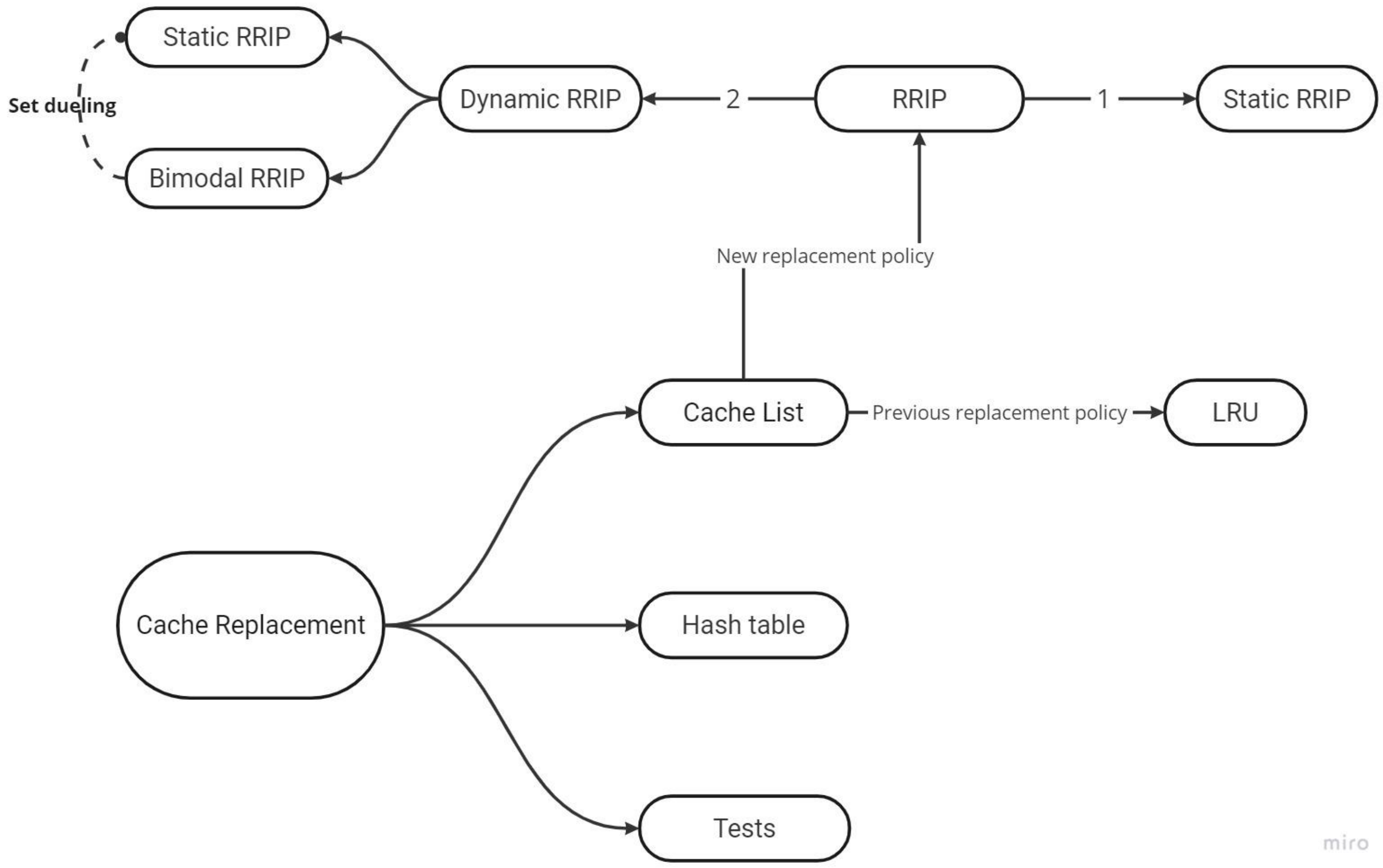
RRIP

High Performance Cache Replacement using Re-Reference Interval Prediction(RRIP)

Contributors: Minimullina Aliya,
Samoylov Georgy, Grishin Mikhail.


Static RRIP
Set dueling
Bimodal RRIP
Dynamic RRIP
2
RRIP
1
Static RRIP
New replacement policy
Cache List
Previous replacement policy
LRU
Cache Replacement
Hash table
Tests
miro

A List Node (the RRIP list is implemented using Doubly Linked List)

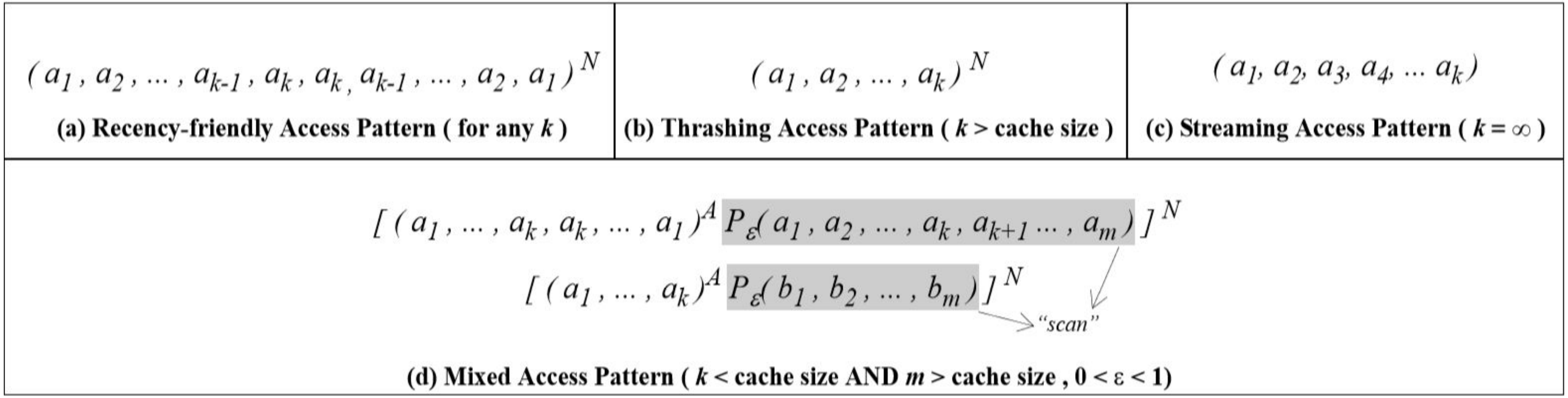
```
struct node_t {
    struct node_t *next, *prev;
    long data;      //the data stored in this Node
    unsigned value; //the RRIP value stored by a 2-bit
register per Node
};
```

RRPV of 0 represents a near-immediate re-reference interval of the cache block;

RRPV of 2 — a long re-reference interval;

RRPV of 3 — a distant re-reference interval.

Cache Access Patterns



$(a_1, a_2, \ldots, a_{k-1}, a_k, a_k, a_{k-1}, \ldots, a_2, a_1)^N$ **(a) Recency-friendly Access Pattern (for any *k*)**	$(a_1, a_2, \ldots, a_k)^N$ **(b) Thrashing Access Pattern (*k* > cache size)**	$(a_1, a_2, a_3, a_4, \ldots a_k)$ **(c) Streaming Access Pattern (*k* = ∞)**

$$[\,(a_1, \ldots, a_k, a_k, \ldots, a_1)^A\, \underbrace{P_\varepsilon(a_1, a_2, \ldots, a_k, a_{k+1} \ldots, a_m)}_{\text{“scan”}}\,]^N$$

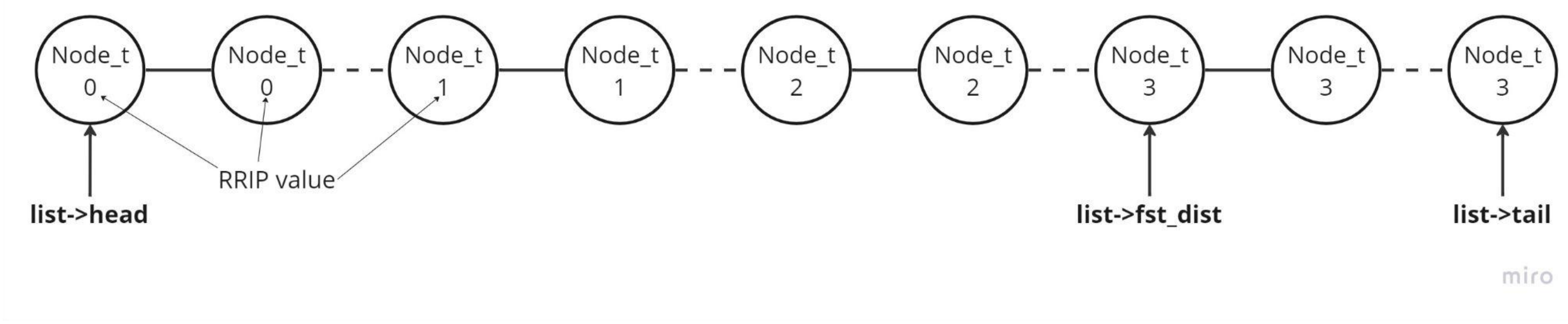
$$[\,(a_1, \ldots, a_k)^A\, \underbrace{P_\varepsilon(b_1, b_2, \ldots, b_m)}_{\text{“scan”}}\,]^N$$

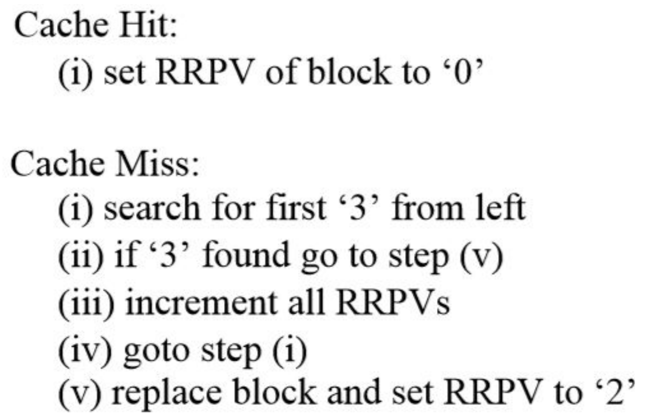
(d) Mixed Access Pattern (*k* < cache size AND *m* > cache size , $0 < \varepsilon < 1$)

In general, for associativity A, active working set size w ($w < A$), and scan length S_{len}, M-bit SRRIP is scan-resistant when

$$S_{len} <= (2^M - 1) * (A - w) \qquad \text{(Eq. 1)}$$

A List (a collection of Nodes in ascending order according to their RRIP)



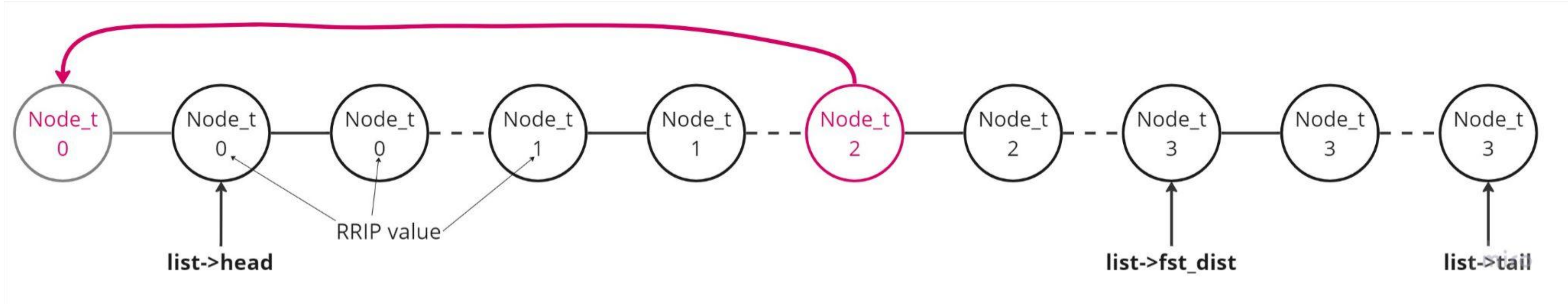


```
Cache Hit:
    (i) set RRPV of block to '0'

Cache Miss:
    (i) search for first '3' from left
    (ii) if '3' found go to step (v)
    (iii) increment all RRPVs
    (iv) goto step (i)
    (v) replace block and set RRPV to '2'
```

The RRIP replacement policy

A List (a collection of Nodes in ascending order according to their RRIP)



Cache Hit:

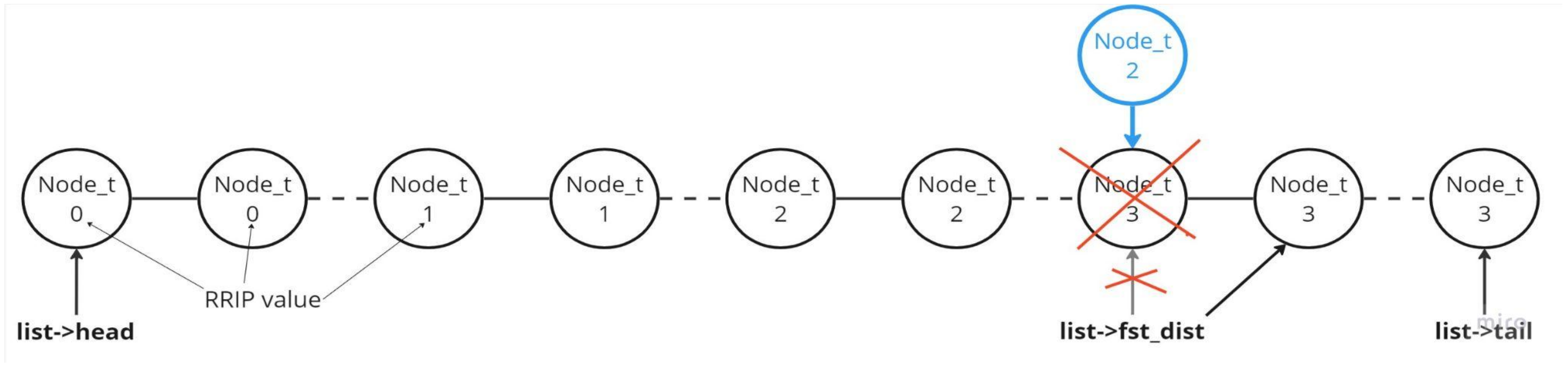
(i) set RRPV of block to ‘0’

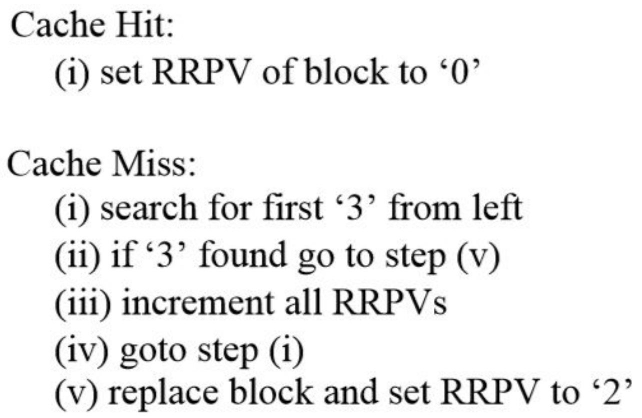
Cache Miss:

(i) search for first ‘3’ from left
(ii) if ‘3’ found go to step (v)
(iii) increment all RRPVs
(iv) goto step (i)
(v) replace block and set RRPV to ‘2’

The RRIP replacement policy

A List (a collection of Nodes in ascending order according to their RRIP)




Cache Hit:

(i) set RRPV of block to ‘0’

Cache Miss:

(i) search for first ‘3’ from left

(ii) if ‘3’ found go to step (v)

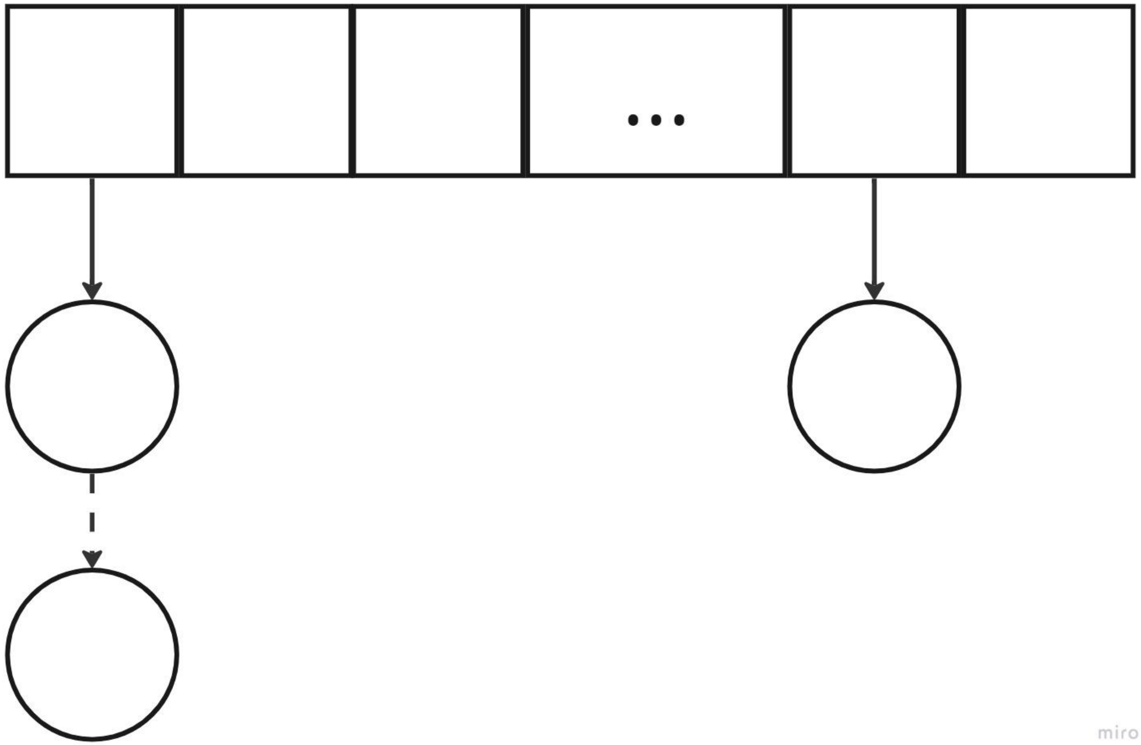
(iii) increment all RRPVs

(iv) goto step (i)

(v) replace block and set RRPV to ‘2’

The RRIP replacement policy

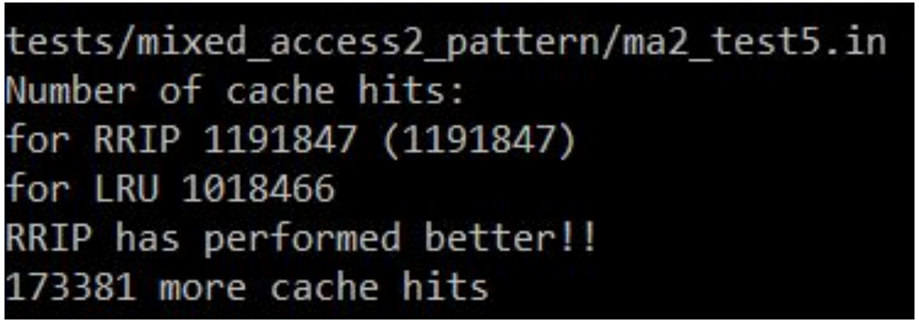
Hash table



Comparison of efficiency

☺️with LRU
☺️with RRIP that is implemented by simple hash(an array of double pointers)

Example:



```
tests/mixed_access2_pattern/ma2_test5.in
Number of cache hits:
for RRIP 1191847 (1191847)
for LRU 1018466
RRIP has performed better!!
173381 more cache hits
```

TODO: DRRIP replacement policy

SRRIP is scan-resistant.
DRRIP is scan-resistant and thrash-resistant.

(under construction)