**

| Access | Frame # |
|---|---|
| 0 | 0x1000 |
| 0 | 0x1001 |
| 0 | 0x1002 |
| 0 | 0x1003 |
| 0 | 0x1004 |


Page table attacks against SGX
[S&P14, SEC17]

Cache attacks against SGX
[DIMVA17, WOOT17, EuroSec17]

# Side-channel attacks against in-memory FS



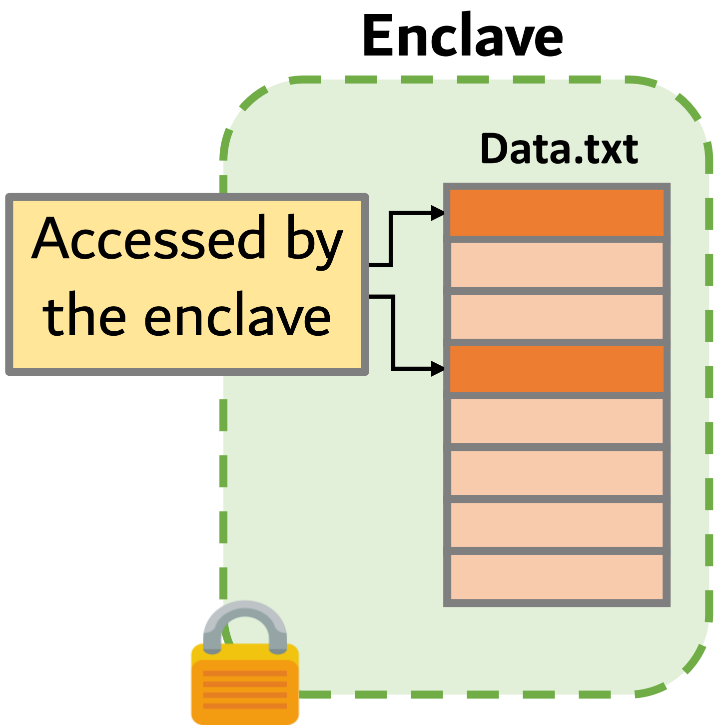

**Page Table**

| Access | Frame # |
| --- | --- |
| 1 | 0x1000 |
| 0 | 0x1001 |
| 0 | 0x1002 |
| 1 | 0x1003 |
| 0 | 0x1004 |

Page table attacks against SGX
[S&P14, SEC17]

Cache attacks against SGX
[DIMVA17, WOOT17, EuroSec17]

# Side-channel attacks against in-memory FS




**Page Table**

| Access | Frame # |
|---|---|
| 1 | 0x1000 |
| 0 | 0x1001 |
| 0 | 0x1002 |
| 1 | 0x1003 |
| 0 | 0x1004 |

**Cache**

| |
|---|
| cache-set 0 |
| cache-set 1 |
| cache-set 2 |
| cache-set 3 |

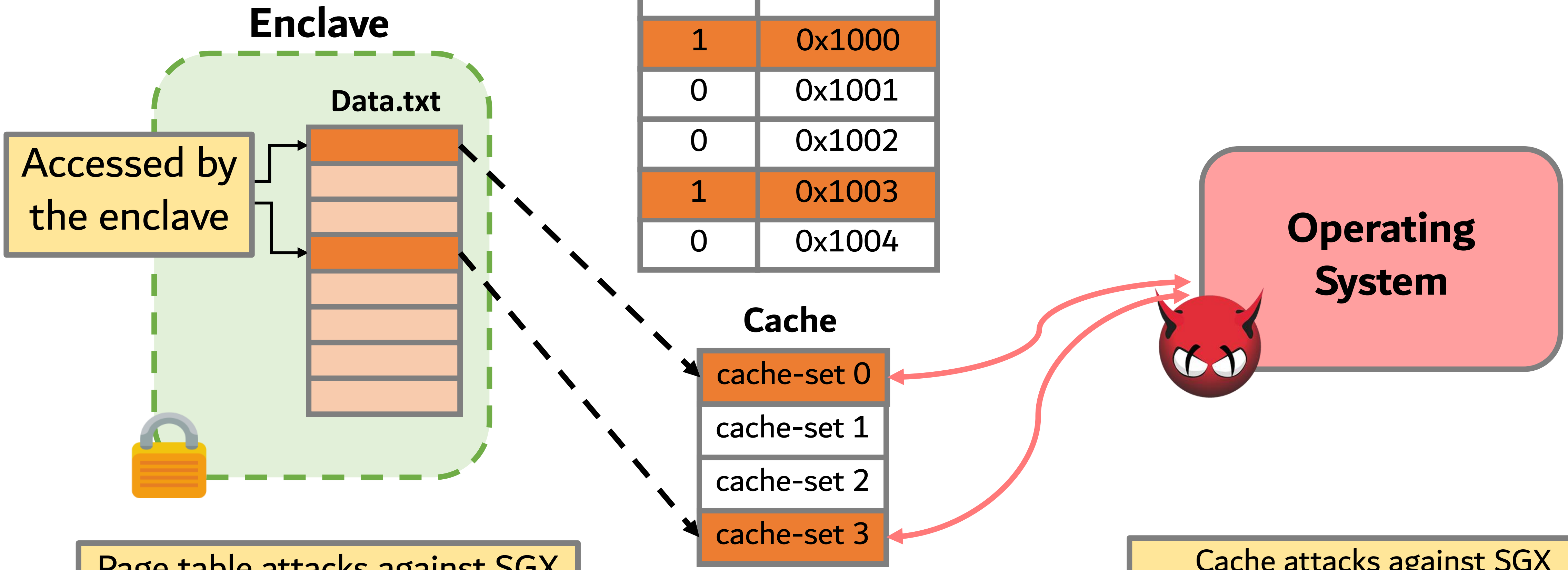

Page table attacks against SGX
[S&P14, SEC17]

Cache attacks against SGX
[DIMVA17, WOOT17, EuroSec17]

# Side-channel attacks against in-memory FS

**Page Table**

| Access | Frame # |
|---|---|
| 1 | 0x1000 |
| 0 | 0x1001 |
| 0 | 0x1002 |
| 1 | 0x1003 |
| 0 | 0x1004 |

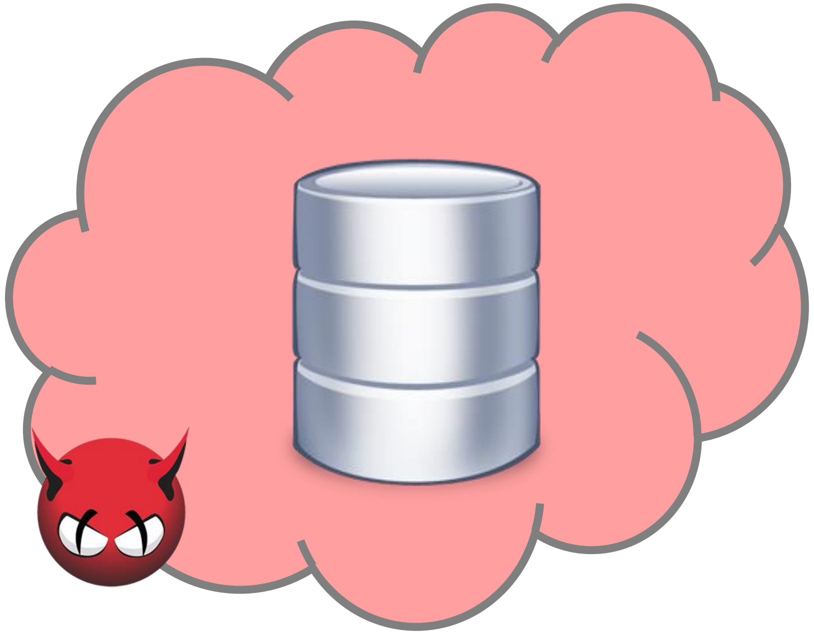



Page table attacks against SGX
[S&P14, SEC17]

Cache attacks against SGX
[DIMVA17, WOOT17, EuroSec17]

# Case Study: Attacking SQlite

Doctor

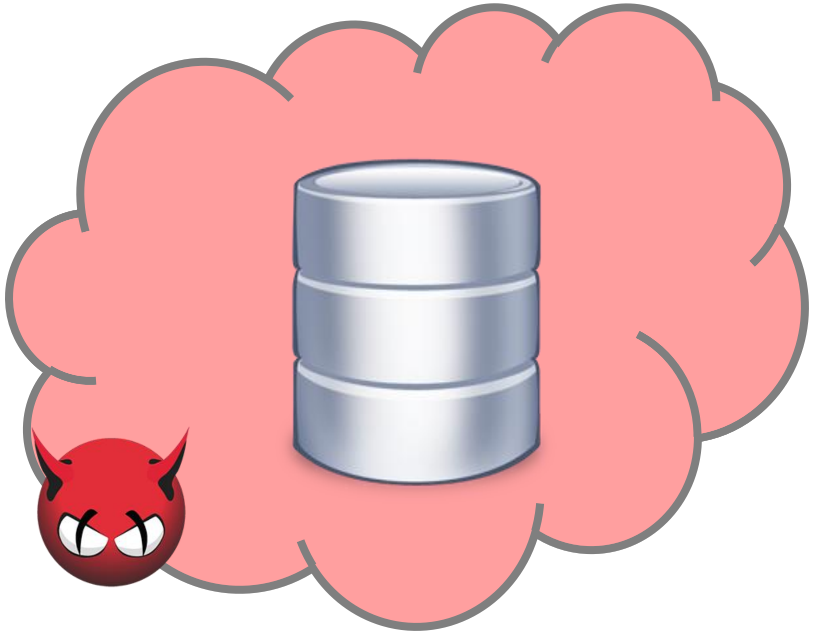


Cloud

# Case Study: Attacking SQlite

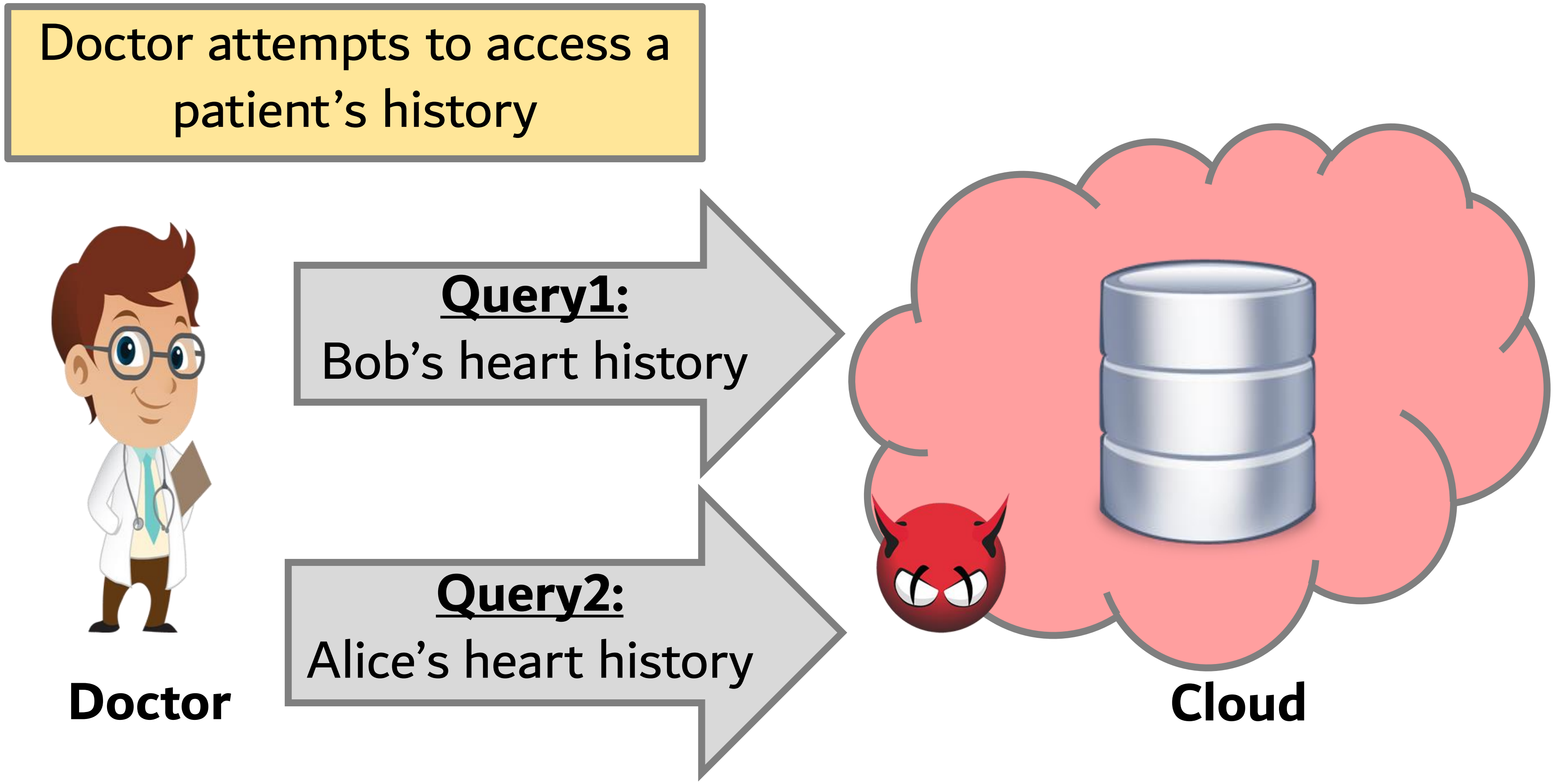
Doctor attempts to access a patient's history

Doctor

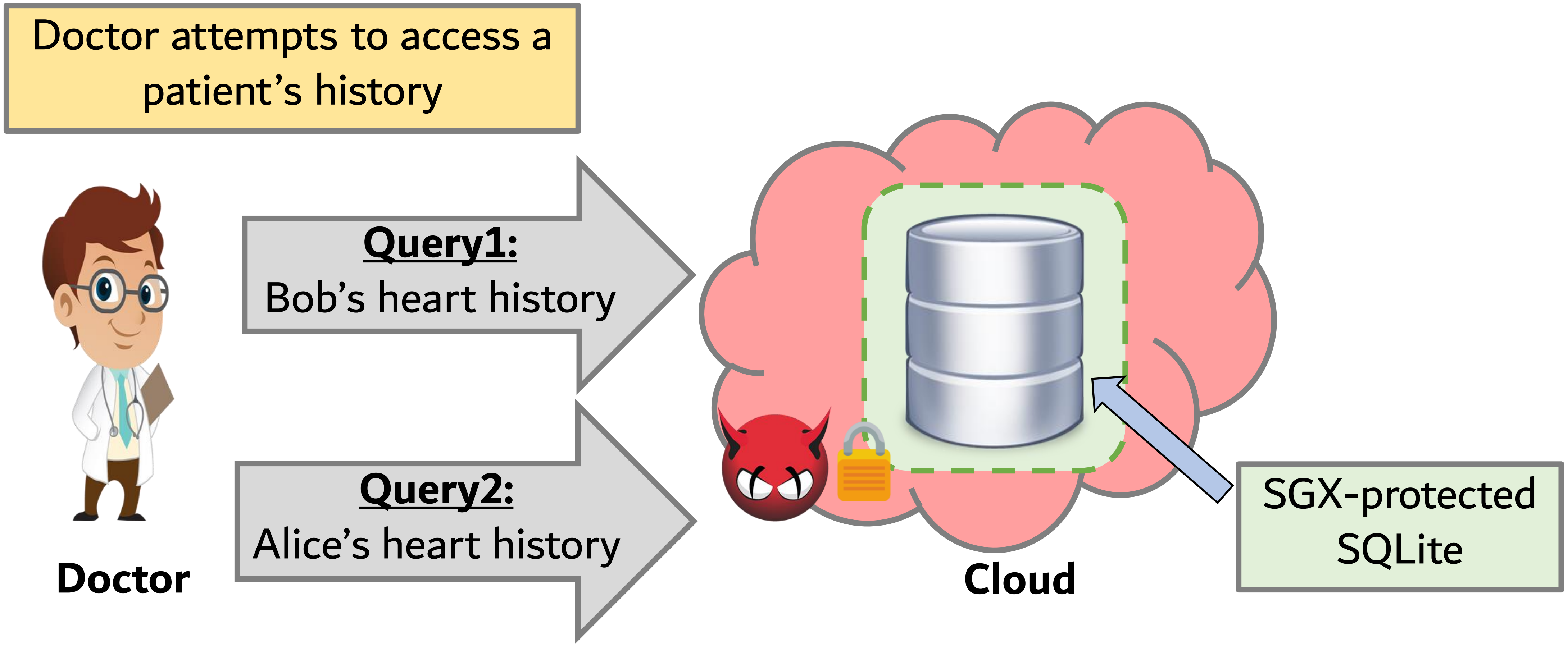


Cloud

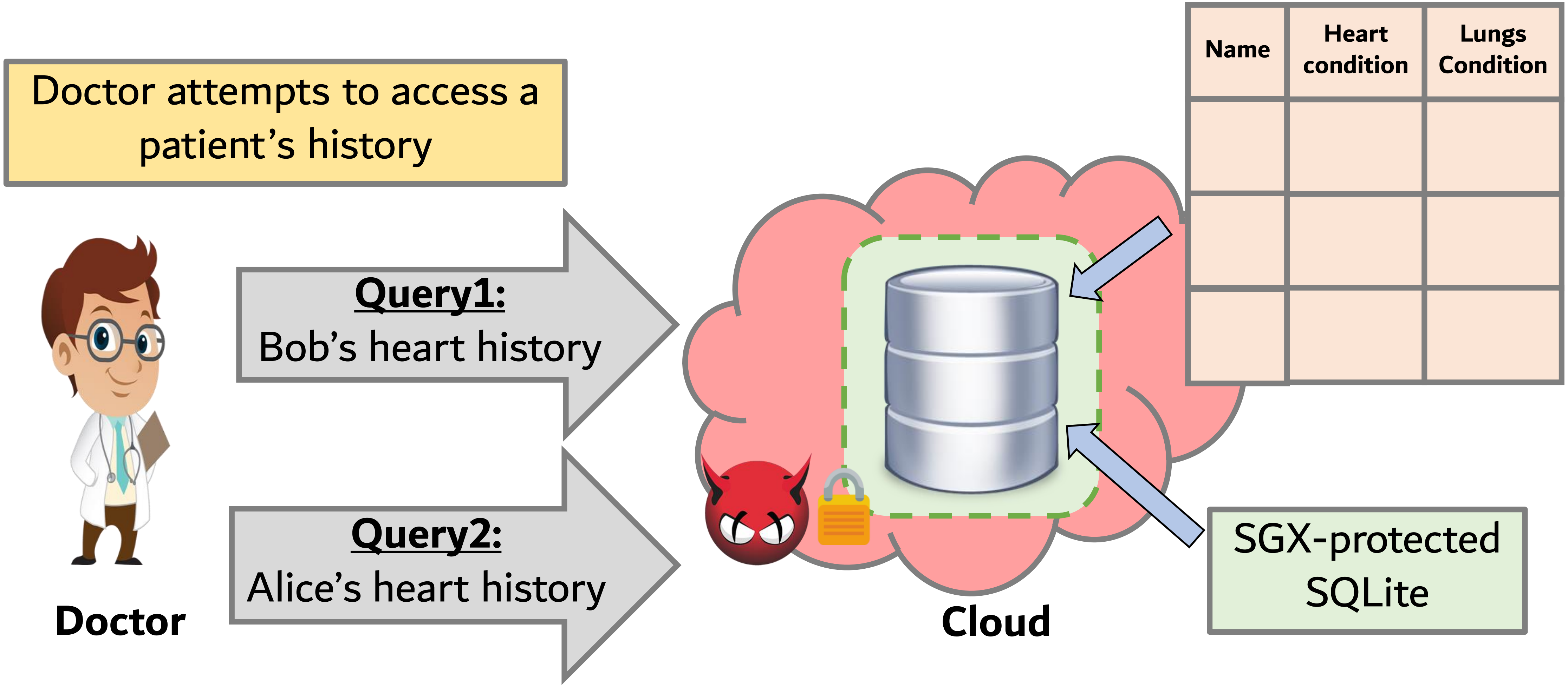
# Case Study: Attacking SQlite



# Case Study: Attacking SQlite



# Case Study: Attacking SQlite

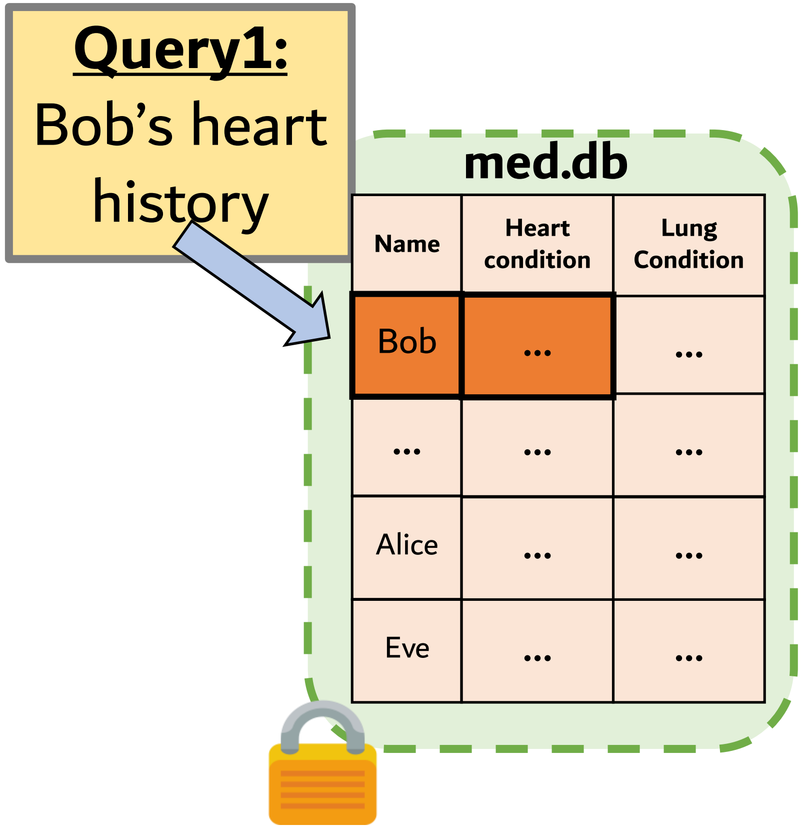


# What the attacker sees?

**med.db**

| Name | Heart condition | Lung Condition |
|---|---|---|
| Bob | … | … |
| … | … | … |
| Alice | … | … |
| Eve | … | … |

# What the attacker sees?



# What the attacker sees?

**Query1:**
Bob's heart history

**med.db**

| Name | Heart condition | Lung Condition |
|---|---|---|
| Bob | ... | ... |
| ... | ... | ... |
| Alice | ... | ... |
| Eve | ... | ... |

## Syscall Snooping Attack

1. open("med.db", ..);
2. pread64(...,4096,0);
3. pread64(...,4096,4096);
4. pread64(...,4096,32768);

# What the attacker sees?

**<u>Query1:</u>**
Bob’s heart history

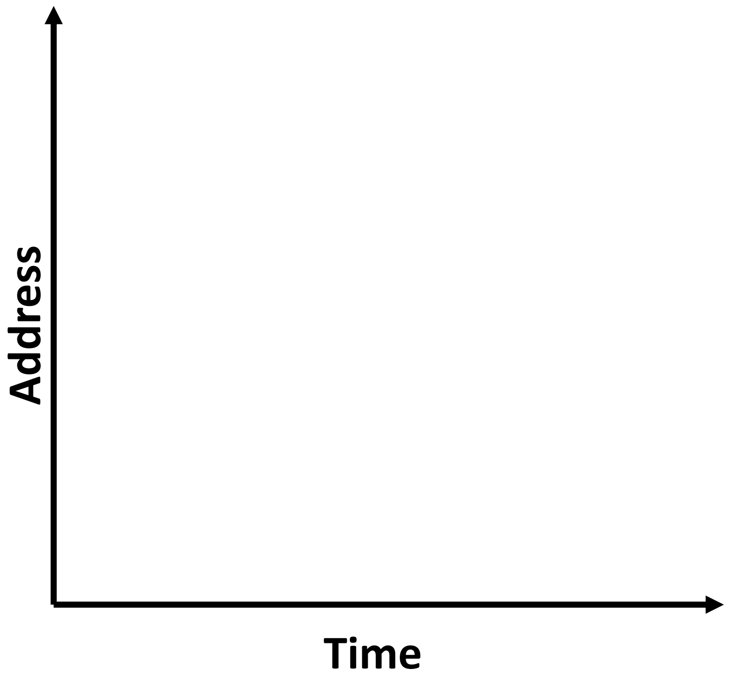
**med.db**

| Name | Heart condition | Lung Condition |
|---|---|---|
| Bob | … | … |
| … | … | … |
| Alice | … | … |
| Eve | … | … |

## Syscall Snooping Attack

1. open(”med.db”, ..);
2. pread64(…,4096,0);
3. pread64(…,4096,4096);
4. pread64(…,4096,32768);

## Page Table Attack



# What the attacker sees?

**Query1:**
Bob's heart history

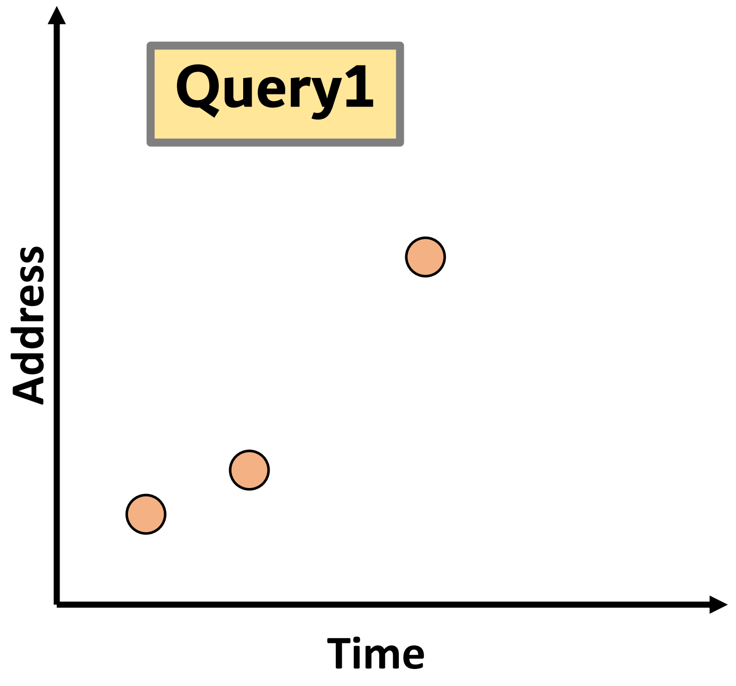
**med.db**

| Name | Heart condition | Lung Condition |
|---|---|---|
| Bob | ... | ... |
| ... | ... | ... |
| Alice | ... | ... |
| Eve | ... | ... |

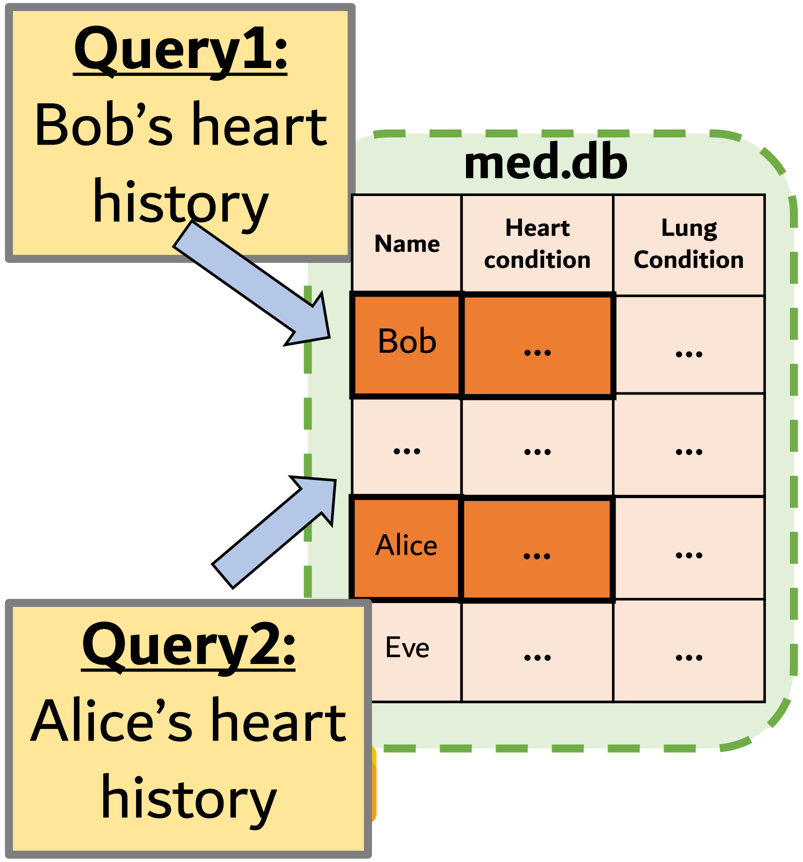
## Syscall Snooping Attack

1. open("med.db", ..);
2. pread64(...,4096,0);
3. pread64(...,4096,4096);
4. pread64(...,4096,32768);

## Page Table Attack



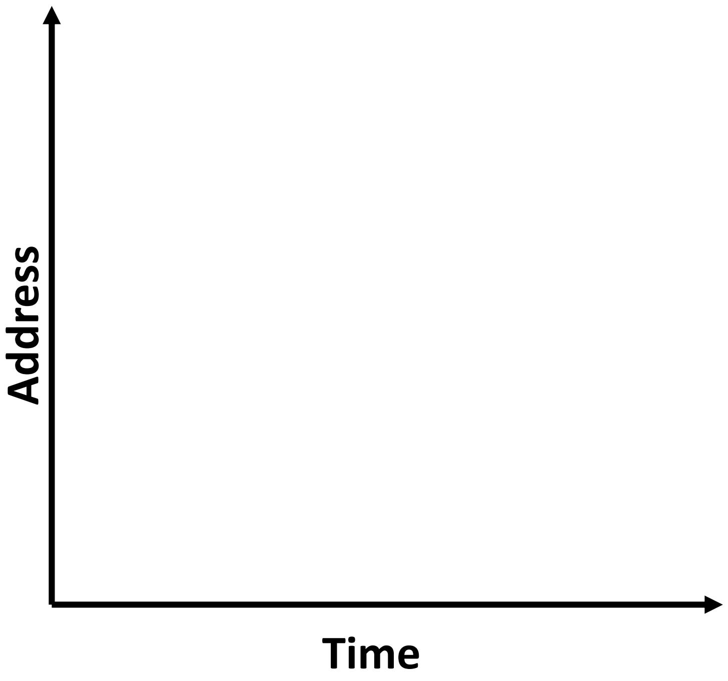
# What the attacker sees?



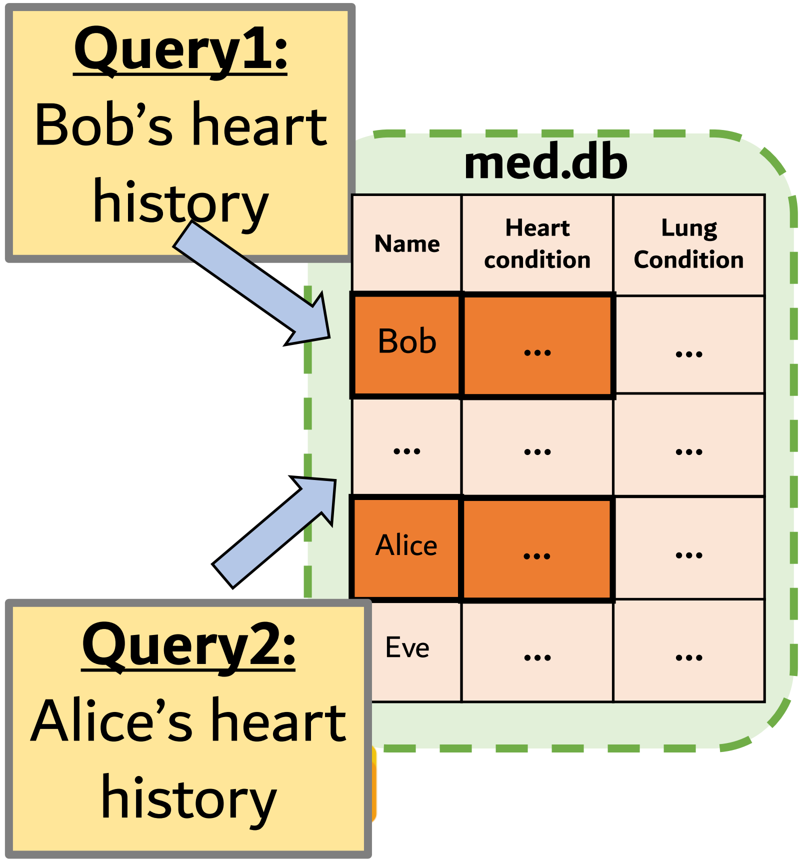

## Syscall Snooping Attack

1. open("med.db", ..);
2. pread64(…,4096,0);
3. pread64(…,4096,4096);
4. pread64(…,4096,32768);

## Page Table Attack



# What the attacker sees?

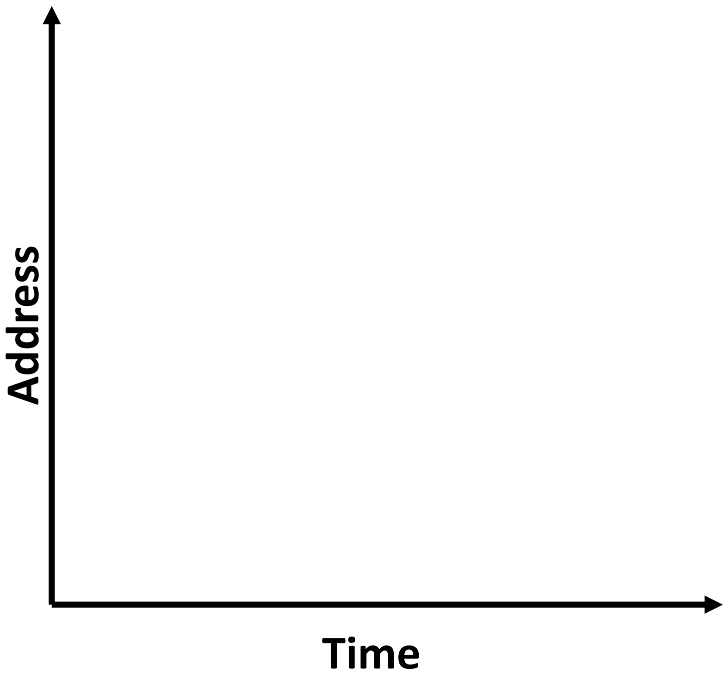



## Syscall Snooping Attack

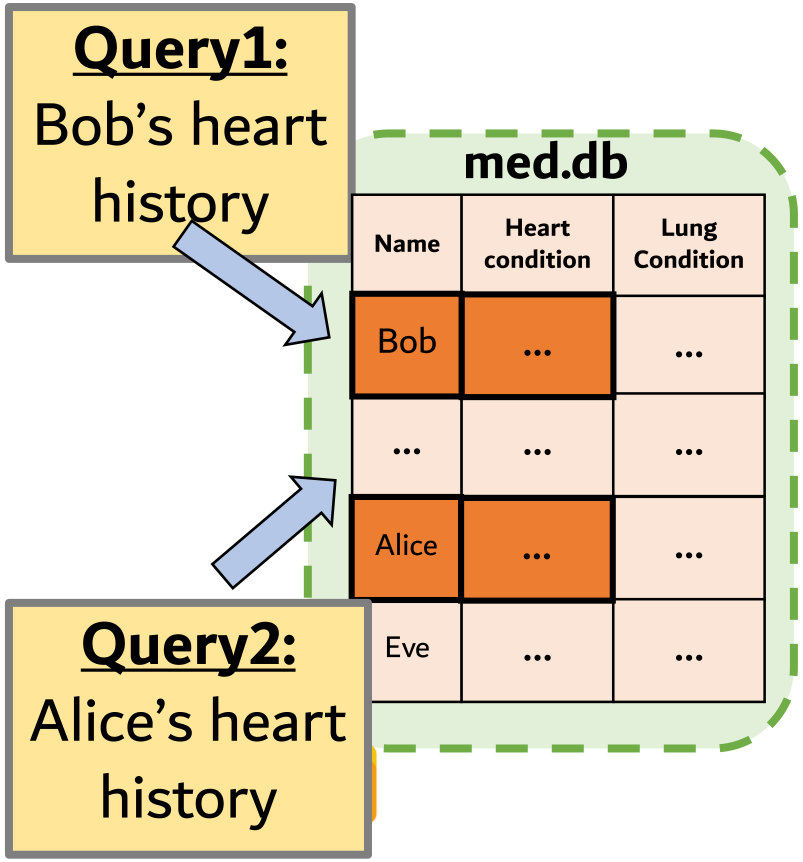
1. open("med.db", ..);
2. pread64(...,4096,0);
3. pread64(...,4096,4096);
4. pread64(...,4096,32768);

1. open("med.db", ..);
2. pread64(...,4096,0);
3. pread64(...,4096,4096);
4. pread64(...,4096,40960);

## Page Table Attack



# What the attacker sees?

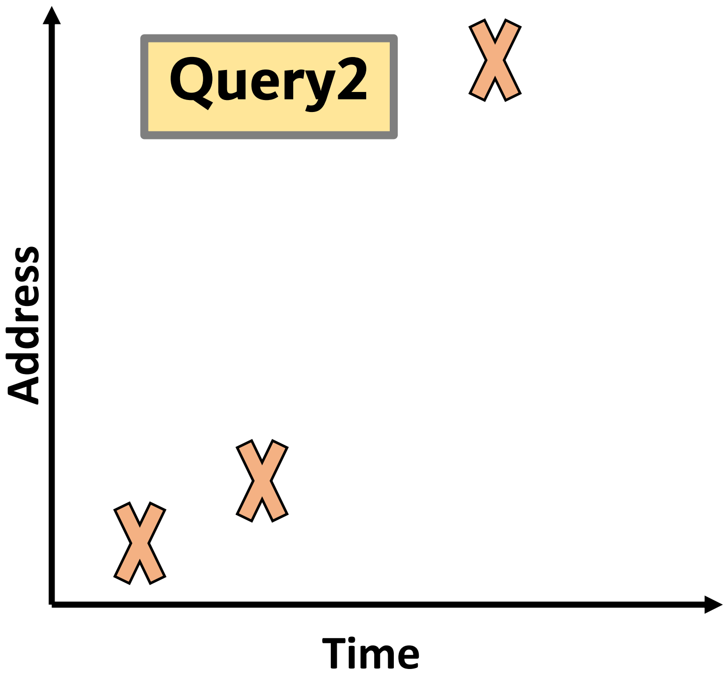



## Syscall Snooping Attack

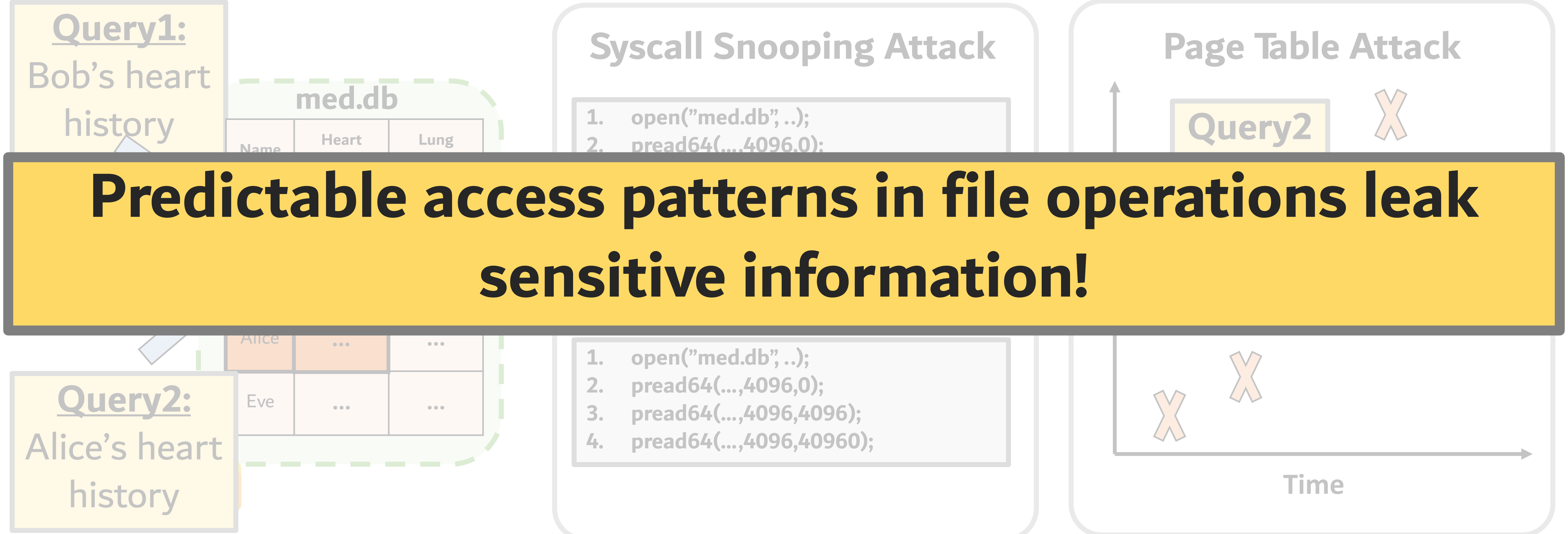
```
1. open("med.db", ..);
2. pread64(...,4096,0);
3. pread64(...,4096,4096);
4. pread64(...,4096,32768);
```

```
1. open("med.db", ..);
2. pread64(...,4096,0);
3. pread64(...,4096,4096);
4. pread64(...,4096,40960);
```

## Page Table Attack



# What the attacker sees?




**Predictable access patterns in file operations leak sensitive information!**

# What should we do?

# What should we do?

**Masking individual memory side-channels is risky**

# What should we do?

**Masking individual memory side-channels is risky**

**Memory side-channels rely on predictable access patterns**

# What should we do?

**Masking individual memory side-channels is risky**

**Memory side-channels rely on predictable access patterns**

**How to provide strong protection despite memory traces?**

# What should we do?

**Masking individual memory side-channels is risky**

**Memory side-channels rely on predictable access patterns**

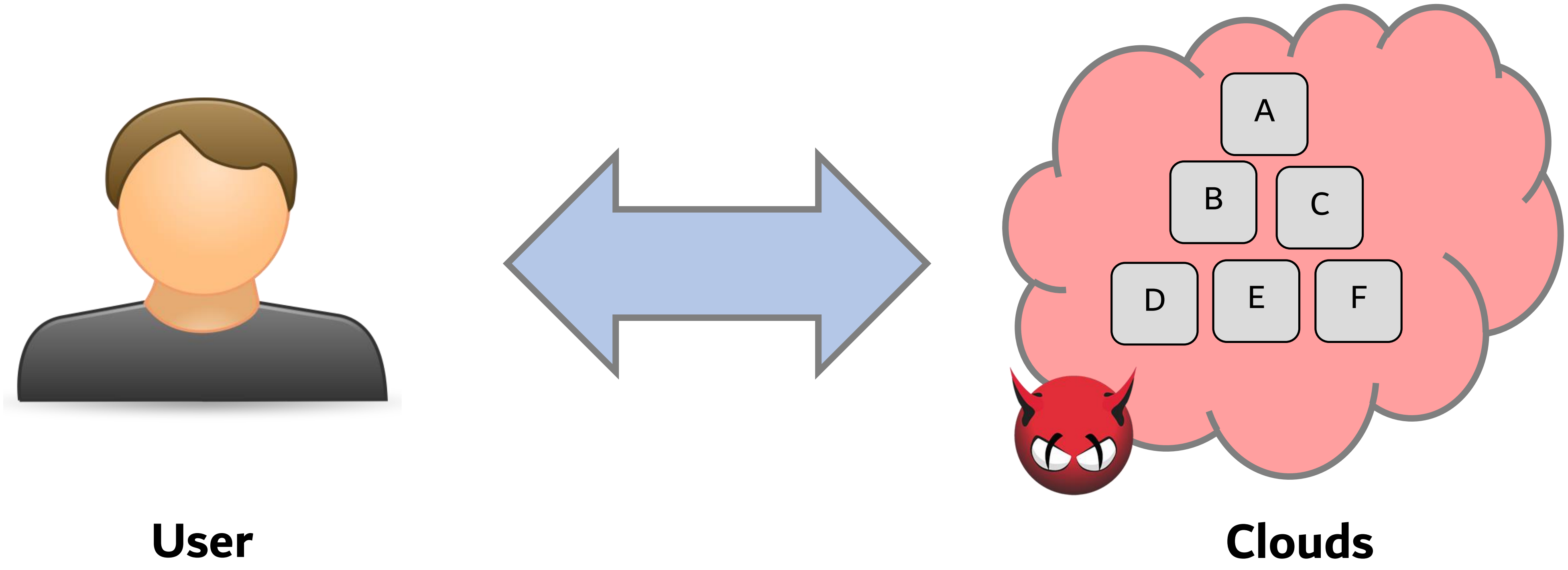
**How to provide strong protection despite memory traces?**

**Oblivious RAM is one possible solution to this problem**

# Oblivious RAM

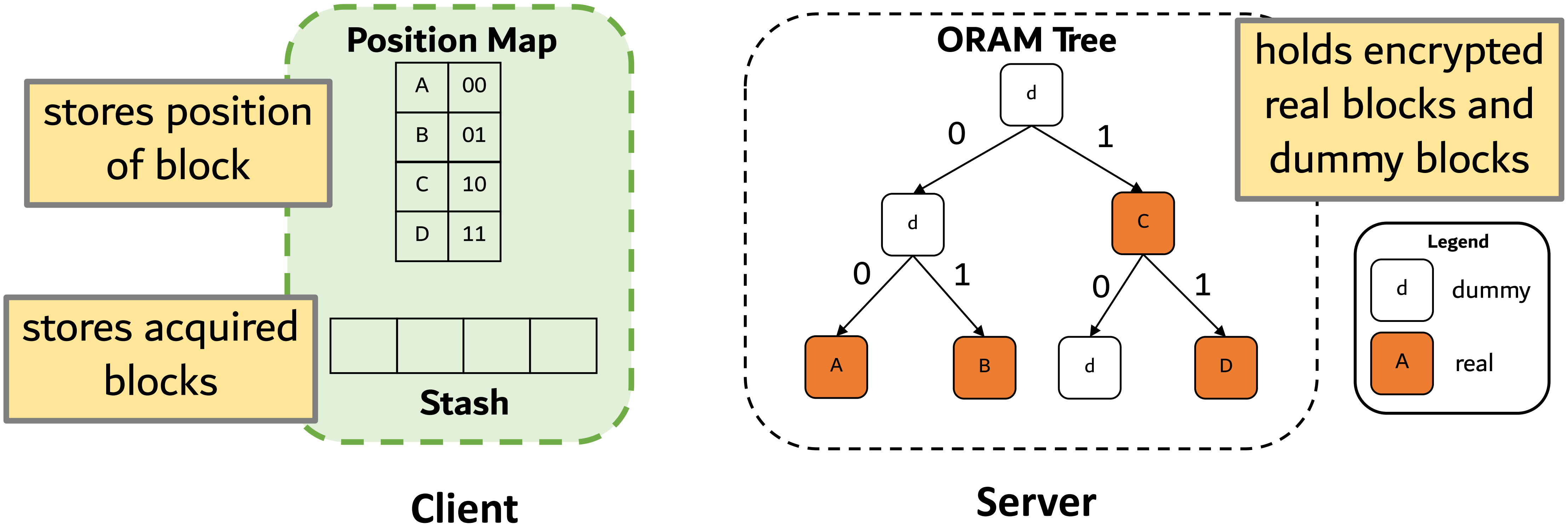
**<u>User's goal:</u>**
Securely access data stored in the cloud

**<u>Attacker's goal:</u>**
Figure out what data-block is being accessed



# Path ORAM

Improved variant of Oblivious RAM [Stephanov et. al, CCS12]



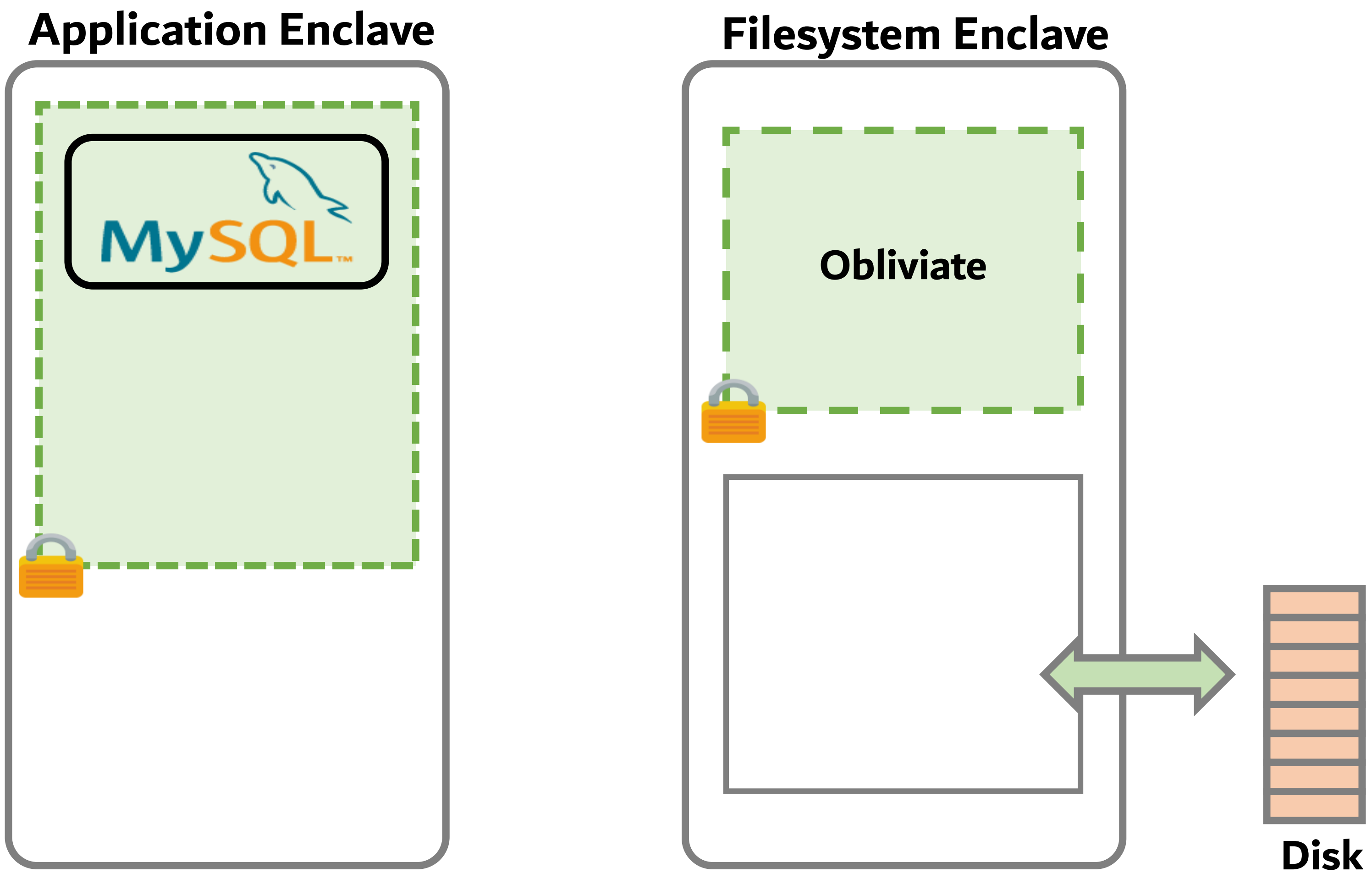

Operating
System


Operating
System

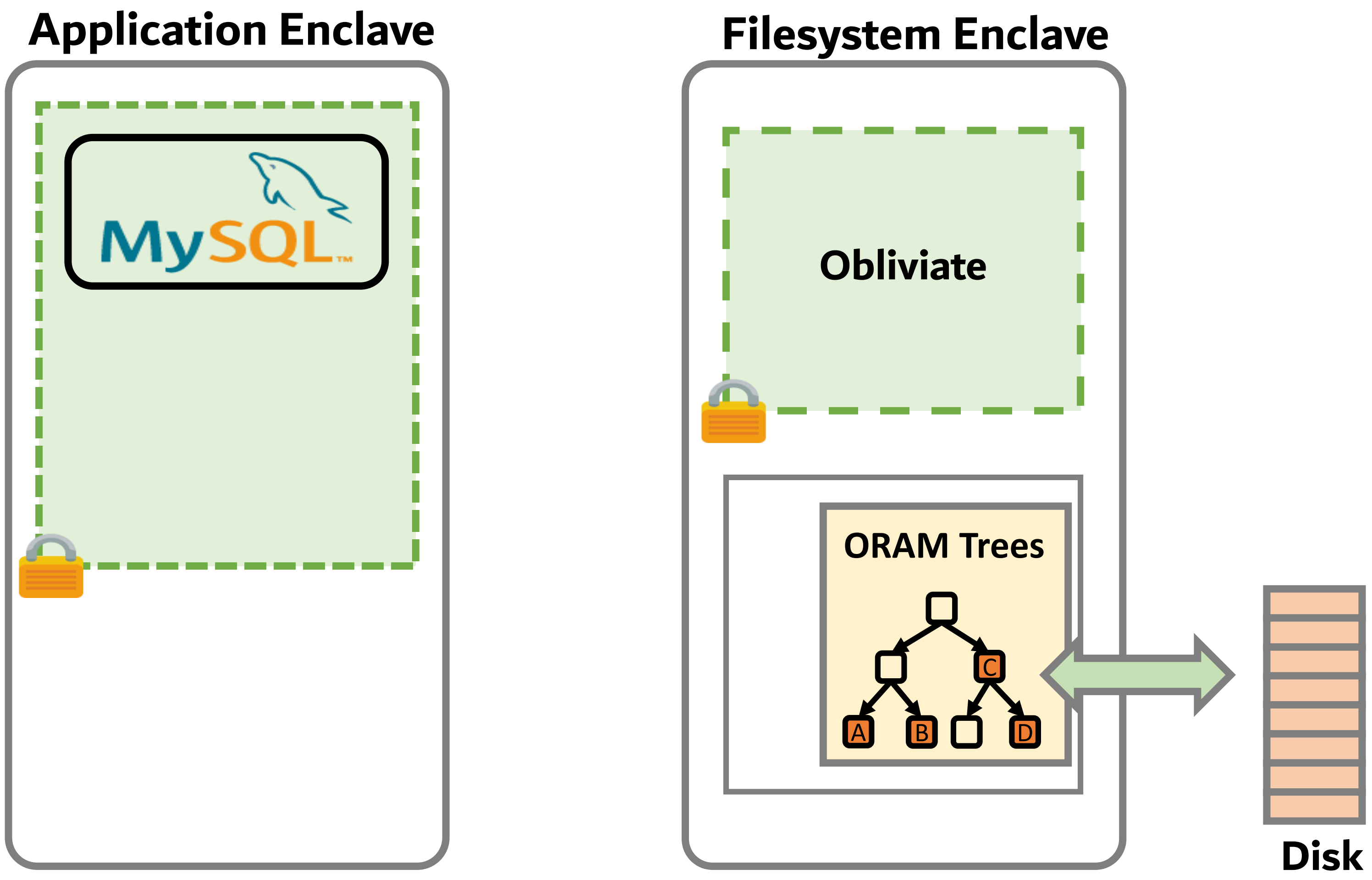

OBLIVIATE!
Operating System


OBLIVIATE!
Operating System

# Obliviate: memory charm against the OS ☺

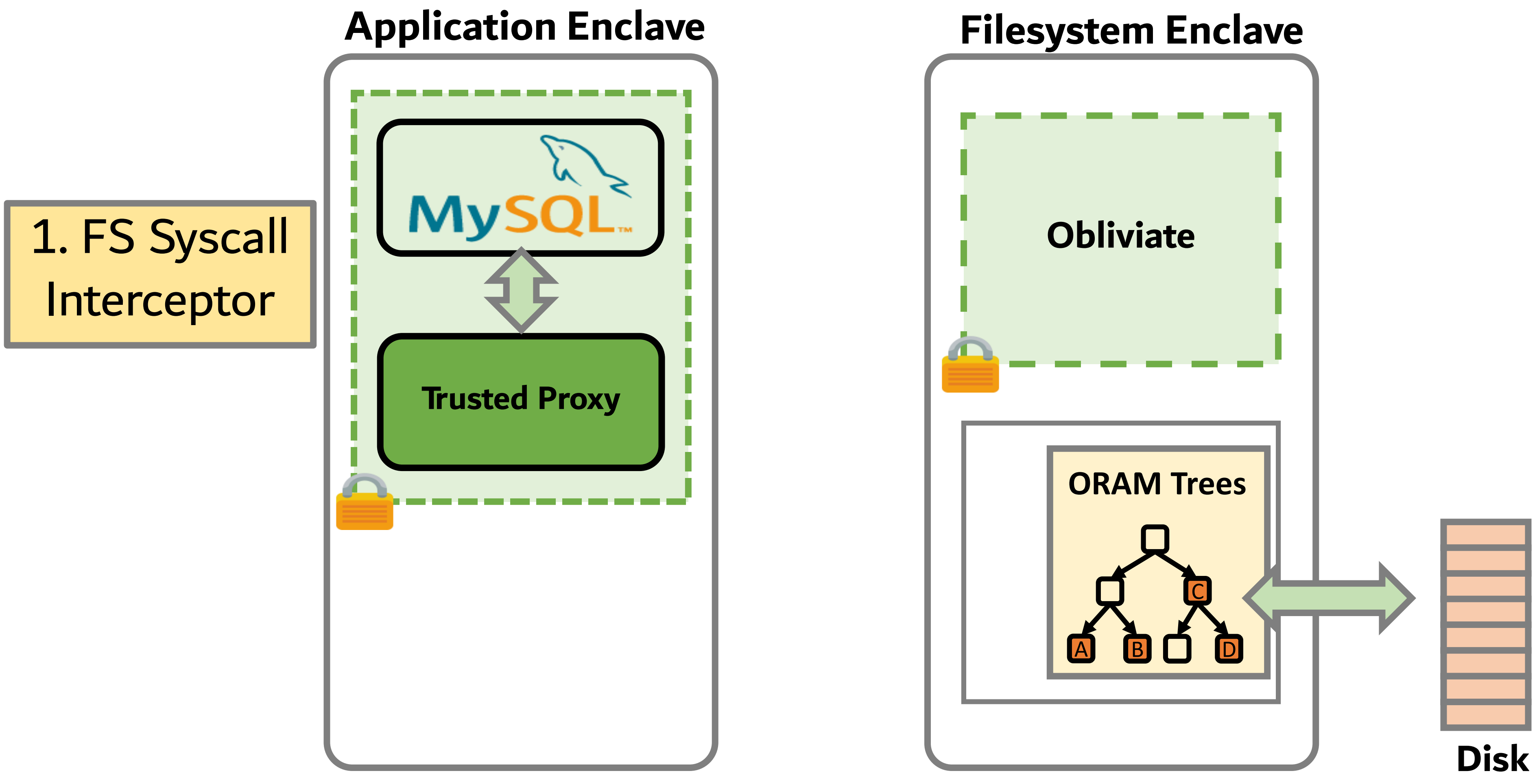


# Obliviate: memory charm against the OS ☺

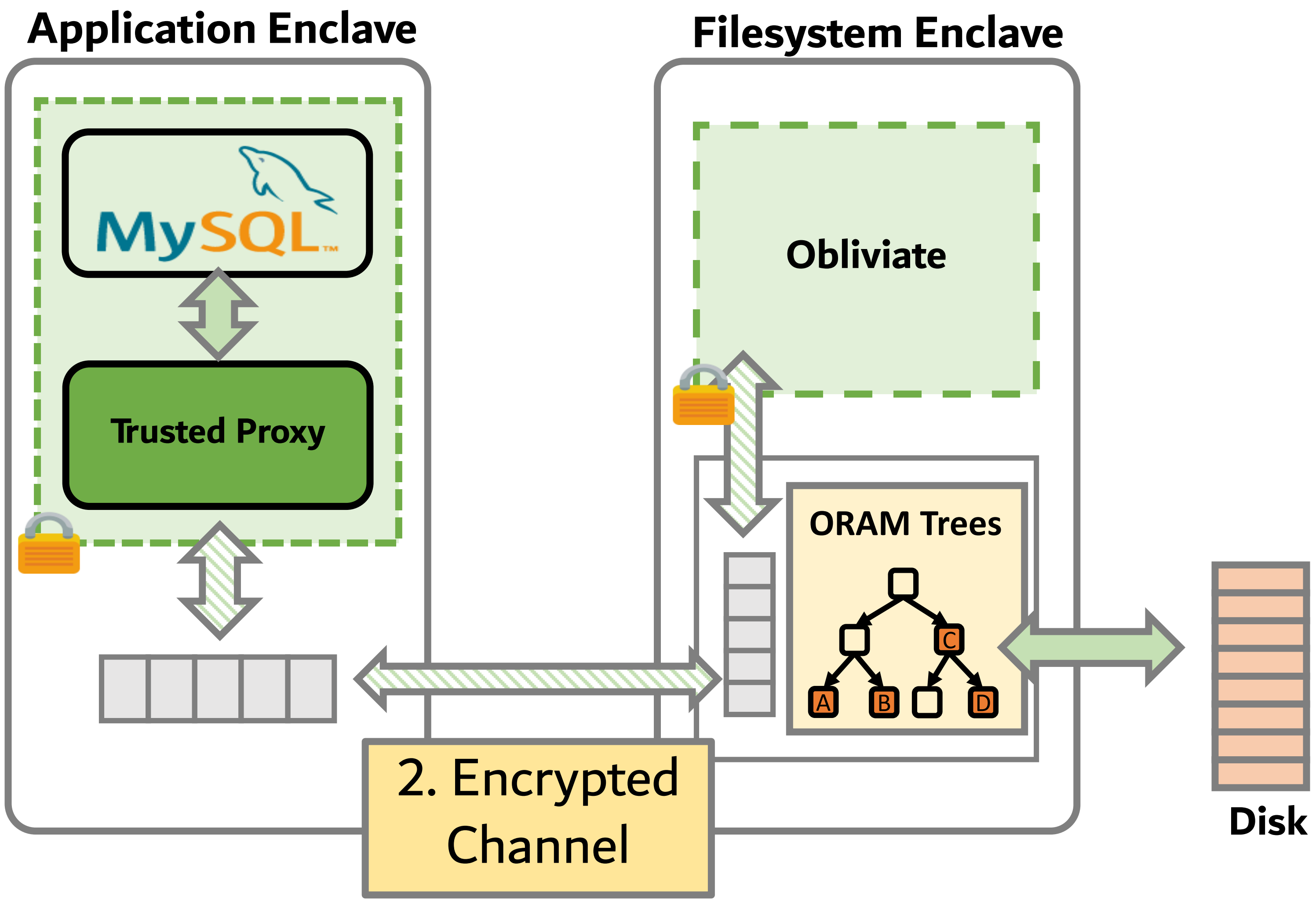



(Init) load all files into ORAM Tree(s)

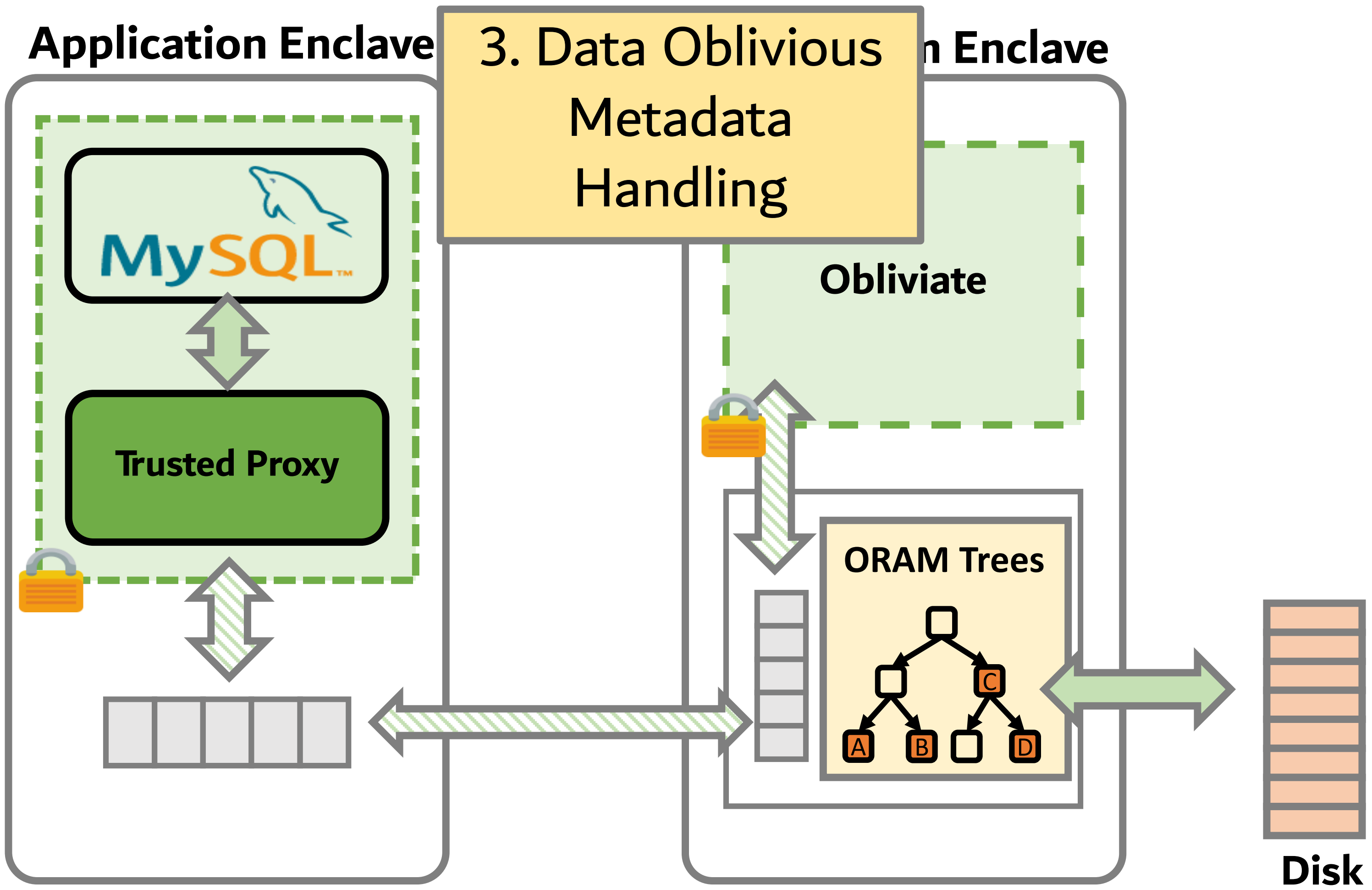
# Obliviate: memory charm against the OS ☺



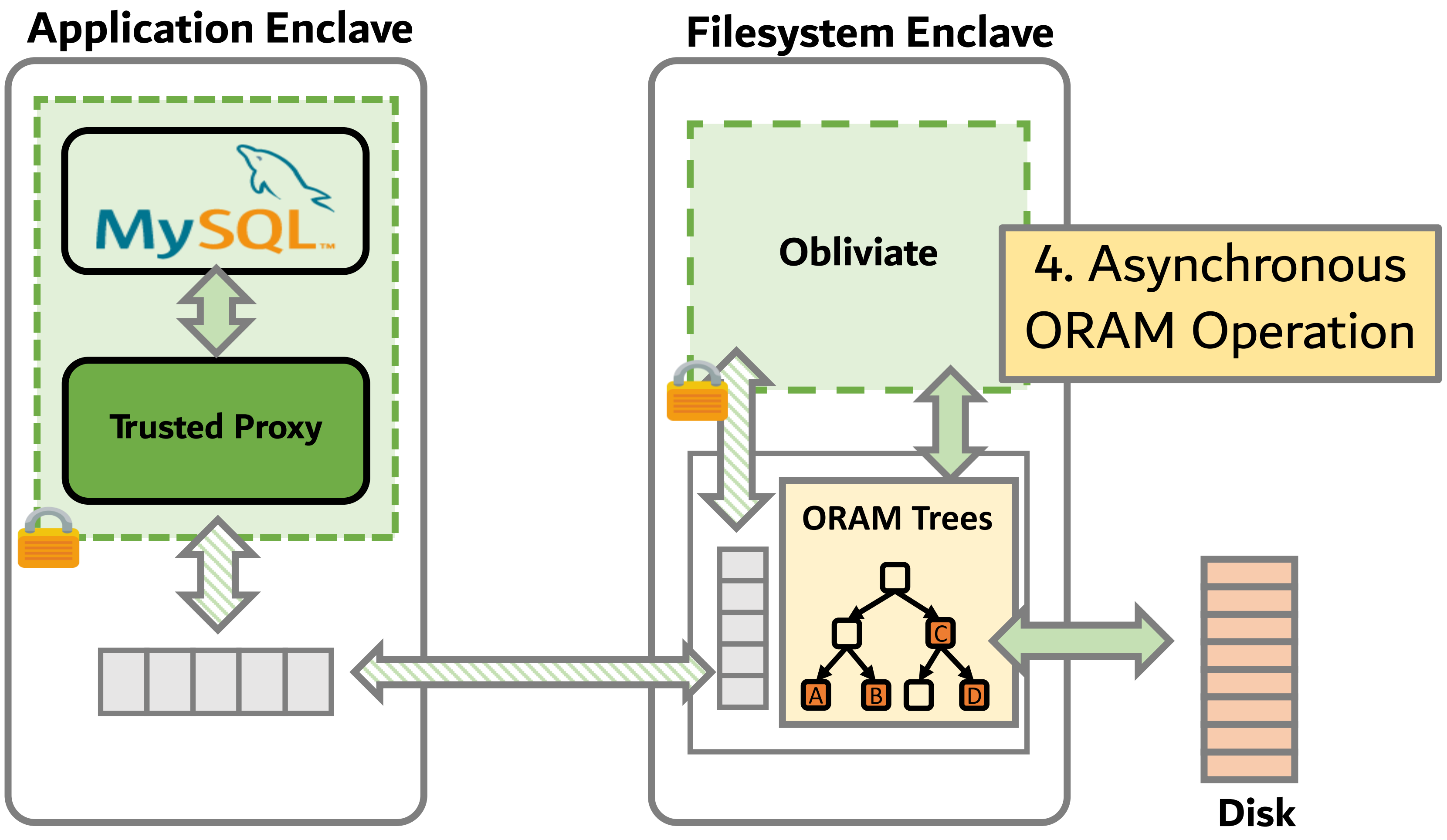
# Obliviate: memory charm against the OS ☺



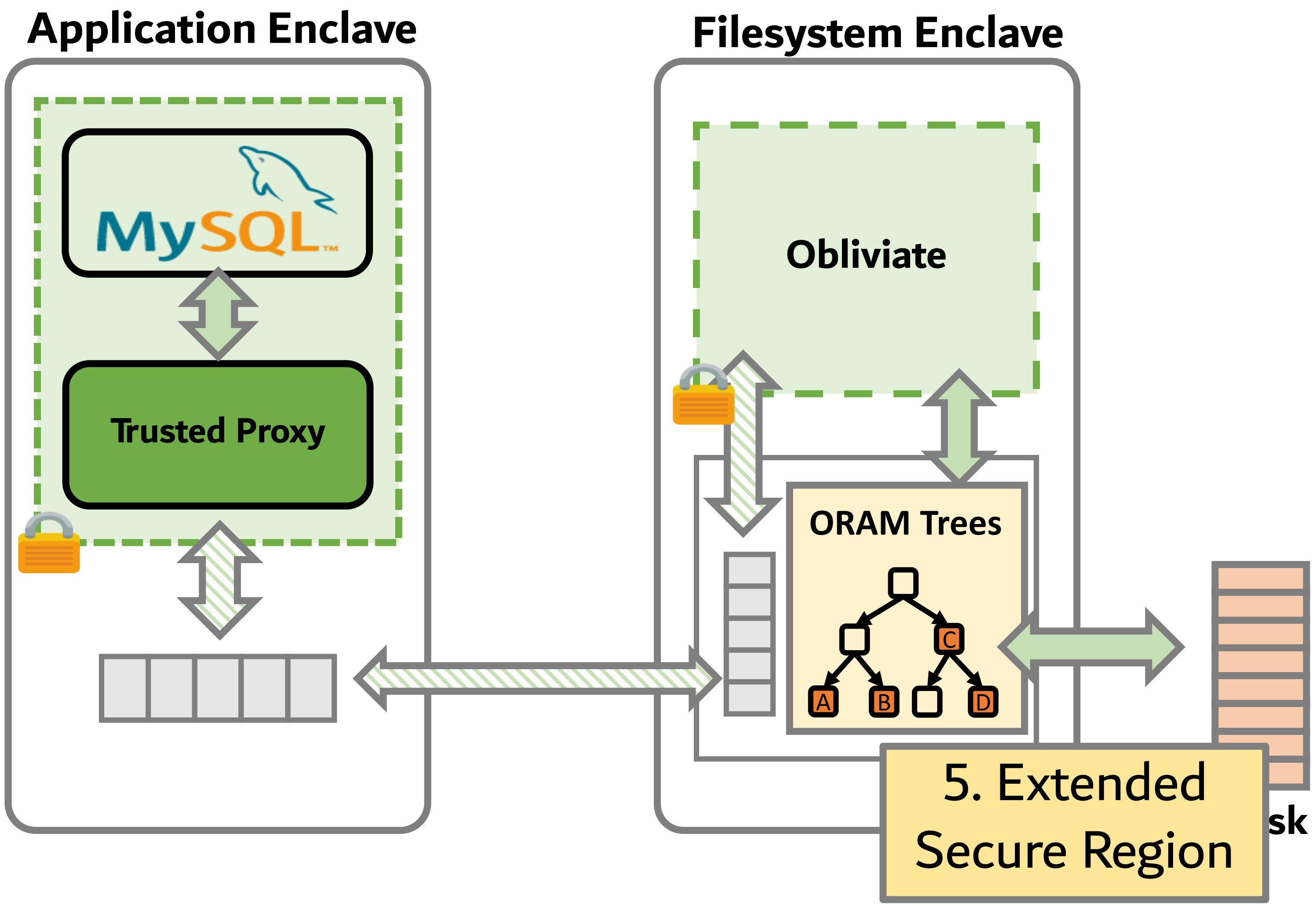
# Obliviate: memory charm against the OS ☺



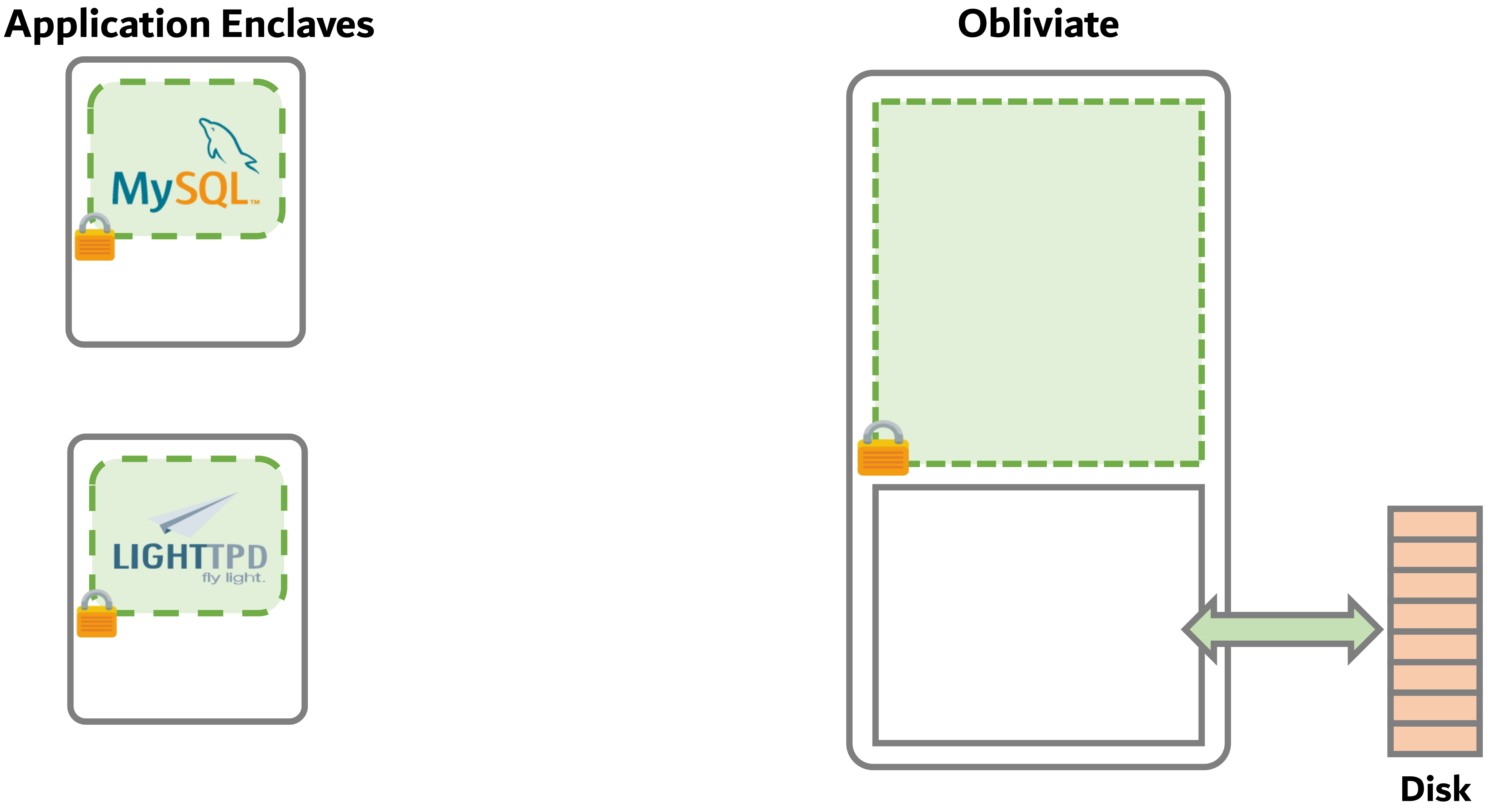
# Obliviate: memory charm against the OS ☺



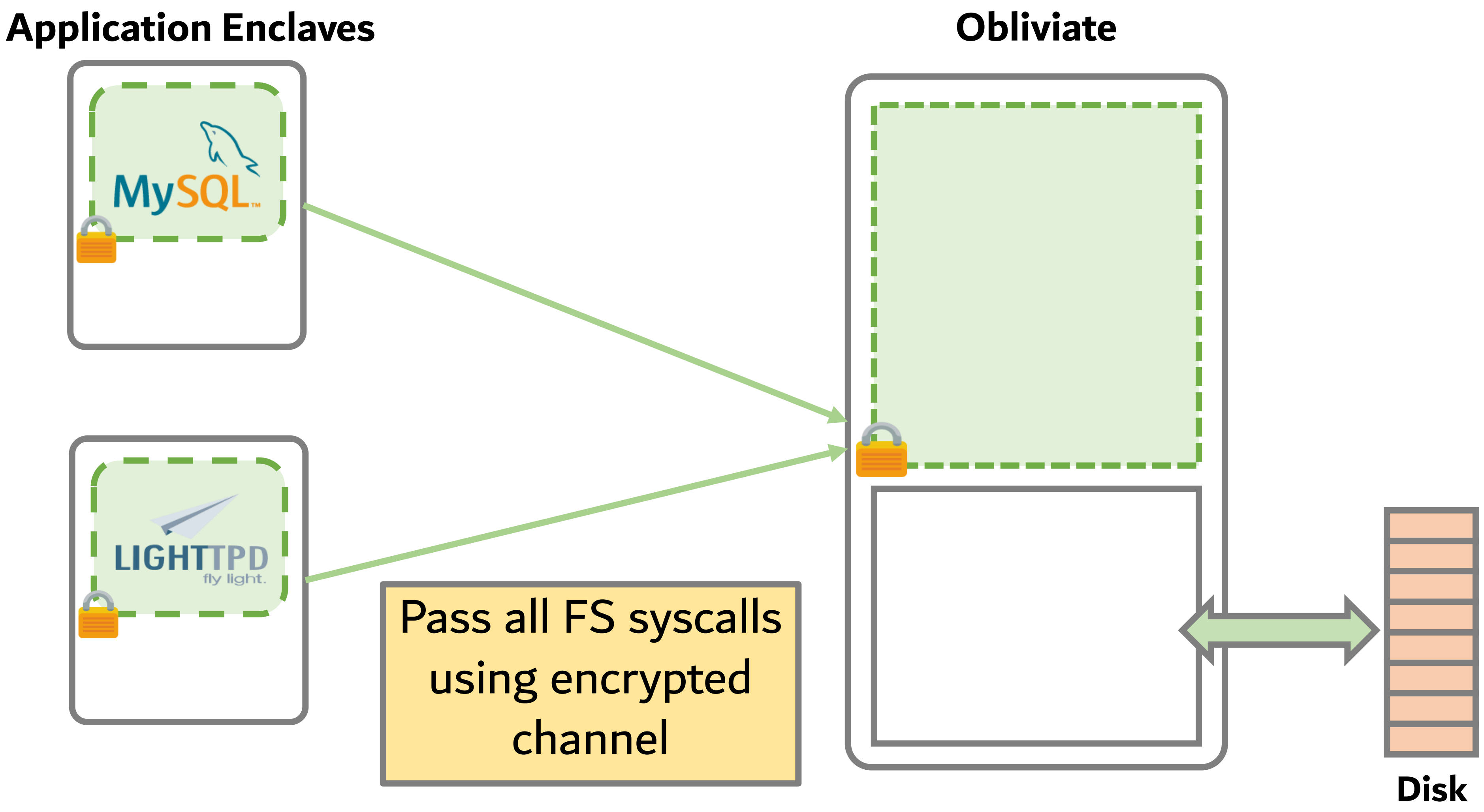
# Obliviate: memory charm against the OS ☺



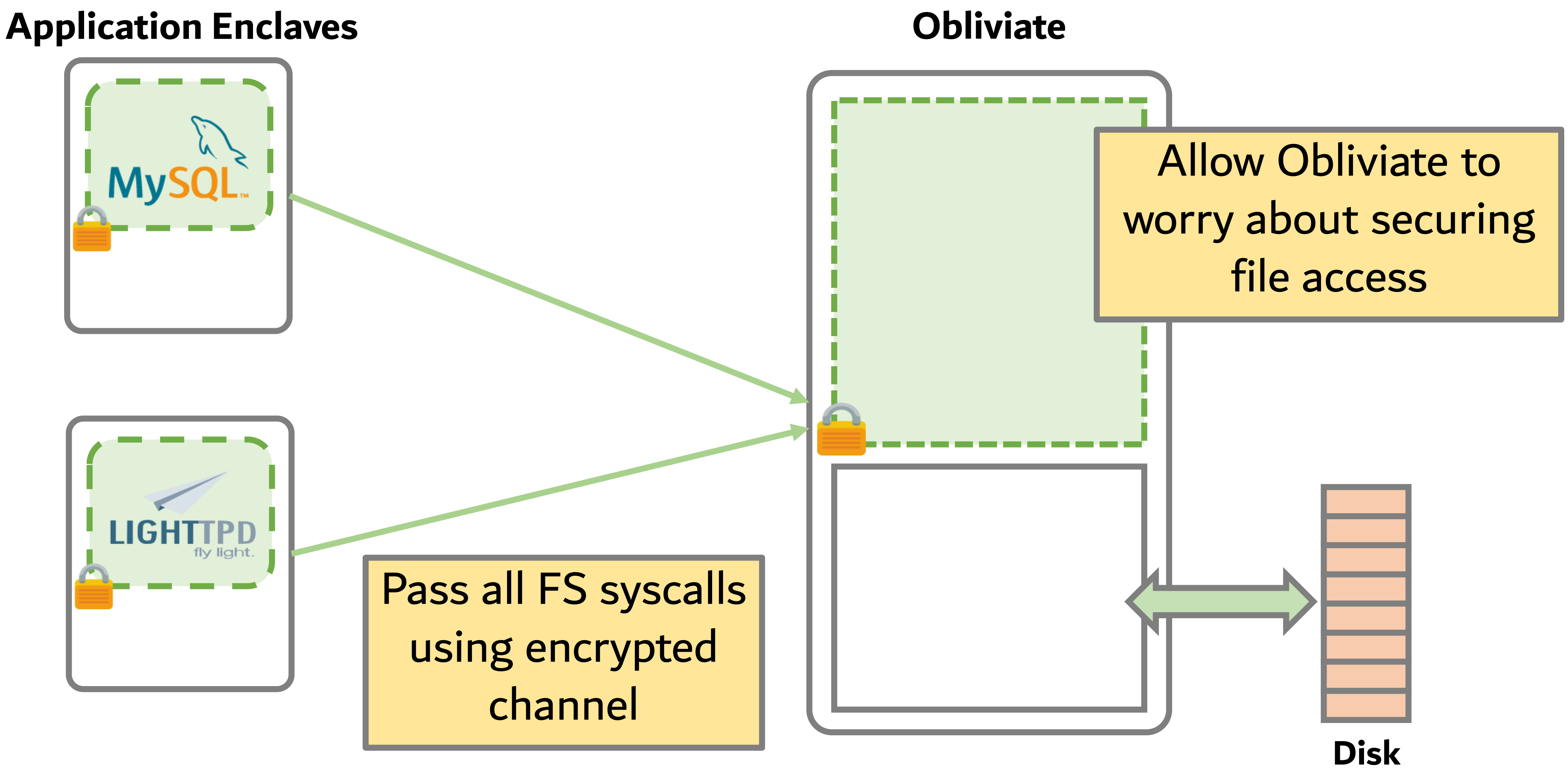
# Decoupling file system support



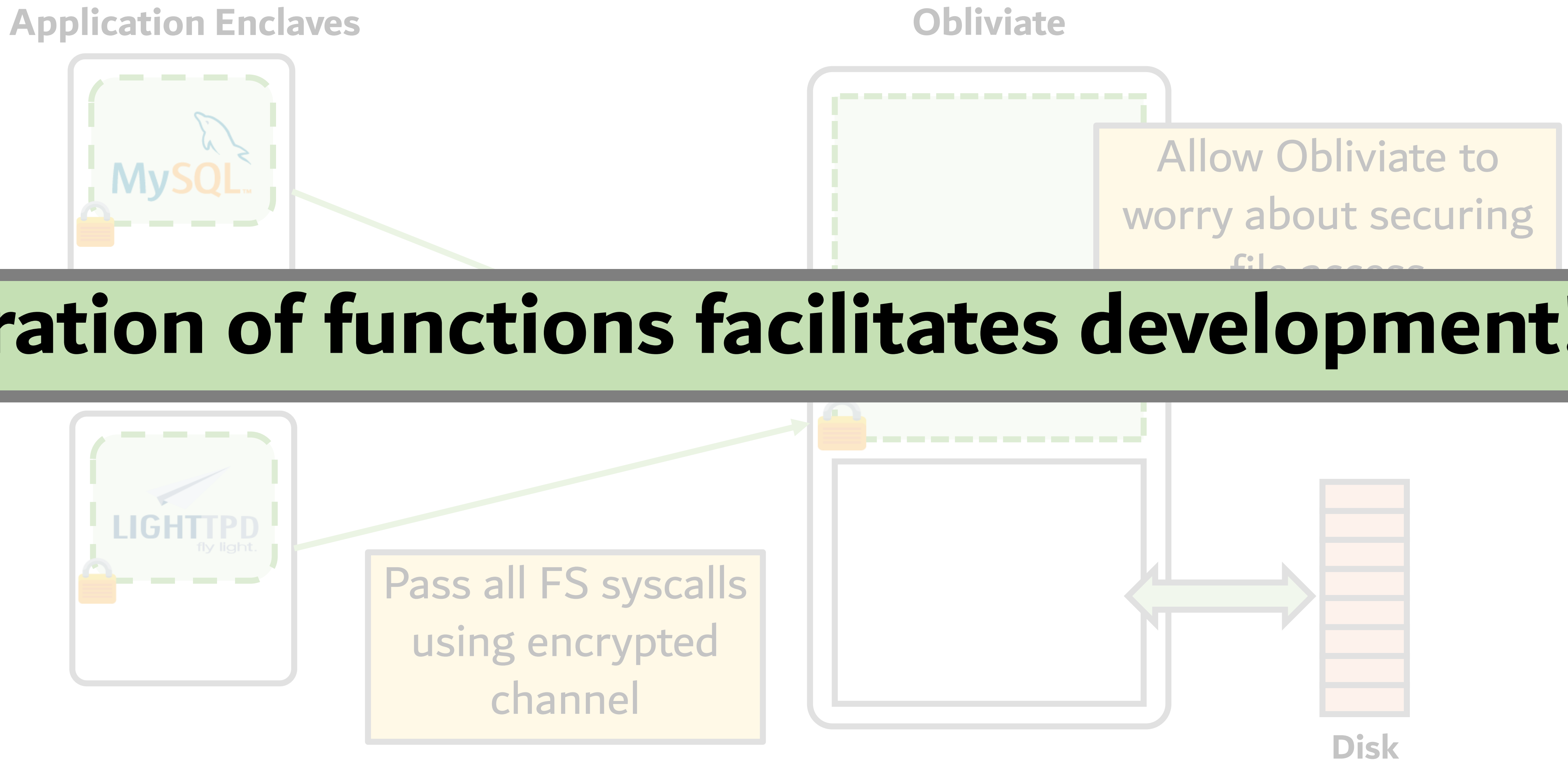
# Decoupling file system support



# Decoupling file system support

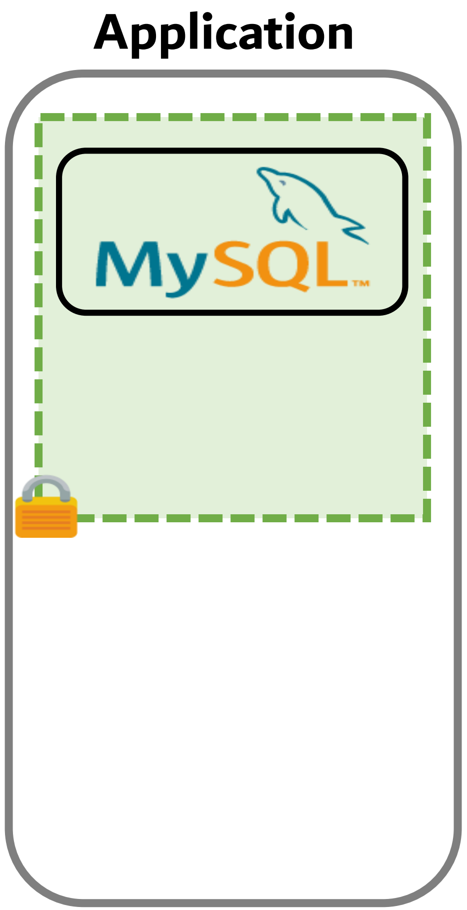


# Decoupling file system support

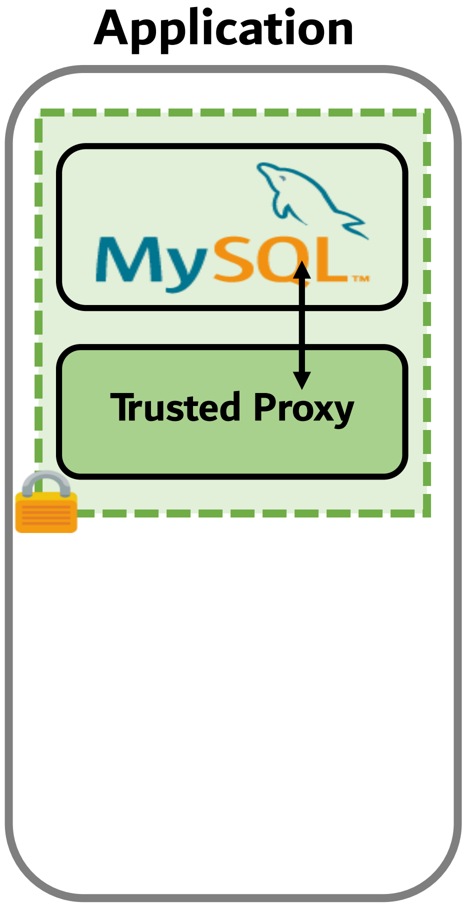



**Separation of functions facilitates development!**

# Legacy application support

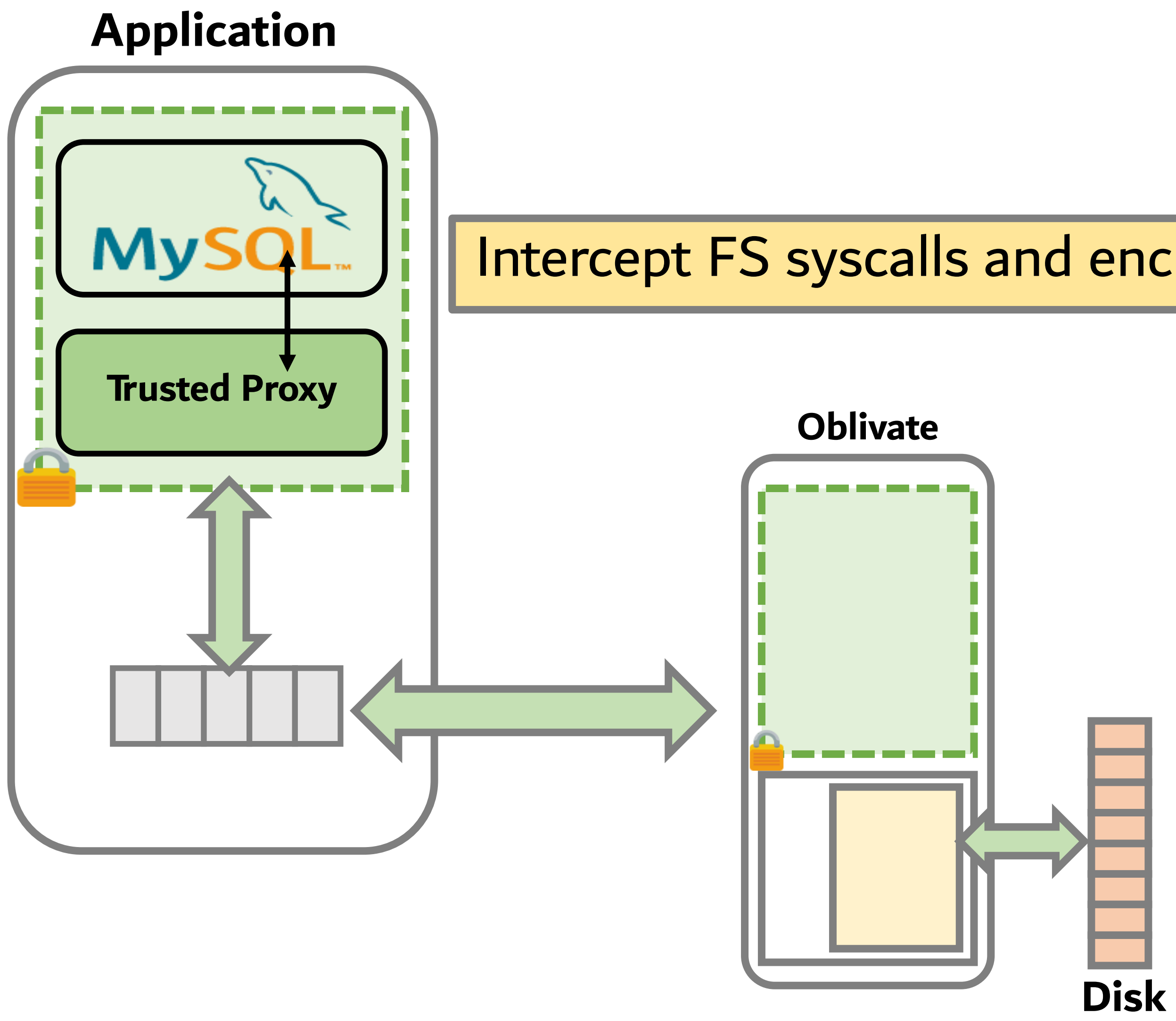


# Legacy application support




Intercept FS syscalls and encrypt

# Legacy application support

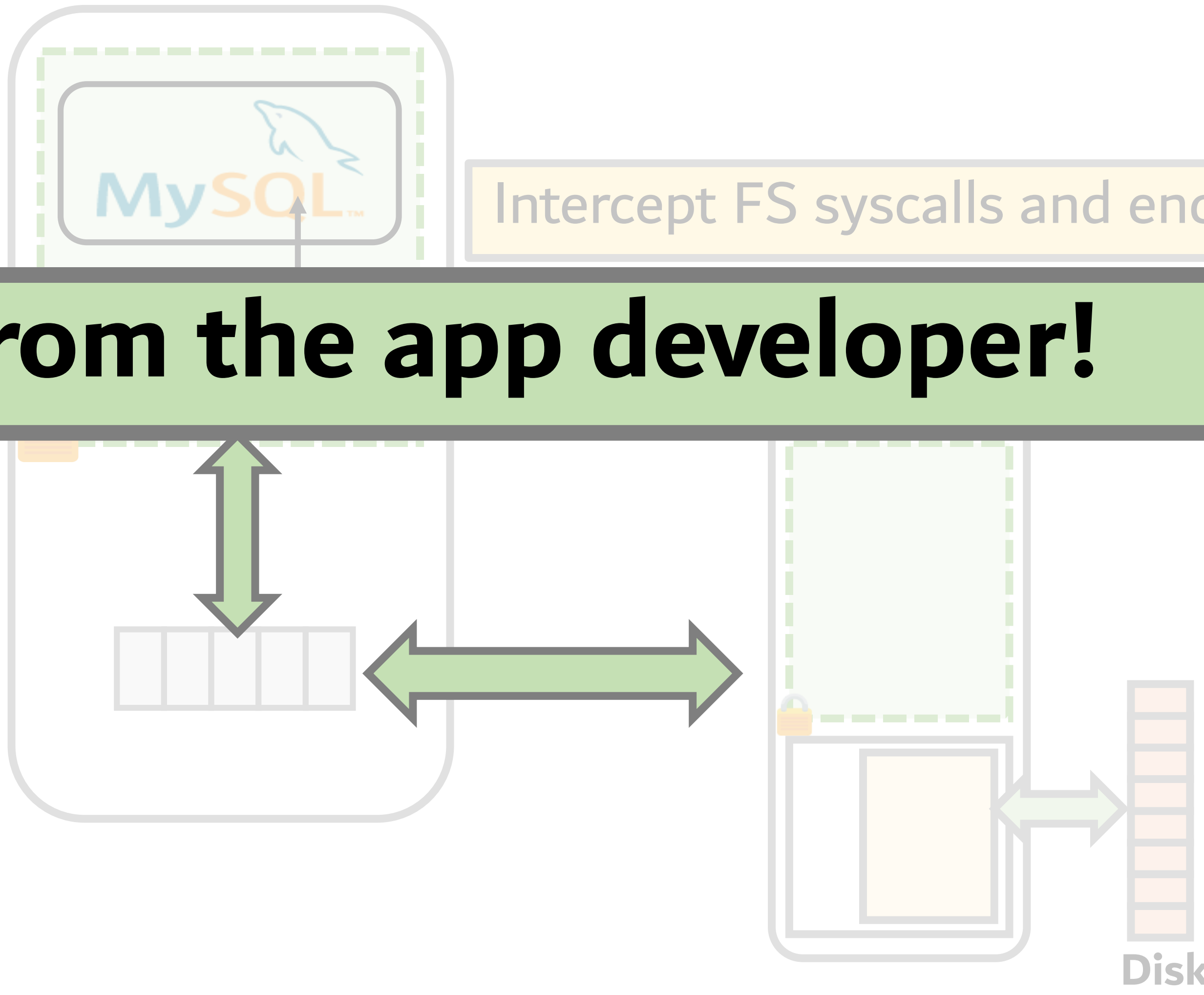



Intercept FS syscalls and encrypt

Exit-less message queue
(SCONE [OSDI16], ELEOS [EuroSys17])

# Legacy application support

Application

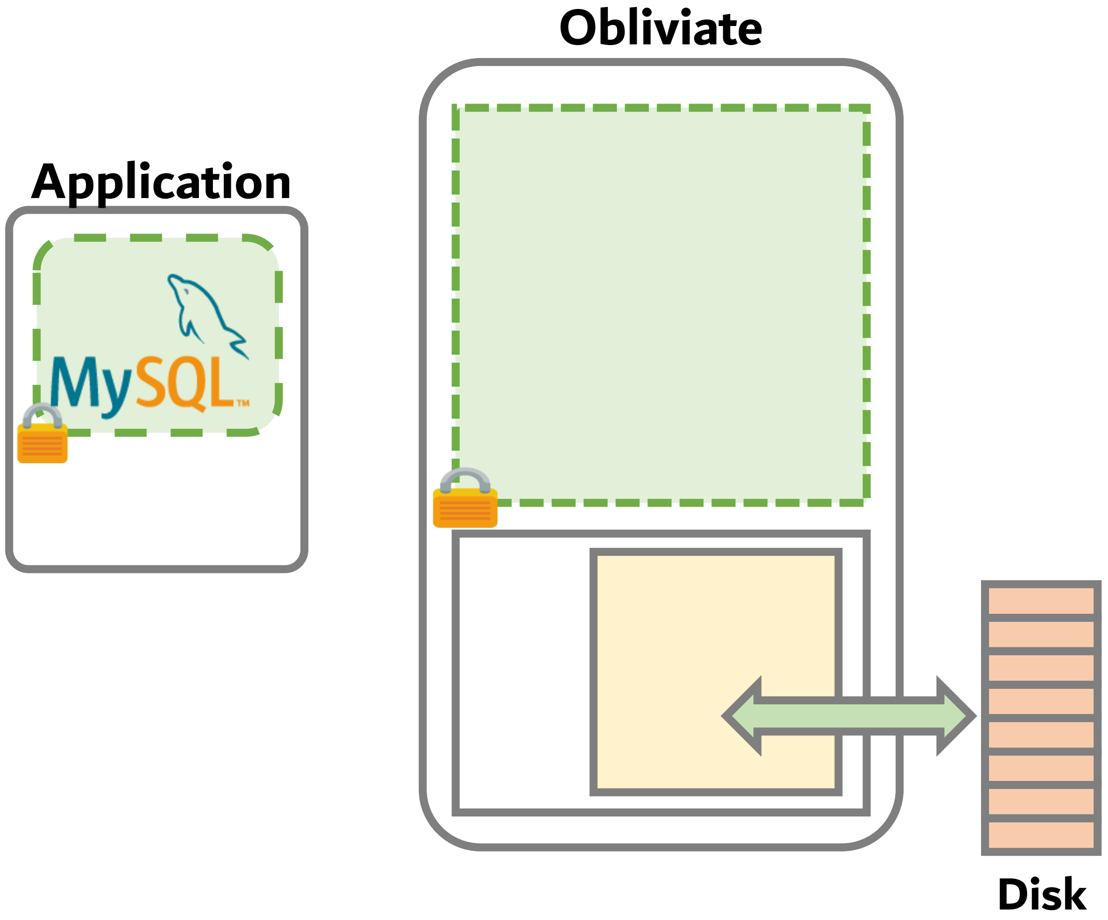



Intercept FS syscalls and encrypt

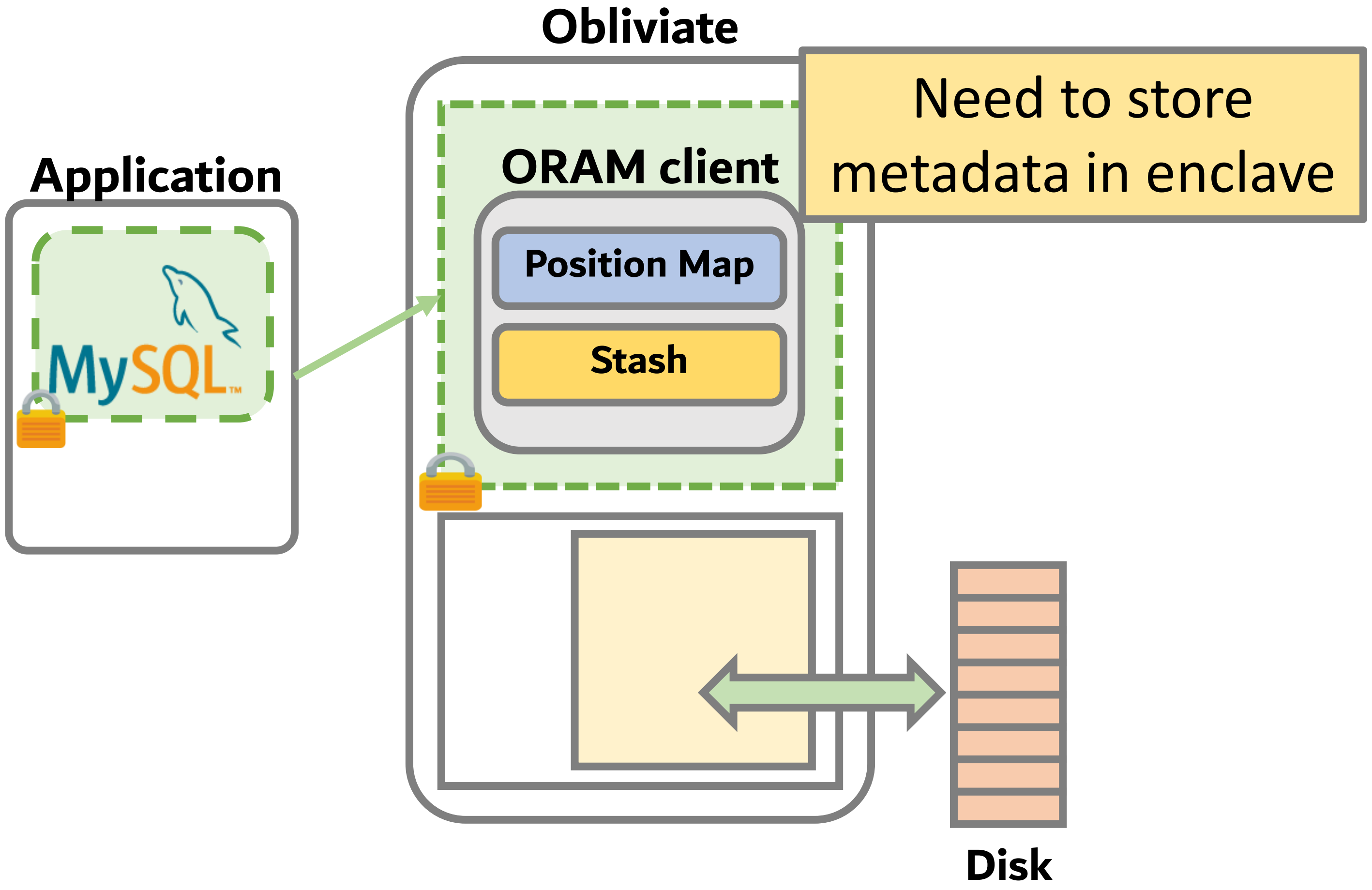
**No changes from the app developer!**

(SCONE [OSDI16], ELEOS [EuroSys17])

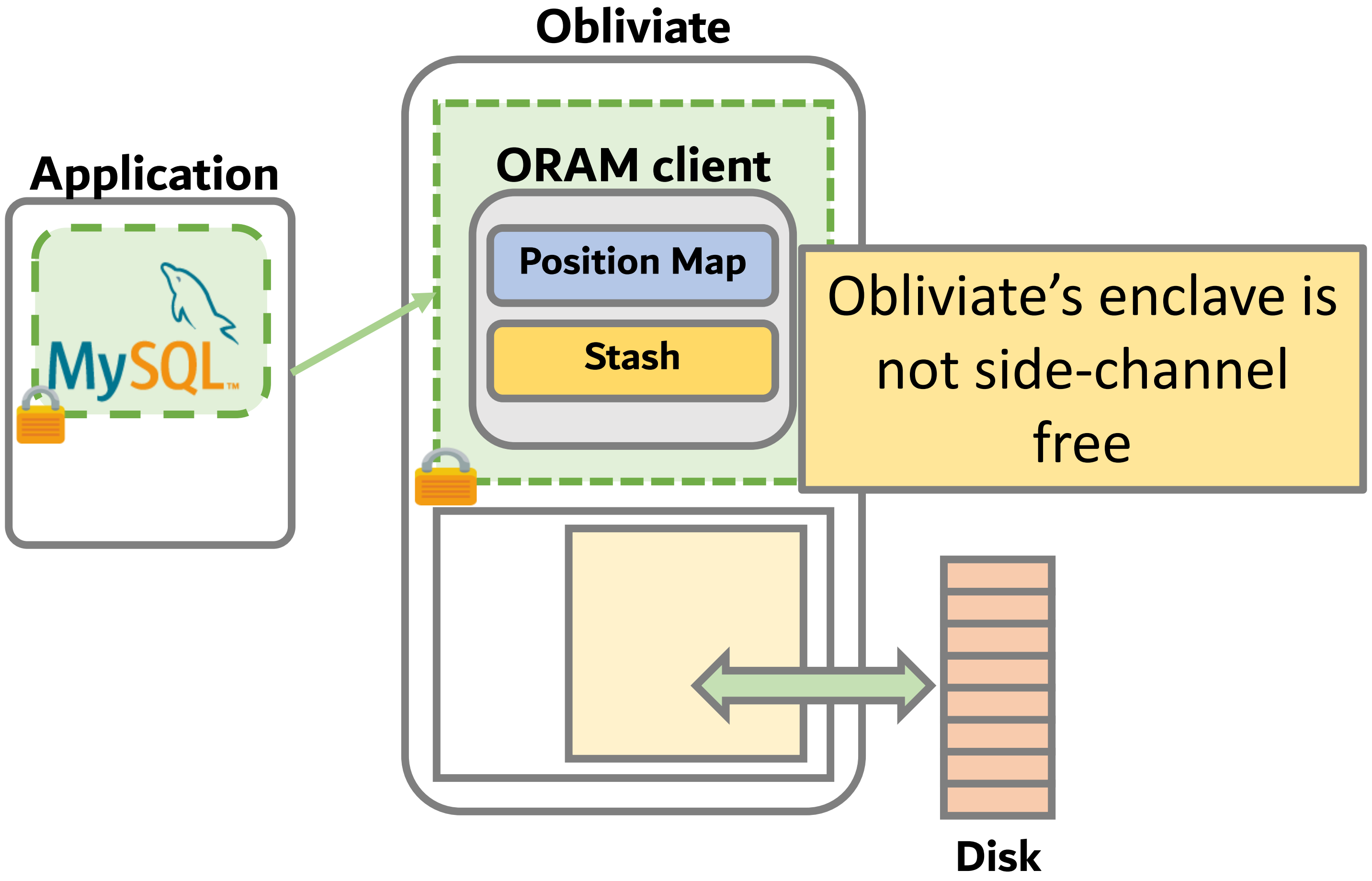
# Securing ORAM



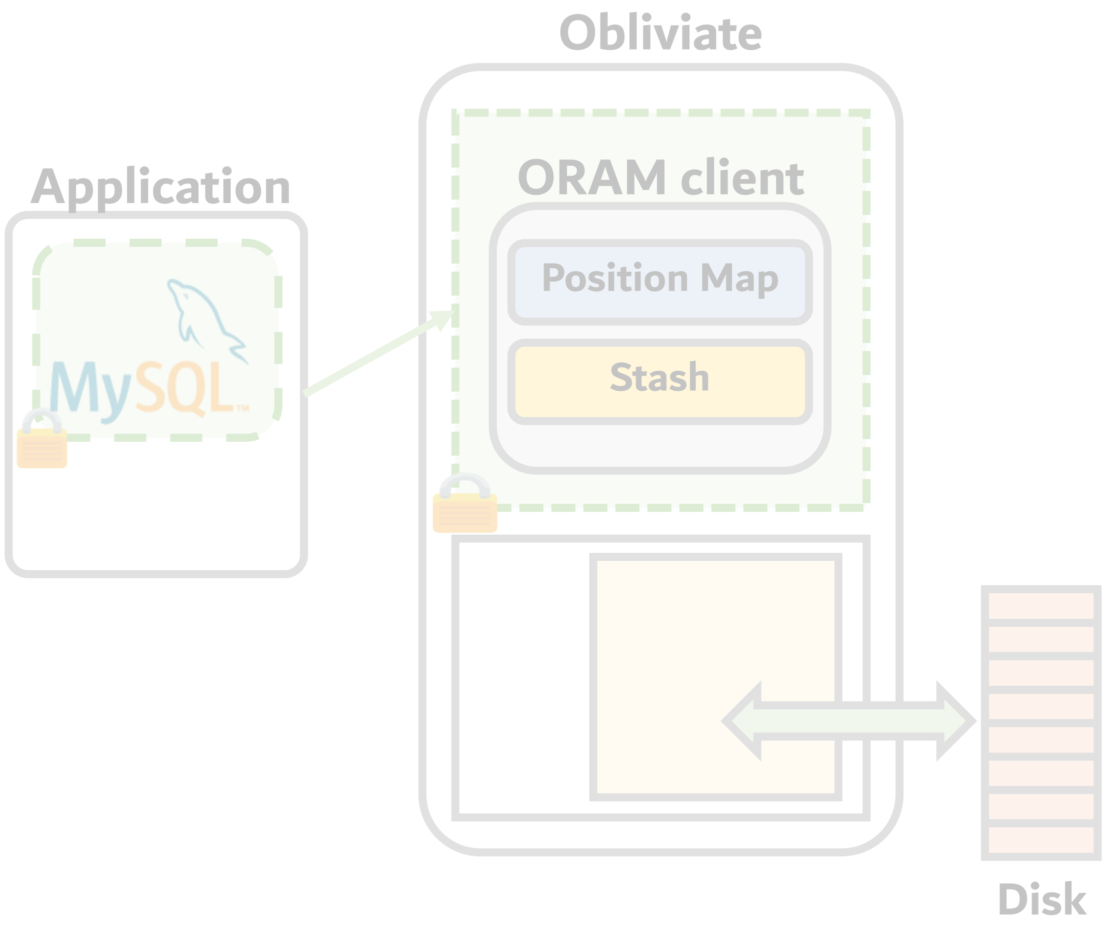
# Securing ORAM

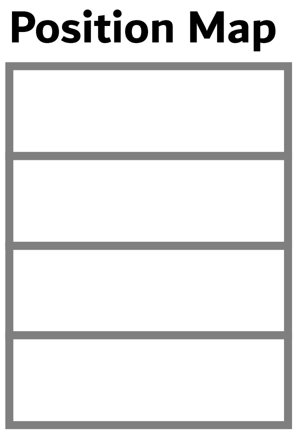


# Securing ORAM

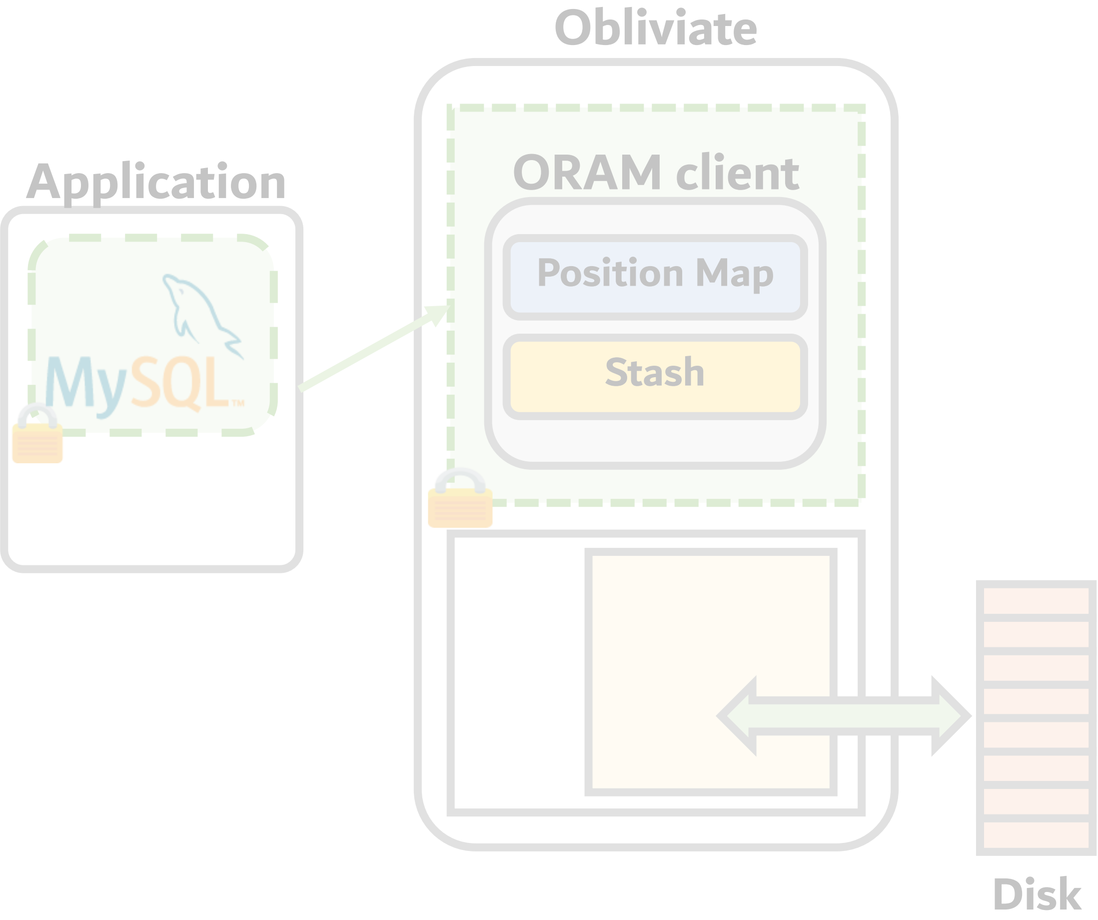


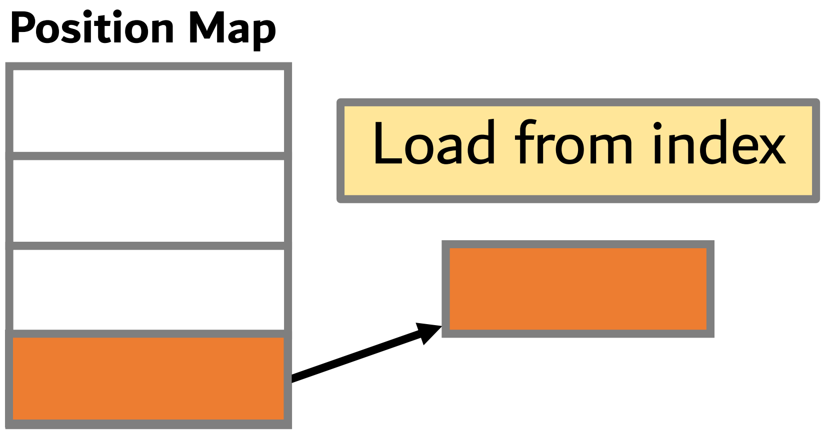
# Securing ORAM



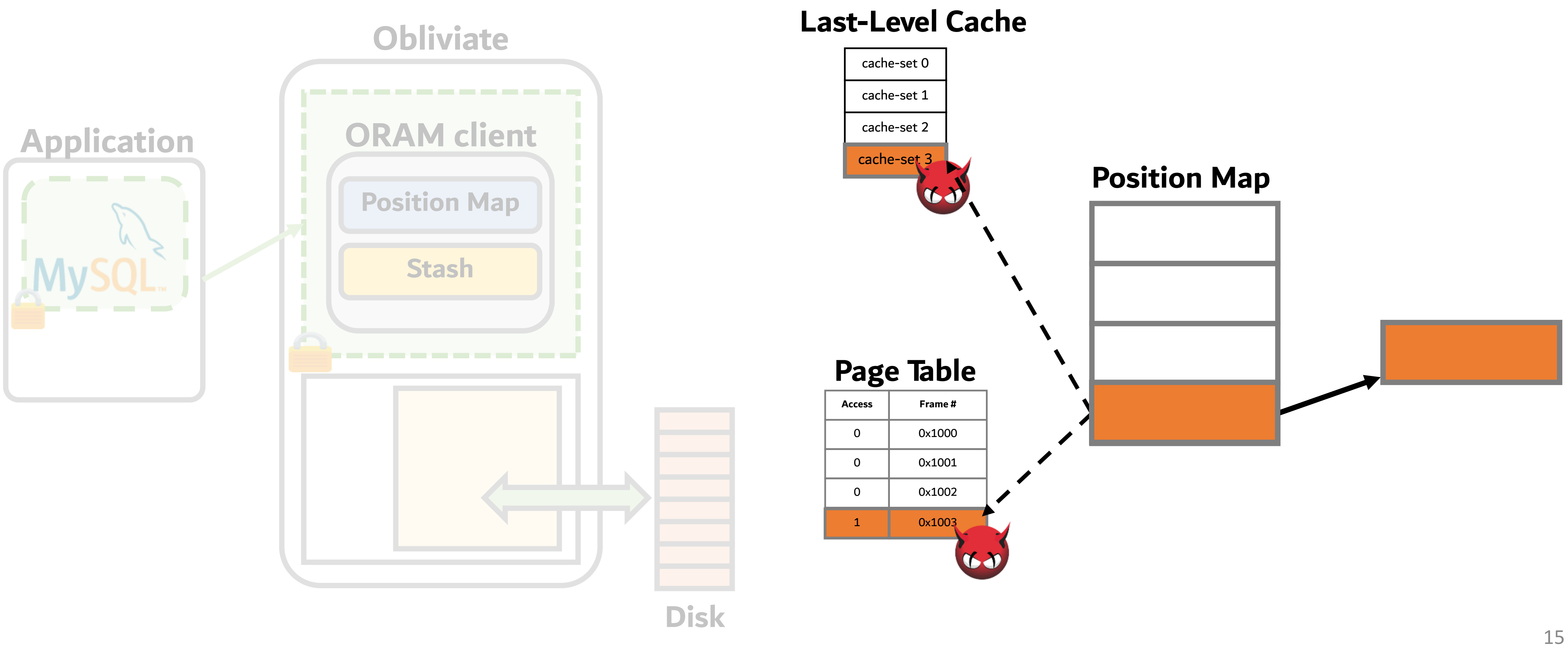



# Securing ORAM

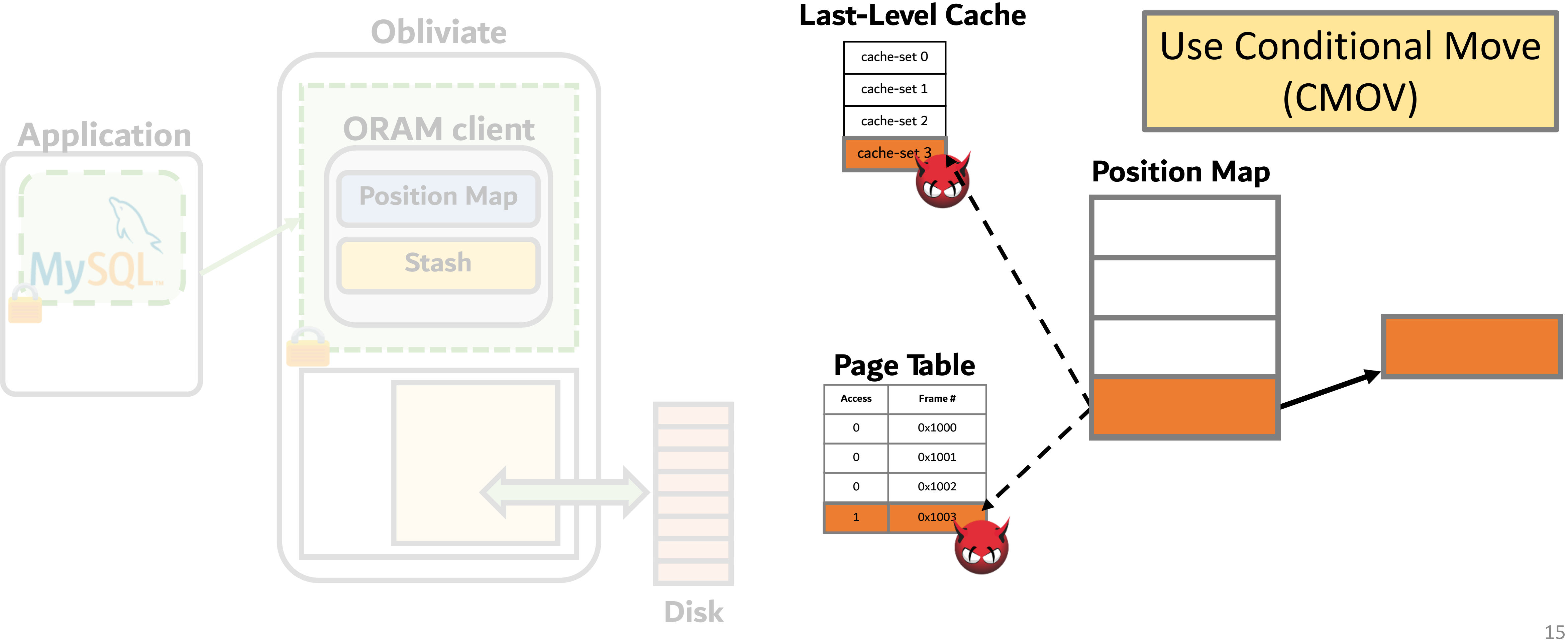


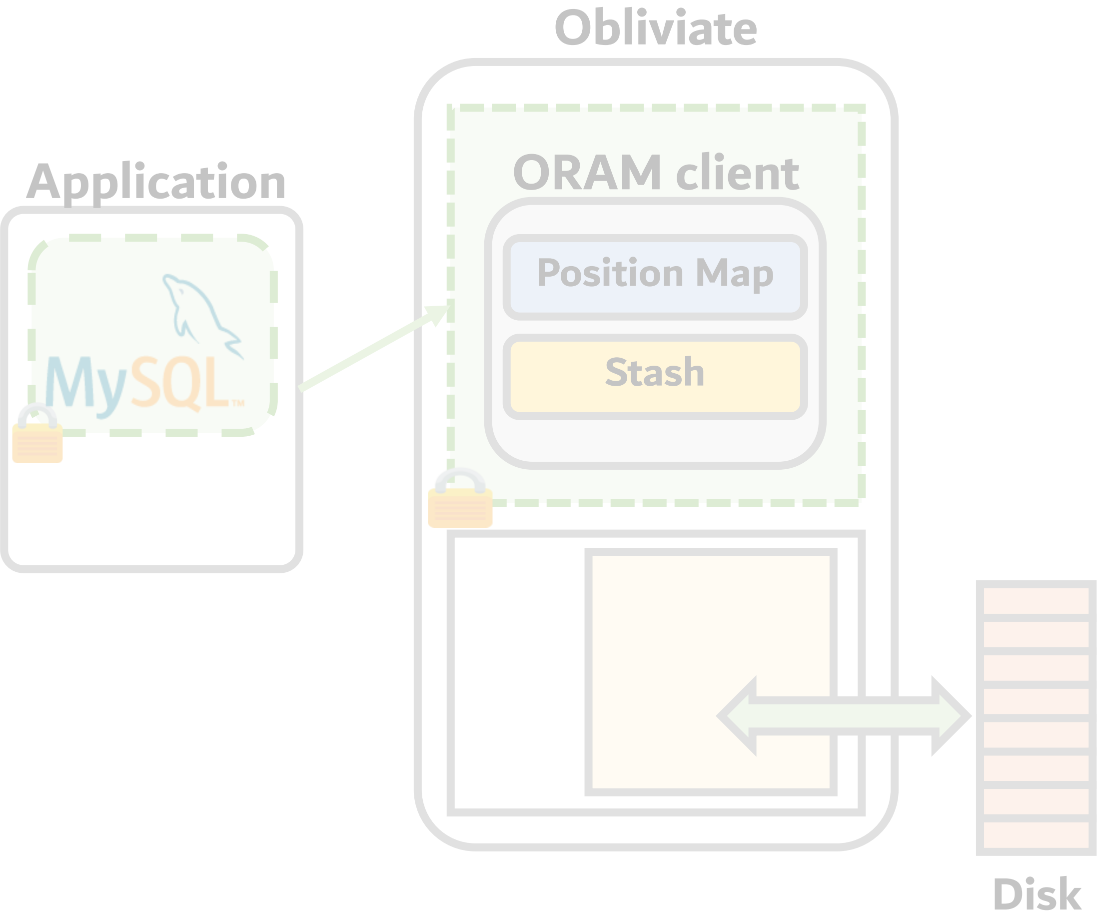
# Securing ORAM

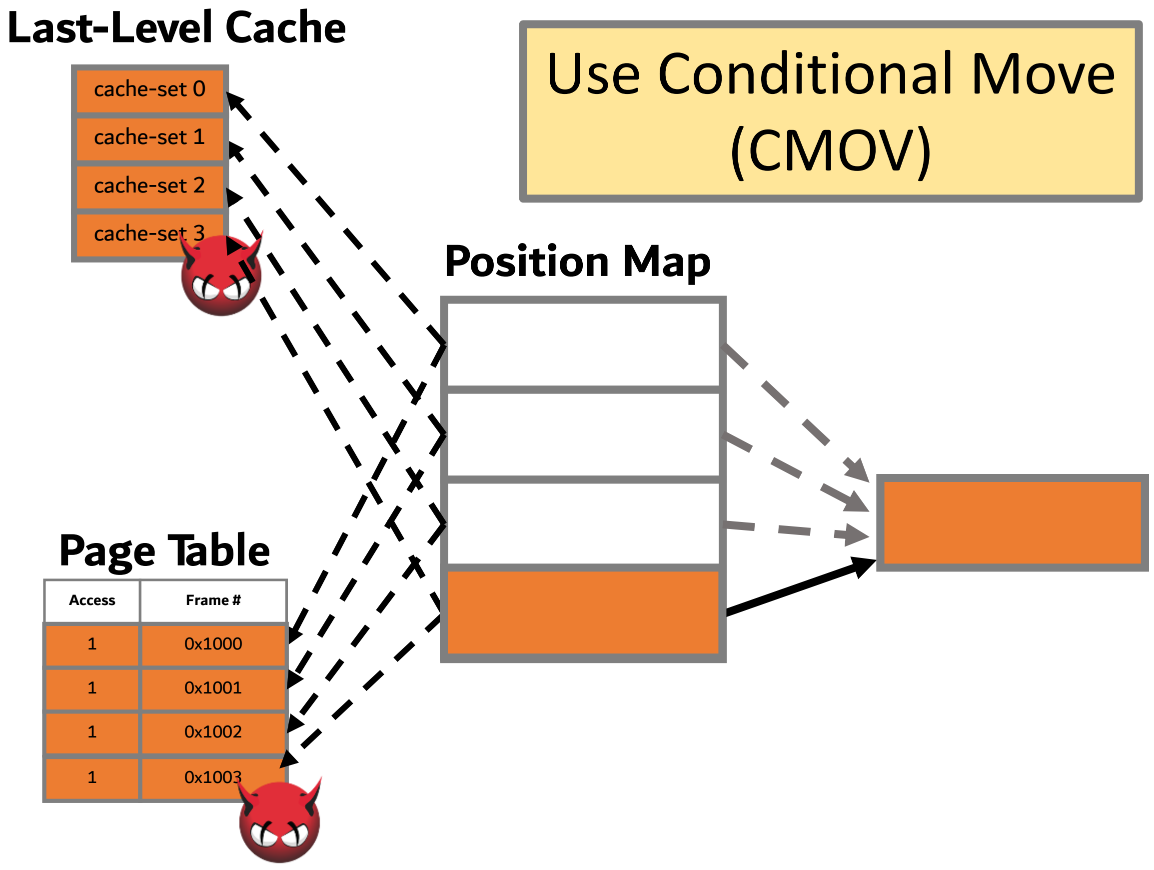


# Securing ORAM



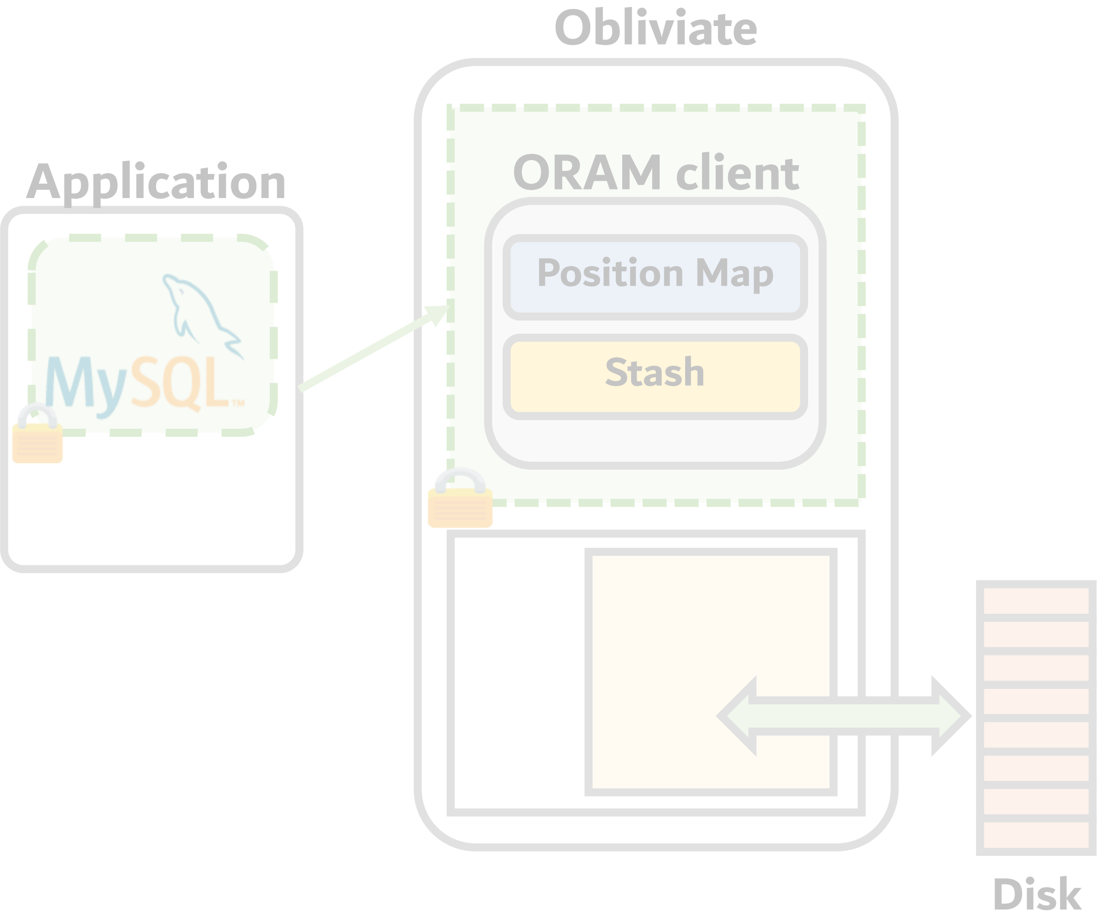
# Securing ORAM

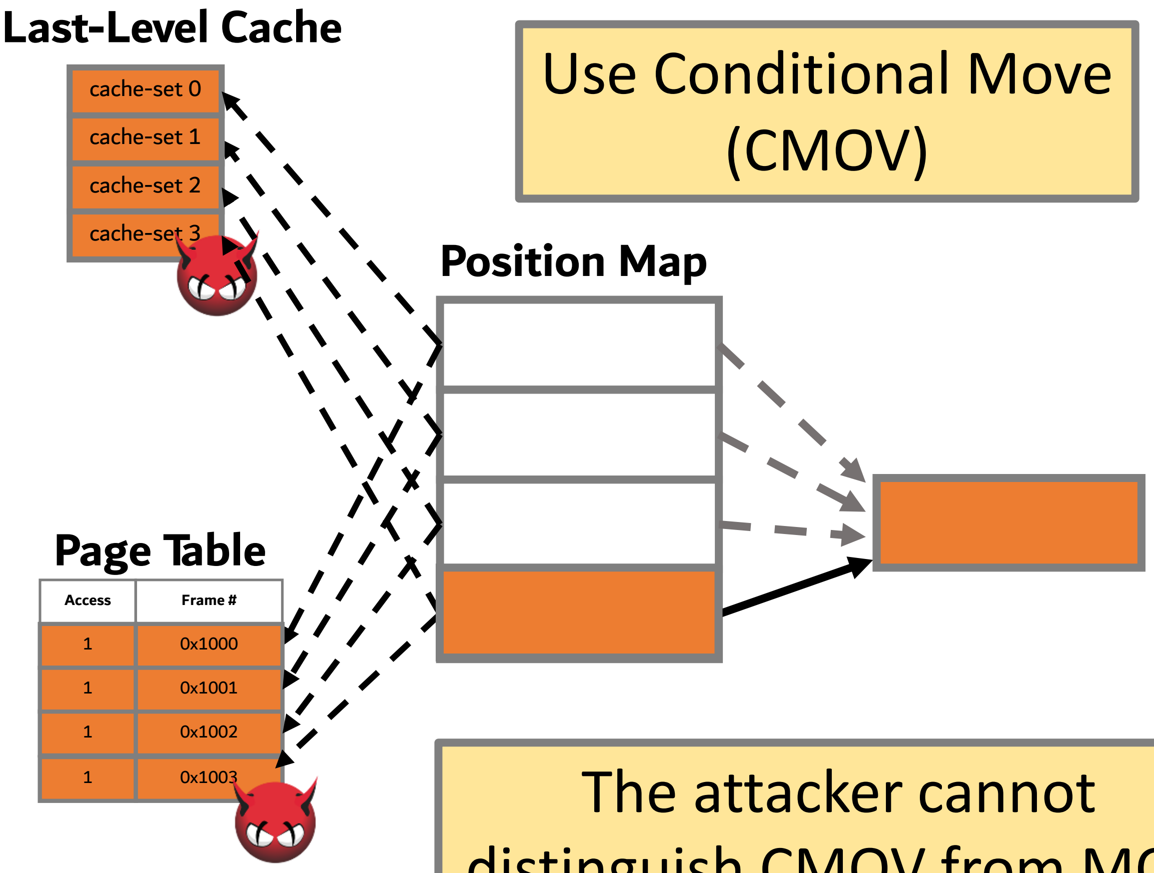



Use Conditional Move (CMOV)

# Securing ORAM

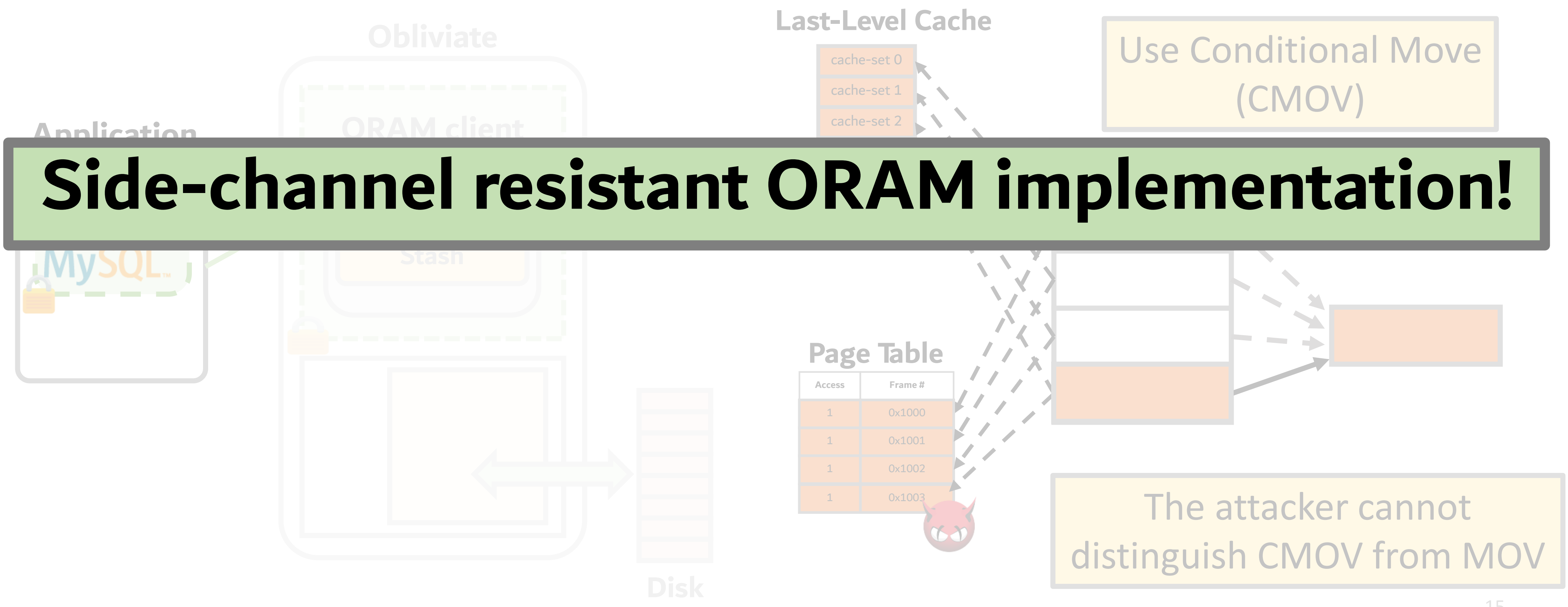



Use Conditional Move (CMOV)

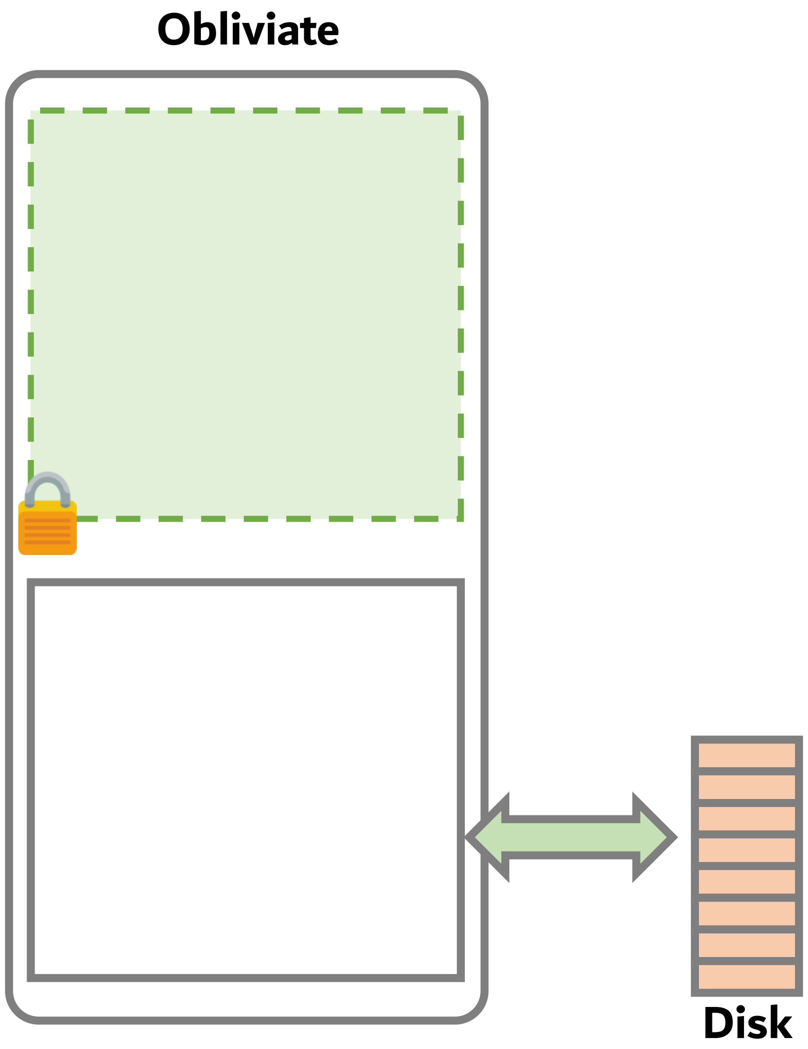
The attacker cannot distinguish CMOV from MOV

# Securing ORAM

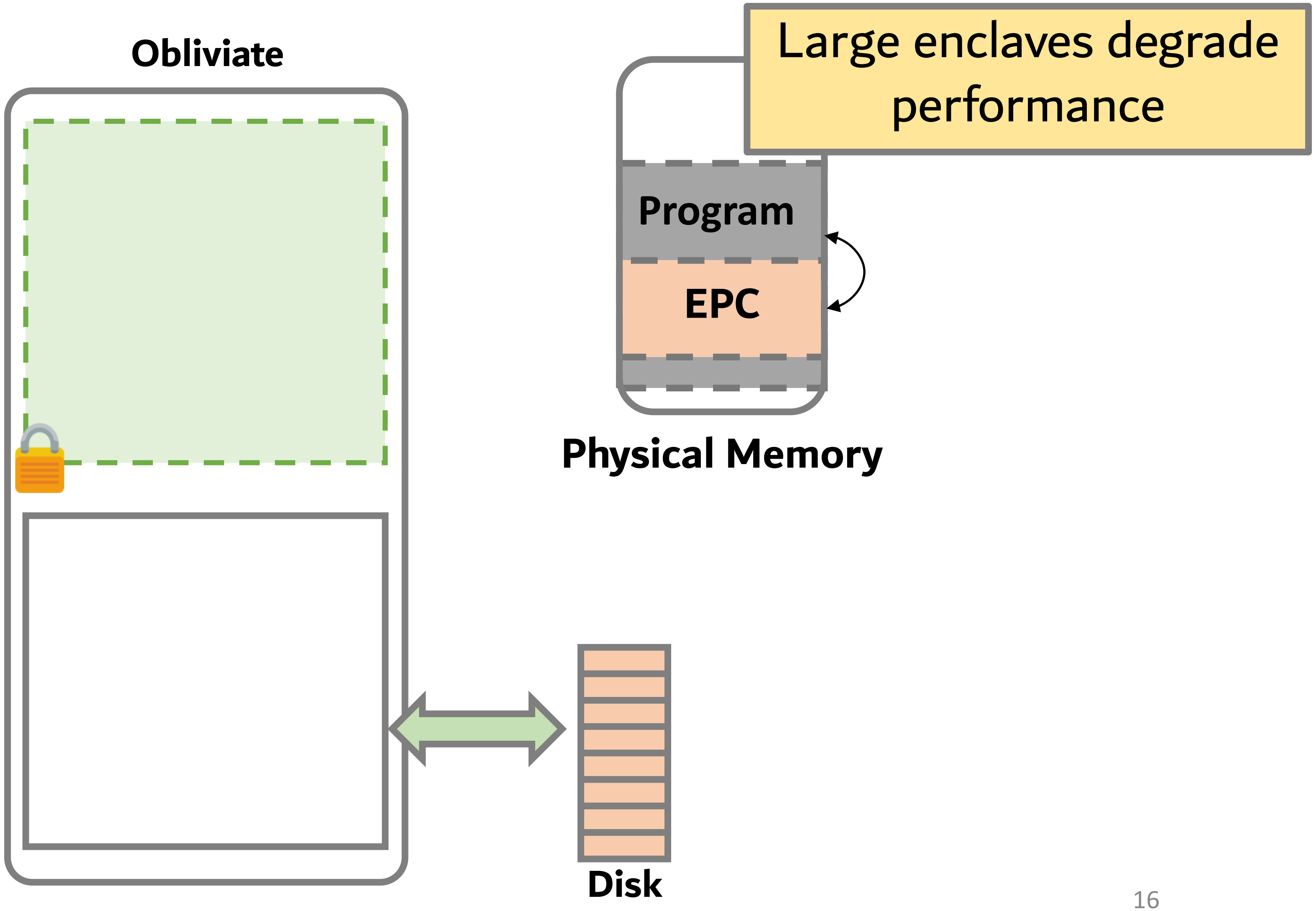



**Side-channel resistant ORAM implementation!**

# Extending Enclave Memory

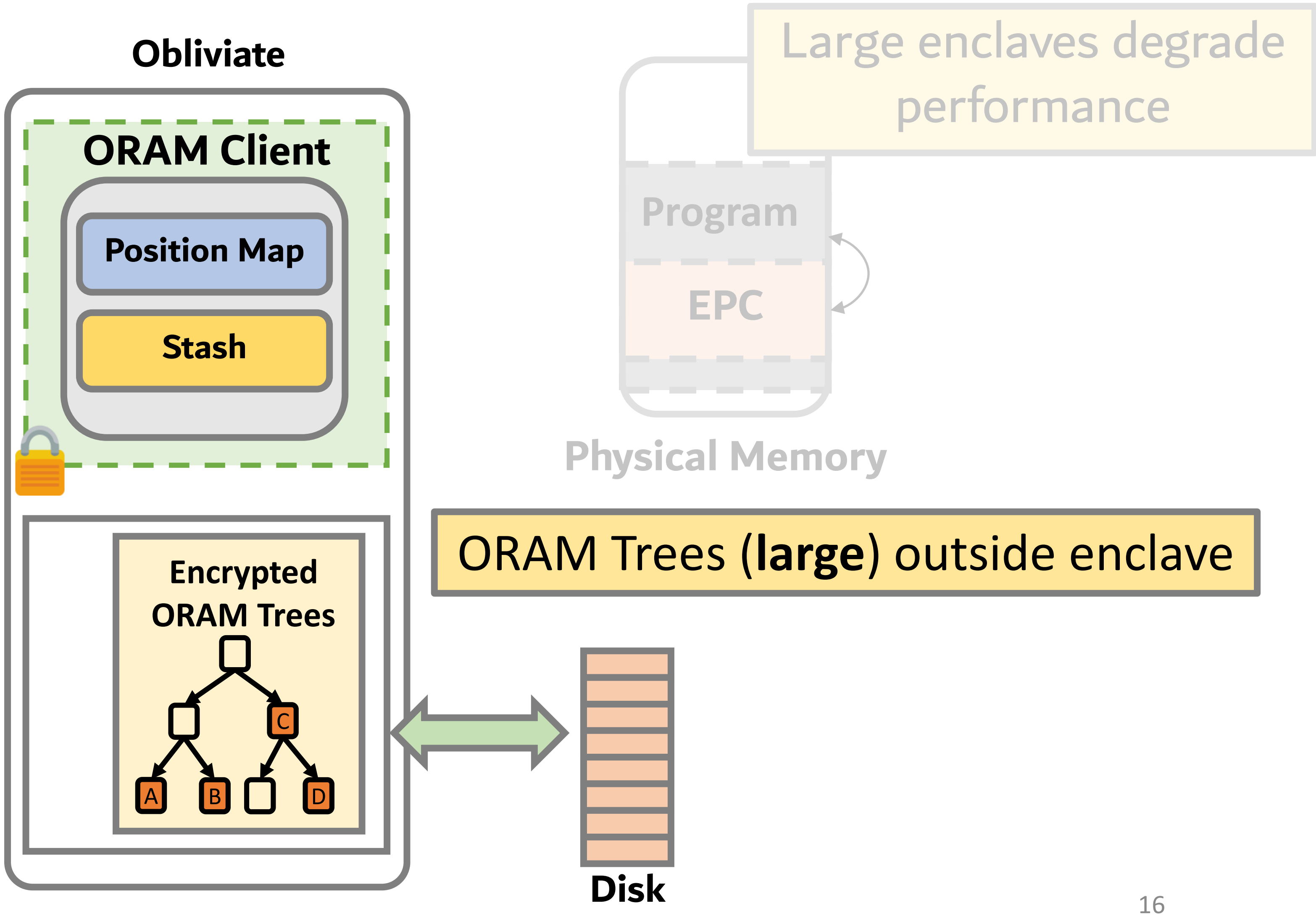


# Extending Enclave Memory



# Extending Enclave Memory

Metadata (**small**) inside enclave

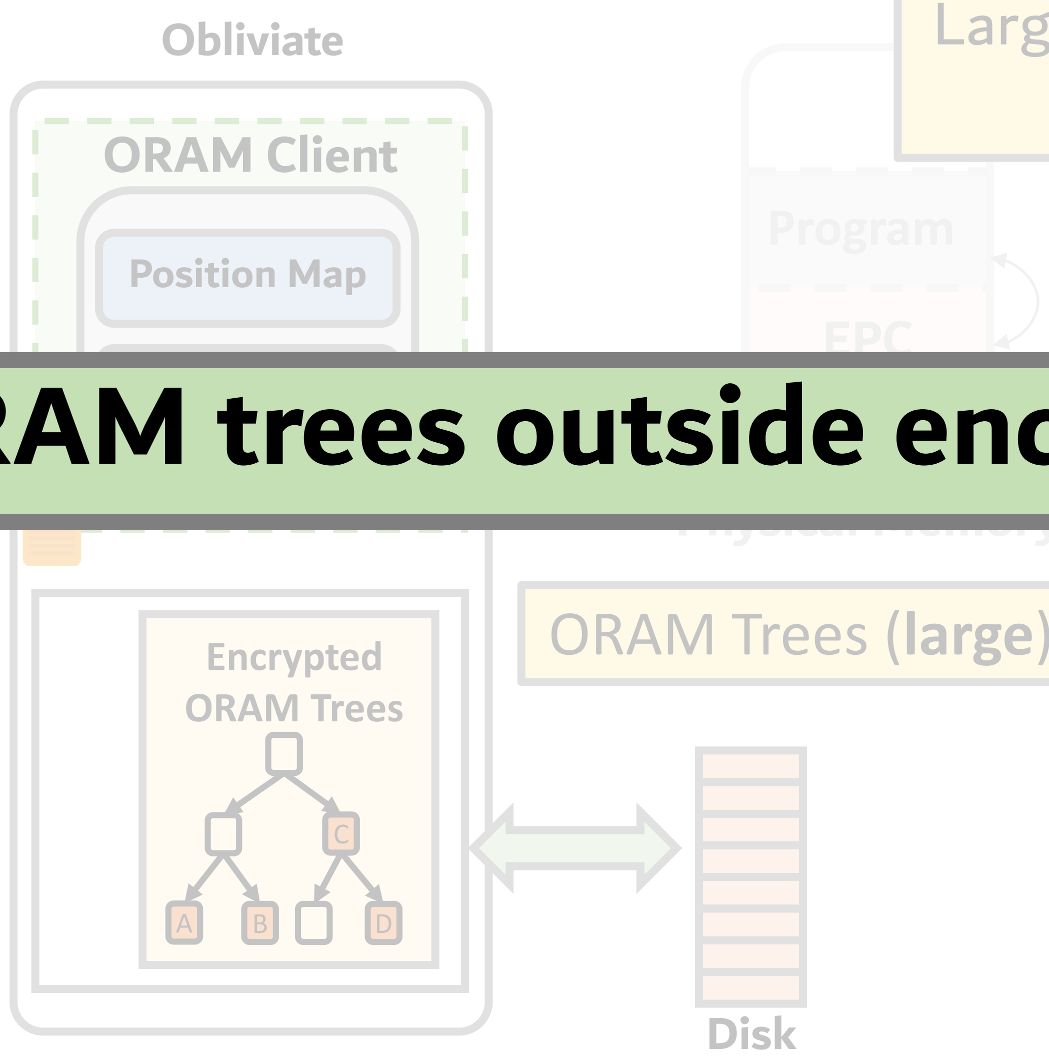



ORAM Trees (**large**) outside enclave

# Extending Enclave Memory

Large enclaves degrade performance

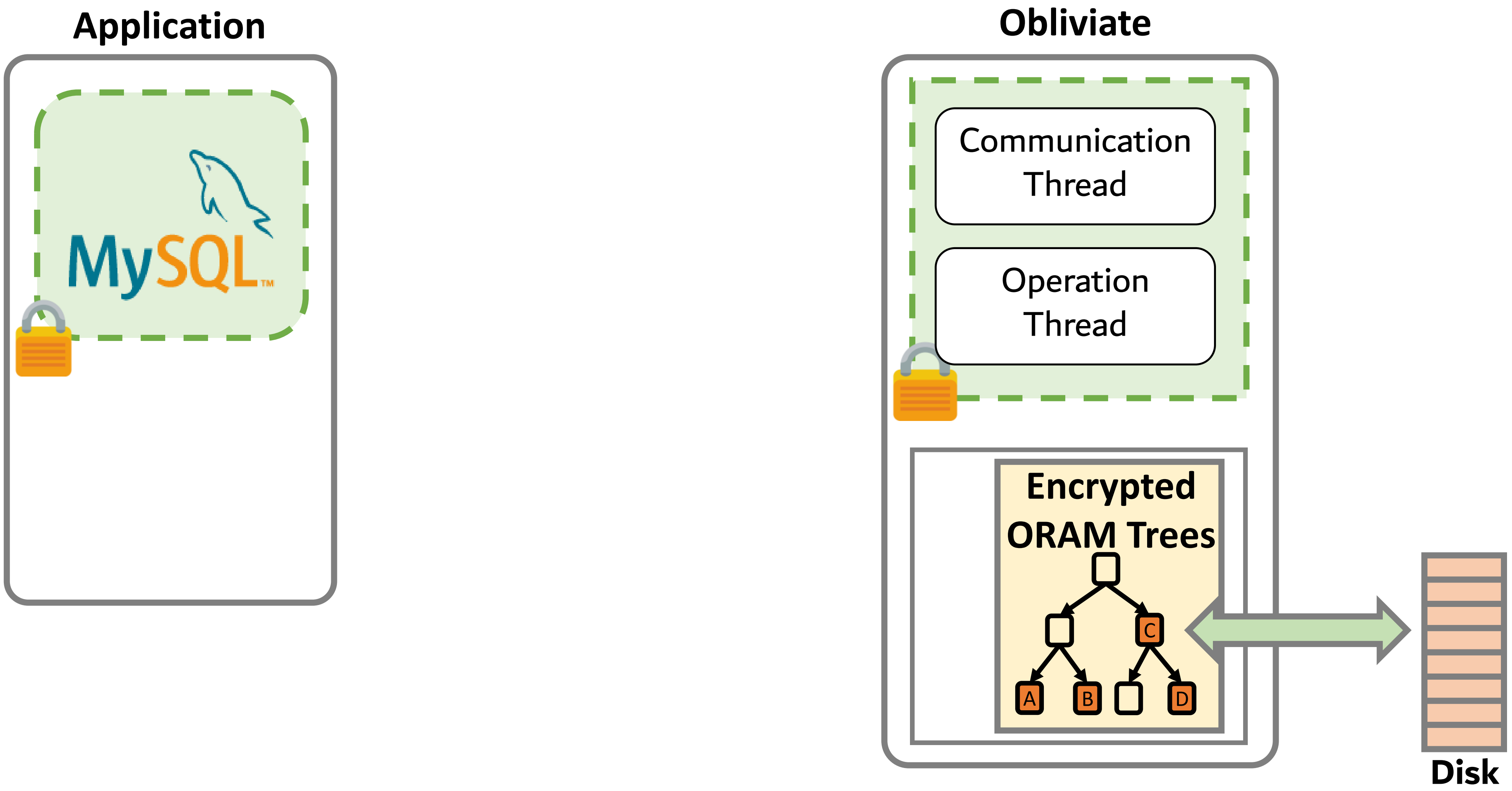
Metadata (**small**) inside enclave



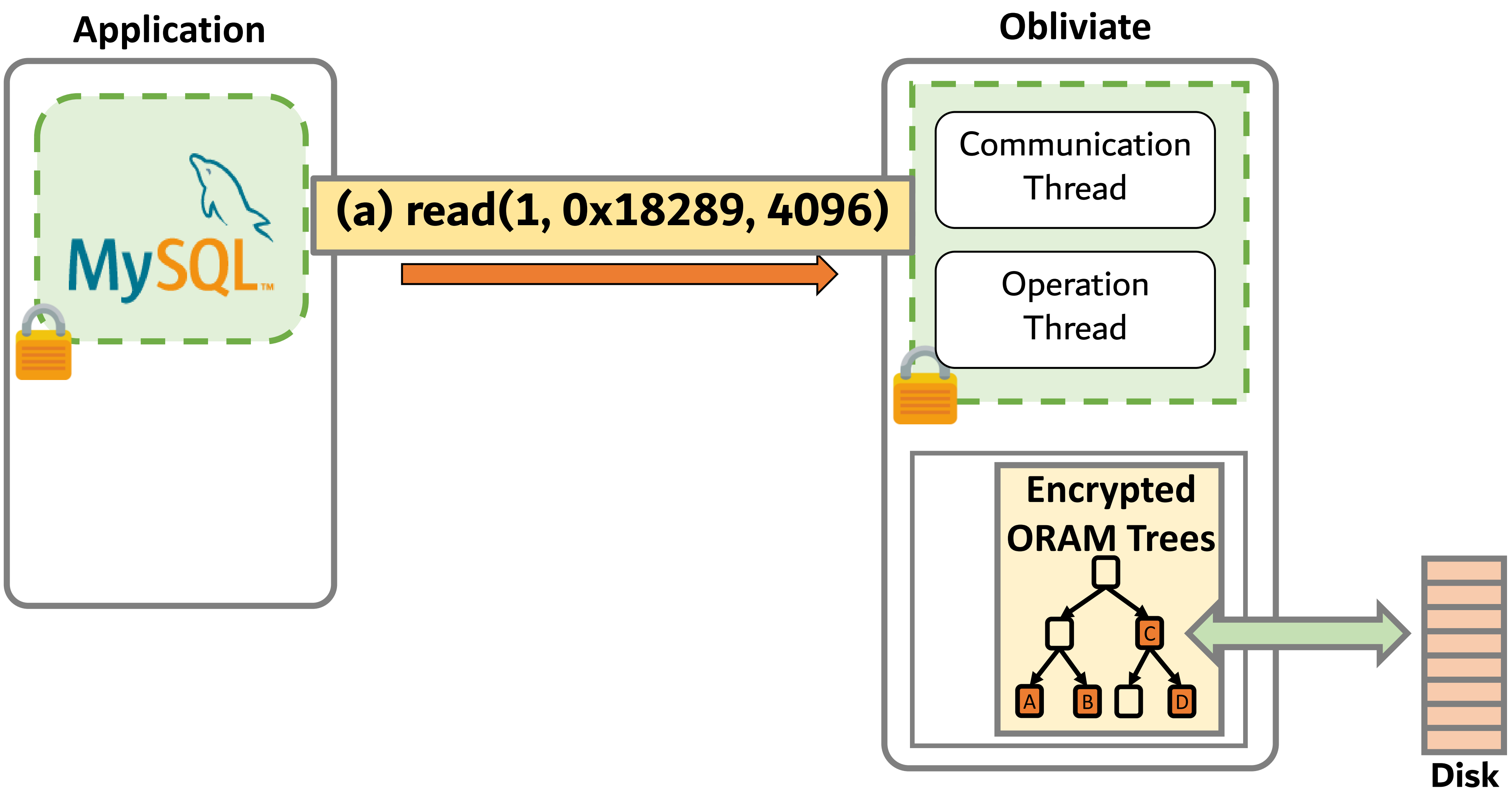

**Encrypted ORAM trees outside enclave!**

ORAM Trees (**large**) outside enclave

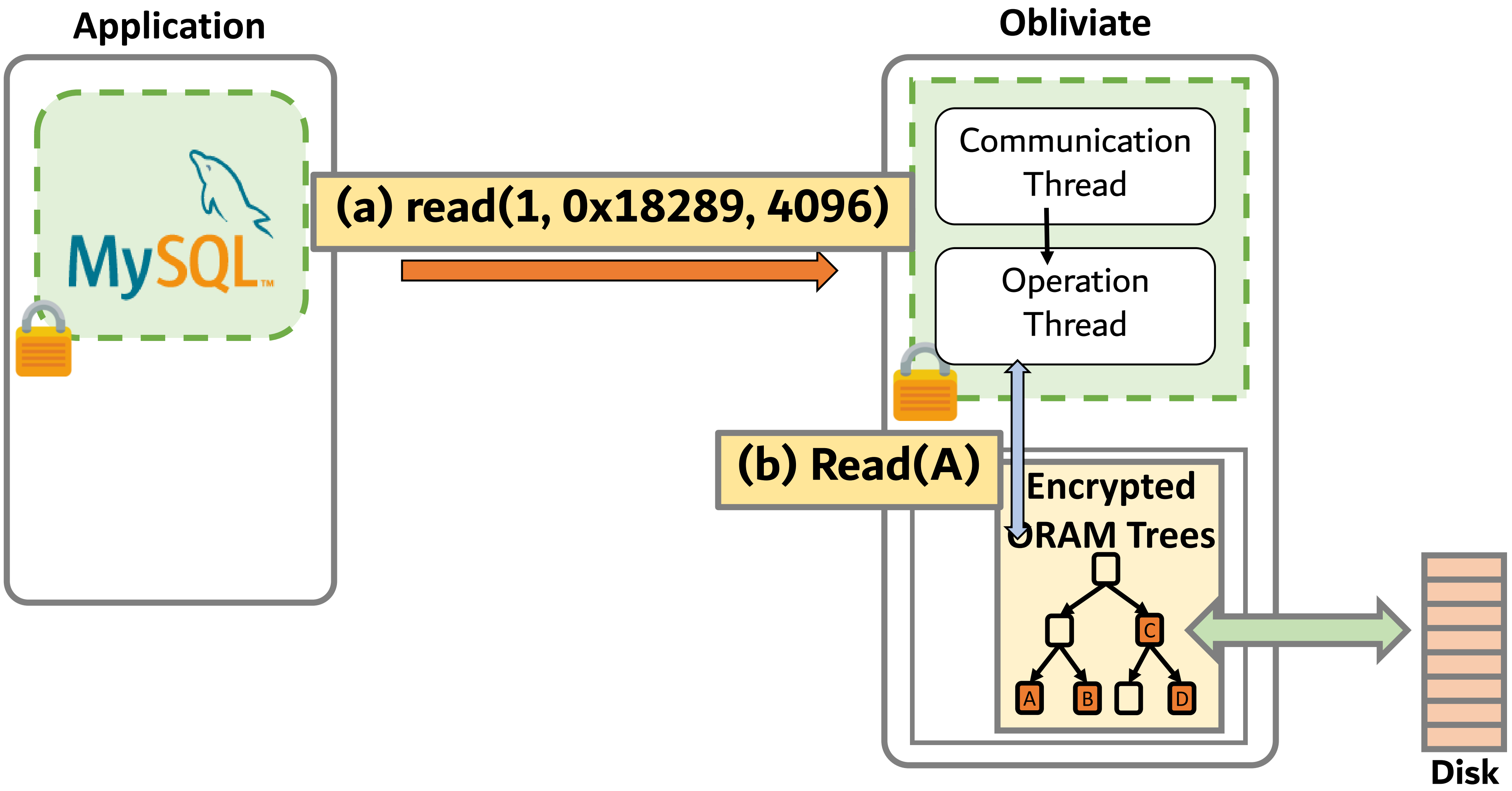
# Leveraging asynchronicity



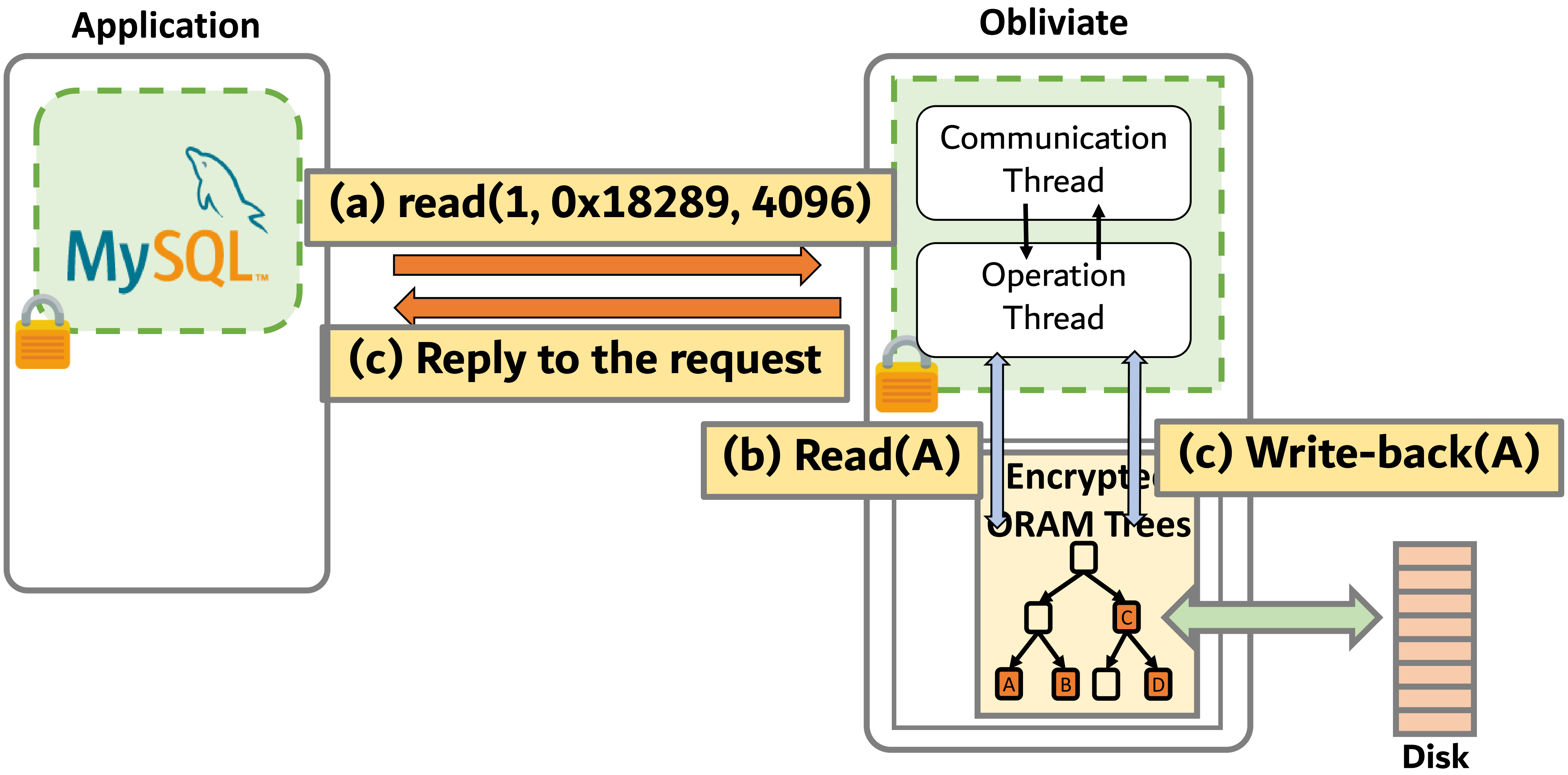
# Leveraging asynchronicity



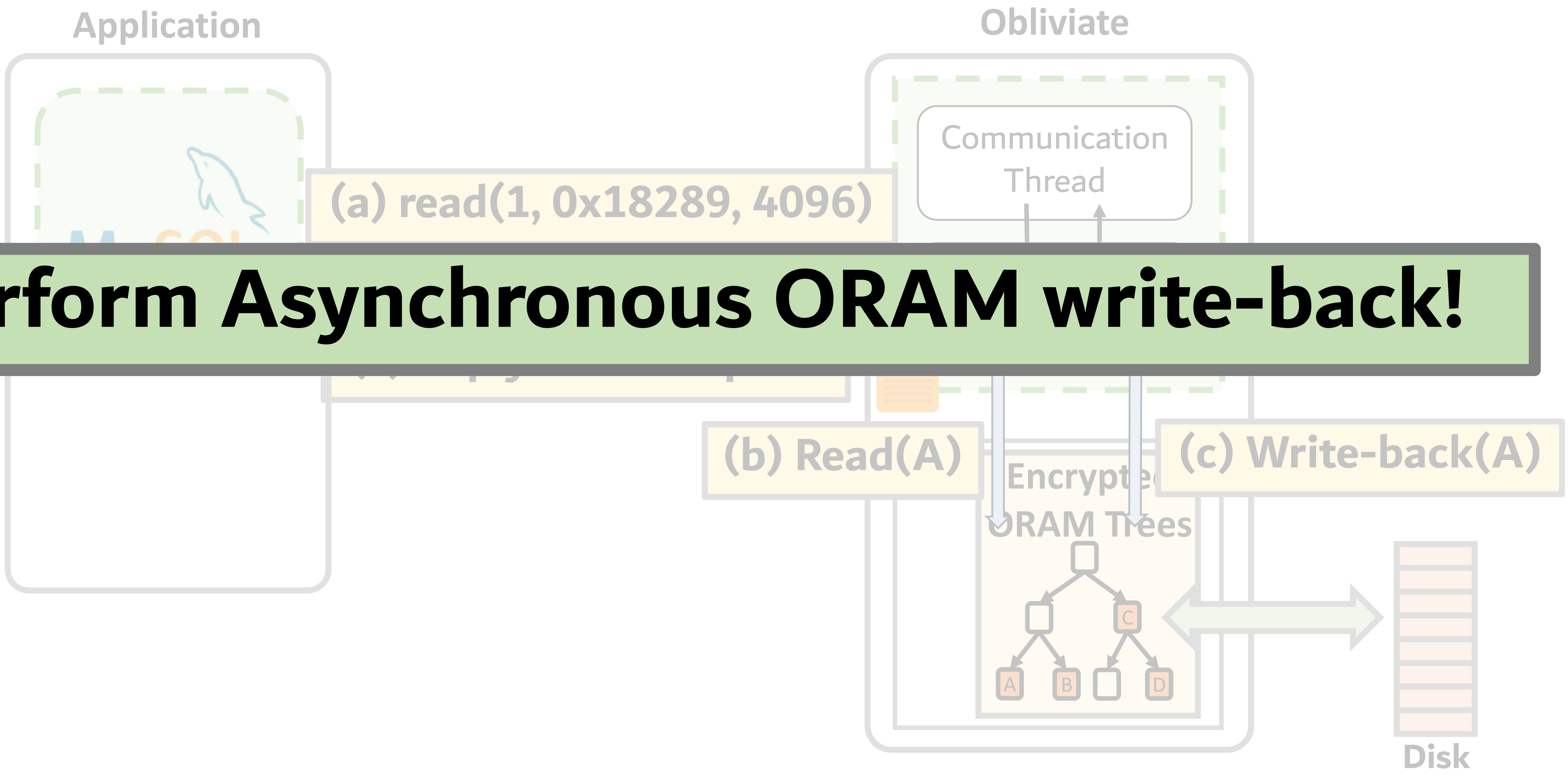
# Leveraging asynchronicity



# Leveraging asynchronicity



# Leveraging asynchronicity




**Perform Asynchronous ORAM write-back!**

# Implementation

1. Obliviate runs using Intel SGX SDK Library

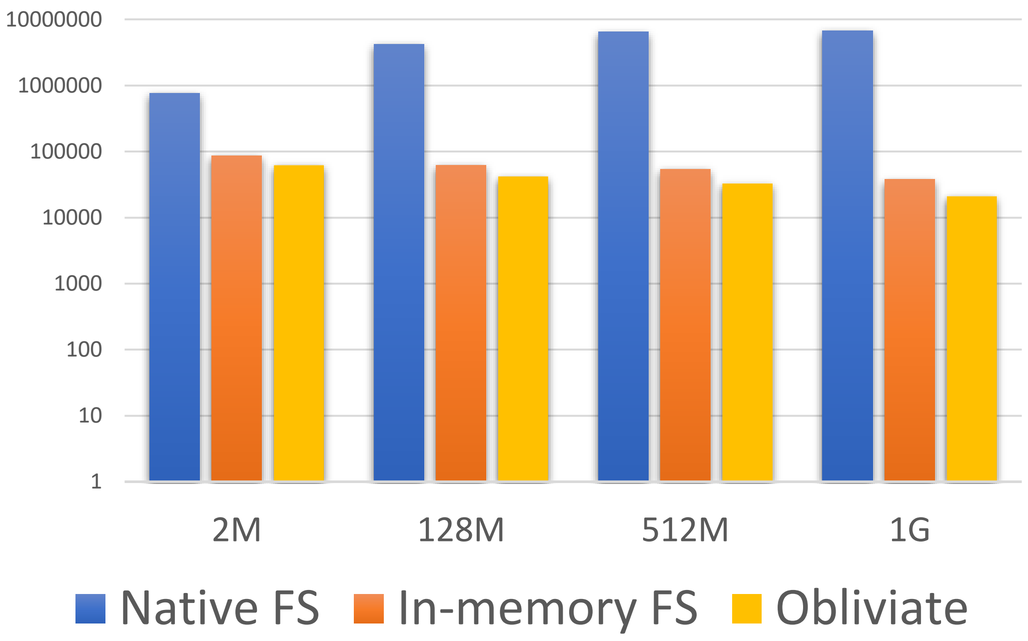
2. Graphene-SGX integration to run *"heavyweight"* applications, e.g., SQLite and Lighttpd

# Performance Evaluation

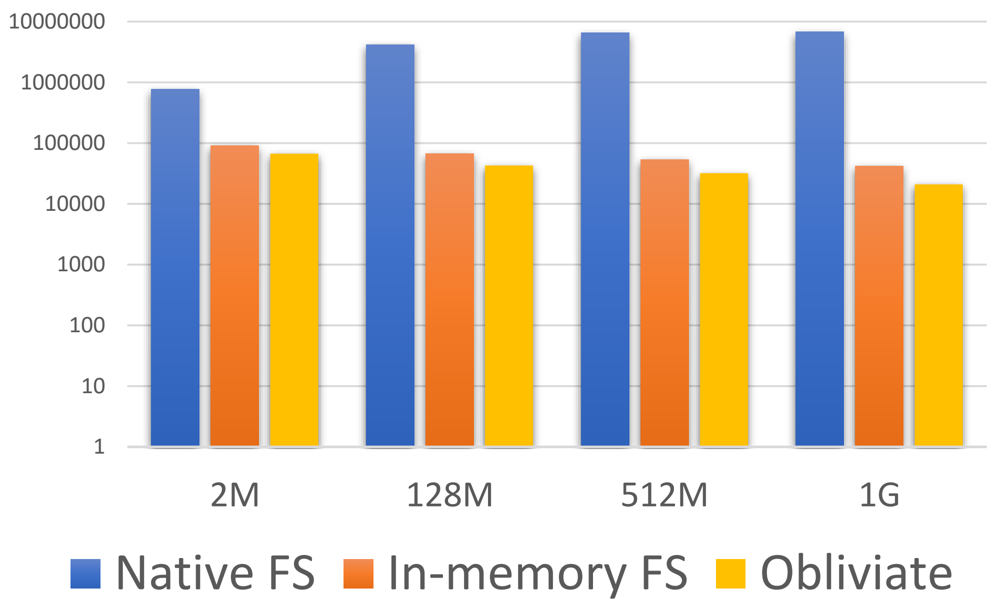
**Evaluated filesystems:**

1. Native Filesystem (Non-SGX)
2. In-memory Filesystem (SGX, based on Graphene-SGX)
3. Obliviate (SGX, based on Intel SGX SDK)

# Iozone Benchmarks

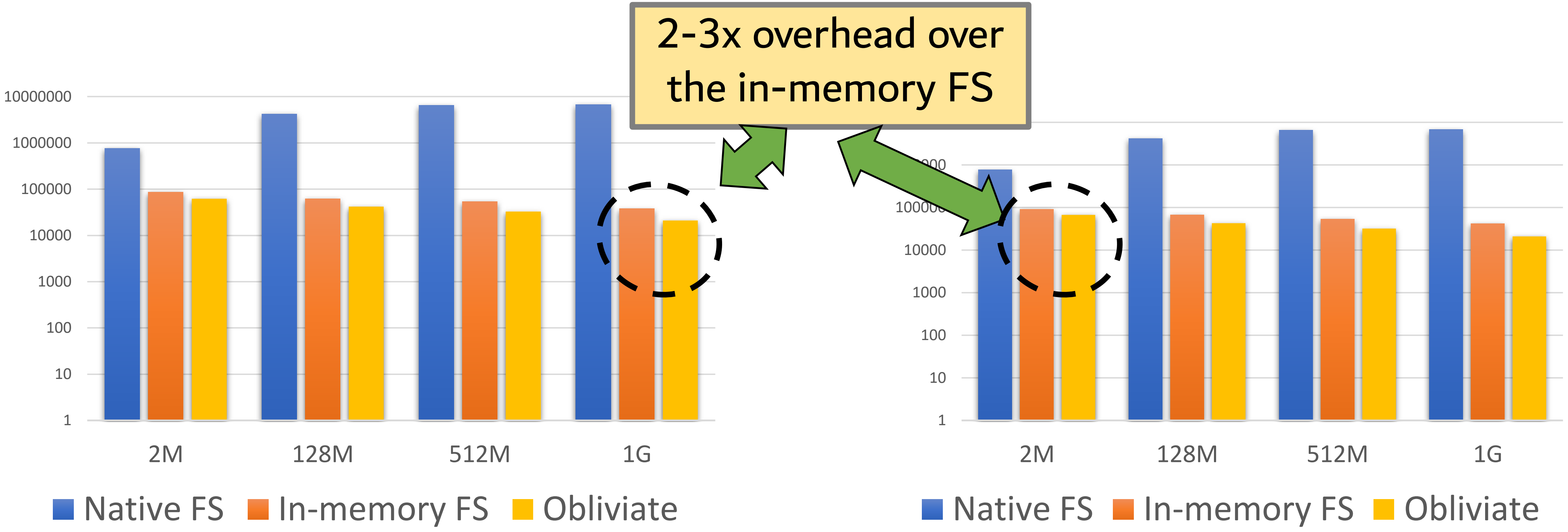



a) Sequential Reads (Bytes/sec)




b) Sequential Writes (Bytes/sec)

# Iozone Benchmarks



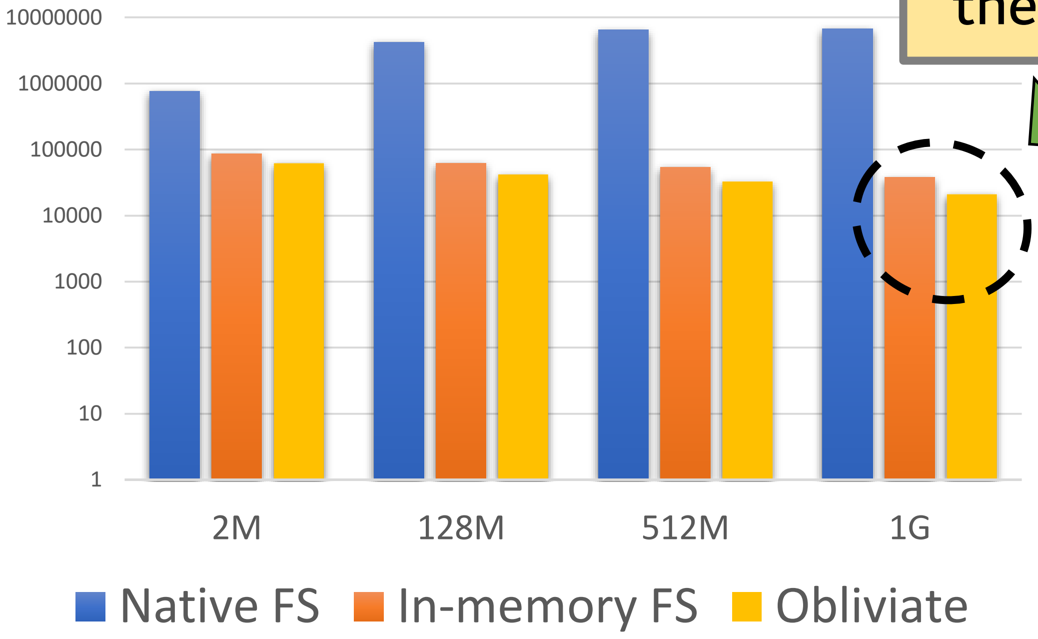

a) Sequential Reads (Bytes/sec)

b) Sequential Writes (Bytes/sec)

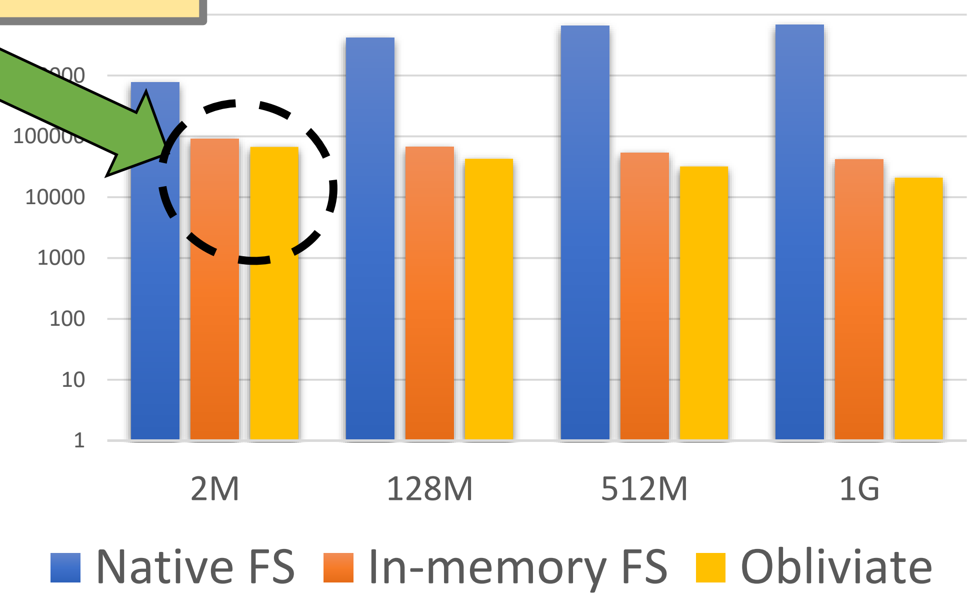
# Iozone Benchmarks

2-3x overhead over the in-memory FS

In-memory FS exerts a lot of pressure on EPC




a) Sequential Reads (Bytes/sec)



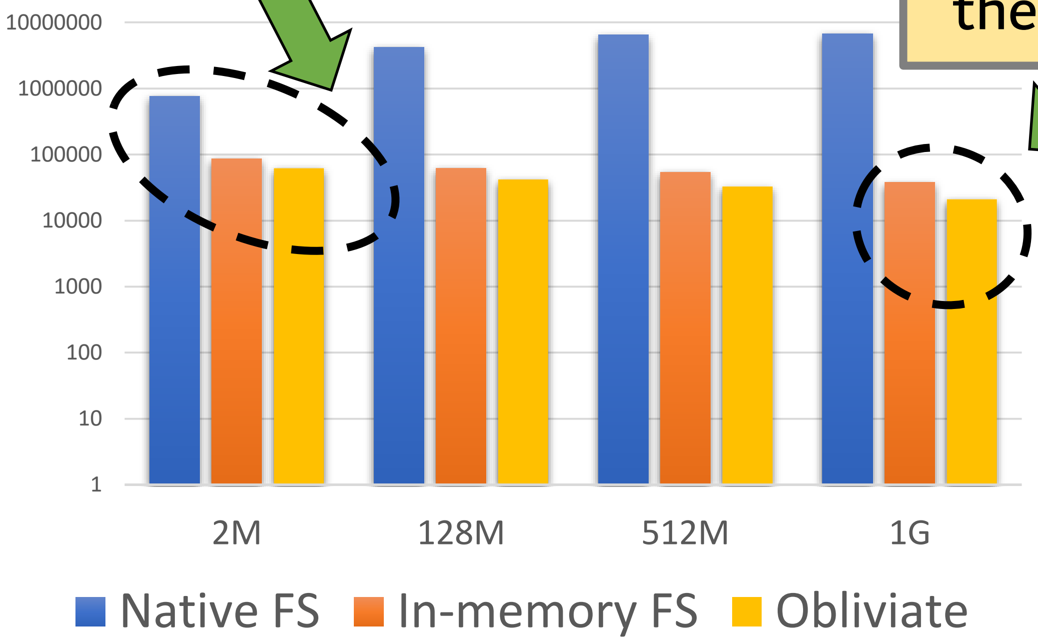

b) Sequential Writes (Bytes/sec)

# Iozone Benchmarks

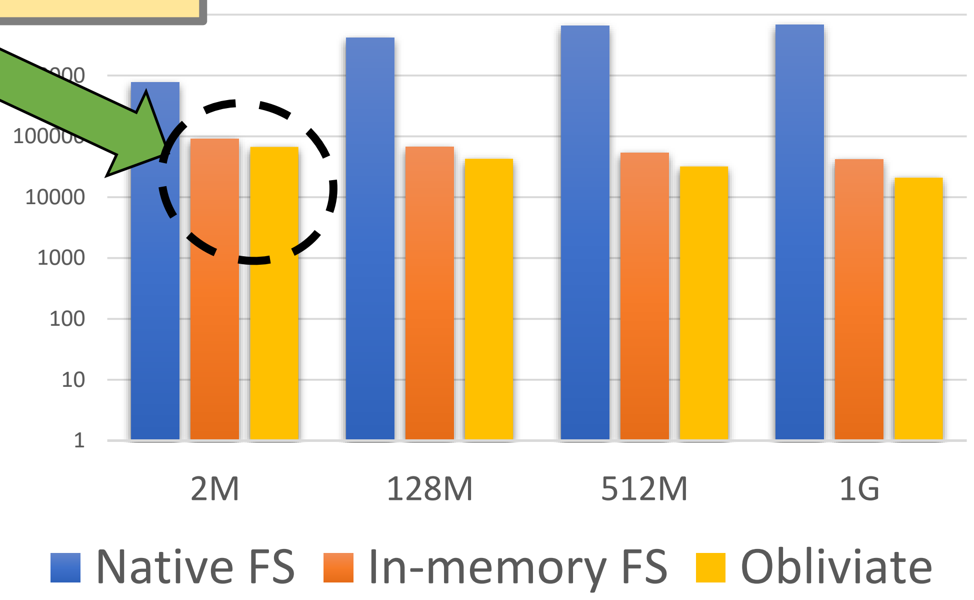
Comparable performance for smaller file sizes

2-3x overhead over the in-memory FS

In-memory FS exerts a lot of pressure on EPC

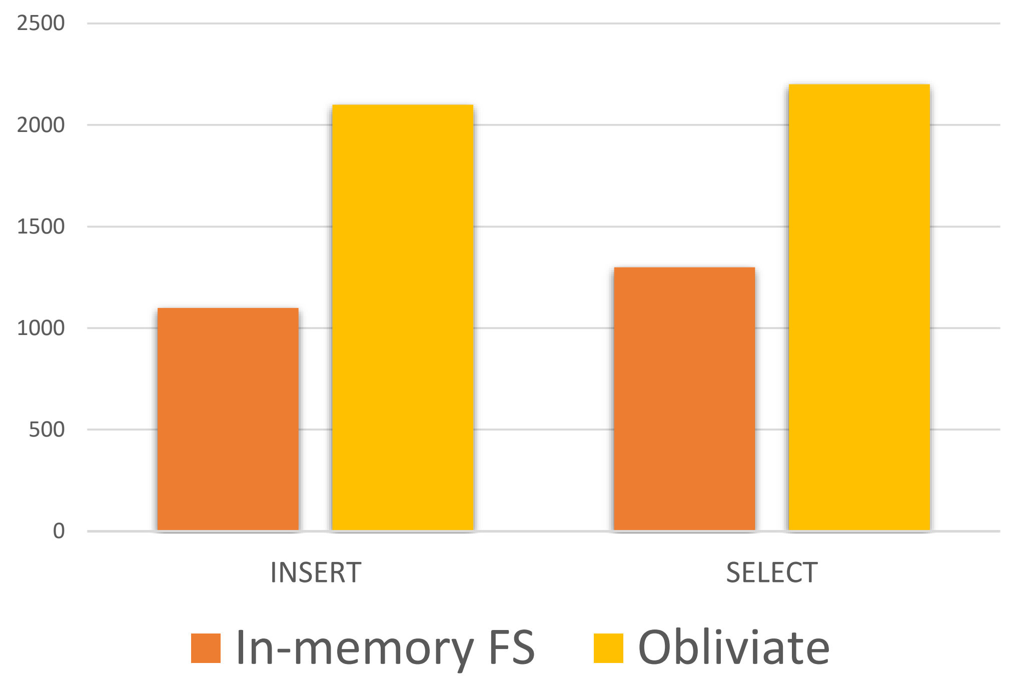



a) Sequential Reads (Bytes/sec)

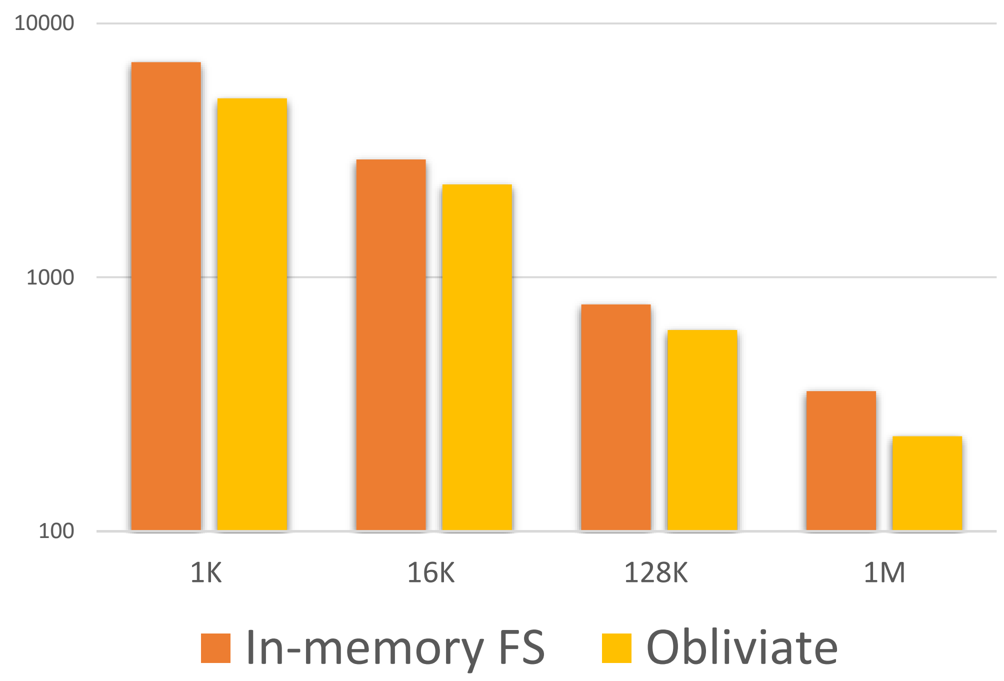



b) Sequential Writes (Bytes/sec)

# Macro-Benchmarks

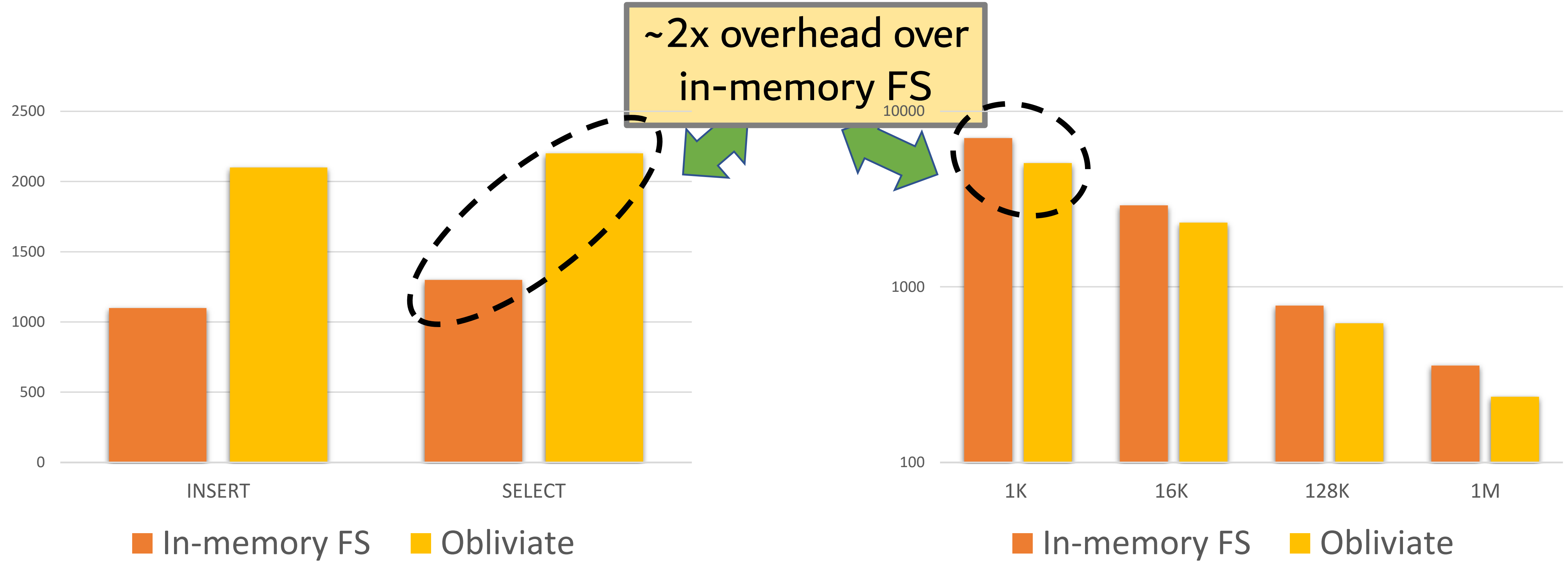



a) SQLite Response Times (milli-sec)




b) Lighttpd Throughput (Req/s)

# Macro-Benchmarks




a) SQLite Response Times (milli-sec)

b) Lighttpd Throughput (Req/s)

# Conclusion

# Conclusion

1. All existing SGX filesystems are vulnerable to side-channels

# Conclusion

1. All existing SGX filesystems are vulnerable to side-channels
2. File system operations can leak sensitive information about program execution.

# Conclusion

1. All existing SGX filesystems are vulnerable to side-channels
2. File system operations can leak sensitive information about program execution.
3. Obliviate provides theoretically-strong defense against side-channels.

# Conclusion

1. All existing SGX filesystems are vulnerable to side-channels
2. File system operations can leak sensitive information about program execution.
3. Obliviate provides theoretically-strong defense against side-channels.

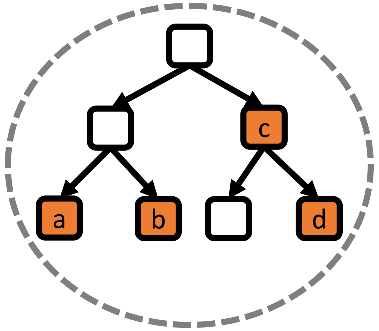
**Opensource:** https://github.com/adilahmad17/Obliviate
**Contact:** ahmad37@purdue.edu

# Thanks! Merci! Shukriya!

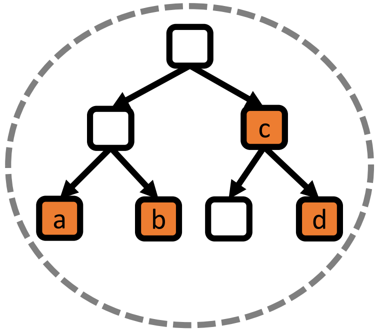
# Extra Slides

# Securing file system

# Securing file system

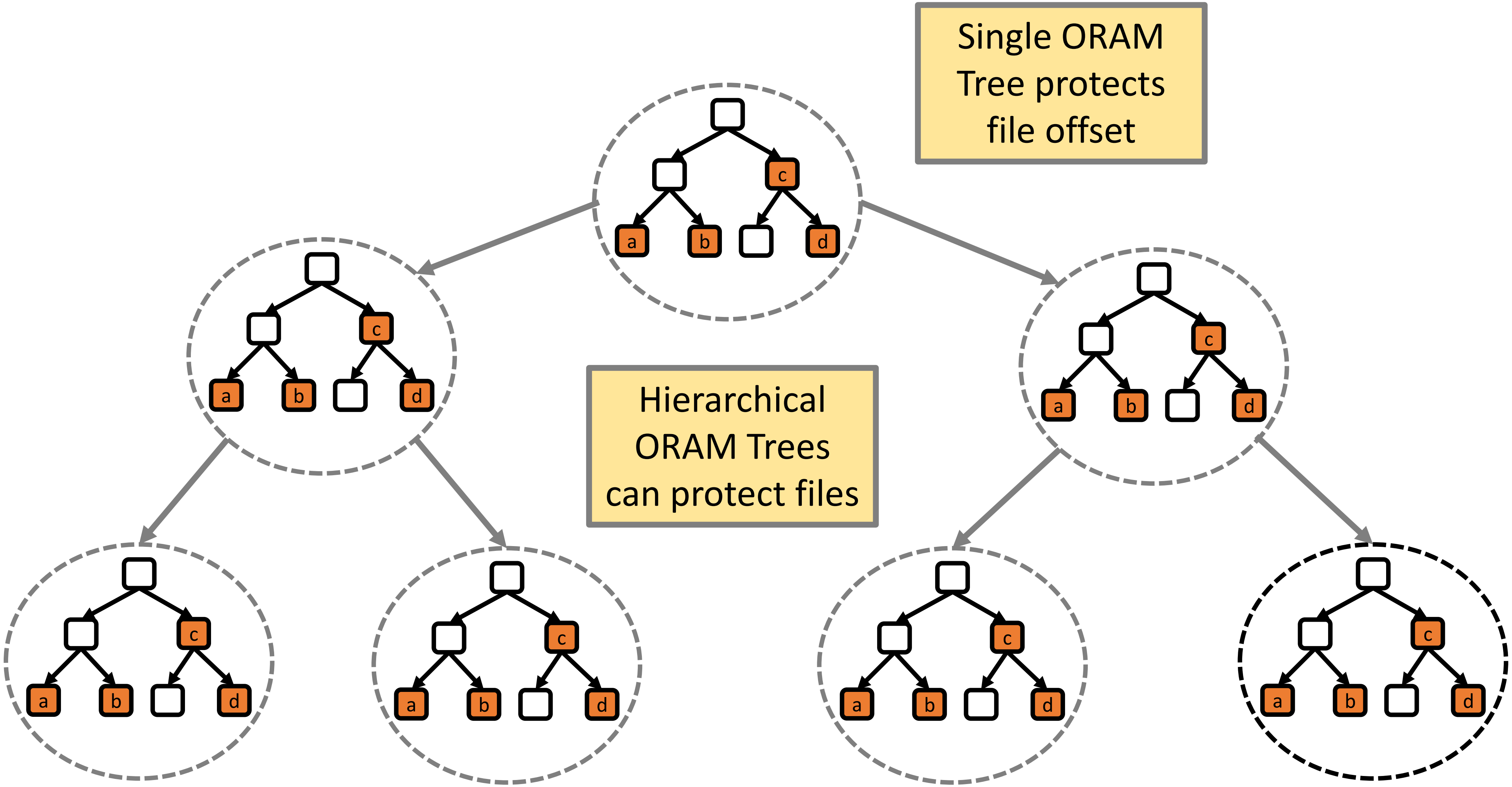


# Securing file system




Single ORAM Tree protects file offset

# Securing file system



# Securing file system