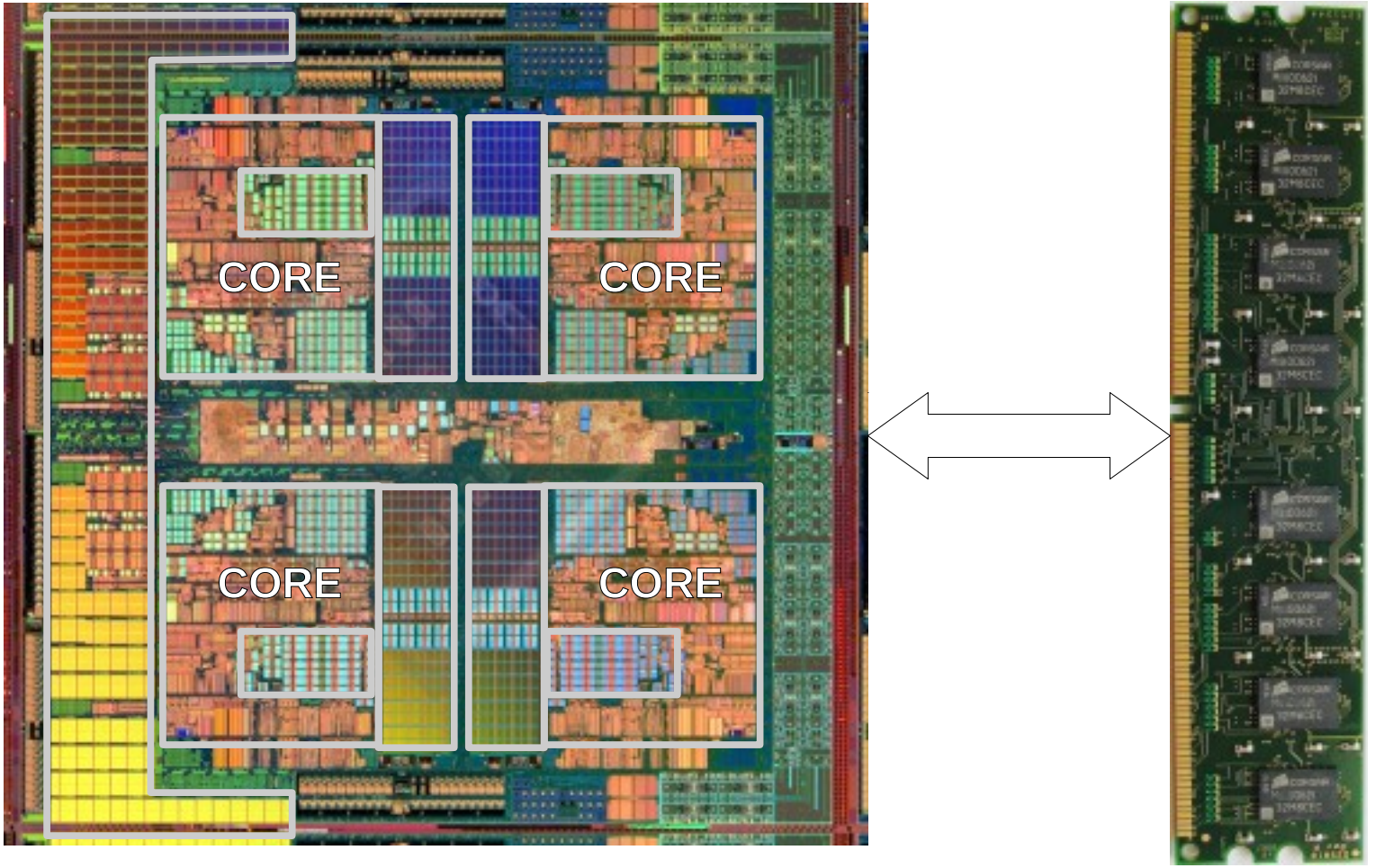
Data Movement Control on a PowerPC

Silas Boyd-Wickizer and Asif Khan

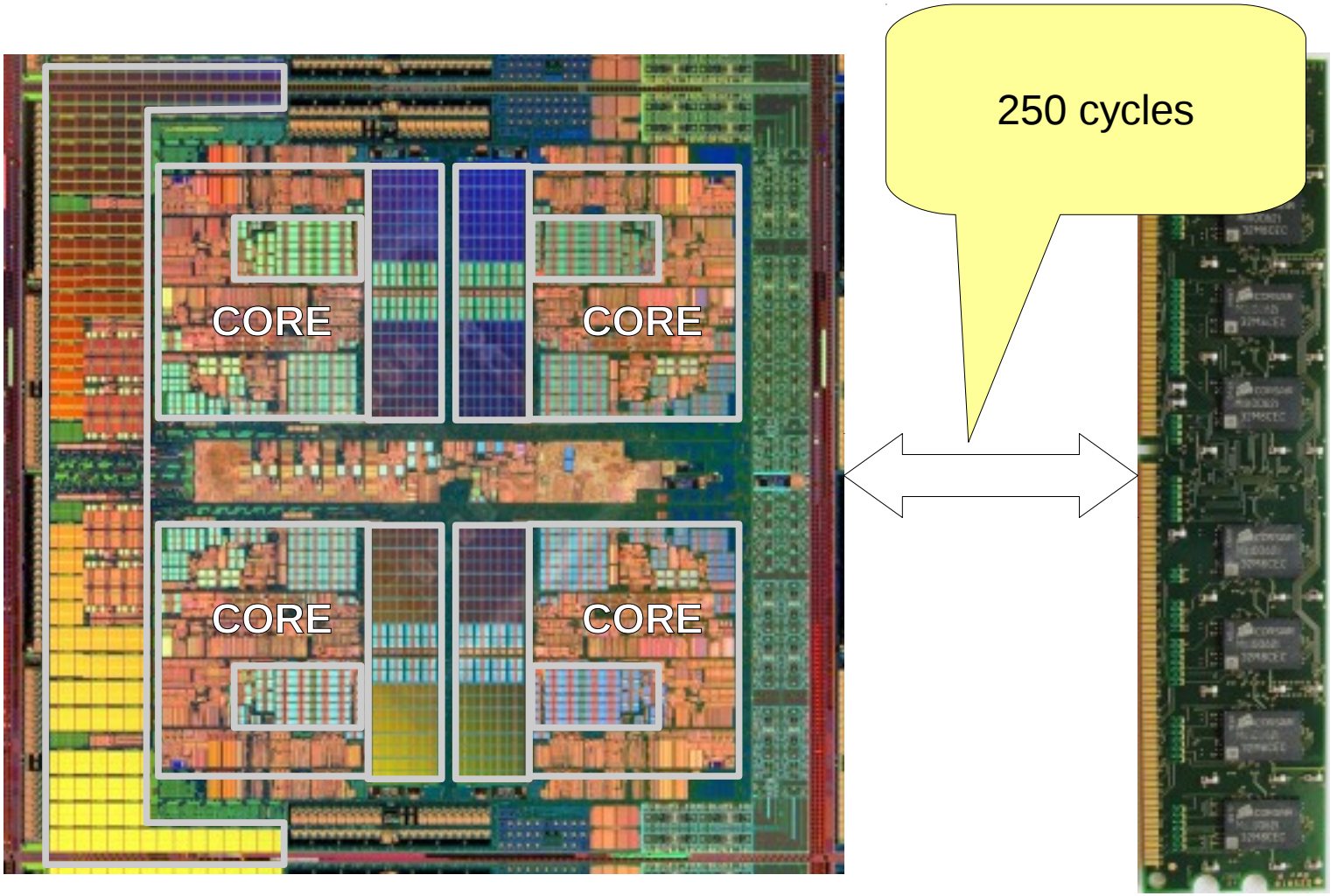
What this presentation is about

- Intuition for why multicore caches are underutilized
- Preliminary design for three new instructions
  - Toy benchmarks show improved performance

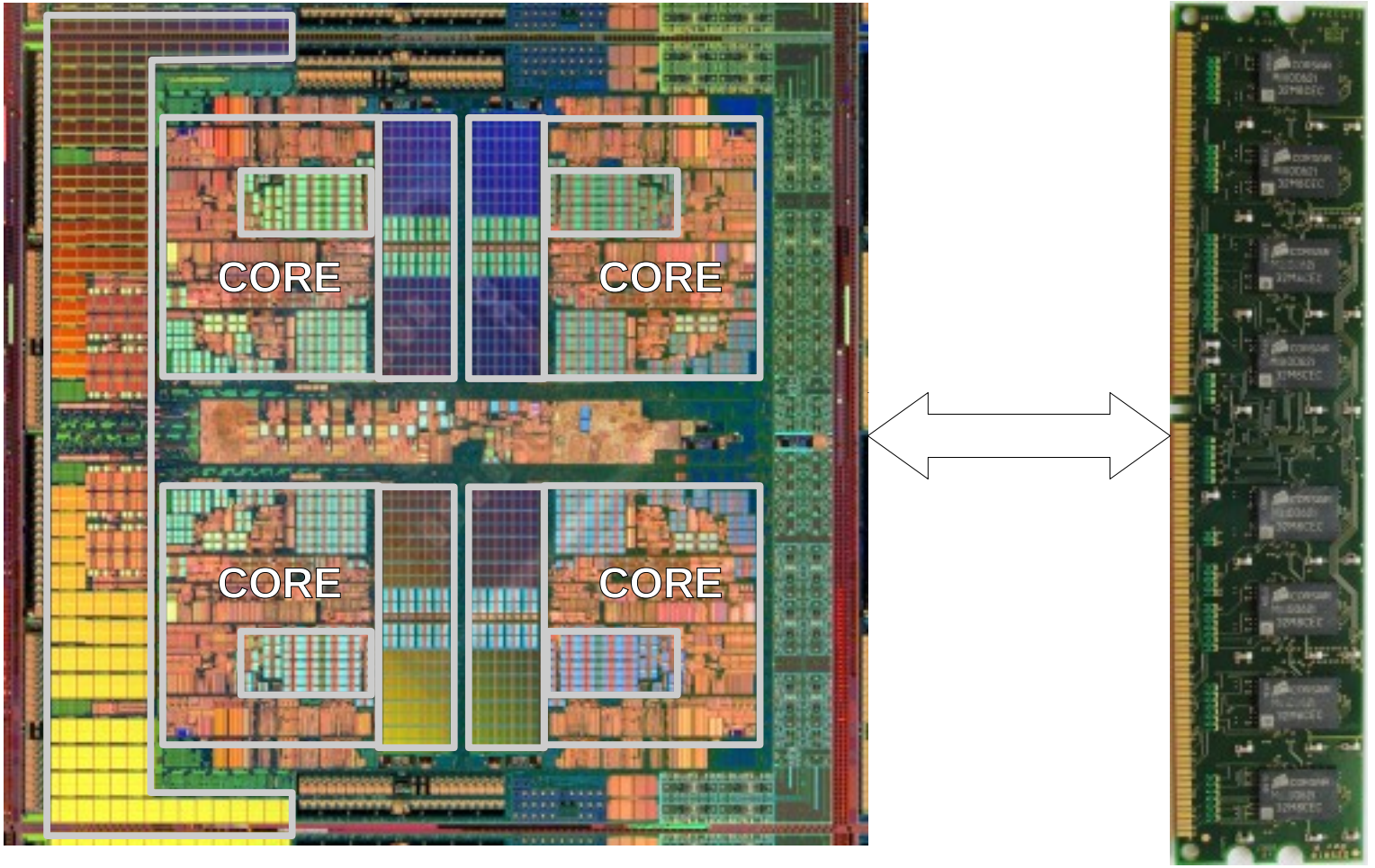
Caches are crucial for performance



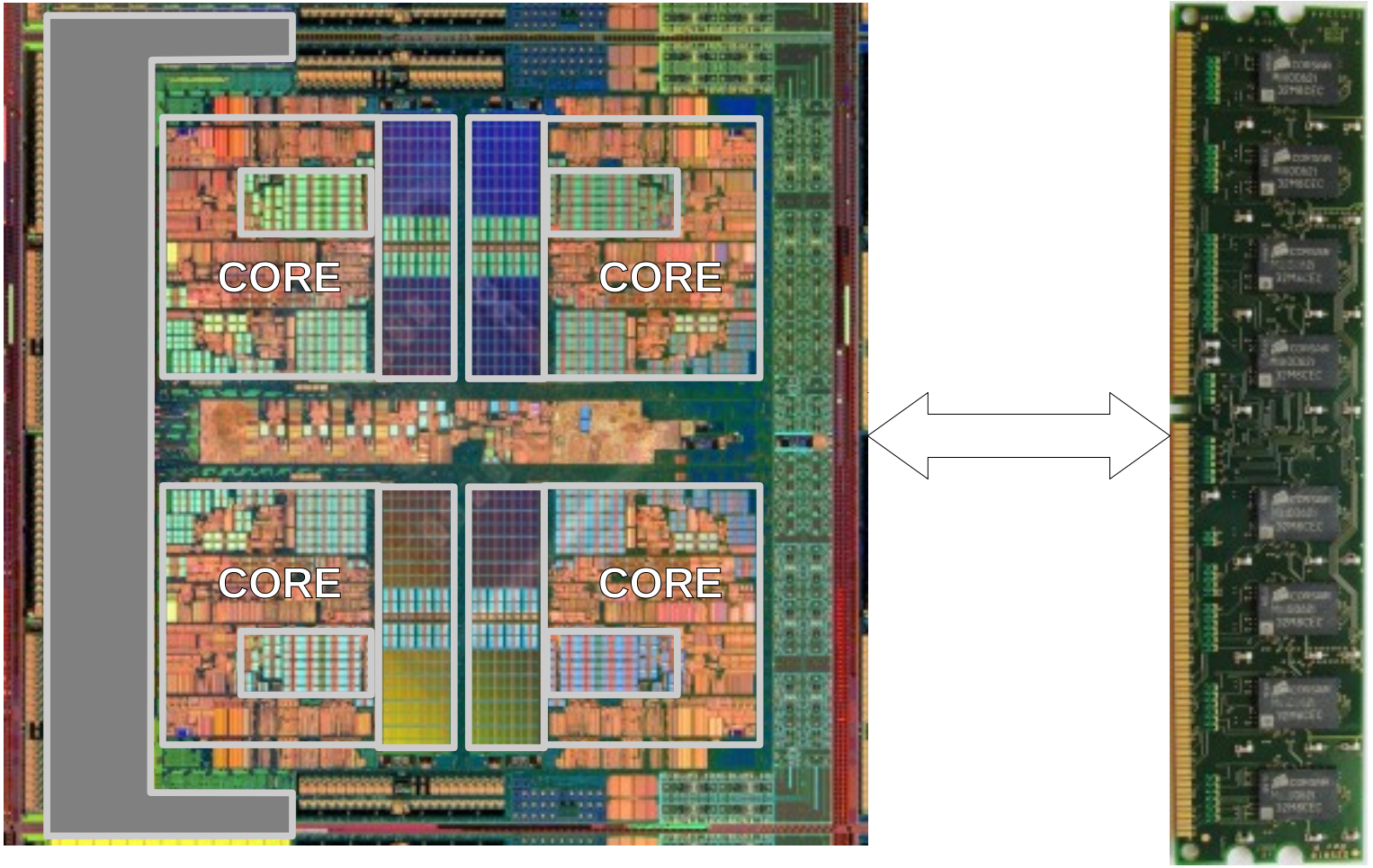
Caches are crucial for performance



Potential solution is one giant shared cache

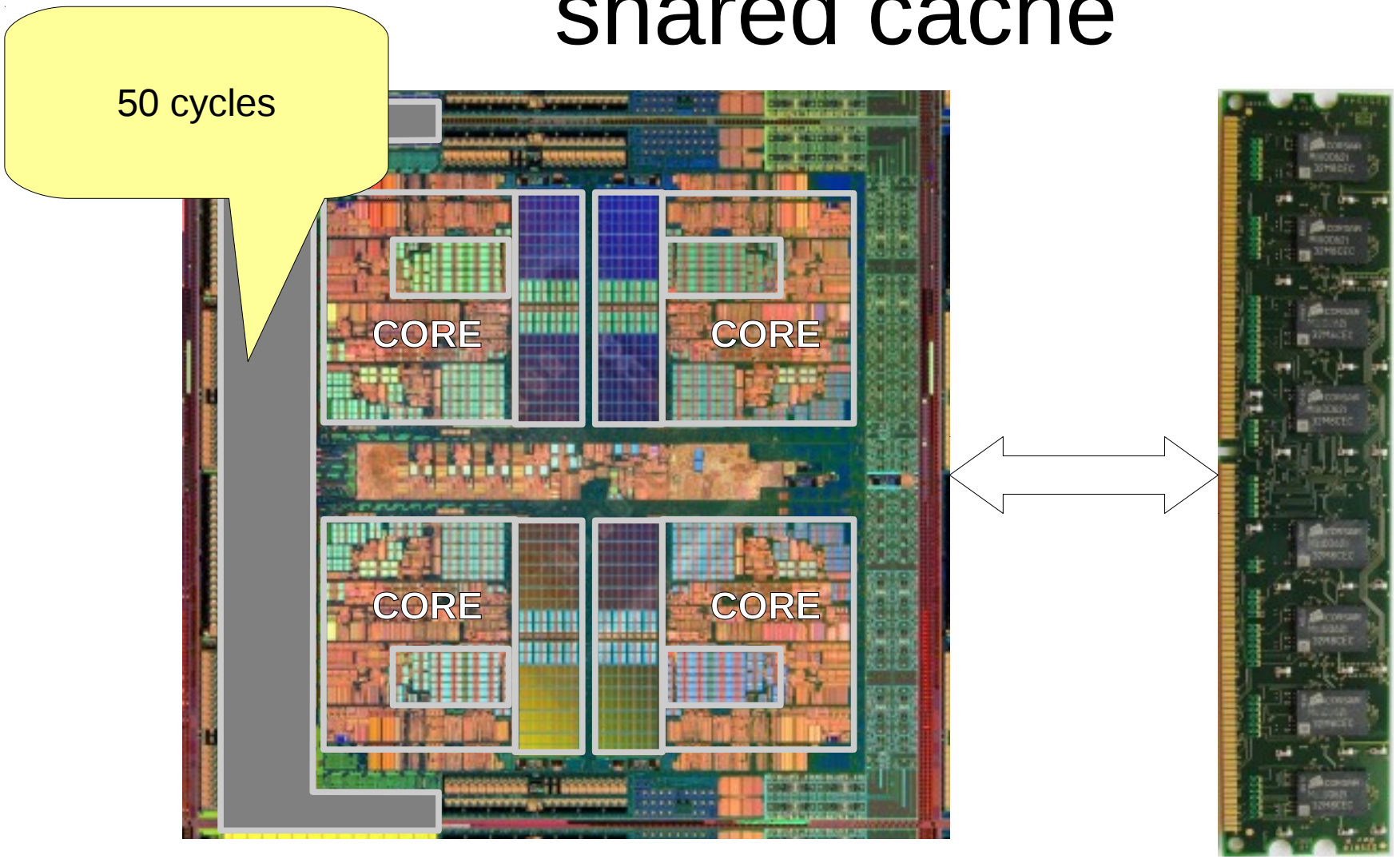


Potential solution is one giant shared cache



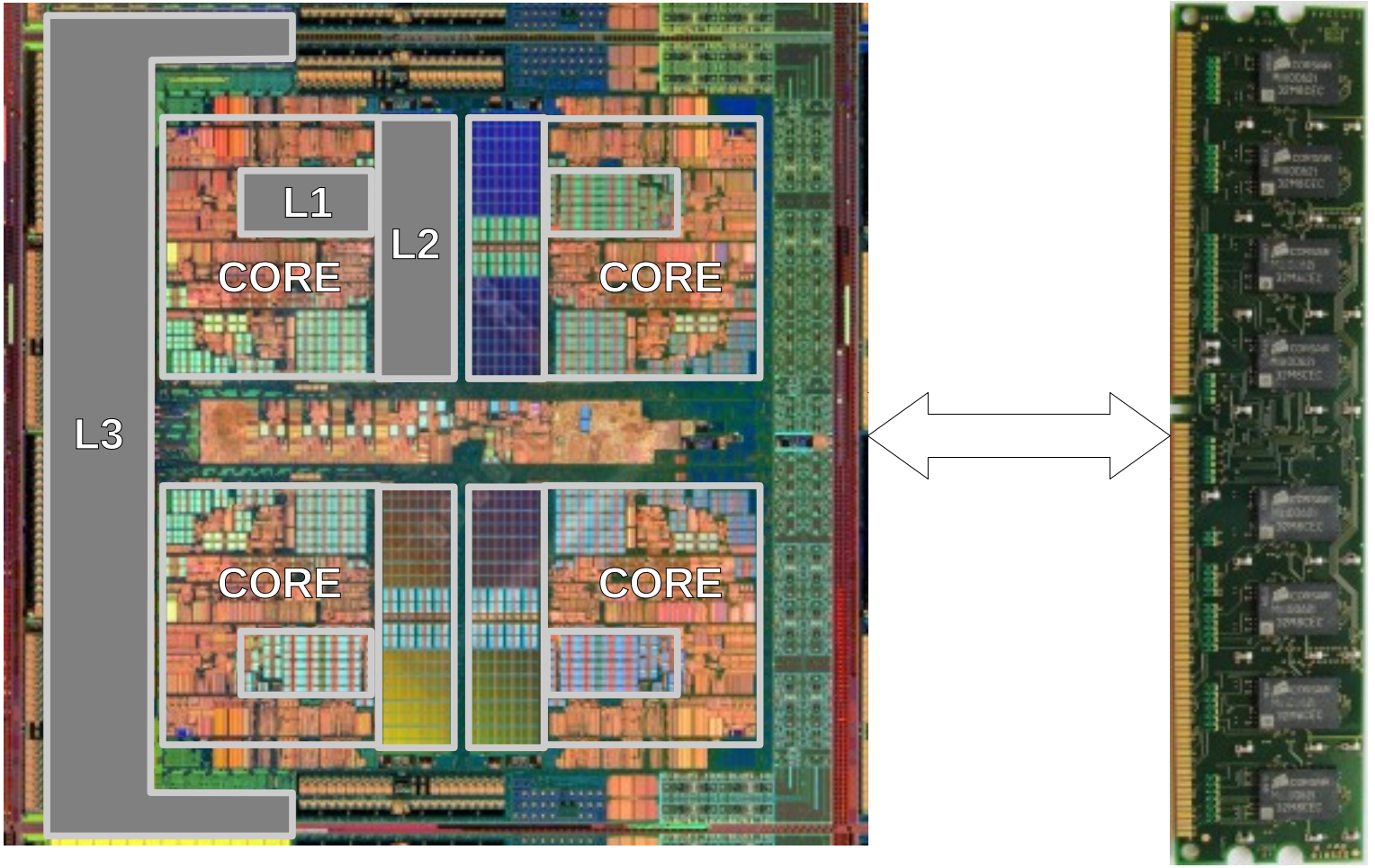
- Applications have access to entire cache capacity, no false sharing issues, etc.

Potential solution is one giant shared cache

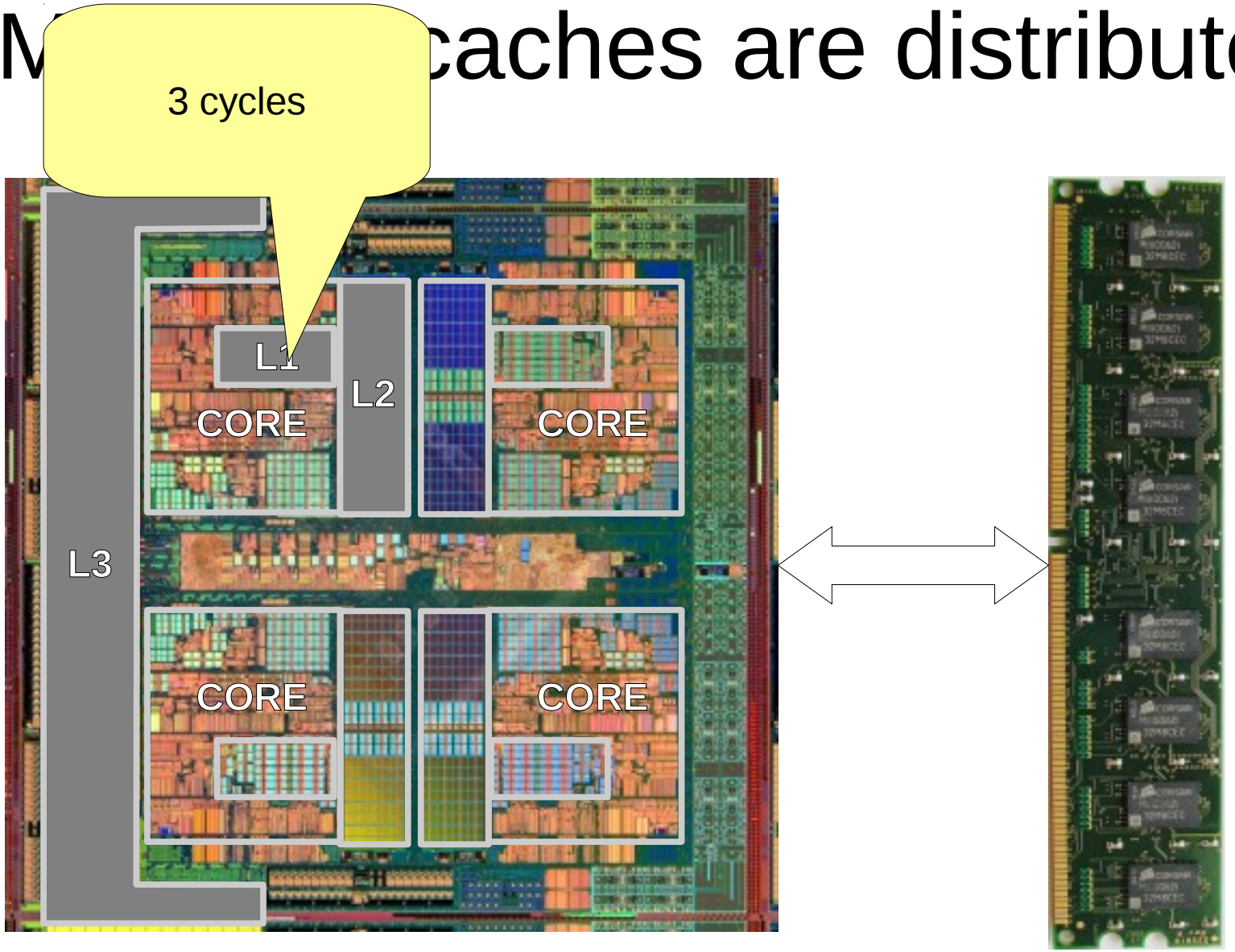


- Applications have access to entire cache capacity, no false sharing issues, etc.

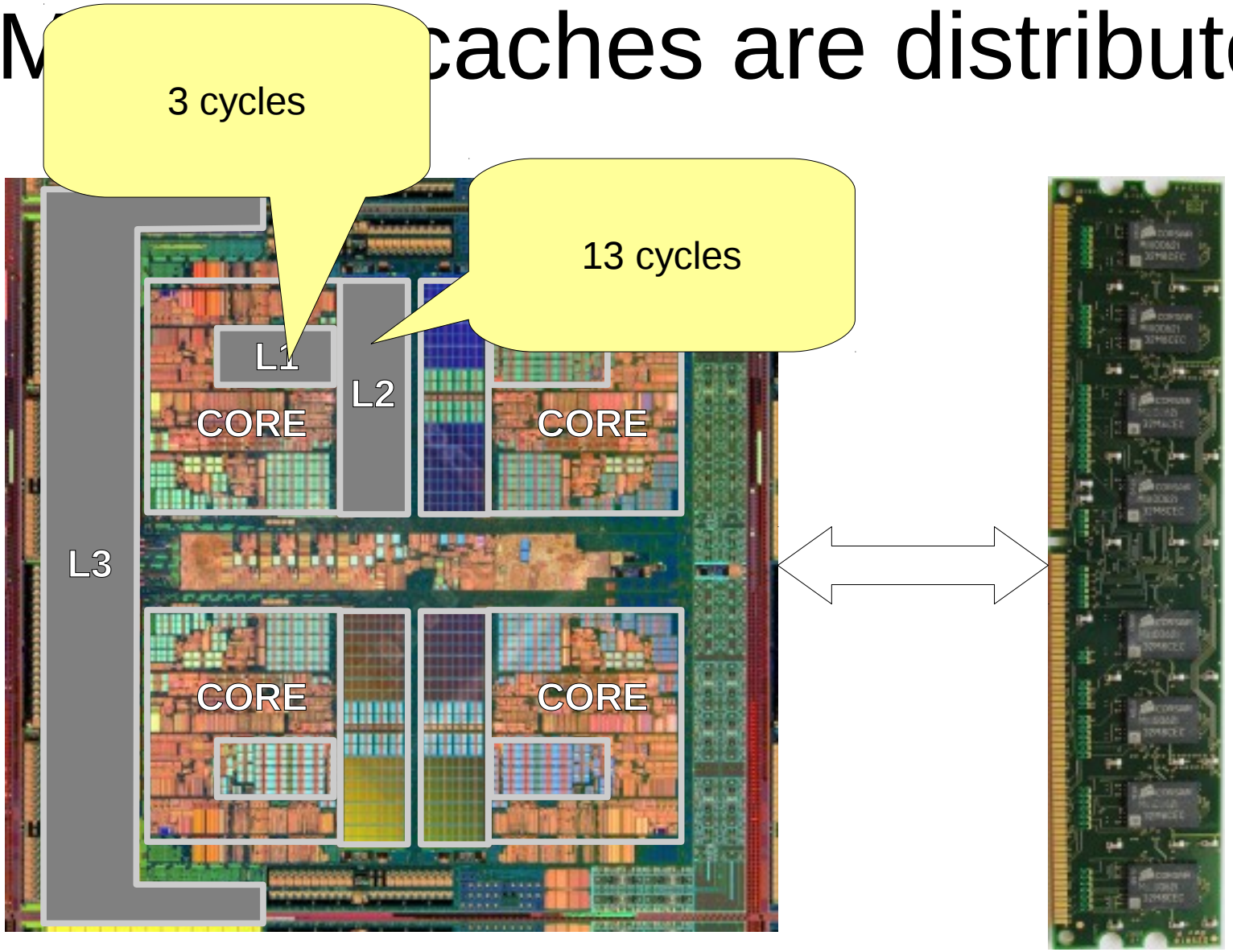
Multicore caches are distributed



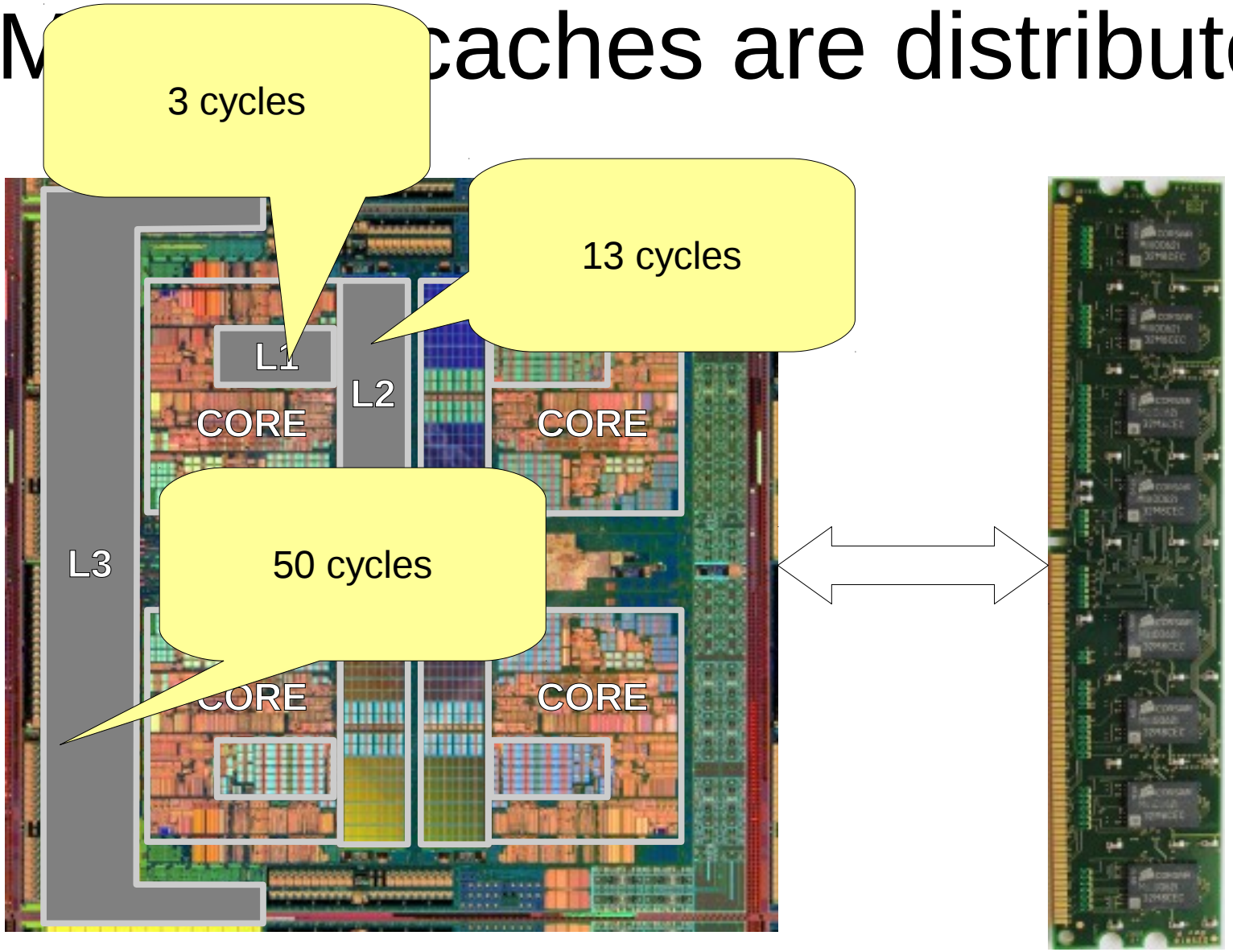
M caches are distributed



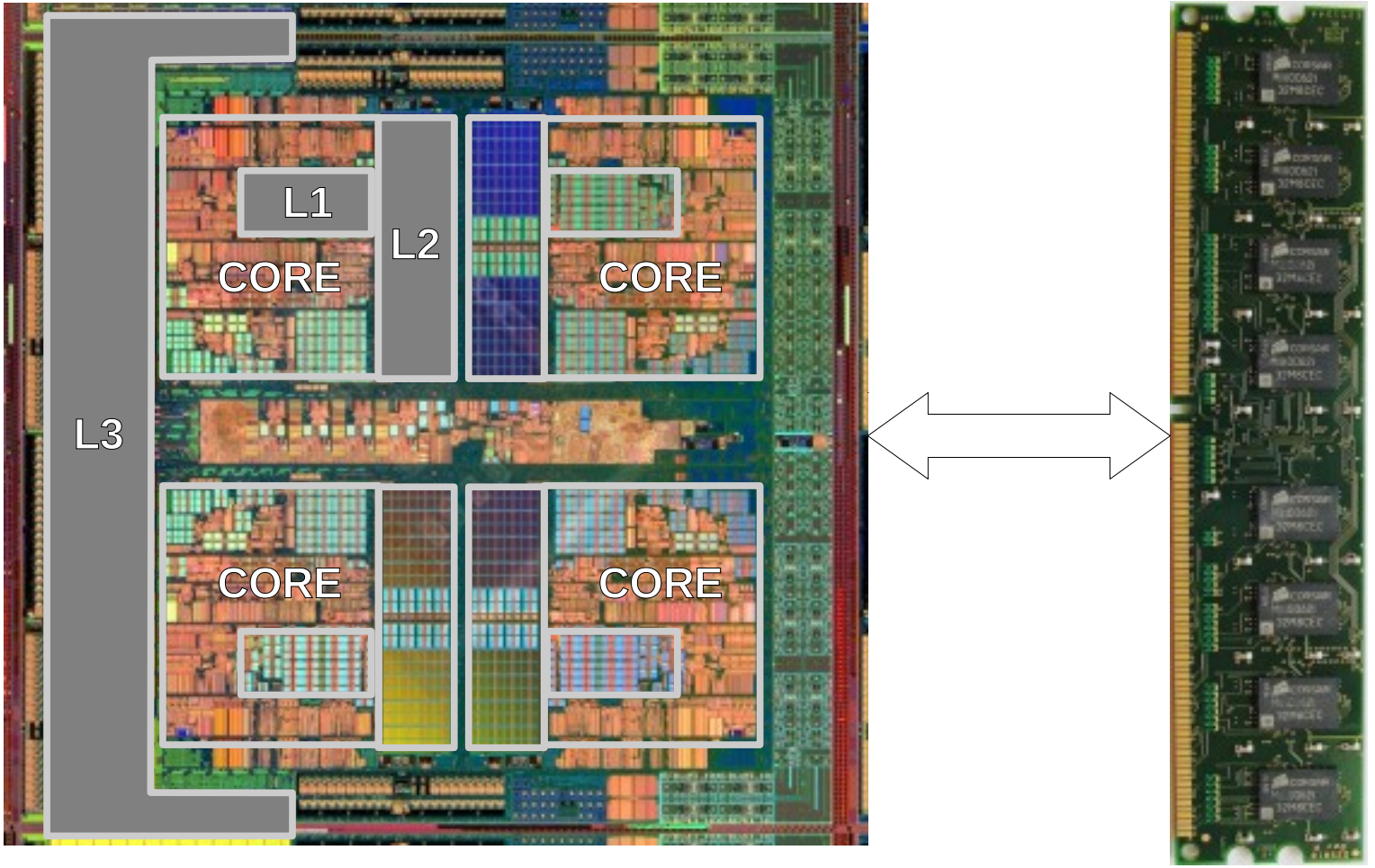
M caches are distributed



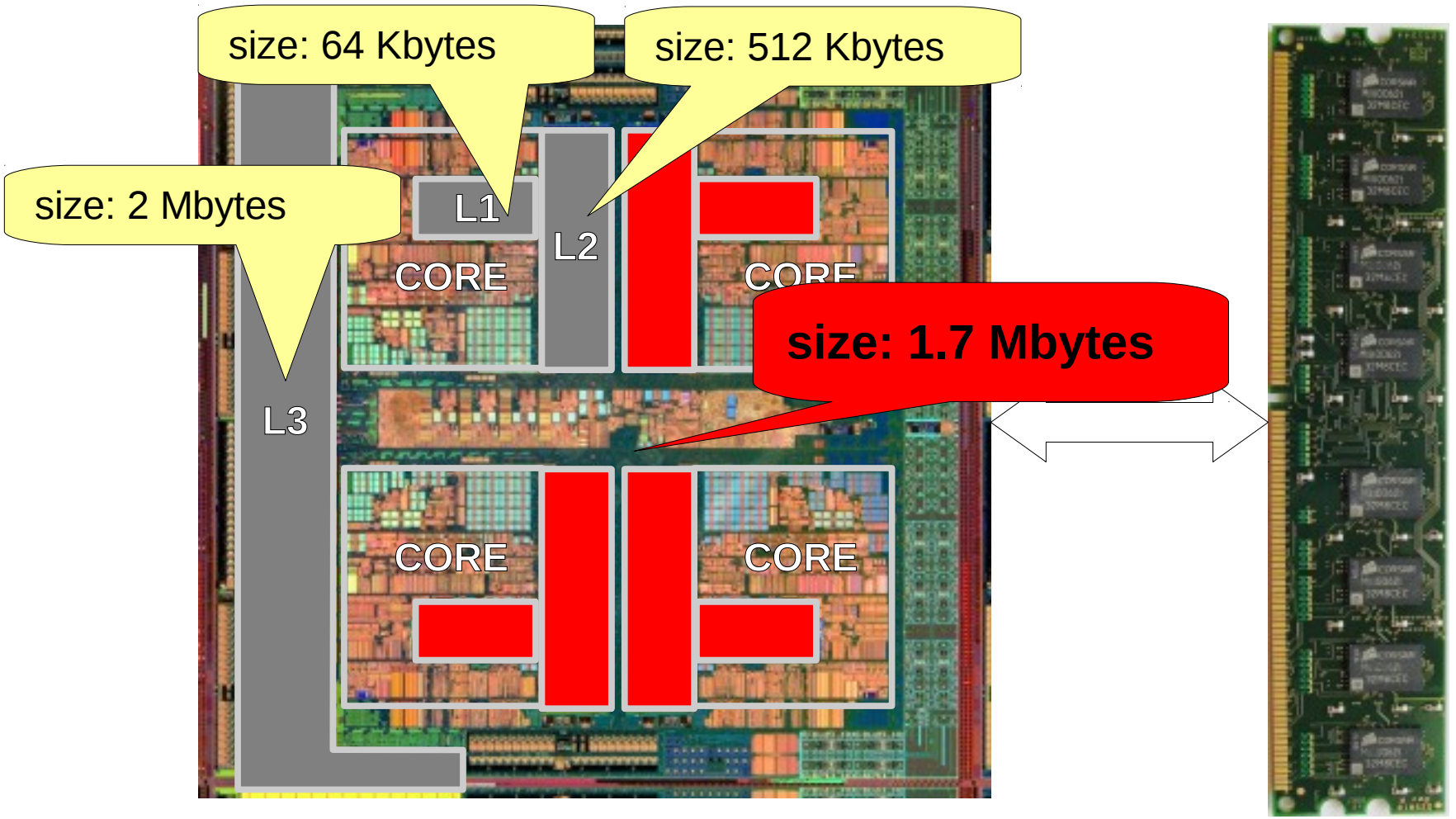
M caches are distributed



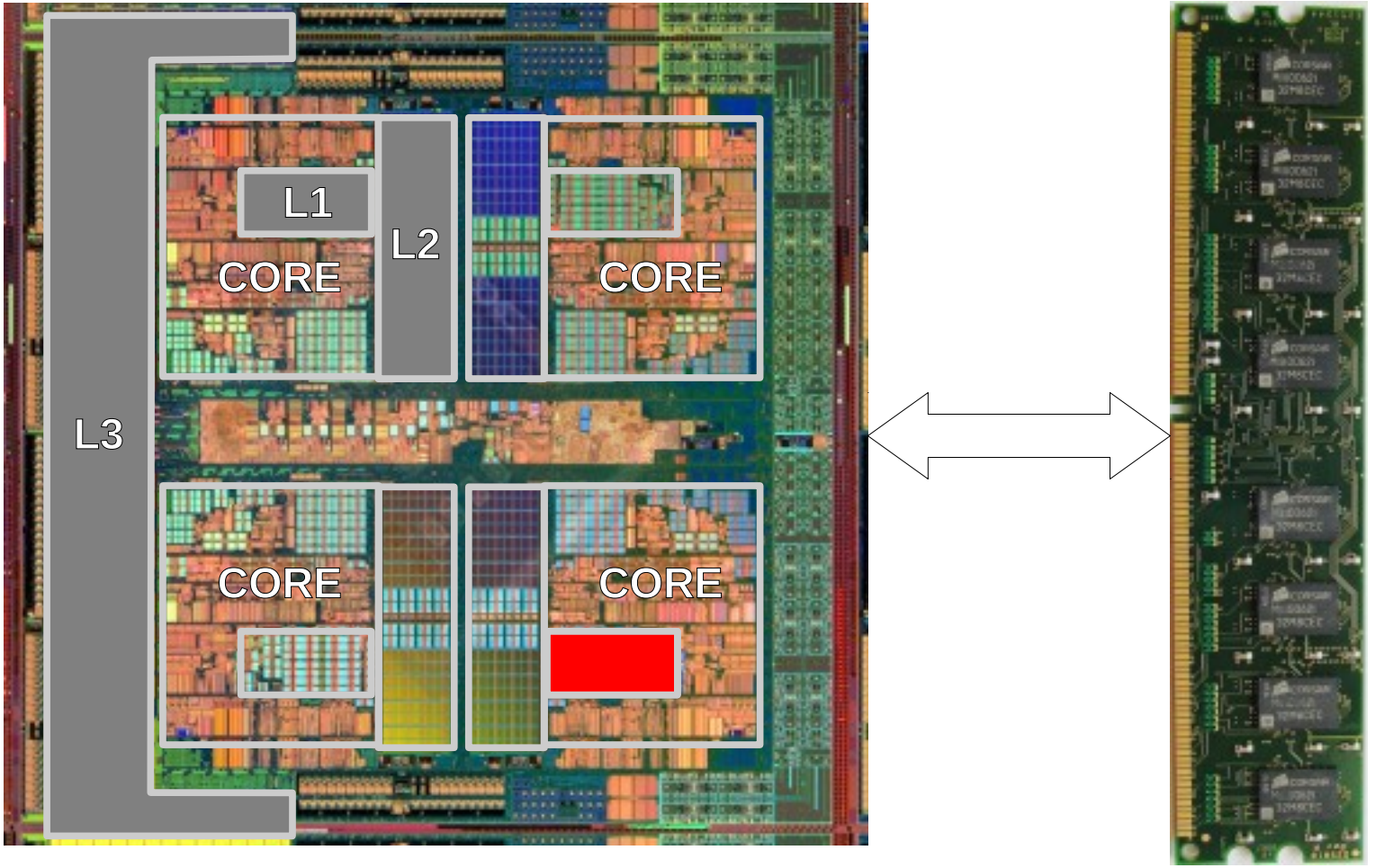
Difficult to use multicore caches efficiently



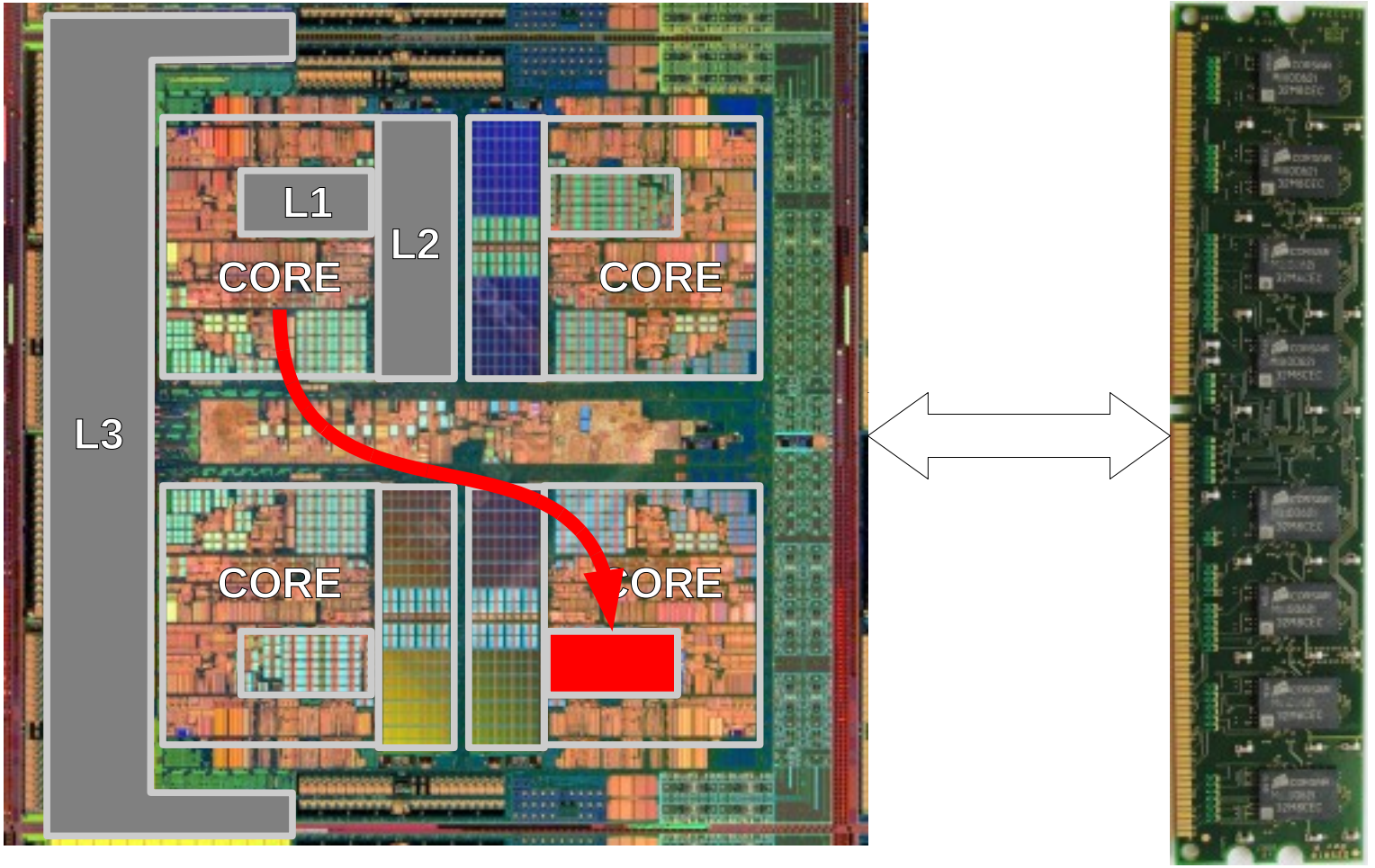
Hard to access all of on-chip cache



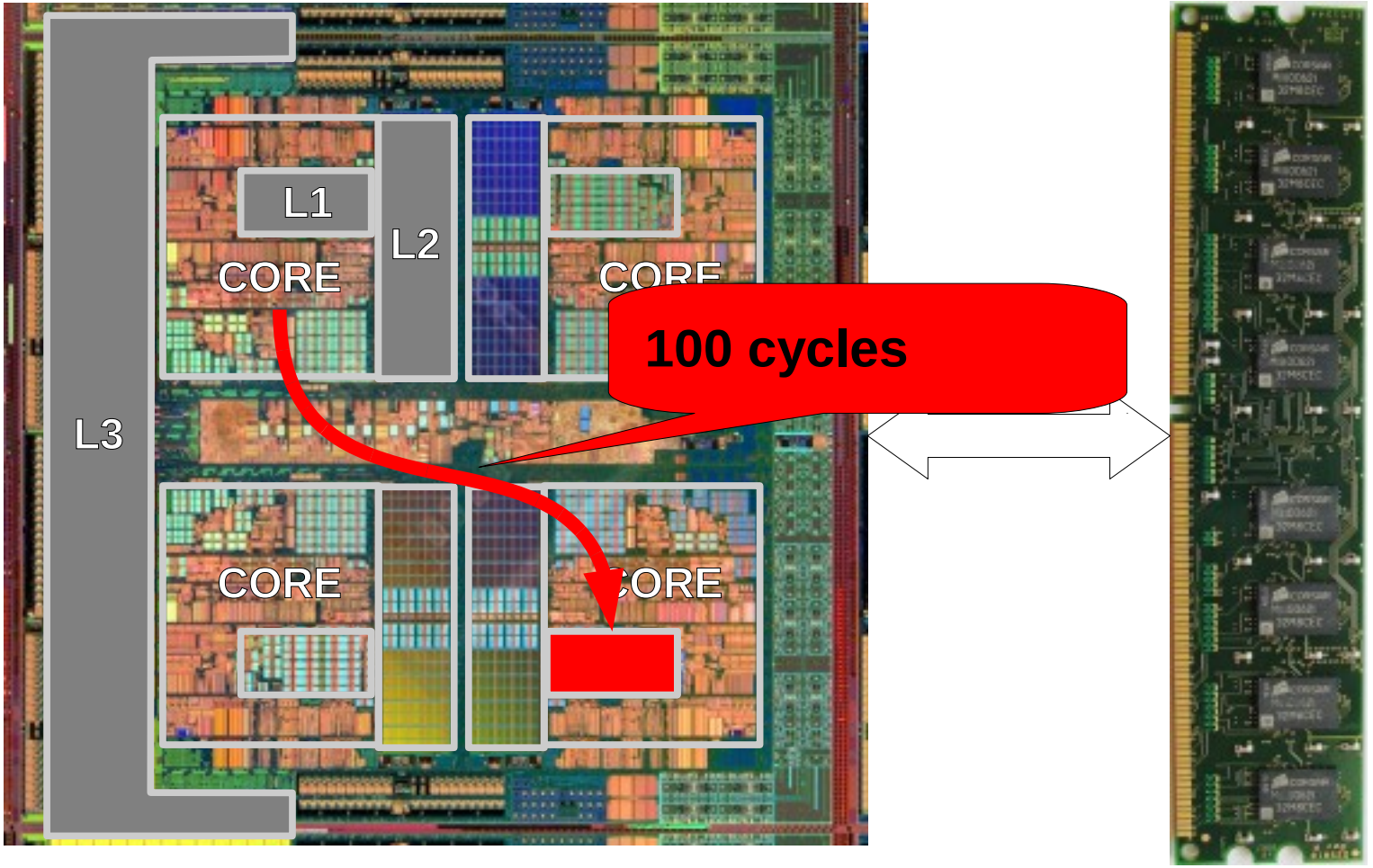
Expensive to access far away caches



Expensive to access far away caches



Expensive to access far away caches



Prototype extensions to hardware: DMC instructions

- `cpush`: store a cache line in another core's cache
- `clookup`: lookup which cache holds an address
- `cmsg`: efficient access to data in another core's cache

- Provide some of the benefits of a single fast shared cache

Prototype extensions to hardware: DMC instructions

- `cpush`: store a cache line in another core's cache
- `clookup`: lookup which cache holds an address
- `cmsg`: efficient access to data in another core's cache

- Provide some of the benefits of a single fast shared cache

Memory hierarchy

- Per-core L1 caches
- Inclusive shared L2
- MSI cache coherence protocol

`cpush`: copy cache line to another core's cache

- `cpush address, core-id`
  - Copies cache line at `address` to core with `core-id`

`cpush`: copy cache line to another core's cache

- `cpush address, core-id`
  - Copies cache line at `address` to core with `core-id`

- If `address` is marked S in source L1, copy to destination, and mark S.
- If `address` is marked M in source L1, set source copy to I, copy to destination, and mark M.
- If `address` is marked I in source L1, ignore

`cpush` example: thread migration

- To migrate thread:
  - source core: saves register values in buffer
  - source core: puts buffer on destination core's run-queue
  - destination core: restores register values to execute thread

`cpush` example: thread migration

- To migrate thread:
  - source core: saves register values in buffer
  - source core: puts buffer on destination core's run-queue
  - destination core: restores register values to execute thread

`cpush` example: thread migration

- To migrate thread:
  - source core: saves register values in buffer
  - source core: puts buffer on destination core's run-queue
  - destination core: restores register values to execute thread
- Source core's cache will hold the buffer and thread's working set

`cpush` example: thread migration

- To migrate thread:
  - source core: saves register values in buffer
  - source core: puts buffer on destination core's run-queue
  - destination core: restores register values to execute thread
- Source core's cache will hold the buffer and thread's working set
- Use `cpush` to move the buffer and thread's working set

`clookup`: lookup location of an address

- `clookup address`
  - returns the nearest core ID that caches `address`

`clookup`: lookup location of an address

- `clookup address`
  - returns the nearest core ID that caches `address`

- If `address` is M or S in source L1, return source ID
- If `address` is invalid in source L1, it's marked S or M in L2 directory, return remote ID
- If `address` in invalid in source L1, and invalid in L2 directory, return -1

`clookup` example: cache management run-times?

- Originally implemented `clookup` to help test `cmsg`
- Some software run-times try to manage cache contents.
- Maintain a map from object/address to cache

`clookup` example: cache management run-times?

- Originally implemented `clookup` to help test `cmsg`
- Some software run-times try to manage cache contents.
- Maintain a map from object/address to cache
  - Essentially tries duplicates hardware state
  - Inaccurate
  - Expensive

`clookup` example: cache management run-times?

- Originally implemented `clookup` to help test `cmsg`
- Some software run-times try to manage cache contents.
- Maintain a map from object/address to cache
  - Essentially tries duplicates hardware state
  - Inaccurate
  - Expensive
- Replace software map with `clookup`

`cmsg`: efficient access to data in another core's cache

- `cmsg address, pc, argument`
  - Looks up that core that caches `address`.
  - Interrupts the core, causing it to execute the function at `pc`, passing `argument` as an argument.

`cmsg`: efficient access to data in another core's cache

- `cmsg address, pc, argument`
  - Looks up that core that caches `address`.
  - Interrupts the core, causing it to execute the function at `pc`, passing `argument` as an argument.
- If `address` is M or S in source L1, return 0, drop message
- If `address` is I in L2, return 0, drop message
- If `address` is cached in a remote L1, return 1, send message

`cmsg`: efficient access to data in another core's cache

- `cmsg address, pc, argument`
  - Looks up that core that caches `address`.
  - Interrupts the core, causing it to execute the function at `pc`, passing `argument` as
- If `address` is M or S in source L1
  message
- If `address` is I in L2, return 0, drop message
- If `address` is cached in a remote L1, return 1, send message

Cost roughly equivalent to L2 cache miss, or ½ the cost of inter-core miss

`cmsg` example: shared data structures

- Many applications used shared data structures
- E.g. Linux uses linked lists in many subsystems

```
do_something:
spin_lock(lock);
item = list_pop(list);
update_metadata(list);
spin_unlock(lock);
```

`cmsg` example: shared data structures

- Many applications used shared data structures
- E.g. Linux uses linked lists in many subsystems

```
do_something:
spin_lock(lock);
item = list_pop(list);
update_metadata(list);
spin_unlock(lock);
```

`cmsg` example: shared data structures

- Many applications used shared data structures
- E.g. Linux uses linke bsystems

Three cache misses:
`list_entry`
`list_entry->next->prev`
`list_entry->prev->next`

```
do_something:
spin_lock(lock);
item = list_pop(list);
update_metadata(list);
spin_unlock(lock);
```

`cmsg` example: shared data structures

- Many applications used shared data structures
- E.g. Linux uses linked lists in many subsystems

```
do_something:
spin_lock(lock);
item = list_pop(list);
update_metadata(list);
spin_unlock(lock);
```

- Source calls cmsg(lock, do_something) and destination calls cmsg(sourceId) to reply
  - About the cost of one inter-core cache miss

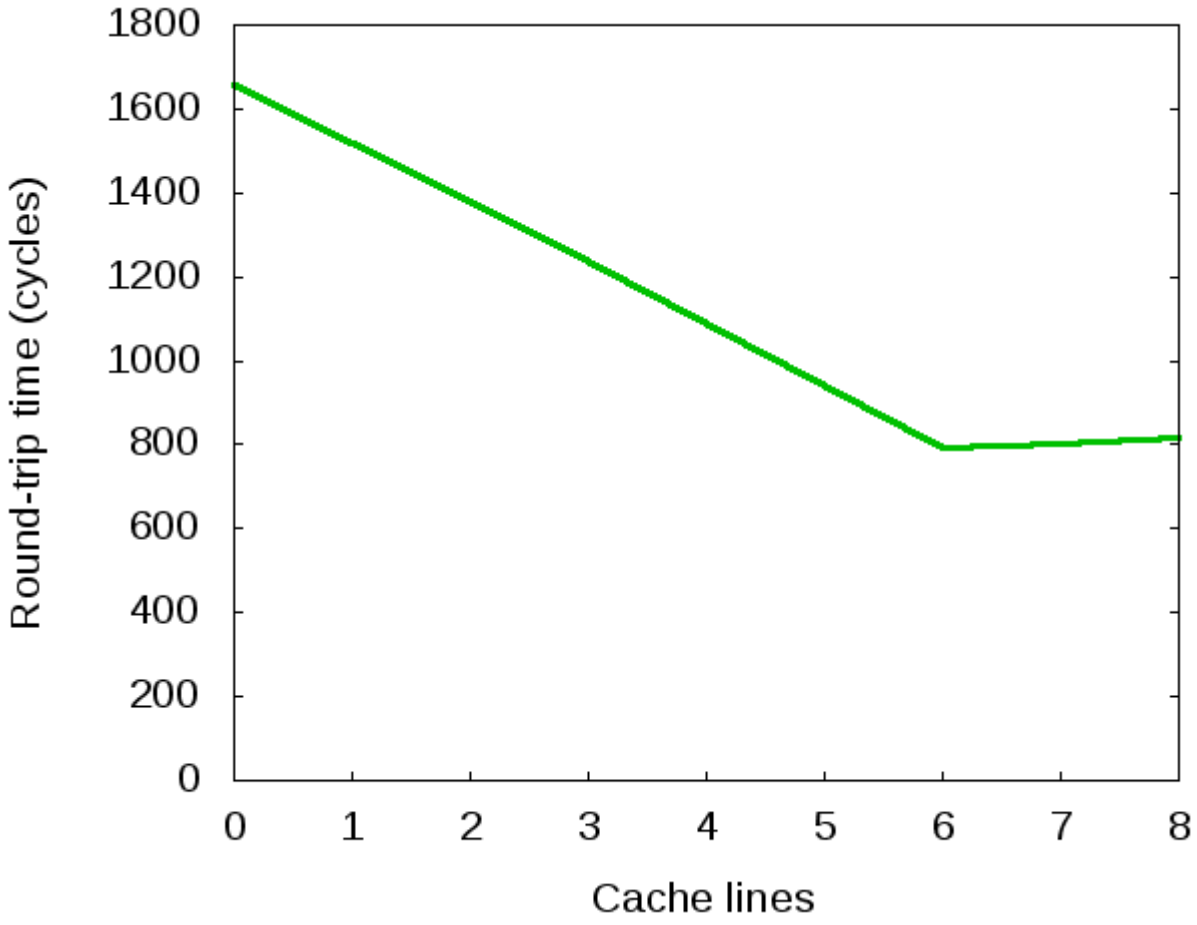
DMC implementation

- Modified an existing PowerPC implementation
  - Runs as on an FPGA as a cycle accurate simulator
  - FPGA is important
- Added/modified about 1000 lines of BSV
- Wrote a software run-time in about 2000 lines of C for testing and benchmarking

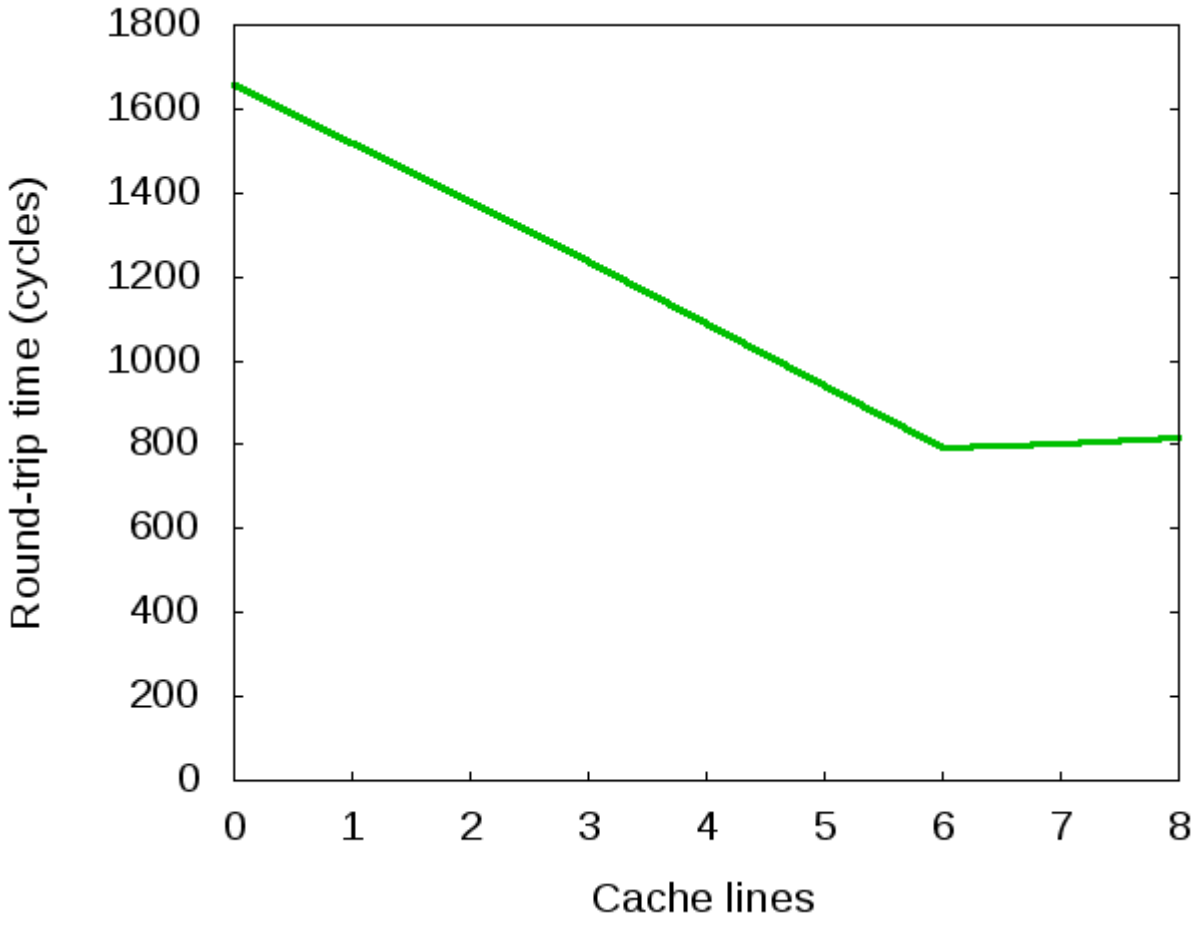
Preliminary evaluation

- Can software improve performance using DMC instructions?
  - Thread migration benchmark
  - List manipulation benchmark
- Dual core, L2 access 31 cycles, DRAM access 255 cycles
- Caveats: no hardware pre-fetching, no SMT, etc.

`cpush` improves thread migration performance

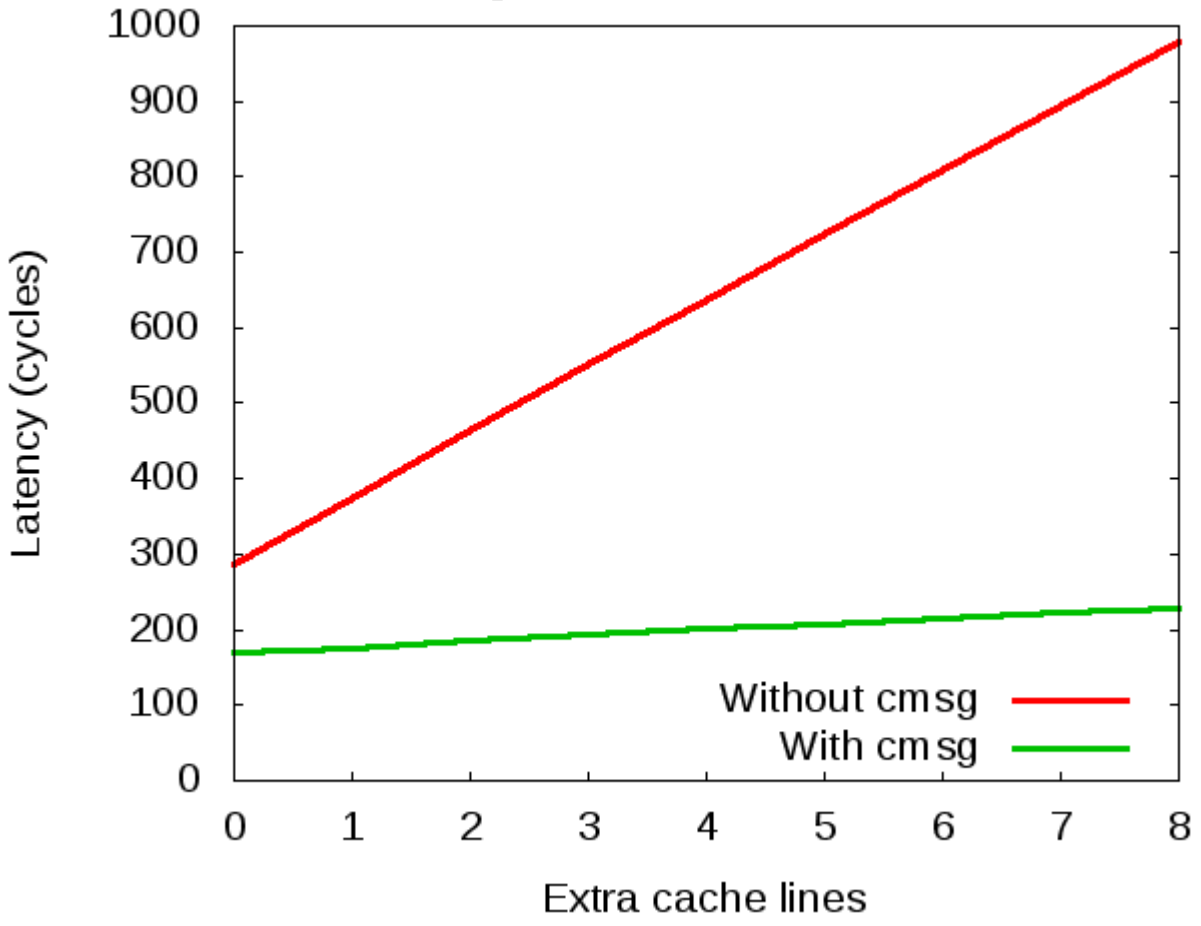


`cpush` improves thread migration performance

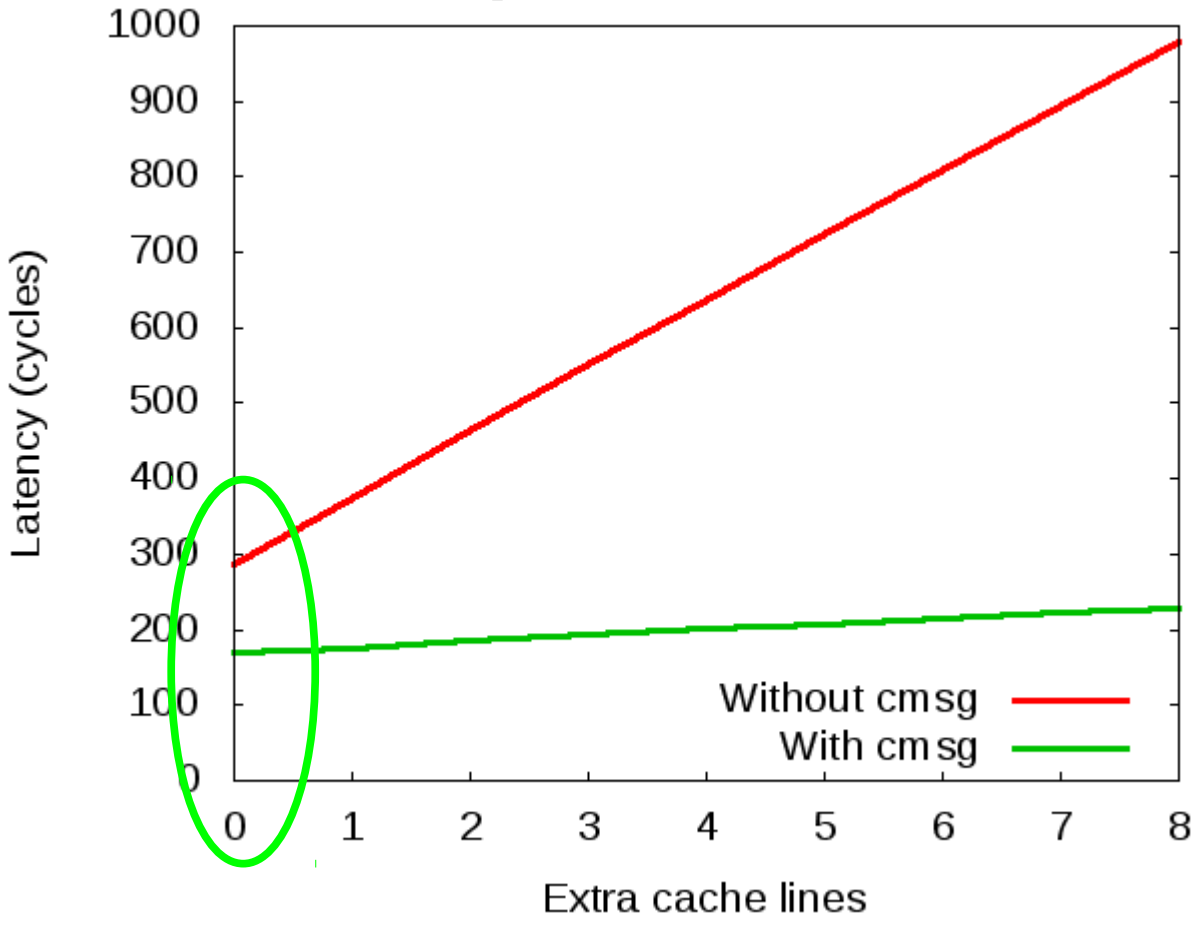


- Pushing 6 cache lines cuts latency in half
- For more the 6 the messages FIFOs fill up

`cmsg` improves the performance of list operations

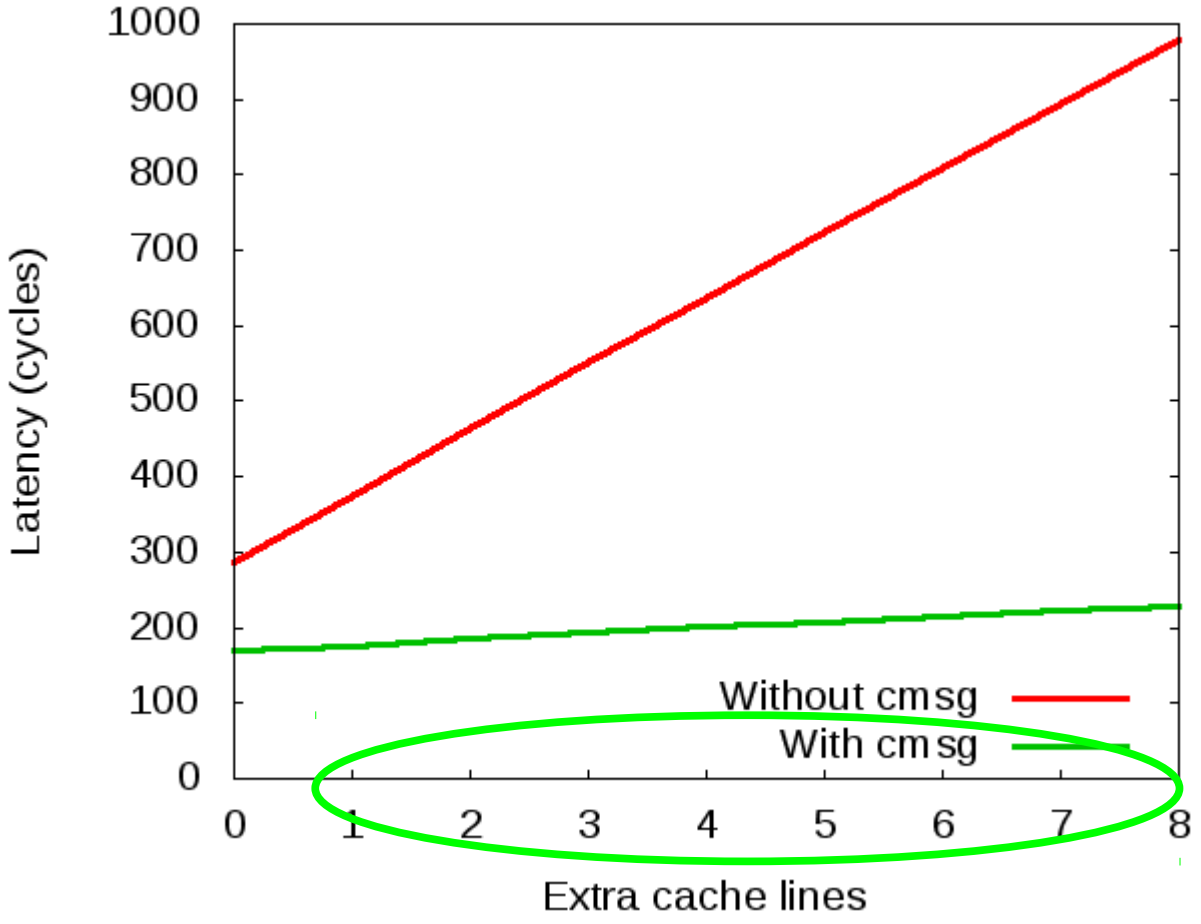


`cmsg` improves the performance of list operations



- Reduce update latency by ½

`cmsg` improves the performance of list operations



- Reduce update latency by ½
- More benefit as benchmark updates more meta-data

Future work

- `cmsg` loose ends
  - How to handle multiple address spaces?
  - How to deal with fairness?
- When shouldn't applications use DMC instructions?
- Experiment with real applications

Related work

- Managing multicore caches
- Computation migration systems

Conclusion

- Three new instructions (`cpush`, `clookup`, and `cmsg`) for managing cache contents
- Promising preliminary results
- Next step: generalize to real workloads