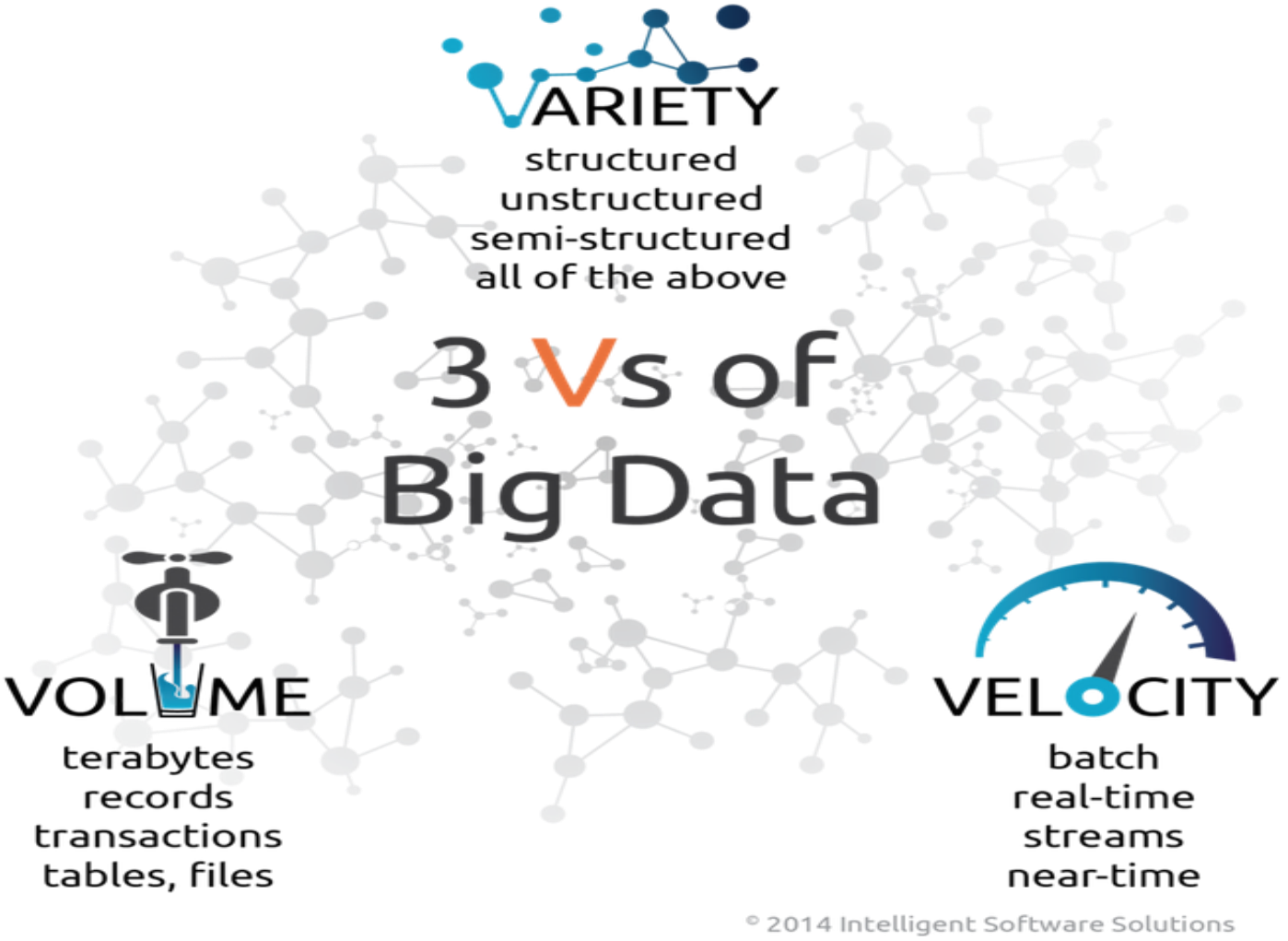
ACCELERATING MACHINE LEARNING SOFTWARE ON INTEL® ARCHITECTURE (IA)

Agenda

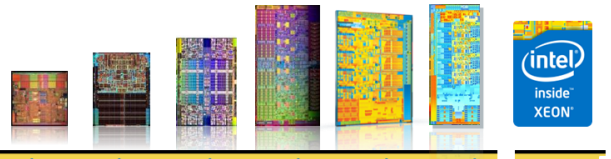
- Challenges of Bigdata
- Accelerating Machine Learning on Intel® IA
- Intel portfolio for AI
- Intel® Math Kernel Library (Intel® MKL)
- Intel® Data Analytics Acceleration Library (Intel® DAAL)

Big Data V's



Machine Learning Software Challenges

- Machine learning open source frameworks and libraries are often not well-optimized for newer Intel-based systems
- Frameworks can be difficult to configure and use
- Need to target heterogeneous hardware from model training in the datacenter to deployment on endpoint systems
- Big data needs high performance computing
- Many big data applications leave performance at the table – Not optimized for underlying hardware



	Intel® Xeon® Processor 64-bit	Intel® Xeon® Processor 5100 series	Intel® Xeon® Processor 5500 series	Intel® Xeon® Processor 5600 series	Intel® Xeon® Processor E5-2600 v2 series	Intel® Xeon® Processor E5-2600 v3 series	Future Intel® Xeon® Processor[1]	Intel® Xeon Phi™ x100 Coprocessor	Intel® Xeon Phi™ x200 Processor & Coprocessor
Up to Core(s)	1	2	4	6	12	18	*Tbd*	57-61	TBD
Up to Threads	2	2	8	12	24	36	*tbd*	228-244	TBD
SIMD Width	128	128	128	128	256	256	512	512	512
Vector ISA	Intel® SSE3	Intel® SSE3	Intel® SSE4.2	Intel® AVX	Intel® AVX	Intel® AVX2	Intel® AVX-512	IMCI 512	Intel® AVX-512

Challenges faced by developers

1. Code refactoring-Advantage of modern vector instructions
2. Core utilization- Maximum efficiency within layers and across layers
3. Availability of data- Balanced prefetching, cache blocking
4. Data format- Contiguous memory, conversions, format for temporal and spatial locality

INTEL® NERVANA™ PORTFOLIO

EXPERIENCES

PLATFORMS

Intel® Nervana™ Cloud & Appliance

Intel® Nervana™ DL Studio

Intel® Computer Vision SDK

Movidius Fathom

intel Saffron™

FRAMEWORKS

Apache Spark MLlib BigDL

neon

TensorFlow

theano

Caffe

mxnet

torch

Caffe2

Chainer

LIBRARIES

python Intel Python Distribution

Intel® Data Analytics Acceleration Library (DAAL)

Intel® Nervana™ Graph*

Intel® Math Kernel Library (MKL, MKL-DNN)

HARDWARE

Compute

Memory & Storage

Networking

INSIDE AI

*Future

Hardware Optimized Libraries and Frameworks

Scaling

- Improve load balancing
- Reduce synchronization events, all-to-all comms

Utilize all the cores

- OpenMP, MPI
- Reduce synchronization events, serial code
- Improve load balancing

Vectorize/SIMD

- Unit strided access per SIMD lane
- High vector efficiency
- Data alignment

Efficient memory/cache use

- Blocking
- Data reuse
- Prefetching
- Memory allocation

Important to use optimized software frameworks and libraries for best AI workload performance

Intel Performance Libraries – way to enable Big Data on Intel® IA

https://cartaworldwide.com/news-insights/big-data-and-mobile-payments-the-game-changer/

Intel® Math Kernel Library
Scientific, engineering, financial, machine learning and energy applications

Intel® Integrated Performance Primitives
Signal/image processing, compression, cryptography

Intel® distribution for Python

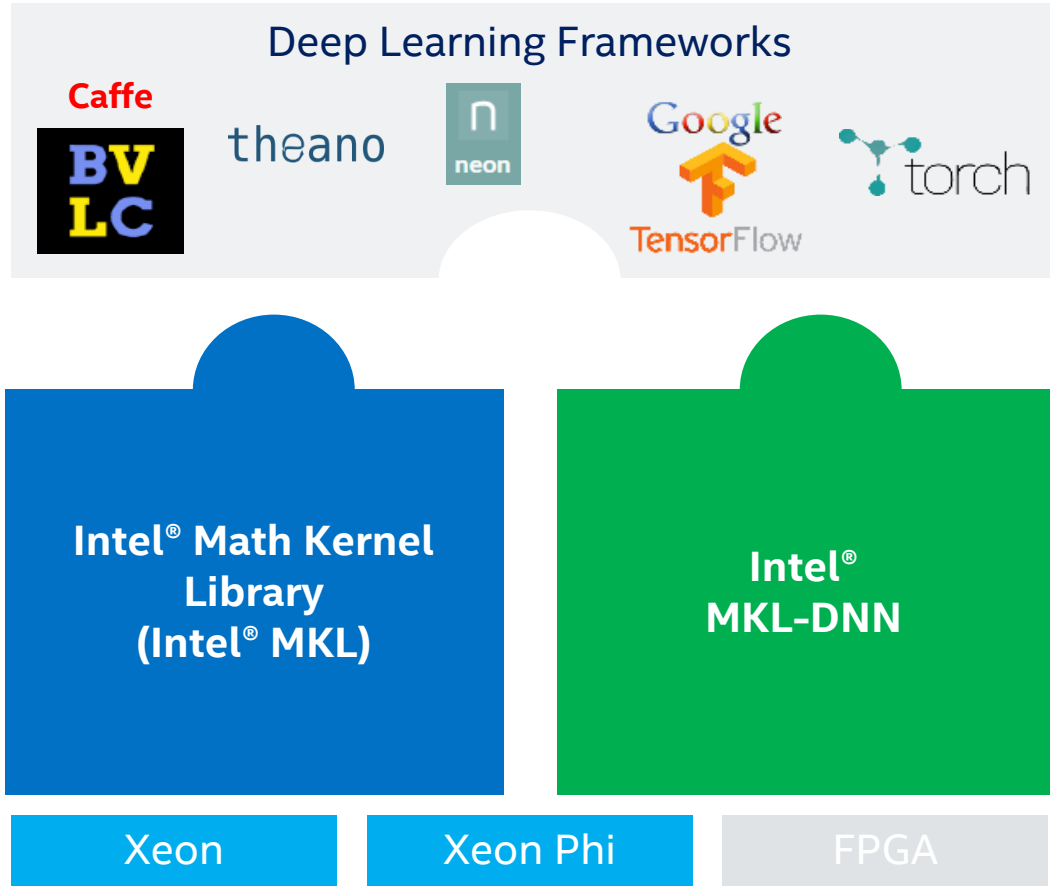
Intel® Data Analytics Acceleration Library
Building blocks covering all stages of the data analysis

OVERVIEW OF INTEL MATH KERNEL LIBRARY

Components of Intel MKL 2017

Linear Algebra	Fast Fourier Transforms	Vector Math	Summary Statistics	And More…	Deep Neural Networks
• BLAS • LAPACK • ScaLAPACK • Sparse BLAS • Sparse Solvers • Iterative • PARDISO* • Cluster Sparse Solver	• Multidimensional • FFTW interfaces • Cluster FFT	• Trigonometric • Hyperbolic • Exponential • Log • Power • Root • Vector RNGs	• Kurtosis • Variation coefficient • Order statistics • Min/max • Variance-covariance	• Splines • Interpolation • Trust Region • Fast Poisson Solver	• Convolution • Pooling • Normalization • ReLU • Softmax

Intel® Math Kernel Library and Intel® MKL-DNN for Deep Learning Framework Optimization



Intel® MKL	Intel® MKL-DNN
DNN primitives + wide variety of other math functions	DNN primitives
C DNN APIs	C/C++ DNN APIs
Binary distribution	Open source DNN code*
Free community license. Premium support available as part of Parallel Studio XE	Apache 2.0 license
Broad usage DNN primitives; not specific to individual frameworks	Multiple variants of DNN primitives as required for framework integrations
Quarterly update releases	Rapid development ahead of Intel MKL releases

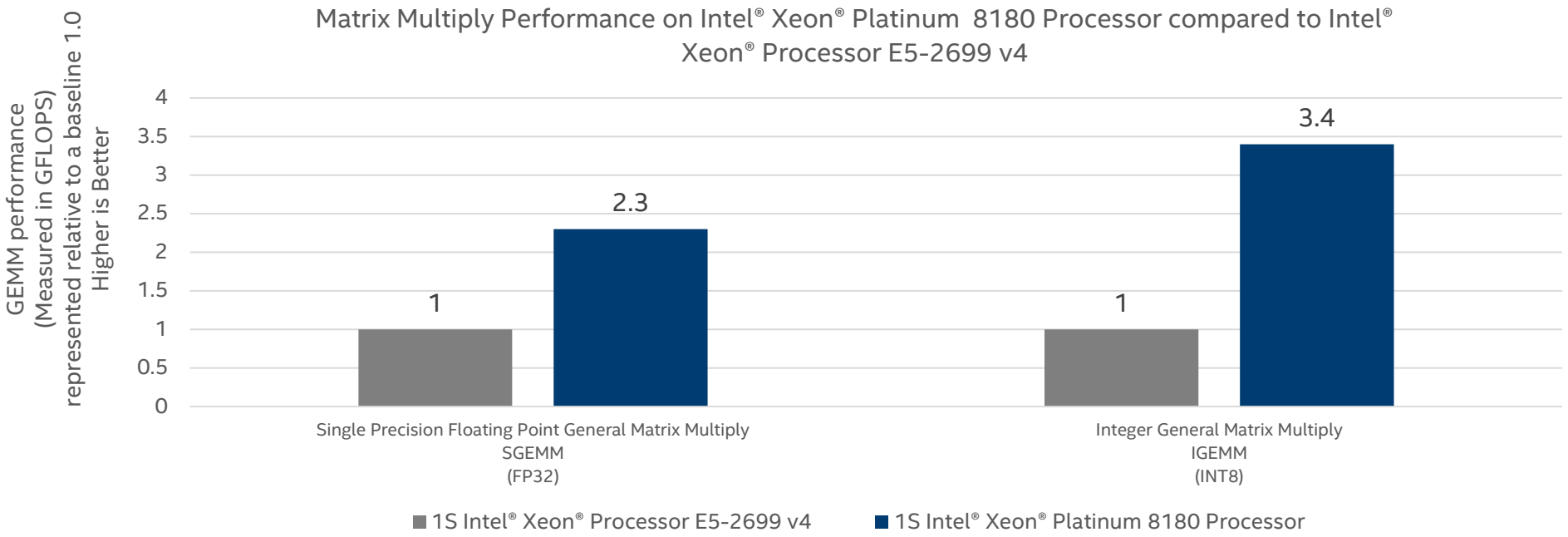
* GEMM matrix multiply building blocks are binary

Deep Neural Network (DNN) Primitives in Intel MKL 2017

Operations	Algorithms
Activation	ReLU
Normalization	batch, local response
Pooling	max, min, average
Convolutional	fully connected, direct 3D batched convolution
Inner product	forward/backward propagation of inner product computation
Data manipulation	layout conversion, split, concat, sum, scale

Up to 3.4x Integer Matrix Multiply Performance

on Intel® Xeon® Platinum 8180 Processor



8bit IGEMM will be available in Intel® Math Kernel Library (Intel® MKL) 2018 Gold to be released by end of Q3 2017

Enhanced matrix multiply performance on Intel® Xeon® Scalable Processor

Configuration Details in the backup slide

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit: http://www.intel.com/performance Source: Intel measured as of June 2017 Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

What is Intel® Optimized Caffe?

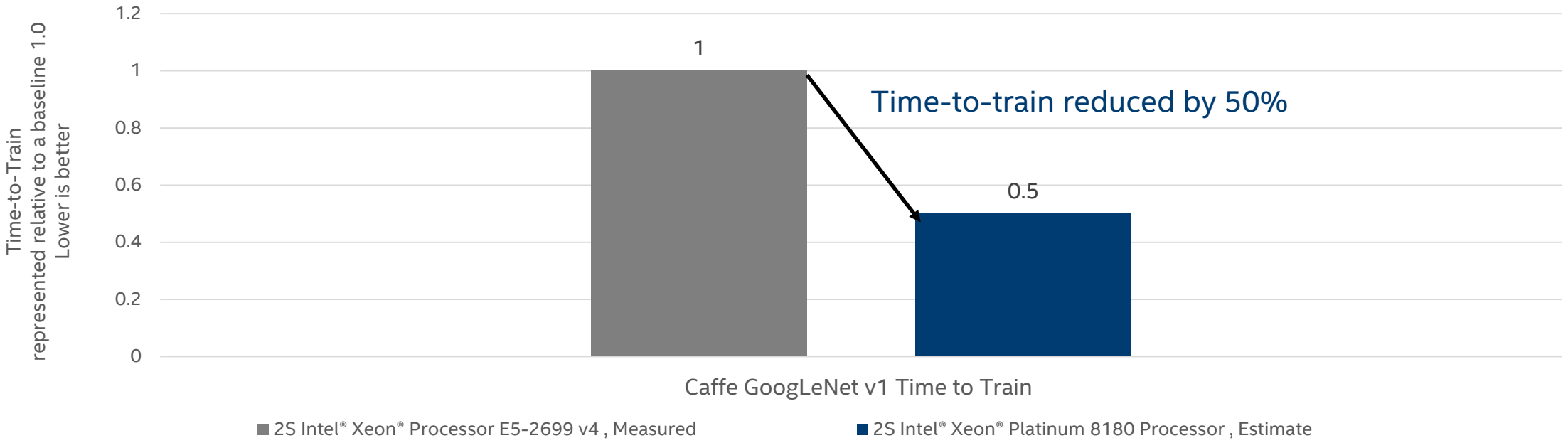
- Caffe- Popular deep learning framework for image classification
- Integrated with Intel® MKL 2017 Deep Learning primitives (BLAS :=mkl)
- Optimized for Intel® Advanced Vector Extensions Intel® AVX2 includes Intel® AVX-512 instructions
- Code Vectorization / Parallelization
- CPU/System specific optimizations
- Better cache usage
- Performance scaling with Multiple Nodes
- Support for distributed training across various nodes
- BVLC Caffe : http://caffe.berkeleyvision.org/
- Intel® Caffe Git : https://github.com/intel/caffe

Up to 2x Faster Training Performance
on Intel® Xeon® Platinum 8180 Processor

Compute

Bandwidth

SW Optimizations



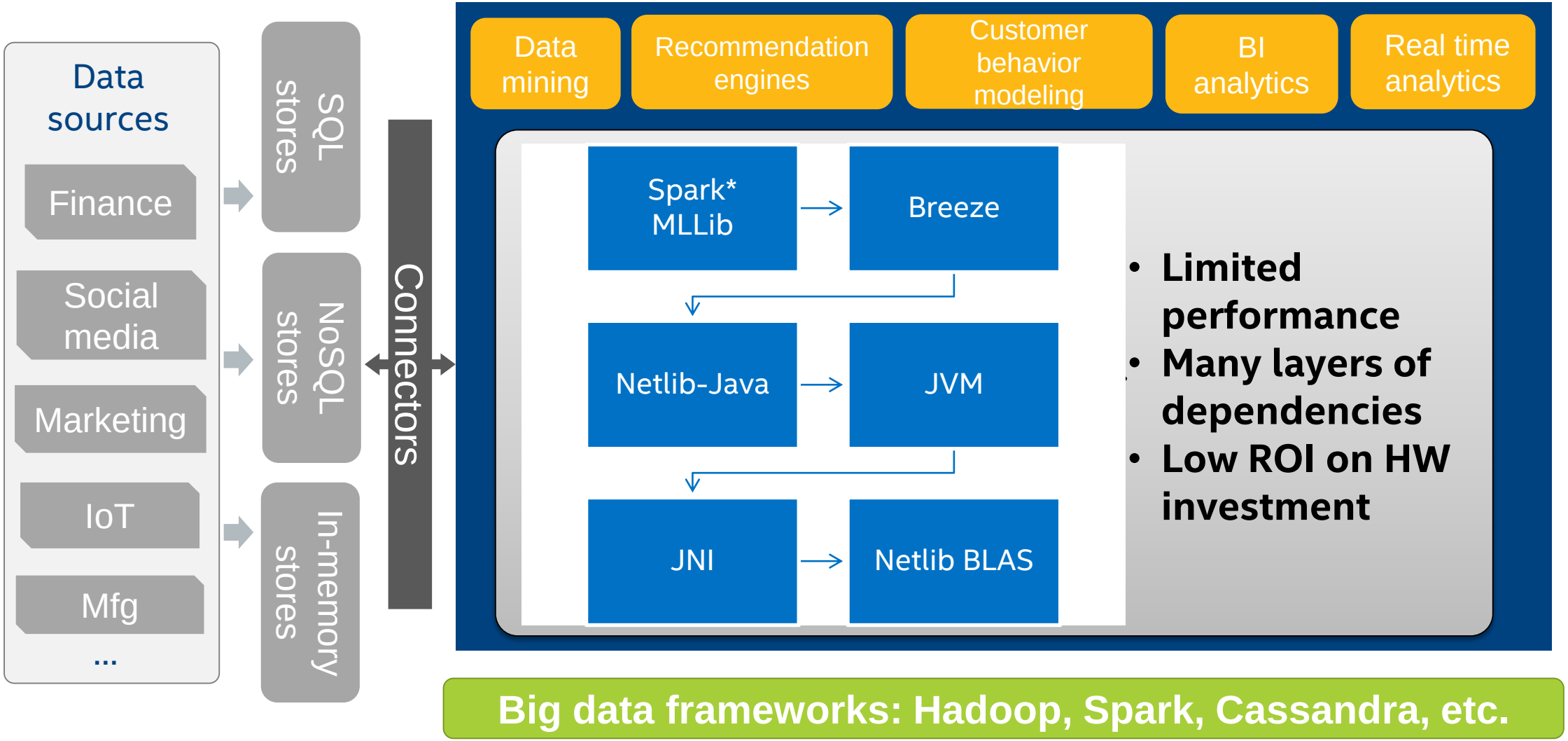
Intel® Xeon® Platinum Processor delivers up to 2x faster training compared to previous generation

TRAINING batch size 96 Configuration Details on Slide: 25, 26, 30
Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products. For more complete information visit: http://www.intel.com/performance Source: Intel measured as of June 2017 Optimization Notice: Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

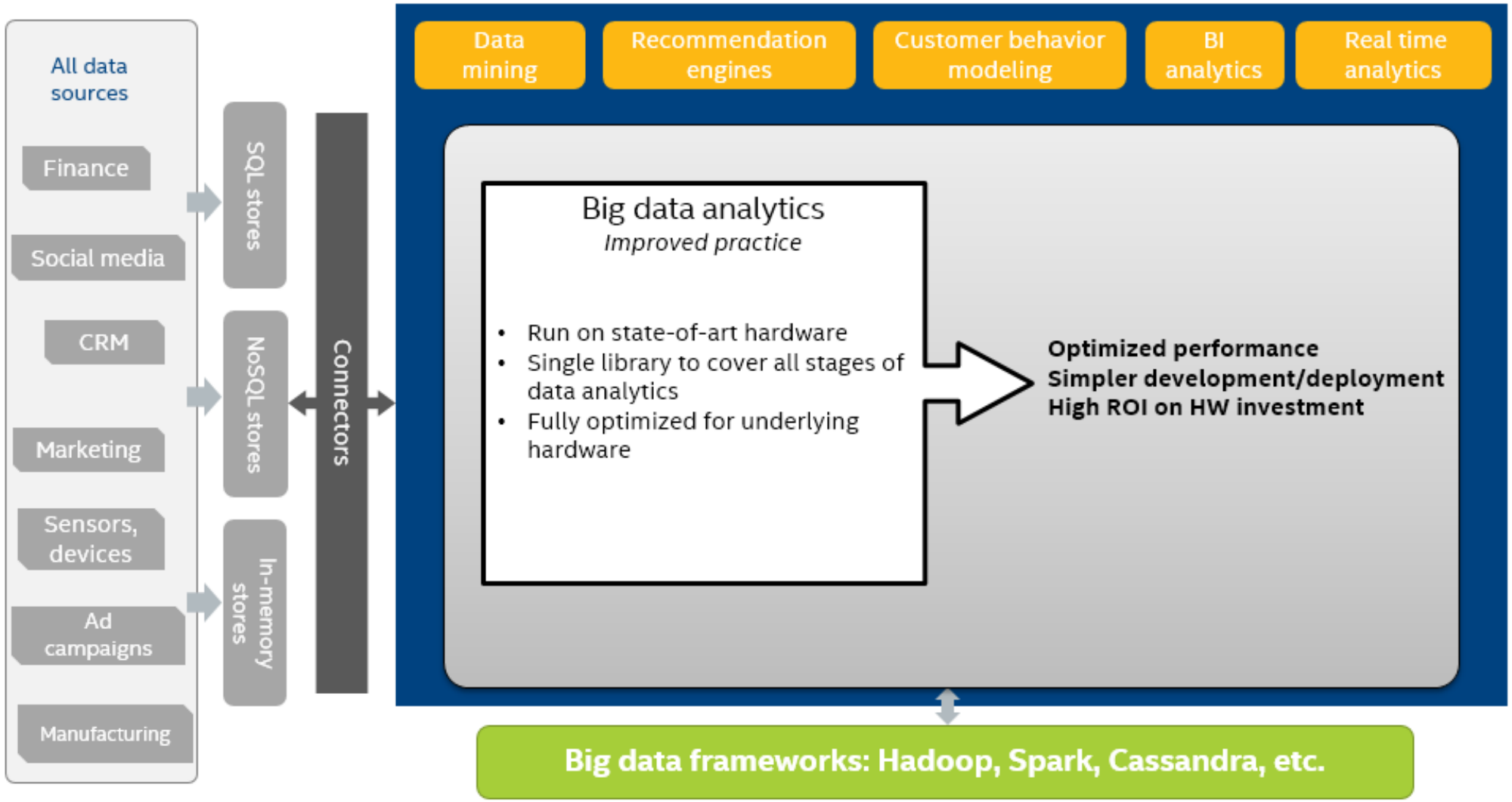
Optimization Notice

OVERVIEW OF INTEL® DATA ANALYTICS ACCELERATION LIBRARY

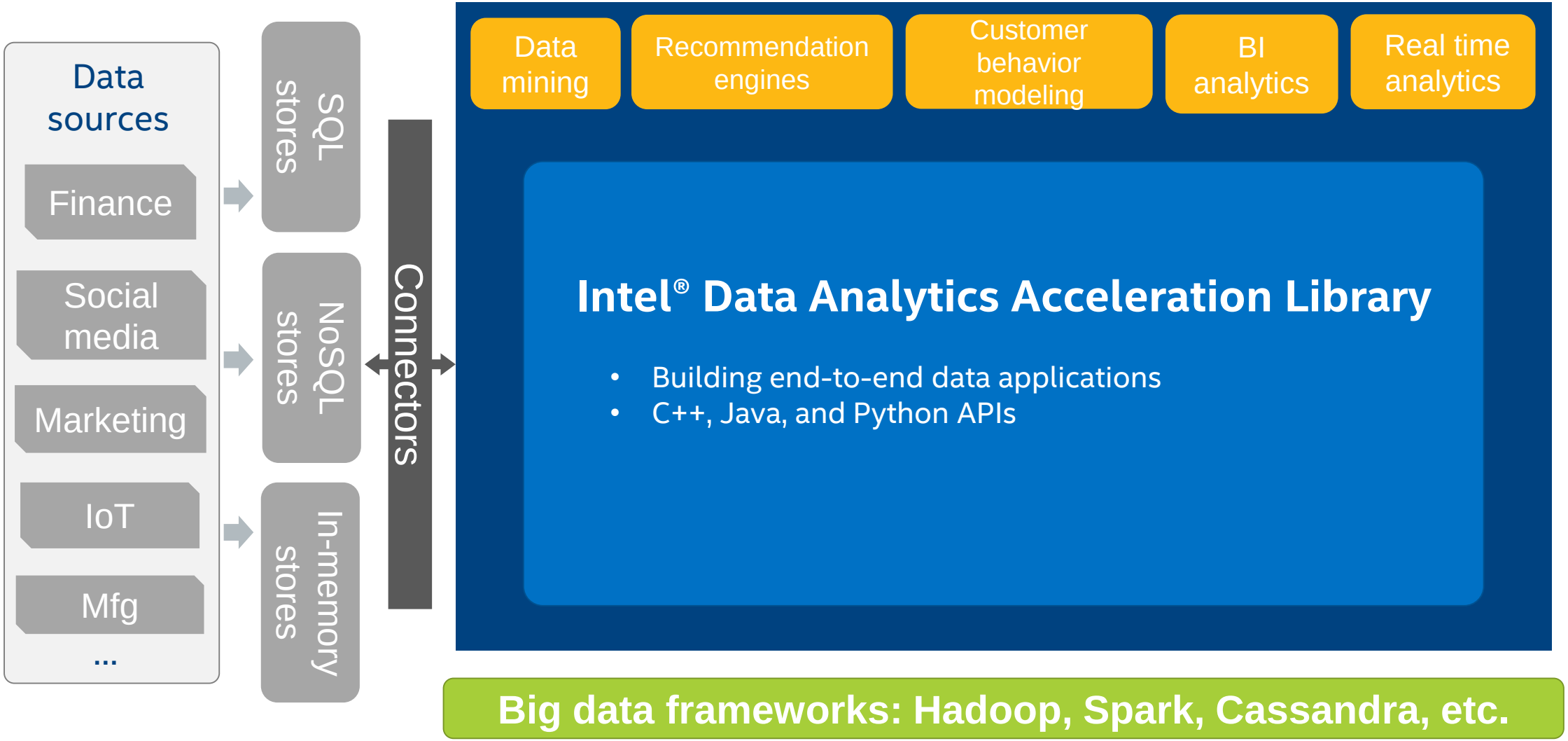
Problem Statement



Desired Solution



Where Intel DAAL Fits?

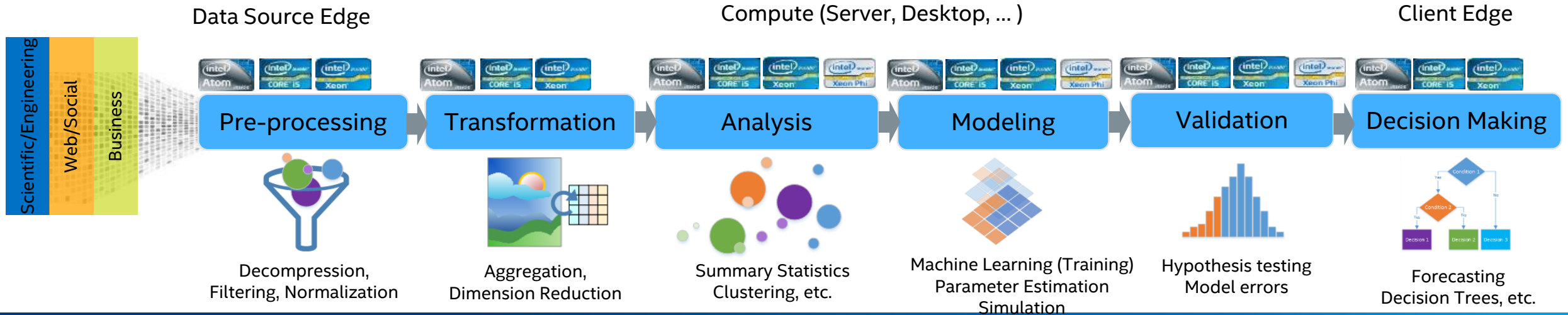


Ideas Behind Intel® DAAL: Heterogeneous Analytics

Data is different, data analytics pipeline is the same

Data transfer between devices is costly, protocols are different

- Need data analysis proximity to Data Source
- Need data analysis proximity to Client
- Data Source device ≠ Client device
- Requires abstraction from communication protocols



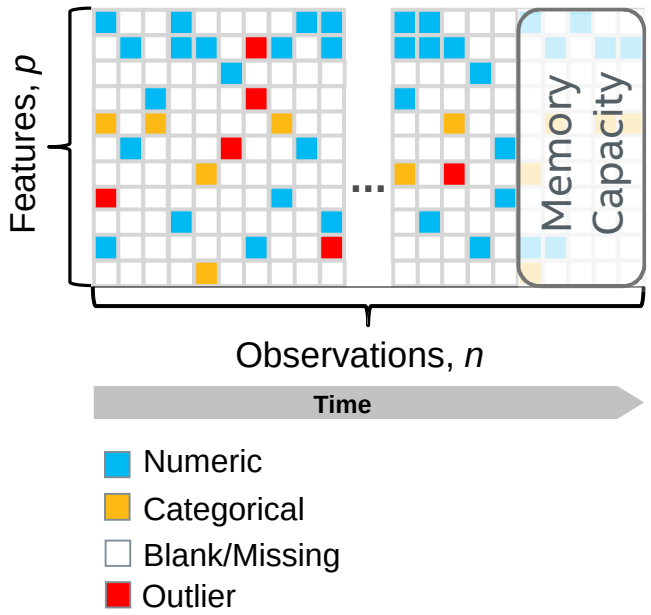
Intel® Data Analytics Acceleration Library (Intel® DAAL)

An Intel-optimized library that provides building blocks for all data analytics stages, from data preparation to data mining & machine learning

- Python, Java & C++ APIs
- Can be used with many platforms (Hadoop*, Spark*, R*, …) but not tied to any of them
- Flexible interface to connect to different data sources (CSV, SQL, HDFS, …)
- Windows*, Linux*, and OS X*

- Developed by same team as the industry-leading Intel® Math Kernel Library
- Open source, Free community-supported and commercial premium-supported options
- Also included in Parallel Studio XE suites

Ideas Behind Intel® DAAL: Effective Data Management, Streaming and Distributed Processing



Big Data Attributes	Computational Solution
Distributed across different devices	•Distributed processing with communication-avoiding algorithms
Huge data size not fitting into device memory	•Distributed processing •Streaming algorithms
Data coming in time	•Data buffering & asynchronous computing •Streaming algorithms
Non-homogeneous data	•Categorical→Numeric (counters, histograms, etc) •Homogeneous numeric data kernels • Conversions, Indexing, Repacking
Sparse/Missing/Noisy data	•Sparse data algorithms •Recovery methods (bootstrapping, outlier correction)

Why Intel® DAAL?

Automatic performance scaling

- Scale-up: from core to multicore to multi-socket
- Scale-out: from in-memory analysis to clusters to cloud

A rich set of analytics algorithms

- Widely applicable to most data mining and machine learning workloads

Leverages decades of R&D work in code optimization on IA

- By the same team behind Intel® Math Kernel Library (Intel® MKL)

Who should use Intel® DAAL?

Software developers

- Need optimized ML algorithms in their apps
- No resources/time/expertise to manually optimize themselves

Data Scientists

- Build and executes math models for domain specific knowledge discovery
- Need to speed up the performance critical parts of their models

Data Analytics ISVs

- Want competitive advantages by making their solutions run faster on IA

Big Data System Integrators

- Want to beef up their product portfolio by providing performance-enhanced alternatives to popular open-source analytics tools

Intel DAAL Components

Data Management

Interfaces for data representation and access. Connectors to a variety of data sources and data formats, such HDFS, SQL, CSV, and user-defined data source/format

Data Sources

Numeric Tables

Compression / Decompression

Serialization / Deserialization

Data Processing Algorithms

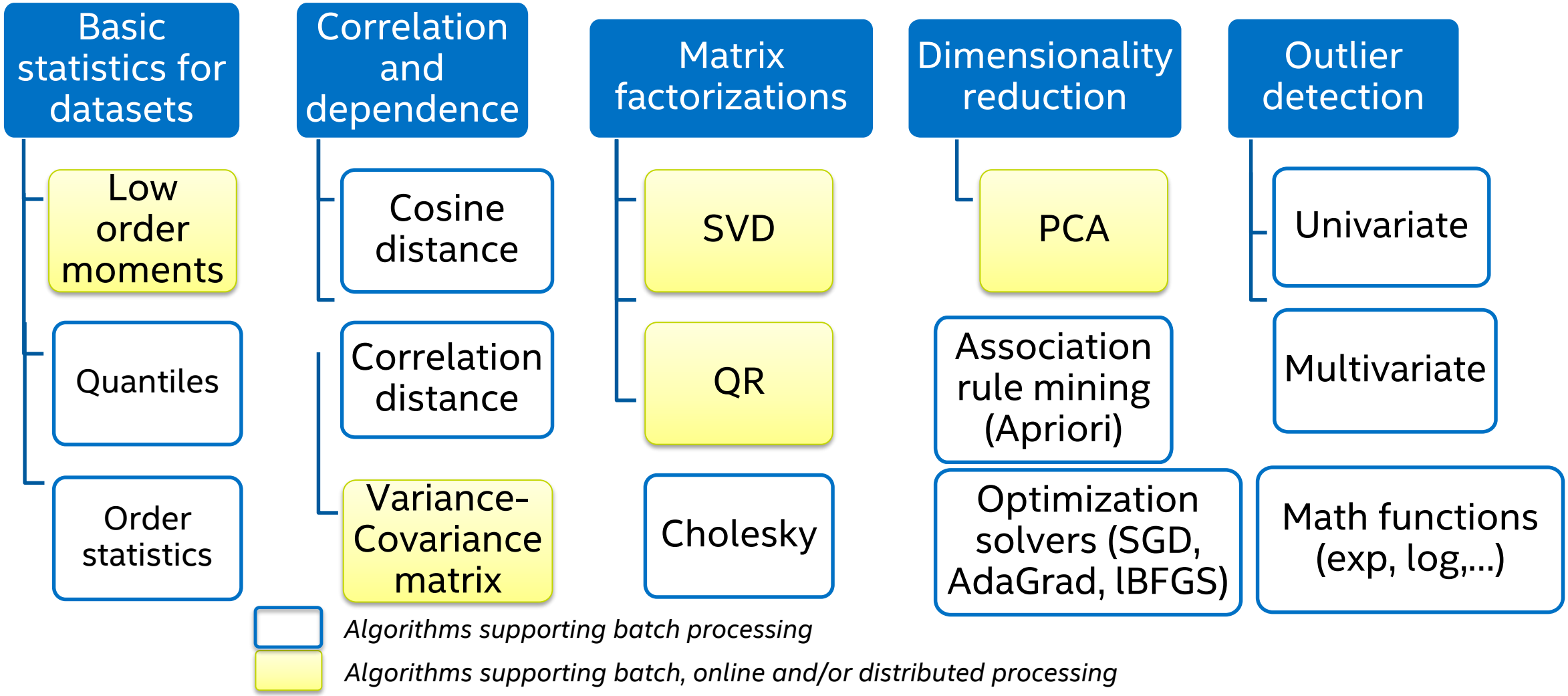
Optimized analytics building blocks for all data analysis stages, from data acquisition to data mining and machine learning

Data Modeling Algorithms

Data structures for model representation, and operations to derive model-based predictions and conclusions

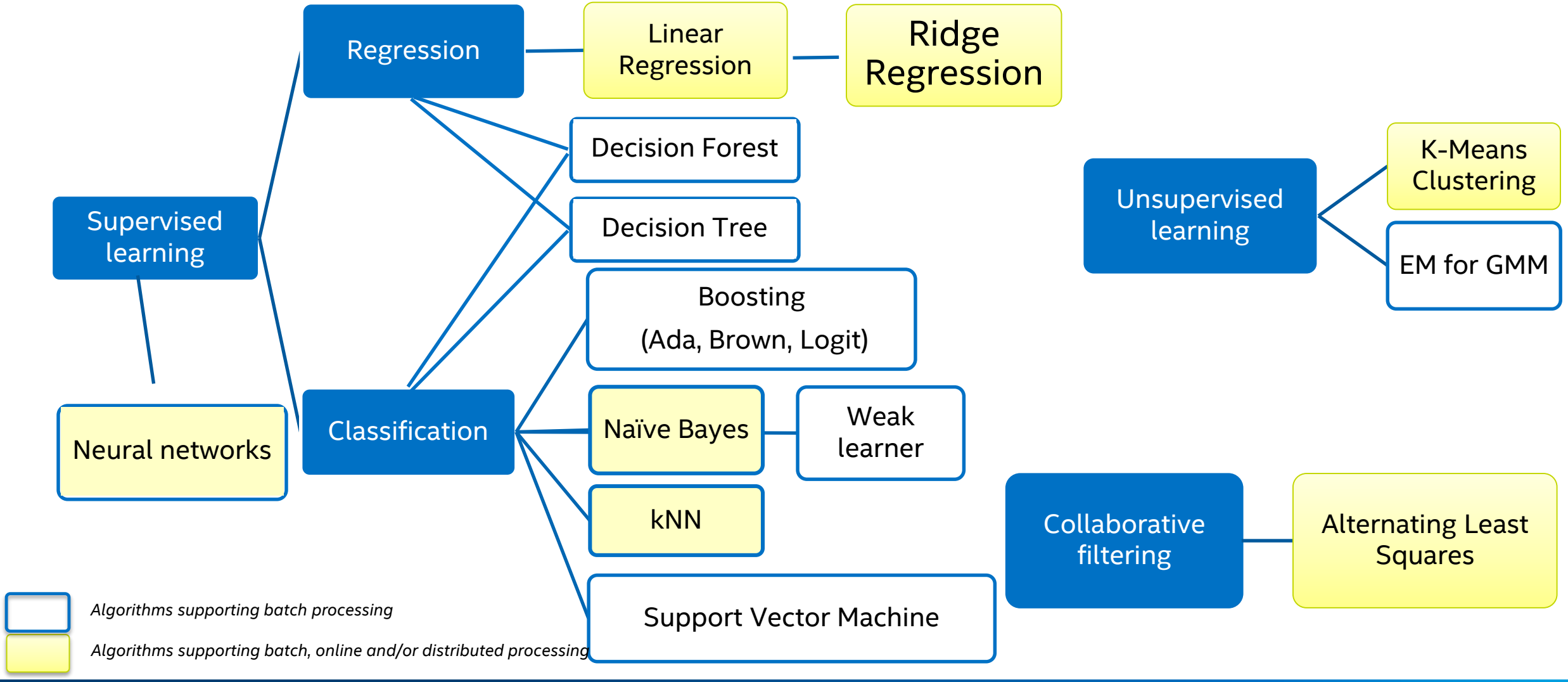
Intel® DAAL Algorithms

Data Transformation and Analysis in Intel® DAAL



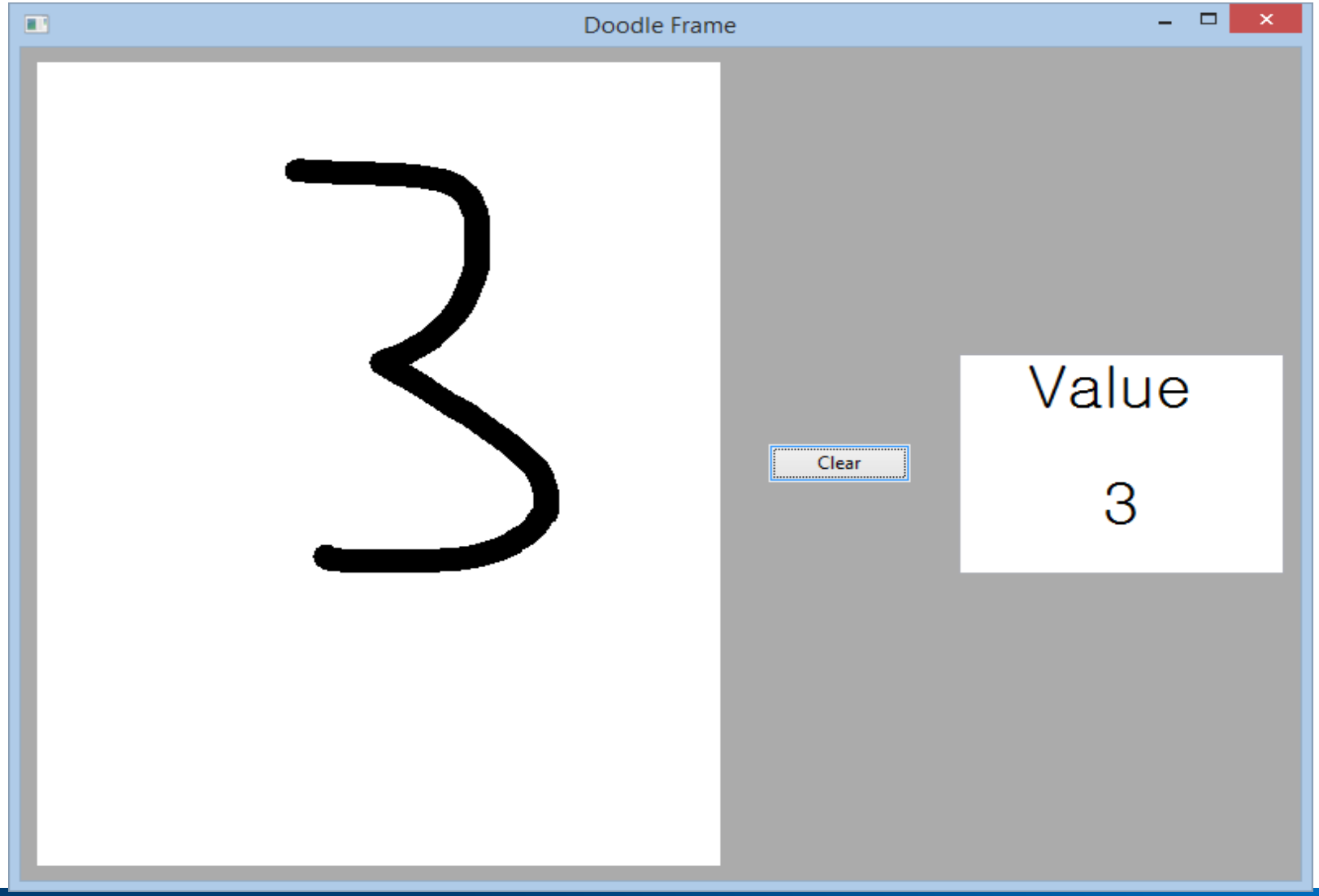
Intel® DAAL Algorithms

Machine Learning in Intel® DAAL



HOW TO USE INTEL® DAAL ALGORITHM'S

Handwritten Digit Recognition



Handwritten Digit Recognition

Training multi-class SVM for 10 digits recognition.

3,823 pre-processed training data.

- available at http://archive.ics.uci.edu/ml/datasets/Optical+Recognition+of+Handwritten+Digits

99.6% accuracy with 1,797 test data from the same data provider.

Confusion matrix:

177.000	0.000	0.000	0.000	1.000	0.000	0.000	0.000	0.000	0.000
0.000	181.000	0.000	0.000	0.000	0.000	0.000	0.000	1.000	0.000
0.000	2.000	173.000	0.000	0.000	0.000	0.000	1.000	1.000	0.000
0.000	0.000	0.000	176.000	0.000	1.000	0.000	0.000	3.000	3.000
0.000	1.000	0.000	0.000	179.000	0.000	0.000	0.000	1.000	0.000
0.000	0.000	0.000	0.000	0.000	180.000	0.000	0.000	0.000	2.000
0.000	0.000	0.000	0.000	0.000	0.000	180.000	0.000	1.000	0.000
0.000	0.000	0.000	0.000	0.000	0.000	0.000	170.000	1.000	8.000
0.000	3.000	0.000	0.000	0.000	0.000	0.000	0.000	166.000	5.000
0.000	0.000	0.000	2.000	0.000	1.000	0.000	0.000	2.000	175.000

Average accuracy: 0.996
Error rate: 0.004
Micro precision: 0.978
Micro recall: 0.978
Micro F-score: 0.978
Macro precision: 0.978
Macro recall: 0.978
Macro F-score: 0.978

Training Handwritten Digits

```
void trainModel()
{
        /* Initialize FileDataSource<CSVFeatureManager> to retrieve input data from .csv file */
        FileDataSource<CSVFeatureManager> trainDataSource(trainDatasetFileName,
                DataSource::doAllocateNumericTable, DataSource::doDictionaryFromContext);

        /* Load data from the data files */
        trainDataSource.loadDataBlock(nTrainObservations);

        /* Create algorithm object for multi-class SVM training */
        multi_class_classifier::training::Batch<> algorithm;

        algorithm.parameter.nClasses = nClasses;
        algorithm.parameter.training = training;

        /* Pass training dataset and dependent values to the algorithm */
        algorithm.input.set(classifier::training::data,trainDataSource.getNumericTable());

        /* Build multi-class SVM model */
        algorithm.compute();

        /* Retrieve algorithm results */
        trainingResult = algorithm.getResult();

        /* Serialize the learned model into a disk file */
        ModelFileWriter writer("./model");
        writer.serializeToFile(trainingResult->get(classifier::training::model));
}
```

Create a numeric table

Create an alg. Obj.

Set input and parameters

Compute

Serialize the learned model

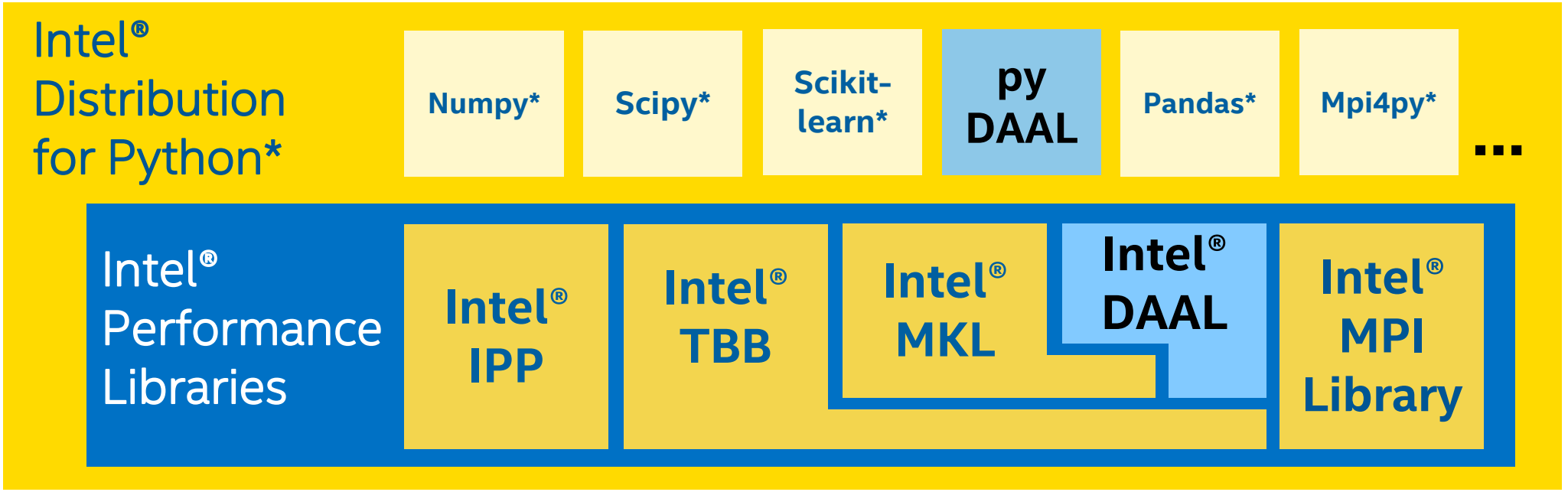
Handwritten Digit Prediction

```
void testDigit()
{
        /* Initialize FileDataSource<CSVFeatureManager> to retrieve the test data
from .csv file */
        FileDataSource<CSVFeatureManager> testDataSource(testDatasetFileName,
                DataSource::doAllocateNumericTable, DataSource::doDictionaryFromContext);
        testDataSource.loadDataBlock(1);

        /* Create algorithm object for prediction of multi-class SVM values */
        multi_class_classifier::prediction::Batch<> algorithm;

        algorithm.parameter.prediction = prediction;

        /* Deserialize a model from a disk file */
        ModelFileReader reader("./model");
        services::SharedPtr<multi_class_classifier::Model> pModel(new
multi_class_classifier::Model());
        reader.deserializeFromFile(pModel);

        /* Pass testing dataset and trained model to the algorithm */
        algorithm.input.set(classifier::prediction::data,
testDataSource.getNumericTable());
        algorithm.input.set(classifier::prediction::model, pModel);

        /* Predict multi-class SVM values */
        algorithm.compute();

        /* Retrieve algorithm results */
        predictionResult = algorithm.getResult();

        /* Retrieve predicted labels */
        predictedLabels = predictionResult->get(classifier::prediction::prediction);
}
```

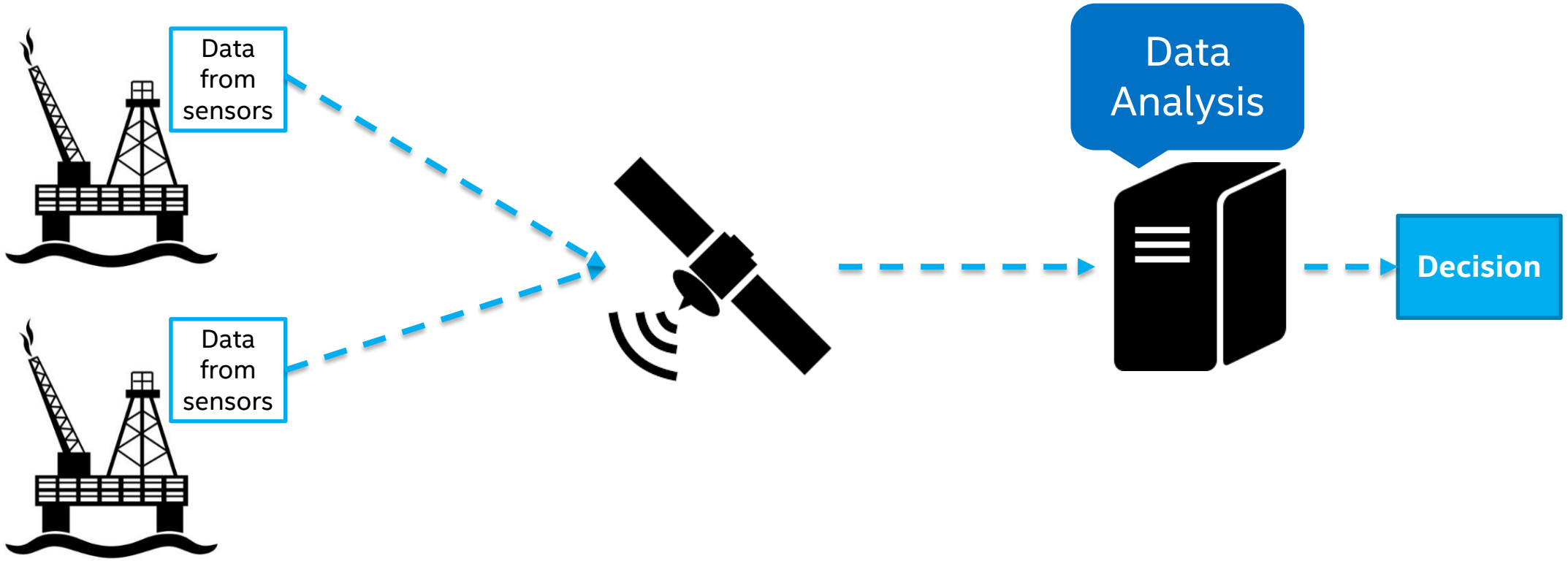
Deserialize learned model

pyDAAL

Intel Python Landscape



Anomaly Detection Problem Example

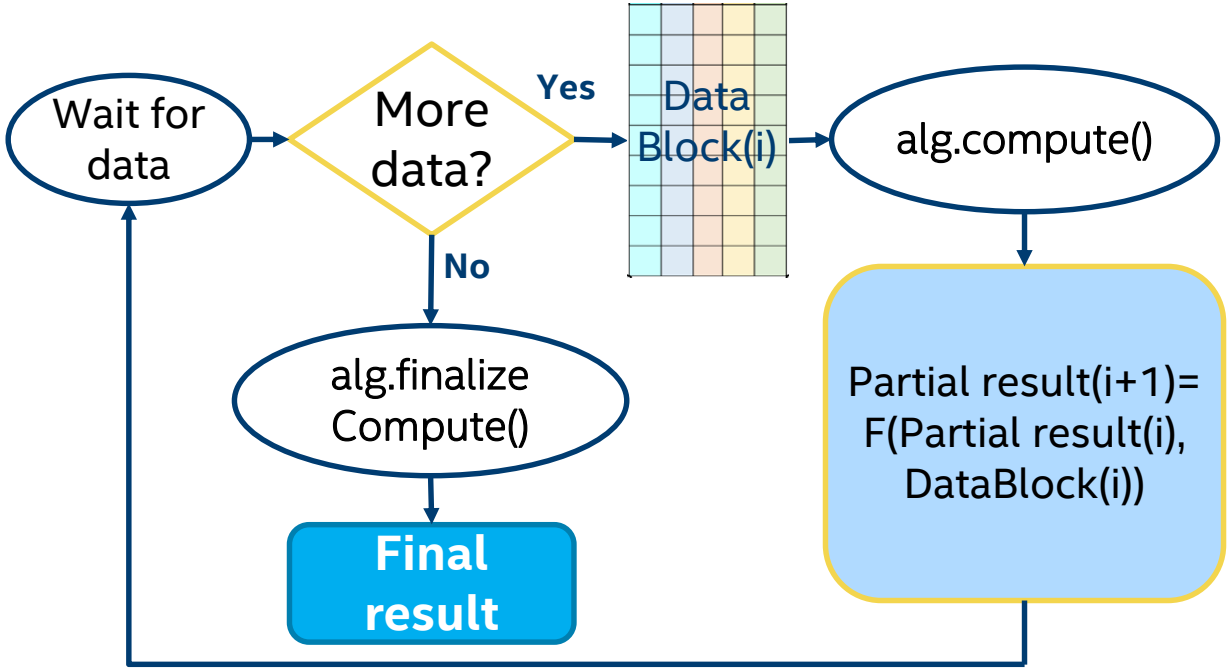


State of the art solution

- The data collected by sensors sent to mainland for analysis and decision making
  - Excessive amount of data is transferred, communication channel is overloaded

Solution: Online Processing

- Update the decision when the new portion of data is available
- The whole data set may not fit into memory but still can be processed on one machine



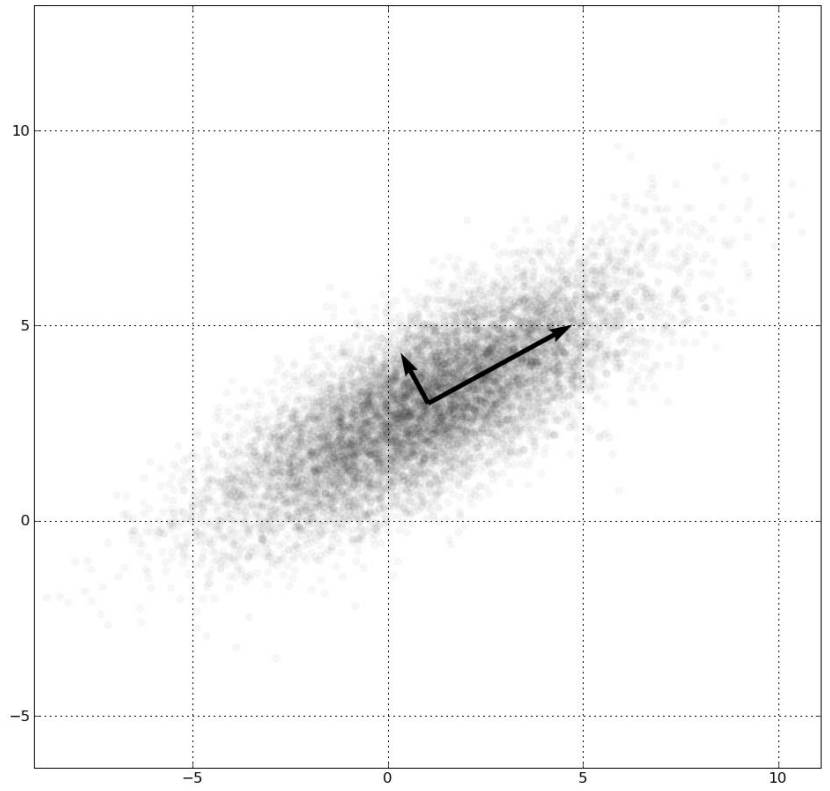
Projection methods for outlier detection

Principal Component Analysis (PCA)

- Computes principal components: the directions of the largest variance, the directions where the data is mostly spread out

PCA for outlier detection

- Project new observation on the space of the first k principal components
- Calculate score distance for the projection using first k singular values
- Compare the distance against threshold



http://i.stack.imgur.com/uYaTv.png

Online Processing

Data sets that does not fit into memory could be processed effectively on a single machine

```
from daal.algorithms.pca import (
   Online_Float64CorrelationDense, data,
    eigenvalues, eigenvectors
)

dataSource = FileDataSource(
    dataFileName,
    DataSourceIface.doAllocateNumericTable,
    DataSourceIface.doDictionaryFromContext
)

algorithm = Online_Float64CorrelationDense()

while(dataSource.loadDataBlock(nVectorsInBlock) >
0):

    algorithm.input.setDataset(data,
       dataSource.getNumericTable())

    algorithm.compute()

result = algorithm.finalizeCompute()

printNumericTable(result.get(eigenvalues))
printNumericTable(result.get(eigenvectors))
```

Initialize data source to retrieve data from CSV files

Create a PCA algorithm using correlation method

Load next block of data from the input file

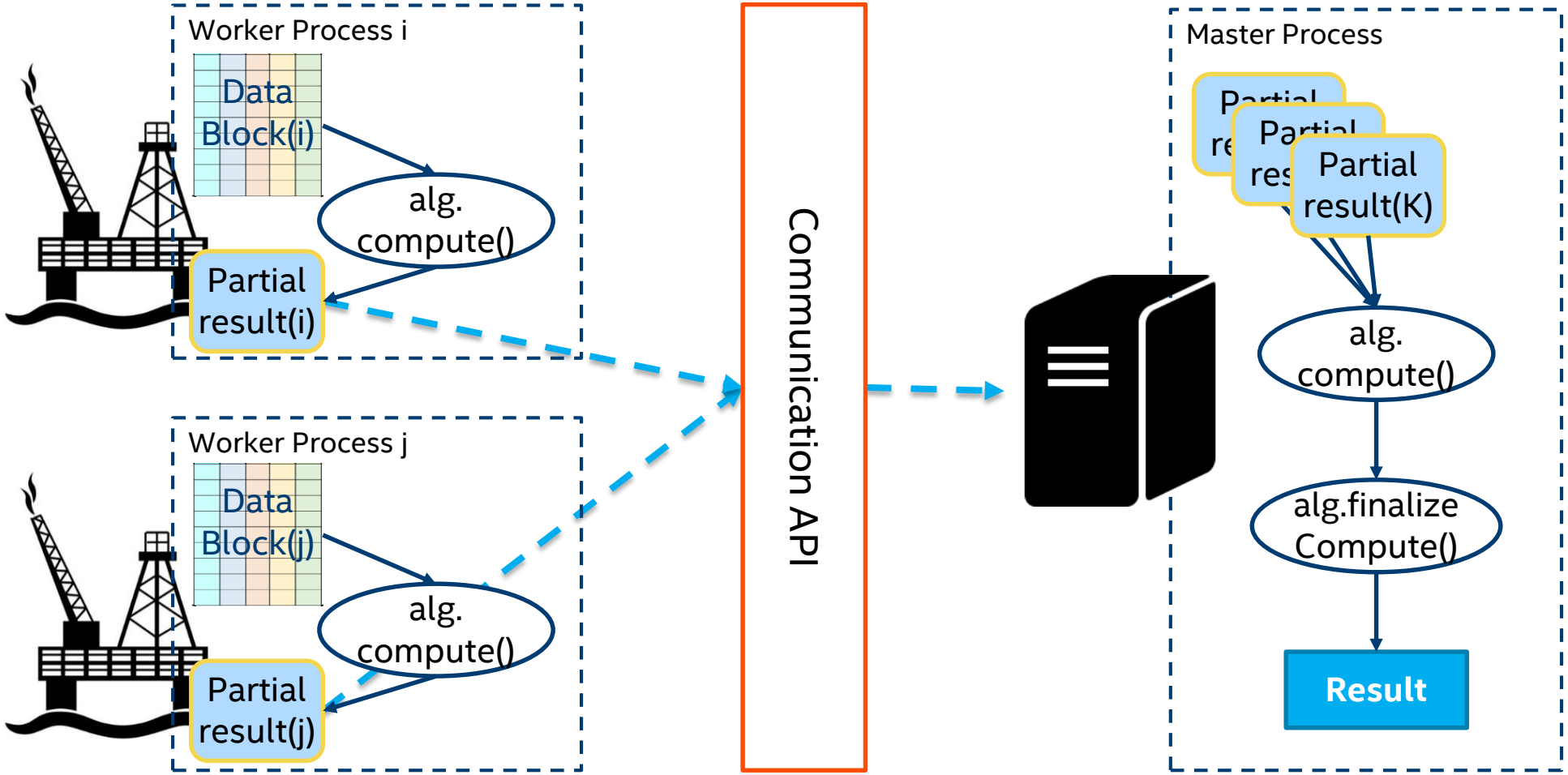
Set input data to the algorithm

Update partial results of the PCA

Retrieve the results of the algorithm

Print the results

Solution: Distributed Processing



Distributed computing landscape

mpi4py

pySpark

Dask/distributed

...

- New distributed computing technologies appear almost every year
- Intel® DAAL provides communication layer agnostic building blocks to create distributed algorithms

Distributed Processing: Step 1 on Worker

```
from daal import step1Local, step2Master
import daal.algorithms.pca as pca
from daal.data_management import (
    OutputDataArchive, InputDataArchive,
    FileDataSource, DataSourceIface
)
```

```
dataSource = FileDataSource(datasetFileNames[rankId], …)
dataSource.loadDataBlock()
```

Initialize data source to retrieve data from CSV files

Retrieve the input data

```
algorithm = pca.Distributed(step1Local)
```

Create a PCA algorithm for using the correlation method on the local node

```
algorithm.input.setDataset(pca.data,
                           dataSource.getNumericTable())
```

Set input data to the algorithm

```
pres = algorithm.compute()
```

Compute partial results of the PCA algorithm

```
dataArch = InputDataArchive()
pres.serialize(dataArch)
nodeResults = dataArch.getArchiveAsArray()
```

Serialize the object that contains the partial results into the array of bytes

```
serializedData = comm.gather(nodeResults)
```

Gather the partial results on master node

Distributed Processing: Step 2 on MASTER

```
if rankId == MPI_ROOT:
  masterAlgorithm = pca.Distributed(step2Master)

  for i in range(nBlocks):
    dataArch =
      OutputDataArchive(serializedData[i])
    dataForStep2FromStep1 =
      pca.PartialResult(pca.correlationDense)
    dataForStep2FromStep1.deserialize(dataArch)

    masterAlgorithm.input.add(
      pca.partialResults, dataForStep2FromStep1)

  masterAlgorithm.compute()

  res = masterAlgorithm.finalizeCompute()

  printNumericTable(res.get(pca.eigenvalues))
  printNumericTable(res.get(pca.eigenvectors))
```

Create a PCA algorithm for using the correlation method on the master node

De-serialize partial results that were gathered from the local nodes

Set local partial results as inputs for the master algorithm

Compute partial results of the PCA algorithm

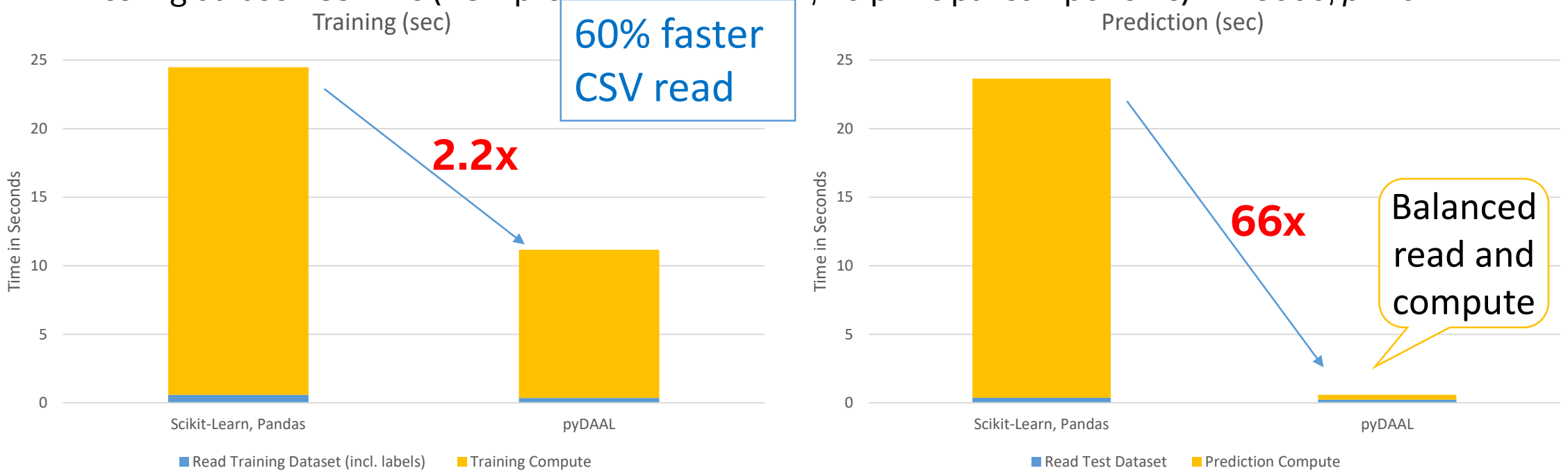
Compute final results of the PCA algorithm

Print the results

Performance Example : Read And Compute

SVM Classification with RBF kernel

- Training dataset: CSV file (PCA-preprocessed MNIST, 40 principal components) *n*=42000, *p*=40
- Testing dataset: CSV file (PCA-preprocessed MNIST, 40 principal components) *n*=28000, *p*=40



System Info: Intel® Xeon® CPU E5-2680 v3 @ 2.50GHz, 504GB, 2x24 cores, HT=on, OS RH7.2 x86_64, Intel® Distribution for Python* 2017 Update 1 (Python* 3.5)

References

Where to find DAAL/MKL/Intel distribution for python*

- Sub-component of Intel Parallel Studio XE : https://software.intel.com/en-us/intel-parallel-studio-xe

- Standalone access: https://software.intel.com/en-us/mkl , https://software.intel.com/en-us/intel-daal
- Free access through High Performance Library (Intel® MKL/IPP/DAAL)
  https://software.intel.com/en-us/performance-libraries
- Git HUB : https://github.com/01org/daal
- YUM/APT repository for Intel® MKL/IPP/DAAL & Intel Distribution for Python for Linux* OS
  https://software.intel.com/en-us/articles/installing-intel-free-libs-and-python-yum-repo
  https://software.intel.com/en-us/articles/installing-intel-free-libs-and-python-apt-repo

Legal Disclaimer & Optimization Notice

INFORMATION IN THIS DOCUMENT IS PROVIDED “AS IS”. NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED BY THIS DOCUMENT. INTEL ASSUMES NO LIABILITY WHATSOEVER AND INTEL DISCLAIMS ANY EXPRESS OR IMPLIED WARRANTY, RELATING TO THIS INFORMATION INCLUDING LIABILITY OR WARRANTIES RELATING TO FITNESS FOR A PARTICULAR PURPOSE, MERCHANTABILITY, OR INFRINGEMENT OF ANY PATENT, COPYRIGHT OR OTHER INTELLECTUAL PROPERTY RIGHT.

Software and workloads used in performance tests may have been optimized for performance only on Intel microprocessors. Performance tests, such as SYSmark and MobileMark, are measured using specific computer systems, components, software, operations and functions. Any change to any of those factors may cause the results to vary. You should consult other information and performance tests to assist you in fully evaluating your contemplated purchases, including the performance of that product when combined with other products.

For more complete information about compiler optimizations, see our Optimization Notice at https://software.intel.com/en-us/articles/optimization-notice#opt-en.

BACKUP

Configuration Details

Platform: 2S Intel® Xeon® CPU E5-2697 v2 @ 2.70GHz (12 cores), HT enabled, turbo enabled, scaling governor set to “performance” via intel_pstate driver, 256GB DDR3-1600 ECC RAM. CentOS Linux release 7.3.1611 (Core), Linux kernel 3.10.0-514.21.1.el7.x86_64. SSD: Intel® SSD 520 Series 240GB, 2.5in SATA 6Gb/s, 25nm, MLC.

Performance measured with: Environment variables: KMP_AFFINITY='granularity=fine, compact,1,0', OMP_NUM_THREADS=24, CPU Freq set with cpupower frequency-set -d 2.7G -u 3.5G -g performance

Deep Learning Frameworks:

- Caffe: (http://github.com/intel/caffe/), revision b0ef3236528a2c7d2988f249d347d5fdae831236. Inference measured with “caffe time --forward_only” command, training measured with “caffe time” command. For “ConvNet” topologies, dummy dataset was used. For other topologies, data was stored on local storage and cached in memory before training. Topology specs from https://github.com/intel/caffe/tree/master/models/intel_optimized_models (GoogLeNet, AlexNet, and ResNet-50), https://github.com/intel/caffe/tree/master/models/default_vgg_19 (VGG-19), and https://github.com/soumith/convnet-benchmarks/tree/master/caffe/imagenet_winners (ConvNet benchmarks; files were updated to use newer Caffe prototxt format but are functionally equivalent). GCC 4.8.5, Intel MKL small libraries version 2017.0.2.20170110.

Configuration Details

Platform: 2S Intel® Xeon® Platinum 8180 CPU @ 2.50GHz (28 cores), HT disabled, turbo disabled, scaling governor set to "performance" via intel_pstate driver, 384GB DDR4-2666 ECC RAM. CentOS Linux release 7.3.1611 (Core), Linux kernel 3.10.0-514.10.2.el7.x86_64. SSD: Intel® SSD DC S3700 Series (800GB, 2.5in SATA 6Gb/s, 25nm, MLC).

Performance measured with: Environment variables: KMP_AFFINITY='granularity=fine, compact', OMP_NUM_THREADS=56, CPU Freq set with cpupower frequency-set -d 2.5G -u 3.8G -g performance

Deep Learning Frameworks:

- Caffe: (http://github.com/intel/caffe/), revision f96b759f71b2281835f690af267158b82b150b5c. Inference measured with "caffe time --forward_only" command, training measured with "caffe time" command. For "ConvNet" topologies, dummy dataset was used. For other topologies, data was stored on local storage and cached in memory before training. Topology specs from https://github.com/intel/caffe/tree/master/models/intel_optimized_models (GoogLeNet, AlexNet, and ResNet-50), https://github.com/intel/caffe/tree/master/models/default_vgg_19 (VGG-19), and https://github.com/soumith/convnet-benchmarks/tree/master/caffe/imagenet_winners (ConvNet benchmarks; files were updated to use newer Caffe prototxt format but are functionally equivalent). Intel C++ compiler ver. 17.0.2 20170213, Intel MKL small libraries version 2018.0.20170425. Caffe run with "numactl -l".

- TensorFlow: (https://github.com/tensorflow/tensorflow), commit id 207203253b6f8ea5e938a512798429f91d5b4e7e. Performance numbers were obtained for three convnet benchmarks: alexnet, googlenetv1, vgg(https://github.com/soumith/convnet-benchmarks/tree/master/tensorflow) using dummy data. GCC 4.8.5, Intel MKL small libraries version 2018.0.20170425, interop parallelism threads set to 1 for alexnet, vgg benchmarks, 2 for googlenet benchmarks, intra op parallelism threads set to 56, data format used is NCHW, KMP_BLOCKTIME set to 1 for googlenet and vgg benchmarks, 30 for the alexnet benchmark. Inference measured with --caffe time -forward_only -engine MKL2017option, training measured with --forward_backward_only option.

- MxNet: (https://github.com/dmlc/mxnet/), revision 5efd91a71f36fea483e882b0358c8d46b5a7aa20. Dummy data was used. Inference was measured with "benchmark_score.py", training was measured with a modified version of benchmark_score.py which also runs backward propagation. Topology specs from https://github.com/dmlc/mxnet/tree/master/example/image-classification/symbols. GCC 4.8.5, Intel MKL small libraries version 2018.0.20170425.

- Neon: ZP/MKL_CHWN branch commit id:52bd02acb947a2adabb8a227166a7da5d9123b6d. Dummy data was used. The main.py script was used for benchmarking , in mkl mode. ICC version used : 17.0.3 20170404, Intel MKL small libraries version 2018.0.20170425.

Configuration Details

Platform: 2S Intel® Xeon® CPU E5-2699 v4 @ 2.20GHz (22 cores), HT enabled, turbo disabled, scaling governor set to "performance" via acpi-cpufreq driver, 256GB DDR4-2133 ECC RAM. CentOS Linux release 7.3.1611 (Core), Linux kernel 3.10.0-514.10.2.el7.x86_64. SSD: Intel® SSD DC S3500 Series (480GB, 2.5in SATA 6Gb/s, 20nm, MLC).

Performance measured with: Environment variables: KMP_AFFINITY='granularity=fine, compact,1,0', OMP_NUM_THREADS=44, CPU Freq set with cpupower frequency-set -d 2.2G -u 2.2G -g performance

Deep Learning Frameworks:

- Caffe: (http://github.com/intel/caffe/), revision f96b759f71b2281835f690af267158b82b150b5c. Inference measured with "caffe time --forward_only" command, training measured with "caffe time" command. For "ConvNet" topologies, dummy dataset was used. For other topologies, data was stored on local storage and cached in memory before training. Topology specs from https://github.com/intel/caffe/tree/master/models/intel_optimized_models (GoogLeNet, AlexNet, and ResNet-50), https://github.com/intel/caffe/tree/master/models/default_vgg_19 (VGG-19), and https://github.com/soumith/convnet-benchmarks/tree/master/caffe/imagenet_winners (ConvNet benchmarks; files were updated to use newer Caffe prototxt format but are functionally equivalent). GCC 4.8.5, Intel MKL small libraries version 2017.0.2.20170110.

- TensorFlow: (https://github.com/tensorflow/tensorflow), commit id 207203253b6f8ea5e938a512798429f91d5b4e7e. Performance numbers were obtained for three convnet benchmarks: alexnet, googlenetv1, vgg(https://github.com/soumith/convnet-benchmarks/tree/master/tensorflow) using dummy data. GCC 4.8.5, Intel MKL small libraries version 2018.0.20170425, interop parallelism threads set to 1 for alexnet, vgg benchmarks, 2 for googlenet benchmarks, intra op parallelism threads set to 44, data format used is NCHW, KMP_BLOCKTIME set to 1 for googlenet and vgg benchmarks, 30 for the alexnet benchmark. Inference measured with --caffe time -forward_only -engine MKL2017option, training measured with --forward_backward_only option.

- MxNet: (https://github.com/dmlc/mxnet/), revision e9f281a27584cdb78db8ce6b66e648b3dbc10d37. Dummy data was used. Inference was measured with "benchmark_score.py", training was measured with a modified version of benchmark_score.py which also runs backward propagation. Topology specs from https://github.com/dmlc/mxnet/tree/master/example/image-classification/symbols. GCC 4.8.5, Intel MKL small libraries version 2017.0.2.20170110.

- Neon: ZP/MKL_CHWN branch commit id:52bd02acb947a2adabb8a227166a7da5d9123b6d. Dummy data was used. The main.py script was used for benchmarking , in mkl mode. ICC version used : 17.0.3 20170404, Intel MKL small libraries version 2018.0.20170425.