

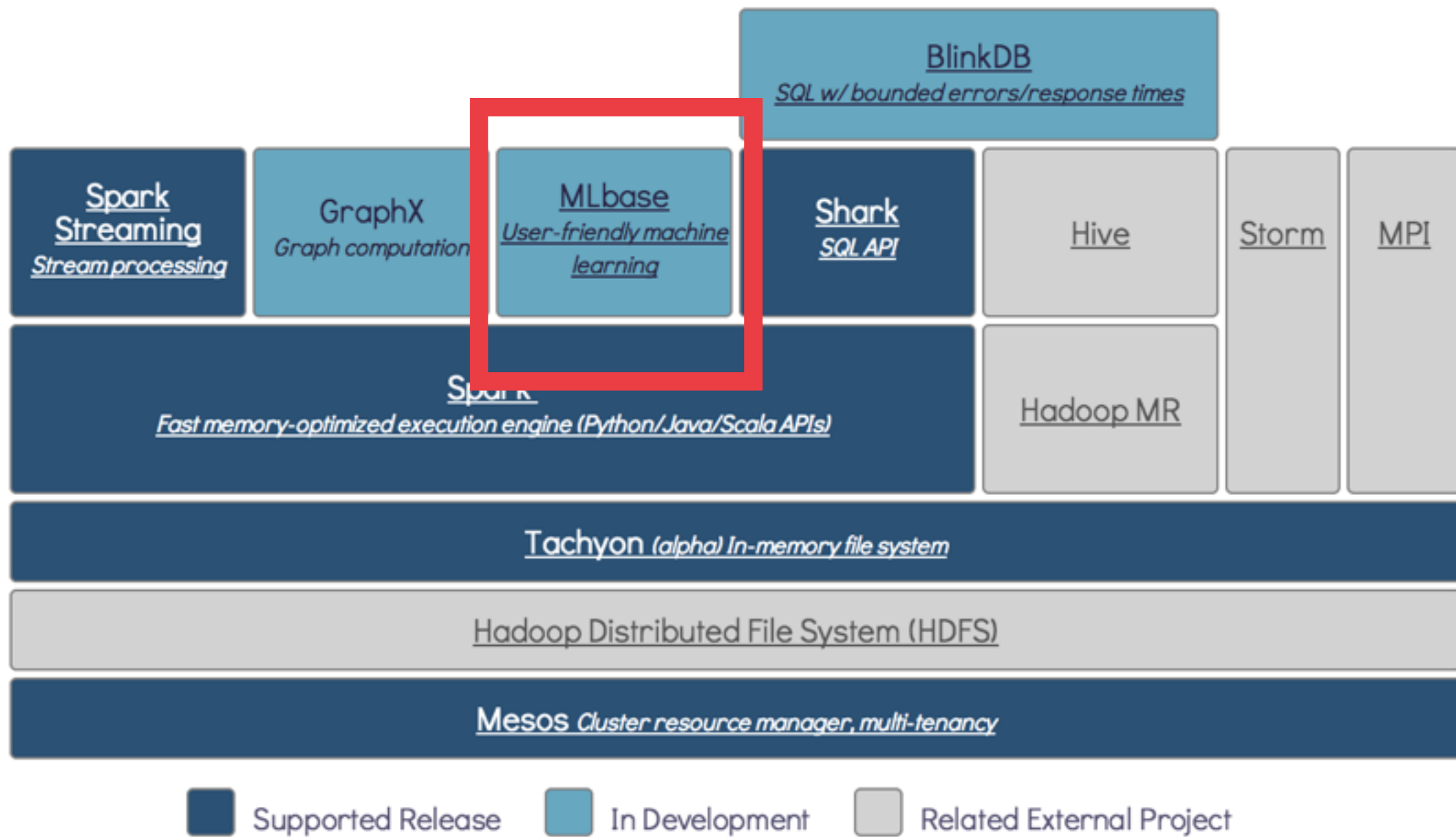
# **GHOSTFACE**

## TOWARDS AN OPTIMIZER FOR MLBASE

**Evan R. Sparks**, Ameet Talwalkar,  
Michael J. Franklin, Michael I. Jordan, Tim Kraska

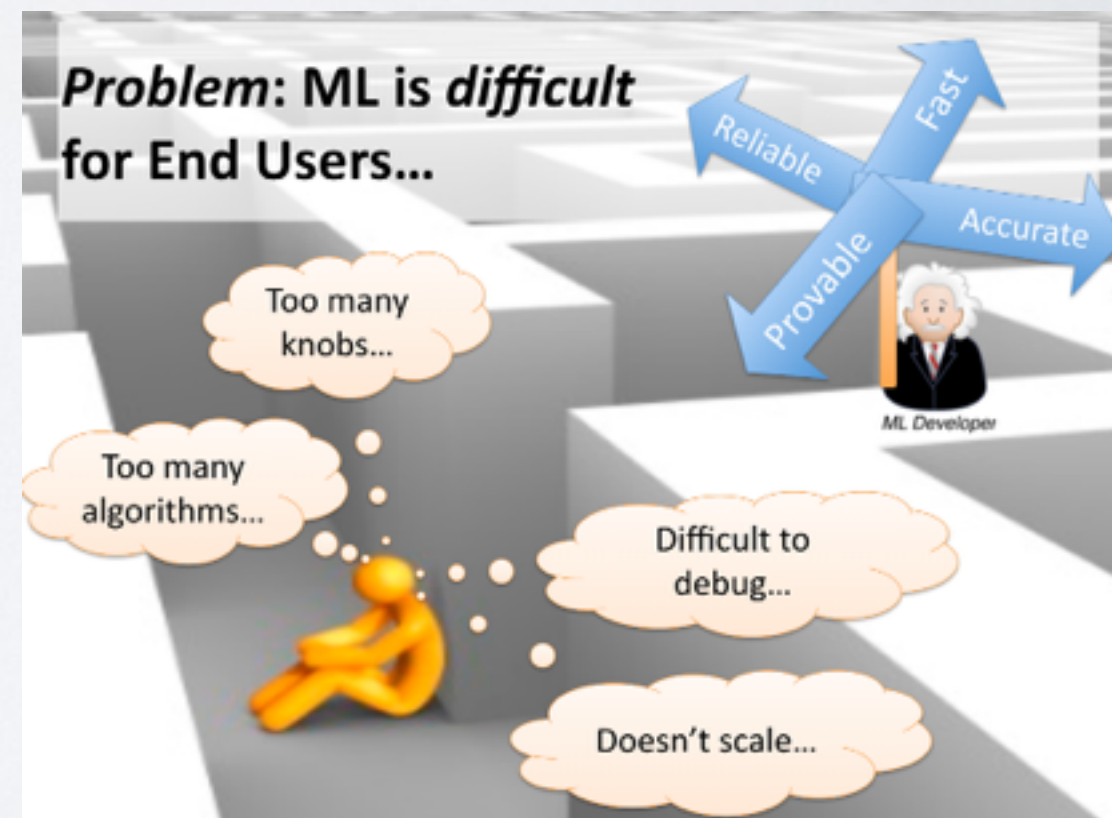
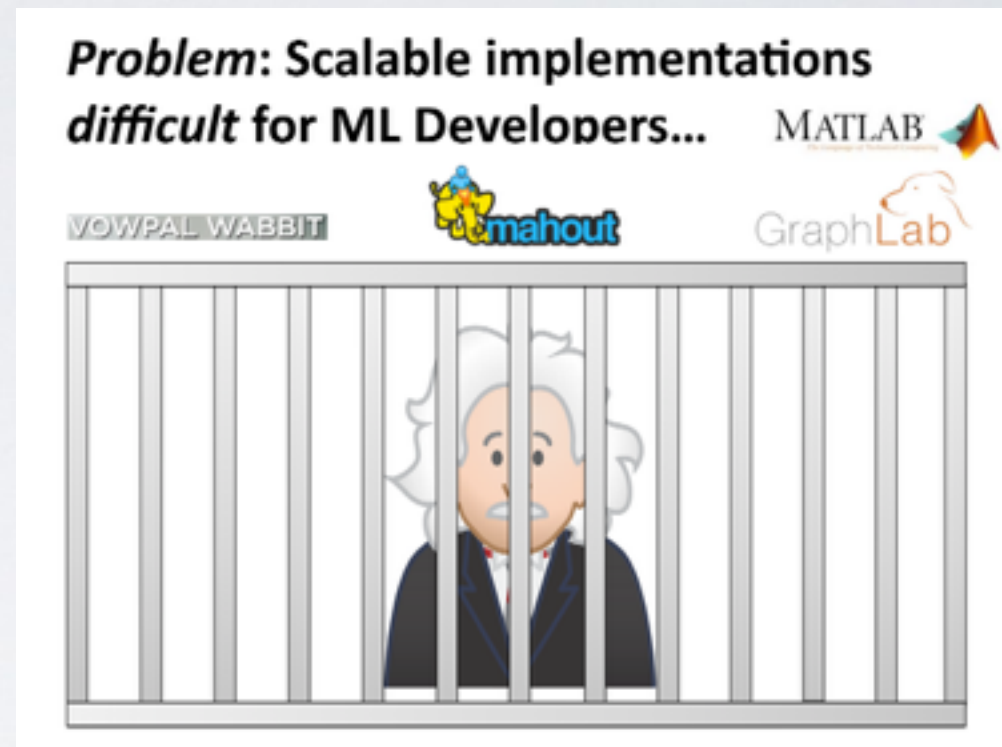
UC Berkeley

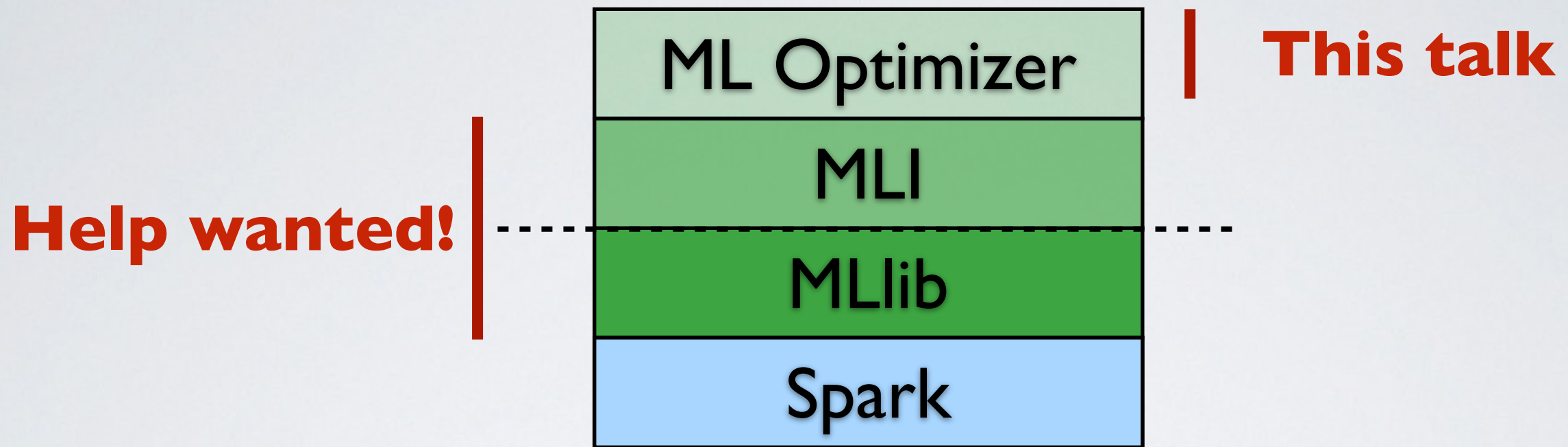
# THE BDAS STACK



# WHAT IS MLBASE?

- Distributed Machine Learning - Made Easy!
- Spark-based platform to simplify the development and usage of large scale machine learning.





# THE MLBASE STACK

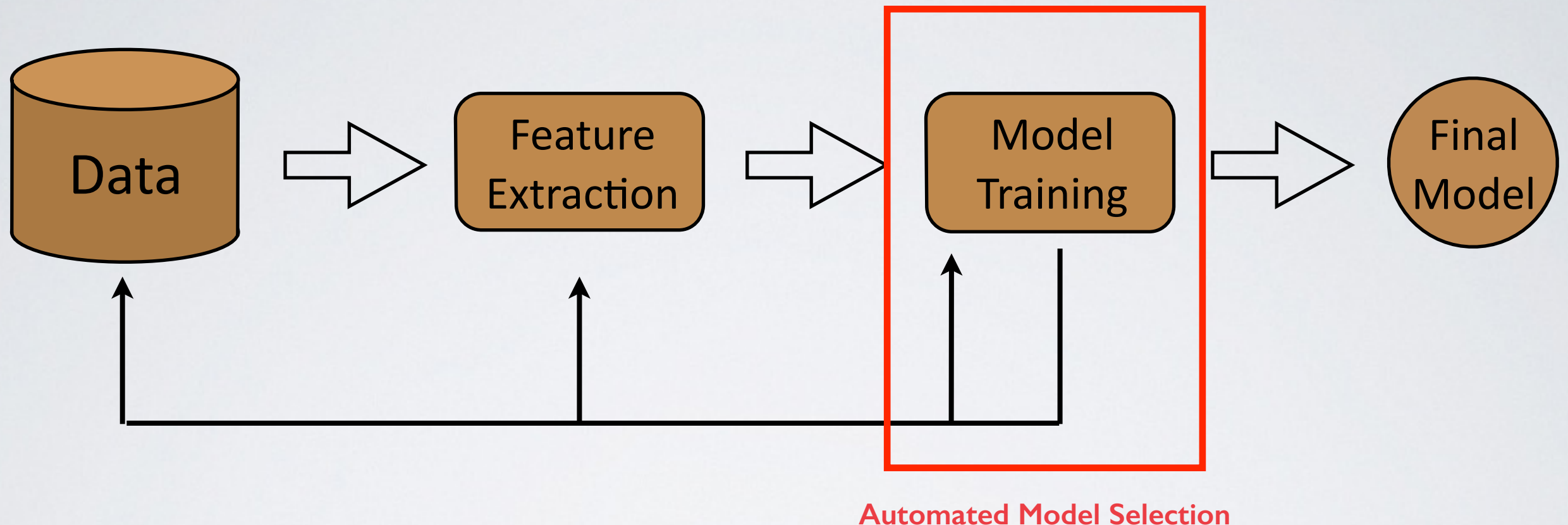
- Spark - Fast, distributed runtime.
- MLlib - Optimized library for standard ML functionality.
- MLI - Experimental API designed to simplify the implementation of new algorithms/feature extractors, etc.
- ML Optimizer - A declarative layer; simplified access to large-scale ML for end users.



# MLLIB/MLI UPDATE

- Released MLlib as a component of Spark 0.8.0
  - Thanks to Ameet Talwalkar, Shivaram Venkataraman, Xinghao Pan, Virginia Smith, Matei Zaharia, and others!
  - 30+ Contributors to MLlib in Spark 1.0
  - Continued development in the AMPLab.
- MLI is an experimental API for developing ML Algos (Sparks, et. al., ICDM 2013)
  - Continued development as research platform at Berkeley.
  - Ideas from MLI have made/making their way into MLlib/Spark
    - Model/Algorithm Abstraction (MLlib 0.8)
    - Matrices as first class citizen, Sparse/Dense Support (MLlib 1.0)
    - Relational Tables (SparkSQL 1.0)
    - Standardized Interfaces, Parallel Model Training (MLlib 1.1)

# A SIMPLE ML PIPELINE

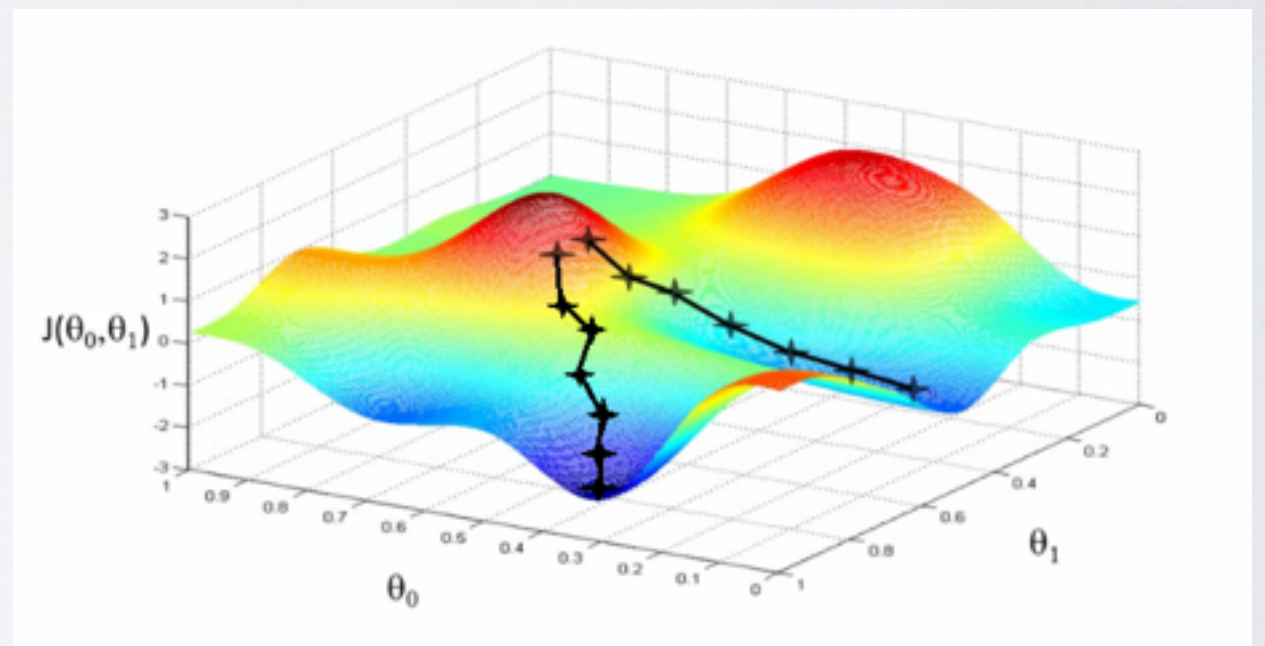


- In practice, model building is an iterative process of continuous refinement.
- Our grand vision is to automate the construction of these pipelines.
- Start with one aspect of the pipeline - model selection.

# TRAINING A MODEL

- For each point in my dataset, compute some delta, update my model.
  - Repeat until converged.
- In practice - requires **multiple passes over the data**.
- From a systems perspective - this is the access pattern. Same pattern holds for lots of algorithms (Naive Bayes, Trees, etc.)
- Minutes to train an SVM on 200GB of data on a 16-node cluster.

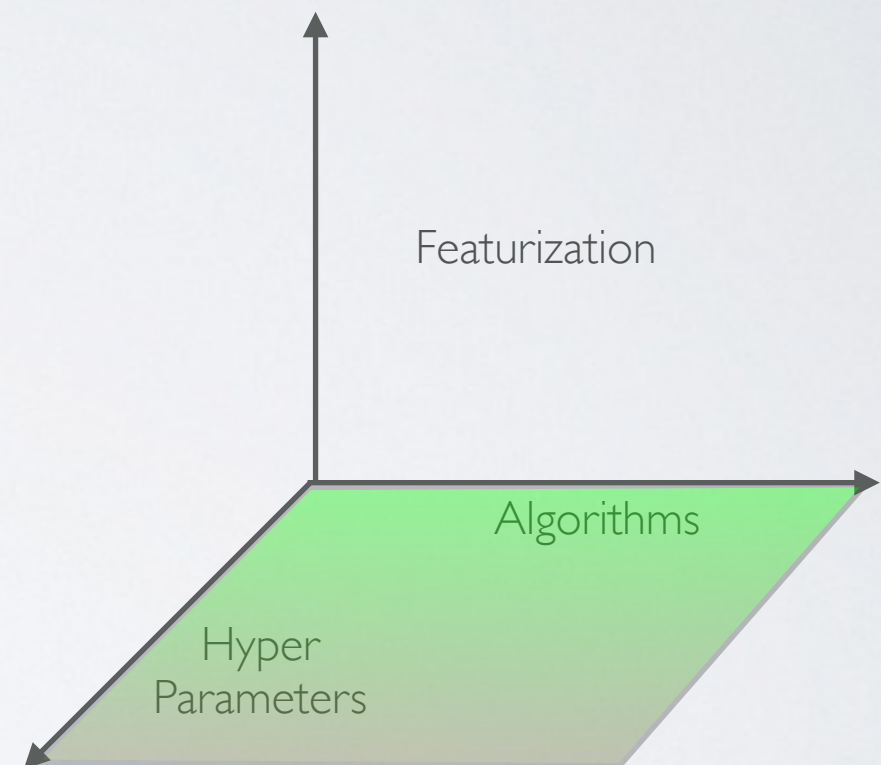
$$w := w - \alpha \nabla Q(w) = w - \alpha \sum_{i=1}^n \nabla Q_i(w),$$





# THE TRICKY PART

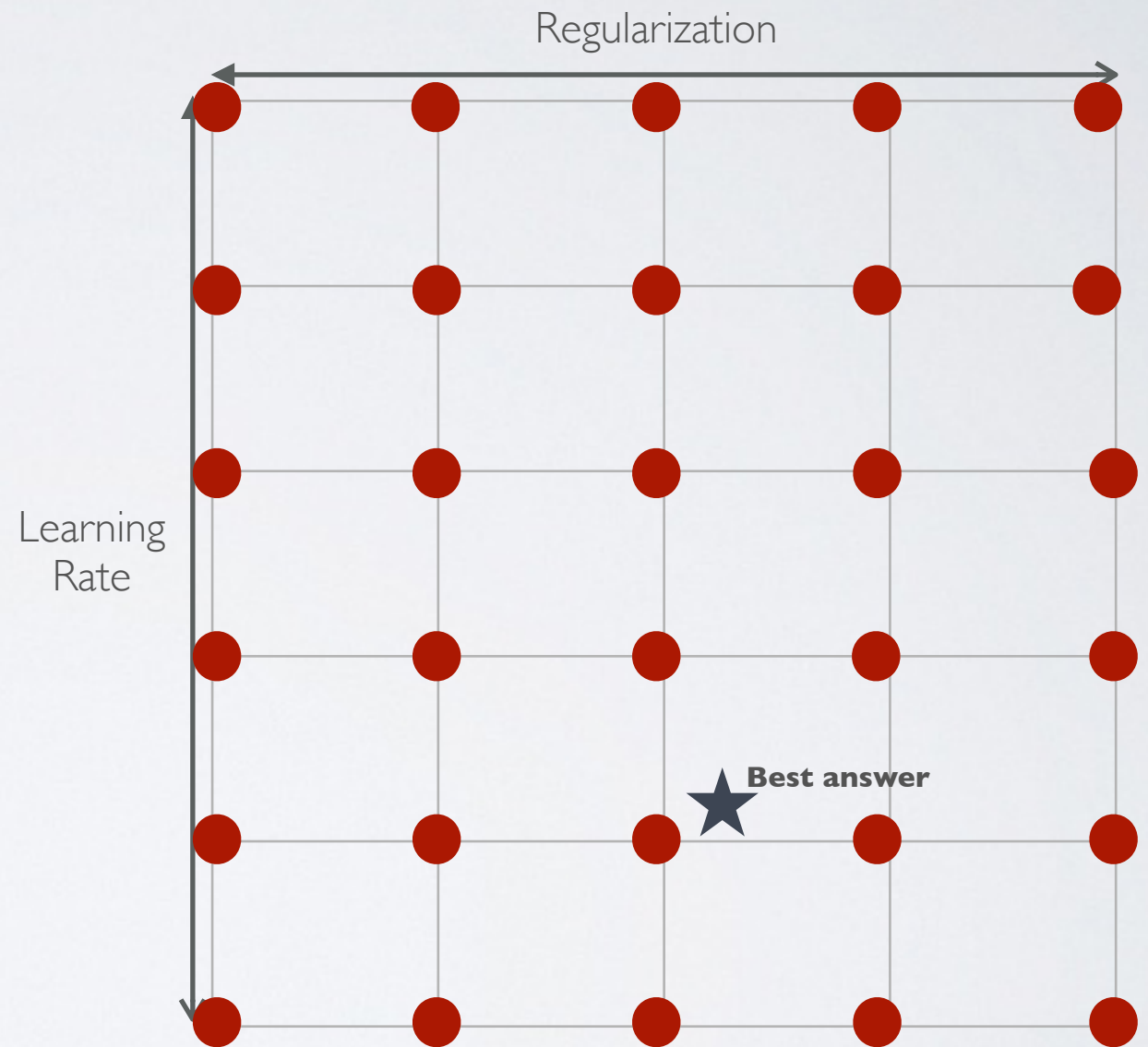
- Algorithms
  - Logistic Regression, SVM, Tree-based, etc.
- Algorithm hyper-parameters
  - Learning Rate, Regularization, Perturbations
- Featurization
  - Text - n-grams, TF-IDF
  - Images - Gabor filters, random convolutions
  - Random projection? Scaling?





# ONE APPROACH

- TRY IT ALL!
  - Grid search every combination of parameters, algorithms, features, etc.
- Drawbacks
  - Models are expensive to compute.
  - Parameter space is huge.
- Some version of this still often done in practice!



# A BETTER APPROACH

- Better resource utilization
  - Through batching
- Algorithmic Speedups
  - Via Early Stopping
- Improved Search



# A TALE OF 3 OPTIMIZATIONS

- **Better Resource Utilization**
- Algorithmic Speedups
- Improved Search



# BETTER RESOURCE UTILIZATION


- Modern memory speeds slower than modern processor speeds:
  - Can read: 0.6b doubles/sec/core. (4.75 GB/s)
  - Can compute: 15b flops/sec/core
  - Means we can do 25 flops/double read.





# WHAT DOES THIS MEAN FOR MODELING?

- Typical model update requires 2-4 flops/double.
- Can do 7-10 model updates in the same amount of time we can do 1 by using otherwise idle cycles.
- Assuming that models are relatively small - fit in cache.
- Train multiple models simultaneously.



A	B	C
1	a	Dog
1	b	Cat
2	c	Cat
2	d	Cat
3	e	Dog
3	f	Horse
4	g	Doge

# WHAT DO WE SEE IN MLI?

- See something between a 2x and 5x increase in models trained/sec when introducing this optimization.
- Overhead from virtualization, network, unboxing, etc.

Batch Size \ D	D	100	500	1000	10000
	1	136.80	114.71	98.14	22.18
	2	260.64	171.79	147.79	30.59
	5	554.32	290.46	189.87	30.19
	10	726.77	390.04	232.23	25.31

(a) Models trained per hour for varying batch sizes and model complexity.

Batch Size \ D	D	100	500	1000	10000
	1	1.00	1.00	1.00	1.00
	2	1.91	1.50	1.51	1.38
	5	4.05	2.53	1.93	1.36
	10	5.31	3.40	2.37	1.14

(b) Speedup factor vs baseline for varying batch size and model complexity.

Figure 5: Effect of batching is examined on 16 nodes with a synthetic dataset. Speedups diminish but remain significant as models increase in complexity.

# BATCH MATRIX VERSION

- Previous numbers - vector/matrix multiplies.
- These - rederived in terms of Matrix-Matrix multiplies.
- Better performance as models get more complicated.
- Still some stuff to do!

Batch Size \ D	D	100	500	1000	10000
1	1	116.29	29.80	24.11	2.12
2	2	112.82	45.01	22.12	2.71
5	5	311.05	87.97	54.54	6.74
10	10	527.50	219.30	154.09	11.42

(a) Models trained per hour for varying batch sizes and model complexity.

Batch Size \ D	D	100	500	1000	10000
1	1	1.00	1.00	1.00	1.00
2	2	0.97	1.51	0.92	1.28
5	5	2.67	2.95	2.26	3.18
10	10	4.54	7.36	6.39	5.40

(b) Speedup factor vs baseline for varying batch size and model complexity.

Figure 5: Effect of batching is examined on 16 nodes with a synthetic dataset. Speedups diminish but remain significant as models increase in complexity.



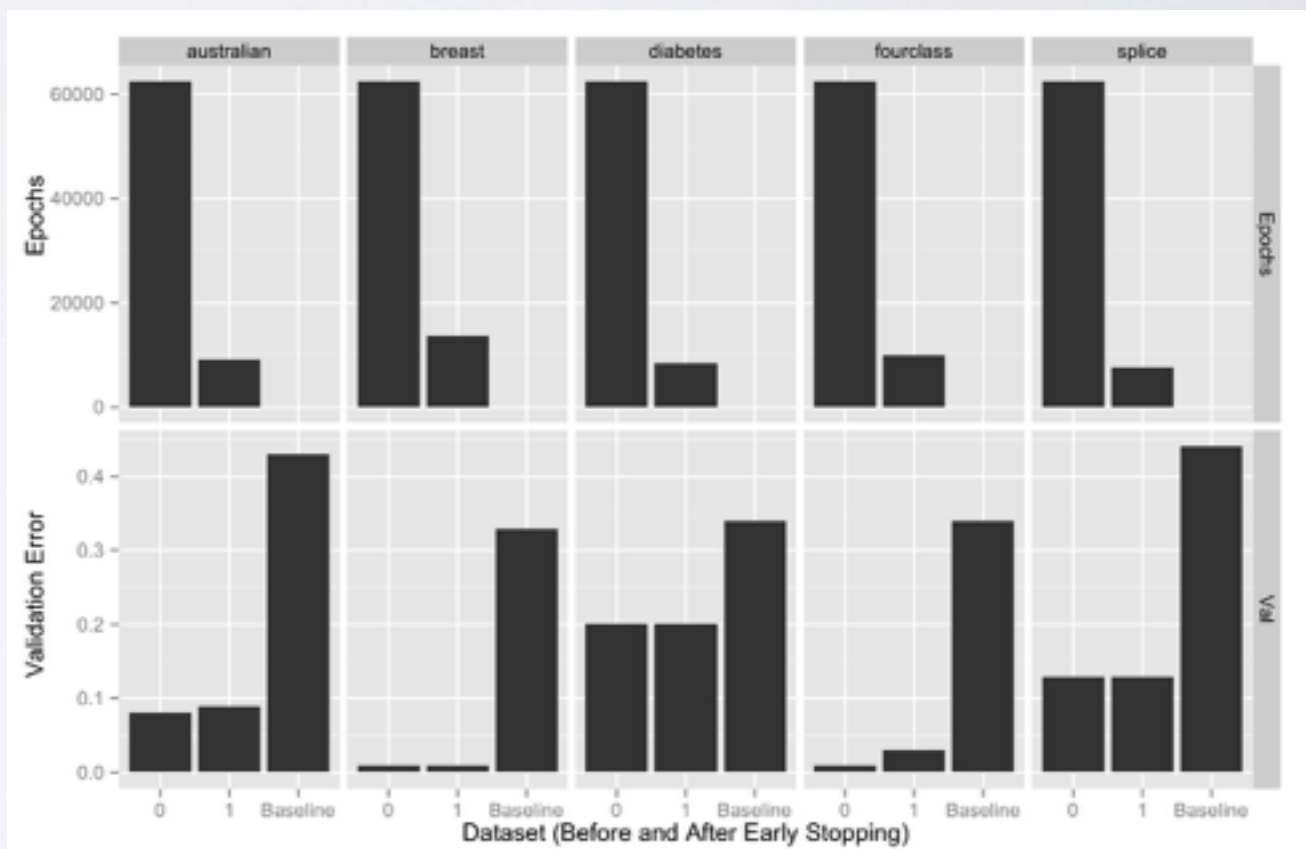
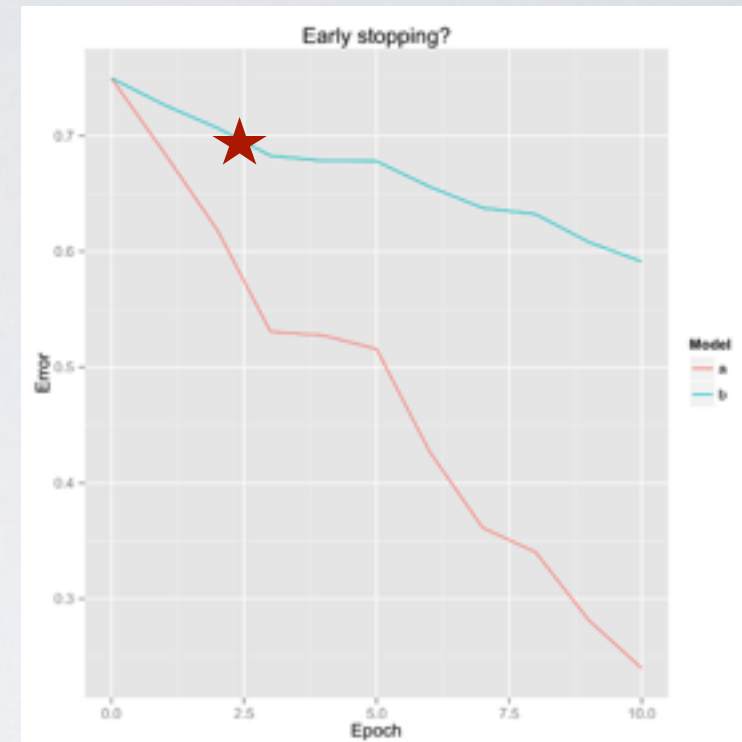
# A TALE OF 3 OPTIMIZATIONS

- Improved Search
- **Algorithmic Speedups**
- Better Resource Utilization



# ALGORITHMIC SPEEDUPS

- Each of the points in our hyperparameter space represents training a full model.
- But we sometimes can be pretty sure that the model isn't going to be helpful.
- So we stop early - models that are junk after one pass over the data stop early.
- Other things we can do here - quasi-Newton methods (Adagrad, LBFGS) speed up convergence without too much computational complexity.

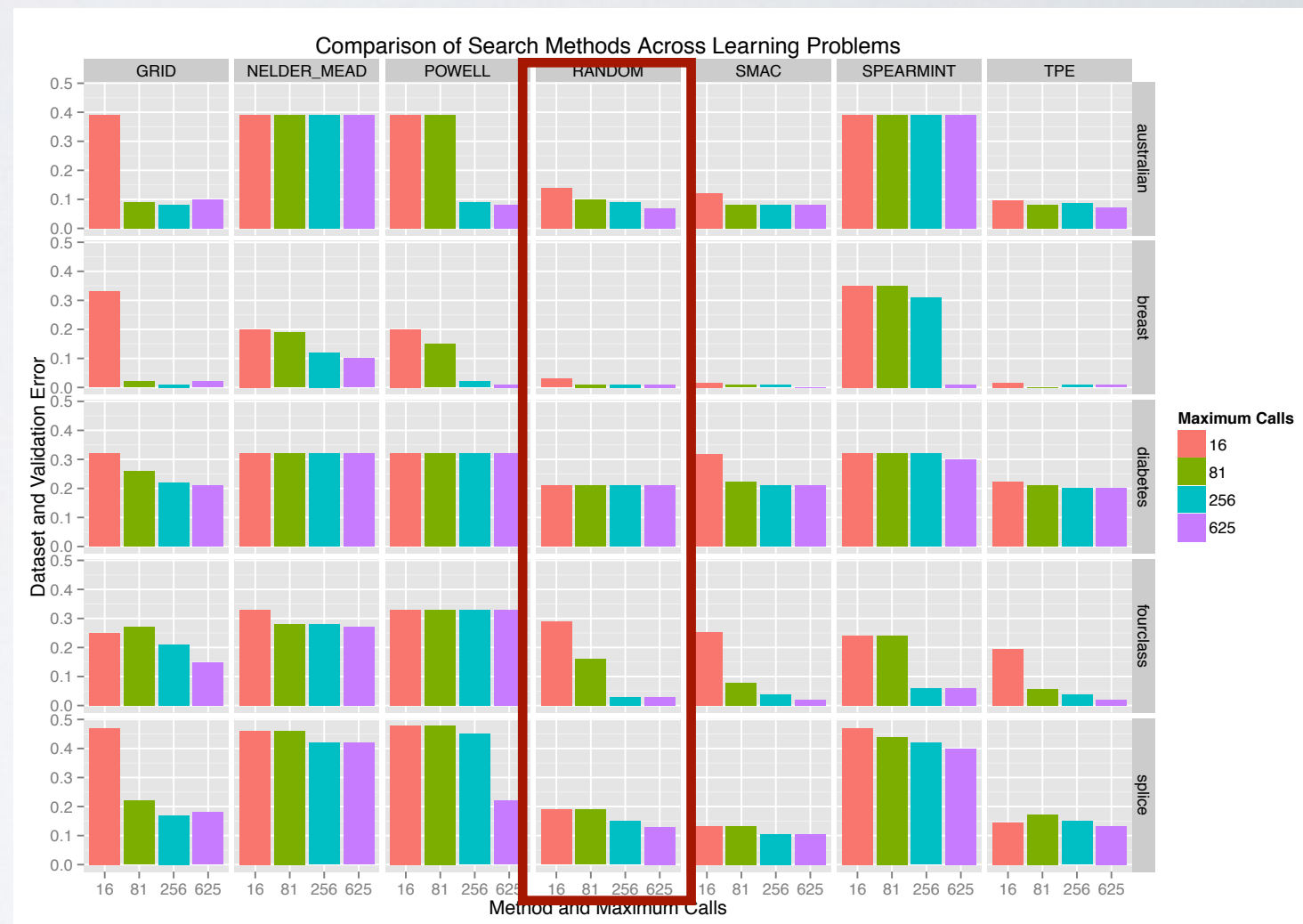


# A TALE OF 3 OPTIMIZATIONS

- Better Resource Utilization
- Algorithmic Speedups
- **Improved Search**

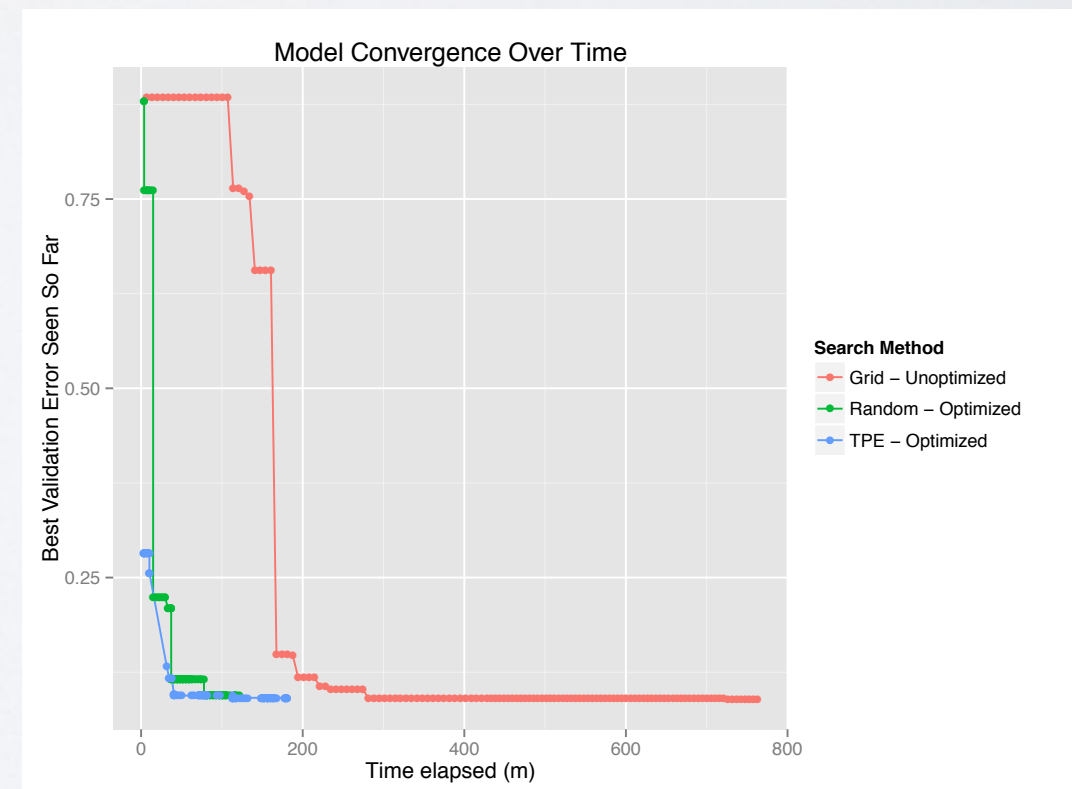
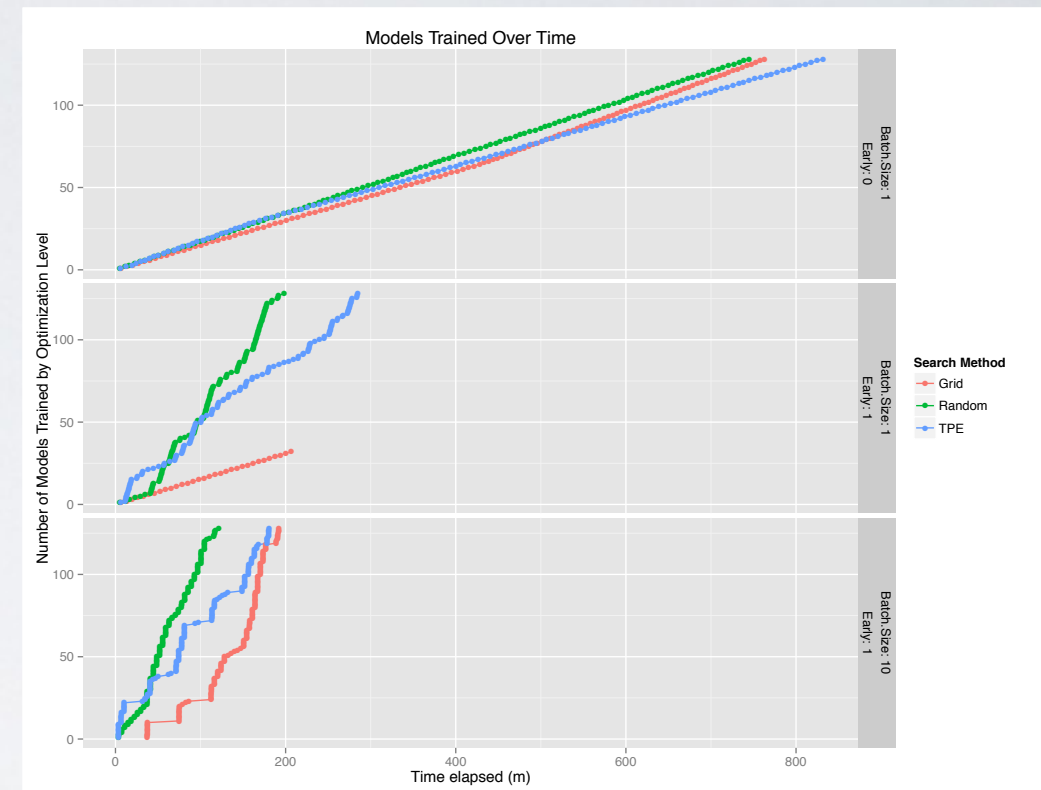
# WHAT SEARCH METHOD SHOULD WE USE?

- Derivative Free Optimization Techniques
  - Grid
  - Random
  - Nelder-Mead
  - Powell's method
  - Bayesian
    - Spearmint, SMAC, TPE
- What should we do?
  - Tried on 5 datasets, optimized over 4 parameters, check what works!



# PUTTING IT ALL TOGETHER

- The first version of the MLbase optimizer
- Given 30GB dense image features (240k x 16k)
- Binary classification (plants vs. non plants)
- Consider two model families
  - 3 hyperparameters for each
  - Learning Rate, Regularization, Model family
- Naive method - sequential grid search - no early stopping.
- More sophisticated - batch random search with early stopping.

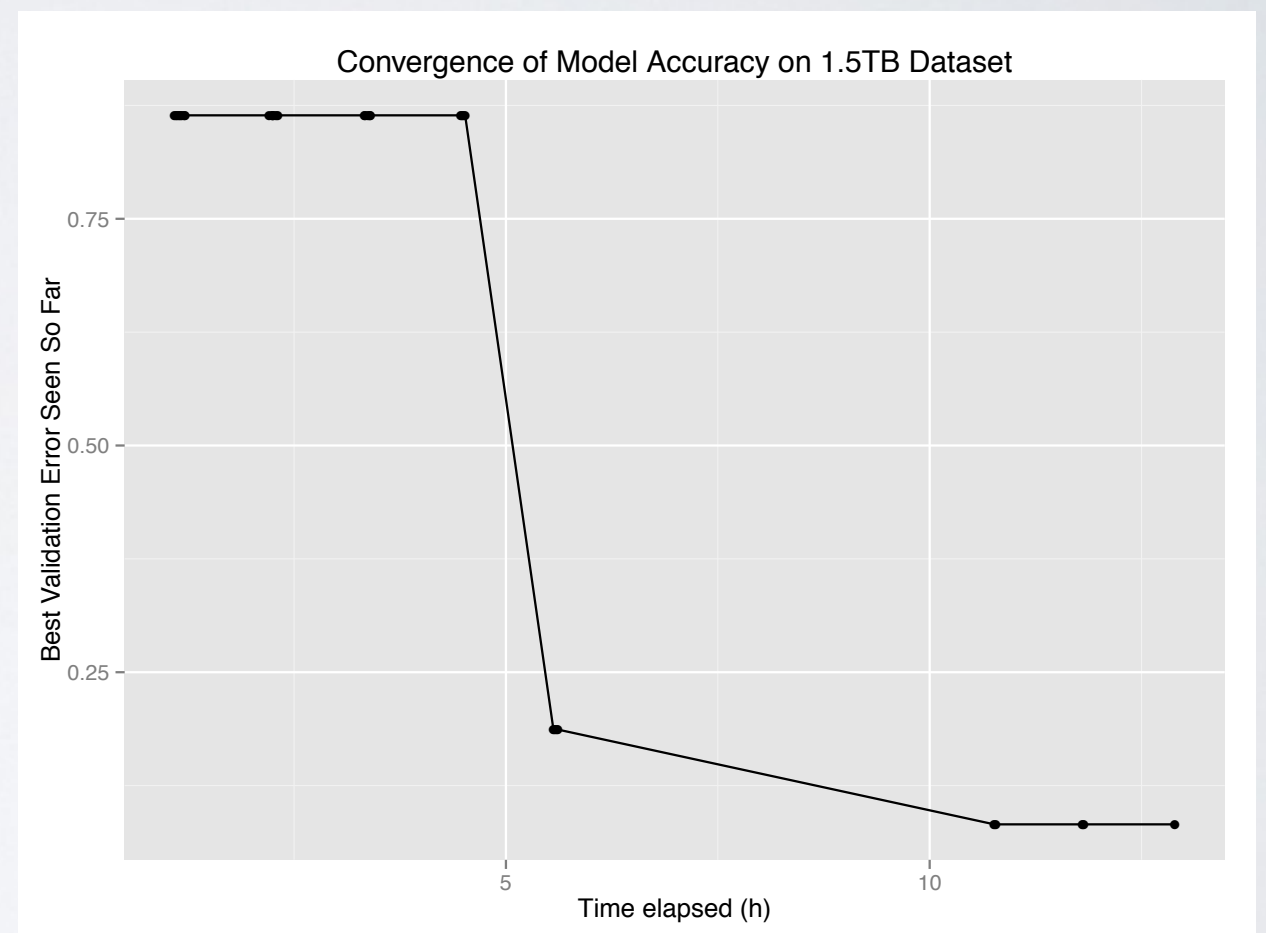




# DOES IT SCALE?

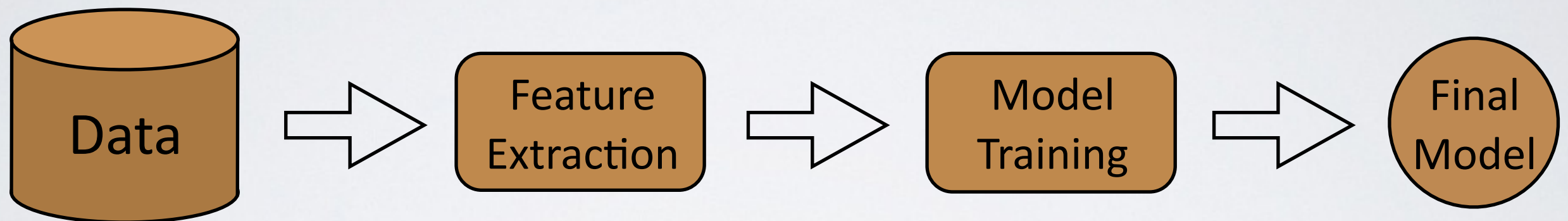
1.5 TB dataset (1.2m x 160k),  
128 nodes, thousands of  
passes over the data.

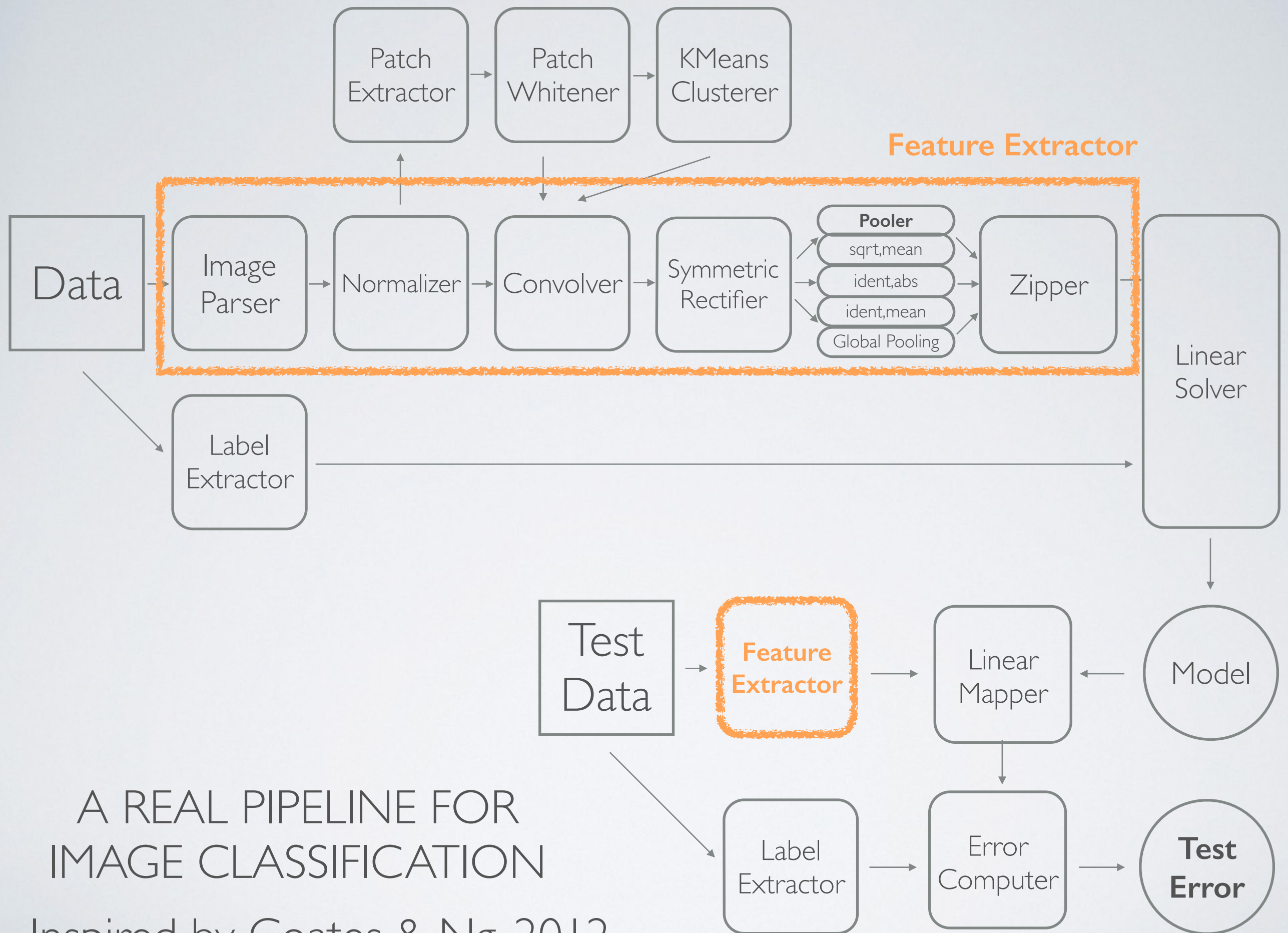
Tried 32 models in 15 hours.  
Good answer after 11.



# REAL WORLD PIPELINES

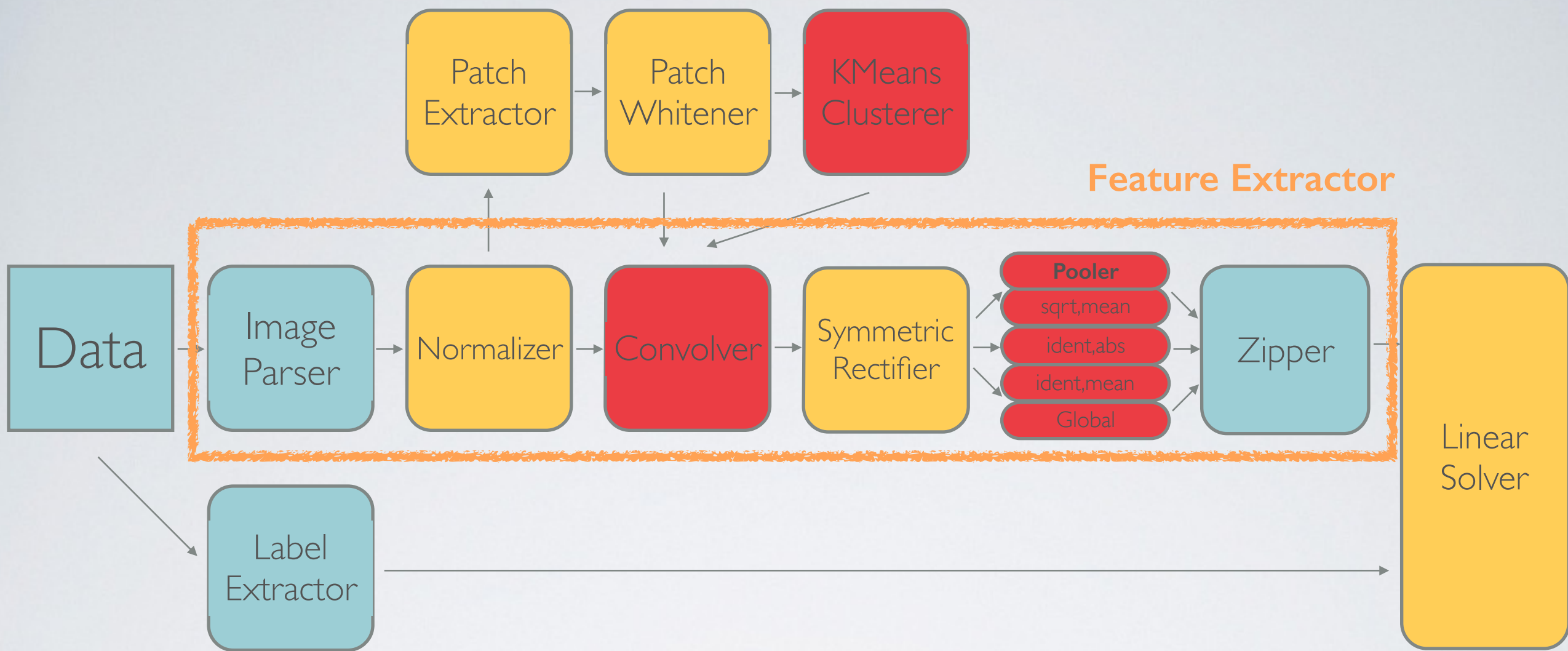
# A SIMPLE ML PIPELINE



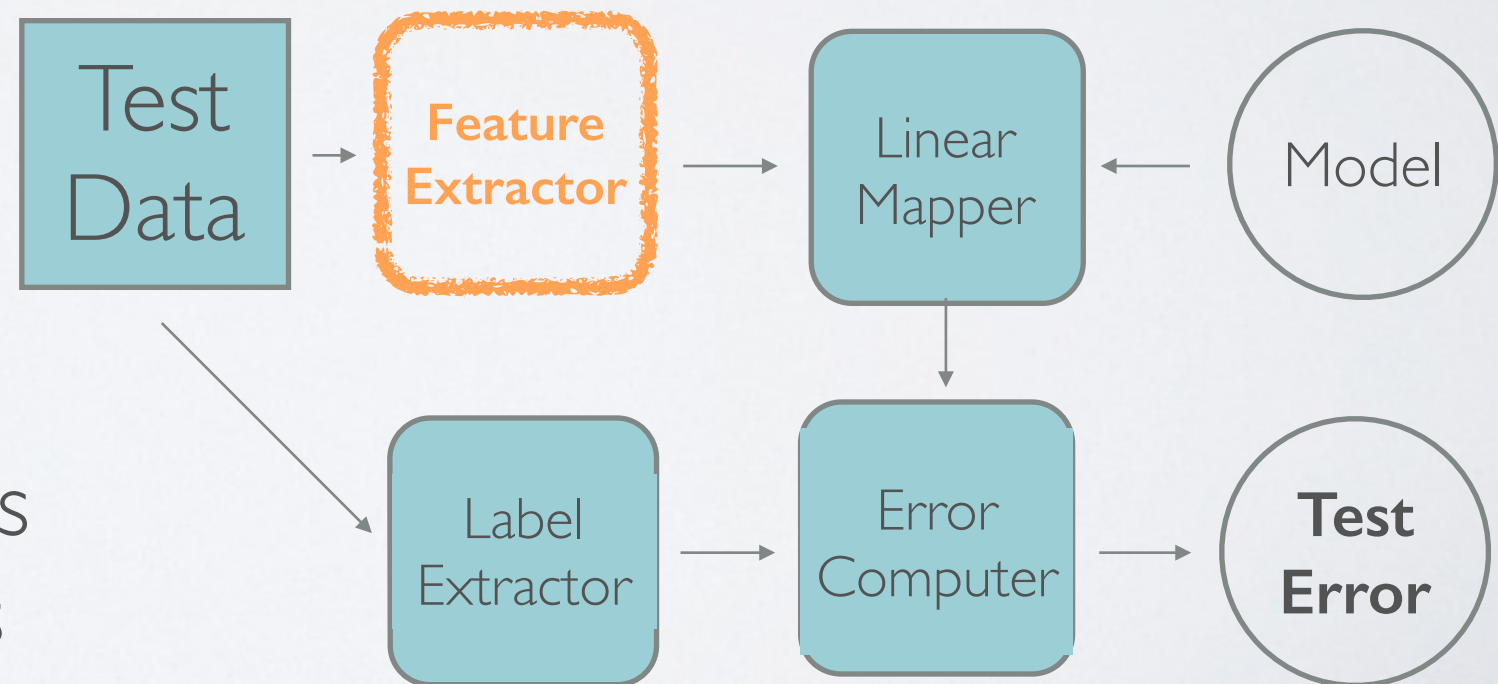


A REAL PIPELINE FOR  
IMAGE CLASSIFICATION  
Inspired by Coates & Ng, 2012





- No Hyperparameters
- A few Hyperparameters
- Lotsa Hyperparameters



# FUTURE WORK

- Choosing between model families (bandits?)
- Ensembling
- Leverage sampling
- Faster learning methods (ADAGRAD, L-BFGS)
- Better parallelism for smaller datasets
- Incorporating feature extraction into pipeline
- Multiple hypothesis testing issues

QUESTIONS?