ECE 5730
Memory Systems
Spring 2009

Cache Content Management

Cornell University

Announcements

- **Quiz 3 on Tuesday**
- **Quiz 2**
  - **Average = 6.8/10**

Where We're Headed

- Non-uniform cache architectures (today)
- Off-line content management
  - Partitioning heuristics (today)
  - Prefetching heuristics
  - Locality optimization
- Combined approaches
- Cache power management
- Cache case studies

More on Prefetching

- **Actual lookup: cache lookup generated by an external source, e.g., the CPU**

regular lookup. CPU actually wants something, go get it

- **Prefetch lookup: cache interrogates itself to see if a line is resident or must be prefetched**

I want to see if the thing I'm going to prefetch is already there (i.e. no stream buffer)

- **Access ratio: (prefetch + actual)/actual**

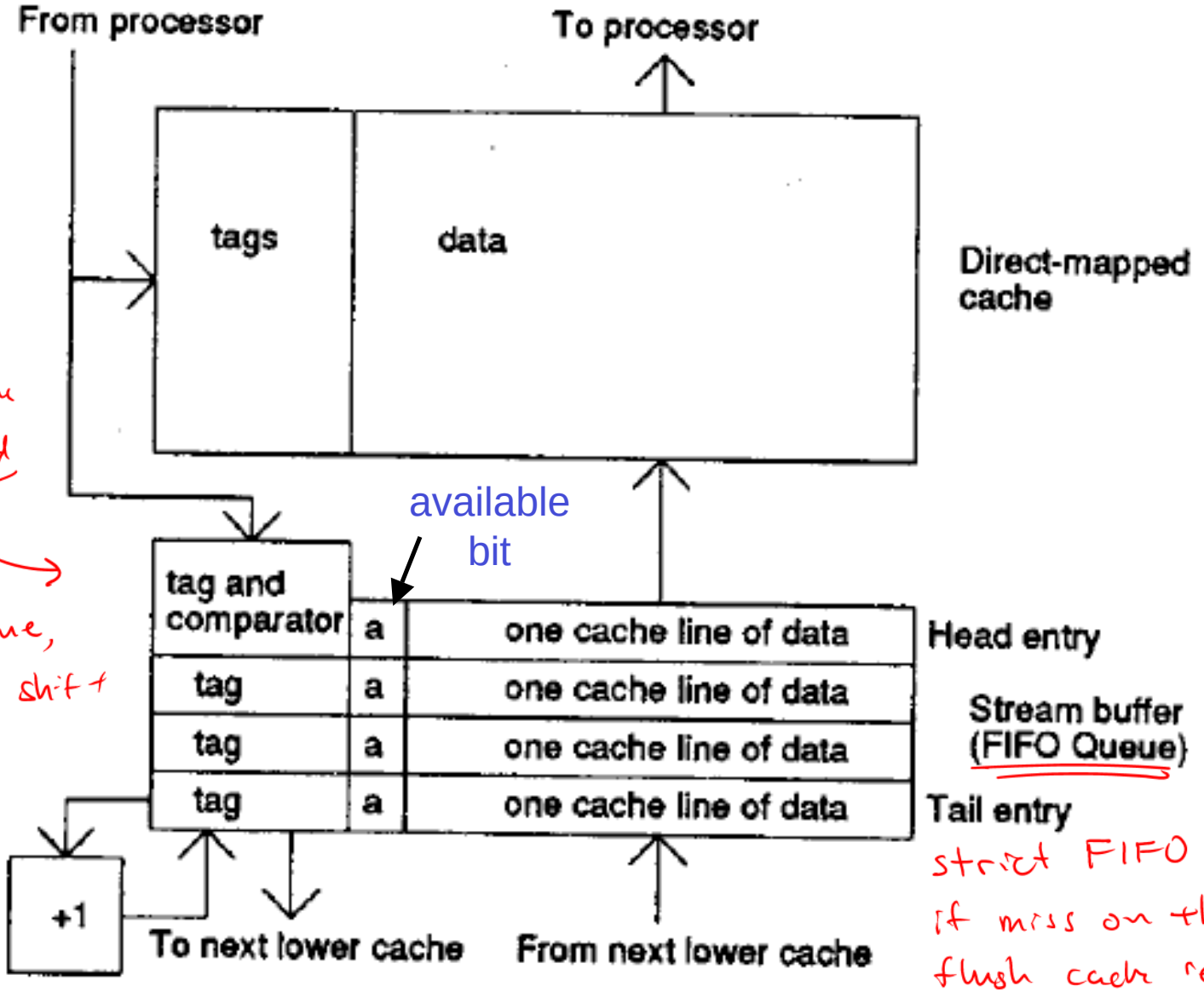
↳ high ratio means lots of prefetching lookups

Simplistic Prefetching Approaches

- **Prefetch next block when current one accessed**
  - **Access ratio = 2**
- **Prefetch next block when current one misses**
- **Tagged prefetch**
  - **Tag bit associated with each cache block**
  - **Initialized to 0 when block brought into the cache**
  - **Set to 1 when the block is accessed by the CPU**
  - **0 to 1 transition causes a prefetch of the next block**

Errata

Stream Buffer Management



only check the head

hit: continue, FIFO queue shift

miss: empty queue

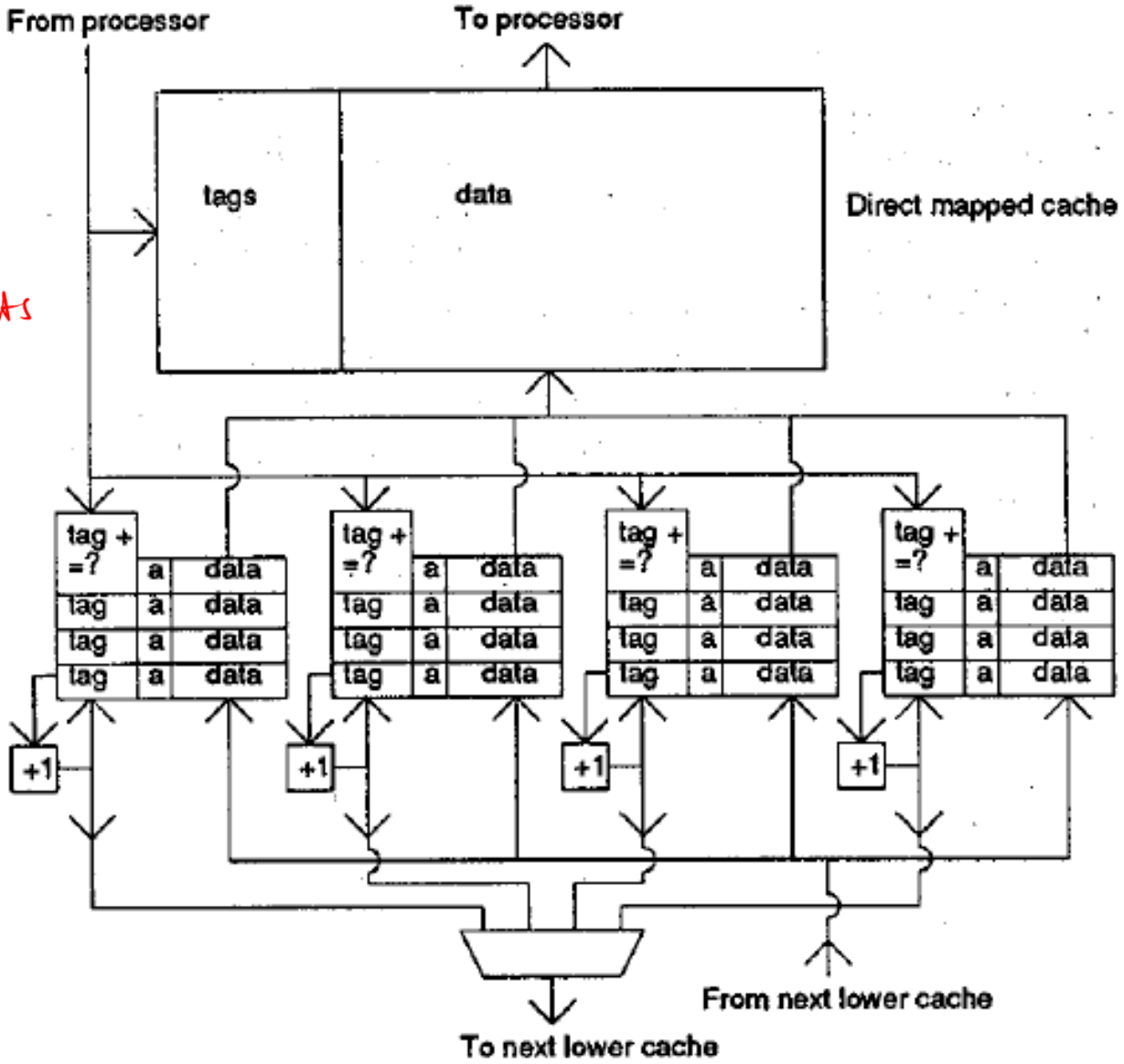
naive! even if position 3 would hit, we'd still flush!

strict FIFO queue, if miss on the head, flush cache regardless if we'd hit on a different cache line

[Jouppi90]

Errata

Multi-Way Stream Buffer Management



The dumping of the queue contents occurs on a miss to the queue head

[Jouppi90]

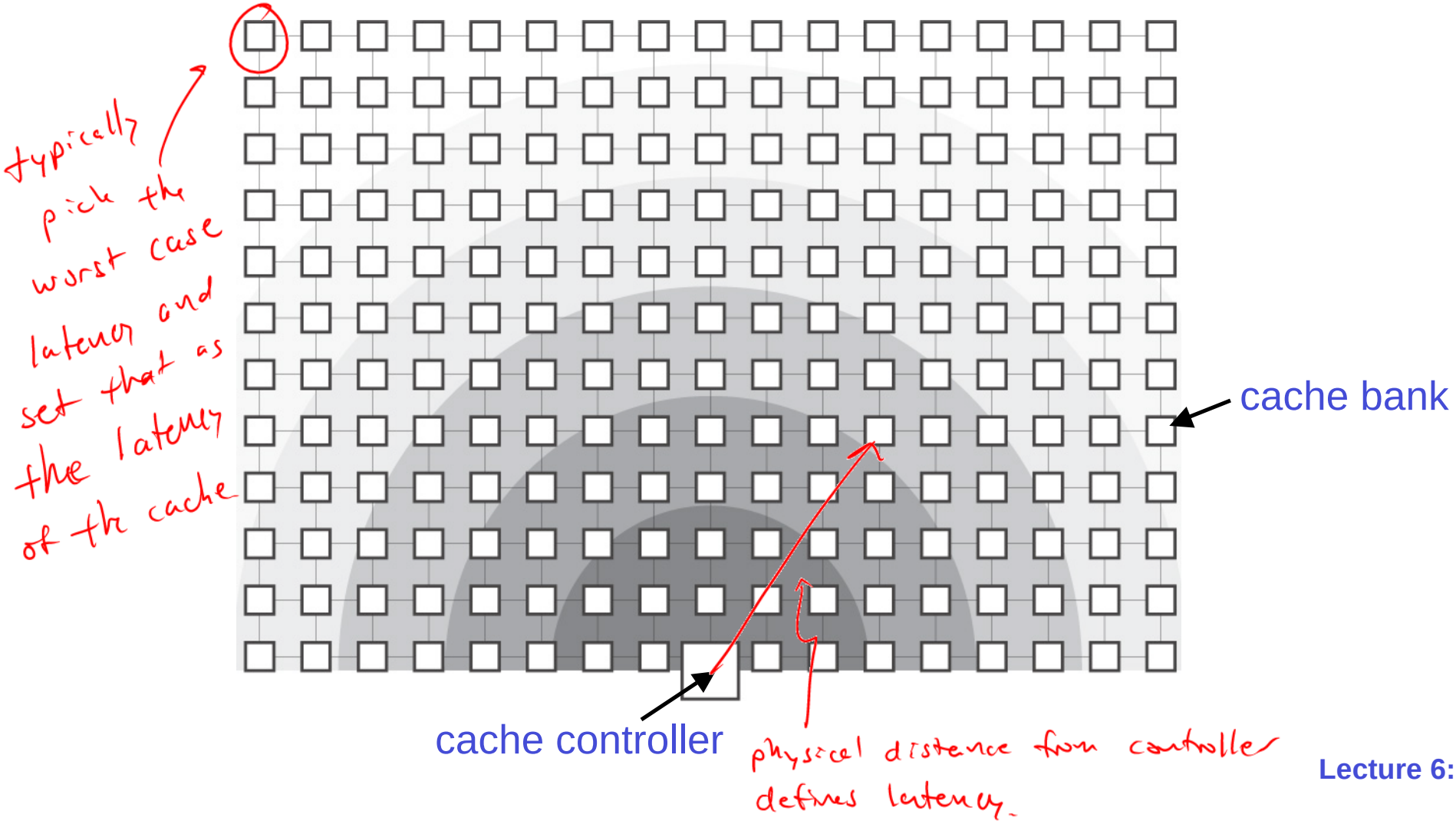
Cache Content Management

- **Partitioning heuristics**
  - **Which items get placed and where (cache level, cache sets/ways, buffers)**
- **Fetching heuristics**
  - **When to bring an item into the cache**
- **Locality optimizations**
  - **Change layout or ordering to improve reuse**

Decisions can be made at runtime (*on-line heuristics*),
at design/compile time (*off-line heuristics*),
or a combination of the two (*combined approaches*)

Non-Uniform Cache Access Time

- **For large L2 or L3 caches (10's of MB), access latencies can vary significantly among banks**



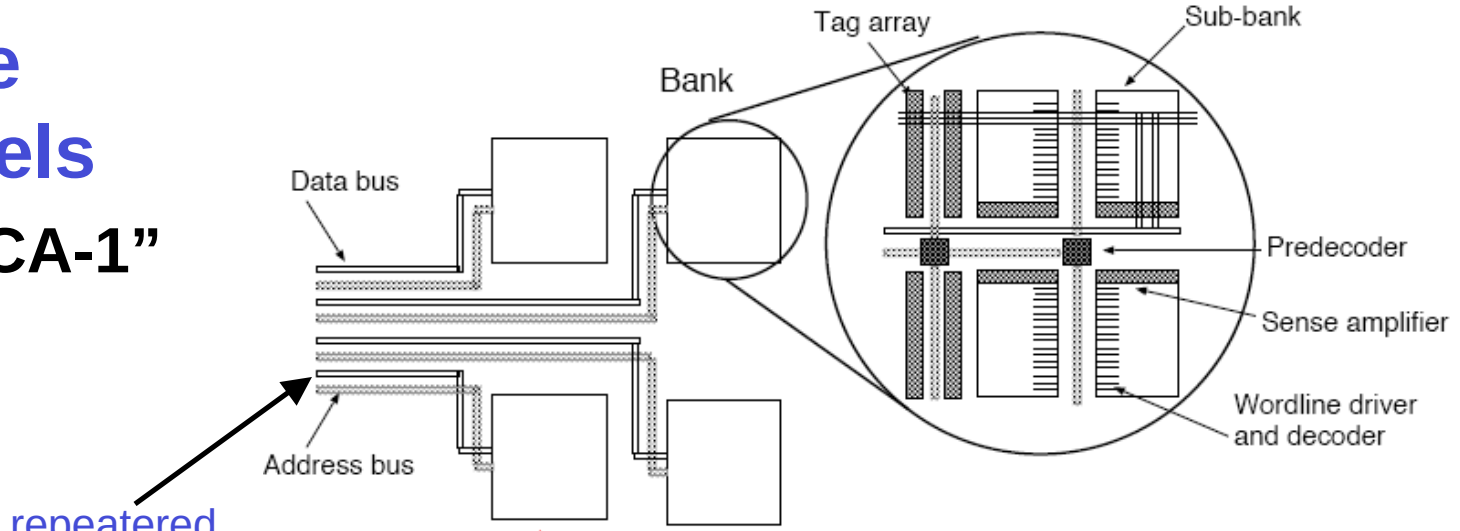
Non-Uniform Cache Architectures

- **Conventional cache: cache access time is set according to the worst case access latency**
  - ↳ farthest cache bank from controller
- **NUCA: cache access time varies depending on distance of the bank from the processor**

- ***Static* NUCA (S-NUCA)**
  - **Delay-oblivious content management**
    - we know about NUCA, but we're very naive/dumb
- ***Dynamic* NUCA (D-NUCA)**
  - **Data with high reuse located in faster banks**
    - smart management of latency differences

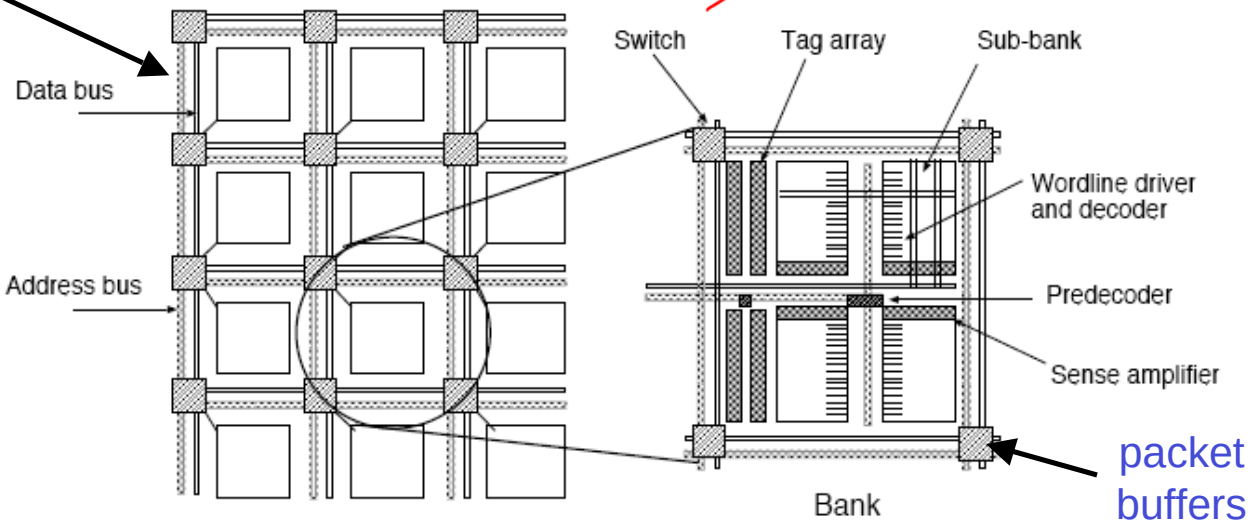
NUCA Physical Organizations

- **Private channels**
  - **"NUCA-1"**
- **Switched channels**
  - **"NUCA-2"**



repeated wires

⇒ wires that have repeaters



It's a router w/ buffers, etc. Helps resolve contention

packet buffers

[Kim02]

2 x 2 mesh network

NUCA Physical Organizations

- **Estimated access latencies**

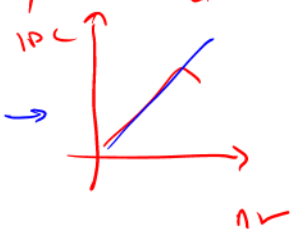
NUCA-1 private

Technology (nm)	L2 size	Num. banks	Unloaded latency bank	min	max	avg.	Conservative Loaded	Conservative IPC	Aggressive Loaded	Aggressive IPC
130	2MB	16	3	7	13	10	11.3	0.54	10.0	0.55
100	4MB	32	3	9	21	15	17.3	0.56	15.3	0.57
70	8MB	32	5	12	26	19	21.9	0.61	19.3	0.63
50	16MB	32	8	17	41	29	34.2	0.59	30.2	0.62

closest, farthest, one request, no contention, no pipeling on the wires, pipelining on the wires, terrible due to wire latency

NUCA-2 switched

Technology (nm)	L2 Size	Num. Banks	Unloaded Latency bank	min	max	avg.	Loaded Latency	IPC
130	2MB	16	3	4	11	8	9.7	0.55
100	4MB	32	3	4	15	10	11.9	0.58
70	8MB	32	5	6	29	18	20.6	0.62
50	16MB	32	8	9	32	21	24.2	0.65

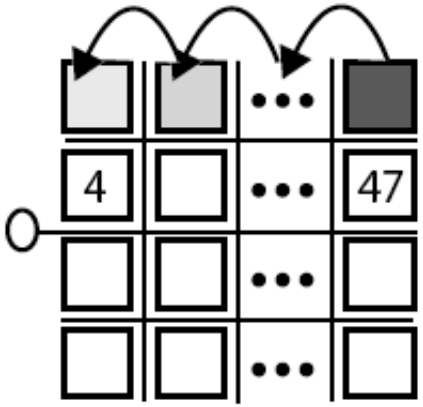


decreasing the # of banks improves latency to the controller, but you lose due to latency in the now more complex sub banks

[Kim02]

Dynamic NUCA (D-NUCA)

- **Cache controller accounts for varying latency**
  - **Line placement**
  - **By migrating lines to different banks**

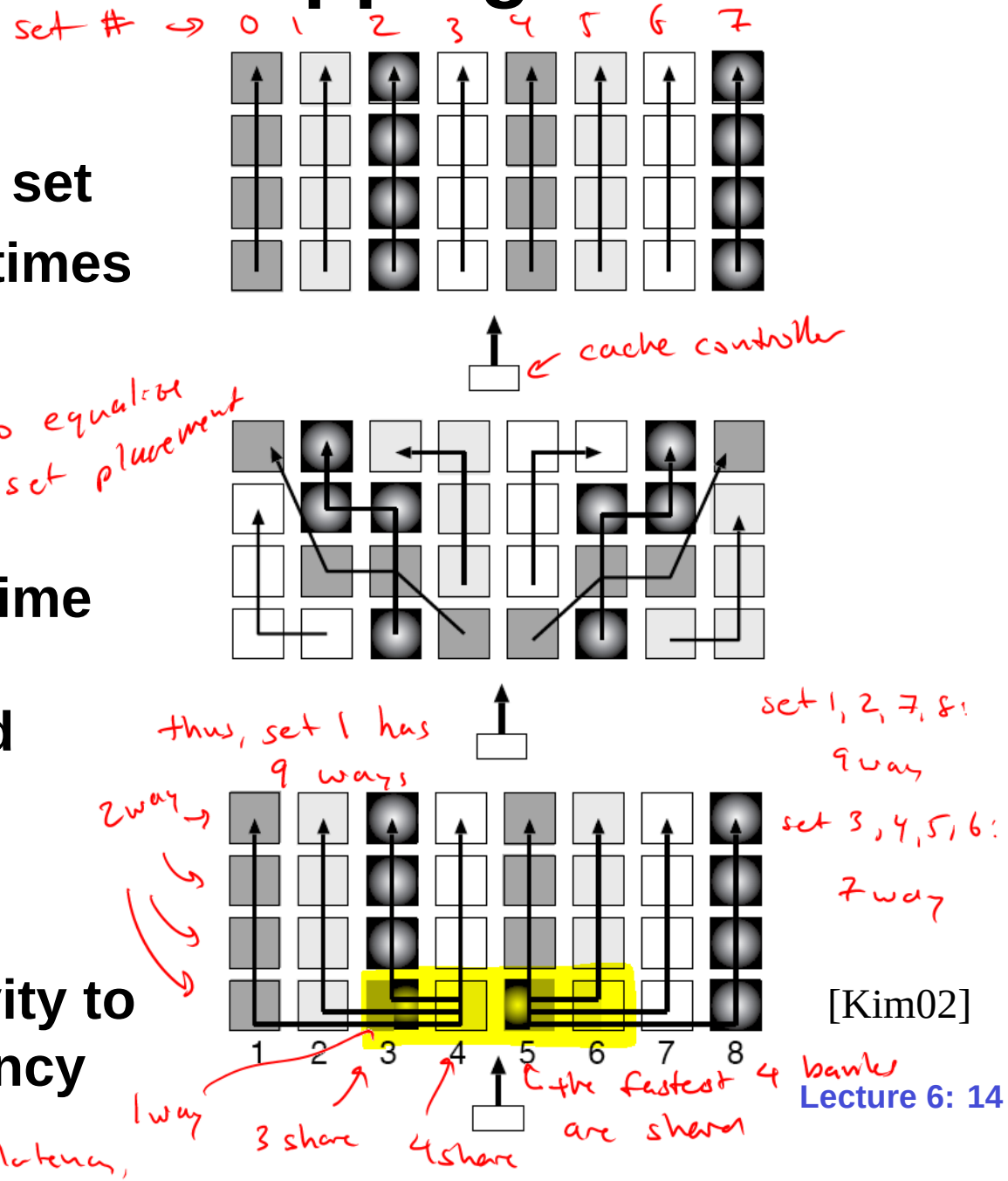


[Kim02]

- **Content management considerations**
  - ***Mapping*: Which banks a line can reside in and how the line is mapped to those banks**
  - ***Search*: How the set of possible locations are searched to find a line**
  - ***Movement*: Under what conditions lines are migrated from one bank to another**

D-NUCA Mapping

- **Simple**
  - **Each column is a set**
  - **Different access times among sets**
- **Fair**
  - **Average access time across all sets is roughly equalized**
- **Shared**
  - **Higher associativity to offset higher latency**



[Kim02]

D-NUCA Search

- **Incremental**
  - Banks are searched in order from closest to furthest
- **Multicast**
  - Address is sent to multiple banks in parallel
- **Smart search**
  - Based on partial tag comparison [Kessler89]
  - Partial tag array in cache controller
  - Hit if a match of partial tag and bank tag

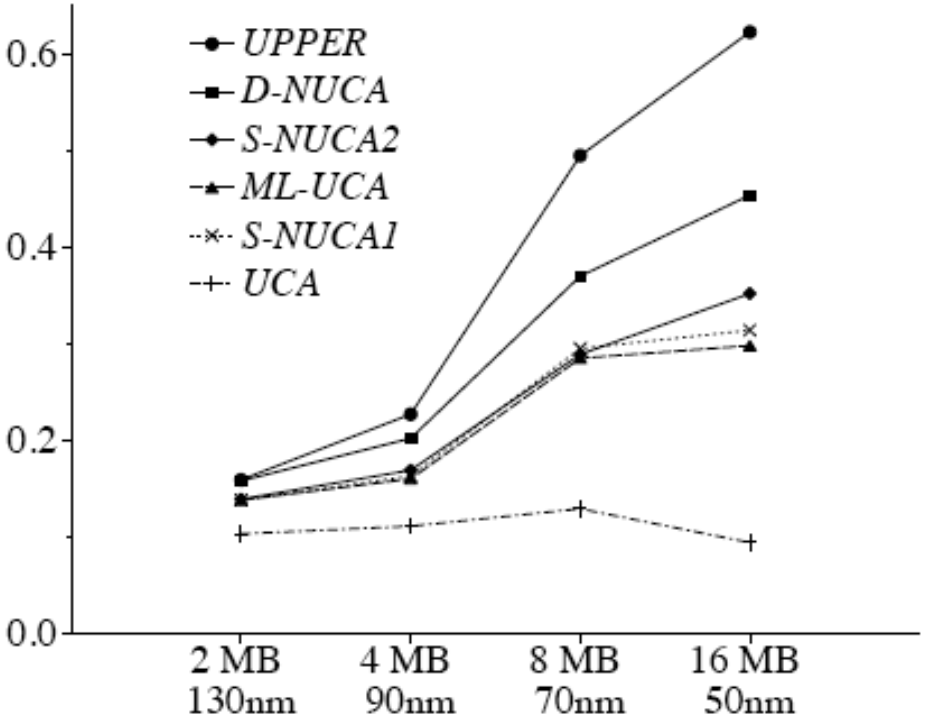
2 schemes
- Perform lookups in parallel
- Perform only first bank lookup in parallel

D-NUCA Line Movement

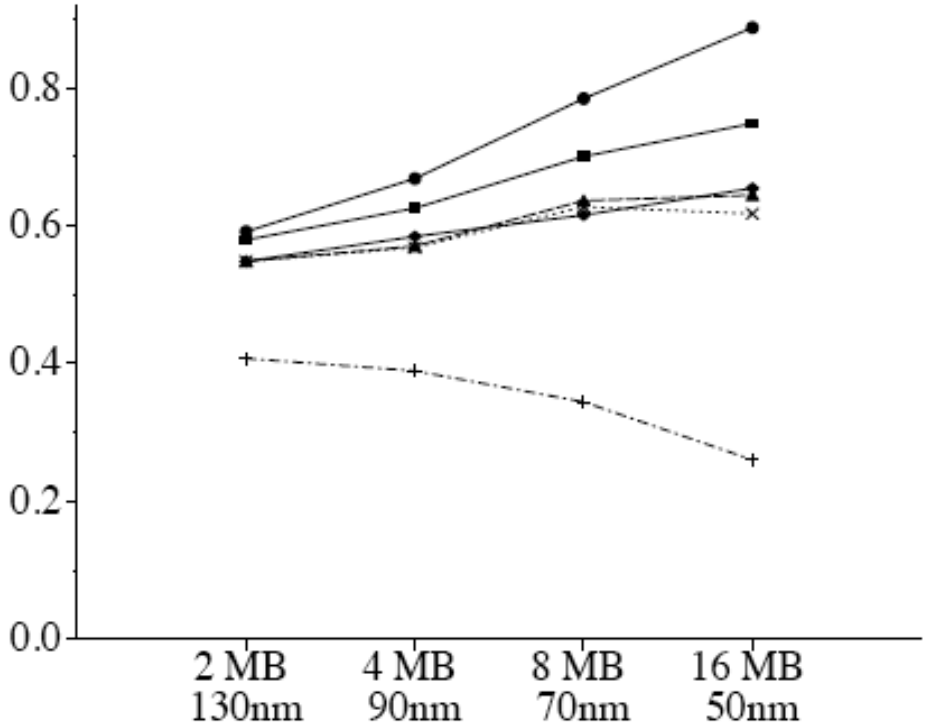
- **Placement on a miss** ↙ similar to set duelg
  - **Place closest, furthest away, or in-between?**
  - **What do we do with the victim?**
    - **Evict (*zero-copy policy*)**
    - **Move to a further bank (one-copy policy)**

- **Migration**
  - **After n hits to a line, swap the line with the one in the bank that is m positions closer to the cache controller**

IPC Comparison



mcf



harmonic mean of benchmarks

[Kim02]

Performance of D-NUCA Alternatives

Policy	Av. Lat.	IPC	Miss Rate	Bank Access	Policy	Av. lat.	IPC	Miss Rate	Bank Access
Search					*Promotion*				
Incremental	24.9	0.65	0.114	89M	1-bank/2-hit	18.5	0.71	0.115	259M
2 mcast + 14 inc	23.8	0.65	0.113	96M	2-bank/1-hit	17.7	0.71	0.114	266M
2 inc + 14 mcast	20.1	0.70	0.114	127M	2-bank/2-hit	18.3	0.71	0.115	259M
2 mcast + 14 mcast	19.1	0.71	0.113	134M	*Eviction*				
Mapping					insert head, evict random, 1 copy	15.5	0.70	0.117	267M
Fast shared	16.6	0.73	0.119	266M	insert middle, evict random, 1 copy	16.6	0.70	0.114	267M
Baseline: simple map, multicast, 1-bank/1-hit, insert at tail						18.3	0.71	0.114	266M

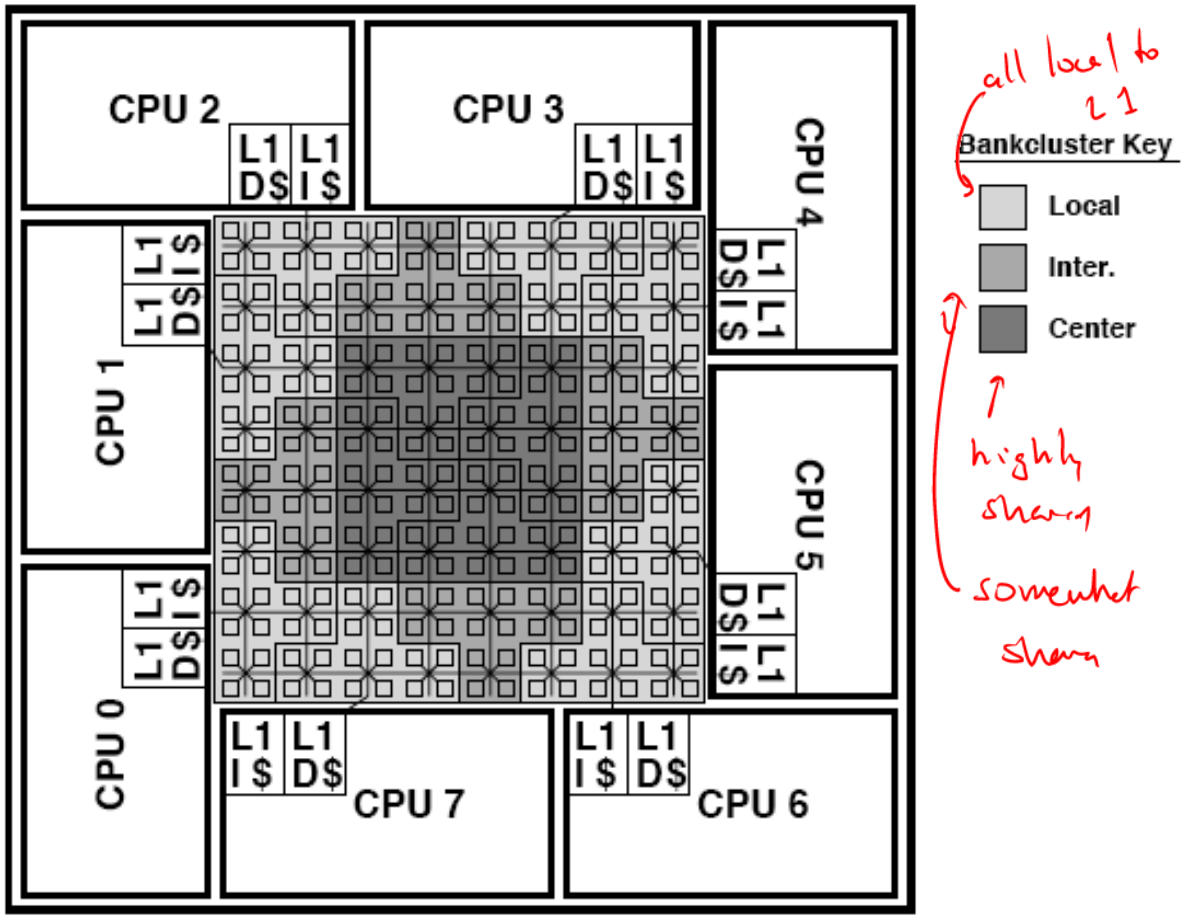
Configuration	Loaded Latency	Average IPC	Miss Rate	Bank Accesses	Tag Bits	Search Array
Base D-NUCA	18.3	0.71	0.113	266M	-	—
SS-performance	18.3	0.76	0.113	253M	7	224KB
SS-energy	20.8	0.74	0.113	40M	7	224KB
SS-performance + shared bank	16.6	0.77	0.119	266M	6	216KB
SS-energy + shared bank	19.2	0.75	0.119	47M	6	216KB
Upper bound	3.0	0.83	0.114	—	-	—
Upper bound + SS-performance	3.0	0.89	0.114	—	7	224KB

[Kim02]

CMP-DNUCA

[Beckmann04]

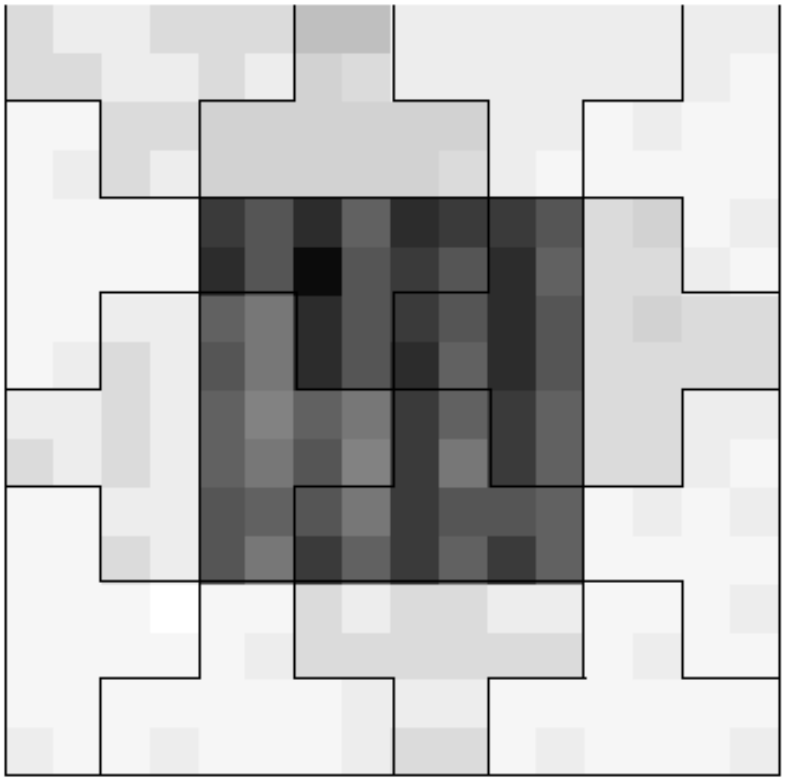
- D-NUCA L2 cache for Chip Multiprocessors
- Where do we store shared data in L2?
- Migrate lines gradually among bankclusters in single steps



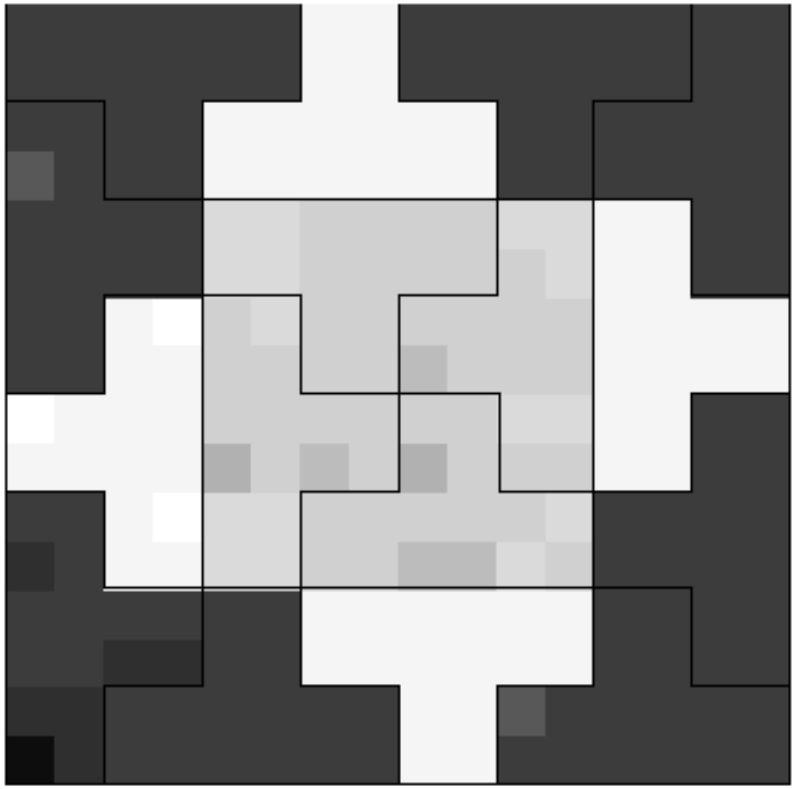
other local ⇒ other inter ⇒ other center ⇒ my center ⇒ my inter ⇒ my local

CMP-DNUCA Hit Distribution

high sharing workload



low sharing workload



dark is high hit count

[Beckmann04]

Off-line Partitioning Heuristics

- **Programmer or compiler partitions code and data between the scratchpad and main memory**

Example Embedded Processor Code

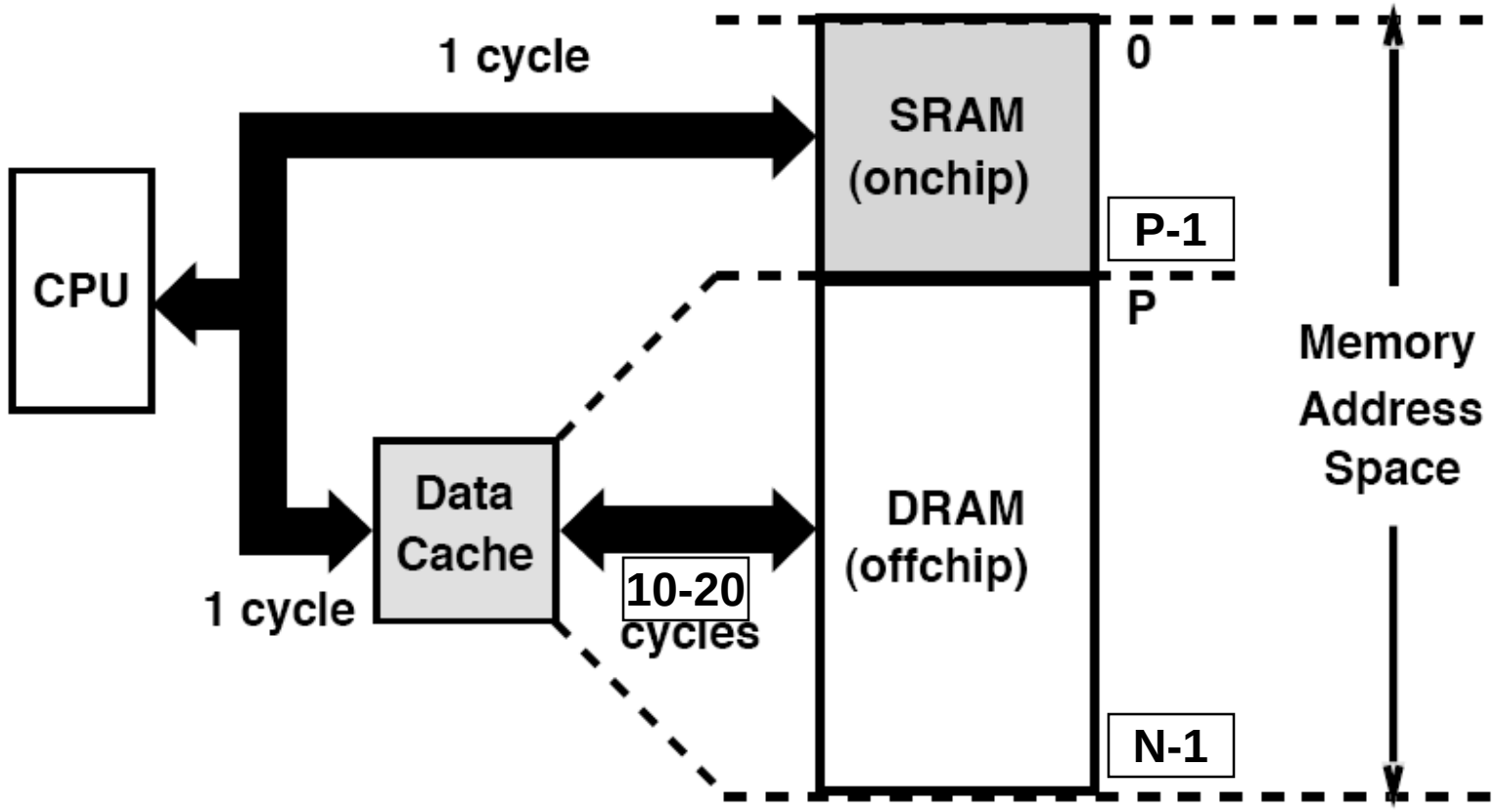
```
/************************* SDRAM TEST *************************/
/*                                                            */
/*  BY: J S Smith                                             */
/*                                                            */
/*  Write 0000FFFF to SDRAM from internal cache               */
/*                                                            */
/*  This code is written for an Analog Devices TS-101         */
/*  This chip has 3 internal caches(1 for program and 2 for data */
/*                                                            */
/*  This will fill data cache 1 will a striding pattern.      */
/*  It will copy this strided pattern to the same data cache 1,  */
/*  to the other data cache 2, to the program cache and finally  */
/*  to the external SDRAM.                                    */
/*                                                            */
/************************************************************/

#define SIZE 512

section  ("data1")     float data_orig[SIZE];
section  ("SDRAM")     float data_external[SZE];
section  ("data1")     float data_int_1[SIZE];
section  ("data2")     float data_int_2[SIZE];
section  ("program")   float data_int_p[SIZE];
```

allows you to partition things

```
int main()
{
        int i;

        /* Fill the original array with the stride pattern. */
        for(i = 0; i < SIZE; i++)
              data_orig[i] = 0x0000FFFF;

        /* Copy from data cache 1 to data cache 1 */
        for(i = 0; i < SIZE; i++)
              data_int_1[i] = data_orig[i];

        /* Copy from data cache 1 to data cache 2 */
        for(i = 0; i < SIZE; i++)
              data_int_2[i] = data_orig[i];

        /* Copy from data cache 1 to program cache */
        for(i = 0; i < SIZE; i++)
               data_int_p[i] = data_orig[i];

        /* Copy from data cache 1 to SDRAM */
        for(i = 0; i < SIZE; i++)
              data_external[i] = data_orig[i];

        return 0;
}
```

Example Embedded Processor

- SPM + d-cache backed by main memory



[Panda00]

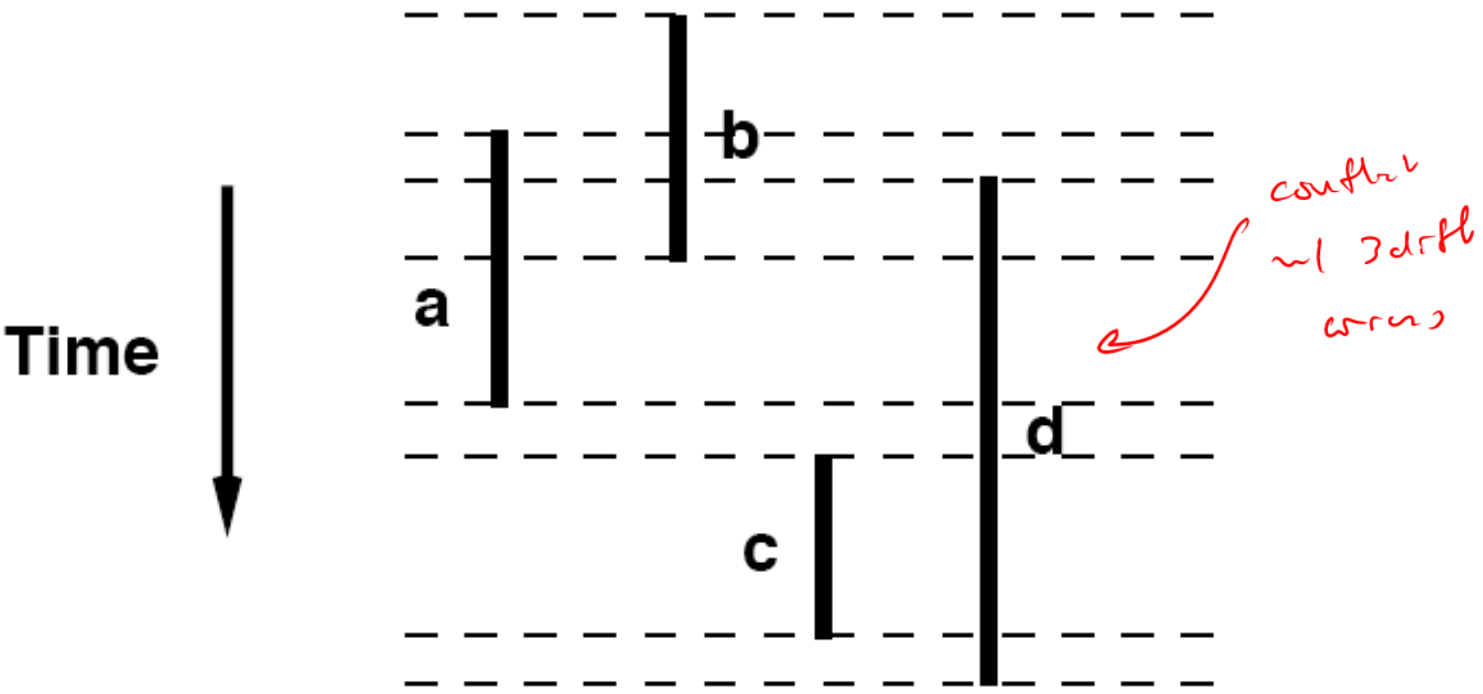
correction of paper figure

Partitioning Factors

- **Where to map scalar variables and constants**
  - **Usually take up little space so map to SPM to avoid d-cache conflicts with arrays**
- **Size of arrays**
  - **Arrays larger than SPM requires book-keeping code to determine which region of the array is addressed**
  - **Assign these large arrays to d-cache**

Partitioning Factors

- **Lifetimes of variables**
  - **Use lifetime analysis to store variables/arrays with disjoint lifetimes in same storage location**



[Panda00]

Partitioning Factors

- **Access frequency of variables**
  - **Use to estimate degree of conflicts with other variables**
    - **Rough metric: *Interference factor***

$$IF(u) = VAC(u) + IAC(u)$$

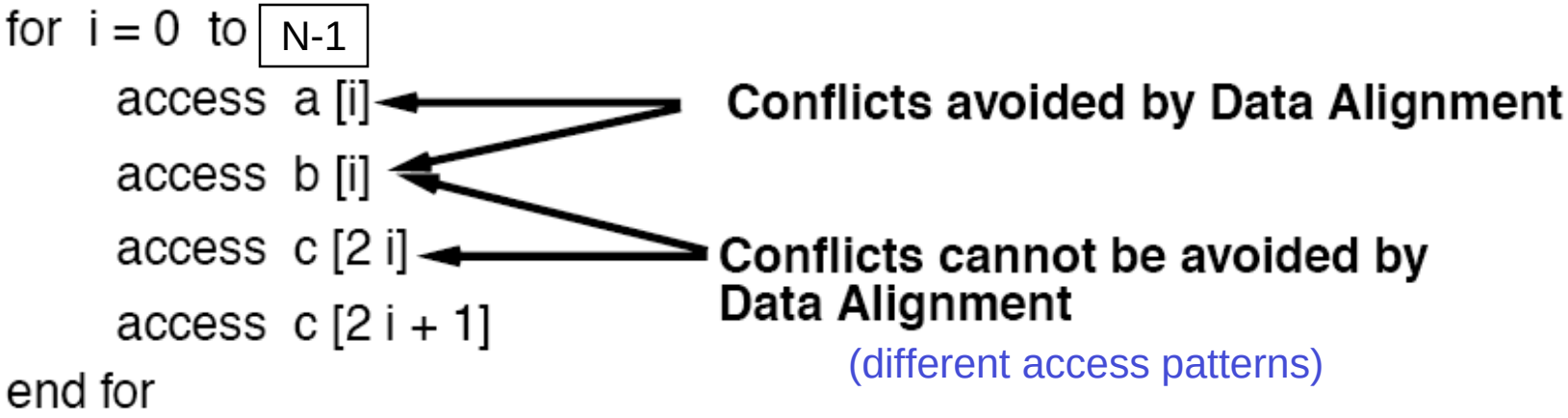
number of accesses to elements of u during its lifetime

number of accesses to other variables during the lifetime of u

    - **High value of IF(u) indicates u is likely to have large number of d-cache conflicts if mapped to DRAM**
      - **Map instead to SPM**

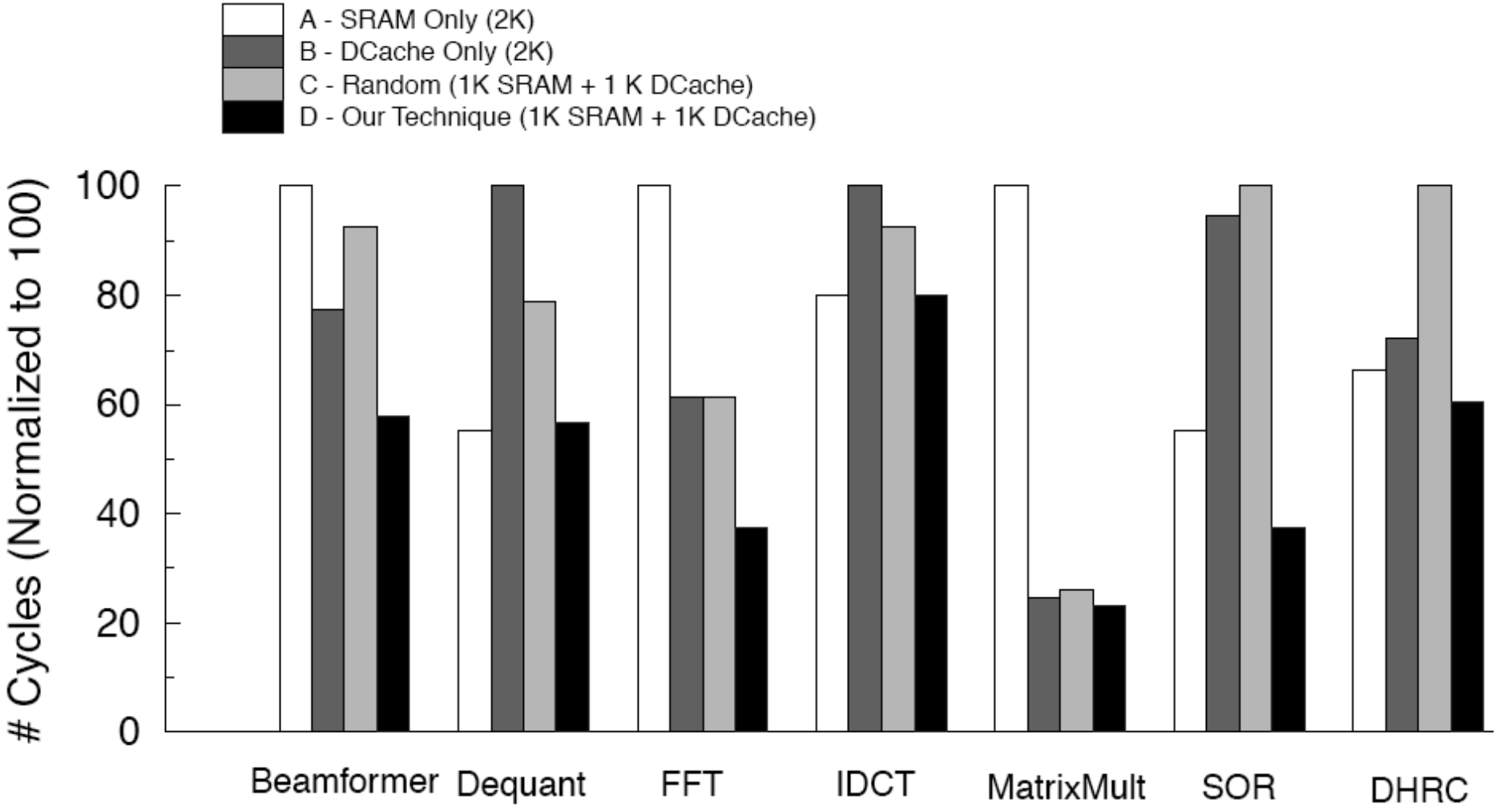
Partitioning Factors

- **Conflicts in loops**
  - **Identify array d-cache conflicts in loops that *cannot* be avoided by memory address assignment**



  - **Map a and b to DRAM and c to SPM**

Performance Comparison



[Panda00]

Next Time

Cache Content Management