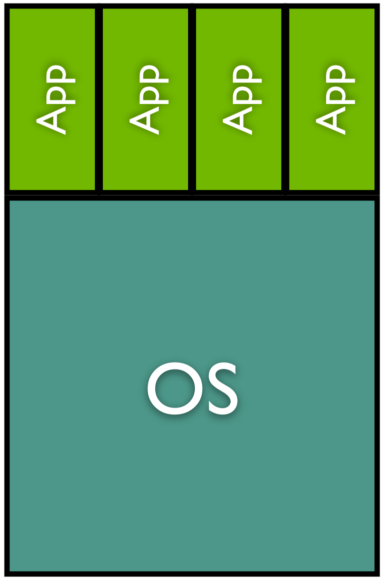
InkTag: Secure Applications on an Untrusted Operating System

Owen Hofmann, Sangman Kim, Alan Dunn, Mike Lee,
Emmett Witchel
UT Austin

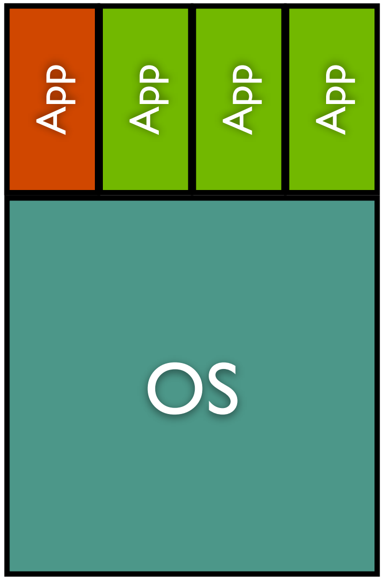
You trust your OS... should you?

- The OS is the software root of trust on most systems
- The OS is a shared vulnerability
  - OS compromise infects all
- The OS is a vulnerable vulnerability
  - Syscall interface a complex attack surface
  - `ioctl()`
- Root often has OS-level privilege



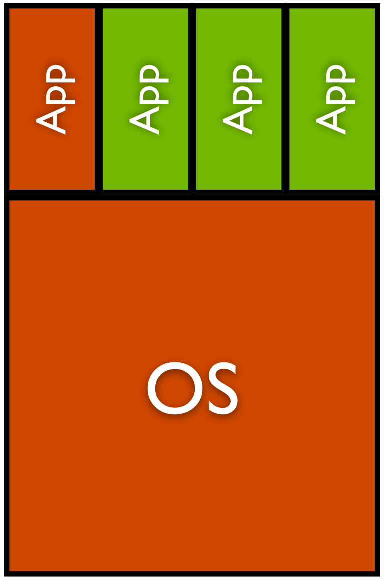
You trust your OS... should you?

- The OS is the software root of trust on most systems
- The OS is a shared vulnerability
  - OS compromise infects all
- The OS is a vulnerable vulnerability
  - Syscall interface a complex attack surface
  - `ioctl()`
- Root often has OS-level privilege



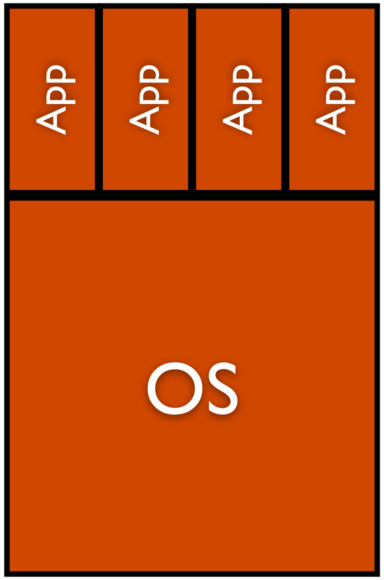
You trust your OS... should you?

- The OS is the software root of trust on most systems
- The OS is a shared vulnerability
  - OS compromise infects all
- The OS is a vulnerable vulnerability
  - Syscall interface a complex attack surface
  - `ioctl()`
- Root often has OS-level privilege



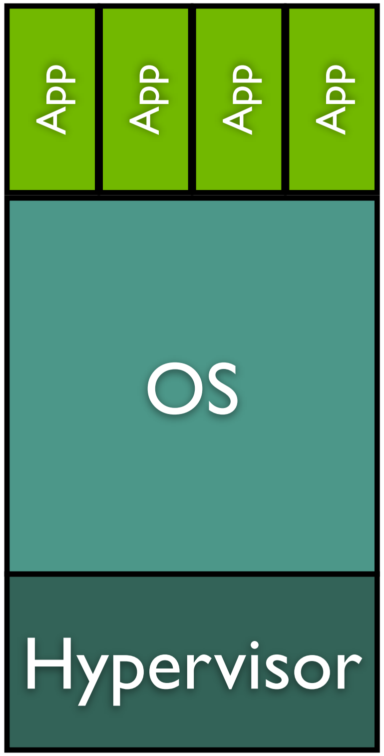
You trust your OS... should you?

- The OS is the software root of trust on most systems
- The OS is a shared vulnerability
    - OS compromise infects all
- The OS is a vulnerable vulnerability
    - Syscall interface a complex attack surface
    - `ioctl()`
- Root often has OS-level privilege



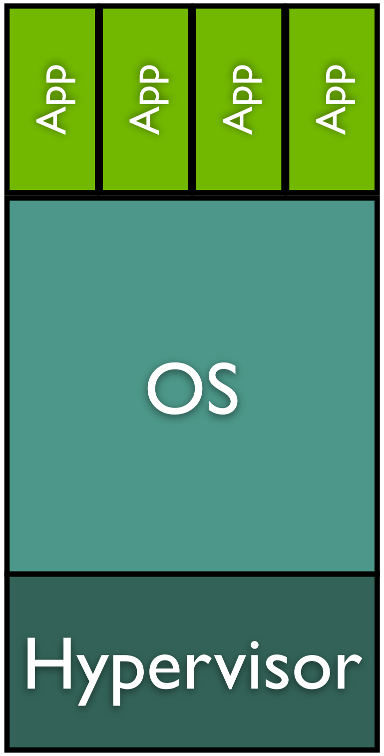
You should trust the hypervisor

- Hypervisors have become a common part of the software stack
  - Provide a layer of indirection under the OS
- Hypervisors can be more trustworthy
  - Fewer lines of code
  - Thinner interface
  - Fewer vulnerabilities



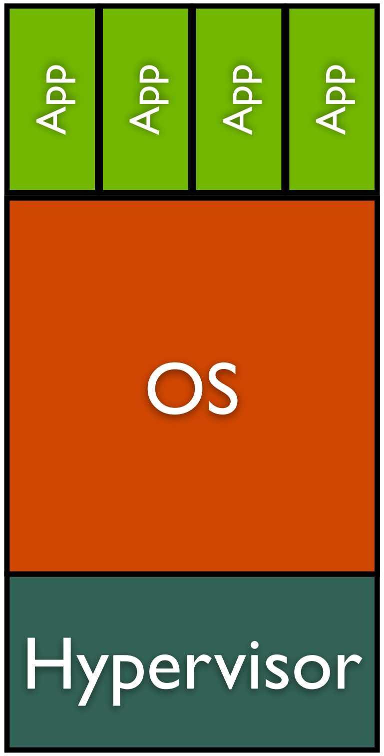
But the OS is still a problem

- Users want trustworthy applications
- Applications still must trust the OS



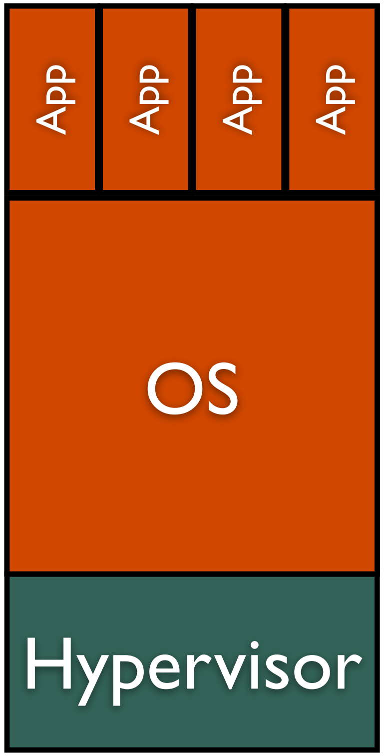
But the OS is still a problem

- Users want trustworthy applications
- Applications still must trust the OS



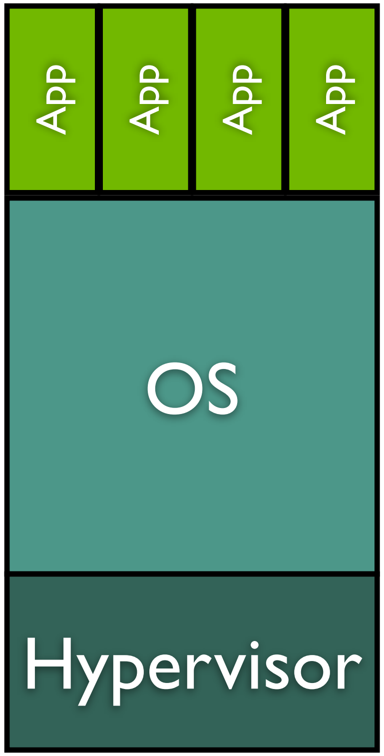
But the OS is still a problem

- Users want trustworthy applications
- Applications still must trust the OS



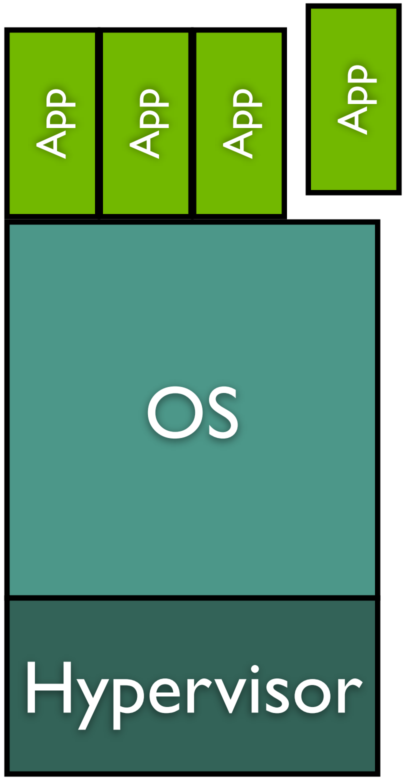
Removing OS trust

- Why can the kernel compromise applications?
- No isolation
- OS still provides all essential services
  - File I/O
  - Memory mapping



Isolate and verify

- Can the hypervisor improve this situation?
- Previous systems have examined this problem
  - Overshadow [ASPLOS ’08]
- Trusted hypervisor isolates an application from an untrusted kernel
- Ensure that the OS follows its contract with the application

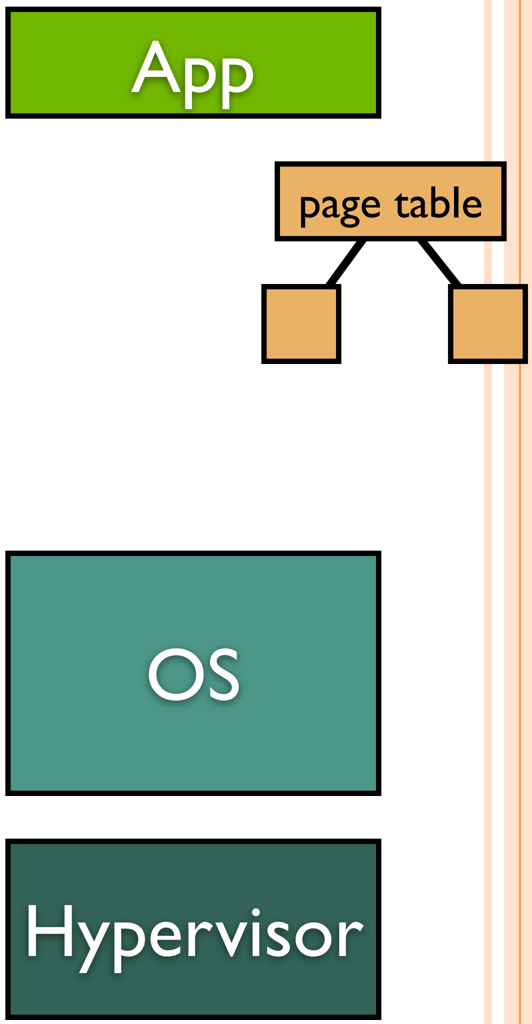


Verifying OS behavior

1. Application asks OS to update high-level state

2. OS updates low-level state

3. Do OS updates match application requests?

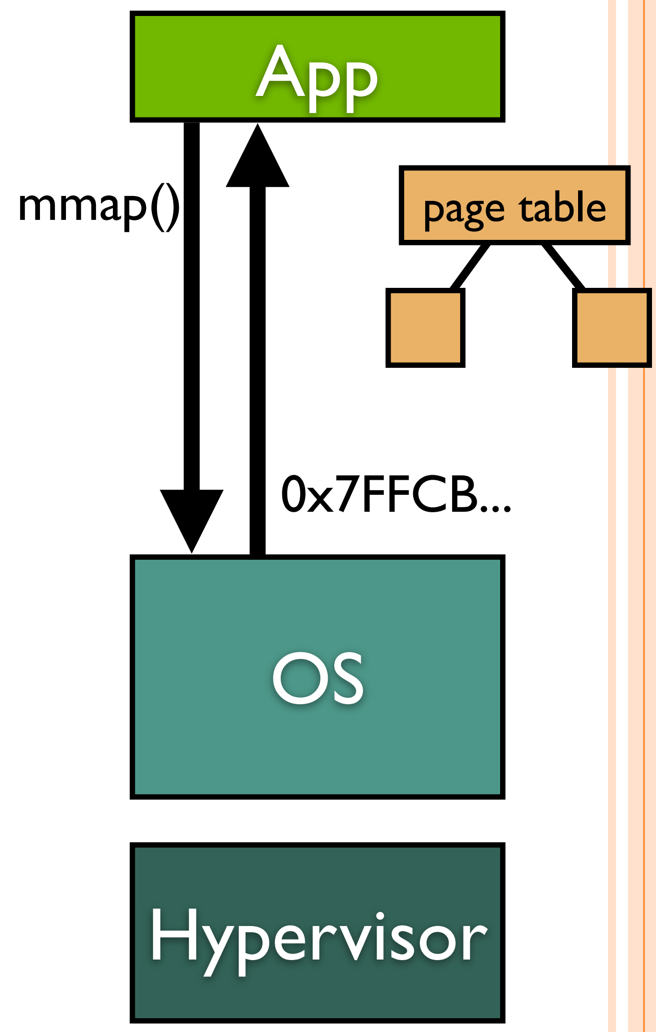


Verifying OS behavior

1. Application asks OS to update high-level state

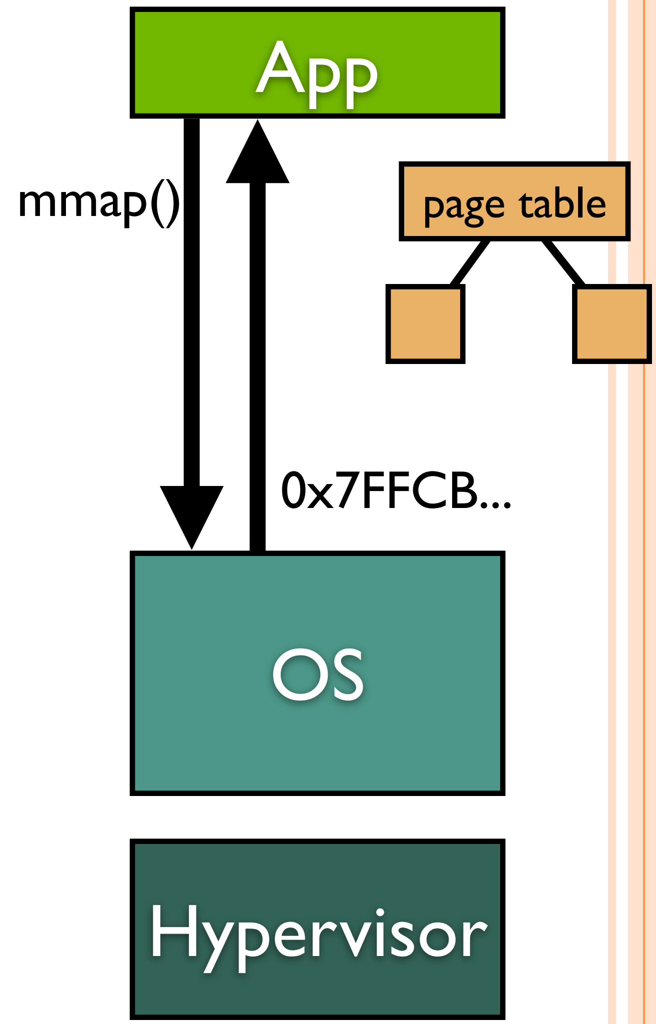
2. OS updates low-level state

3. Do OS updates match application requests?



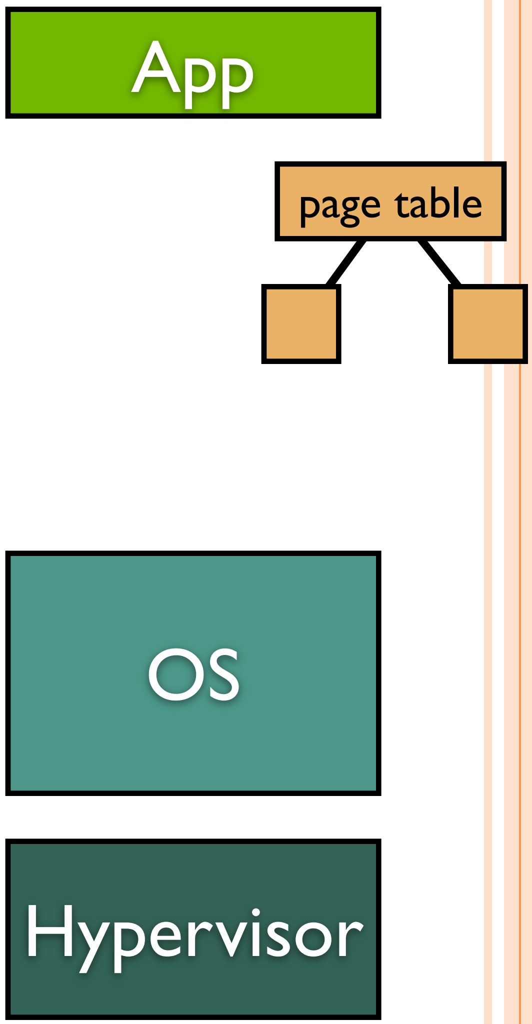
Verifying OS behavior

1. Application asks OS to update high-level state
    - V = mmap(file=F, offset=O);
    - Application expects pages from file *F* at address *V*
2. OS updates low-level state
3. Do OS updates match application requests?



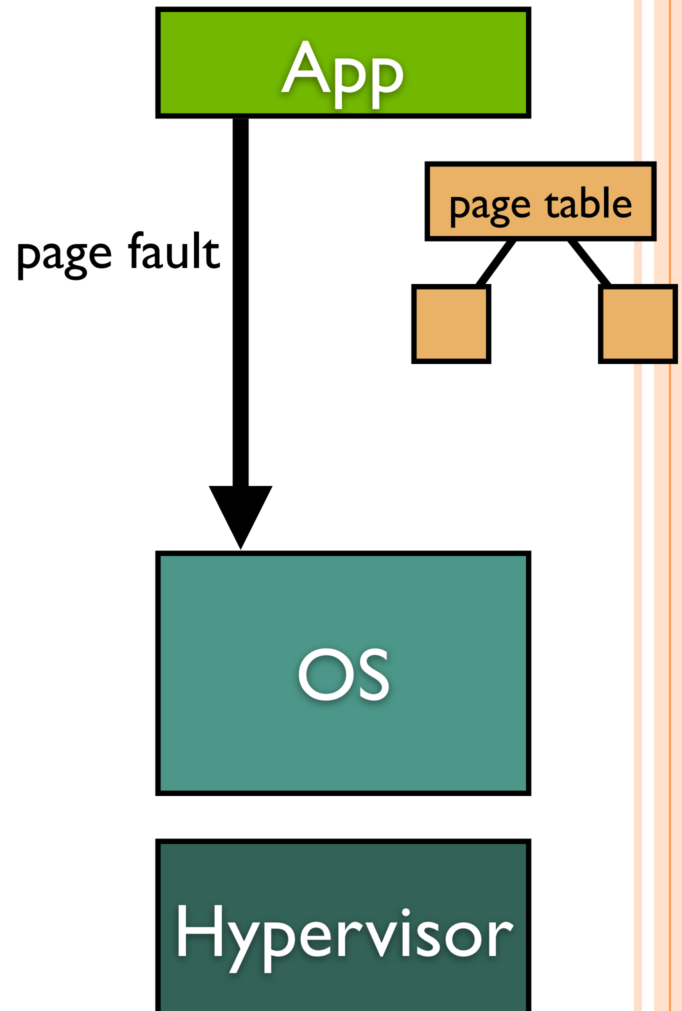
Verifying OS behavior

1. Application asks OS to update high-level state
   - V = mmap(file=F, offset=O);
   - Application expects pages from file *F* at address *V*
2. OS updates low-level state
   - Immediately
   - On-demand (e.g. paging)
3. Do OS updates match application requests?



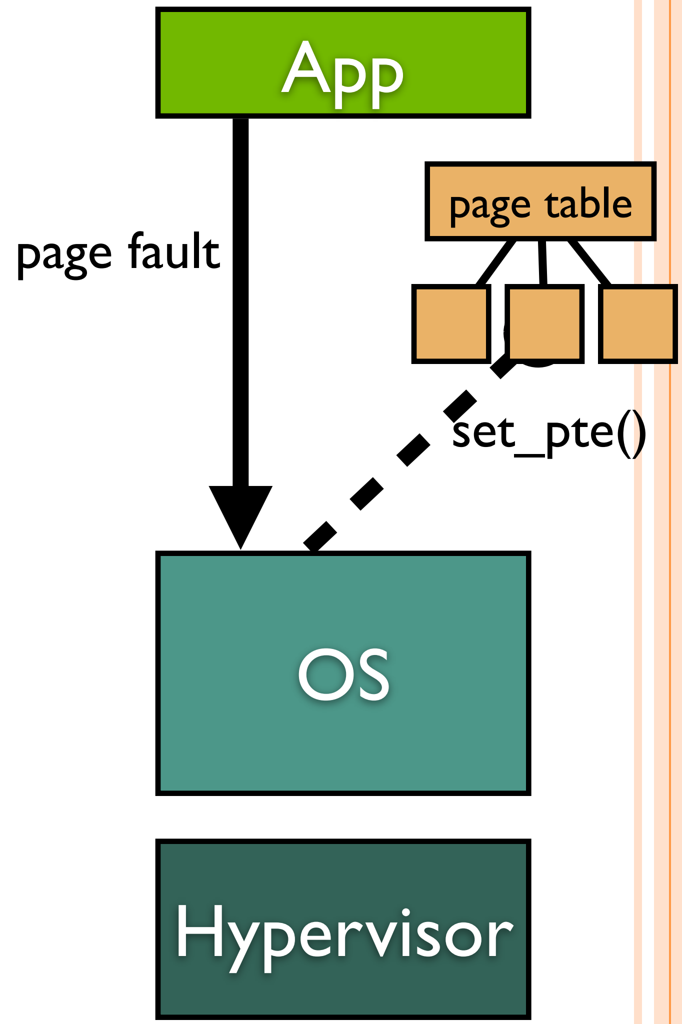
Verifying OS behavior

1. Application asks OS to update high-level state
   - V = mmap(file=F, offset=O);
   - Application expects pages from file *F* at address *V*
2. OS updates low-level state
   - Immediately
   - On-demand (e.g. paging)
3. Do OS updates match application requests?



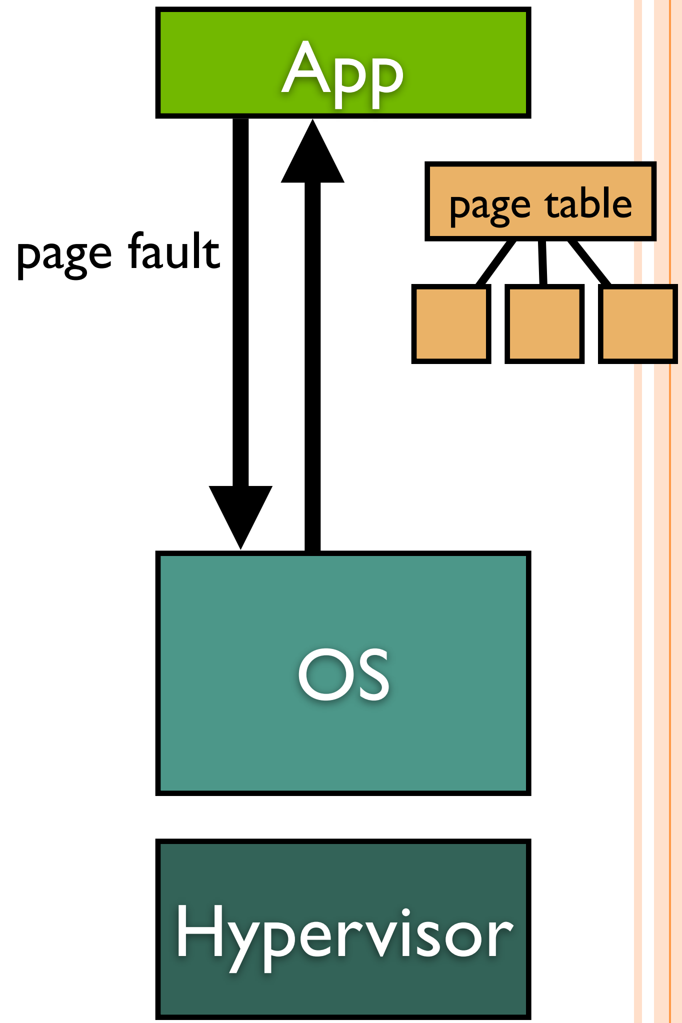
Verifying OS behavior

1. Application asks OS to update high-level state
   - V = mmap(file=F, offset=O);
   - Application expects pages from file *F* at address *V*
2. OS updates low-level state
   - Immediately
   - On-demand (e.g. paging)
3. Do OS updates match application requests?



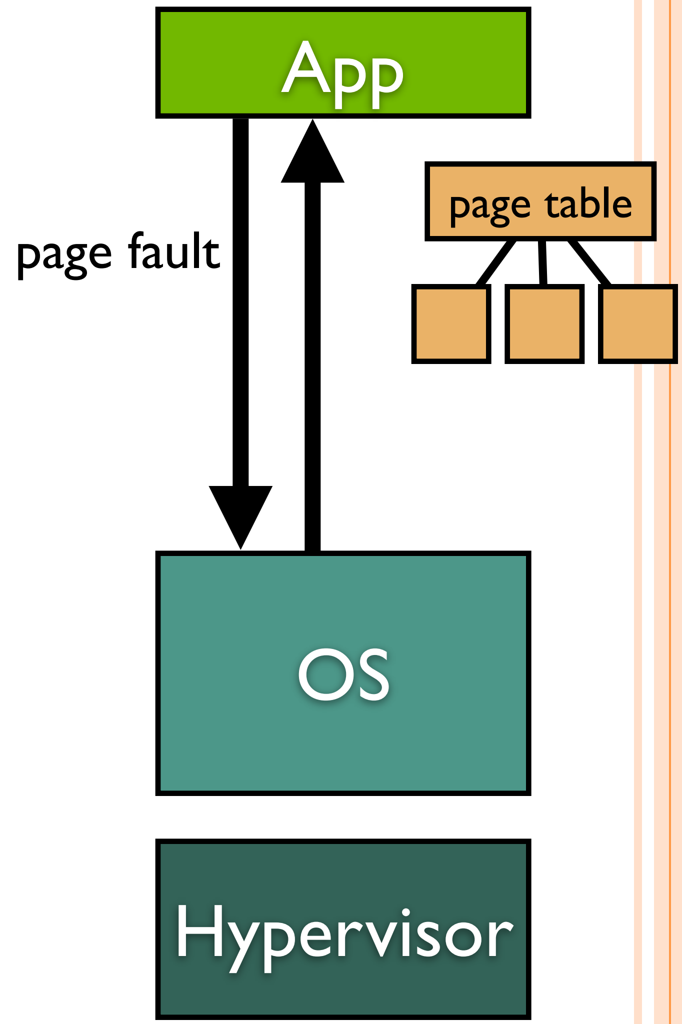
Verifying OS behavior

1. Application asks OS to update high-level state
   - V = mmap(file=F, offset=O);
   - Application expects pages from file *F* at address *V*
2. OS updates low-level state
   - Immediately
   - On-demand (e.g. paging)
3. Do OS updates match application requests?



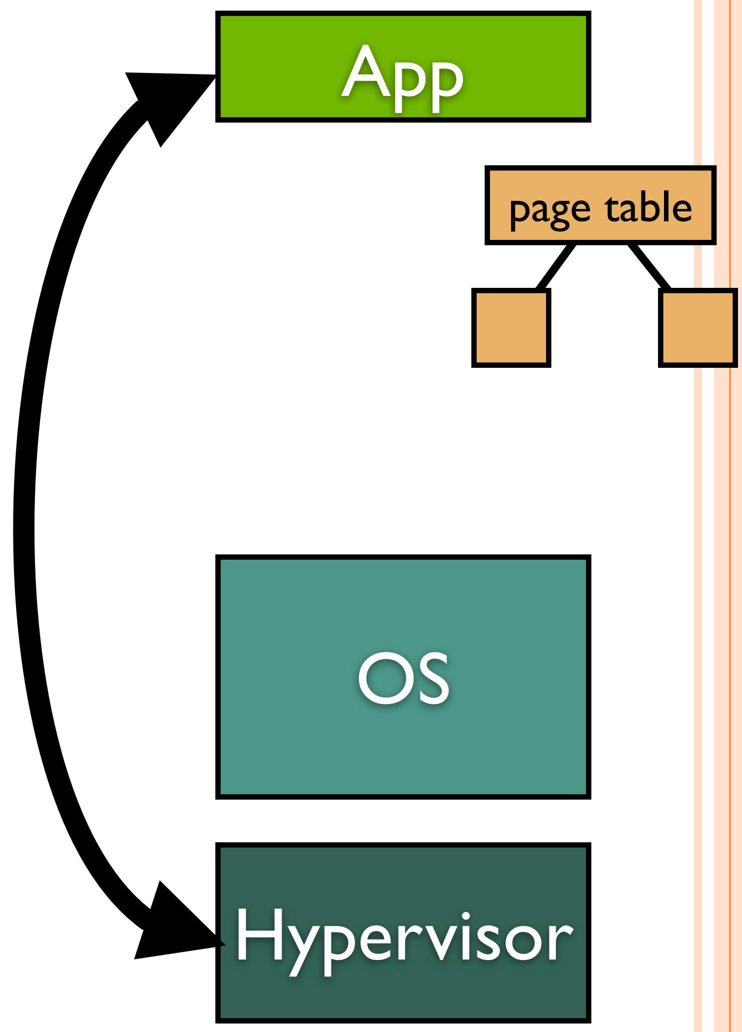
Verifying OS behavior

1. Application asks OS to update high-level state
   - V = mmap(file=F, offset=O);
   - Application expects pages from file *F* at address *V*
2. OS updates low-level state
   - Immediately
   - On-demand (e.g. paging)
3. Do OS updates match application requests?
   - Did the OS map a frame containing data from F at the correct offset?



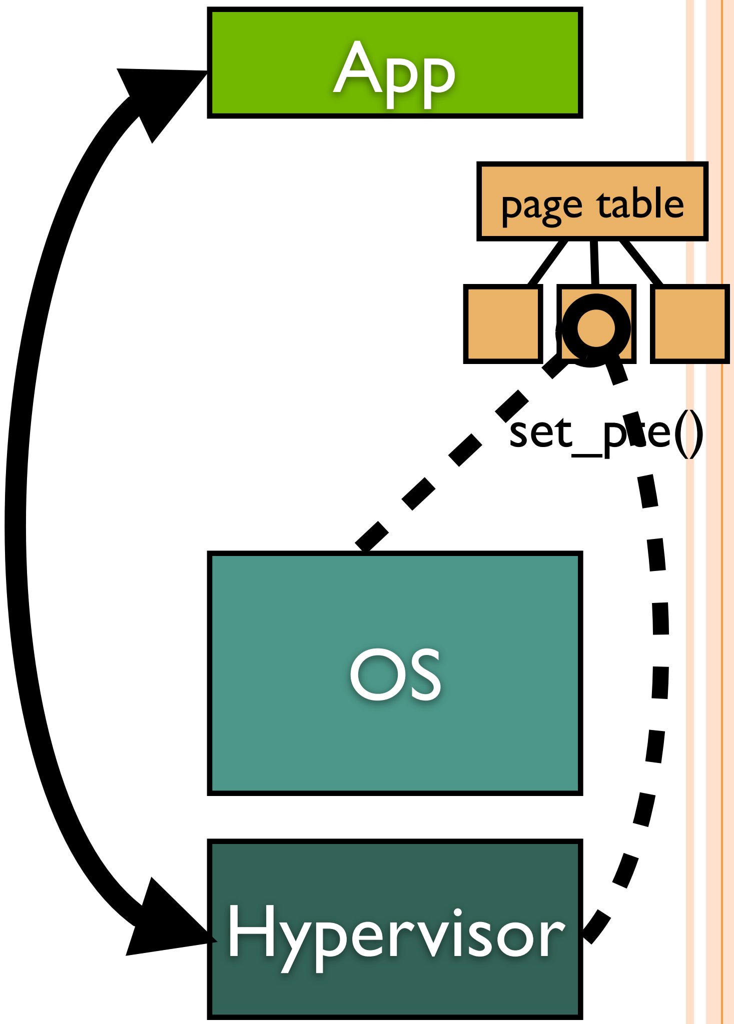
Verifying OS behavior

- Application and hypervisor communicate
  - Synchronize on high-level application state
- Hypervisor interposes on low-level updates
  - Validate updates against expected state
- Hypervisor requires deep visibility into OS, application (semantic gap)



Verifying OS behavior

- Application and hypervisor communicate
  - Synchronize on high-level application state
- Hypervisor interposes on low-level updates
  - Validate updates against expected state
- Hypervisor requires deep visibility into OS, application (semantic gap)



- InkTag: secure applications on an untrusted OS
- Paraverification: require active participation from the untrusted OS for simpler, more efficient hypervisor design

InkTag security guarantees

- Control flow integrity
  - OS cannot change program counter, registers
- Address space integrity
  - OS cannot read or modify application data
- File I/O
  - Applications access the desired files
  - Privacy and integrity for file data
  - Built on address space integrity
- Process control
  - Applications can fork(), exec()
- Access control and naming
  - Applications can define access control policies, use string filenames
- Consistency
  - OS-managed data and hypervisor-managed metadata remain in sync

InkTag security guarantees

- Control flow integrity
    - OS cannot change program counter, registers
- Address space integrity
    - OS cannot read or modify application data
- File I/O
    - Applications access the desired files
    - Privacy and integrity for file data
    - Built on address space integrity
- Process control
    - Applications can fork(), exec()
- Access control and naming
    - Applications can define access control policies, use string filenames
- Consistency
    - OS-managed data and hypervisor-managed metadata remain in sync

InkTag security guarantees

- Control flow integrity
  - OS cannot change program counter, registers
- Address space integrity
  - OS cannot read or modify application data
- File I/O
  - Applications access the desired files
  - Privacy and integrity for file data
  - Built on address space integrity
- Process control
  - Applications can fork(), exec()
- Access control and naming
  - Applications can define access control policies, use string filenames
- Consistency
  - OS-managed data and hypervisor-managed metadata remain in sync

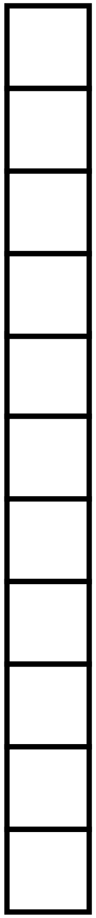
InkTag security guarantees

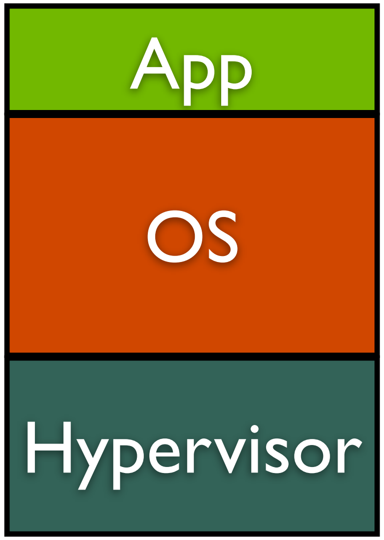
- Address space integrity
- File I/O

- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

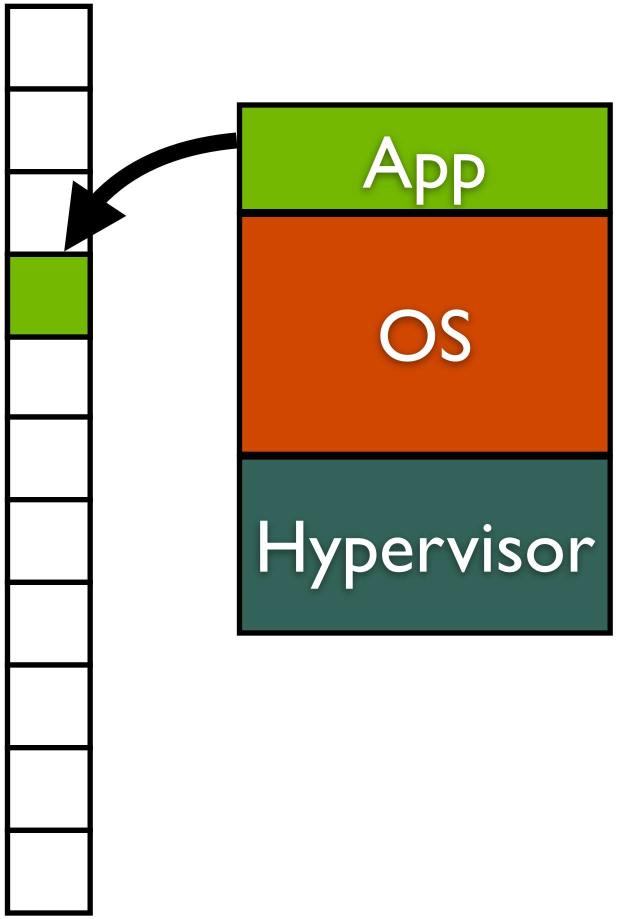
- Common mechanism used by Overshadow, InkTag, others
- OS expects to manage memory
- Show cleartext to application
- Show ciphertext to OS
- Hash for integrity





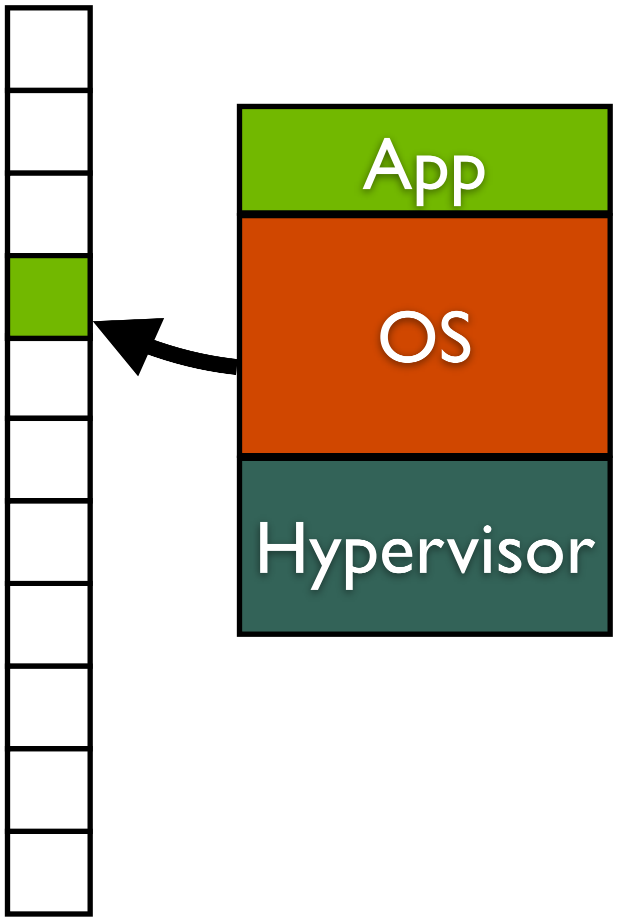
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Common mechanism used by Overshadow, InkTag, others
- OS expects to manage memory
- Show cleartext to application
- Show ciphertext to OS
- Hash for integrity



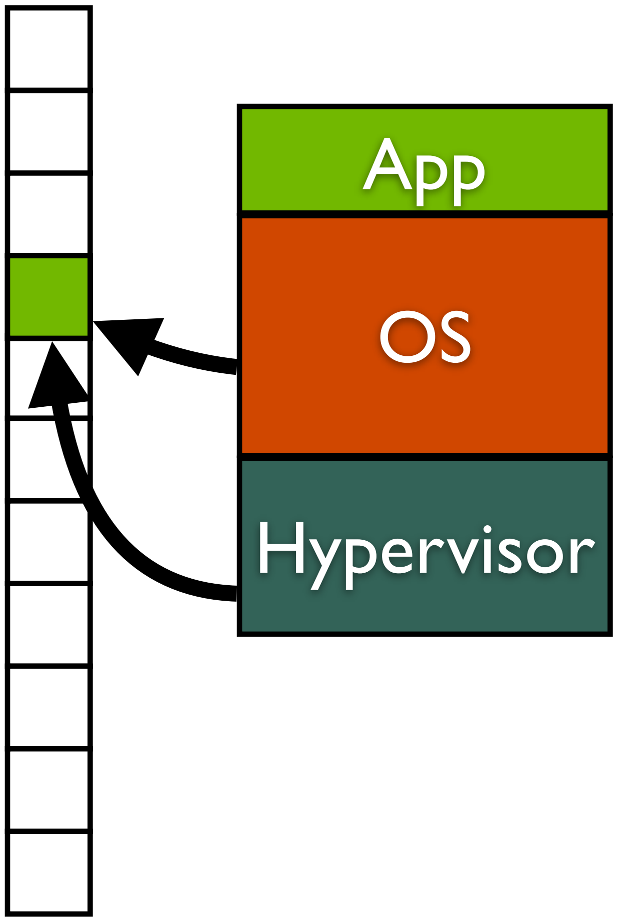
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Common mechanism used by Overshadow, InkTag, others
- OS expects to manage memory
- Show cleartext to application
- Show ciphertext to OS
- Hash for integrity



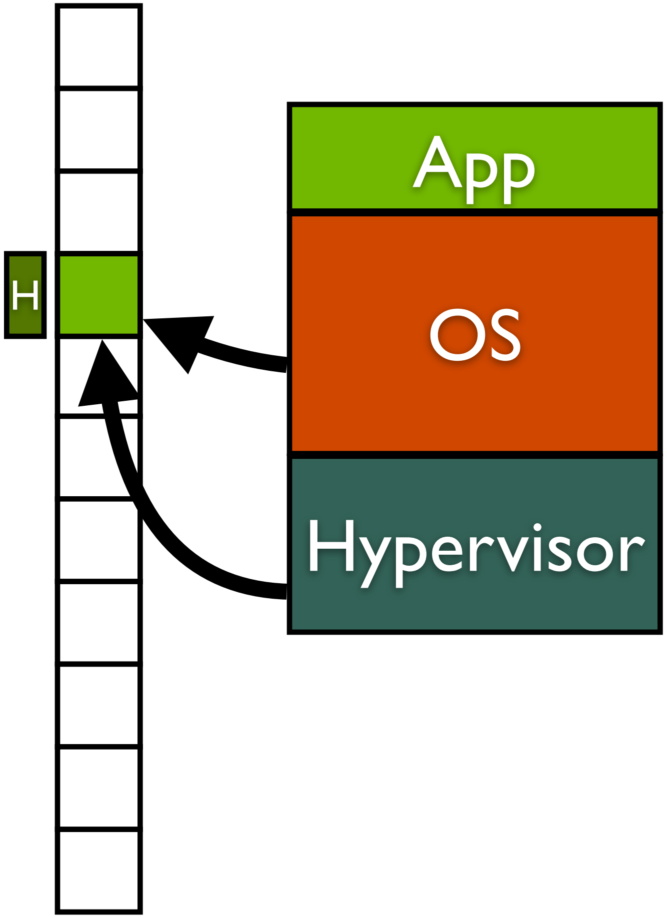
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Common mechanism used by Overshadow, InkTag, others
- OS expects to manage memory
- Show cleartext to application
- Show ciphertext to OS
- Hash for integrity



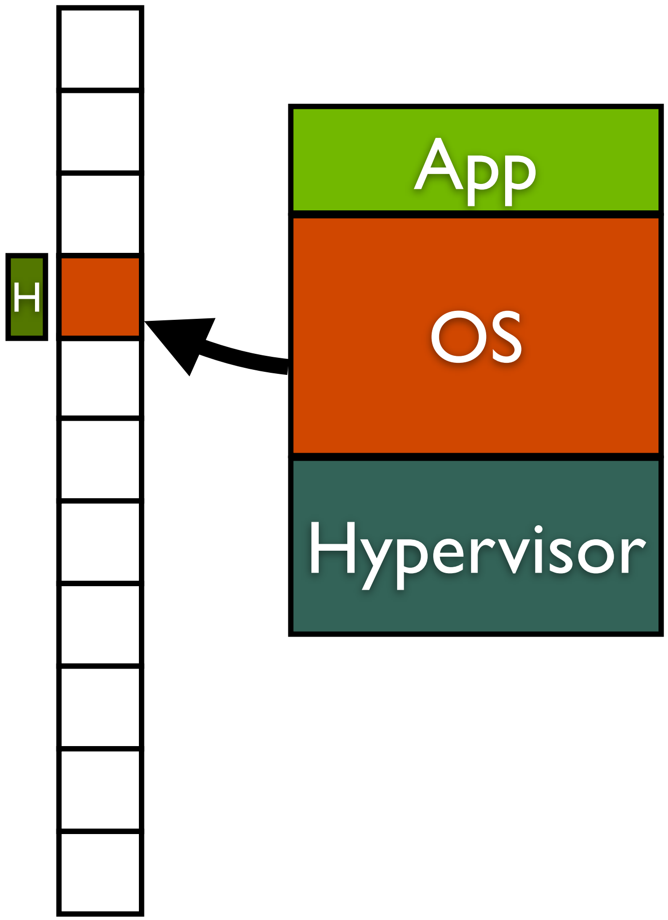
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Common mechanism used by Overshadow, InkTag, others
- OS expects to manage memory
- Show cleartext to application
- Show ciphertext to OS
- Hash for integrity



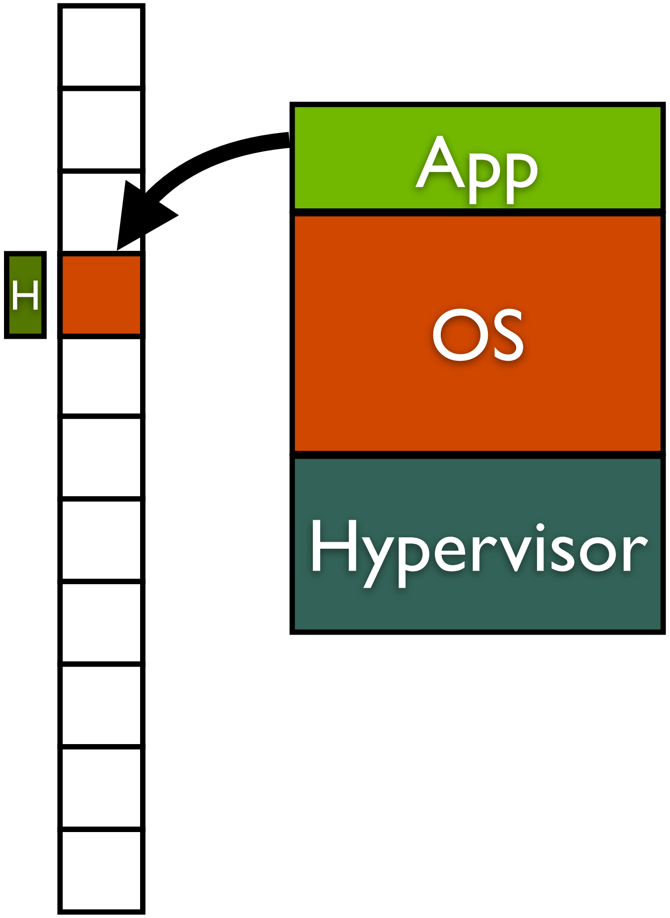
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Common mechanism used by Overshadow, InkTag, others
- OS expects to manage memory
- Show cleartext to application
- Show ciphertext to OS
- Hash for integrity



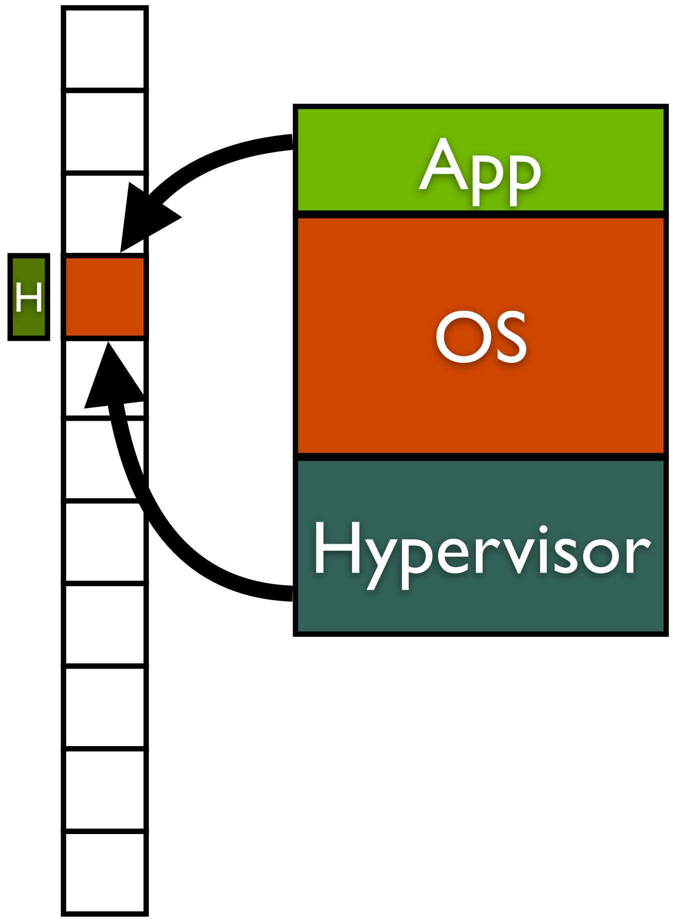
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Common mechanism used by Overshadow, InkTag, others
- OS expects to manage memory
- Show cleartext to application
- Show ciphertext to OS
- Hash for integrity



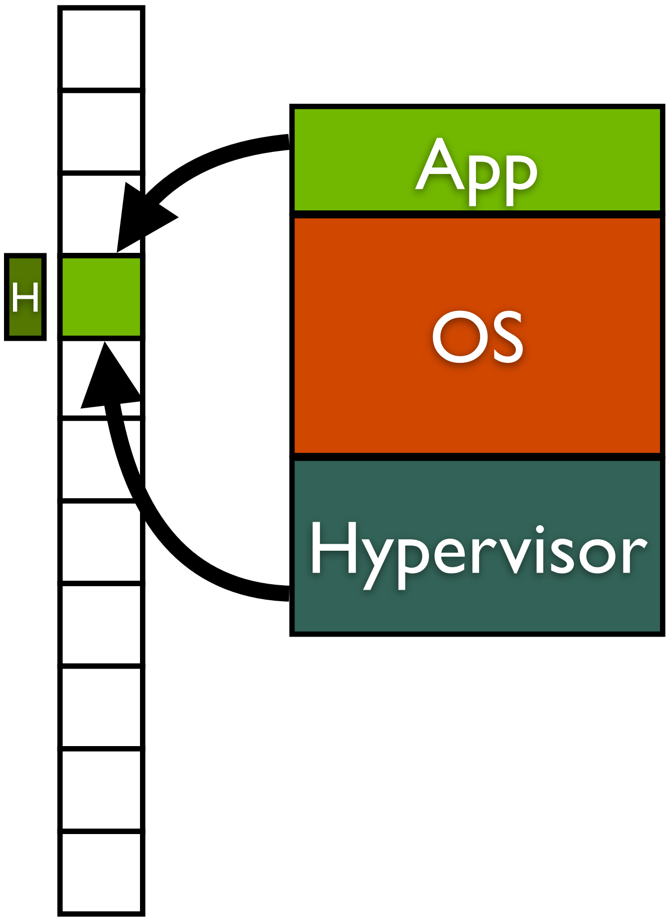
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Common mechanism used by Overshadow, InkTag, others
- OS expects to manage memory
- Show cleartext to application
- Show ciphertext to OS
- Hash for integrity



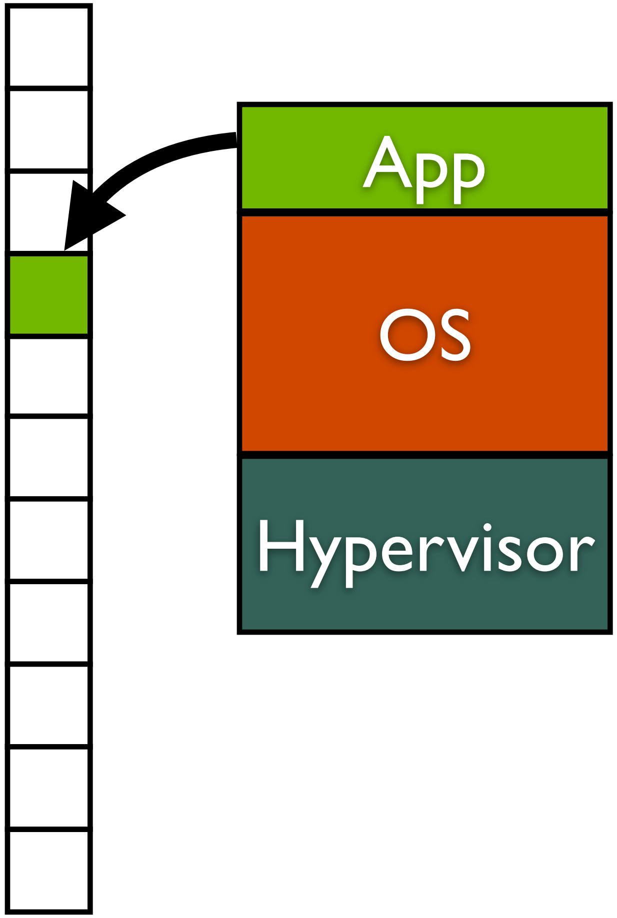
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Common mechanism used by Overshadow, InkTag, others
- OS expects to manage memory
- Show cleartext to application
- Show ciphertext to OS
- Hash for integrity



- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

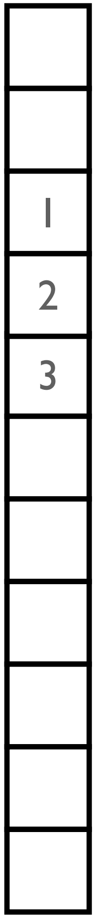
- Common mechanism used by Overshadow, InkTag, others
- OS expects to manage memory
- Show cleartext to application
- Show ciphertext to OS
- Hash for integrity

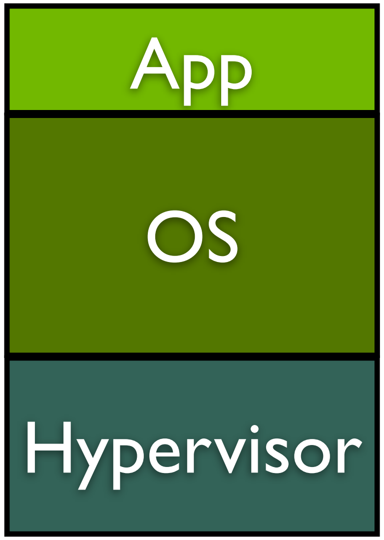


- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

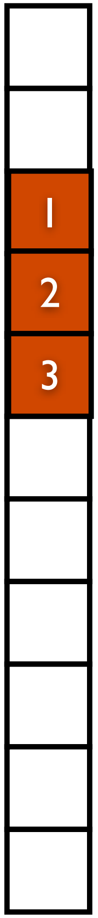
- Position of data in address space must match application requests [mmap()]

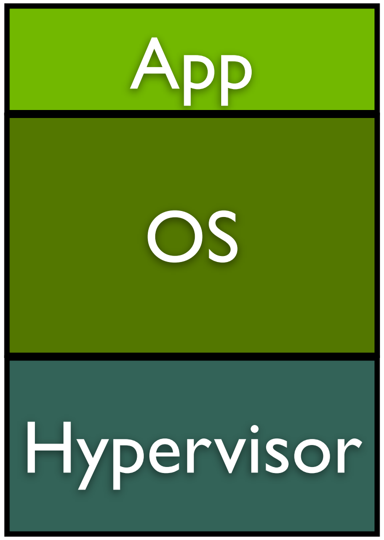
- Ensure OS constructs the correct address space





- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

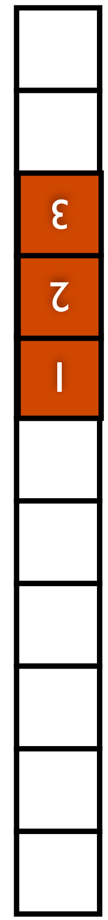




- Position of data in address space must match application requests [mmap()]

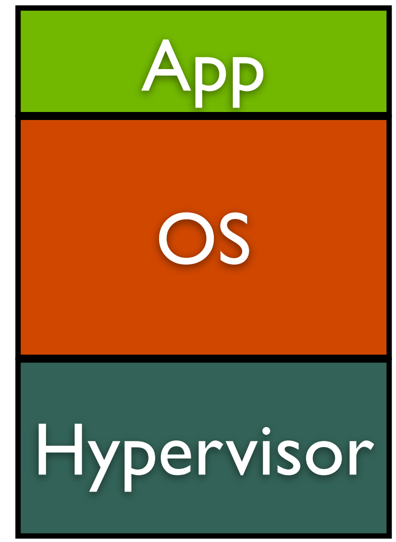
- Ensure OS constructs the correct address space

- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?



- Position of data in address space must match application requests [mmap()]

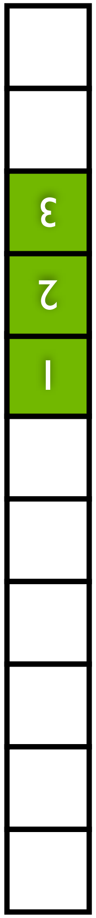
- Ensure OS constructs the correct address space

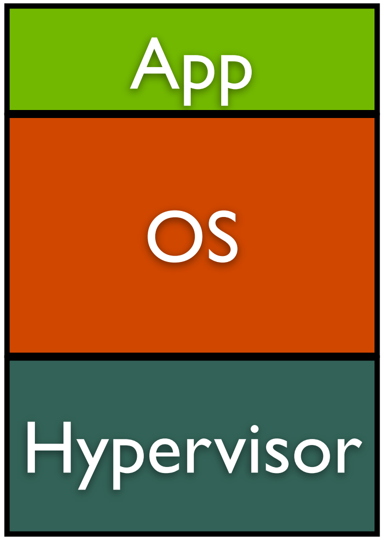


- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

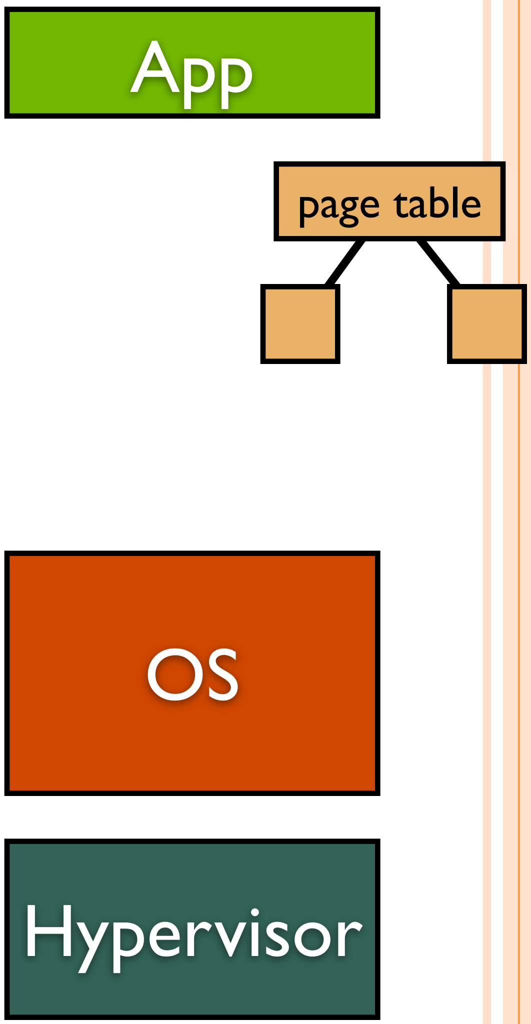
- Position of data in address space must match application requests [mmap()]

- Ensure OS constructs the correct address space



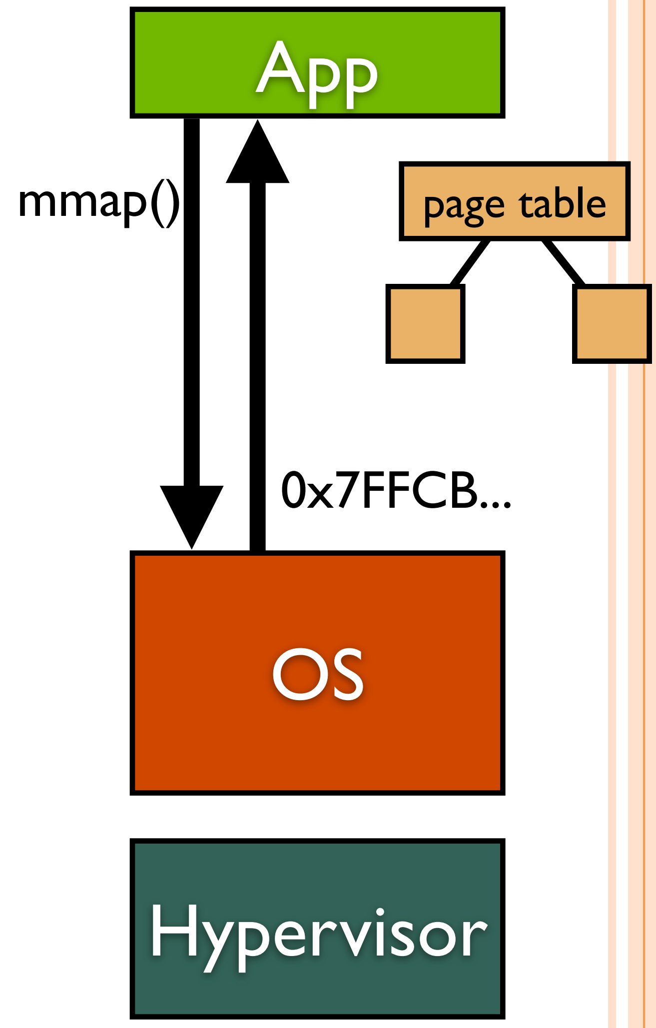


- Ensure OS constructs the correct address space
- Application maps file *F* at addr *V*
- Are page faults to *V* handled correctly?
  - Decrypted physical frame has same hash as *F*
- Interpose on page table updates



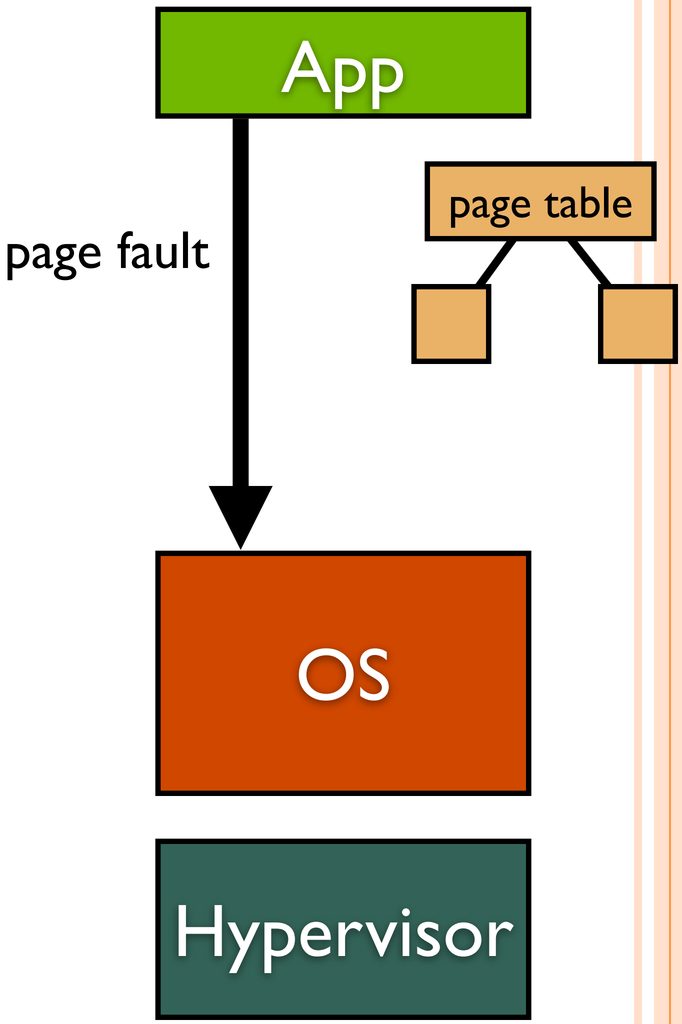
- Match page table updates to application requests

- Ensure OS constructs the correct address space
- Application maps file *F* at addr *V*
- Are page faults to *V* handled correctly?
    - Decrypted physical frame has same hash as *F*
- Interpose on page table updates
- Match page table updates to application requests

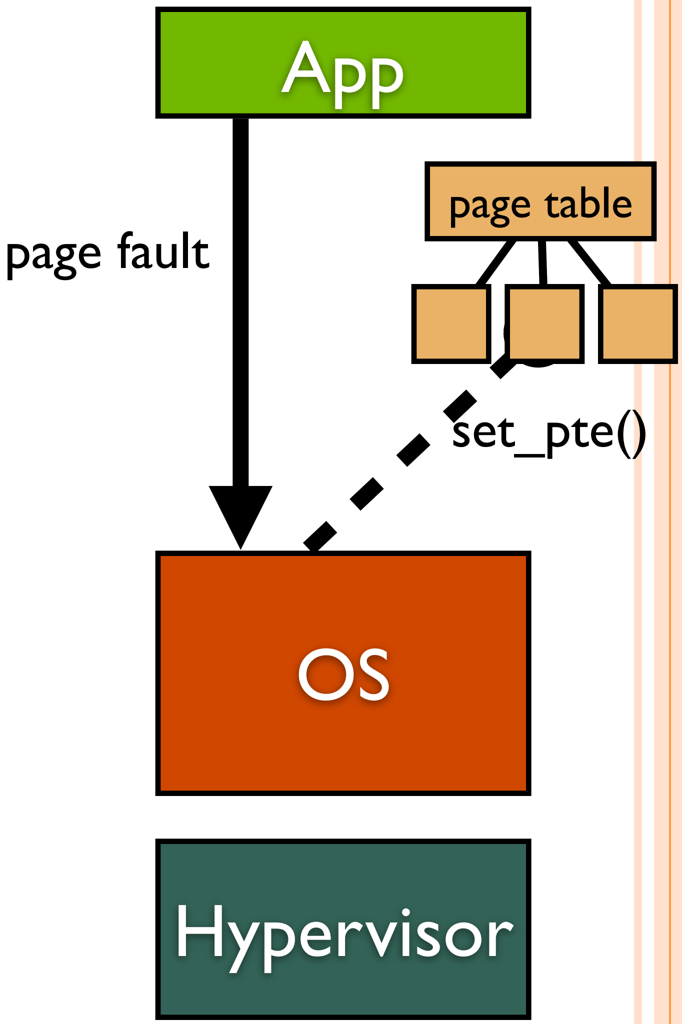


- Ensure OS constructs the correct address space
- Application maps file *F* at addr *V*
- Are page faults to *V* handled correctly?
  - Decrypted physical frame has same hash as *F*
- Interpose on page table updates

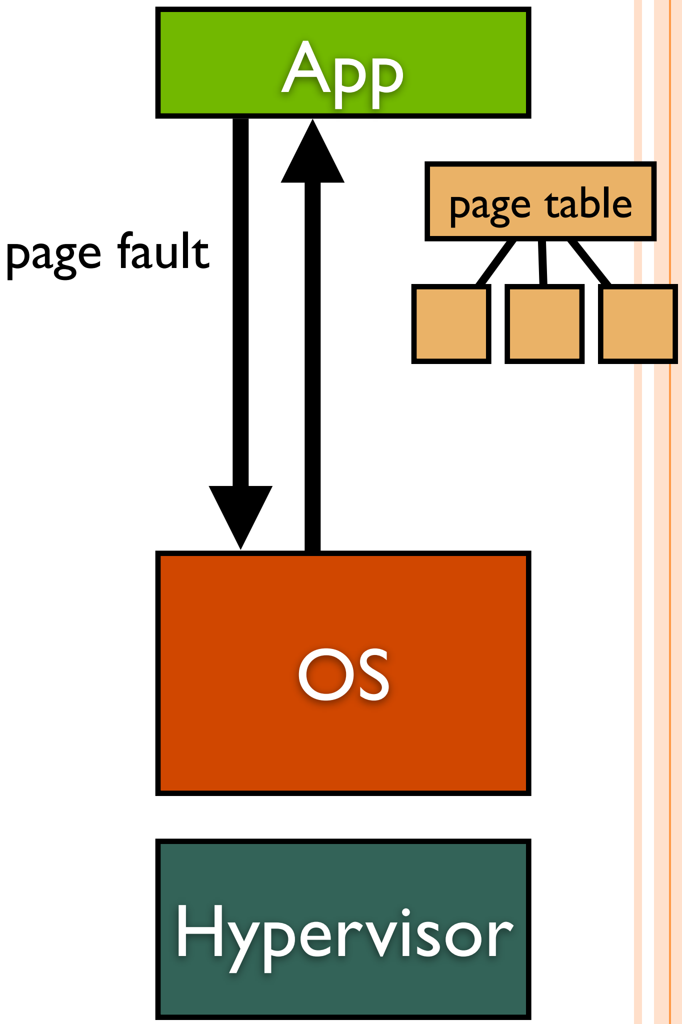
- Match page table updates to application requests



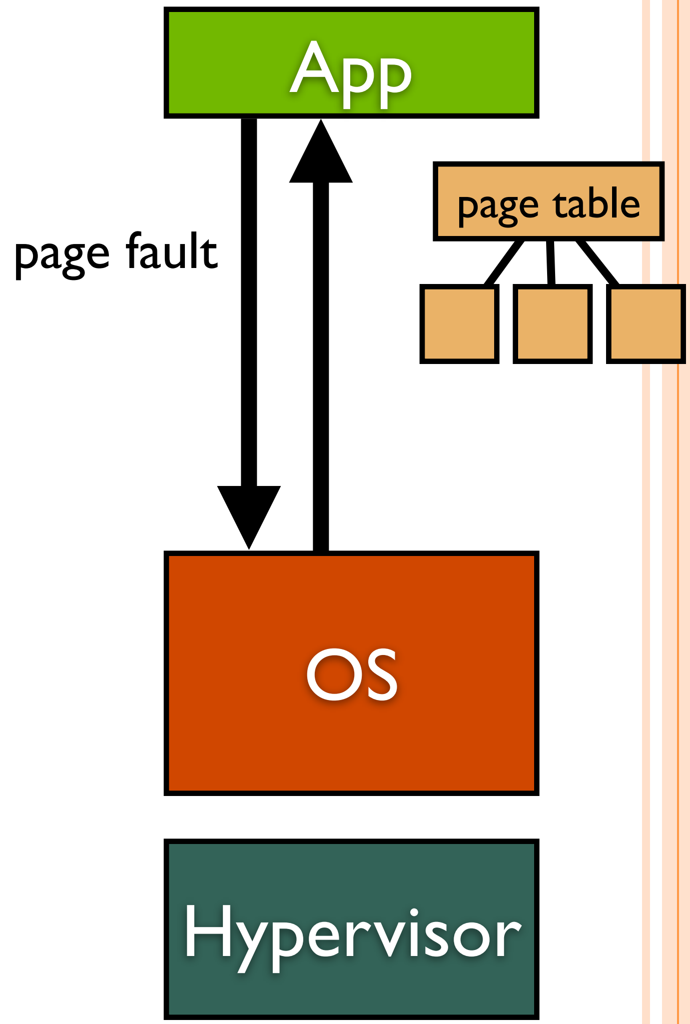
- Ensure OS constructs the correct address space
- Application maps file *F* at addr *V*
- Are page faults to *V* handled correctly?
    - Decrypted physical frame has same hash as *F*
- Interpose on page table updates
- Match page table updates to application requests



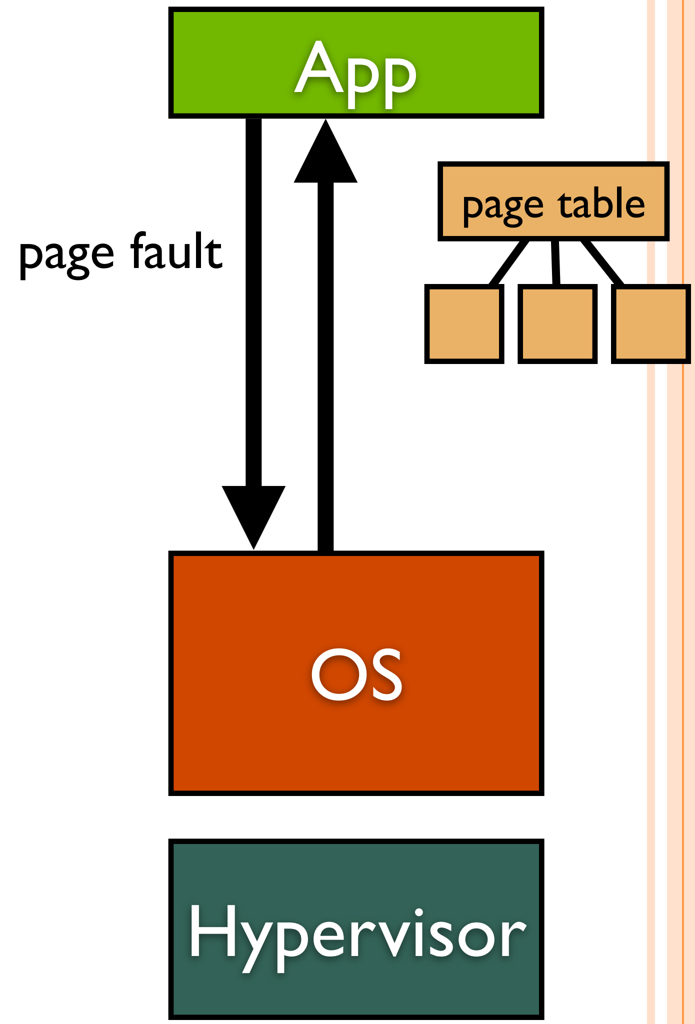
- Ensure OS constructs the correct address space
- Application maps file *F* at addr *V*
- Are page faults to *V* handled correctly?
  - Decrypted physical frame has same hash as *F*
- Interpose on page table updates
- Match page table updates to application requests



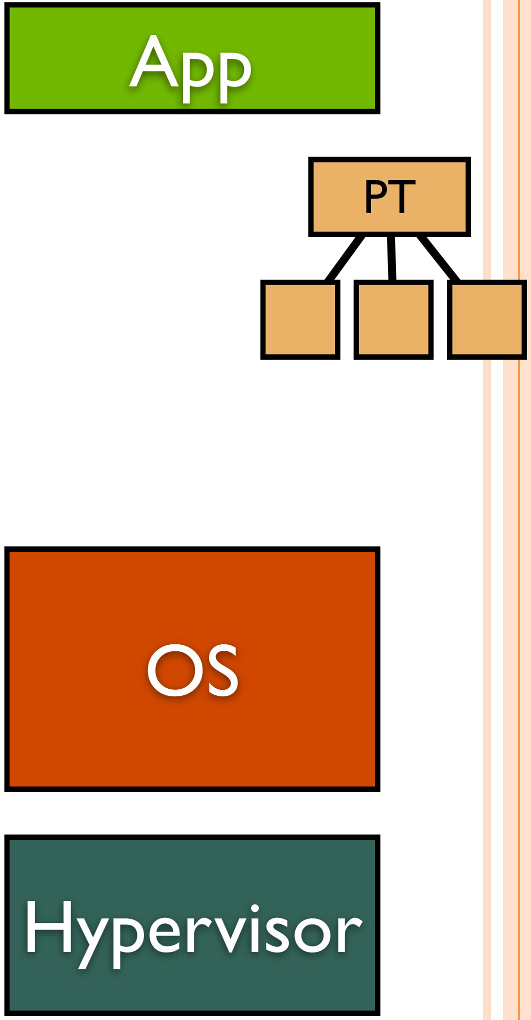
- Ensure OS constructs the correct address space
- Application maps file *F* at addr *V*
- Are page faults to *V* handled correctly?
  - Decrypted physical frame has same hash as *F*
- Interpose on page table updates
  - Disallow arbitrary OS mapping
  - Determine high-level update implied by low-level PTE change
- Match page table updates to application requests



- Ensure OS constructs the correct address space
- Application maps file *F* at addr *V*
- Are page faults to *V* handled correctly?
  - Decrypted physical frame has same hash as *F*
- Interpose on page table updates
  - Disallow arbitrary OS mapping
  - Determine high-level update implied by low-level PTE change
- Match page table updates to application requests
  - Virtual address *V* = file *F*, offset *O*
  - Result of previous mmap() call

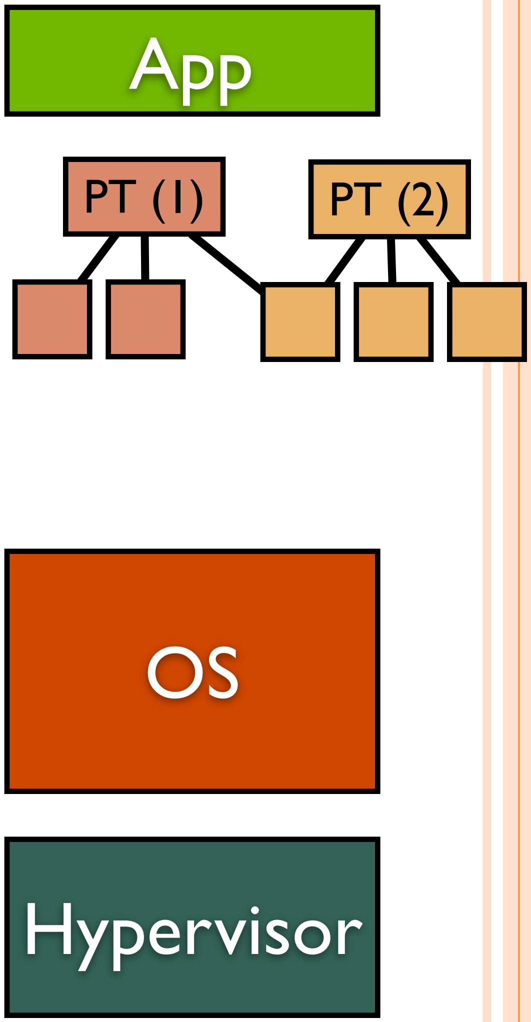


- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?



- Interpreting low-level page table updates
  - OS can construct valid, but confusing page tables
  - Order in which updates are seen matters
- Matching page table updates to application requests
  - Application and hypervisor must communicate complete memory map

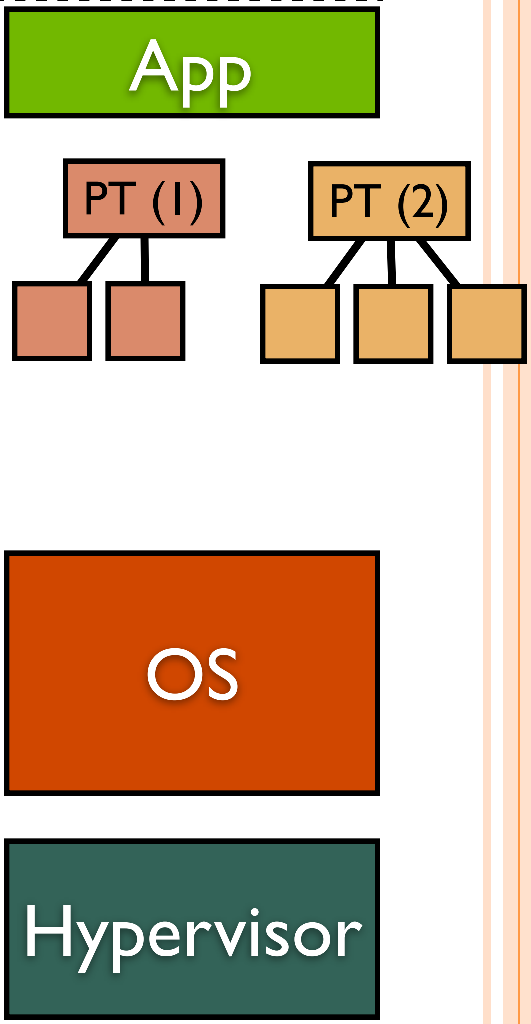
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?



- Interpreting low-level page table updates
  - OS can construct valid, but confusing page tables
  - Order in which updates are seen matters
- Matching page table updates to application requests
  - Application and hypervisor must communicate complete memory map

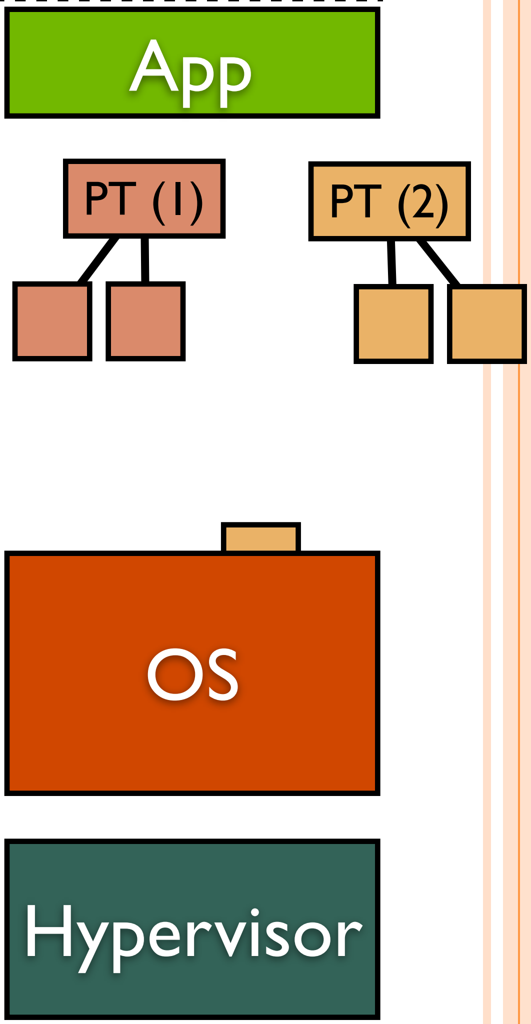
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Interpreting low-level page table updates
  - OS can construct valid, but confusing page tables
  - Order in which updates are seen matters
- Matching page table updates to application requests
  - Application and hypervisor must communicate complete memory map



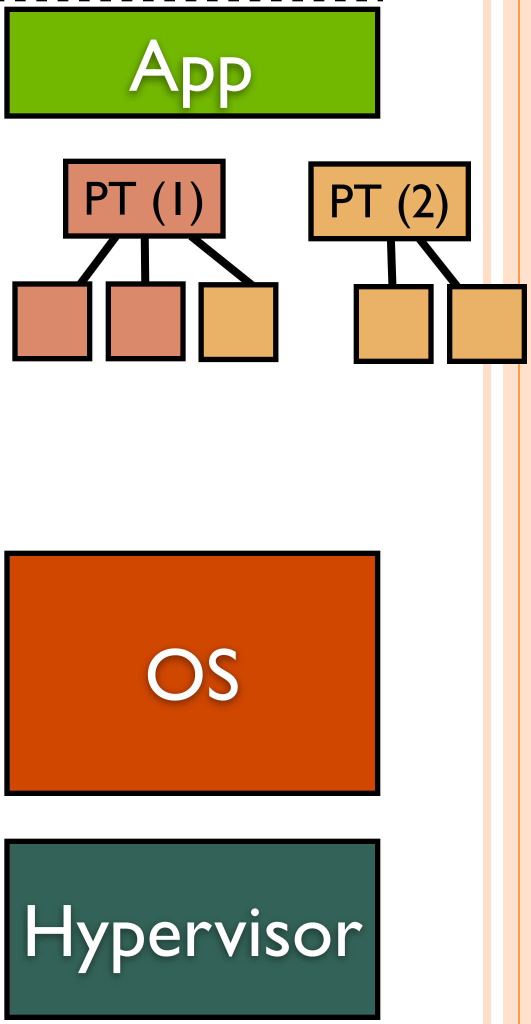
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Interpreting low-level page table updates
  - OS can construct valid, but confusing page tables
  - Order in which updates are seen matters
- Matching page table updates to application requests
  - Application and hypervisor must communicate complete memory map

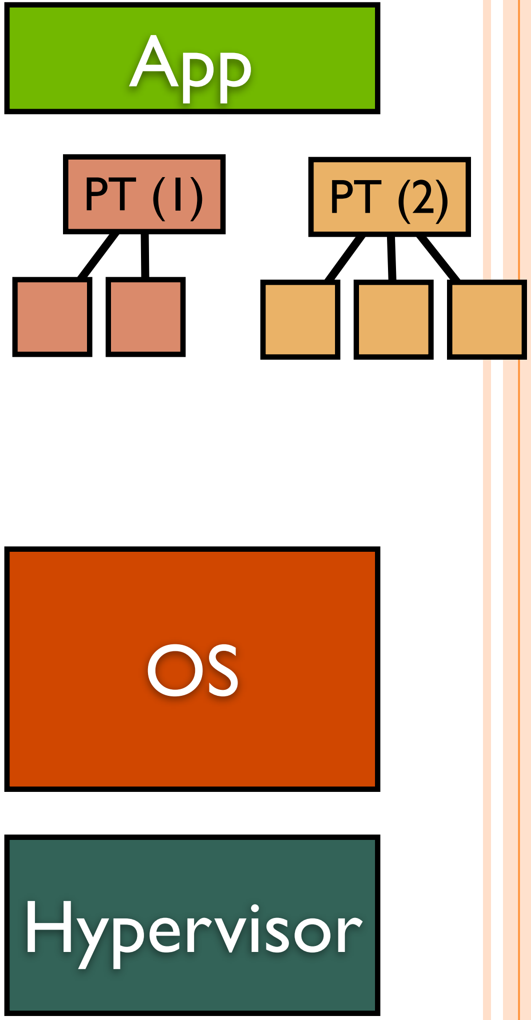


- Basic memory isolation mechanisms
- **Challenges: why is this difficult?**
- Paraverification: how can the untrusted OS help?

- Interpreting low-level page table updates
  - OS can construct valid, but confusing page tables
  - Order in which updates are seen matters
- Matching page table updates to application requests
  - Application and hypervisor must communicate complete memory map



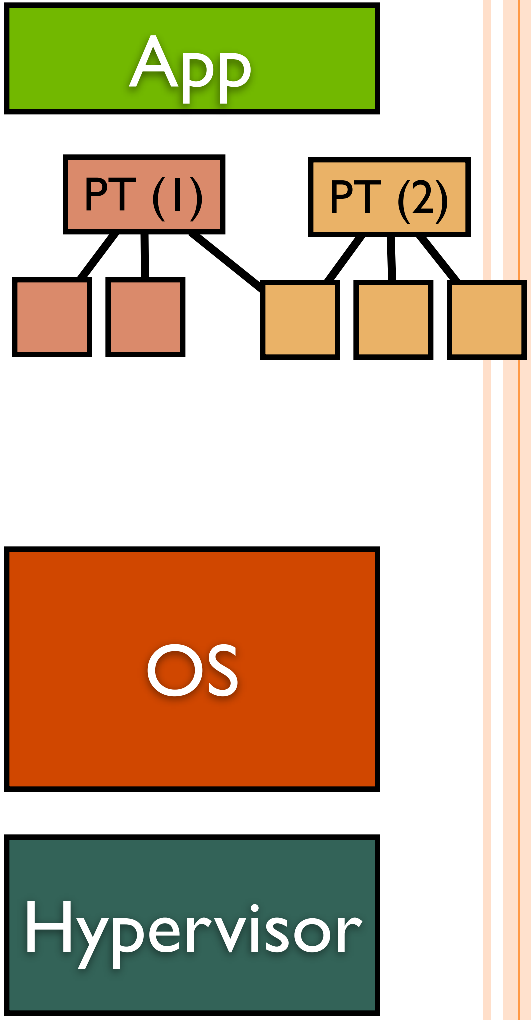
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?



- Interpreting low-level page table updates
  - OS can construct valid, but confusing page tables
  - Order in which updates are seen matters
- Matching page table updates to application requests
  - Application and hypervisor must communicate complete memory map

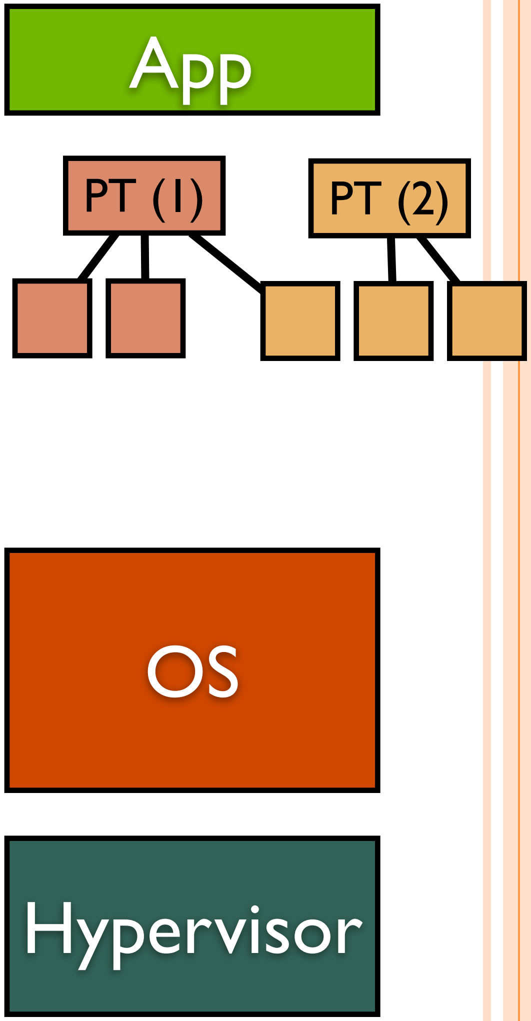
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Interpreting low-level page table updates
  - OS can construct valid, but confusing page tables
  - Order in which updates are seen matters
- Matching page table updates to application requests
  - Application and hypervisor must communicate complete memory map

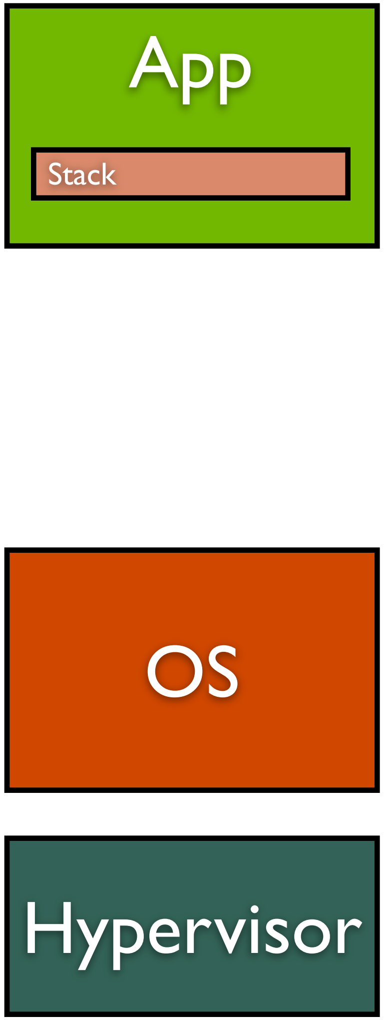


- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- Interpreting low-level page table updates
  - OS can construct valid, but confusing page tables
  - Order in which updates are seen matters
- Matching page table updates to application requests
  - Application and hypervisor must communicate complete memory map

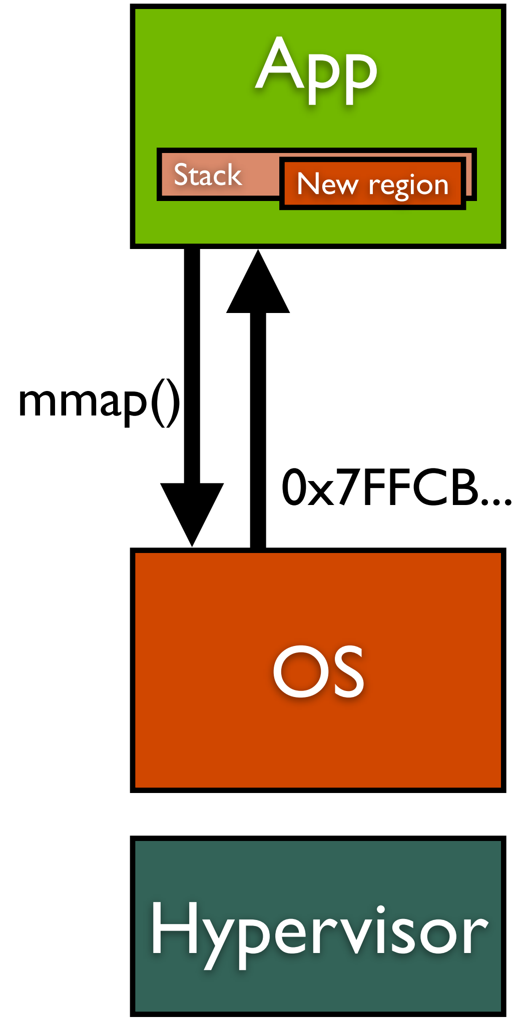


- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?



- Application must validate pointer results returned from kernel
- *Iago attacks* [ASPLOS '13]

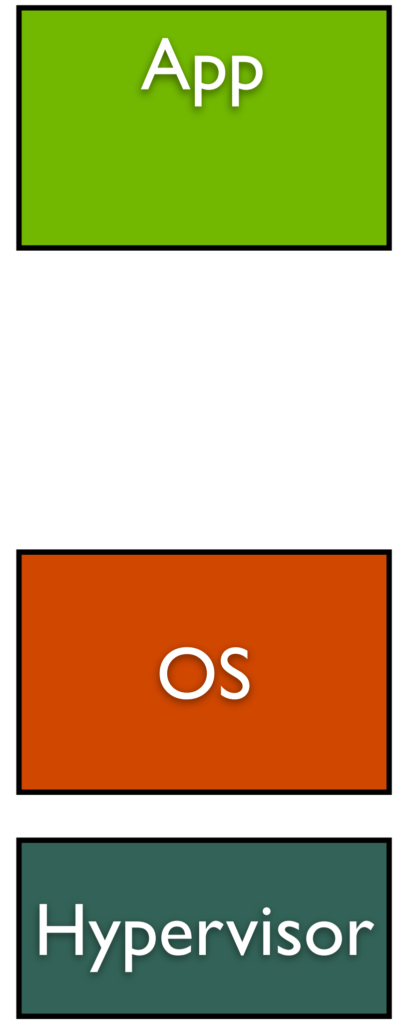
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?



- Application must validate pointer results returned from kernel
- *Iago attacks* [ASPLOS '13]

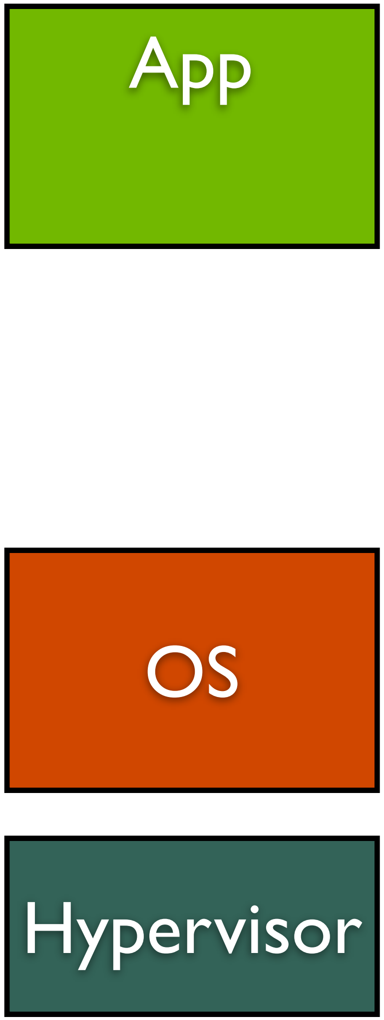
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- The OS updates page tables
  - Can guarantee sanity and ordering
- The OS maintains memory maps
  - Can expose that information to hypervisor and application



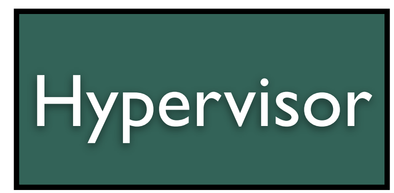
- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?

- *Paraverification*: an untrusted OS helping to verify its own behavior
  - Take inspiration from paravirtualization
  - Extensive use of existing paravirtual interface
- OS must participate, but information cannot be trusted

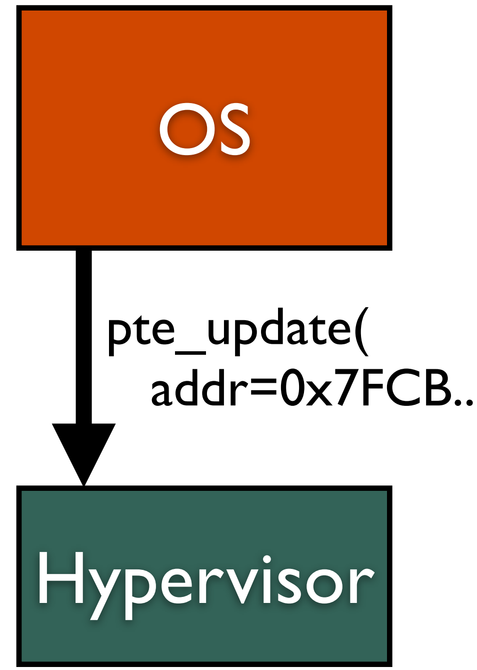


Paraverification: validating PTE updates

- Untrusted OS notifies hypervisor on page table updates
  - Regular structure
  - In update order



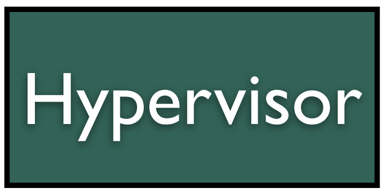
Paraverification: validating PTE updates



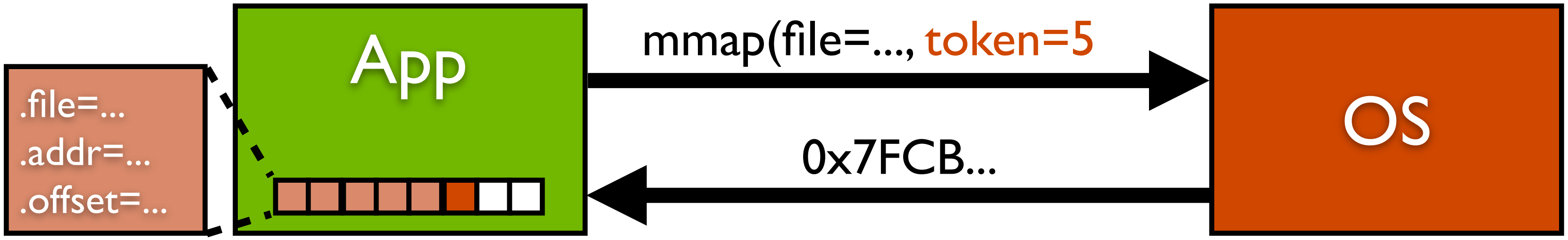
- Untrusted OS notifies hypervisor on page table updates
  - Regular structure
  - In update order

Paraverification: validating PTE updates

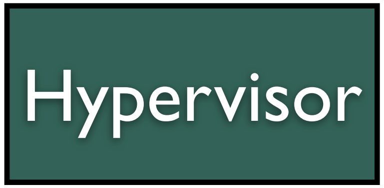
- Application maintains memory mappings in an array of descriptors
  - Interpose on mmap() in libc
- Generate a *token* for each mapping
  - Unforgeable identifier describing requested mapping
  - e.g. HMAC(addr, file, offset)
  - In implementation, integer index



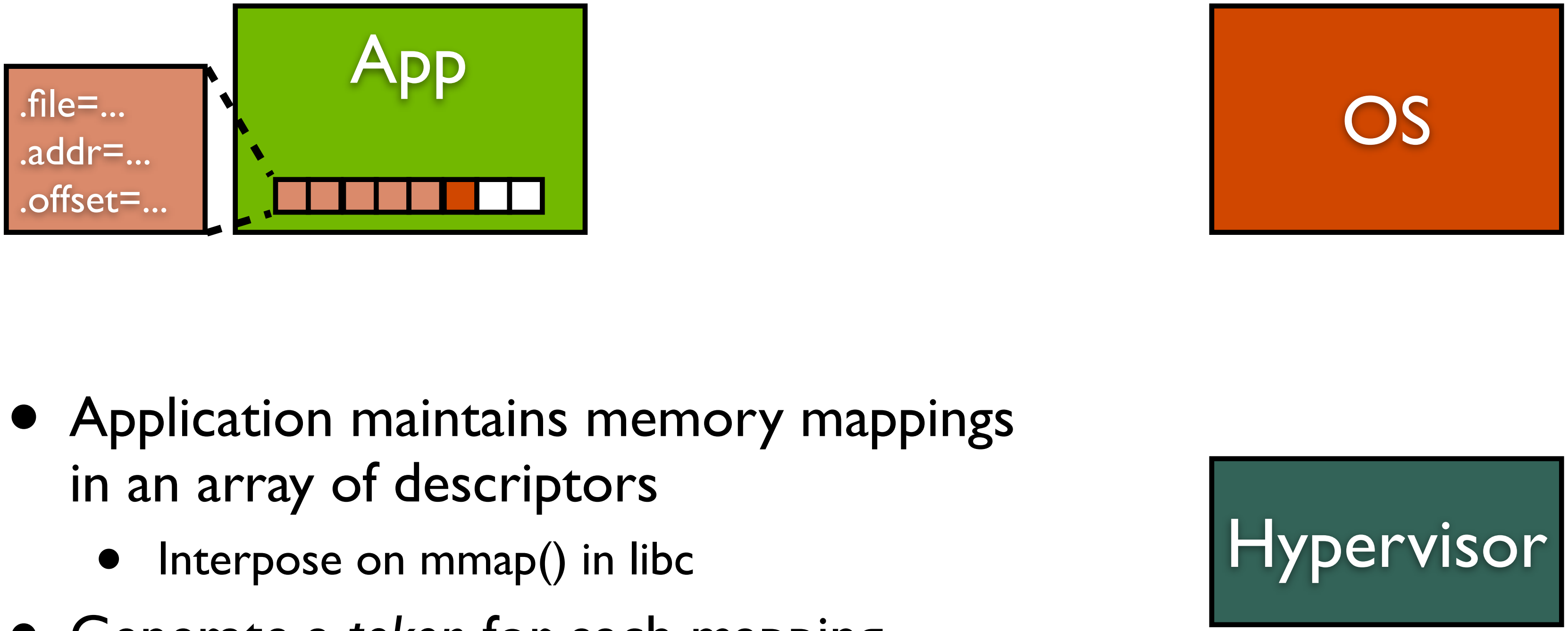
Paraverification: validating PTE updates



- Application maintains memory mappings in an array of descriptors
  - Interpose on mmap() in libc
- Generate a *token* for each mapping
  - Unforgeable identifier describing requested mapping
  - e.g. HMAC(addr, file, offset)
  - In implementation, integer index

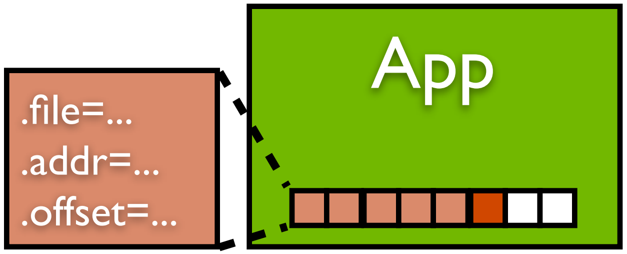


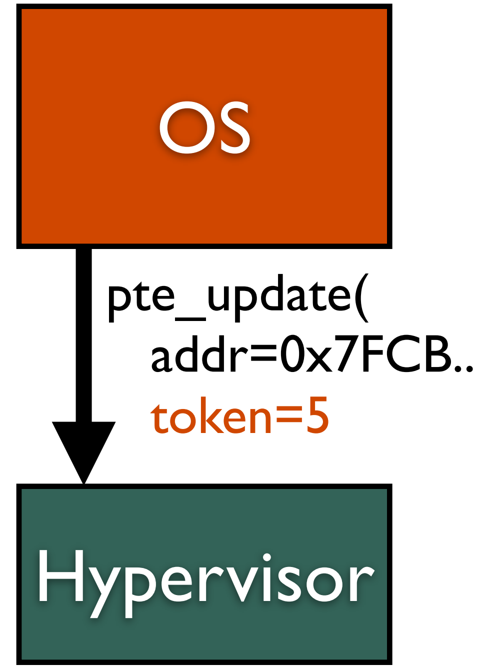
Paraverification: validating PTE updates



- Application maintains memory mappings in an array of descriptors
  - Interpose on mmap() in libc
- Generate a *token* for each mapping
  - Unforgeable identifier describing requested mapping
  - e.g. HMAC(addr, file, offset)
  - In implementation, integer index

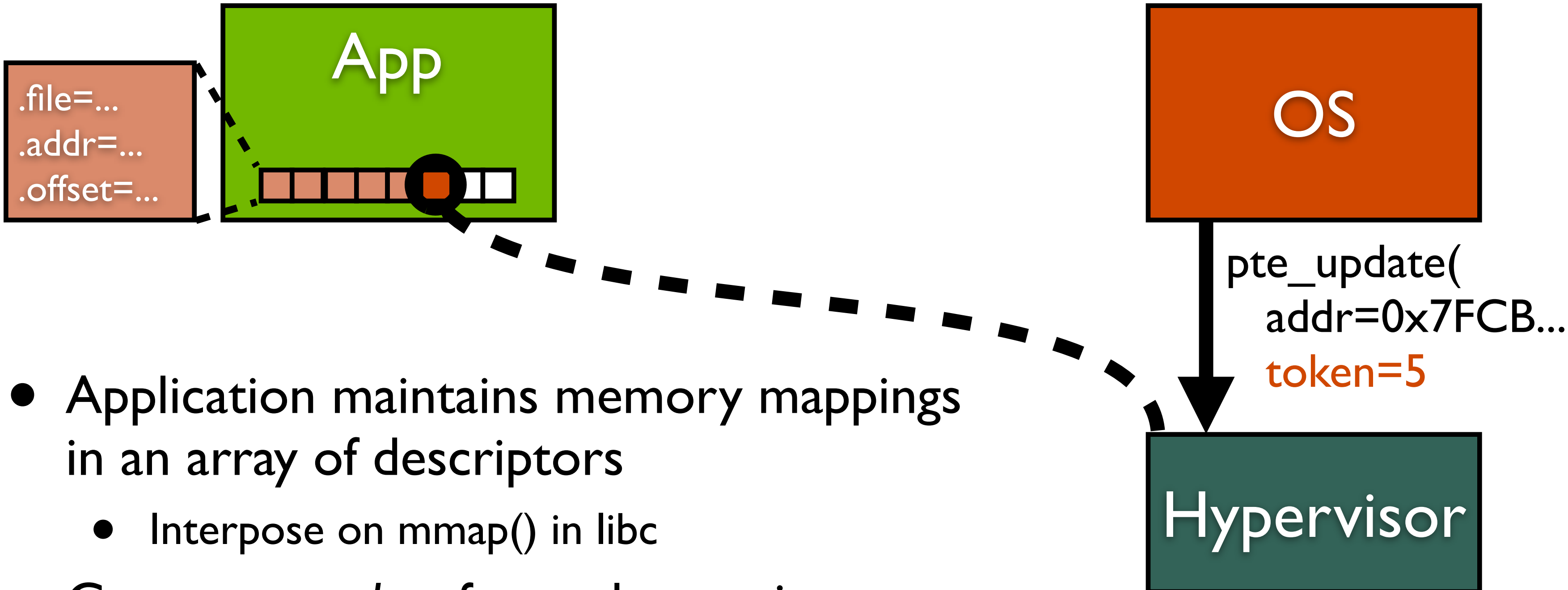
Paraverification: validating PTE updates





- Application maintains memory mappings in an array of descriptors
  - Interpose on mmap() in libc
- Generate a *token* for each mapping
  - Unforgeable identifier describing requested mapping
  - e.g. HMAC(addr, file, offset)
  - In implementation, integer index

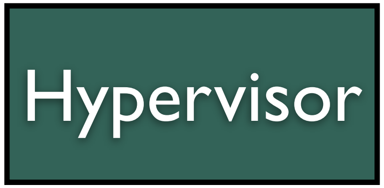
Paraverification: validating PTE updates



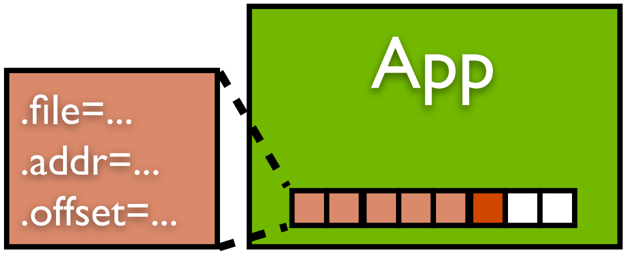
- Application maintains memory mappings in an array of descriptors
  - Interpose on mmap() in libc
- Generate a *token* for each mapping
  - Unforgeable identifier describing requested mapping
  - e.g. HMAC(addr, file, offset)
  - In implementation, integer index

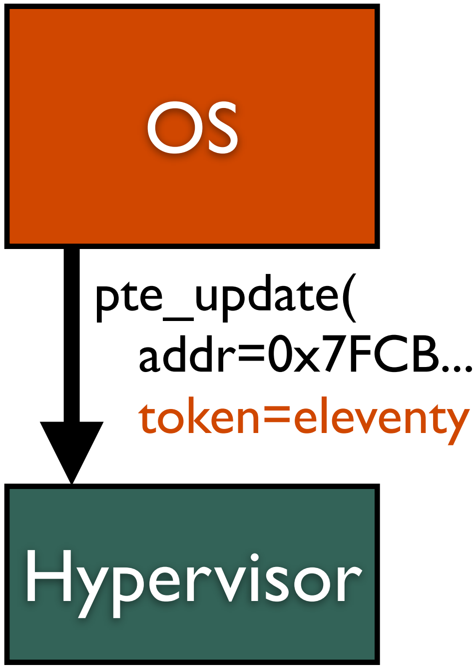
Paraverification: validating PTE updates

- Application memory listing protected from OS
- Entries always allocated in defined virtual address range
- Invalid entries marked



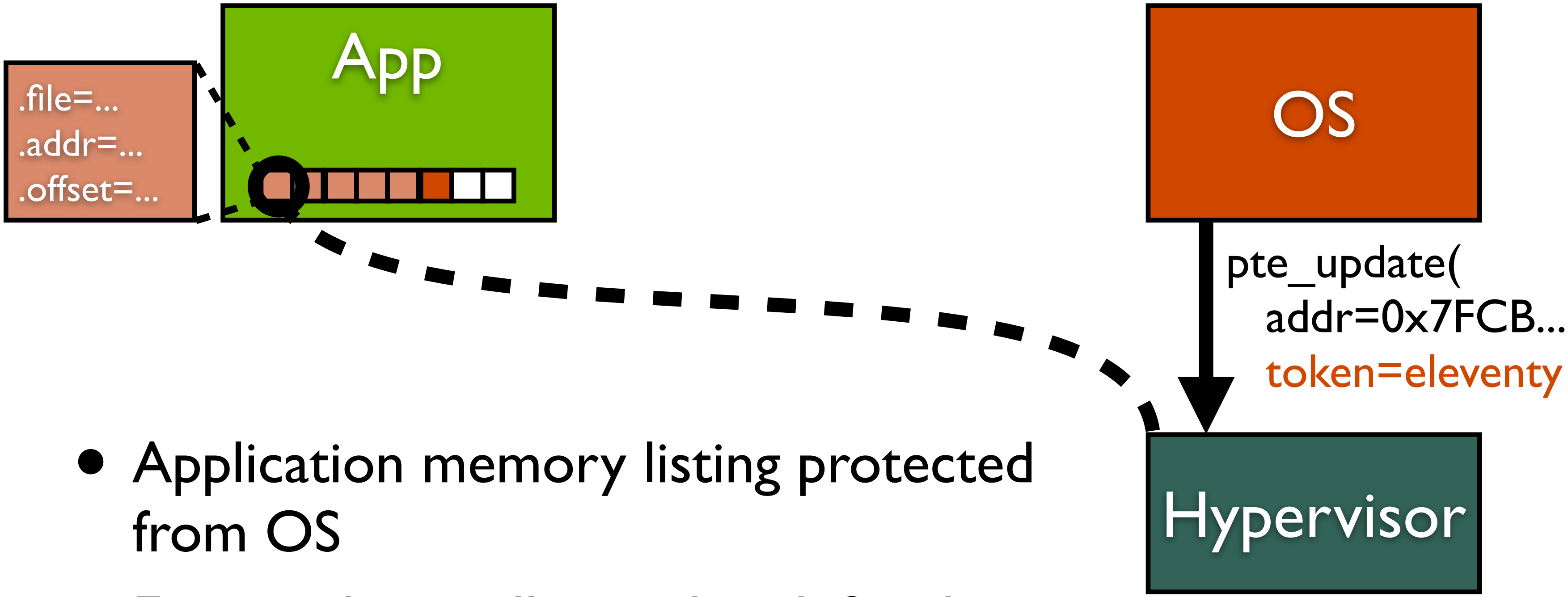
Paraverification: validating PTE updates





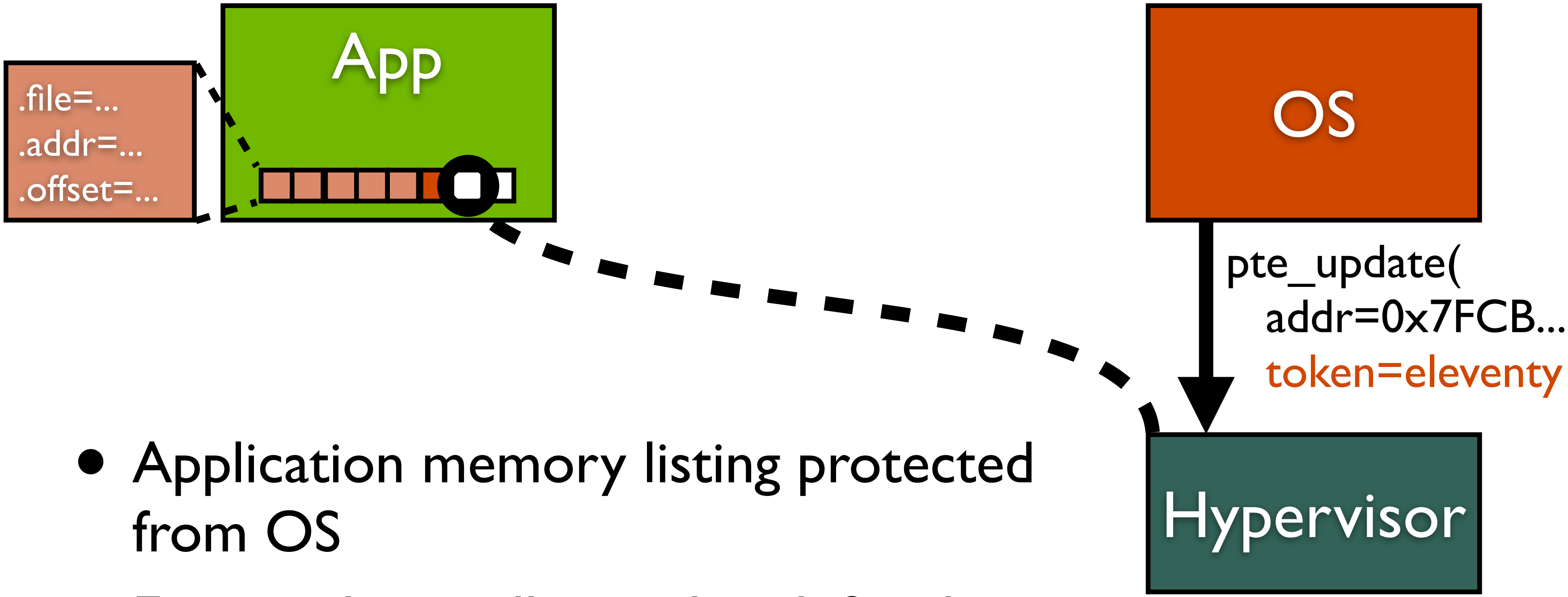
- Application memory listing protected from OS
- Entries always allocated in defined virtual address range
- Invalid entries marked

Paraverification: validating PTE updates



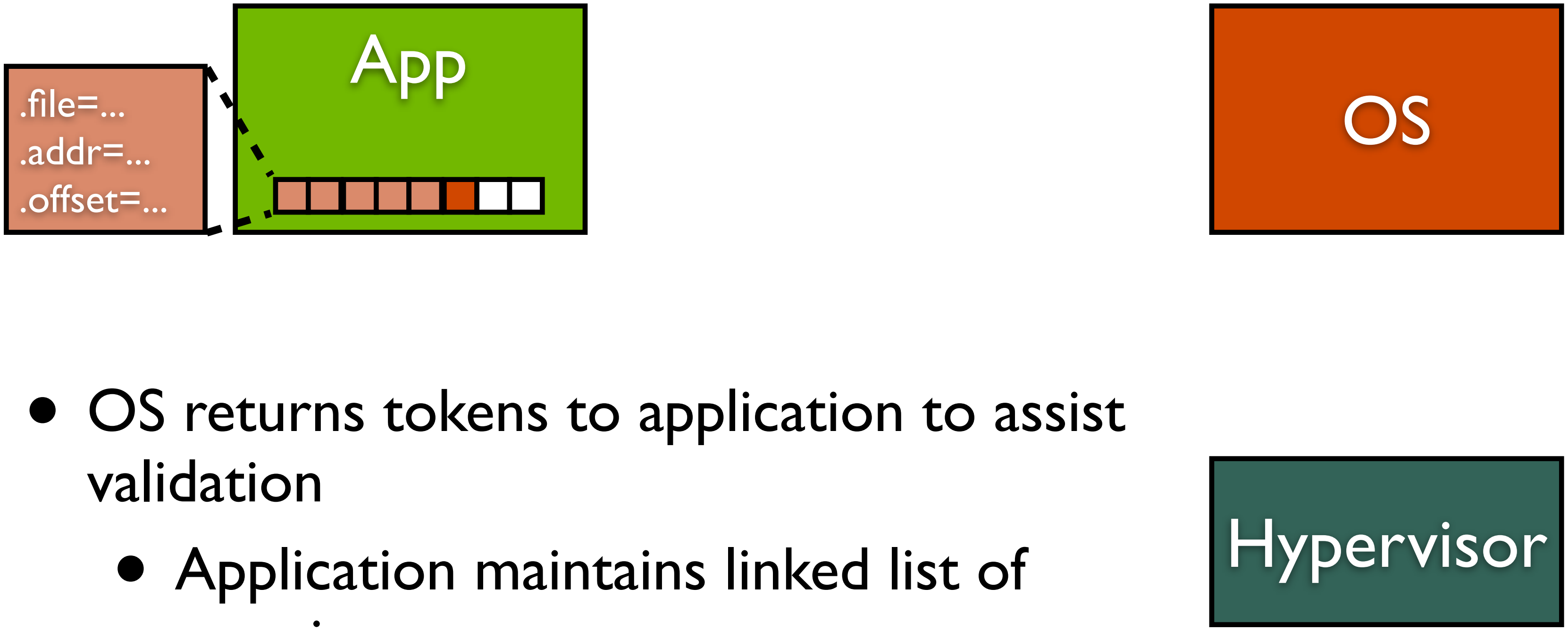
- Application memory listing protected from OS
- Entries always allocated in defined virtual address range
- Invalid entries marked

Paraverification: validating PTE updates



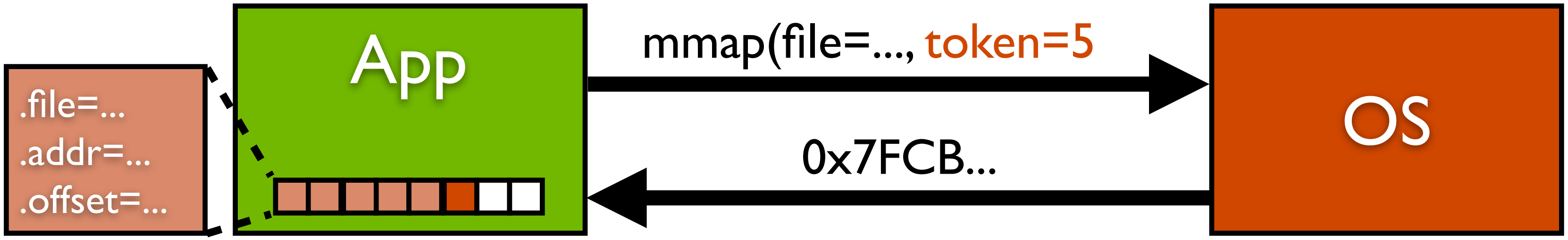
- Application memory listing protected from OS
- Entries always allocated in defined virtual address range
- Invalid entries marked

Paraverification: validating syscall results

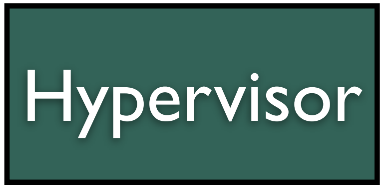


- OS returns tokens to application to assist validation
  - Application maintains linked list of mappings
  - OS specifies previous entry
  - Application checks for overlap, updates list

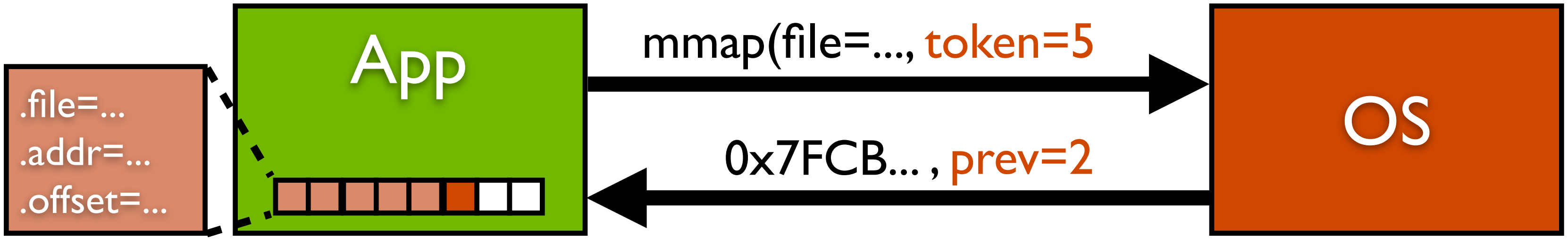
Paraverification: validating syscall results



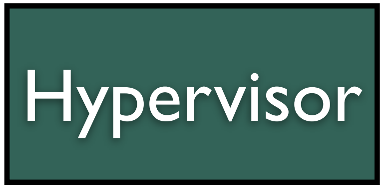
- OS returns tokens to application to assist validation
  - Application maintains linked list of mappings
  - OS specifies previous entry
  - Application checks for overlap, updates list



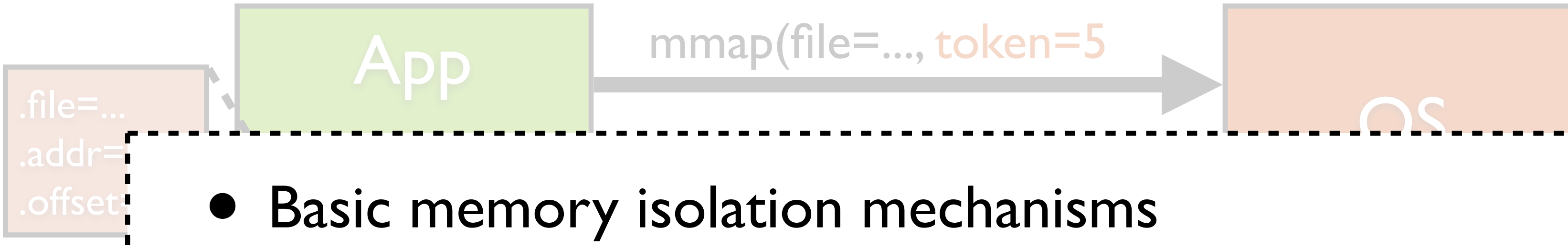
Paraverification: validating syscall results



- OS returns tokens to application to assist validation
  - Application maintains linked list of mappings
  - OS specifies previous entry
  - Application checks for overlap, updates list



Paraverification: validating syscall results



- Basic memory isolation mechanisms
- Challenges: why is this difficult?
- Paraverification: how can the untrusted OS help?
    - Guarantee sane address space updates
    - Expose internal OS information to hypervisor and application

- OS specifies previous entry
- Application checks for overlap, updates list

Implementation & Evaluation

- Prototype built with KVM, qemu, uClibc
  - ~3500 hypervisor LOC
  - Modify libc to validate syscall results
- OS microbenchmarks
  - LMBench
- Applications
  - SPEC
  - Apache
  - DokuWiki

DokuWiki

- PHP CGI binary with InkTag extensions
- InkTag authentication module
  - Use InkTag access control on wiki pages
- Result: hypervisor-enforced security for a PHP application
  - Integrity for all script files
  - Privacy and integrity for application data

InkTag overheads

- LMBench
  - Low-level OS microbenchmarks
  - 5x - 55x slowdown (for μs operations)
  - High context switch latency
- SPEC
  - CPU-bound applications
  - Most applications <= 1.03x
  - gcc - 1.14x; perlbench, h264href - 1.10x
- Apache
  - Long-lived processes, infrequent MM activity
  - 1.02x throughput slowdown, 1.13x latency
- DokuWiki
  - Many short-lived processes, frequent memory mapping
  - 1.54x throughput slowdown

Related work

- Untrusted operating systems
  - XOMOS [Lie et al. SOSP '03]
  - Overshadow [Chen et al. ASPLOS '08]
  - SP^3 [Yang & Shin VEE '08]
  - Cloudvisor [Zhang et al. SOSP '11]

Conclusion

- We can enforce trustworthy services from an untrustworthy OS
- Paraverification simplifies crucial isolation mechanisms