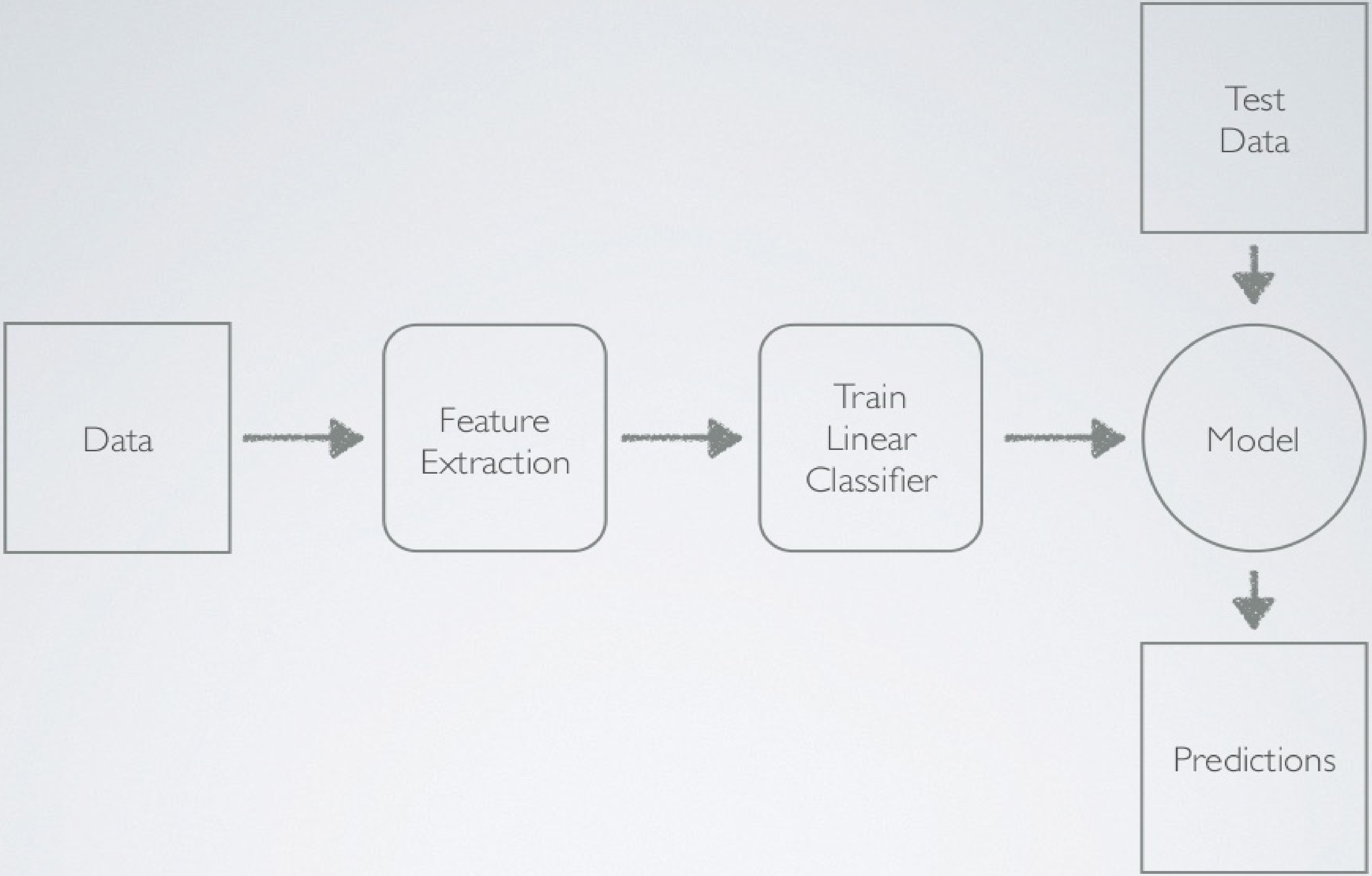
OPTIMIZING TERASCALE MACHINE LEARNING PIPELINES WITH KeystoneML

Apache
Evan R. Sparks, UC Berkeley AMPLab
with Shivaram Venkataraman, Tomer Kaftan, Michael Franklin, Benjamin Recht

WHAT'S A MACHINE LEARNING PIPELINE?

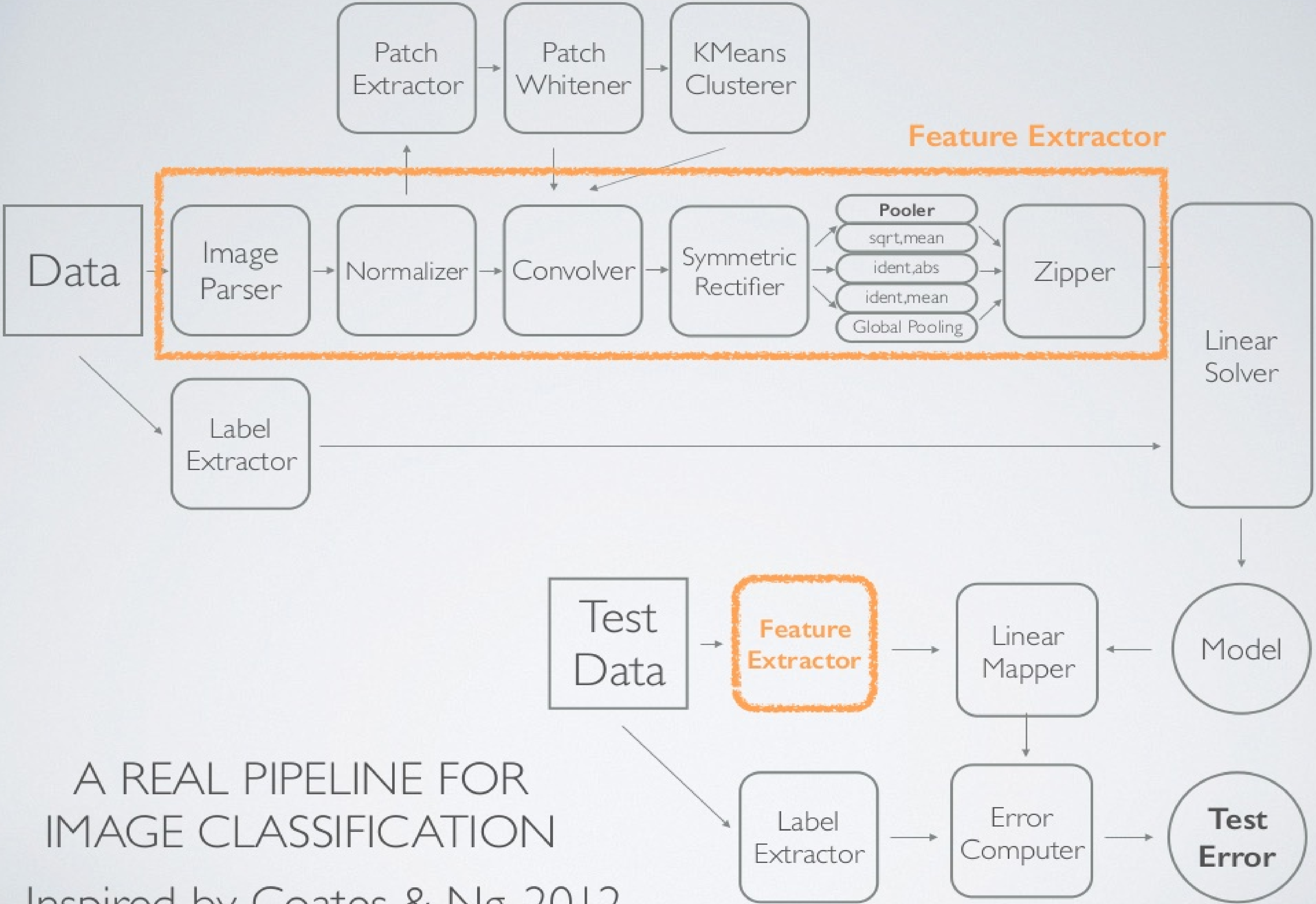
A STANDARD MACHINE LEARNING PIPELINE

Right?



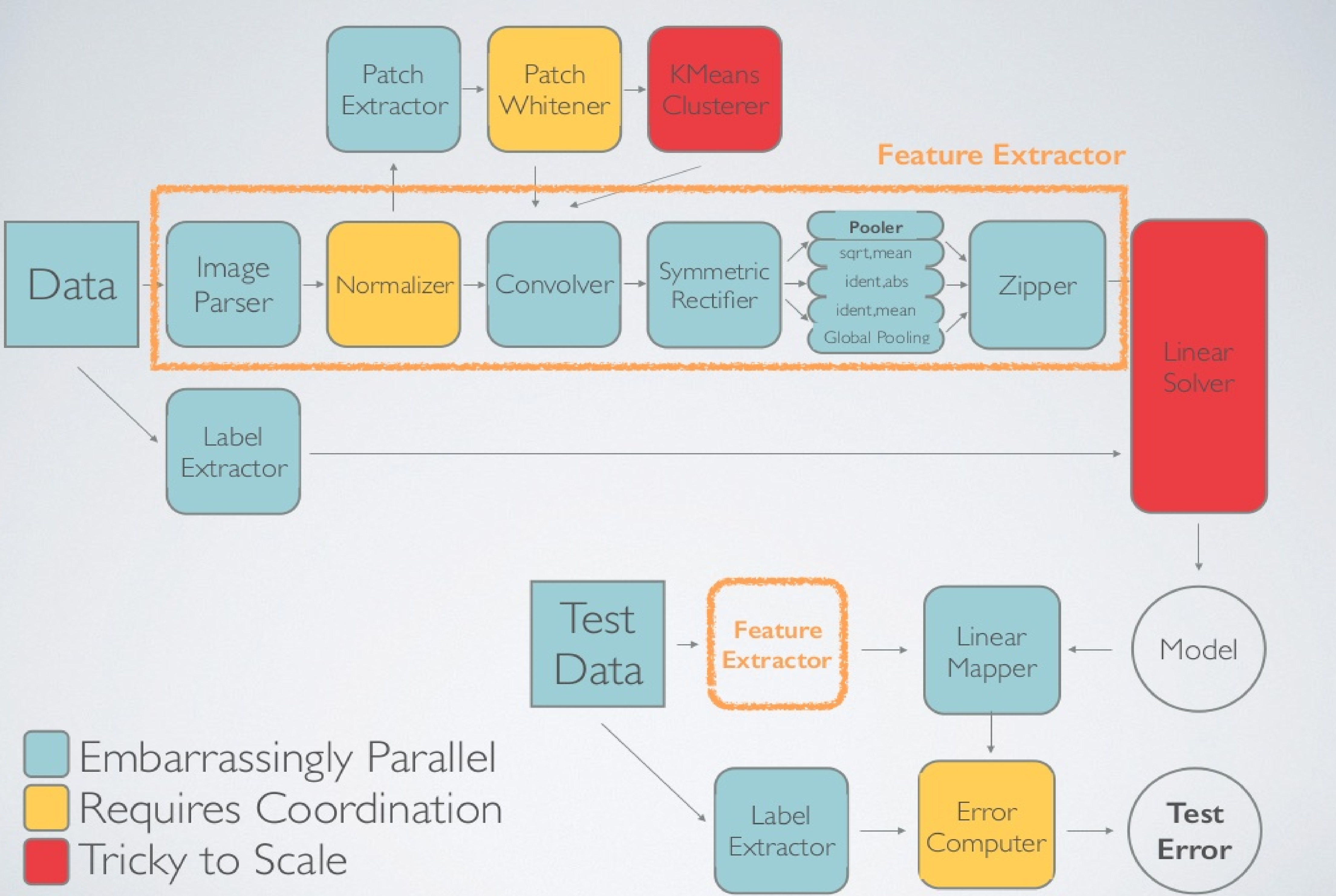
A STANDARD MACHINE LEARNING PIPELINE

That's more like it!



A REAL PIPELINE FOR IMAGE CLASSIFICATION

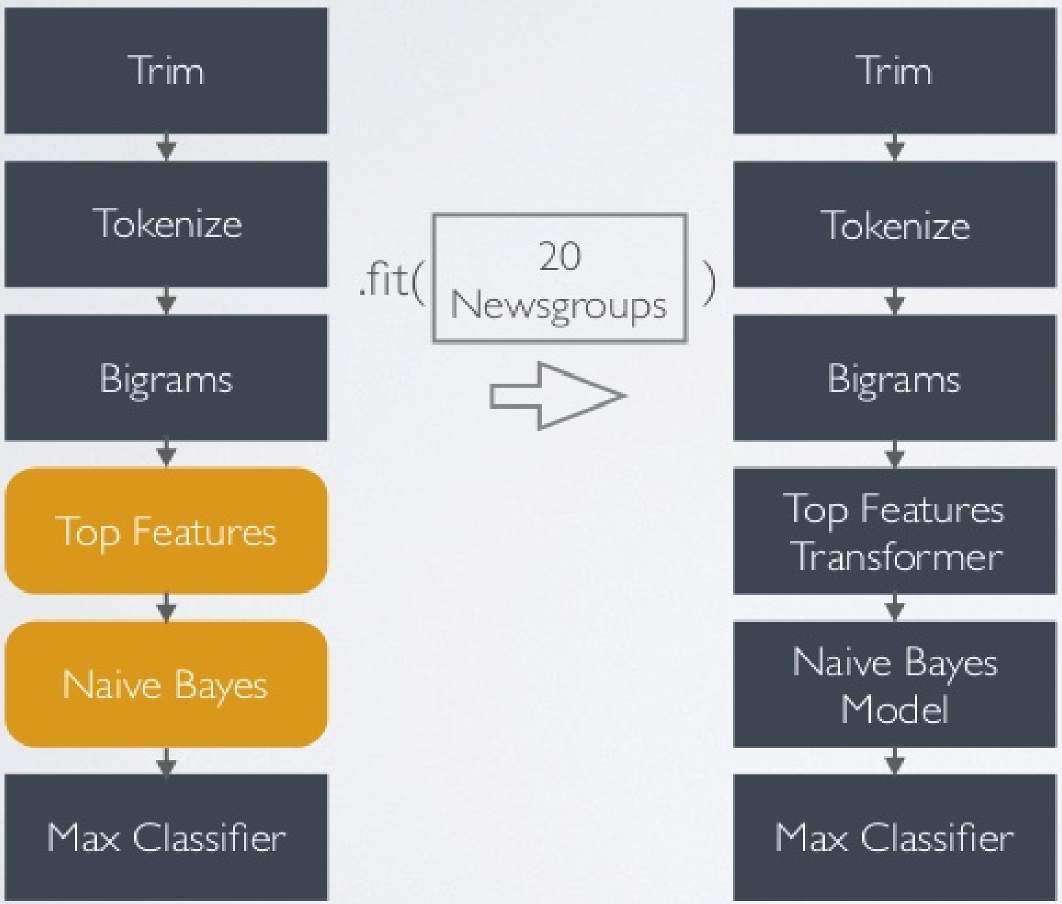
Inspired by Coates & Ng 2012


Patch Extractor
Patch Whitener
KMeans Clusterer
Feature Extractor
Data
Image Parser
Normalizer
Convolver
Symmetric Rectifier
Pooler
sqrt,mean
ident,abs
ident,mean
Global Pooling
Zipper
Linear Solver
Label Extractor
Test Data
Feature Extractor
Linear Mapper
Model
Label Extractor
Error Computer
Test Error
Embarrassingly Parallel
Requires Coordination
Tricky to Scale

ABOUT KEYSTONEML

- Software framework for building ***scalable end-to-end*** machine learning pipelines on ***Apache Spark***.
- Helps us understand what it means to build systems for ***robust***, ***scalable***, end-to-end ***advanced analytics*** workloads and the ***patterns*** that emerge.
- Example pipelines that achieve ***state-of-the-art*** results on ***large scale datasets*** in computer vision, NLP, and speech - ***fast***.
- Open source software, available at: http://keystone-ml.org/

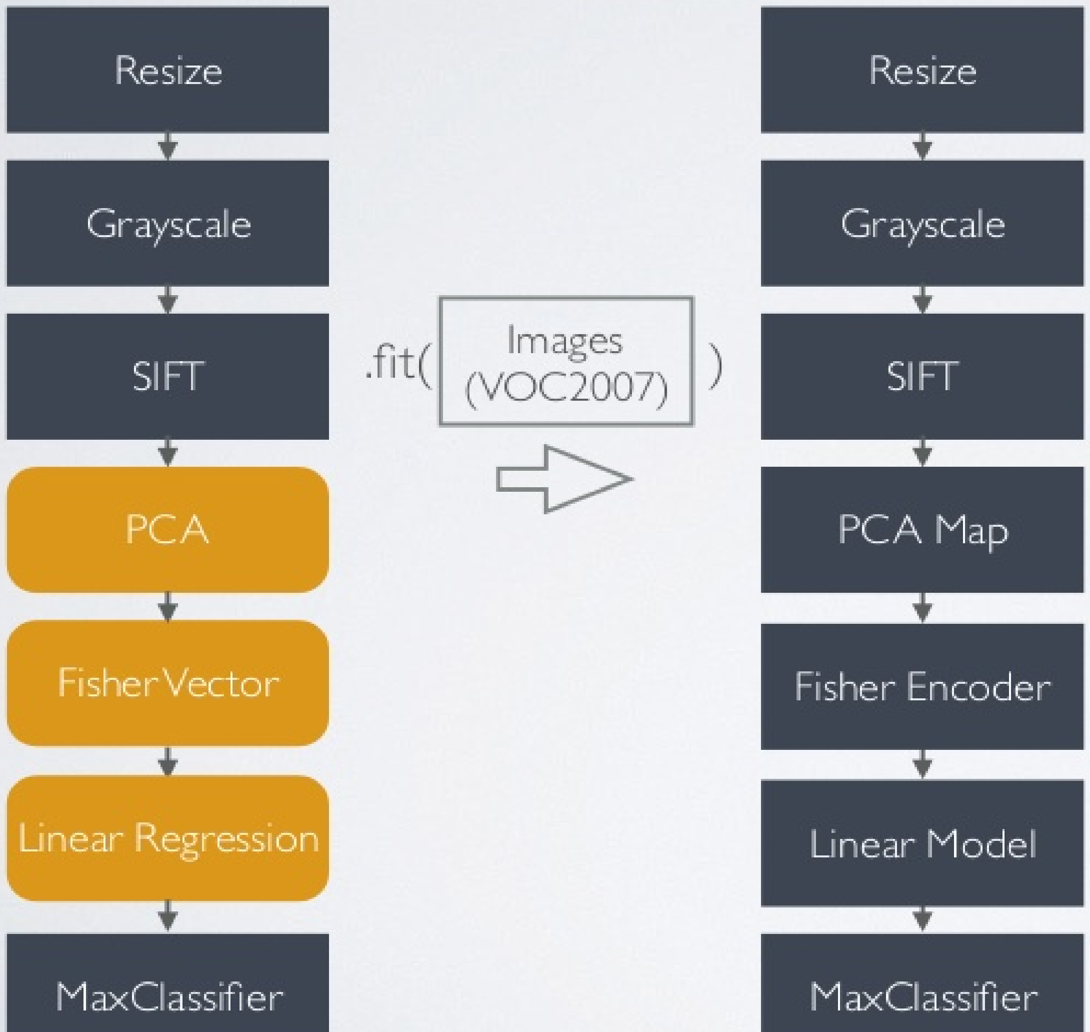
SIMPLE EXAMPLE: TEXT CLASSIFICATION




```
val predictorPipeline = Trim andThen LowerCase() andThen
  Tokenizer() andThen
  NGramsFeaturizer(1 to conf.nGrams) andThen
  TermFrequency(x => 1) andThen
  (CommonSparseFeatures(conf.commonFeatures), training) andThen
  (NaiveBayesEstimator(2), training, labels) andThen
  MaxClassifier
```

Once estimated - apply these steps to your production data in an online or batch fashion.

NOT SO SIMPLE EXAMPLE: IMAGE CLASSIFICATION




```
val predictor = PixelScaler andThen
  GrayScaler andThen
  new Cacher andThen
  new SIFTExtractor(scaleStep = conf.scaleStep) andThen
  new BatchPCATransformer(pca) andThen
  new FisherVector(gmm) andThen
  FloatToDouble andThen
  MatrixVectorizer andThen
  NormalizeRows andThen
  SignedHellingerMapper andThen
  (new LeastSquaresEstimator, trainingData, trainingLabels)
```

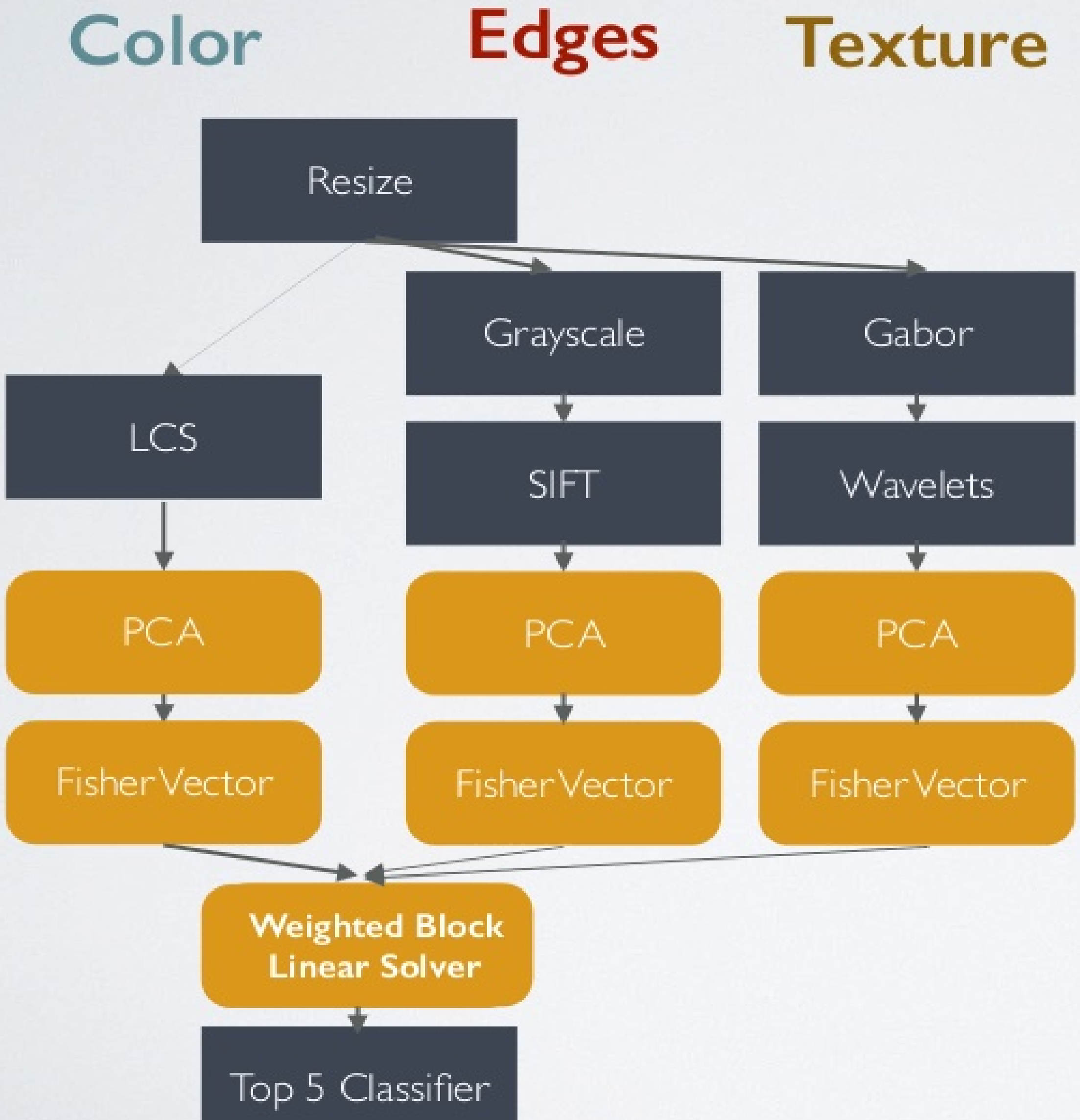
5,000 examples, 40,000 features, 20 classes

Pleasantly parallel featurization and evaluation.

7 minutes on a modest cluster.

Achieves performance of Chatfield et al. 2011

EVEN LESS SIMPLE: IMAGENET



<100 SLOC

1000 class classification.
1,200,000 examples
64,000 features.

90 minutes on 100 nodes.

Upgrading the solver
for higher precision
means changing 1 LOC.

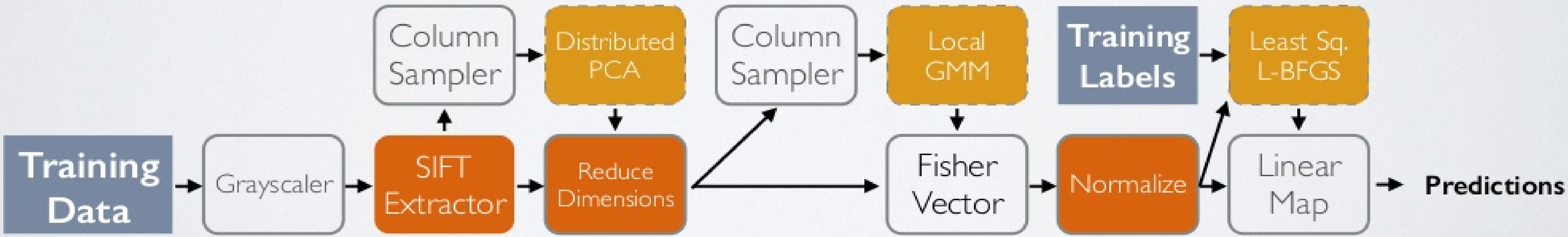
Adding 100,000 more

OPTIMIZING KEYSTONEML PIPELINES

High-level API enables rich space of optimizations

Automated ML operator selection.

Auto-caching for iterative workloads.



KEYSTONEML OPTIMIZER

- Sampling-based ***cost model*** projects resource usage
  - CPU, Memory, Network
- Utilization tracked through pipeline.
  - Decisions made to minimize total cost of execution.
- Catalyst-based optimizer does the heavy lifting.

Stage	n	d	size (GB)
Input	5000	1m pixel JPEG	0.4
Resize	5000	260k pixels	3.6
Grayscale	5000	260k pixels	1.2
SIFT	5000	65000x128	309
PCA	5000	65000x80	154
FV	5000	256x64x2	1.2
Linear Regression	5000	20	0.0007
Max Classifier	5000	1	0.00009

CHOOSING A SOLVER

- Datasets have a number of interesting degrees of freedom.
  - Problem size (n, d, k)
  - sparsity (nnz)
  - condition number
- Platform has degrees of freedom:
  - Memory, CPU, Network, Nodes
- Solvers are predictable!

Objective:

$$\min_{X} |AX - B|_2^2 + \lambda |X|_2^2$$

Where:

$$A \in \mathbb{R}^{n \times d}$$

$$X \in \mathbb{R}^{d \times k}$$

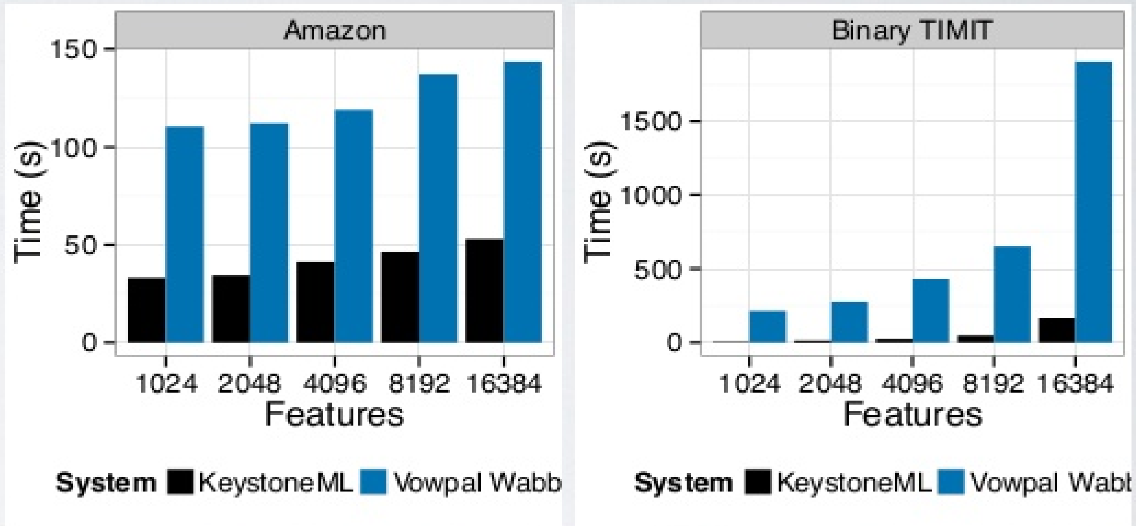
$$B \in \mathbb{R}^{n \times k}$$

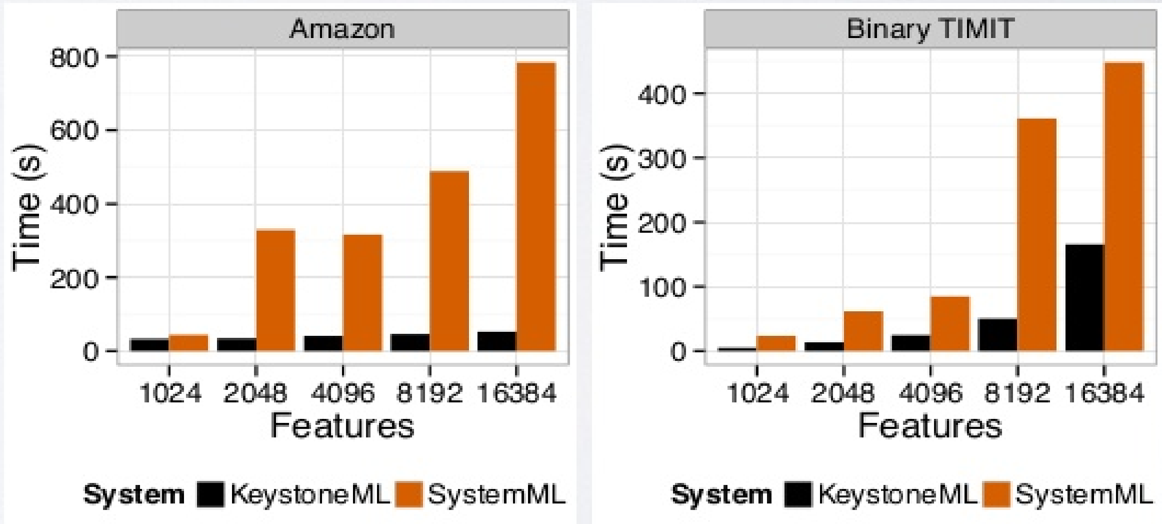
CHOOSING A SOLVER

- Three Solvers
  - Exact, Block, LBFGS
- Two datasets
  - Amazon - >99% sparse, n=65m
  - TIMIT - dense, n=2m
- Exact solve works well for small # features.
- Use LBFGS for sparse problems.
- Block solver scales well to big dense problems.
  - Hundreds of thousands of features.

SOLVER PERFORMANCE

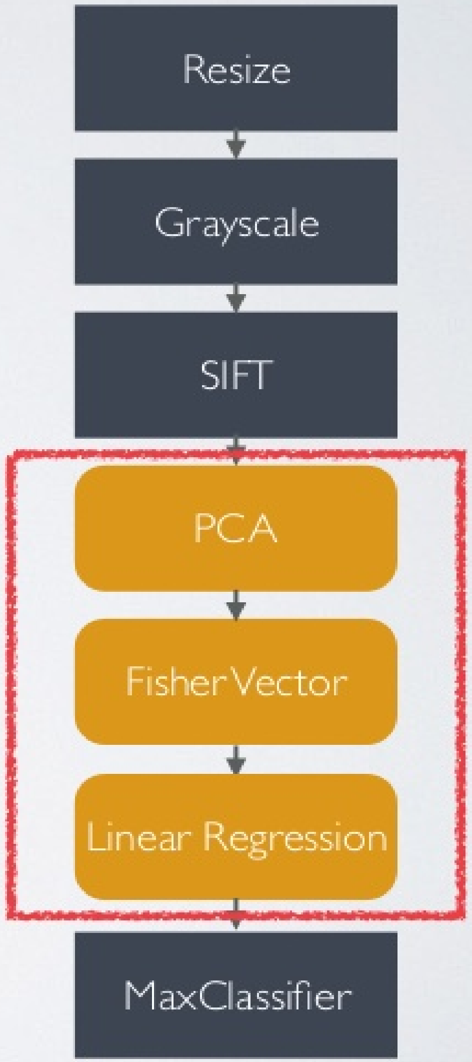
- Compared KeystoneML with:
  - VowpalWabbit - specialized system for large, sparse problems.
  - SystemML - general purpose, optimizing ML system.
- Two problems:
  - Amazon - Sparse text features.
  - Binary TIMIT - Dense phoneme data.
- High Order Bit:
  - KeystoneML ***pipelines featurization*** and ***adapts*** to workload changes.





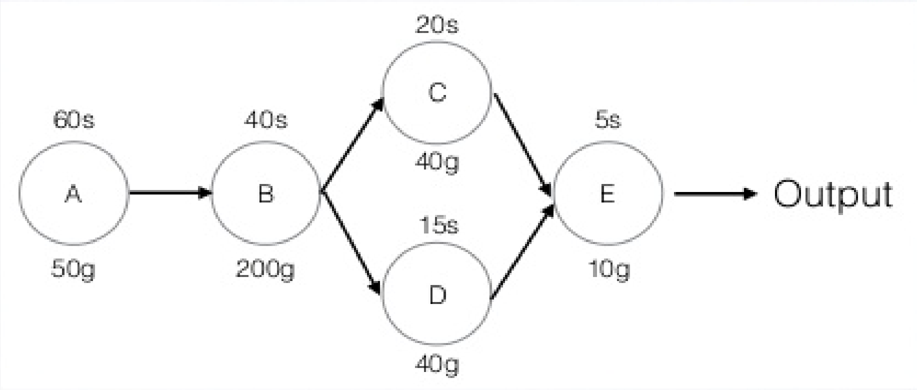
DECIDING WHAT TO SAVE

- Pipelines Generate Lots of intermediate state.
  - E.g. SIFT features blow up a 0.42GB VOC dataset to 300GB.
- Iterative algorithms —> state needed many times.
- How do we determine what to save for later and what to reuse, given fixed resource budget?
- Can we **adapt** to workload changes?



CACHING PROBLEM

- Output is computed via depth-first execution of DAG.
  - Caching "truncates" a path after first visit.
- Want to minimize execution time.
  - Subject to memory constraints.
- Picking optimal set is hard!



Cache set	Time	Memory
ABCDE	140s	340g
B	140s	200g
A	180s	50g
{}	240s	0g

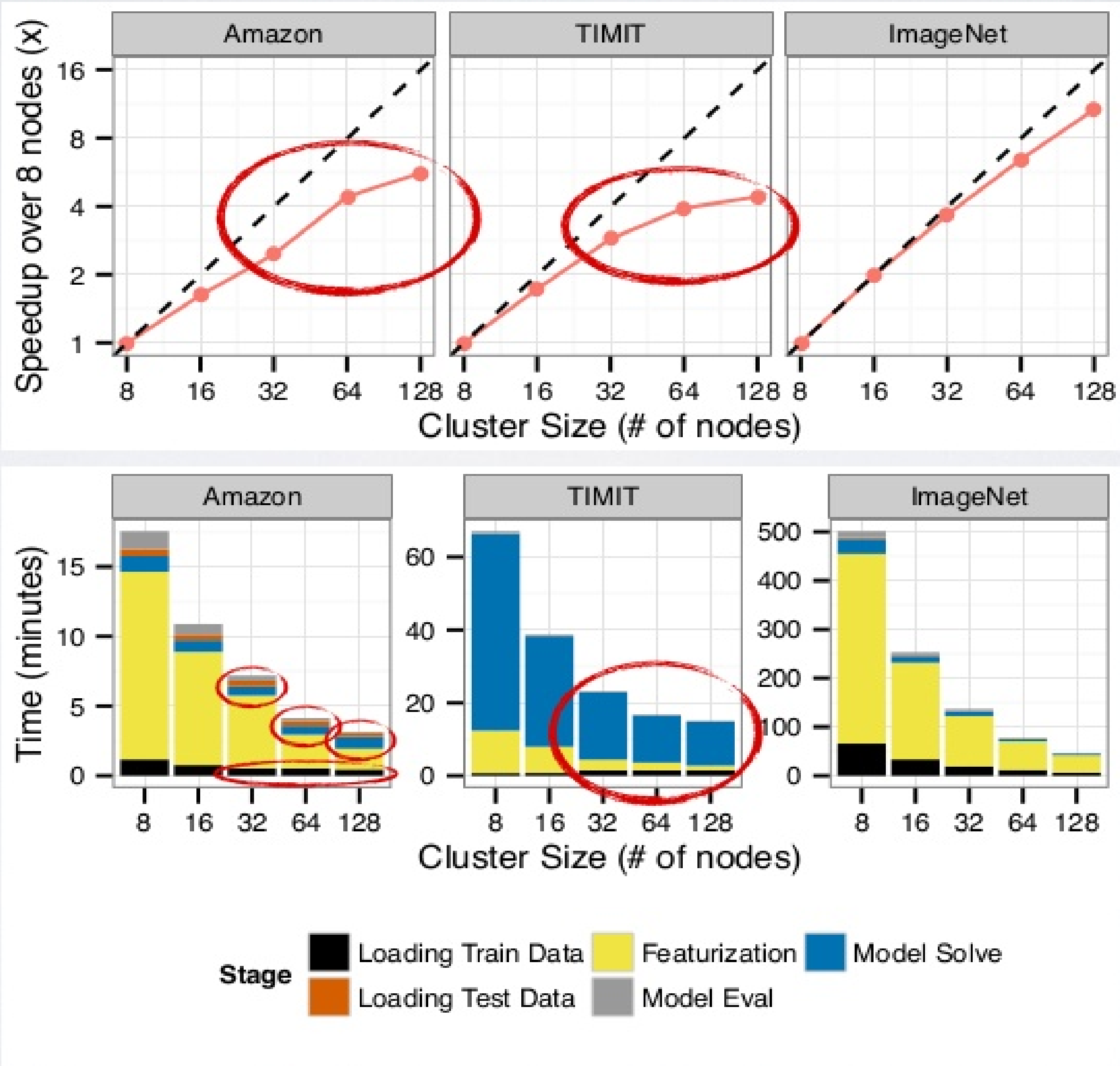
END-TO-END PERFORMANCE

Dataset	Training Examples	Features	Raw Size (GB)	Feature Size (GB)
Amazon	65 million	100k (sparse)	14	89
TIMIT	2.25 million	528k	7.5	8800
ImageNet	1.28 million	262k	74	2500
VOC	5000	40k	0.43	1.5

END-TO-END PERFORMANCE

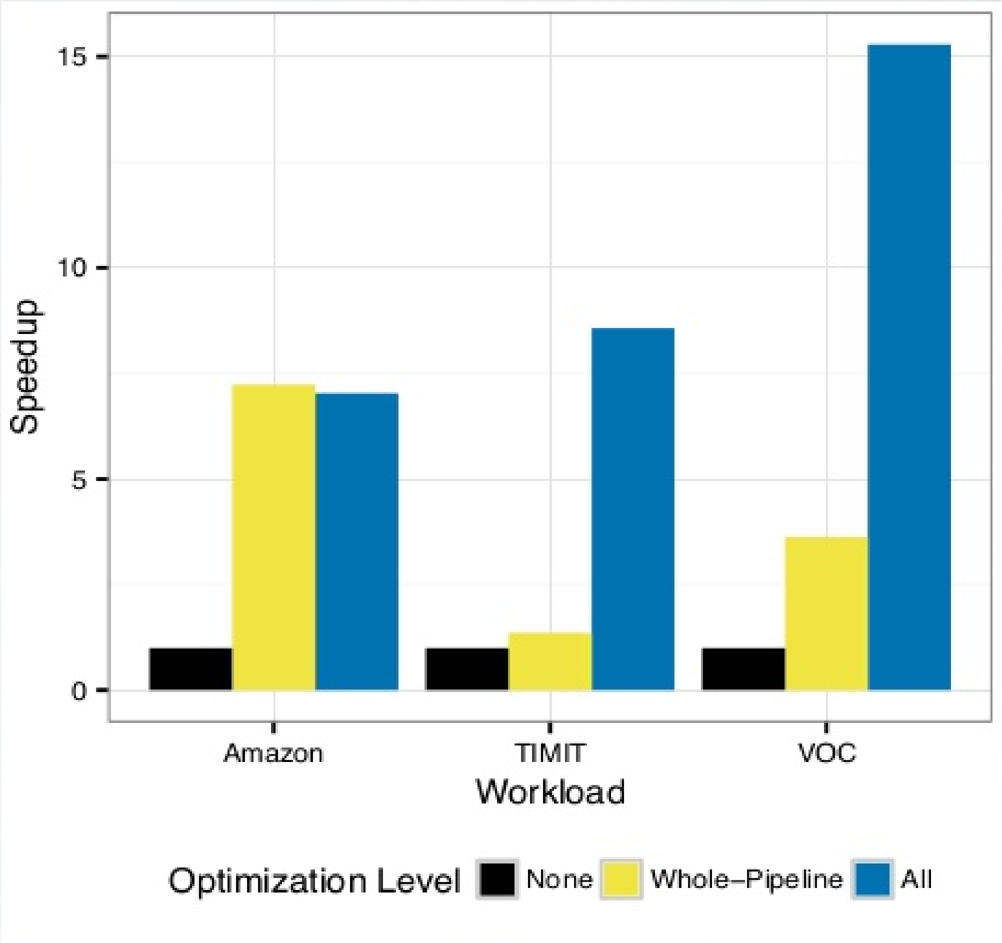
Dataset	KeystoneML Accuracy	Reported Accuracy	KeystoneML Time (m)	Reported Time (m)	Speedup over Reported
Amazon	91.6%	N/A	3.3	N/A	N/A
TIMIT	66.1%	66.3%	138	120	0.87x
ImageNet	67.4%	66.6%	270	5760	21x
VOC	57.2%	59.2%	7	87	12x

END-TO-END PERFORMANCE



END-TO-END PERFORMANCE

- Tested three levels of optimization
  - None
  - Auto-caching only
  - Auto-caching and operator-selection.
- 7x to 15x speedup



QUESTIONS?

Project Page

http://keystone-ml.org/

Code

http://github.com/amplab/keystone

Training

http://goo.gl/axbkkc

BACKUP SLIDES

SOFTWARE FEATURES

- Data Loaders
  - CSV, CIFAR, ImageNet, VOC, TIMIT, 20 Newsgroups
- Transformers
  - NLP - Tokenization, n-gra
    parsing*
  - Images - Convolution, Gr
    FisherVector*, Pooling, W
  - Speech - MFCCs*
  - Stats - Random Features, Normalization, Scaling*, Signed Hellinger Mapping, FFT
  - Utility/misc - Caching, Top-K classifier, indicator label mapping, sparse/dense encoding transformers.
- Estimators
  - Learning - Block linear models, Linear Discriminant Analysis, PCA, ZCA Whitening, Naive Bayes*, GMM*

Just 11k Lines of Code,
5k of which are Tests or JavaDoc.

- Example Pipelines
  - NLP - Amazon Product Review Classification, 20 Newsgroups, Wikipedia
    el
  - , CIFAR, VOC, ImageNet
- Evaluation Metrics
  - Binary Classification
  - Multiclass Classification
  - Multilabel Classification

KEY API CONCEPTS

TRANSFORMERS

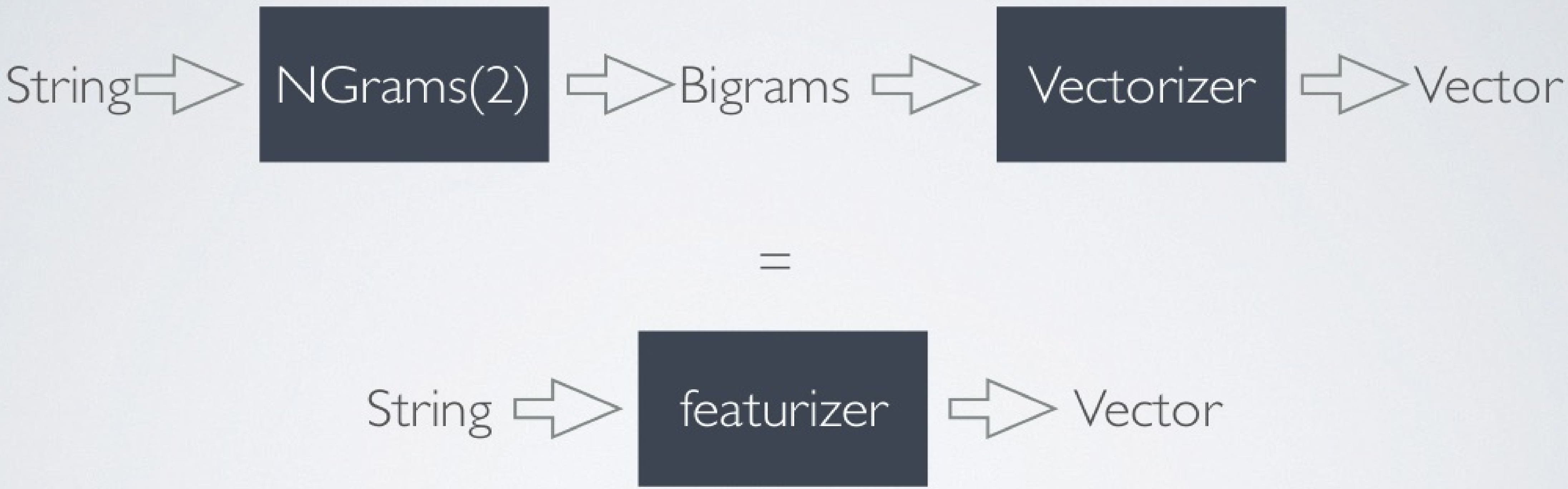

```
abstract class Transformer[In, Out] {
    def apply(in: In): Out
    def apply(in: RDD[In]): RDD[Out] = in.map(apply)
    ...
}
```

TYPE SAFETY HELPS ENSURE ROBUSTNESS

ESTIMATORS

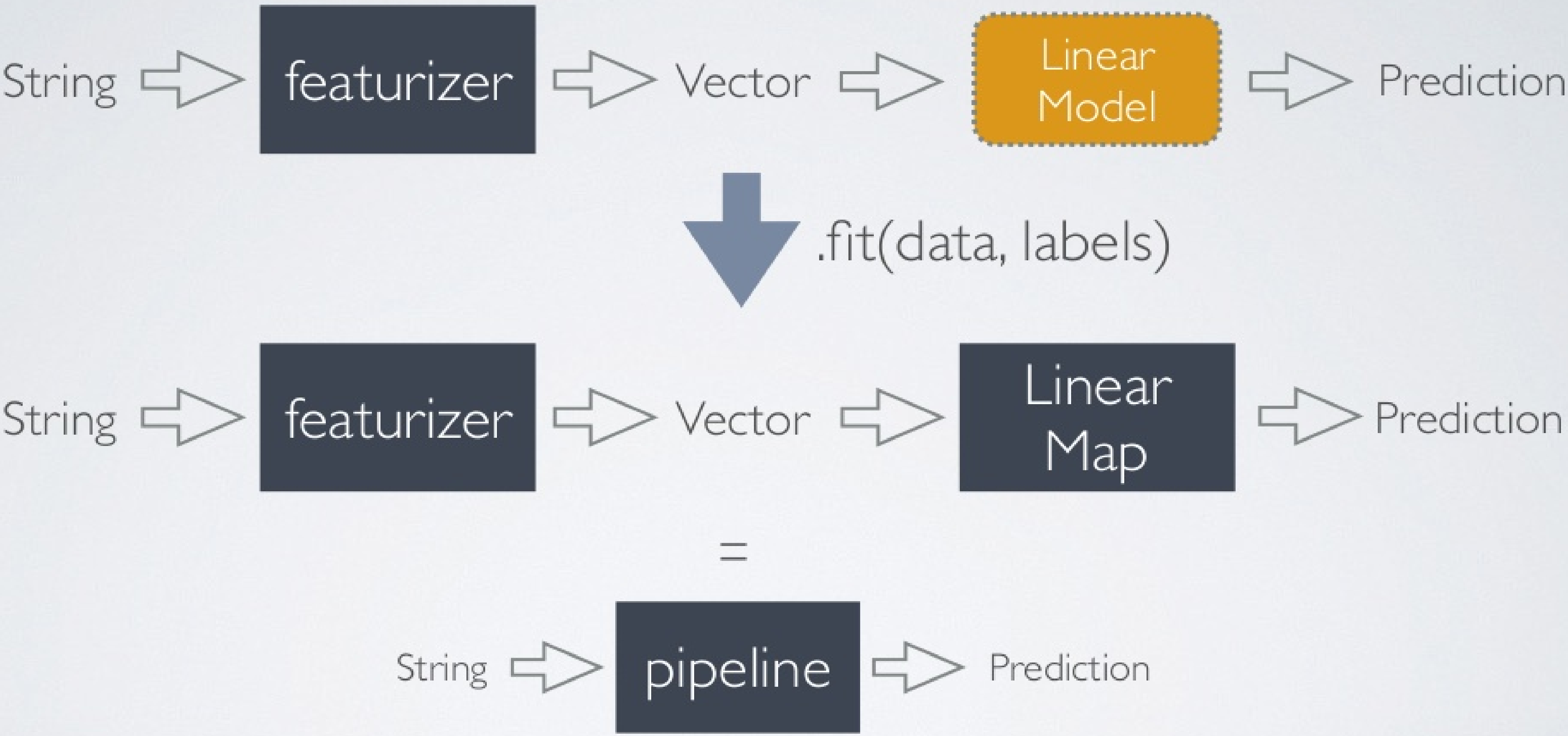

```
abstract class Estimator[In, Out] {
    def fit(in: RDD[In]):Transformer[In,Out]
    …
}
```

CHAINING



val featurizer: Transformer[String, Vector] = NGrams(2) ***then*** Vectorizer

COMPLEX PIPELINES




```
val pipeline = (featurizer thenLabelEstimator LinearModel).fit(data, labels)
```