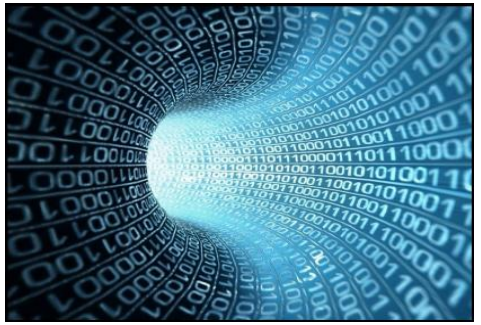
Challenges for Large-scale Data Processing

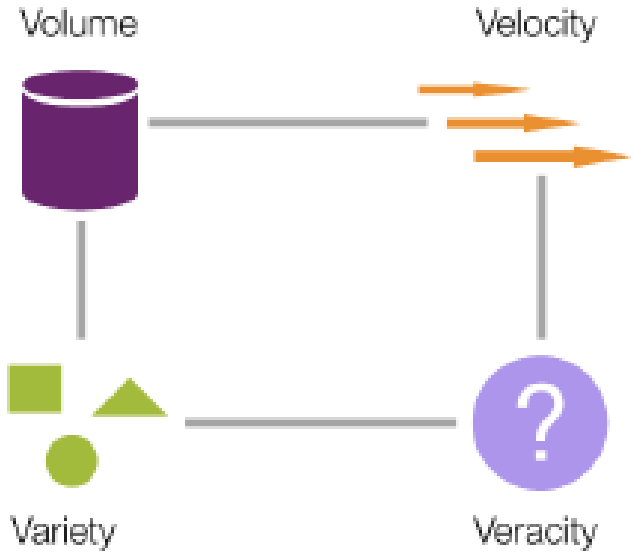
Eiko Yoneki

University of Cambridge Computer Laboratory

2010s: Big Data



- Why Big Data now?
  - Increase of Storage Capacity
  - Increase of Processing Capacity
  - Availability of Data
  - Hardware and software technologies can manage ocean of data



up to 2003 5 exabytes
→ 2012 2.7 zettabytes (500 x more)
→ 2015 ~8 zettabytes (3 x more than 2012)

Massive Data: Scale-Up vs Scale-Out

- Popular solution for massive data processing
  → scale and build distribution, combine theoretically unlimited number of machines in single distributed storage
  → Parallelisable data distribution and processing is key

- Scale-up: add resources to single node (many cores) in system (e.g. HPC)

- Scale-out: add more nodes to system (e.g. Amazon EC2)

Typical Operation with Big Data

- Find similar items → efficient multidimensional indexing
- Incremental updating of models → support streaming
- Distributed linear algebra → dealing with large sparse matrices
- Plus usual data mining, machine learning and statistics
  - Supervised (e.g. classification, regression)
  - Non-supervised (e.g. clustering..)

Technologies

- Distributed infrastructure
  - Cloud (e.g. Infrastructure as a service, Amazon EC2, Google App Engine, Elastic, Azure)

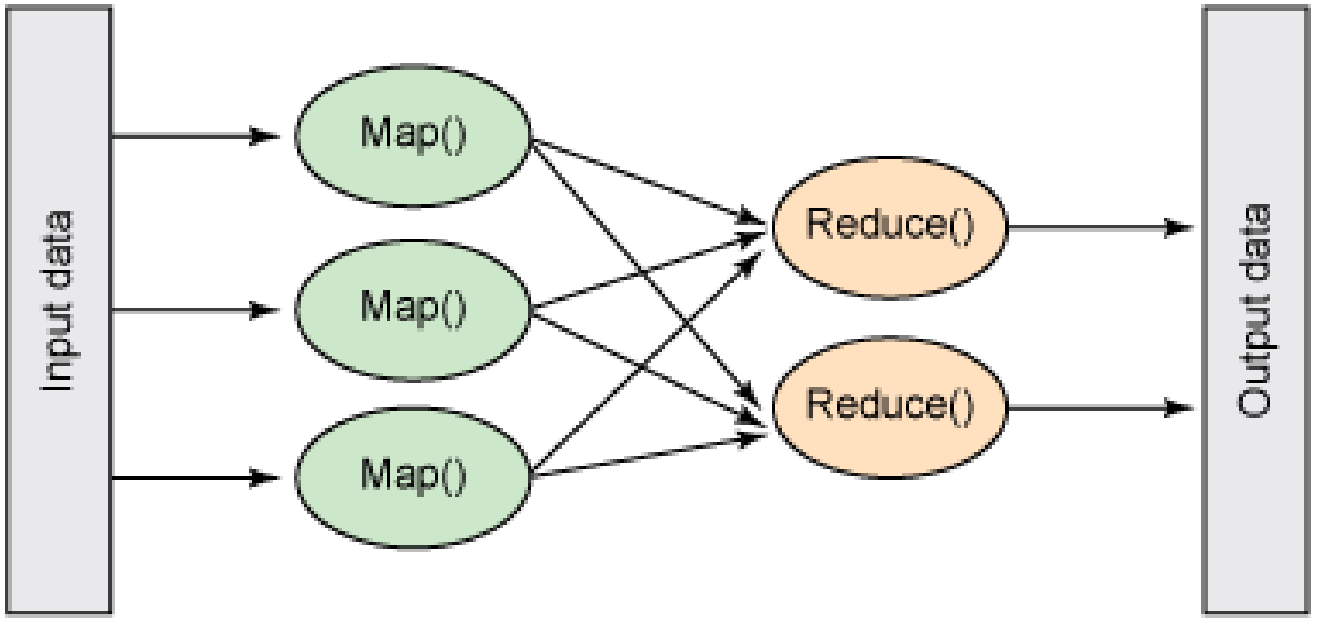
    cf. Many core (parallel computing)
- Storage
  - Distributed storage (e.g. Amazon S3, Hadoop Distributed File System (HDFS), Google File System (GFS))
- Data model/indexing
  - High-performance schema-free database (e.g. NoSQL DB - Redis, BigTable, Hbase, Neo4J)
- Programming model
  - Distributed processing (e.g. MapReduce)

NoSQL (Schema Free) Database

- NoSQL database
  - Operate on distributed infrastructure
  - Based on key-value pairs (no predefined schema)
  - Fast and flexible
- Pros: Scalable and fast
- Cons: Fewer consistency/concurrency guarantees and weaker queries support
- Implementations
  - MongoDB, CouchDB, Cassandra, Redis, BigTable, Hibase …

MapReduce Programming

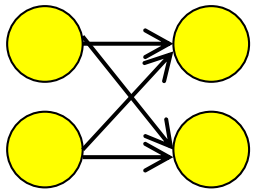
- Target problem needs to be parallelisable
- Split into a set of smaller code (map)
- Next small piece of code executed in parallel
- Results from map operation get synthesised into a result of original problem (reduce)



Data Flow Programming

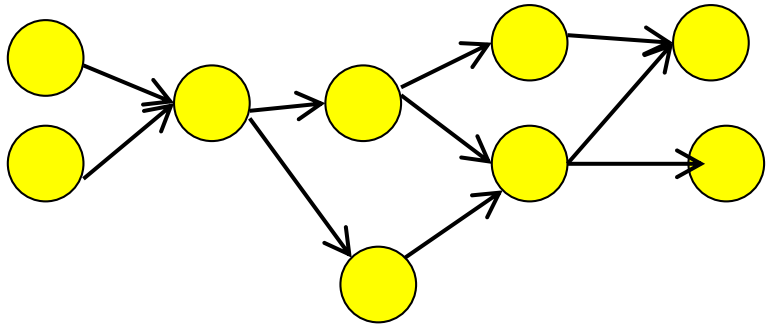
- Non standard programming models
- Data (flow) parallel programming
  - e.g. MapReduce, Dryad/LINQ, NAIAD, Spark, Tensorflow…

MapReduce:
Hadoop

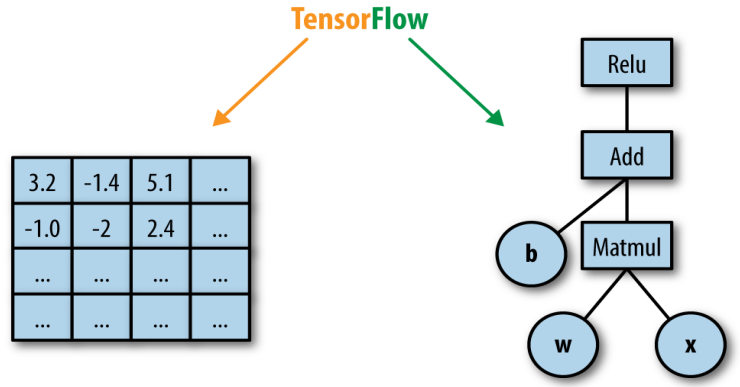


Two-Stage fixed dataflow

DAG (Directed Acyclic Graph)
based: Dryad/Spark…



More flexible dataflow model



Data Processing Stack

Programming

Data Processing Layer

Streaming Processing
Storm, SEEP, Naiad, Spark Streaming, Flink, Milwheel, Google Dataflow...

Query Language
Pig, Hive, SparkSQL, DryadLINQ…

Machine Learning
Tensorflow, Caffe, torch, MLlib…

Execution Engine
MapReduce, Spark, Dryad, Flumejava…

Graph Processing
Pregel, Giraph, GraphLab, PowerGraph, (Dato), GraphX, X-Stream...

Storage Layer

Distributed File Systems
GFS, HDFS, Amazon S3, Flat FS..

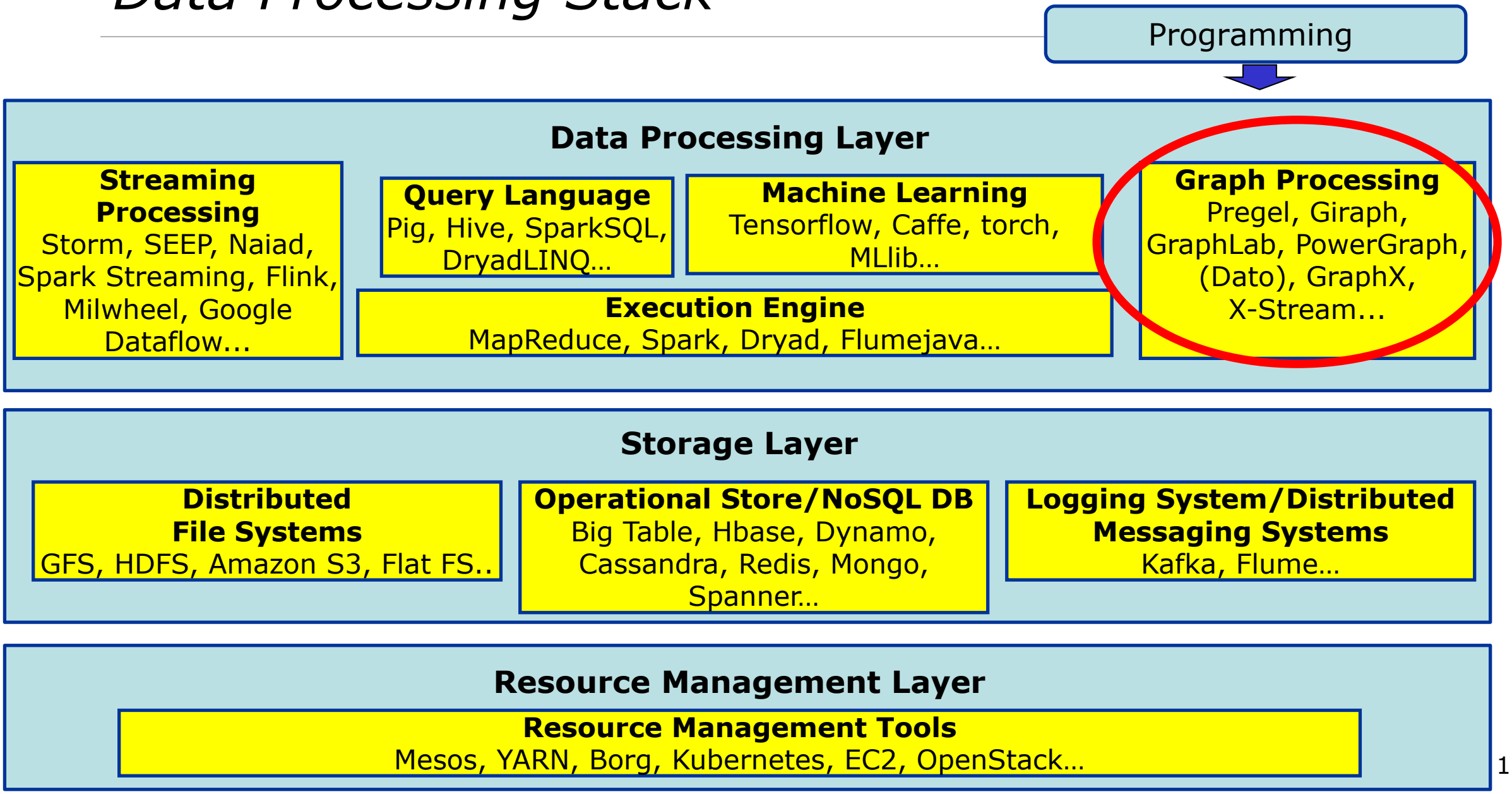
Operational Store/NoSQL DB
Big Table, Hbase, Dynamo, Cassandra, Redis, Mongo, Spanner…

Logging System/Distributed Messaging Systems
Kafka, Flume…

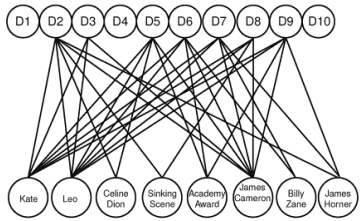
Resource Management Layer

Resource Management Tools
Mesos, YARN, Borg, Kubernetes, EC2, OpenStack…

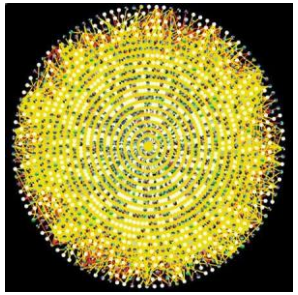
Data Processing Stack



Emerging Massive-Scale Graph Data



Bipartite graph of phrases in documents



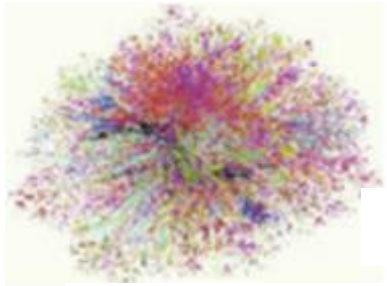
Protein Interactions [genomebiology.com]

Gene expression data

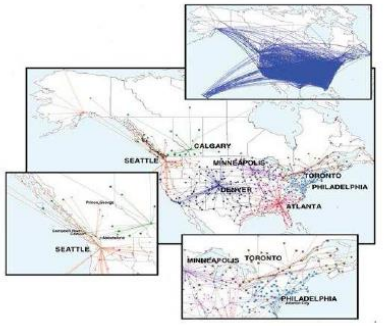
BFS
DFS
CC
SCC
SSSP
ASP
A*
Community
Centrality
Diameter
Page Rank
MIS
SALSA...

Social media data

Brain Networks: 100B neurons(700T links) requires 100s GB memory



Web 1.4B pages(6.6B links)



Airline Graphs

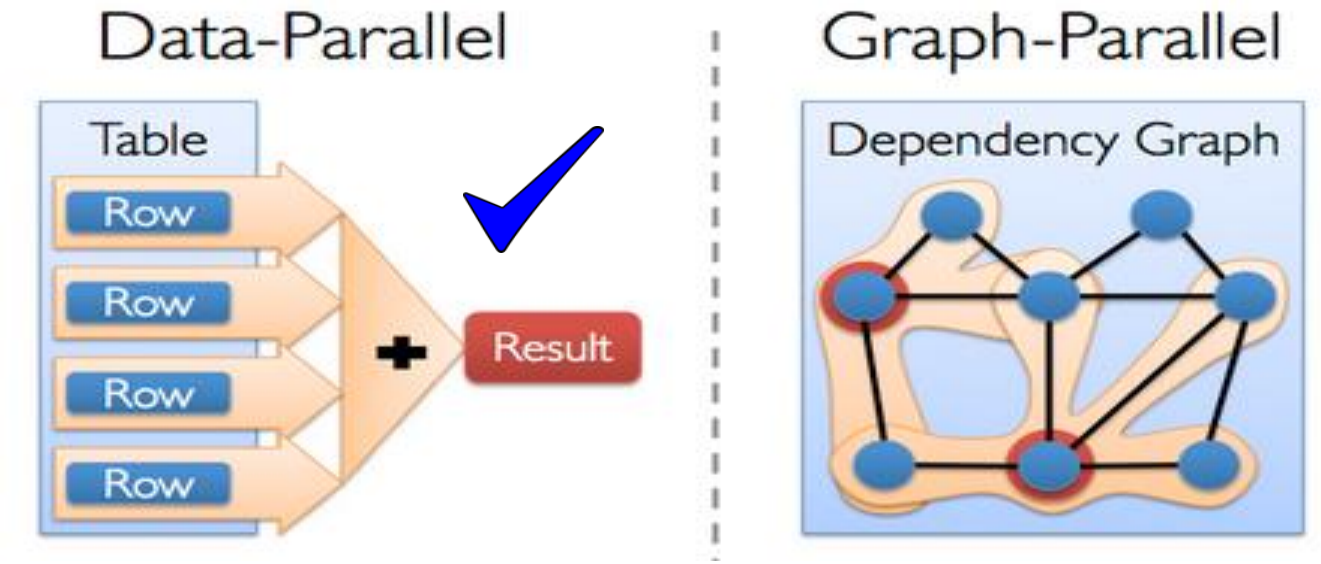
Graph Computation Challenges

1. Graph algorithms (BFS, Shortest path)
2. Query on connectivity (Triangle, Pattern)
3. Structure (Community, Centrality)
4. ML & Optimisation (Regression, SGD)

- Data driven computation: dictated by graph's structure and parallelism based on partitioning is difficult
- Poor locality: graph can represent relationships between irregular entries and access patterns tend to have little locality
- High data access to computation ratio: graph algorithms are often based on exploring graph structure leading to a large access rate to computation ratio

Data-Parallel vs. Graph-Parallel

- *Data-Parallel* for all? *Graph-Parallel* is hard!
  - Data-Parallel (sort/search - randomly split data to feed MapReduce)
  - Not every graph algorithm is parallelisable (interdependent computation)
  - Not much data access locality
  - High data access to computation ratio



Graph-Parallel

- Graph-Parallel (Graph Specific Data Parallel)
  - Vertex-based iterative computation model
  - Use of iterative Bulk Synchronous Parallel Model

    ➔ Pregel (Google), Giraph (Apache), Graphlab, GraphChi (CMU - Dato)

  - Optimisation over data parallel

    ➔ GraphX/Spark (U.C. Berkeley)
  - Data-flow programming – more general framework

    ➔ NAIAD (MSR), TensorFlow..

Bulk synchronous parallel: Example

- Finding the largest value in a connected graph

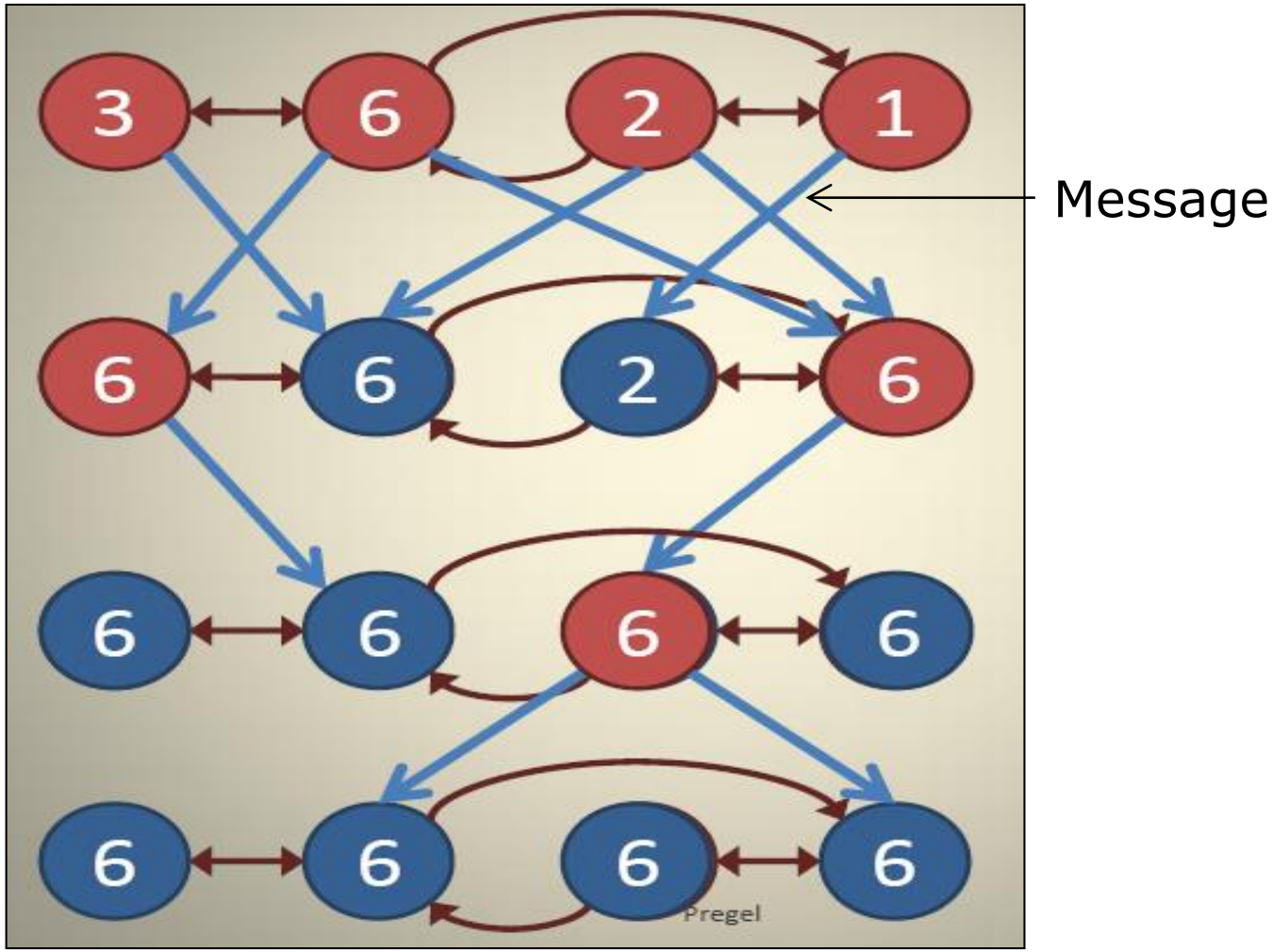
Local Computation

Communication

Local Computation

Communication

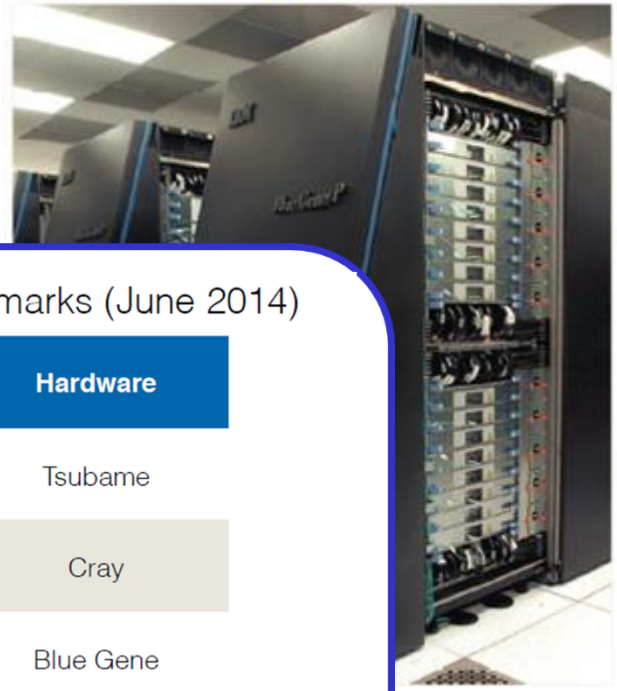
...



Are Large Clusters and Many cores Efficient?

- Brute force approach really efficiently works?
  - Increase of number of cores (including use of GPU)
  - Increase of nodes in clusters

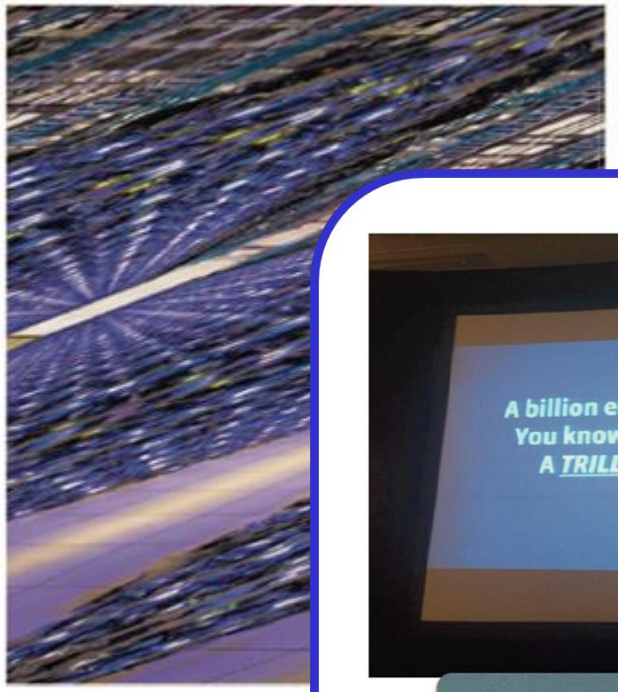
Big Iron

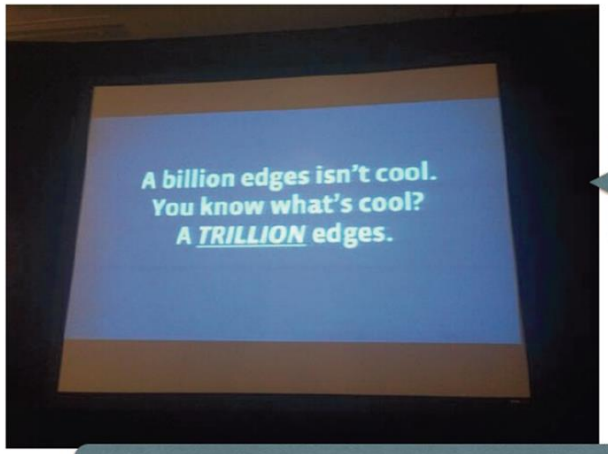


HPC/Graph500 benchmarks (June 2014)

Graph Edges	Hardware
1 trillion	Tsubame
1 trillion	Cray
1 trillion	Blue Gene
1 trillion	NEC

Large Cluster

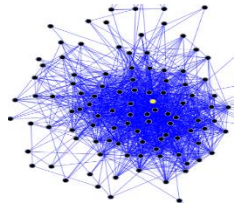




Avery Ching, Facebook @Strata, 2/13/2014

Yes, using 3940 machines

Do we really need large clusters?



- Laptops are sufficient?

Twenty pagerank iterations

System	cores	twitter_rv	uk_2007_05
Spark	128	857s	1759s
Giraph	128	596s	1235s
GraphLab	128	249s	833s
GraphX	128	419s	462s
Single thread	1	300s	651s

Label propagation to fixed-point (graph connectivity)

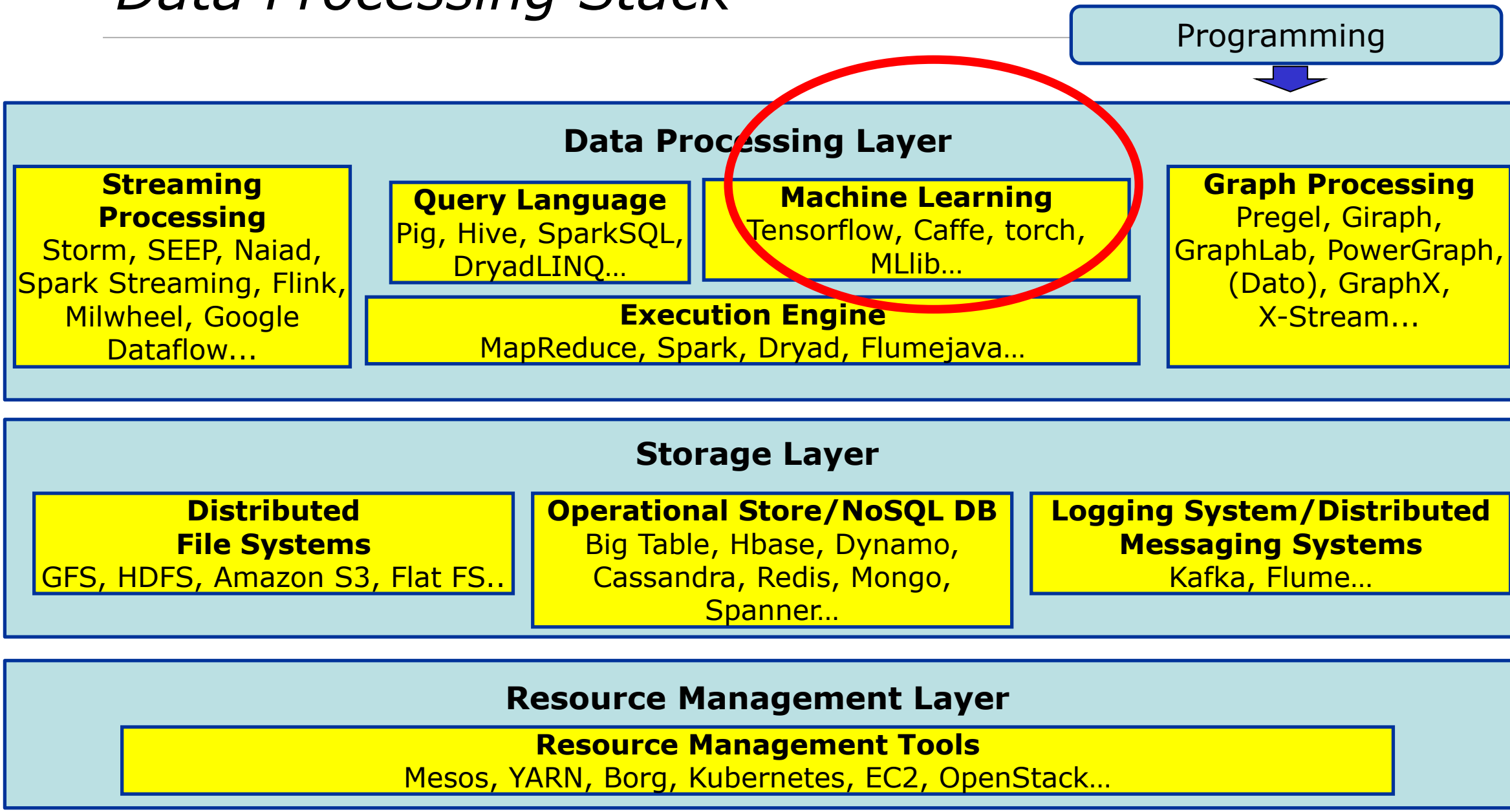
System	cores	twitter_rv	uk_2007_05
Spark	128	1784s	8000s+
Giraph	128	200s	8000s+
GraphLab	128	242s	714s
GraphX	128	251s	800s
Single thread	1	153s	417s

Fixed-point iteration: All vertices active in each iteration (50% computation, 50% communication)

Traversal: Search proceeds in a frontier (90% computation, 10% communication)

from Frank McSherry HotOS 2015

Data Processing Stack

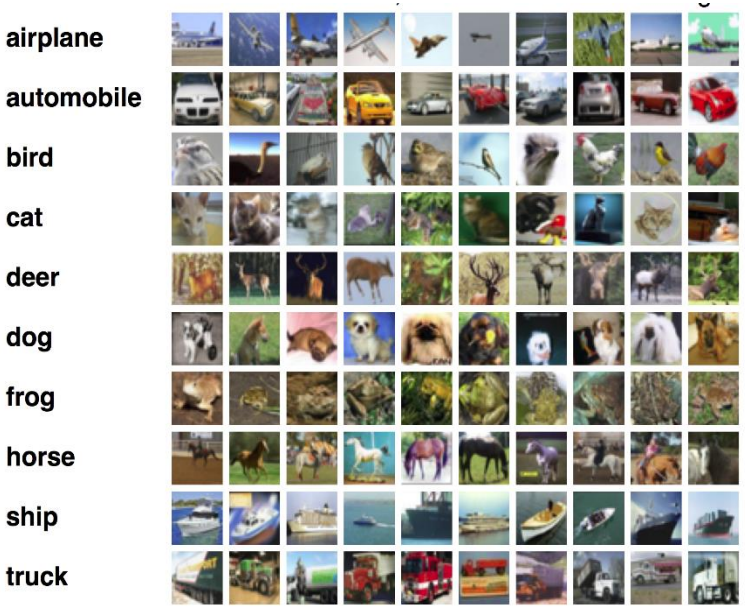


Data Processing for Neural Networks

- Practicalities of training Neural Networks
- Leveraging heterogeneous hardware

Modern Neural Networks Applications:

Image Classification



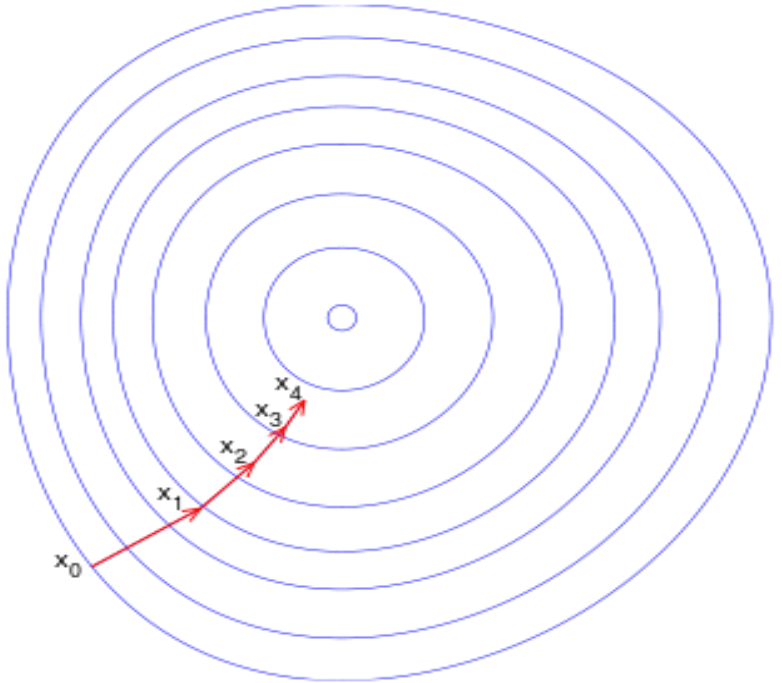
Reinforcement Learning

Image courtesy: The Guardian

Image courtesy: Twitter - Deep Mind AI

Training Procedure

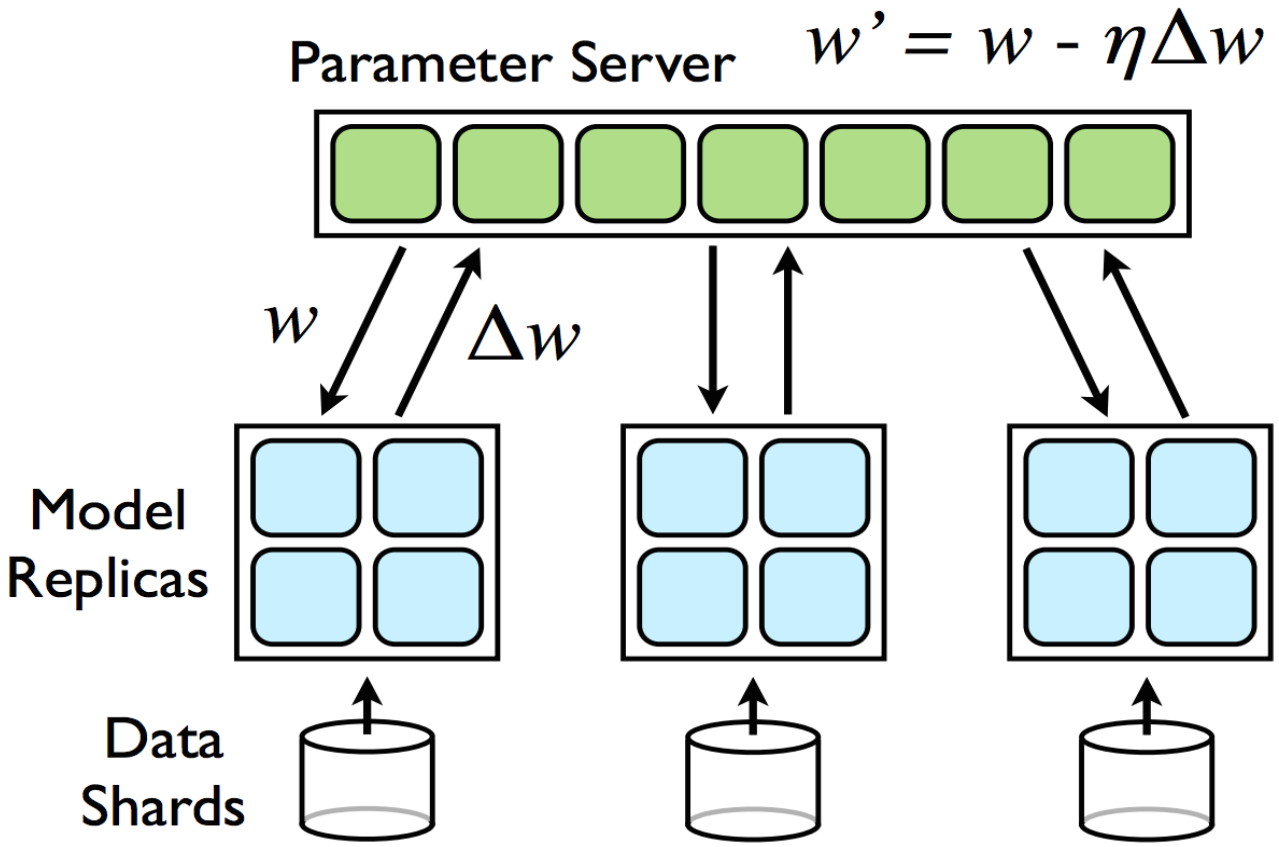
- Optimise the weights of the neurons to yield good predictions
- Use minibatches of inputs to estimate the gradient



Single Machine Setup

- One or more beefy GPUs

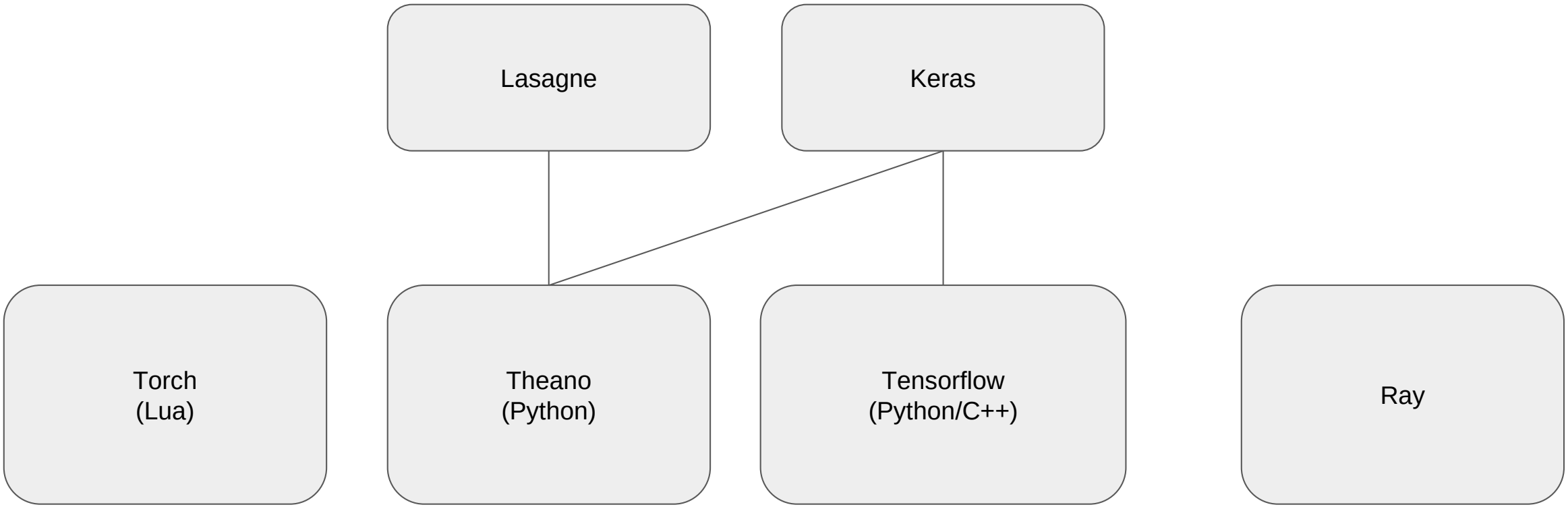
Distribution: Parameter Server Architecture



- Can exploit both Data Parallelism and Model Parallelism

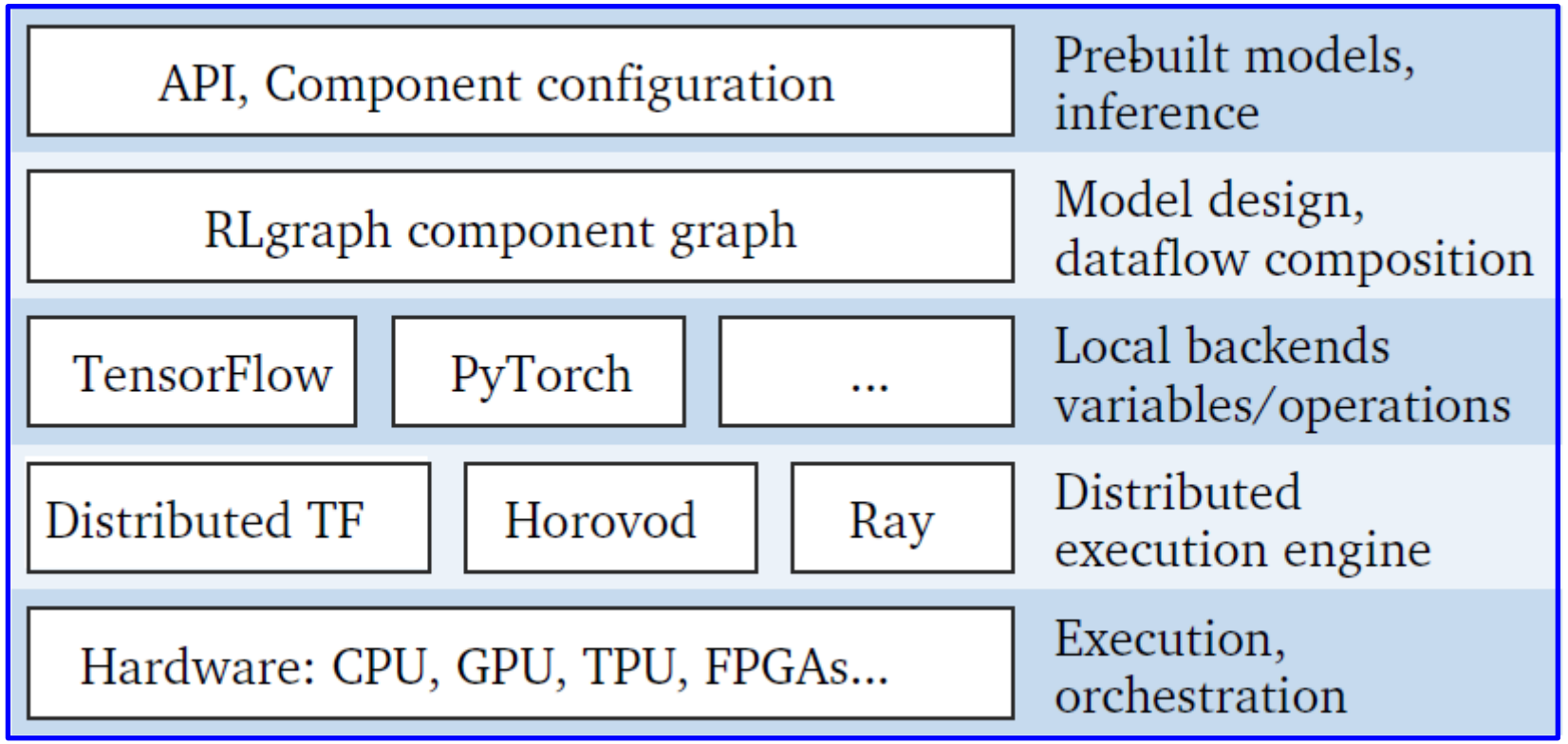
Source: Dean et al.: Large Scale Distributed Deep Networks

Software Platform for ML Applications

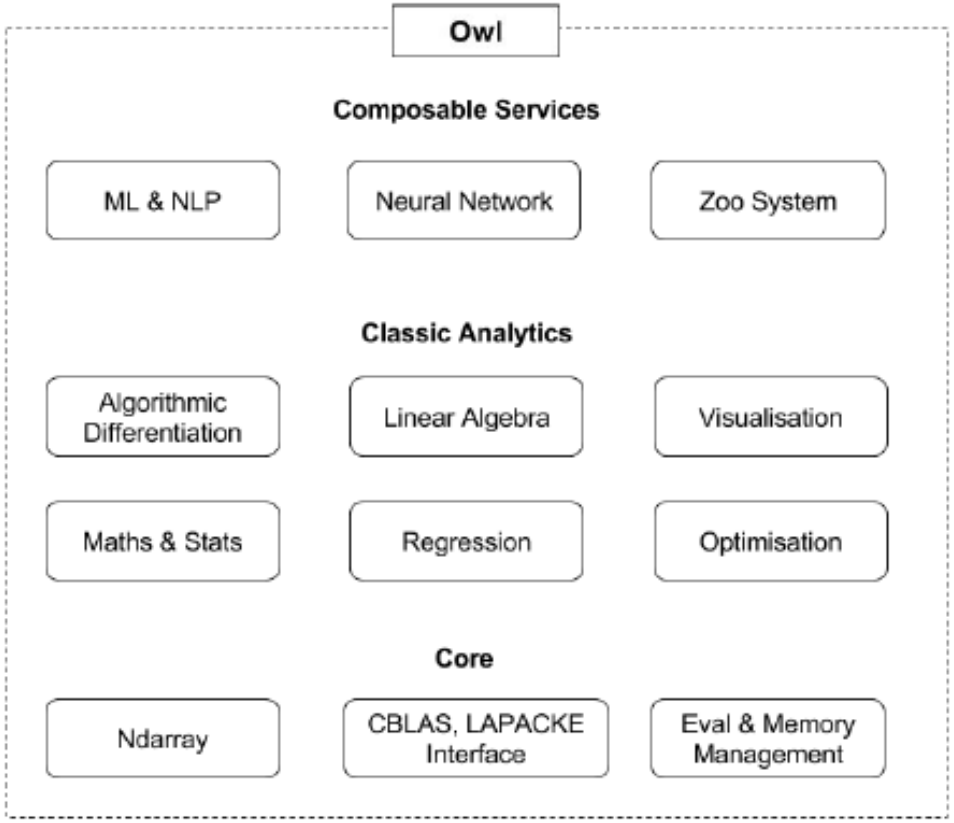


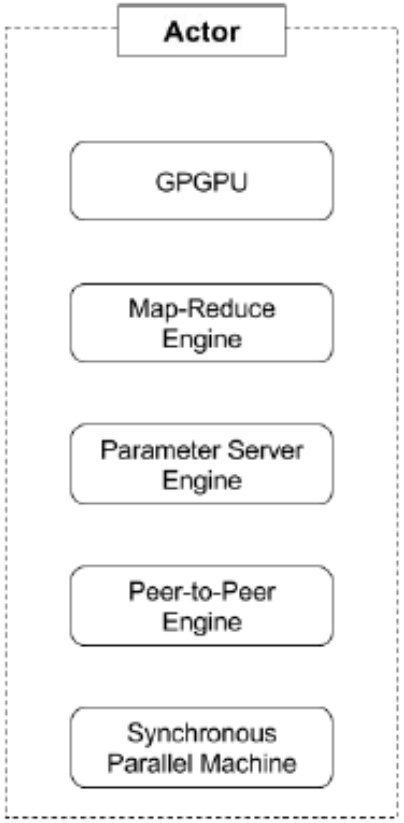
RLgraph: Dataflow Composition

- Our group's work



OWL Architecture for OCaml





Owl + Actor = Distributed & Parallel Analytics

Owl provides numerical backend; whereas Actor implements the mechanisms of distributed and parallel computing. Two parts are connected with functors.

Various system backends allows us to write code once, then run it from cloud to edge devices, even in browsers.

Same code can run in both sequential and parallel mode with Actor engine.

By Liang Wang in 2018

Computer Systems Optimisation

- What is performance?
  - Resource usage (e.g. time, power)
  - Computational properties (e.g. accuracy, fairness, latency)
- How do we improve it:
  - Manual tuning
  - Runtime autotuning
  - Static time autotuning

Build a scheduler for a distributed system

- What heuristics to use?
- How to detect failed workers?

Manually tune the system

✓ Uses programmer understanding
✗ Takes time
✗ Specific to a single environment

Use a Black-box optimization tool

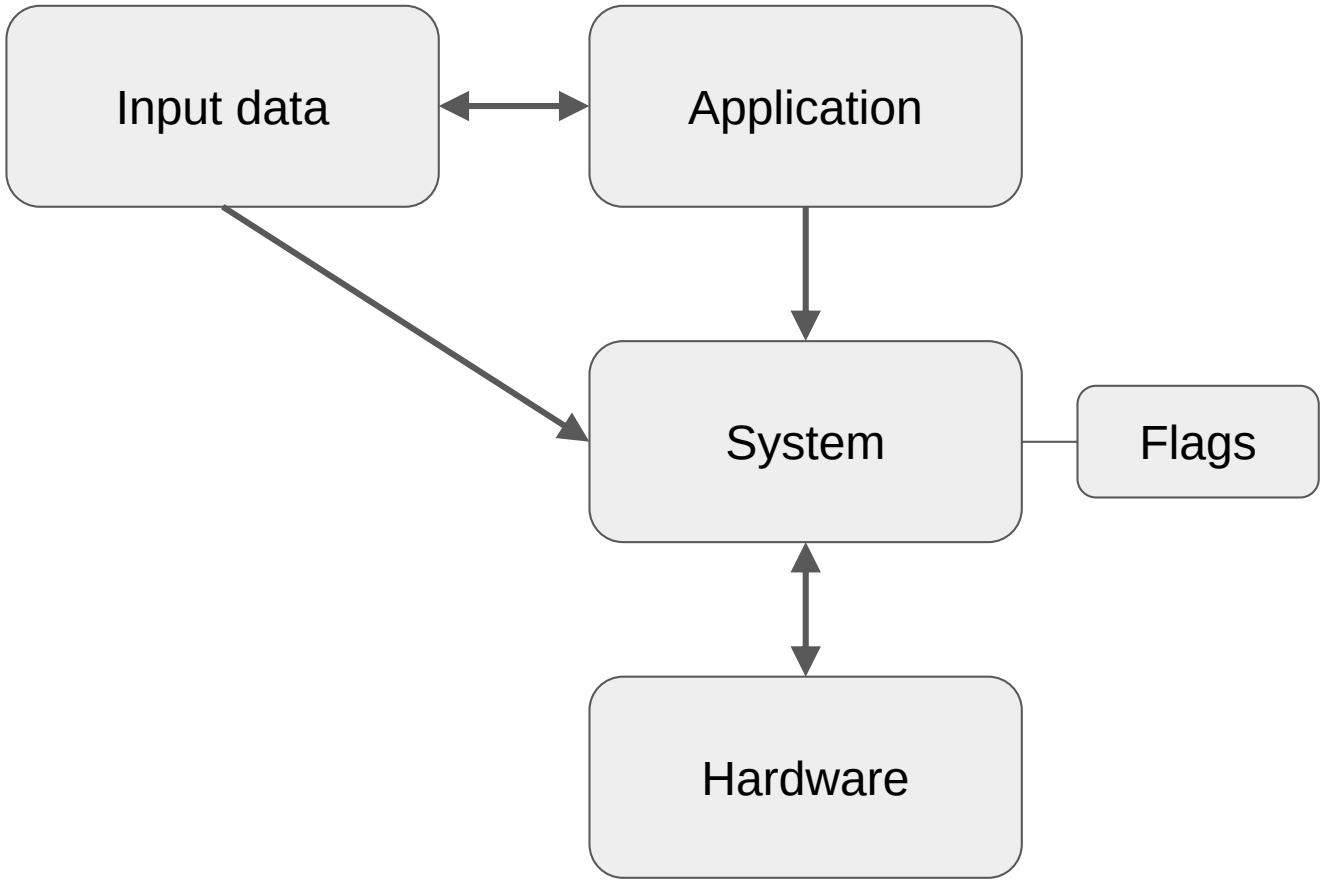
✓ Can surpass human tuning
✗ Does not scale to large parameter spaces
✗ Can only handle simple parameters

Manual Tuning: Profiling

- Always the first step
- Simplest case: Poor man's profiler
  - Debugger + Pause
- Higher level tools
  - Perf, Vtune, Gprof…
- Distributed profiling: a difficult active research area
  - No clock synchronisation guarantee
  - Many resources to consider
  - System logs can be leveraged

→ tune implementation based on profiling (never captures all interactions)

Auto-tuning systems



- Properties:
  - Many dimensions
  - Expensive objective function
  - Understanding of the underlying behaviour

Runtime Autotuning

- Plug and play to respond to a changing environment

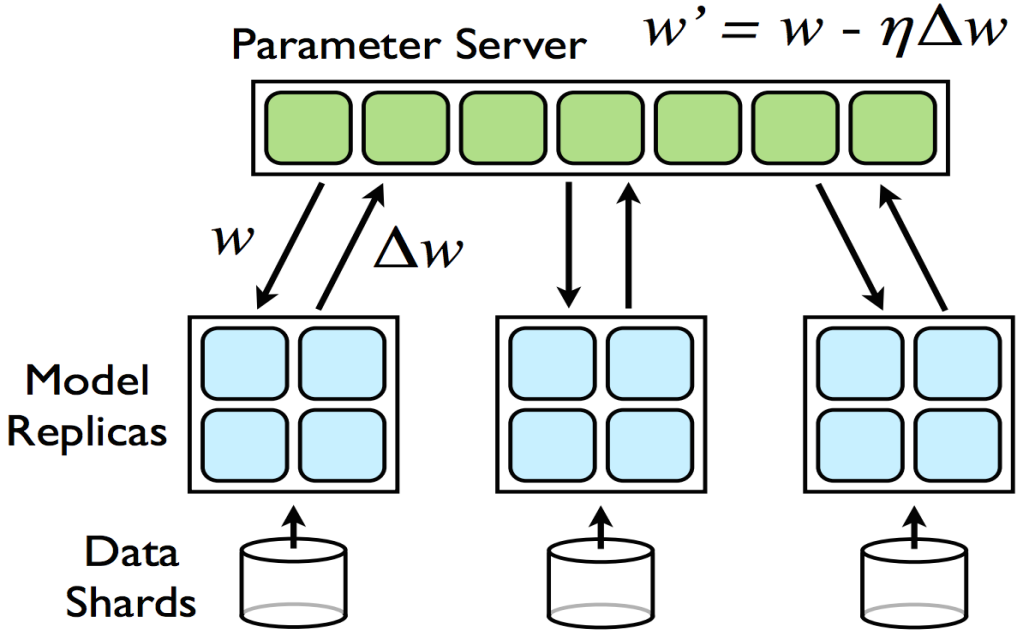
  For parameters that:

  - Can dynamically change
  - Can leverage runtime measurement
  - E.g. Locking strategy

- Often grounded in Control Theory

Optimising Scheduling on Heterogeneous Cluster

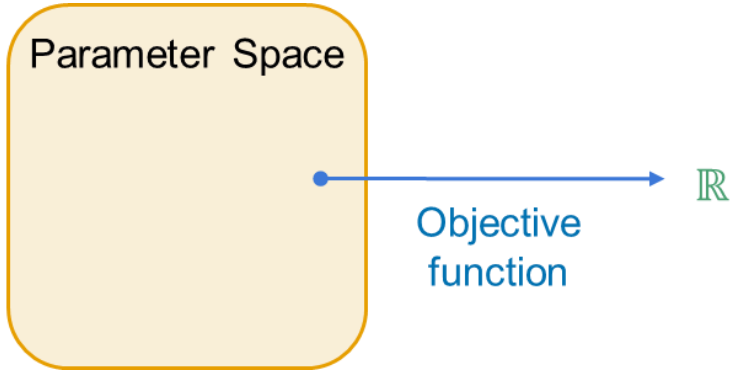
- Which machines to use as workers? As parameter servers?
  - ↗workers => ↗computational power & ↗communication
- How much work to schedule on each worker?
  - Must load balance



Static time Autotuning

Especially useful when:

- There is a variety of environments (hardware, input distributions)
- The parameter space is difficult to explore manually
- Defining a parameter space
  - e.g. Petabricks: A language and compiler for algorithmic choice (2009)
    - BNF-like language for parameter space
    - Uses an evolutionary algorithm for optimisation
    - Applied to Sort, matrix multiplication



Ways to do an Optimisation

Random search: No risk of 'getting stuck'
potentially many samples required

Evolution strategies: Evaluate permutations against fitness function

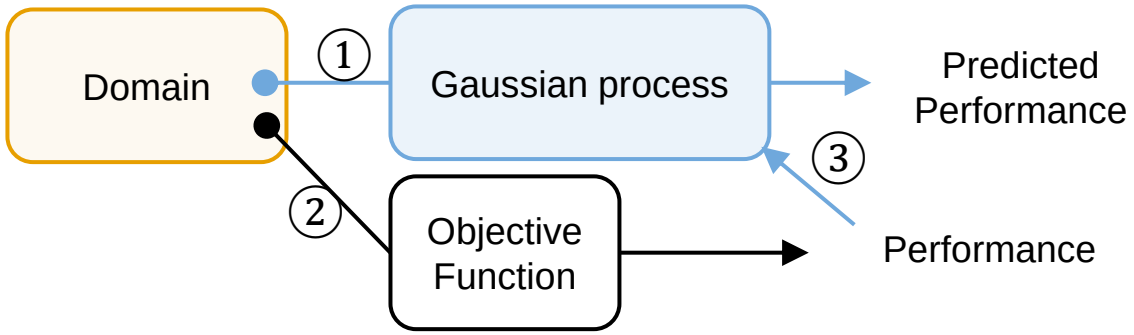
Bayes Opt: Sample efficient, requires continuous function, some configuration

Random Search	Genetic algorithm / Simulated annealing	Bayesian Optimisation
No overhead	Slight overhead	High overhead
High #evaluation	Medium-high #evaluation	Low #evaluation

Bayesian optimisation

- For when Objective function is expensive (e.g. NN hyper-parameter)

→Iteratively build a probabilistic model of objective function



① Find promising point (parameter values with high performance value in the model)
② Evaluate the objective function at that point
③ Update the model to reflect this new measurement

Pros:

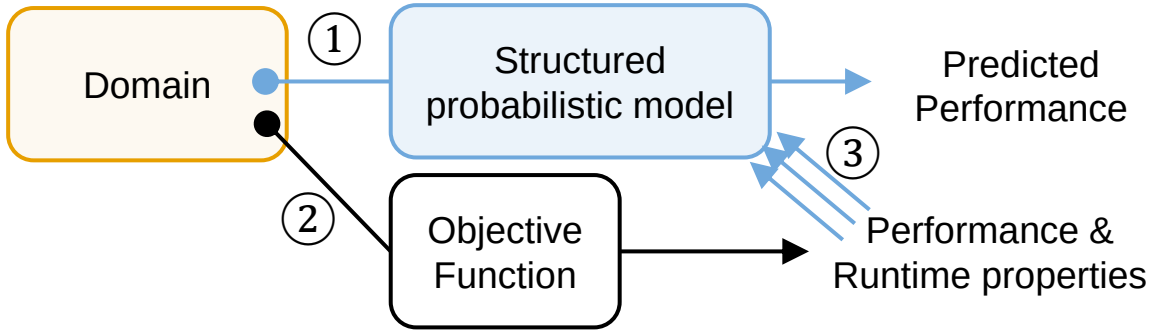
✓ Data efficient: converges in few iterations
✓ Able to deal with noisy observations

Cons:

✗ In many dimensions, model does not converge to the objective function

Solution: Use the known structure of the optimisation problem

Structured Bayesian Optimisation



Three desirable properties:

- Able to use many measurements
- Understand the trend of the objective function
- High precision in the region of the optimum

✓ Better convergence
✓ Use all measurements

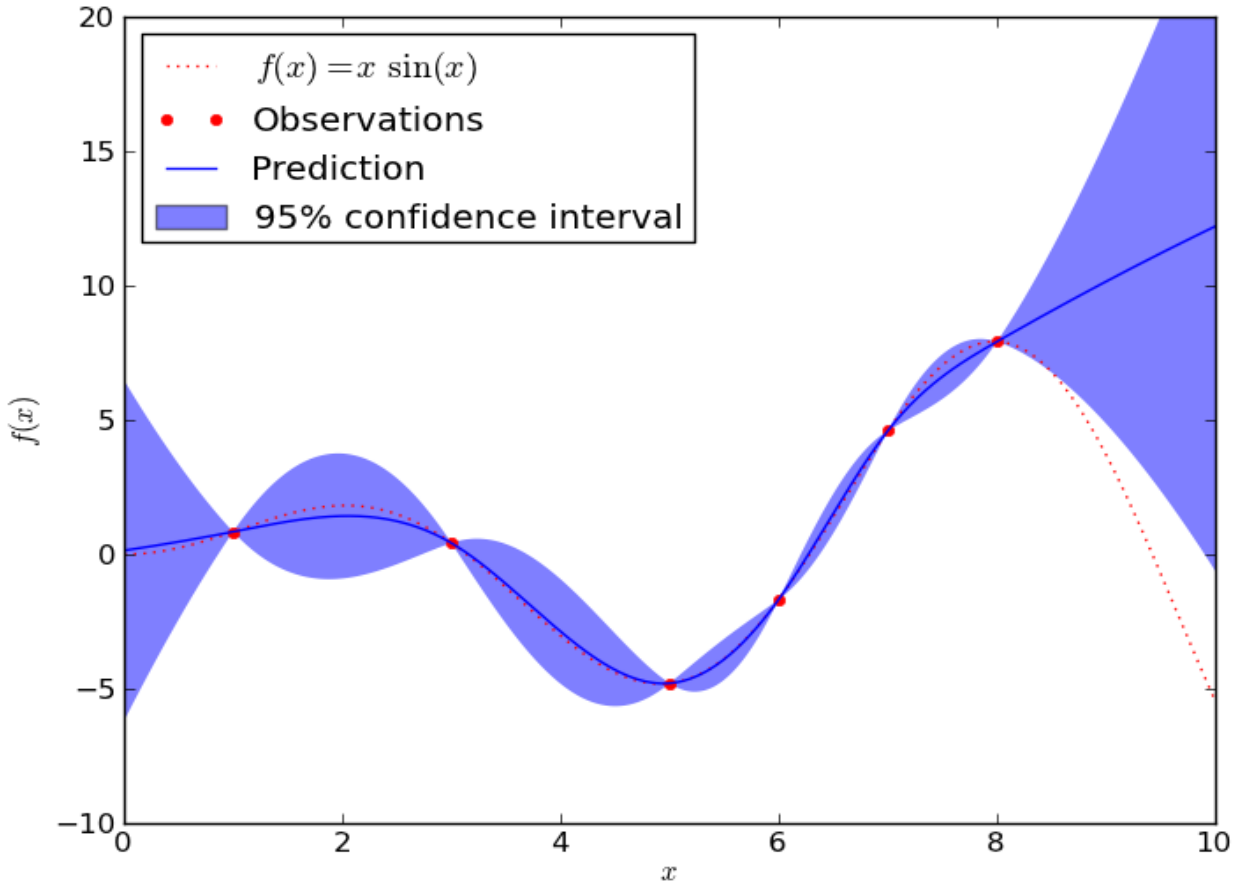
- **BOAT:** a framework to build **B**esp**O**ke **A**uto-**T**uners
- It includes a probabilistic library to express these models

- V. Dalibard, M. Schaarschmidt, and E. Yoneki: BOAT: Building Auto-Tuners with Structured Bayesian Optimization, WWW 2017. (Morning Paper on May 18, 2017)

Probabilistic Model for Bayesian optimisation

Gaussian processes:

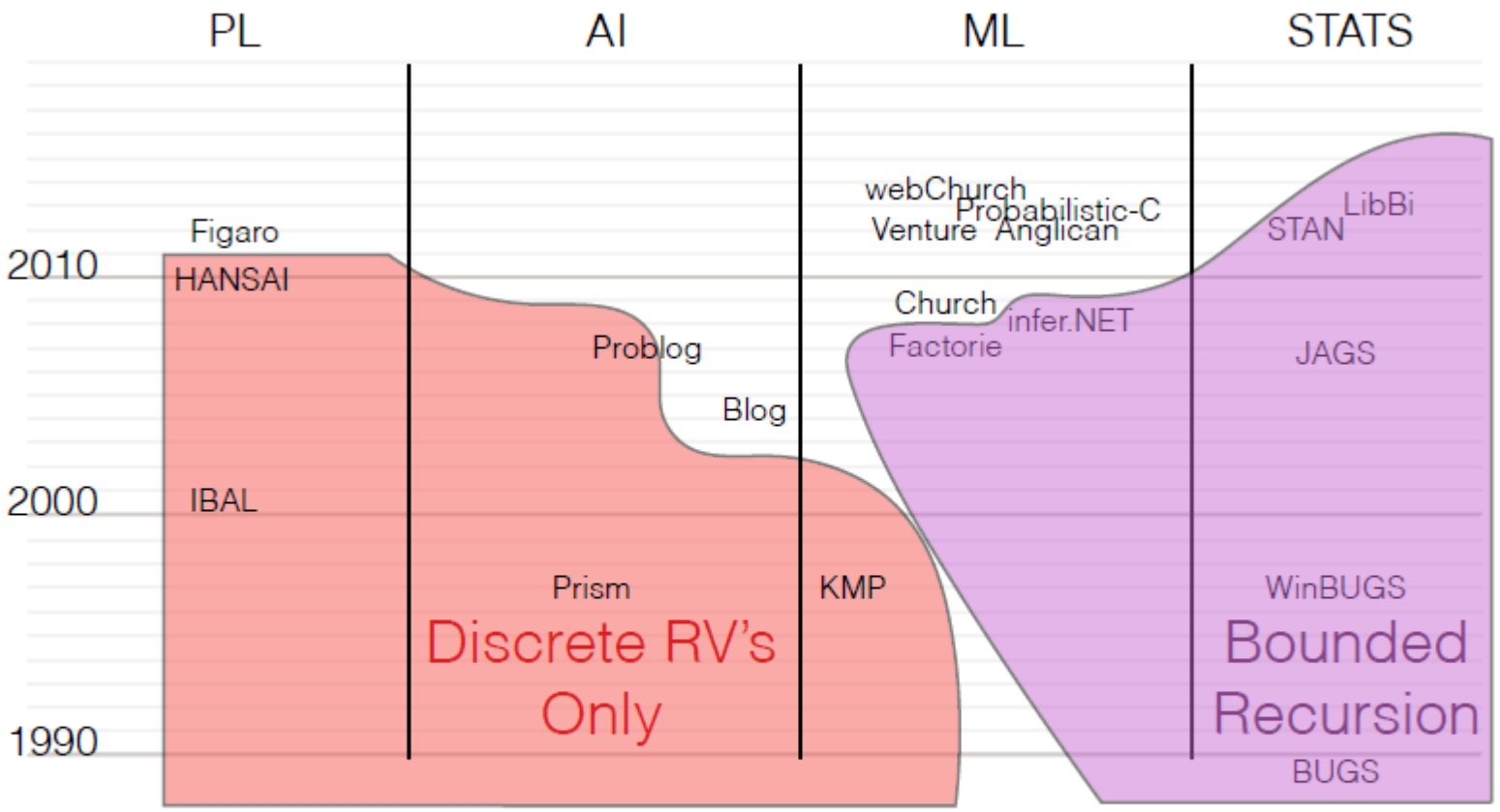
- Do regression: $\mathbb{R}^n \rightarrow \mathbb{R}$
- $O(N^3)$
- Allow for uncertainty



Probabilistic Model

- Probabilistic models incorporate random variables and probability distributions into the model
  - Deterministic model gives a single possible outcome
  - Probabilistic model gives a probability distribution
- Used for various probabilistic logic inference (e.g. MCMC-based inference, Bayesian inference…)

Probabilistic Programming



Edward based on Python

Probabilistic C++

Improbable – Java

Simula Prolog

Computer Systems Optimisation Models

- Long-term planning: requires model of how actions affect future states. Only a few system optimisations fall into this category, e.g. network routing optimisation.
- Short-term dynamic control: major system components are under dynamic load, such as resource allocation and stream processing, where the future load is not statistically dependent on the current load. Bayesian optimisation is sufficient to optimise distinct workloads. For dynamic workload, Reinforcement Learning would perform better.
- Combinatorial optimisation: a set of options must be selected from a large set under potential rules of combination. For this situation, one can either learn online if the task is cheap via random sampling, or via RL and pre-training if the task is expensive, or massively parallel online training given sufficient resources.

Deep Reinforcement Learning

- Given a set of actions with some unknown reward distributions, maximise the cumulative reward by taking the actions sequentially, one action at each time step and obtaining a reward immediately.
- To find the optimal action, one needs to explore all the actions but not too much. At the same time, one needs to exploit the best action found so-far by exploring.
- What makes reinforcement learning different from other machine learning paradigms?
  - There is no supervisor, only a reward signal
  - Feedback is delayed, not instantaneous
  - Time really matters (sequential)
  - Agent's actions affect the subsequent data it receives

Image courtesy: The Guardian

Image courtesy: Twitter - Deep Mind AI

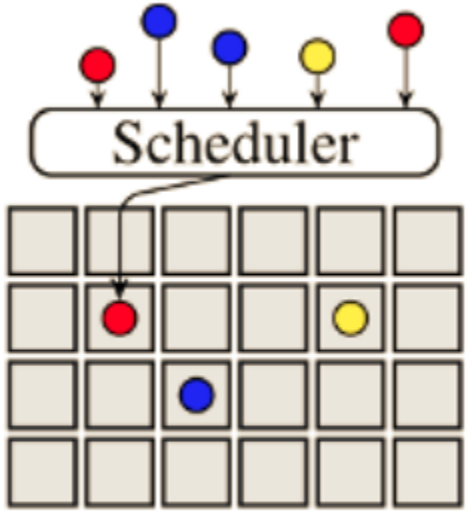
AlphaGo defeating the Go World Champion

Problem: Controlling dynamic behaviour

Many systems problems are combinatorial in nature

Assume workload dynamic, e.g. seasonality, load spikes, shared resources, failures..

- Algorithm: workload → behavior **distribution**
- Involves approximations to NP-complete problems, e.g. bin packing, sub-graph isomorphism, ..



Source: firmament.io

Trade-offs in dynamic control

Single static configuration/rule: E.g. FIFO scheduler

Online estimate of distributions: E.g. join-order in query planning

Workload clustering: Identify distinct classes, e.g. write-heavy, read-heavy workloads, per-class decisions

Fully adaptive: Optimal per-task behavior, unrealistic in practice

Robust, predictable, low deployment cost

Analytical overhead

Training/deployment cost factor

Practical Issues continued…

- Many deep learning tools, **no standard library** for modern RL (~2014-2018)
- **Exploration** in production system not a good idea
  - Unstable, unpredictable
- **Simulations** can oversimplify problem
  - Expensive to build, not justified versus gain
- **Online steps take too long**

Data Processing Stack

Programming

Data Processing Layer

Streaming Processing
Storm, SEEP, Naiad, Spark Streaming, Flink, Milwheel, Google Dataflow...

Query Language
Pig, Hive, SparkSQL, DryadLINQ…

Machine Learning
Tensorflow, Caffe, torch, MLlib…

Execution Engine
MapReduce, Spark, Dryad, Flumejava…

Graph Processing
Pregel, Giraph, GraphLab, PowerGraph, (Dato), GraphX, X-Stream...

Storage Layer

Distributed File Systems
GFS, HDFS, Amazon S3, Flat FS..

Operational Store/NoSQL DB
Big Table, Hbase, Dynamo, Cassandra, Redis, Mongo, Spanner…

Logging System/Distributed Messaging Systems
Kafka, Flume…

Resource Management Layer

Resource Management Tools
Mesos, YARN, Borg, Kubernetes, EC2, OpenStack…

Parallel Processing Stack

Algorithmic Parameters

Application	Physics/Biology Apps	Commercial Apps	Social media analysis
API	Efficient, Declarative, Expressive Programming Language		
High-level	Parallel Programming: lang. (OpenMP, Eigen..) lib. (TensorFlow..)		
Low-level	Low-Level APIs (MPI, OpenCL…), Vector-Compiler, SIMD, SPMD…		
Hardware	Heterogeneous Hardware (CPU/GPU/ASIC, Memory, Disks), VM		

OPTIMISATION + Auto-Tuning


GPU cluster

Cluster (Amazon)


HPC

SSD



FPGA, TPU

Gap between Research and Practice

Device Placement Optimization with Reinforcement Learning

Azalia Mirhoseini [*1 2] Hieu Pham [*1 2] Quoc V. Le [1] Benoit Steiner [1] Rasmus Larsen [1] Yuefeng Zhou [1]
Naveen Kumar [3] Mohammad Norouzi [1] Samy Bengio [1] Jeff Dean [1]

20H with 80GPUs!

Topic Areas

Session 1: Introduction

Session 2: Data flow programming: Map/Reduce to TensorFlow

Session 3: Large-scale graph data processing

Session 4: Stream Data Processing + Guest lecture

Session 5: Hands-on Tutorial: Map/Reduce and Deep Neural Network

Session 6: Machine Learning for Optimisation of Computer Systems

Session 7: Task scheduling, Performance, and Resource Optimisation

Session 8: Project Study Presentation

Summary

- R244 course web page:

www.cl.cam.ac.uk/~ey204/teaching/ACS/R244_2018_2019

- Enjoy the course!