Emulating Optimal Replacement with a Shepherd Cache

Kaushik Rajan[1] & R. Govindarajan

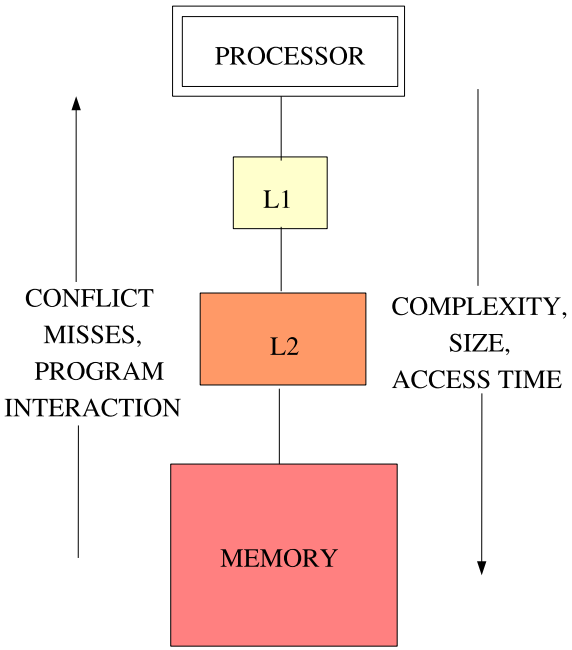
Supercomputer Education and Research Centre
Indian Institute of Science
Bangalore

05-Dec-2007

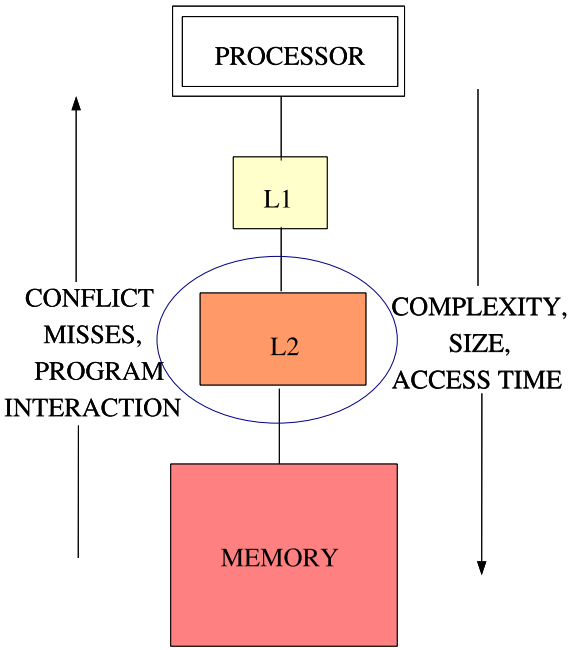
[1]Author acknowledges financial support provided through the MSR PhD Fellowship

Outline

Memory Hierarchy



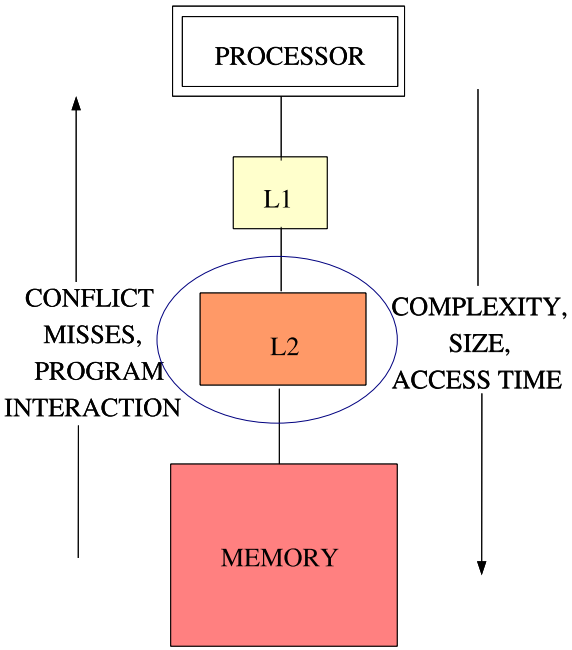
Memory Hierarchy



L2 and lower caches

- Objective : Need to reduce expensive memory accesses
- Design : Large size, Higher associativity, Complex design
- Problem : Do not interact with program directly and observe filtered temporal locality

Memory Hierarchy

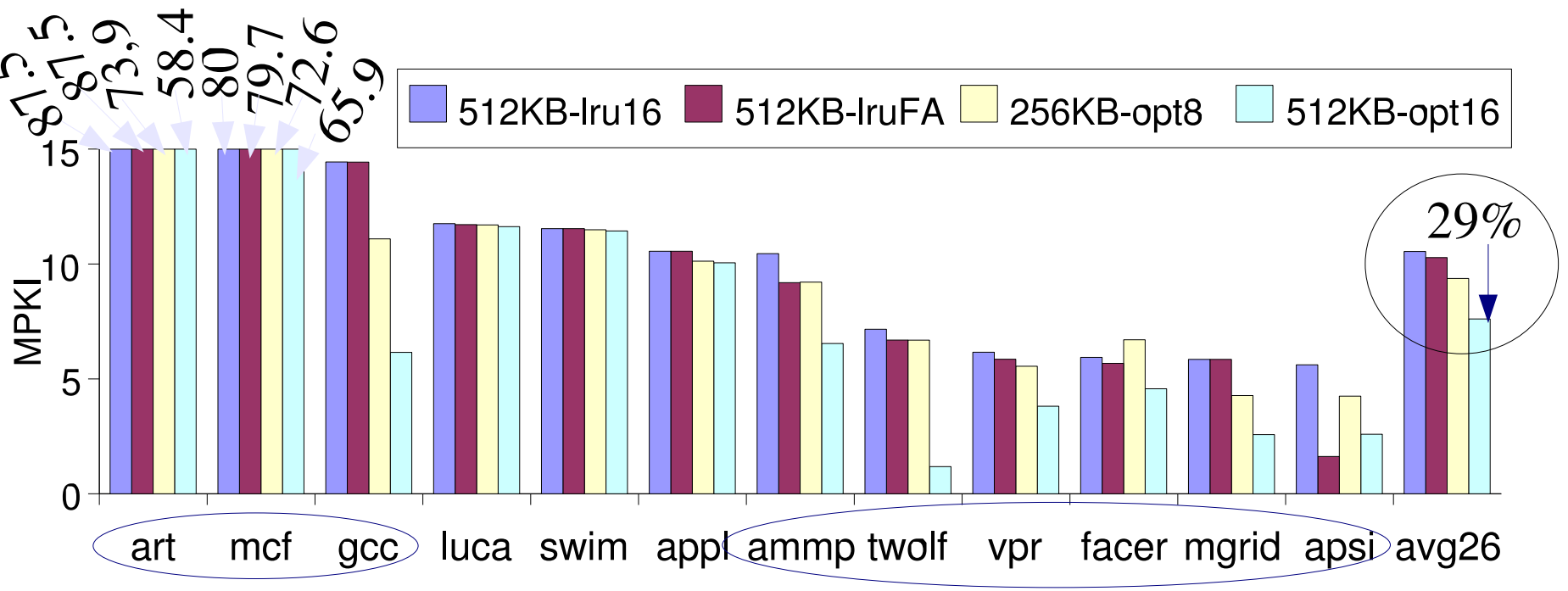


L2 and lower caches

- Objective : Need to reduce expensive memory accesses
- Design : Large size, Higher associativity, Complex design
- Problem : Do not interact with program directly and observe filtered temporal locality

- High Associativity $\Longrightarrow$ replacement policy crucial to performance
- L1 cache services temporal accesses $\Longrightarrow$ Lack of temporal accesses at L2 $\Longrightarrow$ LRU replacement inefficient
- Replacement decisions are taken off the processor critical path

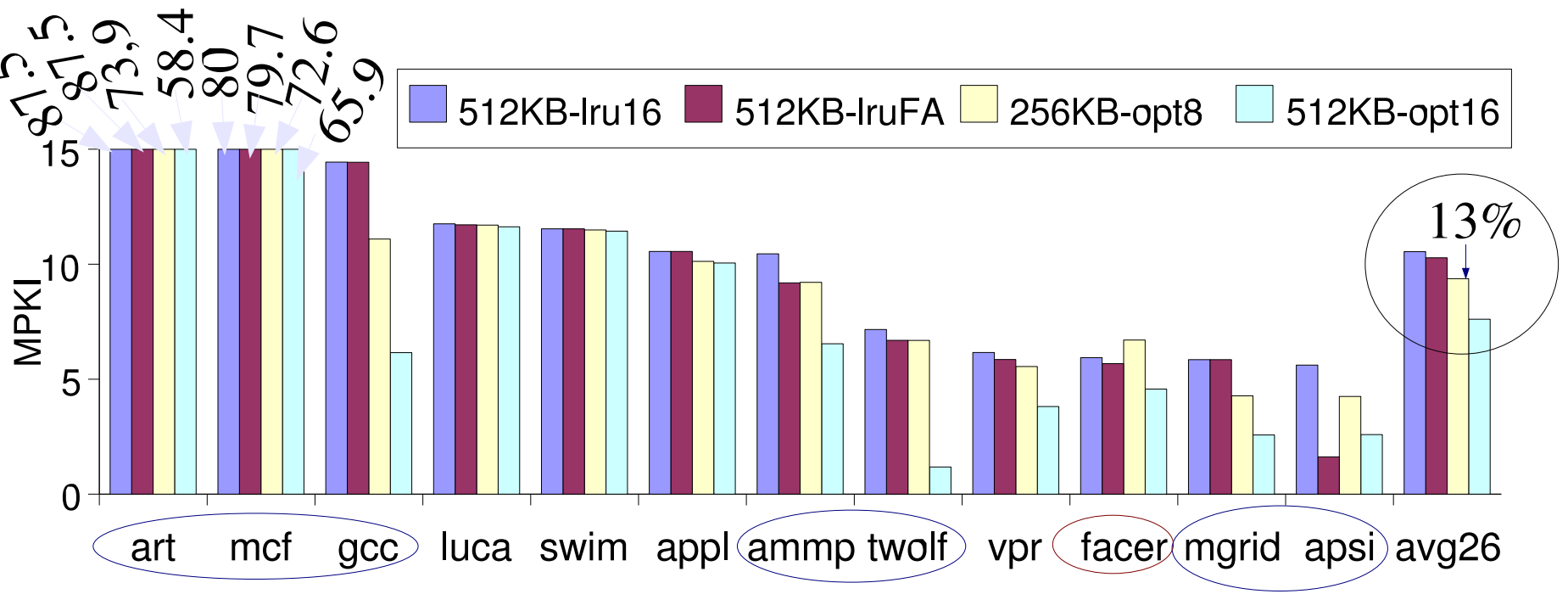
LRU vs OPT



MPKI for SPEC2000 suite, Benchmarks with MPKI < 5 not plotted but count towards average

Huge performance gap between LRU and OPT

LRU vs OPT



MPKI for SPEC2000 suite, Benchmarks with MPKI < 5 not plotted but count towards average

Huge performance gap between LRU and OPT
OPT at half the size preferable to LRU at double the size

Outline

The Optimal Replacement Policy

1. Replacement Candidates : On a miss any replacement policy could either choose to replace any of the lines in the cache or choose not to place the miss causing line in the cache at all.
2. Self Replacement : The latter choice is referred to as a self-replacement or a cache bypass

The Optimal Replacement Policy

1. Replacement Candidates : On a miss any replacement policy could either choose to replace any of the lines in the cache or choose not to place the miss causing line in the cache at all.
2. Self Replacement : The latter choice is referred to as a self-replacement or a cache bypass

Optimal Replacement Policy

On a miss replace the candidate to which an access is least imminent [Belady1966,Mattson1970,McFarling-thesis]

The Optimal Replacement Policy

1. Replacement Candidates : On a miss any replacement policy could either choose to replace any of the lines in the cache or choose not to place the miss causing line in the cache at all.
2. Self Replacement : The latter choice is referred to as a self-replacement or a cache bypass

Optimal Replacement Policy

On a miss replace the candidate to which an access is least imminent [Belady1966,Mattson1970,McFarling-thesis]

3. Lookahead Window : Window of accesses between miss causing access and the access to the least imminent replacement candidate. Single pass simulation of OPT make use of lookahead windows to identify replacement candidates and modify current cache state [Sugumar-SIGMETRICS1993]

Understanding OPT

Access Sequence	A_5	A_1	A_6	A_3	A_1	A_4	A_5	A_2	A_5	A_7	A_6	A_8
OPT order for A_5		0		1		2	3	4				
OPT order for A_6				0	1	2	3				4	

- Consider 4 way associative cache with one set initially containing lines (A_1,A_2,A_3,A_4), consider the access stream shown in table
- Access A_5 misses, replacement decision proceeds as follows

Understanding OPT

Access Sequence	A_5	A_1	A_6	A_3	A_1	A_4	A_5	A_2	A_5	A_7	A_6	A_8
OPT order for A_5		0		1		2	3	4				
OPT order for A_6				0	1	2	3				4	

- Consider 4 way associative cache with one set initially containing lines (A_1,A_2,A_3,A_4), consider the access stream shown in table
- Access A_5 misses, replacement decision proceeds as follows
  1. Identify replacement candidates : (A_1,A_2,A_3,A_4,A_5)

Understanding OPT

Access Sequence	A_5	A_1	A_6	A_3	A_1	A_4	A_5	A_2	A_5	A_7	A_6	A_8
OPT order for A_5		0		1		2	3	4				
OPT order for A_6				0	1	2	3				4	

- Consider 4 way associative cache with one set initially containing lines (A_1,A_2,A_3,A_4), consider the access stream shown in table
- Access A_5 misses, replacement decision proceeds as follows
  1. Identify replacement candidates : (A_1,A_2,A_3,A_4,A_5)
  2. Lookahead and gather imminence order : shown in table, lookahead window circled

Understanding OPT

Access Sequence	A_5	A_1	A_6	A_3	A_1	A_4	A_5	A_2	A_5	A_7	A_6	A_8
OPT order for A_5		0		1		2	3	4				
OPT order for A_6				0	1	2	3				4	

- Consider 4 way associative cache with one set initially containing lines (A_1,A_2,A_3,A_4), consider the access stream shown in table
- Access A_5 misses, replacement decision proceeds as follows
  1. Identify replacement candidates : (A_1,A_2,A_3,A_4,A_5)
  2. Lookahead and gather imminence order : shown in table, lookahead window circled
  3. Make replacement decision : A_5 replaces A_2

Understanding OPT

Access Sequence	A_5	A_1	A_6	A_3	A_1	A_4	A_5	A_2	A_5	A_7	A_6	A_8
OPT order for A_5		0		1		2	3	4				
OPT order for A_6				0	1	2	3				4	

- Consider 4 way associative cache with one set initially containing lines (A_1,A_2,A_3,A_4), consider the access stream shown in table
- Access A_5 misses, replacement decision proceeds as follows
  1. Identify replacement candidates : (A_1,A_2,A_3,A_4,A_5)
  2. Lookahead and gather imminence order : shown in table, lookahead window circled
  3. Make replacement decision : A_5 replaces A_2
- A_6 self-replaces, lookahead window and imminence order in table

Outline

Motivation

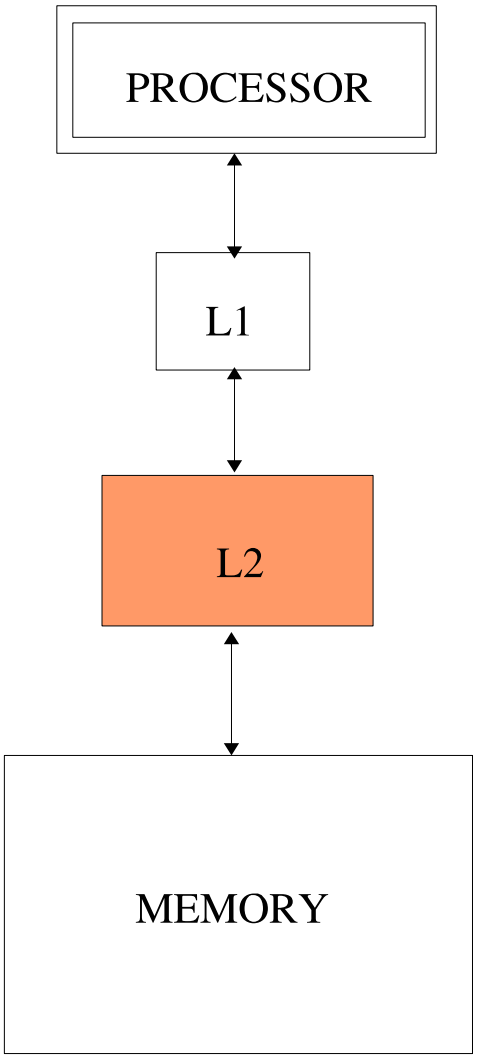
LRU-OPT gap

Smaller OPT cache better than larger LRU cache $\Longrightarrow$ Focus on matching OPT performance even on a smaller cache

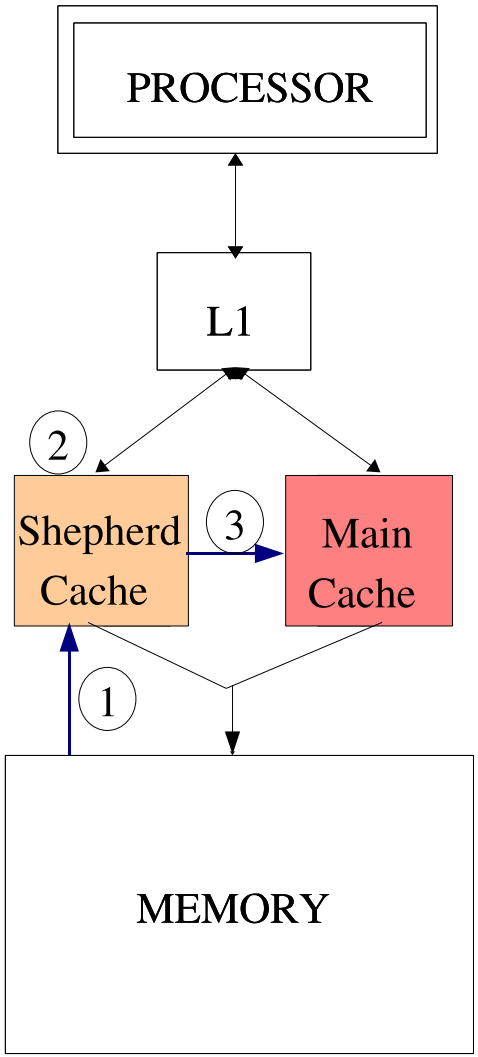
Lookahead

OPT requires lookahead to to find least imminent line $\Longrightarrow$ Use part of cache to aid in emulating OPT for remaining cache

Emulating OPT with a Shepherd Cache

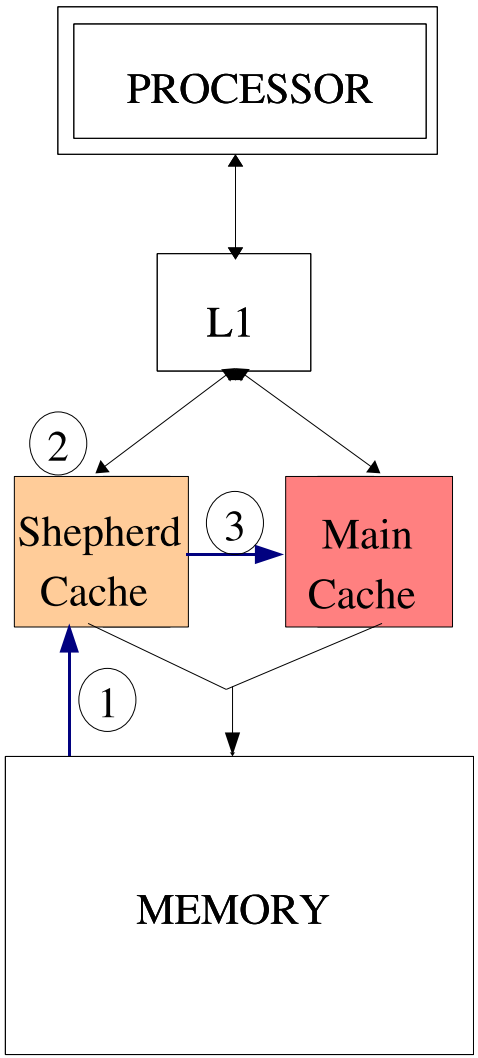


Emulating OPT with a Shepherd Cache



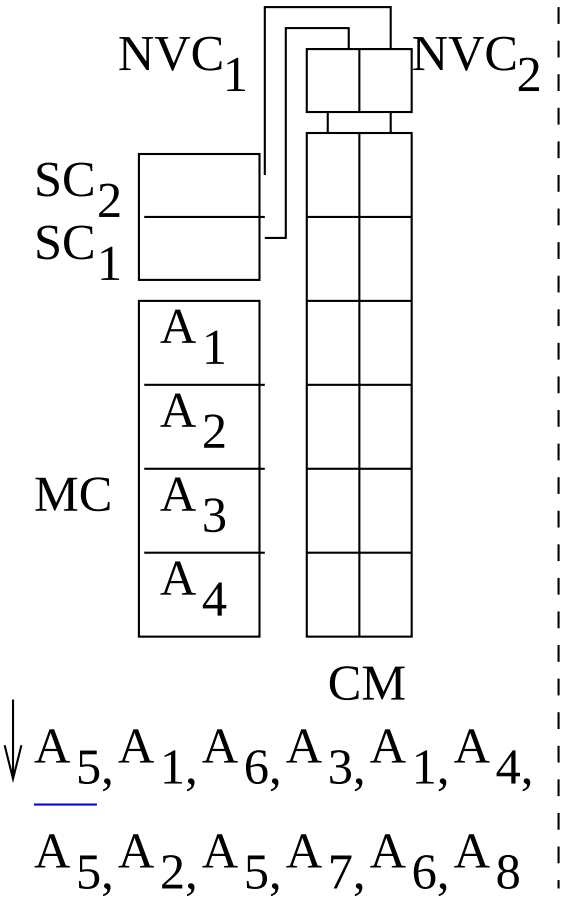
- Split the cache into two logical parts
  - Main Cache (MC) for which optimal replacement is emulated
  - Shepherd Cache (SC) used to provide a lookahead and guide replacements from MC towards OPT

Emulating OPT with a Shepherd Cache

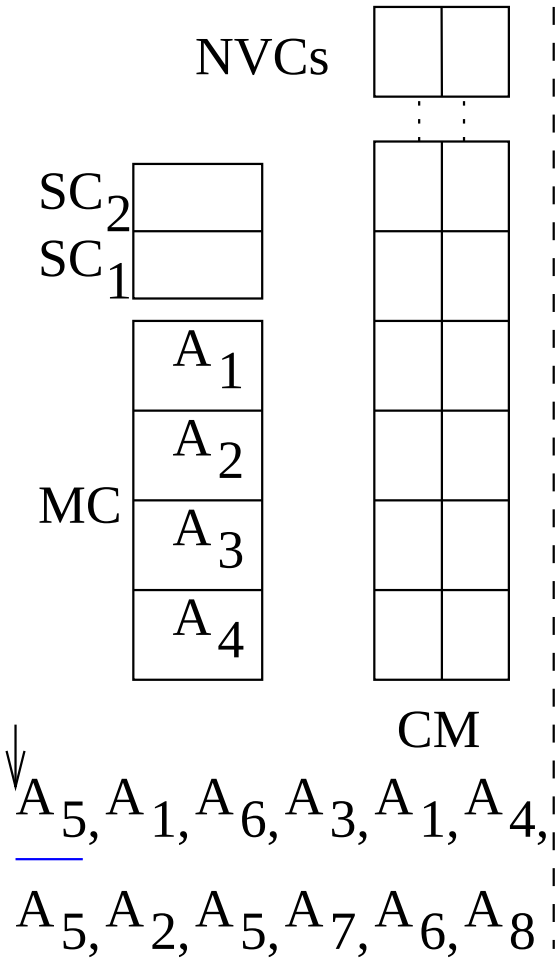


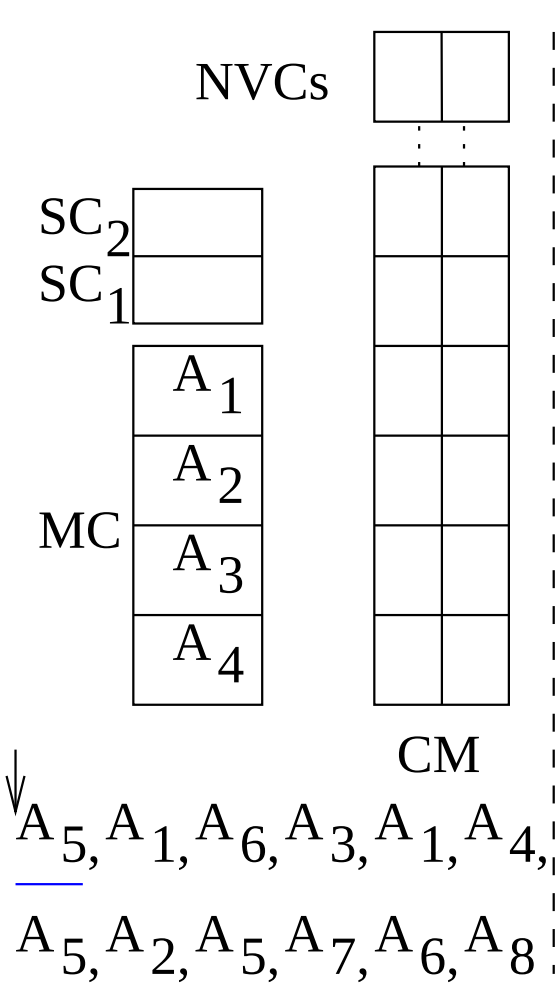
- Split the cache into two logical parts
  - Main Cache (MC) for which optimal replacement is emulated
  - Shepherd Cache (SC) used to provide a lookahead and guide replacements from MC towards OPT
- Operation
  1. Buffer lines temporarily in SC before moving them to MC, SC acts as a FIFO buffer
  2. While in SC, gather imminence information and emulate lookahead
  3. When forced out of SC, make an MC replacement based on the gathered imminence order

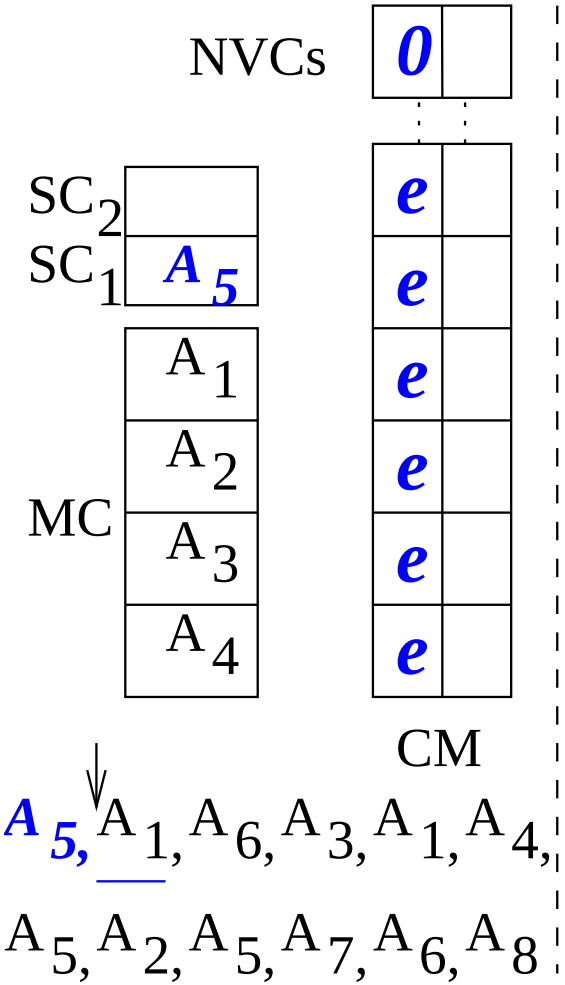
Overview of Shepherd Caching

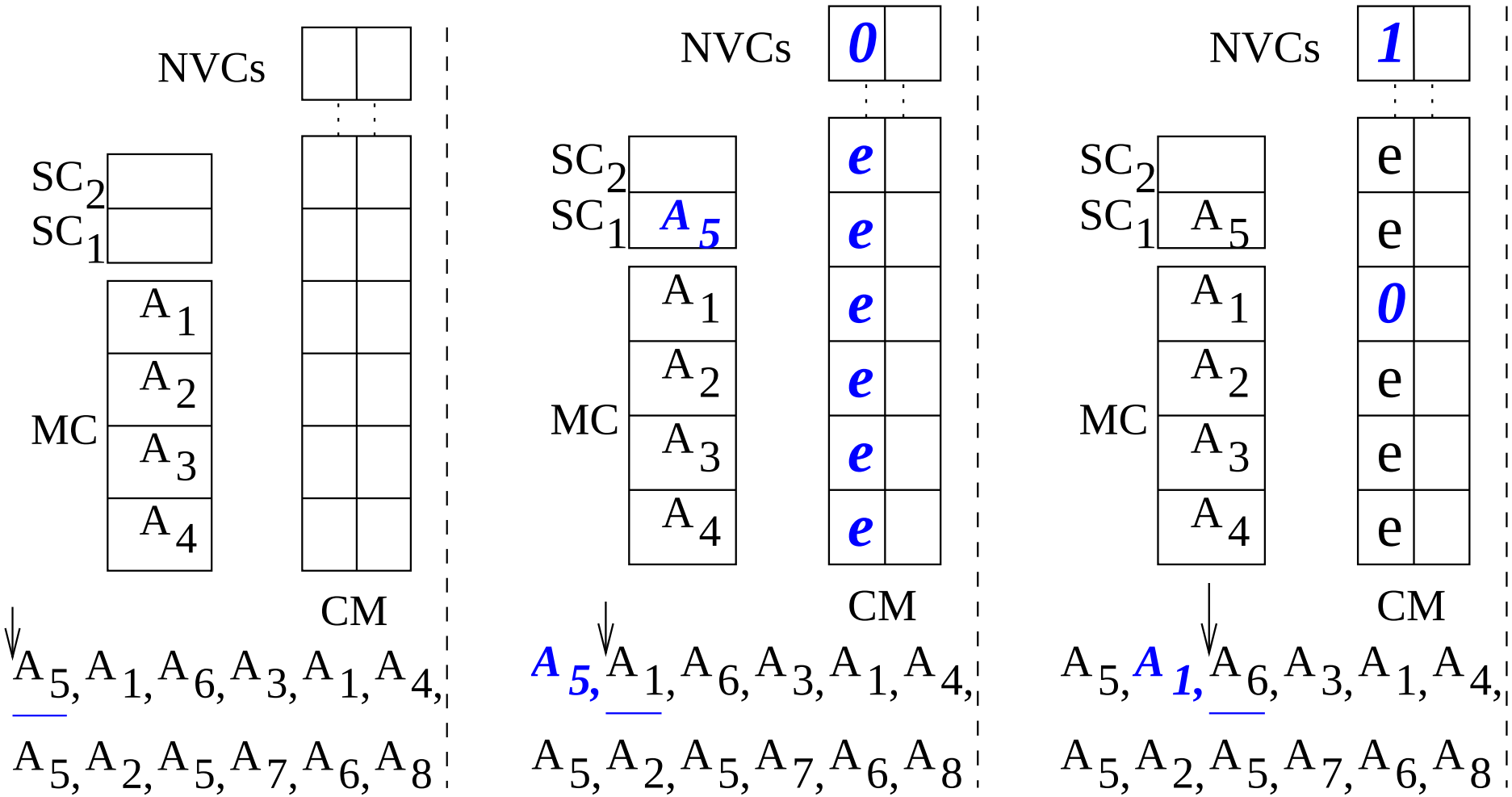


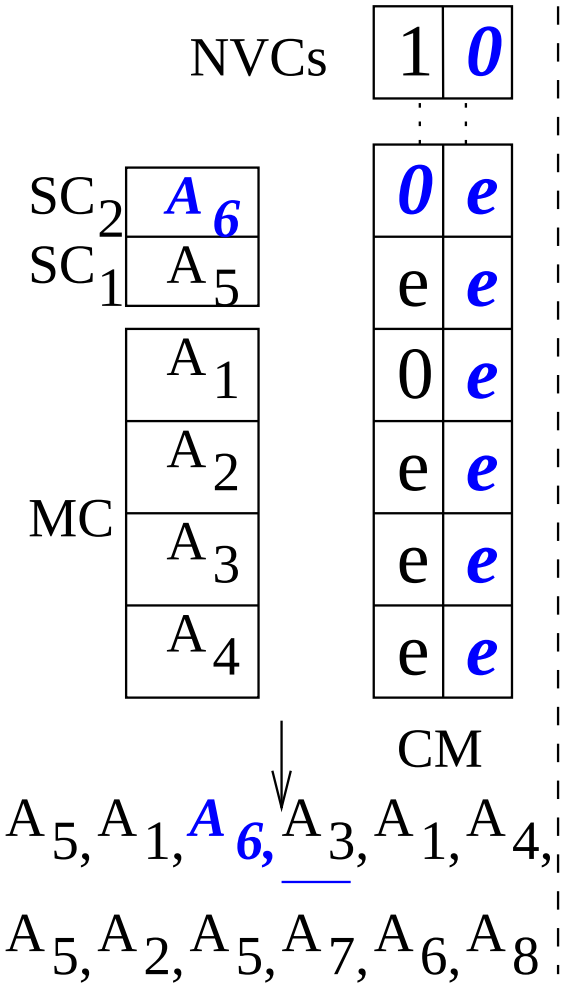
- To emulate MC with 4 ways per set and 2 SC ways per set
- To gather imminence order add a counter matrix (CM)
- CM has one column per SC way to track imminence order w.r.t to it
- CM has one row per SC and MC line as any of them can be a replacement candidate
- Each column has one Next Value Counter (NVC) to track the next value to assign along column

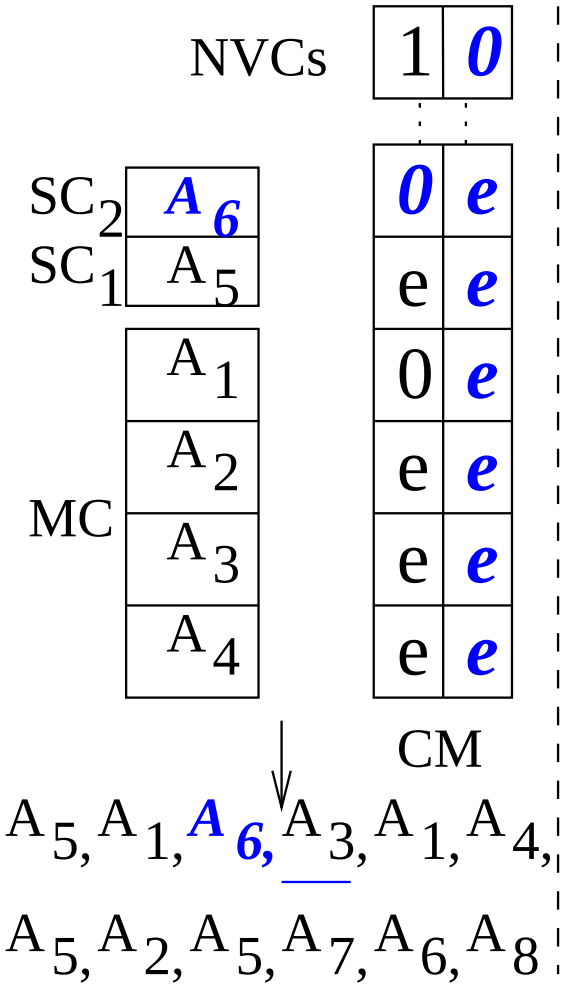

NVCs
SC2
SC1
A1
A2
MC
A3
A4
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

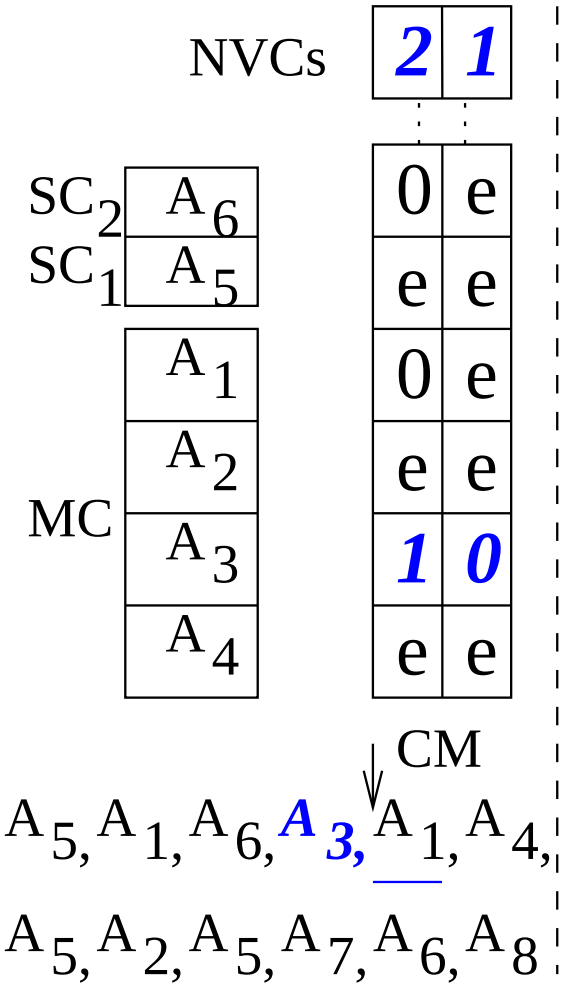

NVCs
SC2
SC1
MC
A1
A2
A3
A4
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

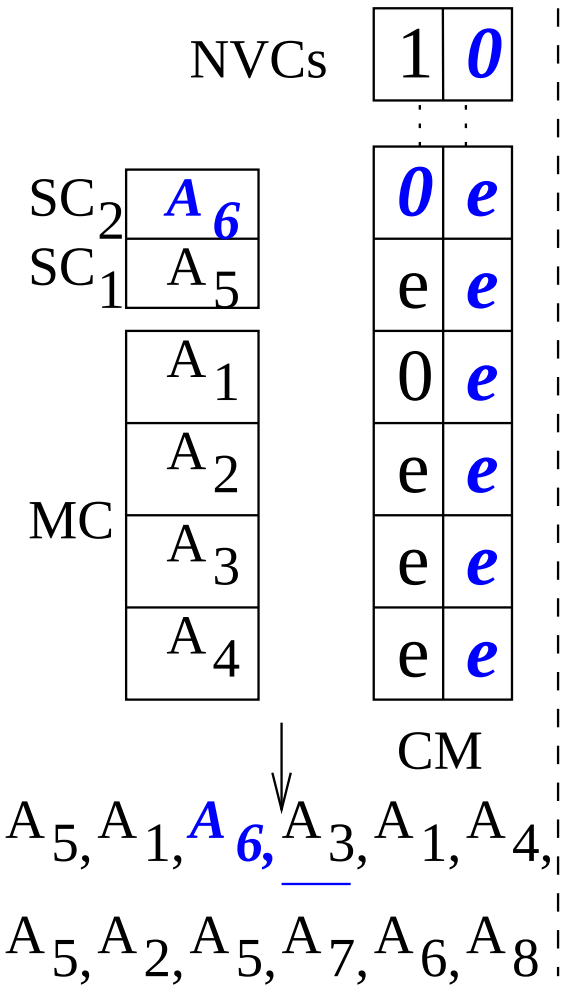

NVCs
0
e
SC2
SC1
A5
MC
A1
A2
A3
A4
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

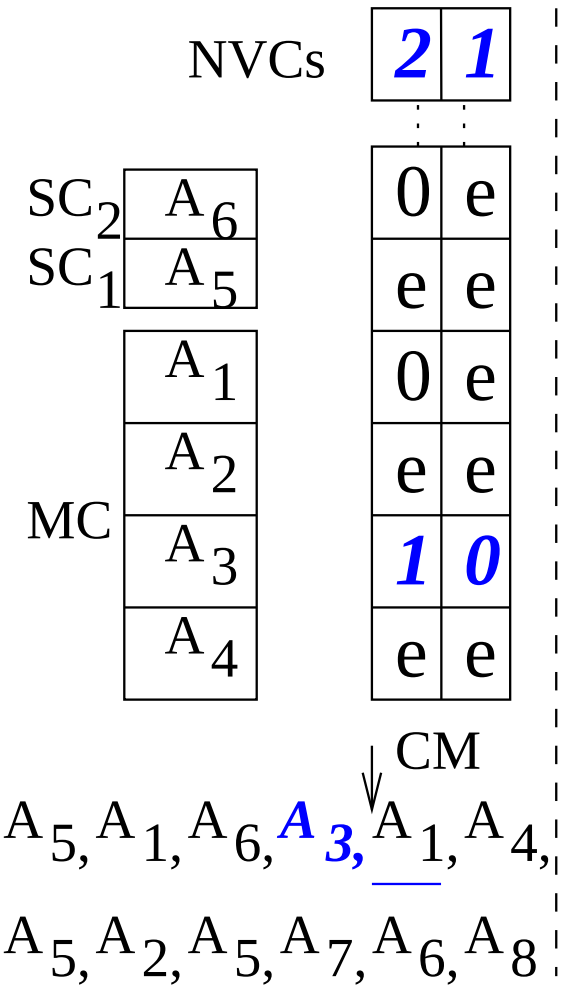

NVCs
SC2
SC1
MC
A1
A2
A3
A4
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8
0
e
A5
NVCs
SC2
SC1
MC
A1
A2
A3
A4
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8
1
e
0
A5
NVCs
SC2
SC1
MC
A1
A2
A3
A4
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

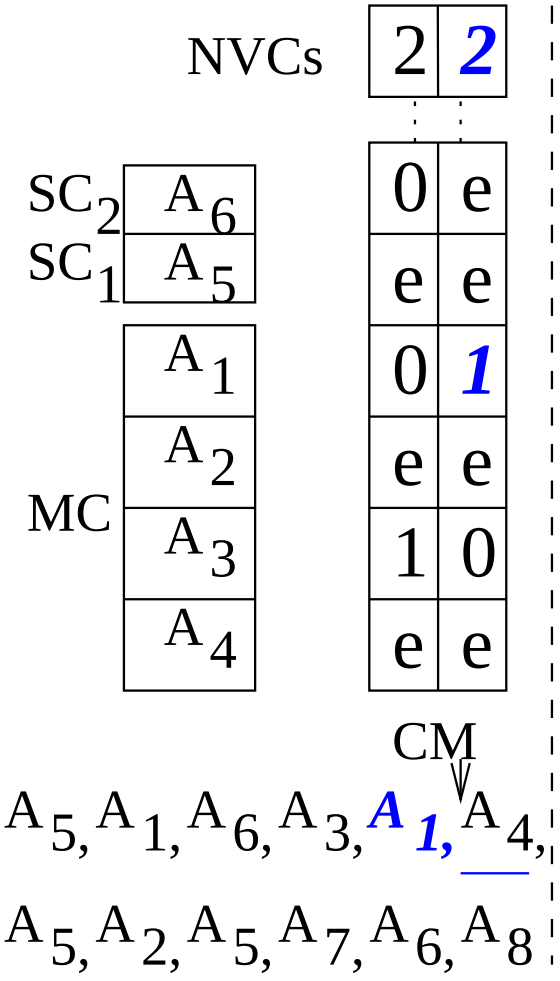

NVCs
1
0
SC2
A6
0
e
SC1
A5
e
e
A1
0
e
A2
e
e
MC
A3
e
e
A4
e
e
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

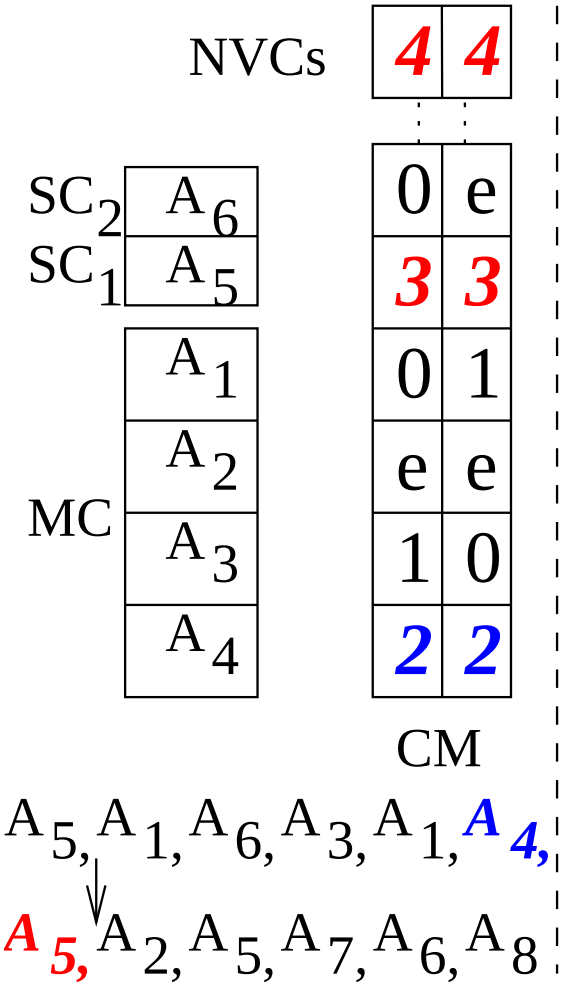

NVCs
1
0
SC2
A6
SC1
A5
MC
A1
A2
A3
A4
0
e
0
e
e
e
e
e
e
e
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

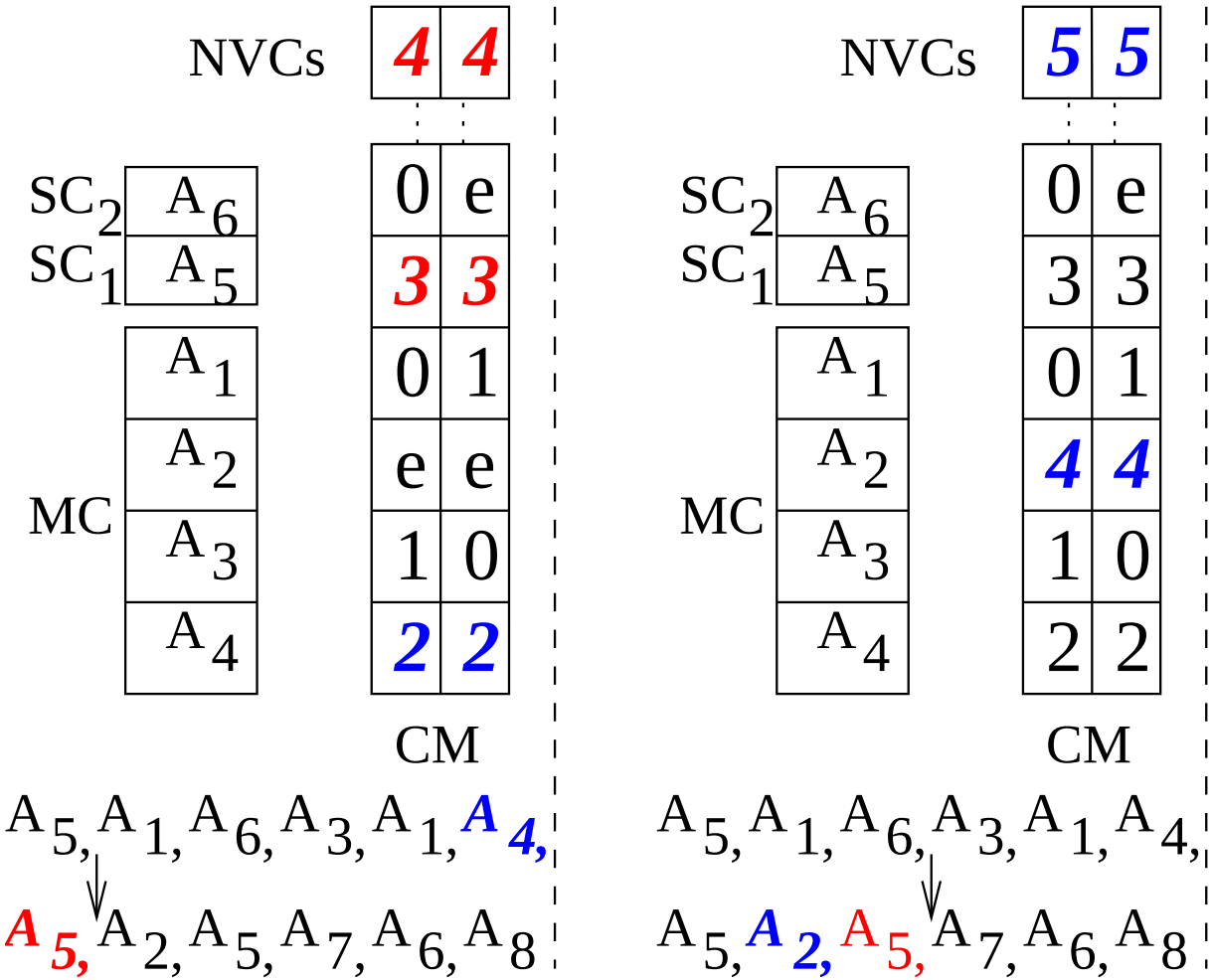

NVCs
2
1
SC2
A6
SC1
A5
MC
A1
A2
A3
A4
0
e
e
e
0
e
e
e
1
0
e
e
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

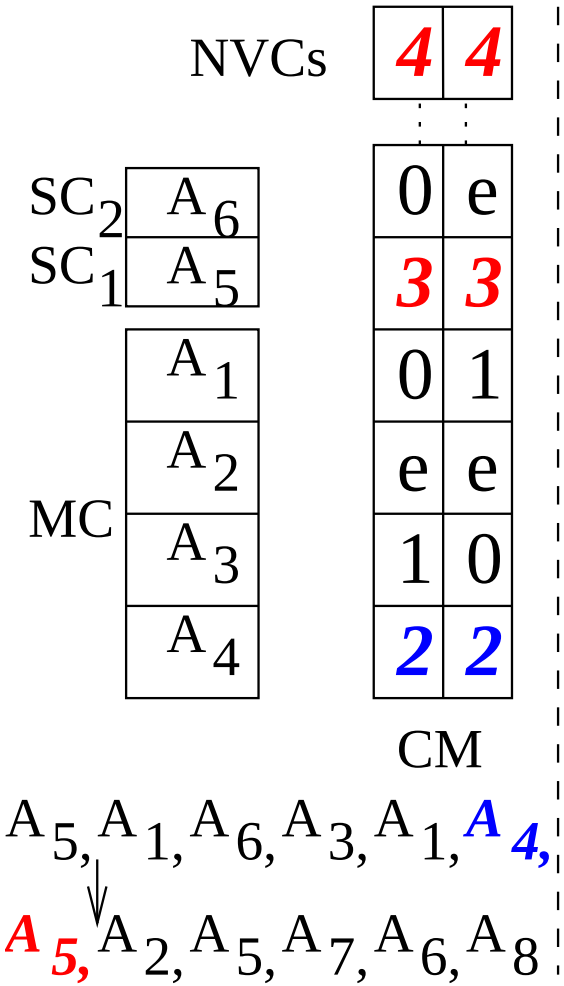

NVCs
1 0
SC2 A6
SC1 A5
MC
A1
A2
A3
A4
0 e
e e
0 e
e e
e e
e e
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

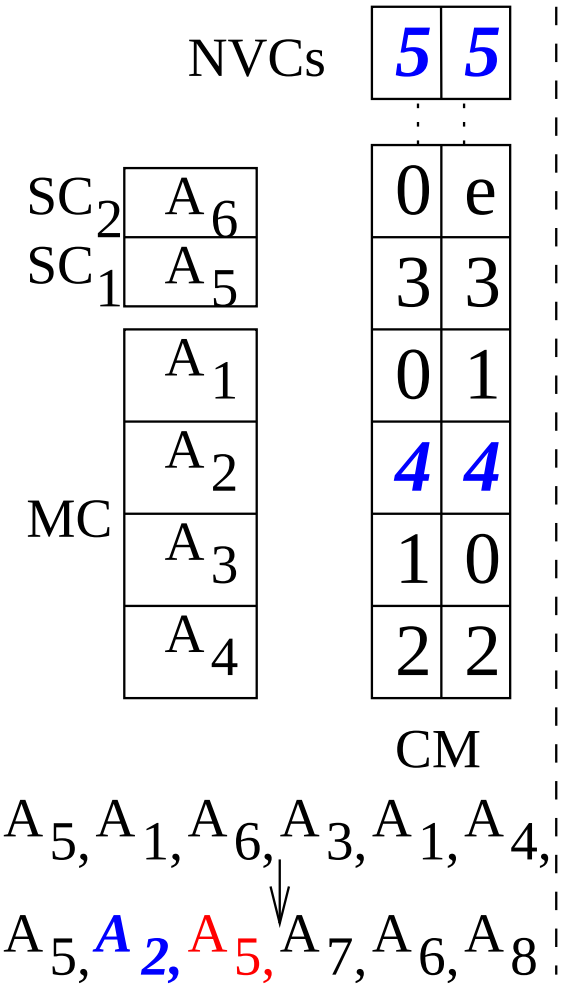

NVCs
2 1
SC2 A6
SC1 A5
MC
A1
A2
A3
A4
0 e
e e
0 e
e e
1 0
e e
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

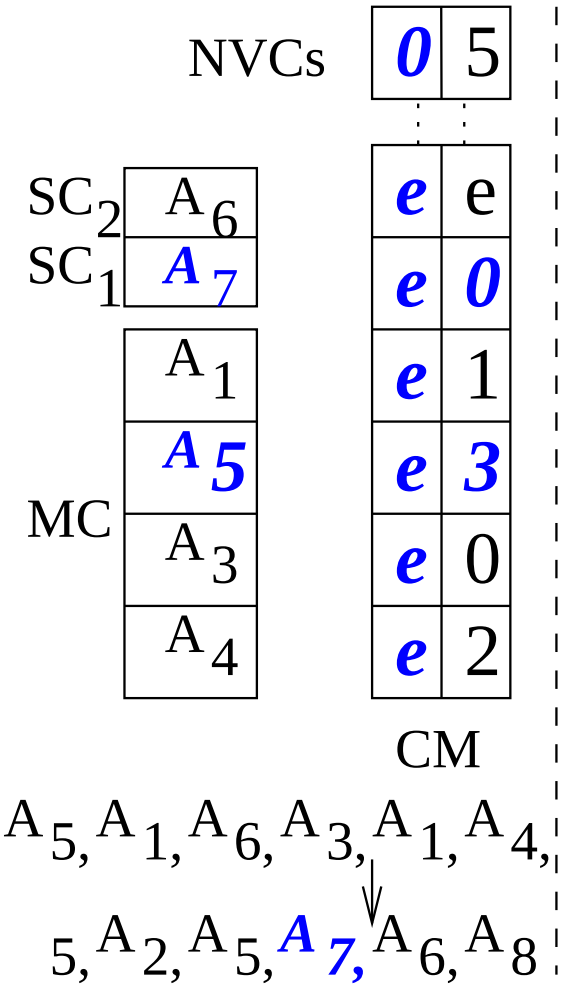

NVCs
2 2
SC2 A6
SC1 A5
MC
A1
A2
A3
A4
0 e
e e
0 1
e e
1 0
e e
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

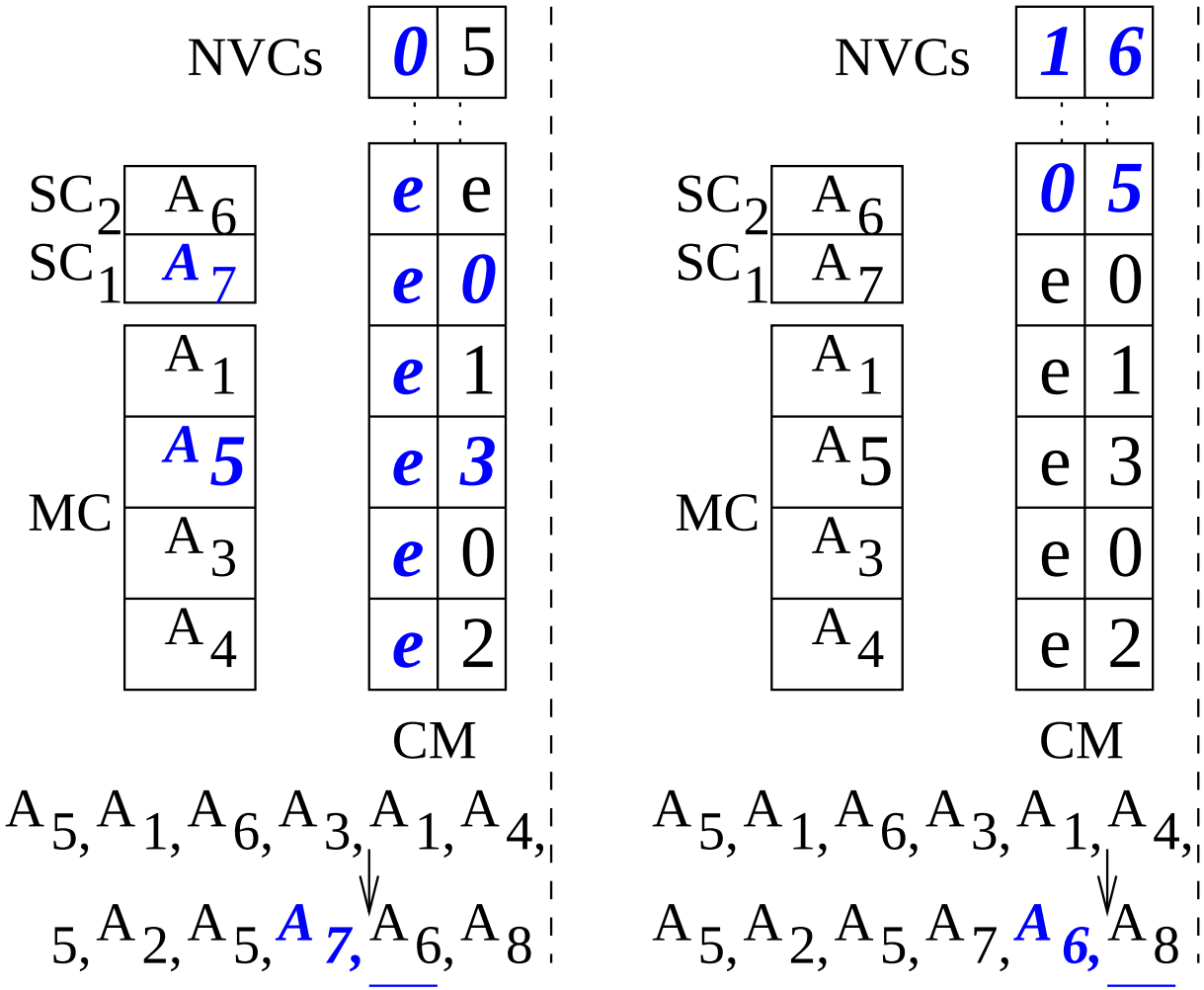

NVCs
4 4
SC2
A6
0 e
SC1
A5
3 3
MC
A1
0 1
A2
e e
A3
1 0
A4
2 2
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

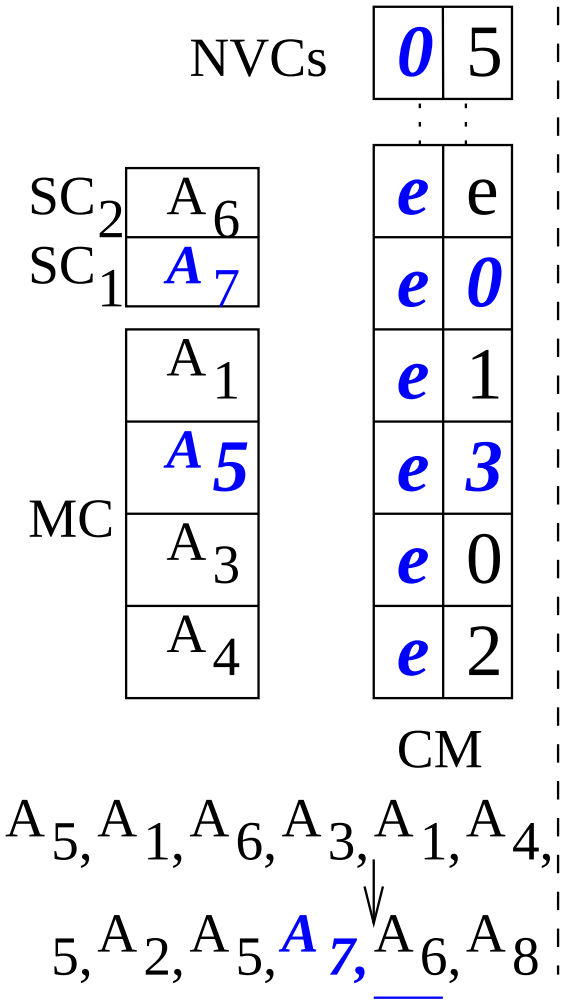

NVCs
4 4
SC2 A6
SC1 A5
MC A1 A2 A3 A4
0 e
3 3
0 1
e e
1 0
2 2
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8
NVCs
5 5
SC2 A6
SC1 A5
MC A1 A2 A3 A4
0 e
3 3
0 1
4 4
1 0
2 2
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

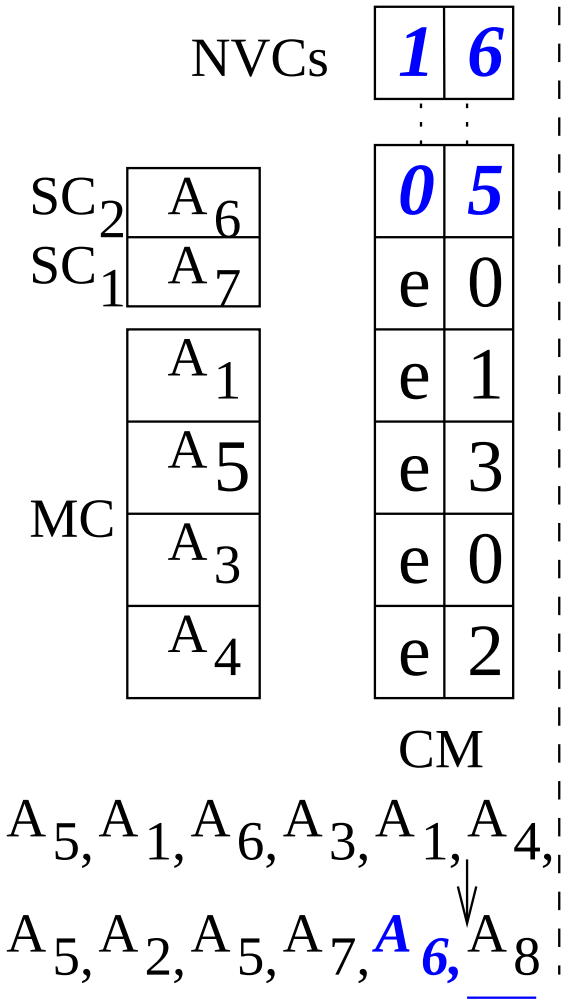

NVCs
4 4
SC2 A6
SC1 A5
MC A1 A2 A3 A4
0 e
3 3
0 1
e e
1 0
2 2
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

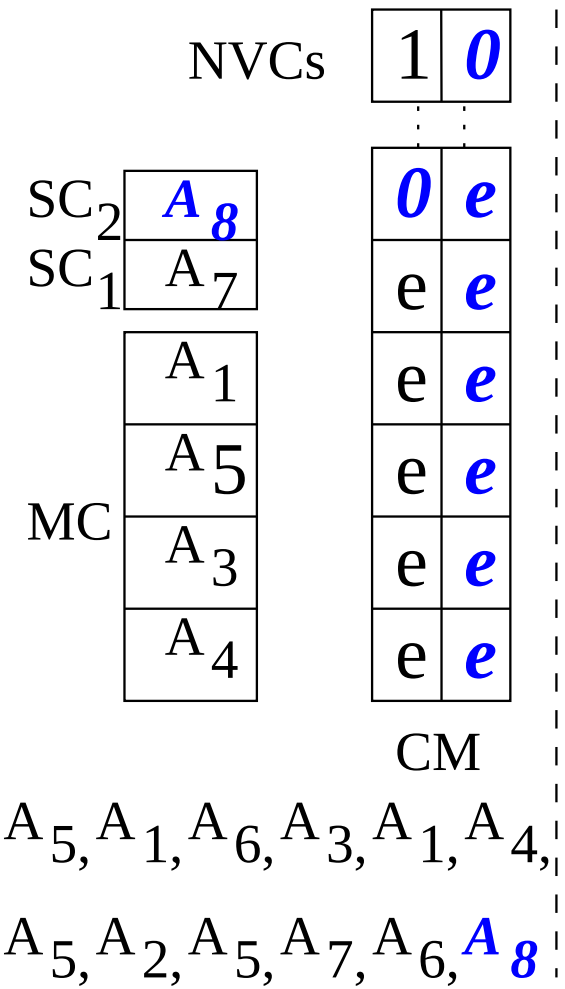

NVCs
5 5
SC2 A6
SC1 A5
MC A1 A2 A3 A4
0 e
3 3
0 1
4 4
1 0
2 2
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8


NVCs
0 5
SC2 A6
SC1 A7
MC A1 A5 A3 A4
e e
e 0
e 1
e 3
e 0
e 2
CM
A5, A1, A6, A3, A1, A4,
5, A2, A5, A7, A6, A8


NVCs
0 5
SC2 A6
SC1 A7
e e
e 0
MC
A1
A5
A3
A4
e 1
e 3
e 0
e 2
CM
A5, A1, A6, A3, A1, A4,
5, A2, A5, A7, A6, A8
NVCs
1 6
SC2 A6
SC1 A7
0 5
e 0
MC
A1
A5
A3
A4
e 1
e 3
e 0
e 2
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

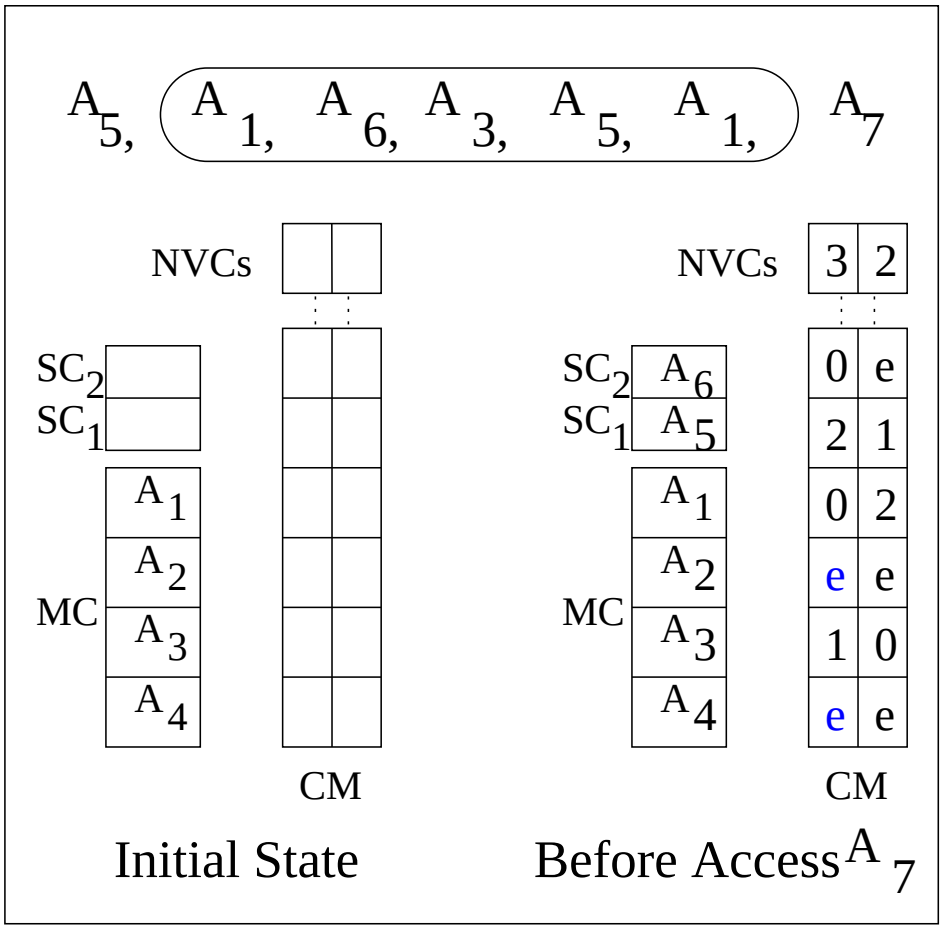

NVCs
0 5
SC2 A6
SC1 A7
MC A1 A5 A3 A4
e e
e 0
e 1
e 3
e 0
e 2
CM
A5, A1, A6, A3, A1, A4,
5, A2, A5, A7, A6, A8


NVCs
1 6
SC2 A6
SC1 A7
MC A1 A5 A3 A4
0 5
e 0
e 1
e 3
e 0
e 2
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8


NVCs
1 0
SC2 A8
SC1 A7
MC A1 A5 A3 A4
0 e
e e
e e
e e
e e
e e
CM
A5, A1, A6, A3, A1, A4,
A5, A2, A5, A7, A6, A8

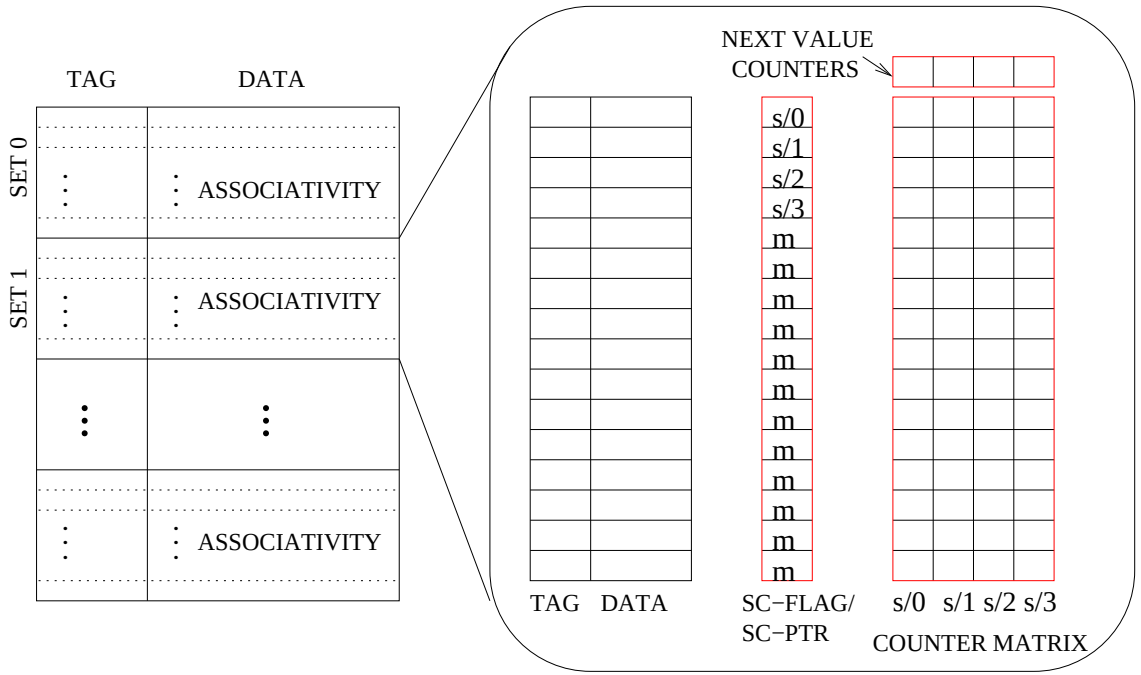
Baseline MC replacement

- Lookahead provided may-not always be sufficient to identify least imminent replacement
- Partial imminence order used to restrict replacement choice to less imminent candidates
- Use baseline MC replacement to choose victim among these
- Baseline MC replacement can be any standard policy, we evaluate LRU, LFU and random
- When line moves from SC to MC, place it in LRU/LFU position



A_5 replaces one of A_2 or A_4 based on baseline MC replacement

Cache Organization



Cache Organization (SC-4, MC-12)

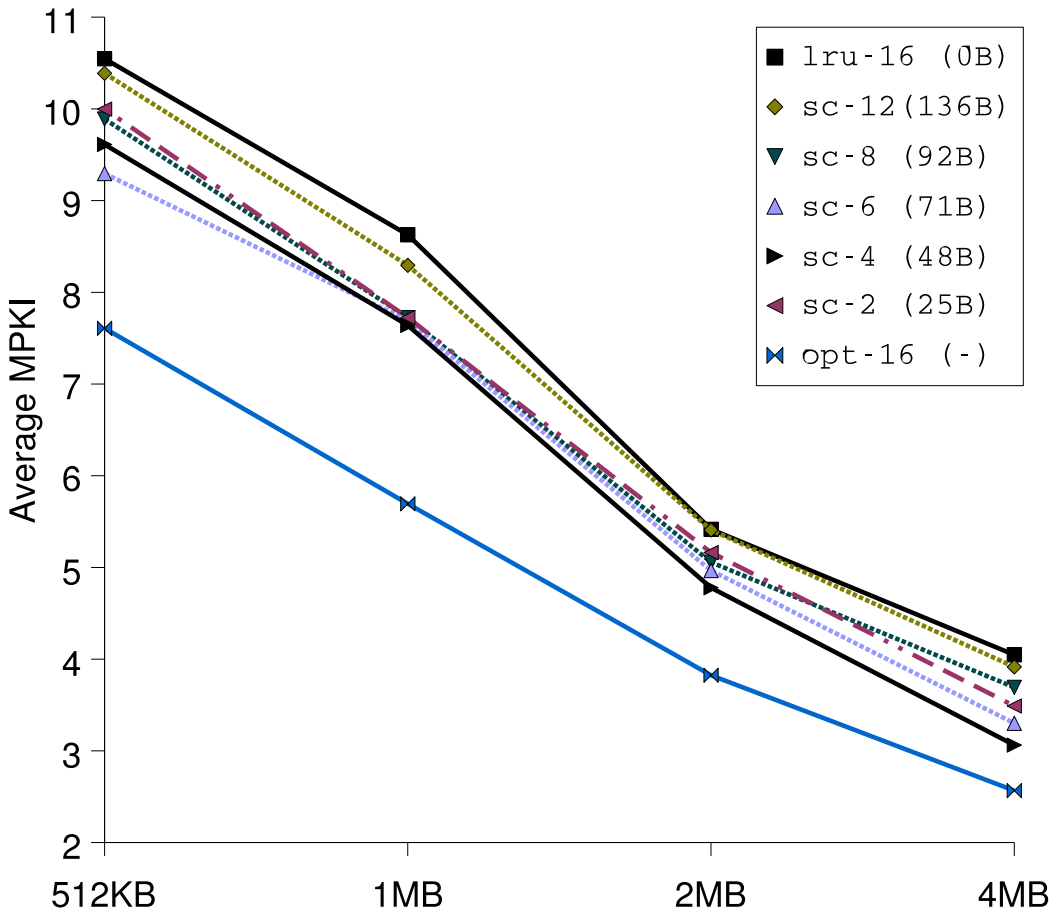
- Divide cache into SC and MC with some ways of each set are SC ways
- Use SC flag to identify SC ways, use SC-ptr to link SC line to its column
- No need to physically move lines from SC to MC, just adjust SC-flag/ptr
- Larger SC-assoc $\Longrightarrow$ more lookahead $\Longrightarrow$ OPT in smaller MC

Outline

Simulation Methodology

1. Memory hierarchy
   - L2 Cache:- 512KB,1MB,2MB and 4MB, 16 way, 128B lines, 8 cycle
   - L1 cache:- Both L1-I and L1-D are 16KB 2 way, 32B lines, 2 cycle
   - Memory:- Infinite size 400 cycle
   - Infinite MSHR and store buffer
2. SC+MC configuration
   - Total assoc 16, performance evaluated for (SC-2, MC-14) to (SC-14, MC-2)
   - Schemes referred to by SC-associativity, SC-4 $\Longrightarrow$ (SC-4, MC-12)
3. Simulation methodology
   - Benchmarks Suite:- SPEC 2006 all 26 benchmarks with ref inputs
   - Instructions:- Fast-forward 2 billion run 3 billion
   - Methodology:- Trace driven for MPKI, Simplescalar for system performance
   - Processor configuration:- Similar to recent studies

Trade-off between SC and MC

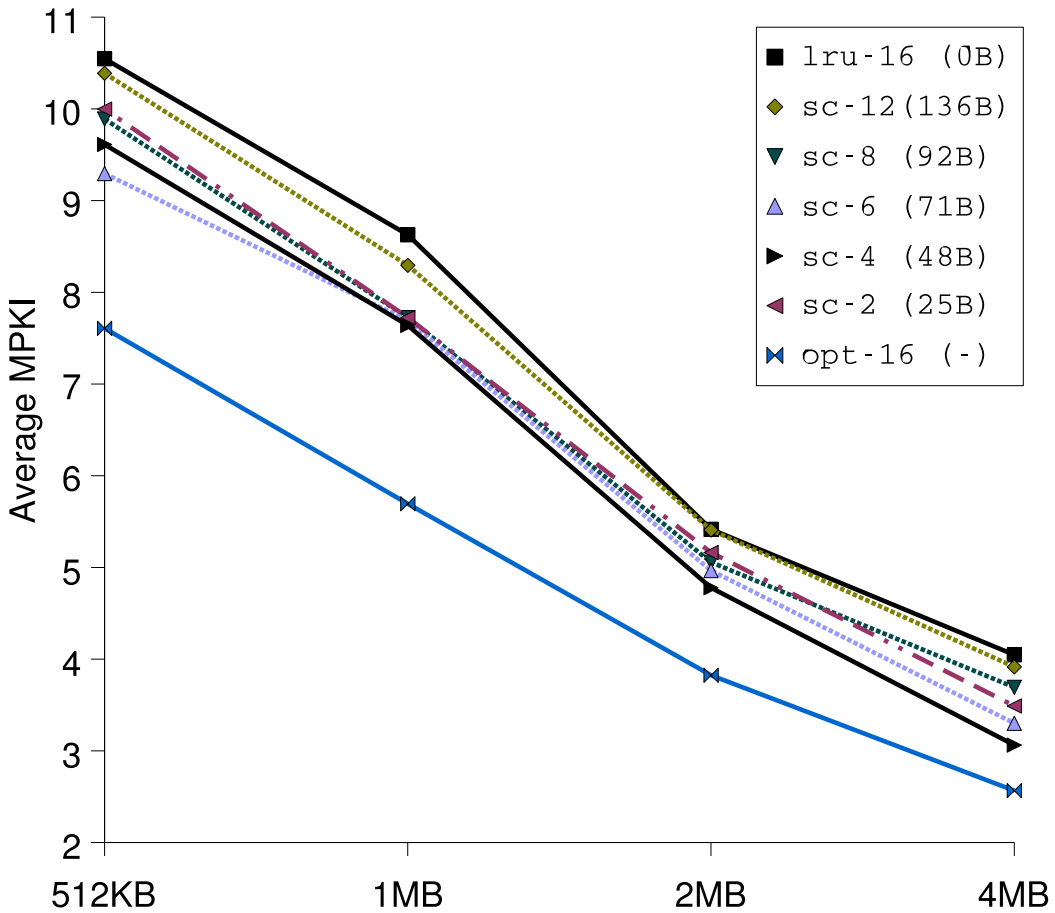


Avg MPKI over SPEC2000 suite

SC-assoc vs MC-assoc

- Marginal gains when MC size less than half the cache
- Peak gains : SC-6 to SC-4
- *Quarter to half of cache should be for lookahead*
- Size overhead 48B-92B, less than one cache block (128B)

Bridging the performance gap



Avg MPKI over SPEC2000 suite

Bridging the LRU-OPT gap

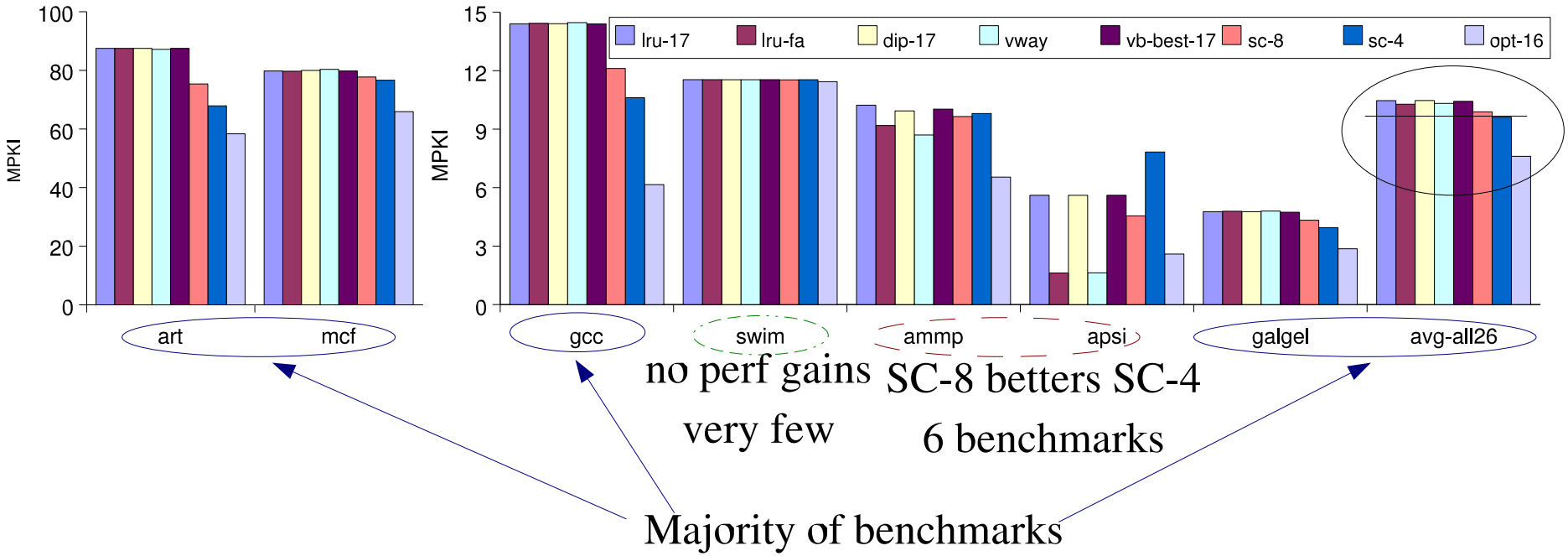
- *SC-4 bridges 32-52% of gap*
- SC moves closer to OPT as cache size increases

Comparison with related work

Scheme	Parameters
DIP [Qureshi-ISCA07]	16/32/64 dedicated sets, epsilon $1/32$
V-way cache [Qureshi-ISCA05]	32 tags per set, tdr 2 global reuse distance replacement
Victim Buffer [Jouppi-ISCA90]	MC+VB, VB same assoc as SC
fa cache	fully associative with LRU
OPT-16	16 way associative OPT replacement

Performance compared with base LRU, DIP, victim buffer with one additional way per set to compensate for size overhead

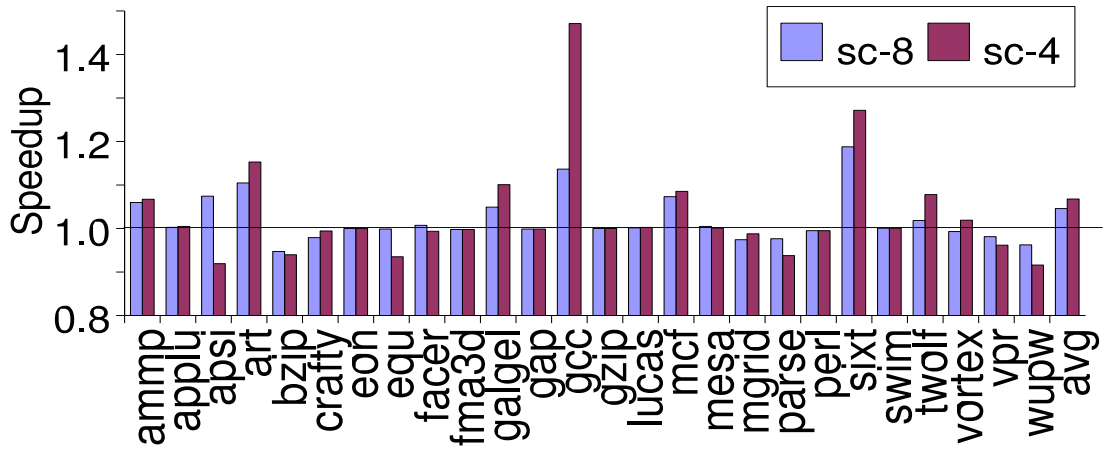
Comparison with related schemes



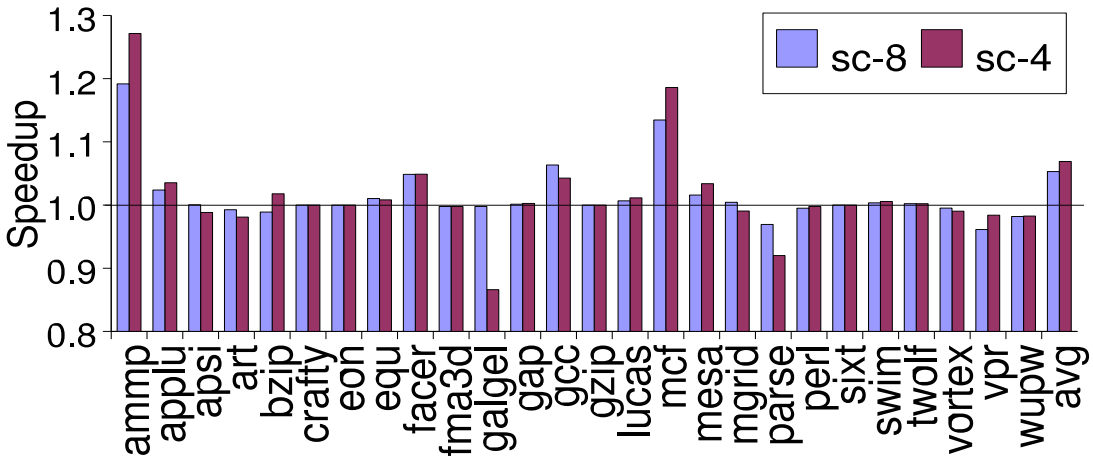
MPKI for 512KB 16 way cache, select benchmarks, all 26 in average

- On average both SC-4 and SC-8 out-performs LRU, DIP, v-way, fa and victim by 4-10%
- For most benchmarks SC-4 performs better than SC-8
- Different SC associativities best for different benchmarks

Speedup in CPI



512KB



2MB

Average speedup of about 7% with SC-4 and 5% with SC-8

Outline

Conclusions and Future Work

Conclusion

- Using part of cache for lookahead and emulating OPT in remaining cache effective at improving L2 performance
- Quarter to Half of cache space should be dedicated to SC
- Performs better than related proposals and successfully bridges 32-52% of the LRU-OPT gap

Future Work

- Different SC associativity works best with different benchmarks, can the SC-associativity be adapted at runtime?
- Study the benefits of dissociating SC and MC?
- Tuning the organization to a shared CMP cache?