Lecture 12: Large Cache Design

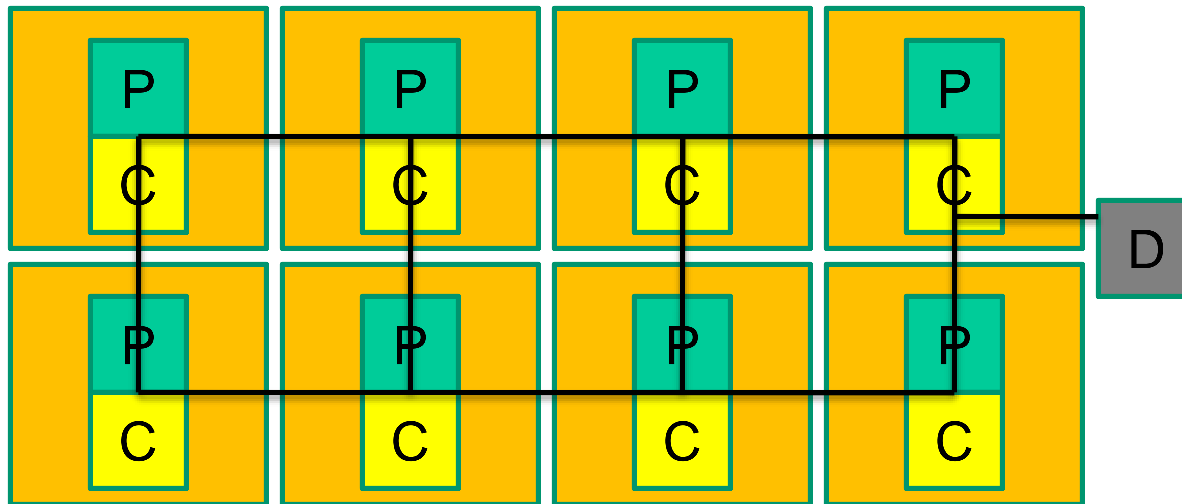
- Topics: Shared vs. private, centralized vs. decentralized, UCA vs. NUCA, recent papers

Shared Vs. Private

- SHR: No replication of blocks
- SHR: Dynamic allocation of space among cores
- SHR: Low latency for shared data in LLC (no indirection thru directory)
- SHR: No interconnect traffic or tag replication to maintain directories
- PVT: More isolation and better quality-of-service
- PVT: Lower wire traversal when accessing LLC hits, on average
- PVT: Lower contention when accessing some shared data
- PVT: No need for software support to maintain data proximity

Innovations for Private Caches: Cooperation

- Cooperative Caching, Chang and Sohi, ISCA'06



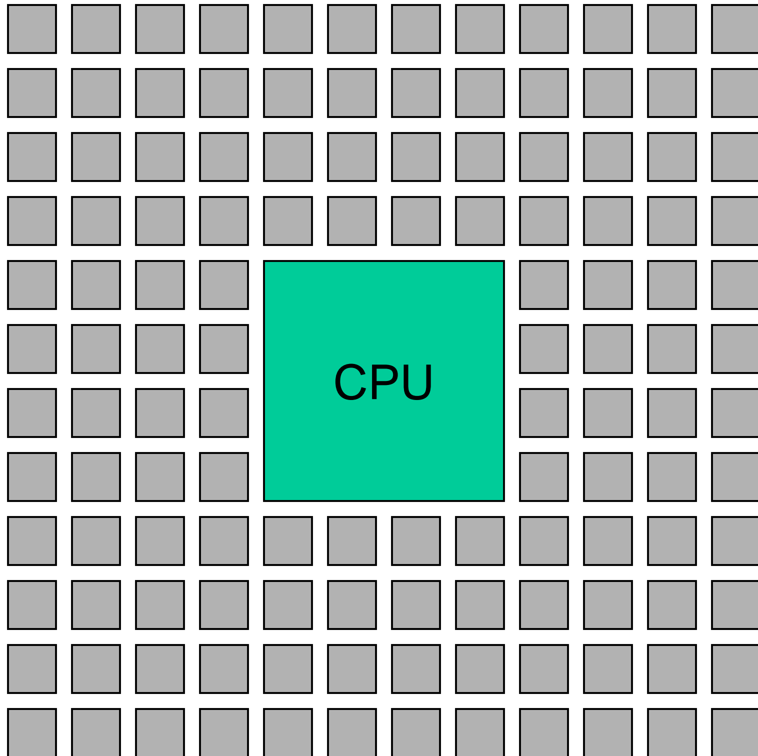
- Prioritize replicated blocks for eviction with a given probability; directory must track and communicate a block's "replica" status

- "Singlet" blocks are sent to sibling caches upon eviction (probabilistic one-chance forwarding); blocks are placed in LRU position of sibling

Dynamic Spill-Receive

- Dynamic Spill-Receive, Qureshi, HPCA'09
- Instead of forcing a block upon a sibling, designate caches as Spillers and Receivers and all cooperation is between Spillers and Receivers
- Every cache designates a few of its sets as being Spillers and a few of its sets as being Receivers (each cache picks different sets for this profiling)
- Each private cache independently tracks the global miss rate on its S/R sets (either by watching the bus or at the directory)
- The sets with the winning policy determine the policy for the rest of that private cache – referred to as set-dueling

Innovations for Shared Caches: NUCA



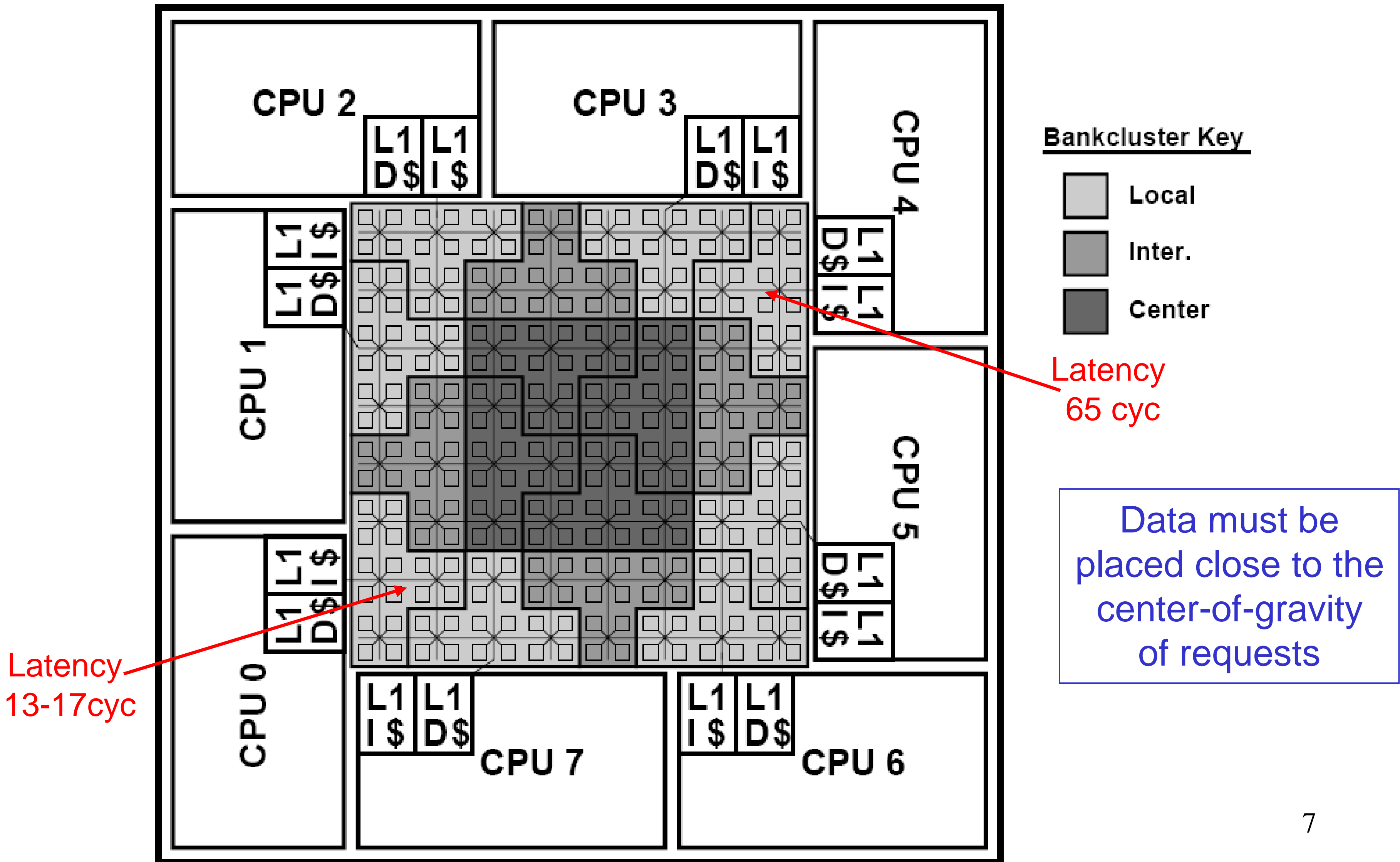
Issues to be addressed for Non-Uniform Cache Access:

- Mapping
- Migration
- Search
- Replication

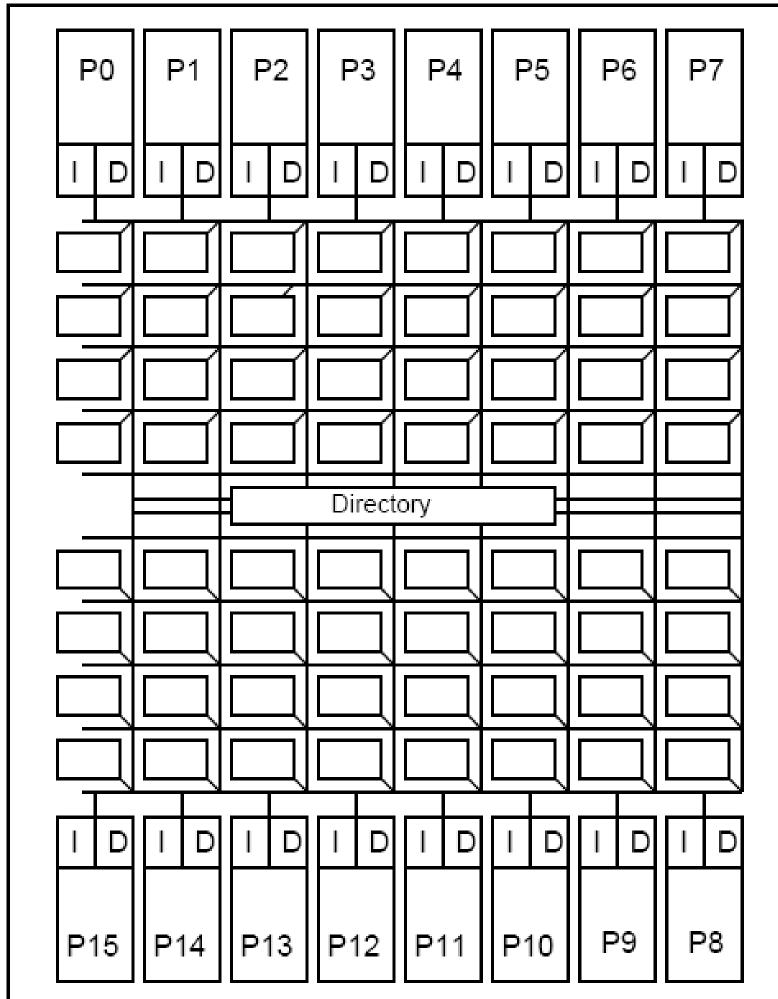
Static and Dynamic NUCA

- Static NUCA (S-NUCA)
  - The address index bits determine where the block is placed; sets are distributed across banks
  - Page coloring can help here to improve locality
- Dynamic NUCA (D-NUCA)
  - Ways are distributed across banks
  - Blocks are allowed to move between banks: need some search mechanism
  - Each core can maintain a partial tag structure so they have an idea of where the data might be (complex!)
  - Every possible bank is looked up and the search propagates (either in series or in parallel) (complex!)

Beckmann and Wood, MICRO'04



Alternative Layout

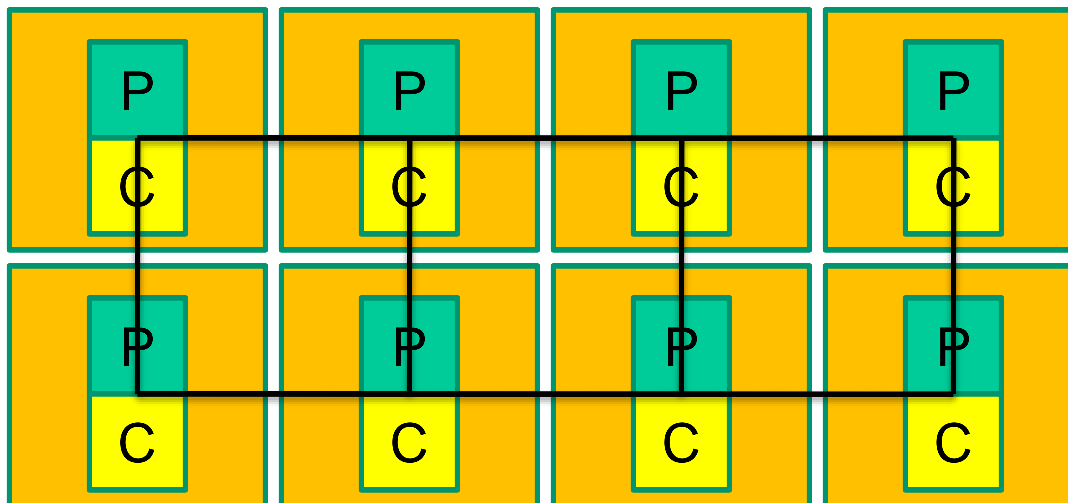


From Huh et al., ICS'05:

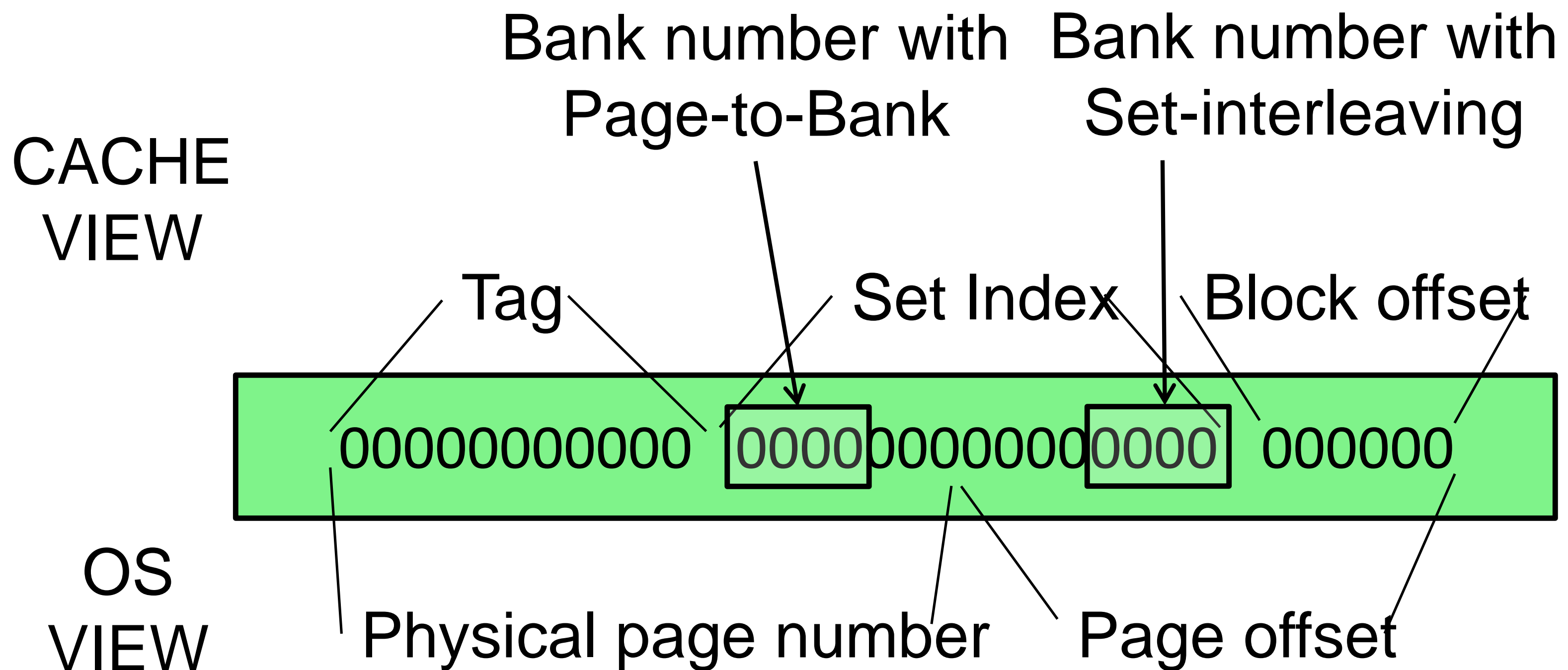
- Paper also introduces the notion of sharing degree
- A bank can be shared by any number of cores between N=1 and 16.
- Will need support for L2 coherence as well

Victim Replication, Zhang & Asanovic, ISCA'05

- Large shared L2 cache (each core has a local slice)
- On an L1 eviction, place the victim in local L2 slice (if there are unused lines)
- The replication does not impact correctness as this core is still in the sharer list and will receive invalidations
- On an L1 miss, the local L2 slice is checked before fwding the request to the correct slice



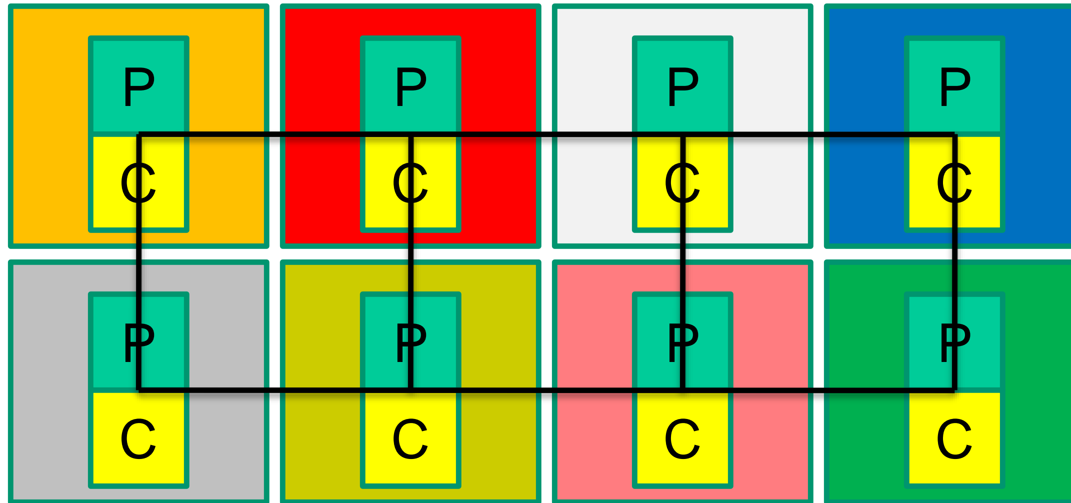
Page Coloring



Cho and Jin, MICRO'06

- Page coloring to improve proximity of data and computation
- Flexible software policies
- Has the benefits of S-NUCA (each address has a unique location and no search is required)
- Has the benefits of D-NUCA (page re-mapping can help migrate data, although at a page granularity)
- Easily extends to multi-core and can easily mimic the behavior of private caches

Page Coloring Example



- Awasthi et al., HPCA'09 propose a mechanism for hardware-based re-coloring of pages without requiring copies in DRAM memory

- They also formalize the cost functions that determine the optimal home for a page

R-NUCA, Hardavellas et al., ISCA’09

- A page is categorized as “shared instruction”, “private data”, or “shared data”; the TLB tracks this and prevents access of a different kind

- Depending on the page type, the indexing function into the shared cache is different
  - “Private data” only looks up the local bank
  - “Shared instruction” looks up a region of 4 banks
  - “Shared data” looks up all the banks

Rotational Interleaving

- Can allow for arbitrary group sizes and a numbering that distributes load

00	01	10	11
10	11	00	01
00	01	10	11
10	11	00	01

Basic Replacement Policies

- LRU: least recently used
- LFU: least frequently used (requires small saturating cntrs)
- pseudo-LRU: organize ways as a tree and track which sub-tree was last accessed
- NRU: every block has a bit; the bit is reset to 0 upon touch; when evicting, pick a block with its bit set to 1; if no block has a 1, make every bit 1

Why the Basic Policies Fail

- Access types that pollute the cache without yielding too many hits: streaming (no reuse), thrashing (distant reuse)
- Current hit rates are far short of those with an oracular replacement policy (Belady): evict the block whose next access is most distant
- A large fraction of the cache is useless – blocks that have serviced their last hit and are on the slow walk from MRU to LRU

Insertion, Promotion, Victim Selection

- Instead of viewing the set as a recency stack, simply view it as a priority list; in LRU, priority = recency
- When we fetch a block, it can be inserted in any position in the list
- When a block is touched, it can be promoted up the priority list in one of many ways
- When a block must be victimized, can select any block (not necessarily the tail of the list)

MIP, LIP, BIP, and DIP

Qureshi et al., ISCA'07

- MIP: MRU insertion policy (the baseline)
- LIP: LRU insertion policy; assumes that blocks are useless and should be kept around only if touched twice in succession
- BIP: Bimodal insertion policy; put most blocks at the tail; with a small probability, insert at head; for thrashing workloads, it can retain part of the working set and yield hits on them
- DIP: Dynamic insertion policy: pick the better of MIP and BIP; decide with set-dueling

RRIP

Jaleel et al., ISCA'10

- Re-Reference Interval Prediction: in essence, insert blocks near the end of the list than at the very end
- Implement with a multi-bit version of NRU: zero counter on touch, evict block with max counter, else increment every counter by one
- RRIP can be easily implemented by setting the initial counter value to max-1 (does not require list management)

UCP

Qureshi et al., MICRO'06

- Utility Based Cache Partitioning: partition ways among cores based on estimated marginal utility of each additional way to each core

- Each core maintains a shadow tag structure for the L2 cache that is populated only by requests from this core; the core can now estimate hit rates if it had W ways of L2

- Every epoch, stats are collected and ways re-assigned

- Can reduce shadow tag storage overhead by using set sampling and partial tags

TADIP

Jaleel et al., PACT'08

- Thread-aware DIP: each thread dynamically decides to use MIP or BIP; threads that use BIP get a smaller partition of cache

- Better than UCP because even for a thrashing workload, part of the working set gets to stay in cache

- Need lots of set dueling monitors, but no need for extra shadow tags

PIPP

Xie and Loh, ISCA'09

- Promotion/Insertion pseudo partitioning: incoming blocks are inserted in arbitrary positions in the list and on every touch, they are gradually promoted up the list with a given probability

- Applications with a large partition are inserted near the head of the list and promoted aggressively

- Partition sizes are decided with marginal utility estimates

- In a few sets, a core gets to use N-1 ways and count hits to each way; other threads only get to use the last way

Aggressor VT

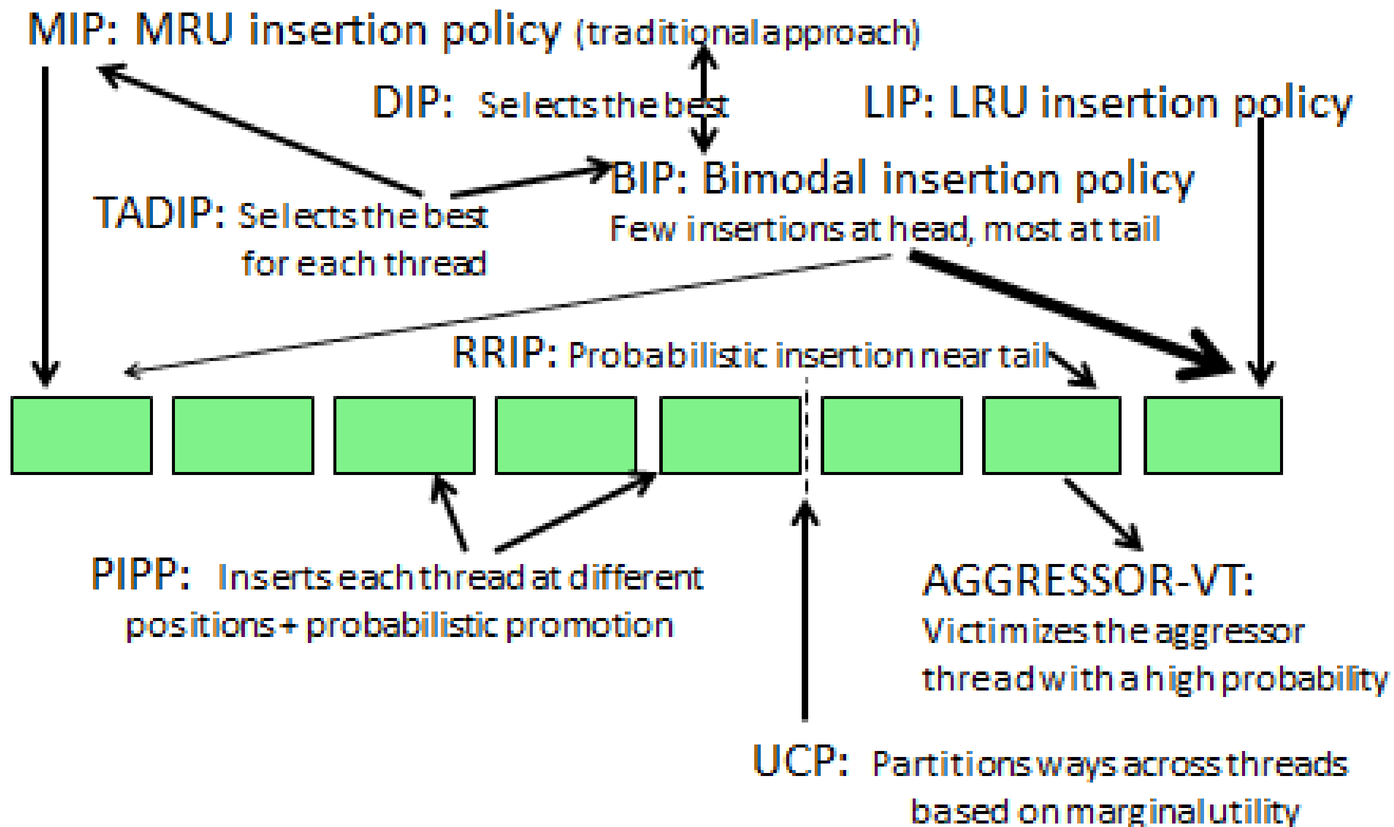
Liu and Yeung, PACT'09

- In an oracle policy, 80% of the evictions belong to a thrashing aggressor thread
- Hence, if the LRU block belongs to an aggressor thread, evict it; else, evict the aggressor thread's LRU block with a probability of either 99% or 50%
- At the start of each phase change, sample behavior for that thread in one of three modes: non-aggr, aggr-99%, aggr-50%; pick the best performing mode

Set Partitioning

- Can also partition sets among cores by assigning page colors to each core
- Needs little hardware support, but must adapt to dynamic arrival/exit of tasks

Overview



Title

- Bullet