THE MISSING PIECE IN COMPLEX ANALYTICS: *SCALABLE, LOW LATENCY MODEL SERVING AND MANAGEMENT WITH VELOX*

Daniel Crankshaw, **Peter Bailis**, **Joseph Gonzalez**, Haoyuan Li,
Zhao Zhang, Ali Ghodsi, Michael Franklin, and Michael I. Jordan
UC Berkeley AMPLab

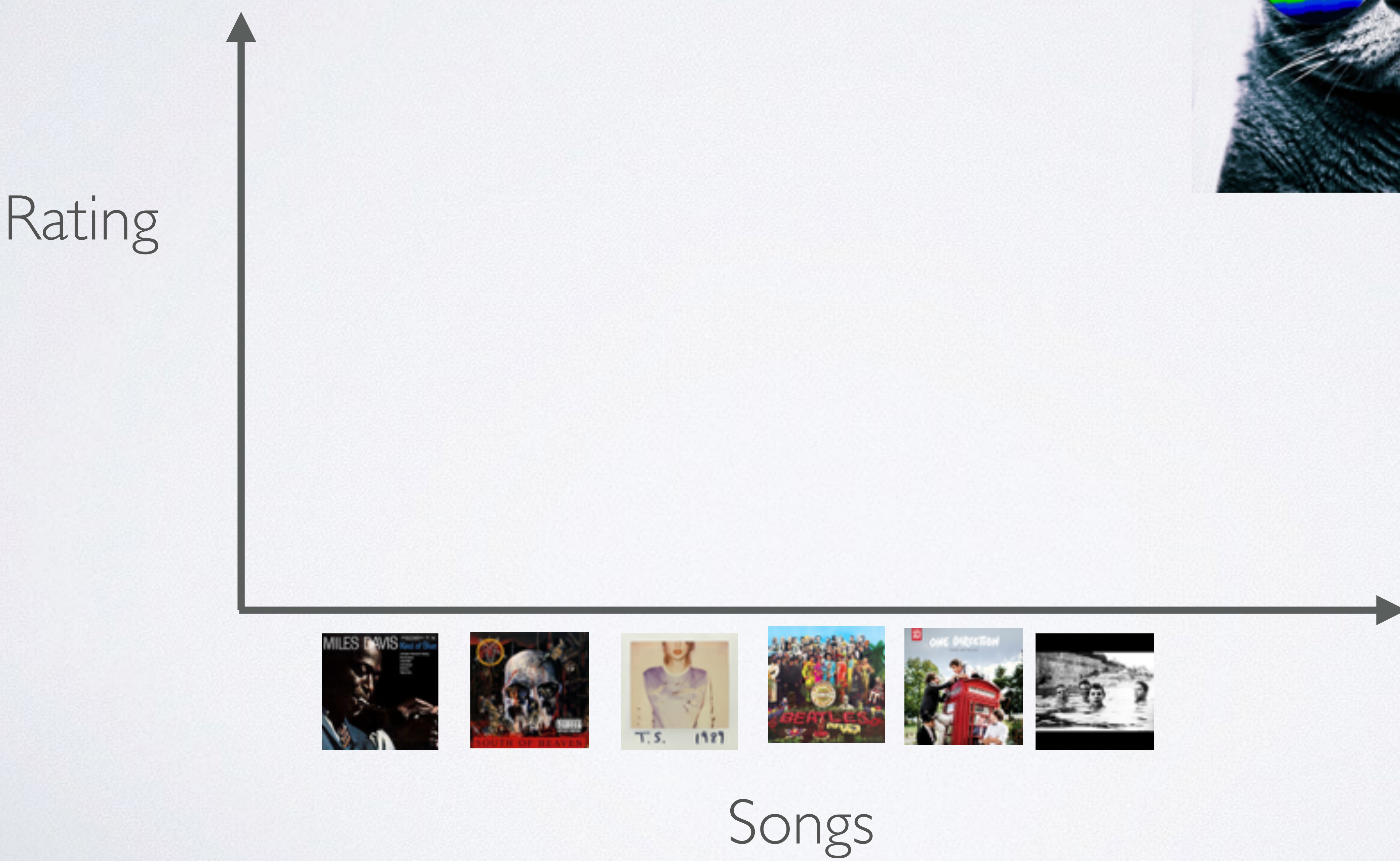
CIDR 2015

Talk Outline

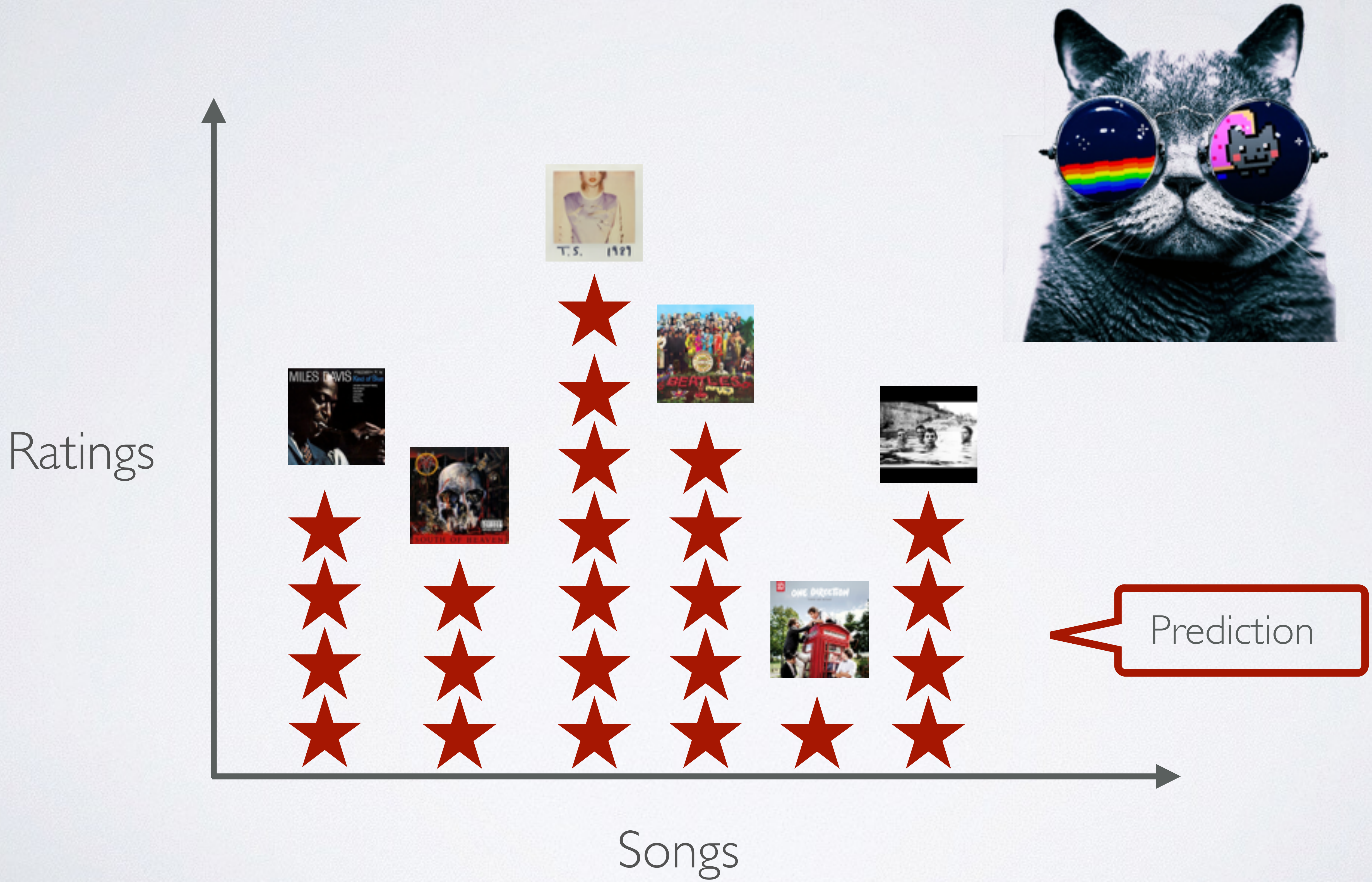
- **ML model management today**
- Velox system architecture
- Key idea: Split model family
- Prediction serving
- Model management
- Next directions

Catify: Music for Cats

MODELING TASK

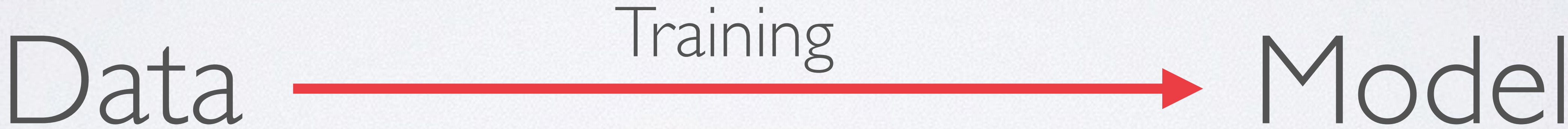


MODELING TASK

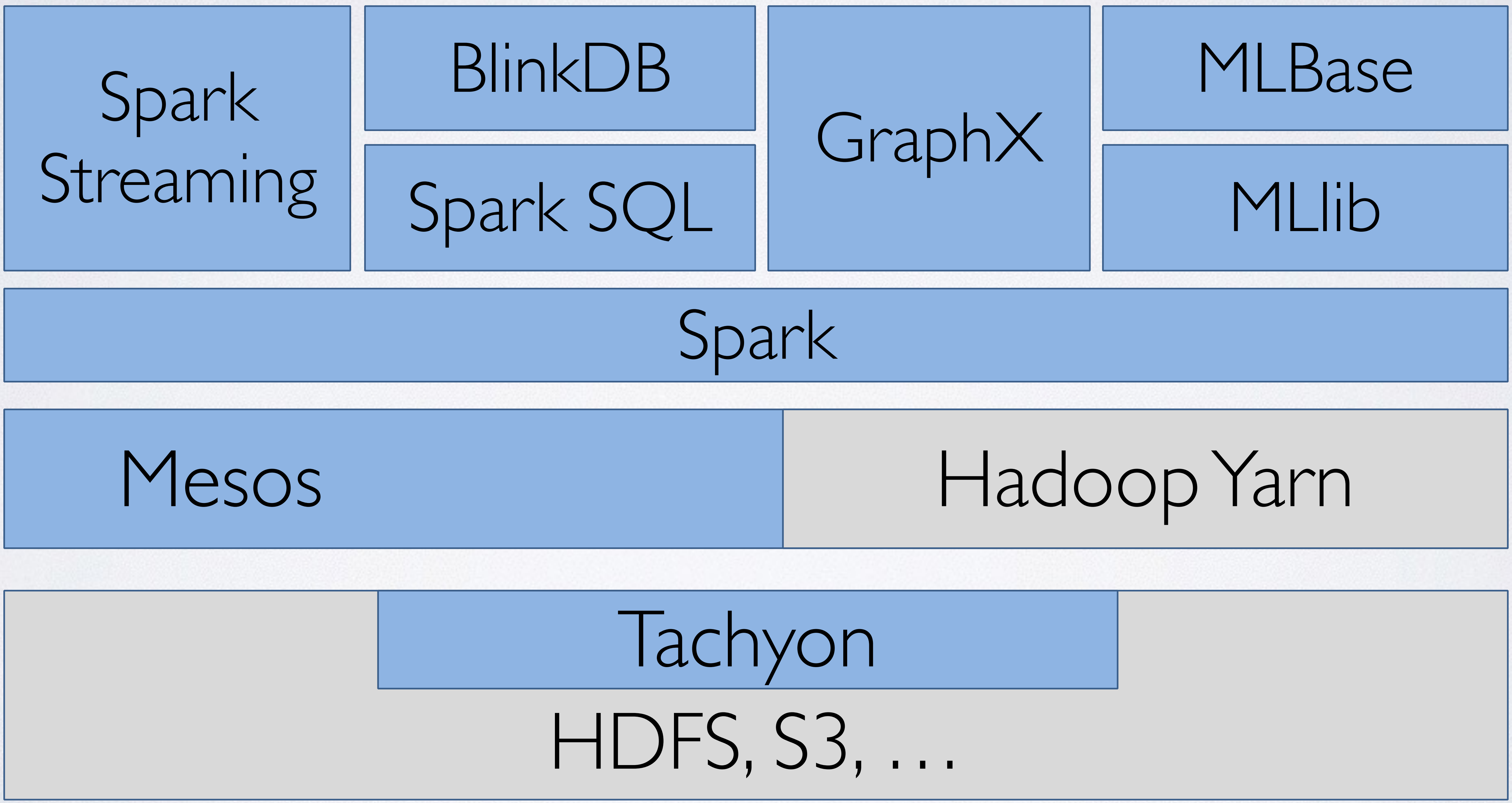


Data

Data → Model


Data
Training
Model

BERKELEY DATA ANALYTICS STACK (BDAS)



Catify: Music for Cats

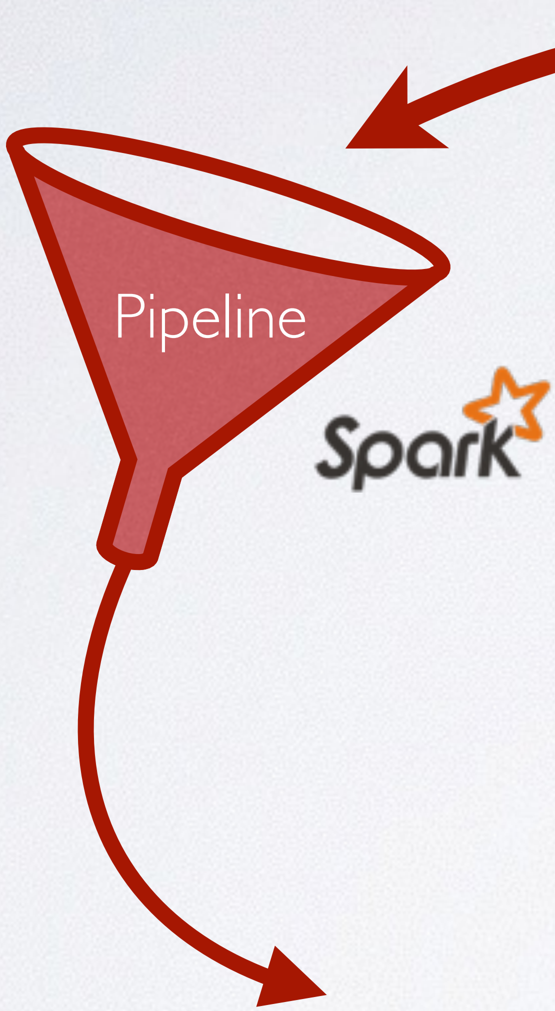
Catify: Music for Cats

CatID	Song	Score
1	16	2.1
1	14	3.7
3	273	4.2
4	14	1.9

Catify: Music for Cats

CatID	Song	Score
1	16	2.1
1	14	3.7
3	273	4.2
4	14	1.9

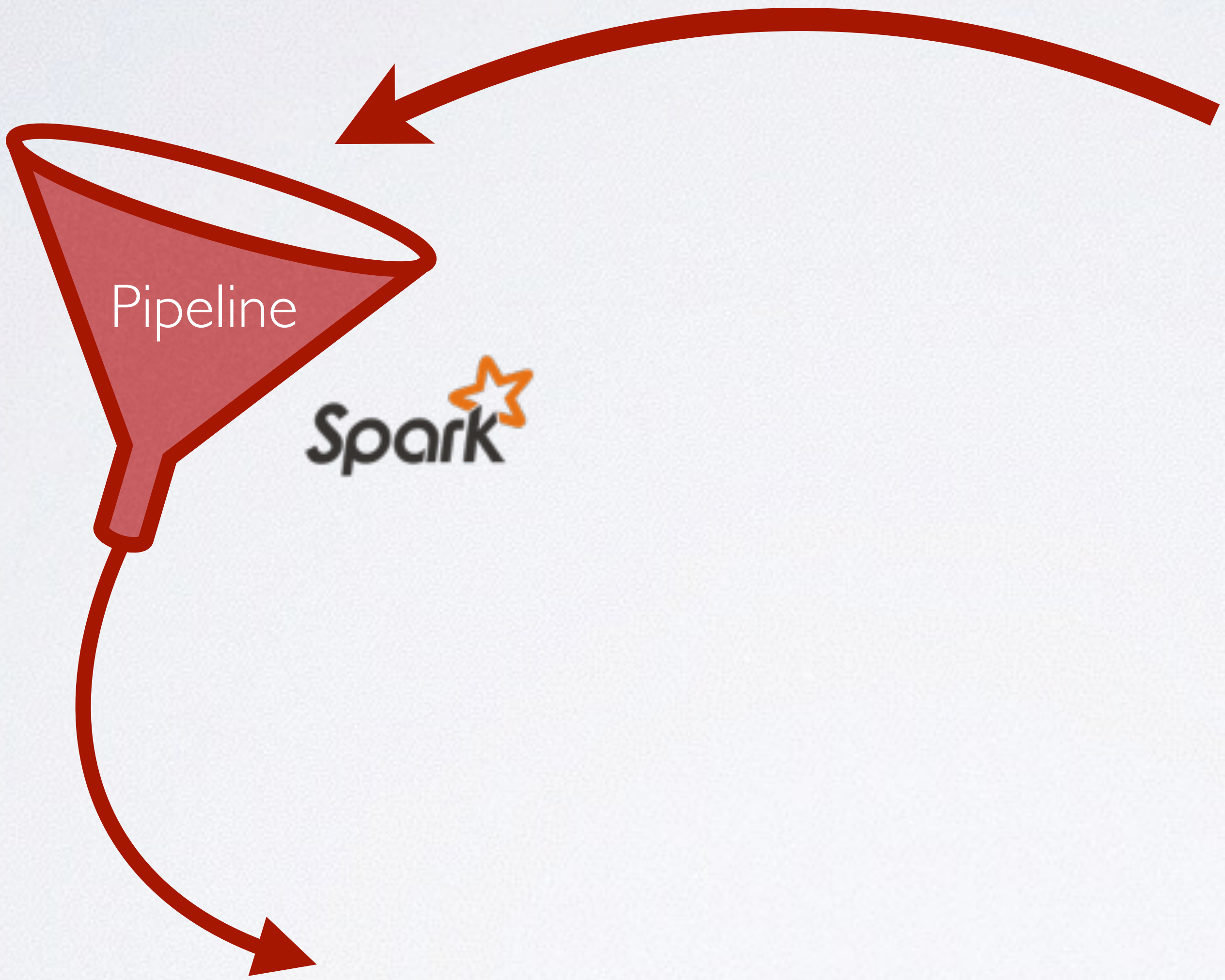
Catify: Music for Cats



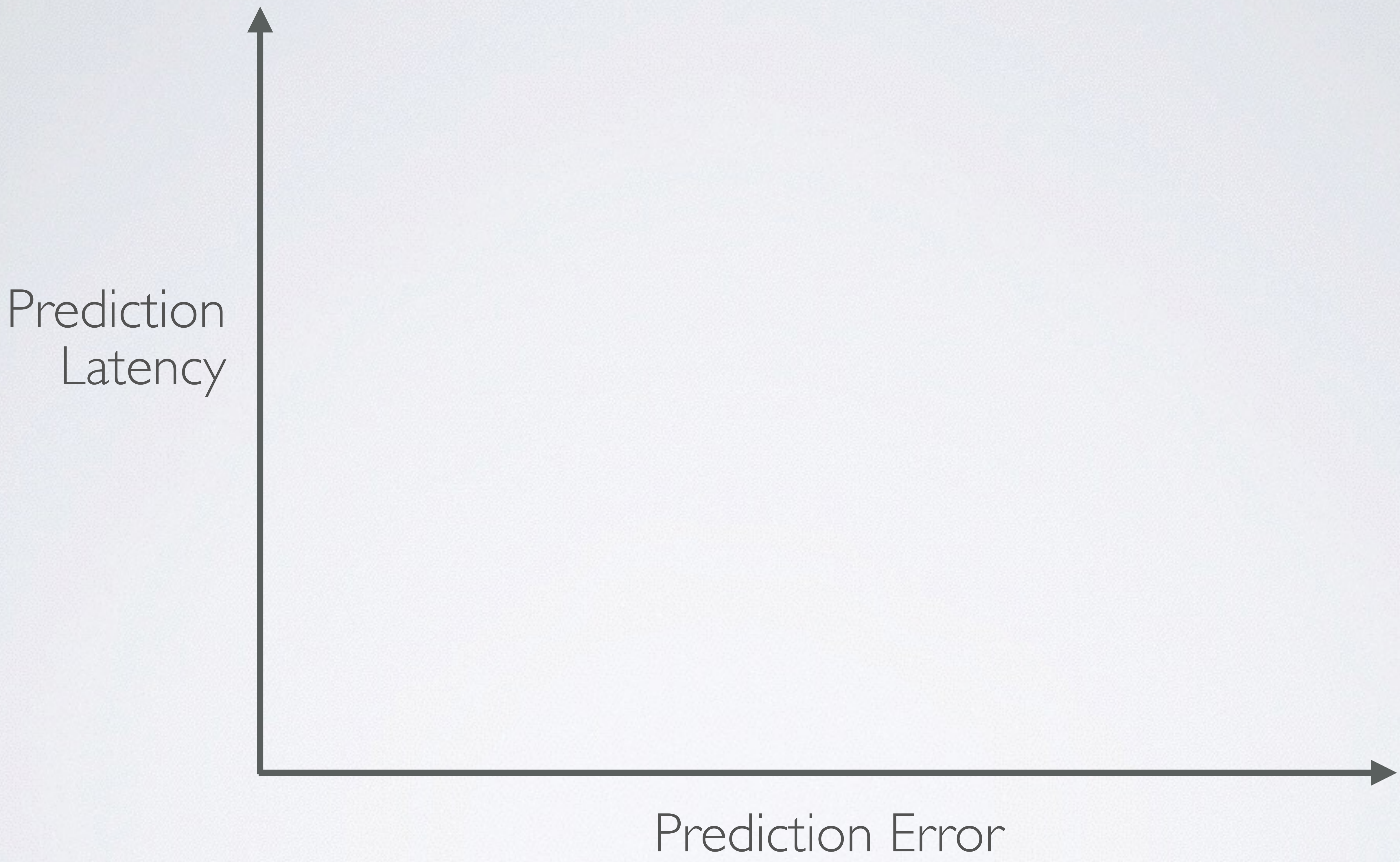
CatID	Song	Score
1	16	2.1
1	14	3.7
3	273	4.2
4	14	1.9

Tachyon + HDFS

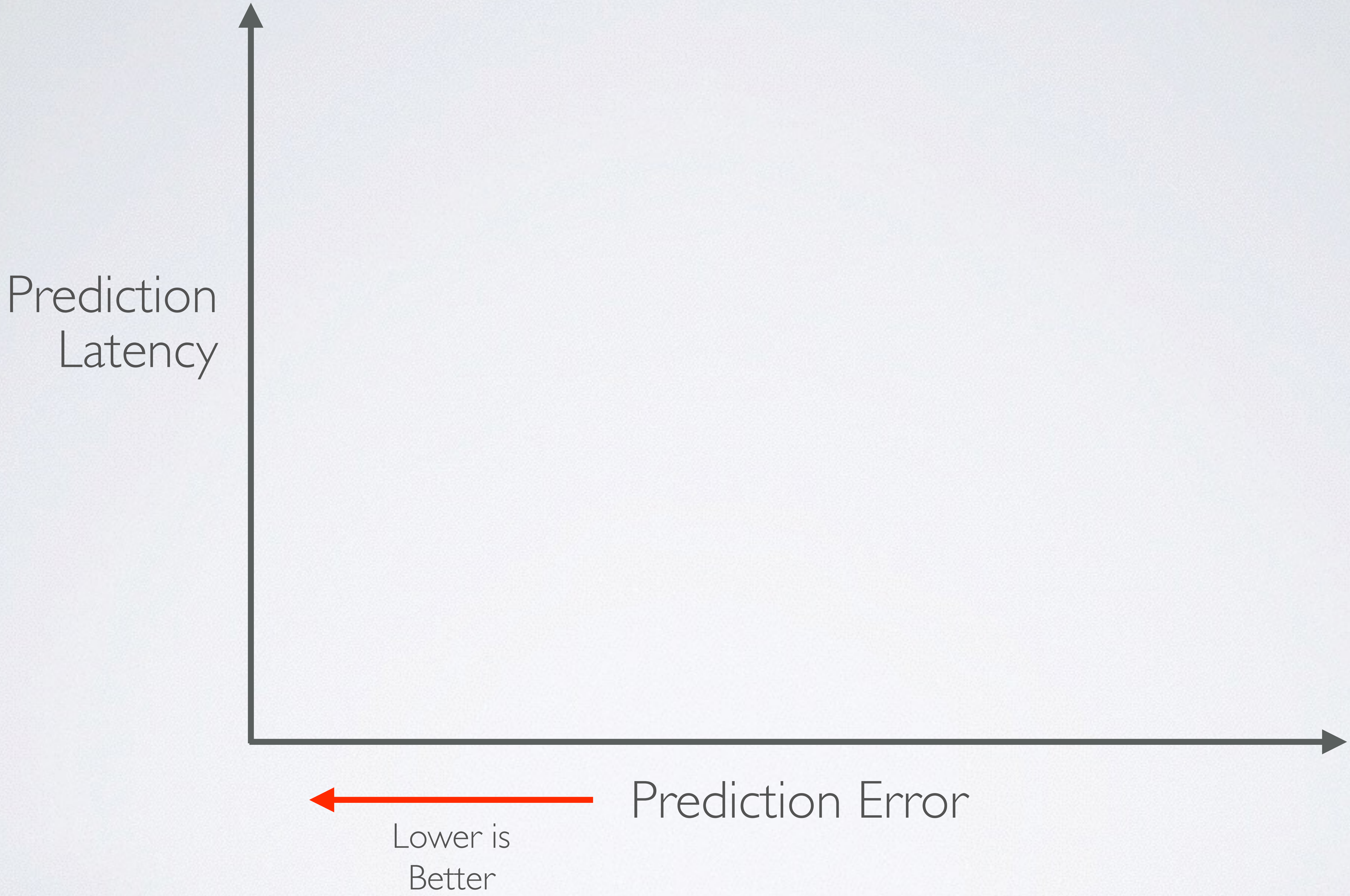
Catify: Music for Cats

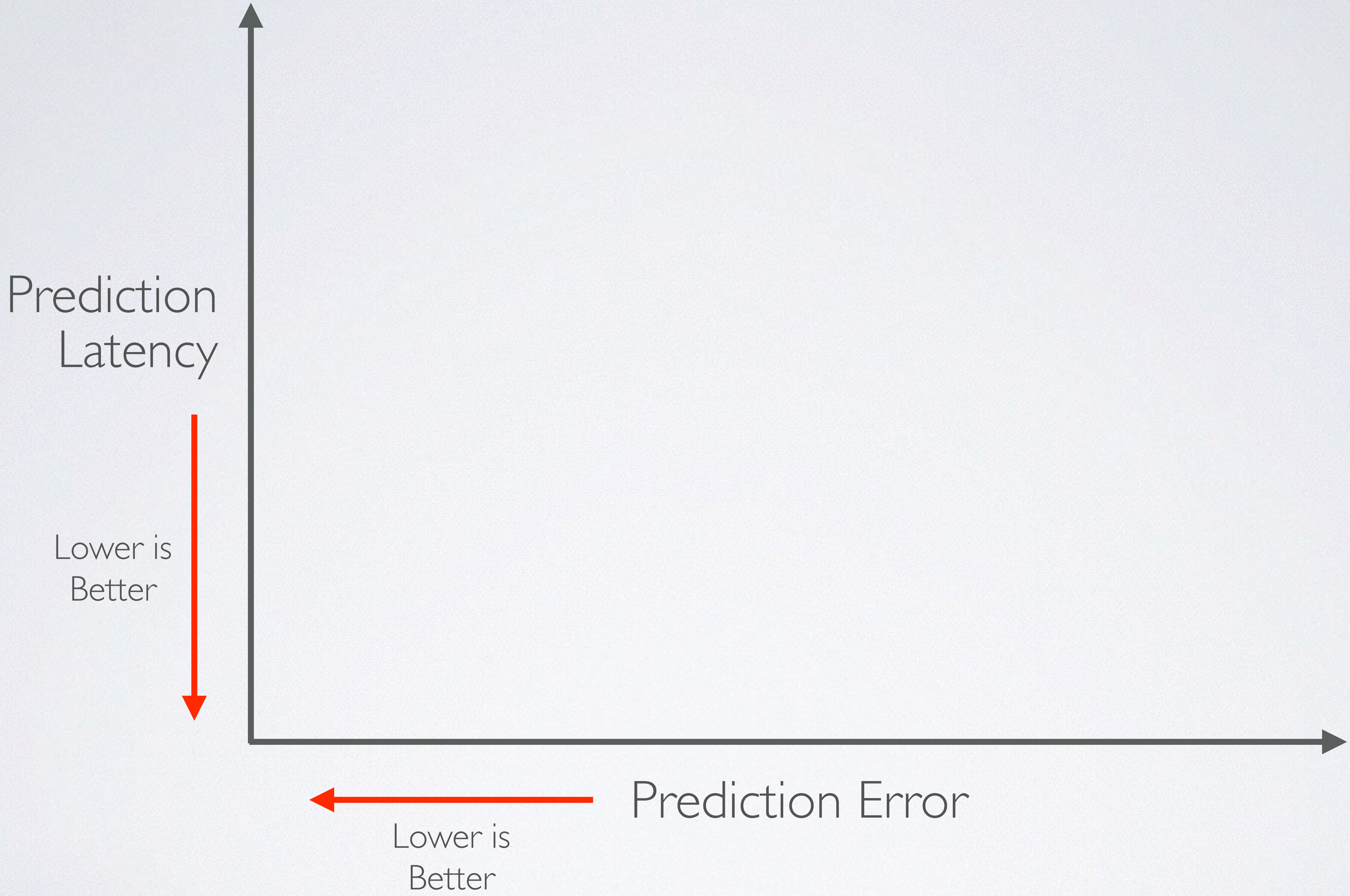


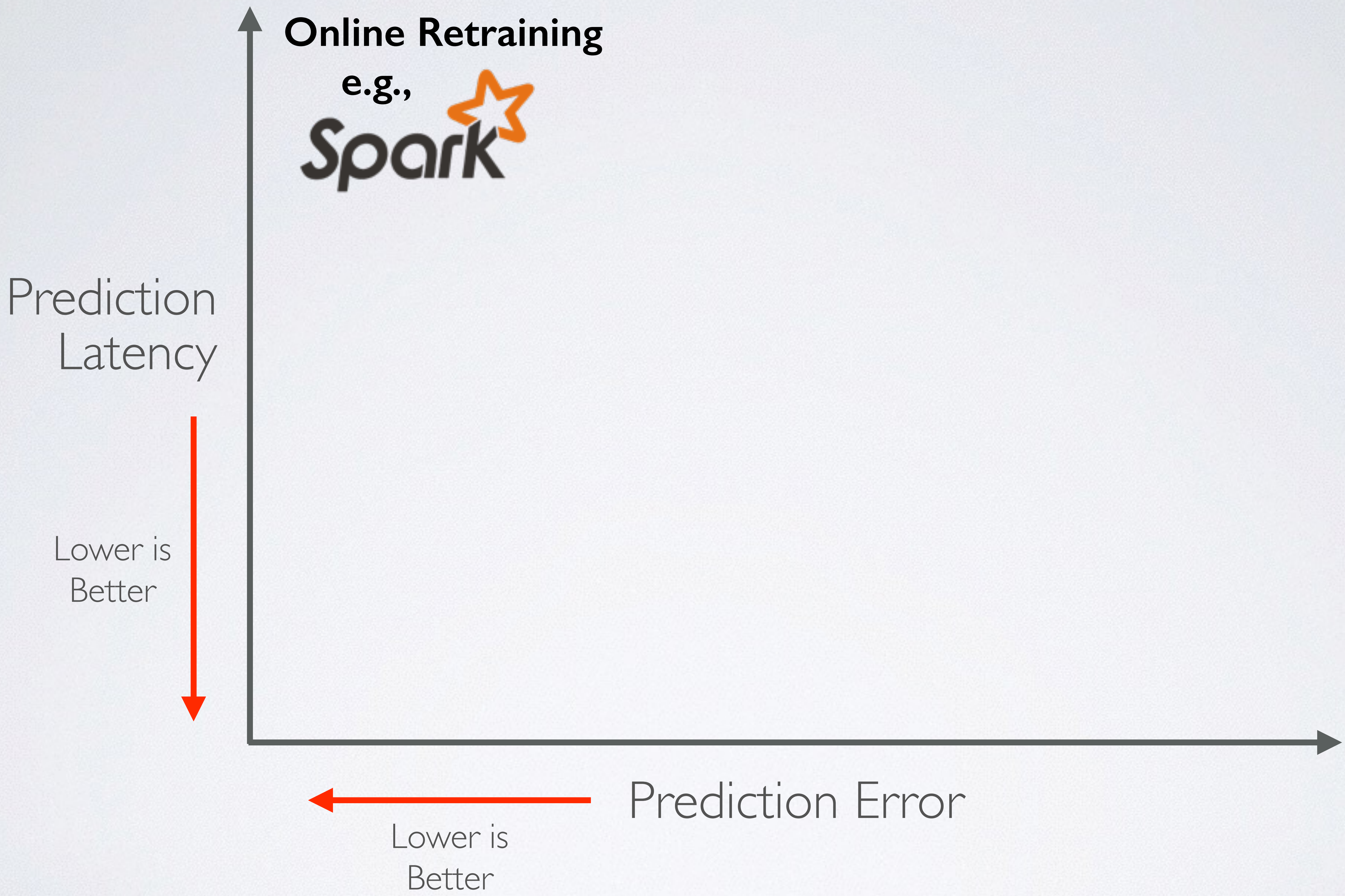
CatID	Song	Score
1	16	2.1
1	14	3.7
3	273	4.2
4	14	1.9



Prediction Latency
Prediction Error


Prediction
Latency
Prediction Error
Lower is
Better

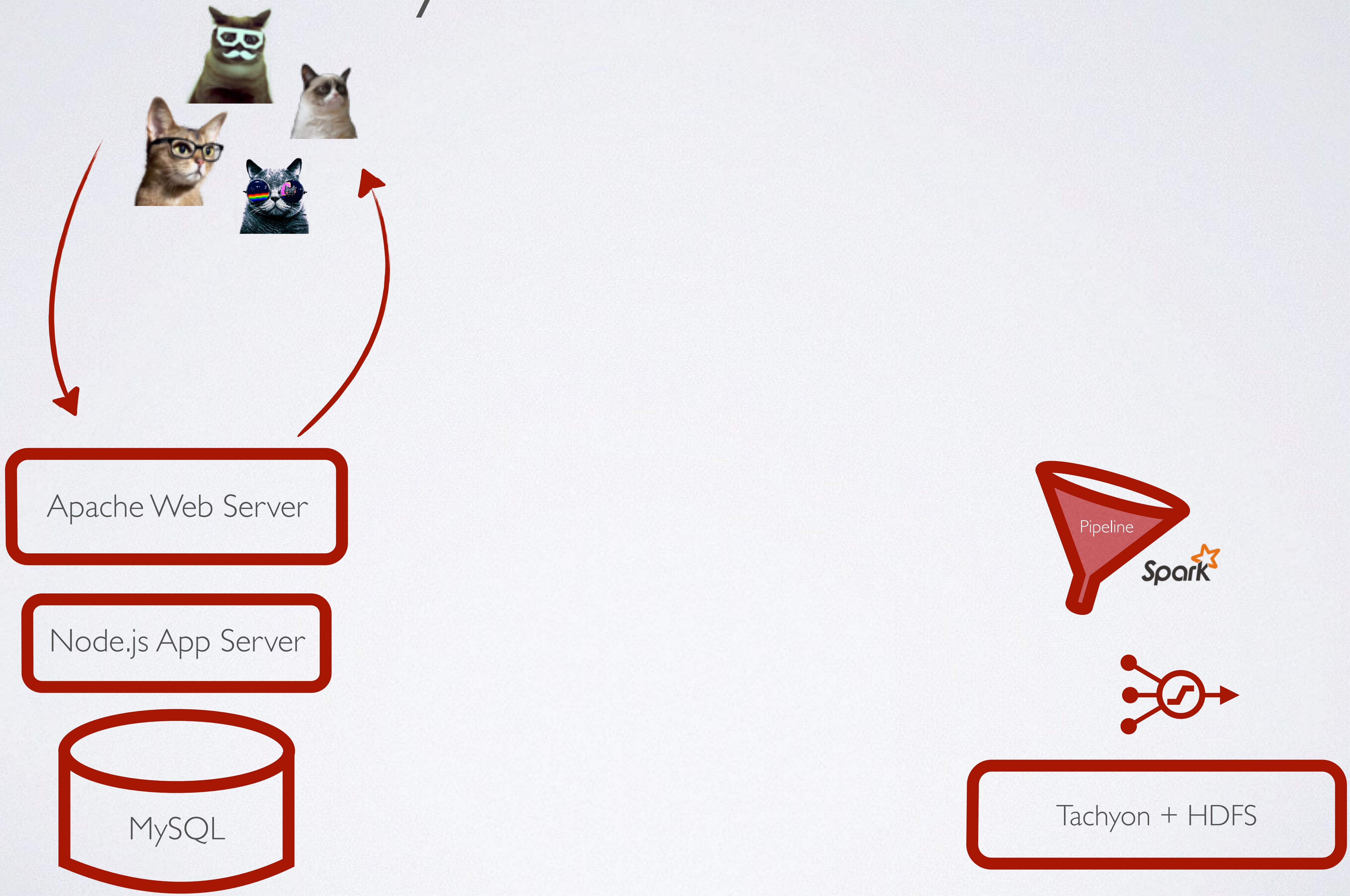

Prediction Latency
Lower is Better
Prediction Error
Lower is Better



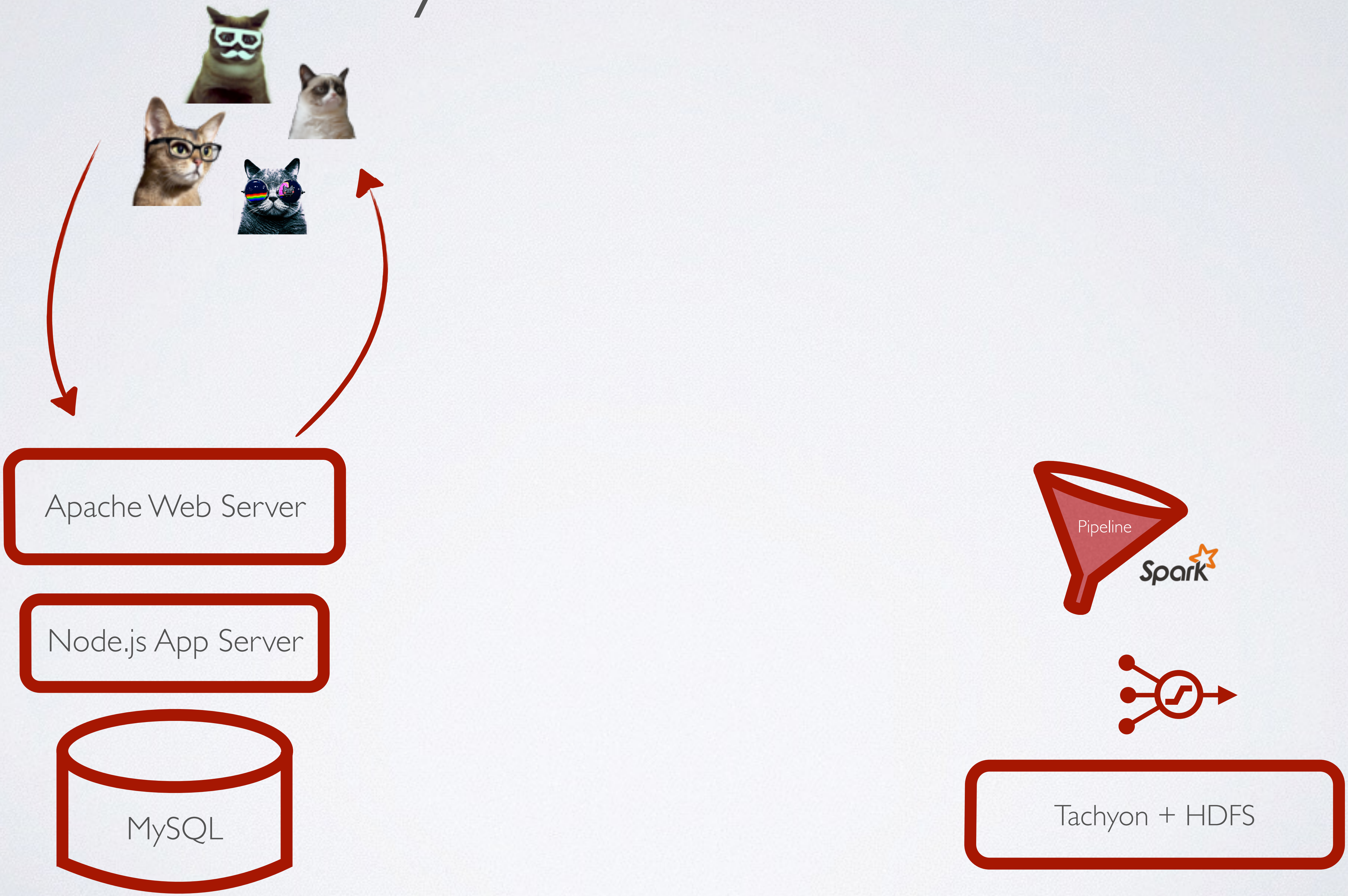
Online Retraining
e.g.,
Spark
Prediction Latency
Lower is Better
Prediction Error
Lower is Better

Catify: Music for Cats

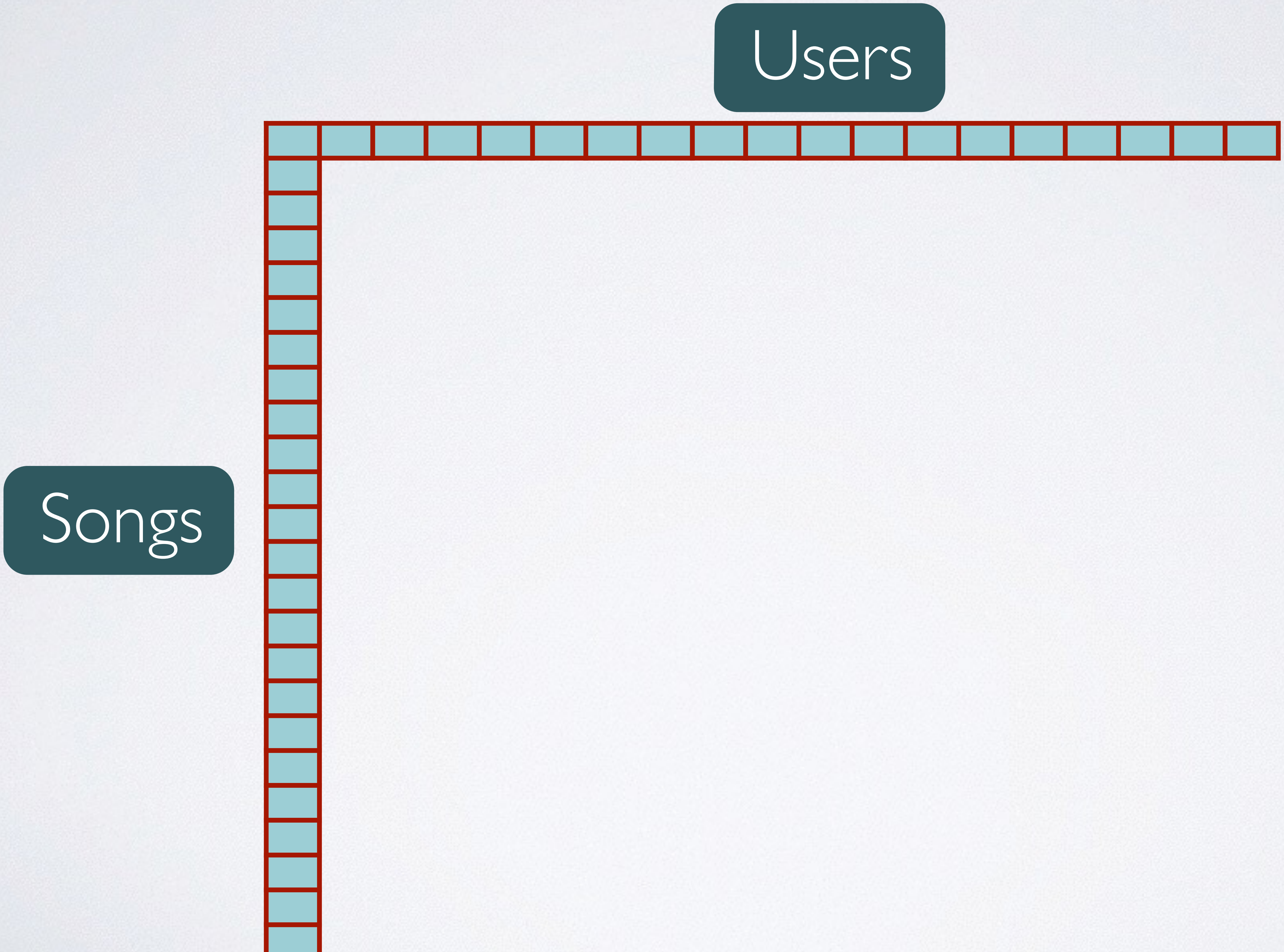
Catify: Music for Cats



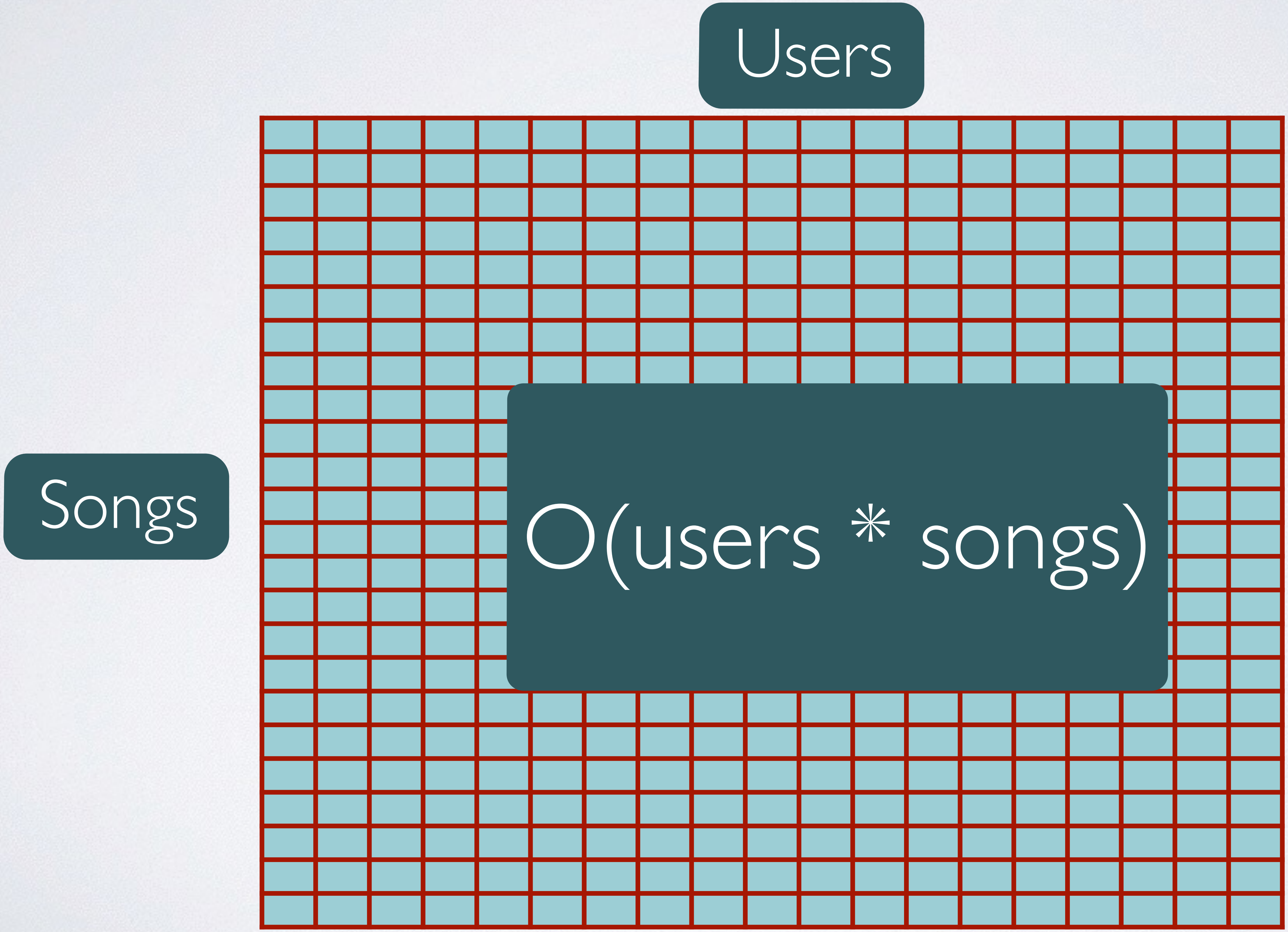
Catify: Music for Cats



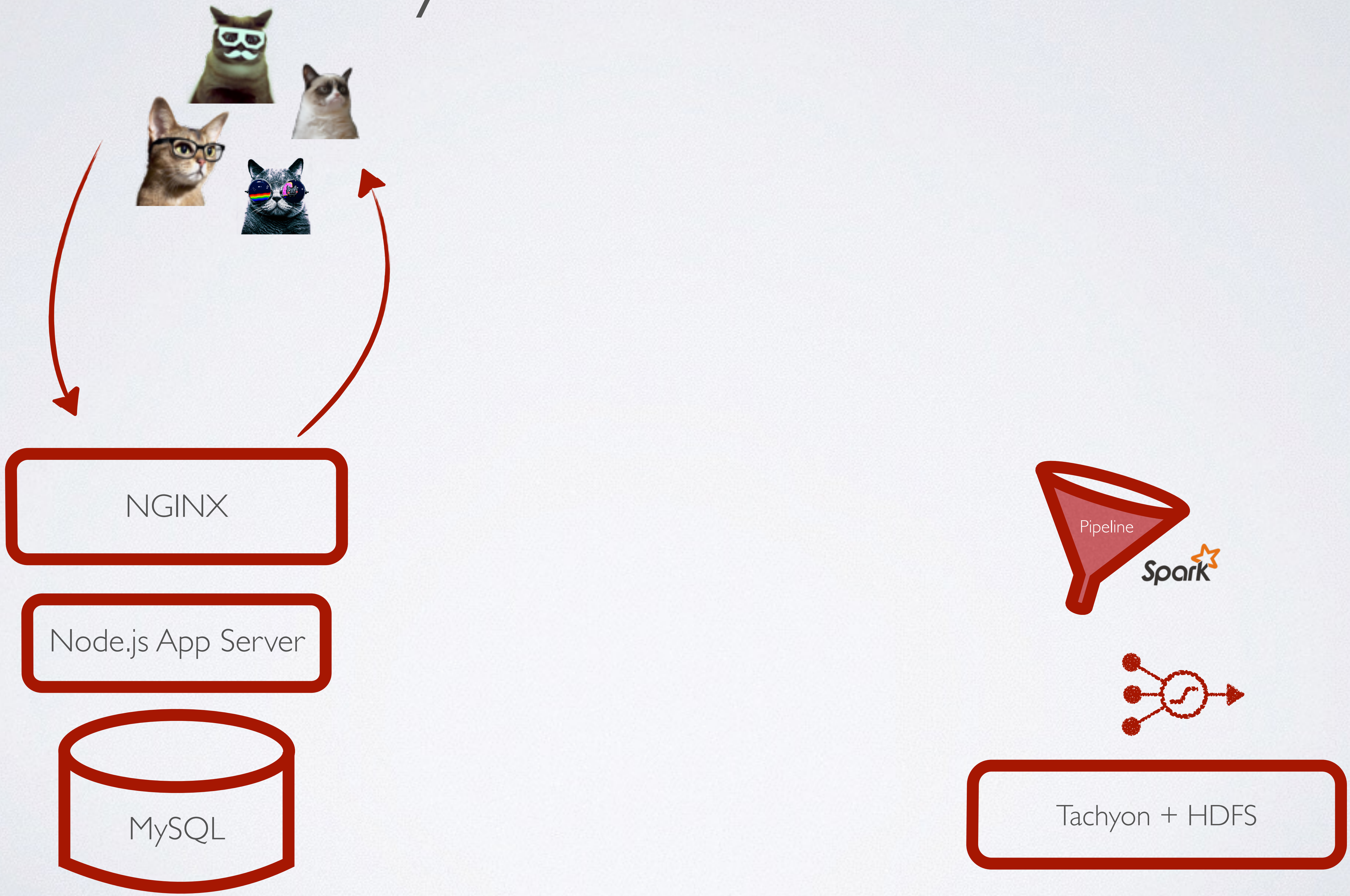
Catify: Music for Cats



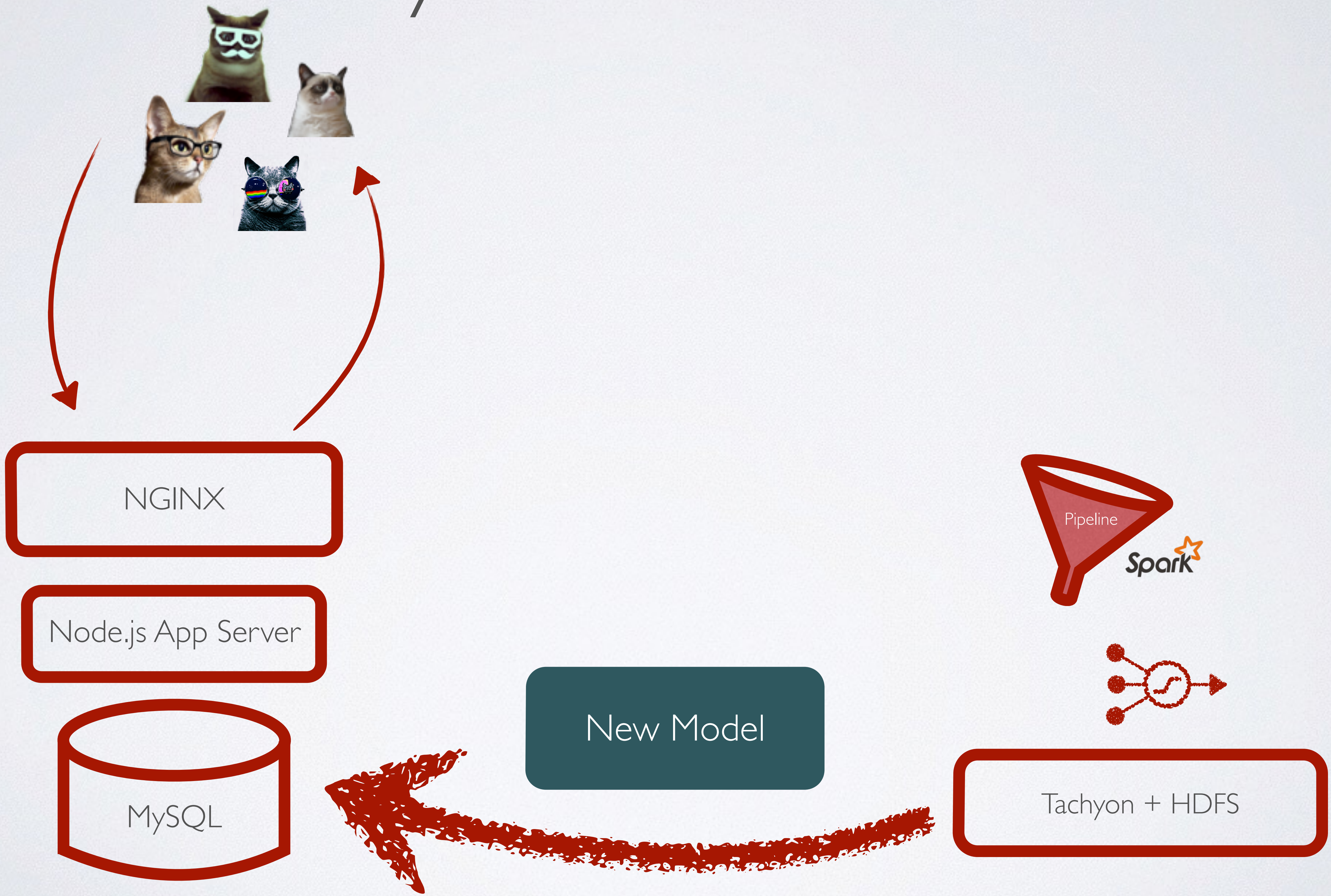
Catify: Music for Cats



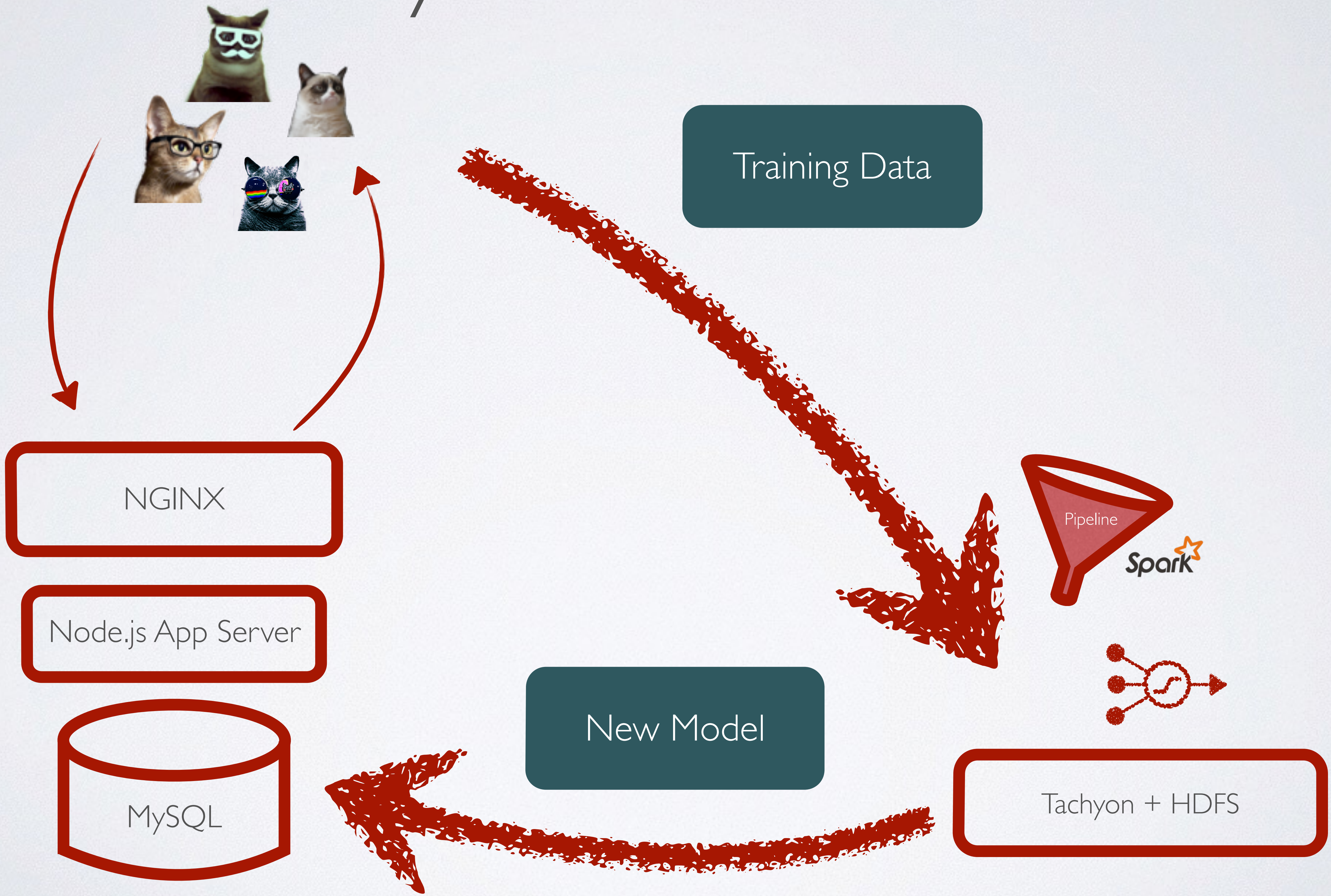
Catify: Music for Cats

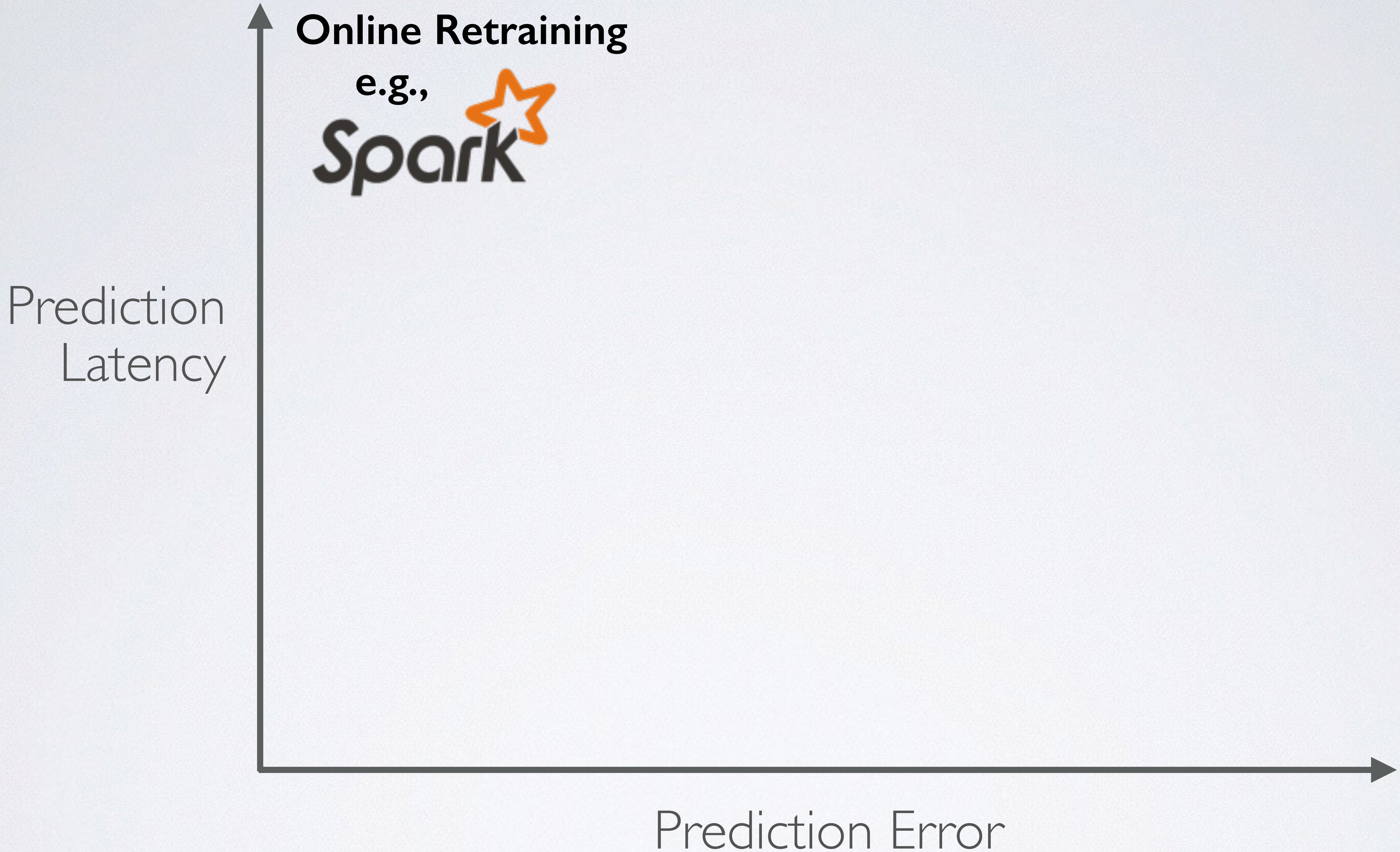


Catify: Music for Cats



Catify: Music for Cats




Online Retraining
e.g.,
Spark
Prediction Latency
Prediction Error


Online Retraining
e.g.,
Spark
Prediction Latency
Full pre-materialization
e.g.,
MySQL
Prediction Error

What’s wrong?

What's wrong?

1. Predictions have either:

What’s wrong?

1. Predictions have either:
    a. High latency, low staleness

What's wrong?

1. Predictions have either:
   a. High latency, low staleness
   b. Low latency, high staleness

What's wrong?

1. Predictions have either:
    a. High latency, low staleness
    b. Low latency, high staleness
2. Limited optimization of model semantics

What's wrong?

1. Predictions have either:
    a. High latency, low staleness
    b. Low latency, high staleness
2. Limited optimization of model semantics
3. Ad-hoc lifecycle management

Talk Outline

- **ML model management today**
- Velox system architecture
- Split model family
- Prediction serving
- Model management
- Next directions

Talk Outline

- ML model management today
- **Velox system architecture**
- Split model family
- Prediction serving
- Model management
- Next directions


Online Retraining
e.g.,
Spark
Prediction Latency
Full pre-materialization
e.g.,
MySQL
Prediction Error

Online Retraining
e.g.,
Spark
Prediction Latency
Full pre-materialization
e.g.,
MySQL
VELOX
Prediction Error

VELOX GOALS

VELOX GOALS

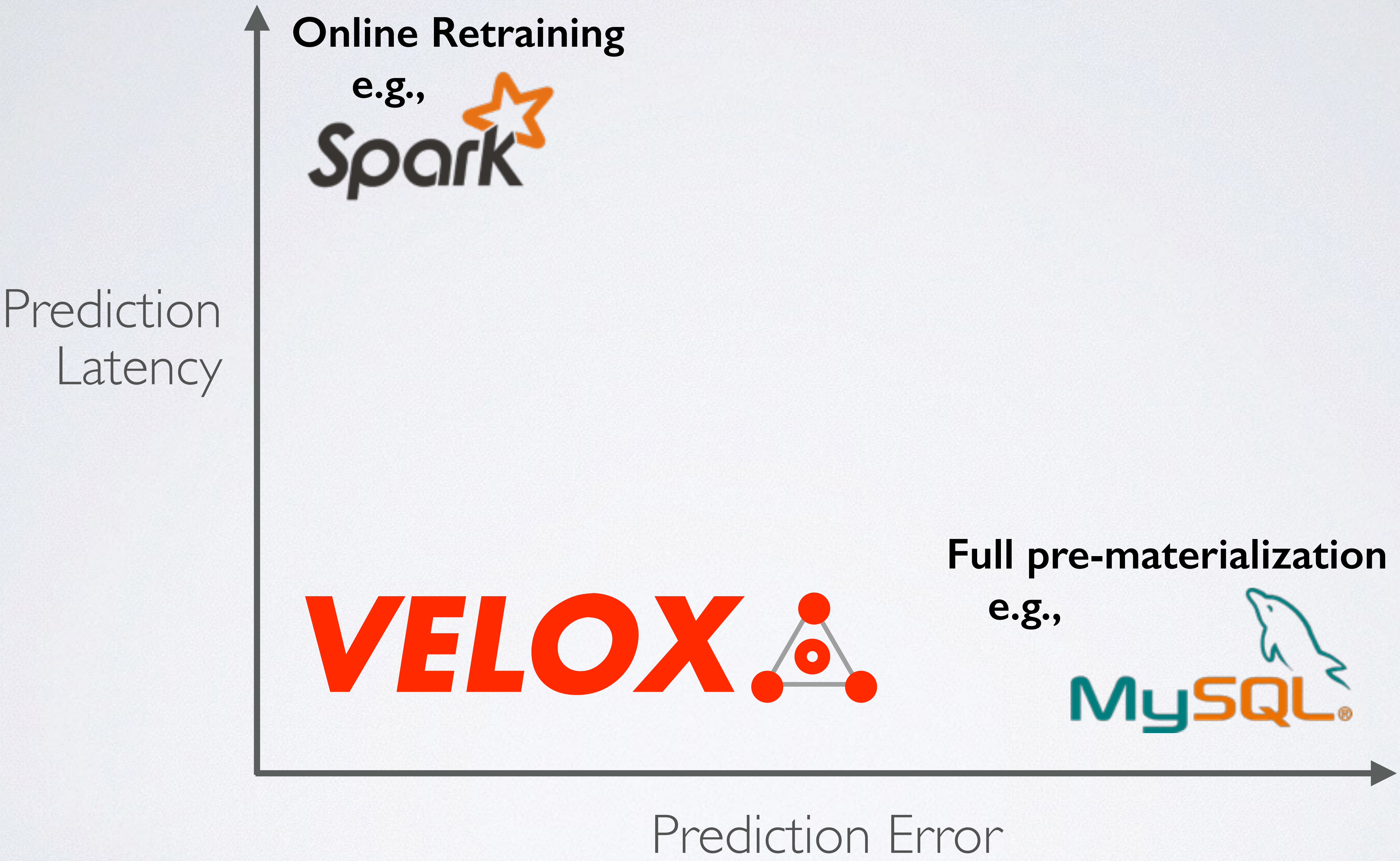
1. Low latency and low error predictions

VELOX GOALS

1. Low latency and low error predictions
2. Cross-cutting model-specific optimizations

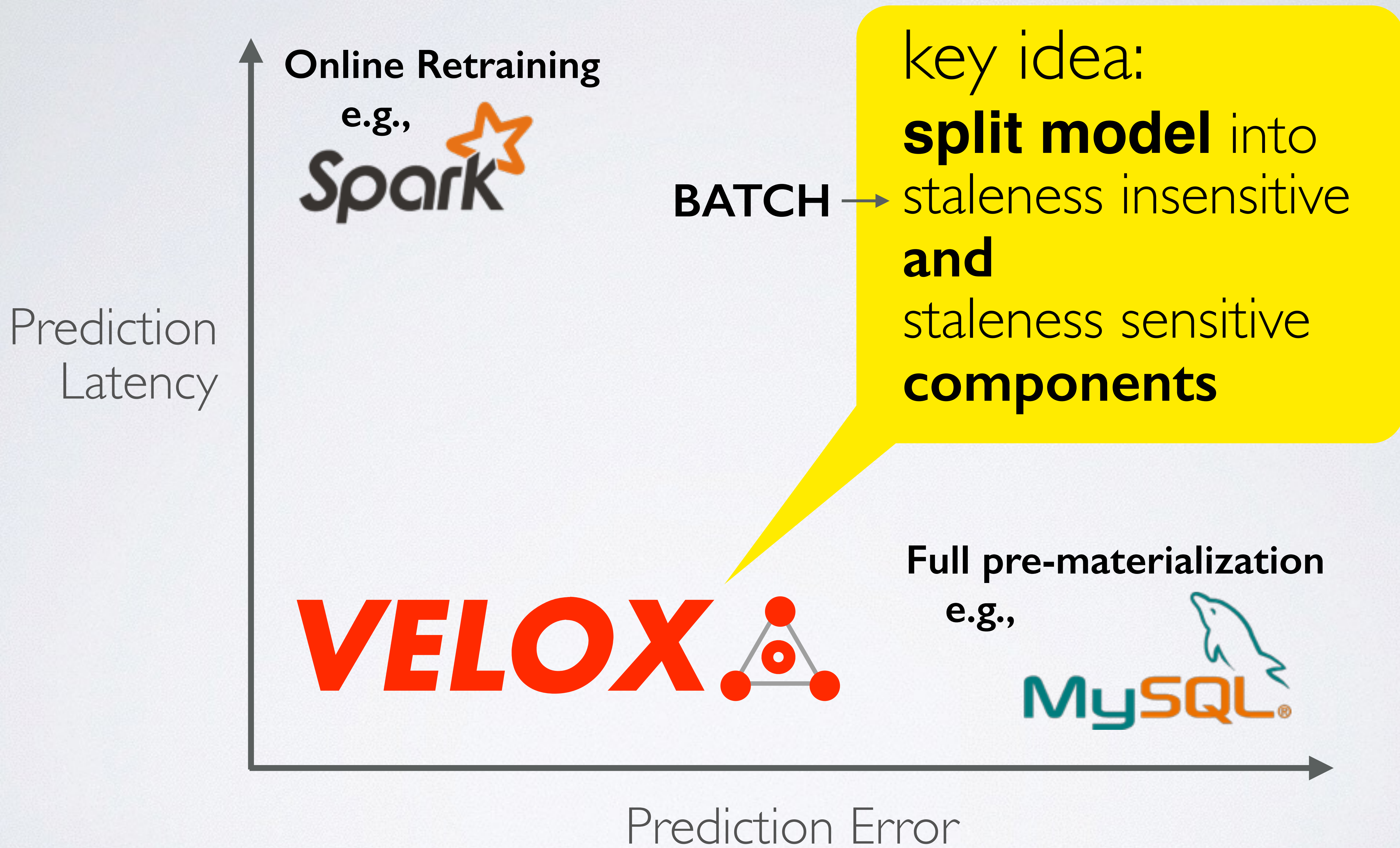
VELOX GOALS

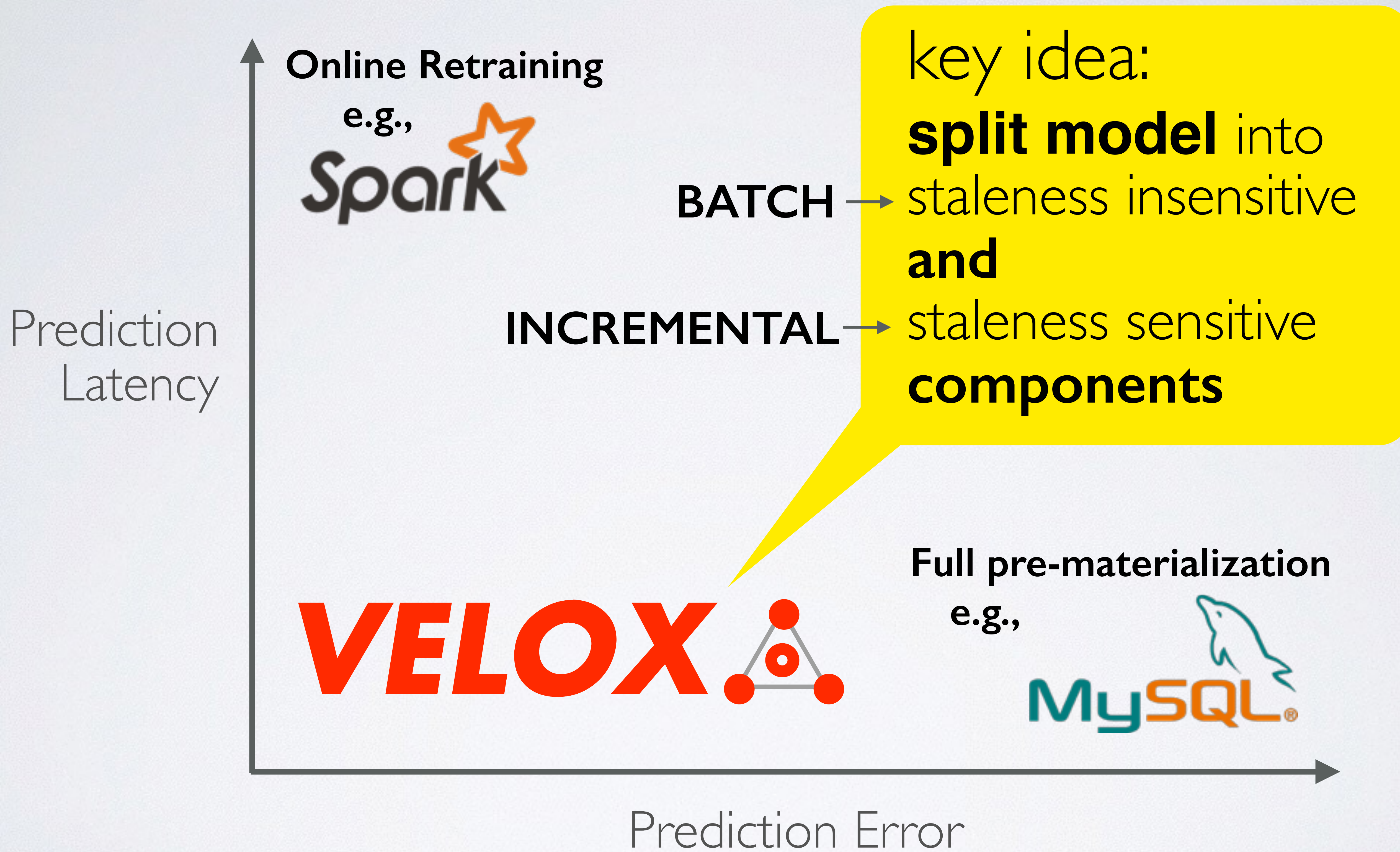
1. Low latency and low error predictions
2. Cross-cutting model-specific optimizations
3. Unified system eases operation



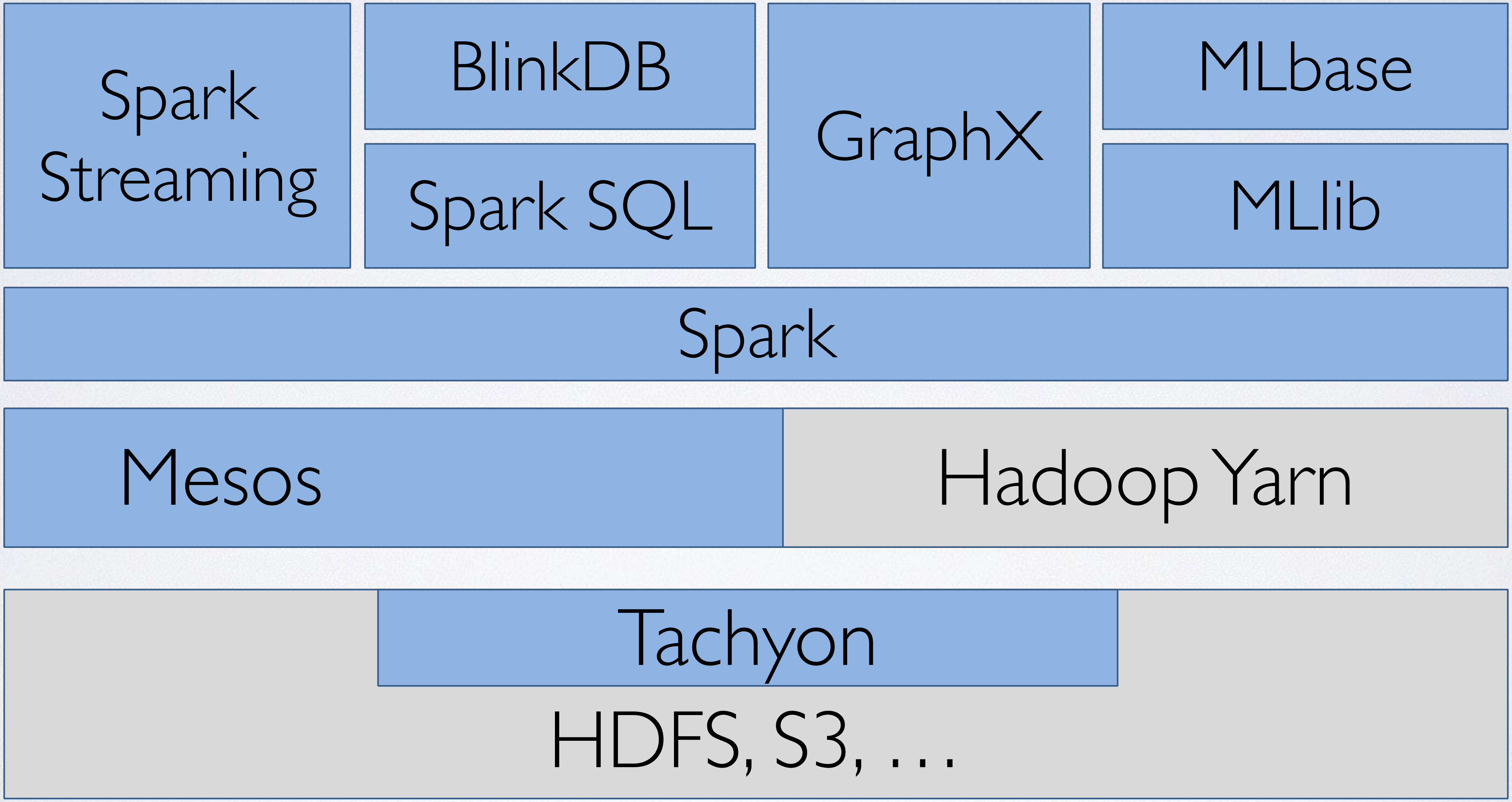
Online Retraining
e.g.,
Spark
Prediction Latency
VELOX
Full pre-materialization
e.g.,
MySQL
Prediction Error


Online Retraining
e.g.,
Spark
key idea:
split model into
staleness insensitive
and
staleness sensitive
components
Prediction Latency
VELOX
Full pre-materialization
e.g.,
MySQL
Prediction Error

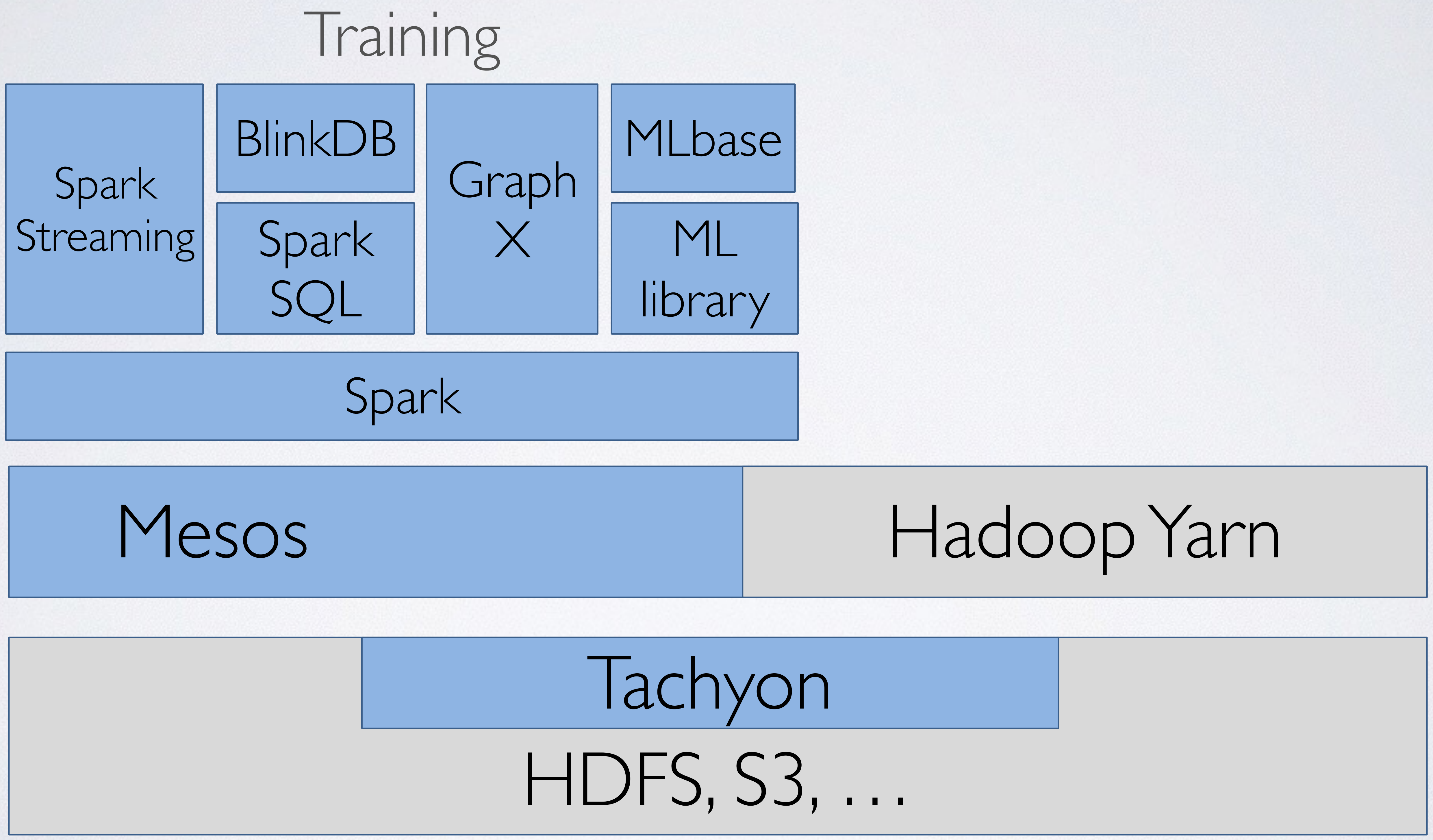

Online Retraining
e.g.,
Spark
BATCH
key idea:
split model into
staleness insensitive
and
staleness sensitive
components
Prediction Latency
VELOX
Full pre-materialization
e.g.,
MySQL
Prediction Error


Online Retraining
e.g.,
Spark
key idea:
split model into
BATCH
staleness insensitive
and
INCREMENTAL
staleness sensitive
components
Prediction
Latency
VELOX
Full pre-materialization
e.g.,
MySQL
Prediction Error

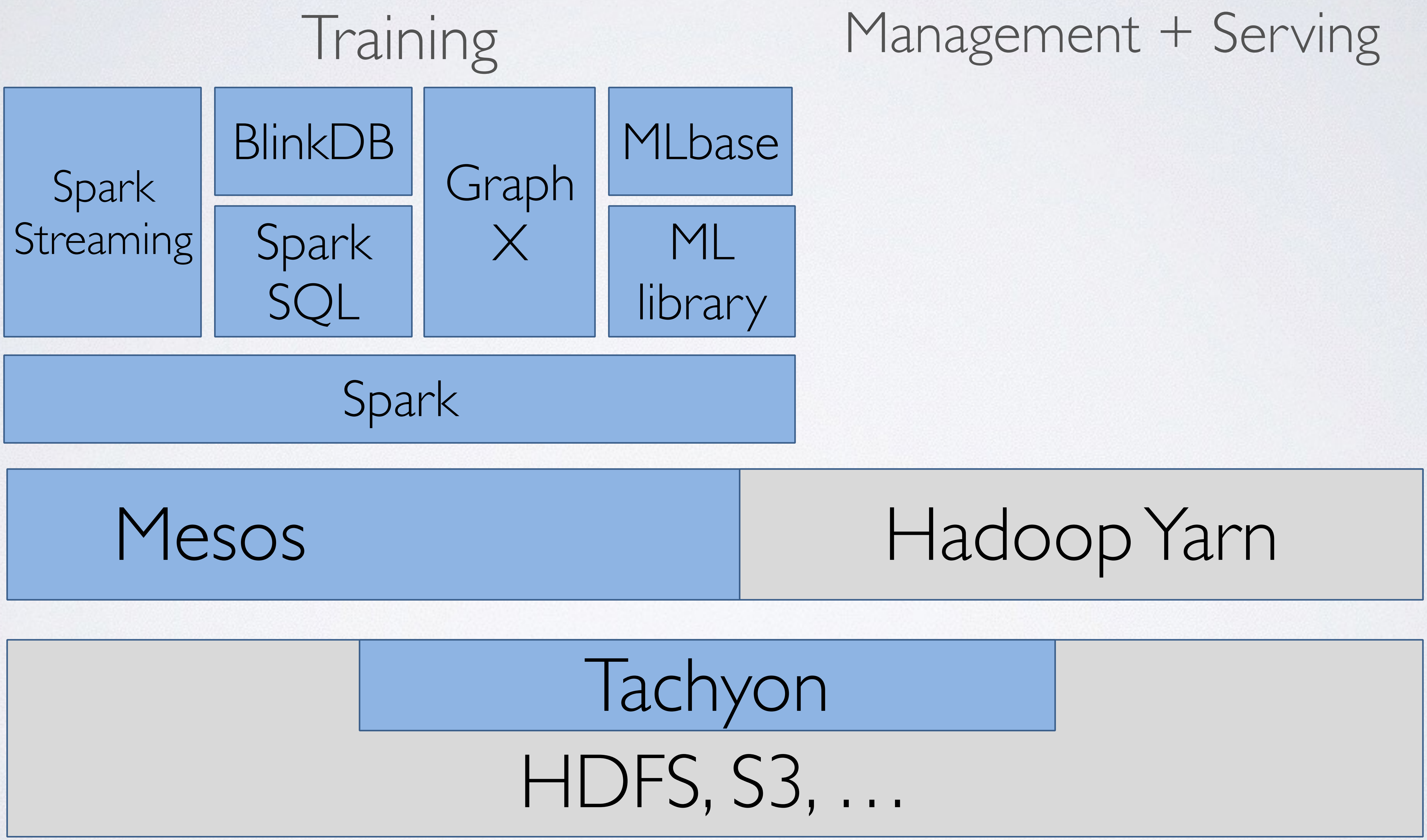
BERKELEY DATA ANALYTICS STACK (BDAS)



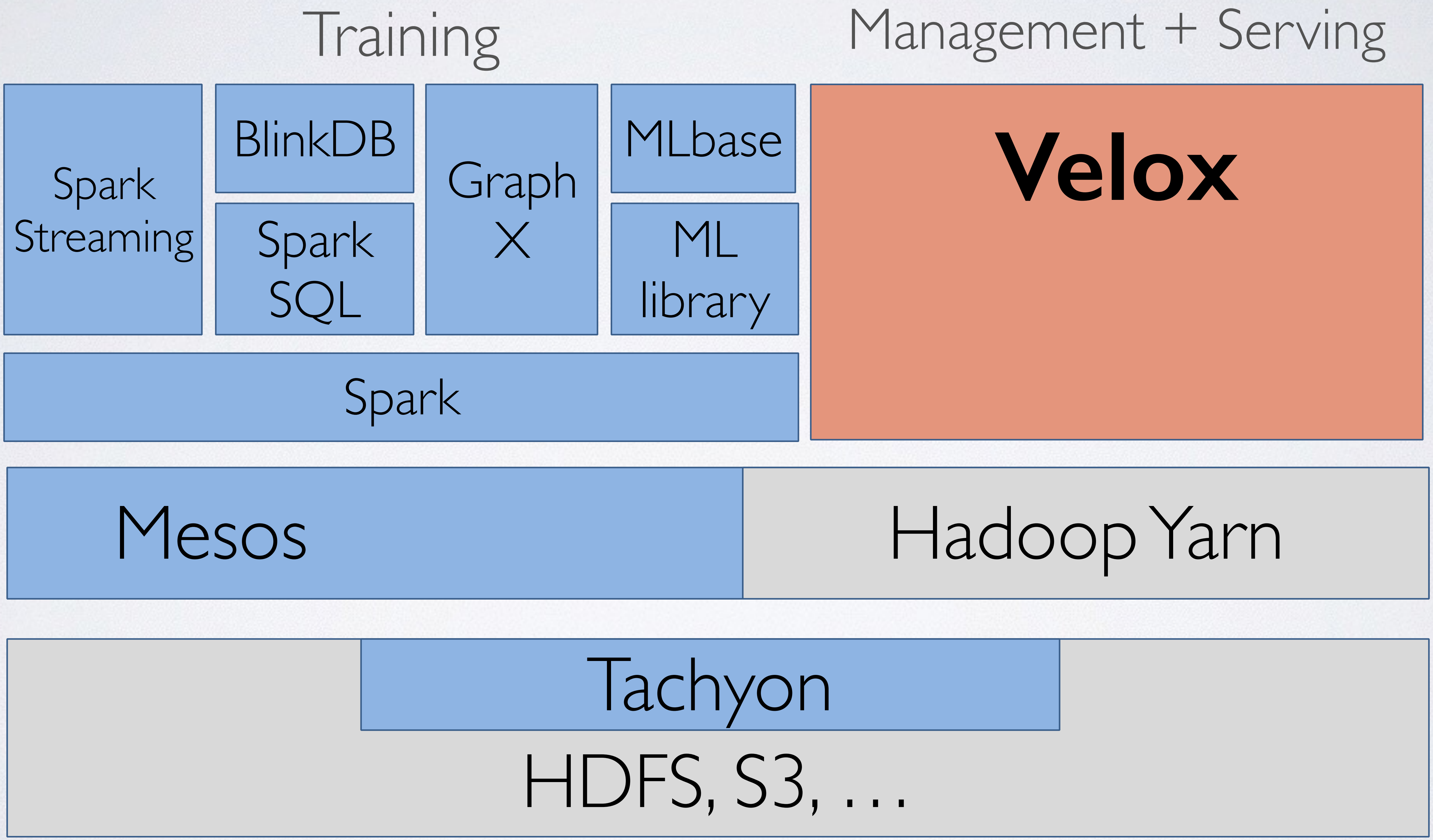
THE MISSING PIECE



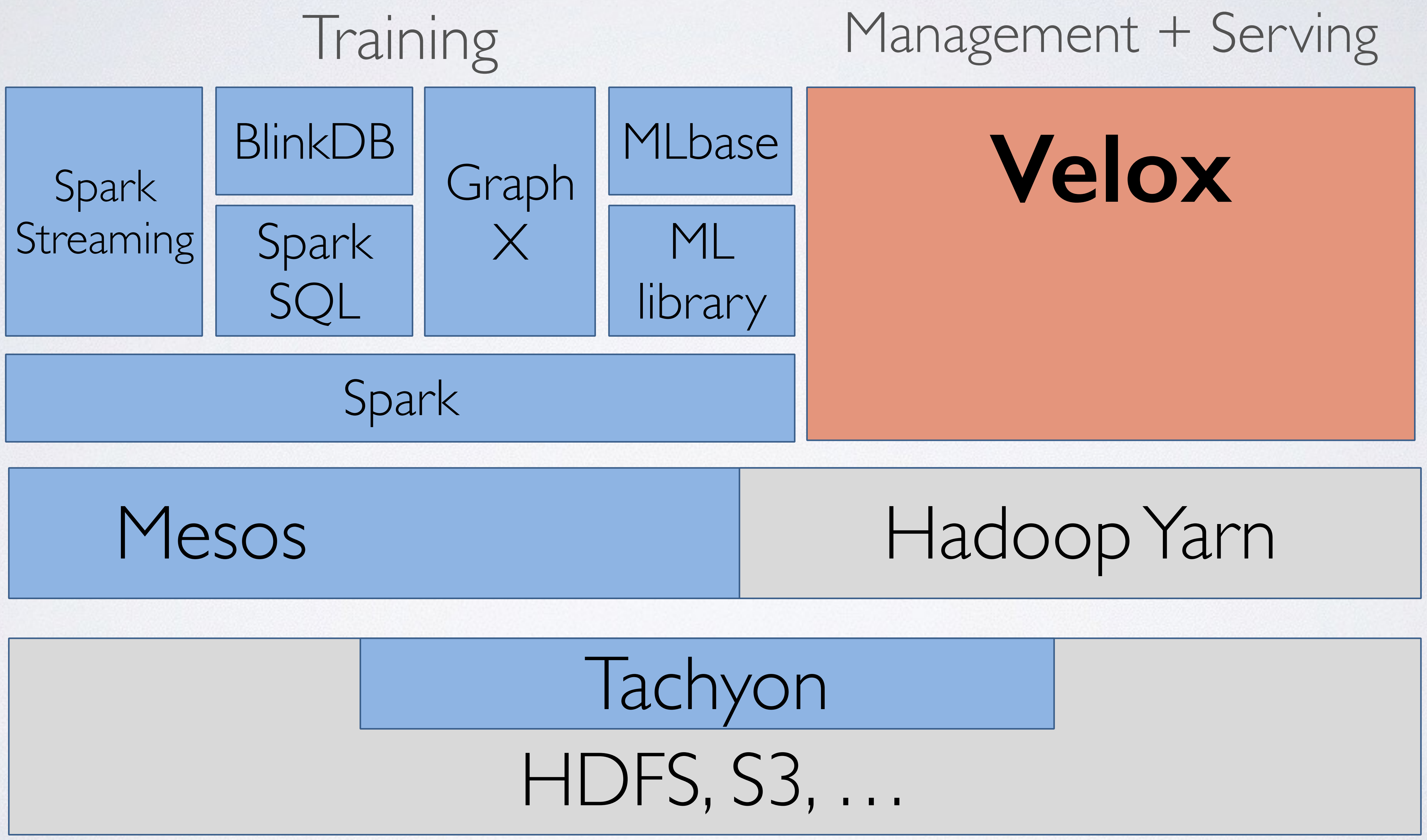
THE MISSING PIECE



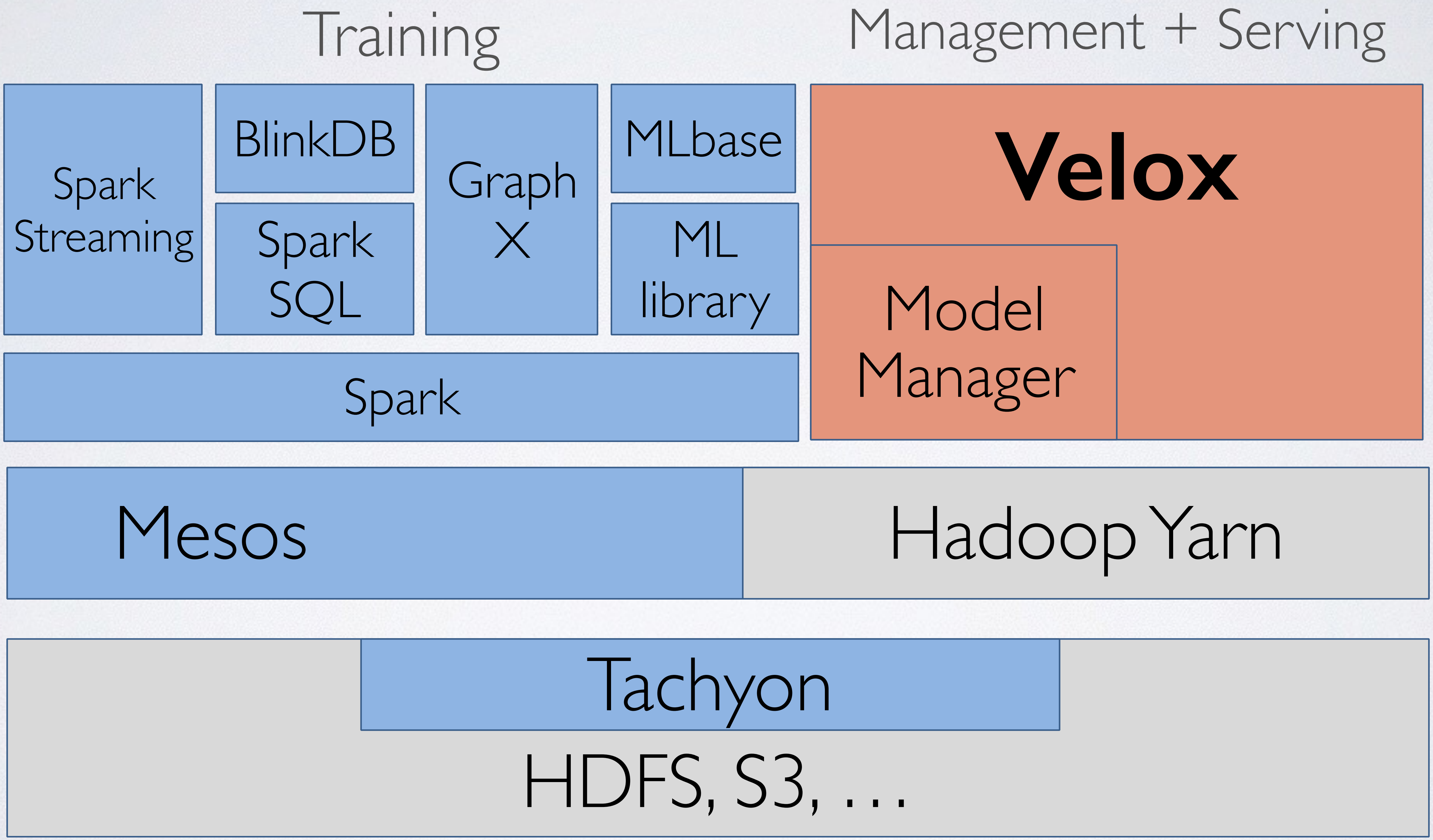
THE MISSING PIECE



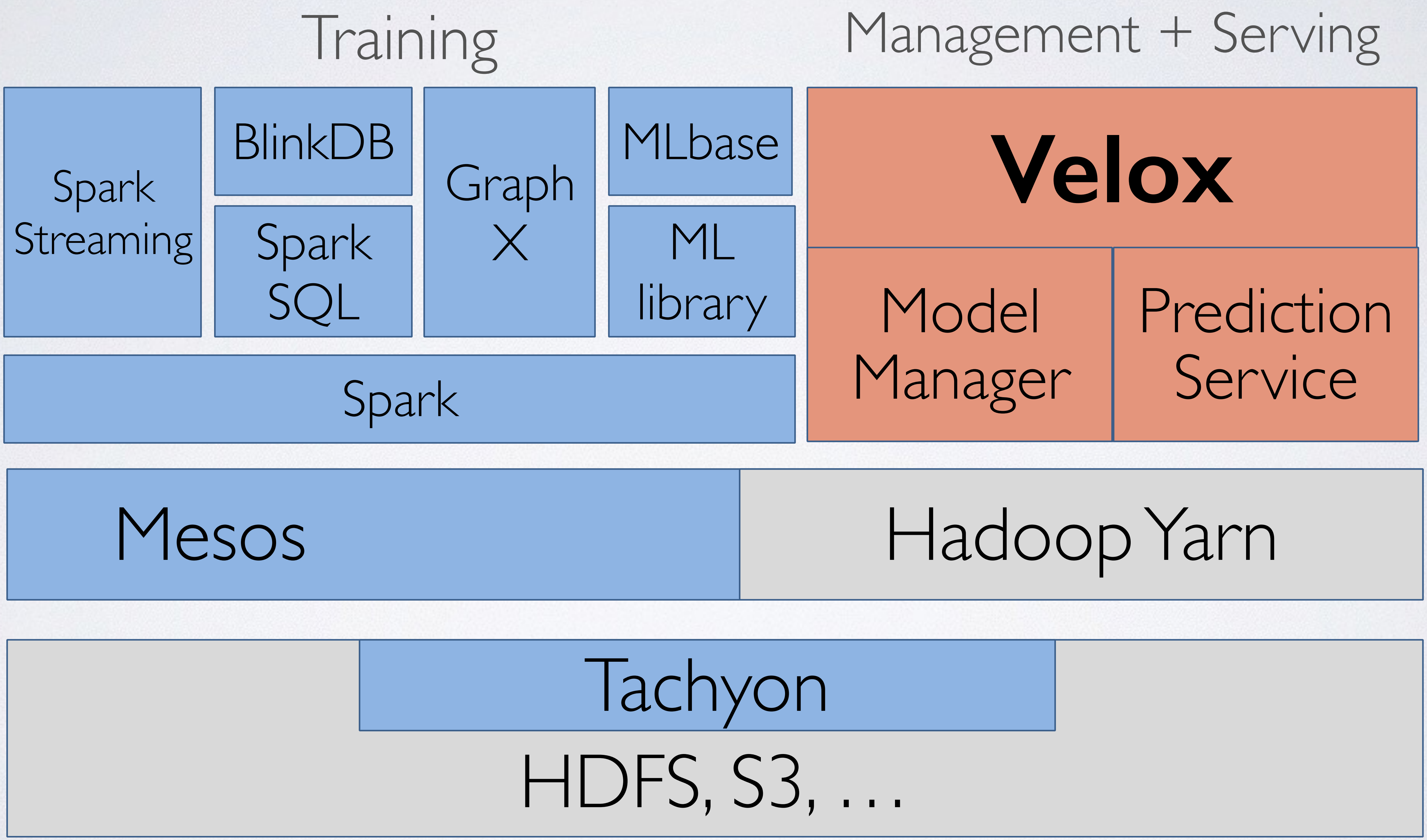
THE MISSING PIECE



THE MISSING PIECE



THE MISSING PIECE



PREDICTION SERVICE

PREDICTION SERVICE

1. Implements model serving API

PREDICTION SERVICE

1. Implements model serving API
2. Low latency; < 10ms response time

PREDICTION SERVICE

1. Implements model serving API
2. Low latency; < 10ms response time
3. "Fuzzy" materialized view of model state

PREDICTION SERVICE

1. Implements model serving API
2. Low latency; < 10ms response time
3. "Fuzzy" materialized view of model state

MODEL MANAGER

PREDICTION SERVICE

1. Implements model serving API
2. Low latency; < 10ms response time
3. “Fuzzy” materialized view of model state

MODEL MANAGER

1. Maintains models via online and batch retraining

PREDICTION SERVICE

1. Implements model serving API
2. Low latency; < 10ms response time
3. "Fuzzy" materialized view of model state

MODEL MANAGER

1. Maintains models via online and batch retraining
2. Stores model catalog, metadata, versioning

PREDICTION SERVICE

1. Implements model serving API
2. Low latency; < 10ms response time
3. "Fuzzy" materialized view of model state

MODEL MANAGER

1. Maintains models via online and batch retraining
2. Stores model catalog, metadata, versioning
3. Contains library of standard models + custom API

Talk Outline

- ML model management today
- **Velox system architecture**
- Key idea: Split model family
- Prediction serving
- Model management
- Next directions

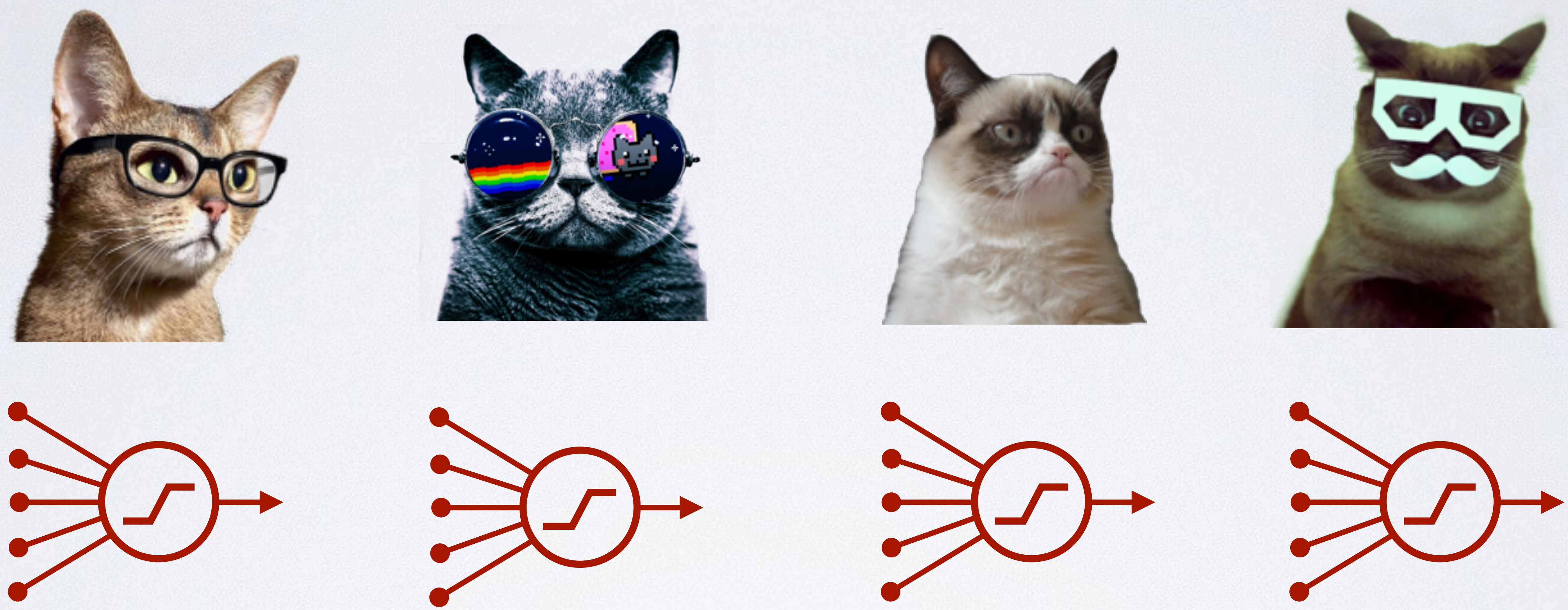
Talk Outline

- ML model management today
- Velox system architecture
- **Key idea: Split model family**
- Prediction serving
- Model management
- Next directions

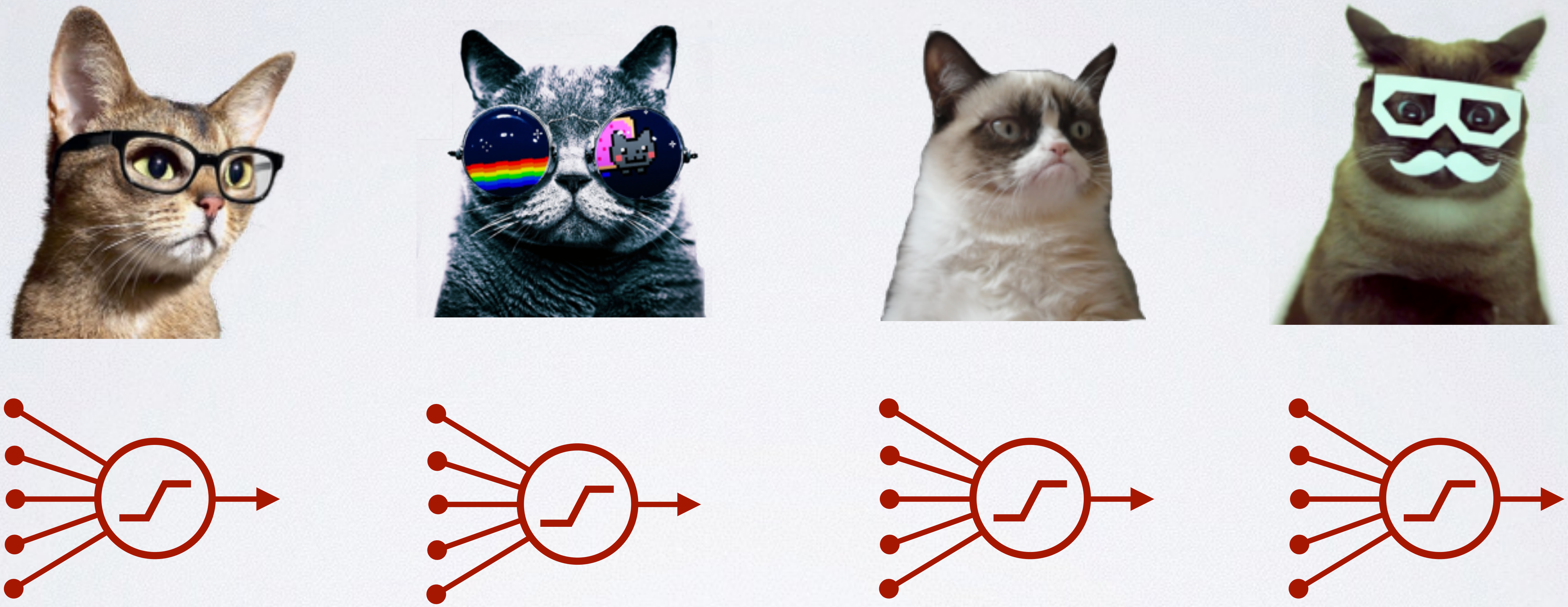
PERSONALIZED MODELING




PERSONALIZED MODELING

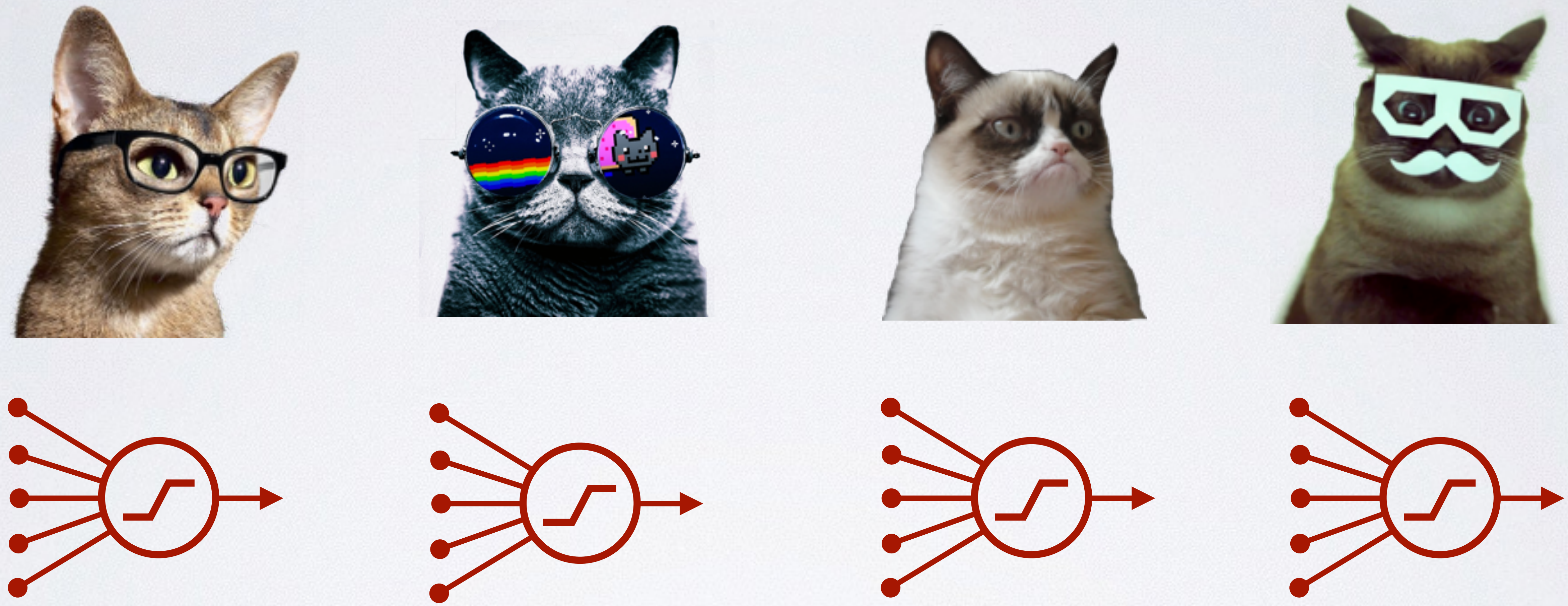


PERSONALIZED MODELING



A Separate Model for Each User?

PERSONALIZED MODELING

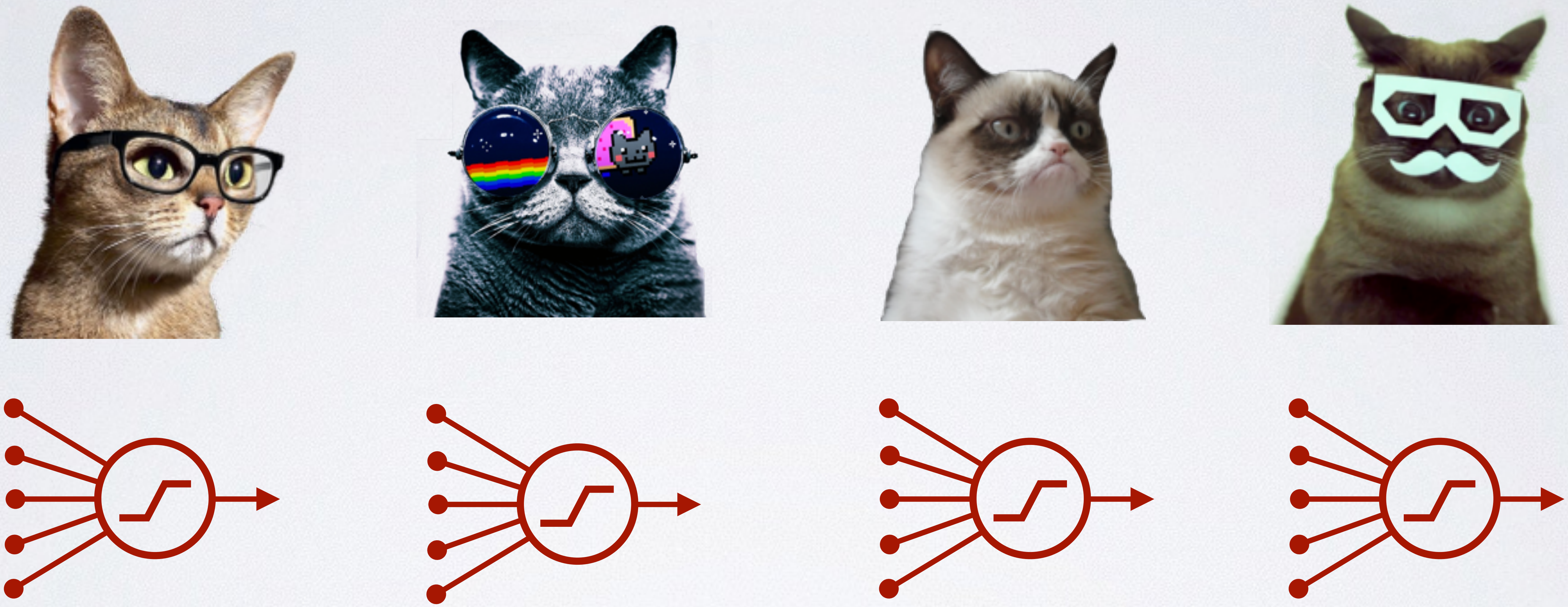


A Separate Model for Each User?

Computationally Inefficient

many complex models

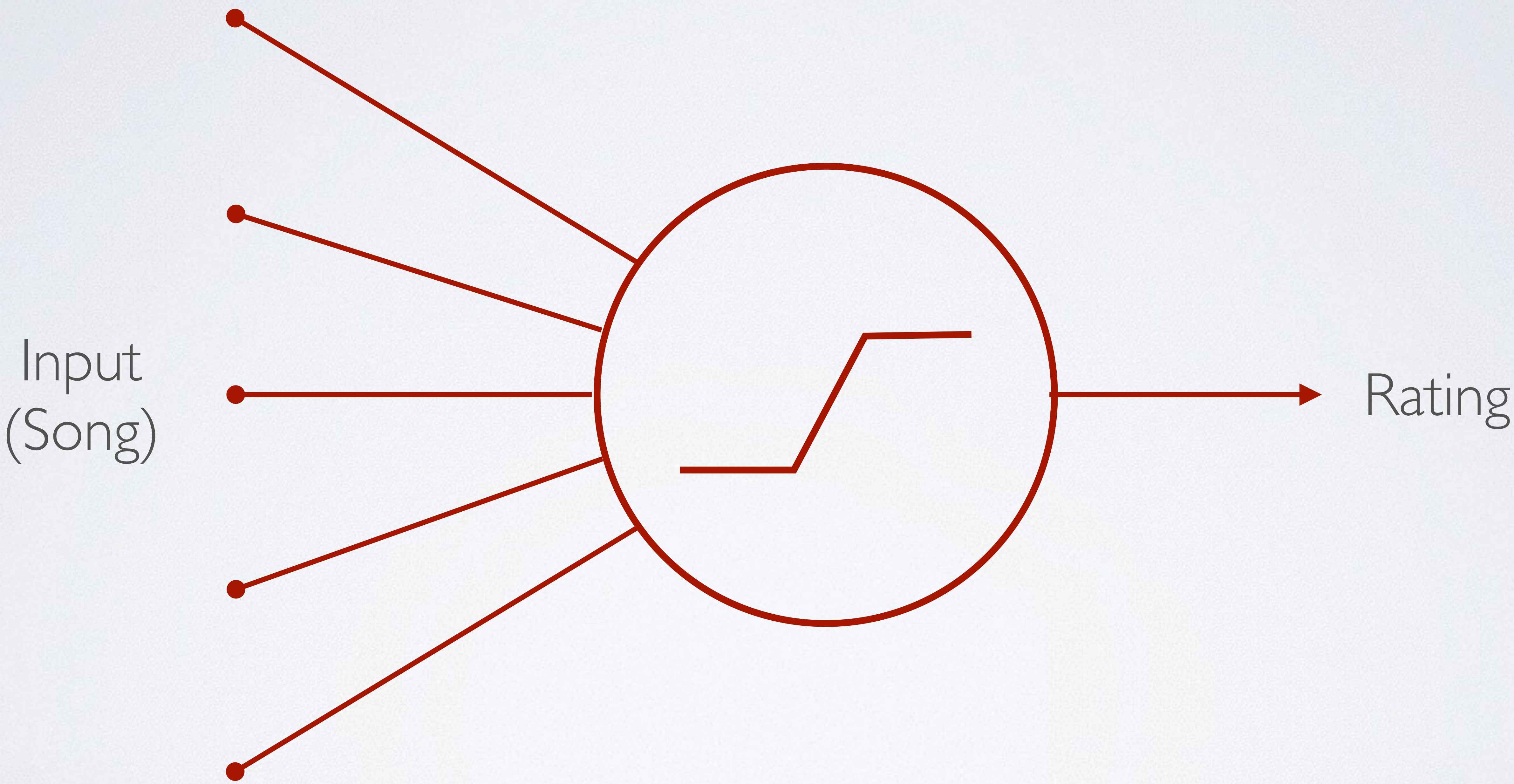
PERSONALIZED MODELING

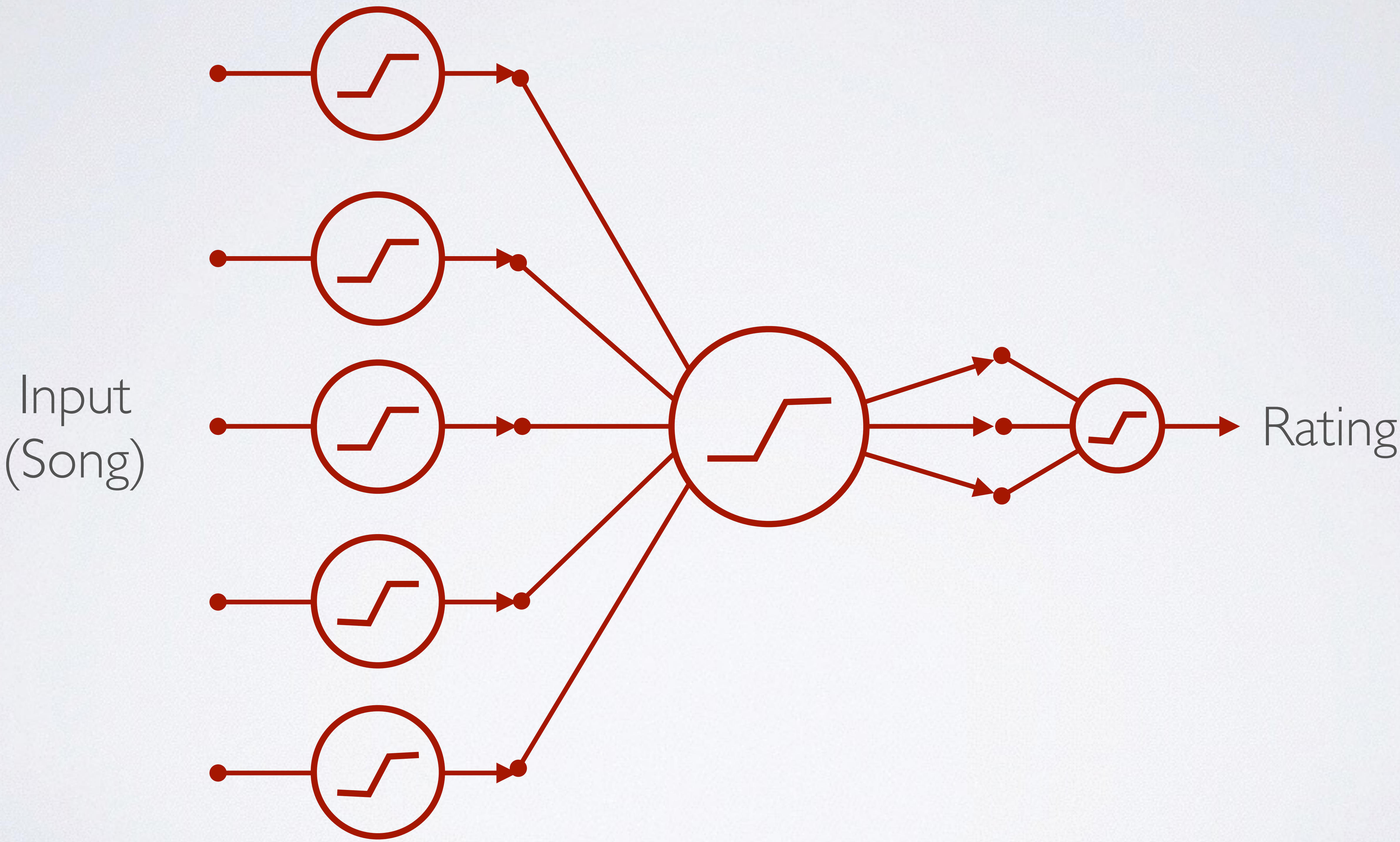


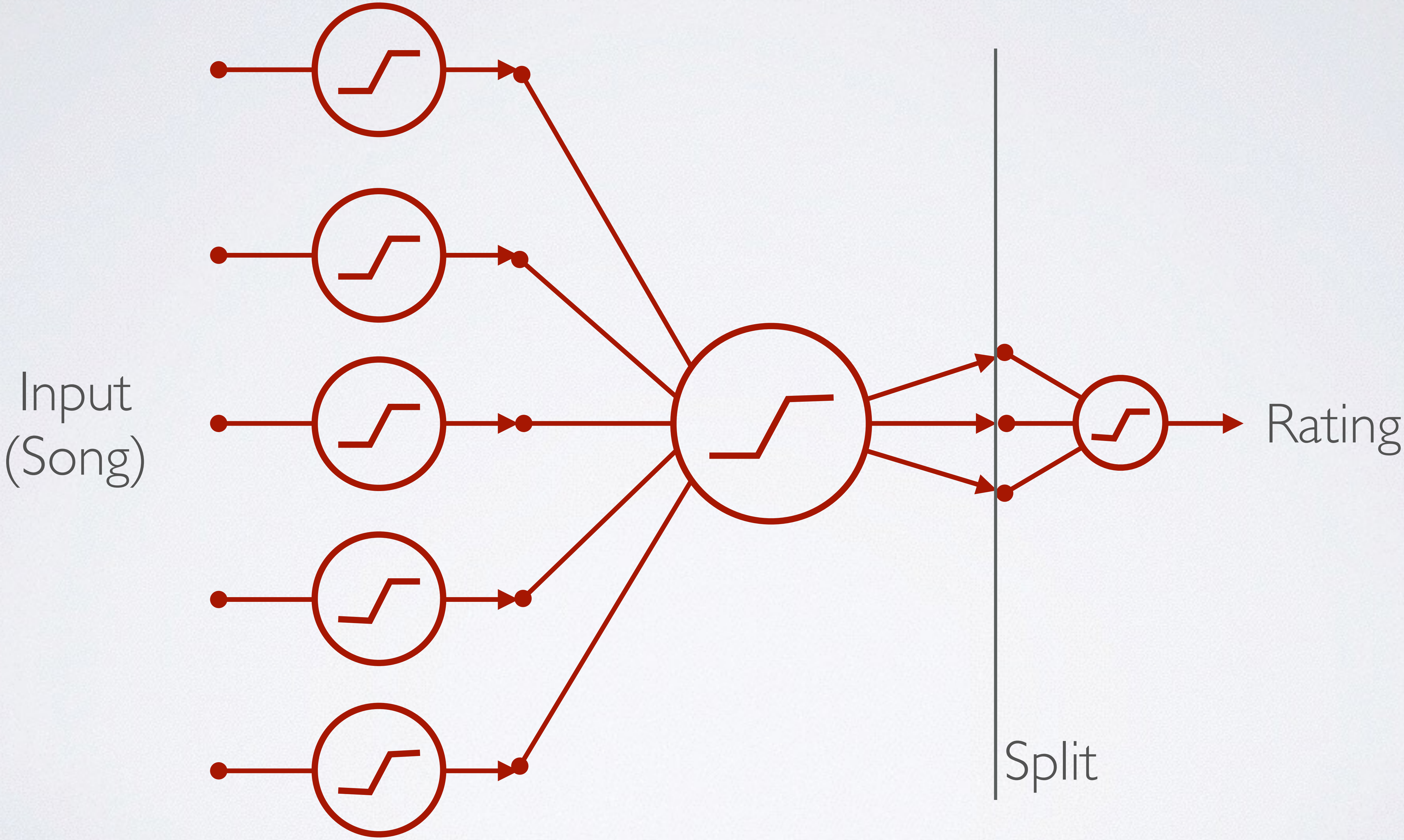
A Separate Model for Each User?

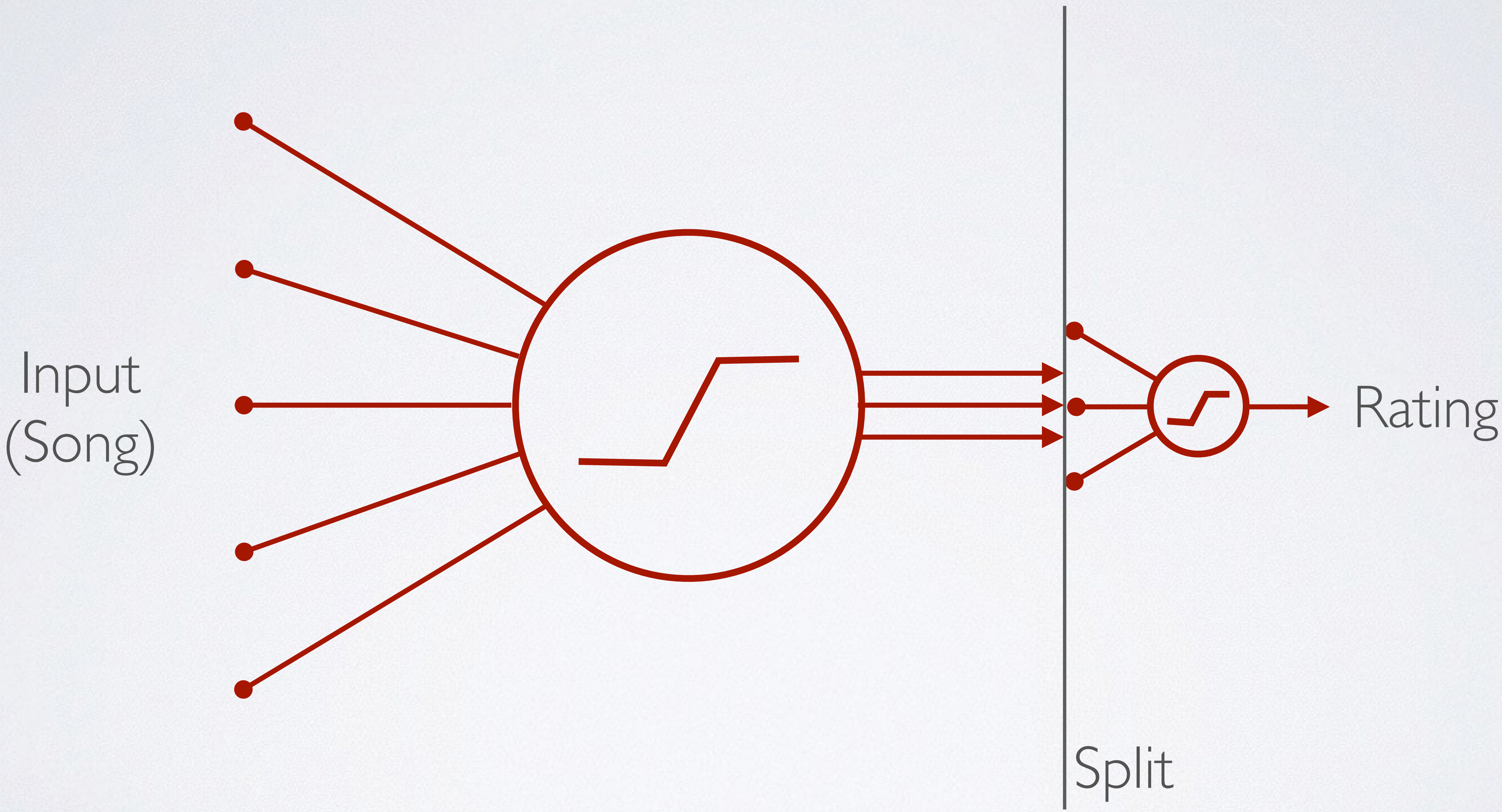
Computationally Inefficient
many complex models

Statistically Inefficient
not enough data per user

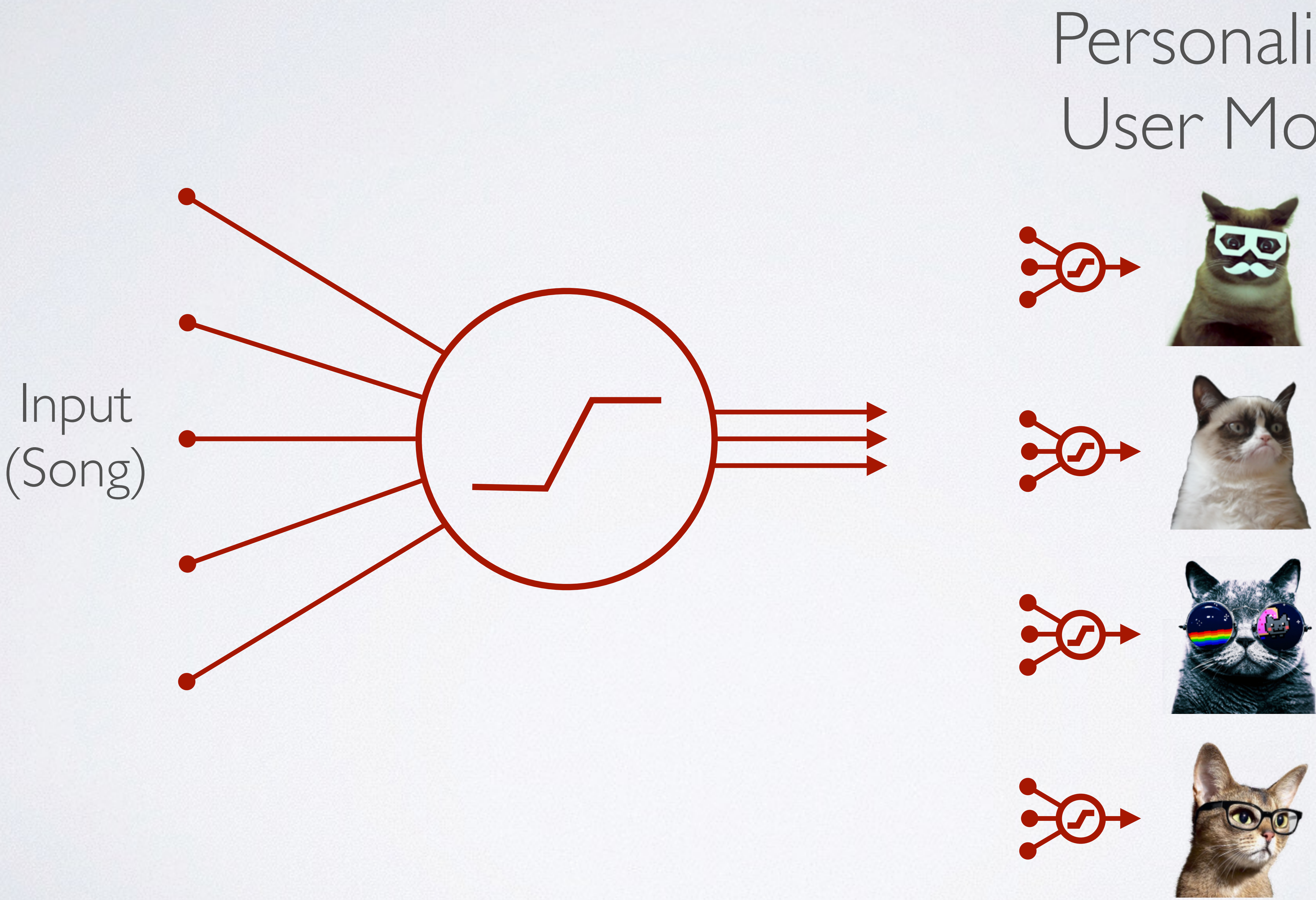

Input
(Song)
Rating


Input
(Song)
Rating

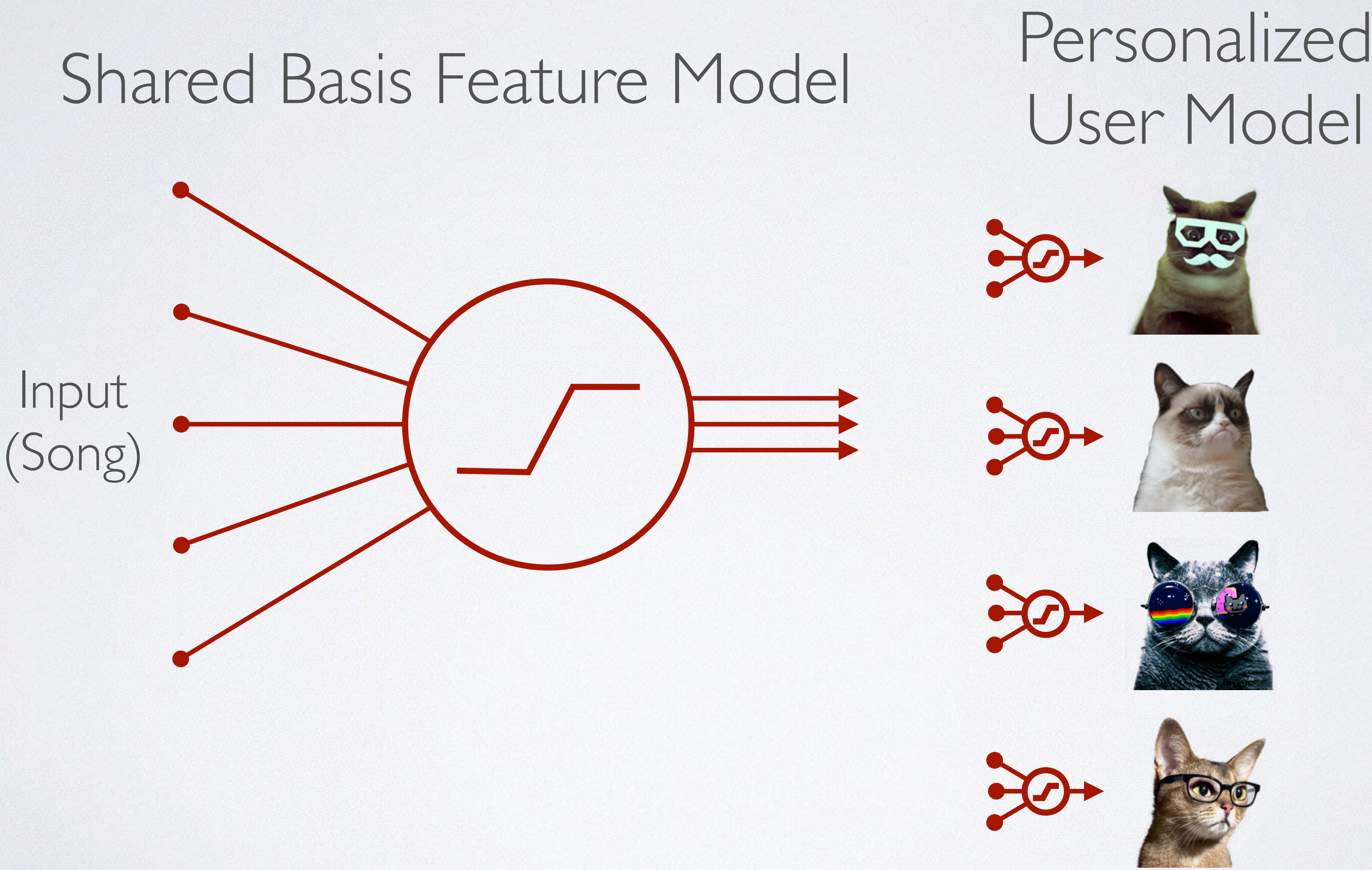

Input
(Song)
Rating
Split


Input
(Song)
Split
Rating

PERSONALIZED MODELING



PERSONALIZED MODELING



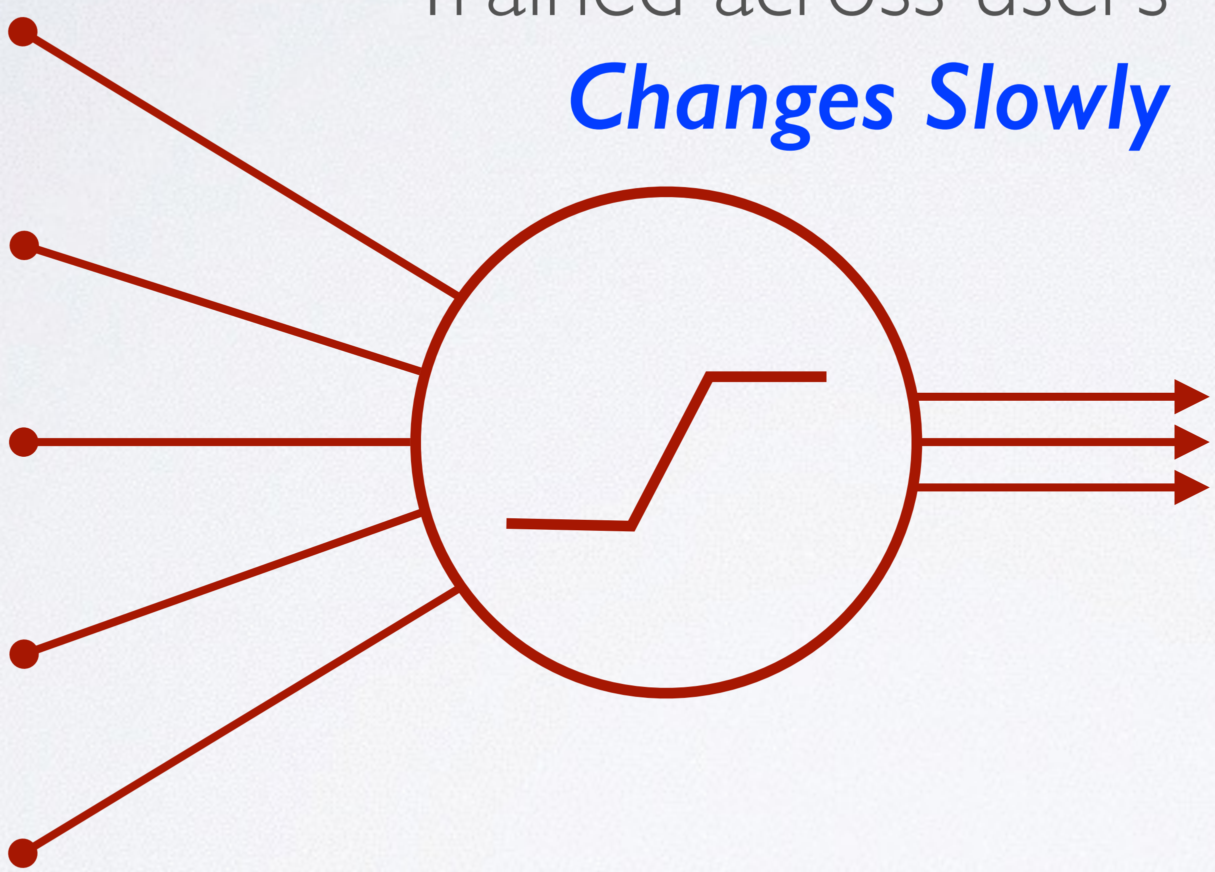
PERSONALIZED MODELING

Shared Basis Feature Model

Trained across users

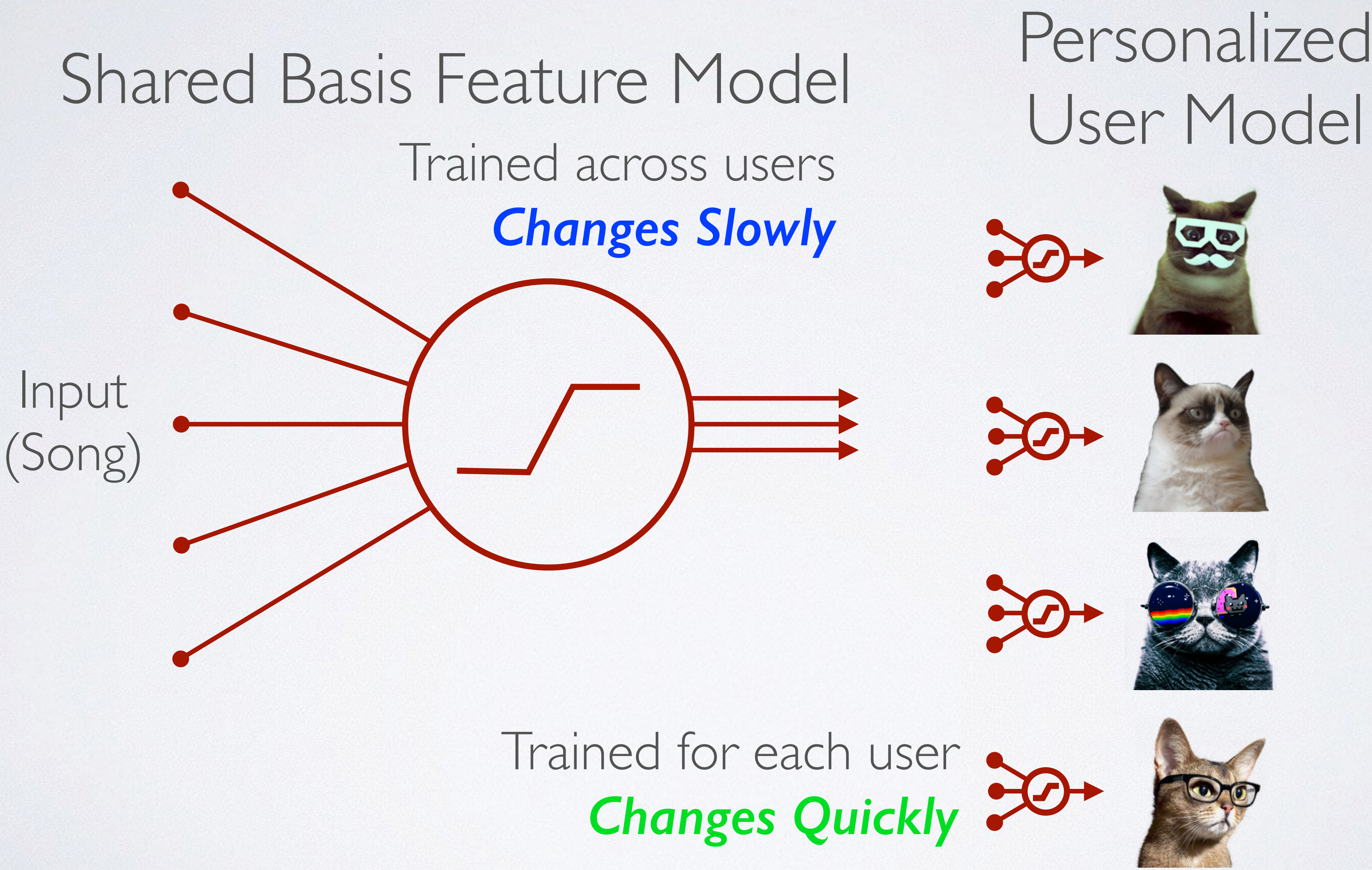
Changes Slowly

Input
(Song)

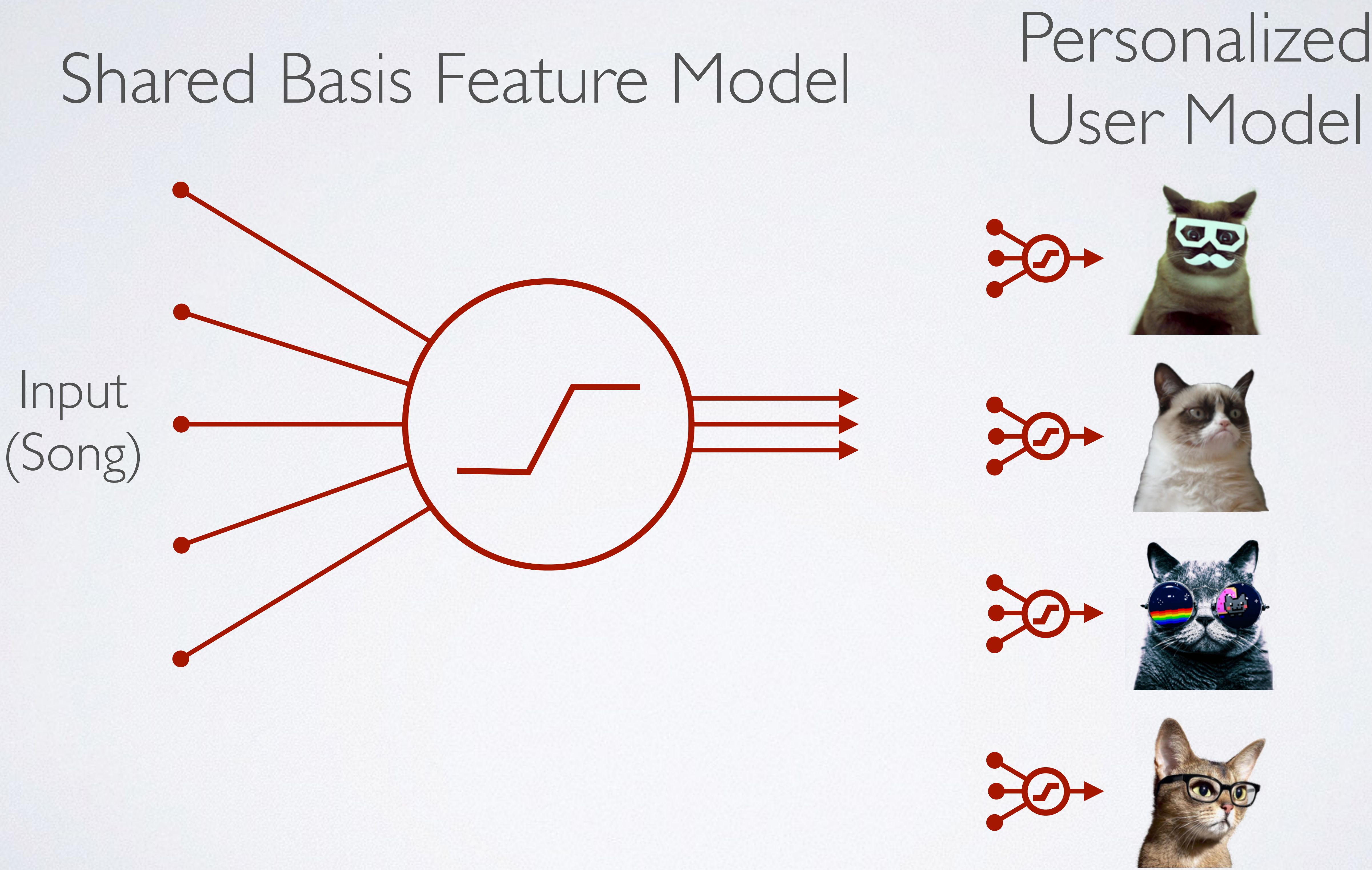


Personalized User Model

PERSONALIZED MODELING



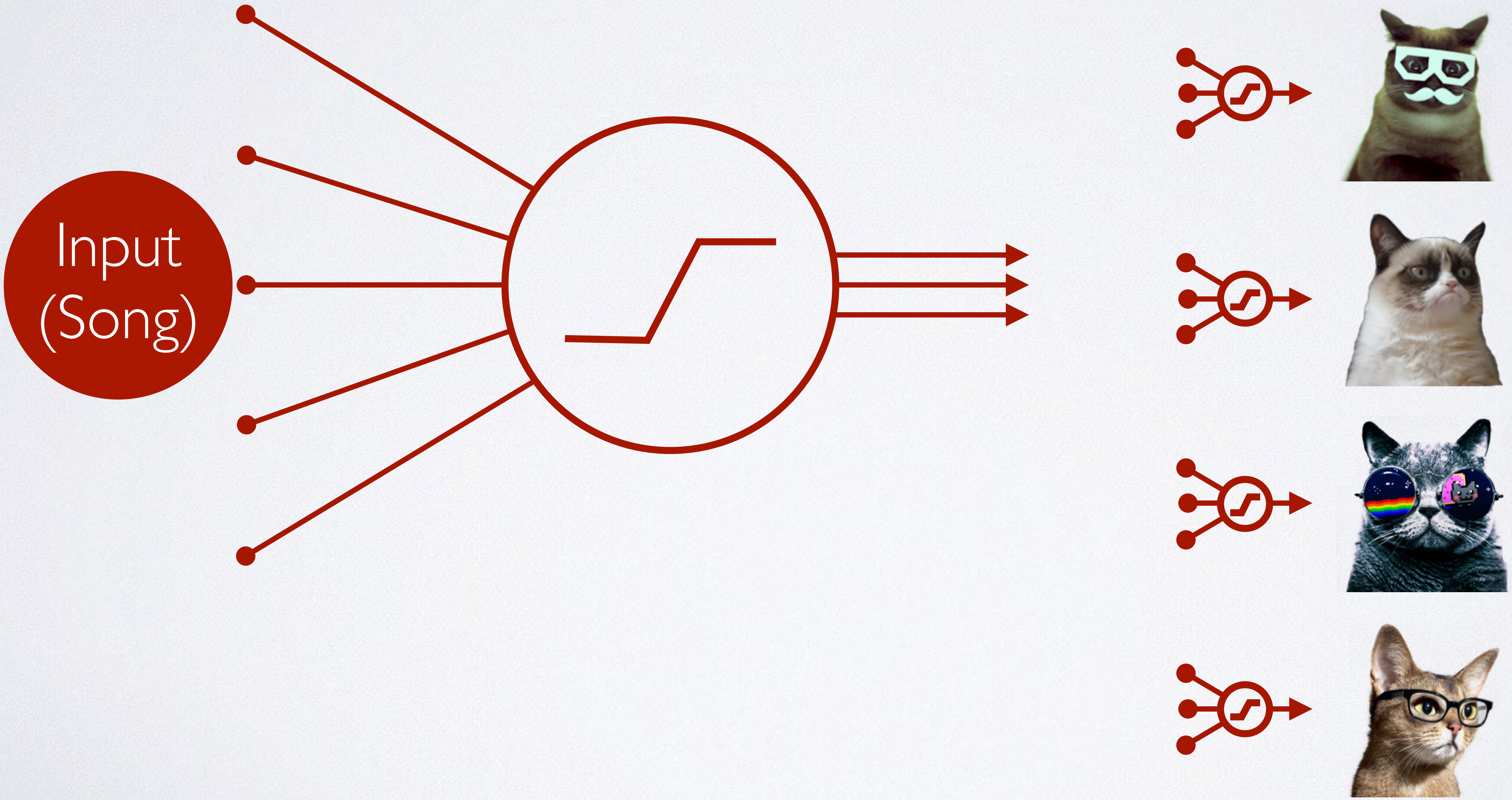
SPLIT MODEL FORMULATION



SPLIT MODEL FORMULATION

Shared Basis Feature Model

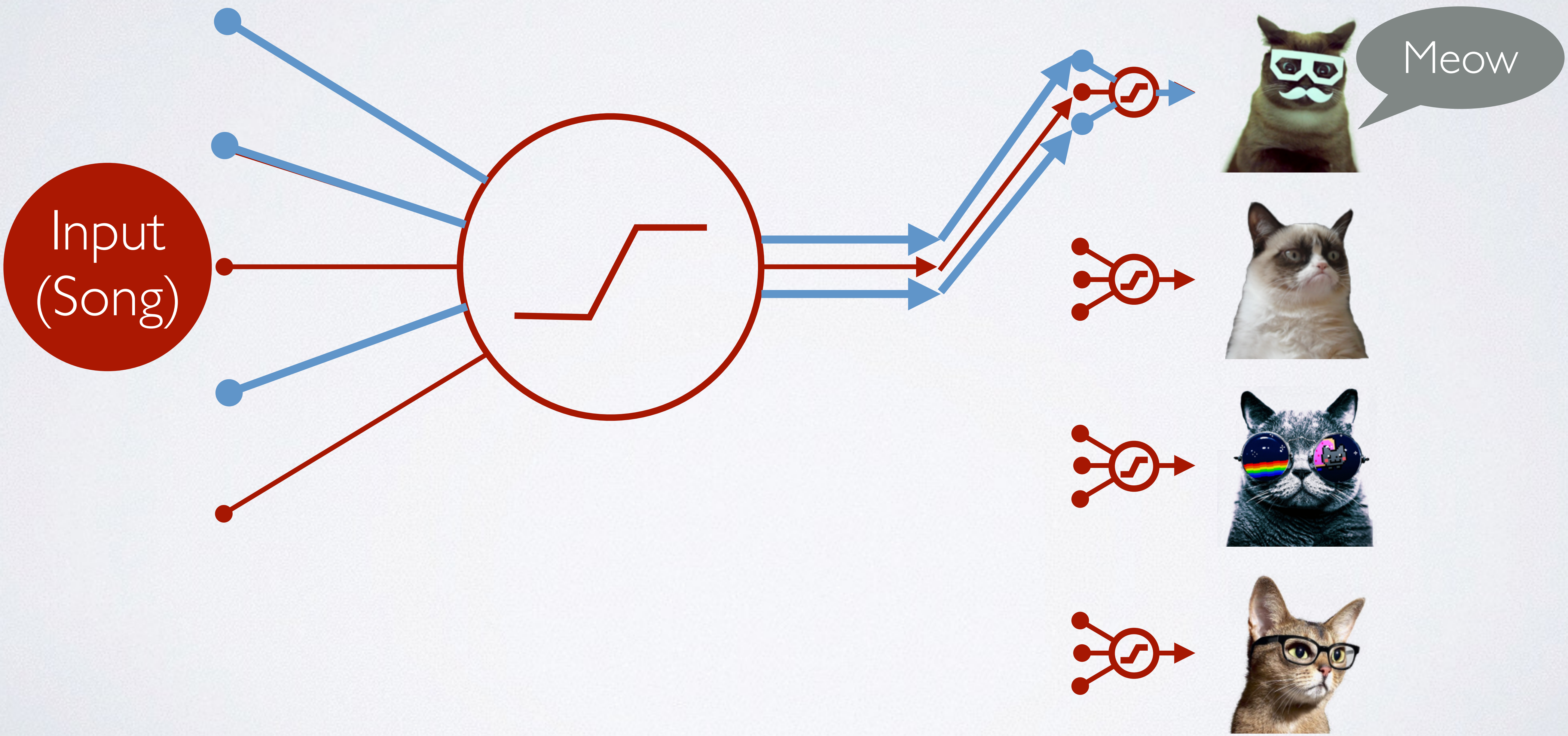
Personalized User Model



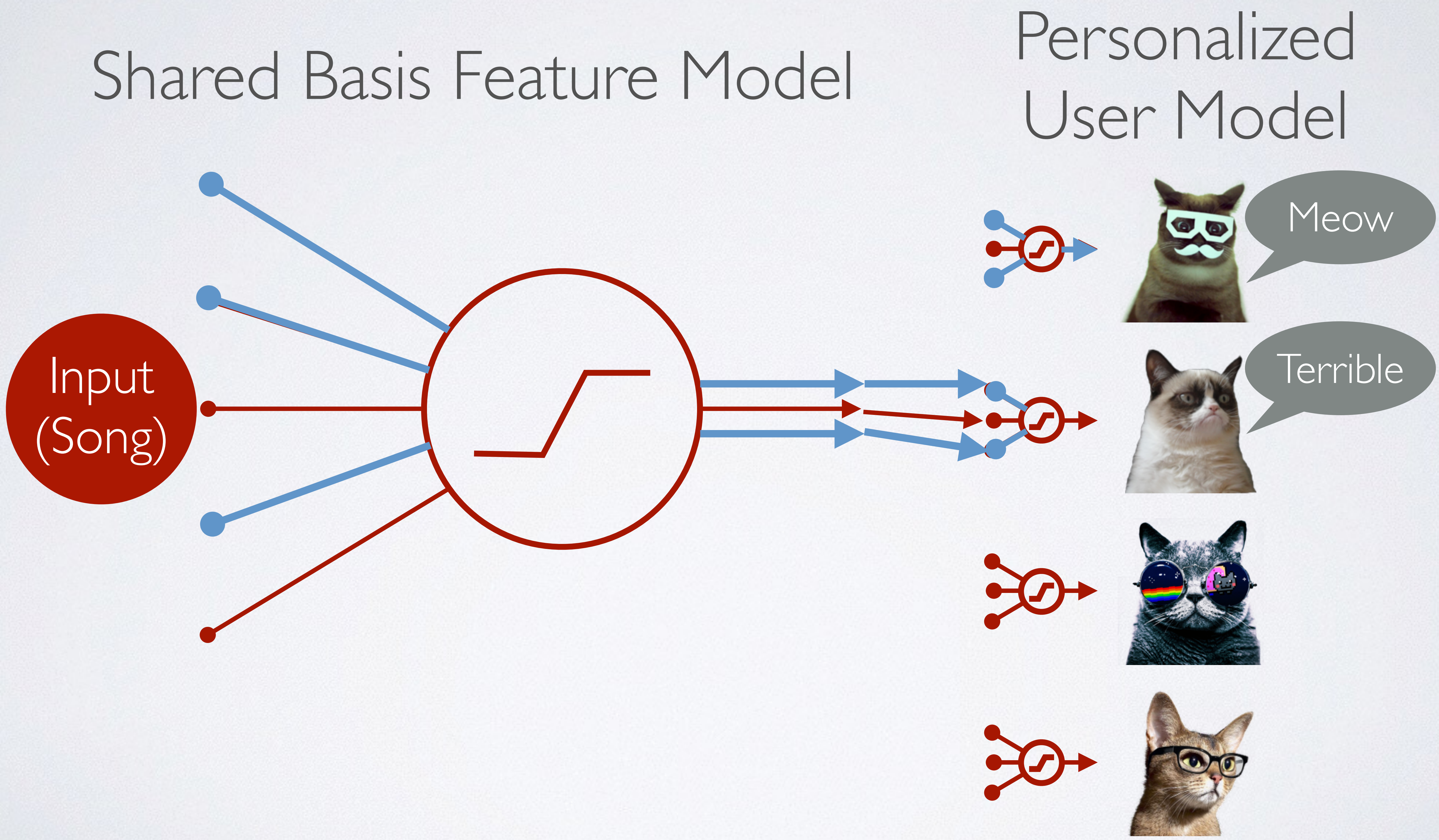
SPLIT MODEL FORMULATION

Shared Basis Feature Model

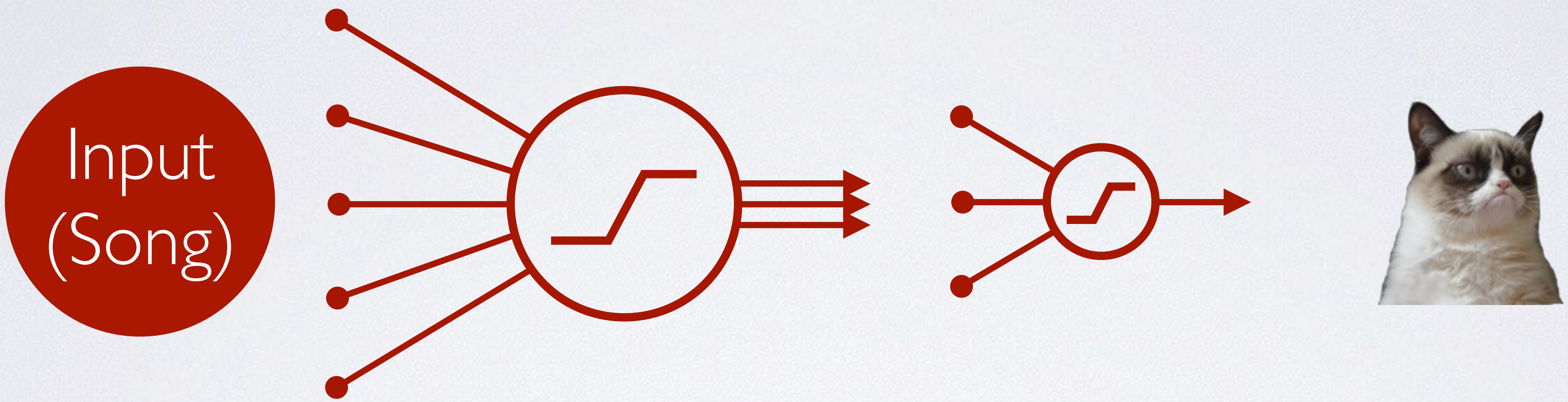
Personalized User Model



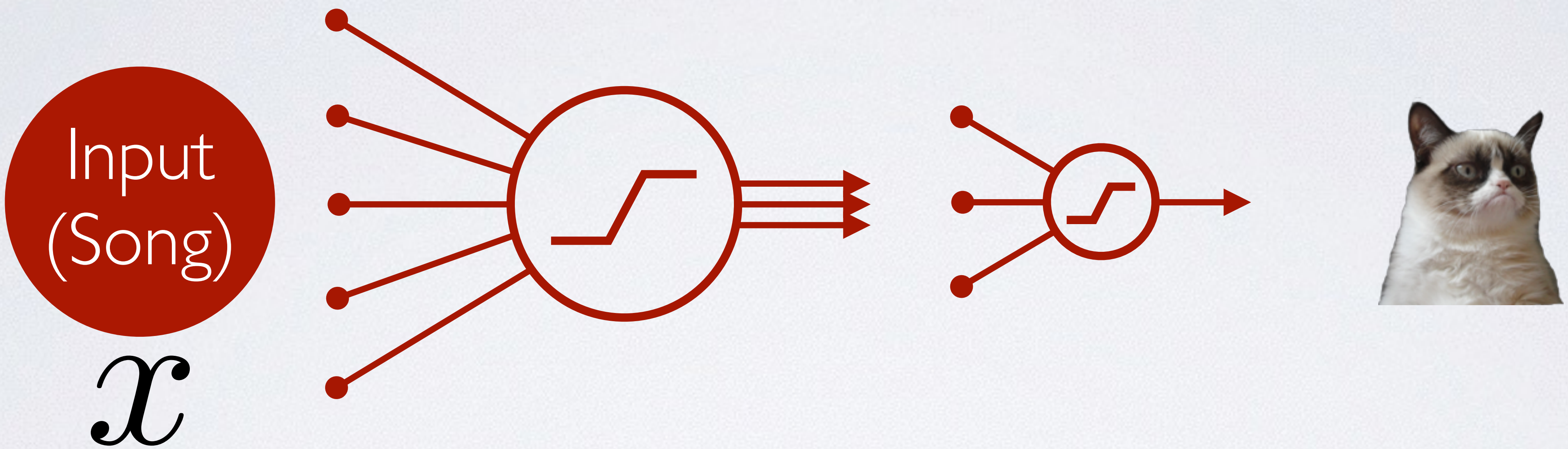
SPLIT MODEL FORMULATION



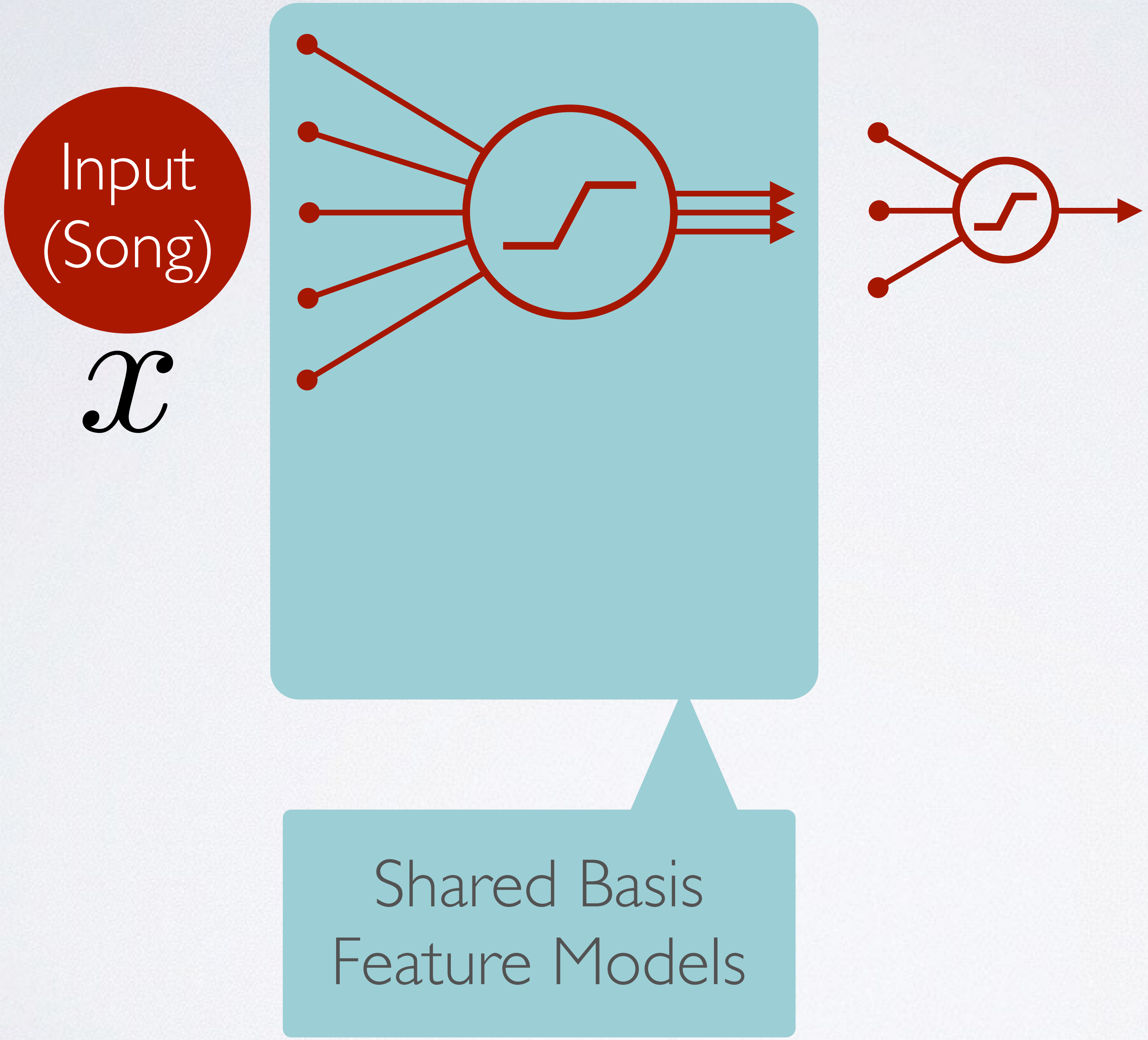
MATHEMATICAL FORMULATION



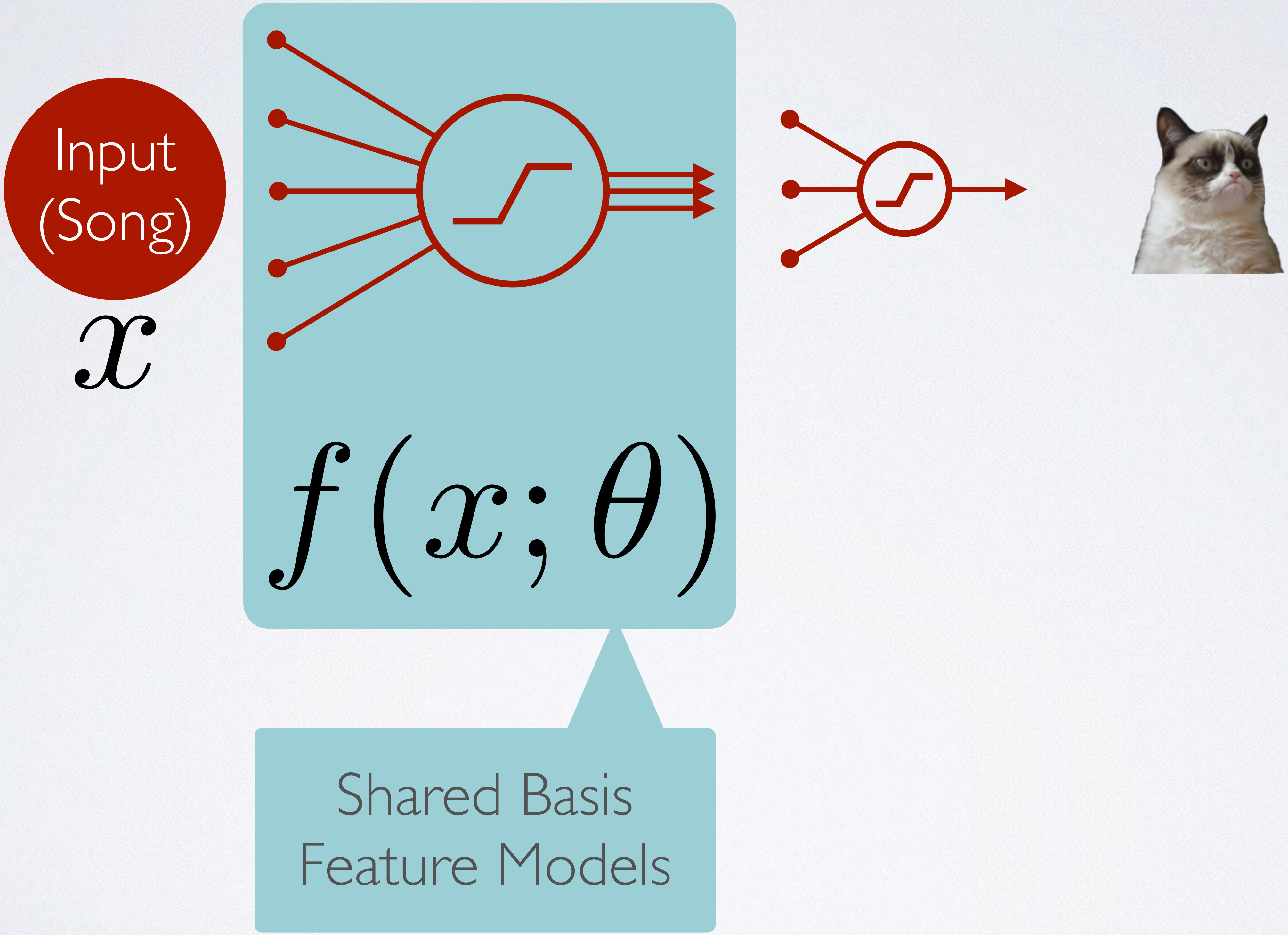
MATHEMATICAL FORMULATION



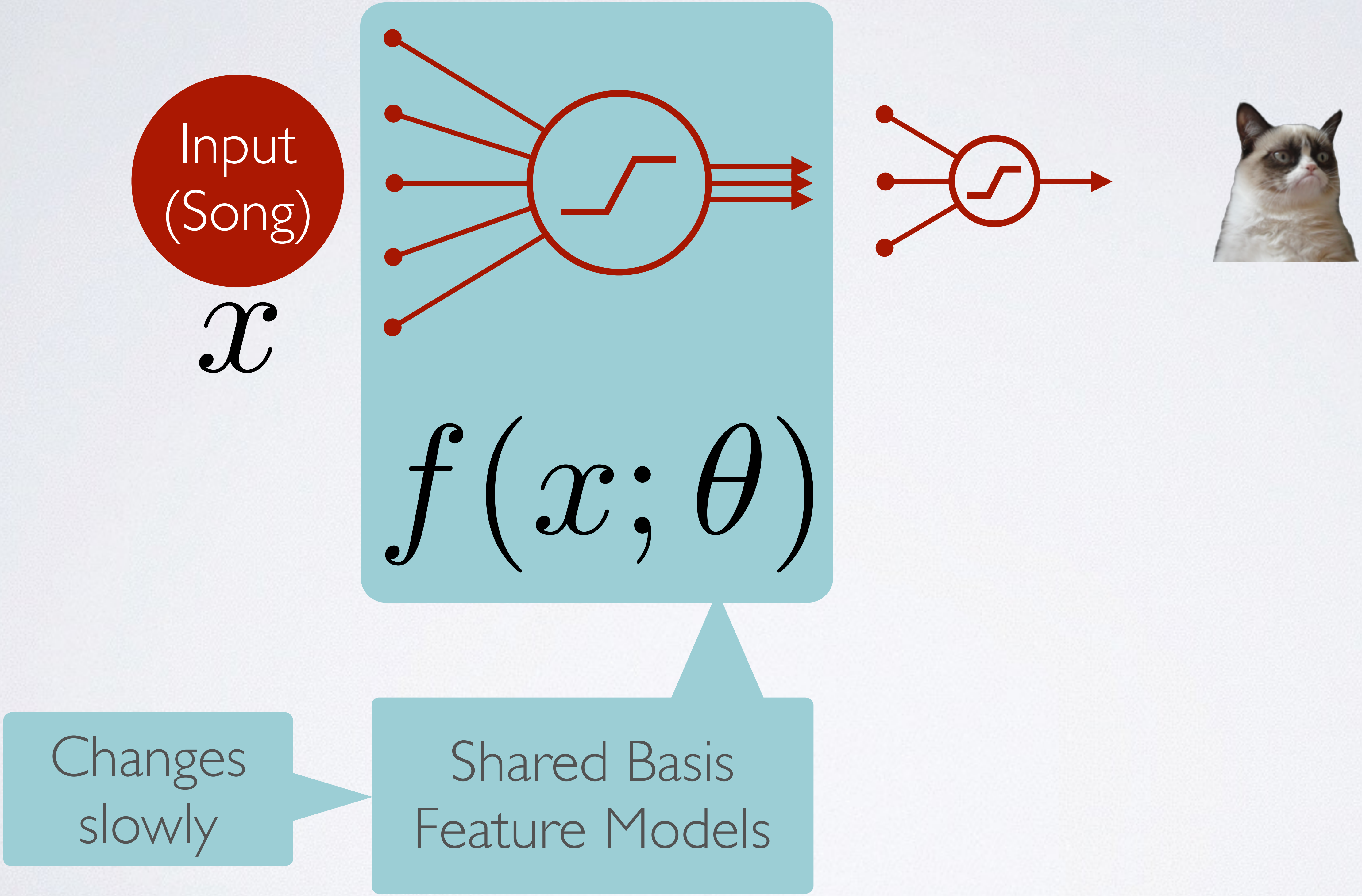
MATHEMATICAL FORMULATION



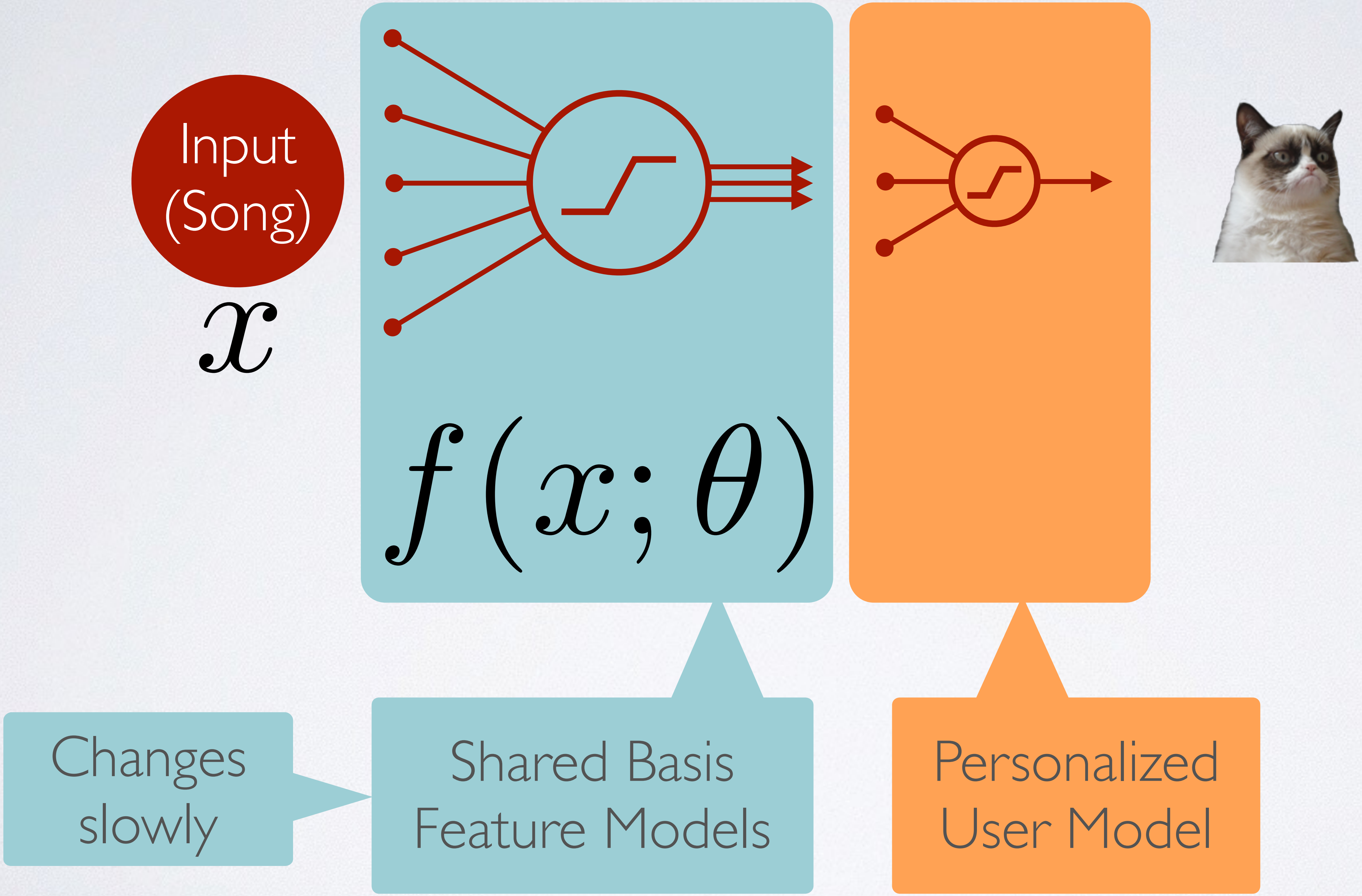
MATHEMATICAL FORMULATION



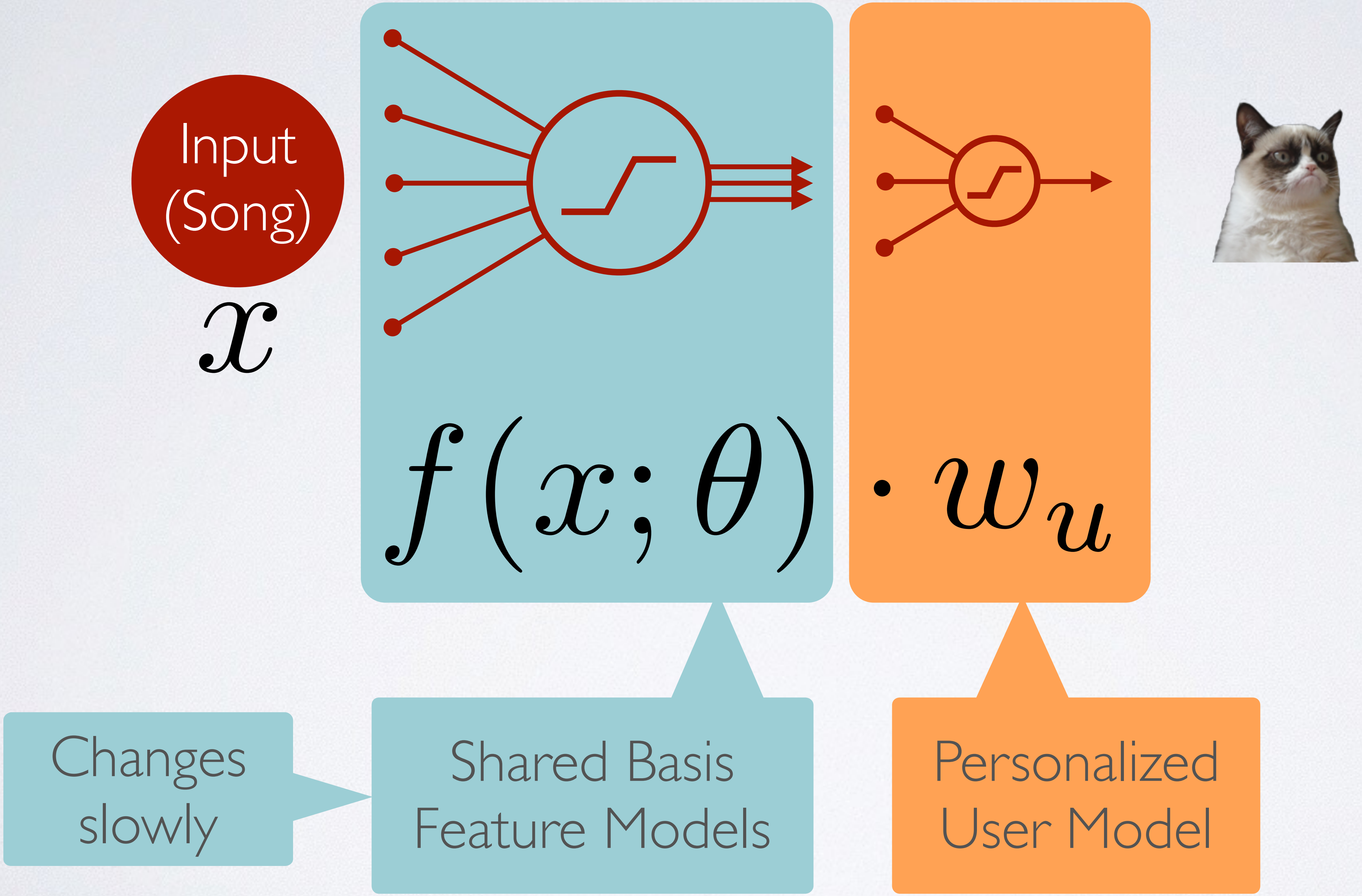
MATHEMATICAL FORMULATION



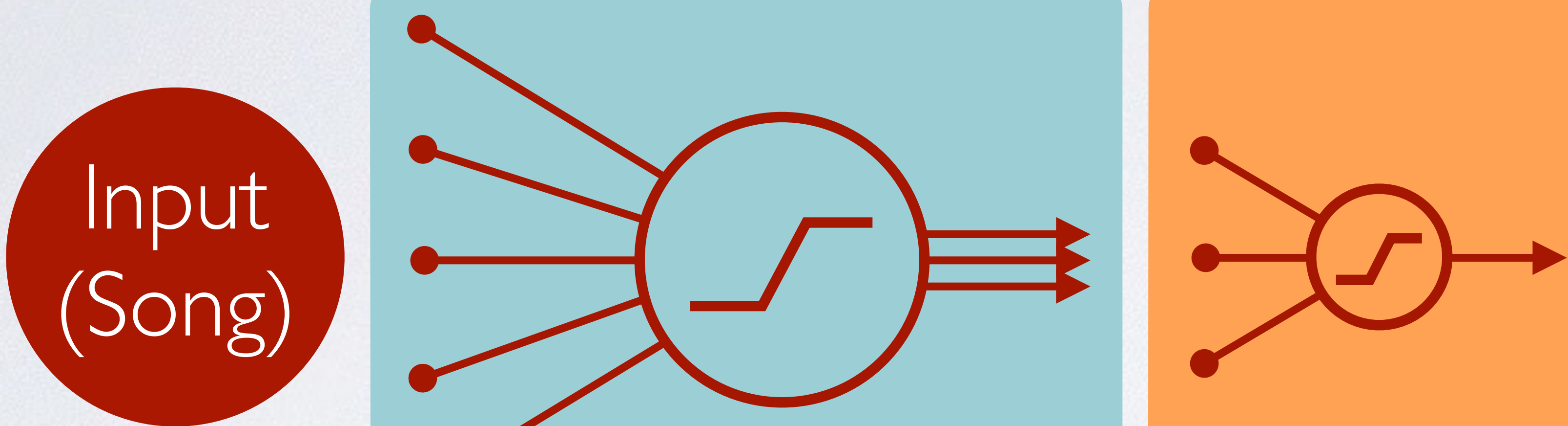
MATHEMATICAL FORMULATION



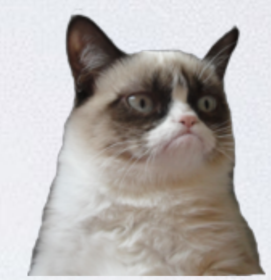
MATHEMATICAL FORMULATION



MATHEMATICAL FORMULATION



x



$$f(x;\theta) \cdot w_u$$

Changes slowly

Shared Basis Feature Models

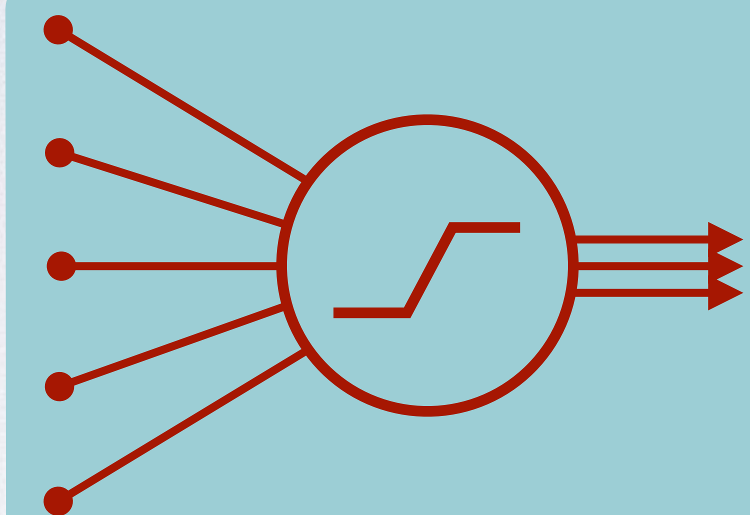
Personalized User Model

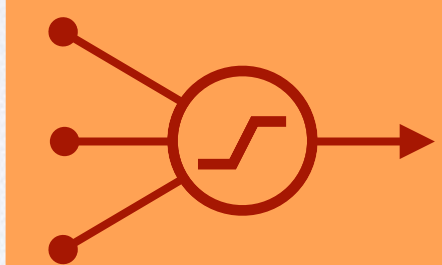
Highly dynamic

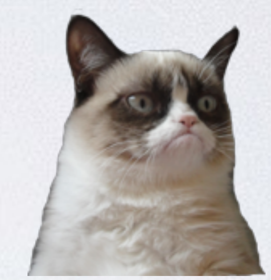
MATHEMATICAL FORMULATION

Input (Song)

x







$$f(x;\theta) \cdot w_u = \text{Rating}$$

Changes slowly

Shared Basis Feature Models

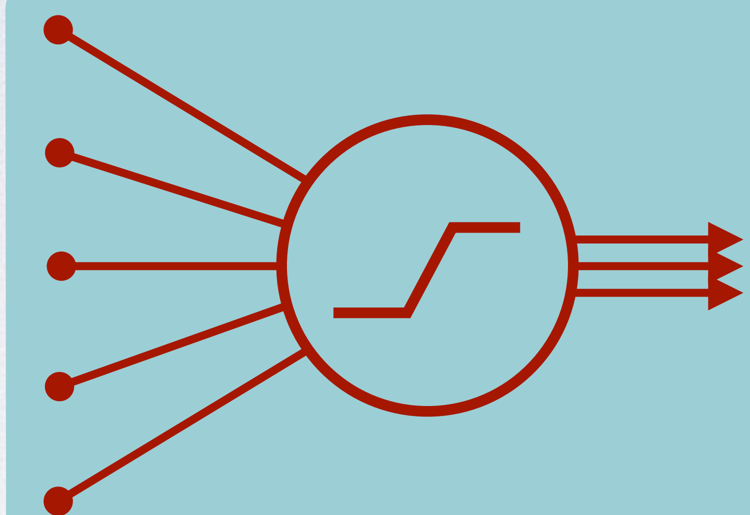
Personalized User Model

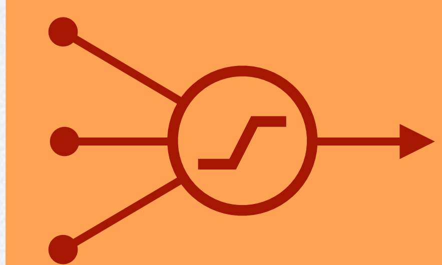
Highly dynamic

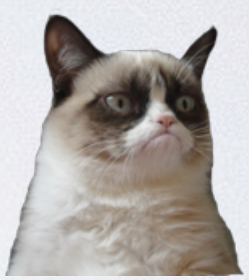
MATHEMATICAL FORMULATION

Input (Song)

x







Terrible

$$f(x;\theta) \cdot w_u = \text{Rating}$$

Changes slowly

Shared Basis Feature Models

Personalized User Model

Highly dynamic

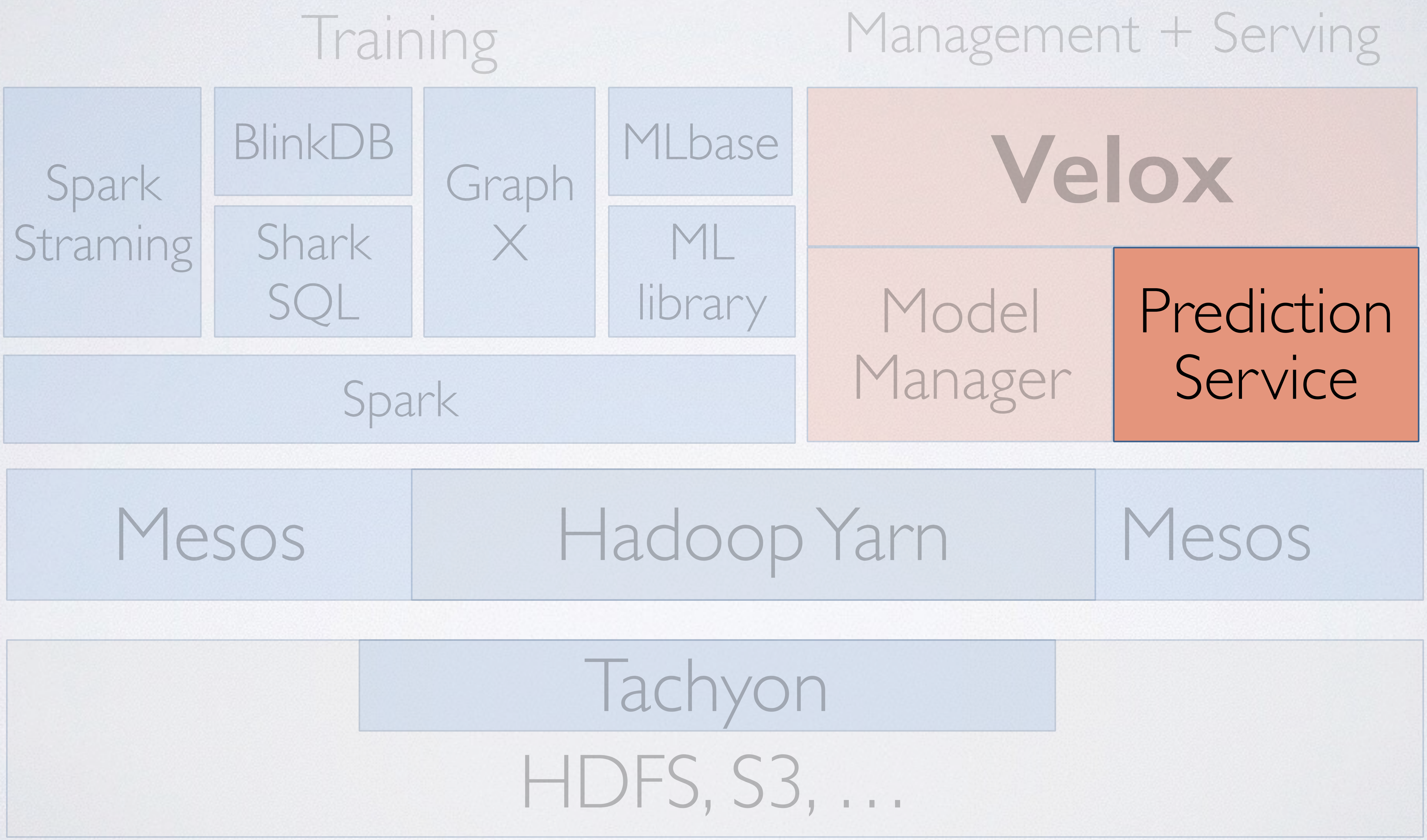
Talk Outline

- ML model management today
- Velox system architecture
- **Key idea: Split model family**
- Prediction serving
- Model management
- Next directions

Talk Outline

- ML model management today
- Velox system architecture
- Key idea: Split model family
- **Prediction serving**
- Model management
- Next directions

System Architecture



PREDICTION API

Simple point queries:

```
GET /velox/catify/predict?userid=22&song=27632
```

PREDICTION API

Simple point queries:

```
GET /velox/catify/predict?userid=22&song=27632
```

More complex ordering queries:

```
GET /velox/catify/predict_top_k?userid=22&k=100
```

PREDICTIONS

```
def predict( u: UUID, x: Context )
```

$$w_u \cdot f(x;\theta)$$

PREDICTIONS

```
def predict( u: UUID, x: Context )
```

Look up user weight

$$w_u \cdot f(x;\theta)$$

PREDICTIONS

```
def predict( u: UUID, x: Context )
```

Look up user weight

Primary key lookup

$$w_u \cdot f(x; \theta)$$

PREDICTIONS

```
def predict( u: UUID, x: Context )
```

Look up user weight

Primary key lookup

Partition query by user: *always local*

$$w_u \cdot f(x;\theta)$$

PREDICTIONS

```
def predict( u: UUID, x: Context )
```

Compute Features

$$w_u \cdot f(x;\theta)$$

user independent

PREDICTIONS

```
def predict( u: UUID, x: Context )
```

Feature computation could be costly

Compute Features

$$w_u \cdot f(x;\theta)$$

user independent

PREDICTIONS

```
def predict( u: UUID, x: Context )
```

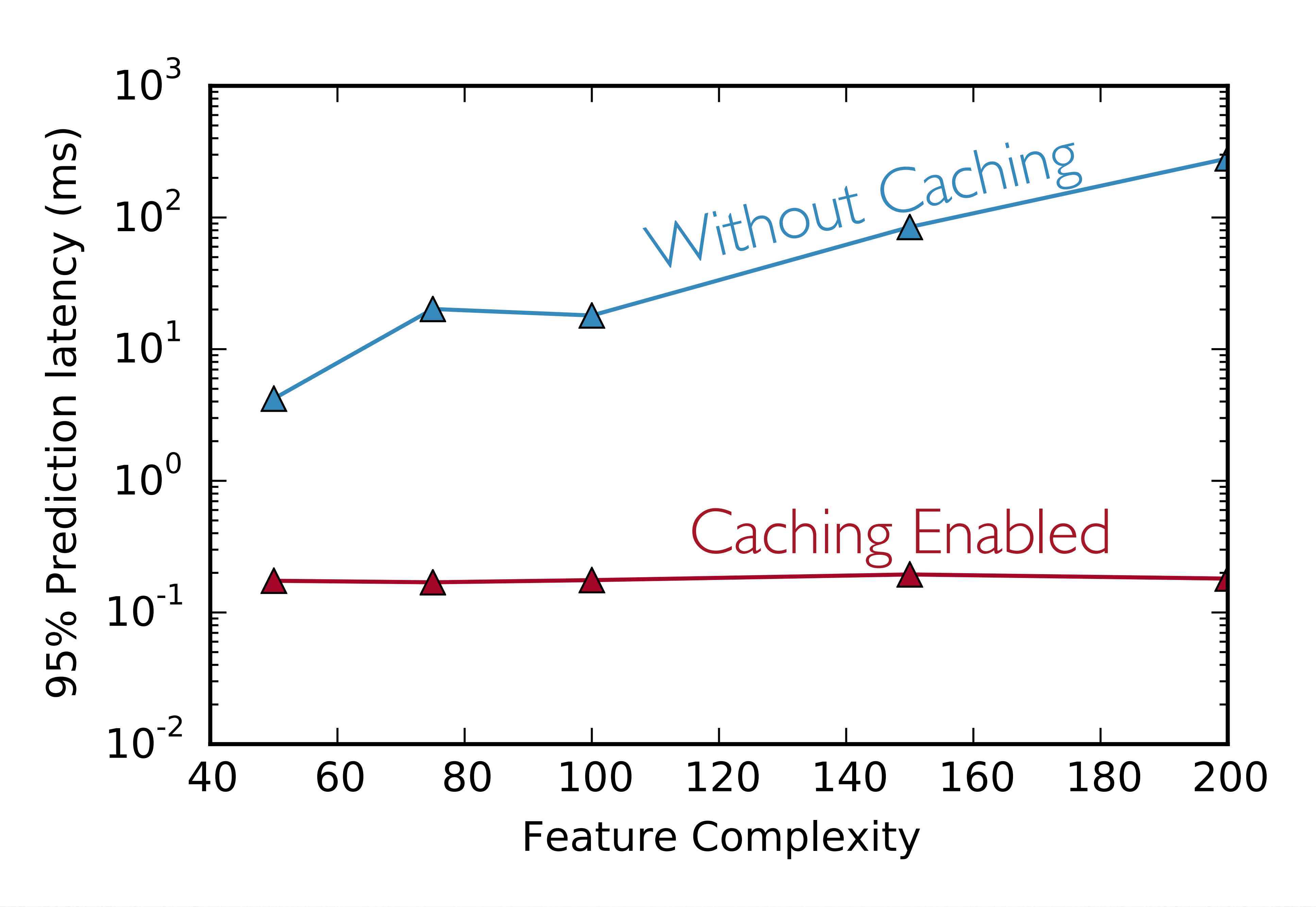
Feature computation could be costly

Compute Features

Cache features for reuse across users

$$w_u \cdot \underbrace{f(x;\theta)}_{\text{user independent}}$$

FEATURE CACHING GAINS



Feature caching leads to order-of-magnitude reduction in latency.

TOP-K QUERIES

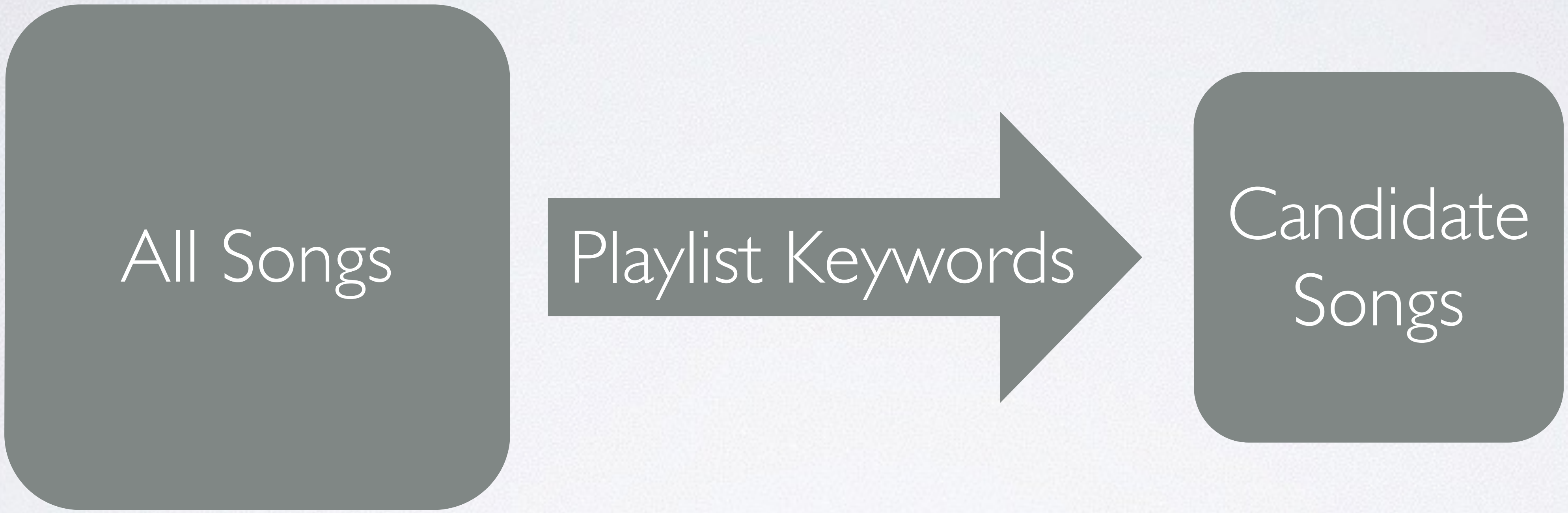
Query predicate to pre-filter candidate set

TOP-K QUERIES

Query predicate to pre-filter candidate set

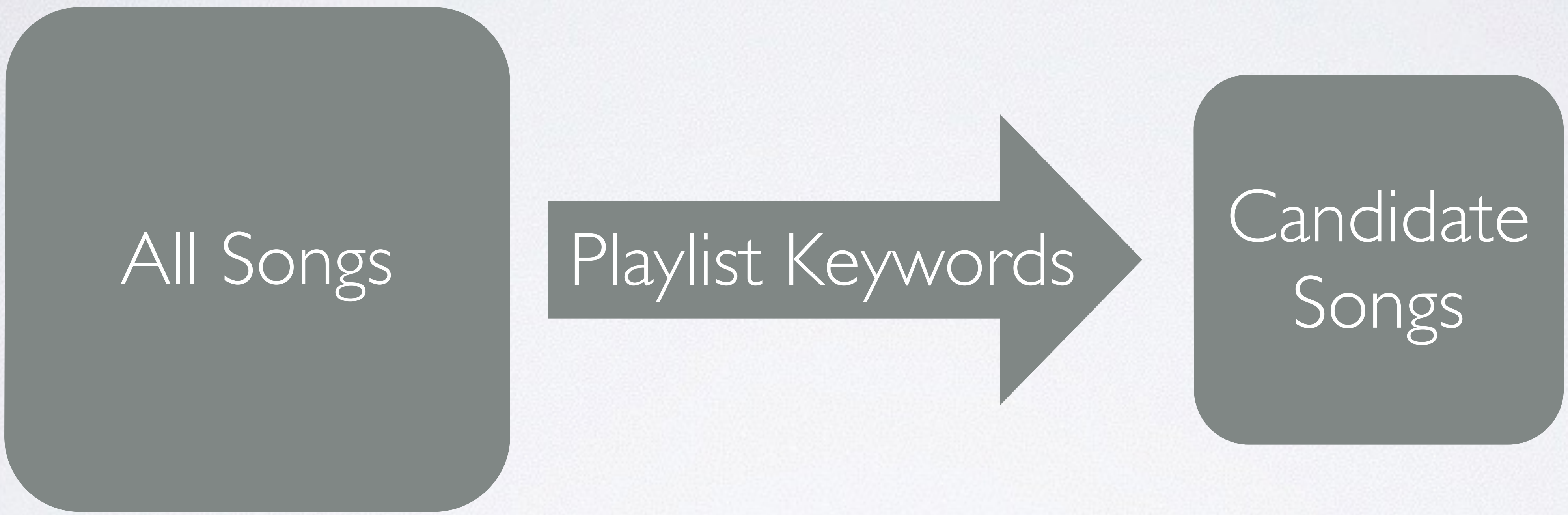
TOP-K QUERIES

Query predicate to pre-filter candidate set



TOP-K QUERIES

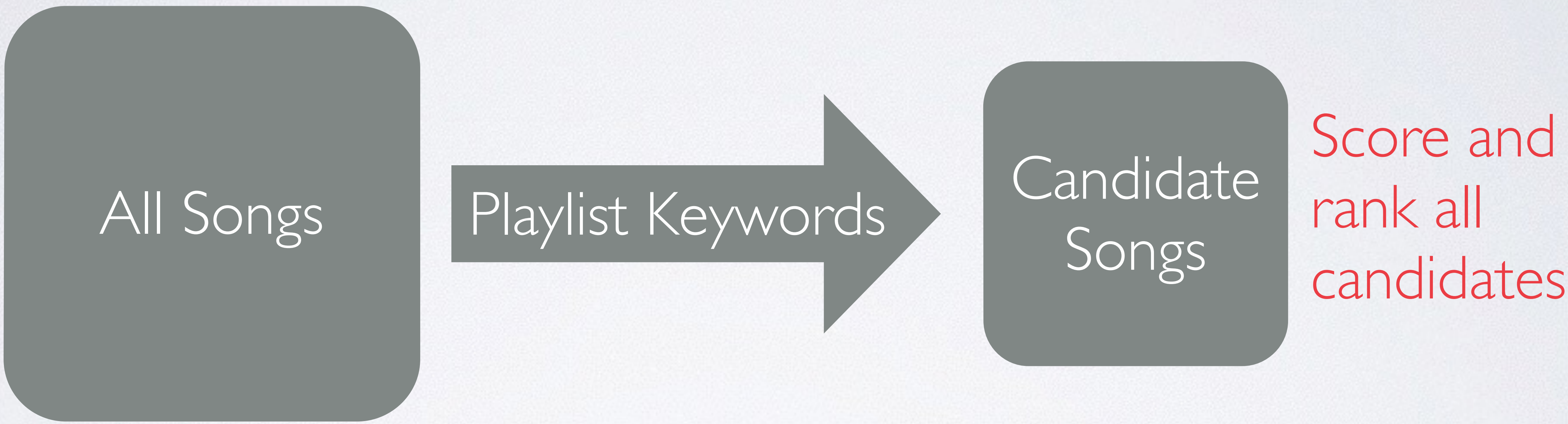
Query predicate to pre-filter candidate set



Score and rank all candidates

TOP-K QUERIES

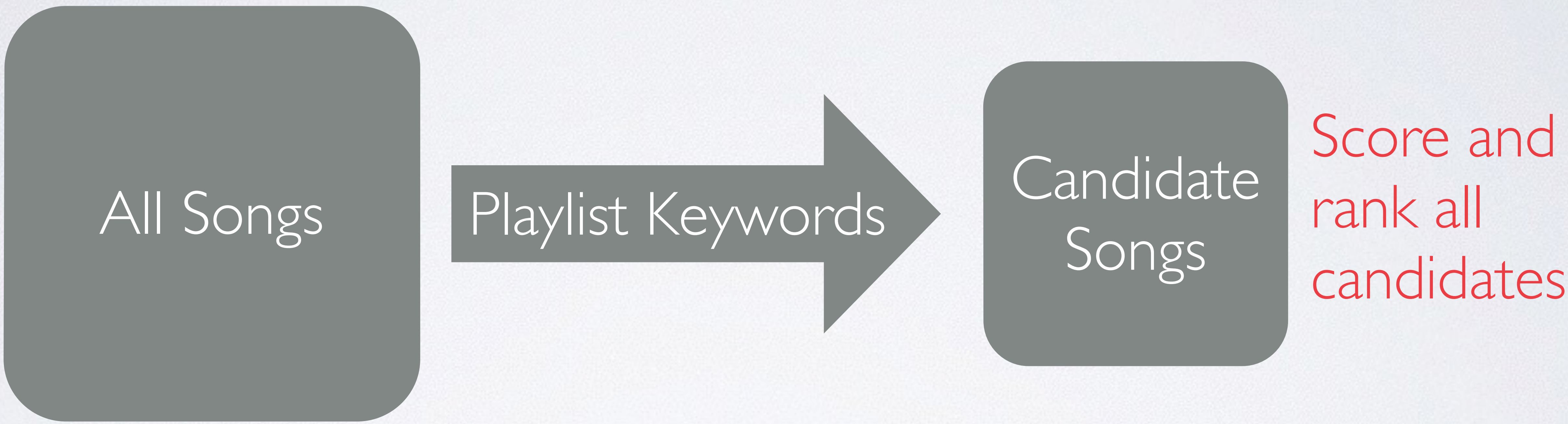
Query predicate to pre-filter candidate set



By exploiting split model design we can leverage:

TOP-K QUERIES

Query predicate to pre-filter candidate set

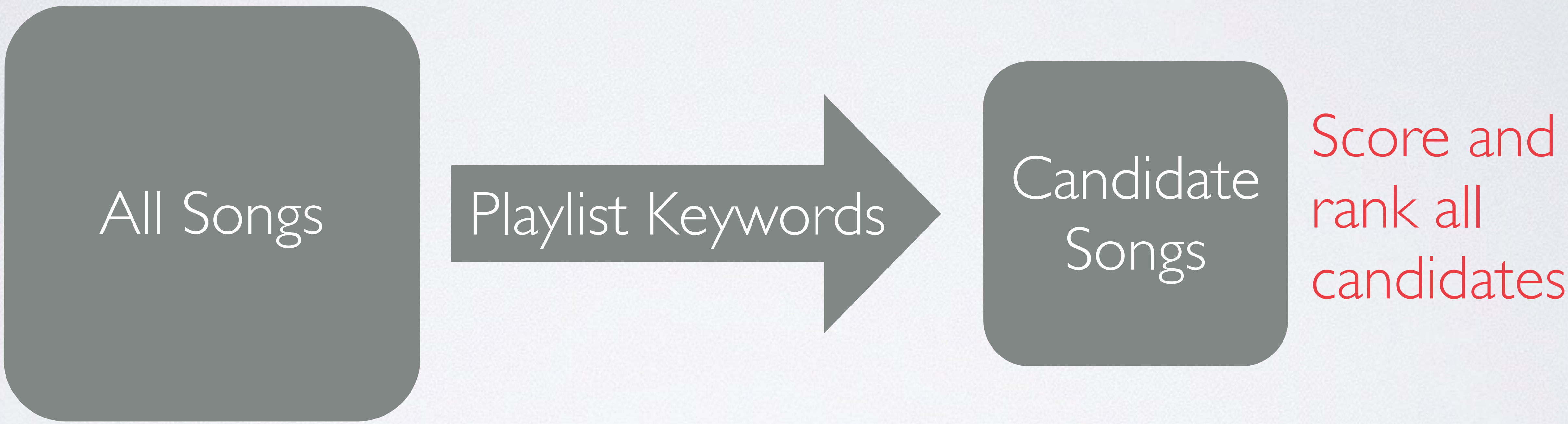


By exploiting split model design we can leverage:

A. Shrivastava, P. Li. *"Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS)."* NIPS'14 Best Paper

TOP-K QUERIES

Query predicate to pre-filter candidate set



By exploiting split model design we can leverage:

A. Shrivastava, P. Li. *"Asymmetric LSH (ALSH) for Sublinear Time Maximum Inner Product Search (MIPS)."* NIPS'14 Best Paper

Y. Low and A. X. Zheng. *"Fast Top-K Similarity Queries Via Matrix Compression."* CIKM 2012

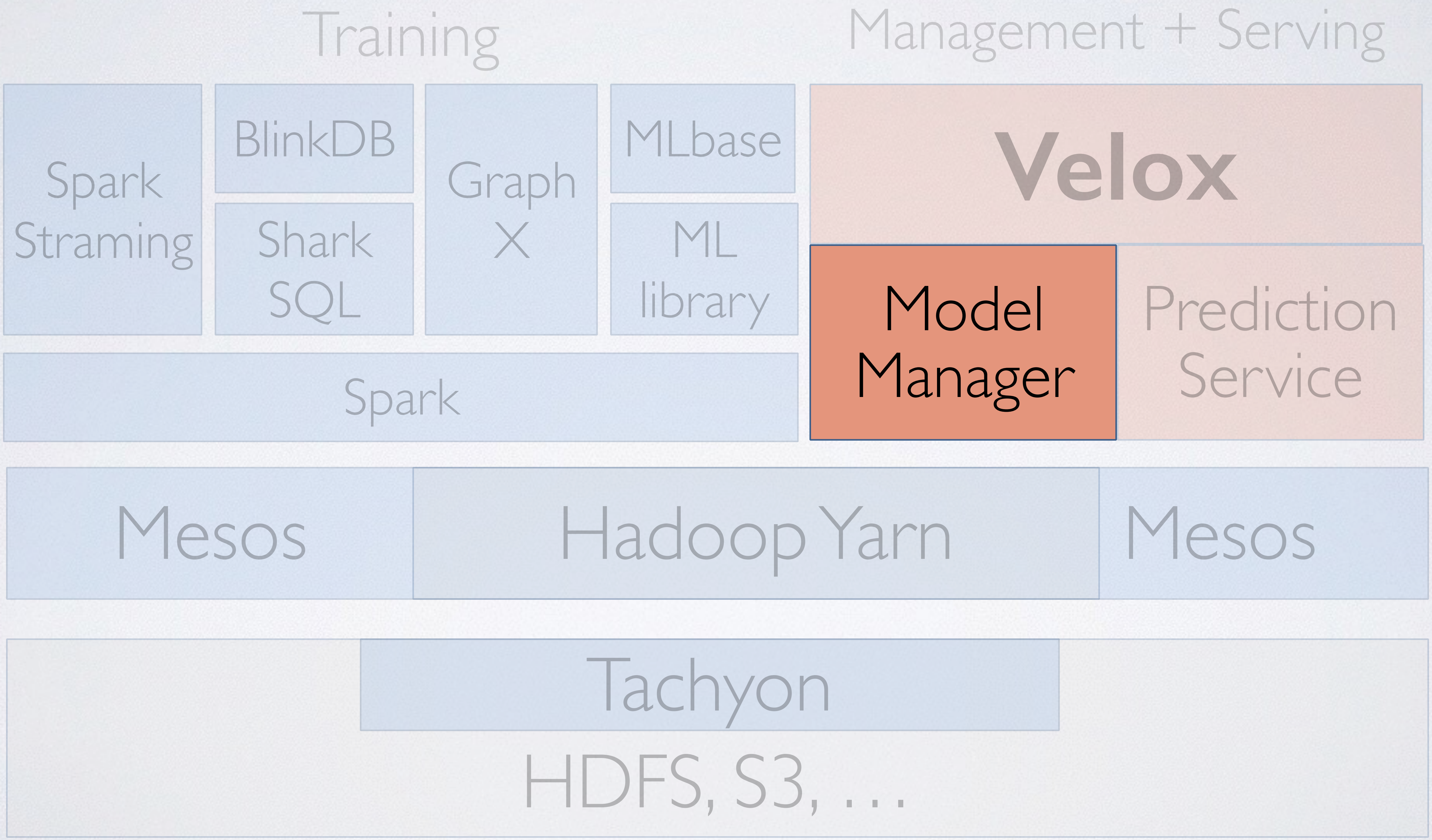
Talk Outline

- ML model management today
- Velox system architecture
- Key idea: Split model family
- **Prediction serving**
- Model management
- Next directions

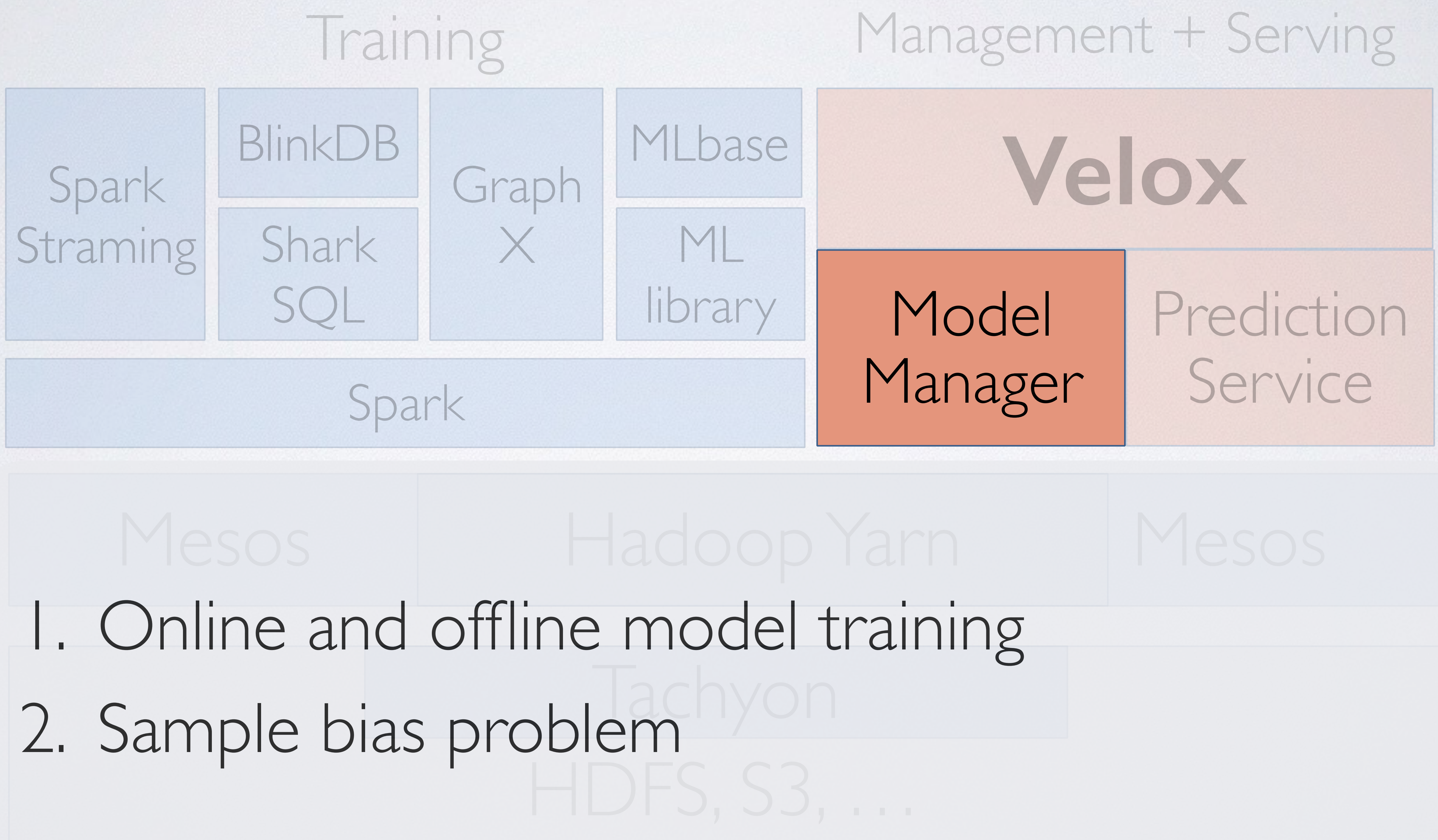
Talk Outline

- ML model management today
- Velox system architecture
- Key idea: Split model family
- Prediction serving
- **Model management**
- Next directions

System Architecture



System Architecture



1. Online and offline model training
2. Sample bias problem

FEEDBACK API

Simple direct value feedback:

```
POST /velox/catify/observe?userid=22&song=27&score=3.7
```

FEEDBACK API

Simple direct value feedback:

```
POST /velox/catify/observe?userid=22&song=27&score=3.7
```

Online Learning

Continuously update
user models in Velox

FEEDBACK API

Simple direct value feedback:

```
POST /velox/catify/observe?userid=22&song=27&score=3.7
```

Online Learning

Continuously update user models in Velox

Offline Learning

Logged to DFS for feature learning in Spark

FEEDBACK API

Simple direct value feedback:

```
POST /velox/catify/observe?userid=22&song=27&score=3.7
```

Online Learning

Continuously update user models in Velox

Offline Learning

Logged to DFS for feature learning in Spark

Evaluation

Continuously assess model performance

ONLINE LEARNING

```
def observe(u: UUID, x: Context, y: Score)
```

$$w_u \cdot f(x;\theta)$$

ONLINE LEARNING

```
def observe(u: UUID, x: Context, y: Score)
```

Update w_u with new training point

$$w_u \cdot f(x;\theta)$$

ONLINE LEARNING

```
def observe(u: UUID, x: Context, y: Score)
```

Update w_u with new training point

Stochastic gradient descent

$$w_u \cdot f(x; \theta)$$

ONLINE LEARNING

```
def observe(u: UUID, x: Context, y: Score)
```

Update w_u with new training point

Stochastic gradient descent
Incremental linear algebra

$$w_u \cdot f(x; \theta)$$

OFFLINE LEARNING

```
def retrain(trainingData: RDD)
```

$$w_u \cdot f(x;\theta)$$

Spark Based Training Algs.

Efficient batch training using Spark

OFFLINE LEARNING

```
def retrain(trainingData: RDD)
```

$$w_u \cdot f(x;\theta)$$

Spark Based Training Algs.

Efficient batch training using Spark

When do we retrain?

OFFLINE LEARNING

```
def retrain(trainingData: RDD)
```

$$w_u \cdot f(x;\theta)$$

Spark Based Training Algs.

Efficient batch training using Spark

When do we retrain?

Periodically

OFFLINE LEARNING

```
def retrain(trainingData: RDD)
```

$$w_u \cdot f(x;\theta)$$

Spark Based Training Algs.

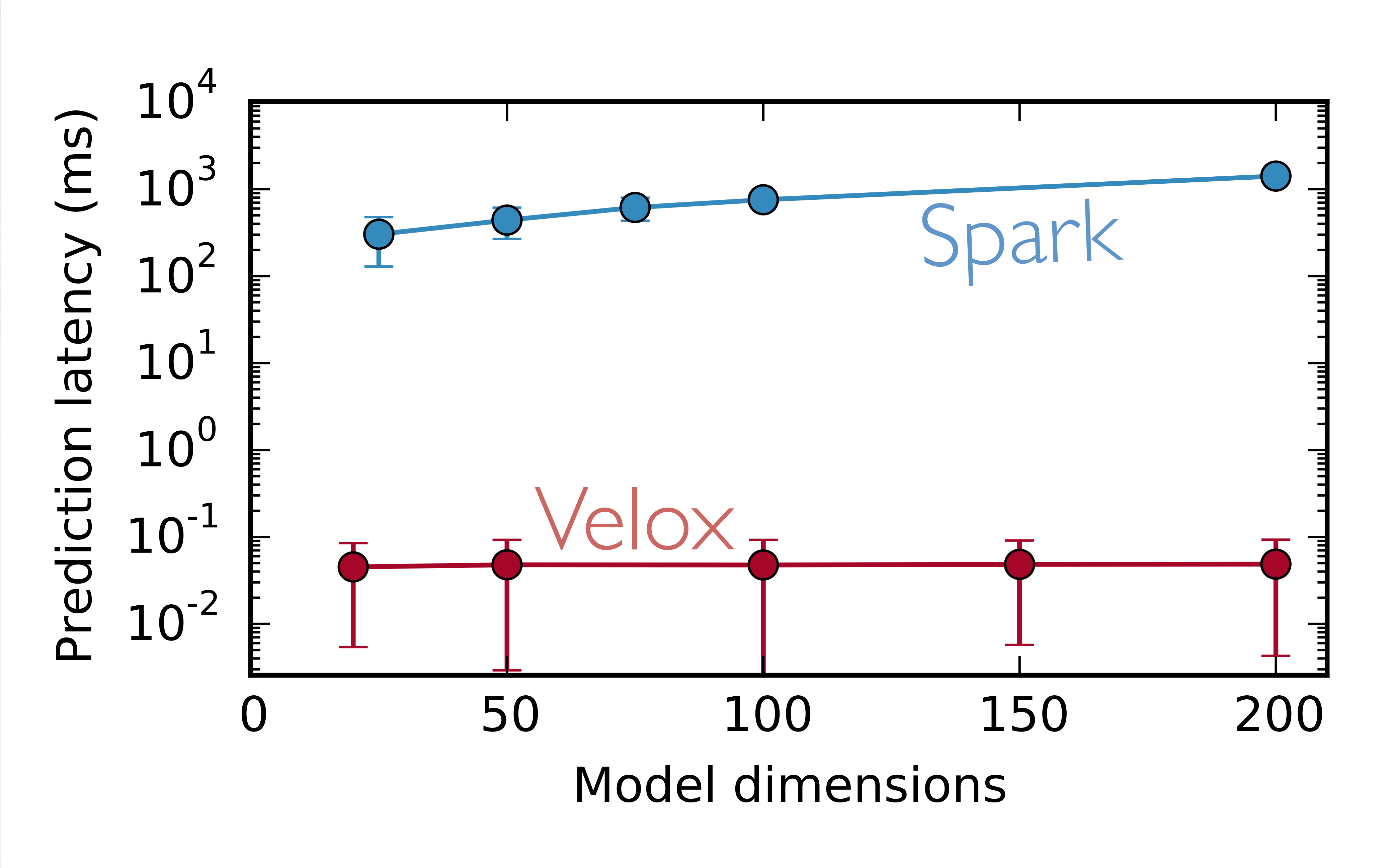
Efficient batch training using Spark

When do we retrain?

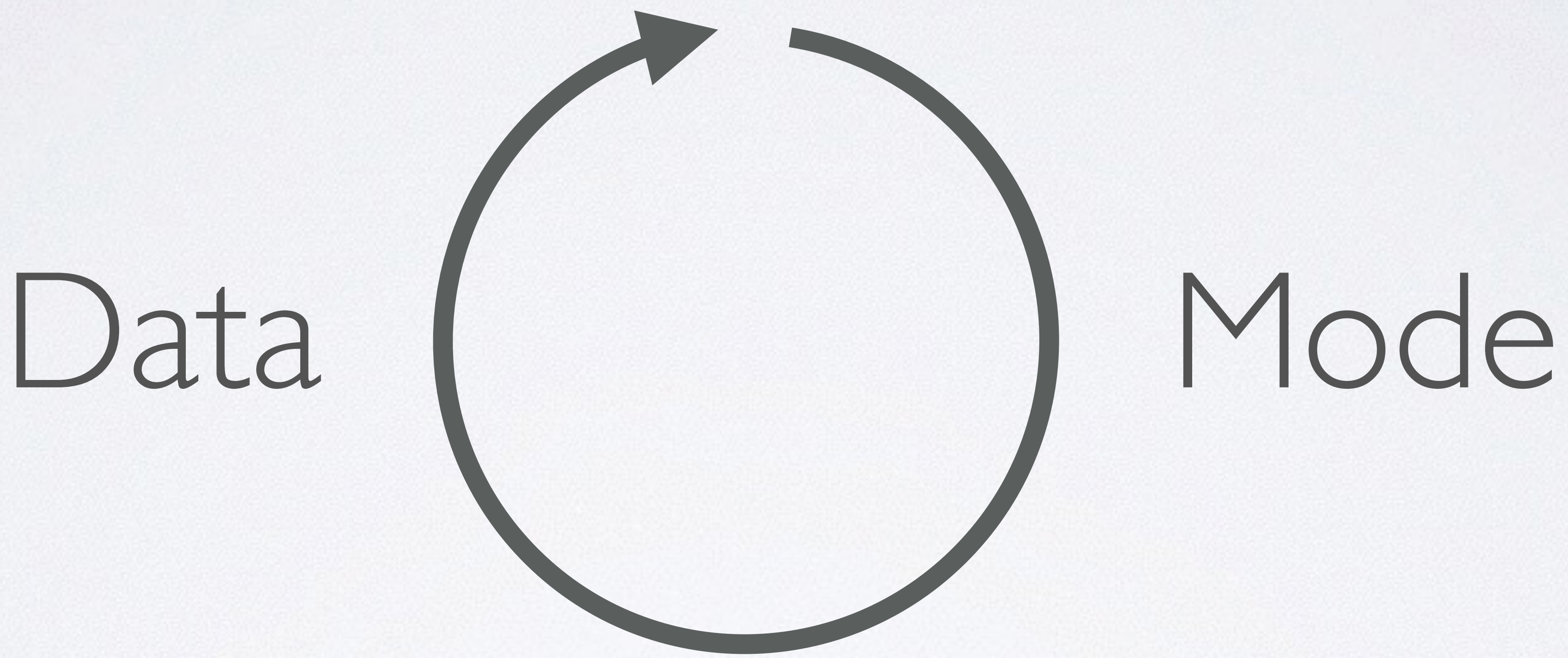
Periodically

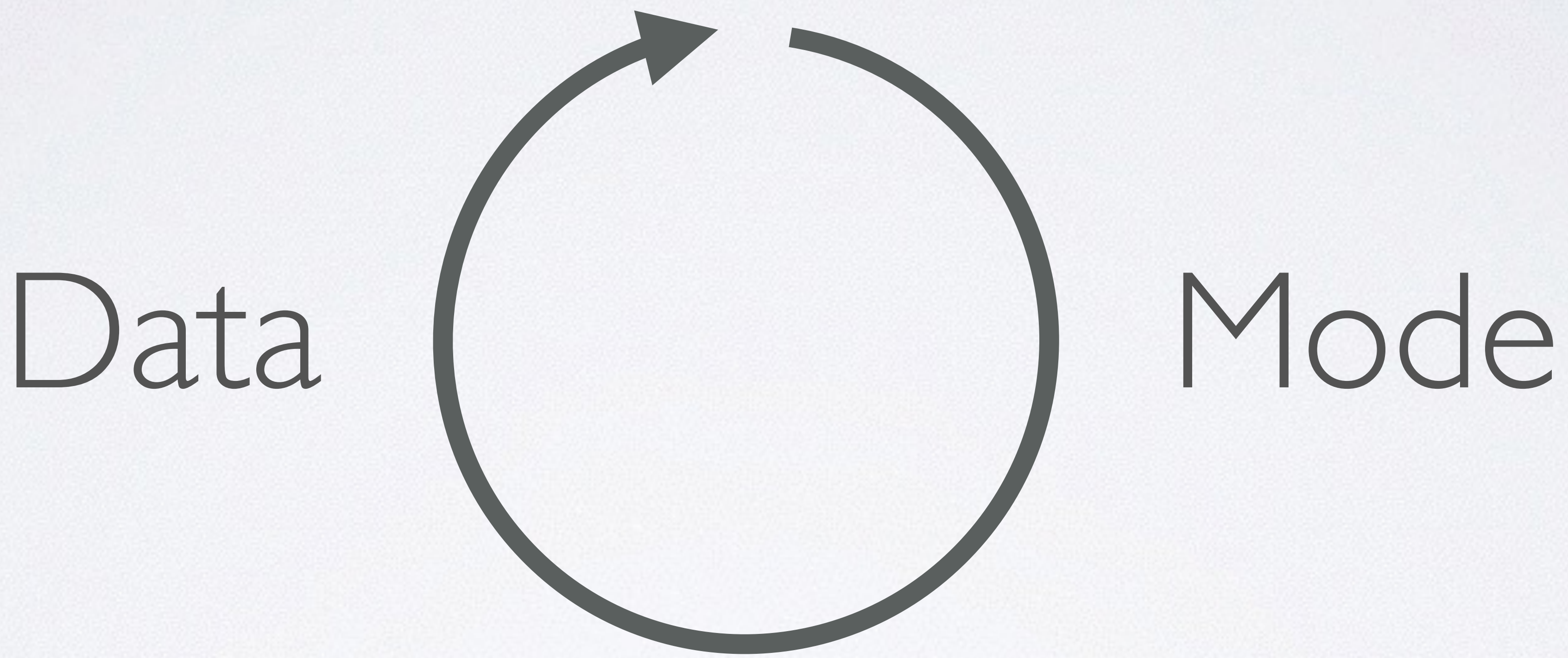
Trigger by the evaluation system

PREDICTION LATENCY
WITH ONLINE TRAINING



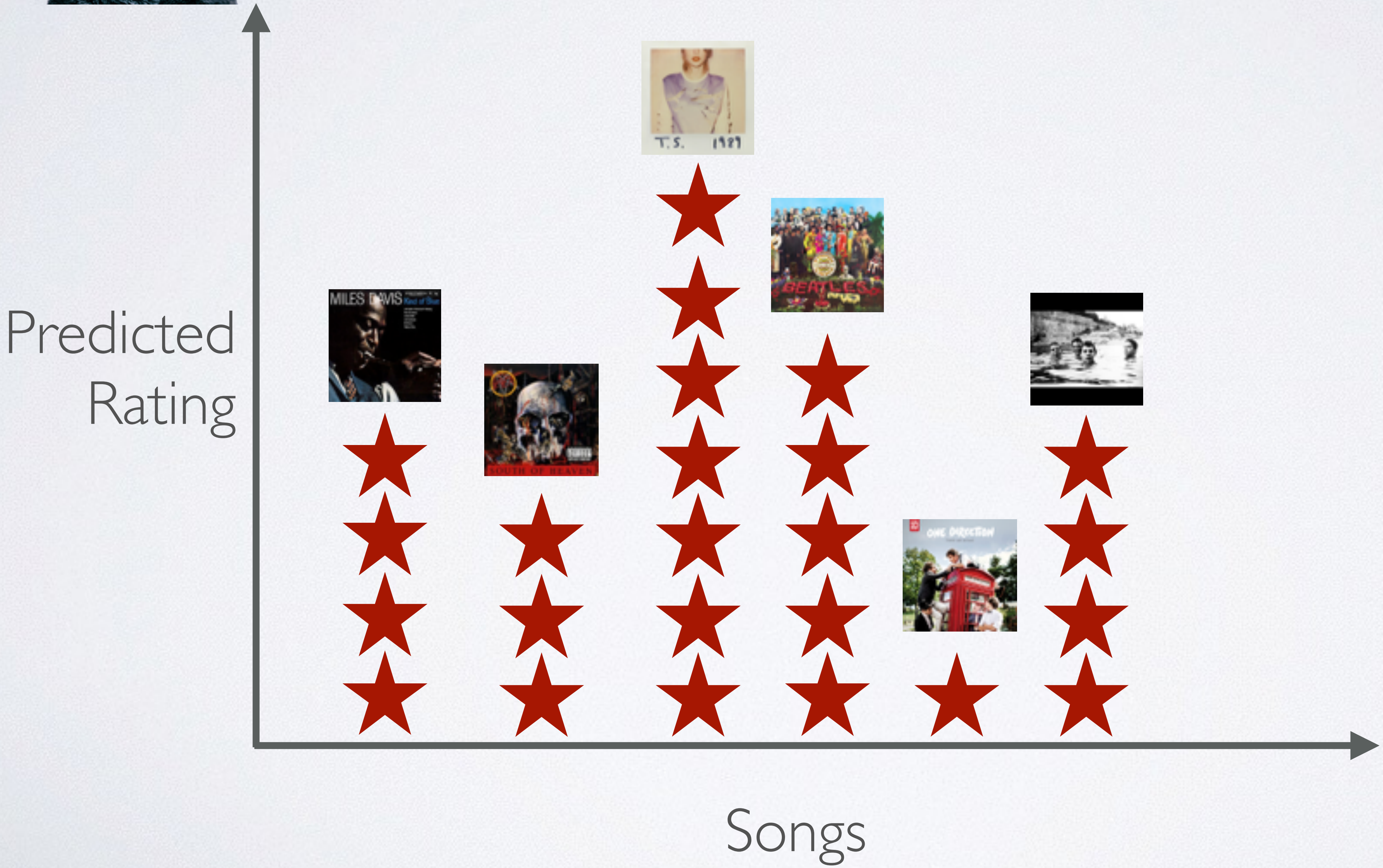
Velox keeps models updated at low latency


Data
Model

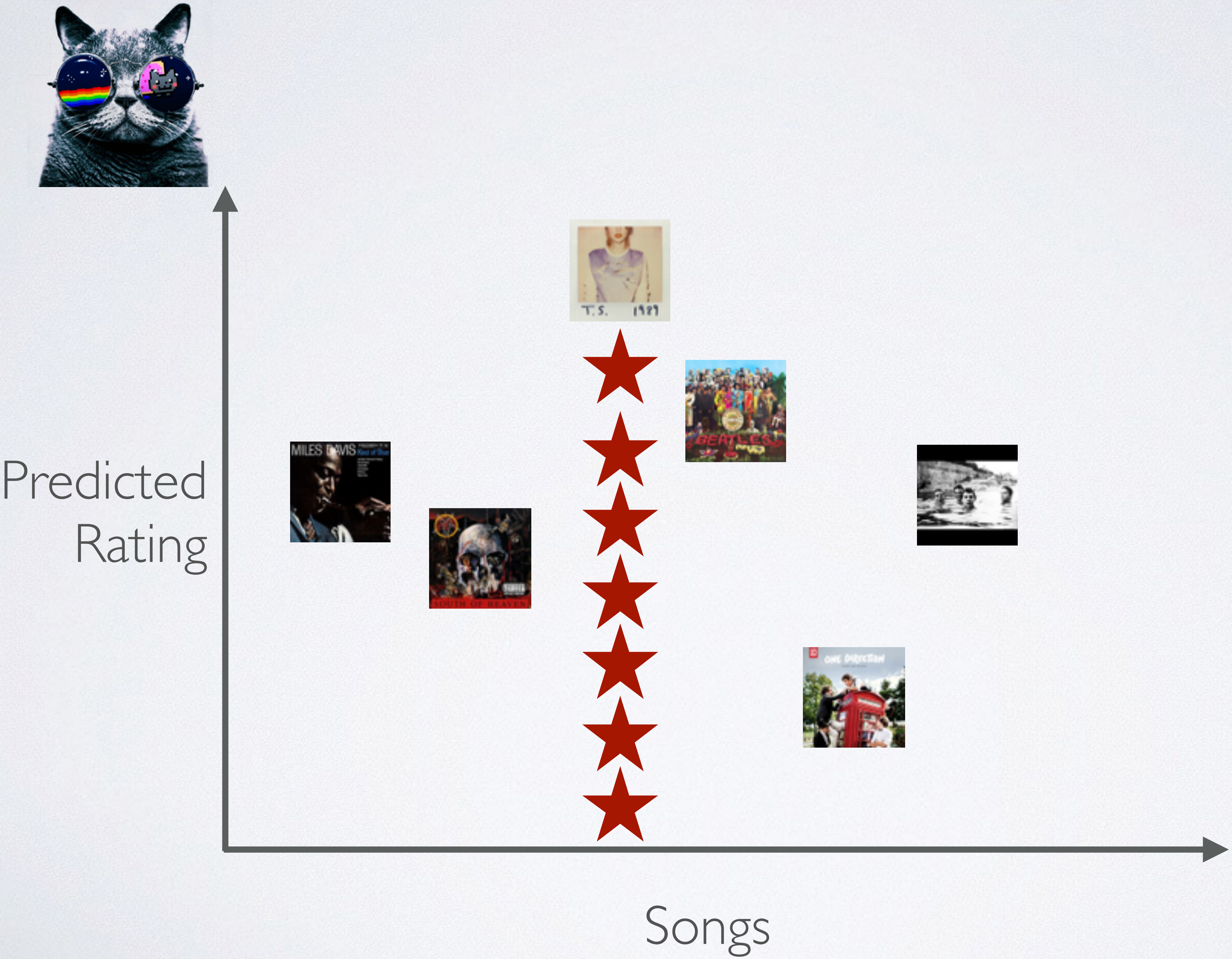


Sample Bias: *model affects the training data*.

ALWAYS SERVE THE BEST SONG?



ALWAYS SERVE THE BEST SONG?

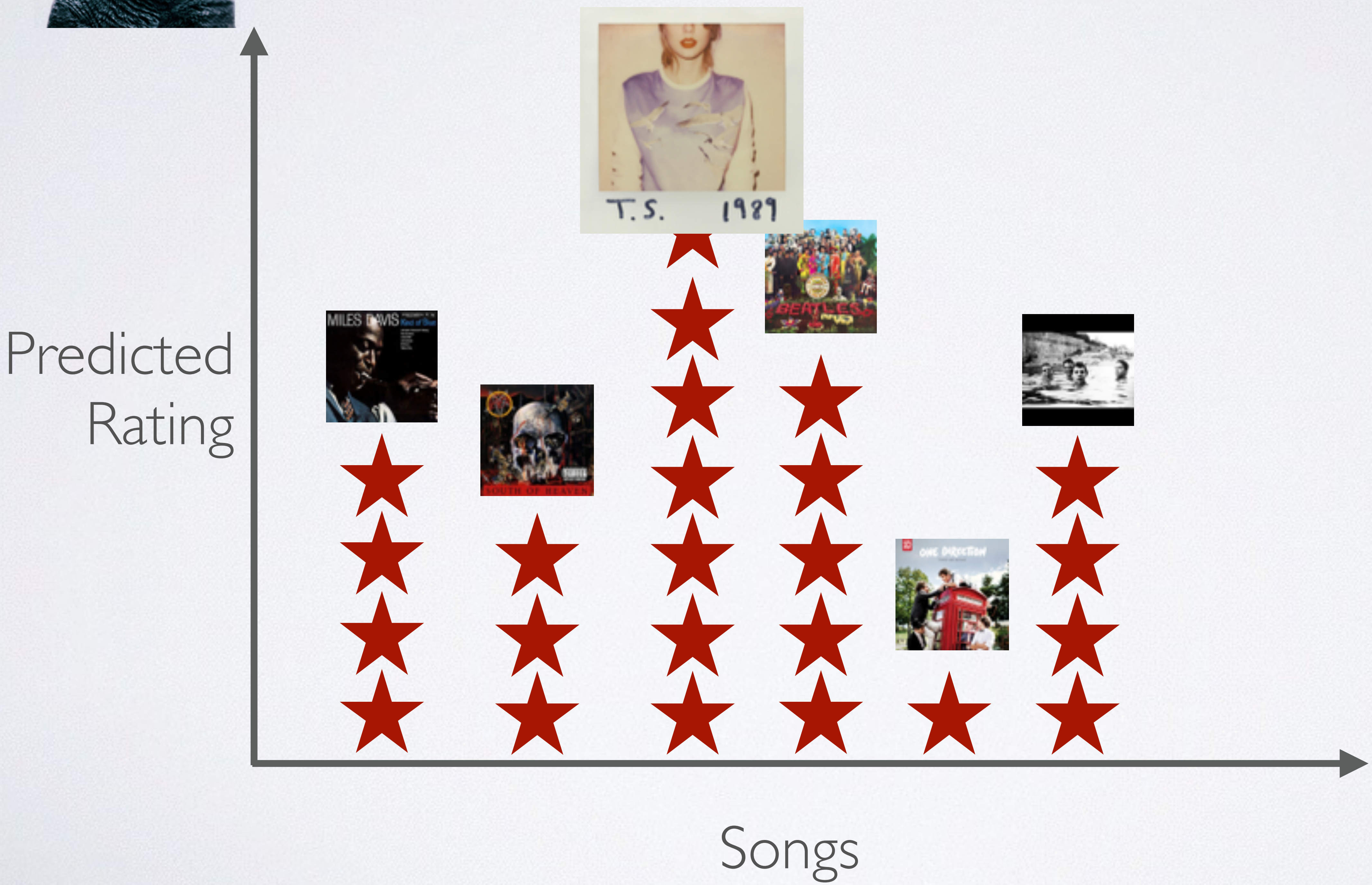


VELOX SOLUTION

With prob. 1- ϵ serve the best predicted song

VELOX SOLUTION

With prob. 1- ϵ serve the best predicted song



VELOX SOLUTION

With prob. 1- ϵ serve the best predicted song

With prob. ϵ pick a ***random*** song

VELOX SOLUTION

With prob. 1- ϵ serve the best predicted song
With prob. ϵ pick a ***random*** song

VELOX SOLUTION

With prob. 1- ϵ serve the best predicted song
With prob. ϵ pick a ***random*** song

VELOX SOLUTION

With prob. 1- ϵ serve the best predicted song

With prob. ϵ pick a ***random*** song

Epsilon Greedy

Active Learning

Opportunity to explore new systems for this emerging analytics workload

Talk Outline

- ML model management today
- Velox system architecture
- Key idea: Split model family
- Prediction serving
- **Model management**
- Next directions

Talk Outline

- ML model management today
- Velox system architecture
- Split model family
- Prediction serving
- Model management
- **Next directions**

OPEN CHALLENGES FOR DATABASE SYSTEMS

OPEN CHALLENGES FOR DATABASE SYSTEMS

- Going beyond the split model family

OPEN CHALLENGES FOR DATABASE SYSTEMS

- Going beyond the split model family
  - **logical model pipeline language**

OPEN CHALLENGES FOR DATABASE SYSTEMS

- Going beyond the split model family
  - **logical model pipeline language**

- More generic training pipelines

OPEN CHALLENGES FOR DATABASE SYSTEMS

- Going beyond the split model family
  - **logical model pipeline language**

- More generic training pipelines
  - **standard set of physical operators**

OPEN CHALLENGES FOR DATABASE SYSTEMS

- Going beyond the split model family
  - **logical model pipeline language**
- More generic training pipelines
  - **standard set of physical operators**
- Automatically choose split for online & offline training

OPEN CHALLENGES FOR DATABASE SYSTEMS

- Going beyond the split model family
  - **logical model pipeline language**
- More generic training pipelines
  - **standard set of physical operators**
- Automatically choose split for online & offline training
  - **view maintenance and query optimization**

OPEN CHALLENGES FOR DATABASE SYSTEMS

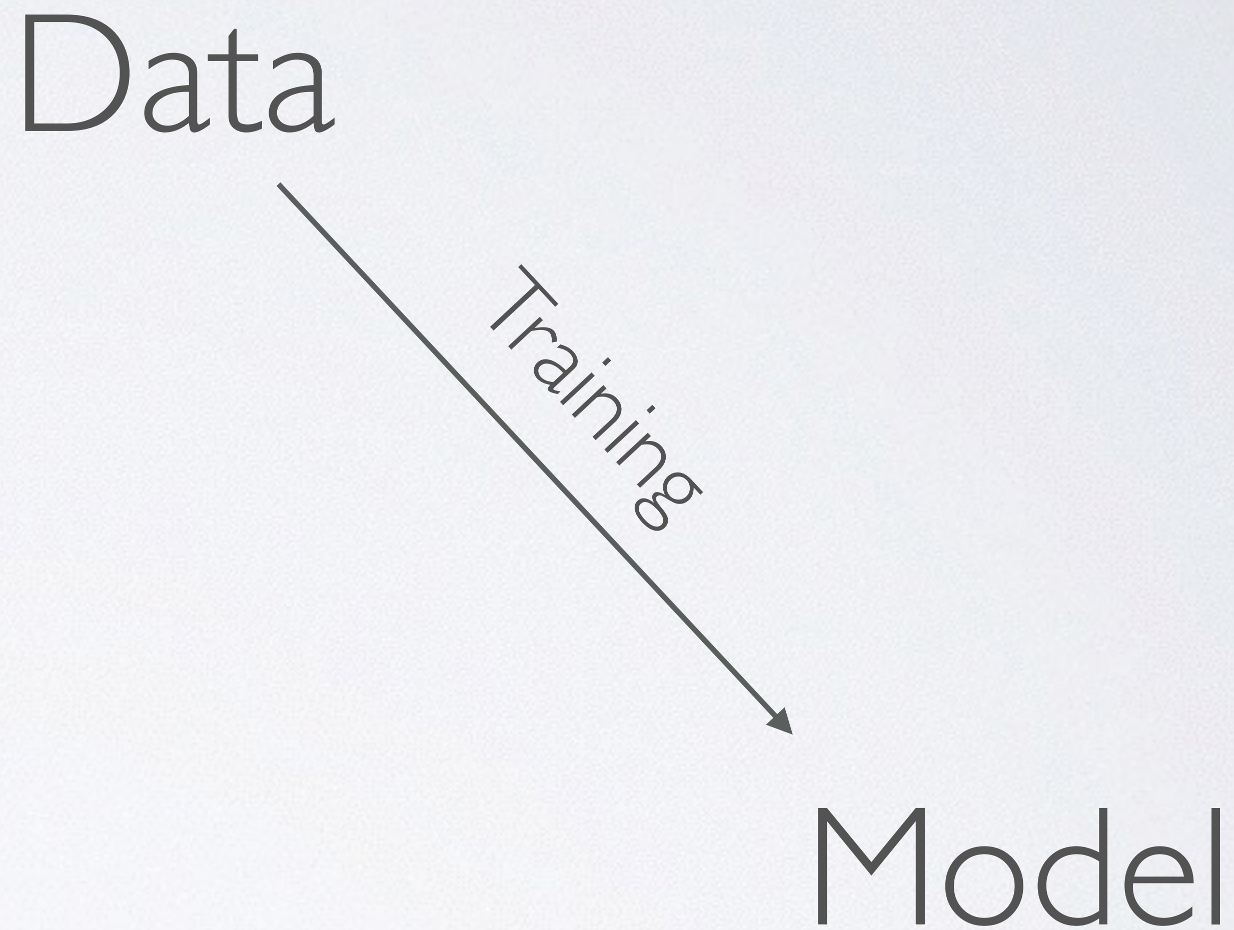
- Going beyond the split model family
  - **logical model pipeline language**
- More generic training pipelines
  - **standard set of physical operators**
- Automatically choose split for online & offline training
  - **view maintenance and query optimization**
- Ensure user privacy

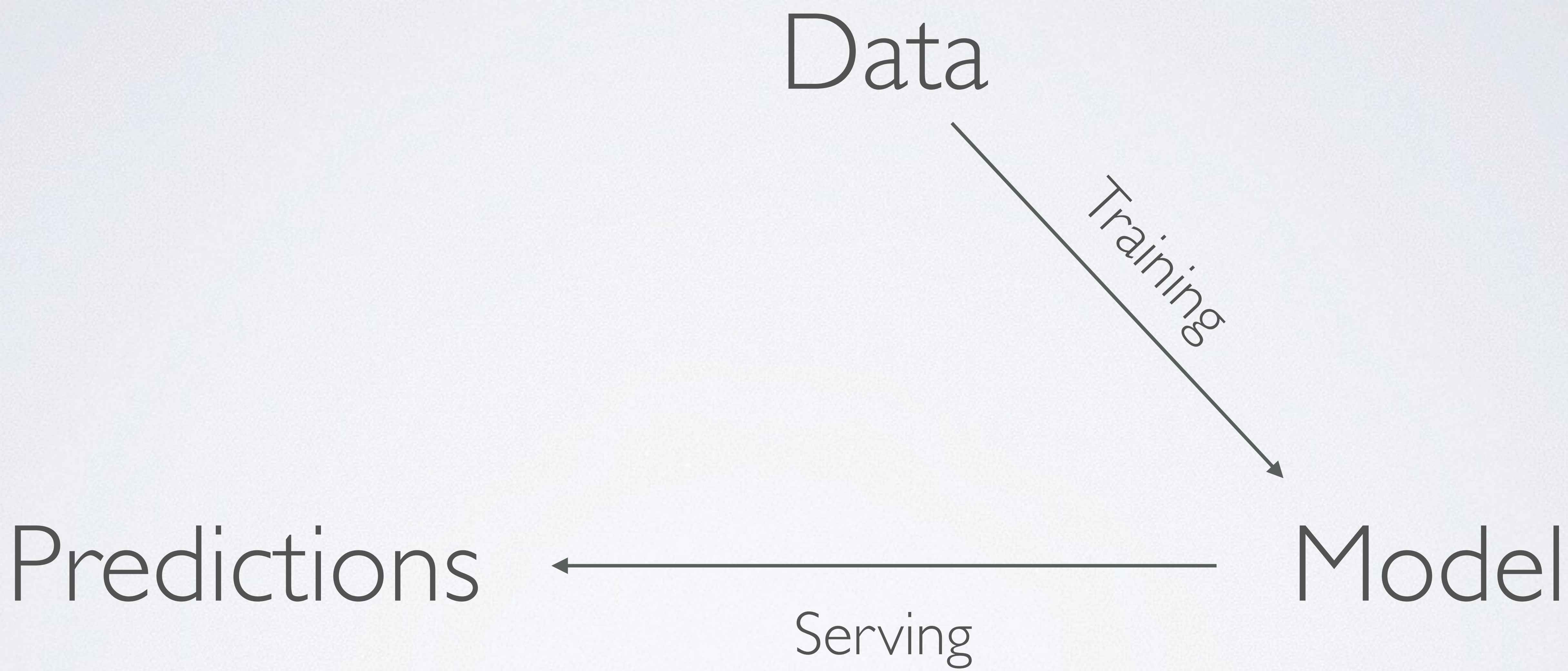
OPEN CHALLENGES FOR DATABASE SYSTEMS

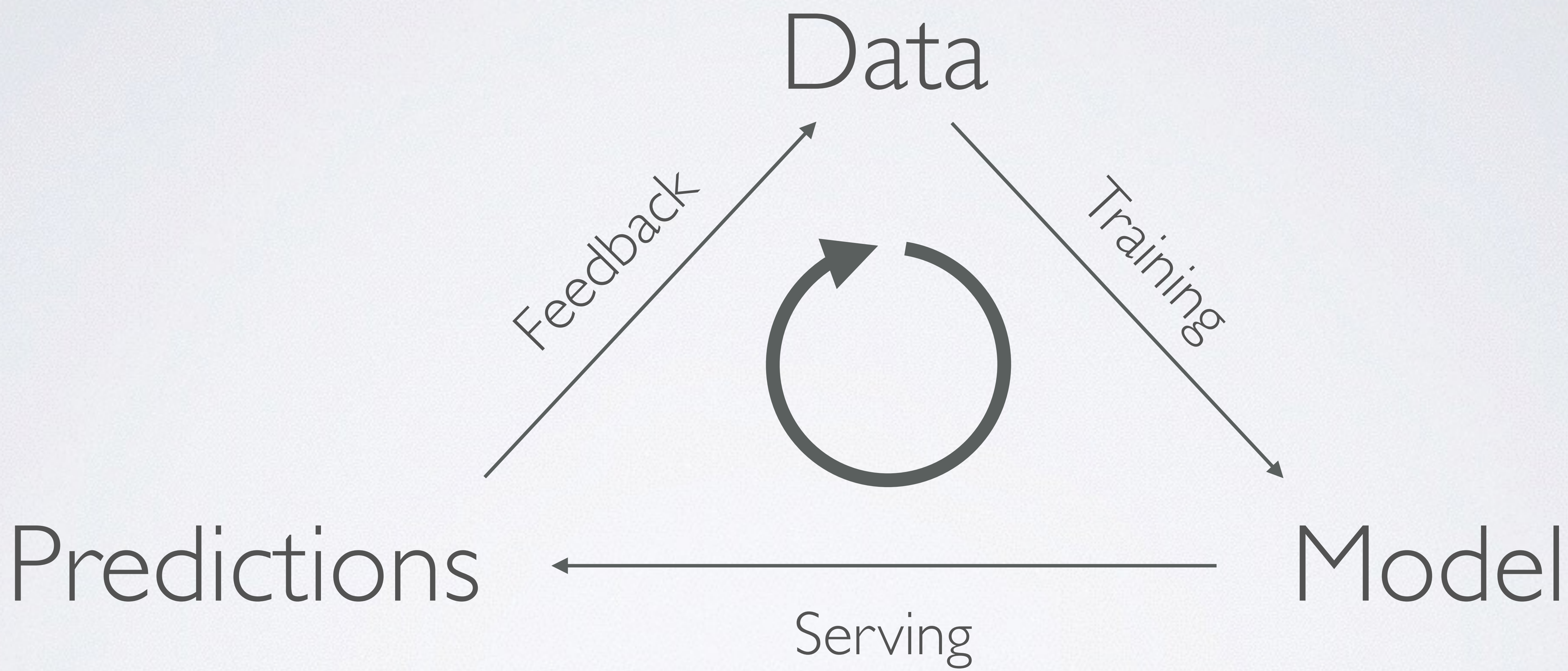
- Going beyond the split model family
  - **logical model pipeline language**
- More generic training pipelines
  - **standard set of physical operators**
- Automatically choose split for online & offline training
  - **view maintenance and query optimization**
- Ensure user privacy
  - **Privacy-Preserving DBMS**

Data

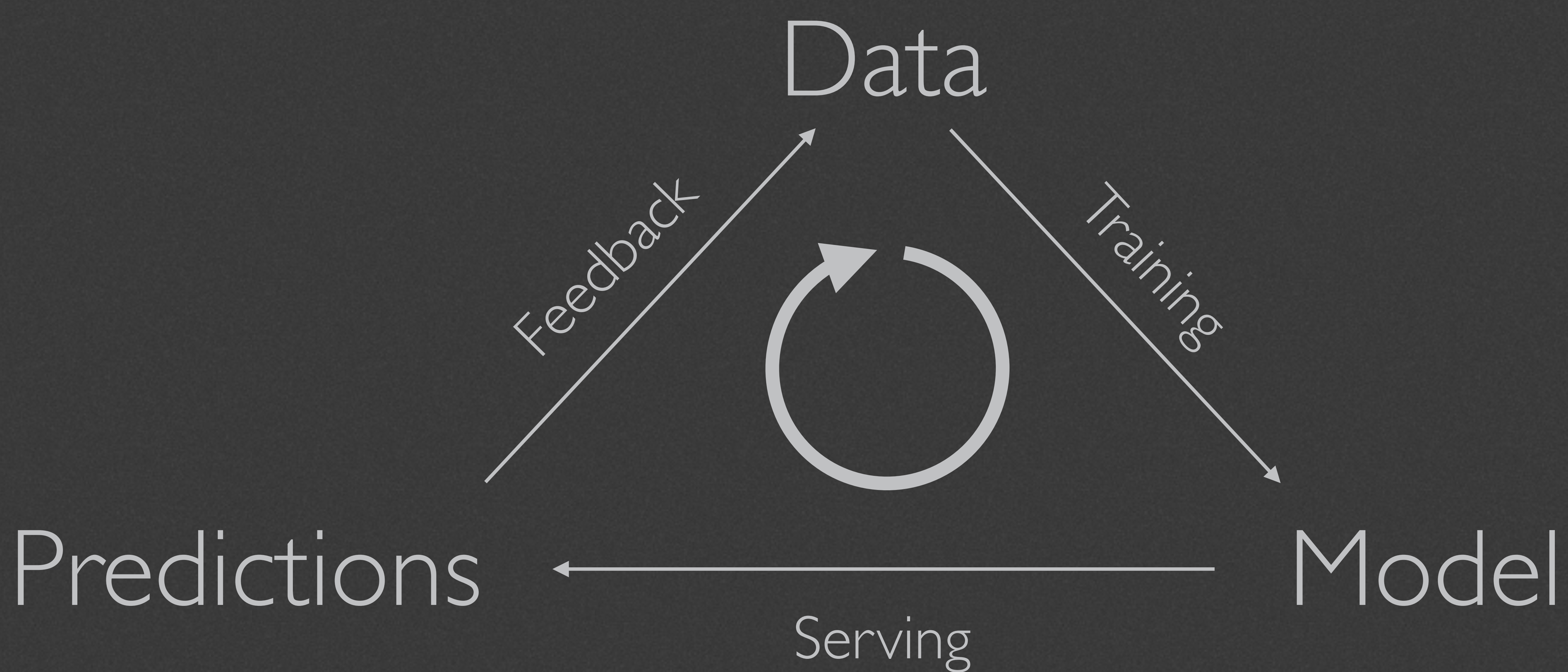
Data → Model


Data
Training
Model

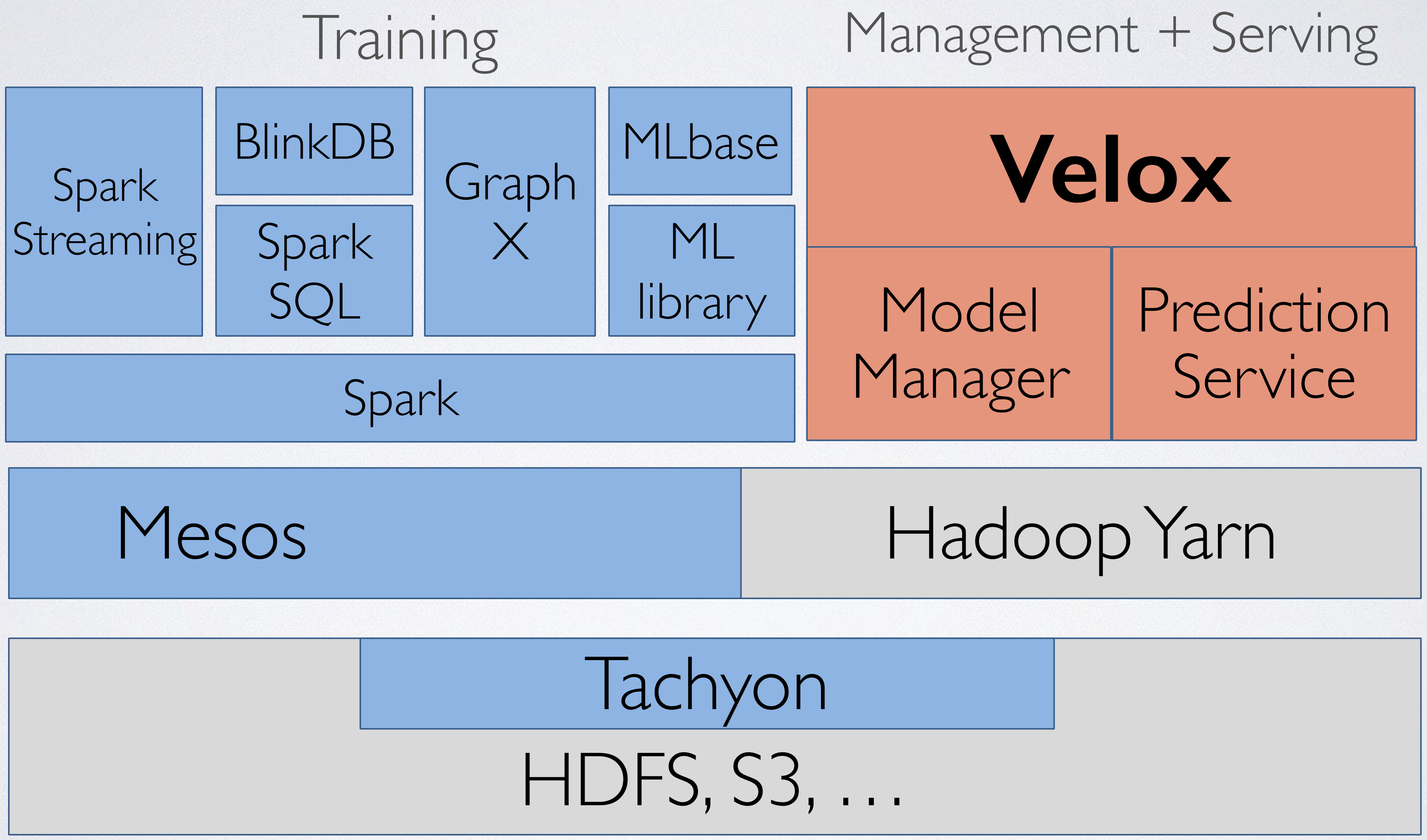

Data
Training
Predictions
Model
Serving

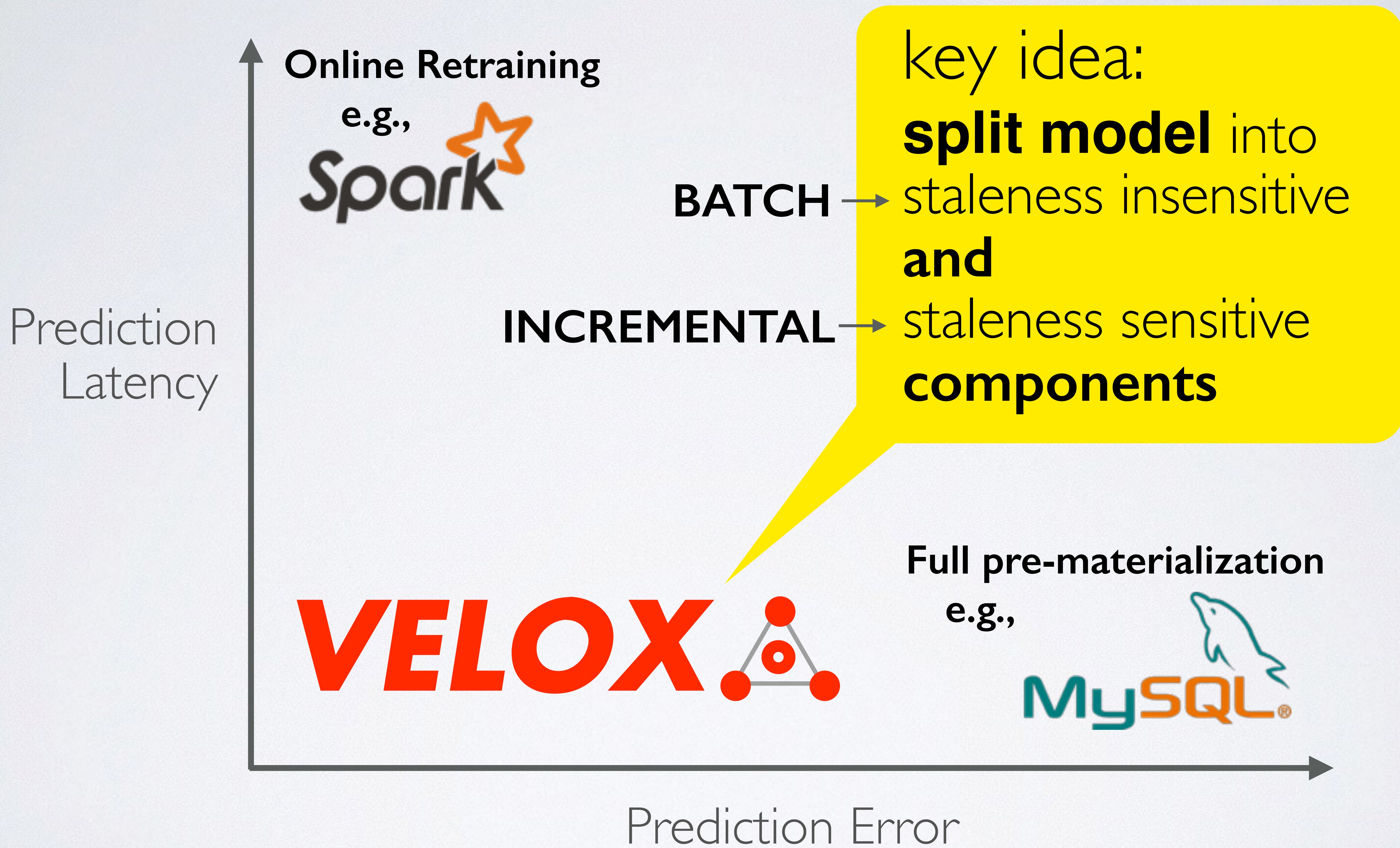

Data
Feedback
Training
Predictions
Model
Serving

The future of research in scalable learning systems will be in the integration of the learning lifecycle:



THE MISSING PIECE




Prediction Latency
Online Retraining
e.g.,
Spark
BATCH
INCREMENTAL
key idea:
split model into
staleness insensitive
and
staleness sensitive
components
VELOX
Full pre-materialization
e.g.,
MySQL
Prediction Error

SUMMARY

SUMMARY

Today: model training and serving relies on **ad-hoc**, **manual** processes spread across **multiple systems**

SUMMARY

Today: model training and serving relies on **ad-hoc**, **manual** processes spread across **multiple systems**

The Velox system **automatically maintains** multiple models while providing **low latency, scalable**, and **personalized predictions**

SUMMARY

Today: model training and serving relies on **ad-hoc**, **manual** processes spread across **multiple systems**

The Velox system **automatically maintains** multiple models while providing **low latency, scalable**, and **personalized predictions**

Velox is coming soon as part of BDAS

SUMMARY

Today: model training and serving relies on **ad-hoc**, **manual** processes spread across **multiple systems**

The Velox system **automatically maintains** multiple models while providing **low latency, scalable**, and **personalized predictions**

Velox is coming soon as part of BDAS

https://amplab.cs.berkeley.edu/projects/velox/

QUESTIONS?