Effective Padding of Multi-Dimensional Arrays to Avoid Cache Conflict Misses

Changwan Hong, Wenlei Bao, Albert Cohen, Sriram Krishnamoorthy, Louis-Noel Pouchet, J. Ramanujan, Fabrice Rastello, P. Sadayappan

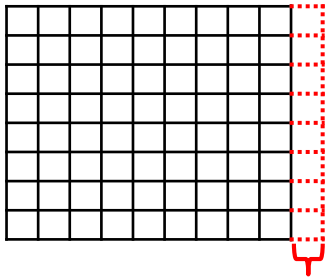
June 15th, 2016

THE OHIO STATE UNIVERSITY

Motivation and Problem Statement

- Typical processors: hierarchical set-associative caches
- Optimize cache set usage and minimize conflict misses => Better performance

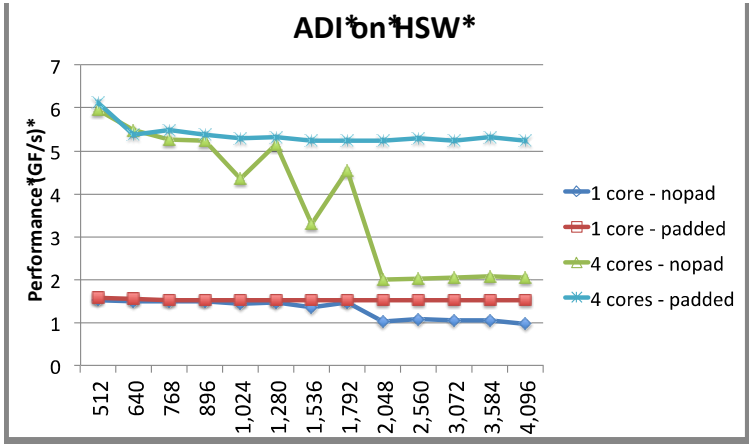
  → A solution : Array Padding



Problem statement: is it possible to find the smallest padding which completely eliminates conflict misses in a hierarchy of set-associative caches?

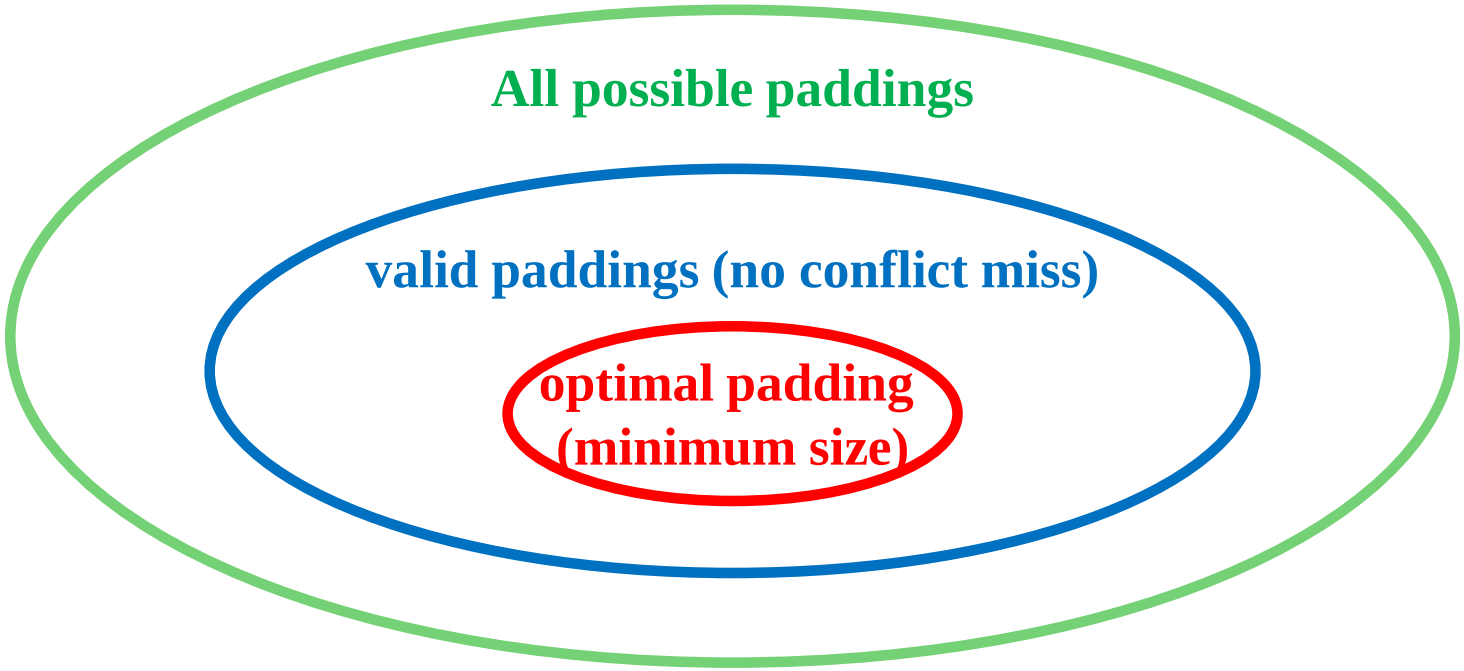
- Reducing/avoiding conflict misses using array padding can have significant performance impact

  ADI on Haswell: padding maintain GF/s for various problem sizes



Valid Padding: Eliminating Conflict Misses

- Conflict miss: cache miss because of a sub-optimal cache usage
- Valid padding: a padding which guarantees no conflict miss within a tile of data
- Optimal padding: a valid padding of minimal size for the padded array



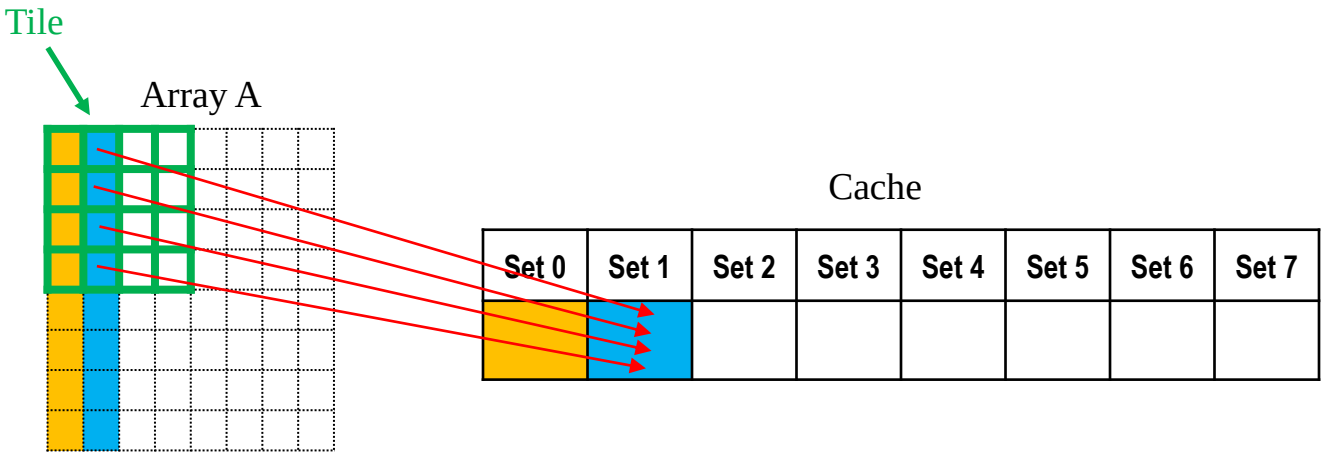
Data Reuse, Set-Associative Caches, and Padding

let us assume 1 cache line contains 1 word, so block size = 1 word

In the example below, Tile size = 4 * 4, associativity = 2

Case 1) without padding : A[8][8]

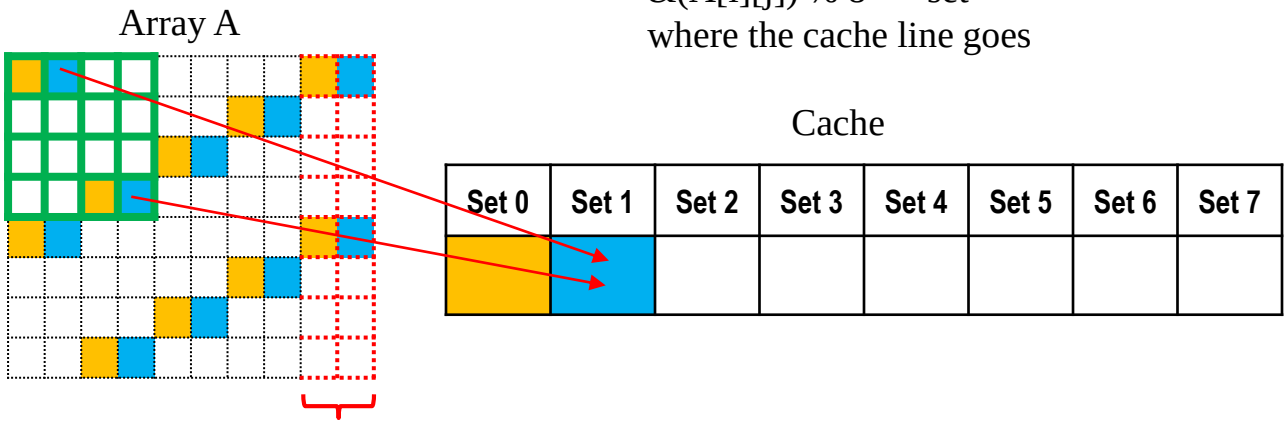
```
for (t=0; t<4; t++) {
  for (i=0; i<4; i++) {
    for (j=0; j<4; j++) {
      access(A[i][j]);
    }
  }
}
```



&(A[i][j]) % 8 => set where the cache line goes

Case 2) with padding : A[8][10]

```
for (t=0; t<4; t++) {
  for (i=0; i<4; i++) {
    for (j=0; j<4; j++) {
      access(A[i][j]);
    }
  }
}
```



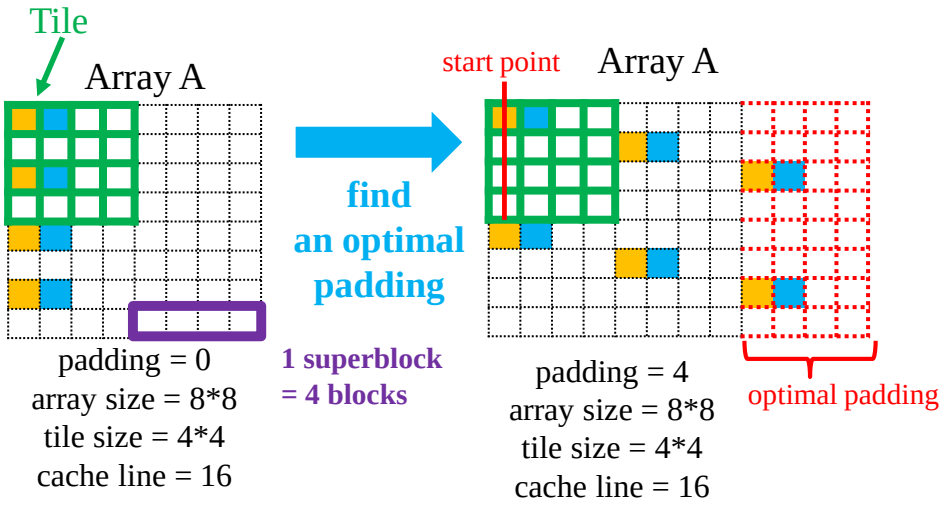
Prior Work and Contributions

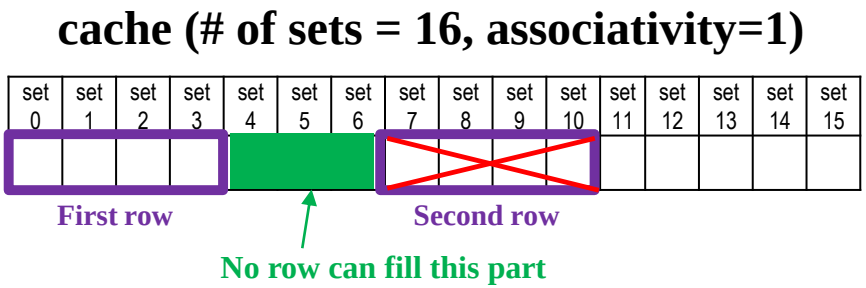
Approach	Feature				Padding computation time
	Set-Associative Cache?	Multi-level Padding?	Arbitrary Tile-size?	Imperfectly nested loops?	
Auto-tuning	Yes	Yes	Yes	Yes	Extremely High
CME (Ghosh) [Toplas 1999]	Yes	Yes	Yes	No	Extremely High
Li&Song [Toplas 2004]	No	No	No	Yes	Very Low
This work	**Yes**	**Yes**	**Yes**	**Yes**	**Very Low**

Analytical Solution: Divisibility Property

Divisibility: tile size divides padded array size → cache set should be uniformly filled

Case 1 (direct-mapped cache): # of cache sets = 16, array size = 8*8, tile size = 4*4





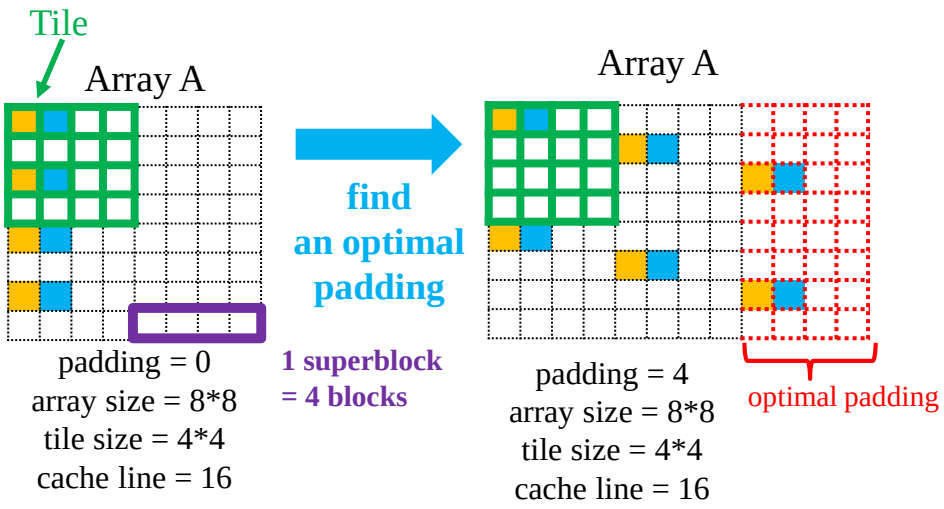
Theorem 1 (Direct-mapped cache).
$d = dimension, D_1 = tile\ width, D_2 = tile\ height, S = \#\ of\ sets\ in\ a\ cache, N_1 = array\ width + padding$

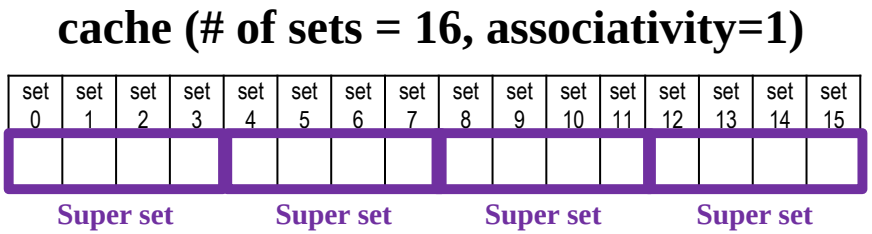
1. $\forall i, 1 \le i \le d, D_i\ divides\ N_i$
2. $\forall i, 1 \le i \le d-1, \gcd(S/\prod_{k=1}^{i} D_k, N_i/D_i) = 1$

Analytical Solution: Divisibility Property

Divisibility: tile size divides padded array size → cache set should be uniformly filled

Case 1 (direct-mapped cache): # of cache sets = 16, array size = 8*8, tile size = 4*4





gcd(4,P)=1

→ **P = 1 superset or 3 superset**

→ **P = 4 or 12**

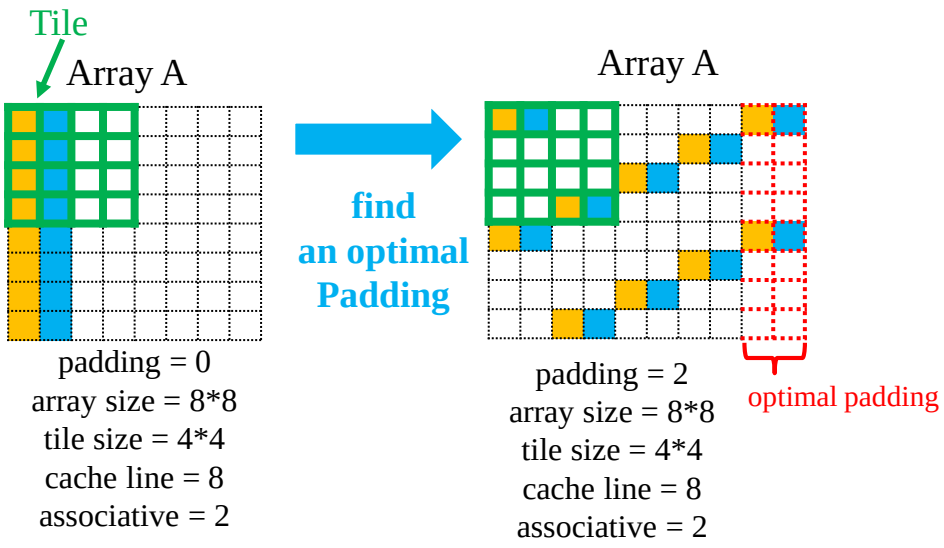
Theorem 1 (Direct-mapped cache).
$d = dimension, D_1 = tile\ width, D_2 = tile\ height, S = \#\ of\ sets\ in\ a\ cache, N_1 = array\ width + padding$

1. $\forall i, 1 \le i \le d, D_i\ divides\ N_i$
2. $\forall i, 1 \le i \le d-1, \gcd(S/\prod_{k=1}^{i} D_k, N_i/D_i) = 1$

Analytical Solution: Divisibility Property

Divisibility: tile size divides padded array size → cache set should be uniformly filled

Case 2 (2-way associative cache): # of cache sets = 8, array size = 8*8, tile size = 4*4,



oiriginal cache (# of sets = 8, 2-way associative)

set 0	set 1	set 2	set 3	set 4	set 5	set 6	set 7

Theorem 2 (Associative cache).

$d = dimension, D_1 = tile\ width, D_2 = tile\ height, S = \#\ of\ sets\ in\ a\ cache, N_1 = array\ width + padding$

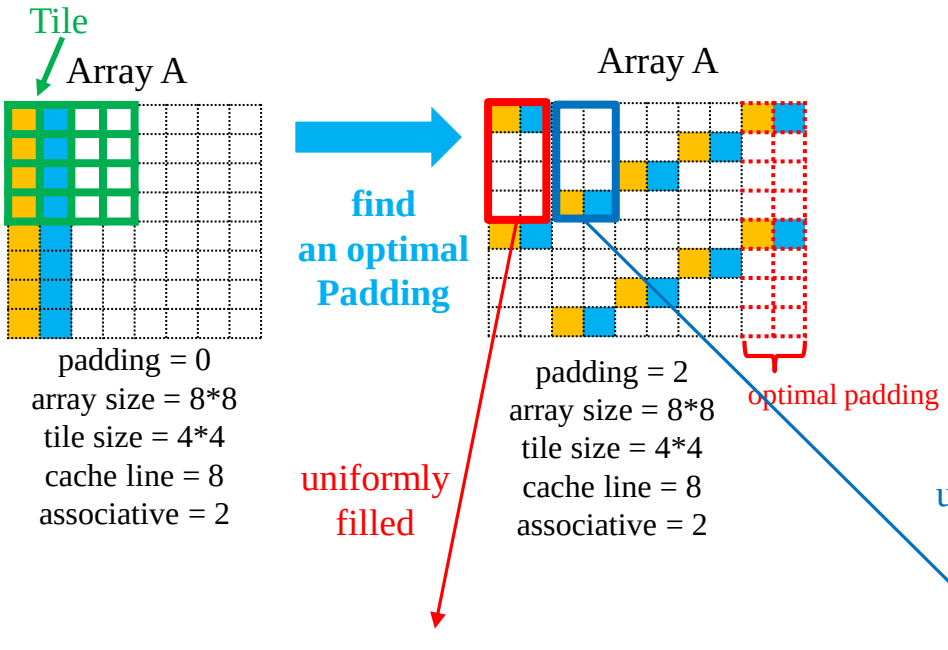
$Let\ g_i = \gcd(S/\prod_{k=1}^{i} g_k, N_i)$

1. $\forall i, 1 \le i \le d-1, \exists j, 1 \le j \le i, \prod_{k=1}^{i} g_k\ divides\ D_j \prod_{i=1}^{j-1} g_i$
2. $\exists i, 1 \le i \le d, S\ divides\ D_i \prod_{k=1}^{i-1} g_i$

Analytical Solution: Divisibility Property

Divisibility: tile size divides padded array size → cache set should be uniformly filled

Case 2 (2-way associative cache): # of cache sets = 8, array size = 8*8, tile size = 4*4,



original cache (# of sets = 8, 2-way associative)

set 0	set 1	set 2	set 3	set 4	set 5	set 6	set 7

divide it into two virtual "direct-mapped" cache

virtual cache 1 (# of sets = 8, direct-mapped)

set 0	set 1	set 2	set 3	set 4	set 5	set 6	set 7

virtual cache 2 (# of sets = 8, direct-mapped)

set 0	set 1	set 2	set 3	set 4	set 5	set 6	set 7

Analytical Solution: Divisibility Property

Divisibility: tile size divides padded array size → cache set should be uniformly filled

Case 1 (direct-mapped cache): # of cache sets = 16, array size = 8*8, tile size = 4*4,

→ Valid padding set = {4,12}

Case 2 (2-way associative cache) : # of cache sets = 8, array size = 8*8,tile size = 4*4,

→ Valid padding set = {4,12} U {2,6,10,14}
= {2,4,6,10,12,14}

High Associativity ➡ More flexibility ➡ Find smaller padding

Analytical Solution: Divisibility Property

Without Divisibility? (tile size < cache size)

In some cases, it is impossible to satisfy the divisibility property.

- **Three arrays with same size are used**
- **Tile size is a perfect square or cube.**

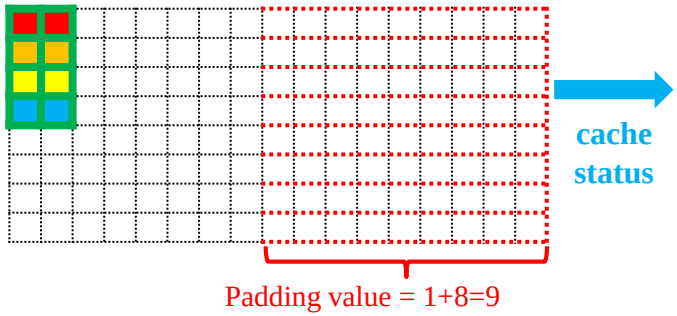
➡ **A general padding scheme is needed**

Computed Solution: Direct-Mapped Cache

Observation 1

Effect of padding value of "X" = Effect of padding value of "X+S" (S = # of sets in a cache)

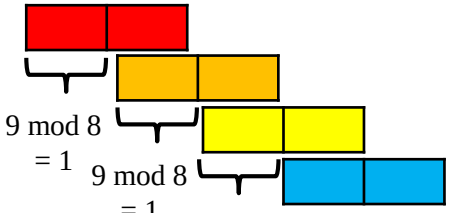
associativity = 2, S(# of sets in a cache) = 8, array size = 8*8, tile size = 4*2

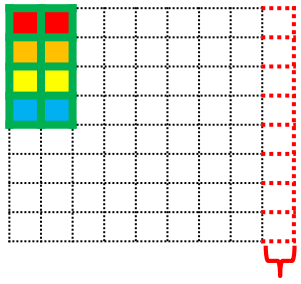


cache status

cache

	set 0	set 1	set 2	set 3	set 4	set 5	set 6	set 7
# of occurrences	1	2	2	2	1	0	0	0

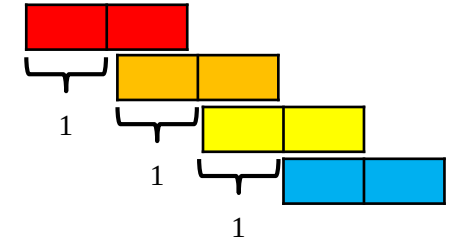




cache status

cache

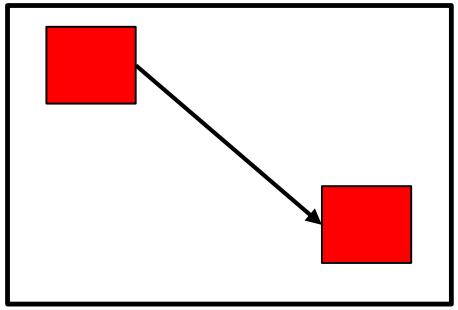
	set 0	set 1	set 2	set 3	set 4	set 5	set 6	set 7
# of occurrences	1	2	2	2	1	0	0	0



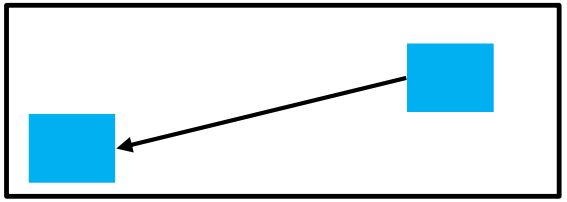
Computed Solution: Direct-Mapped Cache

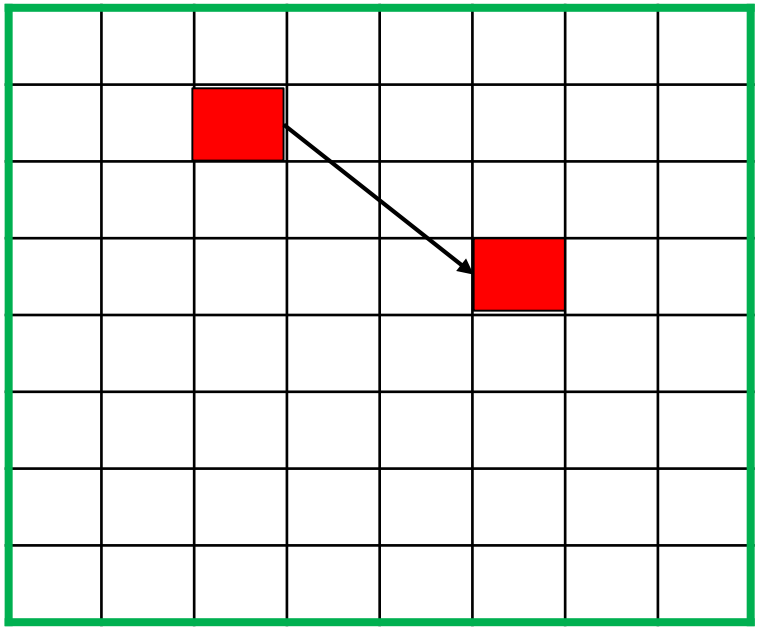
Observation 2
We can quickly determine whether a padding value is valid or not

Forward



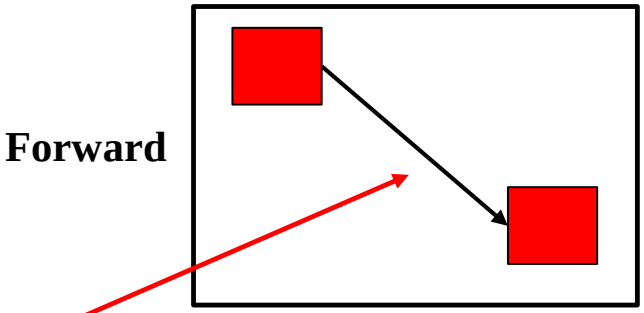
Backward

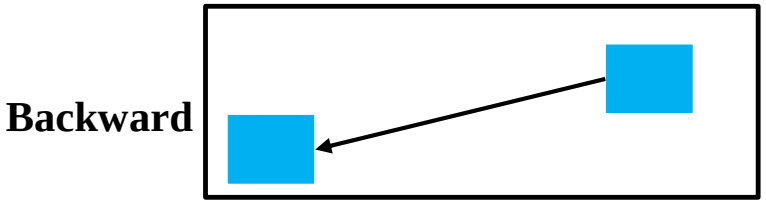


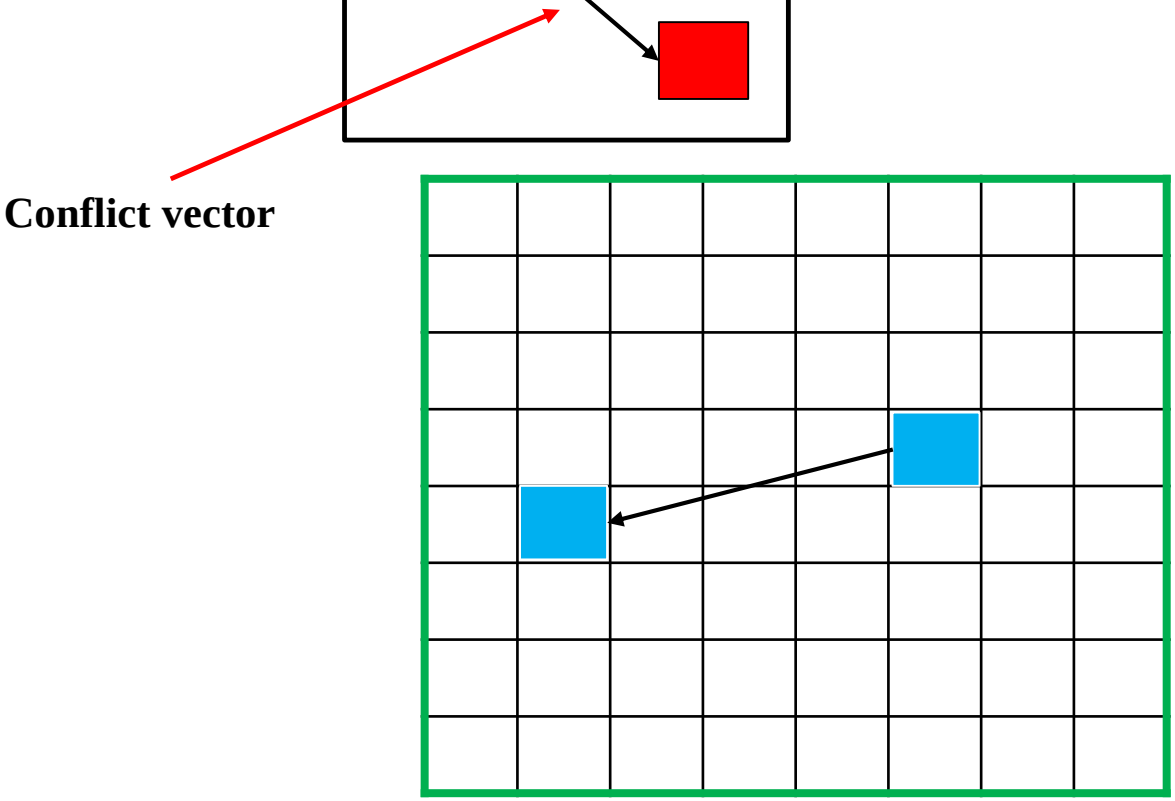


Computed Solution: Direct-Mapped Cache

Observation 2
We can quickly determine whether a padding value is valid or not



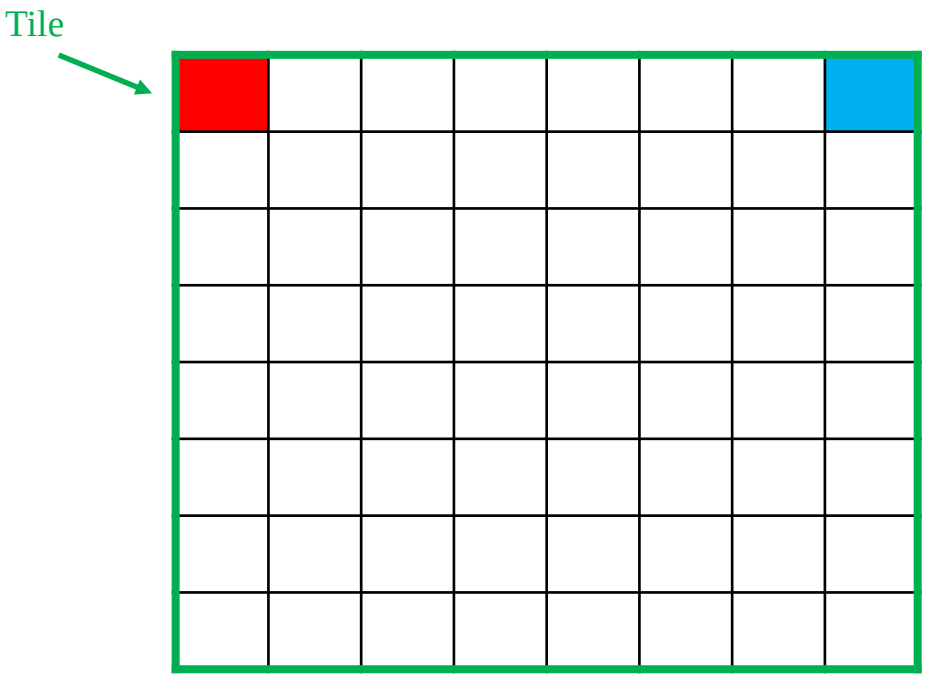




Computed Solution: Direct-Mapped Cache

Observation 2
We can quickly determine whether a padding value is valid or not

Suppose same color blocks means they map to the same cache set.
If a padding make red color and blue color be unique in a tile,
then this padding is valid padding.



Computed Solution: Direct-Mapped Cache

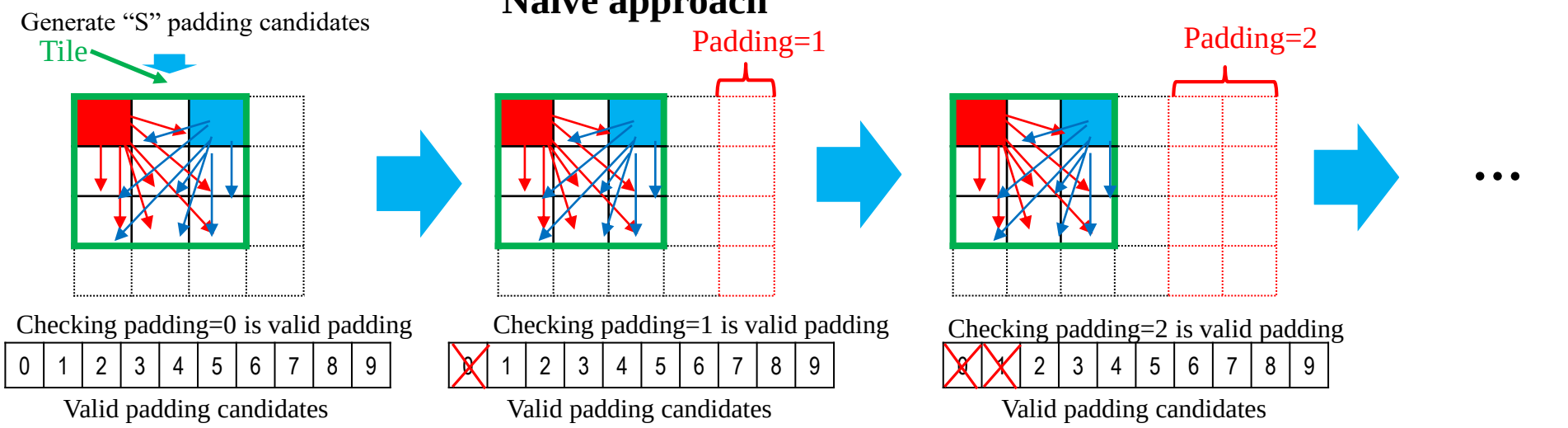
Observation 1
Effect of padding value of "X" = Effect of padding value of "X+S" (S = # of sets in a cache)

Observation 2
We can quickly determine whether a padding value is valid or not

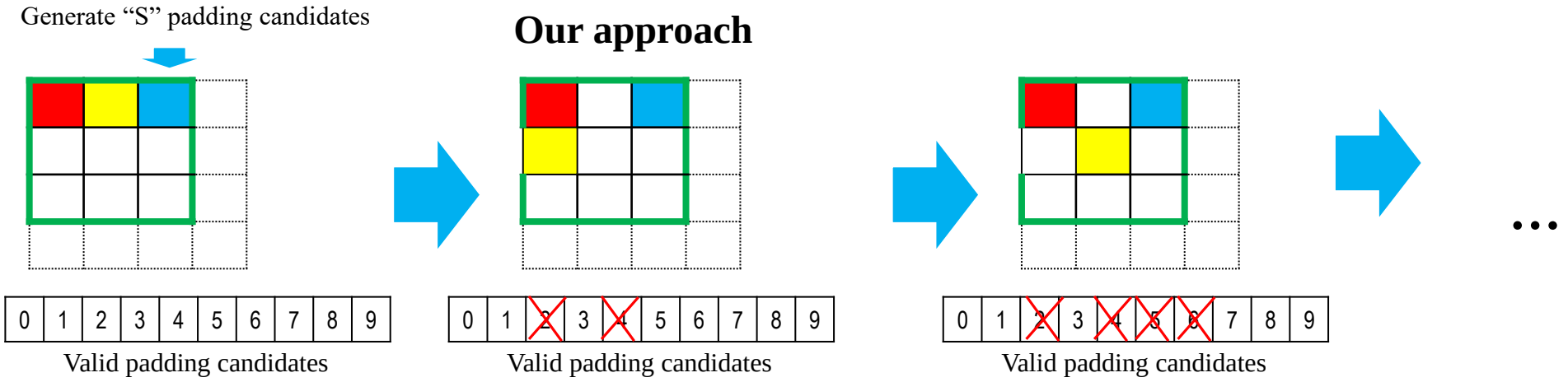
- **For each of "S" padding candidate, we only check which block has same color as top-left corner and top-right corner.**
- **Time complexity = O(S*T) (S = # of sets in a cache, T = # of blocks in a tile)**
- **How can we accelerate it?**
- **Time complexity of O(S log S) exists.**

Computed Solution: Direct-Mapped Cache

Naïve approach

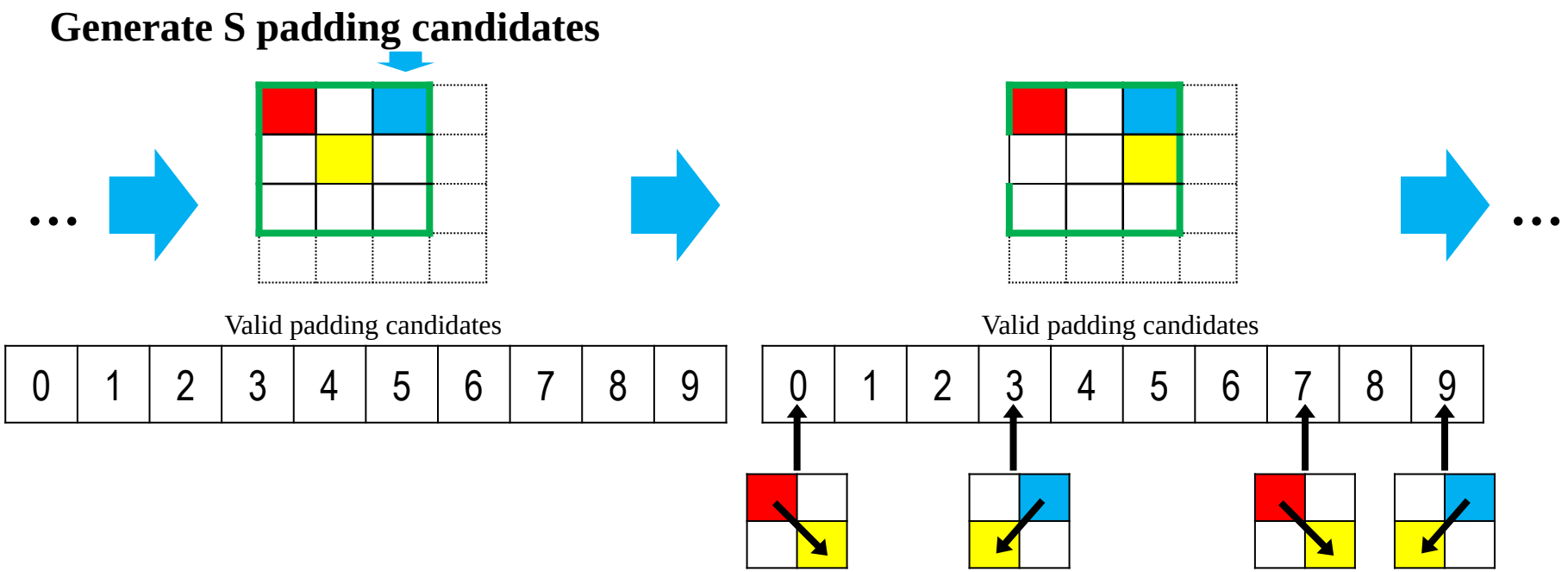


Our approach



Which padding value make “red block and yellow block” or “blue block and yellow block” map to the same set, and eliminate them from candidates

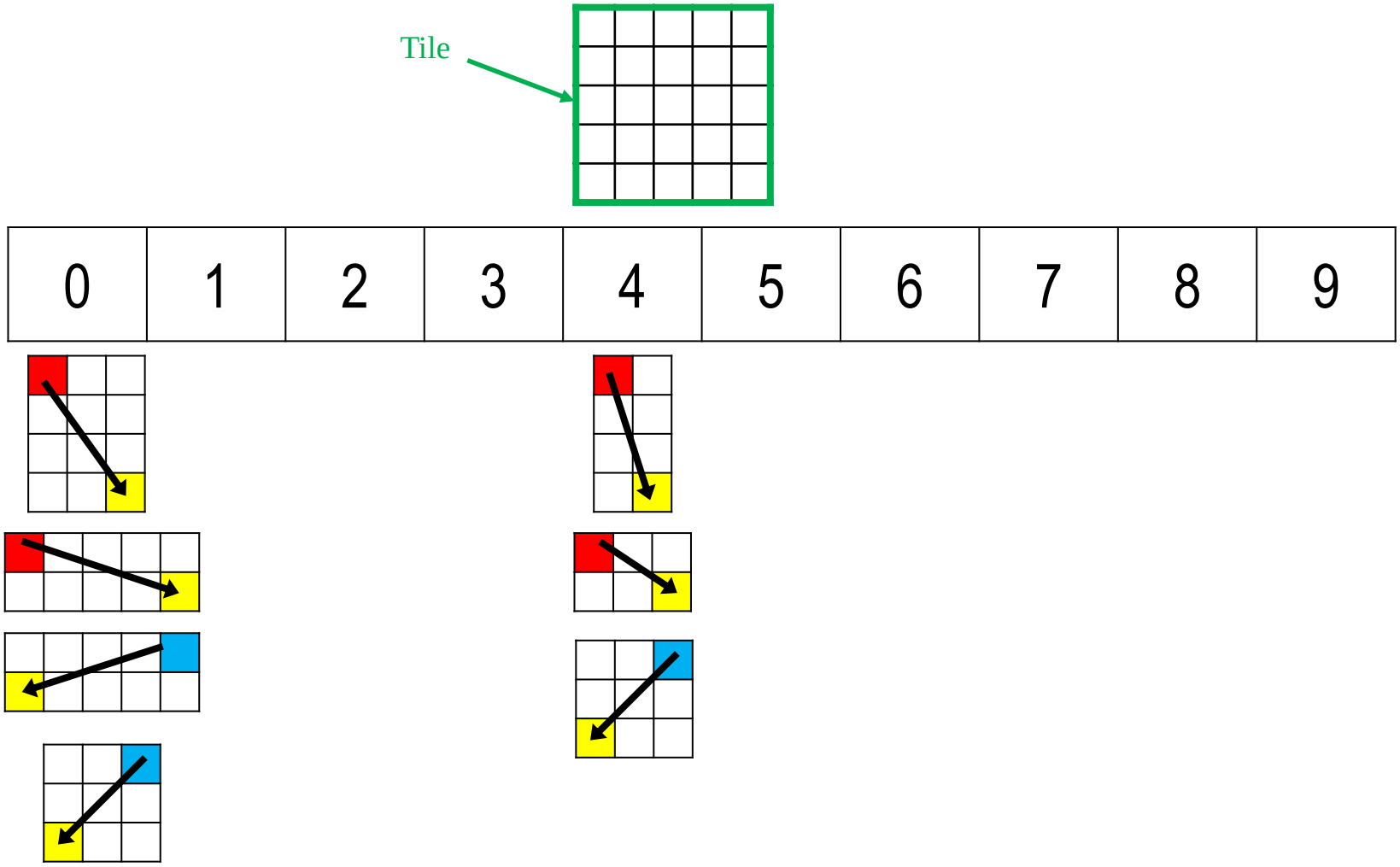
Computed Solution: Set-Associative Cache



Instead of eliminating padding candidates, we add a conflict vector to candidates.

Computed Solution: Set-Associative Cache

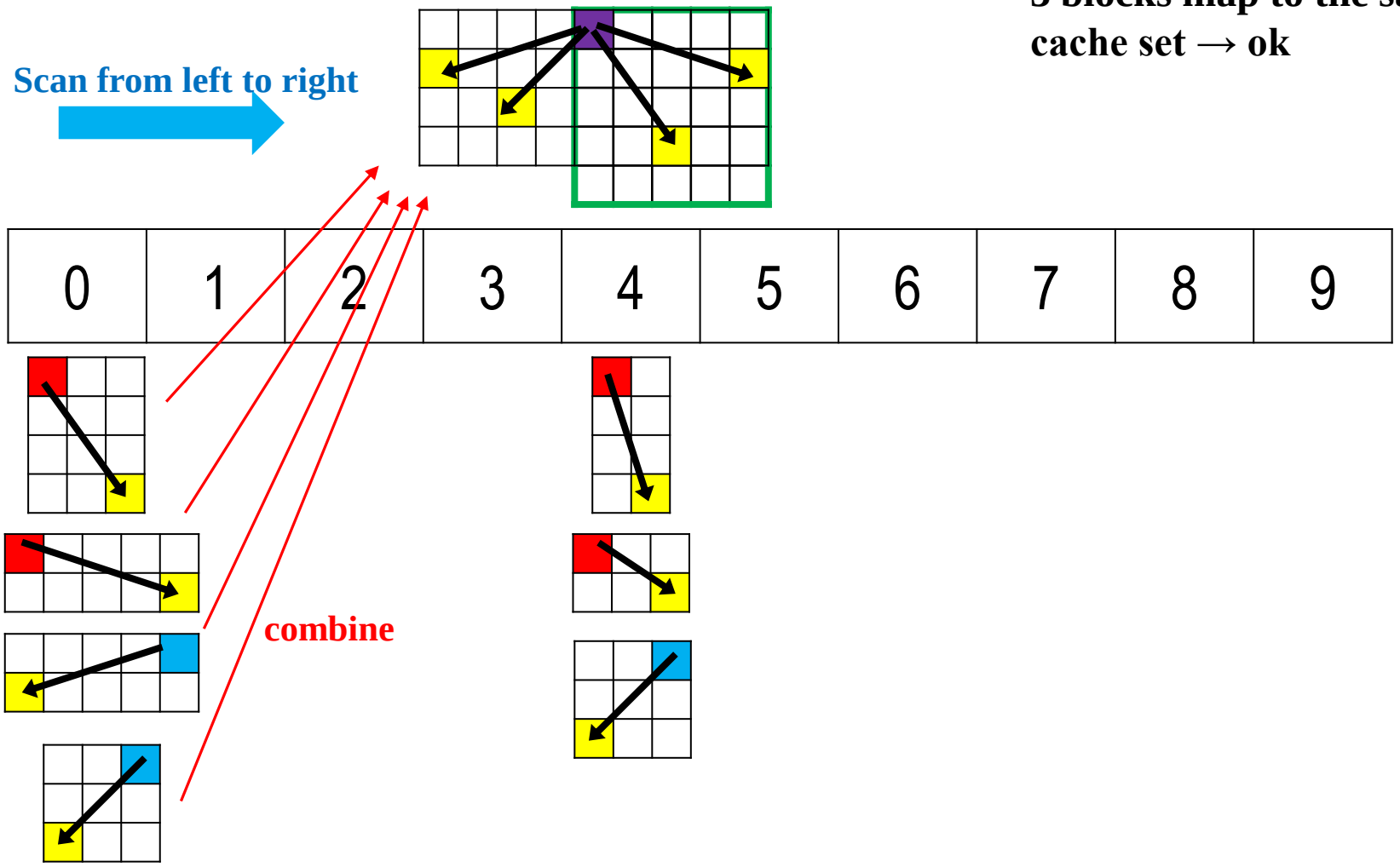
Tile size = 5*5, associativity = 3, # of cache sets = 10



Computed Solution: Set-Associative Cache

Tile size = 5*5, associativity = 3, # of cache sets = 10

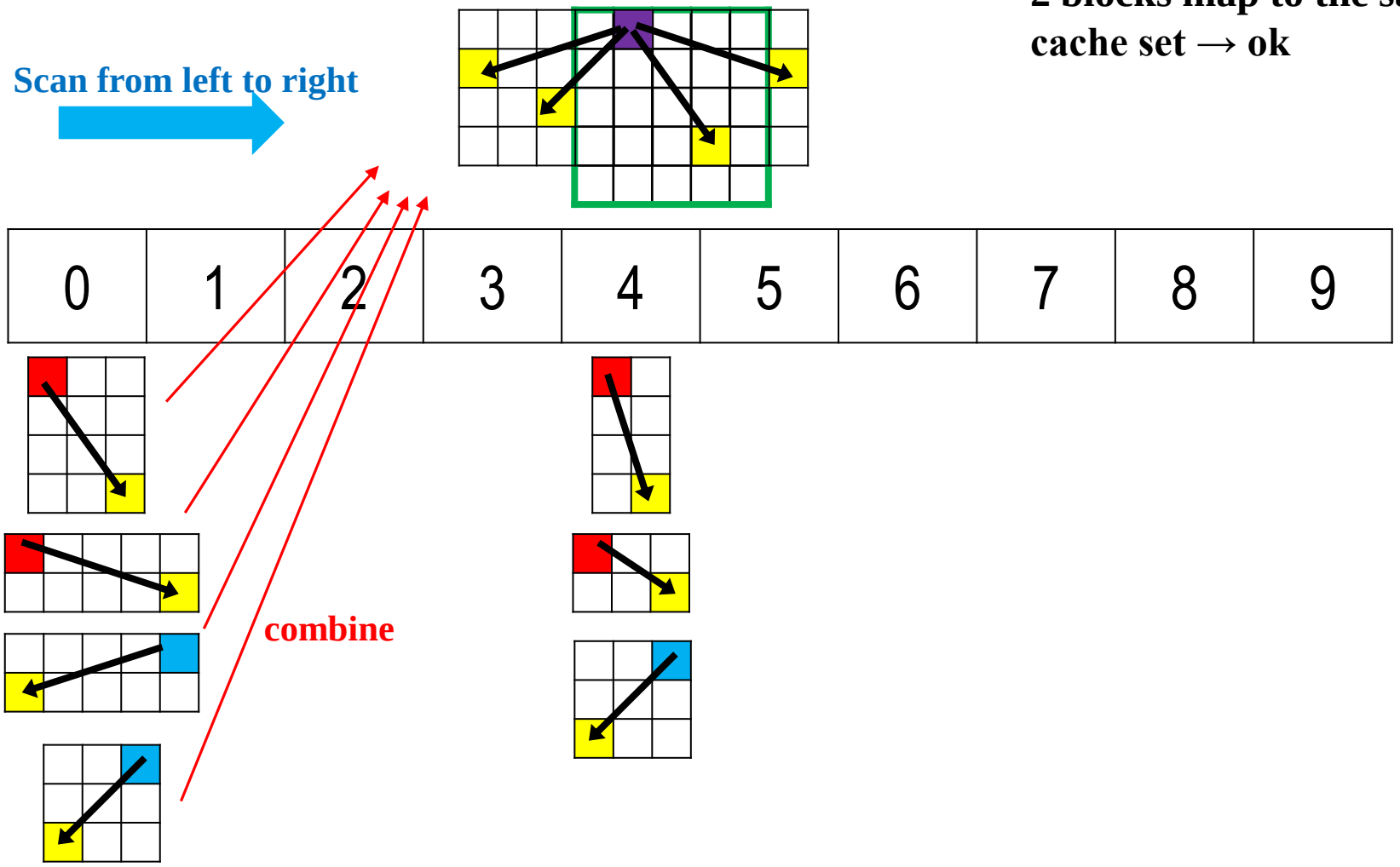
3 blocks map to the same cache set → ok



Computed Solution: Set-Associative Cache

Tile size = 5*5, associativity = 3, # of cache sets = 10

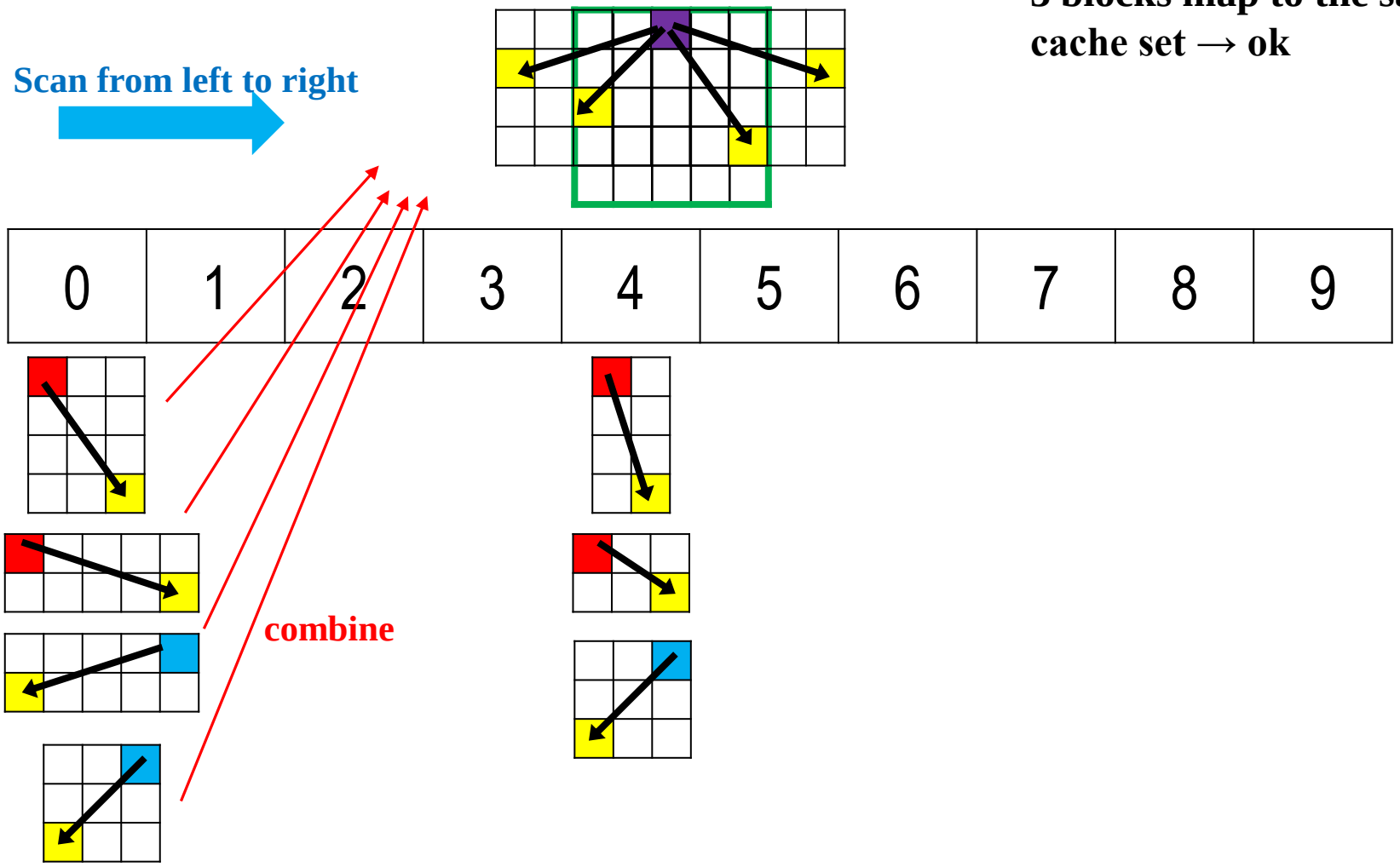
2 blocks map to the same cache set → ok



Computed Solution: Set-Associative Cache

Tile size = 5*5, associativity = 3, # of cache sets = 10

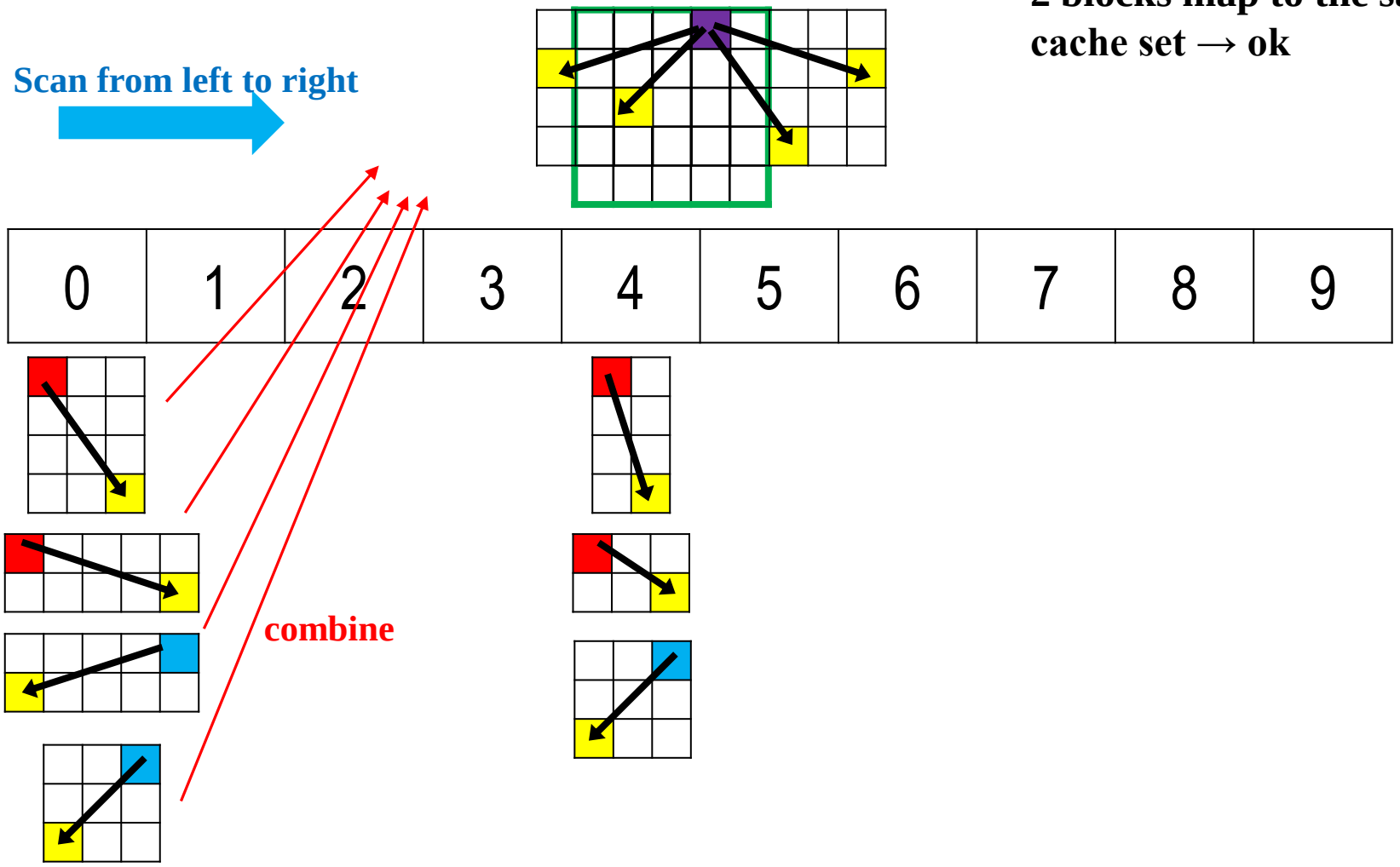
3 blocks map to the same cache set → ok



Computed Solution: Set-Associative Cache

Tile size = 5*5, associativity = 3, # of cache sets = 10

2 blocks map to the same cache set → ok



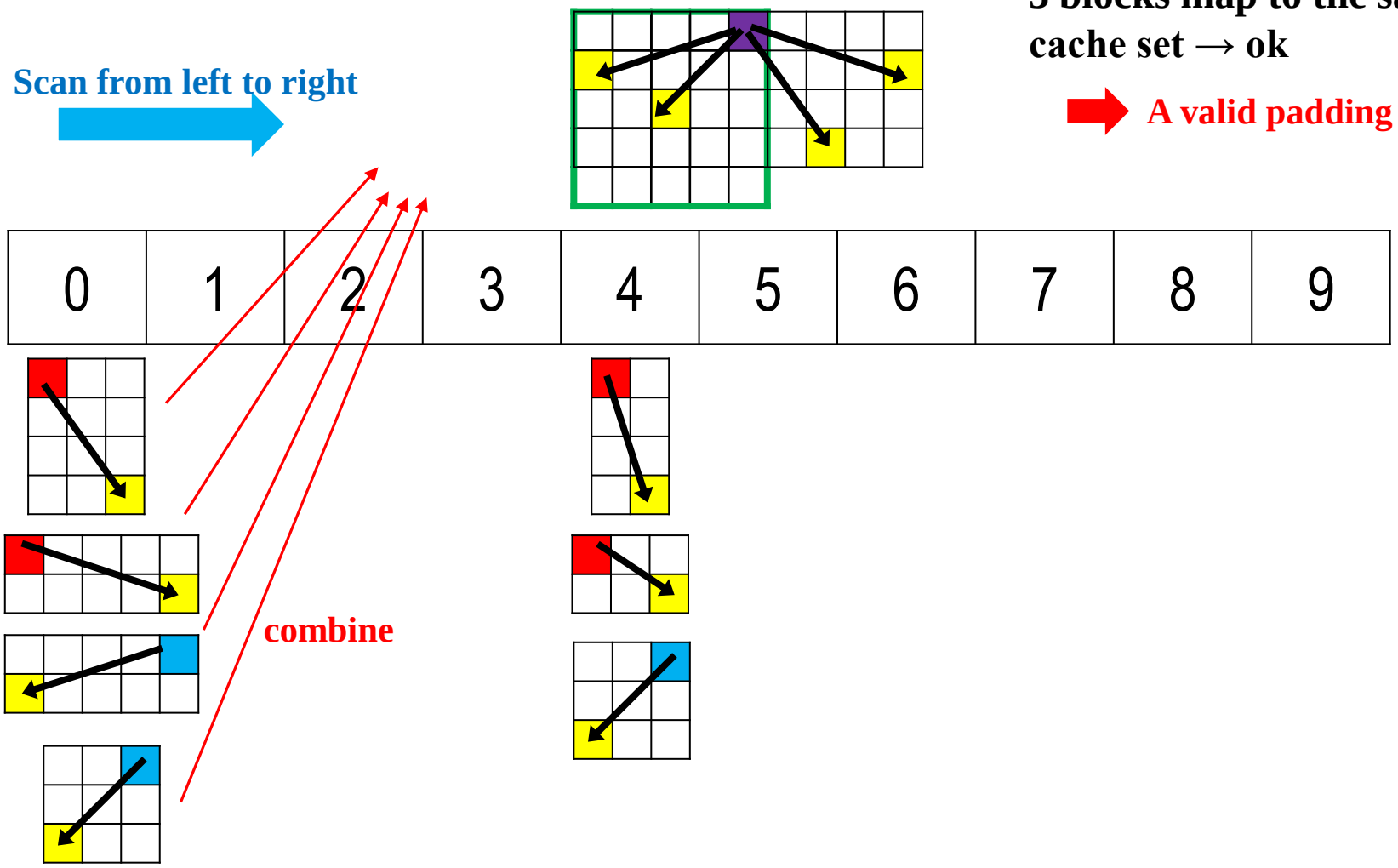
Computed Solution: Set-Associative Cache

Tile size = 5*5, associativity = 3, # of cache sets = 10

3 blocks map to the same cache set → ok

A valid padding

Scan from left to right

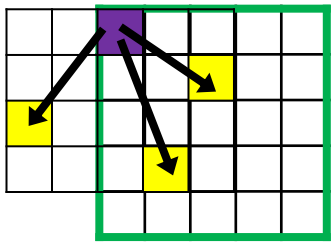


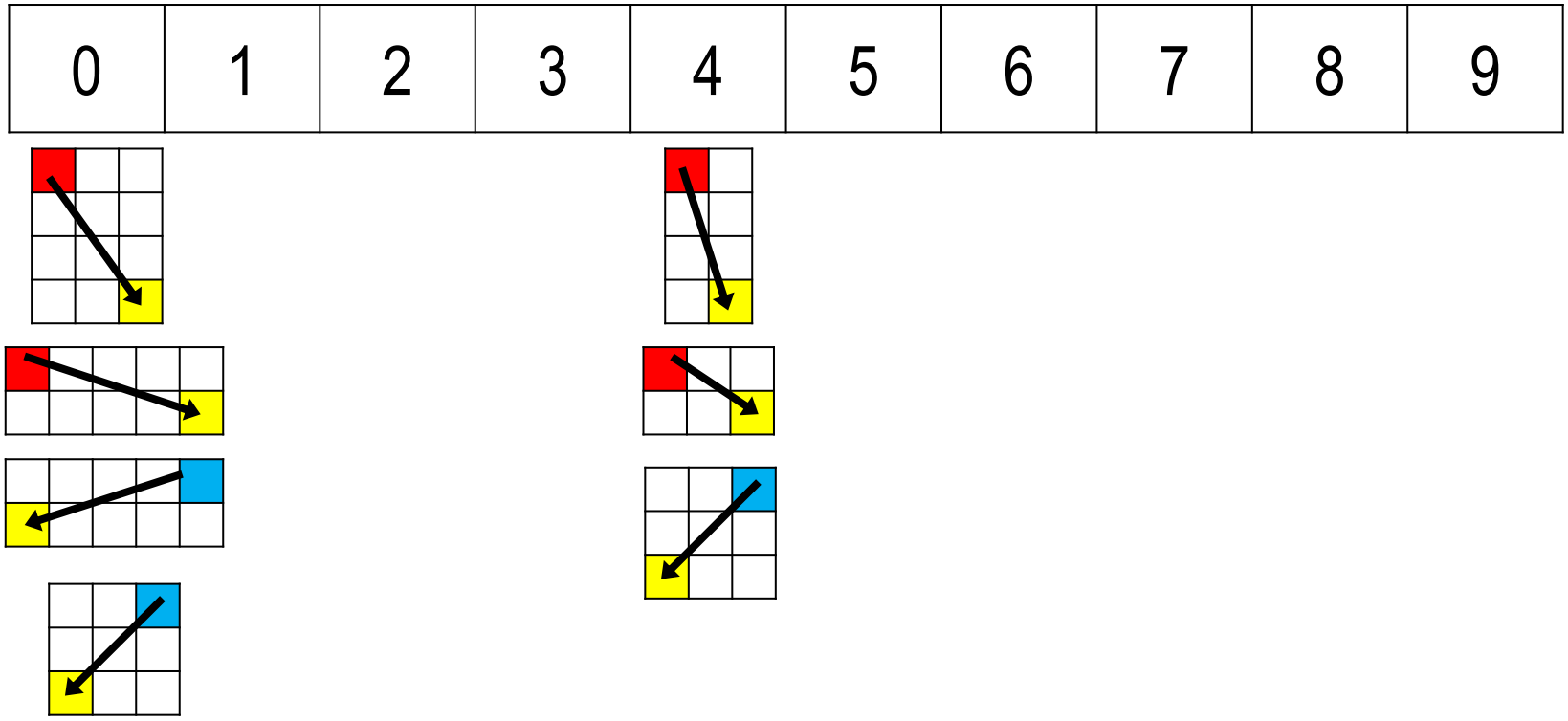
Computed Solution: Set-Associative Cache

Tile size = 5*5, associativity = 3, # of cache sets = 10

3 blocks map to the same cache set → ok

Scan from left to right



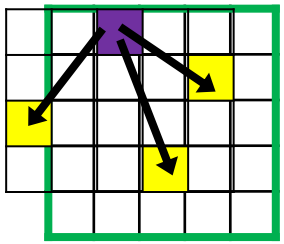


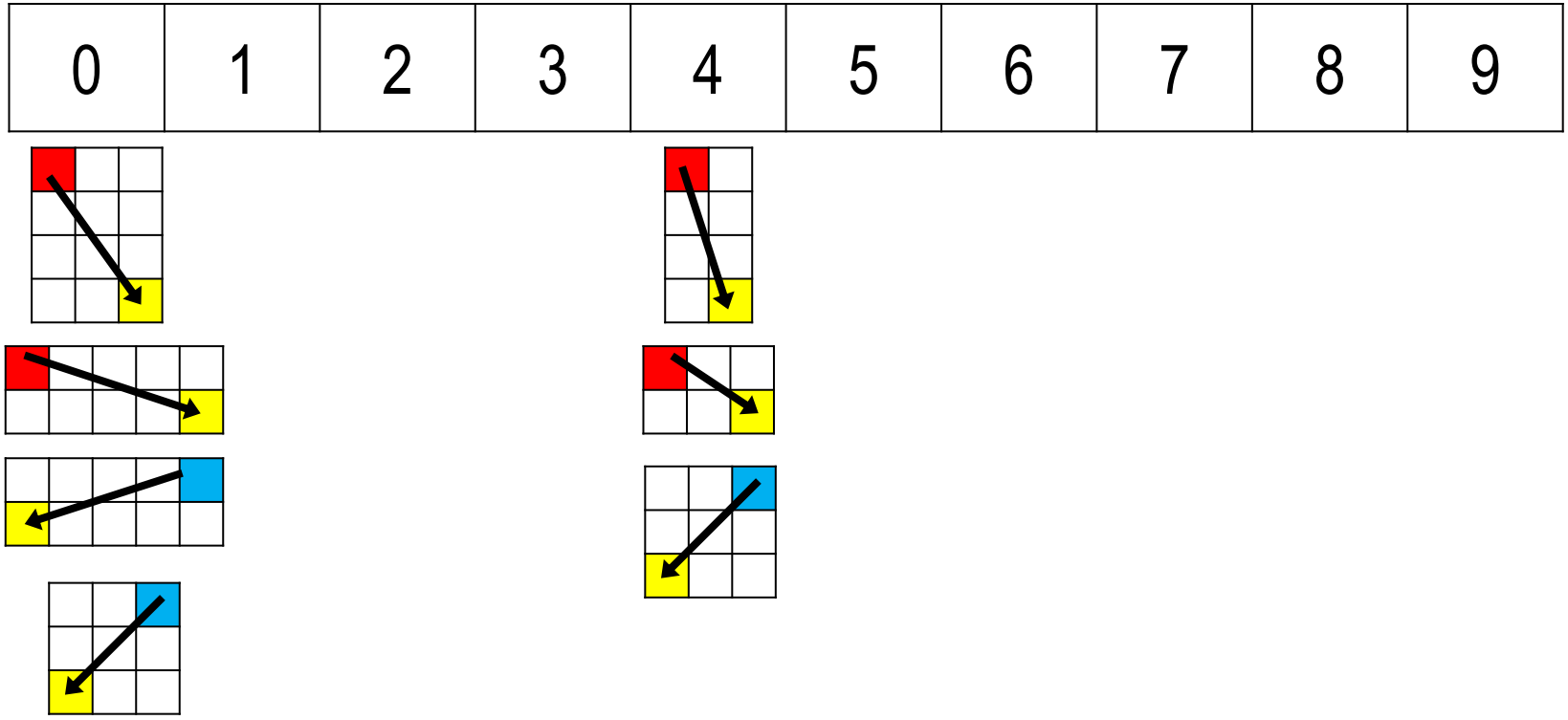
Computed Solution: Set-Associative Cache

Tile size = 5*5, associativity = 3, # of cache sets = 10

3 blocks map to the same cache set → ok

Scan from left to right





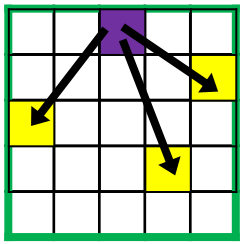
Computed Solution: Set-Associative Cache

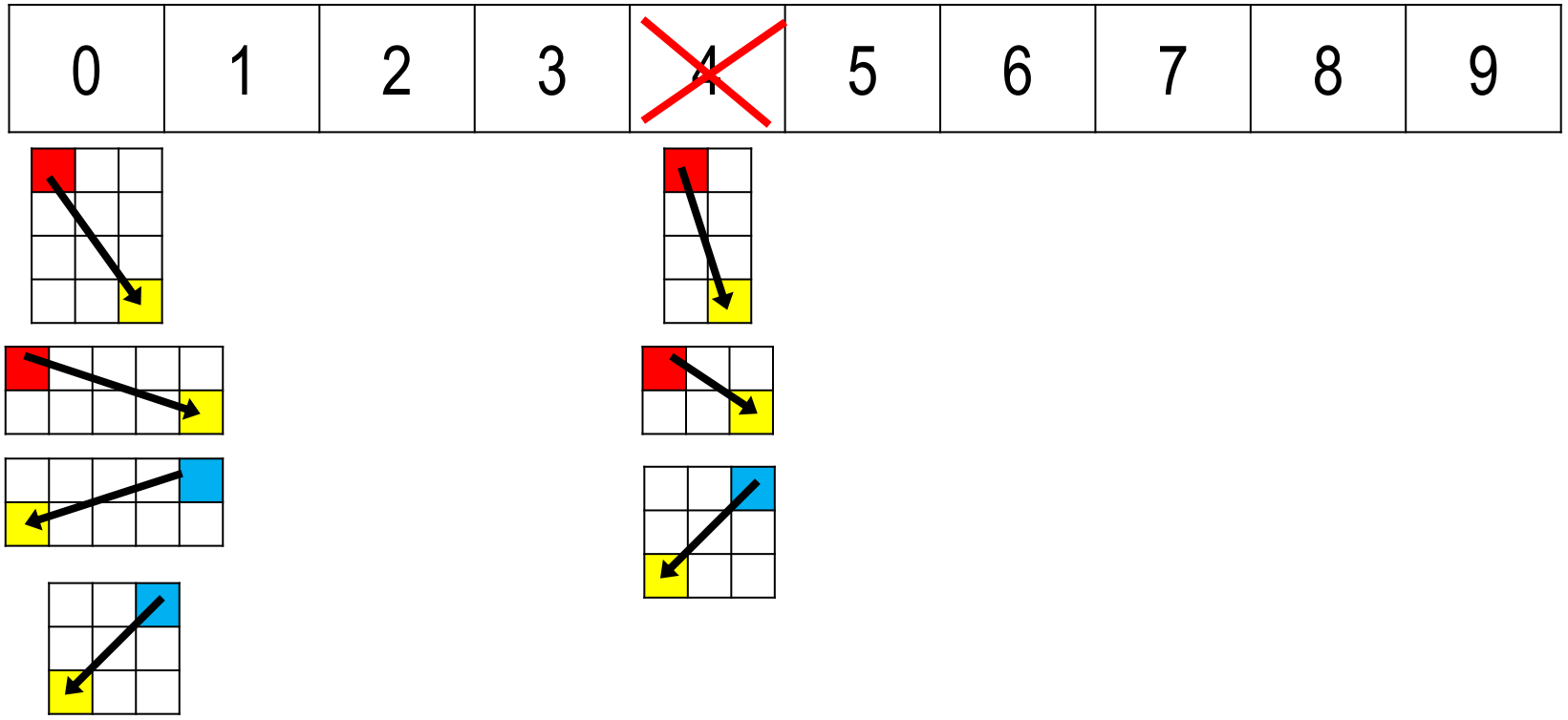
Tile size = 5*5, associativity = 3, # of cache sets = 10

4 blocks map to the same cache set → no

Not a valid padding

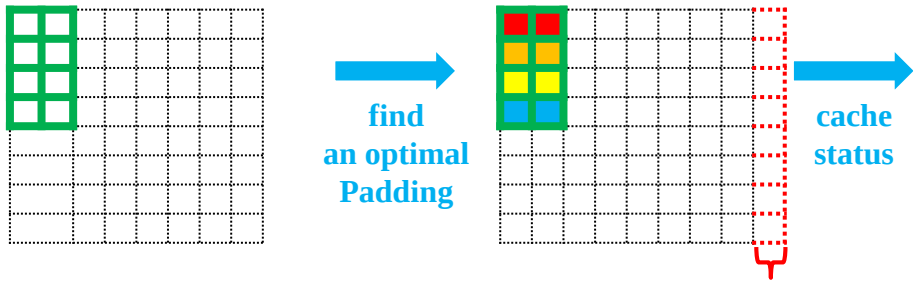
Scan from left to right





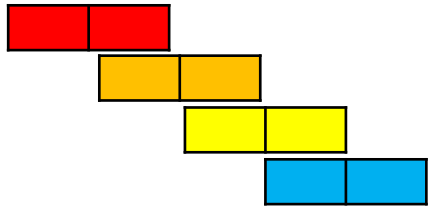
Computed Solution: Inter-Array Padding

associativity = 2,# of cache sets = 8, array size = 8*8,tile size = 4*2, **two identical tiles**

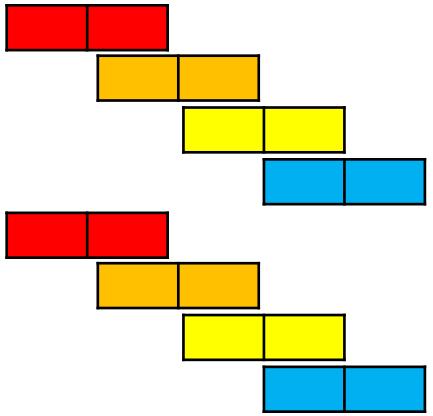


cache

	set 0	set 1	set 2	set 3	set 4	set 5	set 6	set 7
# of occurrences	1	2	2	2	1	0	0	0



Set 0	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7
1	2	2	2	1	0	0	0
1	2	2	2	1	0	0	0
2	4	4	4	2	0	0	0

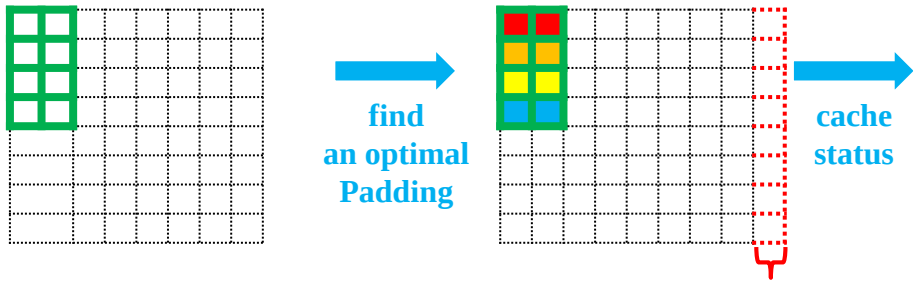


Combining occurrences with inter-padding value = 0

Inter-array padding value = 0 → not valid padding!

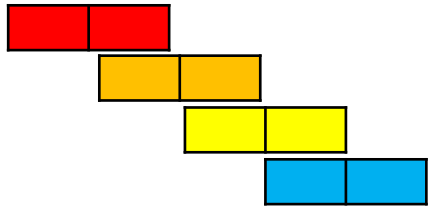
Computed Solution: Inter-Array Padding

associativity = 2,# of cache sets = 8, array size = 8*8,tile size = 4*2, **two identical tiles**

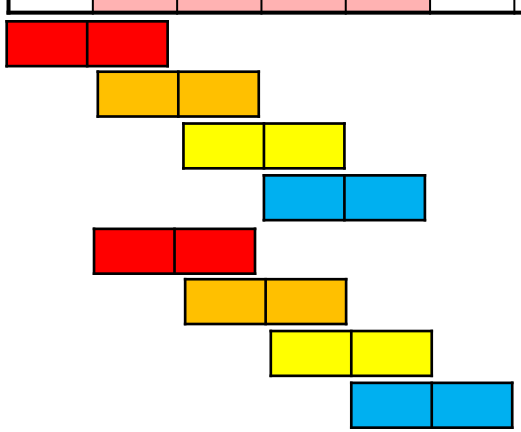


cache

	set 0	set 1	set 2	set 3	set 4	set 5	set 6	set 7
# of occurrences	1	2	2	2	1	0	0	0



Set 0	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 0
1	2	2	2	1	0	0	0	
0	1	2	2	2	1	0	0	0
1	3	4	4	3	1	0	0	

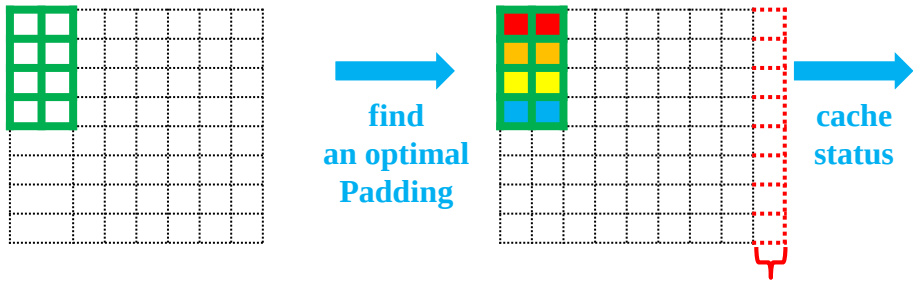


Combining occurrences with inter-padding value = 0

Inter-array padding value = 1 → not valid padding!

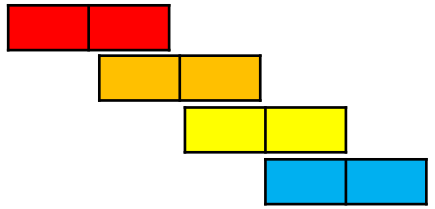
Computed Solution: Inter-Array Padding

associativity = 2,# of cache sets = 8, array size = 8*8,tile size = 4*2, **two identical tiles**

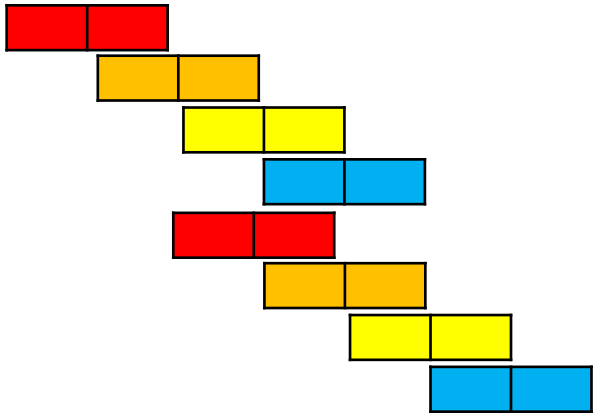


cache

	set 0	set 1	set 2	set 3	set 4	set 5	set 6	set 7
# of occurrences	1	2	2	2	1	0	0	0



Set 0	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 0	Set 1
1	2	2	2	1	0	0	0		
0	0	1	2	2	2	1	0	0	0
1	2	3	4	3	2	1	0		

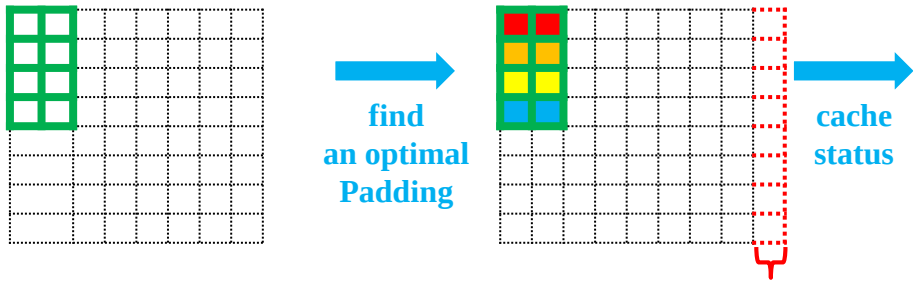


Combining occurrences with inter-padding value = 0

Inter-array padding value = 2 → not valid padding!

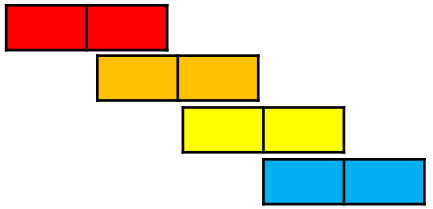
Computed Solution: Inter-Array Padding

associativity = 2,# of cache sets = 8, array size = 8*8,tile size = 4*2, **two identical tiles**



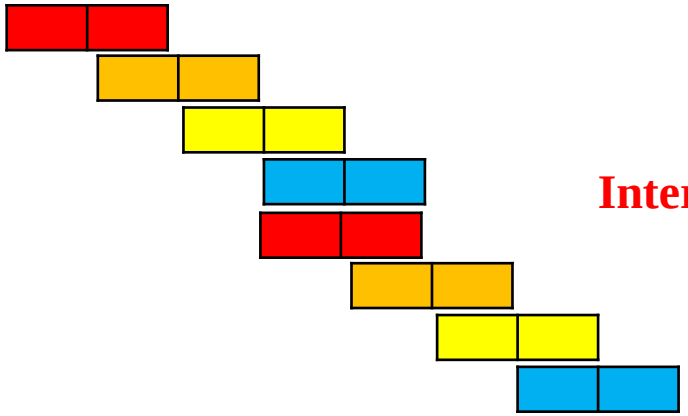
cache

	set 0	set 1	set 2	set 3	set 4	set 5	set 6	set 7
# of occurrences	1	2	2	2	1	0	0	0



Set 0	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 0	Set 1	Set 2
1	2	2	2	1	0	0	0			
+ 0	0	0	1	2	2	2	1	0	0	0
1	2	2	3	3	2	2	1			

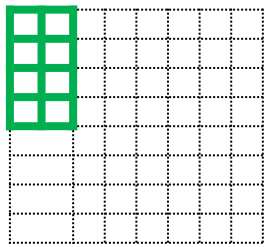
Combining occurrences with inter-padding value = 0



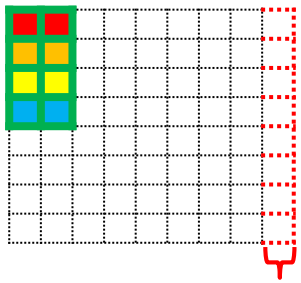
Inter-array padding value = 3 → not valid padding!

Computed Solution: Inter-Array Padding

associativity = 2,# of cache sets = 8, array size = 8*8,tile size = 4*2, **two identical tiles**



find an optimal Padding

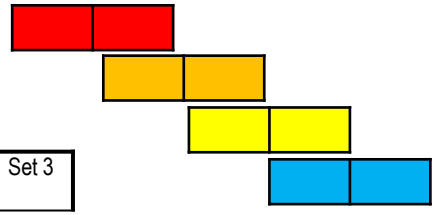


optimal padding

cache status

cache

	set 0	set 1	set 2	set 3	set 4	set 5	set 6	set 7
# of occurrences	1	2	2	2	1	0	0	0



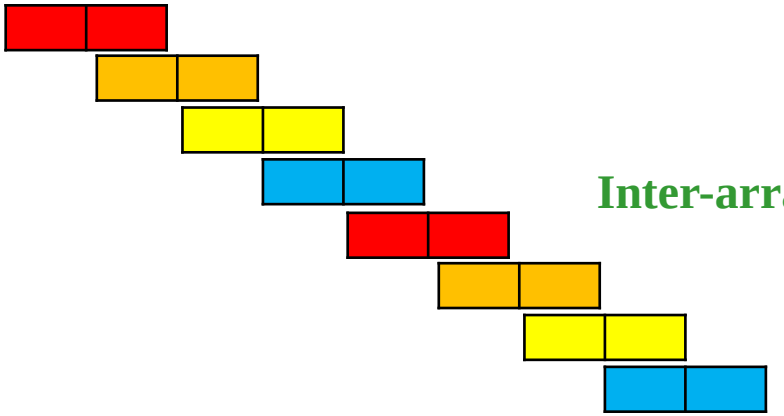
Set 0	Set 1	Set 2	Set 3	Set 4	Set 5	Set 6	Set 7	Set 0	Set 1	Set 2	Set 3
1	2	2	2	1	0	0	0				

\+

1	0	0	0	1	2	2	2	1	0	0	0

Combining occurrences with inter-padding value = 0

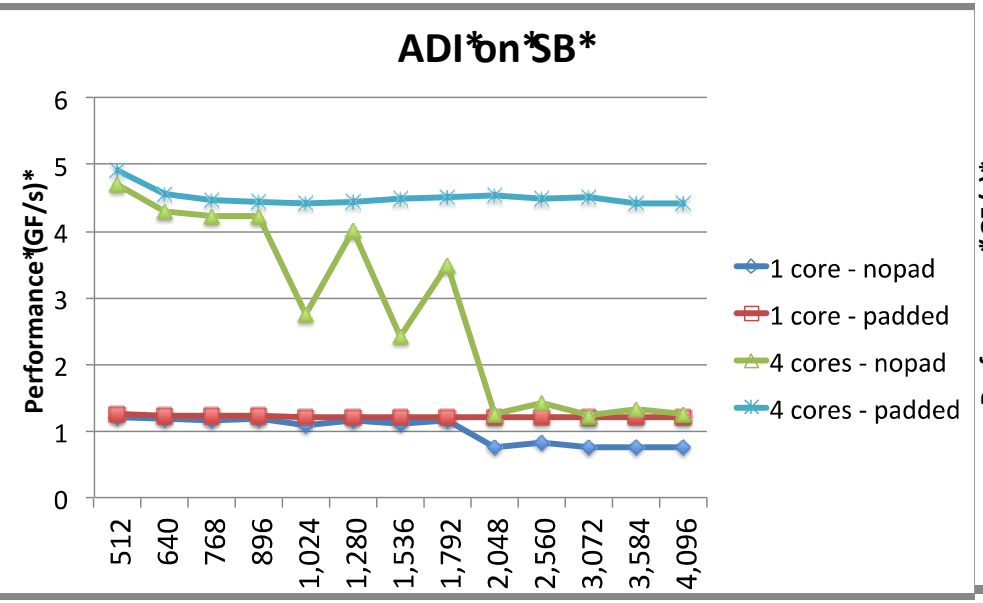
2	2	2	2	2	2	2	2

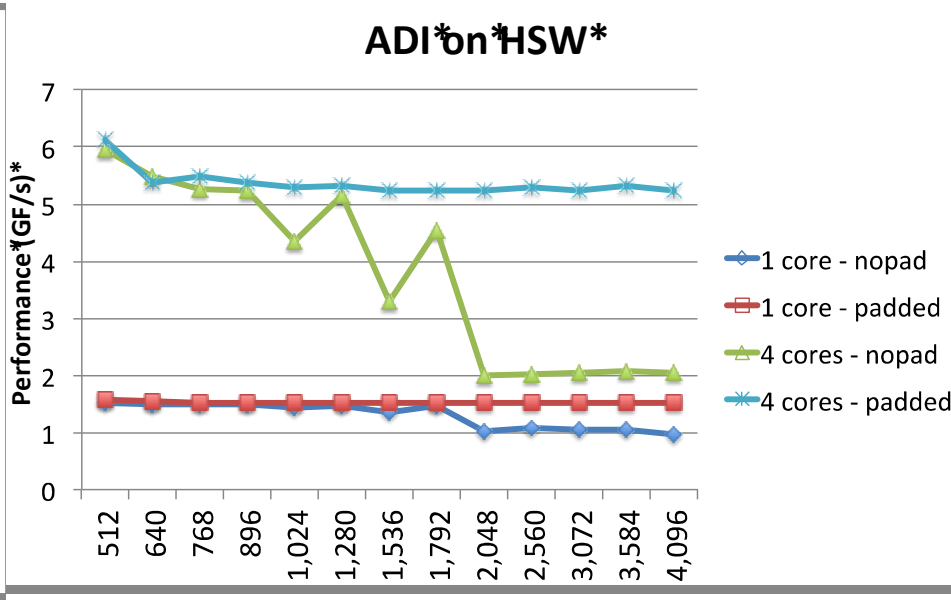


Inter-array padding value = 4 → valid padding!

Experimental Results: SandyBridge vs. Haswell

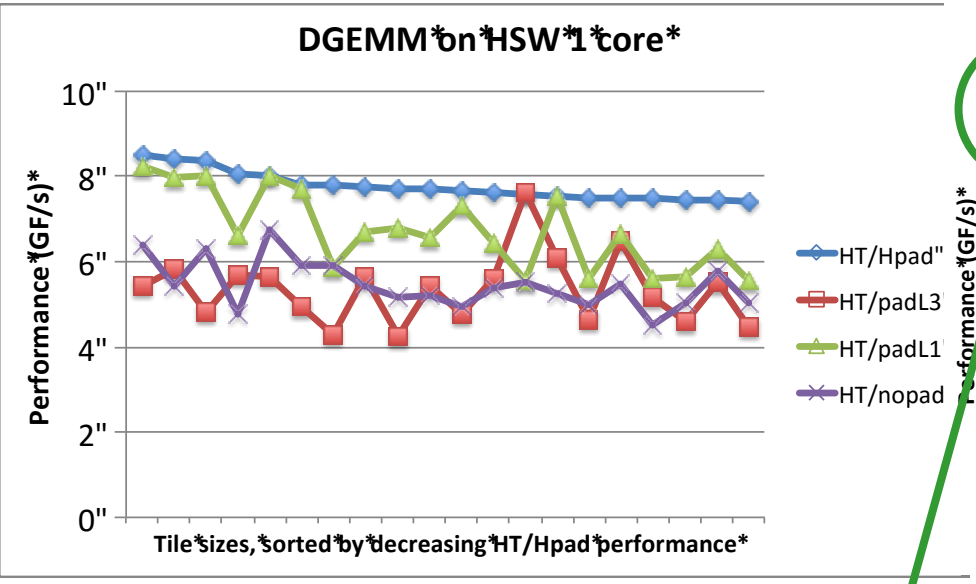
- Different padding values needed for different problem sizes
- Performance (GF/s) stays flat with padding
- Multi-core performance degrades massively without padding

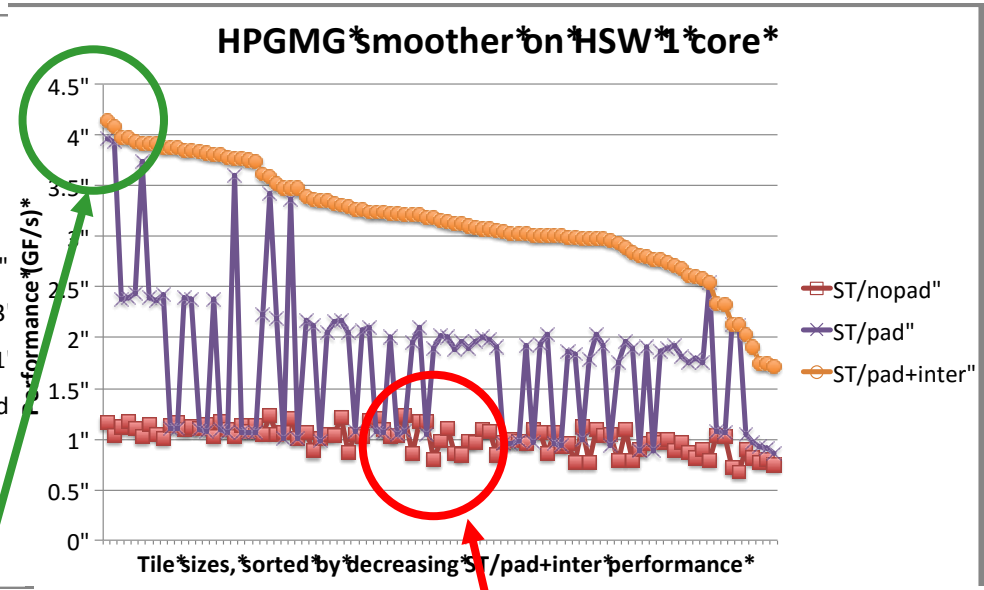




Experimental Results: Hierarchical Padding

- Hierarchical Padding provides additional performance improvement
- The tile size performance ordering is different w/ and w/o padding
- Inter-Array padding provides additional substantial performance



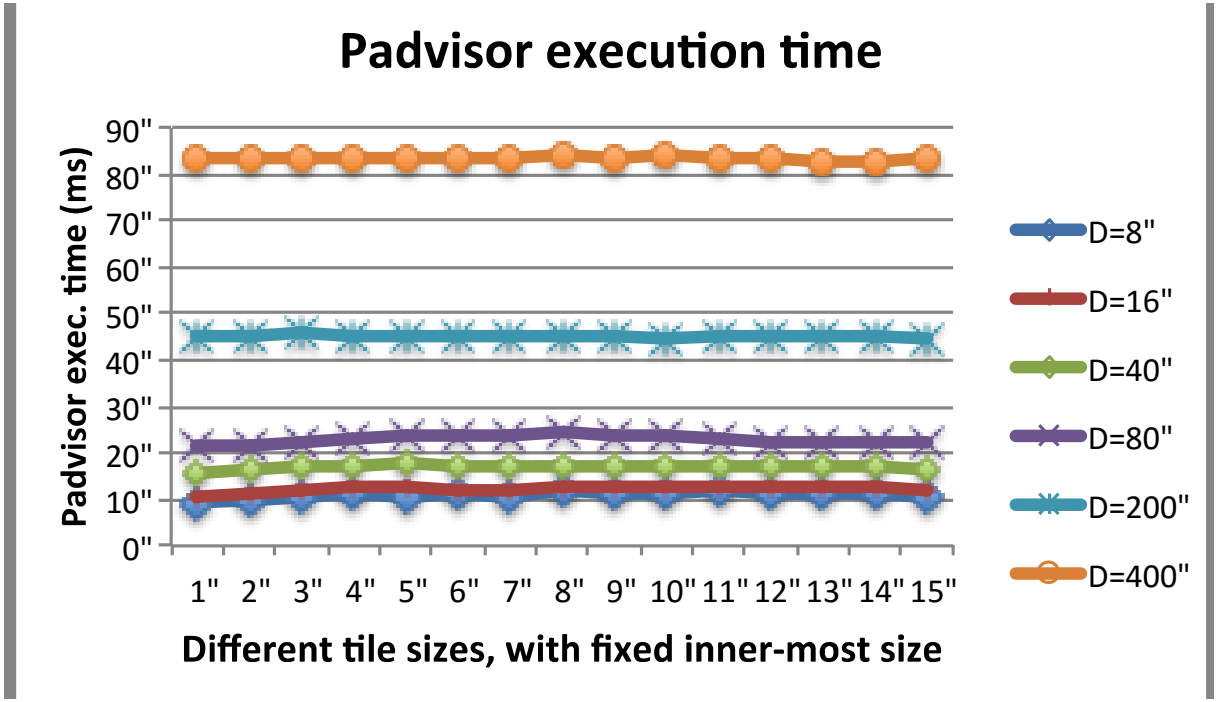


Best tile with padding

Best tile without padding

Experimental Results: Padvisor execution time

- Execution time depends on the tile size
- Complexity of analytical solution is very small
- Naïve implementation is already fast enough for production use



Take-Home Message

- **Conflict misses in set-associative cache may prevent full cache utilization**
  - Smaller utilization => more data traffic, and lower performance
  - **Conflict miss + low utilization = low performance**
- **Avoiding conflict miss is possible using array padding**
  - Potential massive impact on performance, e.g., 1.6x improvement for certain FFT sizes using Intel MKL/FFT, 4x improvement possible for ADI
  - But searching for a good padding value cannot be decoupled from tile size exploration!
- **Prior state of practice: auto-tuning, or limited analytical solutions**
  - Auto-tuning requires to run the code for each candidate padding value
  - Prior analytical work: only for direct-mapped caches or prohibitive time-to-solution
- **Our work: complete treatment of padding using compile-time analysis**
  - **Handles associative caches**
  - Solutions for inter-array padding, intra-array, and **hierarchical padding schemes**
  - Very good computational complexity