Large Scale Machine Learning with the SimSQL System

Chris Jermaine
Rice University

Many Current and Past Rice Team Members
Also, Peter J. Haas at IBM Almaden

This Talk Is About

- Programming environments/execution platforms for big ML

First, an Admission…

- I'm a database guy™

First, an Admission…

- I'm a database guy™
- What does that mean?

First, an Admission…

- I'm a database guy™
- What does that mean?
- To me, means that I worship at the church of **data independence**
  - — Now what in the heck does ***that*** mean?

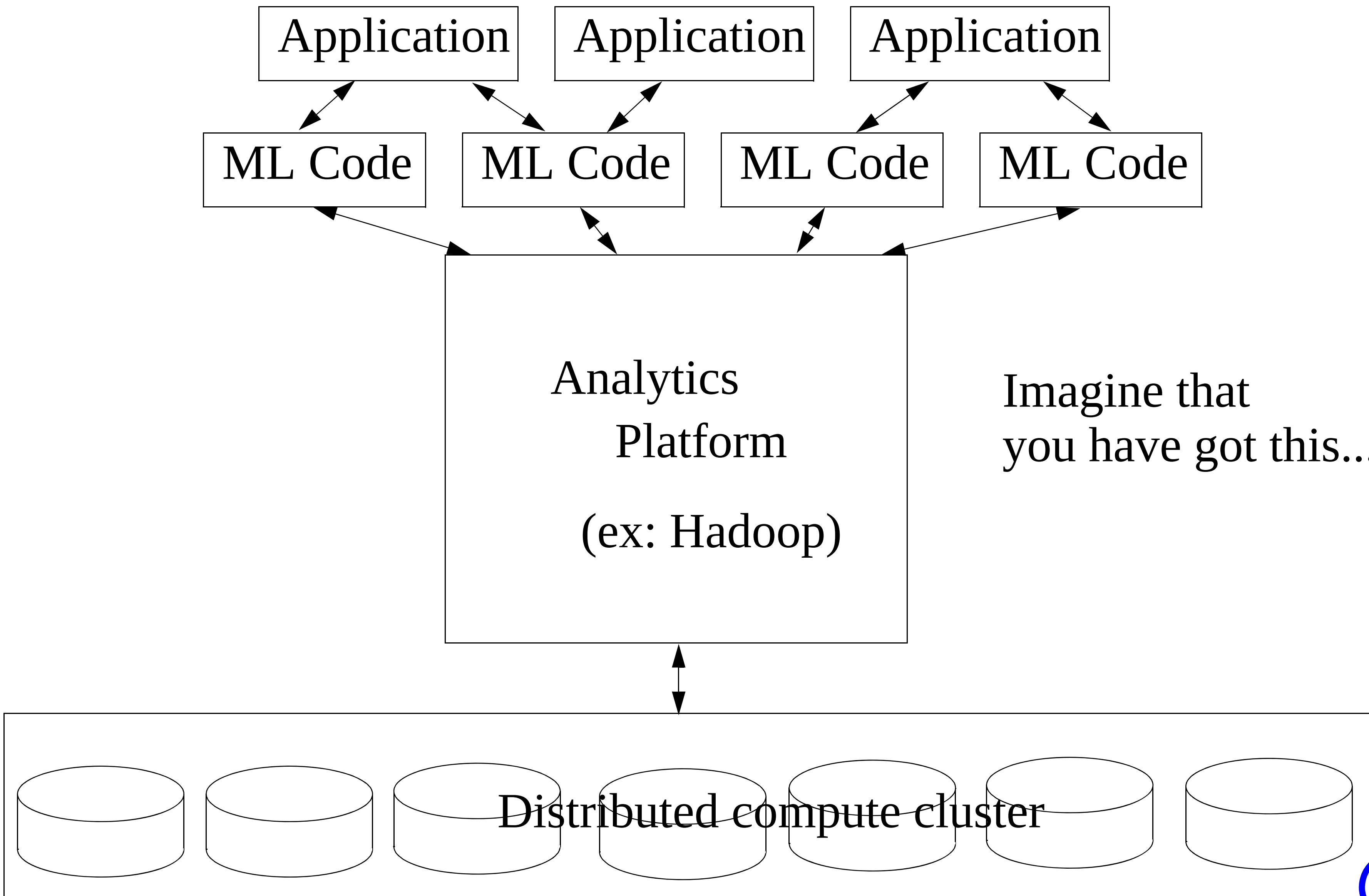
First, an Admission…

- I'm a database guy™
- What does that mean?
- To me, means that I worship at the church of **data independence**
    - — Now what in the heck does ***that*** mean?
- Means that when one designs a data-processing system...
- It should strive for the following ideal:
    - — Coder specifies **what** the computation result should be, not **how** to get there
    - — System itself figures out the **how** (the "declarative" paradigm)
    - — Means code can be independent of data format, size, schema, processing hardware
    - — Same code runs on one box with a GPU and on a 1000-machine cluster

Why Are Declarative and Data Ind. Good?

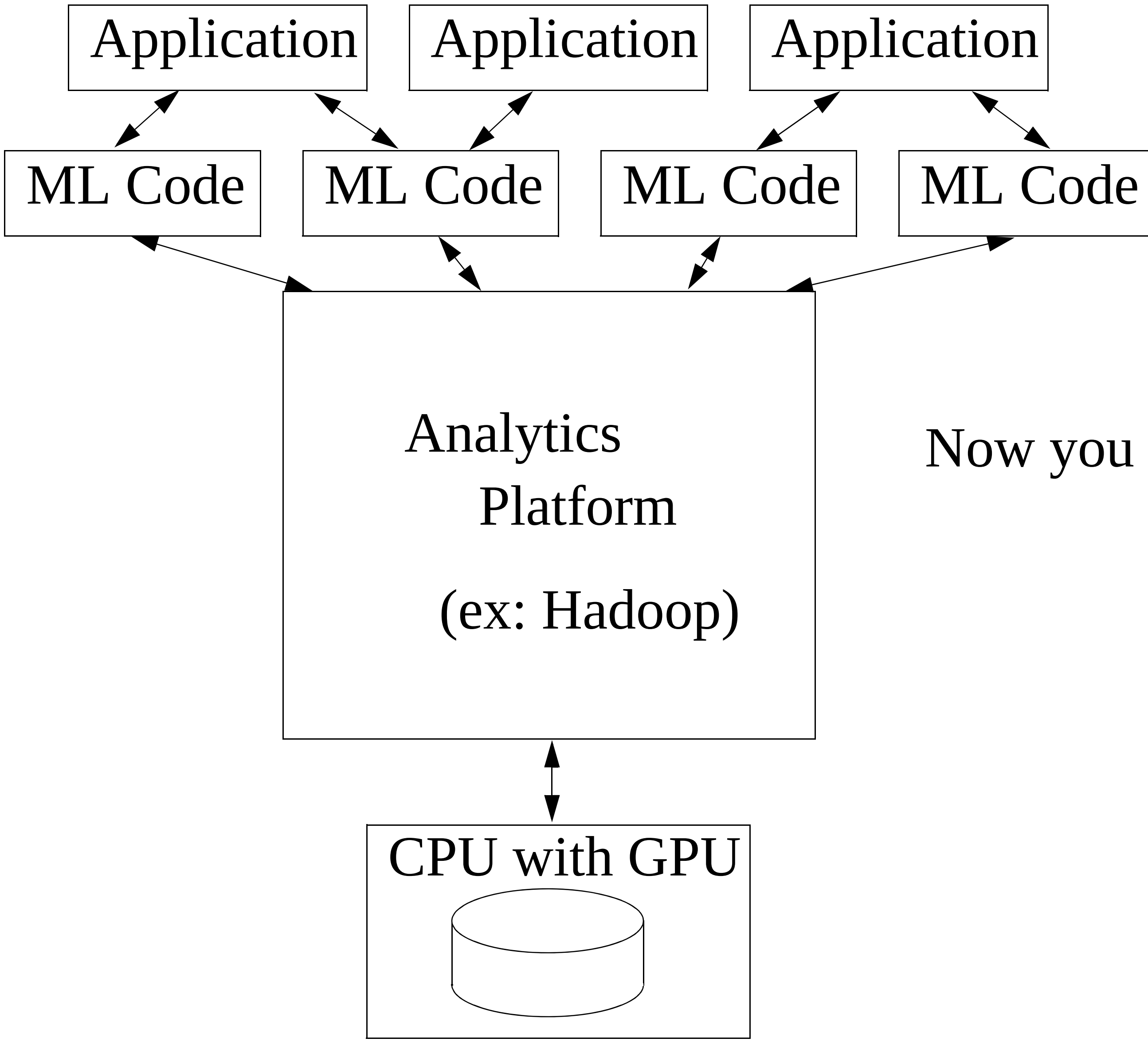
Declarative Arg 1: One Code, Many Backends

Declarative Arg 1: One Code, Many Backends



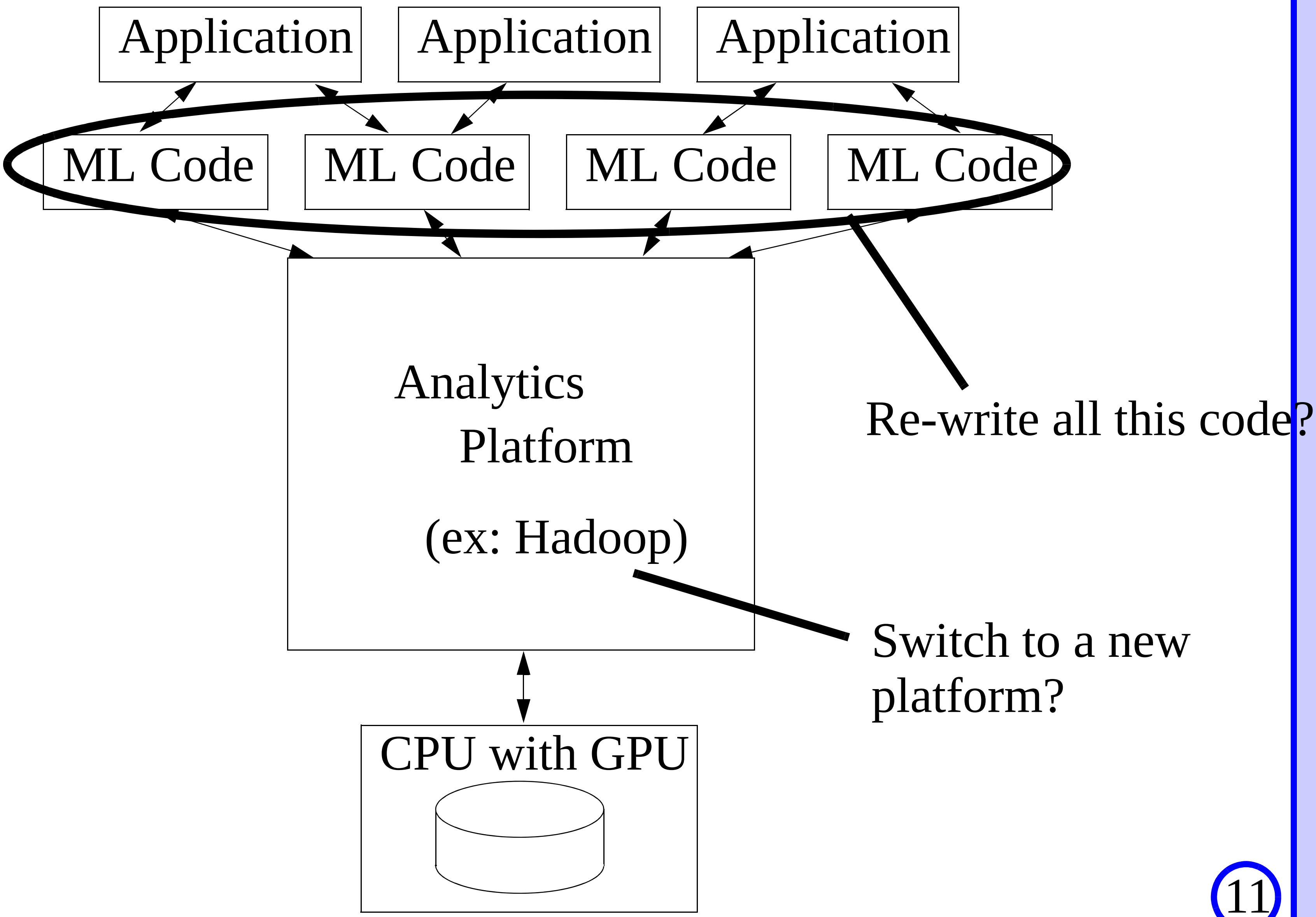
Imagine that
you have got this...

Declarative Arg 1: One Code, Many Backends

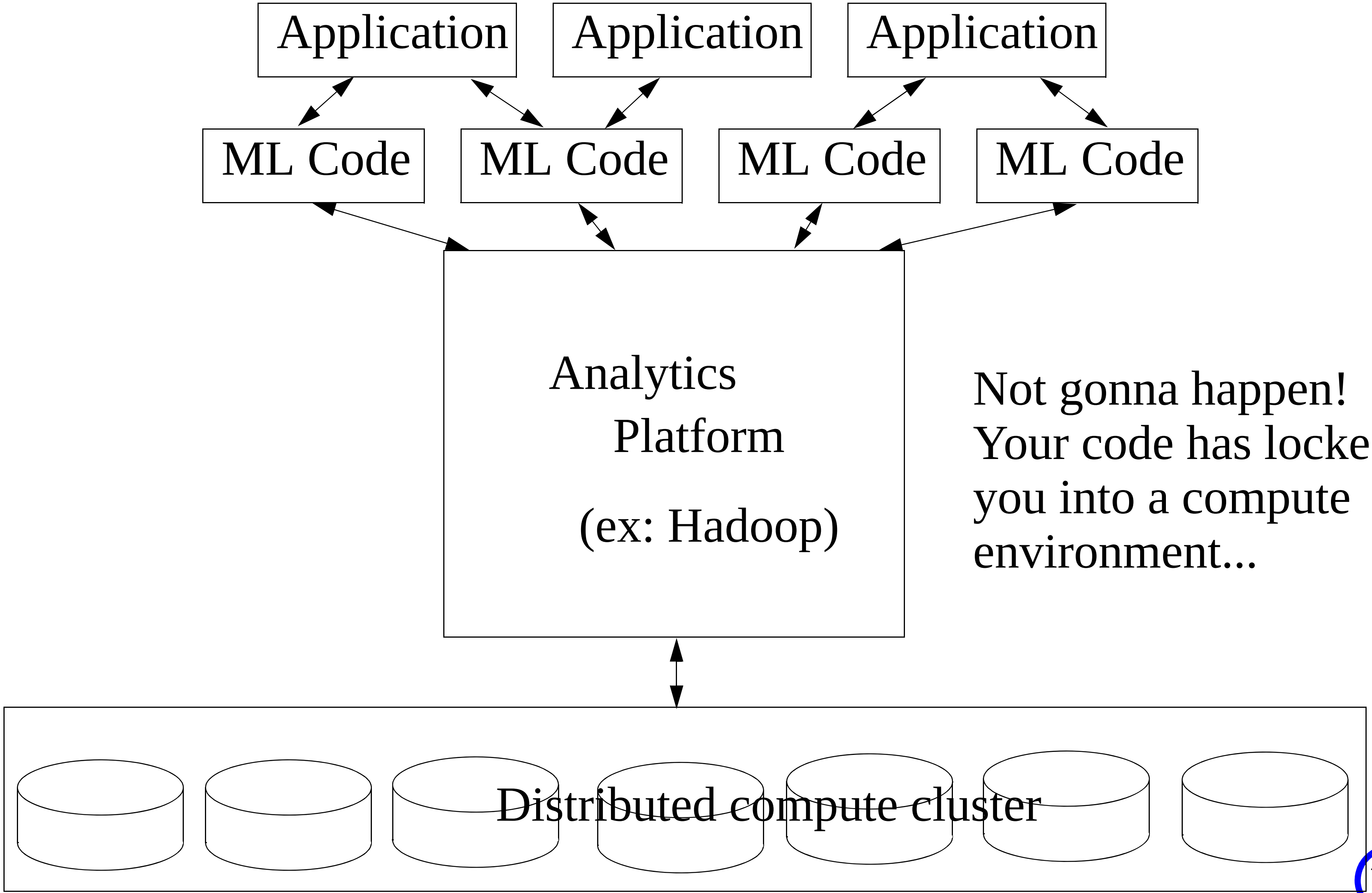


Now you want this...

Declarative Arg 1: One Code, Many Backends

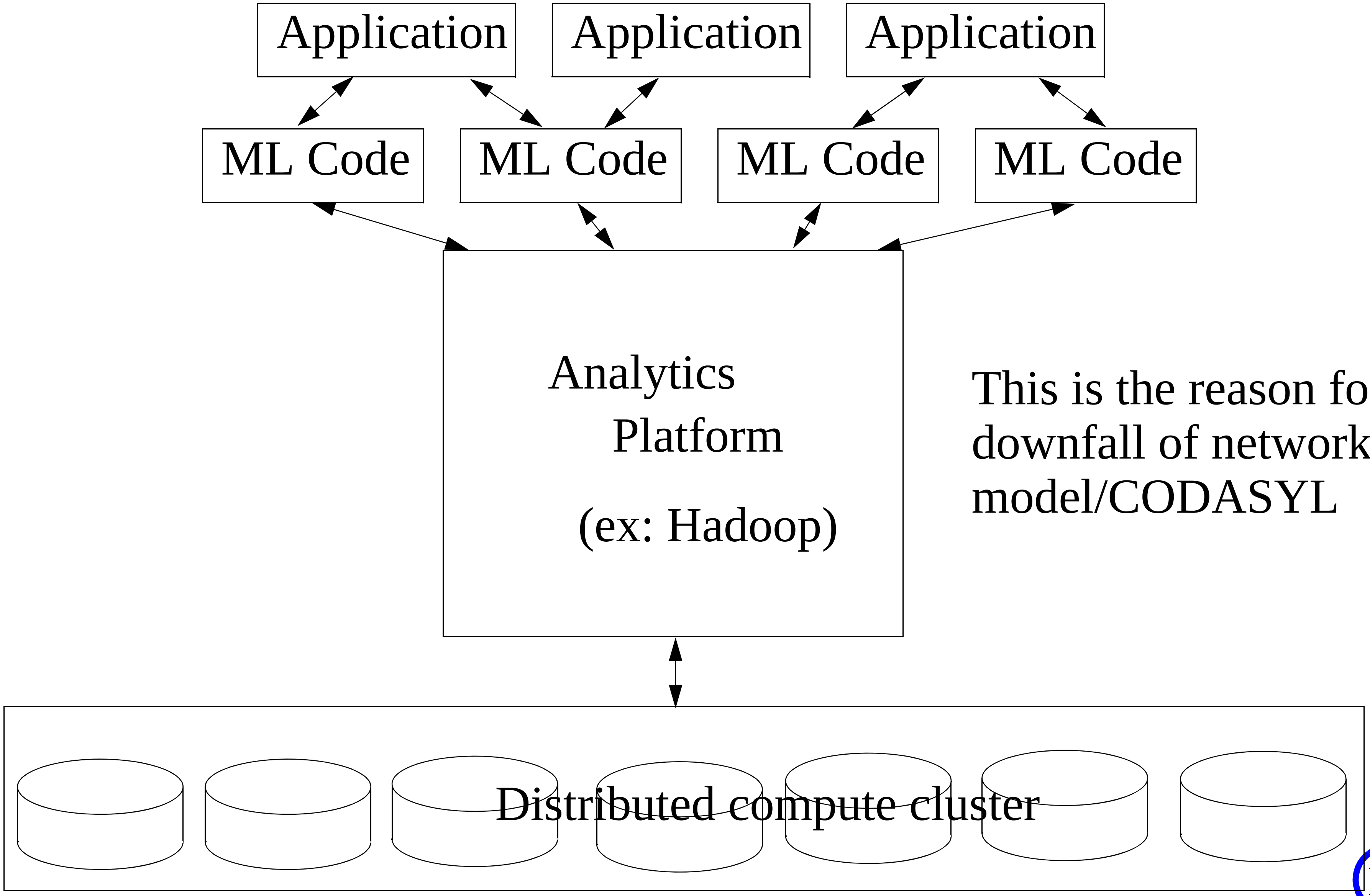


Declarative Arg 1: One Code, Many Backends



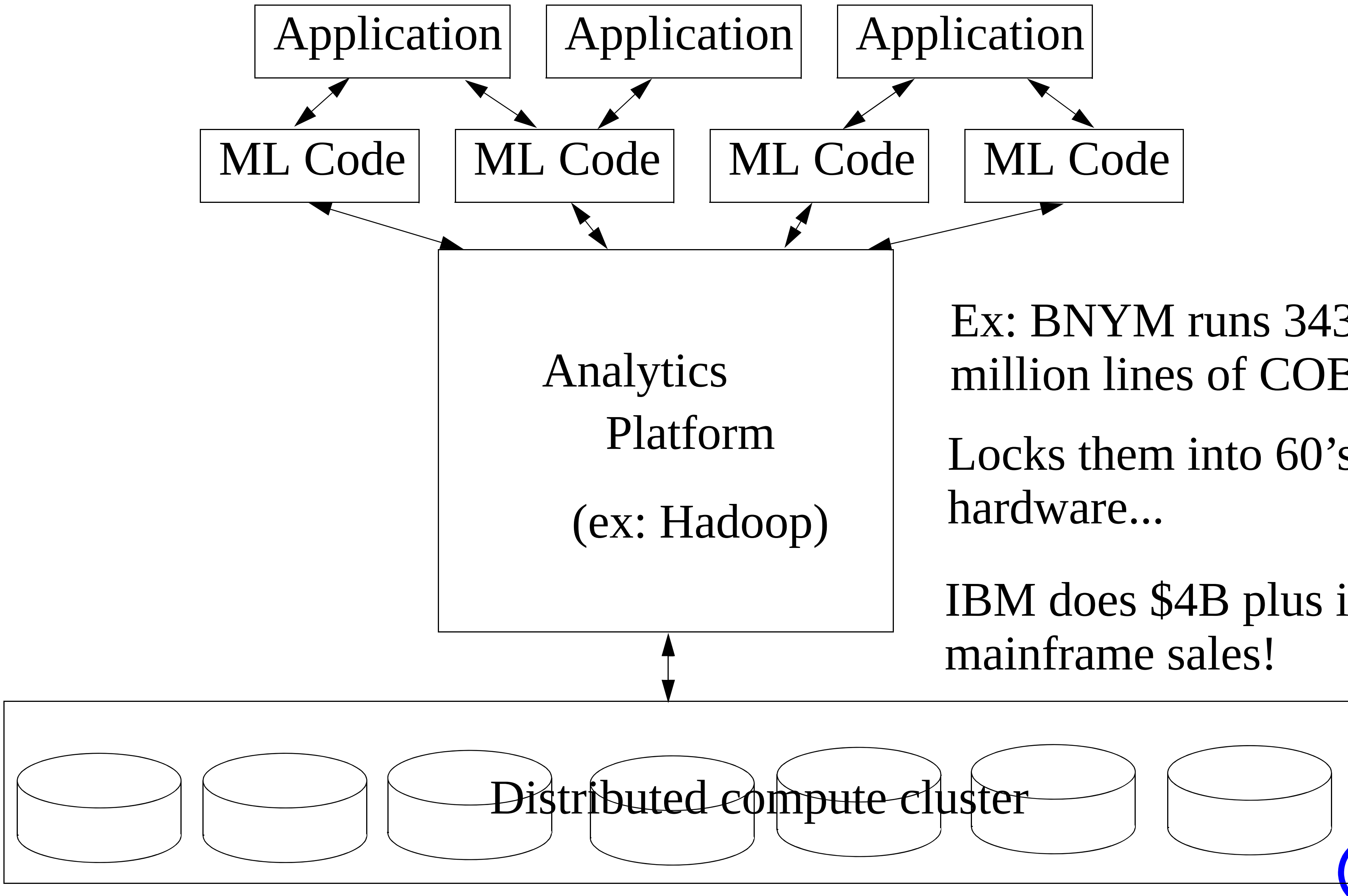
Not gonna happen! Your code has locked you into a compute environment...

Declarative Arg 1: One Code, Many Backends



This is the reason for downfall of network model/CODASYL

Declarative Arg 1: One Code, Many Backends



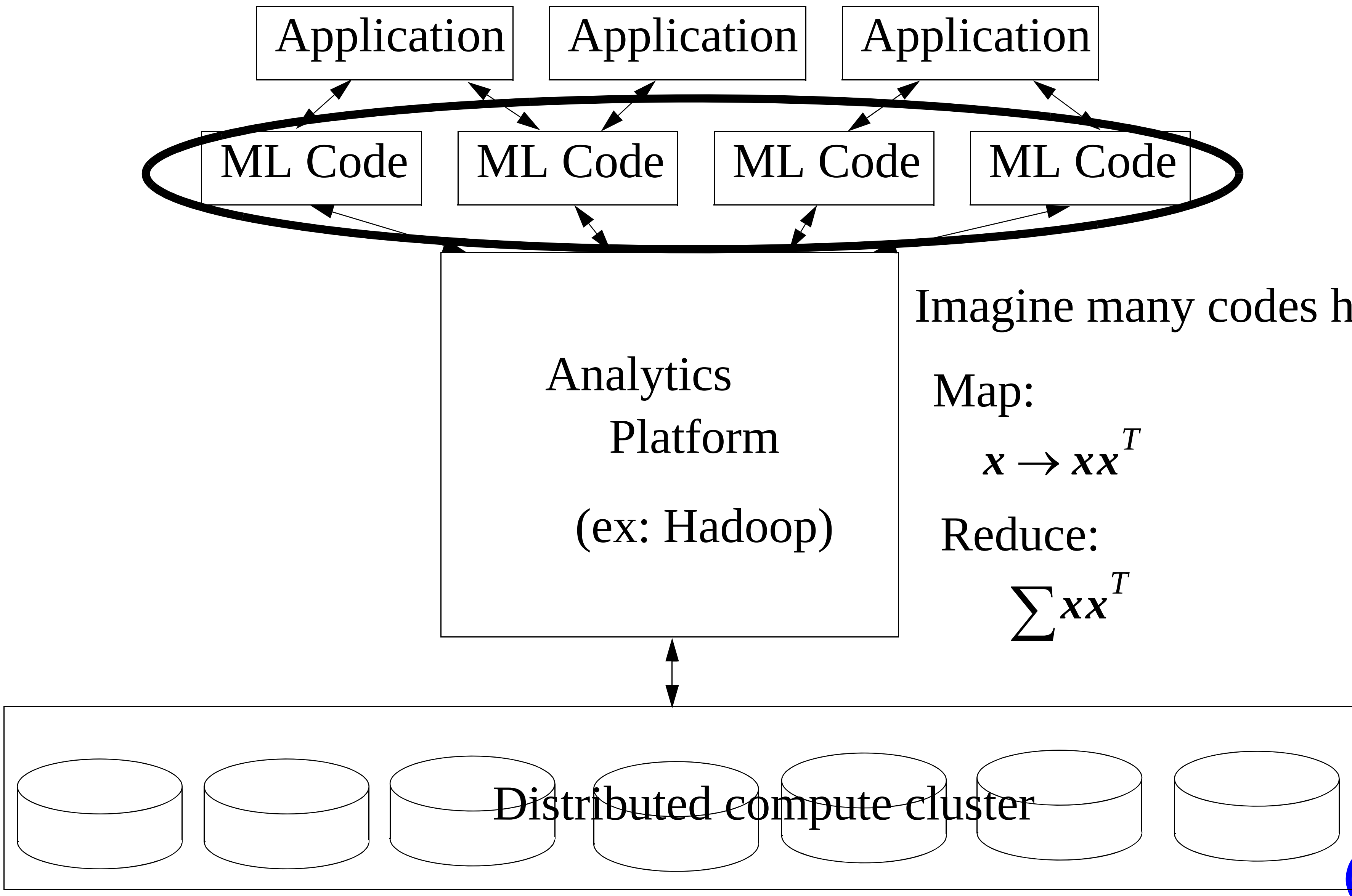
Ex: BNYM runs 343 million lines of COBOL

Locks them into 60's hardware...

IBM does $4B plus in mainframe sales!

Declarative Arg 2: Freedom From Algorithms

Declarative Arg 2: Freedom From Algorithms



Imagine many codes have:

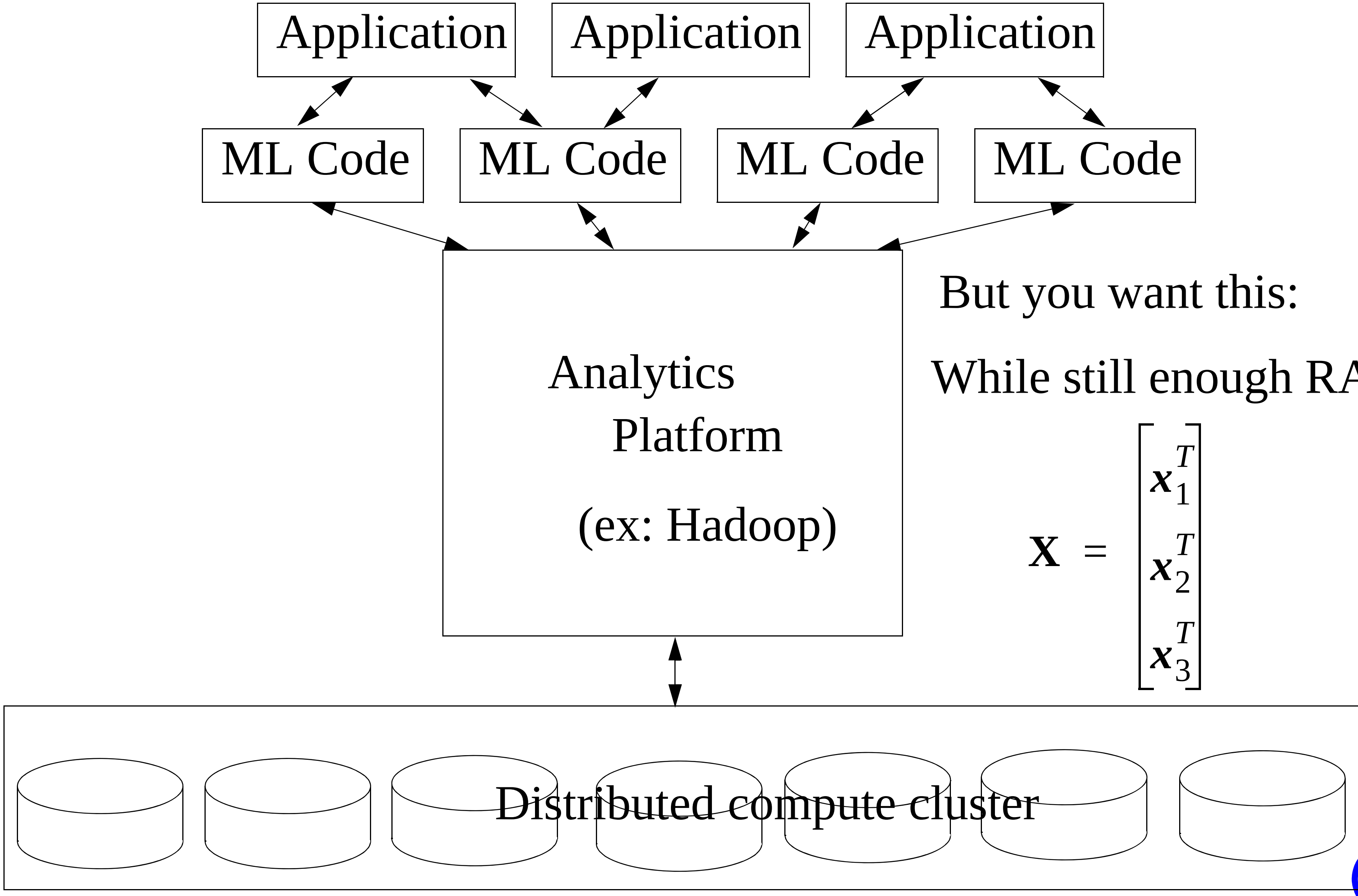
Map:

$$\boldsymbol{x} \rightarrow \boldsymbol{x}\boldsymbol{x}^T$$

Reduce:

$$\sum \boldsymbol{x}\boldsymbol{x}^T$$

Declarative Arg 2: Freedom From Algorithms

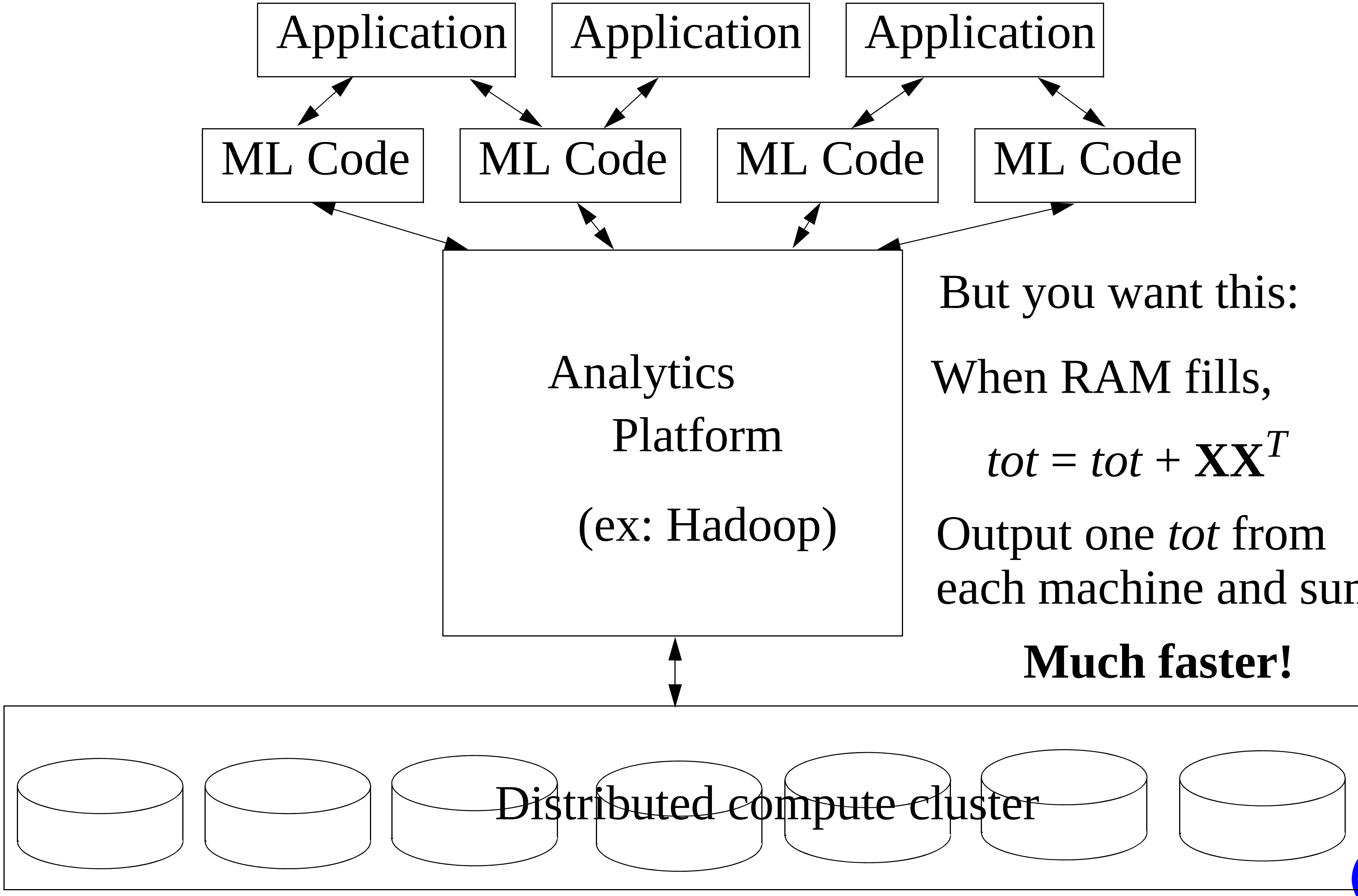


But you want this:

While still enough RAM:

$$\mathbf{X} = \begin{bmatrix} \boldsymbol{x}_1^T \\ \boldsymbol{x}_2^T \\ \boldsymbol{x}_3^T \end{bmatrix}$$

Declarative Arg 2: Freedom From Algorithms



But you want this:

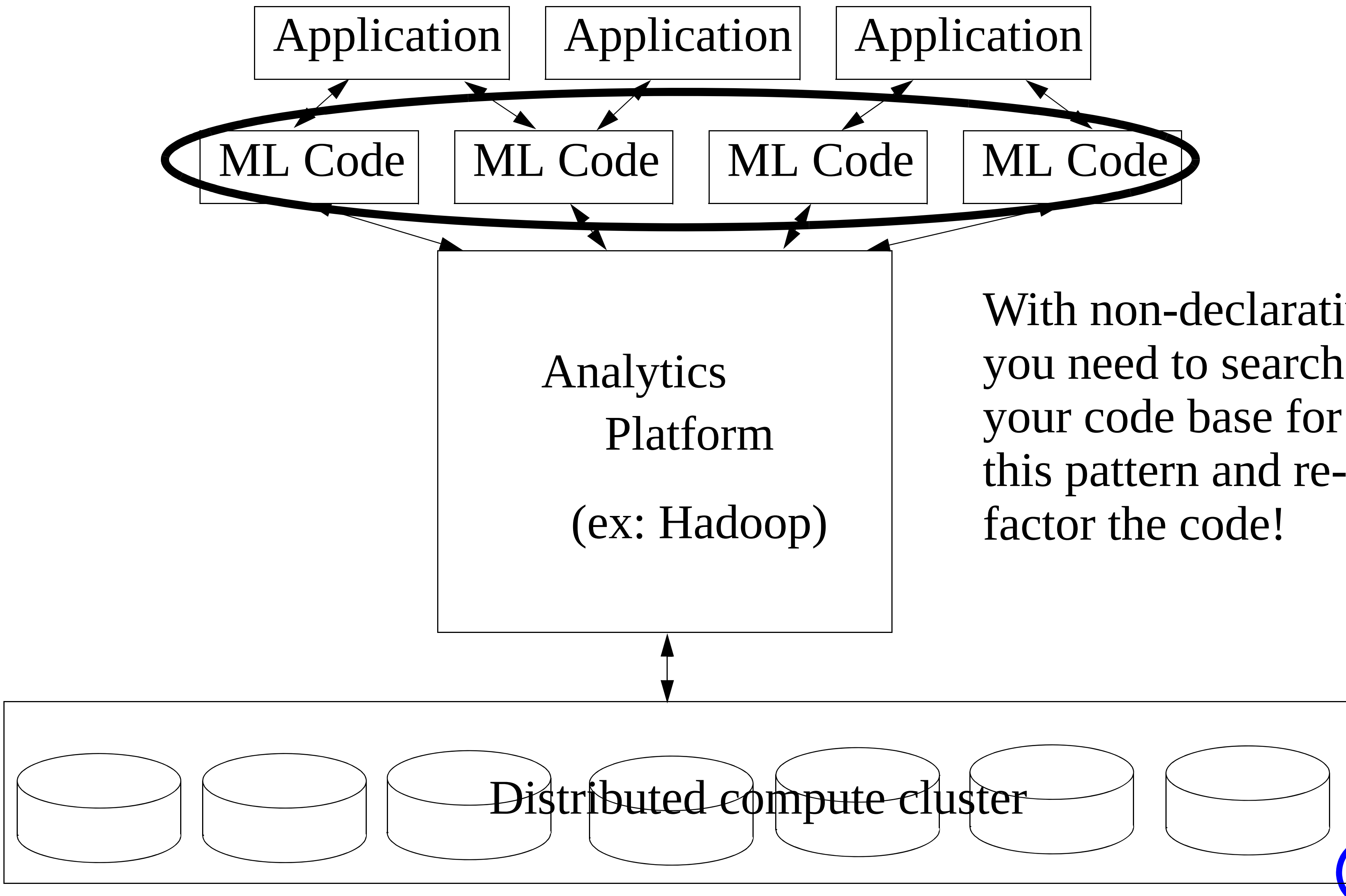
When RAM fills,

$$tot = tot + \mathbf{XX}^T$$

Output one *tot* from each machine and sum

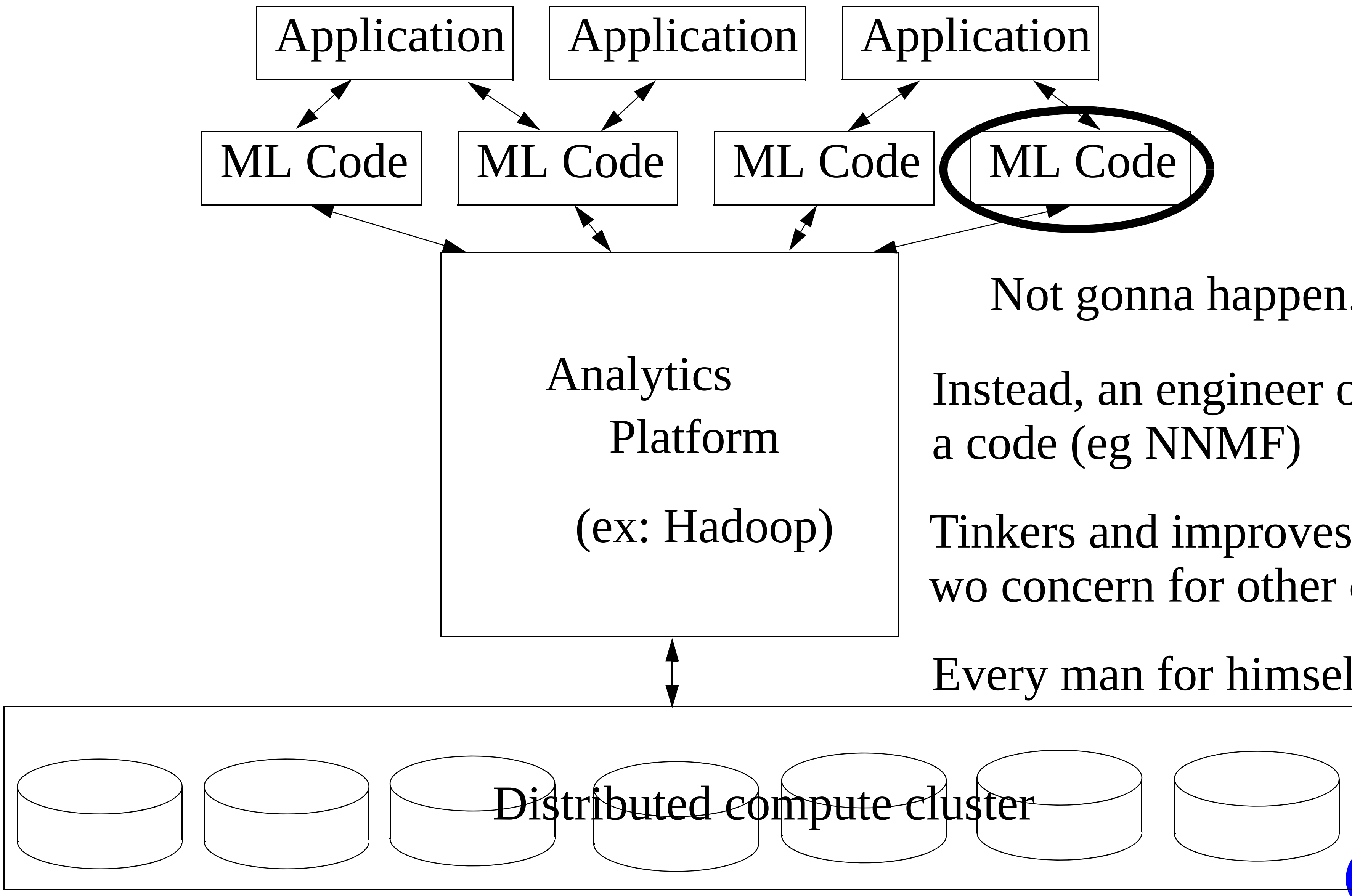
Much faster!

Declarative Arg 2: Freedom From Algorithms



With non-declarative you need to search your code base for this pattern and re-factor the code!

Declarative Arg 2: Freedom From Algorithms



Not gonna happen...

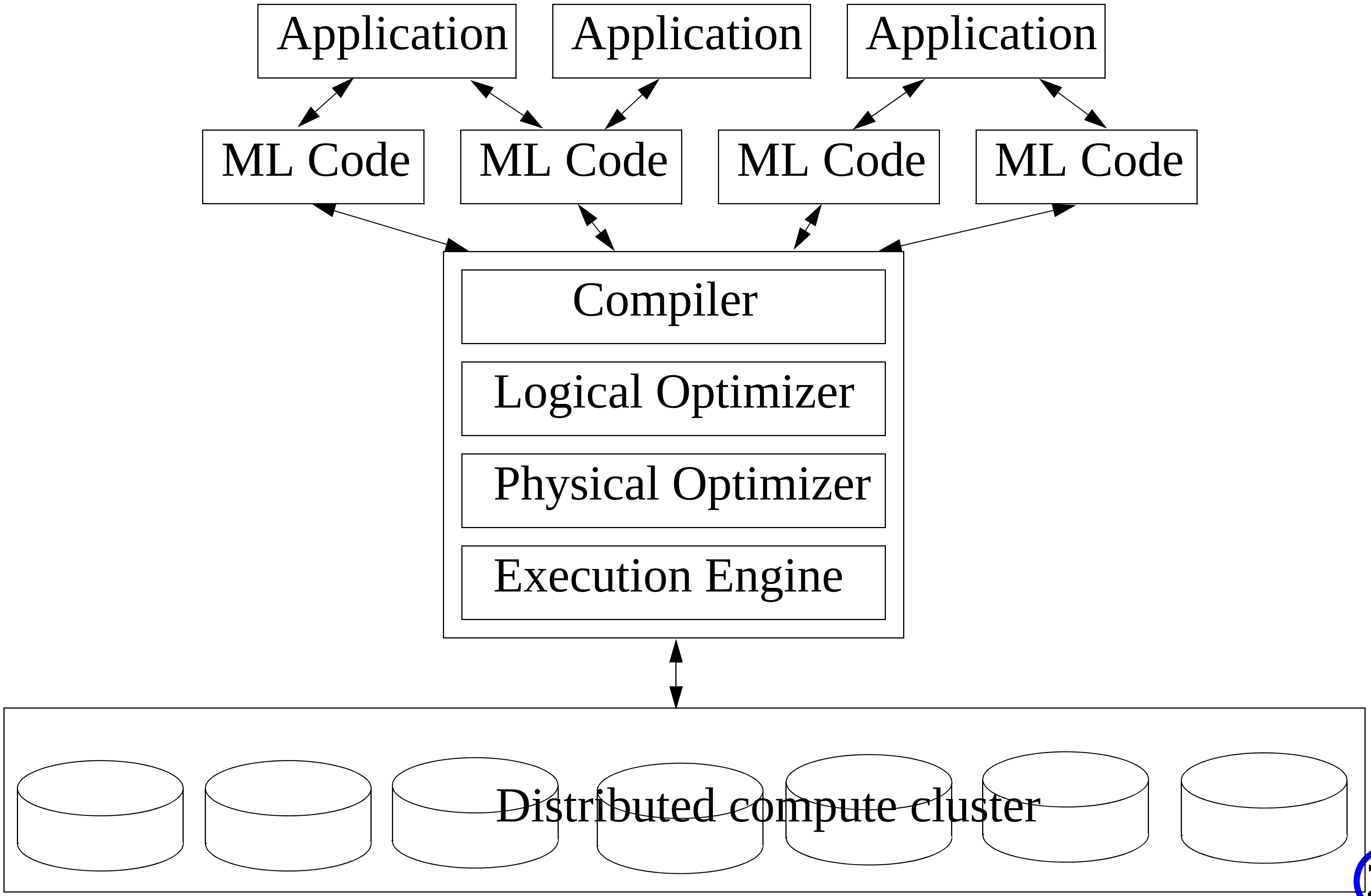
Instead, an engineer owns a code (eg NNMF)

Tinkers and improves it wo concern for other codes

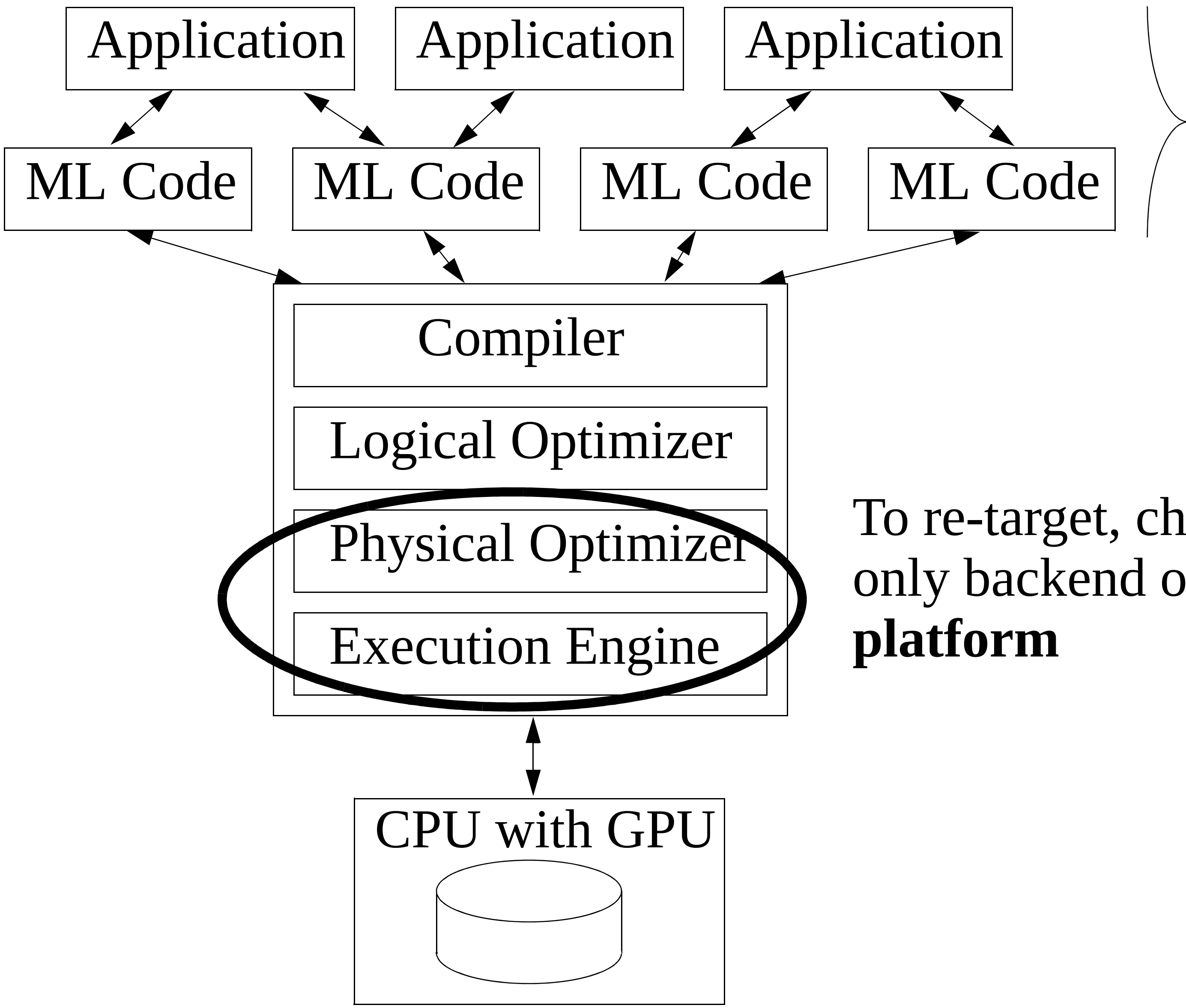
Every man for himself!

Much Better In System Based On Data Ind.

Much Better In System Based On Data Ind.



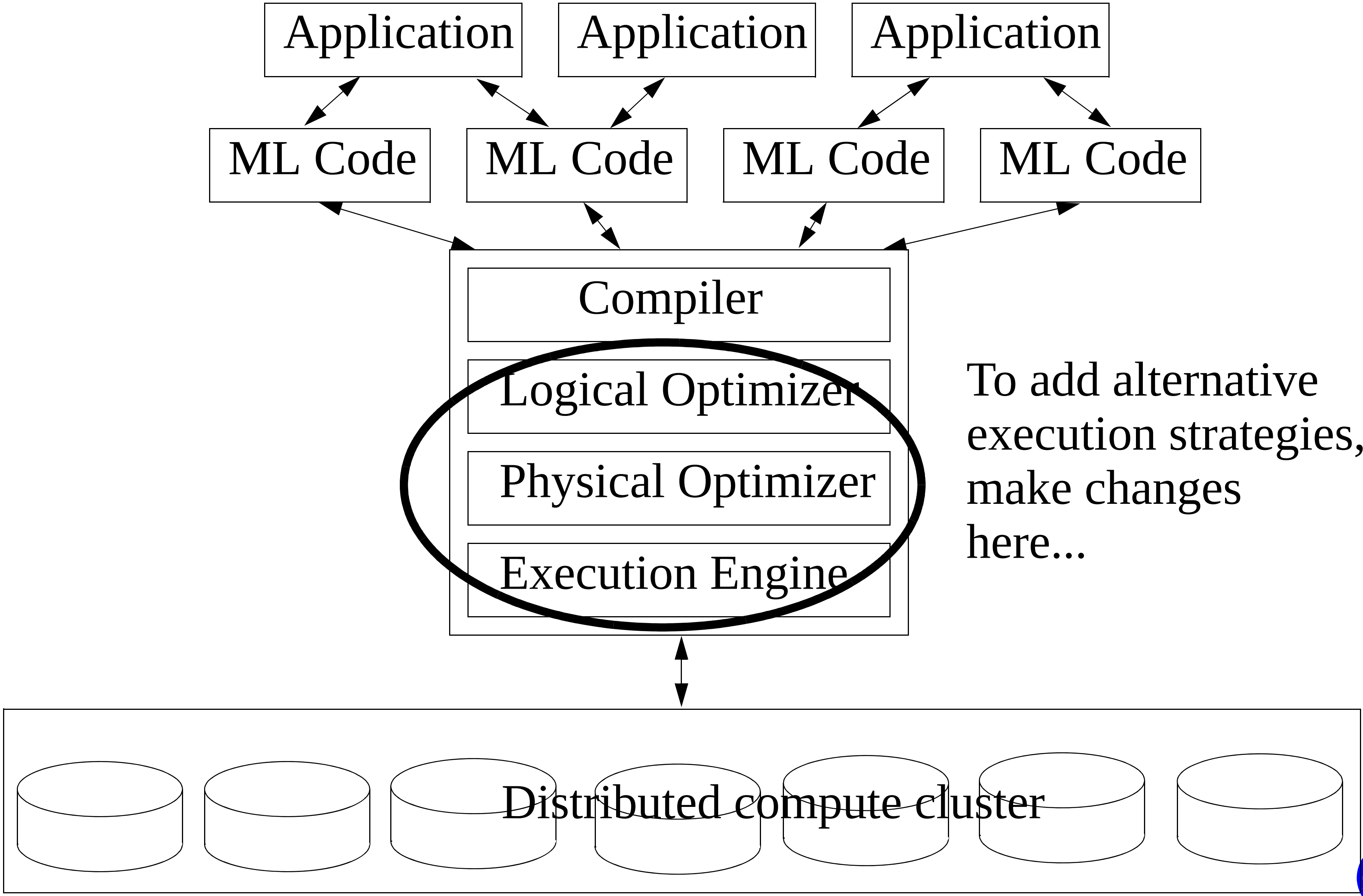
Much Better In System Based On Data Ind.



No Change Here!

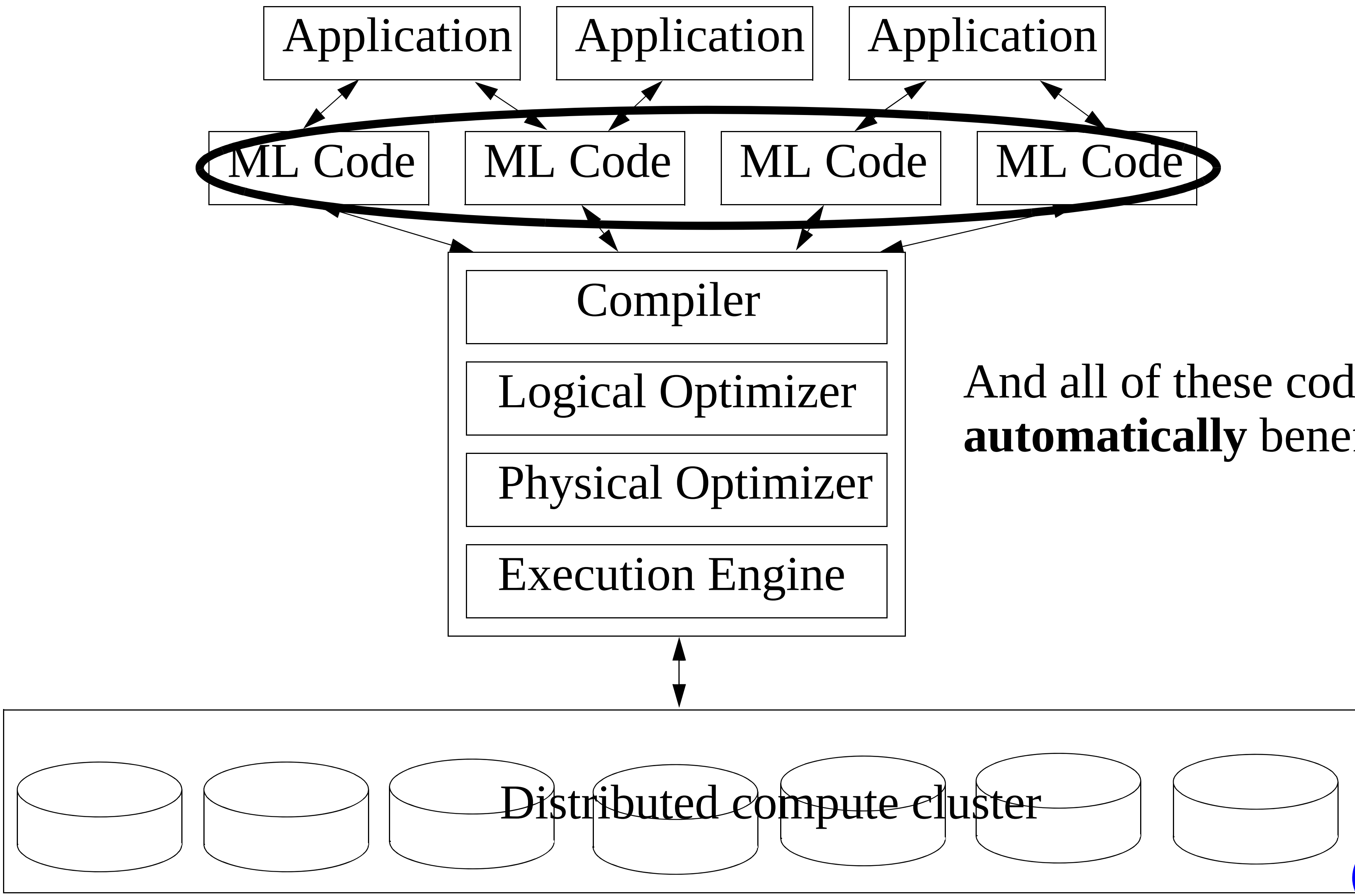
To re-target, change only backend of **platform**

Much Better In System Based On Data Ind.



To add alternative execution strategies, make changes here...

Much Better In System Based On Data Ind.



And all of these codes **automatically** benefit

Huge Win

- Once written, code at top of stack can remain unchanged
  - — #1 selling point of RDBMS tech for 30 years!

Huge Win

- Once written, code at top of stack can remain unchanged
  - — #1 selling point of RDBMS tech for 30 years!
- Unfortunately, not accepted by ML community
  - — ML people write great ML-on-GPU papers
  - — They design platforms such as GraphLab (higher-level, MPI-like framework)

Huge Win

- Once written, code at top of stack can remain unchanged
  - — #1 selling point of RDBMS tech for 30 years!
- Unfortunately, not accepted by ML community
  - — ML people write great ML-on-GPU papers
  - — They design platforms such as GraphLab (higher-level, MPI-like framework)
- Some in DB community have looked at declarative dataflow...
  - — Spark SQL on Spark
  - — Meteor on Stratosphere
  - — Asterix from UC Irvine

Huge Win

- Once written, code at top of stack can remain unchanged
  - — #1 selling point of RDBMS tech for 30 years!
- Unfortunately, not accepted by ML community
  - — ML people write great ML-on-GPU papers
  - — They design platforms such as GraphLab (higher-level, MPI-like framework)
- Some in DB community have looked at declarative dataflow...
  - — Spark SQL on Spark
  - — Meteor on Stratosphere
  - — Asterix from UC Irvine
- But this is far from declarative ML
  - — The codes don’t look anything like math!

The Goal

- Start with mathematical spec of learning algorithm...

1. $r \sim \text{Normal}(\mathbf{A}^{-1}\mathbf{X}^T\tilde{y}, \sigma^2\mathbf{A}^{-1})$

2. $\sigma^2 \sim \text{InvGamma}\left(\frac{(n-1)+p}{2}, \frac{(\tilde{y}-\mathbf{X}r)^T(\tilde{y}-\mathbf{X}r)^T + r^T\mathbf{D}^{-1}r}{2}\right)$

3. $\tau_j^{-2} \sim \text{InvGaussian}\left(\frac{\lambda\sigma}{r_j}, \lambda^2\right)$

— where $\mathbf{A} = \mathbf{X}^T\mathbf{X} + \mathbf{D}^{-1}$, $\mathbf{D}^{-1} = diag(\tau_1^{-2}, \tau_2^{-2}, \ldots)$

This is math for the Bayesian Lasso, lifted from original paper

— Bayesian regression model with regularizing prior on regression coefs

The Goal

- Programmer writes code that looks just like the math...

```
data {
  n: range (responses); p: range (regressors);
  X: array[n, p] of real; y: array[n] of real;
  lam: real
}

var {
  sig: real;
  r, t: array[p] of real; yy, Z: array[n] of real;
}

A <- inv(X ‘* X + diag(t));
yy <- (y[i] - mean(y) | i in 1:n);
Z <- yy - X * r;

init {
  sig ~ InvGamma (1, 1);
  t ~ (InvGauss (1, lam) | j in 1:p);
}

r ~ Normal (A *’ X * yy, sig * A);
sig ~ InvGamma(((n-1) + p)/2,
   (Z ‘* Z + (r * diag(t) ‘* r)) / 2);
for (j in 1:p) {
  t[j] ~ InvGauss (sqrt((lam * sig) / r[j]), lam);
}
```

We call our language “BUDS”

The Goal

- Write code that looks just like the math...

```
data {
  n: range (responses); p: range (regressors);
  X: array[n, p] of real; y: array[n] of real;
  lam: real
}

var {
  sig: real;
  r, t: array[p] of real; yy, Z: array[n] of real;
}

A <- inv(X ‘* X + diag(t));
yy <- (y[i] - mean(y) | i in 1:n);
Z <- yy - X * r;

init {
  sig ~ InvGamma (1, 1);
  t ~ (InvGauss (1, lam) | j in 1:p);
}

r ~ Normal (A *’ X * yy, sig * A);
sig ~ InvGamma(((n-1) + p)/2,
   (Z ‘* Z + (r * diag(t) ‘* r)) / 2);
for (j in 1:p) {
  t[j] ~ InvGauss (sqrt((lam * sig) / r[j]), lam);
}
```

$$\mathbf{A} = \mathbf{X}^T\mathbf{X} + \mathbf{D}^{-1}$$

$$r \sim \mathrm{Normal}(\mathbf{A}^{-1}\mathbf{X}^T\tilde{y}, \sigma^2\mathbf{A}^{-1})$$

$$\sigma^2 \sim \mathrm{InvGamma}\left(\frac{(n-1)+p}{2}, \frac{(\tilde{y}-\mathbf{X}r)^T(\tilde{y}-\mathbf{X}r)^T + r^T\mathbf{D}^{-1}r}{2}\right)$$

$$\tau_j^{-2} \sim \mathrm{InvGaussian}\left(\frac{\lambda\sigma}{r_j}, \lambda^2\right)$$

The Goal

- And the system compiles and executes this for a huge data set
  - — On hundreds or thousands of machines...
  - — Or on a desktop with a GPU...
  - — Or for whatever backend the system can target...

Also Important

- We don’t want to be like everyone and argue for a new DA stack
  - — The world has too many dataflow platforms already

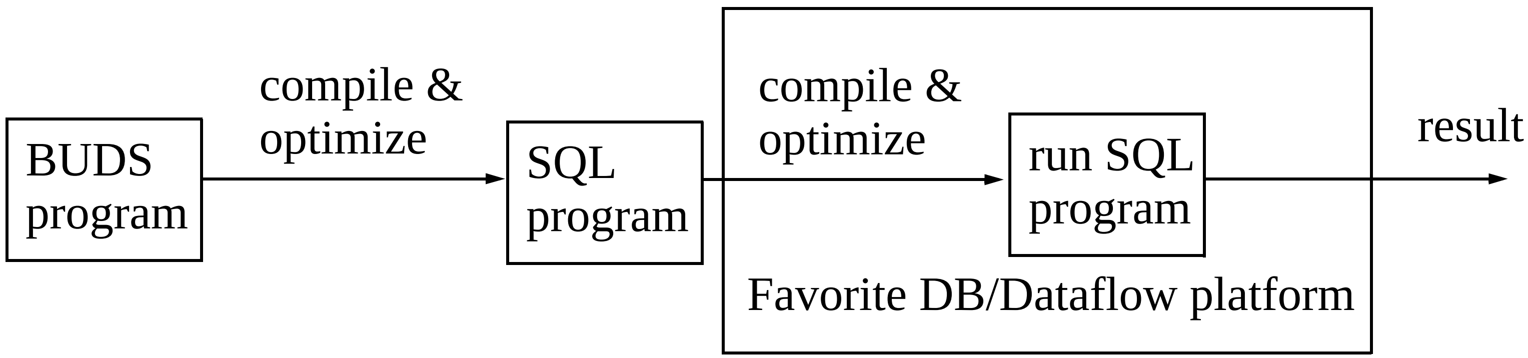
The Fact Is

- SQL or SQL-like declarative language is the LCD for all systems
  - Including commercial databases
  - Free databases
  - And newfangled dataflow platforms

The Fact Is

- SQL or SQL-like declarative language is the LCD for all systems
  - — Including commercial databases
  - — Free databases
  - — And newfangled dataflow platforms

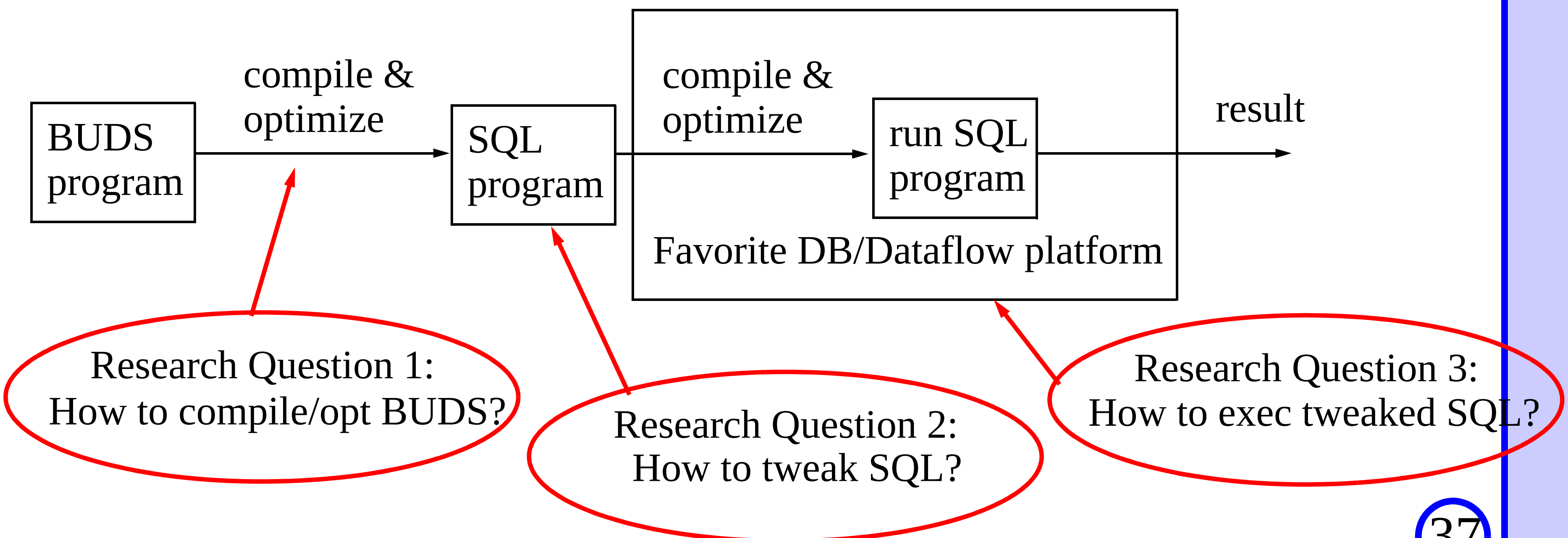
So here’s the workflow we envision...



The Fact Is

- SQL or SQL-like declarative language is the LCD for all systems
  - — Including commercial databases
  - — Free databases
  - — And newfangled dataflow platforms

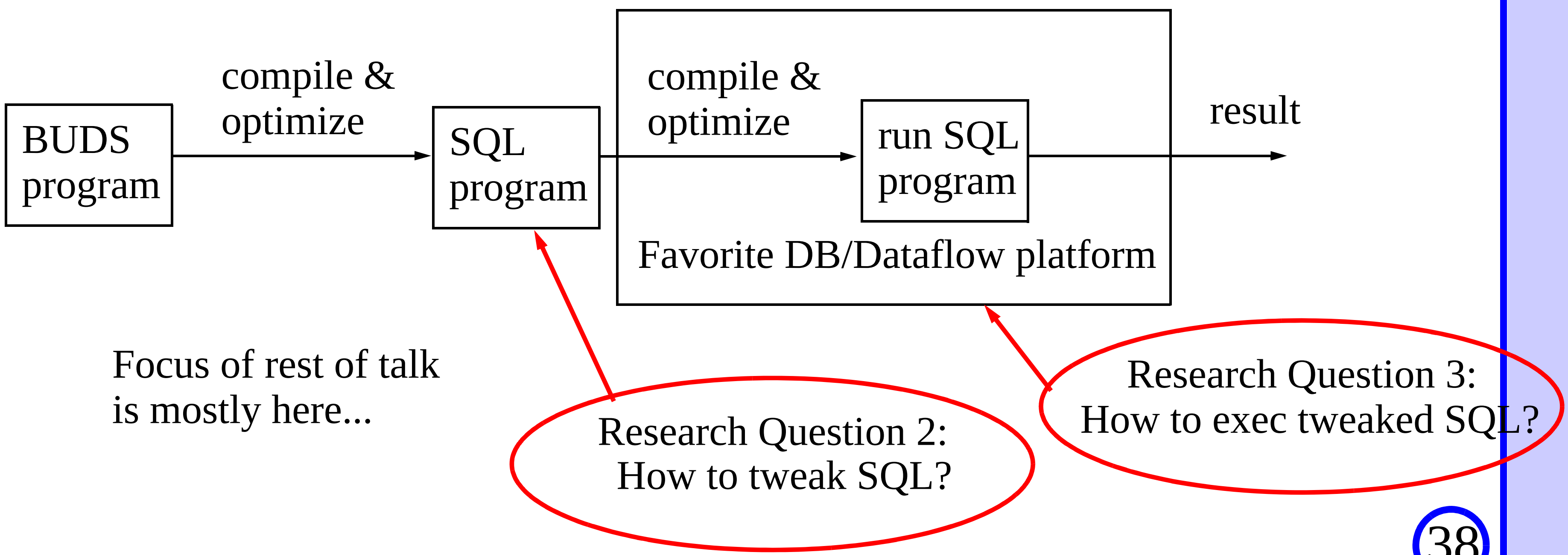
So here's the workflow we envision...



The Fact Is

- SQL or SQL-like declarative language is the LCD for all systems
  - Including commercial databases
  - Free databases
  - And newfangled dataflow platforms

So here's the workflow we envision...



So, How Must SQL/DA Platform Change?

- More extensive support for recursion
- Fancier table functions ("VG functions")
- Add native support for vectors/matrices (as att types)
- Support for executing huge "query" plans (1000's of operations)
- A few new logical/physical operators

So, How Must SQL/DA Platform Change?

- More extensive support for recursion
- Fancier table functions (“VG functions”)
- Add native support for vectors/matrices (as att types)
- Support for executing huge “query” plans (1000’s of operations)
- A few new logical/physical operators

- We built the SimSQL system to demo how this works
  - — Simple shared nothing, parallel DBMS
  - — 100K SLOC
  - — Java, C++, Prolog
  - — Runs queries as Map-only jobs on Hadoop

So, How Must SQL/DA Platform Change?

- More extensive support for recursion
- Fancier table functions (“VG functions”)
- Add native support for vectors/matrices (as att types)
- Support for executing huge “query” plans (1000’s of operations)
- A few new logical/physical operators

- We built the SimSQL system to demo how this works
  - — Simple shared nothing, parallel DBMS
  - — 100K SLOC
  - — Java, C++, Prolog
  - — Runs queries as Map-only jobs on Hadoop

SimSQL's Specialized for Stochastic Algs

- Due to my own Bayesian bias
  - — Though if you can do stochastic, you can do deterministic
- So I'll make a brief foray into MCMC...

MCMC

- Standard Bayesian ML inference method
- Idea is to simulate a Markov chain
- Whose *stationary distribution* is equal to the target posterior
    - — Means that if you run forever then stop, have sample from the target
    - — In theory, can be used with virtually any target distribution

MCMC: Gibbs Sampling

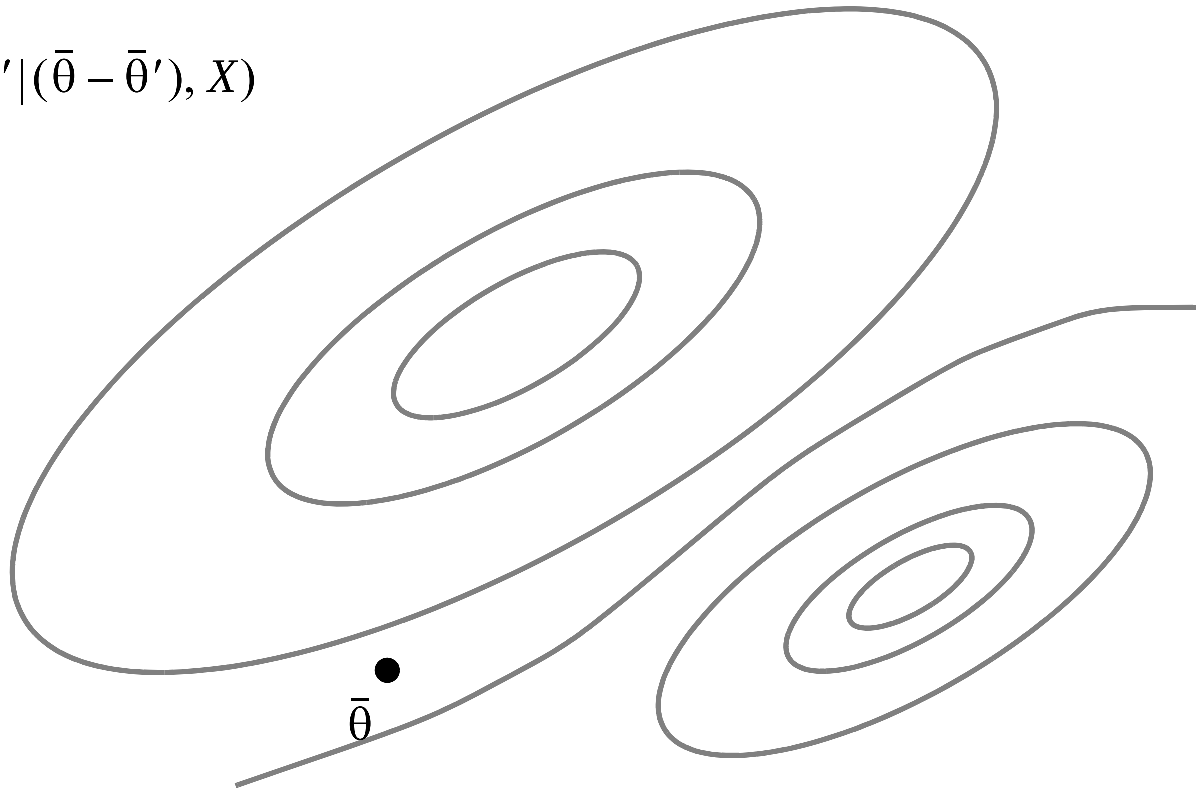
- Many MCMC algorithms; useful example is Gibbs sampling

— Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)

2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$

3. Repeat forever!

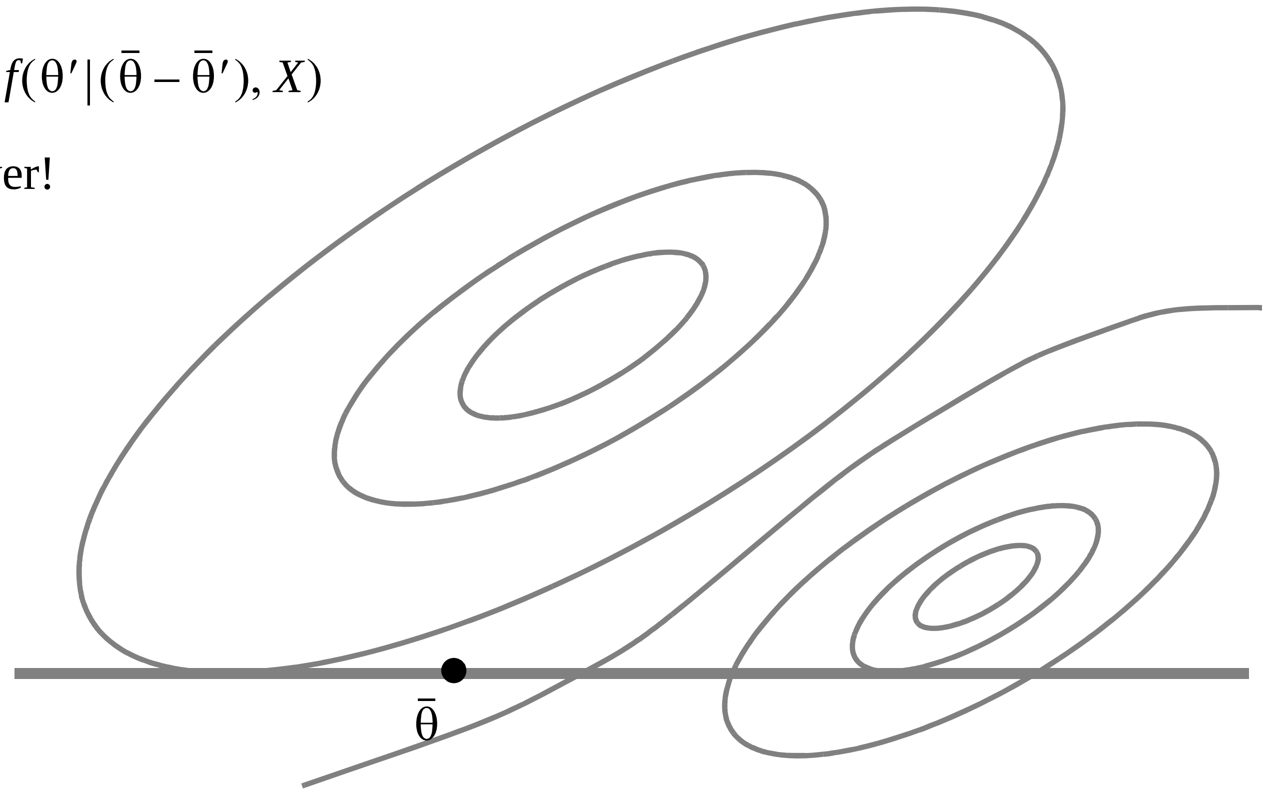


MCMC: Gibbs Sampling

- Many MCMC algorithms; useful example is Gibbs sampling

  — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

  1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)
  2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$
  3. Repeat forever!

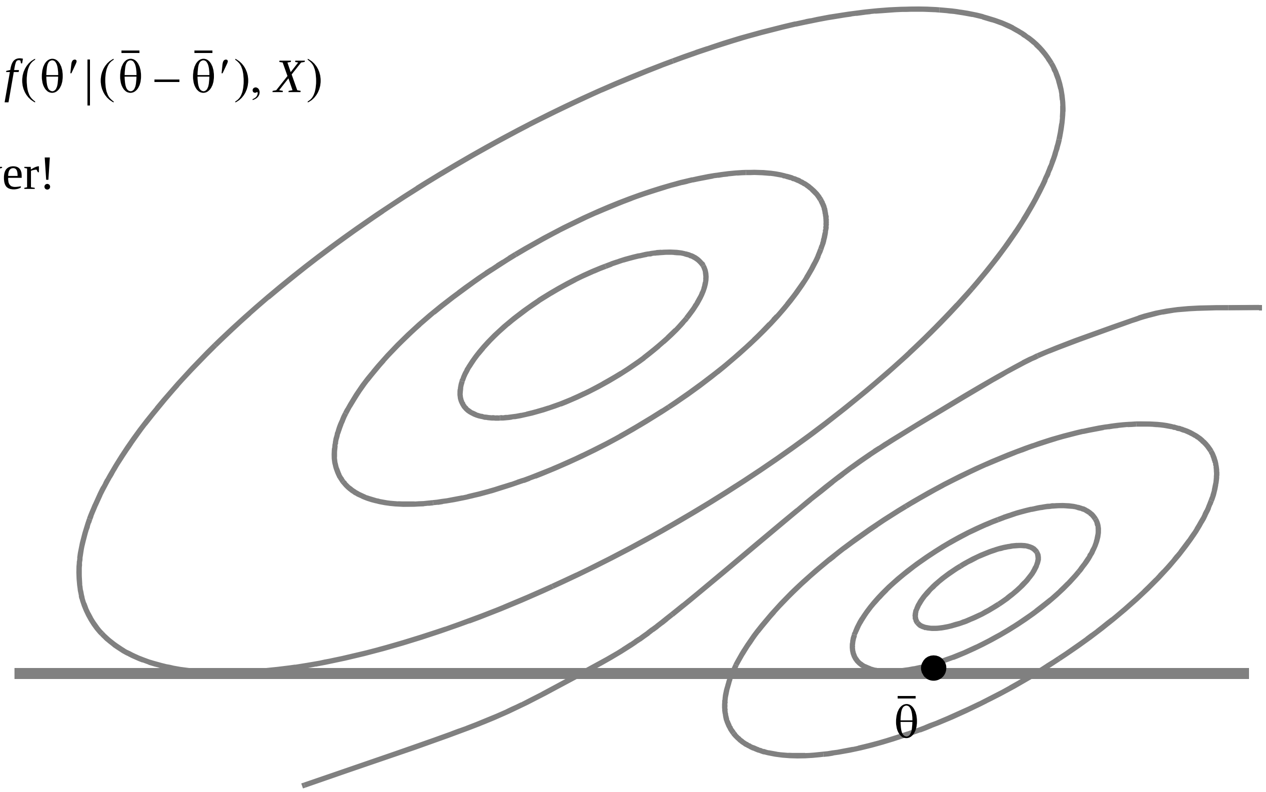


MCMC: Gibbs Sampling

- Many MCMC algorithms; useful example is Gibbs sampling

  — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

  1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)
  2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$
  3. Repeat forever!



MCMC: Gibbs Sampling

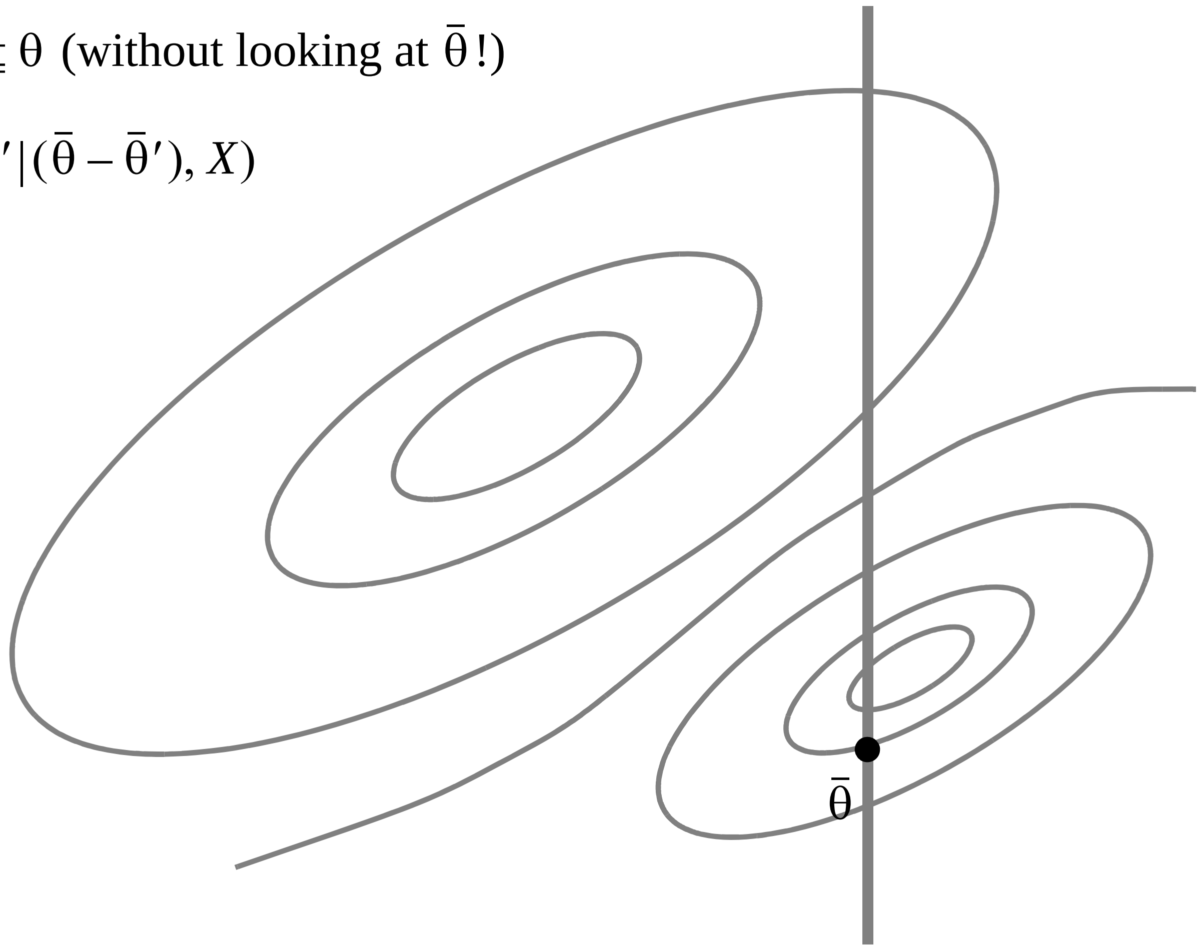
- Many MCMC algorithms; useful example is Gibbs sampling

  — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

  1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)

  2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$

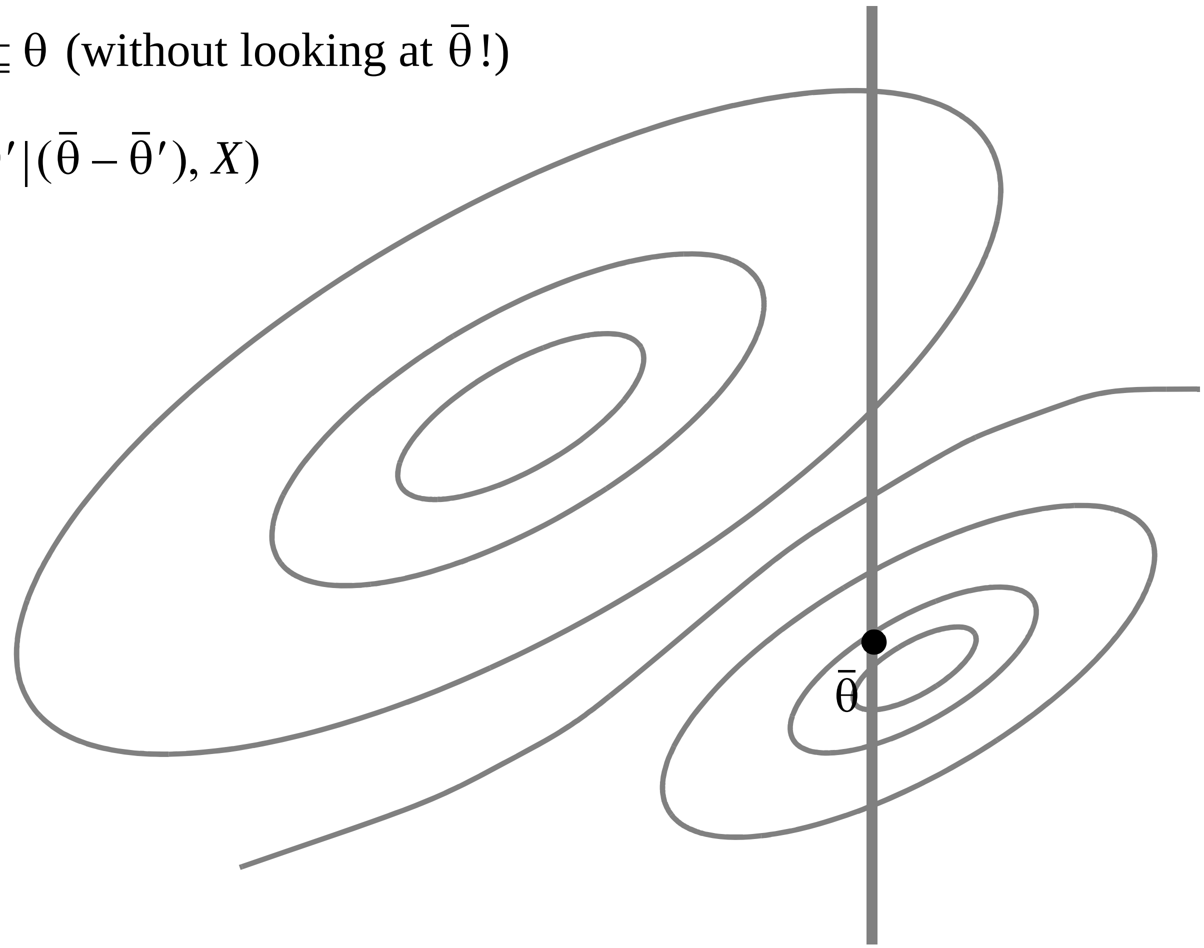
  3. Repeat forever!



MCMC: Gibbs Sampling

- Many MCMC algorithms; useful example is Gibbs sampling
  - — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

    1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)
    2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$
    3. Repeat forever!

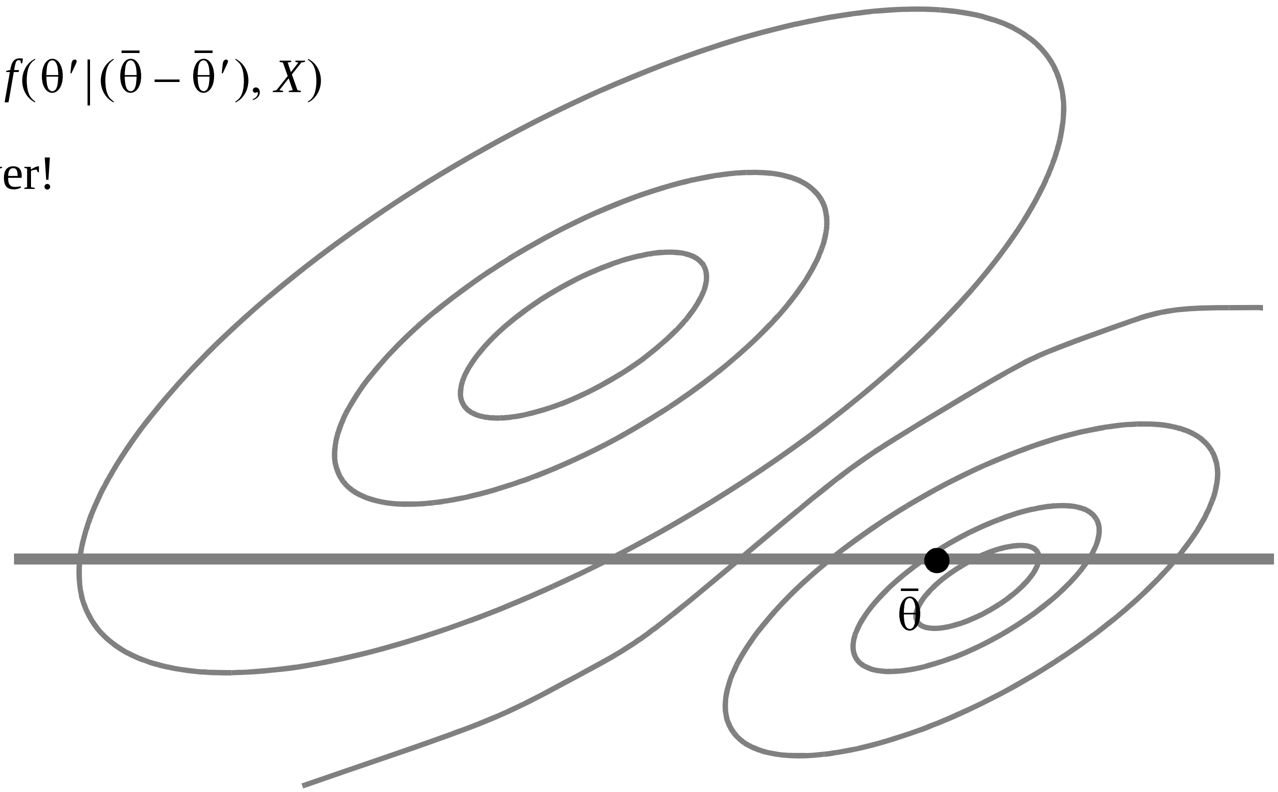


MCMC: Gibbs Sampling

- Many MCMC algorithms; useful example is Gibbs sampling

  — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

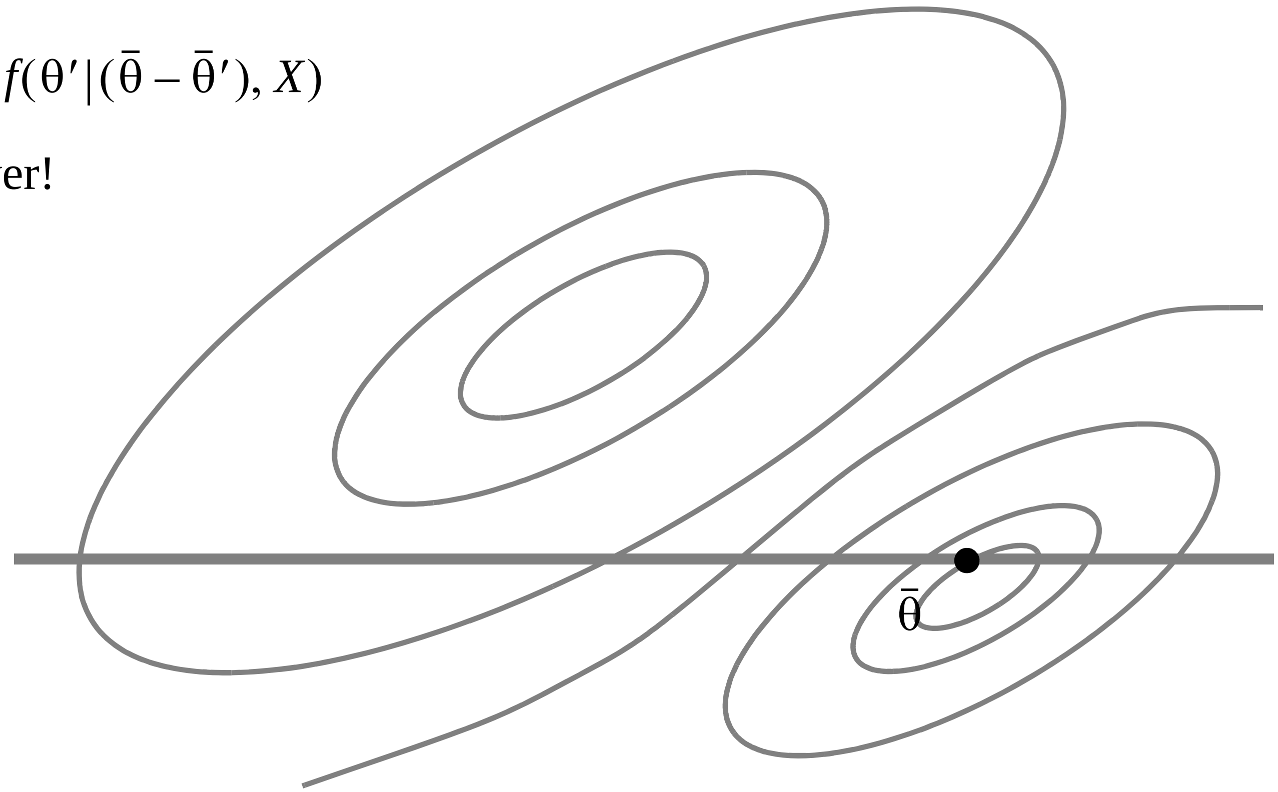
  1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)
  2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$
  3. Repeat forever!



MCMC: Gibbs Sampling

- Many MCMC algorithms; useful example is Gibbs sampling
  - — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

    1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)
    2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$
    3. Repeat forever!



MCMC: Gibbs Sampling

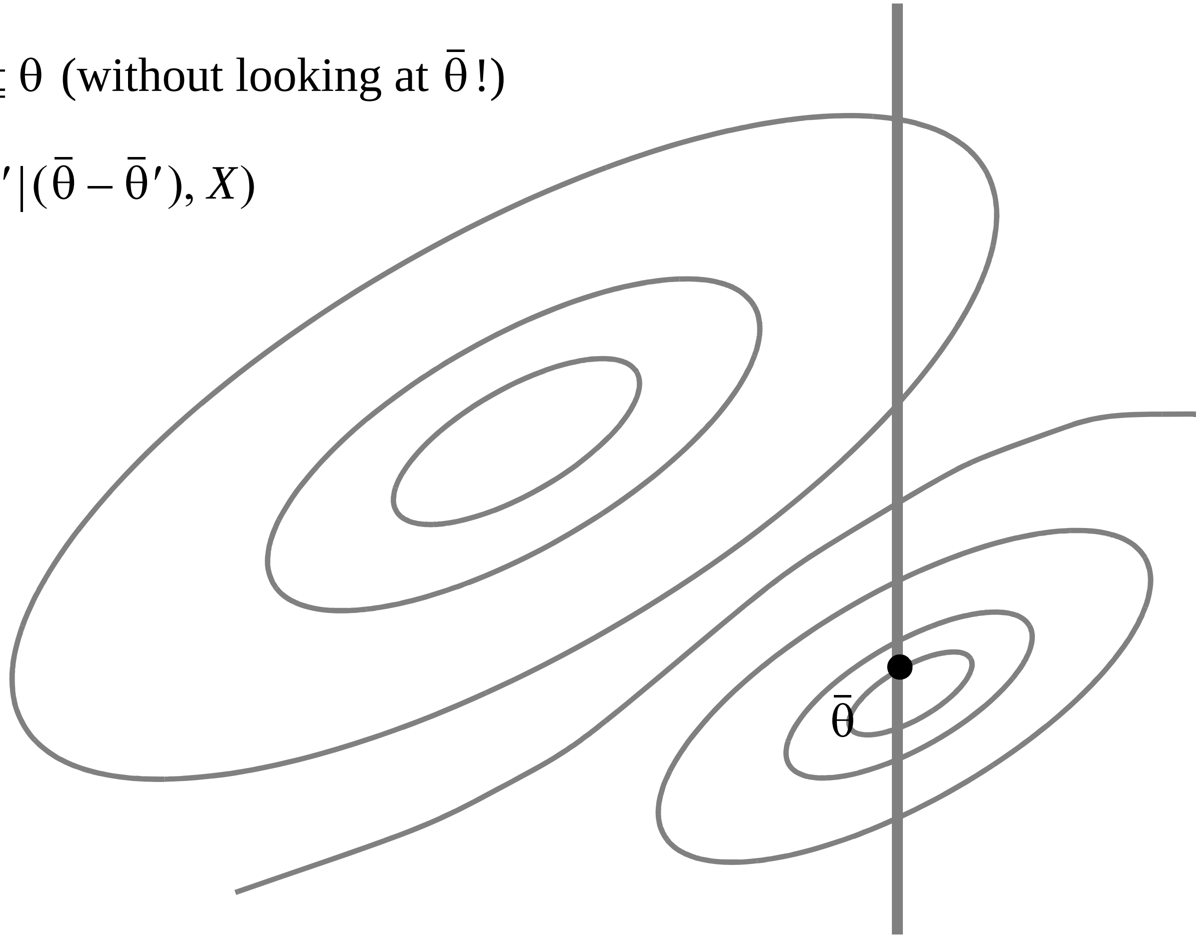
- Many MCMC algorithms; useful example is Gibbs sampling

  — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

  1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)

  2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$

  3. Repeat forever!

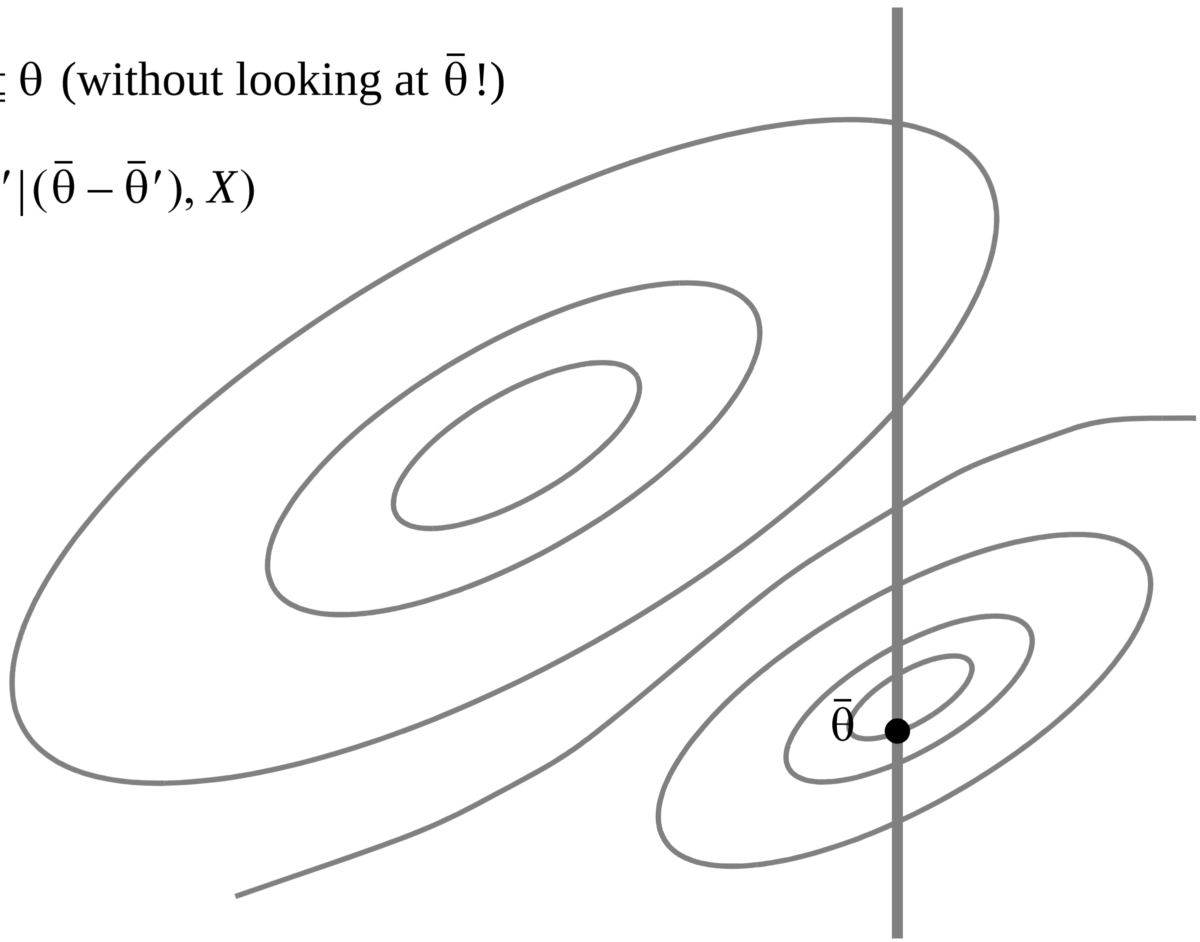


MCMC: Gibbs Sampling

- Many MCMC algorithms; useful example is Gibbs sampling

  — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

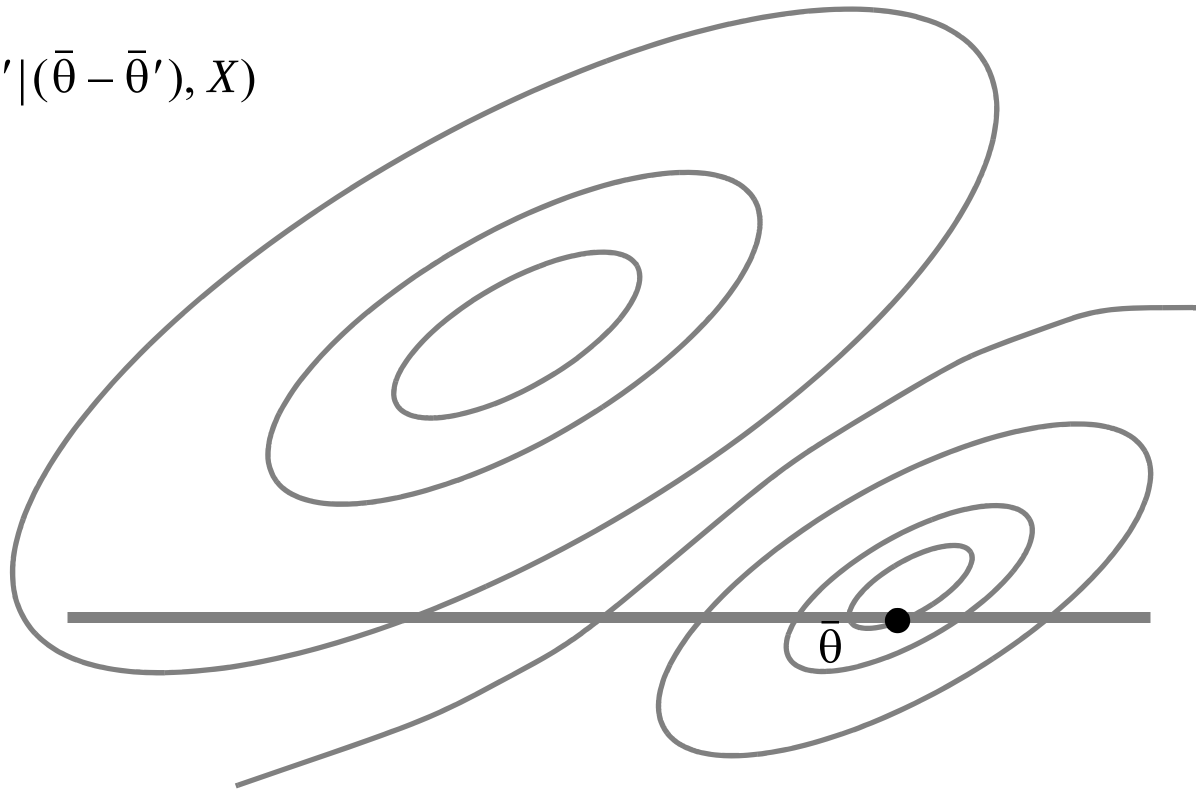
  1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)
  2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$
  3. Repeat forever!



MCMC: Gibbs Sampling

- Many MCMC algorithms; useful example is Gibbs sampling
  - Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

    1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)
    2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$
    3. Repeat forever!

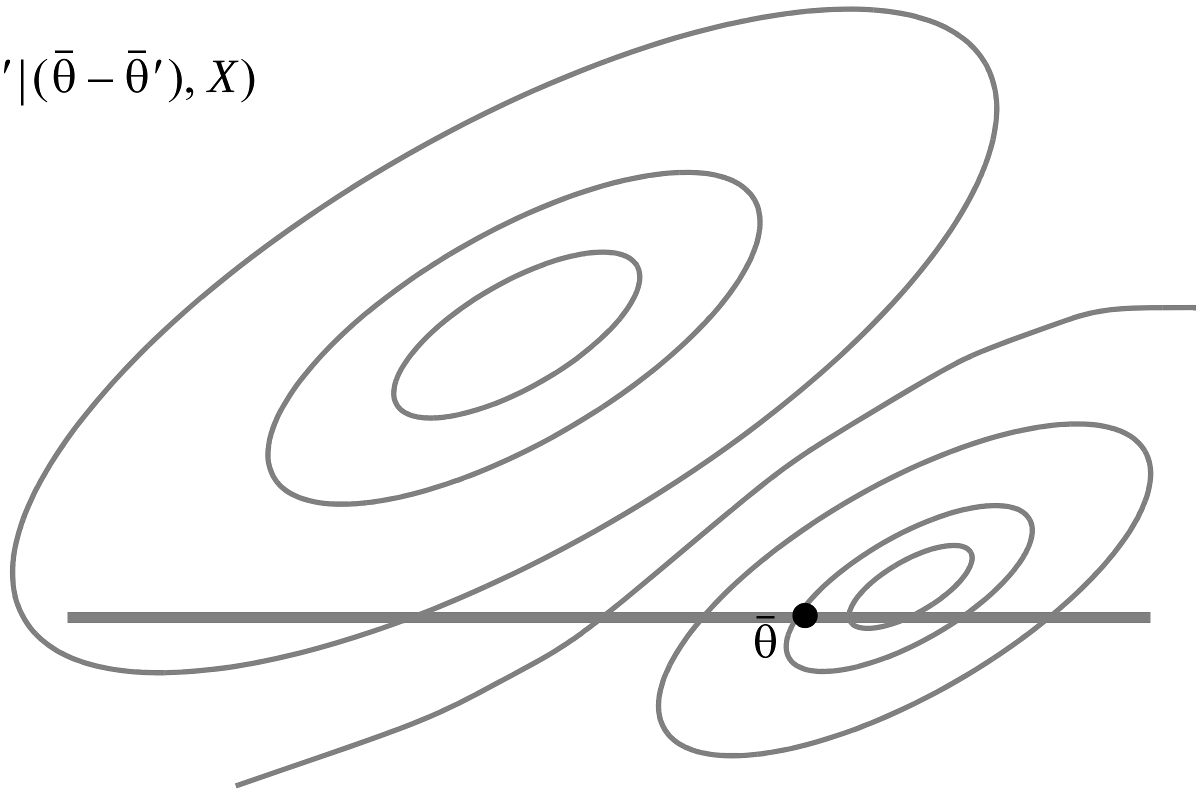


MCMC: Gibbs Sampling

- Many MCMC algorithms; useful example is Gibbs sampling

  — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

  1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)
  2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$
  3. Repeat forever!

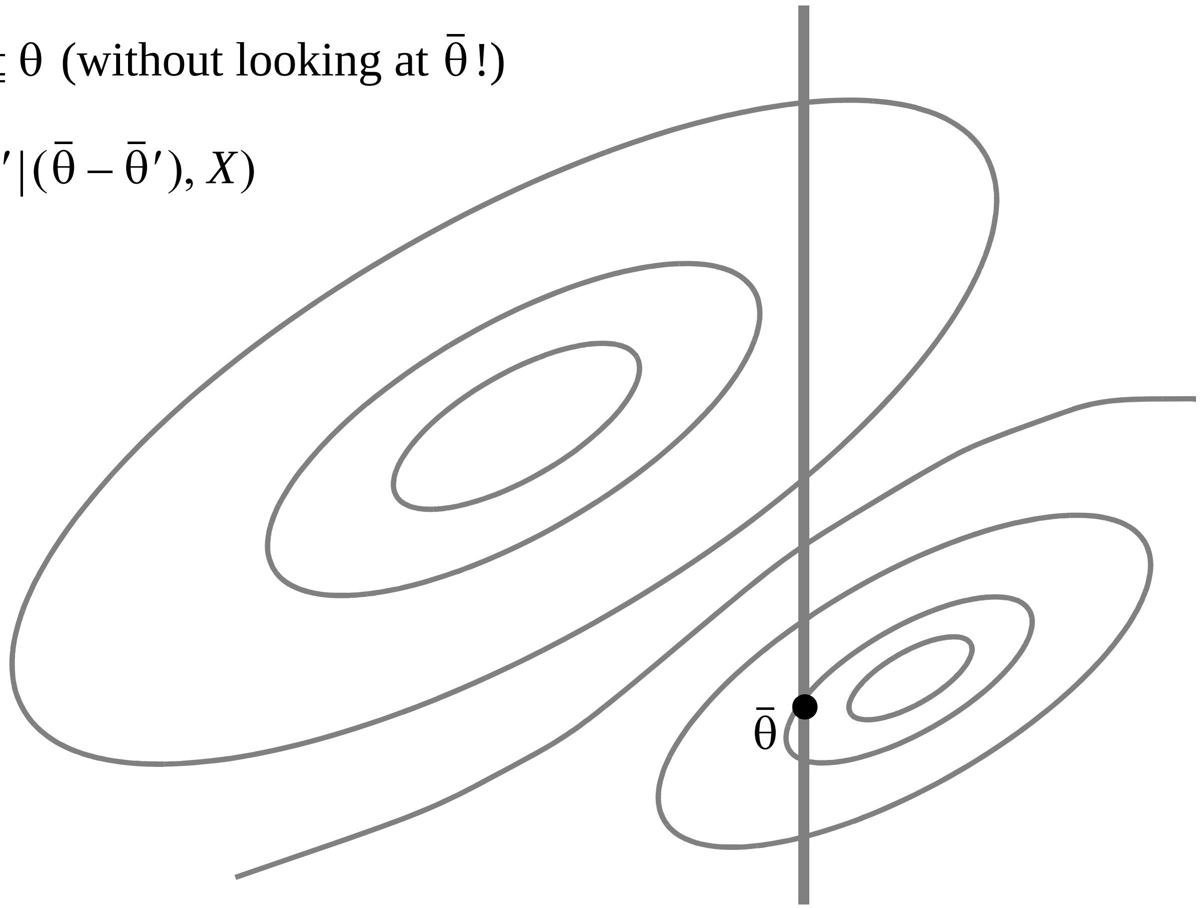


MCMC: Gibbs Sampling

- Many MCMC algorithms; useful example is Gibbs sampling

  — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

  1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)
  2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$
  3. Repeat forever!



MCMC: Gibbs Sampling

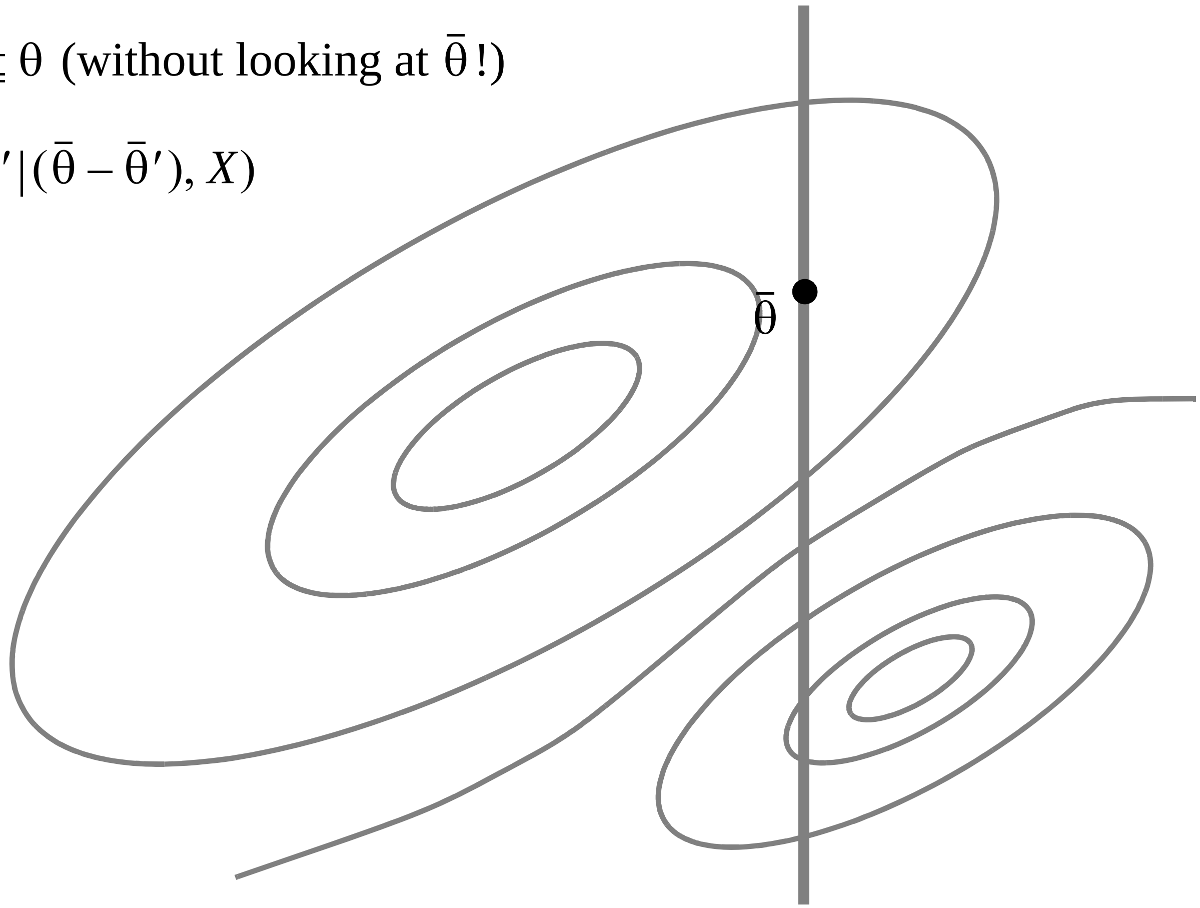
- Many MCMC algorithms; useful example is Gibbs sampling

  — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

  1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)

  2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$

  3. Repeat forever!

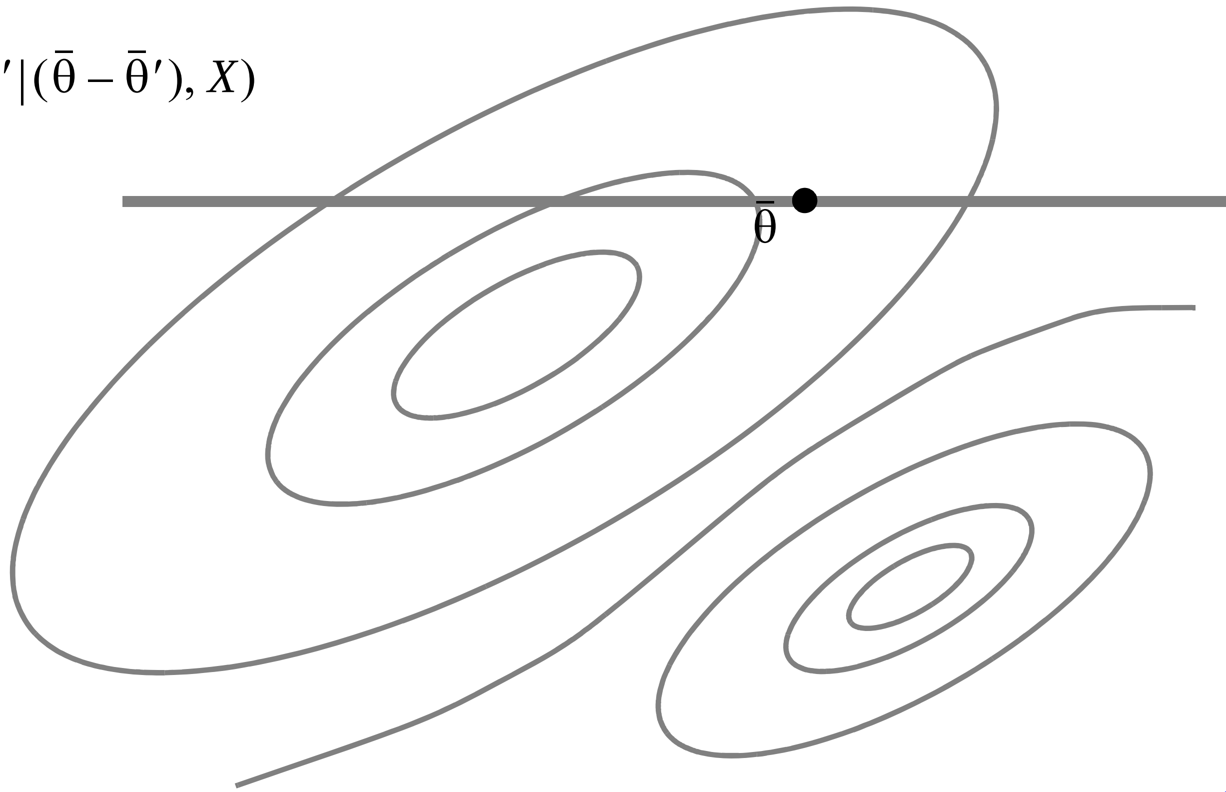


MCMC: Gibbs Sampling

- Many MCMC algorithms; useful example is Gibbs sampling

  — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

  1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)
  2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$
  3. Repeat forever!

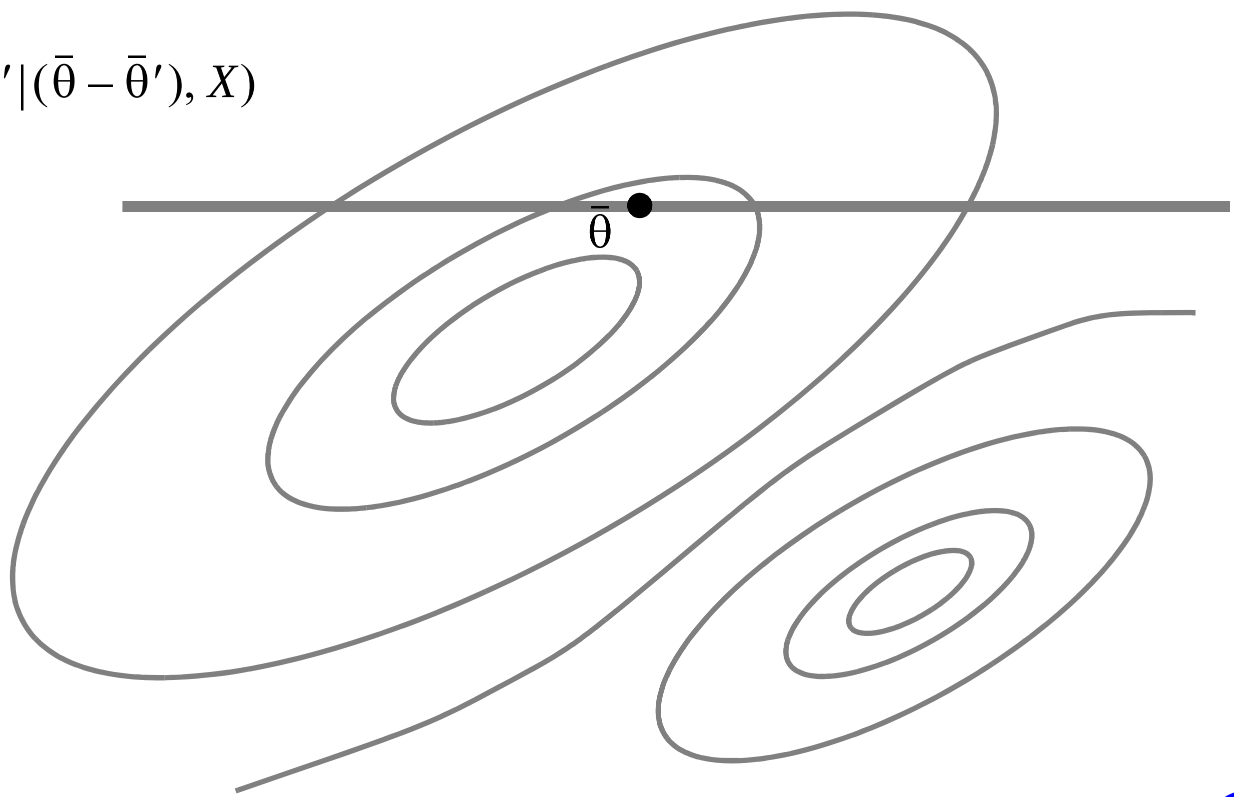


MCMC: Gibbs Sampling

- Many MCMC algorithms; useful example is Gibbs sampling

  — Unknown vars/params in θ; state of chain is described by $\bar{\theta}$

  1. Pick subset $\theta' \subseteq \theta$ (without looking at $\bar{\theta}$!)
  2. Sample $\bar{\theta}' \sim f(\theta' | (\bar{\theta} - \bar{\theta}'), X)$
  3. Repeat forever!



How To Do MCMC Inference Over Big Data?

- Easy to spec MCMC simulations in SimSQL SQL

SimSQL's Version of SQL

- Most fundamental SQL addition is "VG Function" abstraction
- Called via a special, stochastic `CREATE TABLE` statement
- Example; assuming:
  - `SBP(MEAN, STD, GENDER)`
  - `PATIENTS(NAME, GENDER)`
- To create a stochastic table, we might have:

```
CREATE TABLE SBP_DATA(NAME, GENDER, SBP) AS
FOR EACH p in PATIENTS
  WITH Res AS Normal (
    SELECT s.MEAN, s.STD
    FROM SPB s WHERE s.GENDER = p.GENDER)
  SELECT p.NAME, p.GENDER, r.VALUE
  FROM Res r
```

How Does This Work?

```
CREATE TABLE SBP_DATA(NAME, GENDER, SBP) AS
FOR EACH p in PATIENTS
  WITH Res AS Normal (
    SELECT s.MEAN, s.STD
    FROM SPB s WHERE s.GENDER = p.GENDER)
  SELECT p.NAME, p.GENDER, r.VALUE
  FROM Res r
```

Loop through PATIENTS

```
PATIENTS (NAME, GENDER)
(Joe, Male) "p"
(Tom, Male)
(Jen, Female)
(Sue, Female)
(Jim, Male)
```

```
SBP(MEAN, STD, GENDER)
(150, 20, Male)
(130, 25, Female)
```

How Does This Work?

```
CREATE TABLE SBP_DATA(NAME, GENDER, SBP) AS
FOR EACH p in PATIENTS
  WITH Res AS Normal (
    SELECT s.MEAN, s.STD
    FROM SPB s WHERE s.GENDER = p.GENDER)
  SELECT p.NAME, p.GENDER, r.VALUE
  FROM Res r
```

PATIENTS (NAME, GENDER)
(Joe, Male) “p”
(Tom, Male)
(Jen, Female)
(Sue, Female)
(Jim, Male)

SBP(MEAN, STD, GENDER)
(150, 20, Male)
(130, 25, Female)

How Does This Work?

```
CREATE TABLE SBP_DATA(NAME, GENDER, SBP) AS
FOR EACH p in PATIENTS
  WITH Res AS Normal (
    SELECT s.MEAN, s.STD
    FROM SPB s WHERE s.GENDER = p.GENDER)
  SELECT p.NAME, p.GENDER, r.VALUE
  FROM Res r
```

```
PATIENTS (NAME, GENDER)
(Joe, Male) “p”
(Tom, Male)
(Jen, Female)
(Sue, Female)
(Jim, Male)
```

```
SBP(MEAN, STD, GENDER)
(150, 20, Male)
(130, 25, Female)
```

Normal(**150**,**20**)

How Does This Work?

```
CREATE TABLE SBP_DATA(NAME, GENDER, SBP) AS
FOR EACH p in PATIENTS
  WITH Res AS Normal (
    SELECT s.MEAN, s.STD
    FROM SPB s WHERE s.GENDER = p.GENDER)
  SELECT p.NAME, p.GENDER, r.VALUE
  FROM Res r
```

```
PATIENTS (NAME, GENDER)
(Joe, Male) “p”
(Tom, Male)
(Jen, Female)
(Sue, Female)
(Jim, Male)
```

```
SBP(MEAN, STD, GENDER)
(150, 20, Male)
(130, 25, Female)
```

```
Normal(150,20)
```

```
Res(VALUE)
(162)
```

How Does This Work?

```
CREATE TABLE SBP_DATA(NAME, GENDER, SBP) AS
FOR EACH p in PATIENTS
  WITH Res AS Normal (
    SELECT s.MEAN, s.STD
    FROM SPB s WHERE s.GENDER = p.GENDER)
  SELECT p.NAME, p.GENDER, r.VALUE
  FROM Res r
```

```
PATIENTS (NAME, GENDER)
(Joe, Male) “p”
(Tom, Male)
(Jen, Female)
(Sue, Female)
(Jim, Male)
```

```
SBP(MEAN, STD, GENDER)
(150, 20, Male)
(130, 25, Female)
```

```
SBP_DATA (NAME, GENDER, SPB)
(Joe, Male, 162)
```

Normal(150,20)

```
Res(VALUE)
(162)
```

How Does This Work?

```
CREATE TABLE SBP_DATA(NAME, GENDER, SBP) AS
FOR EACH p in PATIENTS
  WITH Res AS Normal (
    SELECT s.MEAN, s.STD
    FROM SPB s WHERE s.GENDER = p.GENDER)
  SELECT p.NAME, p.GENDER, r.VALUE
  FROM Res r
```

```
PATIENTS (NAME, GENDER)
(Joe, Male)
(Tom, Male) “p”
(Jen, Female)
(Sue, Female)
(Jim, Male)
```

```
SBP(MEAN, STD, GENDER)
(150, 20, Male)
(130, 25, Female)
```

```
SBP_DATA (NAME, GENDER, SPB)
(Joe, Male, 162)
```

```
Normal(150,20)
```

```
Res(VALUE)
(135)
```

How Does This Work?

```
CREATE TABLE SBP_DATA(NAME, GENDER, SBP) AS
FOR EACH p in PATIENTS
  WITH Res AS Normal (
    SELECT s.MEAN, s.STD
    FROM SPB s WHERE s.GENDER = p.GENDER)
  SELECT p.NAME, p.GENDER, r.VALUE
  FROM Res r
```

PATIENTS (NAME, GENDER)
(Joe, Male)
(Tom, Male) “p”
(Jen, Female)
(Sue, Female)
(Jim, Male)

SBP(MEAN, STD, GENDER)
(150, 20, Male)
(130, 25, Female)

SBP_DATA (NAME, GENDER, SPB)
(Joe, Male, 162)
(Tom, Male, 135)

Normal(150,20)

Res(VALUE)
(**135**)

How Does This Work?

```
CREATE TABLE SBP_DATA(NAME, GENDER, SBP) AS
FOR EACH p in PATIENTS
  WITH Res AS Normal (
    SELECT s.MEAN, s.STD
    FROM SPB s WHERE s.GENDER = p.GENDER)
  SELECT p.NAME, p.GENDER, r.VALUE
  FROM Res r
```

```
PATIENTS (NAME, GENDER)
(Joe, Male)
(Tom, Male)
(Jen, Female) “p”
(Sue, Female)
(Jim, Male)
```

```
SBP(MEAN, STD, GENDER)
(150, 20, Male)
(130, 25, Female)
```

```
SBP_DATA (NAME, GENDER, SPB)
(Joe, Male, 162)
(Tom, Male, 135)
```

```
Normal(130,25)
```

```
Res(VALUE)
(112)
```

How Does This Work?

```
CREATE TABLE SBP_DATA(NAME, GENDER, SBP) AS
FOR EACH p in PATIENTS
  WITH Res AS Normal (
    SELECT s.MEAN, s.STD
    FROM SPB s WHERE s.GENDER = p.GENDER)
  SELECT p.NAME, p.GENDER, r.VALUE
  FROM Res r
```

```
PATIENTS (NAME, GENDER)
(Joe, Male)
(Tom, Male)
(Jen, Female) “p”
(Sue, Female)
(Jim, Male)
```

```
SBP(MEAN, STD, GENDER)
(150, 20, Male)
(130, 25, Female)
```

```
SBP_DATA (NAME, GENDER, SPB)
(Joe, Male, 162)
(Tom, Male, 135)
(Jen, Female, 112)
```

```
Normal(130,25)
```

```
Res(VALUE)
(112)
```

and so on...

Markov Chain Simulation

- Previous allows for table-valued RVs, not for Markov chains
- But Markov chains are easy in SimSQL
- Here’s a silly Markov chain. We have:
  - `PERSON (name)`
  - `LOCTION (name, dim, val)`
  - `MOVEMENT_VAR (name, dim1, dim2, var)`
  - `MOVEMENT_MEAN (name, dim, mean)`
- We want to randomly start each person at a location
- Then move them all randomly around

Markov Chain Simulation

- To select an initial starting position for each person:

```
CREATE TABLE POSITION[0] (name, dim, val) AS
FOR EACH p IN PERSON
  WITH Pos AS DiscreteChoice (
    SELECT DISTINCT name
    FROM LOCATION)
  SELECT p.name, l.dim, l.val
  FROM Pos, LOCATION l
  WHERE l.name = Pos.val
```

Markov Chain Simulation

- And then to move them all along:

```
CREATE TABLE POSITION[i] (name, dim, val) AS
FOR EACH p IN PERSON
  WITH Pos AS ConditionalNormal (
    (SELECT pos.dim, pos.val
     FROM POSITION[i - 1] pos
     WHERE pos.dim = i MOD 2 AND pos.name = p.name)
    (SELECT m.dim1, m.dim2, m.var
     FROM MOVEMENT_VAR m
     WHERE m.name = p.name)
    (SELECT m.dim, m.mean
     FROM MOVEMENT_MEAN m
     WHERE m.name = p.name))
  SELECT p.name, Pos.dim, Pos.val
  FROM Pos
```

- Now we’ve fully spec’d a distributed Markov chain simulation!

Getting This To Run

- Can use a lot of standard parallel DB techniques to implement
- But some problems are quite unique to SimSQL
  - — No time to talk about them today!
  - — Perhaps informally at end of talk?

How Well Does All of This Work?

- SimSQL is great in theory...
  - — Many will buy the “data independence” argument
  - — Will appreciate being able to specify algs at a very high level
- But isn’t the declarative approach gonna be slow?

How Well Does All of This Work?

- SimSQL is great in theory...
  - — Many will buy the “data independence” argument
  - — Will appreciate being able to specify algs at a very high level
- But isn’t the declarative approach gonna be slow?
- Yes, it’s slow, compared to C/Fortran + MPI
  - — But zero data independence with MPI

How Well Does All of This Work?

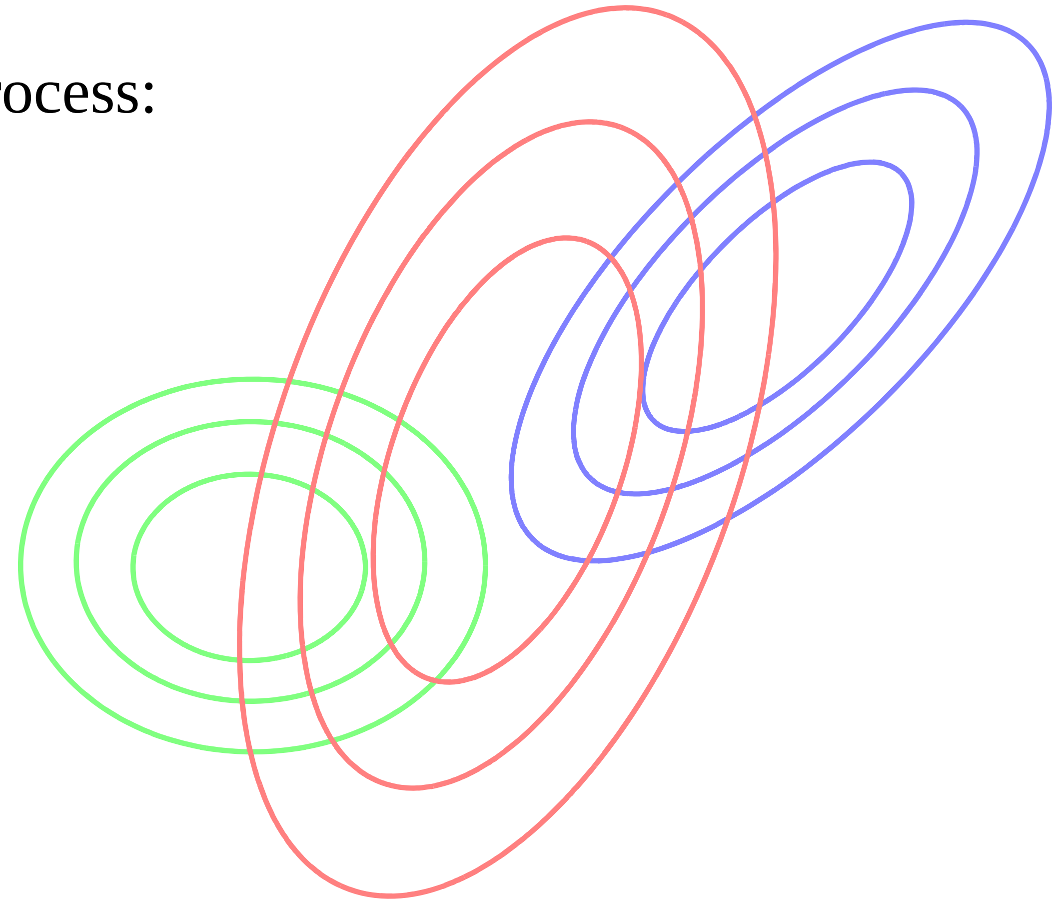
- SimSQL is great in theory...
  - Many will buy the “data independence” argument
  - Will appreciate being able to specify algs at a very high level
- But isn’t the declarative approach gonna be slow?
- Yes, it’s slow, compared to C/Fortran + MPI
  - But zero data independence with MPI
- But does it compete well with other “Big Data” ML platforms?
  - After all, are many that count ML as the primary (or a motivating) application
  - OptiML, GraphLab, SystemML, MLBase, ScalOps, Pregel, Giraph, Hama, Spark, Ricardo, Nyad, DradLinq
  - How might those compare?

How Well Does All of This Work?

- We’ve done a **LOT** of comparisons with other mature platforms
  - — Specifically, GraphLab, Giraph, Spark
  - — More than 70,000 hours of Amazon EC2 time ($100,000 @on-demand price)
  - — I’d wager that few groups have a better understanding of how well these platforms work in practice!
- Note: point is not to show SimSQL is the fastest (it is not)
  - — Only to argue that it can compete well
  - — If it competes, it’s a strong argument for the declarative approach to ML
- Note: this is hand-coded SimSQL SQL
  - — Not SQL compiled from BUDS
  - — Will get those results soon!

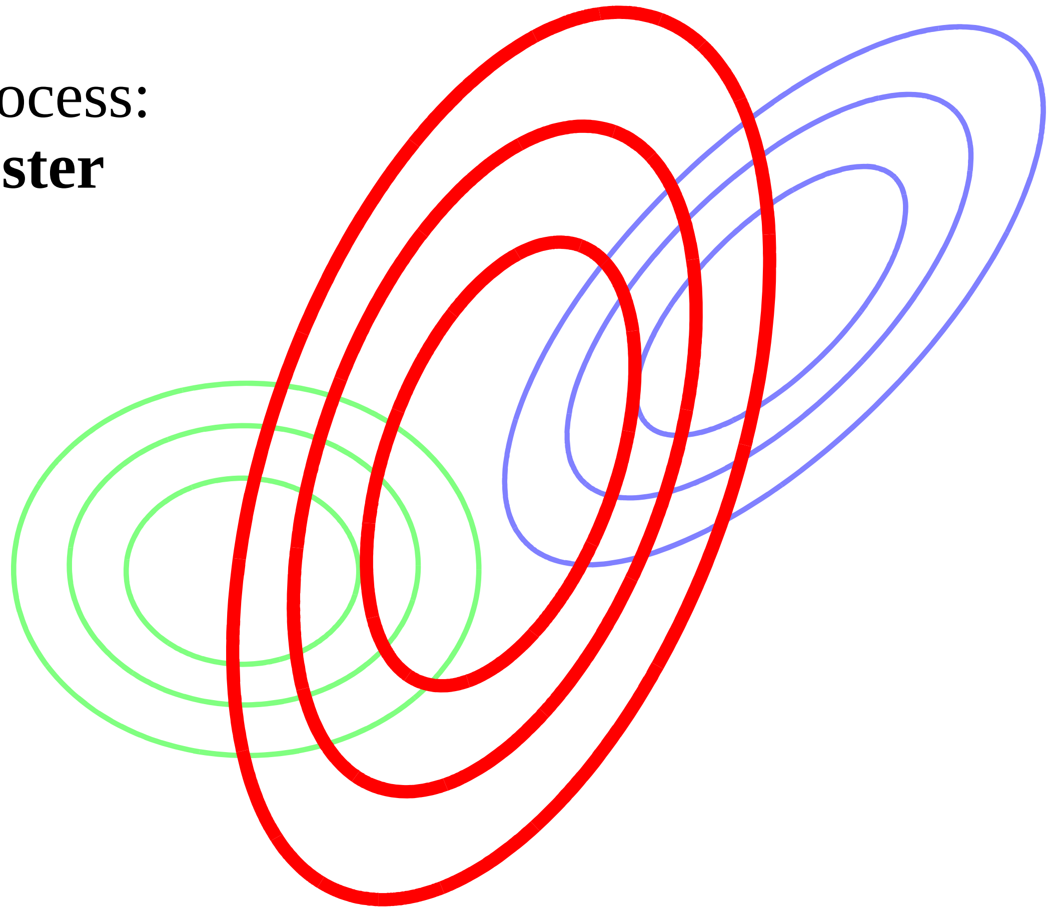
Example One: Bayesian GMM

Generative process:



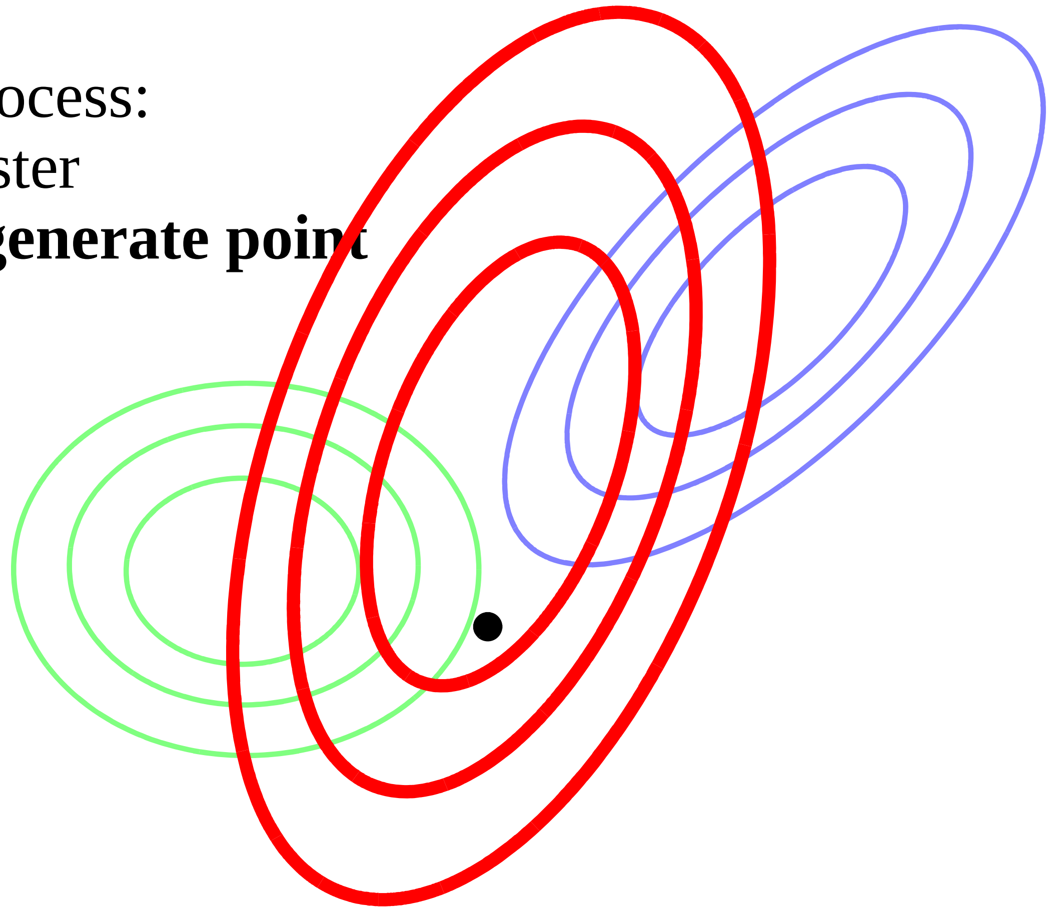
Example One: Bayesian GMM

Generative process:
(1) **Pick a cluster**



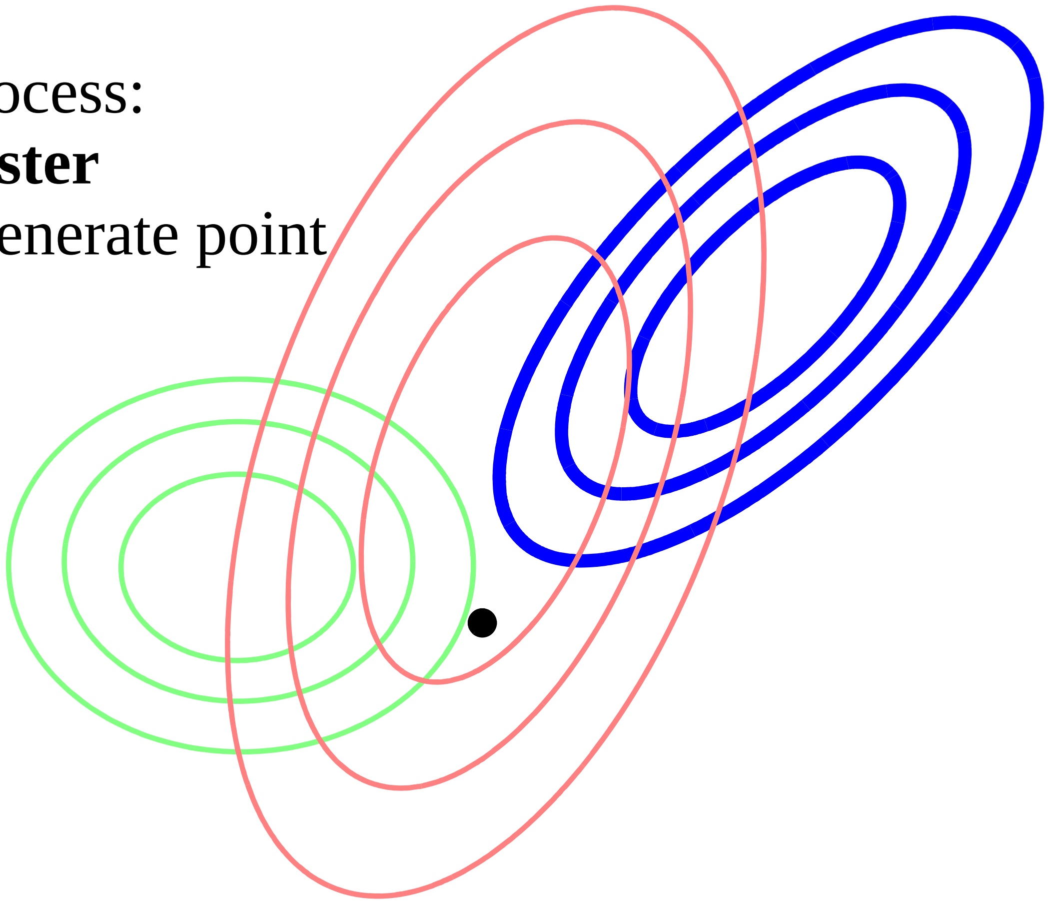
Example One: Bayesian GMM

Generative process:
(1) Pick a cluster
(2) **Use it to generate point**



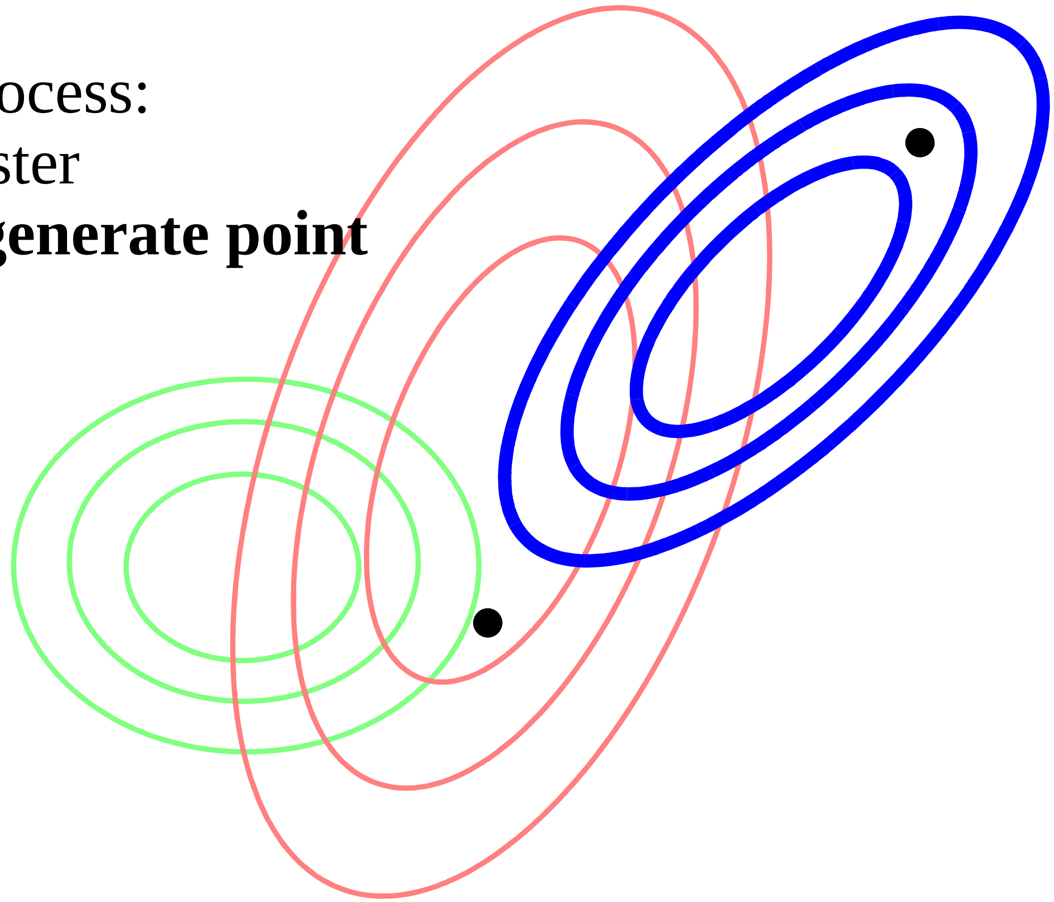
Example One: Bayesian GMM

Generative process:
(1) **Pick a cluster**
(2) Use it to generate point



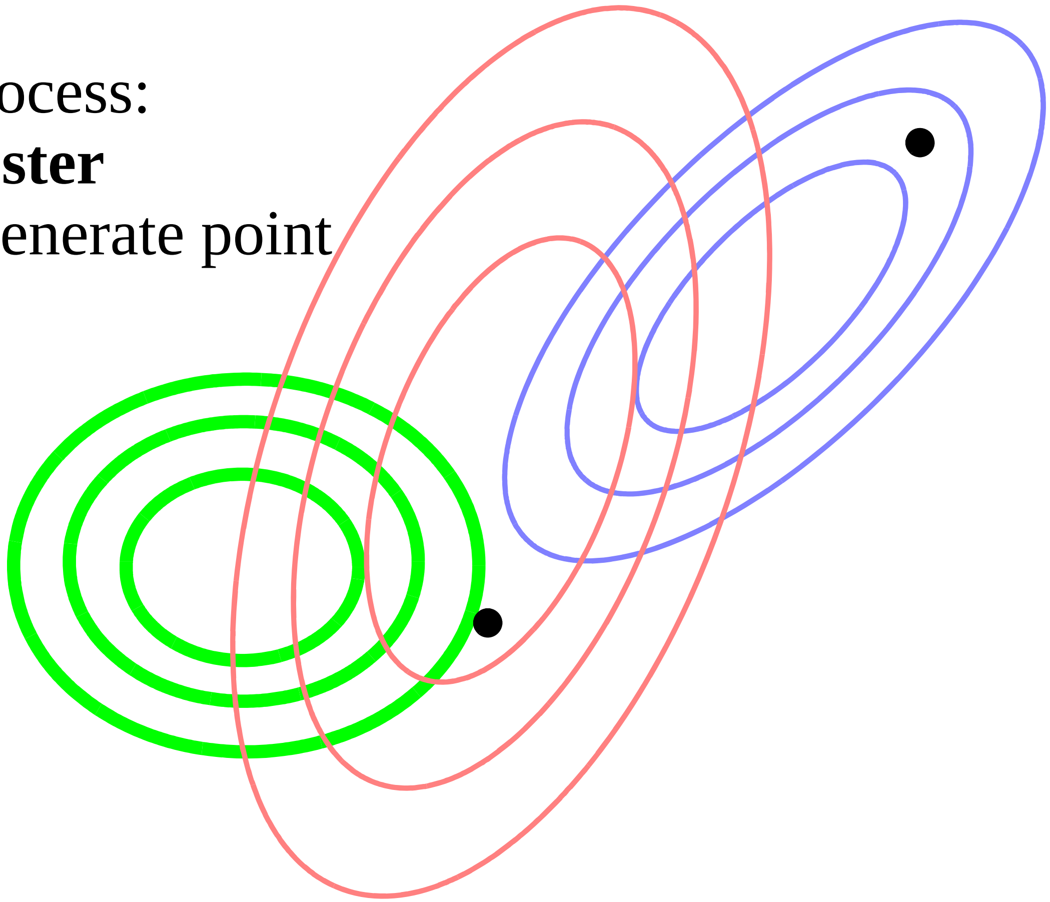
Example One: Bayesian GMM

Generative process:
(1) Pick a cluster
(2) **Use it to generate point**



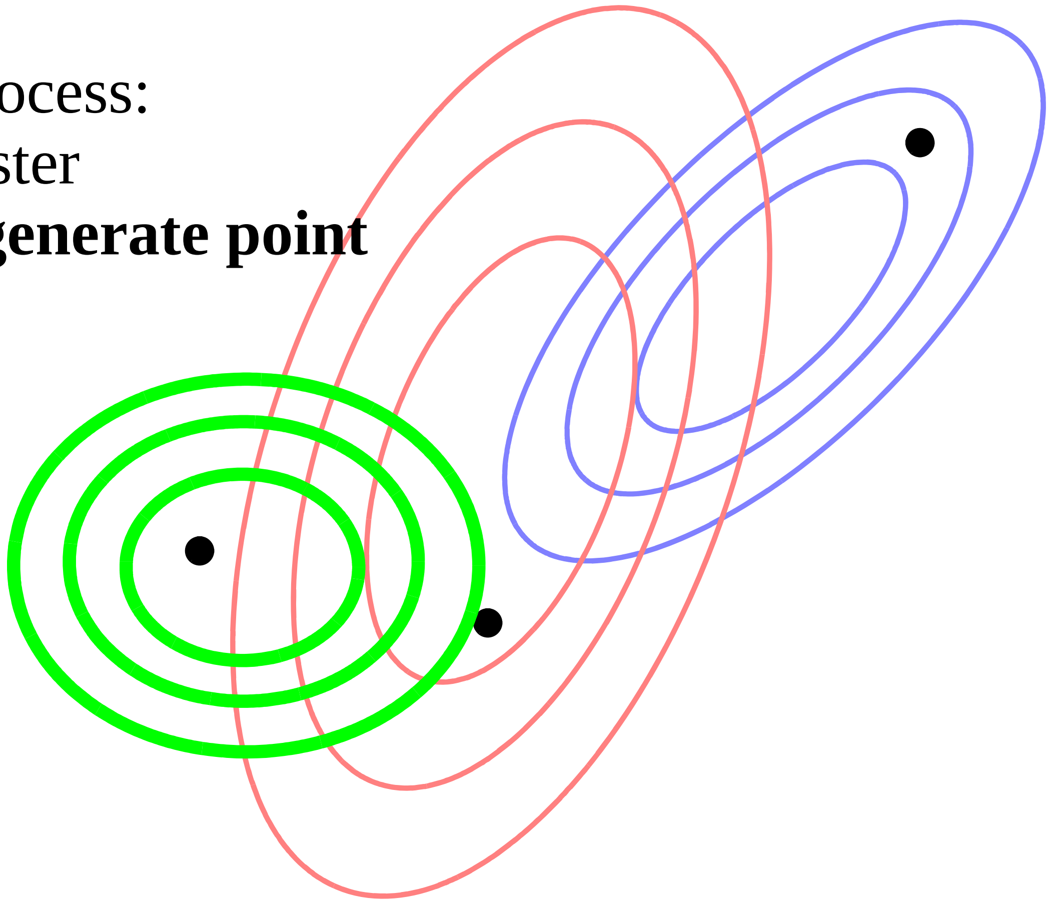
Example One: Bayesian GMM

Generative process:
(1) **Pick a cluster**
(2) Use it to generate point



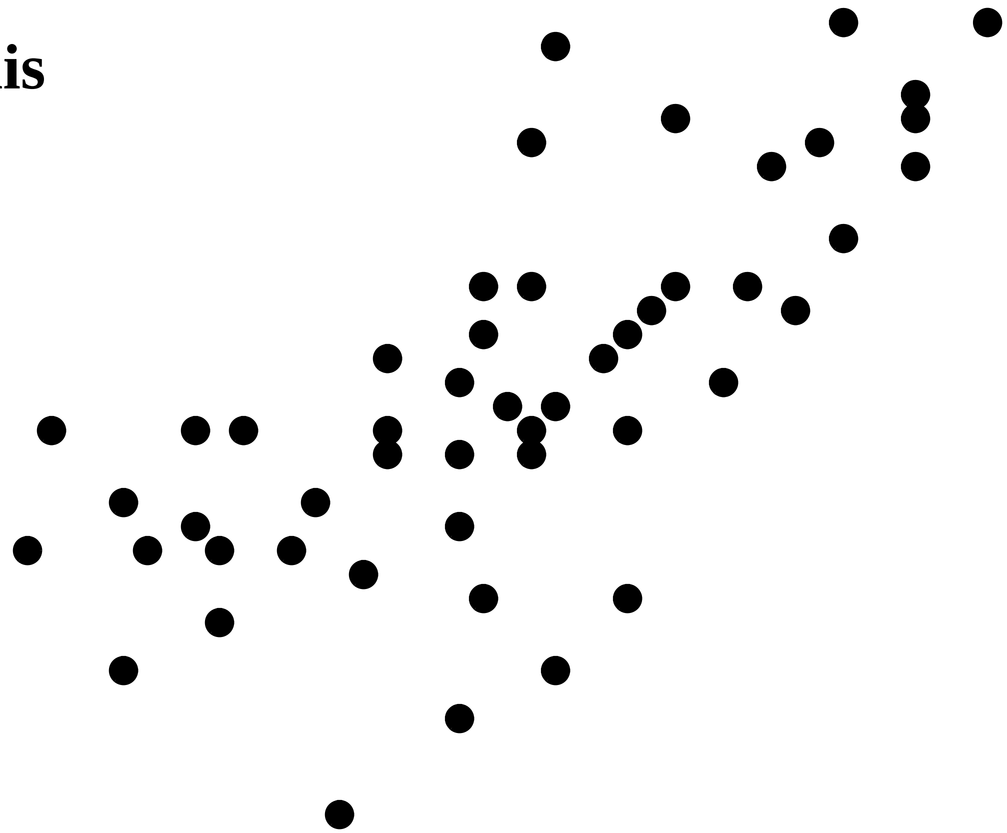
Example One: Bayesian GMM

Generative process:
(1) Pick a cluster
(2) **Use it to generate point**



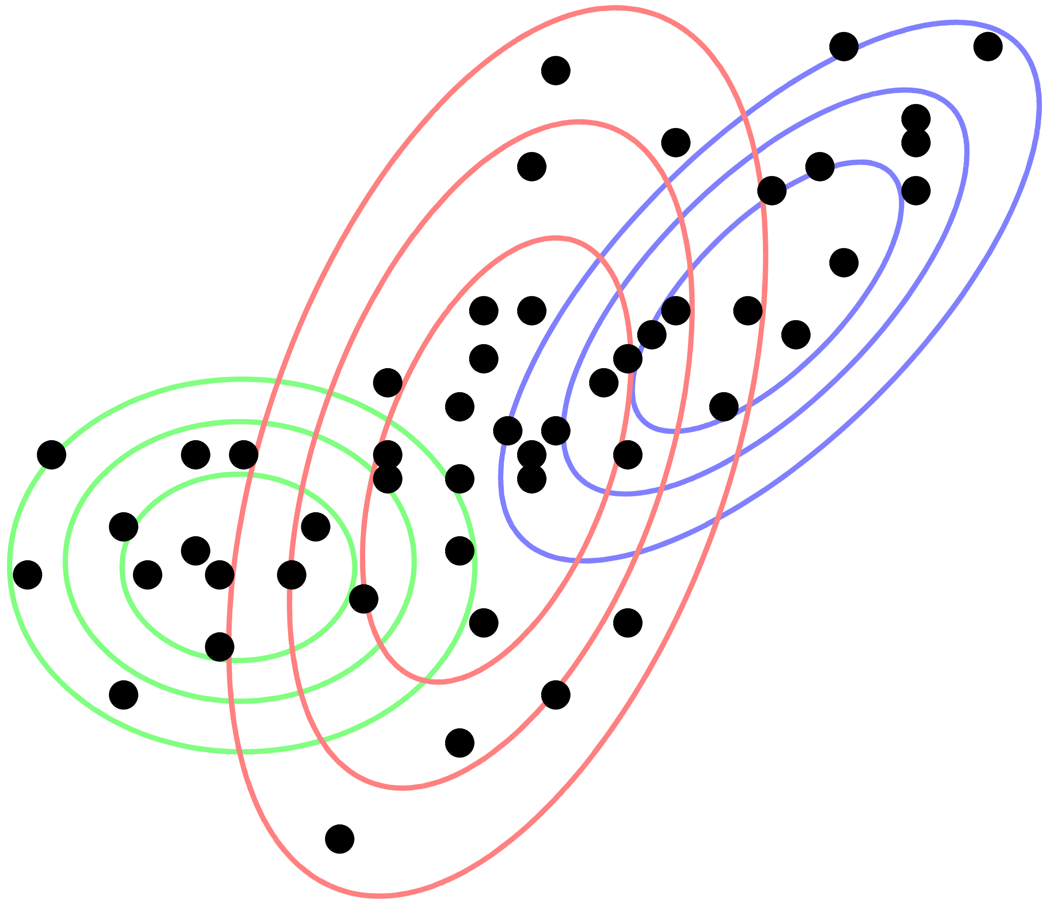
Example One: Bayesian GMM

Then given **this**



Example One: Bayesian GMM

Infer **this**



Example One: Bayesian GMM

- Implemented relevant MCMC simulation on all four platforms
  - — SimSQL, GraphLab, Spark, Giraph

Example One: Bayesian GMM

- Implemented relevant MCMC simulation on all four platforms
  — SimSQL, GraphLab, Spark, Giraph
- Philosophy: be true to the platform
  — Ex: avoid “Hadoop abuse” [Smola & Narayanamurthy, VLDB 2010]

Example One: Bayesian GMM

- Implemented relevant MCMC simulation on all four platforms
  - — SimSQL, GraphLab, Spark, Giraph
- Philosophy: be true to the platform
  - — Ex: avoid “Hadoop abuse” [Smola & Narayanamurthy, VLDB 2010]
- Ran on 10 dimensional data, 10 clusters, 10M points per machine
  - — Full (non-diagonal) covariance matrix
  - — Also on 100 dimensional data, 1M points per machine

Example One: Bayesian GMM

GMM: Initial Implementations					
		10 dimensions			100 dimensions
	lines of code	5 machines	20 machines	100 machines	5 machines
SimSQL	197	27:55 (13:55)	28:55 (14:38)	35:54 (18:58)	1:51:12 (36:08)
GraphLab	661	Fail	Fail	Fail	Fail
Spark (Python)	236	26:04 (4:10)	37:34 (2:27)	38:09 (2:00)	47:40 (0:52)
Giraph	2131	25:21 (0:18)	30:26 (0:15)	Fail	Fail

- Some notes:
  - — Times are HH:MM:SS per iteration (time in parens is startup/initialization)
  - — Amount of data is kept constant per machine in all tests
  - — “Fail” means that even with much effort and tuning, it crashed

Example One: Bayesian GMM

GMM: Initial Implementations					
		10 dimensions			100 dimensions
	lines of code	5 machines	20 machines	100 machines	5 machines
SimSQL	197	27:55 (13:55)	28:55 (14:38)	35:54 (18:58)	1:51:12 (36:08)
GraphLab	661	Fail	Fail	Fail	Fail
Spark (Python)	236	26:04 (4:10)	37:34 (2:27)	38:09 (2:00)	47:40 (0:52)
Giraph	2131	25:21 (0:18)	30:26 (0:15)	Fail	Fail

- Not much difference!
  - — But SimSQL was slower in 100 dims. Why?
    - -No native support for vectors/matrices at time tests were run
    - -Forget array databases, *this* is an important problem!

Example One: Bayesian GMM

GMM: Initial Implementations					
		10 dimensions			100 dimensions
	lines of code	5 machines	20 machines	100 machines	5 machines
SimSQL	197	27:55 (13:55)	28:55 (14:38)	35:54 (18:58)	1:51:12 (36:08)
GraphLab	661	Fail	Fail	Fail	Fail
Spark (Python)	236	26:04 (4:10)	37:34 (2:27)	38:09 (2:00)	47:40 (0:52)
Giraph	2131	25:21 (0:18)	30:26 (0:15)	Fail	Fail

- Spark is surprisingly slow
  - — Is Spark slower due to Python vs. Java?

GMM: Alternative Implementations					
		10 dimensions			100 dimensions
	lines of code	5 machines	20 machines	100 machines	5 machines
Spark (Java)	737	12:30 (2:01)	12:25 (2:03)	18:11 (2:26)	6:25:04 (36:08)

Example One: Bayesian GMM

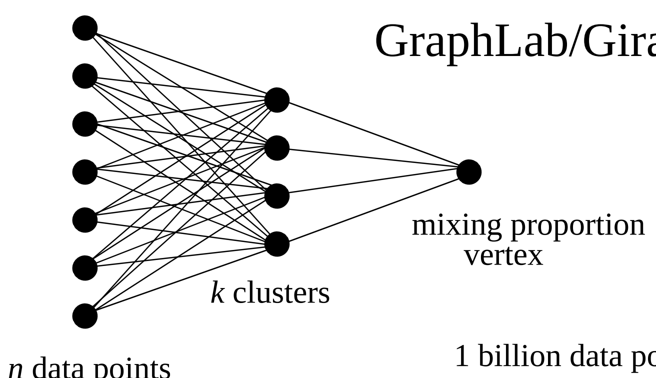
GMM: Initial Implementations					
		10 dimensions			100 dimensions
	lines of code	5 machines	20 machines	100 machines	5 machines
SimSQL	197	27:55 (13:55)	28:55 (14:38)	35:54 (18:58)	1:51:12 (36:08)
GraphLab	661	Fail	Fail	Fail	Fail
Spark (Python)	236	26:04 (4:10)	37:34 (2:27)	38:09 (2:00)	47:40 (0:52)
Giraph	2131	25:21 (0:18)	30:26 (0:15)	Fail	Fail

- What about GraphLab?
  — GraphLab failed every time. Why?

Example One: Bayesian GMM

GMM: Initial Implementations					
		10 dimensions			100 dimensions
	lines of code	5 machines	20 machines	100 machines	5 machines
SimSQL	197	27:55 (13:55)	28:55 (14:38)	35:54 (18:58)	1:51:12 (36:08)
GraphLab	661	Fail	Fail	Fail	Fail
Spark (Python)	236	26:04 (4:10)	37:34 (2:27)	38:09 (2:00)	47:40 (0:52)
Giraph	2131	25:21 (0:18)	30:26 (0:15)	Fail	Fail

- What about GraphLab?
  - — GraphLab failed every time. Why?



1 billion data points by
10 clusters by
1KB = 10TB RAM (6TB RAM in 100-machine cluster)

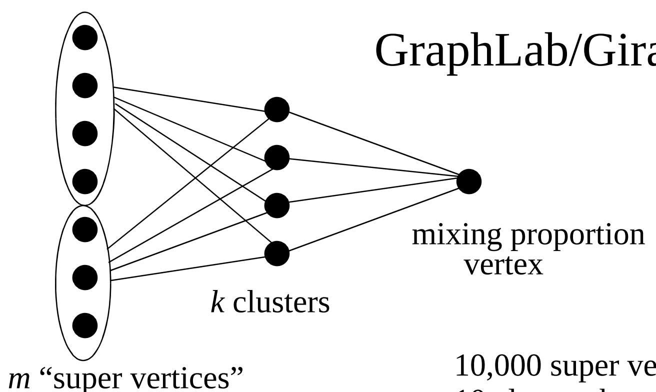
Example One: Bayesian GMM

GMM: Initial Implementations					
		10 dimensions			100 dimensions
	lines of code	5 machines	20 machines	100 machines	5 machines
SimSQL	197	27:55 (13:55)	28:55 (14:38)	35:54 (18:58)	1:51:12 (36:08)
GraphLab	661	Fail	Fail	Fail	Fail
Spark (Python)	236	26:04 (4:10)	37:34 (2:27)	38:09 (2:00)	47:40 (0:52)
Giraph	2131	25:21 (0:18)	30:26 (0:15)	Fail	Fail

- What about GraphLab?
  - — GraphLab failed every time. Why?

To Fix...

GraphLab/Giraph graph model



10,000 super vertices
10 clusters by
1KB = 100 MB RAM (insignificant!)

Example One: Bayesian GMM

GMM: Alternative Implementations					
		10 dimensions			100 dimensions
	lines of code	5 machines	20 machines	100 machines	5 machines
GraphLab (Super Vertex)	681	6:13 (1:13)	4:36 (2:47)	6:09 (1:21)*	33:32 (0:42)

- Super vertex results
  - — GraphLab super vertex screams!

Example One: Bayesian GMM

GMM: Alternative Implementations					
		10 dimensions			100 dimensions
	lines of code	5 machines	20 machines	100 machines	5 machines
GraphLab (Super Vertex)	681	6:13 (1:13)	4:36 (2:47)	6:09 (1:21)*	33:32 (0:42)

- Super vertex results
  - — GraphLab super vertex screams!
  - — But to be fair, others can benefit from super vertices as well...

.

GMM: Super Vertex Implementations				
	10 dimensions, 5 machines		100 dimensions, 5 machines	
	w/o super vertex	with super vertex	w/o super vertex	with super vertex
SimSQL	27:55 (13:55)	6:20 (12:33)	1:51:12 (36:08)	7:22 (14:07)
GraphLab	Fail	6:13 (1:13)	Fail	33:32 (0:42)
Spark (Python)	26:04 (4:10)	29:12 (4:01)	47:40 (0:52)	47:03 (2:17)
Giraph	25:21 (0:18)	13:48 (0:03)	Fail	6:17:32 (0:03)

Example Two: Bayesian Lasso

- Experimental setup
  - — 1K regressors (dense)
  - — 100K points per machine

Example Two: Bayesian Lasso

- Experimental setup
  - — 1K regressors (dense)
  - — 100K points per machine
- Results

Bayesian Lasso				
	lines of code	5 machines	20 machines	100 machines
SimSQL	100	7:09 (2:40:06)	8:04 (2:45:28)	12:24 (2:54:45)
GraphLab (Super Vertex)	572	0:36 (0:37)	0:26 (0:35)	0:31 (0:50)
Spark (Python)	168	0:55 (1:26:59)	0:59 (1:33:13)	1:12 (2:06:30)
Giraph	1871	Fail	Fail	Fail
Giraph (Super Vertex)	1953	0:58 (1:14)	1:03 (1:14)	2:08 (6:31)

Example Two: Bayesian Lasso

- Experimental setup
  - — 1K regressors (dense)
  - — 100K points per machine
- Results

Bayesian Lasso				
	lines of code	5 machines	20 machines	100 machines
SimSQL	100	7:09 (2:40:06)	8:04 (2:45:28)	12:24 (2:54:45)
GraphLab (Super Vertex)	572	0:36 (0:37)	0:26 (0:35)	0:31 (0:50)
Spark (Python)	168	0:55 (1:26:59)	0:59 (1:33:13)	1:12 (2:06:30)
Giraph	1871	Fail	Fail	Fail
Giraph (Super Vertex)	1953	0:58 (1:14)	1:03 (1:14)	2:08 (6:31)

- Interesting points
  - — SimSQL slow (again, lack of support for vectors/matrices is brutal here)...
  - — But Spark is almost as slow for startup (computation of Gram matrix)
  - — Check out GraphLab: super fast!

Example Three: LDA

- Sort of a Bayesian variant on PCA (for dimensionality reduction)
- Experimental setup
  - — Run over a document database, dictionary size of 10K words
  - — 100 “topics” (components) were learned
  - — Constant 2.5M documents per machine
- Note: didn’t do collapsed simulation, since hard to parallelize

Example Three: LDA

- First we considered a “word based” implementation
  - Arguably the most natural
  - One vertex for each word in corpus in graph-based
  - Separate Multnomial call for each word in each doc in SimSQL/Spark
- And a “document based” implementation
  - One vertex for each document in graph-based
  - Update membership for all words at once in SimSQL/Spark (faster ‘cause you broadcast the model, do join with words in doc in user code)

Example Three: LDA

- Results

LDA: Word-based and document-based implementations				
	Word-based, 5 machines		Document-based, 5 machines	
	lines of code	running time	lines of code	running time
SimSQL	126	16:34:39 (11:23:22)	129	4:52:06 (4:34:27)
Spark (Python)	NA	NA	188	≈15:45:00 (≈2:30:00)
Giraph	NA	NA	1358	22:22 (5:46)

- Interesting findings
  - — Only SimSQL can handle word-based imp, but really slow
  - — Only Giraph gives reasonable performance!
  - — Spark unable to join words-in-doc with topic-probs, hence an NA
  - — Giraph unable to load up word-based graph, hence an NA

Example Three: LDA

- Results

LDA: Word-based and document-based implementations				
	Word-based, 5 machines		Document-based, 5 machines	
	lines of code	running time	lines of code	running time
SimSQL	126	16:34:39 (11:23:22)	129	4:52:06 (4:34:27)
Spark (Python)	NA	NA	188	≈15:45:00 (≈2:30:00)
Giraph	NA	NA	1358	22:22 (5:46)

- Interesting findings
  - — Only SimSQL can handle word-based imp, but really slow
  - — Only Giraph gives reasonable performance!
  - — Spark unable to join words-in-doc with topic-probs, hence an NA
  - — Giraph unable to load up word-based graph, hence an NA
- How about super vertex? (handle thousands of docs in a batch)

Example Three: LDA

- Super vertex results

LDA: Super Vertex Implementations				
	lines of code	5 machines	20 machines	100 machines
Giraph	1406	18:49 (2:35)	20:02 (2:46)	Fail
GraphLab	517	39:27 (32:14)	Fail	Fail
Spark (Python)	220	≈3:56:00 (≈2:15:00)	≈3:57:00 (≈2:15:00)	Fail
SimSQL	117	1:00:17 (3:09)	1:06:59 (3:34)	1:13:58 (4:28)

- Interesting findings

  — Only SimSQL can scale to 250M docs on 100 machines

Example Three: LDA

- Super vertex results

LDA: Super Vertex Implementations				
	lines of code	5 machines	20 machines	100 machines
Giraph	1406	18:49 (2:35)	20:02 (2:46)	Fail
GraphLab	517	39:27 (32:14)	Fail	Fail
Spark (Python)	220	≈3:56:00 (≈2:15:00)	≈3:57:00 (≈2:15:00)	Fail
SimSQL	117	1:00:17 (3:09)	1:06:59 (3:34)	1:13:58 (4:28)

- Interesting findings

    — Only SimSQL can scale to 250M docs on 100 machines

    — Even super vertex can't help GraphLab here...

    -10K super vertices on 100 machines
    -each broadcasts 100 different 10K vectors to each topic node
    -10K by 10K by 100 is 10 billion numbers...
    -what if a machine gets 2 or three topic nodes?

Example Three: LDA

- Super vertex results

LDA: Super Vertex Implementations				
	lines of code	5 machines	20 machines	100 machines
Giraph	1406	18:49 (2:35)	20:02 (2:46)	Fail
GraphLab	517	39:27 (32:14)	Fail	Fail
Spark (Python)	220	≈3:56:00 (≈2:15:00)	≈3:57:00 (≈2:15:00)	Fail
SimSQL	117	1:00:17 (3:09)	1:06:59 (3:34)	1:13:58 (4:28)

- Interesting findings
  - — Only SimSQL can scale to 250M docs on 100 machines
  - — Even super vertex can't help GraphLab here...
  - — Spark does quite poorly... might this be due to Python?

Example Three: LDA

- Super vertex results

LDA: Super Vertex Implementations				
	lines of code	5 machines	20 machines	100 machines
Giraph	1406	18:49 (2:35)	20:02 (2:46)	Fail
GraphLab	517	39:27 (32:14)	Fail	Fail
Spark (Python)	220	≈3:56:00 (≈2:15:00)	≈3:57:00 (≈2:15:00)	Fail
SimSQL	117	1:00:17 (3:09)	1:06:59 (3:34)	1:13:58 (4:28)

- Interesting findings

  — Only SimSQL can scale to 250M docs on 100 machines

  — Even super vertex can’t help GraphLab here...

  — Spark does quite poorly... might this be due to Python?

LDA Spark Java Implementation			
lines of code	5 machines	20 machines	100 machines
377	9:47 (0:53)	19:36 (1:15)	Fail

Summary of Findings

- Giraph can be made very fast
  - — Mostly ‘cause of distributed aggregation facilities
  - — But it is still brittle, perhaps due to reliance on main memory

Summary of Findings

- Giraph can be made very fast
  - Mostly ‘cause of distributed aggregation facilities
  - But it is still brittle, perhaps due to reliance on main memory
- GraphLab codes are small and nice, especially considering C++
  - And it can be very fast
  - But lack of distributed agg is a killer... what does this even mean in asynch env?

Summary of Findings

- Giraph can be made very fast
  - — Mostly ‘cause of distributed aggregation facilities
  - — But it is still brittle, perhaps due to reliance on main memory
- GraphLab codes are small and nice, especially considering C++
  - — And it can be very fast
  - — But lack of distributed agg is a killer... what does this even mean in asynch env?
- Spark codes (Python) are startlingly beautiful. Wow!
  - — But Spark was brittle, hard to tune, and often slow

Summary of Findings

- Giraph can be made very fast
  - — Mostly ‘cause of distributed aggregation facilities
  - — But it is still brittle, perhaps due to reliance on main memory
- GraphLab codes are small and nice, especially considering C++
  - — And it can be very fast
  - — But lack of distributed agg is a killer... what does this even mean in asynch env?
- Spark codes (Python) are startlingly beautiful. Wow!
  - — But Spark was brittle, hard to tune, and often slow
- SimSQL codes fully declarative, and often competitive in speed
  - — Only platform to run everything we threw at it
  - — But lack of matrices and vectors really hurts

Summary of Talk

- I've motivated a relational approach to large-scale ML
  - — All about data independence!
  - — Same code works for any data set, compute platform
  - — Just drop in a new physical optimizer and runtime, keep application stack

Summary of Talk

- I've motivated a relational approach to large-scale ML
  - All about data independence!
  - Same code works for any data set, compute platform
  - Just drop in a new physical optimizer and runtime, keep application stack
- I've briefly described SimSQL, our realization of the approach

Summary of Talk

- I’ve motivated a relational approach to large-scale ML
  - — All about data independence!
  - — Same code works for any data set, compute platform
  - — Just drop in a new physical optimizer and runtime, keep application stack
- I’ve briefly described SimSQL, our realization of the approach
- And I’ve given experimental evidence the approach is practical
  - — Our Hadoop targeted optimizer and runtime competes well
  - — And its the only platform to handle everything we threw at it

That's It. Questions?

- Download SimSQL today
  - `http://cmj4.web.rice.edu/SimSQL/SimSQL.html`
- This presentation at
  - `http://cmj4.web.rice.edu/SimSQLNew.pdf`