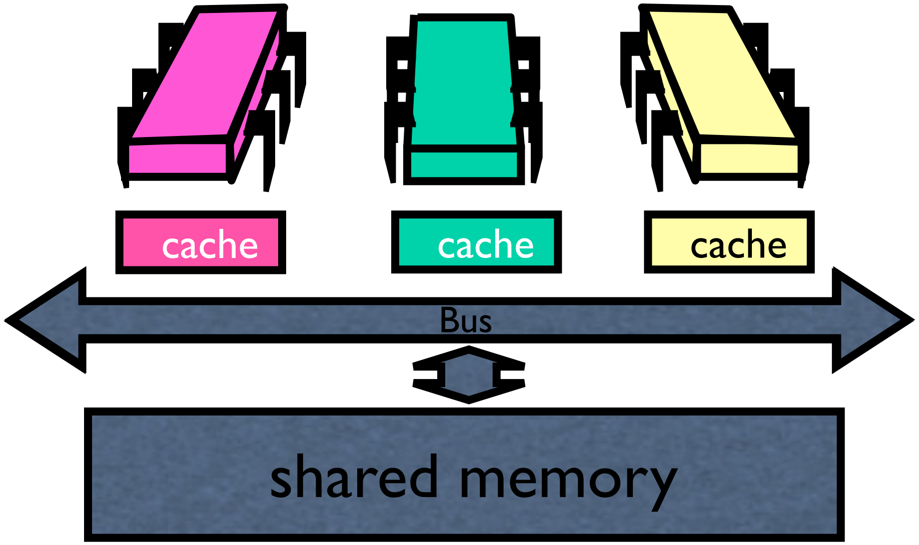
Caching

Christof Fetzer, TU Dresden

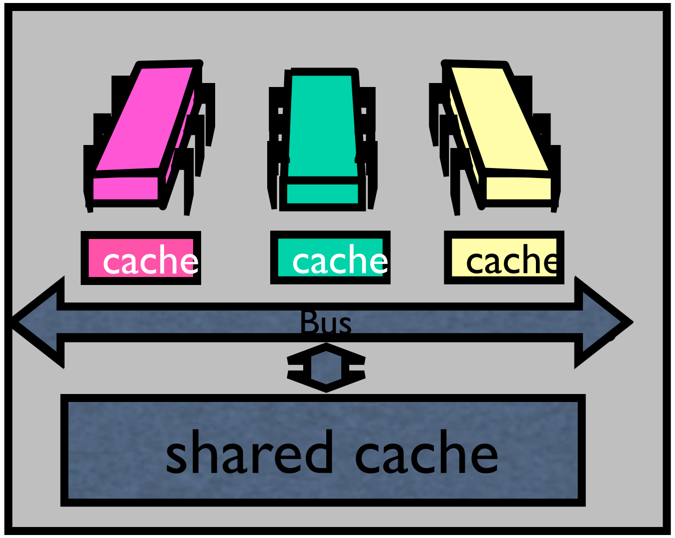
Based on slides by Maurice Herlihy and Nir Shavit

Old days: Symmetric Multiprocessing (SMP)



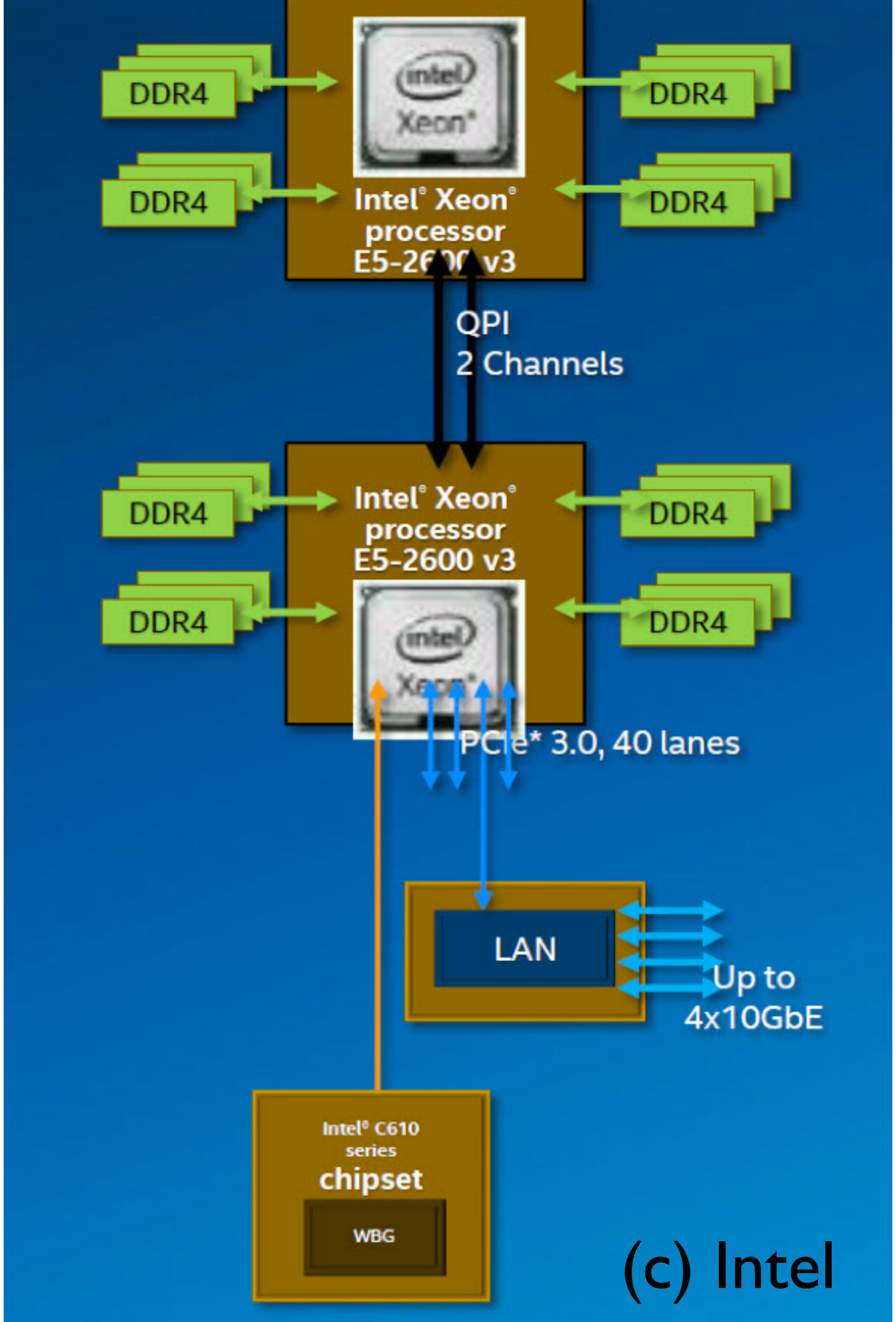
Multicore CPUs

All on the same chip



Intel 2x18 cores (E5-2600v3, 2014)

ccNUMA =
cache **c**oherent
Non **U**niform **M**emory **A**rchitecture



(c) Intel

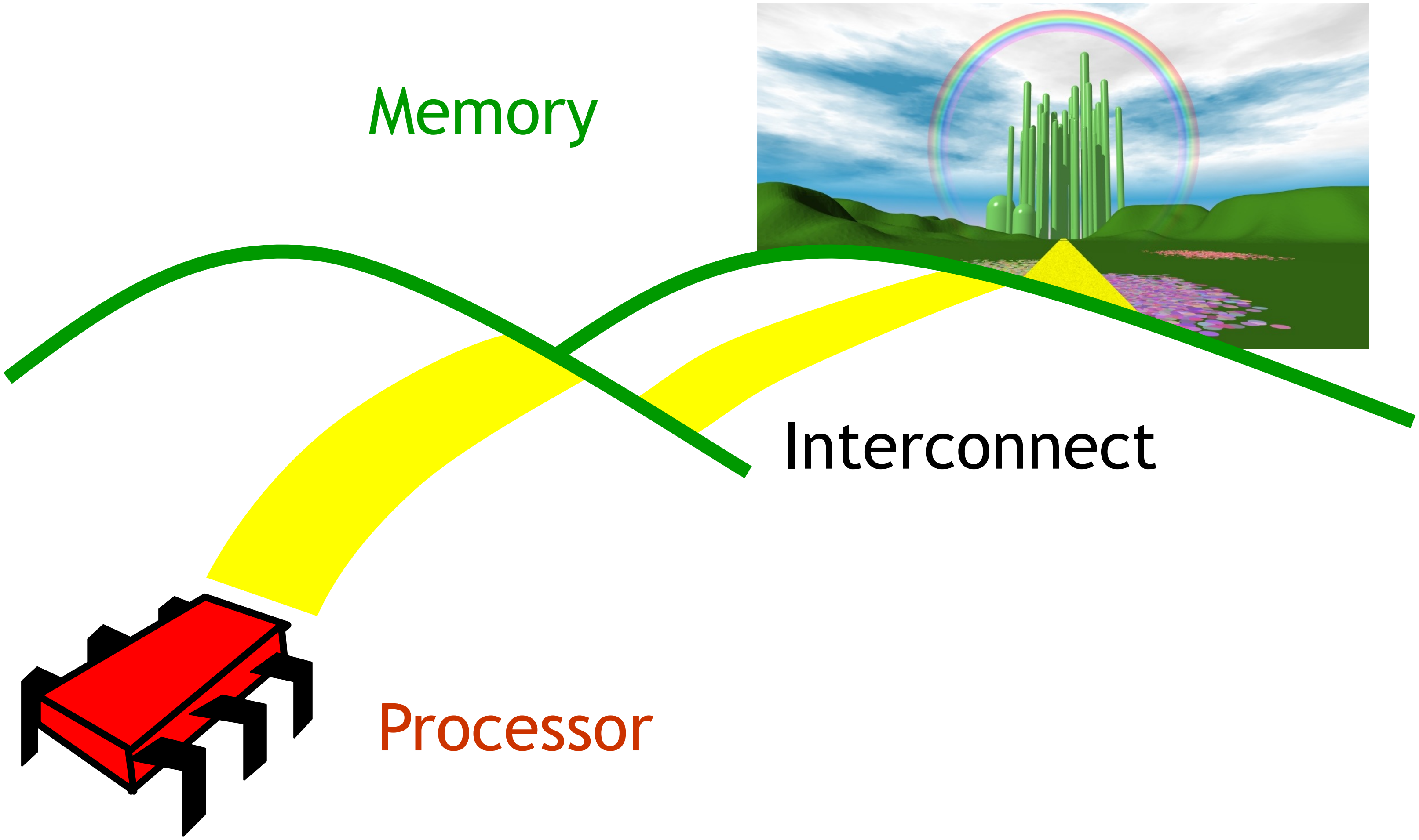
Interconnect

- Bus (old days)
  - Like a tiny (old) Ethernet
  - Broadcast medium
  - Connects
    - Processors to memory
    - Processors to processors
- Network
  - Like tiny LAN
  - State of the art in most systems

Interconnect

- Interconnect is a finite resource
- Processors can be delayed if others are consuming too much
- Avoid algorithms that use too much bandwidth
  - Read/write memory

CPU and Memory are Far Apart

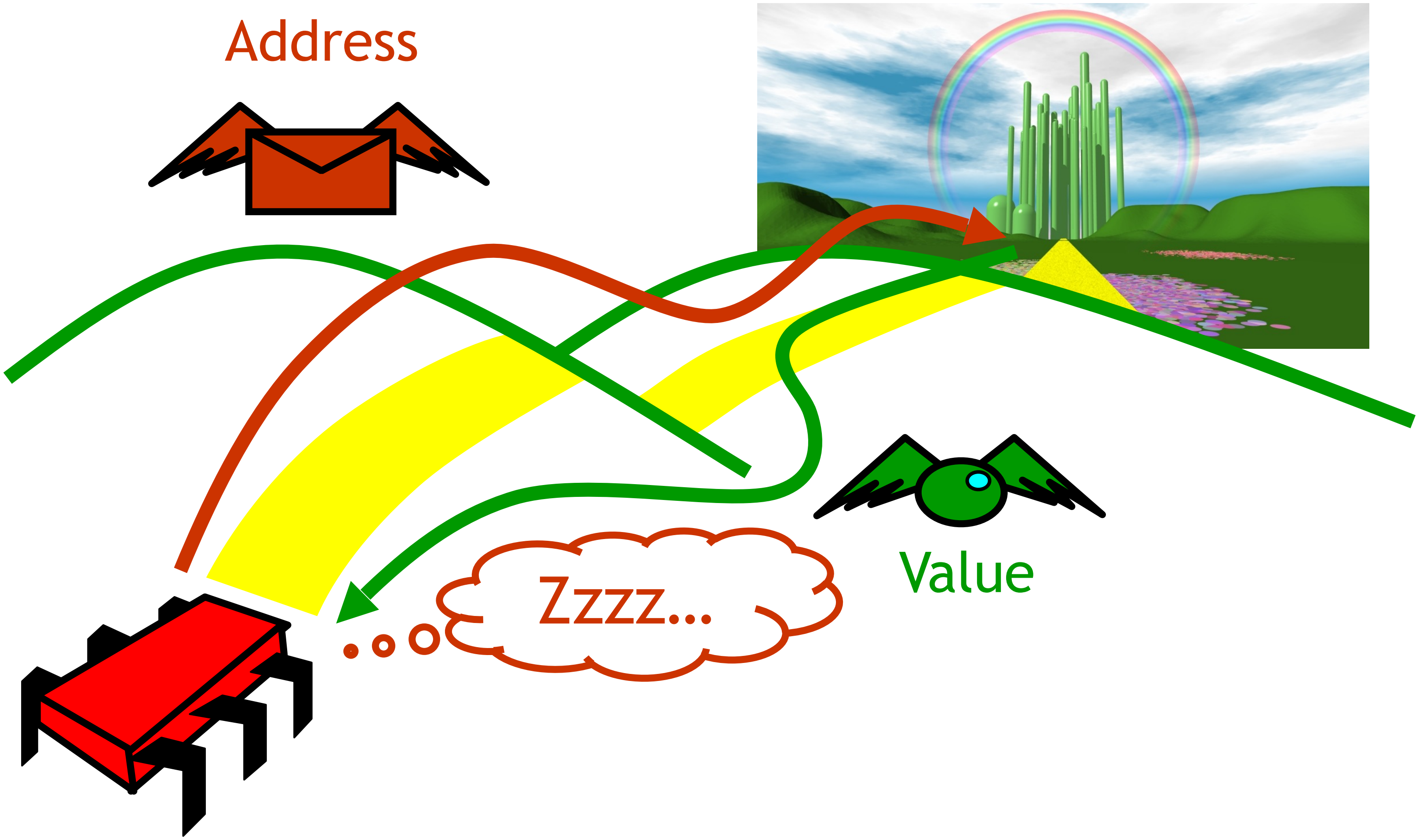


Memory

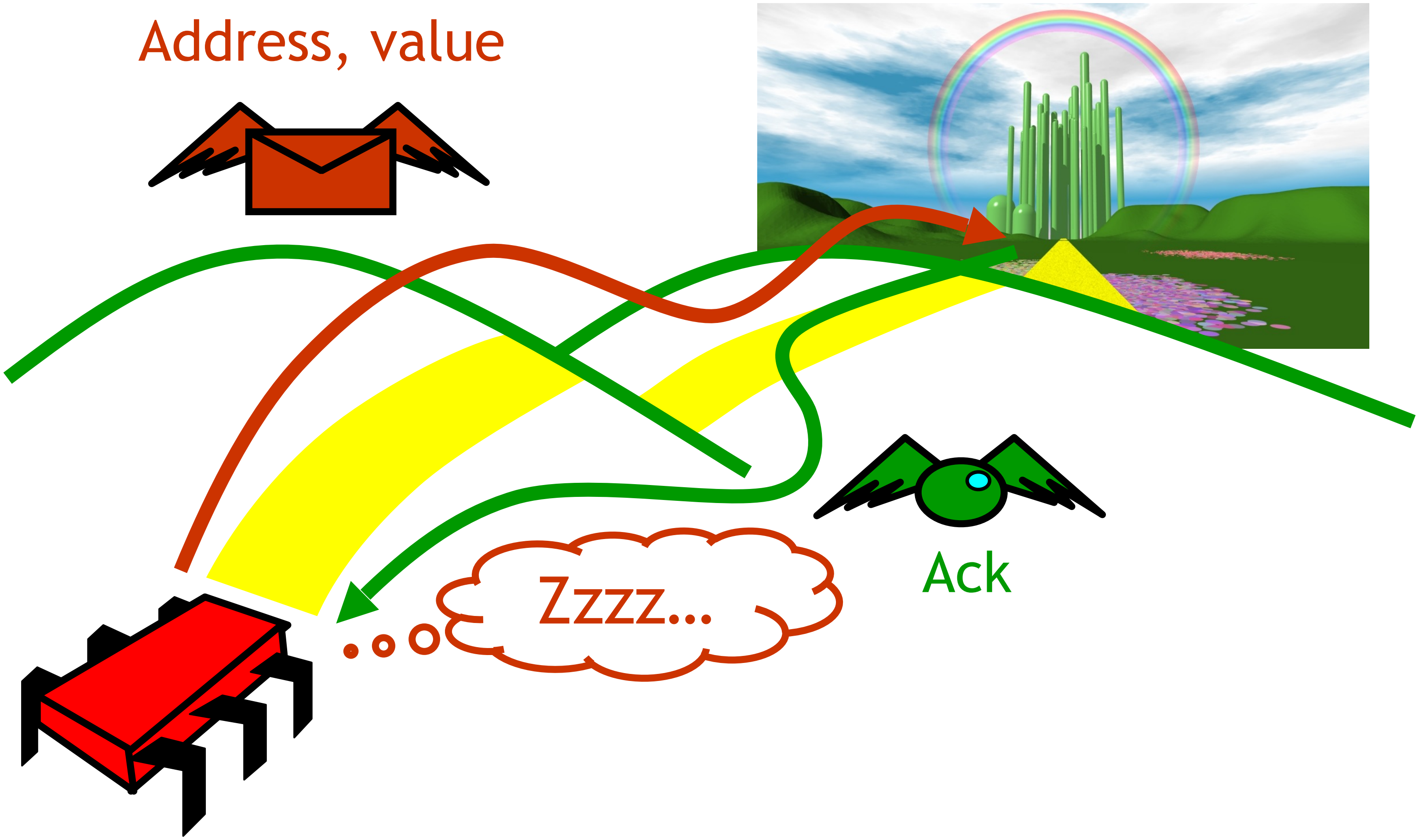
Interconnect

Processor

Reading from Memory



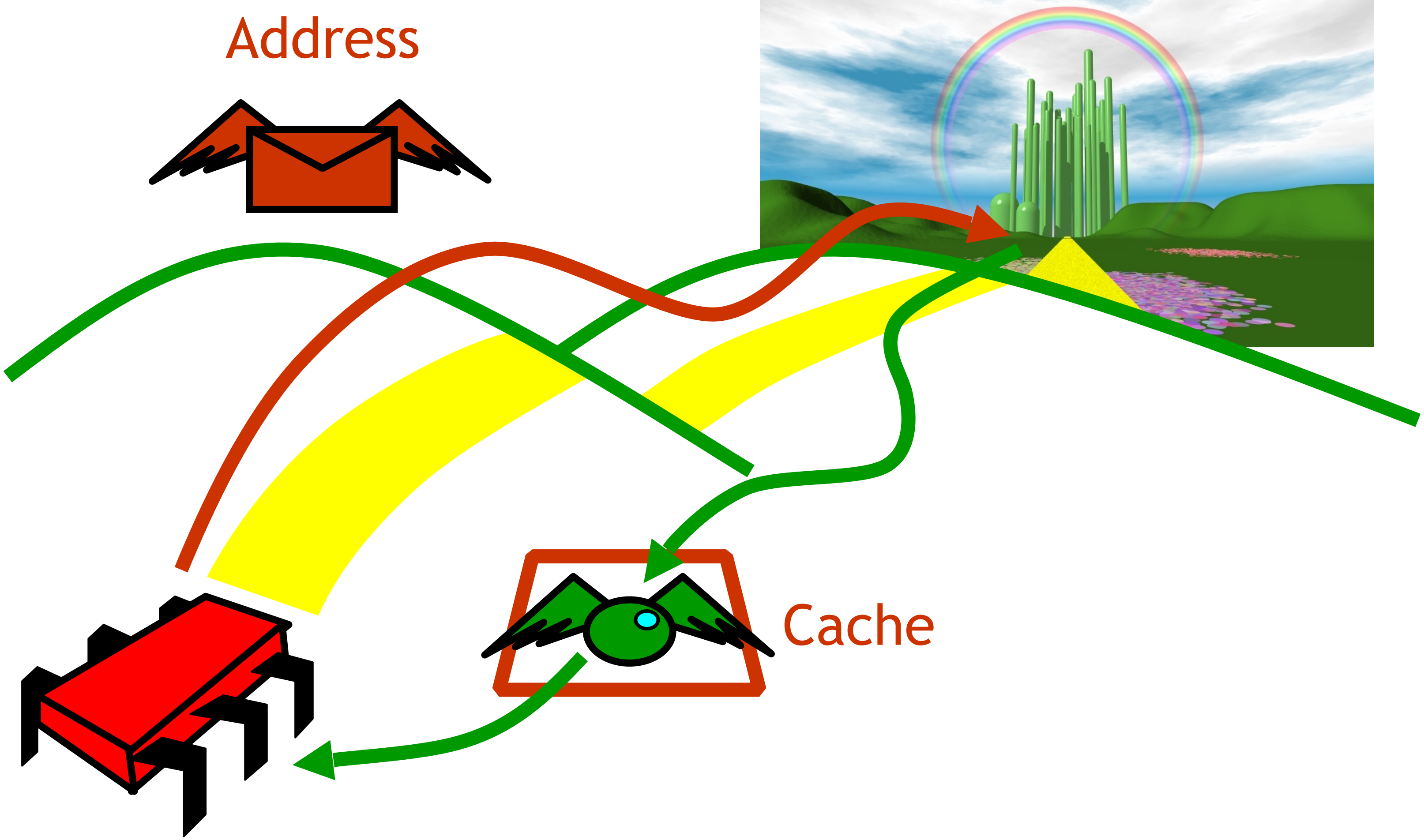
Writing to Memory



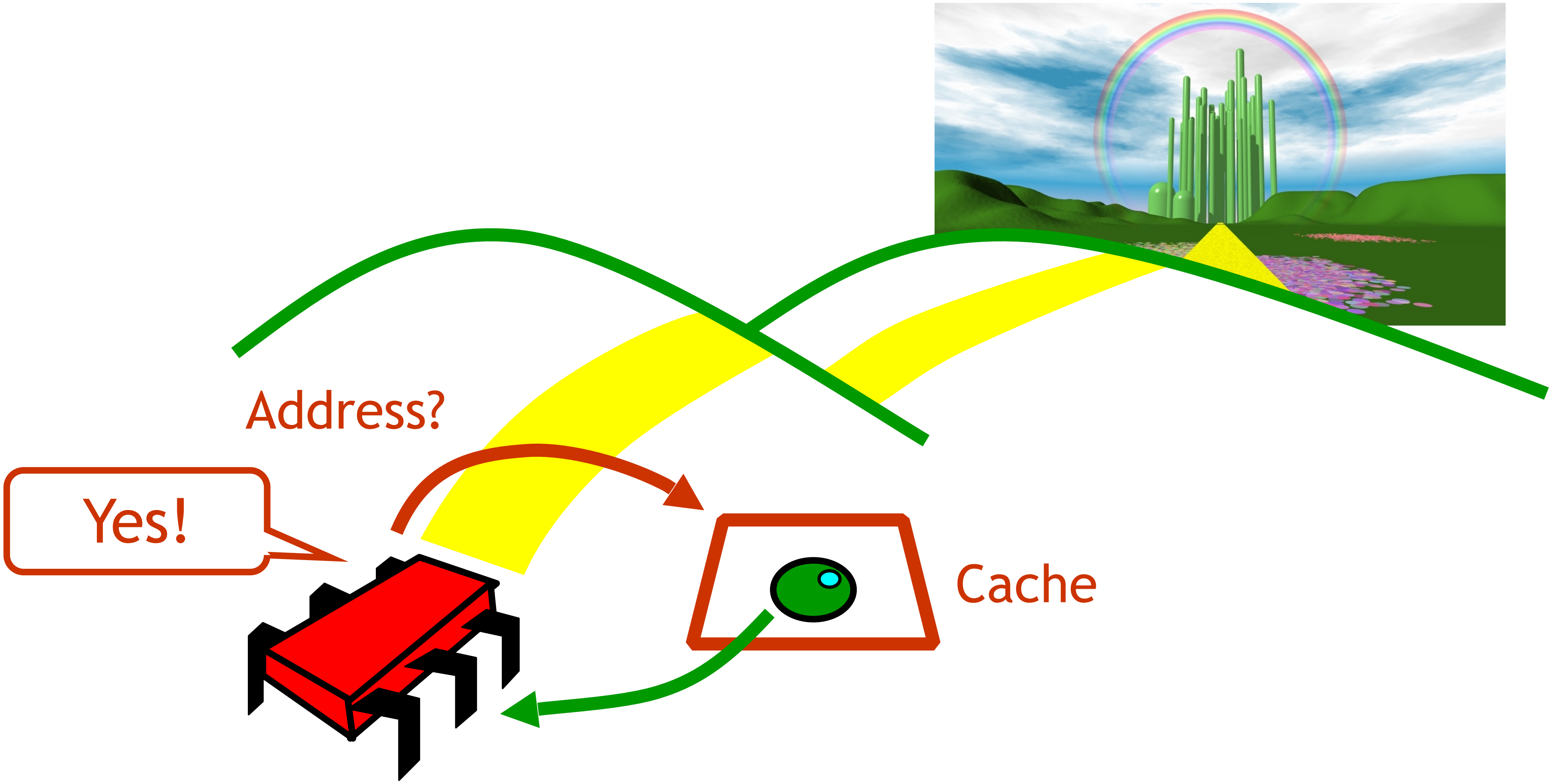
Remote Spinning

- Thread waits for a bit in memory to change
  - E.g., tries to dequeue from an empty buffer, tries to acquire lock owned by another thread
- Spins
  - Repeatedly rereads flag bit
- Huge waste of interconnect bandwidth
  - Generates continuous traffic on bus

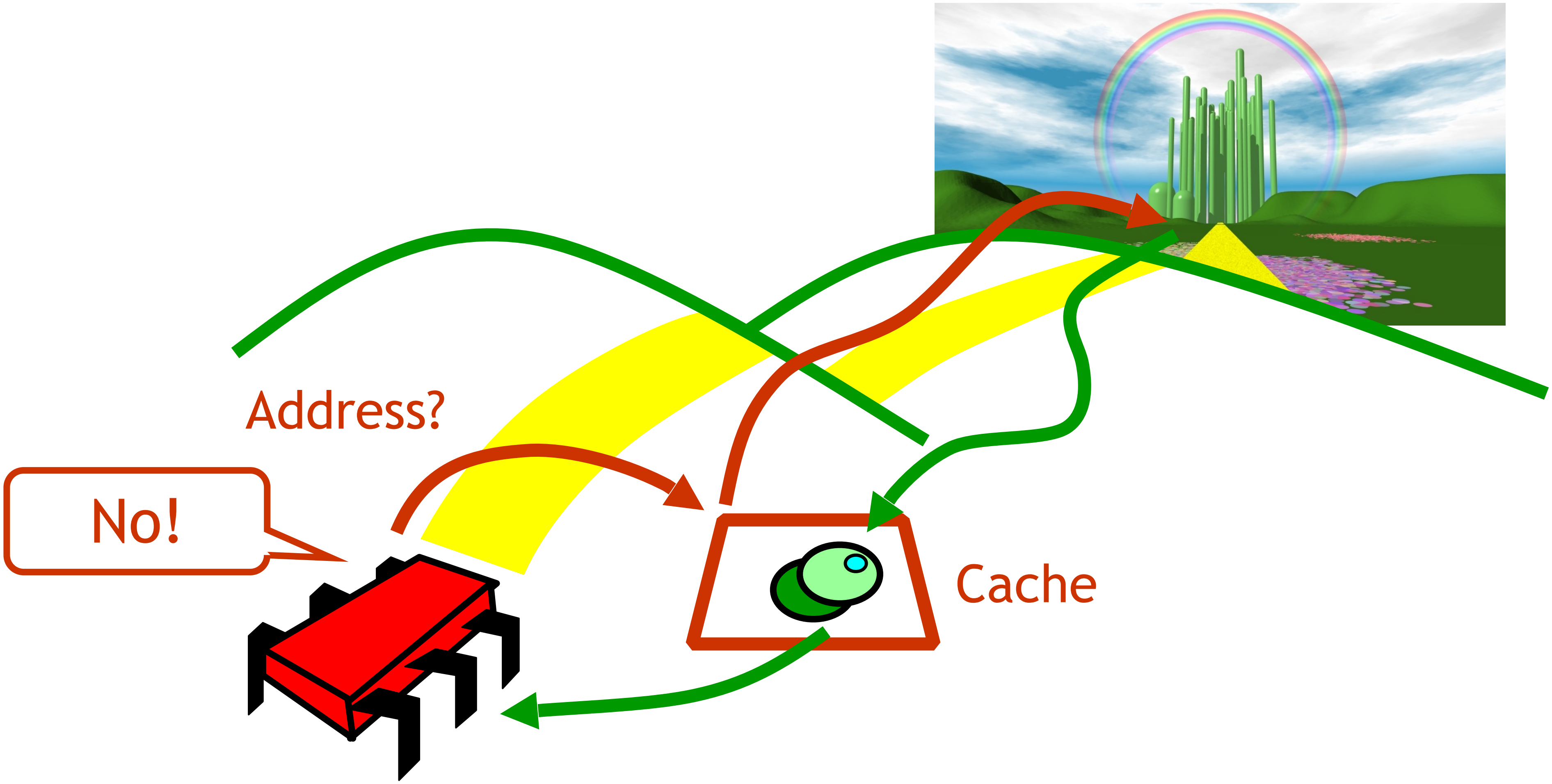
Cache: Reading from Memory



Cache Hit

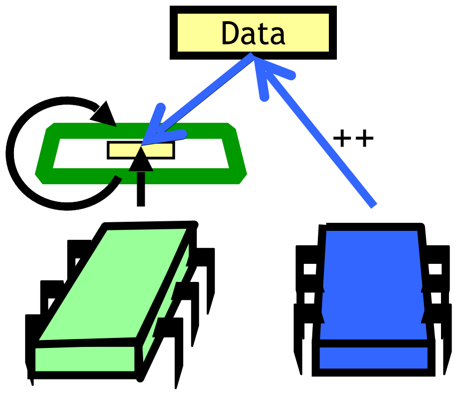


Cache Miss

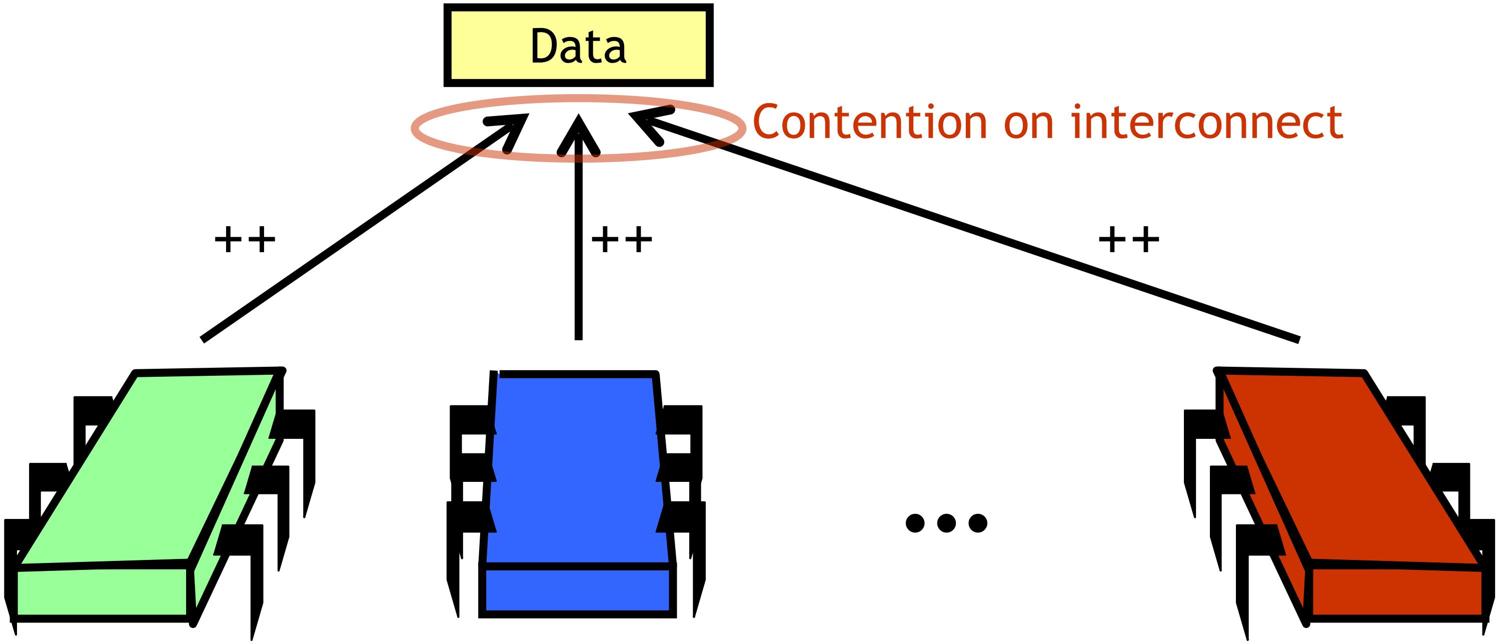


Local Spinning

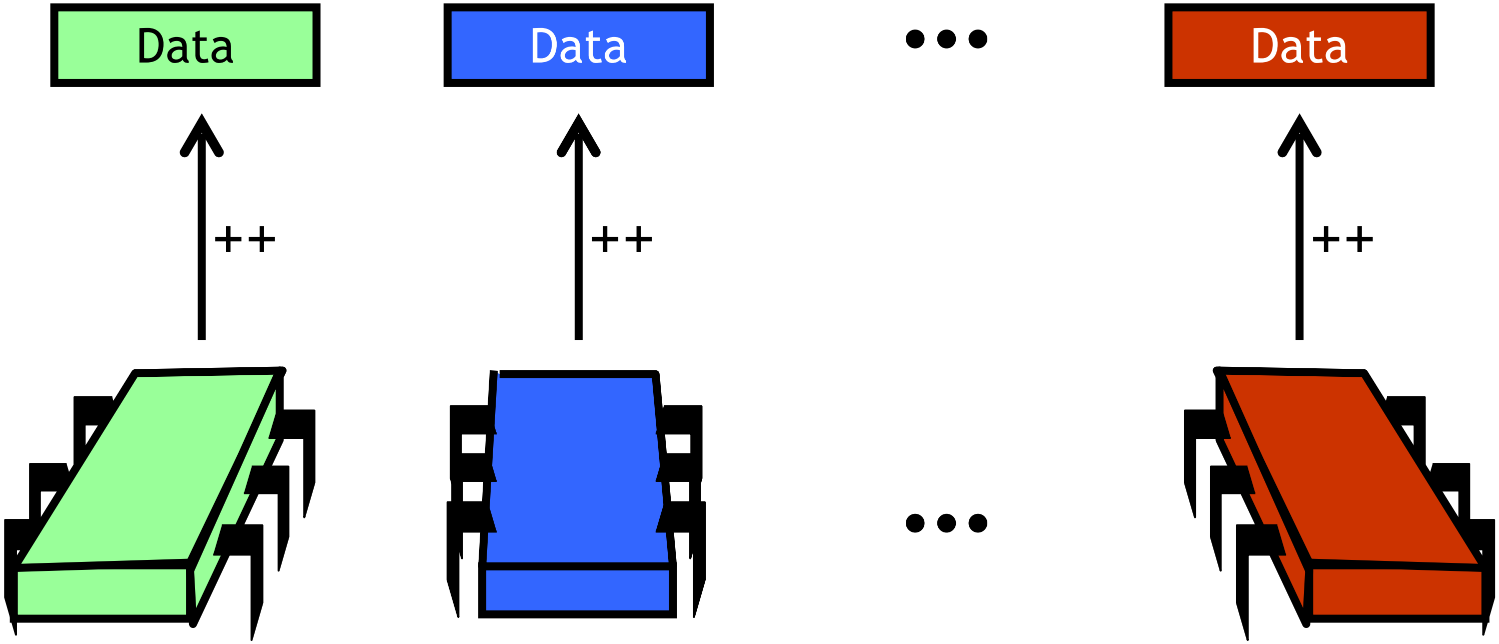
- With caches, spinning becomes practical
- First time
  - Load flag bit into cache
- As long as it does not change
  - Hit in cache (no interconnect used)
- When it changes
  - One-time cost
  - See cache coherence below



Understanding Cache Behavior

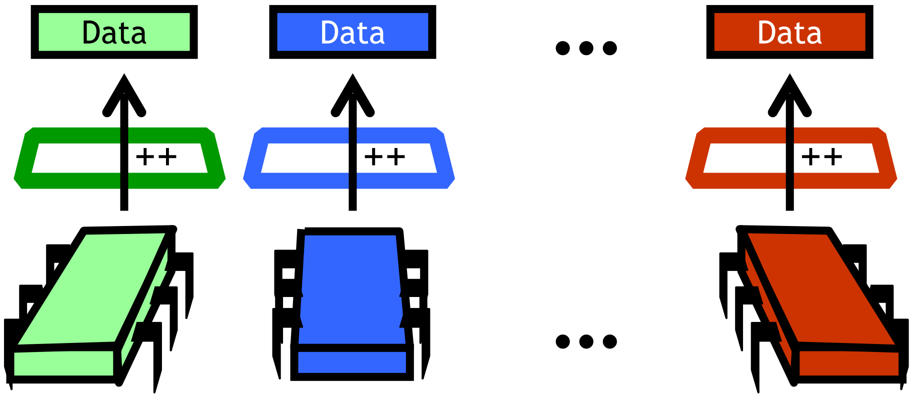


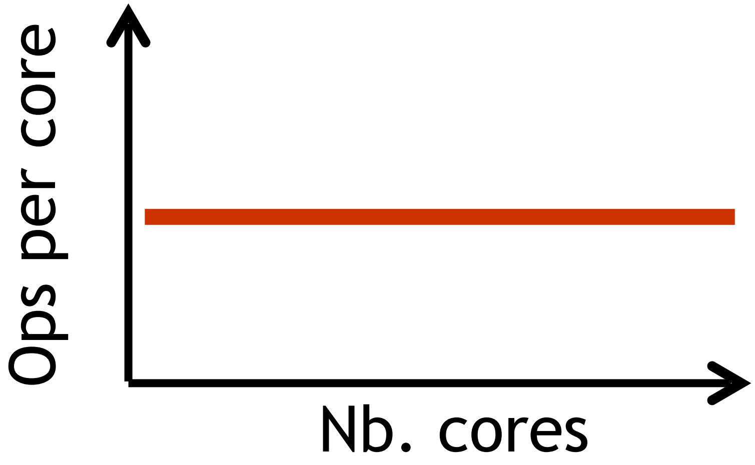
Understanding Cache Behavior



Understanding Cache Behavior

- Expected behavior
  - Data is not shared
  - Cores should read from and write to cache
  - No contention on interconnect
  - Throughput per thread should be constant if enough cores are available

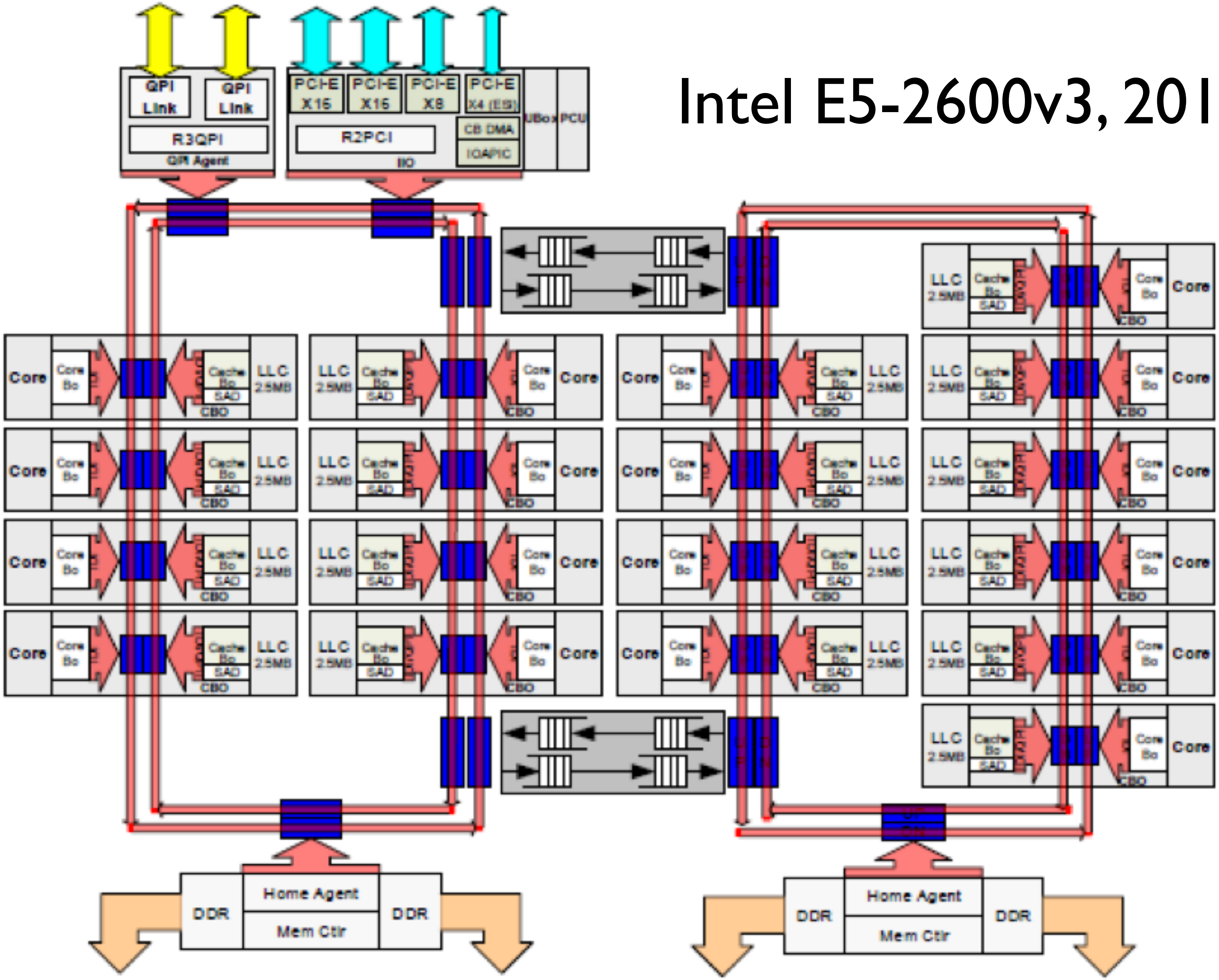




14-18 Core (HCC)

(c) Intel

Intel E5-2600v3, 2014

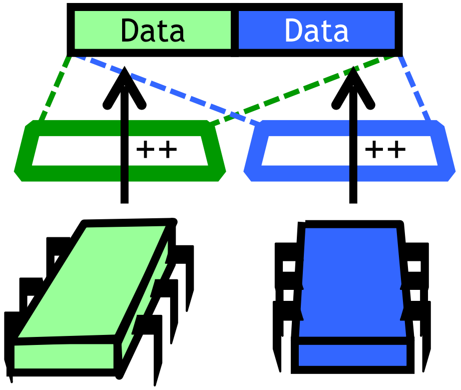


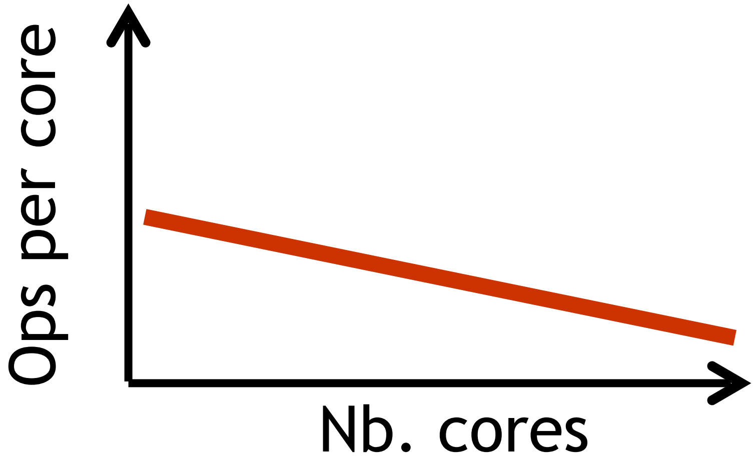
Granularity and Locality

- Caches operate at a larger granularity than a word
- Cache line: fixed-size block containing the address
  - E.g., 64 bytes on Intel i7
- If you use an address now, you will probably use a nearby address soon
  - In the same cache line

Understanding Cache Behavior

- Observed behavior
  - Variables are in the same cache line
  - Cache lines are shared
  - Modification of one variable invalidates full cache line
  - Every write invalidates caches of cores sharing same cache line

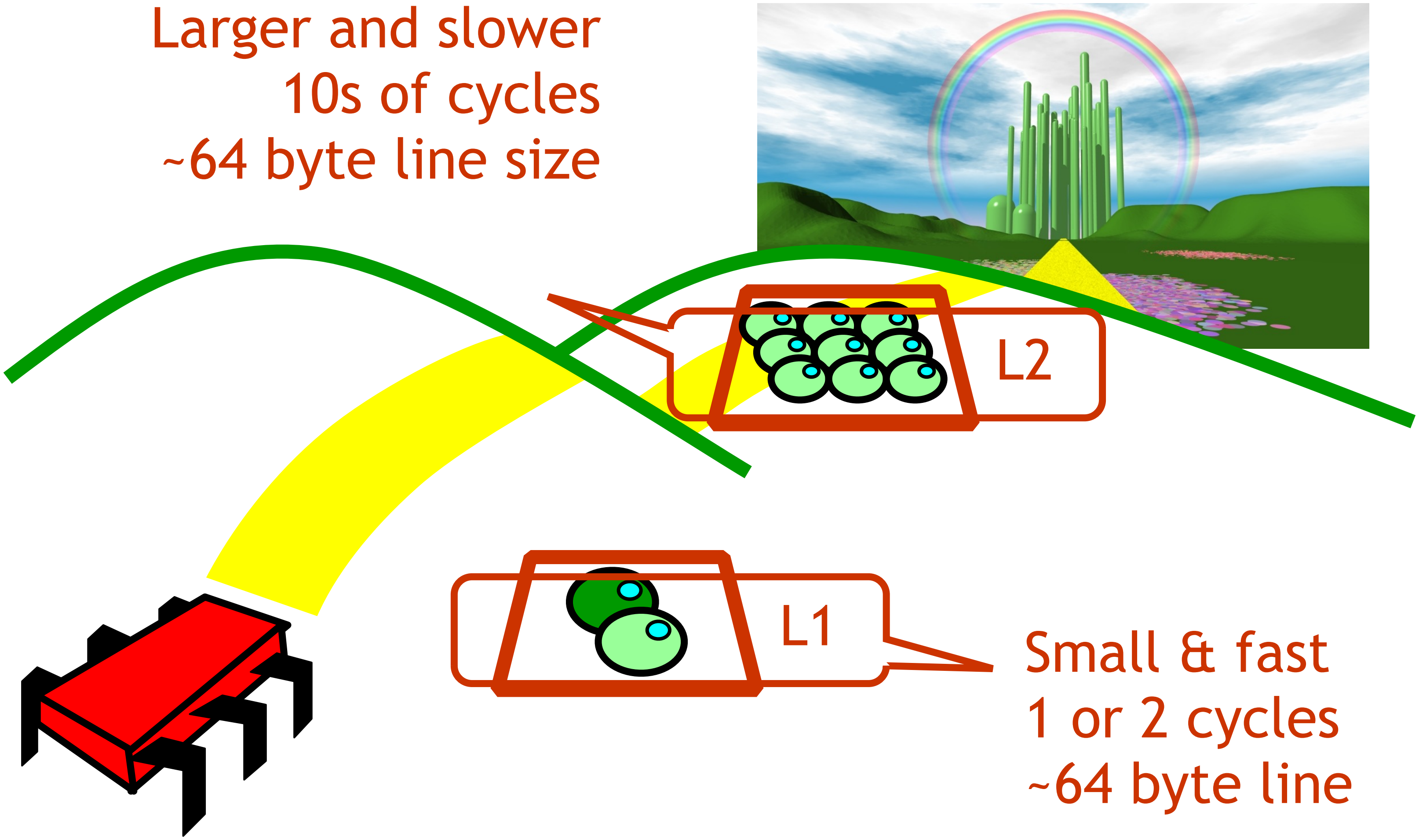




False Sharing

- Two processors may conflict over disjoint addresses
  - If those addresses map on the same cache line
- Large cache line size
  - Increases locality
  - But also increases likelihood of false sharing
- Sometimes need to “scatter” data to avoid this problem

L1 and L2 Caches



Larger and slower
10s of cycles
~64 byte line size

Small & fast
1 or 2 cycles
~64 byte line

Hit Ratio

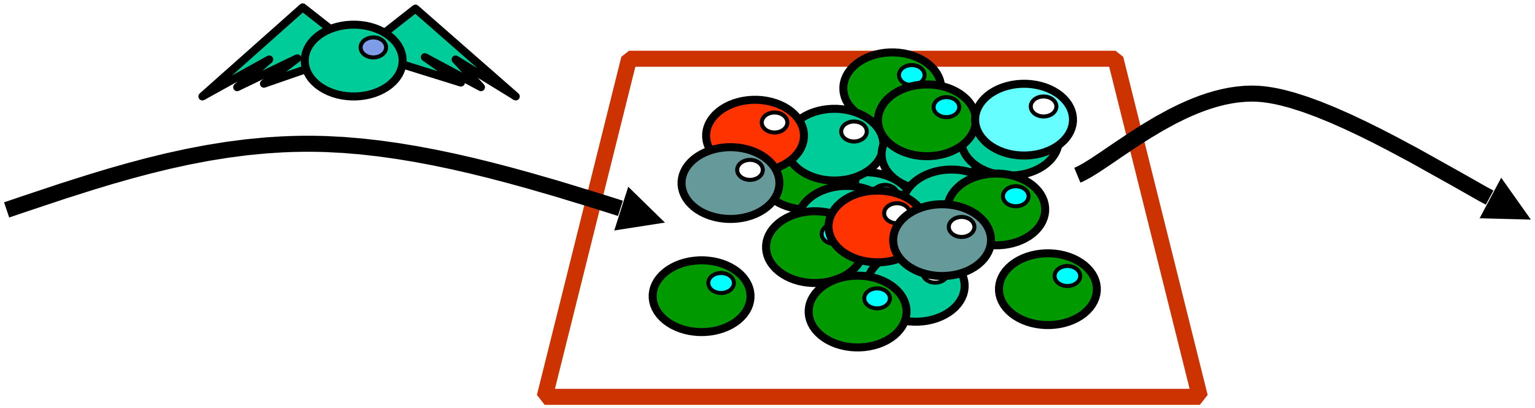
- If you use an address now, you will probably use it again soon
  - Fetch from cache, not memory
- Hit ratio: proportion of requests that hit in the cache
  - Measure of effectiveness of caching mechanism
  - Depends on locality of application

When a Cache Becomes Full…

- Need to make room for new entry
- By evicting an existing entry
- Need a replacement policy
  - Usually some kind of least recently used heuristic

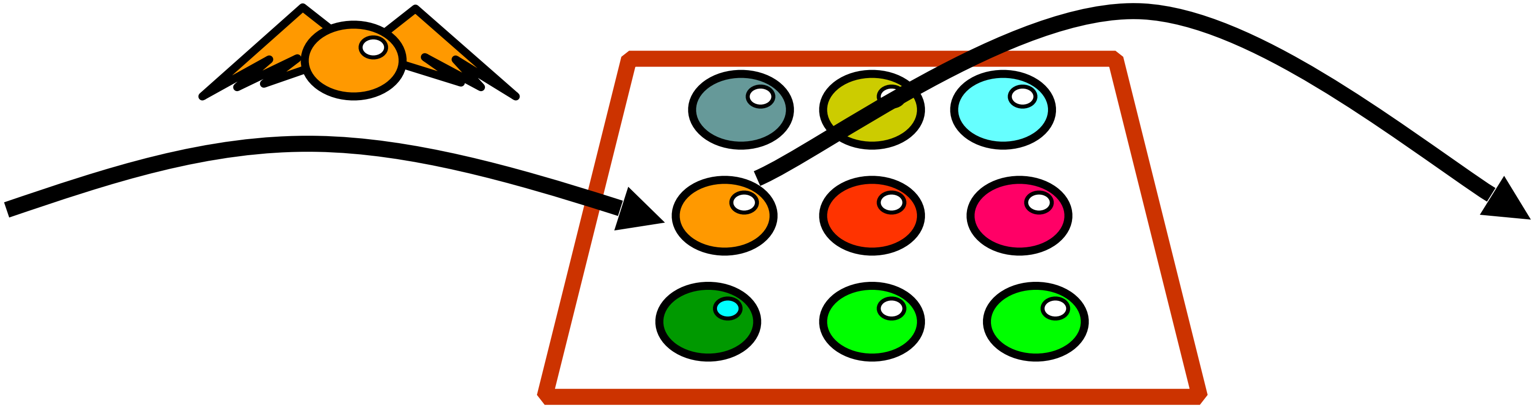
Fully Associative Cache

- Any line can be anywhere in the cache
  - Advantage: can replace any line
  - Disadvantage: hard to find lines



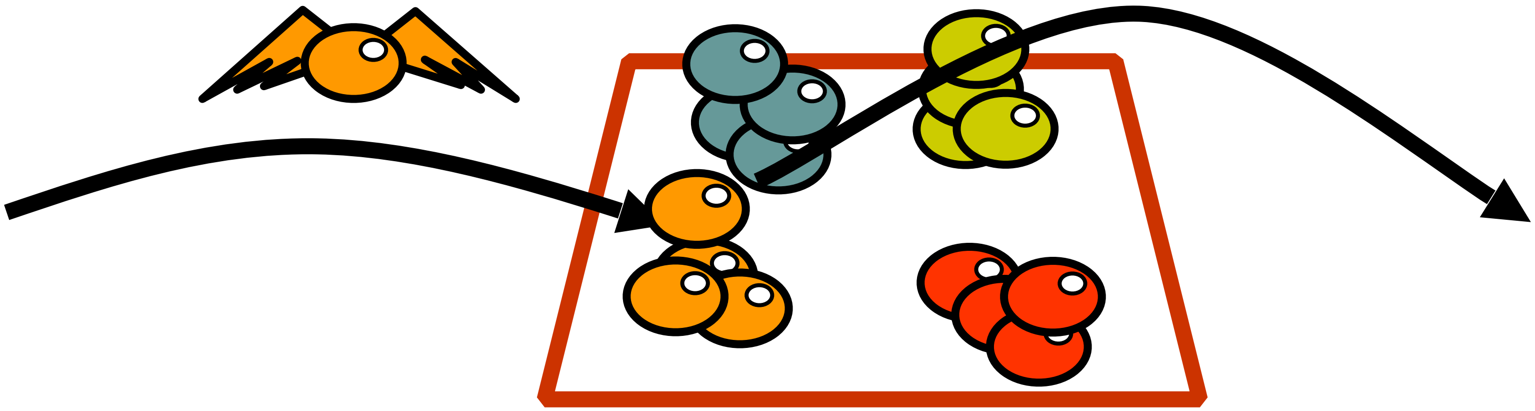
Direct Mapped Cache

- Every address has exactly 1 slot
  - Advantage: easy to find a line
  - Disadvantage: must replace fixed line



K-way Set Associative Cache

- Each slot holds **k** lines
  - Advantage: pretty easy to find a line
  - Advantage: some choice in replacing line



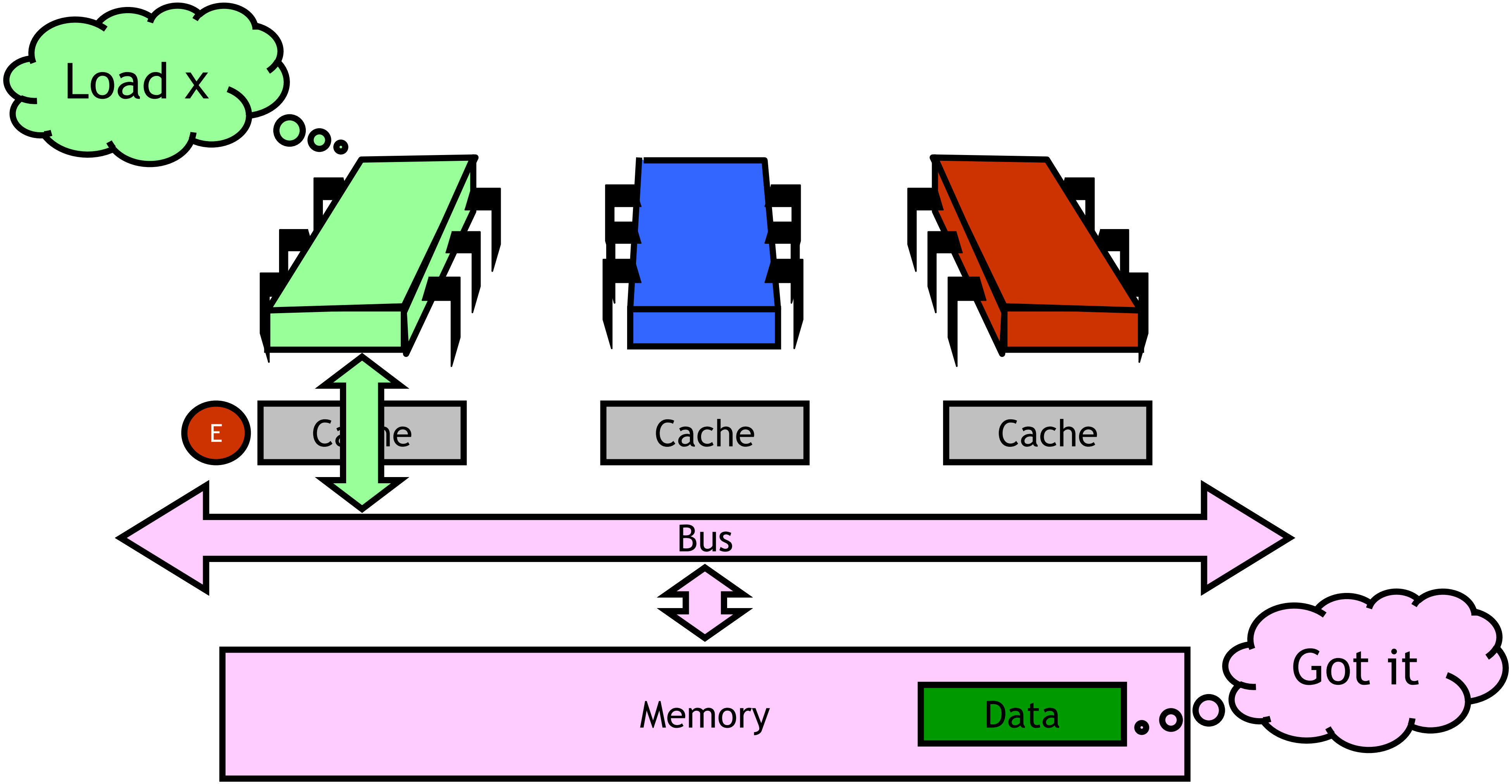
Cache Coherence

- Processor A and B both cache address x
- A writes to x
  - Updates cache
- How does B find out?
- Many cache coherence protocols in literature

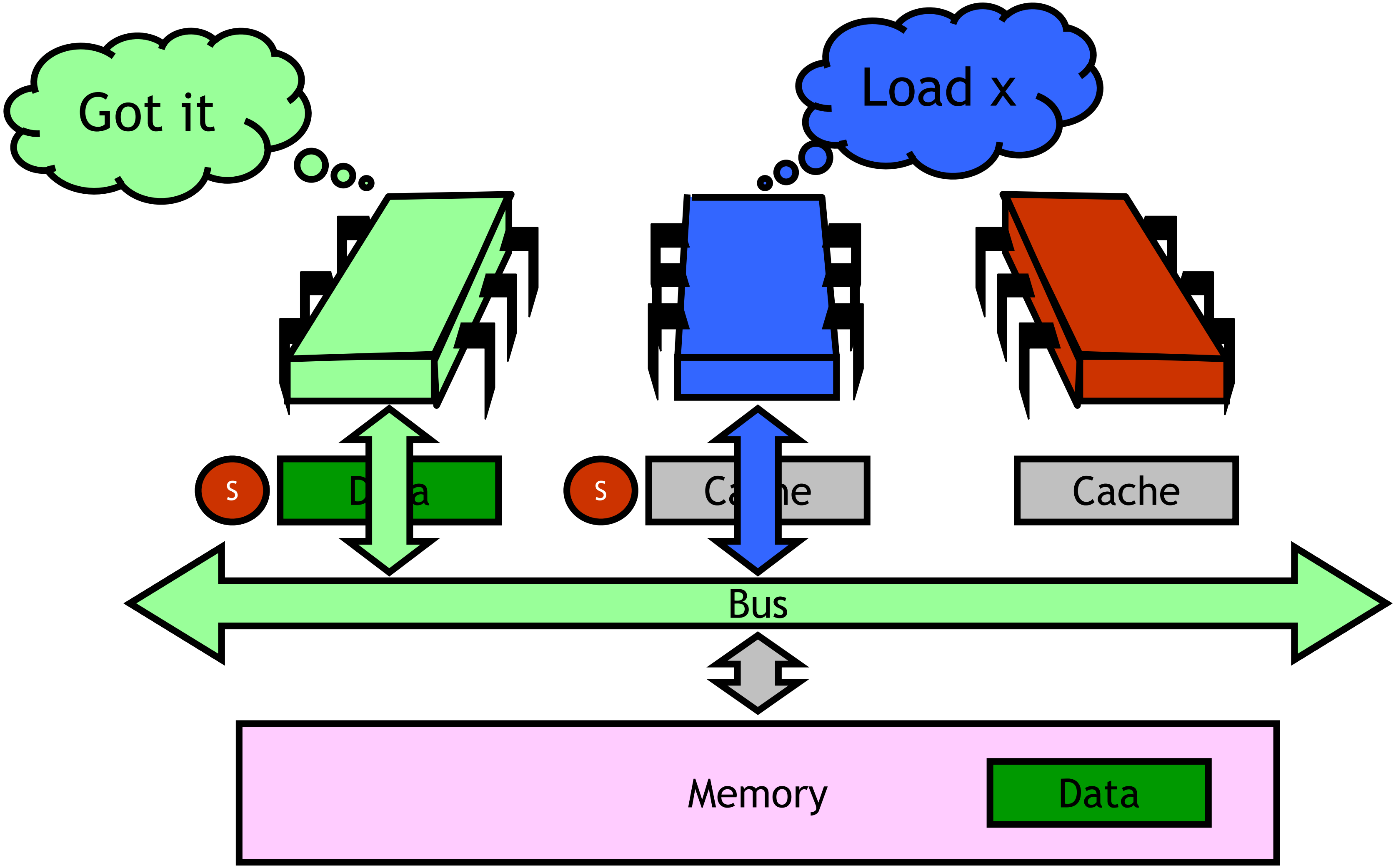
MESI

- Modified
  - Have modified cached data, must write back to memory
- Exclusive
  - Not modified, I have only copy
- Shared
  - Not modified, may be cached elsewhere
- Invalid
  - Cache contents not meaningful

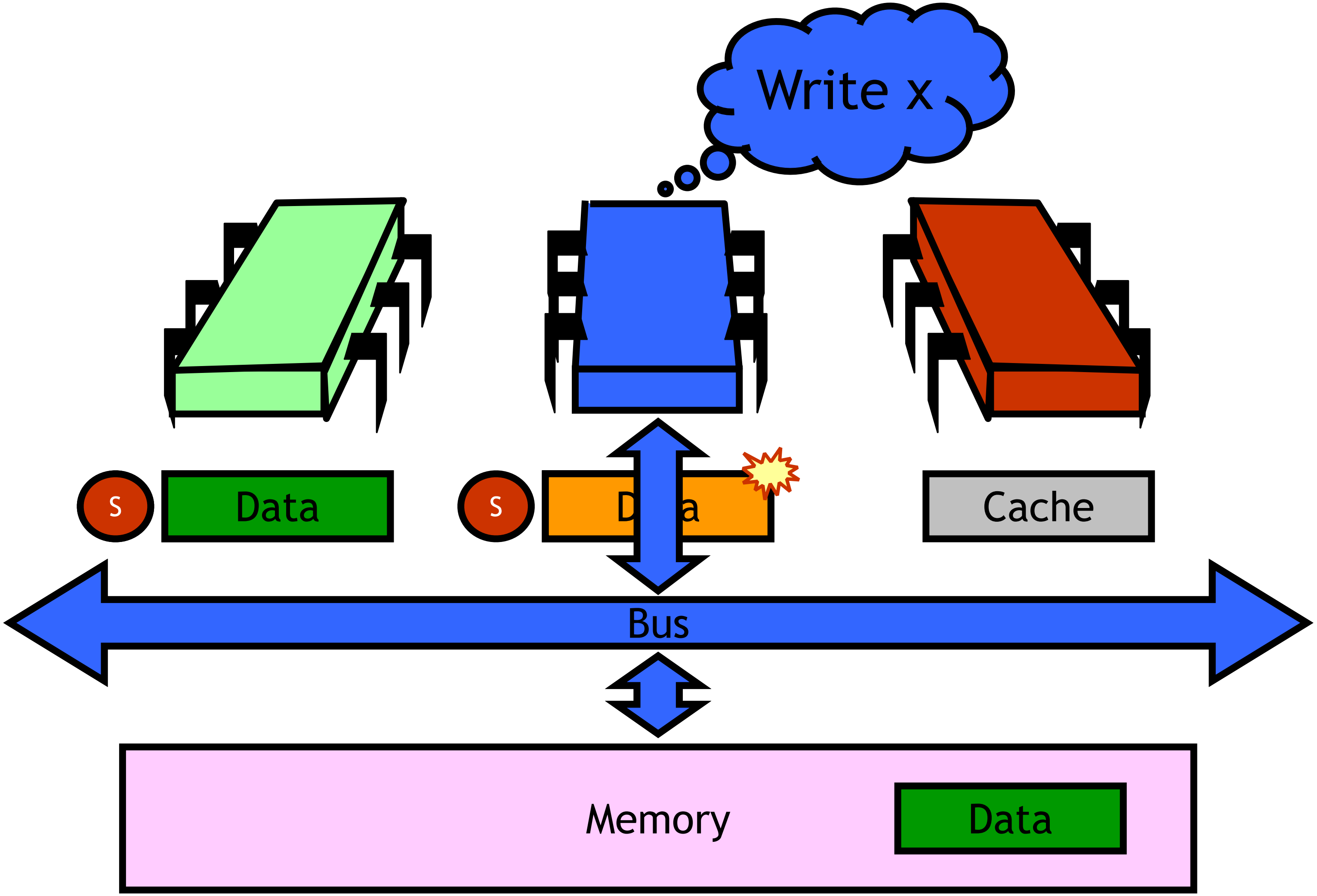
Processor Issues Load Request



2nd Processor Loads Data



Write Data: Write-Through Cache



Write-Through Caches

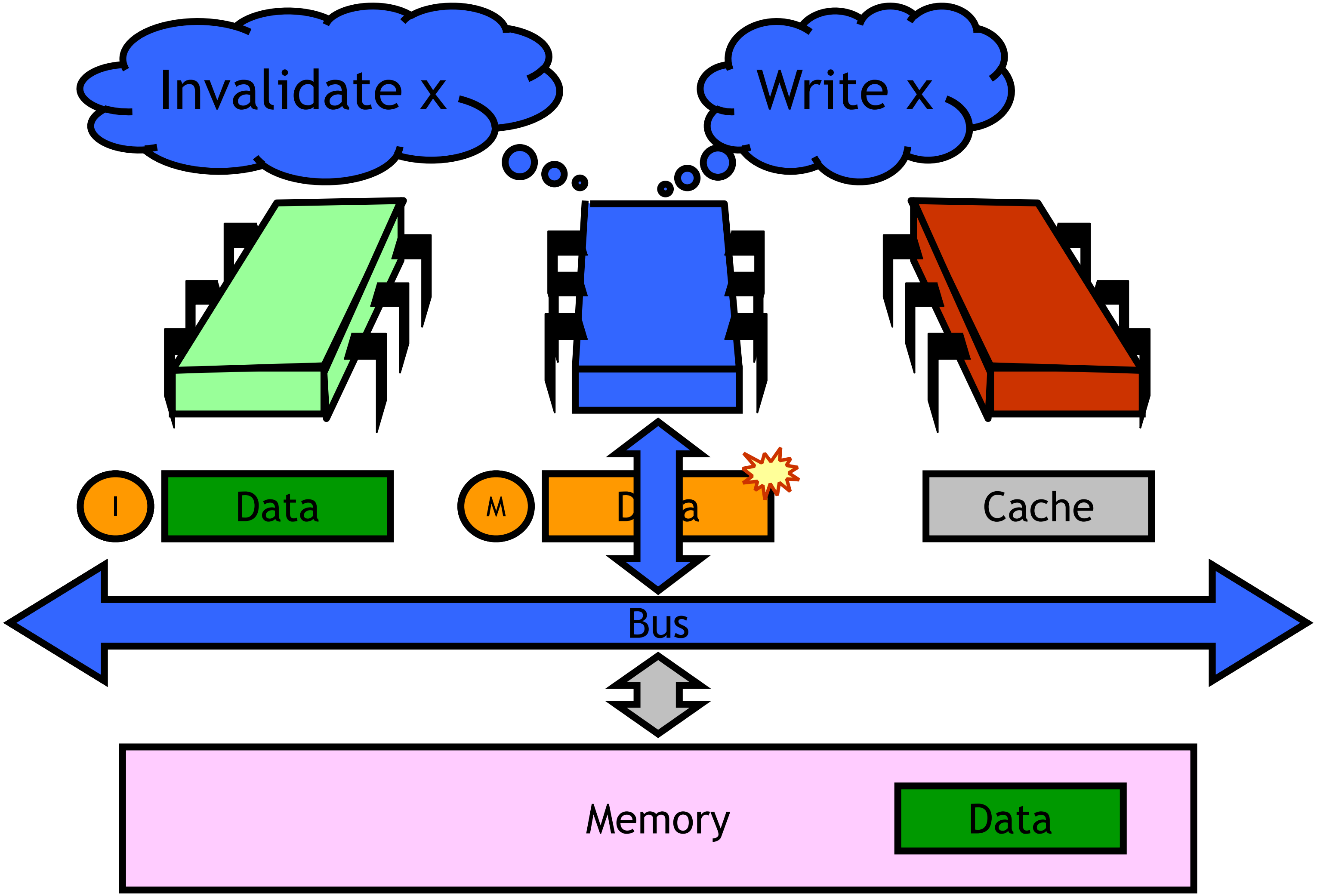
- Immediately broadcast changes
- Good
  - Memory, caches always agree
  - More read hits, maybe
- Bad
  - Bus traffic on all writes
  - Most writes to unshared data
  - For example, loop indexes…

(1)

Write-Back Caches

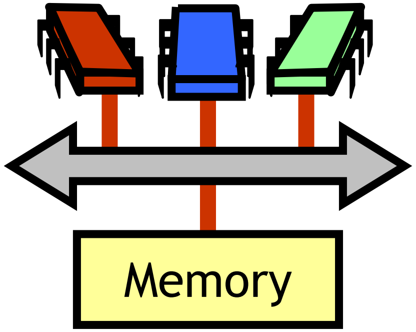
- Accumulate changes in cache
  - Invalidate other copies
- Write back when line evicted
  - Need the cache for something else
  - Another processor wants it

Write Data: Write-Back Cache

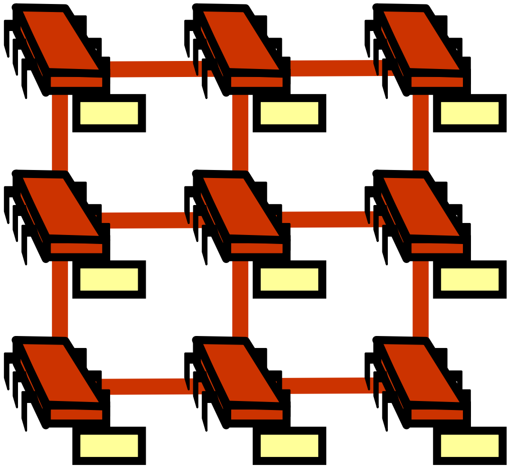


SMP vs. NUMA

- **SMP**: symmetric multiprocessor
- **NUMA**: non-uniform memory access
- **CC-NUMA**: cache-coherent NUMA



SMP



NUMA

Caches

- There are not only L1 and L2
  - but also L3 (shared amongst cores)
  - sometimes L4 (e.g., DRAM chip on core)

- Cores share L3
  - a core needs less L3, others get more
  - better usage of cache - faster applications

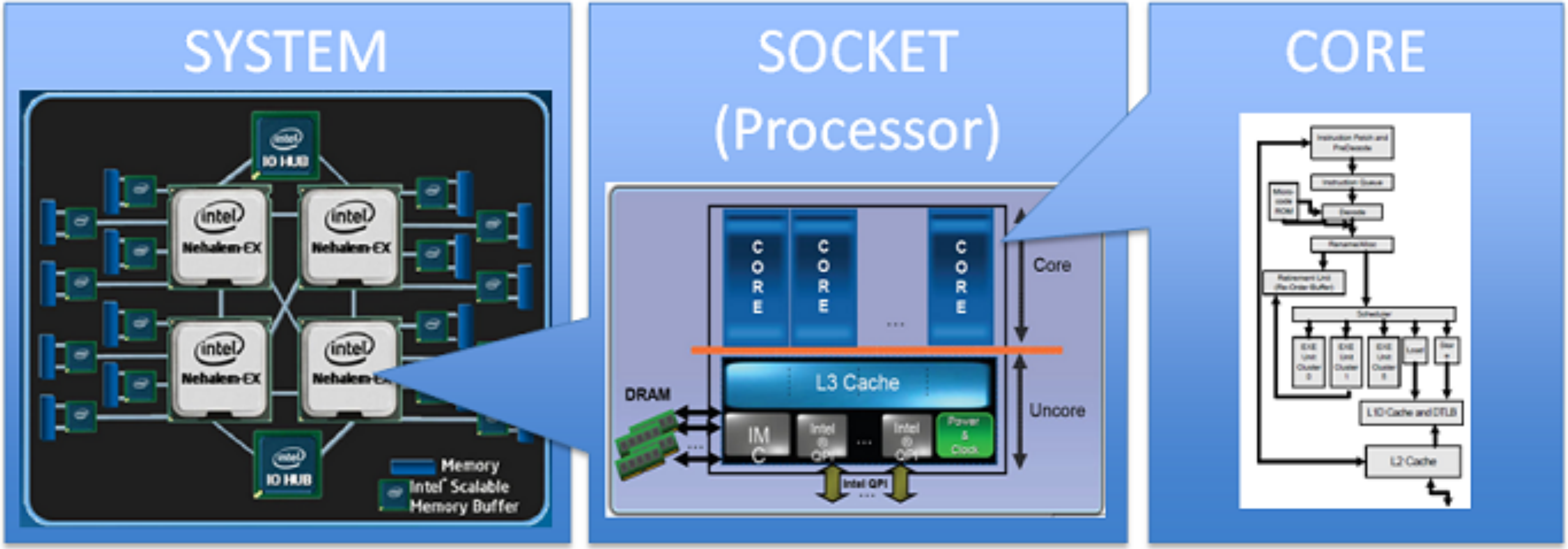
Cache Monitoring & Allocation

- Modern CPUs contain up to 30MB (L1-L3 cache)
- Run lots of applications concurrently

- Problem:
  - cache hit rate has large impact on runtime
  - cache hit rate depends on cache usage of other applications

Architecture

- Servers have 1 - 4 sockets (- some even more)

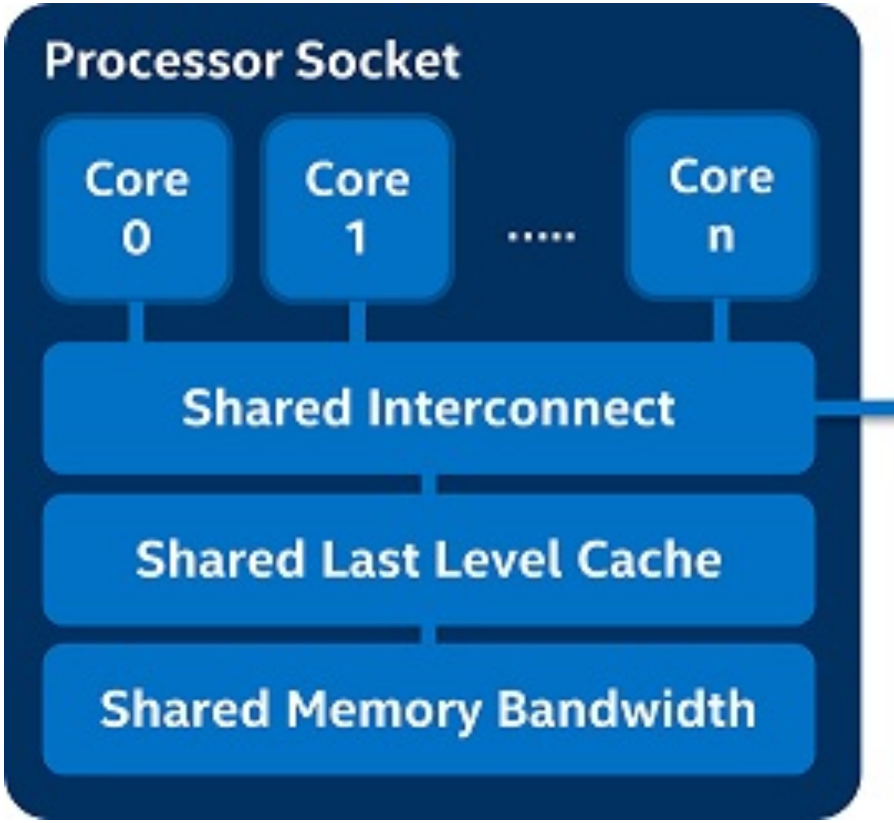


(C) Intel

Architecture

- Cores easier to partition
  - amongst applications

- Difficult to partition:
  - Last level cache (L3)
  - Memory bandwidth



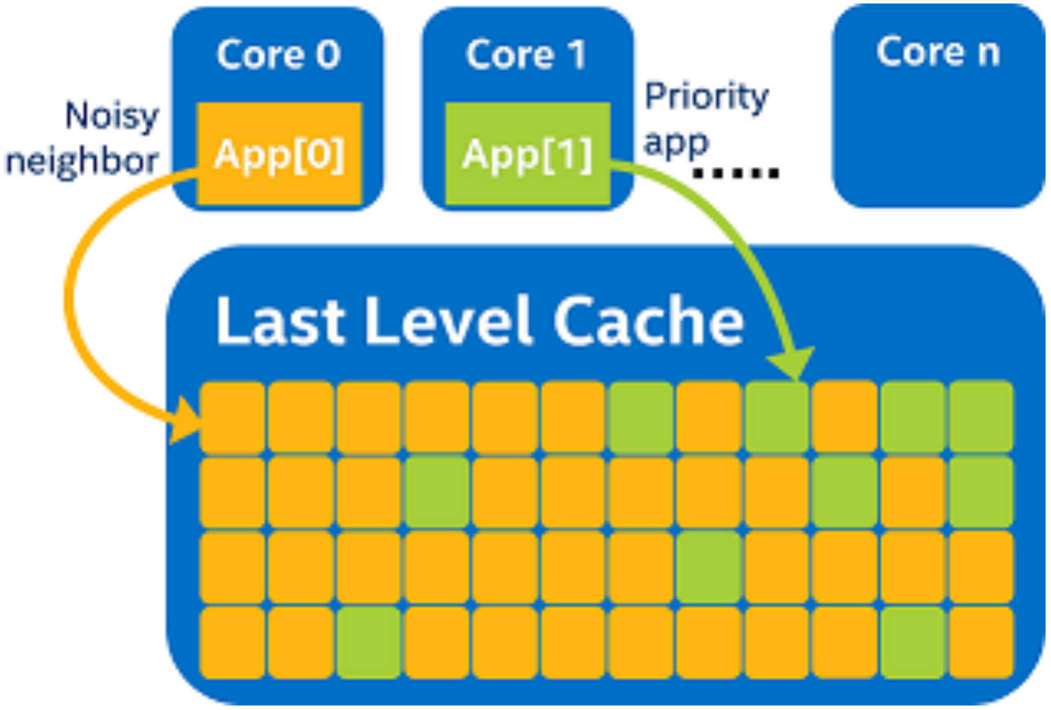
Problem

- **Problem:**
  - Unpredictable performance

- **Example:**
  - application runs great on my private machine
  - but runs badly on shared server or cloud
    - when we have noisy neighbors

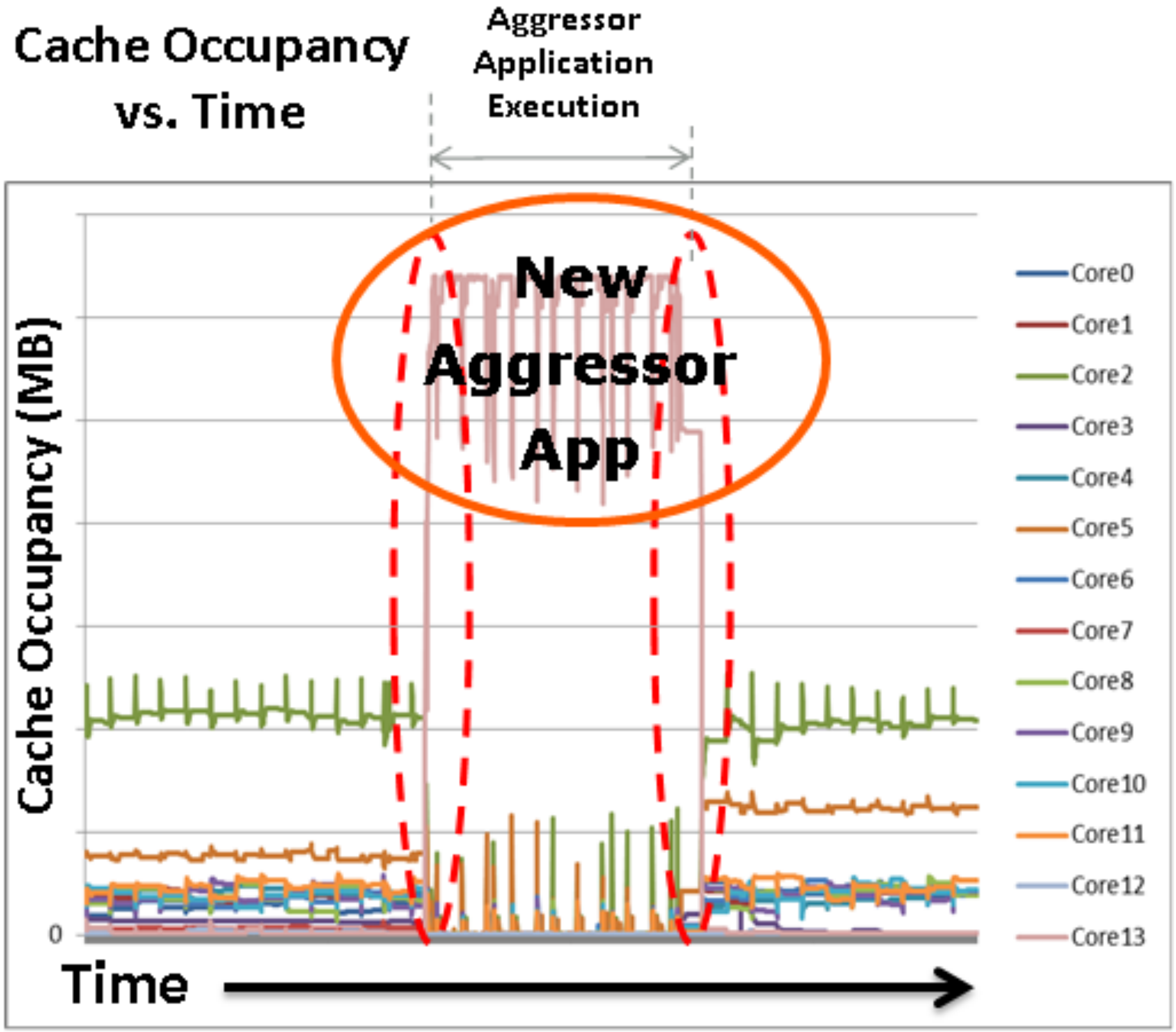
Example

- **Problem:**
  - other application reduces cache hit rate
- **Approaches:**
  - reserve cache for applications
  - migrate noisy neighbors to different socket

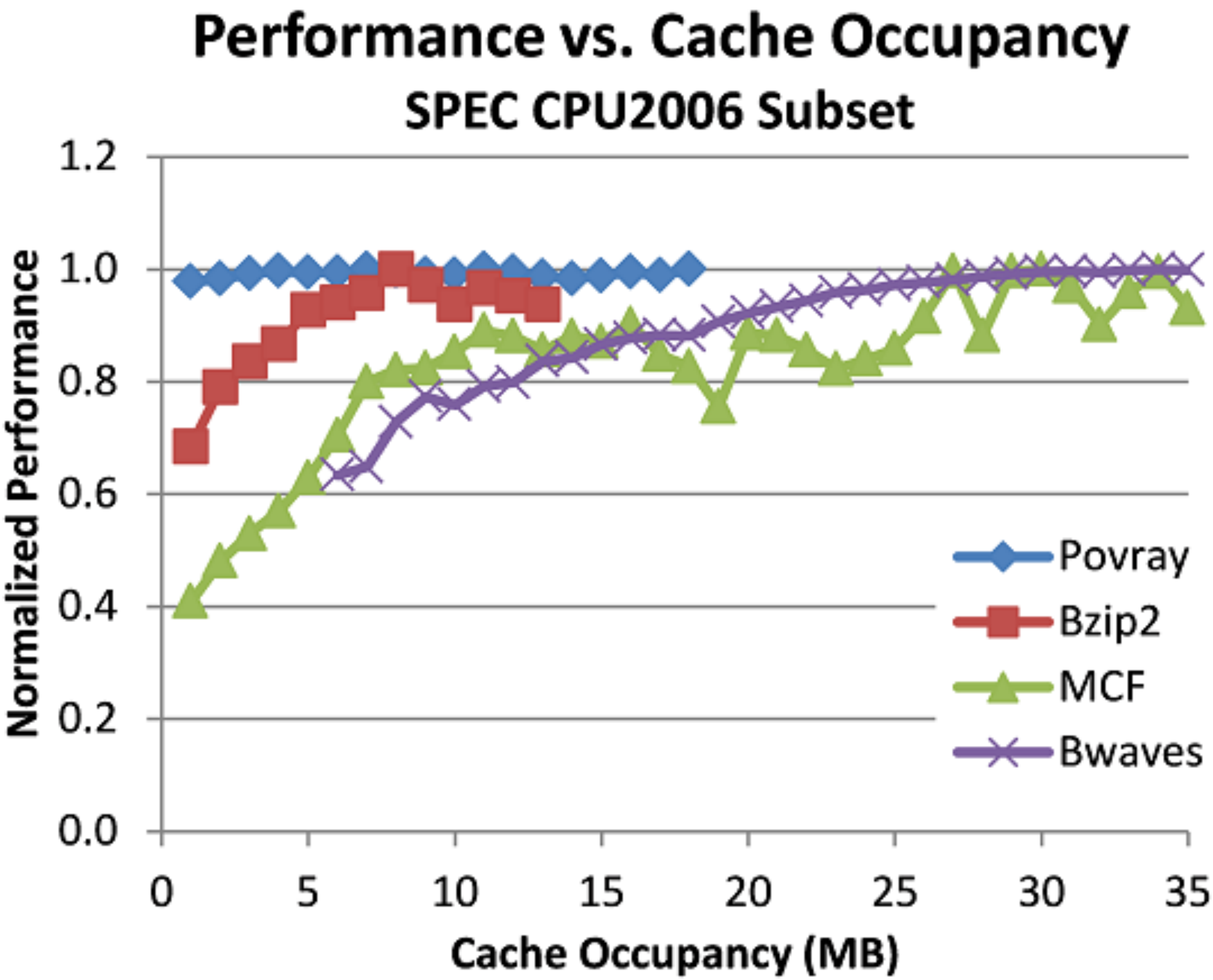


(C) Intel

Monitoring Cache Usage (CMT)



Does this matter?



- Some apps are cache insensitive
- Some are ok with limited cache
- Some need as much cache as possible

Cache Allocation

- Ensure that an application (or VM)
  - has sufficient cache space

- **Approach:**
  - reserve cache entries for given class of VMs/ applications/threads
  - entity does not compete with applications in another class

Memory Bandwidth Monitoring

- Memory can easily partitioned between applications (by the OS / VMM)
- **Problem:**
  - limited memory bandwidth can limit application performance
- **Approach:**
  - Monitor app memory bandwidth usage
  - Migrate noisy apps to different socket

Summary

- Hardware is more complex than our ideal model
  - Weaken consistency for performance
  - Sharing of resources

- Cache influences performance
  - noisy neighbors can reduce performance

- Memory bandwidth influences performance
  - noisy neighbors can reduce performance