CprE 381: Computer Organization and Assembly Level Programming

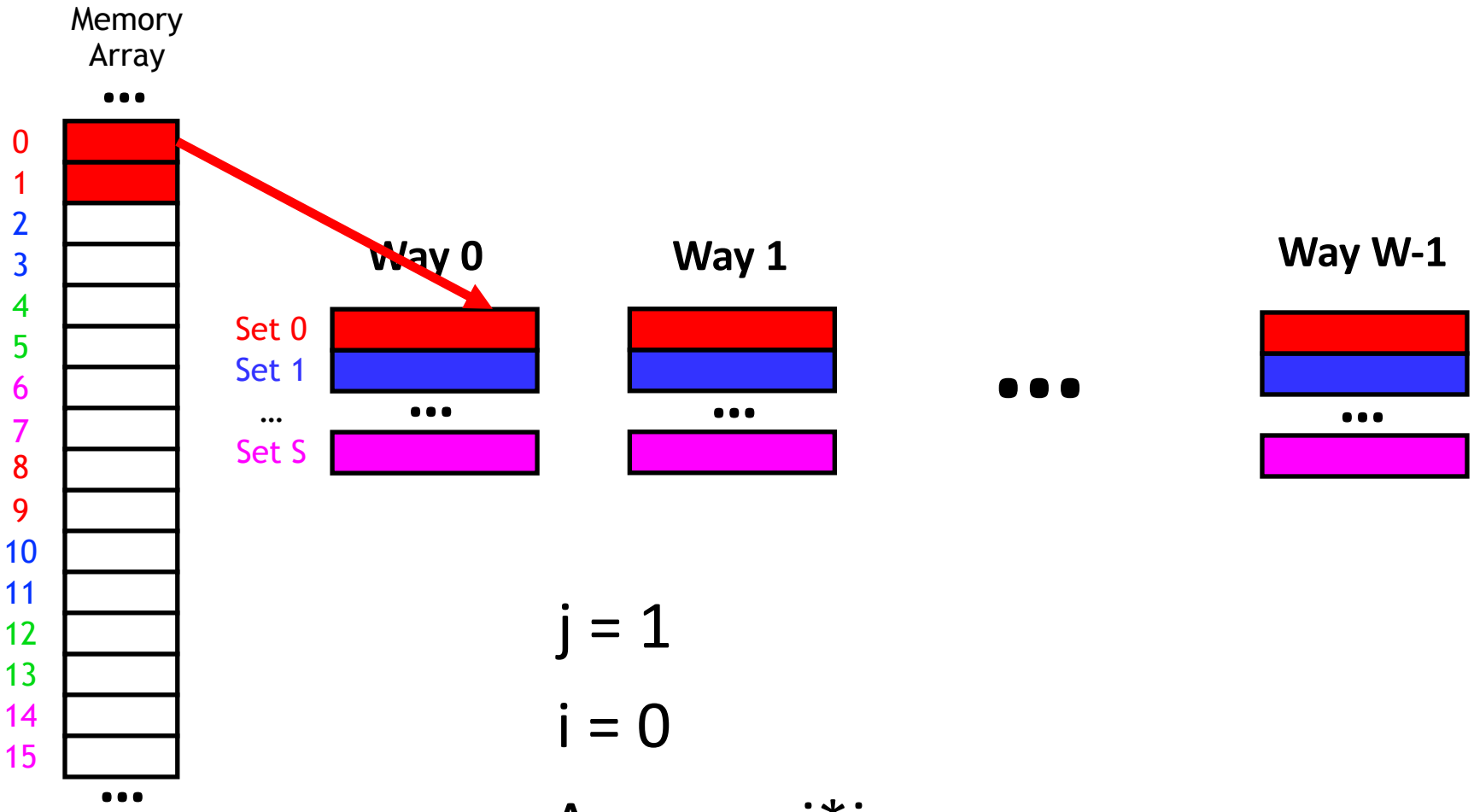
HW Security

Henry Duwe
Electrical and Computer Engineering
Iowa State University

Administrative

- Part 4 due in lab this week
  - WARNING: No extensions!!!
- Final Exam
  - When: Mon May 6 at 7:30am
  - Where: **Marston 2155** (everyone)
  - What: Branch prediction through HW security
- Please take online course assessment

Review: HW11 Cache Test 1

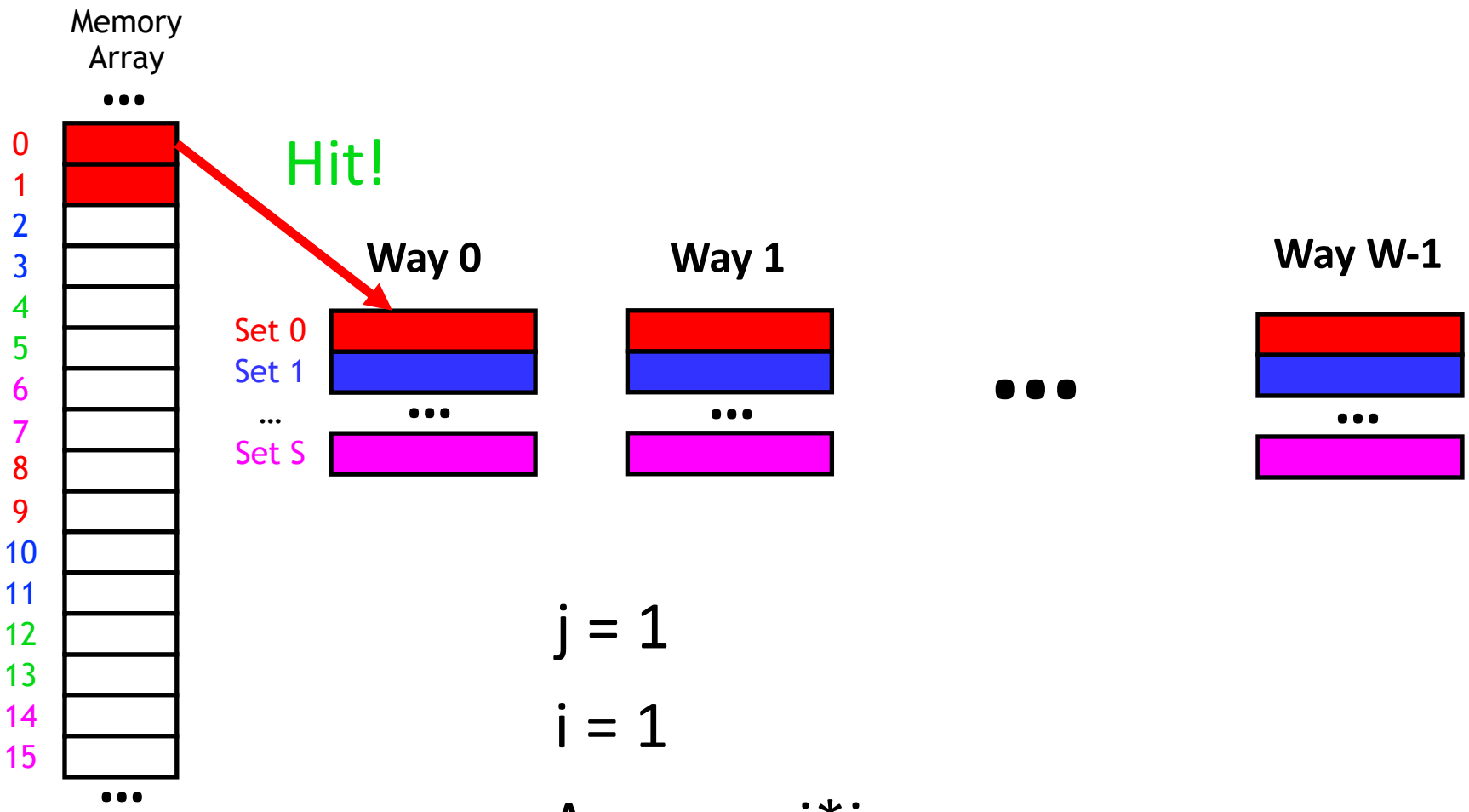


j = 1

i = 0

Access = i*j

Review: HW11 Cache Test 1

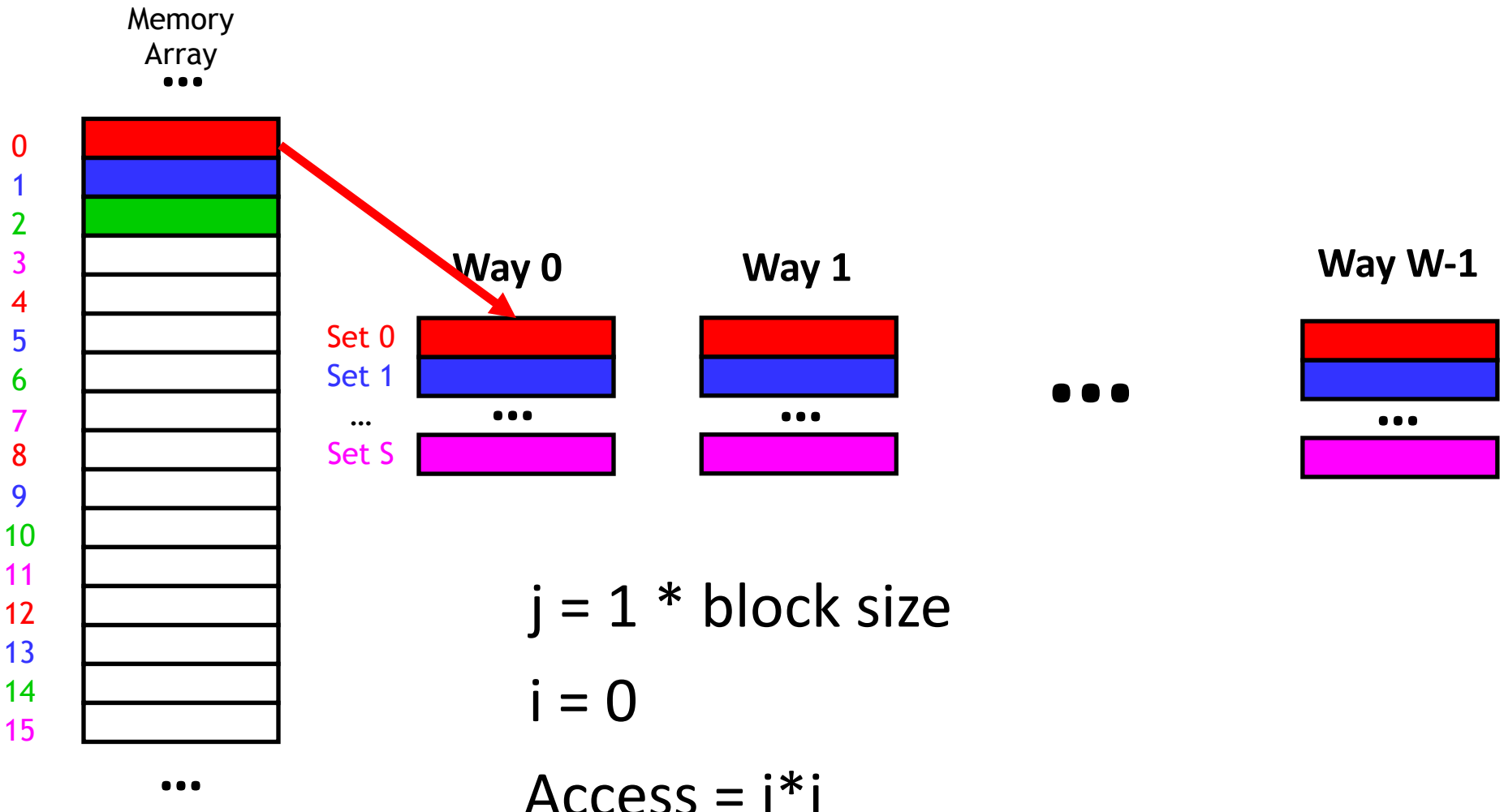


j = 1

i = 1

Access = i*j

Review: HW11 Cache Test 1

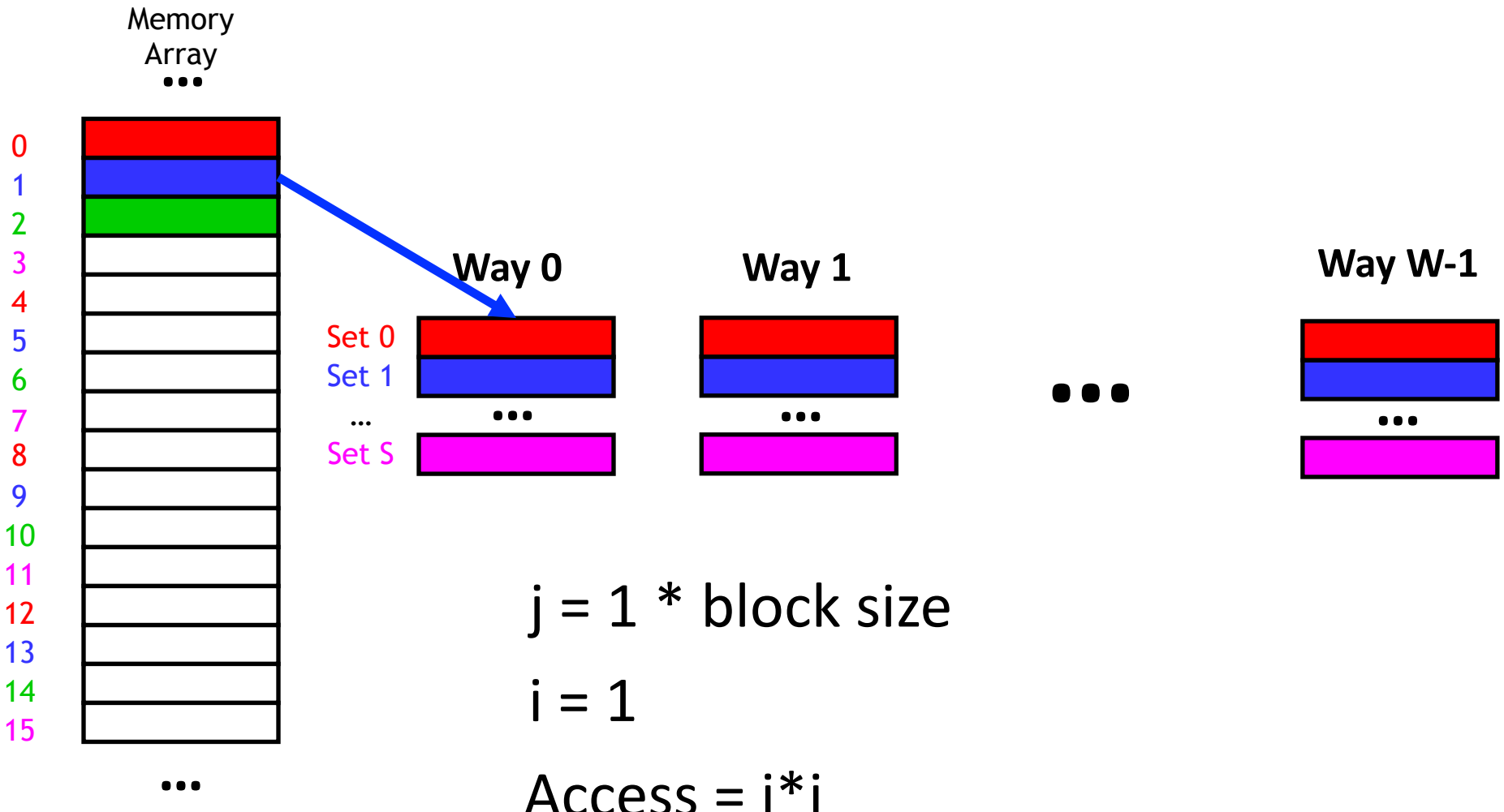


j = 1 * block size

i = 0

Access = i*j

Review: HW11 Cache Test 1

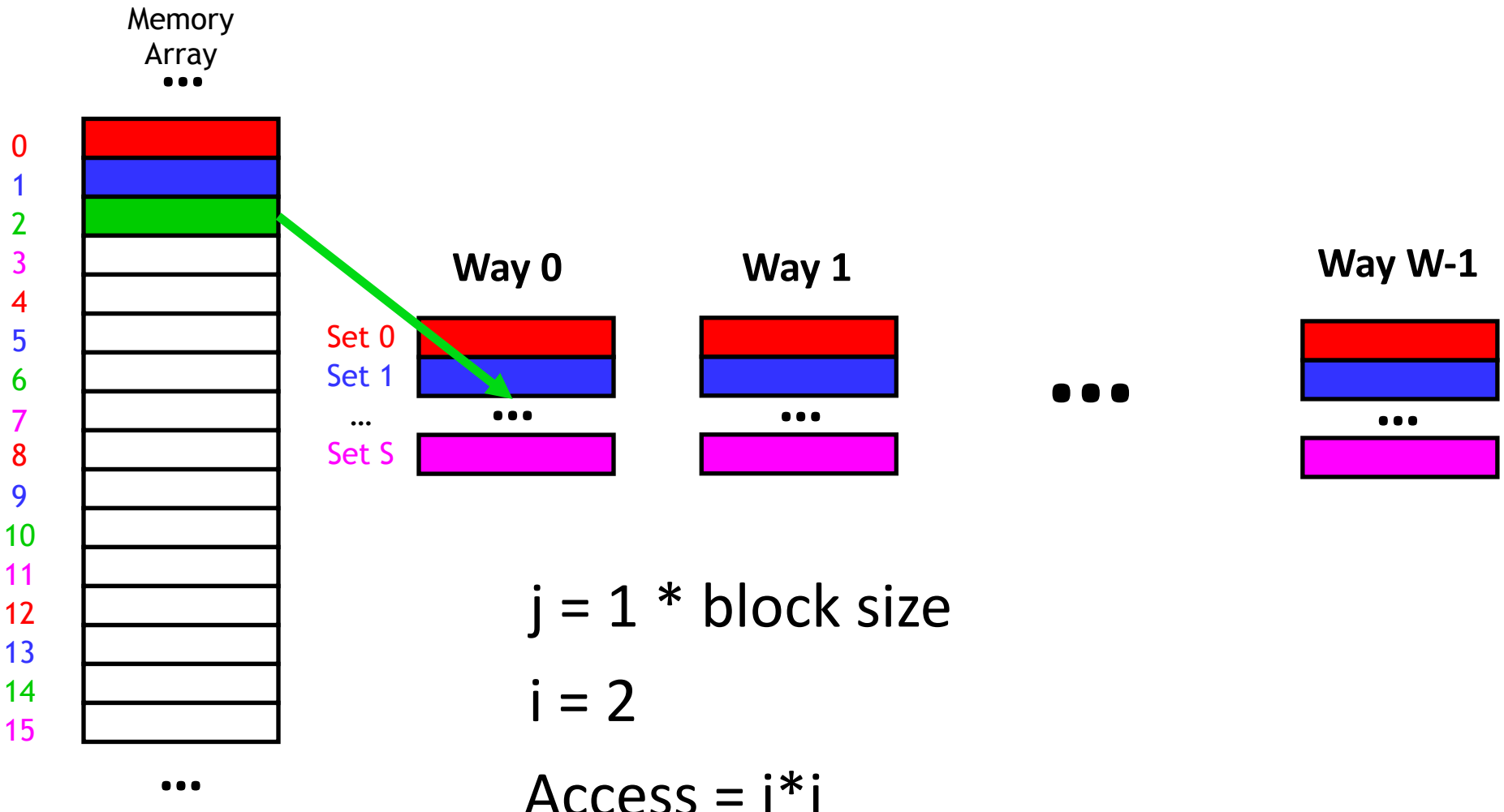


j = 1 * block size

i = 1

Access = i*j

Review: HW11 Cache Test 1

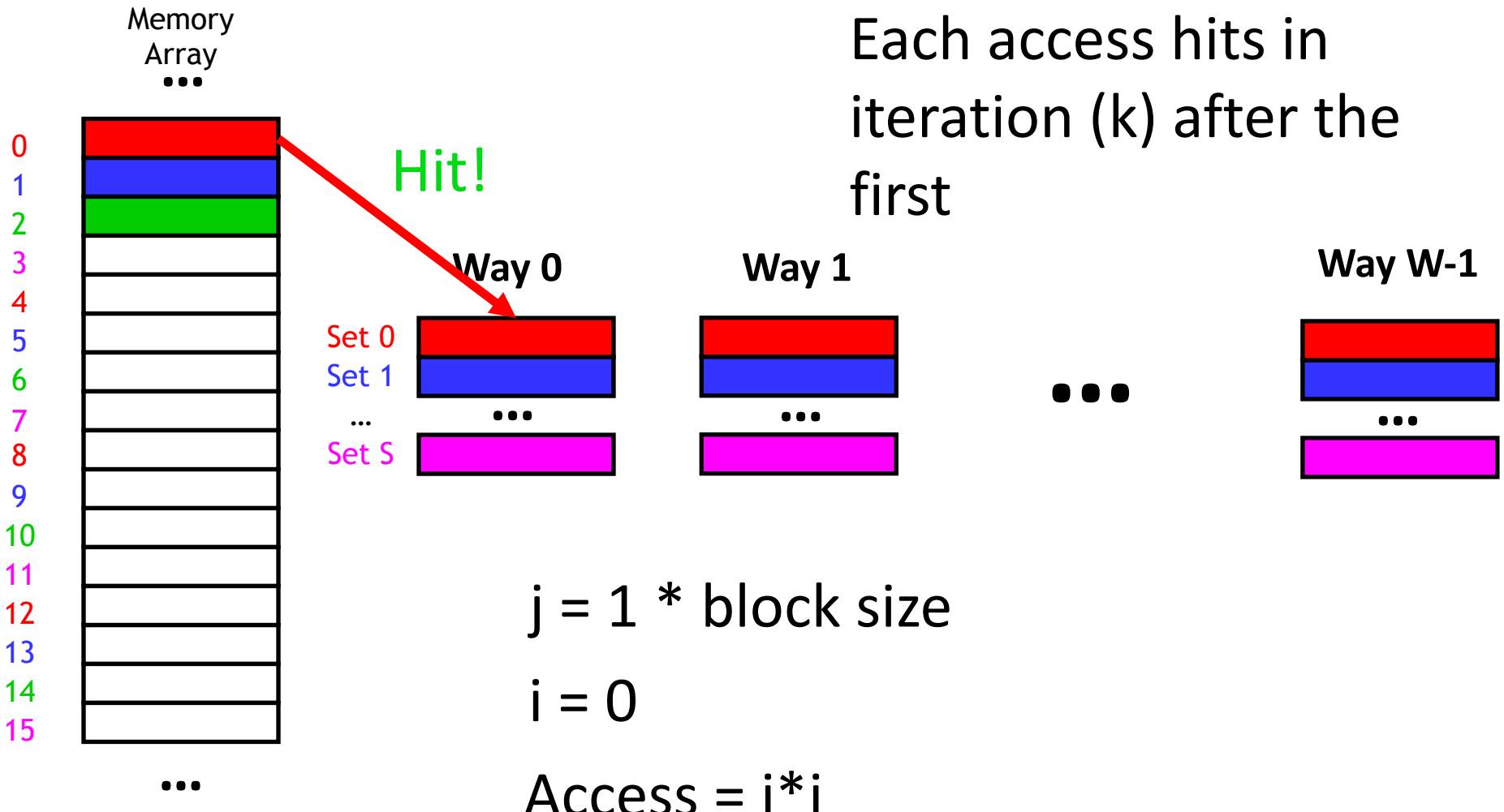


j = 1 * block size

i = 2

Access = i*j

Review: HW11 Cache Test 1



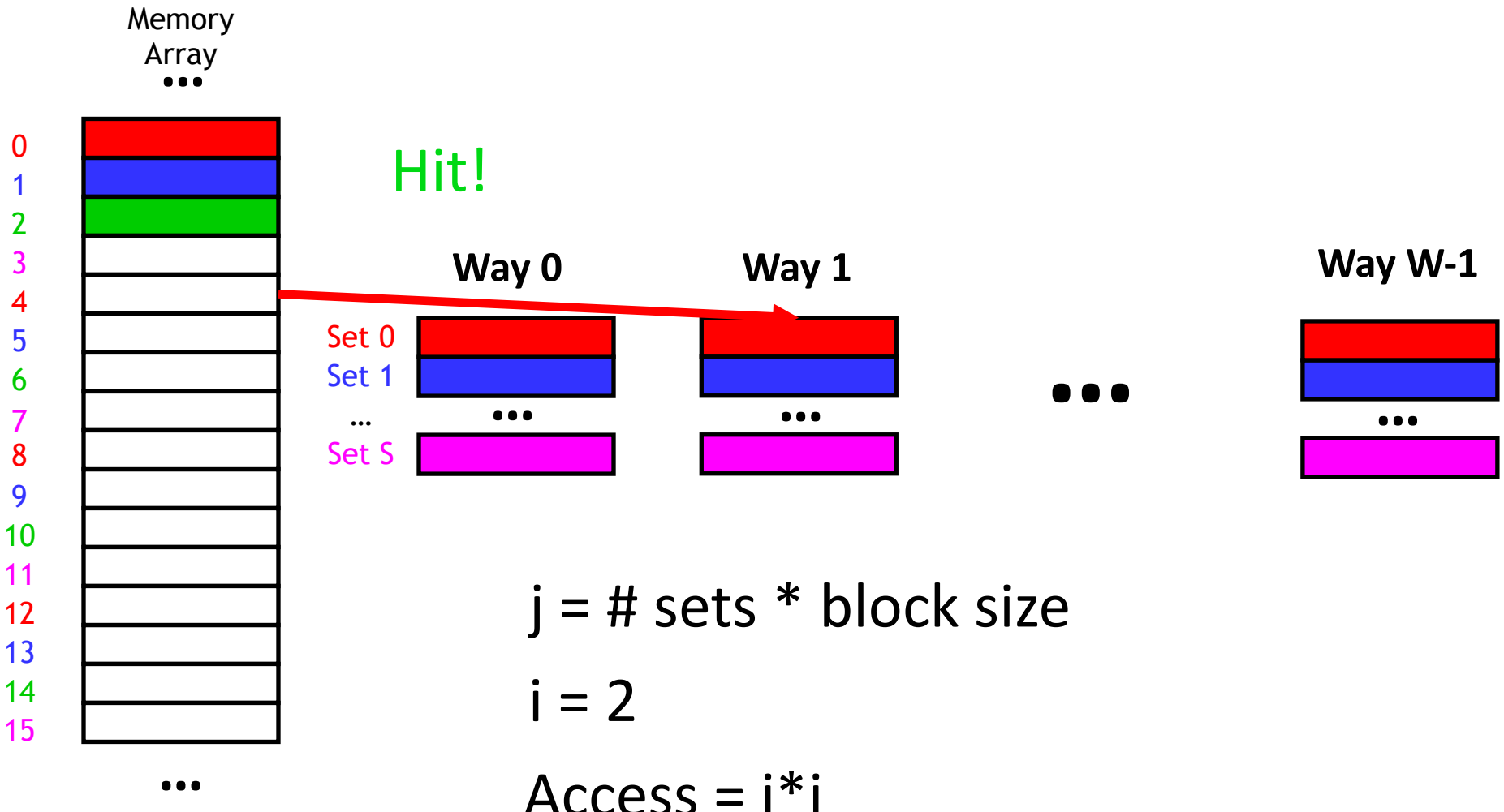
Each access hits in iteration (k) after the first

j = 1 * block size

i = 0

Access = i*j

Review: HW11 Cache Test 1

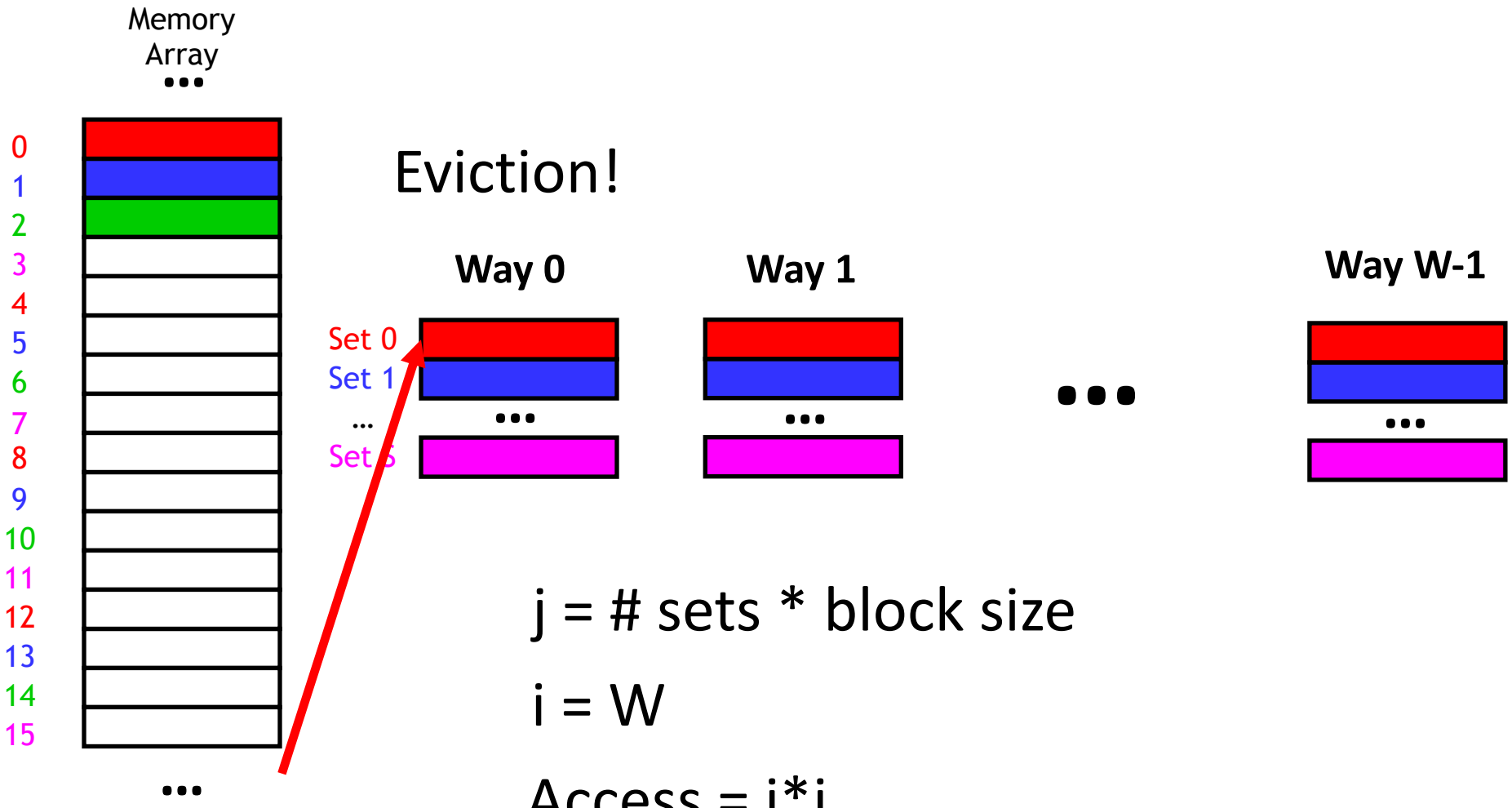


j = # sets * block size

i = 2

Access = i*j

Review: HW11 Cache Test 1

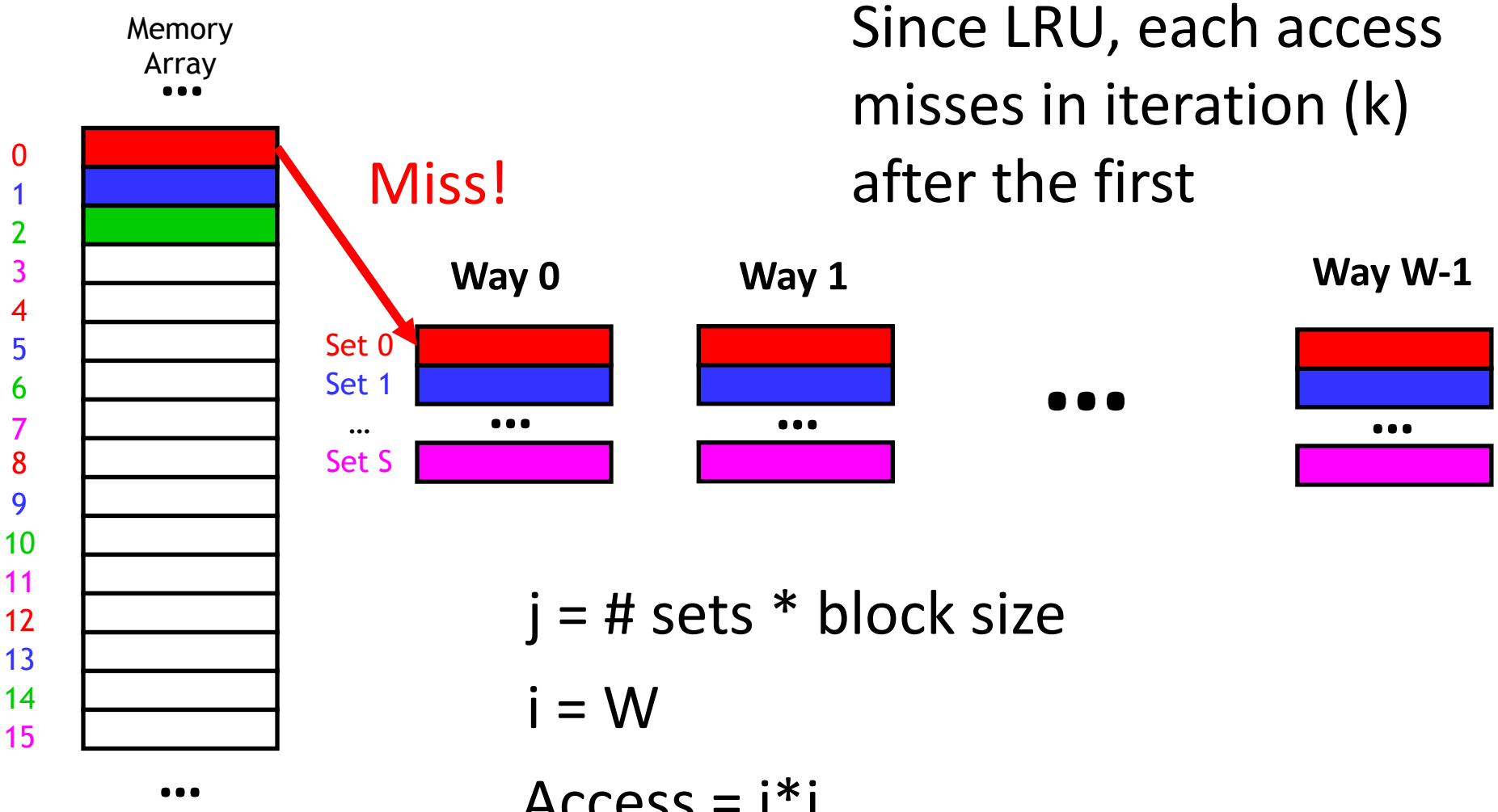


j = # sets * block size

i = W

Access = i*j

Review: HW11 Cache Test 1



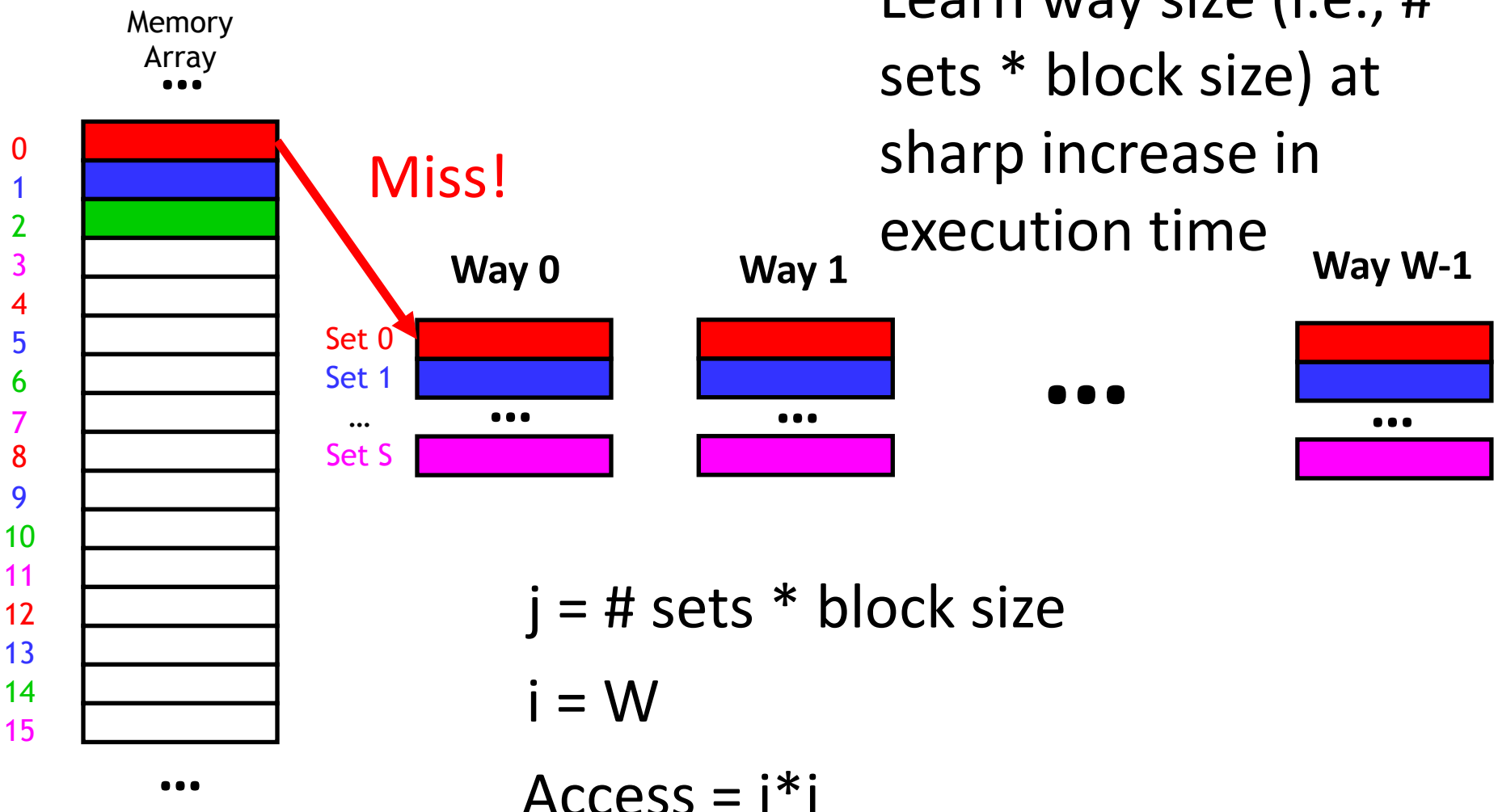
Since LRU, each access misses in iteration (k) after the first

j = # sets * block size

i = W

Access = i*j

Review: HW11 Cache Test 1



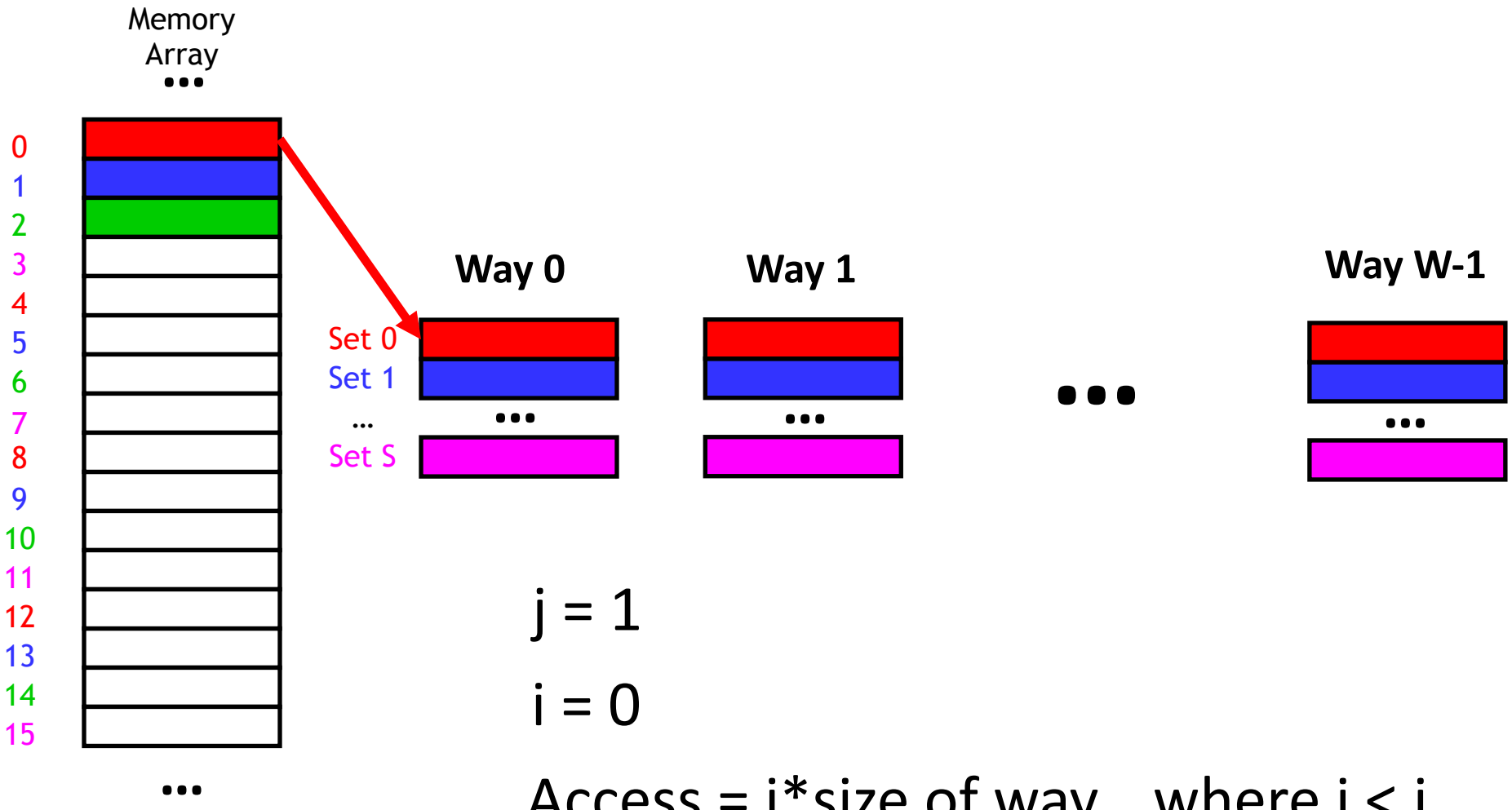
Learn way size (i.e., # sets * block size) at sharp increase in execution time

j = # sets * block size

i = W

Access = i*j

Review: HW11 Cache Test 2

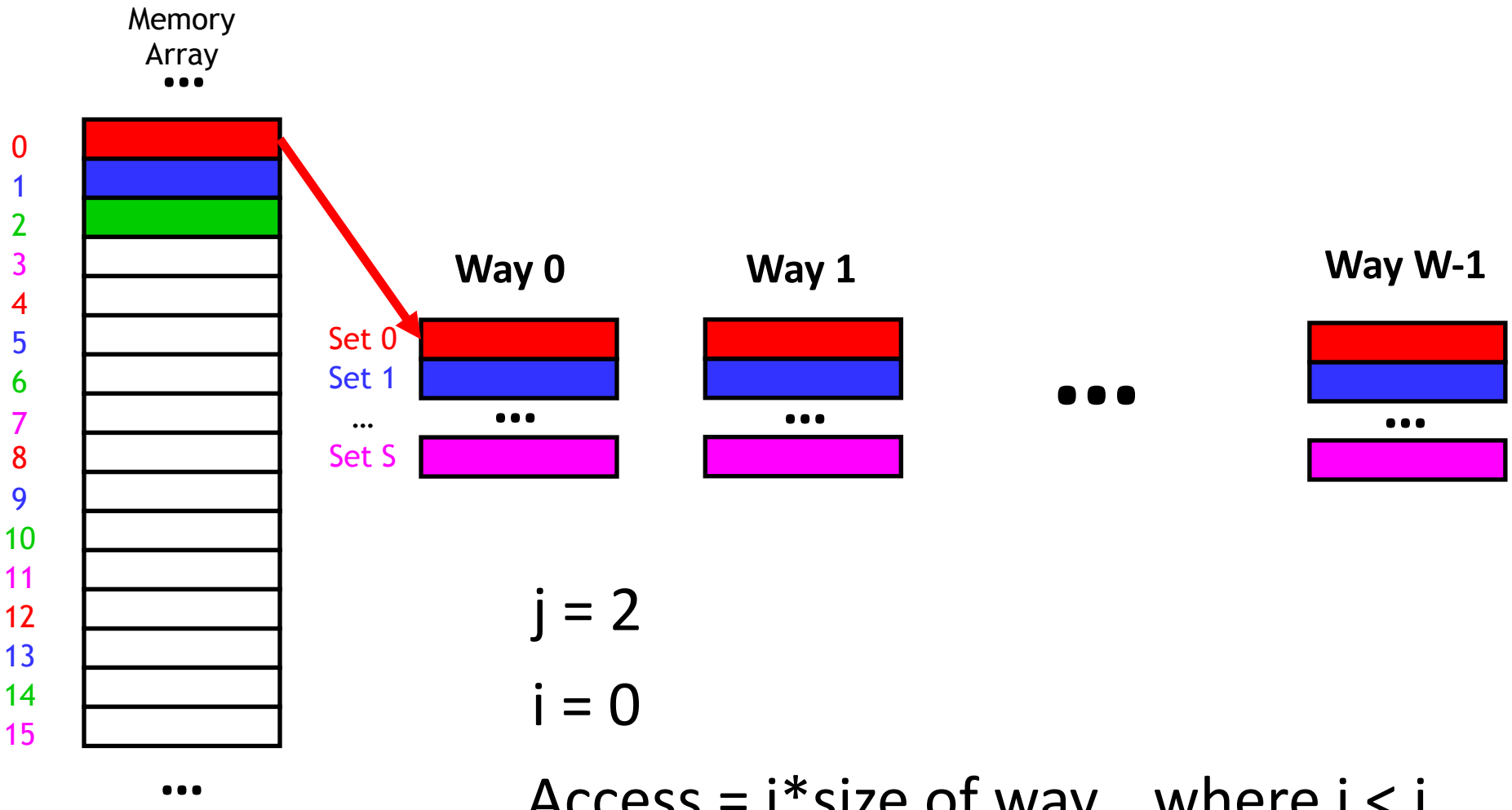


j = 1

i = 0

Access = i*size of way where i < j

Review: HW11 Cache Test 2

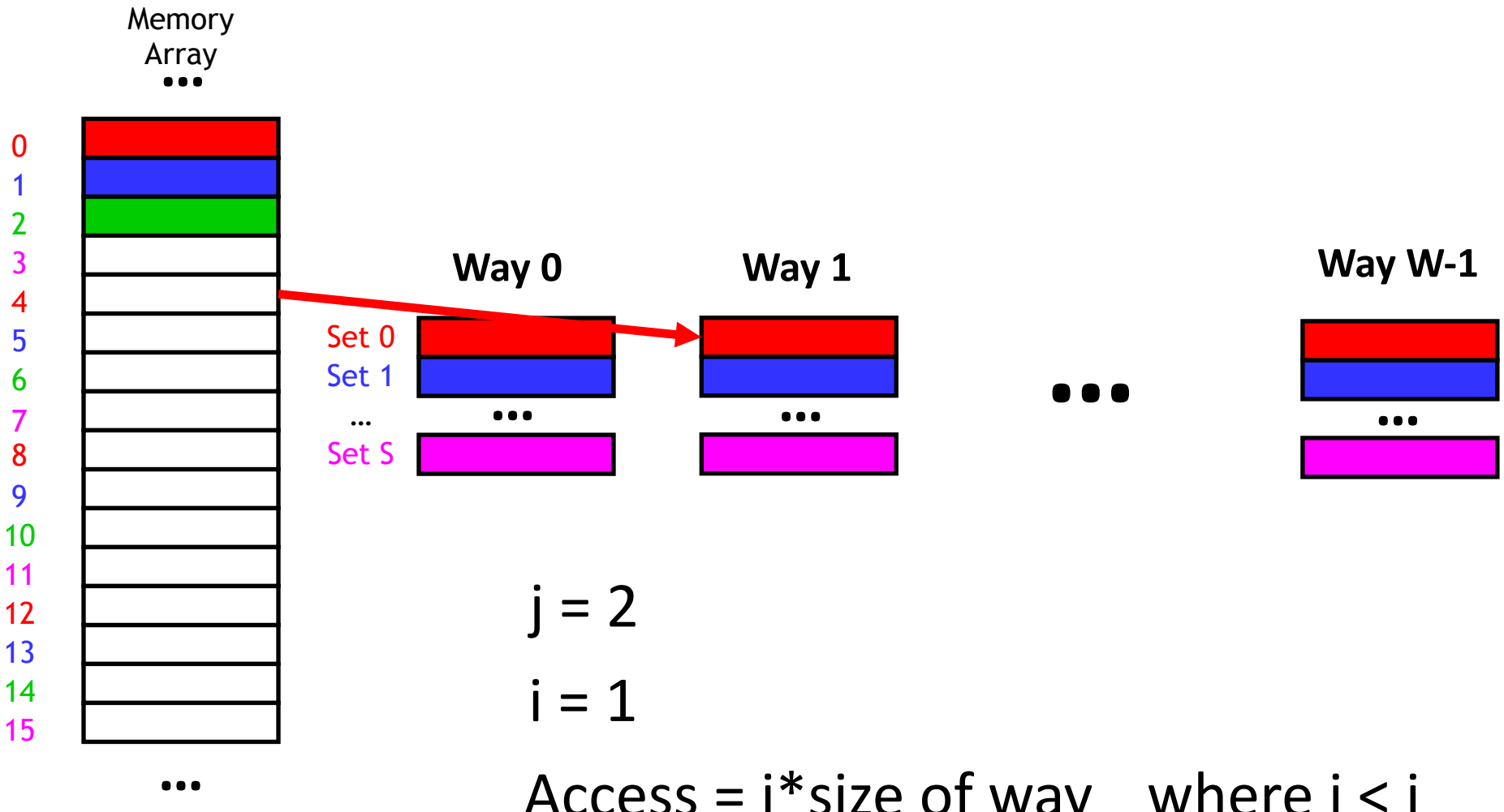


j = 2

i = 0

Access = i*size of way where i < j

Review: HW11 Cache Test 2

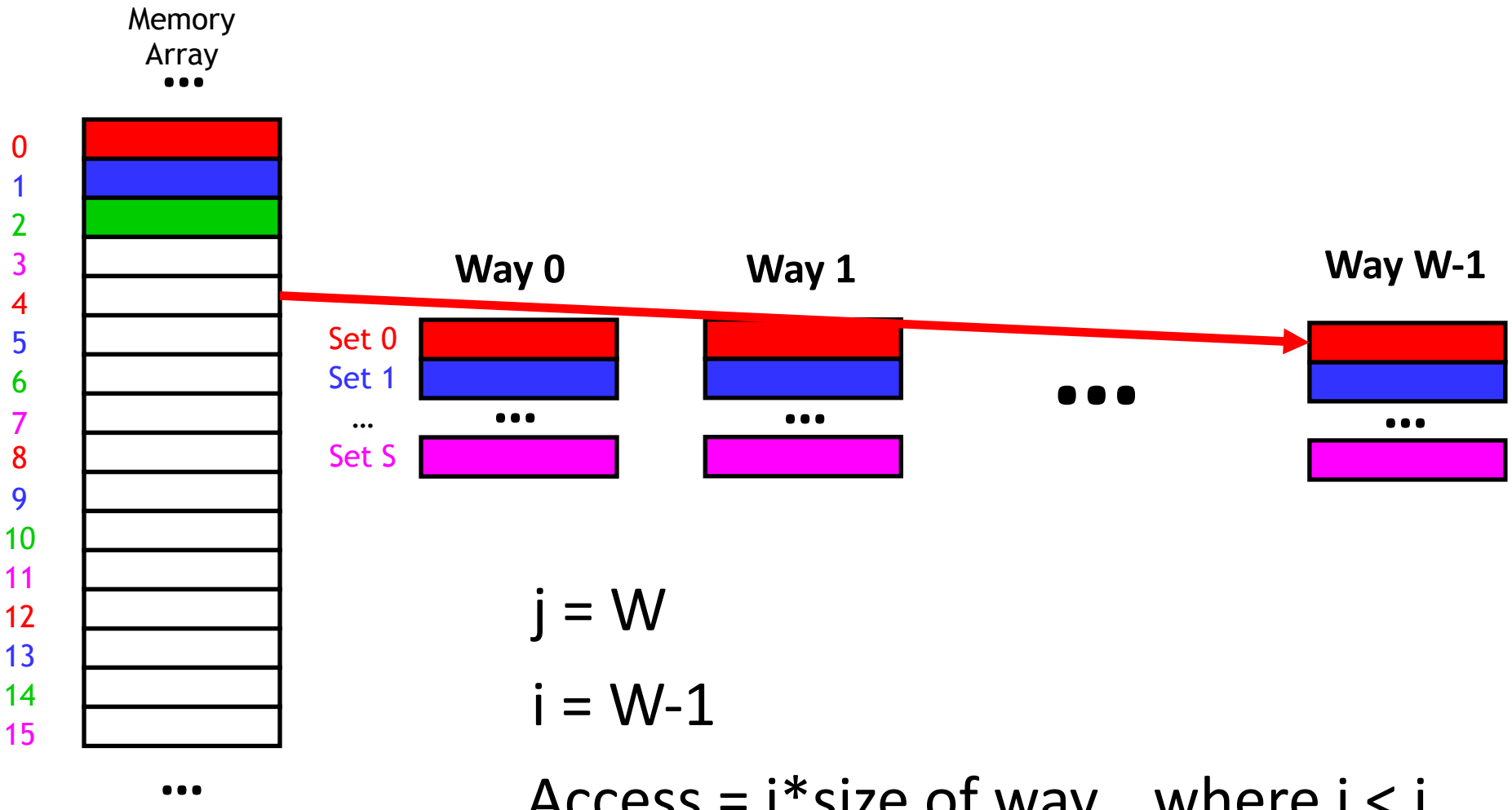


j = 2

i = 1

Access = i*size of way where i < j

Review: HW11 Cache Test 2

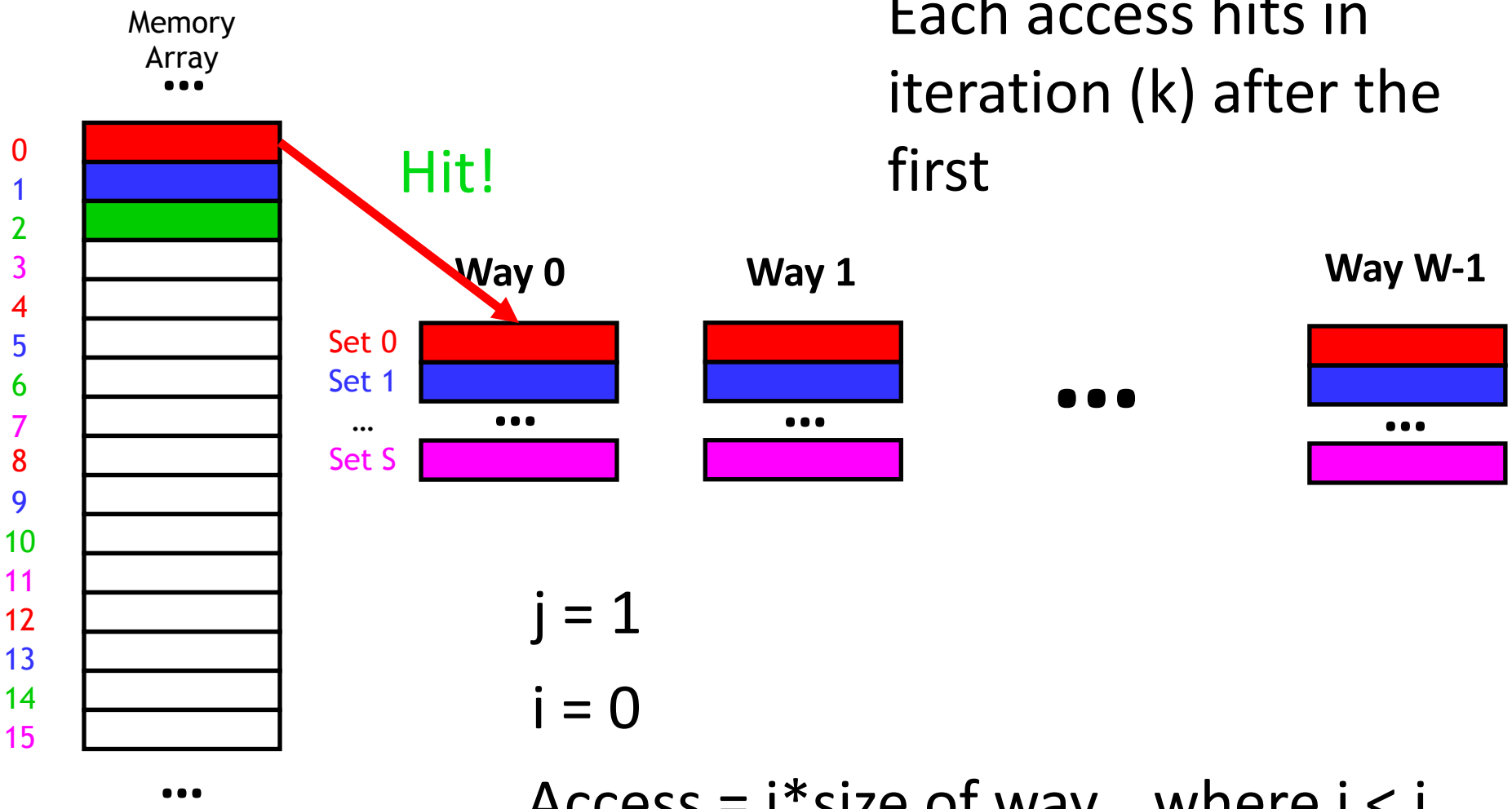


j = W

i = W-1

Access = i*size of way where i < j

Review: HW11 Cache Test 2



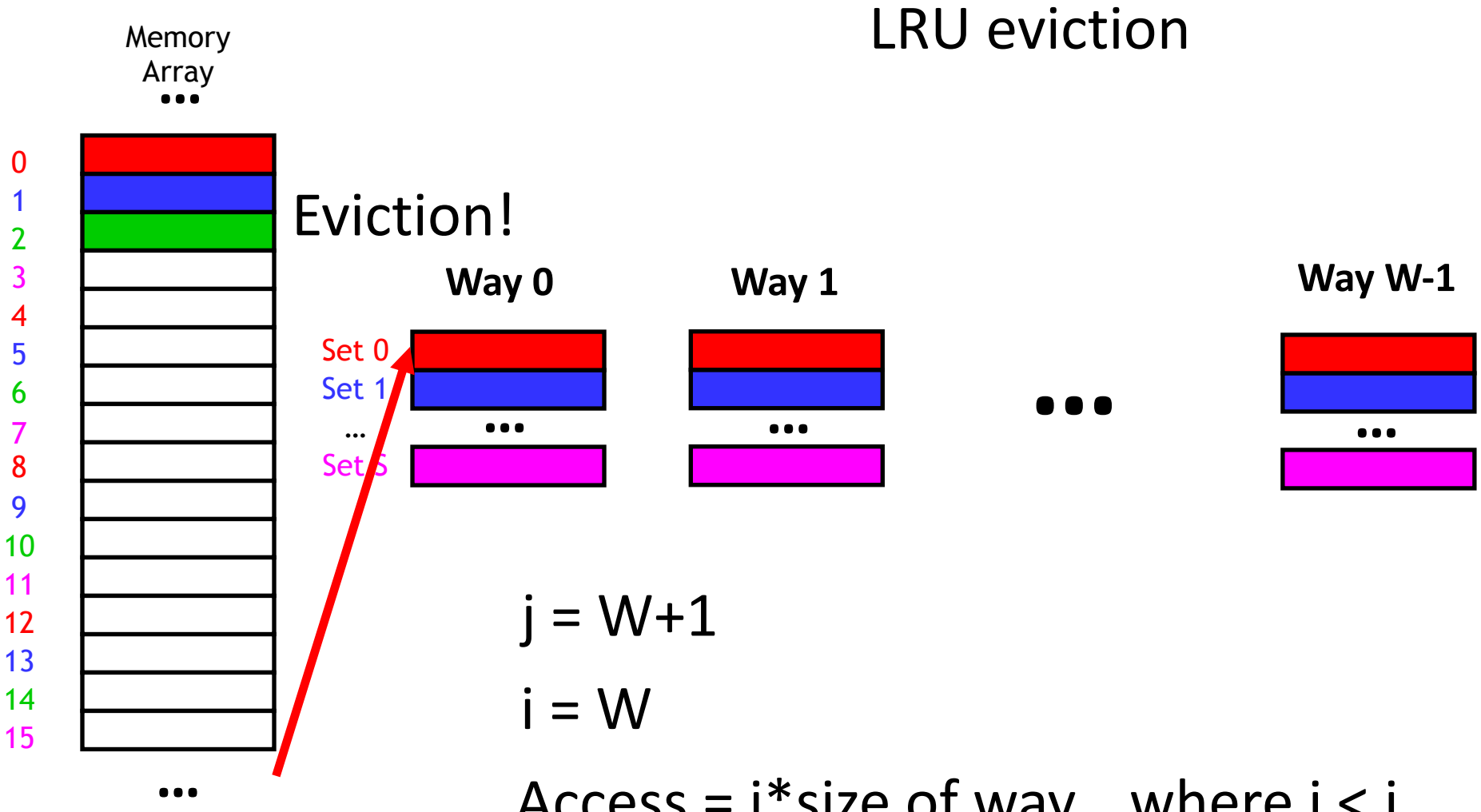
Each access hits in iteration (k) after the first

j = 1

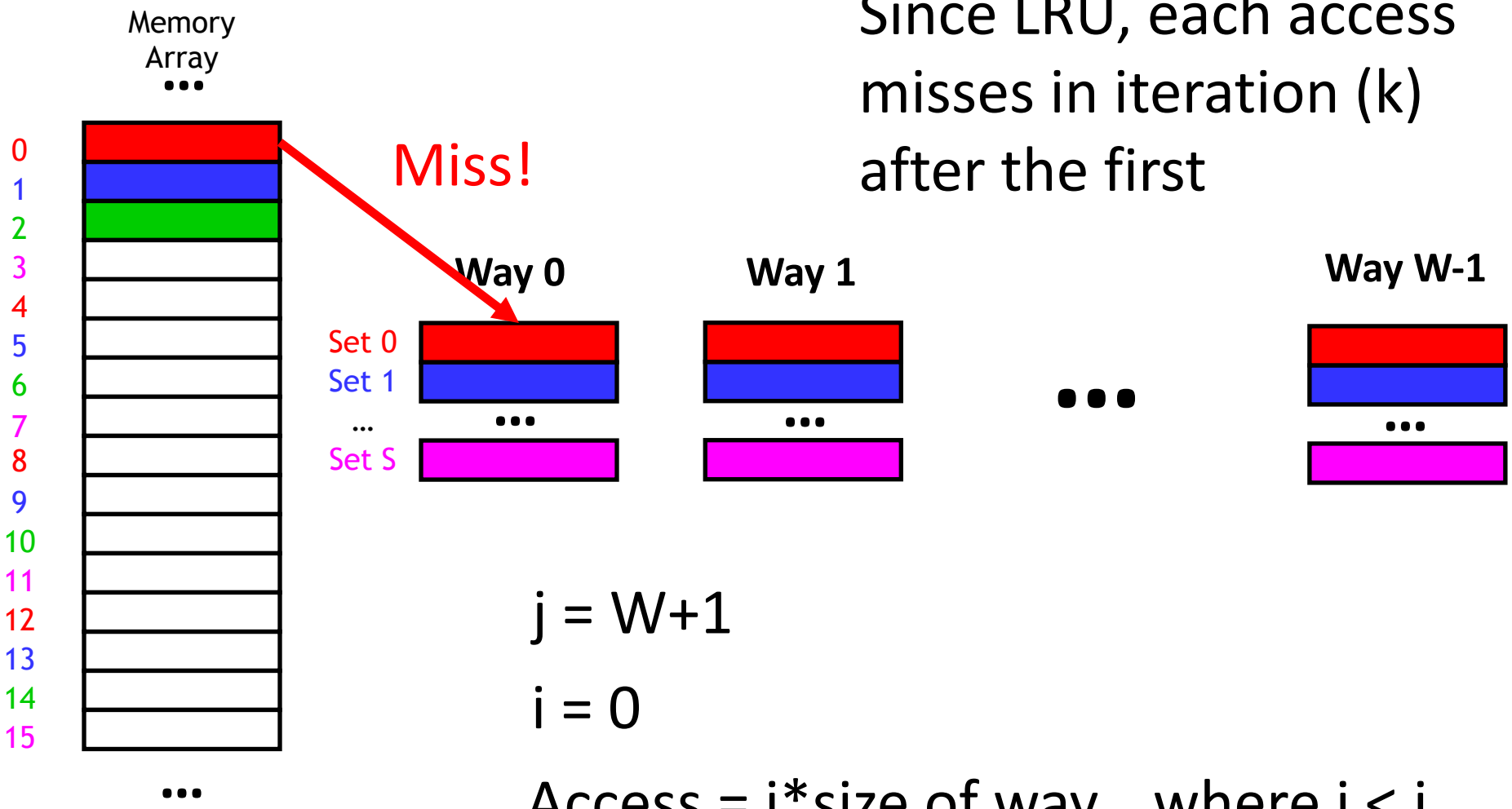
i = 0

Access = i*size of way where i < j

Review: HW11 Cache Test 2



Review: HW11 Cache Test 2



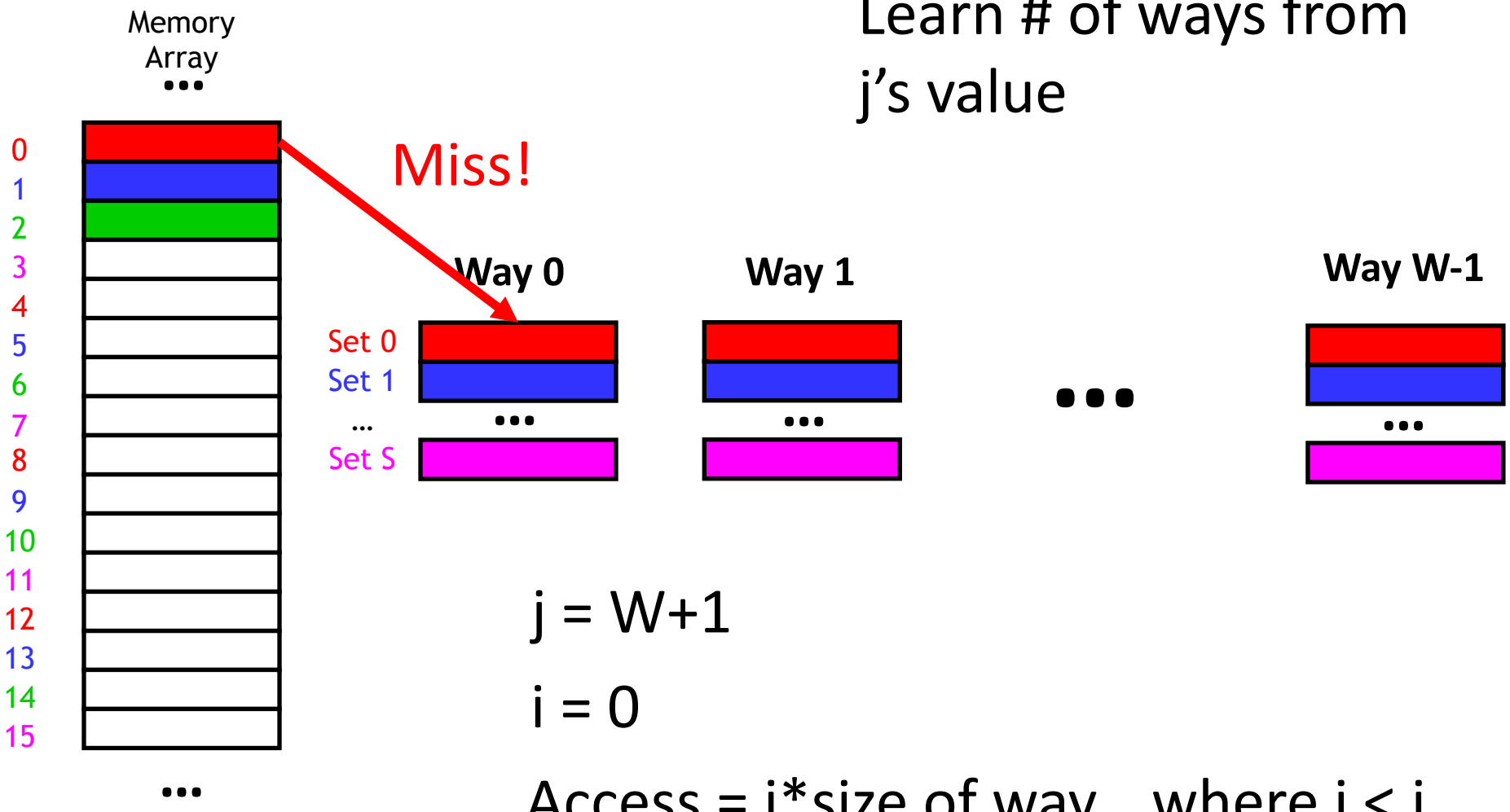
Since LRU, each access misses in iteration (k) after the first

j = W+1

i = 0

Access = i*size of way where i < j

Review: HW11 Cache Test 2



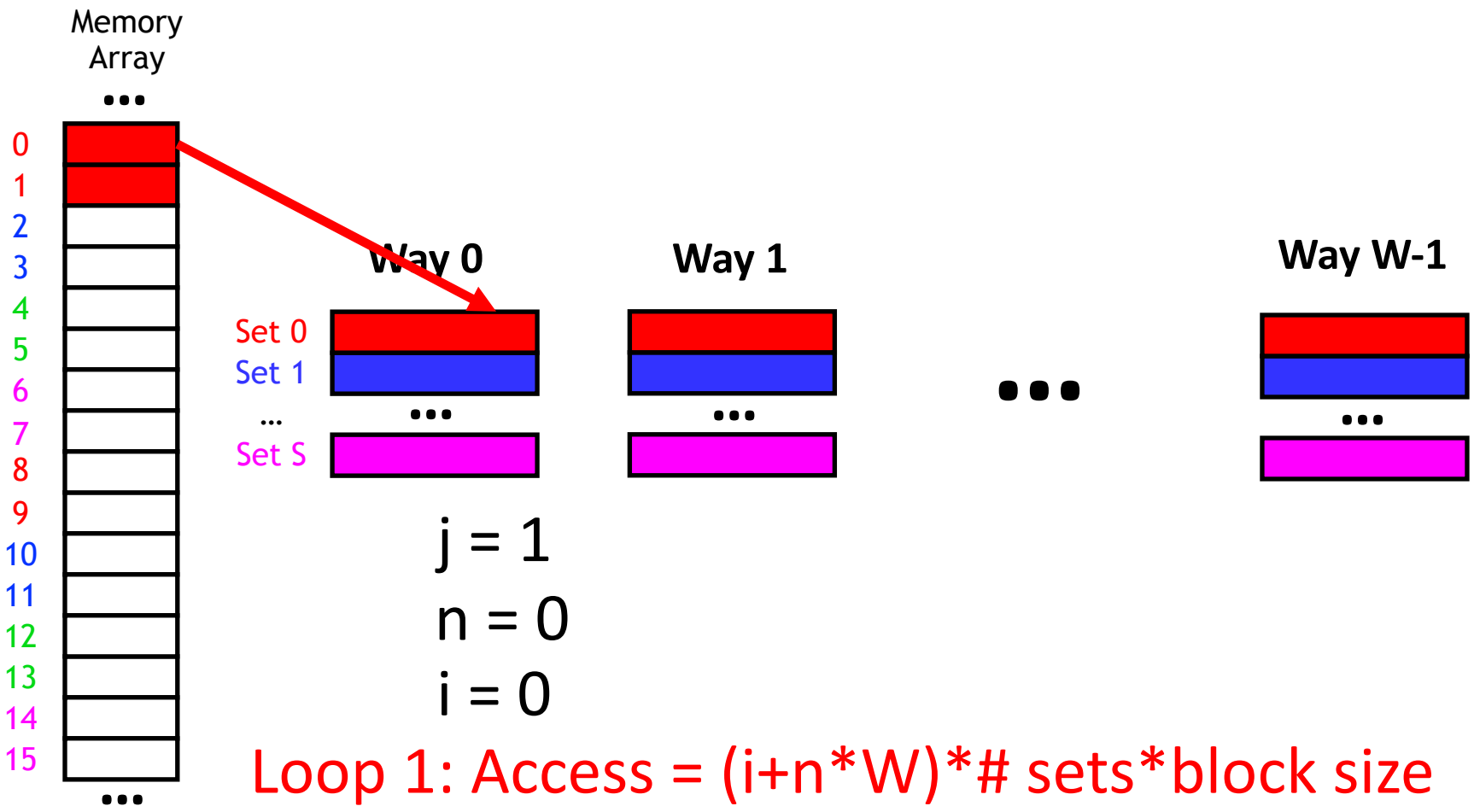
Learn # of ways from j's value

j = W+1

i = 0

Access = i*size of way where i < j

Review: HW11 Cache Test 3



j = 1

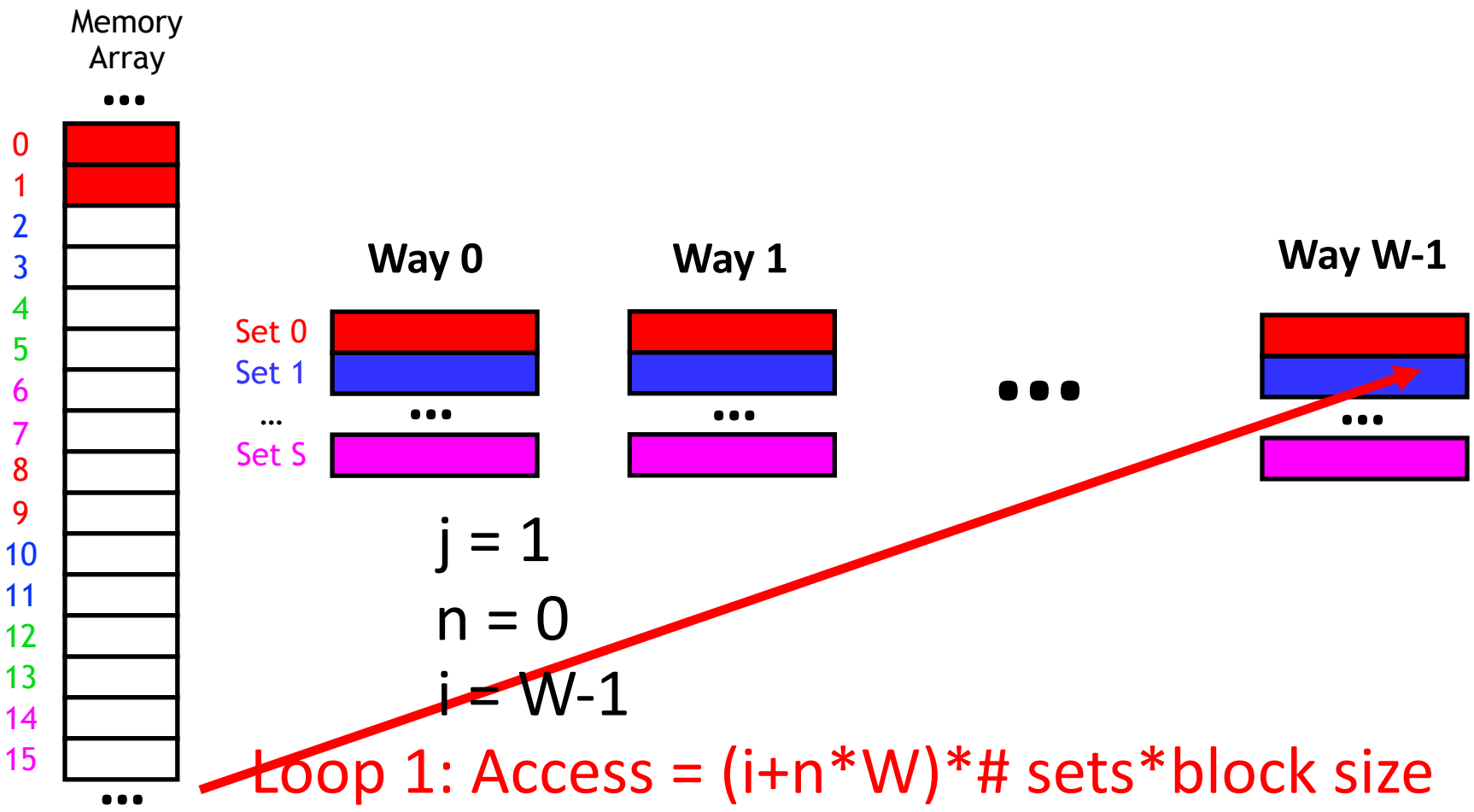
n = 0

i = 0

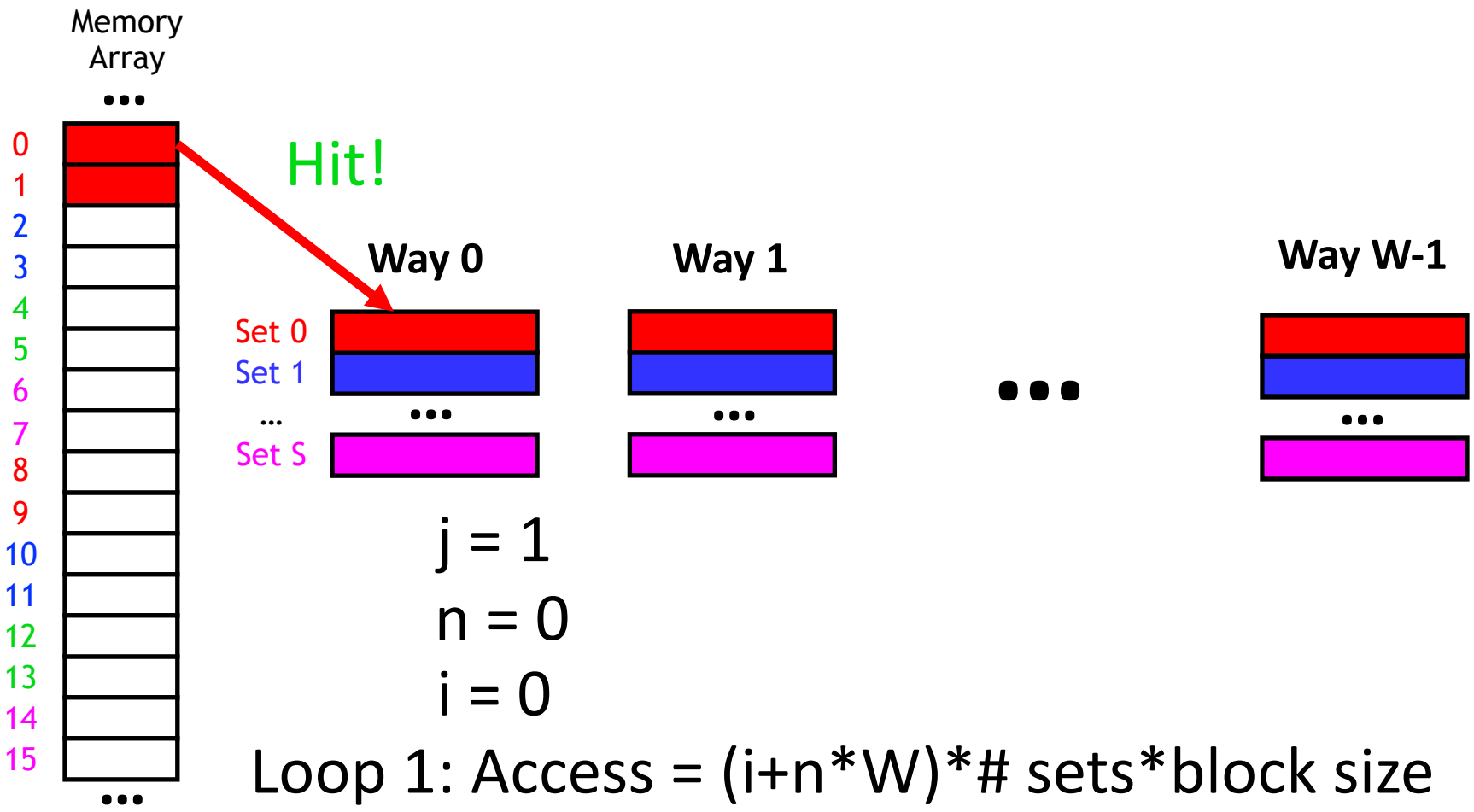
Loop 1: Access = (i+n*W)*# sets*block size

Loop 2: Access = (i+n*W)*# sets*block size+j

Review: HW11 Cache Test 3



Review: HW11 Cache Test 3



j = 1

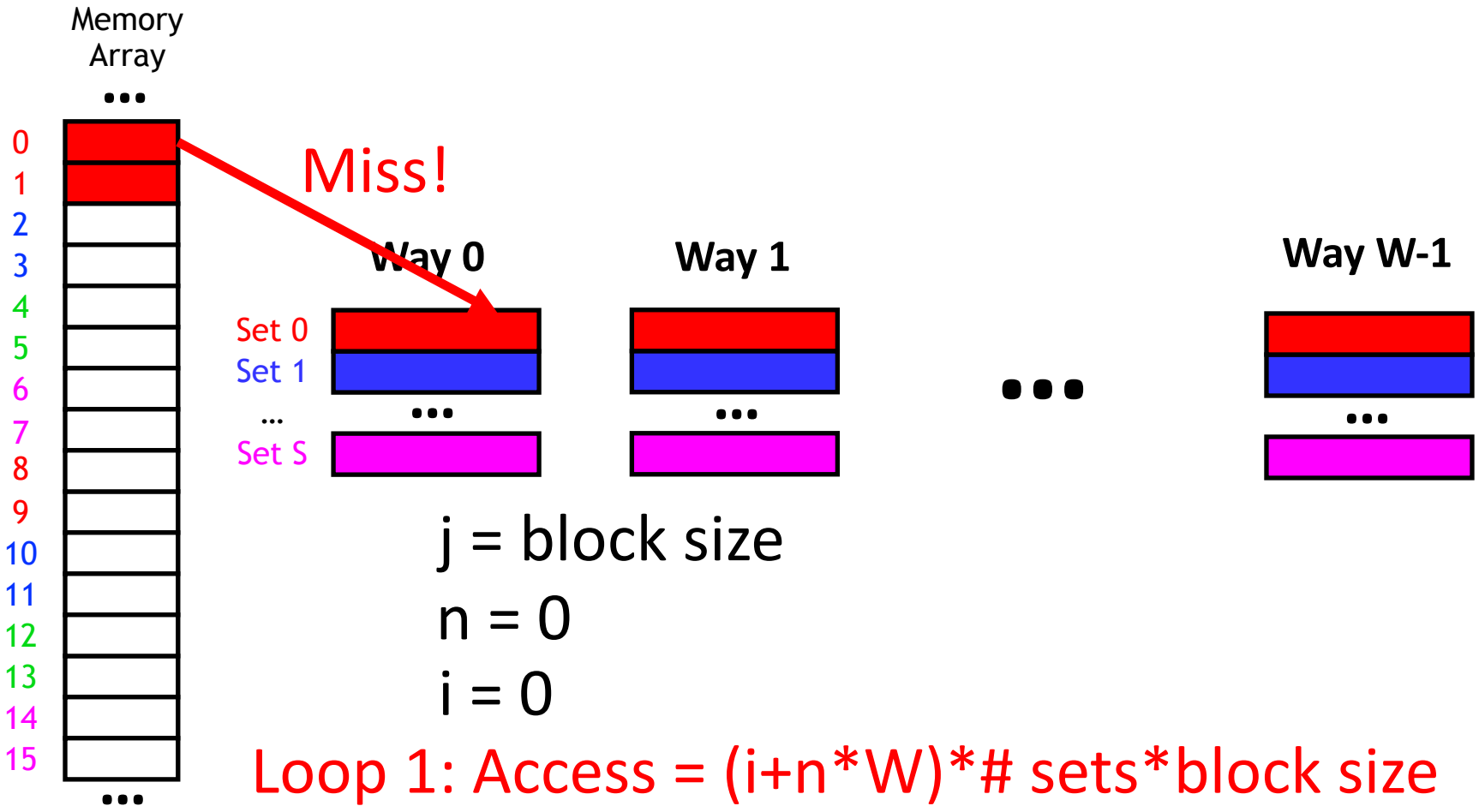
n = 0

i = 0

Loop 1: Access = (i+n*W)*# sets*block size

Loop 2: Access = (i+n*W)*# sets*block size+j

Review: HW11 Cache Test 3



j = block size

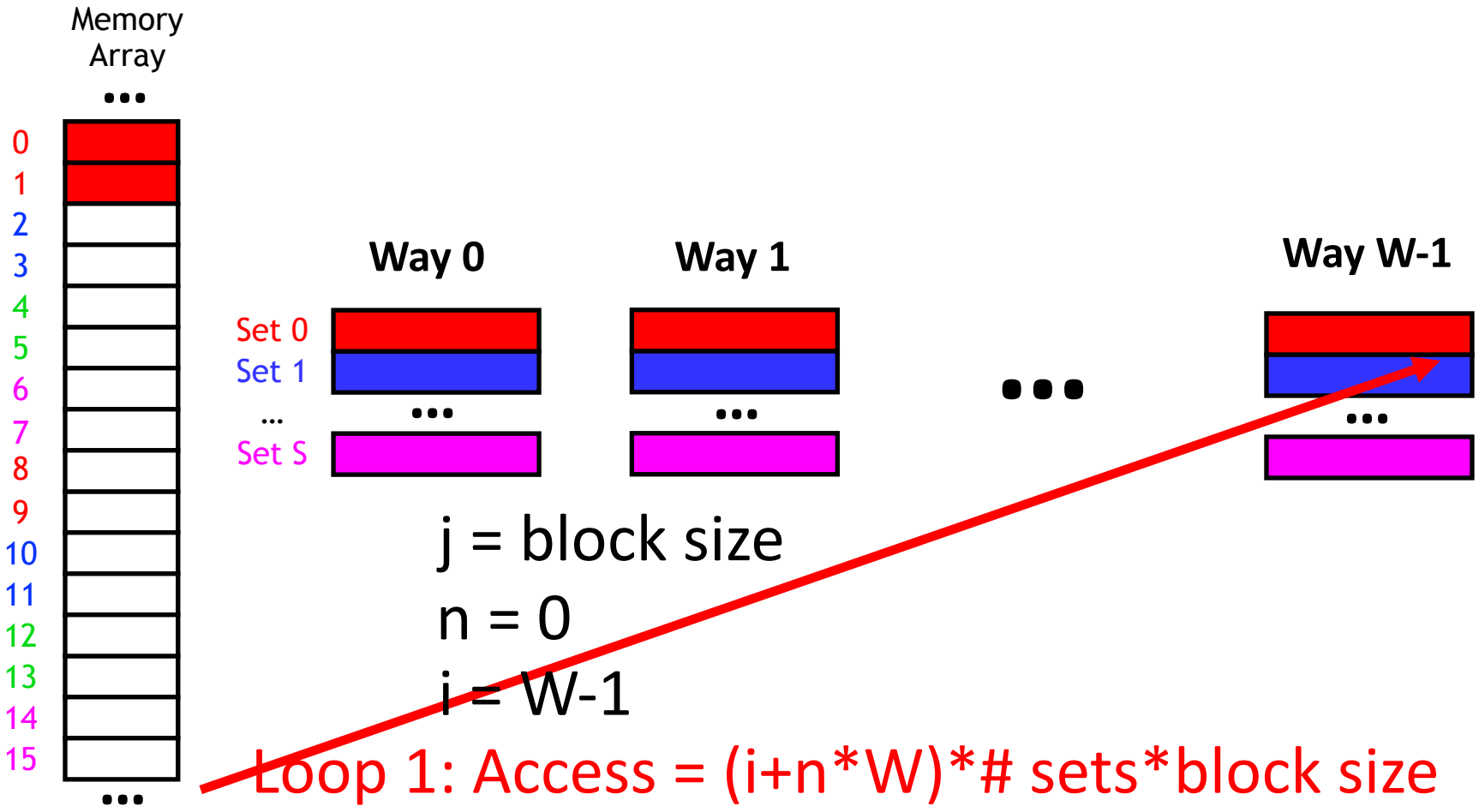
n = 0

i = 0

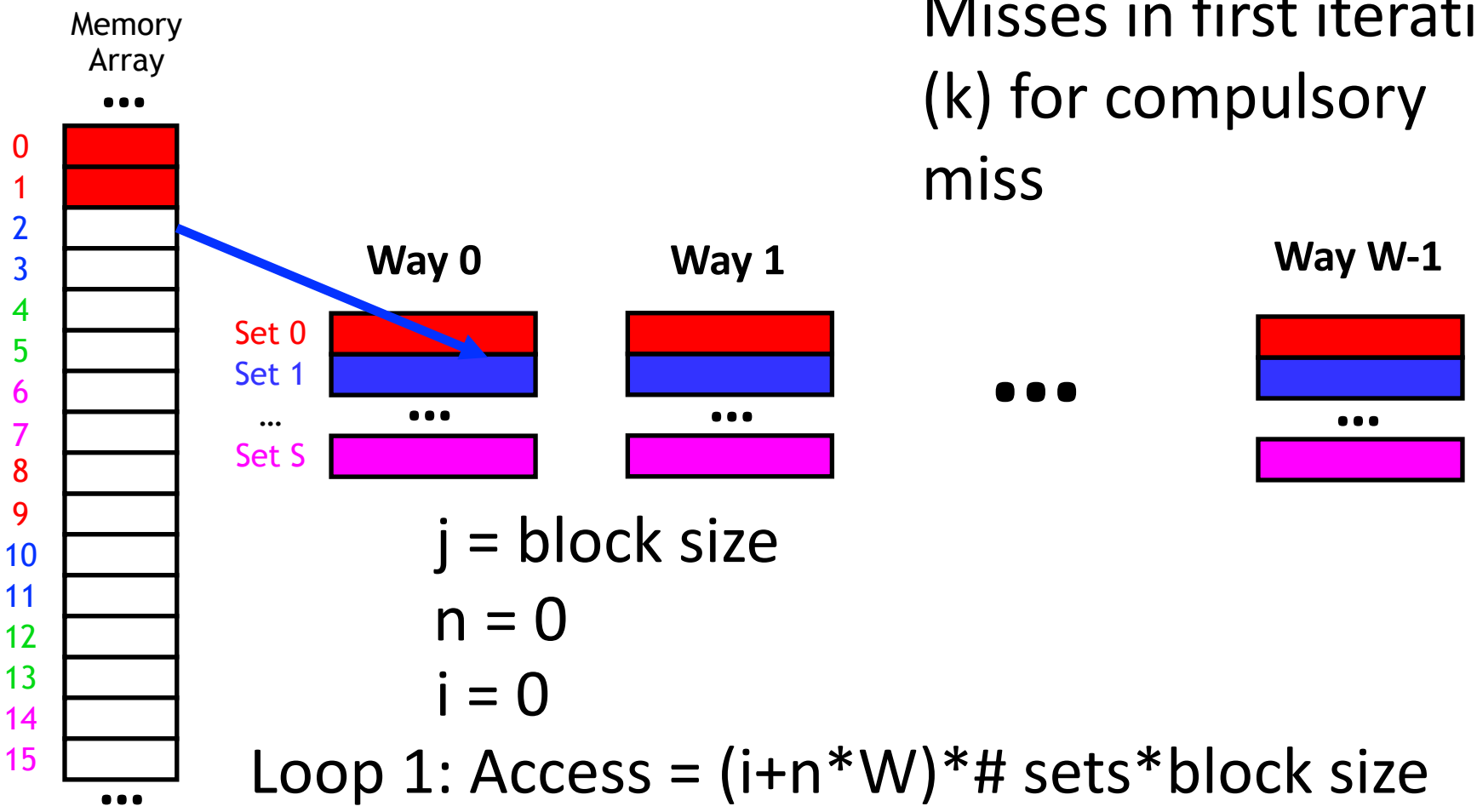
Loop 1: Access = (i+n*W)*# sets*block size

Loop 2: Access = (i+n*W)*# sets*block size+j

Review: HW11 Cache Test 3



Review: HW11 Cache Test 3



Misses in first iteration (k) for compulsory miss

j = block size

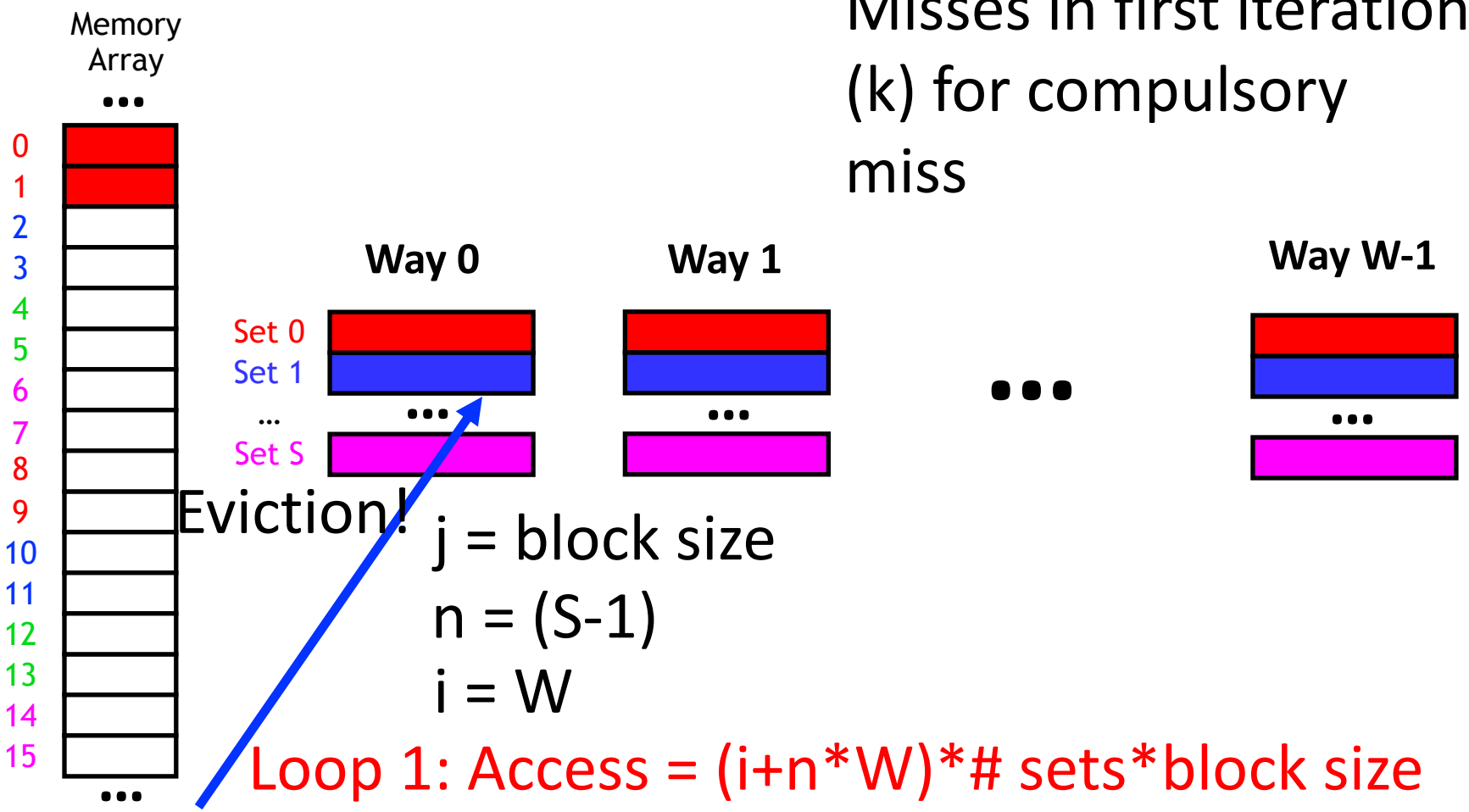
n = 0

i = 0

Loop 1: Access = (i+n*W)*# sets*block size

Loop 2: Access = (i+n*W)*# sets*block size+j

Review: HW11 Cache Test 3



Misses in first iteration (k) for compulsory miss

j = block size

n = (S-1)

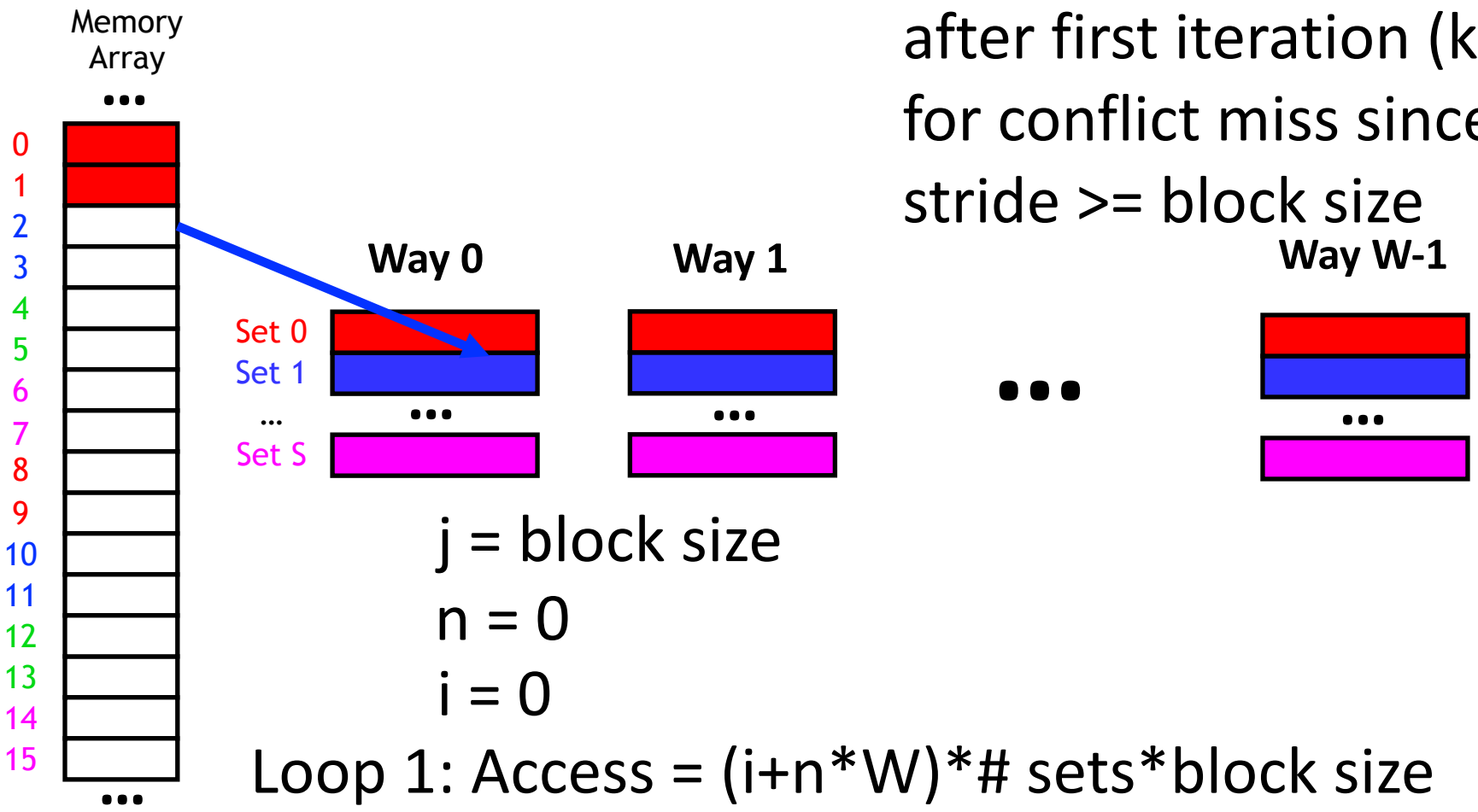
i = W

Loop 1: Access = (i+n*W)*# sets*block size

Loop 2: Access = (i+n*W)*# sets*block size+j

Review: HW11 Cache Test 3

Misses in all iterations after first iteration (k) for conflict miss since stride >= block size



j = block size

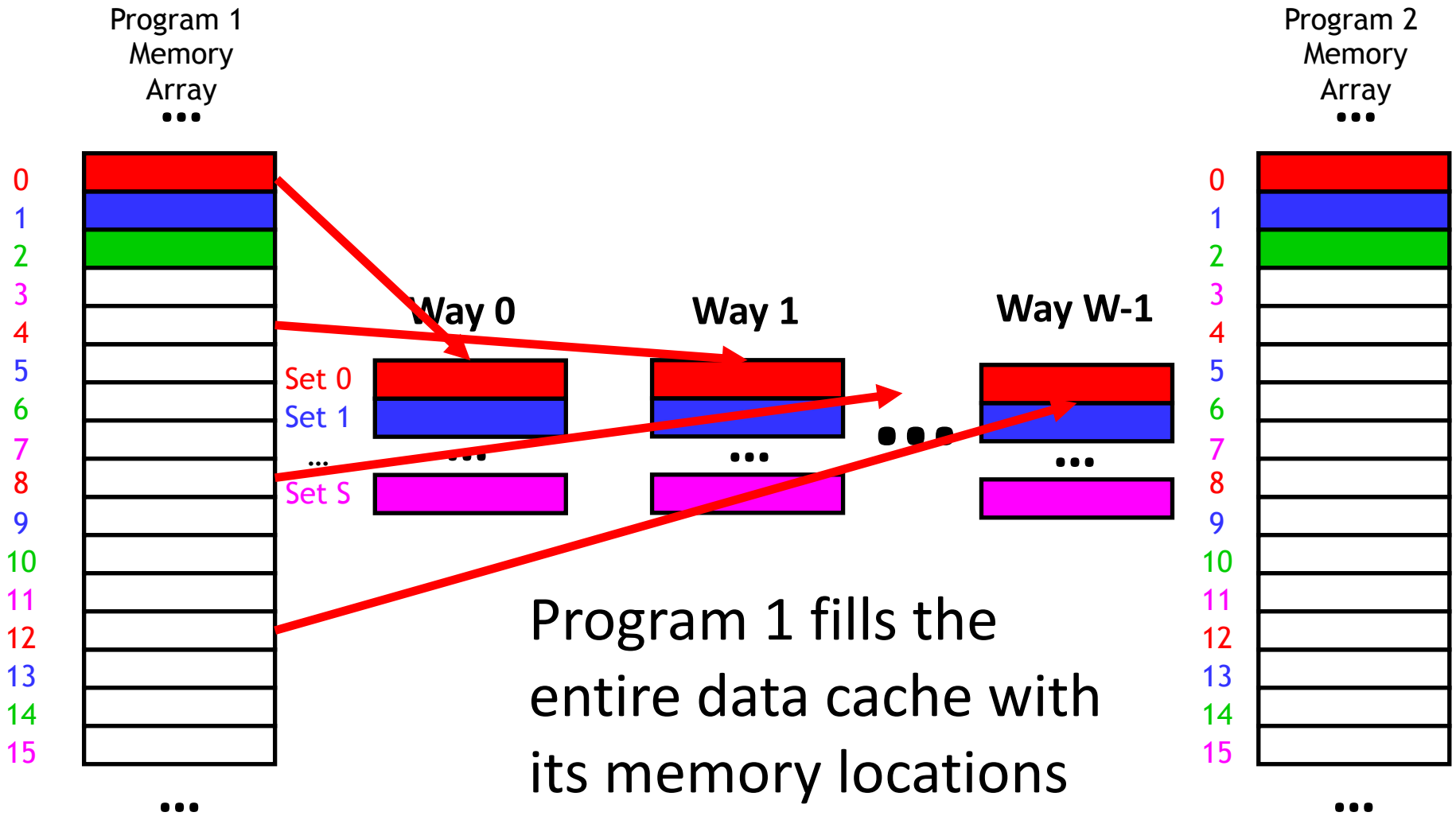
n = 0

i = 0

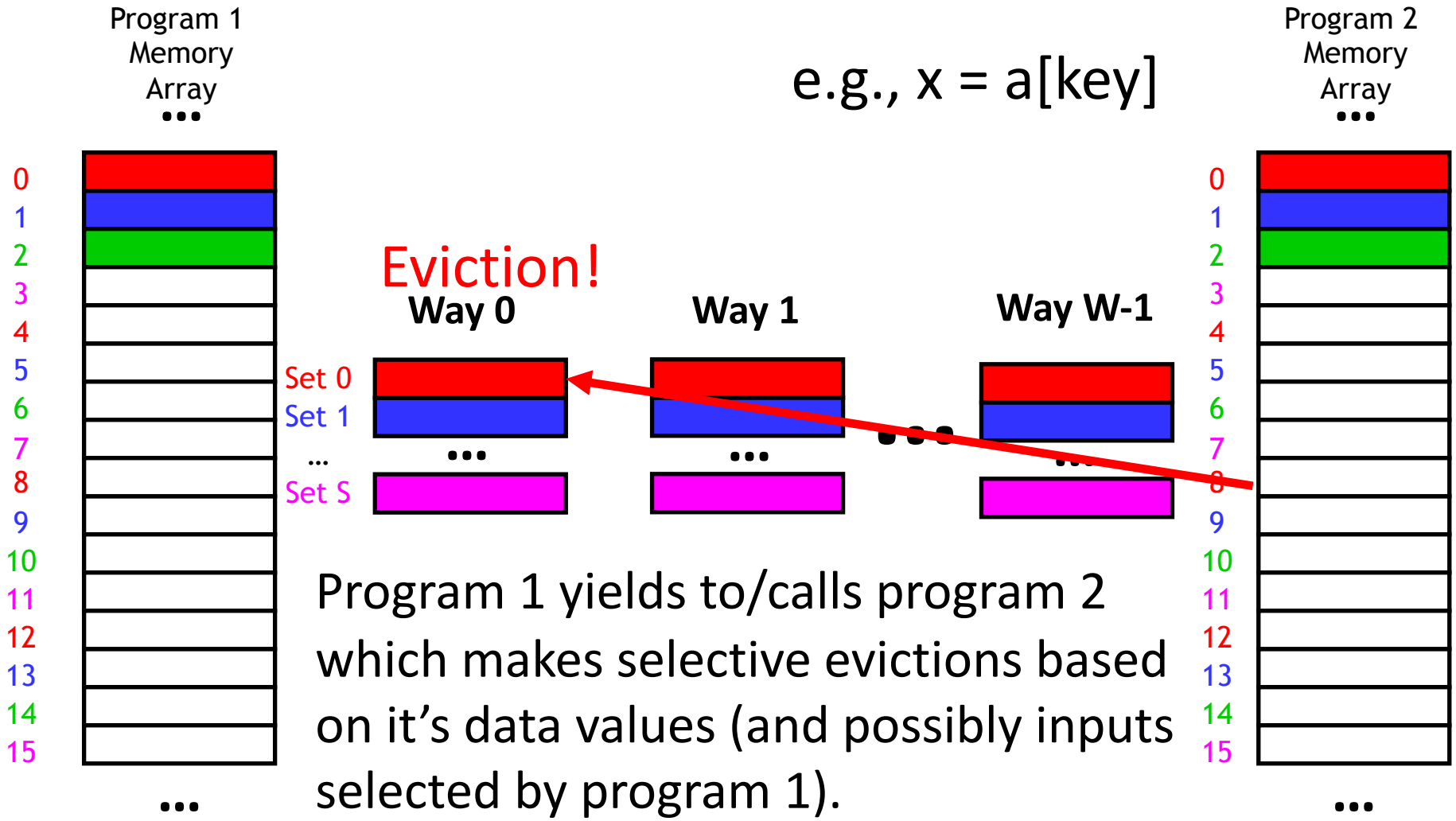
Loop 1: Access = (i+n*W)*# sets*block size

Loop 2: Access = (i+n*W)*# sets*block size+j

A Thought Experiment

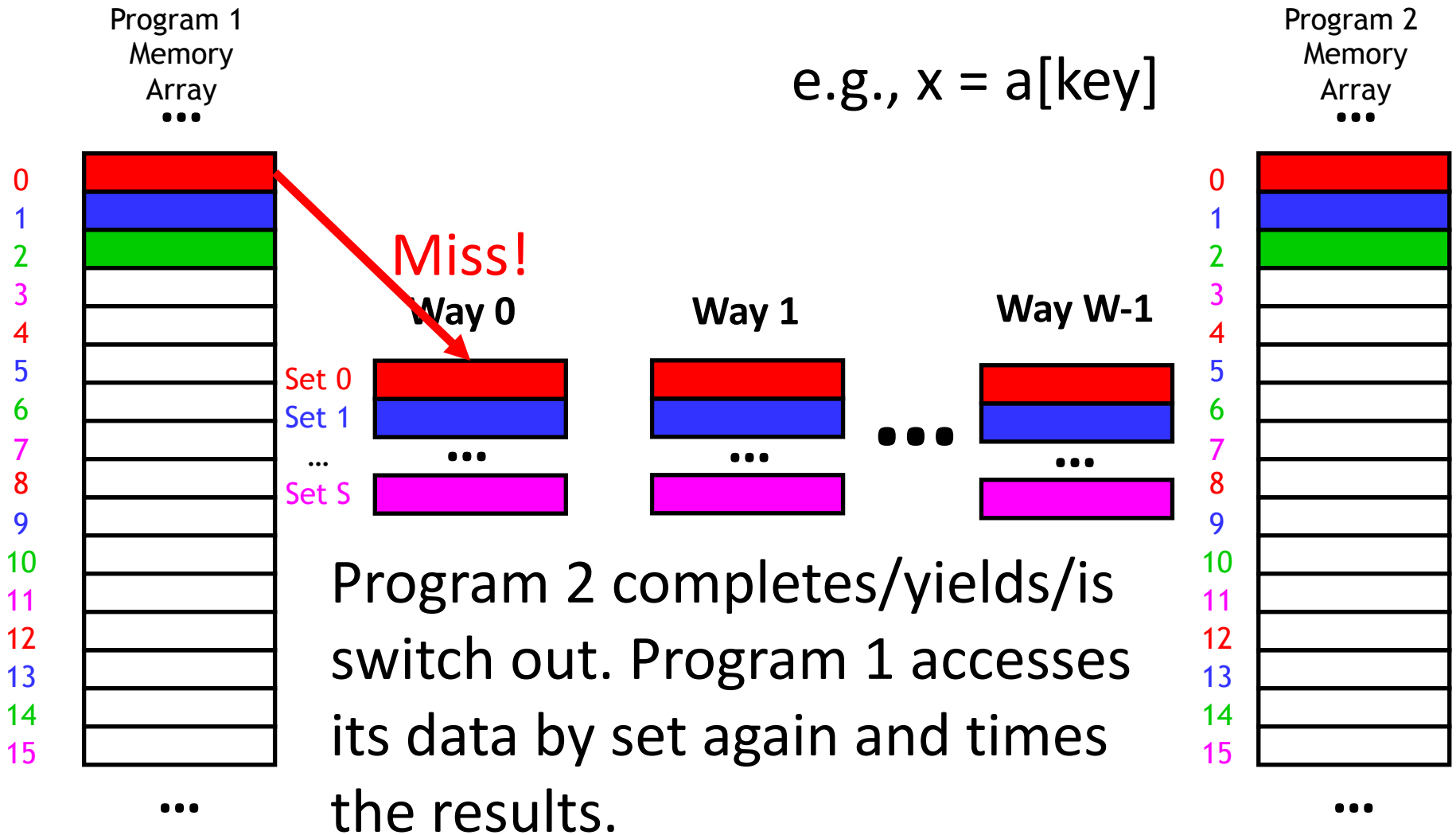


A Thought Experiment



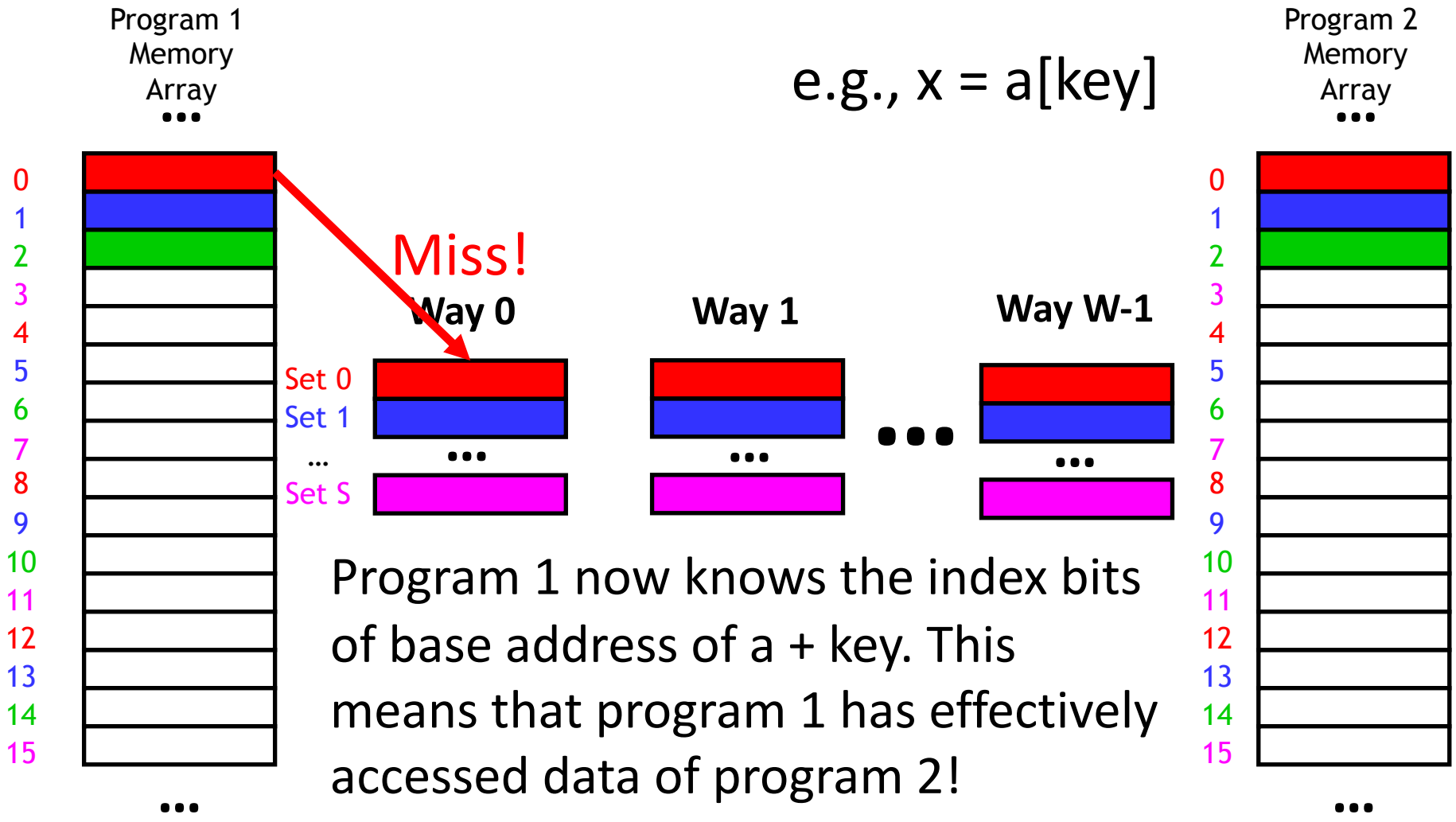
Program 1 yields to/calls program 2 which makes selective evictions based on it's data values (and possibly inputs selected by program 1).

A Thought Experiment

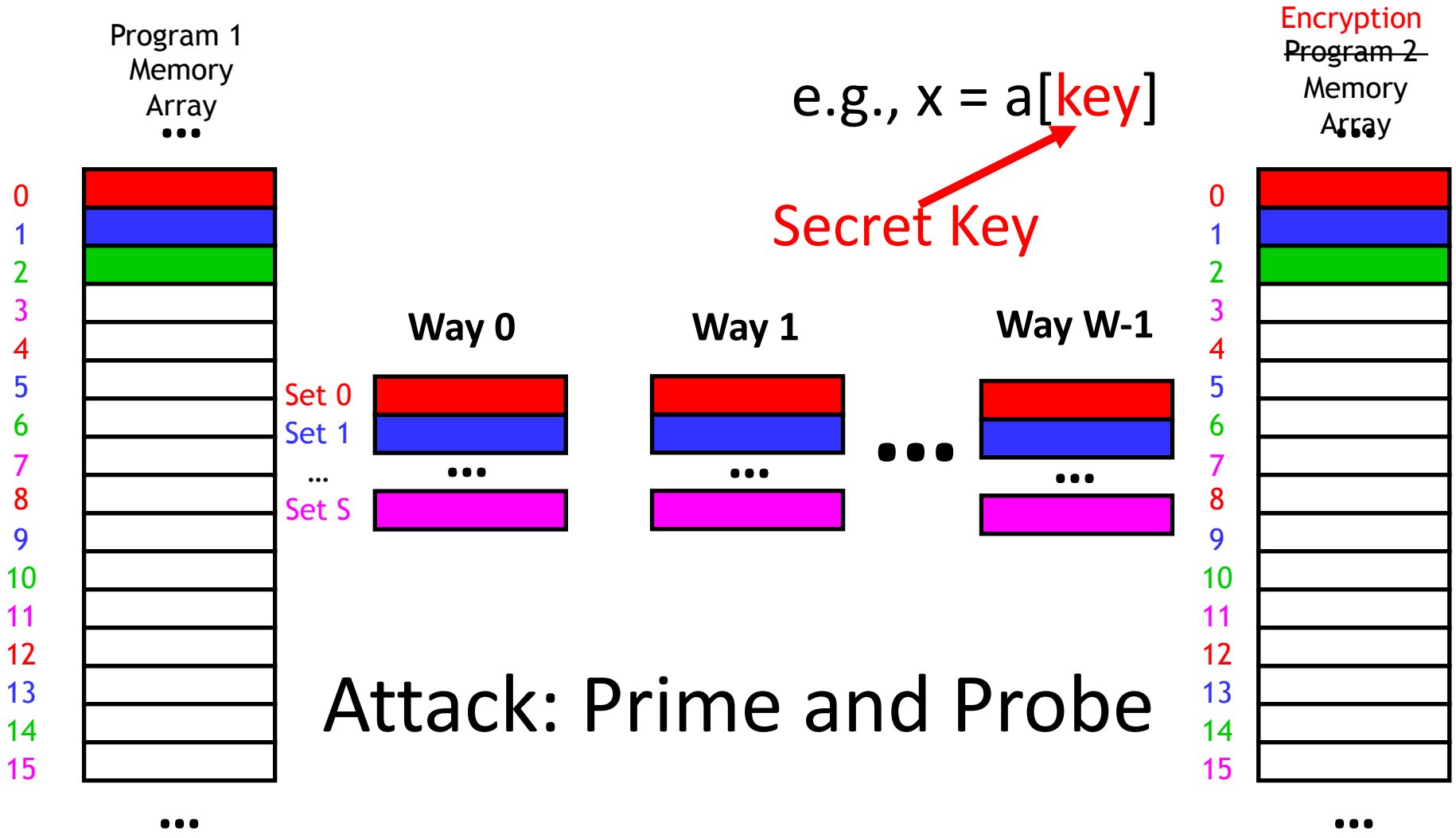


Program 2 completes/yields/is switch out. Program 1 accesses its data by set again and times the results.

A Thought Experiment



A Thought Experiment



This Course's Focus

Dear Bored in Lecture,

- Case and point from recent news:

- The device you design is not reliable?
- The device you design doesn't scale?

Sincerely,
Prof Duwe

Specter Attack

```
if (x < array1_size) {
    y = array2[array1[x] * 256]
}
```

1. Attacker primes micro-architecture
   - Train branch predictor/BTB
   - Prime cache side-channel (prime)
2. Victim loads secret after mispredicted branch
   - Load to secret shouldn't trap since it is speculative (exception not handled until instruction is non-speculative)
3. Victim saves secret in μ-arch state
4. Attacker recalls secret from μ-arch state (probe)

Specter Attack

Victim Code Gadget

```
if (x < array1_size) {
    y = array2[array1[x] * 256]
}
```

1. Attacker primes micro-architecture
   - Train branch predictor/BTB
   - Prime cache side-channel (prime)
2. Victim loads secret after mispredicted branch
   - Load to secret shouldn't trap since it is speculative (exception not handled until instruction is non-speculative)
3. Victim saves secret in μ-arch state
4. Attacker recalls secret from μ-arch state (probe)

Specter Attack

Attacker calls victim many times where x < array1_size

```
if (x < array1_size) {
    y = array2[array1[x] * 256]
}
```

1. Attacker primes micro-architecture
   - Train branch predictor/BTB
   - Prime cache side-channel (prime)
2. Victim loads secret after mispredicted branch
   - Load to secret shouldn't trap since it is speculative (exception not handled until instruction is non-speculative)
3. Victim saves secret in μ-arch state
4. Attacker recalls secret from μ-arch state (probe)

Specter Attack

Attacker calls victim once where x > array1_size, but BTB predicts x < array1_size

```
if (x < array1_size) {
    y = array2[array1[x] * 256]
}
```

1. Attacker primes micro-architecture
   - Train branch predictor/BTB
   - Prime cache side-channel (prime)
2. Victim loads secret after mispredicted branch
   - Load to secret shouldn't trap since it is speculative (exception not handled until instruction is non-speculative)
3. Victim saves secret in μ-arch state
4. Attacker recalls secret from μ-arch state (probe)

Specter Attack

Victim accesses arbitrary data in victims address space

```
if (x < array1_size) {
    y = array2[array1[x] * 256]
}
```

1. Attacker primes micro-architecture
   - Train branch predictor/BTB
   - Prime cache side-channel (prime)
2. Victim loads secret after mispredicted branch
   - Load to secret shouldn't trap since it is speculative (exception not handled until instruction is non-speculative)
3. Victim saves secret in μ-arch state
4. Attacker recalls secret from μ-arch state (probe)

Specter Attack

Victim evicts attackers data based on victims data which shouldn't have been accessed

```
if (x < array1_size) {
    y = array2[array1[x] * 256]
}
```

1. Attacker primes micro-architecture
   - Train branch predictor/BTB
   - Prime cache side-channel (prime)
2. Victim loads secret after mispredicted branch
   - Load to secret shouldn't trap since it is speculative (exception not handled until instruction is non-speculative)
3. Victim saves secret in μ-arch state
4. Attacker recalls secret from μ-arch state (probe)

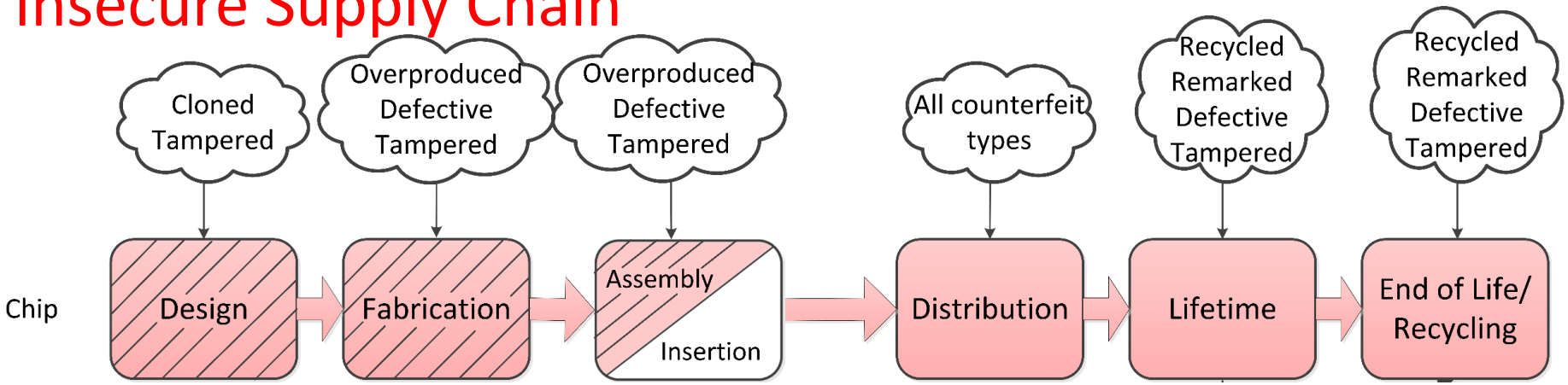
Specter Attack

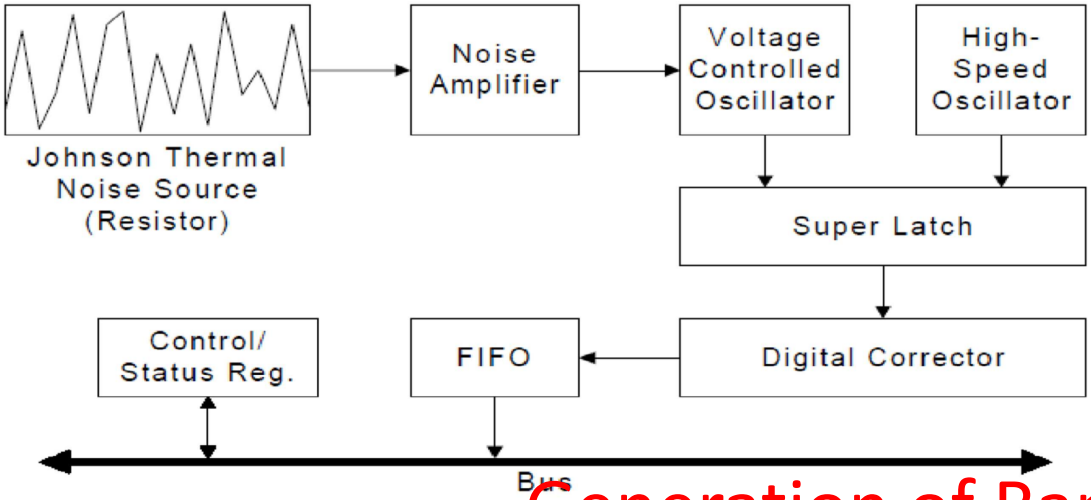
```
if (x < array1_size) {
    y = array2[array1[x] * 256]
}
```

1. Attacker primes micro-architecture
   - Train branch predictor/BTB
   - Prime cache side-channel (prime)
2. Victim loads secret after mispredicted branch
   - Load to secret shouldn't trap since it is speculative (exception not handled until instruction is non-speculative)
3. Victim saves secret in μ-arch state
4. Attacker recalls secret from μ-arch state (probe)

Many Other HW Sec Issues…

Insecure Supply Chain



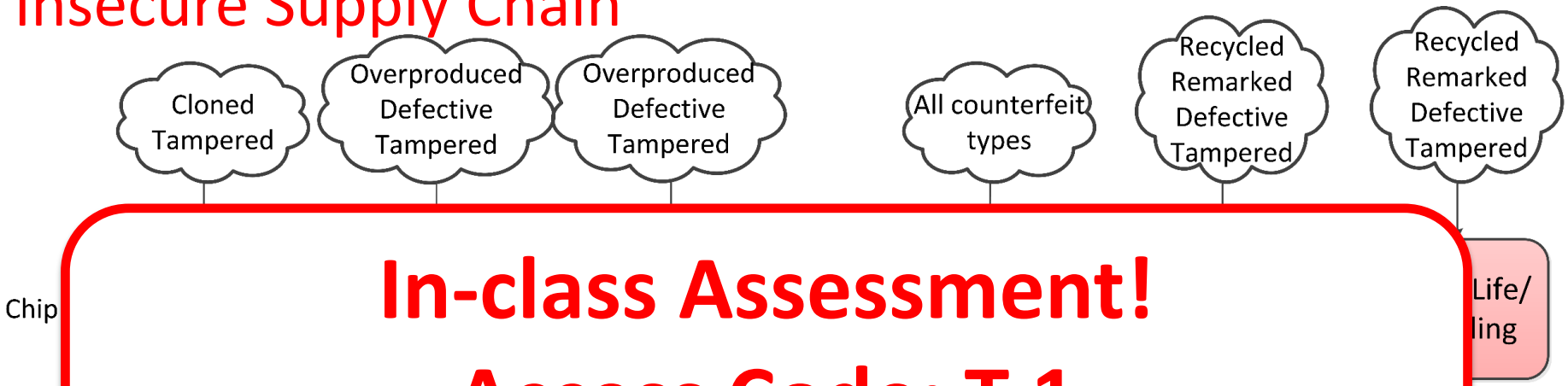


Generation of Random Numbers

Block Diagram of Intel RNG

Many Other HW Sec Issues…

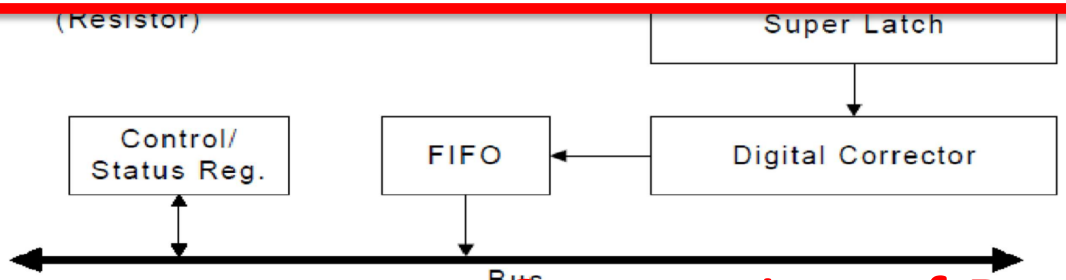
Insecure Supply Chain



In-class Assessment!

Access Code: T-1

Note: sharing access code to those outside of classroom or using access code while outside of classroom is considered cheating



Generation of Random Numbers

Block Diagram of Intel RNG

Acknowledgments

- These slides contain material developed and copyright by:
  - Mark Hill (UW Madison)
  - Marten van Dijk (UConn)