



# Prefetch Distance

- How early prefetch?
  - One iteration is enough?
  - Memory latency and amount of computation between memory accesses
  - Prefetch distance ( $\delta$ )

$$\delta = \left\lceil \frac{l}{s} \right\rceil$$

- $l$ : the average memory latency (measured in cycle)
- $s$ : the estimated cycle time of the shortest possible execution path through one loop iteration



# Example: Prefetch Distance

```
for( i =0 ; i < N-4; i+=4) {  
  fetch (&a[i+4]);  
  fetch (&b[i+4]);  
  ip = ip+a[i]*b[i];  
  ip = ip+a[i+1]*b[i+1];  
  ip = ip+a[i+2]*b[i+2];  
  ip = ip+a[i+3]*b[i+3];  
}
```

← L →

Loading an element

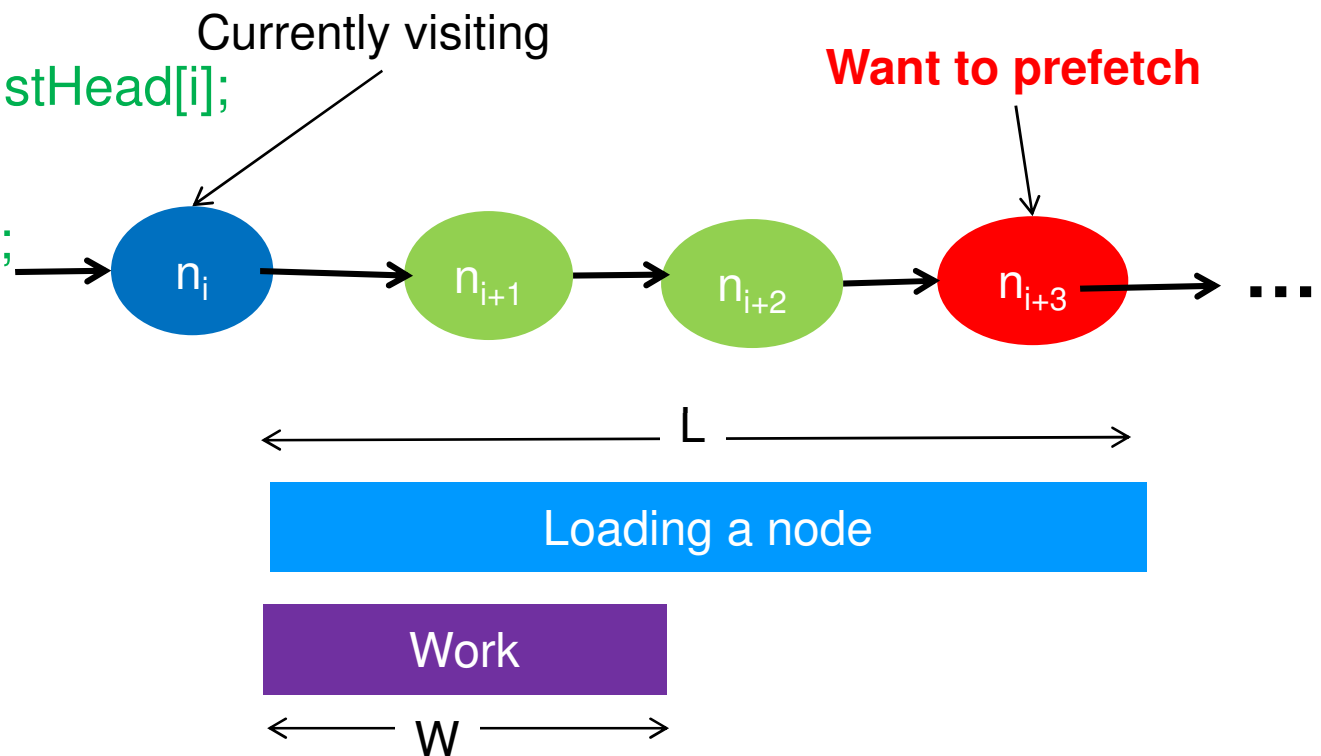
Computation

- If  $L/C = 4$ 
  - We must prefetch 4 elements ahead
- Problem: Memory latency vary at run-time
- Why? And so?



# Linked List

```
for (i=1; i<N; i++) {  
  listNode *p = listHead[i];  
  while(p){  
    work(p->data);  
    p=p->next;  
  }  
}
```

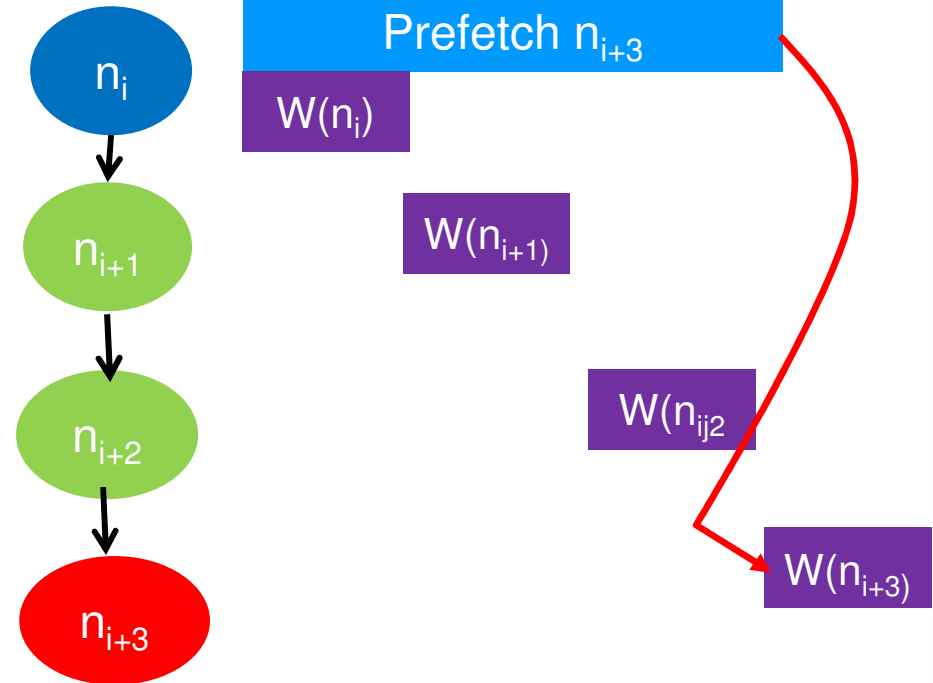


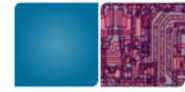
If  $L=3W$ ?



# SW Prefeetch Linked List

```
for (i=1; i<N; i++) {  
  listNode *p = listHead[i];  
  while(p){  
    prefetch ((p->next->next->  
              >next)  
    work(p->data);  
    p=p->next;  
  }  
}
```





# Prefetch Metrics

- # of useful prefetch: # of prefetched block that will be used by demand loads
- **Accuracy** = # of useful prefetch/total # of prefetch
- **Coverage** = # of useful prefetch/total # of cache misses
- **Timeliness**: How timely prefetch cache blocks

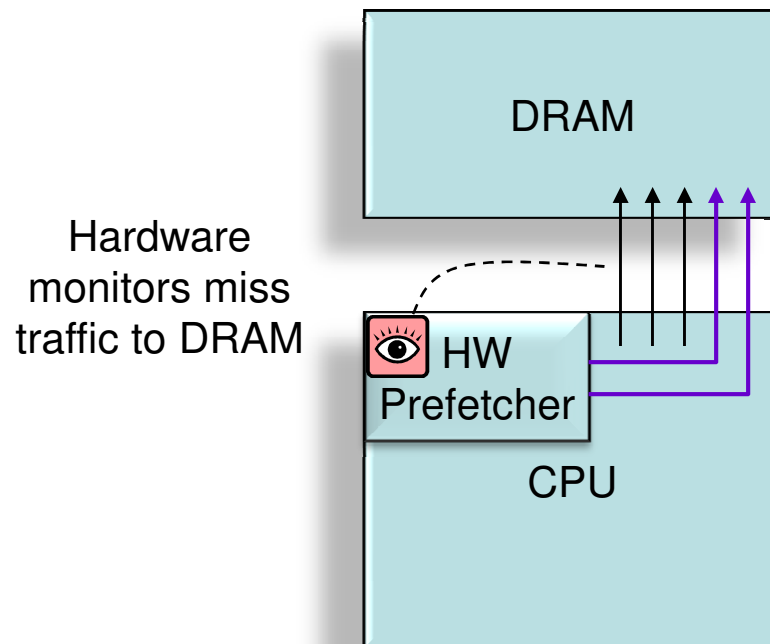


# Limitations of Software Prefetching

- Compiler or programmer need to insert
  - Usually limit to loops
- Prefetch instruction fetch/execution overhead
- Code expansion
- Static decision: Cache miss behavior needs to be predicted at static time
  - Cache sizes vary machine by machine
  - Today's processors; cores share caches.

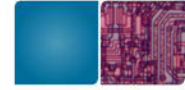


# Hardware Prefetching



Depending on prefetch algorithm/miss patterns, prefetcher injects additional memory requests

Cannot be overly aggressive since prefetches may contend for memory bandwidth, and may pollute the cache (evict other useful cache lines)



# Hardware prefetch schemes

- Hardware prefetch address =  
= **func**(demand memory addresses, pc, memory value, old memory address histories, etc..)

Different prefetchers have different algorithms

looking at only demand memory addresses? : stream, stride

looking at PC addresses or not

looking at memory values? Content based prefetching

old memory address histories? Markov prefetching



# Stream/Stride prefetcher

- Miss address streams
  - 1, 2, 3, 4 .....
  - Prefetch 5, 6, 7
  - Stream prefetch
- Miss address streams
  - 1,4,7,10,....
  - Prefetch 13,16,19,....
  - Stride prefetch



# Instruction Prefetching

- Instructions are sequential.
- Easy to predict.
- First implemented
- Next line prefetcher ( one block ahead prefetcher)
  - Very simple, if a request for cache line  $X$  goes to DRAM, also request  $X+1$ 
    - FPM DRAM already will have the correct page open for the request for  $X$ , so  $X+1$  will likely be available in the row buffer
    - Can optimize by doing Next-Line-Unless-Crossing-A-Page-Boundary prefetching



# Stream Buffer

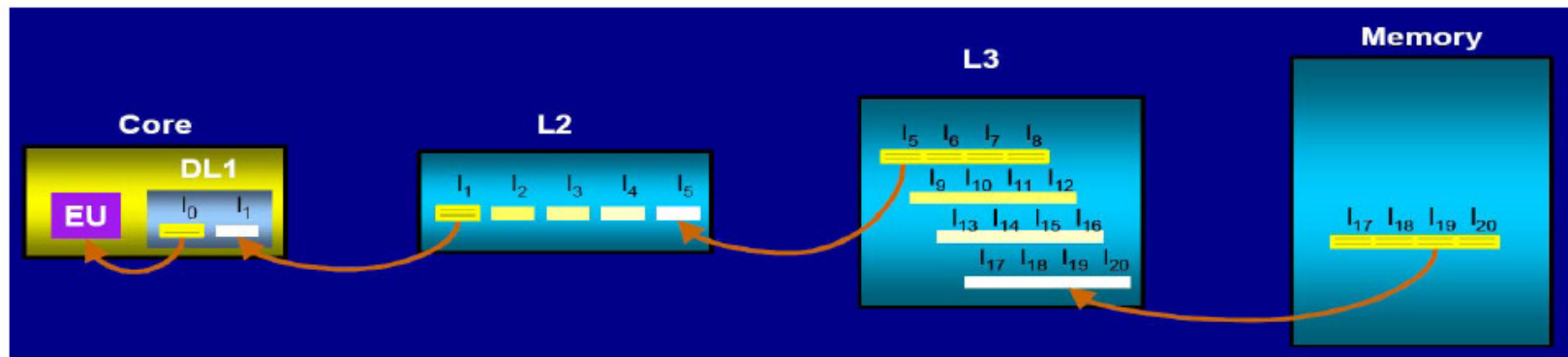
- *Jouppi '90*
- $K$  prefetched blocks → **FIFO stream buffer**
- As each buffer entry is referenced
  - Move it to cache
  - Prefetch a new block to stream buffer
  - Prefetcher buffer hit! → prefetch the next block
- Avoid cache pollution



# Prefetching Aggressive

- Degree of prefetching
  - For one cache miss, how many do we prefetch?
  - E.g.) addr 0x01: → 0x03, 0x04, 0x05, 0x06
- Prefetch distance
  - How far do we prefetch?

# Stream Prefetcher in multi-level cache hierarchy



POWER4 Hardware data prefetch schematic

Different prefetch degree for different memory hierarchy  
Initial distance to hide memory latency

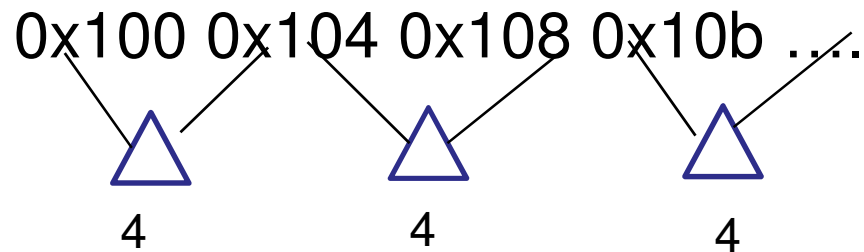


# Stride Prefetcher

Source code level

```
for ( ii = 0; ii < N; ii++)  
Sum +=b[ii];
```

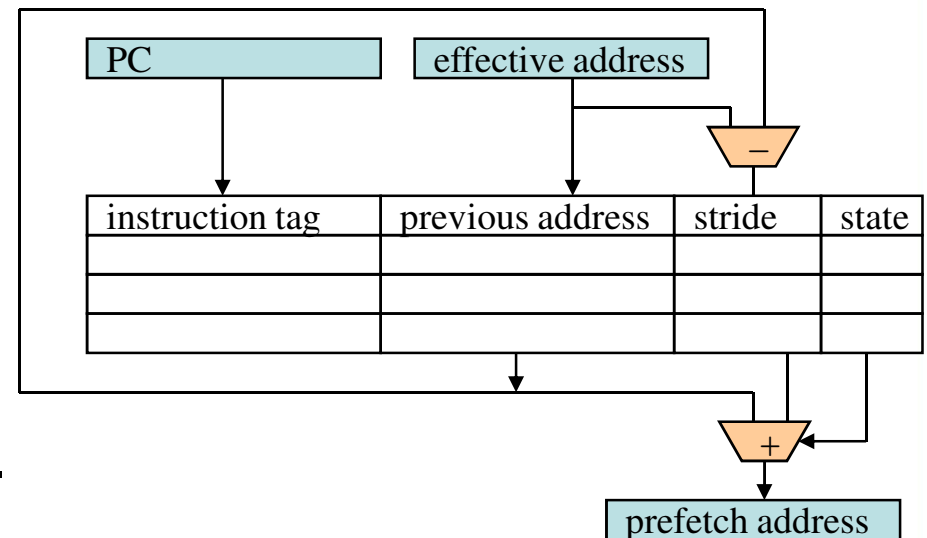
Memory addresses



Per PC information

Chen-Baer '91

Organization of RPT





# Is good to use PC?

- PC information can easily differentiate different address streams
- How soon can we know PC addresses?



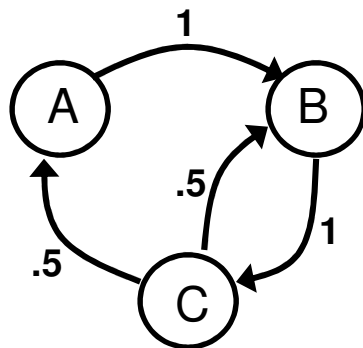
# Markov Prefetching

- Markov prefetching forms address correlations
  - Joseph and Grunwald (ISCA '97)
- Uses global memory addresses as states in the Markov graph
- **Correlation Table** *approximates* Markov graph

## Miss Address Stream

A B C A B C B C ...

## Markov Graph



## Correlation Table

1st predict. 2nd predict.

| miss address | 1st predict. | 2nd predict. |
|--------------|--------------|--------------|
| A            | B            |              |
| B            | C            |              |
| C            | B            | A            |

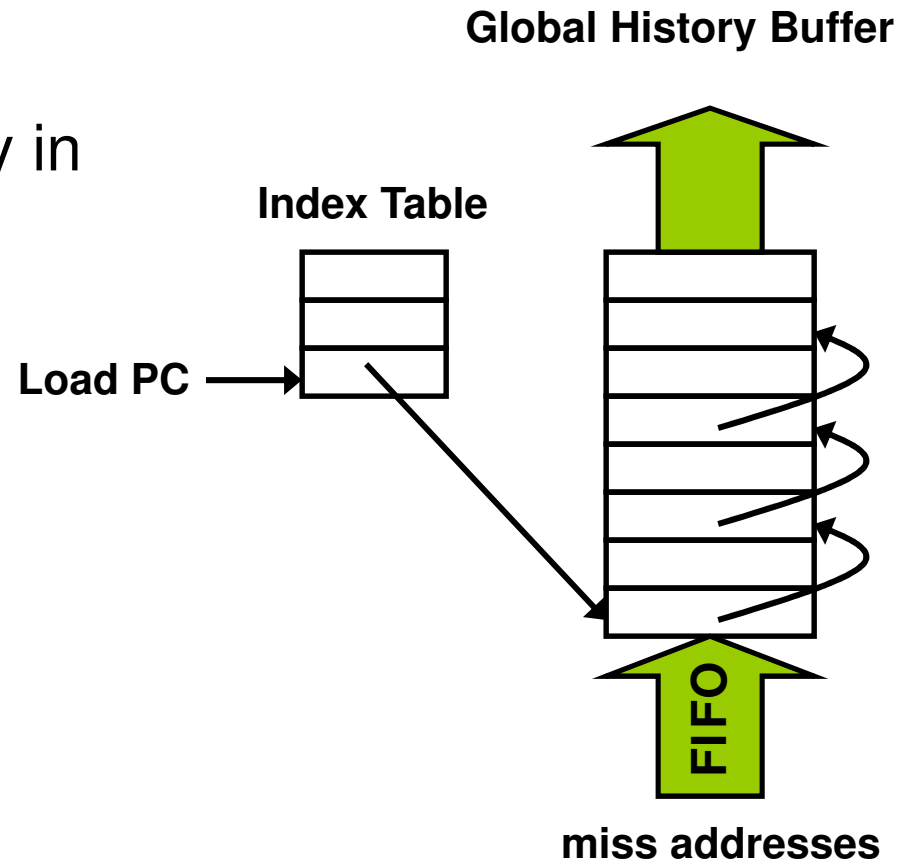


# Markov Prefetching

- History based prefetching
  - → required space for storing history
  - How much space?
  - Is it still good with a large L2 cache?
- What kind of data structures are good for this type?
  - Pointer, link list

# Global History Buffer (GHB)

- Unified frame for different prefetching scheme
- Holds miss address history in FIFO order
- Linked lists within GHB connect related addresses
  - Same static load
  - Same global miss address
  - Same global delta
- Linked list walk is short compared with L2 miss latency
- Nesbit and smith '04

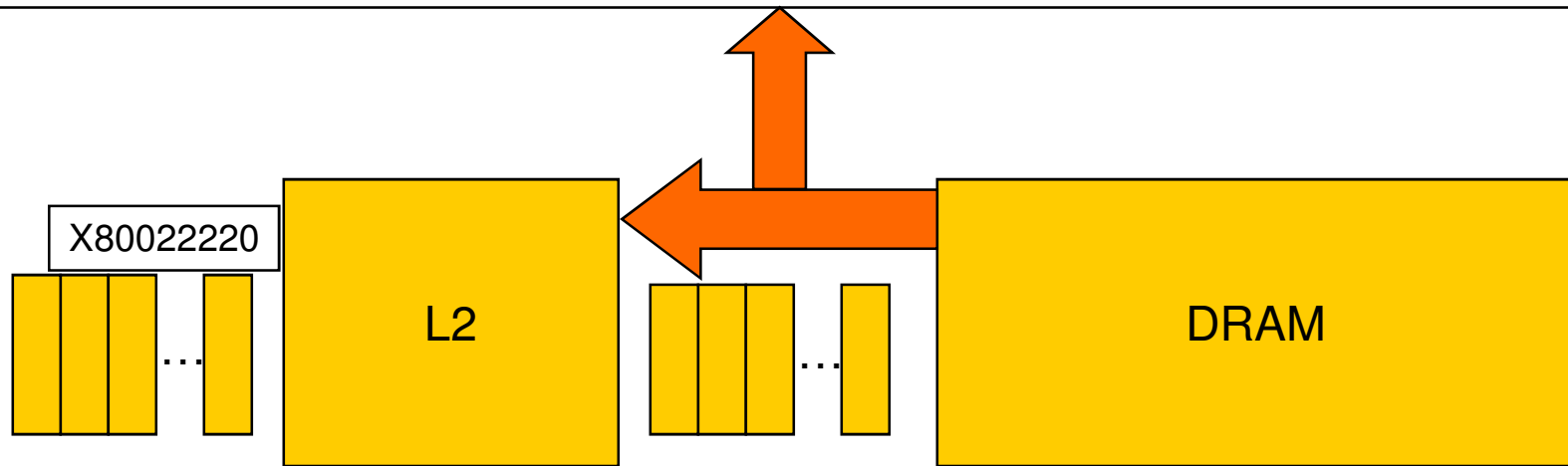
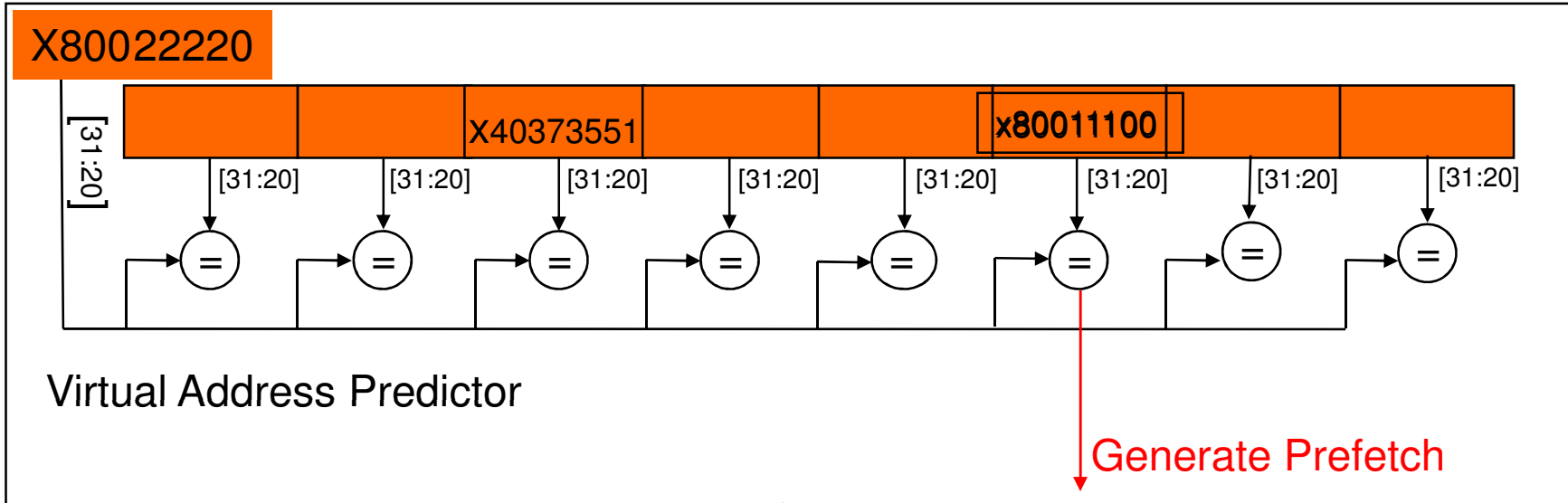




# Contented-Directed Prefetching

- Pointer prefetching scheme
- Look at data of memory
- Search for data which might be memory addresses
  - Cooksey et al. '02

# Content-Directed Prefetching (CDP)





# Pre-computation Based Prefetching

- Speculative execution for prefetching
  - High accuracy and good coverage
  - No architectural changes
  - Multi-processors
    - SMT (later lecture)

```
for (i=1; i<N; i++) {  
    listNode *p = listHead[i];  
    while(p){  
        work(p->data);  
        p=p->next;  
    }  
}
```

original code

```
for (i=1; i<N; i++) {  
    listNode *p = listHead[i];  
    while(p){  
        p=p->next;  
    }  
}
```

speculative execution code



# Review

- S/W prefetching
  - Explicit prefetching requests
  - Prefetch distance, avoid requesting the same cache block (loop unrolling)
- H/W prefetching
  - Observe cache miss address streams (stream, stride, markov, GHB)
  - Observe data in the load (content-directed prefetching)
  - Pre-execution



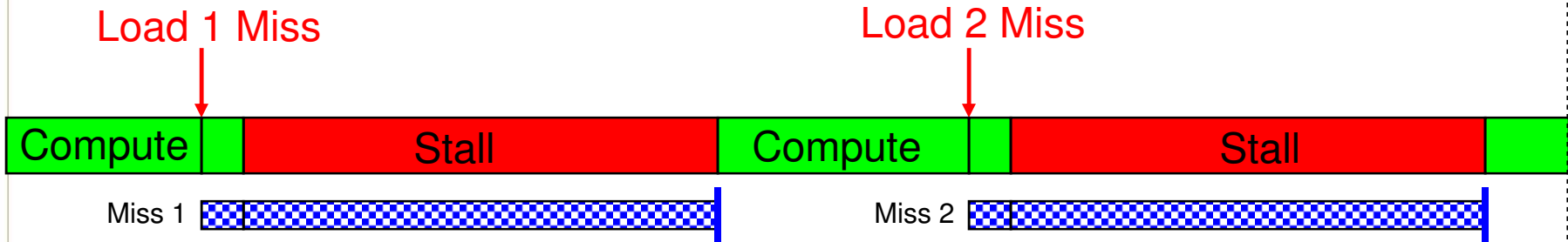
# Runahead Execution

- A technique to obtain the memory-level parallelism benefits of a large instruction window
- When the oldest instruction is an L2 miss:
  - Checkpoint architectural state and enter runahead mode
- In runahead mode:
  - Instructions are speculatively pre-executed
  - The purpose of pre-execution is to generate prefetches
  - L2-miss dependent instructions are marked INV and dropped
- Runahead mode ends when the original L2 miss returns
  - Checkpoint is restored and normal execution resumes

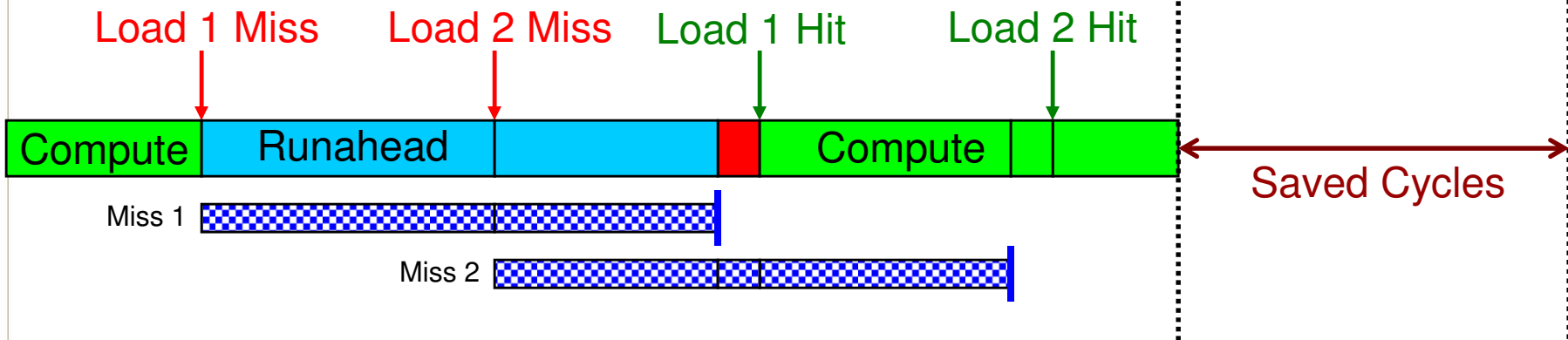
# Runahead Example



*Small Window:*



*Runahead:*





# Overhead of Prefetching

- Overhead of software prefetching
  - Extra instructions
  - Cache pollution
  - Bandwidth consumption
- Overhead of hardware prefetching
  - Transistors (can we use that space for cache ?)
  - Cache pollution
  - Bandwidth consumption



# Programming assignment #2

- Much much much longer than assignment #1
- Helper section KACB 1212 Today 6 pm
- 10% of the total grade and additional 2% extra point (good chance to make up)