# Self-tuning Transactional Memory via Machine Learning and Analytical Modeling

**Paolo Romano**

INESC-ID

Instituto Superior Técnico, Lisbon University

# Roadmap

- Background on Transactional Memory
  - alternative implementations

- TM performance tuning

- Gray box based self-tuning
  - Provisioning and optimization of a Distributed TM
  - Divide and conquer
  - Bootstrapping
  - Hybrid ensembling

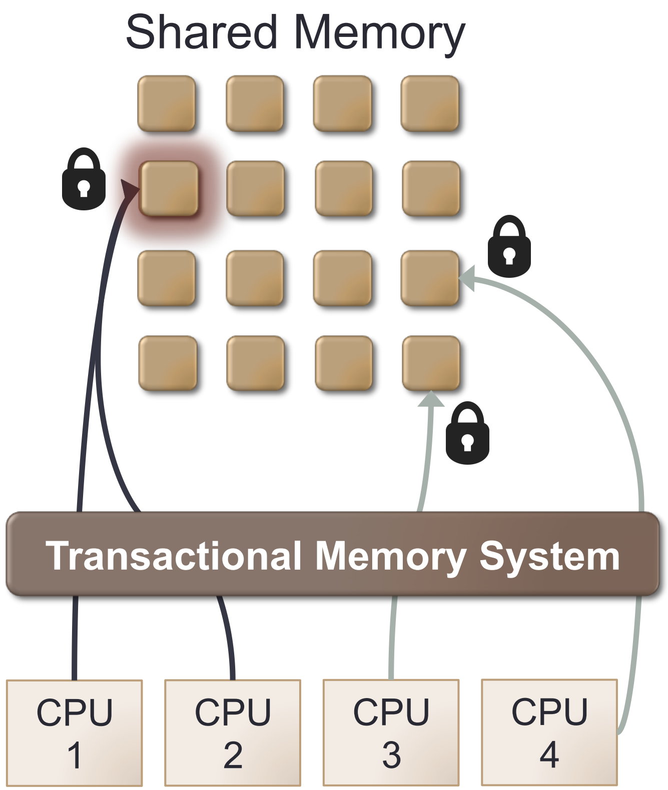
# The multi-core (r)evolution

Multi-cores are now ubiquitous

Concurrent programming is complex

Classic approach: Locking

Transactional Memory abstraction



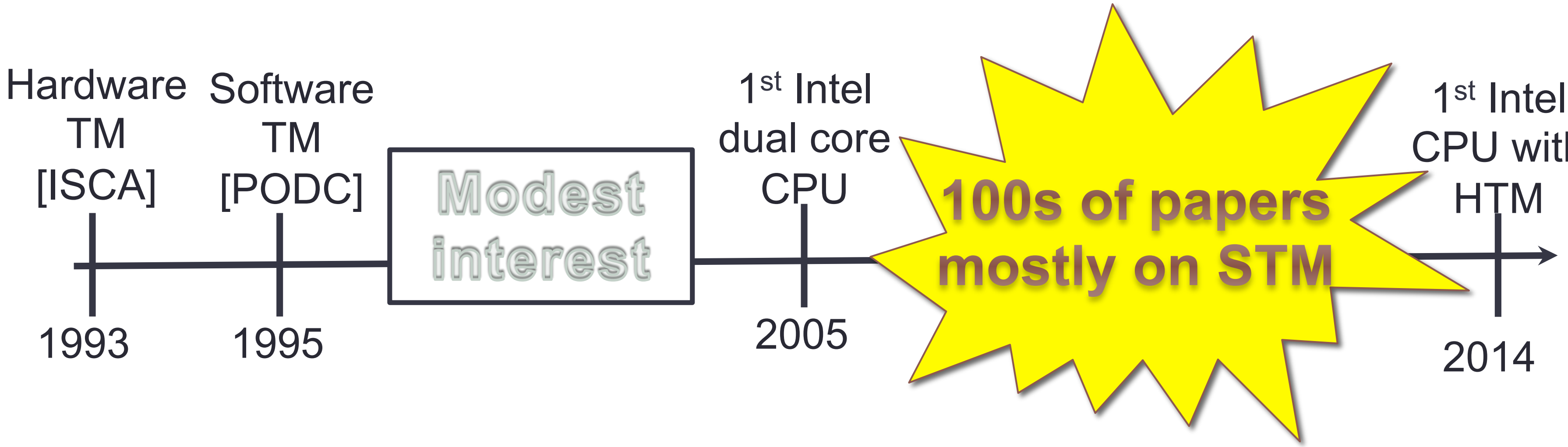

Hard to get right:

- fine-grained locks
- deadlocks
- correctness

```
atomic {
 withdraw(acc1,val);
 deposit(acc2,val);
}
```

Programmer identifies atomic blocks
Runtime implements synchronization

# (A very incomplete)
# Historical perspective on TM




Intel's Haswell CPU targets mainstream computing platforms:

- including desktops, servers, laptops, and tablets

Recently also IBM has integrated HTM supports in its high-end CPUs:

- BG/Q, zEC12, Power8

# Transactional Memory: One abstraction, many implementations

- Software (STM):
  - instrumenting read and write accesses
    - PRO: flexibility
    - CON: instrumentation overheads
- Hardware (HTM):
  - extension of the cache consistency mechanism
    - PRO: no instrumentation overheads
    - CON: hw is inherently limited
- Hybrid (HyTM)
  - mix of the two worlds that tries to achieve the best of both
- Distributed (DTM)
  - natural extension of TM for distributed shared memory
    - PRO: fault-tolerance, potential for higher scalability
    - CON: synchronization costs are amplified

# Software TM

Source program

```
int i = 0

…

atomic {
    i++
}
```

instrumentation

to invoke STM

Compiled program

```
int i = 0
…
TM.begin-tx()
int tmp =
TM.read(&i)
tmp++
TM.write(&i, tmp)
TM.end-tx()
```

- **Non-negligible instrumentation overheads**

- **Highly flexible:**
  - **Avoid inherent restrictions of hardware implementations**
- **Over 10 years of research on STM**

  **➔ highly optimized prototypes and designs**

# Transactional Memory: One abstraction, many implementations

- Software (STM):
  - instrumenting read and write accesses
    - PRO: flexibility
    - CON: instrumentation overheads
- Hardware (HTM):
  - extension of the cache consistency mechanism
    - PRO: no instrumentation overheads
    - CON: hw is inherently limited
- Hybrid (HyTM)
  - mix of the two worlds that tries to achieve the best of both
- Distributed (DTM)
  - natural extension of TM for distributed shared memory
    - PRO: fault-tolerance, potential for higher scalability
    - CON: synchronization costs are amplified

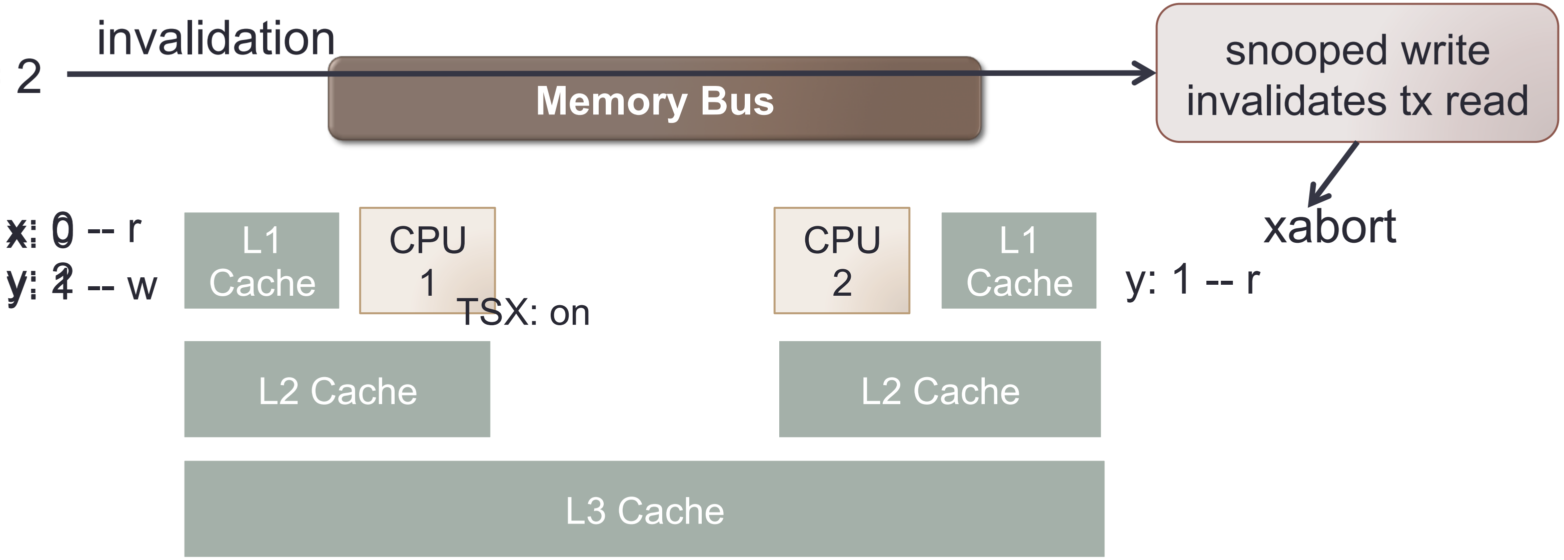
# HTM: Intel Transactional Synchronization Extensions (TSX)

CPU 1

```
xbegin
read x: 0      // Set bit read on x cache line
write y = 1    // Buffer write in L1 cache
xend           // Atomically clean bits and publish
  ⋮
write y = 2
```

CPU 2

```
  ⋮
xbegin
read y: 1
```



# Restrictions of TSX

No progress guarantees:

- A transaction may **always** abort

…due to a number of reasons:

- Forbidden instructions
- Capacity of caches
- Faults and signals
- Contending transactions, aborting each other

TSX alone is not enough

Needs software fall-back:

- global lock → standard HTM
- STM → hybrid TM

# Transactional Memory: One abstraction, many implementations

- Software (STM):
  - instrumenting read and write accesses
    - PRO: flexibility
    - CON: instrumentation overheads
- Hardware (HTM):
  - extension of the cache consistency mechanism
    - PRO: no instrumentation overheads
    - CON: hw is inherently limited
- Hybrid (HyTM)
  - mix of the two worlds that tries to achieve the best of both
- Distributed (DTM)
  - natural extension of TM for distributed shared memory
    - PRO: fault-tolerance, potential for higher scalability
    - CON: synchronization costs are amplified

# Distributed Transactional Memory

- Extends the reach of TM abstraction to distributed applications

- Enhanced scalability, high-availability and fault-tolerance

- Attractive paradigm for the cloud

Cloud·TM
A novel programming paradigm for the Cloud

# At the convergence of two areas

## Distributed Shared Memory

**Transactions allow to:**

1. Deal with remote data races
2. Boost performance by batching remote synchronizations during commit phase

## Distributed Databases

- Natural source of inspiration for DSTMs...
- but DSTMs have unique requirements, e.g.:
  - >70% txs are 100x shorter in DSTM

# Distributed Transactional Memory

# Roadmap

- Background on Transactional Memory
  - alternative implementations
- TM performance tuning
- Gray box based self-tuning
  - Provisioning and optimization of a Distributed TM
  - Divide and conquer
  - Bootstrapping
  - Hybrid ensembling

# TM performance tuning

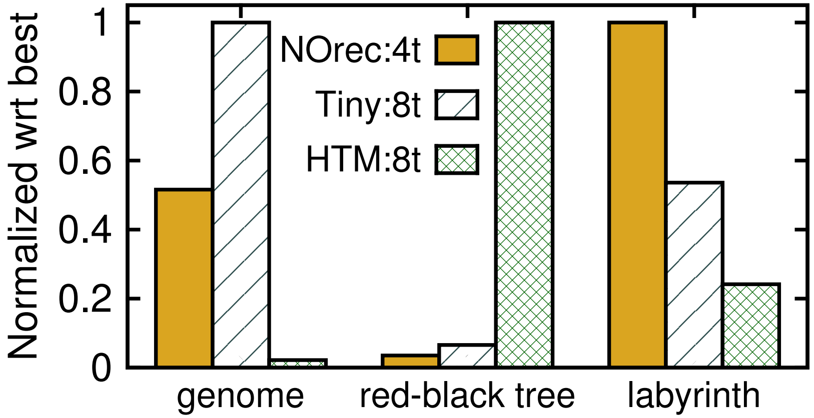
- TM abstraction allows for encapsulating a vast range of alternative implementation strategies
  - no one size fits all solution [SPAA08, PACT14]

- Each implementation comes with various tuning-knobs:
  - number of retries in HTM [ICAC14]
  - granularity of locks in STM [PPoPP08]

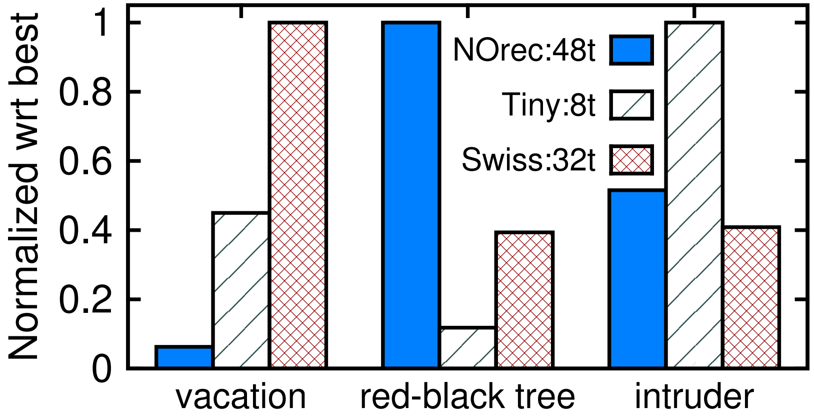
- Parallelism degree:
  - how many threads should be concurrently active? [EuroPar14]

- Thread mapping:
  - on which cores should the active threads be executed? [JPDC14]

# TM tuning: no one size fits all




(a) Throughput/Joule on a single-chip 8-core CPU - Machine A




(b) Throughput on a multi-chip 32-core CPU - Machine B

| Machine ID | Processor / Number of cores / RAM | HTM | RAPL |
|---|---|---|---|
| Machine A | 1 Intel Haswell Xeon E3-1275 3.5GHz / 4 (8 hyper-threads) / 32 GB | Yes | Yes |
| Machine B | 4 AMD Opteron 6172 2.1 Ghz / 48 / 32 GB | No | No |

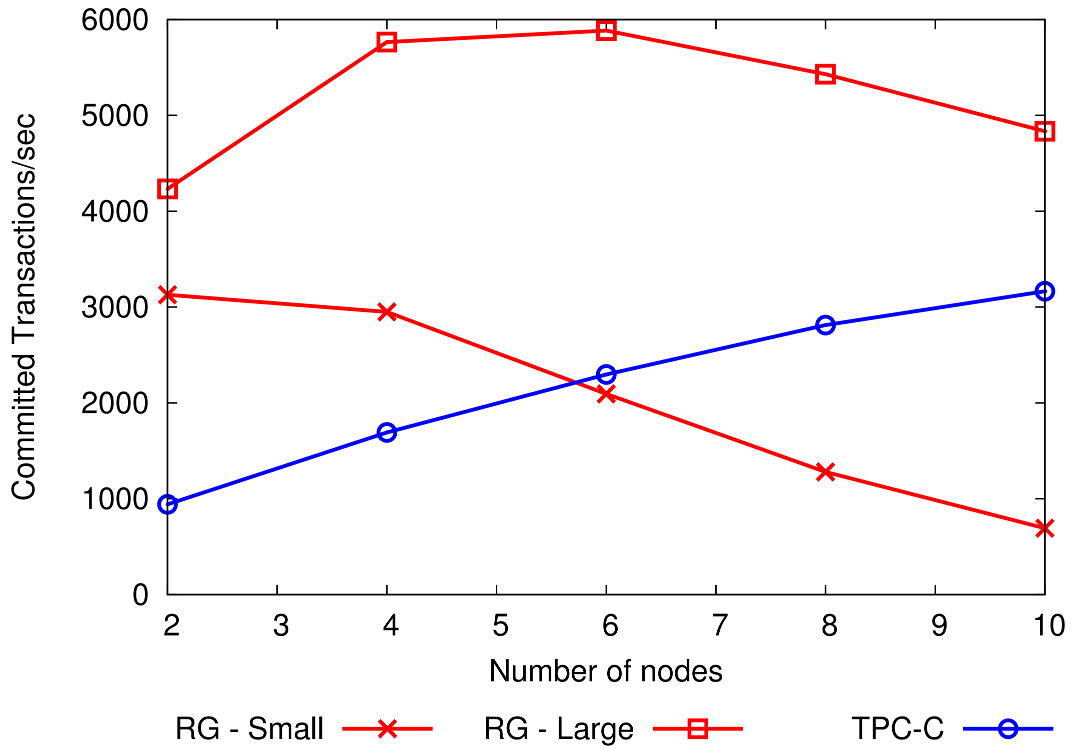
# DTM performance tuning

- Support both scale up and scale out [TAAS14]
  - how many machines should my DTM be provisioned with?
  - how many threads should be active on each machine?

- Communication latencies play a critical factor
  - select the distributed coordination protocol that maximizes efficiency [DSN13]
  - dynamically tune parameters (e.g., batching) of the Group Communication System to enhance efficiency [ICPE15]
  - where should data and code be placed to maximize locality? [ICAC13]

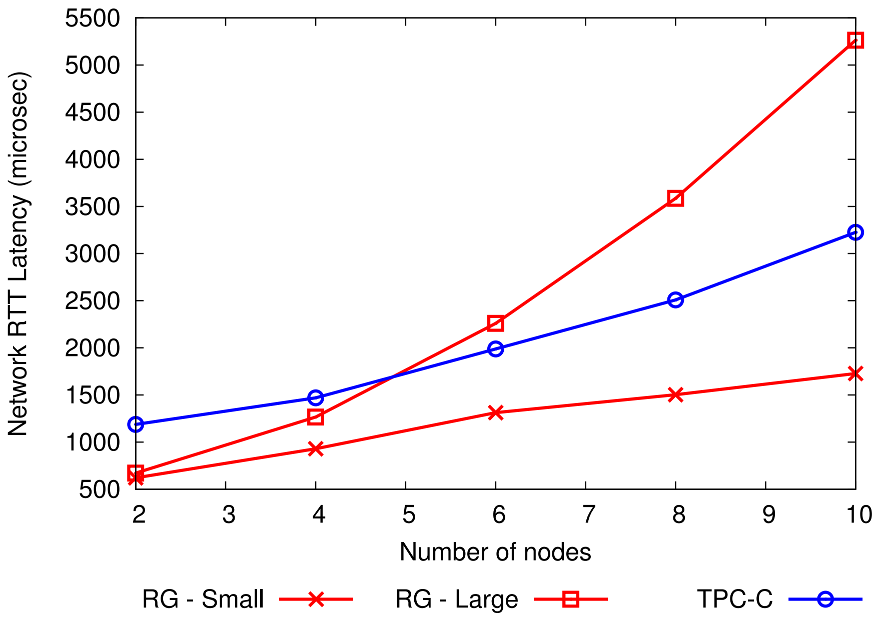
- Cost of exploration can be much higher [Netys13]:
  - launching a new VM is not as simple as spawning a new thread:
    - latency for VM activation, system reconfiguration, state transfer
    - economical cost for VM activation in the cloud

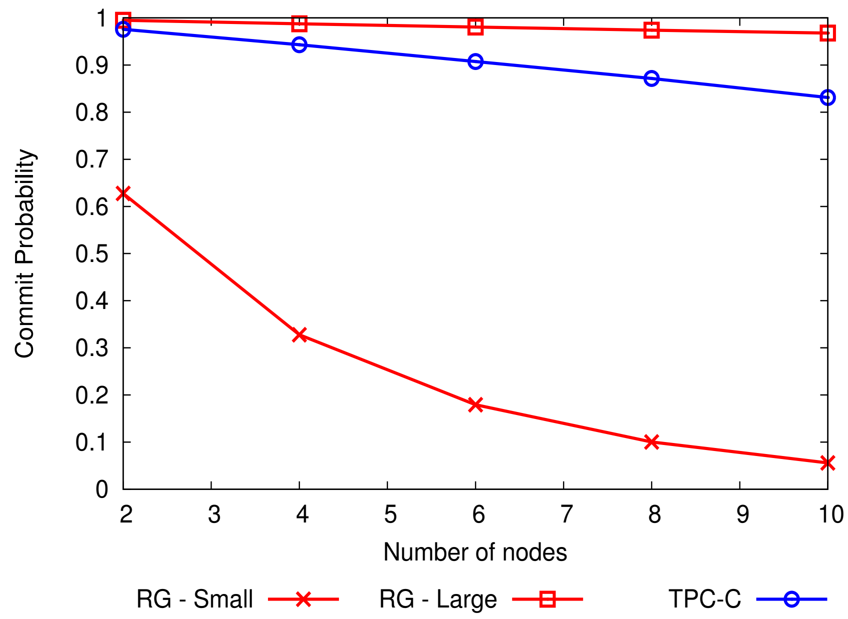
# Performance of Distributed TM




- Heterogeneous, nonlinear scalability trends!

# DTM : Factors limiting scalability



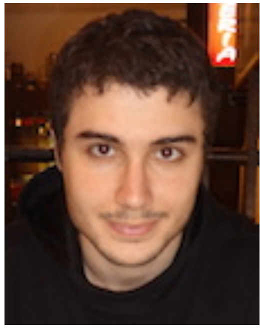

Network latency in commit phase

Aborted transactions because of conflicts

# Roadmap

- Background on Transactional Memory
  - alternative implementations
- TM performance tuning
- Gray box based self-tuning
  - Provisioning and optimization of a Distributed TM
  - Divide and conquer
  - Bootstrapping
  - Hybrid ensembling

# Based on the following papers


1. D. Didona, P. Romano, S. Peluso, F. Quaglia, Transactional Auto Scaler: Elastic Scaling of In-Memory Transactional Data Grids, ACM Transactions on Autonomous and Adaptive Systems (TAAS), 9, 2, 2014, DOI: http://dx.doi.org.10.1145/2620001

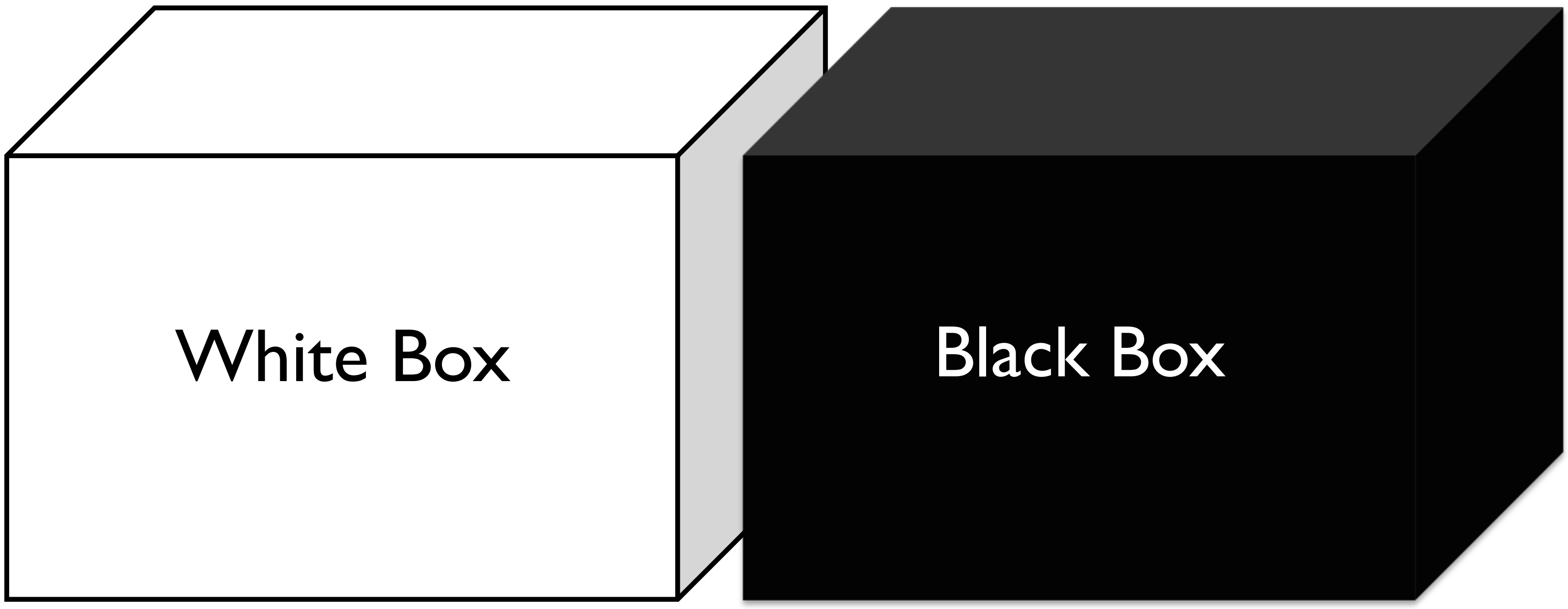
2. D. Didona and Paolo Romano, Performance Modelling of Partially Replicated In-Memory Transactional Stores, IEEE 22nd International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS'14), September 2014

3. D. Didona, P. Romano, F. Quaglia, E. Torre, Combining Analytical Modeling and Machine-Learning to Enhance Robustness of Performance Prediction Models, 6th ACM/SPEC International Conference on Performance Engineering (ICPE), Feb 2015

4. D. Didona, P. Romano, Hybrid Machine Learning/Analytical Models for Performance Prediction: a Tutorial, 6th ACM/SPEC International Conference on Performance Engineering (ICPE), Feb. 2015

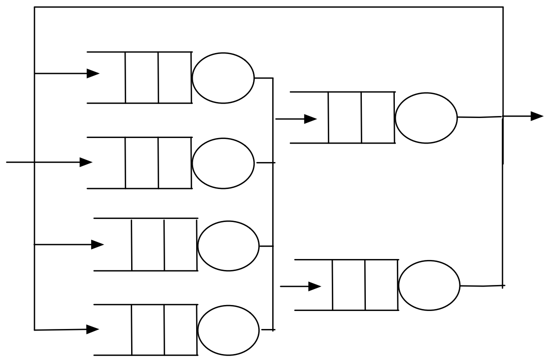
5. D. Didona, P. Romano, Using Analytical Models to Bootstrap Machine Learning Performance Predictors, IEEE International Conference on Parallel and Distributed Systems (ICPADS), December 2015

# Approaches to Performance Modelling



# White box modelling

- Exploit knowledge on internal system dynamics
  - ✧ model dynamics analytically or via simulation



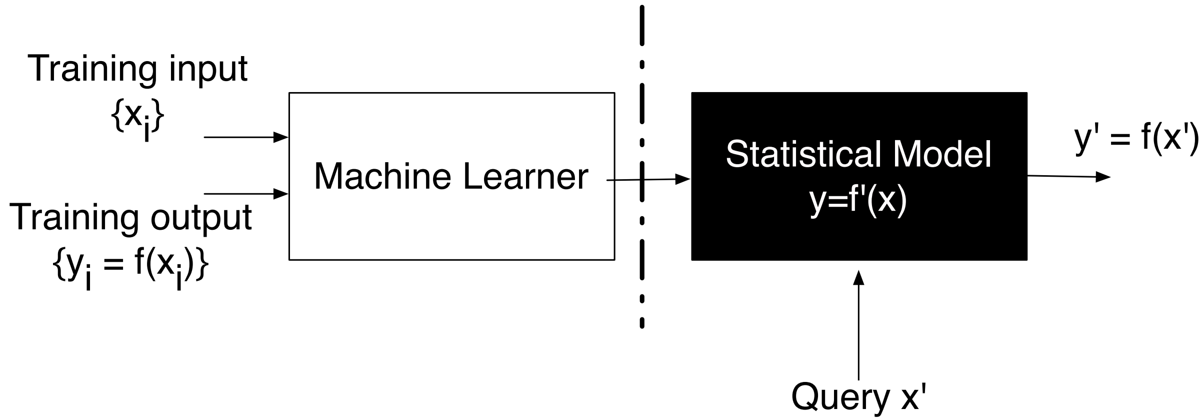
**PROS**

- Good accuracy **on average**
- Minimal or no learning phase

**CONS**

- Simplifying assumptions
  ➔ low accuracy when these assumptions do not hold
- Knowledge of system internals often unavailable

# Black box modelling




**PROS**

- High accuracy in areas already observed (interpolation)
- Do not require knowledge on system's internals

**CONS**

- Poor accuracy in non-observed areas (extrapolation)
- Curse of dimensionality
  - Extensive training phases

# Key Observation & Questions

Pros of white-box are cons of black-box & vicev.

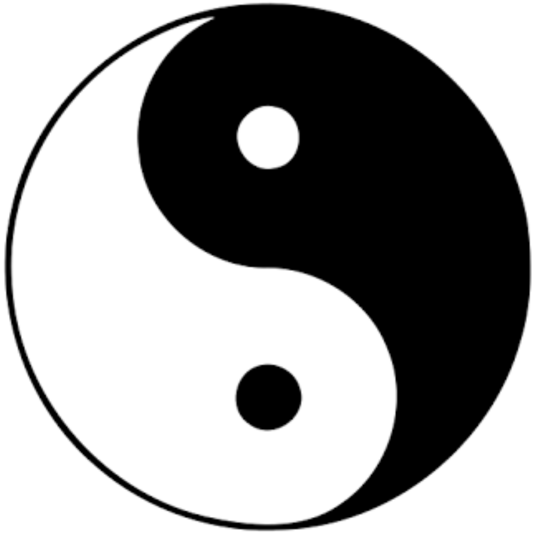
↓

Can we achieve the best of the two worlds?

↓

How can black and white box modelling be reconciled ?

# Gray box modeling



- Combine WB and BB modeling
- Enhance **robustness**
  - Lower training time thx to WBM
  - Incremental learning thx to BBM
- Will present three methodologies:

Divide and conquer

Hybrid ensembling

Bootstrapping

# Gray box modeling

- Will present three methodologies:

Divide and conquer

Bootstrapping

Hybrid ensembling

# Divide and conquer

- Modular approach
  - WBM of what is observable/easy to model
  - BBM of what is un-observable or too complex
- Reconcile their output in a single function

- Higher accuracy in extrapolation thx to WBM
- Apply BBM only to sub-problem
  - Less features, lower training time

# Case study: Infinispan

- Distributed in-memory key-value store:
  - Nodes maintain elements of a dataset
    - Full vs partial replication (# copies per item)
  - Transactional --ACI(D)– manipulation of data
    - Concurrency control scheme (enforce isolation)
    - Replication protocol (disseminate modifications)

# DTM optimization in the Cloud

- Important to model network-bound ops but…
- Cloud hides detail about network ☹
  - No topology info
  - No service demand info
  - Additional overhead of virtualization layer

- BBM of network-bound ops performance
  - Train ML on the target platform

# TAS/PROMPT [TAAS14,Mascots14]

- Analytical modeling (queuing theory based)
  - Concurrency control scheme
    - E.g., encounter time vs commit time locking
  - Replication protocol
    - E.g., PB vs 2PC
  - Replication scheme
    - Partial vs full
  - CPU
- Machine Learning
  - Network bound op (prepare, remote gets)
  - Decision tree regressor

# Analytical model in TAS/PROMPT

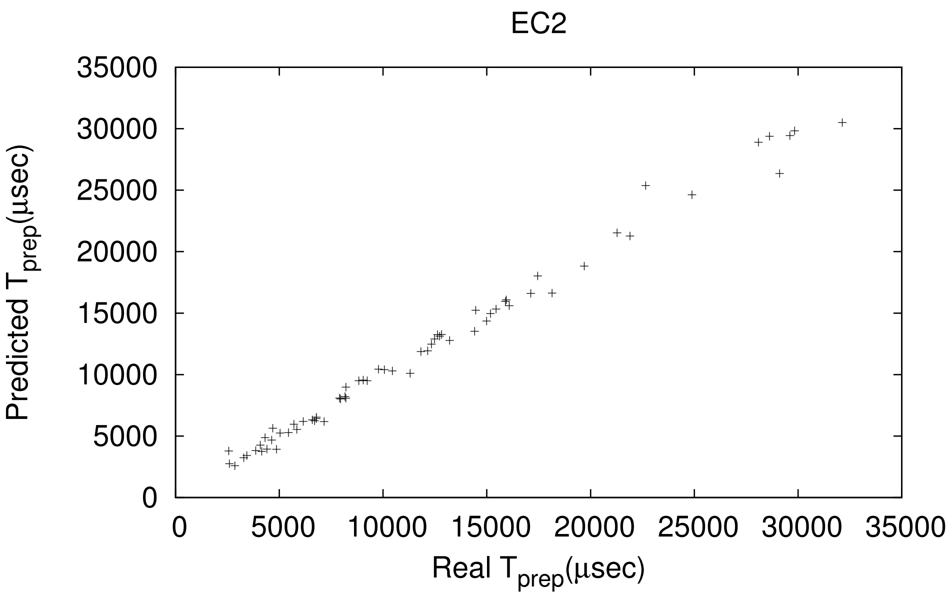
- Concurrency control scheme (lock-based)
  - A lock is a M/G/1 server
  - Conflict prob = utilization of the server
- Replication protocol
  - multi-master/Two-phase Commit based → one model
  - single-master/primary-backup → two models
- Replication scheme
  - Probability of accessing remote data
  - # nodes involved in commit

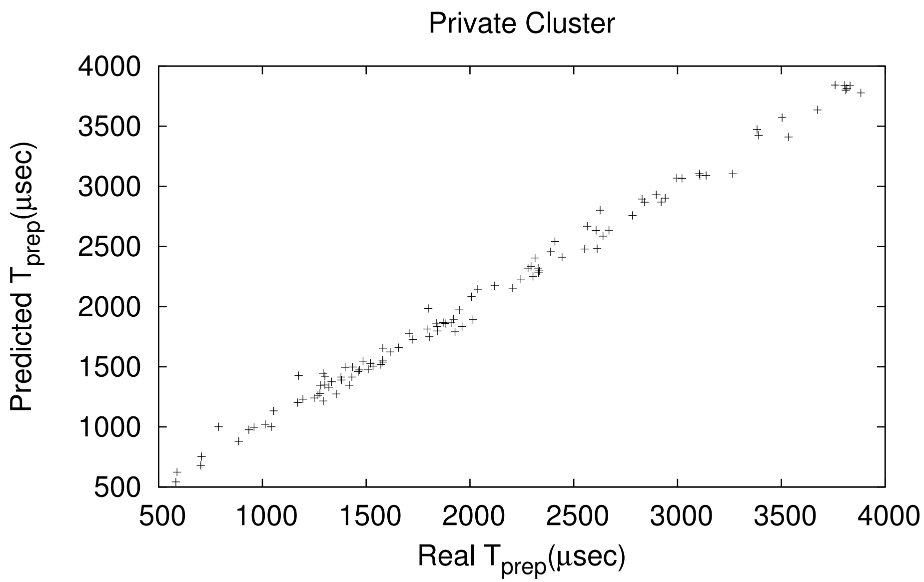
# Machine Learning in TAS/PROMPT

- Decision tree regressor
- Operation-specific models
  - Latency during prepare
  - Latency to retrieve remote data
- Input
  - Operations rate (prepare, commit, remote get…)
  - Size of messages
  - # nodes involved in commit

# ML accuracy for network bound ops

- Seamlessly portable across infrastructures
  - Here, private cloud and Amazon EC2



# AM and ML coupling

At training time, all features are monitorable

At query time they are NOT!

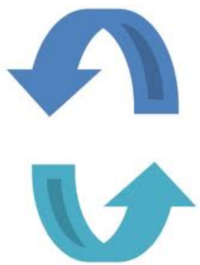
EXAMPLE

- Current config: 5 nodes, full replication
  - Contact all 5 nodes at commit
- Query config: 10 nodes, partial replication
  - How many contacted nodes at commit??

# Model resolution

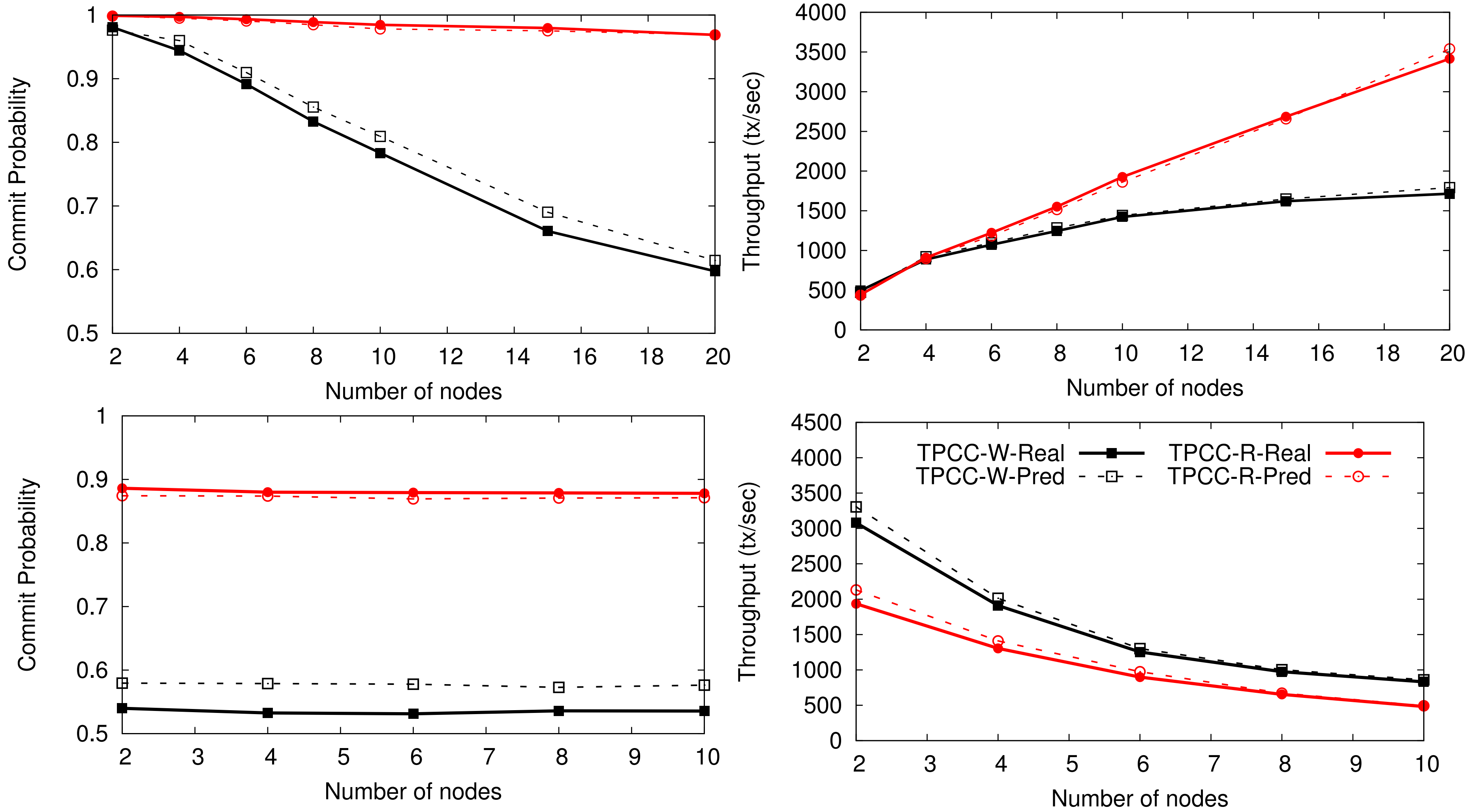
- AM can provide (estimates of) missing input
- Iterative coupling scheme

ML takes some input parameters from AM



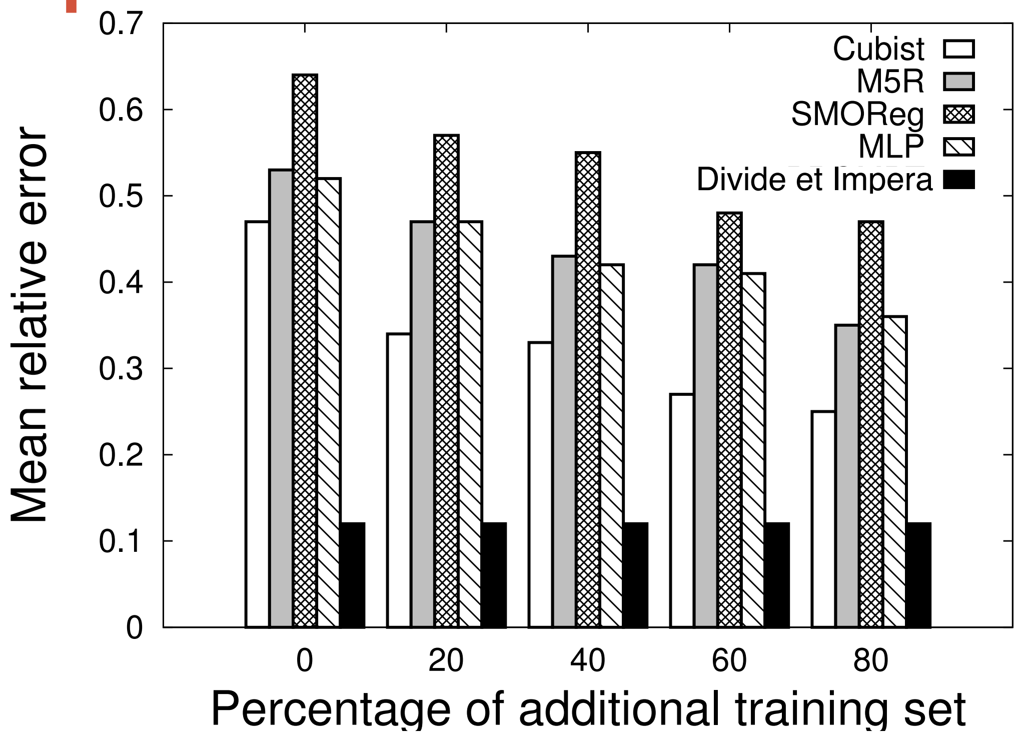
AM takes latencies forecast by ML as input parameter

# Model's accuracy



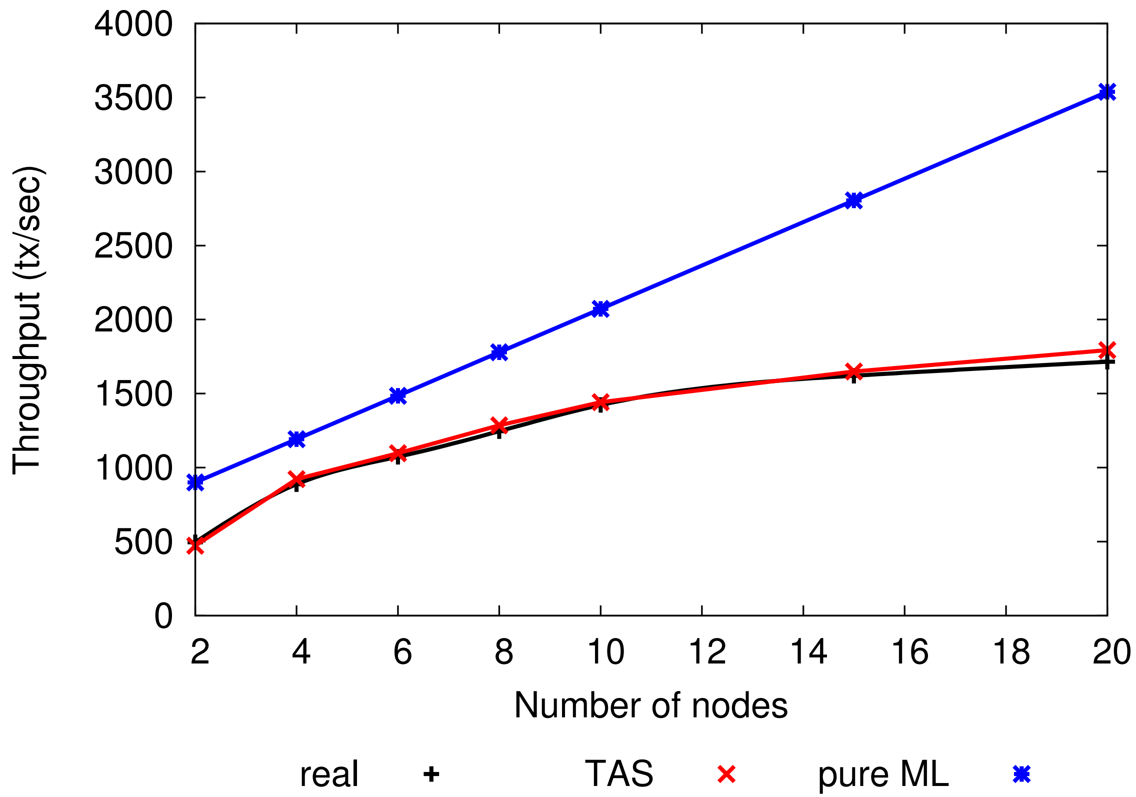

TOP: primary-backup.  BOTTOM: multi-master (2PC-based)

# Comparison with Pure ML, I




- YCSB (transactified) workloads while varying
  - # operations/tx
  - Transactional mix
  - Scale
  - Replication degree

# Comparison with Pure ML, II




- ML trained with TPCC-R and queried for TPCC-W
- Pure ML blunders when faced with new workloads

# Gray box modeling

- Will present three methodologies:

Divide and conquer

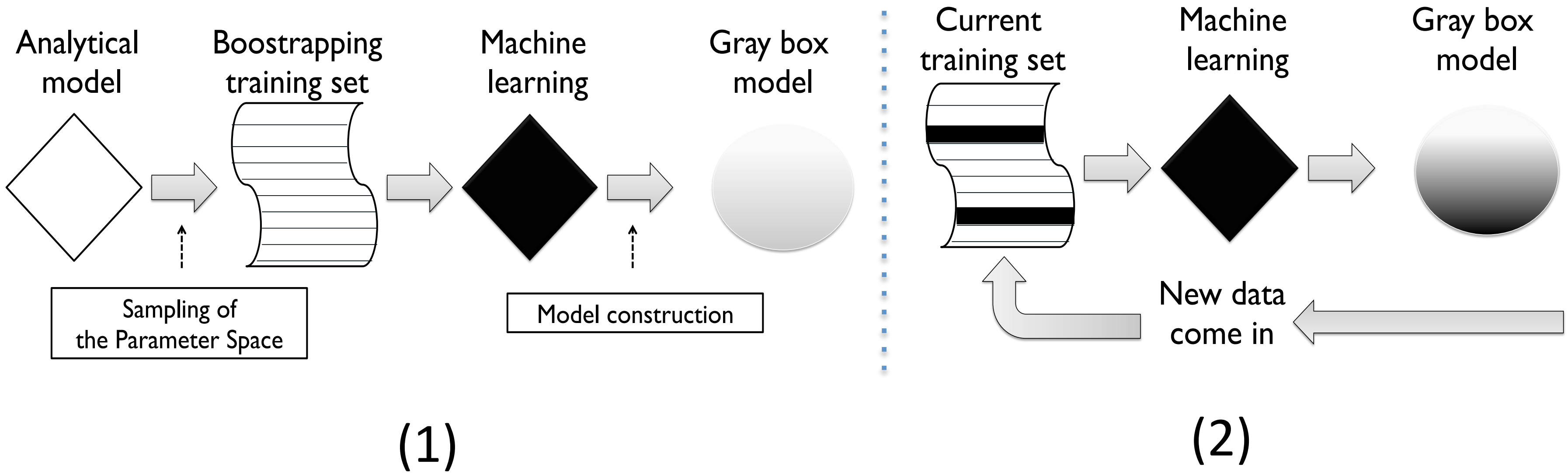
Hybrid ensembling

$\boxed{\text{Bootstrapping}}$

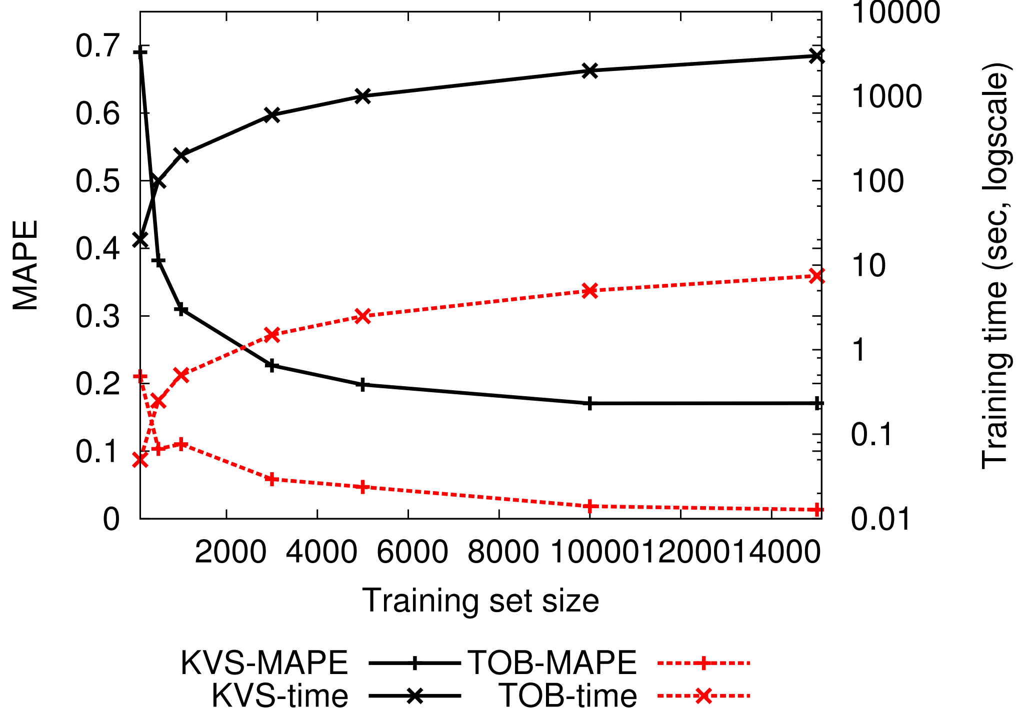
# Bootstrapping

- Obtain zero-training-time ML via initial AM

1. Initial (synthetic) training set of ML from AM
2. Retrain periodically with "real" samples



# How many synthetic samples?

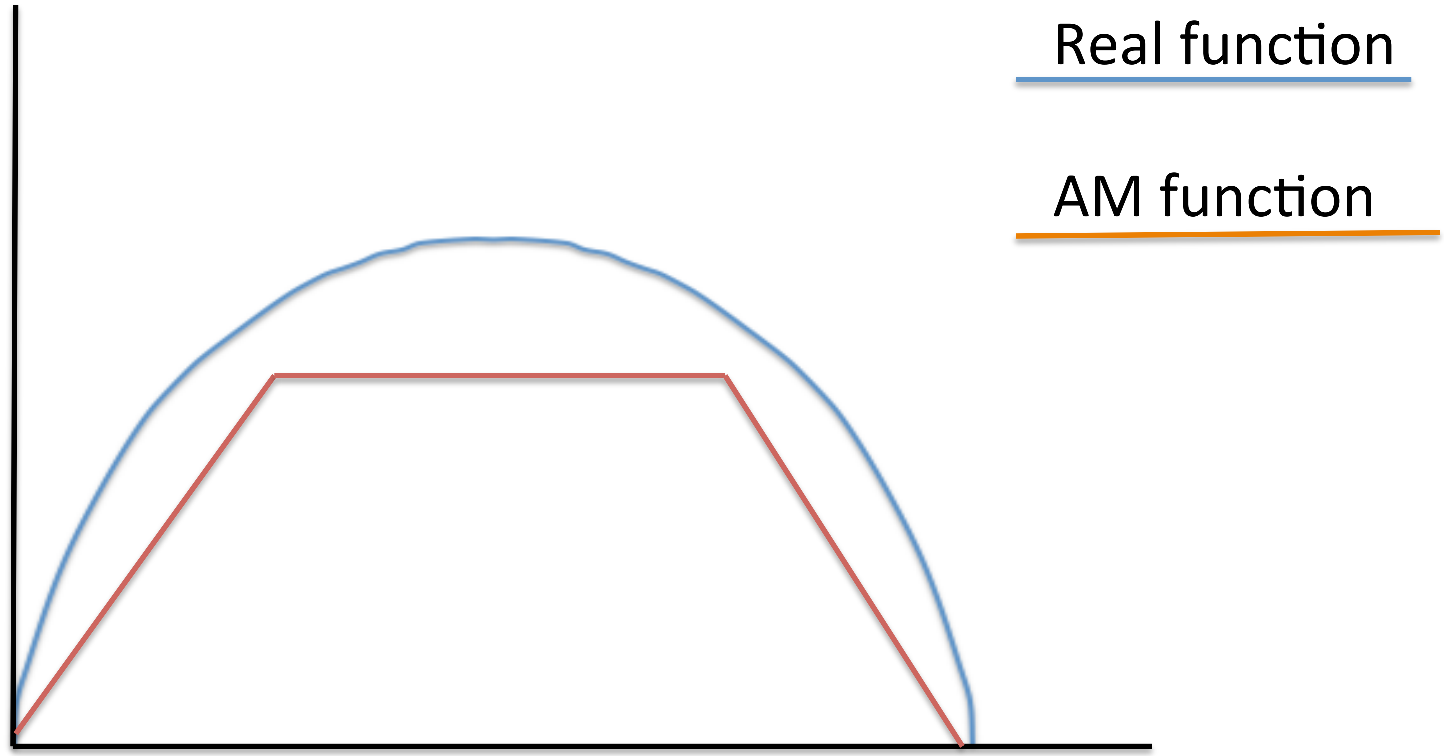



- Important tradeoff
  - Higher # → lower fitting error over the AM output
  - Lower # → higher density of real samples in dataset

# How to update the synthetic training set?

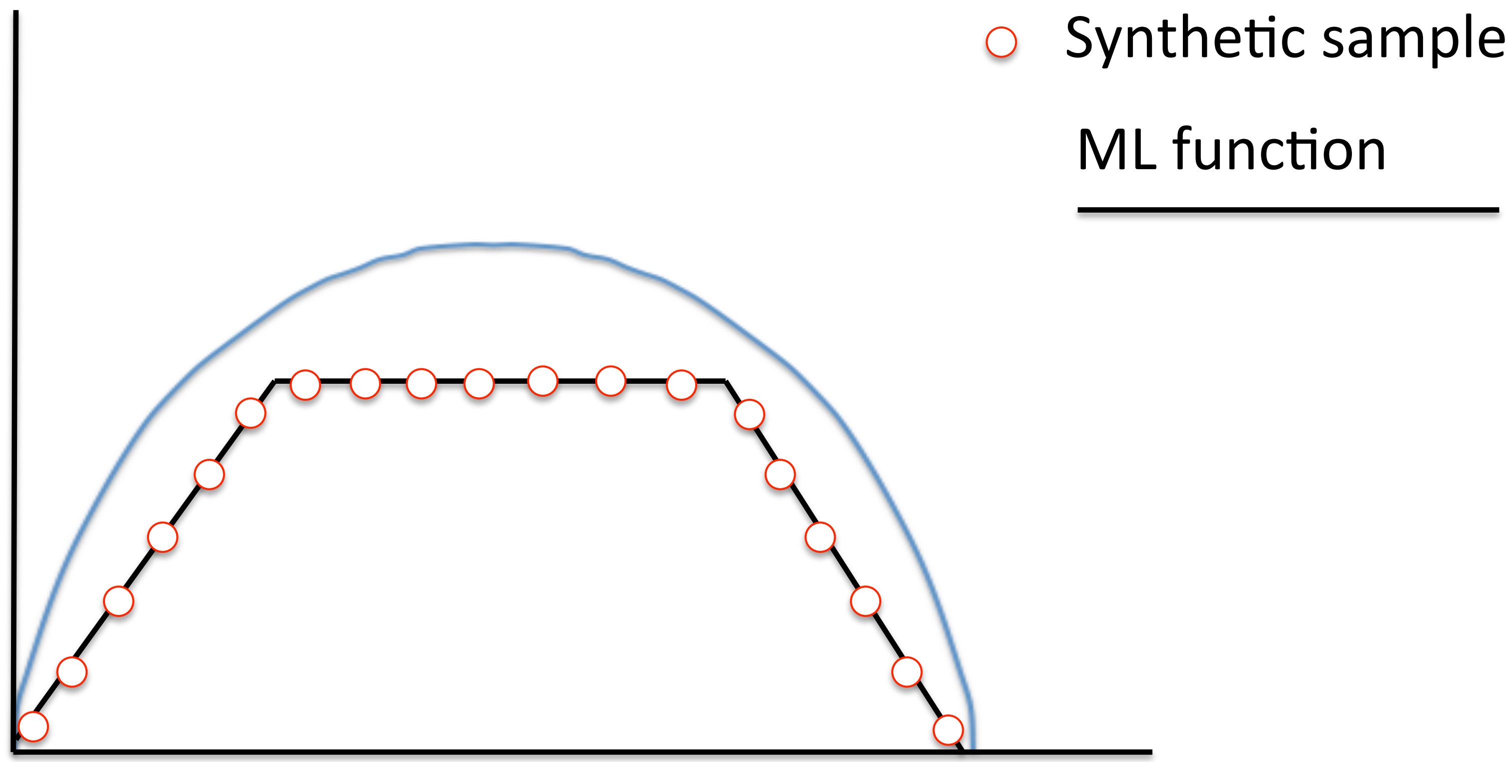
- Merge: simply add real samples to synthetic set
- Replace only the nearest neighbor (RNN)
- Replace neighbors in a given region (RNR)
  - Two variants

# Real vs AM function



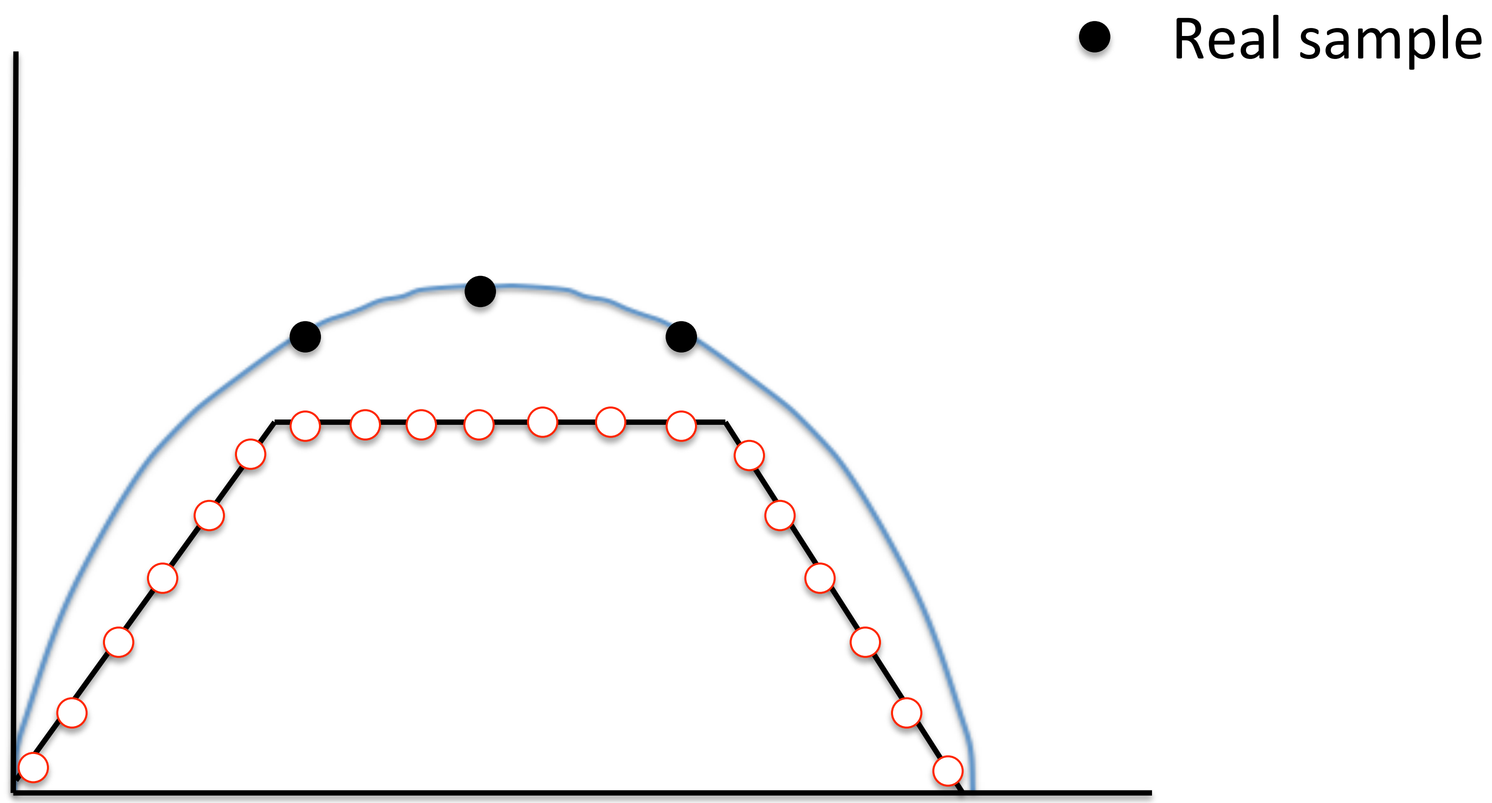
# Real vs learnt

- Assuming enough point to perfectly learn AM



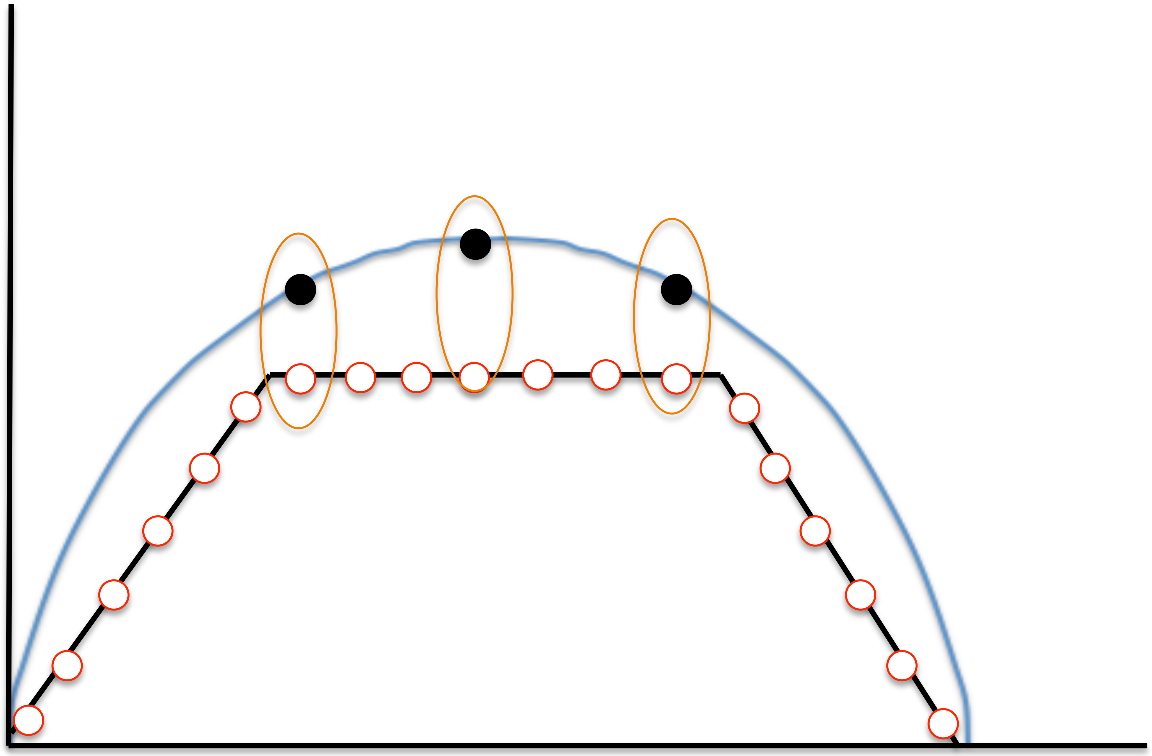
# Merge

- Add real samples to synthetic



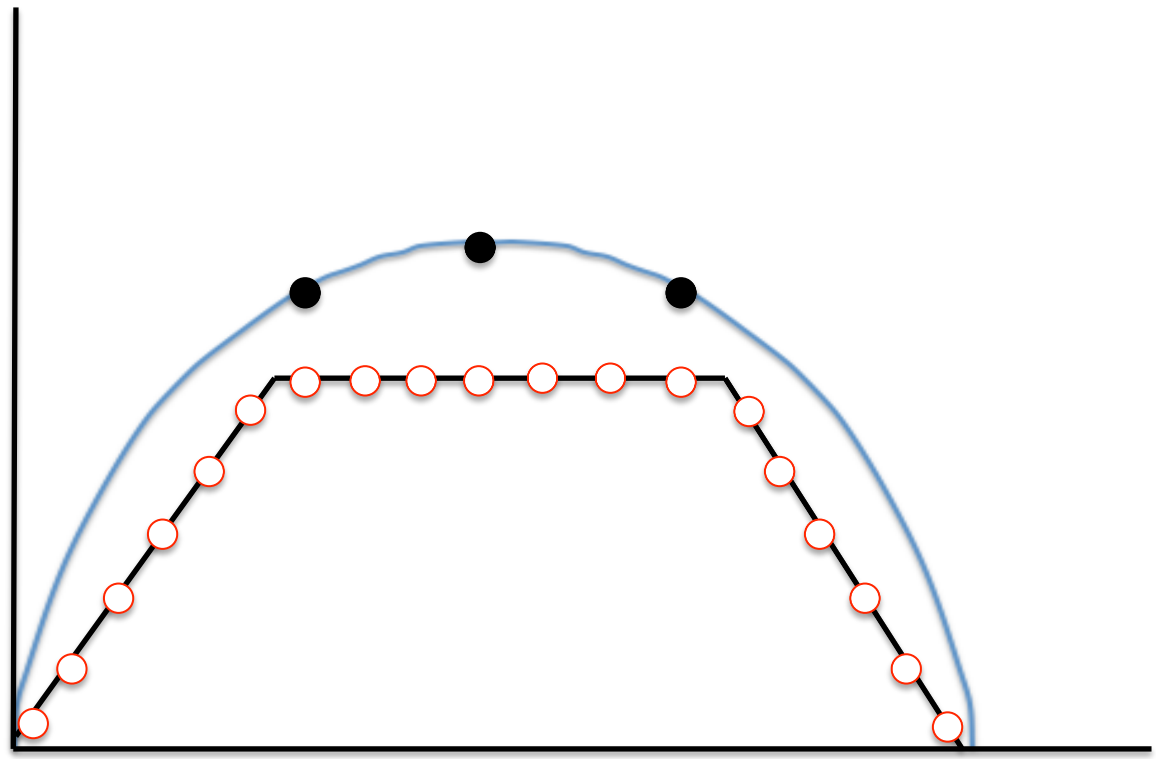
# Merge

- Problem: same/near samples have diff. output



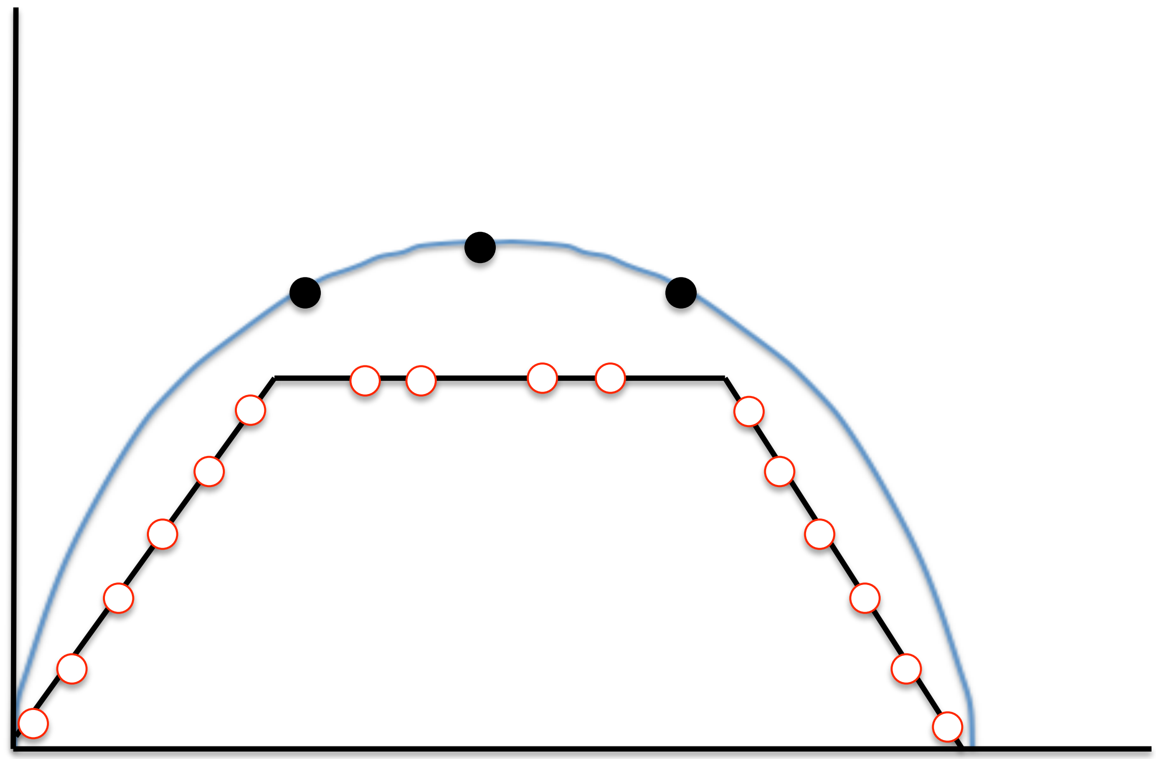
# Replace Nearest Neighbor (RNN)

- Remove nearest neighbor



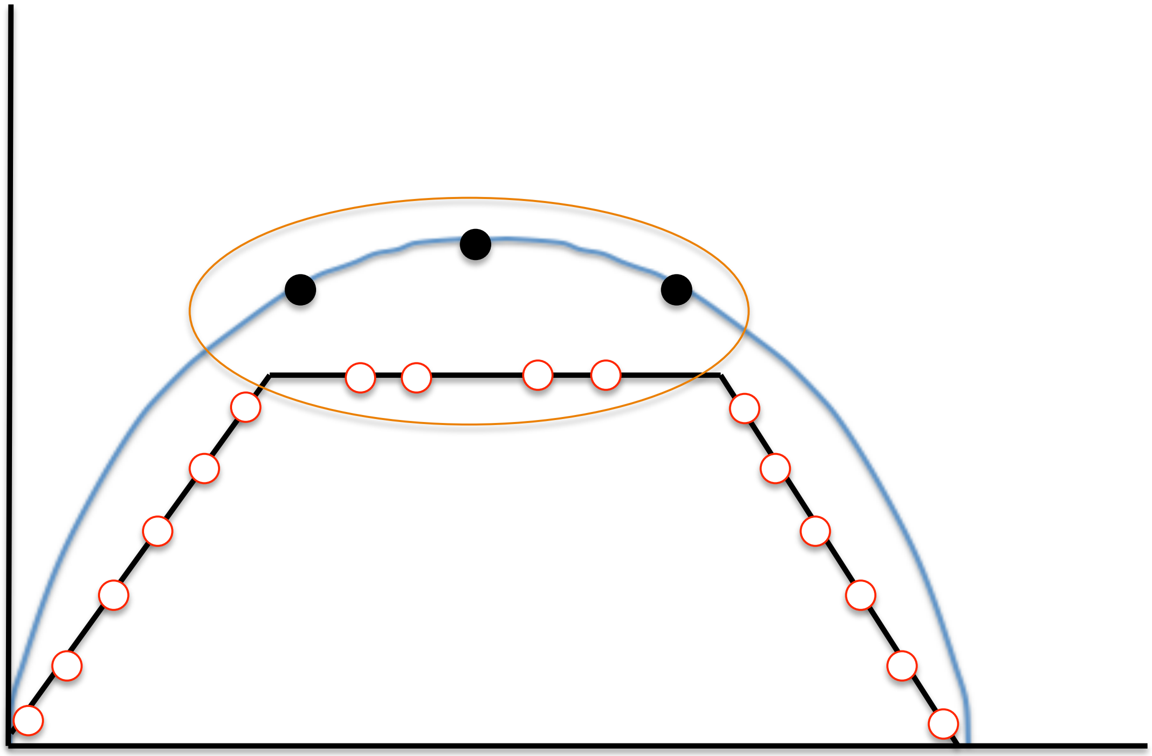
# Replace Nearest Neighbor (RNN)

- Preserve distribution…



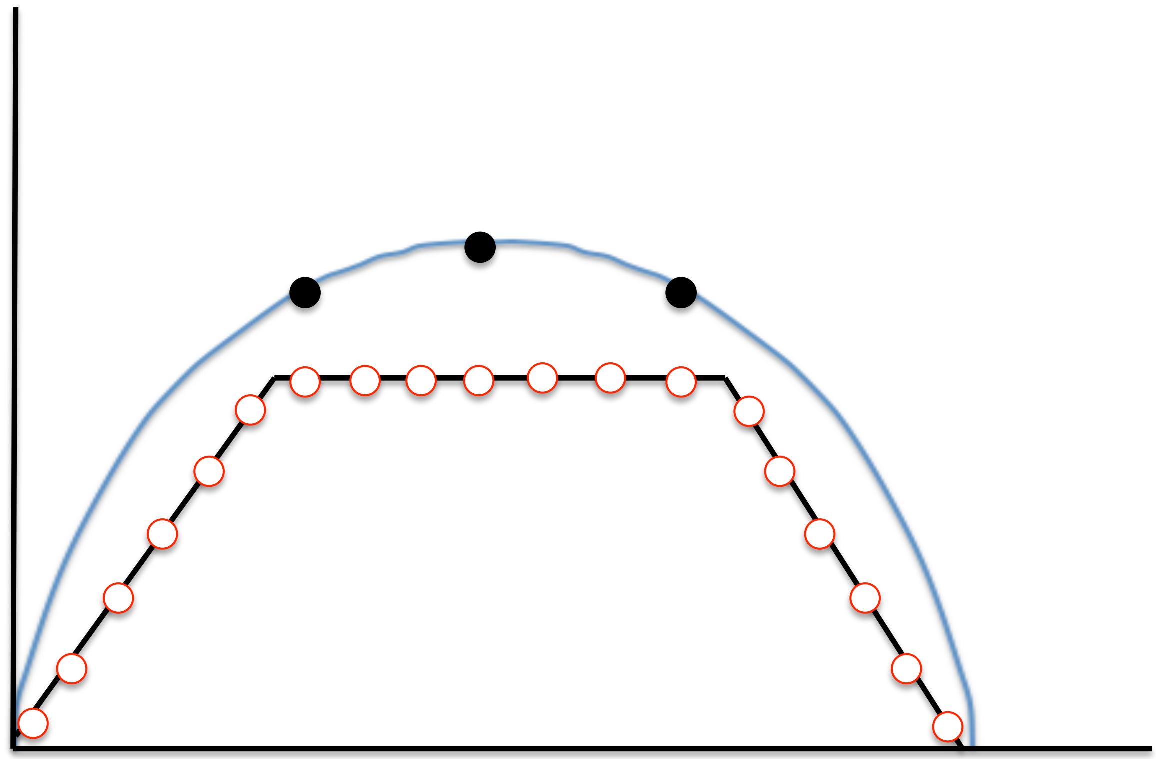
# Replace Nearest Neighbor (RNN)

- … but may induce alternating outputs



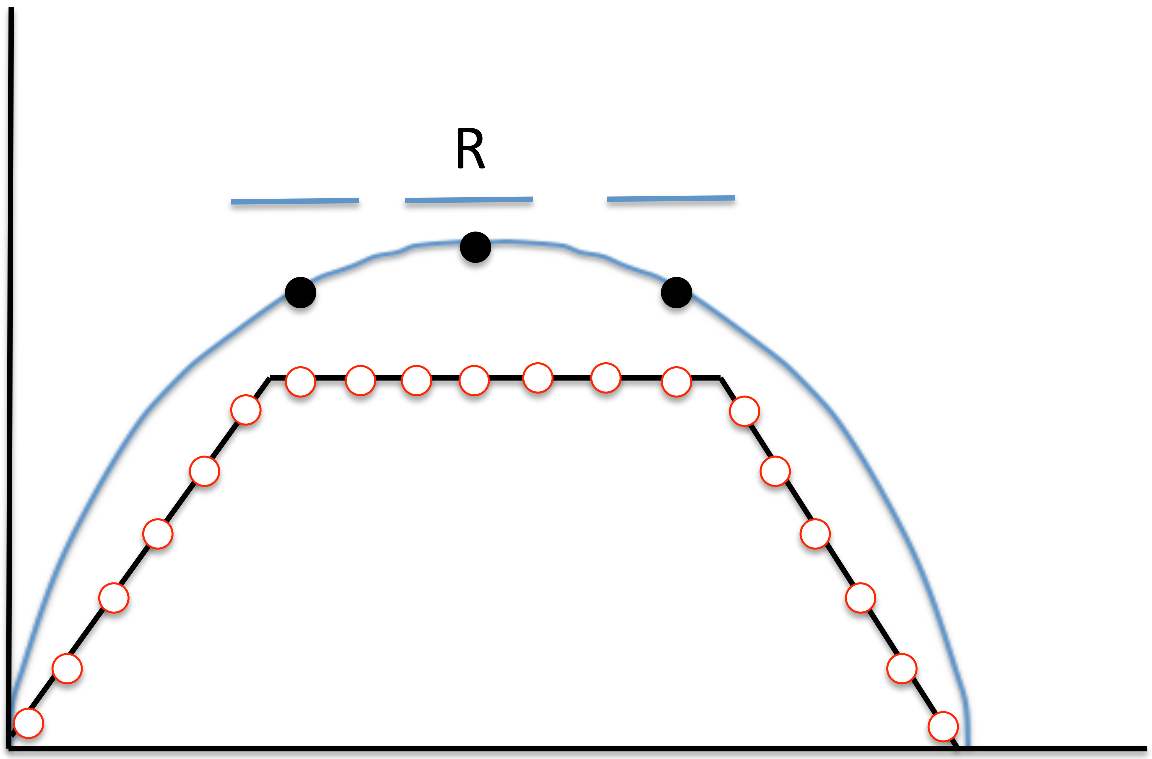
# Replace Nearest Region (RNR)

- Add real and **remove** synth. samples in a radius



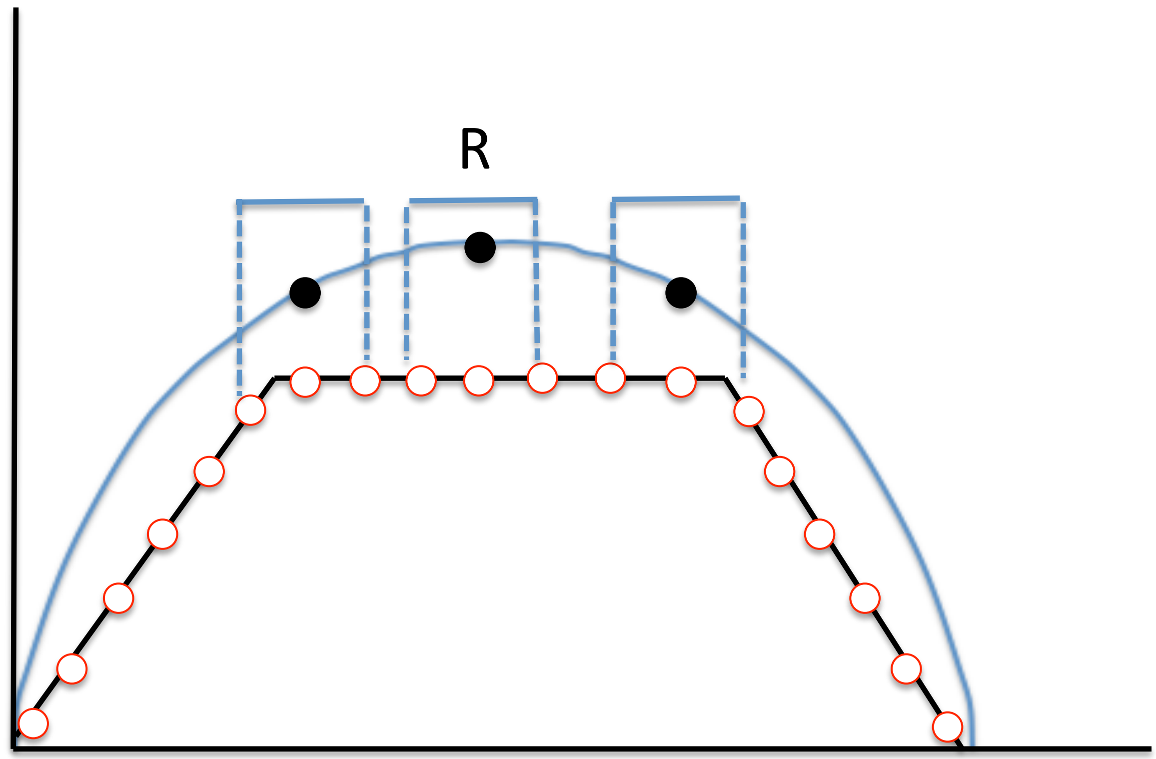
# Replace Nearest Region (RNR)

- R = radius defining neighborhood



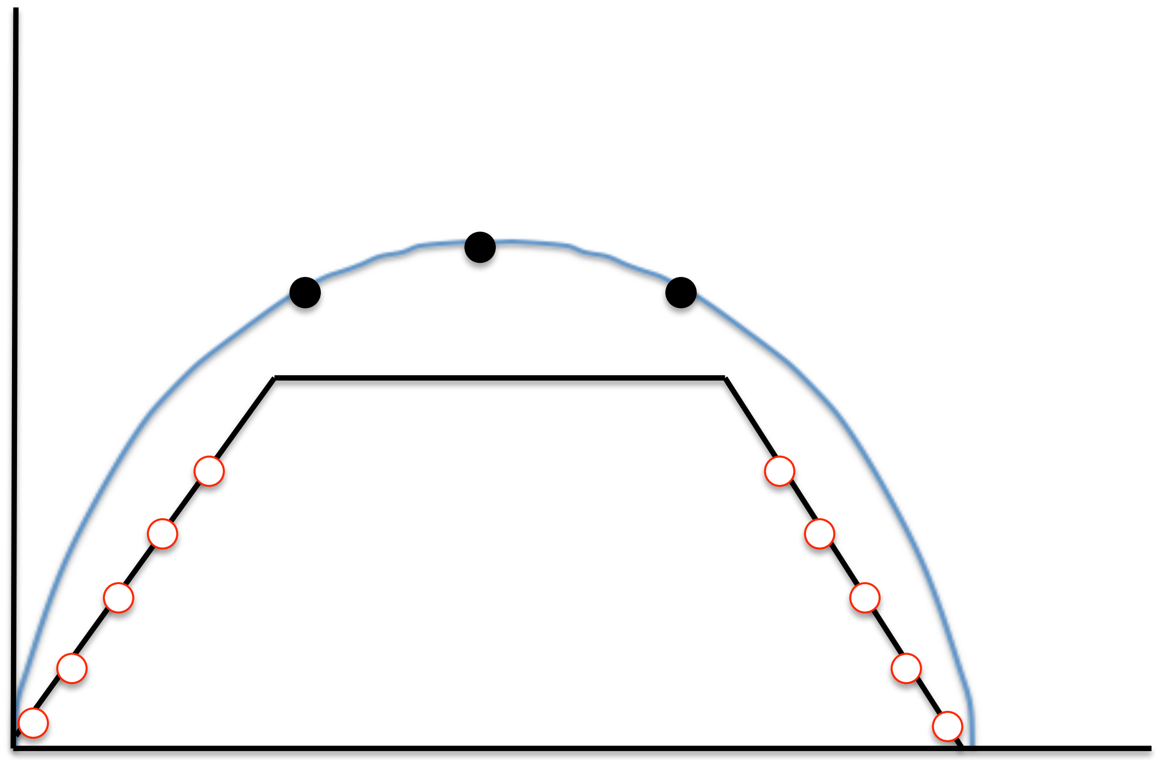
# Replace Nearest Region (RNR)

- R = radius defining neighborhood



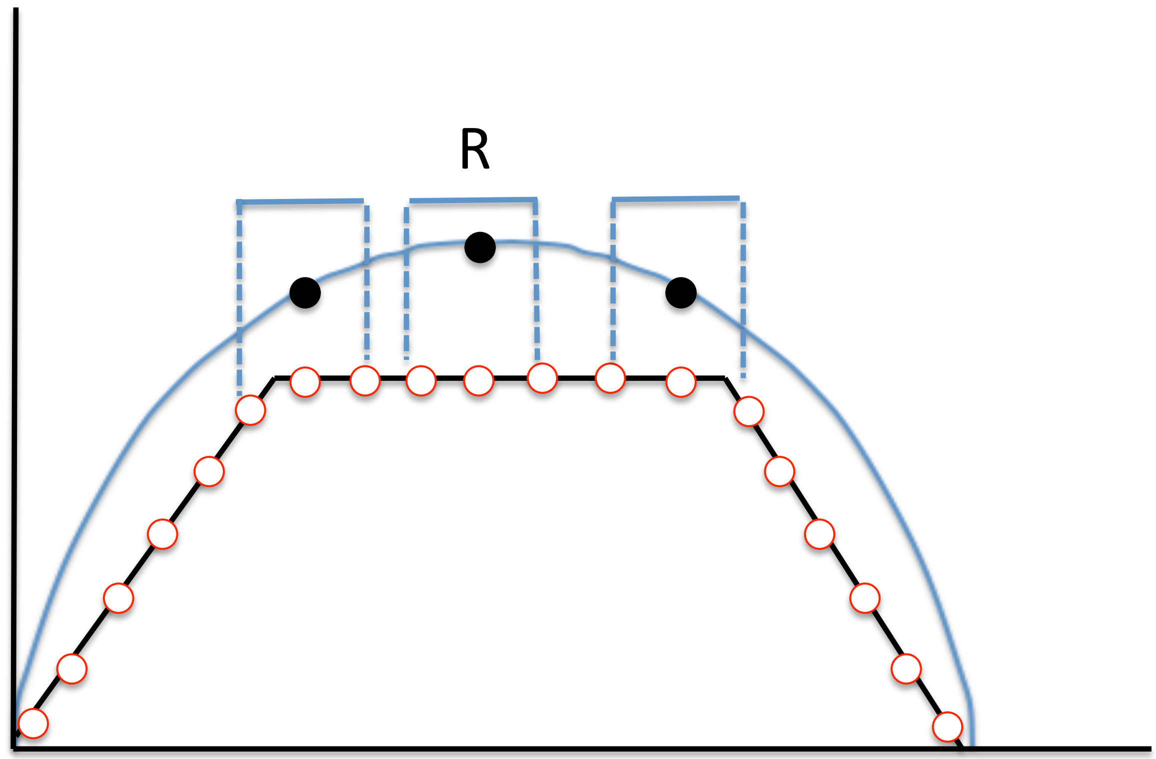
# Replace Nearest Region (RNR)

- Skew samples' distribution



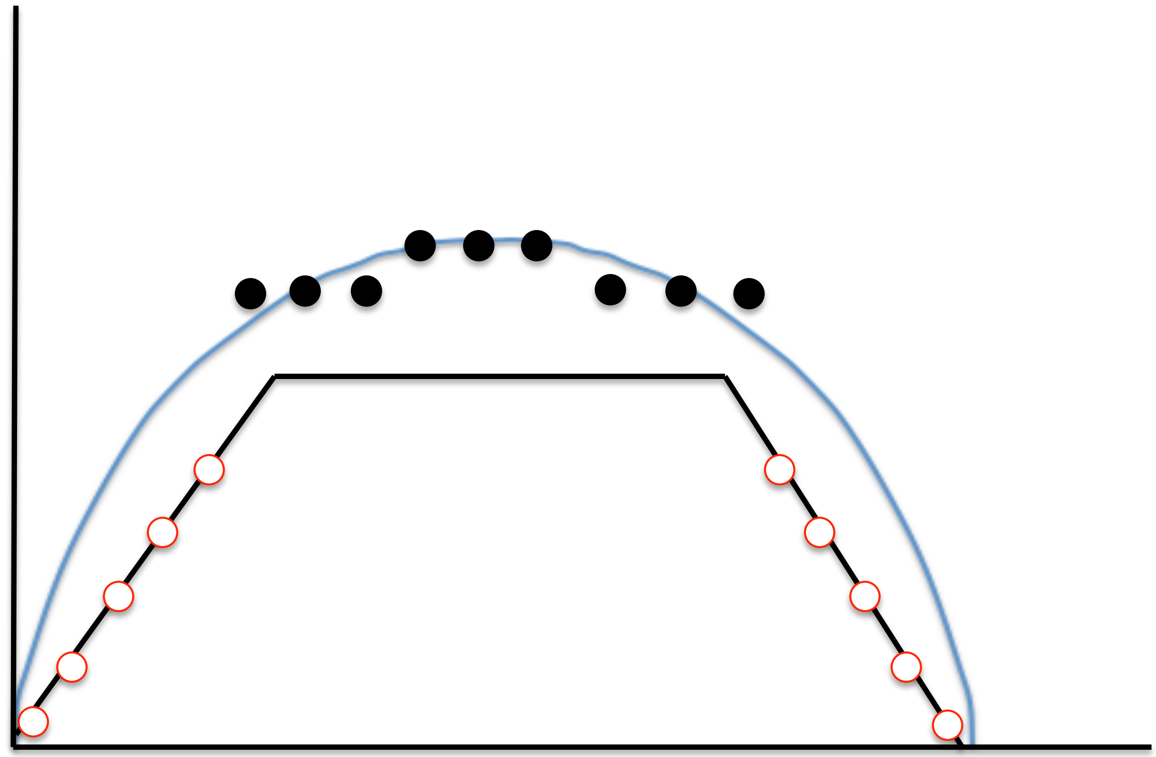
# Replace Nearest Region 2 (RNR2)

- **Replace** all synthetic samples in a radius R



# Replace Nearest Region 2 (RNR2)

- Maintain distribution, piecewise approximation



# Weighting

- Give more relevance to some samples

Fit better the model around **real** samples

- "Trust" **real** samples more than synthetic ones
- Useful especially in Merge

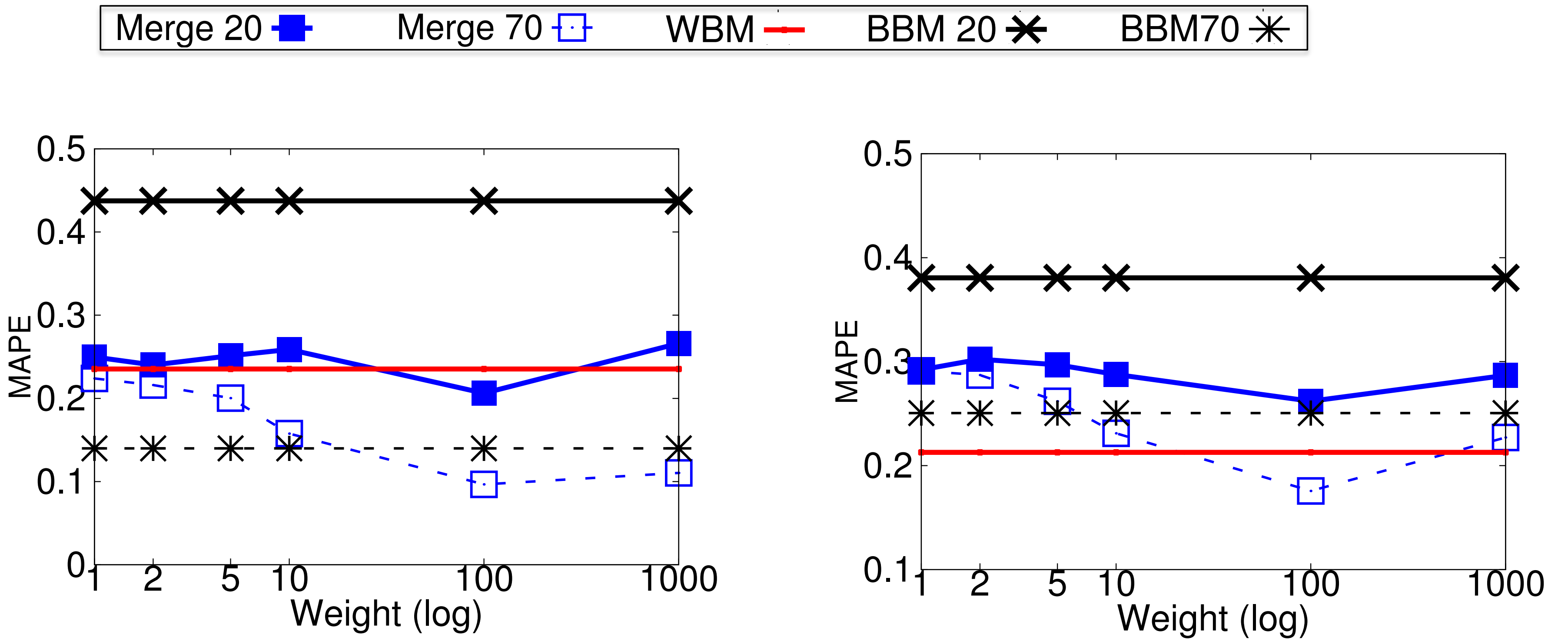
Too high can cause over-fitting!

- Learner fails to generalize

# Evaluation

- Case studies
  - Response time in Total Order Broadcast (TOB)
    - building block at the basis of many DTM
    - 2-dimensional yet highly nonlinear perf. Function
  - Throughput in Distributed TM (Infinispan)
    - 7-dimensional performance function

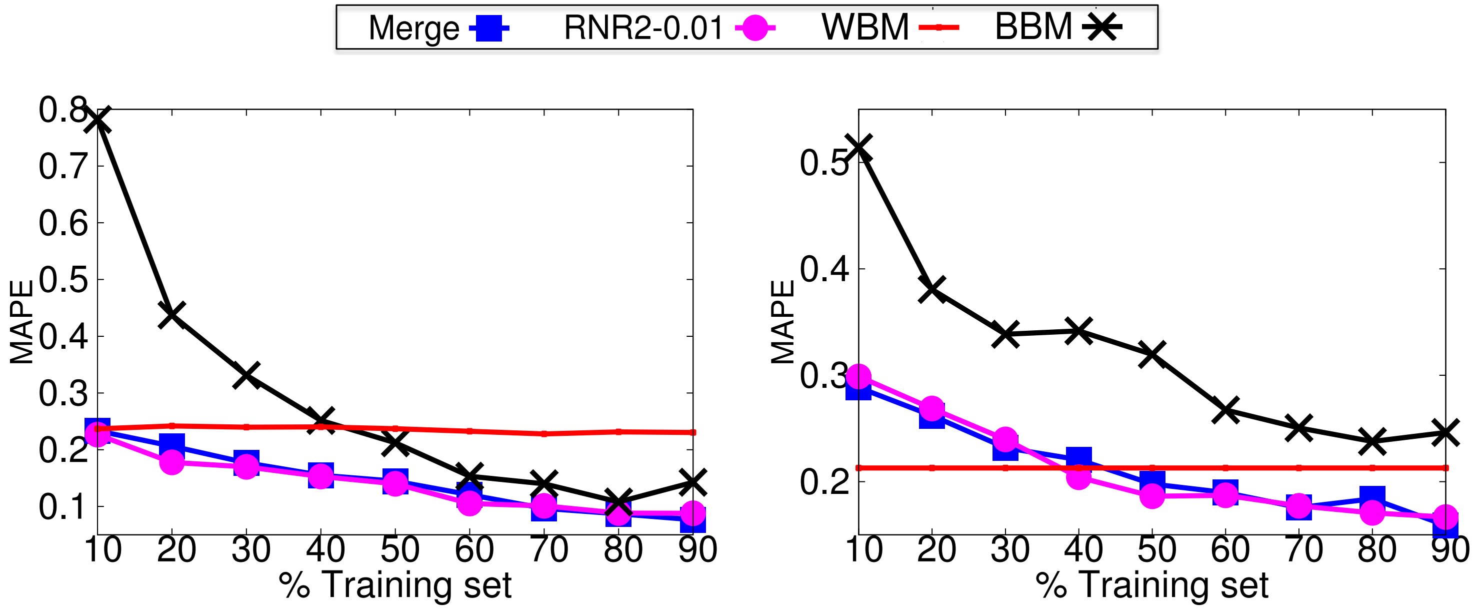
# Weighting




TOB
(10K synthetic samples)

DTM
(10K synthetic samples)

# Update function

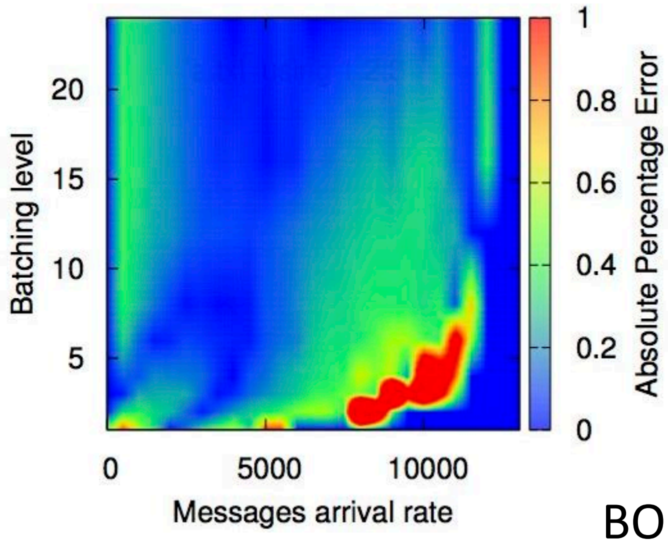



(a) TOB (b) DTP

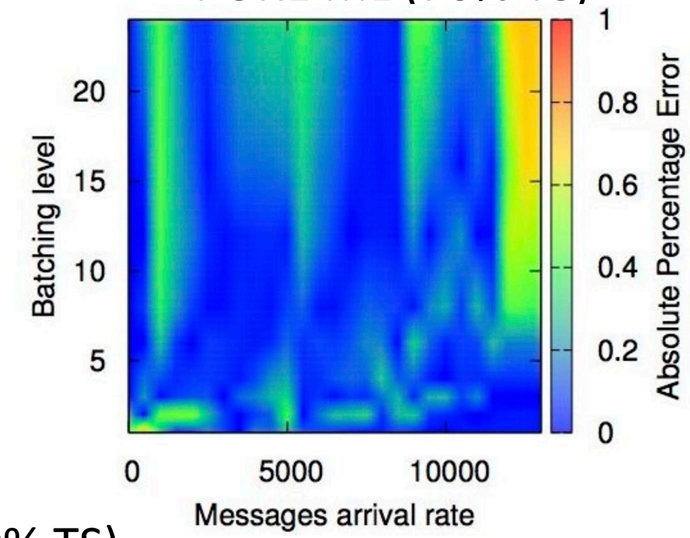
- In both considered case studies, simplicity pays off:
  - the Merge policy performs analogously to RNR2
  - …but, unlike RNR2, Merge is parameter-free

# Visualizing the correction

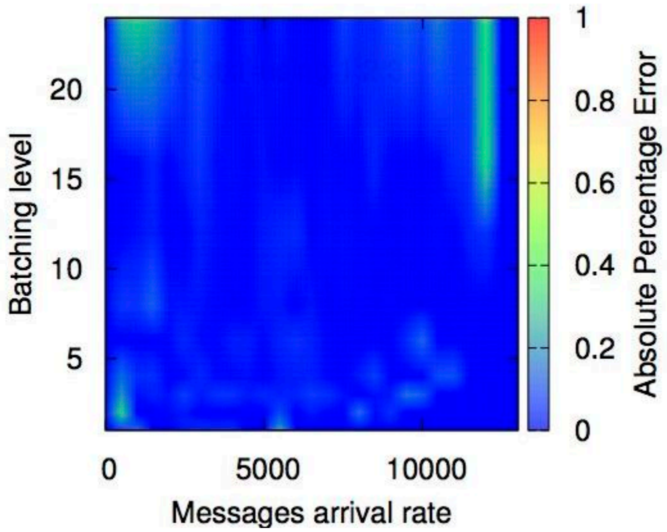
BASE MODEL




PURE ML (70% TS)




BOOTSTRAPPED ML (70% TS)



# Gray box modeling

- Will present three methodologies:

Divide and conquer

Bootstrapping

Hybrid ensembling

Hybrid boosting

Probing

Hybrid KNN

# Hybrid Boosting

Learning the error of a model on a function may be simpler than learning the function itself

- Chain composed by AM + cascade of ML
- $ML_1$ trained over residual error of AM
- $ML_i$, i>1 trained over residual error of $ML_{i-1}$

# Training and Querying Hyboost

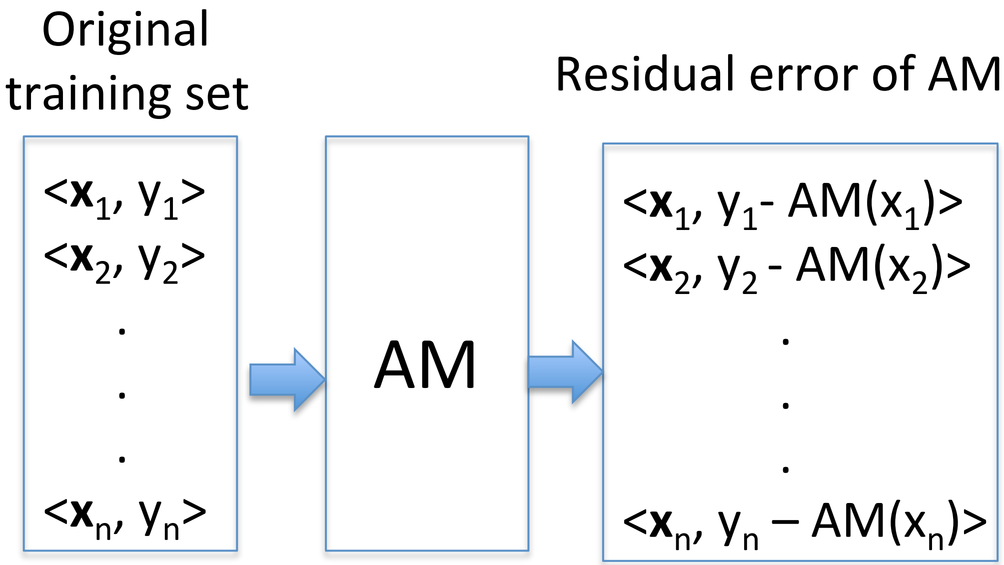
## Training

Original training set

$<\mathbf{x}_1, y_1>$
$<\mathbf{x}_2, y_2>$
.
.
.
$<\mathbf{x}_n, y_n>$

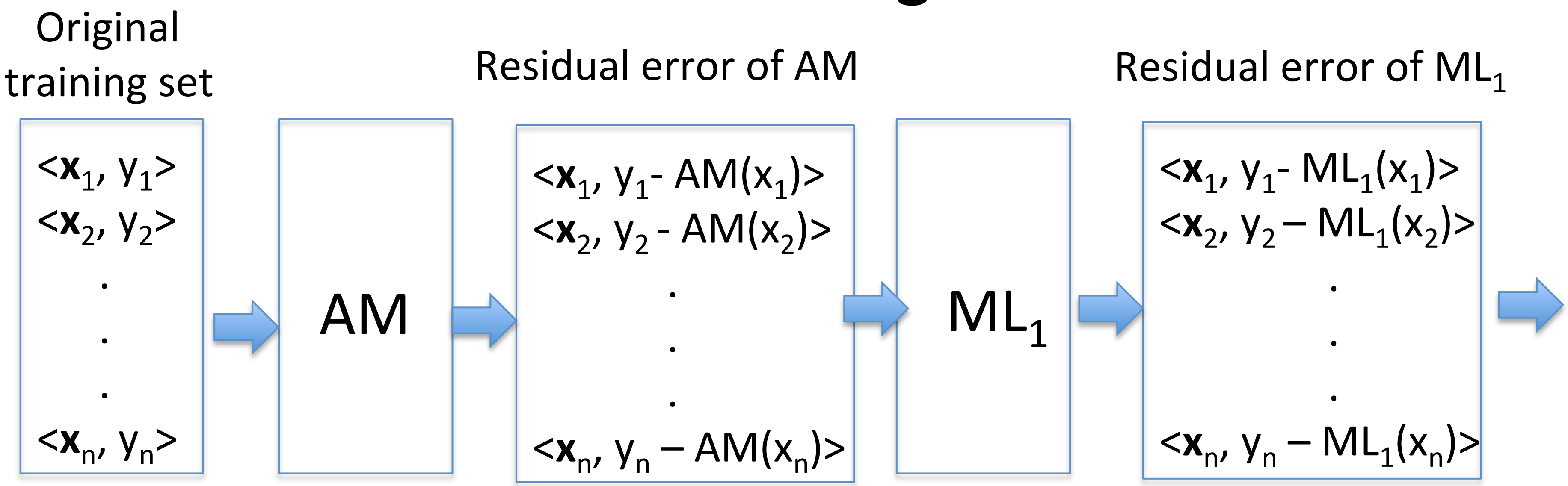
# Training and Querying Hyboost

## Training



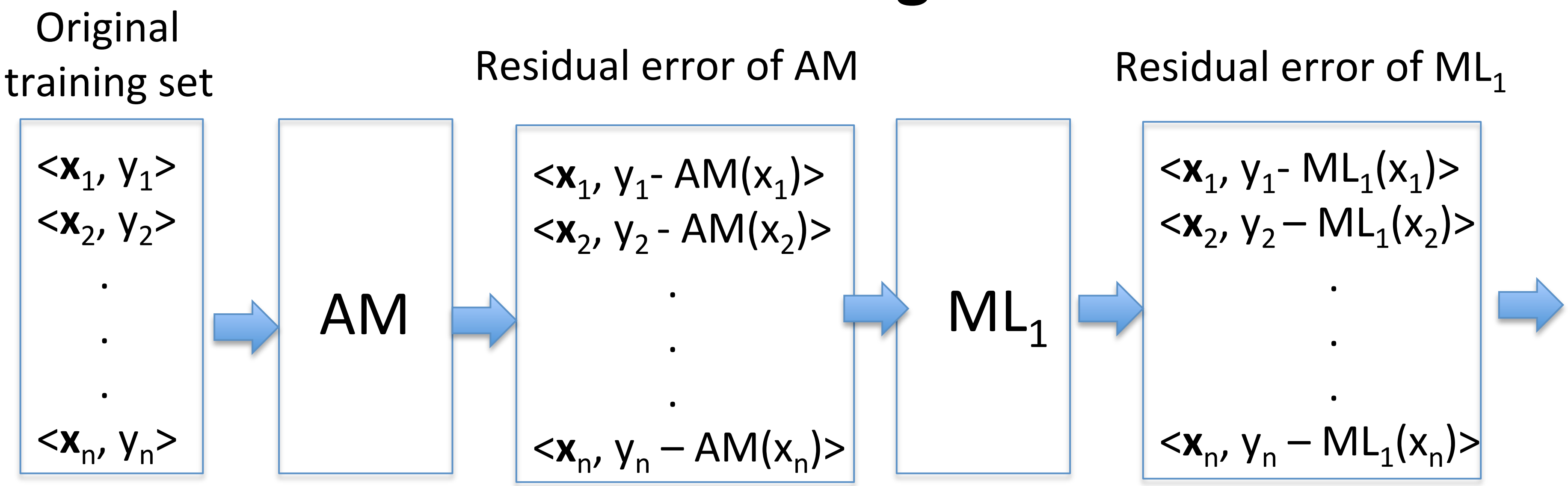
# Training and Querying Hyboost

## Training



# Training and Querying Hyboost

## Training

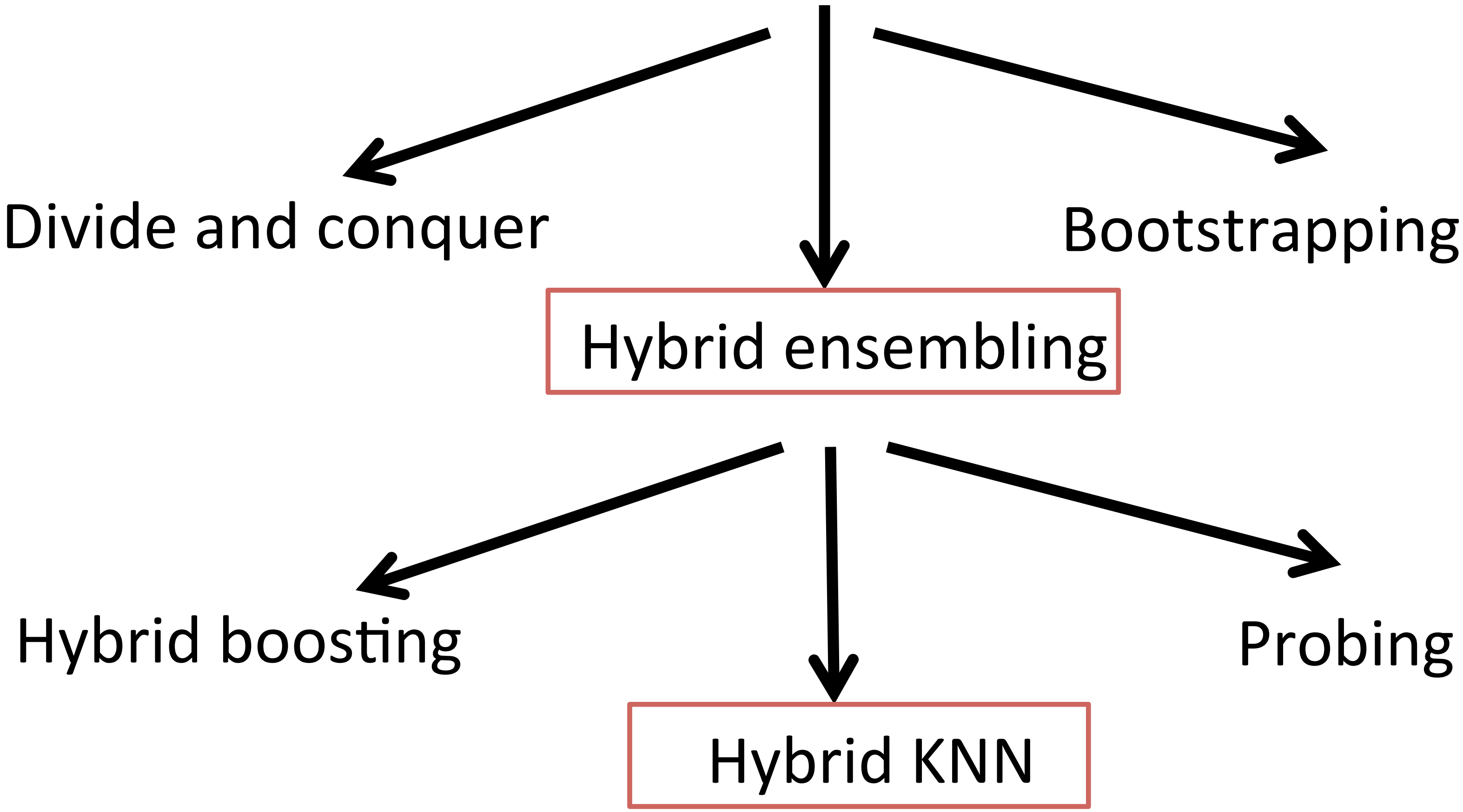



## Query

$$F(\mathbf{x}) = AM(\mathbf{x}) + ML_1(\mathbf{x}) + \ldots + ML_m(\mathbf{x})$$

# Gray box modeling

- Will present three methodologies:

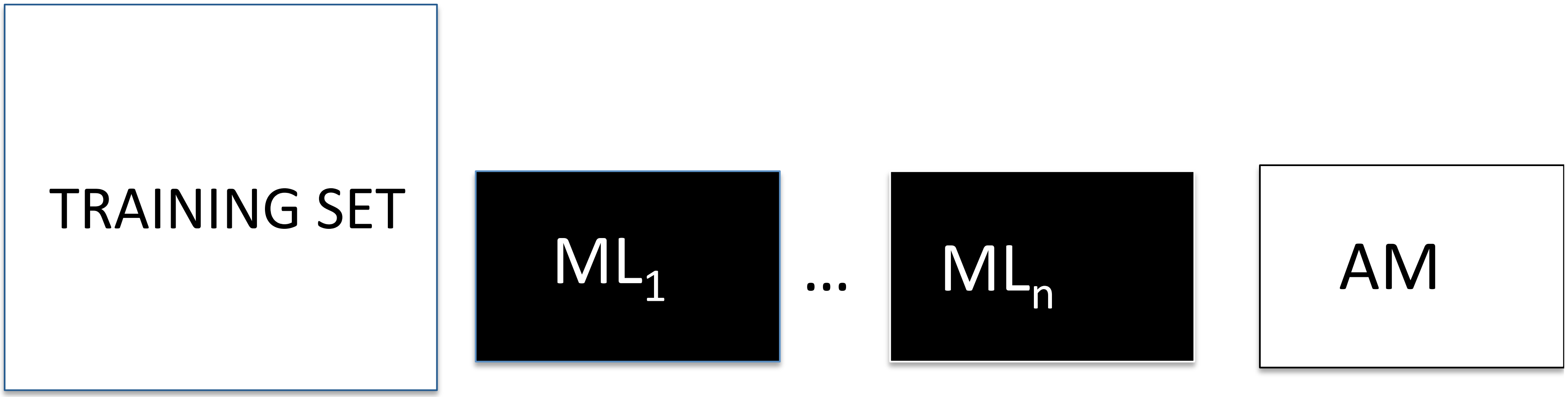


# Hybrid KNN

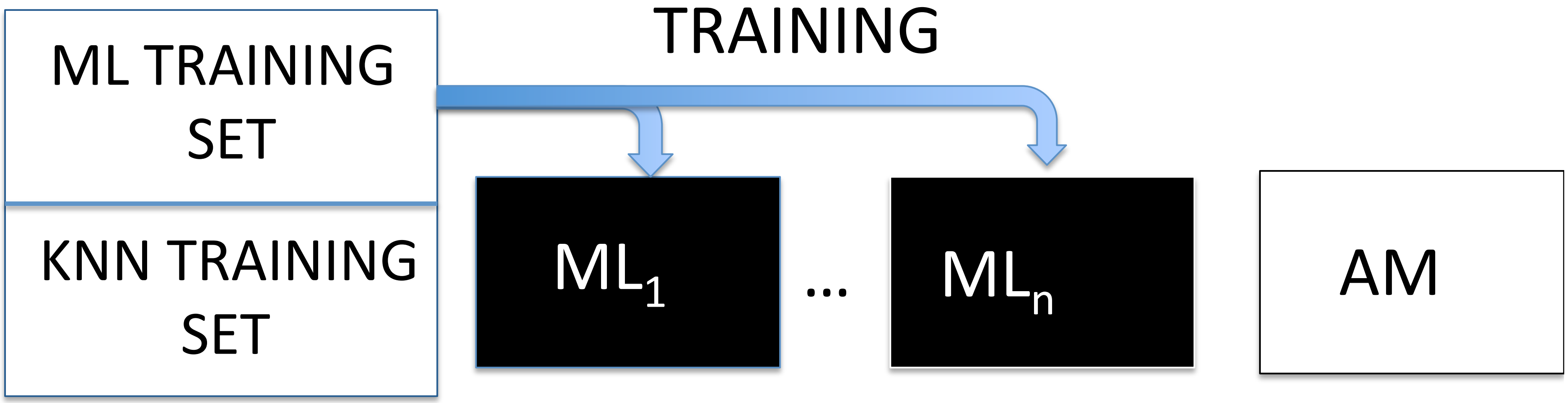
Predict performance of **x** with model that is supposed to be the most accurate for it

- Split training set D into D’, D’’
- Train $ML_1 \ldots ML_N$ on D’
  - ML can differ in nature, parameters, training set…
- For a query sample z
  - Pick the K training samples in D’’ closer to z
  - Find the model with lowest error on the K samples
  - Use such model to predict f(**x**)

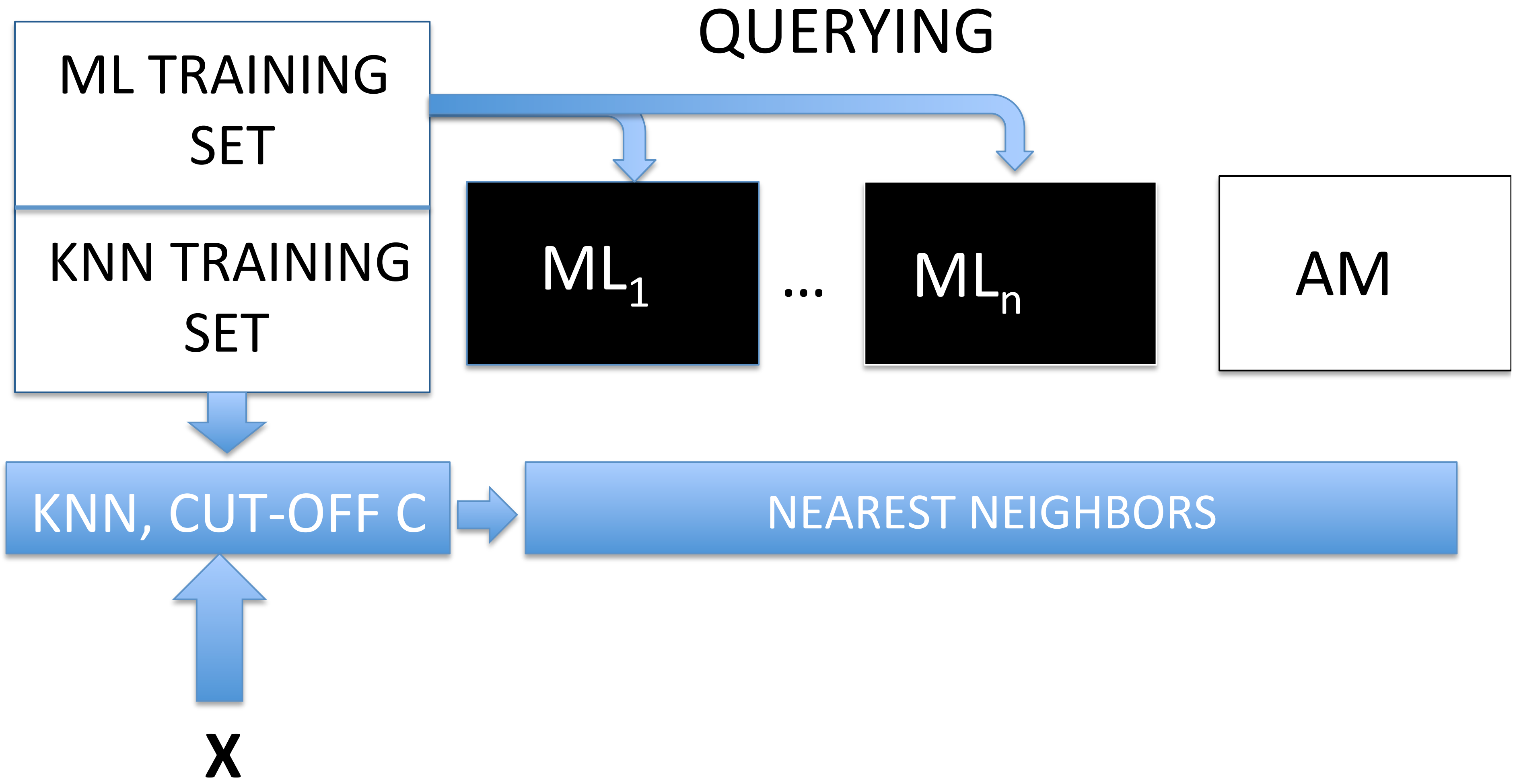
# KNN Training and Querying



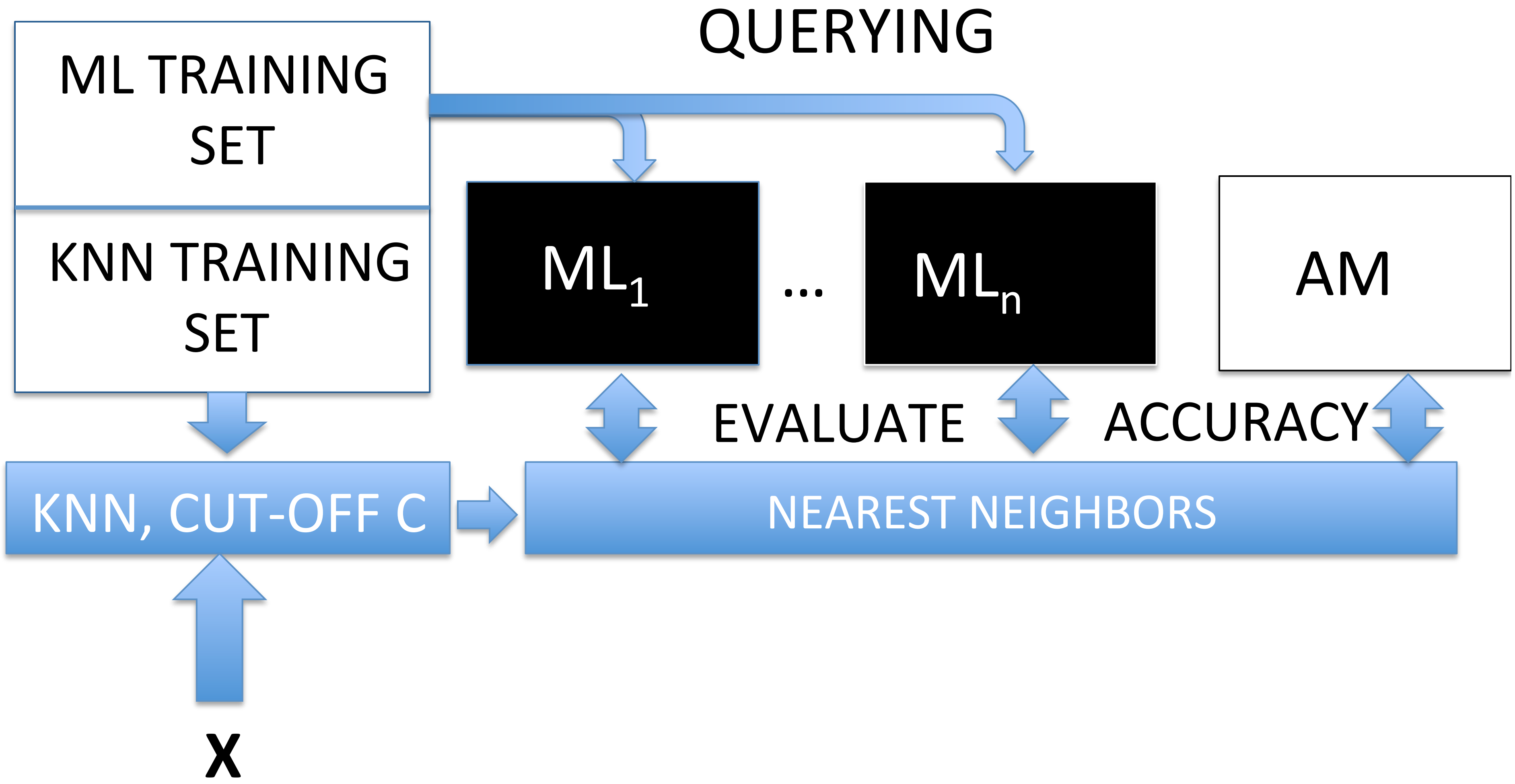
# KNN Training and Querying



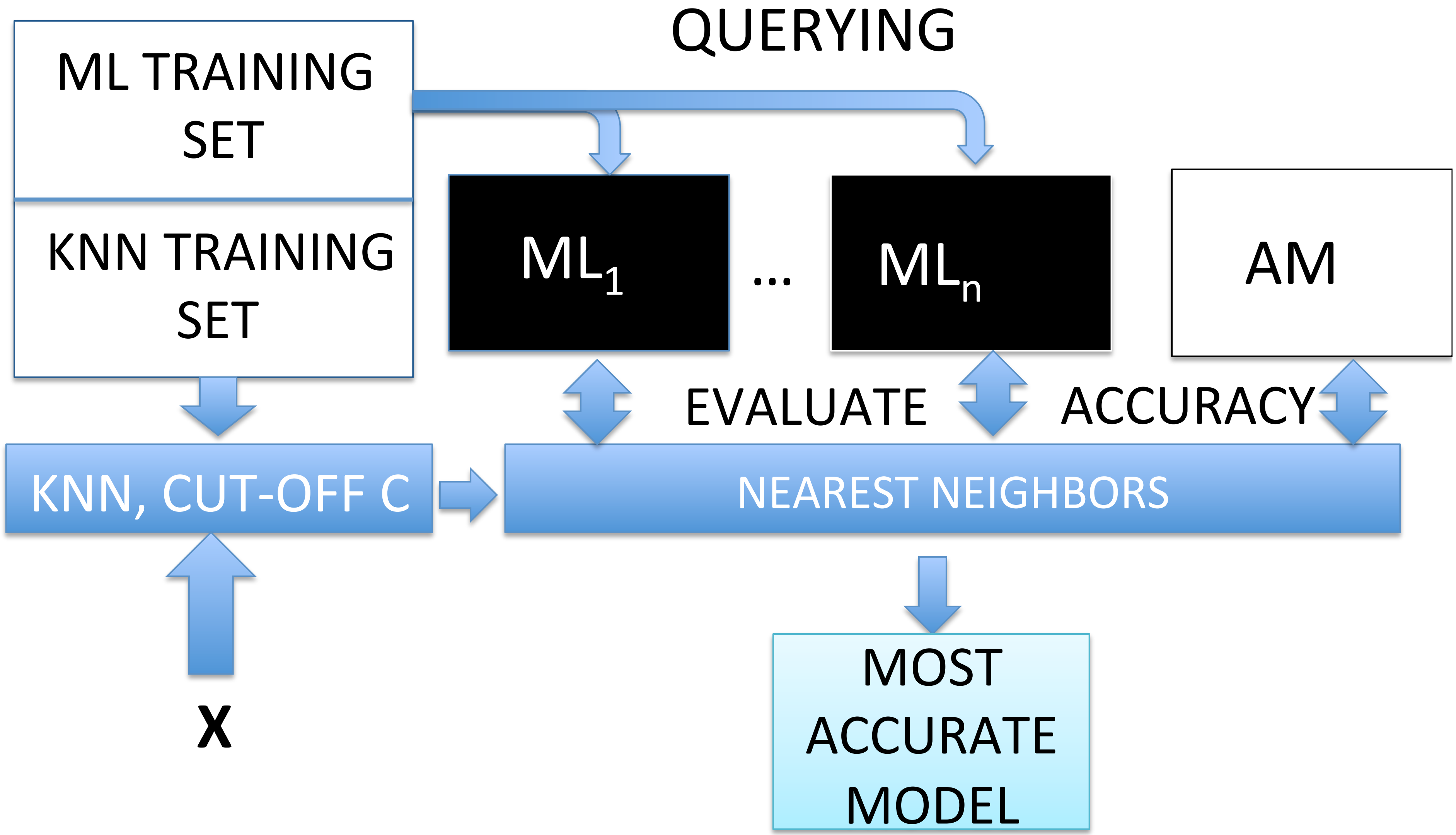
# KNN Training and Querying



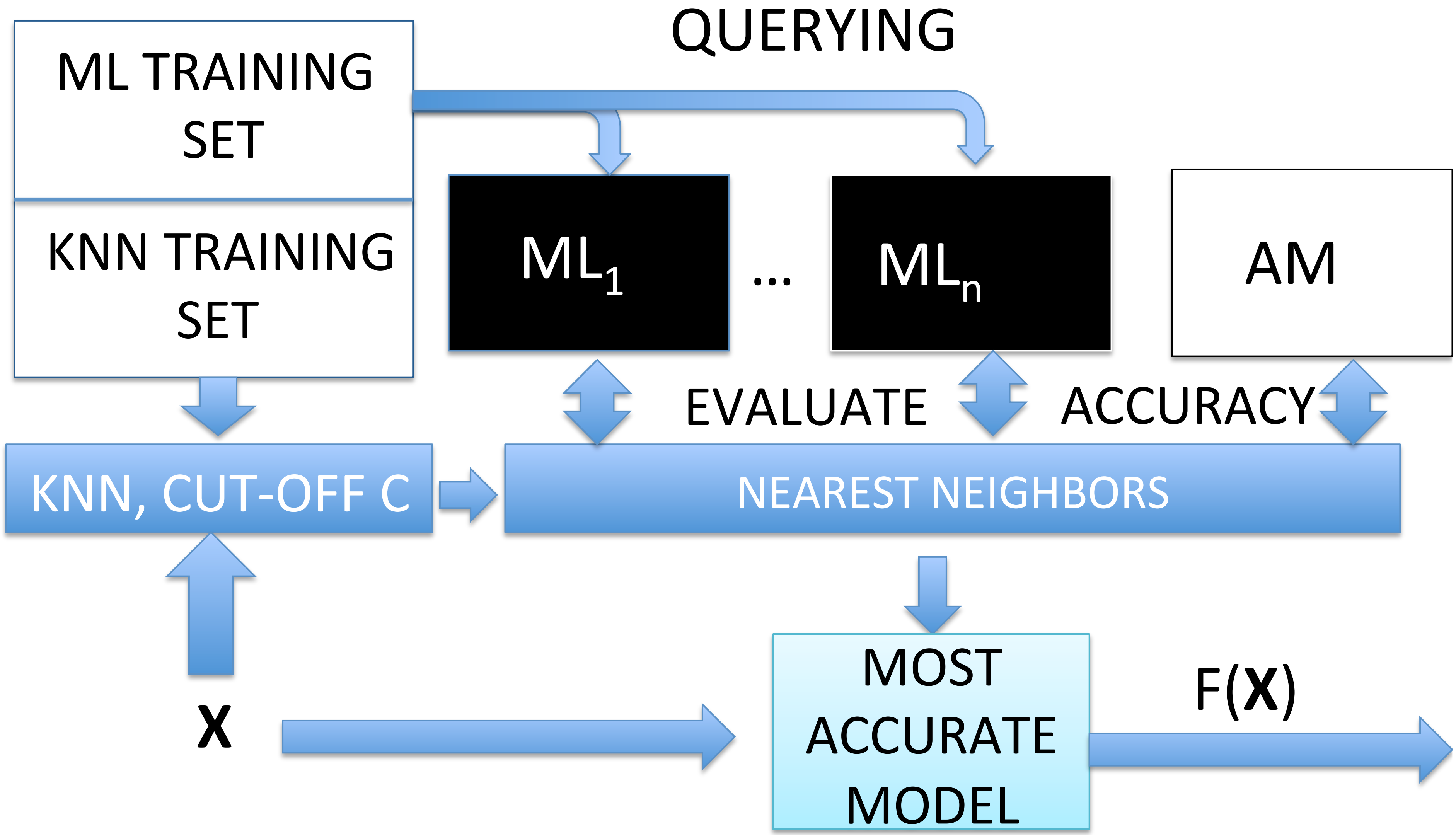
# KNN Training and Querying



# KNN Training and Querying



# KNN Training and Querying



# Gray box modeling

- Will present three methodologies:

Divide and conquer

Hybrid ensembling

Bootstrapping

Hybrid boosting

Hybrid KNN

Probing

# Probing

- Build a ML model as specialized as possible
  - Use AM where it is accurate
  - Train ML only where AM fails

- Differences w.r.t. KNN
  - Training: in KNN, ML is trained on all samples:
    - Here, ML trained on samples for which AM is inaccurate
  - Querying: In KNN, voting decides on ML vs AM
    - Here, binary classifier predicts when the AM is inaccurate

# Probing training and querying

**TRAINING**

Original training set

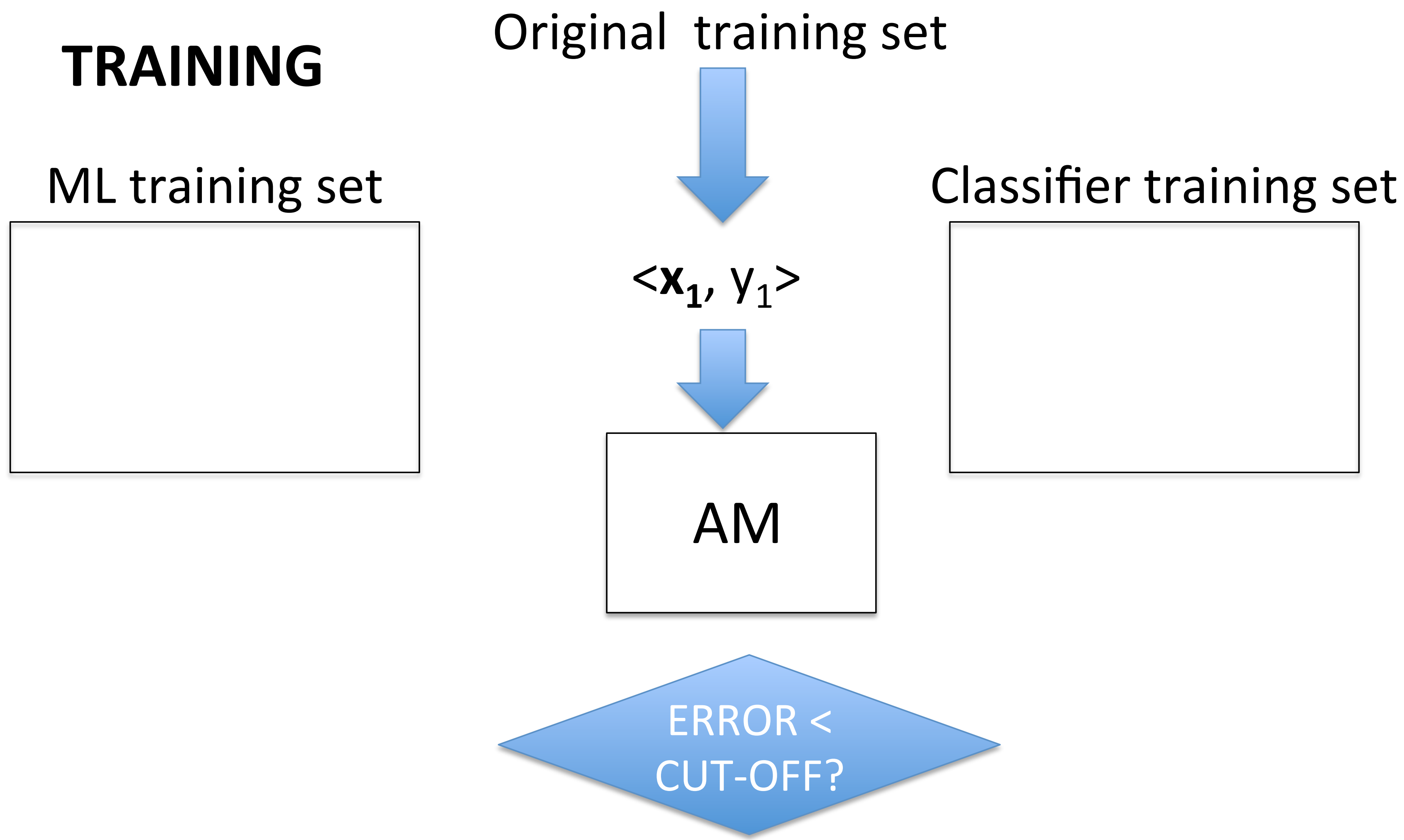
$<\mathbf{x}_1, y_1>$
$<\mathbf{x}_2, y_2>$
.
.
.
$<\mathbf{x}_n, y_n>$

ML training set

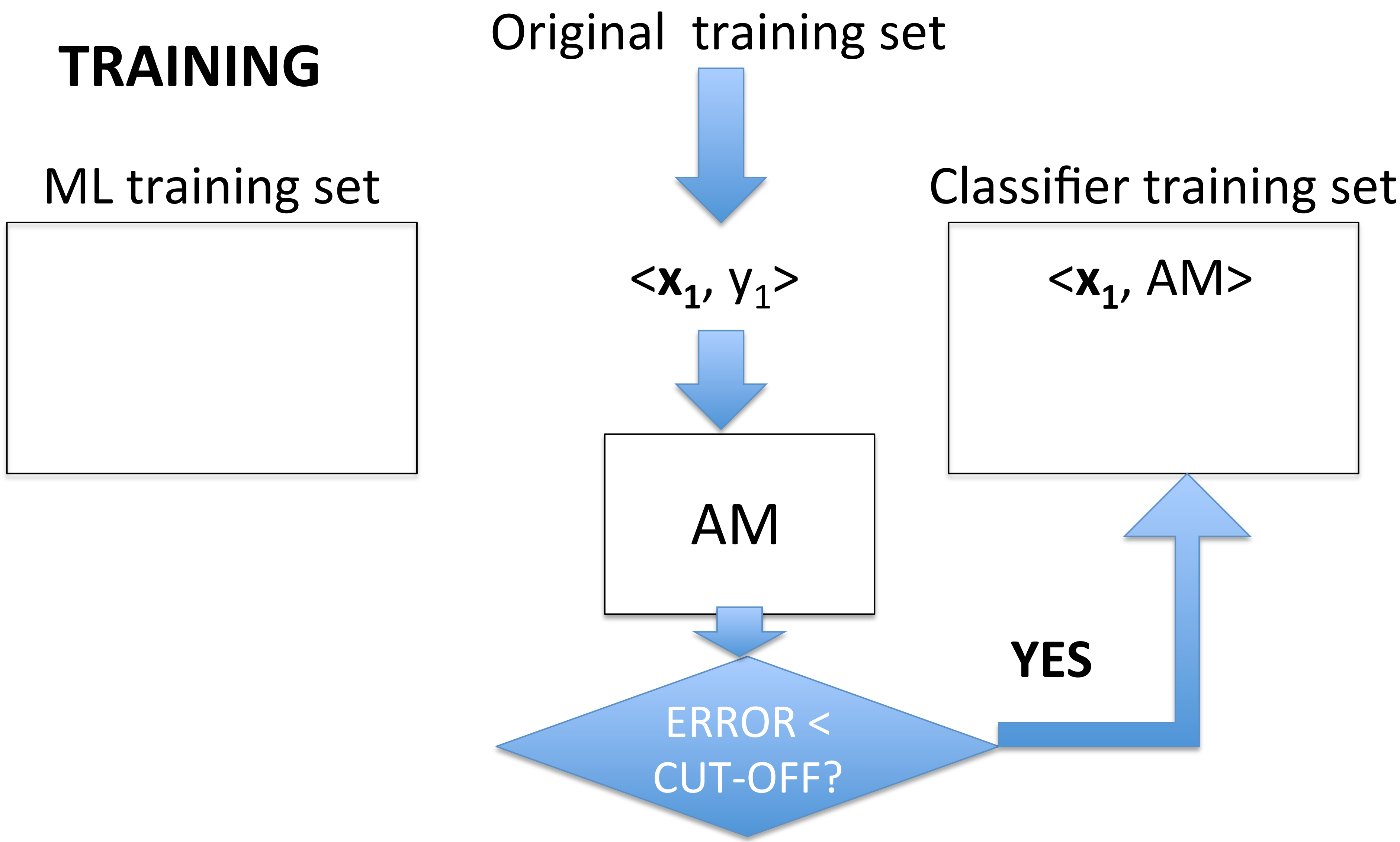
AM

Classifier training set

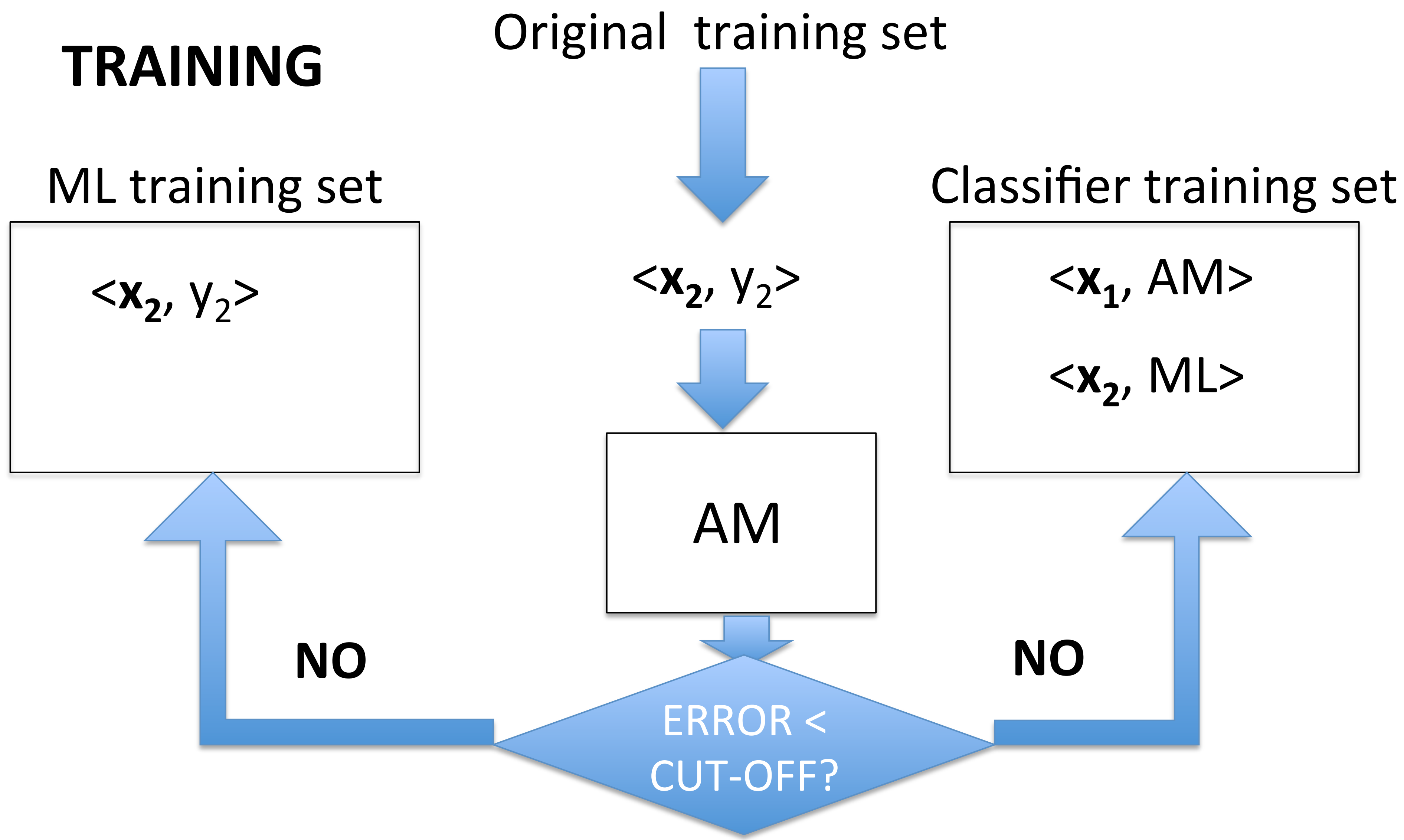
# Probing training and querying



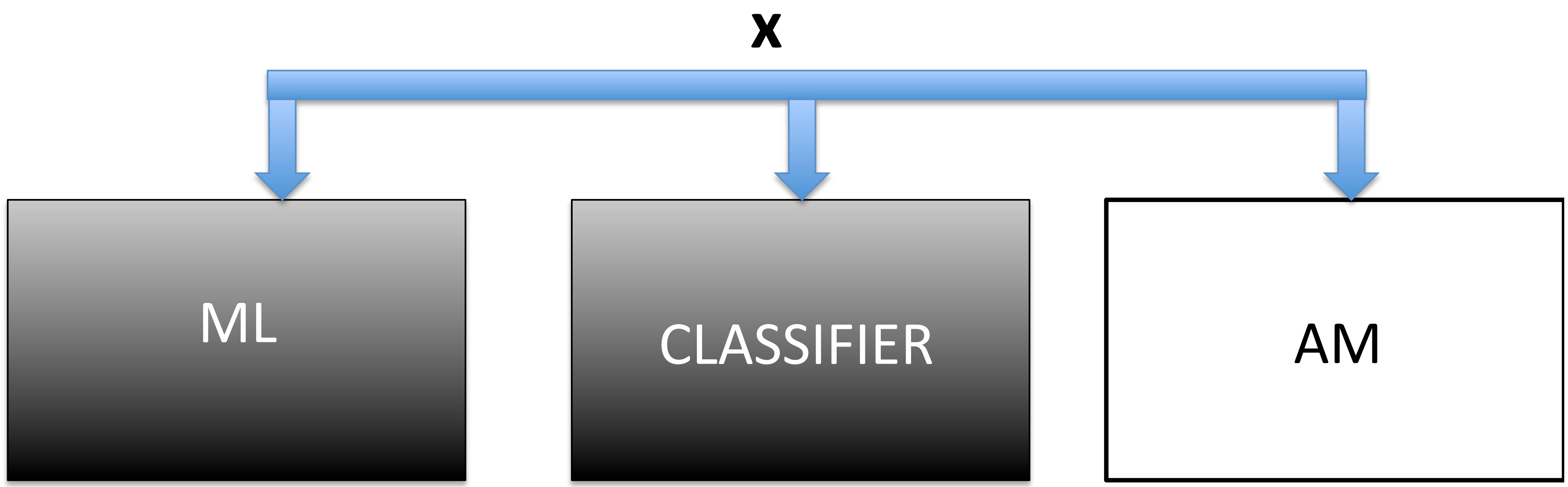
# Probing training and querying



# Probing training and querying



# Probing training and querying




**QUERYING**

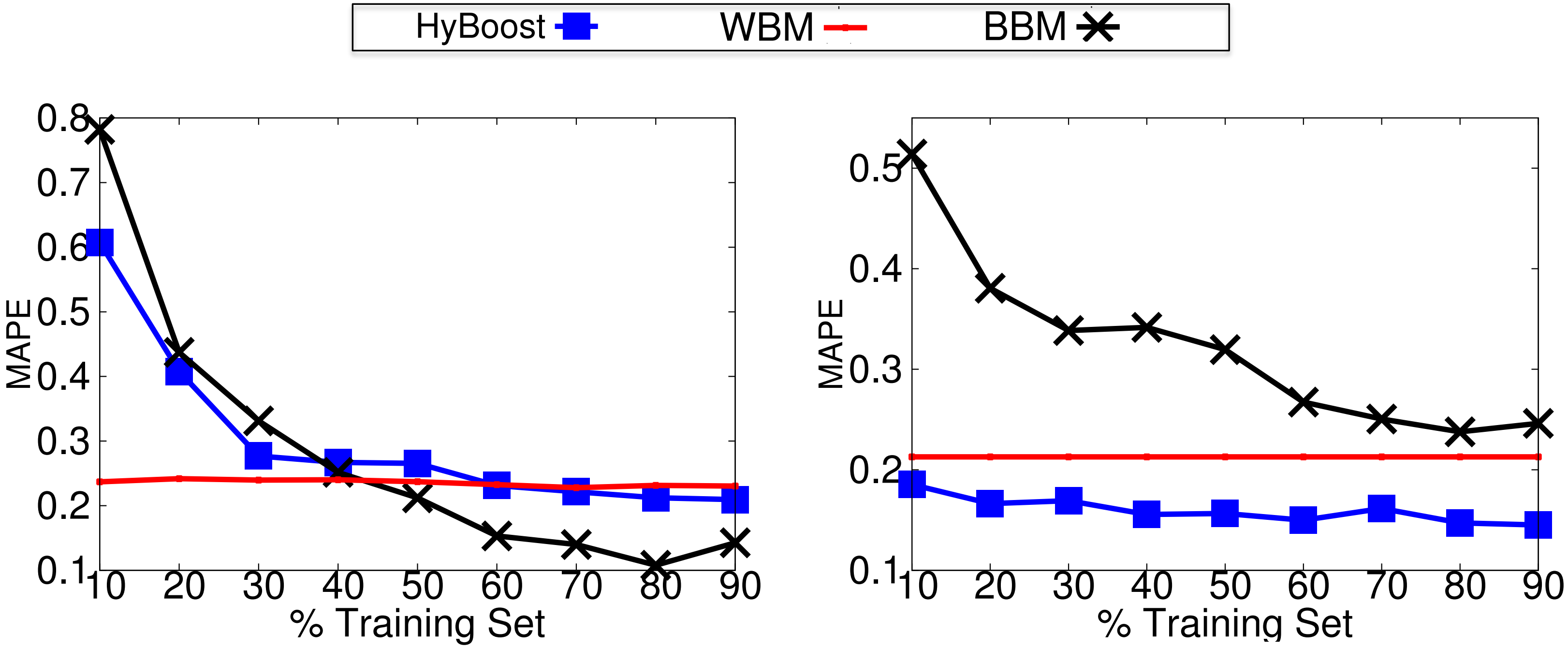
$$F(\mathbf{x}) = \begin{cases} AM(\mathbf{x}) & \text{if } Classify(\mathbf{x}) = AM \\ ML(\mathbf{x}) & \text{otherwise} \end{cases}$$

# Evaluation

- Sensitivity to meta-parameters
  - Hyboost
    - Size of the chain
  - Hybrid KNN
    - Proximity cut-off
  - Probing
    - Minimum AM’s accuracy cut-off

- Comparison among the techniques

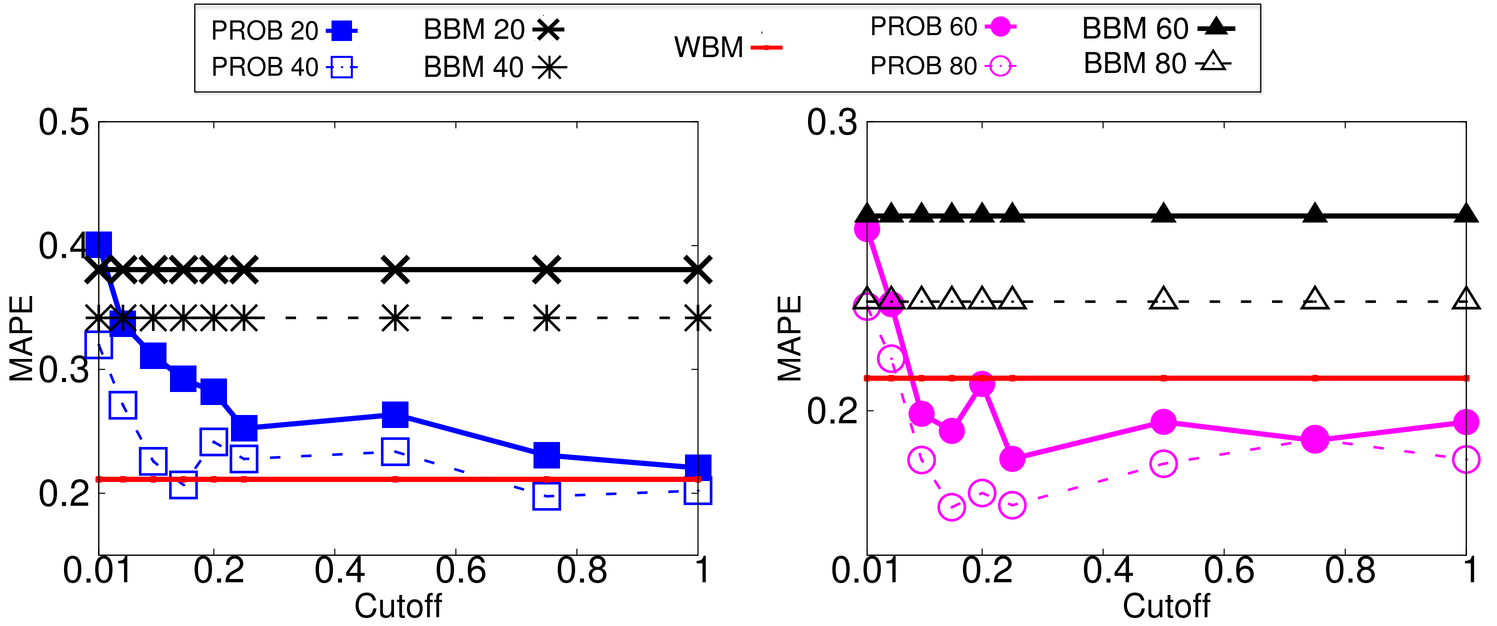
# HyBoost




(a) TOB (b) Infinispan

- Chain composed by AM + Decision Tree
- Longer chains yielded negligible improvements in the considered case studies

# Tuning of hyper-parameters matters

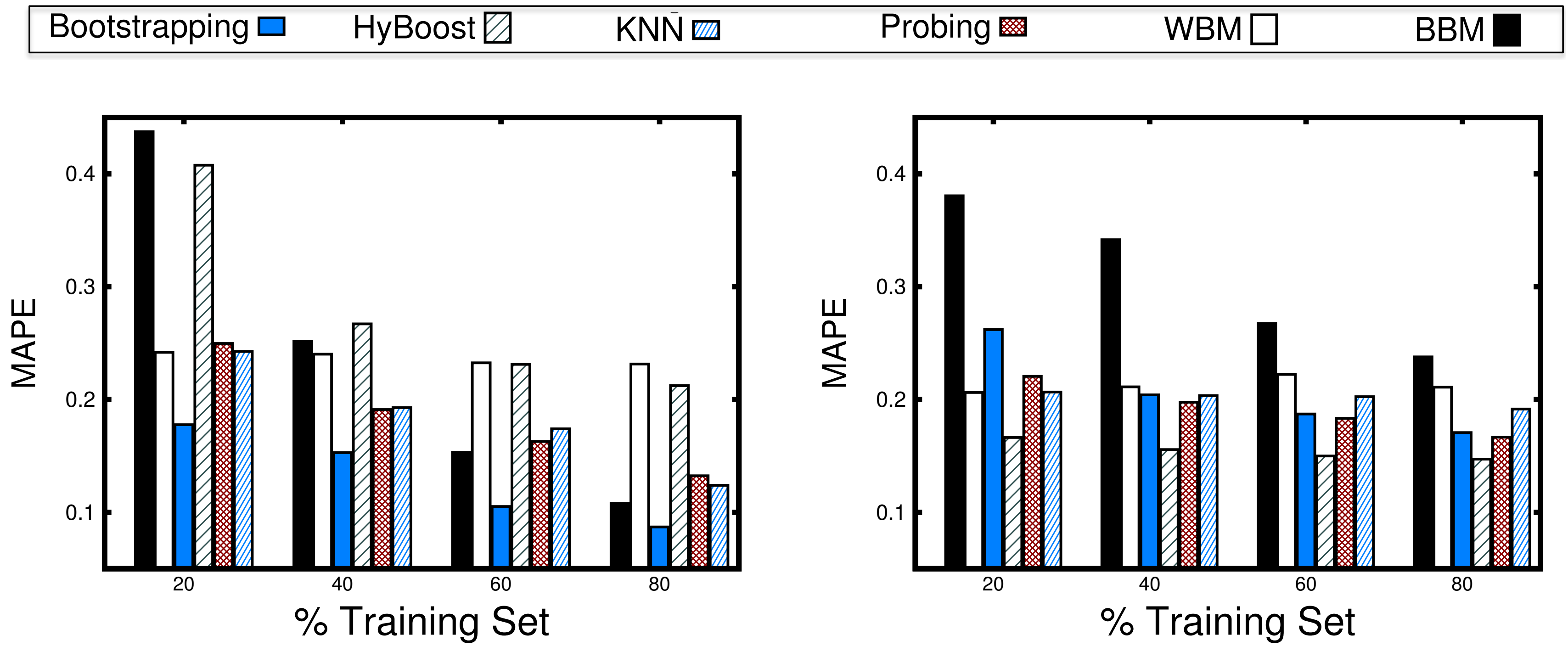



(a) 20% and 40% of training set (b) 60% and 80% of training set

- Comparison
  - Pure AM, Pure ML (Cubist, Decision tree regressor) vs
  - Probing (AM + Cubist)
- Analogous considerations hold for KNN

# No free lunch theorem strikes again

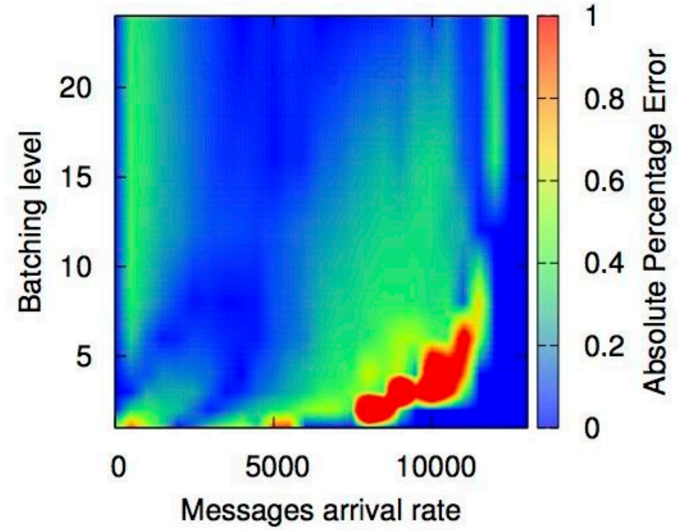
- No one-size-fits-all hybrid model exists
- Tackle choice of best hybrid model via cross-validation

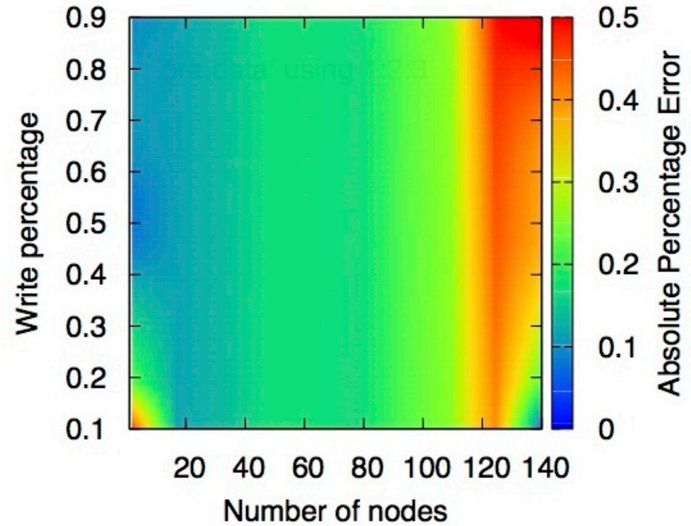



(a) TOB (b) DTP

# AM error vs optimal technique

- Error distribution of the base AM is key




- Hy-boost performed the best for DTM
  - Smooth error function is easy to learn
- Not the case for TOB
  - Highly localized errors better tackled via probing

# Concluding remarks: TM and Self-tuning

- Transactional memory is an attractive alternative to lock-based synchronization:
  - hides complexity behind intuitive abstraction
  - relevance amplified by integration with GCC, commodity (Intel's) and HPC (IBM's) CPUs
- Performance of TM is strongly affected by:
  - workload characteristics
  - choice of the TM implementation
  - plethora of implementation-dependent parameters
- Self-tuning is critical to ensure efficiency!

# Concluding remarks: Which modeling methodology?

- White and black box models can be effectively used in synergy
  - – Increased predictive power via analytical models
  - – Incremental learning capabilities via black box models
- Presented three gray box methodologies:
  - – Divide and conquer, Bootstrapping, Hybrid ensembling
  - – Design, implementation and application to (D)TM
- Careful choice of technique and parameters
  - Use standard techniques for hyper-parameters opt.

# Open questions

- Any other way of hybridizing Black and White modelling?
- Can we further combine them?
    - e.g. use a bootstrapped model in an ensemble?
- Can we infer the best gray box technique by analyzing the error function of the AM model?

# References


- [SPAA08] Torvald Riegel, Christof Fetzer, Pascal Felber, Automatic data partitioning in software transactional memories. SPAA 2008: 152-159
- [PPoPP08] Pascal Felber, Christof Fetzer, Torvald Riegel, Dynamic performance tuning of word-based software transactional memory. PPOPP 2008: 237-246
- [SASO12] D. Didona, D. Carnevale Paolo Romano, S. Galeani, An Extremum Seeking Algorithm for Message Batching in Total Order Protocols, IEEE International Conference on Self-Adaptive and Self-Organizing Systems (SASO 2012), Lyon, France, 10-14 Sept. 2012
- [Netys13] Diego Didona, Pascal Felber, Derin Harmanci, Paolo Romano and Joerg Schenker, Identifying the Optimal Level of Parallelism in Transactional Memory Systems, The International Conference on Networked Systems 2013, **BEST PAPER AWARD**
- [DSN13] M. Couceiro, P. Ruivo, Paolo Romano, L. Rodrigues, Chasing the Optimum in Replicated In-memory Transactional Platforms via Protocol Adaptation, The 43rd Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2013)
- [ICAC 13] Joao Paiva, Pedro Ruivo, Paolo Romano and Luis Rodrigues, AutoPlacer: scalable self-tuning data placement in distributed key-value stores, The 10th International Conference on Autonomic Computing (ICAC 2013), San Jose, CA, USA, 26-28 June 2013 - BEST PAPER AWARD FINALIST
- [PACT14] N. Diegues and Paolo Romano and L. Rodrigues, Virtues and Limitations of Commodity Hardware Transactional Memory, The 23rd International Conference on Parallel Architectures and Compilation Techniques (PACT 2014), August 2014
- [JPDC14] M. Castro et al., Adaptive thread mapping strategies for transactional memory applications, Journal of Parallel and Distributed Computing, Volume 74, Issue 9, September 2014
- [TAAS14] D. Didona, Paolo Romano, S. Peluso, F. Quaglia, Transactional Auto Scaler: Elastic Scaling of In-Memory Transactional Data Grids, ACM Transactions on Autonomous and Adaptive Systems (TAAS), 9, 2, 2014
- [ICPE15] D. Didona, Paolo Romano, F. Quaglia, E. Torre, Combining Analytical Modeling and Machine-Learning to Enhance Robustness of Performance Prediction Models, 6th ACM/SPEC International Conference on Performance Engineering (ICPE), Feb 2015

# THANK YOU

Questions?

romano@inesc-id.pt

www.gsd.inesc-id.pt/~romanop