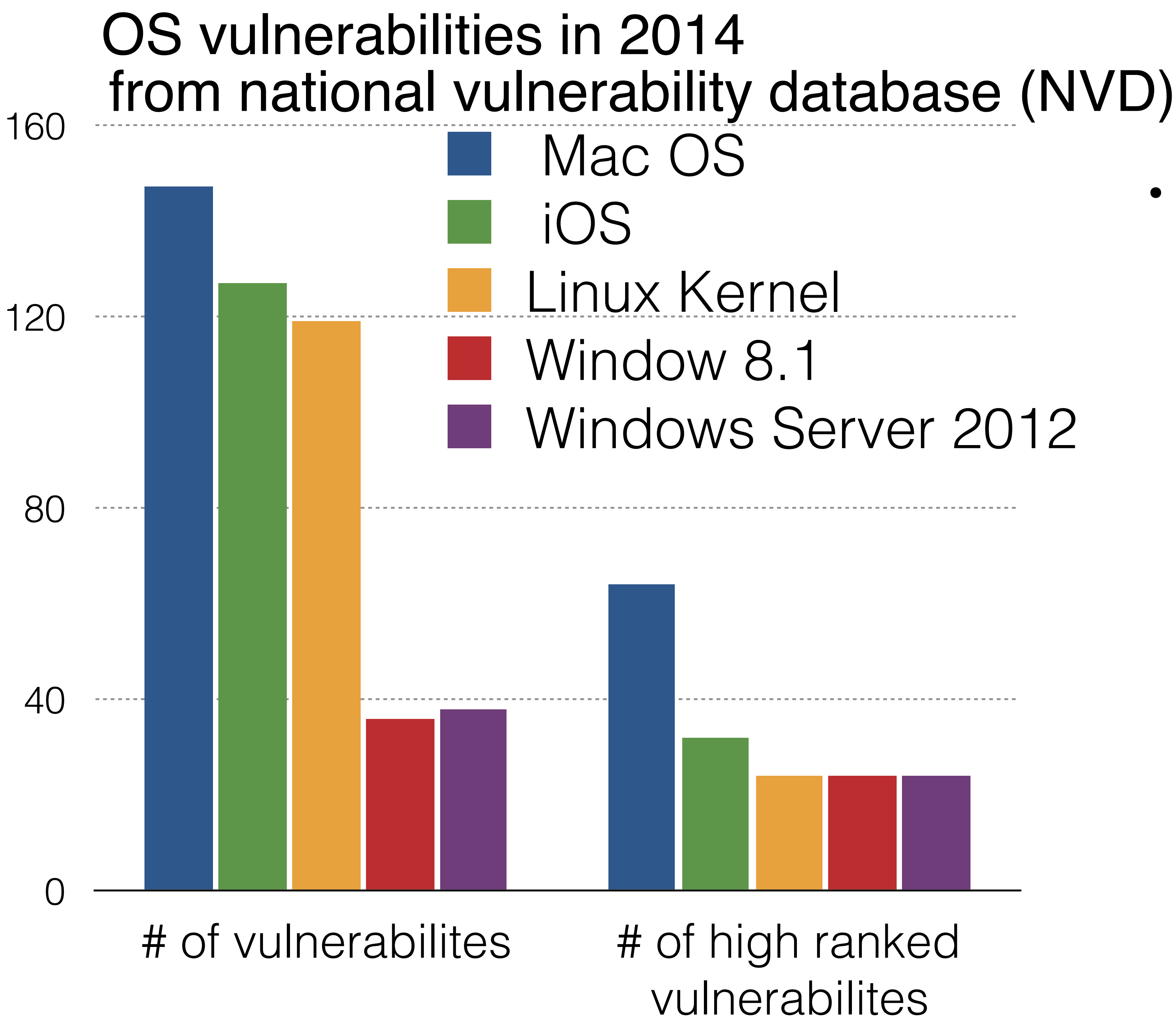
Sego: Pervasive Trusted Metadata for Efficiently Verified Untrusted System Services

Youngjin Kwon, Alan Dunn, Michael Lee,
Owen Hofmann, Yuanzhong Xu, Emmett Witchel

THE UNIVERSITY OF TEXAS AT AUSTIN

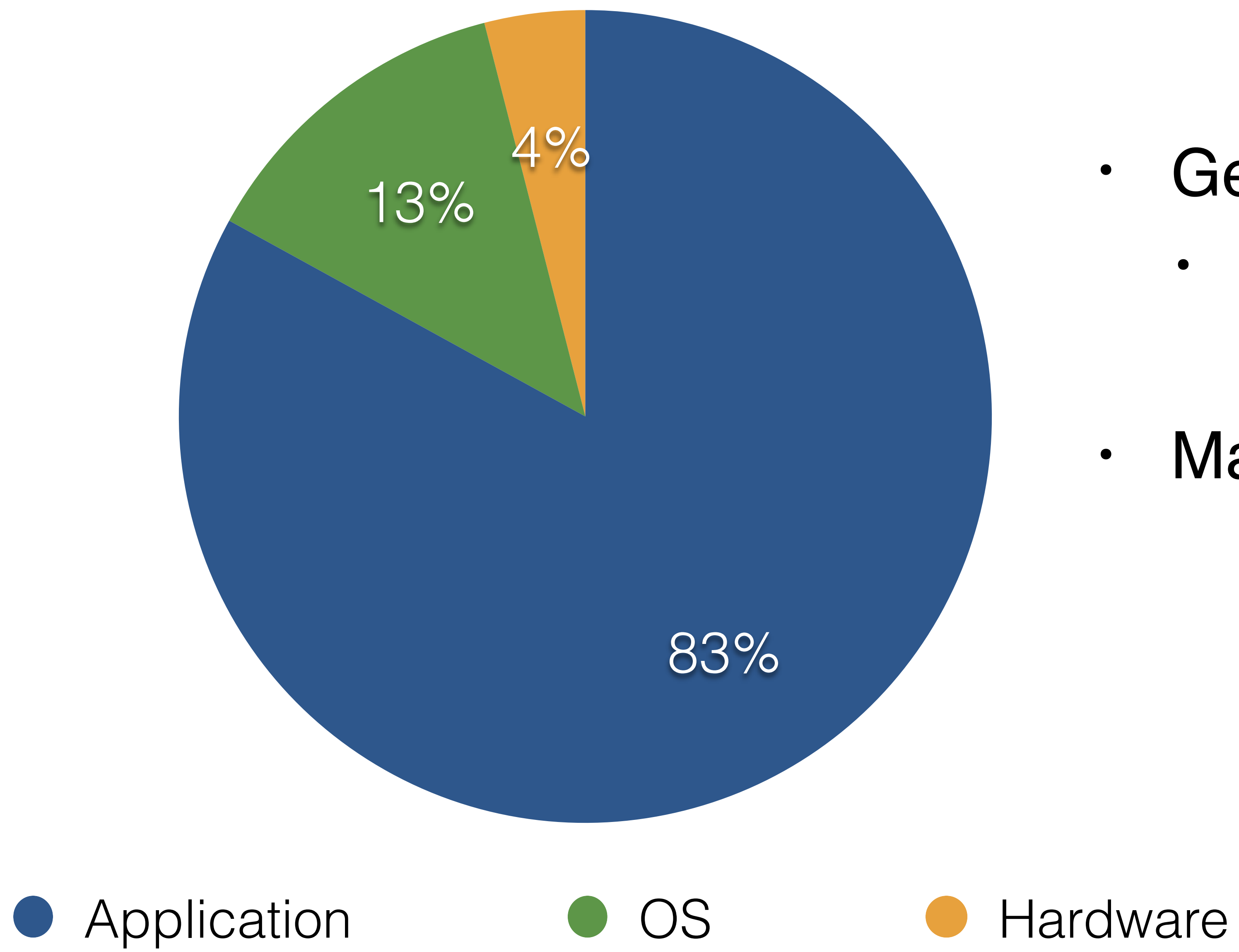
Securing OS is difficult



- Large attack surfaces
  - System calls
  - Ioctl interface
  - 3rd party device driver

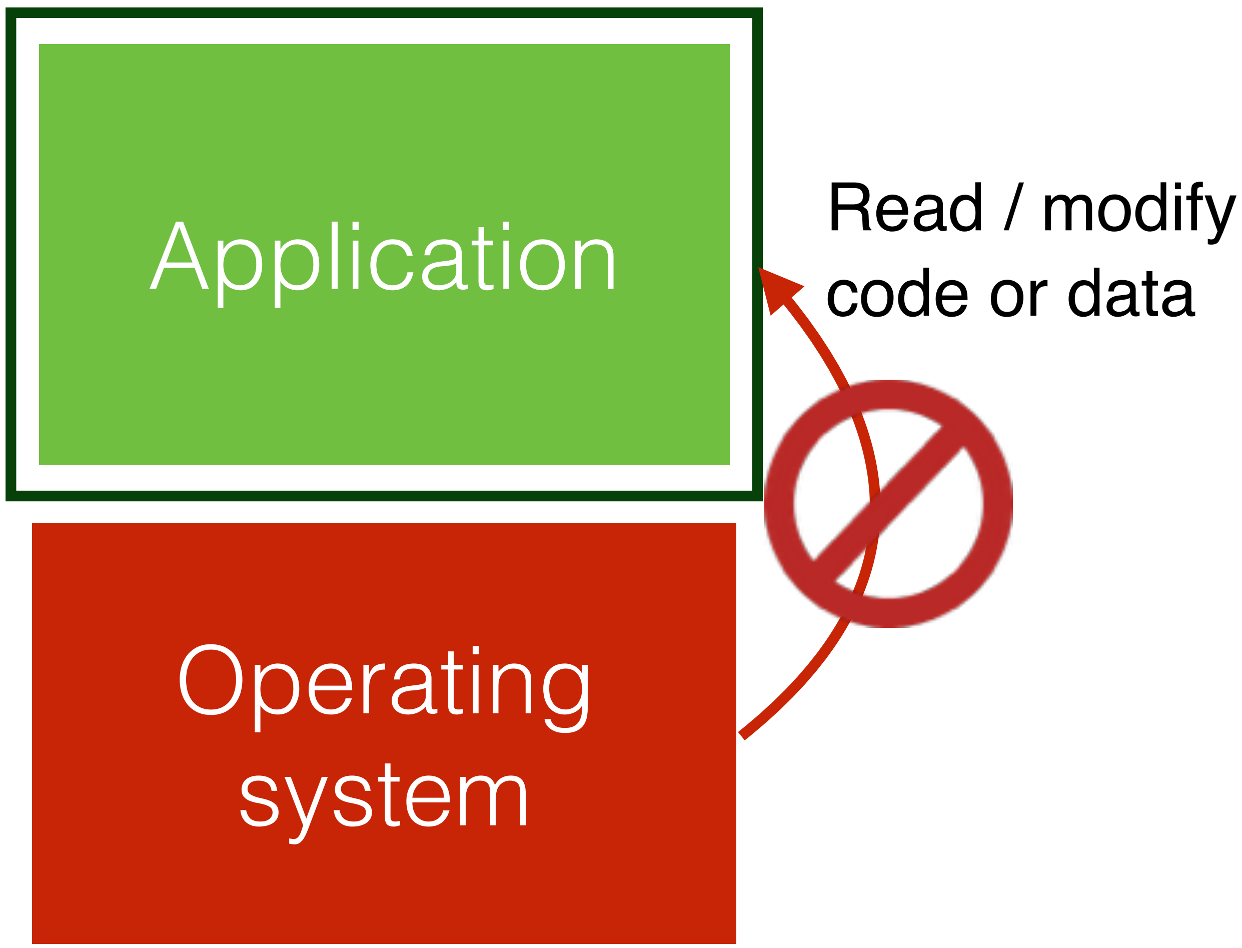
Securing OS is not enough

Vulnerability distribution in 2014 from NVD



- Getting root leads to control OS
  - Privilege escalation vulnerability
- Many APPs run with root permission

Protecting application from malicious OS



- With trusted hypervisor
  - Overshadow (ASPLOS 2008)
  - TrustVisor (IEEE S&P 2010)
  - InkTag (ASPLOS 2013)
  - **Sego (ASPLOS 2016)**
- With compiler instrumentation
  - VirtualGhost (ASPLOS 2014)
- With hardware (SGX) support
  - Haven (OSDI 2014)

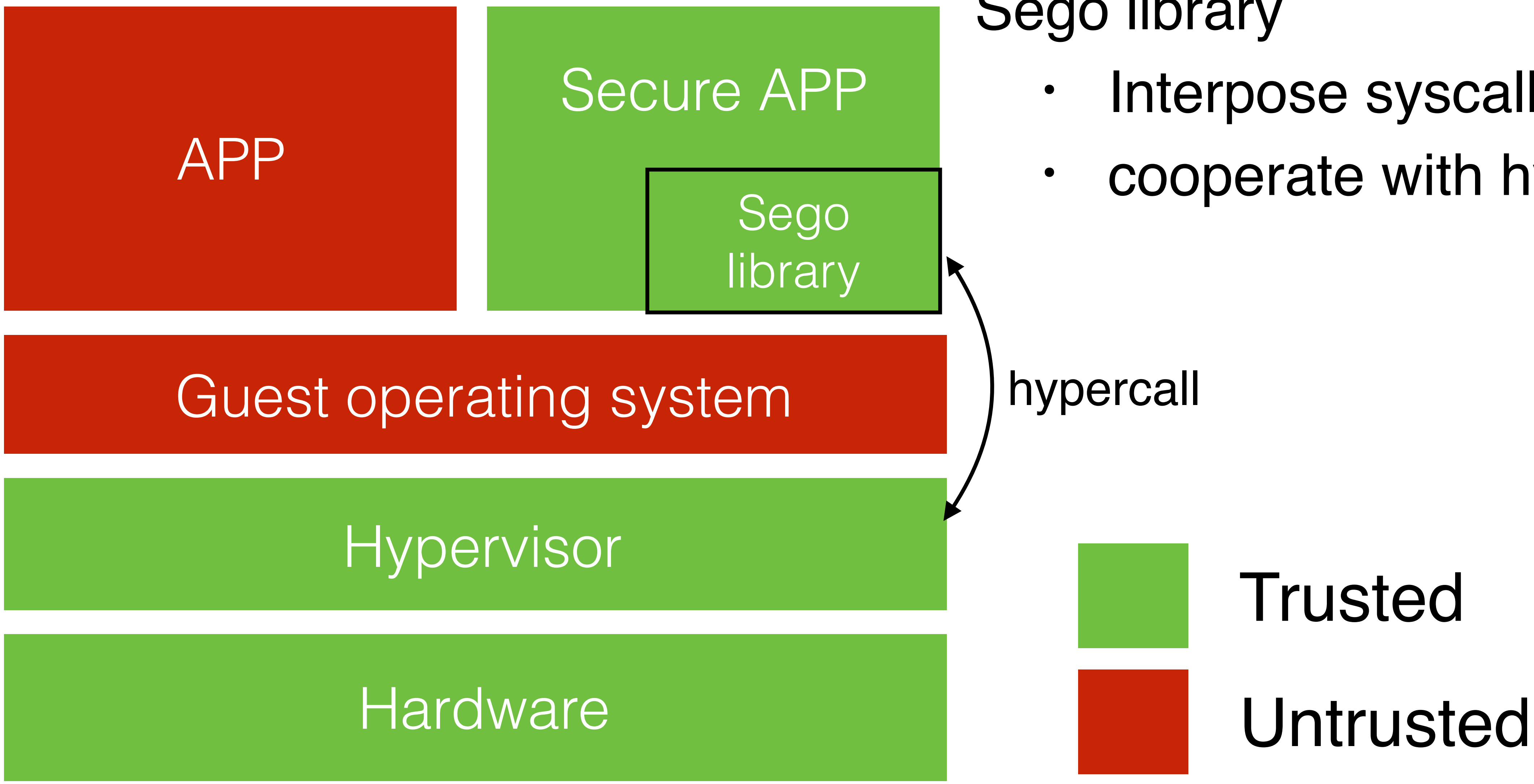
Outline

- Previous system
- Sego eliminates encryption and hashing
- Sego provides crash consistency and recovery
- Conclusion

How do previous systems work?

Trust model

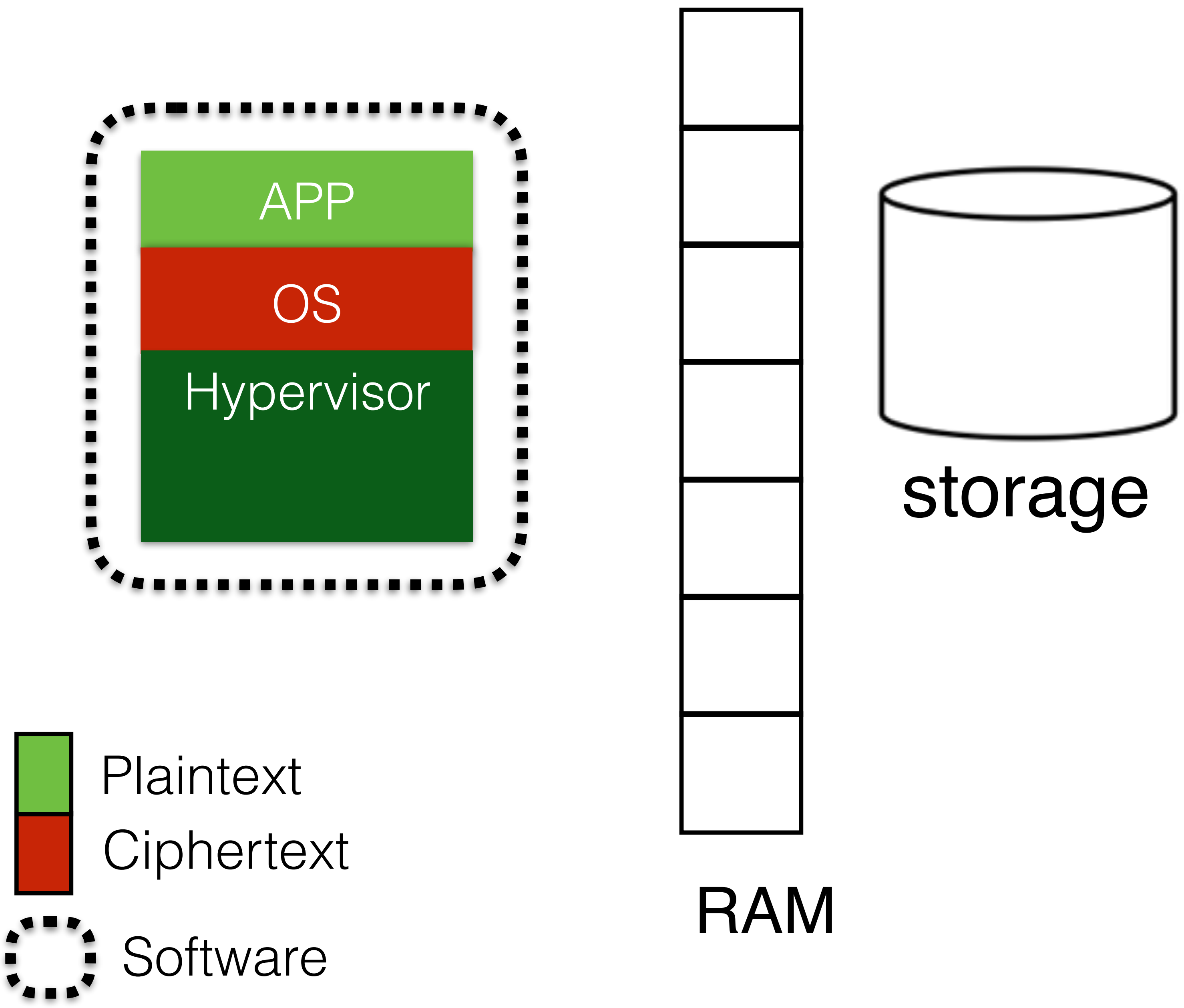
System overview



Sego library

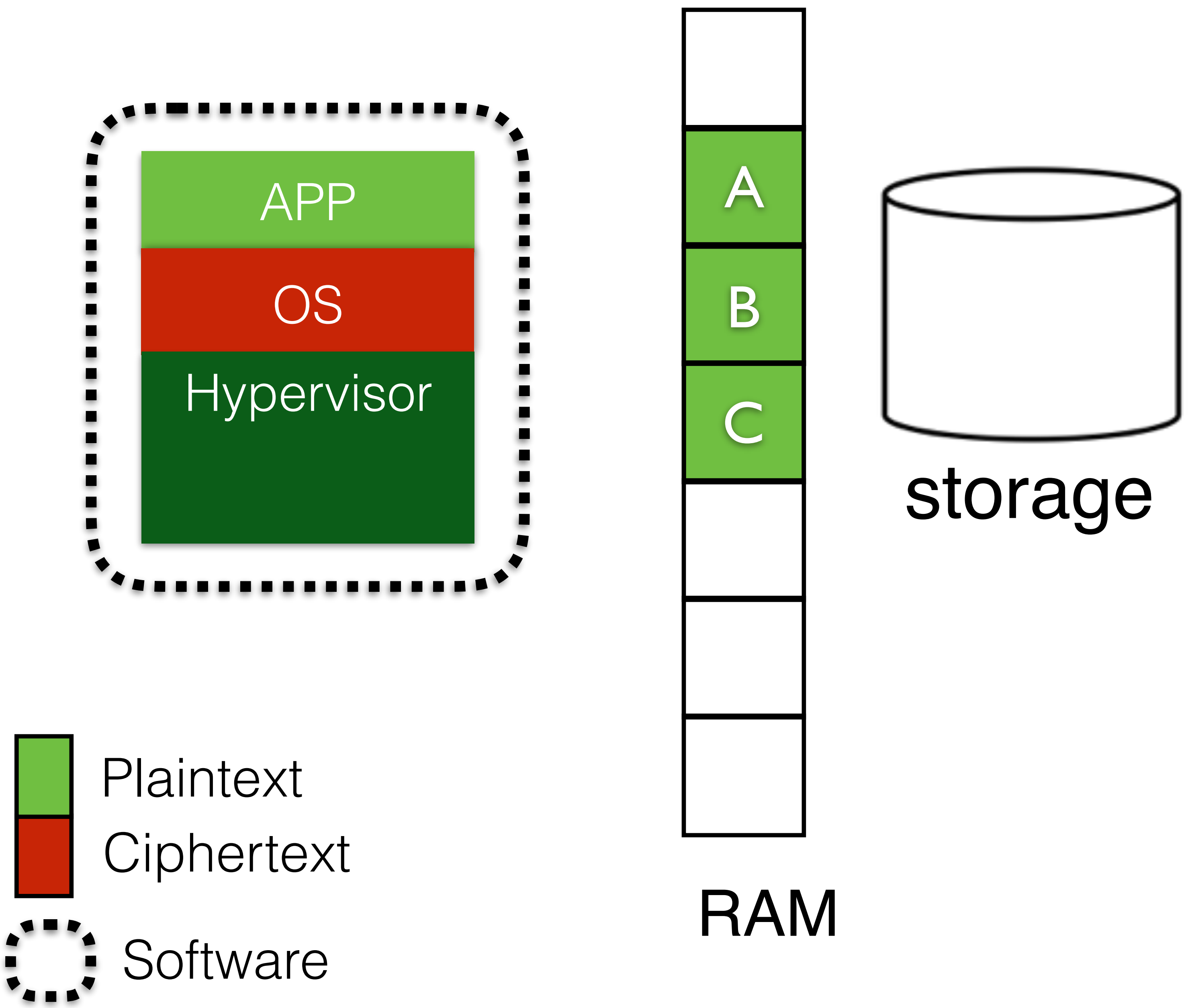
- Interpose syscall
- cooperate with hypervisor

Hypervisor encrypts memory for secrecy

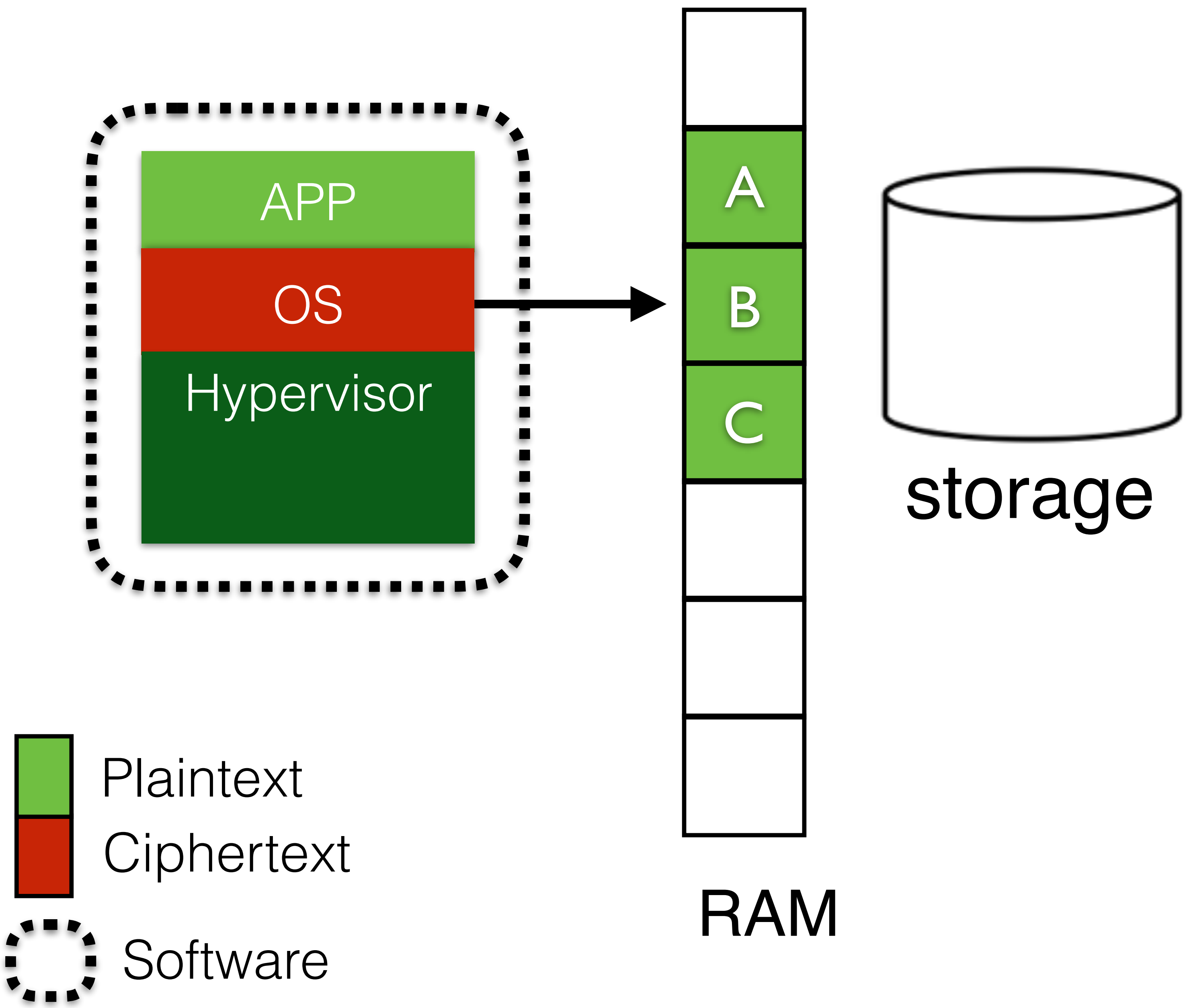


Hypervisor encrypts memory for secrecy

1. APP reads/writes memory page

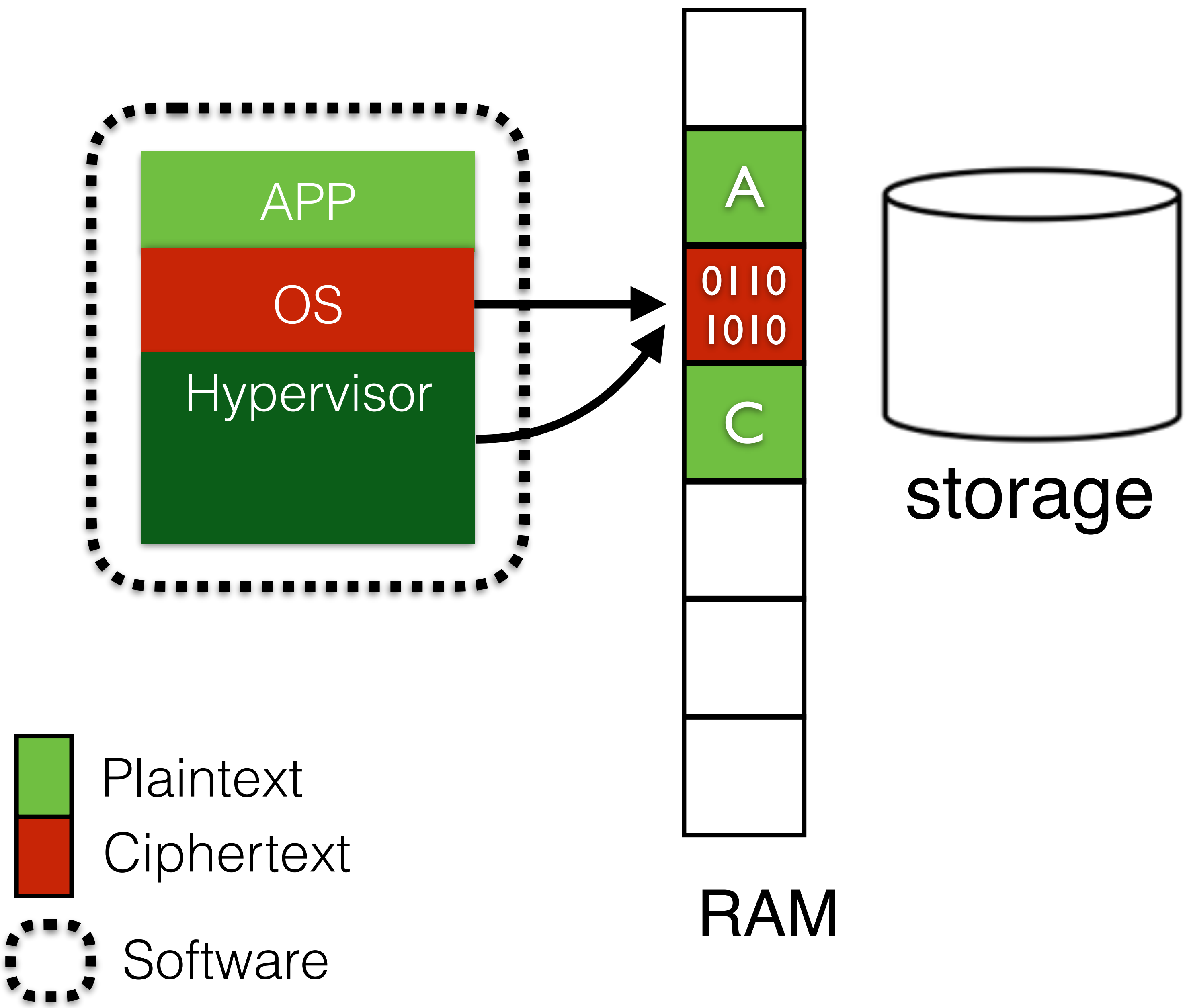


Hypervisor encrypts memory for secrecy



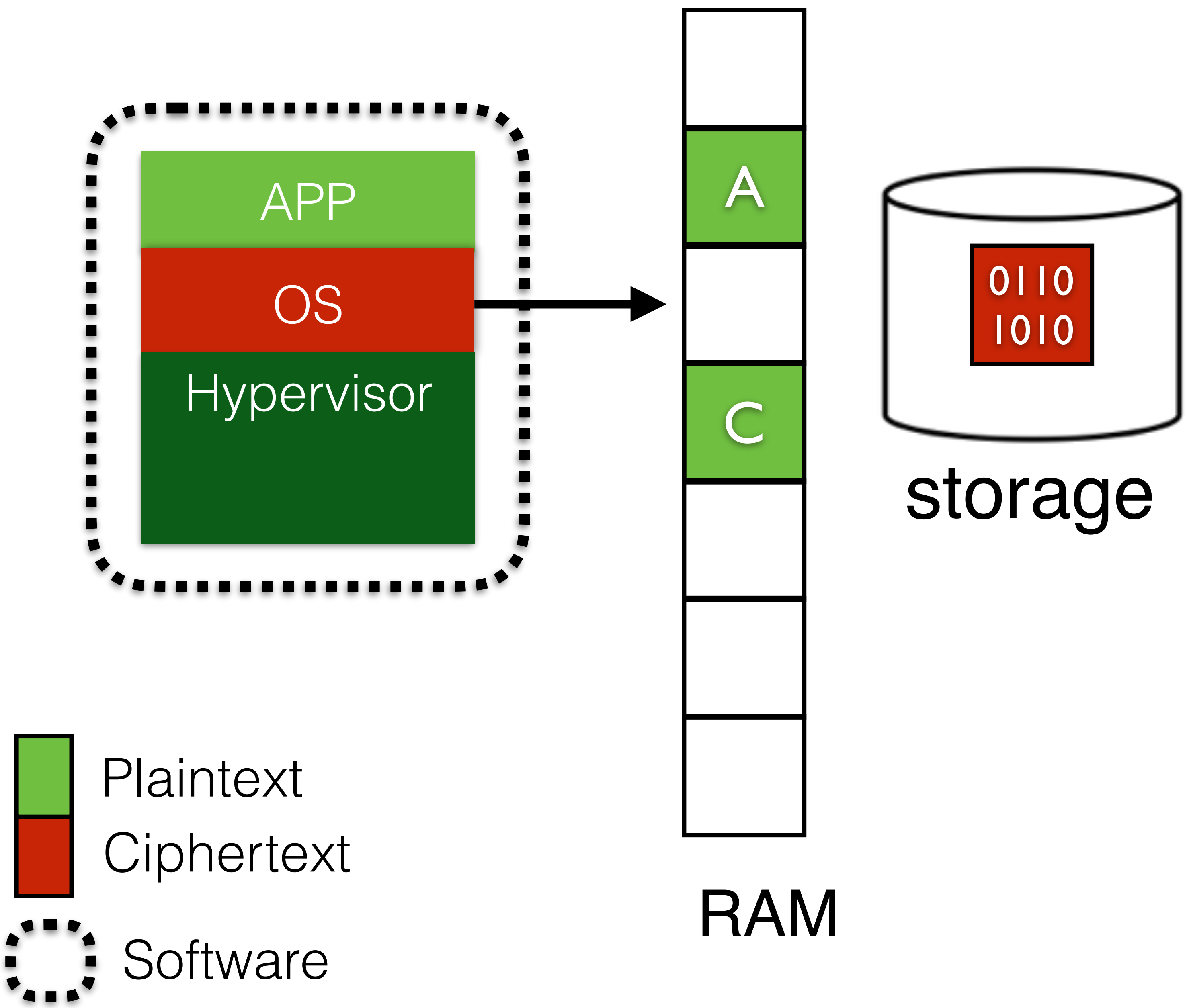
1. APP reads/writes memory page
2. OS wants to swap page

Hypervisor encrypts memory for secrecy



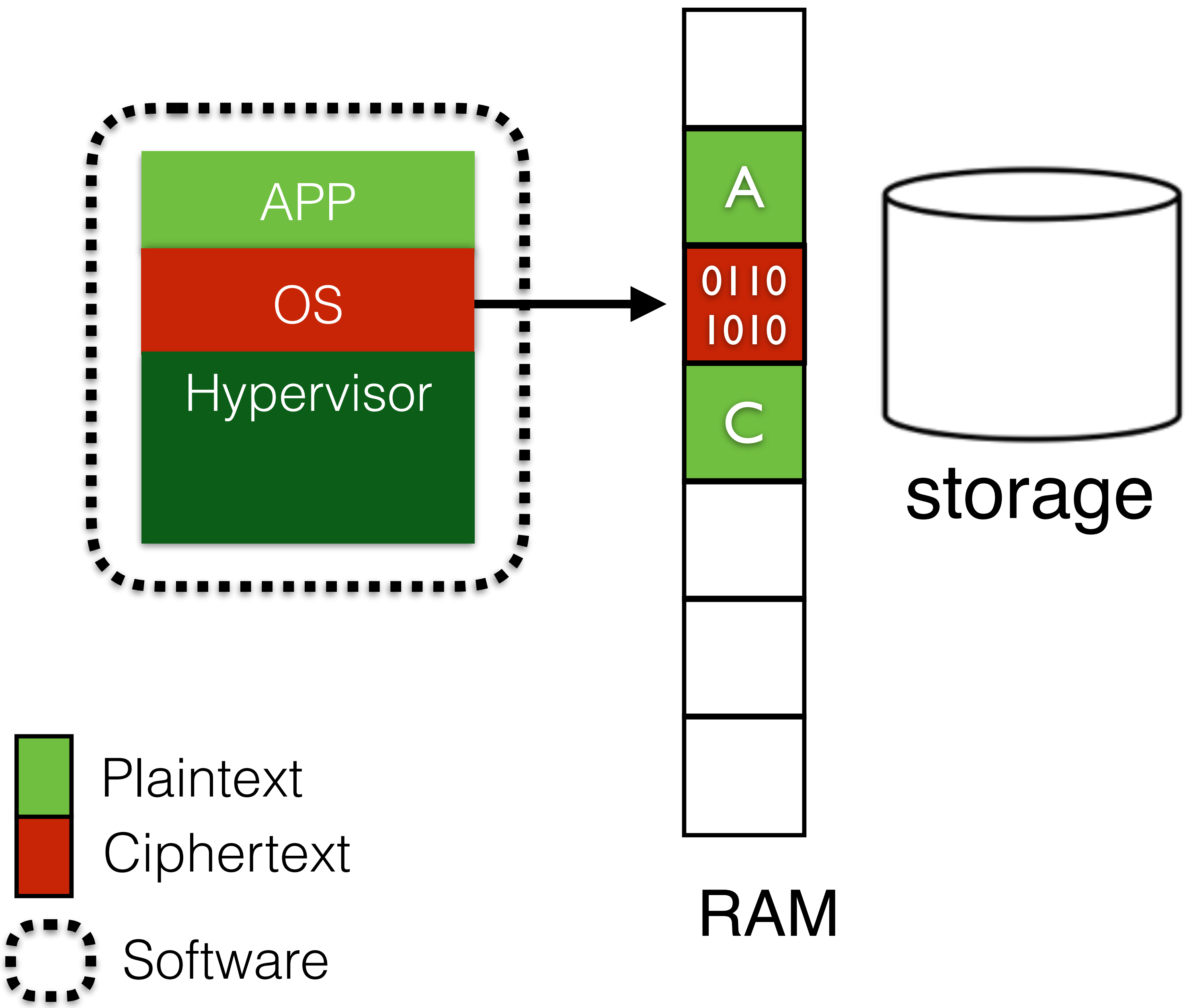
1. APP reads/writes memory page
2. OS wants to swap page
3. Hypervisor blocks OS
   a) Encrypts page

Hypervisor encrypts memory for secrecy



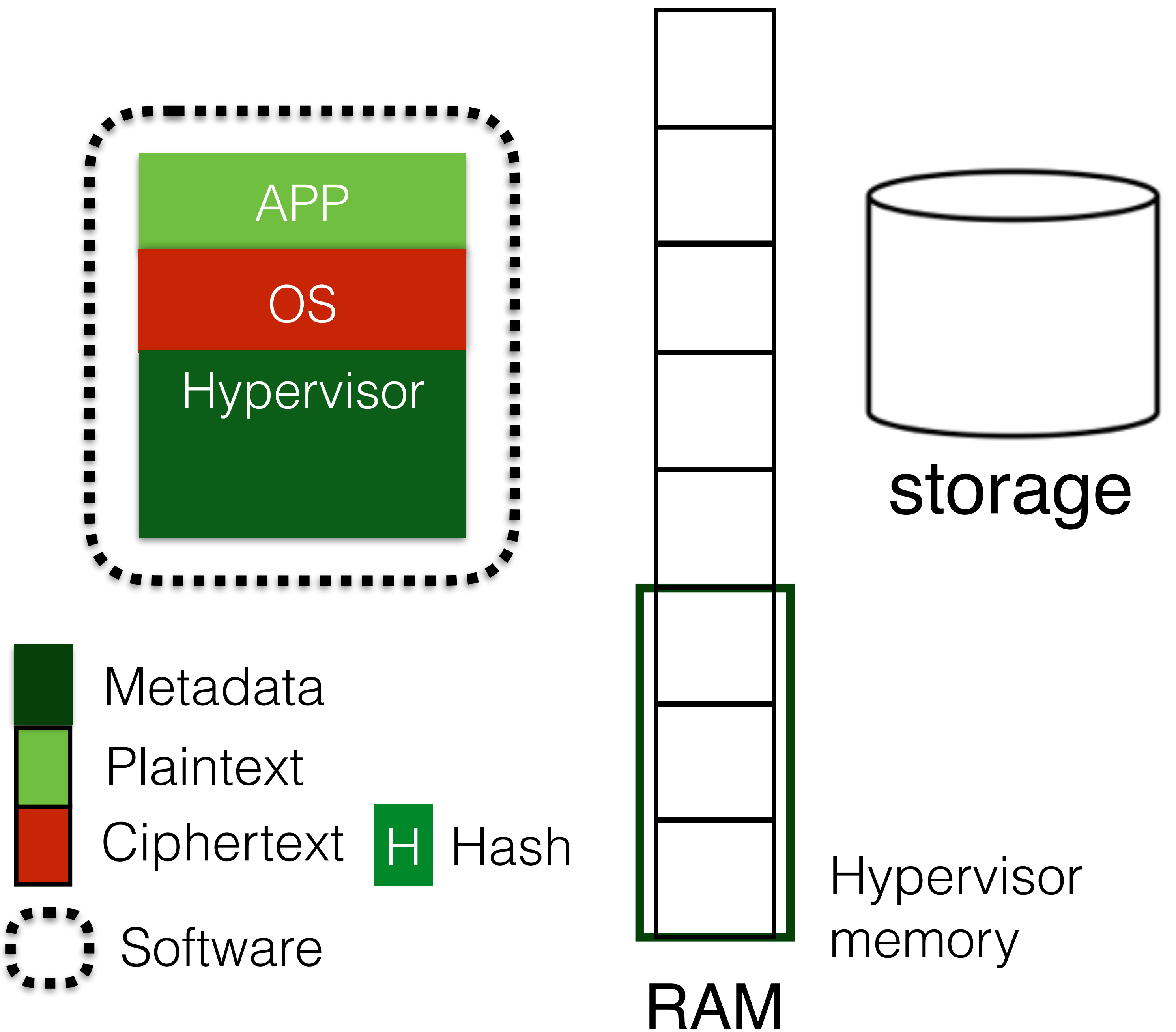
1. APP reads/writes memory page
2. OS wants to swap page
3. Hypervisor blocks OS
   a) Encrypts page
4. OS swaps encrypted page

Hypervisor encrypts memory for secrecy

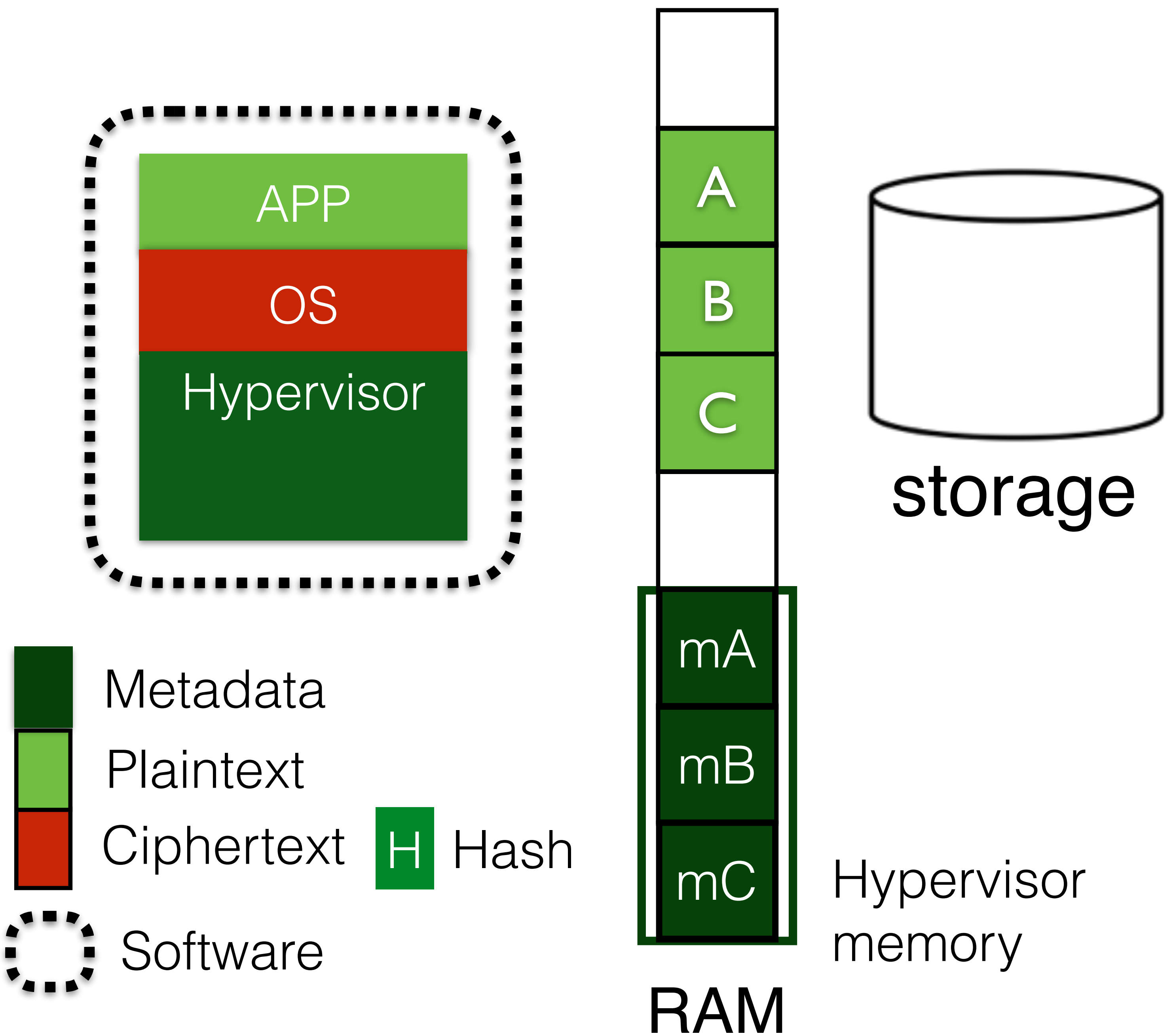


1. APP reads/writes memory page
2. OS wants to swap page
3. Hypervisor blocks OS
   a) Encrypts page
4. OS swaps encrypted page

Hypervisor hashes memory for integrity

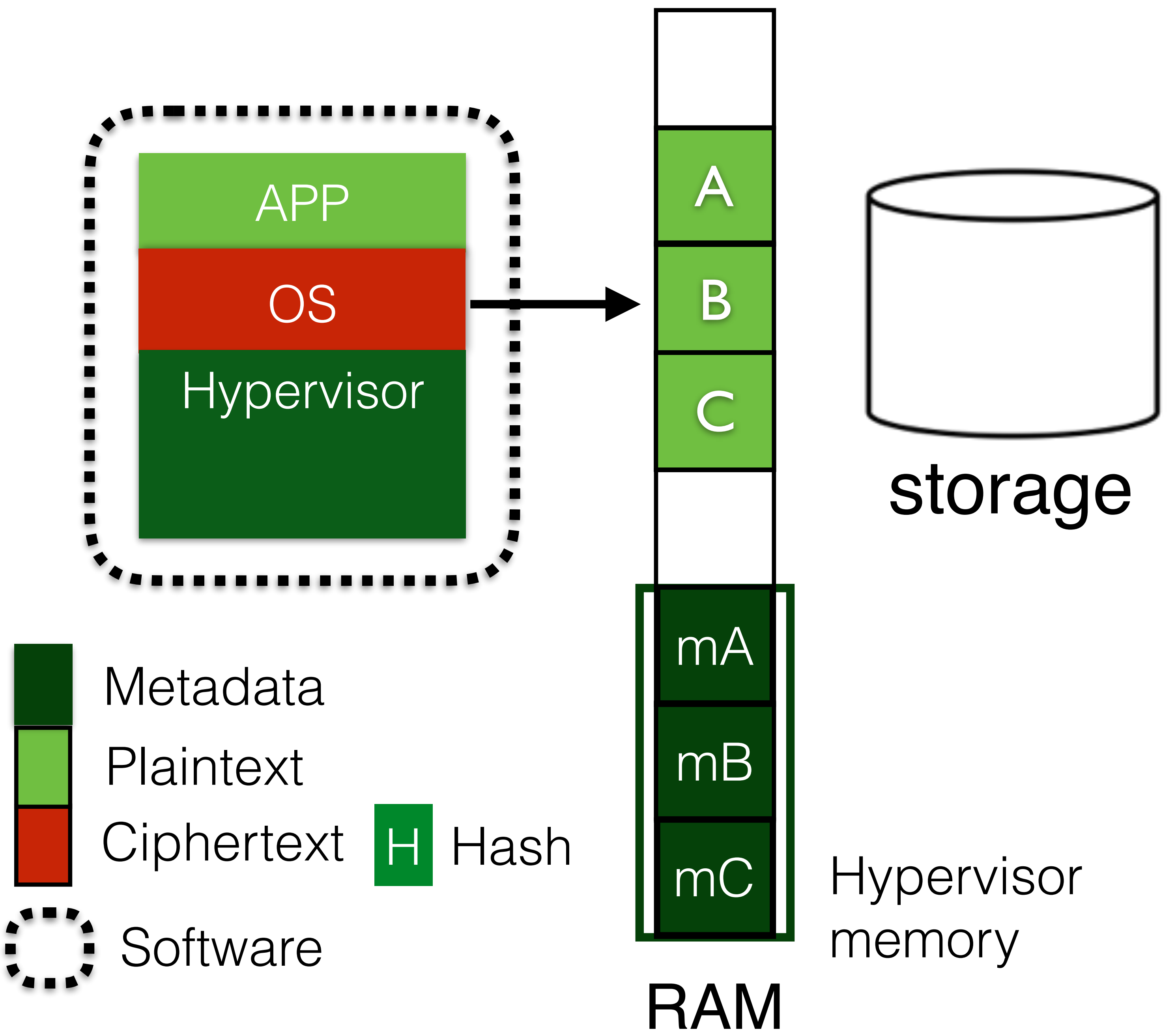


Hypervisor hashes memory for integrity



1. APP reads/writes memory page
   a) HYP maintains metadata

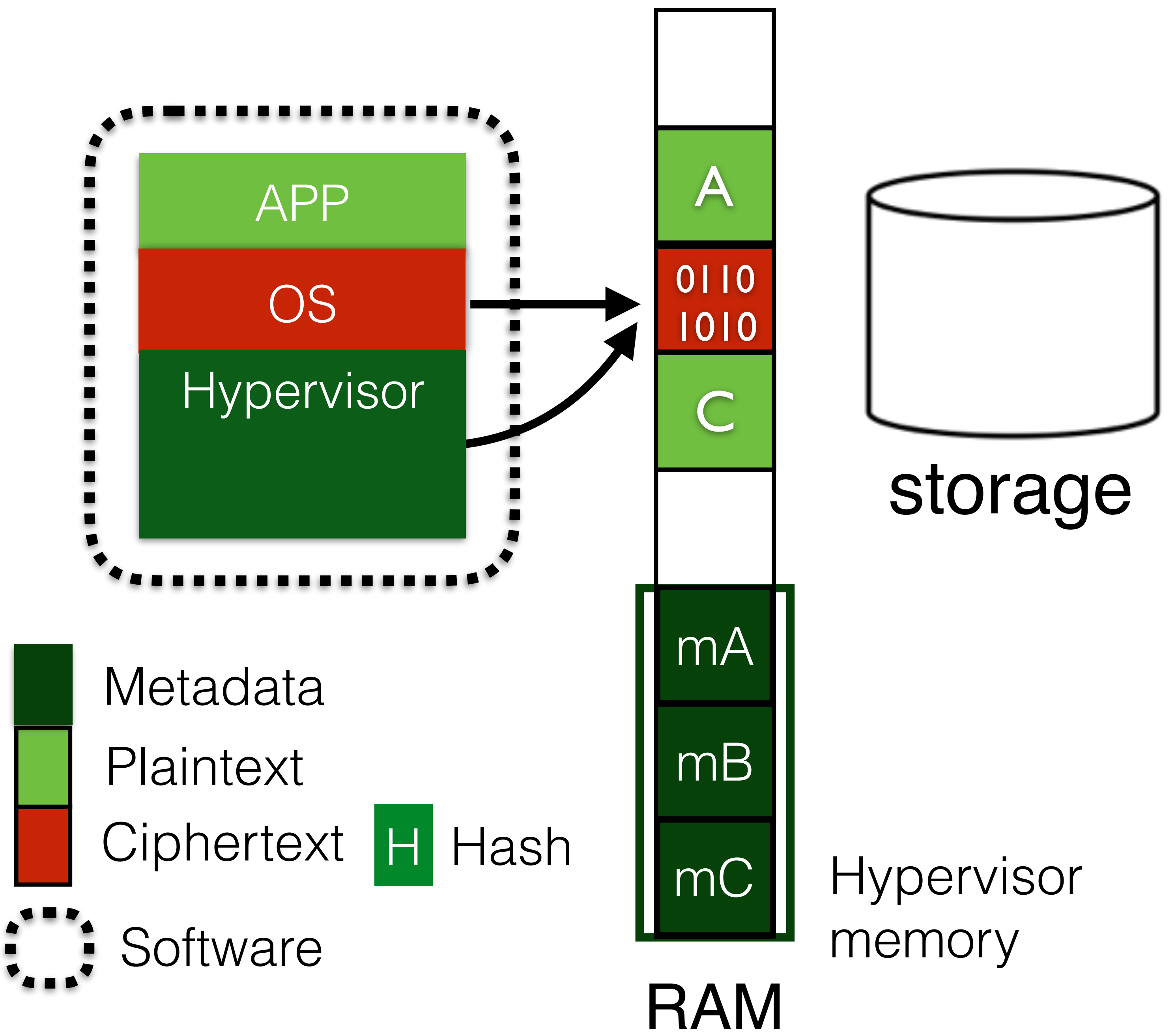
Hypervisor hashes memory for integrity



1. APP reads/writes memory page
   a) HYP maintains metadata

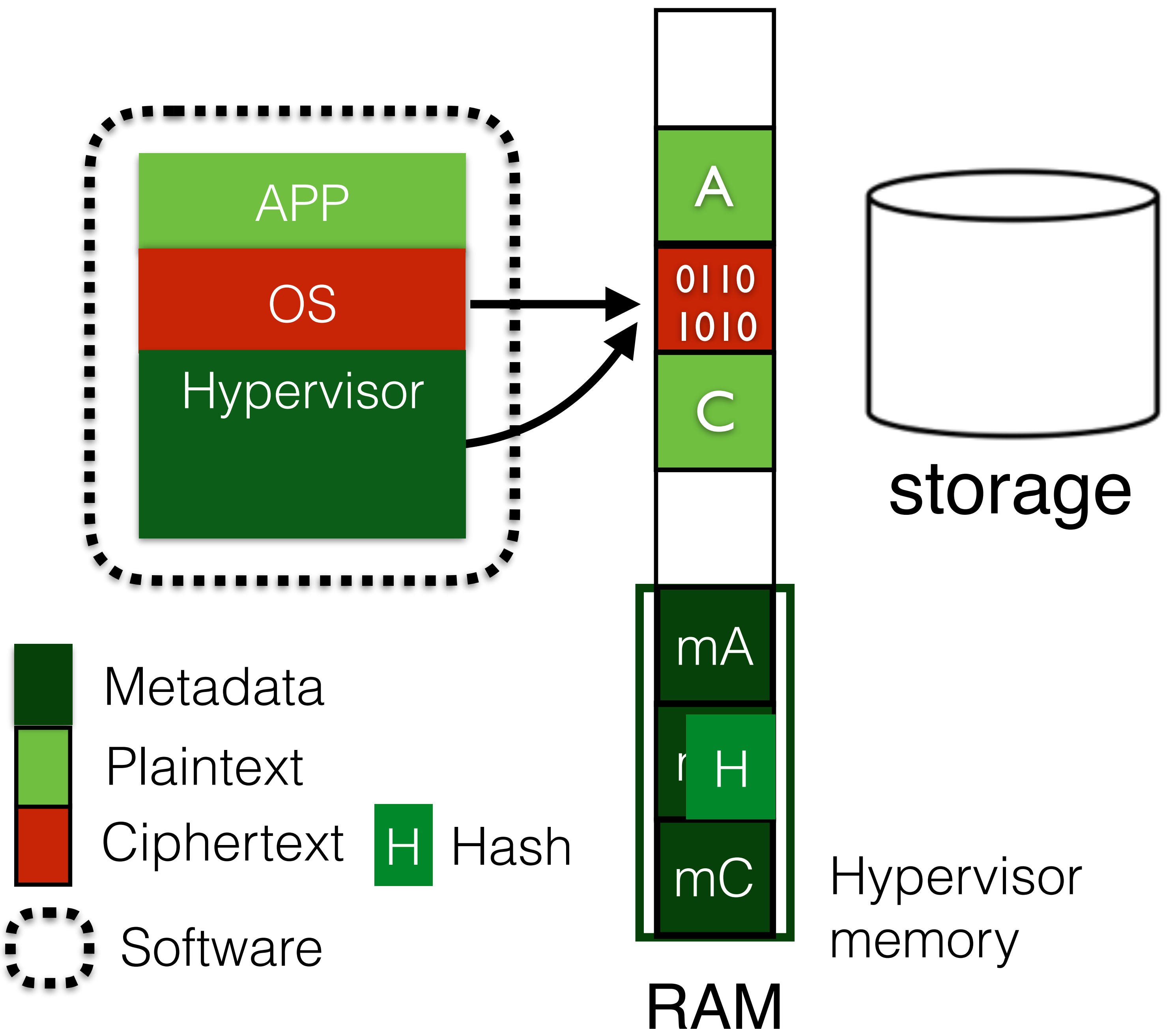
2. OS wants to swap page

Hypervisor hashes memory for integrity



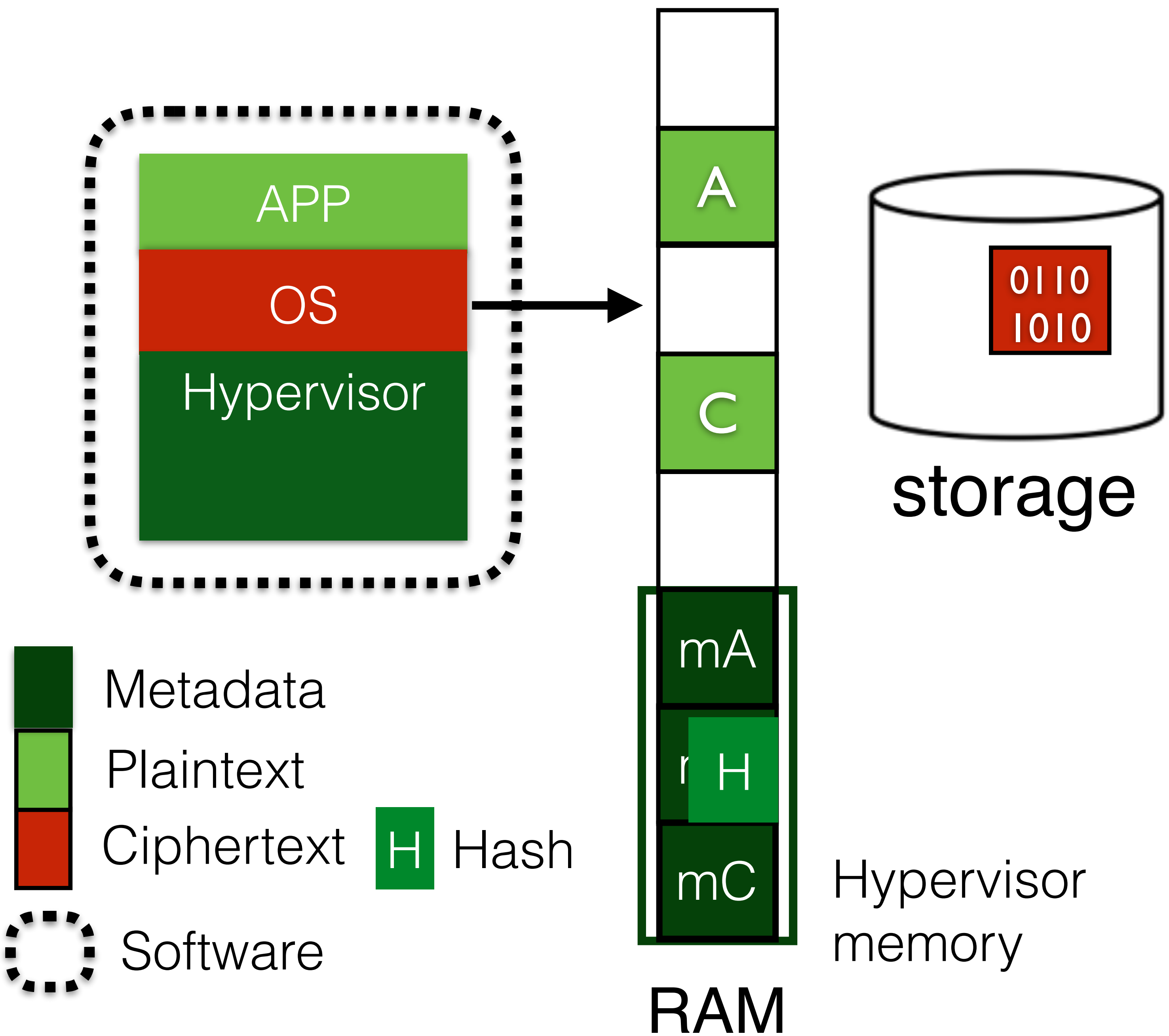
1. APP reads/writes memory page
   a) HYP maintains metadata
2. OS wants to swap page
3. Hypervisor blocks OS
   a) Encrypts page
   b) Hashes page

Hypervisor hashes memory for integrity



1. APP reads/writes memory page
   a) HYP maintains metadata
2. OS wants to swap page
3. Hypervisor blocks OS
   a) Encrypts page
   b) Hashes page

Hypervisor hashes memory for integrity



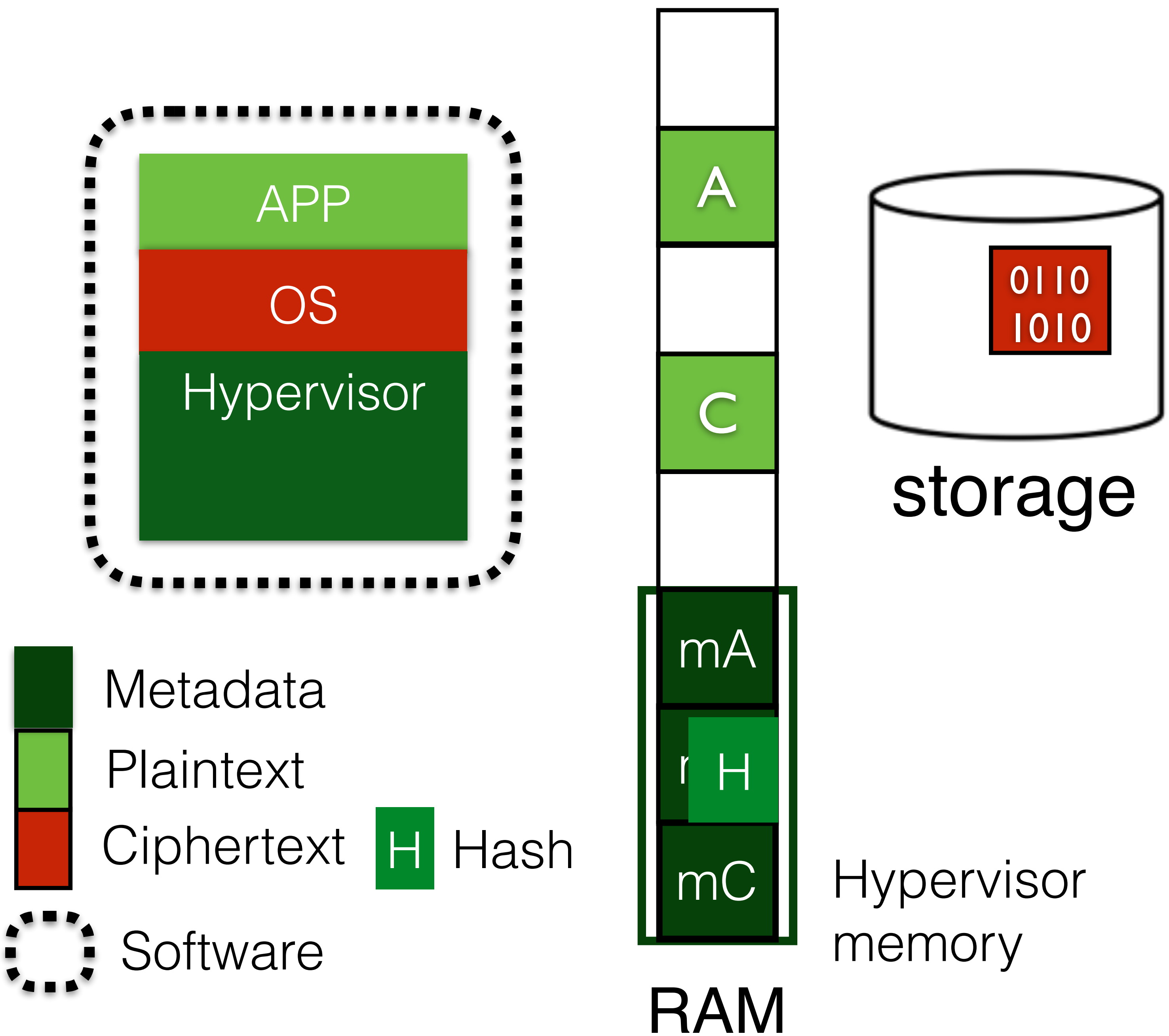
1. APP reads/writes memory page
   a) HYP maintains metadata

2. OS wants to swap page

3. Hypervisor blocks OS
   a) Encrypts page
   b) Hashes page

4. OS swaps the encrypted page

Hypervisor hashes memory for integrity



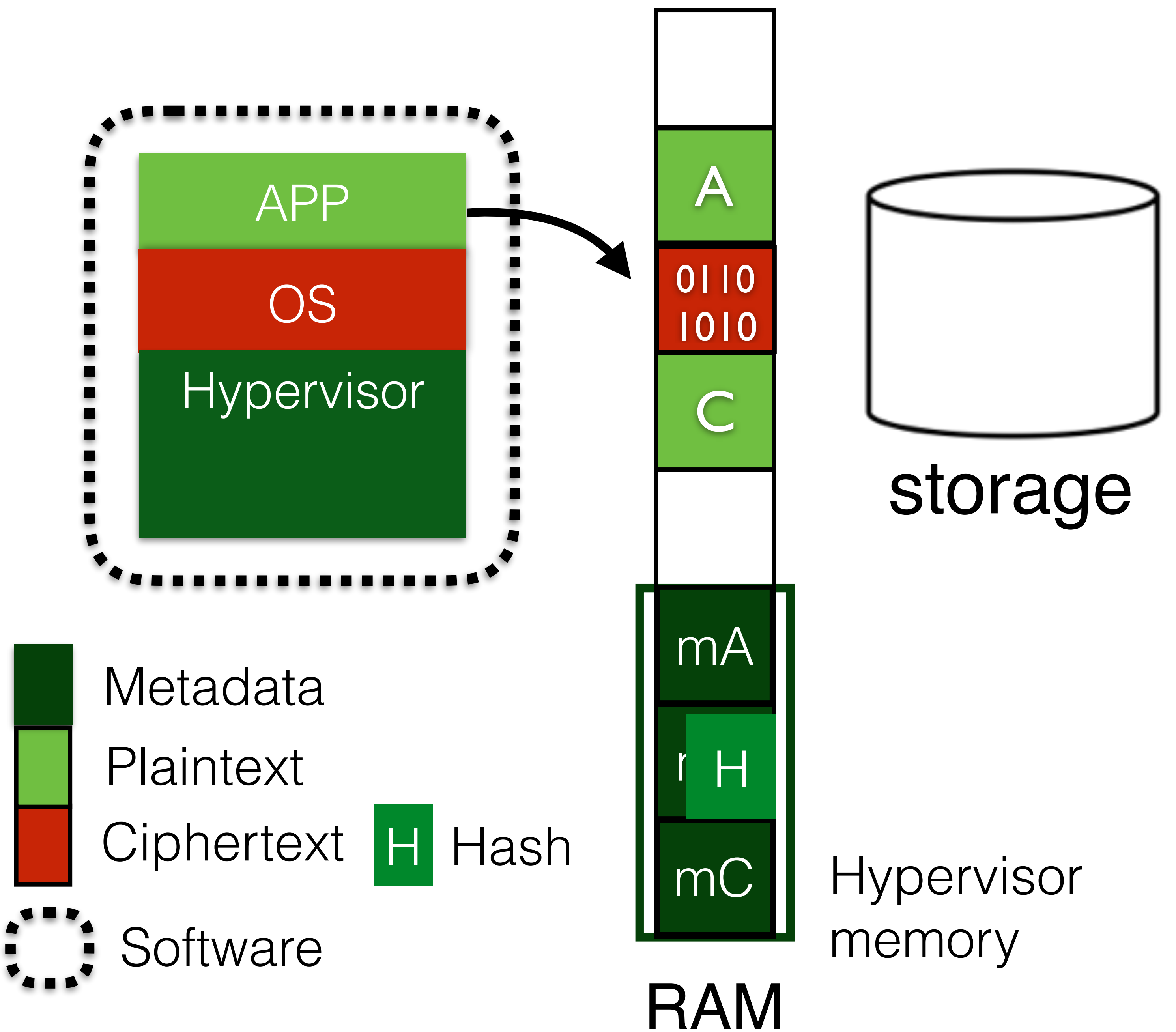
1. APP reads/writes memory page
   a) HYP maintains metadata

2. OS wants to swap page

3. Hypervisor blocks OS
   a) Encrypts page
   b) Hashes page

4. OS swaps the encrypted page

Hypervisor hashes memory for integrity



1. APP reads/writes memory page
   a) HYP maintains metadata

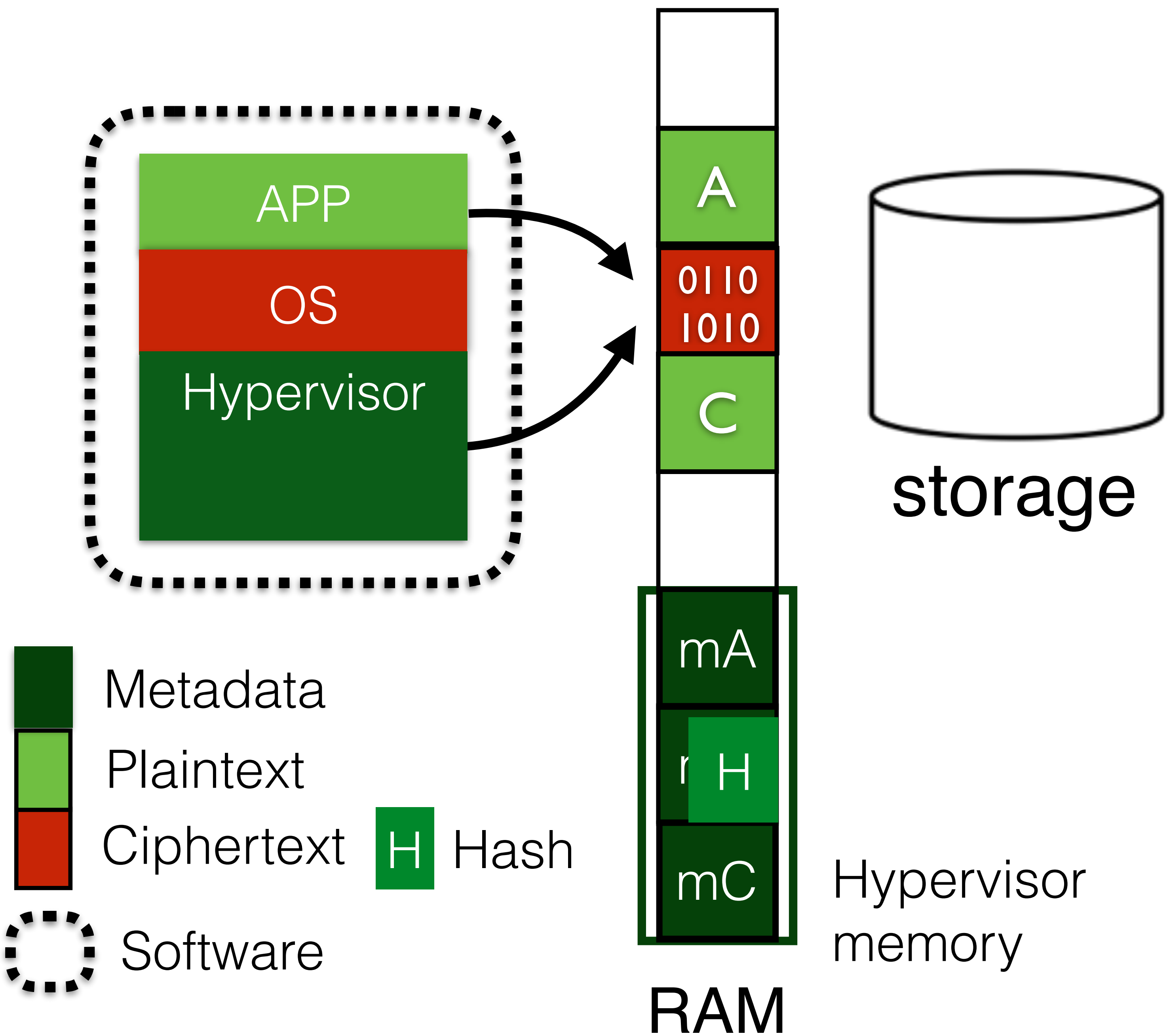
2. OS wants to swap page

3. Hypervisor blocks OS
   a) Encrypts page
   b) Hashes page

4. OS swaps the encrypted page

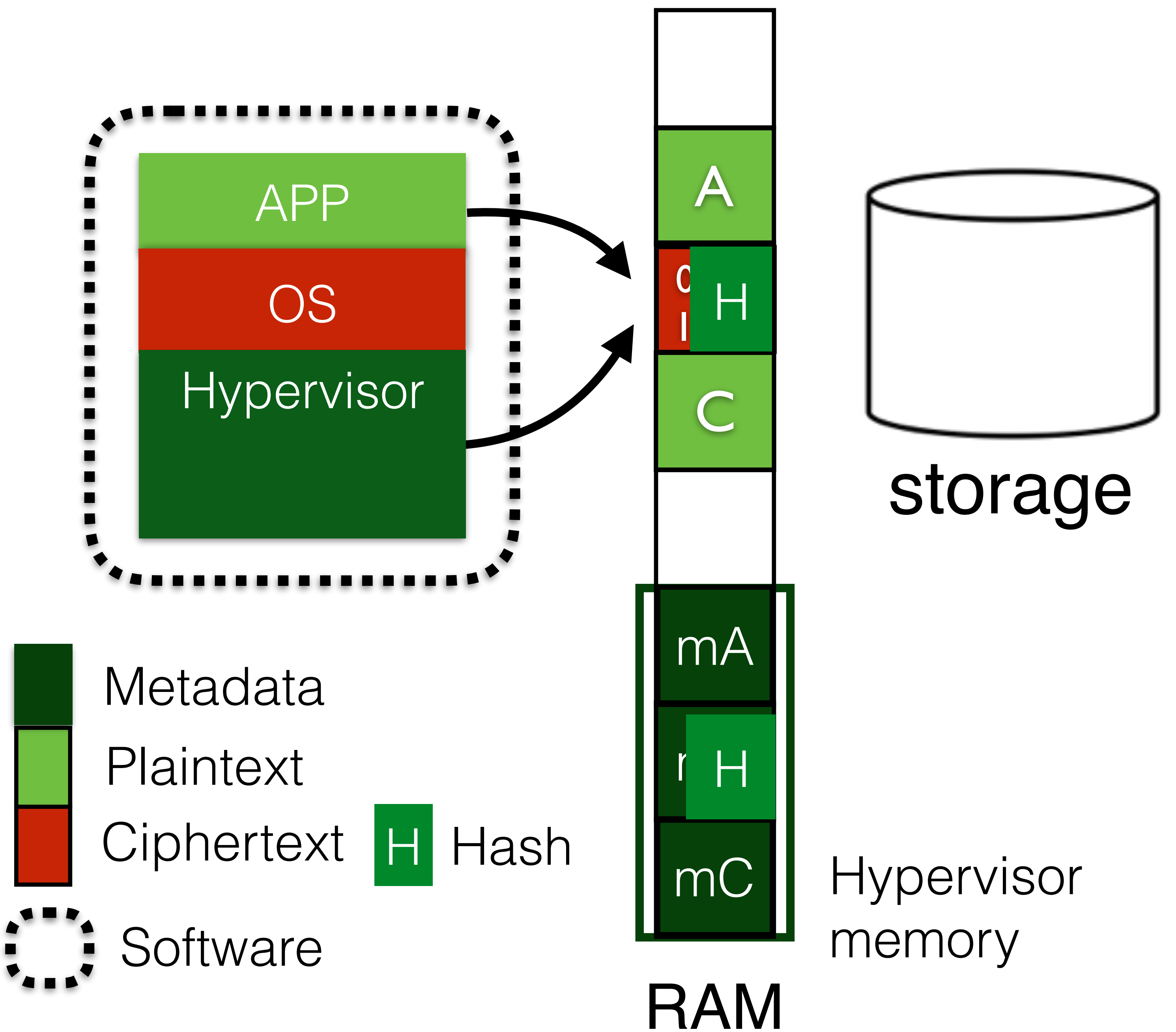
5. APP accesses page
   a) OS swaps in
   b) HYP checks hash
   c) HYP decrypts page

Hypervisor hashes memory for integrity



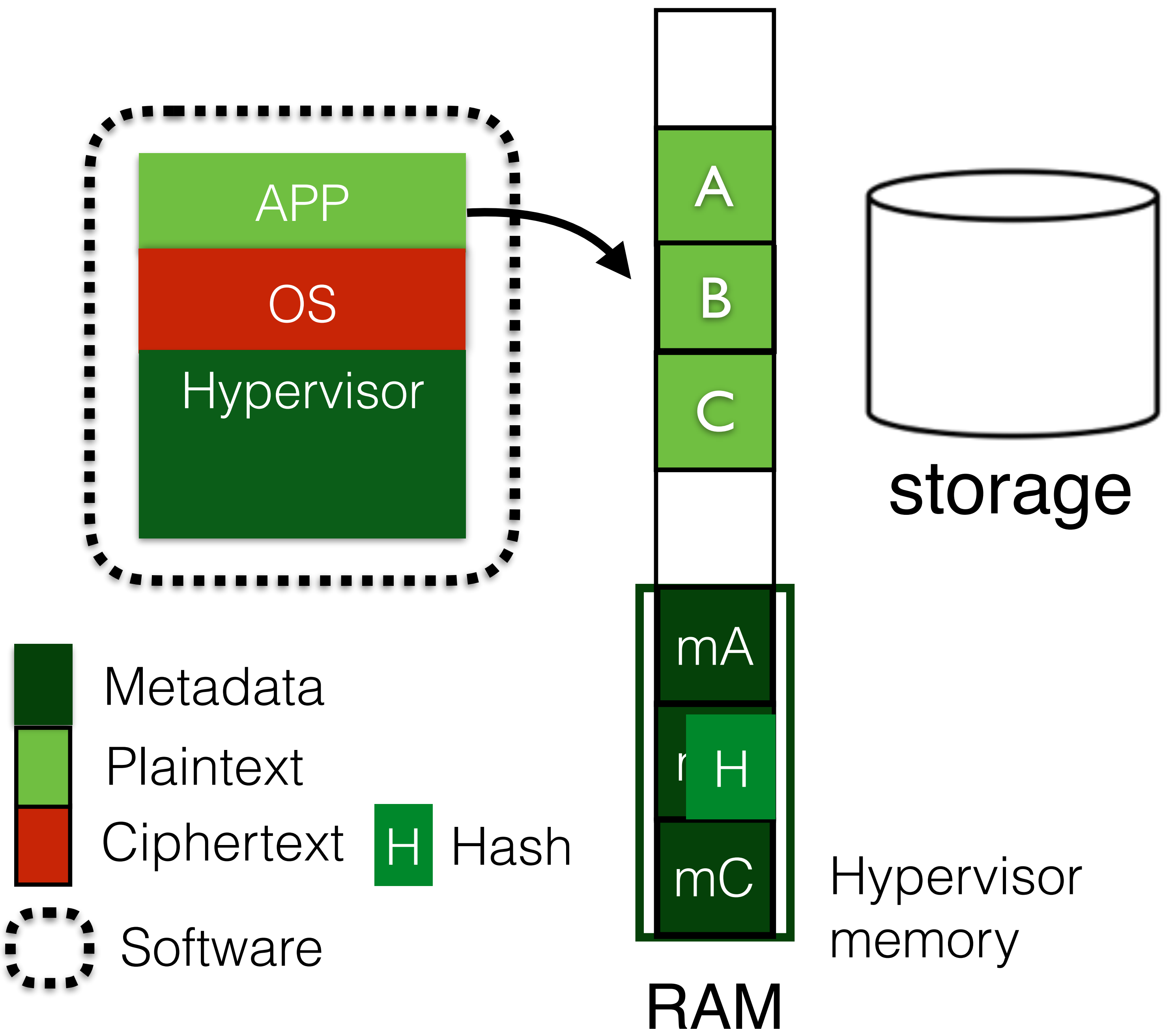
1. APP reads/writes memory page
   a) HYP maintains metadata
2. OS wants to swap page
3. Hypervisor blocks OS
   a) Encrypts page
   b) Hashes page
4. OS swaps the encrypted page
5. APP accesses page
   a) OS swaps in
   b) HYP checks hash
   c) HYP decrypts page

Hypervisor hashes memory for integrity



1. APP reads/writes memory page
   a) HYP maintains metadata
2. OS wants to swap page
3. Hypervisor blocks OS
   a) Encrypts page
   b) Hashes page
4. OS swaps the encrypted page
5. APP accesses page
   a) OS swaps in
   b) HYP checks hash
   c) HYP decrypts page

Hypervisor hashes memory for integrity



1. APP reads/writes memory page
   a) HYP maintains metadata

2. OS wants to swap page

3. Hypervisor blocks OS
   a) Encrypts page
   b) Hashes page

4. OS swaps the encrypted page

5. APP accesses page
   a) OS swaps in
   b) HYP checks hash
   c) HYP decrypts page

Performance cost of encryption and hashing

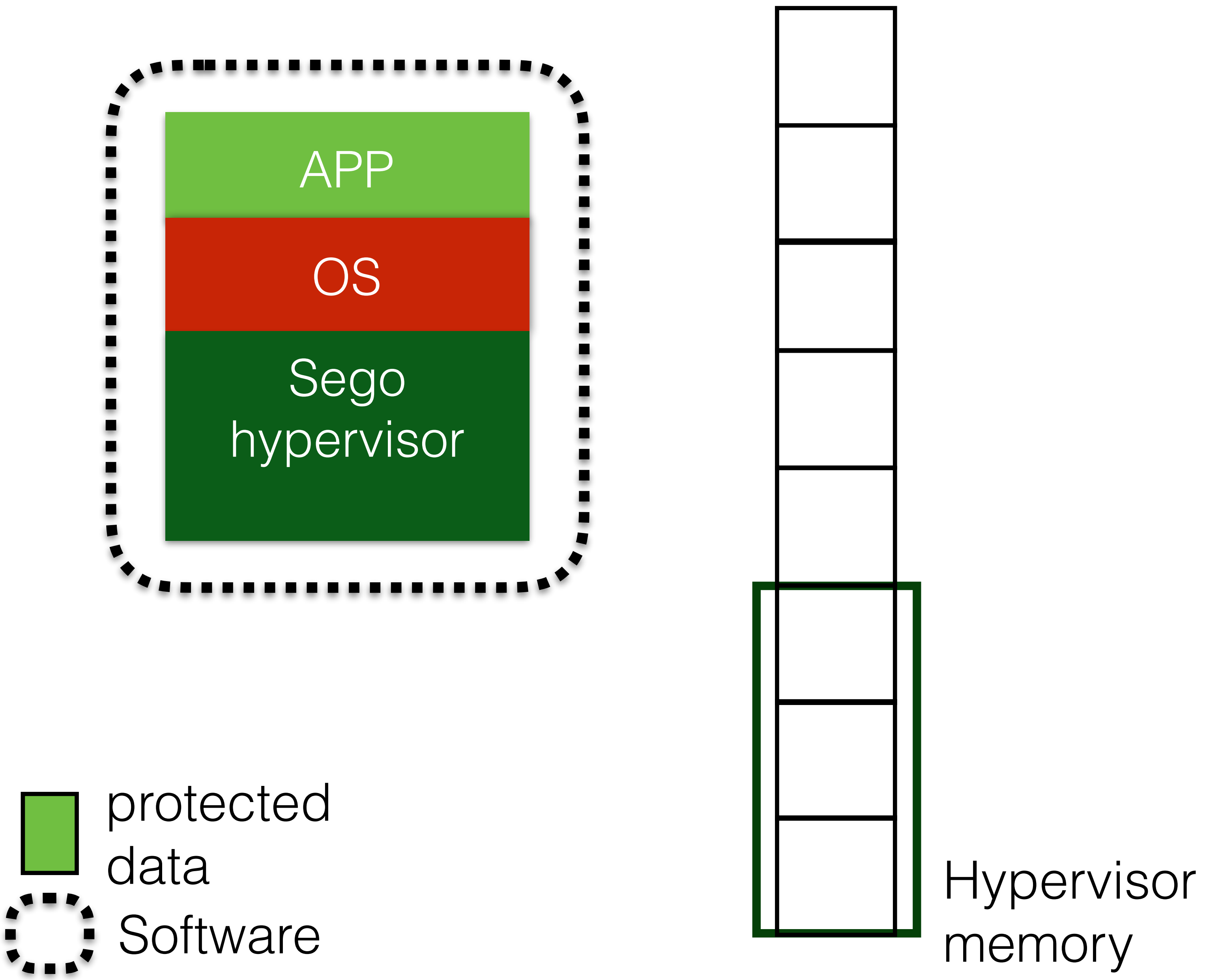
- Performance of encryption and hashing
  - AES-NI (GCM) supported in processor
  - 800MB/s - 1.2 GB/s
- Performance of a single IO device
  - Commodity SSD : 520MB/s
  - Fusion-io ioDrive : 1GB ~ 1.5GB/s
- IO bandwidth can overwhelm encryption bandwidth!

OS Memory Services

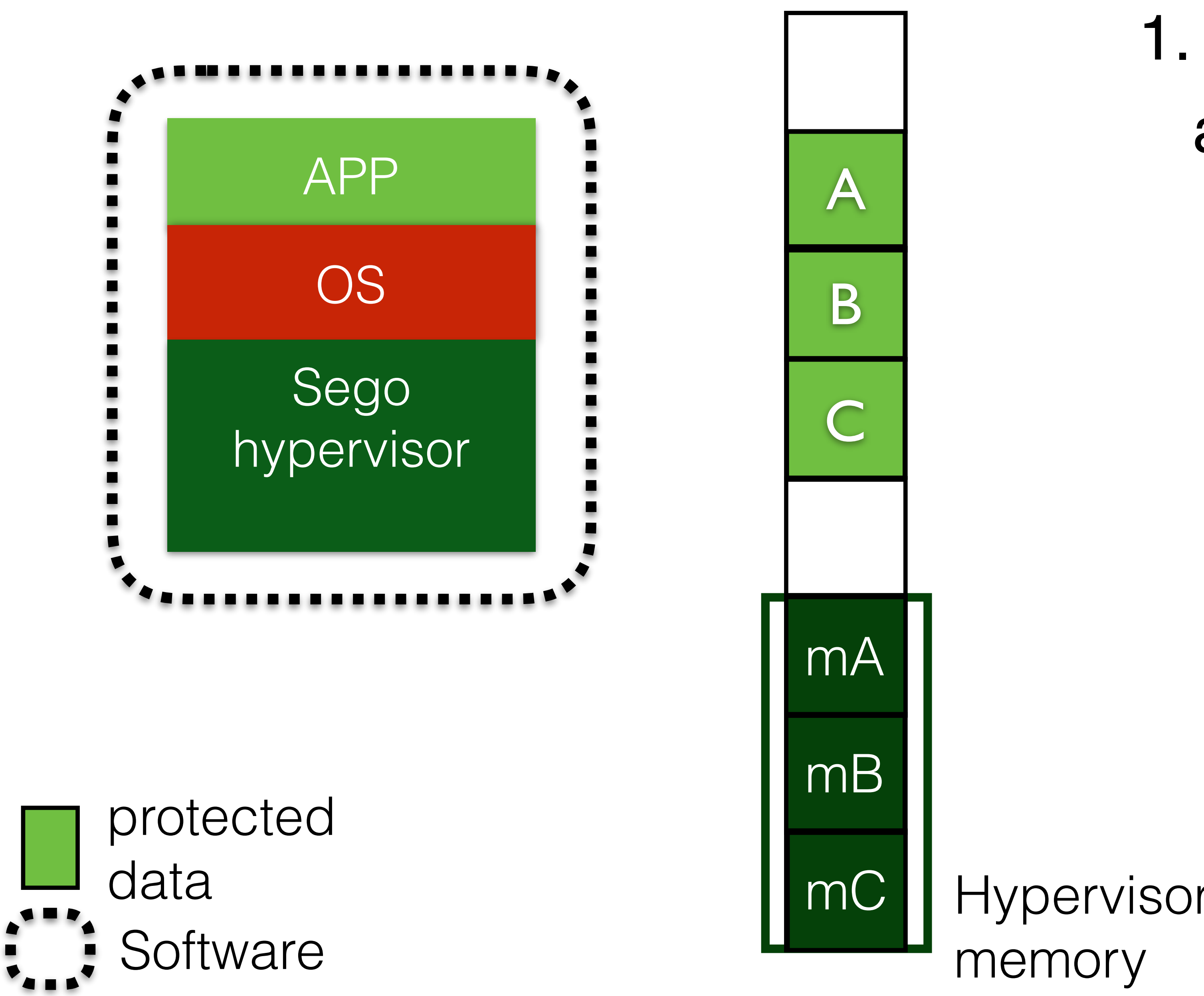
- Modern services require OS to touch memory
  - Transparent page sharing
    - Multiple virtual machines consume less memory
    - Overshadow/InkTag can not support it
  - Memory compaction
    - OS defragments memory for large pages
    - Better TLB utilization
- We must make OS access to APP pages more efficient

Sego eliminates encryption and hashing by using trusted metadata

Replace encryption and hashing with hypercalls

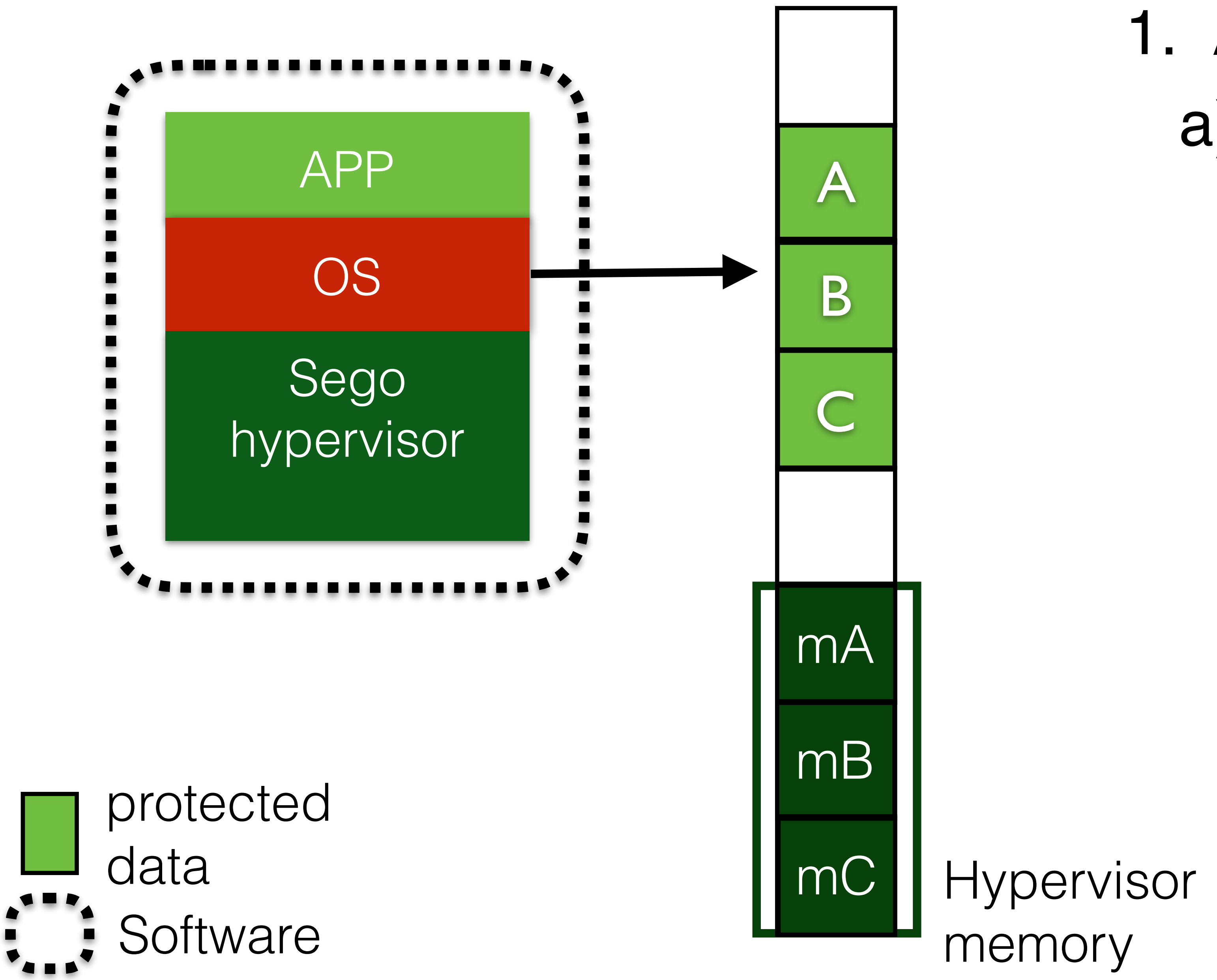


Replace encryption and hashing with hypercalls



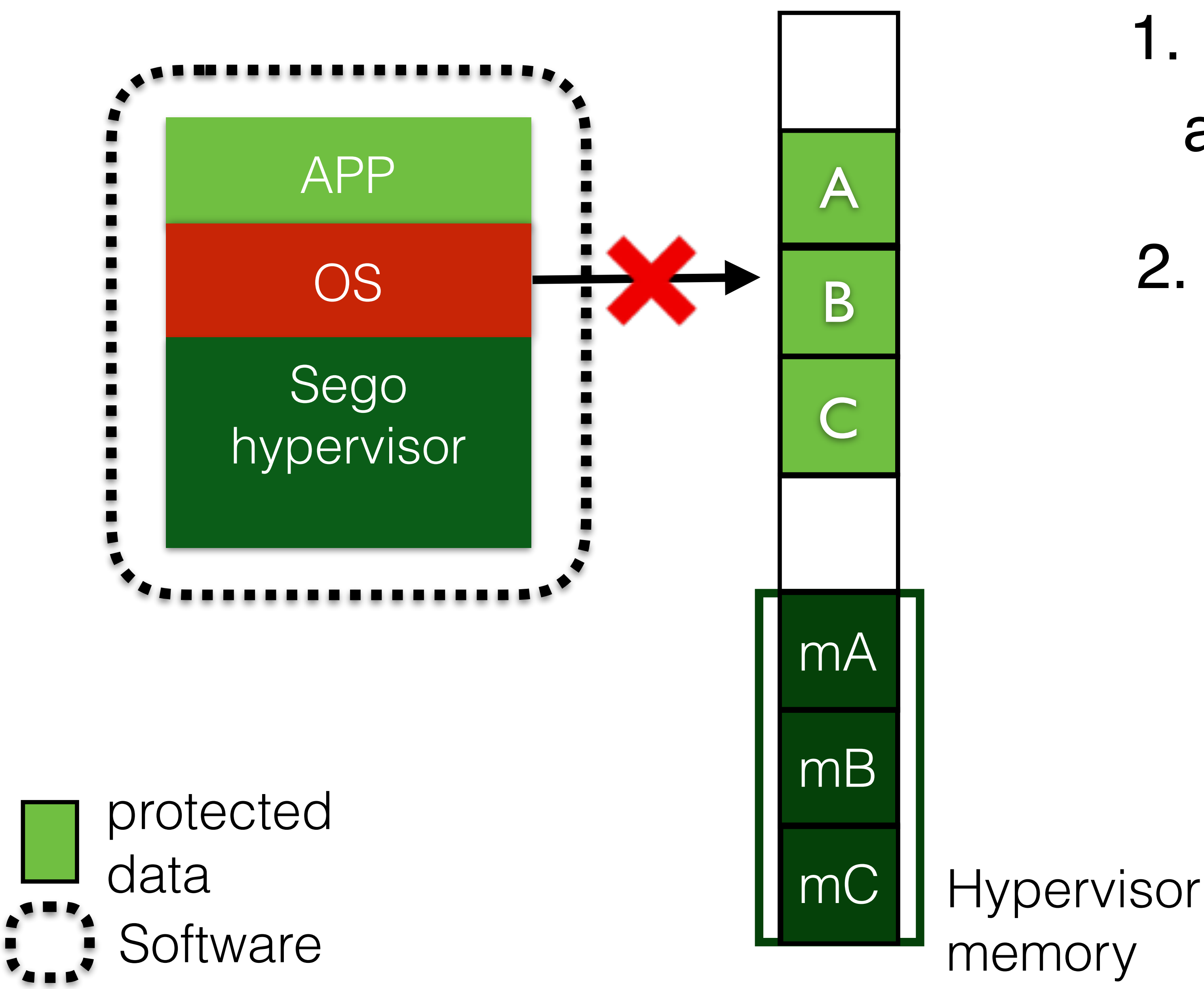
1. APP reads/writes memory page
   a) HYP maintains metadata

Replace encryption and hashing with hypercalls



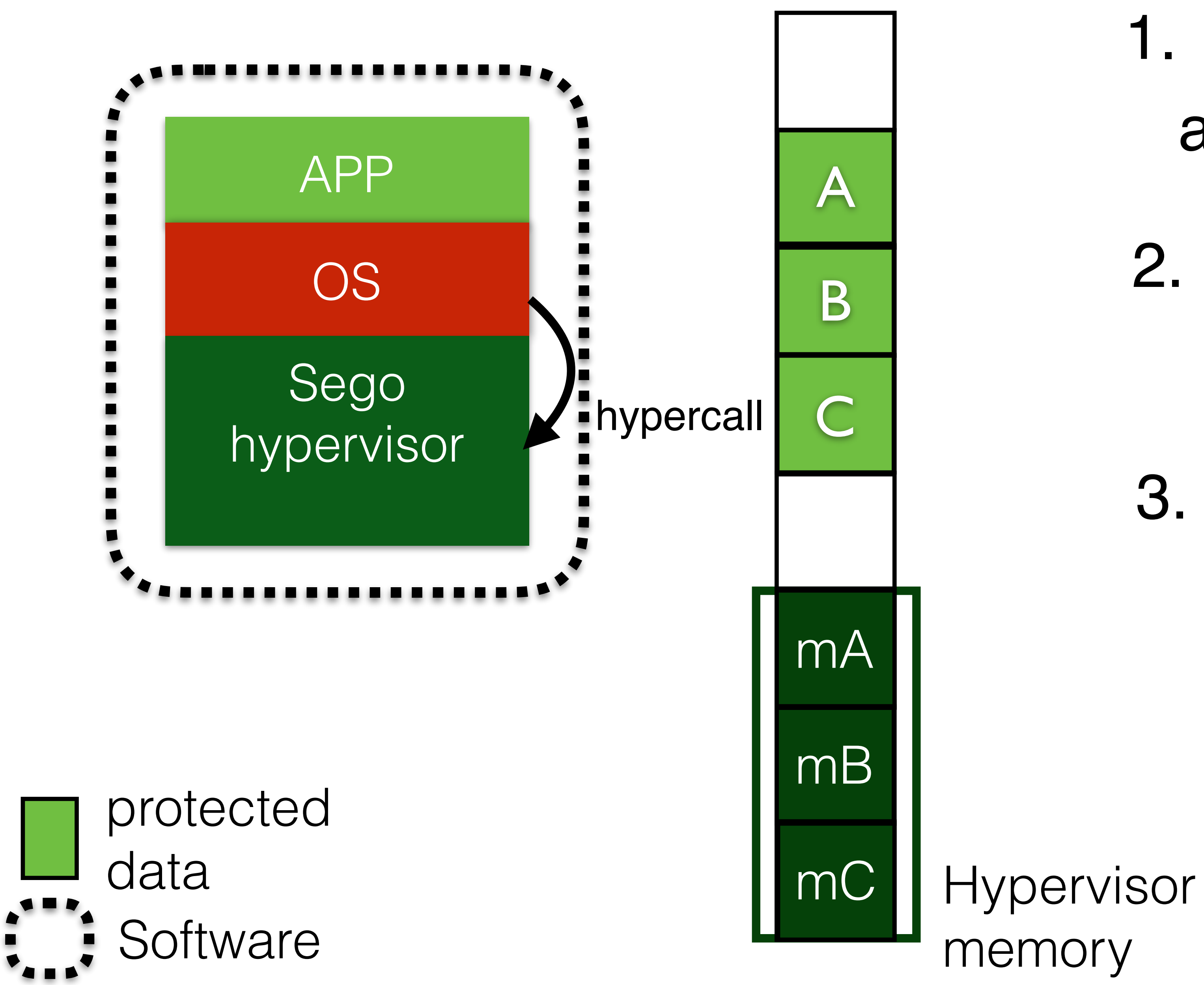
1. APP reads/writes memory page
   a) HYP maintains metadata

Replace encryption and hashing with hypercalls



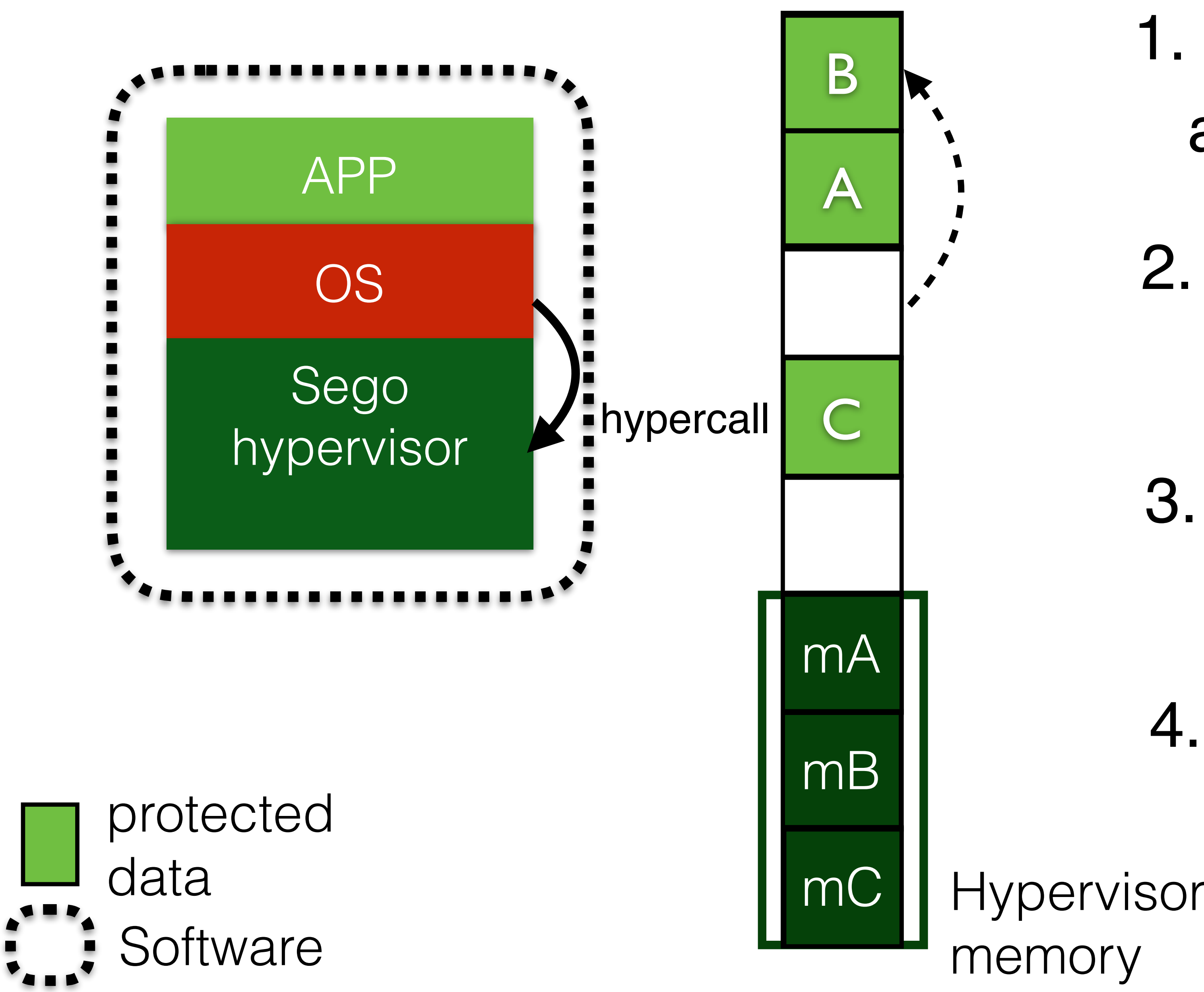
1. APP reads/writes memory page
   a) HYP maintains metadata
2. OS is not allowed to access protected memory pages

Replace encryption and hashing with hypercalls



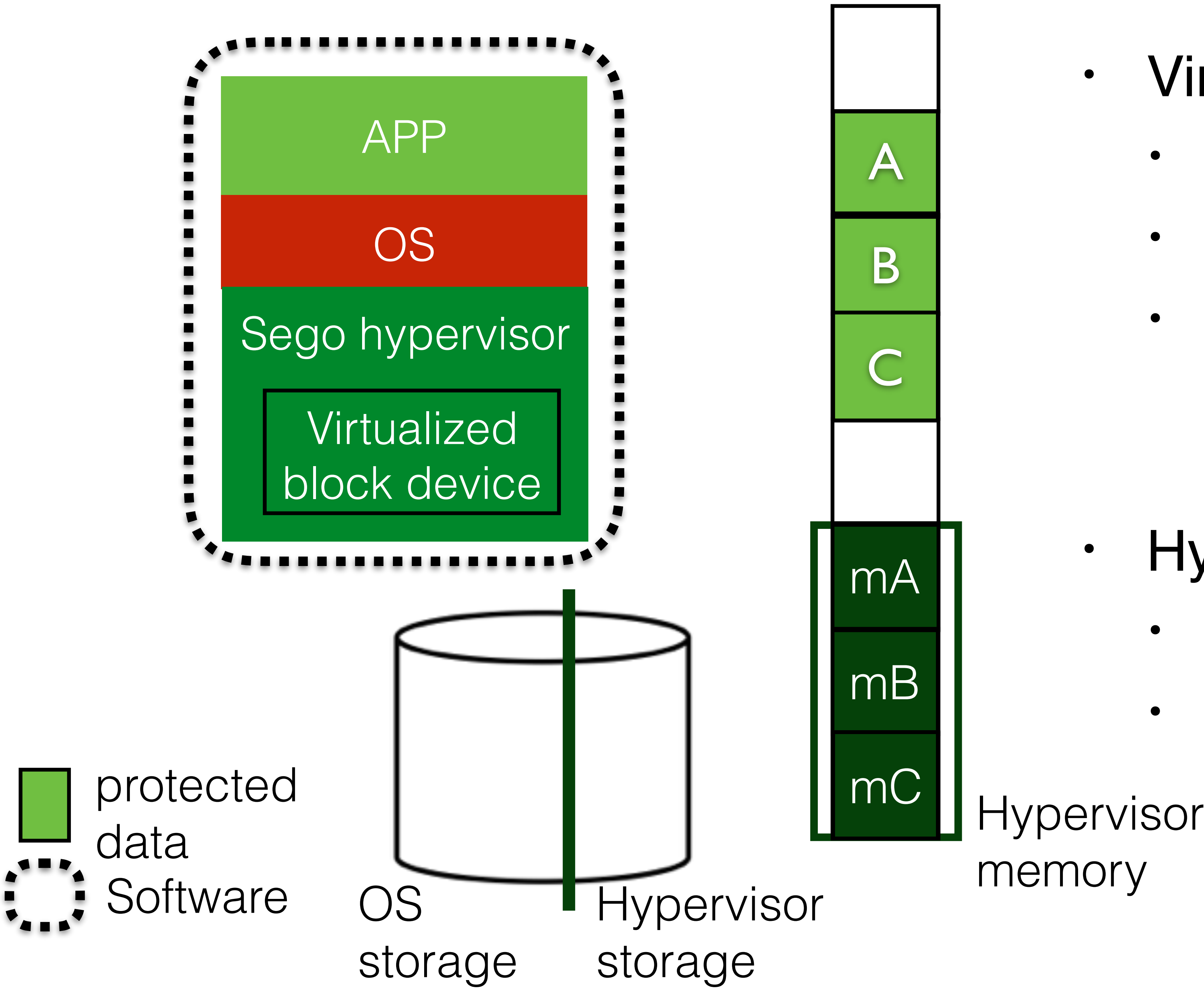
1. APP reads/writes memory page
   a) HYP maintains metadata
2. OS is not allowed to access protected memory pages
3. OS sends hypercall to move memory pages

Replace encryption and hashing with hypercalls



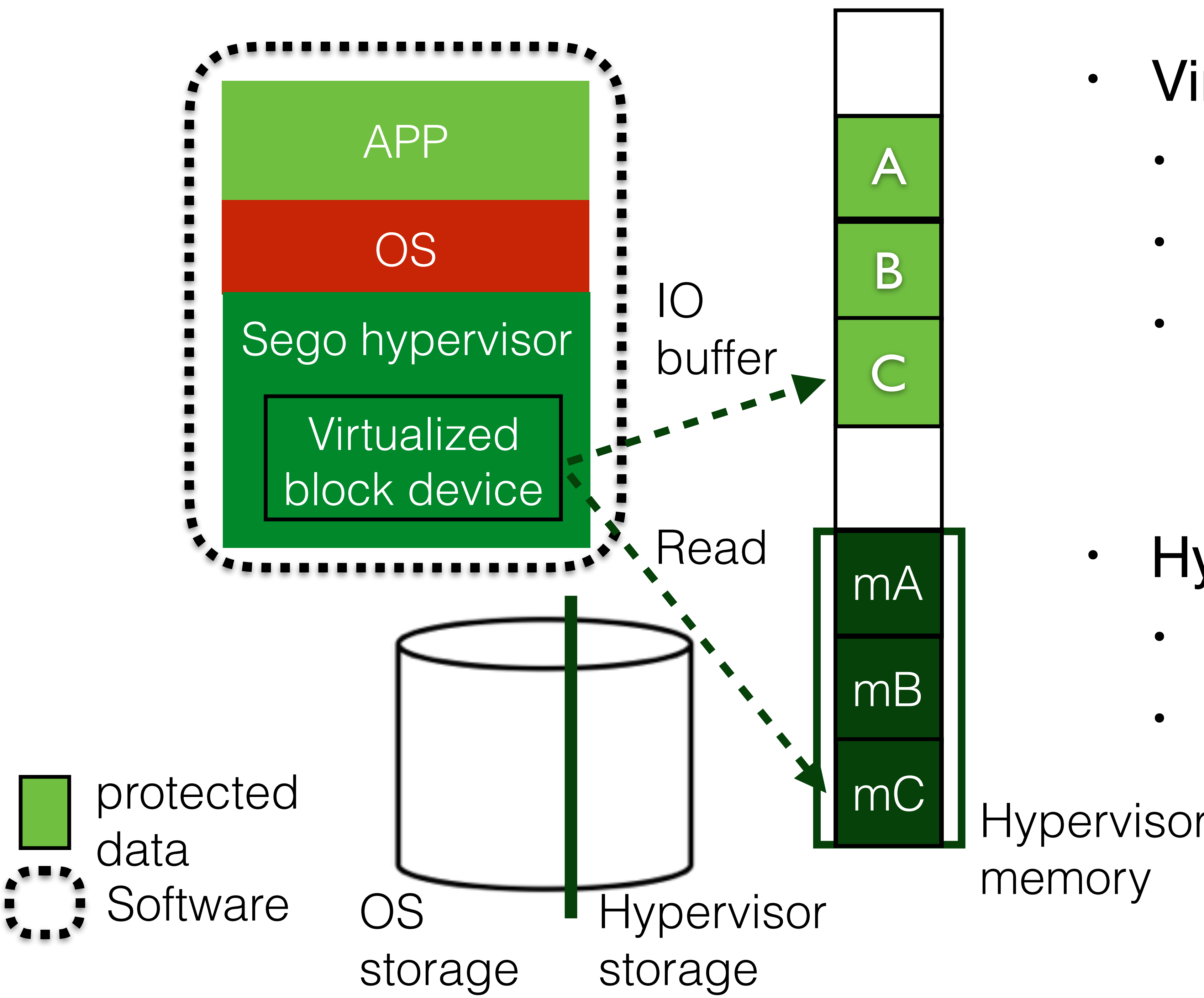
1. APP reads/writes memory page
   a) HYP maintains metadata
2. OS is not allowed to access protected memory pages
3. OS sends hypercall to move memory pages
4. Hypervisor moves the memory page

Sego persists data with metadata



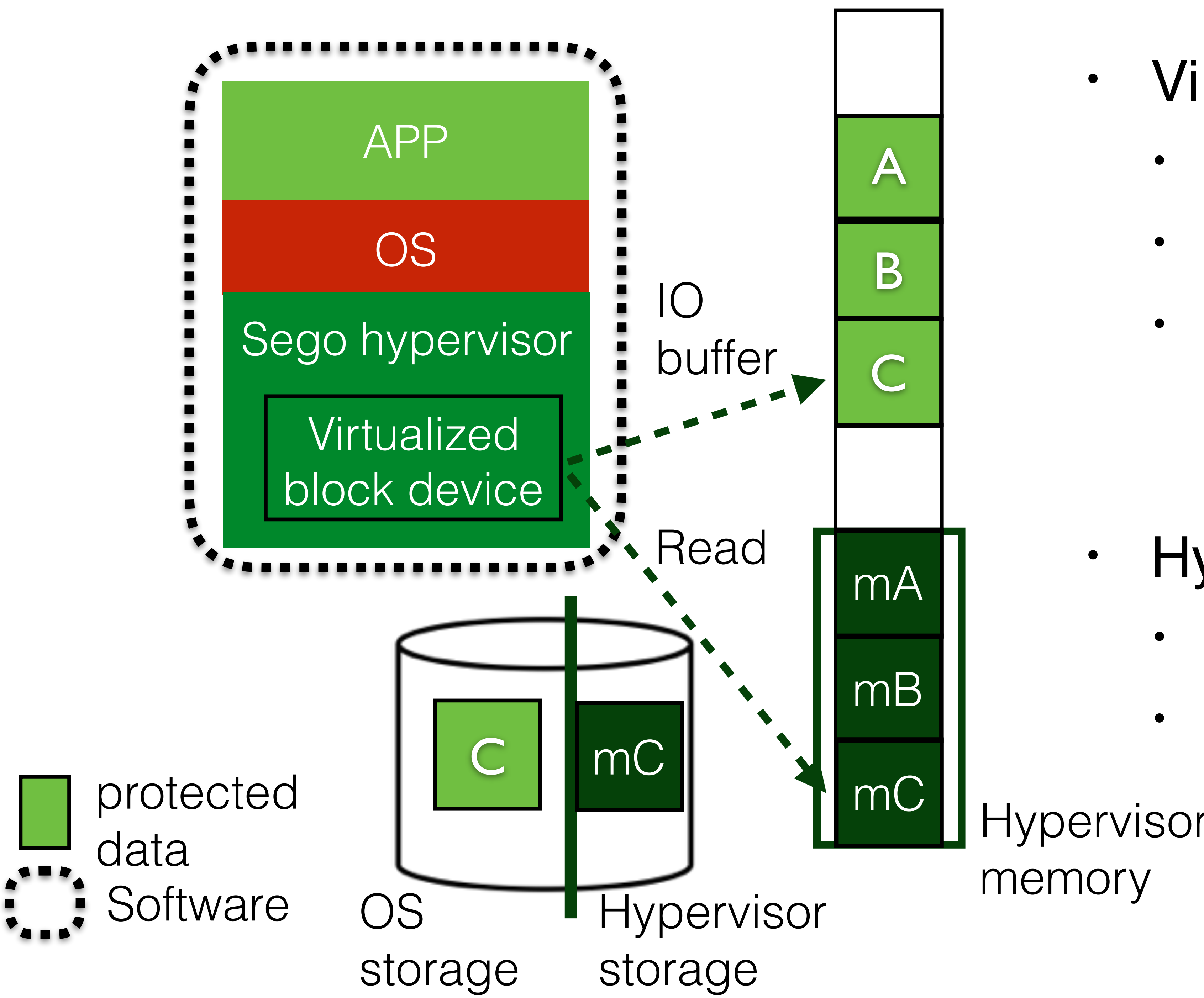
- Virtualized block device
  - Virtual hard disk/SSD
  - Sees/controls all I/O
  - Buffers guest IO in host memory
- Hypervisor storage
  - Invisible to OS
  - Holds trusted metadata

Sego persists data with metadata



- Virtualized block device
  - Virtual hard disk/SSD
  - Sees/controls all I/O
  - Buffers guest IO in host memory
- Hypervisor storage
  - Invisible to OS
  - Holds trusted metadata

Sego persists data with metadata

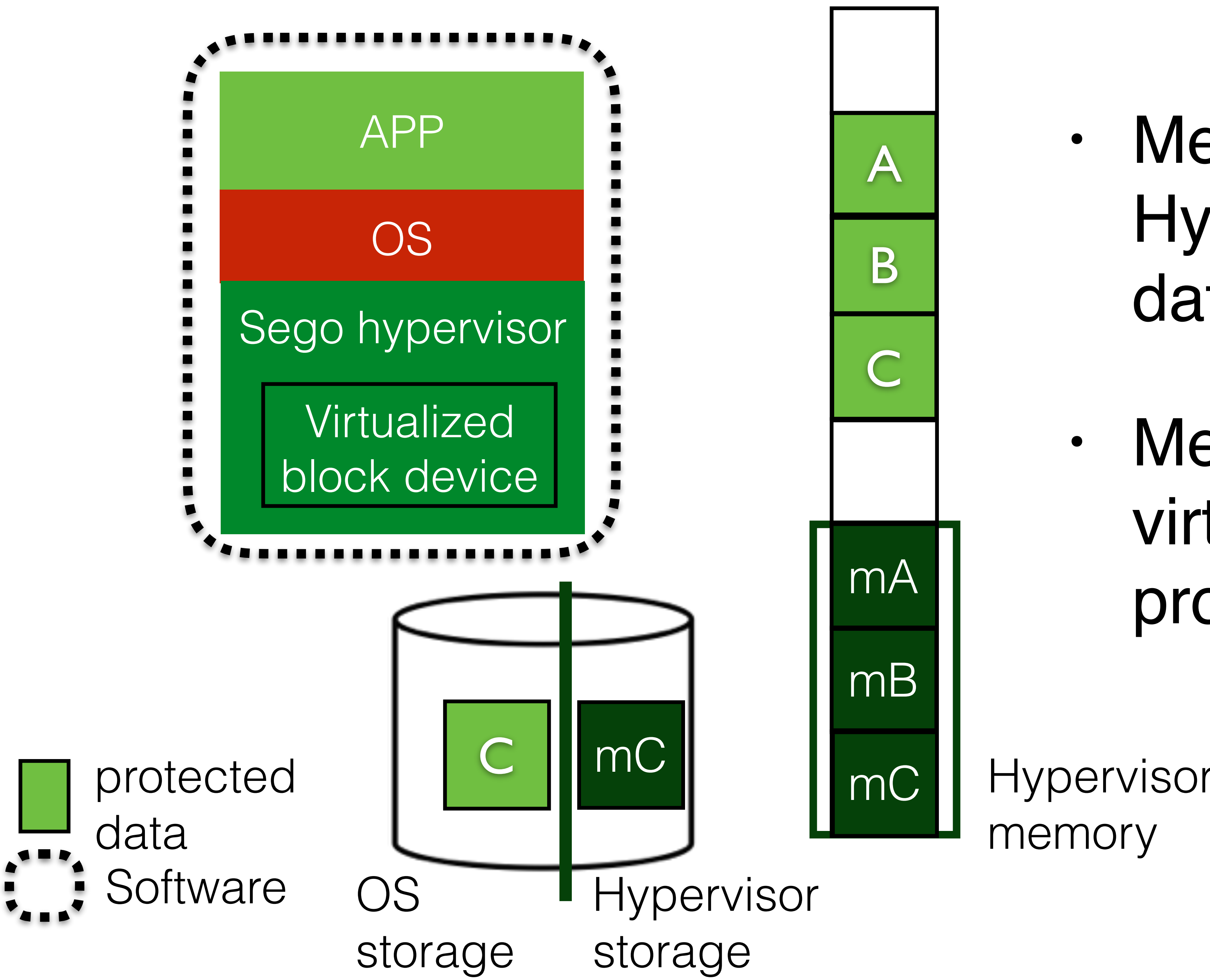


- Virtualized block device
  - Virtual hard disk/SSD
  - Sees/controls all I/O
  - Buffers guest IO in host memory
- Hypervisor storage
  - Invisible to OS
  - Holds trusted metadata

Pervasive trusted metadata

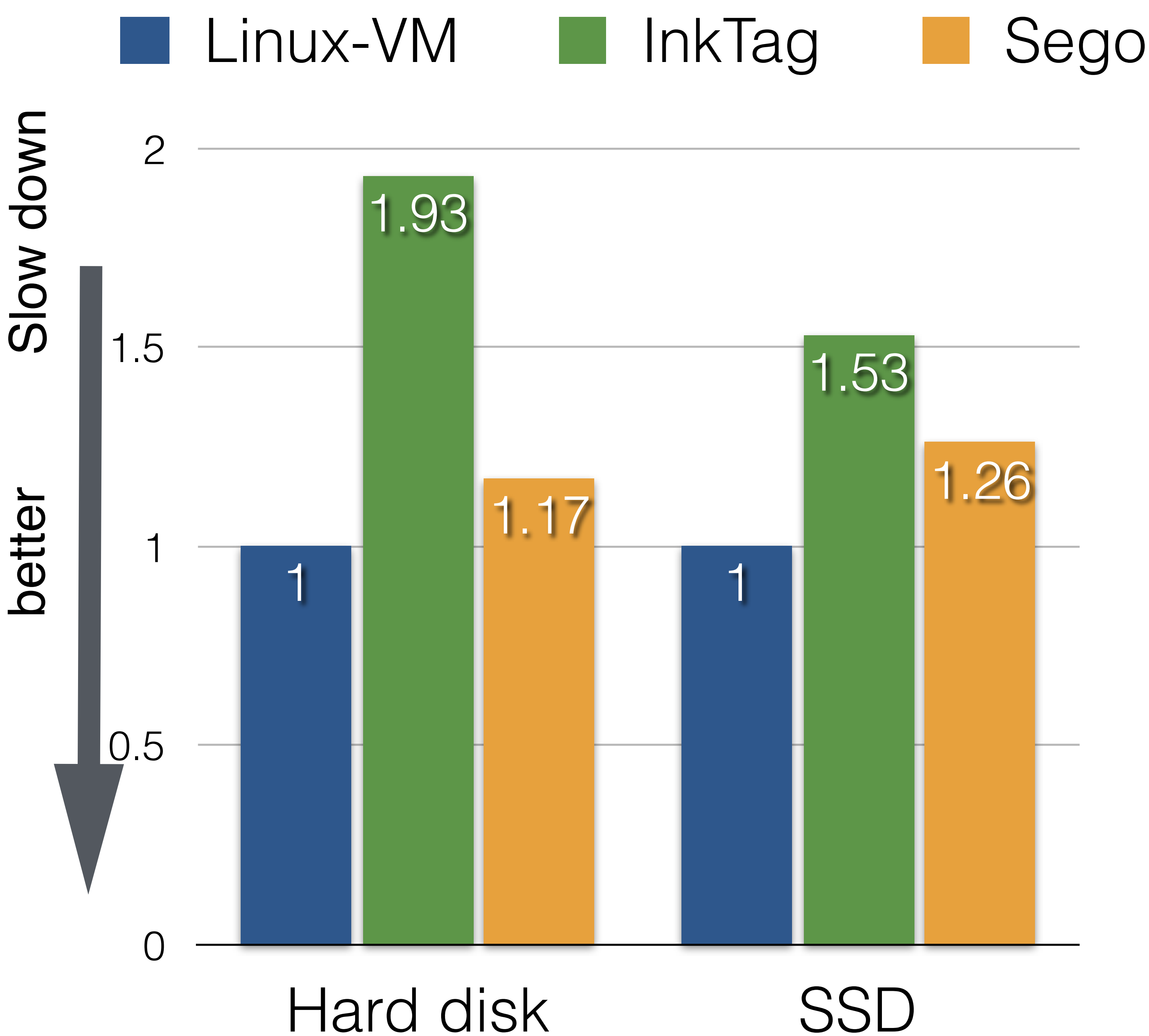
- Metadata is everywhere
  - To protect data in memory : hypervisor memory
  - To protect data in storage : hypervisor storage
- Metadata is shared
  - Hypervisor and virtualized block device share metadata

Sego protects data with pervasive metadata



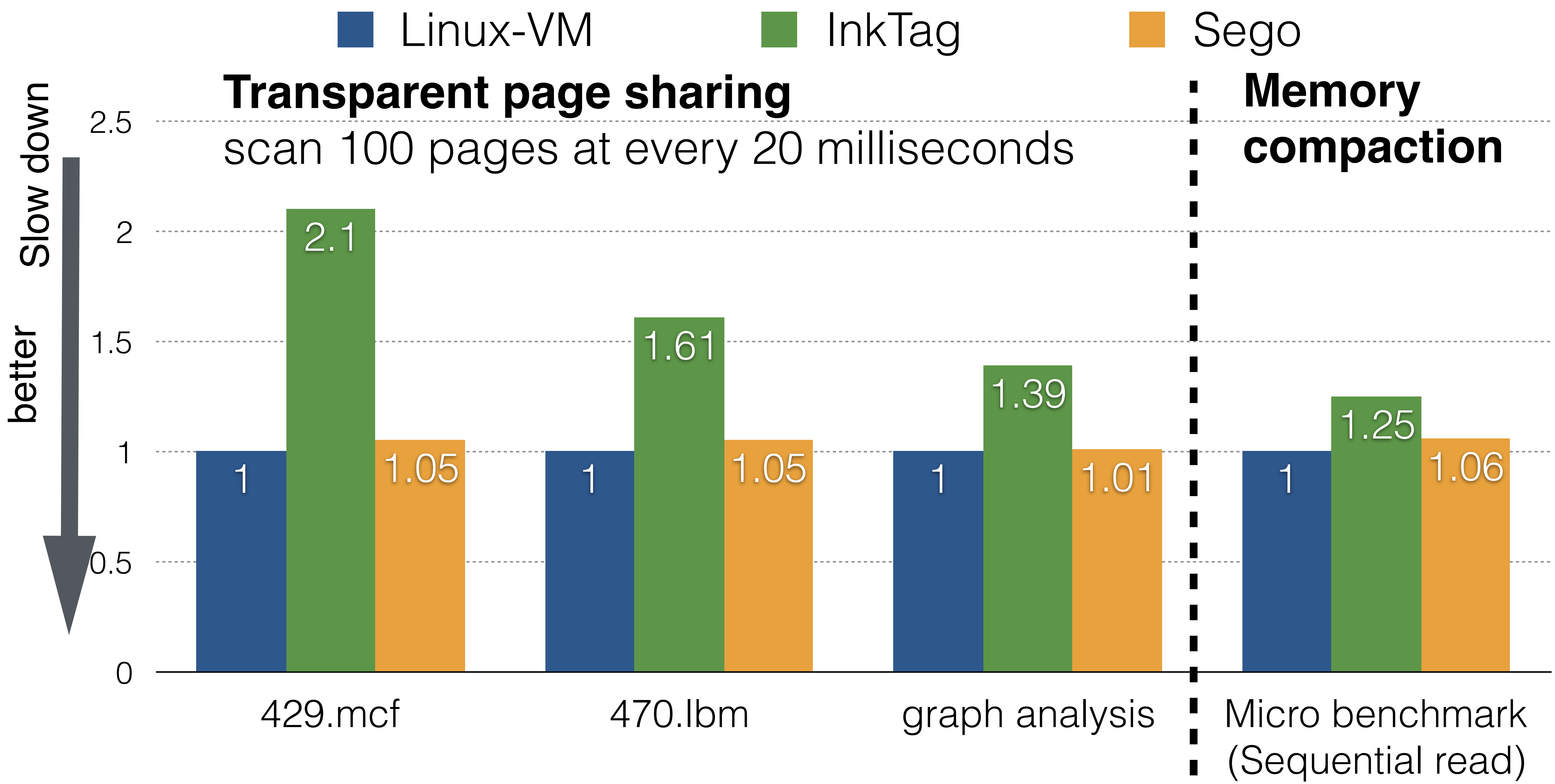
- Metadata in memory: for Hypervisor protecting data
- Metadata in storage: for virtualized block device protecting data

Sequential read



- InkTag/Overshadow
  - Protect app by encryption and hashing
- SSD (250MB/s)
  - 13 ~ 15% improvement by removing encryption and hashing
- Hard disk
  - IO batching optimization

OS touches protected memory



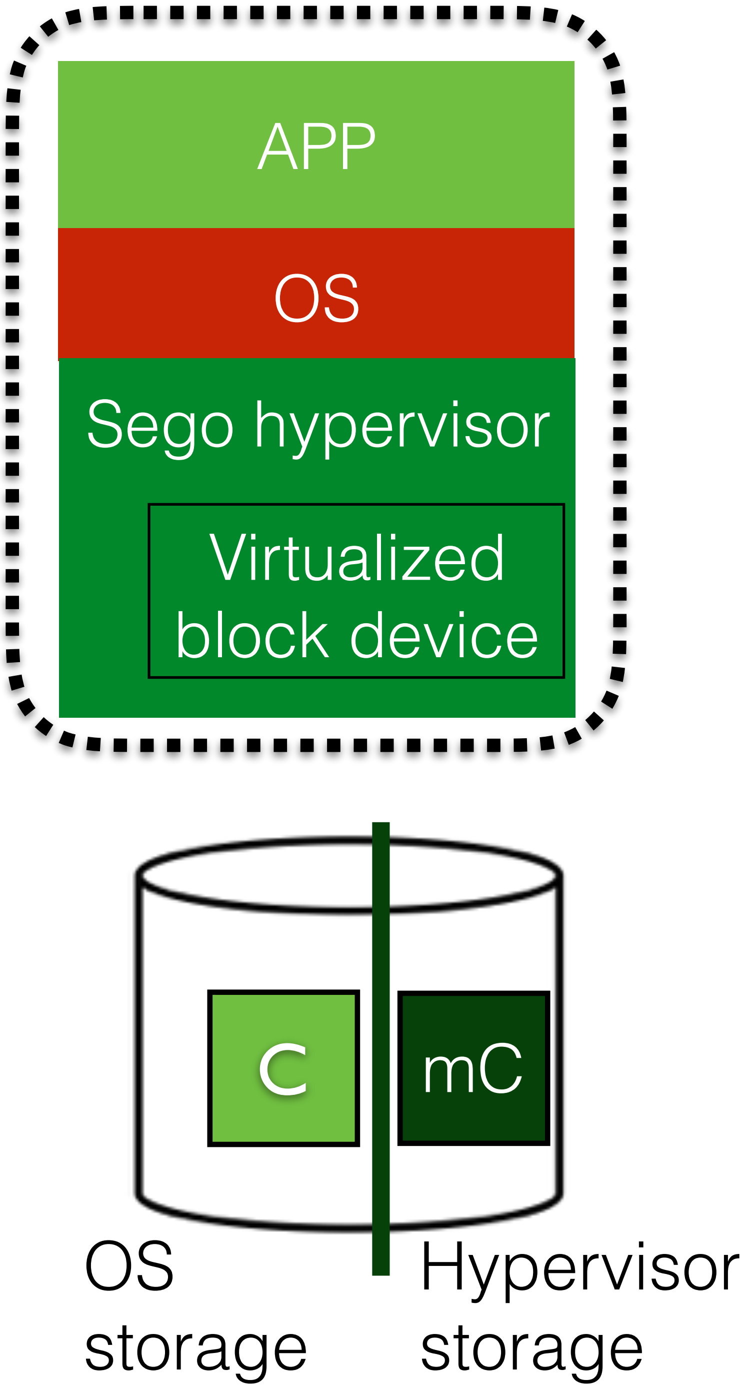
Sego provides crash consistency and secure recovery without trusting OS

Guest OS crash

- **hypervisor, virtualized block device, and metadata are alive**
- **APP and os are dead**

Hypervisor crash

Sego can’t trust OS journal

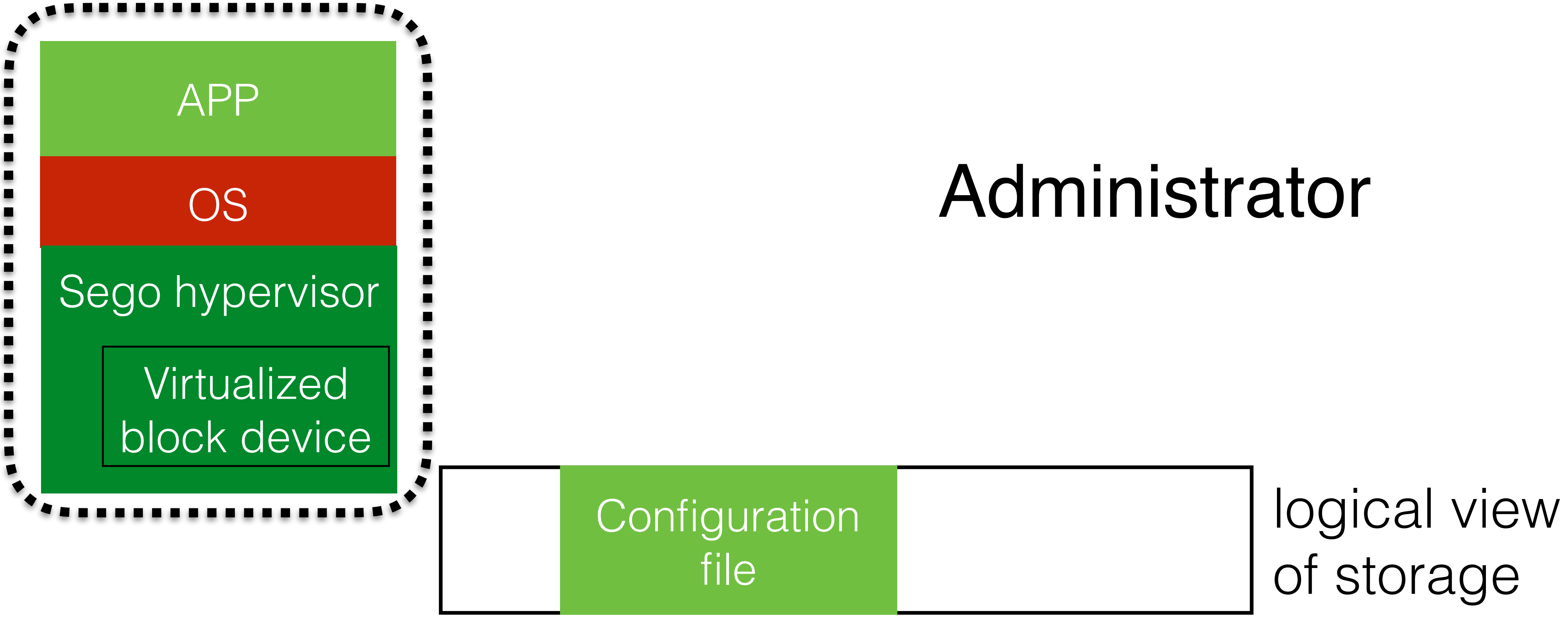


- Modern file systems use journals
- Journals have complex write ordering and recovery

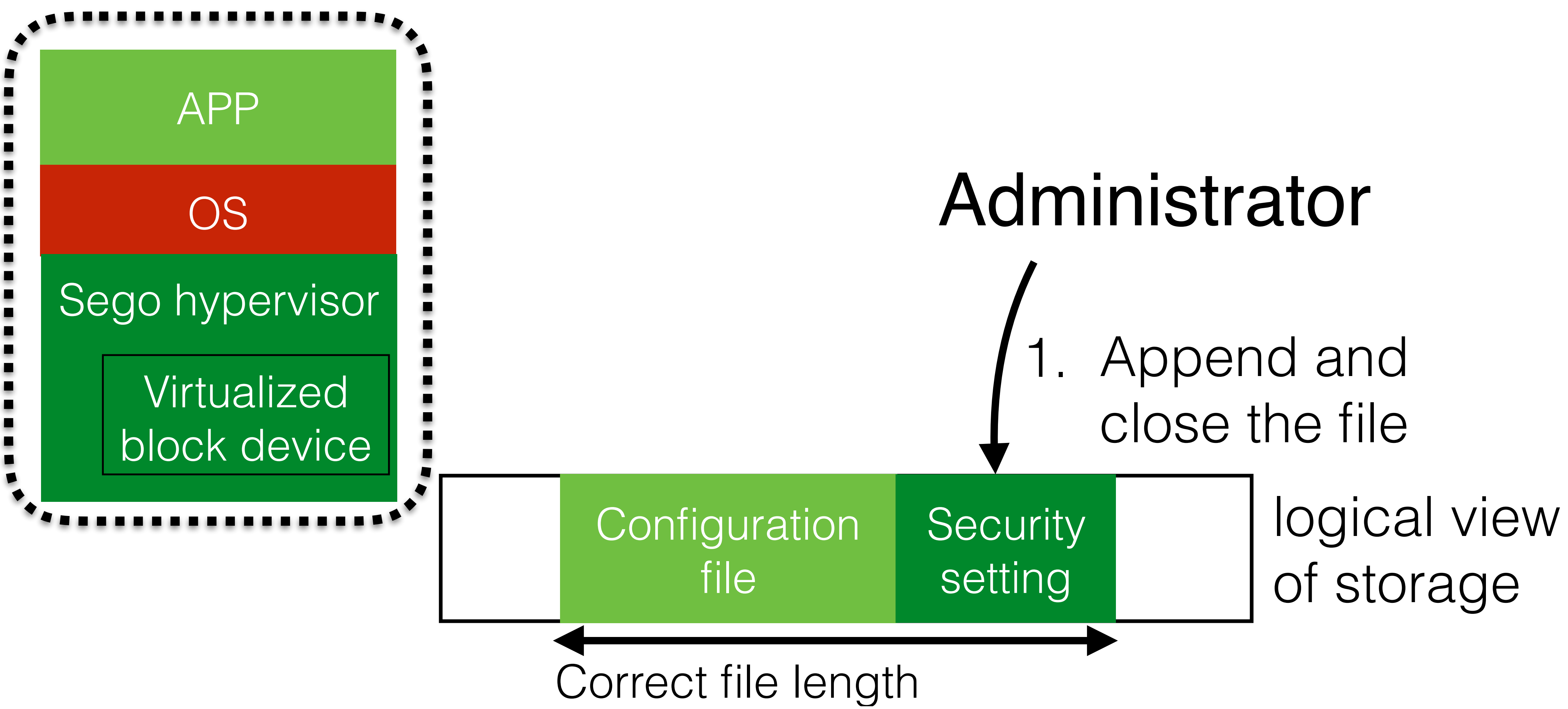
Challenges

- Journal makes recovery easier for OS
  - But more difficult for Sego!
  - Hypervisor cannot trust OS

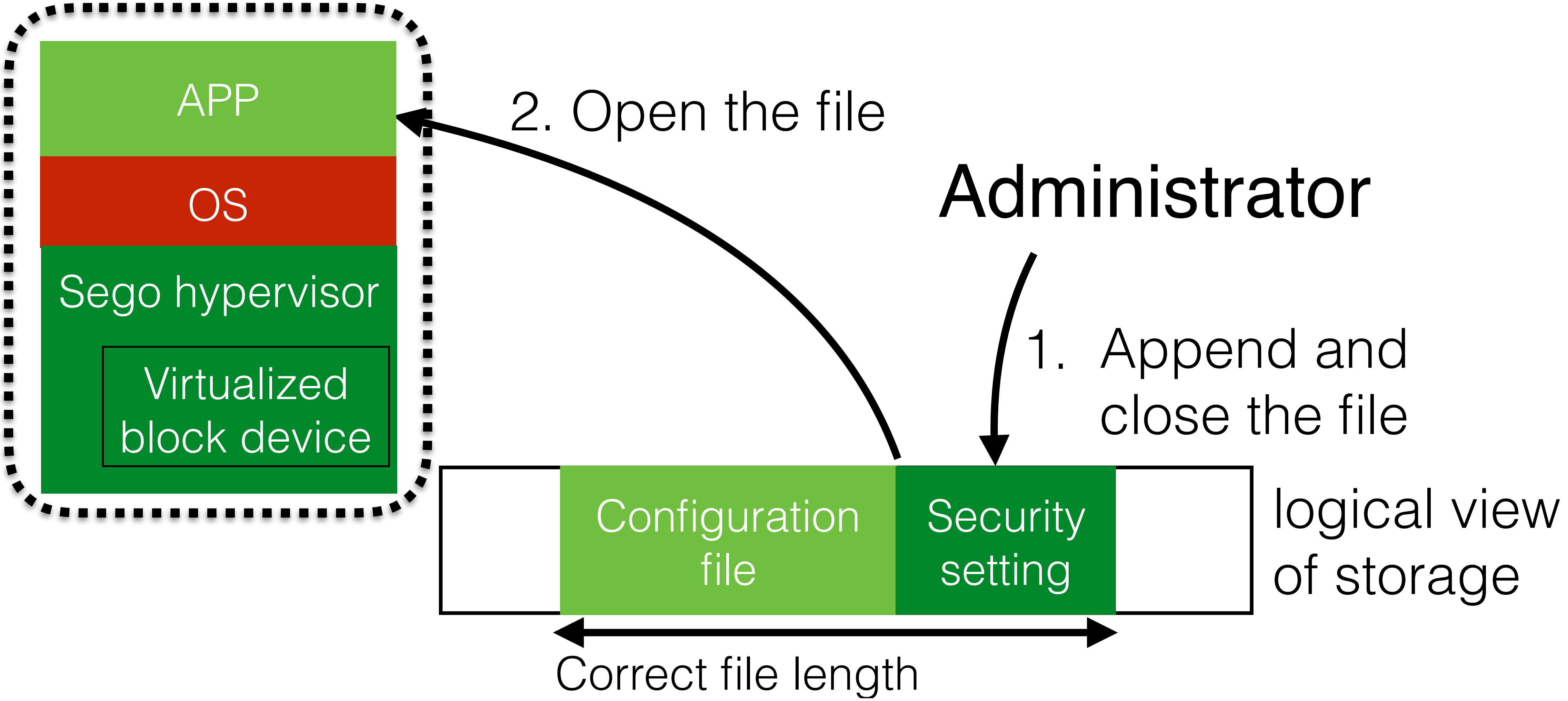
File length attack



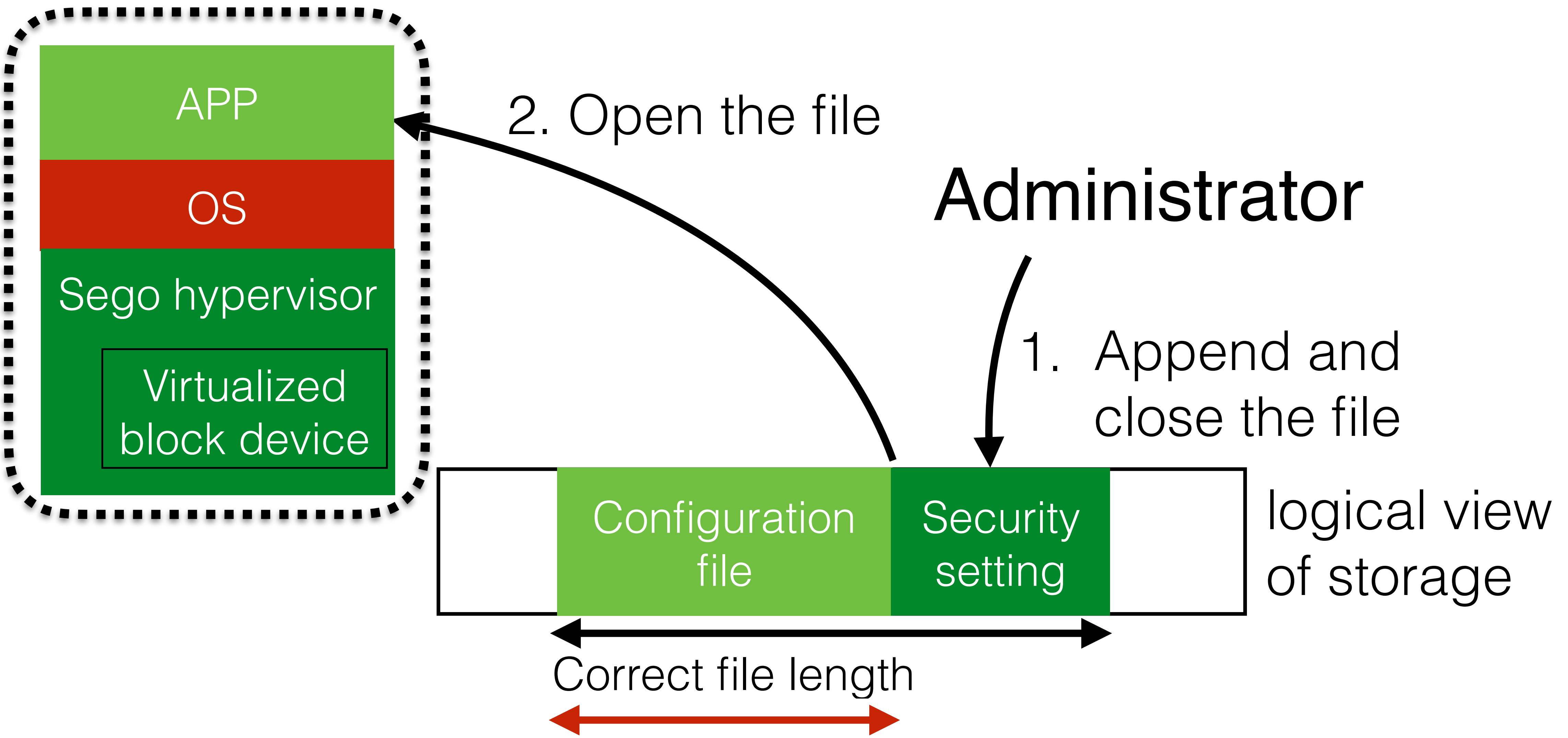
File length attack



File length attack

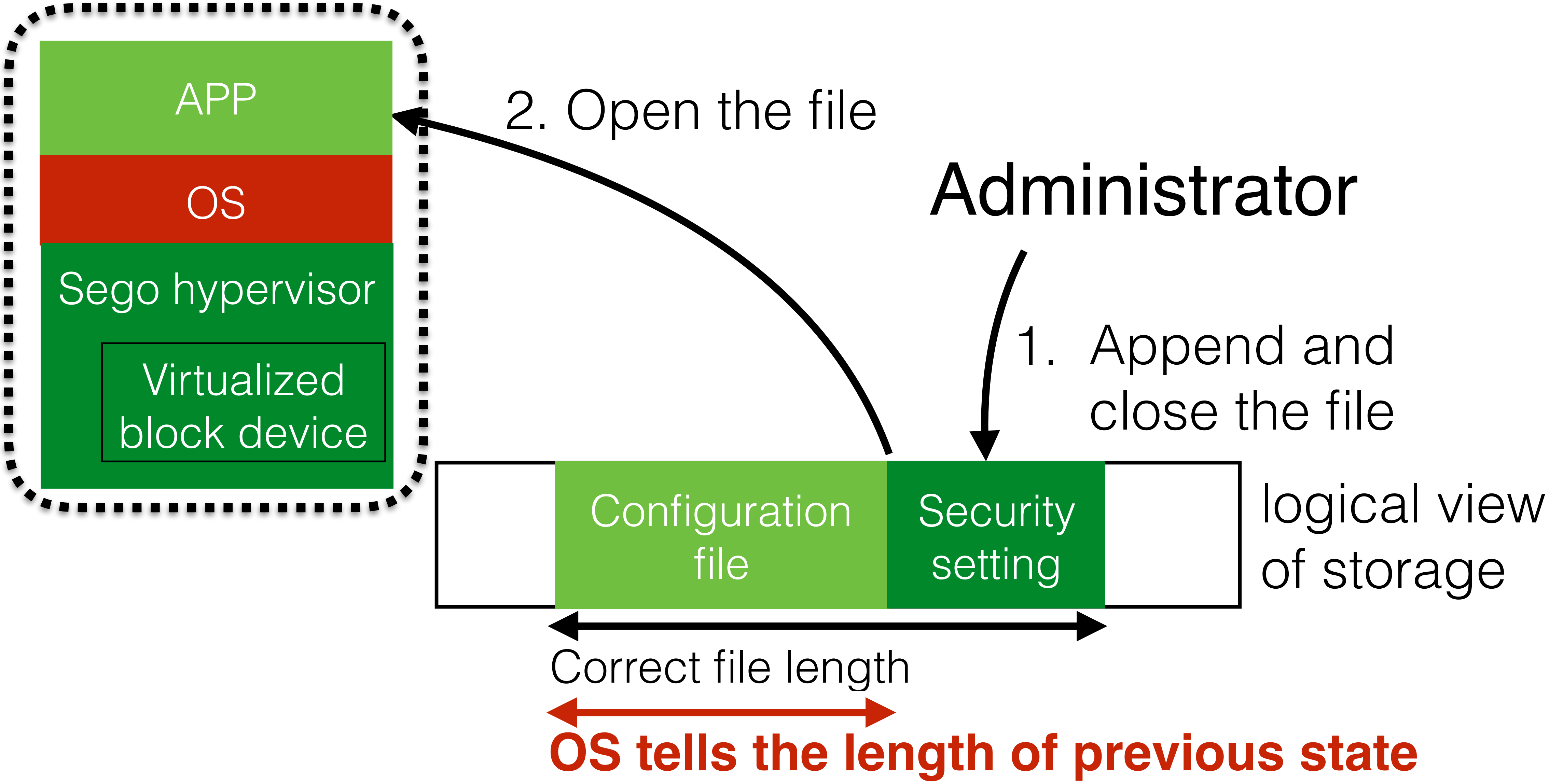


File length attack



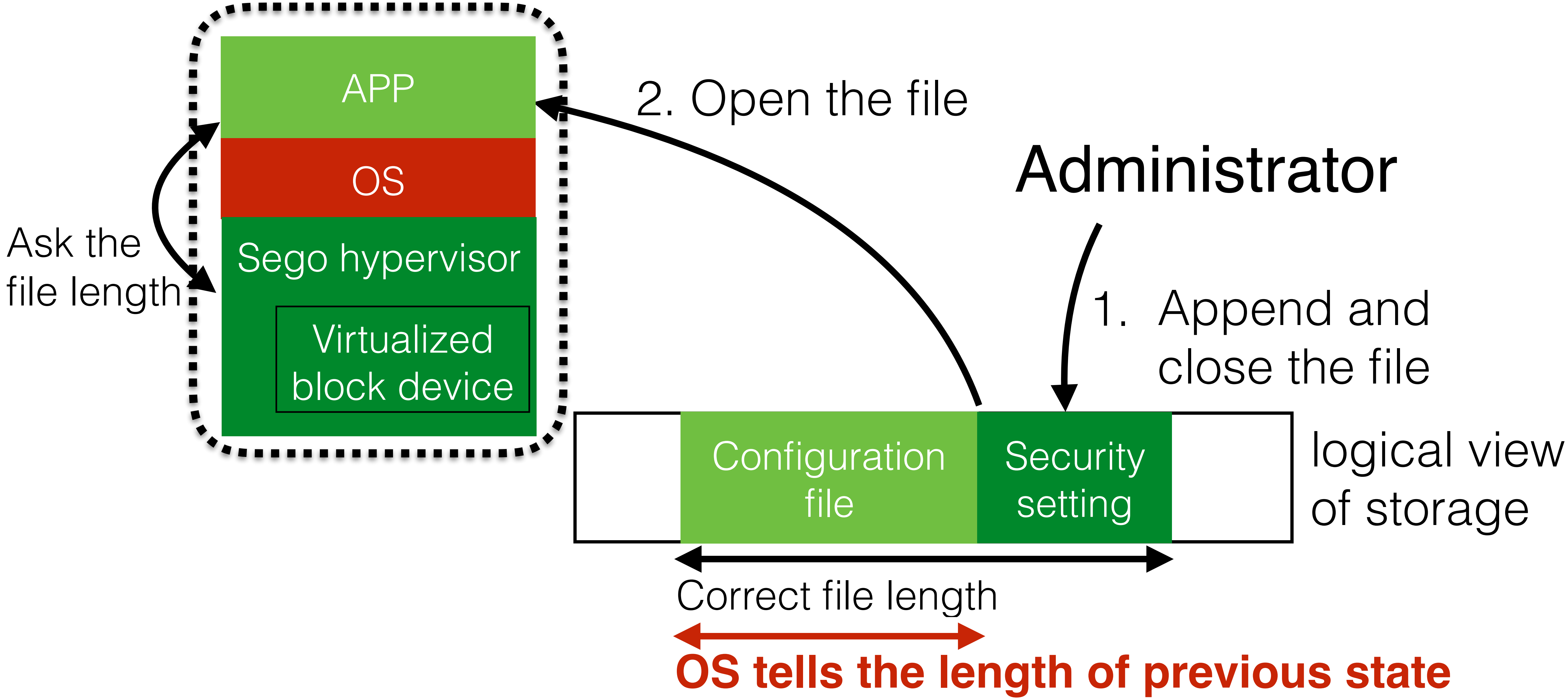
OS tells the length of previous state

File length attack



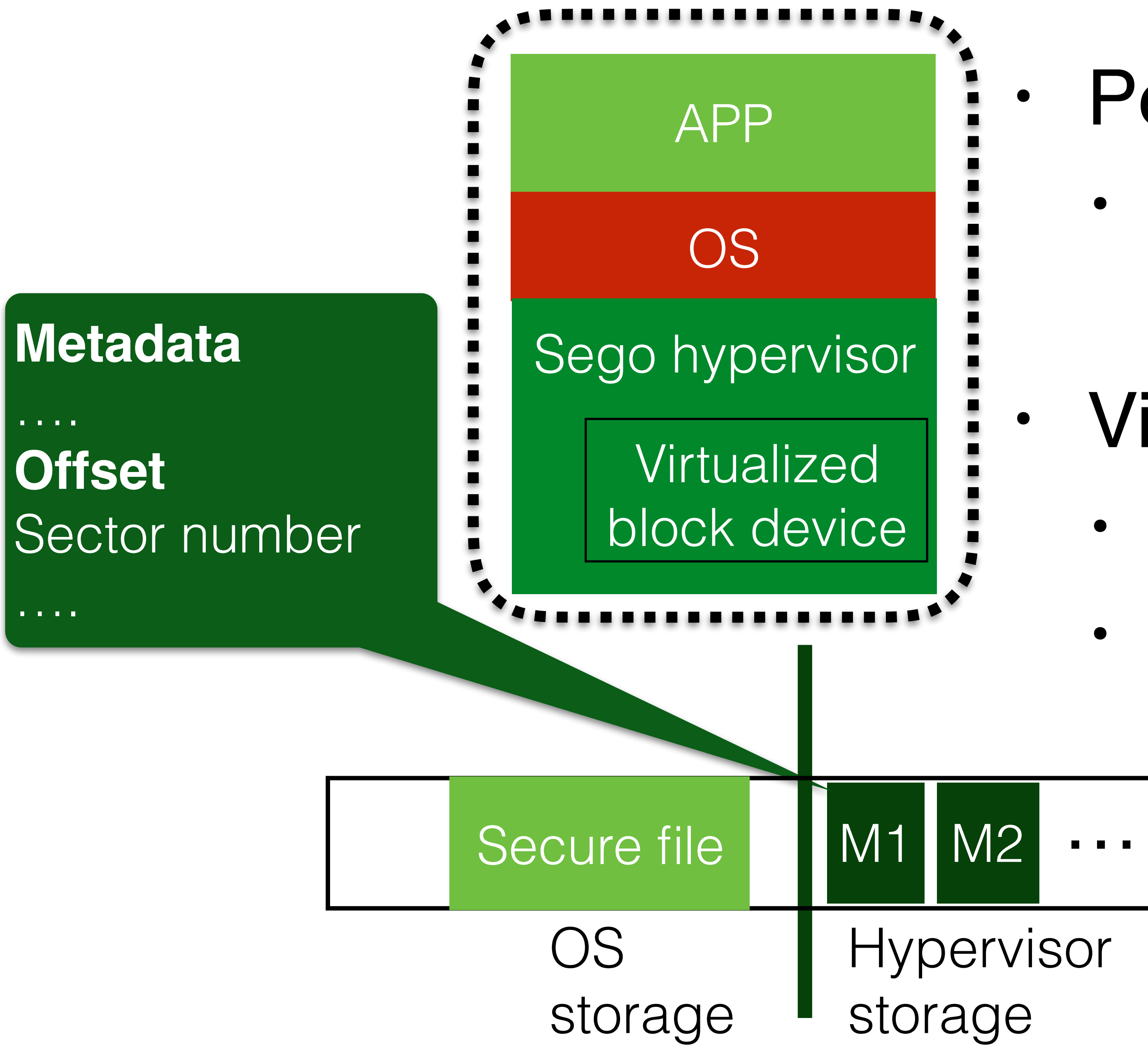
If the APP believes the OS length, OS can do the file length attack (undo the security setting)

File length attack



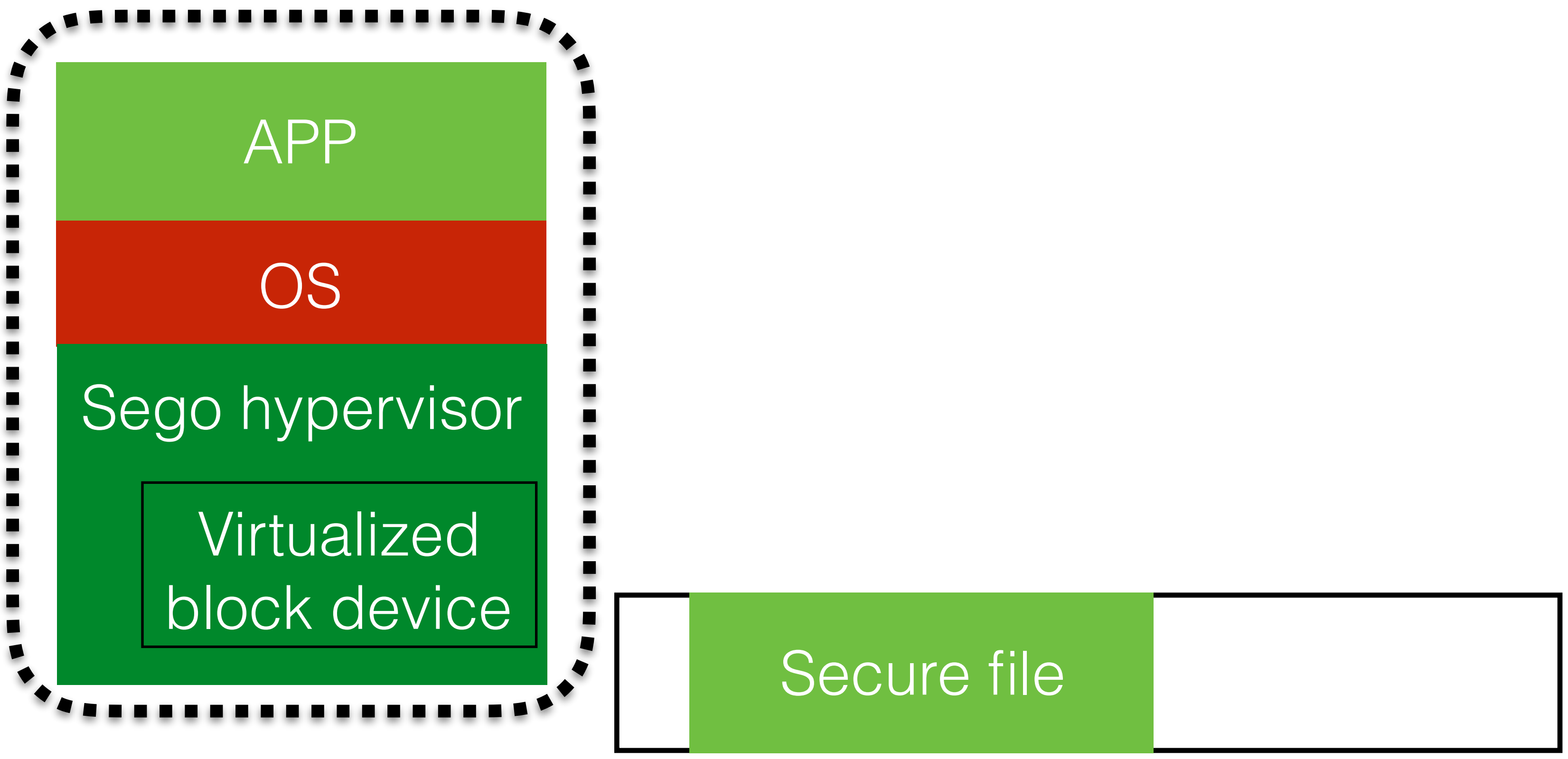
If the APP believes the OS length,
OS can do the file length attack
(undo the security setting)

Virtualized block device tracks file length with metadata

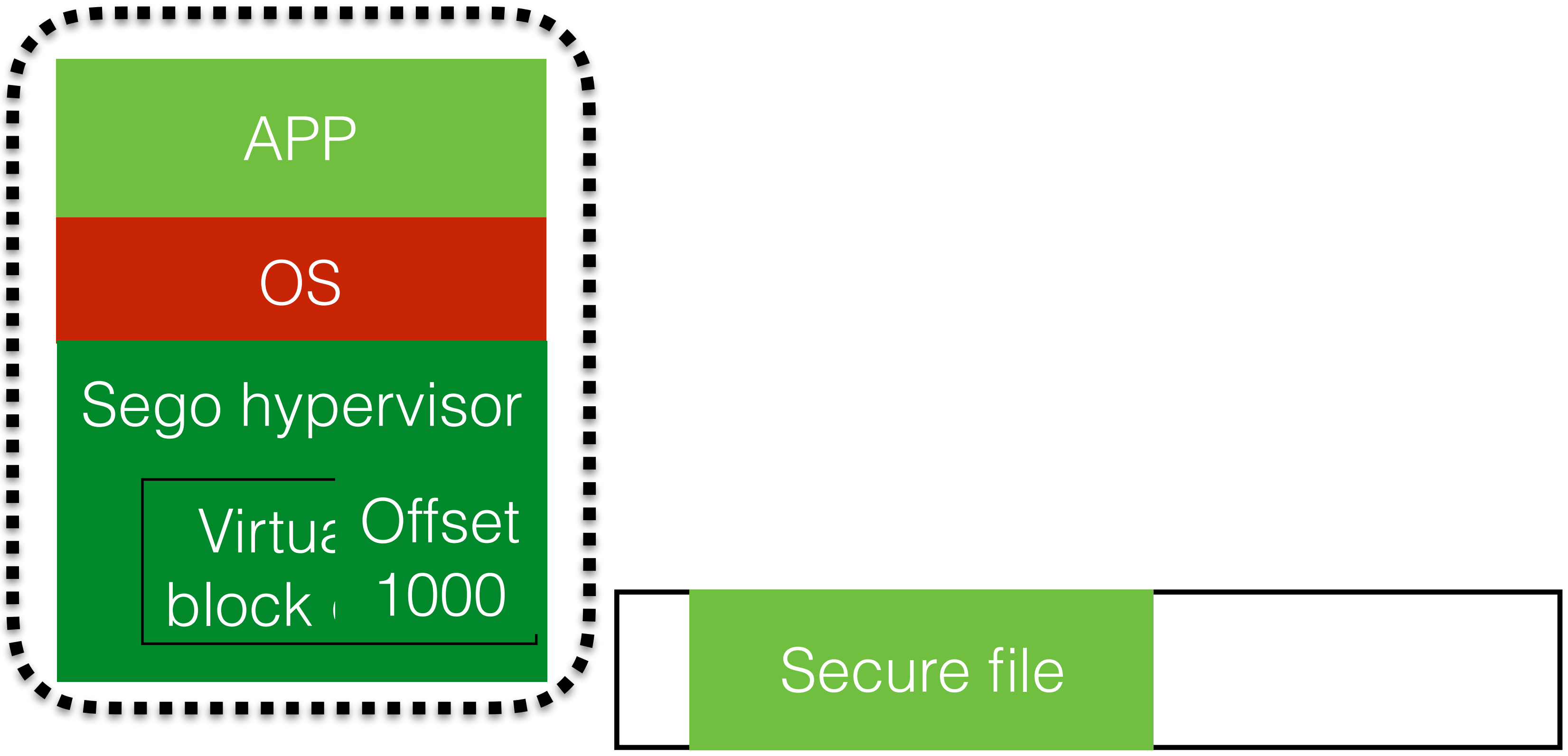


- Pervasive metadata model
  - Metadata is shared
- Virtualized block device
  - Tracks a maximum offset
  - Shares the file length with hypervisor

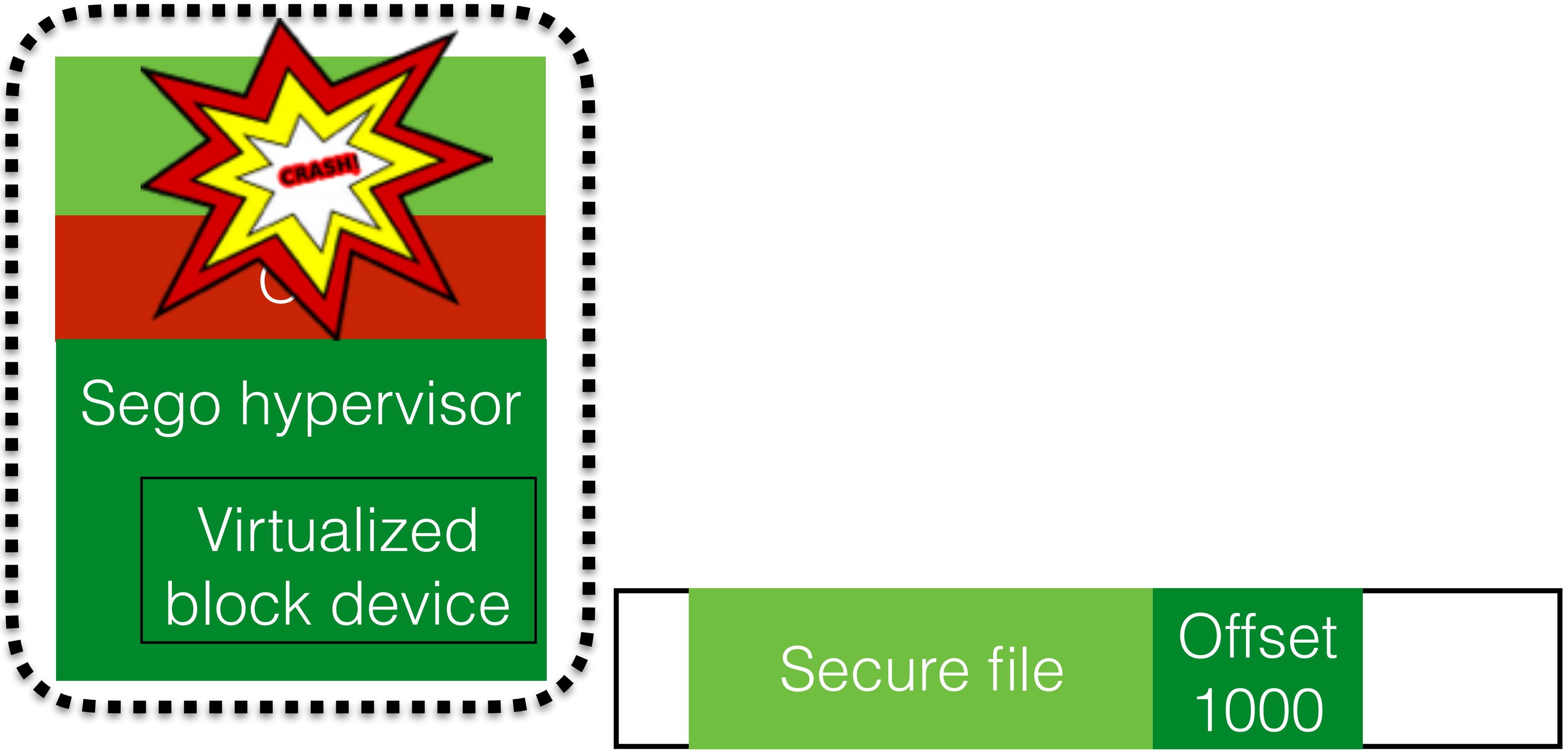
Append crash scenario



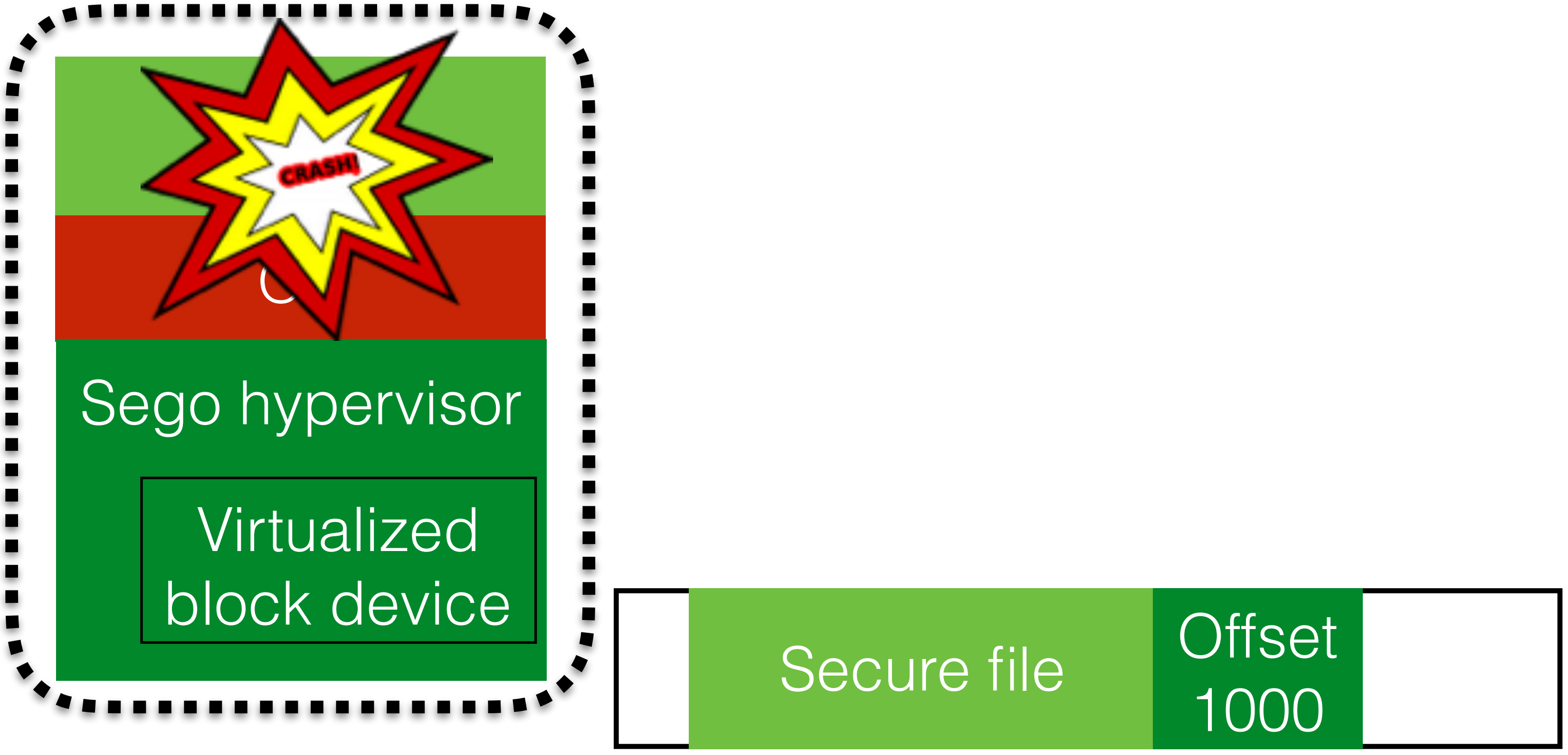
Append crash scenario



Append crash scenario

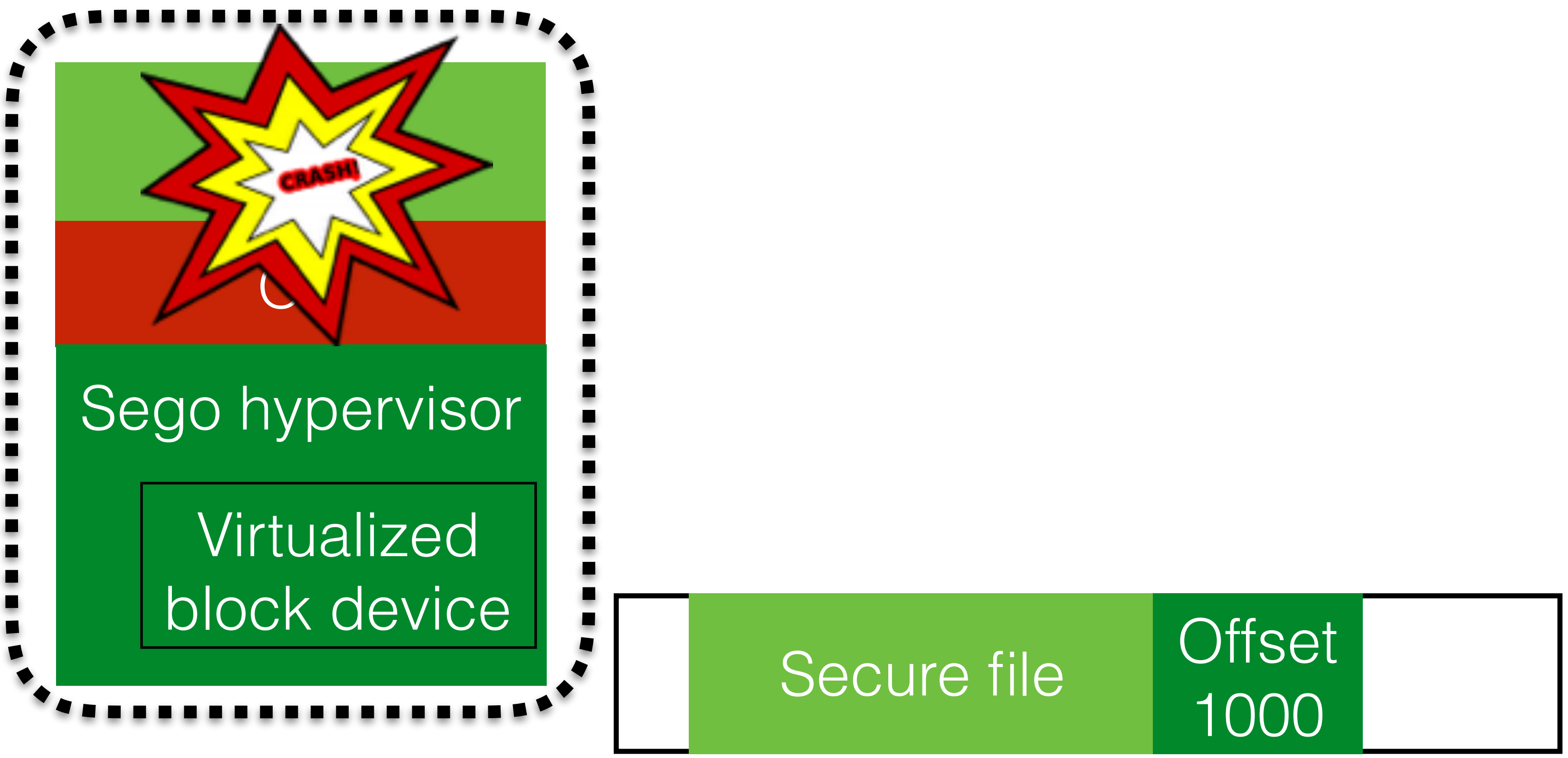


Append crash scenario



Write ordering by OS file system

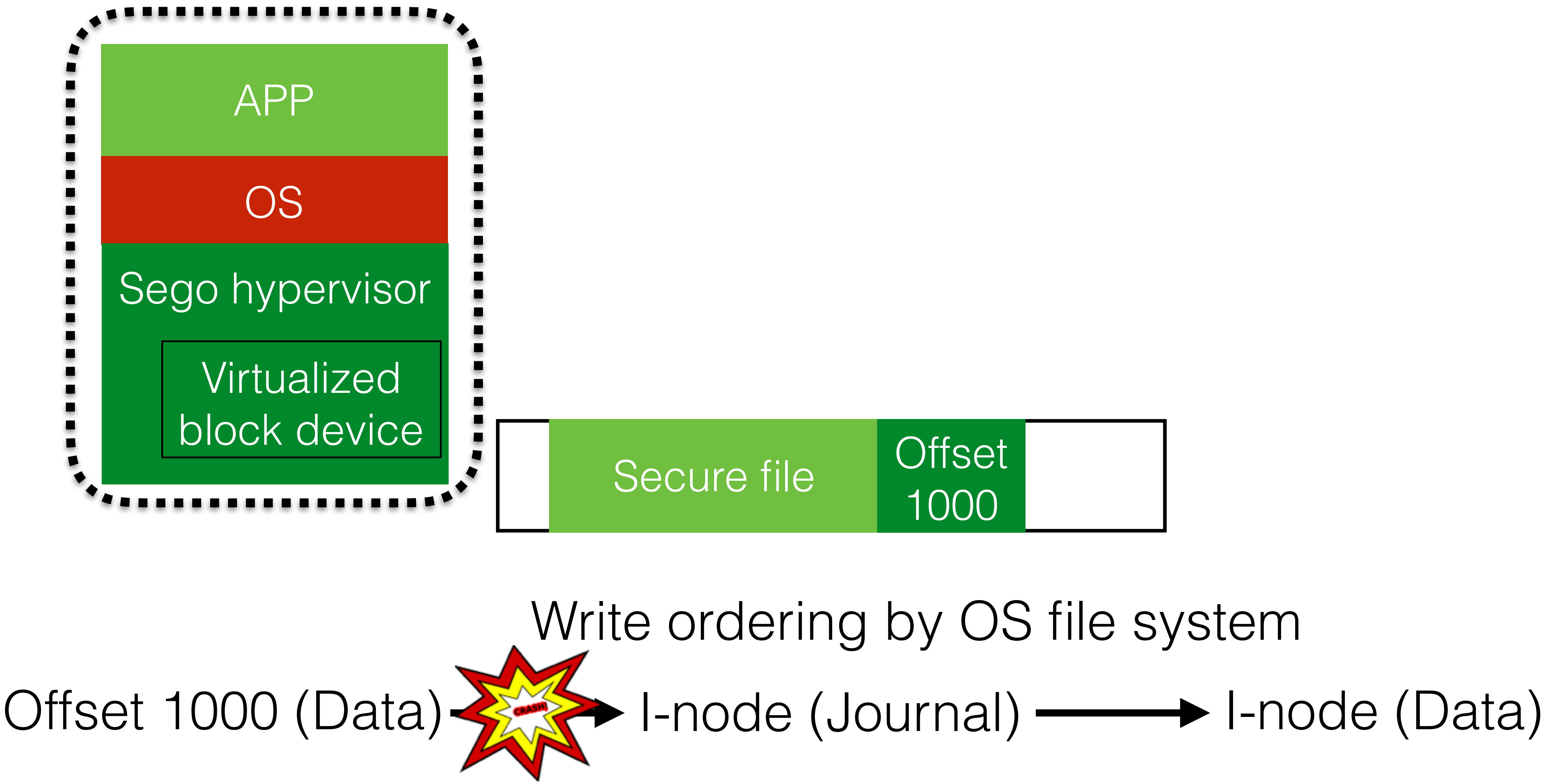
Append crash scenario



Write ordering by OS file system

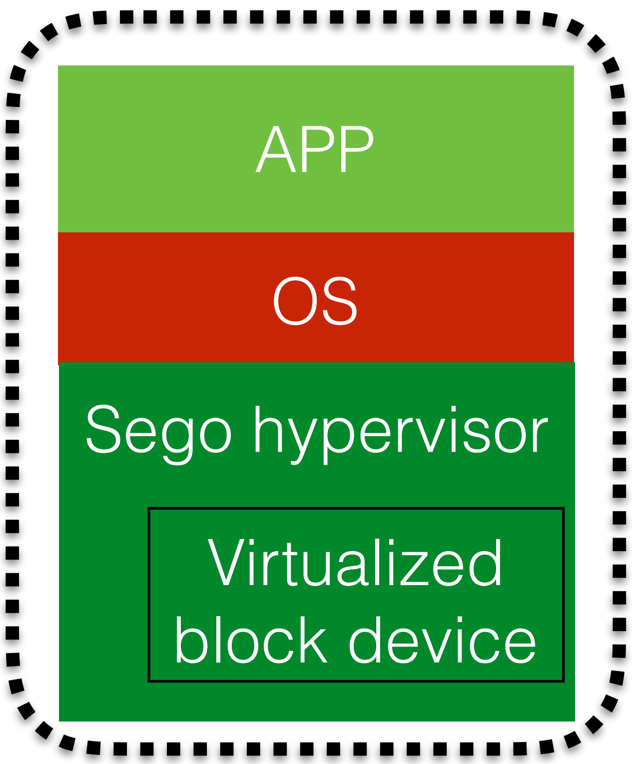
Offset 1000 (Data) → I-node (Journal) → I-node (Data)

Append crash scenario



Journaling filesystem discards the write during recovery

Append crash scenario



Offset 1000 **Discarded block**

Data is persisted

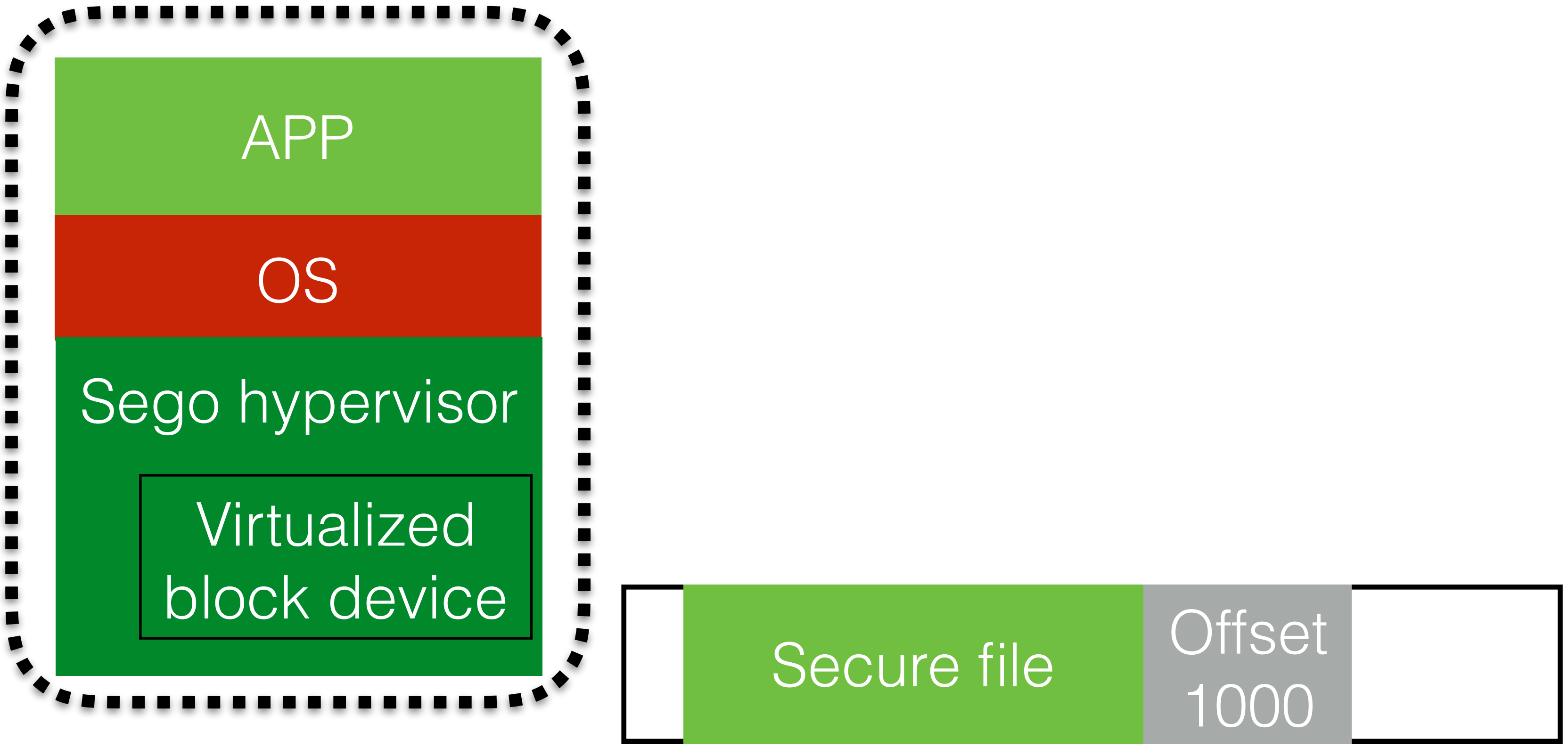
OS I-node has stale length

Write ordering by OS file system

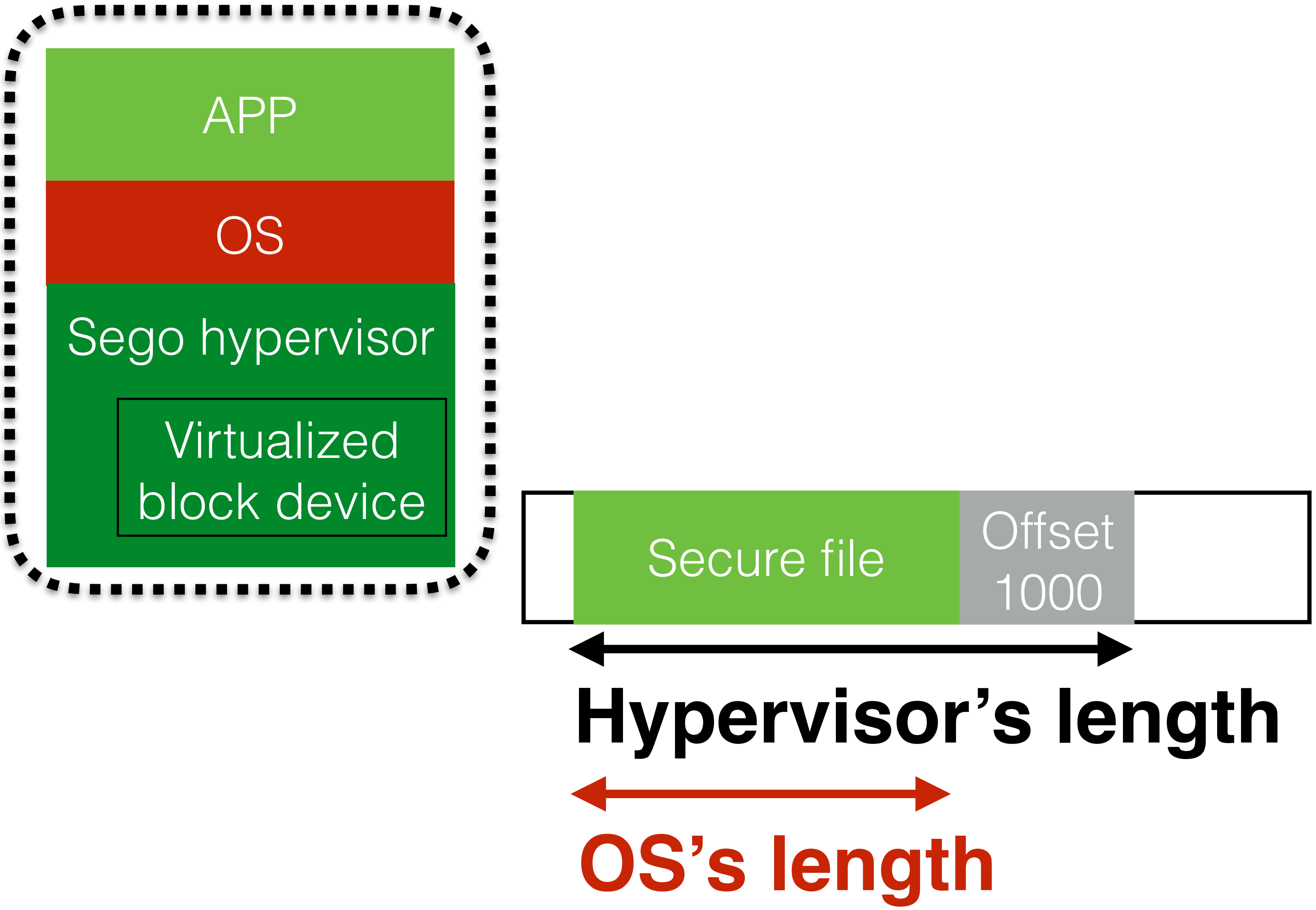
Offset 1000 (Data) → I-node (Journal) → I-node (Data)

Journaling filesystem discards the write during recovery

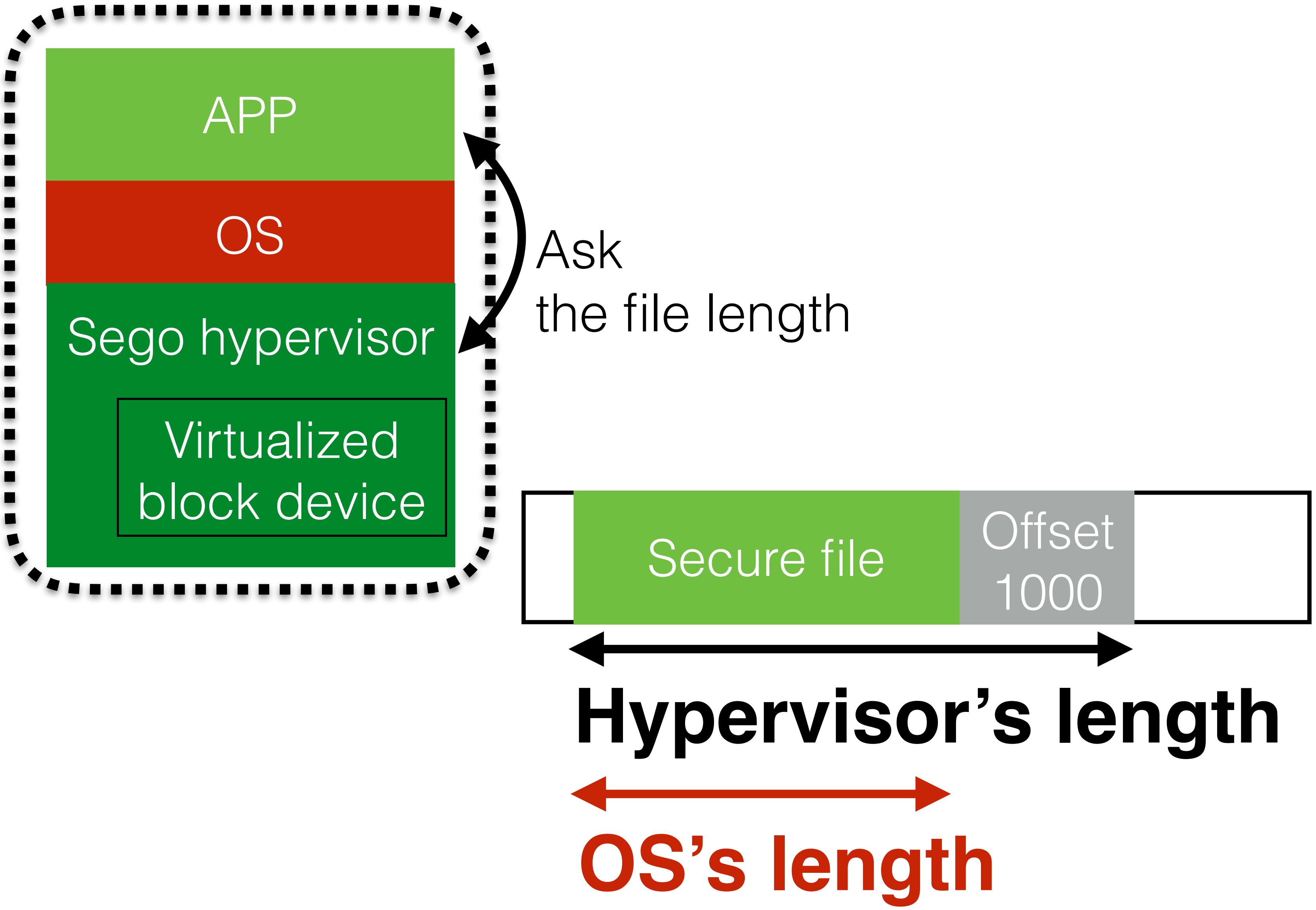
Journal file system creates inconsistency problem



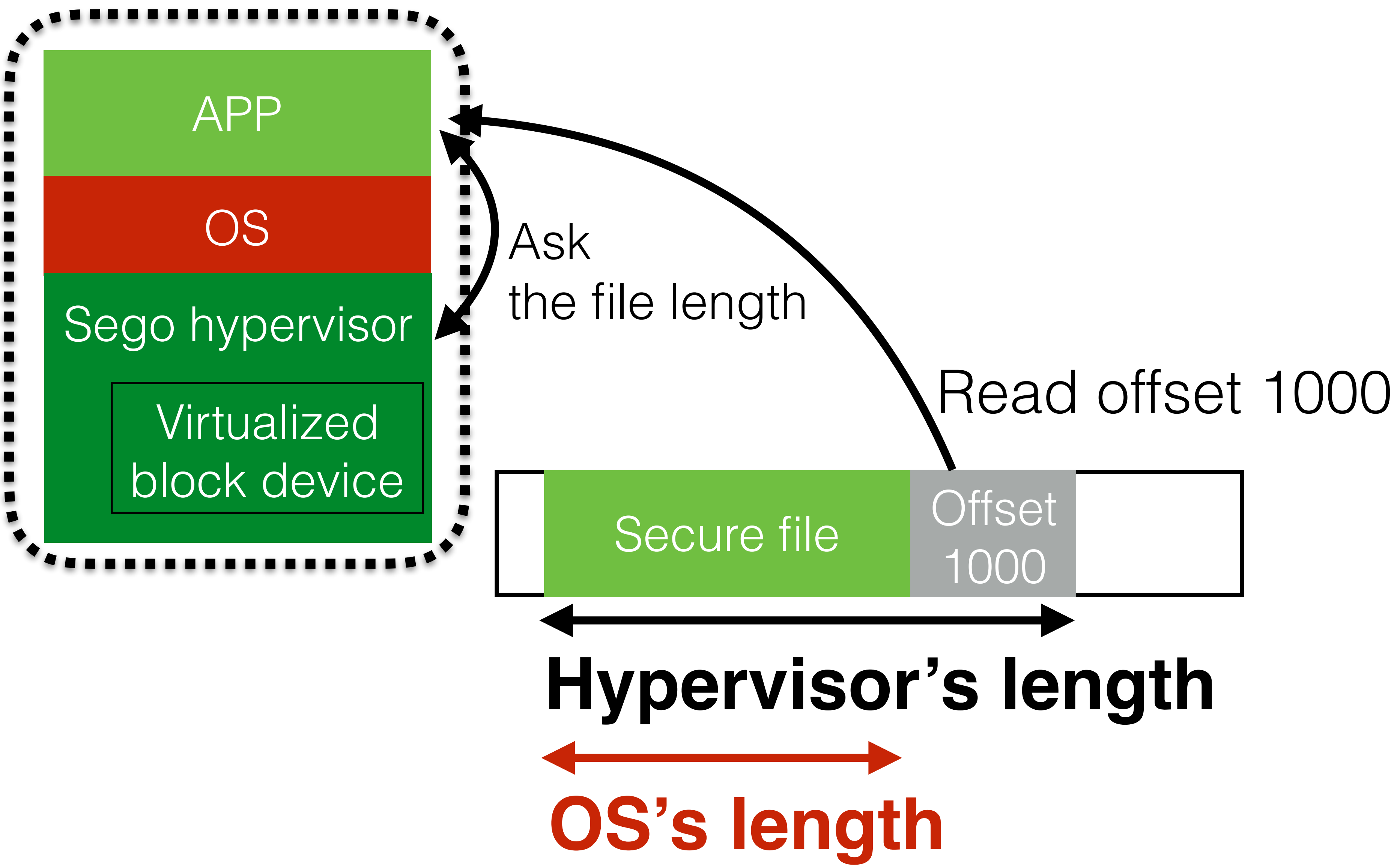
Journal file system creates inconsistency problem



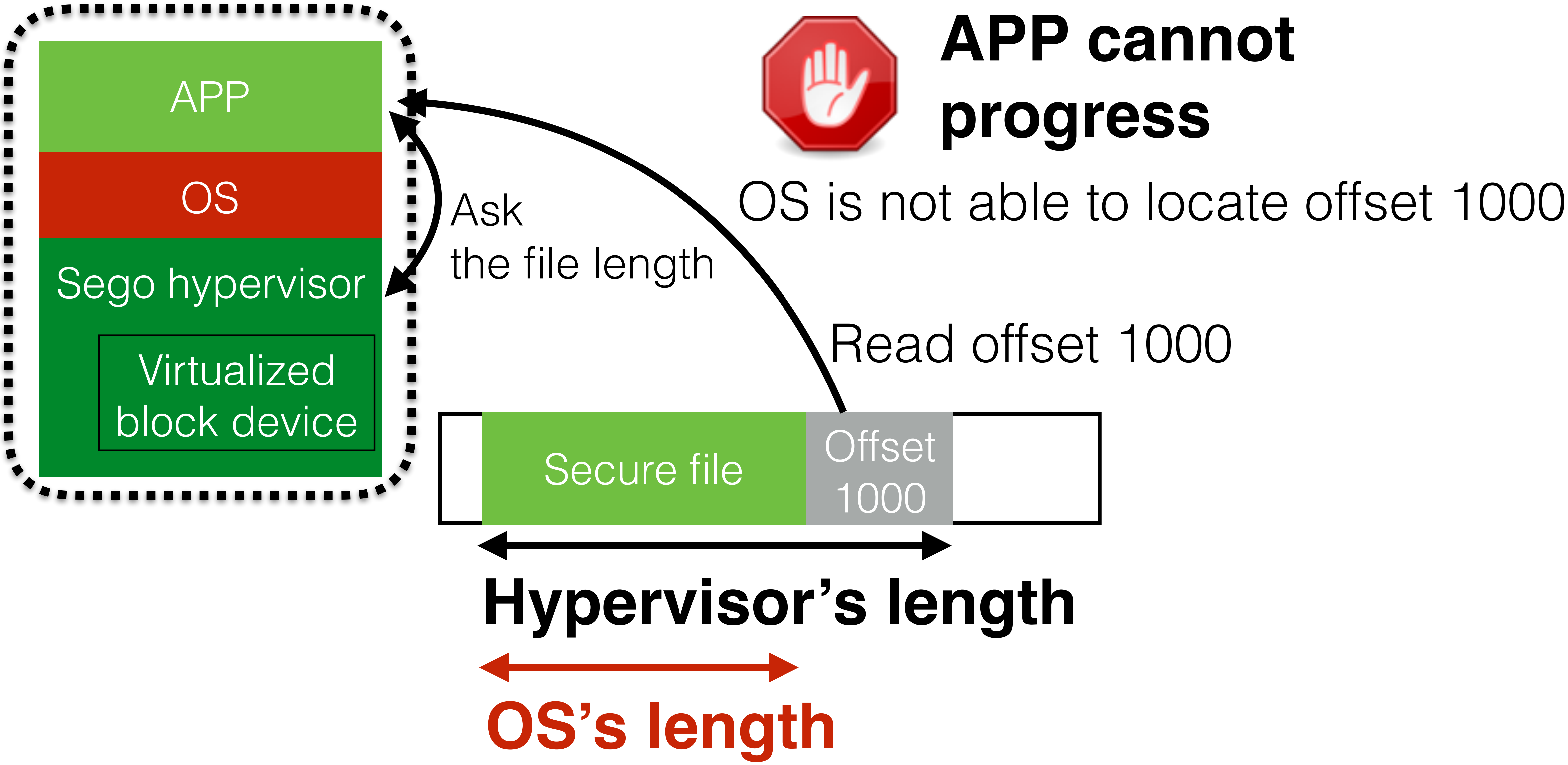
Journal file system creates inconsistency problem



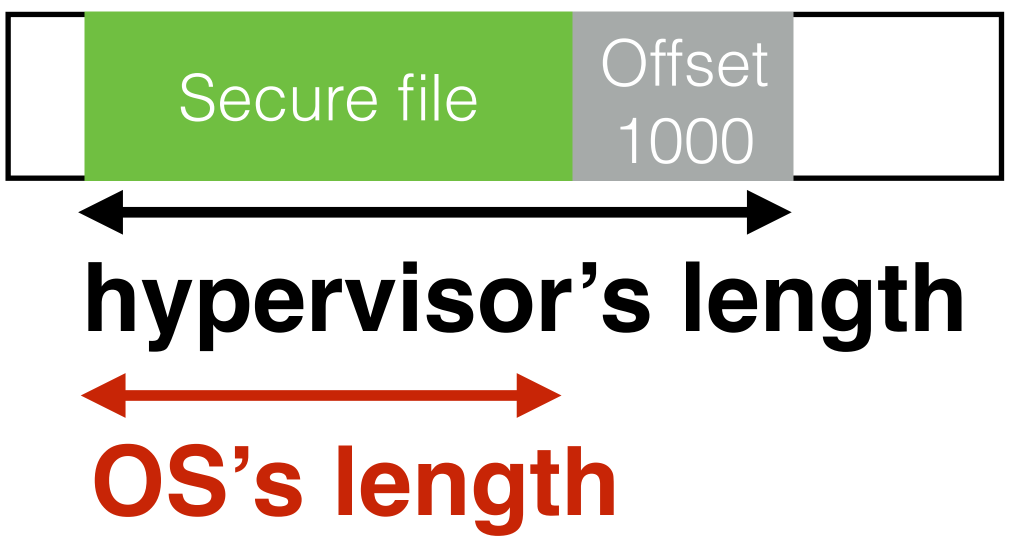
Journal file system creates inconsistency problem



Journal file system creates inconsistency problem



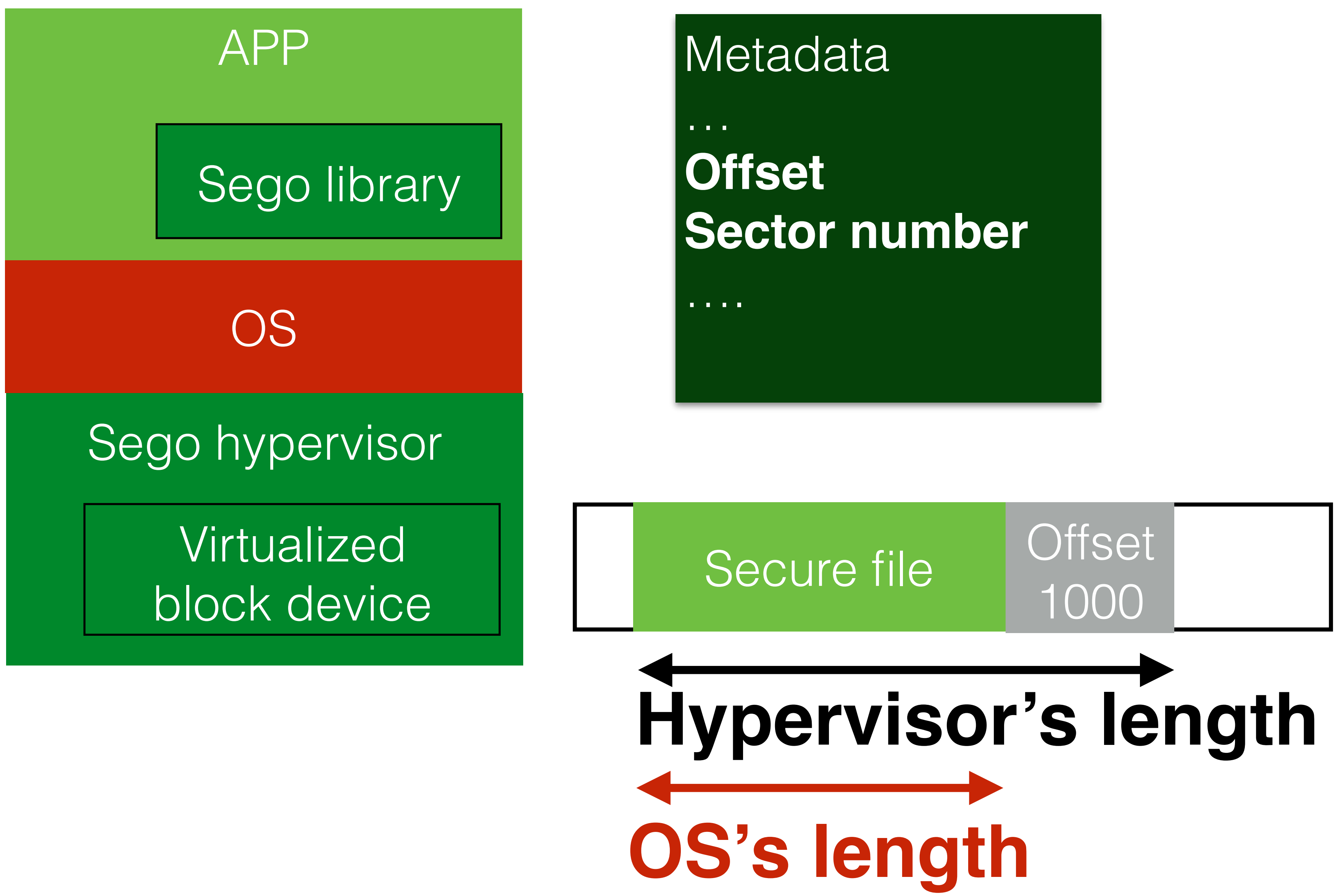
Sego cannot trust journal file system



- This OS recovery is legal
  - Hypervisor cannot trust it
  - Legal or malicious?
- If APP believes OS's length
  - OS can use this crash for the file length attack
- If APP believes hypervisor's length
  - APP cannot progress in legal recovery case

Sego recovers secure file with metadata

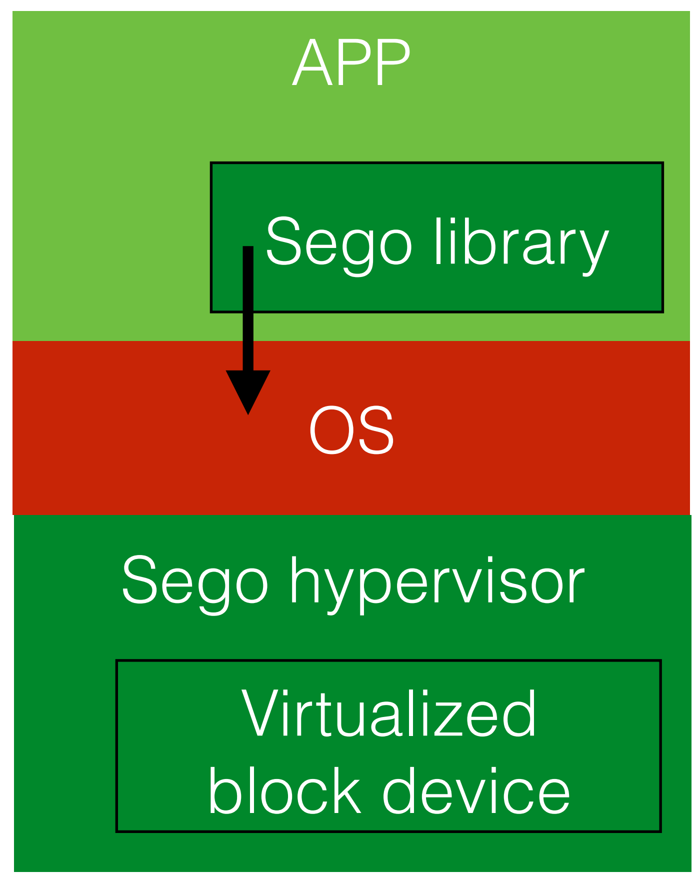
Recovery procedure



Sego recovers secure file with metadata

Recovery procedure

APP	Opens the file
LIB	Get OS length



Metadata
…
Offset
Sector number
….

Sego recovers secure file with metadata

Recovery procedure

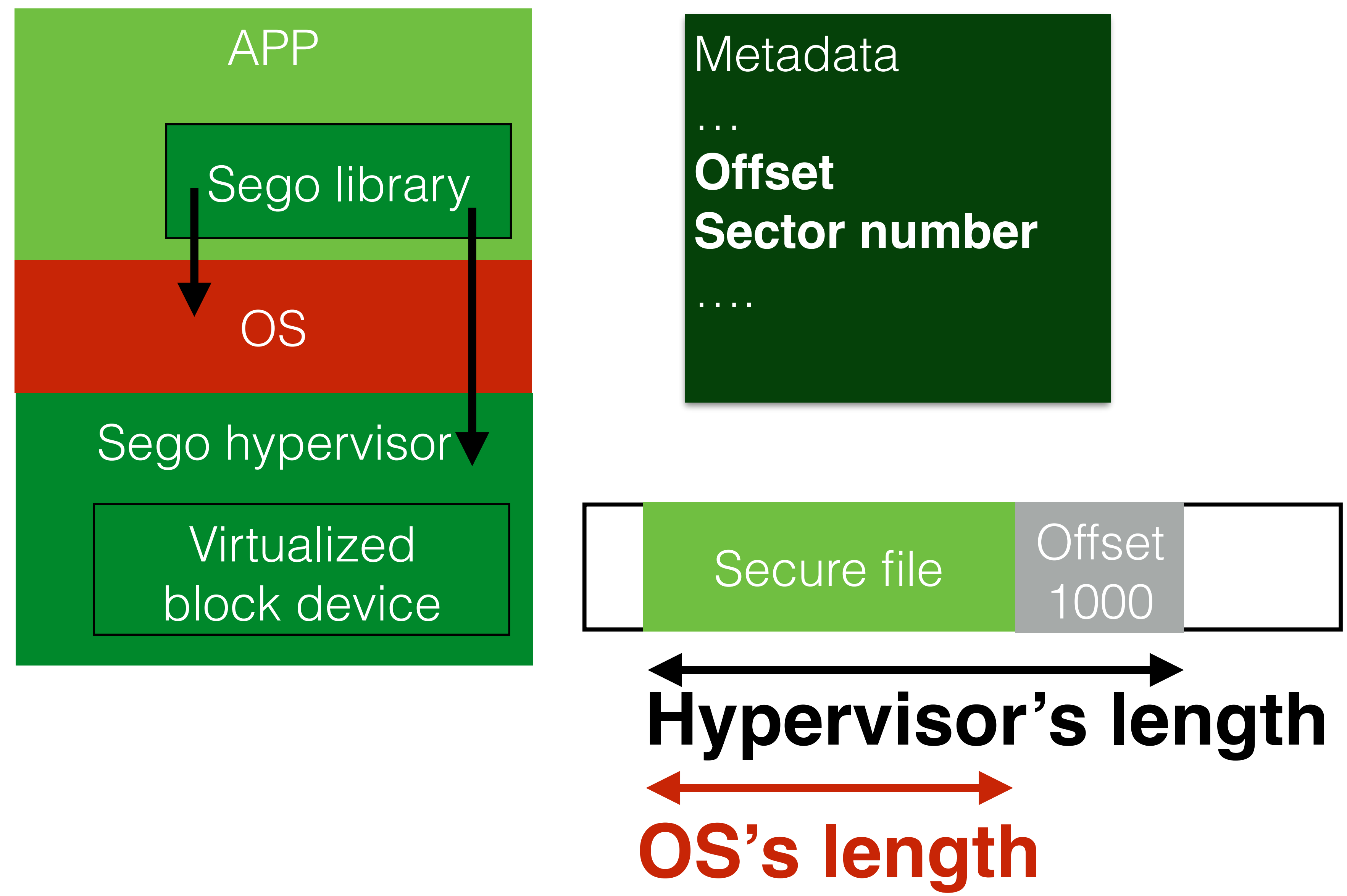
APP Opens the file

LIB Get OS length

↓

LIB Give OS length to Sego hypervisor

↓



Sego recovers secure file with metadata

Recovery procedure

APP Opens the file

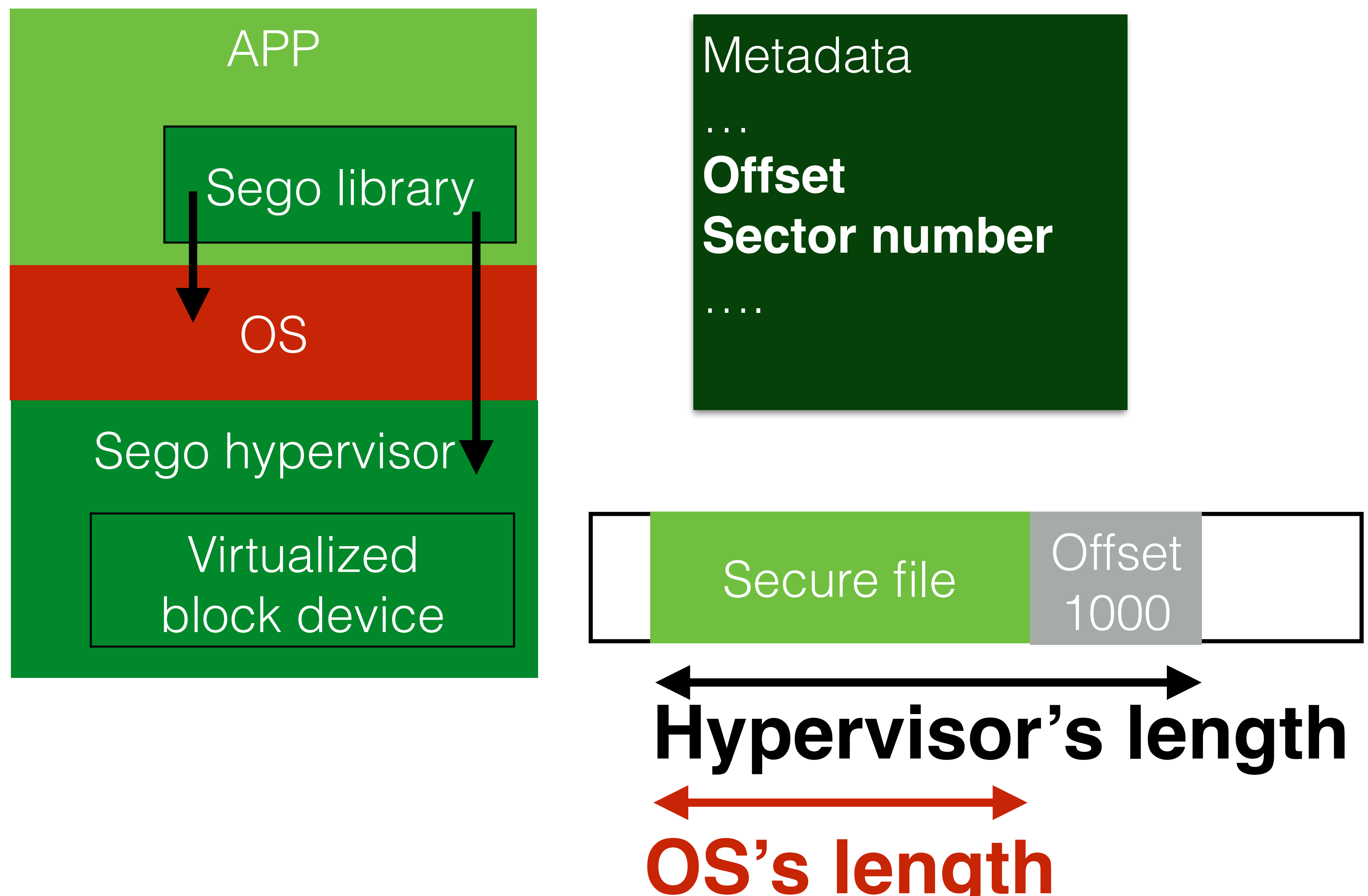
LIB Get OS length

↓

LIB Give OS length to Sego hypervisor

Length mismatch

↓



Sego recovers secure file with metadata

Recovery procedure

APP Opens the file

LIB Get OS length

↓

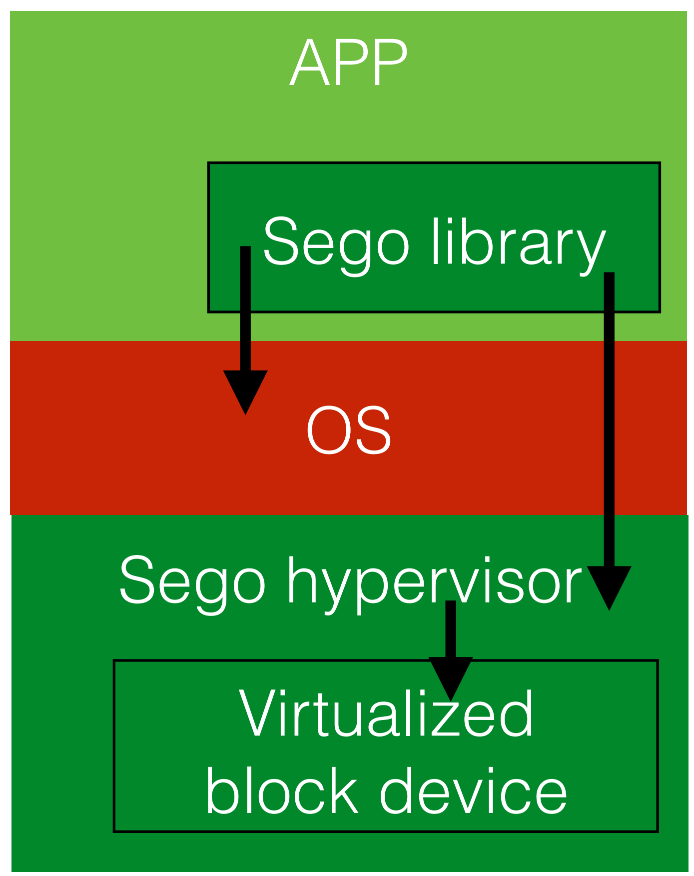
LIB Give OS length to Sego hypervisor

Length mismatch

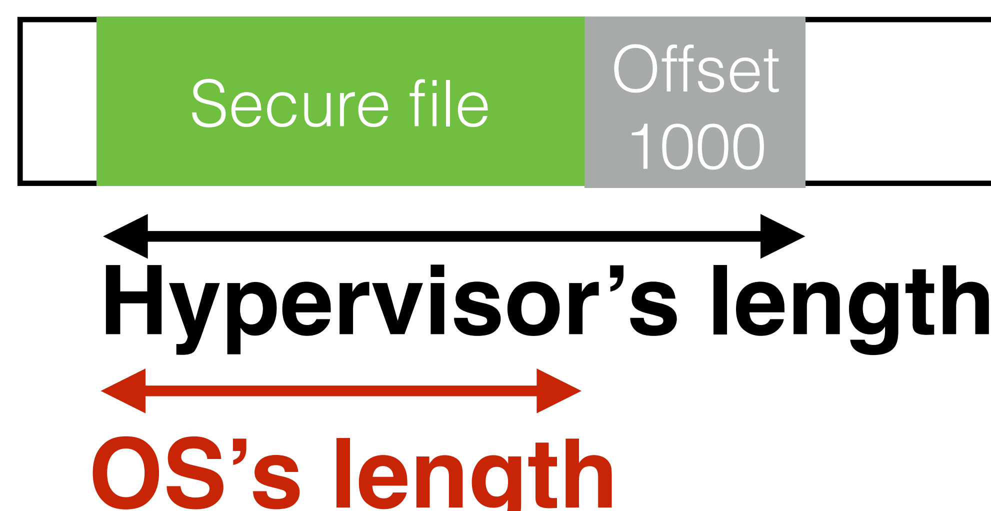
↓

HYP Requests recovery of offset 1000

↓



Metadata
…
Offset
Sector number
….



Sego recovers secure file with metadata

Recovery procedure

APP Opens the file

LIB Get OS length

↓

LIB Give OS length to Sego hypervisor

Length mismatch

↓

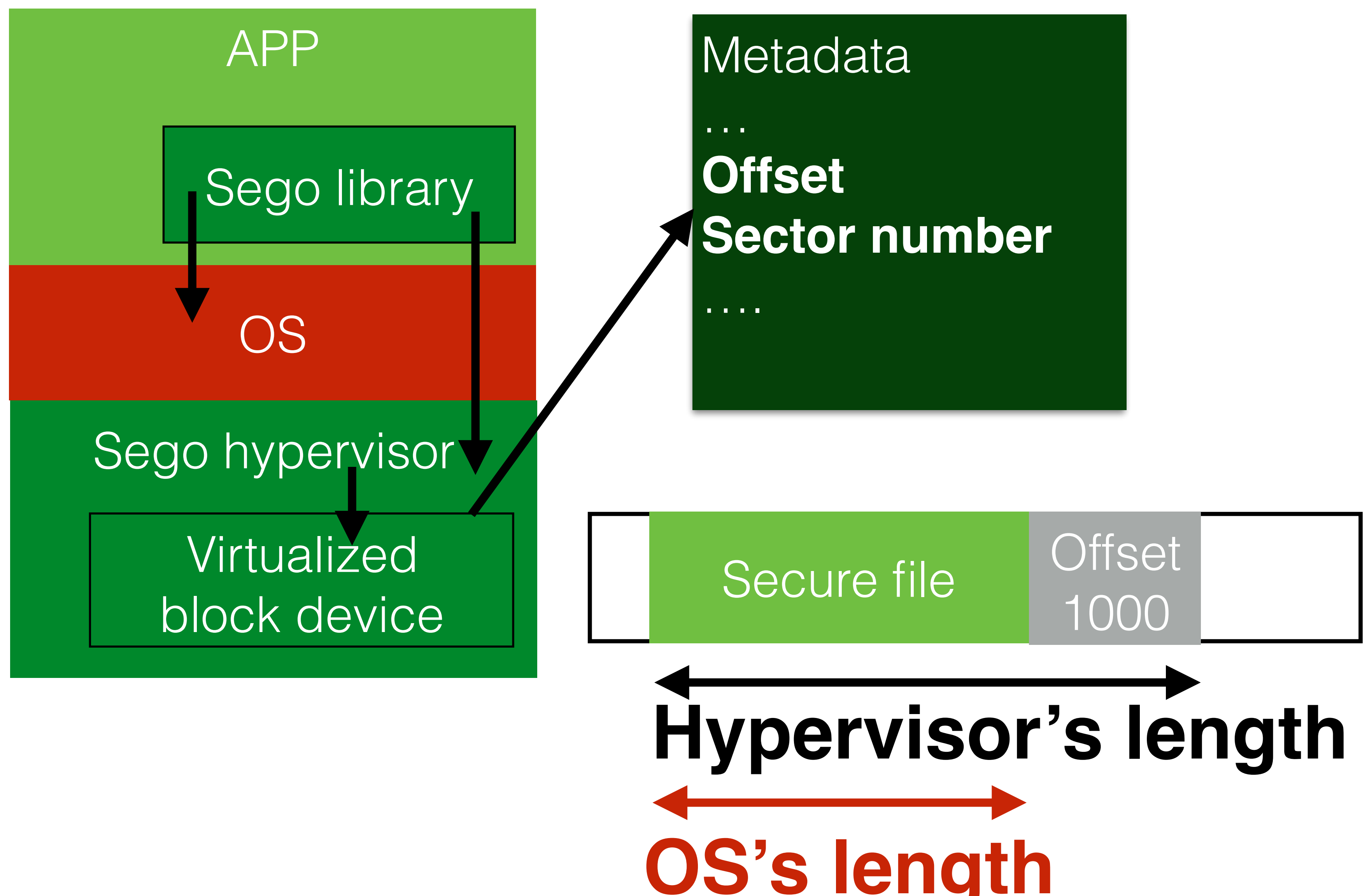
HYP Requests recovery of offset 1000

↓

Virtualized block device

Read the data by looking up metadata

↓



Sego recovers secure file with metadata

Recovery procedure

APP Opens the file
LIB Get OS length

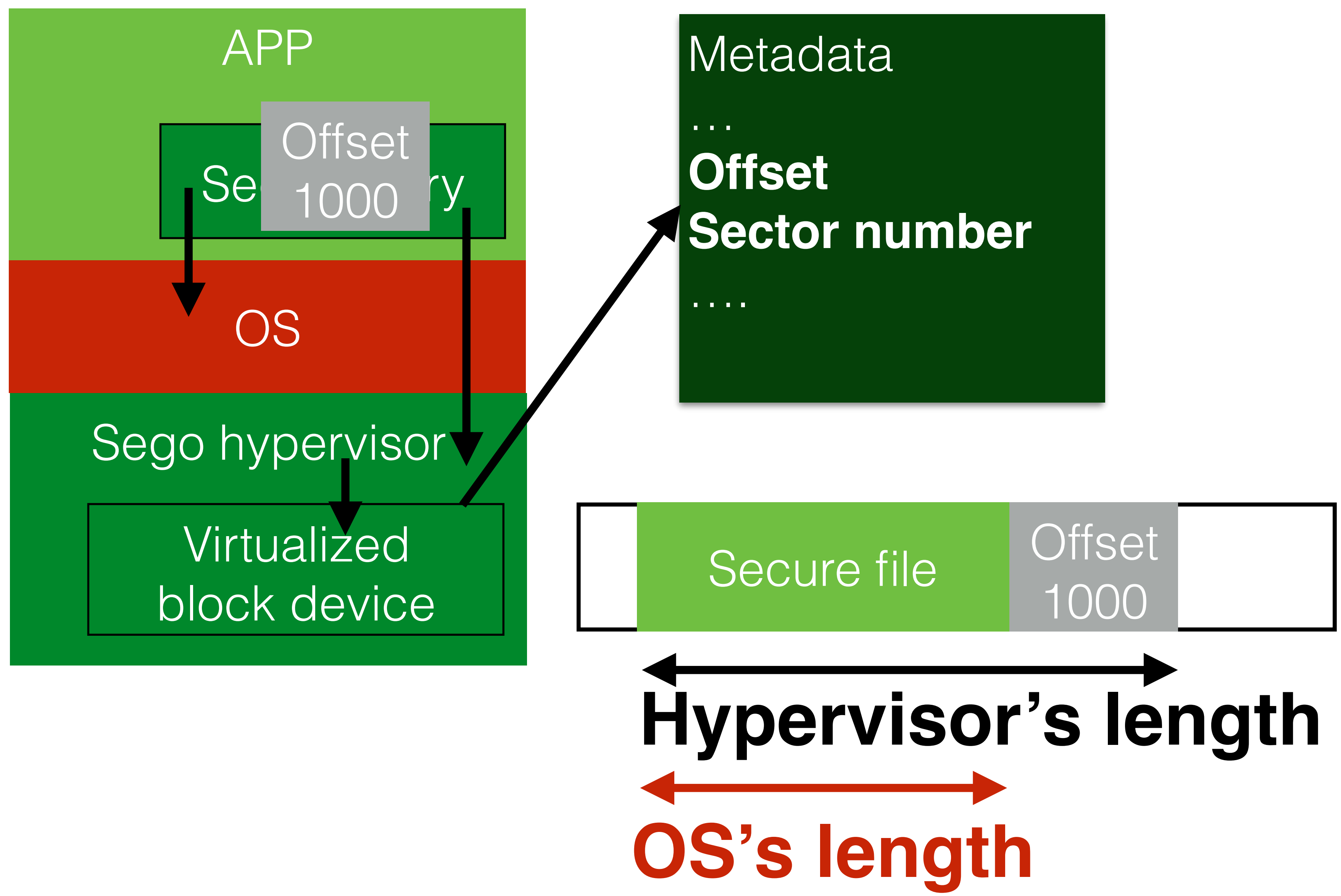
LIB Give OS length to Sego hypervisor

Length mismatch

HYP Requests recovery of offset 1000

Virtualized block device
Read the data by looking up metadata

Copies the offset 1000 to the library



Sego recovers secure file with metadata

Recovery procedure

APP Opens the file

LIB Get OS length

↓

LIB Give OS length to Sego hypervisor

Length mismatch

↓

HYP Requests recovery of offset 1000

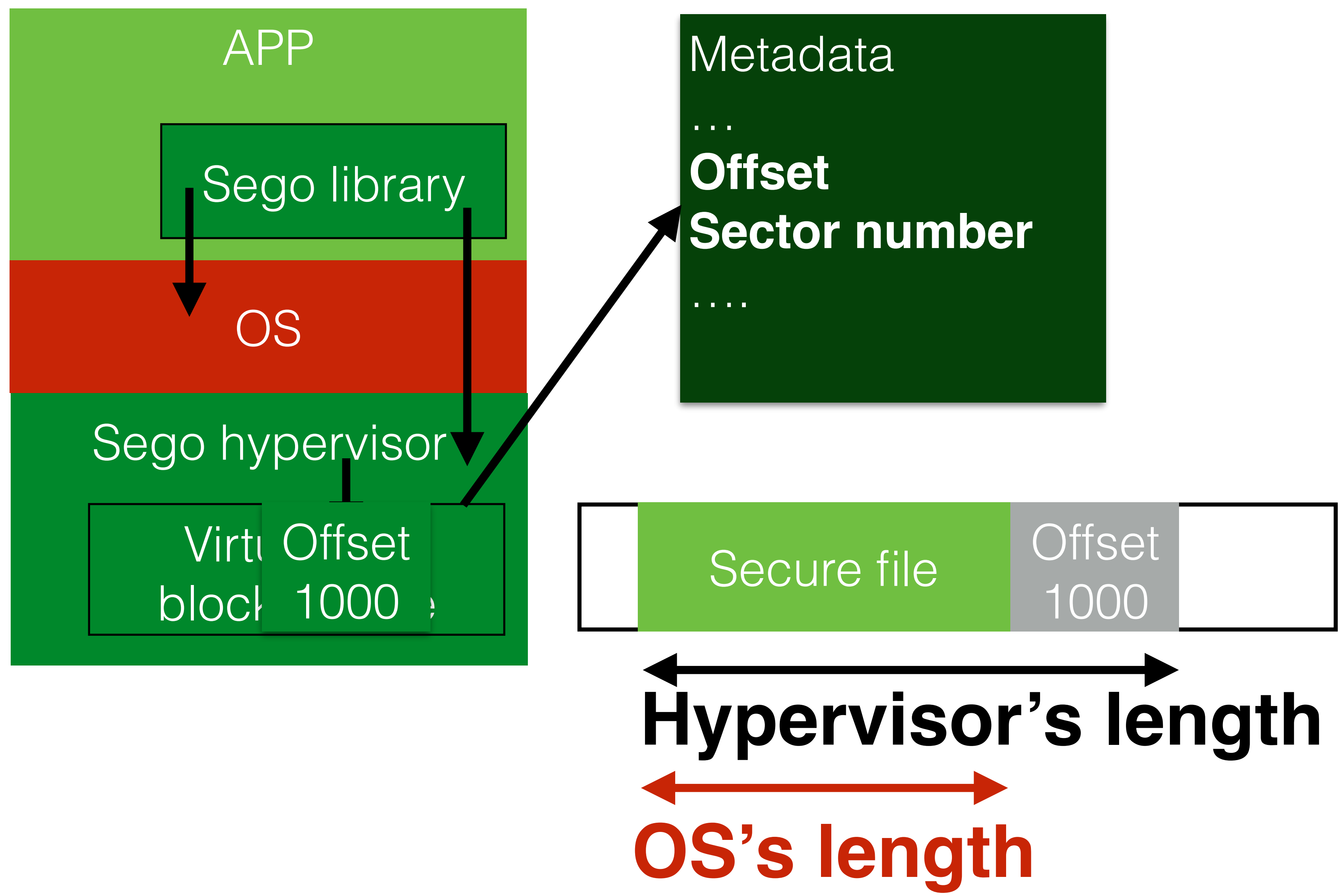
↓

Virtualized block device

Read the data by looking up metadata

↓

Copies the offset 1000 to the library



Sego recovers secure file with metadata

Recovery procedure

APP Opens the file

LIB Get OS length

↓

LIB Give OS length to Sego hypervisor

Length mismatch

↓

HYP Requests recovery of offset 1000

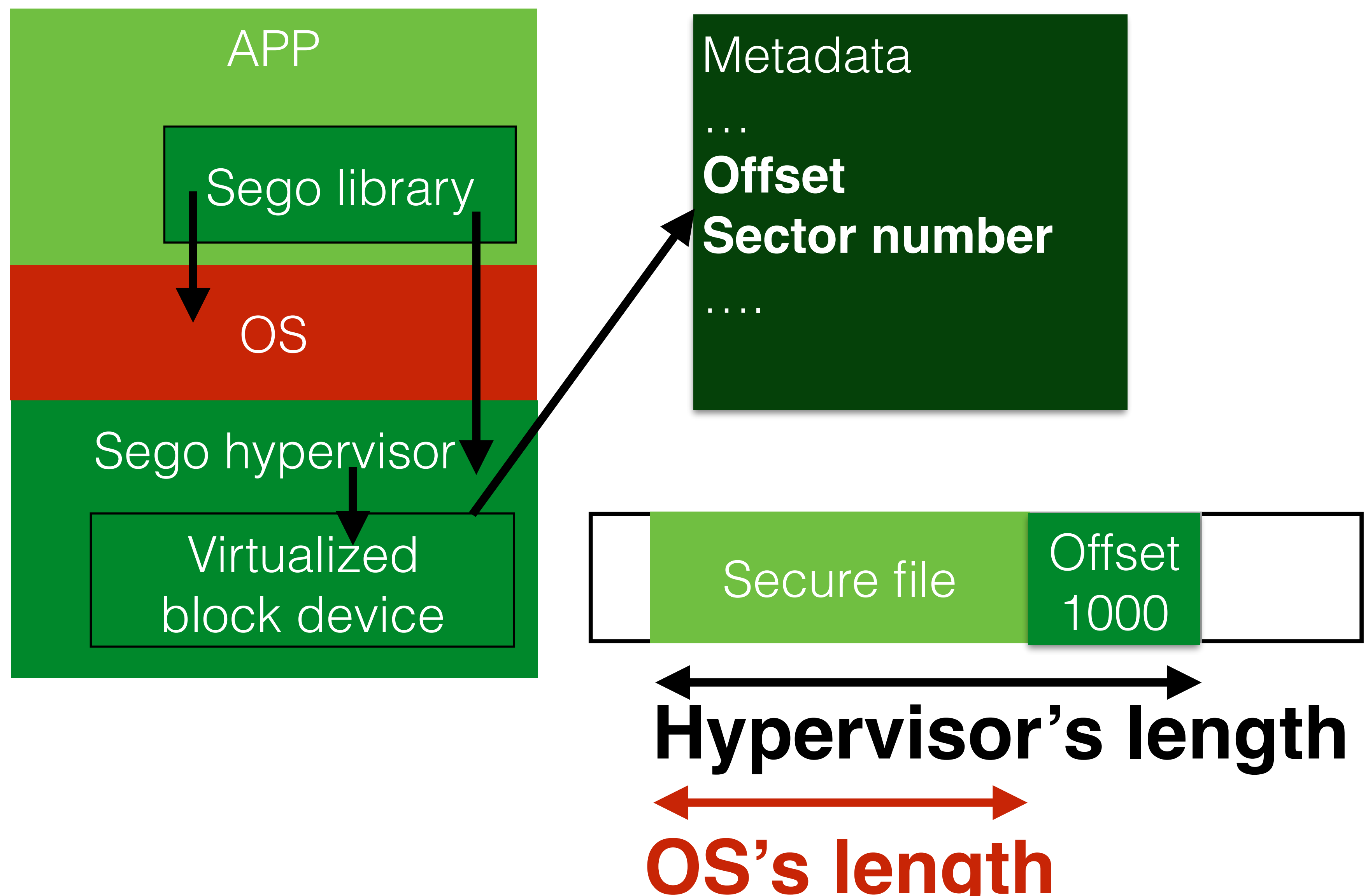
↓

Virtualized block device

Read the data by looking up metadata

↓

Copies the offset 1000 to the library



Sego recovers secure file with metadata

Recovery procedure

APP Opens the file

LIB Get OS length

↓

LIB Give OS length to Sego hypervisor

Length mismatch

↓

HYP Requests recovery of offset 1000

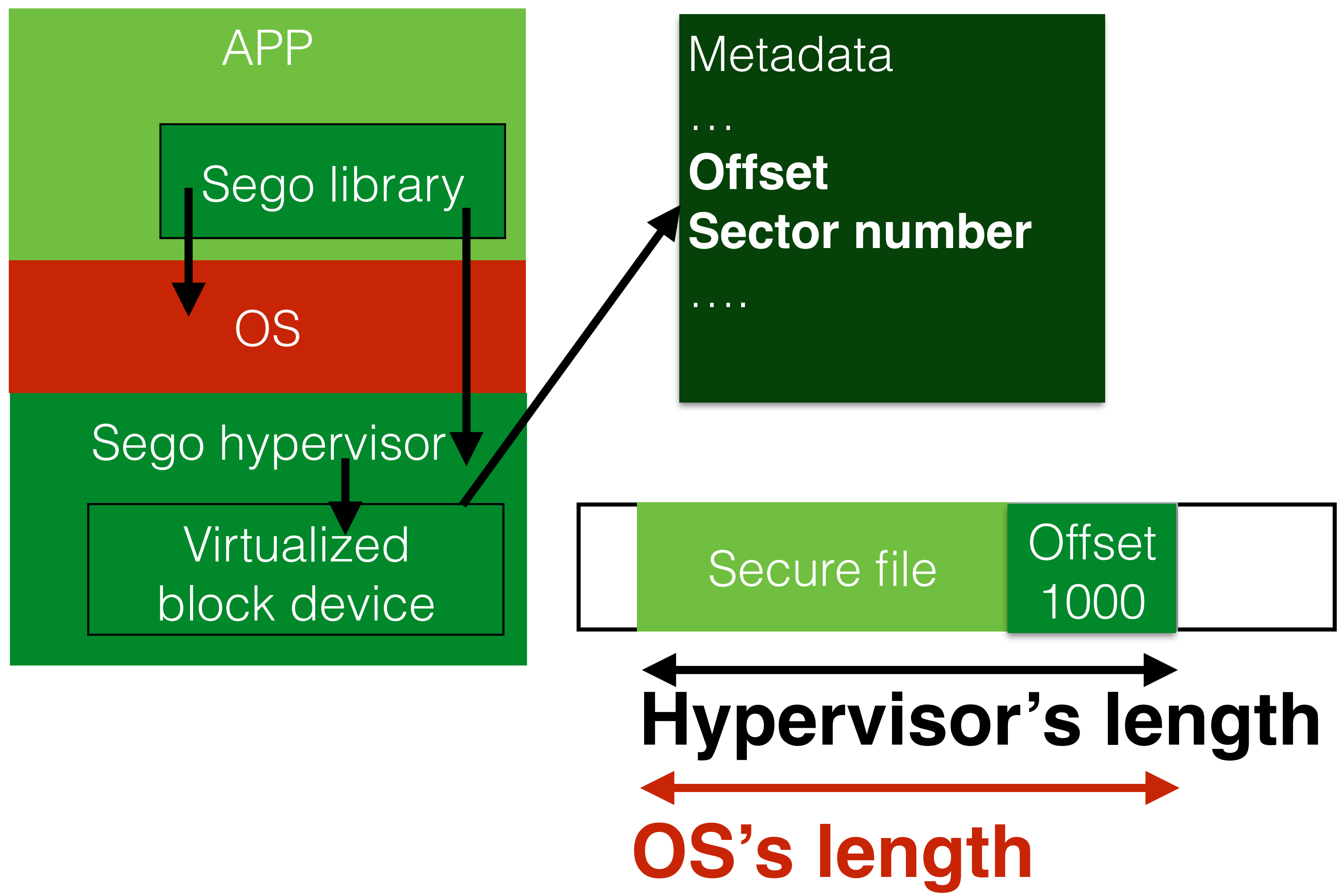
↓

Virtualized block device

Read the data by looking up metadata

↓

Copies the offset 1000 to the library



Other crash cases

Recovery target	Inconsistency	Detection
File creation	File is created in hypervisor but not in OS	When the APP opens the file
File length	File length of hypervisor and OS is different	When OS reboots from crash
Data recovery	Hypervisor loses blocks because OS discards them	When the APP opens the file
Block commit (hypervisor crash)	Block write might not be committed in virtual block	Hypervisor runs FSCK
Crash while recovery	One of the above	Hypervisor runs FSCK

Fault injection

- Fault injector
  - Modify previous framework for modern OS
    - Nooks (Swift et al., SOSP 2003)
    - Rio file cache (Chen et al., ASPLOS 1996)
  - Fault distribution is based on real-world fault study
    - An empirical study of operating system error (SOSP 2001)
    - Faults in linux: Ten years later (ASPLOS 2011)
    - A study of linux file system evolution (FAST 2013)

Crash recovery experiment

recovery	4 writing processes	Git
No crash	51 (51%)	114 (76%)
File creation	40 (40%)	29 (19%)
File length	2 (2%)	7 (5%)
Data Recovery	1 (1%)	0

- Experiment
  - 4 processes write each secure file and verify them
  - Git : add files (20MB), sync, and add files (30MB).
  - 20 randomly selected faults are injected

Without Sego's recovery
Application keeps crashing

Sego correctly recovers every case

Sego overhead

Benchmark	Slowdown to Linux-VM
OpenLDAP	Insert (15.9%), Query (3.6%), Delete (15.0%)
Apache	Throughput (7.5%), Latency (8.2%)
Grep	Small file (10.1%), Large file (8.3%)
DokuWiki	90/10 read/write web pages (49%)

Conclusions

- Sego proposes the pervasive metadata model for
  - eliminating encryption and hashing for performance without losing security guarantees
  - detecting file system inconsistencies and recovery from crashes
- We hope the trusted metadata model will be adapted to device virtualization

Questions?

Fault injector - https://github.com/ut-osa/fault-injection