

数据存储管理技术的更新换代

张晓东

美国俄亥俄州立大学

The Ohio State University

Numbers Everyone Should Know (Jeff Dean, Google)²

- L1 cache reference: 0.5 ns
- Branch mis-predict: 5 ns
- L2 cache reference: 7 ns
- Mutex lock/unlock: 25 ns
- Main memory reference: 100 ns
- Compress 1K Bytes with Zippy: 3000 ns
- Send 2K Bytes over 1 GBPS network: 20000 ns
- Read 1 MB sequentially from memory: 250000 ns
- Round trip within data center: 500000 ns
- Disk seek: 1000000 ns
- Read 1MB sequentially from disk: 2000000 ns
- Send one packet from CA to Europe: 15000000 ns

Replacement Algorithms in Data Storage Management

- **A replacement algorithm decides**
 - Which data entry to be evicted when the data storage is full.
 - *Objective*: keep to-be-reused data, replace ones not to-be-reused
 - Making a critical decision: a miss means an increasingly long delay
- **Widely used in all memory-capable digital systems**
 - Small buffers: cell phone, Web browsers, e-mail boxes ...
 - Large buffers: virtual memory, I/O buffer, databases ...
- **A simple concept, but hard to optimize**
 - More than 40 years tireless algorithmic and system efforts
 - LRU-like algorithms/implementations have serious limitations.

Least Recent Used (LRU) Replacement

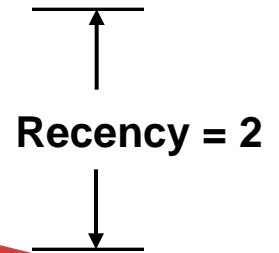
- LRU is most commonly used replacement for data management.
- Blocks are ordered by an **LRU order** (from bottom to top)
- Blocks enter from the top (MRU), and leave from bottom (LRU)

The stack is long, the bottom is the only exit.

- **Recency** – the distance from a block to the top of the LRU stack

- **Upon a hit** Recency of Block 2 is its distance to the top of stack

Move block 2 to the top of stack



Upon a Hit to block 2

LRU stack

Least Recent Used (LRU) Replacement

- LRU is most commonly used replacement for data management.
- Blocks are ordered by an **LRU order** (from bottom to top)
- Blocks enter from the top, and leave from

The stack is long, the bottom is the only exit.

Load block 6
from disk

Upon a Miss to
block 6

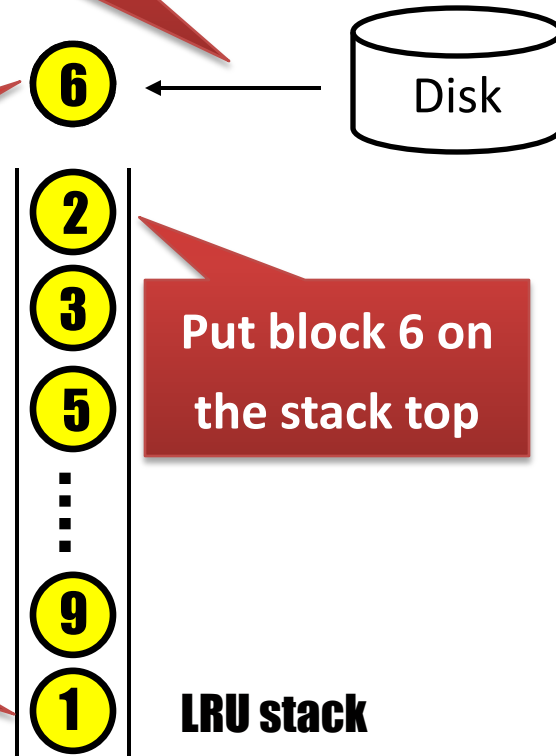
- **Recency** – the distance from a block to the top of the LRU stack

Put block 6 on
the stack top

- **Upon a hit** – move block to the top

Replacement – the block 1 at the
stack bottom is evicted

- **Upon a miss**



LRU is a Classical Problem in Theory and Systems

- **First LRU paper**
 - L. Belady, IBM System Journal, 1966
- **Analysis of LRU algorithms**
 - Aho, Denning & Ulman, JACM, 1971
 - **Rivest**, CACM, 1976
 - Sleator & **Tarjan**, CACM, 1985
 - **Knuth**, J. Algorithm, 1985
 - **Karp**, et. al, J. Algorithms, 1991
- **Many papers in systems and databases**
 - ASPLOS, ISCA, SIGMETRICS, SIGMOD, VLDB, USENIX...

The Problem of LRU:

Inability to Deal with Certain Access Patterns

- **File Scanning**
 - One-time accessed data evict to-be-reused data (cache pollution)
 - A common data access pattern (50% data in NCAR accessed once)
 - LRU stack holds them until they reach to the bottom.
- **Loop-like accesses**
 - A loop size $k+1$ will miss k times for a LRU stack of k
- **Access with different frequencies (mixed workloads)**
 - Frequently accessed data can be replaced by infrequent ones

Why Flawed LRU is so Powerful in Practice

- **What is the major flaw?**
 - The assumption of “recent used will be reused” is wrong
 - This prediction is based on a simple metrics of “**recency**”
 - **Some are cached too long, some are evicted too early.**
- **Why it is so widely used?**
 - Works well for data accesses following LRU assumption
 - A simple data structure to implement

Challenges of Addressing the LRU Problem

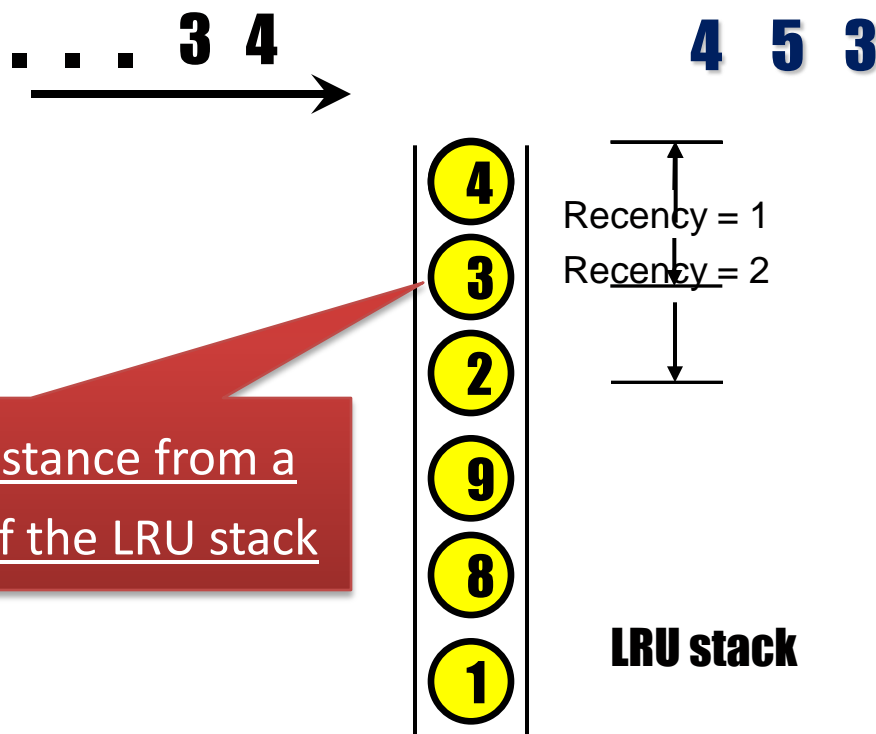
- **Two types of Efforts have been made**
 - Detect **specific access patterns**: handle it case by case
 - Learn insights into accesses with **complex algorithms**
 - Most published papers could not be turned into reality
- **Two Critical Goals**
 - Fundamentally address the LRU problem
 - Retain LRU merits: low overhead and its assumption
- **The goals are achieved by a set of three papers**
 - The **LIRS** algorithm (SIGMETRICS'02)
 - **Clock-pro**: a system implementation (USENIX'05)
 - **BP-Wrapper**: lock-contention free assurance (ICDE'09)

Outline

- **The LIRS Algorithm**
 - How the LRU problem is fundamentally addressed
 - How a data structure with low complexity is built
- **Clock-pro**
 - Turn LIRS algorithm into system reality
- **BP-Wrapper**
 - free lock contention so that LIRS and others can be implemented without approximation
- **What would we do for multicore processors?**
- **Research impact in daily computing operations**

Recency vs Reuse Distance

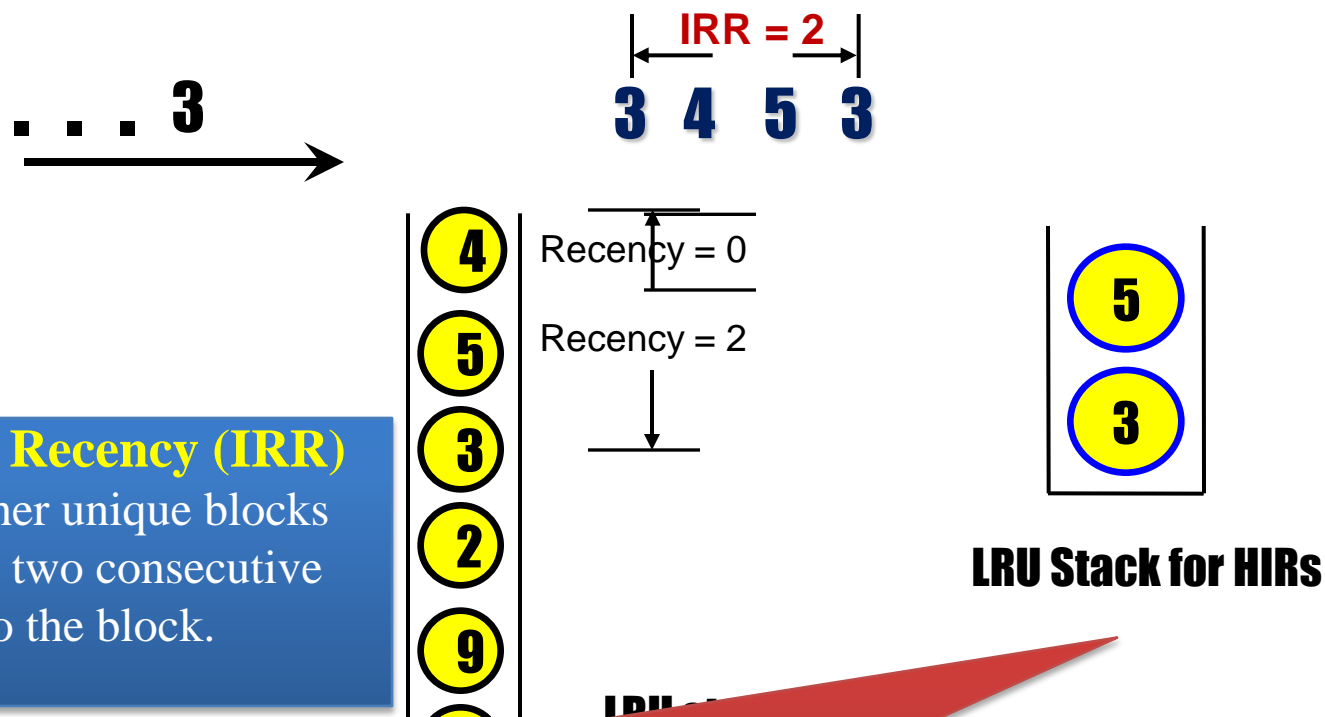
- **Recency** – the distance between last reference to the current time
- **Reuse Distance** (Inter reference recency) – the distance between two consecutive reference to the block (**deeper and more useful information**)



Recency is the distance from a block to the top of the LRU stack

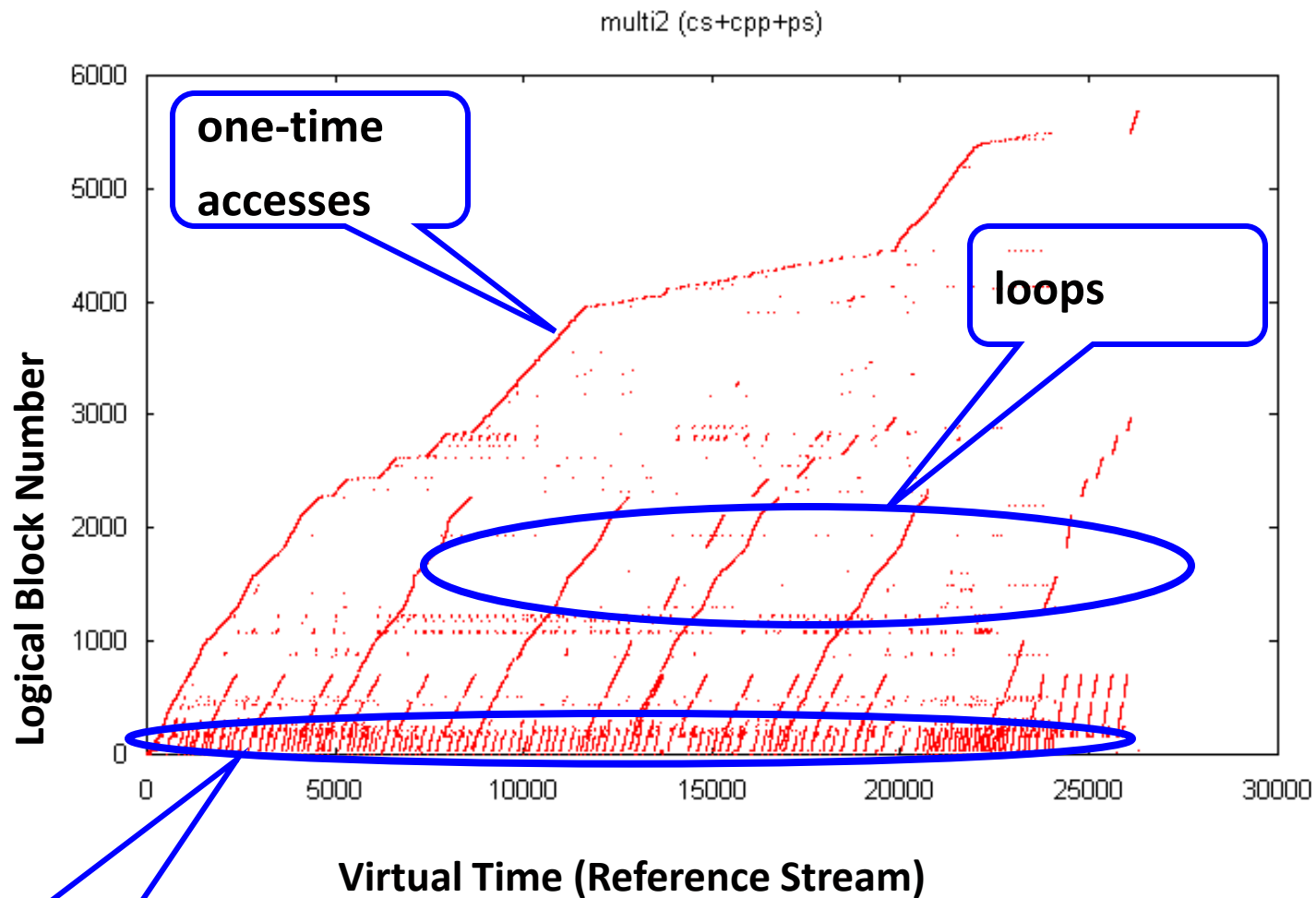
Recency vs Reuse Distance

- **Recency** – the distance between last reference to the current time
- **Reuse Distance** (Inter reference recency) – the distance between two consecutive reference to the block (**deeper and more useful info**)

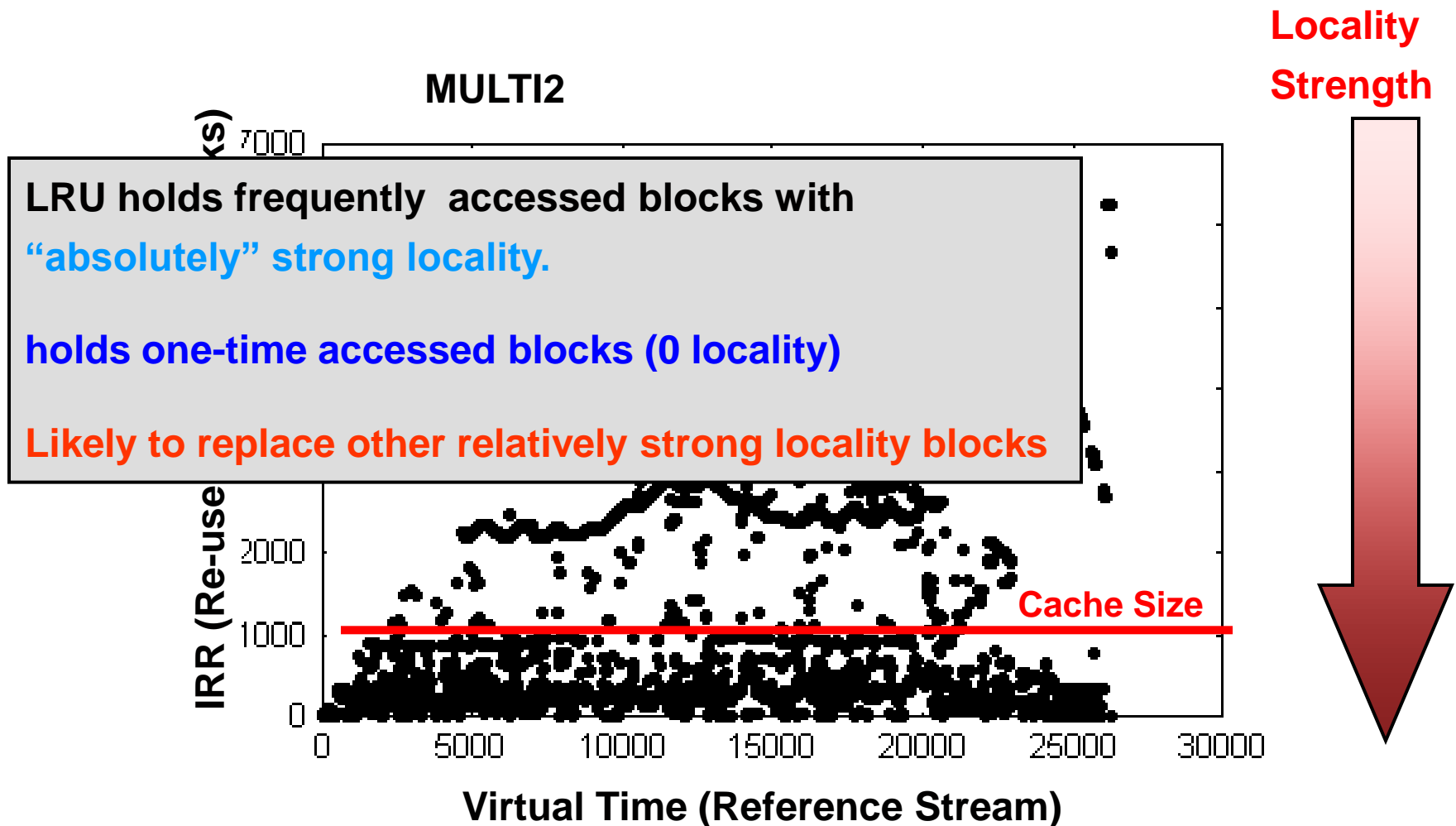


IRR is the recency of the block being accessed last time – need an extra stack to help, increasing complexity.

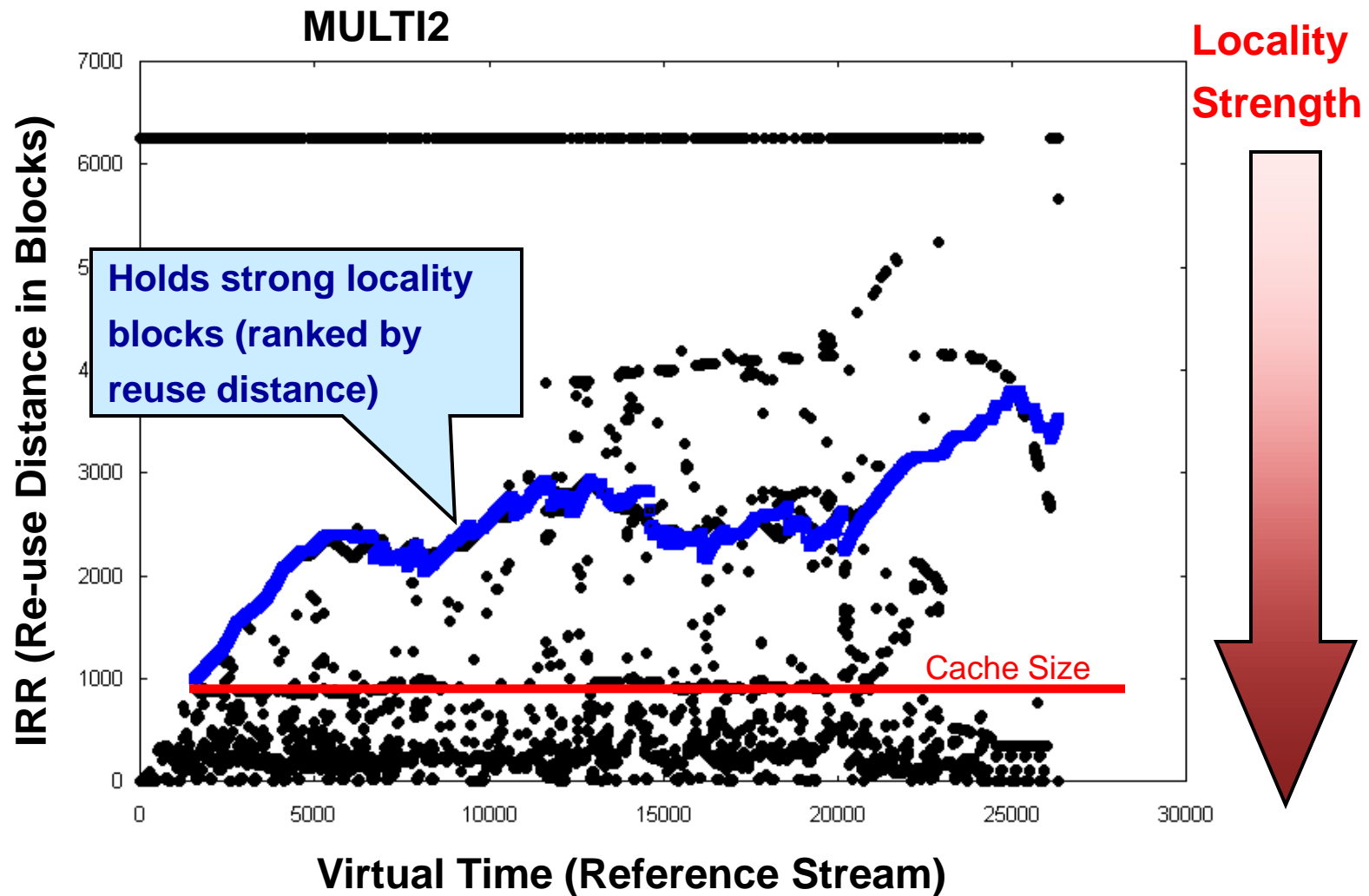
Diverse Locality Patterns on an Access Map



What Blocks does LRU Cache (measured by IRR)?



LIRS: Only Cache Blocks with Low Reuse Distances



Basic Ideas of LIRS (SIGMETRICS'02)

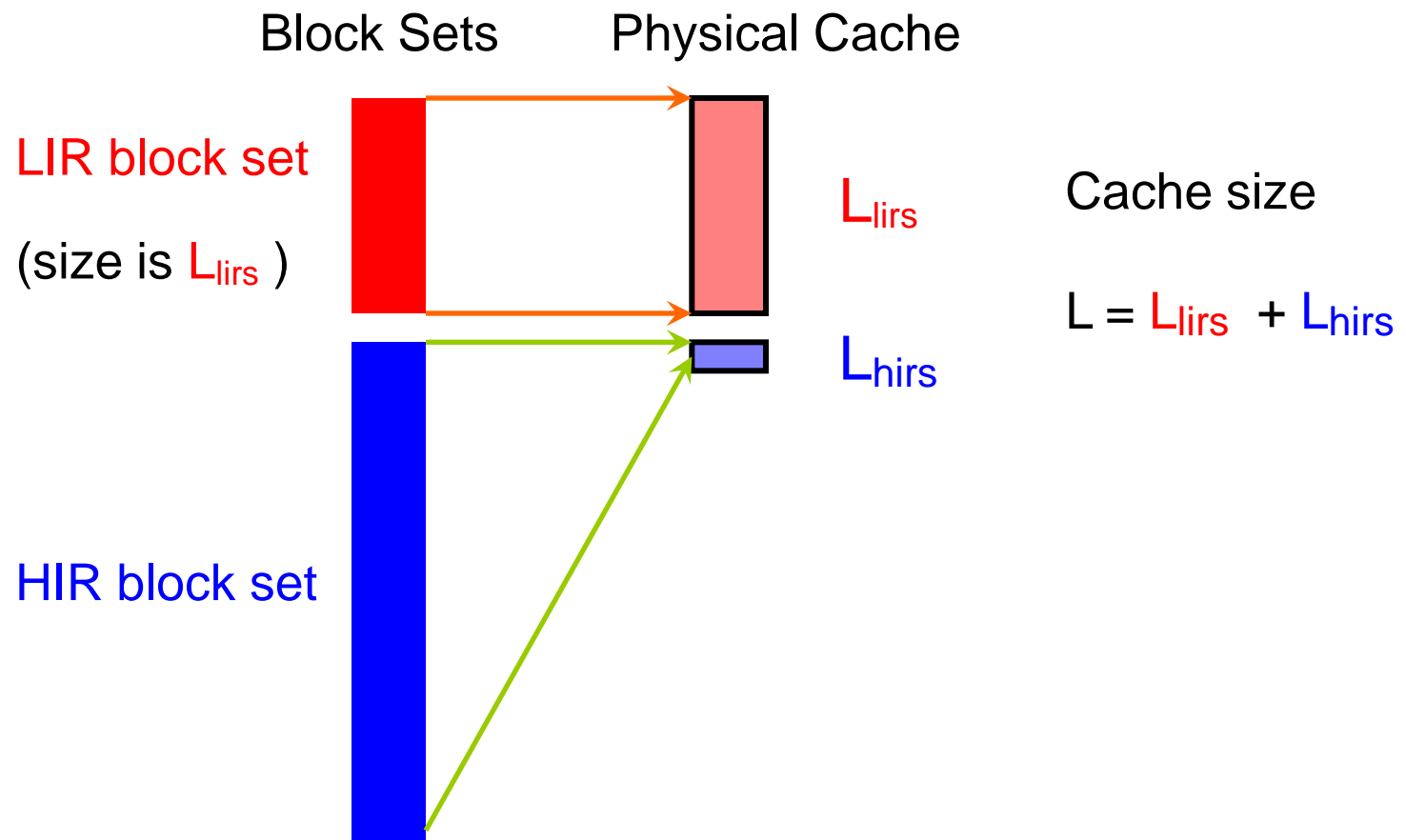
- **LIRS: Low Inter-Reference recency Set**
 - Low IRR blocks are kept in buffer cache
 - High IRR blocks are candidates for replacements
- **Two stacks are maintained**
 - A large LRU stack contains low IRR resident blocks
 - A small LRU stack contains high IRR blocks
 - The large stack also records resident/nonresident high IRR blocks
- **IRRs are measured by the two stacks**
 - After a hit to a resident high IRR block in small stack, the block becomes low IRR block and goes to the large stack if it can also be found in the large stack => low IRR , otherwise, top it locally
 - The low IRR block in the bottom stack will become a high IRR block and go to the small stack when the large stack is full.

Low Complexity of LIRS

- **Both recencies and IRRs are recorded in each stack**
 - The block in the bottom of LIRS has the maximum recency
 - A block is low IRR if it can be found in in both stacks
 - No explicit comparisons and measurements are needed
- **Complexity of LIRS = LRU = $O(1)$ although**
 - Additional object movements between two stacks
 - Pruning operations in stacks

Data Structure: Keep LIR Blocks in Cache

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



LIRS Operations

- Initialization: All the referenced blocks are given an LIR status until LIR block set is full.

We place resident HIR blocks in a small LRU Stack.

- Upon accessing an LIR block (a hit)
- Upon accessing a resident HIR block (a hit)
- Upon accessing a non-resident HIR block (a miss)



LIRS stack

- resident in cache
- LIR block
- HIR block

Cache size

$$L = 5$$

$$L_{\text{lir}} = 3$$

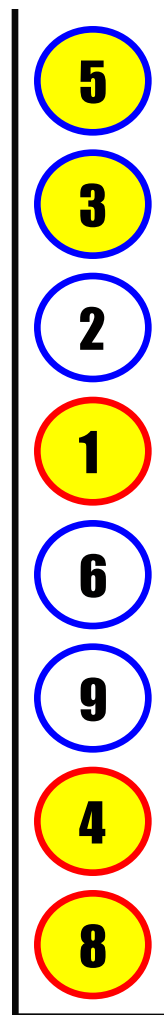

$$L_{\text{hir}} = 2$$



LRU Stack for HIRs

Access an LIR Block (a Hit)

... 5 9 7 5 3 8 4



● resident in cache

○ LIR block

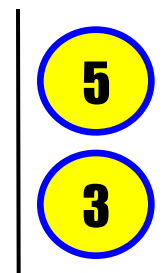
○ HIR block

Cache size

$L = 5$

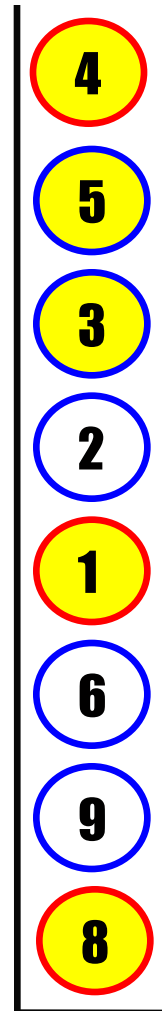
$L_{\text{lir}} = 3$

$L_{\text{hir}} = 2$



Access an LIR Block (a Hit)

... 5 9 7 5 3 8



● resident in cache

○ LIR block

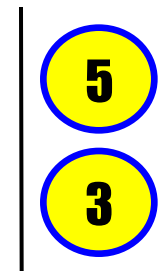
○ HIR block

Cache size

$L = 5$

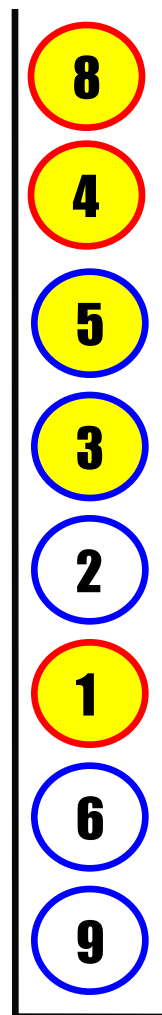

$L_{lir} = 3$

$L_{hir} = 2$



Access an LIR block (a Hit)

... 5 9 7 5 3 8



● resident in cache

○ LIR block

○ HIR block

Cache size

$$L = 5$$

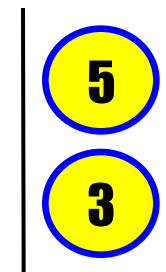
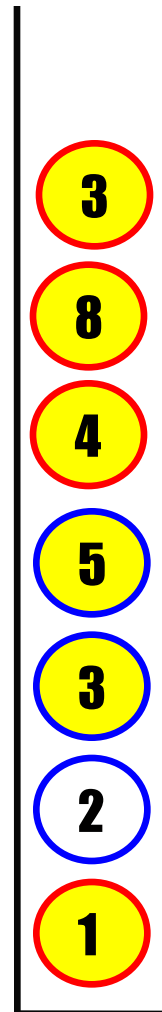
$$L_{\text{lir}} = 3$$

$$L_{\text{hir}} = 2$$



Access a Resident HIR Block (a Hit)

... 5 9 7 5 3



● resident in cache

○ LIR block

○ HIR block

Cache size

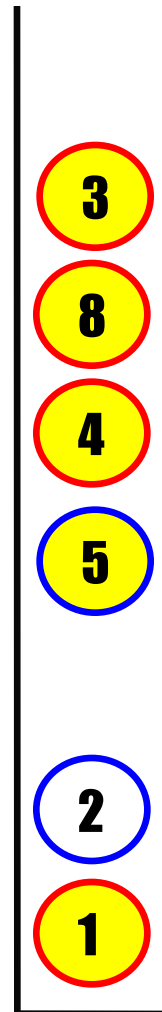
$L = 5$

$L_{lir} = 3$

$L_{hir} = 2$

Access a Resident HIR Block (a Hit)

... 5 9 7 5 3



● resident in cache

○ LIR block

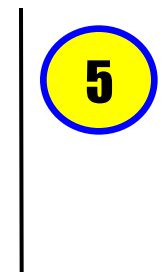
○ HIR block

Cache size

$L = 5$

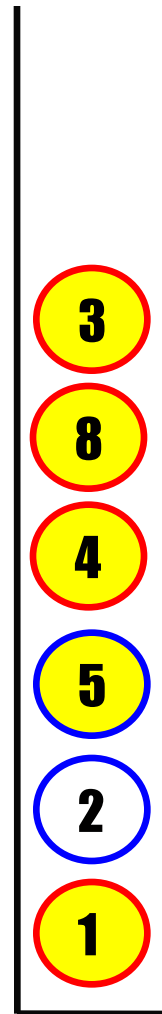
$L_{lir} = 3$

$L_{hir} = 2$



Access a Resident HIR Block (a Hit)

... 5 9 7 5 3



● resident in cache

○ LIR block

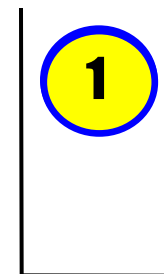
○ HIR block

Cache size

$L = 5$

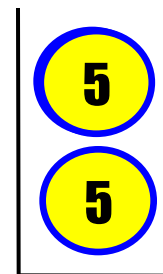
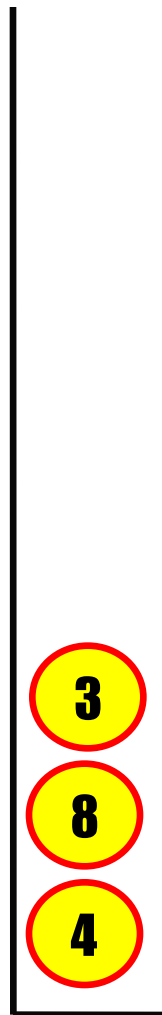
$L_{lir} = 3$

$L_{hir} = 2$



Access a Resident HIR Block (a Hit)

... 5 9 7 5



● resident in cache

○ LIR block

○ HIR block

Cache size

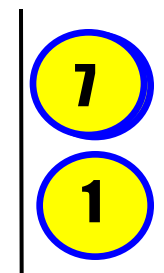
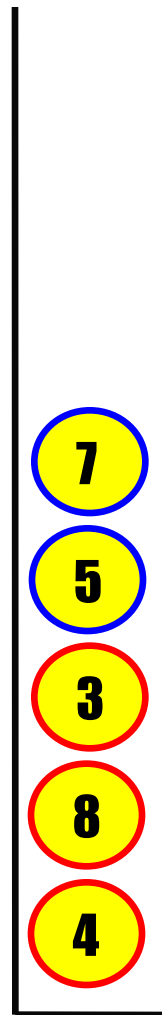
$L = 5$

$L_{lir} = 3$

$L_{hir} = 2$

Access a Non-Resident HIR block (a Miss)

... 5 9 7



● resident in cache

○ LIR block

○ HIR block

Cache size

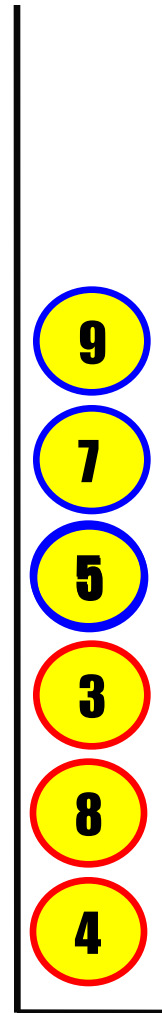
$L = 5$

$L_{lir} = 3$

$L_{hir} = 2$

Access a Non-Resident HIR block (a Miss)

... 5 9



● resident in cache

○ LIR block

○ HIR block

Cache size

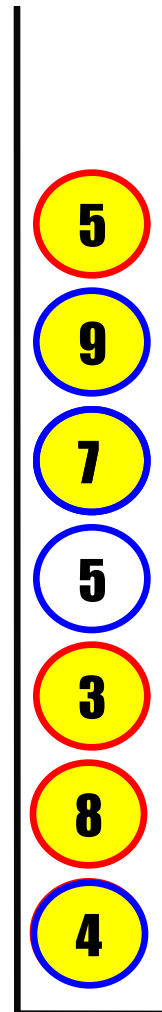
$L = 5$




$L_{\text{lir}} = 3$

$L_{\text{hir}} = 2$

Access a Non-Resident HIR block (a Miss)

... 5



-  resident in cache
 -  LIR block
 -  HIR block
- Cache size

$$L = 5$$

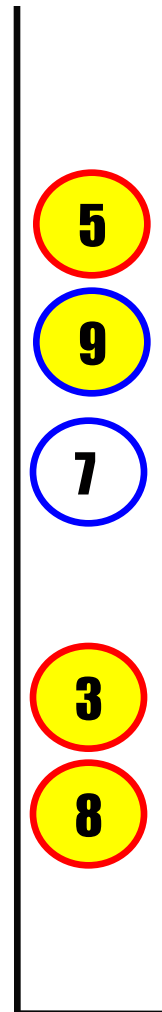
$$L_{\text{lir}} = 3$$

$$L_{\text{hir}} = 2$$



Access a Non-Resident HIR block (a Miss)

...



● resident in cache

○ LIR block

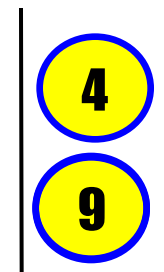
○ HIR block

Cache size

$L = 5$

$L_{lir} = 3$

$L_{hir} = 2$



How LIRS addresses the LRU problem

- **File scanning**: one-time access blocks will be replaced timely; (due to their high IRRs)
- **Loop-like accesses**: a section of loop data will be protected in low IRR stack; (misses only happen in the high IRR stack)
- **Accesses with distinct frequencies**: Frequently accessed blocks in short reuse distance will NOT be replaced. (dynamic status changes)

Performance Evaluation

- Trace-driven simulation on different patterns shows
 - LIRS **outperforms** existing replacement algorithms in almost all the cases.
 - The performance of LIRS is **not sensitive** to its only parameter **L_{hirs}** .
 - Performance is not affected even when **LIRS stack size** is **bounded**.
 - The time/space overhead is **as low as LRU**.
 - LRU is **a special case** of LIRS (without recording resident and non-resident HIR blocks, in the large stack).

Two Technical Issues to Turn it into Reality

- **High overhead in implementations**

- For each data access, a set of operations defined in replacement algorithms (e.g. LRU or LIRS) are performed
- This is not affordable to any systems, e.g. OS, buffer caches ...
- An approximation with reduced operations is required in practice

- **High lock contention cost**

- For concurrent accesses, the stack(s) need to be locked for each operation
- Lock contention limits the scalability of the system

- **Clock-pro** and **BP-Wrapper** addressed these two issues

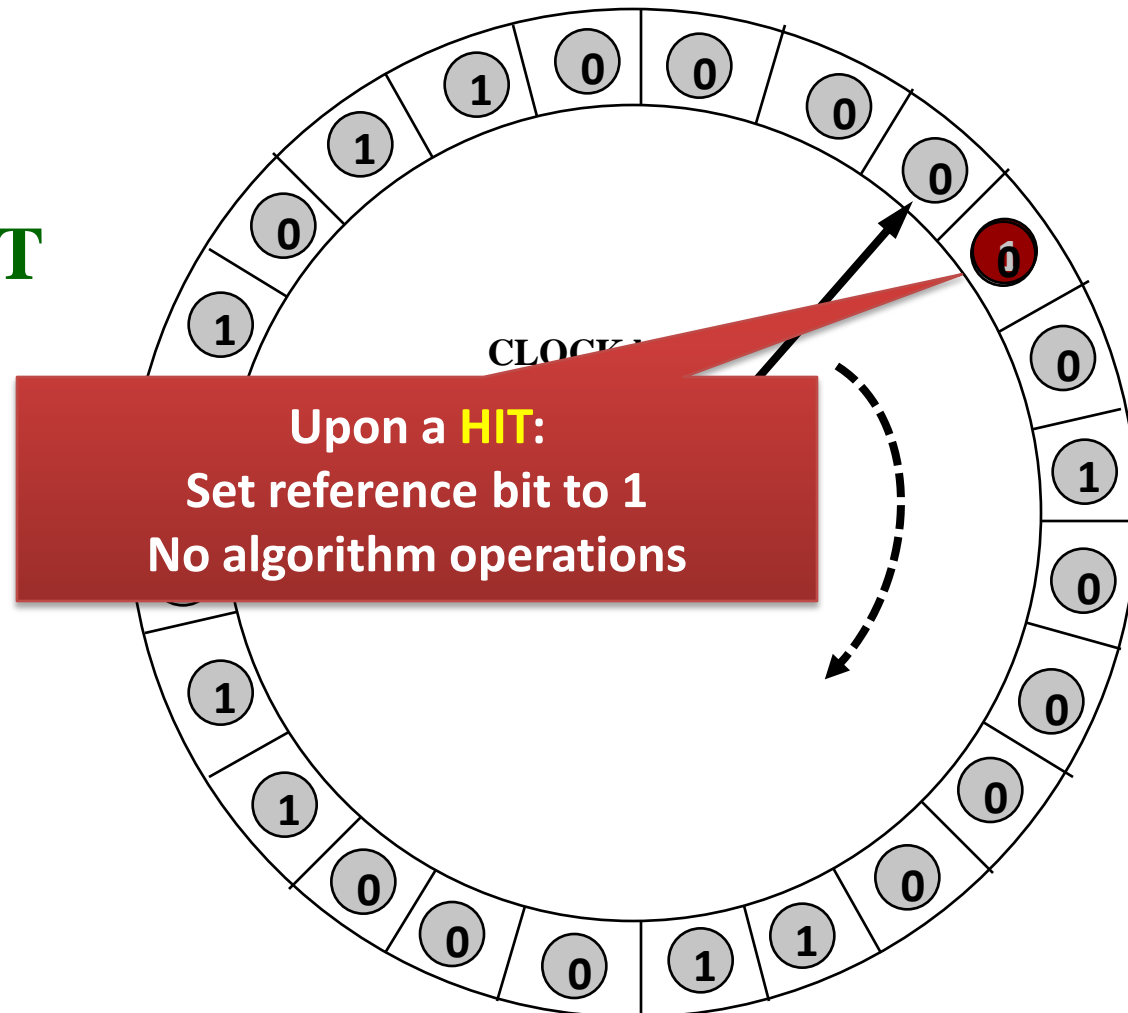
Only Approximations can be Implemented in OS

- The dynamic changes in LRU and LIRS cause some computing overhead, thus OS kernels cannot directly adopt them.
- An approximation reduce overhead at the cost of lower accuracy.
- The clock algorithm for LRU approximation was first implemented in the Multics system in 1968 at MIT by Corbato (1990 Turing Award Laureate)
- **Objective: LIRS approximation for OS kernels.**

Basic Operations of CLOCK Replacement

- All the resident pages are placed around a circular list, like a clock;
- Each page is associated with a **reference bit**, indicating if the page has been accessed.

On a block HIT



Unbalanced R&D on LRU versus CLOCK

LRU related work

- ❑ FBR (1990, SIGMETRICS)
- ❑ LRU-2 (1993, SIGMOD)
- ❑ 2Q (1994, VLDB)
- ❑ SEQ (1997, SIGMETRICS)
- ❑ LRFU (1999, OSDI)
- ❑ EELRU (1999, SIGMETRICS)
- ❑ MQ (2001, USENIX)
- ❑ LIRS (2002, SIGMETRICS)
- ❑ ARC (2003, FAST)

CLOCK related work

- ❑ GCLOCK (1978, ACM TDBS)



1968, Corbato



2003, Corbato

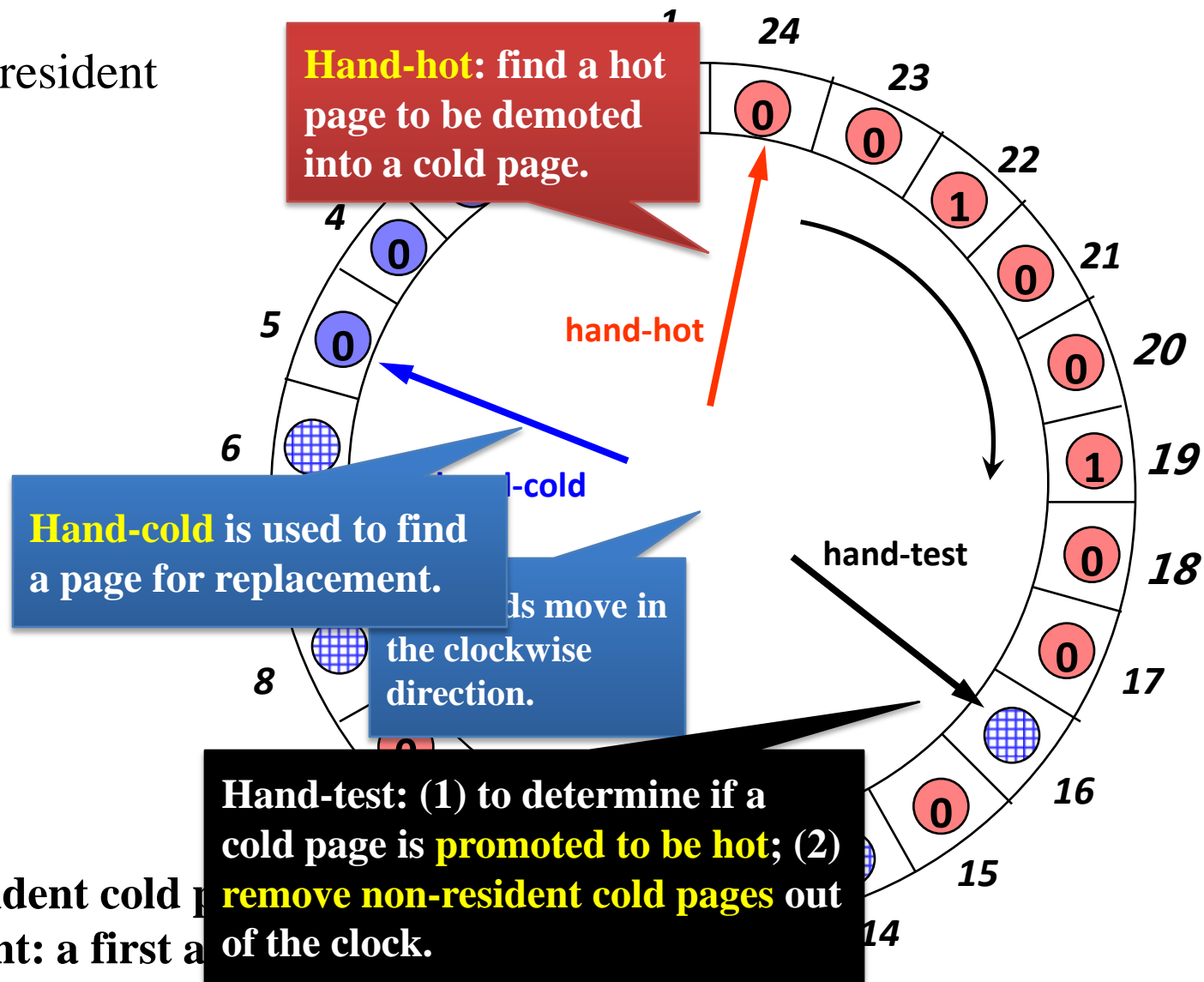
- ❑ CAR (2004, FAST)
- ❑ CLOCK-Pro (2005, USENIX)

0 1 Cold resident

0 1 Hot

Cold non-resident

CLOCK-Pro (USENIX'05)



Two reasons for a resident cold page to be replaced:

- (1) A fresh replacement: a first arrival at the head of the clock.
- (2) It is demoted from a hot page.

Trade-offs between Hit Ratios and Low Lock Contention

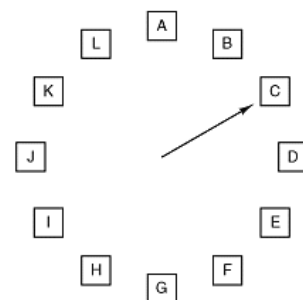
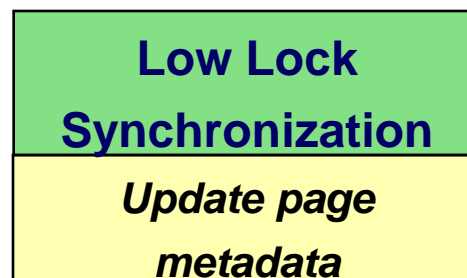
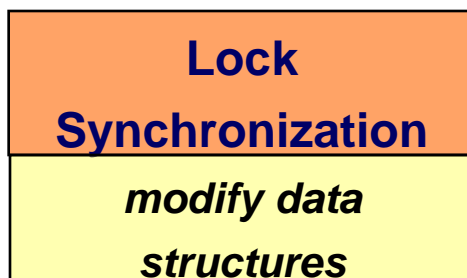
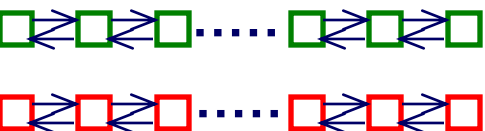
for **high hit ratio**

LRU-k, 2Q, LIRS,
ARC, SEQ,

Our Goal: to have both!

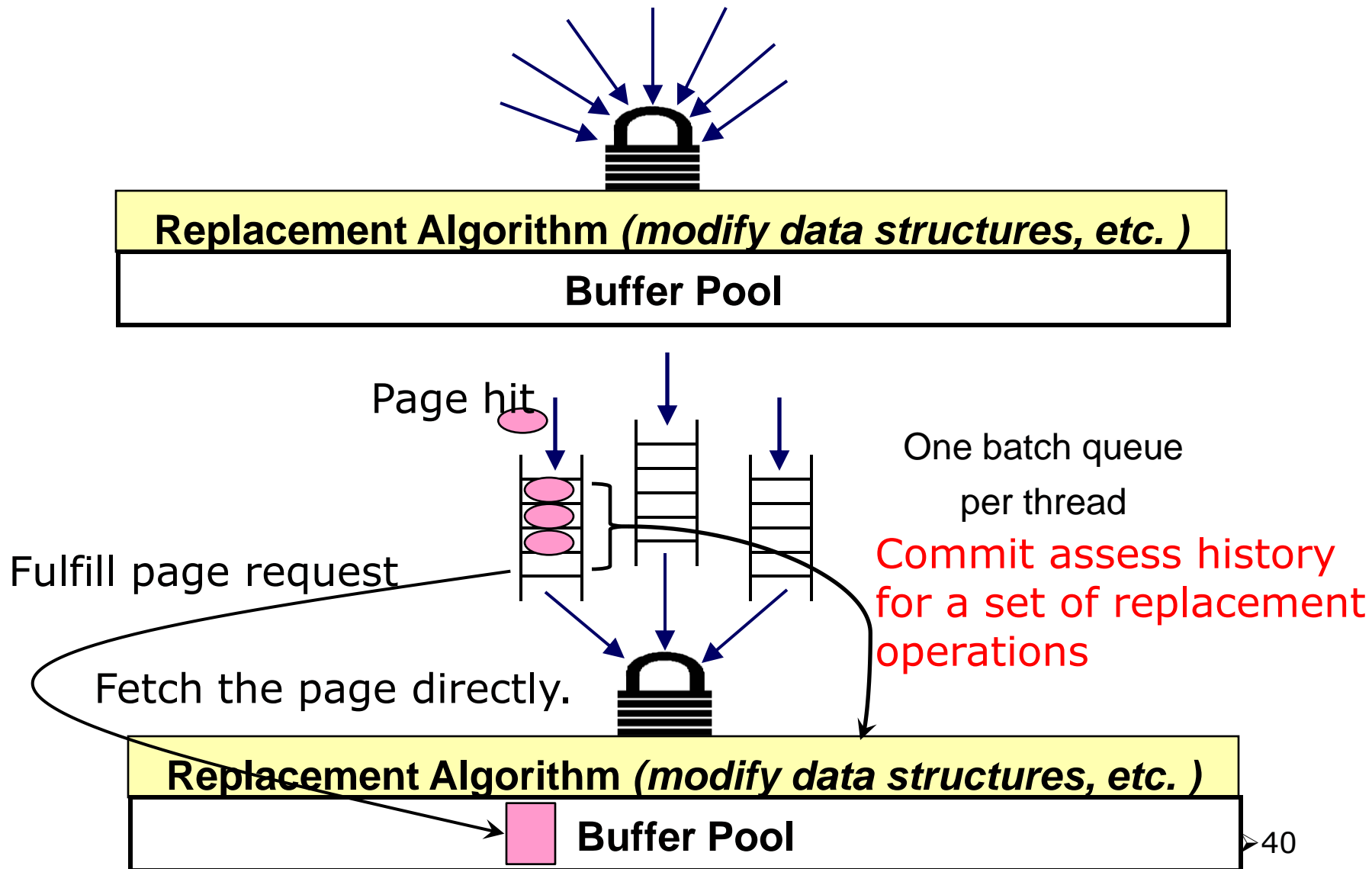
for **high scalability**

CLOCK, CLOCK-Pro,
and CAR

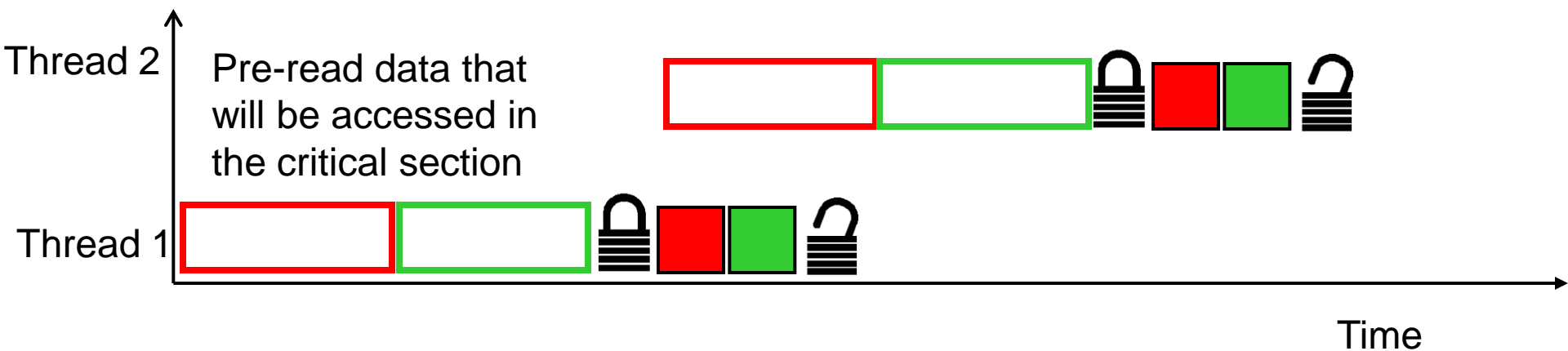
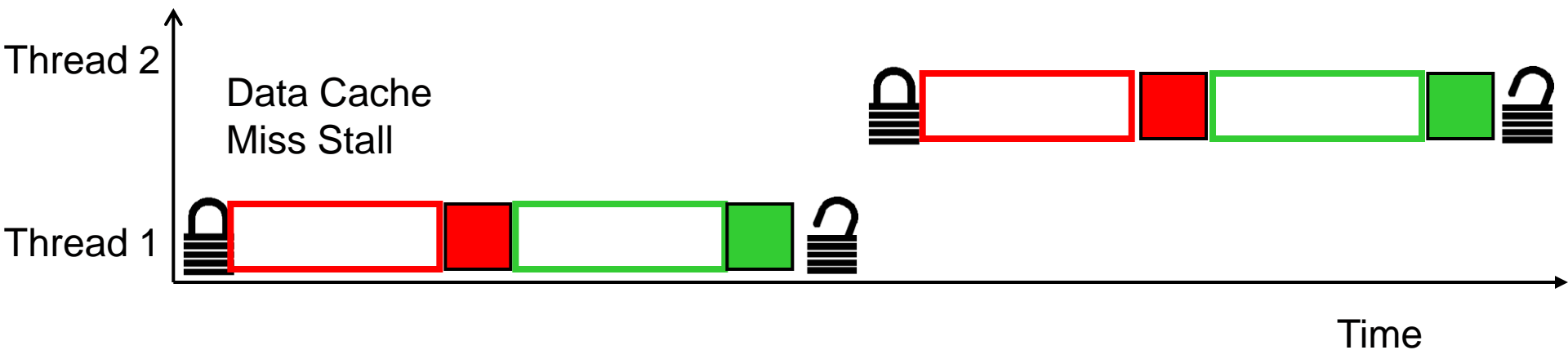


- Clock-based approximations lower hit ratios (compared to original ones).
- The transformation can be **difficult** and demand **great efforts**;
- Some algorithms **do not have** clock-based approximations.

Reducing Lock Contention by Batching Requests



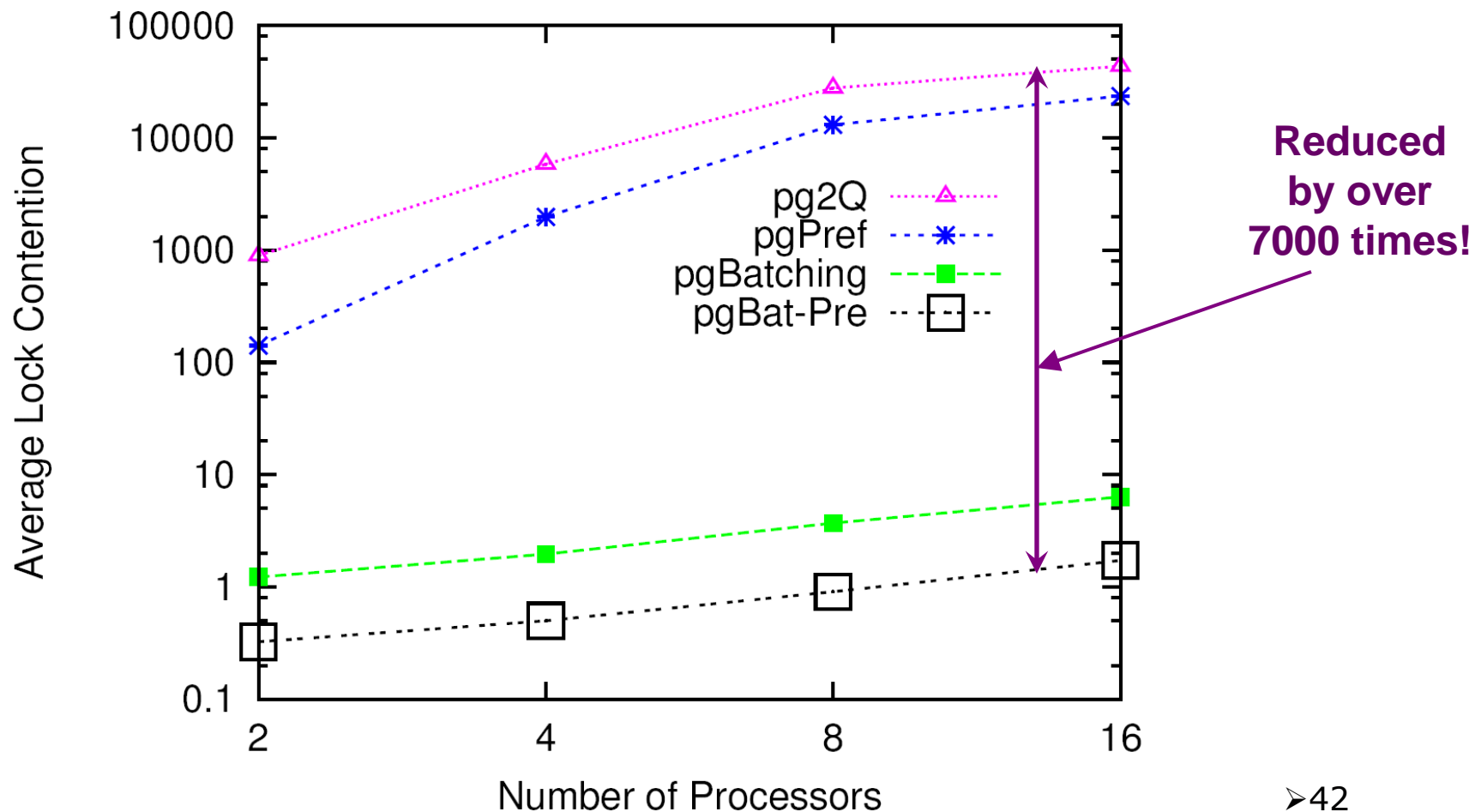
Reducing Lock Holding Time by Prefetching



Lock Contention Reduction by BP-Wrapper (ICDE'09)

Lock contention: a lock cannot be obtained without blocking;

Number of lock acquisitions (contention) per million page accesses.



Impact of LIRS in Academic Community

- LIRS is a **benchmark** to compare replacement algorithms
 - Reuse distance is first used in replacement algorithm design
 - A paper in SIGMETRICS'05 confirmed that LIRS outperforms all the other replacement.
 - LIRS has become a topic to teach in both graduate and undergraduate classes of OS, performance evaluation, and databases at many US universities.
 - The LIRS paper (SIGMETRICS'02) is highly and continuously cited.
- Linux Memory Management group has established an Internet Forum on **Advanced Replacement** for LIRS

LIRS has been adopted in MySQL

- MySQL is the most widely used relational database
 - 11 million installations in the world
 - The busiest Internet services use MySQL to maintain their databases for high volume Web sites: **google, YouTube, wikipedia, facebook, ...**
 - LIRS is managing the **buffer pool** of MySQL
 - The adoption is the most recent version **(5.1)**, November 2008.

MySQL :: The world's most popular open source database - Mozilla Firefox

File Edit View History Bookmarks Tools Help

http://dev.mysql.com/sources/doxygen/mysql-5.1/pgman_8hpp-source.html

MySQL.com Developer Zone Partners & Solutions Customer Login

DevZone Downloads Documentation Articles Forums Bugs Forge Blogs

Search

The world's most popular open source database

MySQL Newsletter
Subscribe Today!

zmanda
Open Source Backup

Recommended Servers for MySQL

Contact a MySQL Representative

Login | Register

Main Page Modules Namespaces Classes Files Related Pages

File List File Members

Search for

The world's most popular open source database

mysql/src/5.1-dbg/storage/ndb/src/kernel/blocks/pgman.hpp

Go to the [documentation of this file](#).

```

00001 /* Copyright (C) 2003 MySQL AB
00002
00003     This program is free software; you can redistribute it and/or modify
00004     it under the terms of the GNU General Public License as published by
00005     the Free Software Foundation; either version 2 of the License, or
00006     (at your option) any later version.
00007
00008     This program is distributed in the hope that it will be useful,
00009     but WITHOUT ANY WARRANTY; without even the implied warranty of
00010     MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
00011     GNU General Public License for more details.
00012
00013     You should have received a copy of the GNU General Public License

```

Done

MySQL :: The world's most popular open source database - Mozilla Firefox

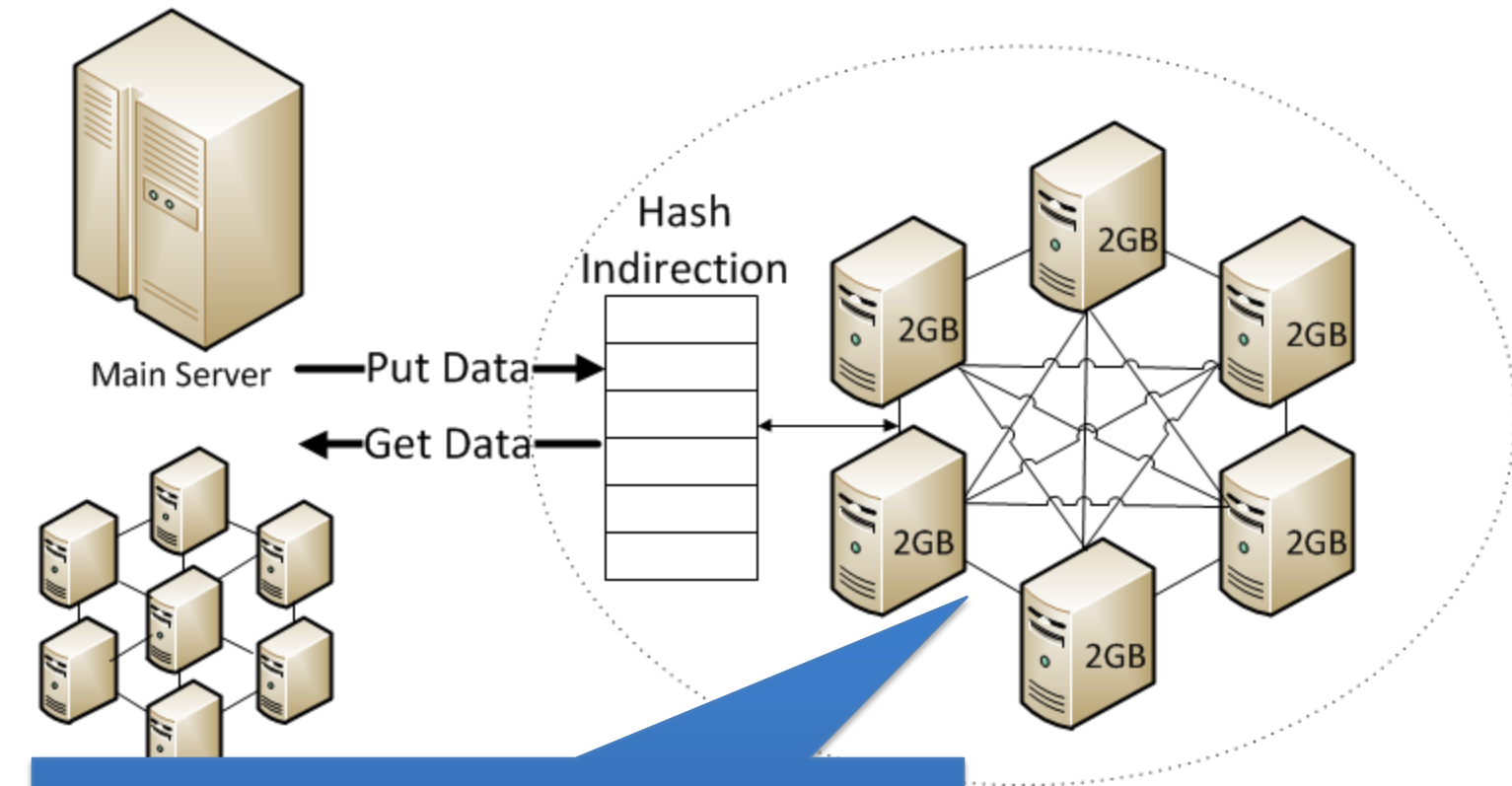
File Edit View History Bookmarks Tools Help

http://dev.mysql.com/sources/doxygen/mysql-5.1/pgman_8hpp-source.html

00058 * A local check point (LCP) periodically performs a complete pageout of
00059 * dirty pages. It must iterate over a list which will cover all pages
00060 * which had been dirty since LCP start.
00061 *
00062 * A clean page is a candidate ("victim") for being "unmapped" and
00063 * "evicted" from the cache, to allow another page to become resident.
00064 * This process is called "page replacement".
00065 *
00066 * PAGE REPLACEMENT
00067 *
00068 * Page replacement uses the LIRS algorithm (Jiang-Zhang).
00069 *
00070 * The "recency" of a page is the time between now and the last request
00071 * for the page. The "inter-reference recency" (IRR) of a page is the
00072 * time between the last 2 requests for the page. "Time" is advanced by
00073 * request for any page.
00074 *
00075 * Page entries are divided into "hot" ("lir") and "cold" ("hir"). Here
00076 * lir/hir refers to low/high IRR. Hot pages are always resident but
00077 * cold pages need not be.
00078 *
00079 * Number of hot pages is limited to slightly less than number of cache
00080 * pages. Until this number is reached, all used cache pages are hot.
00081 * Then the algorithm described next is applied. The algorithm avoids
00082 * storing any of the actual recency values.

Done

Infinispan (a Java-based Open Software)

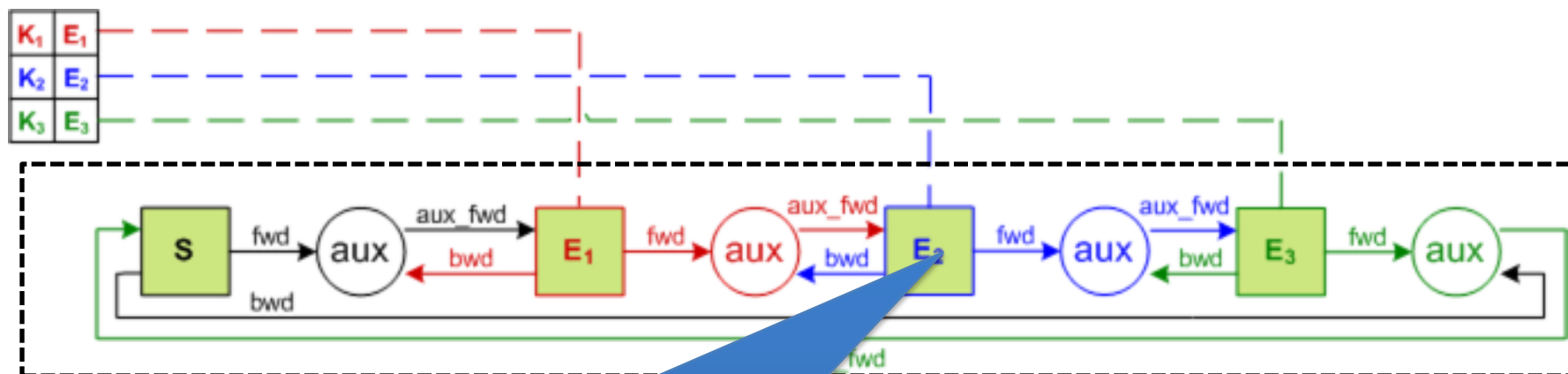


The data grid forms a huge in-memory cache being managed using **LIRS** (Least Recently Used) (Infinispan Data Grid 2GB in-memory heap)

BP-Wrapper is used to ensure lock-free

Concurrentlinkedhashmap as a Software Cache

- Linked list structure (a Java class)



Elements are Linked and managed using
LIRS replacement policy. **BP-Wrapper**
 ensures lock contention-free

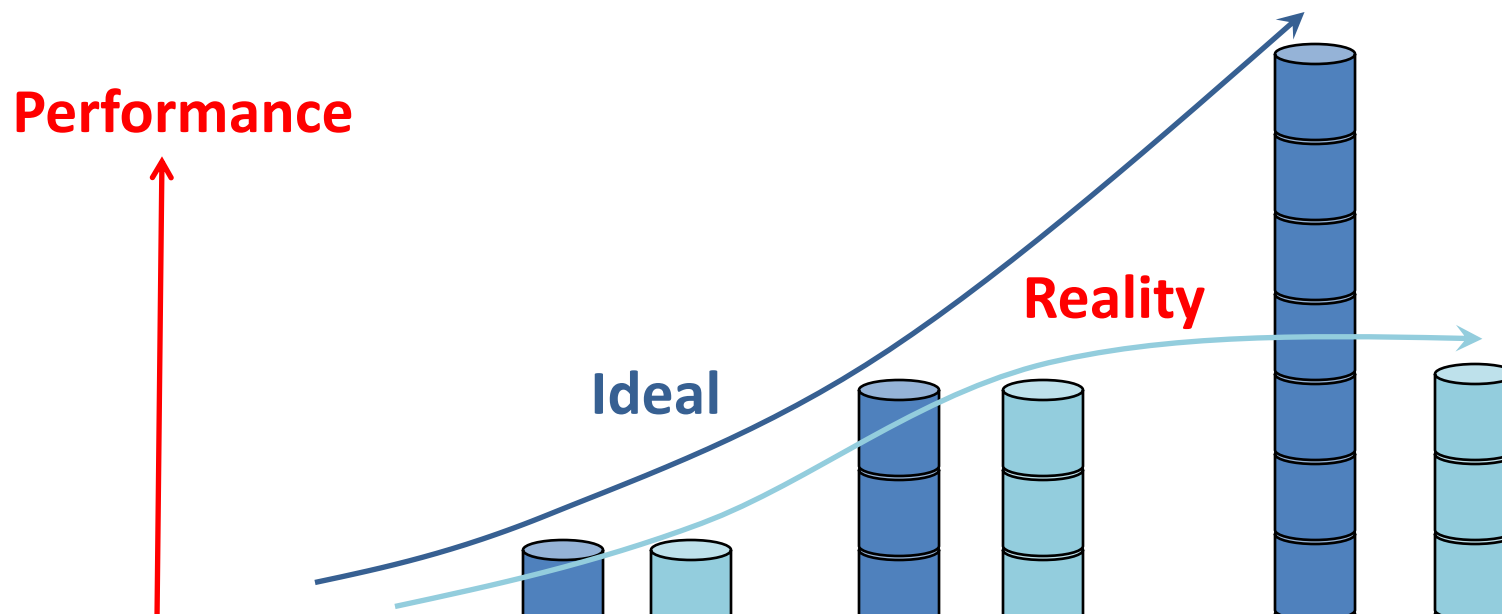
LIRS Principle in Hardware Caches

- A cache replacement hardware implementation based on Re-Reference Interval prediction (RRIP)
 - Presented in ISCA'10 by Intel
 - Two bits are added to each cache line to measure reuse-distance in a static and dynamic way
 - Performance gains are up to 4-10%
 - Hardware cost may not be affordable in practice.

Impact of Clock-Pro in OS and Other Systems

- Clock-pro has been adopted in FreeBSD/NetBSD (open source Unix)
- Two patches in Linux kernel for users
 - **Clock-pro patches** in 2.6.12 by **Rik van Riel**
 - **PeterZClockPro2** in 2.6.15-17 by **Peter Zijlstra**
- Clock-pro is patched in Apache Derby (a relational DB)
- Clock-pro is patched in OpenLDAP (directory accesses)

Multi-core Cannot Deliver Expected Performance as It Scales



Throughput = Concurrency/Latency

- Exploiting parallelism
- Exploiting locality

“The Troubles with Multicores”, David Patterson, IEEE Spectrum, July, 2010

“Finding the Door in the Memory Wall”, Erik Hagersten, HPCwire, Mar, 2009

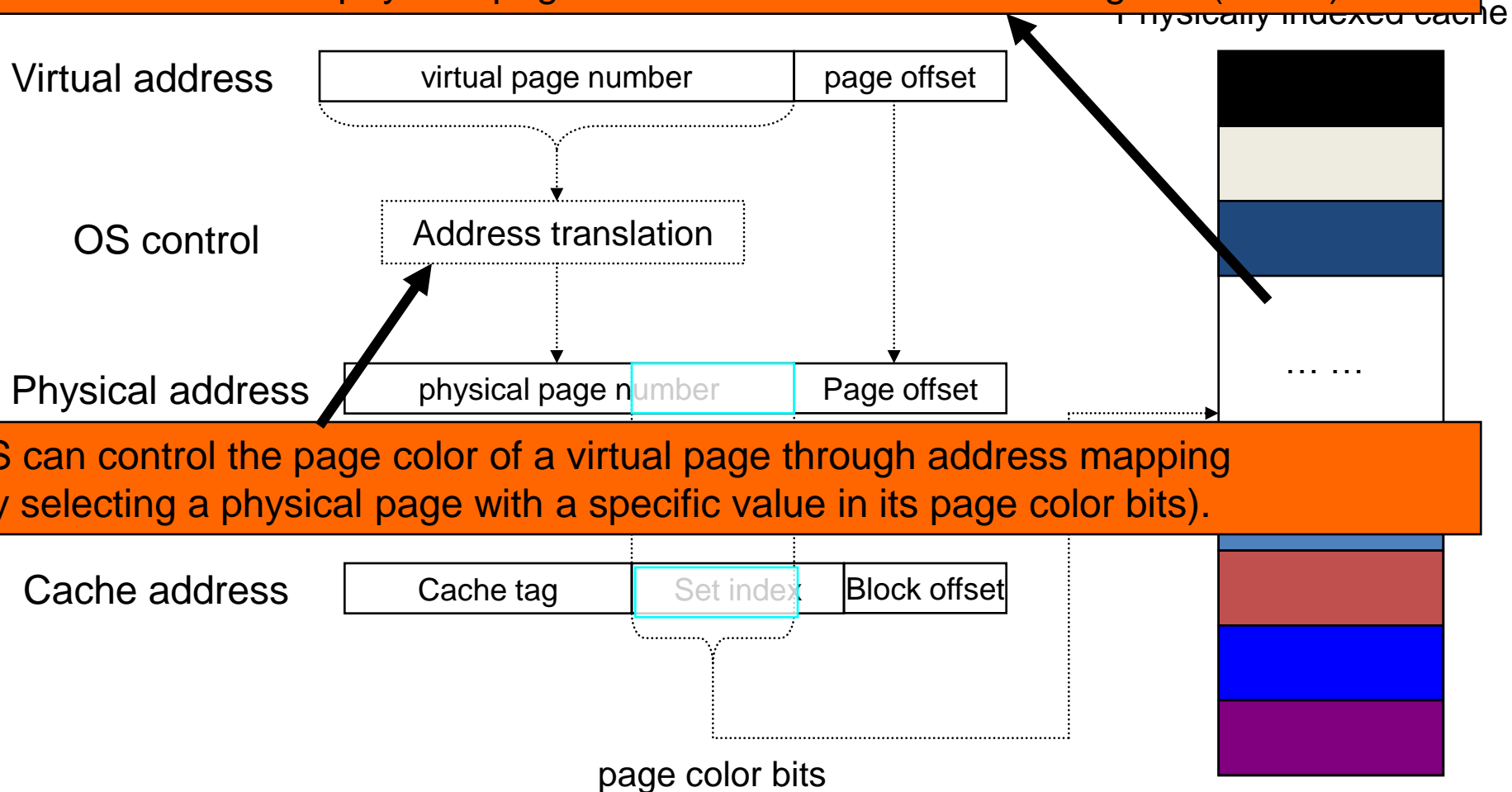
“Multicore Is Bad News For Supercomputers”, Samuel K. Moore, IEEE Spectrum, Nov, 2008

Challenges of Managing LLC in Multi-cores

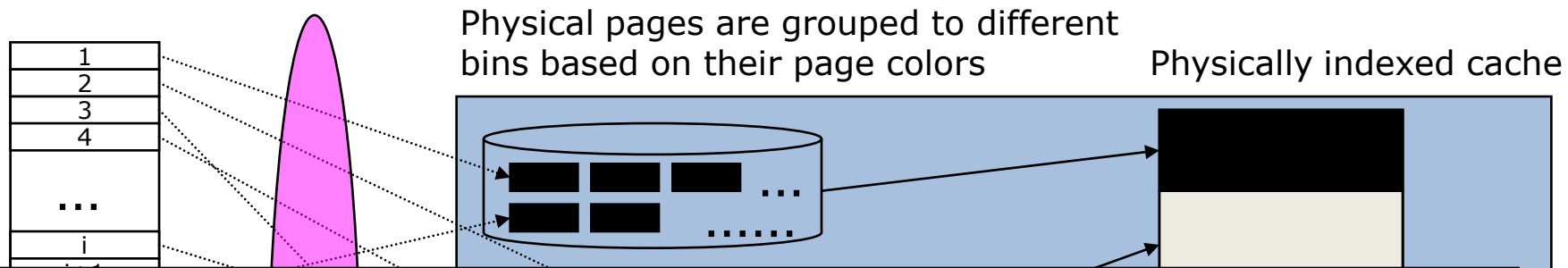
- Recent theoretical results about LLC in multicores
 - **Single core**: optimal offline LRU algorithm exists
 - Online LRU is k -competitive (k is the cache size)
 - **Multicore**: finding an offline optimal LRU is **NP-complete**
 - Cache partitioning for threads: an optimal solution in theory
- System Challenges in practice
 - LLC **lacks necessary hardware mechanism** to control inter-thread cache contention
 - LLC share the same design with single-core caches
 - System software has **limited information and methods** to effectively control cache contention

OS Cache Partitioning in Multi-cores (HPCA'08)

- Physically indexed caches are divided into multiple regions (colors).
- All cache lines in a physical page are cached in one of those regions (colors).

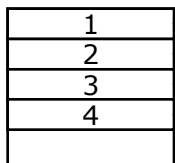


Shared LLC can be partitioned into multiple regions



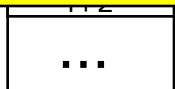
Shared cache is partitioned between two processes through OS address mapping.

Process 1



Address mapping

Main memory space needs to be partitioned too (**co-partitioning**).



Process 2

Implementations in Linux and its Impact

- **Static partitioning**
 - Predetermines the amount of cache blocks allocated to each running process at the beginning of its execution
- **Dynamic cache partitioning**
 - Adjusts cache allocations among processes dynamically
 - Dynamically changes processes' cache usage through OS page address re-mapping (page re-coloring)
- **Current Status of the system facility**
 - Open source in Linux kernels
 - adopted as a software solution in Intel SSG in May 2010
 - used in applications of Intel platforms, e.g. automation

Final Remarks: Why LIRS-related Efforts Make the Difference?

- **Caching the most deserved data blocks**
 - Using reuse-distance as the ruler, approaching to the optimal
 - 2Q, LRU-k, ARC, and others can still cache non-deserved blocks
- **LIRS with its two-stack yields constant operations: $O(1)$**
 - Consistent to LRU, but recording much more useful information
- **Clock-pro turns LIRS into reality in production systems**
 - None of other algorithms except ARC have approximation versions
- **BP-Wrapper ensures lock contention free in DBMS**
- **OS partitioning executes LIRS principle in LLC in multicores**
 - Protect strong locality data, and control weak locality data

Acknowledgement to Co-authors and Sponsors

- Song Jiang
 - Ph.D.'04 at William and Mary, faculty at Wayne State
- Feng Chen
 - Ph.D.'10 , Intel Labs (Oregon)
- Xiaoning Ding
 - Ph.D. '10 , Intel Labs (Pittsburgh)
- Qingda Lu
 - Ph.D.'09 , Intel (Oregon)
- Jiang Lin
 - Ph.D'08 at Iowa State, AMD
- Zhao Zhang
 - Ph.D.'02 at William and Mary, faculty at Iowa State
- P. Sadayappan, CSE faculty
- Continuous support from the National Science Foundation

Thank You !

Xiaodong Zhang

✉: zhang@cse.ohio-state.edu

<http://www.cse.ohio-state.edu/~zhang>