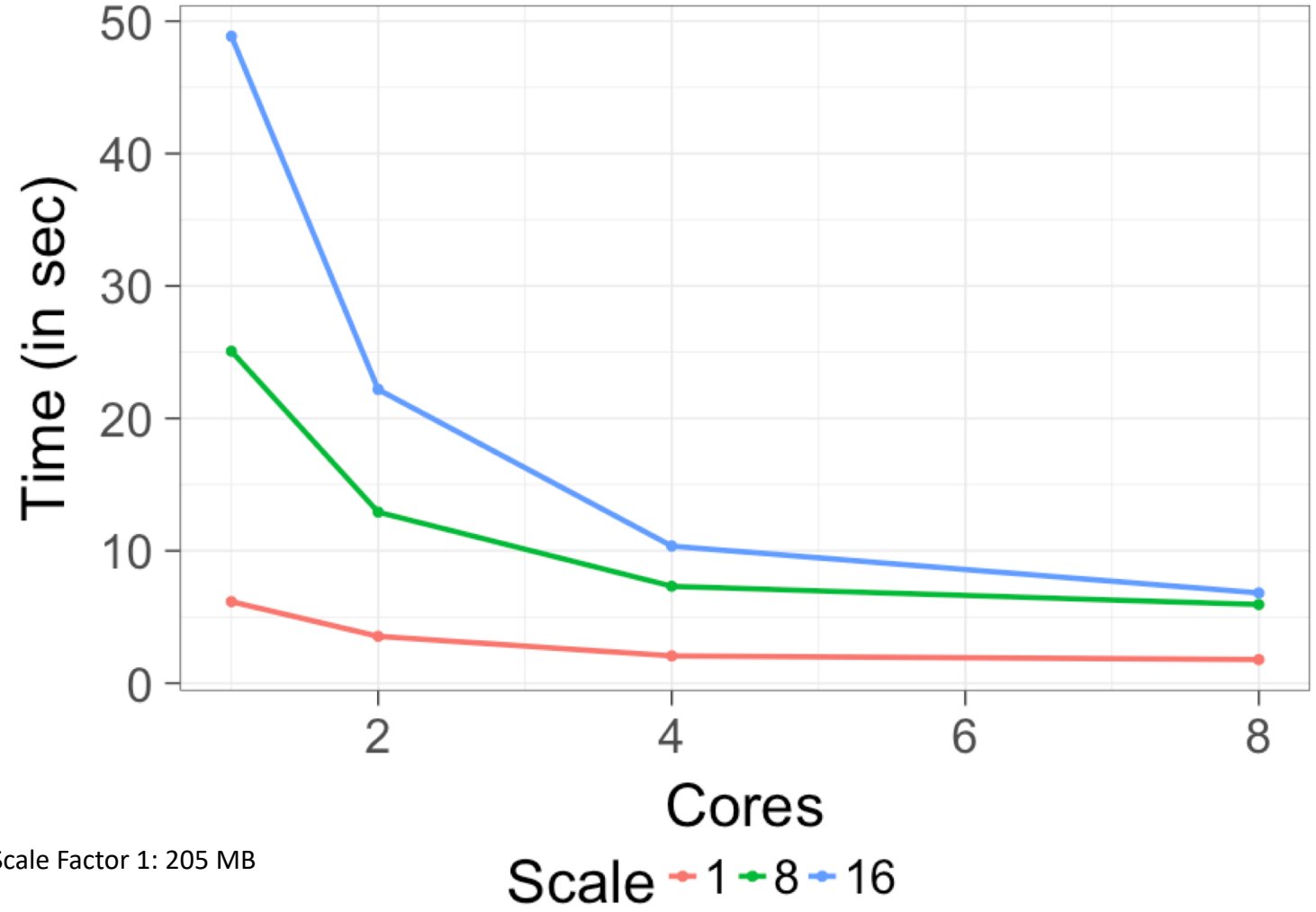
There are only two kinds of languages: the ones people complain about and the ones nobody uses —Bjarne Stroustrup

Scalable Machine Learning.

Parallelization Approaches

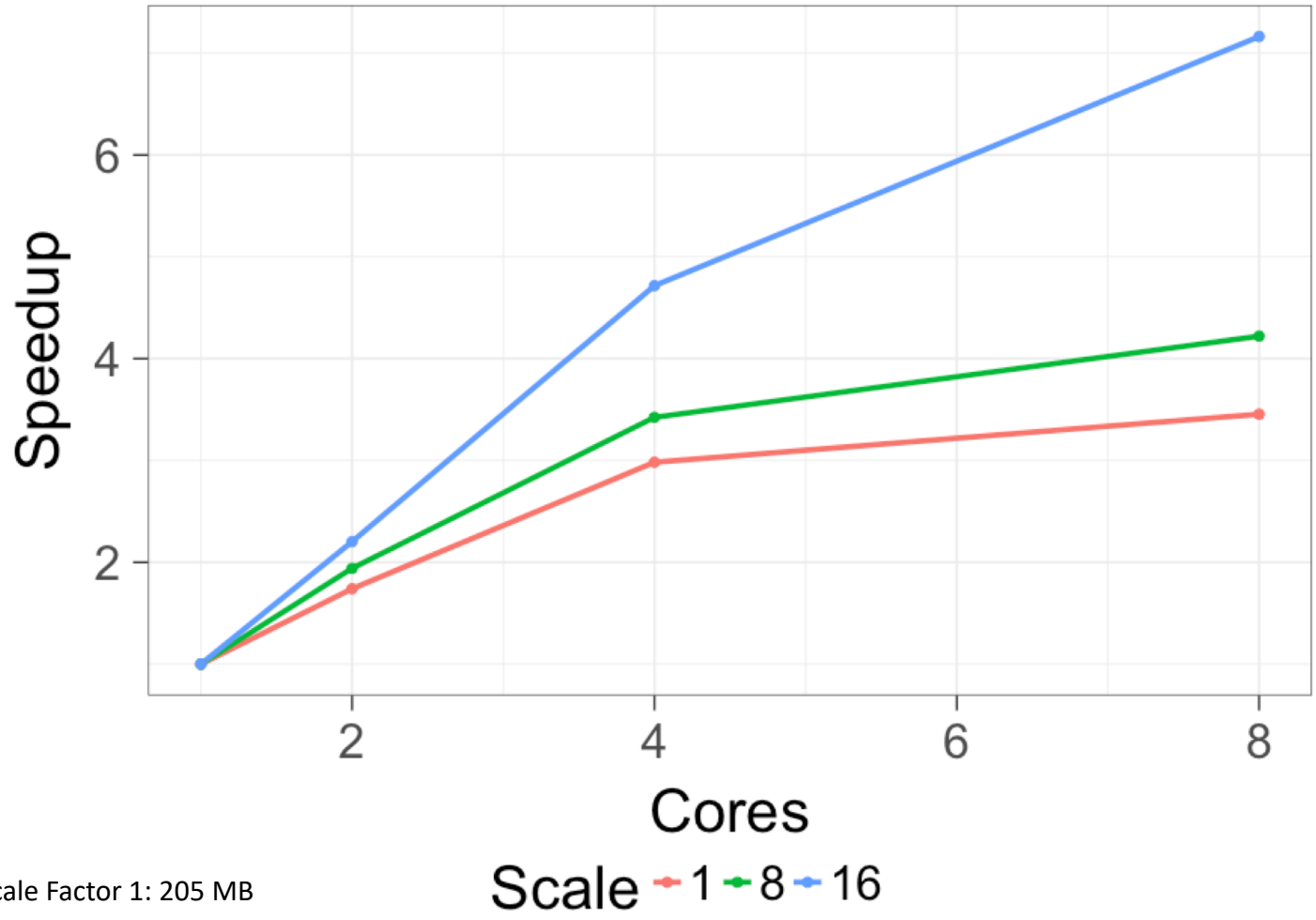
Distributed Deep Learning

Runtime



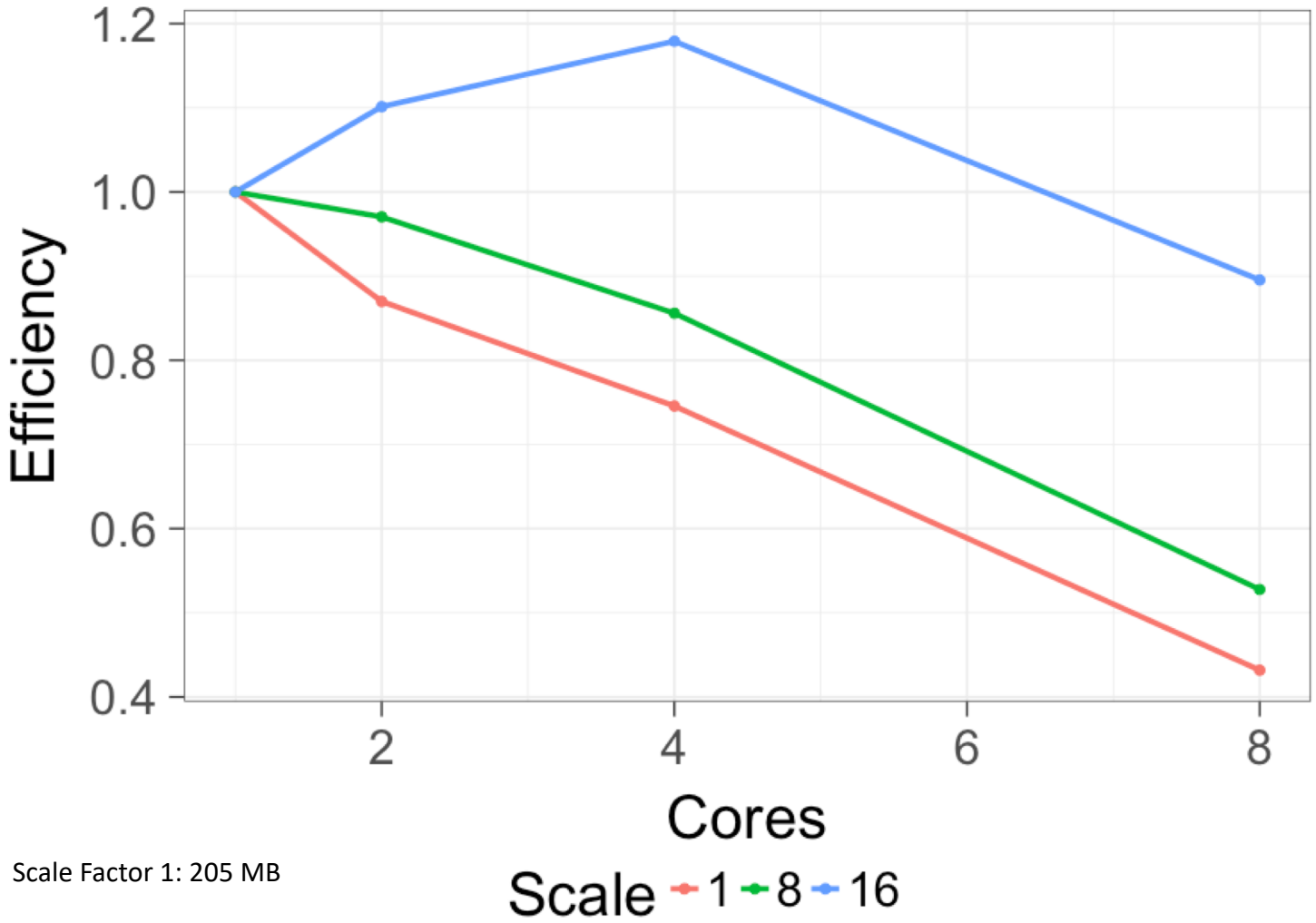
Scale Factor 1: 205 MB

Speedup



Scale Factor 1: 205 MB

Efficiency



Scale Factor 1: 205 MB

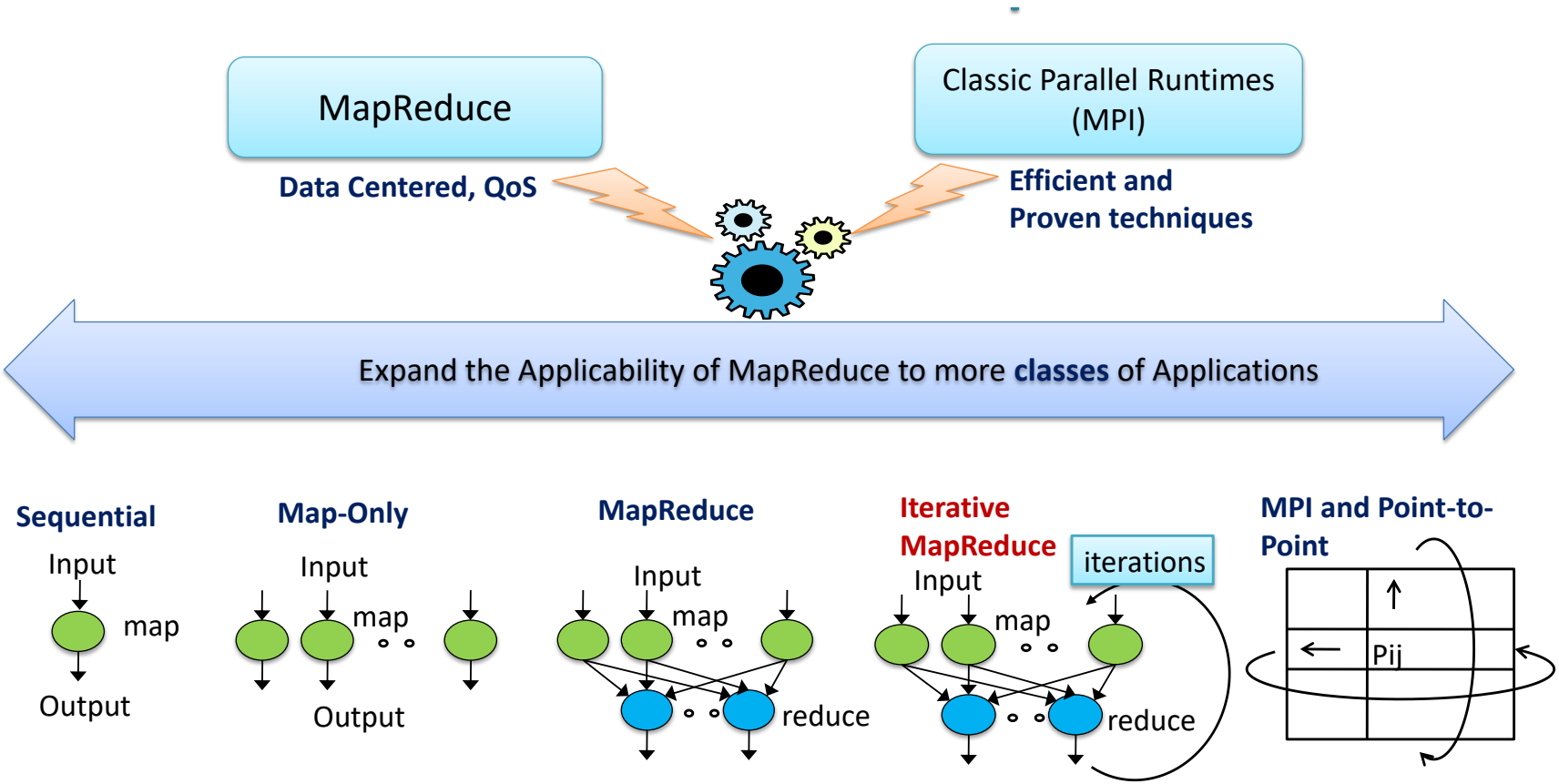
Limitations of Machine Learning Libraries.

- Limitations of Python and R standard libraries:
  - Model has to fit into memory of a single machine
  - Computational resources on a single machine constrained
- Mitigations:
  - Some libraries provide disk-based extensions for datasets that cannot be fitted into memory
  - Some algorithms provide thread-based parallelism to exploit multiple cores on a single machines
- Nevertheless, often long training times due to limitations in available compute resources
- Distributed libraries needed for:
  - To store data into memory of multiple machines
  - Provide more compute resources for training

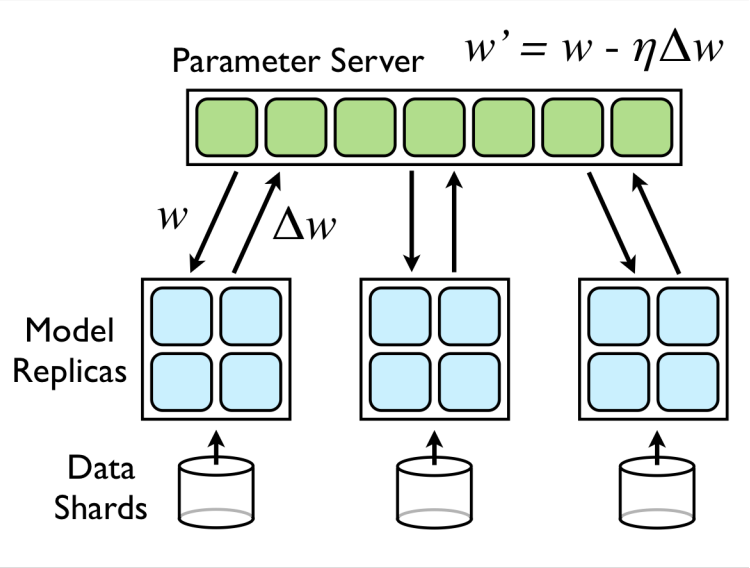
- Machine Learning (ML) needs high performance computing: Big Data and Big Model
- **All the different programming models** (Spark, Flink, Storm, Naiad, MPI/OpenMP) have the **same high level approach:**
  - First: Break Problem **Data** and/or **Model-parameters** into parts assigned to separate nodes, processes, threads
  - Then: In parallel, do computations typically leaving data untouched but changing model-parameters.

- Shortcomings of Hadoop/MapReduce for scalable machine learning:
  - Hadoop writes to disk frequently (e.g. during shuffle and between multi-stage jobs)
  - Spark and Flink spawn many processes and do not support AllReduce directly;
  - MPI does in-place combined reduce/broadcast
- **Iterative algorithms** are fundamental in learning a non-trivial model
- **Model training** and **Hyper-parameter tuning** steps run the iterative algorithms many times

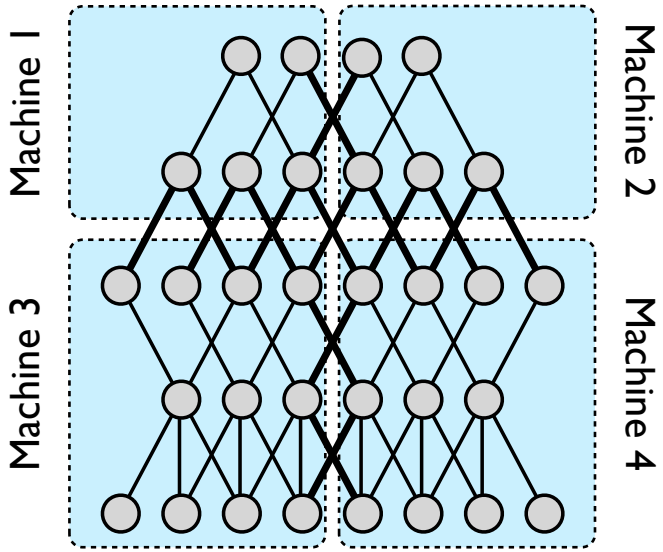
Iterative MapReduce.



Judy Qui, Scalable High Performance Data Analytics: Harp and Harp-DAAL, http://dsc.soic.indiana.edu/presentations/Qiu_Harp_DAAL_June_27_2017.pdf, 2017



Data Parallelism

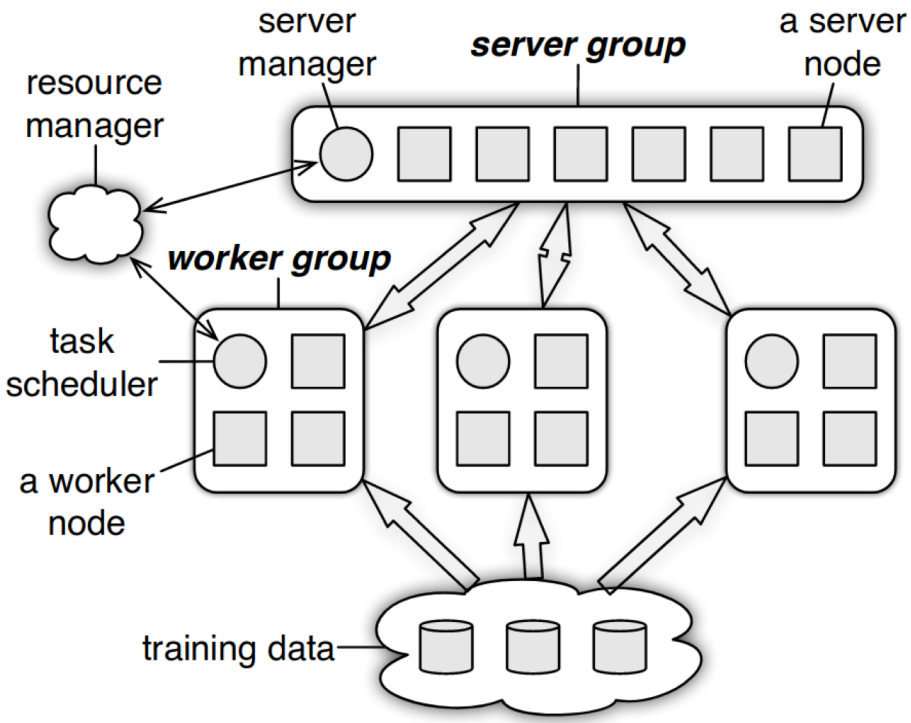


Model Parallelism

Source: Dean et al., Large Scale Distributed Deep Networks, https://static.googleusercontent.com/media/research.google.com/en//archive/large_deep_networks_nips2012.pdf, 2012

Parameter Server

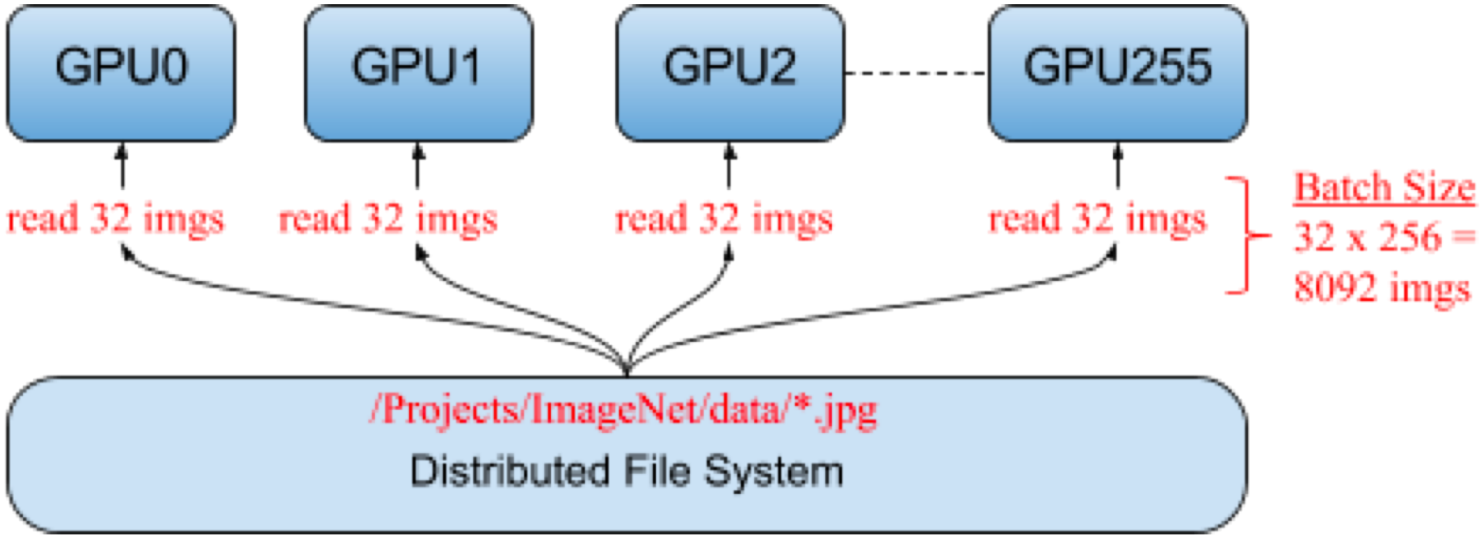
- Abstraction used for managing models across multiple worker processes
- Used for exploring data parallelism
- Properties:
  - Management of Worker nodes
  - Synchron vs. Asynchronous mode
  - Distributed vs. Non-Distributed



MLLib

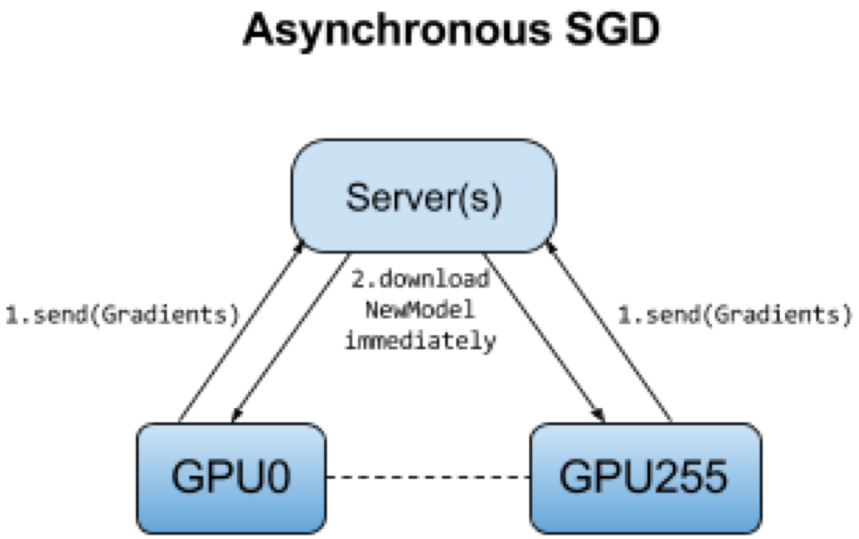
- Spark-based Machine Learning Library
  - Classification: logistic regression, naive Bayes, random forest, …
  - Collaborative filtering: ALS, …
  - clustering: k-means
  - Principal Component Analysis: SVD
- Spark In-Memory capabilities ideally suited for iterative machine learning implementation
- Driver serves in many use cases as parameter server

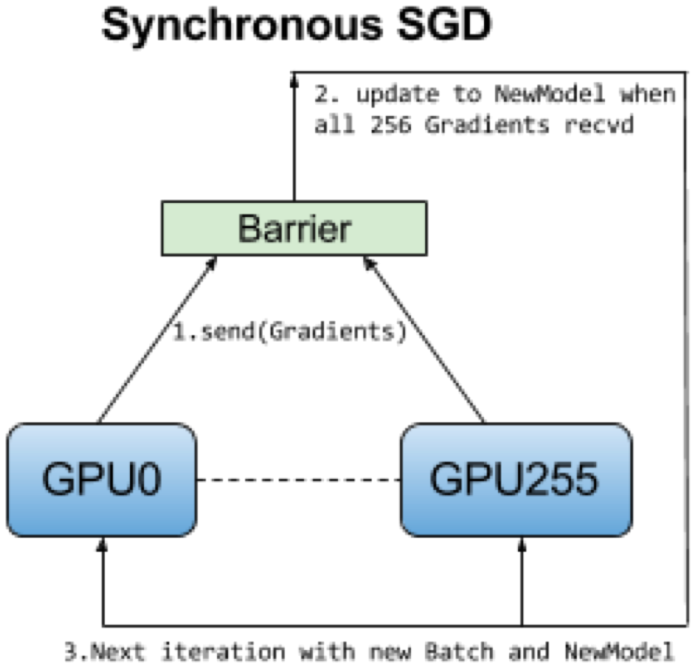
Distributed Deep Learning Using Tensorflow



https://www.oreilly.com/ideas/distributed-tensorflow

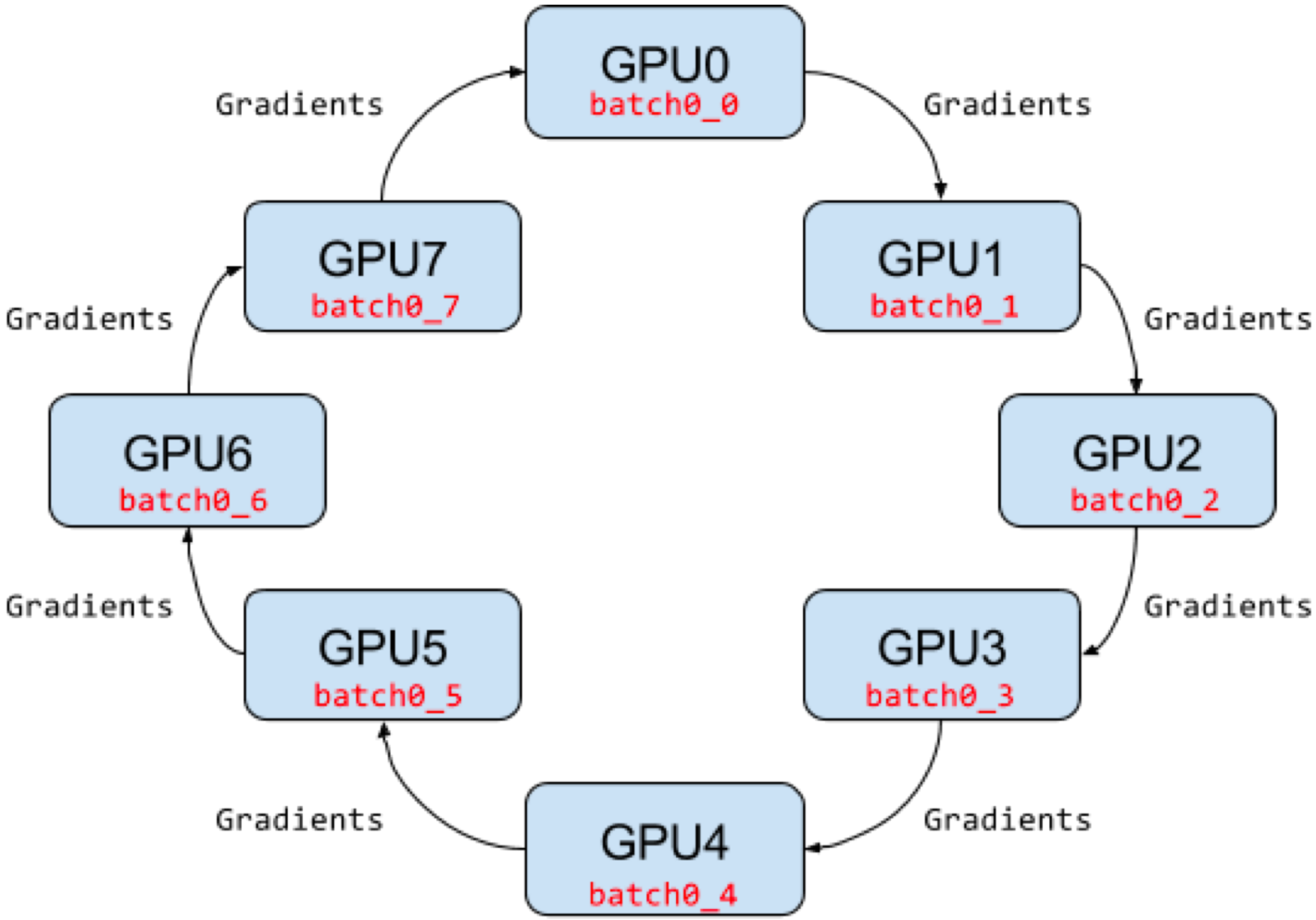
Distributed Deep Learning Using Tensorflow: Parameter Server





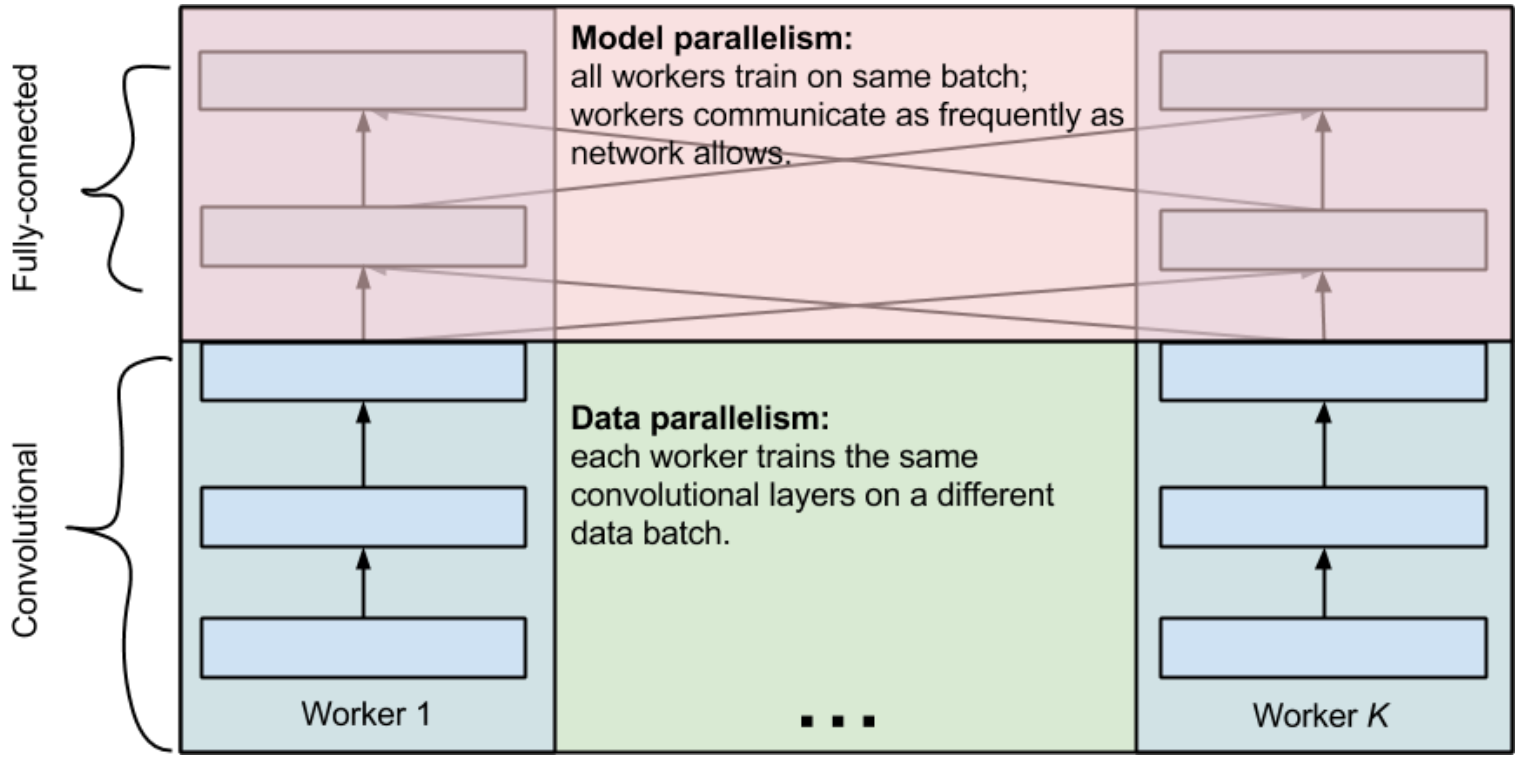
https://www.oreilly.com/ideas/distributed-tensorflow

Distributed Deep Learning Using Tensorflow: AllReduce

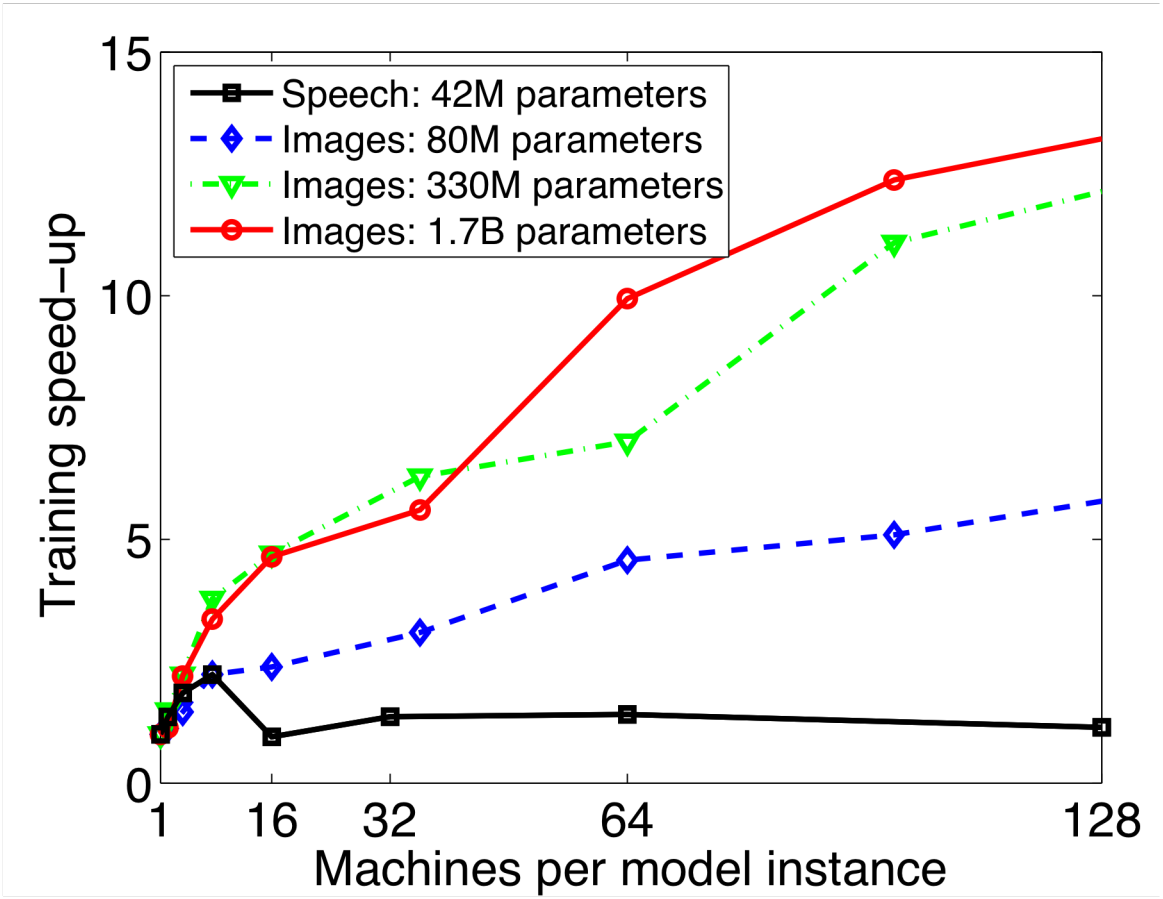


https://www.oreilly.com/ideas/distributed-tensorflow

Example AlexNet

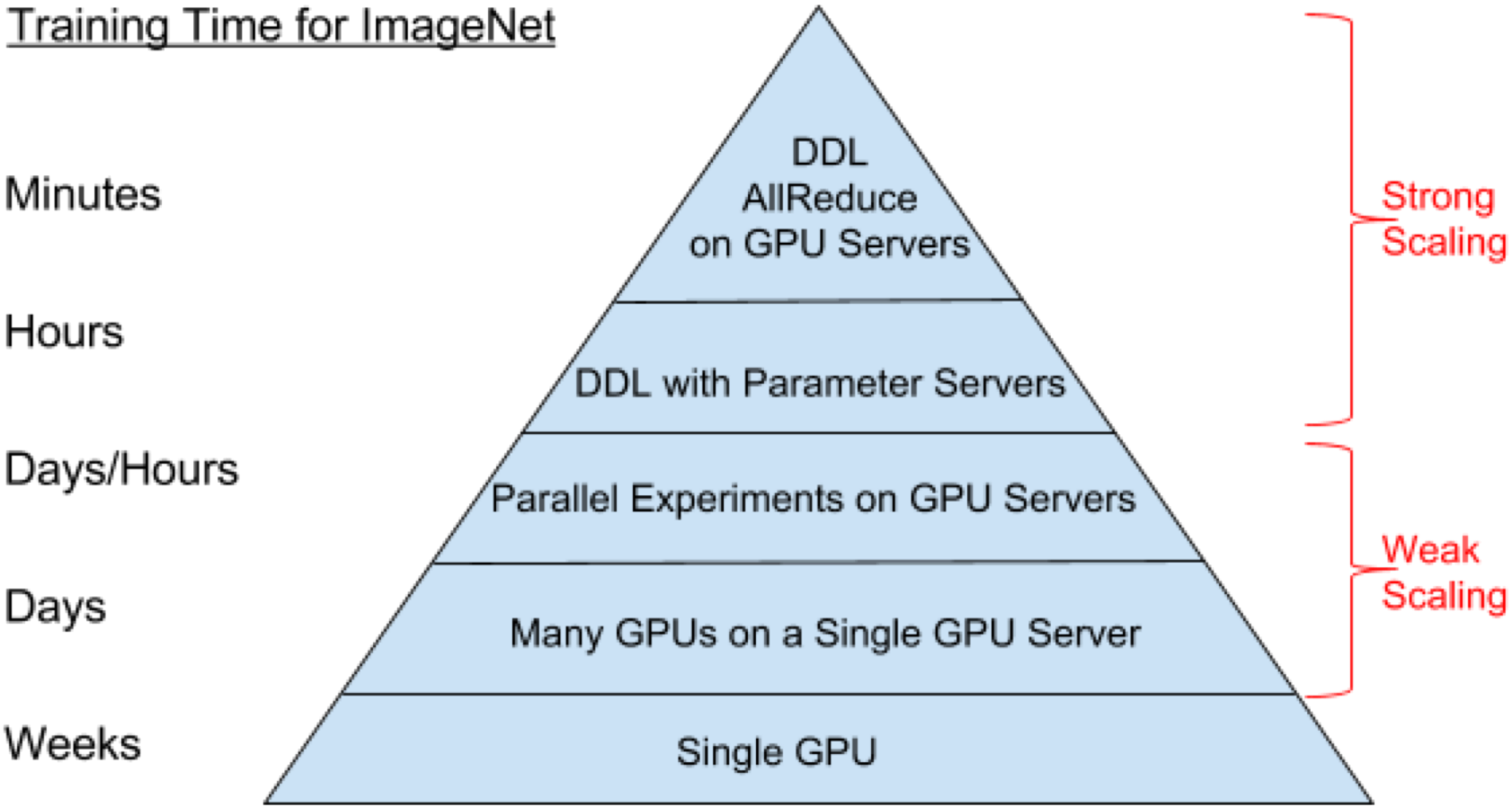


Google DistBelief: Model Parallelism



Source: Dean et al., Large Scale Distributed Deep Networks, https://static.googleusercontent.com/media/research.google.com/en//archive/large_deep_networks_nips2012.pdf, 2012

Deep Learning Hierarchy of Needs

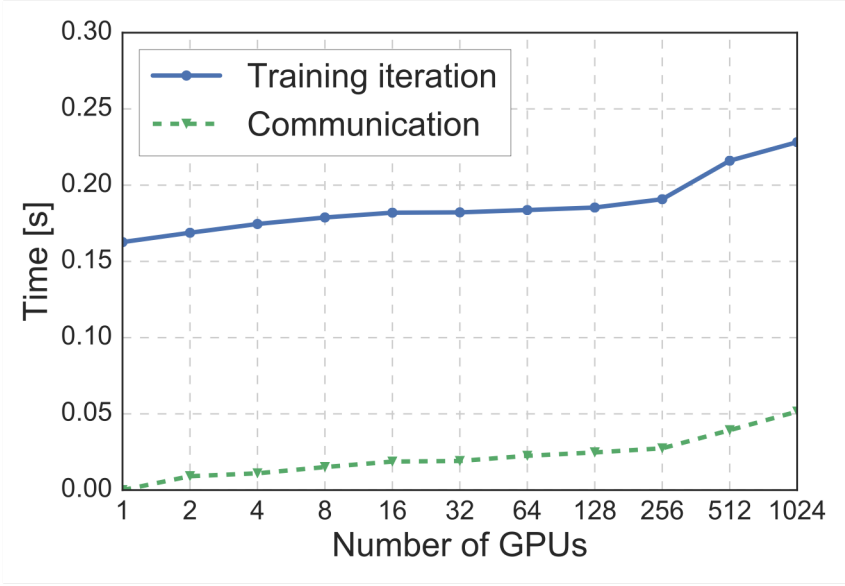


https://www.oreilly.com/ideas/distributed-tensorflow

Scaling Deep Learning Example

Table 1: 90-epoch training time and single-crop validation accuracy of ResNet-50 for ImageNet reported by different teams.

Team	Hardware	Software	Minibatch size	Time	Accuracy
He *et al.* [5]	Tesla P100 × 8	Caffe	256	29 hr	75.3 %
Goyal *et al.* [4]	Tesla P100 × 256	Caffe2	8,192	1 hr	76.3 %
Codreanu *et al.* [3]	KNL 7250 × 720	Intel Caffe	11,520	62 min	75.0 %
You *et al.* [10]	Xeon 8160 × 1600	Intel Caffe	16,000	31 min	75.3 %
This work	Tesla P100 × 1024	Chainer	32,768	15 min	74.9 %



128 nodes, where each node has two Intel Xeon E5-2667 processors (3.20 GHz, eight cores), 256 GB memory and eight NVIDIA Tesla P100 GPUs. The nodes are interconnected by Mellanox Infiniband FDR.

"As the underlying communication libraries, we used NCCL version 2.0.5 and Open MPI version 1.10.2. While computation was generally done in single precision, in order to reduce the communication overhead during all-reduce operations, we used half-precision floats for communication."

"Our scaling efficiency when using 1024 GPUs is 70% and 80% in comparison to single-GPU and single-node (i.e., 8 GPUs) baselines, respectively."

Akiba et al., Extremely Large Minibatch SGD: Training ResNet-50 on ImageNet in 15 Minutes, https://arxiv.org/pdf/1711.04325.pdf, 2017

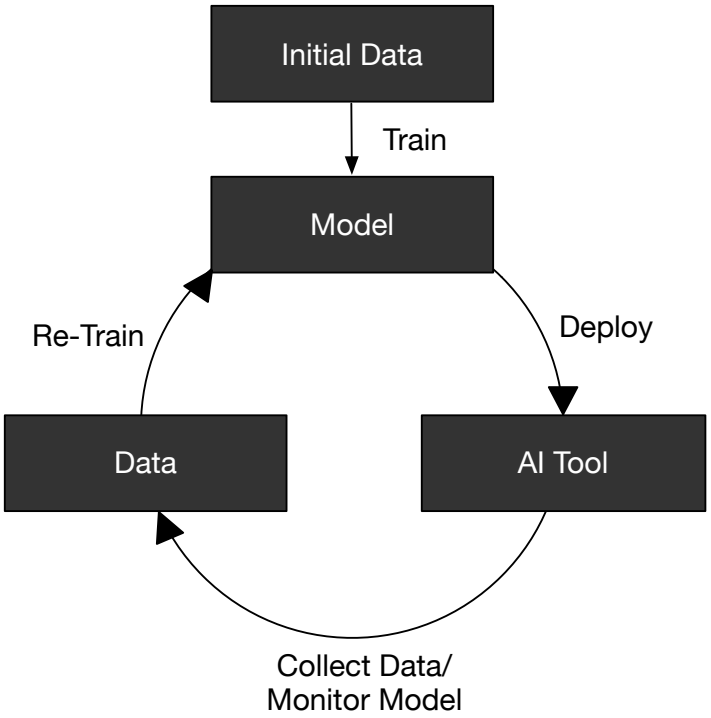
References

- Li et al., Scaling Distributed Machine Learning with the Parameter Server, OSDI, 2014
- Xing et al., Petuum: A New Platform for Distributed Machine Learning on Big Data, KDD, 2015
- Meng et al., MLlib: Machine Learning in Apache Spark, Journal of Machine Learning Research, 2016
- H2O, H2O version 3, 2017.

Operationalization of AI and Machine Learning.

- Edge Computing
- Streaming
- Model Management

Motivations

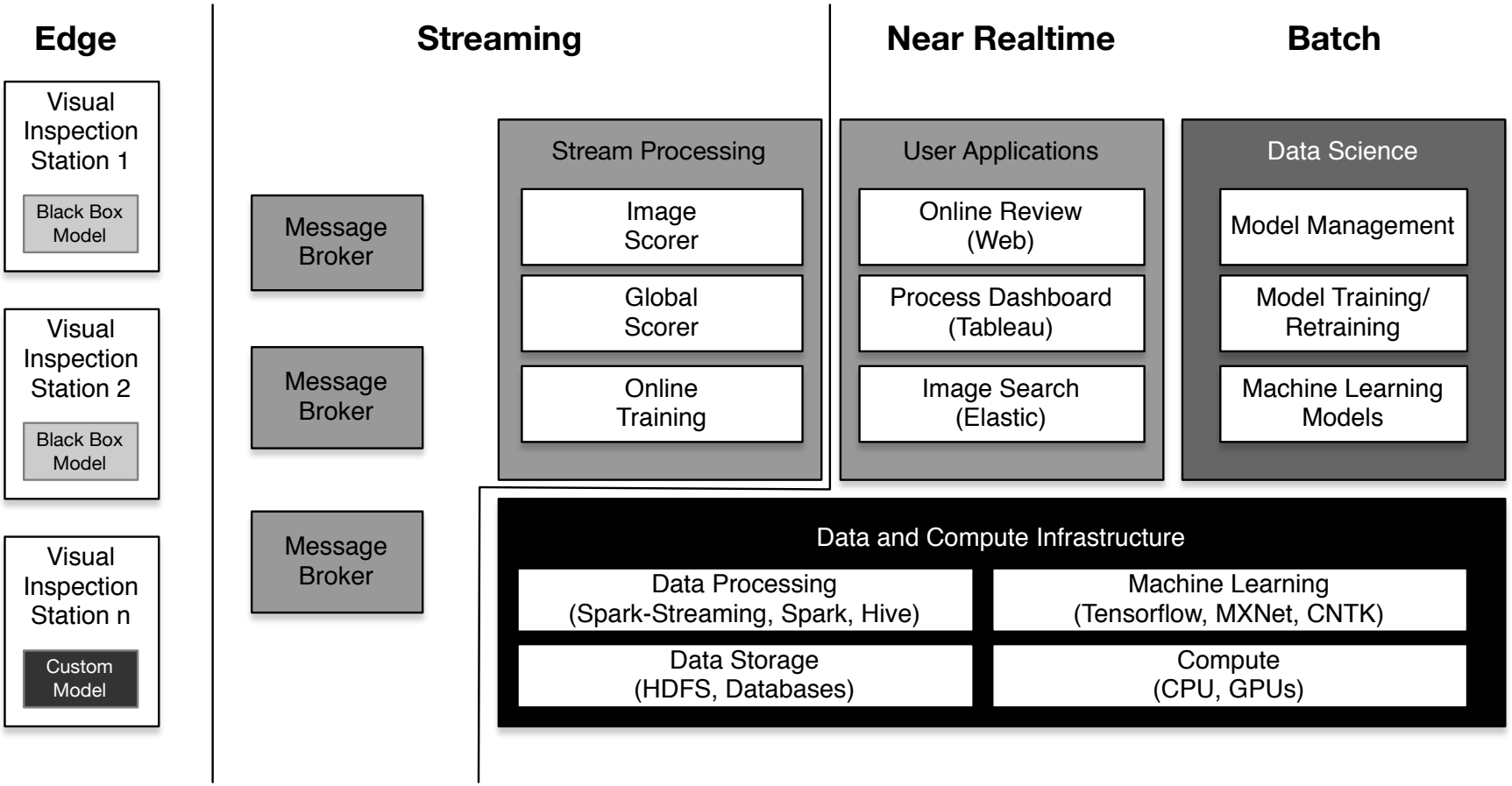


Developing AI applications is a complex task with many challenges from collecting data, training and deploying models.

Architectures need to support end-to-end data collection, training and deployment lifecycle.

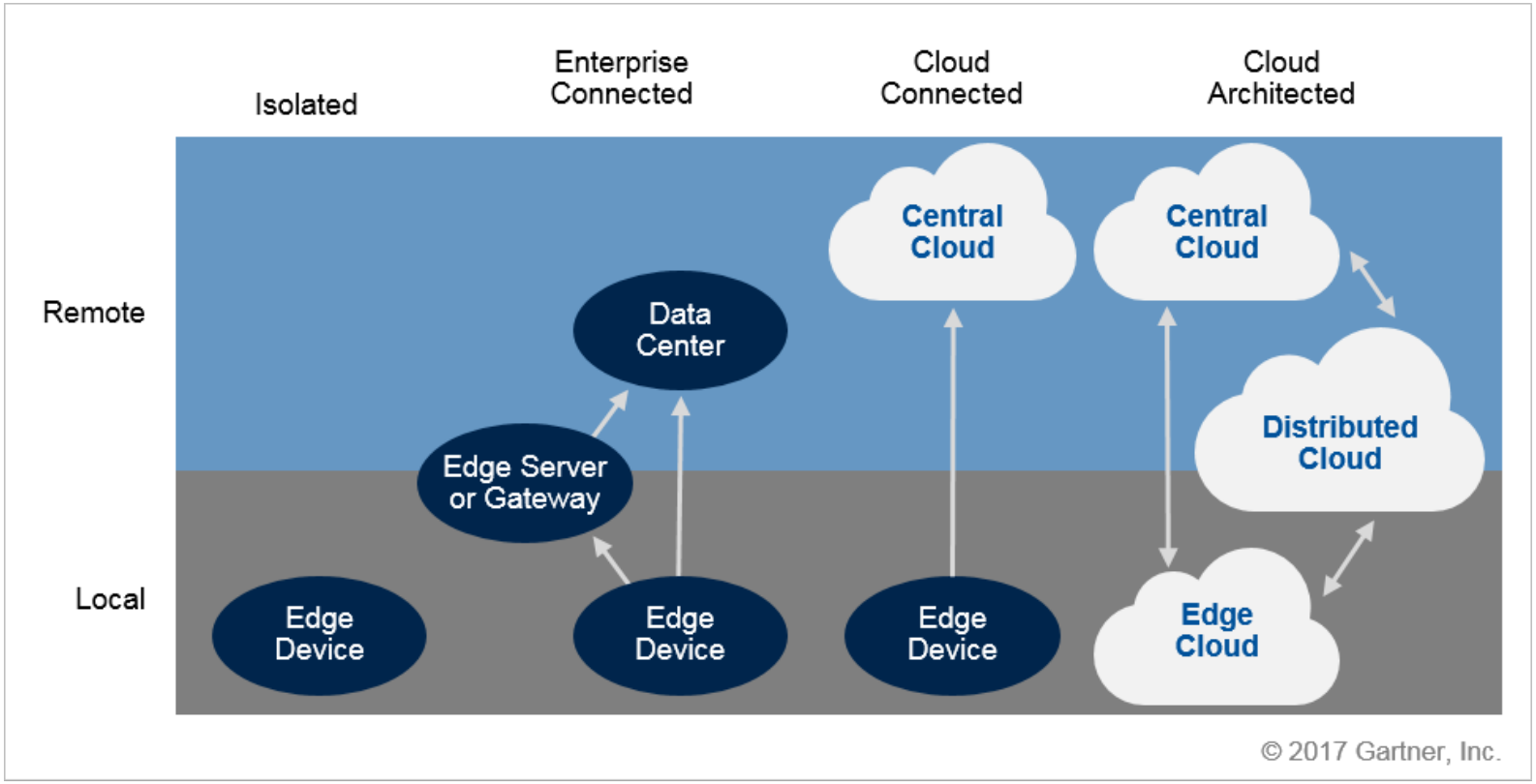
Luckow et al, Applications, Architectures and Best Practices for AI Vision Systems in Automotive Manufacturing, 2017.

AI Applications is more than Model Training



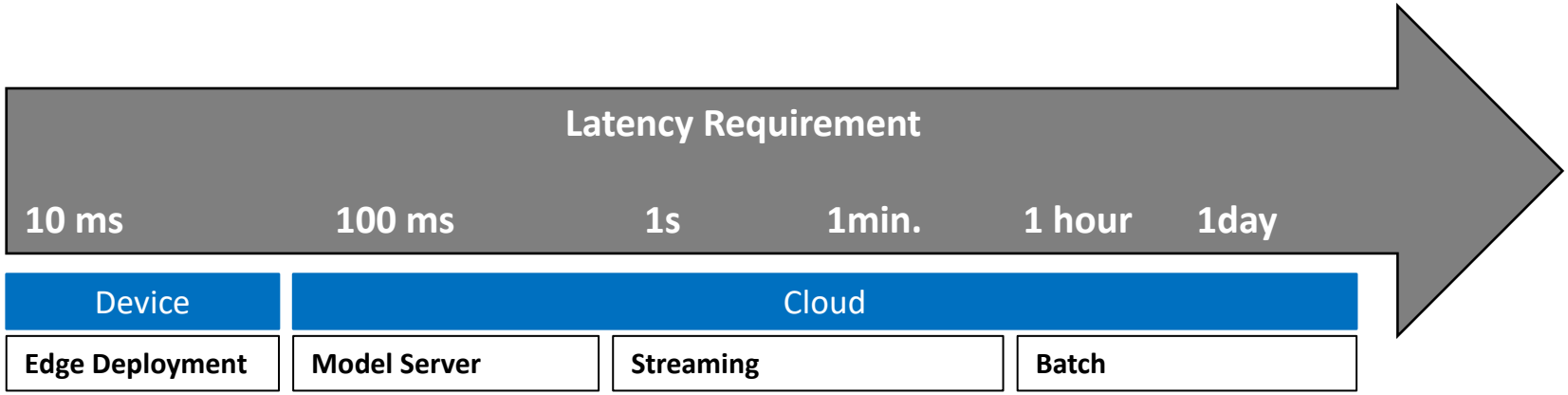
Luckow et al, Applications, Architectures and Best Practices for AI Vision Systems in Automotive Manufacturing, 2017.

Model Deployment



Source: Gartner (October 2017)

Deployment Options



Considerations:

- Network latencies and bandwidths
- Computational and data complexity of model

Adapted from:
https://cdn.oreillystatic.com/en/assets/1/event/269/Best%20practices%20for%20productionizing%20Apache%20Spark%20MLlib%20models%20Presentation.pdf

- ResNet-50 requires 7.72 Billion operations to process one 225x225 image
- How to scale this to realtime? For example, perception for robotics or autonomous driving?
  - 230 Gops for 30fps
  - 9.4 Tops for 12 HD cameras
  - 3 deep neural networks per inference -> 338Tops

Model Deployment Options

- Edge Inference:
  - Apple CoreML, https://developer.apple.com/documentation/coreml, 2017
  - Nvidia, TensorRT, https://developer.nvidia.com/tensorrt, 2017
- Write Model Results to Database
- Cloud/Backend
  - Crankshaw et al., Clipper: A Low-Latency Online Prediction Serving System, NSDI, 2017
  - Tensorflow Serve, https://www.tensorflow.org/serving/, 2017
- Streaming

Edge versus Cloud.

	Edge	Cloud
Advantages	Low latencies, offline, real-time autonomous capabilities, privacy	Centralization simplifies architectures,
Disadvantages	Higher complexity as models and data need to be managed across devices and backend, platform specific	Higher latencies, connectivity required, costs for compute resources & network,

Table 4: AI Deployment Trade-Offs

Luckow et al, Applications, Architectures and Best Practices for AI Vision Systems in Automotive Manufacturing, 2017.

Edge Performance.

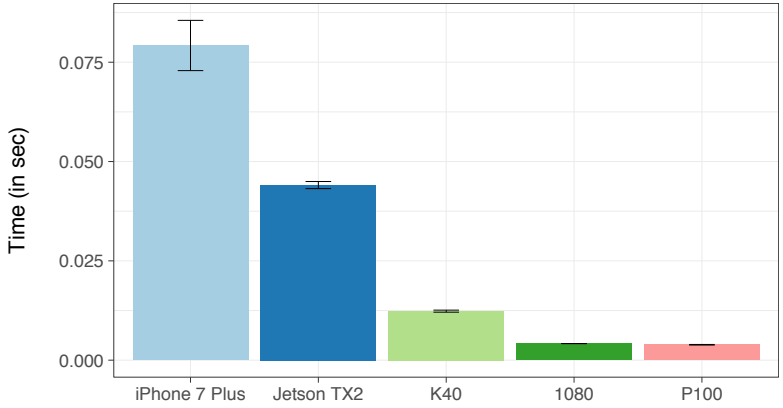


Figure 4: Tiny YOLO model inference times

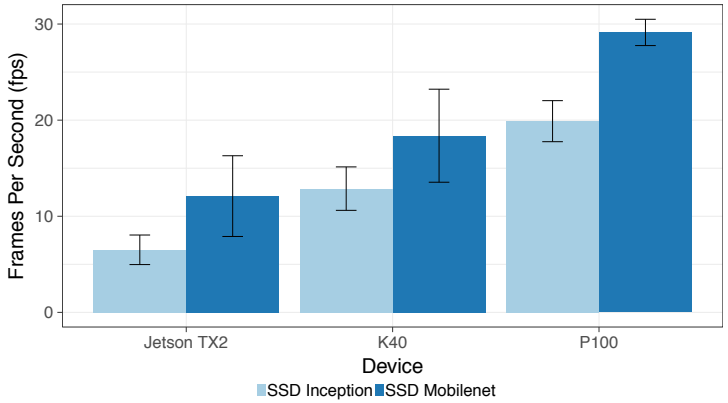
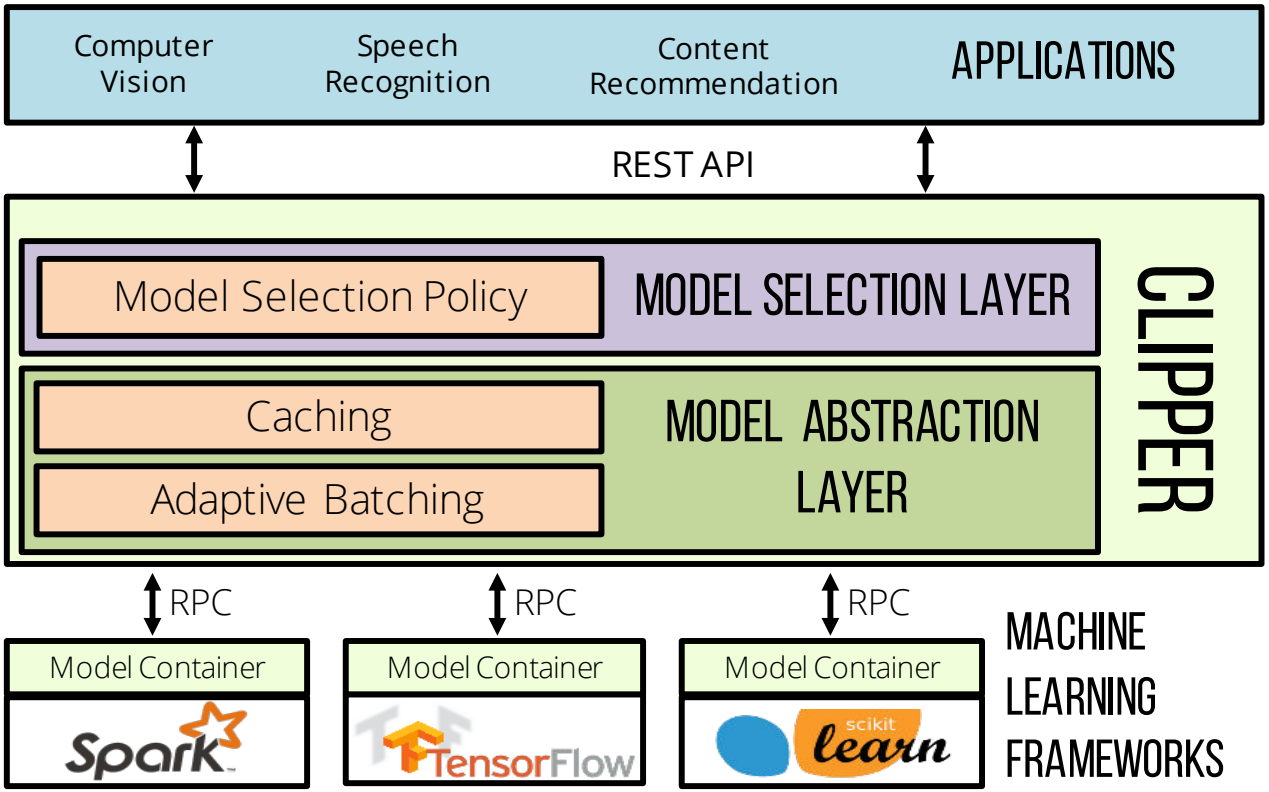
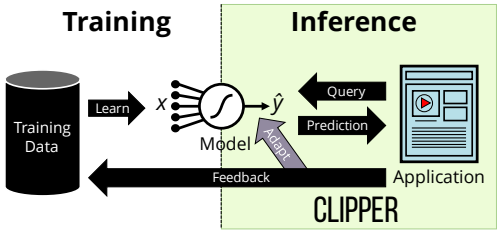


Figure 7: Yard Management SSD Inference Performance

Prediction Services

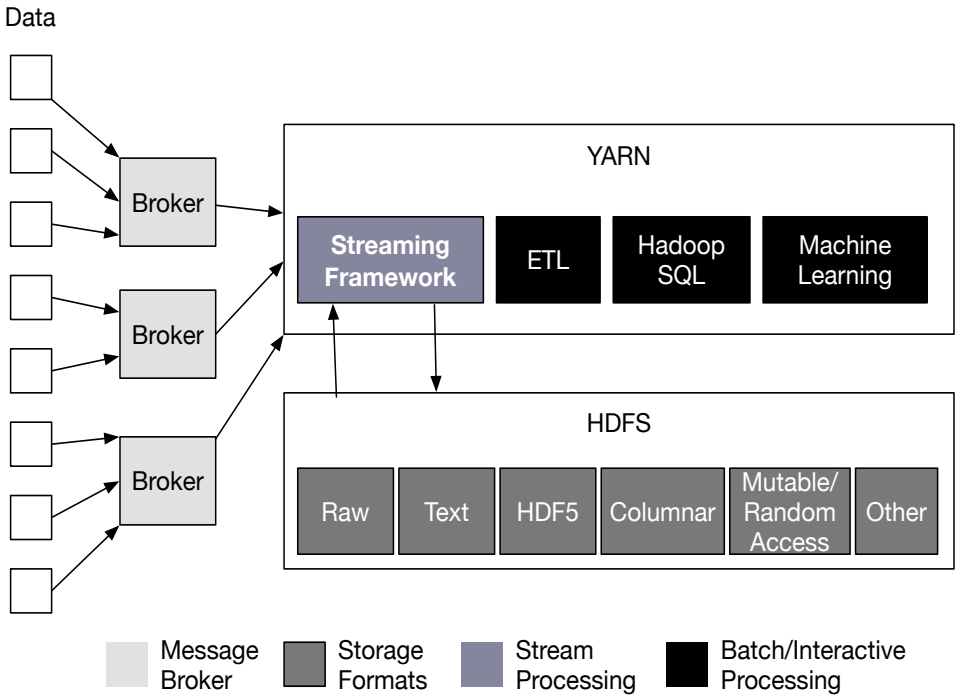




Stream Processing

- Stream processing, is a computing technique in which incoming data about what is happening (event data) is processed as it arrives to generate higher-level, more-useful, summary information (Gartner)
- **Streaming system:** a type of data processing engine that is designed with infinite datasets in mind [1].
  - **Bounded data:** a type of dataset that is finite in size.
  - **Unbounded data:** a type of dataset that is infinite in size (at least theoretically).
- **Database Table vs. Stream:**
  - **Table:** a holistic view of a dataset at a specific point in time. SQL systems have traditionally dealt in tables.
  - **Stream:** an element-by-element view of the evolution of a dataset over time.

Stream Processing Components



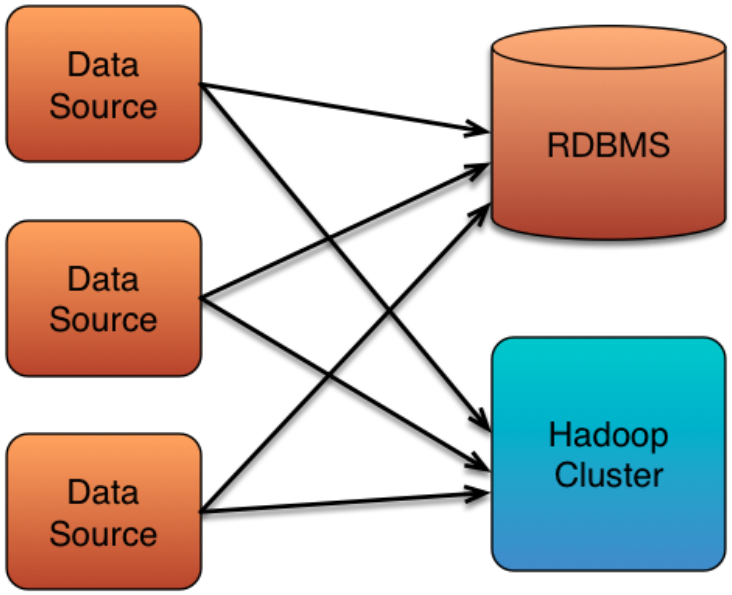
Message Broker:

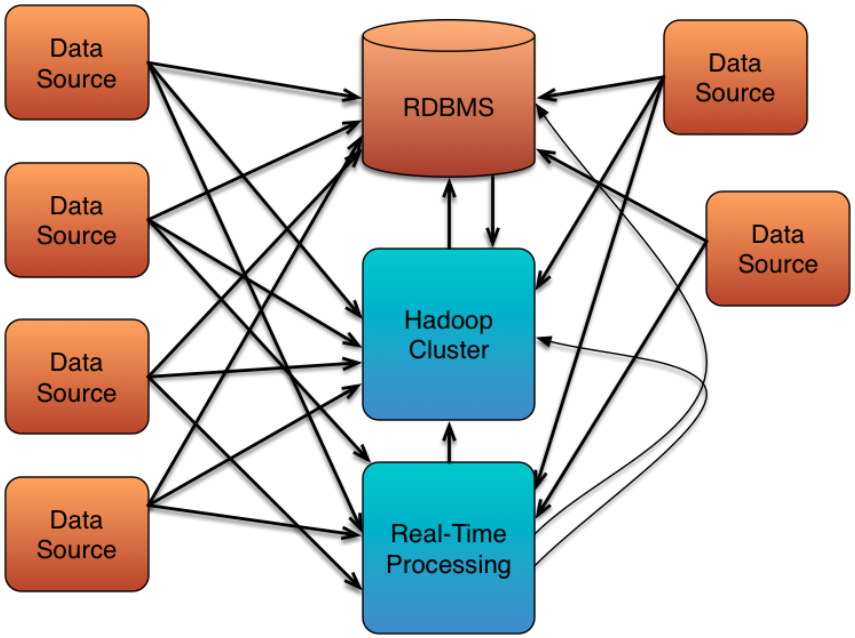
- Decouples data producer and consumer providing a reliable, durable data storage and transport.
- Durable, replay-able data source to streaming processing applications.
- Example: Kafka

Stream Processing Engines:

- Data processing engines optimized for unbounded data feeds
- In comparison to batch processing engines, it provides abstraction to manage windowing and completeness
- Examples: Flink, Spark-Streaming, Heron/Storm

Kafka Motivation



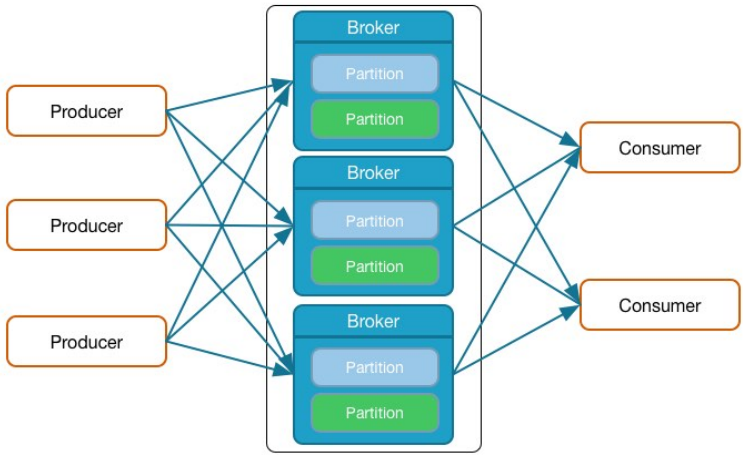


Stream processing with Kafka, Tim Berglund (Confluent), https://conferences.oreilly.com/strata/strata-ca/public/schedule/proceedings, 2018

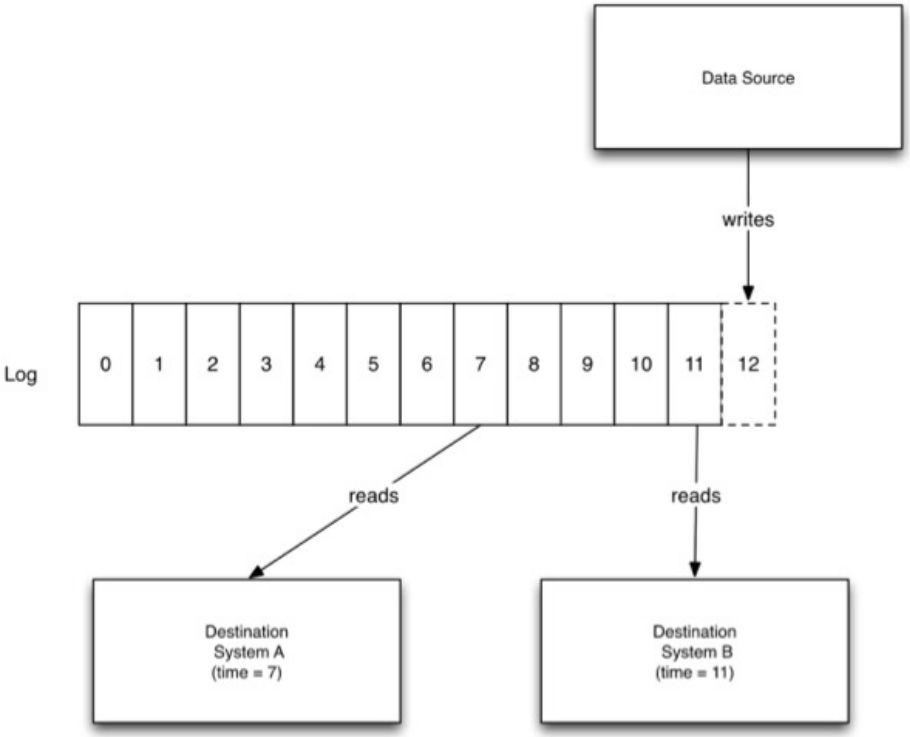
Kafka Architecture and Abstractions

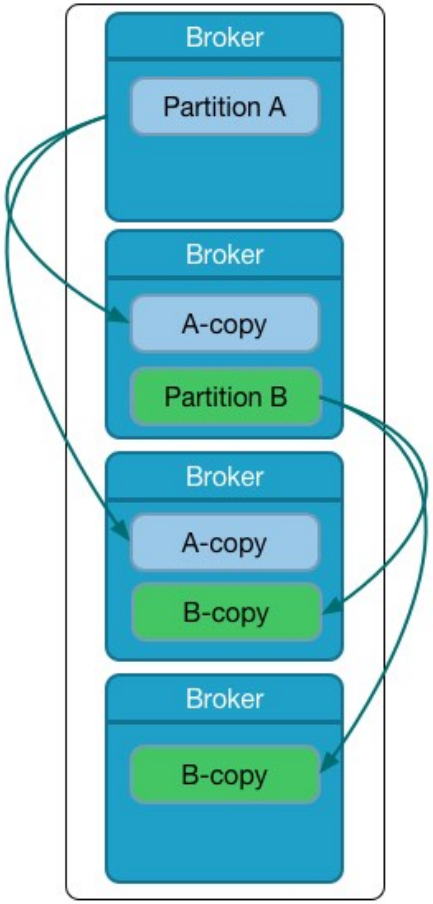
Producers send data to the Kafka cluster
Consumers read data from the Kafka cluster
Brokers are the main components of the Kafka cluster

Messages are grouped in so called topics and distributed in partitions across multiple brokers
Each Partition is stored on the Broker's disk as one or more log files
Each message in the log is identified by its *offset*

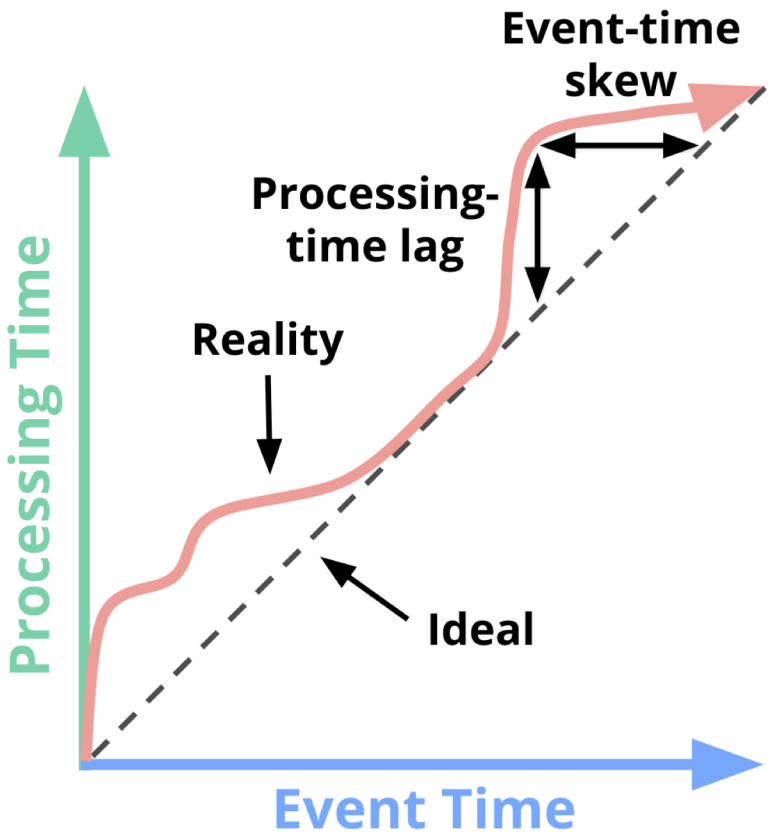


Message Storage: Logs



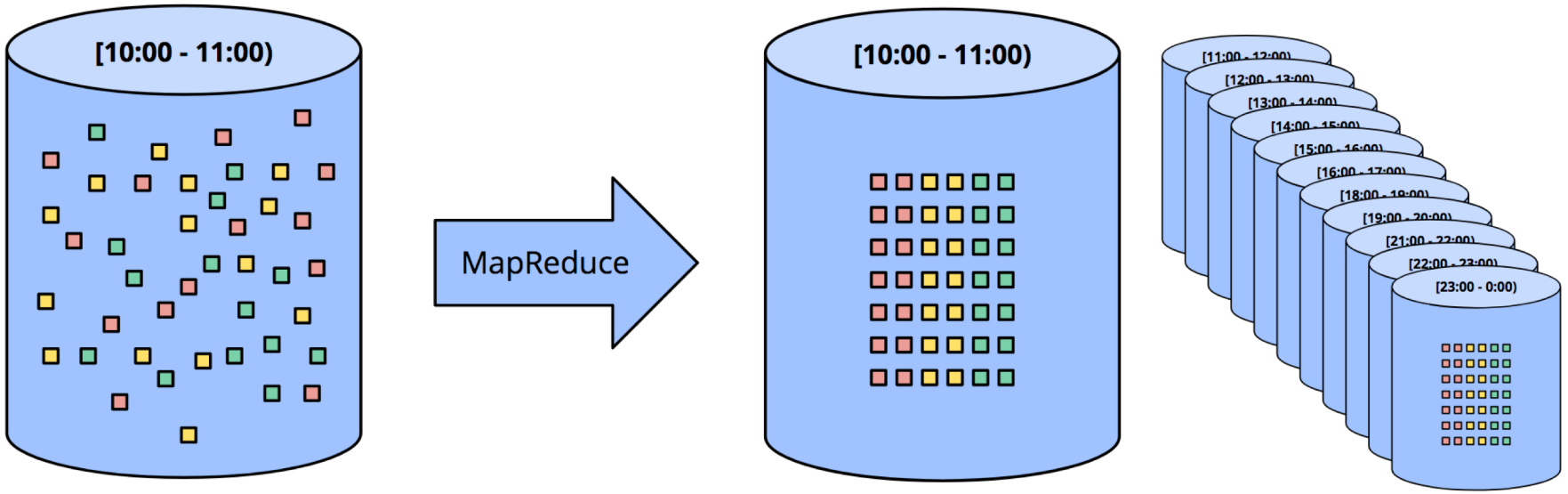


Stream processing with Kafka, Tim Berglund (Confluent), https://conferences.oreilly.com/strata/strata-ca/public/schedule/proceedings, 2018



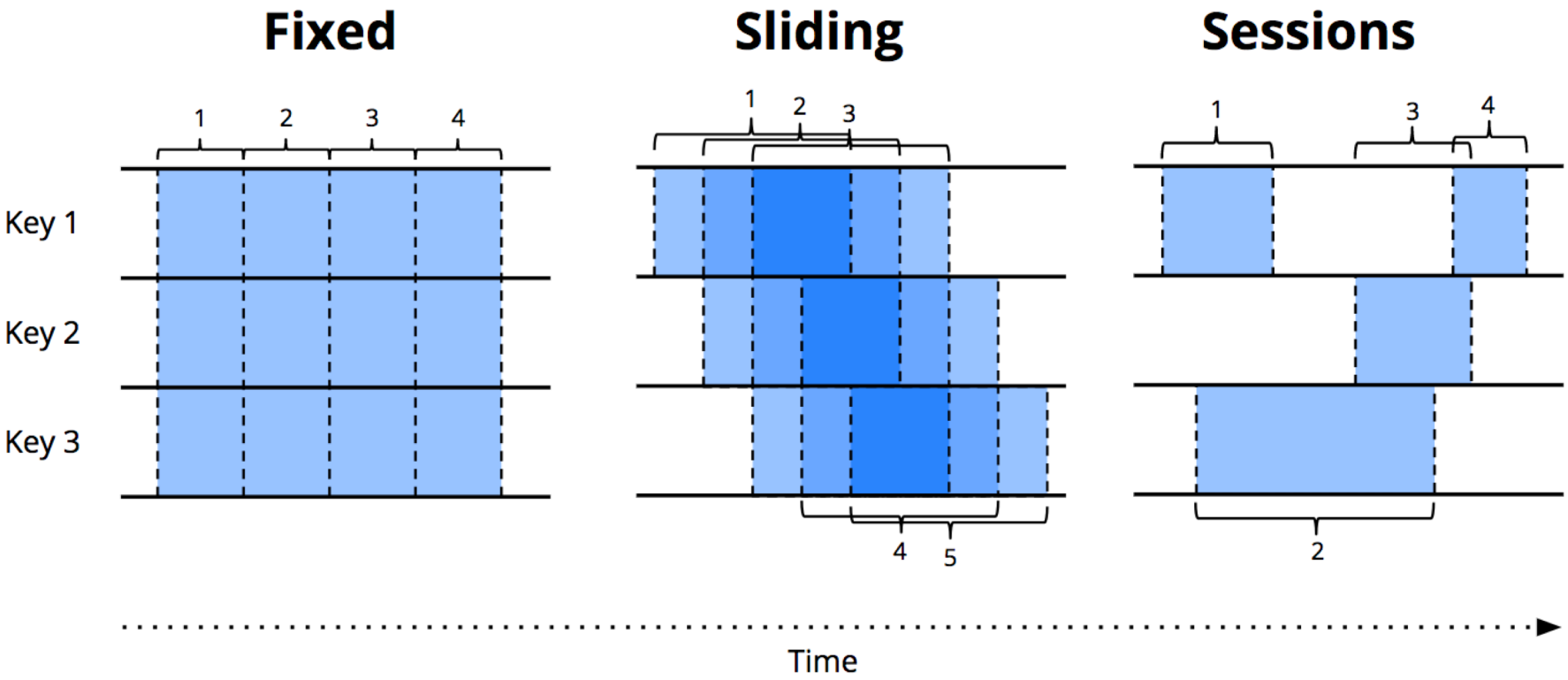
- **Event time**, which is the time at which events actually occurred.
- **Processing time**, which is the time at which events are observed in the system.

Mini-Batch Stream Processing



Tyler Akidau, Reuven Lax and Slava Chernyak, Streaming Systems, O'Reilly, 2018.

Stream Processing: Windows

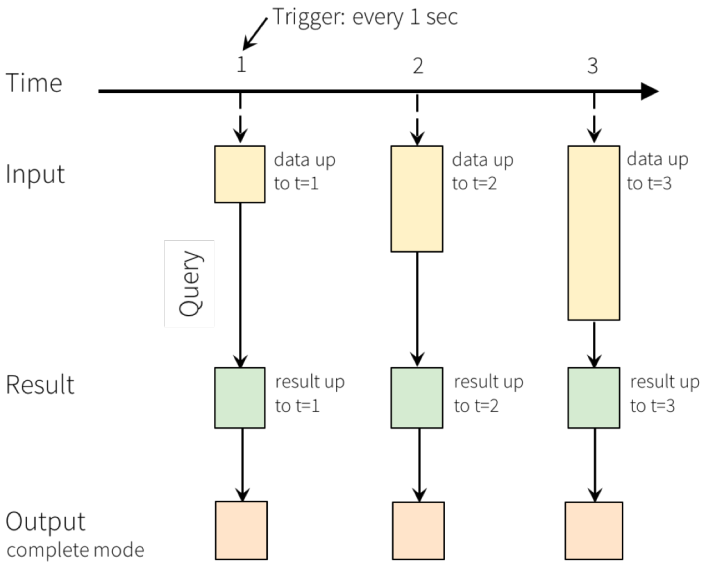


Tyler Akidau, Reuven Lax and Slava Chernyak, Streaming Systems, O'Reilly, 2018.

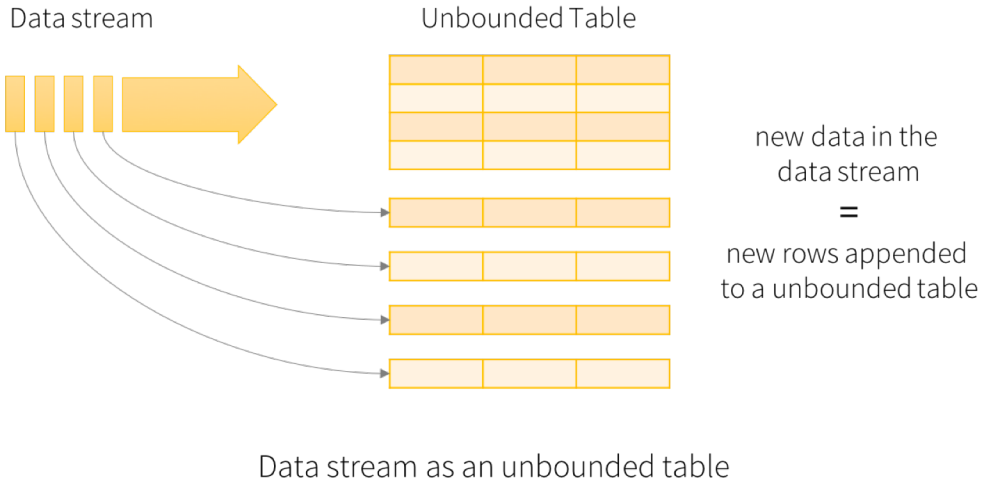
Spark Streaming

- Spark Streaming provides a high-level abstraction called *discretized stream* or *DStream*, which represents a continuous stream of data.
- DStreams can be created either from input data streams from sources such as Kafka, Flume, and Kinesis, or by applying high-level operations on other DStreams.
- Internally, a DStream is represented as a sequence of RDDs.

Spark Structured Streaming.



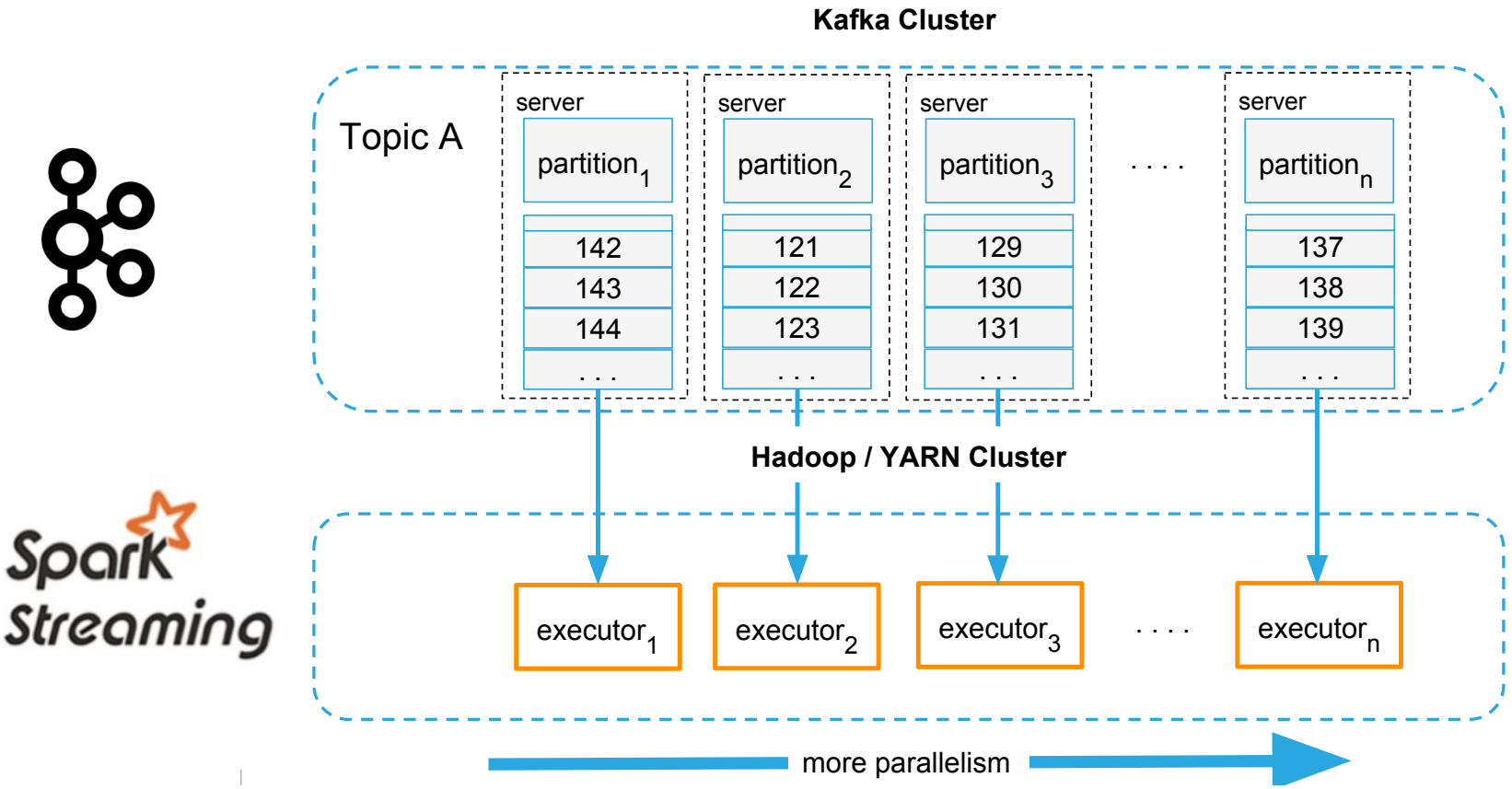
Programming Model for Structured Streaming



Data stream as an unbounded table

Spark Streaming 2.x introduced the Streaming Table Abstraction

Spark Streaming and Kafka



https://cdn.oreillystatic.com/en/assets/1/event/269/How%20to%20build%20leakproof%20stream%20processing%20pipelines%20with%20Apache%20Kafka%20and%20Apache%20Spark%E2%80%8B%20Presentation.pdf

Flink: Streaming Abstractions (1)

Layered abstractions to navigate simple to complex use cases

High-level Analytics API

SQL / Table API (dynamic tables)

```
SELECT room, TUMBLE_END(rowtime, INTERVAL '1' HOUR), AVG(temp)
FROM sensors
GROUP BY TUMBLE(rowtime, INTERVAL '1' HOUR), room
```

Stream- & Batch Data Processing

DataStream API (streams, windows)

```
val stats = stream
    .keyBy("sensor")
    .timeWindow(Time.seconds(5))
    .sum((a, b) -> a.add(b))
```

Stateful Event-Driven Applications

Process Function (events, state, time)

```
def processElement(event: MyEvent, ctx: Context, out: Collector[Result]) = {
  // work with event and state
  (event, state.value) match { … }

  out.collect(…) // emit events
  state.update(…) // modify state

  // schedule a timer callback
  ctx.timerService.registerEventTimeTimer(event.timestamp + 500)
}
```

5

Spark vs. Flink

- Spark Mini-batch Model:
  - Higher latencies
  - Higher throughput
- Flink Continuous Mode:
  - Long-running pre-scheduled tasks
  - Lower Latencies
  - Higher Throughput
- Spark Continuous Mode in development

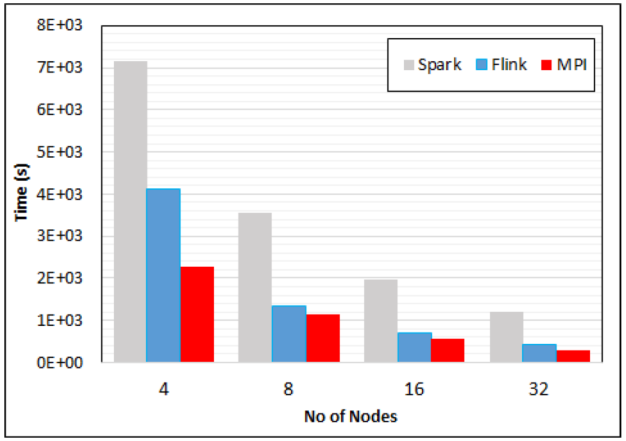


Fig. 11. K-Means execution time on varying number of nodes with 20 processes in each node with 10 million points and 16000 centroids. Each point has 100 attributes.

Fox et al, Anatomy of machine learning algorithm implementations in MPI, Spark, and Flink, https://www.researchgate.net/publication/318436923_Anatomy_of_machine_learning_algorithm_implementations_in_MPI_Spark_and_Flink 2017.

- Kafka, https://kafka.apache.org/
- Twitter Storm, http://storm-project.net/
- Zaharia et al., Discretized Streams: Fault-Tolerant Streaming Computation at Scale, SOSP, 2013
- Carbone et al., Apache Flink: Stream and Batch Processing in a Single Engine, IEEE Engineering Bulletins, 2015

Summary

Key Issues

- **Infrastructure:**
  - HDFS
  - YARN
  - Architecture, Understanding of important trade-offs: Sequential Read Performance for HDFS
- **Map Reduce:**
  - Abstraction, Components, Implementations
  - Important Patterns: Summarization, Joins
  - Performance: Combiner, Shuffle Optimizations

- **Spark**
  - Short-comings of Hadoop
  - Advantages of Spark
  - GroupByKey vs. combineByKey
  - ML with Spark

- **Data Science Process and Methods**
  - Regression vs. Classification
  - Model Validation: Precision, Recall, Accuracy

Key Issues

- **Scalable Machine Learning:**
  - Scalability Metrics:
    - Speedup
    - Efficiency
  - Model vs. Data Parallelism
  - Parallel Algorithms
    - K-Means
    - Parallel Deep Learning

Key Issues

- **Deployment**:
  - Batch
  - Database
  - Streaming
  - Model Server
  - Edge
- **Streaming:**
  - Continuous vs. Mini-Batch
  - Architectures and Tools

Contact:
Andre Luckow
luckow@nm.ifi.lmu.de