CS 61C: Great Ideas in Computer Architecture (Machine Structures)

Instructor:
Michael Greenbaum

New-School Machine Structures (It's a bit more complicated!)

Today's Lecture

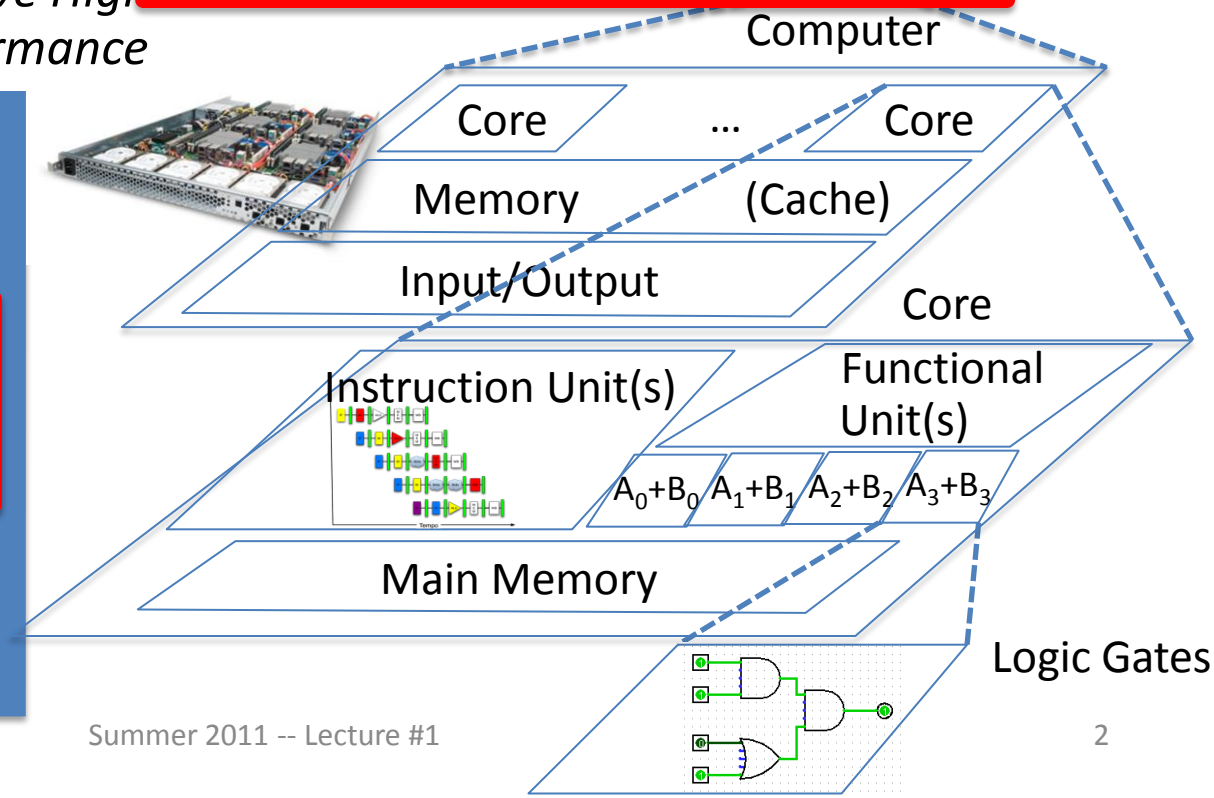
Software | *Hardware*

- Parallel Requests
  Assigned to computer
  e.g., Search "Katz"
- Parallel Threads
  Assigned to core
  e.g., Lookup, Ads
- Parallel Instructions
  >1 instruction @ one time
  e.g., 5 pipelined instructions
- Parallel Data
  >1 data item @ one time
  e.g., Add of 4 pairs of words
- Hardware descriptions
  All gates @ one time

Harness Parallelism & Achieve High Performance

Warehouse Scale Computer

Smart Phone



Review

- Cloud Computing
  - Benefits of WSC computing for all parties involved
  - “Elastic” pay as you go resource allocation
  - Amazon demonstrates that you can make money by selling cycles and storage
- Warehouse-Scale Computers (WSCs)
  - Power ultra large-scale Internet applications
  - Emphasis on cost, power efficiency
  - PUE - Ratio of total power consumed over power used for computing.

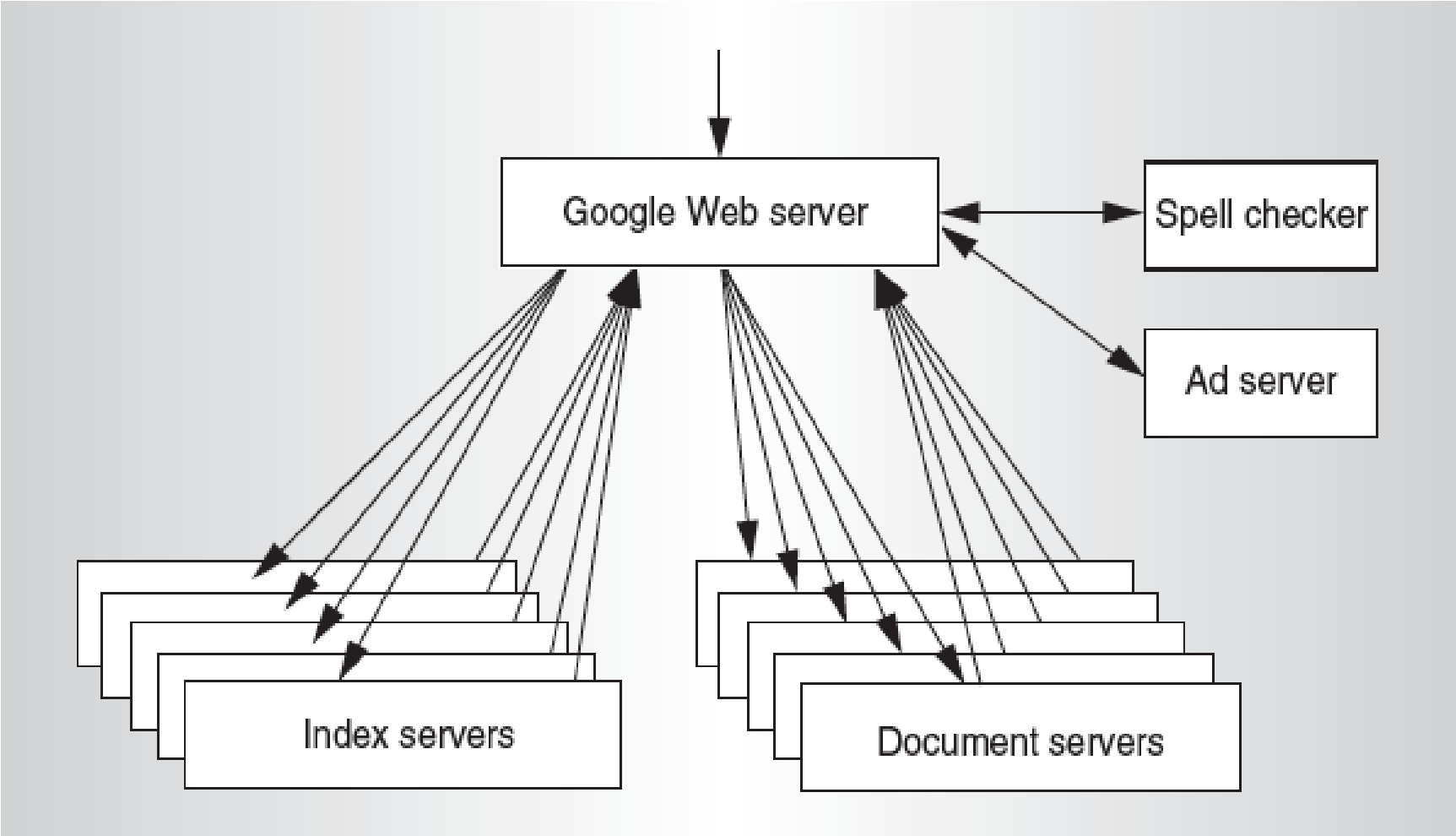
Agenda

- Request Level Parallelism
- MapReduce
- Administrivia
- MapReduce Processing
- Break
- Modern Microarchitectures

Request-Level Parallelism (RLP)

- Lots of independent tasks
  - In a WSC for a large internet app, hundreds to thousands per second.
    - Mostly involve read-only databases
    - Little read-write (aka "producer-consumer") sharing
    - Rarely involve read–write data sharing or synchronization across requests
- Computation easily partitioned within a request and across different requests

Google Query-Serving Architecture



Anatomy of a Web Search

- Google “Randy H. Katz”
  - Direct request to “closest” Google Warehouse Scale Computer
  - Front-end load balancer directs request to one of many arrays (cluster of servers) within WSC
  - Within array, select one of many Google Web Servers (GWS) to handle the request and compose the response pages
  - GWS communicates with Index Servers to find documents that contain the search words, “Randy”, “Katz”, uses location of search as well
  - Return document list with associated relevance score

Anatomy of a Web Search

- In parallel,
  - Ad system: books by Katz at Amazon.com
  - Images of Randy Katz
- Use docids (document IDs) to access indexed documents
- Compose the page
  - Result document extracts (with keyword in context) ordered by relevance score
  - Sponsored links (along the top) and advertisements (along the sides)

Google

randy katz | Search | Instant is on ▾

About 244,000 results (0.13 seconds) Advanced search

- Everything
- Images
- Videos
- News
- Shopping
- More

Berkeley, CA
Change location

All results
Sites with images

More search tools

Professor **Randy** H. **Katz**, CS Division, EECS Department, University ...
- 6 visits - 3/11/10

Aug 3, 2010 ... Professor **Randy** H. **Katz**. Electrical Engineering and Computer Science Department. The United Microelectronics Corporation Distinguished ...
bnrg.eecs.berkeley.edu/~**randy**/ - Cached - Similar

Randy H. **Katz** | EECS at UC Berkeley
Randy H. **Katz**. Professor. Research Areas. Computer Architecture ...
www.eecs.berkeley.edu/Faculty/Homepages/**katz**.html - Cached - Similar

Biographical Sketch
I was elected to the National Academy of Engineering in 2000. Last updated ...
daedalus.cs.berkeley.edu/~**randy**/biosketch.html - Cached - Similar

Randy Katz | RAD Lab
Randy Katz; Professor; randy [at] eecs [dot] berkeley [dot] edu. Address ...
radlab.cs.berkeley.edu/directory/**randy-katz** - Cached

Show more results from berkeley.edu

Images for **randy katz** - Report images

Randy Katz - Wikipedia, the free encyclopedia
Dr. **Randy** Howard **Katz** is a distinguished professor at University of California, Berkeley of the Electrical Engineering and Computer Science Department. ...
en.wikipedia.org/wiki/**Randy_Katz** - Cached - Similar

Randy Katz profiles | LinkedIn
View the profiles of professionals named **Randy Katz** on LinkedIn. There are 22 professionals named **Randy Katz**, who use LinkedIn to exchange information, ...
www.linkedin.com/pub/dir/**Randy**/**Katz** - Cached

Anatomy of a Web Search

- Implementation strategy
  - Randomly distribute the entries
  - Make many copies of data (aka “replicas”)
  - Load balance requests across replicas
- Redundant copies of indices and documents
  - Breaks up hot spots, e.g., “Justin Bieber”
  - Increases opportunities for request-level parallelism
  - Makes the system more tolerant of failures

Agenda

- Request Level Parallelism
- MapReduce
- Administrivia
- MapReduce Processing
- Break
- Modern Microarchitectures

Data-Level Parallelism (DLP)

- Lots of data that can be operated on in parallel
- We've already seen SIMD instructions as a way to exploit this.
- What about larger scale DLP? Lots of data spread across many disks? WSC level quantities of data?

Problem Trying To Solve

- How process large amounts of raw data (crawled documents, request logs, …) every day to compute derived data (inverted indicies, page popularity, …) when computation conceptually simple but input data large and distributed across 100s to 1000s of servers?
- Challenge: Distribute and load balance data, tolerate faults.
- Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Communications of the ACM*, Jan 2008.

MapReduce Solution

- Reduce the complexity of an otherwise complex parallel programming task by providing a rigid structure.
- Create a robust implementation
  - Fault tolerant, handles all the pesky details
- To solve problem, simply formulate it in the MapReduce format
  - Surprisingly versatile, can also chain together multiple MapReduce tasks.

Inspiration: cs61a map/reduce

- `(define (square x) (* x x))`
  `(reduce + (map square '(1 2 3 4)))`
- ***Map*** - Applies the operation "square" to every element in the list (result: '(1 4 9 16))
- ***Reduce*** - Combines the resulting data using the provided operator + (1 + 4 + 9 + 16 => 30).

MapReduce Format

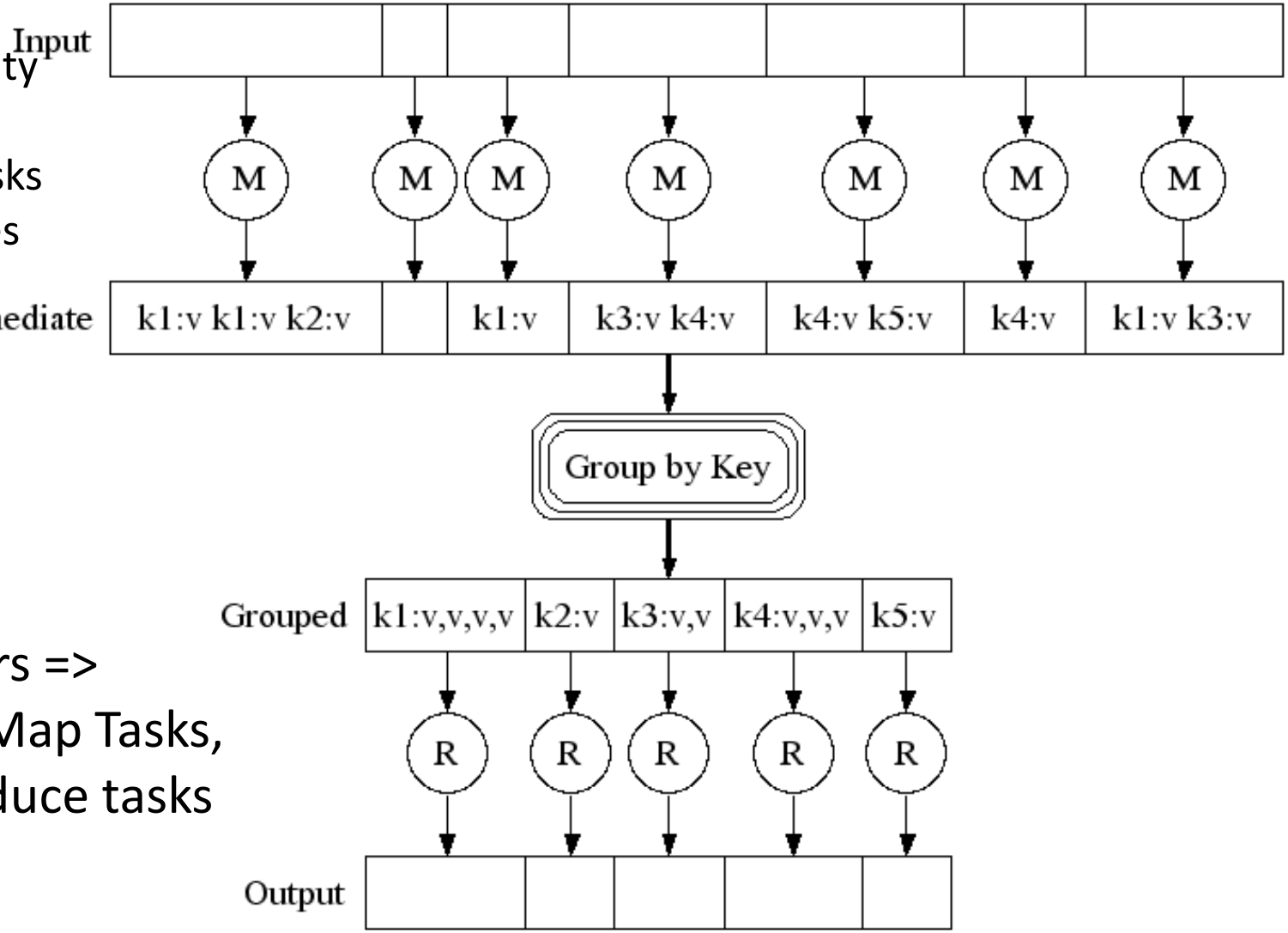
- Apply Map function to user supplied record of key/value pairs
- Compute set of intermediate key/value pairs
- Apply Reduce operation to all values that share same key in order to combine derived data properly
- User supplies Map and Reduce operations

Data-Parallel "Divide and Conquer" (MapReduce Processing)

- **Map:**
  - **Map: (K,V) -> list((K1,V1), (K2,V2), … , (KnVn))**
    - Processes input key/value pair
    - Produces set of intermediate key/value pairs
  - Slice data into "shards" or "splits", distribute these to workers, compute sub-problem solutions
- **Reduce:**
  - **Reduce: (K, list(V1, V2, …, Vn)) -> (K, combinedValue)**
    - Combines all intermediate values for a particular key
    - Produces a set of merged output values (usually just one)
  - Collect and combine sub-problem solutions

MapReduce Execution

Fine granularity tasks: many more map tasks than machines



2000 servers =>
≈ 200,000 Map Tasks,
≈ 5,000 Reduce tasks

MapReduce Popularity at Google

	Aug-04	Mar-06	Sep-07	Sep-09
Number of MapReduce jobs	29,000	171,000	2,217,000	3,467,000
Average completion time (secs)	634	874	395	475
Server years used	217	2,002	11,081	25,562
Input data read (TB)	3,288	52,254	403,152	544,130
Intermediate data (TB)	758	6,743	34,774	90,120
Output data written (TB)	193	2,970	14,018	57,520
Average number servers /job	157	268	394	488

Google Uses MapReduce For …

- Extracting the set of outgoing links from a collection of HTML documents and aggregating by target document
- Stitching together overlapping satellite images to remove seams and to select high-quality imagery for Google Earth
- Generating a collection of inverted index files using a compression scheme tuned for efficient support of Google search queries
- Processing all road segments in the world and rendering map tile images that display these segments for Google Maps
- Fault-tolerant parallel execution of programs written in higher-level languages across collections of input data
- More than 10,000 MR programs at Google in 4 years

What if Ran Google Workload on EC2?

	Aug-04	Mar-06	Sep-07	Sep-09
Number of MapReduce jobs	29,000	171,000	2,217,000	3,467,000
Average completion time (secs)	634	874	395	475
Server years used	217	2,002	11,081	25,562
Input data read (TB)	3,288	52,254	403,152	544,130
Intermediate data (TB)	758	6,743	34,774	90,120
Output data written (TB)	193	2,970	14,018	57,520
Average number of servers per job	157	268	394	488
Average Cost/job EC2	$17	$39	$26	$38
Annual Cost if on EC2	$0.5M	$6.7M	$57.4M	$133.1M

Agenda

- Request Level Parallelism
- MapReduce
- Administrivia
- MapReduce Processing
- Break
- Modern Microarchitectures

Administrivia

- Final Review - Today, **4pm-6pm, 277 Cory.**
- HKN Course Surveys administered during lecture tomorrow.
- Project 3 Face-to-Face grading, Wednesday, 8/10 in 200 SD lab.
- Final Exam - Thursday, 8/11, 9am - 12pm 2050 VLSB
  - Two-sided handwritten cheat sheet. Green sheet provided.
    - Use the back side of your midterm cheat sheet!
  - We'll be holding some extra OH Tuesday/Wednesday, we'll keep you posted.

CS61c in… Social Media

- "I've noticed a pattern. Knowledge of what is generally considered "low-level" programming is waning."
- Cites important topics that programmers should know, among them: Floating Point, Fixed Point, Cache behavior, Bitwise operators, Branch Prediction.
- Inspired by this, I'll talk about Fixed Point a bit tomorrow.
- Criticism: Also important to know when low level programming tricks are relevant. Not every task requires a performance programmer.

http://altdevblogaday.com/2011/08/06/demise-low-level-programmer/

Agenda

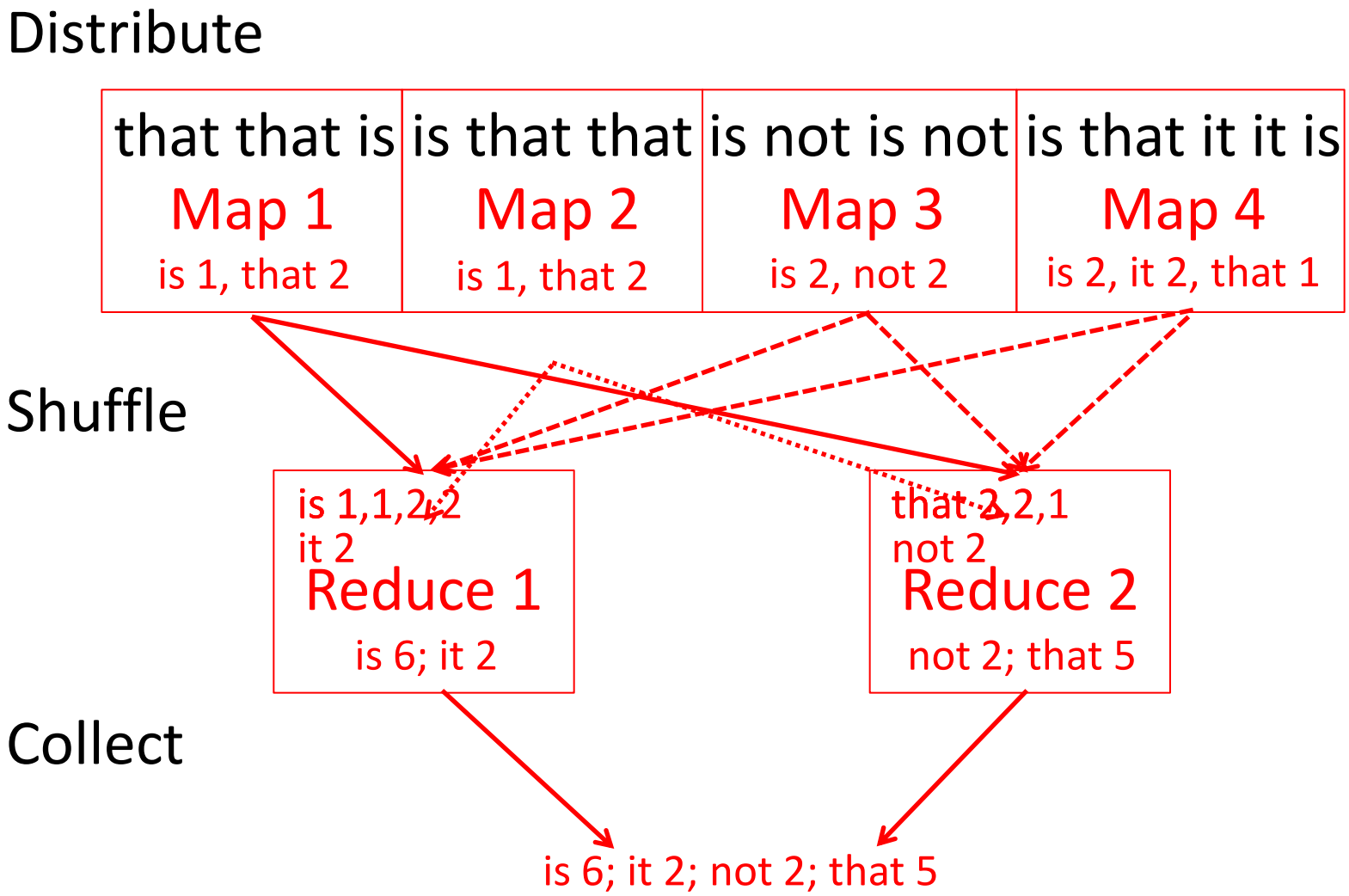
- Request Level Parallelism
- MapReduce
- Administrivia
- MapReduce Processing
- Break
- Modern Microarchitectures

MapReduce Processing Example: Count Word Occurrences

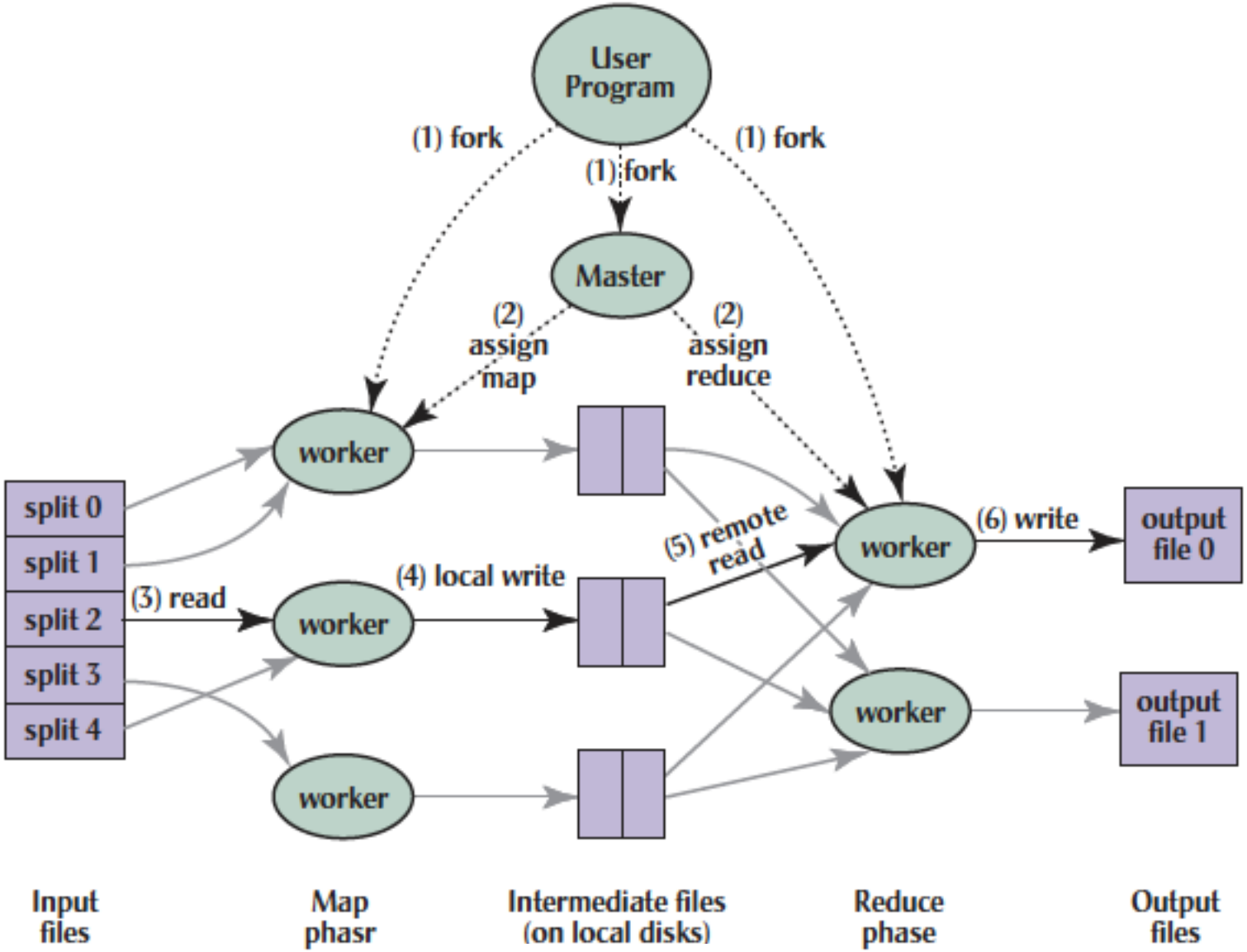
```
map(String input_key, String input_value):
  // input_key: document name
  // input_value: document contents
  for each word w in input_value:
    EmitIntermediate(w, "1"); // Produce count of words

reduce(String output_key, Iterator intermediate_values):
  // output_key: a word
  // output_values: a list of counts
  int result = 0;
  for each v in intermediate_values:
    result += ParseInt(v); // get integer from key-value
  Emit(AsString(result));
```

Example: Count Word Occurrences



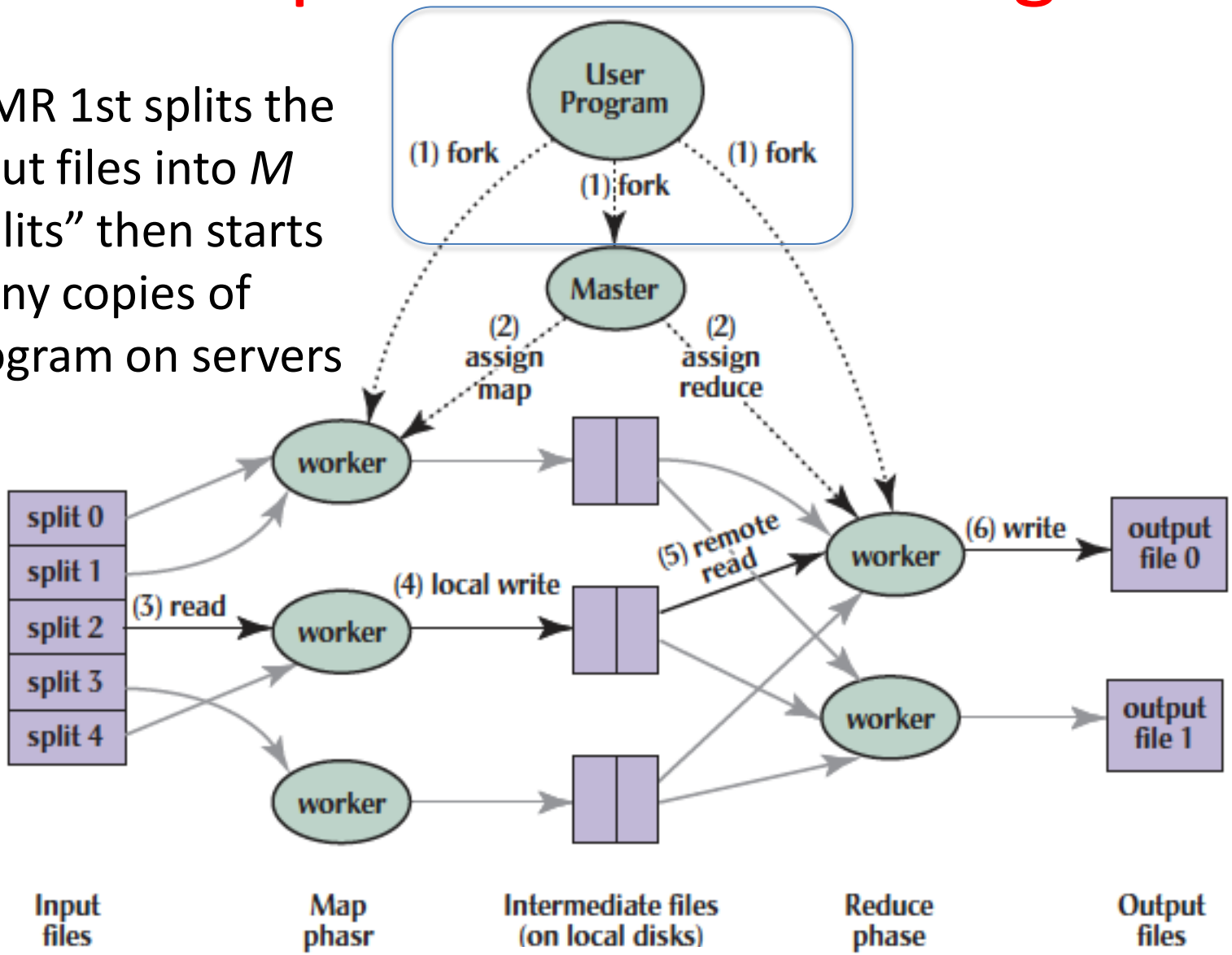
MapReduce Processing



Shuffle phase

MapReduce Processing

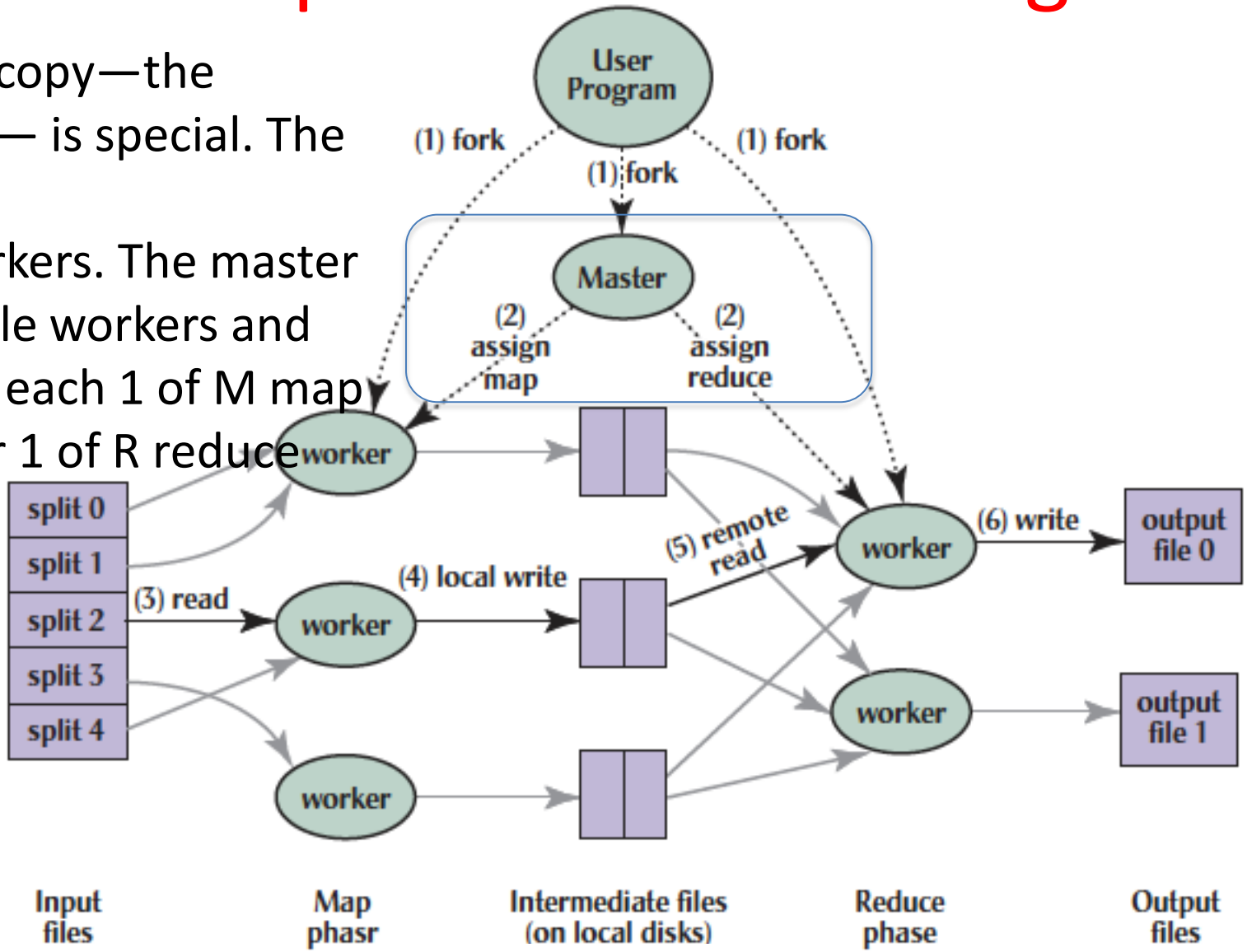
1. MR 1st splits the input files into *M* "splits" then starts many copies of program on servers



Shuffle phase

MapReduce Processing

2. One copy—the master— is special. The rest
are workers. The master picks idle workers and assigns each 1 of M map tasks or 1 of R reduce tasks.

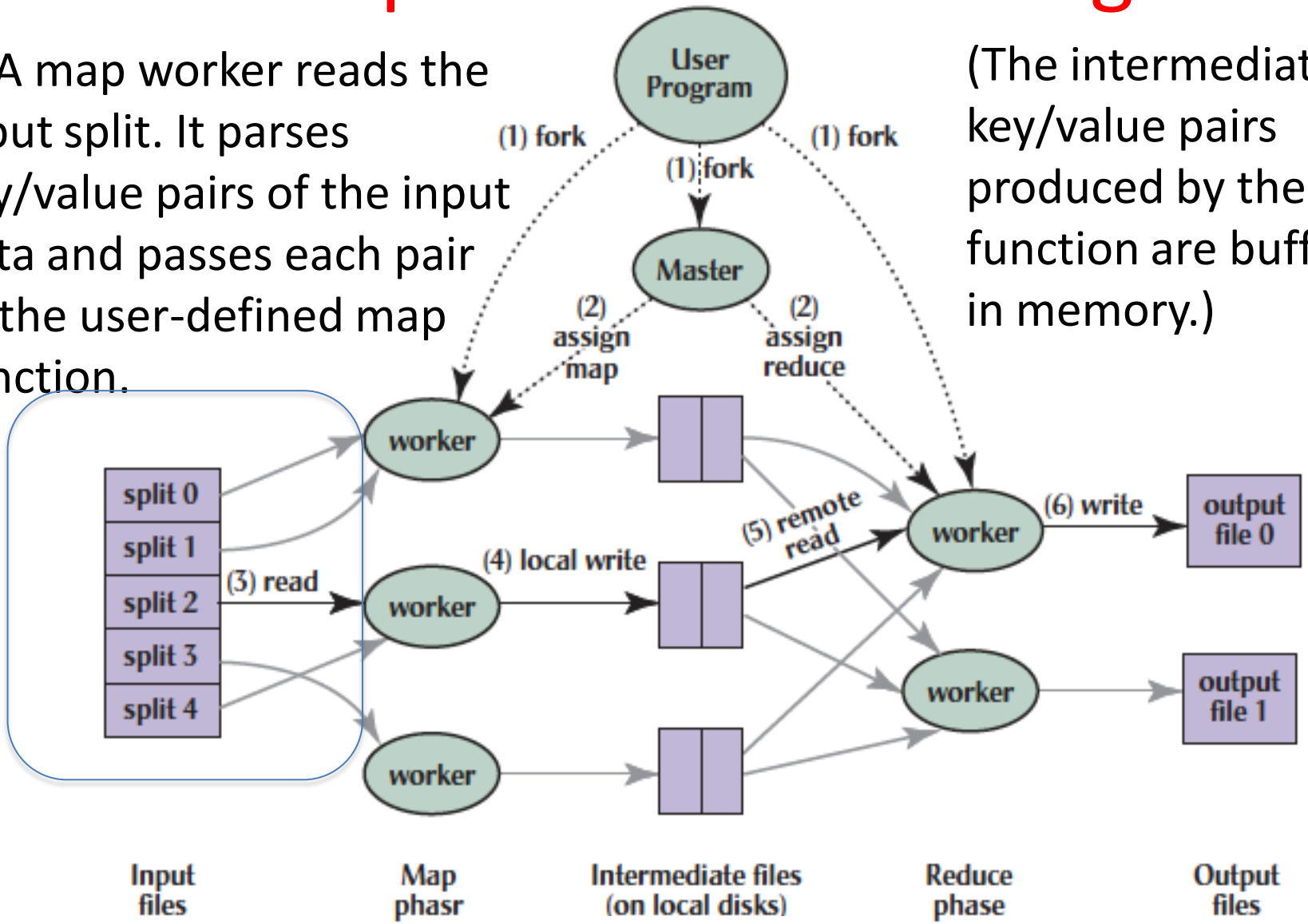


Shuffle phase

MapReduce Processing

3. A map worker reads the input split. It parses key/value pairs of the input data and passes each pair to the user-defined map function.

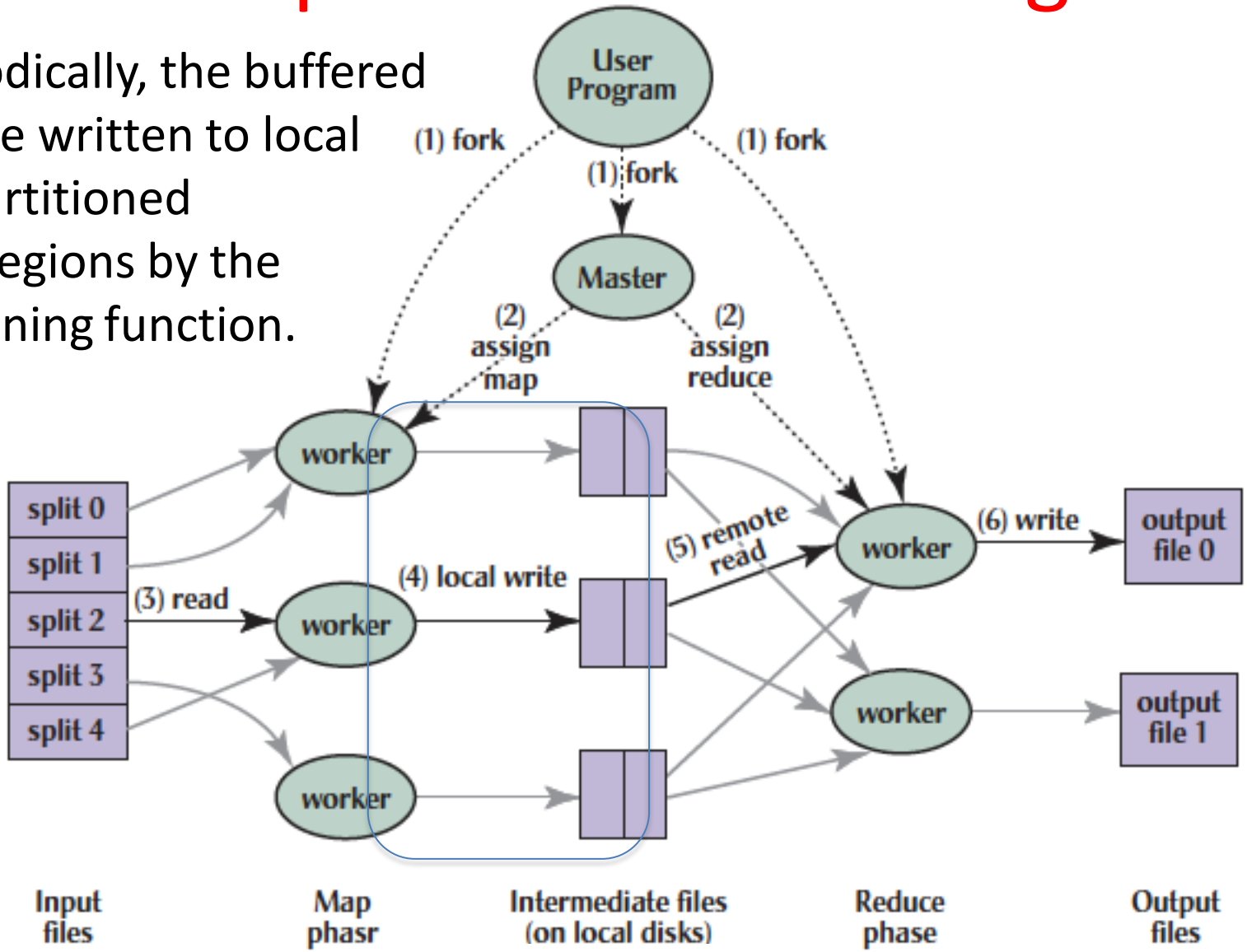
(The intermediate key/value pairs produced by the map function are buffered in memory.)



Shuffle phase

MapReduce Processing

4. Periodically, the buffered pairs are written to local disk, partitioned into *R* regions by the partitioning function.

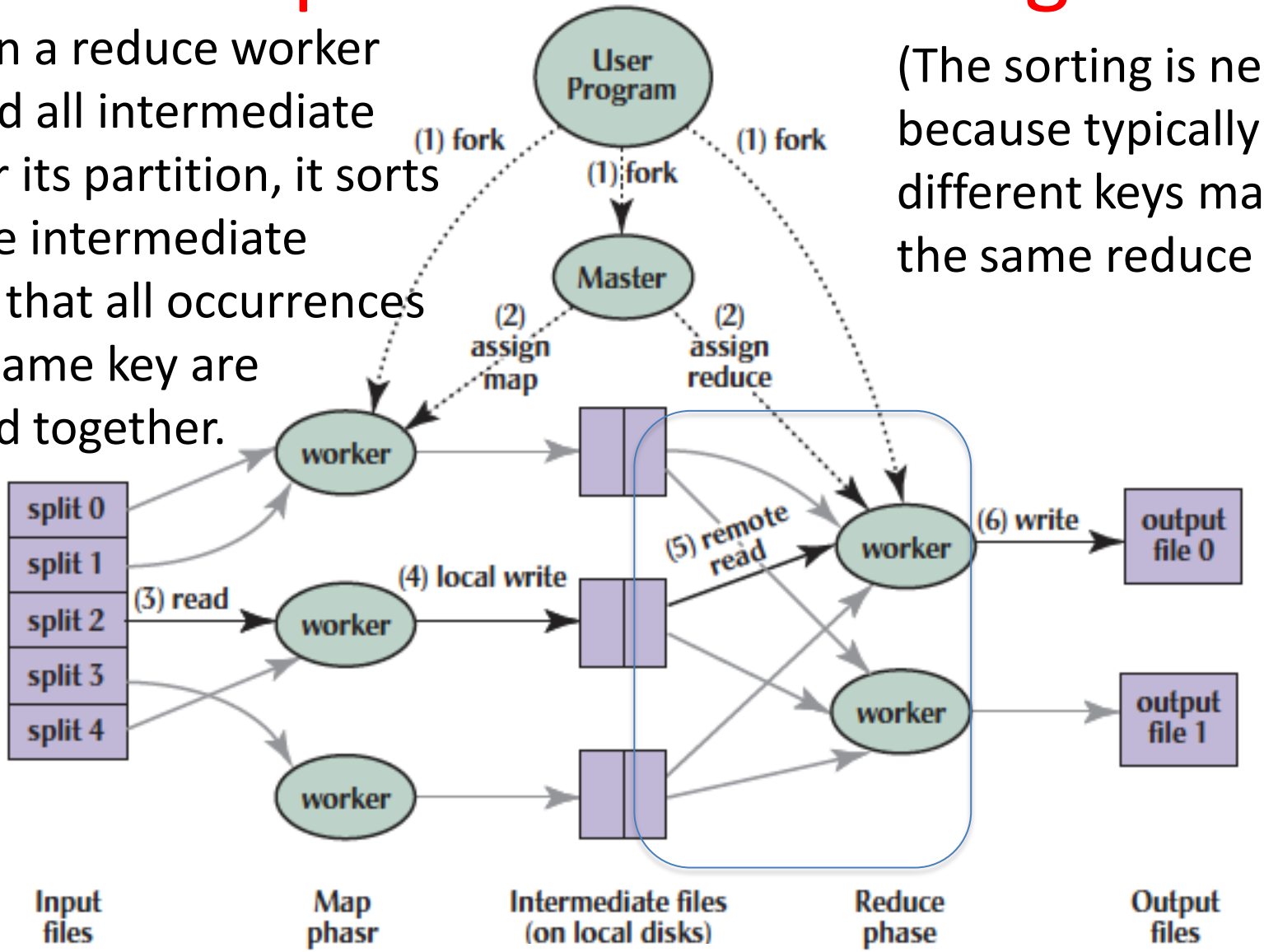


Shuffle phase

MapReduce Processing

5. When a reduce worker has read all intermediate data for its partition, it sorts it by the intermediate keys so that all occurrences of the same key are grouped together.

(The sorting is needed because typically many different keys map to the same reduce task)

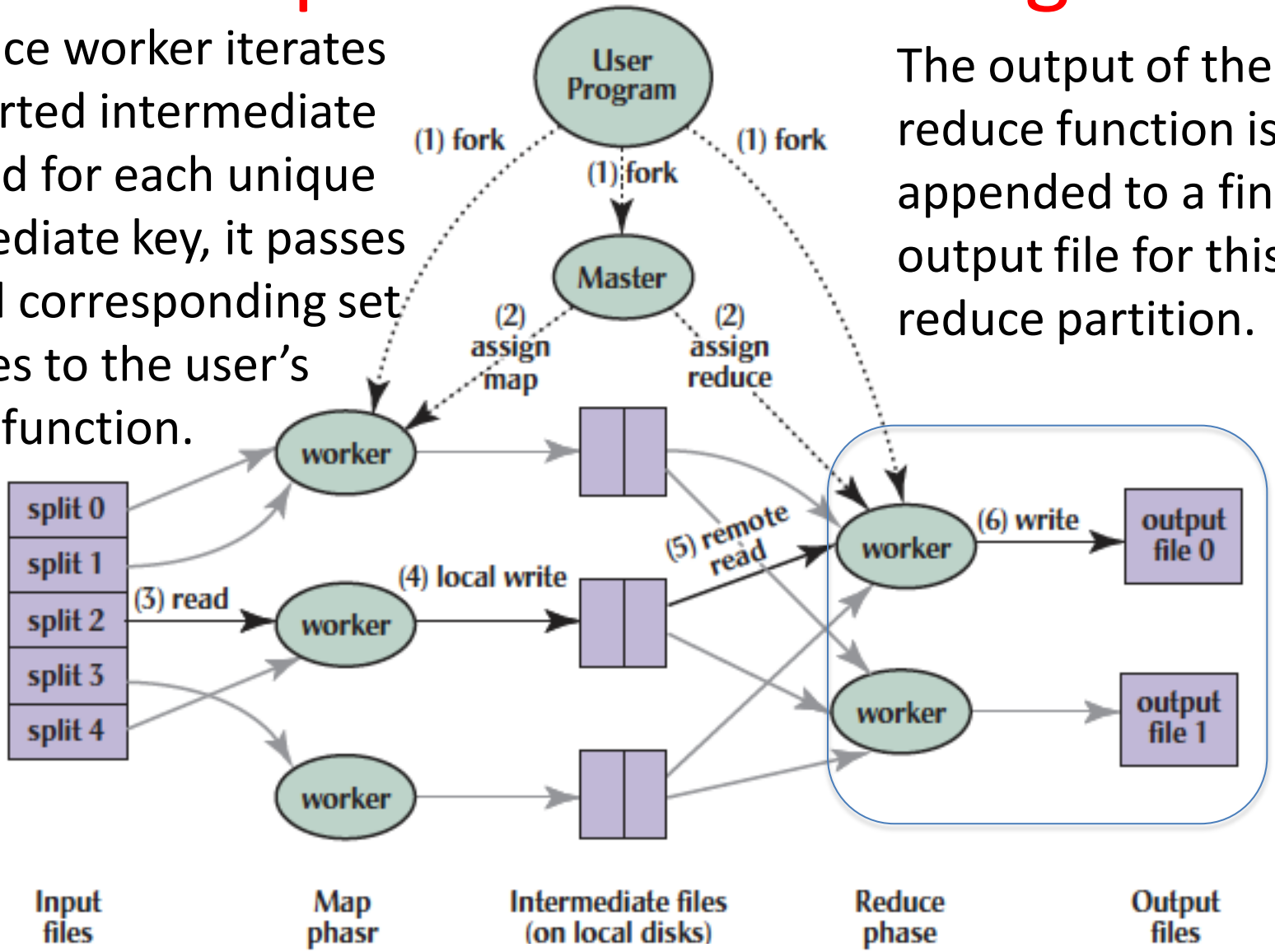


Shuffle phase

MapReduce Processing

6. Reduce worker iterates over sorted intermediate data and for each unique intermediate key, it passes key and corresponding set of values to the user's reduce function.

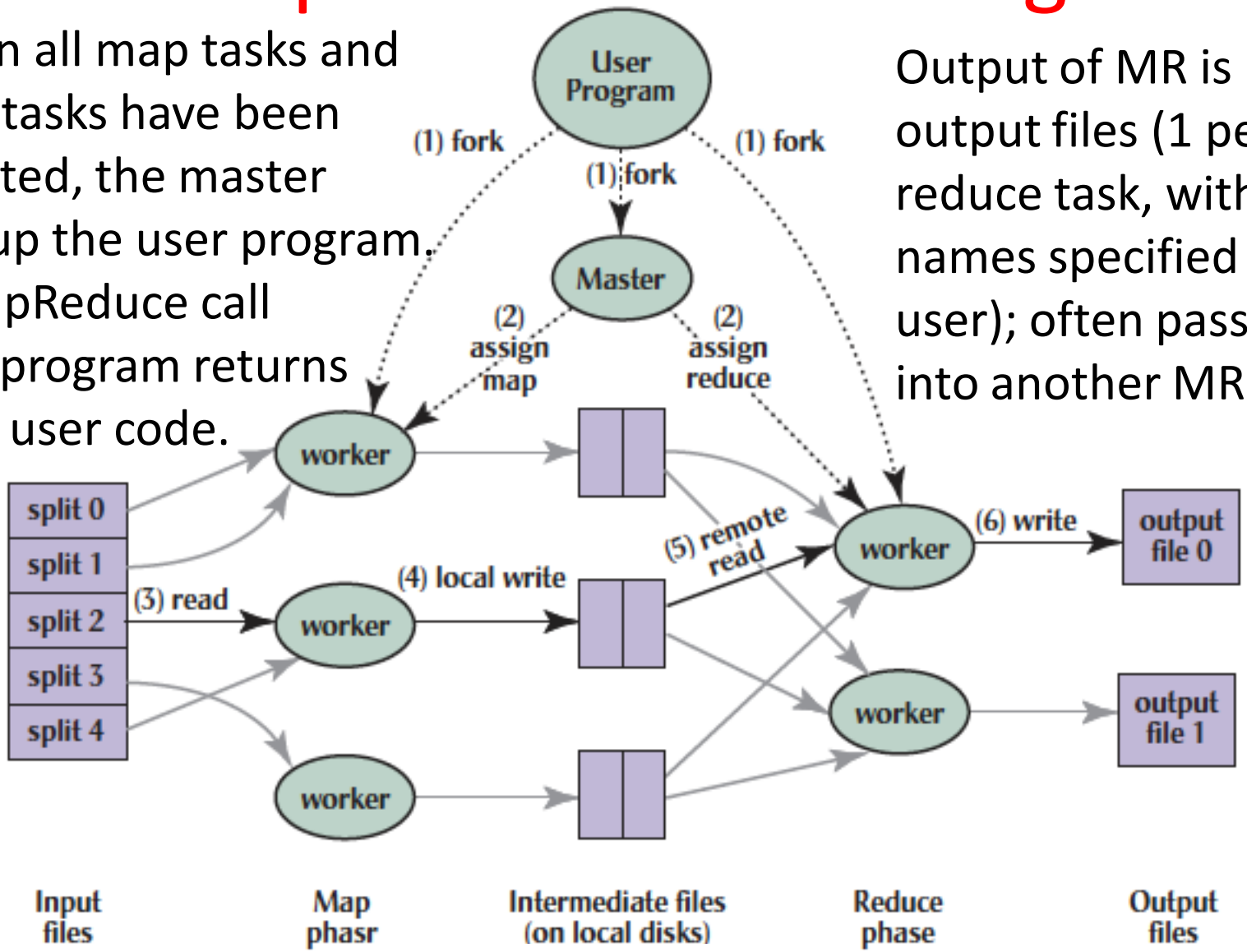
The output of the reduce function is appended to a final output file for this reduce partition.



Shuffle phase

MapReduce Processing

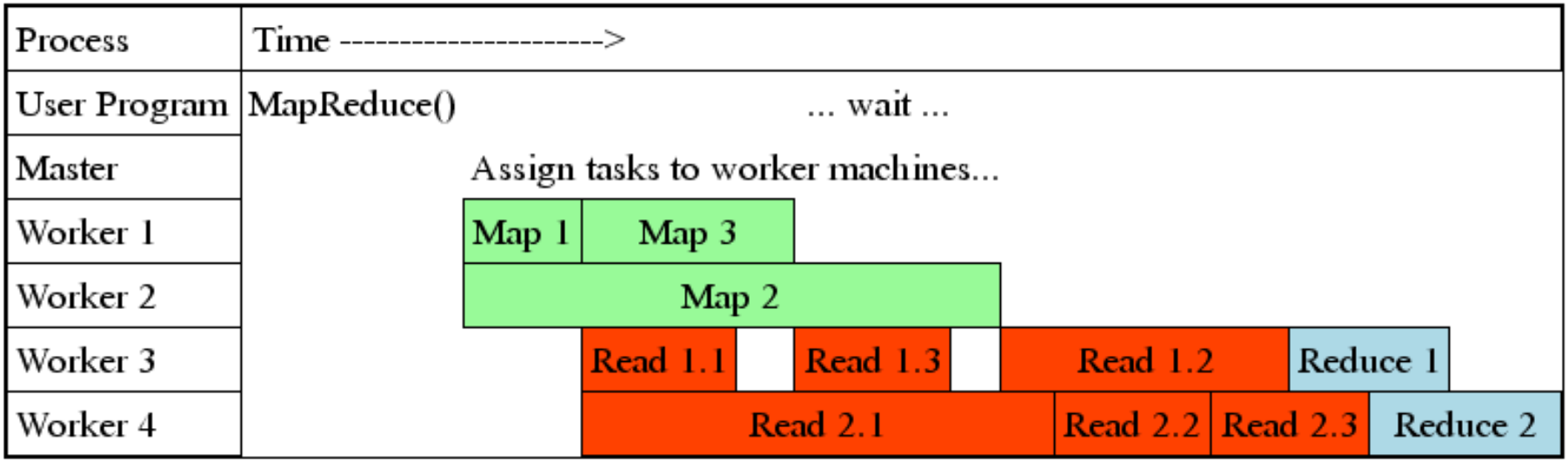
7. When all map tasks and reduce tasks have been completed, the master wakes up the user program. The MapReduce call In user program returns back to user code.



Output of MR is in *R* output files (1 per reduce task, with file names specified by user); often passed into another MR job.

Shuffle phase

MapReduce Processing Time Line



- Master assigns map + reduce tasks to "worker" servers
- As soon as a map task finishes, worker server can be assigned a new map or reduce task
- Data shuffle begins as soon as a given Map finishes
- Reduce task begins as soon as all data shuffles finish
- To tolerate faults, reassign task if a worker server "dies"

Example MapReduce Job Running

- ~41 minutes total
  - ~29 minutes for Map tasks & Shuffle tasks
  - ~12 minutes for Reduce tasks
  - 1707 worker servers used
- Map (Green) tasks read 0.8 TB, write 0.5 TB
- Shuffle (Red) tasks read 0.5 TB, write 0.5 TB
- Reduce (Blue) tasks read 0.5 TB, write 0.5 TB

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

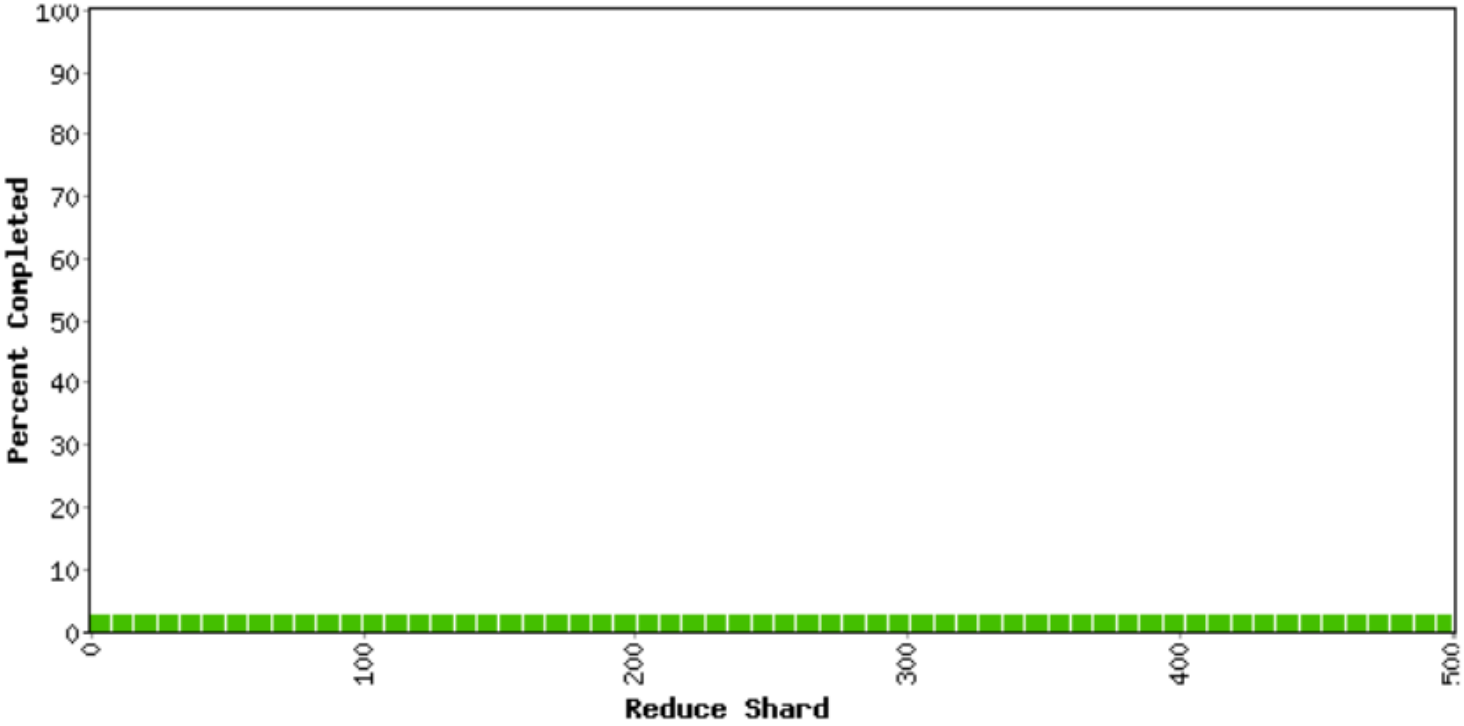
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 00 min 18 sec

323 workers; 0 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	0	323	878934.6	1314.4	717.0
Shuffle	500	0	323	717.0	0.0	0.0
Reduce	500	0	0	0.0	0.0	0.0

Counters

Variable	
Mapped (MB/s)	72.5
Shuffle (MB/s)	0.0
Output (MB/s)	0.0
doc-index-hits	145825686
docs-indexed	506631
dups-in-index-merge	0
mr-operator-calls	508192
mr-operator-	506631



MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

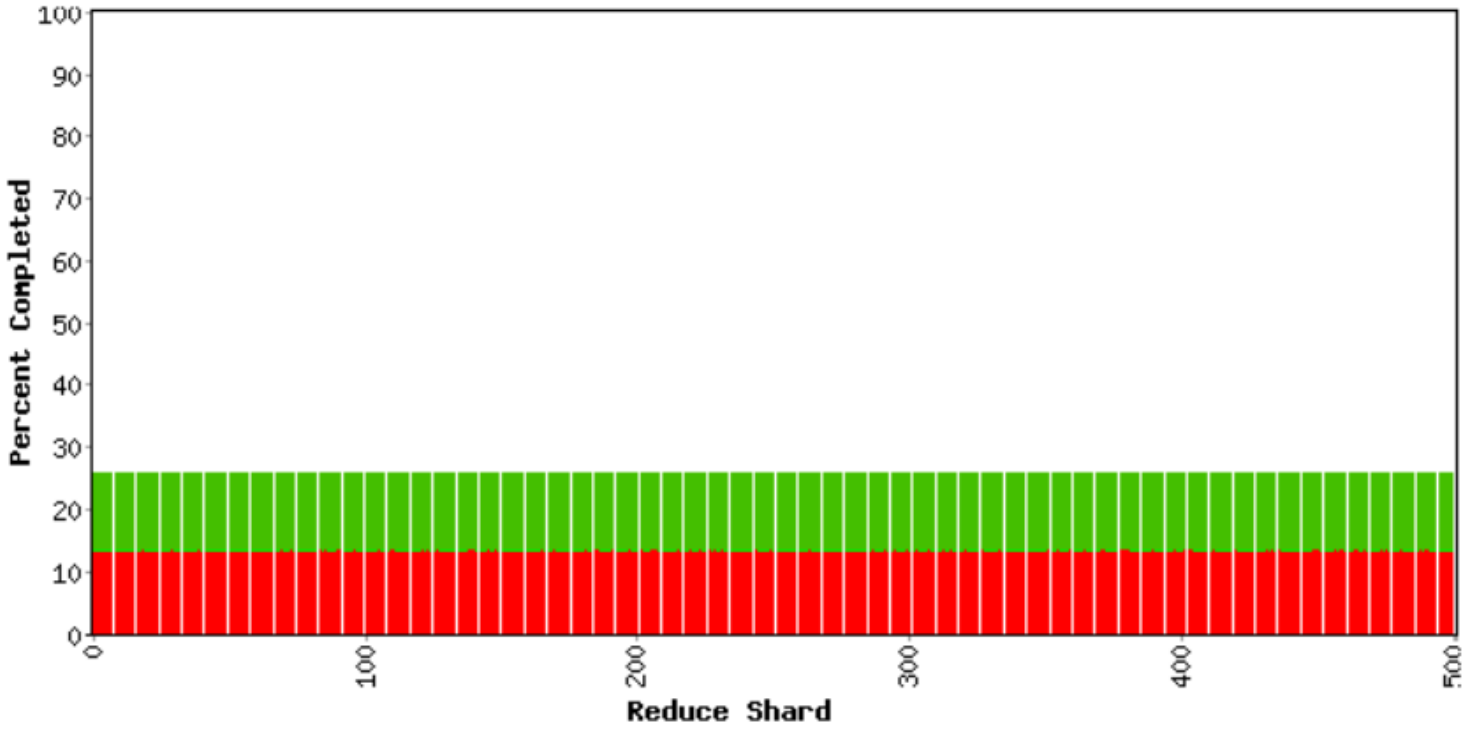
Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 05 min 07 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	1857	1707	878934.6	191995.8	113936.6
Shuffle	500	0	500	113936.6	57113.7	57113.7
Reduce	500	0	0	57113.7	0.0	0.0

Counters

Variable	
Mapped (MB/s)	699.1
Shuffle (MB/s)	349.5
Output (MB/s)	0.0
doc-index-hits	5004411944
docs-indexed	17290135
dups-in-index-merge	0
mr-operator-calls	17331371
mr-operator-outputs	17290135

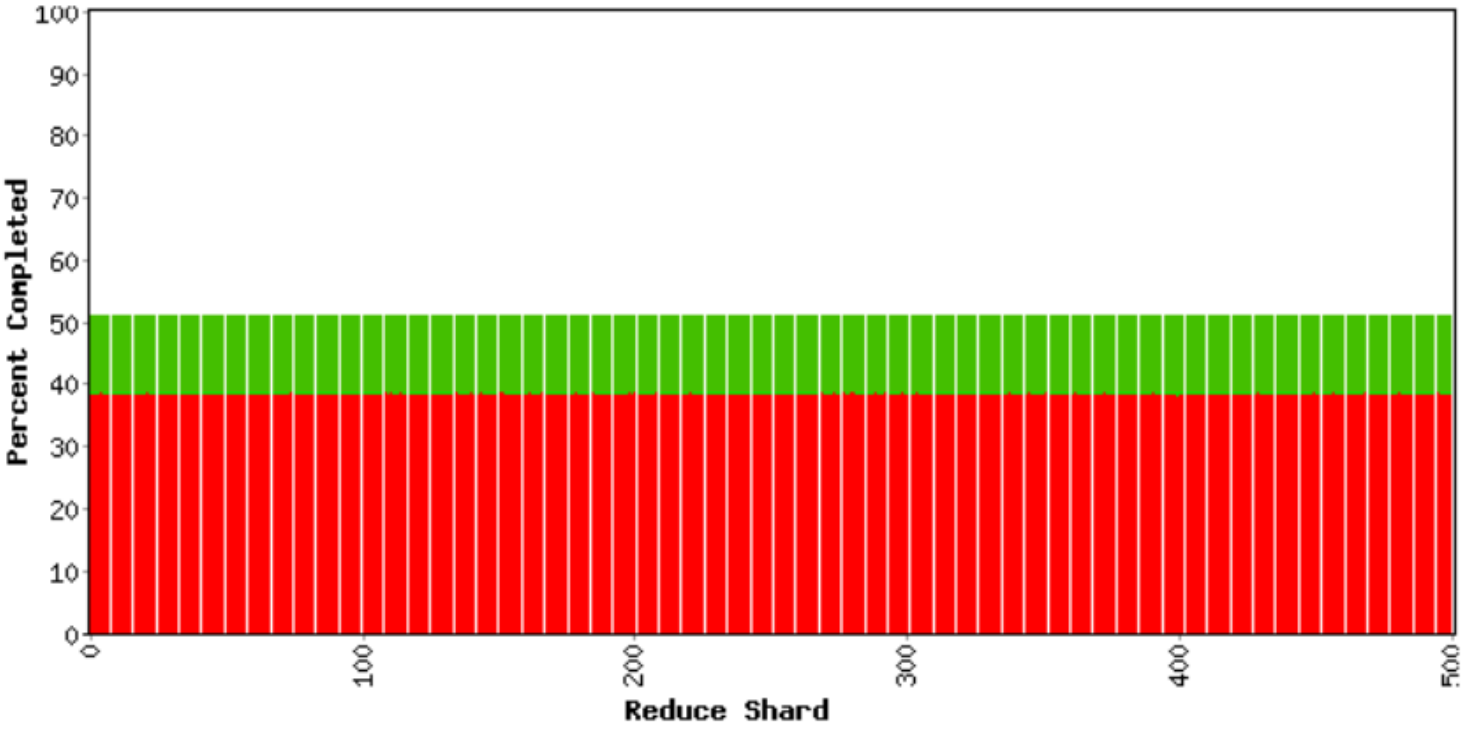


MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 10 min 18 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	5354	1707	878934.6	406020.1	241058.2
Shuffle	500	0	500	241058.2	196362.5	196362.5
Reduce	500	0	0	196362.5	0.0	0.0



Counters

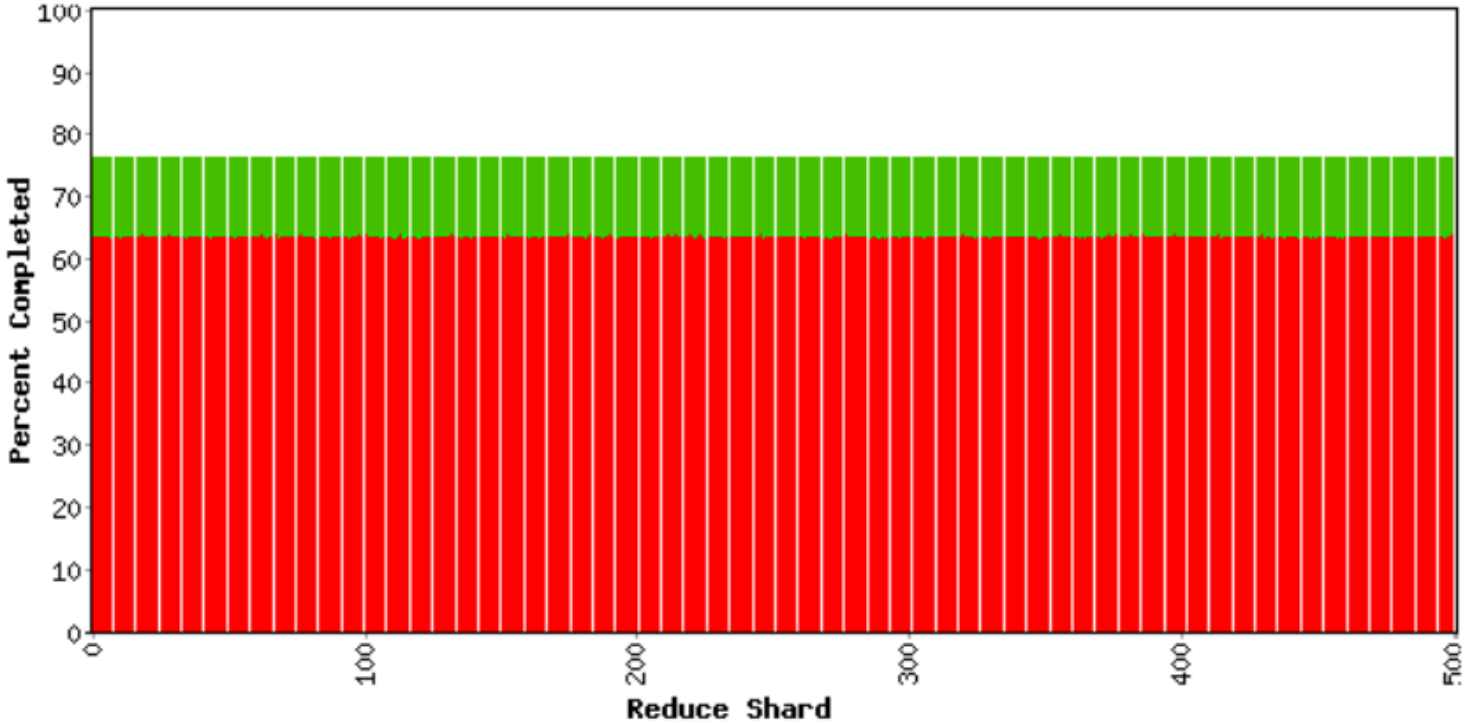
Variable	
Mapped (MB/s)	704.4
Shuffle (MB/s)	371.9
Output (MB/s)	0.0
doc-index-hits	5000364228
docs-indexed	17300709
dups-in-index-merge	0
mr-operator-calls	17342493
mr-operator-outputs	17300709

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 15 min 31 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	8841	1707	878934.6	621608.5	369459.8
Shuffle	500	0	500	369459.8	326986.8	326986.8
Reduce	500	0	0	326986.8	0.0	0.0



Counters

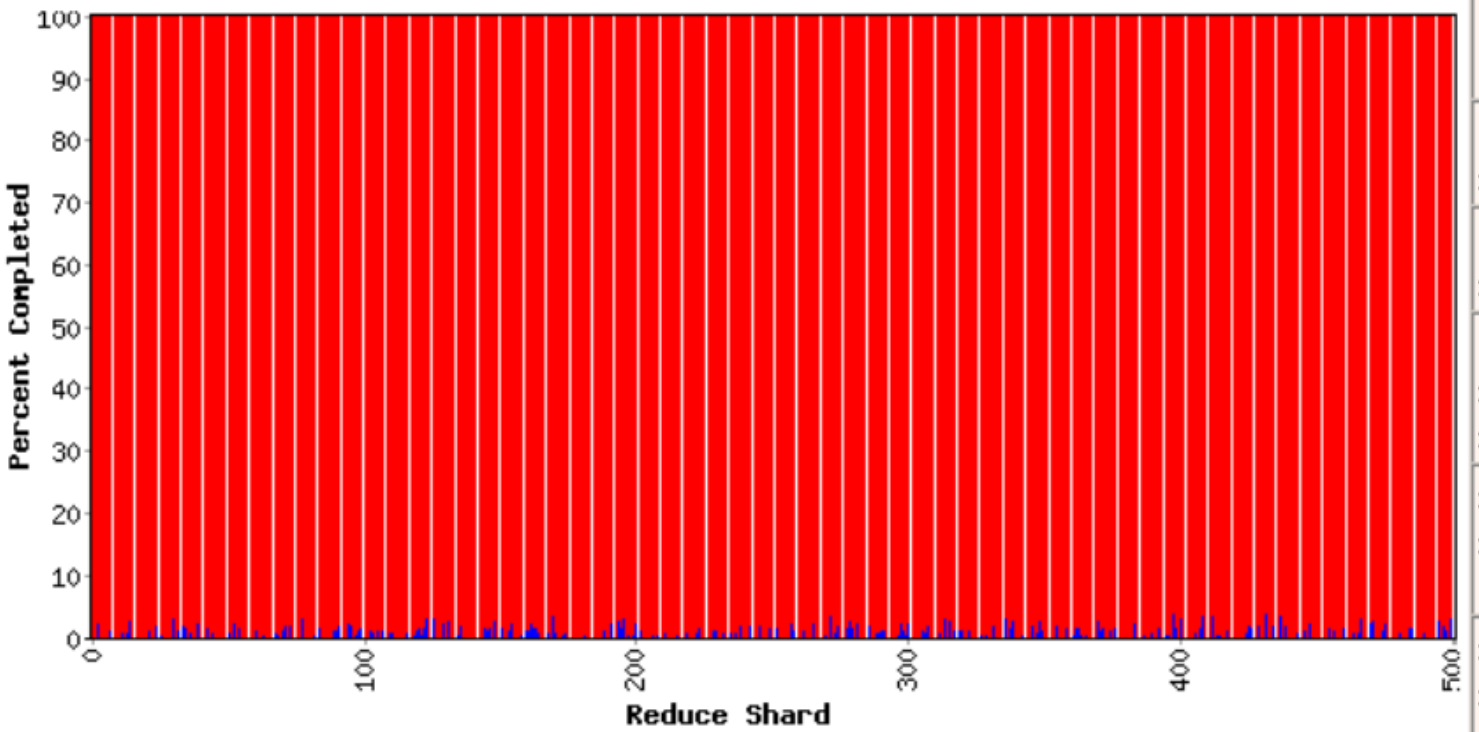
Variable	
Mapped (MB/s)	706.5
Shuffle (MB/s)	419.2
Output (MB/s)	0.0
doc-index-hits	4982870667
docs-indexed	17229926
dups-in-index-merge	0
mr-operator-calls	17272056
mr-operator-outputs	17229926

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 29 min 45 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	195	305	523499.2	523389.6	523389.6
Reduce	500	0	195	523389.6	2685.2	2742.6



Counters

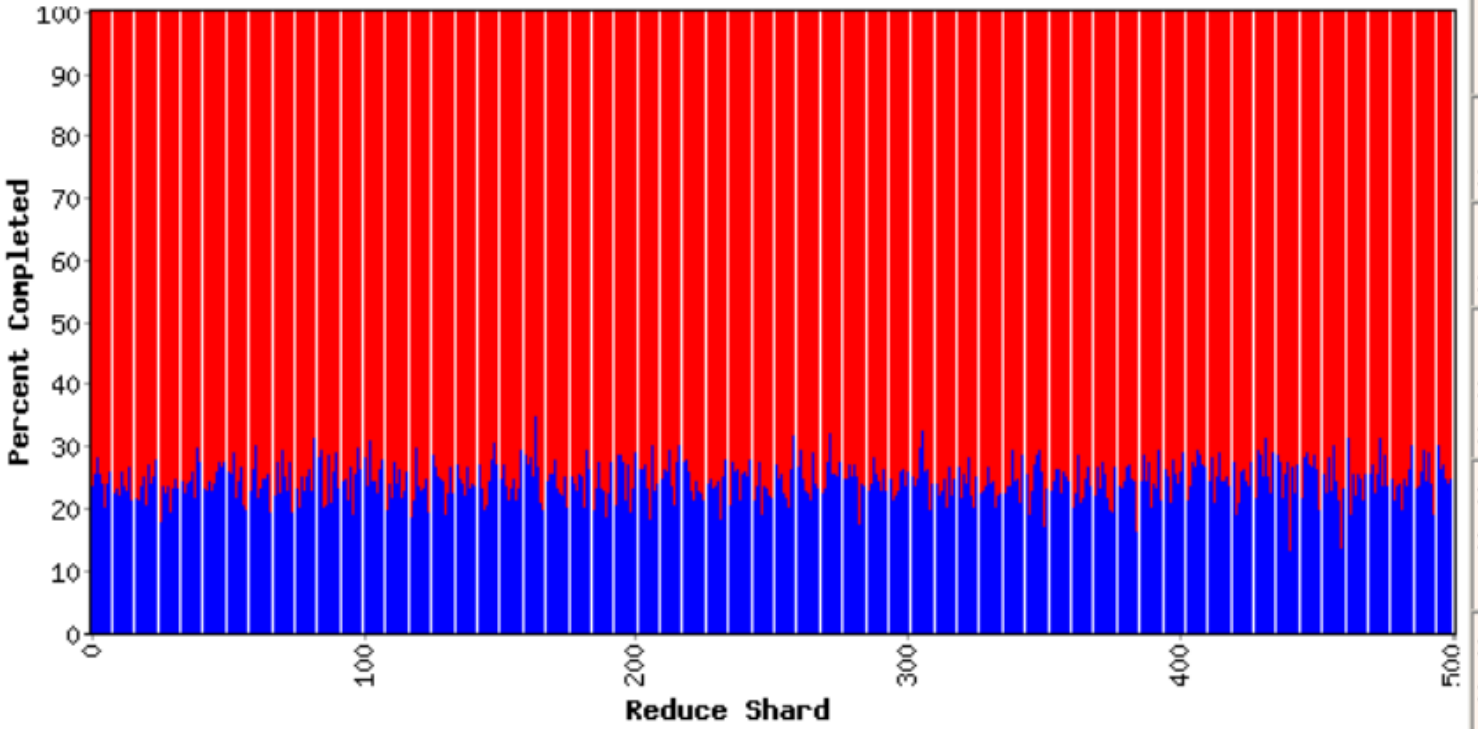
Variable		
Mapped (MB/s)	0.3	
Shuffle (MB/s)	0.5	
Output (MB/s)	45.7	
doc-index-hits	2313178	105
docs-indexed	7936	
dups-in-index-merge	0	
mr-merge-calls	1954105	
mr-merge-outputs	1954105	

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 31 min 34 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	133837.8	136929.6



Counters

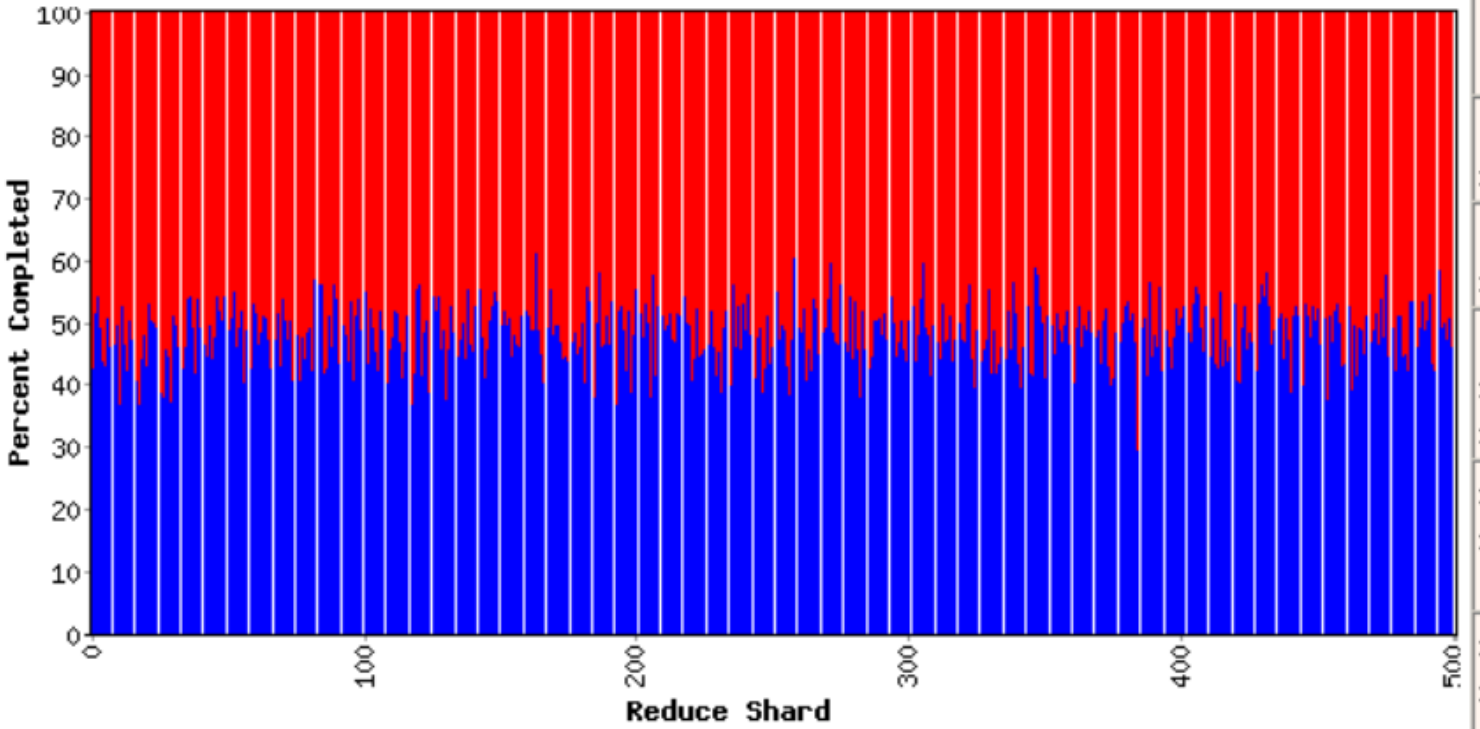
Variable	
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.1
Output (MB/s)	1238.8
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51738599
mr-merge-outputs	51738599

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 33 min 22 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	263283.3	269351.2



Counters

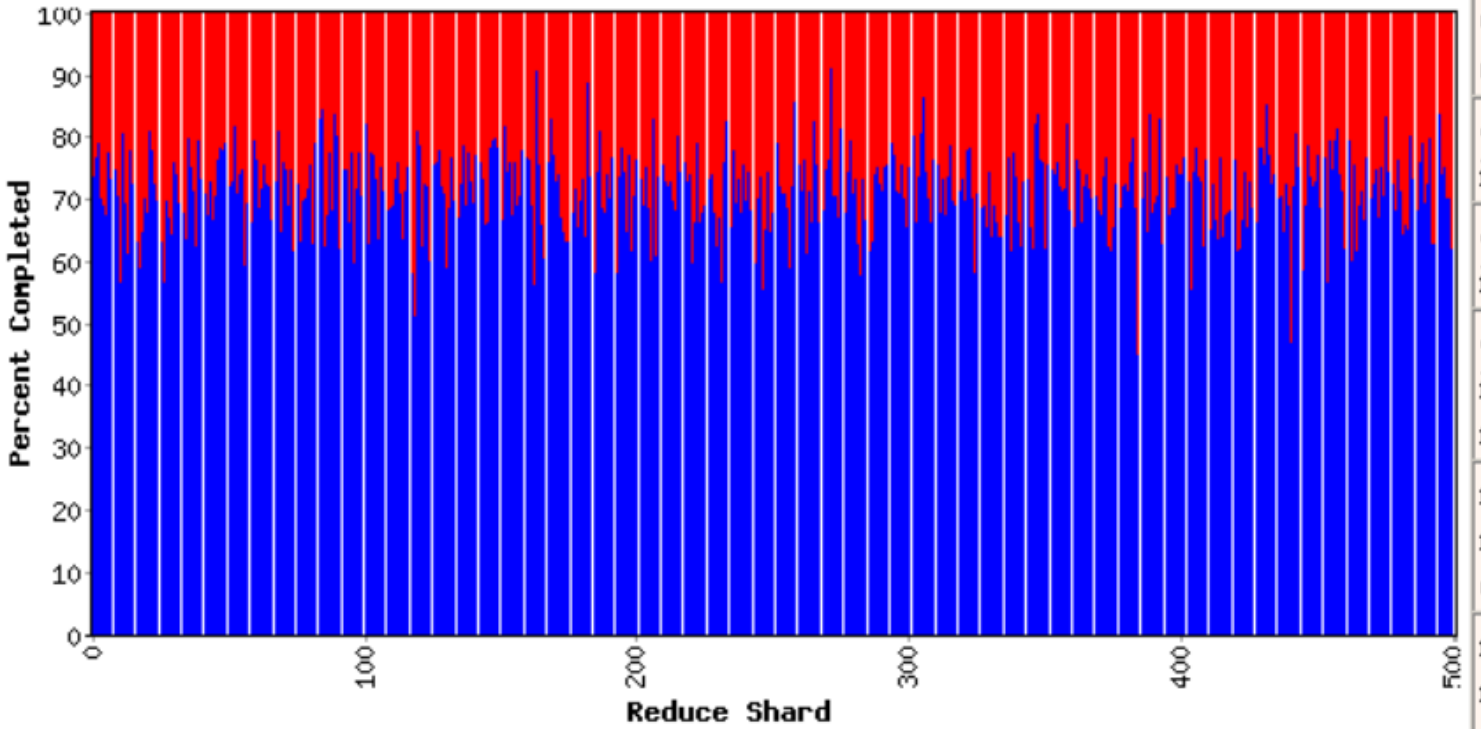
Variable	
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1225.1
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51842100
mr-merge-outputs	51842100

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 35 min 08 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	523499.5	523499.5
Reduce	500	0	500	523499.5	390447.6	399457.2



Counters

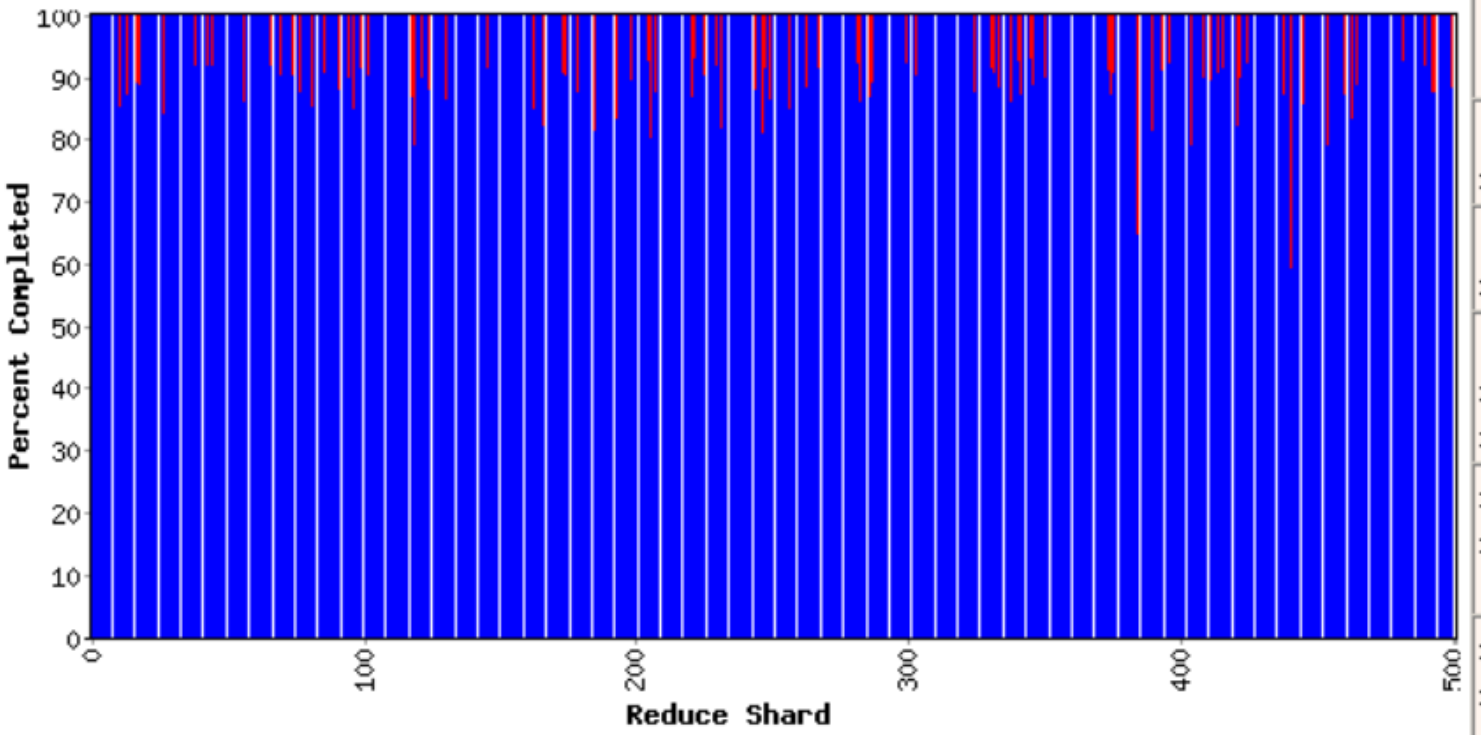
Variable	
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	1222.0
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	51640600
mr-merge-outputs	51640600

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 37 min 01 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	520468.6	520468.6
Reduce	500	406	94	520468.6	512265.2	514373.3



Counters

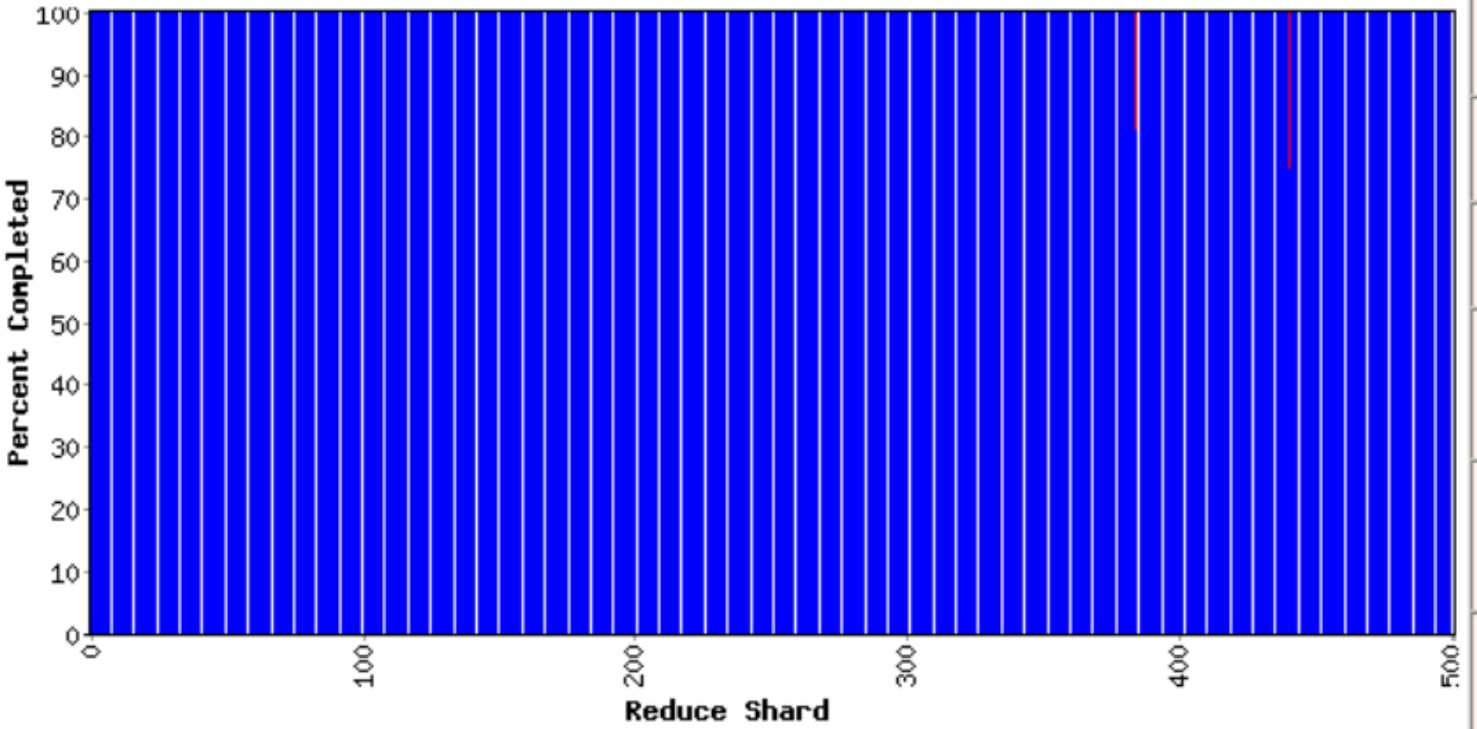
Variable	
Mapped (MB/s)	0.0
Shuffle (MB/s)	0.0
Output (MB/s)	849.5
doc-index-hits	0
docs-indexed	0
dups-in-index-merge	0
mr-merge-calls	35083350
mr-merge-outputs	35083350

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 38 min 56 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519781.8	519781.8
Reduce	500	498	2	519781.8	519394.7	519440.7



Counters

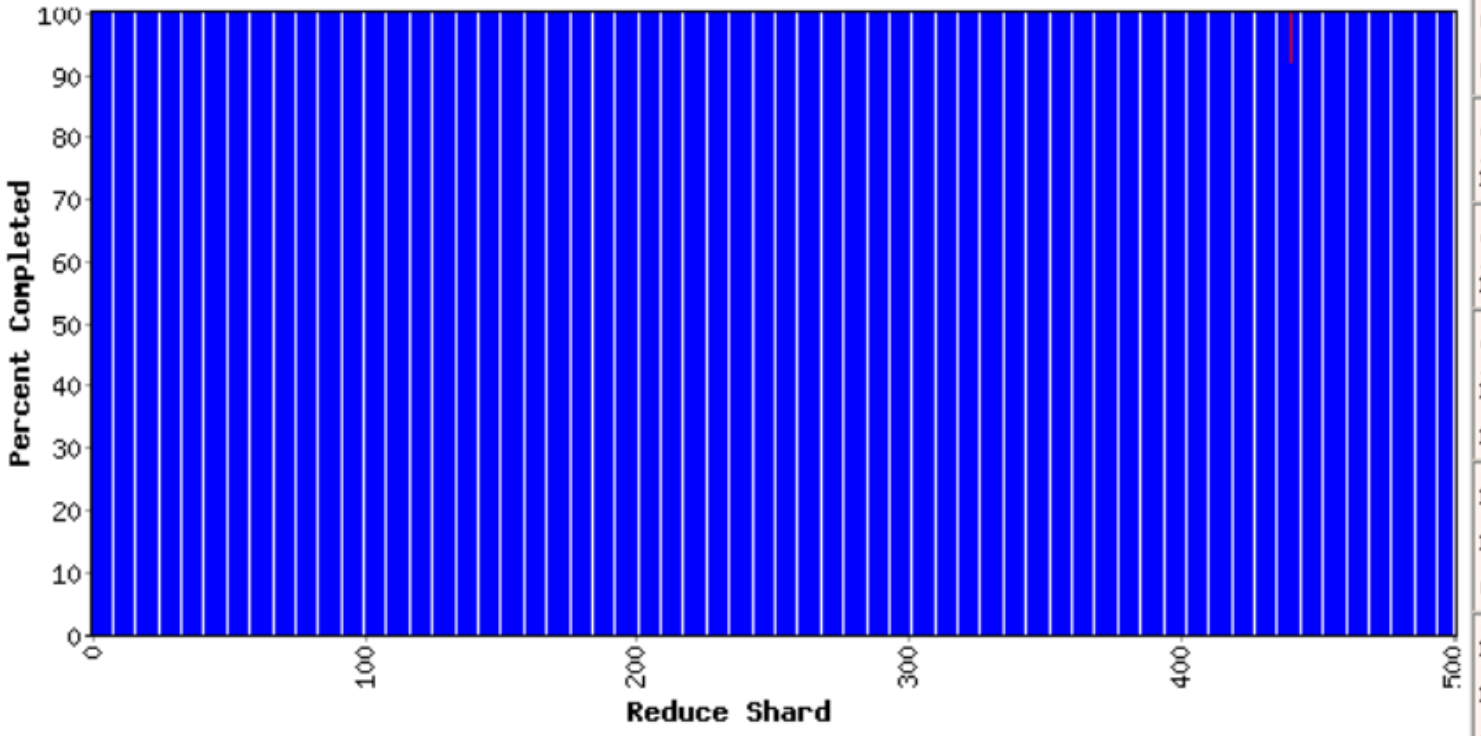
Variable		
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	9.4	
doc-index-hits	0	1056
docs-indexed	0	3
dups-in-index-merge	0	
mr-merge-calls	394792	3
mr-merge-outputs	394792	3

MapReduce status: MR_Indexer-beta6-large-2003_10_28_00_03

Started: Fri Nov 7 09:51:07 2003 -- up 0 hr 40 min 43 sec

1707 workers; 1 deaths

Type	Shards	Done	Active	Input(MB)	Done(MB)	Output(MB)
Map	13853	13853	0	878934.6	878934.6	523499.2
Shuffle	500	500	0	523499.2	519774.3	519774.3
Reduce	500	499	1	519774.3	519735.2	519764.0



Counters

Variable		
Mapped (MB/s)	0.0	
Shuffle (MB/s)	0.0	
Output (MB/s)	1.9	
doc-index-hits	0	105
docs-indexed	0	
dups-in-index-merge	0	
mr-merge-calls	73442	
mr-merge-outputs	73442	

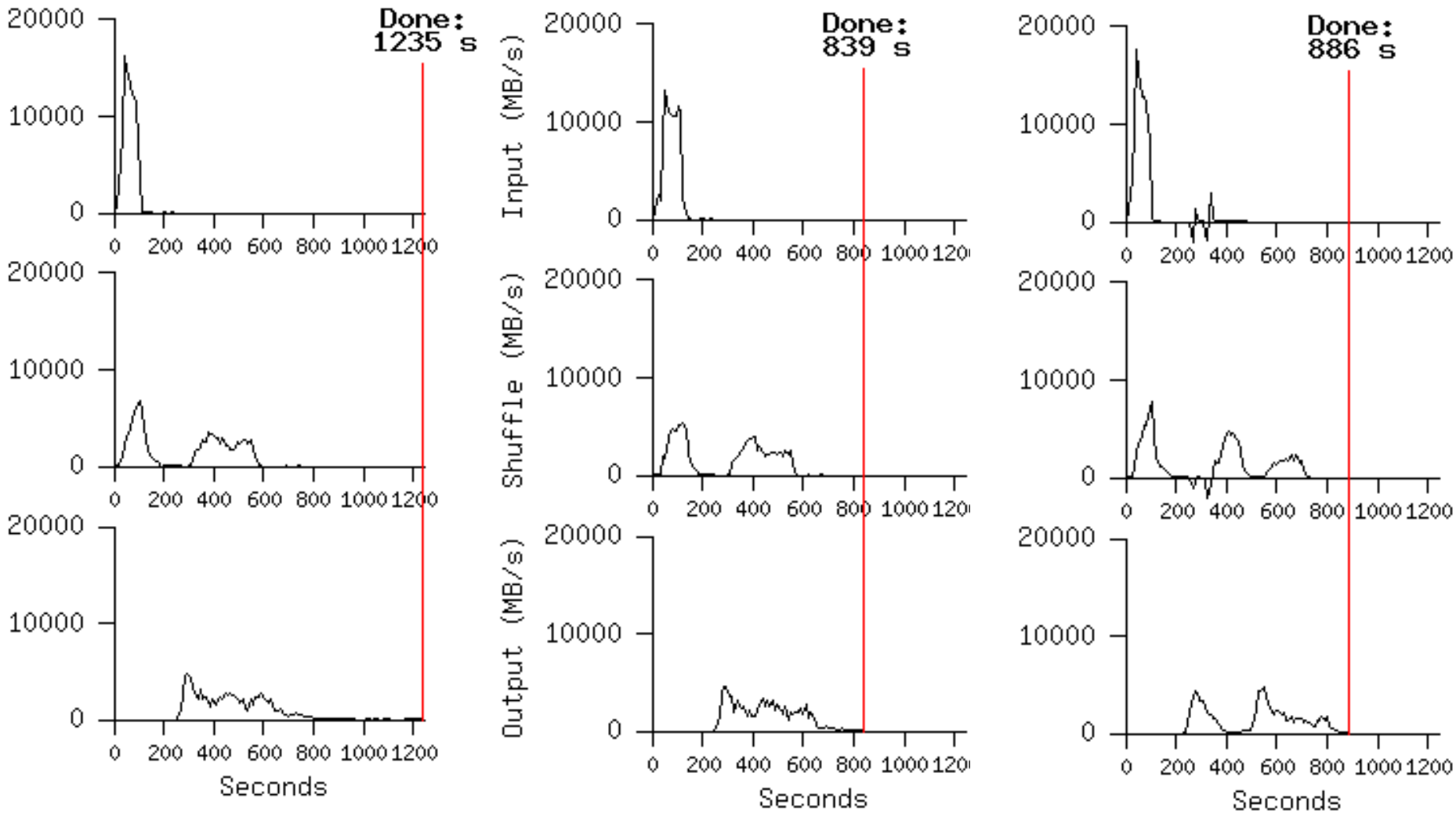
MapReduce Failure Handling

- On worker failure:
  - Detect failure via periodic heartbeats
  - Re-execute completed and in-progress map tasks
  - Re-execute in progress reduce tasks
  - Task completion committed through master
- Master failure:
  - Could handle, but don't yet (master failure unlikely)
- Robust: lost 1600 of 1800 machines once, but finished fine

MapReduce Redundant Execution

- Slow workers significantly lengthen completion time
  - Other jobs consuming resources on machine
  - Bad disks with soft errors transfer data very slowly
  - Weird things: processor caches disabled (!!)
- Solution: Near end of phase, spawn backup copies of tasks
  - Whichever one finishes first "wins"
- Effect: Dramatically shortens job completion time
  - 3% more resources, large tasks 30% faster

Impact of Redundant Execution, Failure for 10B record Sort using 1800 servers



no backup tasks

with backup tasks

with 200 failures

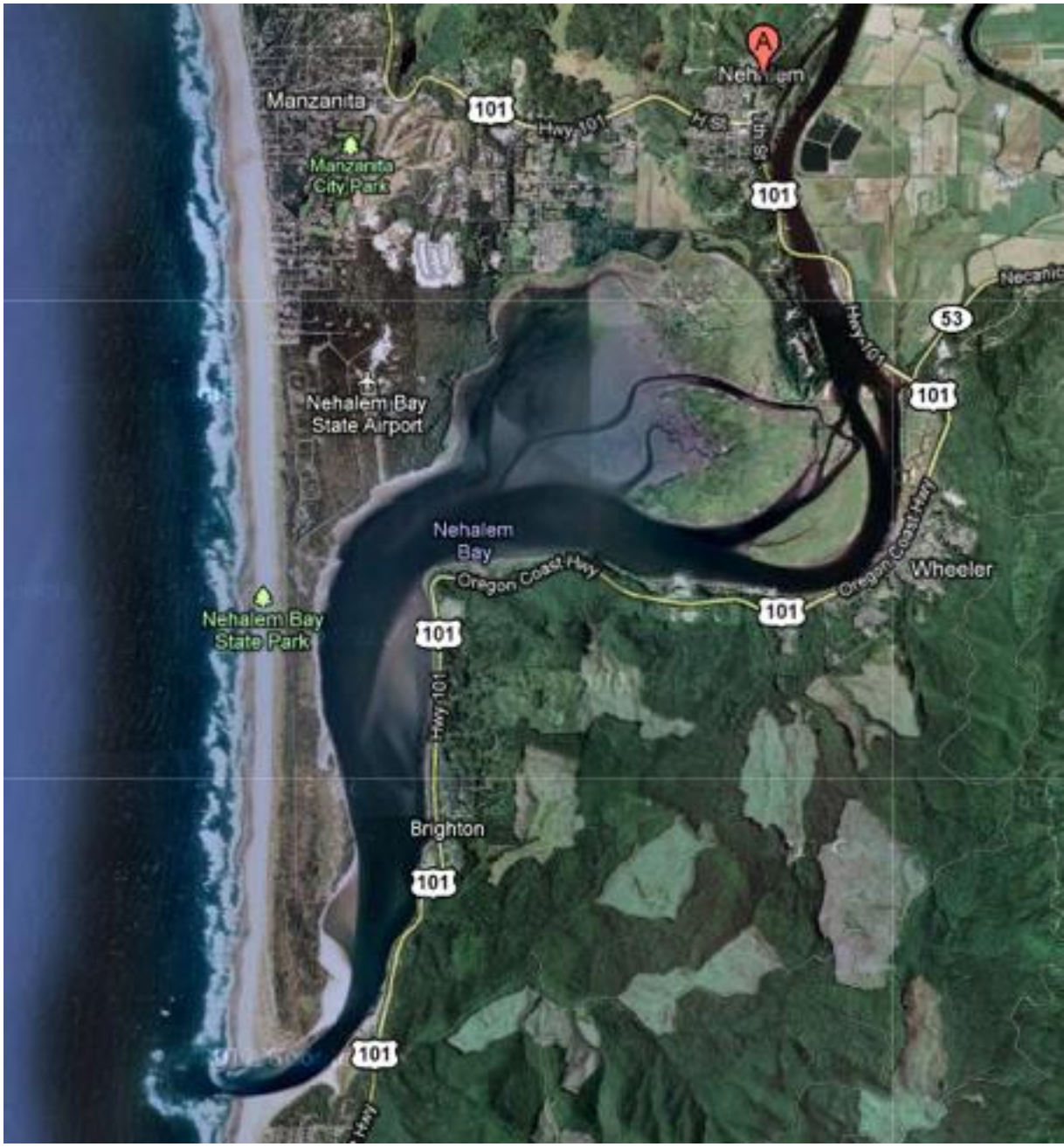
Agenda

- Request Level Parallelism
- MapReduce
- Administrivia
- MapReduce Processing
- Break
- Modern Microarchitectures

Intel Nehalem

- Look at microprocessor from Intel in servers in your 61C labs and your laptops
- Nehalem is code name for microarchitecture at heart of Core i7 and Xeon 5500 series server chips
  - Intel legal said had to pick names of rivers for code names (vs. “Destroyer”)
- First released at end of 2008
- Die size 263mm^2 at 45 nm
- 731M transistors

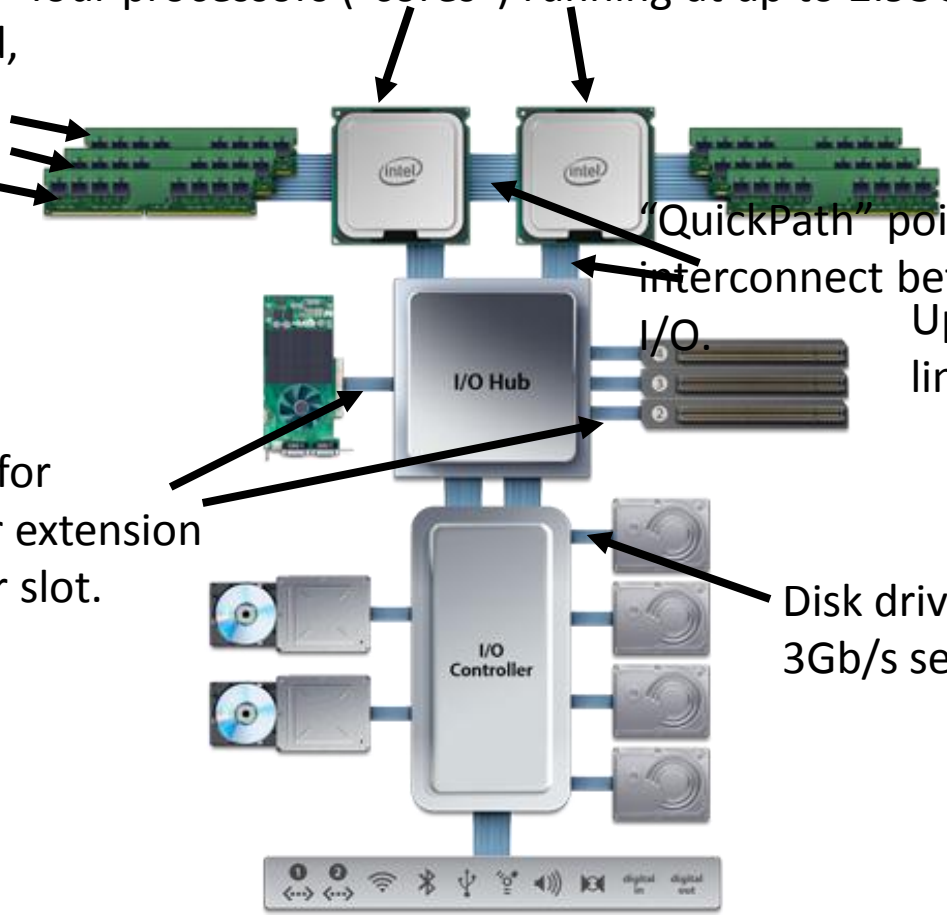
Nehalem River, Oregon



Nehalem System Example: Apple Mac Pro Desktop 2010

Two Nehalem Chips ("Sockets"), each containing four processors ("cores") running at up to 2.93GHz

Each chip has three DRAM channels attached, each 8 bytes wide at 1.066Gb/s (3*8.5GB/s). Can have up to two DIMMs on each channel (up to 4GB/DIMM)



"QuickPath" point-point system interconnect between CPUs and I/O.

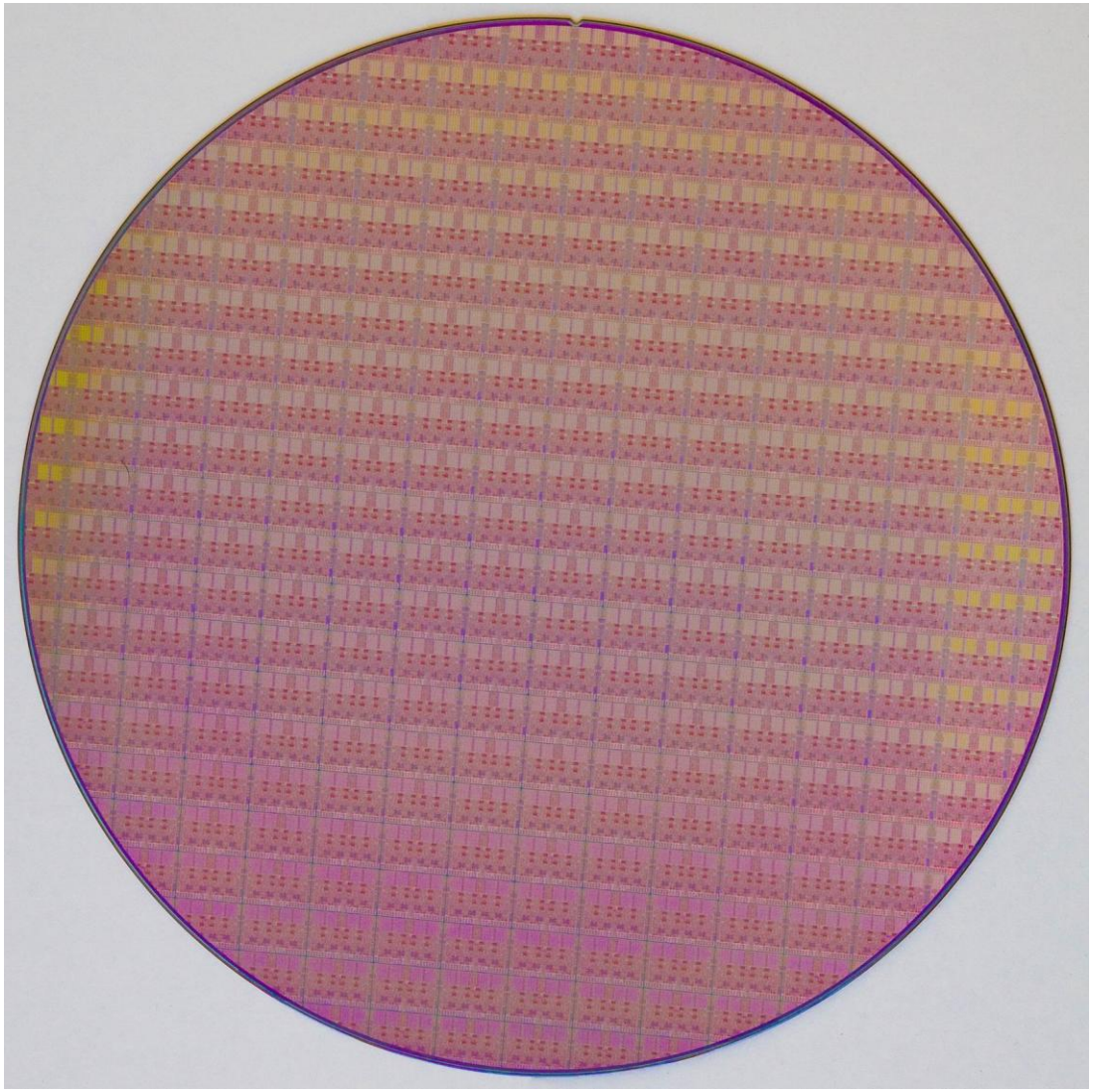
Up to 25.6 GB/s per link.

PCI Express connections for Graphics cards and other extension boards. Up to 8 GB/s per slot.

Disk drives attached with 3Gb/s serial ATA link

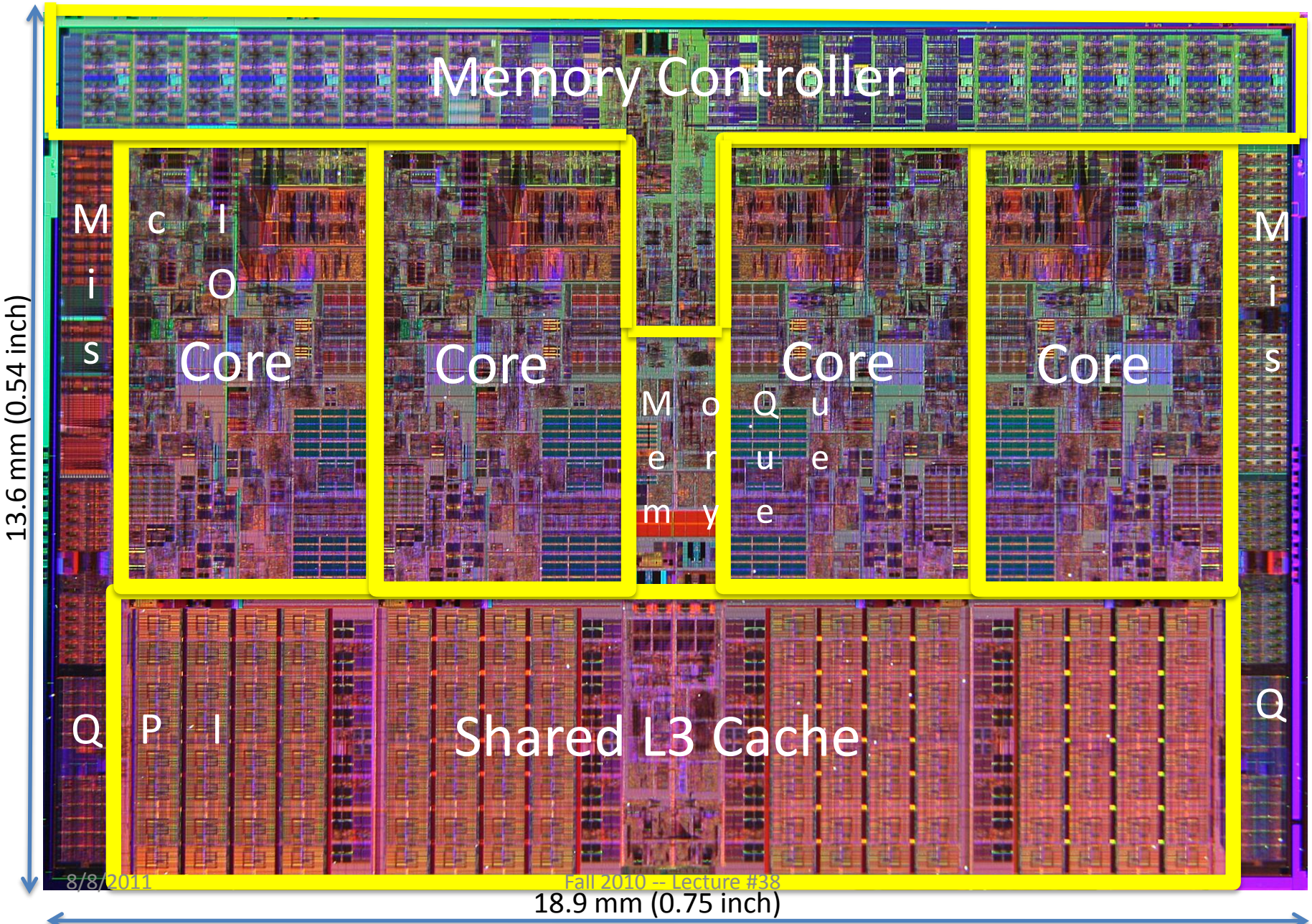
Slower peripherals (Ethernet, USB, Firewire, WiFi, Bluetooth, Audio)

Nehalem 12-inch Wafer: 280 dies

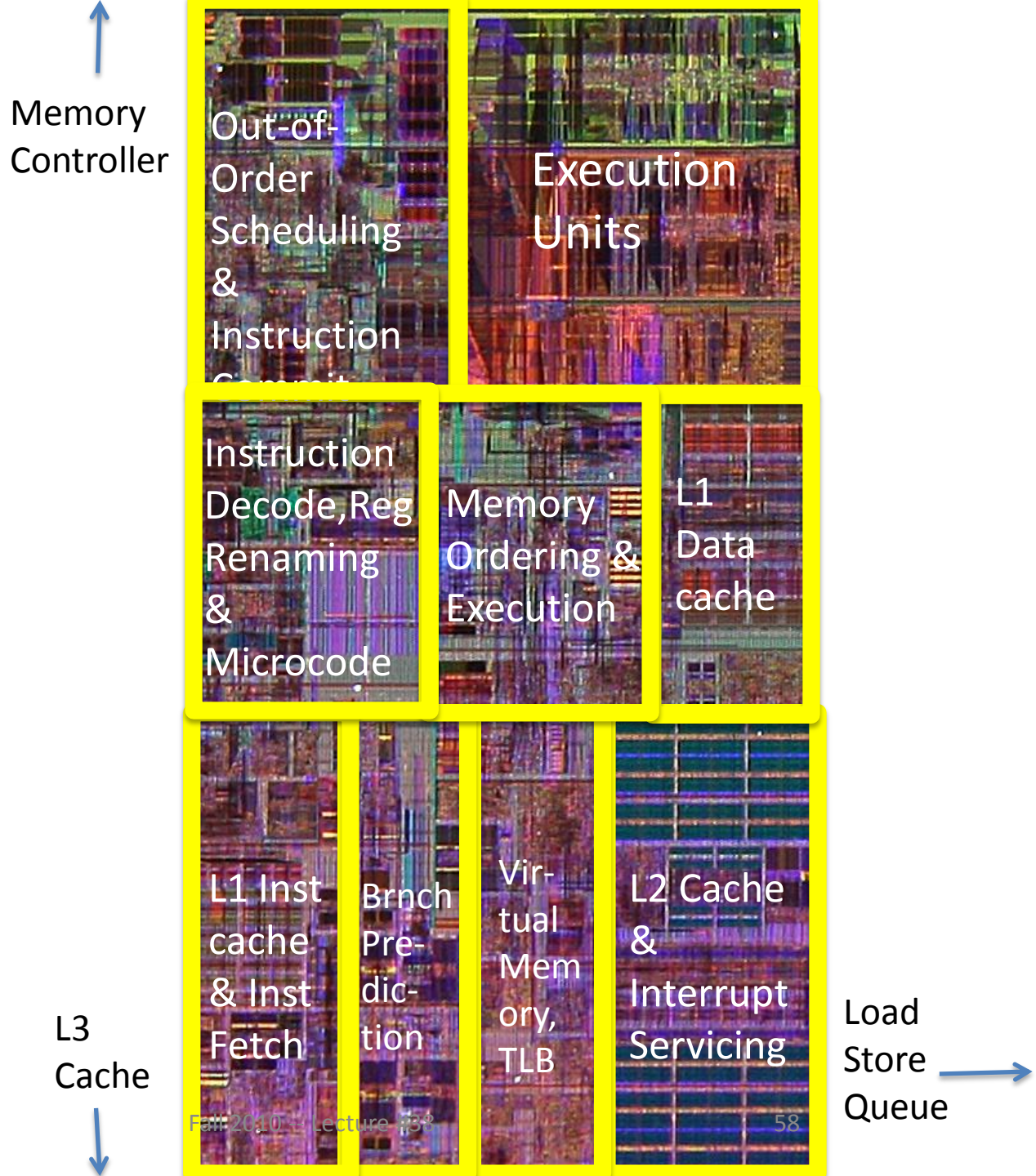


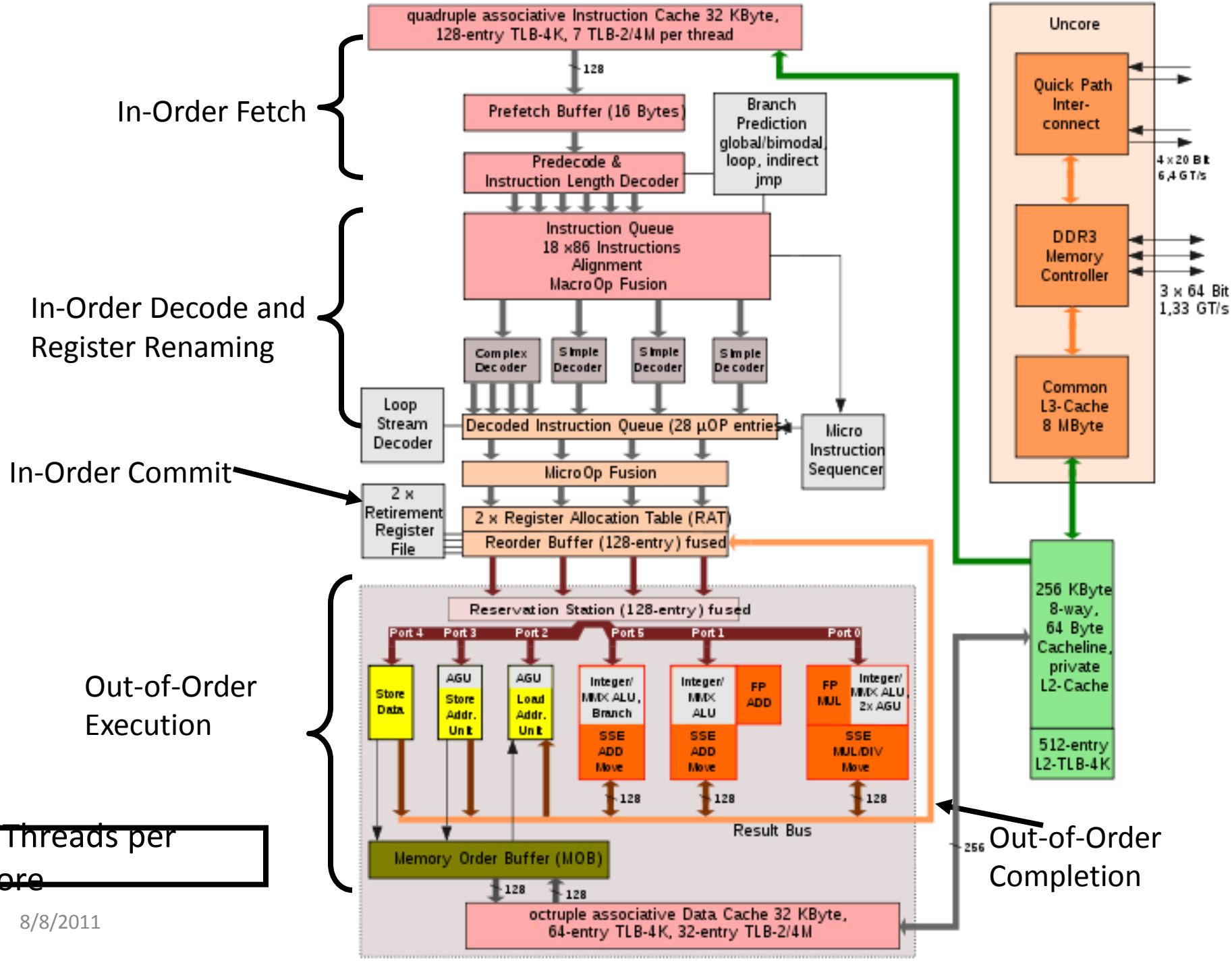
Same diameter as medium Domino's pizza

Nehalem Die Photo



Core Area Breakdown





2 Threads per Core

Extending Performance and Energy Efficiency

- Intel® SSE4.2 Instruction Set Architecture (ISA) Leadership in 2008

Accelerated String and Text Processing

Faster XML parsing
Faster search and pattern matching
Novel parallel data matching and comparison operations

STTNI

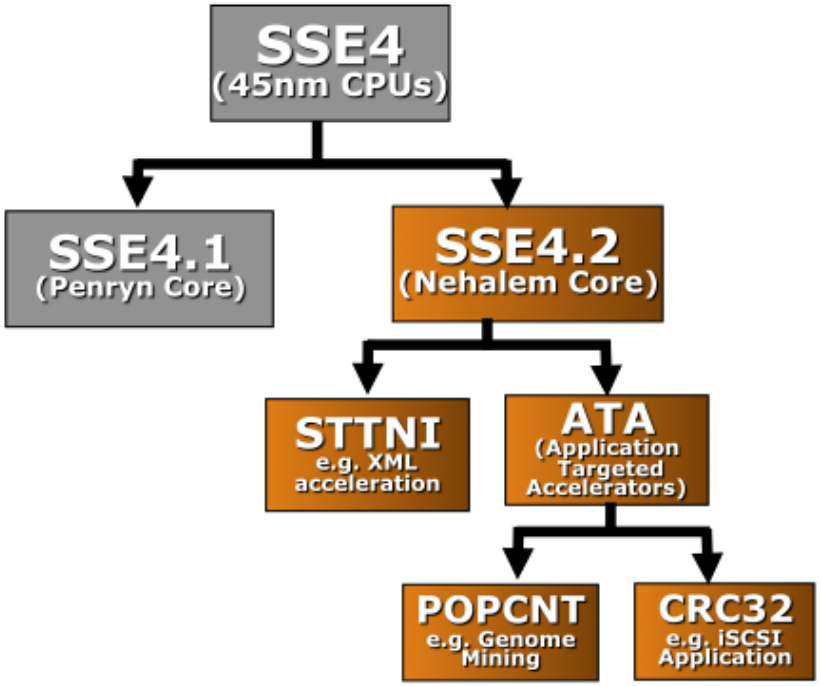
Accelerated Searching & Pattern Recognition of Large Data Sets

Improved performance for Genome Mining,
Handwriting recognition.
Fast Hamming distance / Population count

New Communications Capabilities

Hardware based CRC instruction
Accelerated Network attached storage
Improved power efficiency for Software I-SCSI, RDMA, and SCTP

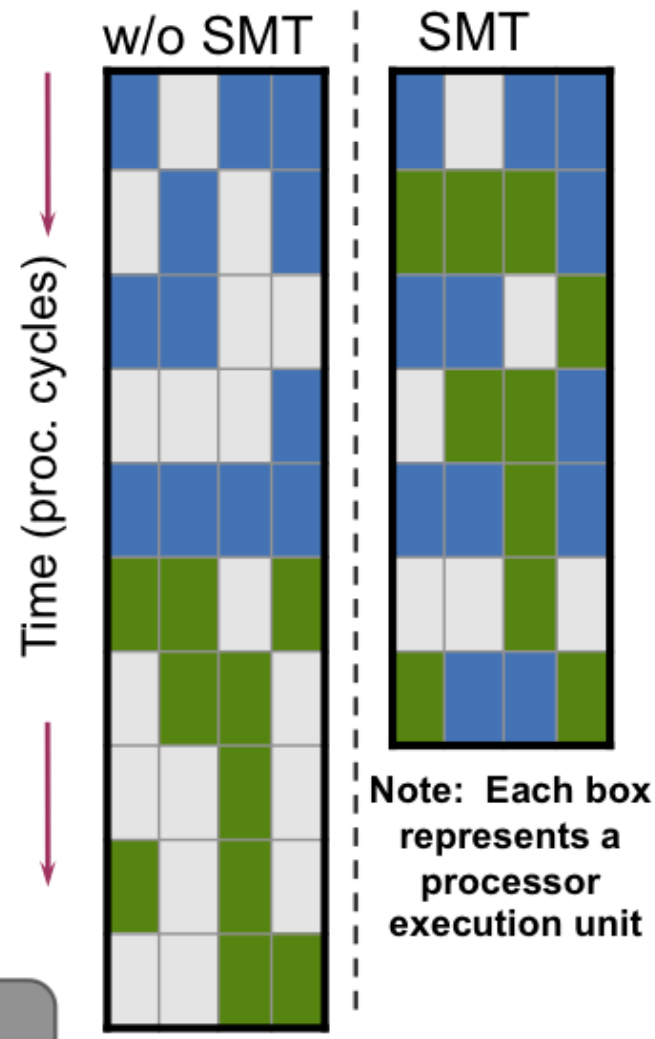
ATA



What should the applications, OS and VMM vendors do?:
Understand the benefits & take advantage of new instructions in 2008.
Provide us feedback on instructions ISV would like to see for next generation of applications

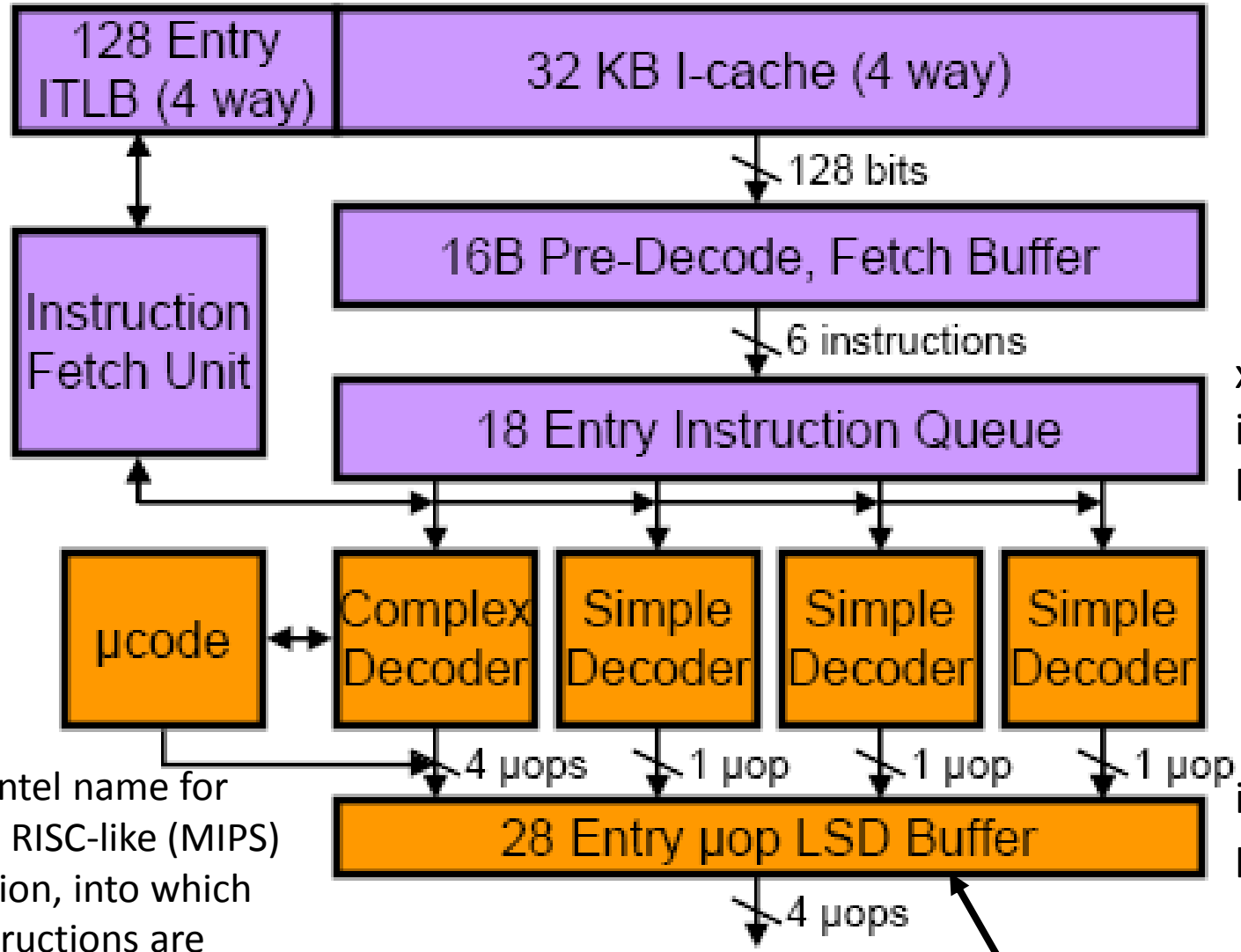
Intel® Hyper-Threading Technology

- Also known as Simultaneous Multi-Threading (SMT)
  - Run 2 threads at the same time per core
- Take advantage of 4-wide execution engine
  - Keep it fed with multiple threads
  - Hide latency of a single thread
- Most ***power efficient*** performance feature
  - Very low die area cost
  - Can provide significant performance benefit depending on application
  - Much more efficient than adding an entire core
- Intel® Core™ microarchitecture (Nehalem) advantages
  - Larger caches
  - Massive memory BW



Simultaneous multi-threading enhances performance and energy efficiency

Front-End Instruction Fetch & Decode



x86 instruction bits

internal µOP bits

µOP is Intel name for internal RISC-like (MIPS) instruction, into which x86 instructions are translated

Loop Stream Detector (can run short loops out of the buffer)

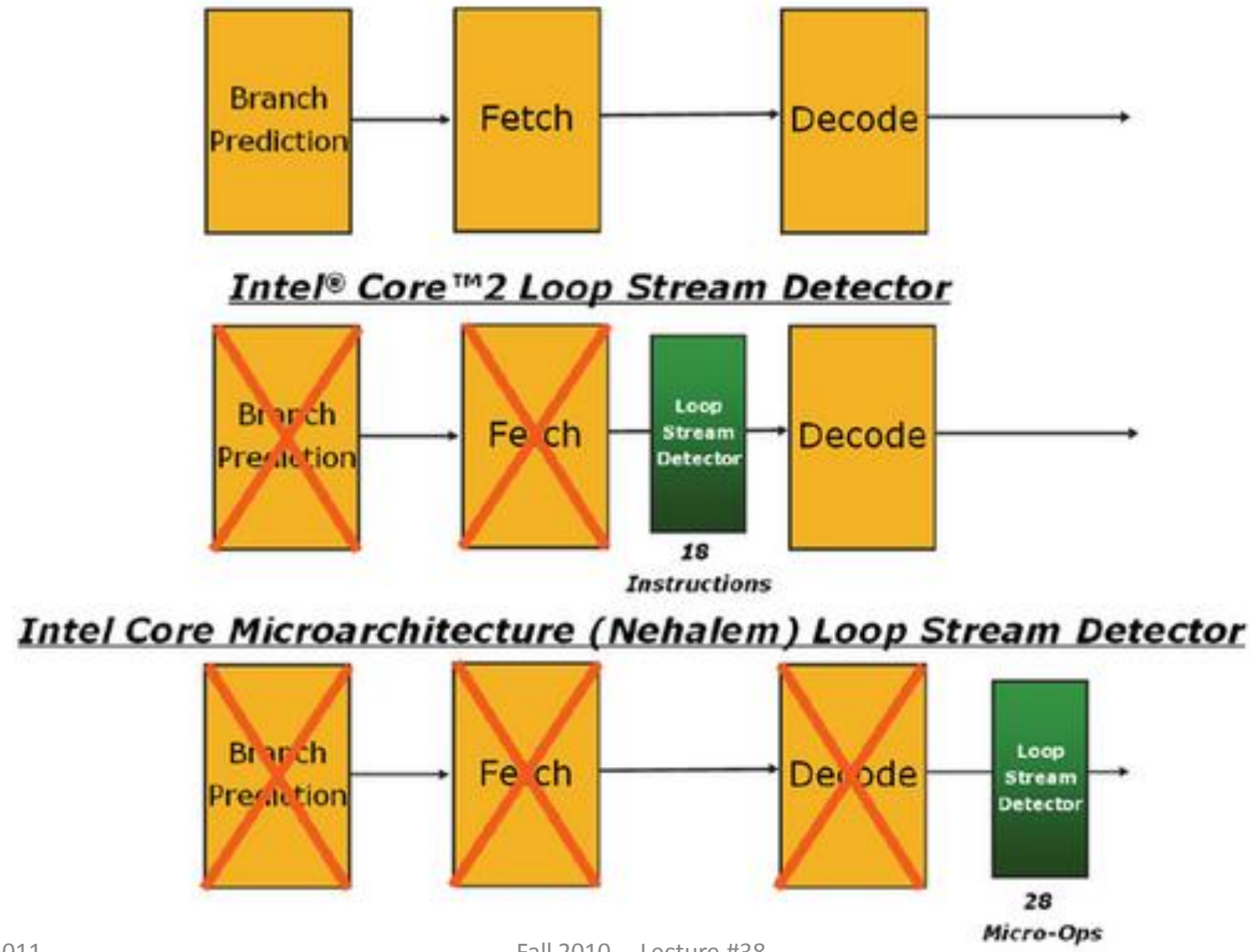
x86 Decoding

- Translate up to 4 x86 instructions into µOPS (≈MIPS or RISC instructions) each cycle
- Only first x86 instruction in group can be complex (maps to 1-4 µOPS), rest must be simple (map to one µOP)
- Even more complex instructions, jump into microcode engine which spits out stream of µOPS

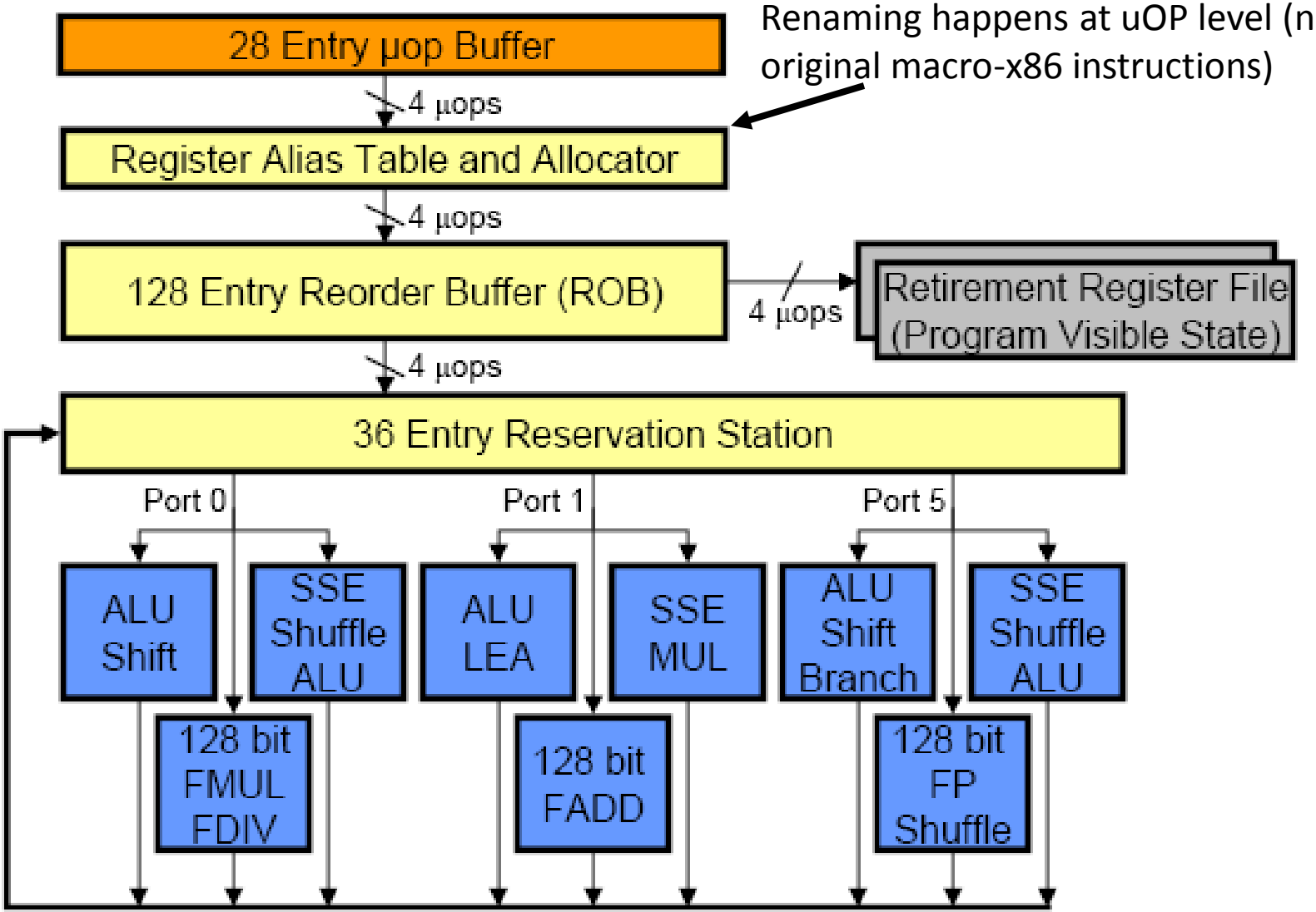
Branch Prediction

- Part of instruction fetch unit
- Several different types of branch predictor
  - Details not public
- Two-level Branch Table Buffer
- Loop count predictor
  - How many backwards taken branches before loop exit
- Return Stack Buffer
  - Holds subroutine targets
  - Separate return stack buffer for each SMT thread

Loop Stream Detectors save Power



Out-of-Order Execution Engine

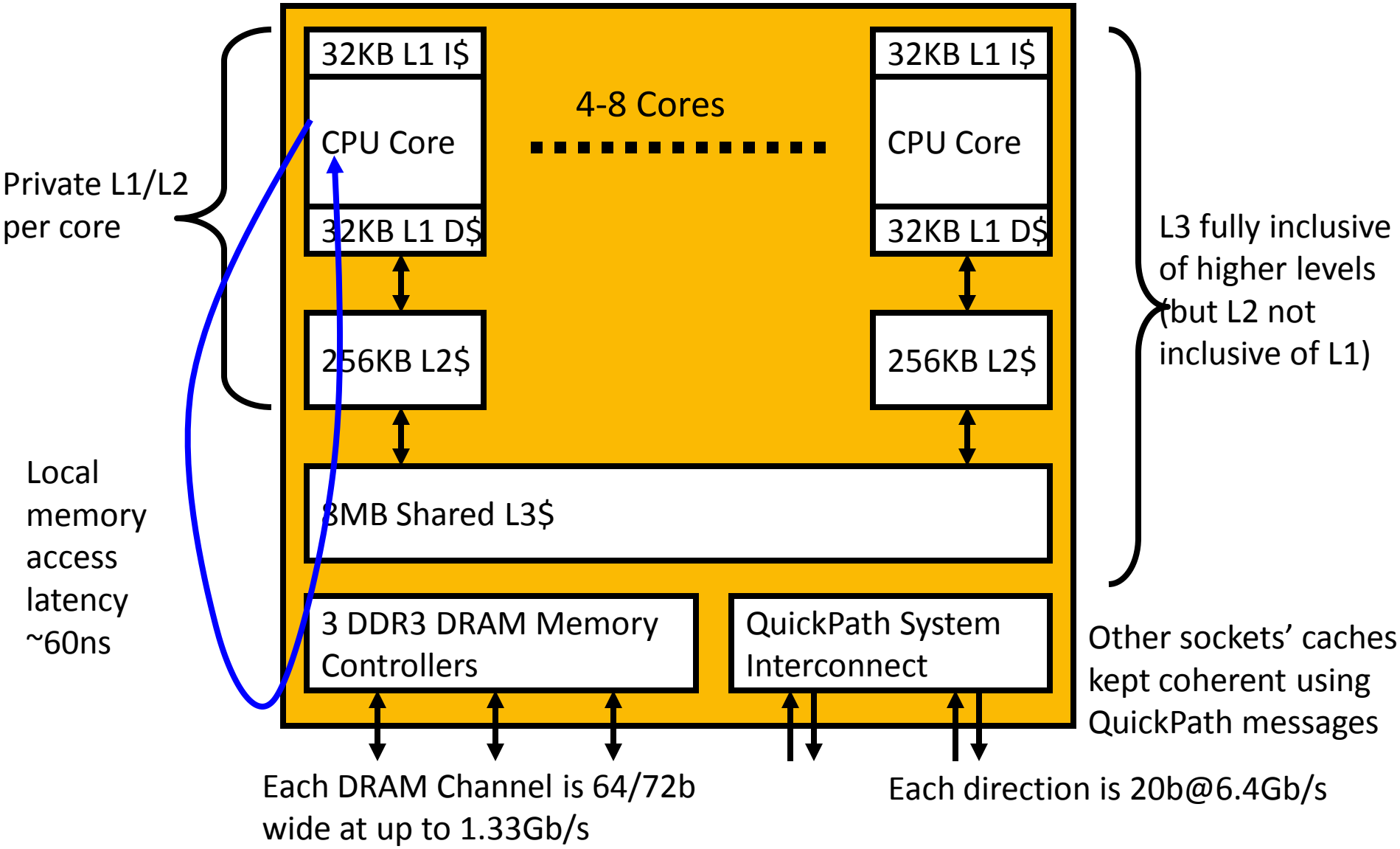


Renaming happens at uOP level (not original macro-x86 instructions)

Multithreading effects in Out-of-Order Execution Core

- Reorder buffer (remembers program order and exception status for in-order commit) has 128 entries divided statically and equally between both threads
- Reservation stations (instructions waiting for operands for execution) have 36 entries competitively shared by threads

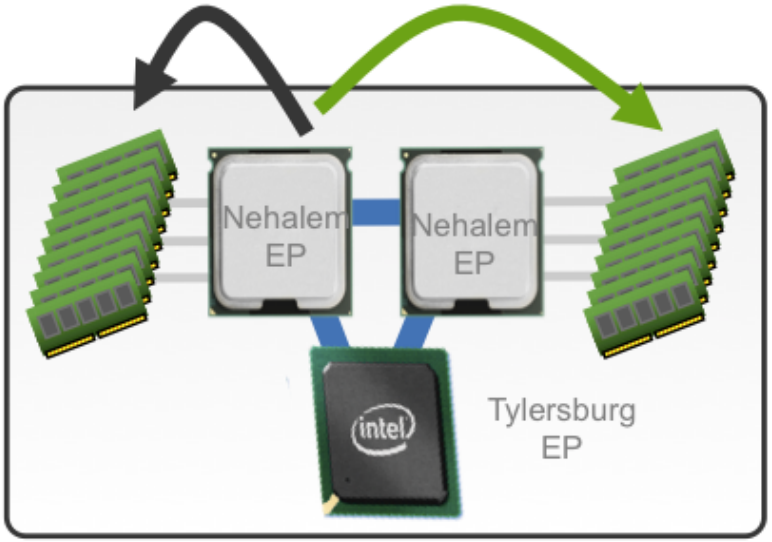
Nehalem Memory Hierarchy Overview

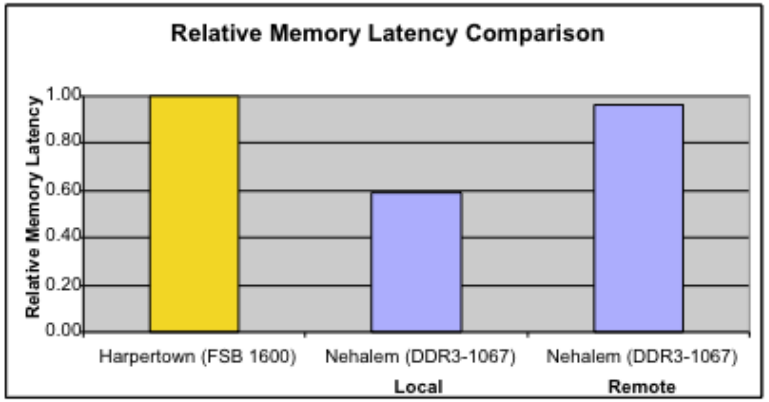


Non-Uniform Memory Access (NUMA)

- FSB architecture
  - All memory in one location
- Starting with Intel® Core™ microarchitecture (Nehalem)
  - Memory located in multiple places
- Latency to memory dependent on location
- Local memory has highest BW, lowest latency
- Remote Memory still very fast

Ensure software is NUMA-optimized for best performance

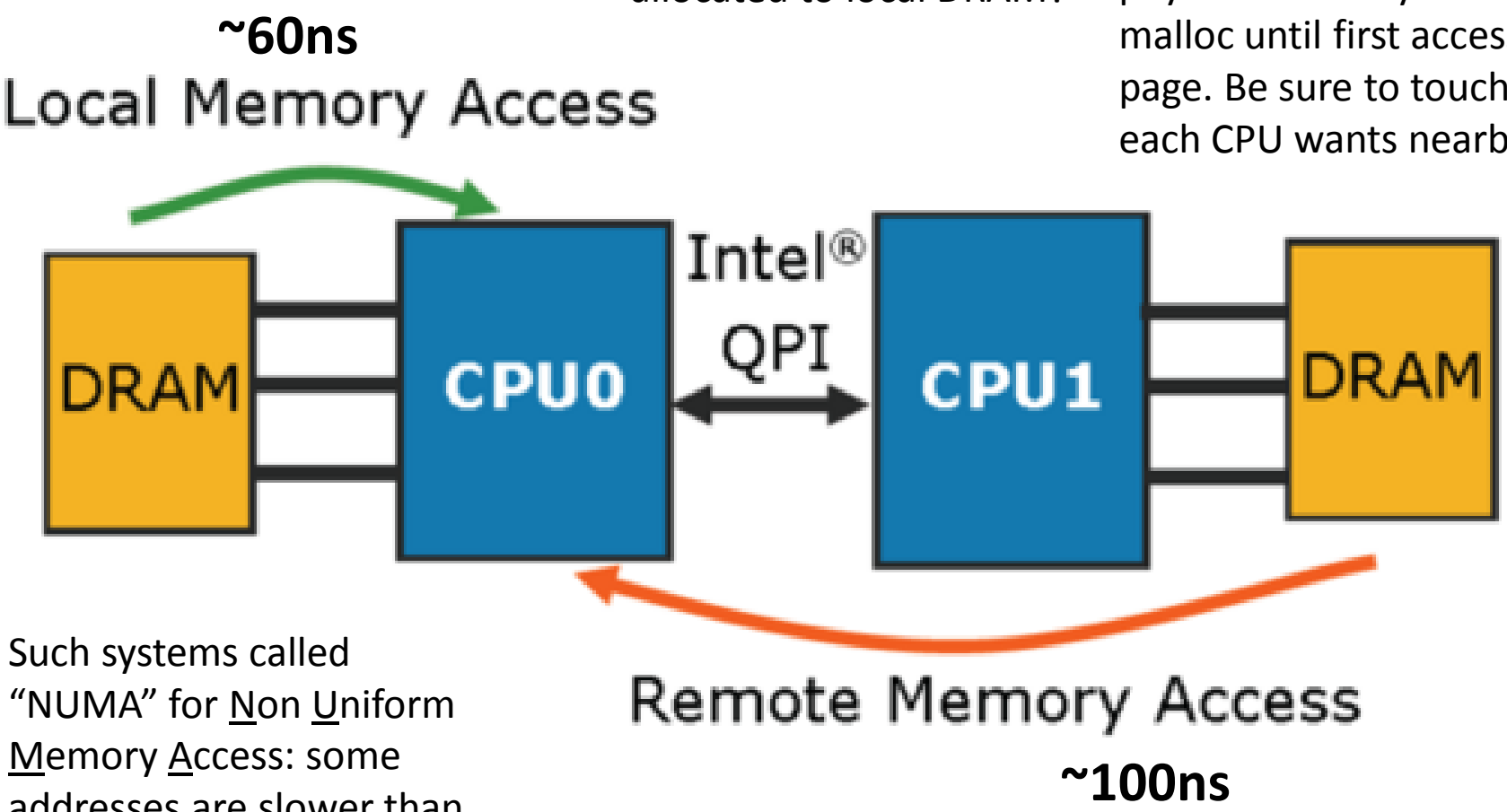




All Sockets can Access all Data

How ensure that get data allocated to local DRAM?

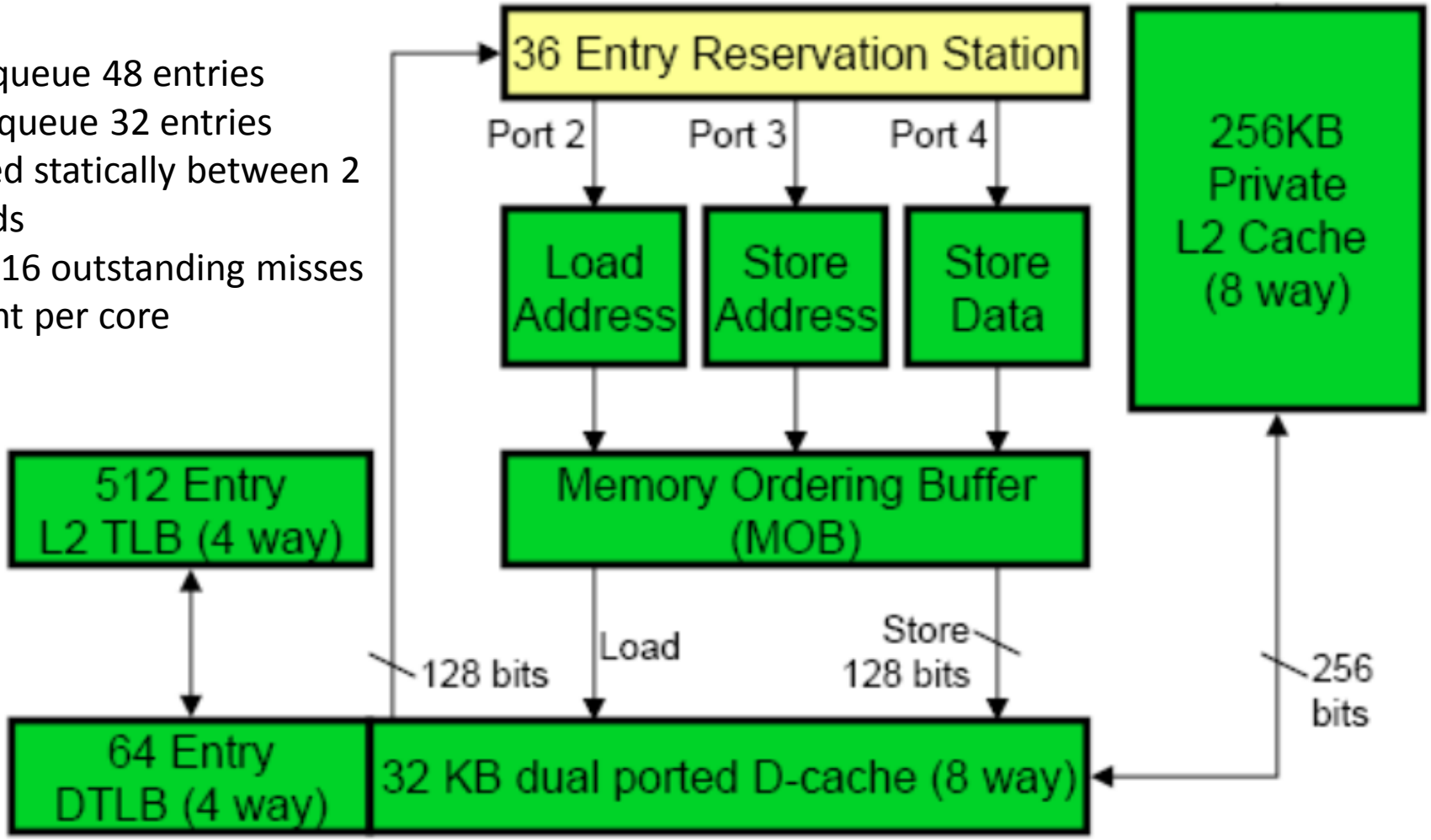
OS doesn't allocate pages to physical memory after malloc until first access to page. Be sure to touch what each CPU wants nearby

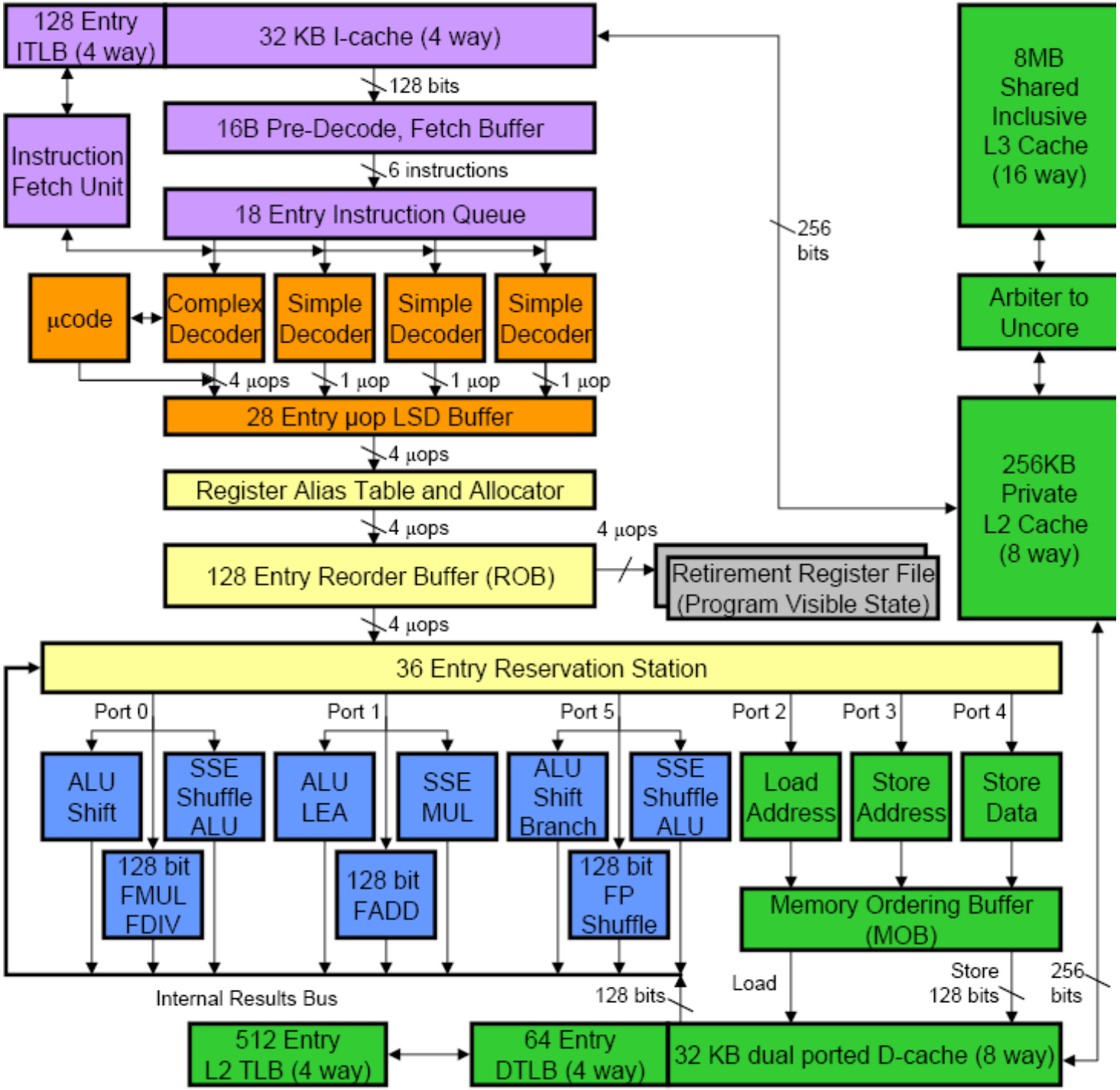


Such systems called "NUMA" for Non Uniform Memory Access: some addresses are slower than others

Core's Private Memory System

Load queue 48 entries
Store queue 32 entries
Divided statically between 2 threads
Up to 16 outstanding misses in flight per core




128 Entry ITLB (4 way)
32 KB I-cache (4 way)
128 bits
16B Pre-Decode, Fetch Buffer
Instruction Fetch Unit
6 instructions
18 Entry Instruction Queue
μcode
Complex Decoder
Simple Decoder
Simple Decoder
Simple Decoder
4 μops
1 μop
1 μop
1 μop
28 Entry μop LSD Buffer
4 μops
Register Alias Table and Allocator
4 μops
4 μops
128 Entry Reorder Buffer (ROB)
Retirement Register File (Program Visible State)
4 μops
36 Entry Reservation Station
Port 0
Port 1
Port 5
Port 2
Port 3
Port 4
ALU Shift
SSE Shuffle ALU
ALU LEA
SSE MUL
ALU Shift Branch
SSE Shuffle ALU
Load Address
Store Address
Store Data
128 bit FMUL FDIV
128 bit FADD
128 bit FP Shuffle
Memory Ordering Buffer (MOB)
Internal Results Bus
128 bits
Load
Store 128 bits
256 bits
512 Entry L2 TLB (4 way)
64 Entry DTLB (4 way)
32 KB dual ported D-cache (8 way)
256 bits
8MB Shared Inclusive L3 Cache (16 way)
Arbiter to Uncore
256KB Private L2 Cache (8 way)

What to do with So Many Features?

- *"Introduction to Performance Analysis on Nehalem Based Processors"*, 72 pages
- software.intel.com/sites/products/collateral/hpc/vtune/performance_analysis_guide.pdf

  "Software optimization based on performance analysis of large existing applications, in most cases, reduces to optimizing the code generation by the compiler and optimizing the memory access. Optimizing the code generation by the compiler requires inspection of the assembler of the time consuming parts of the application and verifying that the compiler generated a reasonable code stream. Optimizing the memory access is a complex issue involving the bandwidth and latency capabilities of the platform, hardware and software prefetching efficiencies and the virtual address layout of the heavily accessed variables."

Summary

- Request-Level Parallelism
  - High request volume, each largely independent of other
  - Use replication for better request throughput, availability
- MapReduce Data Parallelism
  - Divide large data set into pieces for independent parallel processing
  - Combine and process intermediate results to obtain final result
- Intel Nehalem
  - Speculative execution: branch prediction, out of order execution, data prefetching
  - Hardware translation and optimization of instruction sequences
  - Opportunistic acceleration (Turbo Mode)

Acknowledgements

- Figures/Info from Intel, David Kanter at Real World Technologies.
- These slides contain material developed and copyright by:
  - Krste Asanovic (UCB)
  - Beeman Strong (Intel)
- UCB material derived from course CS152.