Professor Dr. Dieter Kranzlmüller

Dr. Andre Luckow

Maximilian Höb

Data and Advanced Analytics Infrastructure and Applications

03.04.2018

Audience

- VL HPC and Parallel Computing, Dr. Karl Fuerlinger
- VL, Big Data Management and Analytics, Professor Dr. Matthias Schubert
- Other topics: Distributed Computing, Databases?

- Skills:
  - Linux
  - Python
  - Parallel Applications

Credits

- Big Data MOOC, Professor Geoffrey Fox, Indiana University
- Collaborators: Shantenu Jha, Judy Qui
- Introduction to Data Science, Bill Howe, University of Washington
- Parallel and Distributed Computing, Bettina Schnor, University of Potsdam
- Many more… see references in slides.

Objective of Lecture

- Identify and explain the functions of the main components of big data and machine learning technologies.
- Be able to explain the tradeoffs of various big data infrastructure and machine learning solutions.
- Understand and analyze performance and cost tradeoffs for different architecture options.
- Understand the current state-of-the art and research in the domain of data infrastructure and machine learning.
- Formulate big data solutions to several common types of data problems.
- Be able to use big data platforms such as Spark for writing data analytics code and applications

Aufbau Vorlesung

	Dienstag, 03.04.2019	**Mittwoch, 04.04.2019**	**Donnerstag, 05.04.2018**	**Freitag, 06.04.2018**	**Samstag, 07.04.2018**
09 – 10:30 Uhr	Motivation and Introduction	Hadoop (HDFS, MapReduce, Yarn)	SQL on Hadoop	Deep Learning	Operationalization of AI (Streaming, Model Deployment)
10:45 – 12:15 Uhr	Introduction HPC Hadoop (HDFS, MapReduce, Yarn)	Spark	Data Science Machine Learning	Scalable Machine Learning & Performance Analysis	Exercise
12:45 – 14:15 Uhr	Exercise	Exercise	Exercise	Exercise	Exercise
14:30 – 16:00 Uhr	Exercise	Exercise	Exercise	Exercise	Review

Exercise

- You are expected to program! This will require you to read the documentation of various tools and frameworks:
  - Python
  - Pandas
  - Scikit-Learn
  - Spark
  - Keras/Tensorflow
- Every person is expected to submit his exercise at the end of each day.
- Plagiarism will not be tolerated.

Exercise

- Deadline for Turning in Exercises:
  - The deadline for turning in Exercises will be announced at the begin of each exercise.
  - Tentative Plan:
    - 04/03: Exercise 1 and 2
    - 04/04: Exercise 3 and 4
    - 04/05: Exercise 5 and 6
    - 04/06: Exercise 7 and 8
    - 04/07: Exercise 9
  - https://github.com/scalable-infrastructure/exercise-students-2018
  - Please send exercises to: mailto:maximilian.hoeb@nm.ifi.lmu.de and mailto:luckow@nm.ifi.lmu.de

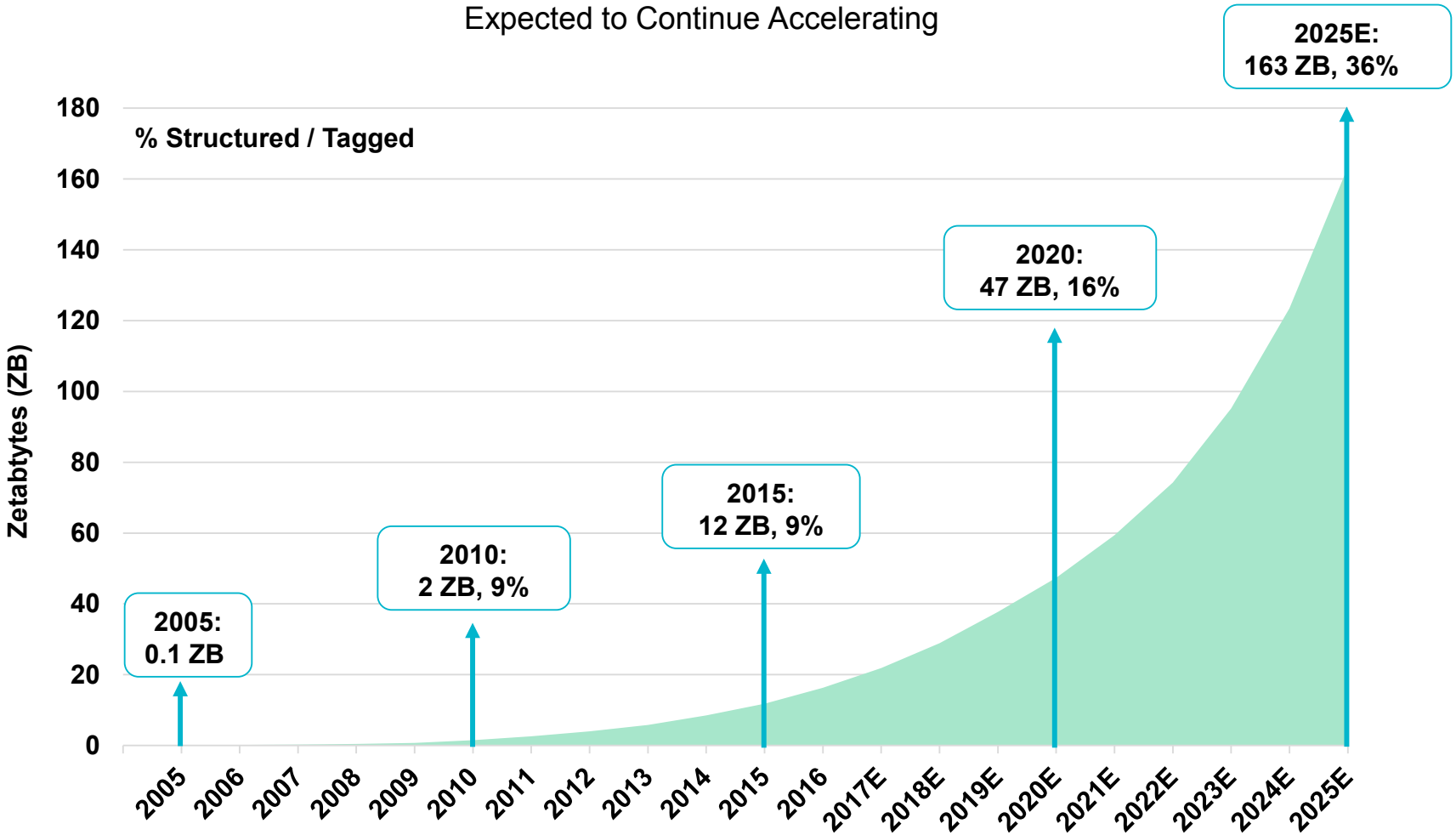
Motivation

- Amount of data generated in sciences and enterprise environments is increasing
- Big Data describes data that can not be process with traditional data processing technologies
- Google, Amazon, Microsoft & Facebook operate on large-scale infrastructures for processing peta-scale data
- Enterprise Computing requirements (in traditional industry fields, i.e. non-IT) with respect to data and compute are increasing:
  - Large volumes of structured/unstructured data (sensors, logs)
  - Realtime processing requirements
- Machine Learning is becoming the tool of choice for many data analysis tasks and data products.

Data Volumes

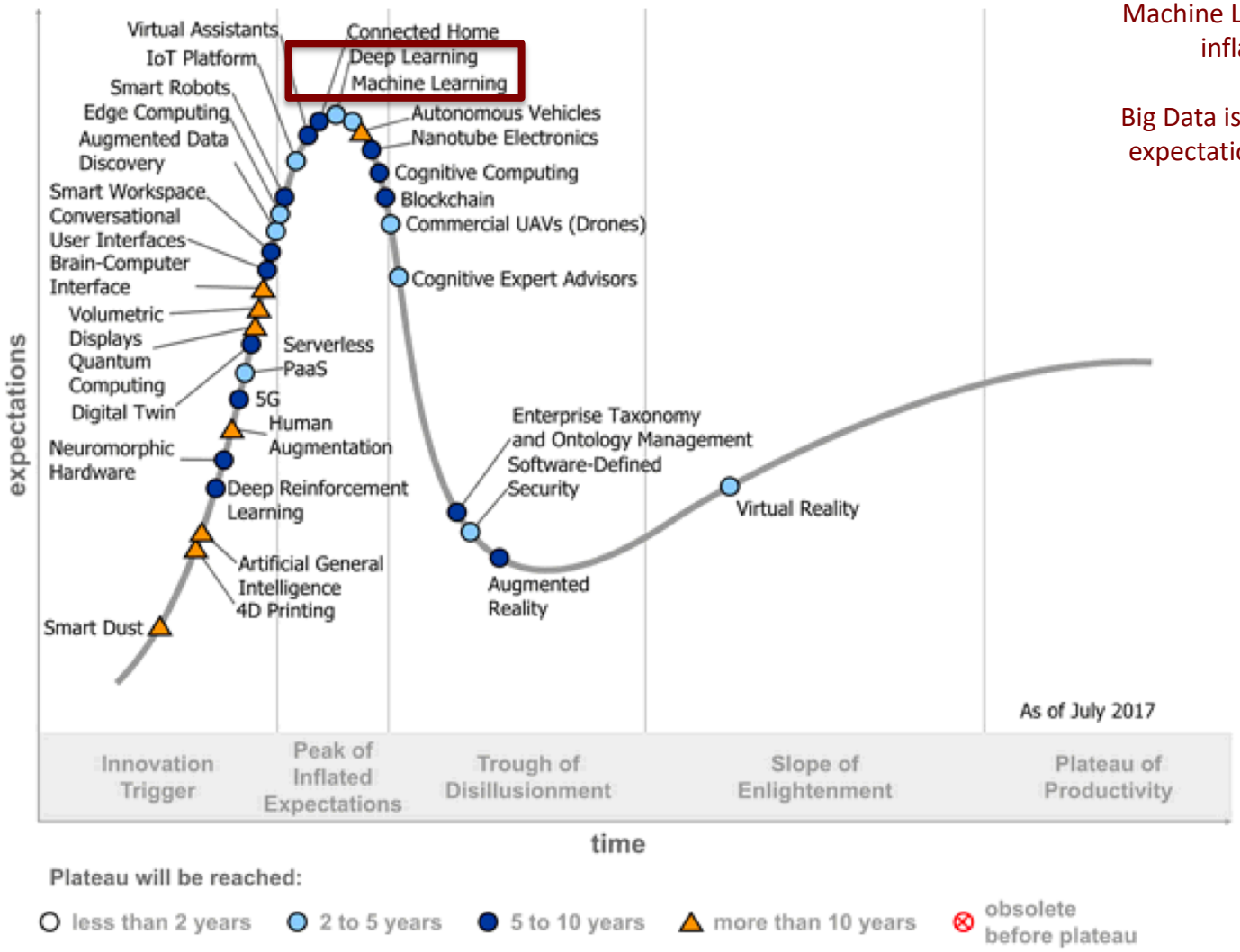
Information Created Worldwide =

Expected to Continue Accelerating



Source: Mary Meeker, Internet Trends, http://www.kpcb.com/internet-trends, 2017.
IDC DataAge 2025 Study, sponsored by Seagate, https://www.seagate.com/files/www-content/our-story/trends/files/Seagate-WP-DataAge2025-March-2017.pdf, 2017.
Note: 1 petabyte = 1MM gigabytes, 1 zeta byte = 1 million petabytes, 1 zeta byte = 1 billion TB

Gartner Hypecycle Emerging Technologies 2017.

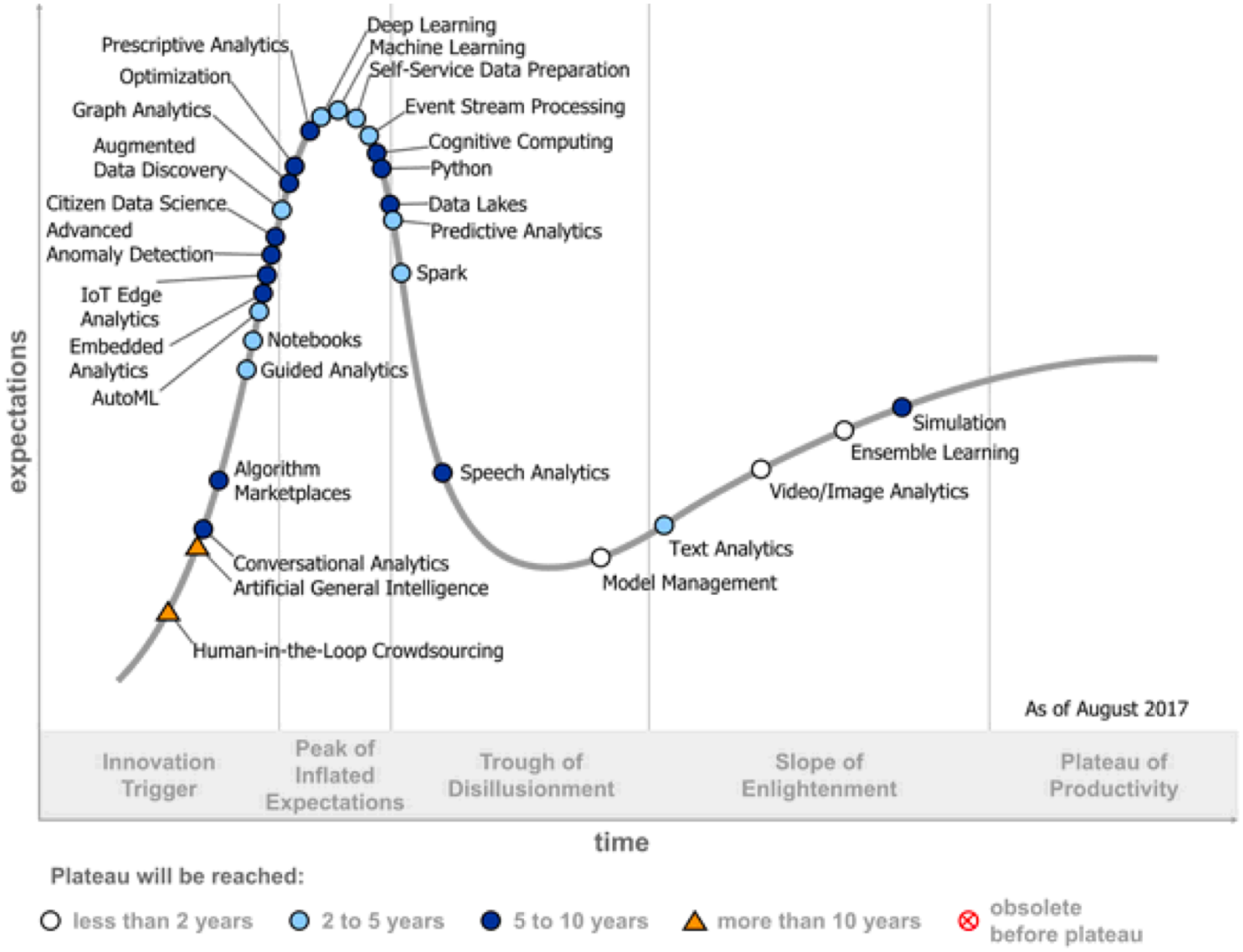


Machine Learning is on the Peak of inflated expectations.

Big Data is past the peak of inflated expectations, but it is also all over the Hypecycle.

https://www.gartner.com/document/3768572

Hypecycle Machine Learning and Data Science



https://www.gartner.com/document/3772081

Data-driven Science

- Jim Gray: The Fourth Paradigm of Science: move from empirical, theoretical, computational resource to data.
- Data Exploration to unify theory, experiment and simulation:
  - Data captured by instruments or generated by simulation
  - Scientists use data mining and statistics to extract knowledge from data

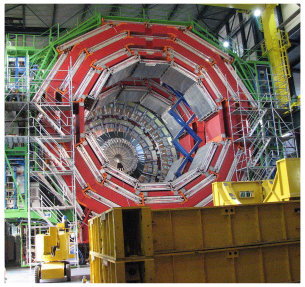
http://research.microsoft.com/en-us/collaboration/fourthparadigm/

- Chris Anderson, Wired:

The end of theory: the data deluge makes scientific method obsolete

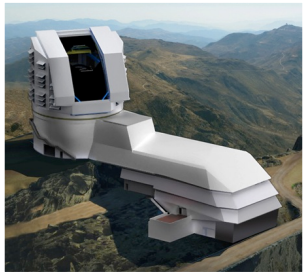
http://archive.wired.com/wired/issue/16-07

Data-intensive Science Examples



High Energy Physics:

- LHC at CERN produces multiple TB of data per day.
- Data is processed and distributed across Tier 1 and 2 sites.



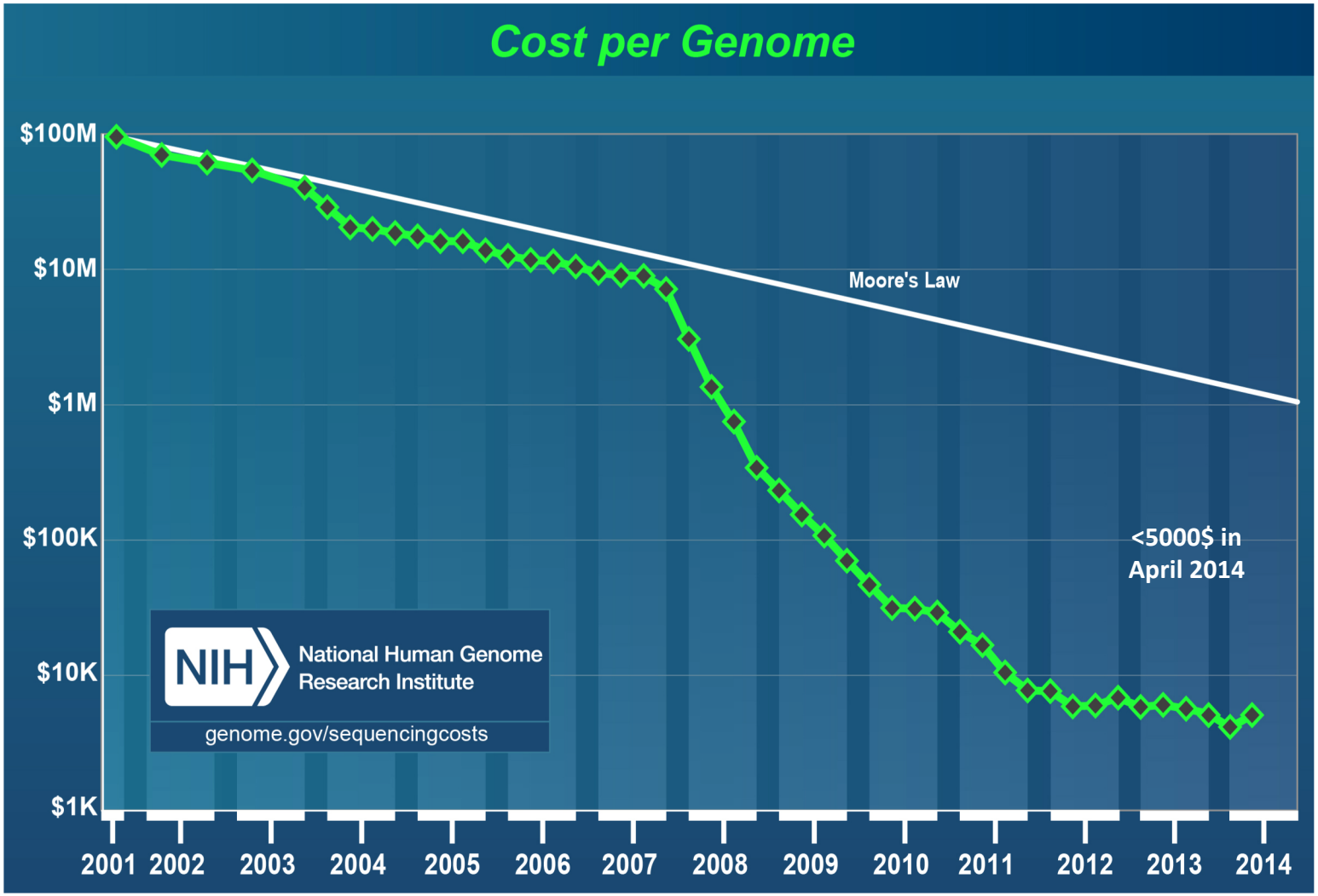
Astronomy:

- Sloan Digital Sky Survey (80 TB over 7 years).
- LSST will produce 15 TB per day (for 10 years).

Geonomics:

- Data volume increasing with every new generation of sequence machine. A machine can produce TB/day.
- Size of human genome: 3.2 GB
- Costs for sequencing are decreasing.

DNA Sequencing (1)

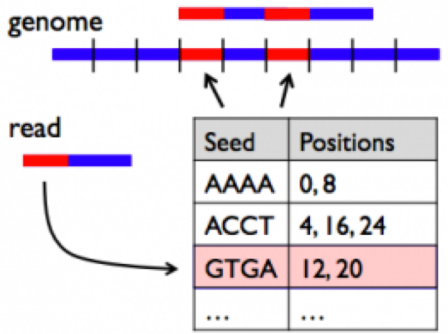


http://www.genome.gov/sequencingcosts

DNA Sequencing (2)

HiSeq X Ten enables "extreme" volume $1000 genome sequencing for institutes performing population and disease studies.
http://www.illumina.com/systems/hiseq-x-sequencing-system.ilmn

- Raw FASTQ data can be up to 500 GB in size
- Processing of data requires multiple steps to translate outputs from sequence machine (short reads) into unique characteristics using reference genome



https://amplab.cs.berkeley.edu/projects/snap/

Light Source Data

https://www.bnl.gov/

For many types of experiments, data needs to be processed in a time-sensitive if not real-time manner, to support steering of the experiments

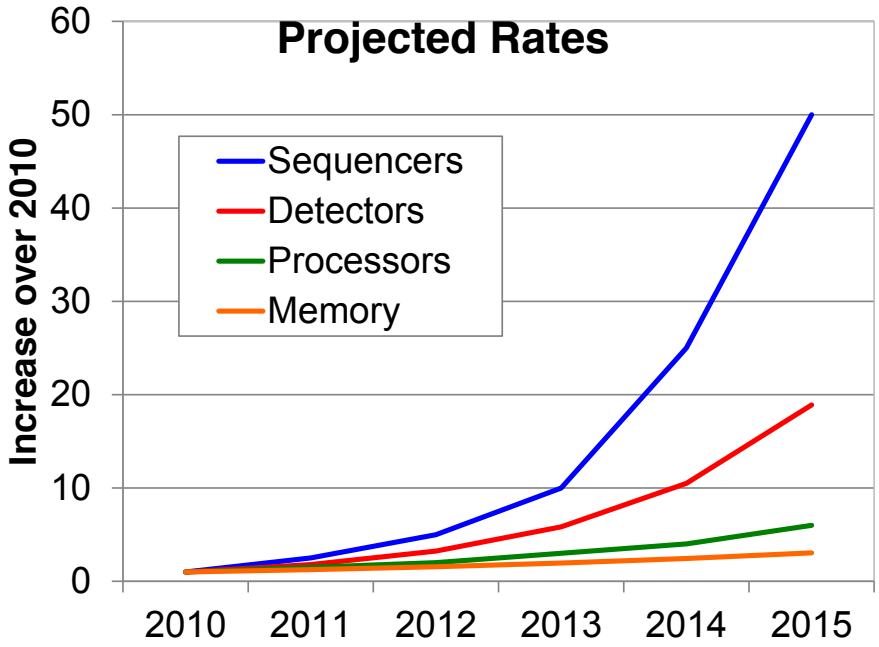
Stream processing capabilities are increasingly important to analyze and derive realtime insights on incoming data from experiments, simulations, and Internet-of-Things (IoT) sensors.

There is a range of experimental facilities and observational systems with a variety of streaming data and computational characteristics, such as National Synchrotron Light Source (NSLS-II), APS and ALS. These light sources are projected to generate data at a rate of 20 GB/sec.

https://www.youtube.com/watch?v=RKqof77pKBc&feature=youtu.be&t=1m53s

Geoffrey Fox, Shantenu Jha, Lavanya Ramakrishnan, LBNL, STREAM 2016, https://science.energy.gov/~/media/ascr/pdf/programdocuments/docs/2017/STREAM2016.pdf

Data Growth is Outpacing Computing Growth



- **Sequences and detectors (used in light sources, particle accelerators) are growing faster than processor speeds**
- **And all are growing faster than memory density**

Kathy Yelick, HPC Meets Big Data: Everything Old is New Again, TTI Vanguard, 2013

Data and Simulation.

Experimental facilities use simulation in design
– Risk mitigation and engineering

Simulations generate big data
– Massive scale and massive throughput simulations

Data access improves science
– Community data sets democratize and improve quality

Data requires simulation
– Missing, noisy; models to interpret

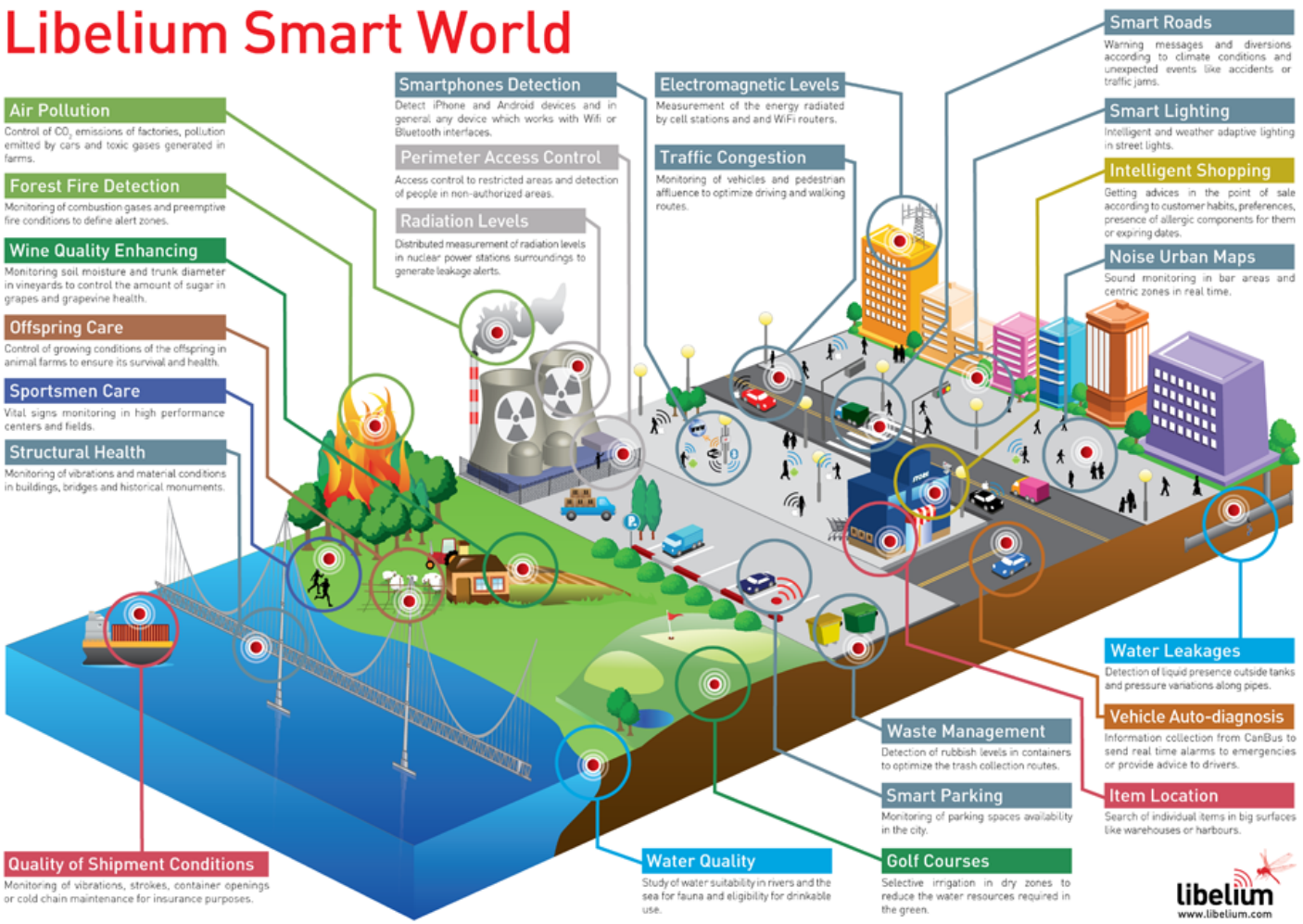
Data analytics uses big computing
– Both capacity and novel architectures

Focus of this lecture

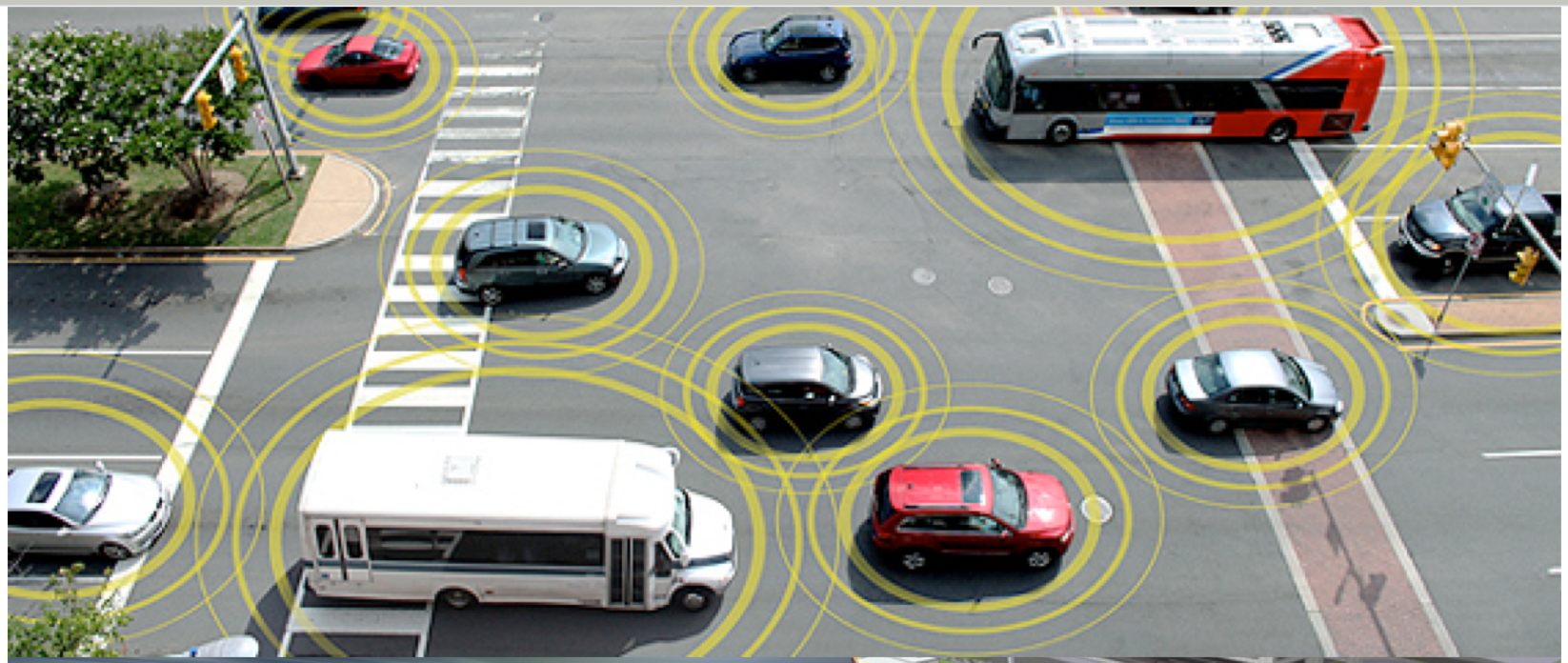
Kathy Yelick, HPC Meets Big Data: Everything Old is New Again, TTI Vanguard, 2013

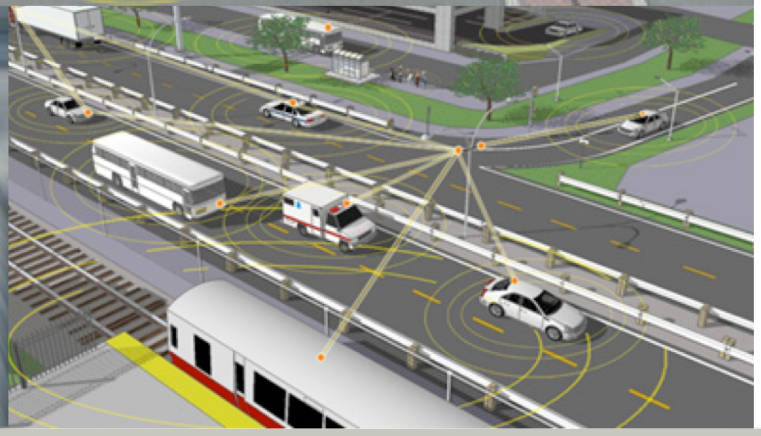
Internet of Things: Driving the Data Deluge

- The Internet of Things (IoT) is the network of physical objects that contain embedded technology to communicate and sense or interact with their internal states or the external environment (Gartner).
- Pervasive, connected sensor devices and the need to process this data will push Big Data requirements to the next level


Libelium Smart World
Air Pollution
Control of CO_2 emissions of factories, pollution emitted by cars and toxic gases generated in farms.
Forest Fire Detection
Monitoring of combustion gases and preemptive fire conditions to define alert zones.
Wine Quality Enhancing
Monitoring soil moisture and trunk diameter in vineyards to control the amount of sugar in grapes and grapevine health.
Offspring Care
Control of growing conditions of the offspring in animal farms to ensure its survival and health.
Sportsmen Care
Vital signs monitoring in high performance centers and fields.
Structural Health
Monitoring of vibrations and material conditions in buildings, bridges and historical monuments.
Smartphones Detection
Detect iPhone and Android devices and in general any device which works with Wifi or Bluetooth interfaces.
Perimeter Access Control
Access control to restricted areas and detection of people in non-authorized areas.
Radiation Levels
Distributed measurement of radiation levels in nuclear power stations surroundings to generate leakage alerts.
Electromagnetic Levels
Measurement of the energy radiated by cell stations and and WiFi routers.
Traffic Congestion
Monitoring of vehicles and pedestrian affluence to optimize driving and walking routes.
Smart Roads
Warning messages and diversions according to climate conditions and unexpected events like accidents or traffic jams.
Smart Lighting
Intelligent and weather adaptive lighting in street lights.
Intelligent Shopping
Getting advices in the point of sale according to customer habits, preferences, presence of allergic components for them or expiring dates.
Noise Urban Maps
Sound monitoring in bar areas and centric zones in real time.
Water Leakages
Detection of liquid presence outside tanks and pressure variations along pipes.
Vehicle Auto-diagnosis
Information collection from CanBus to send real time alarms to emergencies or provide advice to drivers.
Item Location
Search of individual items in big surfaces like warehouses or harbours.
Waste Management
Detection of rubbish levels in containers to optimize the trash collection routes.
Smart Parking
Monitoring of parking spaces availability in the city.
Golf Courses
Selective irrigation in dry zones to reduce the water resources required in the green.
Water Quality
Study of water suitability in rivers and the sea for fauna and eligibility for drinkable use.
Quality of Shipment Conditions
Monitoring of vibrations, strokes, container openings or cold chain maintenance for insurance purposes.
libelium
www.libelium.com

Intelligent Transport Systems.





http://www.slideshare.net/annegalang/smart-cities-installation

Intelligent Transport Systems

Intelligent Infrastructure

Arterial and Freeway Management
- Traffic Signal Control, Lane Management
- Surveillance, Enforcement

Crash Prevention and Safety
- Warning Systems
- Pedestrian Safety

Traffic Incident Management
- Surveillance, Detection
- Response, Clearance

Emergency Management
- Hazardous Material Management
- Emergency Medical Services

Electronic Payment and Pricing
- Toll Collection
- Multi-Use Payment

Roadway Operations
- Asset Management
- Work Zone Management

Transit Management
- Operations and Fleet Management
- Transportation Demand Management

Traveller Information
- Pre-trip and En-Route Information
- Tourism and Events

Road Weather Information
- Surveillance and Prediction
- Traffic Control

Information Management
- Information Warehousing Services
- Archived Data Management

Commercial Vehicle Operations
- Carrier Operations, Fleet Management
- Credentials Administration

Intermodal Freight
- Freight and Asset Tracking
- International Border Crossing

Intelligent Vehicles

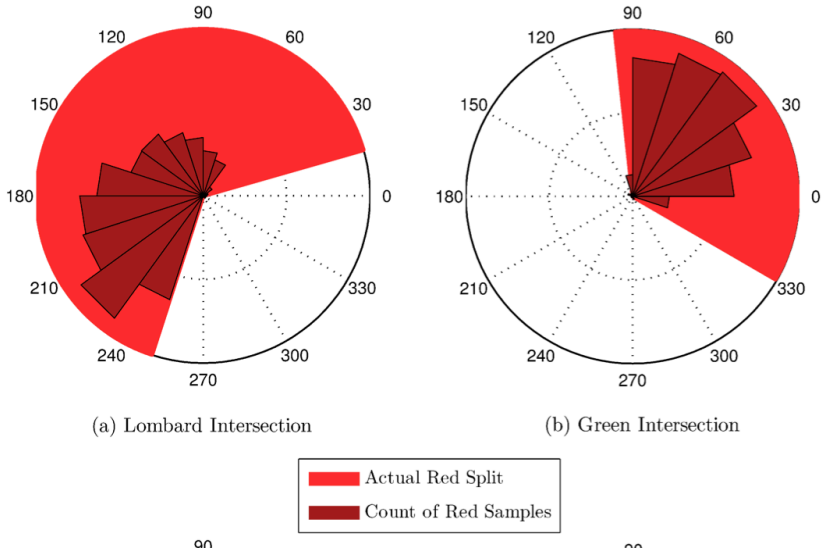
Collision Avoidance
- Obstacle Detection
- Collision-Avoidance Sensor Technologies

Driver Assistance
- Navigation, Route Guidance
- On-Board Monitoring

Collision Notification
- Advanced Automated Collision Notification
- In-Vehicle Crash Sensors

http://www.itsoverview.its.dot.gov/
http://www.oecd.org/sti/44379113.pdf

Intelligent Traffic Systems



(a) Lombard Intersection

(b) Green Intersection

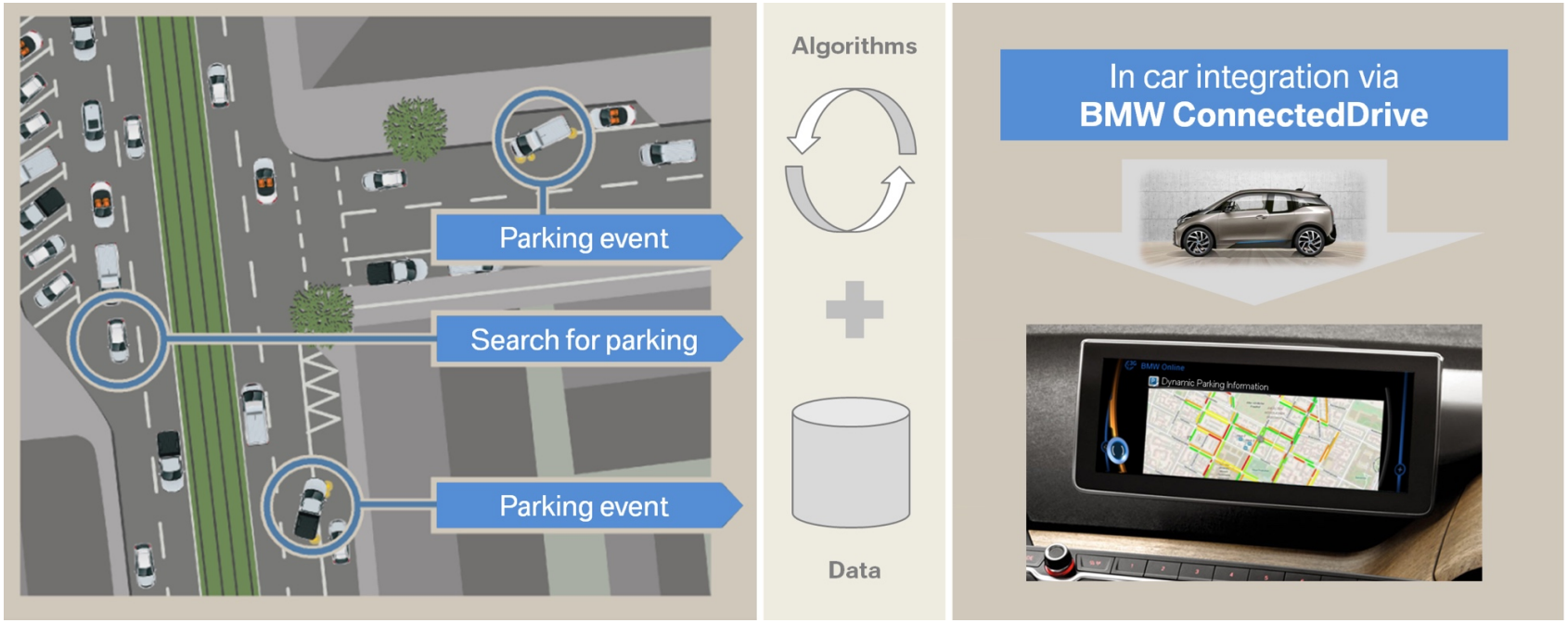
Fayazi/Vahidi, Delay Analysis with Application in Estimating the Traffic Signal Phase and Timing from Low-Frequency Vehicular Probe Data

A Car or a Computer on Four Wheels?

http://www.kpcb.com/insights/2013-internet-trends, http://www.wired.com/2014/01/bmw-builds-self-drifting-car/

Connected Vehicle

Connected Vehicle

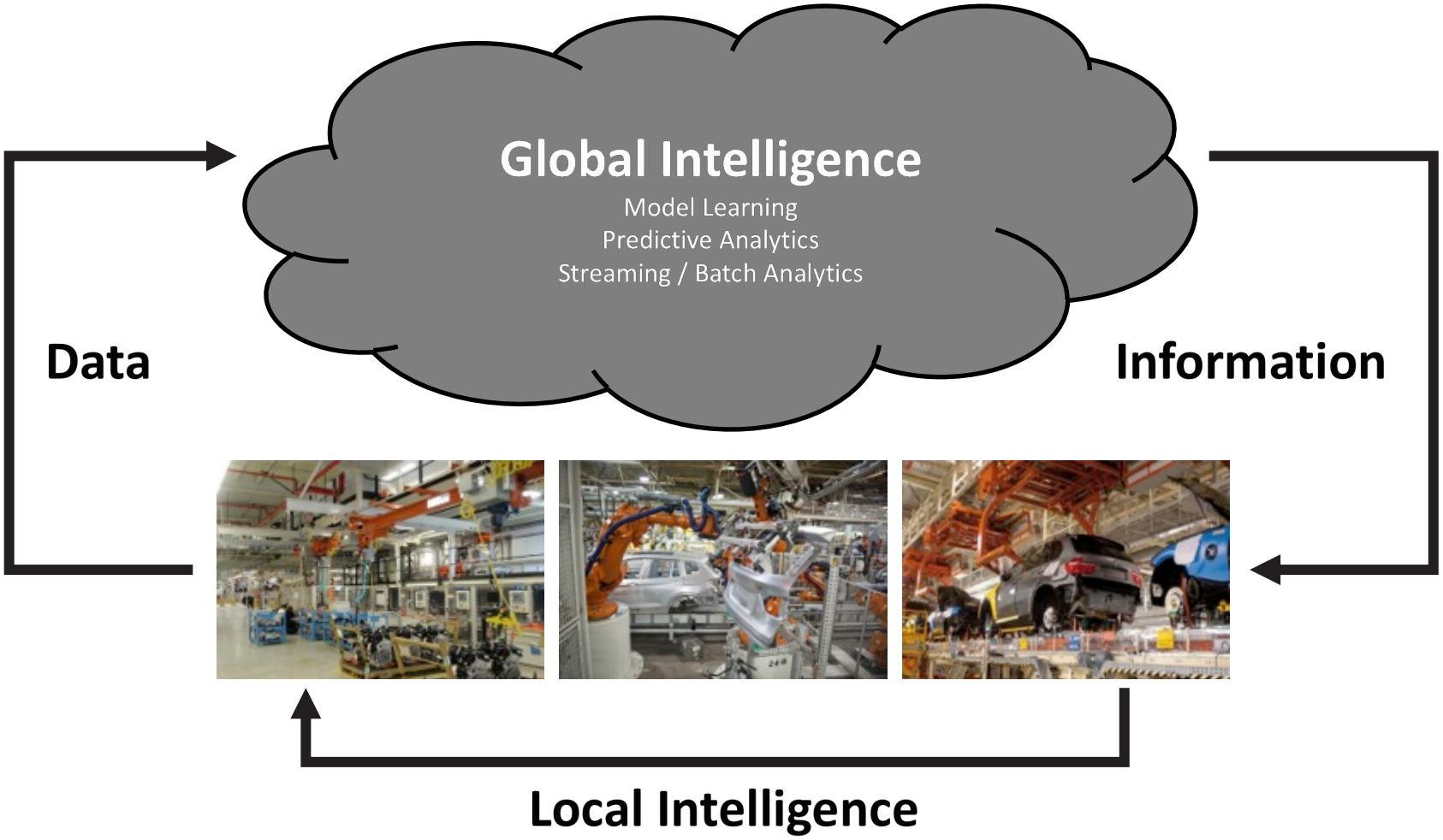


http://www.press.bmwgroup.com/deutschland/pressDetail.html?title=bmw-connecteddrive-entwickelt-lösungen-zur-intelligenten-parkplatzsuche-straßenrandseitige

Industrial Internet

- **The Industrial Internet describes the proliferation of connected machines in industrial environments and the resulting ability to optimize global operations using data generated by sensors during the production process (adapted from GE, Gartner).**
- Related Terms: Industry 4.0 (German federal initiative), IT/Operational Technology convergence (Gartner)

Industrial Internet



Other Data Applications

- Consumer Services:
  - Web search, recommendation Engines (Amazon, Netflix), Social networks, Video analytics (YouTube), Internet of Things (NEST, wearables, fitness tracker, connected vehicles,…)
- Industrial Manufacturing:
  - Supply Chain and Logistics, Assembly Quality, Smart Machines
- Government:
  - Census, Archiving, Image surveillance, Situation Assessment

See: http://grids.ucs.indiana.edu/ptliupages/publications/NISTUseCase.pdf

Characterizing Big Data – 3V's.

- ***Volume:*** Industry sources estimate that on average about 480 TB of data were collected by every automotive manufacturer in 2013. It is expected that this size is going to increase to 11.1 PB per year in 2020 (IHS).
- ***Velocity:*** Data ingest rates and processing requirements vary widely, from batch-oriented data derived from service events to real-time event processing, inducing high requirements on a data infrastructure.
- ***Variety:*** Heterogeneous data sets (structured and unstructured) generated by different sources stored in different formats impose significant challenges during the ingest process, and also motivate sophisticated analytics approaches ranging from descriptive statistics and visualizations to complex predictions.

Doug Laney, Gartner, http://blogs.gartner.com/doug-laney/deja-vvvue-others-claiming-gartners-volume-velocity-variety-construct-for-big-data/

Data versus Insight

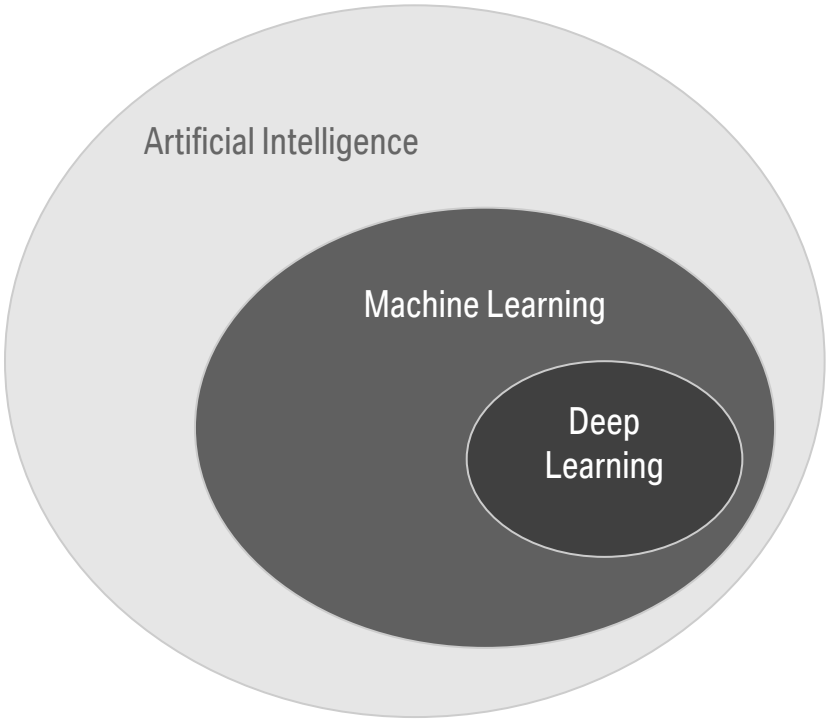
Data and information

Insight and action

Data, and the ability to analyze and apply data, is central to how firms compete and collaborate.

Source: Gartner

Machine Learning and AI



The term Artificial Intelligence (AI) was coined by John McCarthy and refers to the science and engineering of building intelligent machines mimicking human intelligence.

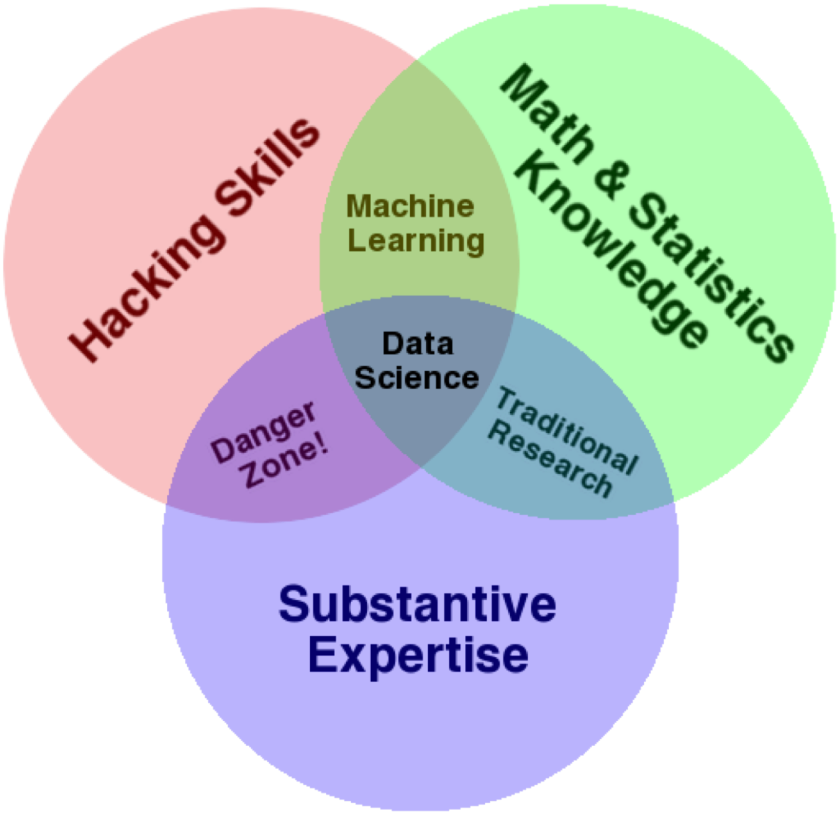
AI is a high-level term used for a broad set of concepts, in particular for systems that use machine learning and deep learning for advanced pattern recognition and prediction.

[1] John McCarthy. What is artificial intelligence? http://jmc.stanford.edu/articles/ whatisai/whatisai.pdf, 2007.

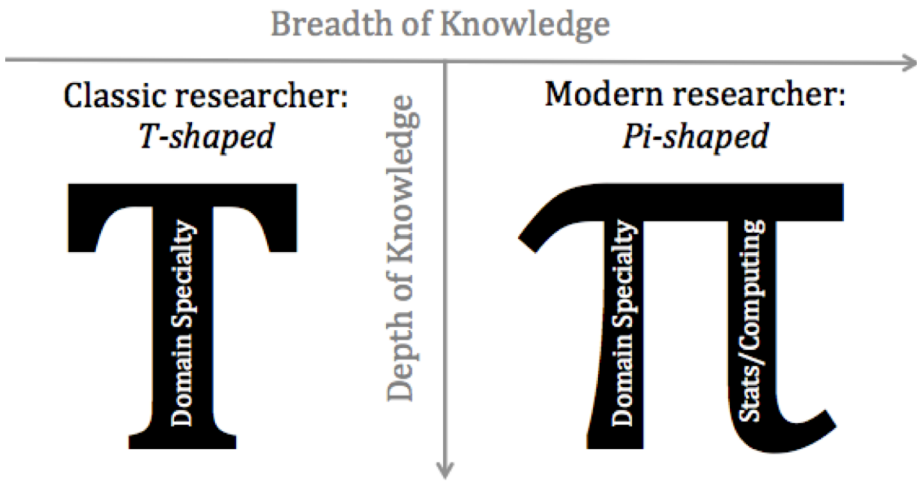
What is Data Science?

- **Data Science** is the extraction of actionable knowledge directly from data through a process of discovery, hypothesis, and analytical hypothesis analysis.
- A **Data Scientist** is a practitioner who has sufficient knowledge of the overlapping regimes of expertise in business needs, domain knowledge, analytical skills and programming expertise to manage the end-to-end scientific method process through each stage in the big data lifecycle.

Data Science in Industry and Science

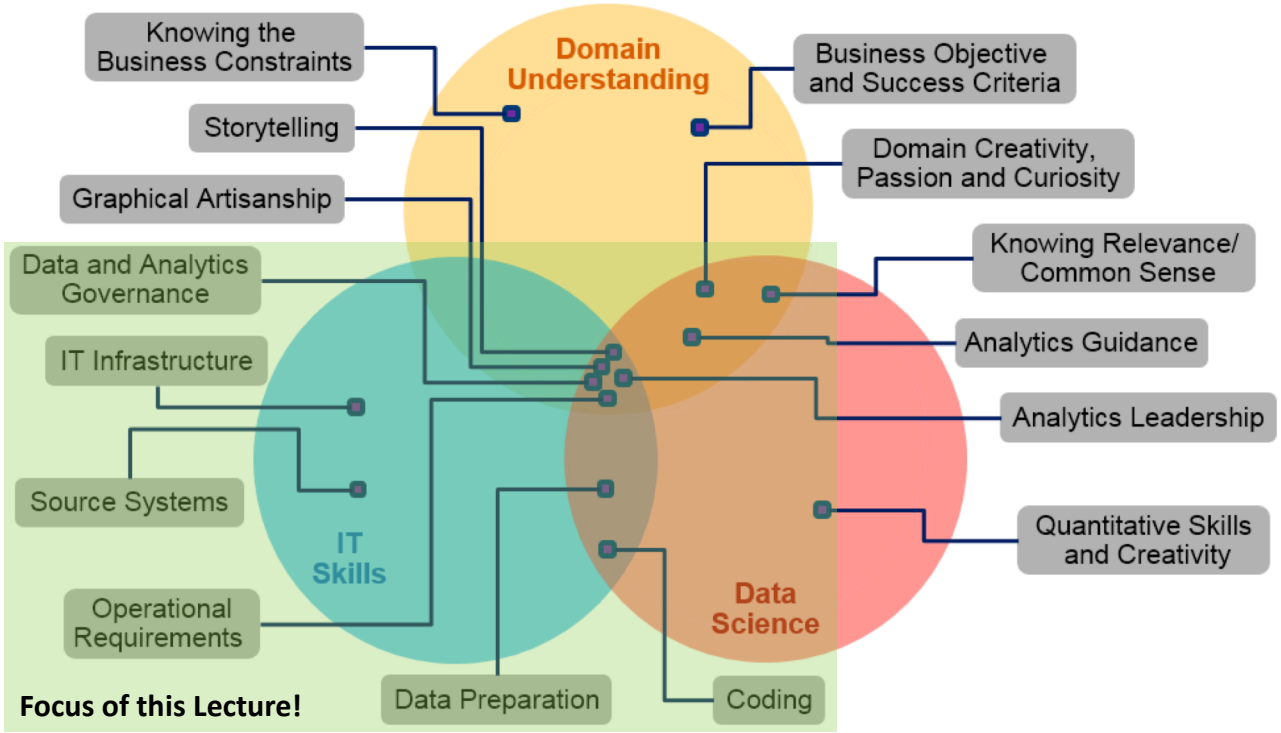


Drew Conway, Data Science Definition



Jake VanderPlas, Hacking Academia: Data Science and the University

Data Science Skills



Source: Gartner

What will you learn in this course?

- Tools vs. **Abstraction**
- Desktop vs. **HPC/Cloud**
- **Data Structure** vs. Statistics
  - “80% of statistics is sums and averages” Aaron Kimball

- **Skills:**
  - Data Engineering: data preparation, manipulation and cleaning
  - Running machine learning models
  - Scale machine learning models
  - Interpreting model output

Introduction to Parallel and Distributed Computing

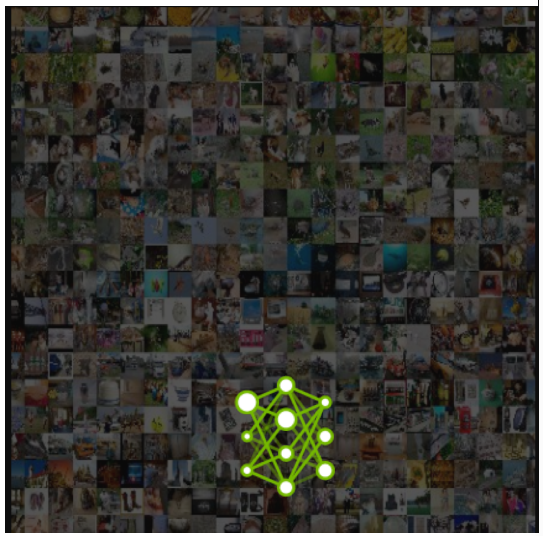
Adapted from:

Bettina Schnor: Paralleles Rechnen I and II, https://www.cs.uni-potsdam.de/bs/teaching/docs/courses/ss2017/pr/

Karl Fuerlinger: Parallel and High Performance Computing, http://www.nm.ifi.lmu.de/teaching/Vorlesungen/2017ws/parallel/

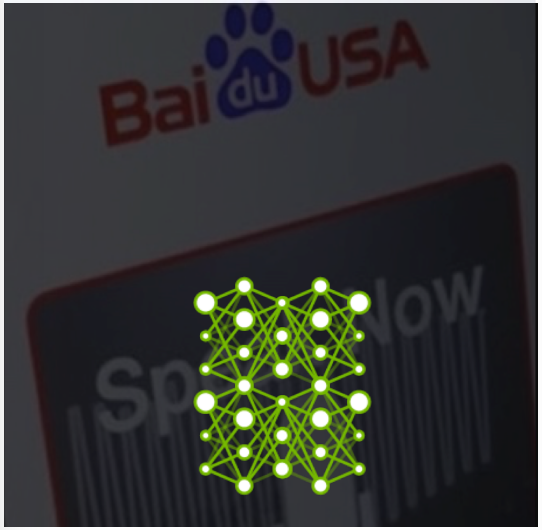
Big Data needs Big Compute (1)

7 ExaFLOPS
60 Million Parameters



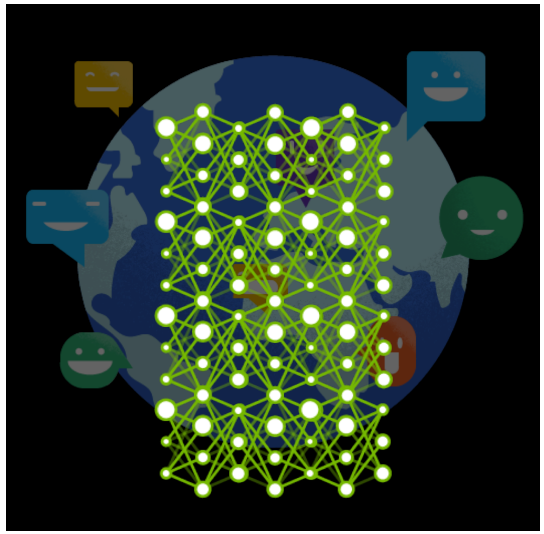
2015 – Microsoft ResNet
Superhuman Image Recognition

20 ExaFLOPS
300 Million Parameters



2016 – Baidu Deep Speech 2
Superhuman Voice Recognition

100 ExaFLOPS
8.7 Billion Parameters



2017 – Google Neural Machine Translation
Near Human Language Translation

Big Data needs Big Compute (2)

Important Property of Neural Networks

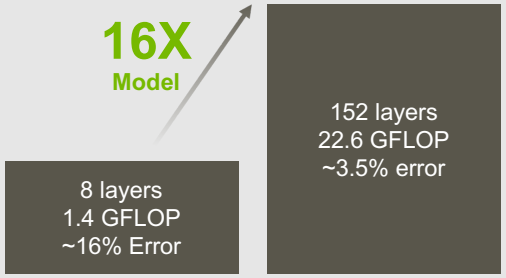
Results get better with

more data +
bigger models +
more computation

(Better algorithms, new insights and improved techniques always help, too!)

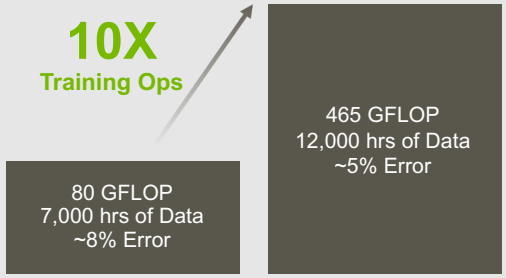
Google

IMAGE RECOGNITION



Microsoft

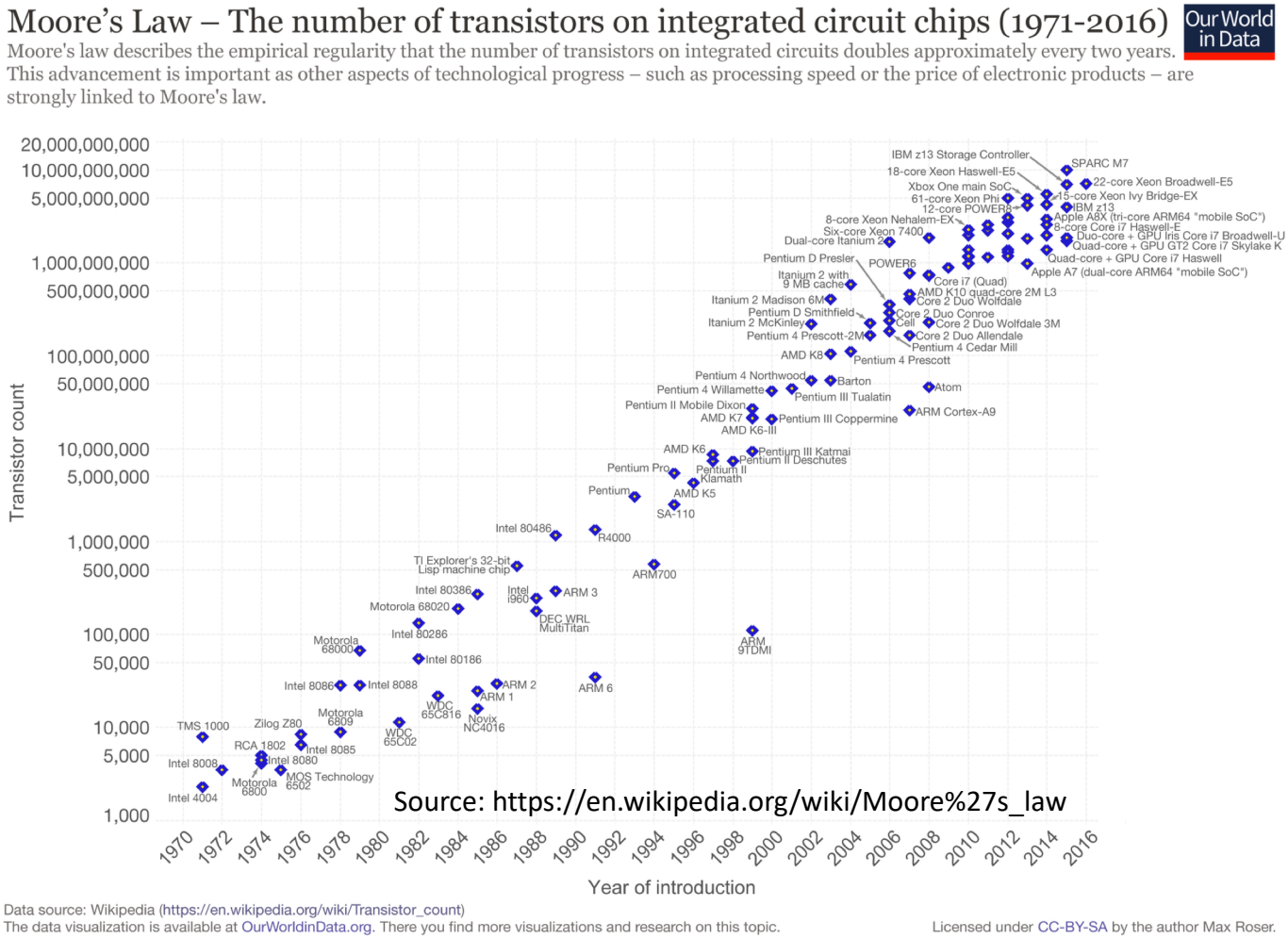
SPEECH RECOGNITION

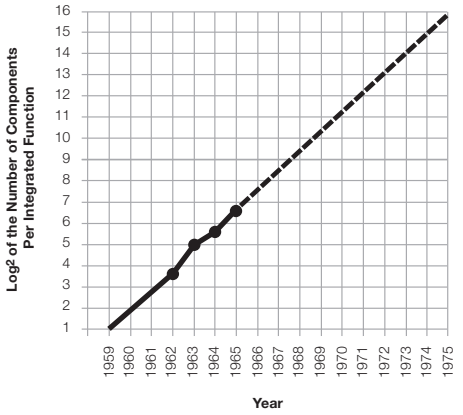


Baidu 百度

Bill Dally, Scaling Machine Learning, ScaleML 2018, https://www.matroid.com/scaledml/2018/bill.pdf

Motivation and Moore's Law (1)



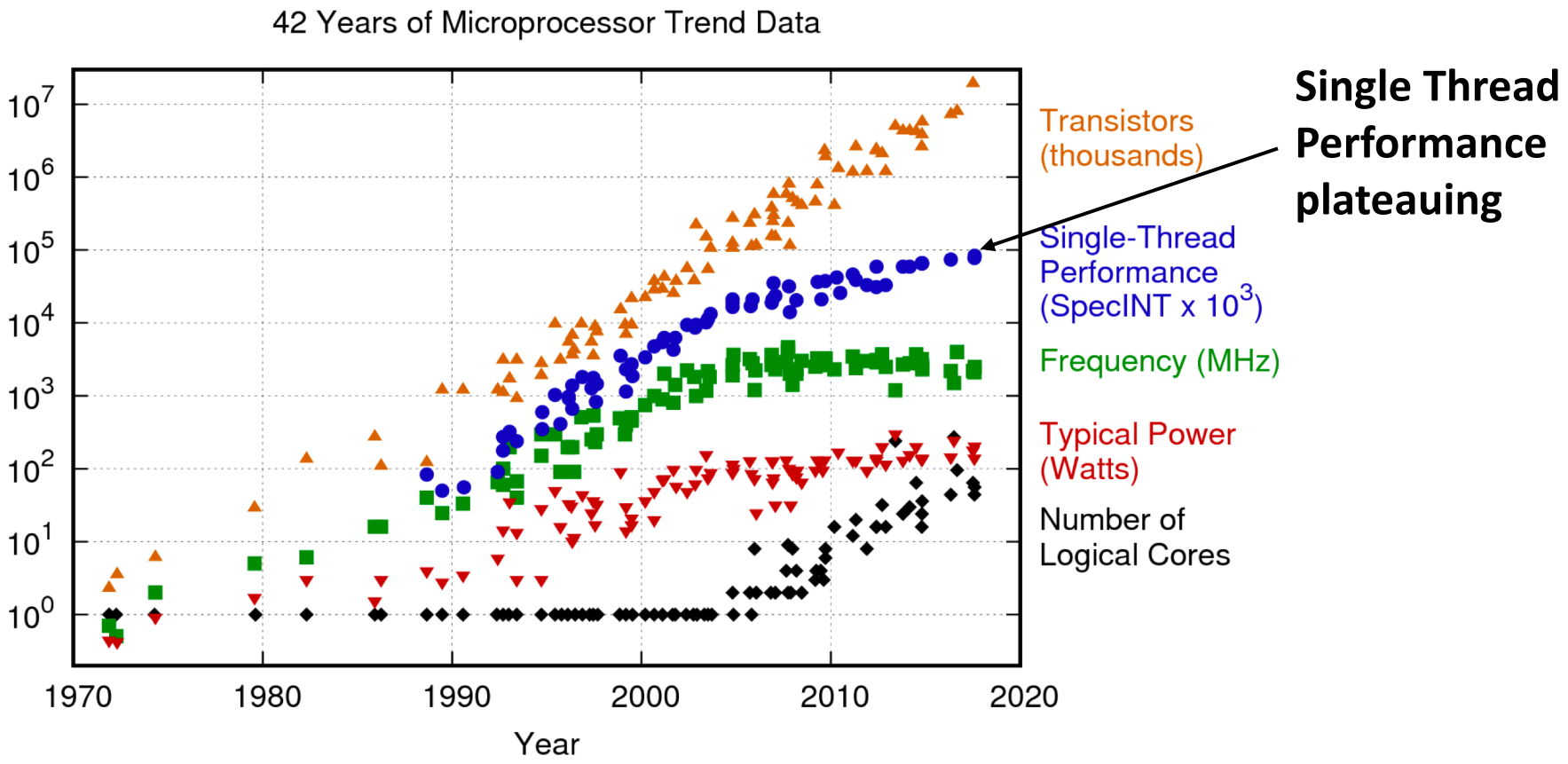


Gordon Moore, Cramming more components onto integrated circuits, 1965

Will computers be eventually just fast enough?

The number of transistors in integrated circuits doubles approximately every two years.

Motivation and Moore's Law (2)



Karl Rupp, 42 Years of Microprocessor Trend Data,
https://www.karlrupp.net/2018/02/42-years-of-microprocessor-trend-data/, 2018

Moore's Law and Motivation

- Clock speed of integrated circuit chips stagnates since a couple of years
- There are fundamental physical limits:
  - Speed of light = 300000 km/sec = 30 x 109 cm/sec
  - Need at least 1 atom to store 1 bit
- Data and machine learning not just requires compute, but also Memory and I/O (which is more difficult to scale)!

Parallel computing enables us to overcome these issues!

Why to parallelize and distribute?

- **Performance**: increased peak performance, higher throughput, shorter time-to-solution
  - Solve a problem faster (time to solution)
  - Solve a larger problem in the same amount of time (quality of the result)
  - Run many instances of the same program to improve error bounds (e.g., ensemble studies of simulation runs)
- **Economics:** Scale-out to commodity hardware more economically than scale-up.
- **Availability and Reliability:** Use of redundancy to improve availability and overall reliability.

Distributed Computing Defintions

- A **distributed system** is a collection of independent computers that appear to the users of the system as a single coherent system (Tanenbaum & Enslow).
- A **parallel computer** is a set of processors that are able to work cooperatively to solve a computational problem (Ian Forster et al)

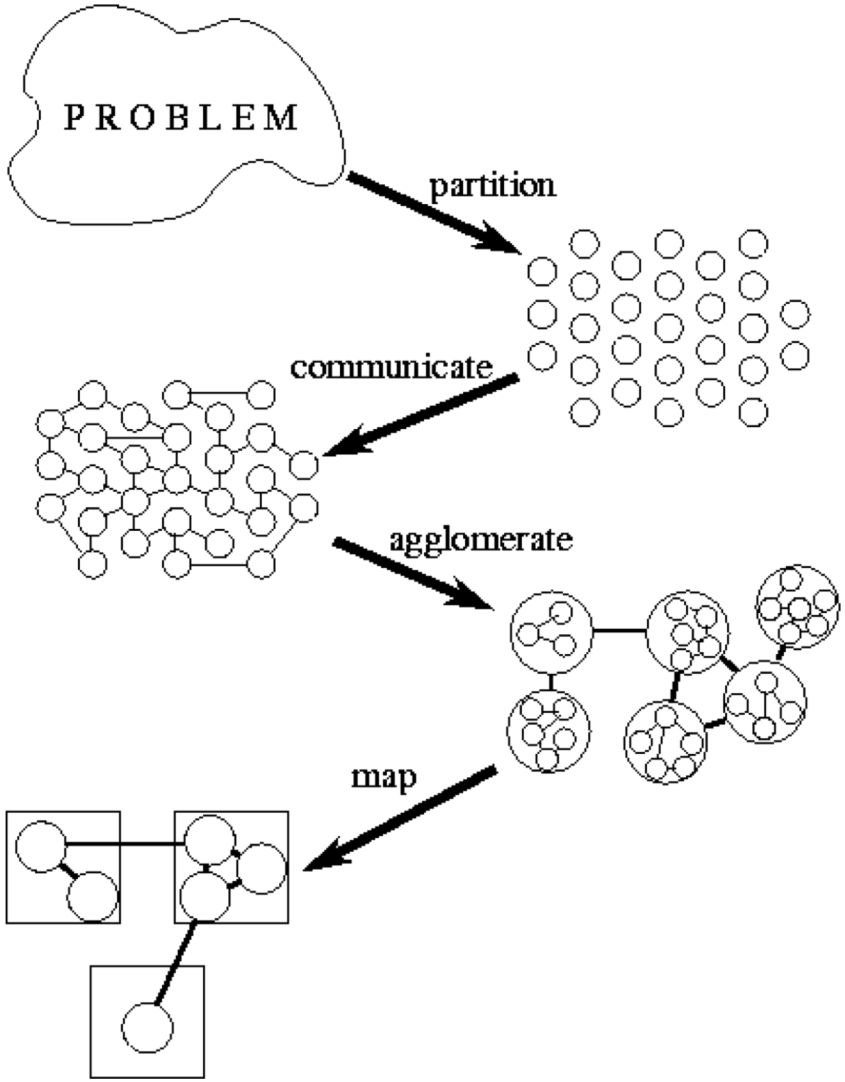
Distributed vs. Parallel Computing

- Often no strict distinction made between parallel and distributed computing
- **Parallel Computing:**
  - High-Performance focused
  - Tightly coupled. Components are used at the same time to achieve a specific goal
- **Distributed Computing:**
  - Usually refers to more loosely coupled systems
  - No shared memory, processed communicate via message passing
  - High heterogeneity and dynamism
  - Failure occur often; difficult to detect
  - Independent systems providing a service
  - Often geographically distributed

Types of Parallelism

- Parallelism exists in several forms (even in single-core CPUs)
  - Partially hidden from the programmer/compiler
  - Partially configurable/tunable
  - Partially explicit programming required
- **Instruction Level Parallelism (ILP)**
  - Hardware tries to discover parallelism in the instruction stream
- **Data Level Parallelism (DLP)**
  - Often same operation on different data items can be run in parallel
  - Vector processing (supercomputers) and SIMD in modern CPUs
  - MapReduce, Data-Flow-oriented parallel processing
- **Task-Level Parallelism (TLP)**
  - Creation whole subprograms that can be executed in parallel
  - Programmer specifies independent threads of control
  - Functional Parallelism: Subtasks execute different code on different data items

Parallel Programming: PCAM Model



Partition: Decompose work into tasks that can be executed concurrently

Communicate: Design communication structure depending on which data needs to be exchange how often.

Agglomeration: Aggregate tasks to optimize performance.

Map: Map tasks to processes.

Source: Ian Foster, Designing and Bulding Parallel Programs, http://www.mcs.anl.gov/~itf/dbpp/

Parallel Programming Models (1)

- **Shared Memory.** In the shared-memory programming model, tasks share a common address space, which they read and write asynchronously.
  - Various mechanisms such as locks and semaphores may be used to control access to the shared memory. An advantage of this model from the programmer's point of view is that the notion of data ``ownership'' is lacking, and hence there is no need to specify explicitly the communication of data from producers to consumers.
  - Example: Threads
    - Threads can be created dynamically (mid-execution) in some models
    - Each thread has a set of private variables, e.g., local stack variables Also a set of shared variables, e.g., static and global variables, shared common blocks, or global heap.
    - Threads communicate implicitly by writing and reading shared variables.

Parallel Programming Models (2)

- **Message passing:** Message-passing programs create multiple tasks, with each task encapsulating local data. Each task is identified by a unique name, and tasks interact by sending and receiving messages to and from named tasks.
  - These systems are said to implement a *single program multiple data* (SPMD) programming model because each task executes the same program but operates on different data.
- **Data Parallelism:** Exploitation of the concurrency that derives from the application of the same operation to multiple elements of a data structure (e.g. add 2 to all records).
  - A data-parallel program consists of a sequence of such operations. As each operation on each data element can be thought of as an independent task, the natural granularity of a data-parallel computation is small. Data locality is key! Data partitioning and co-locating tasks to data critical. Well supported in frameworks, such as Hadoop and Spark

MPI: What is MPI?

- A *message-passing library specification*
  - extended message-passing model
  - not a language or compiler specification
  - not a specific implementation or product
- For parallel computers, clusters, and heterogeneous networks
- Designed to provide access to advanced parallel hardware for
  - end users
  - library writers
  - tool developers

MPI: Programming Model

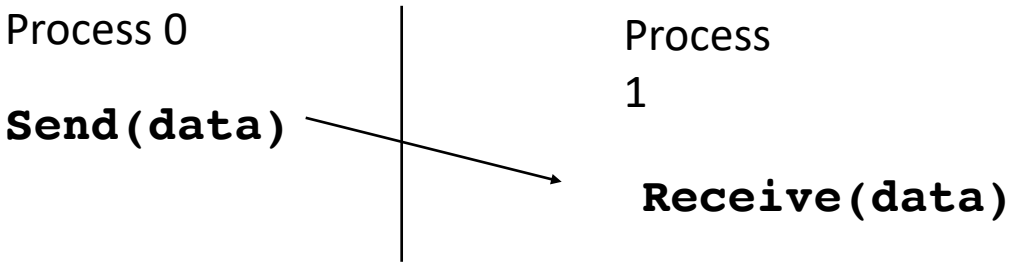
- A *process* is (traditionally) a program counter and address space.
  - Processes may have multiple *threads* (program counters and associated stacks) sharing a single address space. MPI is for communication among processes, which have separate address spaces.
  - Interprocess communication consists of synchronization and movement of data from one process's address space to another's

- An MPI Program consists of a collection of processes.
  - Usually fixed at program startup time. MPI-2 added support for dynamic processes
  - No shared data.
  - Typically shared data is partitioned over local processes.

Ewin Lusk and William Gropp, An Introduction to MPI Parallel Programming with the Message Passing Interface, http://www.mcs.anl.gov/research/projects/mpi/tutorial/mpiintro/MPIIntro.PPT
Karl Fuerlinger: Parallel and High Performance Computing, http://www.nm.ifi.lmu.de/teaching/Vorlesungen/2017ws/parallel/

MPI: Cooperative Operations for Communication

- The message-passing approach makes the exchange of data cooperative.
- Data is explicitly sent by one process and received by another.
- An advantage is that any change in the receiving process's memory is made with the receiver's explicit participation.
- Communication and synchronization are combined.



MPI: Collective Operations

- Collective operations are called by all processes in a communicator.
- **MPI_BCAST** distributes data from one process (the root) to all others in a communicator.
- **MPI_REDUCE** combines data from all processes in communicator and returns it to one process.
- In many numerical algorithms, **SEND/RECEIVE** can be replaced by **BCAST/REDUCE**, improving both simplicity and efficiency.

A Minimal MPI Program (C)


```
#include "mpi.h"
#include <stdio.h>

int main( int argc, char *argv[] )
{
    MPI_Init( &argc, &argv );
    printf( "Hello, world!\n" );
    MPI_Finalize();
    return 0;
}
```

MapReduce


```
map(String key, String value):
  // key: document name
  // value: document contents
  for each word w in value:
    EmitIntermediate(w, "1");

reduce(String key, Iterator values):
  // key: a word
  // values: a list of counts
  int result = 0;
  for each v in values:
    result += ParseInt(v);
  Emit(AsString(result));
```

map	(k1,v1)	→ list(k2,v2)
reduce	(k2,list(v2))	→ list(v2)

Users write a *client application* that submits one or more *jobs* that contain user-supplied map and reduce code and a job configuration file to a cluster of machines.

The job contains a *map* function and a *reduce* function, along with *job configuration* information that controls various aspects of its execution.

Properties:

- Simplicity: no socket programming, no threading or fancy synchronization logic, no management of retries, no special techniques to deal with enormous amounts of data
- Automatic parallelization and distribution of work
- Fault tolerance: Failure is not an exception; it's the norm. MapReduce treats failure as a first-class citizen and supports re-execution of failed tasks on healthy worker nodes in the cluster.

Jeff Dean et al., MapReduce: Simplified Data Processing on Large Clusters, https://static.googleusercontent.com/media/research.google.com/en//archive/mapreduce-osdi04.pdf, 2004.
Eric Sammer, Hadoop Operations, O'Reilly, 2012

Parallel Application Patterns

- **Embarrassingly Parallel:**
  - Independent Tasks without need for communication
- **Master/Worker:**
  - One compute element for administrative tasks while all others solve the actual problem is called the master-worker
  - The master distributes work and collects results
  - Master potential bottleneck
  - Pattern used in many data-intensive applications
- **Further patterns:**
  - Grid-based simulations: a grid that defines discrete positions for the physical quantities under consideration. Mathematical transformations conducted on each grid position considering neighboring cells
  - Divide-n-Conquer, Shared Queue, Recursive Splitting

Performance of Parallel Applications: Speedup and Efficiency

n : = Number Processes

$T(1)$: = Runtime of sequential program

$T(n)$: = Runtime of parallel program on n-processes

- *Speedup:*

$$S(n) = \frac{T(1)}{T(n)}$$

- Efficiency:

$$E(n) = \frac{S(n)}{n}$$

- Strong Scaling: Problem size remains constant. Number of processors varied.
- Super-linear Speed-up: $S(n) > n$.

Performance of Parallel Applications: Amdahl's Law

How much faster can a given problem be solved with *N* workers instead of one? How much more *work* can be done with *N* workers instead of one?

In a idealistic world: using *N* processes for a problem that takes a time *T* to be solved sequentially will now ideally take only *T* /*N*

Runtime:

$$T = s + p$$

s: sequential part
p: parallel part

Parallel runtime:

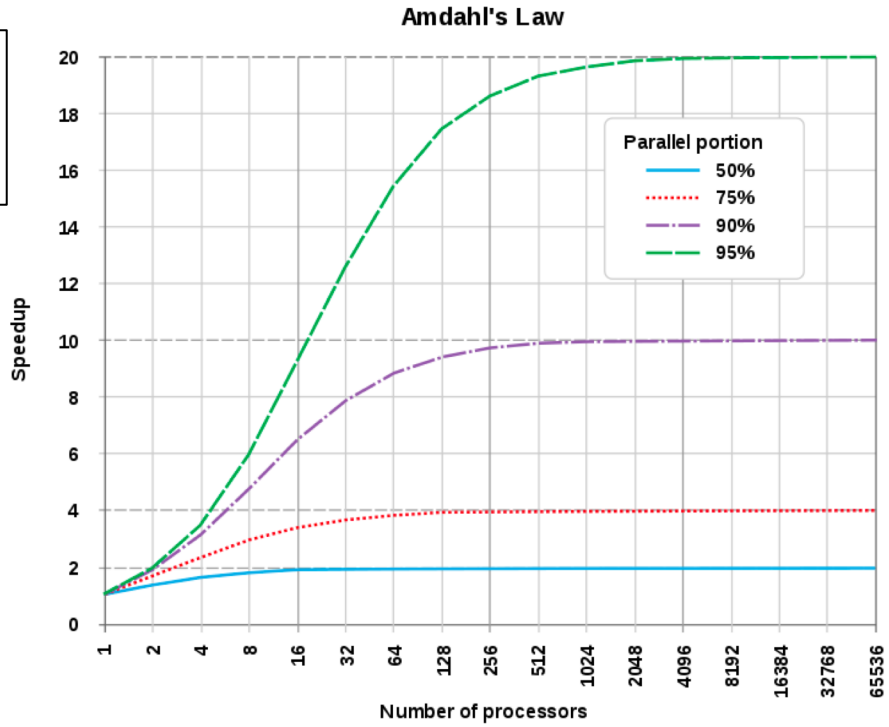
$$T(n) = s + \frac{p}{n}$$

Performance of Parallel Applications: Amdahl's Law

Amdahl's Law: The application speedup for constant size workloads is limited to the inverse of the sequential part of the application

$$S(n) = \frac{1}{s + \frac{1-s}{n}}$$

$$S(n) = \frac{1}{s}$$



Weak Scaling: How much more work can my program do in a given amount of time when I put a larger problem on *N* CPUs?

Given the results of Amdahl's law, weak scaling is very important!

https://en.wikipedia.org/wiki/Amdahl%27s_law

G. M. Amdahl. Validity of the single processor approach to achieving large scale computing capabilities. In: AFIPS '67 (Spring): Proceedings of the April 18-20, 1967, Spring Joint Computer Conference (ACM, New York, NY, USA), 483–485.

Strong versus Weak Scaling

- **Strong Scaling:** The **amount of work stays constant** no matter how many workers are used. The goal of parallelization is minimization of time to solution for a given problem.
- **Weak Scaling:** Time to solution is not the primary objective, but **larger problem sizes** (for which e.g. available memory is the limiting factor) are of interest.

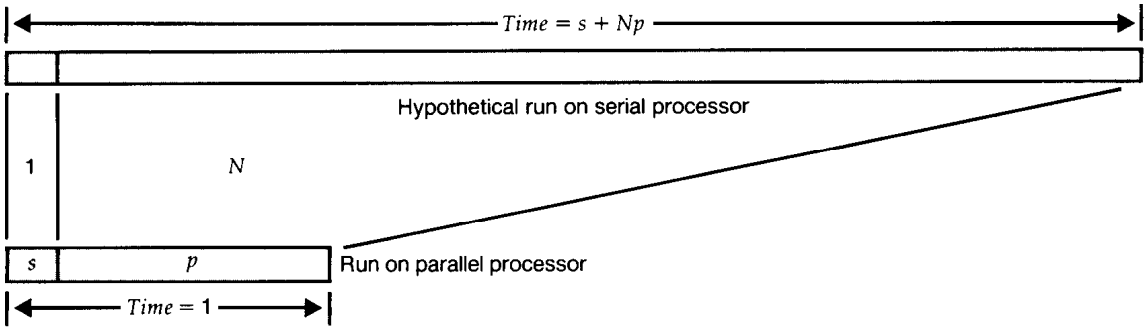


FIGURE 2b. Scaled-Sized Model for $Speedup = s + Np$

Gustafson's Law

J. L. Gustafson. Reevaluating Amdahl's law. Commun. ACM 31(5), (1988) 532–533. ISSN 0001-0782.
Georg Hager and Gerhard Wellein, Introduction to High Performance Computing for Scientists and Engineers, CRC Press, 2011.

L1 cache reference	0.5 ns
Branch mispredict	5 ns
L2 cache reference	7 ns
Mutex lock/unlock	25 ns
Main memory reference	100 ns
Compress 1K bytes with Zippy	3,000 ns
Send 2K bytes over 1 Gbps network	20,000 ns
Read 1 MB sequentially from memory	**250,000 ns**
Round trip within same datacenter	500,000 ns
Disk seek	**10,000,000 ns**
Read 1 MB sequentially from disk	**20,000,000 ns**
Send packet USA->Germany->USA	150,000,000 ns

Source: Jeff Dean, Google

Why is Distributed Computing hard?

Peter Deutsch/James Gosling's 8 Fallacies of Distributed Computing:

- The network is reliable.
- Latency is zero.
- Bandwidth is infinite.
- The network is secure.
- Topology doesn't change.
- There is one administrator.
- Transport cost is zero.
- The network is homogeneous.

Arnon Rotem-Gal-Oz, Fallacies of Distributed Computing Explained, http://www.rgoarchitects.com/Files/fallacies.pdf

Fault Tolerance.

- “A distributed system is a system on which I cannot get any work done, because some machine I have never heard of has crashed.” (Leslie Lamport)

- Challenges:
  - Processes may fail in different ways
  - There is no way to reliable detect the failure of a process

- Failures are the norm rather than the exception in large-scale systems.

The Joys of Real Hardware

Typical first year for a new cluster:

~0.5 overheating (power down most machines in <5 mins, ~1-2 days to recover)
~1 PDU failure (~500-1000 machines suddenly disappear, ~6 hours to come back)
~1 rack-move (plenty of warning, ~500-1000 machines powered down, ~6 hours)
~1 network rewiring (rolling ~5% of machines down over 2-day span)
~20 rack failures (40-80 machines instantly disappear, 1-6 hours to get back)
~5 racks go wonky (40-80 machines see 50% packetloss)
~8 network maintenances (4 might cause ~30-minute random connectivity losses)
~12 router reloads (takes out DNS and external vips for a couple minutes)
~3 router failures (have to immediately pull traffic for an hour)
~dozens of minor 30-second blips for dns
~1000 individual machine failures
~thousands of hard drive failures
slow disks, bad memory, misconfigured machines, flaky machines, etc.

Long distance links: wild dogs, sharks, dead horses, drunken hunters, etc.

Source: Jeff Dean, Google

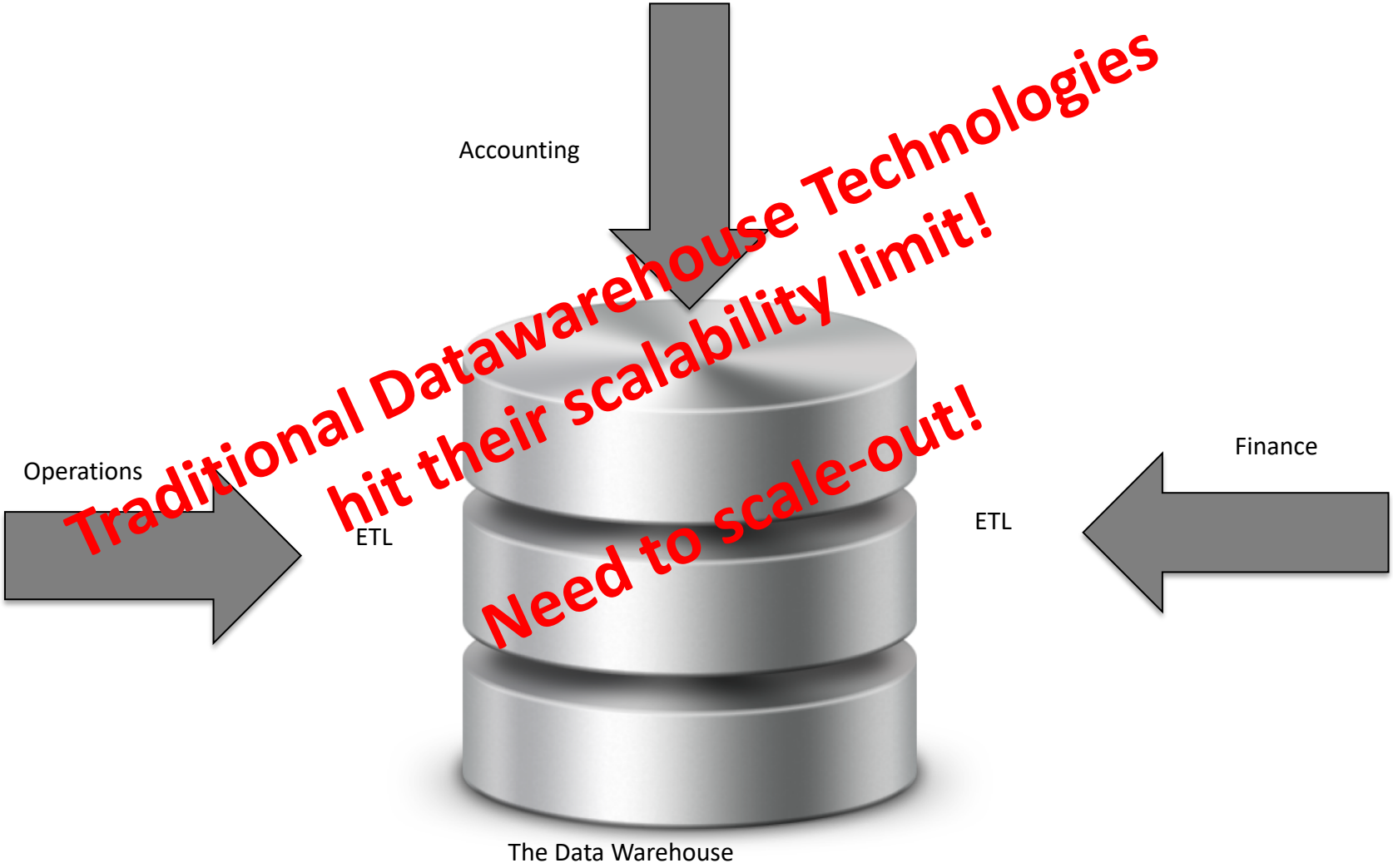
Consistency: Every read receives the most recent write or an error

Availability: Every request receives a (non-error) response – without guarantee that it contains the most recent write

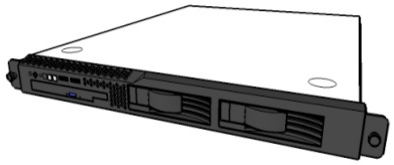
Partition Tolerance: The system continues to operate despite an arbitrary number of messages being dropped (or delayed) by the network between nodes

These properties are conflicting. Trade-offs need to be considered!

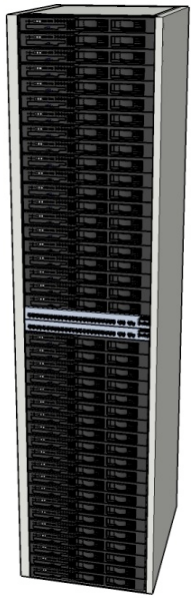
Traditional Approaches?



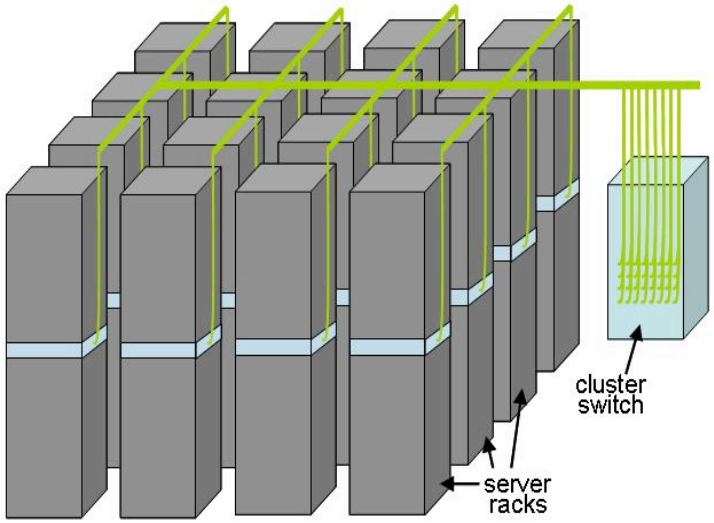
Google's Style Distributed Computing.



Servers
- CPUs
- DRAM
- Disks



Racks
- 40-80 servers
- Ethernet switch



Clusters

"Datacenter as computer" "Warehouse-scale Machines"

Jeff Dean, http://www.cs.cornell.edu/projects/ladis2009/talks/dean-keynote-ladis2009.pdf

Why is parallelism hard?

- Finding enough parallelism: Dealing with Amdahl's Law
- Granularity: Finding the right units of parallel execution
- Dependencies: Preserving sequential semantics
- Coordination and synchronization: Orchestrating the parallel execution
- Locality and Load Balance: Important for performance
- Performance measurement and modeling: How to perform close to optimal? What part of application is worth optimization?

Limitations of HPC for Big Data

- An increasing number of HPC application requires the processing of large input data
- Processing data with HPC tools is hard: For example, to process a large file full of DNA-sequencing reads in parallel, we must manually split it up into smaller files and submit a job for each of those files to the cluster scheduler.
- Tools written for HPC environments often fail to decouple the in-memory data models from the lower-level storage models
- Many tools only know how to read data from a POSIX filesystem in a single stream, making it difficult to make tools naturally parallelize
- Storage backends, like databases, make it difficult to scale compute
- Recent systems in the Hadoop ecosystem provide abstractions that allow users to treat a cluster of computers more like a single computer—to automatically split up files and distribute storage over many machines, divide work into smaller tasks and execute them in a distributed manner, and recover from failures.
- The Hadoop ecosystem can automate a lot of the hassle of working with large data sets, and is far cheaper than HPC.

Cyberinfrastructure (HPC, Grid)

- **Cyberinfrastructure** is infrastructure that supports distributed research and learning: Links data, people, computers (NSF)
- Cyberinfrastructure consists of computing systems, data storage systems, advanced instruments and data repositories, visualization environments, and people, all linked together by software and high performance networks to improve research productivity and enable breakthroughs not otherwise possible (Craig Steward)
- Also referred to as Grid, e-Science
- Examples: XSEDE, EGI

Cloud Computing is a large-scale distributed computing paradigm that is driven by economies of scale, in which a pool of abstracted, virtualized, dynamically-scalable, managed computing power, storage, platforms, and services are delivered on demand to external customers over the Internet (Foster).

Example: Amazon Web Services, Google Cloud, Microsoft Azure

"A pool of abstracted **highly scalable**, and **managed compute infrastructure** capable of **hosting end-customer applications** and **billed by consumption**. "

(Forrester Research)

"A style of computing where massively scalable **IT-related capabilities** are provided '**as a service**' across the Internet to multiple external customers."

(Gartner)

Hadoop and MapReduce

Adapted from:

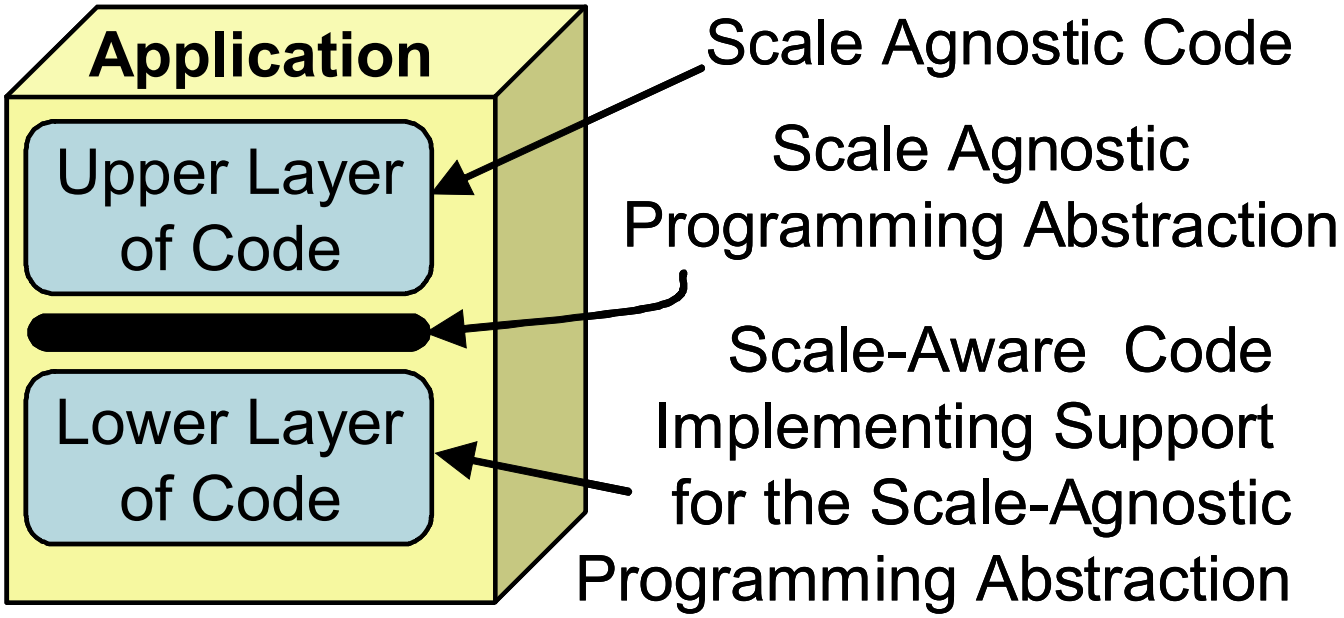
T. White, Hadoop: The Definitive Guide, O'Reilly 2012-2018

- **Distributed Applications:** Distributed applications are those that (a) need multiple resources, or (b) would benefit from the use of multiple resources; examples of how they could benefit include, increased peak time to solution or reliability.
- Developing distributed applications is hard!
  - Heterogeneous middleware, point solutions, usability, …
  - Occupied by plumbing; insufficient reasoning about when/how to distribute
- Distributed computing at scale needs performance, flexibility, and extensibility

- **Abstraction:** An abstraction is any process, mechanism or infrastructure to support a commonly occurring usage (e.g., computation, communication, and/or composition). Jha et. al., Distributed Programming Abstractions

- Abstractions help to mitigate the complexities of distributed systems.

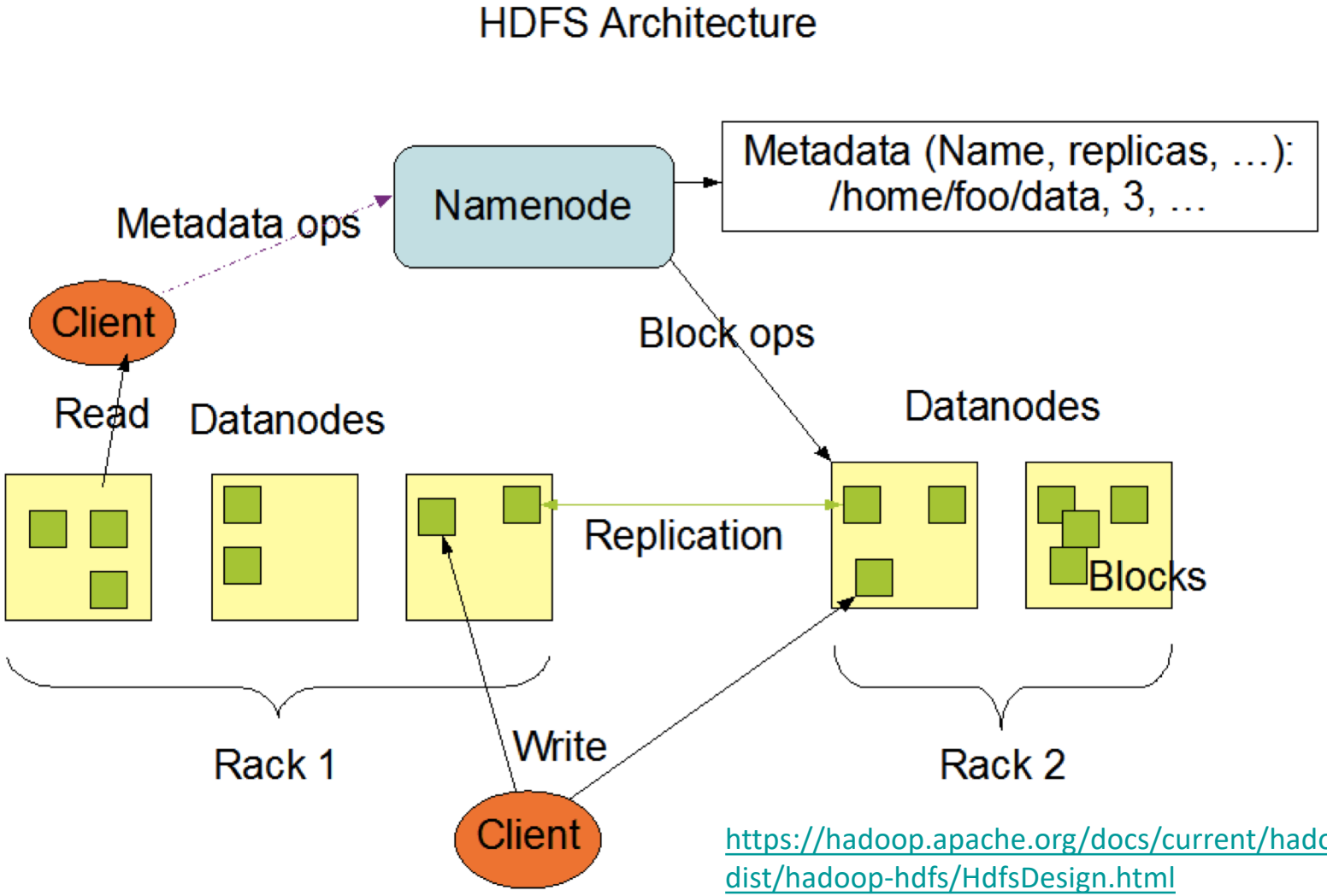
- What are scalable distributed programming abstractions?


Application
Upper Layer of Code
Lower Layer of Code
Scale Agnostic Code
Scale Agnostic Programming Abstraction
Scale-Aware Code Implementing Support for the Scale-Agnostic Programming Abstraction

- **MapReduce** is a programming abstraction designed for exploring large data sets:
  - **First phase:** map a user supplied function onto data distributed across multiple computers and generate a set of intermediate key/value pairs
  - **Second phase:** reduces the returned values from these map instances into a single result by merging all intermediate values associated with the same intermediate key
- **Hadoop** is a open source software infrastructure for support the **MapReduce** programming model.

- **Hadoop Distributed File System (HDFS):** distributed filesystem
- **YARN:** Yet Another Resource Negotiator
- **MapReduce:** Application Framework

Hadoop Filesystem



https://hadoop.apache.org/docs/current/hadoop-project-dist/hadoop-hdfs/HdfsDesign.html

- Jeffrey Dean and Sanjay Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In OSDI'04: Proceedings of the 6th conference on Symposium on Operating Systems Design & Implementation, pages 137-150, Berkeley, CA, USA, 2004. USENIX Association.
- Sanjay Ghemawat, Howard Gobioff, and Shun-Tak Leung. The Google File System. SIGOPS Oper. Syst. Rev., 37(5):29, 2003.
- Konstantin Shvachko, Hairong Kuang, Sanjay Radia, Robert Chansler, The Hadoop Filesystem, MSST'2010, https://dl.acm.org/citation.cfm?id=1914427
- Hadoop File System. http://hadoop.apache.org/common/docs/current/hdfs_design.html

Unix Philosophy:

- Write programs that do one thing and do it well
- Write programs to work together
- Write programs that handle text streams, because

that is a universal interface

The Unix Pipe | represents a universal abstraction for connecting programs

Unix: MapReduce

Read a HTTP access log file an determine the most frequent HTTP return codes

```
ubuntu@ip-10-186-164-81:~$ head /data/NASA_access_log_Jul95
199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
```

```
cat /data/NASA_access_log_Jul95 |
```

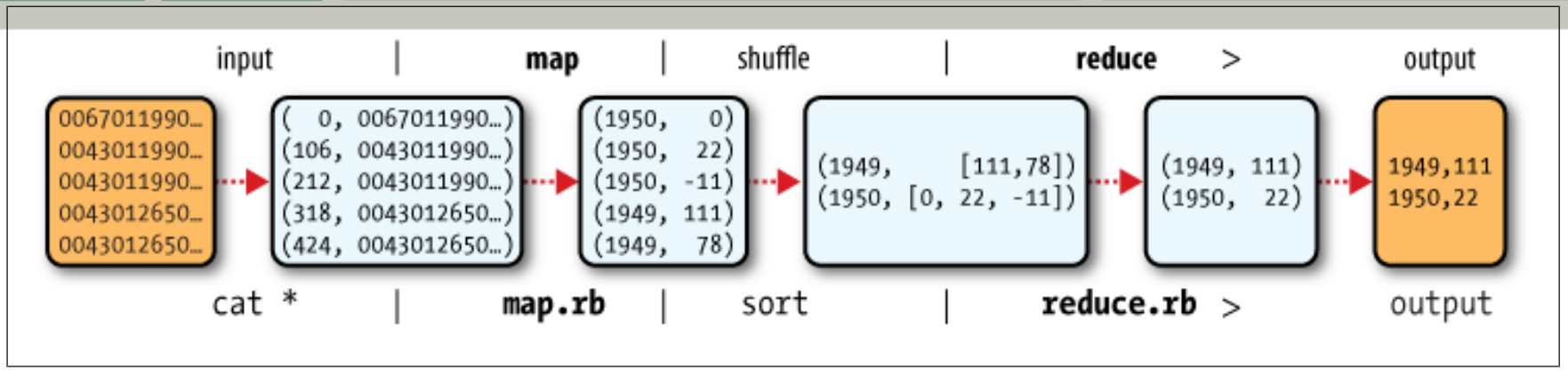
- **MapReduce** is a key/value-based API originally designed for processing text streams.
- Different file formats are supported via the **`InputFormat`** API. The `InputFormat` defines how input formats are partitioned and distributed to Mappers.
- The mapper tasks input is an **`InputSplit.`**

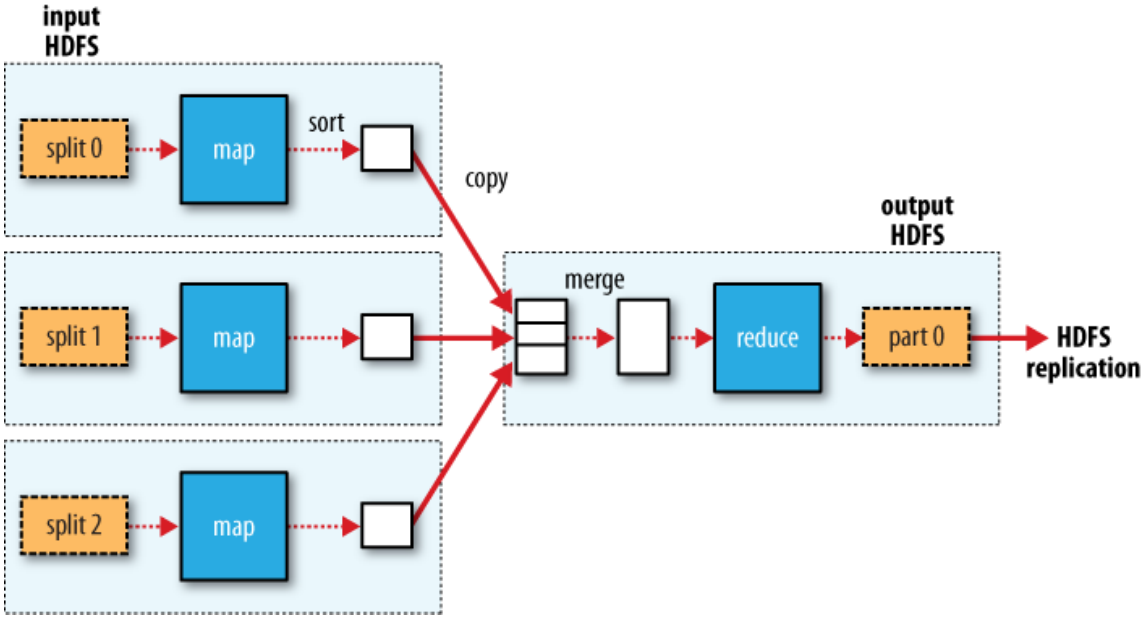
```
ubuntu@ip-10-186-164-81:~$ head /data/NASA_access_log_Jul95
199.72.81.55 - - [01/Jul/1995:00:00:01 -0400] "GET /history/apollo/ HTTP/1.0" 200 6245
unicomp6.unicomp.net - - [01/Jul/1995:00:00:06 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
199.120.110.21 - - [01/Jul/1995:00:00:09 -0400] "GET /shuttle/missions/sts-73/mission-sts-73.html HTTP/1.0" 200 4085
burger.letters.com - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/countdown/liftoff.html HTTP/1.0" 304 0
199.120.110.21 - - [01/Jul/1995:00:00:11 -0400] "GET /shuttle/missions/sts-73/sts-73-patch-small.gif HTTP/1.0" 200 4179
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /images/NASA-logosmall.gif HTTP/1.0" 304 0
burger.letters.com - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/video/livevideo.gif HTTP/1.0" 200 0
205.212.115.106 - - [01/Jul/1995:00:00:12 -0400] "GET /shuttle/countdown/countdown.html HTTP/1.0" 200 3985
d104.aa.net - - [01/Jul/1995:00:00:13 -0400] "GET /shuttle/countdown/ HTTP/1.0" 200 3985
129.94.144.152 - - [01/Jul/1995:00:00:13 -0400] "GET / HTTP/1.0" 200 7074
```

Input Split 1

Input Split 2

MapReduce in Detail

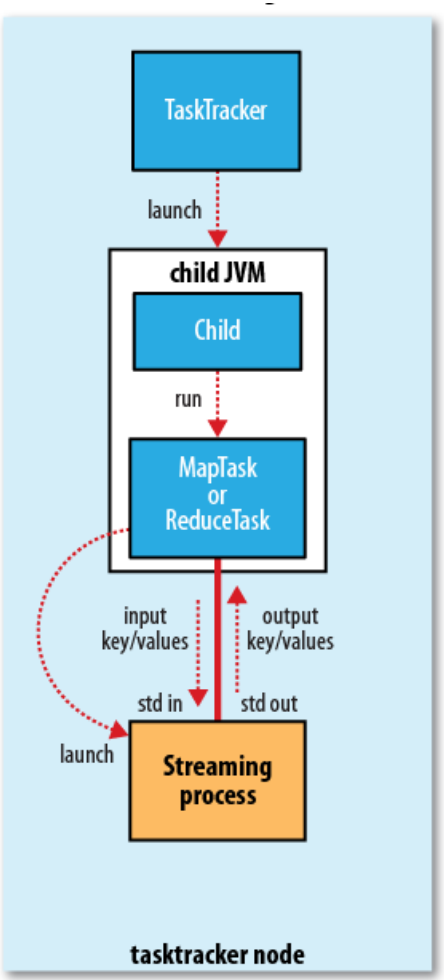




Tom White, Hadoop: The Definitive Guide, O'Reilly, 2013

Hadoop Streaming API

- Programming language agnostic way to express MapReduce application
- Use stdin/stdout of programming framework to communicate with MapReduce framework
- Python example for MapReduce Log File Analysis



Source: T. White, Hadoop: The Definitive Guide, 2012

Hadoop Streaming API


```
def emit(key, value):
    """
    Emits a key->value pair.  Key and value should be strings.
    """
    try:
        print "\t".join( (key, value) )
    except:
        pass

def run_map():
    """Calls map() for each input value."""
    for line in sys.stdin:
        line = line.rstrip()
        map(line)

def run_reduce():
    """Gathers reduce() data in memory, and calls reduce()."""
    prev_key = None
    values = []
    for line in sys.stdin:
        line = line.rstrip()
        key, value = re.split("\t", line, 1)
        if prev_key == key:
            values.append(value)
        else:
            if prev_key is not None:
                reduce(prev_key, values)
            prev_key = key
```

Output of map function is written to **`stdout`**.

`InputSplits` are passed as **`stdin`** into application. Utility function for reading lines and passing it to map function.

Hadoop Streaming API


```
def map(line):
    try:
        words = line.split()
        http_response_code = words[-2]
        emit(http_response_code, str(1))
    except:
        pass

def reduce(key, values):
    emit(key, str(sum(__builtin__.map(int,values))))
```

How to count the HTTP Response codes with Map Reduce?

Hadoop Streaming API

- **Test local**:

```
cat /data/NASA_access_log_Jul95|
        python map_reduce.py map |
        sort |
        python map_reduce.py reduce
```

- **Test cluster:**

```
hadoop fs -put map_reduce.py
hadoop jar hadoop-streaming.jar
        -files map_reduce.py
        -input input-nasa/
        -output output-nasa/
        -mapper 'map_reduce.py map'
        -reducer 'map_reduce.py reduce'
```

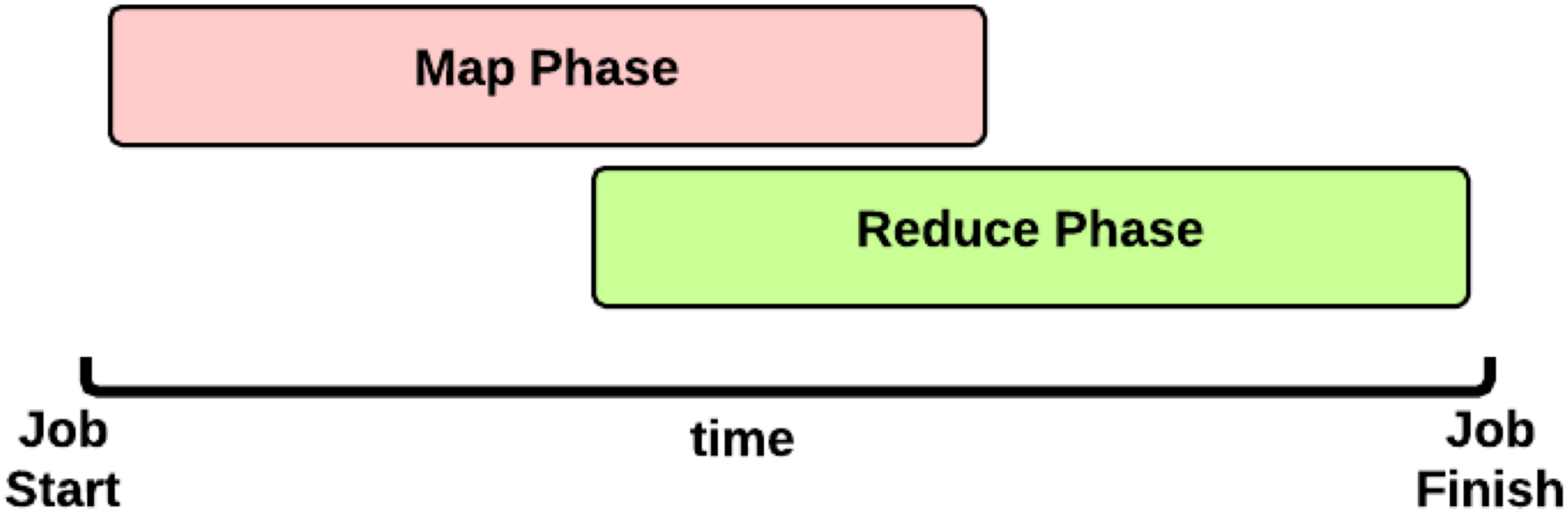
Hadoop MapReduce Components (1)

- **Record Reader:** *T*he record reader translates an input split generated by input format into records.
- **Map:** In the mapper, user-provided code is executed on each key/value pair from the record reader to produce zero or more new key/value pairs, called the intermediate pairs.
- **Combiner:** The combiner, an optional localized reducer, can group data in the map phase. It takes the intermediate keys from the mapper and applies a user-provided method to aggregate values in the small scope of that one mapper.
- **Partitioner:** The partitioner takes the intermediate key/value pairs from the mapper (or combiner if it is being used) and splits them up into shards, one shard per reducer.

Hadoop MapReduce Components (2)

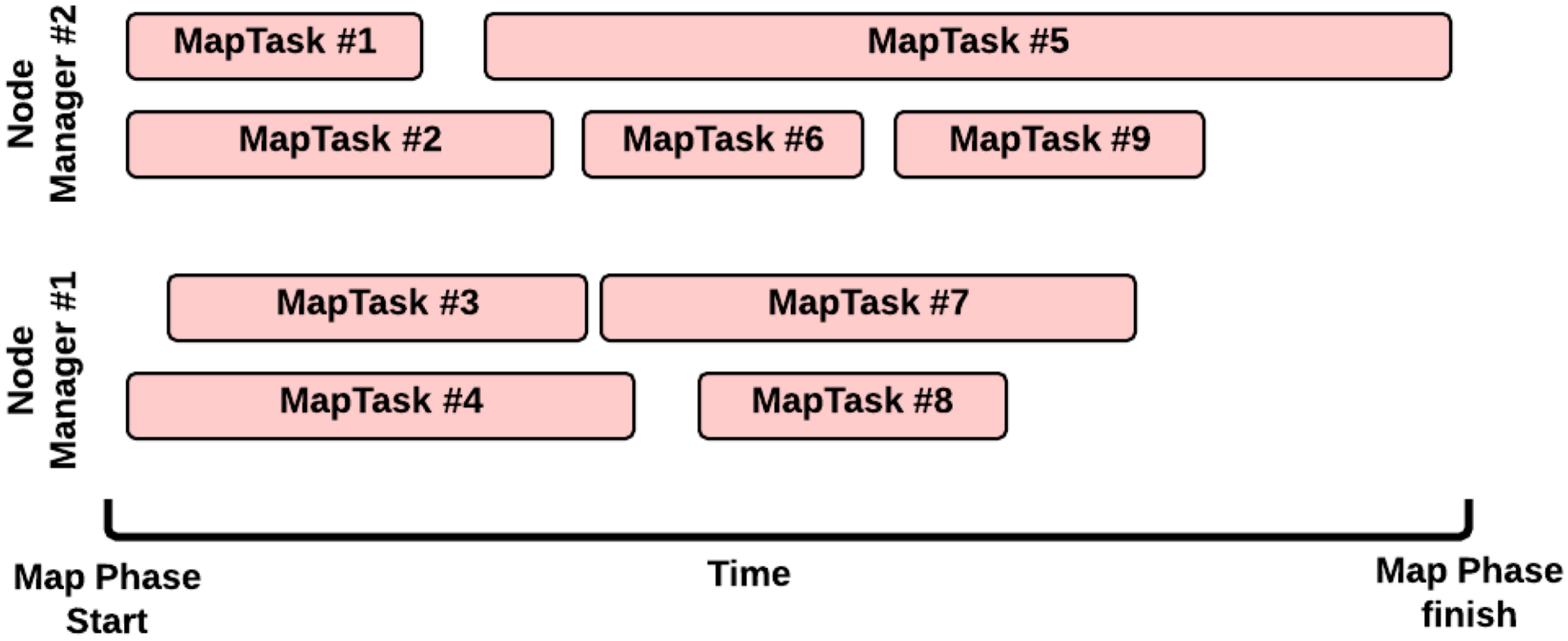
- **Shuffle and Sort:** The reduce task starts with the *shuffle and sort* step. This step takes the output files written by all of the partitioners and downloads them to the local machine in which the reducer is running.
- **Reduce:** The reducer takes the grouped data as input and runs a function once per key grouping. The function is passed the key and an iterator over all of the values associated with that key.
- **Output Format:** The output format translates the final key/value pair from the function and writes it out to a file by a record writer.

Anatomy of a MapReduce Job



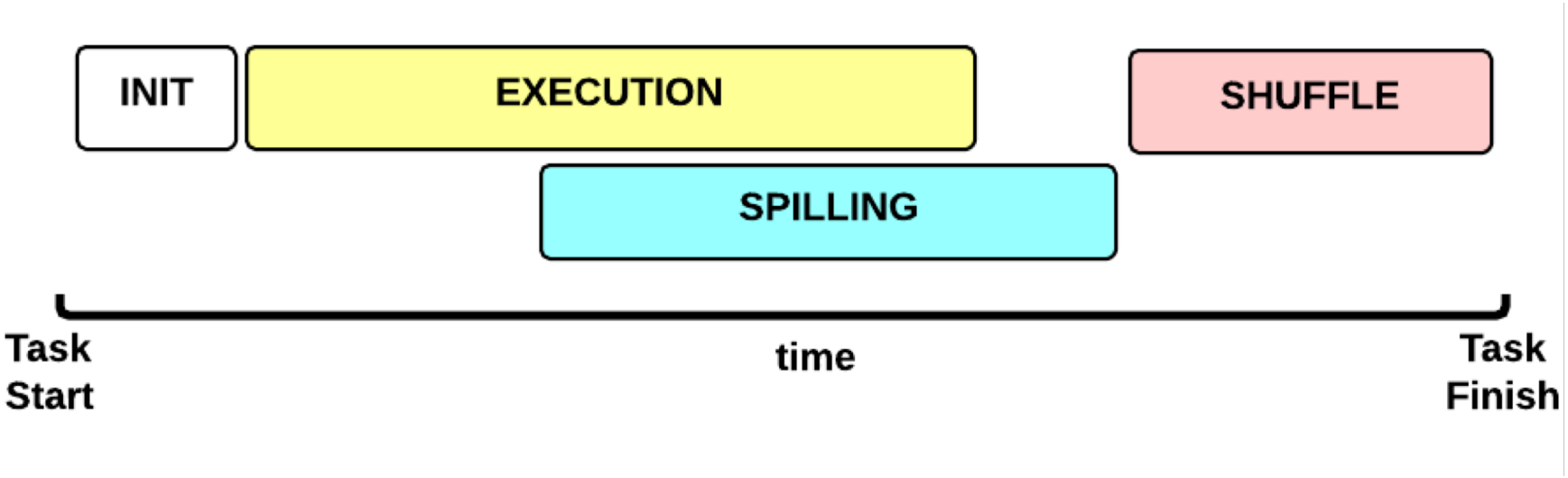
http://ercoppa.github.io/HadoopInternals/AnatomyMapReduceJob.html

Anatomy of a MapReduce Job

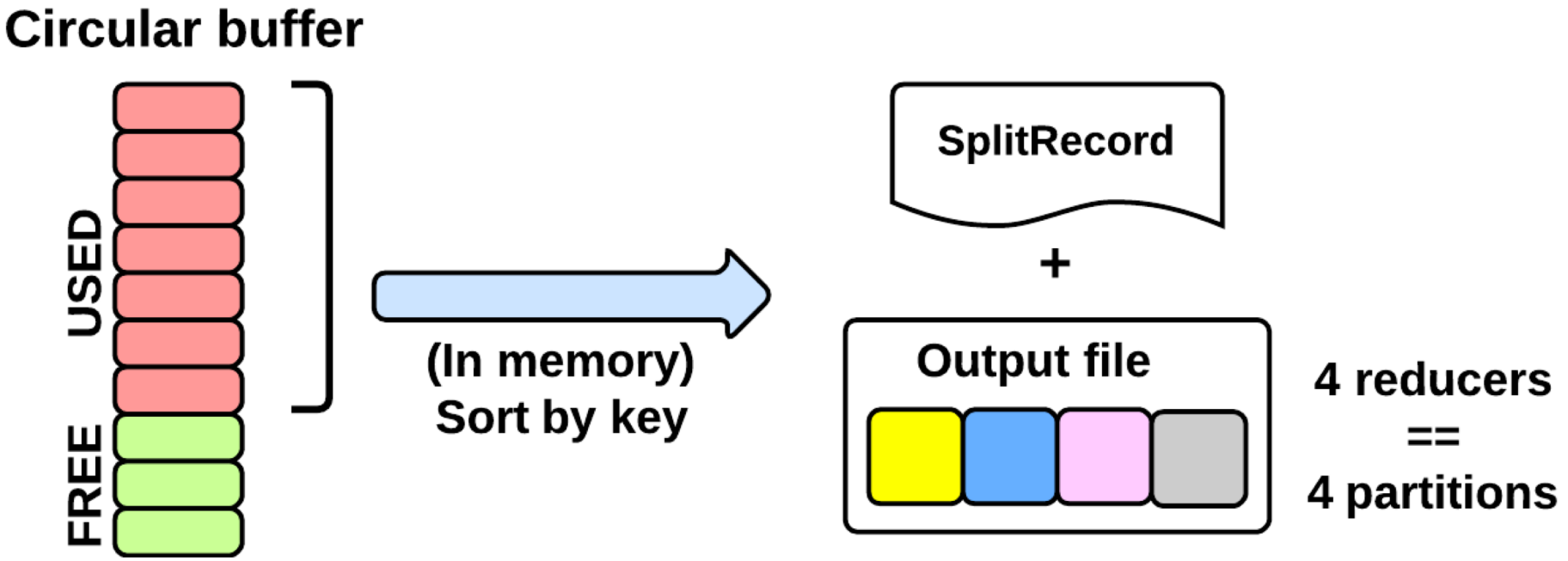


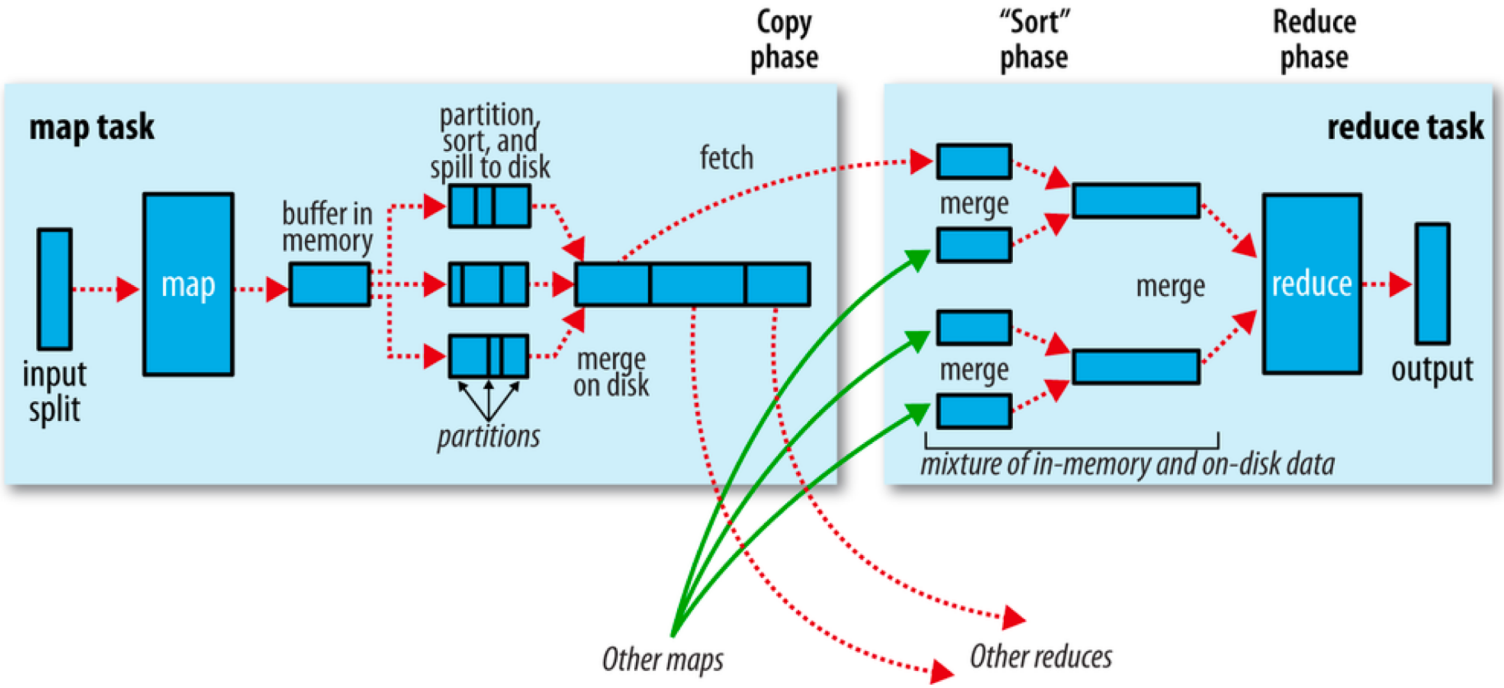
http://ercoppa.github.io/HadoopInternals/AnatomyMapReduceJob.html

Anatomy of a MapReduce Job

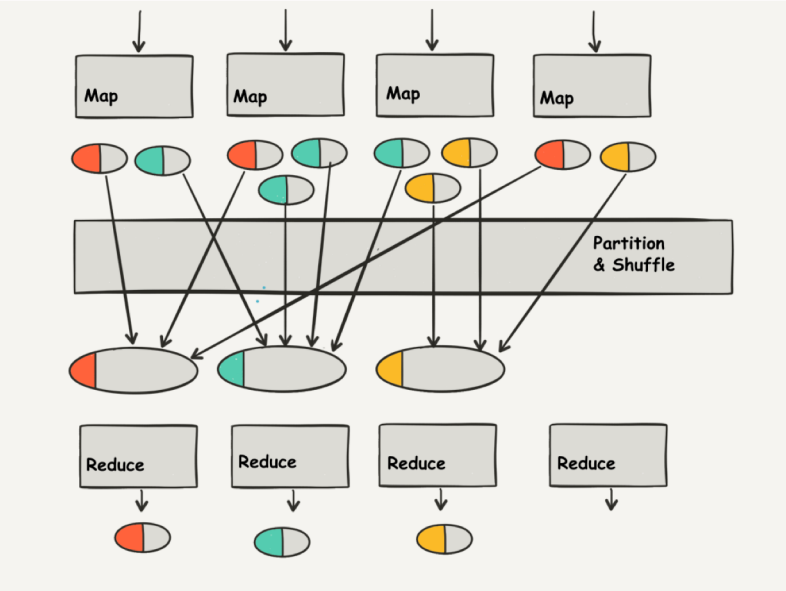


Anatomy of a MapReduce Job




Copy phase
"Sort" phase
Reduce phase
map task
reduce task
input split
map
buffer in memory
partition, sort, and spill to disk
partitions
merge on disk
fetch
merge
merge
merge
reduce
output
mixture of in-memory and on-disk data
Other maps
Other reduces

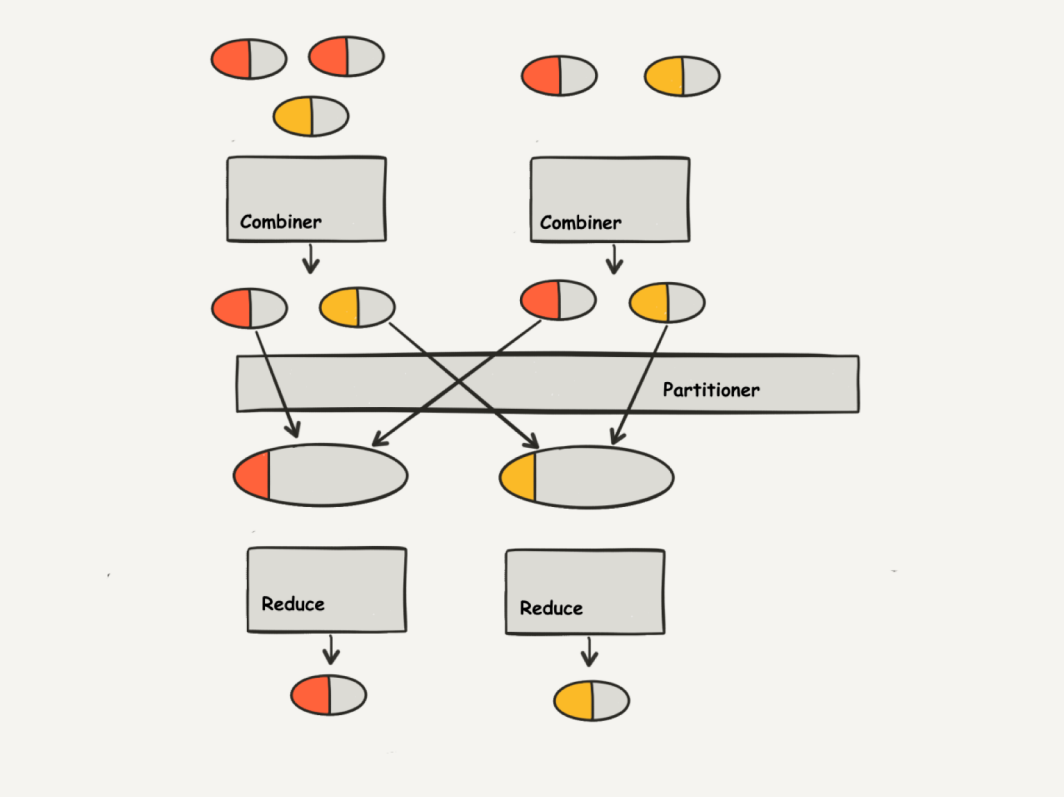
Shuffling



Maas, Garillot, Learning Spark Streaming, O'Reilly, 2017.

- Shuffling is the most "expensive" part of a MapReduce Job
- High demands on resources: CPU, RAM, disk and network IO
- Recommendations:
  - Avoid shuffle if possible
  - Minimize amount of data that gets shuffled
  - Use Compression
- A common benchmark for evaluating the shuffle performance of different frameworks is TeraSort.

Optimizing Shuffling: Combiners



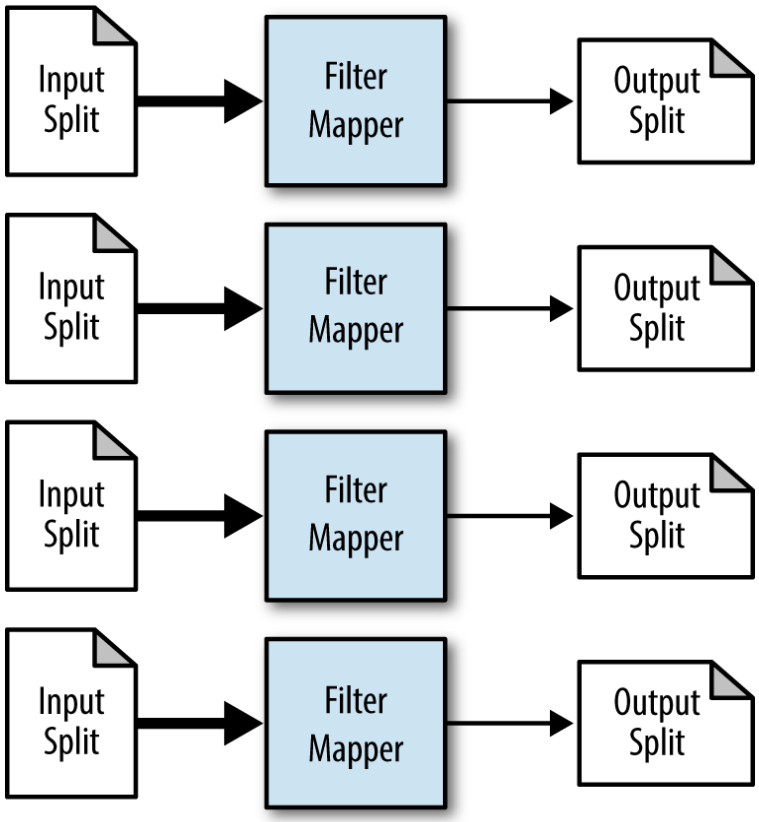
Maas, Garillot, Learning Spark Streaming, O'Reilly, 2017.

- **Combiner:** The combiner, an optional localized reducer, can group data in the map phase. It takes the intermediate keys from the mapper and applies a user-provided method to aggregate values in the small scope of that one mapper.
- Combiners allow the summarization of intermediate results as early as possible, before paying the price of sending data across the network.

Common MapReduce Patterns

- Framework for solving your data computation issues, without being specific to the problem domain.
- See Miner, Shook, MapReduce Design Patterns, O'Reilly, 2012.

MapReduce Patterns: Filtering

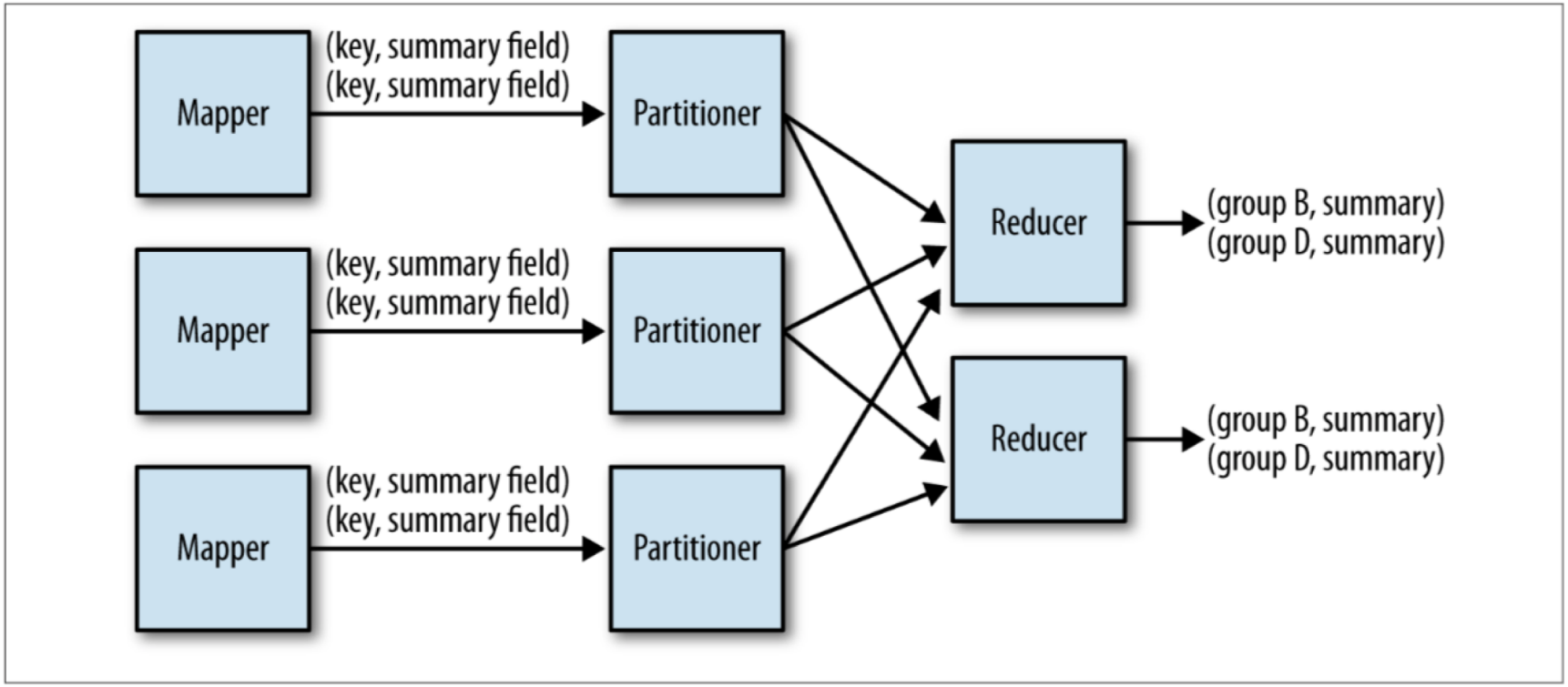



```
map(key, record):
   if we want to keep record then
      emit key,value
```

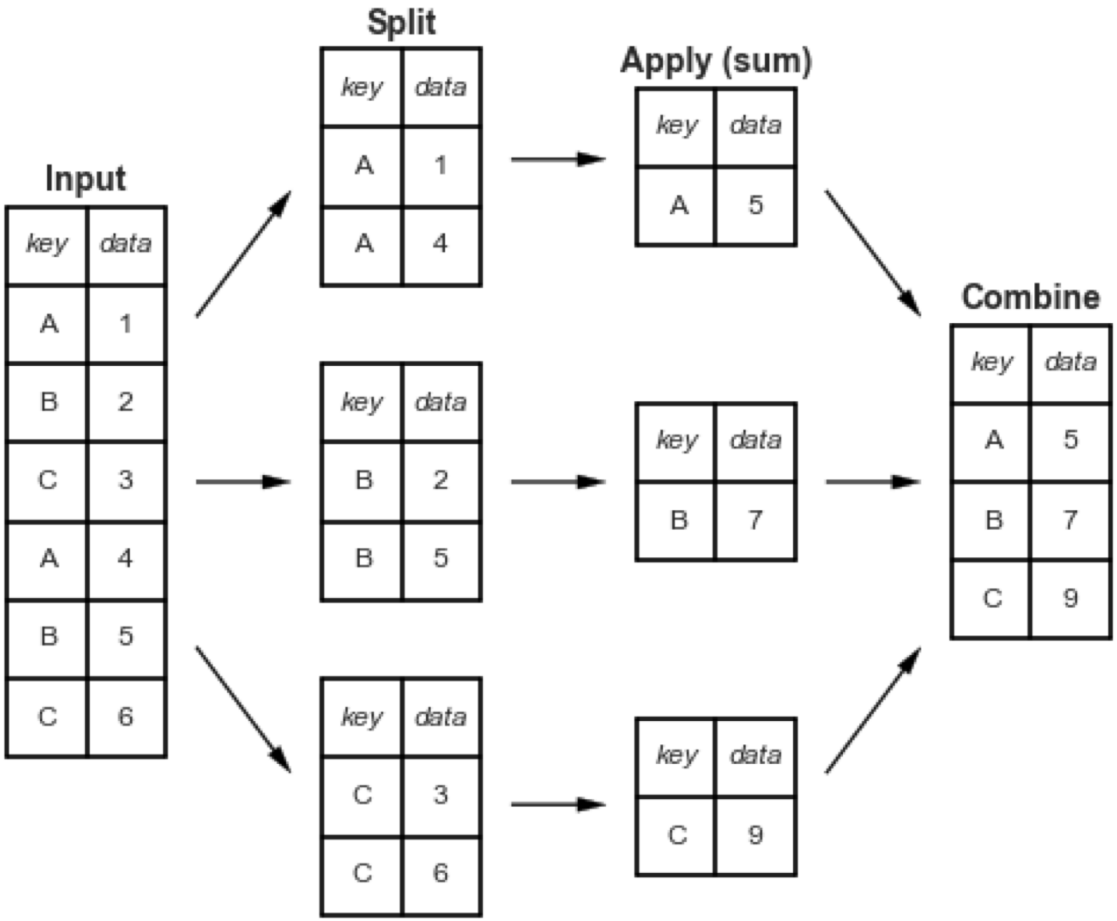
- Filter out records that are not of interest and keep ones that are.
- Examples:
  - Distributed Grep
  - Data cleaning/preparation
  - Sampling
  - Removing low-scoring data
- Embarrassingly parallel: no reducer and shuffle needed!

MapReduce Patterns: Summarization

Intent: Group records together by a key field and calculate a numerical aggregate per group to get a top-level view of the larger data set.
Example: Word Count, Average/Median/Standard deviation



MapReduce Pattern: Summarization



Split-Apply-Combine-Pattern

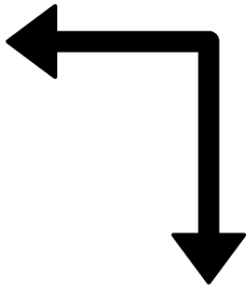
https://jakevdp.github.io/PythonDataScienceHandbook/03.08-aggregation-and-grouping.html

MapReduce Patterns: Summarization Min/Max/Count

Map Output / Combiner Input

	Input Key	Input Value		
	User	Minimum	Maximum	Count
Group 1	12345	10	10	1
	12345	8	8	1
	12345	21	21	1
Group 2	54321	1	1	1
	54321	47	47	1
	99999	7	7	1
	99999	12	12	1

Combiner executes over Group 1 and 2.
Does not execute over last two rows.



Combiner Output / Reducer Input

Output Key	Output Value		
	Minimum	Maximum	Count
12345	8	21	3
54321	1	47	2
99999	7	7	1
99999	12	12	1

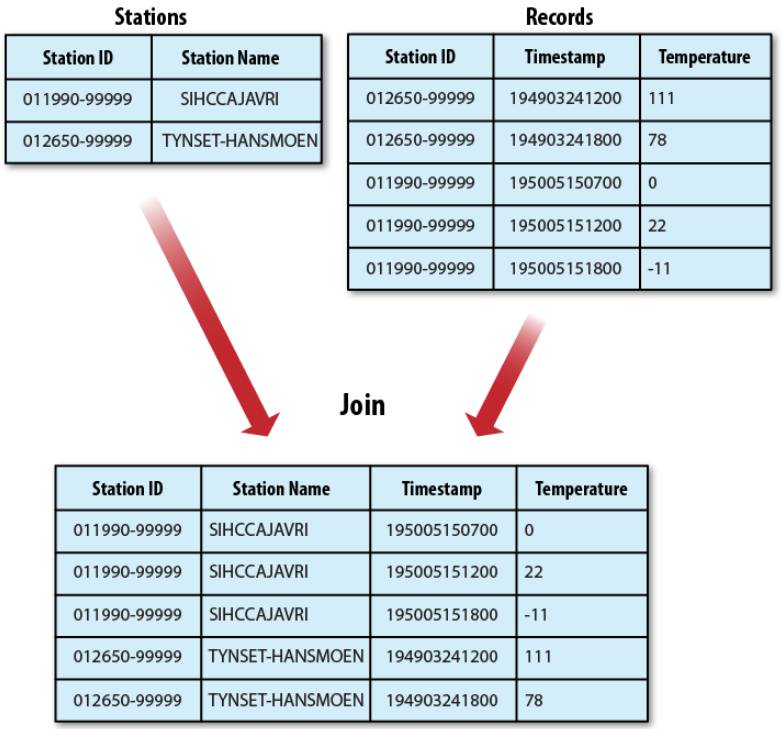
Donald Miner, Adam Shook, MapReduce Design Patterns, O'Reilly, 2012 (page 25).

MapReduce Patterns: Summarization Average

- An average is not an associative operation!
- The combiner cannot be used compute average directly
- Implement combiner by keeping track of the count and average (or sum).
- These two values can be multiplied to preserve the sum for the final reduce phase.

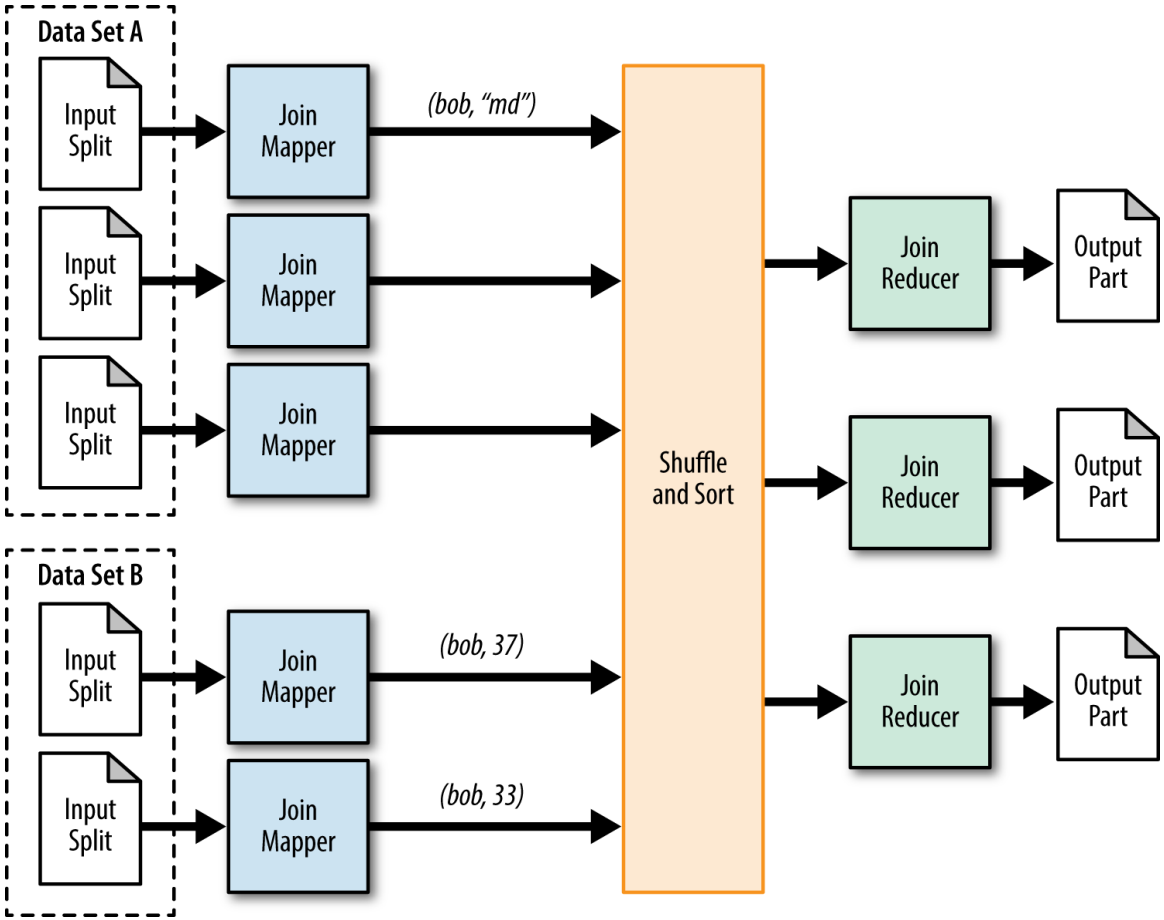
MapReduce Patterns: Joins

- **Reduce-Side Join:** more general, but both datasets have to go through shuffle phase (expensive!)
- **Map-Side Join:**
  - **Replicated:** one dataset is replicated to all Map tasks. Only feasible for small datasets.
  - **Composite:** Each input dataset must be divided into the same number of partitions and sort with the same key



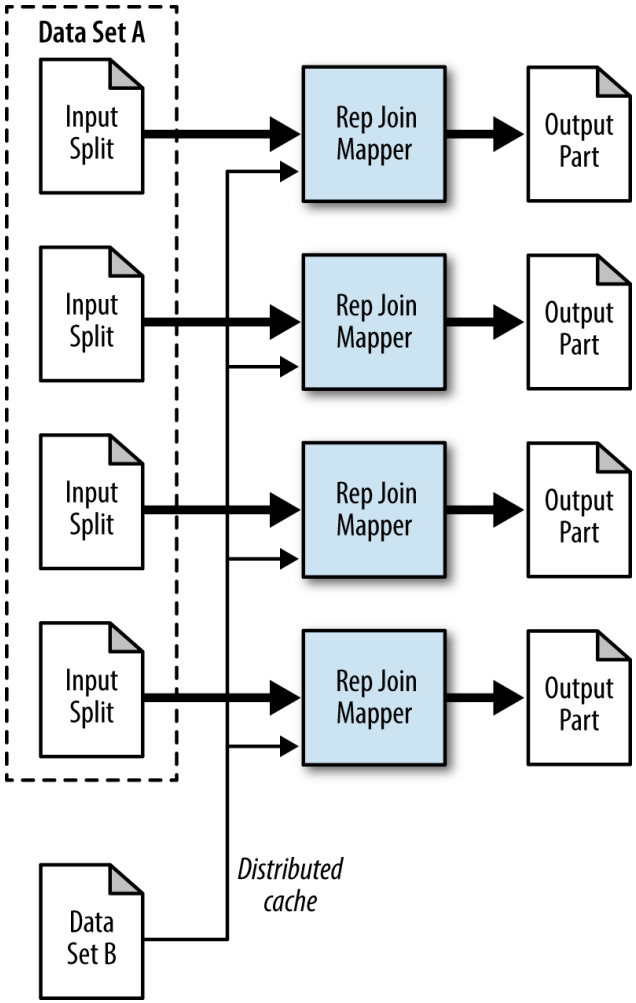
Source: T. White, Hadoop: The Definitive Guide, 2012

Reduce-Side Join

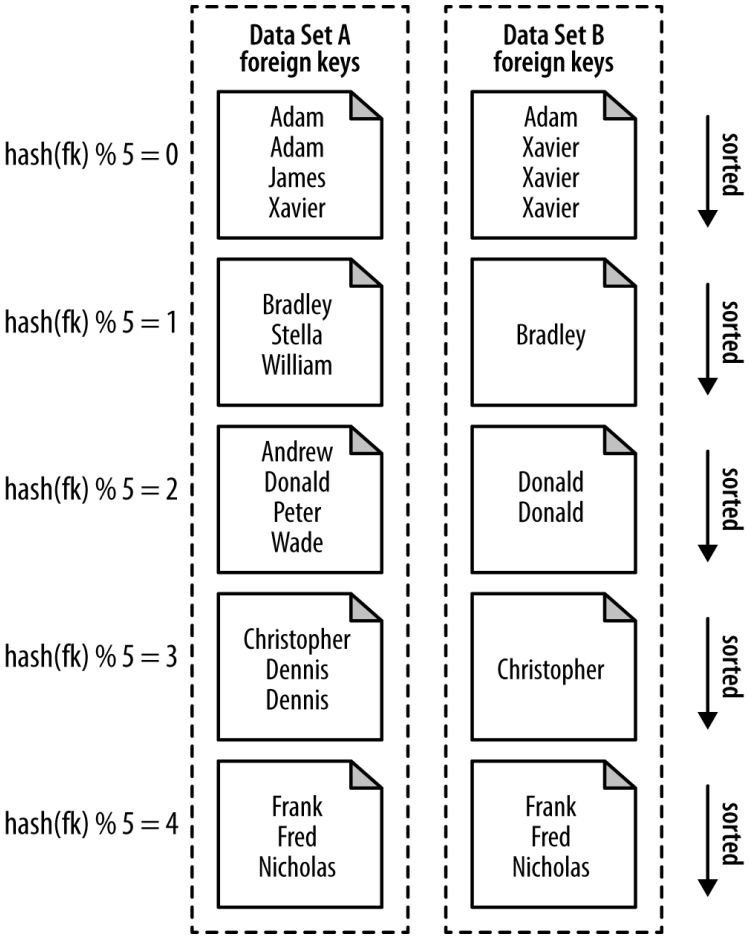


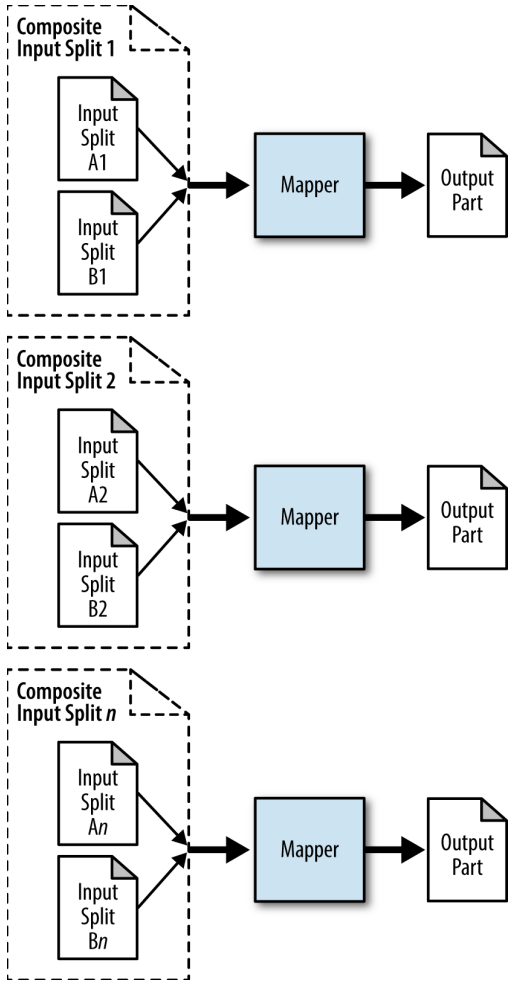
Donald Miner, Adam Shook, MapReduce Design Patterns, O'Reilly, 2012 (page 25).

Map-Side Join (Replicated)



Map-Side Join (Composite)





Donald Miner, Adam Shook, MapReduce Design Patterns, O'Reilly, 2012 (page 25).

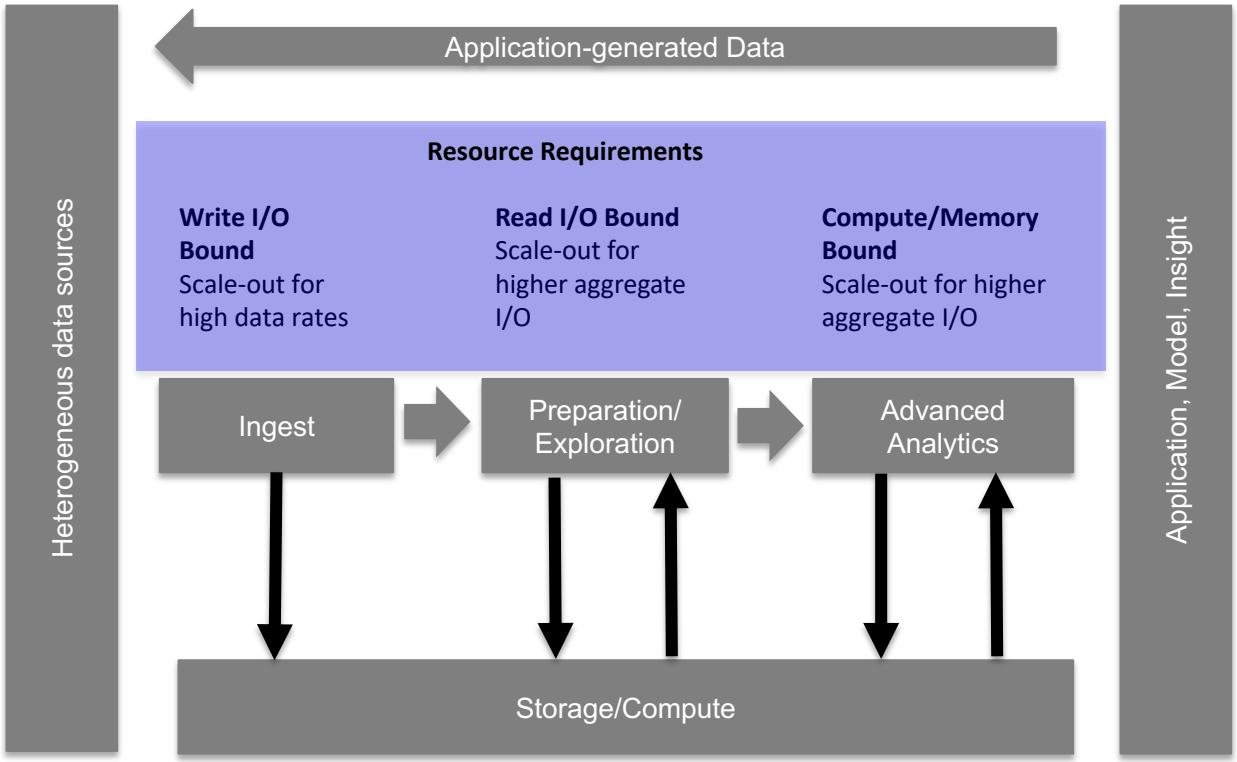
- **Machine Learning dataflow:** Iterative computations and keep track of state
  - There is both model and data, but only communicate the model
  - Collective communication operations, such as AllReduce AllGather, important
- We will revisit MapReduce patterns for Machine Learning in Scalable Machine Learning part of lecture.

Limitations of MapReduce

- **Iterative jobs:** Many common machine learning algorithms apply a function repeatedly to the same dataset to optimize a parameter (e.g., through gradient descent). While each iteration can be expressed as a MapReduce/Dryad job, each job must reload the data from disk, incurring a significant performance penalty.
- **Interactive analytics:** Hadoop is often used to run ad-hoc exploratory queries on large datasets, through SQL interfaces such as Pig and Hive. Ideally, a user would be able to load a dataset of interest into memory across a number of machines and query it repeatedly. However, with Hadoop, each query incurs significant latency (tens of seconds) because it runs as a separate MapReduce job and reads data from disk.
- **Caching layer needed for iterative jobs and interactive analytics**
- **Other criticism:**
  - **Rigid programming model**
  - **Not all algorithms suited for MapReduce**

http://static.usenix.org/legacy/events/hotcloud10/tech/full_papers/Zaharia.pdf

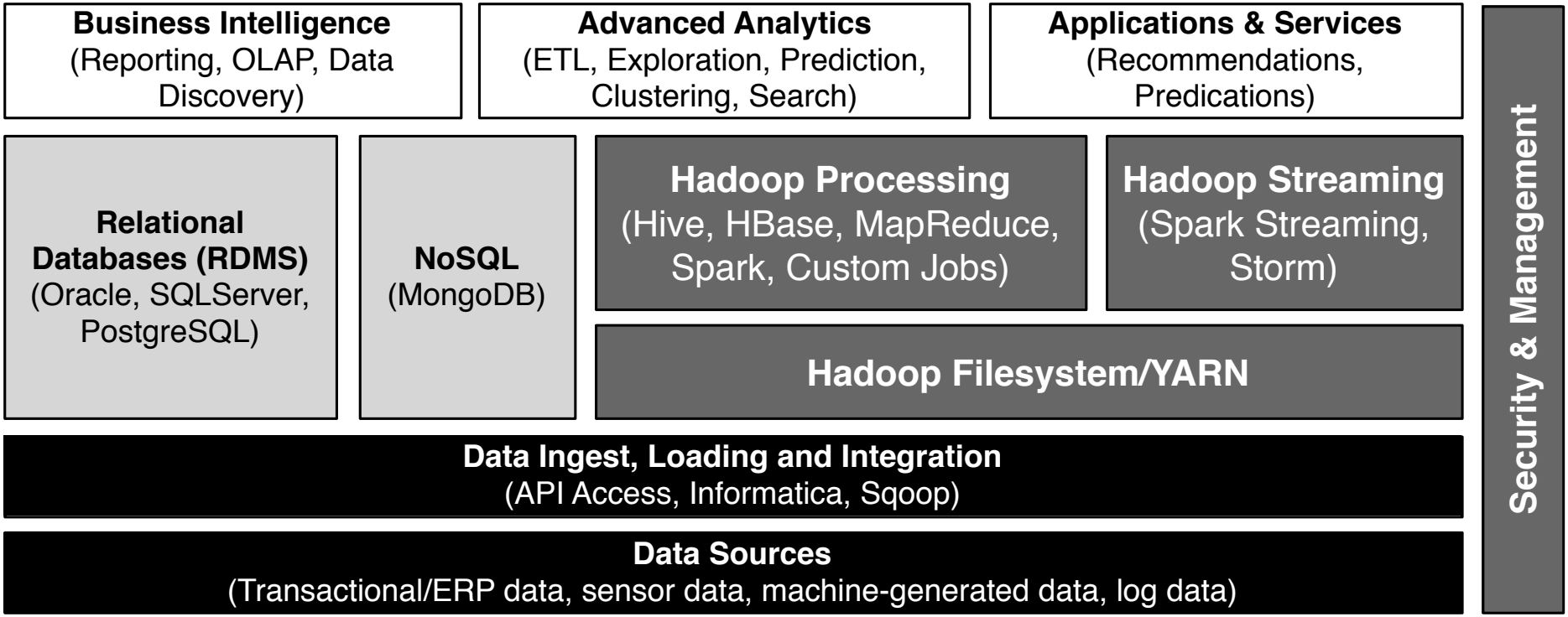
Data Lifecycle and Challenges



- **Data Lake** refers to the ability to retain large volumes of raw data in its original form in a Hadoop infrastructure and utilizing Hadoop-based tools to process, refine, combine and analyze the data.

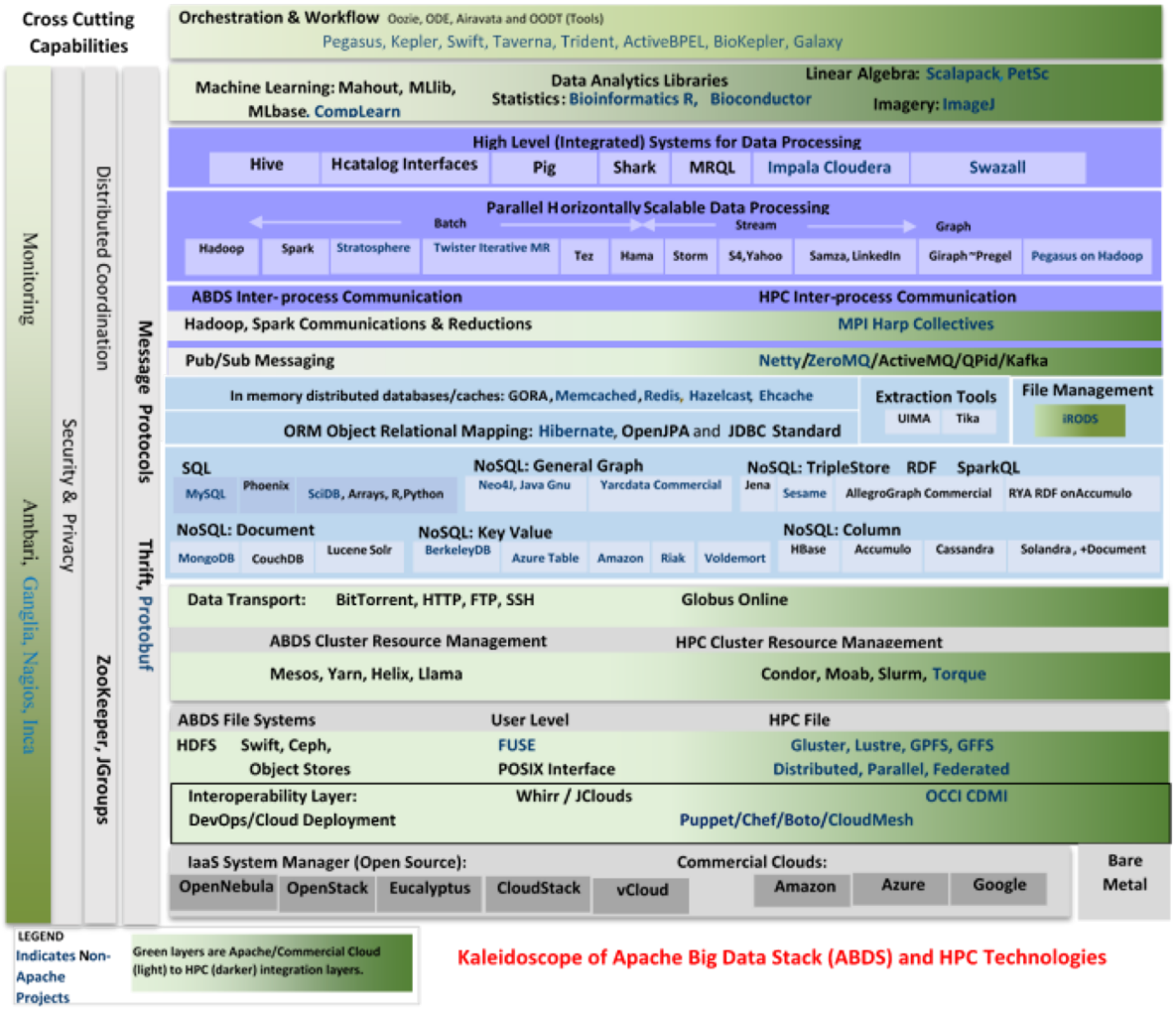
- **Properties:**
  - Economically storage and processing based on commodity hardware and open source
  - Flexible use: Different open data format (Text, ORC, Parquet) accessed using schema-on-read can be combined with different forms of processing (MapReduce, Spark/RDD, SQL, Machine Learning)

Apache Hadoop Stack



Apache Hadoop & HPC Integrated

On-going research
Indiana University,
Rutgers

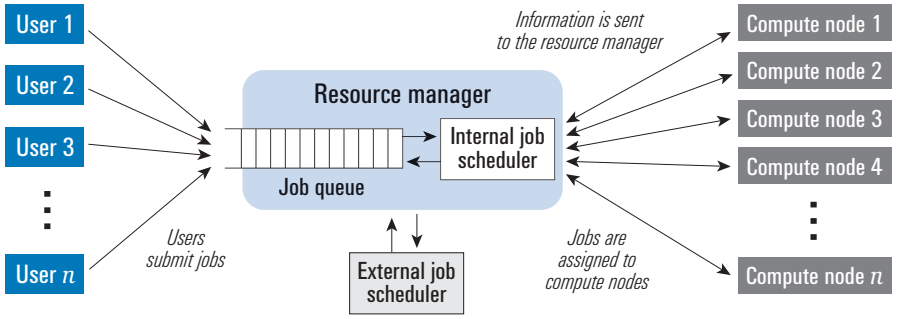


Kaleidoscope of Apache Big Data Stack (ABDS) and HPC Technologies

G. Fox, Big Data at the Intersection of Clouds and HPC, http://grids.ucs.indiana.edu/ptliupages/publications/

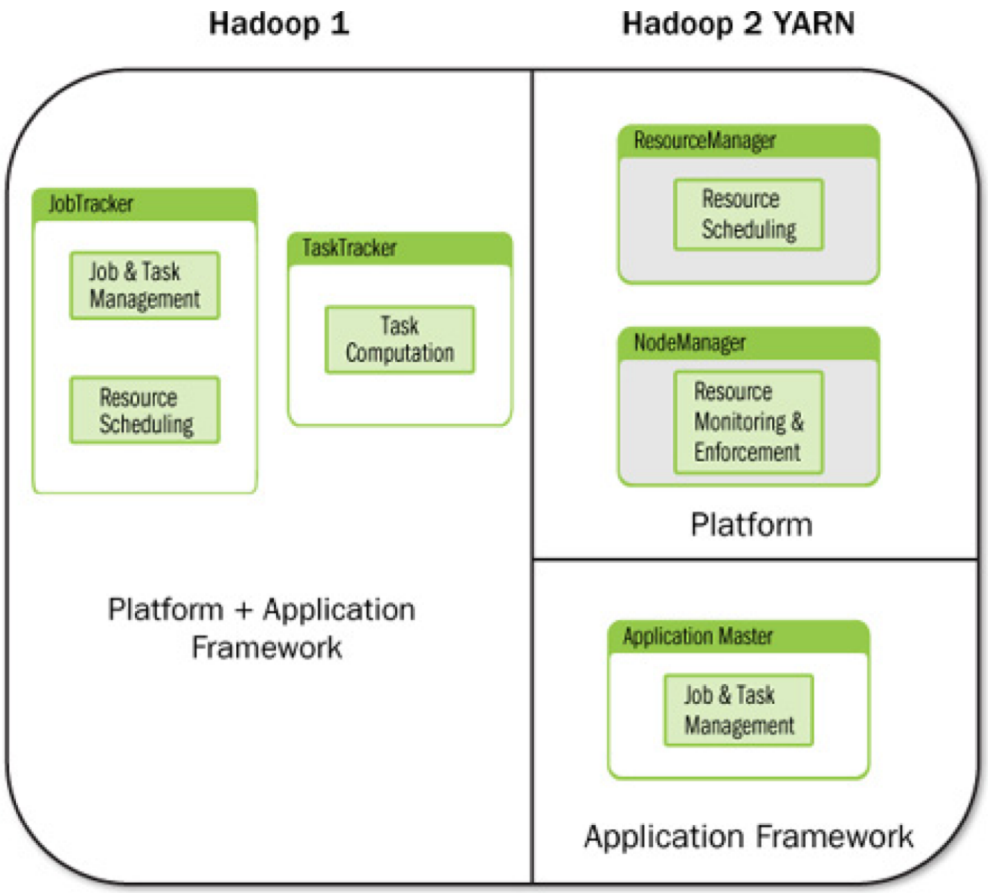
Cluster Scheduling

- The Resource Management System handles the computational resources of the cluster (CPU, Memory).
- Introduced in the context cluster computing:
  - Examples: SLURM, PBS, Torque, LSF. Palmetto uses PBS (http://citi.clemson.edu/palmetto/pages/userguide.html)



Iqbal et al., Job Scheduling in HPC Clusters, 2005

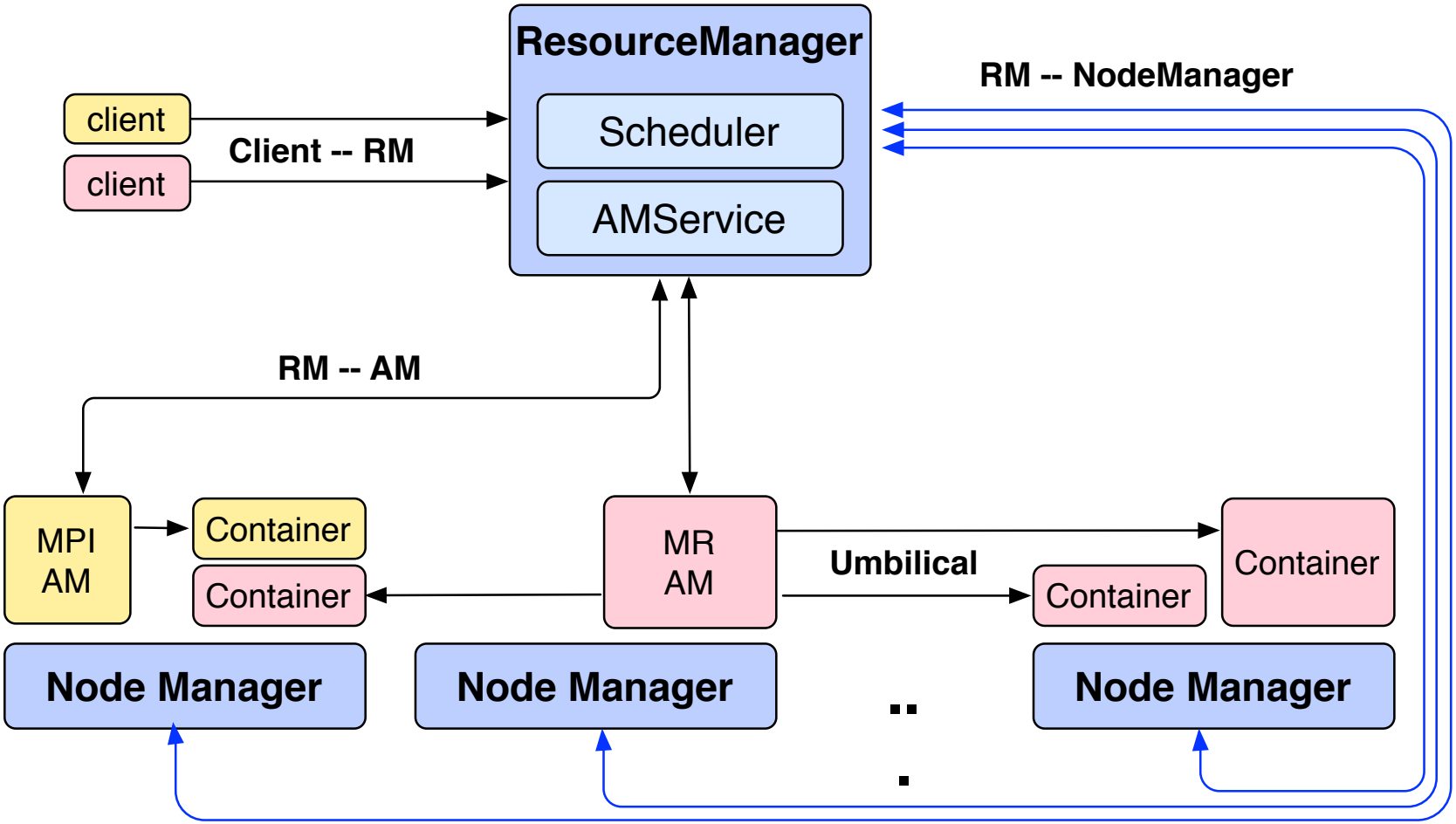
Hadoop Resource Management



- YARN decouples resource management and application framework
- YARN was specially designed for data-intensive applications
- Main differences to HPC:
  - Short-running, fine-grained data parallelism – **many small tasks in contrast to few large parallel jobs**
  - Negotiable resource requirements
  - Dynamic resource requirements that change during application lifecycle

Murphy et al., Apache Hadoop YARN: Moving beyond MapReduce and Batch Processing with Apache Hadoop 2, Addison-Wesley, 2014

Hadoop Resource Management: YARN



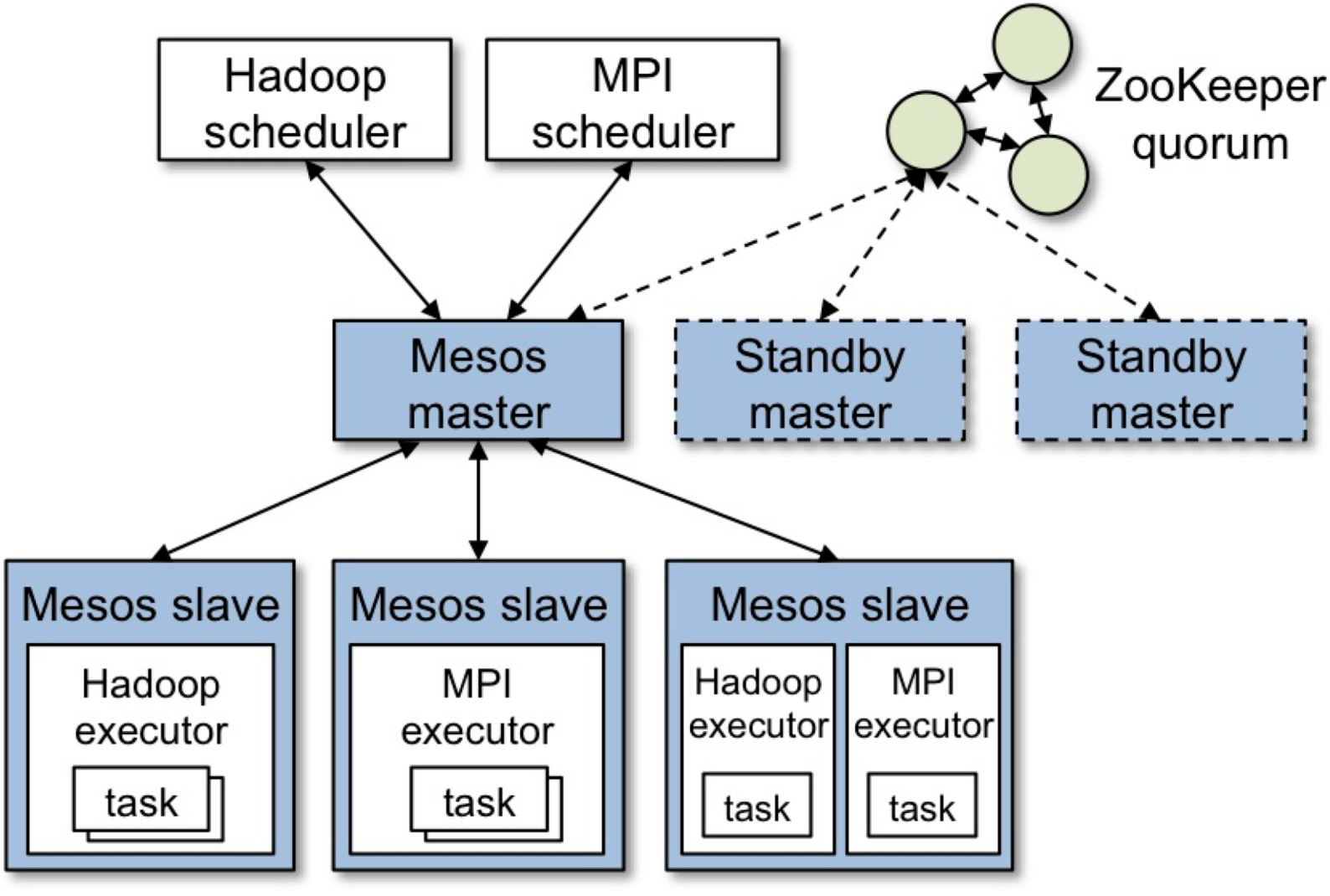
YARN Properties

- Enable fine-grained resource sharing - assigning complete nodes hurts utilization and should be avoided.
- More dynamic than traditional HPC model
- When the container starts, all data files, executables, and necessary dependencies are copied to local storage on the node
- Unlike some resource schedulers in which clients request hard limits, YARN allows applications to adapt (if possible) to the current cluster environment

YARN Properties

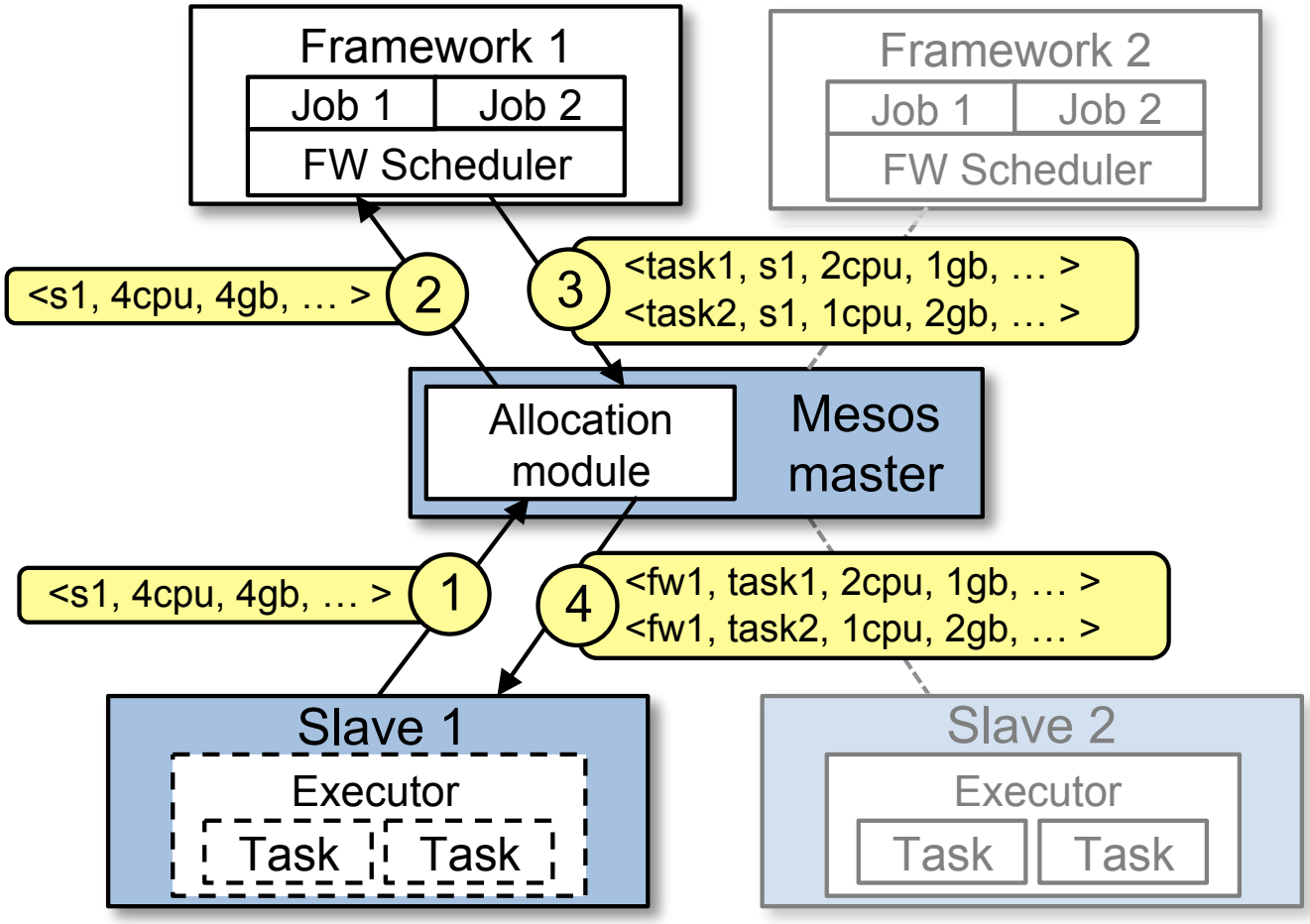
- Location-Awareness: Furthermore, the Capacity scheduler itself is locality aware, and is very good at automatically allocating resources on not only preferred nodes/racks, but also nodes/racks that are close to the preferred ones.
- Pluggable Scheduler:
  - Capacity Scheduler
  - Fair Scheduler

Mesos Architecture



Hindmann, Mesos: A Platform For Fine-Grained Resource Sharing in the Data Center, https://people.eecs.berkeley.edu/~alig/papers/mesos.pdf

Mesos Architecture



Hindmann, Mesos: A Platform For Fine-Grained Resource Sharing in the Data Center, https://people.eecs.berkeley.edu/~alig/papers/mesos.pdf

High Performance Computing vs. Big Data Computing

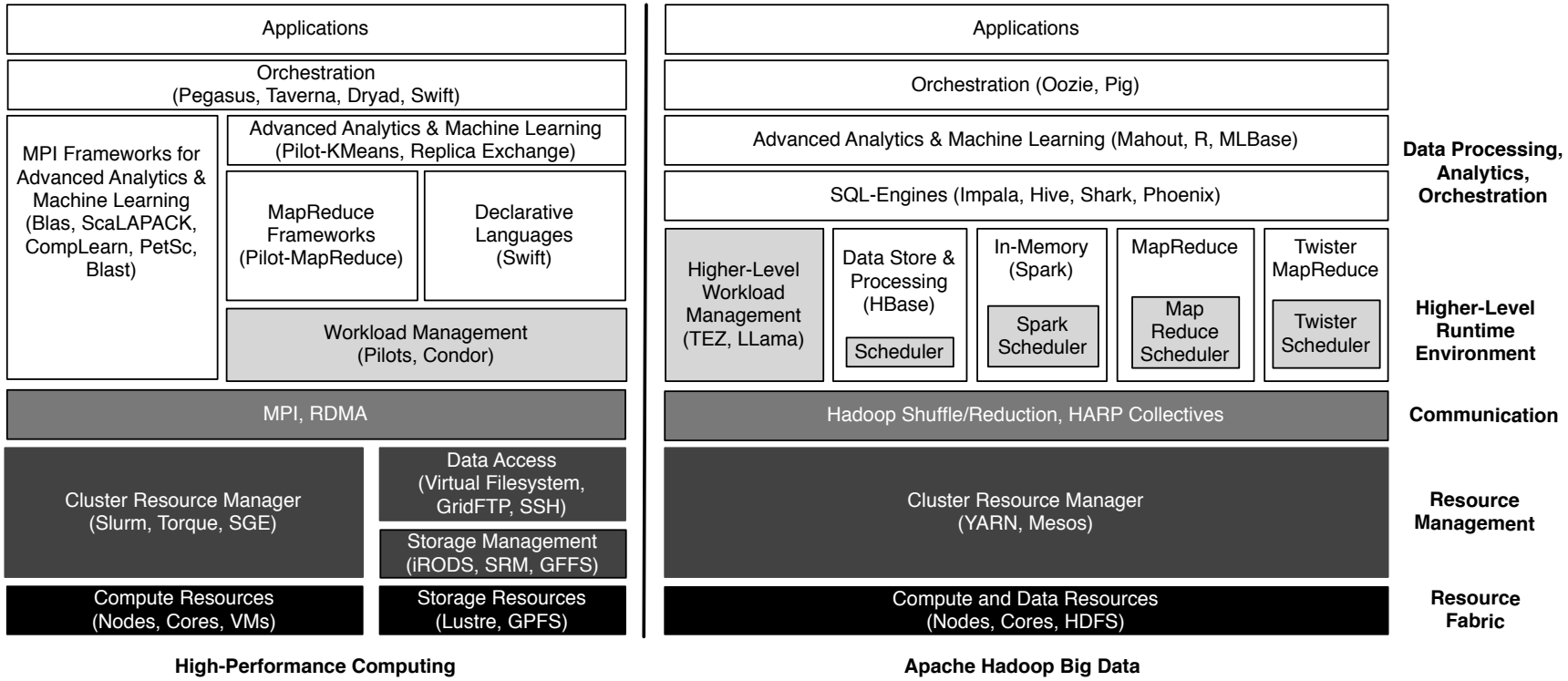


Fig. 1. **HPC and ABDS architecture and abstractions:** The HPC approach historically separated data and compute; ABDS co-locates compute and data. The YARN resource manager heavily utilizes multi-level, data-aware scheduling and supports a vibrant Hadoop-based ecosystem of data processing, analytics and machine learning frameworks. Each approach has a rich, but hitherto distinct resource management and communication capabilities.

Quelle: Shantenu Jha, Judy Qiu, Andre Luckow, Pradeep Mantha, Geoffrey C.Fox: *A Tale of Two Data-Intensive Paradigms: Applications, Abstractions, and Architectures*, https://arxiv.org/pdf/1403.1528.pdf

Cluster Scheduling

- Eric Baldeschwieler et al., Apache Hadoop YARN: Yet Another Resource Negotiator, https://www.cs.cmu.edu/~garth/15719/papers/yarn.pdf
- Hindemann, Meso: A Platform for Fine-Grained Resource Sharing in the Data Center, Technical Report, University of California, Berkley, 2010
- Verma et al., Large-scale cluster management at Google with Borg, Proceedings of the European Conference on Computer Systems (EuroSys), ACM, 2015
- Kubernetes, https://kubernetes.io/>
- Burns et al., Borg, Omega, and Kubernetes, ACM Queue, 2016