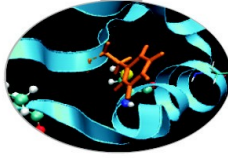


Welcome!

Master BBS in Data Science
Giuseppe Fiameni
March 16th 2016

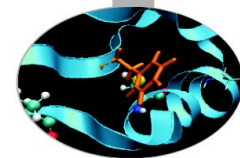
Goals



During this course, you will learn:

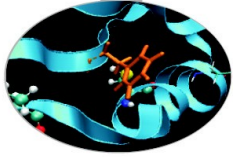
- the trends and challenges surrounding the BigData definition
- how the most relevant technologies and methods in this domain work
 - *Apache Hadoop*
 - *Map-Reduce*
 - *Spark, SparkSQL*
 - *MLLIB*
- how to structure and program your code using Python

Today's agenda



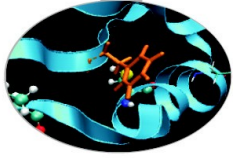
Time	Descrizione	Insegnante
9:30-10:30	Parallel Data Processing in HPC	Fiameni
10:30 - 12:30	Parallel Data Processing with Hadoop in HPC	Fiameni
12:30 – 13.30	Esercitazioni MapReduce in Python	D'Onorio De Meo/Fiameni
14:30 – 17:30	Esercitazioni MapReduce in Python	D'Onorio De Meo/Fiameni
17:30 – 18:30	Breve introduzione Python Numpy/Pandas	D'Onorio De Meo

Materials

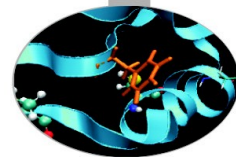


<https://goo.gl/N77tEy>

Quick poll

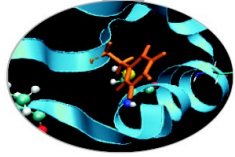


- How many of you already know part of the mentioned Big Data technologies?
- How many of you know any HPC methods (MPI, OpenMP, etc.)?
- How many of you know Python?
- Scala?



Before starting

Size of computational applications



Computational Dimension:

number of operations needed to solve the problem, in general is a function of the size of the involved data structures (n , n^2 , n^3 , $n \log n$, etc.)

flop - Floating point operations

indicates an arithmetic floating point operation.

flop/s - Floating points operations per second

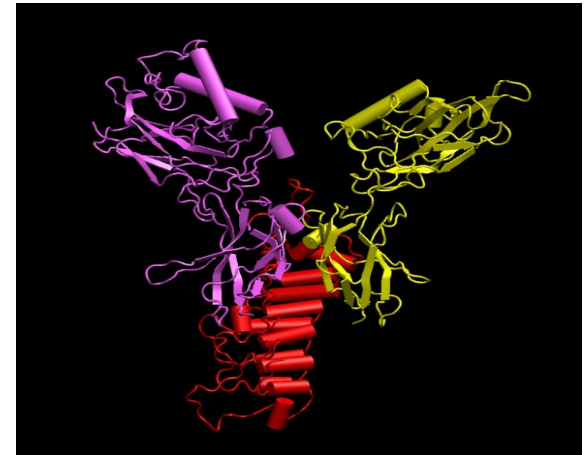
is a unit to measure the speed of a computer.

computational problems today:

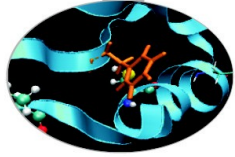
10^{15} - 10^{22} flop

One year has about 3×10^7 seconds!

Most powerful computers today have reach a sustained performance is of the order of Tflop/s - Pflop/s (10^{12} - 10^{15} flop/s).



Example: Weather Prediction



Forecasts on a global scale (**.....too accurate and inefficient!!**)

- **3D Grid to represent the Earth**

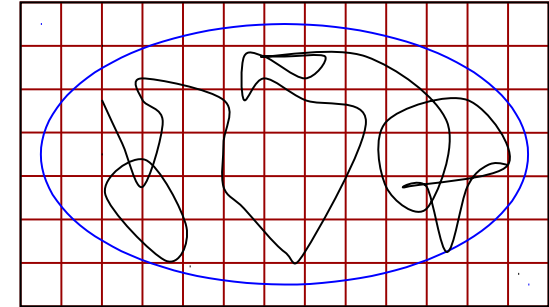
- Earth's circumference: $\cong 40000$ km
- radius: $\cong 6370$ km
- Earth's surface: $\cong 4\pi r^2 \cong 5 \cdot 10^8$ km²

- **6 variables:**

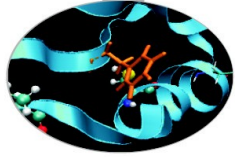
- temperature
- pressure
- humidity
- wind speed in the 3 Cartesian directions

- **cells of 1 km on each side**

- 100 slices to see how the variables evolve on the different levels of the atmosphere
- a 30 seconds time step is required for the simulation with such resolution
- Each cell requires about 1000 operations per time step (Navier-Stokes turbulence and various phenomena)



Example: Weather Prediction / 1



Grid: $5 \cdot 10^8 \cdot 100 = 5 \cdot 10^{10}$ cells

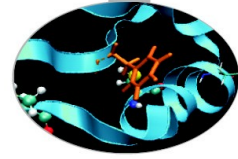
- each cell is represented with 8 Byte
- Memory space:
 - $(6 \text{ var}) \cdot (8 \text{ Byte}) \cdot (5 \cdot 10^{10} \text{ cells}) \cong 2 \cdot 10^{12} \text{ Byte} = 2\text{TB}$

A 24 hours forecast needs:

- $24 \cdot 60 \cdot 2 \cong 3 \cdot 10^3$ time-step
- $(5 \cdot 10^{10} \text{ cells}) \cdot (10^3 \text{ oper.}) \cdot (3 \cdot 10^3 \text{ time-steps}) = 1.5 \cdot 10^{17} \text{ operations !}$

A computer with a power of 1Tflop/s will take $1.5 \cdot 10^5$ sec.

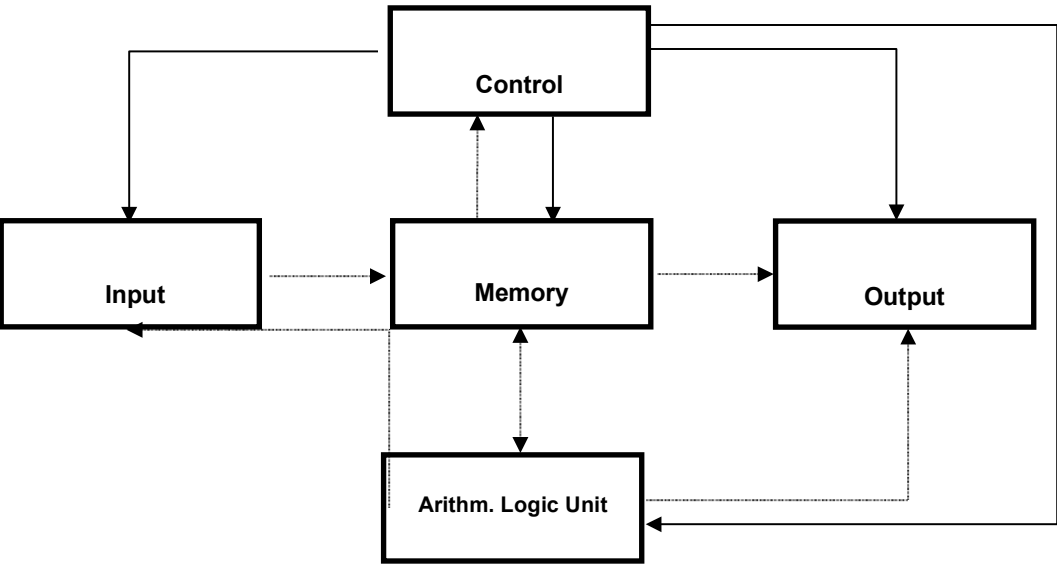
- 24 hours forecast will need 2days to run ... but we shall obtain a very accurate forecast



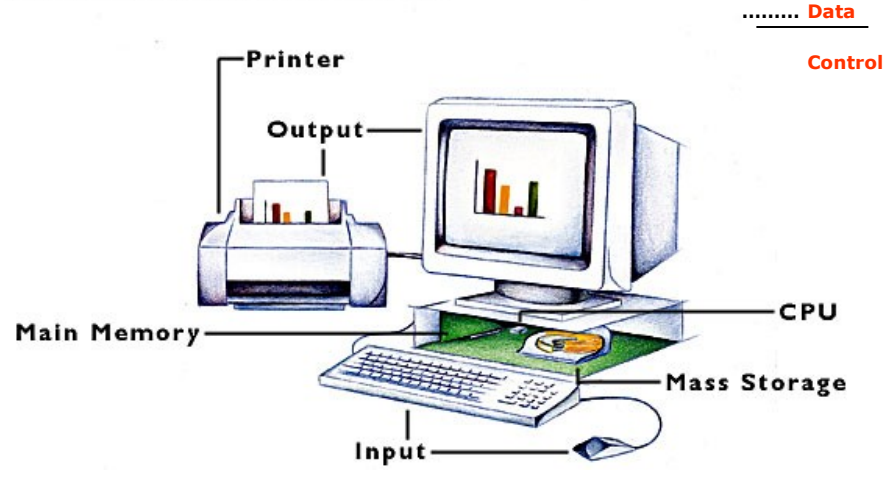
Von Neumann Model

Instructions are processed sequentially

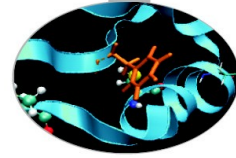
- A single instruction is loaded from memory (fetch) and decoded
- Compute the addresses of operands
- Fetch the operands from memory;
- Execute the instruction ;
- Write the result in memory (store).



Von Neumann Model of Computer Architecture



Speed of Processors: Clock Cycle and Frequency



The *clock cycle* τ is defined as the time between two adjacent pulses of oscillator that sets the time of the processor.

The number of these pulses per second is known as *clock speed* or *clock frequency*, generally measured in GHz (gigahertz, or billions of pulses per second).

The clock cycle controls the synchronization of operations in a computer: All the operations inside the processor last a multiple of τ .

Processor	τ (ns)	freq (MHz)
CDC 6600	100	10
Cyber 76	27.5	36,3
IBM ES 9000	9	111
Cray Y-MP C90	4.1	244
Intel i860	20	50
PC Pentium	< 0.5	> 2 GHz
Power PC	1.17	850
IBM Power 5	0.52	1.9 GHz
IBM Power 6	0.21	4.7 GHz

Increasing the clock frequency:

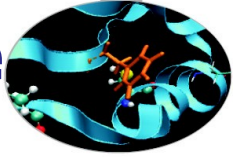
The *speed of light* sets an upper limit to the speed with which electronic components can operate .

Propagation velocity of a signal in a vacuum:

300.000 Km/s = 30 cm/ns

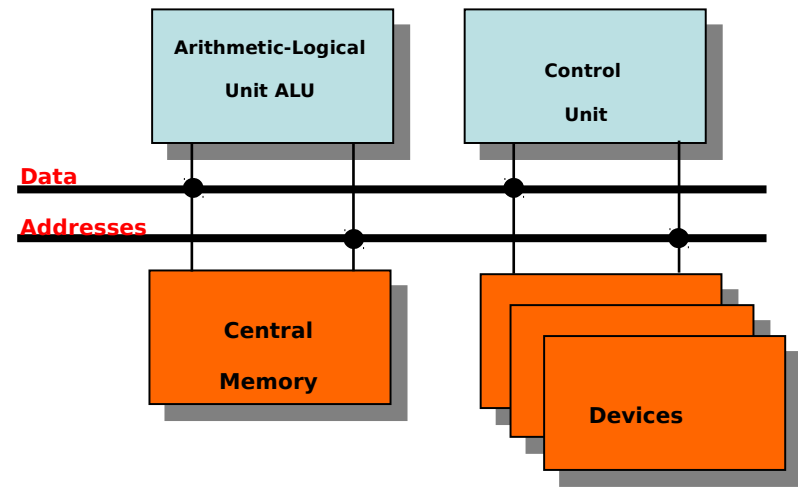
Heat dissipation problems inside the processor. Also Quantum tunnelling expected to become important.

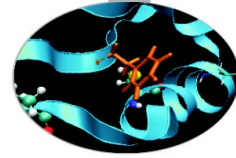
Other factors that affect Performance



In addition to processor power, other factors affect the performance of computers:

- **Size of memory**
- **Bandwidth between processor and memory**
- **Bandwidth toward the I/O system**
- **Size and bandwidth of the cache**
- **Latency between processor, memory, and I/O system**





Memory hierarchies

Time to run code = clock cycles running code + clock cycles waiting for memory

Memory access time: the *time* required by the processor to *access* data or to write data from / to *memory*

The hierarchy exists because :

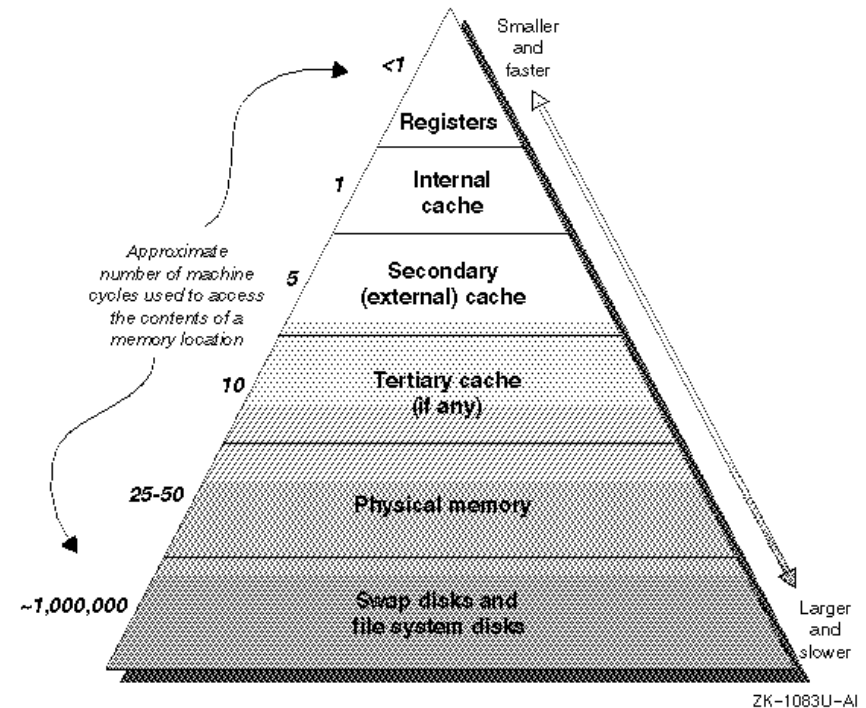
- fast memory is expensive and small
- slow memory is cheap and big

Latency

- how long do I have to wait for the data?
- (cannot do anything while waiting)

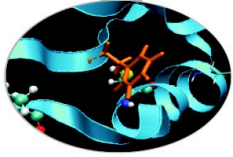
Throughput

- how many bytes/second. but not important if waiting.

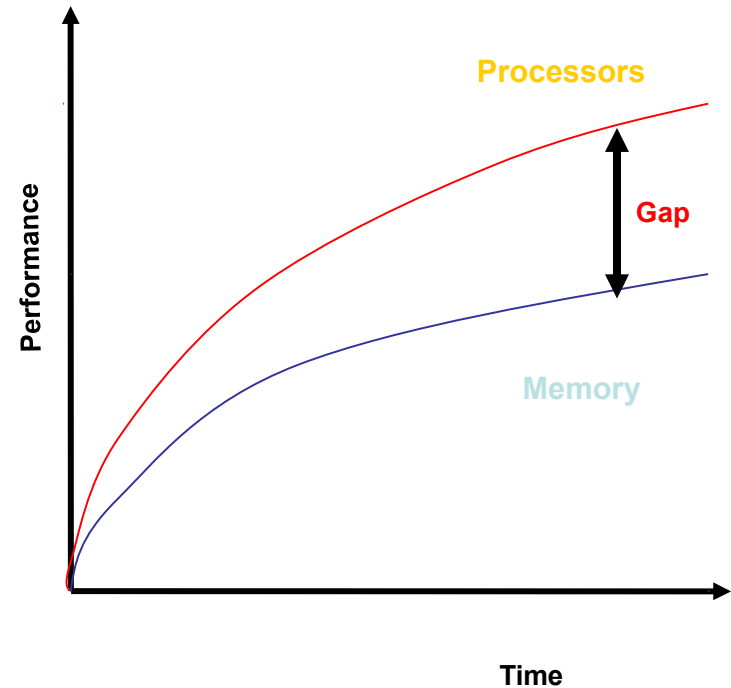


Total time = latency + (amount of data / throughput)

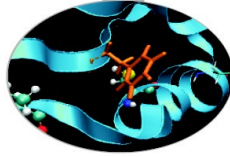
Memory access



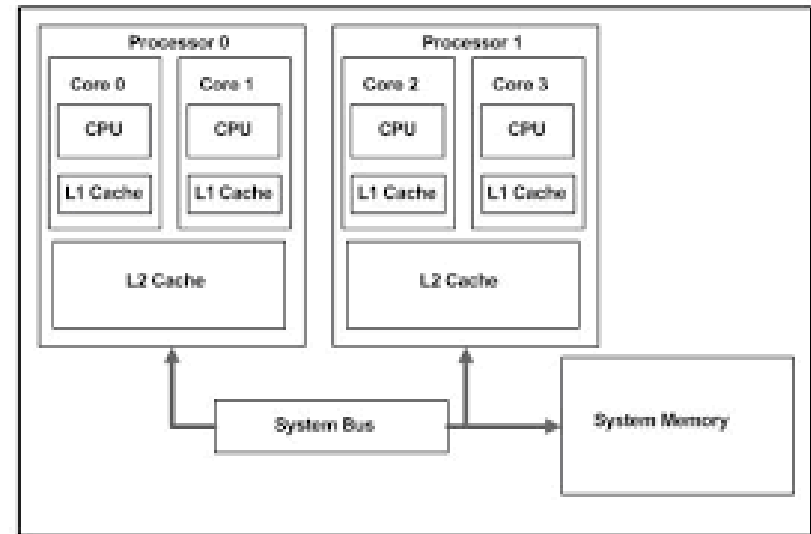
- Important problem for the performance of any computer is access to main memory. Fast processors are useless if memory access is slow!
- Over the years the difference in speed between processors and main memory has been growing.



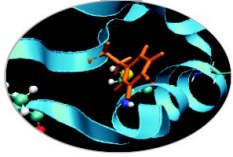
Cache Memory



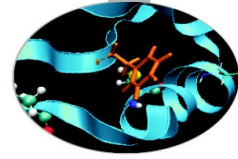
- High speed, small size memory used as a buffer between the main memory and the processor. When used correctly, reduces the time spent waiting for data from main memory.
- Present as various “levels” (e.g. L1, L2, L3, etc) according to proximity to the functional units of the processor.
- Cache efficiency depends on the locality of the data references:
 - **Temporal locality** refers to the re-use of data within relatively small time frame.
 - **Spatial locality** refers to the use of data within close storage locations (e.g. one dimensional array).
- Cache can contain Data, Instructions or both.



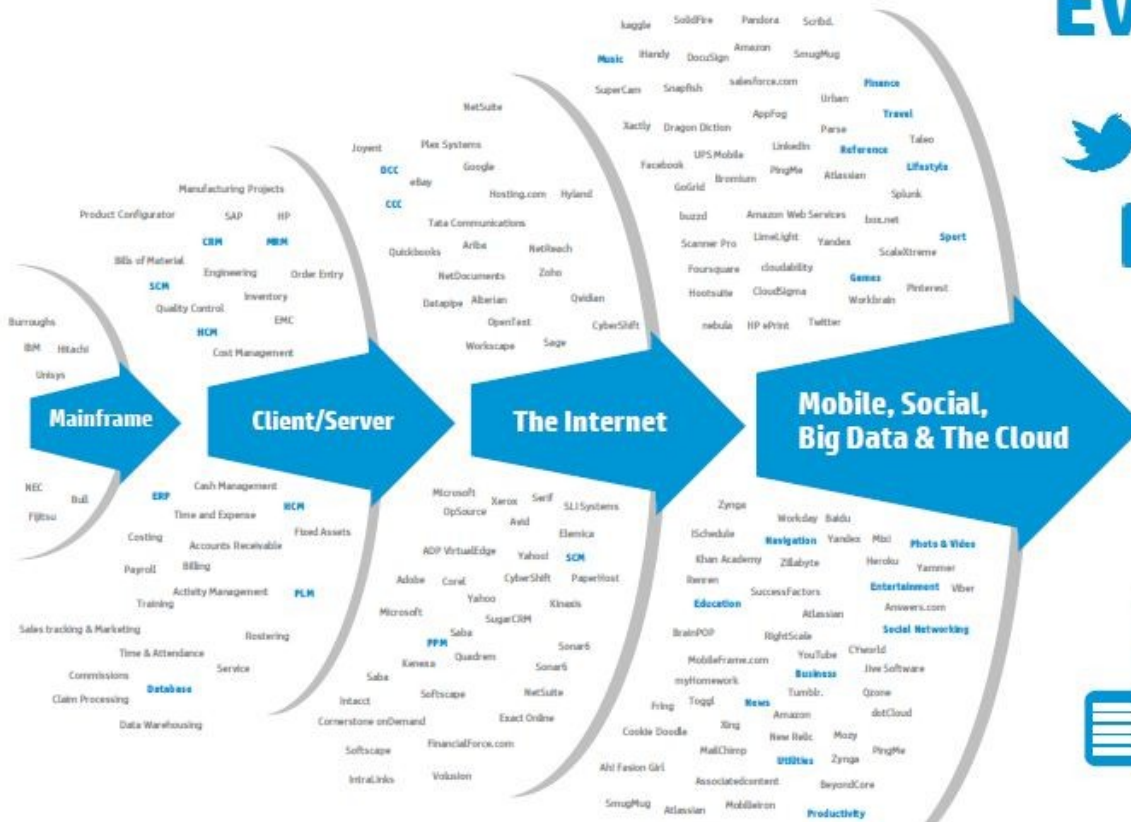
Aspects of parallelism



- It has been recognised for a long time that constant performance improvements cannot be obtained just by increasing factors such as processor clock speed - parallelism is needed.
- In HPC parallelism can be present at many levels:
 - **Functional parallelism within the CPU.**
 - **Pipelining and vectorisation**
 - **Multi-processor and multi-core**
 - **Accelerators**
 - **Parallel I/O**



A new style of IT emerging



Every 60 seconds



98,000+ tweets



695,000 status updates



11 million instant messages



698,445 Google searches



168 million+ emails sent

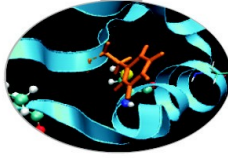


1,820TB of data created

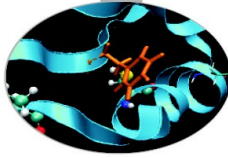


217 new mobile web users

Why?



- **Overwhelming amounts of data generated** by all kinds of devices, networks and programs, *e.g. sensors, mobile devices, internet, social networks, computer simulations, satellites, radiotelescopes, LHC, etc.*
- **Increasing storage capacity**
 - *Storage capacity has doubled every 3 years since 1980 with prices steadily going down*
 - *1,8 zettabytes: an estimation for the data stored by humankind in 2011 (Digital Universe study of International Data Corporation)*
- **Massive data can produce high-value information and knowledge**
- **Critical for data analysis, decision support, forecasting, business intelligence, research, (data-intensive) science, etc.**



A buzz word!

- With different meanings depending on your perspective - e.g. 100 TBs is big for a transaction processing system, but small for a world-wide search engine

A simple "definition" (Wikipedia)

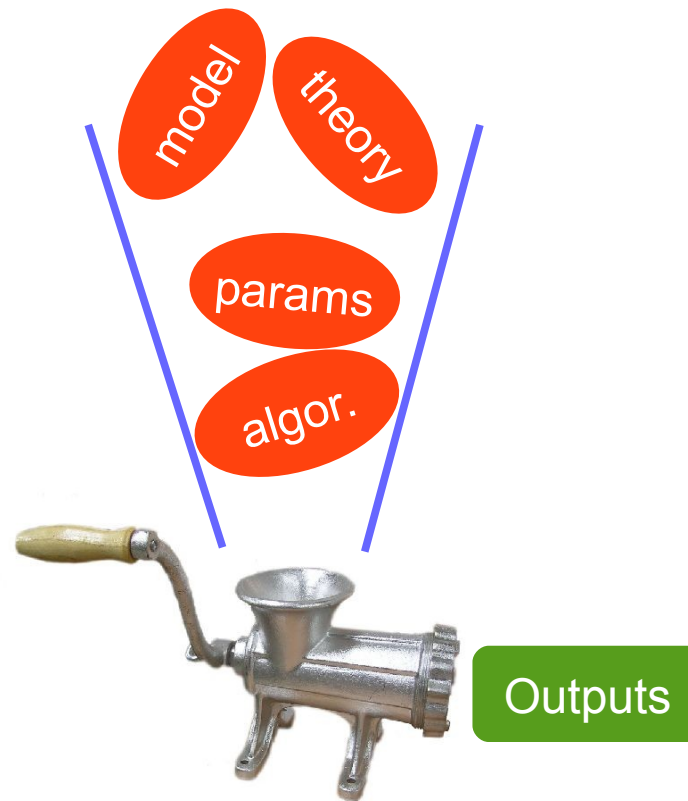
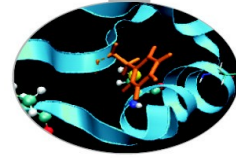
- ***Consists of data sets that grow so large that they become awkward to work with using on-hand database management tools***
 - Difficulties: capture, storage, search, sharing, analytics, visualizing

How big is big?

- Moving target: terabyte (10^{12} bytes), petabyte (10^{15} bytes), exabyte (10^{18}), zetabyte (10^{21})

Scale is only one dimension of the problem!

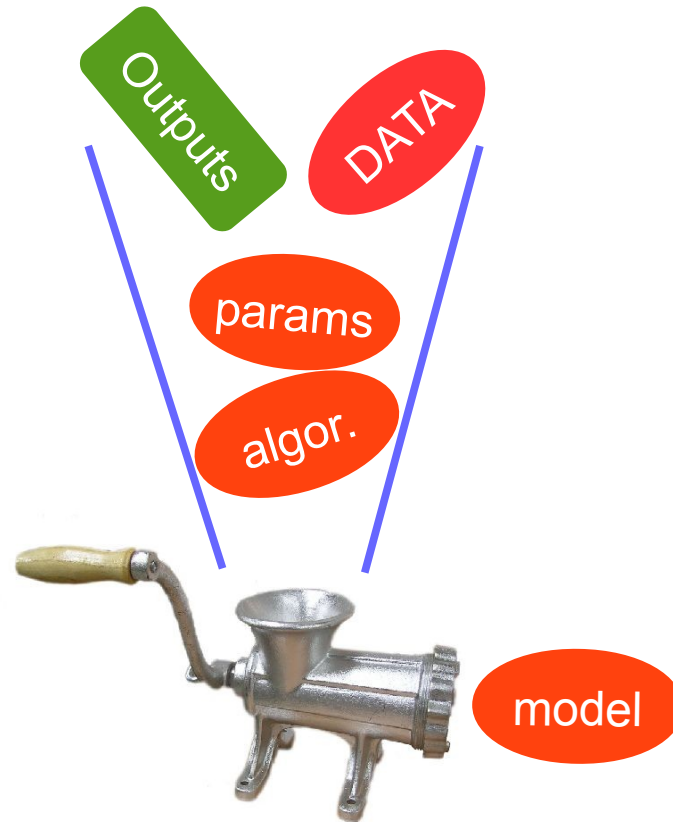
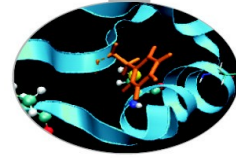
Operate without models *forward problem*



Before...

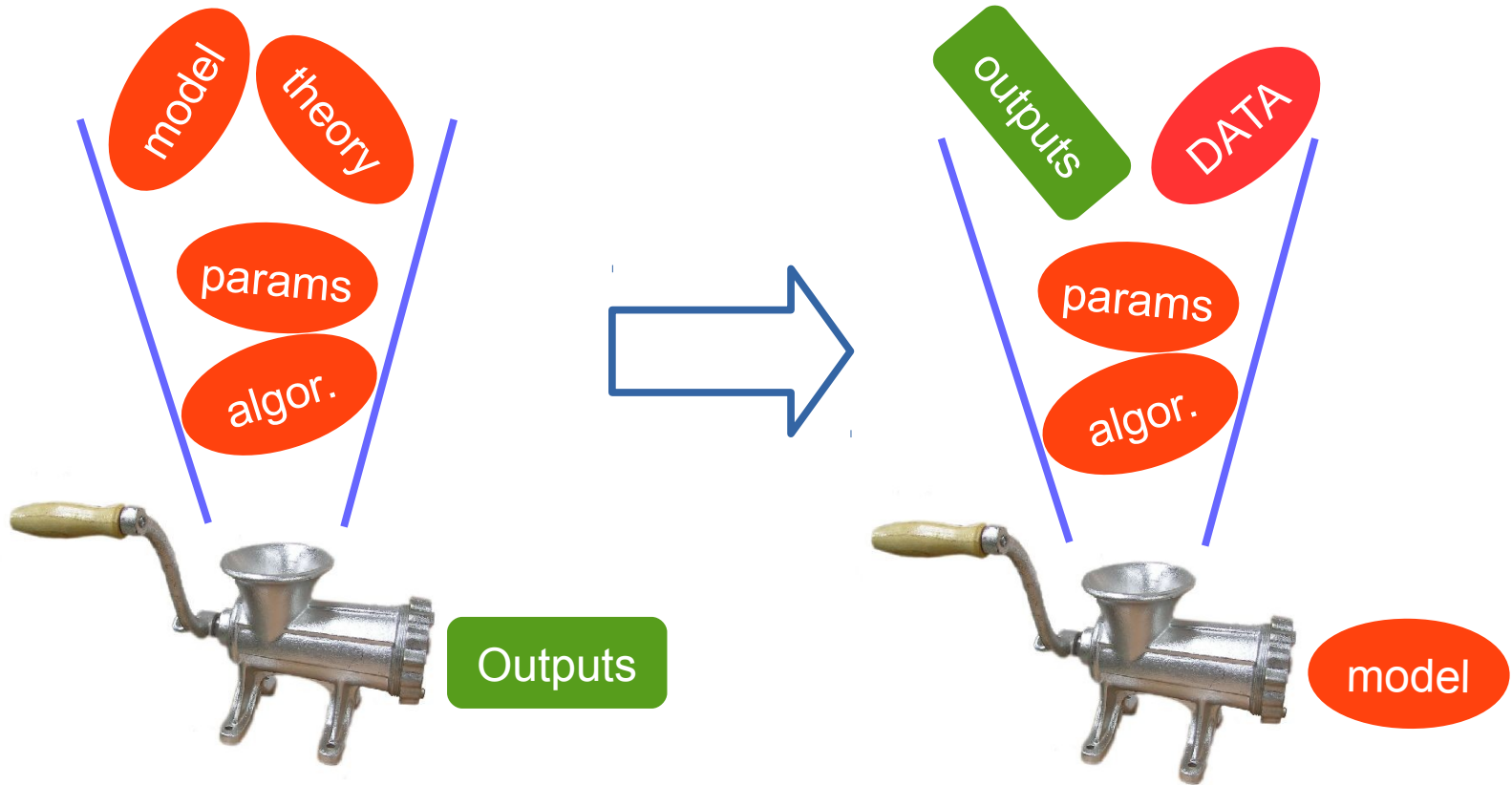
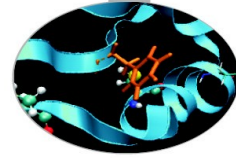
Operate without models

inverse problem

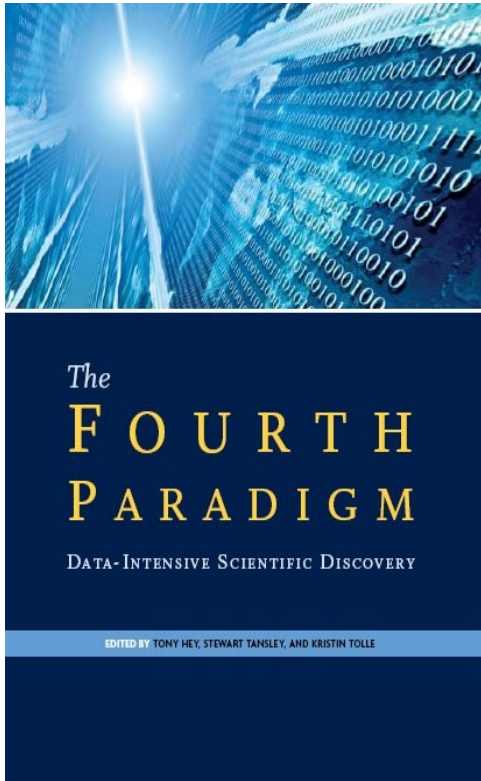
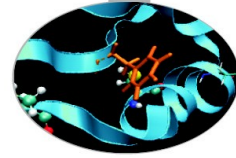


Now, future...

Operate without models (Big Data)



Where all began...a decade ago!



Science Paradigms

- Thousand years ago: science was **empirical**
describing natural phenomena
- Last few hundred years: **theoretical** branch
using models, generalizations
- Last few decades: a **computational** branch
simulating complex phenomena
- Today: **data exploration** (eScience)
unify theory, experiment, and simulation
 - Data captured by instruments or generated by simulator
 - Processed by software
 - Information/knowledge stored in computer
 - Scientist analyzes database/files using data management and statistics

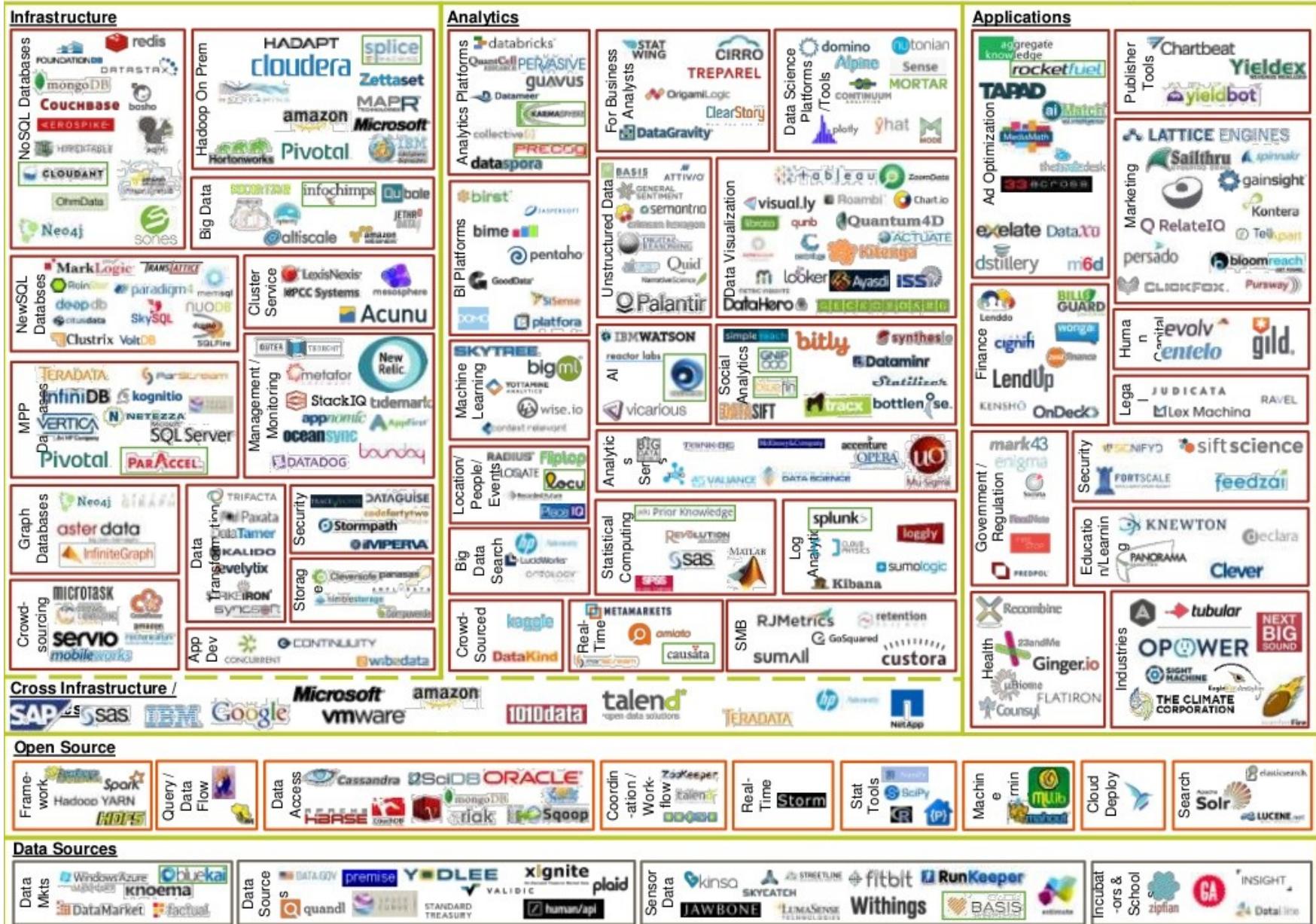
$$\left(\frac{\dot{a}}{a}\right)^2 = \frac{4\pi G\rho}{3} - K\frac{c^2}{a^2}$$



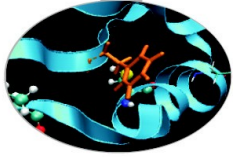
http://research.microsoft.com/en-us/collaboration/fourthparadigm/4th_paradigm_book_complete_lr.pdf

BIG DATA LANDSCAPE, VERSION 3.0

Exited: Acquisition or IPO



Dimensions of the problem



- **Volume**

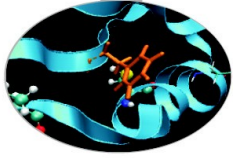
- Refers to massive amounts of data
- Makes it hard to store and manage, but also to analyze (big analytics)

- **Velocity**

- Continuous data streams are being captured (e.g. from sensors or mobile devices) and produced
- Makes it hard to perform online processing

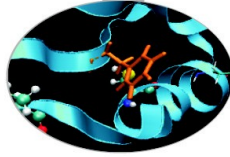
- **Variety**

- Different data formats (sequences, graphs, arrays, ...), different semantics, uncertain data (because of data capture), multiscale data (with lots of dimensions)
- Makes it hard to integrate and analyze



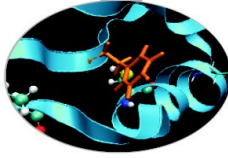
What do we do when there is too much data to process?

Parallel data processing!

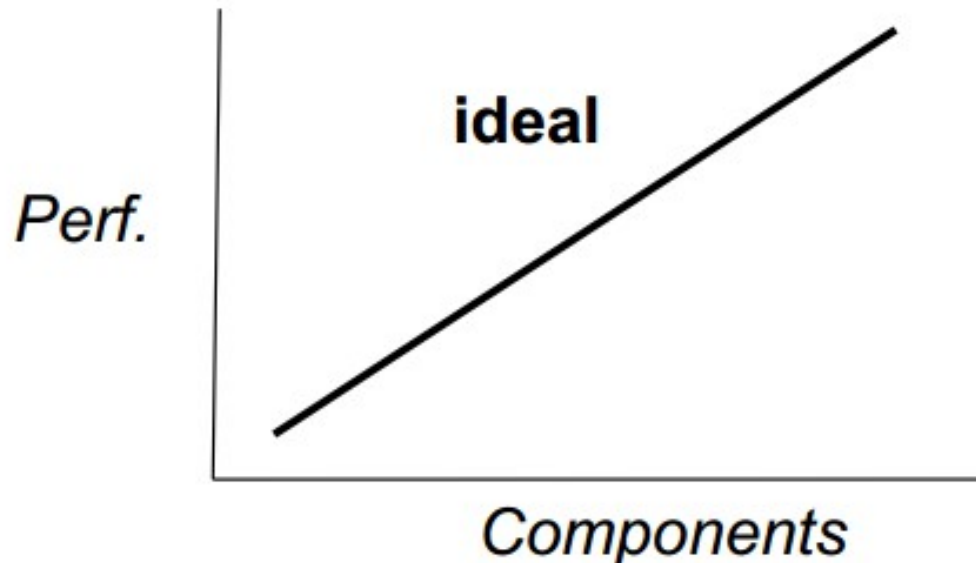


- **Exploit a massively parallel computer**
 - A computer that interconnects lots of CPUs, RAM and disk units
- **To obtain**
 - High performance through data-based parallelism
 - High throughput for transaction-oriented (OLTP) loads
 - Low response time for decision-support (OLAP) queries
 - High availability and reliability through data replication
 - Extensibility with the ideal goals
 - Linear speed-up
 - Linear scale-up

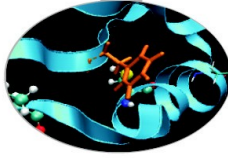
Speed-up



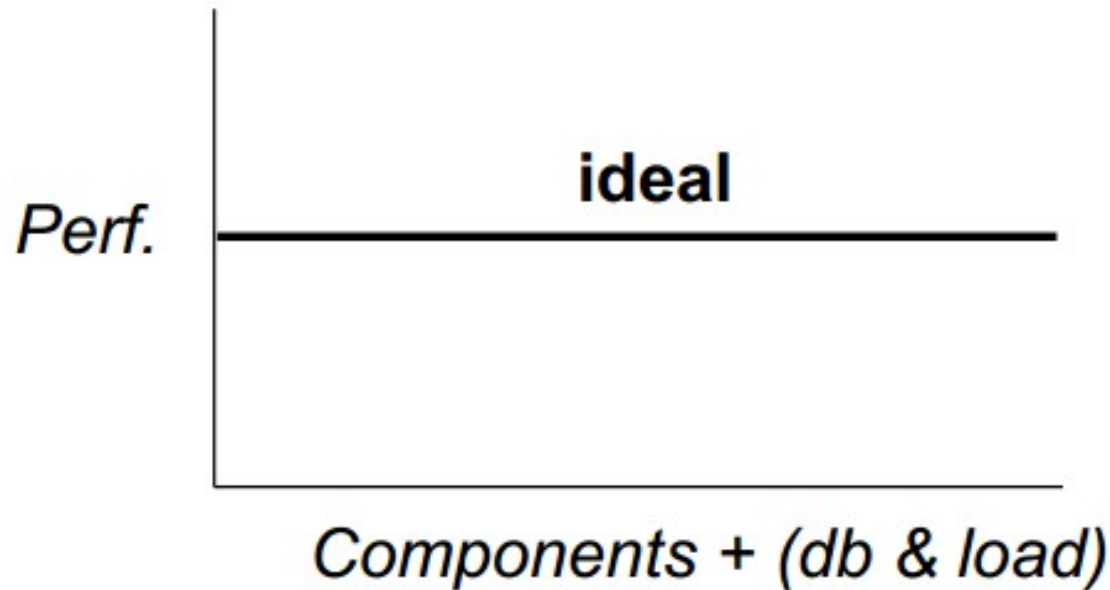
- Linear increase in performance for a constant database size and load, and proportional increase of the system components (CPU, memory, disk)



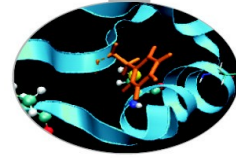
Scale-up



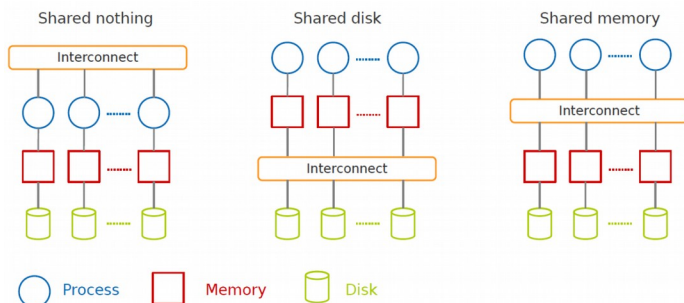
- Sustained performance for a linear increase of database size and load, and proportional increase of components



Parallel Architectures for Data Processing

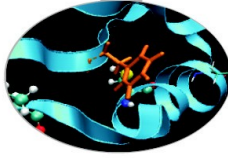


- Three main alternatives, depending on how processors, memory and disk are interconnected
 - ***Shared-memory computer***
 - ***Shared-disk cluster***
 - ***Shared-nothing cluster***



DeWitt, D. and Gray, J. "Parallel database systems: the future of high performance database systems". *ACM Communications*, 35(6), 85-98, 1992.

Shared Memory

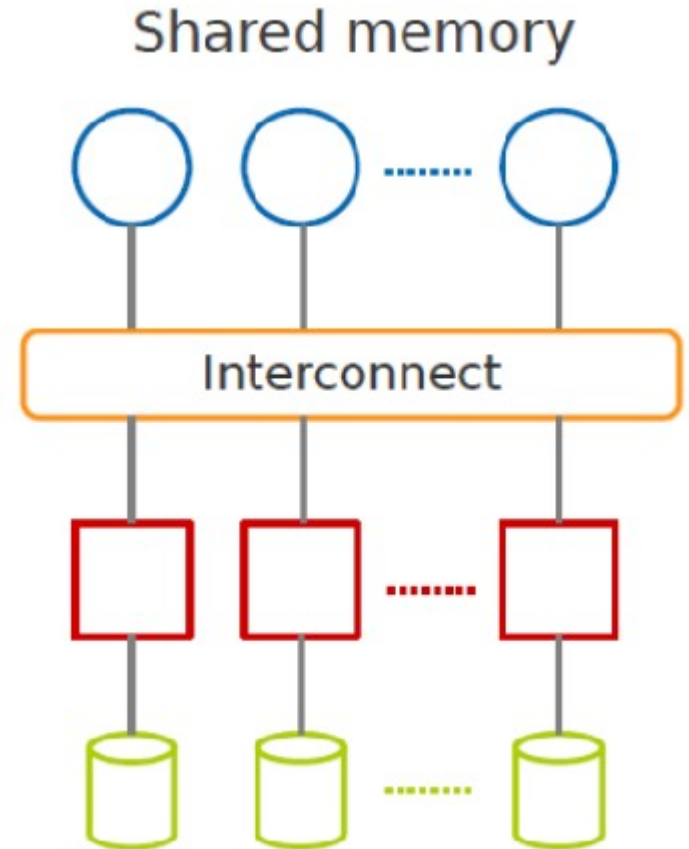


- All memory and disk are shared
 - Symmetric Multiprocessor (SMP)
 - Recent: Non Uniform Memory

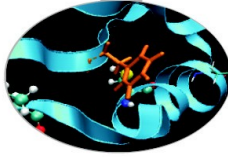
+ Simple for apps, fast com., load balancing

- Complex interconnect limits extensibility, cost

- For write-intensive workloads, not for big data



Shared Disk

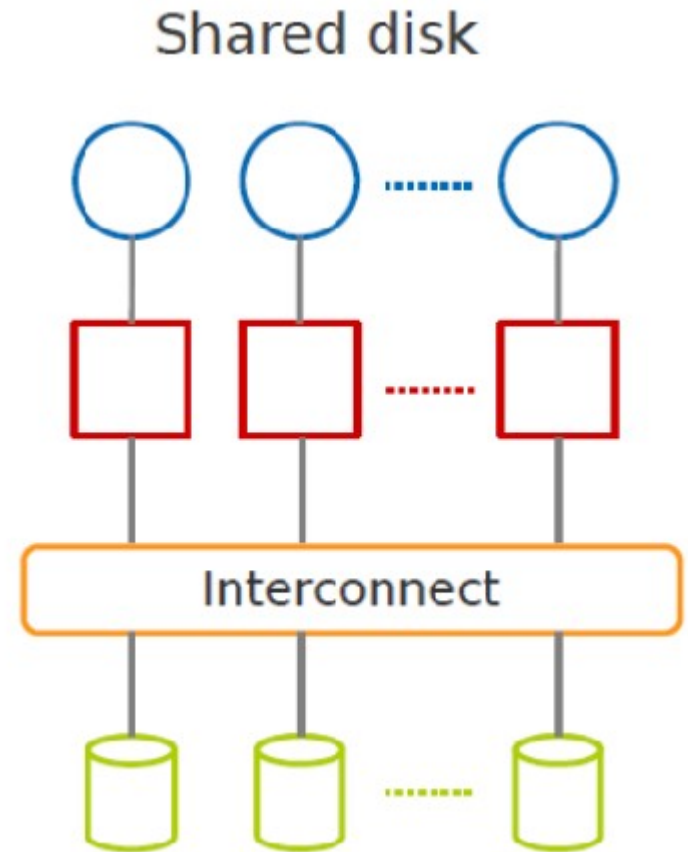


- Disk is shared, memory is private
 - Storage Area Network (SAN) to interconnect memory and disk (block level)
 - Needs distributed lock manager (DLM) for cache coherence

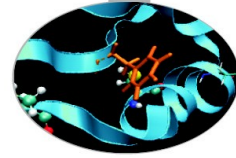
+ Simple for apps, extensibility

- Complex DLM, cost

- For write-intensive workloads or big data



Shared Nothing



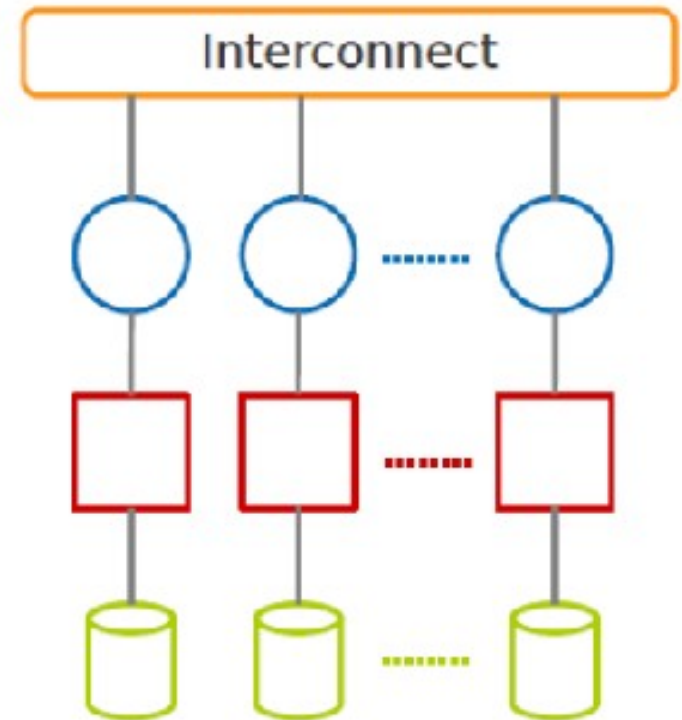
- No sharing of memory or disk across nodes
 - No need for DLM
 - But needs data partitioning

+ highest extensibility, cost

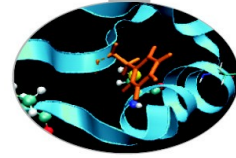
- updates, distributed trans

- For **big data** (read intensive)

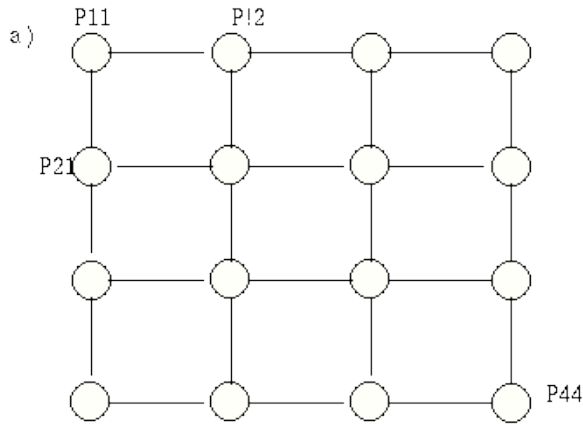
Shared nothing



Example networks



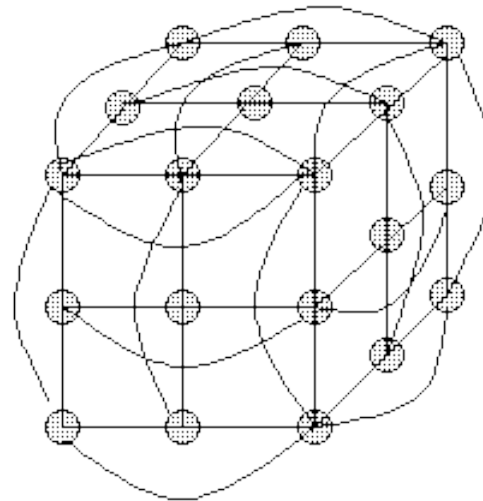
EXAMPLE



2D mesh of width 4 with
no wraparound connections
on edge or corner nodes

corner nodes have degree 2
edge nodes have degree 3

c)



$k = 3$ $w = 3$

i.e. $3^3 = 27$ nodes

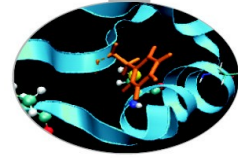
with wraparound connections

all nodes have degree 6 ($2k$)

MESH Topology

Toroidal Topology

Commodity Interconnects



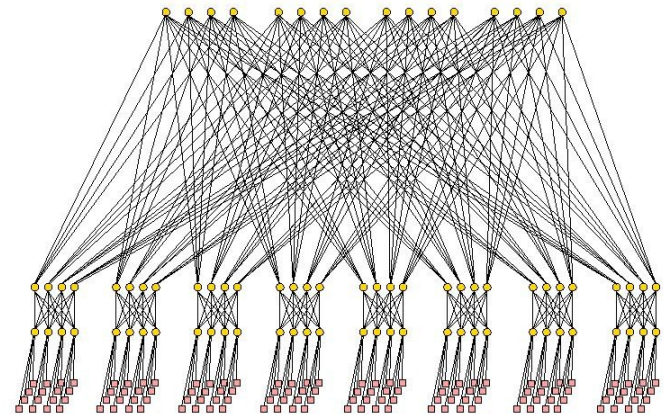
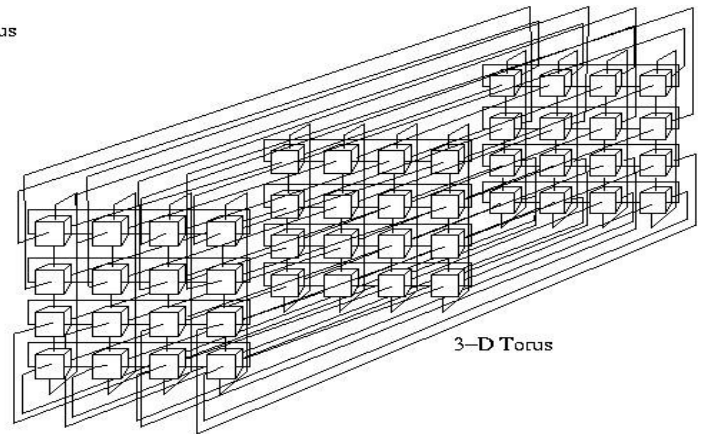
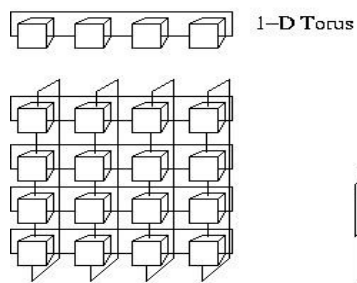
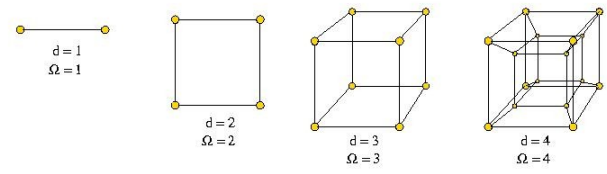
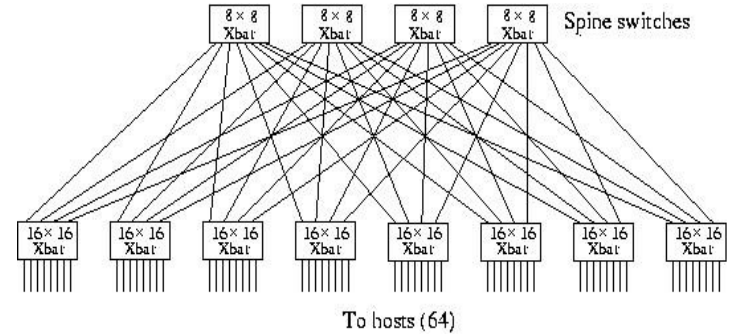
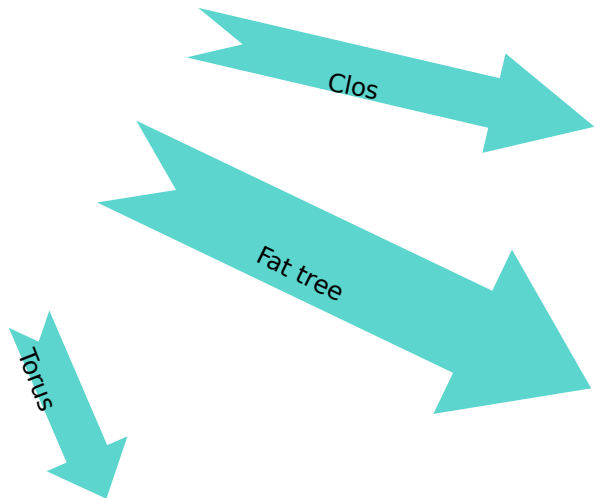
Gig Ethernet

Myrinet

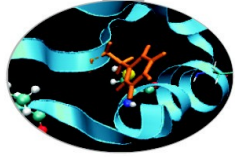
Infiniband

QsNet

SCI



Simple Model for Parallel Data



Shared-nothing architecture

- The most general and scalable

Set-oriented

- Each dataset **D** is represented by a table of rows

Key-value

- Each row is represented by a `<key, value>` pair where

→ *Key uniquely identifies the value in D*

→ *Value is a list of (attribute name : attribute value)*

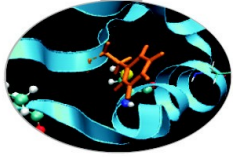
Can represent structured (relational) data or NoSQL data

- But graph is another story (see Pregel, DEX or Spark)

Examples

- `<row-id#5, (part-id:5, part-name:iphone5, supplier:Apple)>`
- `<doc-id#10, (content:<html> html text ... </html>)>`
- `<akeyword, (doc-id:id1, doc-id:id2, doc-id:id10)>`

Considerations



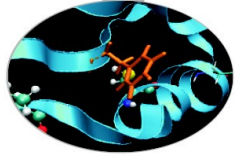
Big datasets

- Data partitioning and indexing
 - *Problem with skewed data distributions*
- Parallel algorithms for algebraic operators
 - *Select is easy, Join is difficult*
- Disk is very slow (10K times slower than RAM)
 - *Exploit main memory data structures and compression*

Query parallelization and optimization

- Automatic if the query language is declarative (e.g. SQL)
- Programmer assisted otherwise (e.g. MapReduce)

Considerations (cont.)



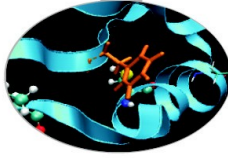
Transaction support

- Hard: need for distributed transactions (distributed locks and 2PC)
 - NoSQL systems don't provide transactions

Fault-tolerance and availability

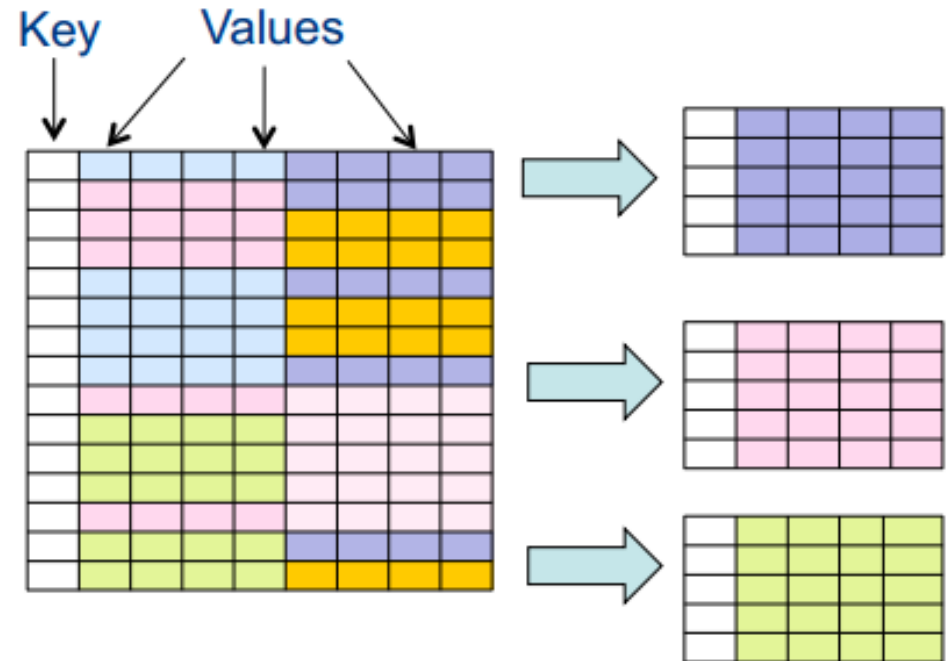
- With many nodes (e.g. several thousand), node failure is the norm

Data Partitioning



Vertical partitioning

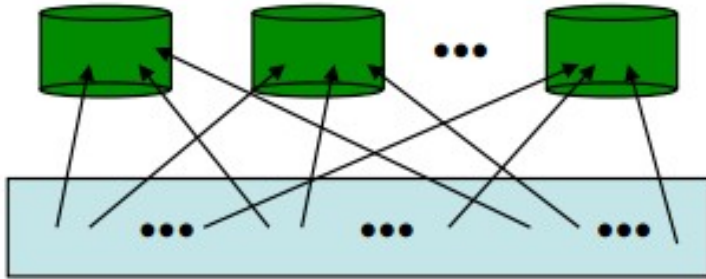
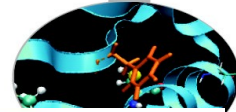
- Basis for column stores (e.g. MonetDB): efficient for OLAP queries
- Easy to compress, e.g. using Bloom filters



Horizontal partitioning (sharding)

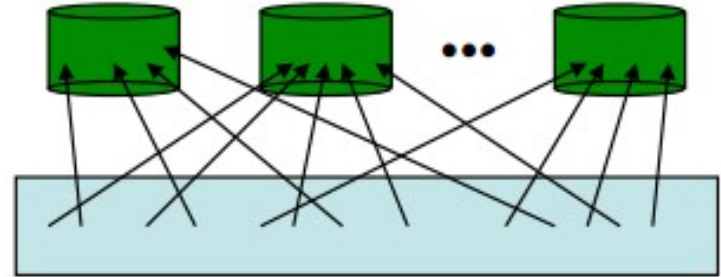
- Shards can be stored (and replicated) at different nodes

Sharding Schemes



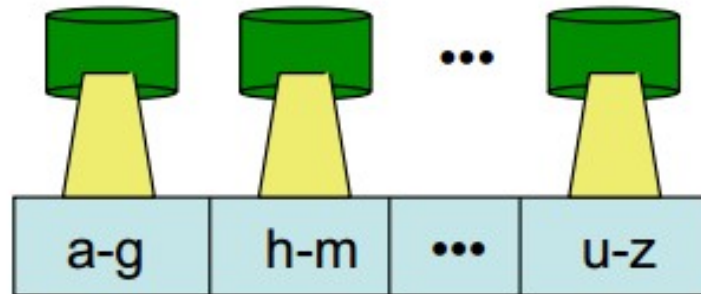
Round-Robin

- i th row to node $(i \bmod n)$
- perfect balancing
- but full scan only



Hashing

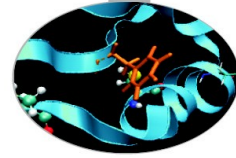
- (k,v) to node $h(k)$
- exact-match queries
- but problem with skew



Range

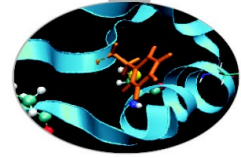
- (k,v) to node that holds k 's interval
- exact-match and range queries
- deals with skew

Different classes of applications

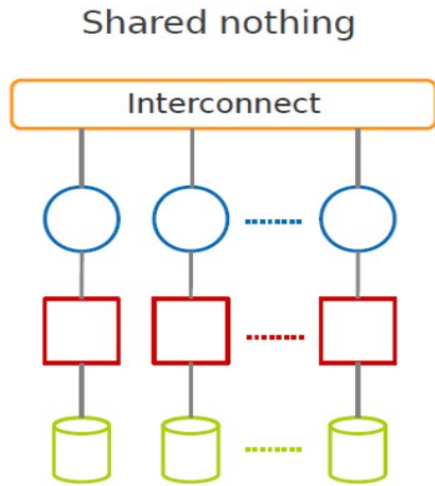


- **MPI (Message Passing Interface)**
 - A shared disk infrastructure for processing large data sets with a parallel algorithm on clusters
- **OpenMP (Open MultiProcessing)**
 - A shared memory infrastructure for processing large data sets with a parallel algorithm on a node
- **Map Reduce (Hadoop)**
 - A shared nothing architecture for processing large data sets with a distributed algorithm on clusters

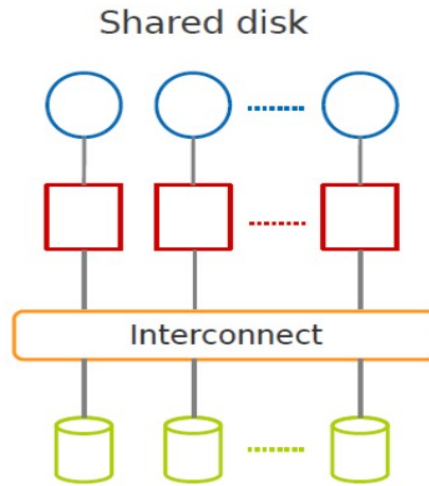
Parallel Architectures



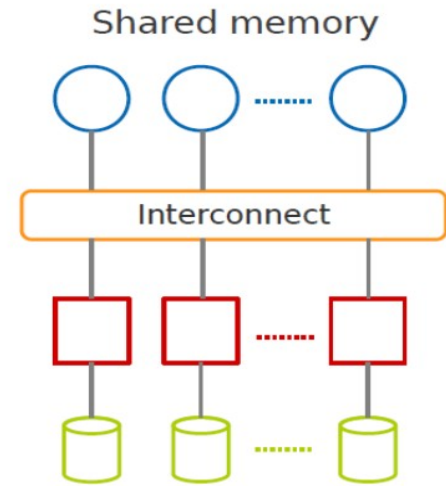
Map Reduce/Hadoop



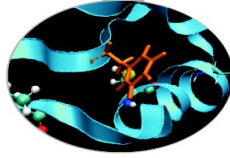
MPI



OpenMP



Programming Models: What is MPI?



- **Message Passing Interface (MPI)**
 - World's most popular distributed API
 - MPI is "de facto standard" in scientific computing
 - C and FORTRAN, ver. 2 in 1997
- **What is MPI good for?**
 - Abstracts away common network communications
 - Allows lots of control without bookkeeping
 - Freedom and flexibility come with complexity
 - 300 subroutines, but serious programs with fewer than 10
- **Basics:**
 - One executable run on every node
 - Each node process has a rank ID number assigned
 - Call API functions to send messages

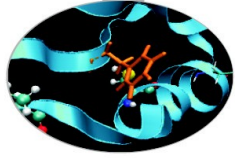


<http://www.mpi-forum.org/>

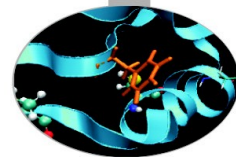
<http://forum.stanford.edu/events/2007/plenary/slides/Olukotun.ppt>

<http://www.tbray.org/ongoing/When/200x/2006/05/24/On-Grids>

Challenges with MPI

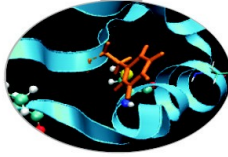


- **Deadlock is possible...**
 - Blocking communication can cause deadlock
 - "crossed" calls when trading information
 - example:
 - Proc1: MPI_Receive(Proc2, A);
MPI_Send(Proc2, B);
 - Proc2: MPI_Receive(Proc1, B);
MPI_Send(Proc1, A);
 - There are some solutions - MPI_SendRecv()
- **Large overhead from comm. mismanagement**
 - Time spent blocking is wasted cycles
 - Can overlap computation with non-blocking comm.
- **Load imbalance is possible! Dead machines?**
- **Things are starting to look hard to code!**



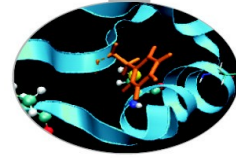
Brief recap

Map Reduce flow

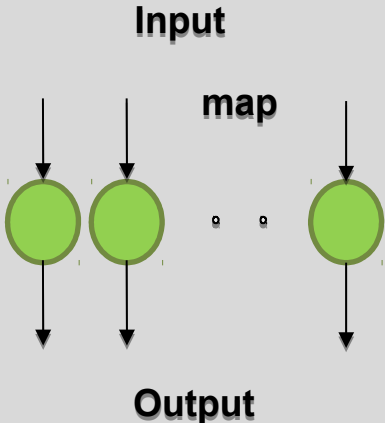
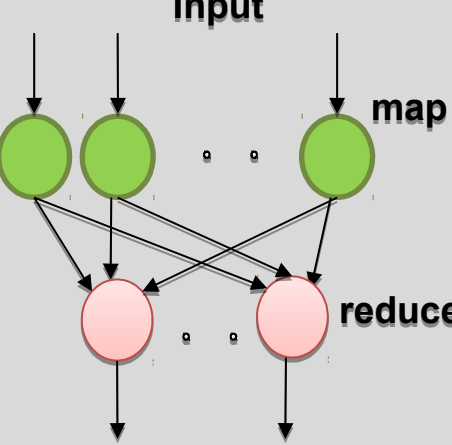
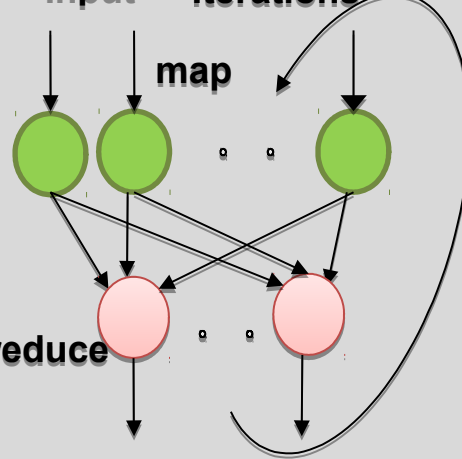
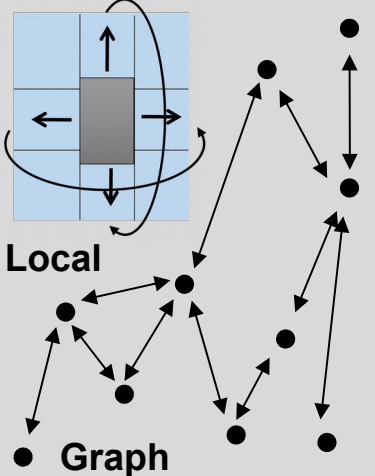


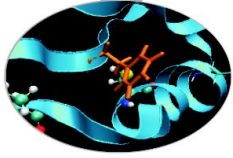
- Programmers must specify:
 - map** $(k, v) \rightarrow \text{list}(\langle k', v' \rangle)$
 - reduce** $(k', \text{list}(v')) \rightarrow \langle k'', v'' \rangle$
 - All values with the same key are reduced together
- Optionally, also:
 - partition** $(k', \text{number of partitions}) \rightarrow \text{partition for } k'$
 - Often a simple hash of the key, e.g., $\text{hash}(k') \bmod n$
 - Divides up key space for parallel reduce operations
 - combine** $(k', v') \rightarrow \langle k', v' \rangle^*$
 - Mini-reducers that run in memory after the map phase
 - Used as an optimization to reduce network traffic
- The execution framework handles everything else...

“Everything Else”



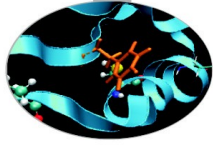
- The execution framework handles everything else...
 - **Scheduling**: assigns workers to map and reduce tasks
 - **“Data distribution”**: moves processes to data
 - **Synchronization**: gathers, sorts, and shuffles intermediate data
 - **Errors and faults**: detects worker failures and restarts
- Limited control over data and execution flow
 - All algorithms must be expressed in m, r, c, p
- You don't know:
 - Where mappers and reducers run
 - When a mapper or reducer begins or finishes
 - Which input a particular mapper is processing
 - Which intermediate key a particular reducer is processing

(1) Map Only	(2) Classic MapReduce	(3) Iterative Map Reduce or Map-Collective	(4) Point to Point or Map-Communication
 <p>Input</p> <p>map</p> <p>Output</p>	 <p>Input</p> <p>map</p> <p>reduce</p>	 <p>Input</p> <p>Iterations</p> <p>map</p> <p>reduce</p>	 <p>Local</p> <p>Graph</p>
<p>BLAST Analysis Local Machine Learning Pleasingly Parallel</p>	<p>High Energy Physics (HEP) Histograms Distributed search Recommender Engines</p>	<p>Expectation maximization Clustering e.g. K- means Linear Algebra, PageRank</p>	<p>Classic MPI PDE Solvers and Particle Dynamics Graph Problems</p>
<p>MapReduce and Iterative Extensions (Spark, Twister)</p>			<p>MPI, Giraph</p>
<p>Integrated Systems such as Hadoop + Harp with Compute and Communication model separated</p>			



Are emerging data analytics techniques the new El Dorado?

Where and When using Apache Hadoop



Where

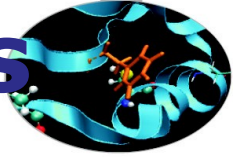
- Batch data processing, not real-time
- Highly parallel data intensive distributed applications
- Very large production deployments

When

- Process lots of unstructured data
- When your processing can easily be made parallel
- Running batch jobs is acceptable
- When you have access to lots of cheap hardware

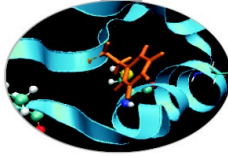


Advantages/Disadvantages

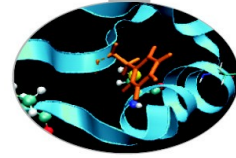


- **Now it's easy to program for many CPUs**
 - Communication management effectively gone
 - I/O scheduling done for us
 - Fault tolerance, monitoring
 - machine failures, suddenly-slow machines, etc are handled
 - Can be much easier to design and program!
- **But ... it further restricts solvable problems**
 - Might be hard to express problem in MapReduce
 - Data parallelism is key
 - Need to be able to break up a problem by data chunks
 - MapReduce is closed-source (to Google) C++
 - Hadoop is open-source Java-based rewrite

Overall limitations



- *MapReduce provides an easy-to-use framework for parallel programming, but is it the most efficient and best solution to program execution in datacenters?*
- **DeWitt and Stonebraker: "MapReduce: A major step backwards"** – MapReduce is far less sophisticated and efficient than parallel query processing
- **MapReduce is a parallel processing framework, not a database system, nor a query language**
 - It is possible to use MapReduce to implement some of the parallel query processing function
- **What are the real limitations?**
 - Inefficient for general programming (and not designed for that)
 - Hard to handle data with complex dependence, frequent updates, etc.
 - High overhead, bursty I/O, difficult to handle long streaming data
 - Limited opportunity for optimization

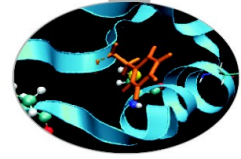


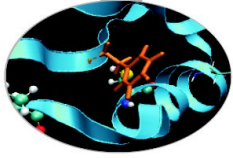
MapReduce can be classified as a SIMD (single-instruction, multiple-data) problem.

- Indeed, the map step is highly scalable because the same instructions are carried out over all data. Parallelism arises by breaking the data into independent parts with no forward or backward dependencies (side effects) within a Map step; that is, the Map step may not change any data (even its own).
- The reducer step is similar, in that it applies the same reduction process to a different set of data (the results of the Map step).
- In general, the MapReduce model provides a functional, rather than procedural, programming model. Similar to a functional language, MapReduce cannot change the input data as part of the mapper or reducer process, which is usually a large file. Such restrictions can at first be seen as inefficient; however, the lack of side effects allows for easy scalability and redundancy.

An HPC cluster, on the other hand, can run SIMD and MIMD (multiple-instruction, multiple-data) jobs.

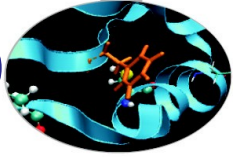
- The programmer determines how to execute the parallel algorithm. Users, however, are not restricted when creating their own MapReduce application within the framework of a typical HPC cluster.





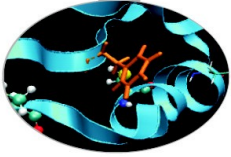
When to use Apache Hadoop

When to use Apache Hadoop



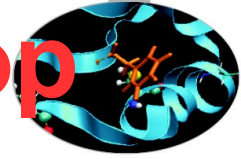
- **Your Data Sets Are Really Big**
 - *Don't even think about Hadoop if the data you want to process is measured in MBs or GBs.* If the data driving the main problem you are hoping to use Hadoop to solve is measured in GBs, save yourself the hassle and use Excel, a SQL BI tool on Postgres, or some similar combination. On the other hand, if it's several TB or (even better) measured in petabytes, Hadoop's superior scalability will save you a considerable amount of time and money
- **You Celebrate Data Diversity**
 - One of the advantages of the Hadoop Distributed File System (HDFS) is it's really flexible in terms of data types. It doesn't matter whether your raw data is structured, semi-structured (like XML and log files), unstructured (like video files).

When to use Apache Hadoop



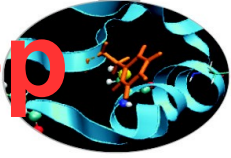
- **You Find Yourself Throwing Away Perfectly Good Data**
 - One of the great things about Hadoop is its capability to store petabytes of data. If you find that you are throwing away potentially valuable data because its costs too much to archive, you may find that setting up a Hadoop cluster allows you to retain this data, and gives you the time to figure out how to best make use of that data.

When to NOT use MR + Hadoop



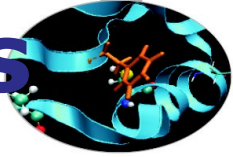
- **You Need Answers in a Hurry**
 - Hadoop is probably not the ideal solution if you need really fast access to data. The various SQL engines for Hadoop have made big strides in the past year, and will likely continue to improve. But if you're using Map-Reduce to crunch your data, expect to wait days or even weeks to get results back.
- **Your Queries Are Complex and Require Extensive Optimization**
 - Hadoop is great because it gives you a massively parallel cluster for low-cost Intel servers and scads of cheap hard disk capacity. While the hardware and scalability is straightforward, getting the most out of Hadoop typically requires a hefty investment in the technical skills required to optimize queries.

When to NOT use MR + Hadoop



- **You Require Random, Interactive Access to Data**
 - The pushback from the limitations of the batch-oriented MapReduce paradigm in early Hadoop led the community to improve SQL performance and boost its capability to serve interactive queries against random data. While SQL on Hadoop is getting better, in most cases it's not a reason in of itself to adopt Hadoop.
- **You Want to Store Sensitive Data**
 - Hadoop is evolving quickly and is able to do a lot of things that it couldn't do just a few years ago. But one of the things that it's not particularly good at today is storing sensitive data. Hadoop today has basic data and use access security. And while these features are improving by the month, the risks of accidentally losing personally identifiable information due to Hadoop's less-than-stellar security capabilities is probably not worth the risk.

Advantages/Disadvantages



- **Now it's easy to program for many CPUs**
 - Communication management effectively gone
 - I/O scheduling done for us
 - Fault tolerance, monitoring
 - machine failures, suddenly-slow machines, etc are handled
 - Can be much easier to design and program!
 - Can cascade several (many?) Map-Reduce tasks
- **But ... it further restricts solvable problems**
 - Might be hard to express problem in Map-Reduce
 - Data parallelism is key
 - Need to be able to break up a problem by data chunks
 - Map-Reduce is closed-source (to Google) C++
 - Hadoop is open-source Java-based rewrite