Multiprocessor Architecture Basics

Companion slides for

The Art of Multiprocessor Programming

by Maurice Herlihy & Nir Shavit

Multiprocessor Architecture

- Abstract models are (mostly) OK to understand algorithm correctness
- To understand how concurrent algorithms perform
- You need to understand something about multiprocessor architectures

Pieces

- Processors
- Threads
- Interconnect
- Memory
- Caches

Design an Urban Messenger Service in 1980

- Downtown Manhattan
- Should you use
  - Cars (1980 Buicks, 15 MPG)?
  - Bicycles (hire recent graduates)?
- Better use bicycles

Technology Changes

- Since 1980, car technology has changed enormously
  - Better mileage (hybrid cars, 35 MPG)
  - More reliable
- Should you rethink your Manhattan messenger service?

Processors

- Cycle:
  - Fetch and execute one instruction
- Cycle times change
  - 1980: 10 million cycles/sec
  - 2005: 3,000 million cycles/sec

Computer Architecture

- Measure time in cycles
  - Absolute cycle times change
- Memory access: ~100s of cycles
  - Changes slowly
  - Mostly gets worse

Threads

- Execution of a sequential program
- Software, not hardware
- A processor can run a thread
- Put it aside
  - Thread does I/O
  - Thread runs out of time
- Run another thread

Interconnect

- Bus
  - Like a tiny Ethernet
  - Broadcast medium
  - Connects
    - Processors to memory
    - Processors to processors
- Network
  - Tiny LAN
  - Mostly used on large machines

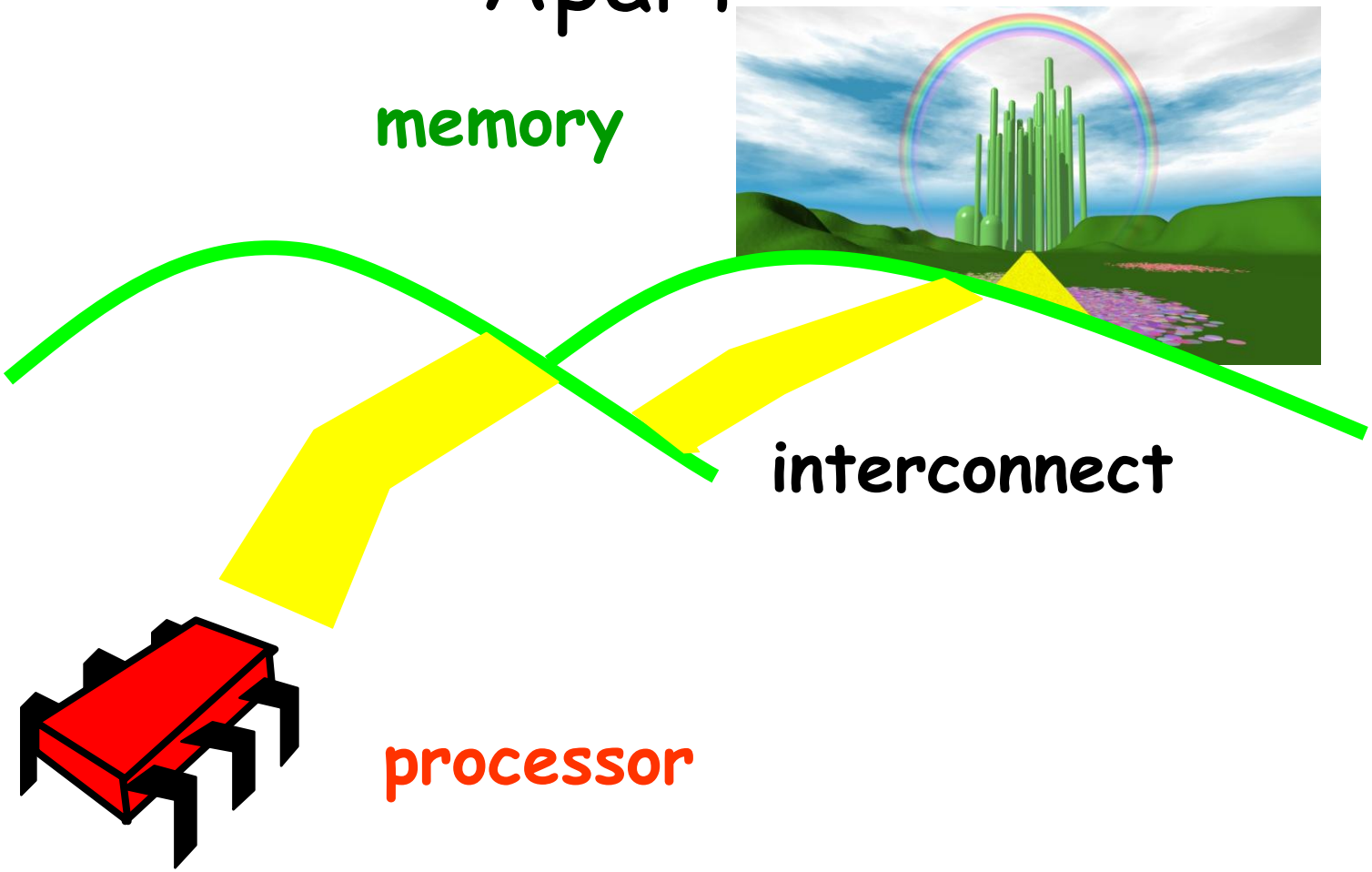
Interconnect

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

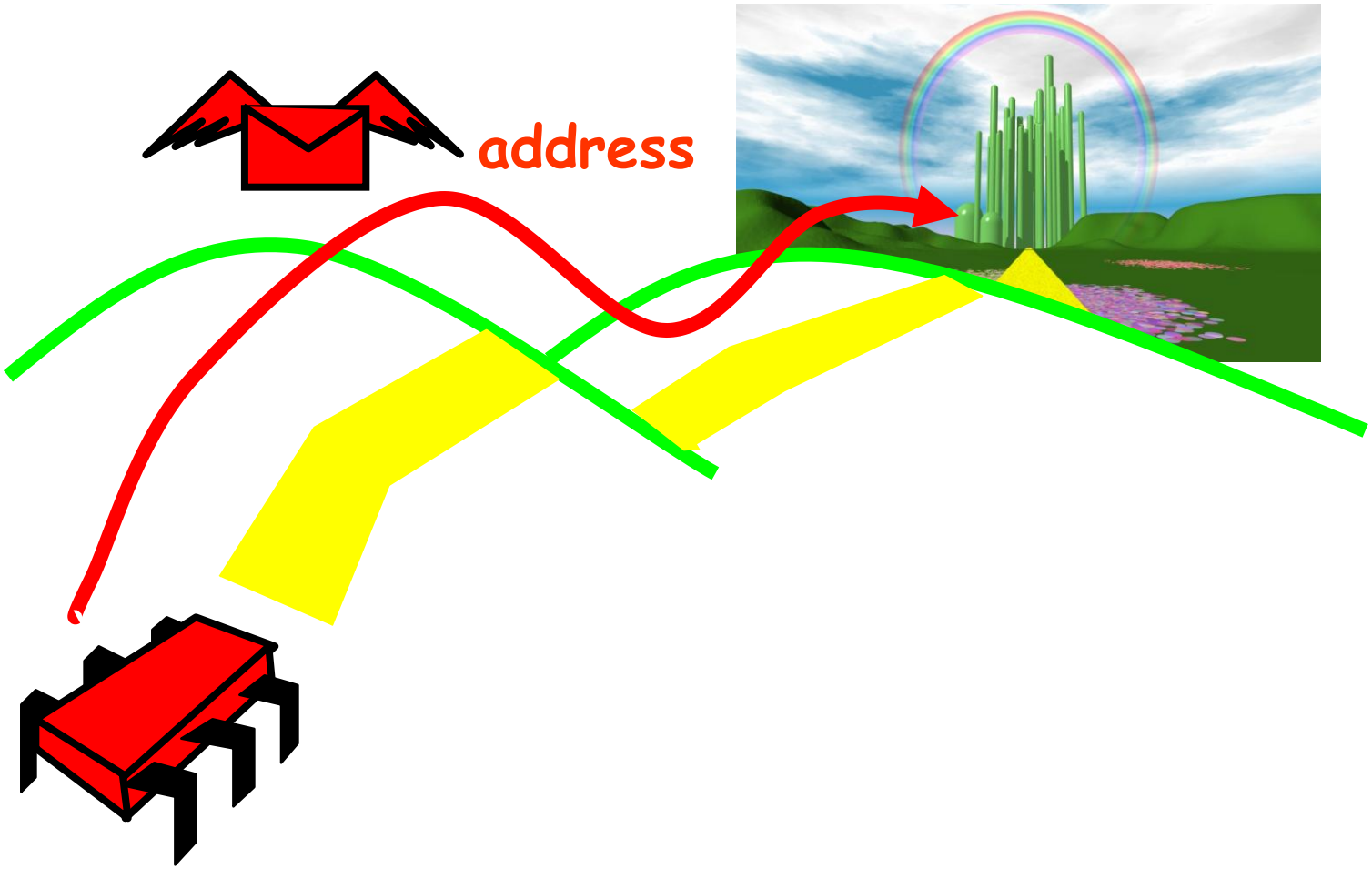
Analogy

- You work in an office
- When you leave for lunch, someone else takes over your office.
- If you don't take a break, a security guard shows up and escorts you to the cafeteria.
- When you return, you may get a different office

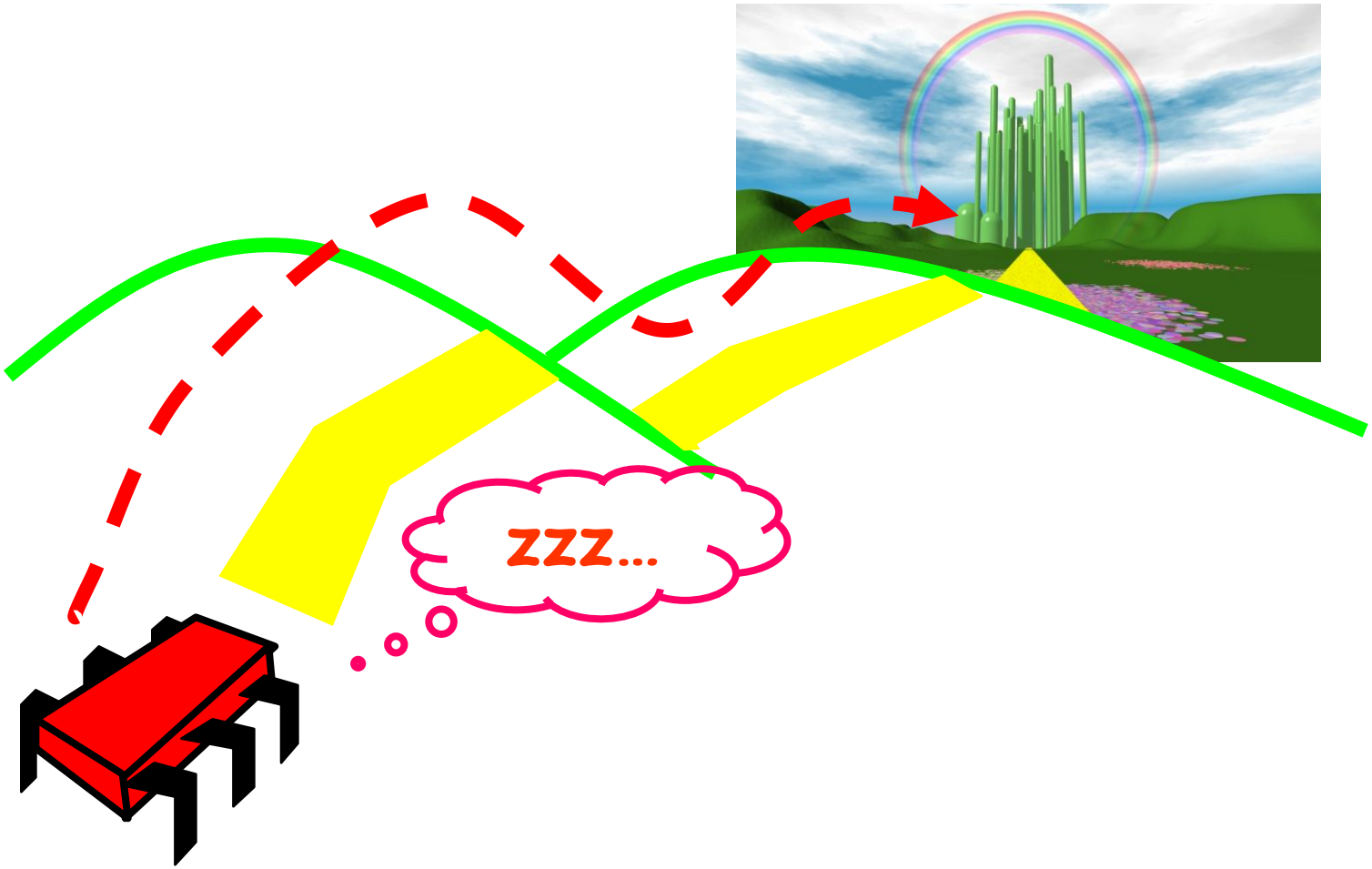
Processor and Memory are Far Apart



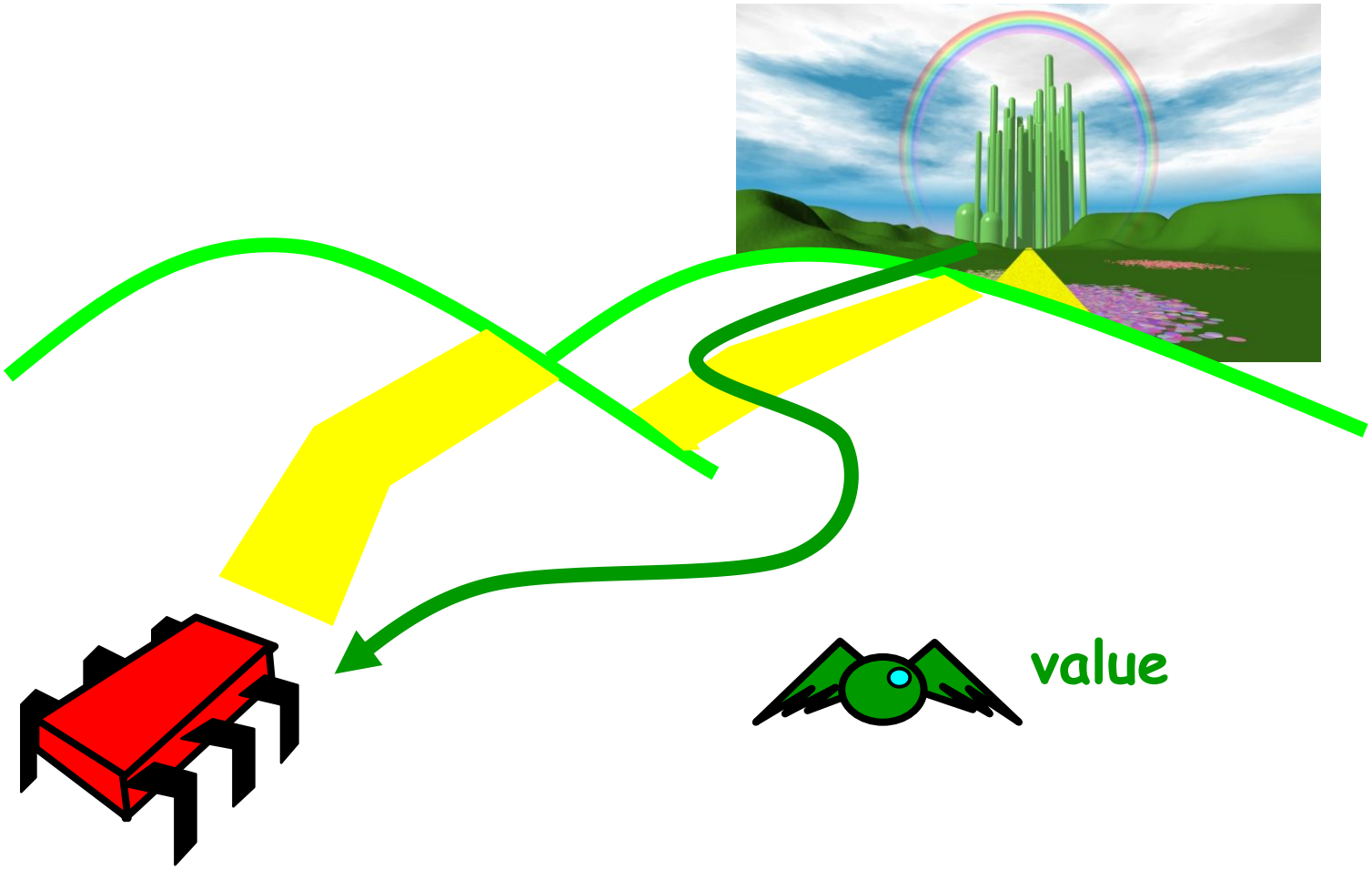
Reading from Memory



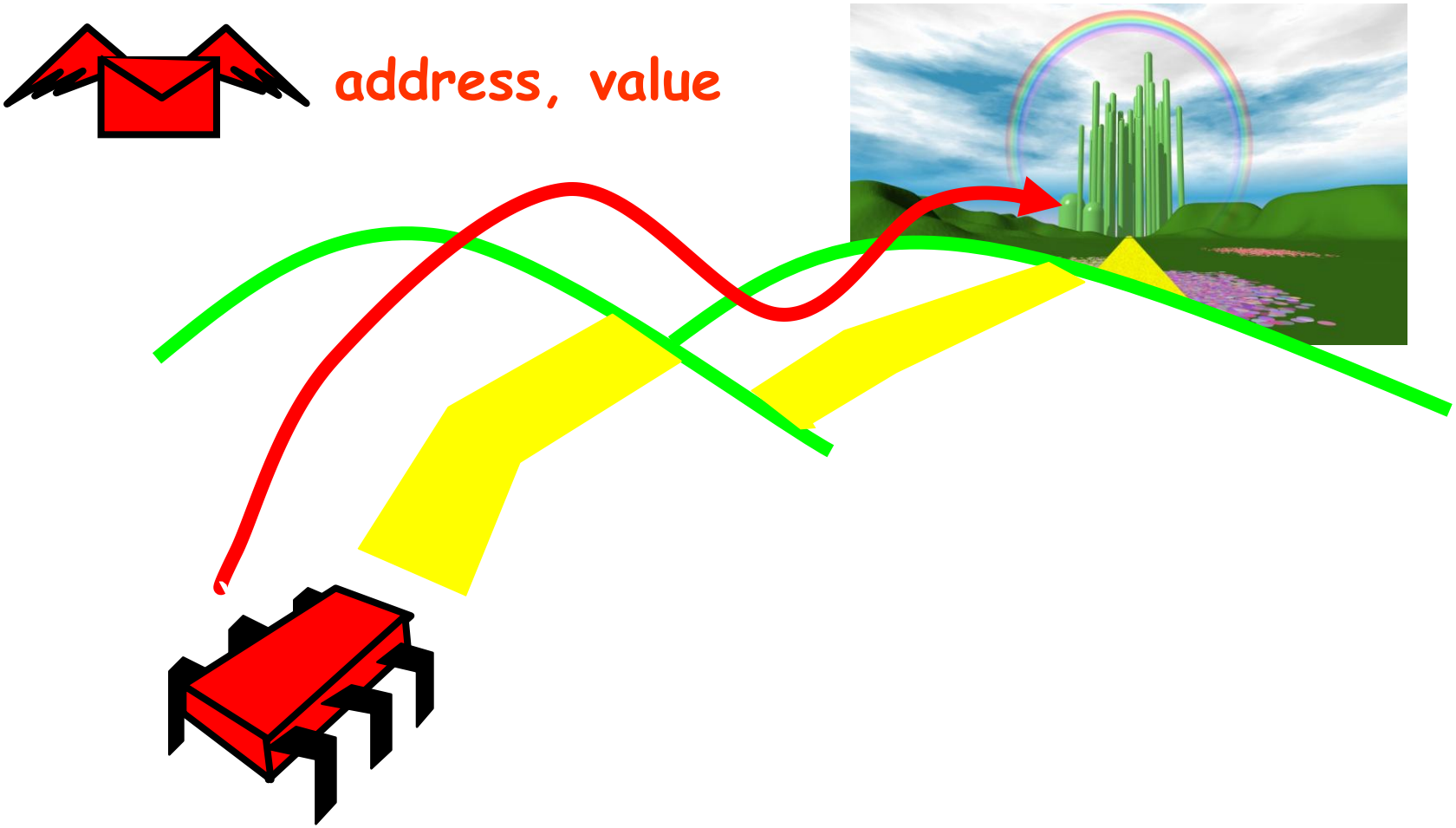
Reading from Memory



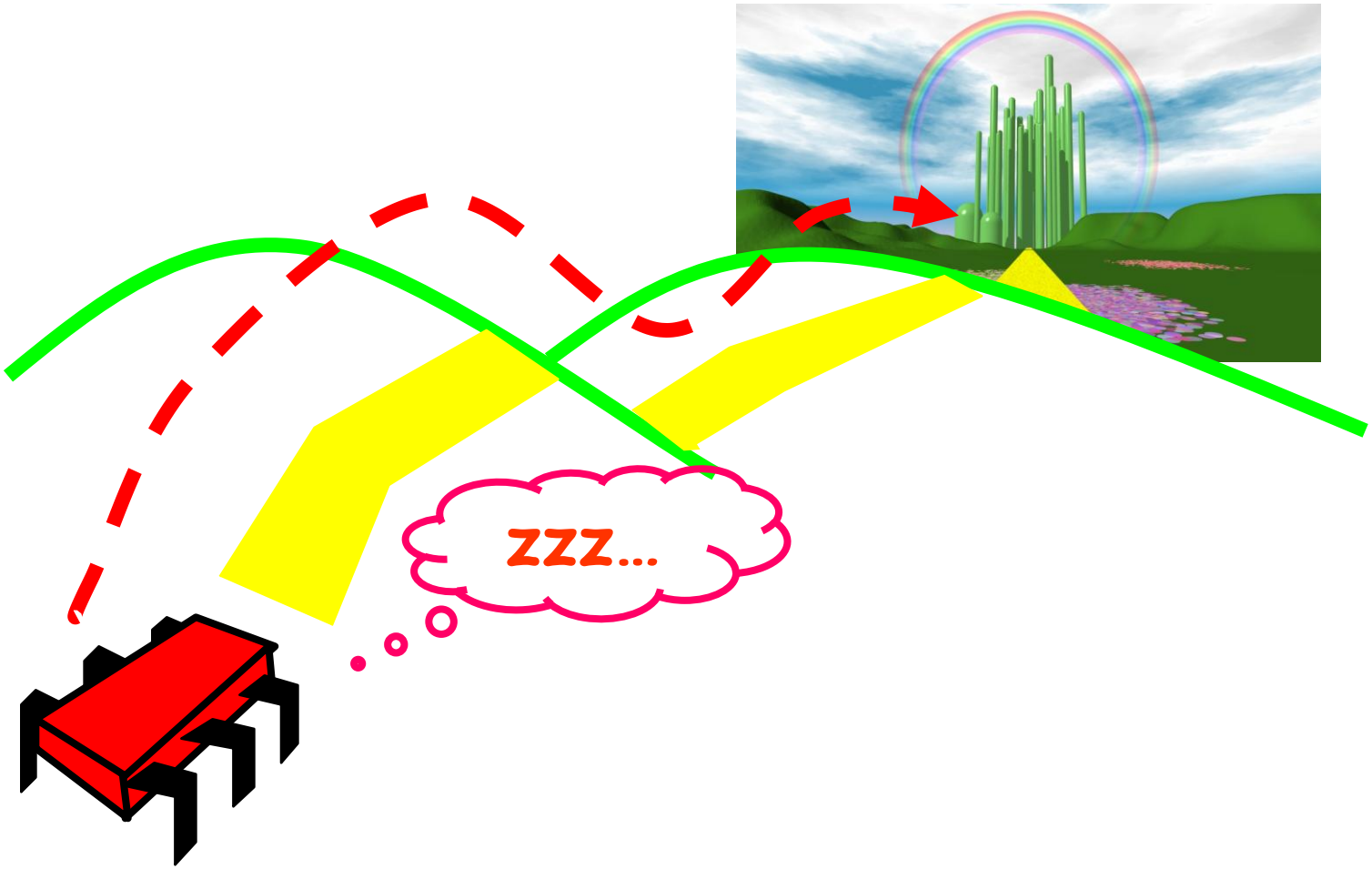
Reading from Memory



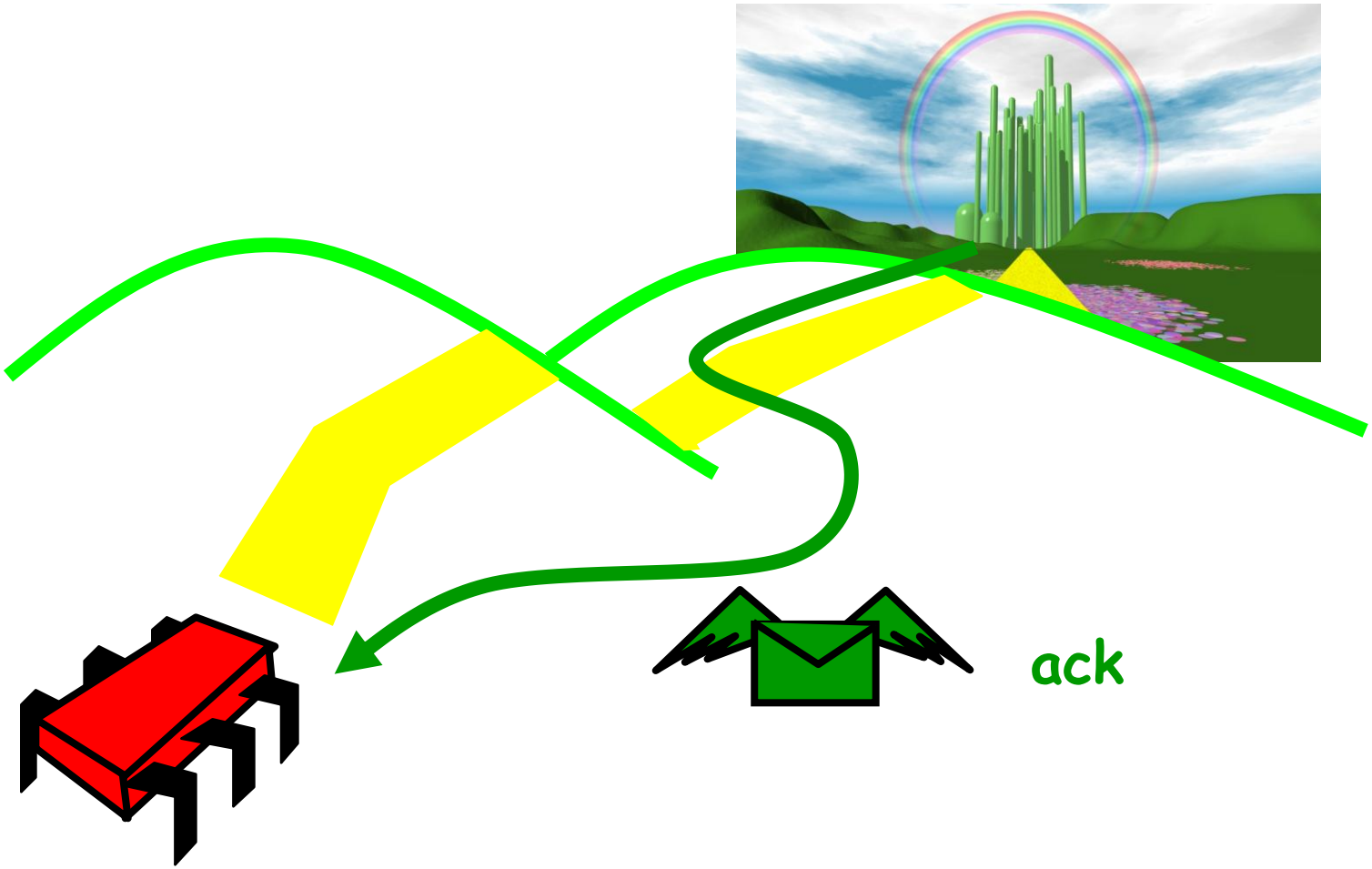
Writing to Memory



Writing to Memory



Writing to Memory



Remote Spinning

- Thread waits for a bit in memory to change
  - Maybe it tried to dequeue from an empty buffer
- Spins
  - Repeatedly rereads flag bit
- Huge waste of interconnect bandwidth

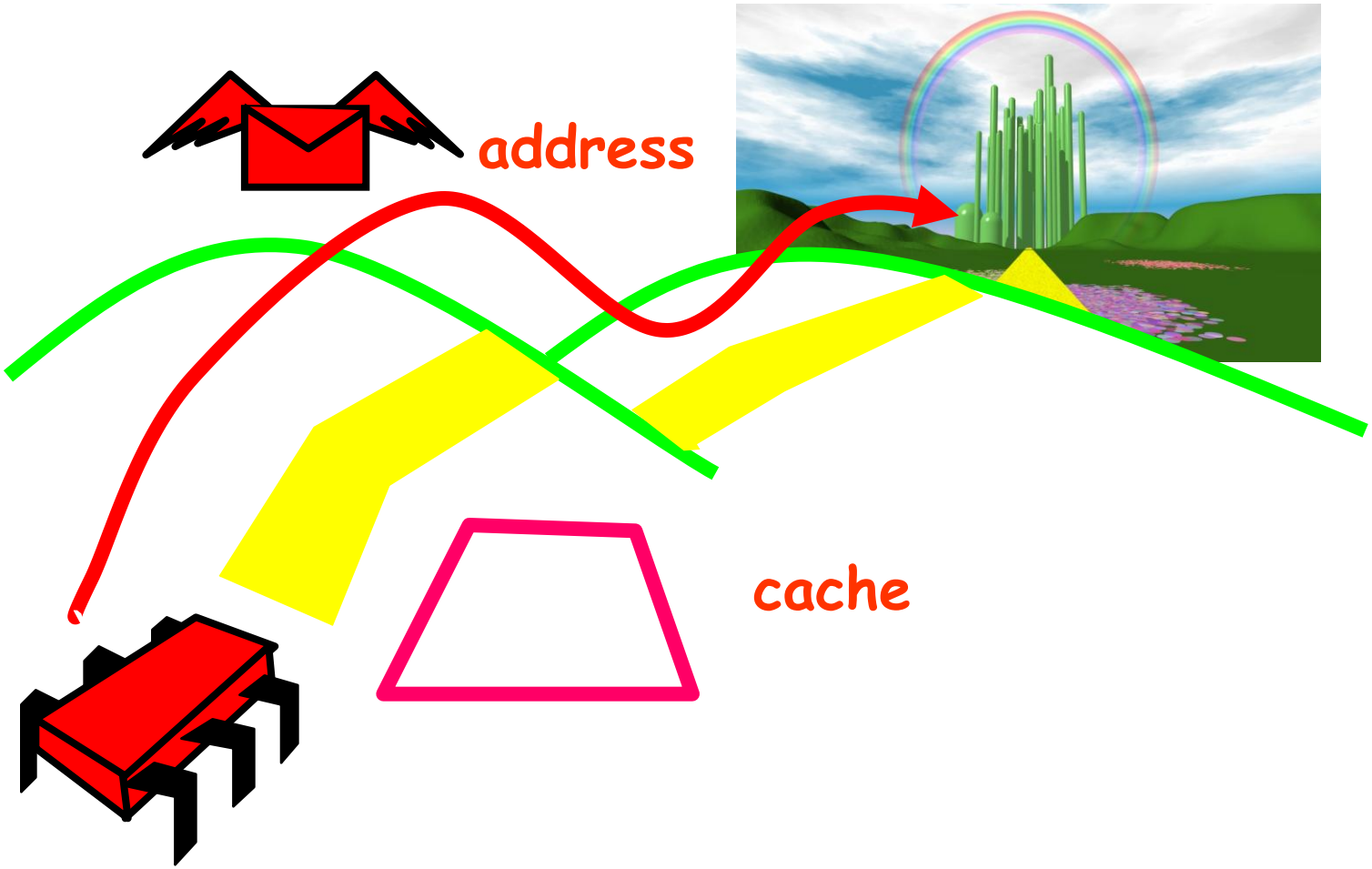
Analogy

- In the days before the Internet …
- Alice is writing a paper on aardvarks
- Sources are in university library
  - Request book by campus mail
  - Book arrives by return mail
  - Send it back when not in use
- She spends a lot of time in the mail room

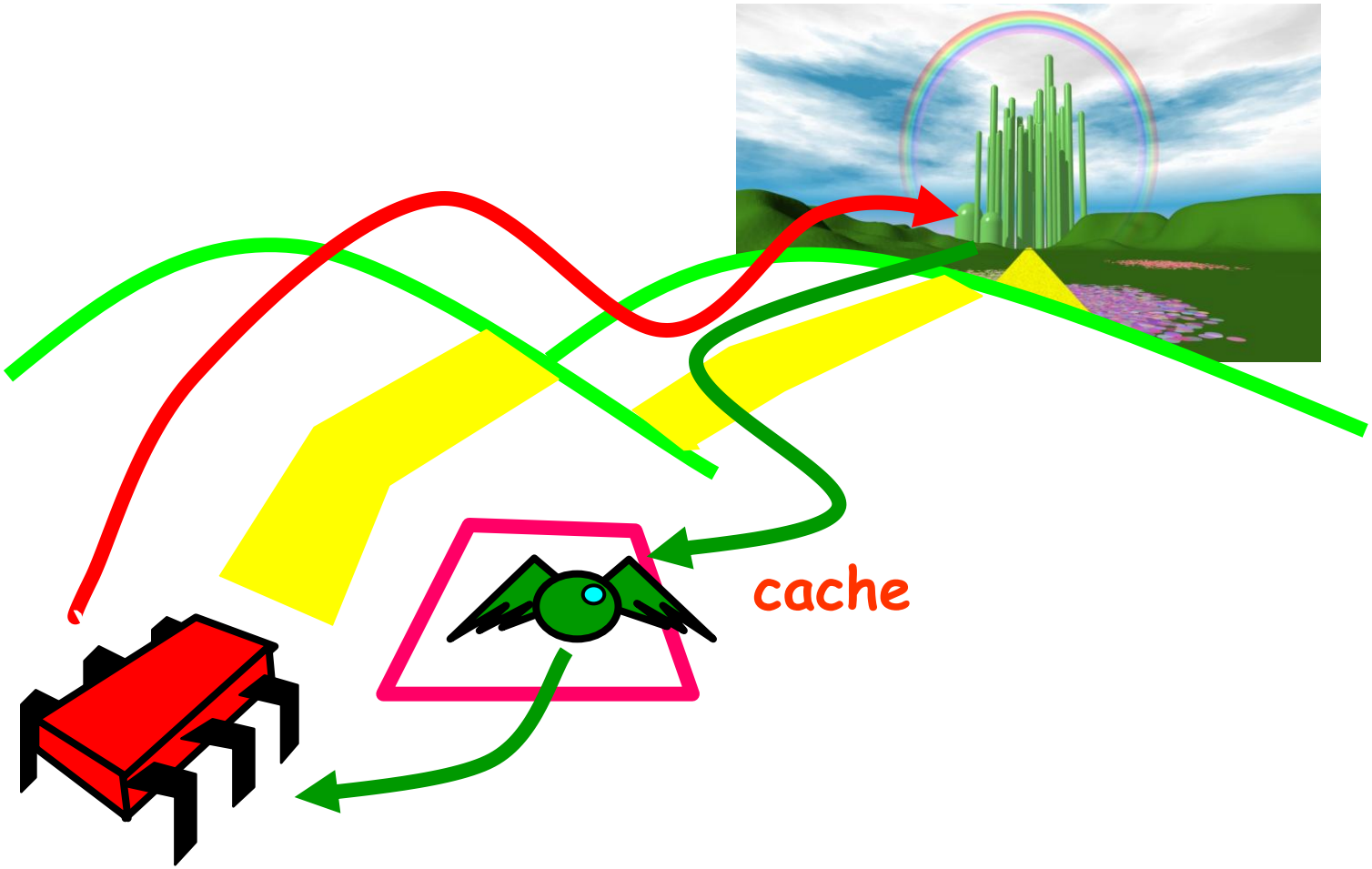
Analogy II

- Alice buys
  - A desk
    - In her office
    - To keep the books she is using now
  - A bookcase
    - in the hall
    - To keep the books she will need soon

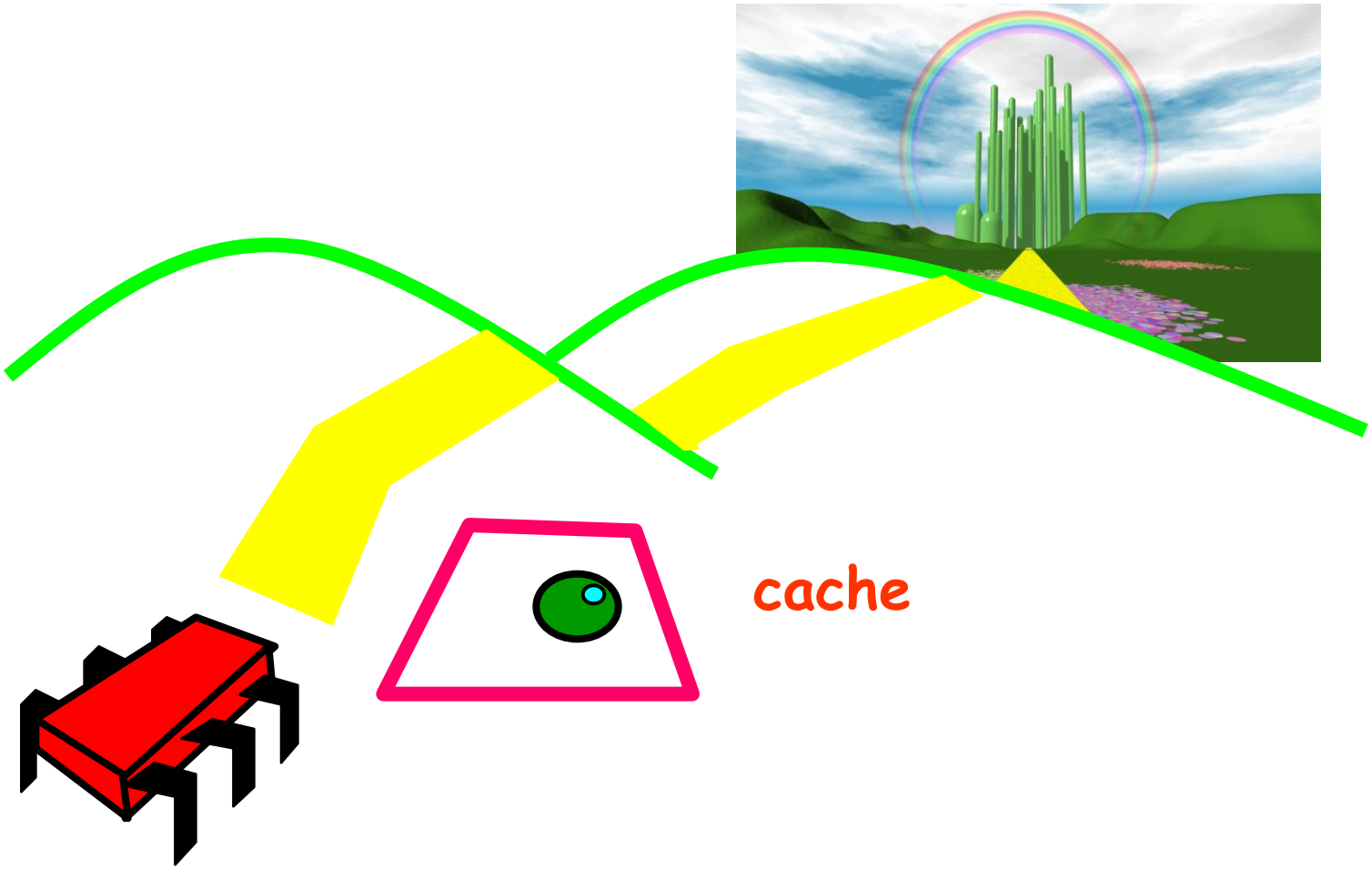
Cache: Reading from Memory



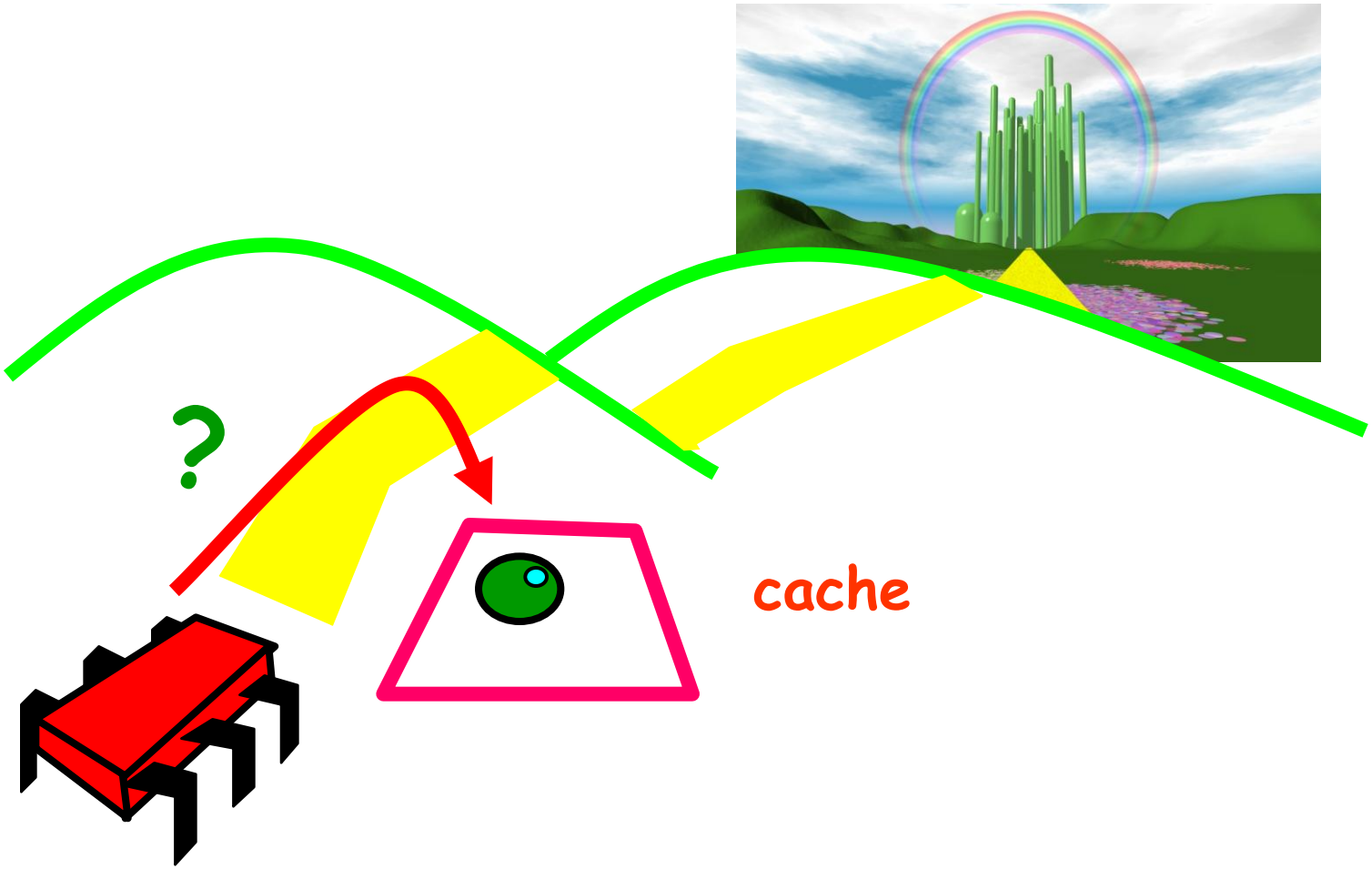
Cache: Reading from Memory



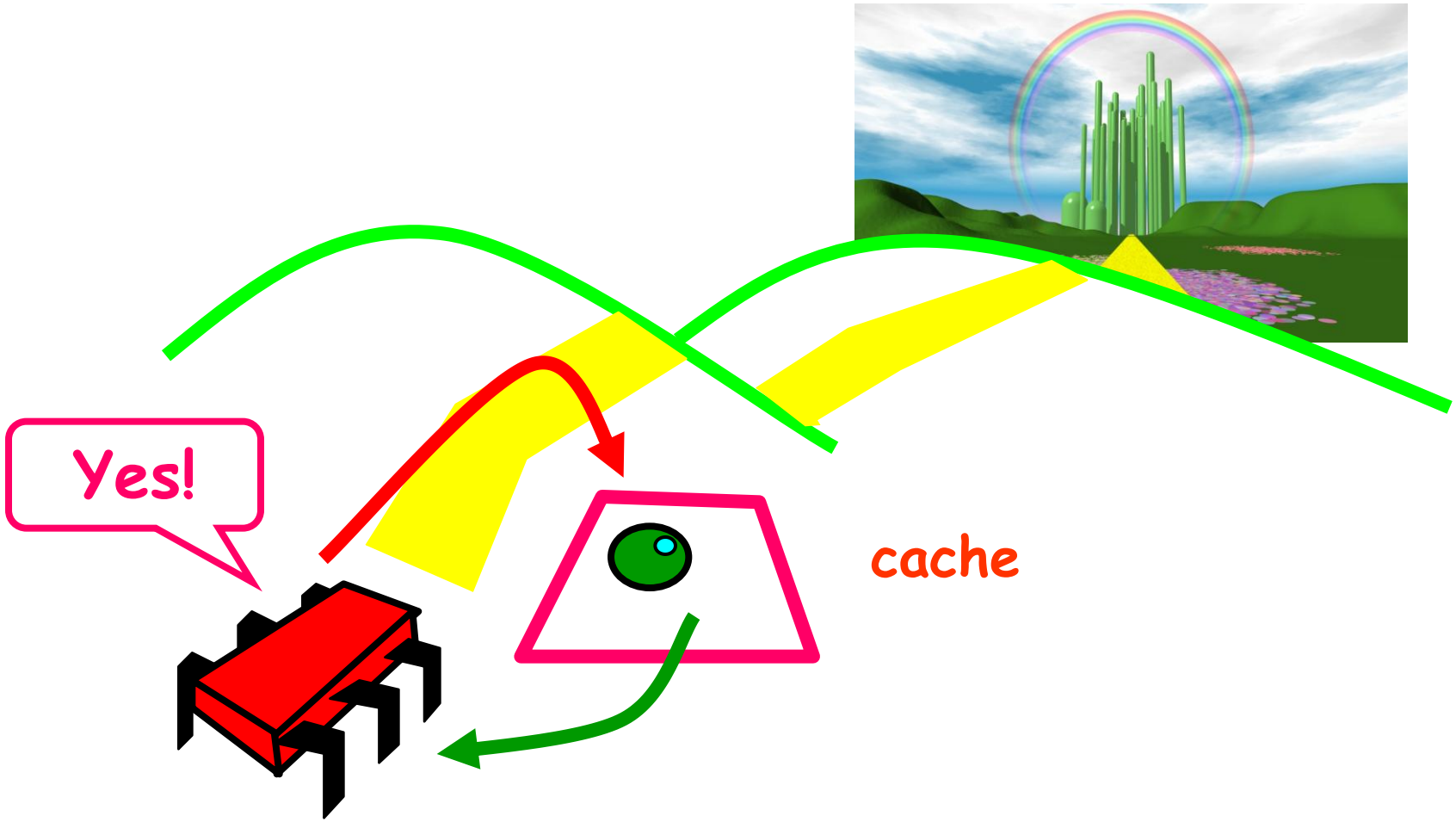
Cache: Reading from Memory



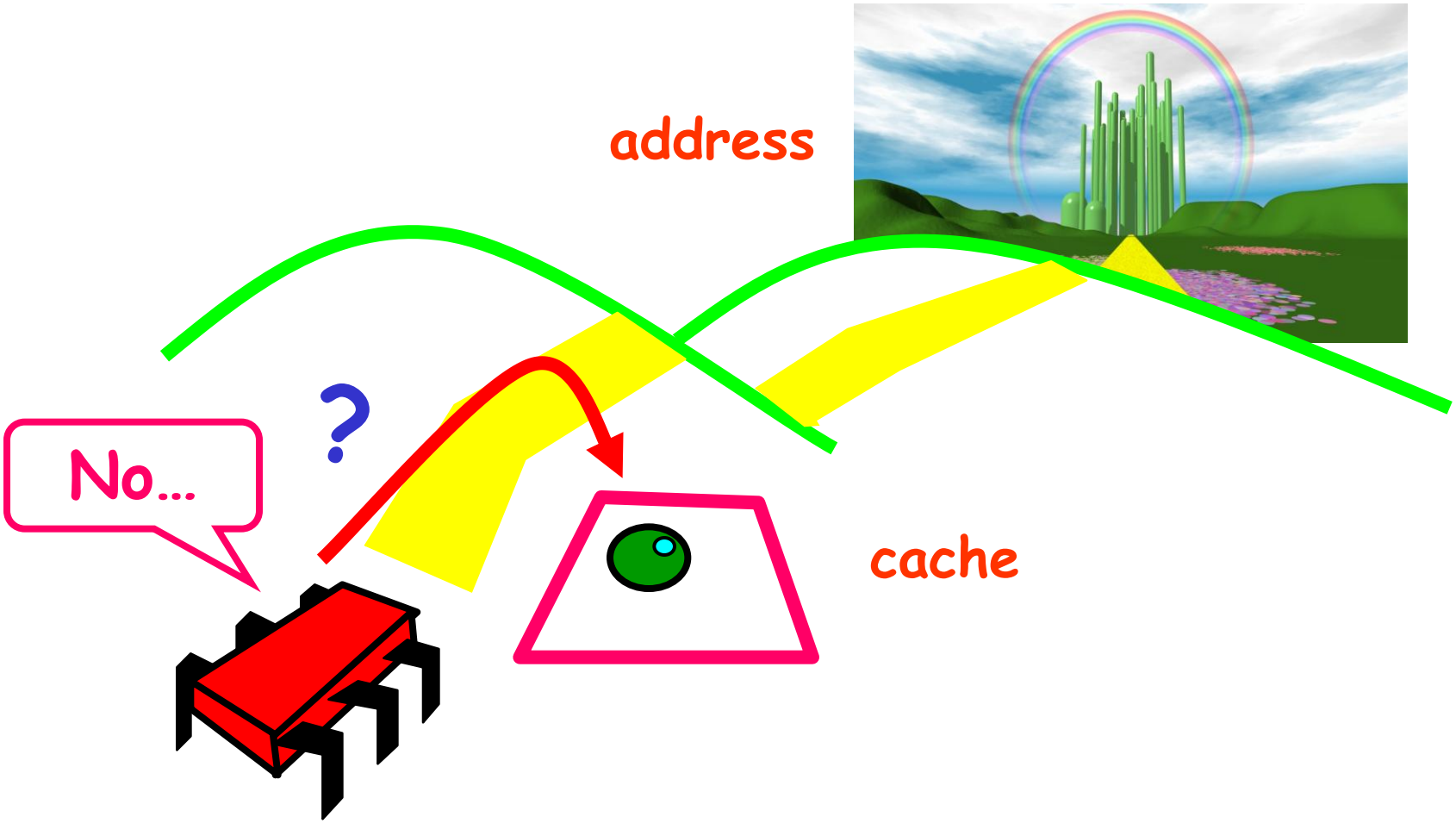
Cache Hit



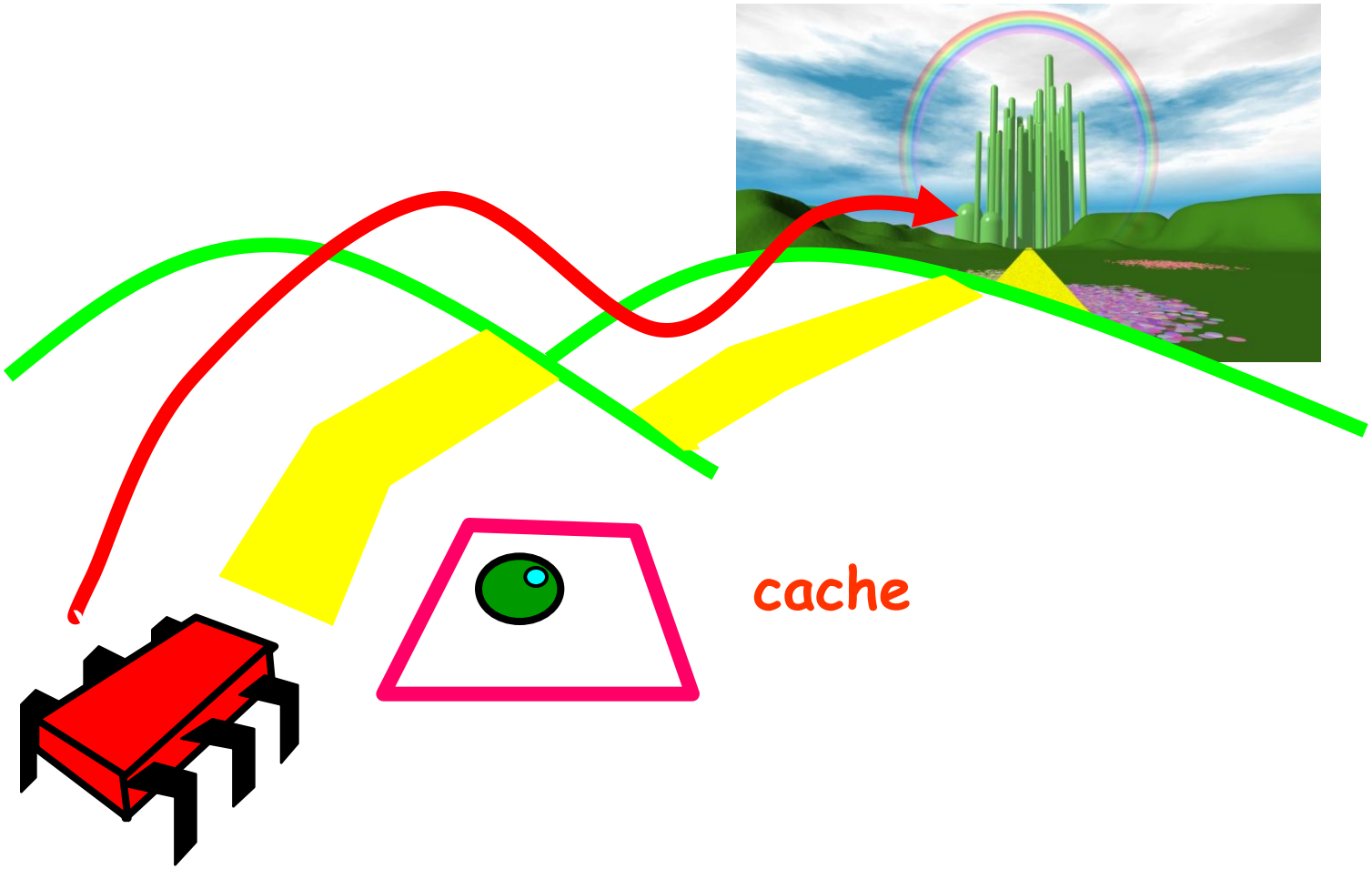
Cache Hit



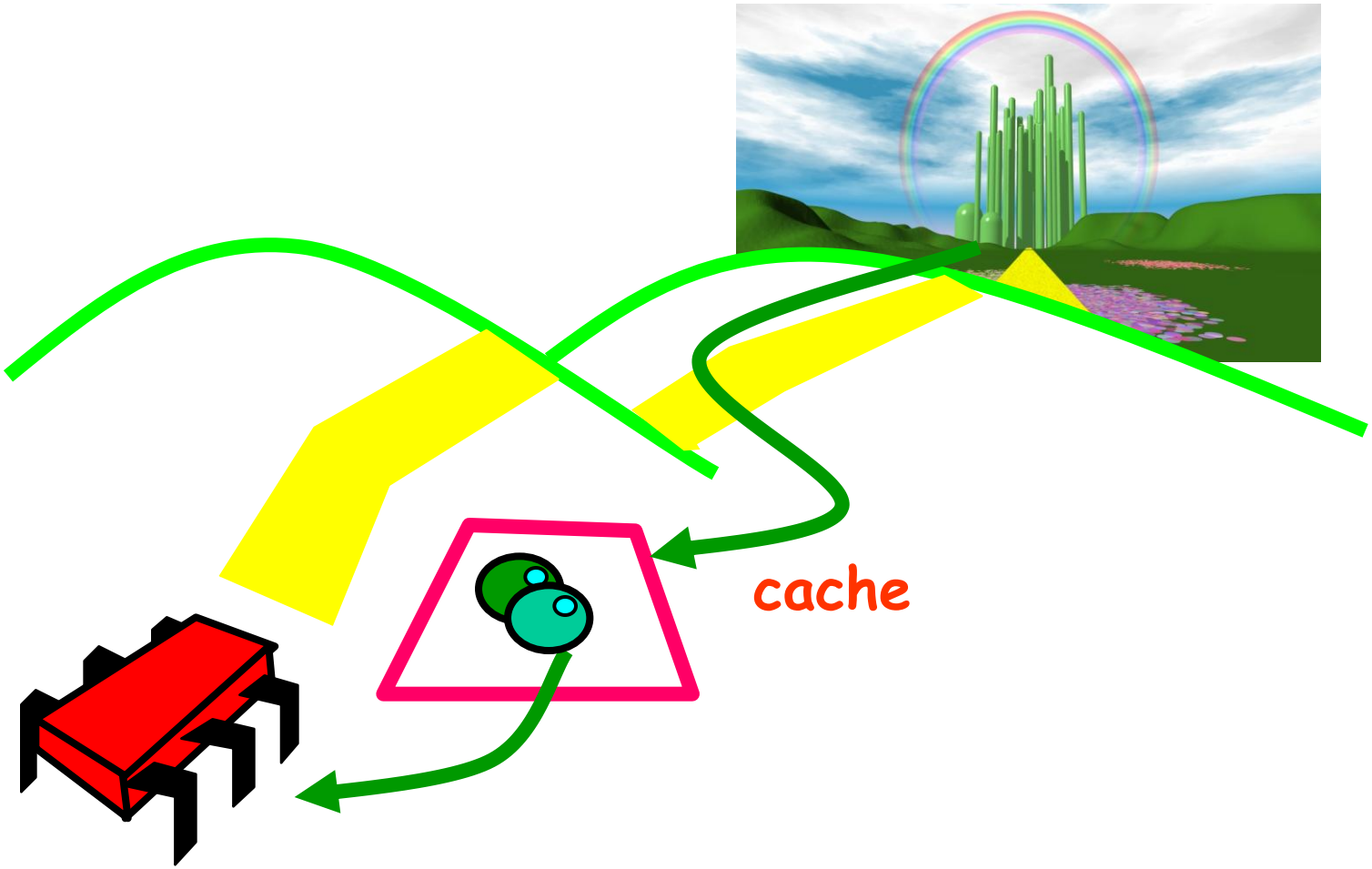
Cache Miss



Cache Miss



Cache Miss



Local Spinning

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

Granularity

- Caches operate at a larger granularity than a word
- Cache line: fixed-size block containing the address

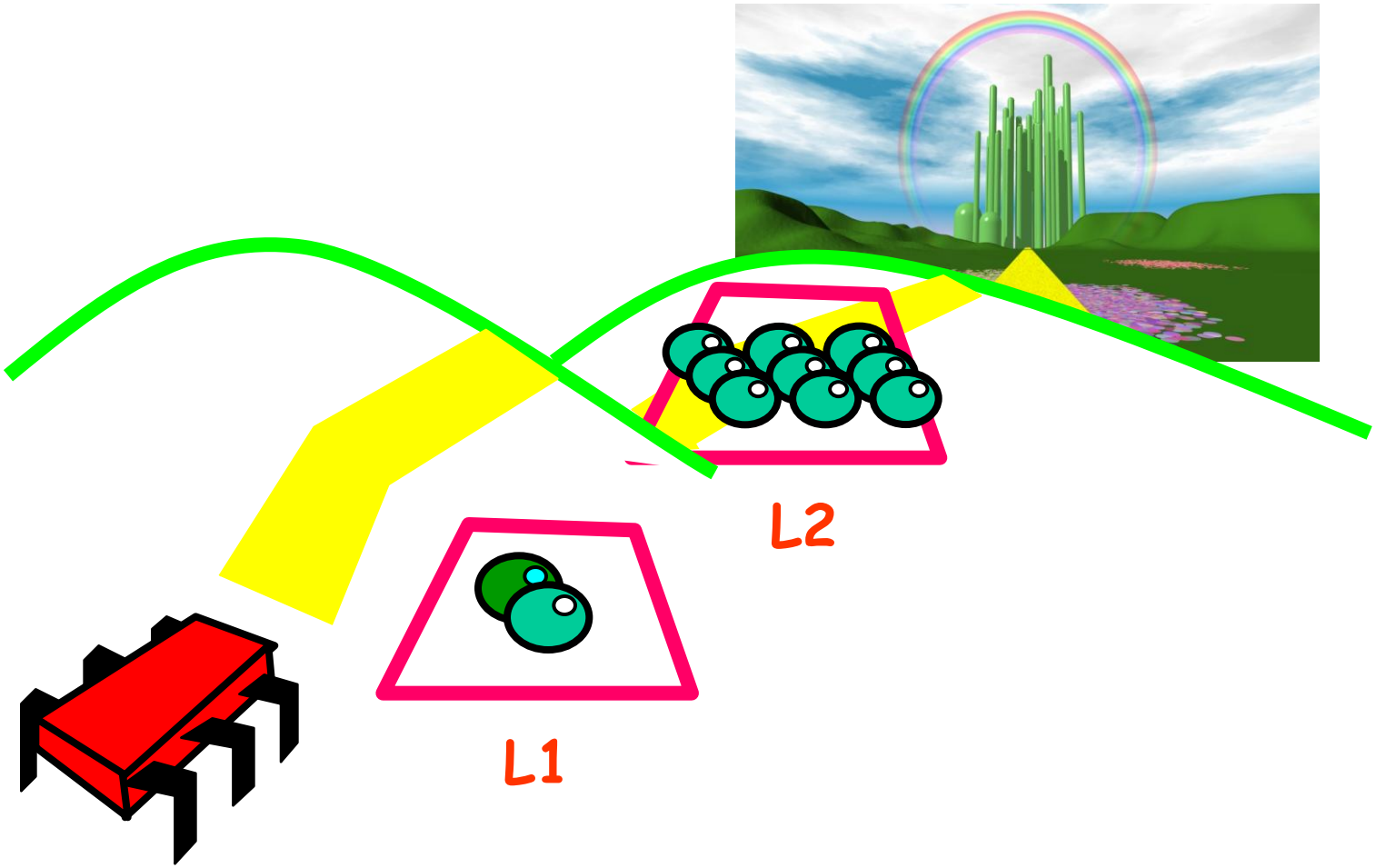
Locality

- If you use an address now, you will probably use it again soon
  - Fetch from cache, not memory
- If you use an address now, you will probably use a nearby address soon
  - In the same cache line

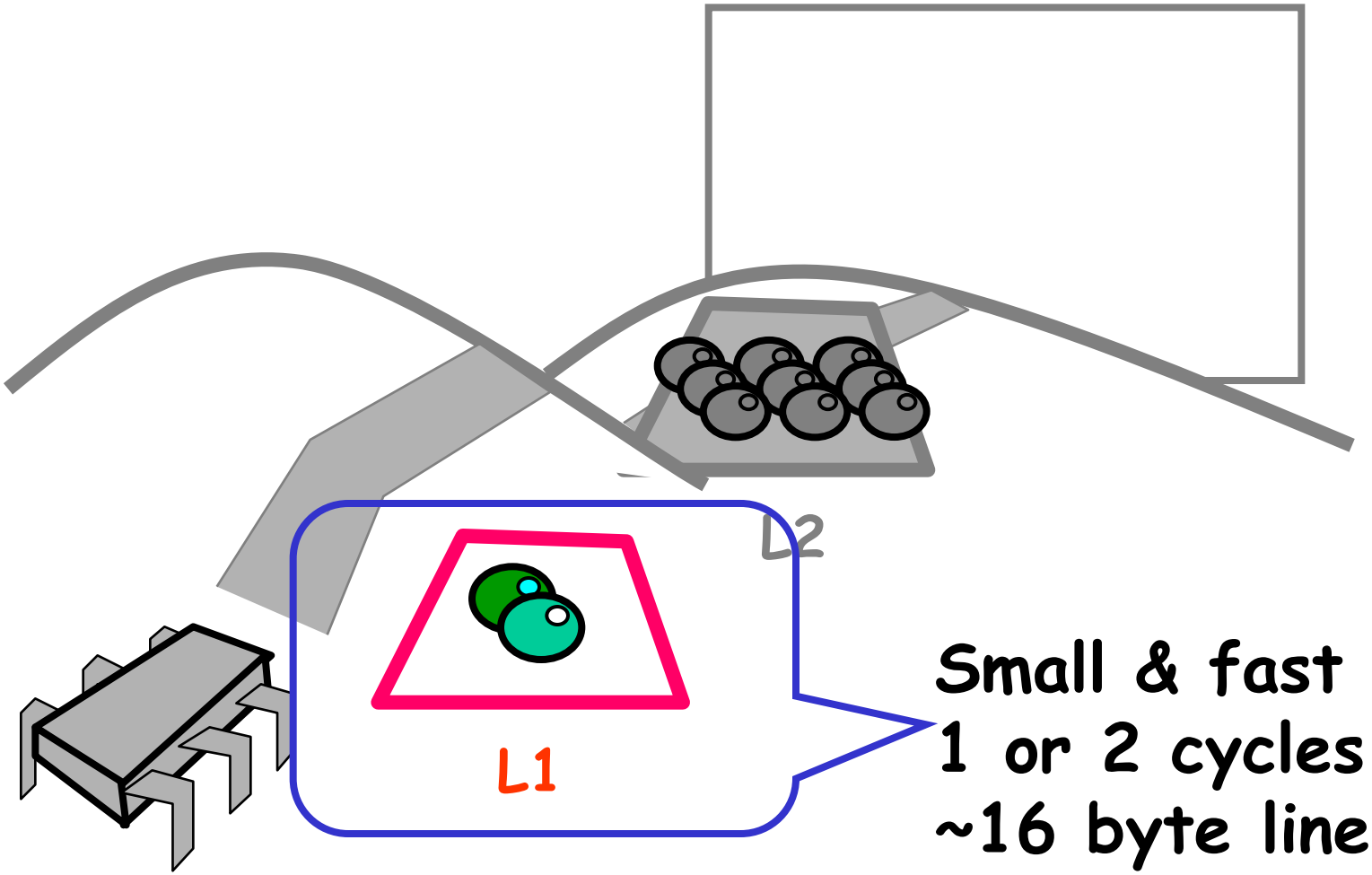
Hit Ratio

- Proportion of requests that hit in the cache
- Measure of effectiveness of caching mechanism
- Depends on locality of application

L1 and L2 Caches

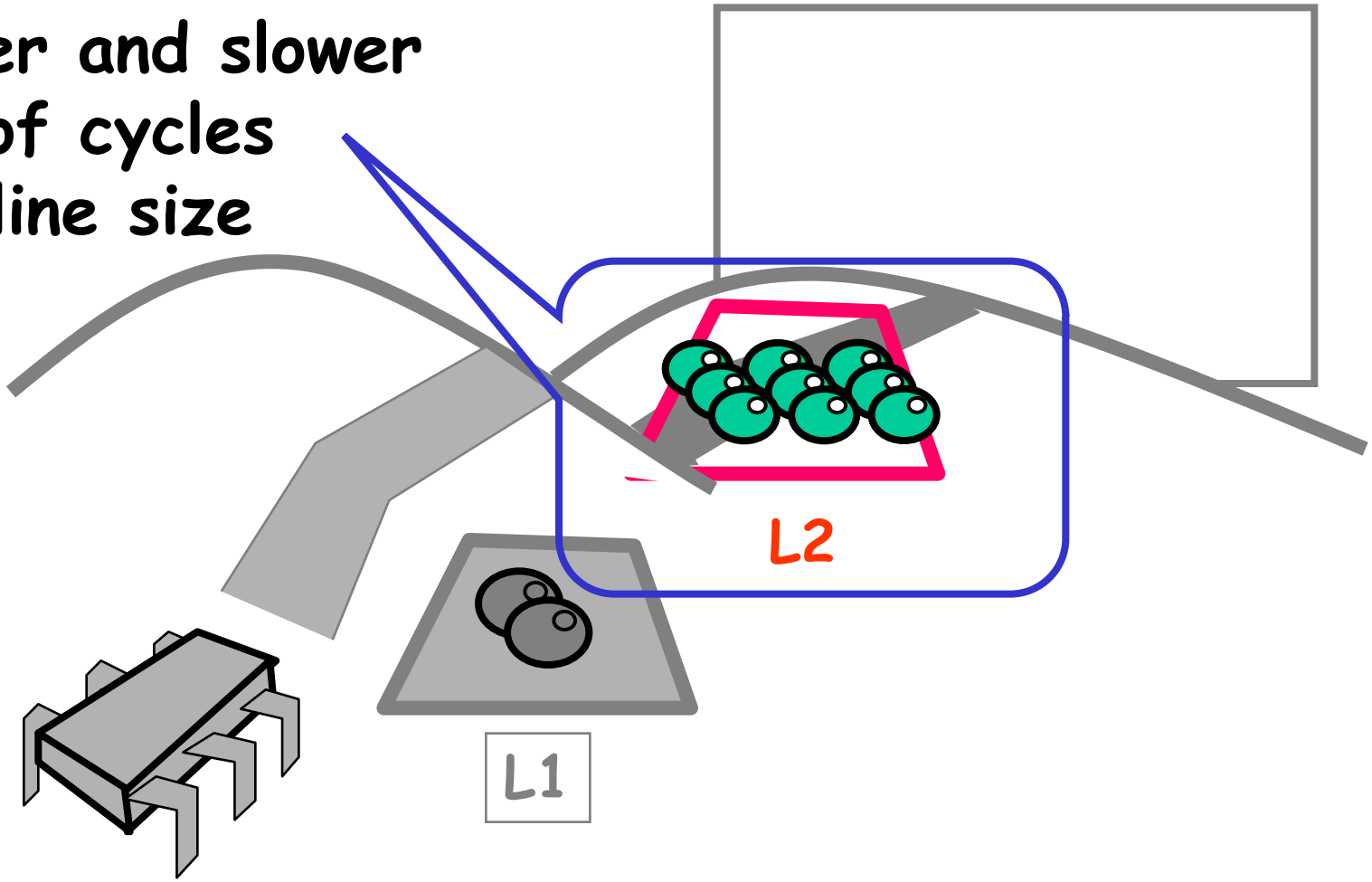


L1 and L2 Caches



L1 and L2 Caches

Larger and slower
10s of cycles
~1K line size

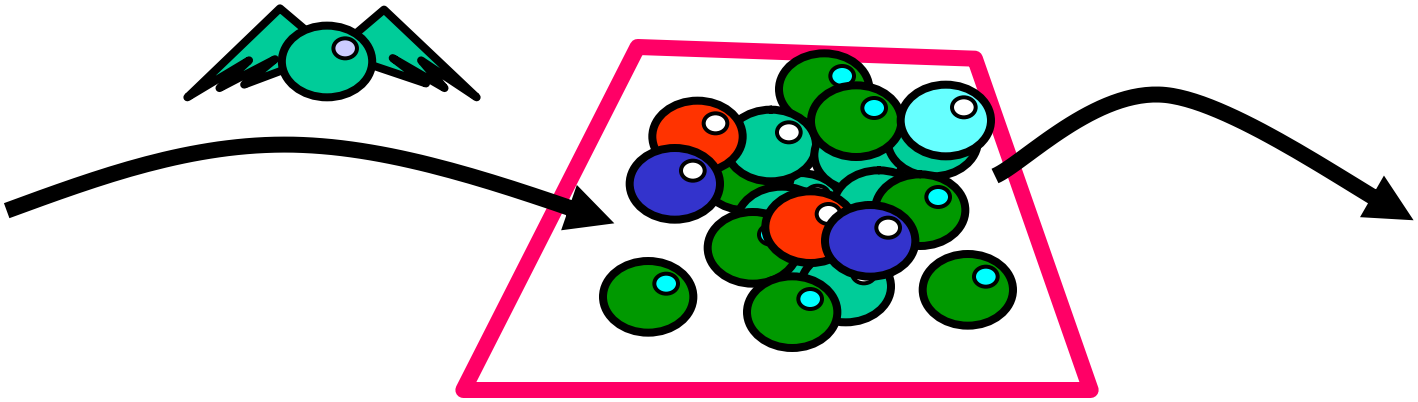


When a Cache Becomes Full…

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

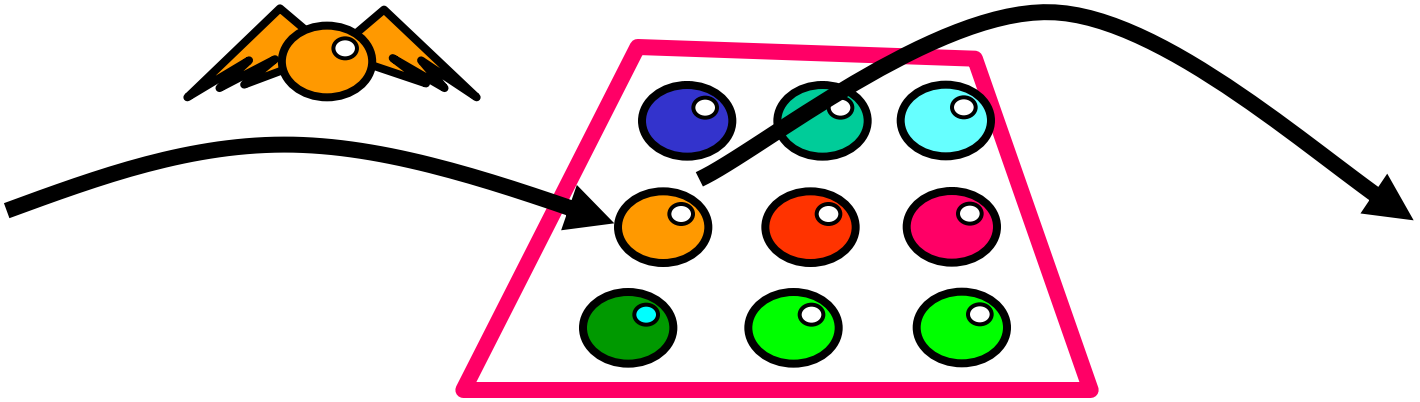
Fully Associative Cache

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



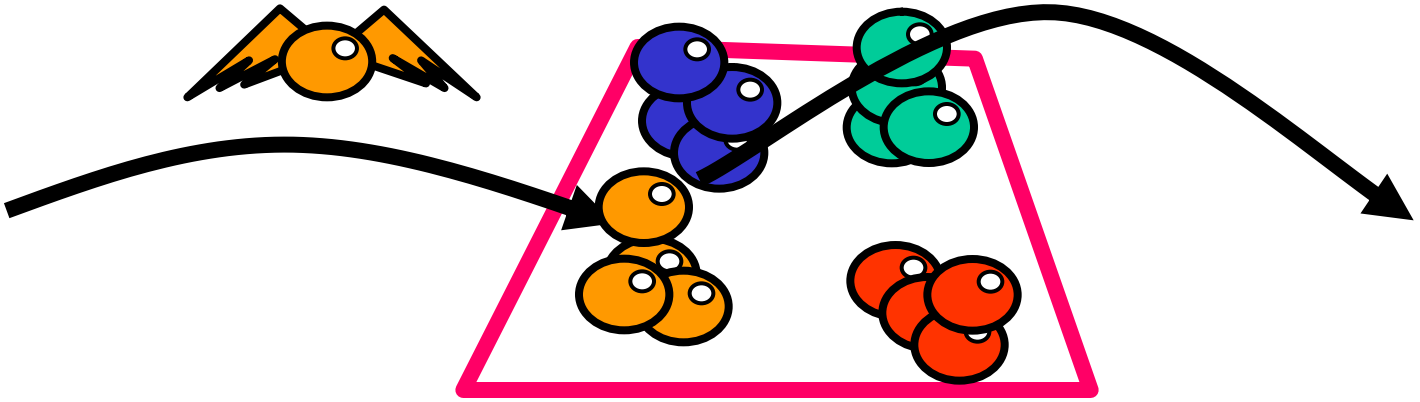
Direct Mapped Cache

- 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



Contention

- Alice and Bob are both writing research papers on aardvarks.
- Alice has encyclopedia vol AA-AC
- Bob asks library for it
  - Library asks Alice to return it
  - Alice returns it & rerequests it
  - Library asks Bob to return it…

Contention

- Good to avoid memory contention.
- Idle processors
- Consumes interconnect bandwidth

Contention

- Alice is still writing a research paper on aardvarks.
- Carol is writing a tourist guide to the German city of Aachen
- No conflict?
  - Library deals with volumes, not articles
  - Both require same encyclopedia volume

False Sharing

- Two processors may conflict over disjoint addresses
- If those addresses lie on the same cache line

False Sharing

- Large cache line size
  - increases locality
  - But also increases likelihood of false sharing
- Sometimes need to "scatter" data to avoid this problem

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

MESI

- Modified
  - Have modified cached data, must write back to memory
- Exclusive
  - Not modified, I have only copy

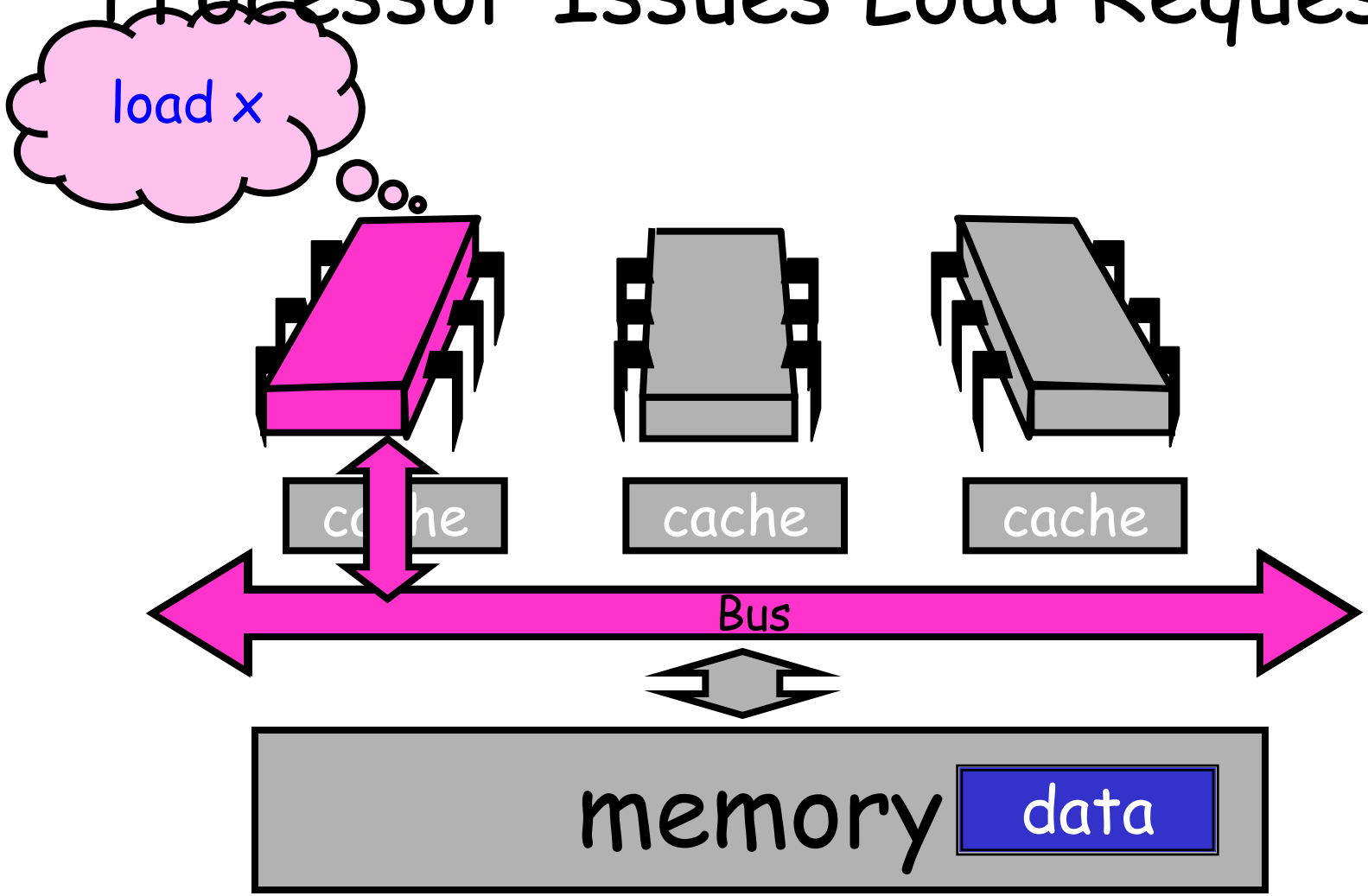
MESI

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

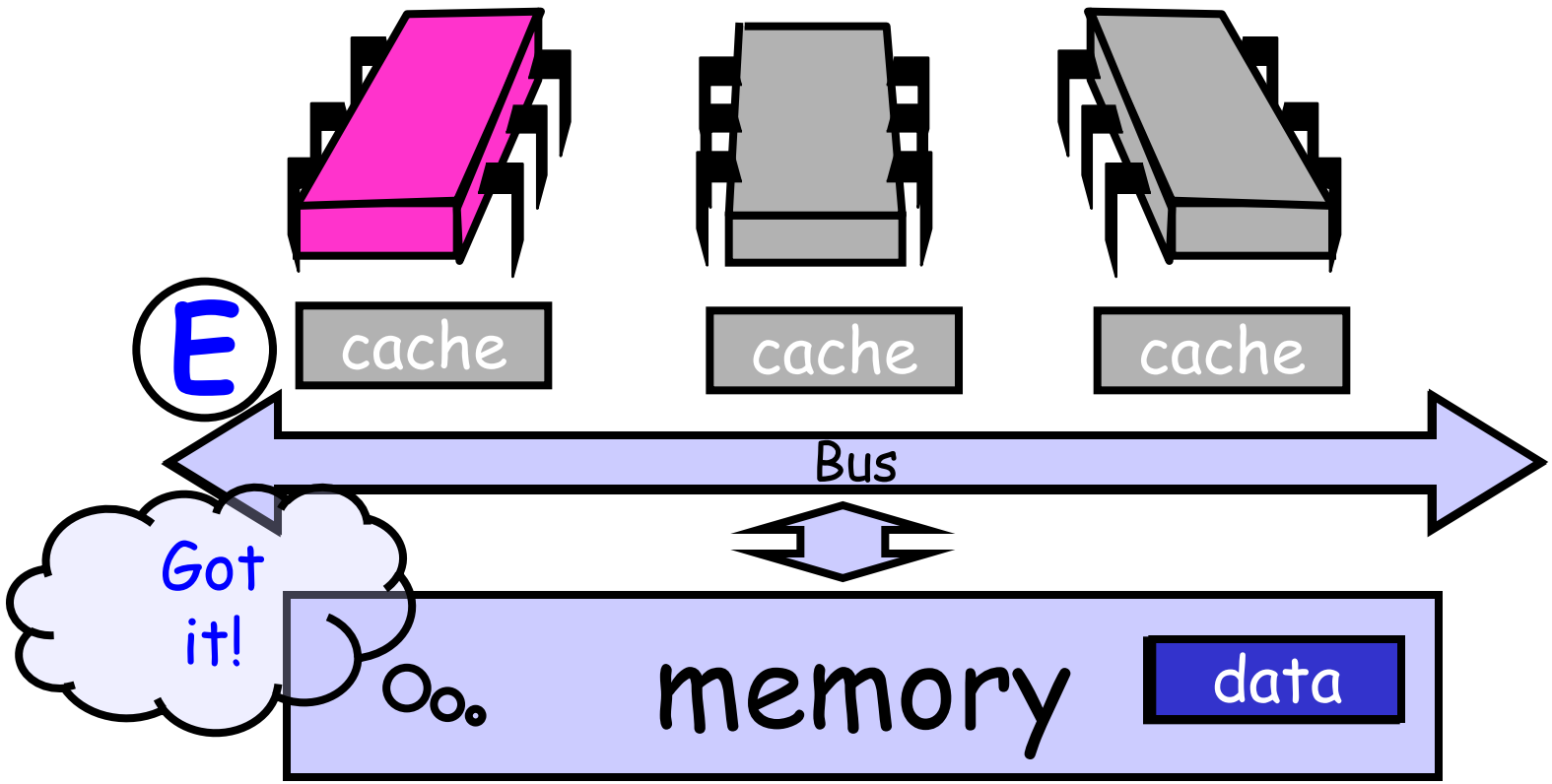
MESI

- 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

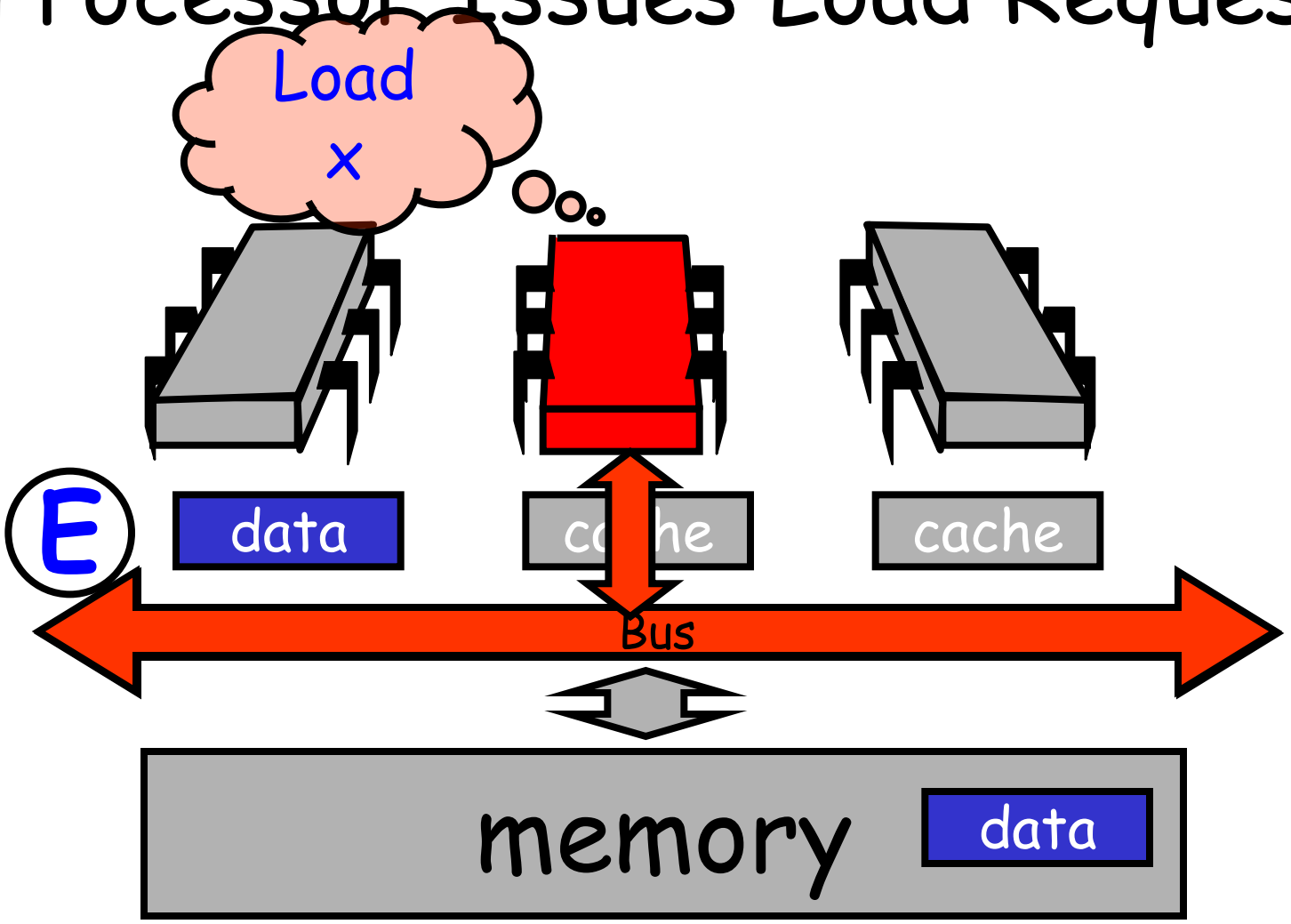
Processor Issues Load Request



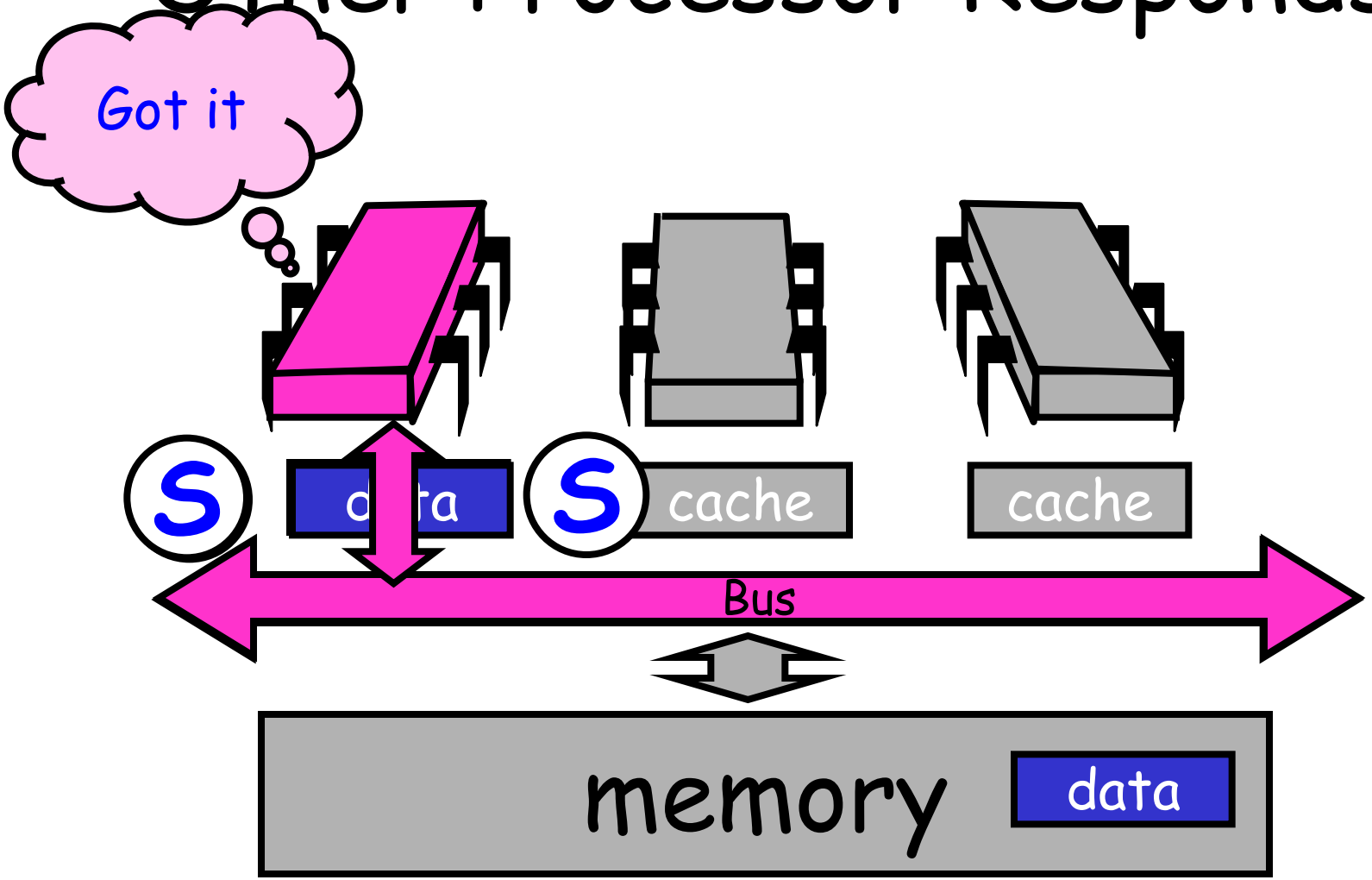
Memory Responds



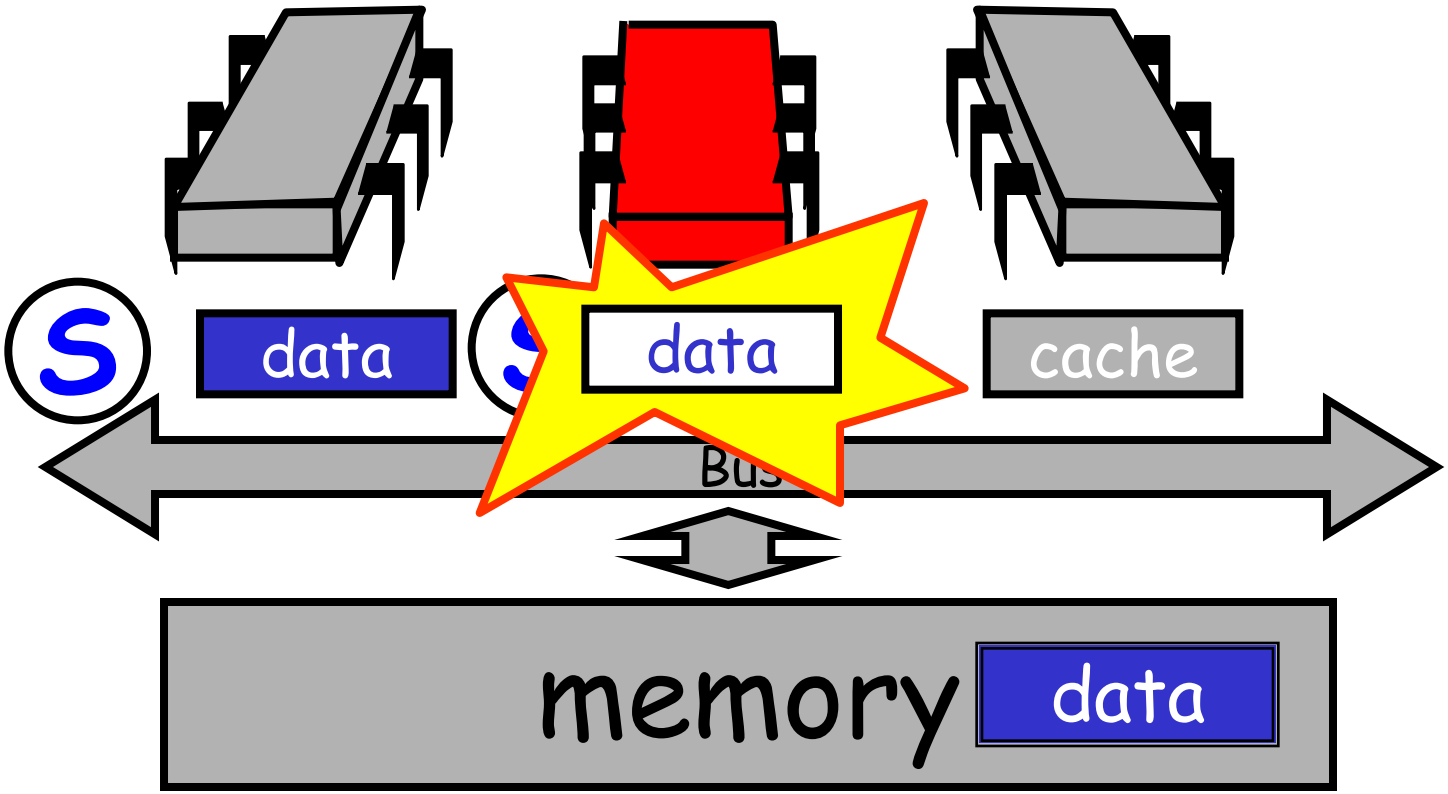
Processor Issues Load Request



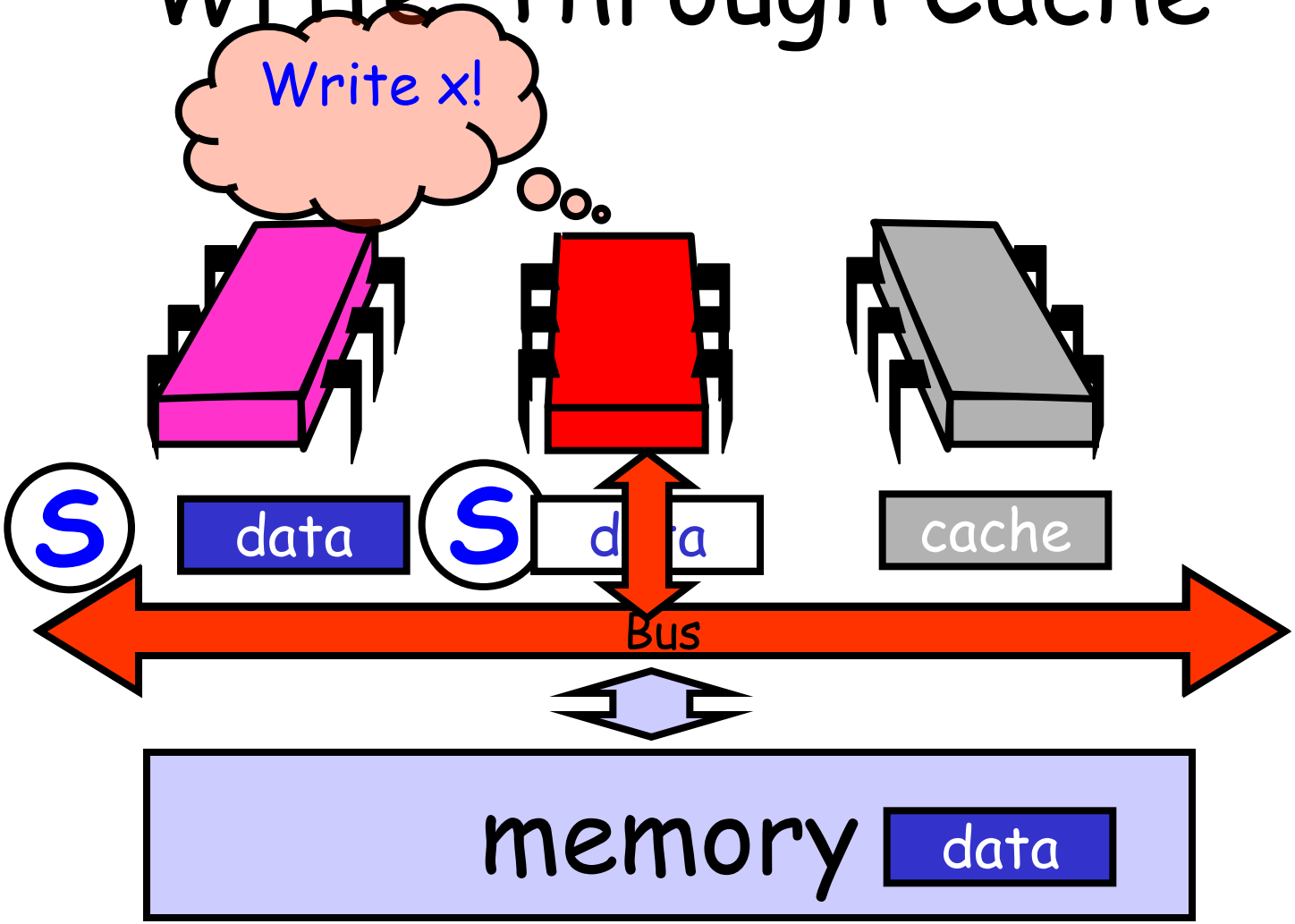
Other Processor Responds



Modify Cached 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 …

Write-Through Caches

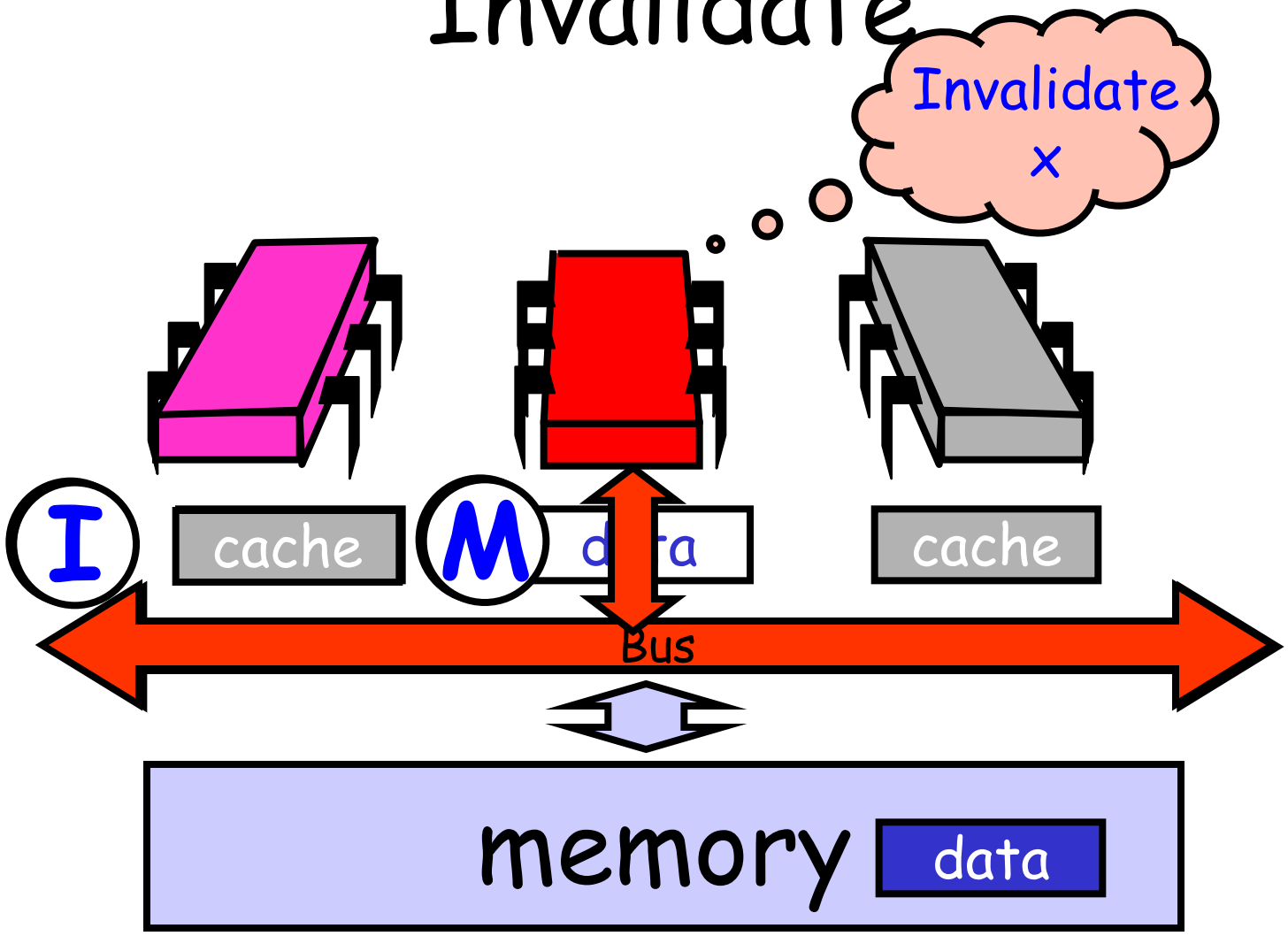
- Immediately broadcast changes
- Good
  - Memory, caches always agree
  - More read hits, maybe
- Bad
  - Bus traffic on all writes
  - Most writes to unshared data
  - For example, loop indexes …

"show stoppers"

Write-Back Caches

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

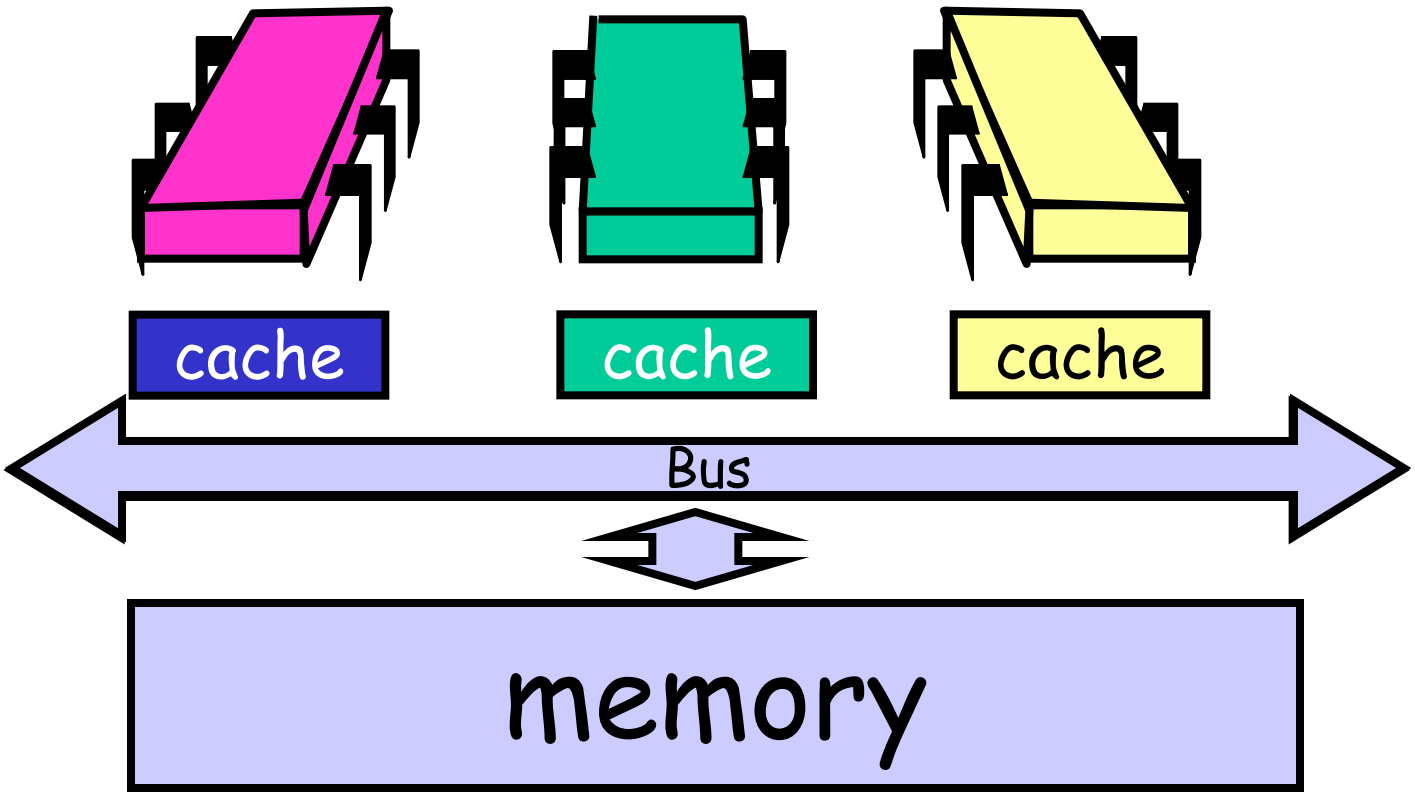
Invalidate



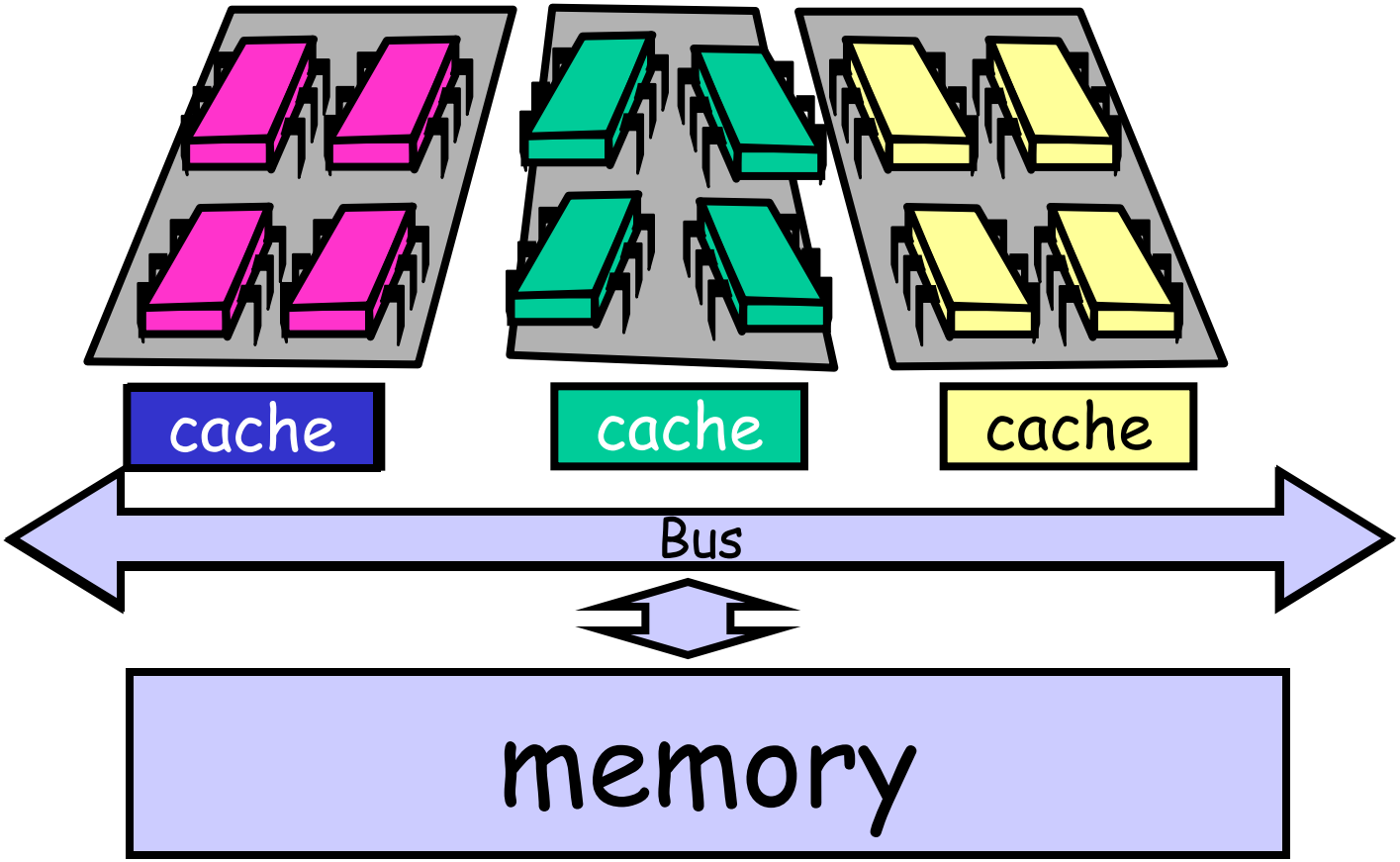
Multicore Architectures

- The university president
  - Alarmed by fall in productivity
- Puts Alice, Bob, and Carol in same corridor
  - Private desks
  - Shared bookcase
- Contention costs go way down

Old-School Multiprocessor



Multicore Architecture



Multicore

- Private L1 caches
- Shared L2 caches
- Communication between same-chip processors now very fast
- Different-chip processors still not so fast

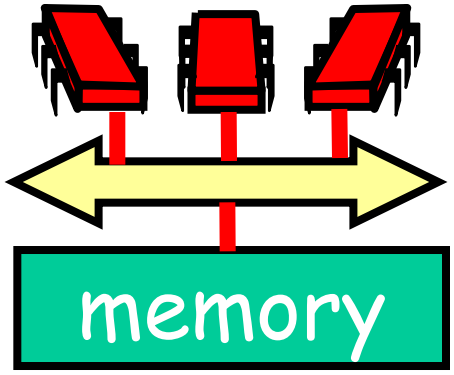
NUMA Architectures

- Alice and Bob transfer to NUMA State University
- No centralized library
- Each office basement holds part of the library

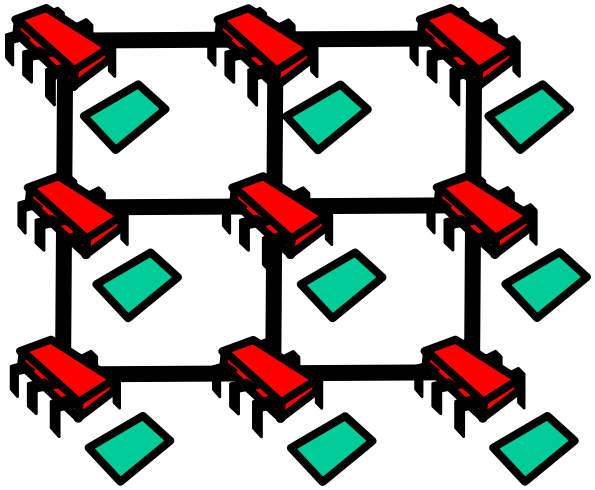
Distributed Shared-Memory Architectures

- Alice's has volumes that start with A
  - Aardvark papers are convenient: run downstairs
  - Zebra papers are inconvenient: run across campus

SMP vs NUMA



SMP



NUMA

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

Spinning Again

- NUMA without cache
  - OK if local variable
  - Bad if remote
- Cc-NUMA
  - Like SMP

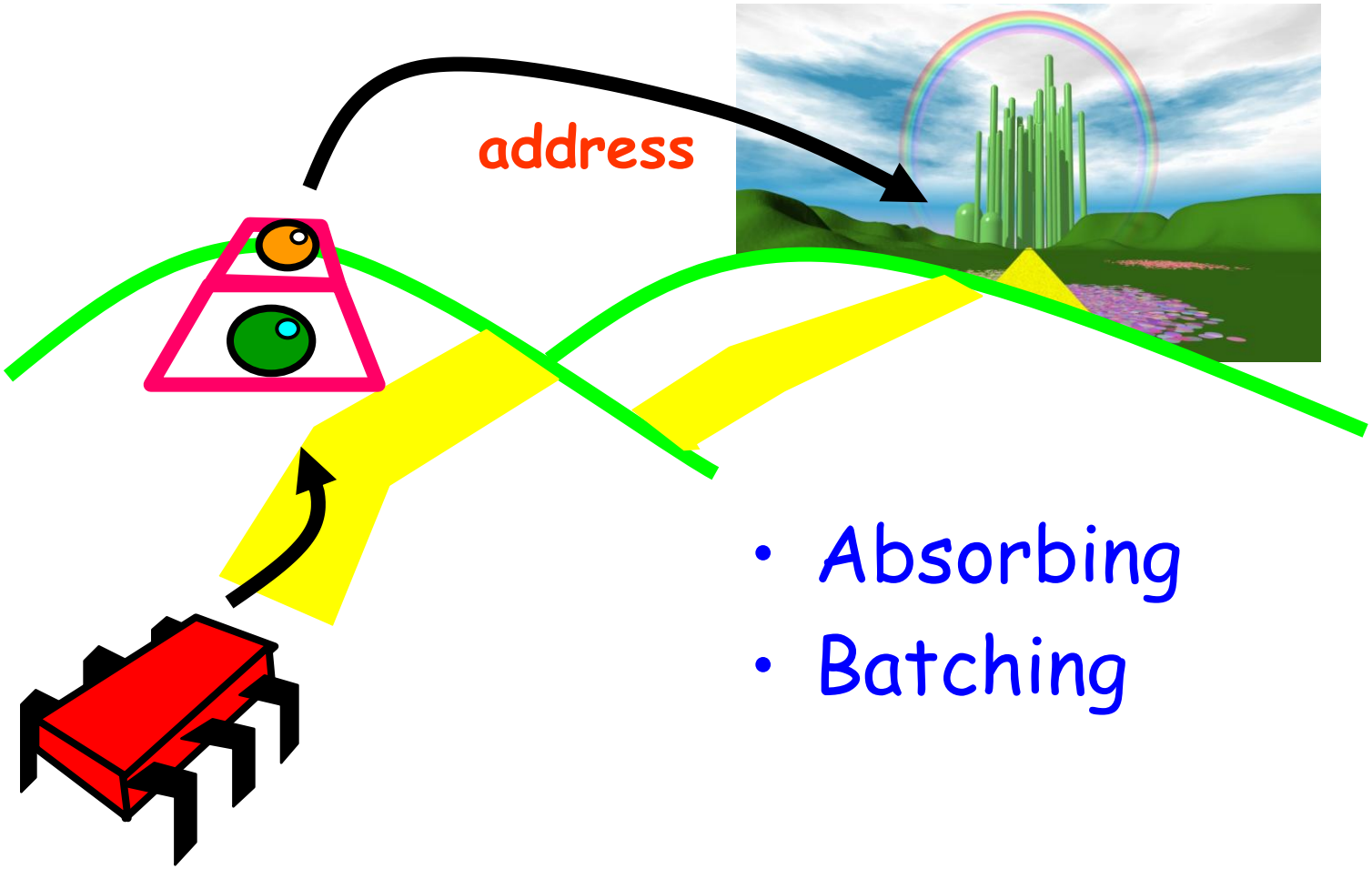
Relaxed Memory

- Remember the flag principle?
  - Alice and Bob's flag variables false
- Alice writes true to her flag and reads Bob's
- Bob writes true to his flag and reads Alice's
- One must see the other's flag true

Not Necessarily So

- Sometimes the compiler reorders memory operations
- Can improve
  - cache performance
  - interconnect use
- But unexpected concurrent interactions

Write Buffers



- Absorbing
- Batching

Volatile

- In Java, if a variable is declared volatile, operations won't be reordered
- Expensive, so use it only when needed