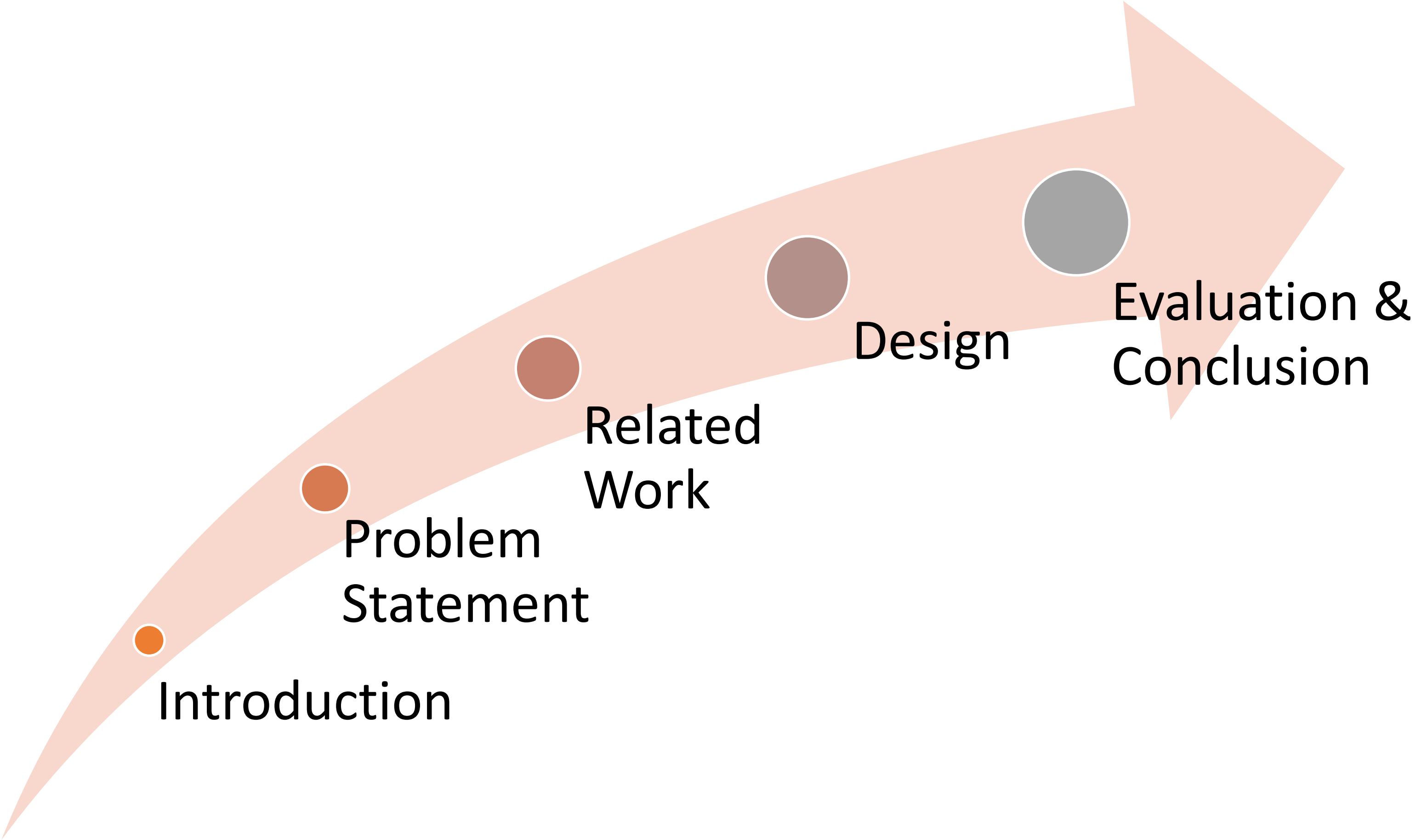
SecureFS: A Secure File System for Intel SGX

Sandeep Kumar and Smruti R. Sarangi
Indian Institute of Technology Delhi, India

The 24th International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2021)
Donostia / San Sebastian, Spain on October 6-8, 2021.

Outline

Outline

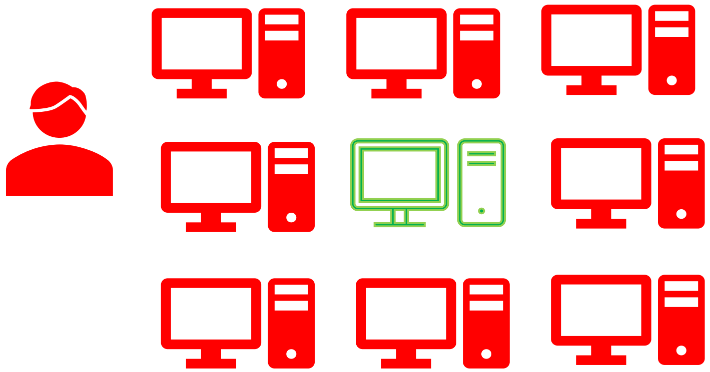

Introduction
Problem Statement
Related Work
Design
Evaluation & Conclusion

Introduction

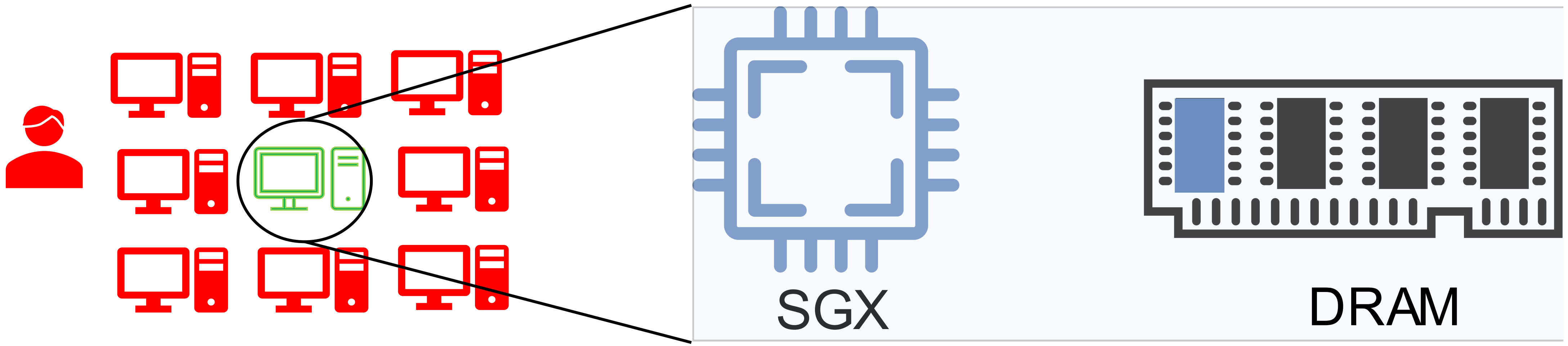
What is SGX, and why should I care?

Intel **S**ecure **G**uard e**X**tension

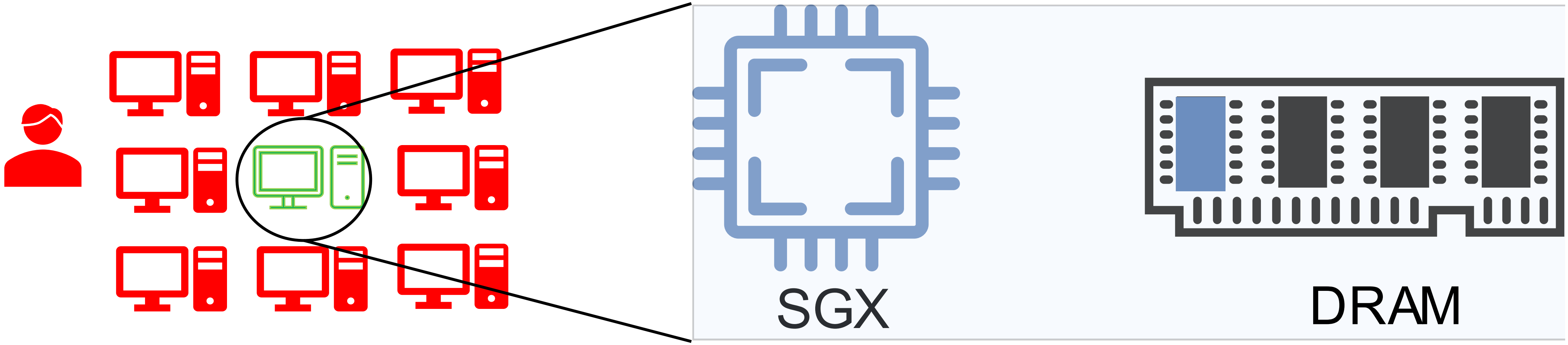
Intel Secure Guard eXtension



Intel Secure Guard eXtension



Intel Secure Guard eXtension



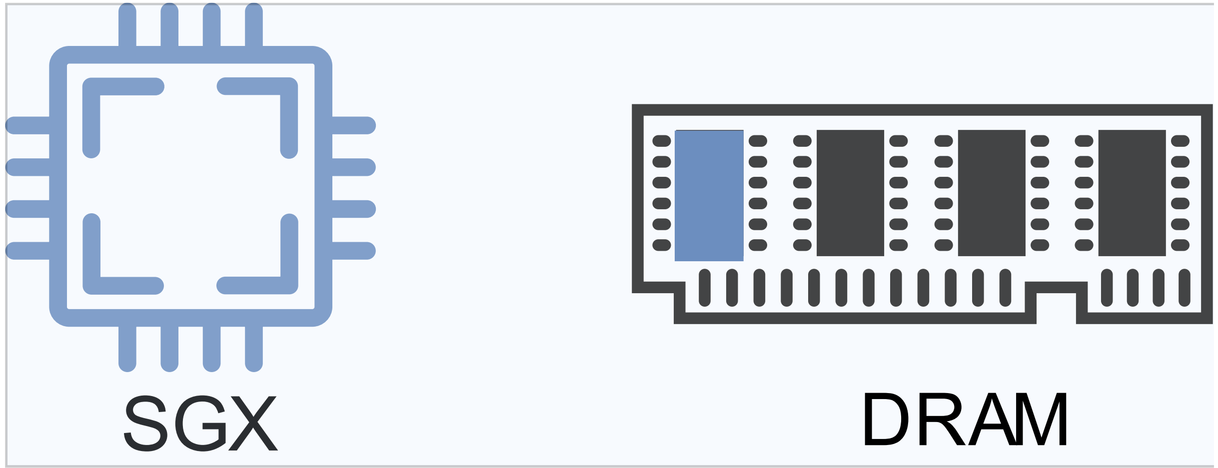
Secure

Encrypted Memory

No-Snooping or tampering

Hardware-managed

Intel SGX: Limitations



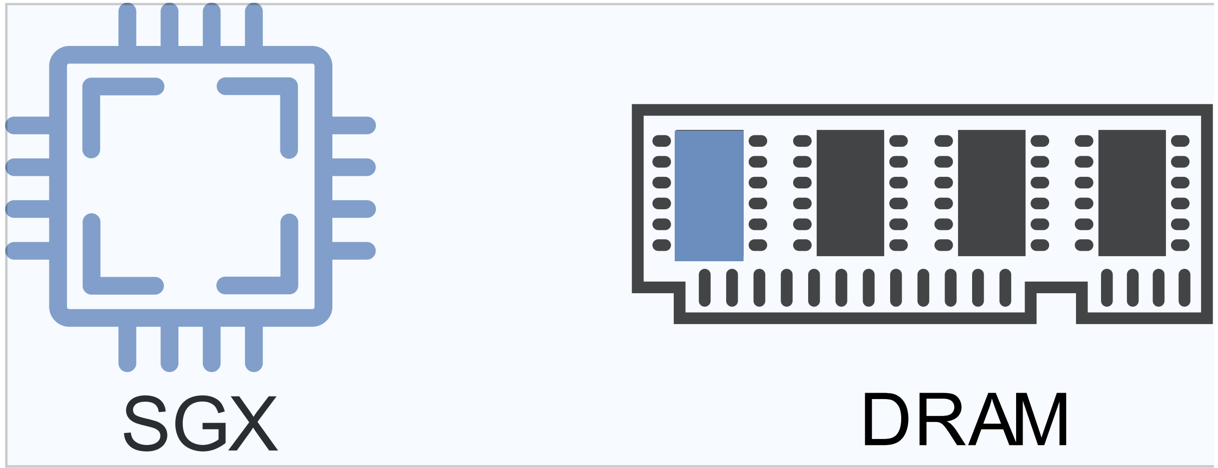
Intel SGX: Limitations

Limited amount of trusted memory.

- 128 MB, 92 MB usable.

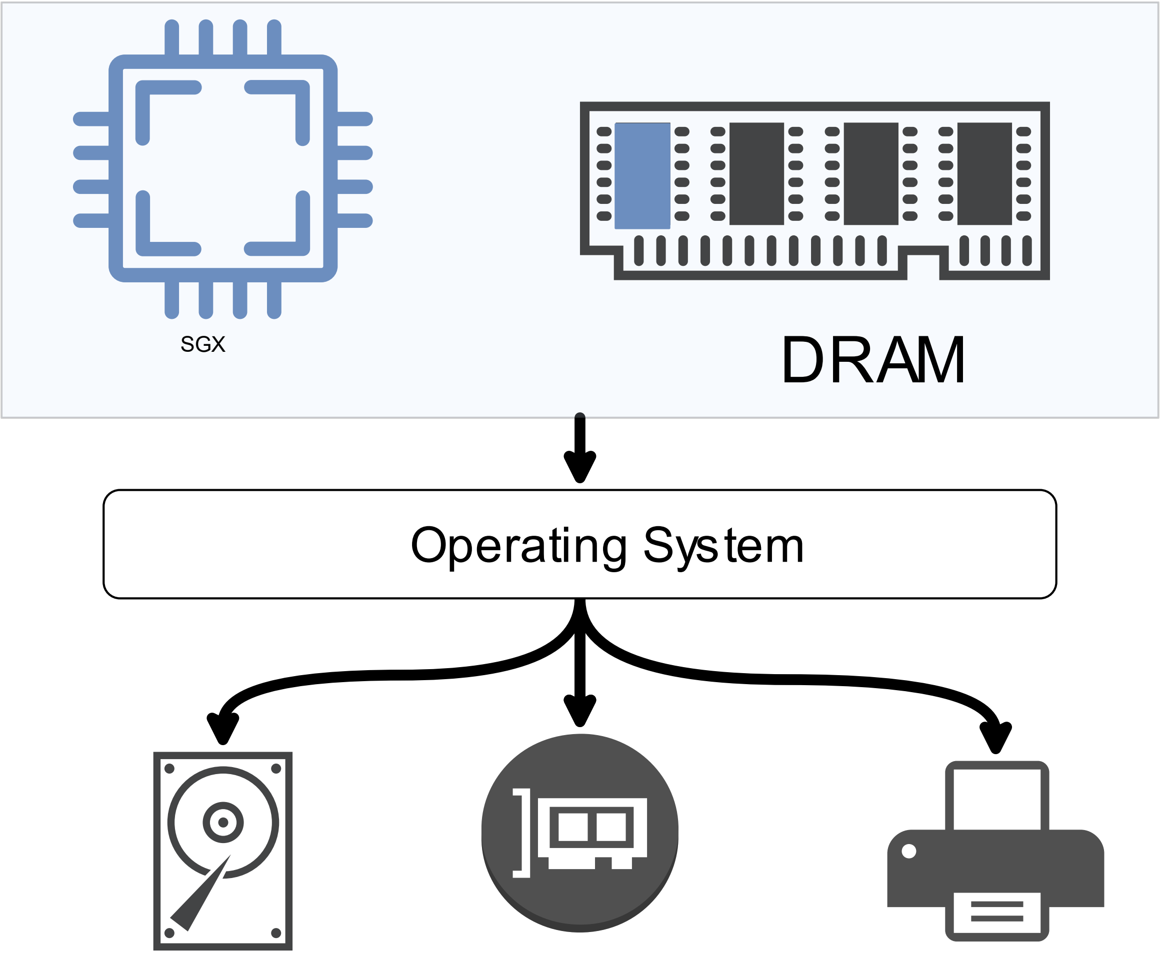
SGX transparently handles it.

- Faults are costly.



Intel SGX: Limitations

Intel SGX: Limitations

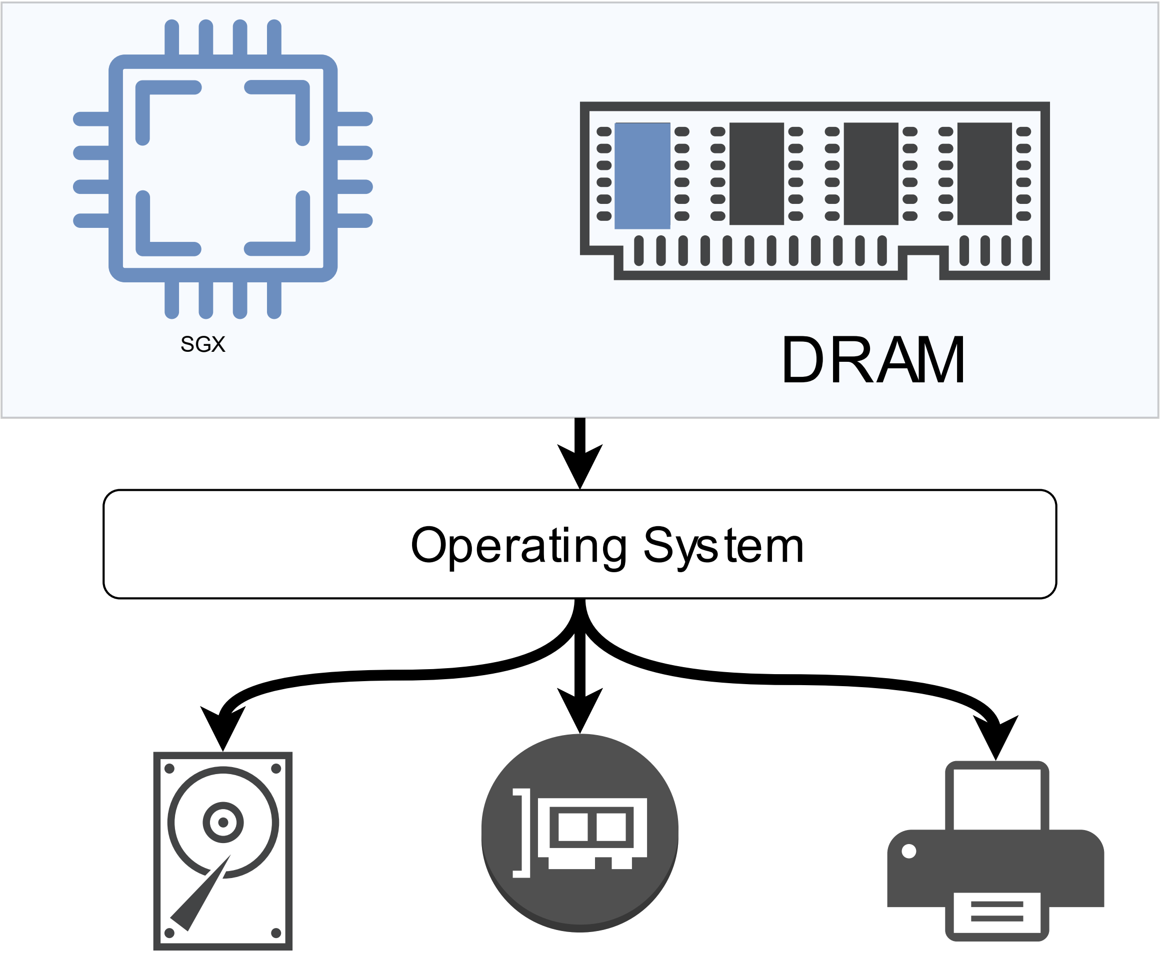


Intel SGX: Limitations

Operating System is NOT trusted.

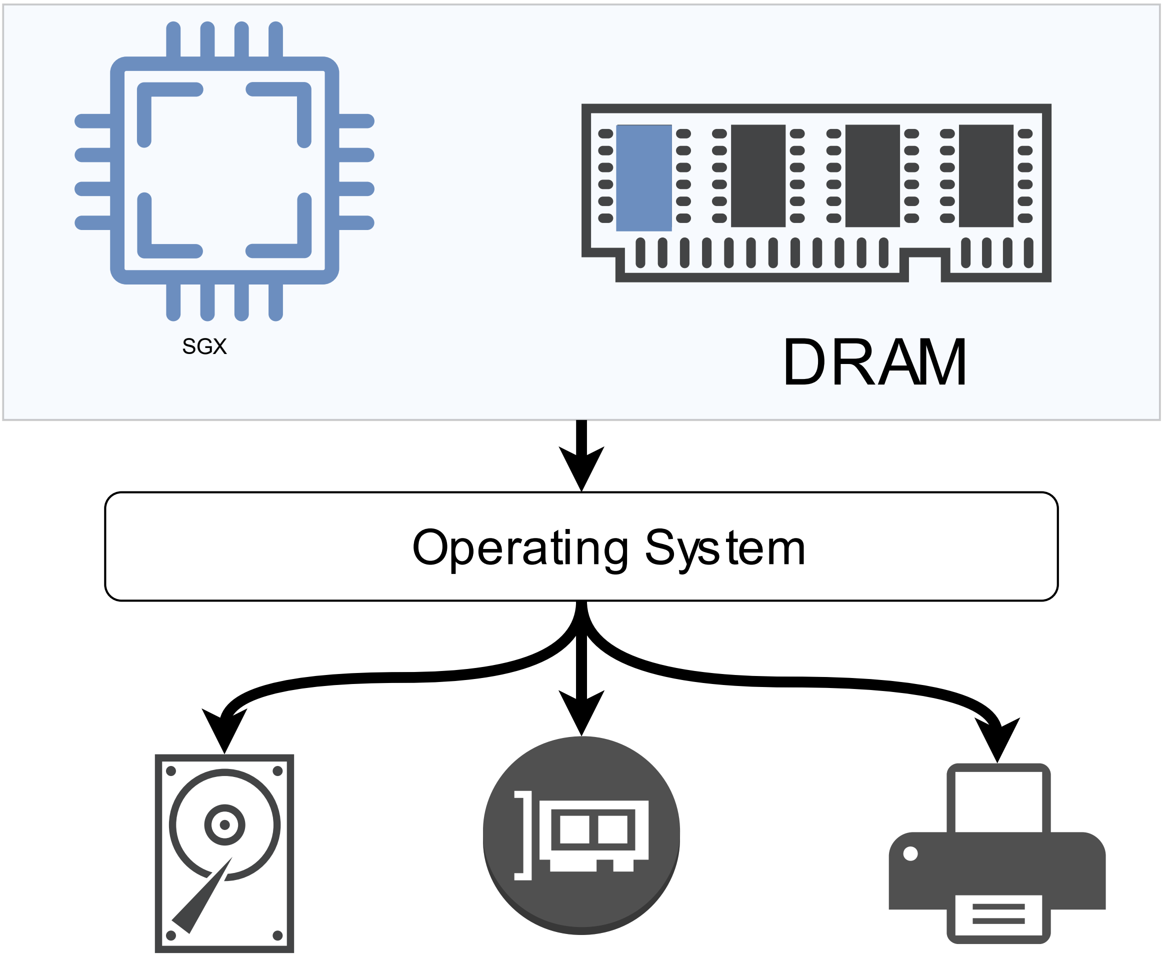
No direct system calls in SGX.

No secure file system access.

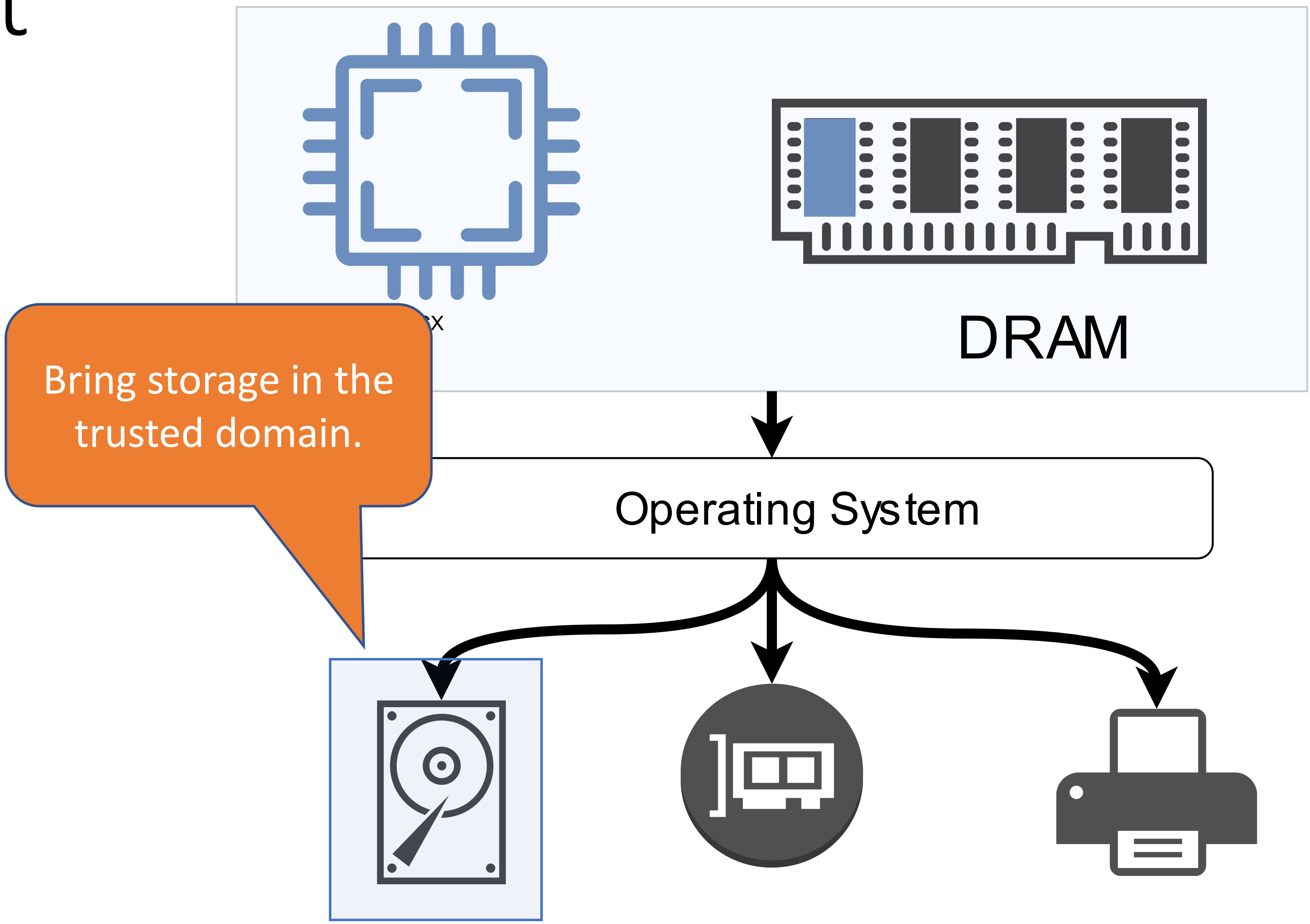


Problem Statement

Problem Statement

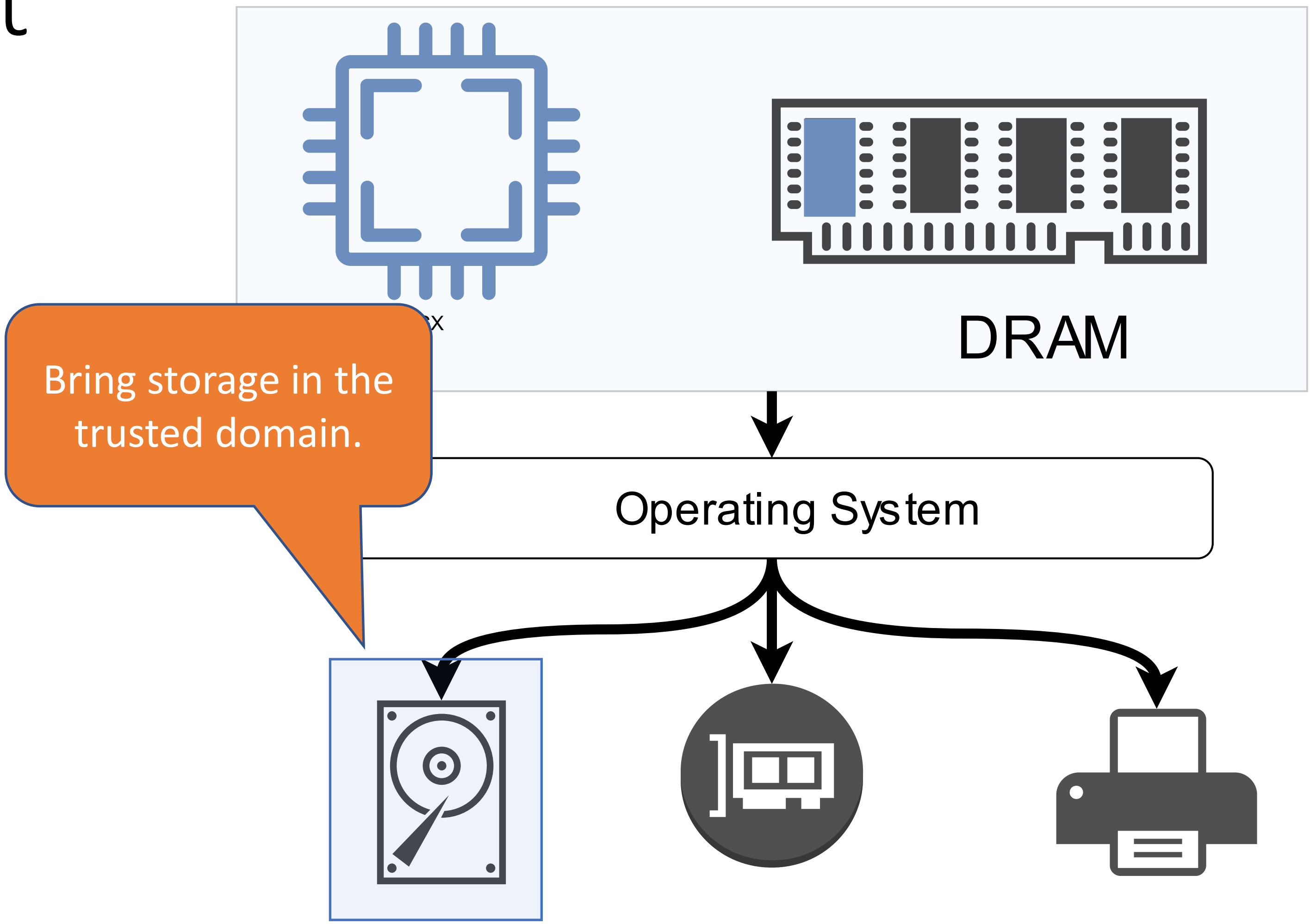


Problem Statement



Problem Statement

A fast and secure file system that is immune to replay attacks.



Related Work

[1]: Judicael B. Djoko, Jack Lange, and Adam J. Lee. 2019. NeXUS: Practical and Secure Access Control on Untrusted Storage Platforms using Client-Side SGX. DSN, 2019

[2]: Chia che Tsai, Donald E. Porter, and Mona Vij. 2017. Graphene-SGX: A Practical Library OS for Unmodified Applications on SGX. In USENIX Annual Technical Conference

[3]: Intel: https://software.intel.com/content/www/us/en/develop/articles/overview-of-intel-protected-file-system-library-using-software-guard-extensions.html

Where to store the data?

Where to store the data?

Encrypted file systems

- Data is encrypted prior to sending it to the disk.

In-Memory file systems

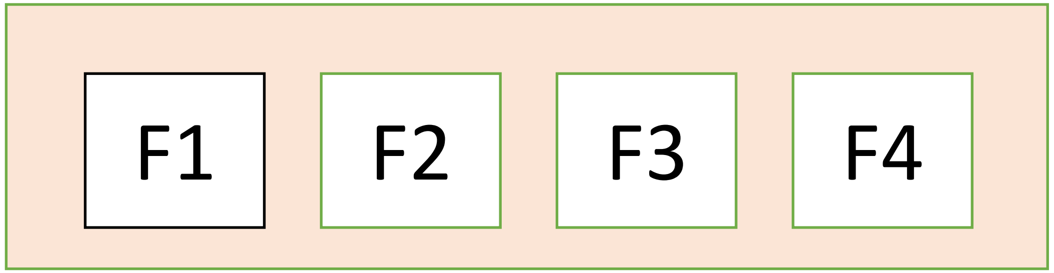
- An in-memory file system is maintained.

Hybrid file systems

- A combination of an encrypted file system and an in-memory file system

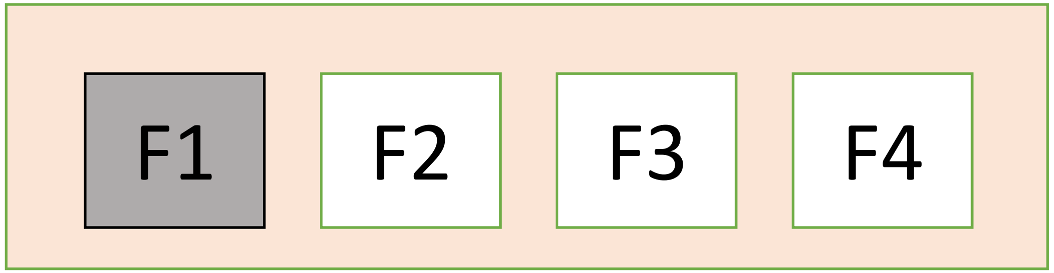
Hybrid design

File “F” is split into equal size “chunks”.



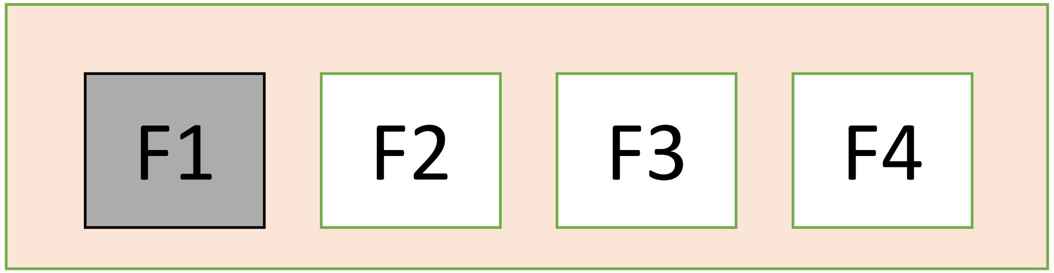
Hybrid design

File "F" is split into equal size "chunks".



Hybrid design

File “F” is split into equal size “chunks”.



$$H = Hash(D)$$

Hybrid design

File "F" is split into equal size "chunks".

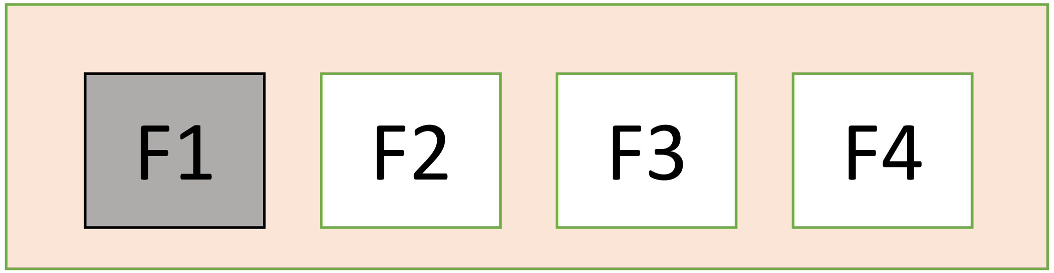
F1	F2	F3	F4

$$H = Hash(D)$$

$$Key(k) \leftarrow Random()$$

Hybrid design

File “F” is split into equal size “chunks”.



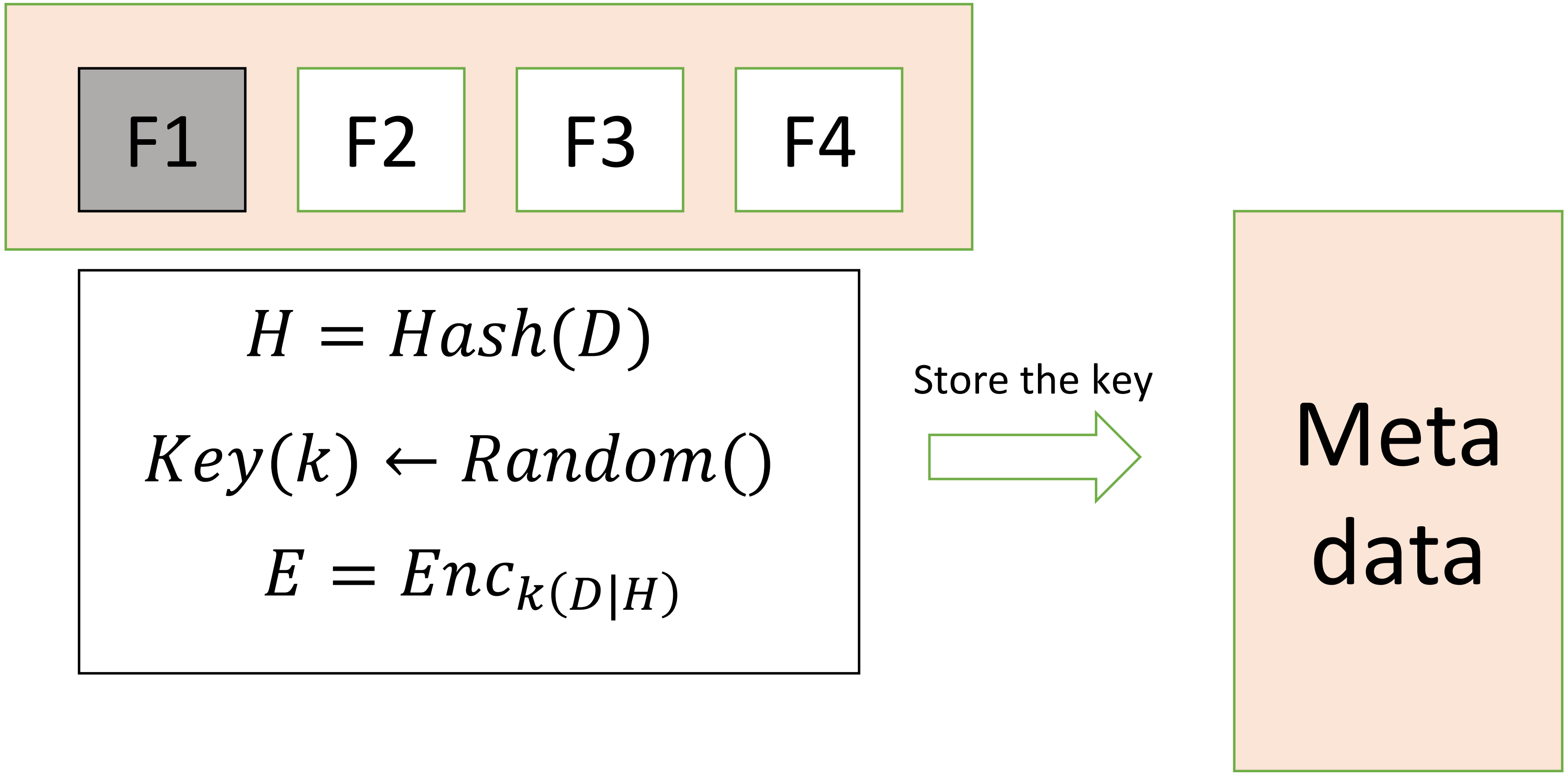
$$H = Hash(D)$$

$$Key(k) \leftarrow Random()$$

$$E = Enc_{k(D|H)}$$

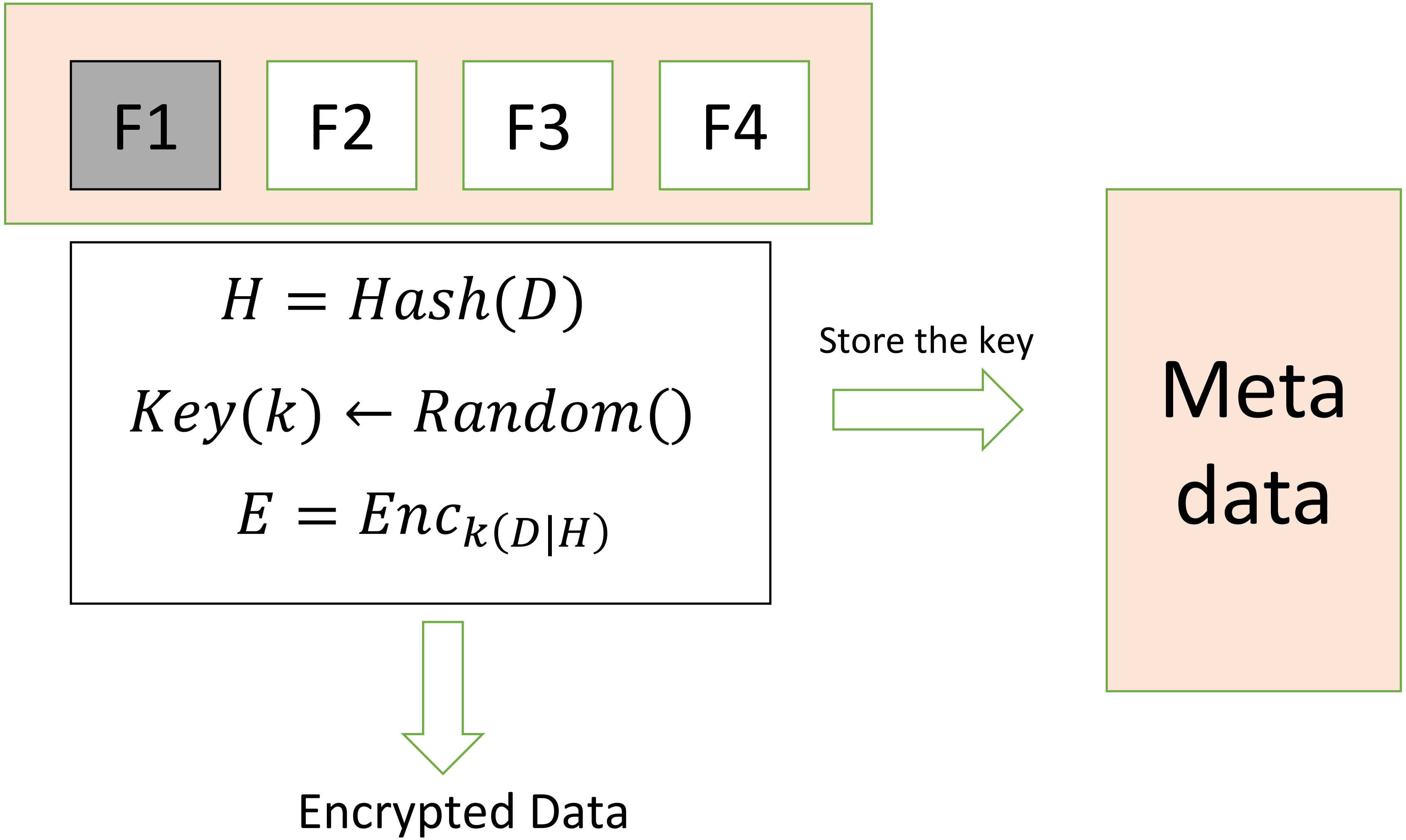
Hybrid design

File “F” is split into equal size “chunks”.



Hybrid design

File “F” is split into equal size “chunks”.



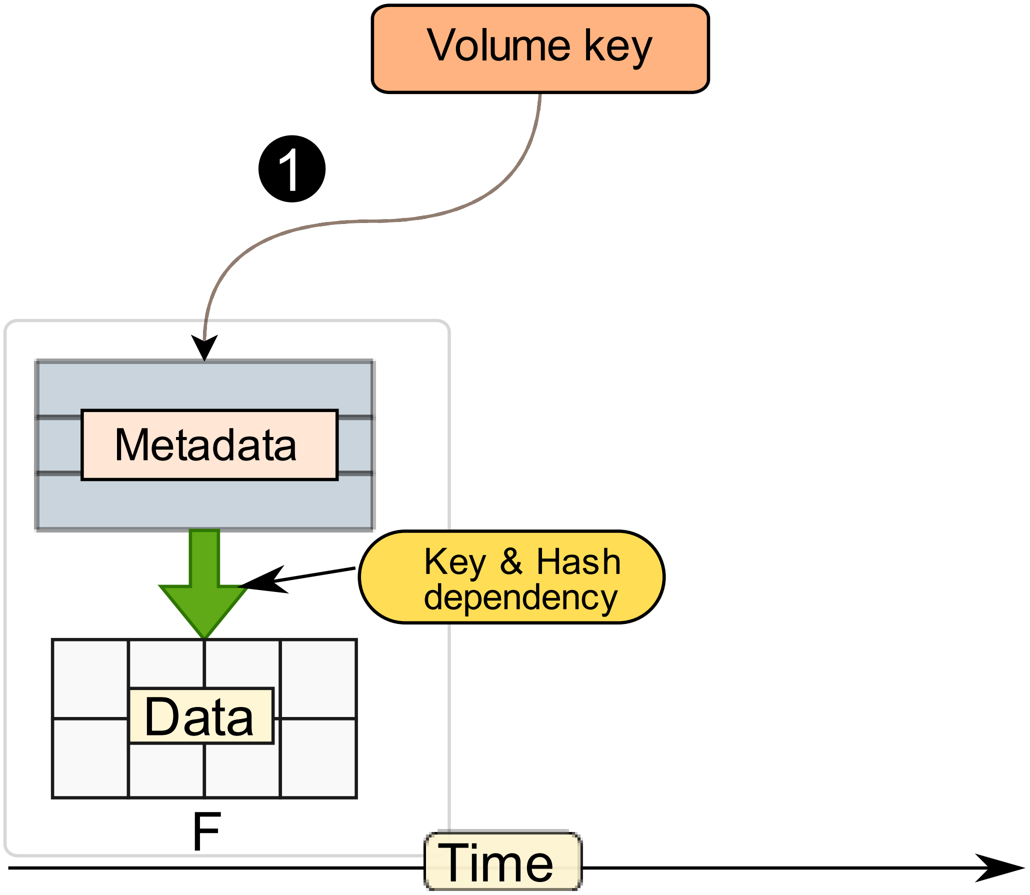
$$H = Hash(D)$$

$$Key(k) \leftarrow Random()$$

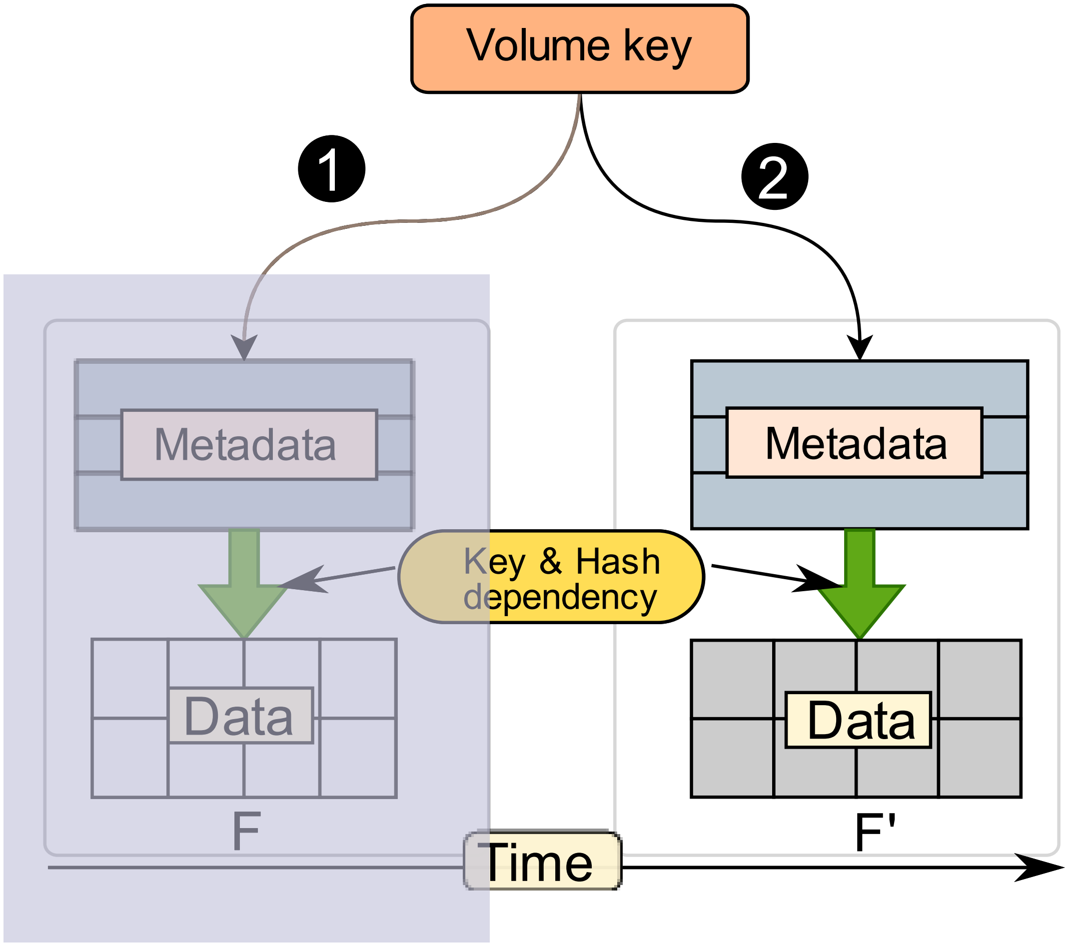
$$E = Enc_{k(D|H)}$$

Replay Attacks: Security Aspect

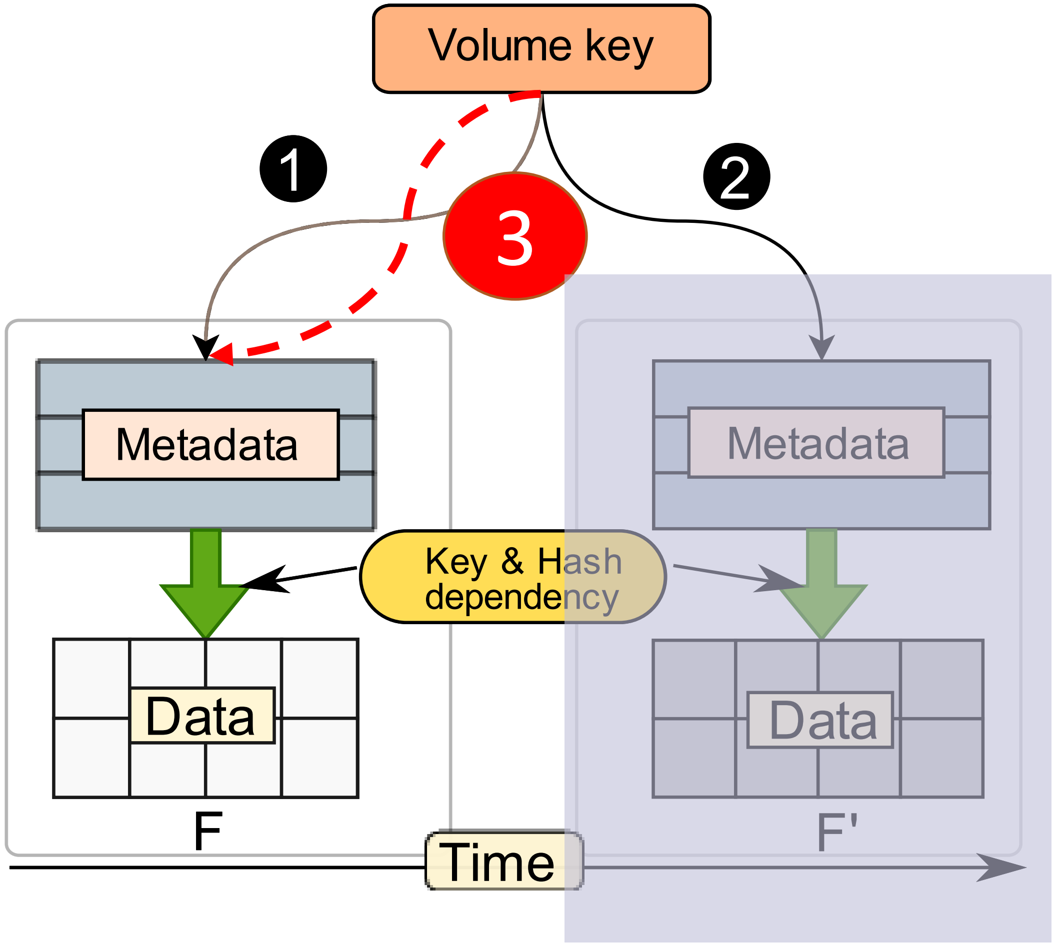
Replay Attacks:
Security Aspect



Replay Attacks:
Security Aspect

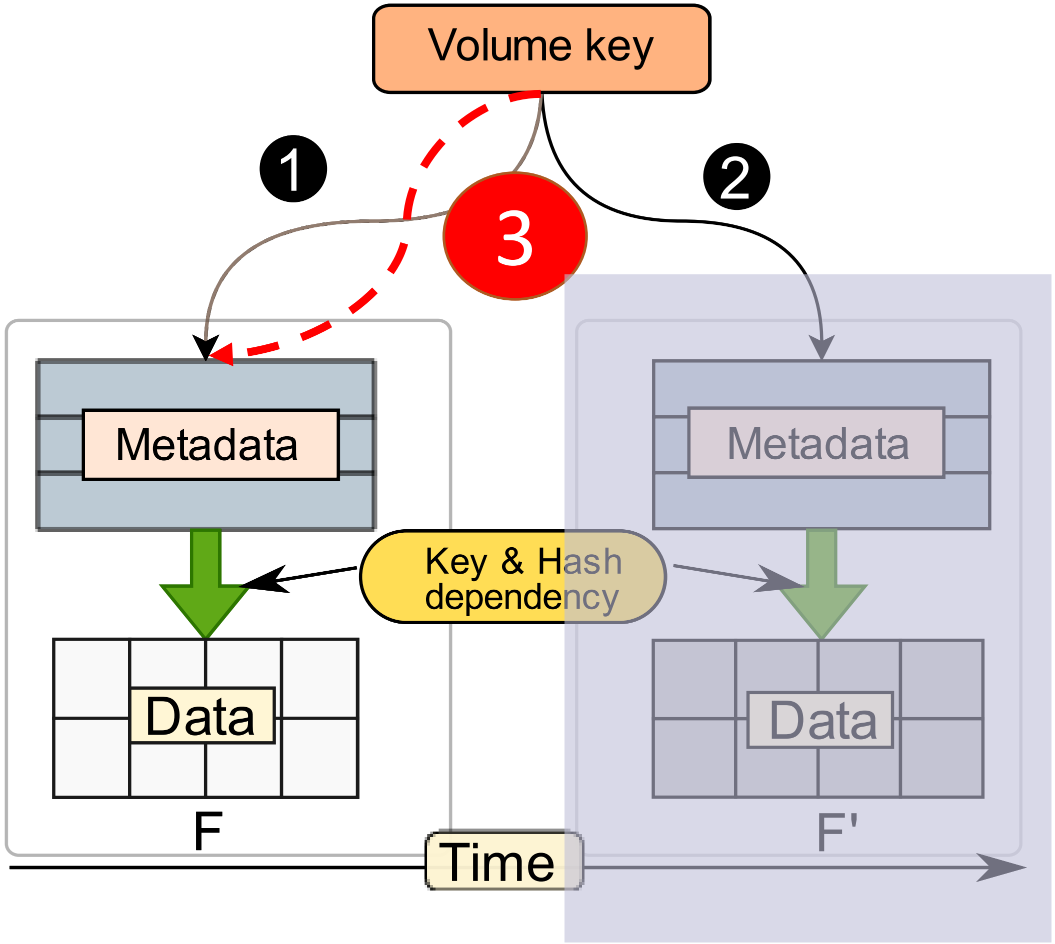


Replay Attacks:
Security Aspect

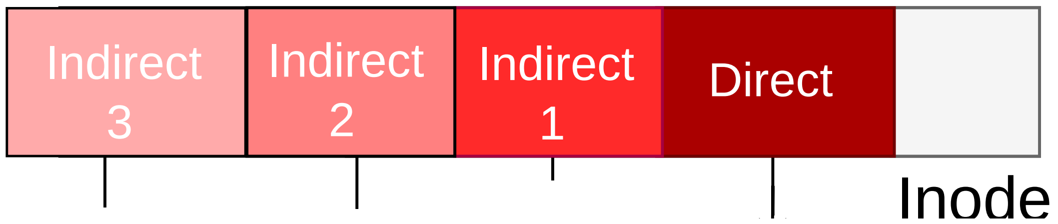


Replay Attacks:
Security Aspect

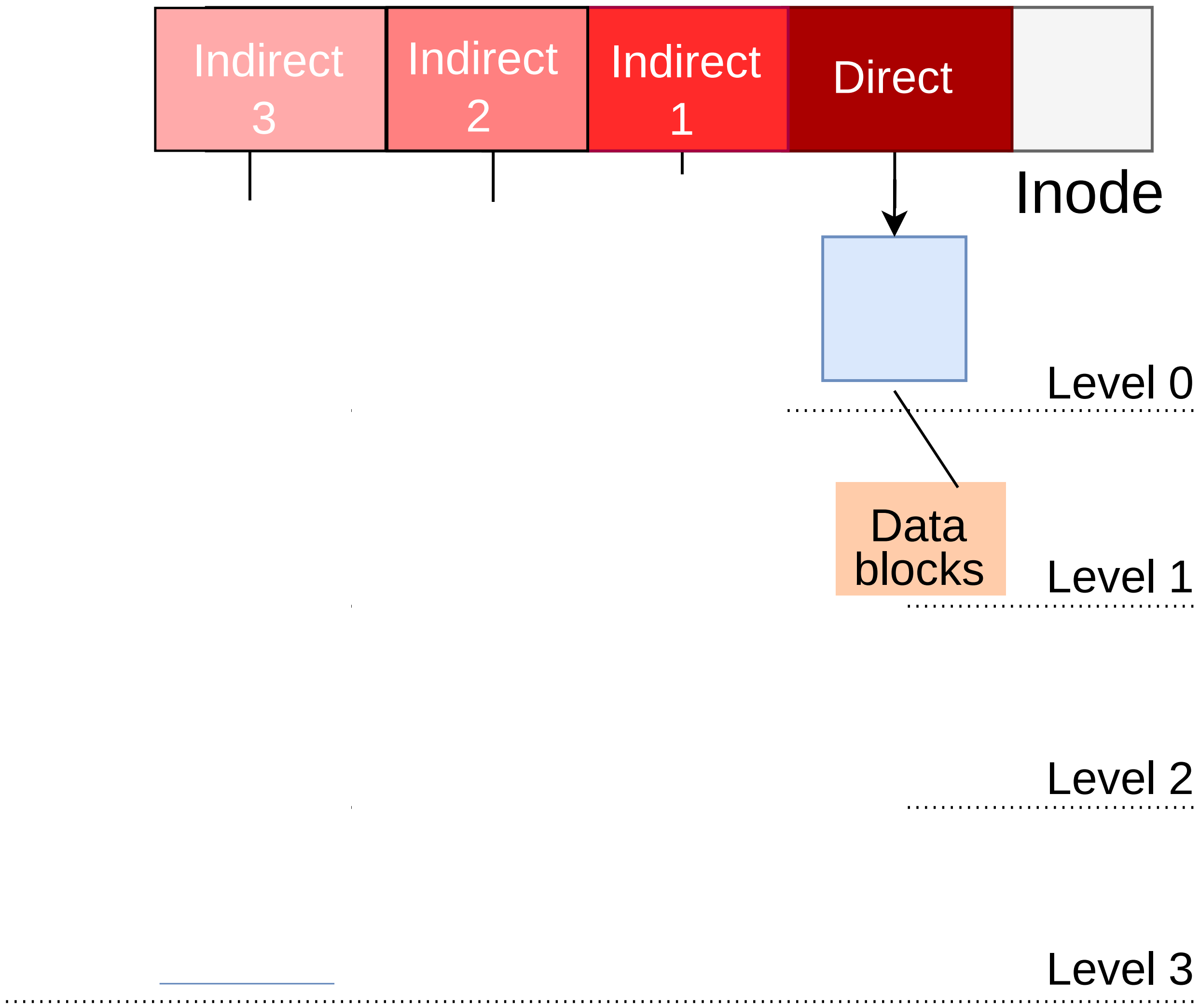
- We were able to mount this attack on the current state-of-the-arts [1,2,3] .
- Just encrypting data and metadata blocks is not enough.



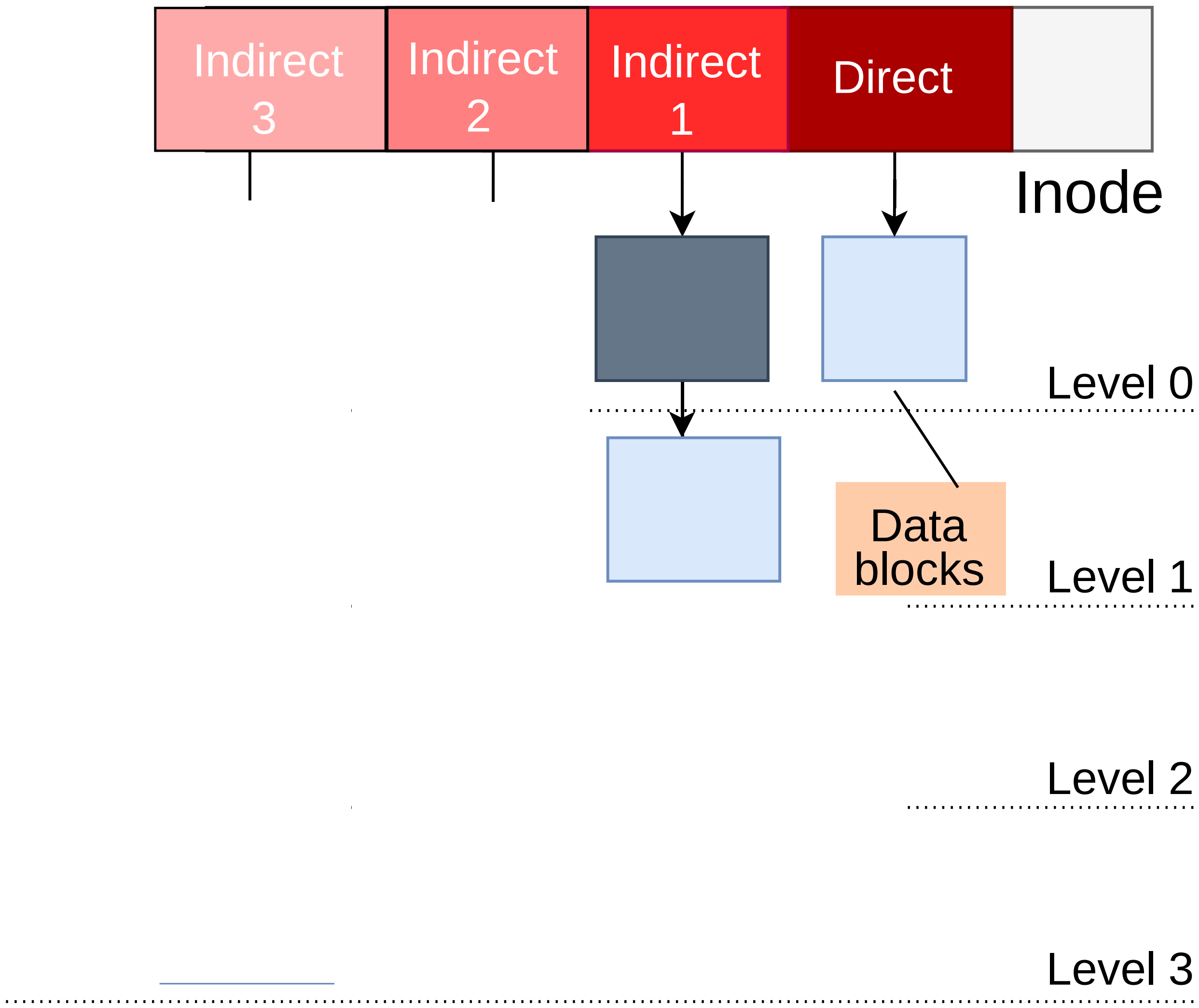
Inode-based
Design



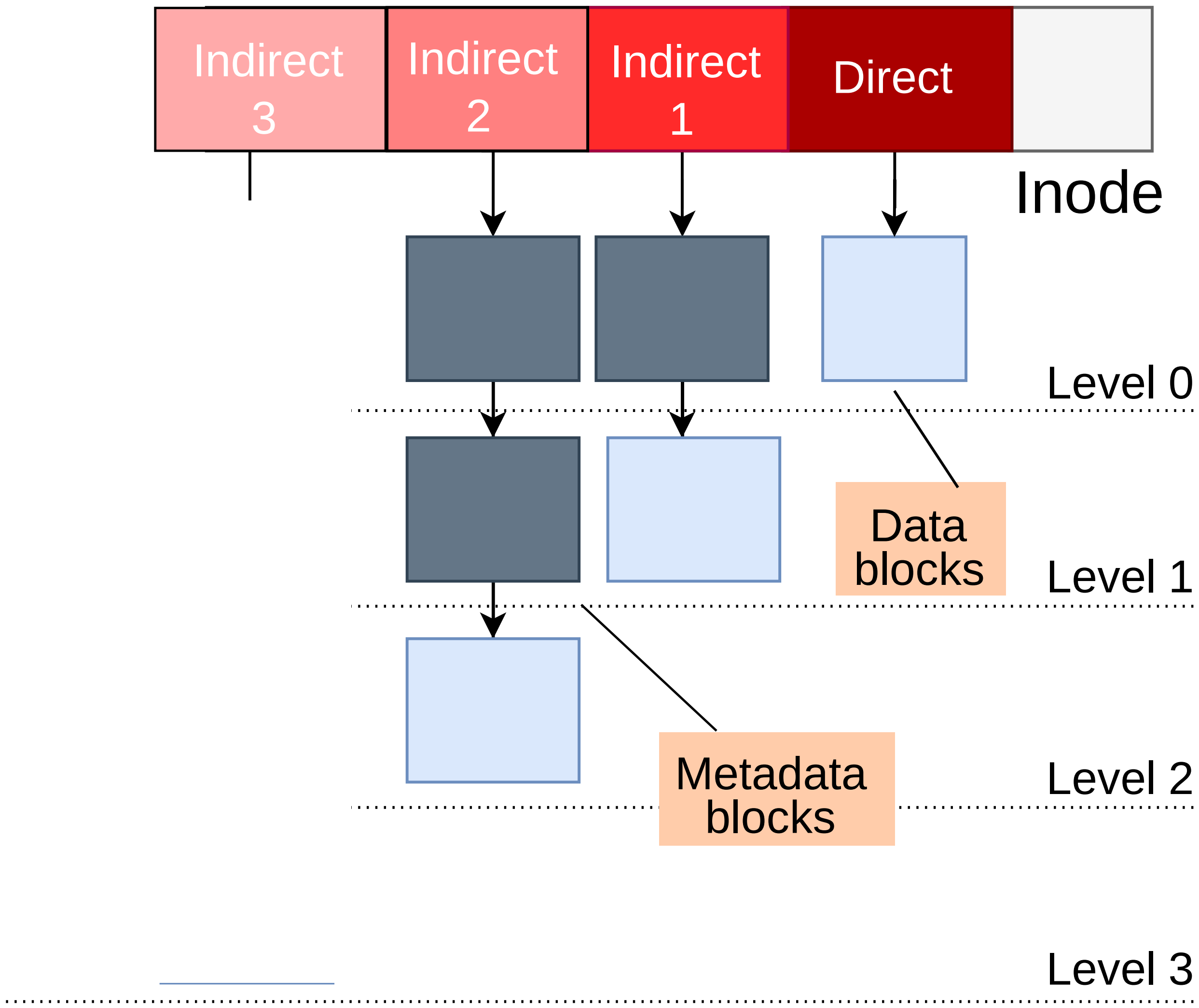
Inode-based
Design



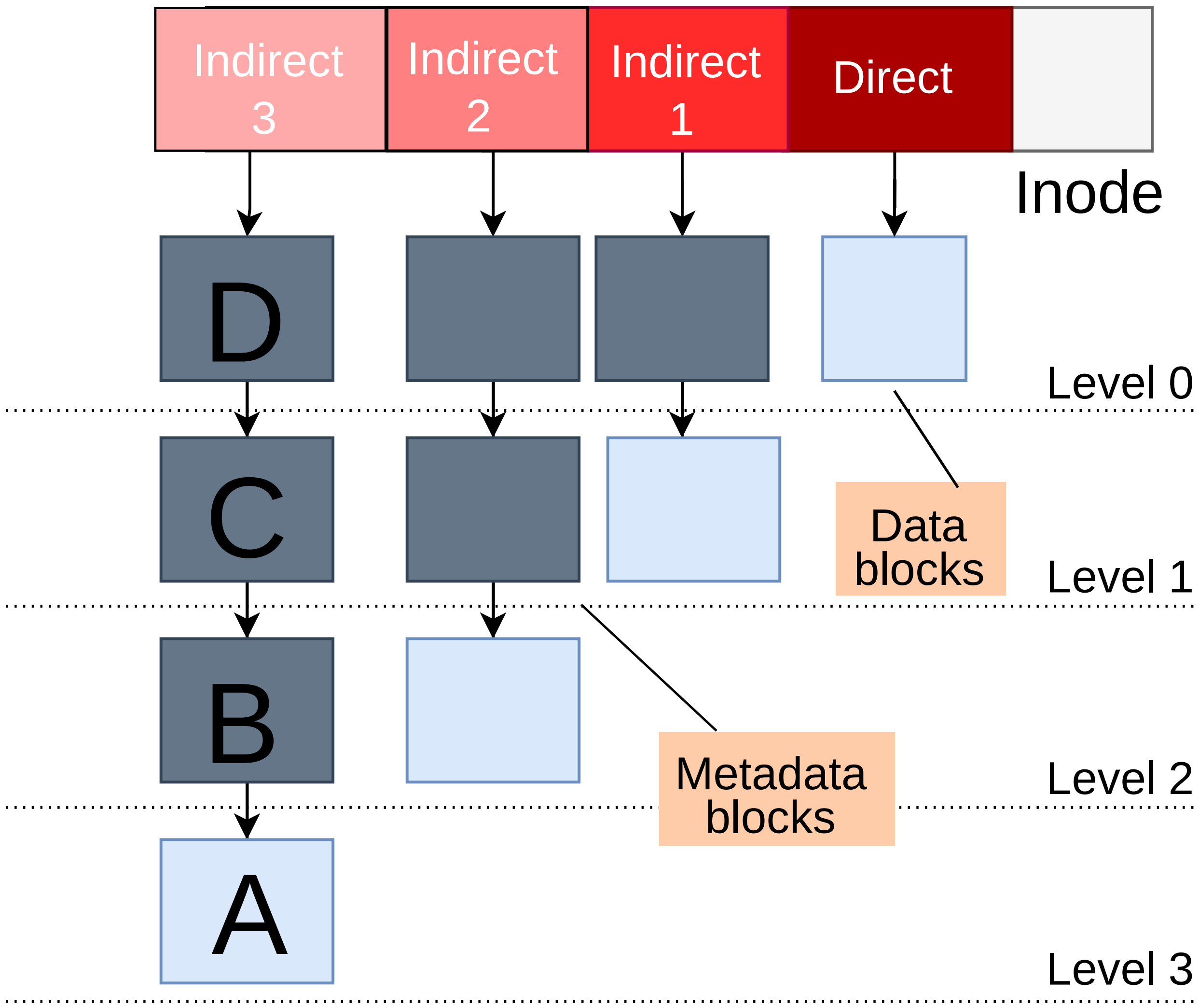
Inode-based
Design



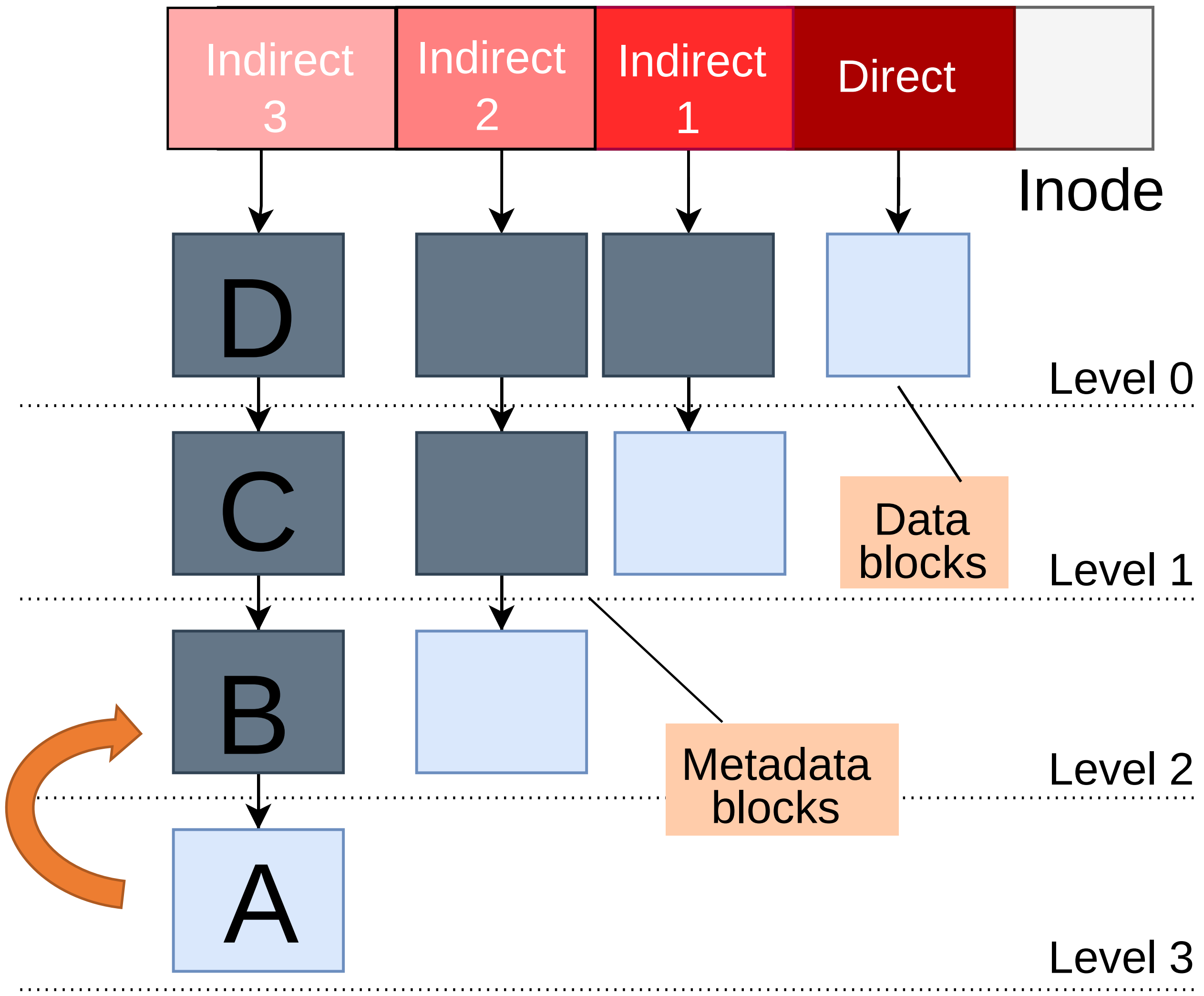
Inode-based
Design



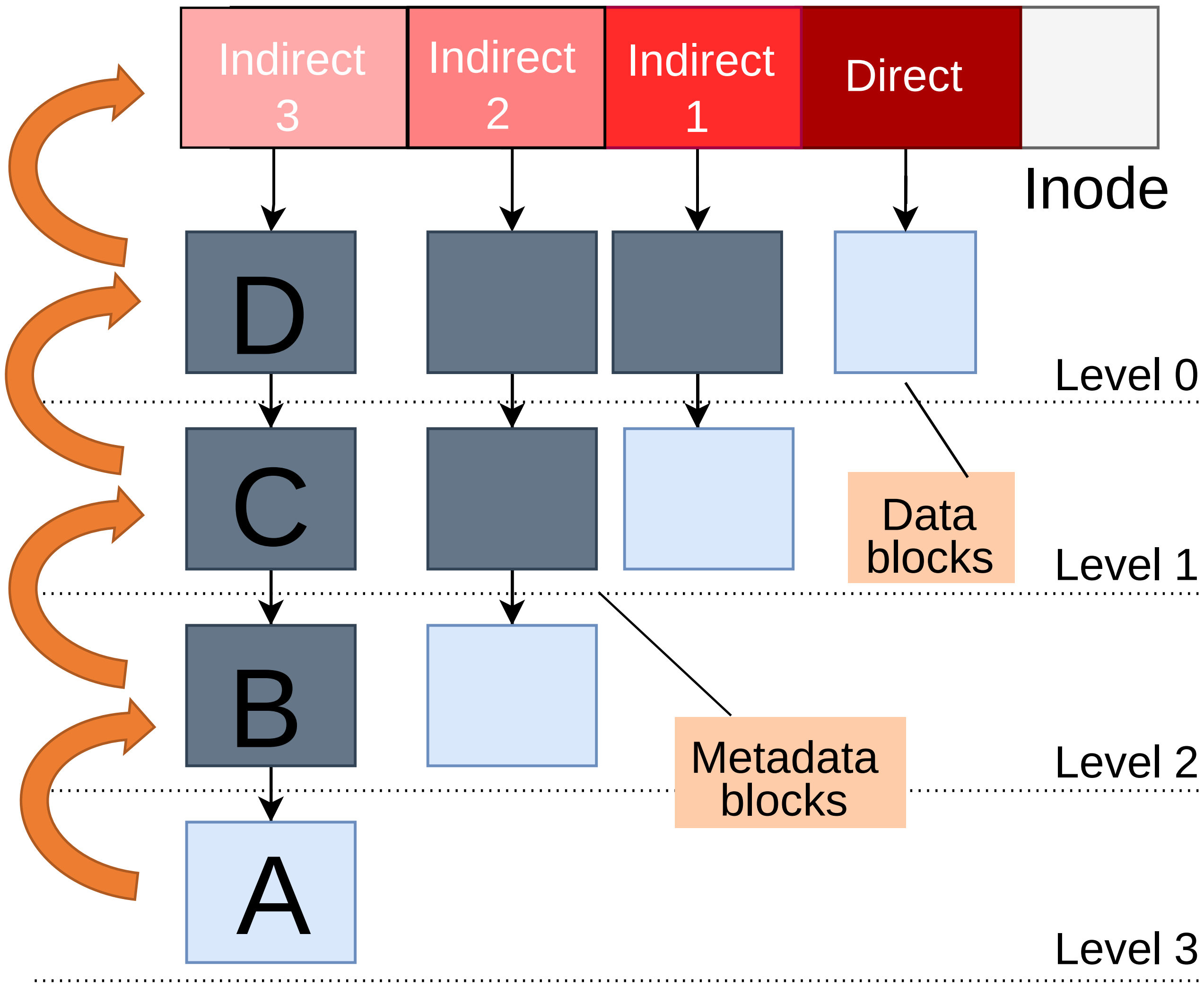
Inode-based Design



Inode-based
Design



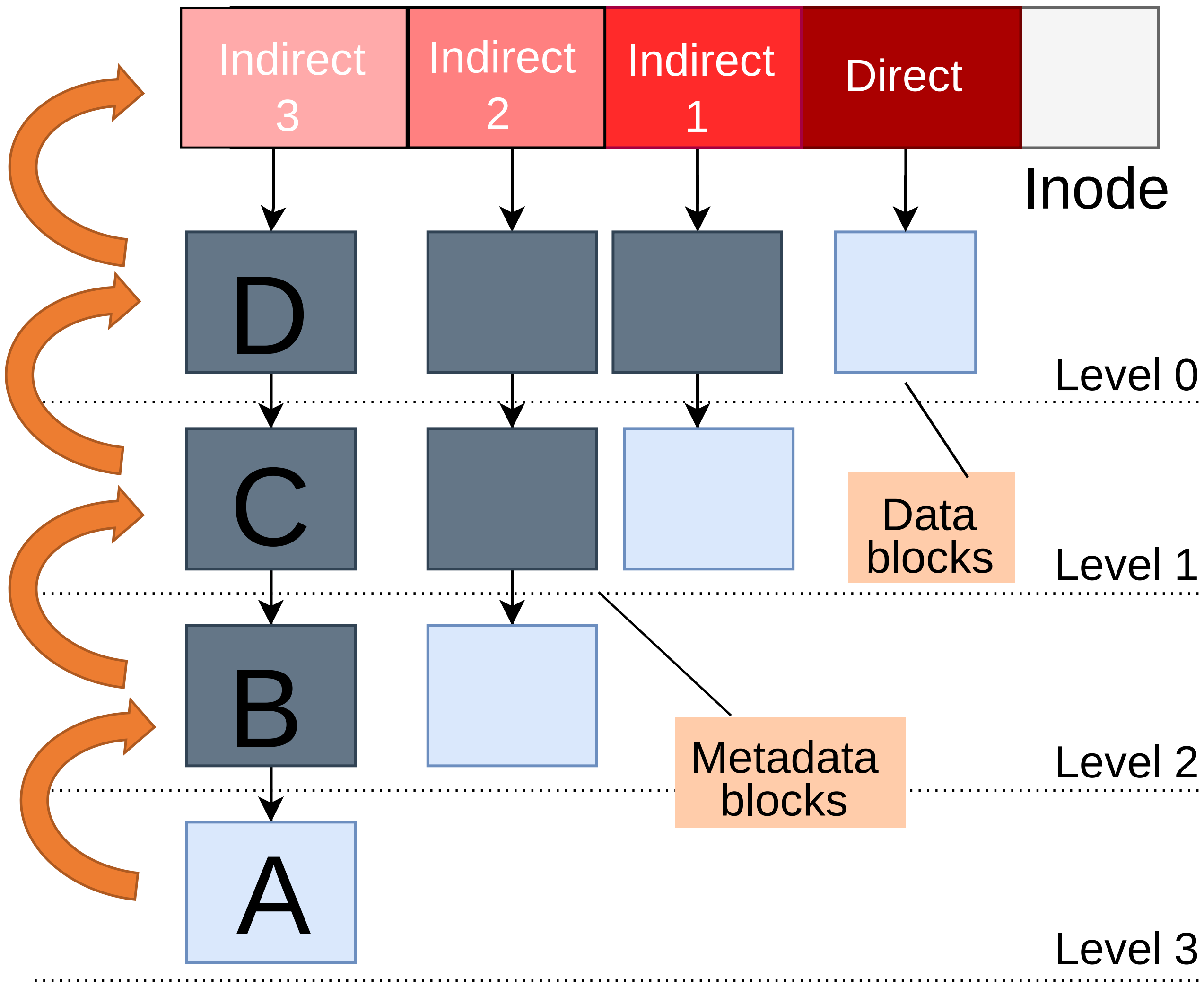
Inode-based
Design



Inode-based Design

- The keys are stored in the parent nodes.
- A write on the child node requires updating its parent – till the root.

Cascaded Updates



Compatibility Issues

Compatibility Issues

- **Intel Protected File System API:**
  - sgx_fopen
  - sgx_fopen_auto_key
  - sgx_fclose
  - sgx_fread
  - sgx_fwrite
  - sgx_fflush
  - sgx_ftell
  - sgx_fseek

Compatibility Issues

- Intel Protected Files:
  - Requires modification to the source code.
  - Vulnerable to replay attacks.

- **Intel Protected File System API:**
  - sgx_fopen
  - sgx_fopen_auto_key
  - sgx_fclose
  - sgx_fread
  - sgx_fwrite
  - sgx_fflush
  - sgx_ftell
  - sgx_fseek

Key Takeaways

Key Takeaways

Replay attacks in a secure file system violates the freshness property of the file system.

- It's a non-trivial issue as Intel SGX semantics does not provide freshness guarantees for data on rest.

An inode-based file system, though optimal for modern file systems, does not meet the requirement of a secure file system.

- There is a need for a new metadata management system.

The file system should be backward compatible and should work without any source code modifications.

SecureFS Design: Characterization

What is expected from a secure file system?

Workloads

Workloads

Domain	Benchmark
Database/Datastores	SQLite
	Redis
	MongoDB
Machine learning & Deep learning	CNN
	SVM
License managers	License3j
	OpenSSL
Block chain	Bitcoin
	Libcatena
Web services	Lighttpd
	Memcached

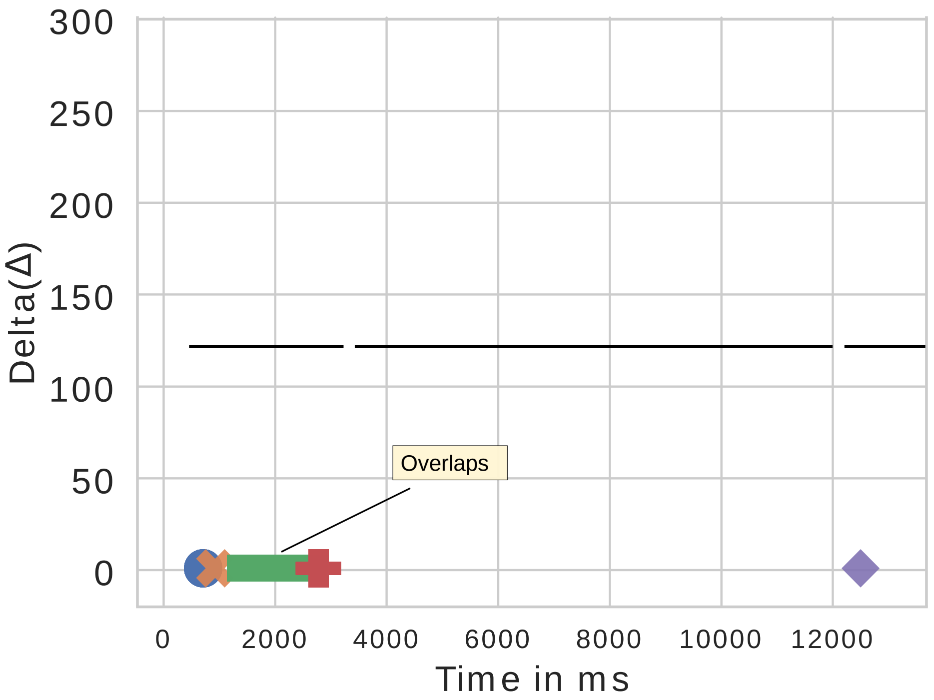
Interaction with the File System

Interaction with the File System

- Read and write pattern.
- Size of each access.
- #Read vs #Write
- Percentage of a file accessed.

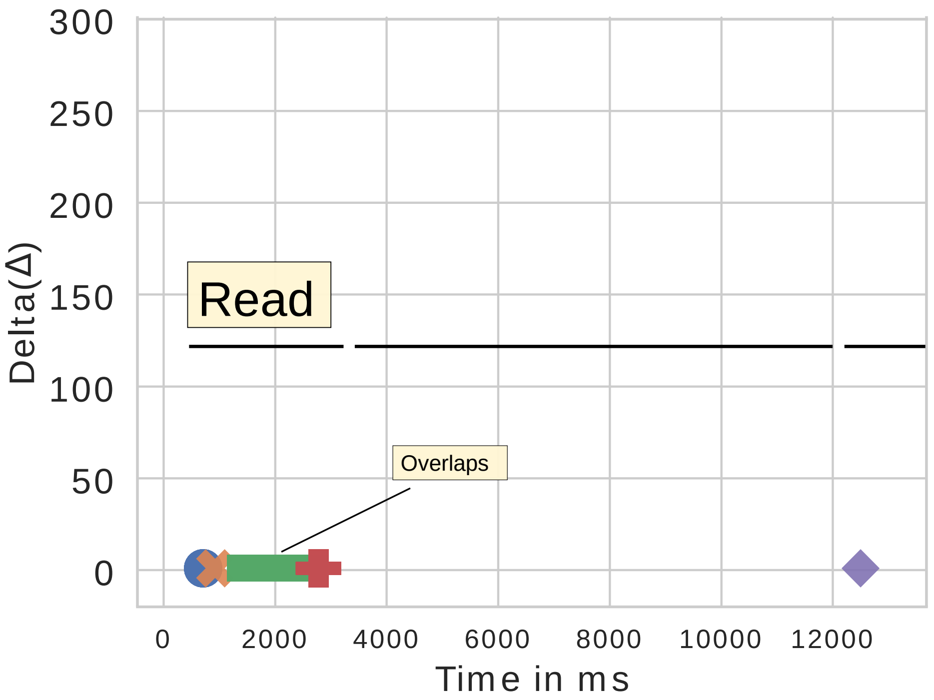
Typical Access Pattern

Typical Access Pattern



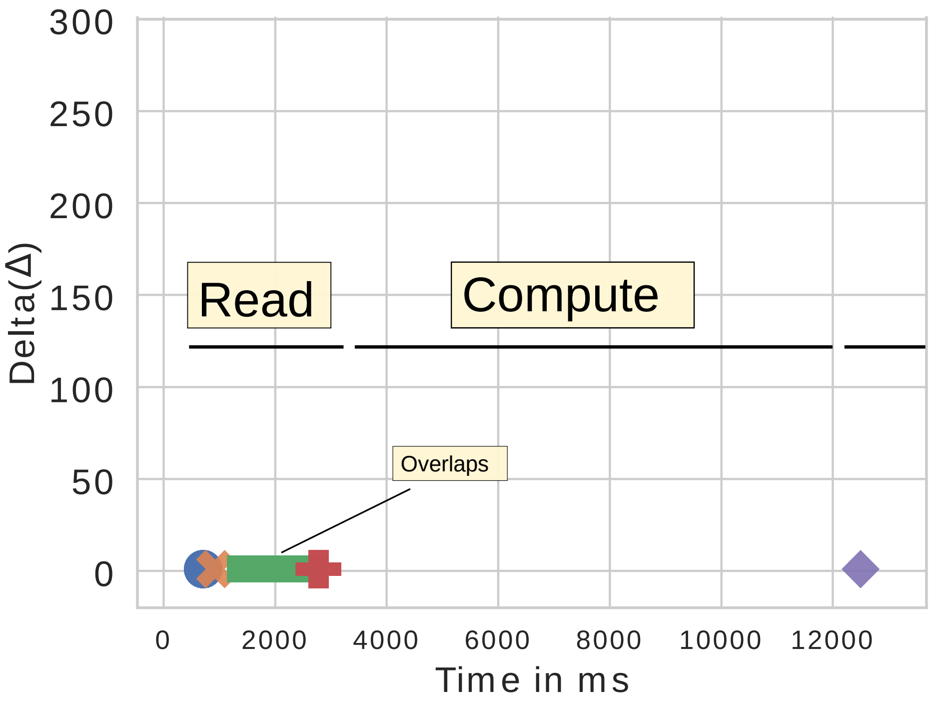
Sequential Data Access

Typical Access Pattern



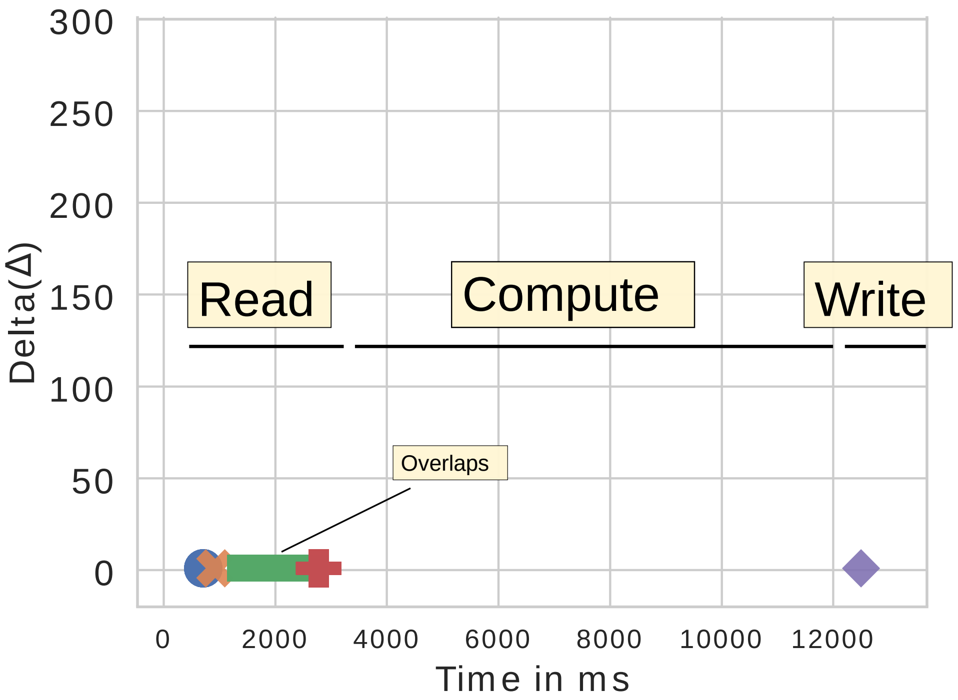
Sequential Data Access

Typical Access Pattern



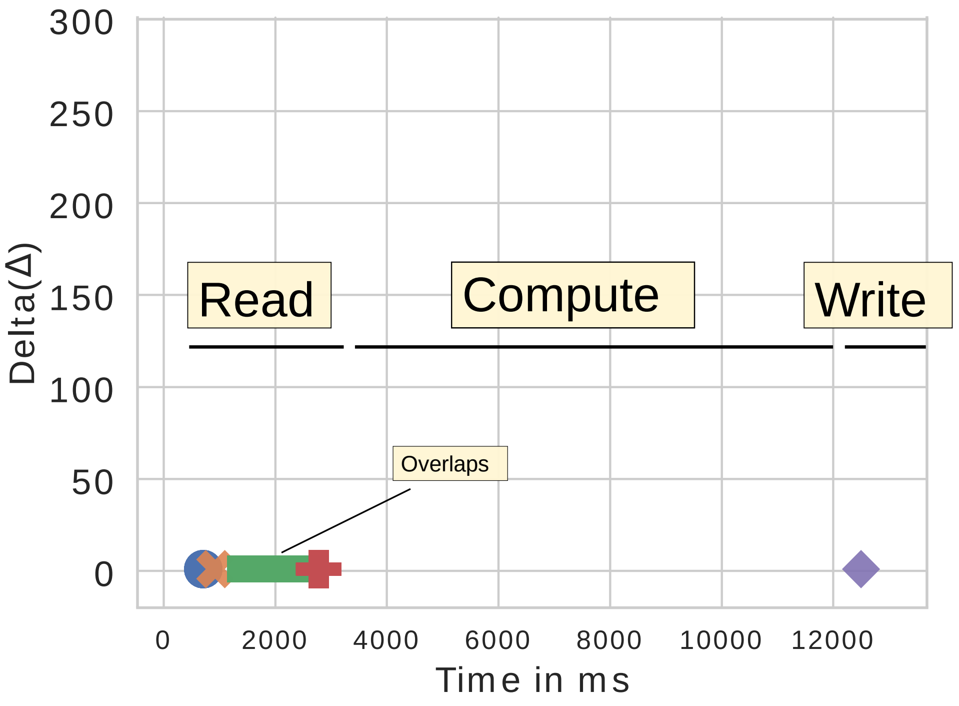
Sequential Data Access

Typical Access Pattern

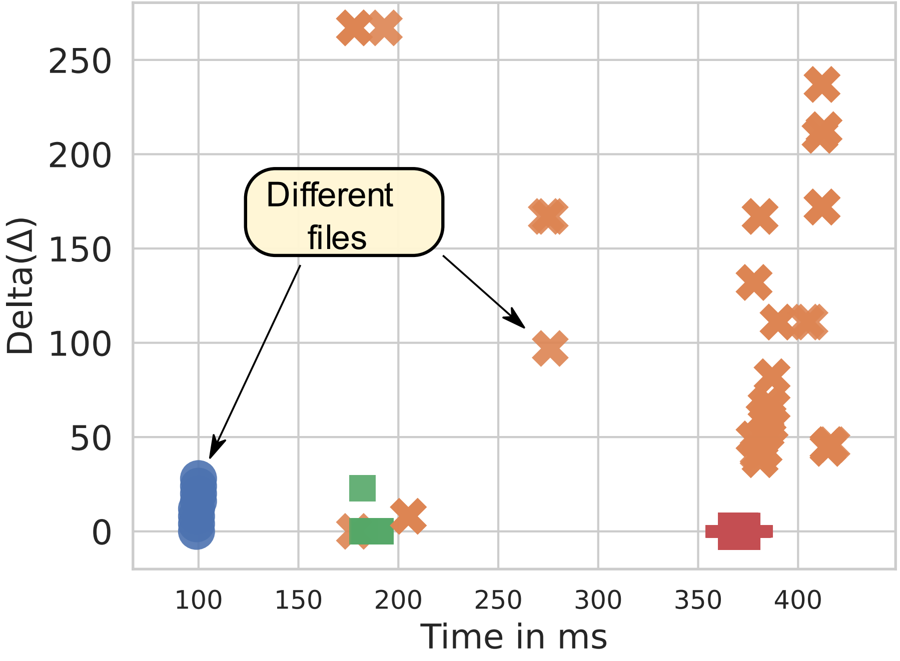


Sequential Data Access

Typical Access Pattern



Sequential Data Access



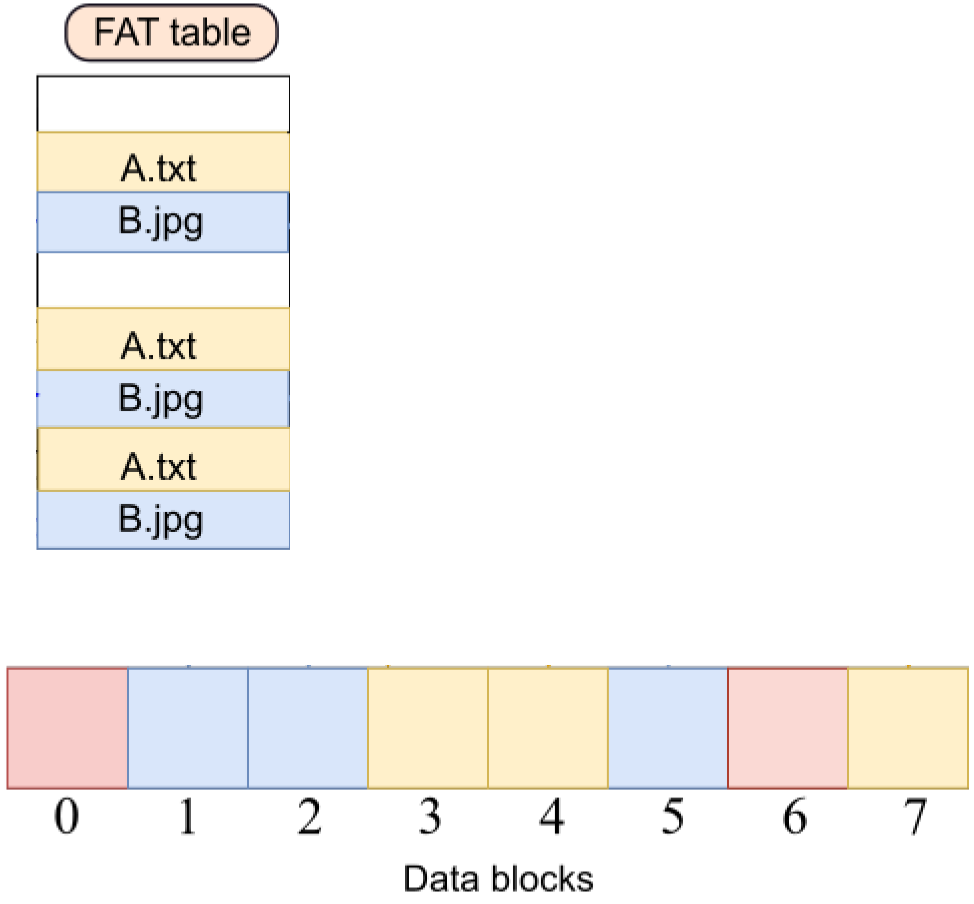
Random Data Access

SecureFS Design

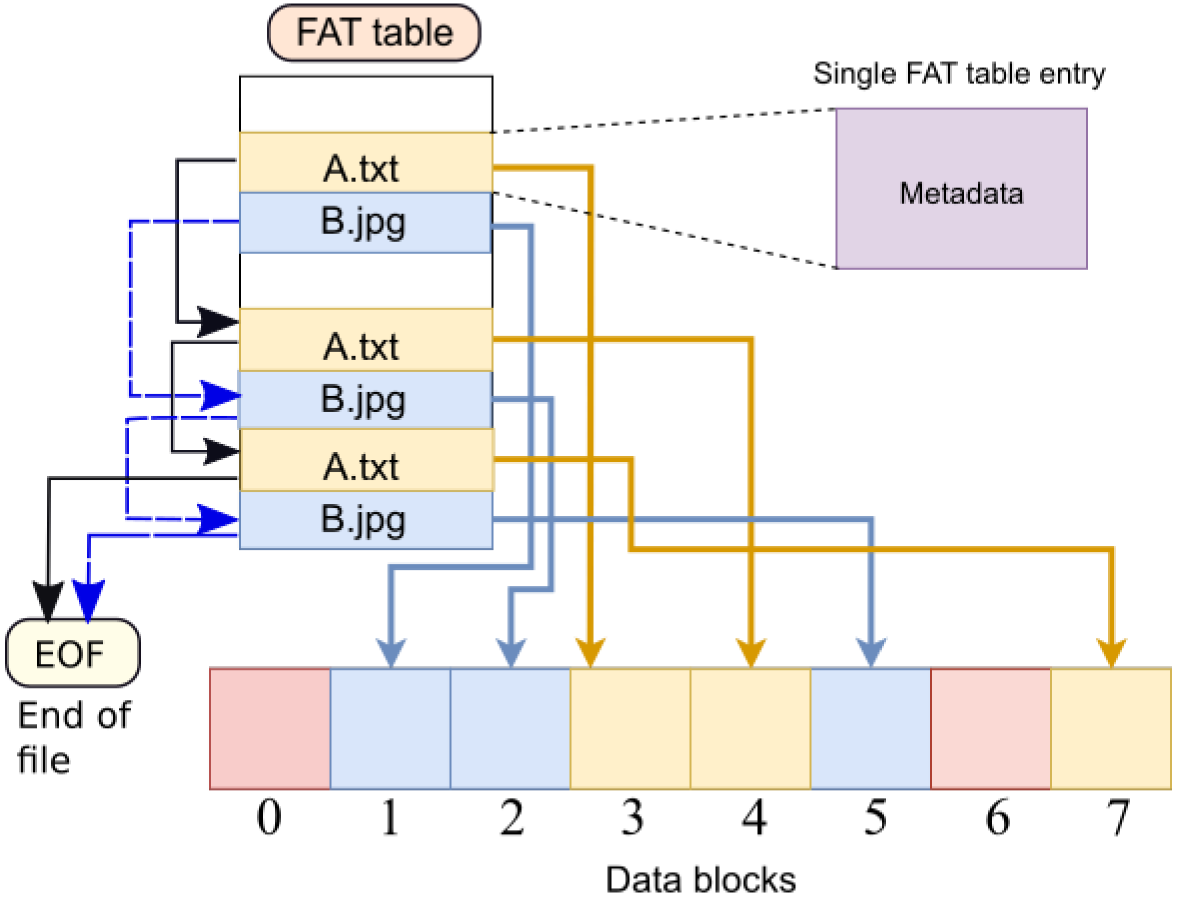
Performance Aspect

A FAT based file system

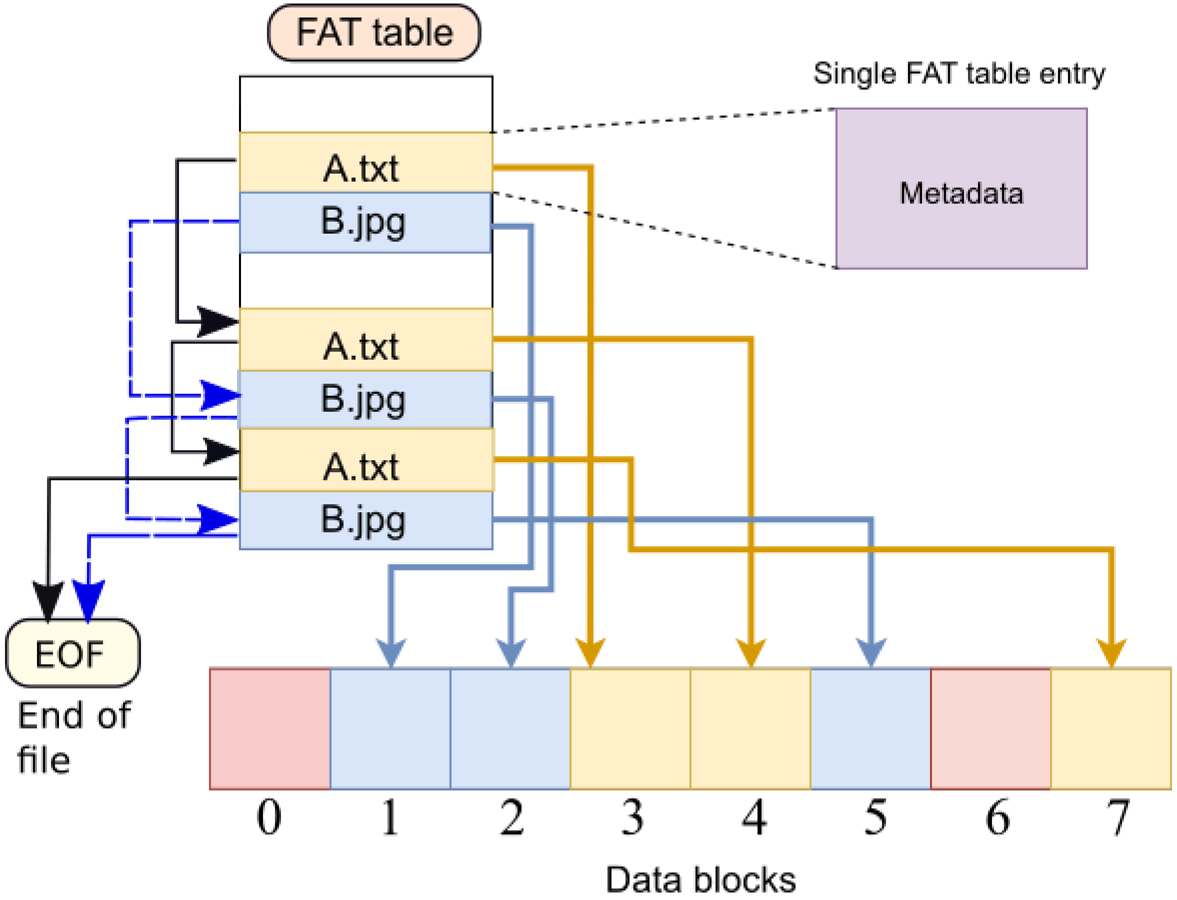
A FAT based file system



A FAT based file system



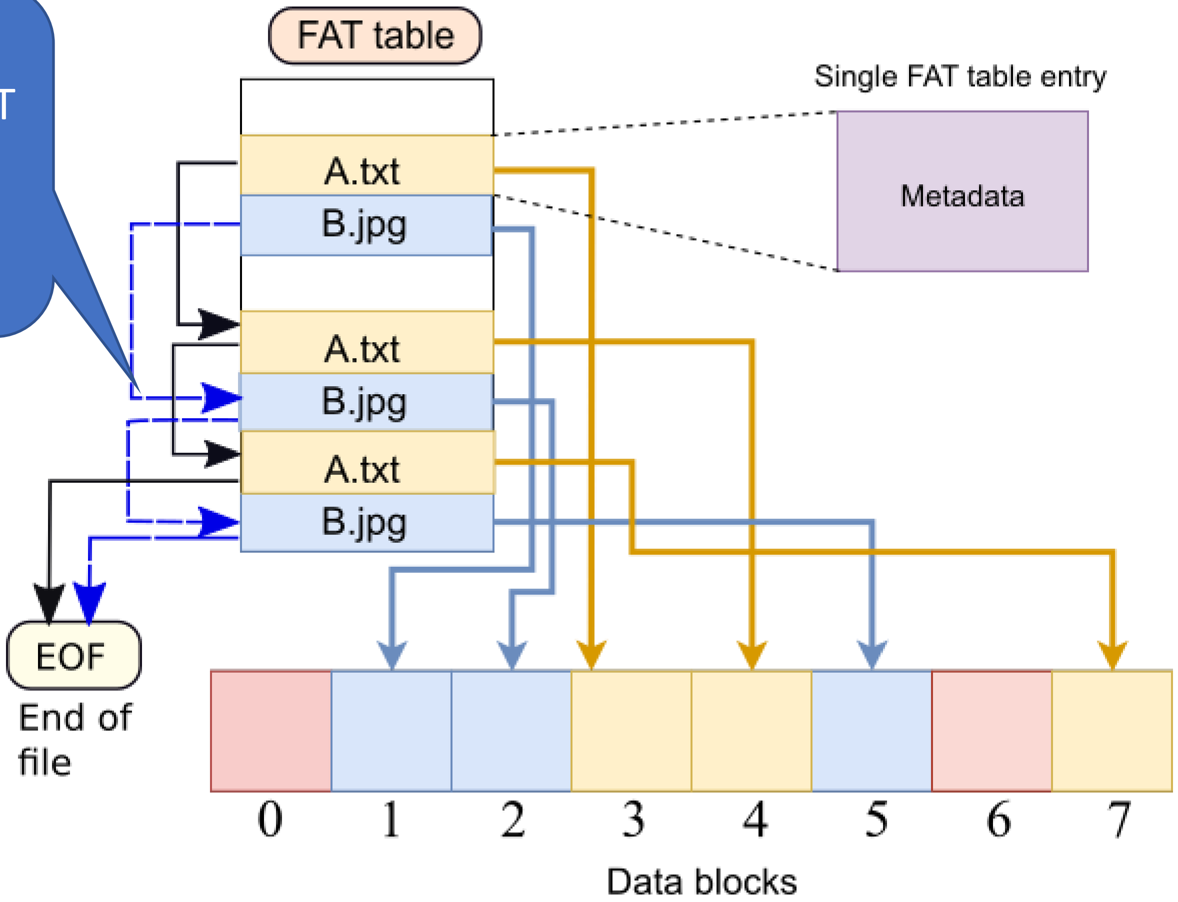
A FAT based file system



A single update in the FAT-entry is enough.

A FAT based file system

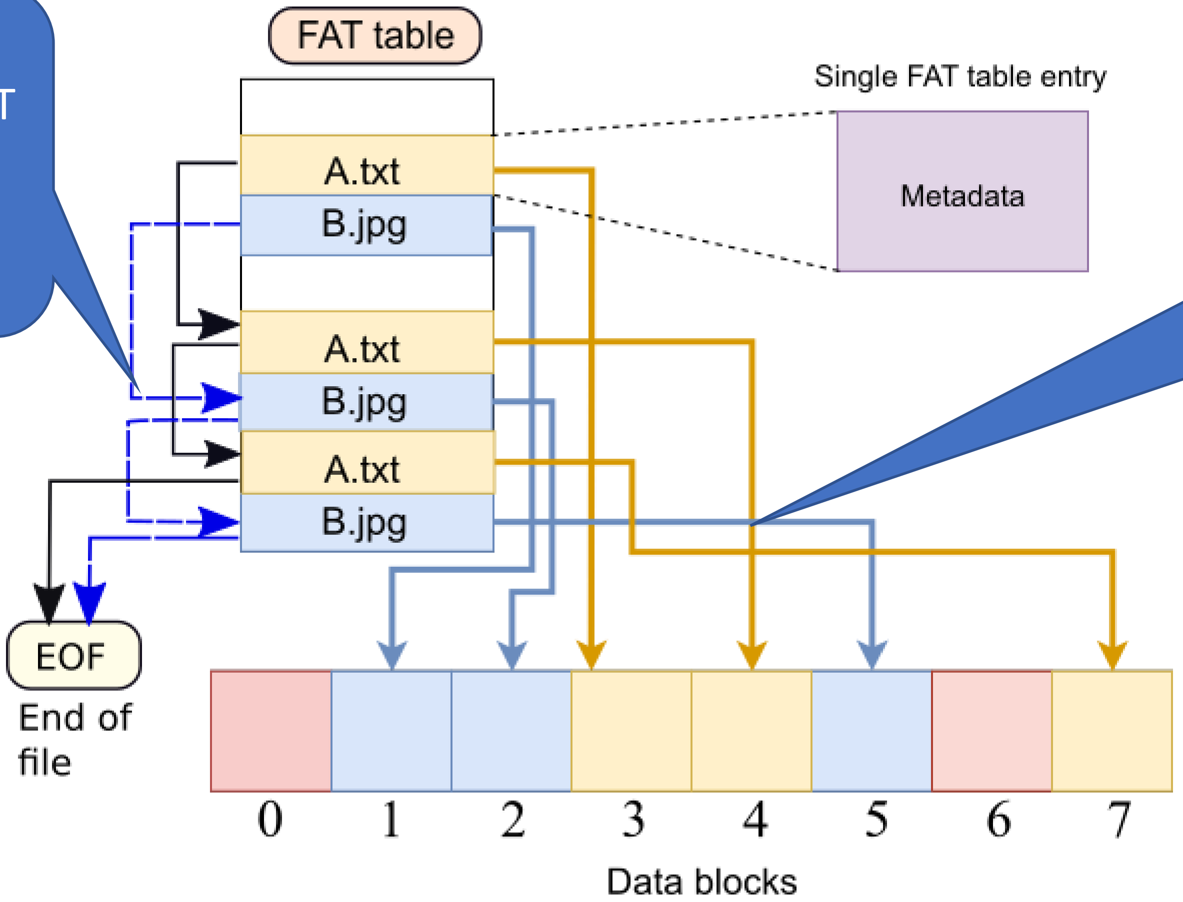
Pointers with-in the FAT table points to chunks of files



A single update in the FAT-entry is enough.

A FAT based file system

Pointers with-in the FAT table points to chunks of files



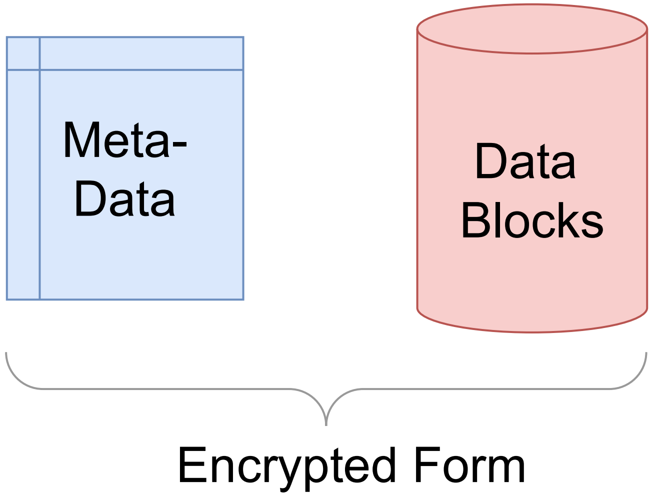
One-to-one mapping between the FAT table entries and the data chunks

A single update in the FAT-entry is enough.

SecureFS Design

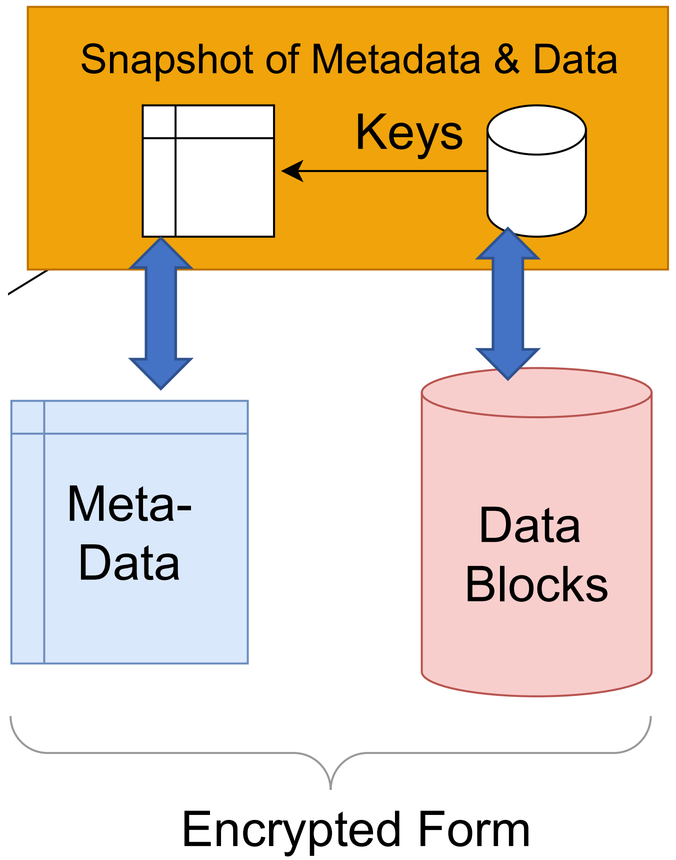
Security Aspect

Root-of-trust



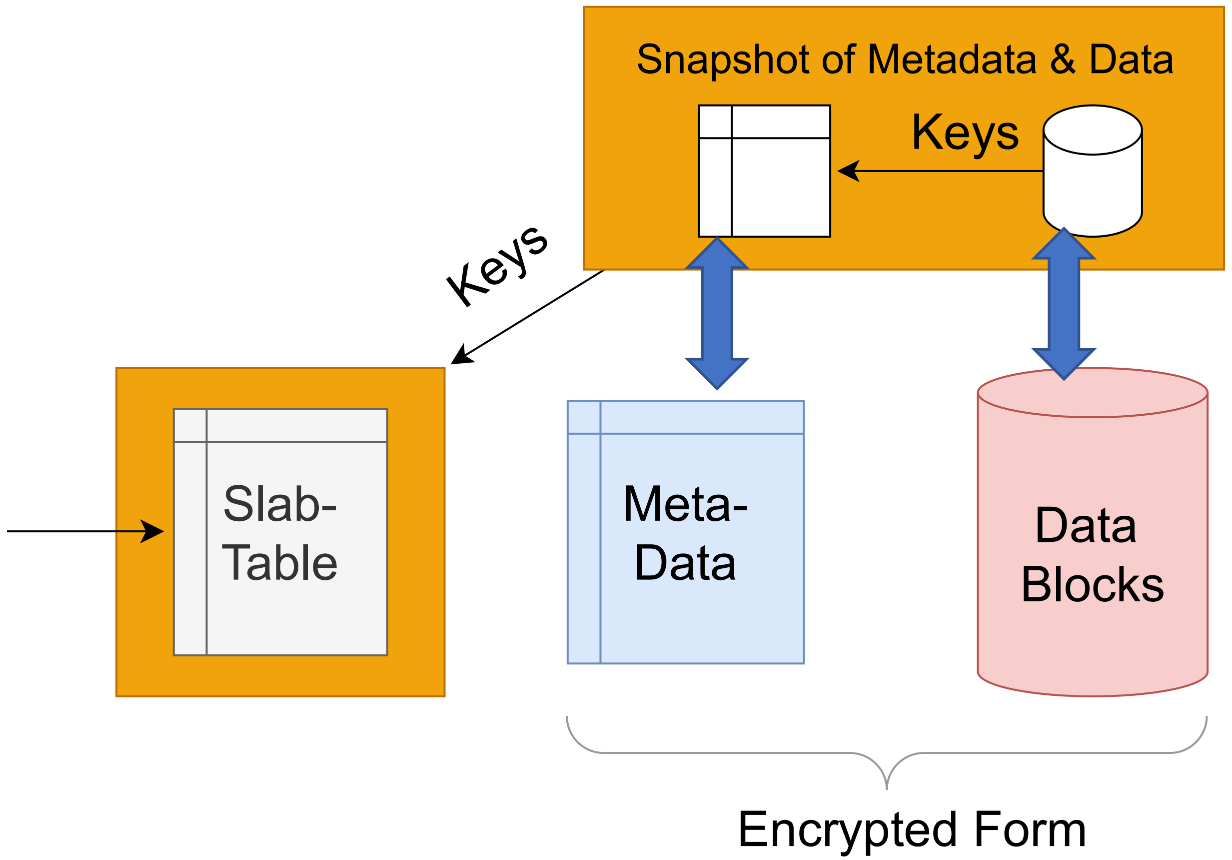
Root-of-trust

Trusted Region (Data in plaintext)



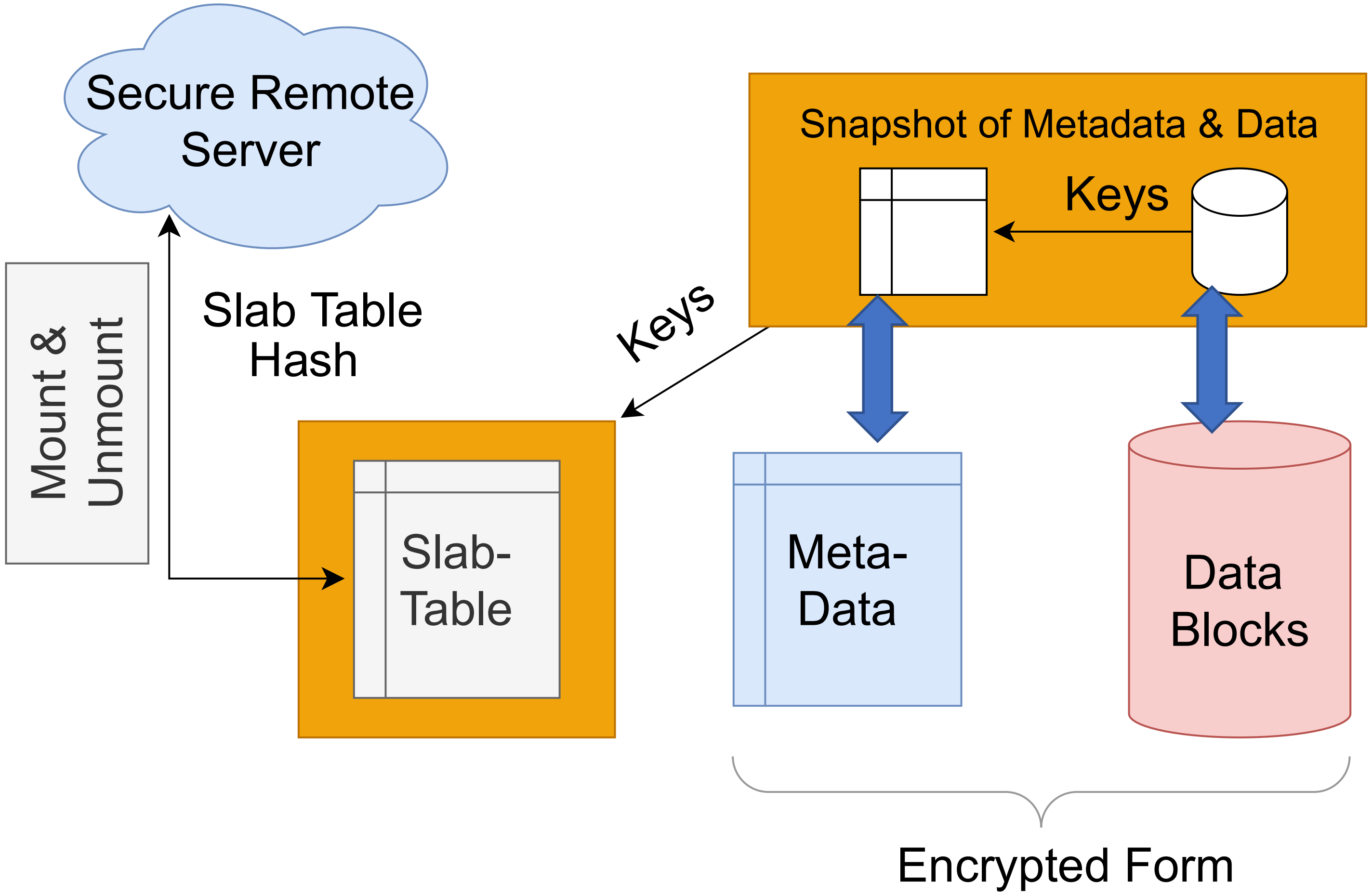
Root-of-trust

Trusted Region (Data in plaintext)



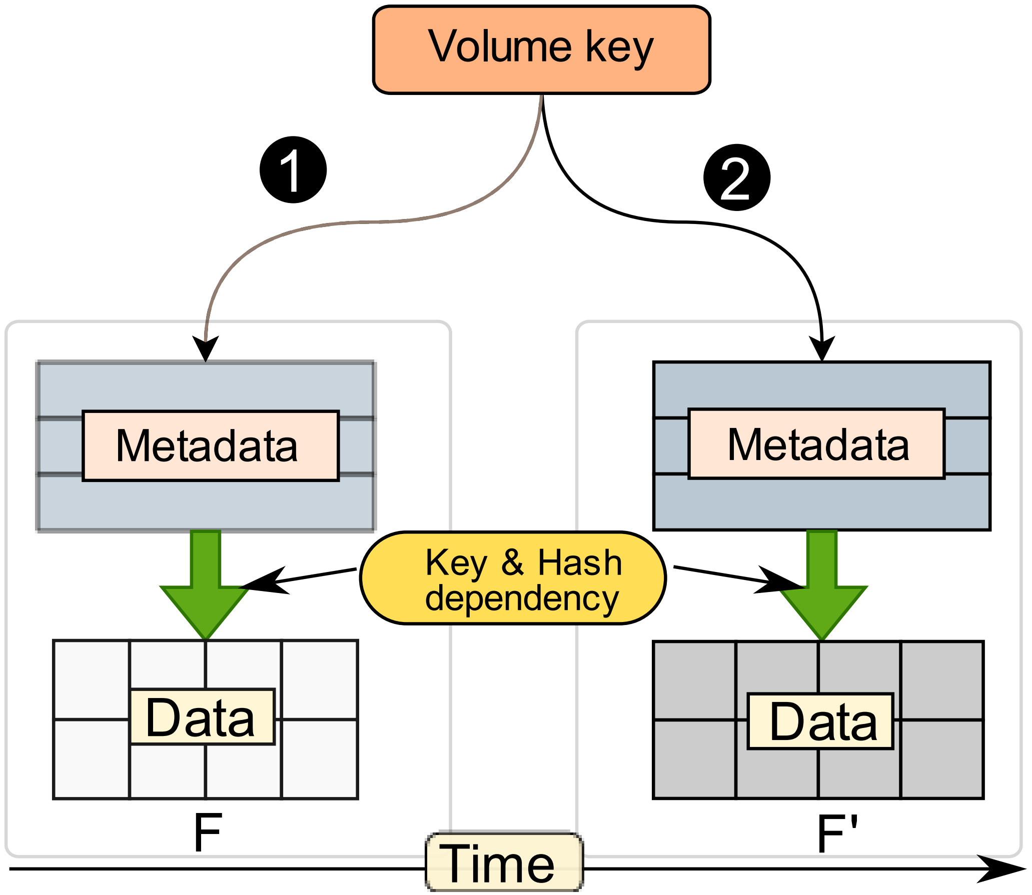
Root-of-trust

Trusted Region (Data in plaintext)

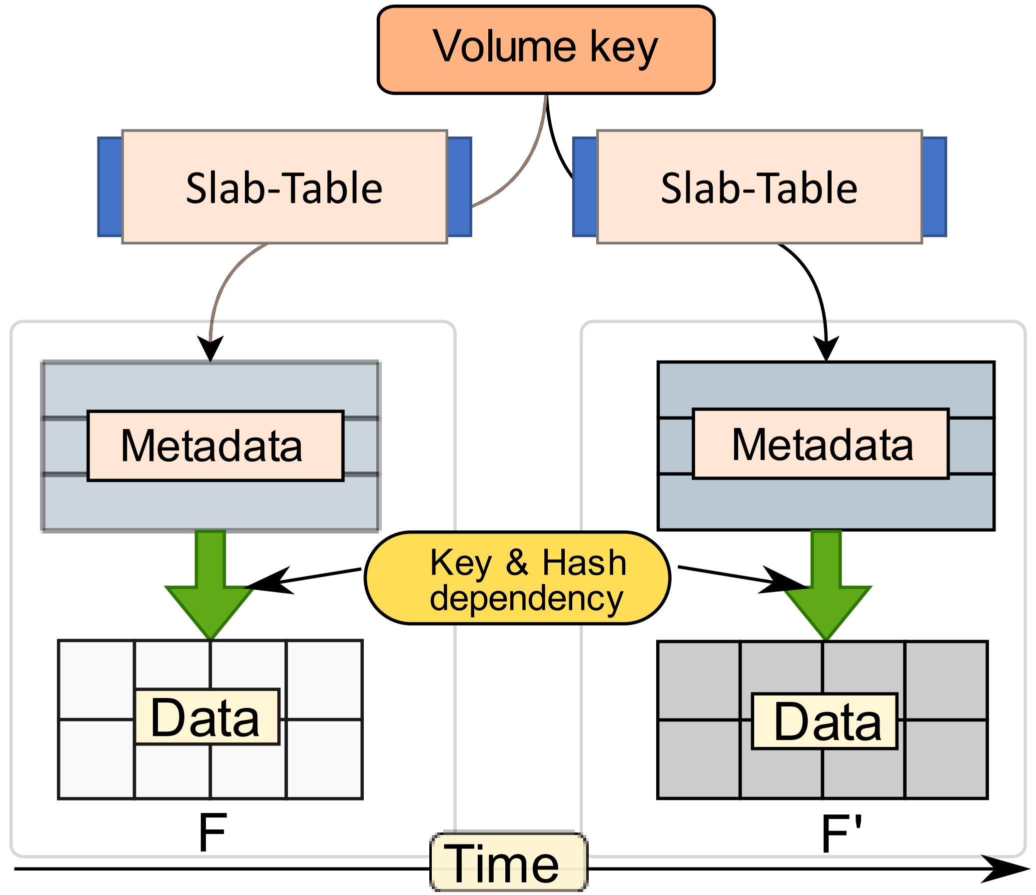


Preventing Replay Attacks

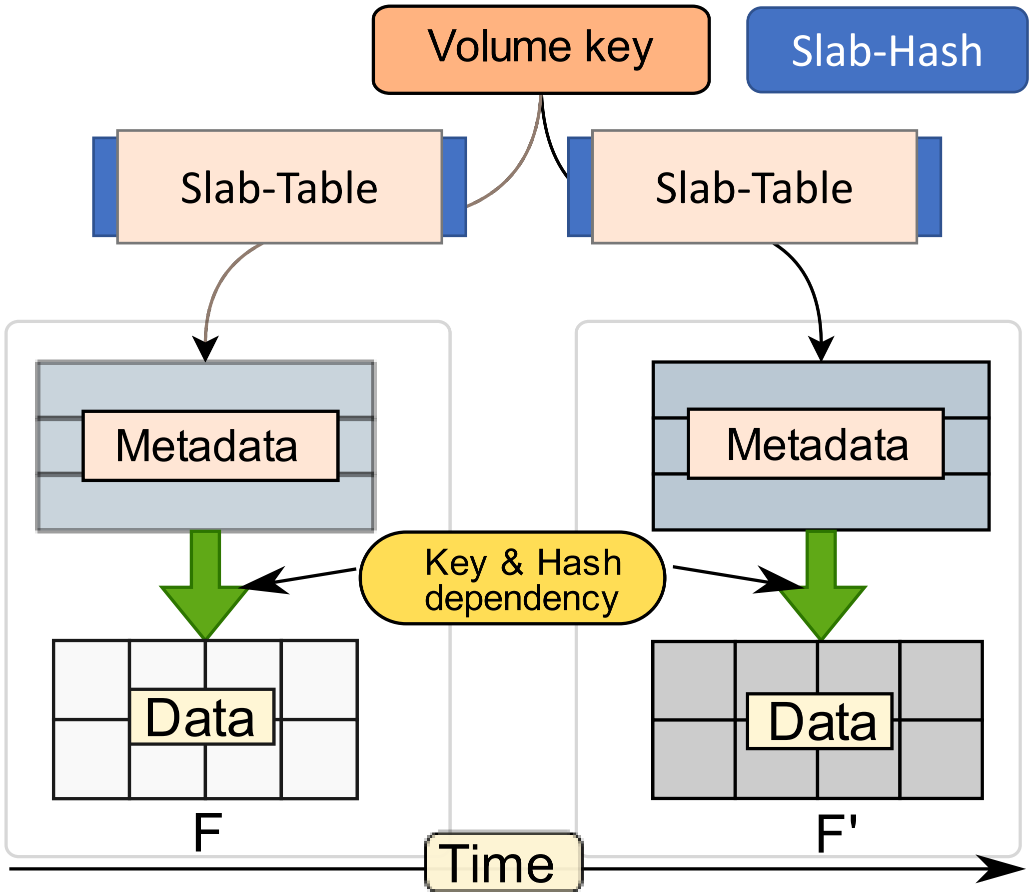
Preventing Replay Attacks



Preventing Replay Attacks

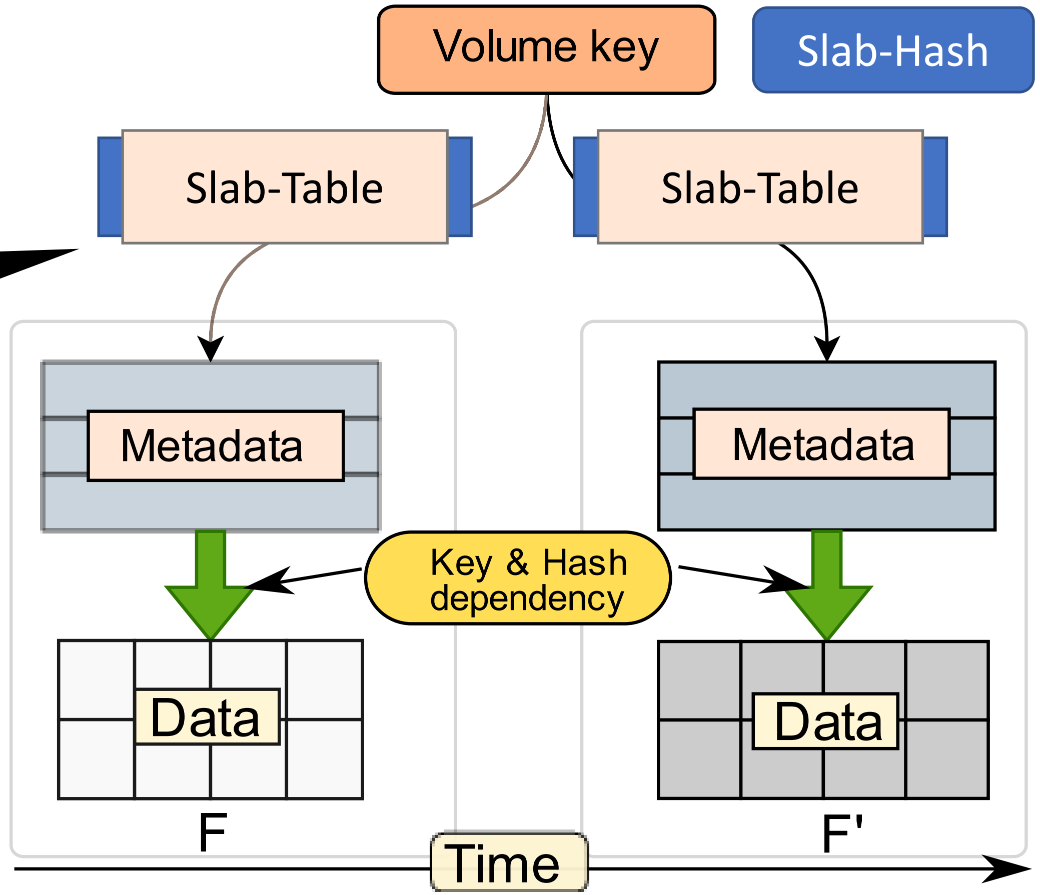


Preventing Replay Attacks



Preventing Replay Attacks

In the event of a replay attack, the initialization of the slab-table will fail.



Evaluation

SecureFS FAT and Inode mode.

Hardware Setting

Model: Intel Core i7-10700 CPU, 2.90 GHz	DRAM: 16 GB	Disk: 256 GB (SSD)
CPUs: 1 Socket, 8 Cores, 2 HT	L1: 256 KB, L2: 2 MB, L3: 16 MB	
AES hardware support: YES	SHA hardware support: **NO**	

System Settings

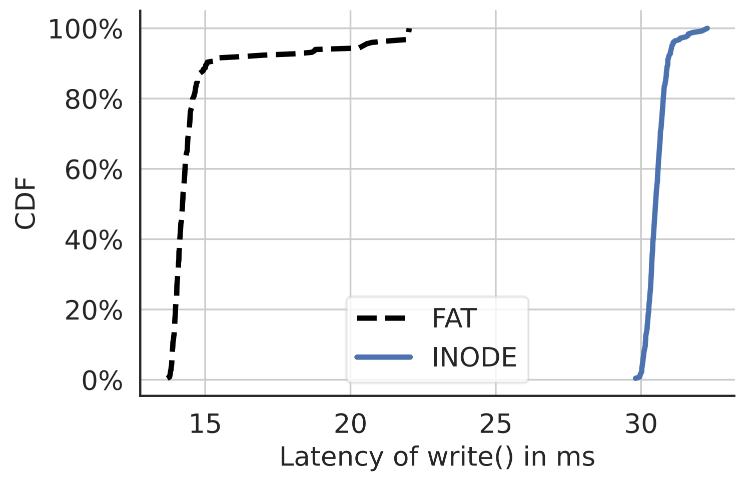
Linux kernel: 5.9	DVFS: fixed frequency (performance)	ASLR: Off
Python version: 3.6	Java version: 1.8	GCC: 9.3.0

SGX Settings

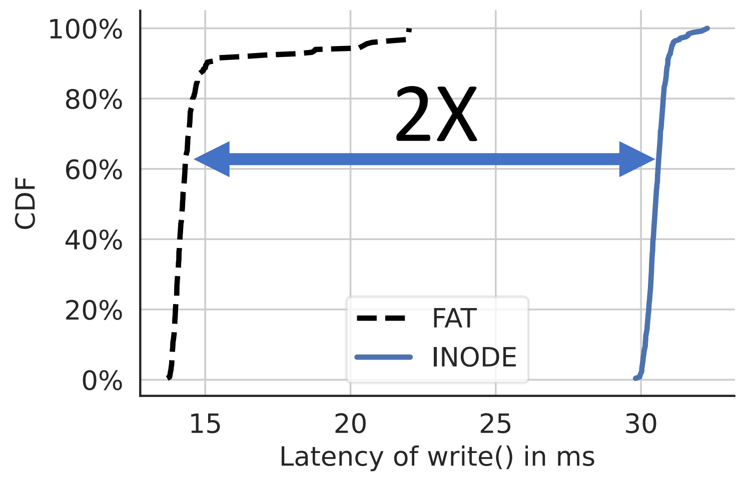
PRM: 128 MB	Driver version: 2.11	SDK version: 2.13

SecureFS: FAT vs INODE

SecureFS: FAT vs INODE



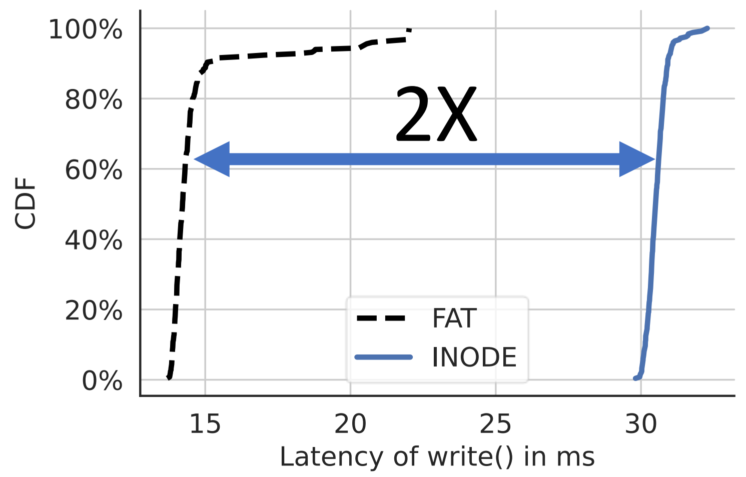
SecureFS: FAT vs INODE



SecureFS: **FAT vs INODE**

The effect of cascaded-updates can be seen.

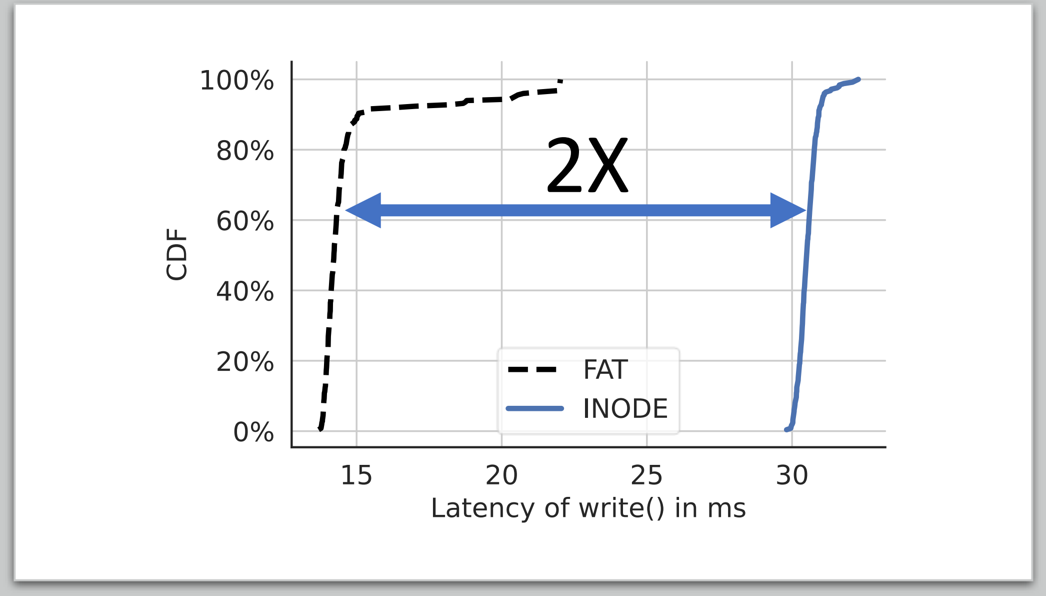
Reads are not affected.

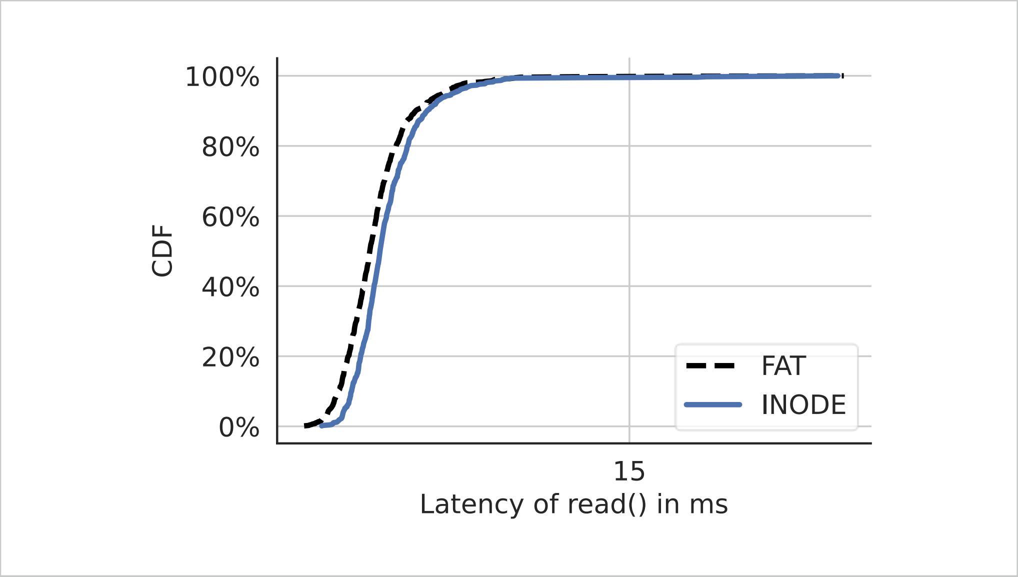


SecureFS: FAT vs INODE

The effect of cascaded-updates can be seen.

Reads are not affected.

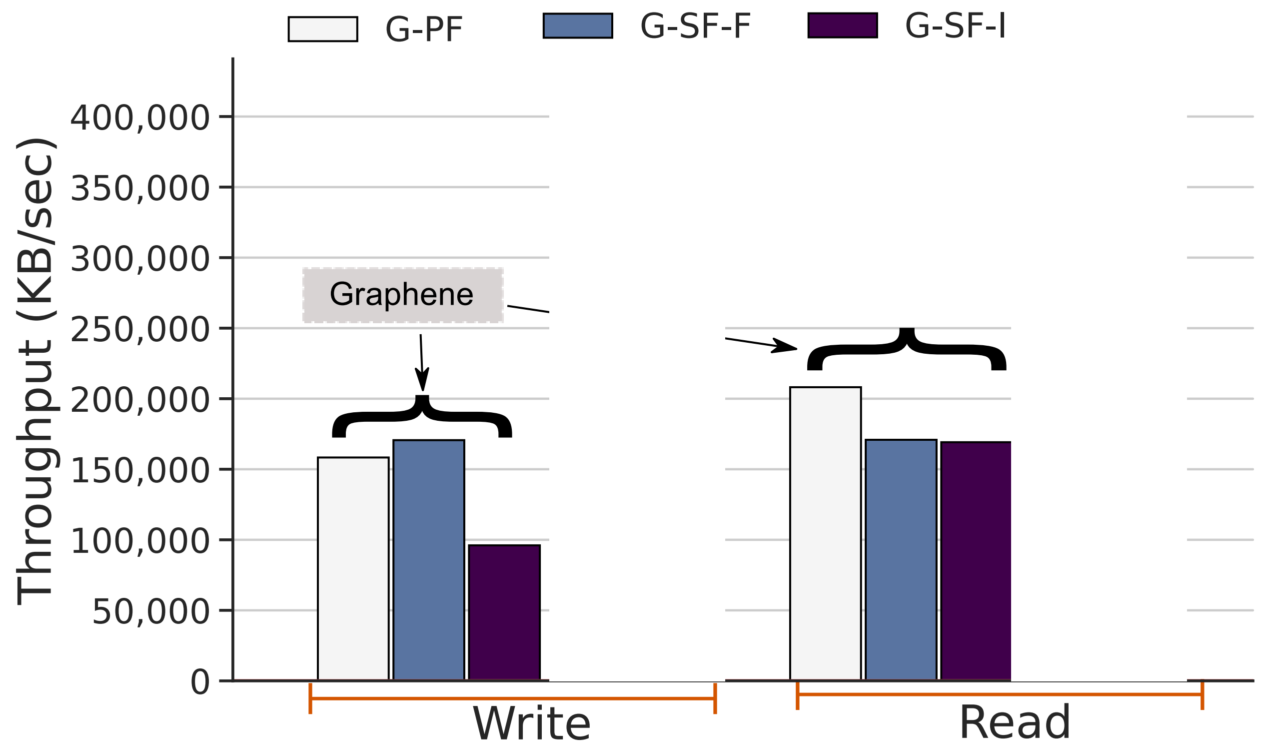




SecureFS Performance

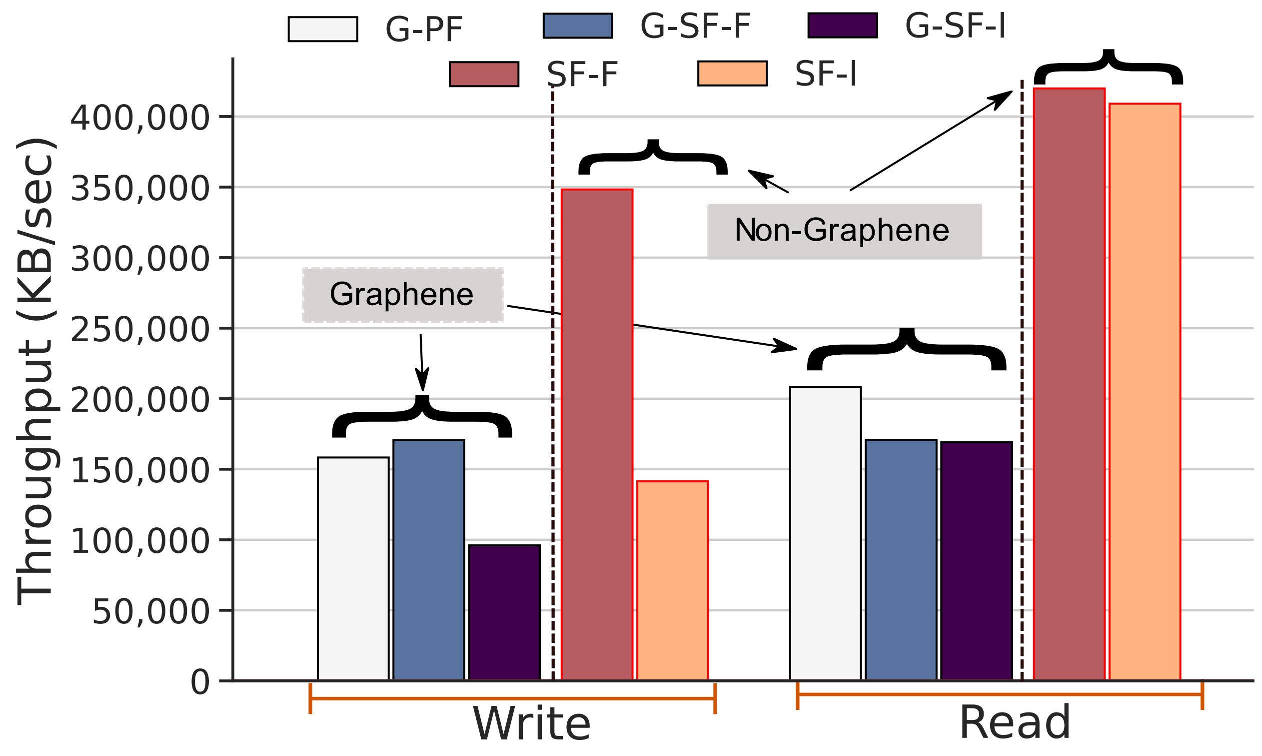
- G- Graphene Mode
    - PF: Graphene Protected Files.
    - SF-F: SecureFS FAT Mode
    - SF-I: SecureFS INODE Mode

SecureFS Performance



- G- Graphene Mode
  - PF: Graphene Protected Files.
  - SF-F: SecureFS FAT Mode
  - SF-I: SecureFS INODE Mode

SecureFS Performance



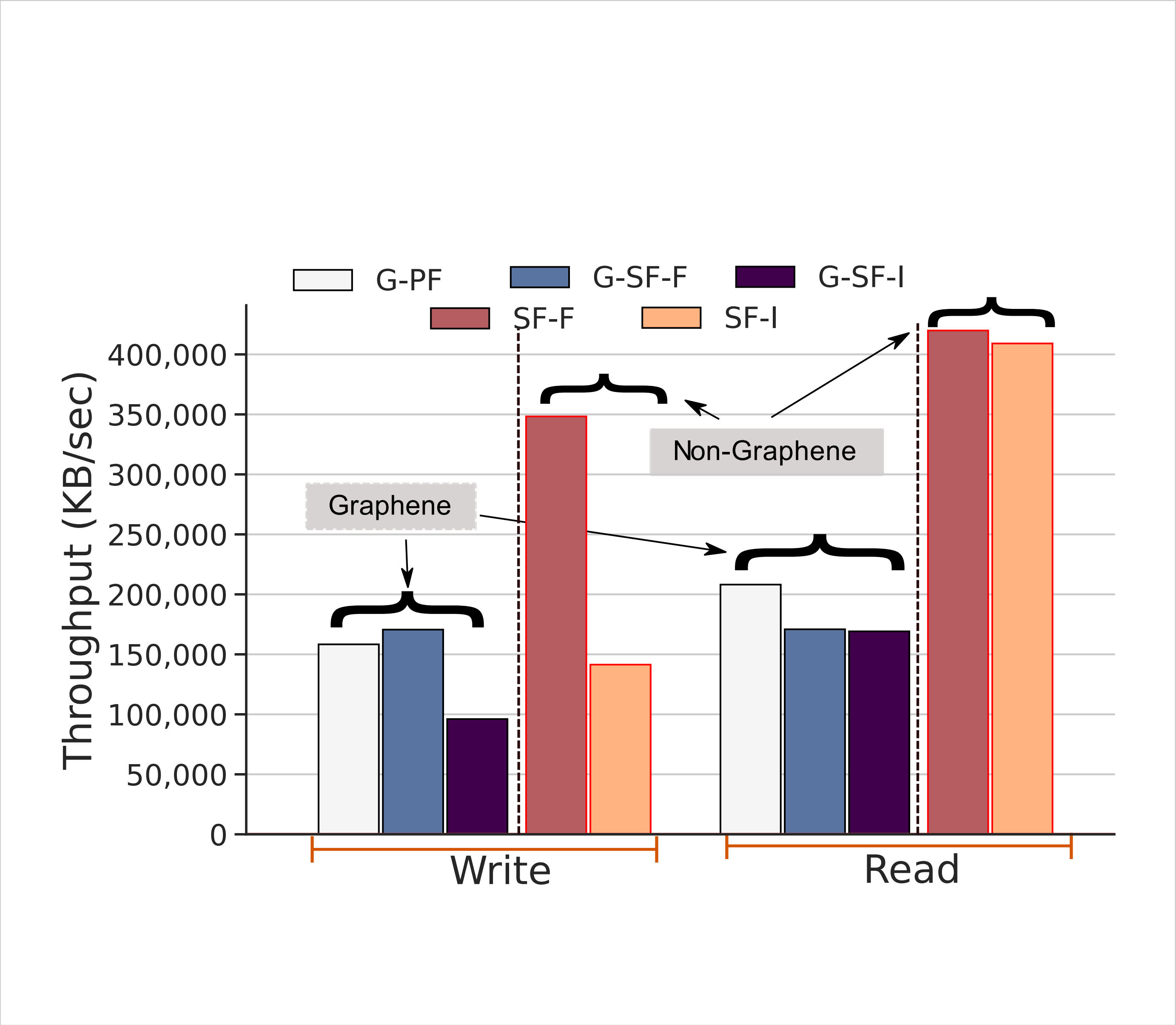
- G- Graphene Mode
  - PF: Graphene Protected Files.
  - SF-F: SecureFS FAT Mode
  - SF-I: SecureFS INODE Mode

SecureFS Performance

Similar performance if SecureFS is mapped within Graphene.

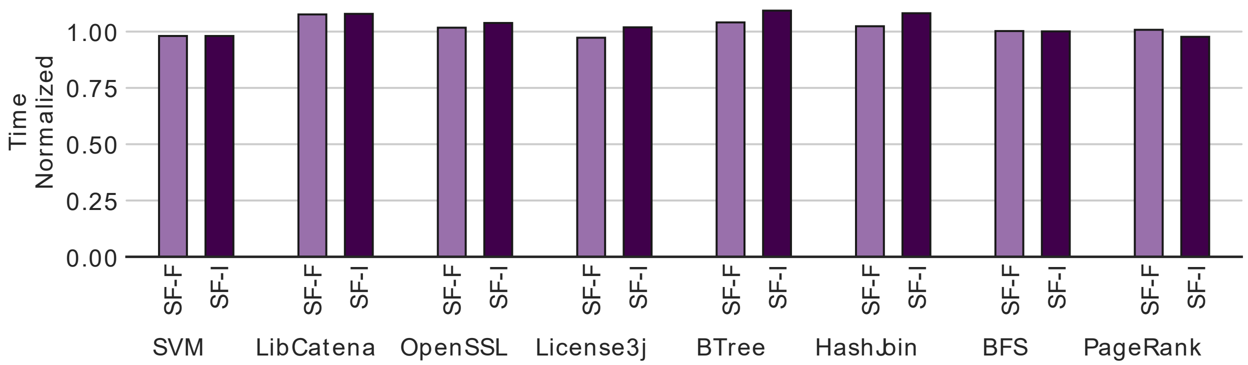
Performance improves by 120% if it is mapped outside.

- G- Graphene Mode
  - PF: Graphene Protected Files.
  - SF-F: SecureFS FAT Mode
  - SF-I: SecureFS INODE Mode



SecureFS Performance

SecureFS Performance

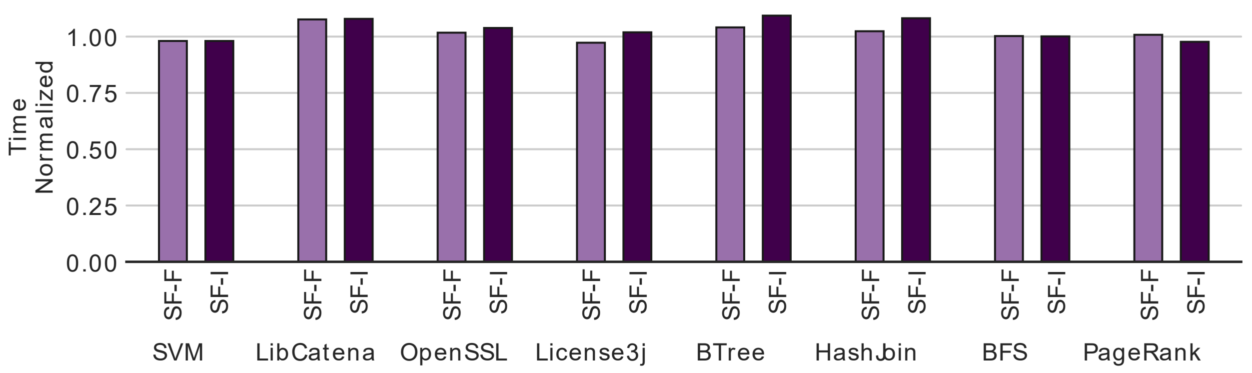


- SF-F: SecureFS FAT Mode
- SF-I: SecureFS INODE Mode

SecureFS Performance

SecureFS-FAT has a negligible slowdown of 1.8% over Nexus.

- SF-F: SecureFS FAT Mode
- SF-I: SecureFS INODE Mode



Conclusion

Conclusion

Conclusion

Secure File System Needs

- We showed that the needs of secure file systems are very different as compared to normal file systems.

Encryption is not enough

- We showed that relying on just encryption of data is not enough. We need a root-of-trust.

Replay attacks

- Using our novel file system design, we showed that we could provide additional security guarantees namely immunity from replay attacks.

Efficiency

- We provide additional security guarantees while ensuring minimal performance overhead (1.8%).

Thank You

- Contact:
  - sandeep.kumar@cse.iitd.ac.in
  - srsarangi@cse.iitd.ac.in
  - Department of Computer Science and Engineering
  - Indian Institute of Technology Delhi, India

This work has been partially supported by the Semiconductor Research Corporation, and the Ministry of Science and Technology (Govt. of India) via the grant DST/INT/JST/P-30/2016.