Иерархия алгоритмов

LRU

LFU

MQ

LRD

CLOCK

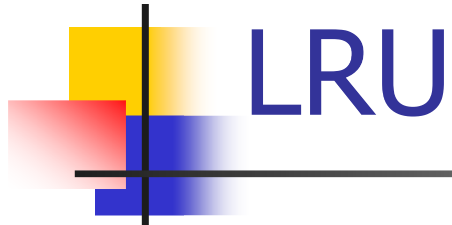
ARC

GCLOCK

CAR

Improved GCLOCK (V1,V2)

CART



LRU

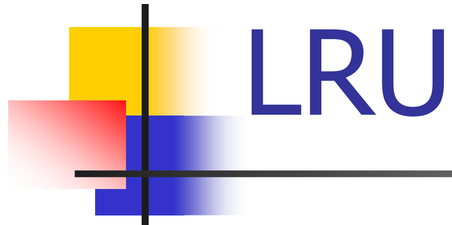
- LRU – “least recently used”

Один из наиболее известных и популярных алгоритмов замещения.

Суть: при необходимости выталкивания из кэша страницы выбирается та, которая дольше всего не использовалась.

Достоинства:

- константное время выполнения и использование памяти.



LRU

Недостатки:

- приходится защищать блокировкой (при multi-thread) страницу MRU => упорядочивать работу со страницами.
- алгоритм не учитывает ситуации, когда к определенным страницам обращения производятся часто, но с периодом, превышающим размер кэша (т.е. страница успевает покинуть кэш).
- как следствие предыдущего пункта, плохо работают scan-операции

CLOCK

Автор: Frank Corbato

Суть: Каждой странице в кэше ставится в соответствие флаг (1 бит) – “page reference bit”.

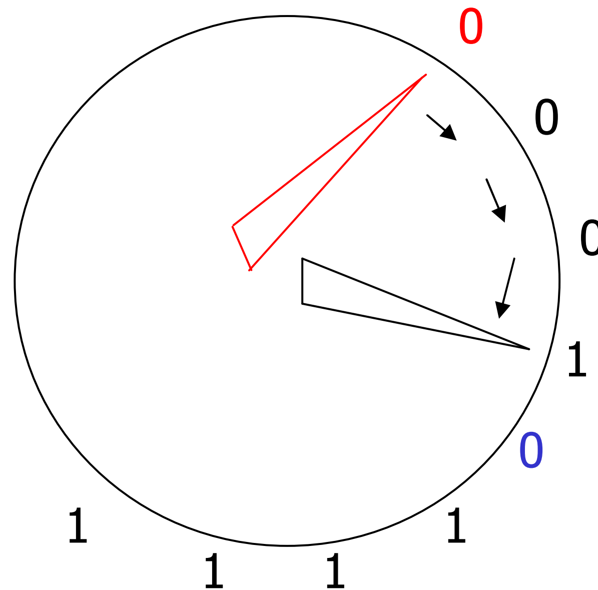
При попадании страницы в кэш, бит устанавливается в значение 1, точно также действуем и в случае, когда адресуемая запросом страница уже находится в кэше.

CLOCK

При необходимости произвести замещение страницы, производится цикличный просмотр всех страниц в кэше. Бит очередной страницы устанавливается в 0, если ранее значение было 1, если же находится страница со значением 0 – она замещается. Просмотренные страницы помещаются в конец очереди.

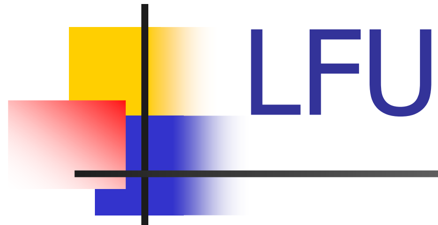
CLOCK

Таким образом, данный алгоритм действует подобно стрелке часов, пробегая по циклу страницы в кэше до первой, содержащей нулевой бит.



CLOCK

CLOCK устраняет первый недостаток LRU, остальные, очевидно, остаются.



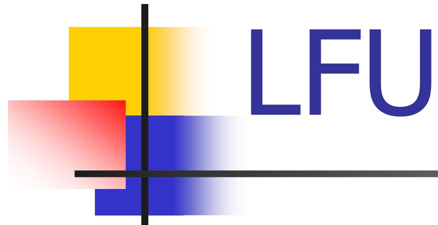
LFU

Алгоритм LFU – «least frequently used»

Каждой странице ставится в соответствие счетчик (Reference counter, далее RC);

При попадании страницы в кэш ее RC устанавливается равным 1.

При каждом обращении к станице в кэше, ее RC увеличивается на единицу.

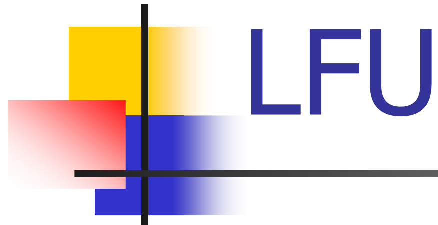


LFU

При необходимости выталкивания страницы из кэша, выталкивается страница с наименьшим RC.

Очевидно, LFU обладает множеством существенных недостатков:

- много обращений к странице за короткое время => большой RC => зависание страниц в кэше.

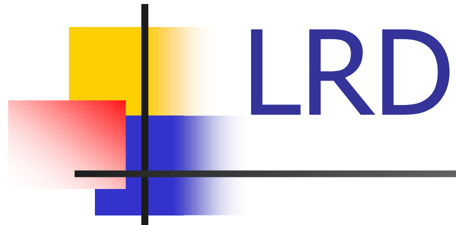


LFU

- алгоритм вообще не учитывает «возраст» страниц

Эти существенные недостатки делают невозможным применение LFU в современных БД.

Однако, существует множество гораздо более эффективных модификаций LFU.



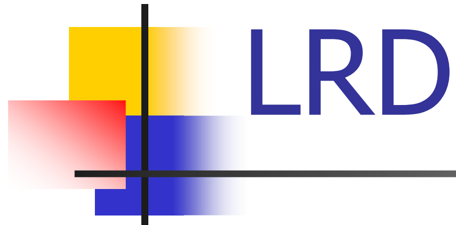
LRD

Класс алгоритмов LRD - least reference density – представляет собой усовершенствованную модель LFU

В добавок к RC, вводятся:

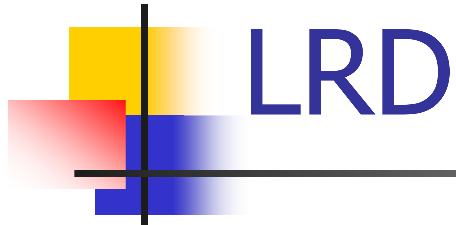
GRC – global reference counter – глобальный счетчик запросов страниц,

FC – First reference Counter – номер запроса, в момент которого страница была помещена в кэш.



LRD

Таким образом, разность вида GRC – FC(i) несет в себе информацию о «возрасте» i-ой страницы в кэше, что можно использовать при выталкивании.



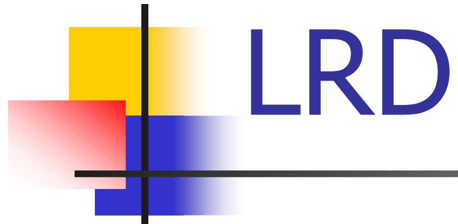
LRD

Когда нужно освободить место в кэше, для каждой страницы вычисляется ее RD – reference density по формуле

RD(i) = RC(i) / (GRC – FC(i)),

GRC – FC(i)>=1

Выталкивается страница с минимальным RD.



LRD

RC	FC	
2	20	
4	26	
1	40	
3	45	
3	5	
6	2	
1	37	
3	17	

GRC

50

RD(1) = 1/15

RD(2) = 1/6

RD(3) = 1/10

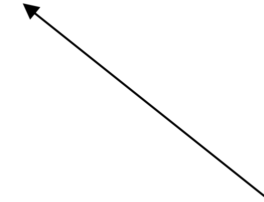
RD(4) = 3/5

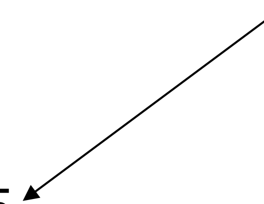
RD(5) = 1/15

RD(6) = 1/8

RD(7) = 1/13

RD(8) = 1/11





Замещаемая страница

GCLOCK

GCLOCK – generalized CLOCK – результат совмещения концепций алгоритмов CLOCK и LFU.

В GCLOCK, в отличие от CLOCK, вместо page reference bit используется RC.

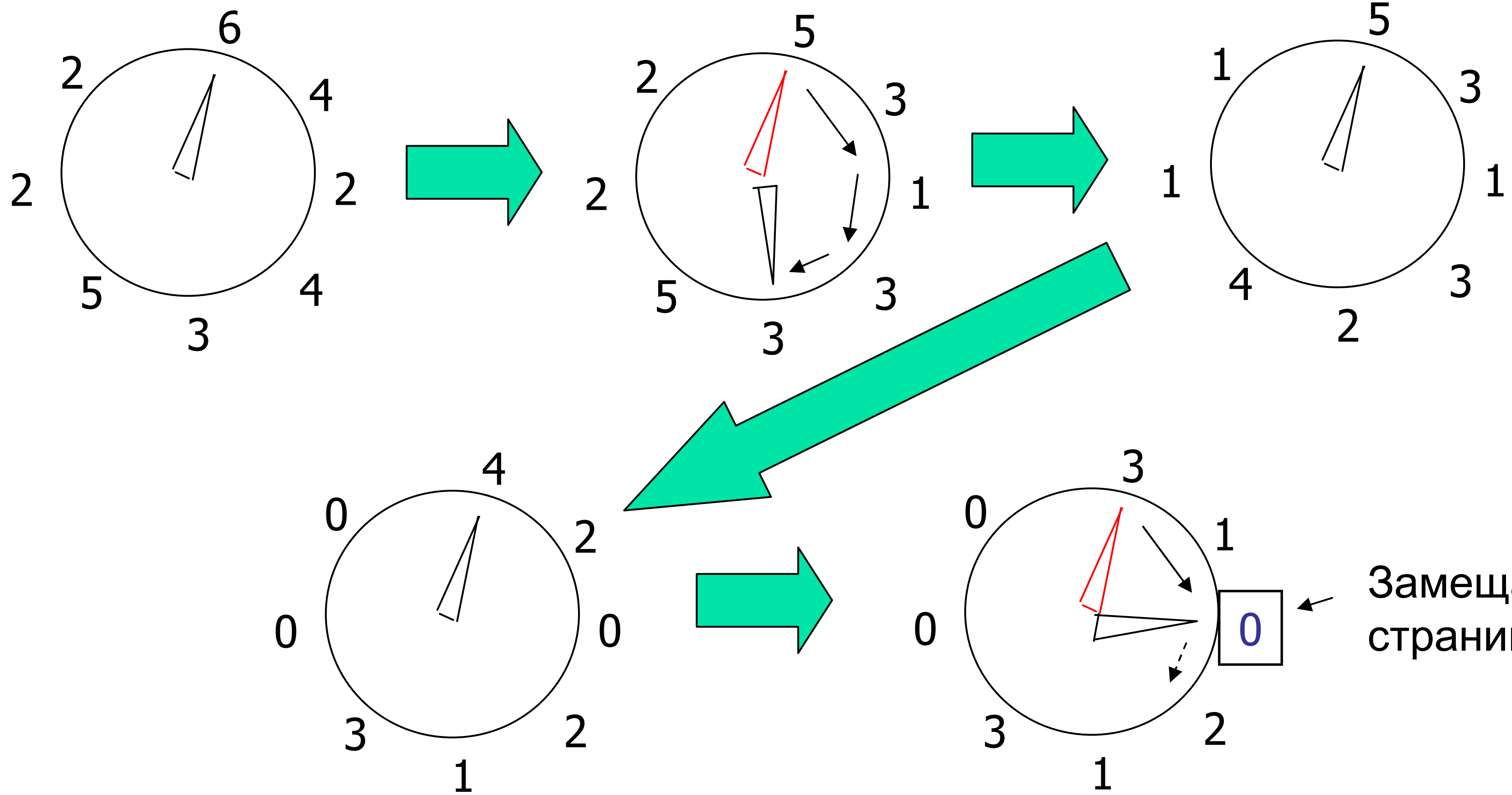
Принцип действия аналогичен CLOCK:

При помещении страницы в кэш RC устанавливается равным 1, при обращении к странице в кэше: ++RC.

GCLOCK

Однако, в отличие от оригинального CLOCK, при необходимости замещения страницы, для выбора оной может понадобится более одного полного обхода страниц по циклу, каждый раз уменьшая RC на 1, пока не встретится страница с RC = 0.

GCLOCK



GCLOCK

Недостатки:

- неконстантное время работы
- Алгоритм имеет предрасположенность к удалению «молодых» страниц из кэша.

(возможное решение – инициализация RC значением, большим 1).

Improved GCLOCK

Дополнительно усовершенствовать модель CLOCK можно, введя типизацию страниц в кэше.

Поставим в соответствие типам(i) страниц веса F(i) и R(i), где

F(i) – значение, которое получает RC страницы типа i при ее помещении в кэш.

R(i) – значение, на которое увеличивается RC при обращении к странице в кэше.

Improved GCLOCK

Варианты использования весов:

V1: first reference: RC(i) = Fi

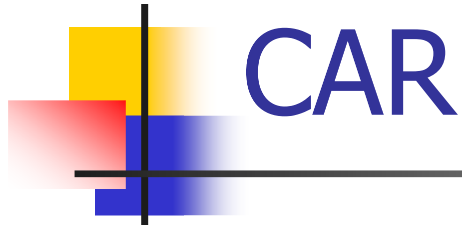
each rereference: RC(i) = RC(i) + Ri

V2: first reference: RC(i) = Fi

each rereference: RC(i) = Ri

При Fi = Ri = 1 для любого I

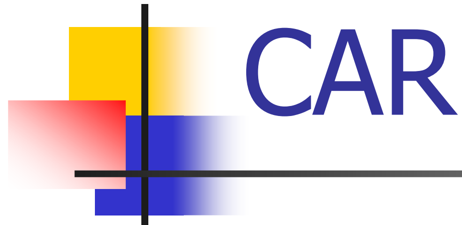
V2 => CLOCK



CAR

Вернемся к алгоритму CLOCK, а точнее к его недостаткам:

- цикличное обращение к страницам с периодом, превышающим размер кэша.
- и, как следствие, плохо работающие scan-операции.



CAR

В алгоритме CAR – CLOCK with Adaptive Replacement – данные недостатки отсутствуют.

Суть алгоритма:

Пусть размер кэша составляет **c** страниц. CAR работает с 4 списками станиц: T1, T2, B1, B2.

T1 и T2 содержат списки страниц, содержащиеся в кэше,

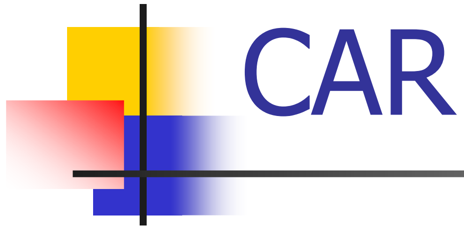
B1 и B2 содержат «историю» страниц – список страниц, побывавших в кэше относительно недавно.

CAR

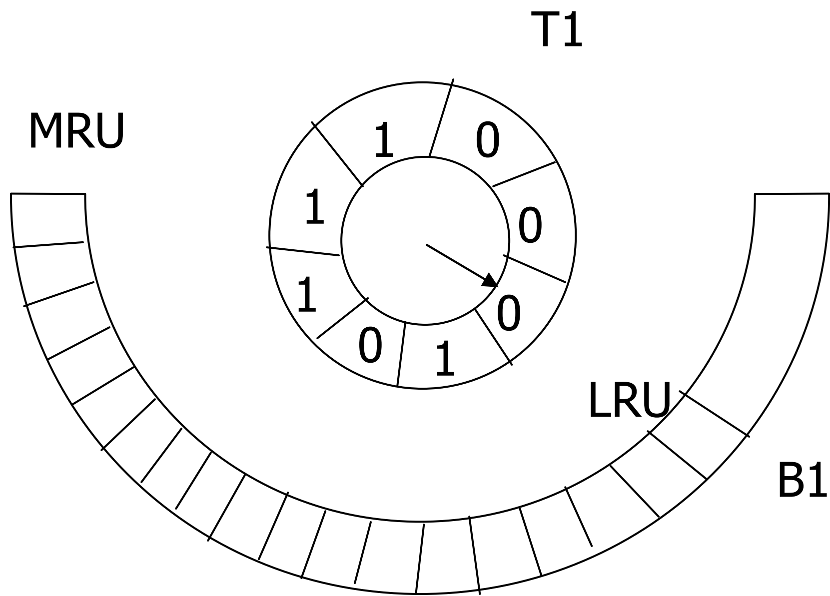
T2 и B2 содержат страницы, к которым обращались более одного раза(страницы **«долгосрочной полезности»**)

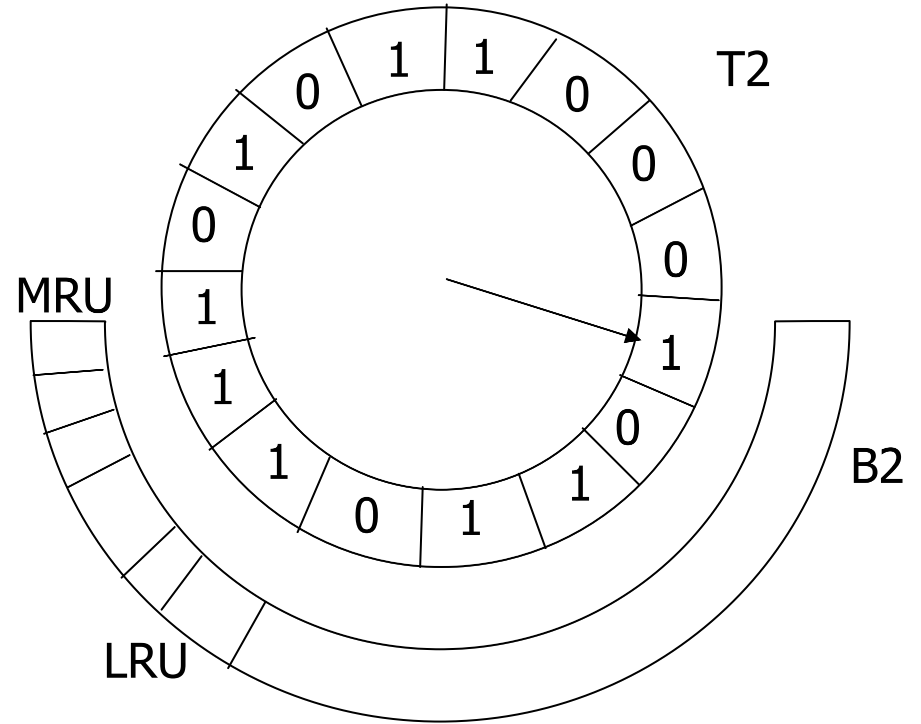
B1 – страницы, к которым обращались ровно один раз(страницы **«краткосрочной полезности»**)

T1 может содержать и те и другие.
T1, T2 – CLOCK;   B1,B2 - LRU



CAR





CAR

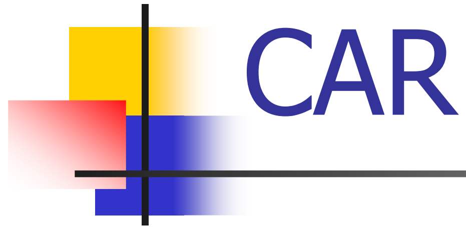
Для списков выполнены следующие условия:

1) $0 <= |T1| + |T2| <= c$
2) $0 <= |T1| + |B1| <= c$
3) $0 <= |T2| + |B2| <= 2c$
4) $0 <= |T1| + |T2| + |B1| + |B2| <= 2c$
5) If $(|T1| + |T2| < c)$ then $B1 \cup B2$ – empty
6) If $(|T1| + |T2| + |B1| + |B2| >= c)$ then $|T1| + |T2| = c$
7) Как только кэш заполнен, он остается заполненным всегда.

CAR

Для T1 вводится параметр p – число, характеризующее тот размер списка T1, к которому мы стремимся в данный момент.

p будет играть роль при выборе из какого списка (T1 и T2) выбрасывать страницу.



CAR

Пример замещения:

$C = 25,\ p = 7$

$p = 7 \Rightarrow$ работаем с T1

T1

MRU

LRU

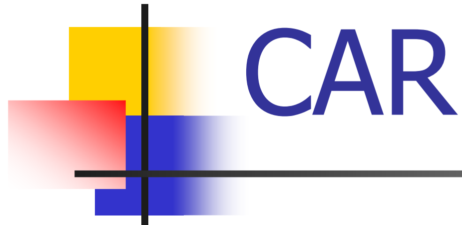
B1

T2

MRU

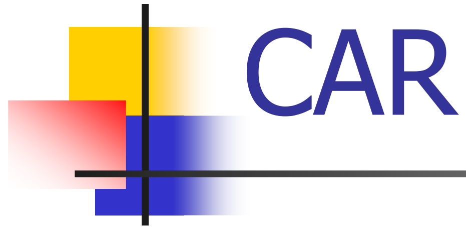
LRU

B2



CAR

Таким образом, в случае обнаружения требуемой страницы x среди страниц в «истории», происходит увеличение размера кэша под соответствующий тип страниц («краткосрочной» (x in B1) или «долгосрочной» (x in B2) полезности).



CAR

New pages

C A C H E

T1

Clock in T1 header page has ref.bit = 1

T2

T1 cache miss

B1 History hit

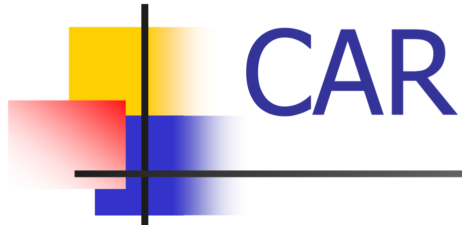
T2 cache miss

B2 History hit

B1

B2

H I S T O R Y



CAR

- Представленная версия алгоритма рассчитана на single-threaded work. Конечная версия – multi-threaded, достигается путем сериализации cache misses (требуется для обеспечения целостности списков). Cache hits не сериализуются.

ARC

ARC – Adaptive Replacement Cache – алгоритм, использующий в работе принципы, схожие с CAR, за исключением того, что работа со списками T1 и T2 ведется не посредством CLOCK, а через LRU. Следствием последнего является заимствование ARC некоторых недостатков недостатков LRU.

CART

- CART – CAR with Temporal filtering
- Проблема CAR – «correlated references» на страницу, которые часто не являются гарантией «долгосрочной полезности» страницы. В результате, эти страницы загрязняли кэш.

CART

Аналогично CAR, CART работает с 4 списками: T1,T2 (CLOCK, битовый флаг) – в кэше, B1 и B2 (LRU)– «история». Дополнительно, T1 и T2 теперь имеют «бит фильтра», показывающего, имеет ли страница «долгосрочную полезность»(L) или «короткосрочную»(S).

Также подверглись небольшим изменениям условия 2 и 3 CAR. Для CART они выглядят так:

2) $0 <= |T2| + |B2| <= c$

3) $0 <= |T1| + |B1| <= 2c$

CART

C A C H E

L(1) (q ↗)

target size: p

S(1) T1 T2

L(0,1)(q ↘)

S->L (p↗)

(p ↘)

S(0)

L(0)

B1

target size: q

B2

H I S T O R Y

CART

- Таким образом, T1 играет роль списка «наиболее используемых страниц», T2 хранит в себе страницы «долгосрочной полезности».
- Для T1 и B1 определены «запланированные» размеры p и q соответственно.

MQ

MQ – multiple queue – алгоритм, использующий несколько LRU очередей. Обозначим эти очереди Q0, Q1, ... , Qn. Также используется буфер «истории» - Qout (FIFO).

Каждая страница имеет свой RC – reference counter, который также хранится и для страниц, находящихся в буфере «истории».

MQ

Рассмотрим алгоритм MQ.

Пусть x – страница – цель запроса.

Если x при запросе находится в кэше (в списке с номером i), производятся следующие действия:

1) RC(x) увеличивается на единицу
2) Страница удаляется из списка Qi
3) Страница помещается в конец списка с номером k >= i, где k = F(RC(x)). Функция F является настраиваемым параметром системы(очевидно, должна быть возрастающей). Как вариант, используется log2(x) (с округлением).

MQ

При такой системе сохраняется недостаток, аналогичный недостатку алгоритма LFU – страницы с очень высоким RC могут перестать использоваться, но при этом останутся «висеть» в кэше.

MQ

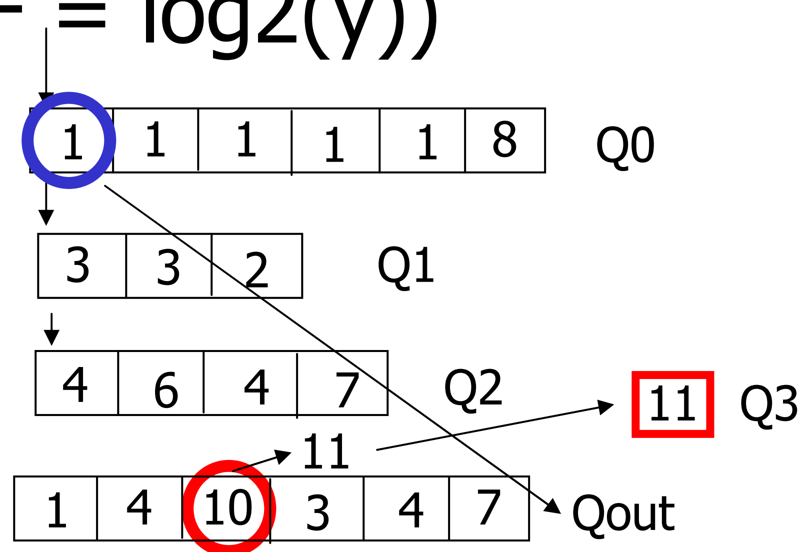
Для борьбы с этим явлением в MQ введено понятие expiredTime, связанное с каждой страницей в кэше. При помещении страницы в кэш, expiredTime = currentTime + lifeTime (параметр системы). При каждом запросе для всех страниц проверяется currentTime > expiredTime. При выполнении условия, страница перемещается в хвост нижележащего списка.

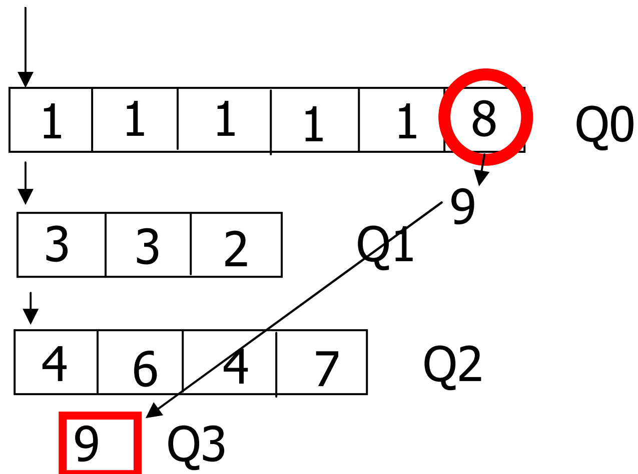
MQ

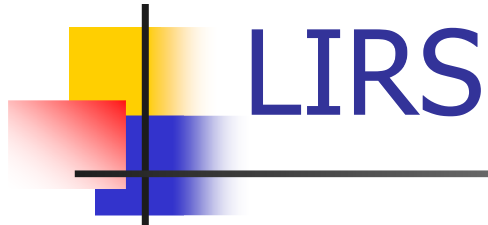
MQ устраняет недостатки LRU, связанные с устойчивостью к сканированию и поддержки страниц «долгосрочной» полезности.

MQ

Примеры: (F = log2(y))







LIRS

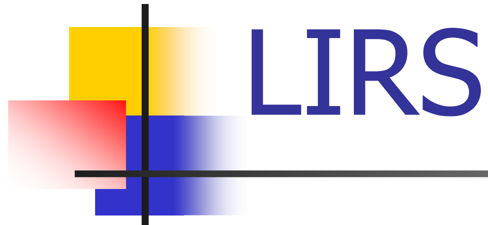
LIRS - Low Inter-reference Recency Set

Суть:

X – некоторая страница

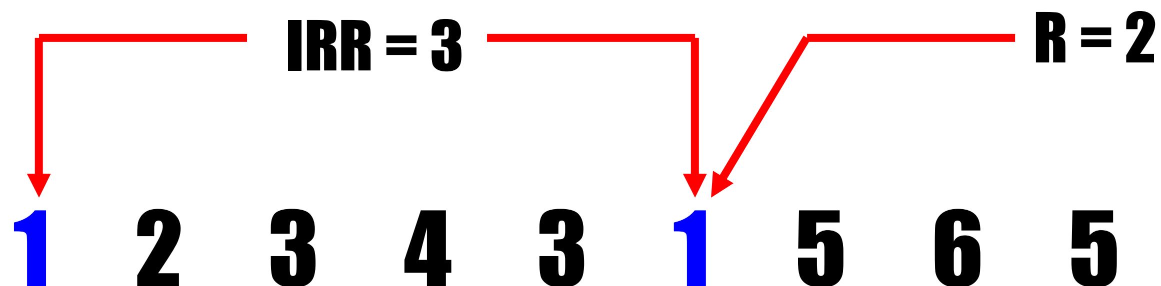
Для каждого блока вводятся понятия:

IRR(x) – количество уникальных обращений к блокам между двумя последними обращениями к x. Может быть неопределено.



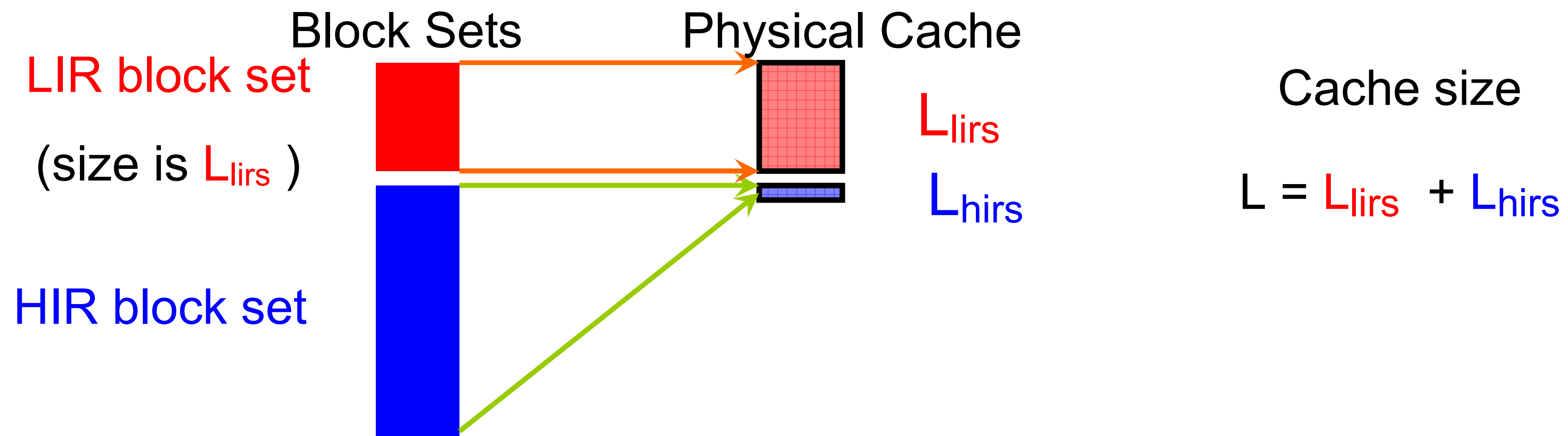
LIRS

R(x) – количество уникальных обращений к блокам с момента последнего обращения к x.



LIRS

Основная идея алгоритма: если IRR страницы высок, он вероятнее всего, будет высок и далее.



Low IRR (LIR) block and High IRR (HIR) block

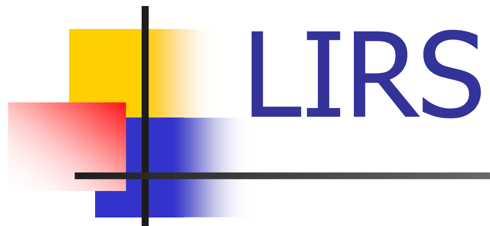
LIRS

V time / Blocks	1	2	3	4	5	6	7	8	9	10	R	IRR
A	X					X		X			1	1
B			X		X						3	1
C				X							4	inf
D		X					X			X	0	3
E									X		1	Inf

The resident HIR block (E) is replaced !

LIRS

V time / Blocks	1	2	3	4	5	6	7	8	9	10	R	IRR
A	X					X		X			2	1
B			X		X						3	1
C				X							4	inf
D		X					X			X	0	2
E									X		1	Inf



LIRS

V time / Blocks	1	2	3	4	5	6	7	8	9	10	R	IRR
A	X					X		X			2	1
B			X		X						3	1
C				X							4	inf
D		X					X			X	0	2
E									X		1	Inf

E is replaced, D enters LIR set

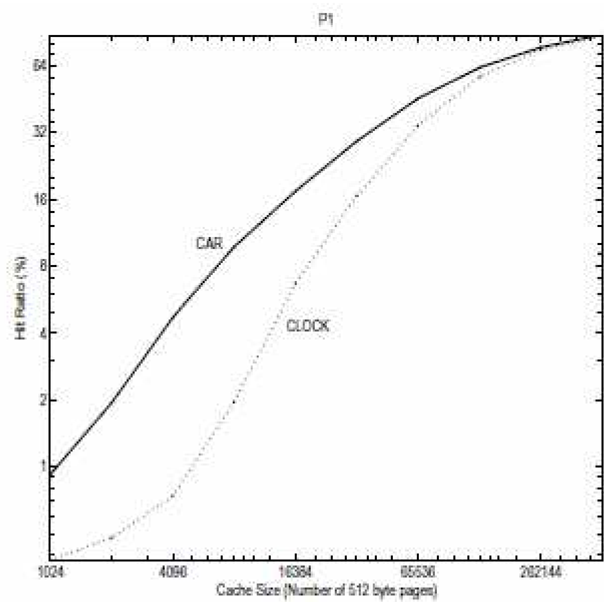
LIRS

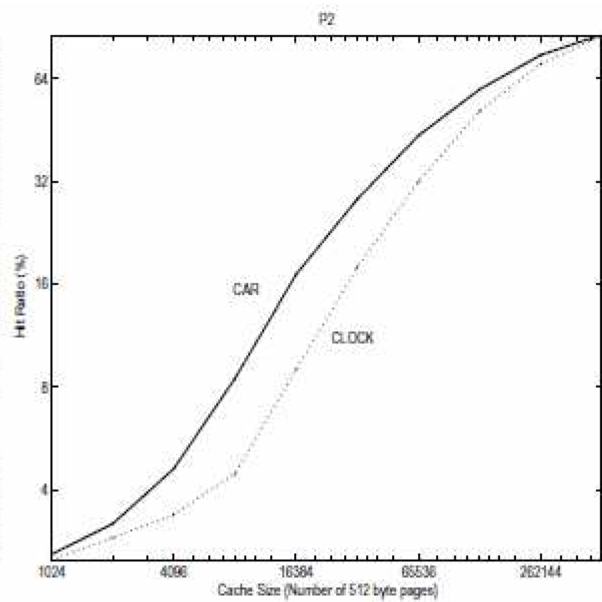
V time / Blocks	1	2	3	4	5	6	7	8	9	10	R	IRR
A	X					X		X			2	1
B			X		X						4	1
C				X						X	0	4
D		X					X				3	3
E									X		1	Inf

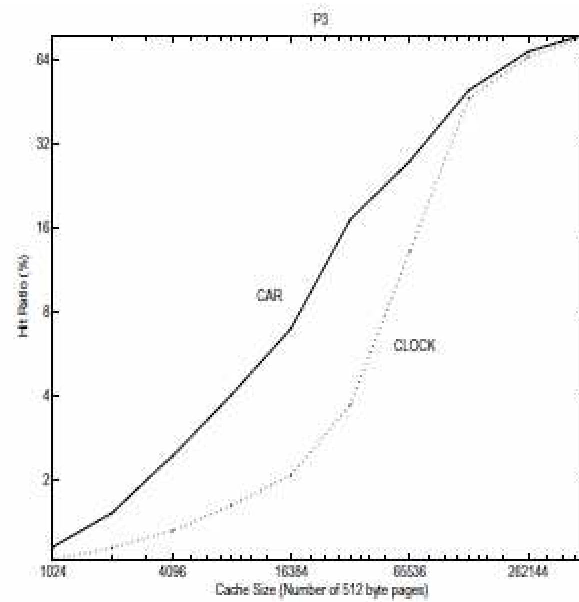
E is replaced, C can not enter LIR set

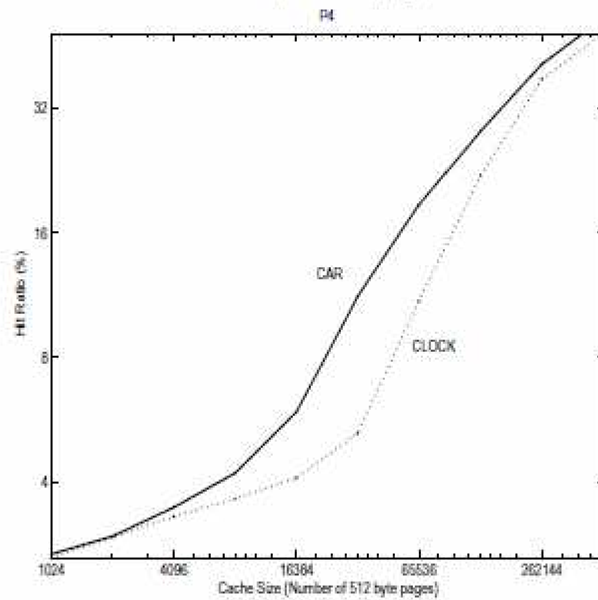
Experiments

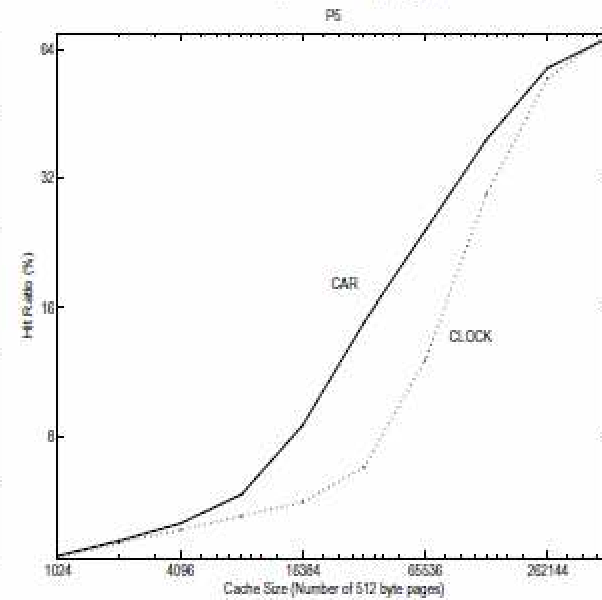
Trace Name	Number of Requests	Unique Pages
P1	32055473	2311485
P2	12729495	913347
P3	3912296	762543
P4	19776090	5146832
P5	22937097	3403835
P6	12672123	773770
P7	14521148	1619941
P8	42243785	977545
P9	10533489	1369543
P10	33400528	5679543
P11	141528425	4579339
P12	13208930	3153310
P13	15629738	2497353
P14	114990968	13814927
ConCat	490139585	47003313
Merge(P)	490139585	47003313
DS1	43704979	10516352
SPC1	41351279	6050363
S1	3995316	1309698
S2	17253074	1693344
S3	16407702	1689882
Merge (S)	37656092	4692924

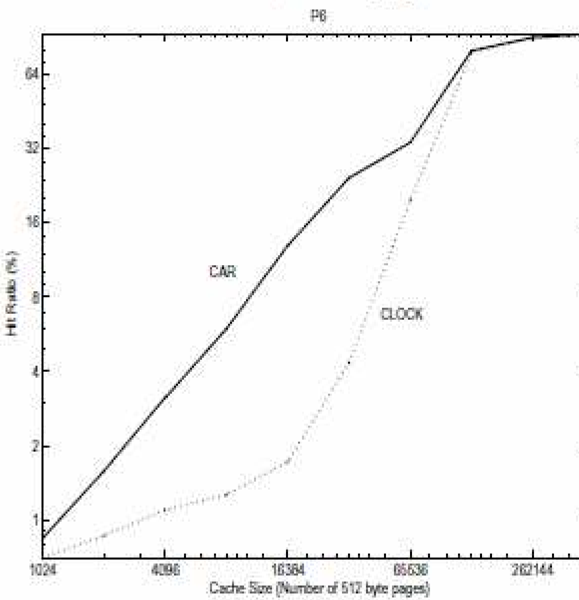

P1
CAR
CLOCK
Hit Ratio (%)
Cache Size (Number of 512 byte pages)


P2
CAR
CLOCK
Hit Ratio (%)
Cache Size (Number of 512 byte pages)


P3
CAR
CLOCK
Hit Ratio (%)
Cache Size (Number of 512 byte pages)


P4
CAR
CLOCK
Hit Ratio (%)
Cache Size (Number of 512 byte pages)


P5
CAR
CLOCK
Hit Ratio (%)
Cache Size (Number of 512 byte pages)

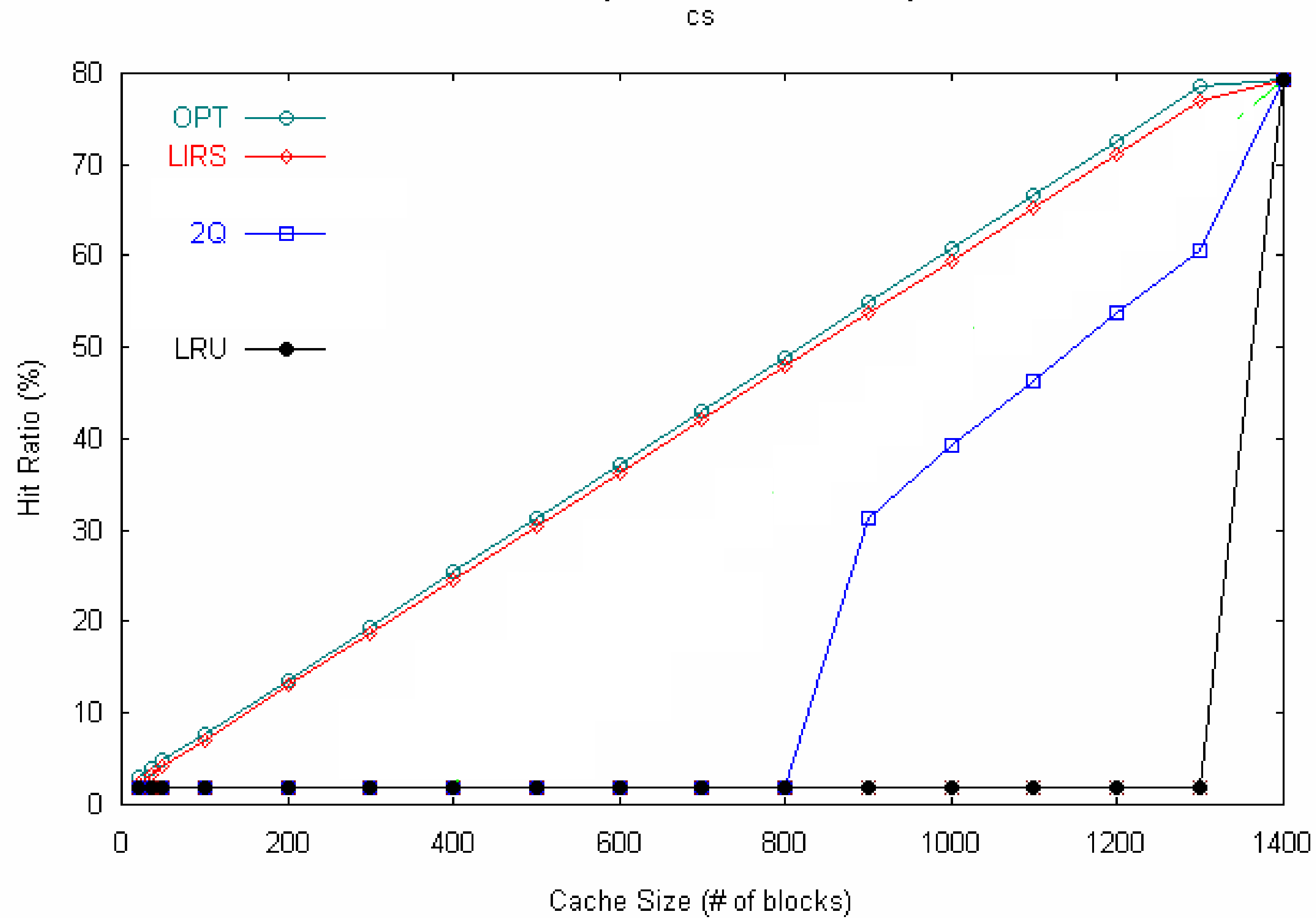

P6
CAR
CLOCK
Hit Ratio (%)
Cache Size (Number of 512 byte pages)

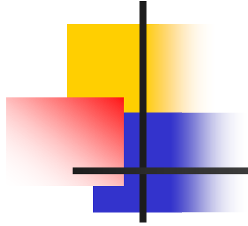
Experiments

Workload	c (pages)	space (MB)	LRU	CLOCK	ARC	CAR	CART
P1	32768	16	16.55	17.34	28.26	29.17	29.83
P2	32768	16	18.47	17.91	27.38	28.38	28.63
P3	32768	16	3.57	3.74	17.12	17.21	17.54
P4	32768	16	5.24	5.25	11.24	11.22	9.25
P5	32768	16	6.73	6.78	14.27	14.78	14.77
P6	32768	16	4.24	4.36	23.84	24.34	24.53
P7	32768	16	3.45	3.62	13.77	13.86	14.79
P8	32768	16	17.18	17.99	27.51	28.21	28.97
P9	32768	16	8.28	8.48	19.73	20.09	20.75
P10	32768	16	2.48	3.02	9.46	9.63	9.71
P11	32768	16	20.92	21.51	26.48	26.99	27.26
P12	32768	16	8.93	9.18	15.94	16.25	16.41
P13	32768	16	7.83	8.26	16.60	17.09	17.74
P14	32768	16	15.73	15.98	20.52	20.59	20.63
ConCat	32768	16	14.38	14.79	21.67	22.06	22.24
Merge(P)	262144	128	38.05	38.60	39.91	39.90	40.12
DS1	2097152	1024	11.65	11.86	22.52	25.31	21.12
SPC1	1048576	4096	9.19	9.31	20.00	20.00	21.91
S1	524288	2048	23.71	25.26	33.43	33.42	33.62
S2	524288	2048	25.91	27.84	40.68	41.86	42.10
S3	524288	2048	25.26	27.13	40.44	41.67	41.87
Merge(S)	1048576	4096	27.62	29.04	40.44	41.01	41.83

stats

Для циклически-повторяющихся запросов





Конец

CAR

Алгоритм замещения страницы в кэше:

(p – “запланированный” размер T1)

Если |T1| >= p, тогда, используя алгоритм CLOCK, выбираем замещаемую страницу из T1, причем все встреченные нами страницы с битовым флагом = 1 помещаем в хвост списка T2, устанавливая битовый флаг в 0. Выбранную для замещения страницу помещаем в B1, отмечая ее как MRU.

Если же |T1| < p, то замещаемую страницу выбираем из T2 обычным CLOCK. Выбранную страницу помещаем в B2 как MRU.

CAR

Полный алгоритм работы CAR:

Обозначим запрашиваемую страницу за x.

Если (x in T1 ᴜ T2){

/*cache hit*/ битовый флаг x = 1;}

Иначе {/*cache miss*/

Если (|T1| + |T2| = c) {/*cache full*/ replace();

Если (x in B1 ᴜ B2) и (|T1| + |B1| = c){ /* необходимая нам страница находится в «истории» */ удаляем LRU страницу из B1.}

Если (x not in B1 ᴜ B2) и (|T1| + |B1| + |T2| + |B2| = 2c) {/* необходимой нам страницы в «истории» нет */ удаляем LRU страницу из B2.}

}

CAR

if (x not in B1 ∪ B2) { вставляем страницу x в хвост T1 с битовым флагом 0}

else if(x in B1){ /*Adapt:*/ p=min(p + max(1,|B2|/|B1|) ,c); перемещаем x из B1 в хвост T2(битовый флаг = 0)}

else{ /*x in B2. Adapt:*/p=max(p –max(1,|B1|/|B2|), 0); перемещаем x в конец T2(битовый флаг = 0)}

}

CART

Принцип действия «filter bit»:

1) Каждая страница в T2 и B2 помечается как L.
2) Каждая страница в B1 помечается как S
3) Страницы в T1 могут быть как L, так и S
4) «Верхняя» страница T1 может быть замещена только если ее битовый флаг 0 и filter bit = S

CART

- если filter bit «верхней» страницы T1 = L, она помещается в хвост T2, ее битовый флаг = 0.
- !!! Если битовый флаг header page T1 = 1, она помещается в хвост T1(битовый флаг = 0). Если битовый флаг header page T2 = 1, то она также(!) помещается в хвост T1(битовый флаг = 0).

(далее x – requested page)

- if x not in (T1 ∪ T2 ∪ B1 ∪ B2) => filter bit (x) = S
- if (x in T1 and |T1|>=|B1|) => filter bit(x) = L
- if(x in B1) => filter bit(x) (old = S new = L)

Алгоритм замещения CART

```
 - while(reference bit(head page T2) == 1)
{
        move T2 head page to T1 tail(ref bit = 0));
        ++q (if possible);
} while(filter bit(head page T1)==L
                OR ref bit(head page T1) == 1)
{
        if(filter bit(head page T1) == L)
        {
            move T1 head page to T2 tail(ref bit = 0);
            --q (if possible);
        }
        else
        {
            move T1 head page to T1 tail(ref bit = 0)
            if(|T1| >= min(p+1,|B1|) AND filter bit(moved page) == S)
                filter bit(moved page) = L;
        }
}
If( |T1|  >= p) move T1 head page to MRU pos. in B1
Else  move T2 head page to MRU pos. in B2
```

CART

History replacement:

```
If(|B1| + |B2|  = c + 1)
{
   if(|B1| > q)
        remove bottom page from B1
   else
        remove bottom page from B2
}
```