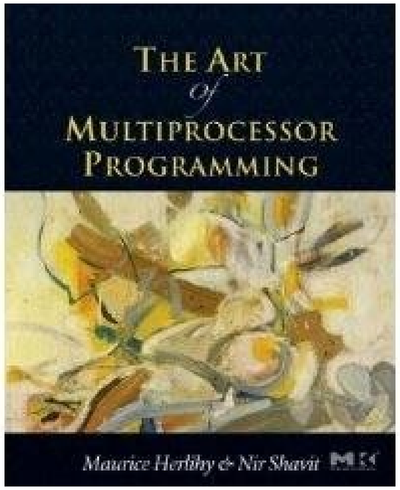
Multiprocessor Architecture Basics

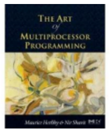

The Art of Multiprocessor Programming
Maurice Herlihy & Nir Shavit

Companion slides for
The Art of Multiprocessor Programming
by Maurice Herlihy & Nir Shavit

With some very minor changes by APS

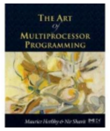
Multiprocessor Architecture

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

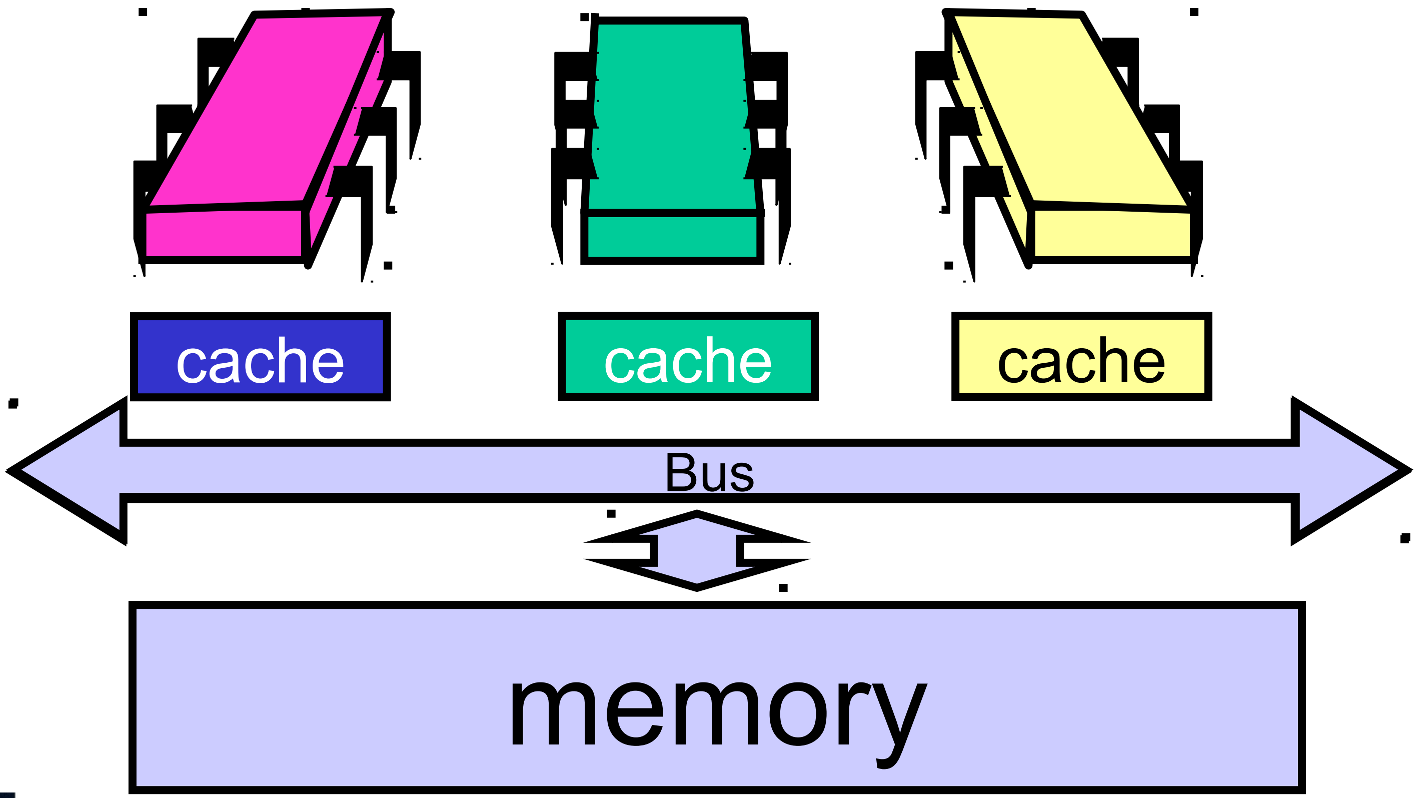


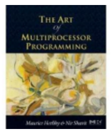
Pieces

- Processors
- Threads
- Interconnect
- Memory
- Caches



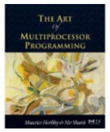
Old-School Multiprocessor



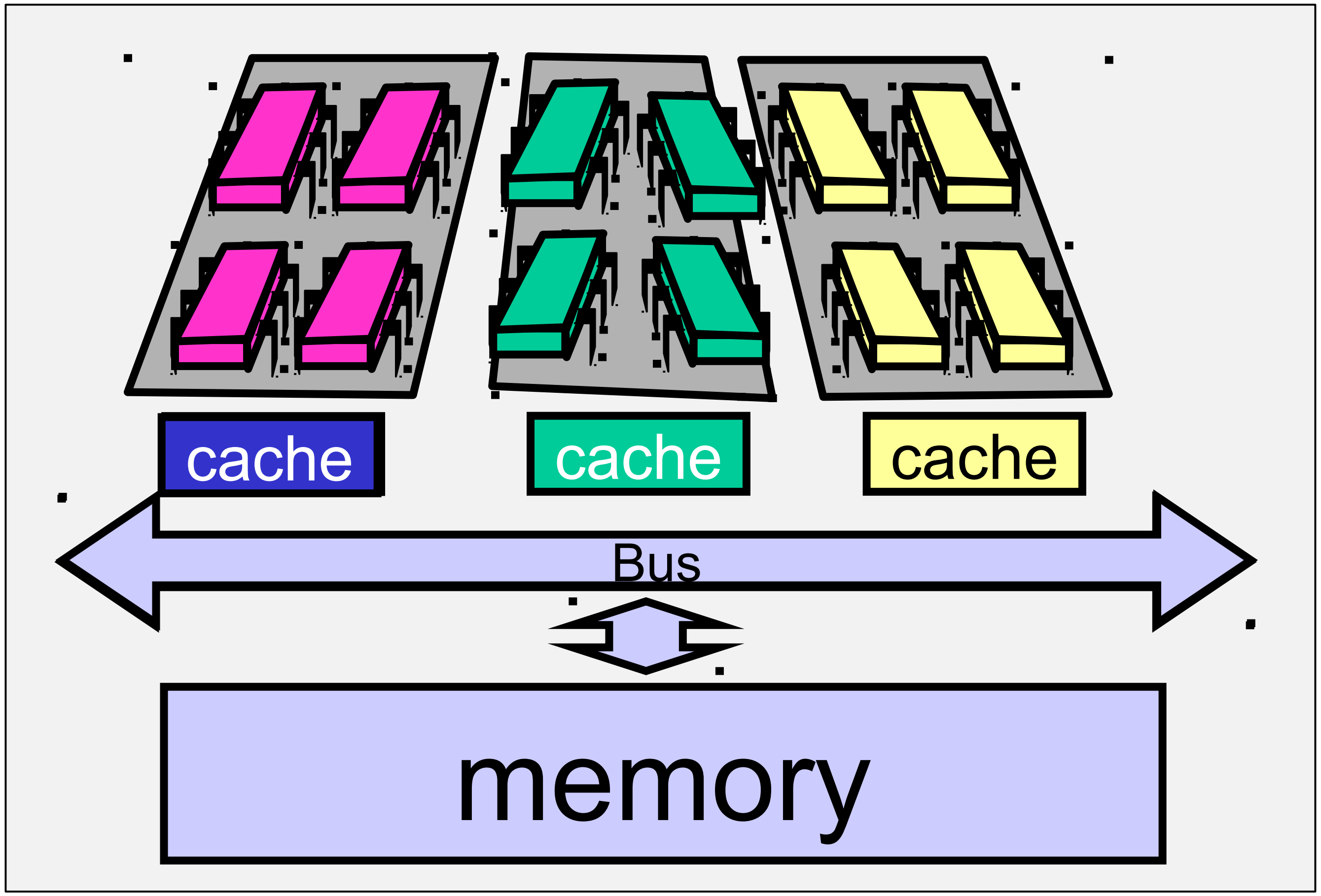


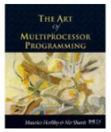
Old School

- Processors on different chips
- Processors share off chip memory resources
- Communication between processors typically slow



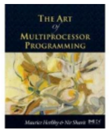
Multicore Architecture



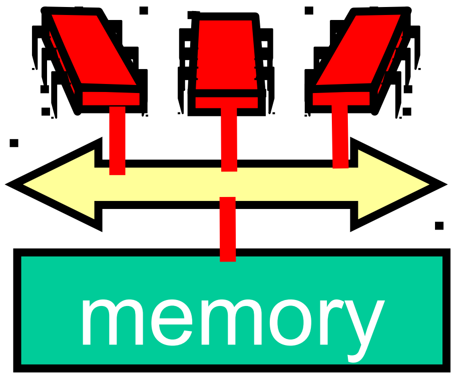


Multicore

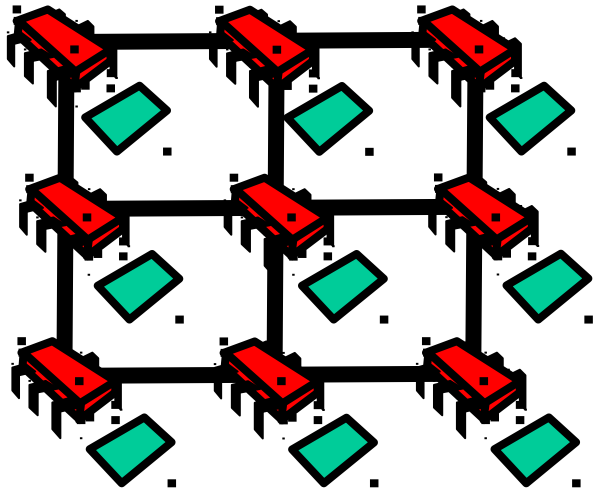
- All Processors on same chip
- Processors share on chip memory resources
- Communication between processors now very fast



SMP vs NUMA

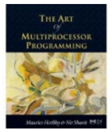


SMP



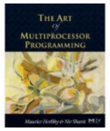
NUMA

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



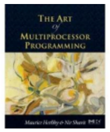
Future Multicores

- Short term: SMP
- Long Term: most likely a combination of SMP and NUMA properties



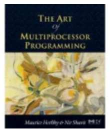
Understanding the Pieces

- Lets try to understand what the pieces that make the multiprocessor machine are
- And how they fit together



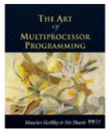
Processors

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



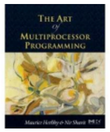
Computer Architecture

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



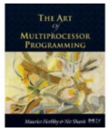
Threads

- 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



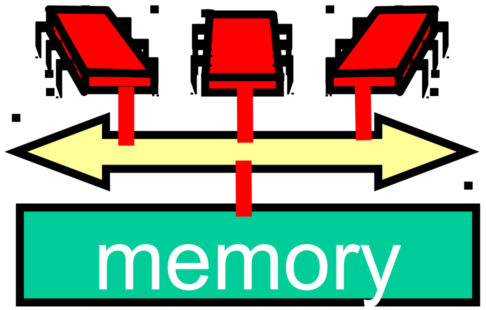
Analogy

- 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

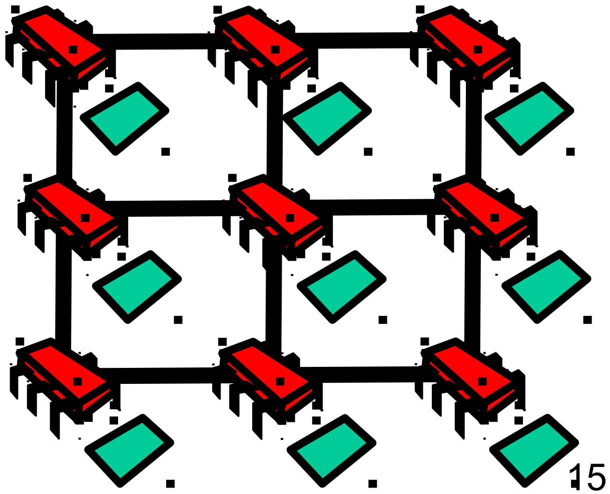


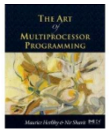
Interconnect

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



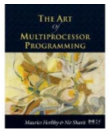
SMP



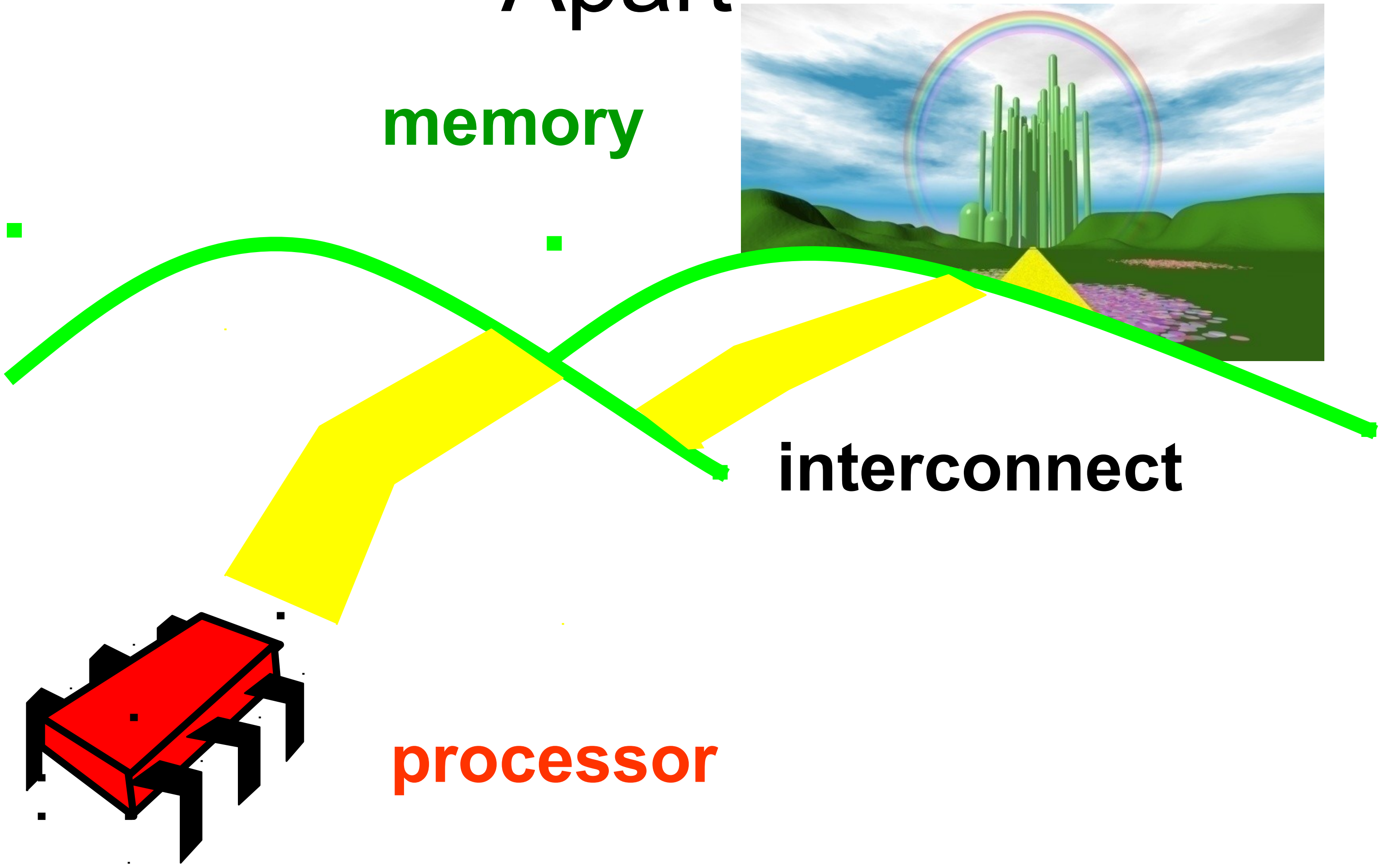


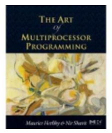
Interconnect

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

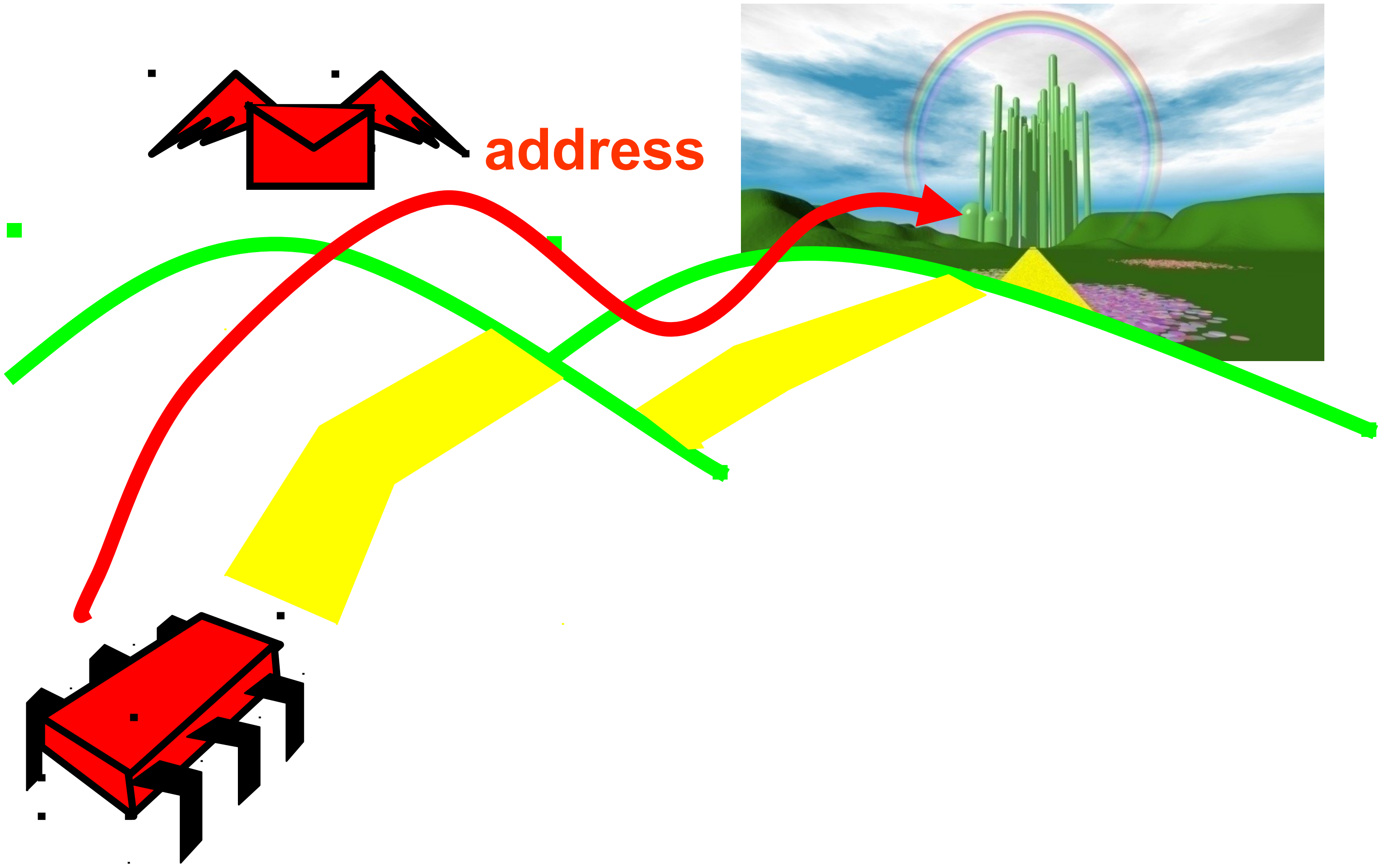


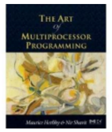
Processor and Memory are Far Apart



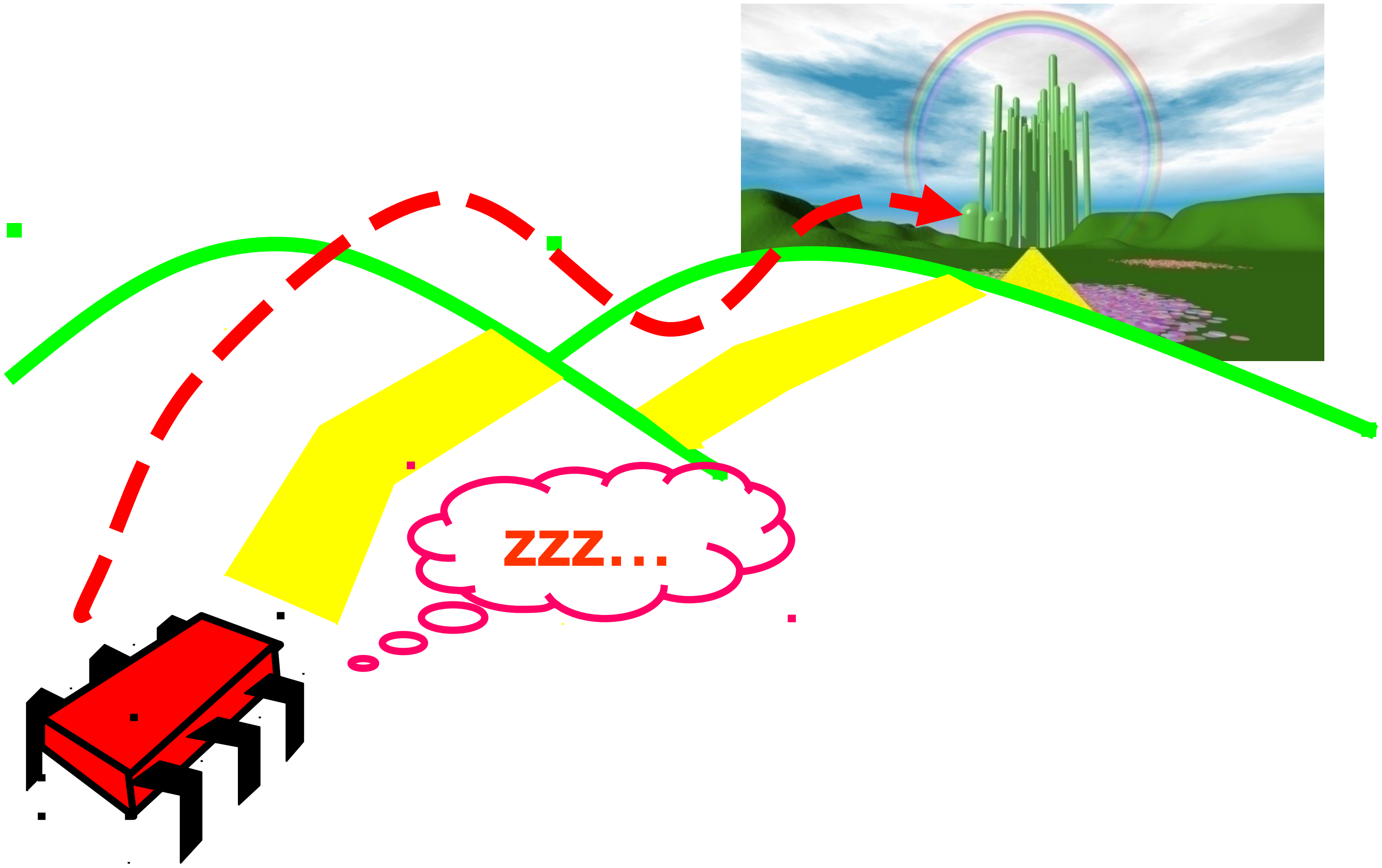


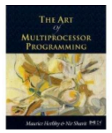
Reading from Memory



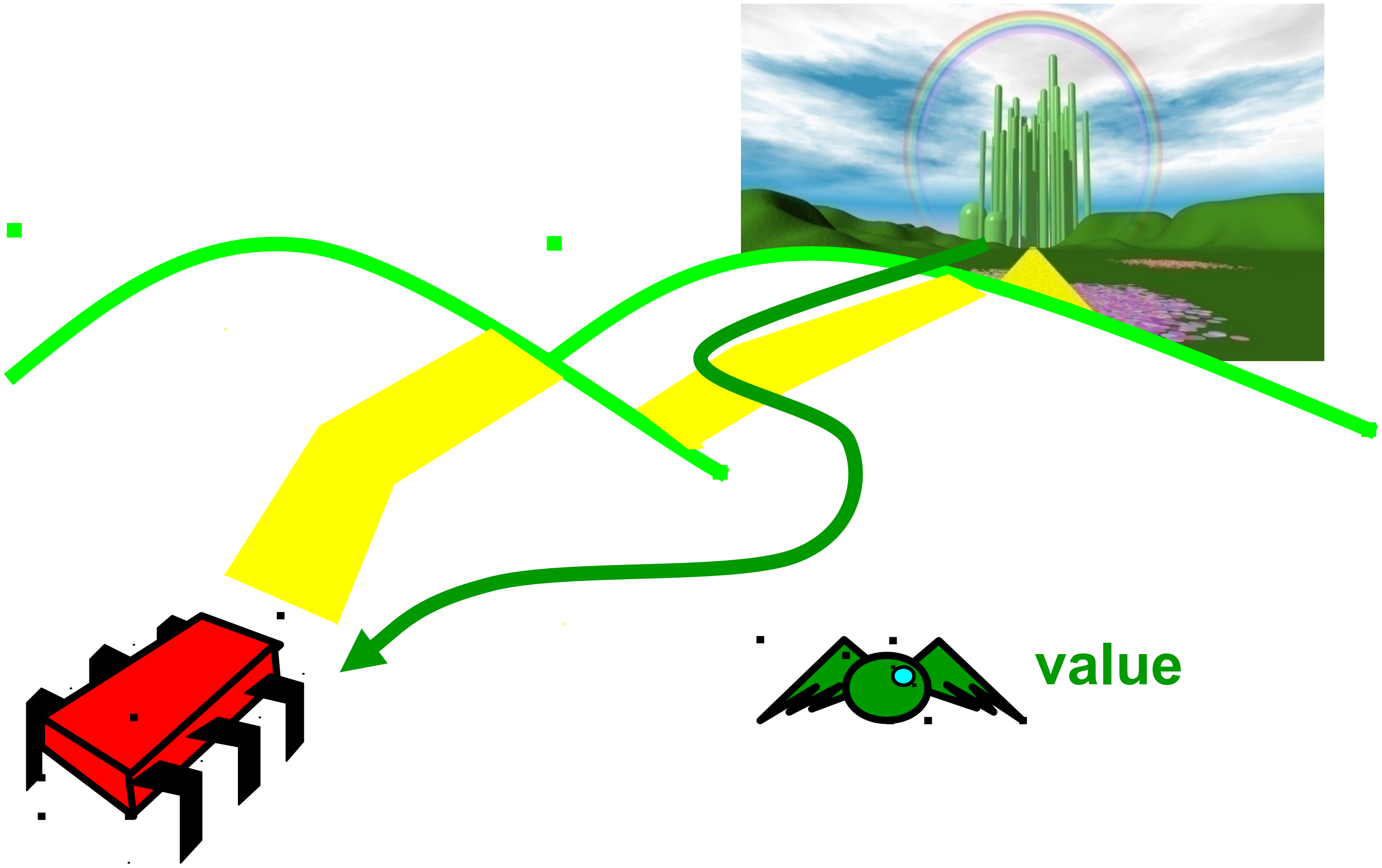


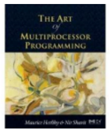
Reading from Memory



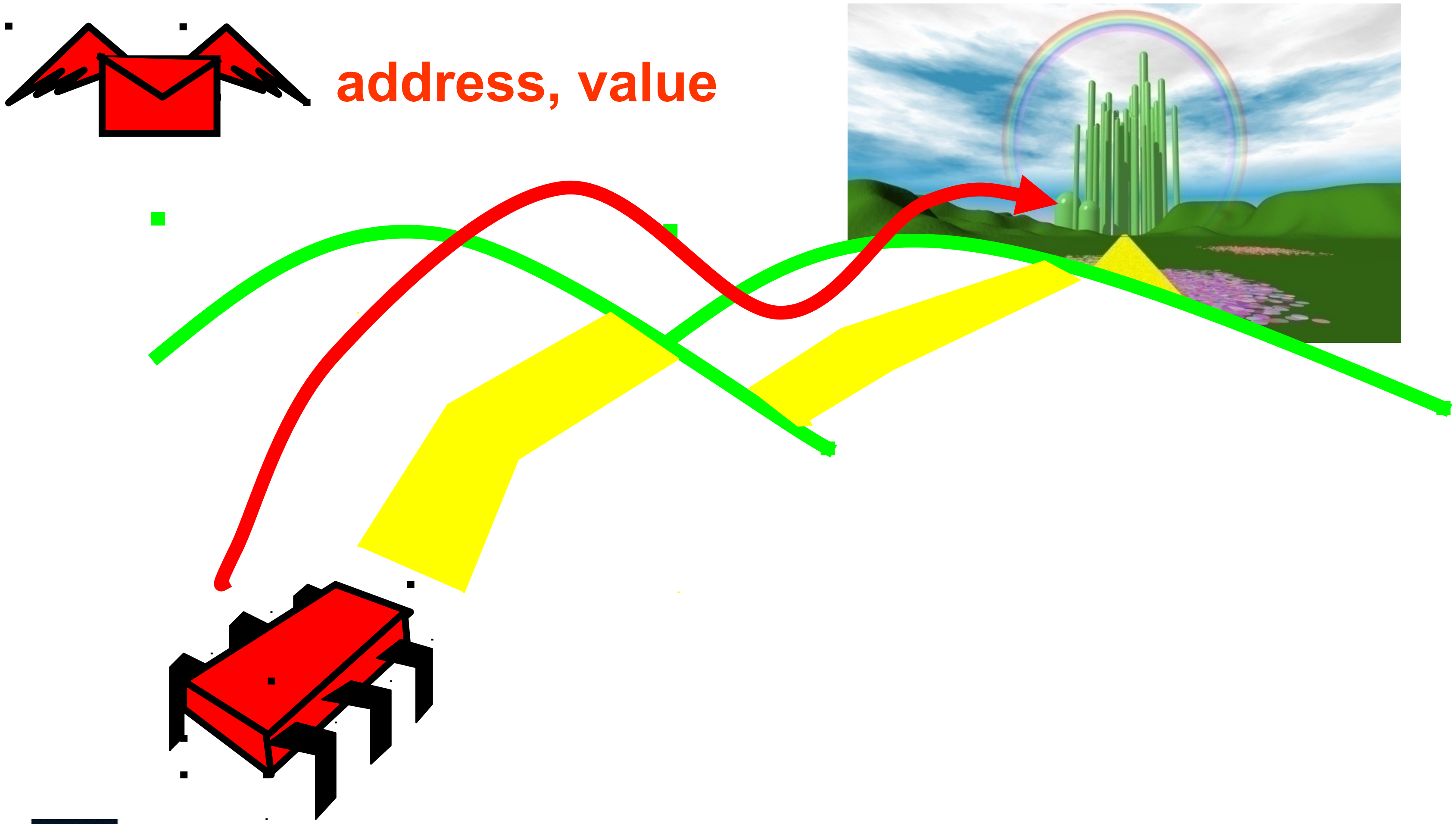


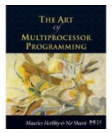
Reading from Memory



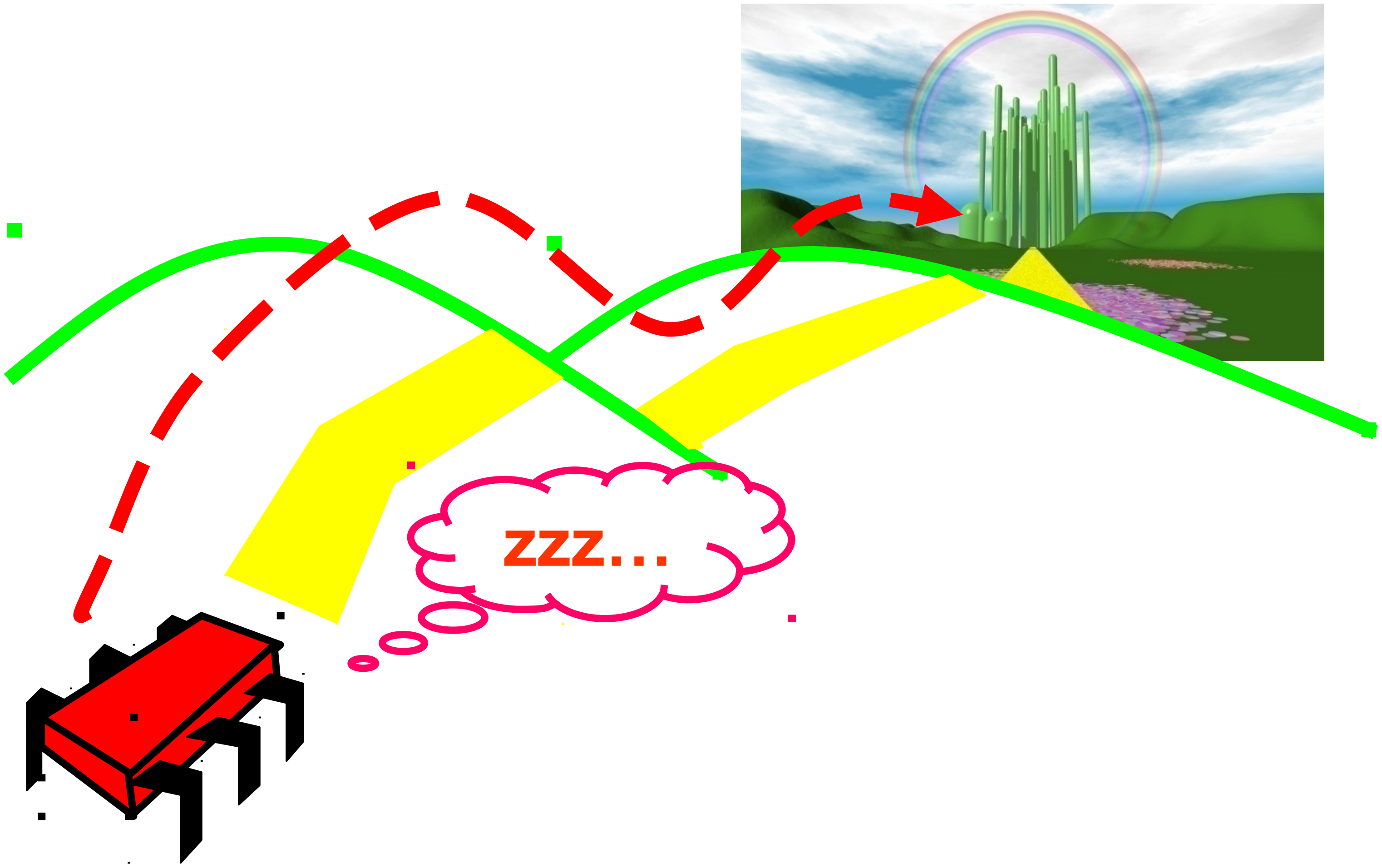


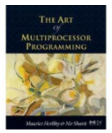
Writing to Memory



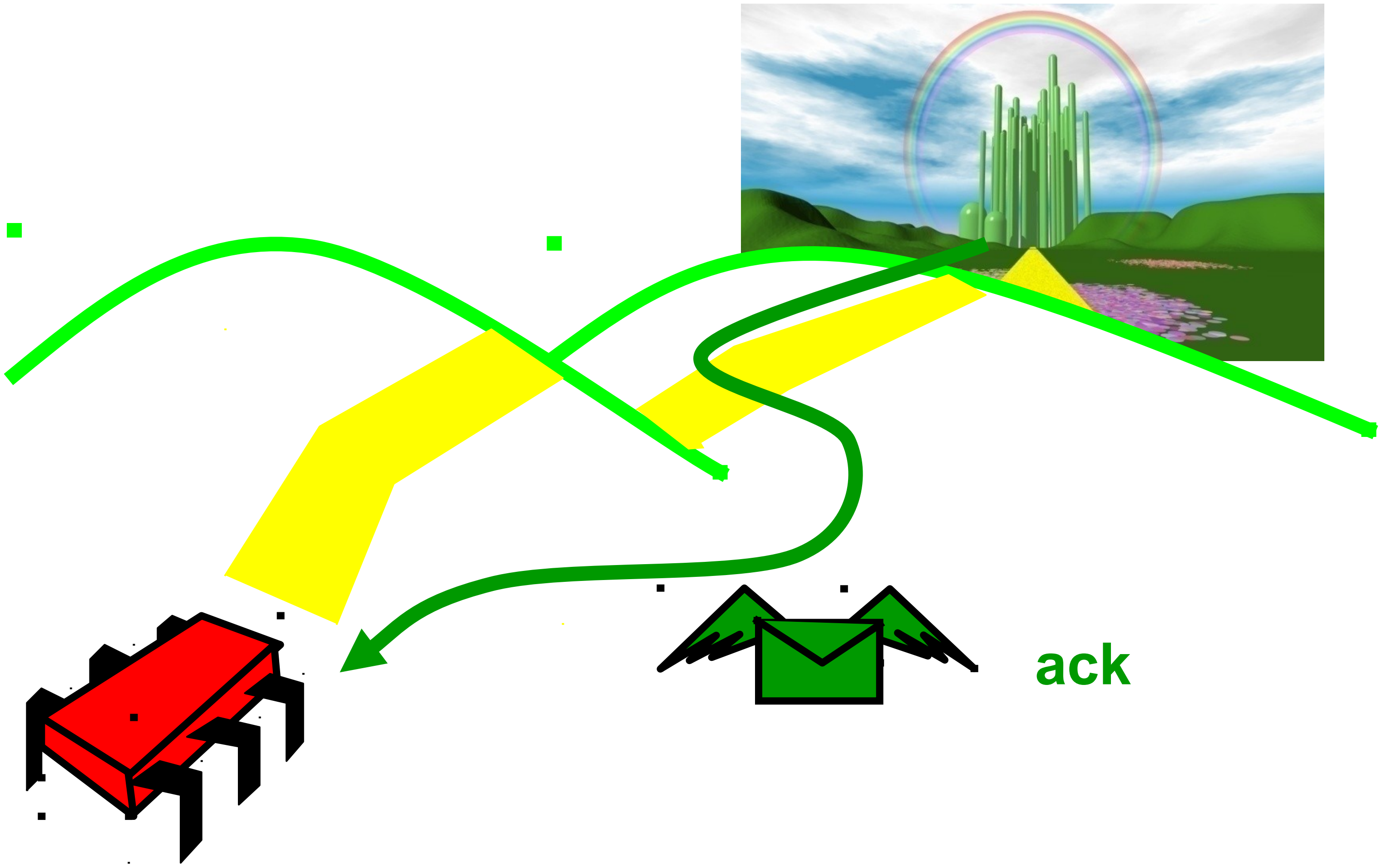


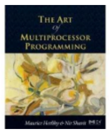
Writing to Memory



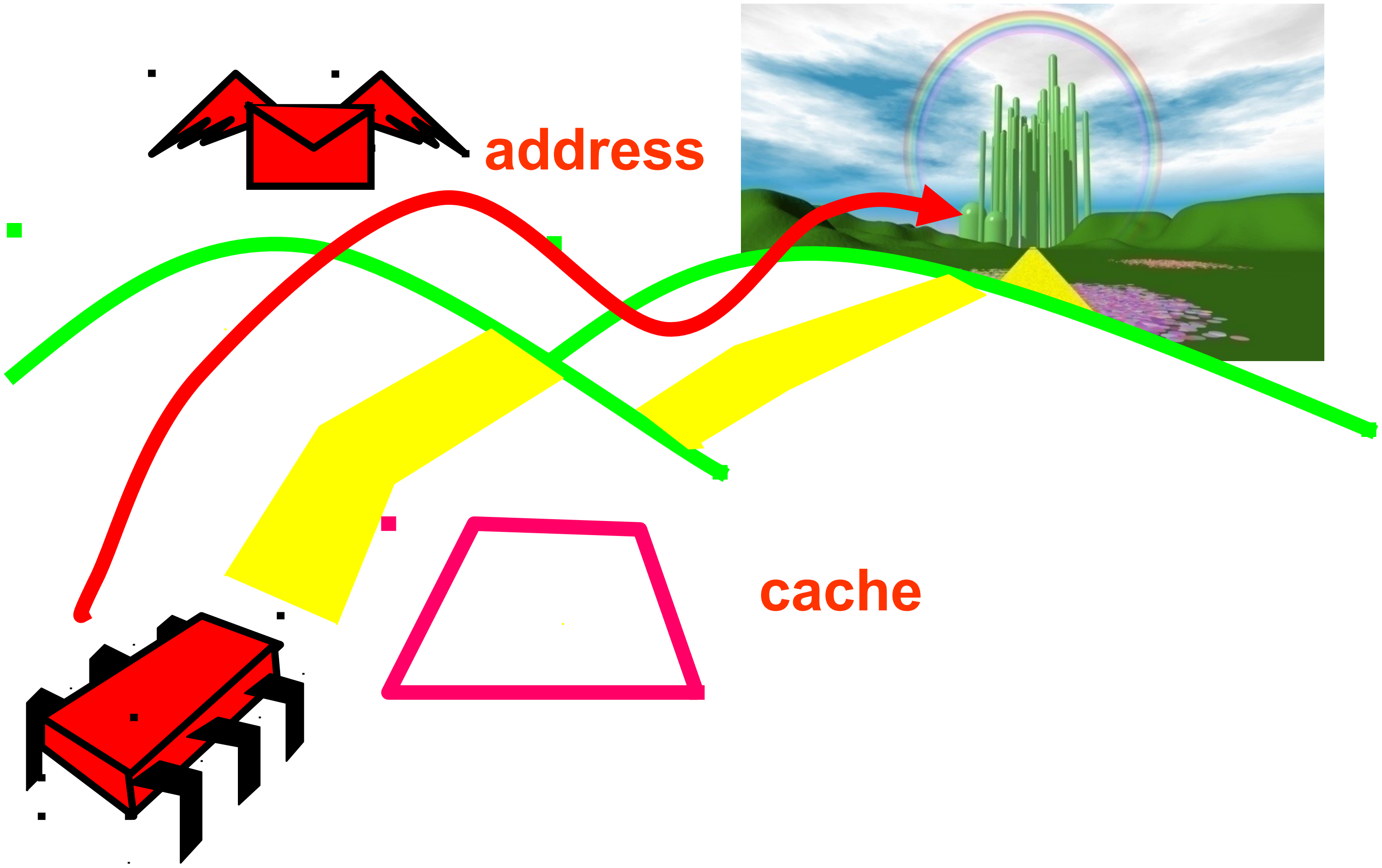


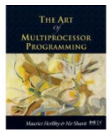
Writing to Memory



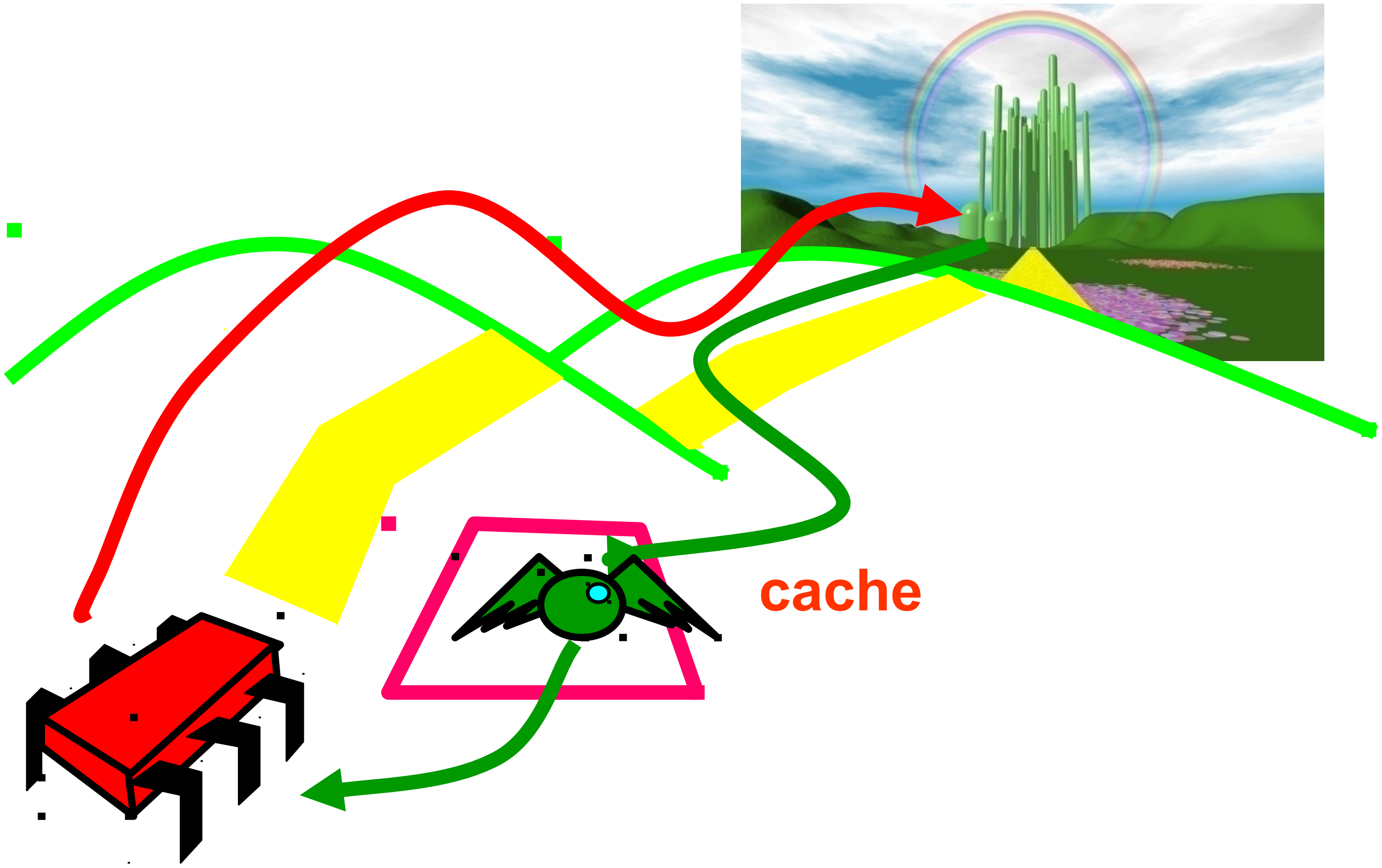


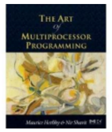
Cache: Reading from Memory



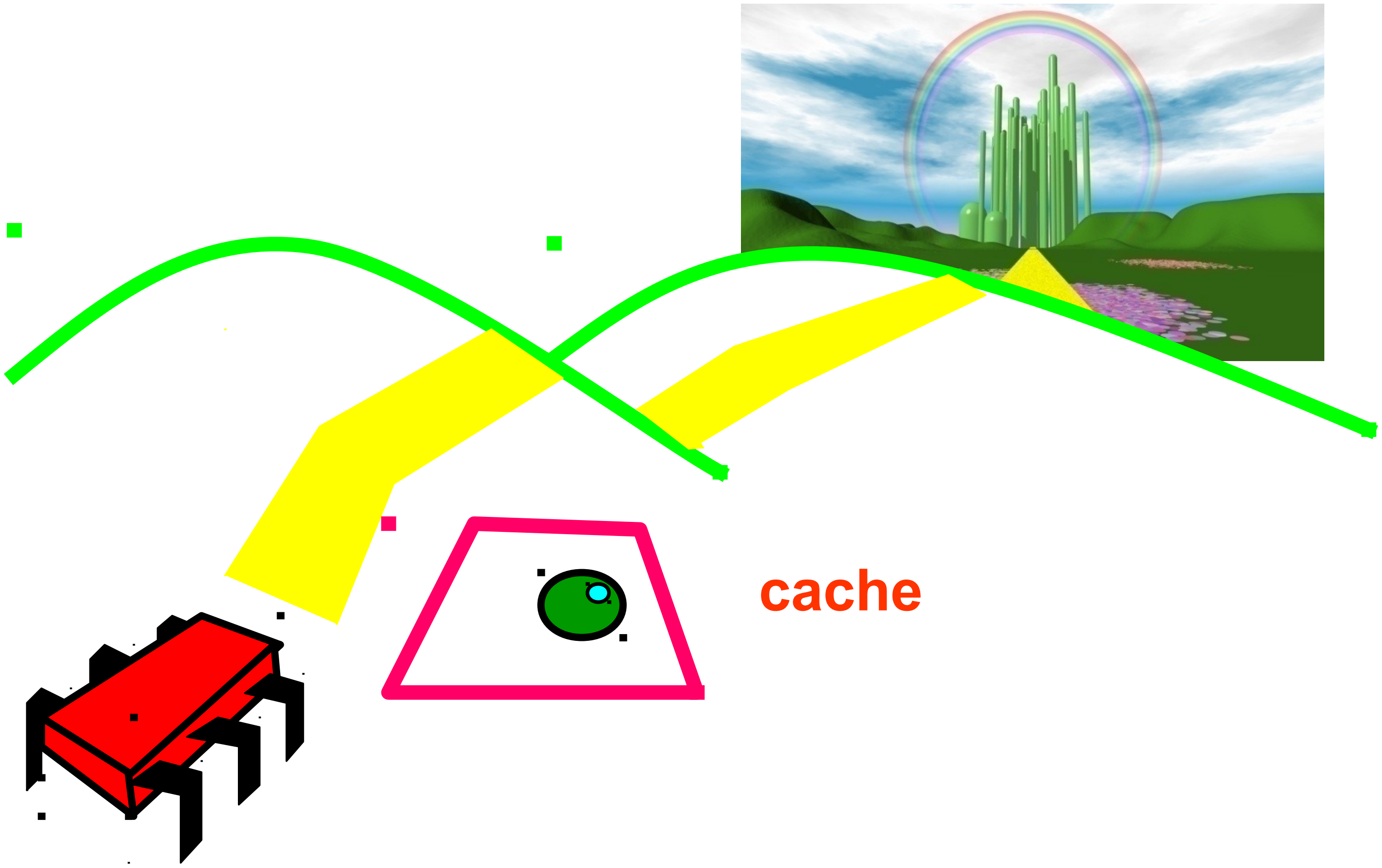


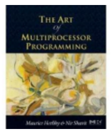
Cache: Reading from Memory



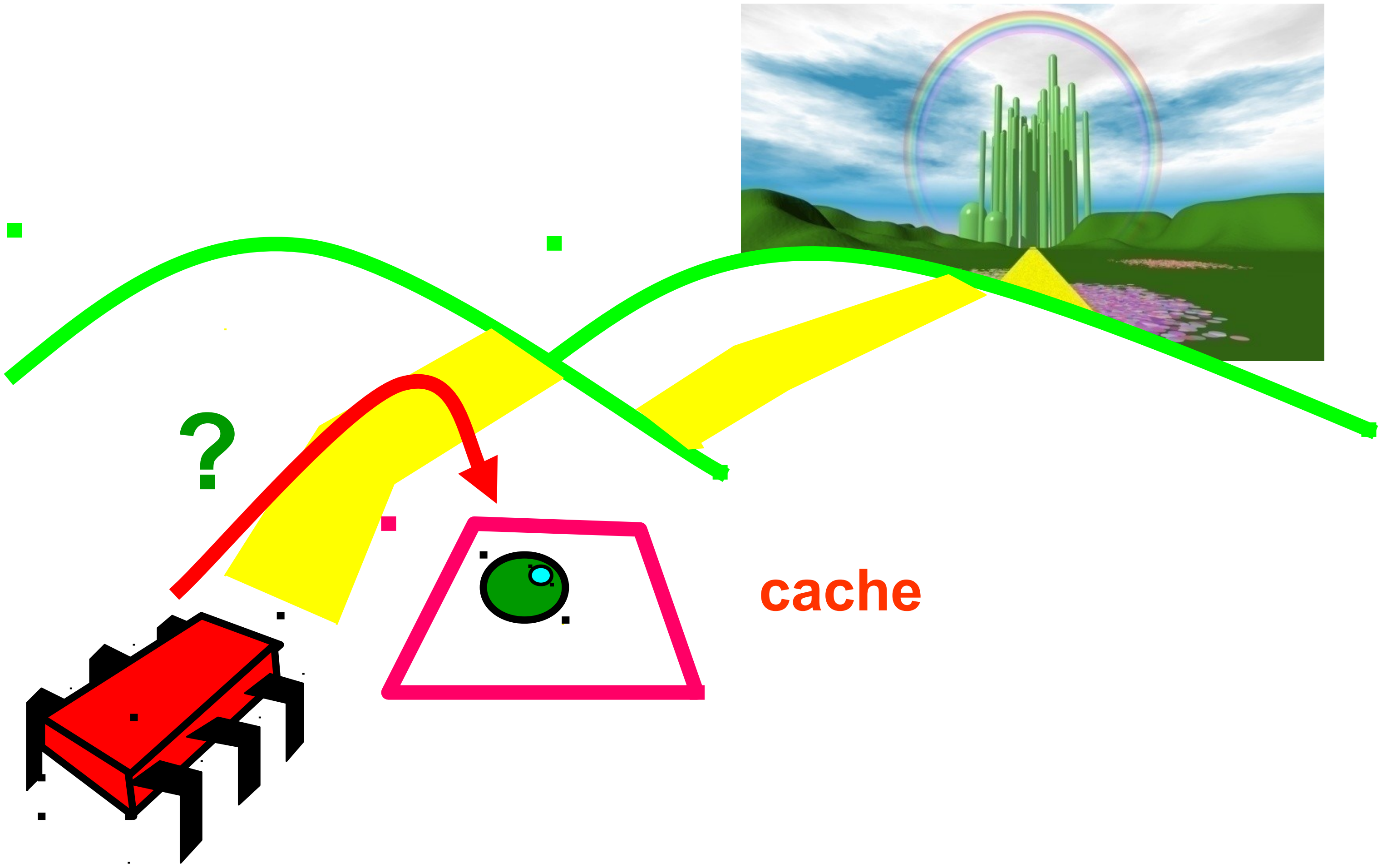


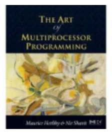
Cache: Reading from Memory



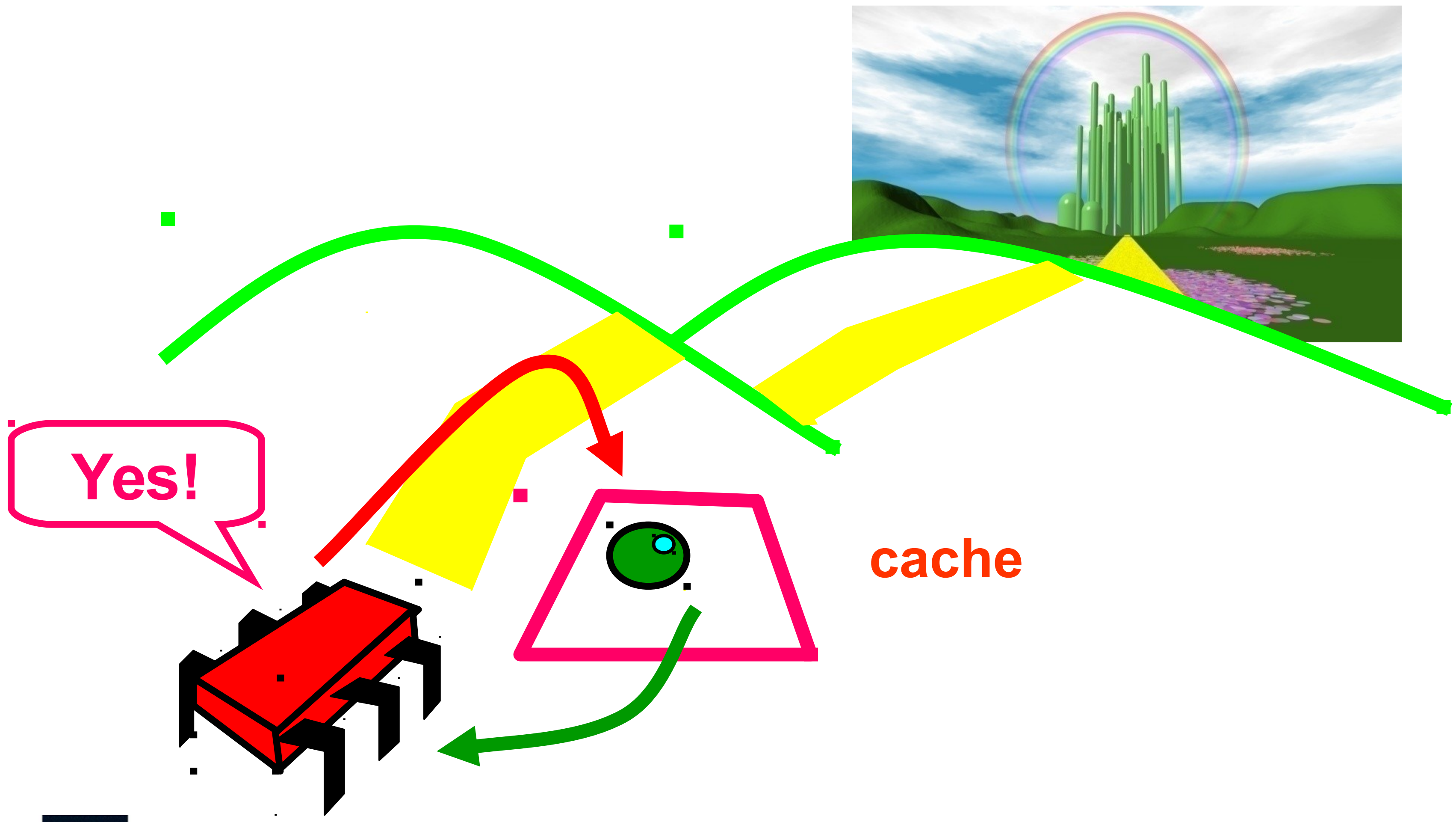


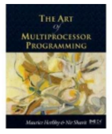
Cache Hit



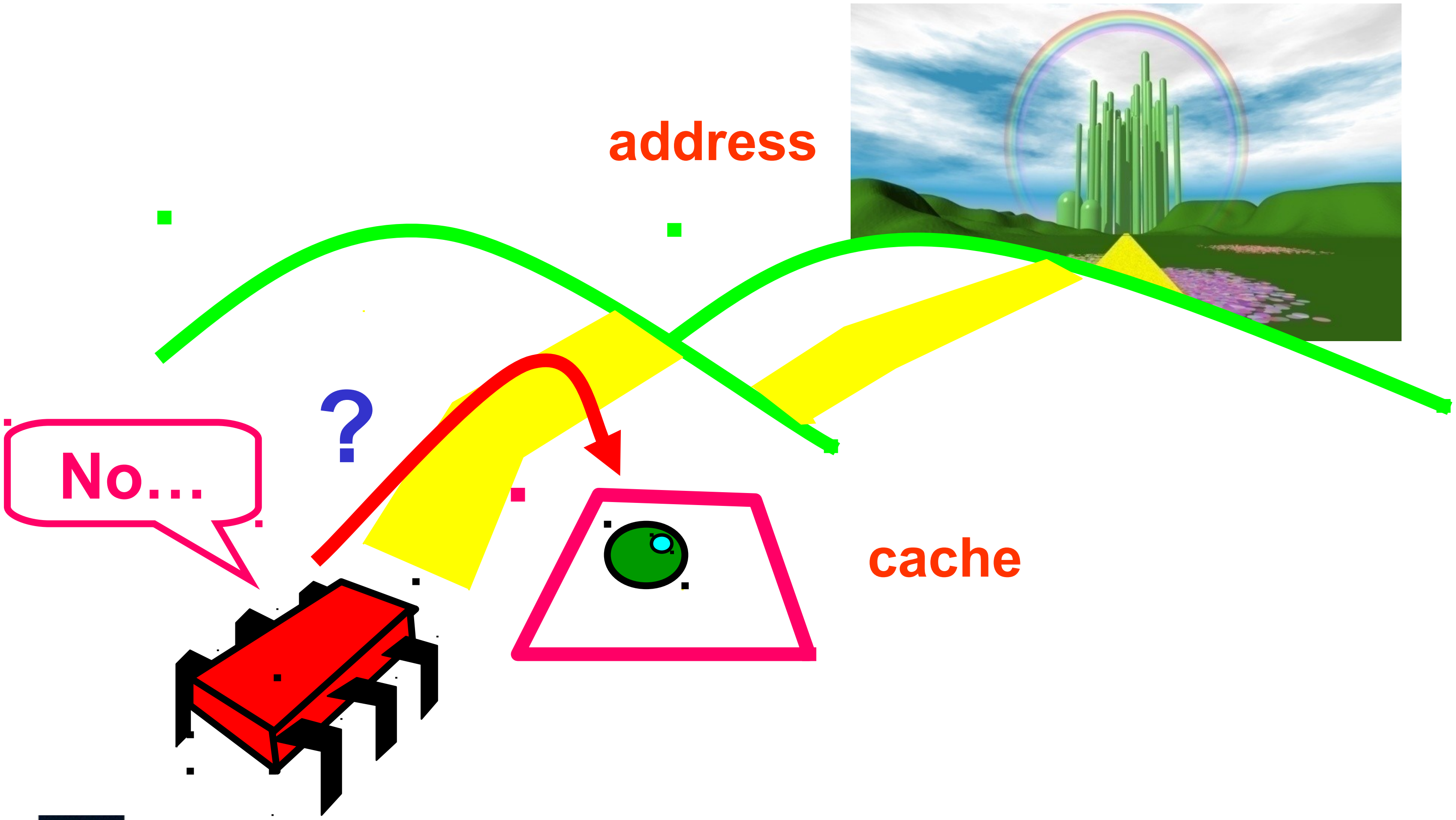


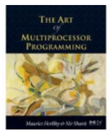
Cache Hit



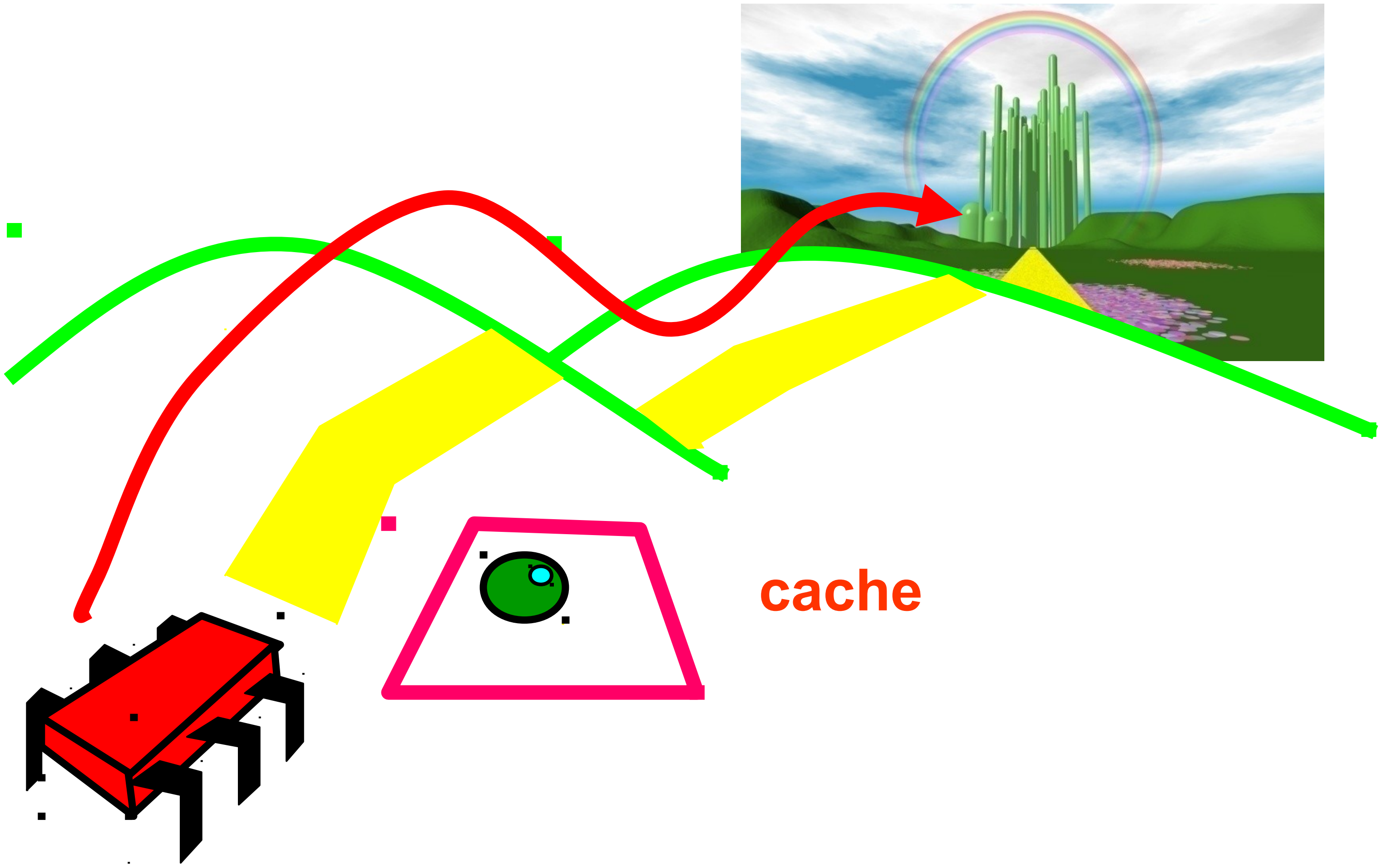


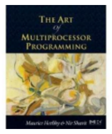
Cache Miss



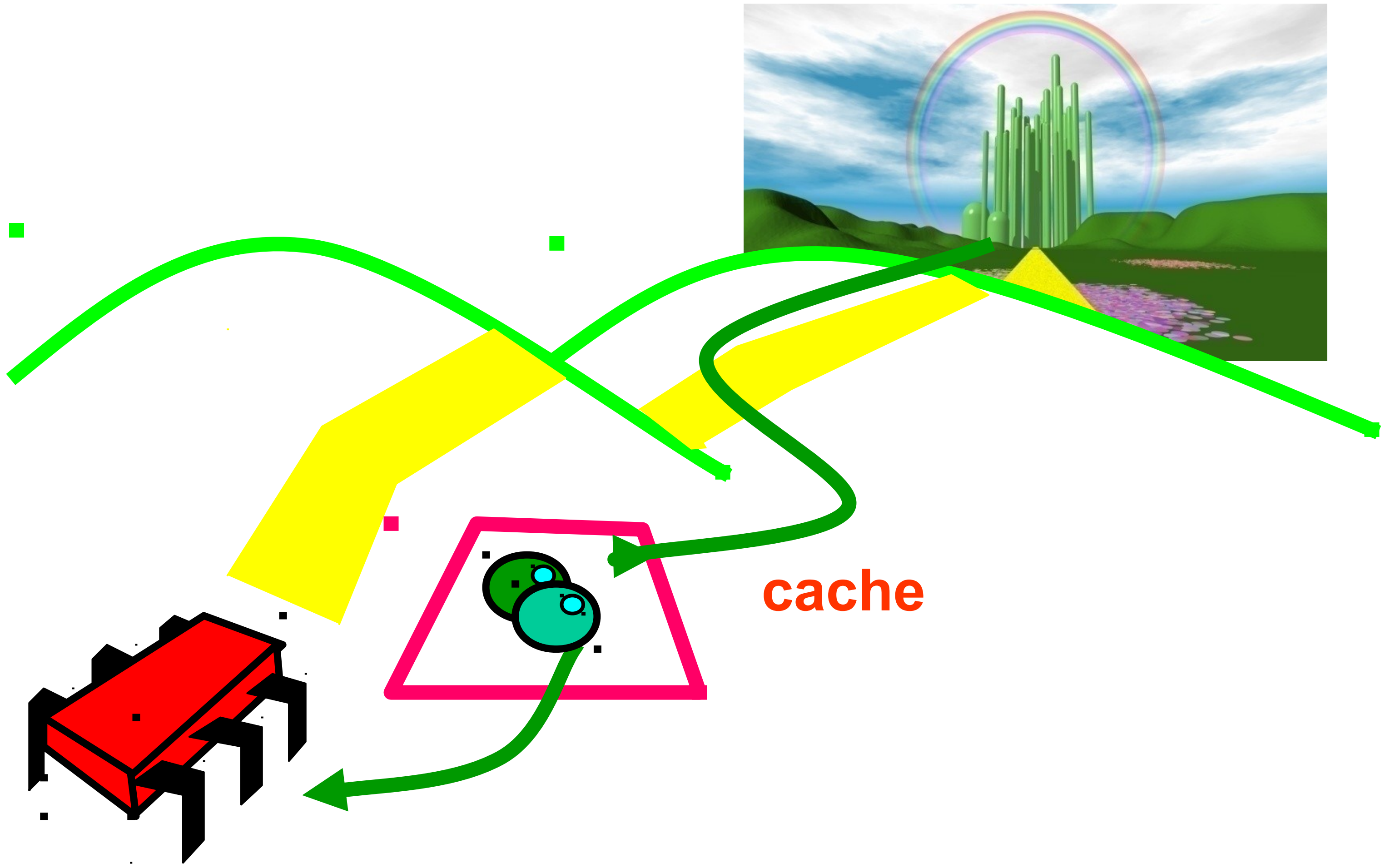


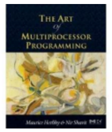
Cache Miss





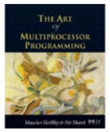
Cache Miss





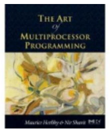
Local Spinning

- 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



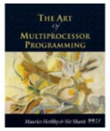
Granularity

- Caches operate at a larger granularity than a word
- Cache line: fixed-size block containing the address (today 64 or 128 bytes)



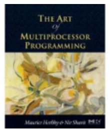
Locality

- 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

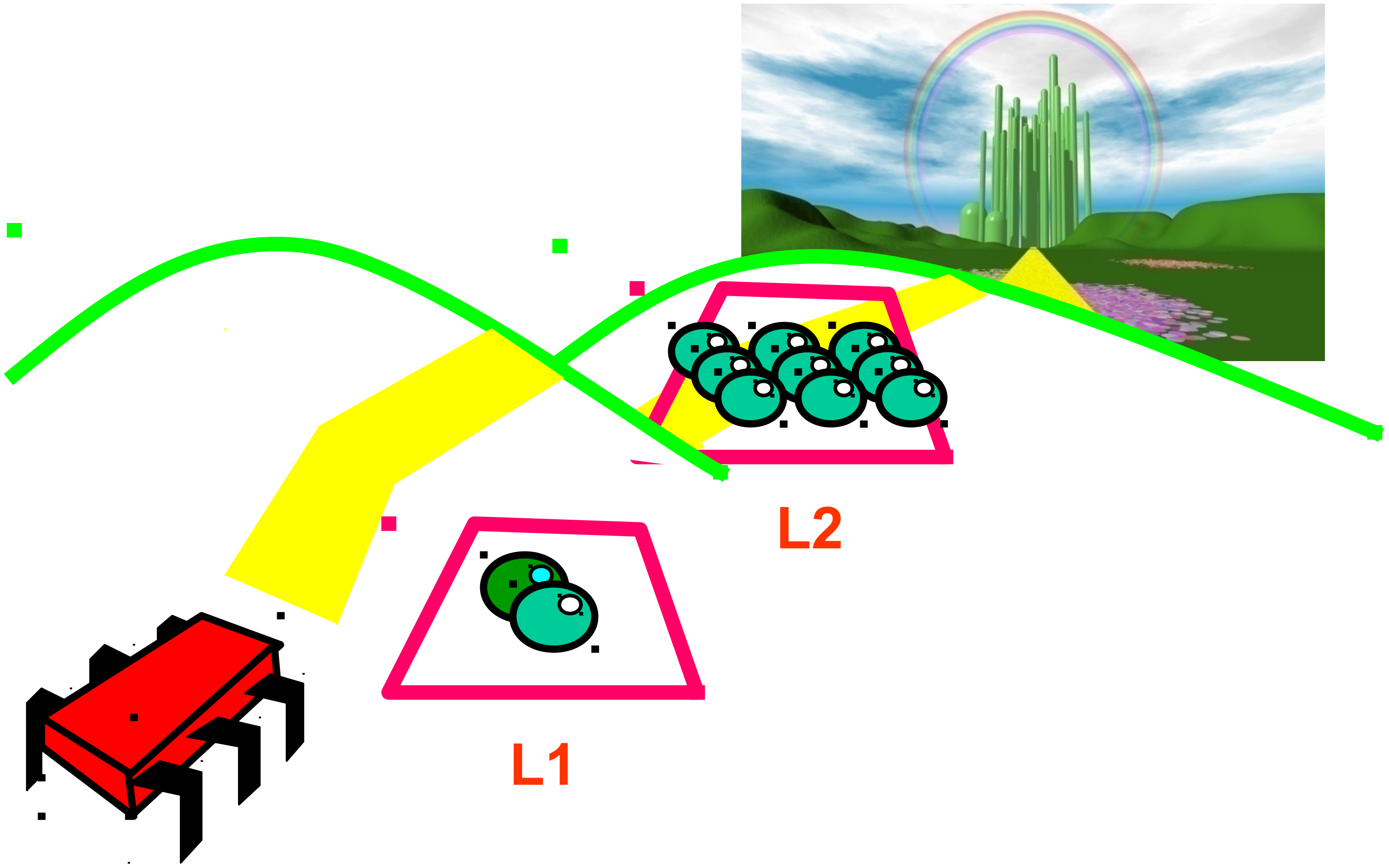


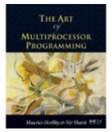
Hit Ratio

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

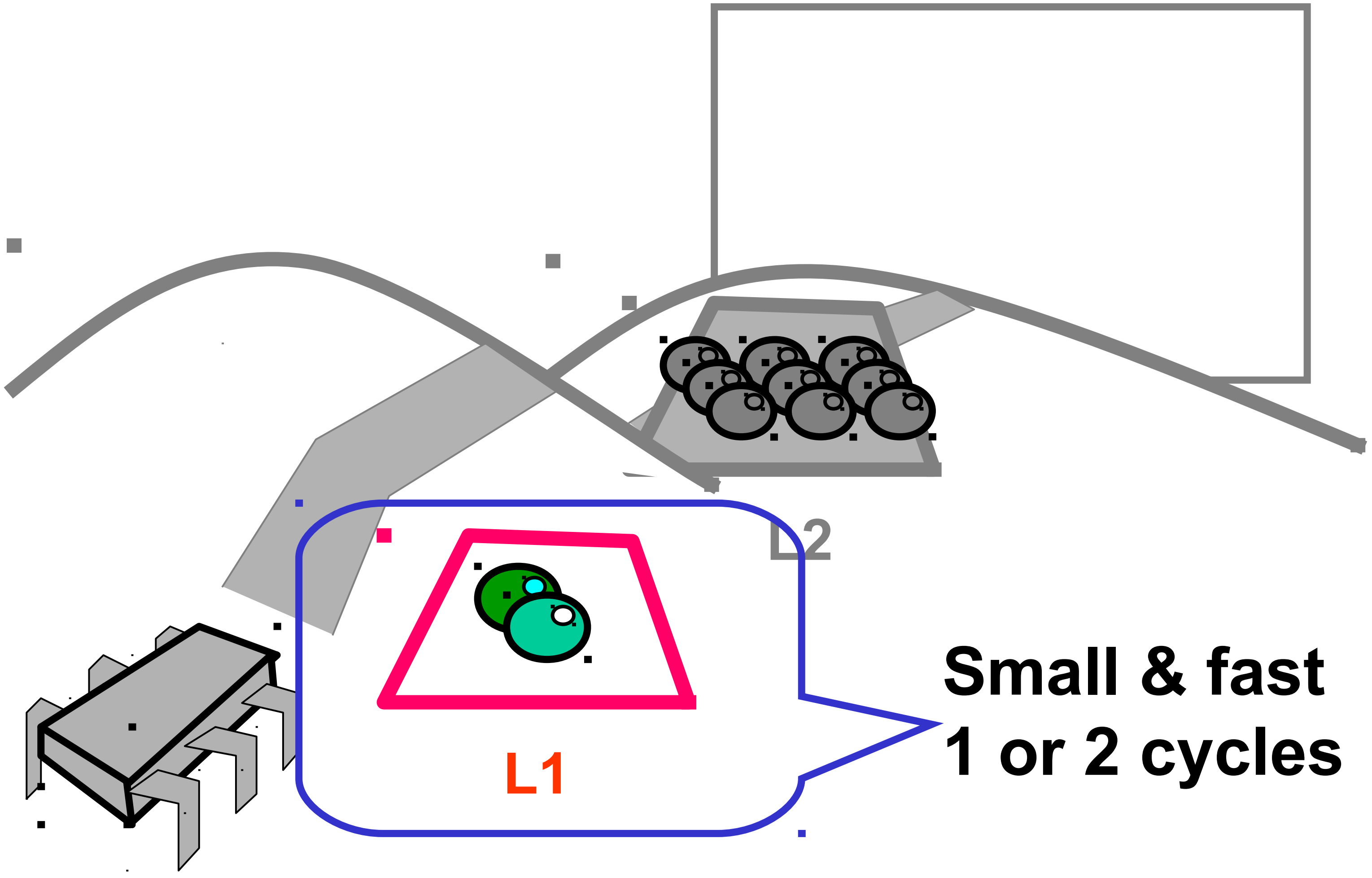


L1 and L2 Caches

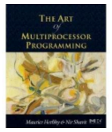




L1 and L2 Caches

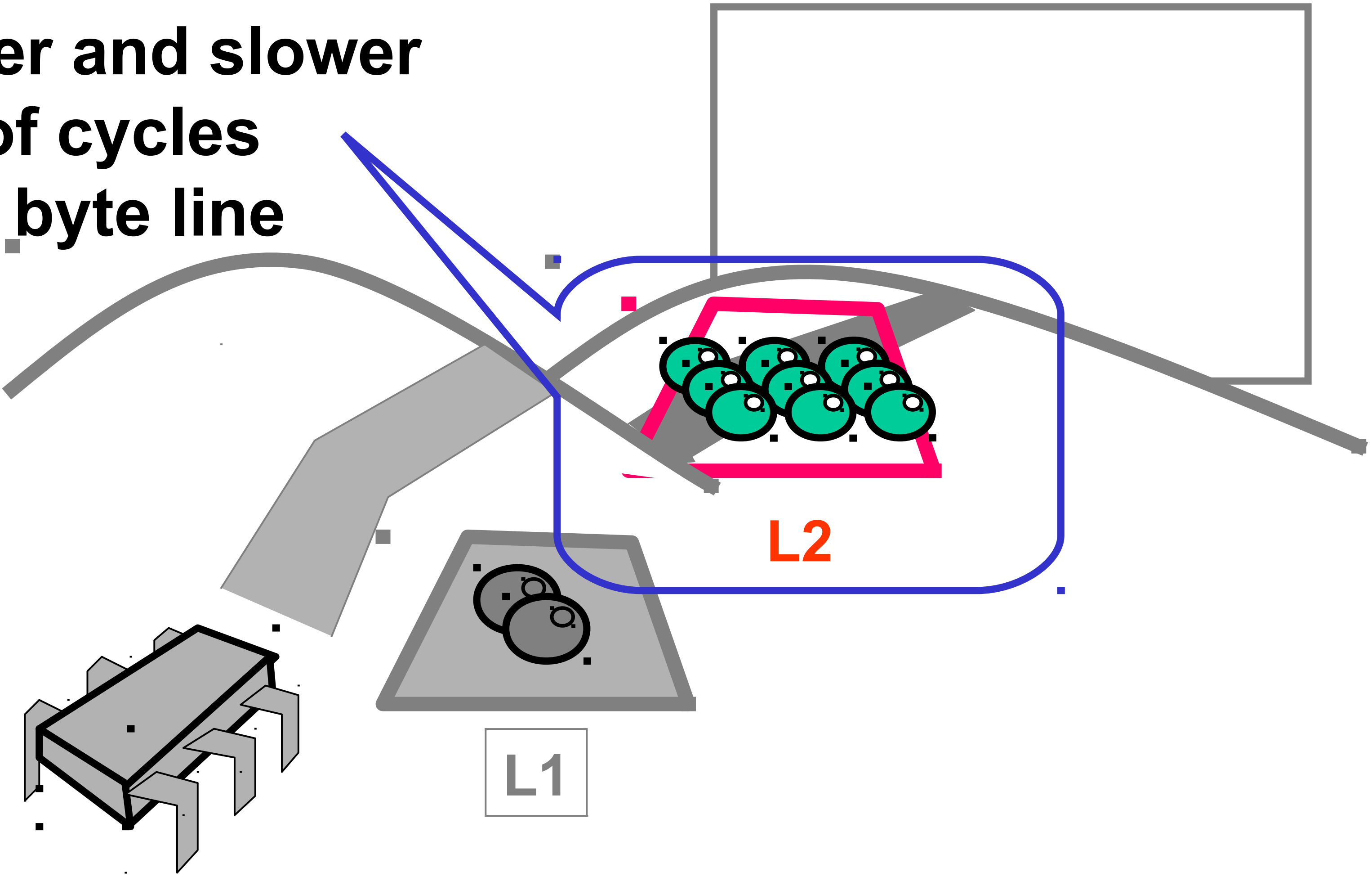


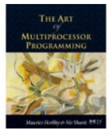
Small & fast
1 or 2 cycles



L1 and L2 Caches

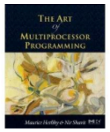
Larger and slower
10s of cycles
~128 byte line





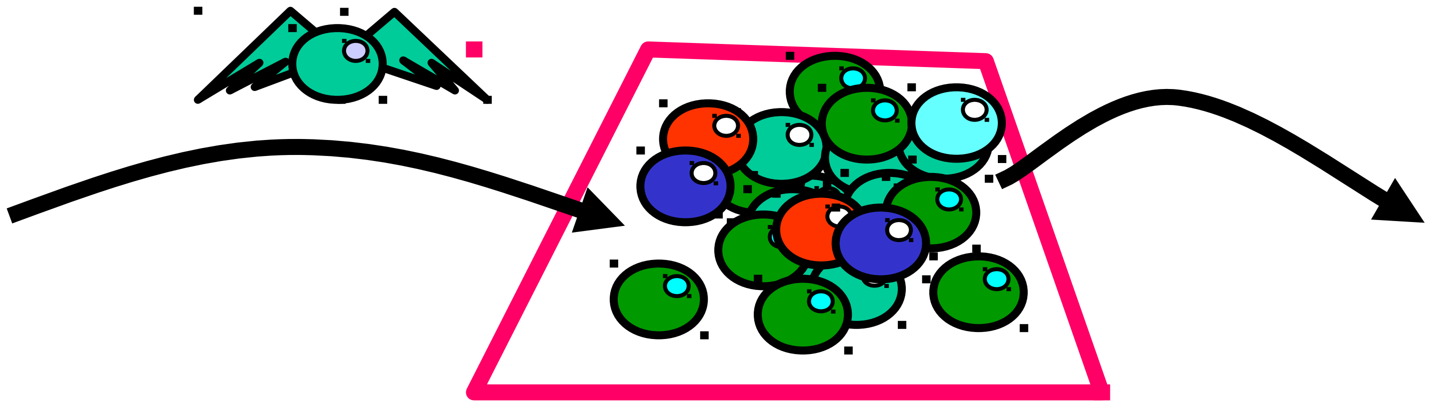
When a Cache Becomes Full…

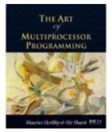
- Need to make room for new entry
- By evicting an existing entry
- Need a replacement policy
  - Usually some kind of least recently used heuristic



Fully Associative Cache

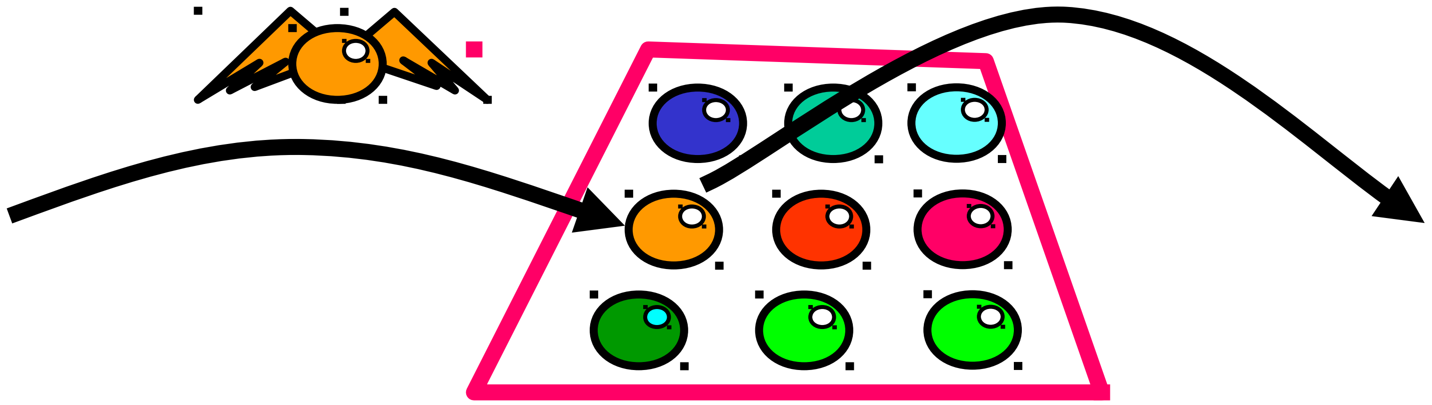
- Any line can be anywhere in the cache
  - Advantage: can replace any line
  - Disadvantage: hard to find lines

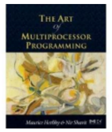




Direct Mapped Cache

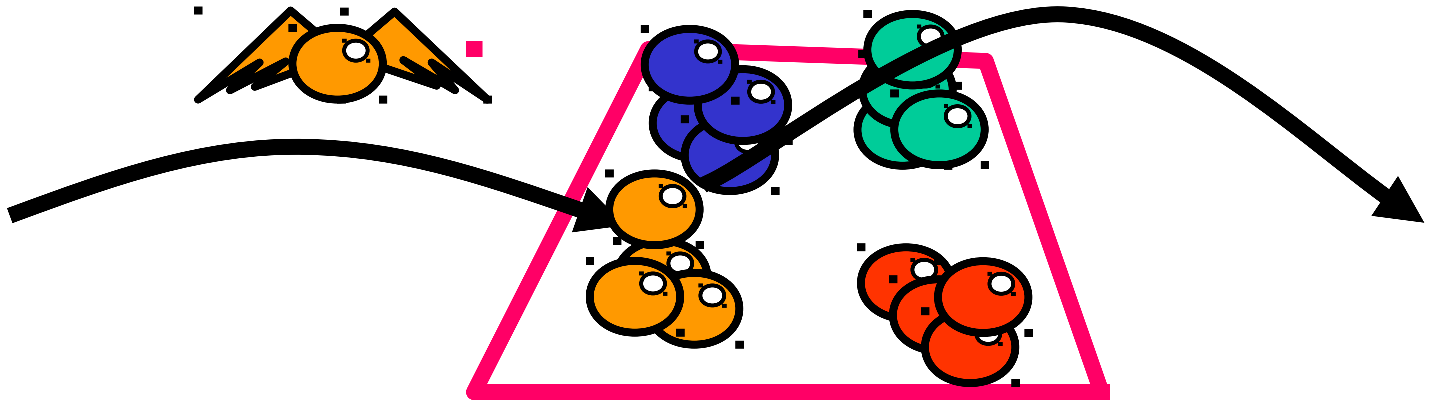
- Every address has exactly 1 slot
  - Advantage: easy to find a line
  - Disadvantage: must replace fixed line

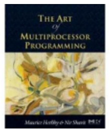




K-way Set Associative Cache

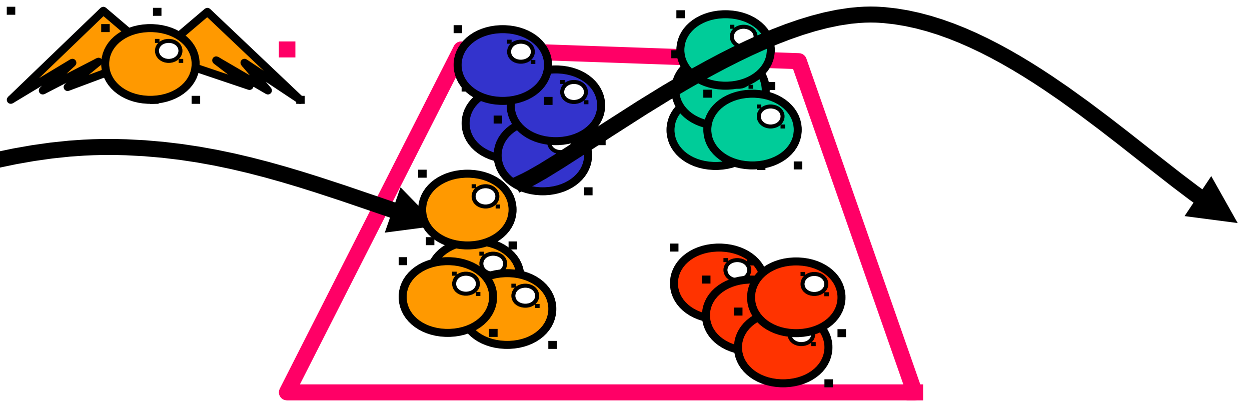
- Each slot holds k lines
  - Advantage: pretty easy to find a line
  - Advantage: some choice in replacing line

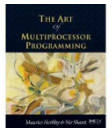




Multicore Set Associativity

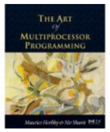
- k is 8 or even 16 and growing…
  - Why? Because cores share sets
  - Threads cut effective size if accessing different data





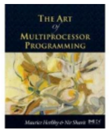
Cache Coherence

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



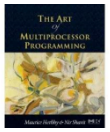
MESI

- Modified
  - Have modified cached data, must write back to memory



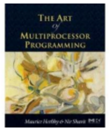
MESI

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



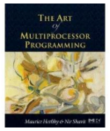
MESI

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

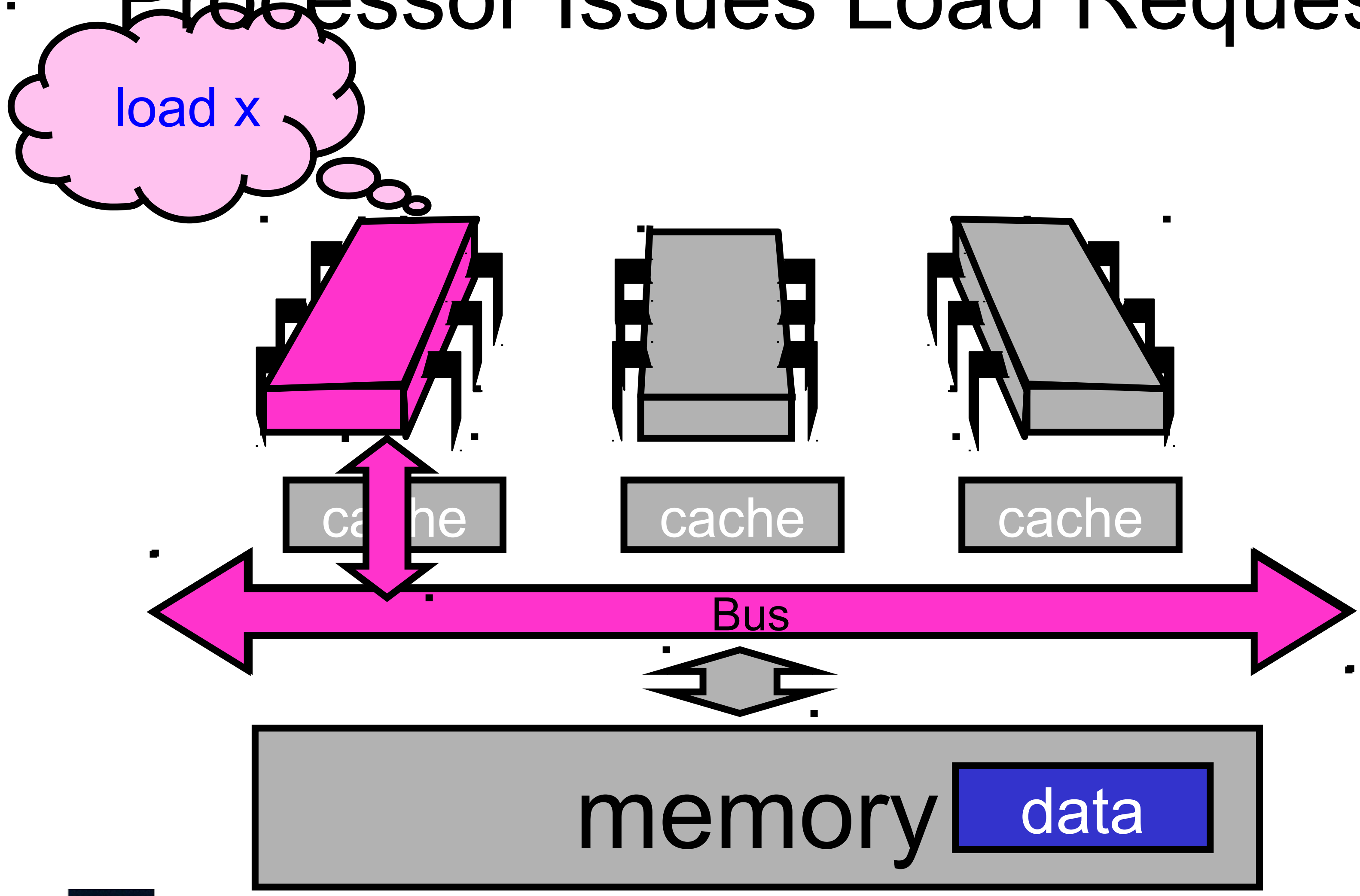


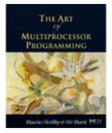
MESI

- 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

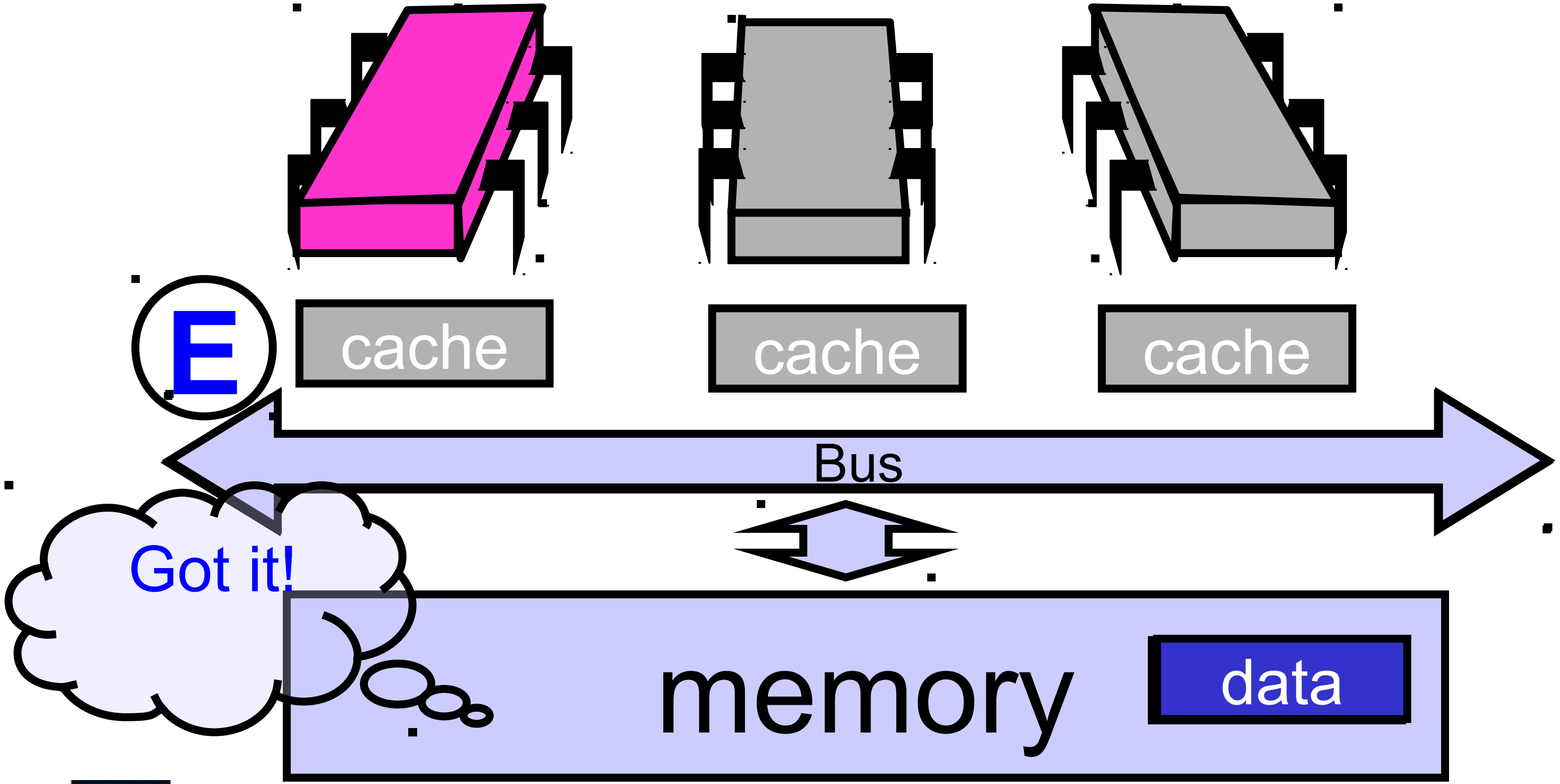


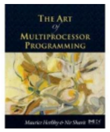
Processor Issues Load Request



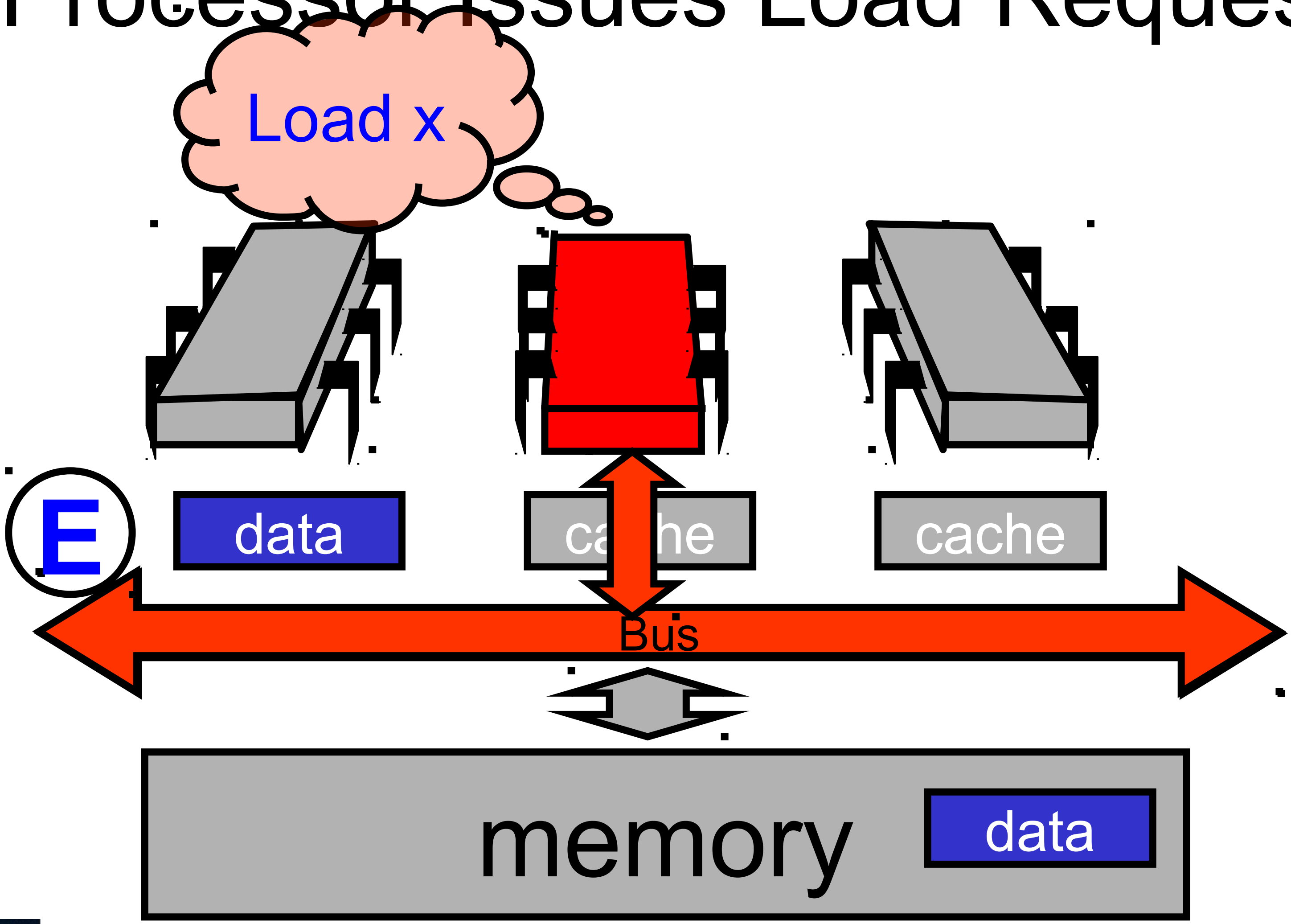


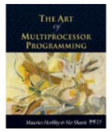
Memory Responds



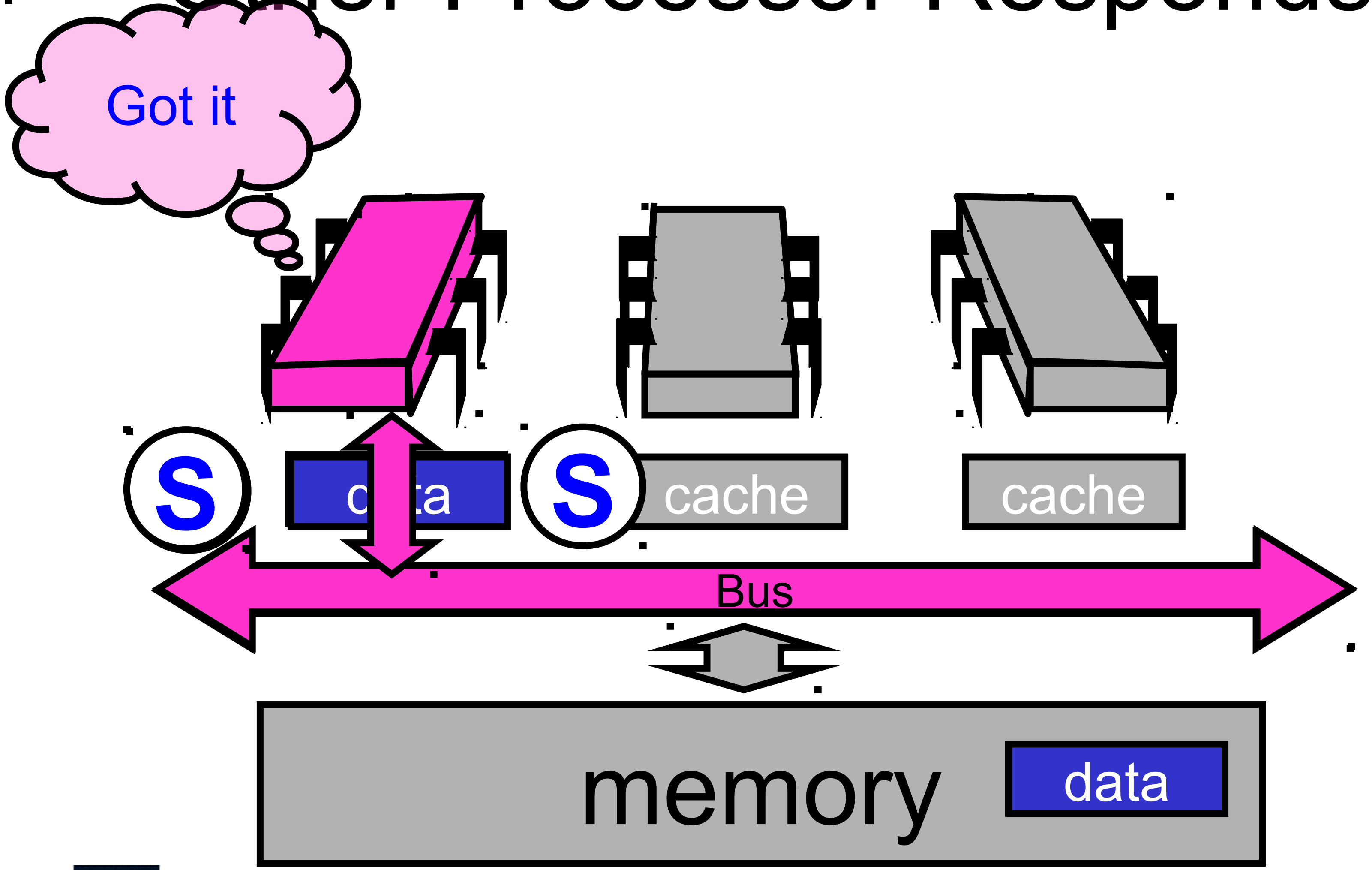


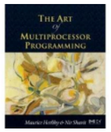
Processor Issues Load Request



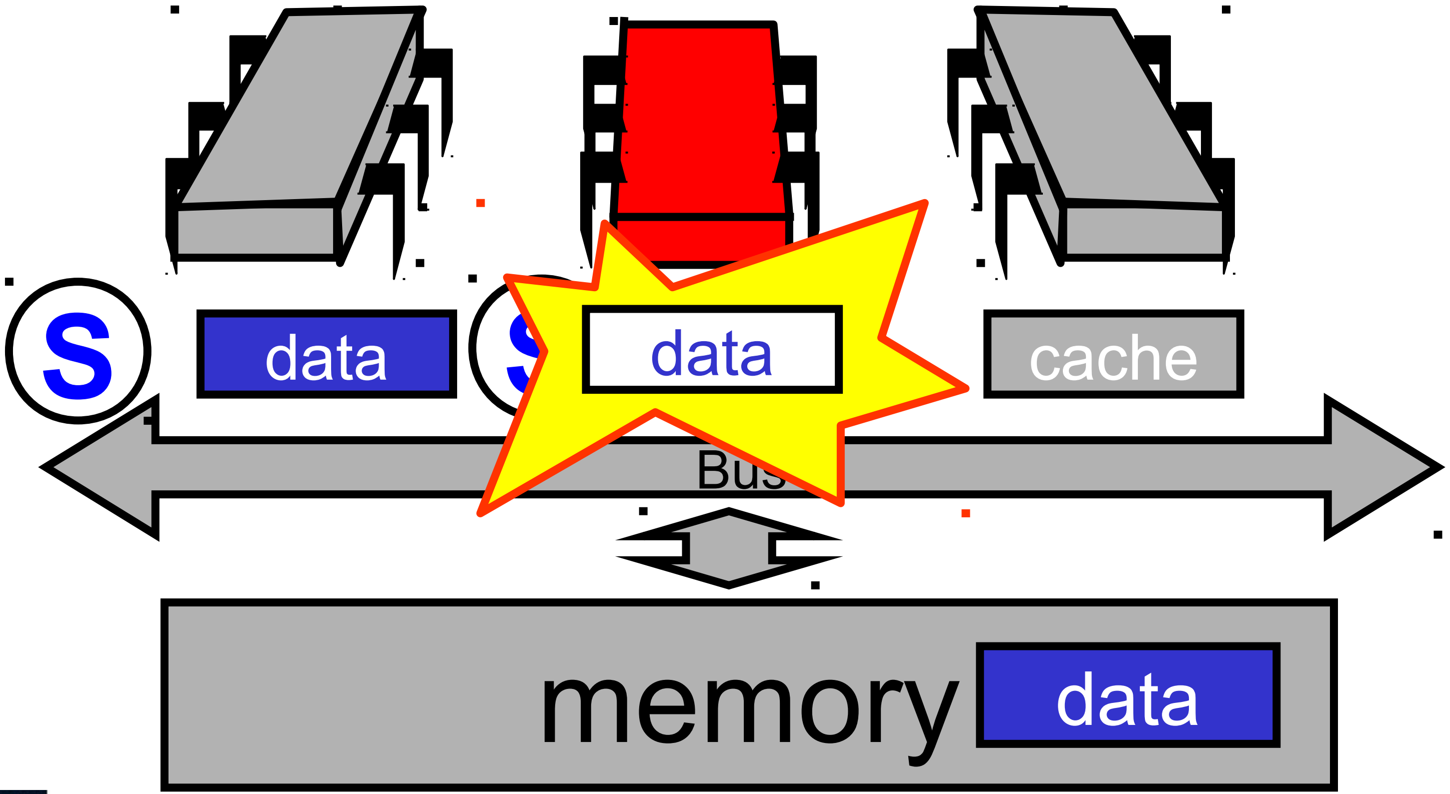


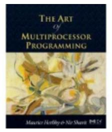
Other Processor Responds



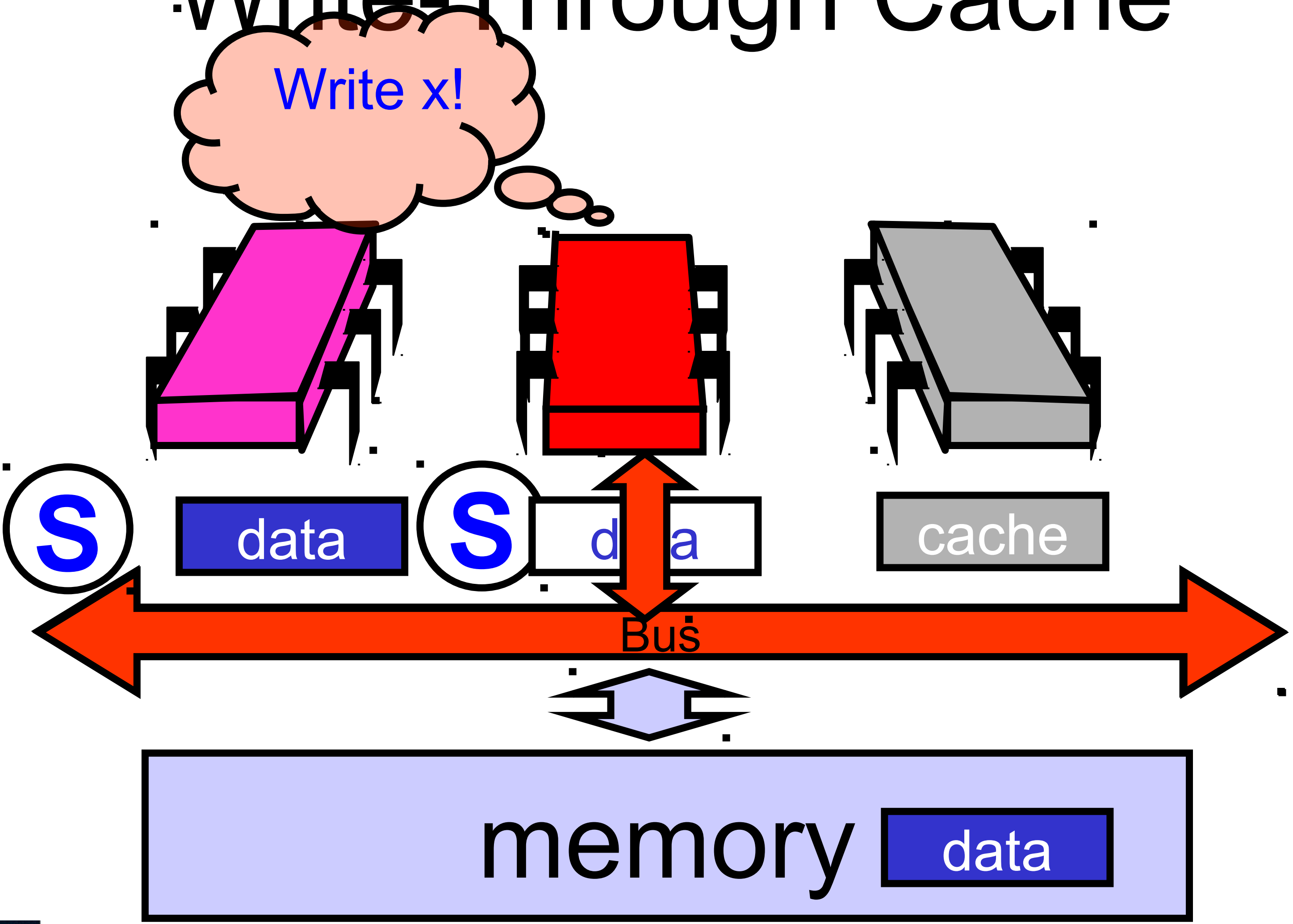


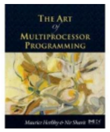
Modify Cached Data





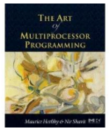
Write-Through Cache





Write-Through Caches

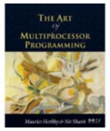
- Immediately broadcast changes
- Good
  - Memory, caches always agree
  - More read hits, maybe
- Bad
  - Bus traffic on all writes
  - Most writes to unshared data
  - For example, loop indexes …



Write-Through Caches

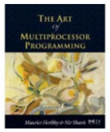
- Immediately broadcast changes
- Good
  - Memory, caches always agree
  - More read hits, maybe
- Bad
  - Bus traffic on all writes
  - Most writes to unshared data
  - For example, loop indexes …

"show stoppers"

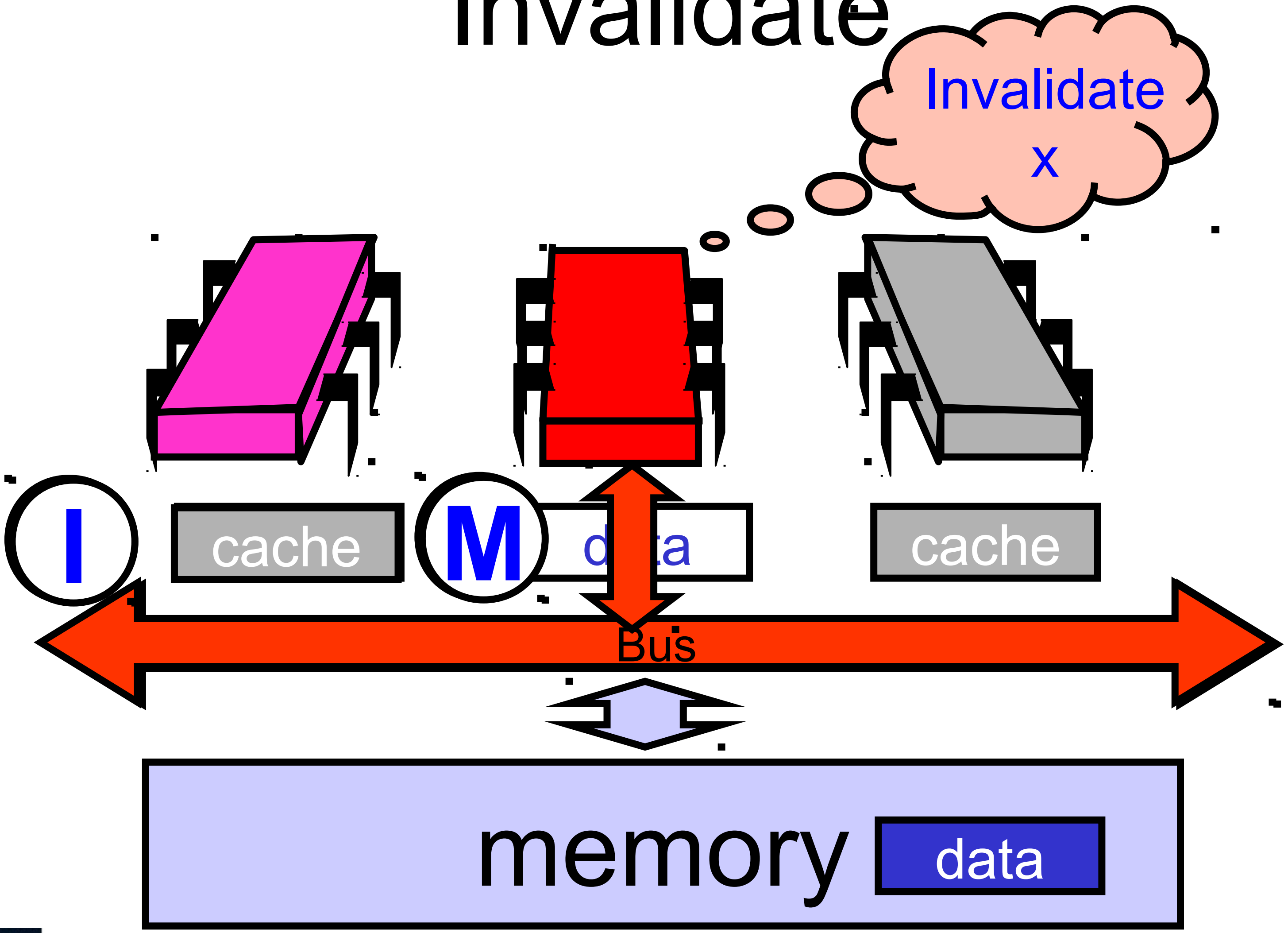


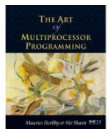
Write-Back Caches

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



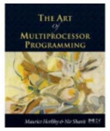
Invalidate



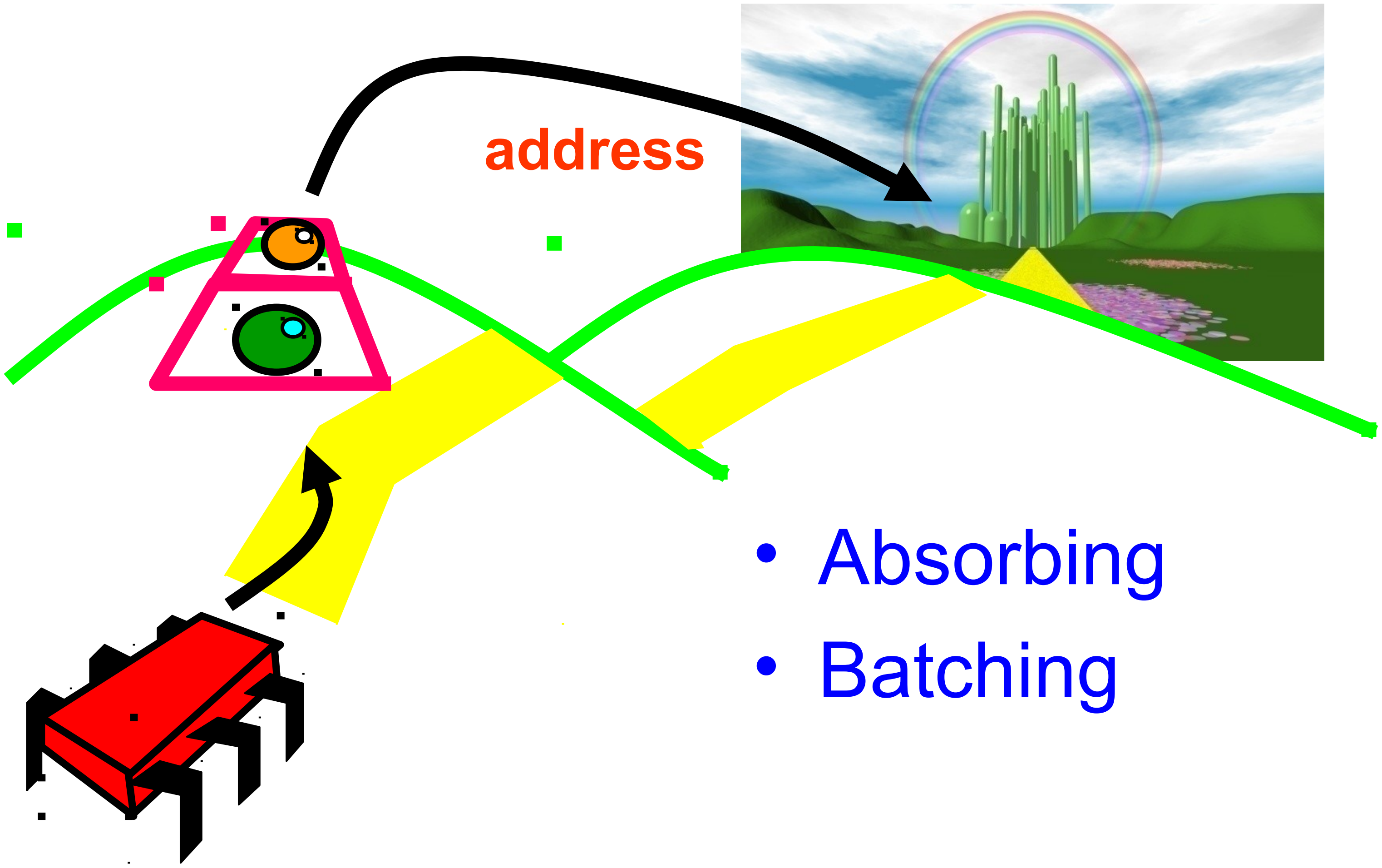


Problem

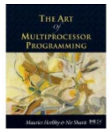
- Sometimes the compiler reorders memory operations
- Can improve
  - cache performance
  - interconnect use
- But unexpected concurrent interactions



Write Buffers



- Absorbing
- Batching



Volatile

- In Java, if a variable is declared volatile, operations won't be reordered
- Write buffer always spilled to memory before thread is allowed to continue a write
- Expensive, so use it only when needed