COMP 322: Fundamentals of Parallel Programming

Lecture 35: Cloud Computing, Map Reduce

Vivek Sarkar
Department of Computer Science, Rice University
vsarkar@rice.edu

https://wiki.rice.edu/confluence/display/PARPROG/COMP322

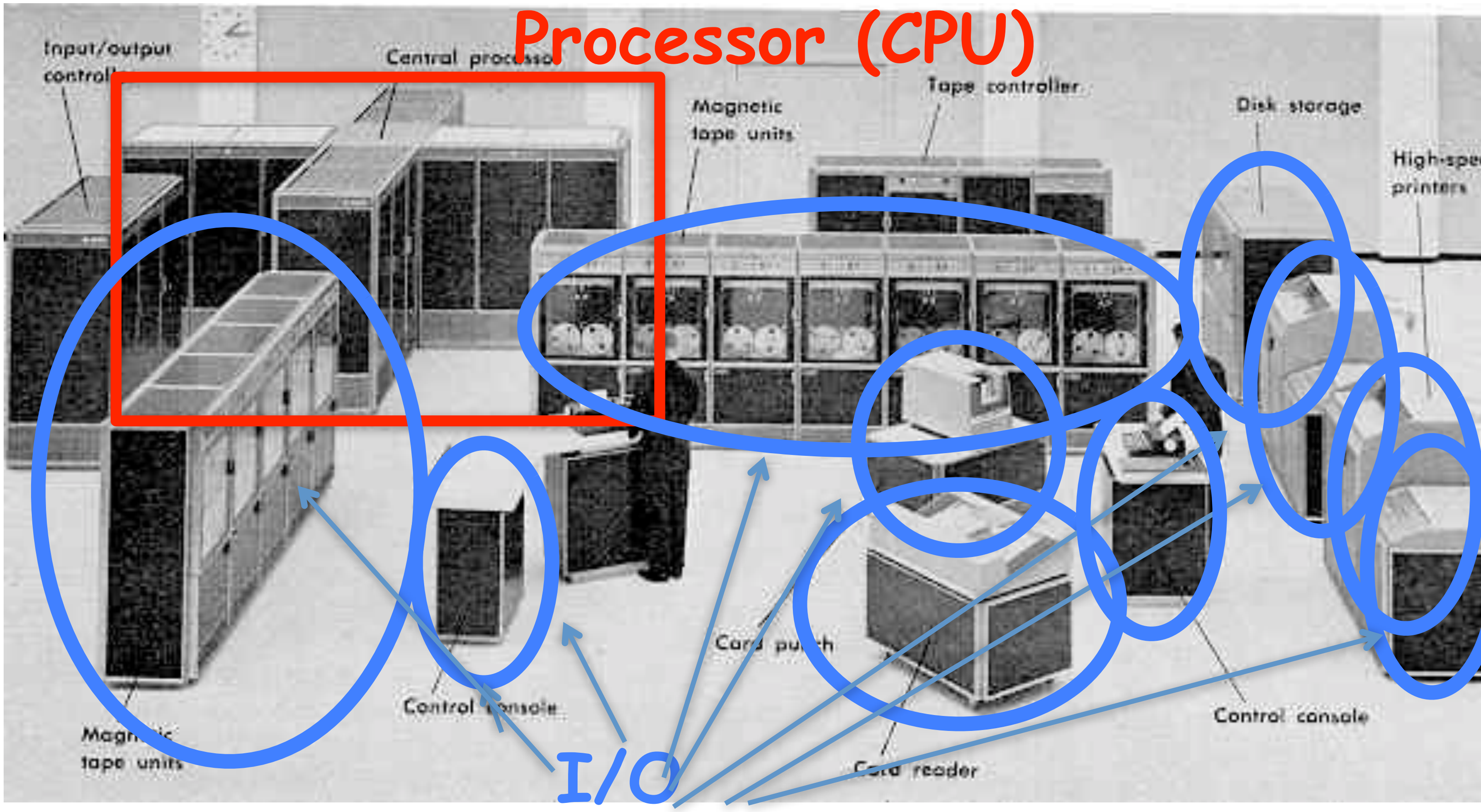
Acknowledgments for Today's Lecture

- Slides from Lectures 1 and 2 in UC Berkeley CS61C course, "Great Ideas in Computer Architecture (Machine Structures), Spring 2012, Instructor: David Patterson
  - http://inst.eecs.berkeley.edu/~cs61c/sp12/
- Slides from MapReduce lecture in Stanford CS 345A course
  - http://infolab.stanford.edu/~ullman/mining/2009/mapreduce.ppt
- Slides from COMP 422 lecture on MapReduce
  - http://www.clear.rice.edu/comp422

Outline

- <u>Warehouse Scale Computers and Cloud Computing</u>
- Map Reduce Programming Model and Runtime System

Computer Eras: Mainframe 1950s-60s



"Big Iron": IBM, UNIVAC, … build $1M computers for businesses => COBOL, Fortran, timesharing OS

Minicomputer Eras: 1970s-80s

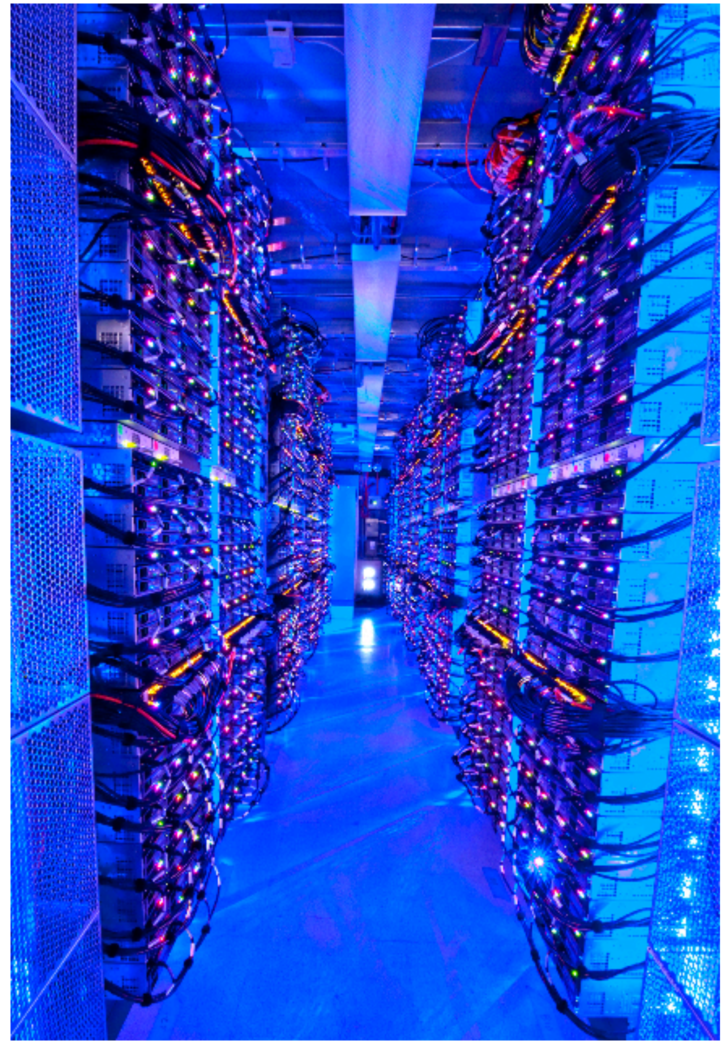
Using integrated circuits, Digital, HP… build $10k computers for labs, universities => C, UNIX OS

PC Era: Mid 1980s - Mid 2000s

Using microprocessors, IBM, Apple, … build $1k computer for 1 person => Basic, DOS, ...

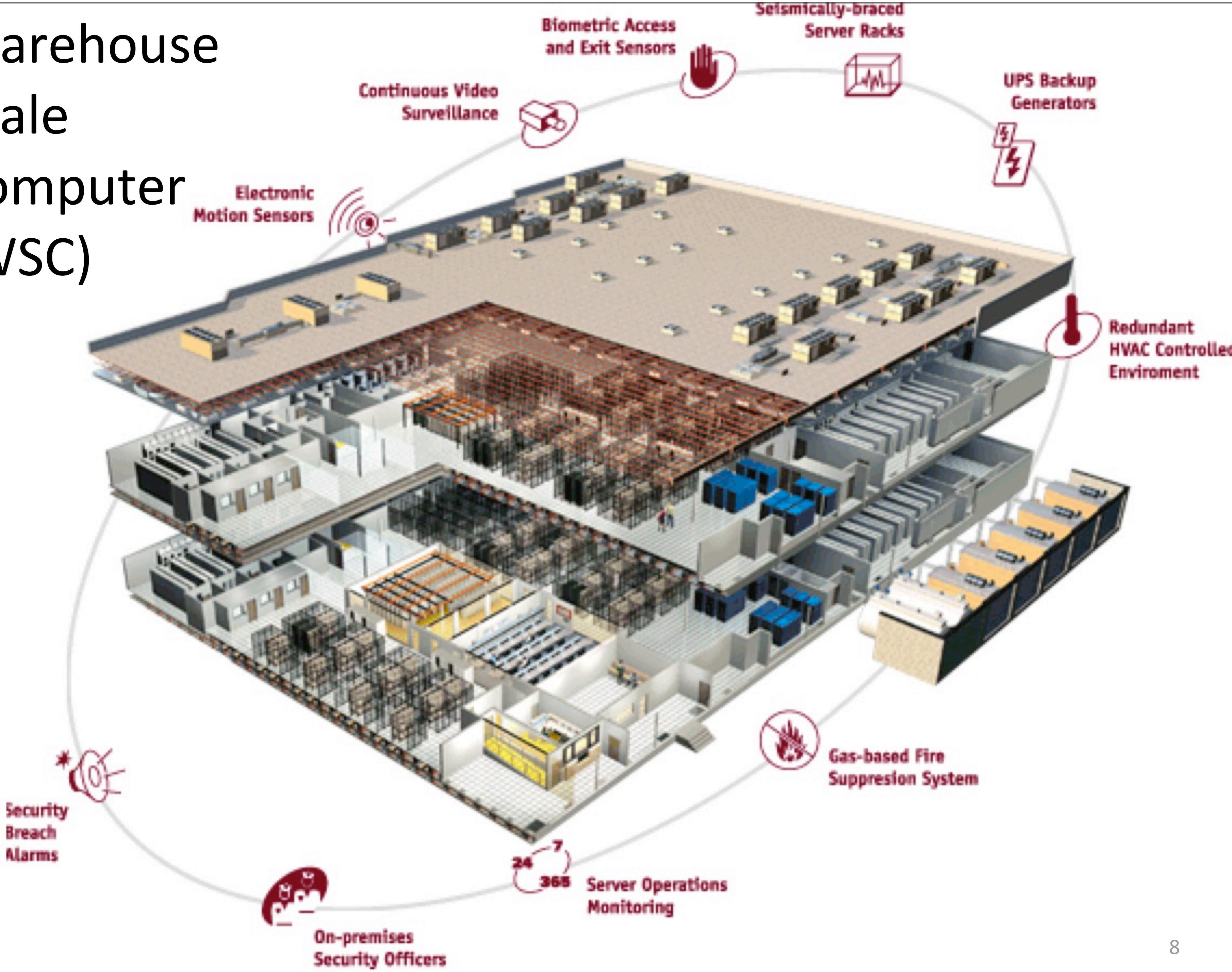
PostPC Era: Late 2000s - ??

Personal Mobile Devices (PMD): Relying on wireless networking, Apple, Nokia, … build $500 smartphone and tablet computers for individuals
=> Objective C, Android OS



Cloud Computing: Using Local Area Networks, Amazon, Google, … build $200M Warehouse Scale Computers with 100,000 servers for Internet Services for PMDs
=> MapReduce, Ruby on Rails

Warehouse Scale Computer (WSC)



Parallelism is the dominant technology trend in Cloud Computing

Software

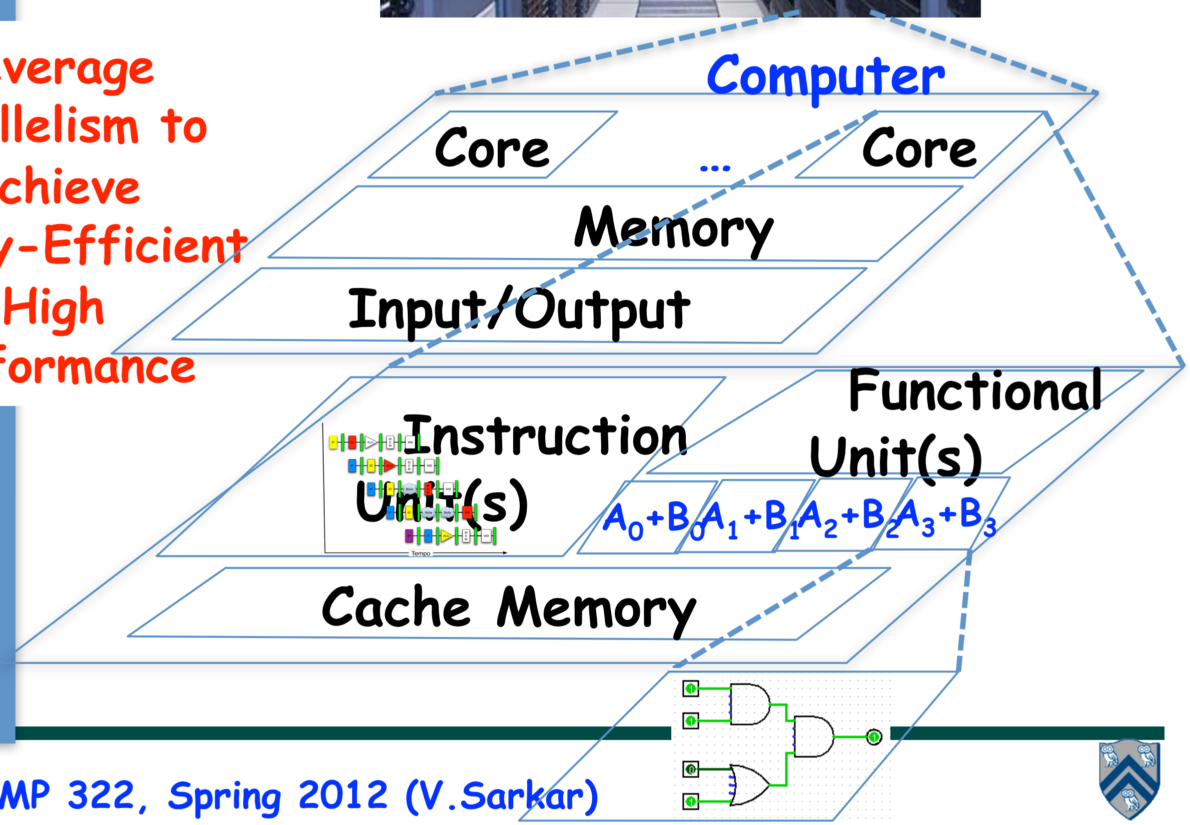
- **Parallel Requests**
  Assigned to computer
  e.g., Search "Rice Marching Owl Band"
- **Parallel Threads**
  Assigned to core
  e.g., Lookup, Ads
- **Parallel Instrs**
  >1 instruction/cycle
  e.g., 5 pipelined instructions
- **Parallel Data**
  >1 data access/cycle
  e.g., Load of 4 consecutive words

Hardware

Warehouse Scale Computer

Smart Phone

Leverage Parallelism to Achieve Energy-Efficient High Performance



Parallelism enables "Cloud Computing" as a Utility

- Offers computing, storage, communication at pennies per hour
- No premium to scale:

  1000 computers @ 1 hour
  = 1 computer @ 1000 hours

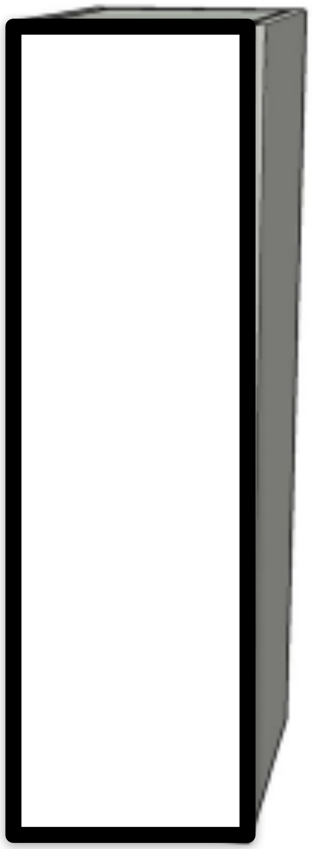
- Illusion of infinite scalability to cloud user
  - —As many computers as you can afford
- Leading examples: Amazon Web Services (AWS), Google App Engine, Microsoft Azure
  - —Economies of scale pushed down cost of largest datacenter by factors 3X to 8X
  - —Traditional datacenters utilized 10% - 20%
  - —Make profit offering pay-as-you-go use service at less than your costs for as many computers as you need
  - —Strategic capability for company's needs

2012 AWS Instances & Prices

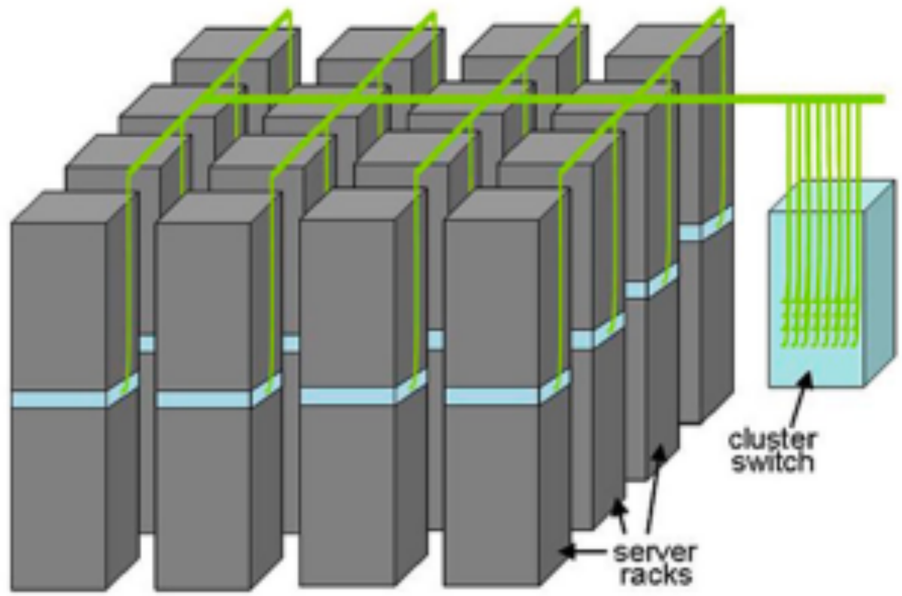
Instance	Per Hour	Ratio to Small	Compute Units	Virtual Cores	Compute Unit/ Core	Memory (GB)	Disk (GB)	Address
Standard Small	$0.08	1.0	1.0	1	1.00	1.7	160	32 bit
Standard Large	$0.34	4.0	4.0	2	2.00	7.5	850	64 bit
Standard Extra Large	$0.68	8.0	8.0	4	2.00	15.0	1690	64 bit
High-Memory Extra Large	$0.50	5.9	6.5	2	3.25	17.1	420	64 bit
High-Memory Double Extra Large	$1.20	14.1	13.0	4	3.25	34.2	850	64 bit
High-Memory Quadruple Extra	$2.40	28.2	26.0	8	3.25	68.4	1690	64 bit
High-CPU Medium	$0.17	2.0	5.0	2	2.50	1.7	350	32 bit
High-CPU Extra Large	$0.68	8.0	20.0	8	2.50	7.0	1690	64 bit
Cluster Quadruple Extra Large	$1.30	15.3	33.5	16	2.09	23.0	1690	64 bit
Eight Extra Large	$2.40	28.2	88.0	32	2.75	60.5	1690	64 bit

Equipment Inside a WSC

Server (in rack format):
$1\frac{3}{4}$ inches high "1U", x 19 inches x 16-20 inches: 8 cores, 16 GB DRAM, 4x1 TB disk



7 foot Rack: 40-80 servers + Ethernet local area network (1-10 Gbps) switch in middle ("rack switch")

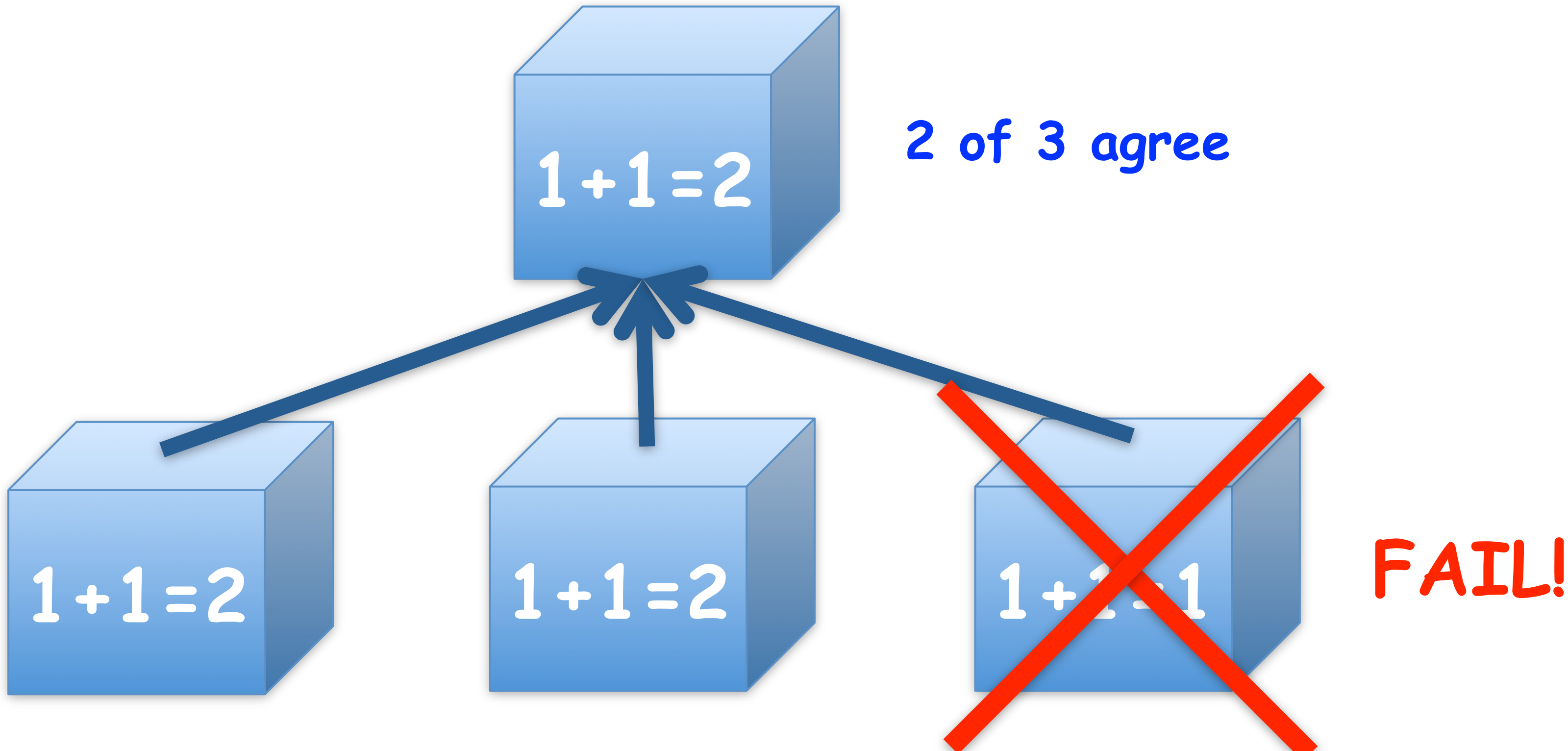


Array (aka cluster):
16-32 server racks + larger local area network switch ("array switch") 10X faster => cost 100X: cost $f(N^2)$

Server, Rack, Array

Parallelism enables Redundancy

- Redundancy so that a failing piece doesn't make the whole system fail



Increasing transistor density reduces the cost of redundancy

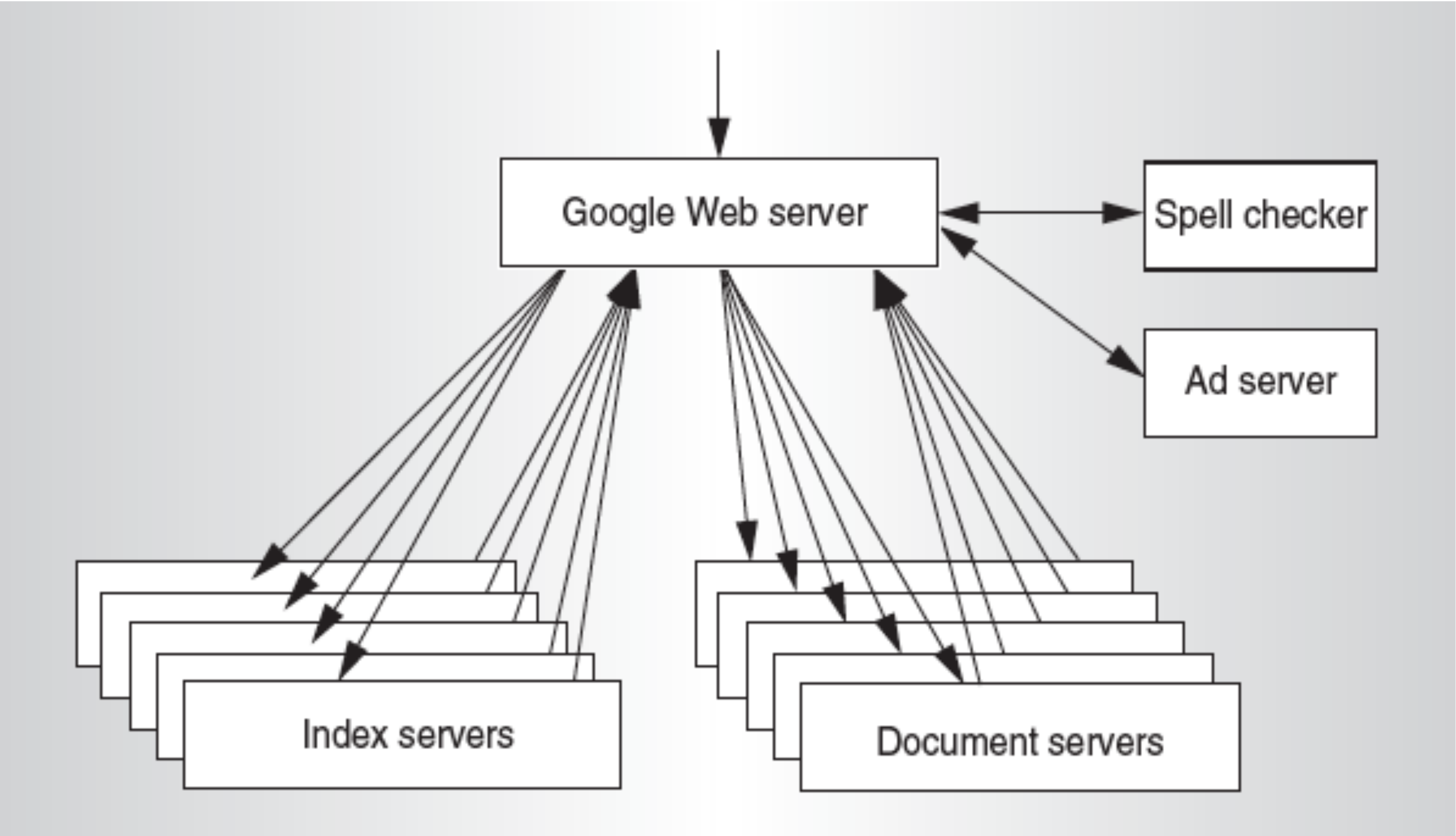
Redundancy enables Fault Tolerance and Resilience

- Applies to everything from datacenters to storage to memory
  - Redundant datacenters so that can lose 1 datacenter but Internet service stays online
  - Redundant disks so that can lose 1 disk but not lose data (Redundant Arrays of Independent Disks/RAID)
  - Redundant memory bits of so that can lose 1 bit but no data (Error Correcting Code/ECC Memory)

Request-Level Parallelism (RLP)

- Hundreds or thousands of requests per second
  - Not from your laptop or cell-phone, but from popular Internet services like Google search
  - Such requests are largely independent
    - 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 "Rice Marching Owl Band"
  1. Direct request to "closest" Google Warehouse Scale Computer
  2. Front-end load balancer directs request to one of many clusters of servers within WSC
  3. Within cluster, select one of many Google Web Servers (GWS) to handle the request and compose the response pages
  4. GWS communicates with Index Servers to find documents that contain the search words, "Rice", "Marching", "Owl", "Band". Uses location of search as well.
  5. Return document list with associated relevance score

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
  - —Indices and documents can be safely duplicated since they cannot be mutated
    - – Read-only or append-only semantics
- Different approach to distributed computing than MPI!

Outline

- **Warehouse Scale Computers and Cloud Computing**

- **<u>Map Reduce Programming Model and Runtime System</u>**

Motivation: Large Scale Data Processing

- Want to process terabytes of raw data
  - documents found by a web crawl
  - web request logs
- Produce various kinds of derived read-only/append-only data
  - inverted indices
    - e.g. mapping from words to locations in documents
  - various representations of graph structure of documents
  - summaries of number of pages crawled per host
  - most frequent queries in a given day
  - ...
- Input data is large
- Need to parallelize computation so it takes reasonable time
  - need hundreds/thousands of CPUs
- Need for fault tolerance

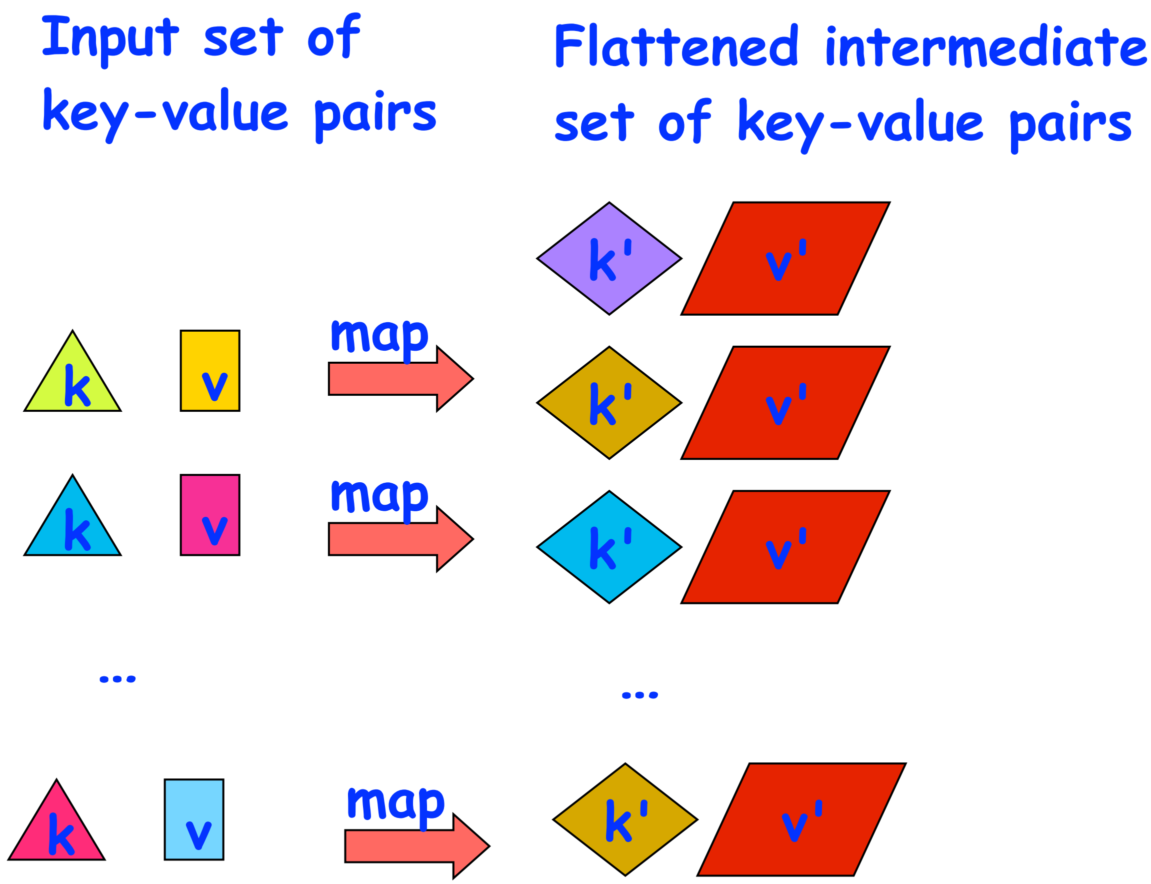
MapReduce Solution

- Apply Map function f to user supplied record of key-value pairs
- Compute set of intermediate key/value pairs
- Apply Reduce operation g to all values that share same key to combine derived data properly
  - Often produces smaller set of values
- User supplies Map and Reduce operations in functional model so that the system can parallelize them, and also re-execute them for fault tolerance

Operations on Sets of Key-Value Pairs

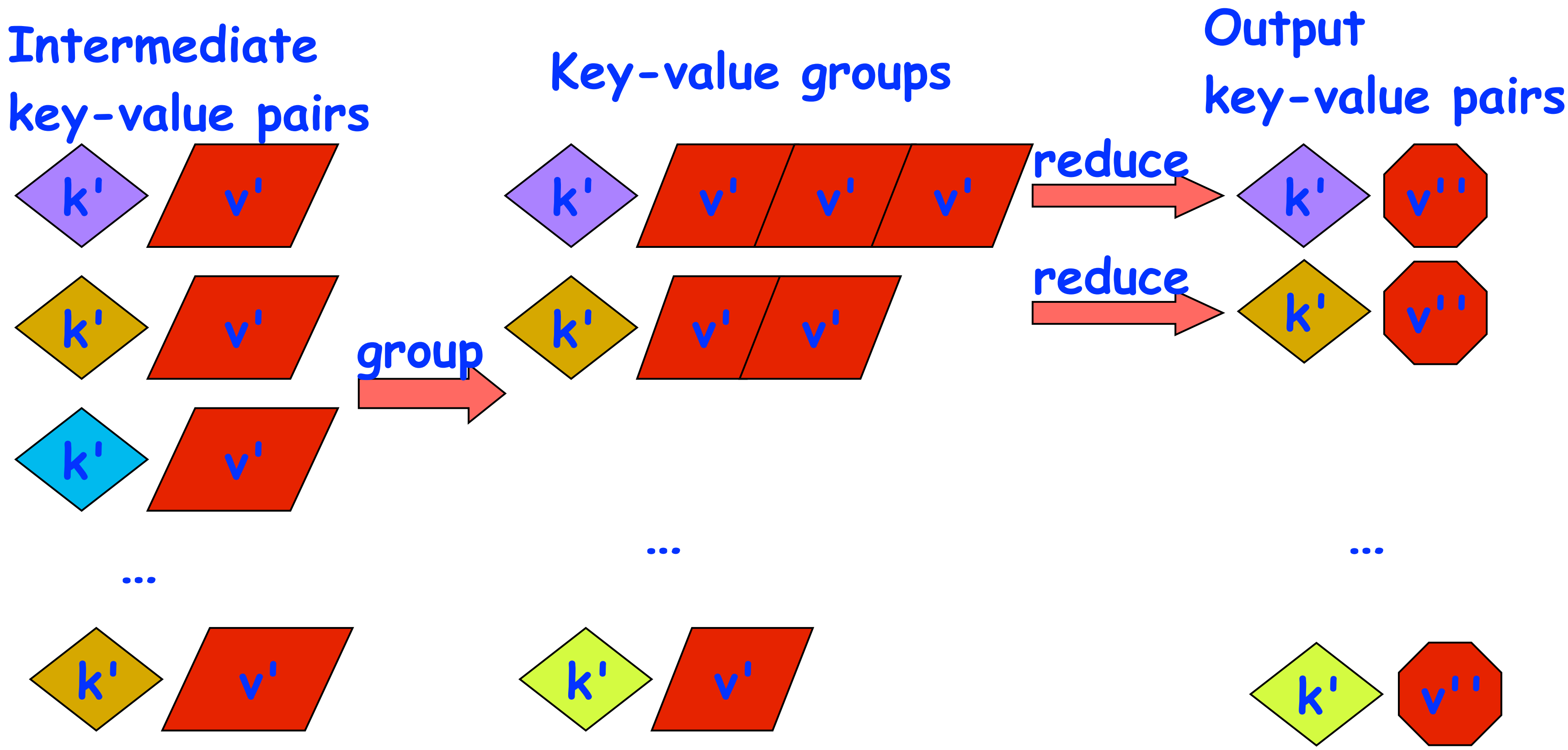
- Input set is of the form {(k1, v1), . . . (kn, vn)}, where (ki, vi) consists of a key, ki, and a value, vi.
  - Assume that the key and value objects are immutable, and that equality comparison is well defined on all key objects.
- Map function f generates sets of intermediate key-value pairs, f(ki,vi) = {(k1' ,v1'),...(km',vm')}. The kj' keys can be different from ki key in the input of the map function.
  - Assume that a flatten operation is performed as a post-pass after the map operations, so as to avoid dealing with a set of sets.
- Reduce operation groups together intermediate key-value pairs, {(k', vj')} with the same k', and generates a reduced key-value pair, (k',v''), for each such k', using reduce function g

MapReduce: The Map Step



Source: http://infolab.stanford.edu/~ullman/mining/2009/mapreduce.ppt

MapReduce: The Reduce Step



Source: http://infolab.stanford.edu/~ullman/mining/2009/mapreduce.ppt

WordCount example

Input: set of words

Output: set of (word,count) pairs

Algorithm:

1. For each input word W, emit (W, 1) as a key-value pair (map step).

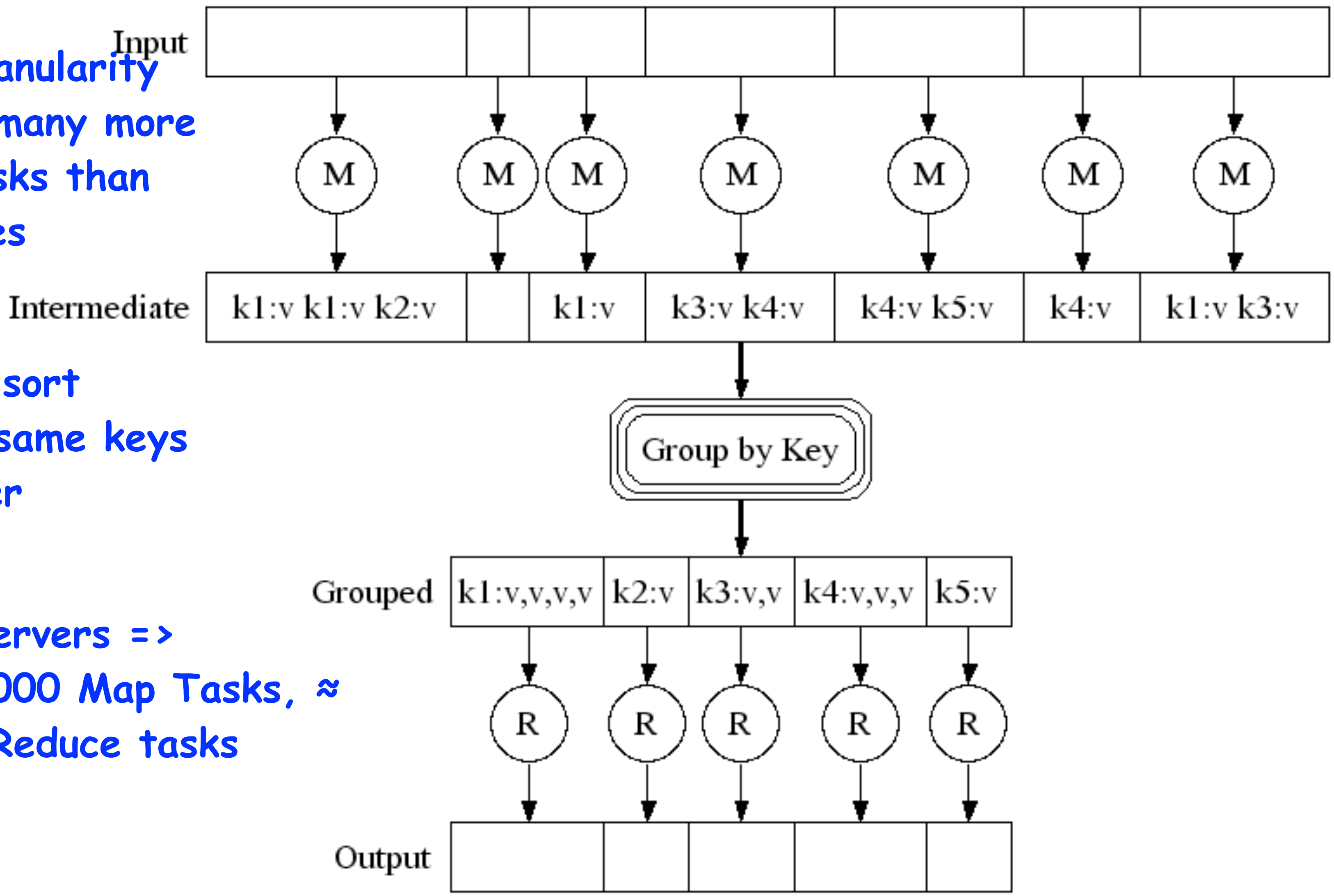
2. Group together all key-value pairs with the same key (reduce step).

3. Perform a sum reduction on all values with the same key(reduce step).

- **All map operations in step 1 can execute in parallel with only local data accesses**
- **Step 2 may involve a major reshuffle of data as all key-value pairs with the same key are grouped together.**
- **Step 3 performs a standard reduction algorithm for all values with the same key, and in parallel for different keys.**

MapReduce Execution

Fine granularity tasks: many more map tasks than machines



Bucket sort to get same keys together

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

Execution Setup

- Map invocations distributed by partitioning input data into M splits
  - —Typically 16 MB to 64 MB per piece
- Input processed in parallel on different servers
- Reduce invocations distributed by partitioning intermediate key space into R pieces
  - —E.g., hash(key) mod R
- User picks M >> no. servers, R > no. servers
  - —Big M helps with load balancing, recovery from failure
  - —One output file per R invocation, so not too many

Google Uses MapReduce For …

- Web crawl: Find outgoing links from HTML documents, aggregate by target document
- Google Search: Generating inverted index files using a compression scheme
- Google Earth: Stitching overlapping satellite images to remove seams and to select high-quality imagery
- Google Maps: Processing all road segments on Earth and render map tile images that display segments
- More than 10,000 MR programs at Google in 4 years,
  run 100,000 MR jobs per day (2008)

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