

EECS-317 Data Management and Information Processing

Lecture 16 – Little, Medium, & Big Data

Steve Tarzia

Spring 2019

Northwestern

Last Lecture: Number Representations

- Computers represent numbers with different binary encodings
- **Text** can represent decimal numbers in various formats (eg., CSV, JSON).
- **Integers** represent whole numbers
 - Remember that $2^{10} = 1024 \approx 1000$, $2^{32} \approx 4 \text{ billion}$
 - Signed integers use two's complement
 - Used for *counting* and *identifying* records.
- **Fixed point** adds an implicit radix point to an integer.
 - Allows representing fractional quantities as integers, but with limited range.
 - Used for numbers that *should round off*, like prices.
- **Floating point** is a binary scientific notation representation
 - Can represent tiny fractional values and huge values with equal precision
 - **Single precision** ≈ 7 decimal digits, **Double precision** ≈ 16 decimal digits of precision
 - Used for *measurements* and *calculations*.
 - Float subtraction can lead to *catastrophic cancellation*.

Truly “Big Data” requires many machines

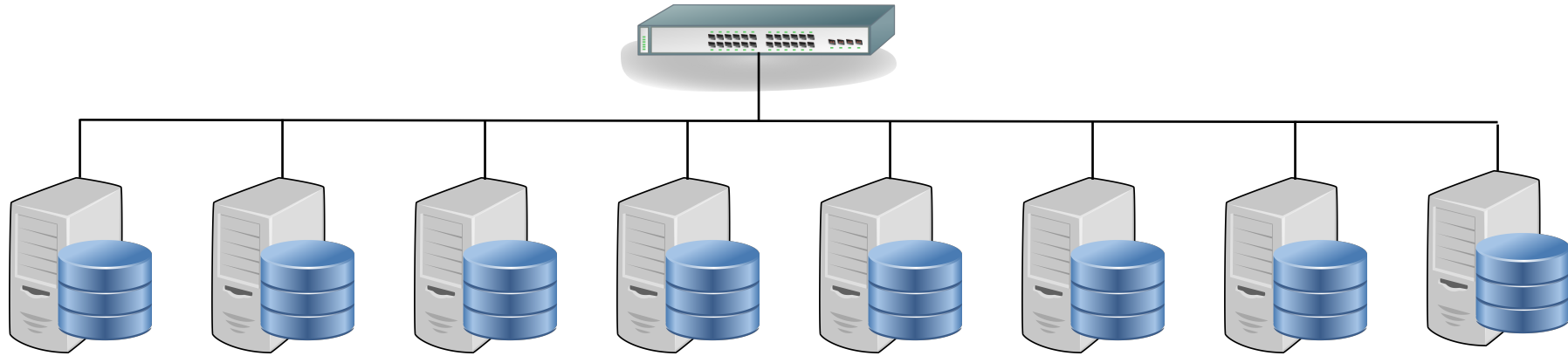
Why not just add more disks to one machine?

- Each CPU has limited *input/output* capacity
 - Because there are only a few dozen I/O “pins” (wires) on the CPU.
- Each CPU has limited *memory* for fast access to data
- Each CPU has limited *processing* (arithmetic) capacity

However, coordinating multiple machines is very difficult.

Distributing a database

- Create a “cluster” of computers connected to each other.
- Each “node” in the cluster stores a fraction of the data set.



- Distributed database examples:
 - MongoDB, Cassandra, Amazon DynamoDB, ...
- Distributed filesystems also use the same basic idea:
 - Hadoop HDFS, Google File System (Colossus, BigTable), Amazon S3, ...

Challenges in distributed DBs

- Balancing storage and processing loads
- Finding data (on which node is it?)
 - Must be careful to involve only a few nodes in each query.
 - If all nodes are involved in every query, then we gained nothing by adding more machines! (Every node would be effectively handling the full load.)
- Analyses that combine data or process all of the data must be scalable
 - No one node can work on all the data
- Fault tolerance
 - If we have dozens or hundreds of nodes, some are bound to fail
- Consistency
 - The different nodes cannot provide contradictory information

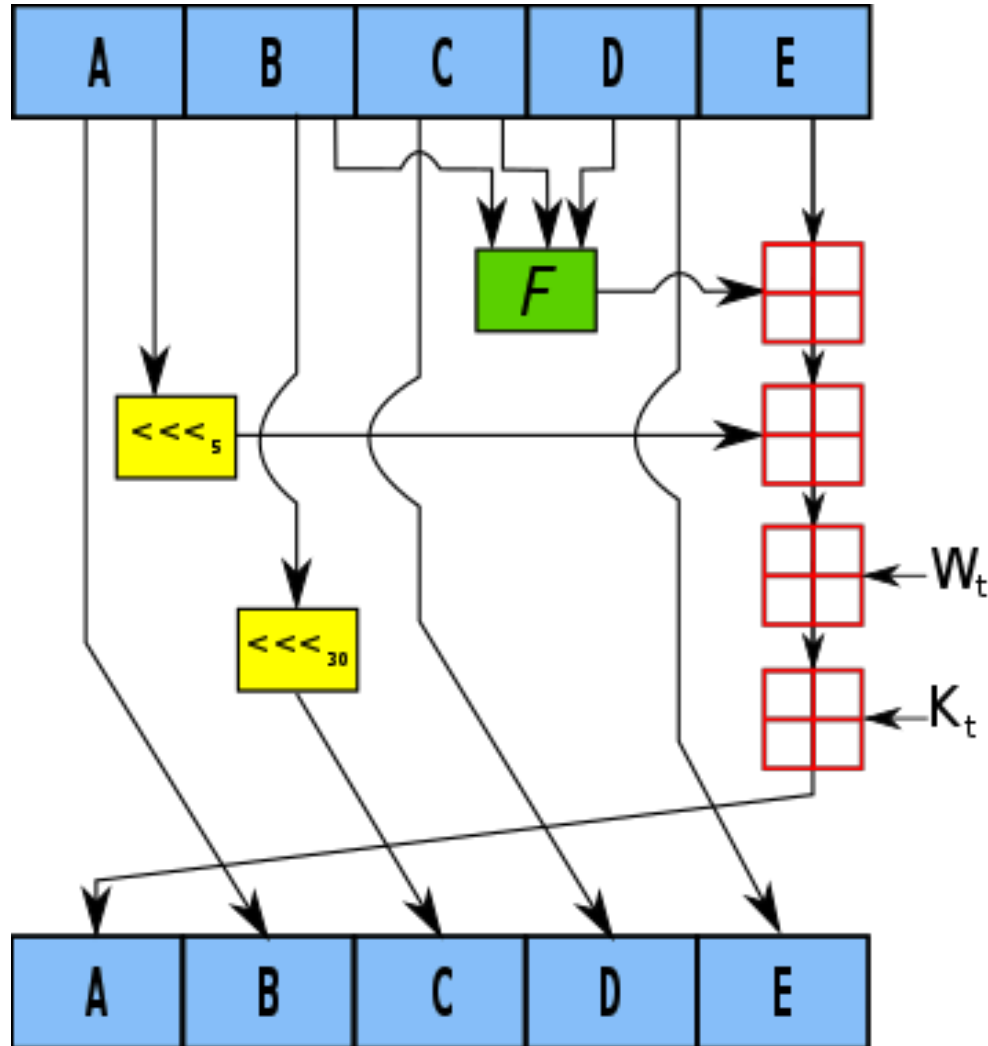
Hashing is the basis of distributed DBs

- A *hash* is an algorithm that takes a value and returns a pseudo-random value derived from it.
- It's a *constant* but *unpredictable* mapping
 - A long sequence of arithmetic operations
- MD5 is a standard hash function:
 - "Steve" → f6e997429bf8cb7b3b98b310a9f7ca30
 - "steve" → 2666b87c682f5072f62bab0955d485ce
 - "Janice" → 3837607db4754c036425cb1b2a7c8766
 - "1" → b026324c6904b2a9cb4b88d6d61c81d1
 - "Steve" → f6e997429bf8cb7b3b98b310a9f7ca30
 - tale_of_two_cities.txt (806,878 characters)
_ _ → 3ab56b74562a714a5638f94446581977
- The same input always gives the same output
- Length of the input can vary, but output has fixed length

We can define all kinds of hash functions

- If we are dealing with text, and we want to map to a number 0-99, any of these are possible hash functions:
 - Use the length of the string (modulo 100)
 - Use the ASCII encoding of the first letter in the text (modulo 100)
 - Count the ones in the binary representation of the text (modulo 100)
 - Multiply the length of the string by the ASCII encoding of the first letter in the text and then subtract the ASCII encoding of the last letter in the text (modulo 100)
- These all produce a number in the range 0-99 and they always give the same result for a given input, but the output is not well-balanced.
 - Some numbers may be output much more frequently than others.
- Standard hashes like MD5 and SHA-1 are very carefully designed

SHA-1 hash function illustration



One iteration within the SHA-1 compression function:

A, B, C, D and E are 32-bit **words** of the state;
 F is a nonlinear function that varies;


\lll_n denotes a left bit rotation by n places;
 n varies for each operation;

W_t is the expanded message word of round t ;

K_t is the round constant of round t ;

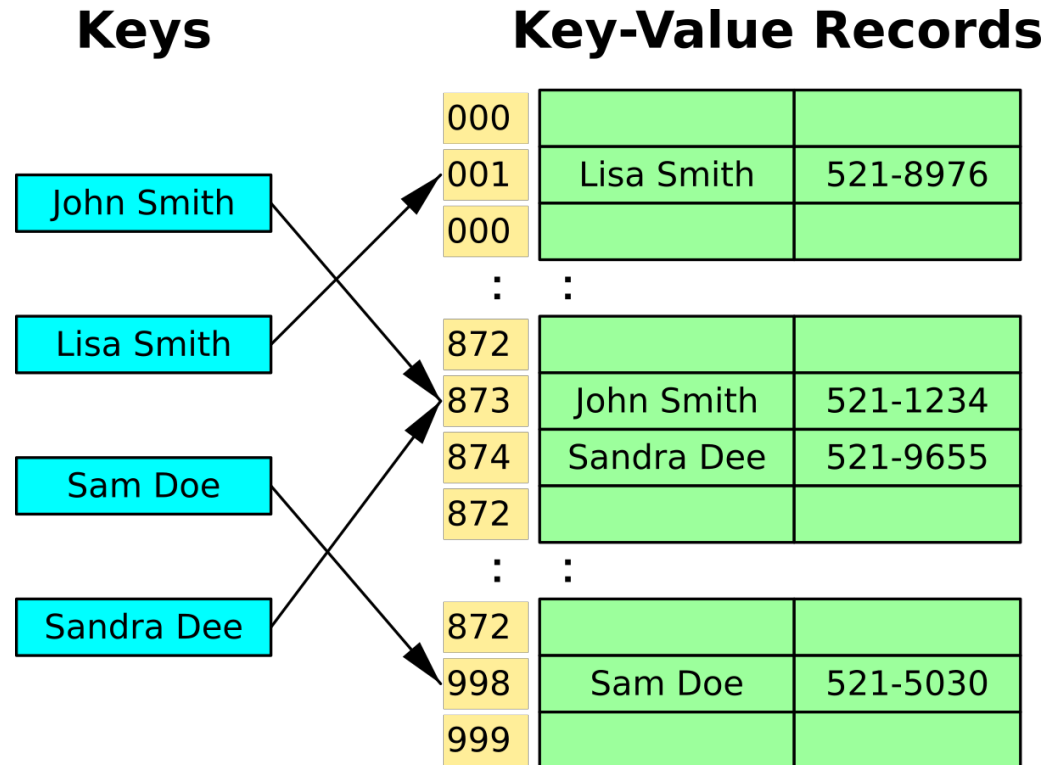
\boxplus denotes addition modulo 2^{32} .

Hash Table

- Stores (*key*, *value*) pairs
 - This abstract data type is called a *dictionary*, or *map*.
 - For example:
 - A word and its definition.
 - “word” → “a single distinct meaningful element of speech or writing, ...”
 - “hash” → “a dish of cooked meat cut into small pieces and cooked again, ...”
 - A database table’s primary key and the rest of the columns in the row:
 - StaffID → [StfFirstName, StfLastName, StfStreetAddress, StfCity, StfState, ...]
 - 98005 → [“Suzanne”, “Viescas”, “15127 NE 24th, #383”, ...]
 - 98007 → [“Gary”, “Hallmark”, “Route 2, Box 203B”, ...]
- 
- key* *value*

Hash Table mechanics

- *Hash the key* to determine the address where the value is stored

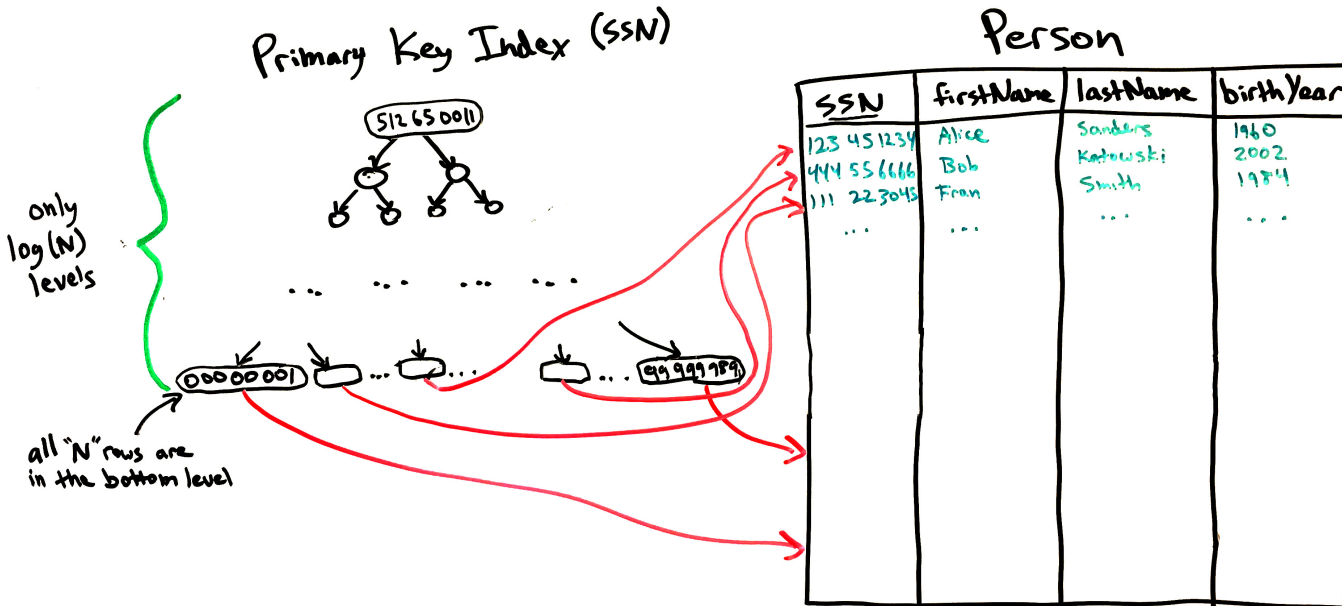


- If the address is already filled, then use the next open slot
 - This is called a *collision* and there are other strategies besides “linear probing”

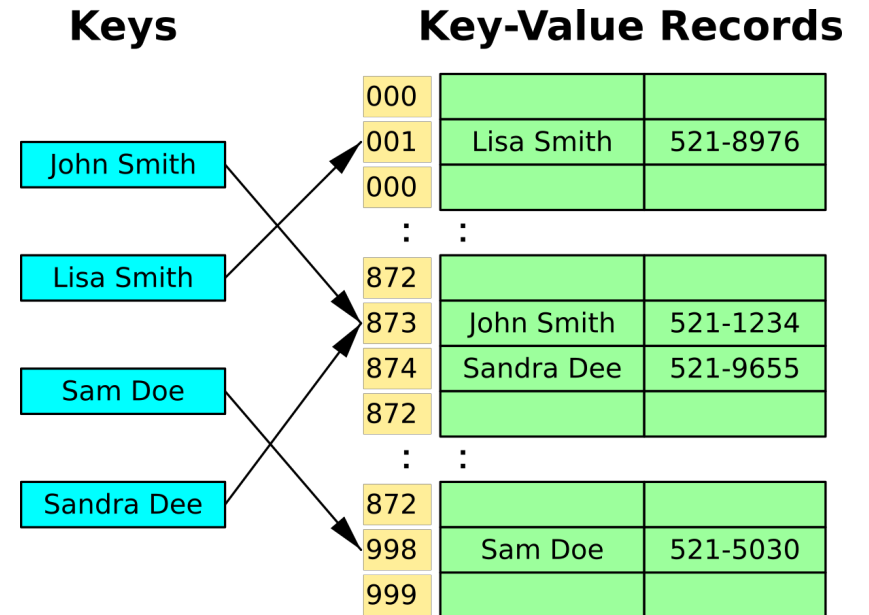
Hash Indexes in SQL databases

- A hash table is an alternative to a binary search tree
 - It lets you find the data in one step!
 - However, it does not support range queries.
 - Data is randomly scattered

Tree-based table index

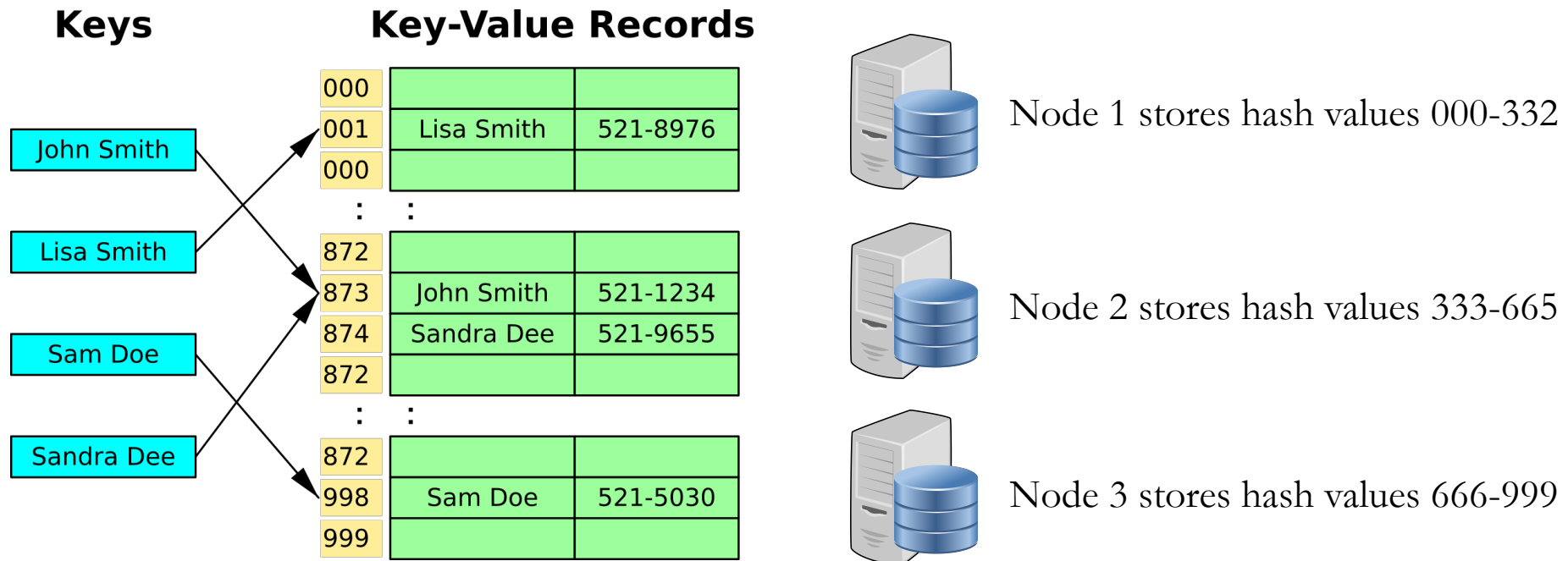


Hash-based table index



Distributed Hash Table

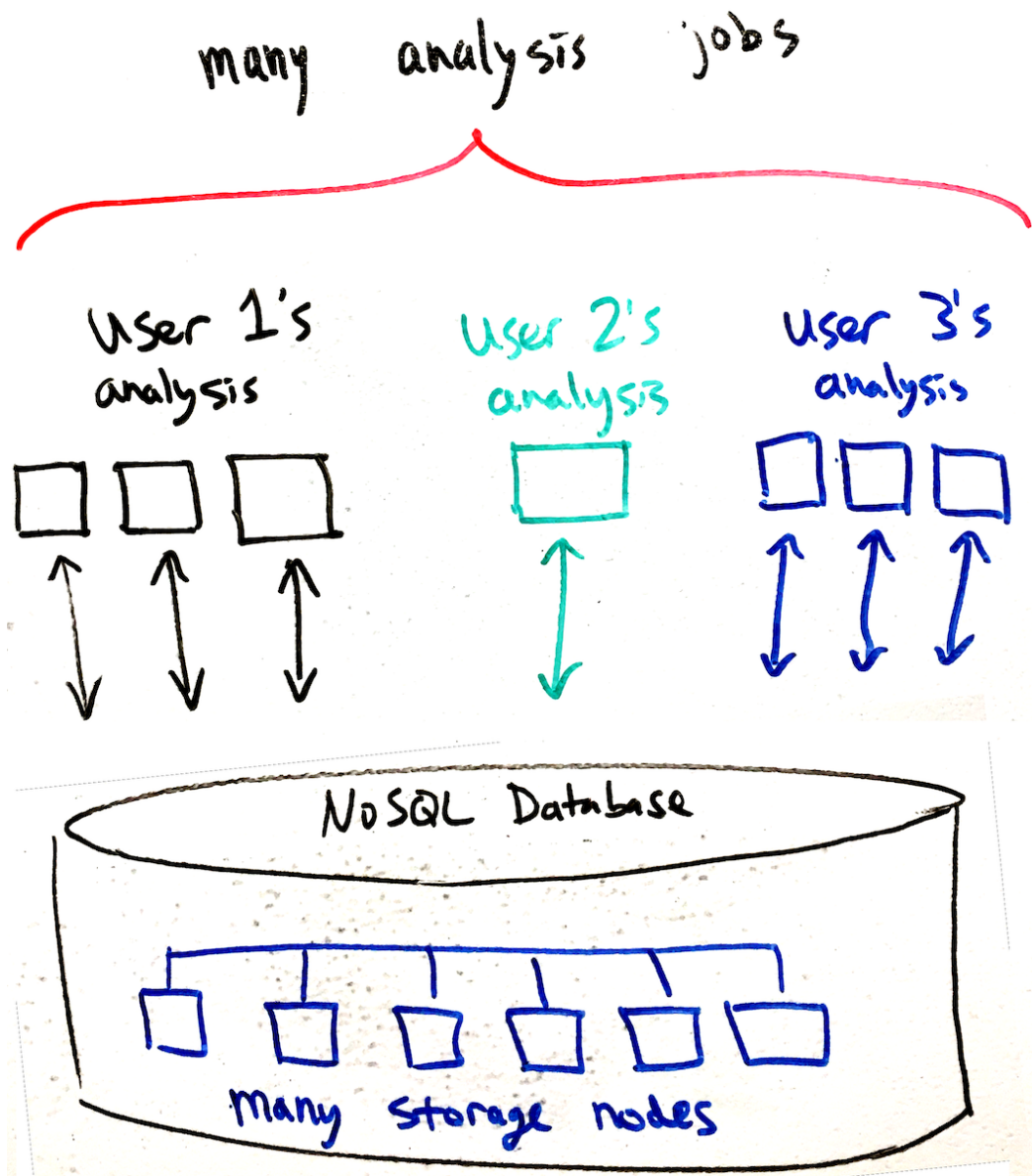
- Each cluster node is responsible for a range of hash values
- Client software can compute the key hash to determine which node to query for the data:



NoSQL databases

- NoSQL databases are distributed key-value stores
- Like one big table with just a primary key
- They have a map/dictionary interface, do not support SQL queries.
 - You can only:
 - *get* a value for a key.
 - *put* a value for a key.
 - Each operation only affects the node(s) storing that key
 - Very **scalable!** (can grow large without slowing down)
- If we wanted to support full SQL, JOINS would have to pull data from many nodes in the cluster and performance would be slow.

Scalable analysis with a NoSQL database



- A distributed database can handle many concurrent queries.
- A single analysis can be broken into multiple “workers” that run on different CPU cores or computers.
- Similarly, a high-scale website or app can be built on top of a NoSQL database, with many frontend webservers/apps fetching and storing data at once.

NoSQL fault tolerance

- Because there can be hundreds of nodes, NoSQL databases are designed to tolerate node failure. (Failures are more likely in a larger population.)
- Each key-value pair must be stored on multiple nodes
- However, data replication introduces *consistency* problems
 - If two nodes report different values, then which do we use?
- Common solution is to hash each key-value pair to three nodes
 - Use the “majority opinion” when reading.
 - Consider a write as successful if at least two nodes succeed at storing it.
- NoSQL database clusters are redundant at the software level
 - Can use inexpensive, unreliable servers because system tolerates node deaths.
- SQL database servers are redundant at the hardware level.
 - Use an expensive server: redundant power supplies, fans, disks, battery backup, etc.

Variety of data processing tools

Data processing tasks vary, so a variety of tools exist to handle different:

- Data size
- Data persistence (the need to reuse data)
- Structured/unstructured data.
- Numerical/textual/file data
- Types of calculations to be done.

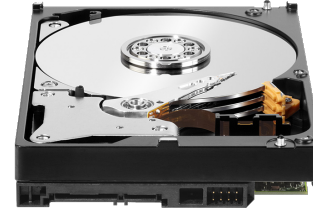
Data Size

- **Little** data – can fit in a computer’s memory (RAM)
 - Up to a few **gigabytes** (billion bytes)
- **Medium** data – can fit in one computer’s filesystem (disks).
 - Up to a few **terabytes** (trillion bytes)
- **Big** data – cannot be stored or processed on one computer.
 - 10s of terabytes, petabytes (10^{15}), even exabytes (10^{18})!



RAM capacity:

- 8 GB on a typical laptop
- 1.5 TB on a good server



Disk capacity:

- 256 GB on a typical laptop
- 12 TB on a good desktop



Cluster capacity:

- Up to thousands of computers can be connected.
- Capacity is **unlimited**.

Little Data tools

- Matlab
- General-purpose programming languages:
 - R, Python, C++ etc.
 - Program variables and data structures are stored in the computer's memory.
 - For example, lists and dictionaries are stored in memory.
 - Filesystem access is not truly built-into the language.
 - However, programmer can call functions to read/write data from disk.
 - Data from files or databases must be copied to in-memory data structures to gain access.
- Excel (spreadsheet is saved to a file, but it's fully in RAM while open)

Medium Data tools

These allow data to remain on disk during analysis.

- SQL relational databases
 - Users create tables, define indexes, load data, and write queries.
 - DBMS (the database server software) handles all the details:
 - Tables are stored on disk.
 - Queries are answered by pulling the appropriate data from disk.
 - SQL databases can be used within tools like Matlab, Python, R, etc. to make the analysis more easily handle lots of data.

Other “medium data” tools

“Do it yourself”

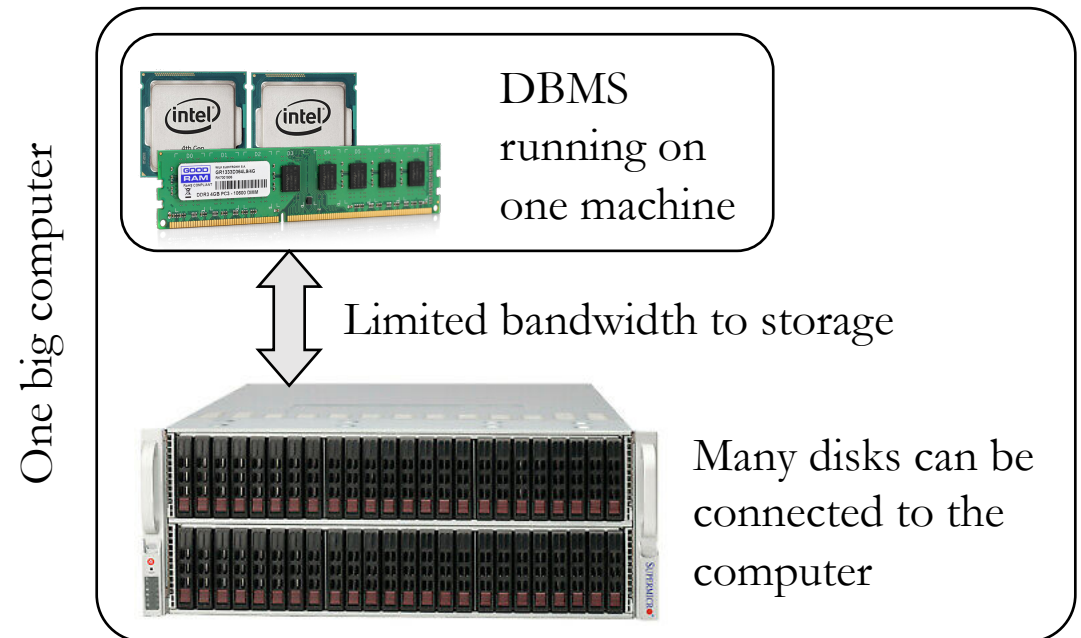
- Streaming
 - Open a file and read through it.
 - Don’t load it all in memory, just examine each small piece and move on.
 - Useful for very simple types of processing, like keyword searching or a single aggregation (like looking for the max value).
 - For example, `grep` command for searching text files.
- File-based indexing
 - Use different folders/filenames to organize data in one dimension.

Commercial

- SAS/STAT
 - A statistical analysis software package that can work with data on disk.
 - Internally, it works like a SQL DBMS with one table.

Large database servers

- A single hard disk has limited capacity (~ 12 TB), and limited throughput (~ 150 MB/s).
- However, you can actually connect 10s or 100s of disks to a single computer (especially for a DB server).
 - Use RAID *storage arrays*.
- Capacity is practically unlimited, but a single computer has:
 - Limited compute power.
 - Limited I/O bandwidth (theoretical max of ~ 80 GB/s on PCI-express v4).



“Medium Data” tools are actually pretty scalable

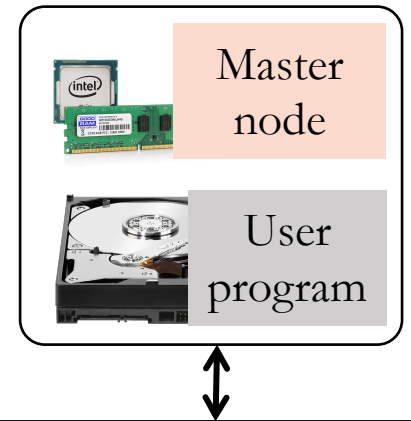
- Stack Overflow database (150GB) is stored on our MySQL server.
- After adding all the appropriate indexes, analysis on the whole data set can be done very quickly with SQL.
- Amazon uses an Oracle SQL database for product information.
 - However, they handle shopping carts with their NoSQL database (Dynamo).
- However, sometimes you need to work faster or support many simultaneous analyses, and this is where “Big Data” tools are useful.

“Big Data” problems

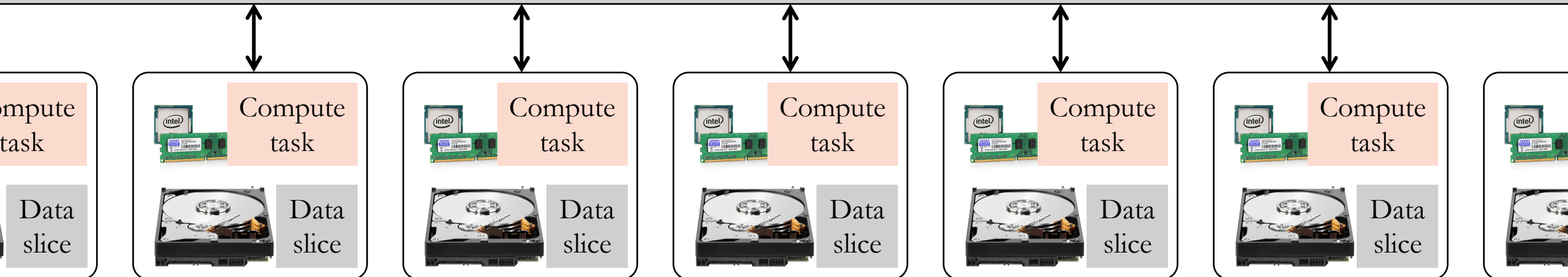
- Let's say you need to run analyses of 10 petabytes of data each day:
 - Theoretically, a single computer can do this in no less than:
 $10^{15} \text{ bytes} / (80 * 10^9 \text{ bytes/sec}) = 12,500 \text{ seconds} = \mathbf{3.5 \text{ hours}}$.
 - However, in practice you may find the performance is much slower than this.
- To speed up an analysis you can use many machines to work on the problem in **parallel**.
 - The analysis is somehow split into pieces (subproblems)
 - Each machine works on subproblems that contribute to the final answer.

Distributed computing

- Data & analysis are distributed across many machines.
- No one machine has *direct* access to all the data.
- Machines must communicate over a network to share data and intermediate results.
- We need some special way to define these analyses.

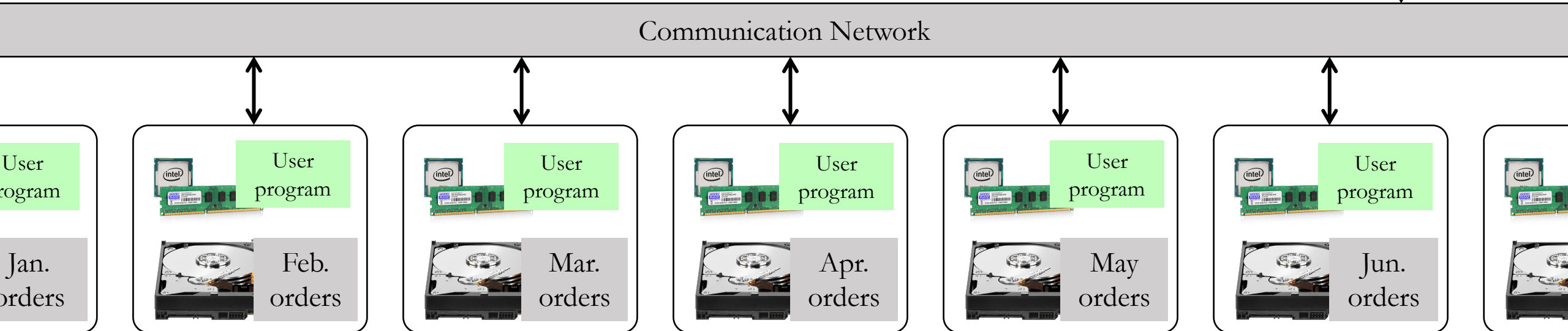
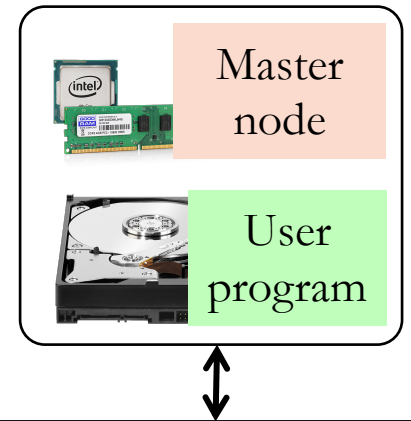


Communication Network



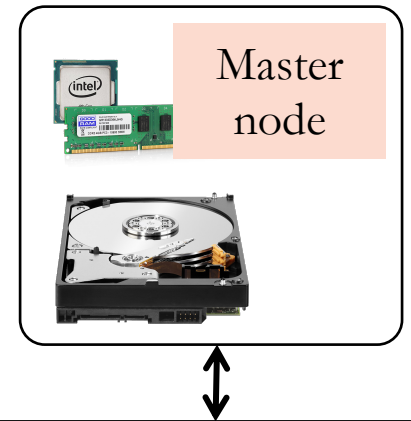
Example: Find the largest sales order

1. Distribute instructions to all the machines (nodes).
2. Each node computes its own maximum.
3. Results are sent to a single node.
4. Maximum of the maximums is calculated.

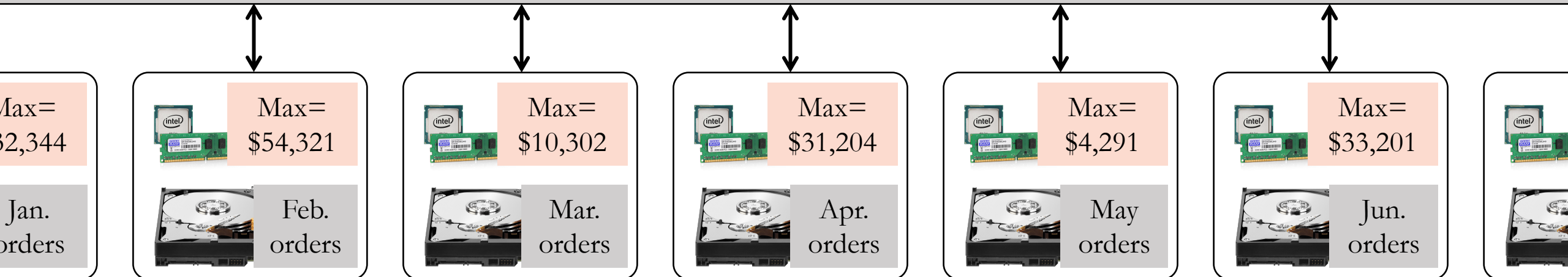


Example: Find the largest sales order

1. Distribute instructions to all the machines (nodes).
2. Each node computes its own maximum.
3. Results are sent to a single node.
4. Maximum of the maximums is calculated.

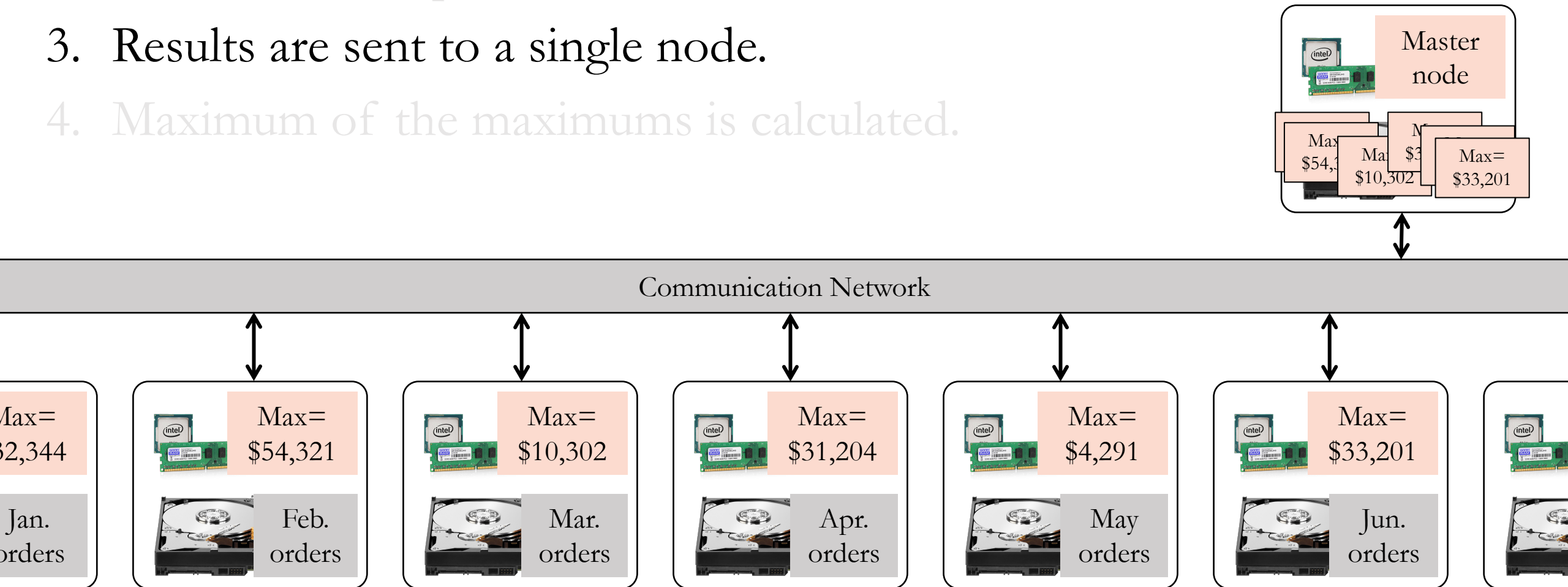


Communication Network



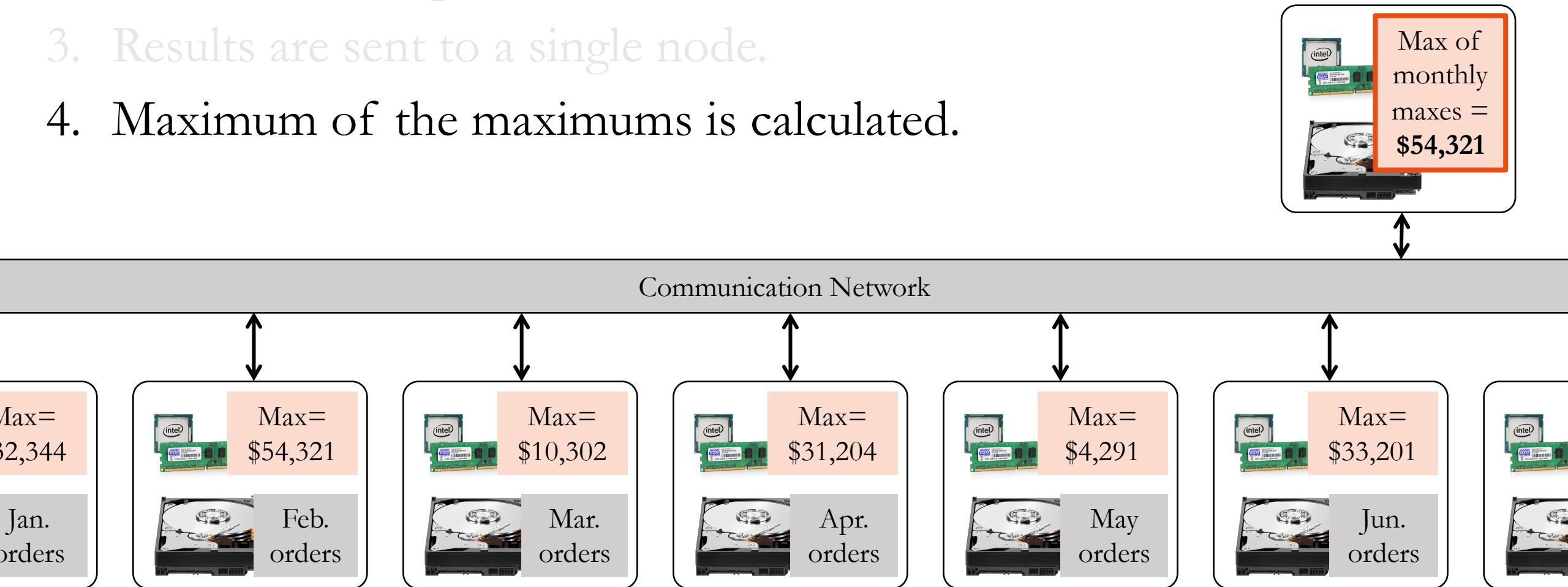
Example: Find the largest sales order

1. Distribute instructions to all the machines (nodes).
2. Each node computes its own maximum.
3. Results are sent to a single node.
4. Maximum of the maximums is calculated.



Example: Find the largest sales order

1. Distribute instructions to all the machines (nodes).
2. Each node computes its own maximum.
3. Results are sent to a single node.
4. Maximum of the maximums is calculated.

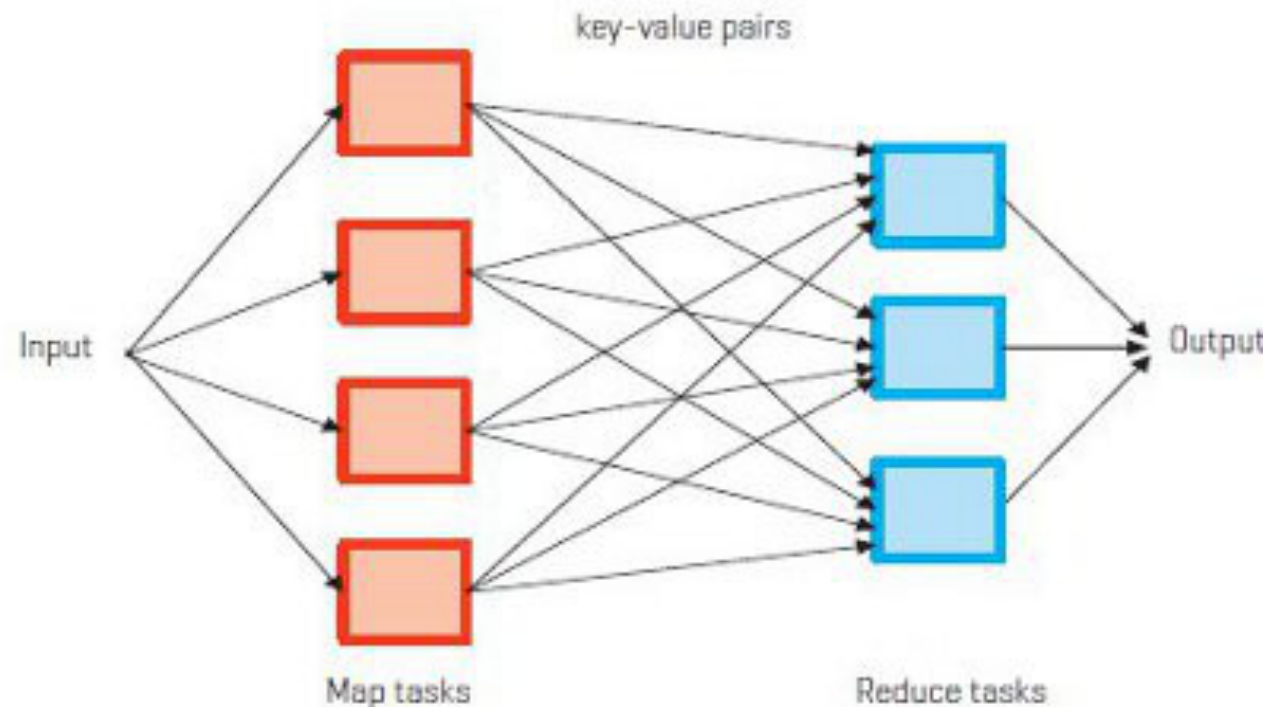


Hadoop is a distributed computing platform

- It's a type of computing cluster for storing data and running analyses.
- Main components are:
 - **HDFS** – *Hadoop Distributed File System* for storing the data to be analyzed and also for storing intermediate and final results.
 - Data is distributed across all the nodes (machines) in the cluster.
 - HDFS is closely coupled to the analysis engine.
 - **MapReduce** – a programming model for defining the parallel analysis steps.
 - Analysis is usually written in Java, but other languages are possible (Python, R, etc.)
 - Analysis is defined in terms of two main operations – *Map* and *Reduce*.
 - Based on a 2004 Google article: <https://ai.google/research/pubs/pub62>

MapReduce

- A MapReduce analysis has to follow a certain pattern.
- First, **Map** does something, producing many intermediate results.
- Later, **Reduce** aggregates the intermediate results.

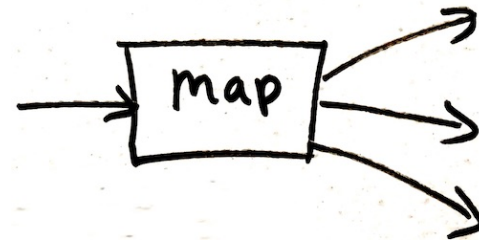


MapReduce programming model

- Input & Output: each a set of key/value pairs
- Programmer's task is to define just two functions that will complete the work in parallel:

map(in_key, in_value) -> list(out_key, intermediate_value)

- Processes input key/value pair
- Produces a **set** of intermediate pairs



reduce(out_key, list(intermediate_value)) -> list(out_value)

- Combines all intermediate values for a particular key
- Produces a set of merged output values (usually just one)

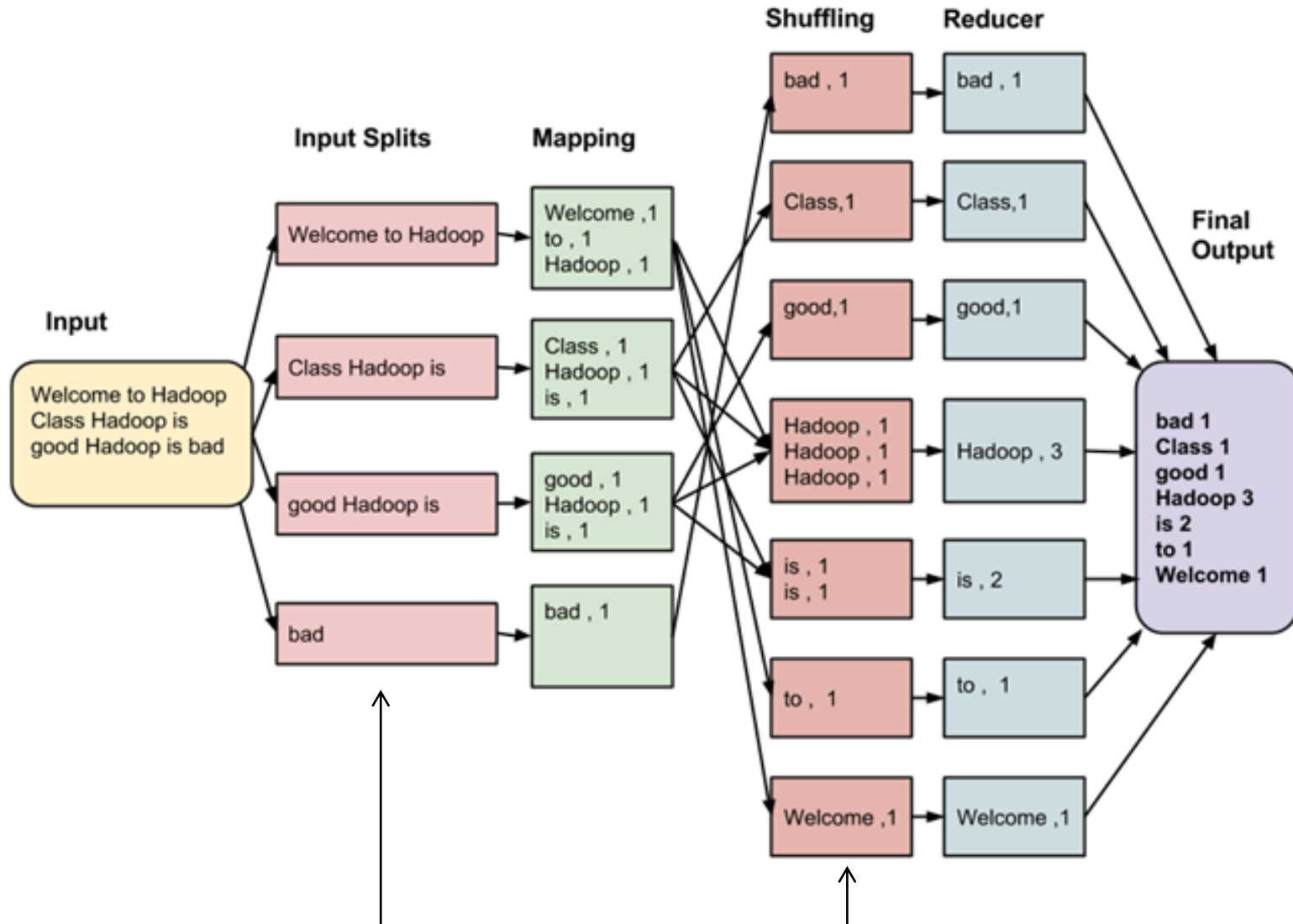


MapReduce example: counting words

```
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");
```

```
reduce(String output_key, List intermediate_values) :  
    // output_key: a word  
    // output_values: a list of counts  
    int result = 0;  
    for each v in intermediate_values:  
        result += v;  
    Emit(result);
```


Counting words example



Splitting and shuffling steps are done automatically by the Hadoop runtime system

Beware of the *Hype!*

- Hadoop is designed for huge scale, but it's inherently slower than traditional approaches: <https://dl.acm.org/citation.cfm?id=1559865>
- It lacks indexes, has a lot of performance overhead.
- Requires a special environment (a Hadoop cluster).
- Don't use Hadoop/MapReduce or NoSQL unless you really need to!
- Probably 75% of Hadoop/NoSQL usage would be better served with a different tool (usually a SQL database).

Recap: NoSQL DBs and Hadoop

- **Hash functions** map deterministically to pseudo-random values.
- A hash table is a data structure to store (*key, value*) pairs:
 - Hashing the key determines the data's storage address.
 - So, hashing is an alternative to binary search trees for indexing data.
- **Big Data** requires databases (*or filesystems*) distributed over many computers
 - NoSQL databases are distributed hash tables. Lack the full power of SQL.
- **Hadoop** is a distributed computing platform, using MapReduce:
 - **Map** function takes input and produces a bunch of intermediate results.
 - **Reduce** function takes intermediate results and produces an output.
- Stop and think before using “big data” tools (SQL is pretty scalable!)