Cache (Memory) Performance Optimization

Improving Cache Performance

Average memory access time =
Hit time + Miss rate x Miss penalty

To improve performance:

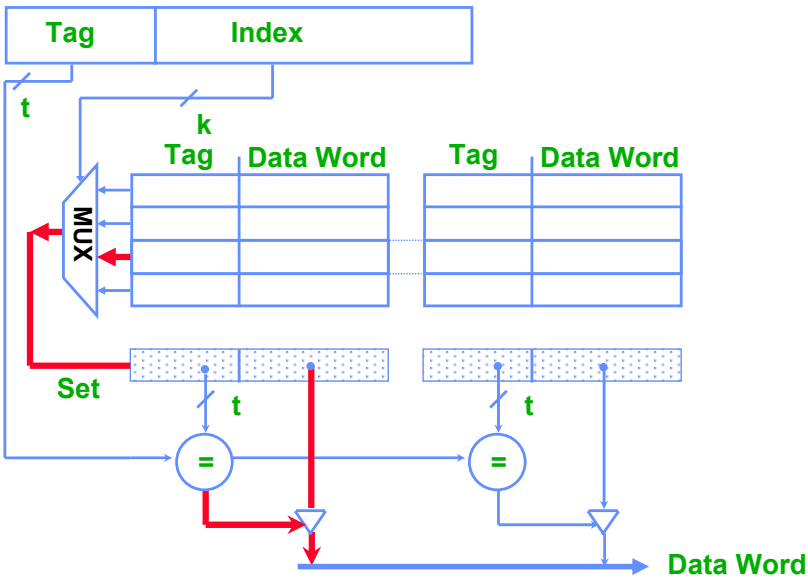
- **reduce the miss rate (e.g., larger cache)**
- **reduce the miss penalty (e.g., L2 cache)**
- **reduce the hit time**

The simplest design strategy is to design the largest primary cache without slowing down the clock or adding pipeline stages

2

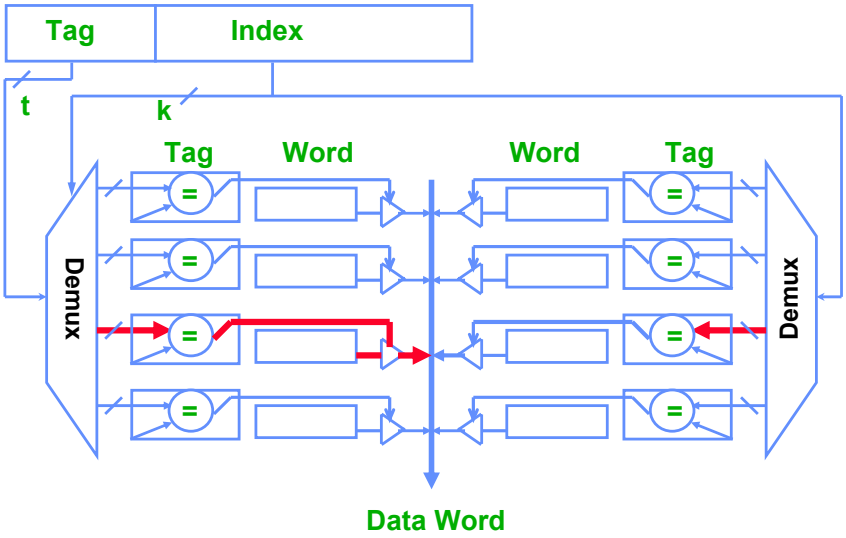
Design the largest primary cache without slowing down the clock
Or adding pipeline stages.

RAM-tag 2-way Set-Associative Cache



3

CAM-tag 2-way Set-Associative Cache



4

Causes for Cache Misses

- ***Compulsory:*** **first-reference to a block *a.k.a.* cold start misses**
  - **misses that would occur even with infinite cache**
- ***Capacity:*** **cache is too small to hold all data needed by the program**
  - **misses that would occur even under perfect placement & replacement policy**
- ***Conflict:*** **misses that occur because of collisions due to block-placement strategy**
  - **misses that would not occur with full associativity**

5

Block-level Optimizations

- **Tags are too large, i.e., too much overhead**
  - **Simple solution: Larger blocks, but miss penalty could be large.**
- **Sub-block placement**
  - **A valid bit added to units smaller than the full block, called sub-blocks**
  - **Only read a sub-block on a miss**
  - ***If a tag matches, is the word in the cache?***

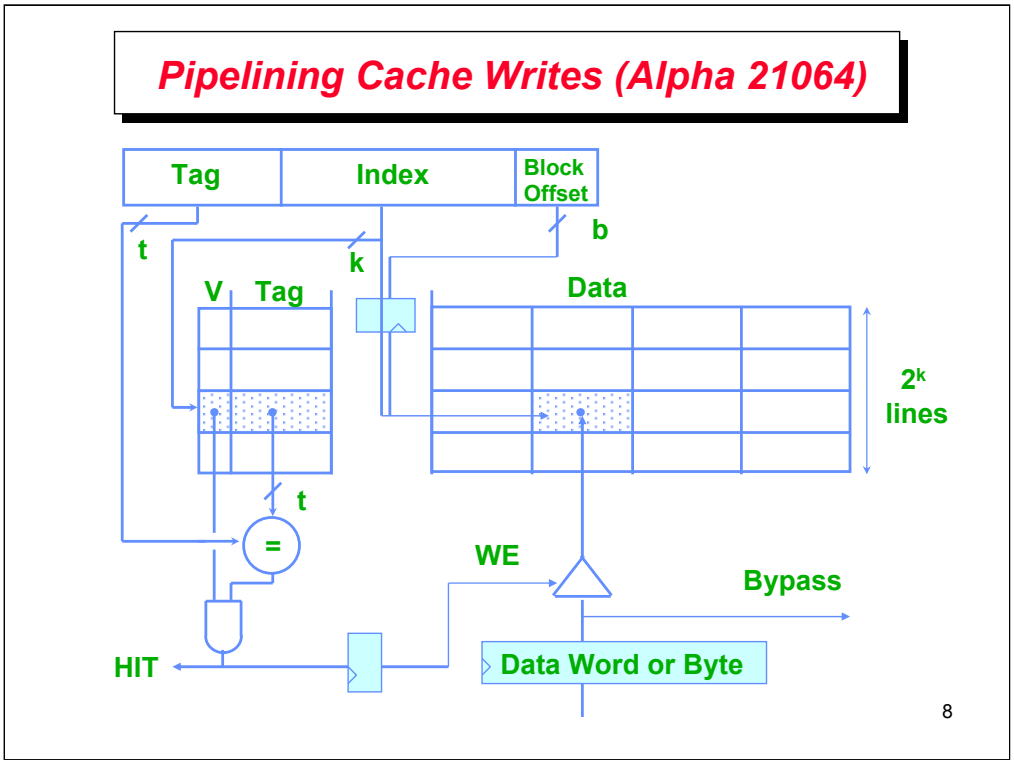
100	*1*		*1*		*1*		*1*	
300	*1*		*1*		*0*		*0*	
204	*0*		*1*		*0*		*1*	

6

Main reason for subblock placement is to reduce tag overhead.

Write Alternatives

- Writes take two cycles in memory stage, one cycle for tag check plus one cycle for data write if hit
- Design data RAM that can perform read *and* write in one cycle, restore old value after tag miss
- Hold write data for store in single buffer ahead of cache, write cache data during next store's tag check
  - Need to bypass from write buffer if read matches write buffer tag



Writes usually take longer than reads because the tags have to
Be checked before writing the data.
First, tags and data are split, so they can be addressed independently.

Prefetching

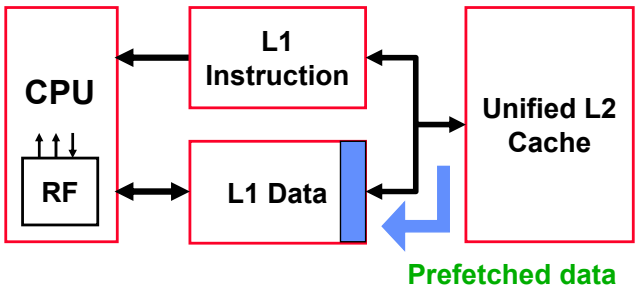
- **Speculate on future instruction and data accesses and fetch them into cache(s)**
  - **Instruction accesses easier to predict than data accesses**
- **Varieties of prefetching**
  - **Hardware prefetching**
  - **Software prefetching**
  - **Mixed schemes**
- ***What types of misses does prefetching affect?***

9

Reduces compulsory misses, can increase conflict and capacity misses.

Issues in Prefetching

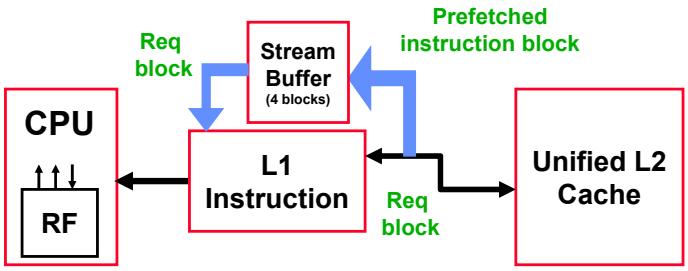
- Usefulness – should produce hits
- Timeliness – not late and not too early
- Cache and bandwidth pollution



10

Hardware Prefetching of Instructions

- **Instruction prefetch in Alpha AXP 21064**
  - Fetch two blocks on a miss; the requested block and the next consecutive block
  - Requested block placed in cache, and next block in instruction stream buffer



11

Need to check the stream buffer if the requested block is in there.

Never more than one 32-byte block in the stream buffer.

Hardware Data Prefetching

- **One Block Lookahead (OBL) scheme**
  - Initiate prefetch for block **b + 1** when block **b** is accessed
  - ***Why is this different from doubling block size?***
- **Prefetch-on-miss:**
  - Prefetch **b + 1** upon miss on **b**
- **Tagged prefetch:**
  - Tag bit for each memory block
  - Tag bit = 0 signifies that a block is demand fetched or if a prefetched block is referenced for the first time
  - Prefetch for **b + 1** initiated only if tag bit = 0 on **b**

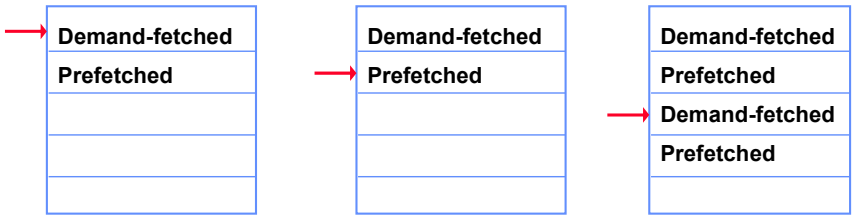
12

HP PA 7200 uses OBL prefetching

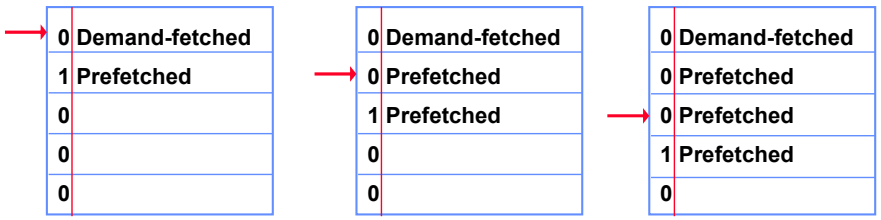
Tag prefetching is twice as effective as prefetch-on-miss in reducing miss rates.

Comparison of Variant OBL Schemes

Prefetch-on-miss accessing contiguous blocks



Tagged prefetch accessing contiguous blocks



13

Software Prefetching

```
for(i=0; i < N; i++) {
    fetch( &a[i + 1] );
    fetch( &b[i + 1] );
    SUM = SUM + a[i] * b[i];
}
```

- ***What property do we require of the cache for prefetching to work ?***

14

Cache should be non-blocking or lockup-free.

By that we mean that the processor can proceed while the prefetched Data is being fetched; and the caches continue to supply instructions And data while waiting for the prefetched data to return.

Software Prefetching Issues

- **Timing is the biggest issue, not predictability**
  - If you prefetch very close to when the data is required, you might be too late
  - Prefetch too early, cause pollution
  - Estimate how long it will take for the data to come into L1, so we can set **P** appropriately
  - ***Why is this hard to do?***

```
for(i=0; i < N; i++) {
    fetch( &a[i + P] );
    fetch( &b[i + P] );
    SUM = SUM + a[i] * b[i];
}
```

Don't ignore cost of prefetch instructions

15

Compiler Optimizations

- **Restructuring code affects the data block access sequence**
  - Group data accesses together to improve spatial locality
  - Re order data accesses to improve temporal locality
- **Prevent data from entering the cache**
  - Useful for variables that are only accessed once
- **Kill data that will never be used**
  - Streaming data exploits spatial locality but not temporal locality

16

Miss-rate reduction without any hardware changes.

Hardware designer's favorite solution.

Loop Interchange

```
for(j=0; j < N; j++) {
    for(i=0; i < M; i++) {
        x[i][j] = 2 * x[i][j];
    }
}
```

```
for(i=0; i < M; i++) {
    for(j=0; j < N; j++) {
        x[i][j] = 2 * x[i][j];
    }
}
```

What type of locality does this improve?

17

Loop Fusion

```
for(i=0; i < N; i++)
    for(j=0; j < M; j++)
        a[i][j] = b[i][j] * c[i][j];

for(i=0; i < N; i++)
    for(j=0; j < M; j++)
        d[i][j] = a[i][j] * c[i][j];
```

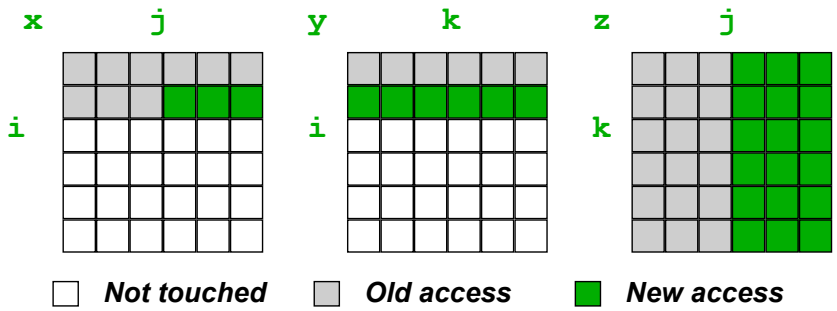
```
for(i=0; i < M; i++)
    for(j=0; j < N; j++) {
        a[i][j] = b[i][j] * c[i][j];
        d[i][j] = a[i][j] * c[i][j];
    }
```

What type of locality does this improve?

18

Blocking

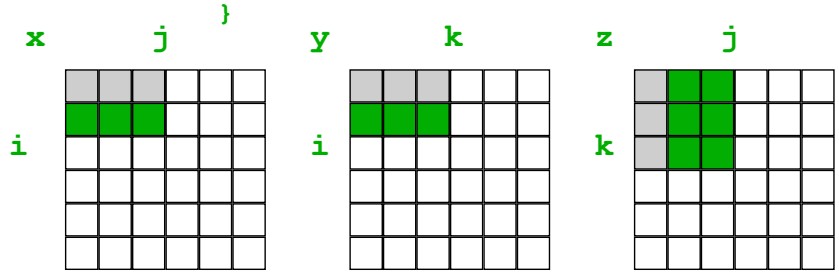
```
for(i=0; i < N; i++)
     for(j=0; j < N; j++) {
        r = 0;
        for(k=0; k < N; k++)
          r = r + y[i][k] * z[k][j];
        x[i][j] = r;
     }
```



19

Blocking

```
for(jj=0; jj < N; jj=jj+B)
    for(kk=0; kk < N; kk=kk+B)
       for(i=0; i < N; i++)
           for(j=jj; j < min(jj+B,N); j++) {
              r = 0;
              for(k=kk; k < min(kk+B,N); k++)
                 r = r + y[i][k] * z[k][j];
              x[i][j] = x[i][j] + r;
           }
```



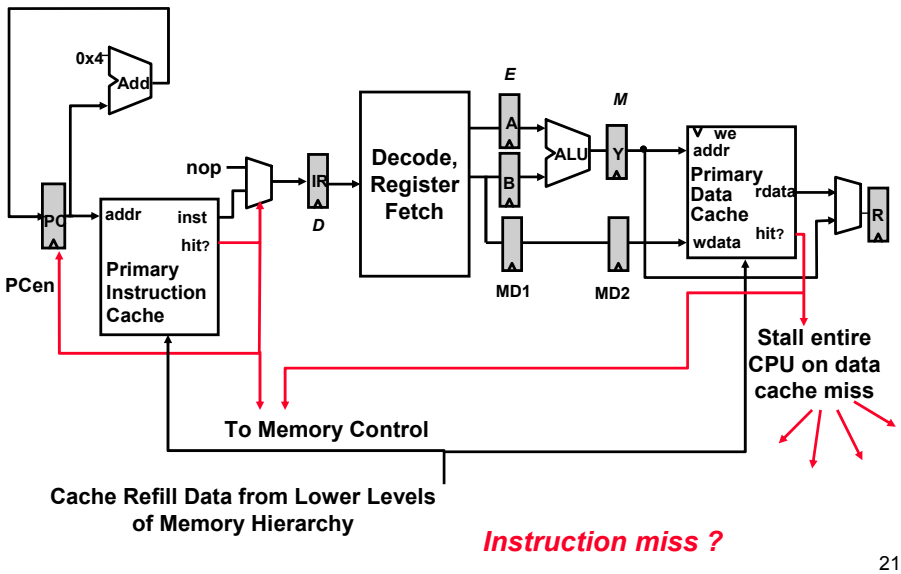
What type of locality does this improve?

20

Y benefits from spatial locality

z benefits from temporal locality

CPU-Cache Interaction (5-stage pipeline)



Instruction miss ?

21

Memory (DRAM) Performance

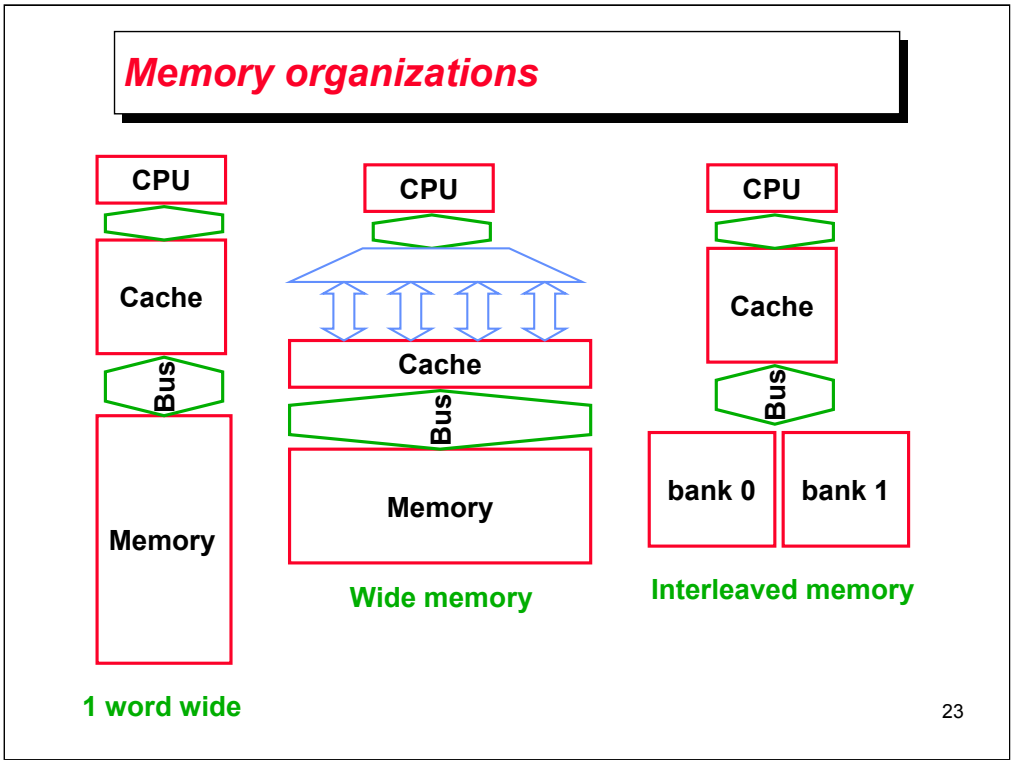
1 Gb DRAM

50-100 ns access time
Needs refreshing

- **Upon a cache miss**
  - **4 clocks to send the address**
  - **24 clocks for the access time per word**
  - **4 clocks to send a word of data**
- **Latency worsens with increasing block size**

22

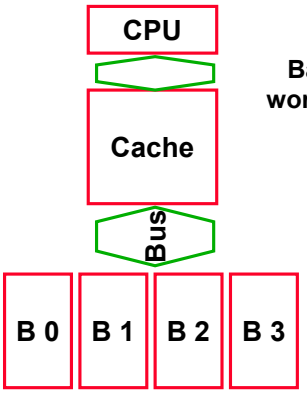
Need 128 or 116 clocks, 128 for a dumb memory.



Alpha AXP 21064 256 bits wide memory and cache.

Interleaved Memory

- **Banks are often 1 word wide**
- **Send an address to all the banks**
- ***How long to get 4 words back?***



Bank i contains all words whose address modulo N is i

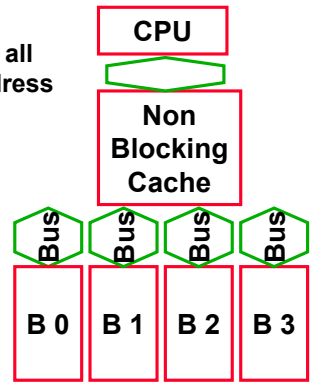
24

4 + 24 + 4* 4 clocks = 44 clocks from interleaved memory.

Independent Memory

- **Send an address to all the banks**
- ***How long to get 4 words back?***

Bank i contains all words whose address modulo N is i



25

4 + 24 + 4 = 32 clocks from main memory for 4 words.

Memory Bank Conflicts

Consider a 128-bank memory in the NEC SX/3 where each bank can service independent requests

```
int x[256][512];
   for(j=0; j < 512; j++)
      for(i=0; i < 256; i++)
         x[i][j] = 2 * x[i][j];
```

Consider column `k` `x[i][k], 0 <= i < 256`

Address of elements in column is `i*512 + k`

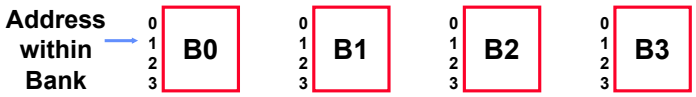
Where do these addresses go?

26

Bank Assignment

Bank number = Address MOD NumBanks

Address within bank = $\left\lfloor \frac{\text{Address}}{\text{NumBanks}} \right\rfloor$



Should randomize address to bank mapping.

Could use odd/prime number of banks. ***Problem?***

Bank number for `i*512 + k` **should depend on** `i`
Can pick more significant bits in address

27