# **DAWG**: A Defense Against Cache Timing Attacks in Speculative Execution Processors

Vladimir Kiriansky, Ilia Lebedev,
Saman Amarasinghe, Srinivas Devadas, Joel Emer
{vlk, ilebedev, saman, devadas, emer}@csail.mit.edu

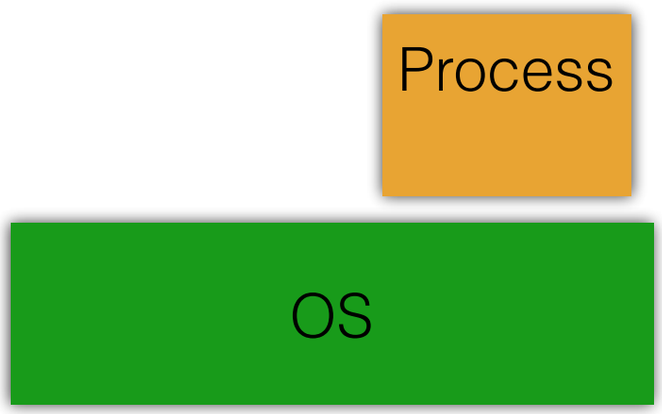
MICRO'18

October 24, 2018
Fukuoka, Japan

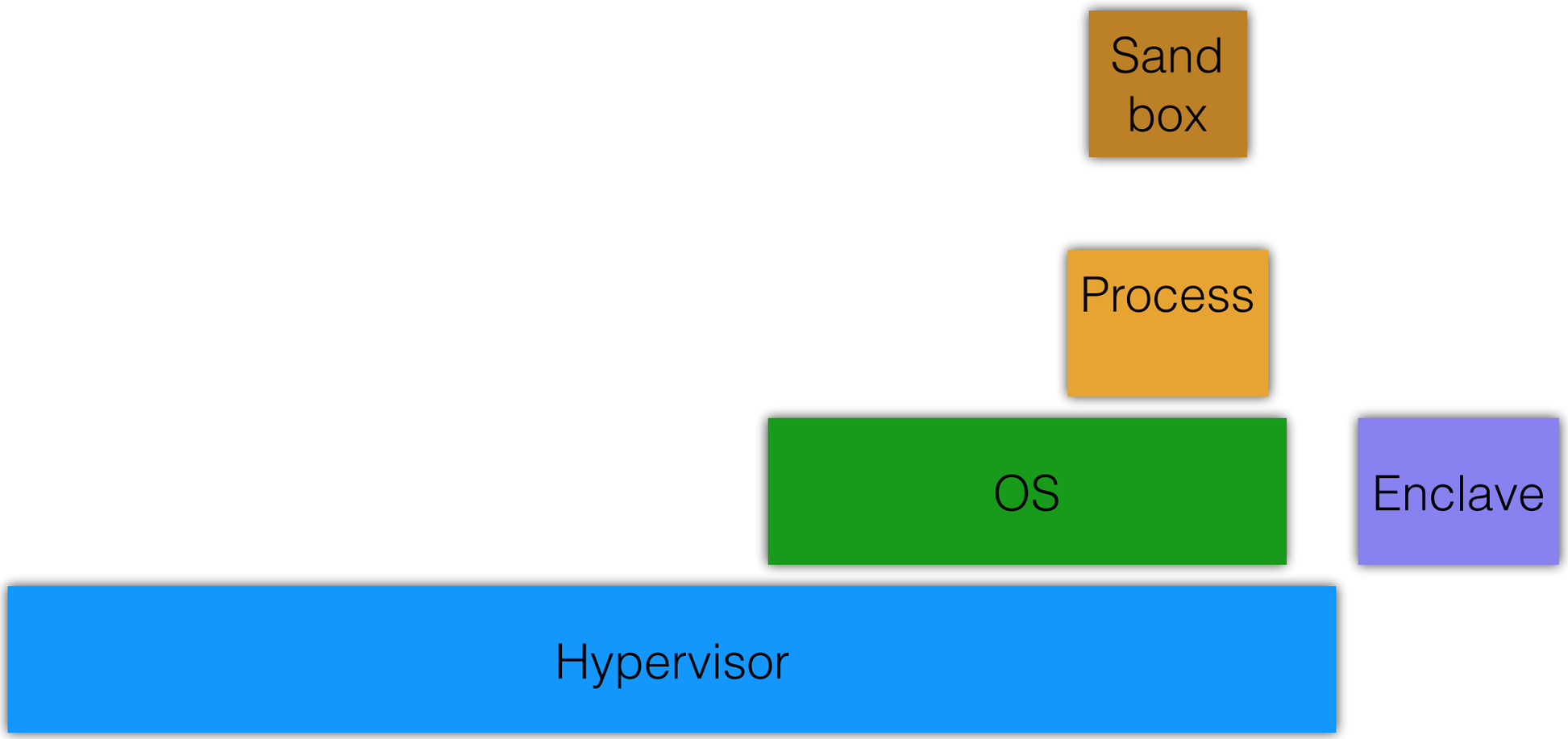
# Outline

- Cache access timing attacks
- DAWG protection mechanism: Cache, Core
- OS support: System Calls, Resource Management
- Performance and security evaluation
- Conclusion & Q/A

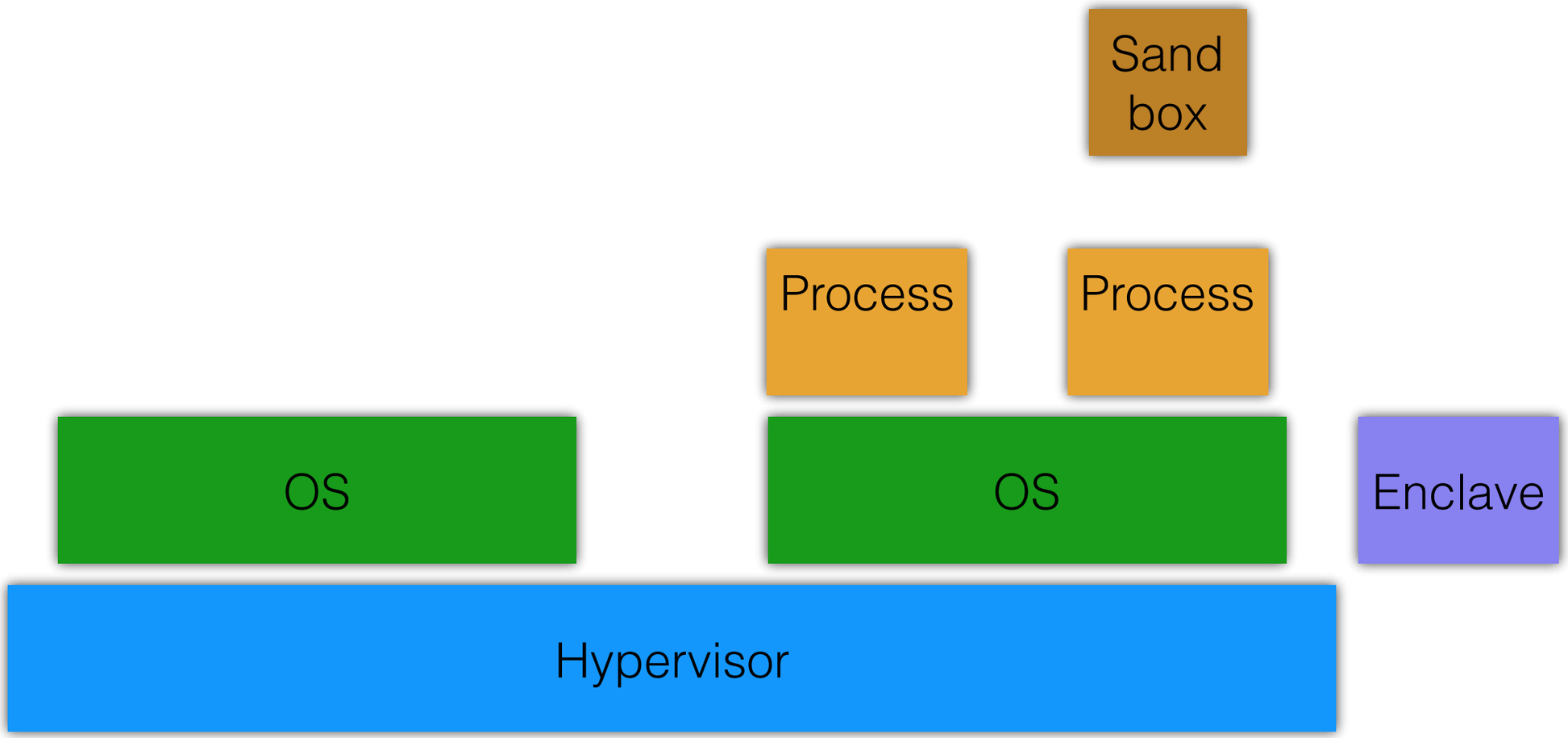
# Trust Boundaries



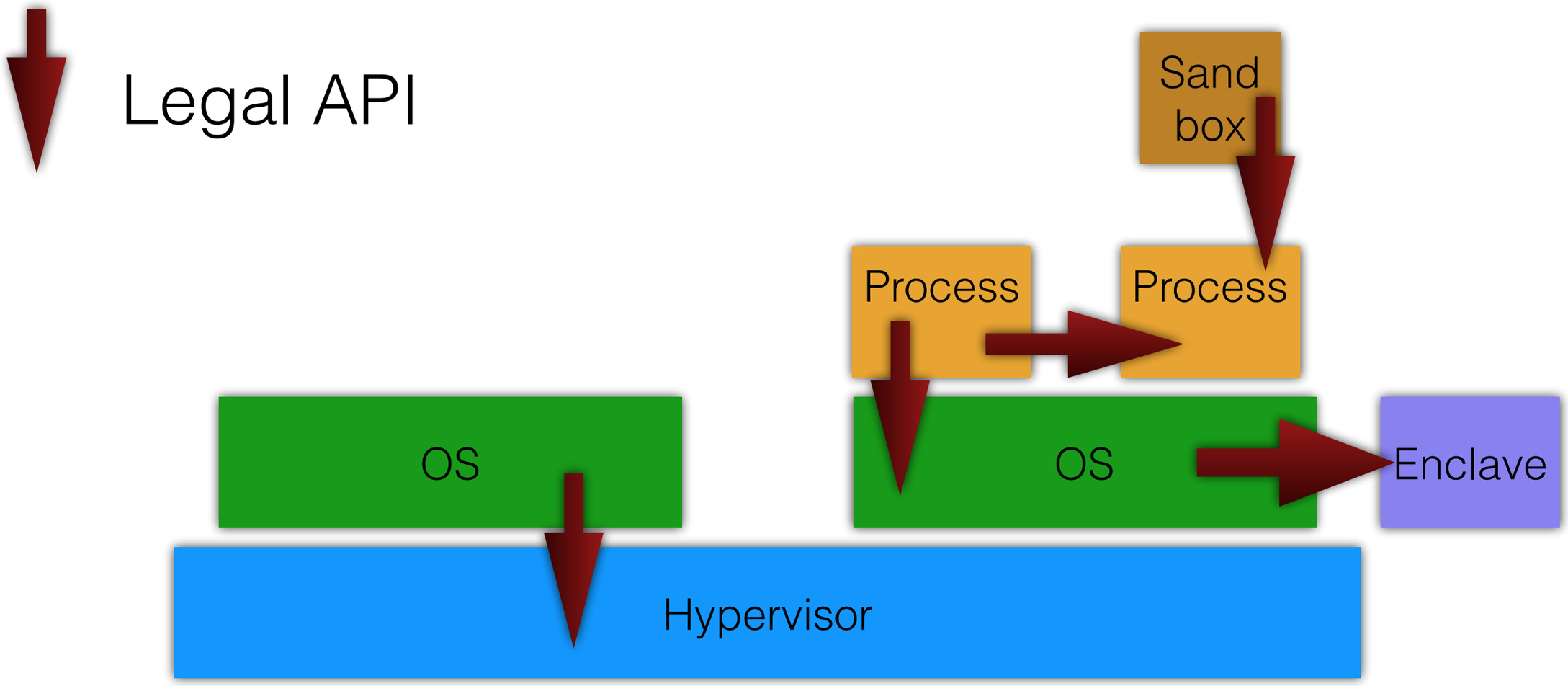
# Trust Boundaries



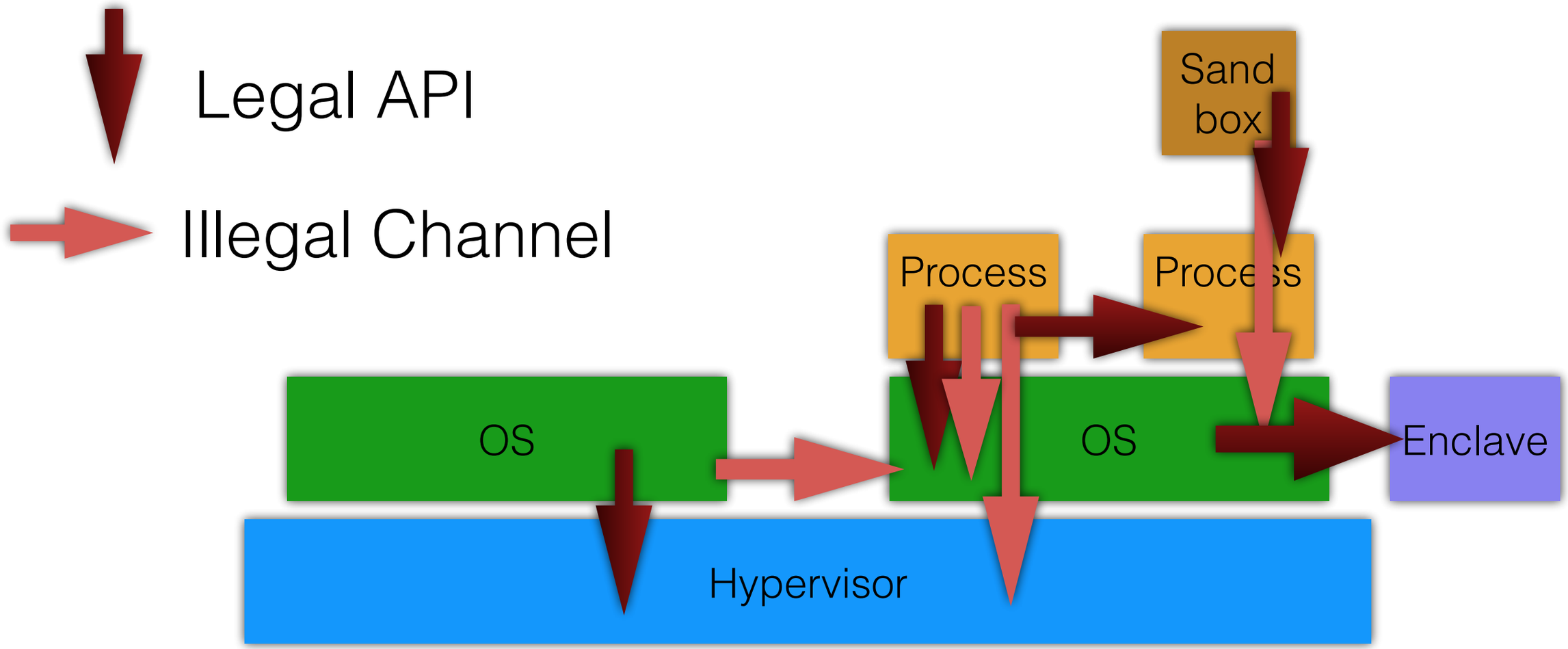
# Trust Boundaries



# Trust Boundary Crossing APIs / Attack Vectors



# Trust Boundary Crossing APIs / Attack Vectors



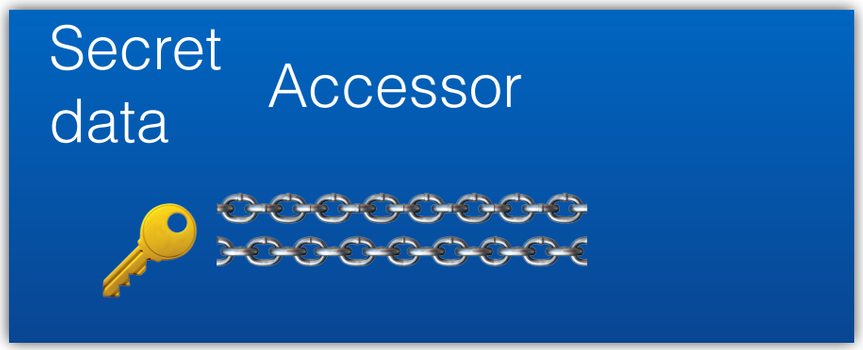
# Side Channels and Covert Channels

Victim's
Protection Domain


Attacker's
Protection Domain

# Side Channels and Covert Channels

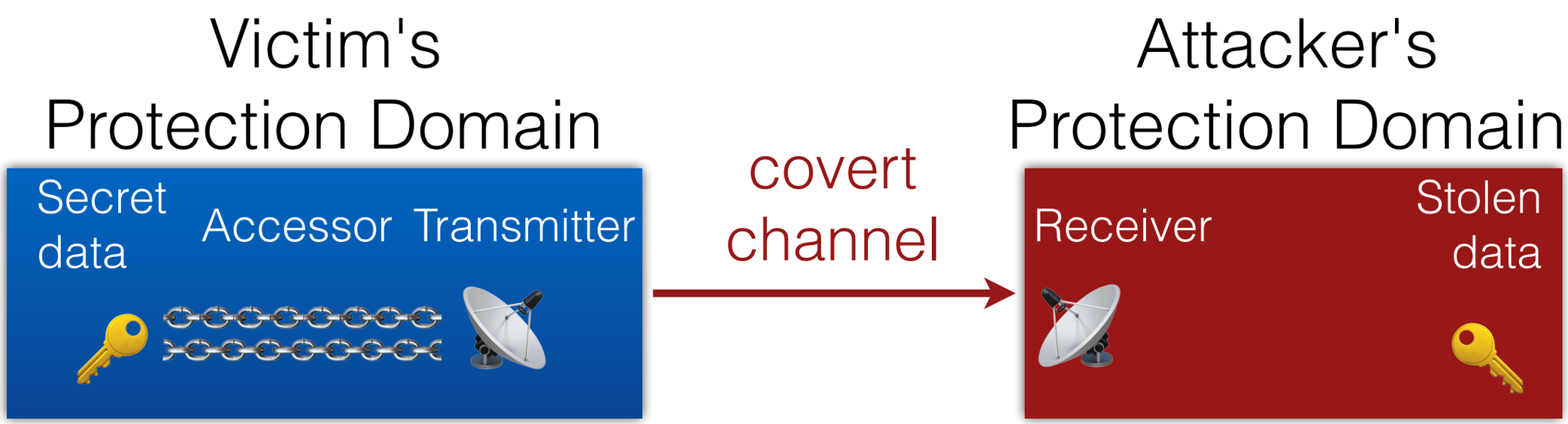
Victim's Protection Domain




Attacker's Protection Domain

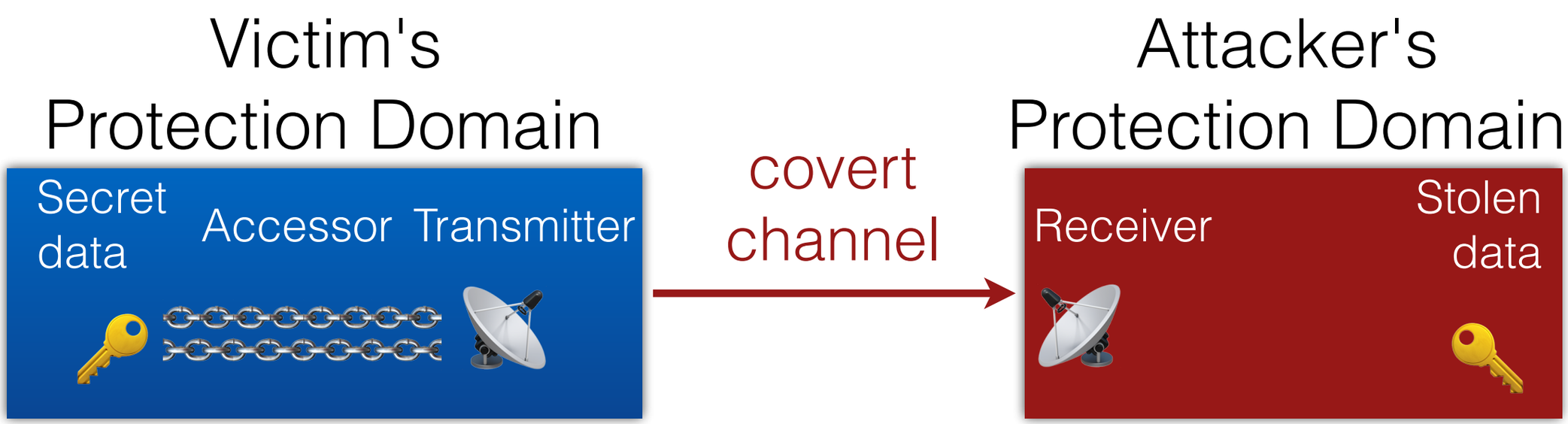
- Accessor
  - Existing code - non-speculative, traditional
  - Synthesized - Spectre 1.0, 1.1 - unresolved

# Side Channels and Covert Channels




- Accessor
  - Existing code - non-speculative, traditional
  - Synthesized - Spectre 1.0, 1.1 - unresolved

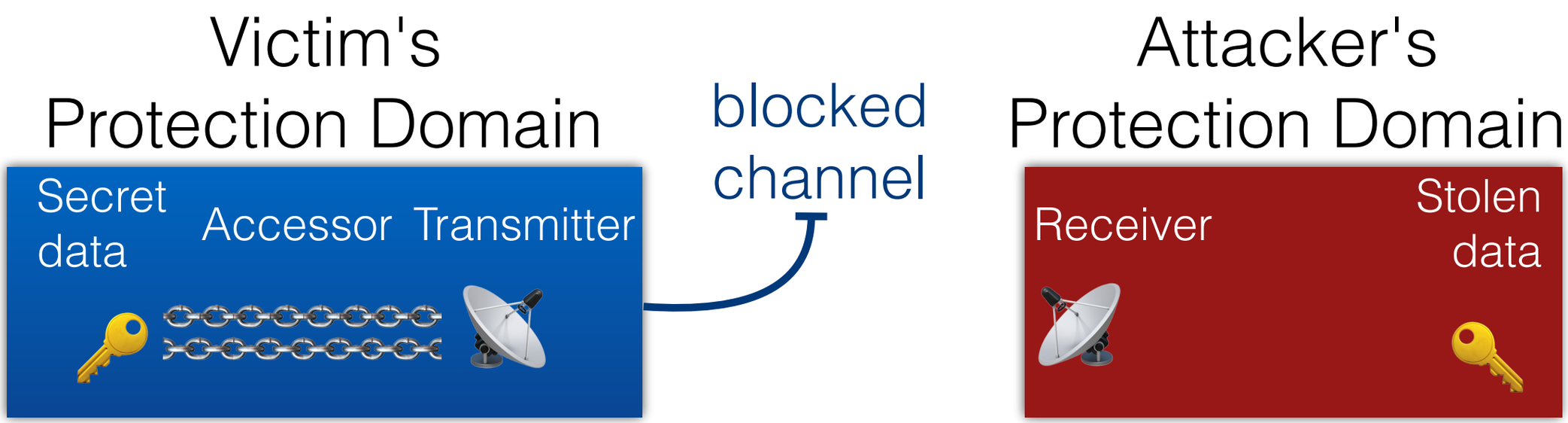
# Side Channels and Covert Channels




- Accessor
  - Existing code - non-speculative, traditional
  - Synthesized - Spectre 1.0, 1.1 - unresolved

- Channel = micro-architectural state:
  cache, TLB, branch predictor state, etc.

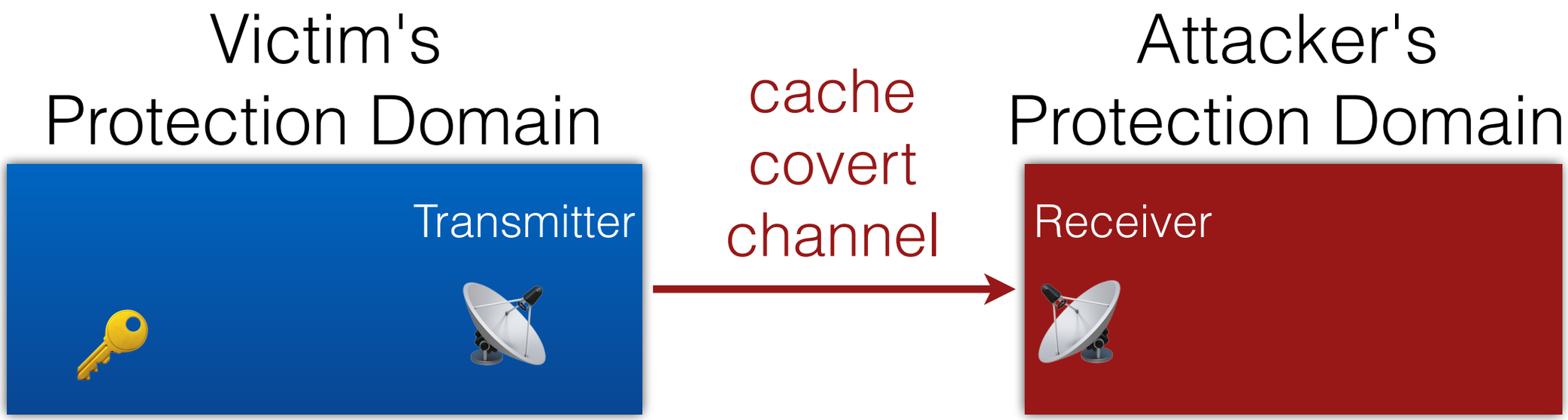
# Side Channels and Covert Channels




- Accessor
  - Existing code - non-speculative, traditional
  - Synthesized - Spectre 1.0, 1.1 - unresolved

- Channel = micro-architectural state:
  **cache**, TLB, branch predictor state, etc.

# Cache Covert Channel



# Cache Covert Channel: Shared Cache Ways

2-way
Cache Set

# Cache Covert Channel: Shared Cache Ways

[Flush+Reload, Evict+Reload, Thrash+Reload]

1. Receiver evicts block A
   Flush / Evict / Thrash


2-way
Cache Set

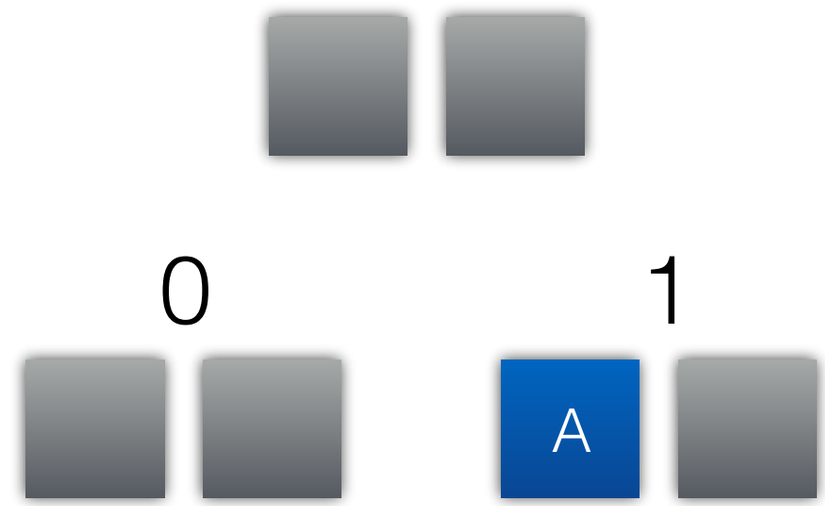
# Cache Covert Channel: Shared Cache Ways

1. Receiver evicts block A
   Flush / Evict / Thrash

2-way
Cache Set

# Cache Covert Channel: Shared Cache Ways

1. Receiver evicts block A
   Flush / Evict / Thrash
2. Transmitter sends a 0 or 1 secret bit via access to A

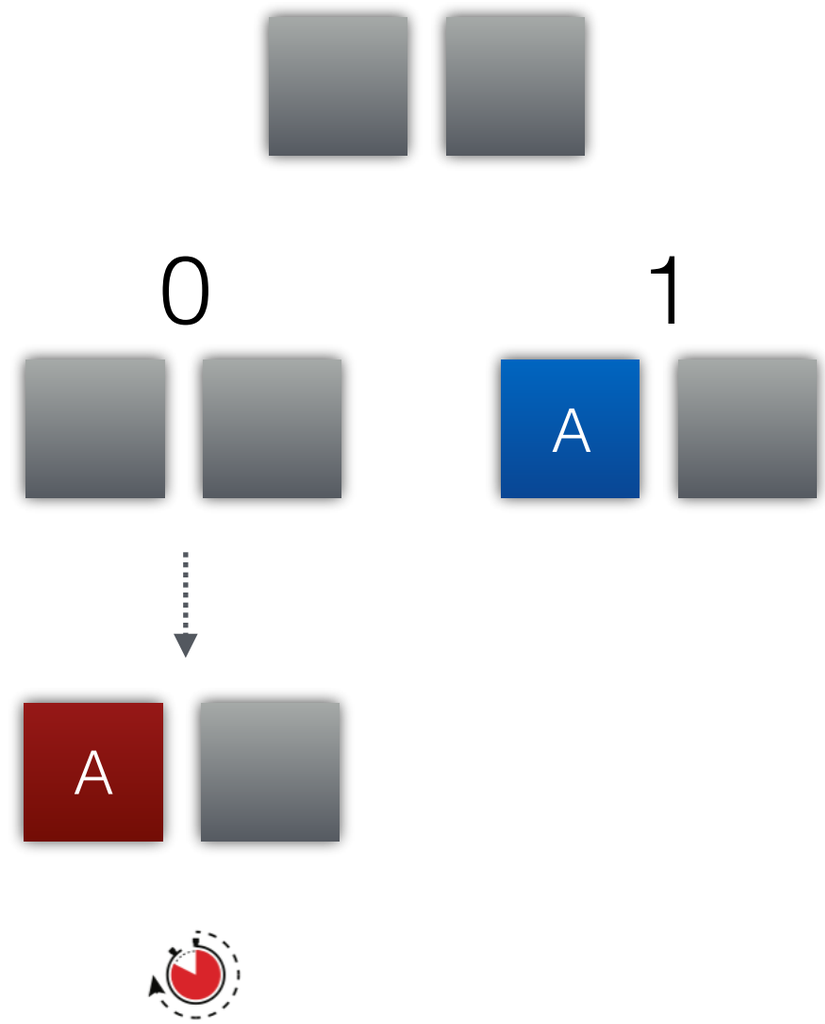


# Cache Covert Channel: Shared Cache Ways

1. Receiver evicts block A
   Flush / Evict / Thrash

2. Transmitter sends a 0 or 1 secret bit via access to A

3. Receiver times access to A



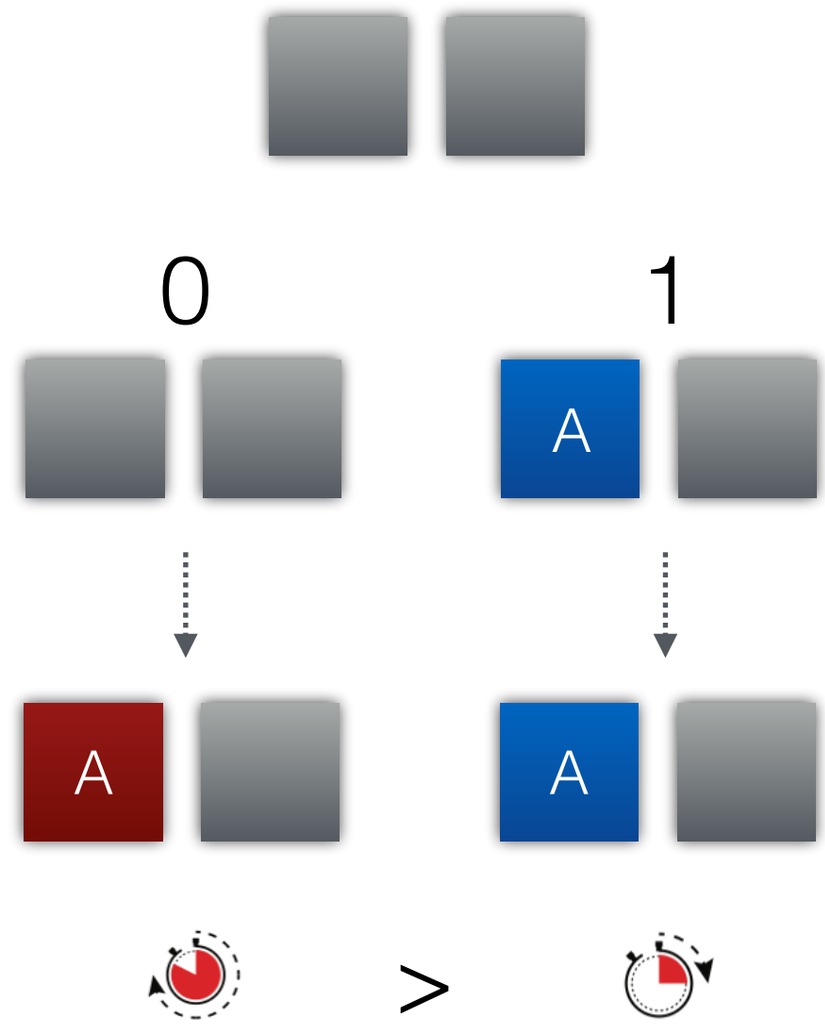
# Cache Covert Channel: Shared Cache Ways

1. Receiver evicts block A
   Flush / Evict / Thrash

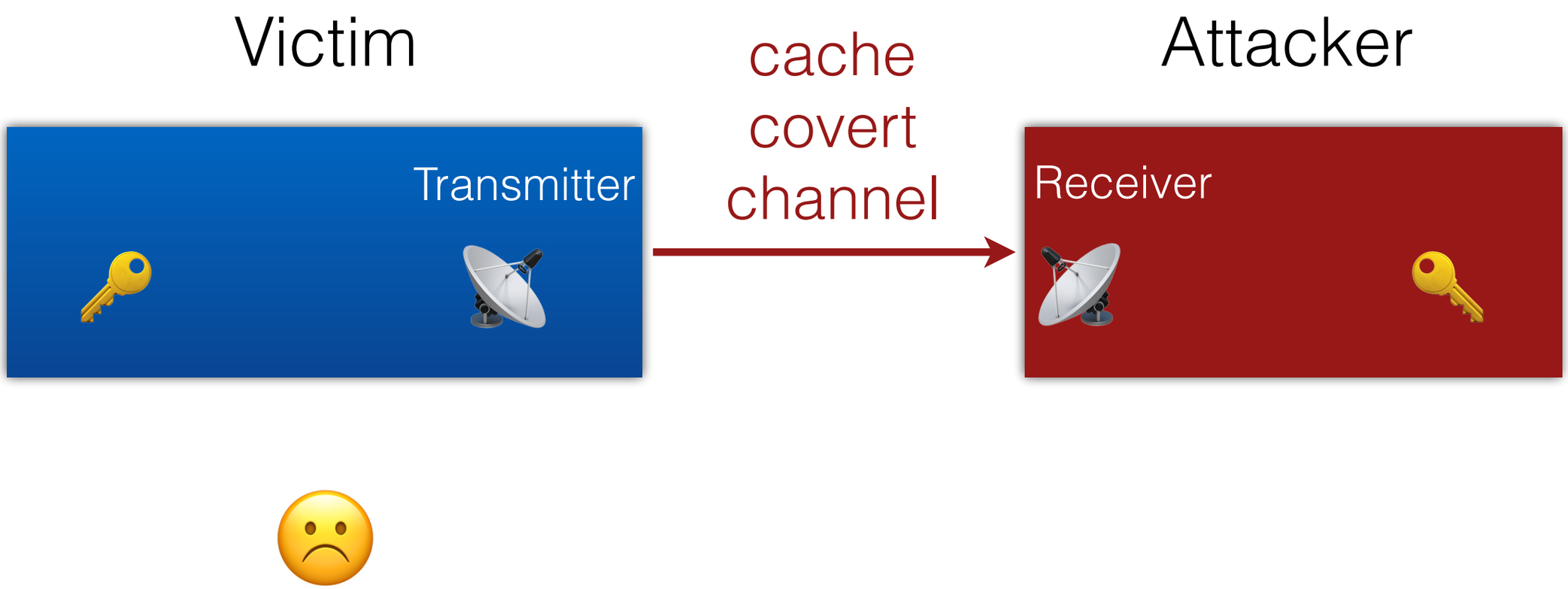
2. Transmitter sends a 0 or 1 secret bit via access to A

3. Receiver times access to A

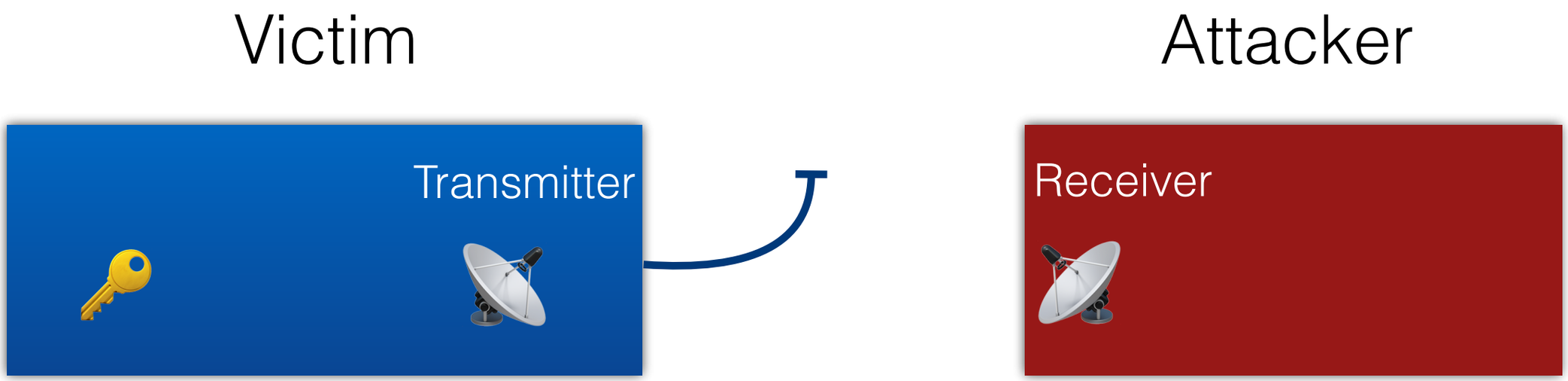
infers secret bit 🔑



# Cache Covert Channel

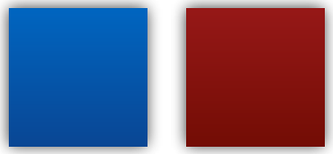


# Block
# Cache Covert Channel?



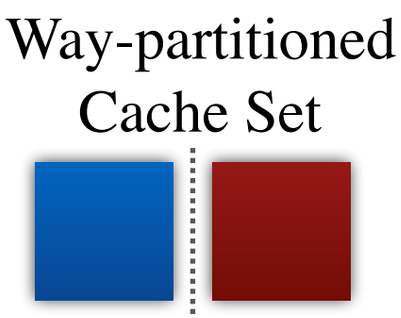
# DAWG: Dynamically Allocated Way Guard

- Cache Protection Domains
- Non-interference by any action:
  hit / flush / eviction / fill



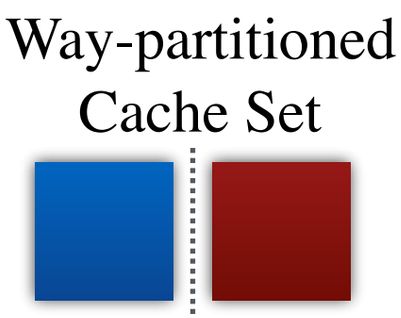
# DAWG: Dynamically Allocated Way Guard

- Cache Protection Domains
- Non-interference by any action:
  hit / flush / eviction / fill
- Partitioned ways of set-associative structures
  - Domain-private cache tag state

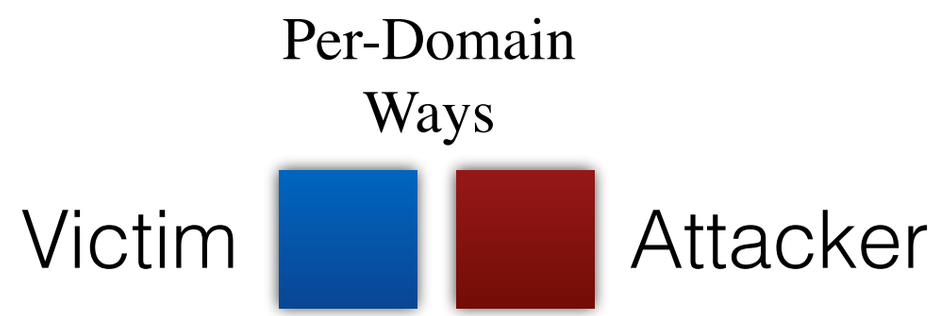


# DAWG: Dynamically Allocated Way Guard

- Cache Protection Domains
- Non-interference by any action:
  hit / flush / eviction / fill
- Partitioned ways of set-associative structures
  - Domain-private cache tag state
  - Domain-private replacement metadata

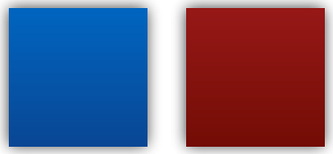


# No Cache Covert Channel: Private Cache Ways



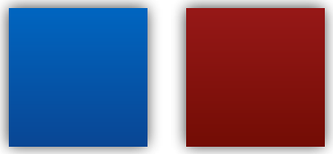
# No Cache Covert Channel: Private Cache Ways

1. Receiver evicts block A?
   Flush / Evict / Thrash



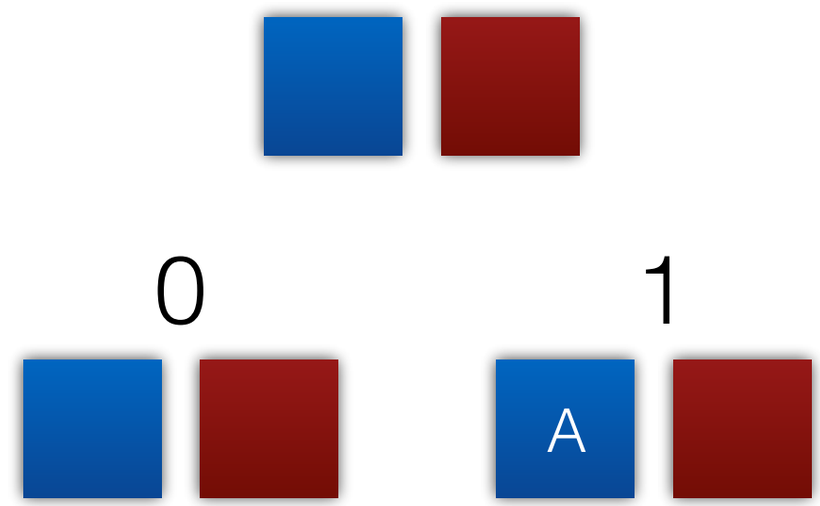
# No Cache Covert Channel: Private Cache Ways

1. Receiver evicts block A
   ~~Flush~~ / ~~Evict~~ / Thrash



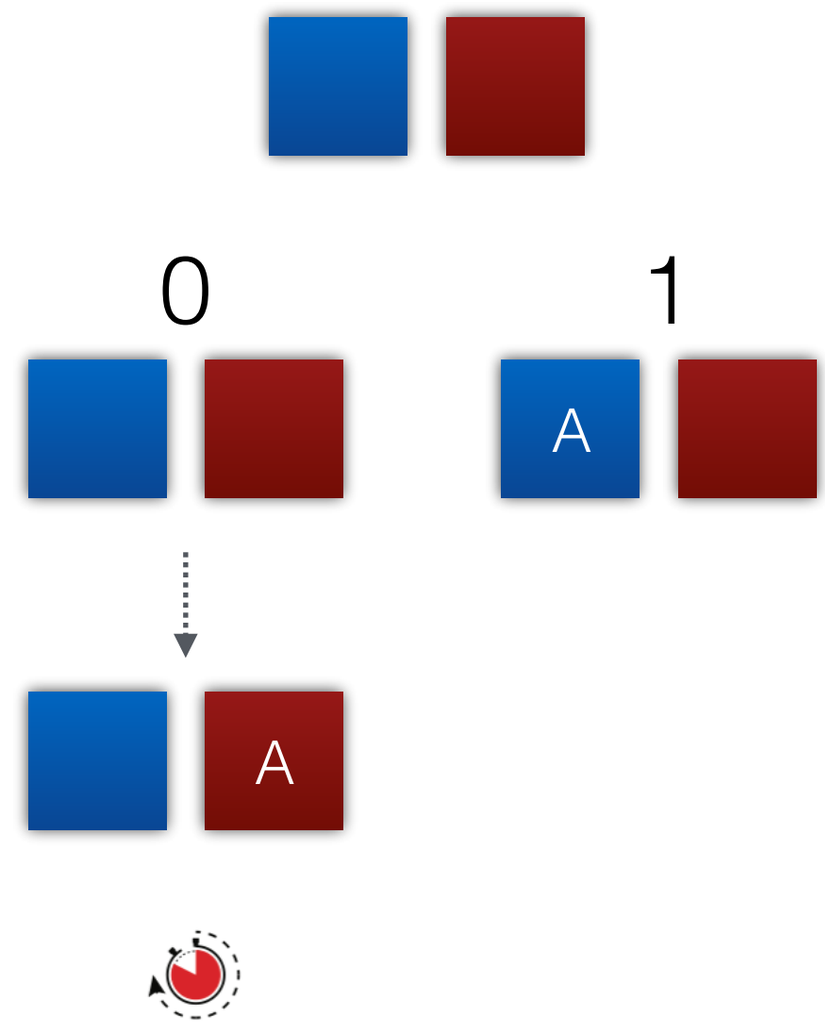
# No Cache Covert Channel: Private Cache Ways

1. Receiver evicts block A
   ~~Flush~~ / ~~Evict~~ / Thrash
2. Transmitter sends a 0 or 1 secret bit via access to A



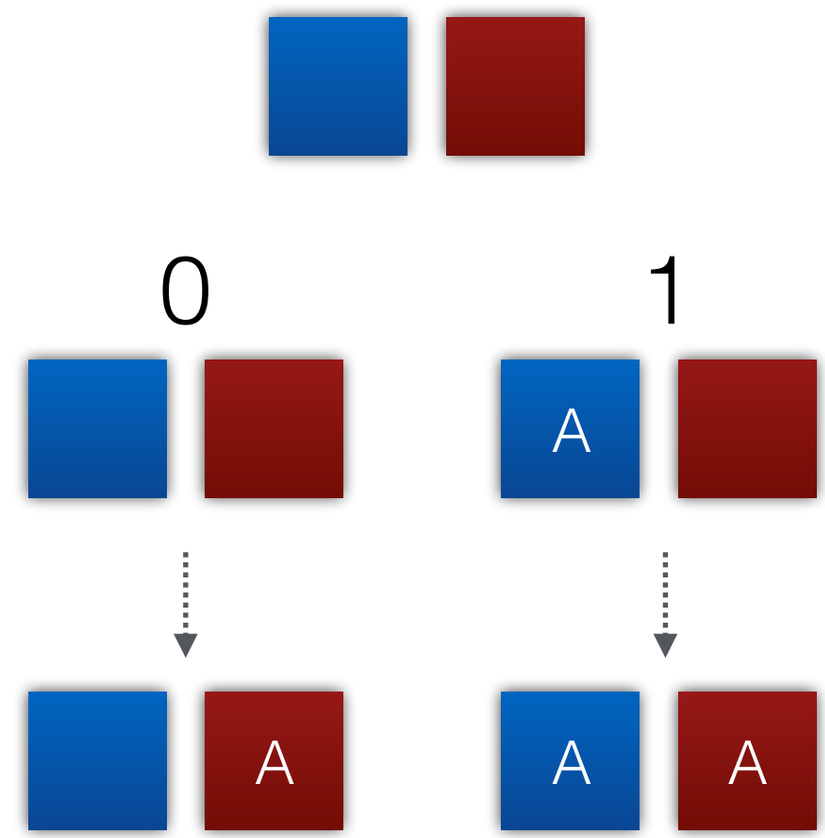
# No Cache Covert Channel: Private Cache Ways

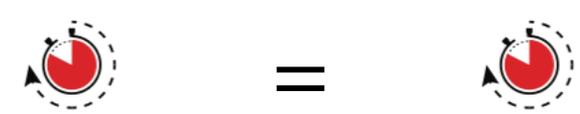
1. Receiver evicts block A
   ~~Flush~~ / ~~Evict~~ / Thrash
2. Transmitter sends a 0 or 1 secret bit via access to A
3. Receiver times access to A



# No Cache Covert Channel: Private Cache Ways

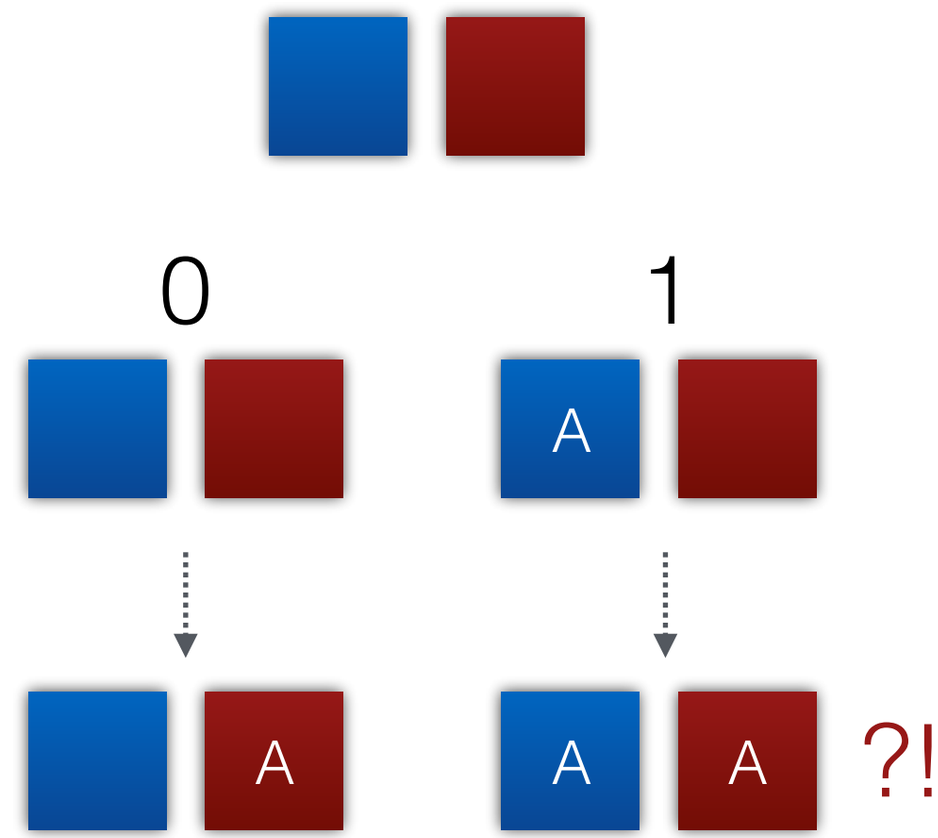
1. Receiver evicts block A
   ~~Flush~~ / ~~Evict~~ / Thrash
2. Transmitter sends a 0 or 1 secret bit via access to A
3. Receiver times access to A

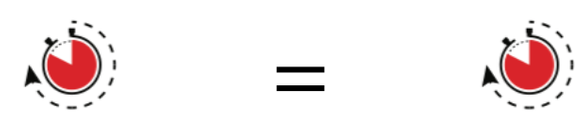



no signal  =

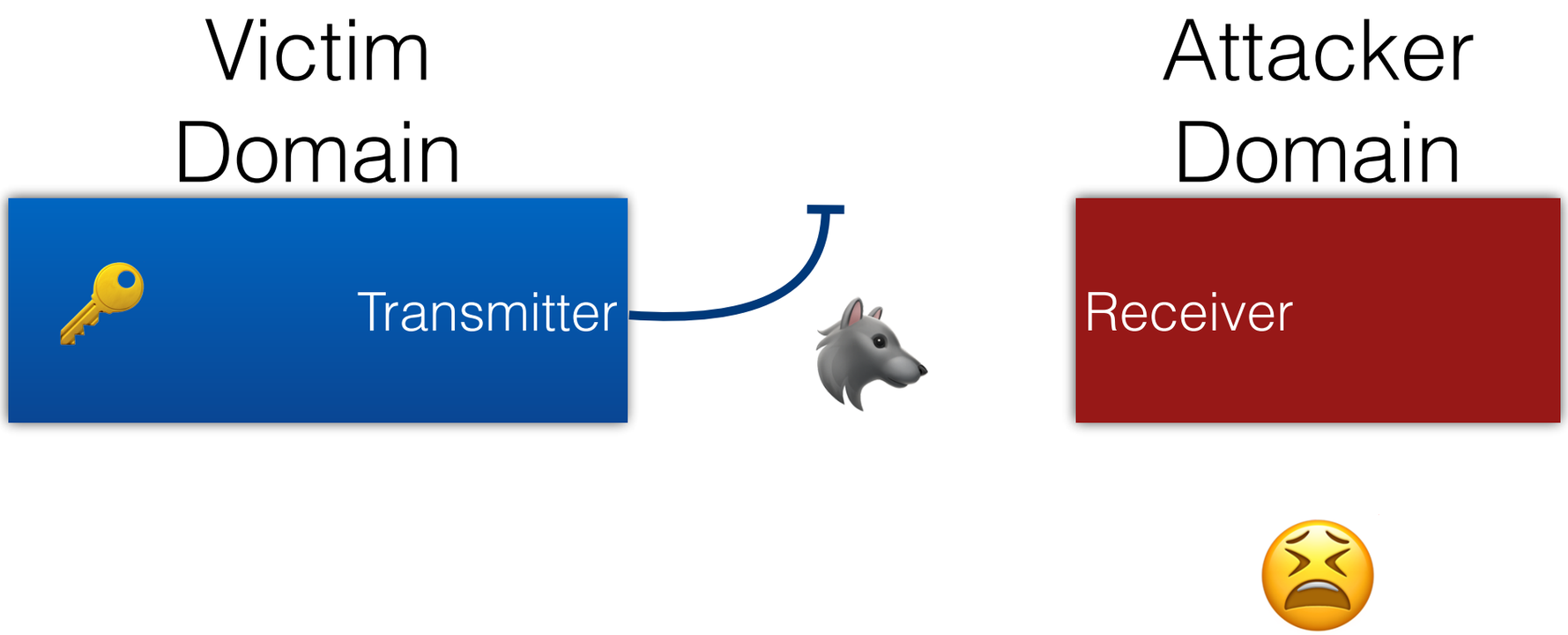
# No Cache Covert Channel: Private Cache Ways

1. Receiver evicts block A
   ~~Flush~~ / ~~Evict~~ / Thrash
2. Transmitter sends a 0 or 1 secret bit via access to A
3. Receiver times access to A




no signal  =

# No Cache Covert Channel



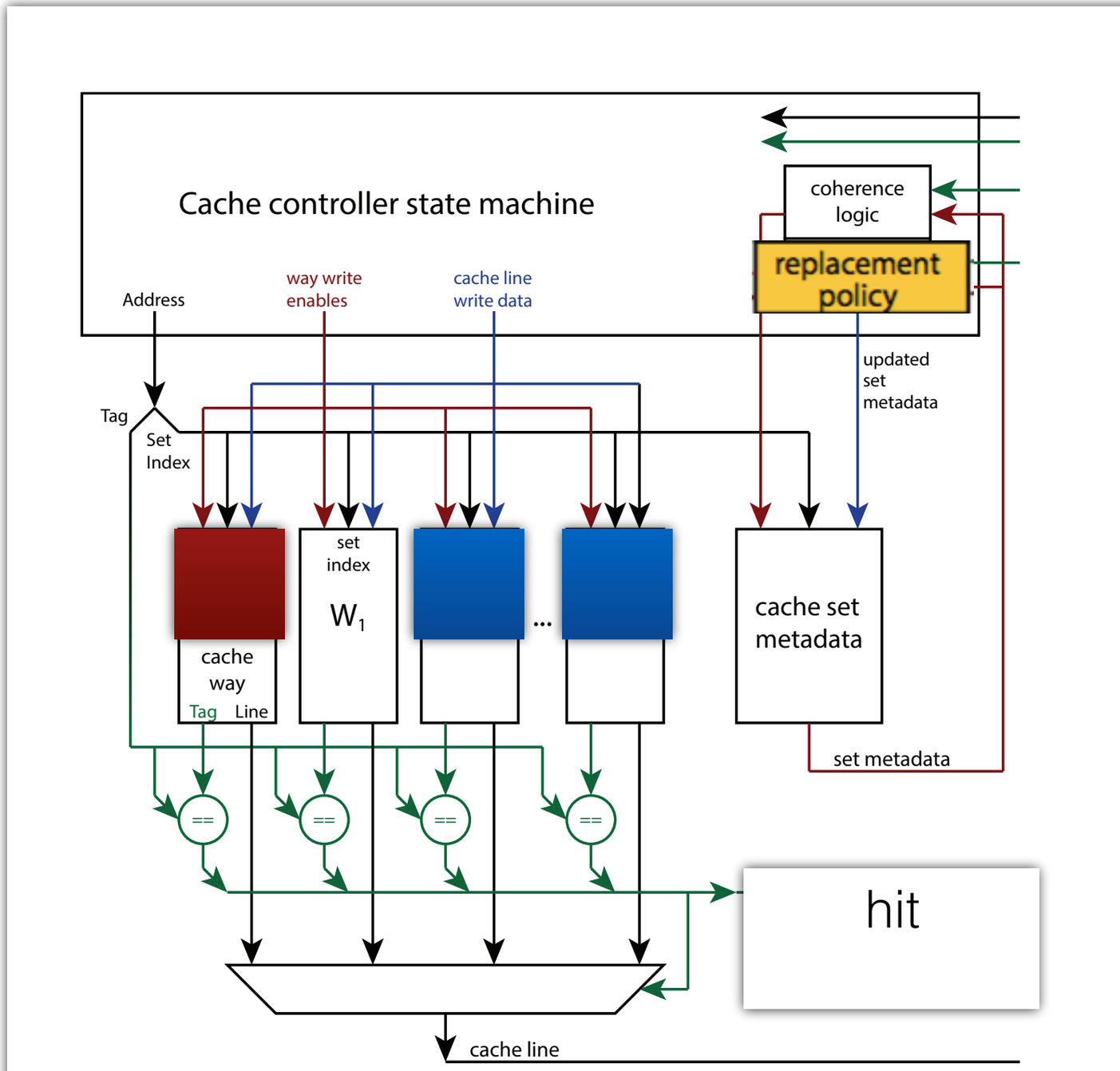
# CAT: QoS Cache Partitioning

- Starting point in production silicon:
  Intel's Cache Allocation Technology for LLC
- Iyer et al [SC'04, SIGMETRICS'07, **MICRO'07**]
  From concept to reality in Haswell [HPCA'16]
- Not a security barrier

Quality of Service goal: prevent one application from dominating the cache

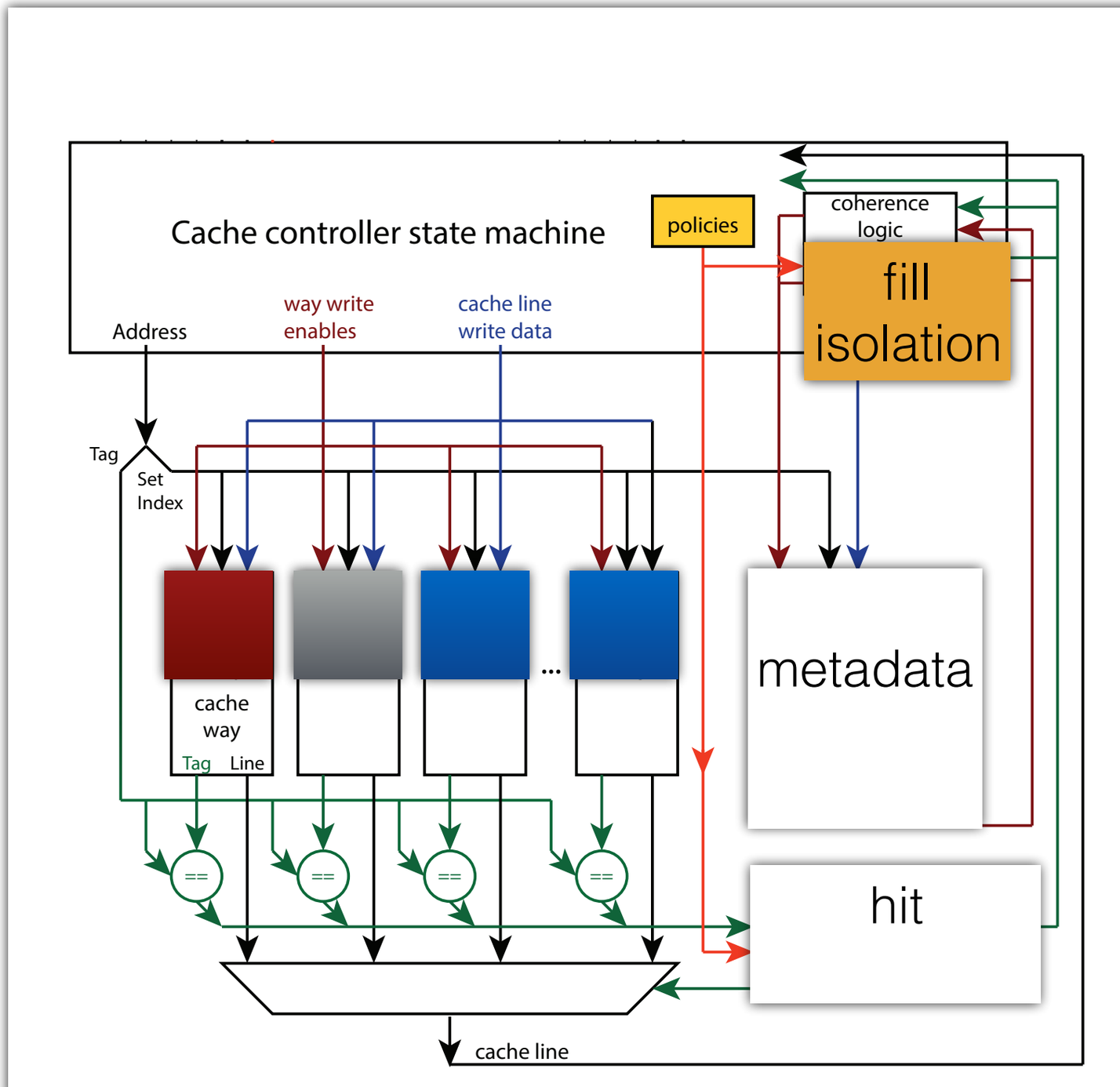
# CAT: Way-Partitioned Set-associative Caches

- Way-partitioning LLC
- Protection domain IDs
  - Fill mask



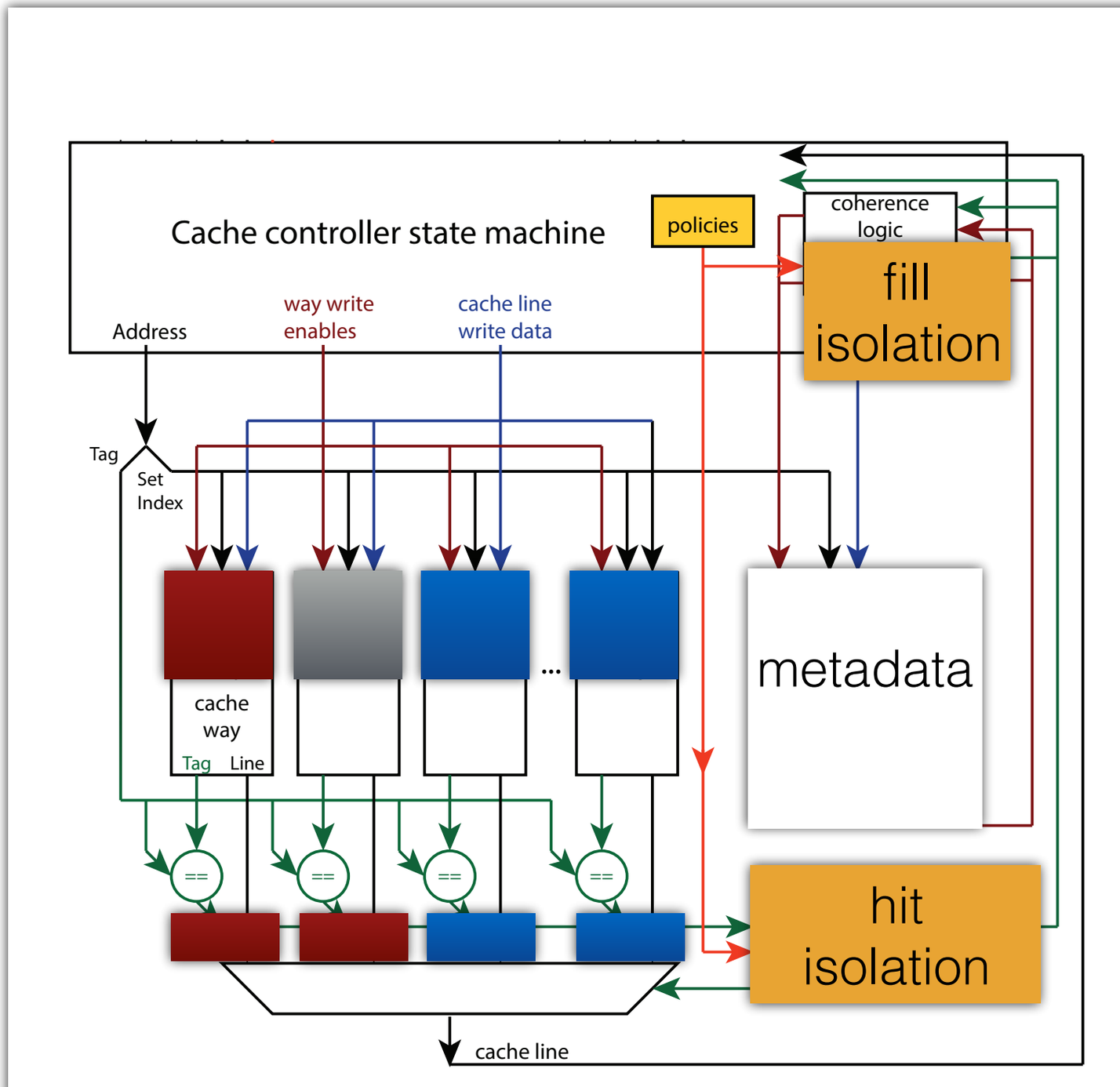
# DAWG: Dynamically Allocated Way Guard

- Way-partitioning **L1**-L3
- Protection domain IDs
  - Fill mask



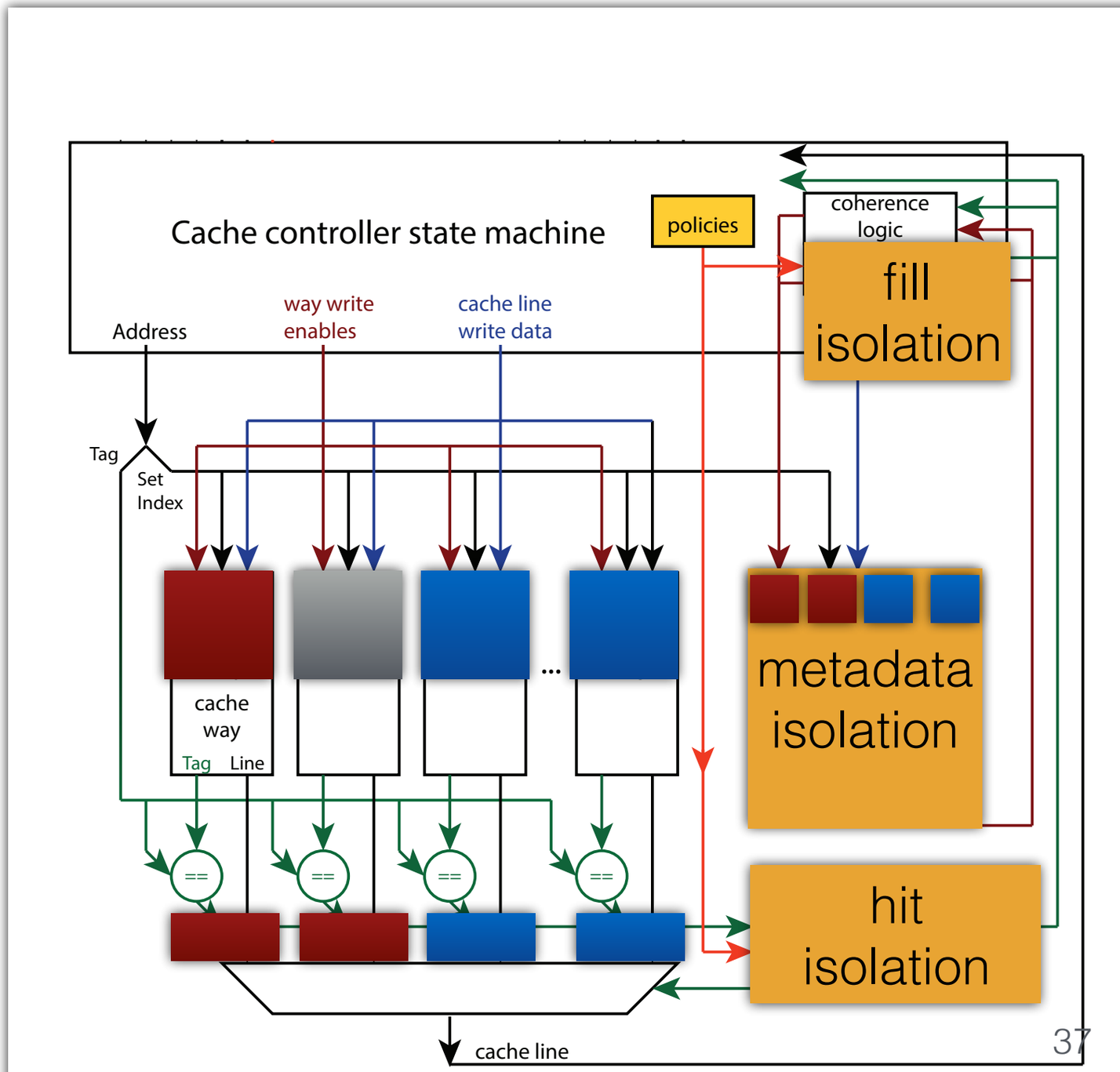
# DAWG: Dynamically Allocated Way Guard

- Way-partitioning **L1**-L3
- Protection domain IDs
  - Fill mask
  - **Hit mask**
    **- Hits**



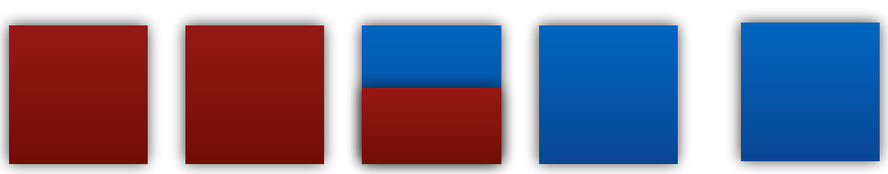
# DAWG: Dynamically Allocated Way Guard

- Way-partitioning **L1**-L3
- Protection domain IDs
  - Fill mask
  - **Hit mask**
    **- Hits**
    **- PLRU updates**



# Higher Security than QoS Cache Partitioning

- Production QoS way-partitioning (CAT) by design allows hits across domains
- Not a security barrier

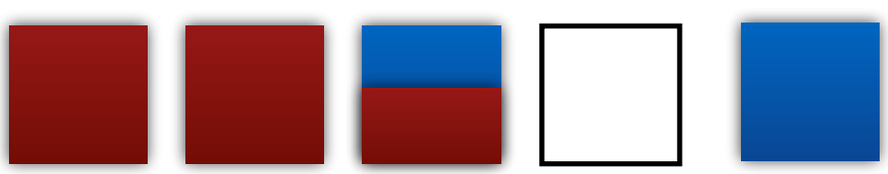


Hits Cross-Domain

| | CAT | **DAWG** |
|---|---|---|
| Way allocation | ✅ | ✅ |
| Hits in victim | ❌ | ✅ |
| | | |
| | | |

# Higher Security than QoS Cache Partitioning

- Production QoS way-partitioning (CAT) by design allows hits across domains
- Not a security barrier

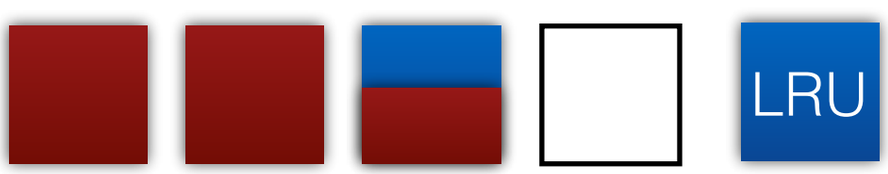


Flush

| | CAT | **DAWG** |
|---|---|---|
| Way allocation | ✅ | ✅ |
| Hits in victim | ❌ | ✅ |
| Flush in victim | ❌ | ✅ |
| | | |

# Higher Security than QoS Cache Partitioning

- Production QoS way-partitioning (CAT) by design allows hits across domains
- Not a security barrier

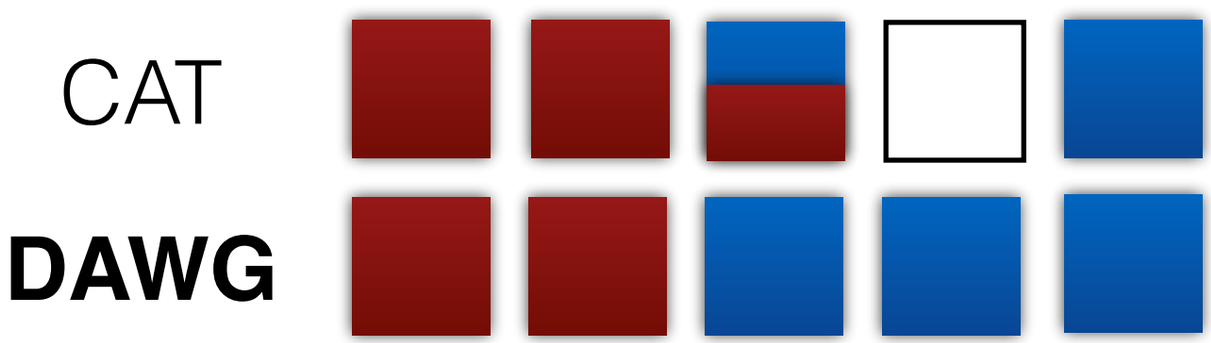



| | CAT | **DAWG** |
|---|---|---|
| Way allocation | ✅ | ✅ |
| Hits in victim | ❌ | ✅ |
| Flush in victim | ❌ | ✅ |
| PLRU/NRU leak | ❌ | ✅ |

# Shared Memory ↛ Shared Cache

| | CAT | **DAWG** |
|---|---|---|
| Hits in victim | ❌ | ✅ |
| Flush in victim | ❌ | ✅ |

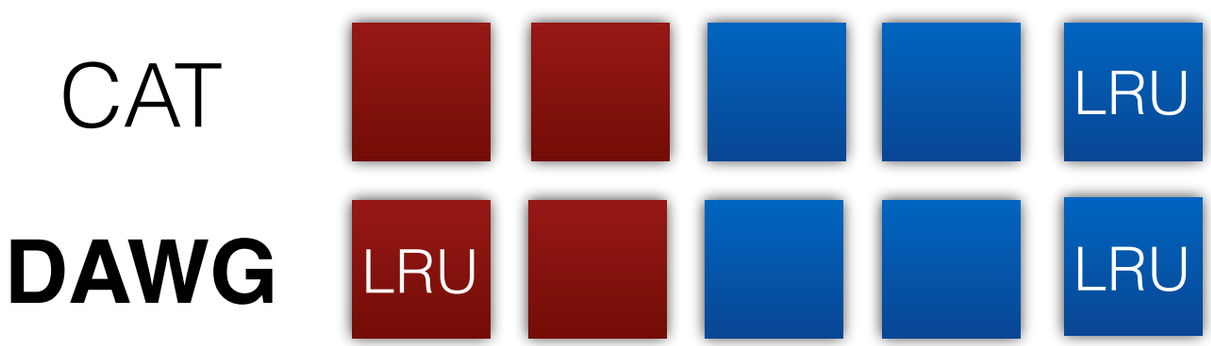
~~Flush+Reload~~
~~Evict+Reload~~
~~Thrash+Reload~~



# Shared Sets ↛ Shared Metadata

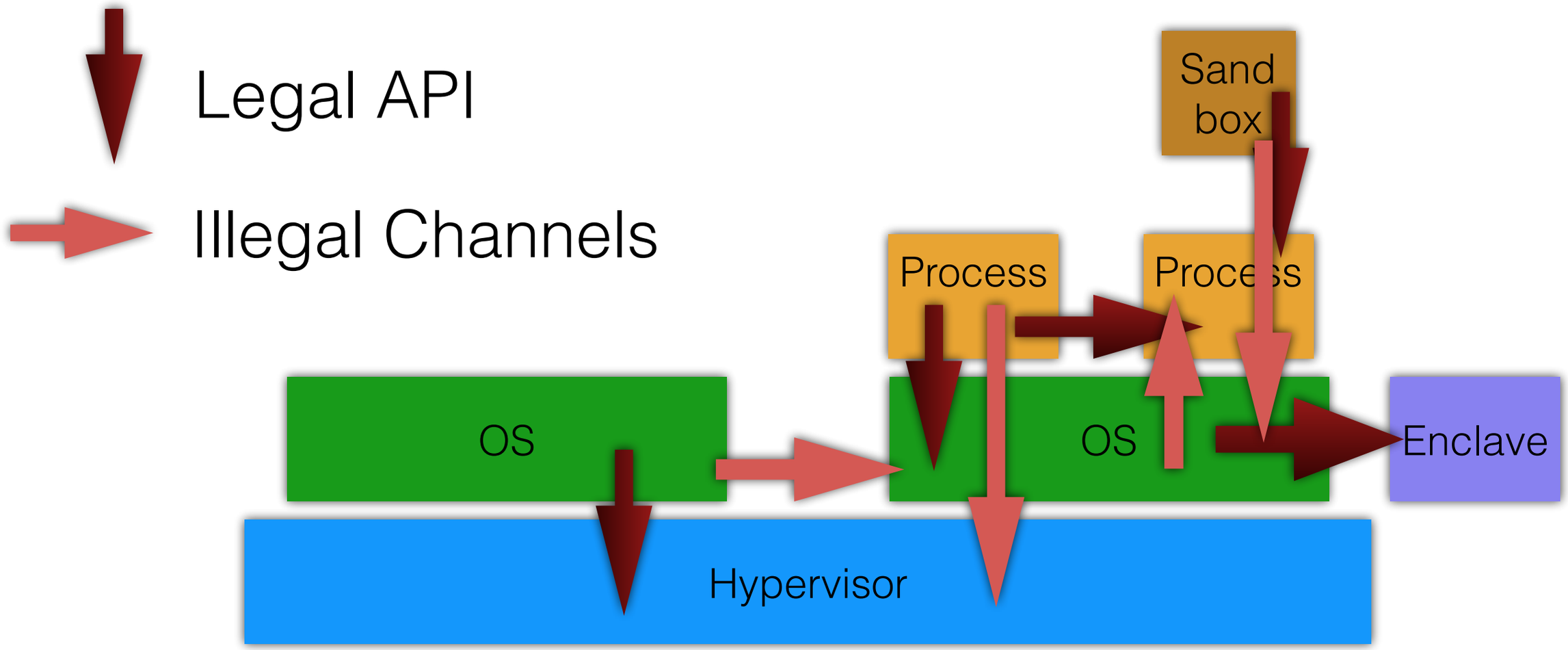
| | CAT | **DAWG** |
|---|---|---|
| PLRU/NRU leak | ❌ | ✅ |

~~PLRU-Prime+Probe~~

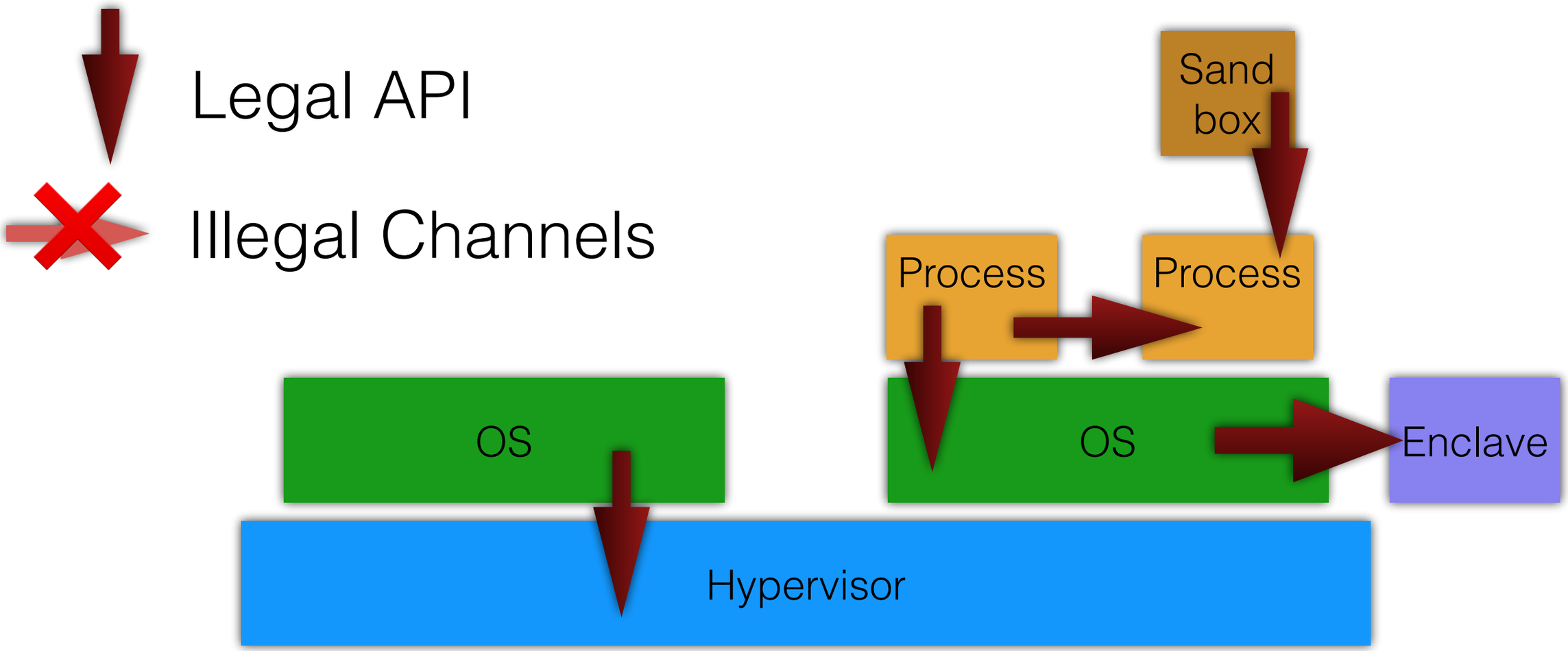


# OS Support and Resource Management

# Protection Domain Isolation

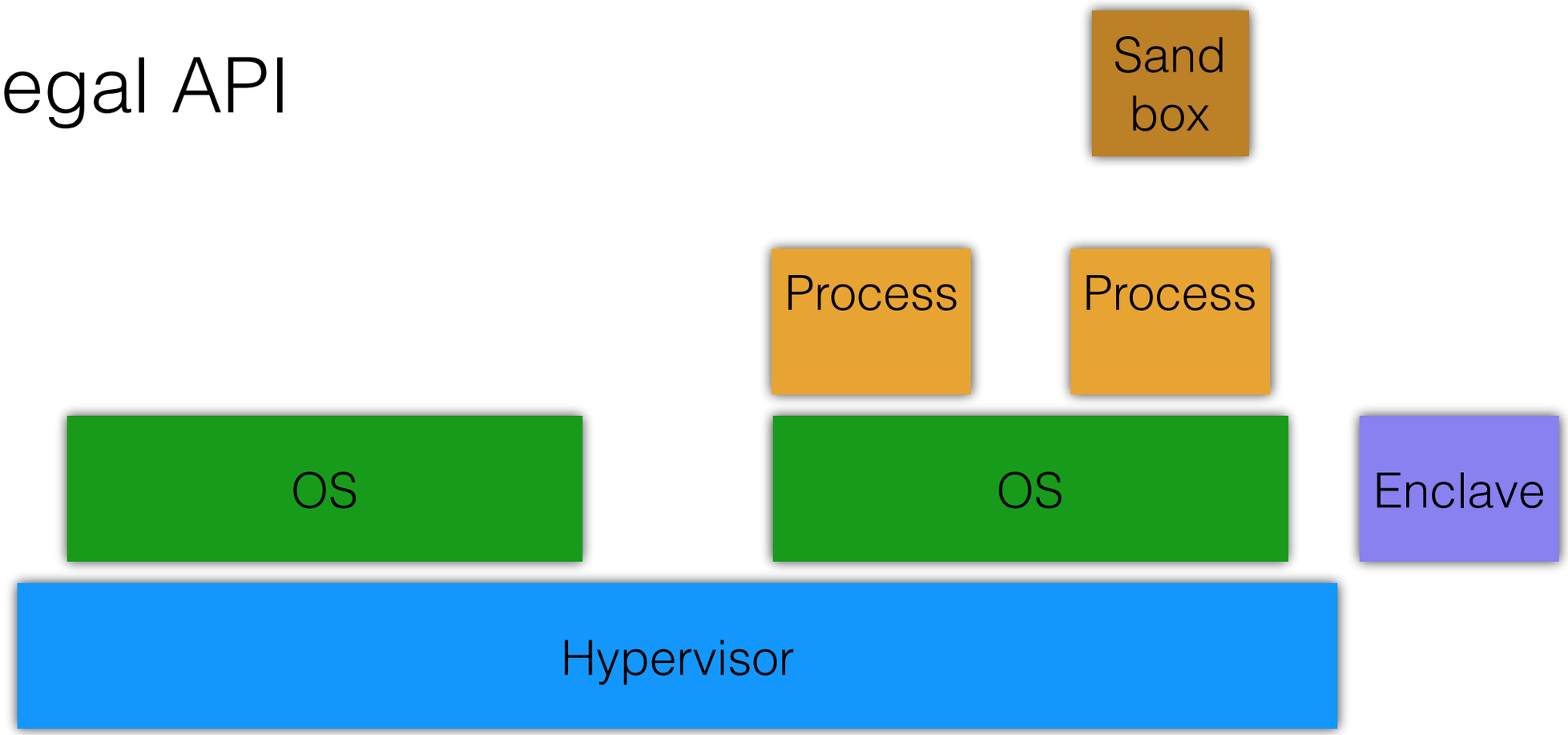


# Protection Domain Isolation



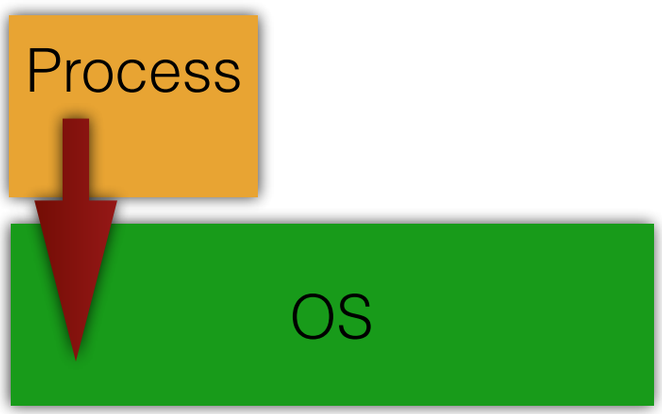
# Protection Domain Isolation

Legal API



# Fast System Calls

1. OS can access everything in process memory
2. In/out arguments in cache (dirty)
3. OS must not leak



# Core & OS changes: Domain Descriptors

- Existing support for CAT

Domain Descriptors
Global

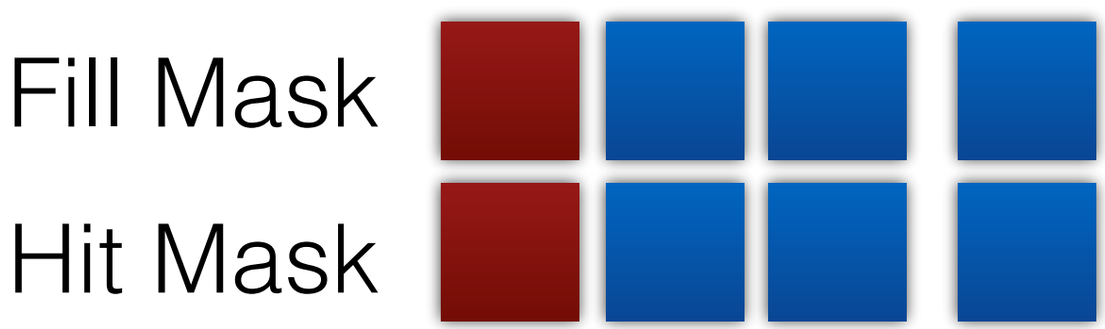
Fill Mask 

0 0111

1 1000

# Core & OS changes: Domain Descriptors

- Existing support for CAT + DAWG




Domain Descriptors
Global

0 0111,0111

1 1000,1000

# Core & OS changes: Domain Selectors

- Existing support for SMAP (Supervisor Mode Access Protection)

Few routines access user-data & toggle SMAP

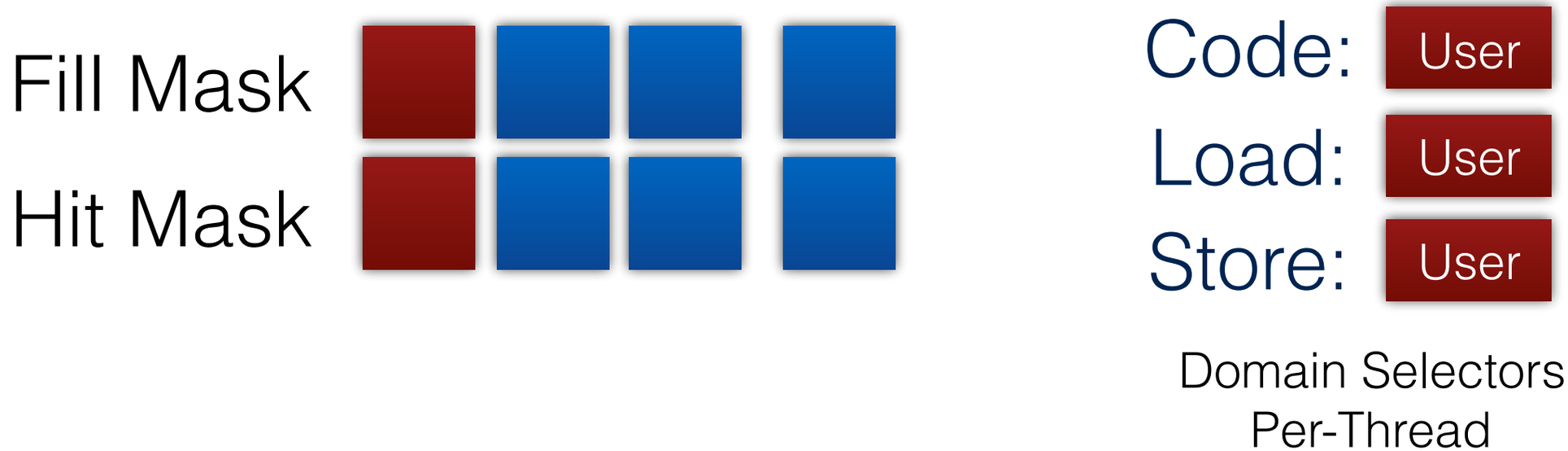
```
copy_from_user
copy_to_user
    ...
```

# Core & OS changes: Domain Selectors

- Existing support for SMAP + DAWG
- Core MSR: separate code / load / store selectors

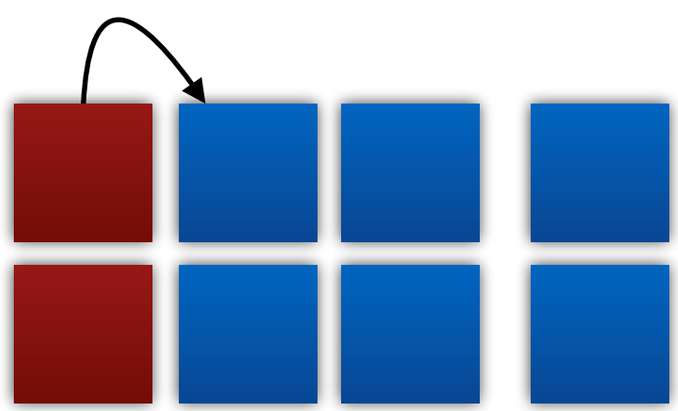
# Core & OS changes: System calls

- Existing support for CAT & SMAP + DAWG
- Core MSR: separate code / load / store selectors



# Core & OS changes: System calls

- Existing support for CAT & SMAP + DAWG
- Core MSR: separate code / load / store selectors



`copy_from_user`

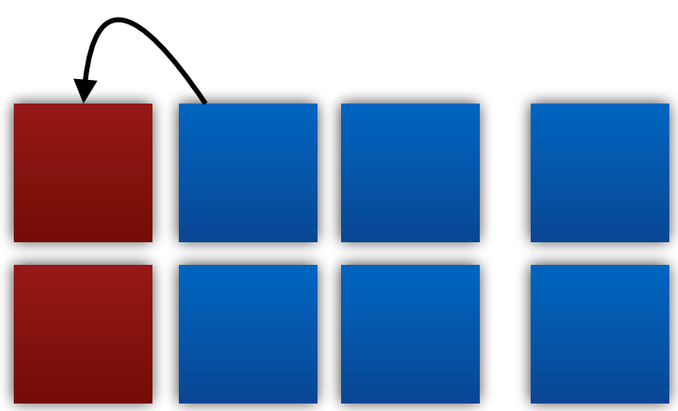
Code: OS

Load: User

Store: OS

# Core & OS changes: System calls

- Existing support for CAT & SMAP + DAWG
- Core MSR: separate code / load / store selectors



`copy_to_user`

Code: OS

Load: OS

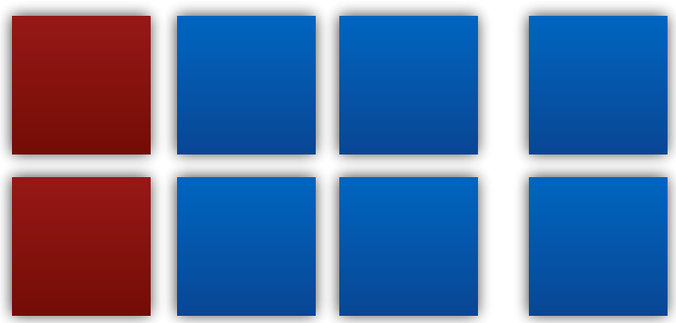
Store: User

# Resource Management

- Extends CAT support + secure domain reallocation
- Secure dynamic way reassignment

Fill Mask

Hit Mask

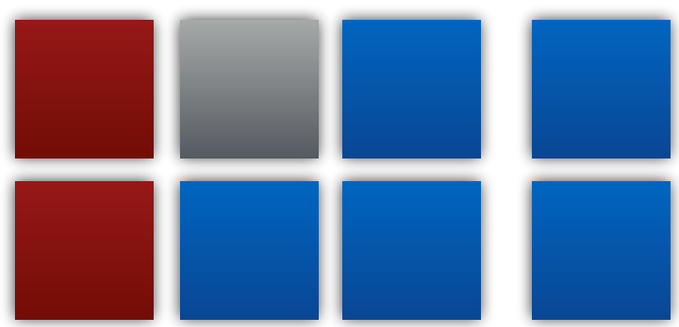


# Secure Dynamic Way Reassignment

- Secure way sanitization
- Concurrent for shared caches

Fill Mask

Hit Mask

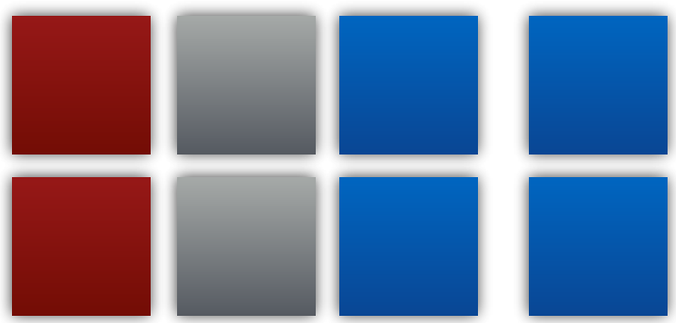


Flush blocks in revoked way

# Secure Dynamic Way Reassignment

Fill Mask

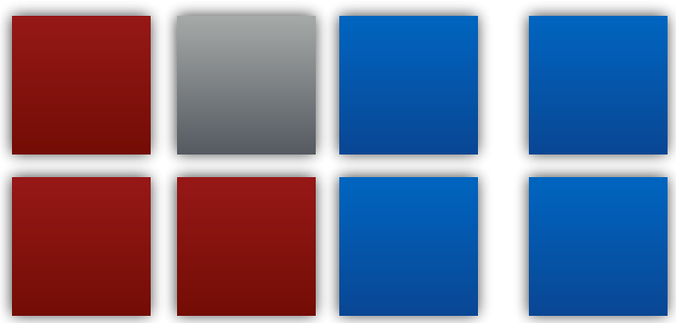
Hit Mask



# Secure Dynamic Way Reassignment

Fill Mask

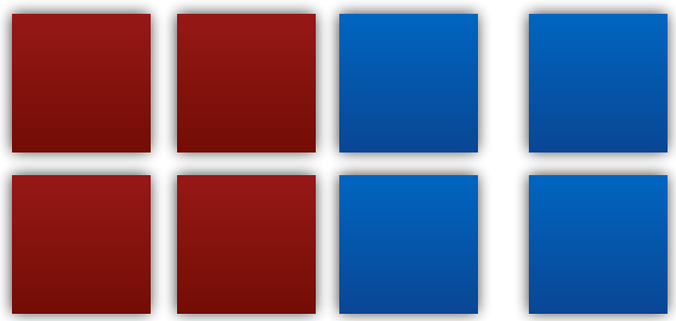
Hit Mask



# Secure Dynamic Way Reassignment

Fill Mask

Hit Mask

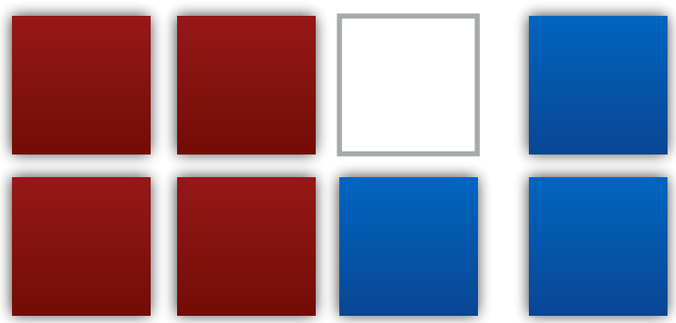


# DAWG Beyond Cache Partitioning

- Cache Way Locking

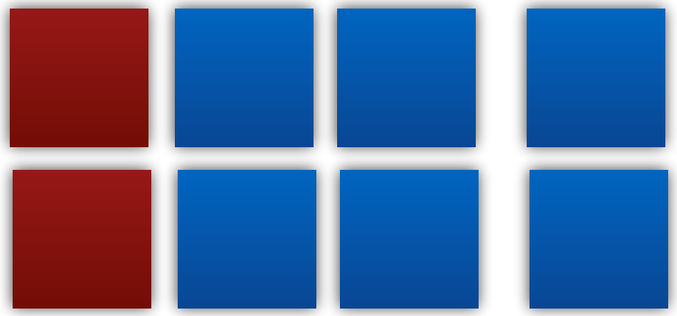
Fill Mask

Hit Mask



# Core & OS changes

- Shared libraries, memory mapped I/O, VM page sharing, and cache coherence
- Details in our paper



# Performance Evaluation

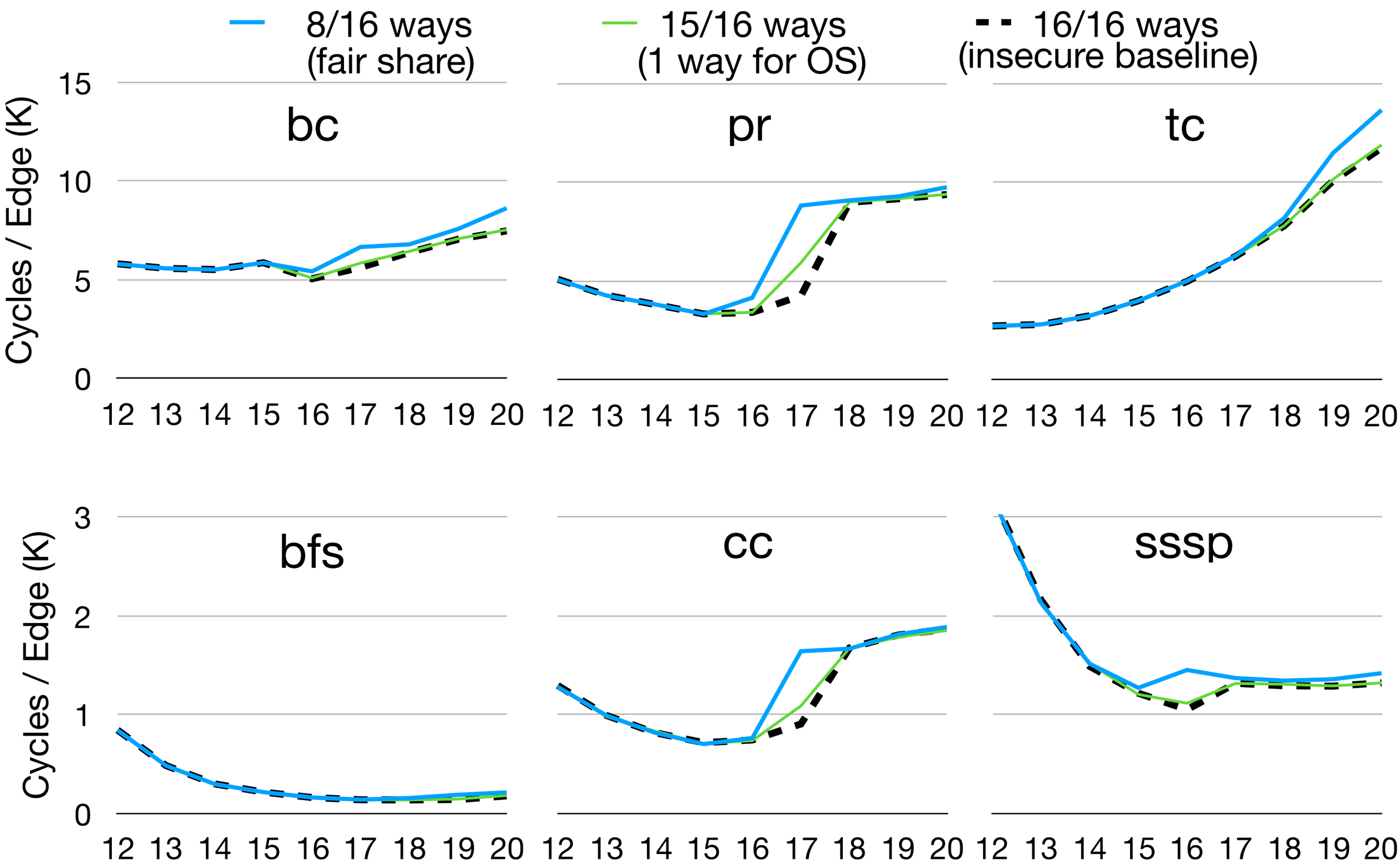
# Matching Performance of QoS Cache Partitioning

- Typical use case:
  public cloud VM isolation
  (no page sharing, no core sharing, no SMT)

→ DAWG's performance is identical to production LLC way-partitioning (Intel's CAT)

VM1 VM2

# Way-Partitioning

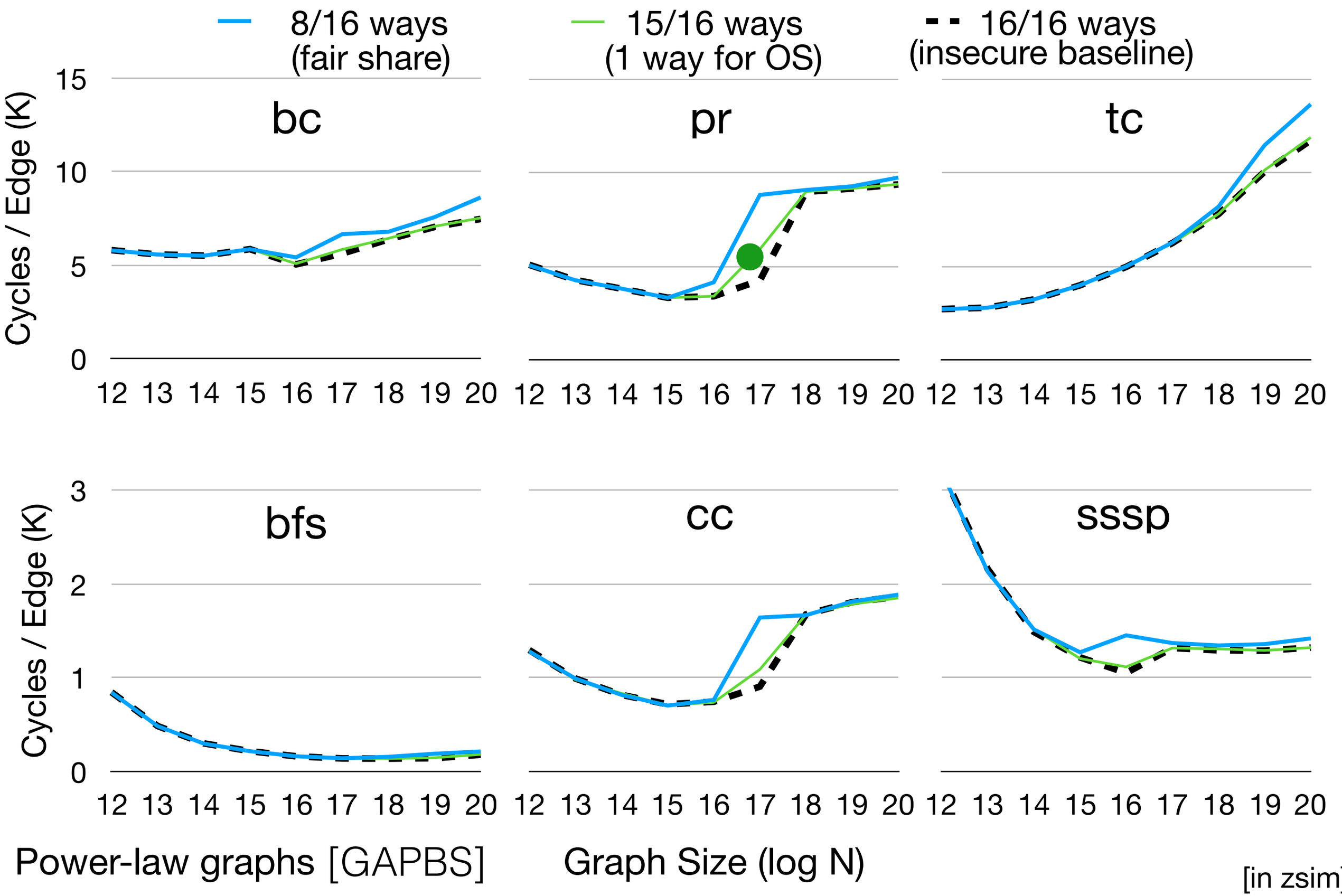



Power-law graphs [GAPBS]

Graph Size (log N)

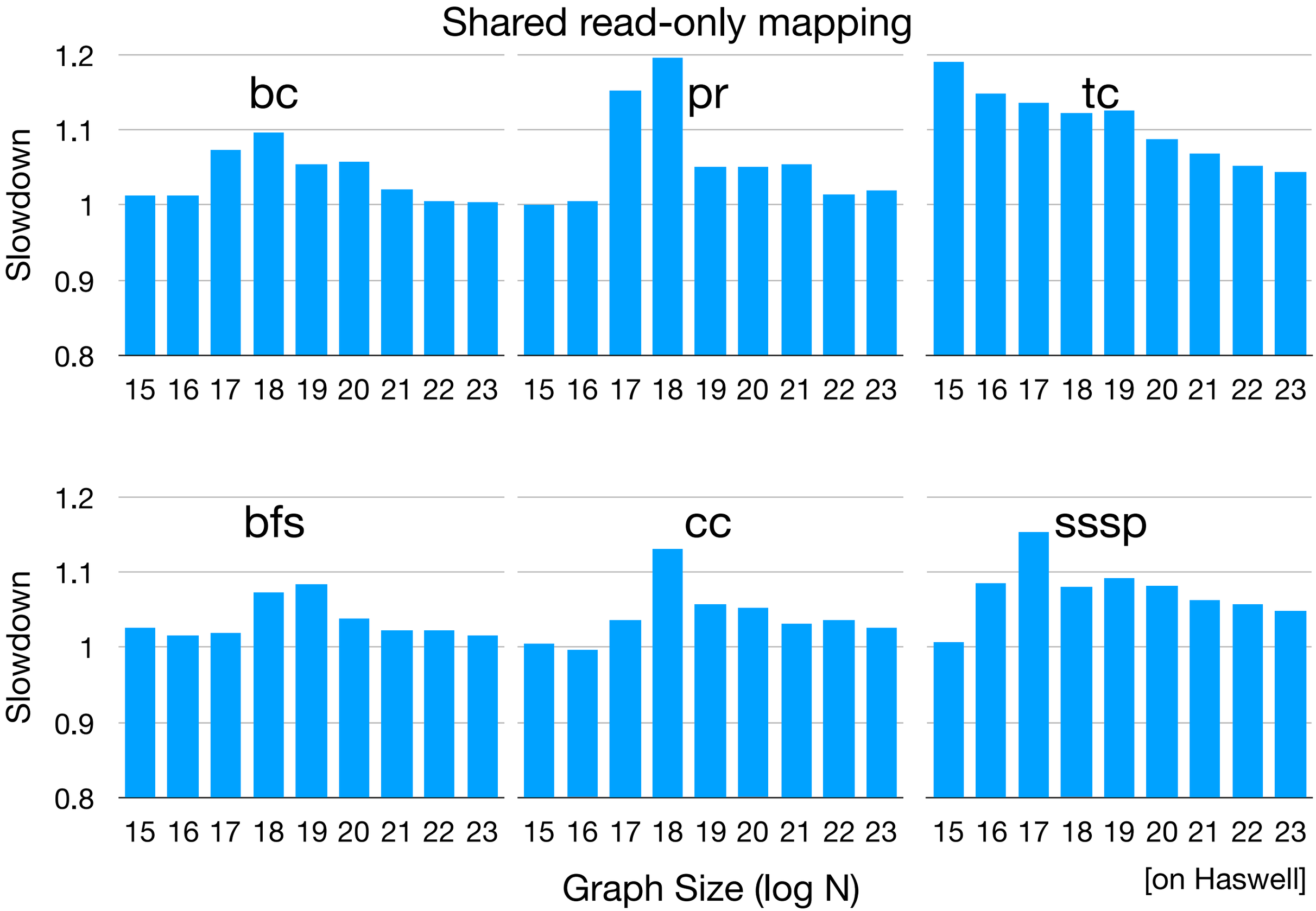
[in zsim]

# Way-Partitioning




[in zsim]

# Shared Data: DAWG vs CAT

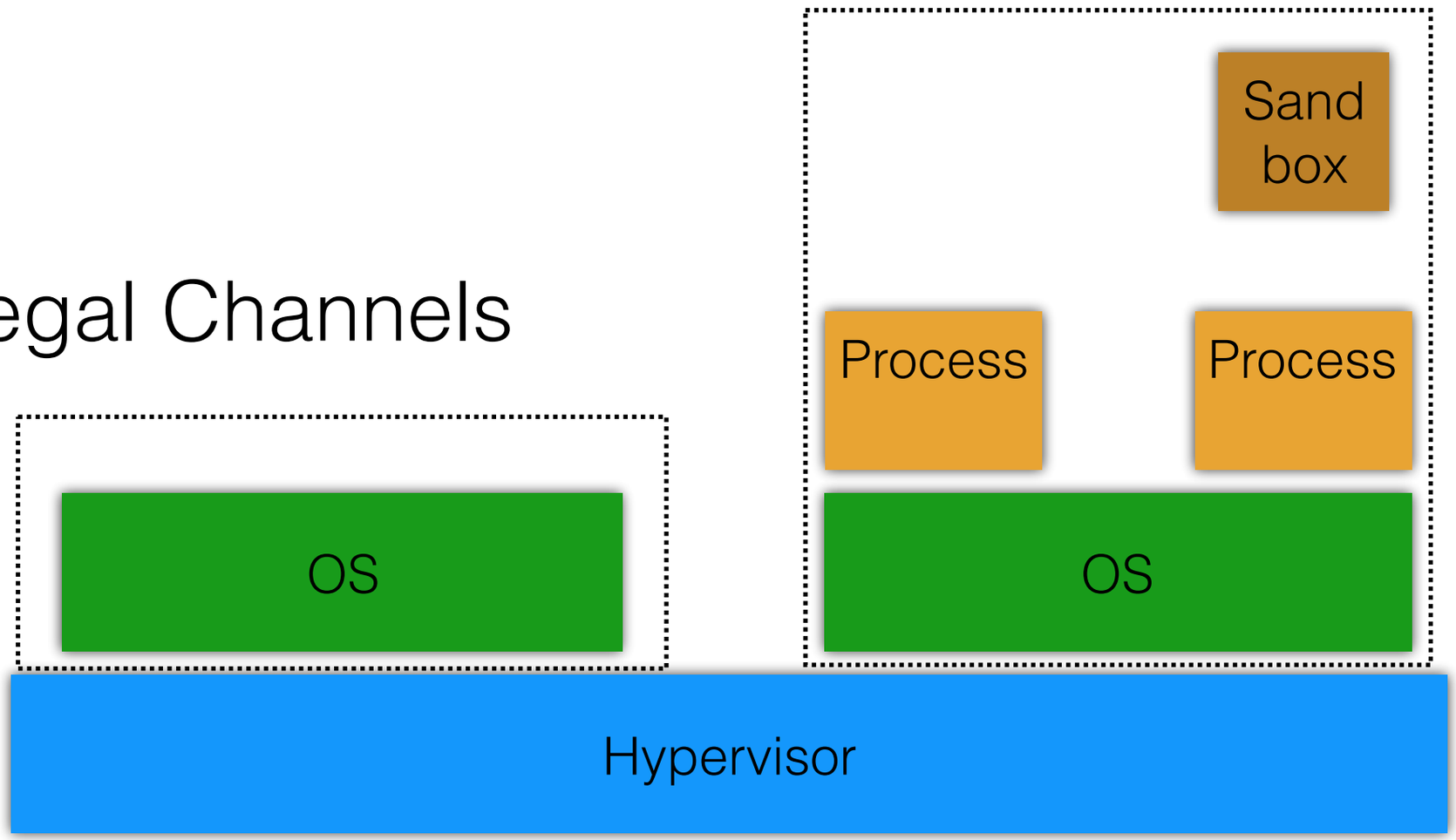



[on Haswell]

# Security Evaluation

# Cache Partitioning ≈ Dedicated Host Per Domain

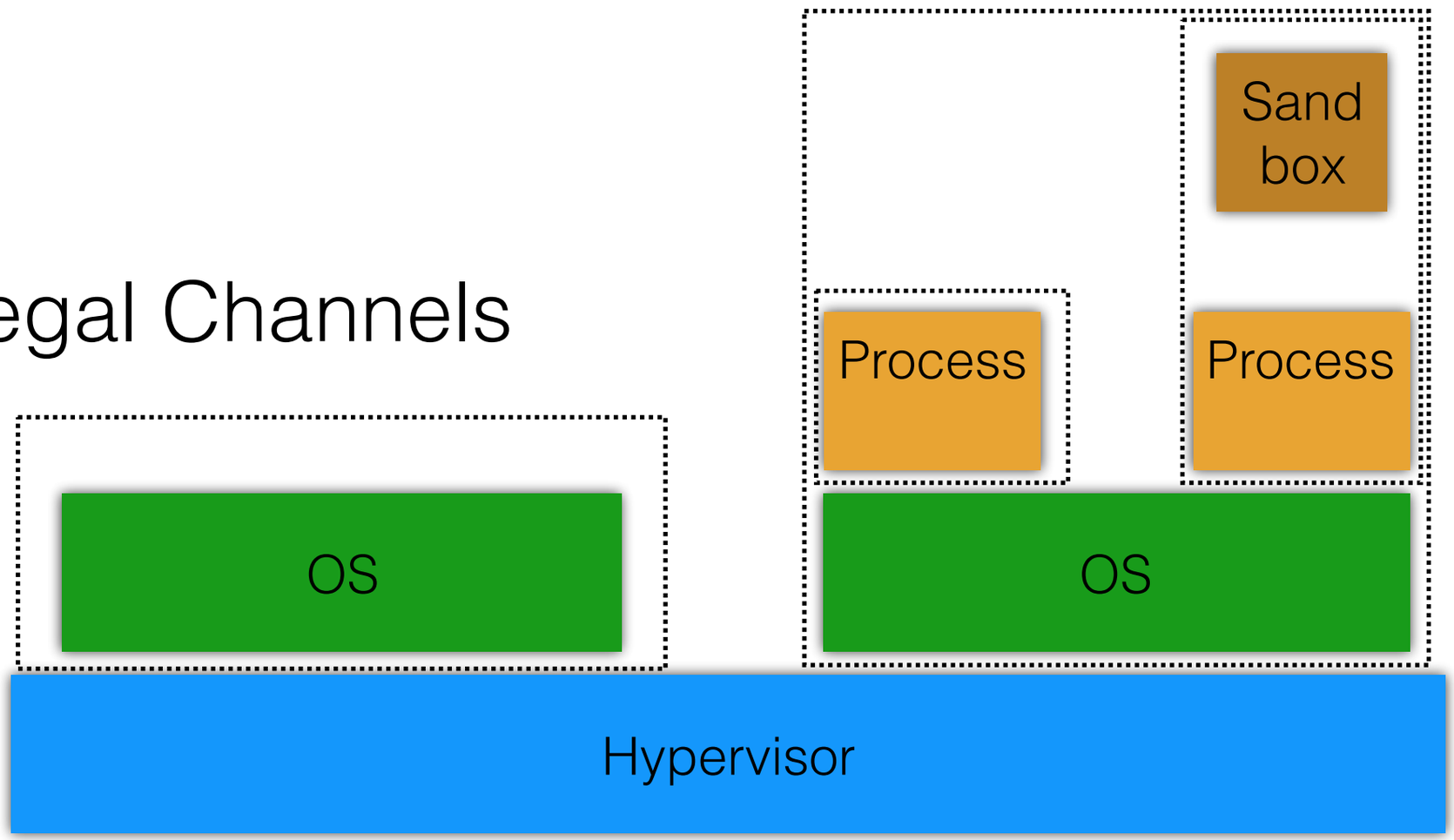
Illegal Channels




Isolating peers

# Cache Partitioning ≈ Dedicated Host Per Domain

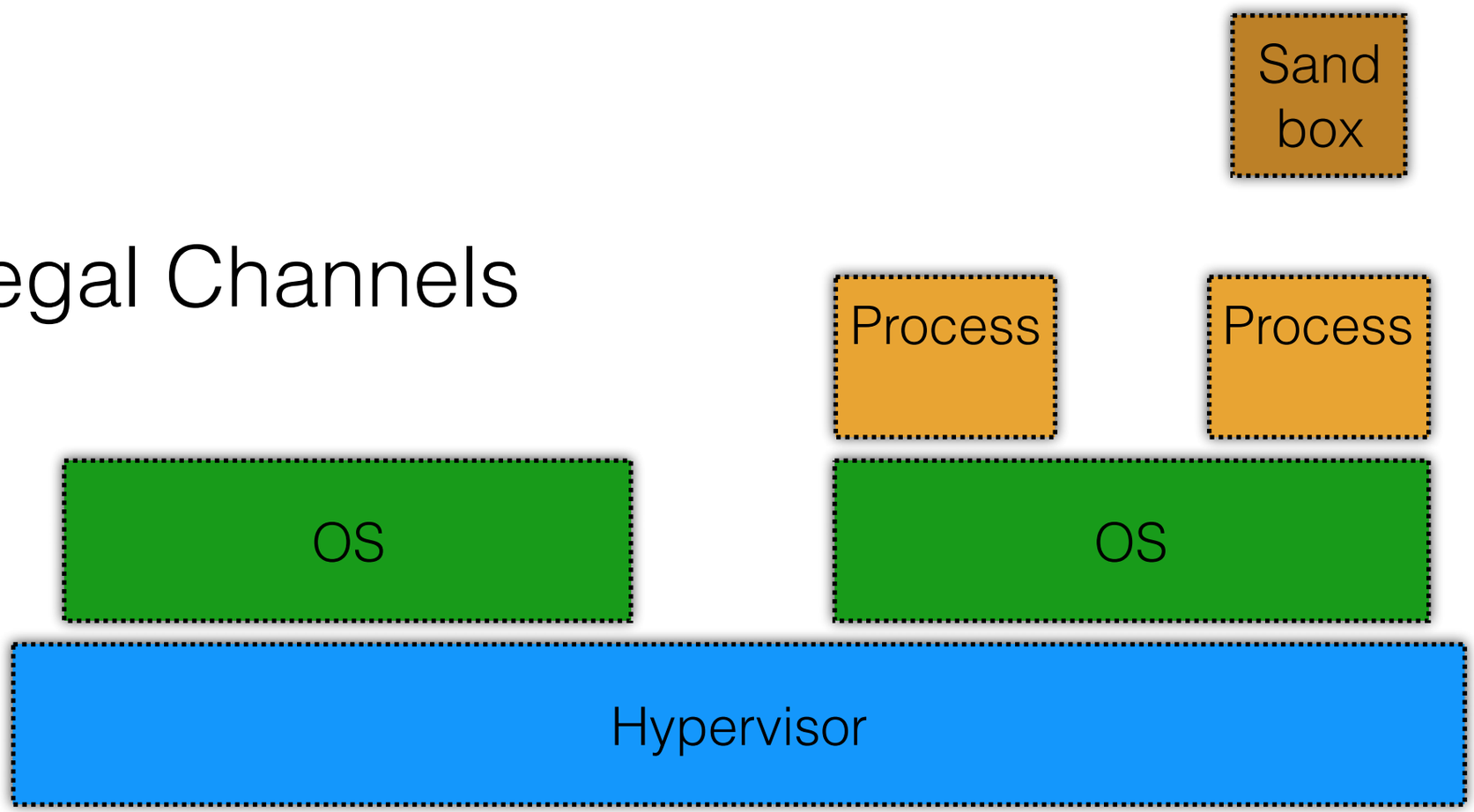
Illegal Channels




Isolating peers

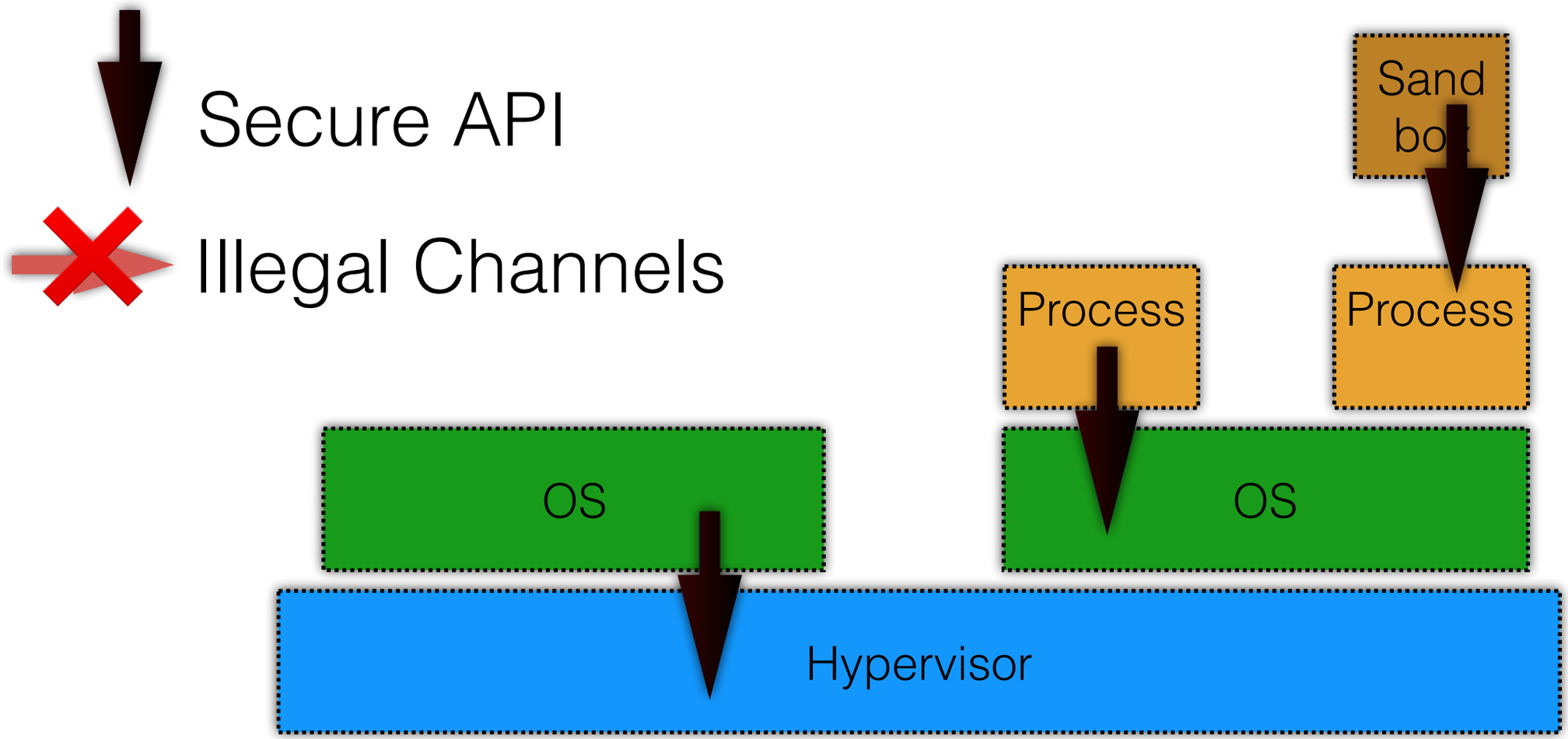
# Cache Partitioning ≈ Dedicated Host Per Domain

Illegal Channels



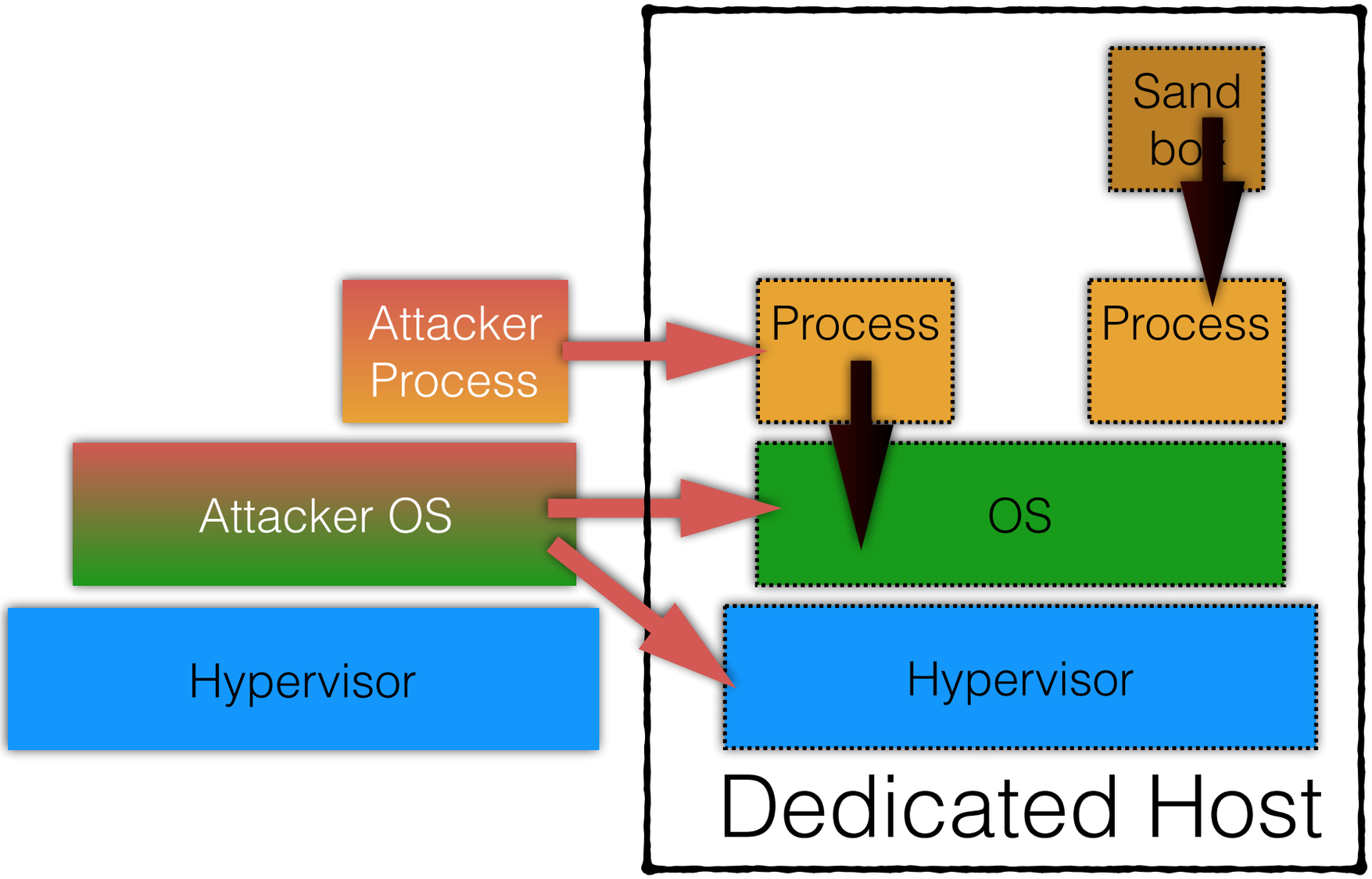

Isolating peers and parents

# Cache Partitioning ≈ Dedicated Host Per Domain



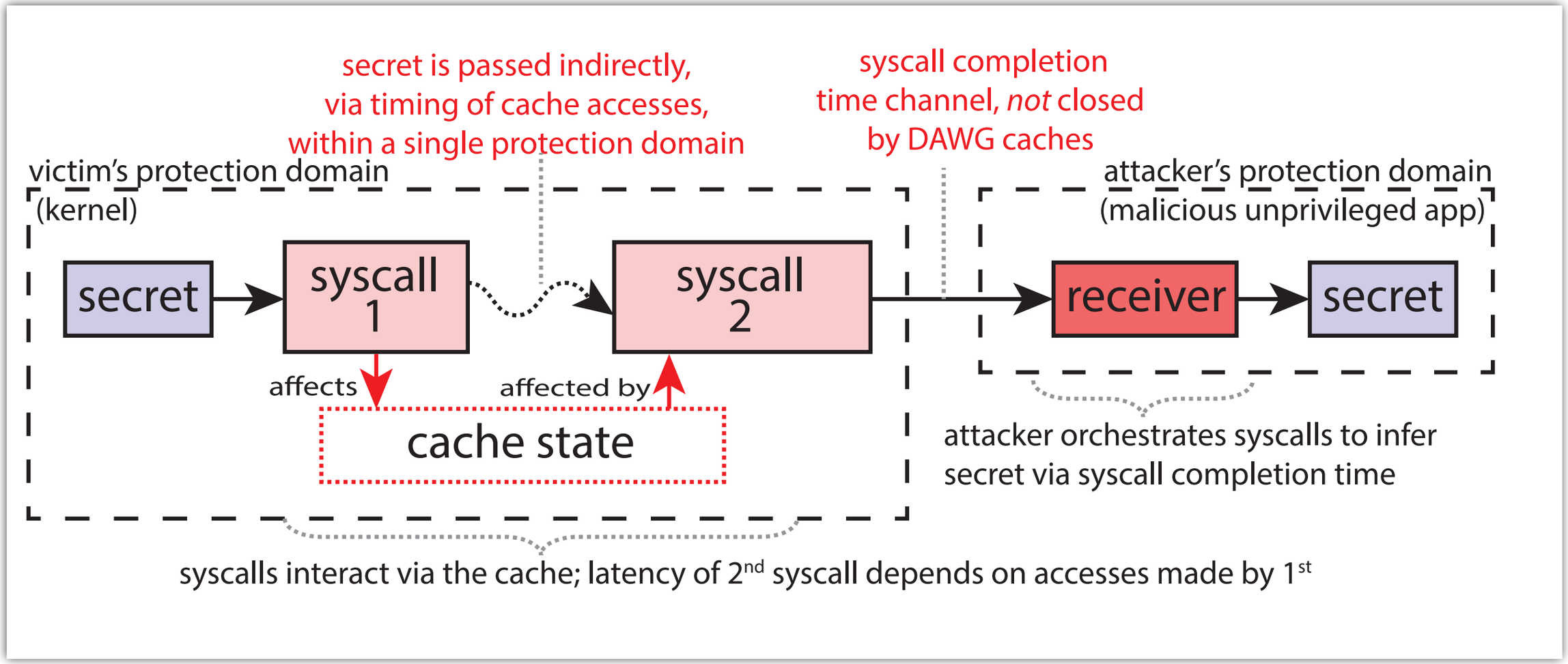

Secure communication

# Dedicated Host Insufficient: Remote Cache Timing Attacks




- High-bandwidth remote cache timing attack

# Remote Cache Reflection: Attacks and Defenses




- High-bandwidth remote cache timing attack

# Conclusion

- Partitioning is the foundation
- Minimal changes to hardware: Build on CAT
- Minimal changes to OS: Build on SMAP
- Minimal performance overhead:
  Zero or small over CAT QoS
- DAWG applies beyond caches: TLB, etc

# Thanks

Vladimir Kiriansky
vlk@csail.mit.edu

DARPA
NSF
U.S. DEPARTMENT OF ENERGY | Office of Science
SPAWAR
CSAIL
MIT COMPUTER SCIENCE AND ARTIFICIAL INTELLIGENCE LABORATORY
DELTA
TOYOTA RESEARCH INSTITUTE

# Backup Slides

# Beyond Cache Partitioning: Code Prioritization

# Beyond Cache Partitioning Streaming Data Isolation

- Graph application use case:
  1-way for streaming edges
  3-ways for per-vertex data