Understanding CPU Caches

Ulrich Drepper

Introduction

Discrepancy main CPU and main memory speed

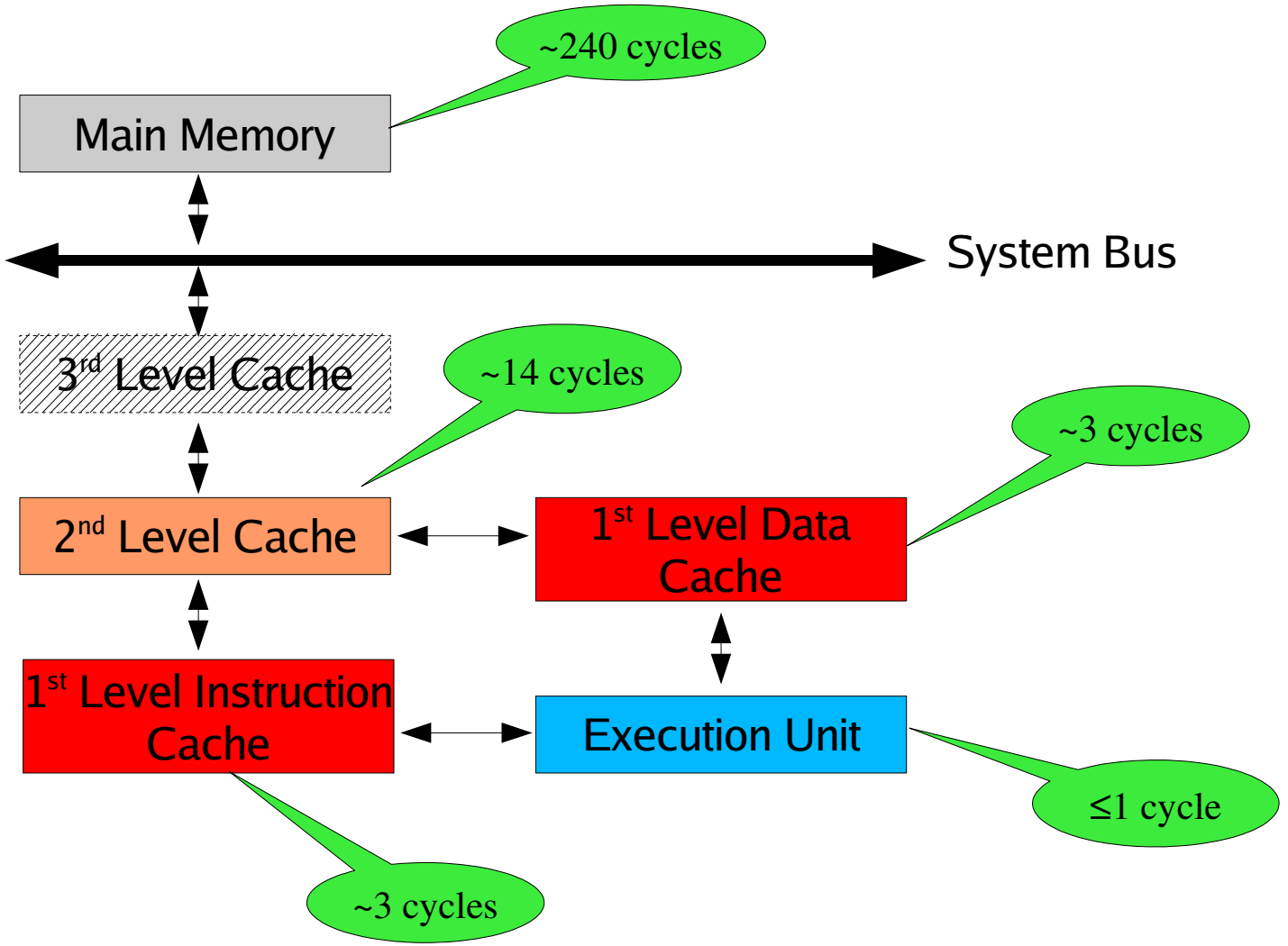
- Intel lists for Pentium M nowadays:
  - ~240 cycles to access main memory
- The gap is widening
- Faster memory is too expensive

The Solution for Now

CPU caches: additional set(s) of memory added between CPU and main memory

- Designed to not change the programs' semantics
- Controlled by the CPU/chipset
- Can have multiple layers with different speed (i.e., cost) and size

How Does It Look Like

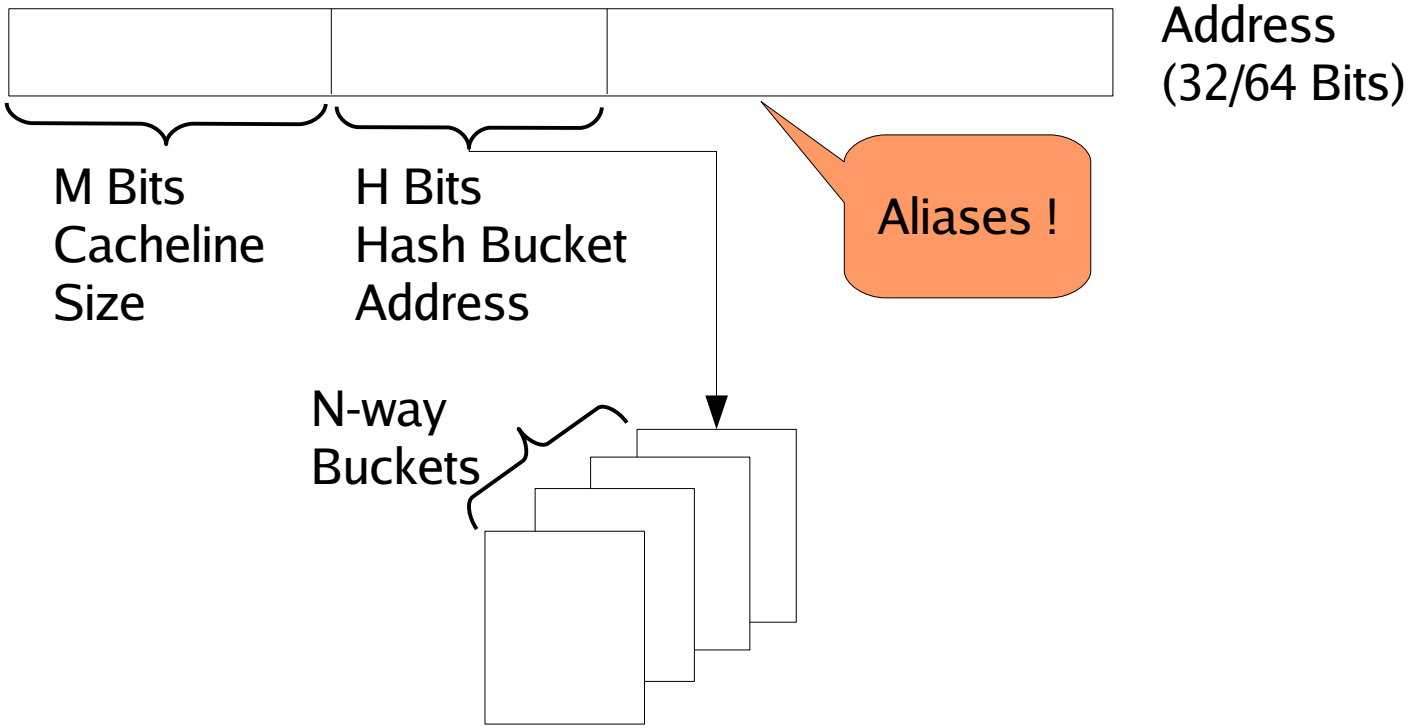


Cache Usage Factors

Numerous factors decide cache performance:

- Cache size
- Cacheline handling
  - associativity
- Replacement strategy
- Automatic prefetching

Cache Addressing



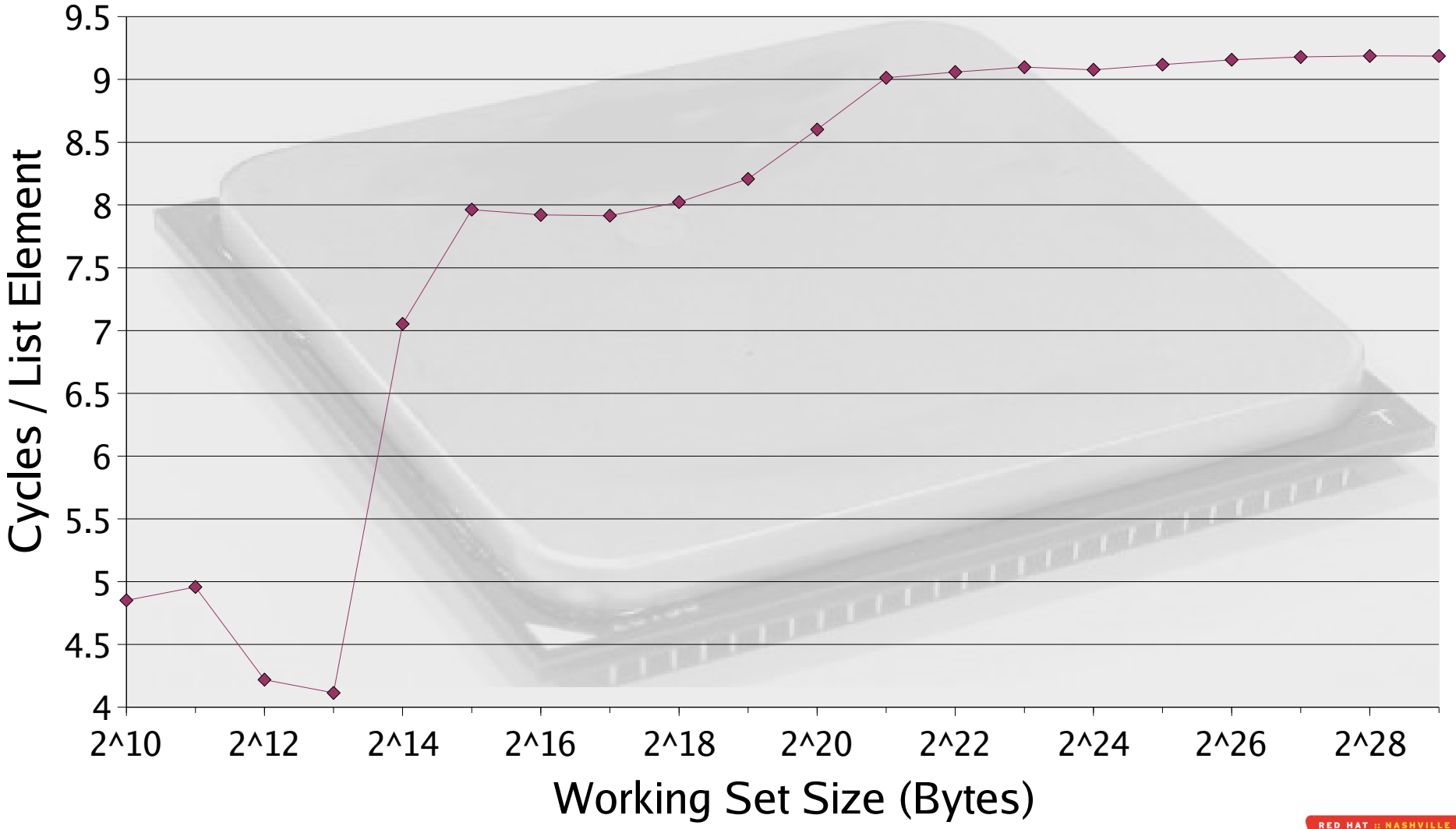
Observing the Effects

Test Program to see the effects:

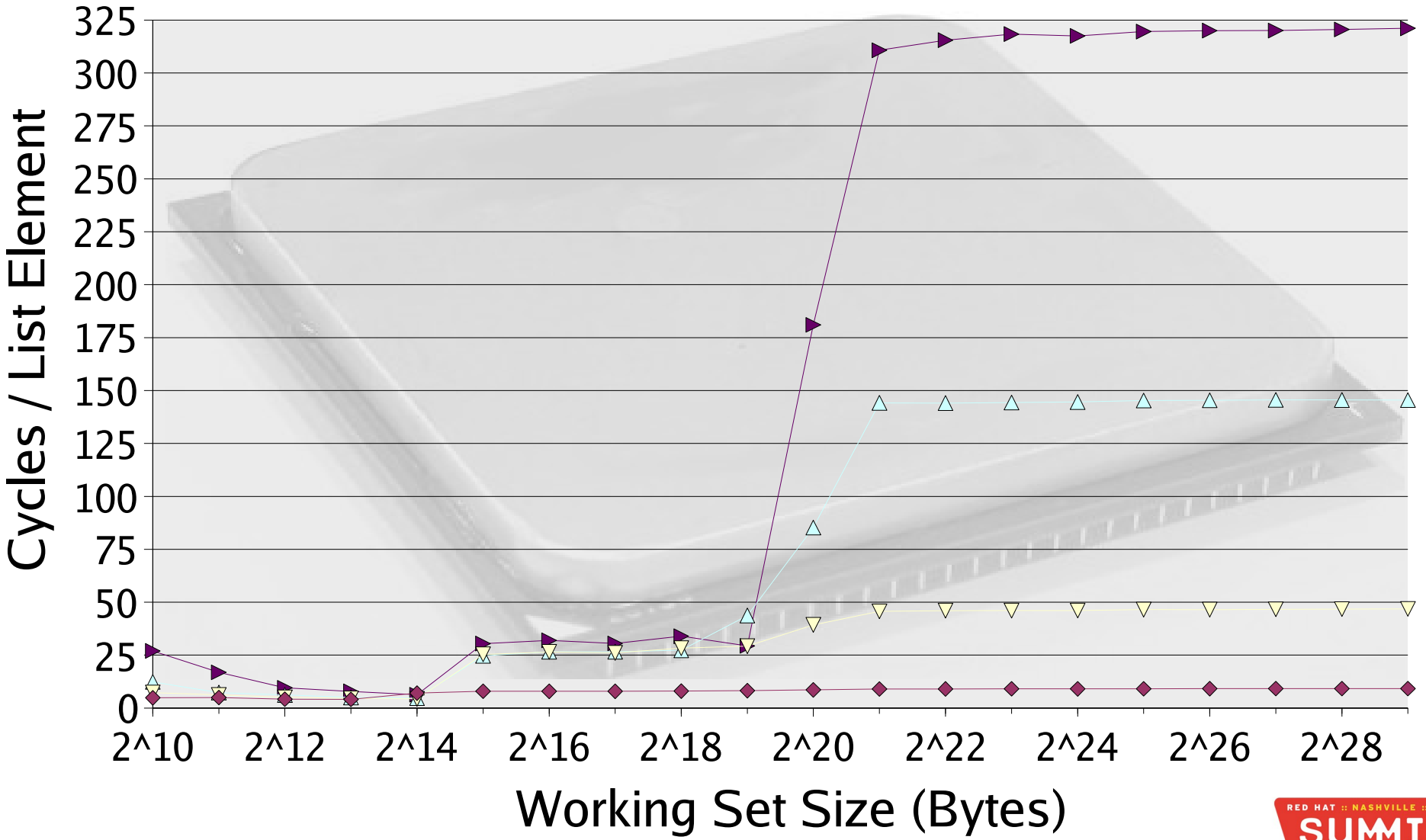
- Walks single linked list
  - Sequential in memory
  - Randomly distributed
- Write to list elements

```
struct l {
  struct l *n;
  long pad[NPAD];
};
```

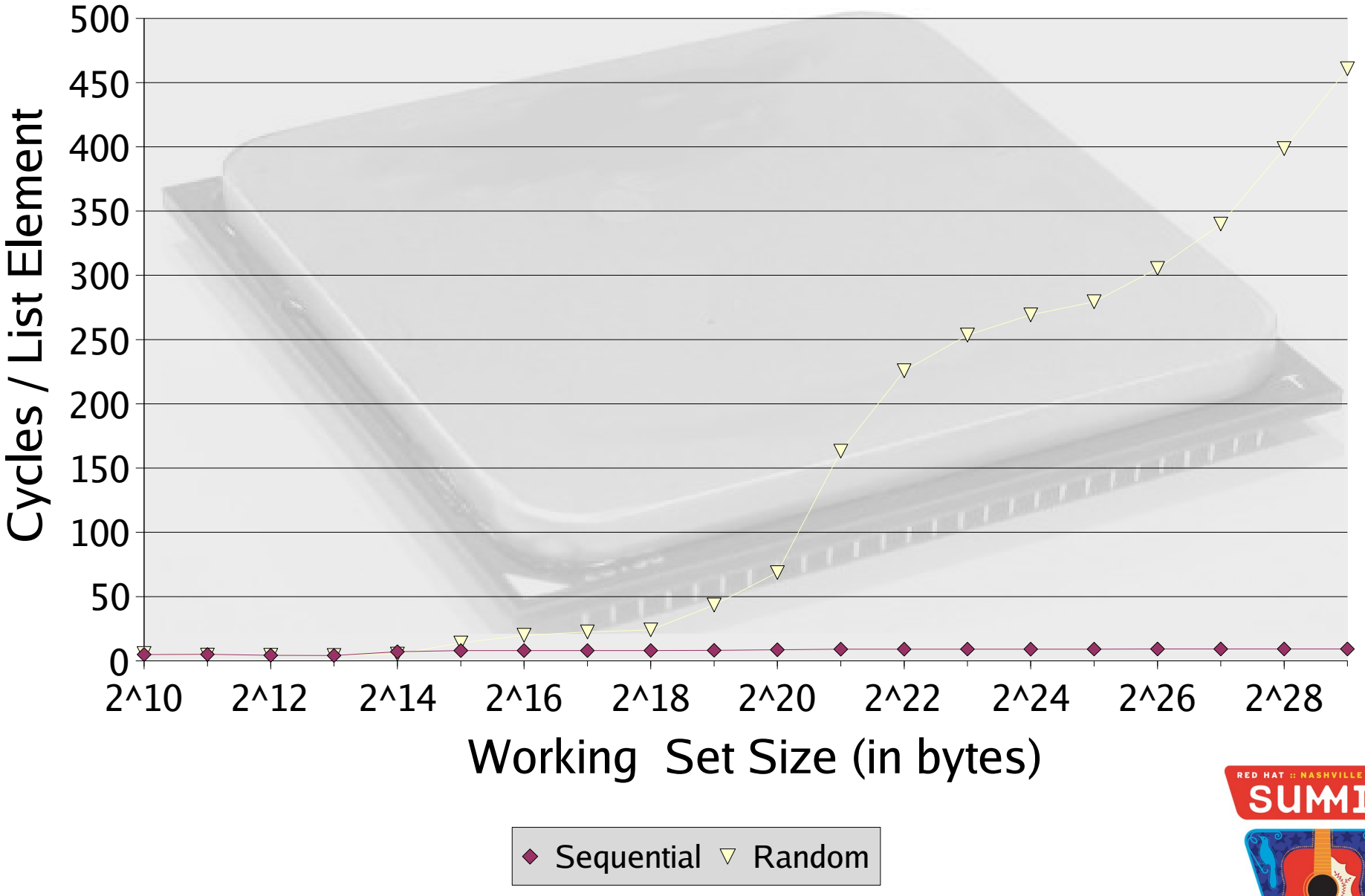
Sequential Access (NPAD=0)



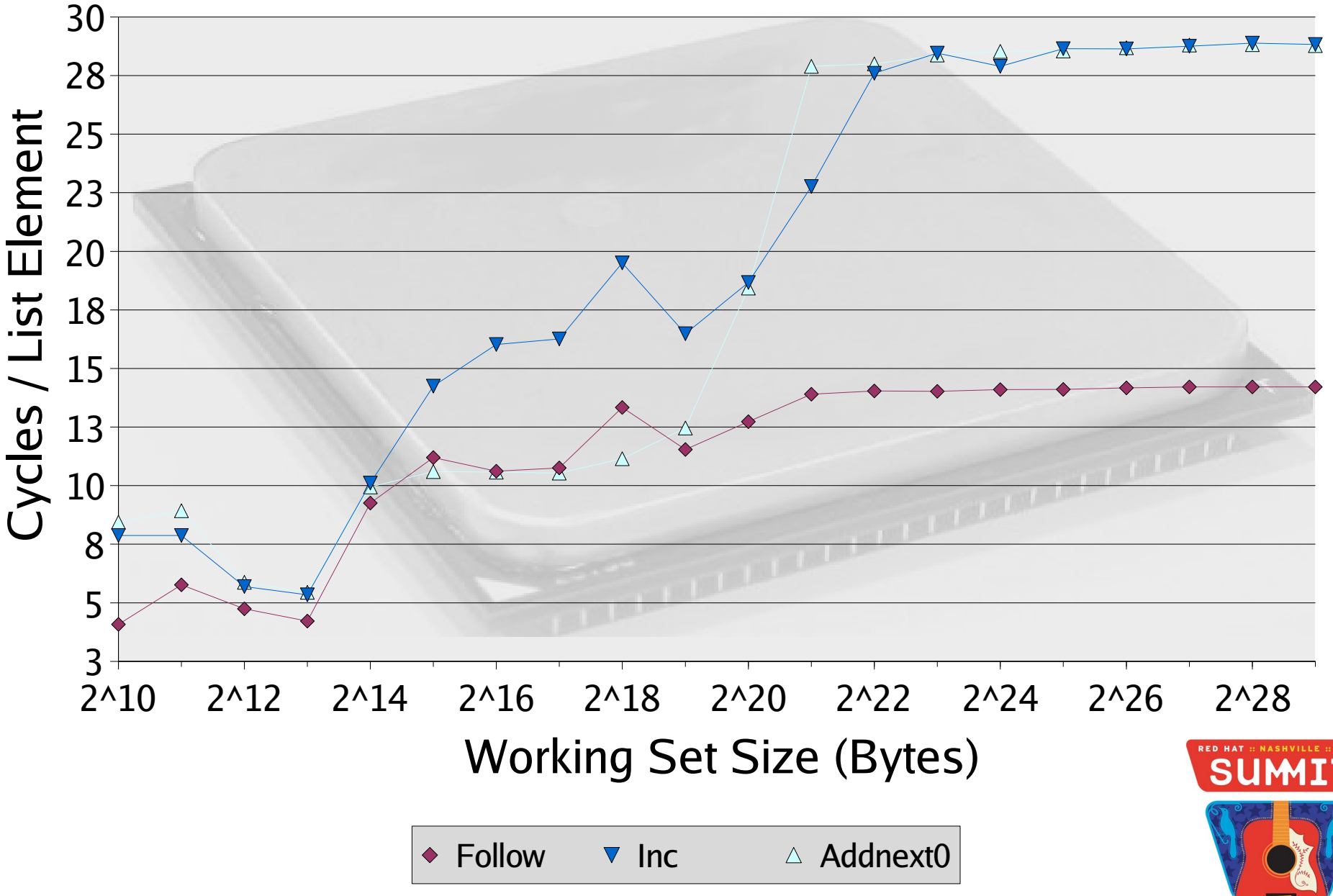
Sequential List Access



Sequential vs Random Access (NPAD=0)



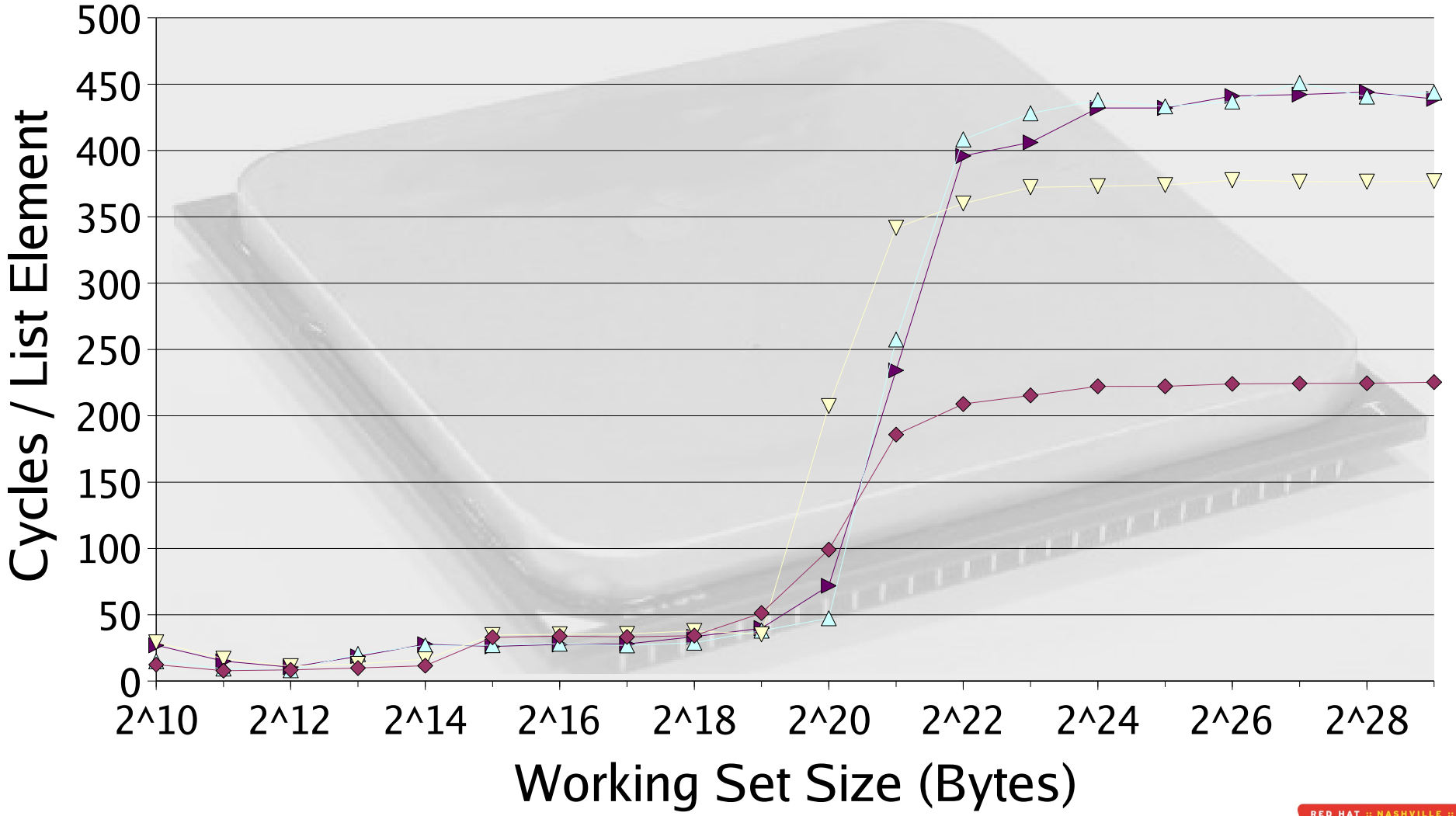
Sequential Access (NPAD=1)



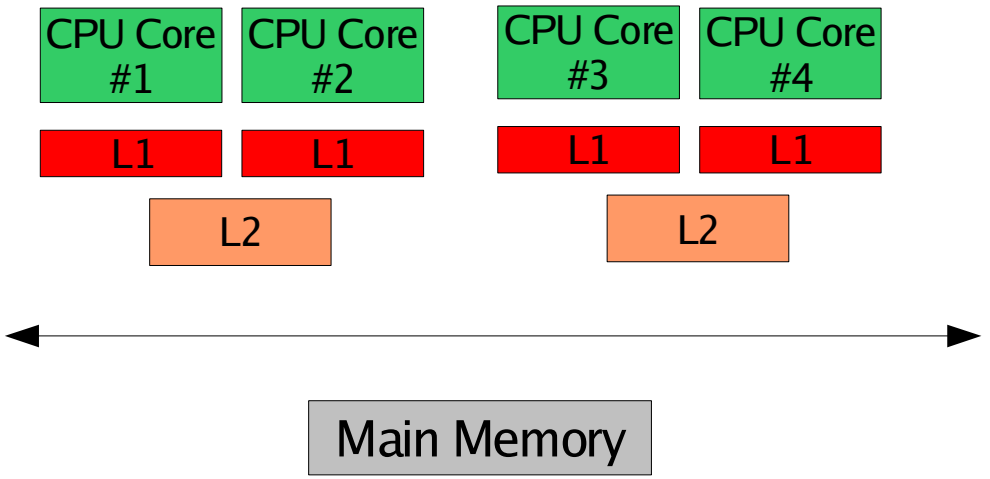
Optimizing for Caches I

- Use memory sequentially
  - For data, use arrays instead of lists
  - For instructions, avoid indirect calls
- Chose data structures as small as possible
- Prefetch memory

Sequential Access w/ vs w/out L3

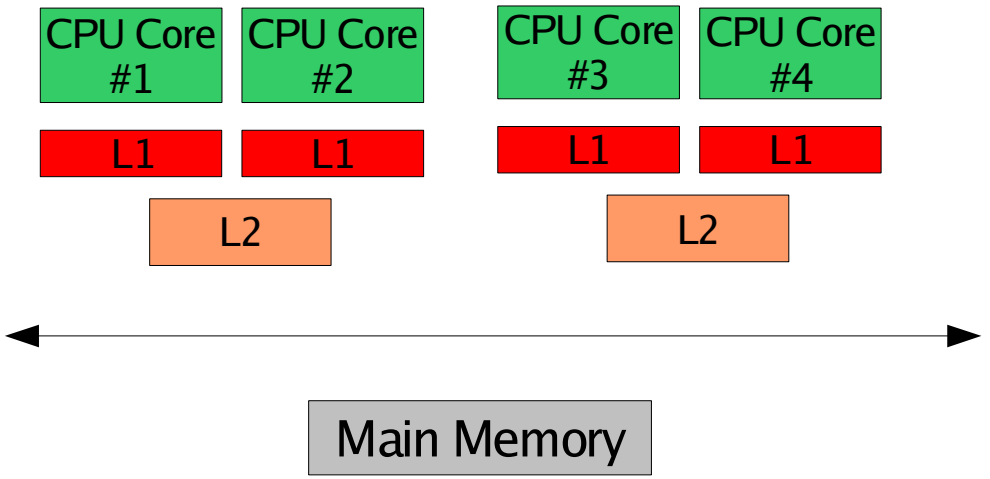


More Fun: Multithreading



1. CPU Core #1 and #3 read from a memory location; L2 the relevant L1 contain the data
2. CPU Core #2 writes to the memory location
   a) Notify L1 of core #1 that content is obsolete
   b) Notify L2 and L1 of second proc that content is obsolete

More Fun: Multithreading

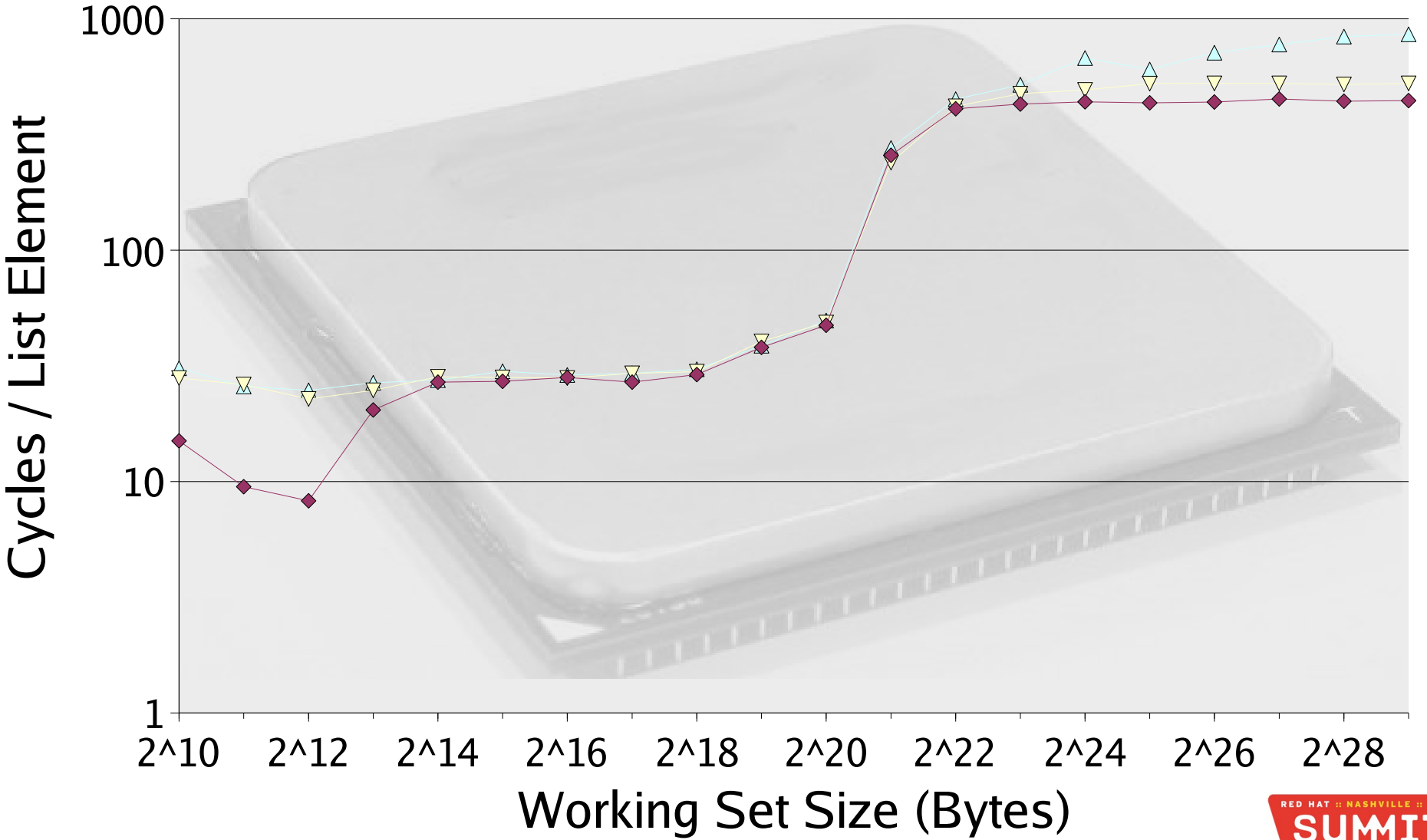


3. Core #4 writes to the memory location

   a) Wait for core #2's cache content to land in main memory

   b) Notify core #2's L1 and L2 that content is obsolete

Sequential Increment 128 Byte Elements



Optimizing for Caches II

Cacheline ping-pong is deadly for performance

- If possible, write always on the same CPU
- Use per-CPU memory; lock thread to specific CPU
- Avoid placing often independently read and written-to data in the same cacheline

Questions?