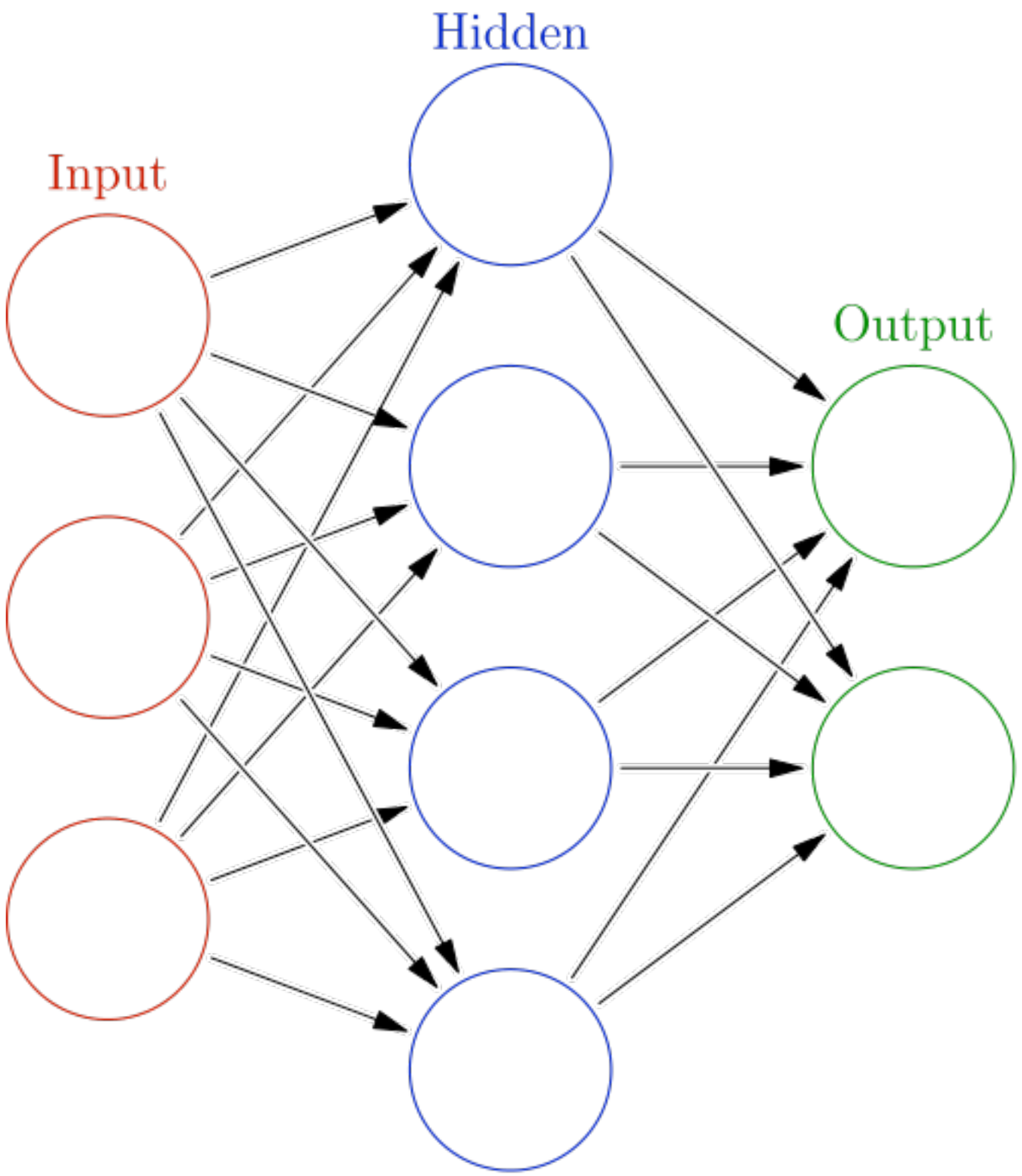
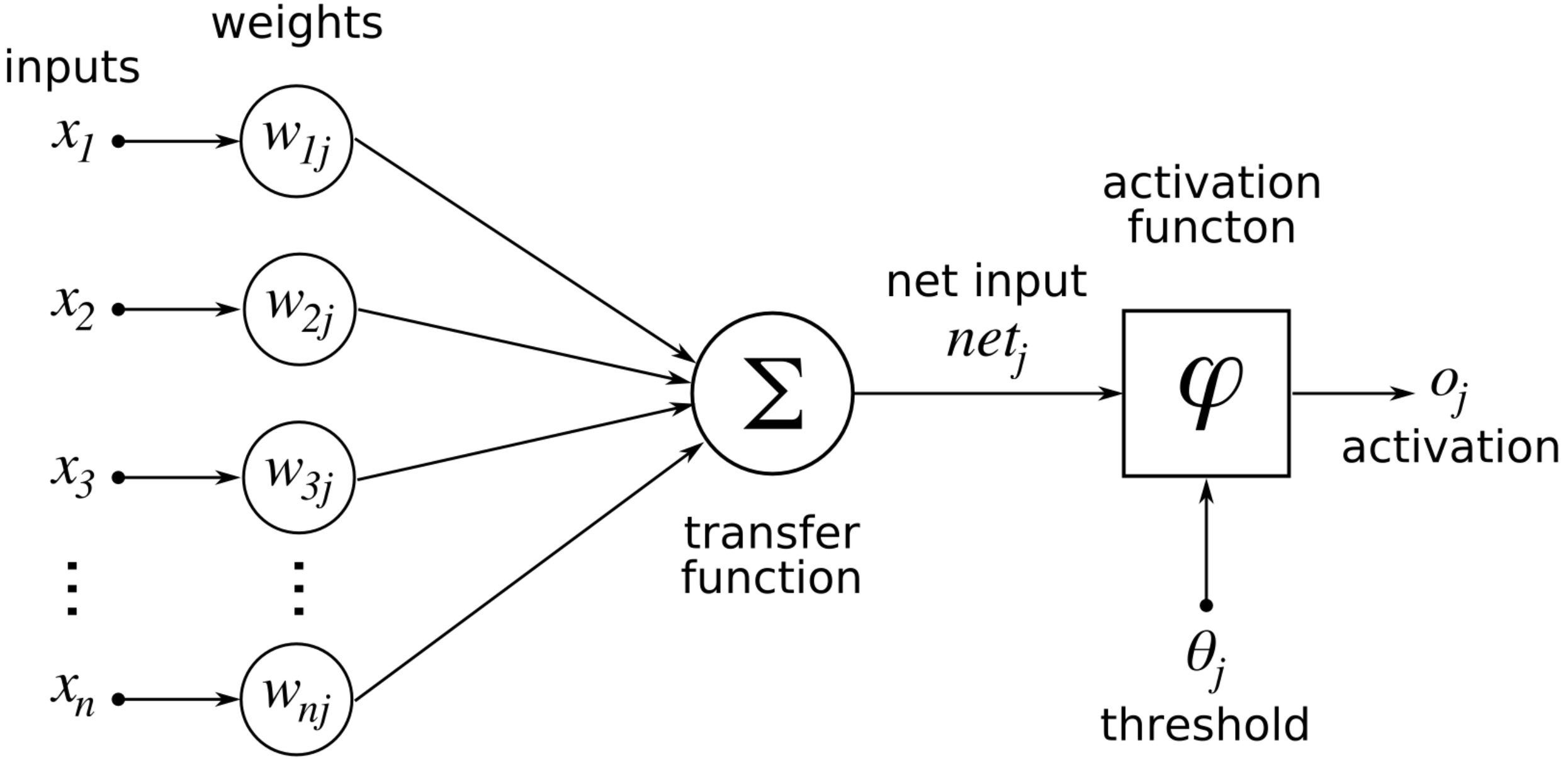
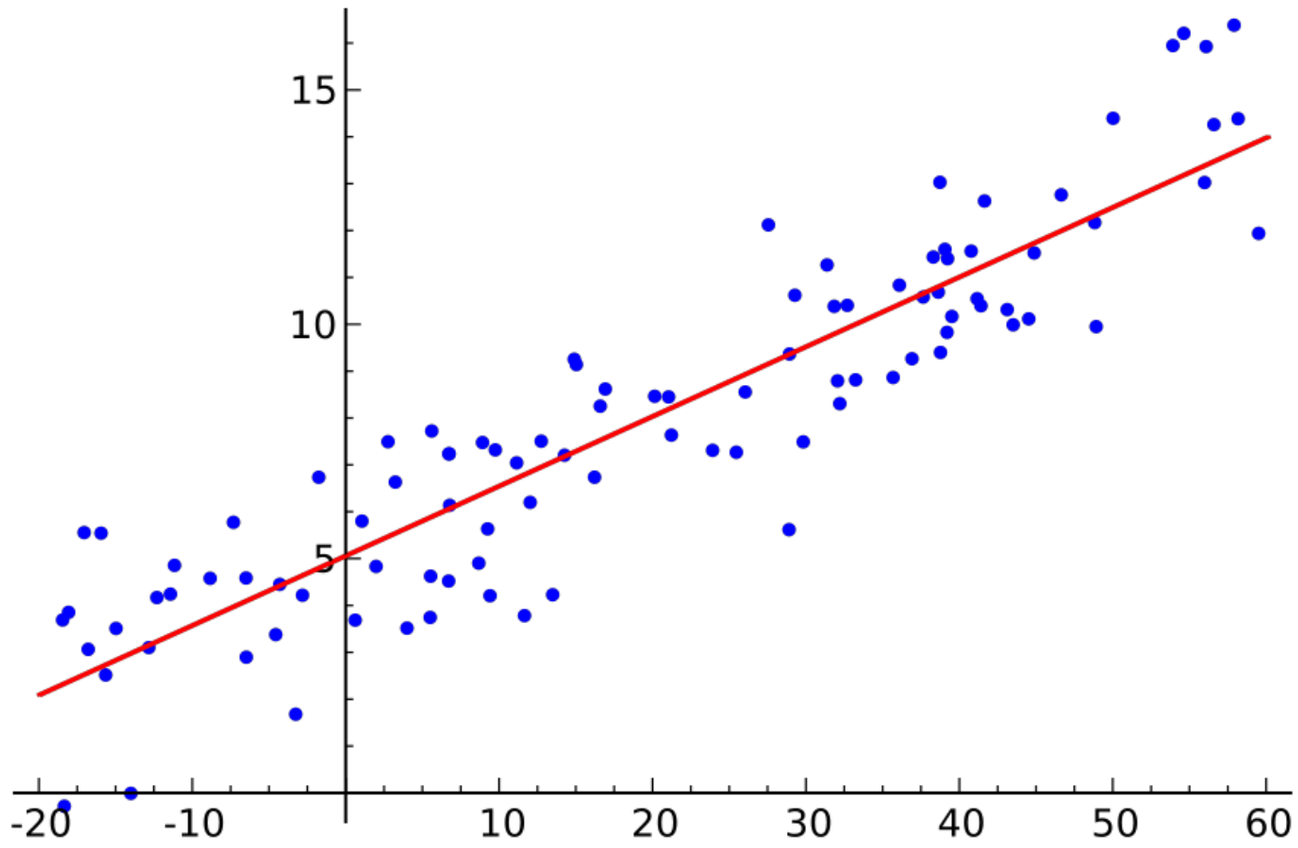
Building Brains - Parallelisation Strategies of Large-Scale Deep Learning Neural Networks on Parallel Scale Out Architectures Like Apache Spark

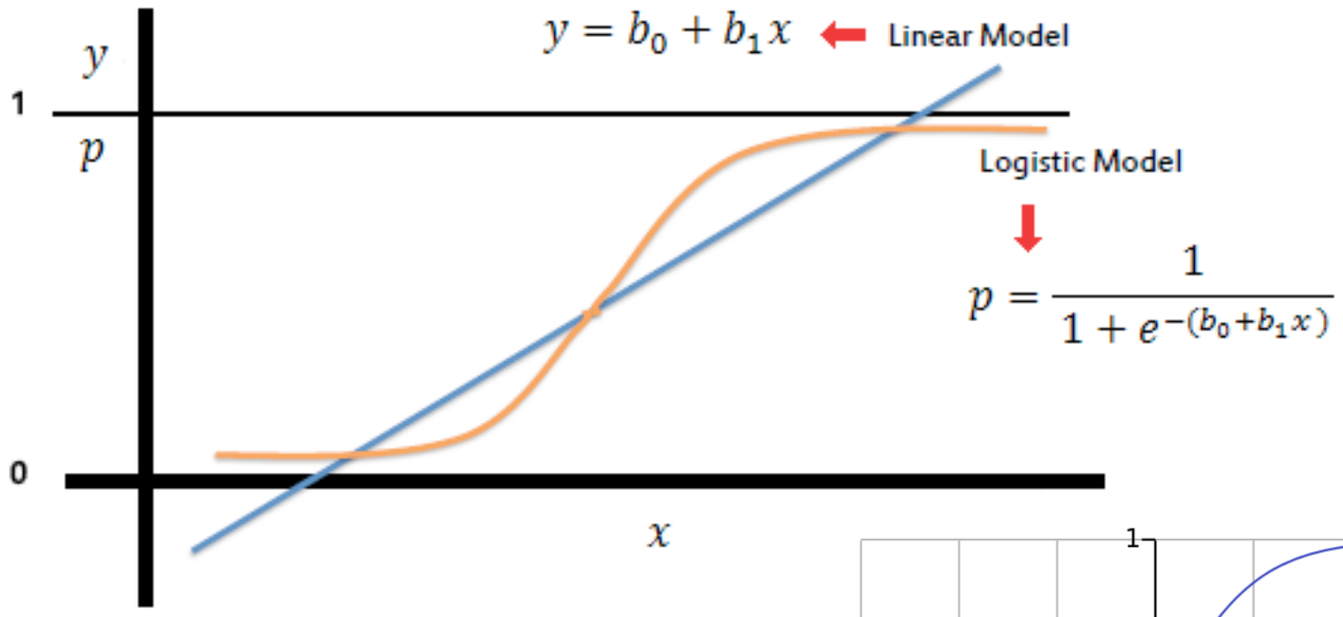
Romeo Kienzler, IBM Watson IoT


Hidden
Input
Output

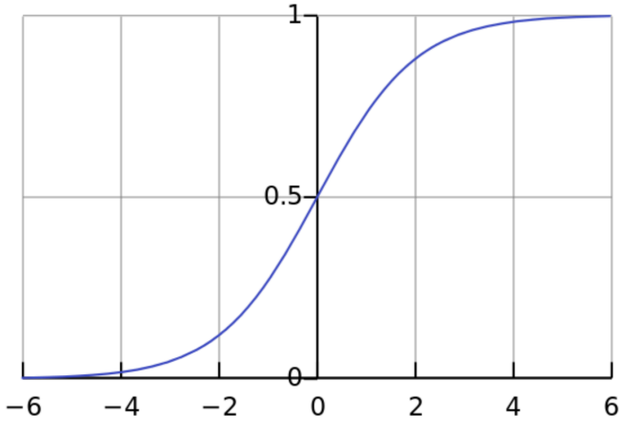

weights
inputs
x_1
w_{1j}
x_2
w_{2j}
x_3
w_{3j}
x_n
w_{nj}
$\sum$
transfer
function
net input
net_j
activation
functon
φ
o_j
activation
θ_j
threshold



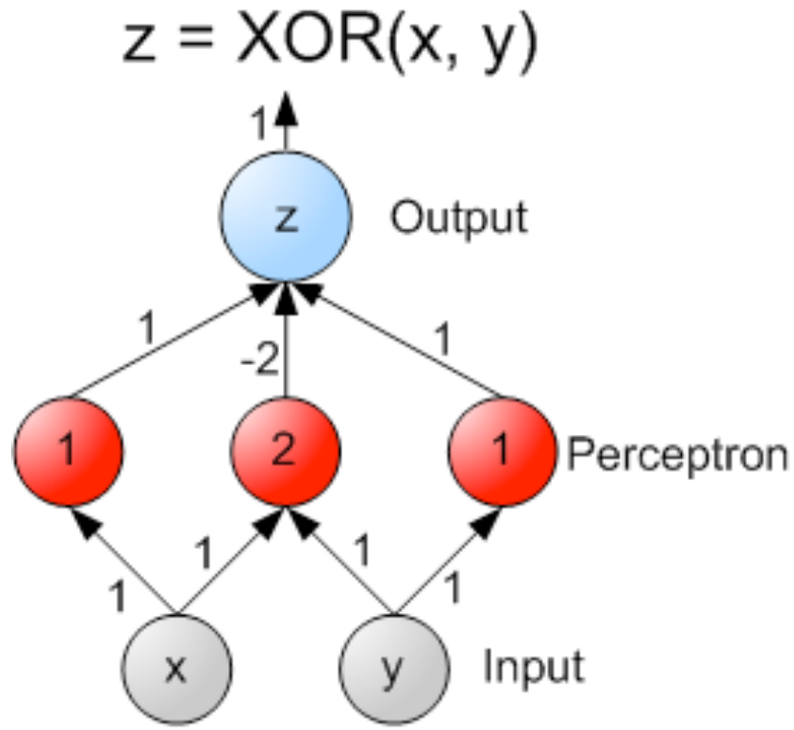
$$y_i = \beta_1 x_{i1} + \cdots + \beta_p x_{ip} + \varepsilon_i = \mathbf{x}_i^{\mathrm{T}} \boldsymbol{\beta} + \varepsilon_i, \qquad i = 1, \ldots, n$$


$y = b_0 + b_1 x$
Linear Model
y
1
p
Logistic Model
$p = \frac{1}{1 + e^{-(b_0 + b_1 x)}}$
0
x

$$S(t) = \frac{1}{1 + e^{-t}}$$


1
0.5
0
−6
−4
−2
0
2
4
6

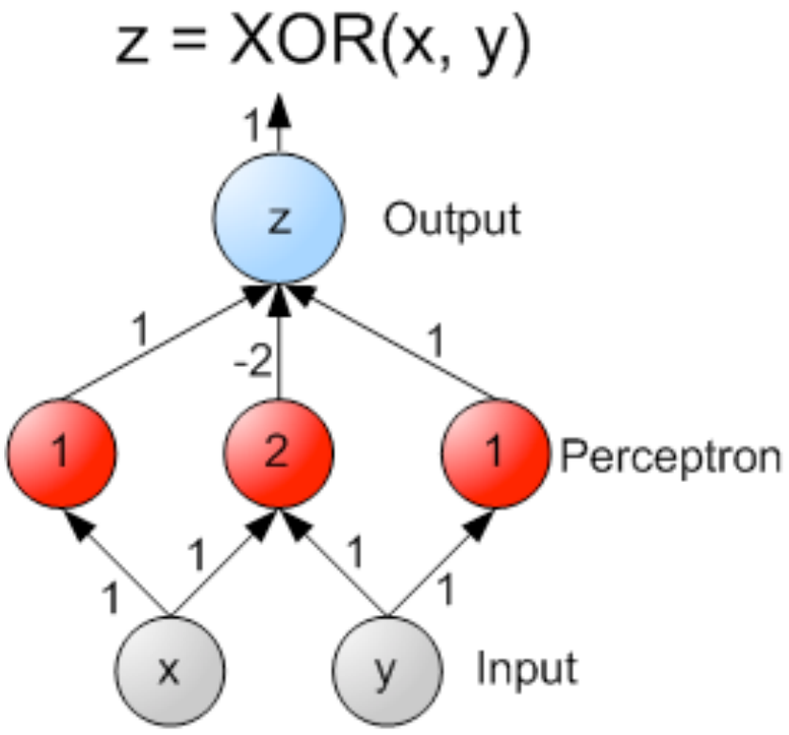
the forward pass



$$o = f(\sum_{k=1}^{n} i_k W_k)$$

 Romeo Kienzler

back propagation



$$z^{(2)} = XW^{(1)} \quad (1)$$

$$a^{(2)} = f(z^{(2)}) \quad (2)$$

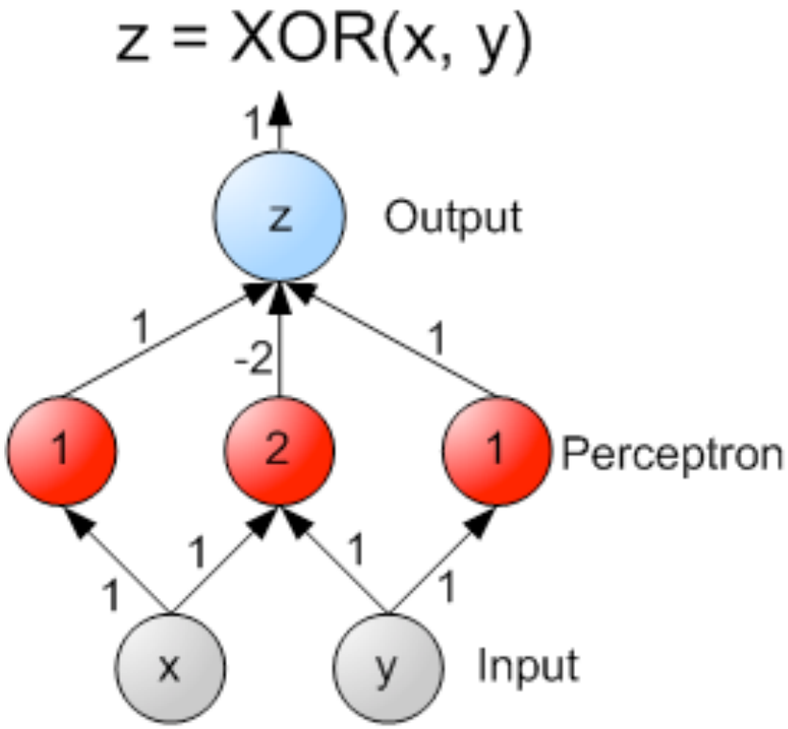
$$z^{(3)} = a^{(2)}W^{(2)} \quad (3)$$

$$\hat{y} = f(z^{(3)}) \quad (4)$$

$$J = \sum \frac{1}{2}(y - \hat{y})^2 \quad (5)$$

 Romeo Kienzler

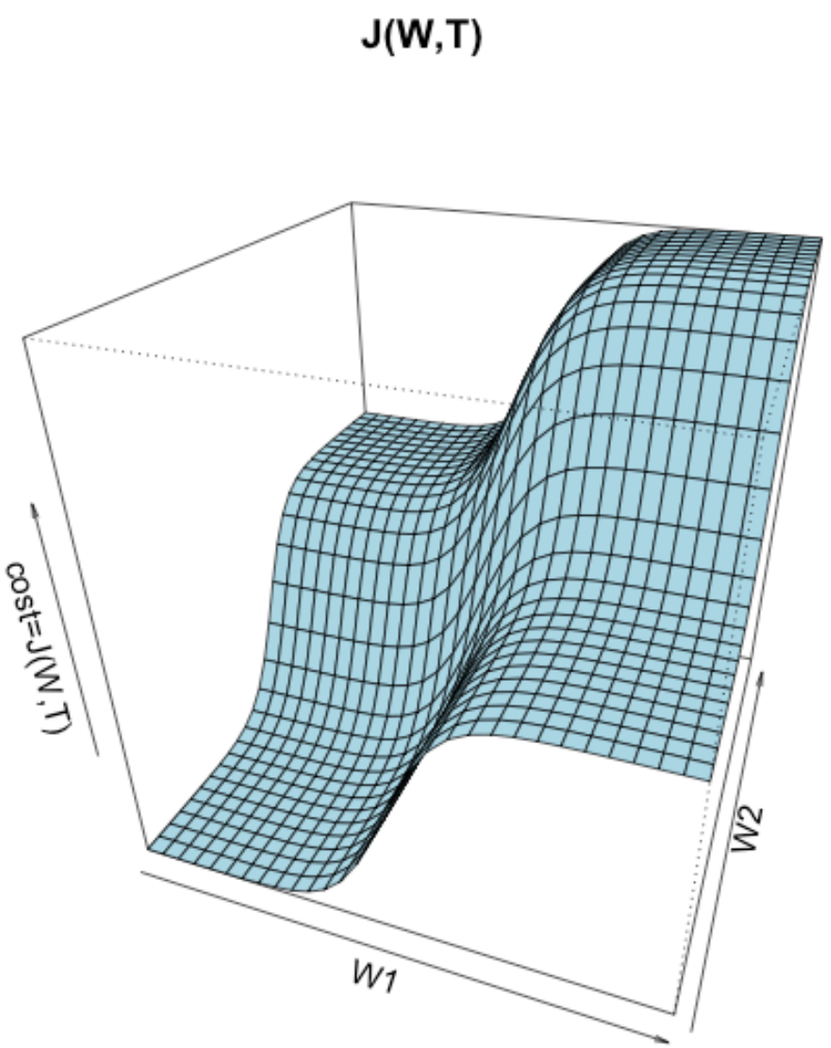
back propagation



$$J = \sum \frac{1}{2}(y - f(\, f(\, XW^{(1)}\,)\, W^{(2)}\,)\,)^2$$

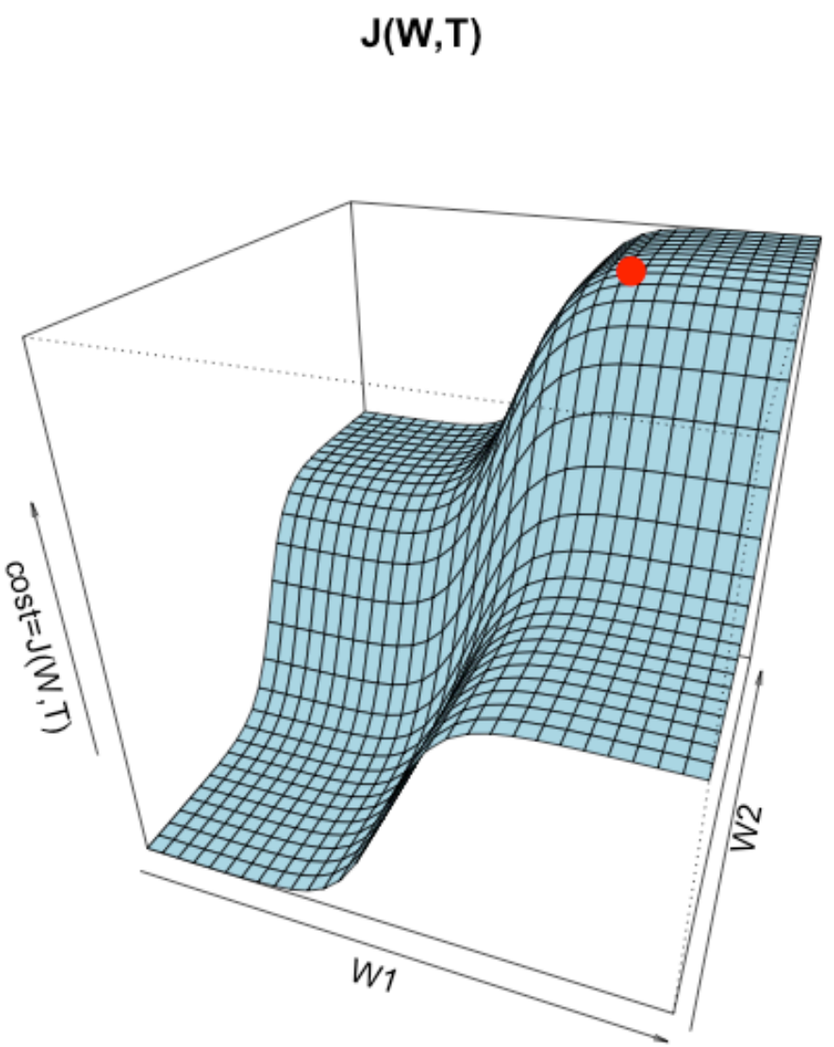
 Romeo Kienzler

gradient descent



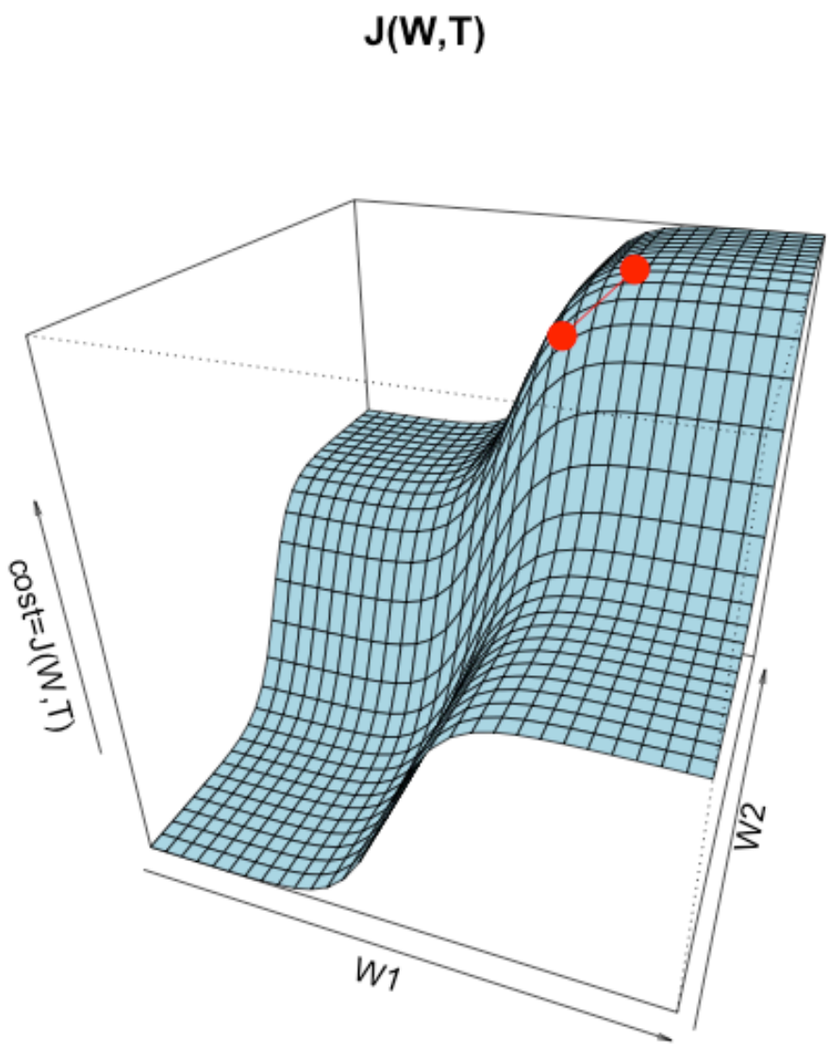
 Romeo Kienzler

gradient descent



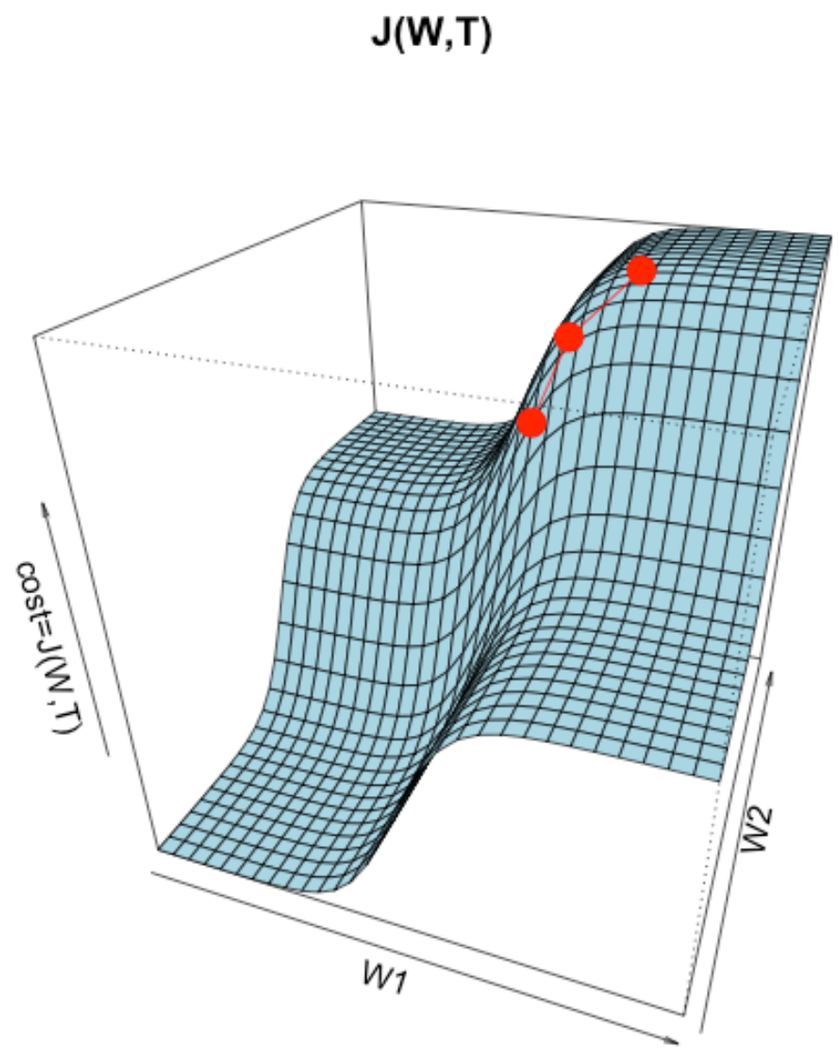
 Romeo Kienzler

gradient descent



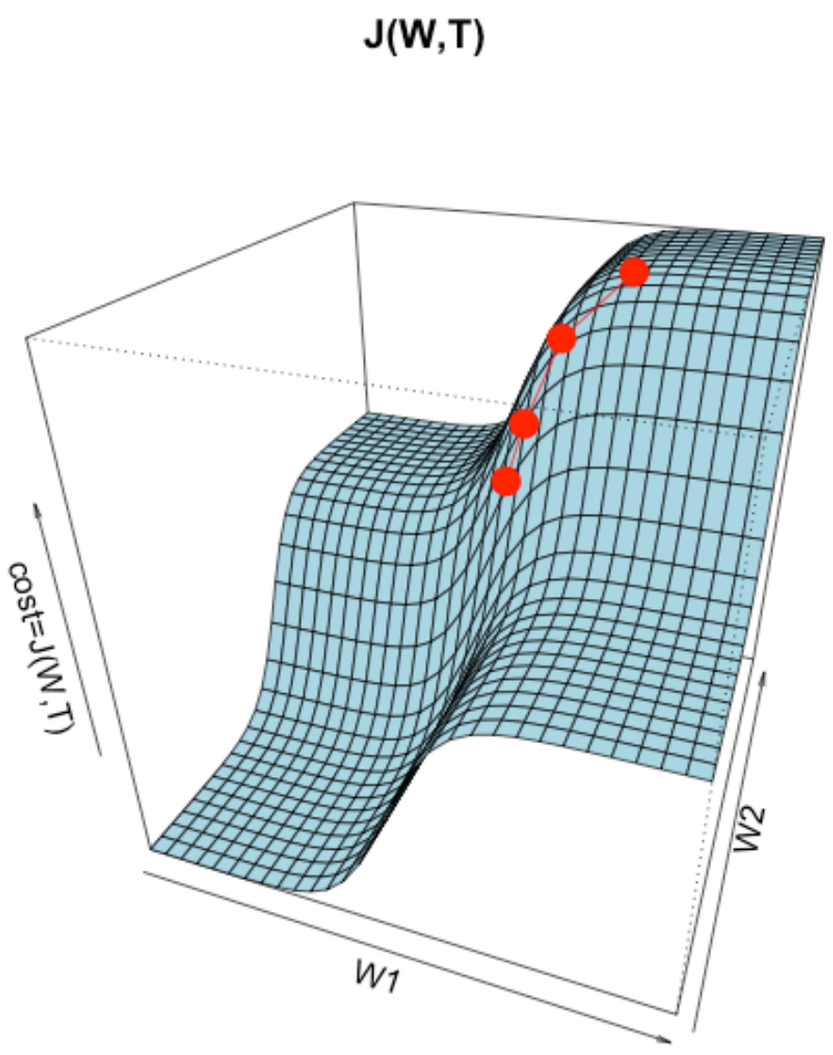
 Romeo Kienzler

gradient descent



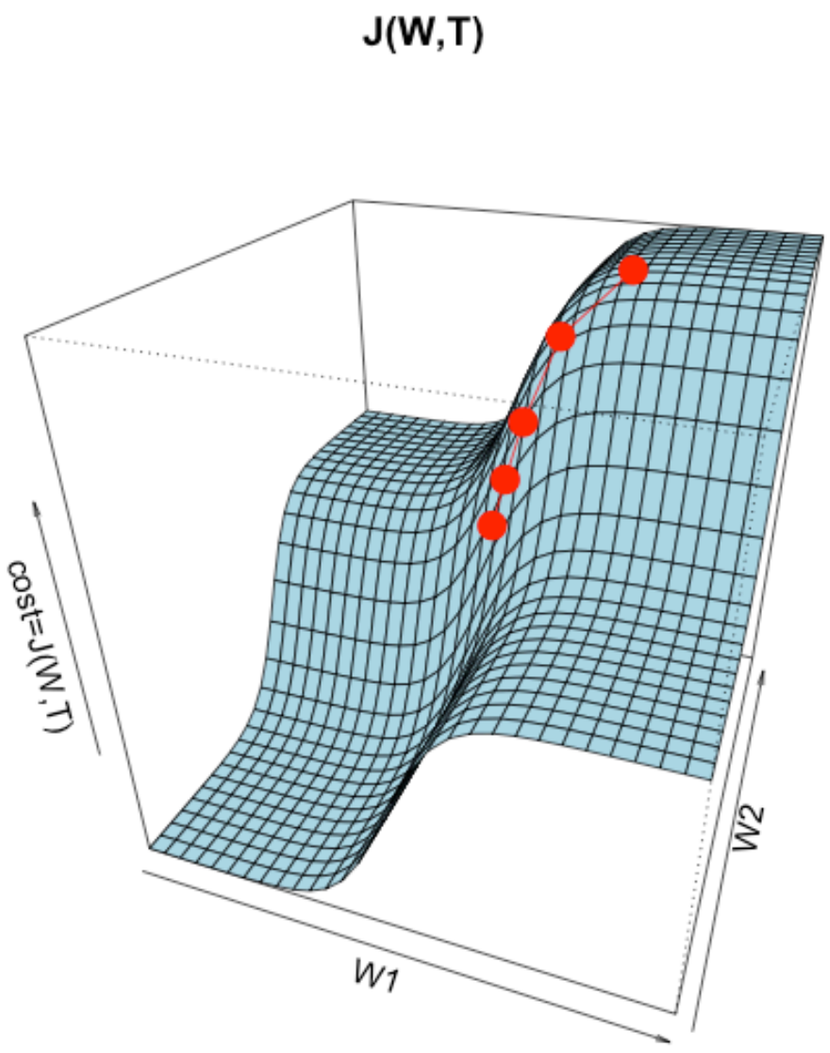
 Romeo Kienzler

gradient descent



 Romeo Kienzler

gradient descent



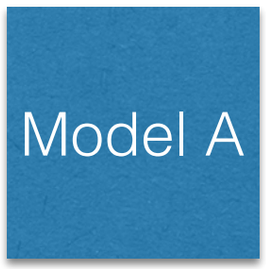
 Romeo Kienzler

parallelisation

- inter-model parallelism
- data parallelism
- intra-model parallelism
- pipelined parallelism

inter-model parallelism

aka. hyper parameter space exploration / tuning

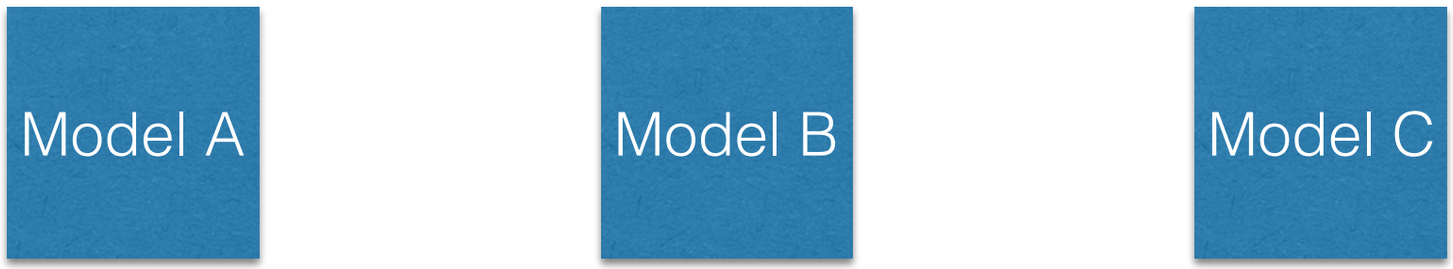


inter-model parallelism

aka. hyper parameter space exploration / tuning

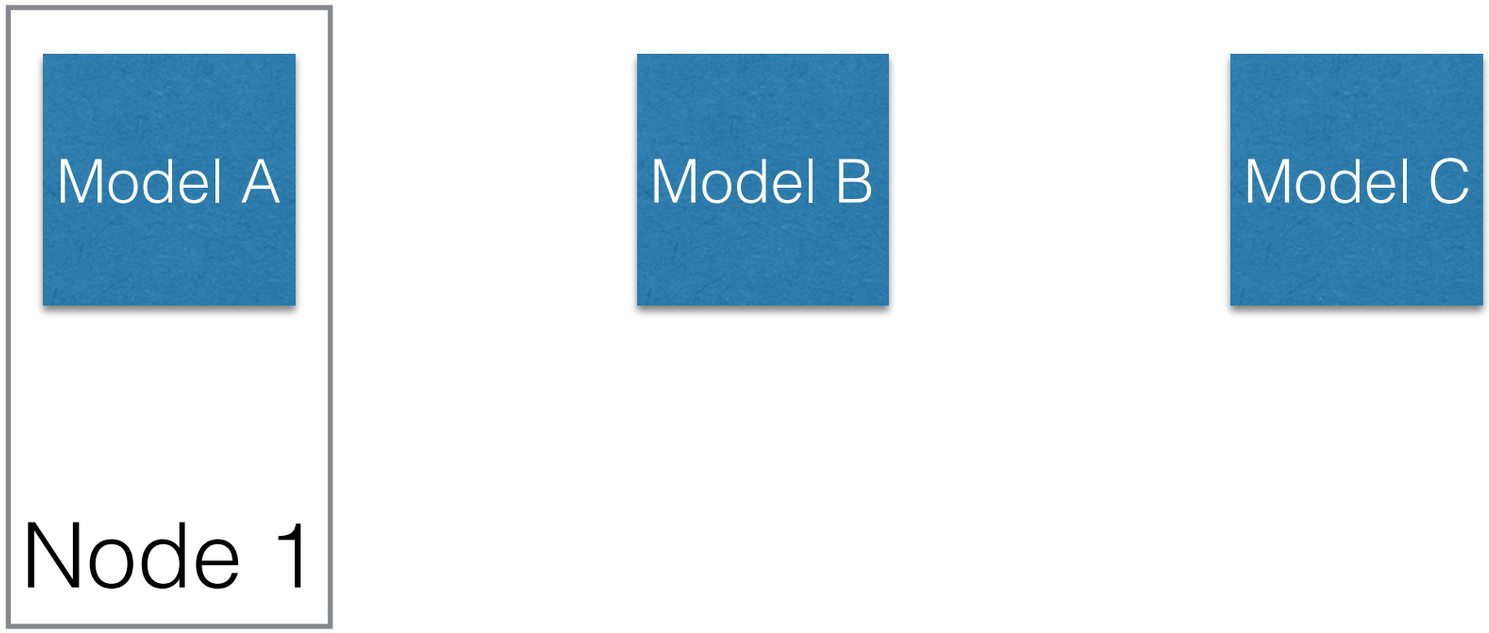
inter-model parallelism

aka. hyper parameter space exploration / tuning



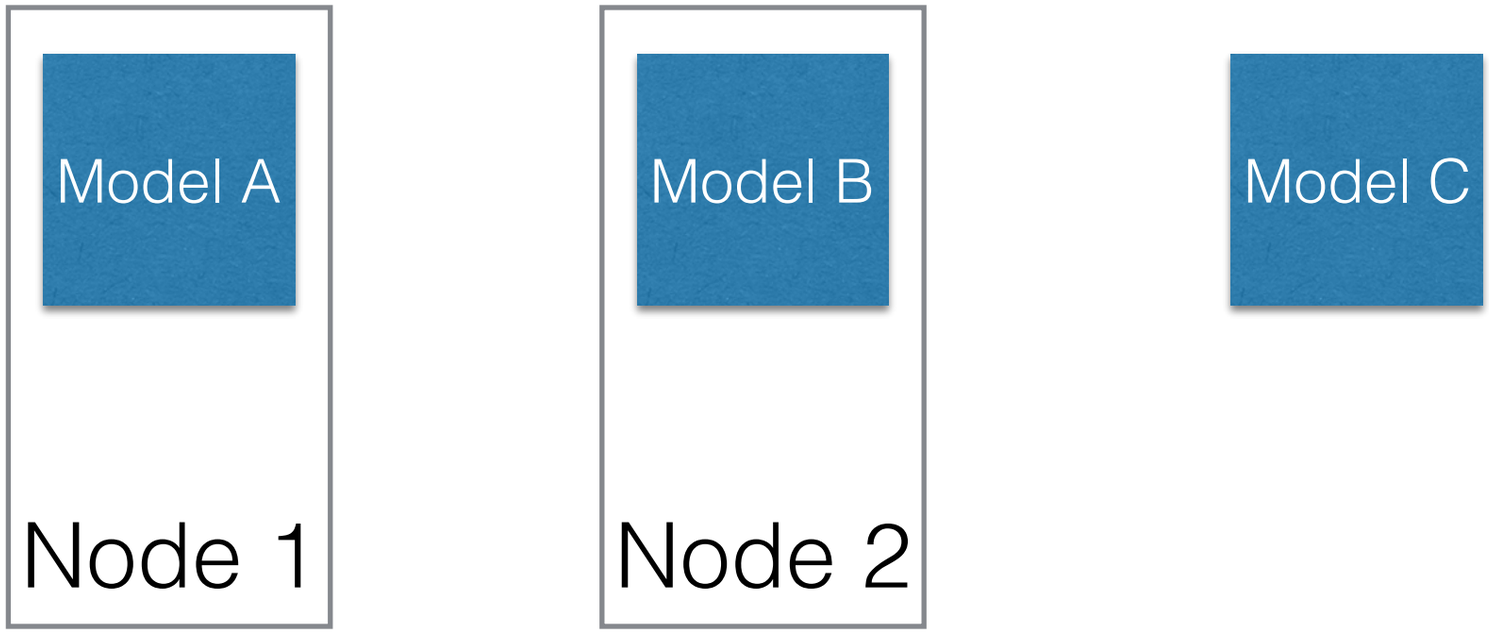
inter-model parallelism

aka. hyper parameter space exploration / tuning



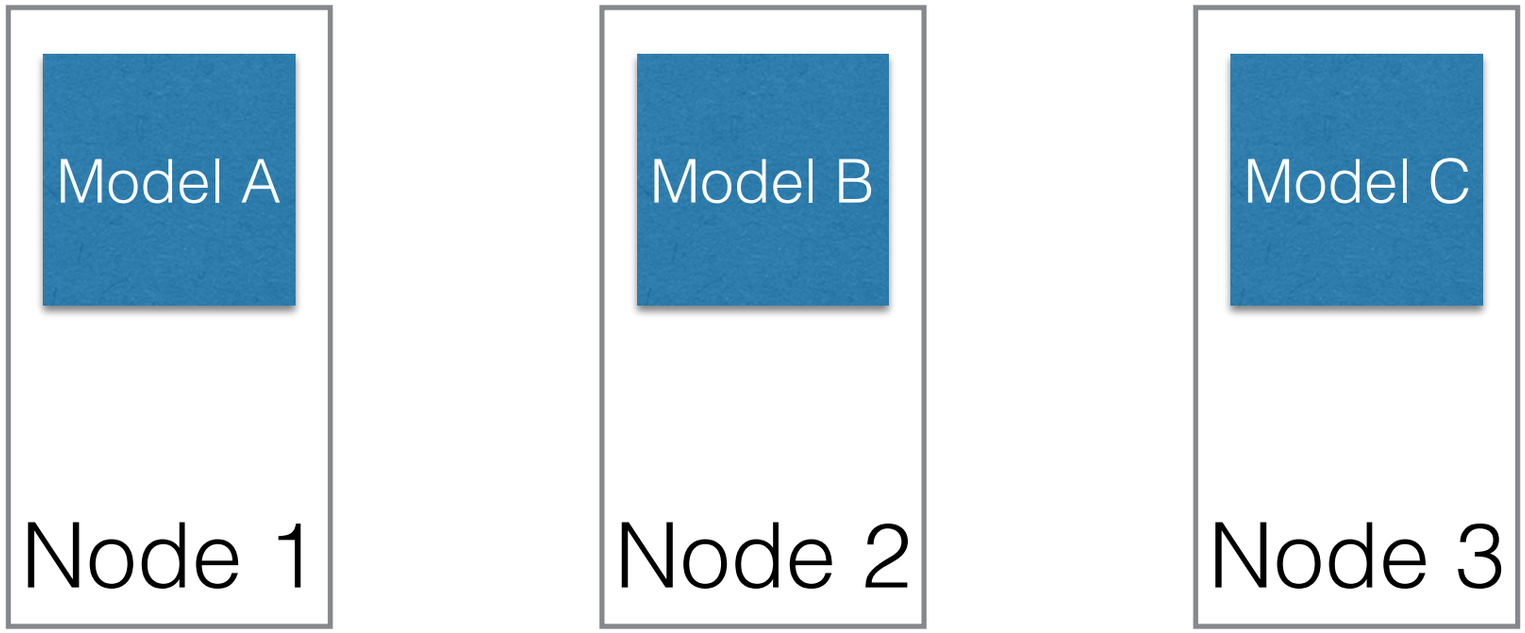
inter-model parallelism

aka. hyper parameter space exploration / tuning



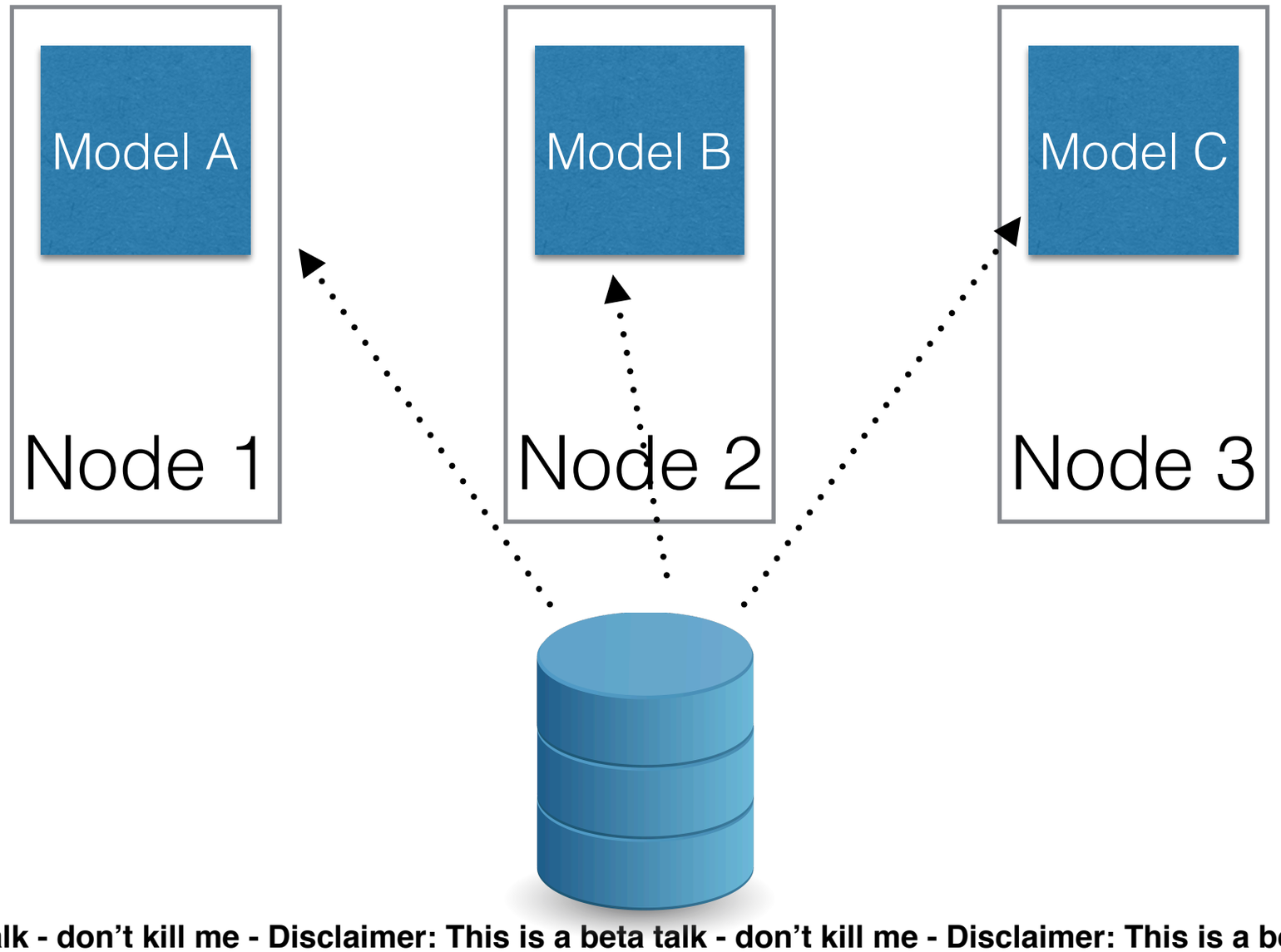
inter-model parallelism

aka. hyper parameter space exploration / tuning



inter-model parallelism

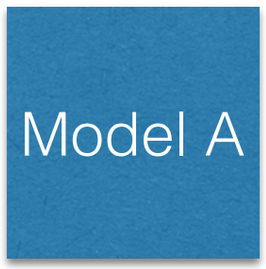
aka. hyper parameter space exploration / tuning



Romeo Kienzler

data parallelism

aka. “Jeff Dean style” parameter averaging

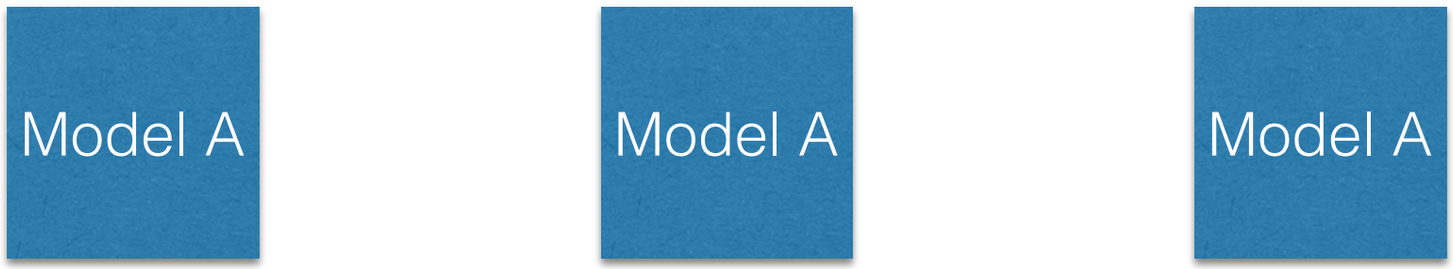


data parallelism

aka. “Jeff Dean style” parameter averaging

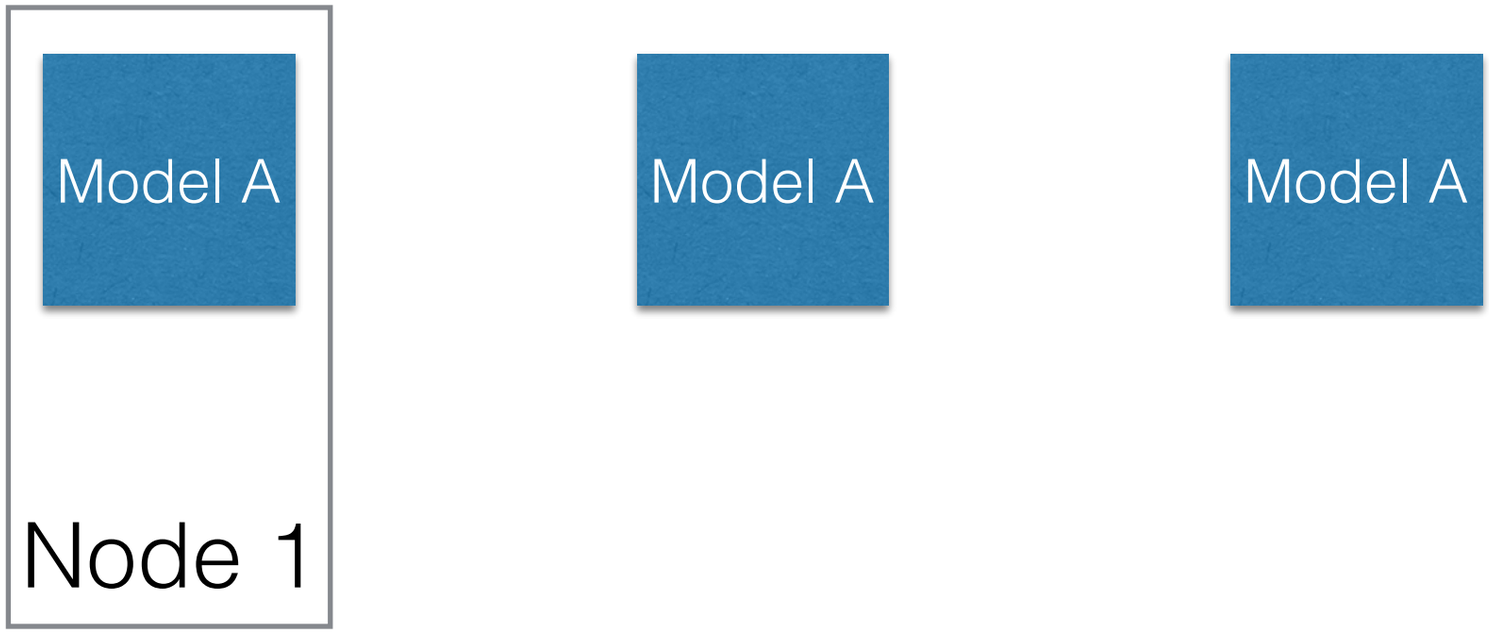
data parallelism

aka. “Jeff Dean style” parameter averaging



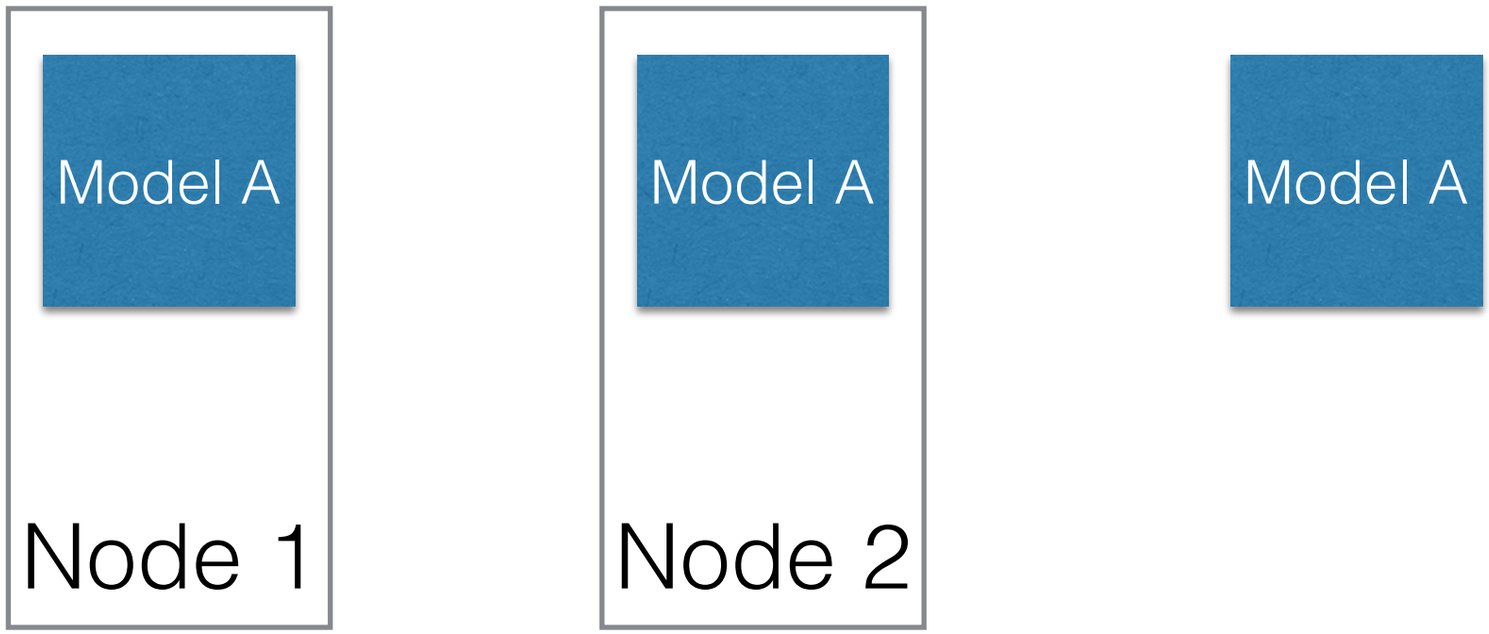
data parallelism

aka. “Jeff Dean style” parameter averaging



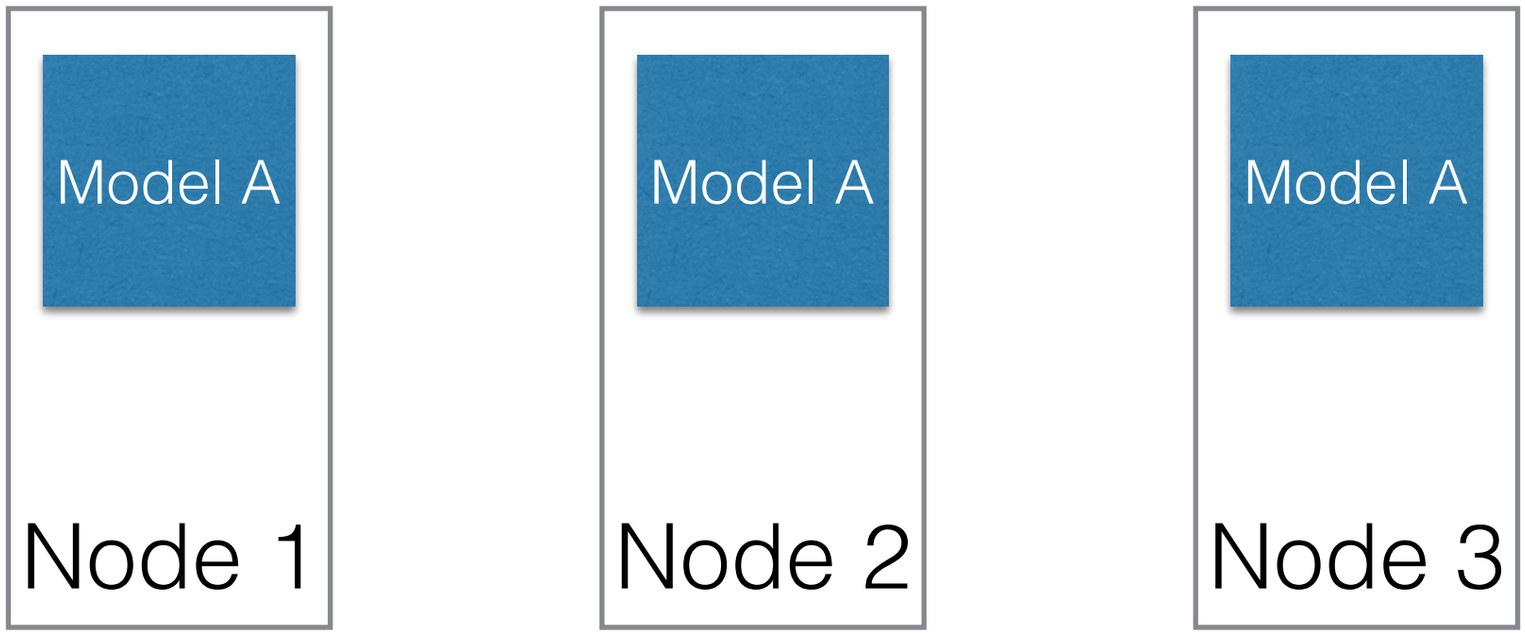
data parallelism

aka. “Jeff Dean style” parameter averaging



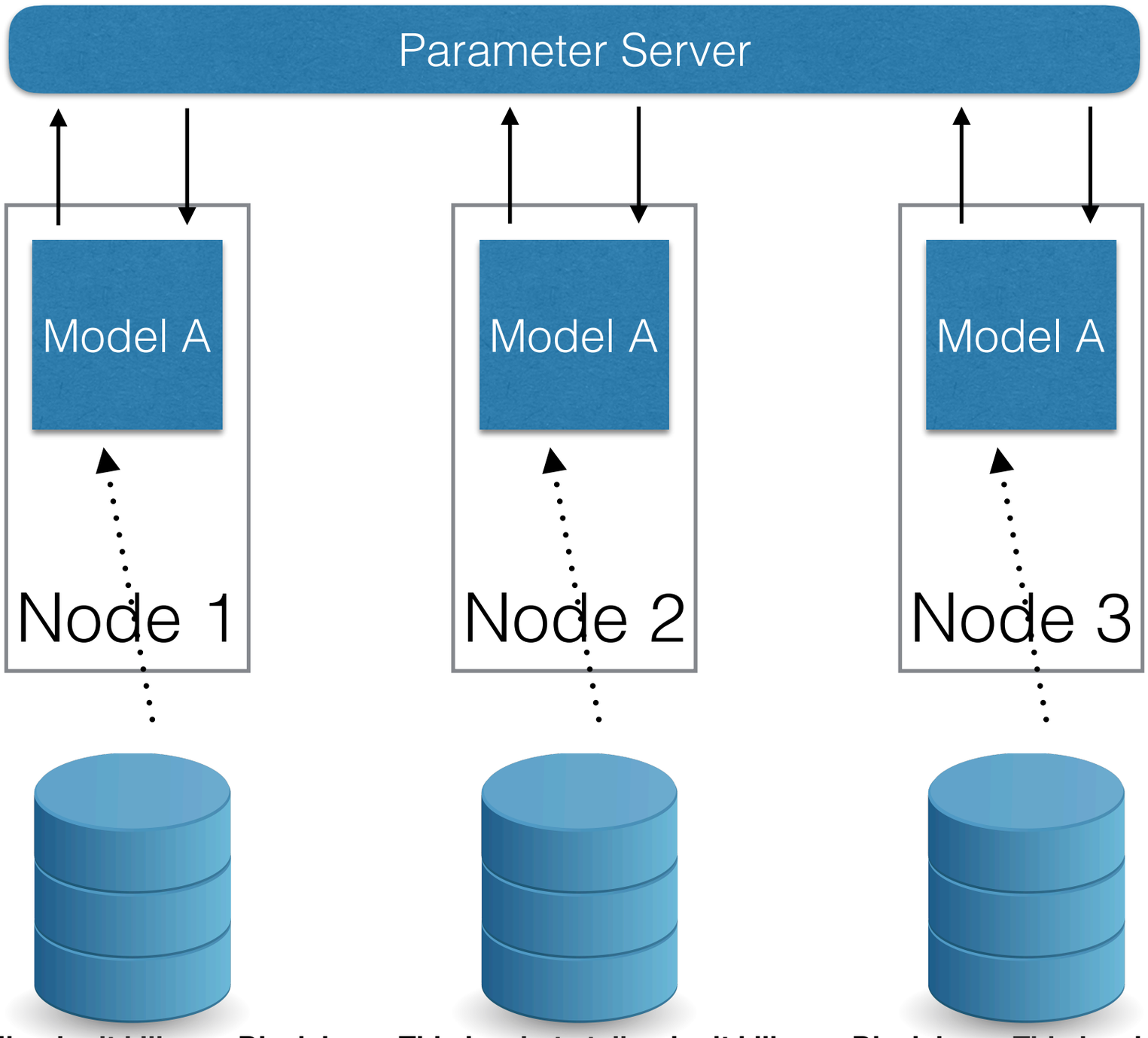
data parallelism

aka. “Jeff Dean style” parameter averaging

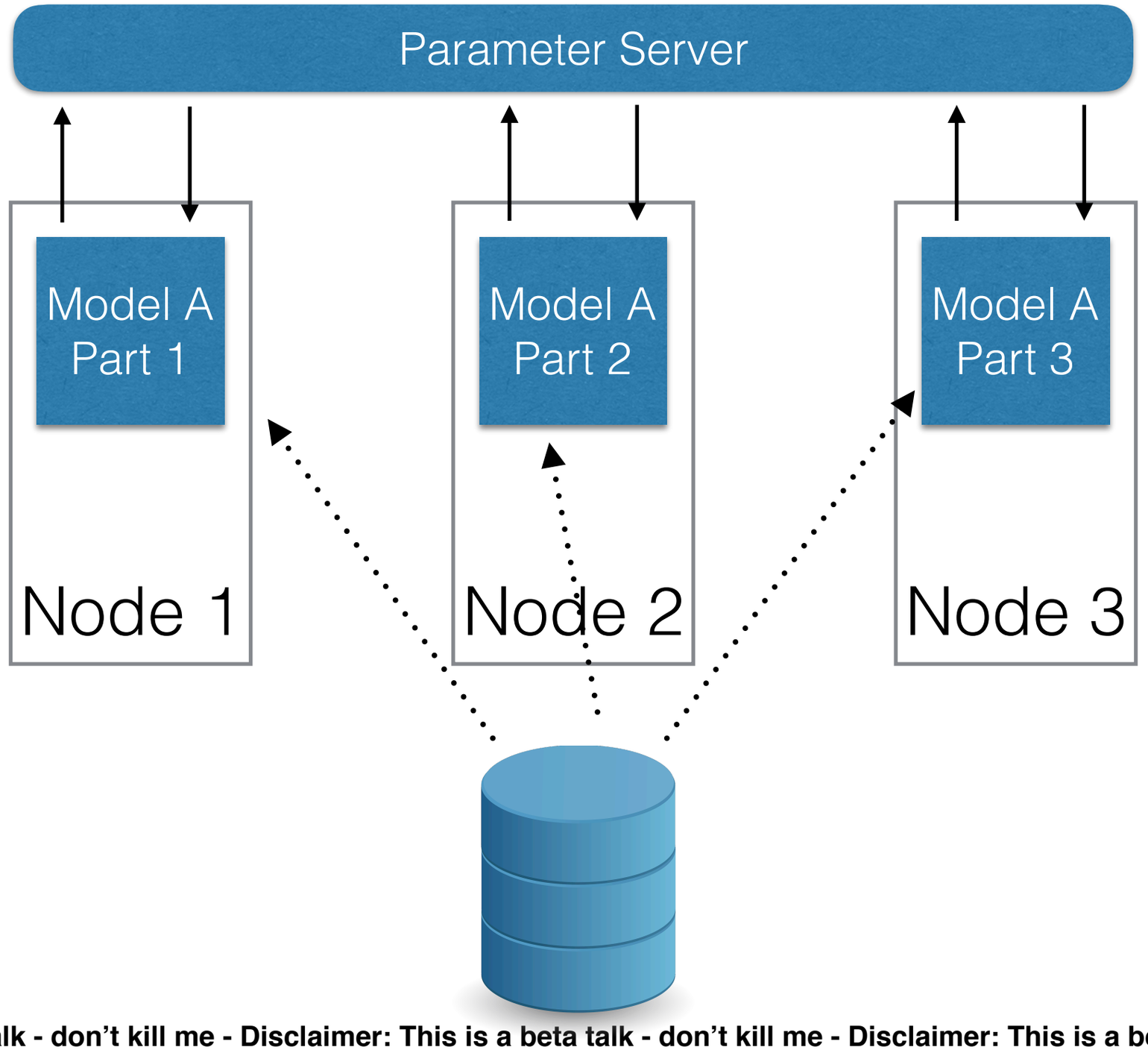


data parallelism

aka. “Jeff Dean style” parameter averaging

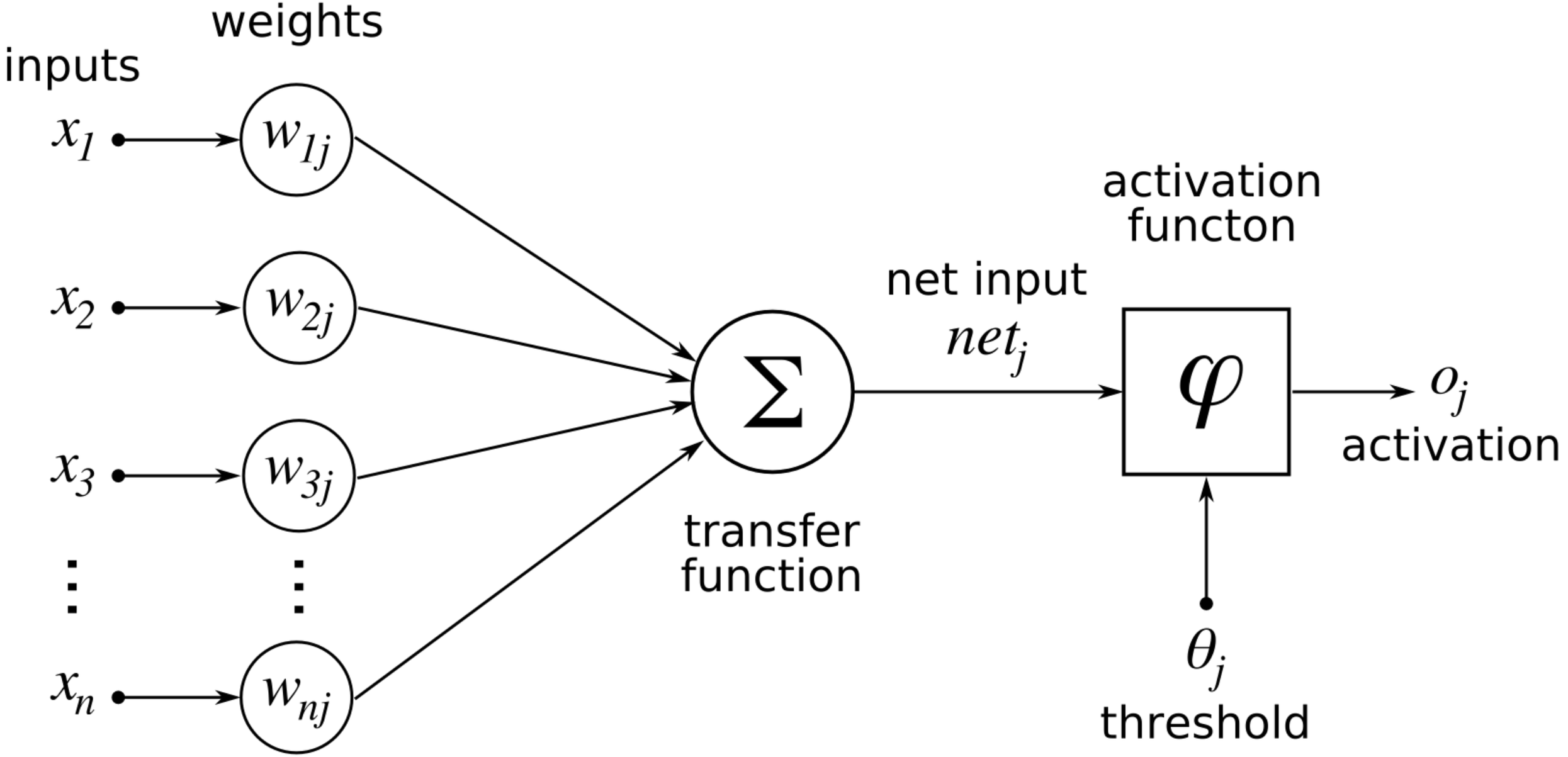


intra-model parallelism



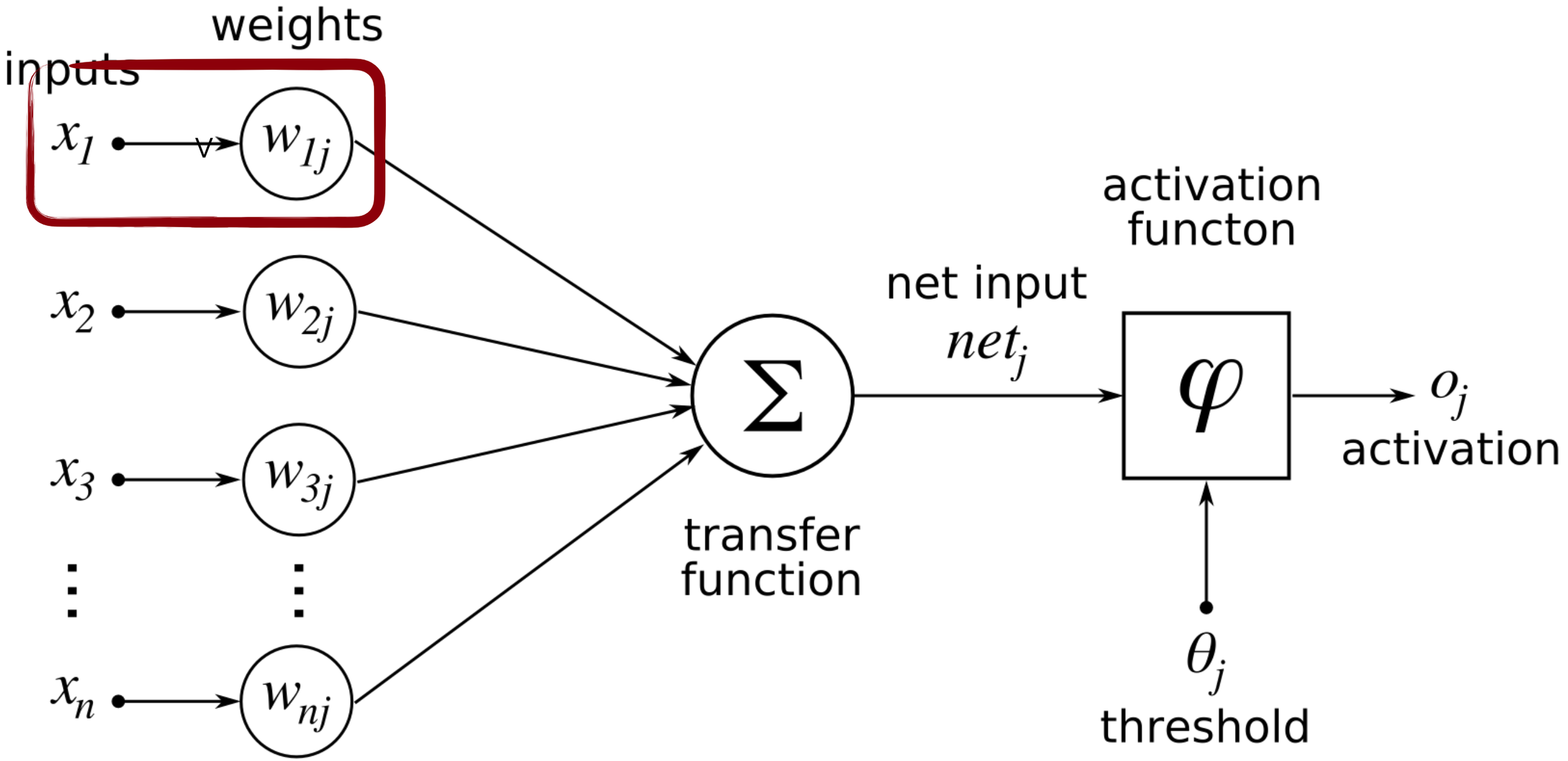
 Romeo Kienzler

intra-model parallelism



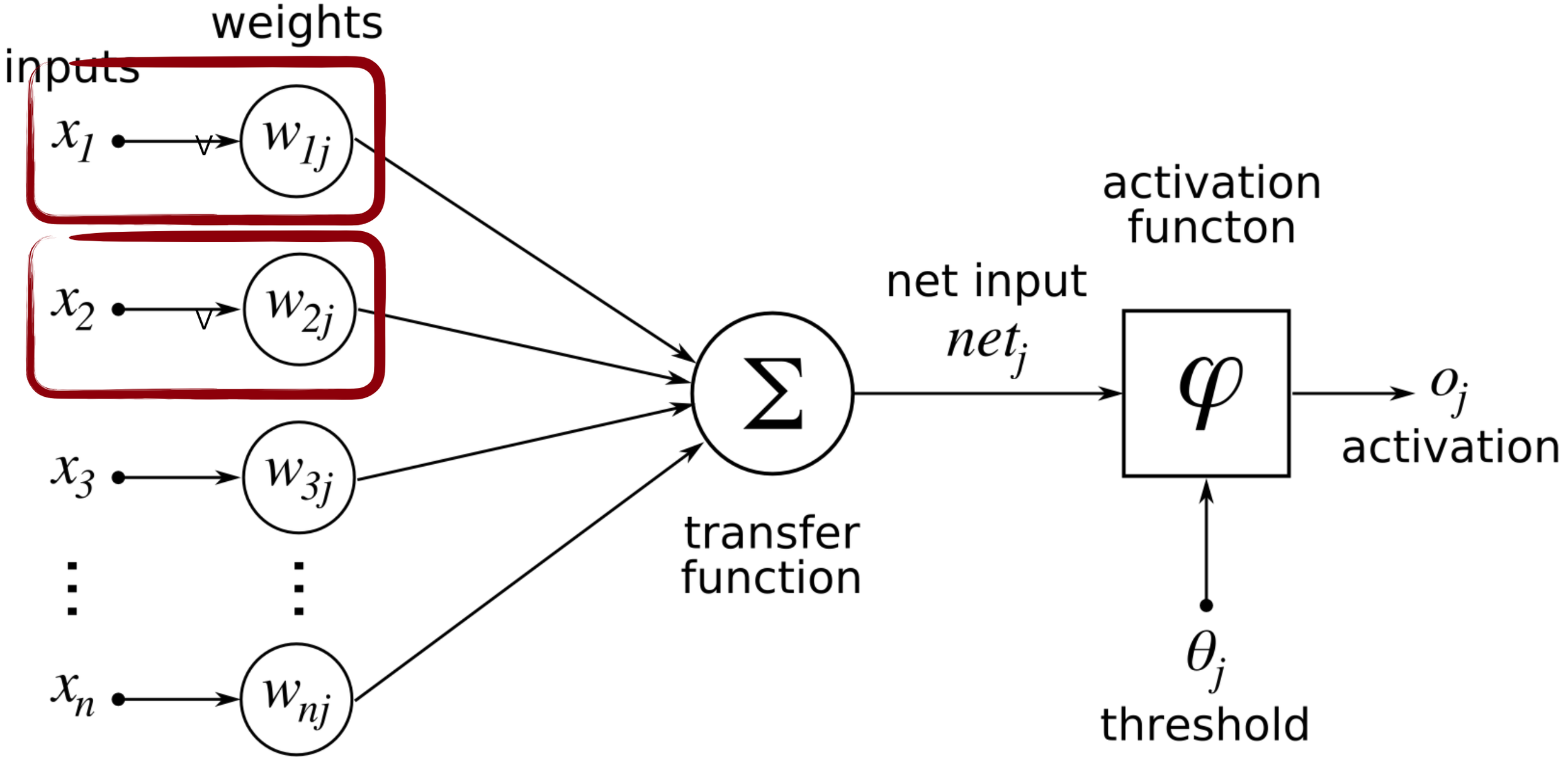
 Romeo Kienzler

intra-model parallelism



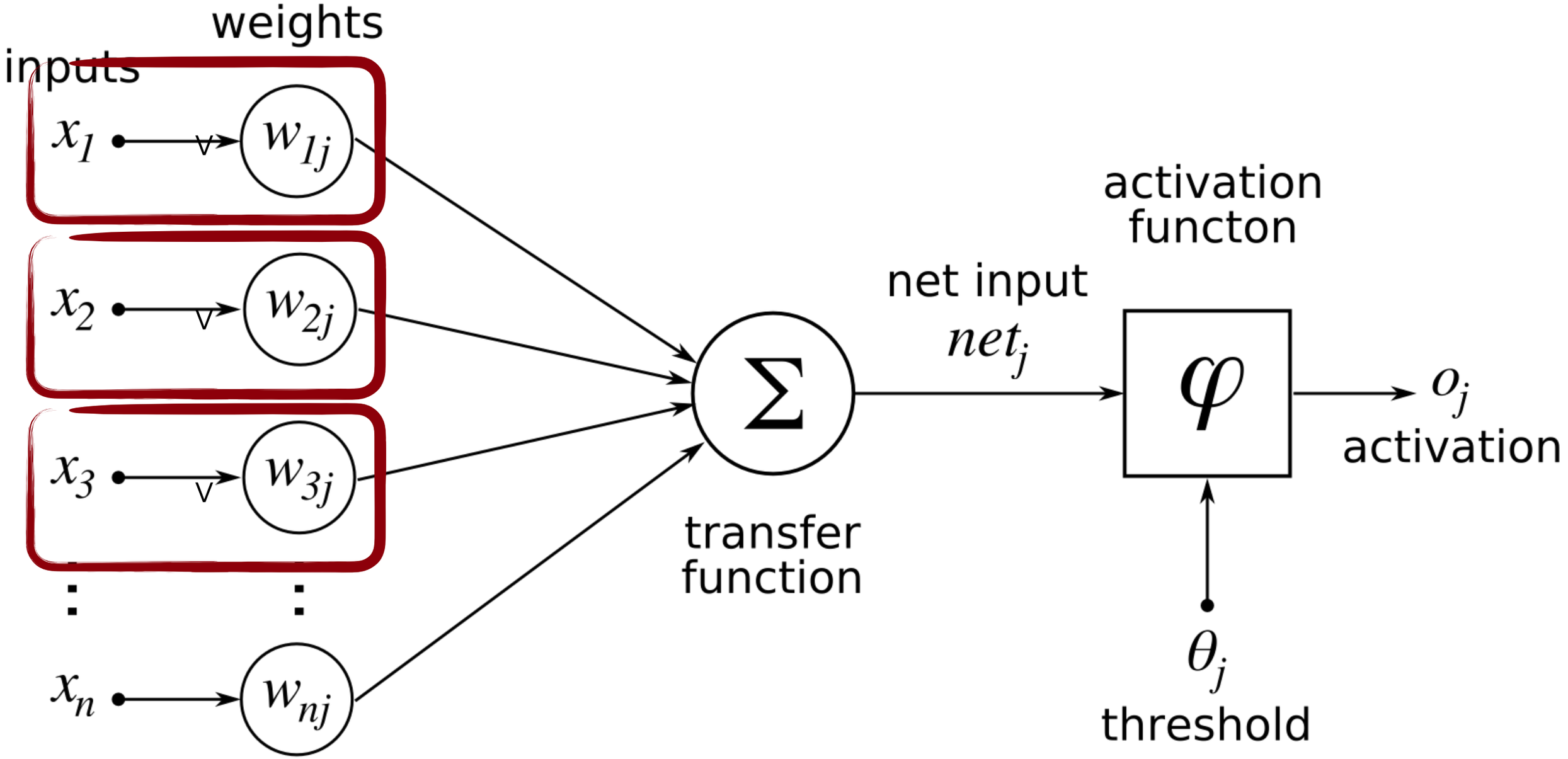
 Romeo Kienzler

intra-model parallelism



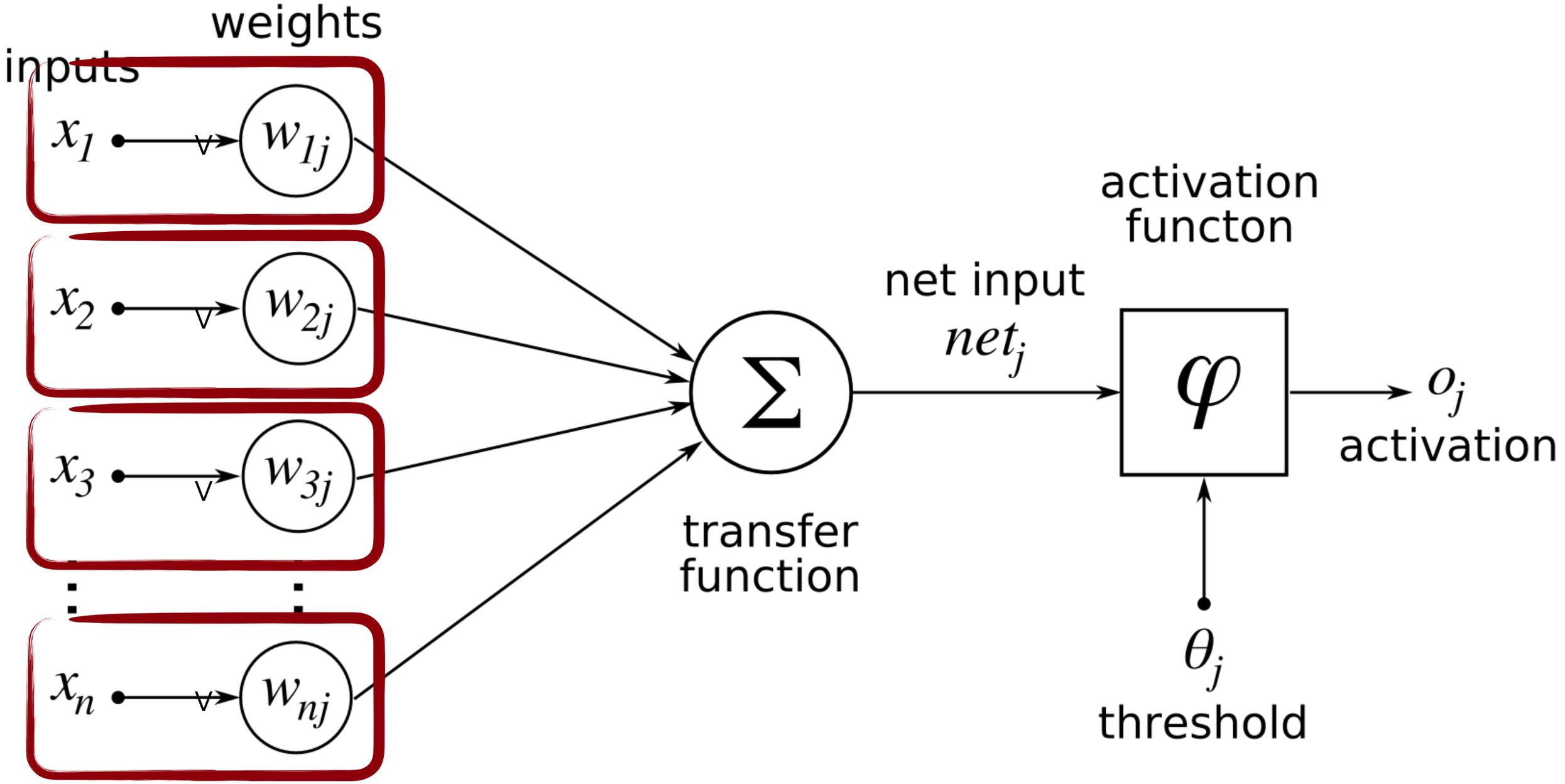
 Romeo Kienzler

intra-model parallelism



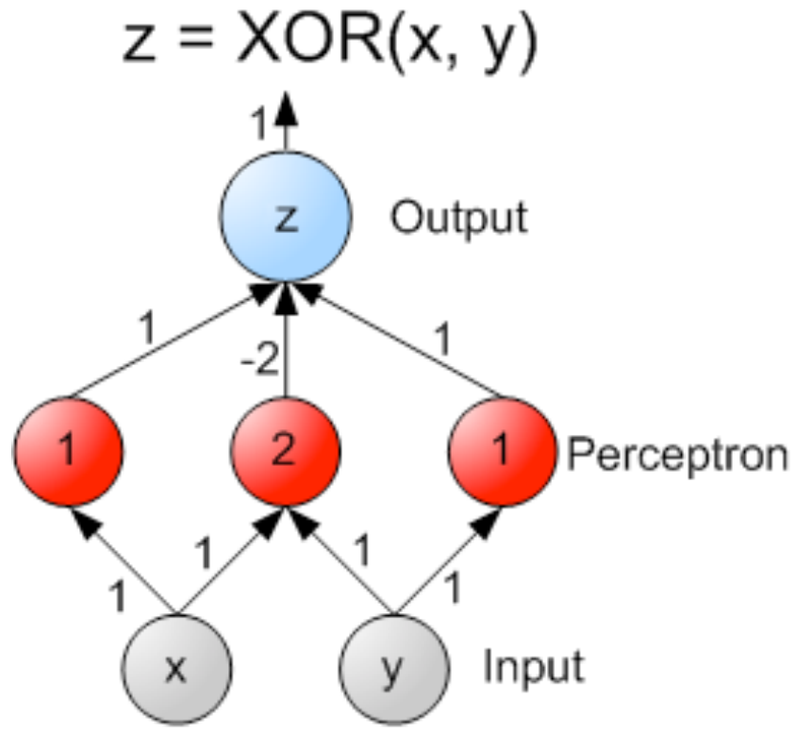
 Romeo Kienzler

intra-model parallelism



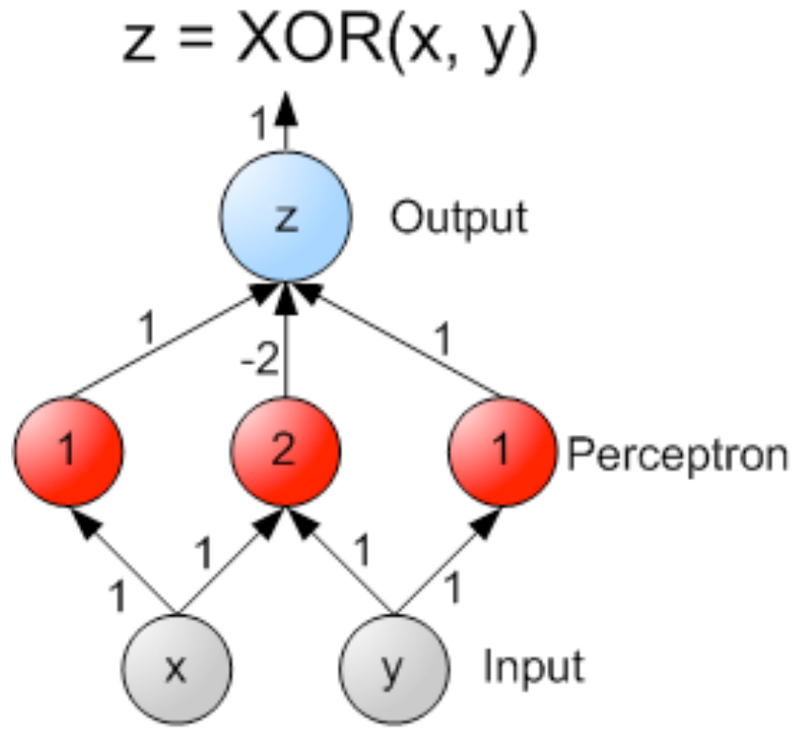
 Romeo Kienzler

pipelined parallelism



 Romeo Kienzler

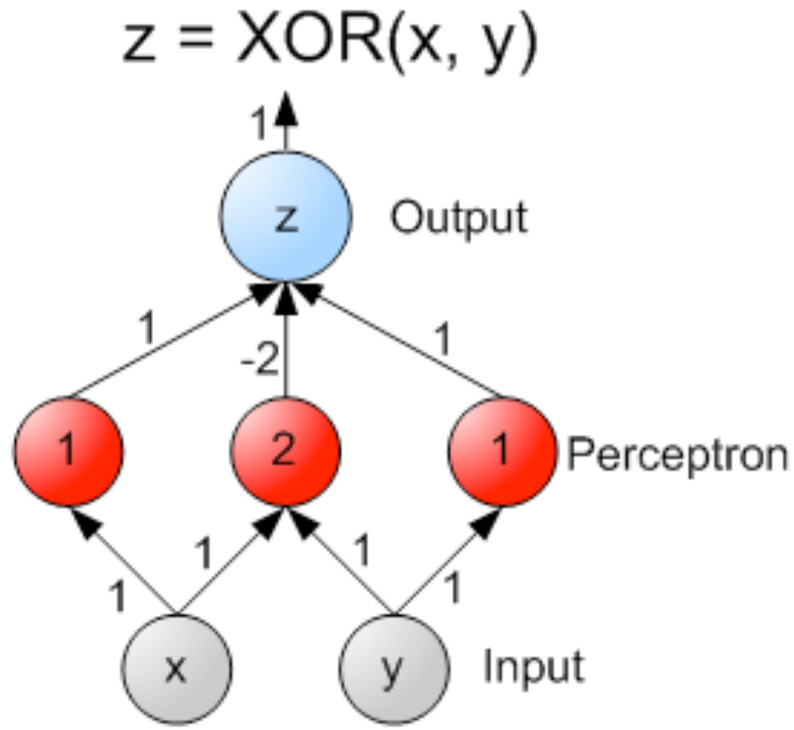
pipelined parallelism



0 0
0 1
1 0
1 1

 Romeo Kienzler

pipelined parallelism



0 0

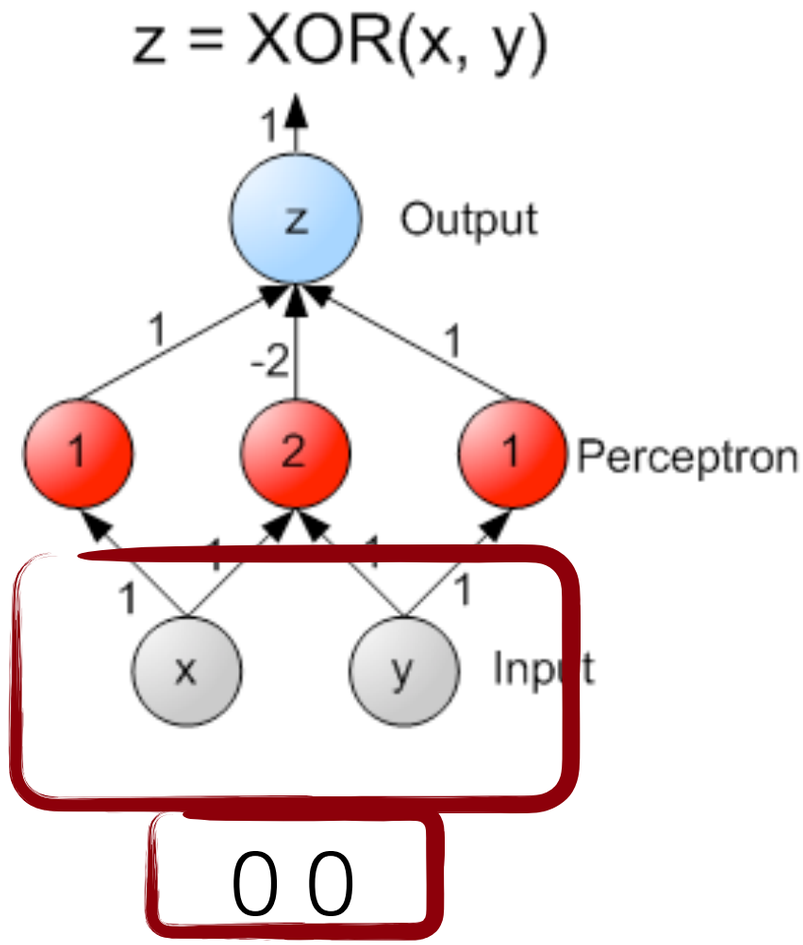
0 1

1 0

1 1

 Romeo Kienzler

pipelined parallelism



0 0

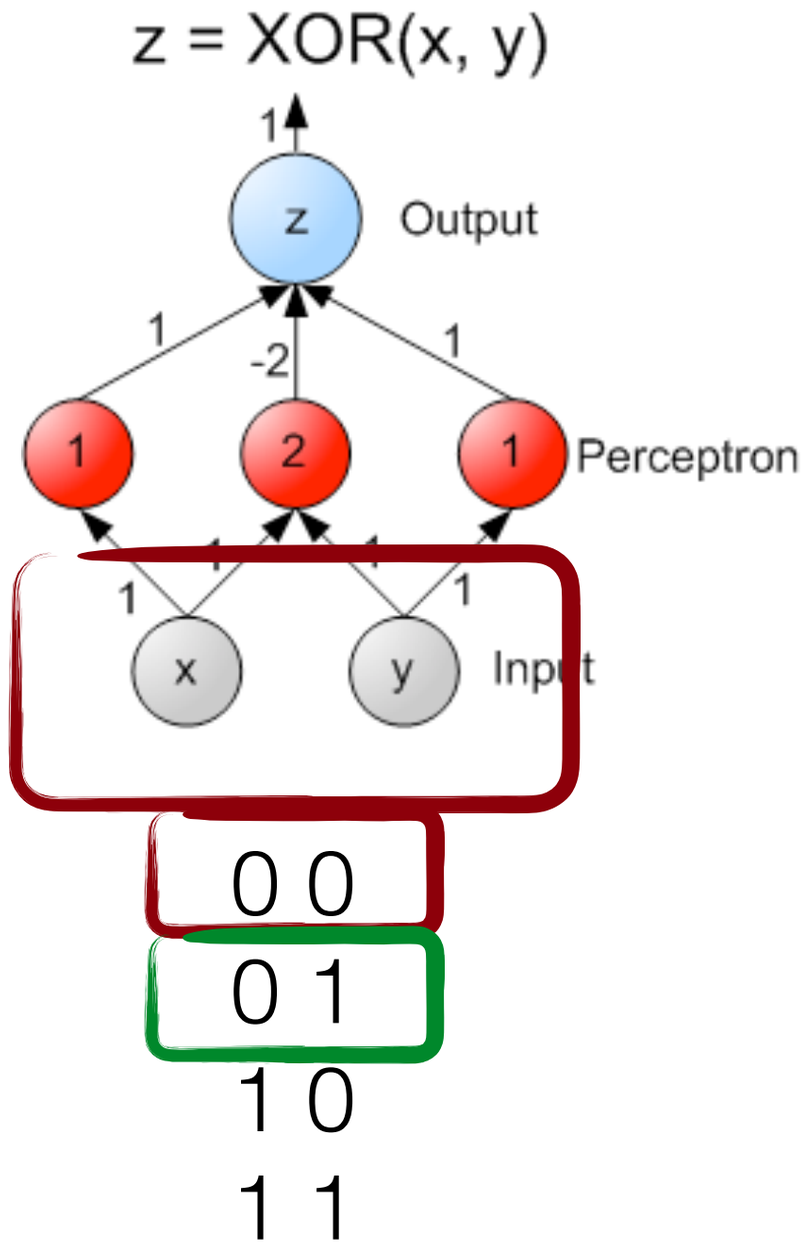
0 1

1 0

1 1

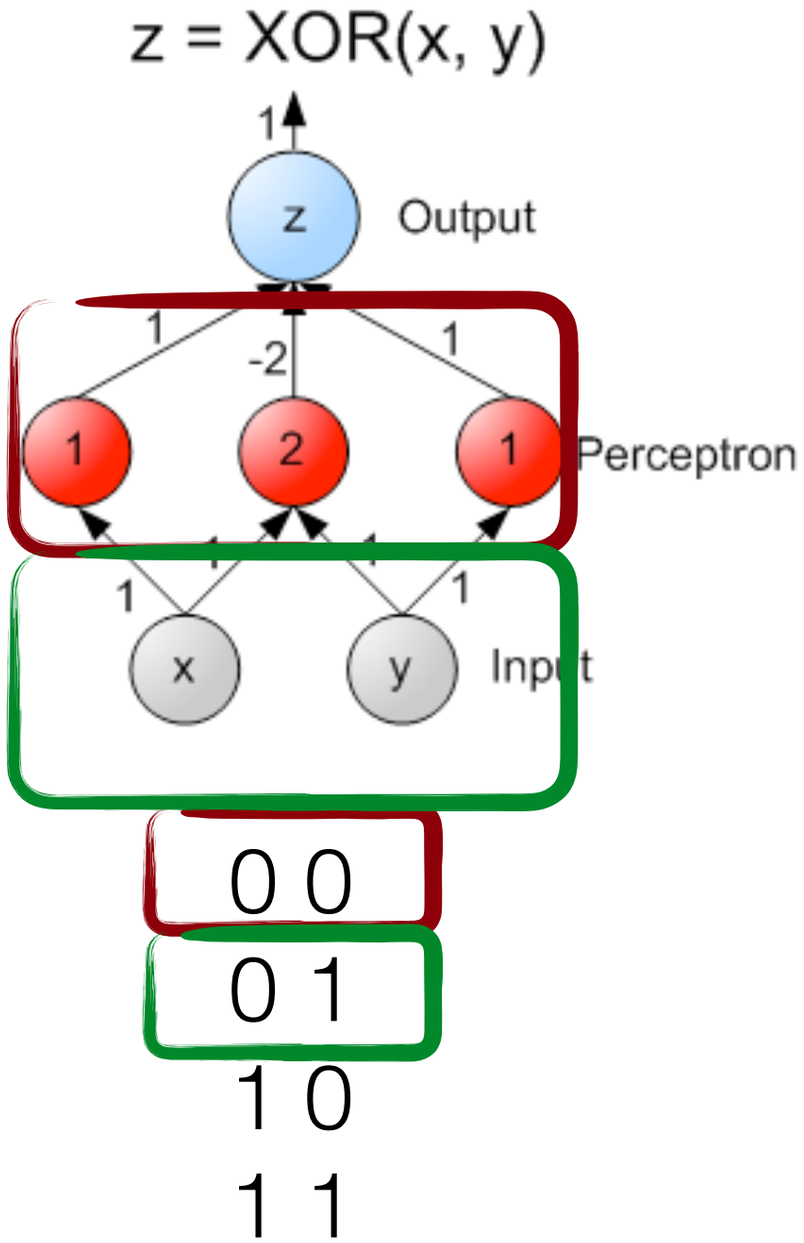
 Romeo Kienzler

pipelined parallelism



 Romeo Kienzler

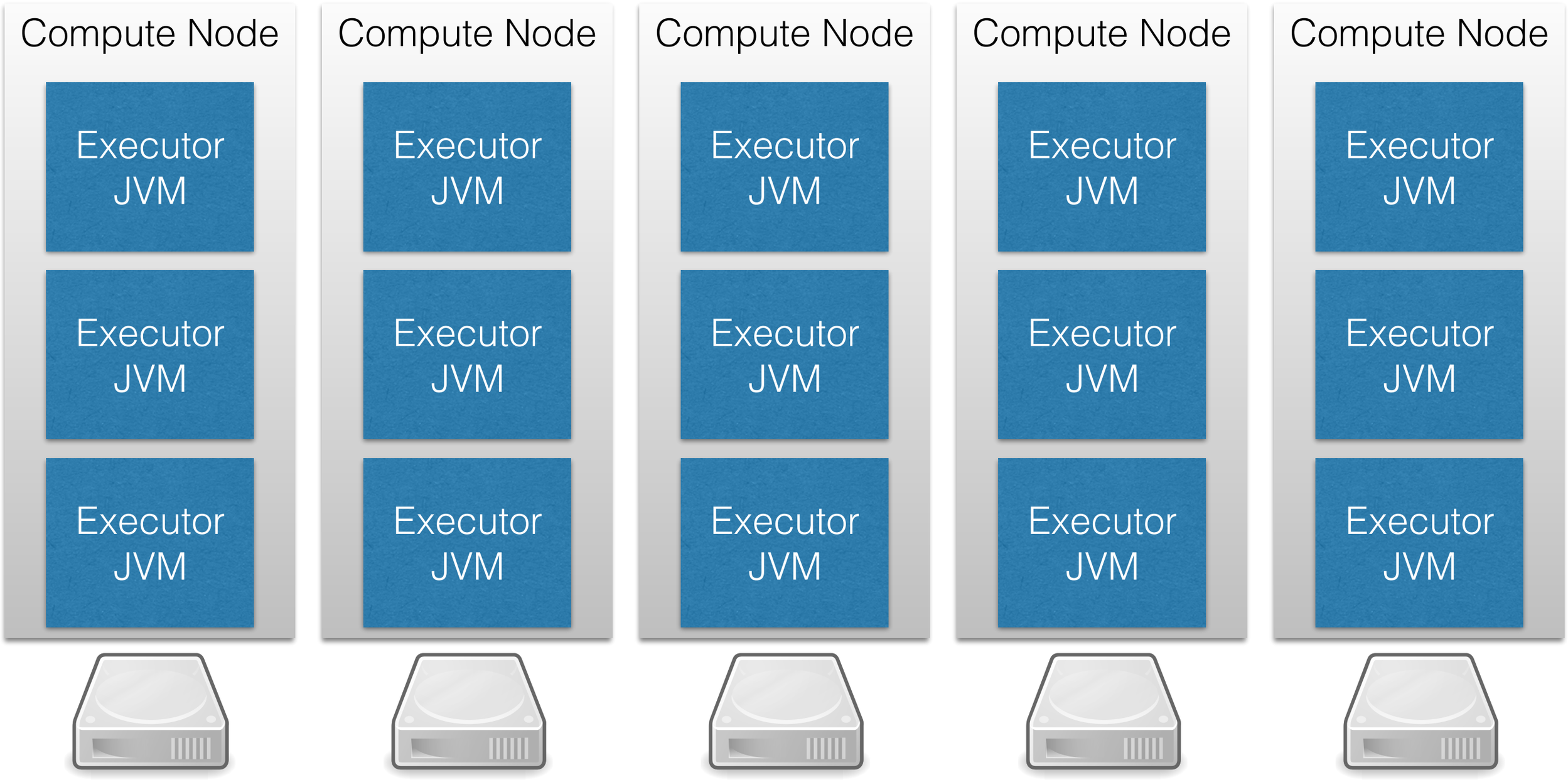
pipelined parallelism



 Romeo Kienzler

Driver JVM

Apache Spark



DeepLearning4J

```
new MultiLayerNetwork(conf);
```

VS.

```
new SparkDl4jMultiLayer(sc, conf, tm);
```

```
TrainingMaster tm = new ParameterAveragingTrainingMaster.Builder(int dataSetObjectSize)
        ... (your configuration here)
        .build();
```

DeepLearning4J

```
new MultiLayerNetwork(conf);
```

VS.

```
new SparkDl4jMultiLayer(sc, conf, tm);
```

data parallelism

```
TrainingMaster tm = new ParameterAveragingTrainingMaster.Builder(int dataSetObjectSize)
    ... (your configuration here)
    .build();
```

ND4J (DeepLearning4J)

- Tensor support (Linear Buffer + Stride)
- Multiple implementations, one interface
  - vectorized c++ code (JavaCPP), off-heap data storage, BLAS (OpenBLAS, Intel MKL, cuBLAS)
  - GPU (CUDA 8)

ND4J (DeepLearning4J)

- Tensor support (Linear Buffer + Stride)
- Multiple implementations, one interface
  - vectorized c++ code (JavaCPP), off-heap data storage, BLAS (OpenBLAS, Intel MKL, cuBLAS)
  - GPU (CUDA 8)

Apache SystemML

- Custom machine learning algorithms
- Declarative ML
- Transparent distribution on data-parallel framework
  - Scale-up
  - Scale-out
- Cost-based optimiser generates low level execution plans

2007-2008: Multiple projects at IBM Research – Almaden involving machine learning on Hadoop.

2009: We form a **dedicated team** for scalable ML

2009-2010: Through engagements with customers, we observe how data scientists create **ML solutions**.

2007 2008 2009 2010

Research

2011 2012 2013 2014

June 2015: IBM Announces open-source SystemML

September 2015: Code available on Github

November 2015: SystemML enters Apache incubation

February 2016: First release (0.9) of Apache SystemML

June 2016: Second Apache release (0.10)

2015

2016

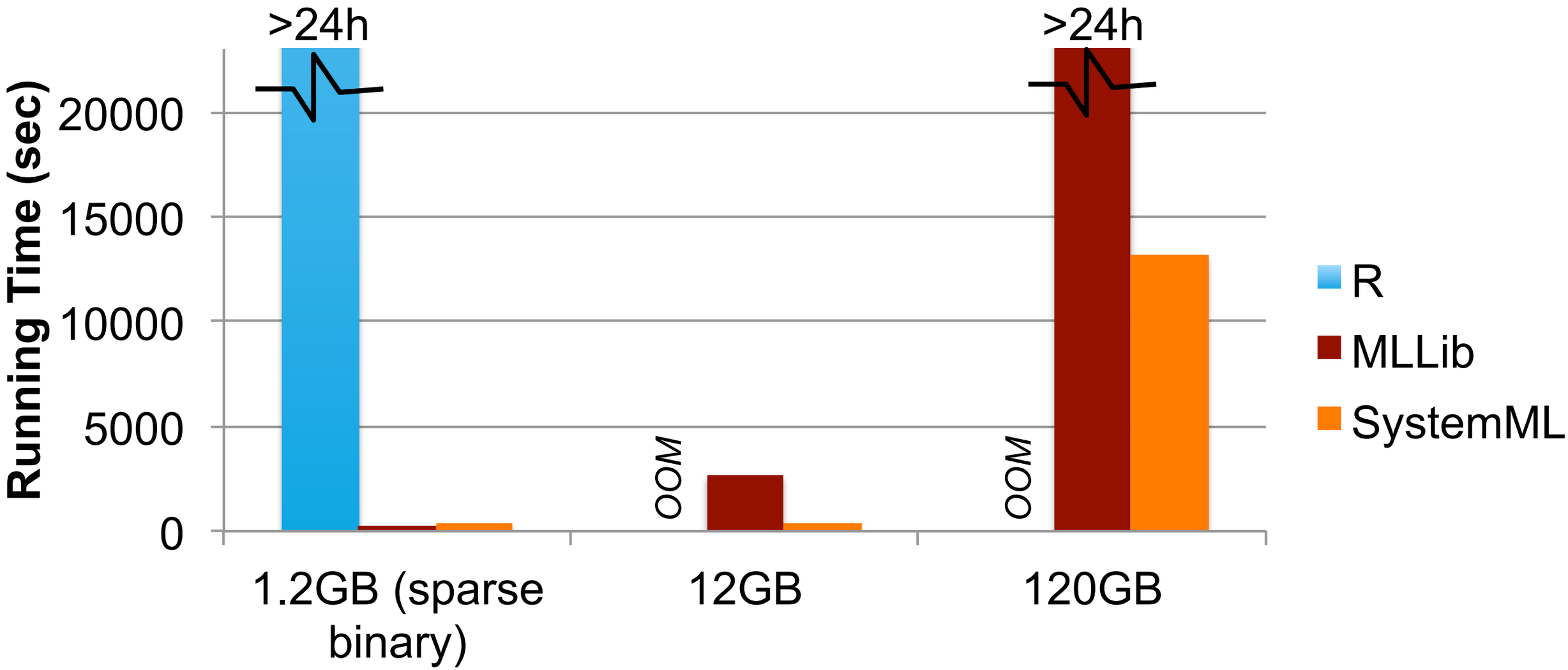
Apache SystemML

```
U = rand(nrow(X), r, min = -1.0, max = 1.0);
V = rand(r, ncol(X), min = -1.0, max = 1.0);
while(i < mi) {
   i = i + 1; ii = 1;
   if (is_U)
      G = (W * (U %*% V - X)) %*% t(V) + lambda * U;
   else
      G = t(U) %*% (W * (U %*% V - X)) + lambda * V;
   norm_G2 = sum(G ^ 2); norm_R2 = norm_G2;
   R = -G; S = R;
   while(norm_R2 > 10E-9 * norm_G2 & ii <= mii) {
     if (is_U) {
       HS = (W * (S %*% V)) %*% t(V) + lambda * S;
       alpha = norm_R2 / sum (S * HS);
       U = U + alpha * S;
     } else {
       HS = t(U) %*% (W * (U %*% S)) + lambda * S;
       alpha = norm_R2 / sum (S * HS);
       V = V + alpha * S;
     }
     R = R - alpha * HS;
     old_norm_R2 = norm_R2; norm_R2 = sum(R ^ 2);
     S = R + (norm_R2 / old_norm_R2) * S;
     ii = ii + 1;
   }
   is_U = ! is_U;
}
```

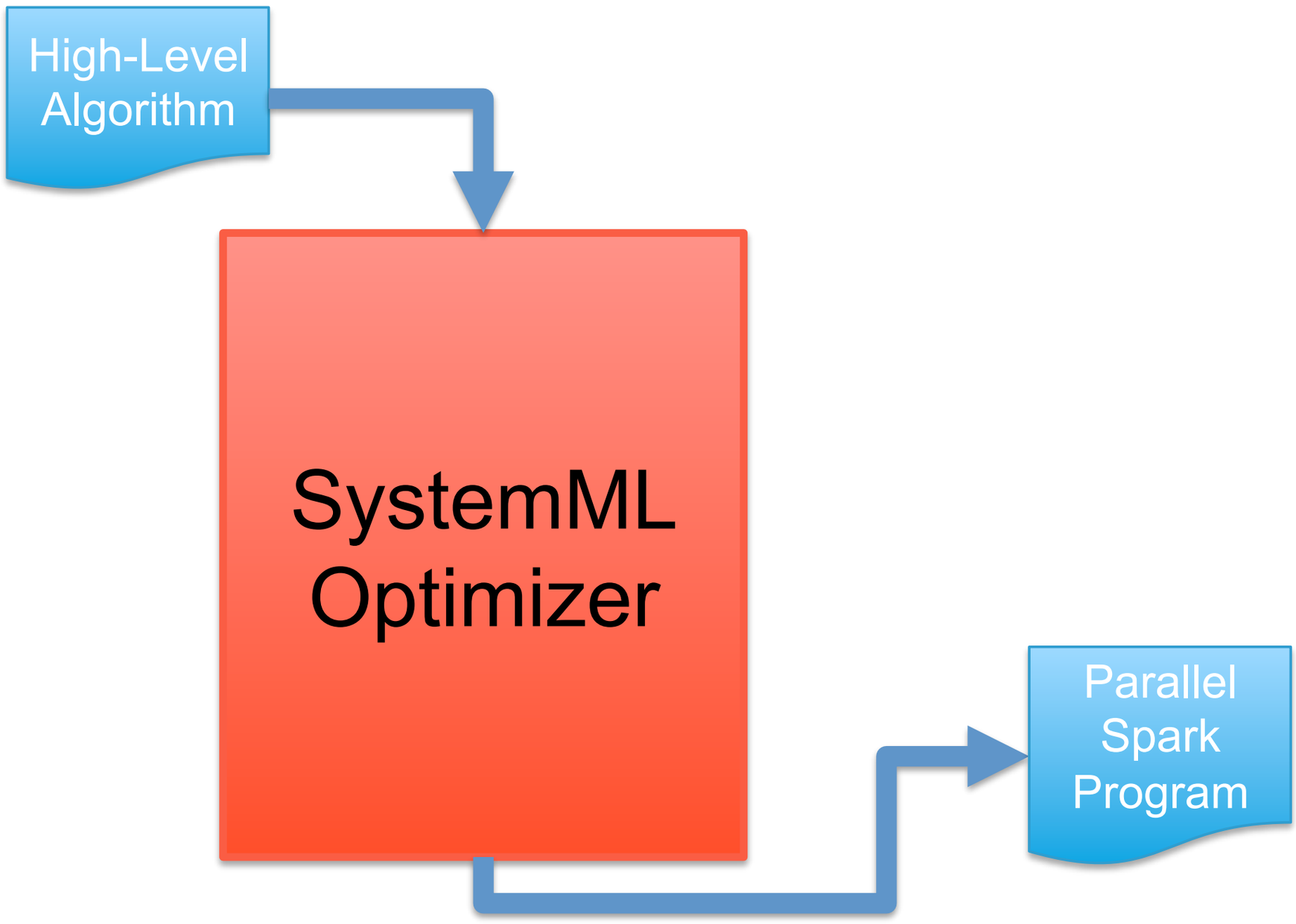
Apache SystemML

```
U = rand(nrow(X), r, min = -1.0, max = 1.0);
V = rand(r, ncol(X), min = -1.0, max = 1.0);
while(i < mi) {
   i = i + 1; ii = 1;
   if (is_U)
      G = (W * (U %*% V - X)) %*% t(V) + lambda * U;
   else
      G = t(U) %*% (W * (U %*% V - X)) + lambda * V;
   norm_G2 = sum(G ^ 2); norm_R2 = norm_G2;
   R = -G; S = R;
   while(norm_R2 > 10E-9 * norm_G2 & ii <= mii) {
     if (is_U) {
       HS = (W * (S %*% V)) %*% t(V) + lambda * S;
       alpha = norm_R2 / sum (S * HS);
       U = U + alpha * S;
     } else {
       HS = t(U) %*% (W * (U %*% S)) + lambda * S;
       alpha = norm_R2 / sum (S * HS);
       V = V + alpha * S;
     }
     R = R - alpha * HS;
     old_norm_R2 = norm_R2; norm_R2 = sum(R ^ 2);
     S = R + (norm_R2 / old_norm_R2) * S;
     ii = ii + 1;
   }
   is_U = ! is_U;
}
```

SystemML:
compile and run at scale
no performance code needed!


>24h
>24h
Running Time (sec)
20000
15000
10000
5000
0
OOM
OOM
1.2GB (sparse binary)
12GB
120GB
R
MLLib
SystemML

Architecture



Architecture

High-level language front-ends

Cost Based Optimizer

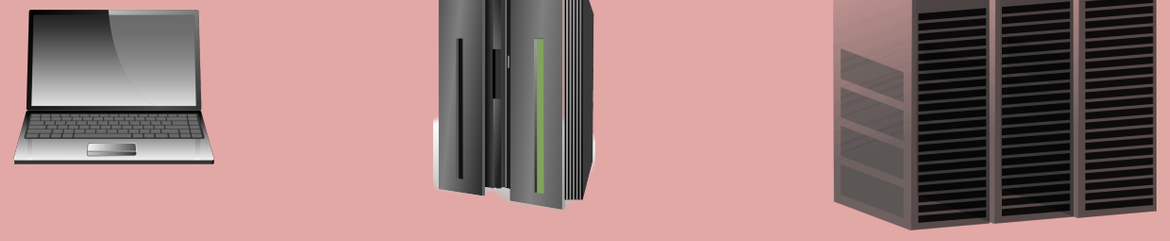
High-Level Operations (HOPs)

General representation of statements in the data analysis language

Low-Level Operations (LOPs)

General representation of operations in the runtime framework

Multiple execution environments



Architecture

High-level language front-ends

High-Level Operations (HOPs)

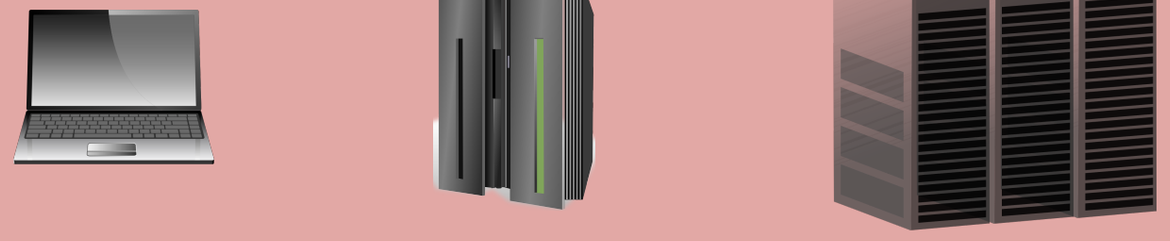
General representation of statements in the data analysis language

Low-Level Operations (LOPs)

General representation of operations in the runtime framework

Cost Based Optimizer

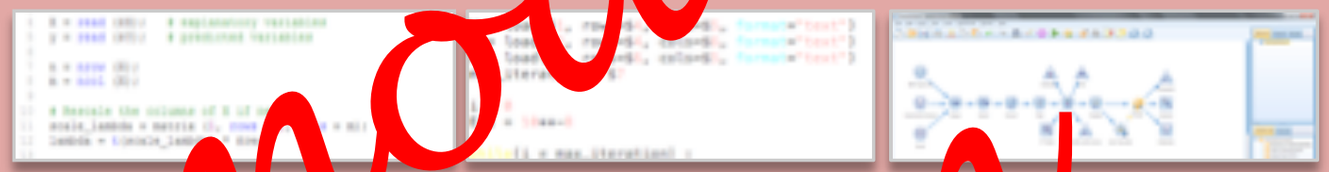
Multiple execution environments



data parallelism

Architecture

High-level language front-ends



Cost Based Optimizer

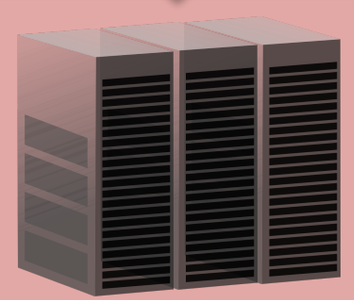
High-Level Operations (HOPs)

General representation of statements in the data analysis language

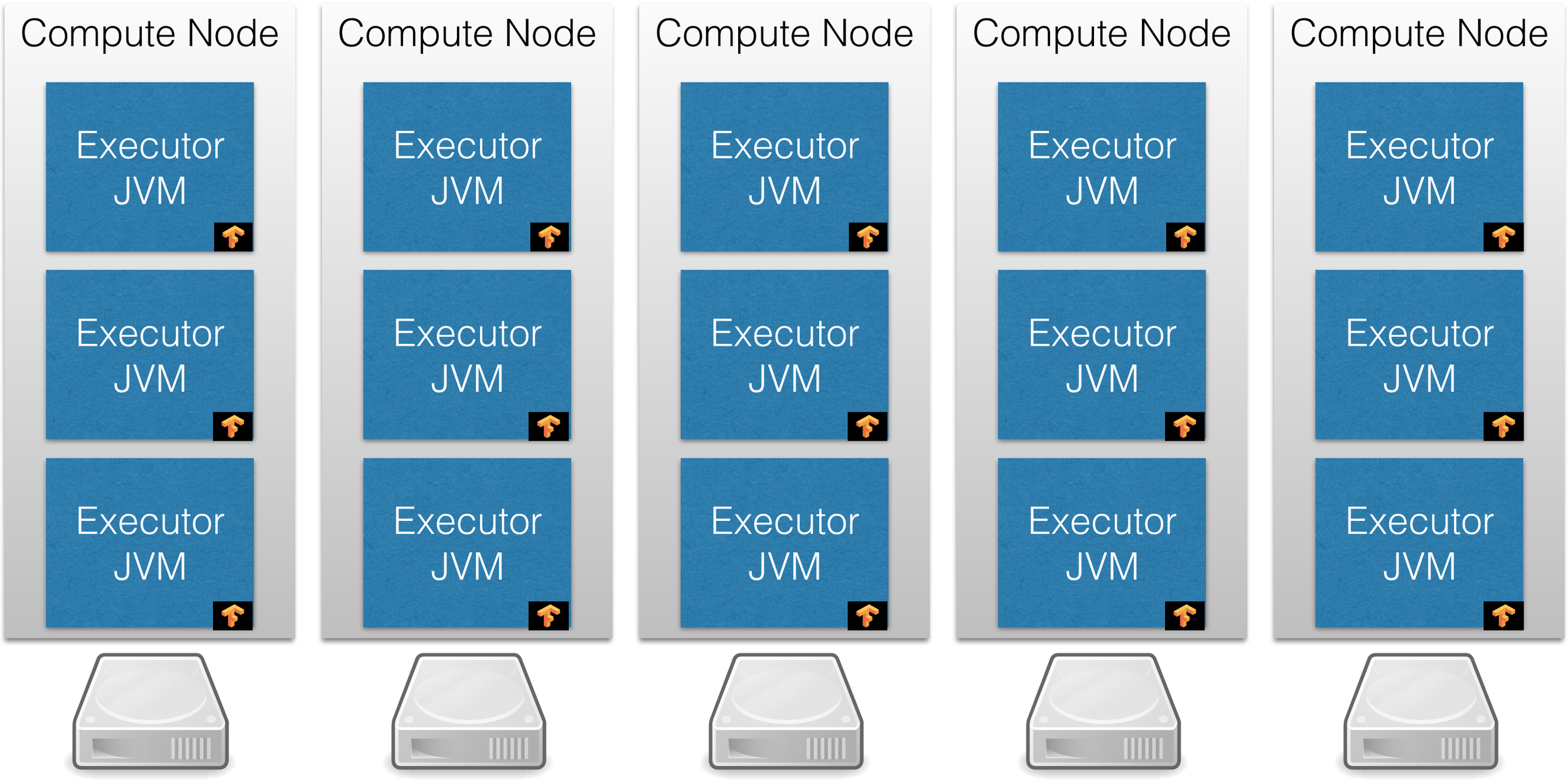
Low-Level Operations (LOPs)

General representation of operations in the runtime framework

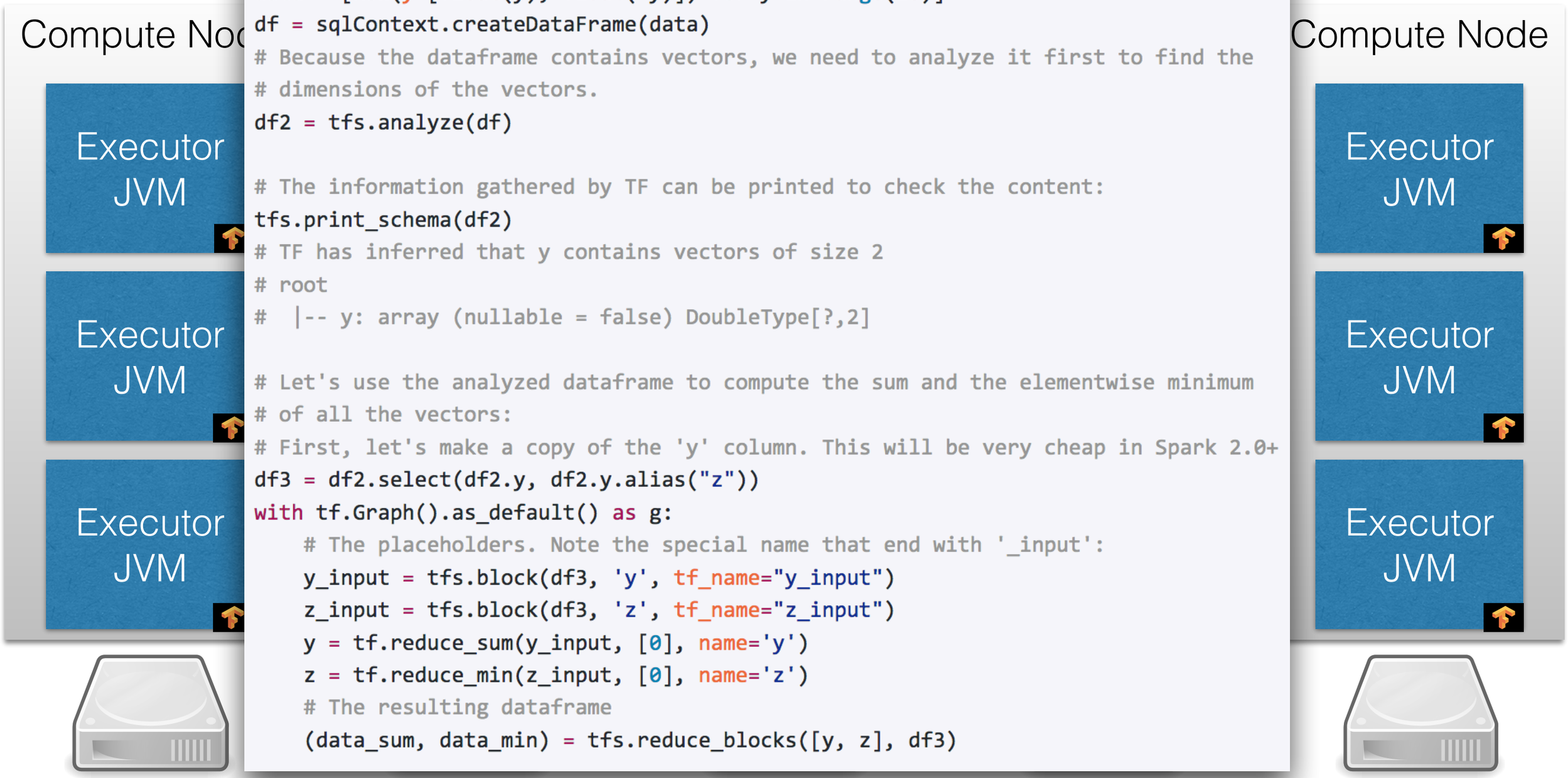
Multiple execution environments



TensorFrames



TensorFrames




```
# Build a DataFrame of vectors
data = [Row(y=[float(y), float(-y)]) for y in range(10)]
df = sqlContext.createDataFrame(data)
# Because the dataframe contains vectors, we need to analyze it first to find the
# dimensions of the vectors.
df2 = tfs.analyze(df)

# The information gathered by TF can be printed to check the content:
tfs.print_schema(df2)
# TF has inferred that y contains vectors of size 2
# root
#  |-- y: array (nullable = false) DoubleType[?,2]

# Let's use the analyzed dataframe to compute the sum and the elementwise minimum
# of all the vectors:
# First, let's make a copy of the 'y' column. This will be very cheap in Spark 2.0+
df3 = df2.select(df2.y, df2.y.alias("z"))
with tf.Graph().as_default() as g:
    # The placeholders. Note the special name that end with '_input':
    y_input = tfs.block(df3, 'y', tf_name="y_input")
    z_input = tfs.block(df3, 'z', tf_name="z_input")
    y = tf.reduce_sum(y_input, [0], name='y')
    z = tf.reduce_min(z_input, [0], name='z')
    # The resulting dataframe
    (data_sum, data_min) = tfs.reduce_blocks([y, z], df3)
```


Romeo Kienzler

TensorFrames

```
# Build a DataFrame of vectors
data = [Row(y=[float(y), float(-y)]) for y in range(10)]
df = sqlContext.createDataFrame(data)
# Because the dataframe contains vectors, we need to analyze it first to find the
# dimensions of the vectors.
df2 = tfs.analyze(df)

# The information gathered by TF can be printed to check the content:
tfs.print_schema(df2)
# TF has inferred that y contains vectors of size 2
# root
#  |-- y: array (nullable = false) DoubleType[?,2]

# Let's use the analyzed dataframe to compute the sum and the elementwise minimum
# of all the vectors:
# First let's make a copy of the 'y' column. This will be very cheap in Spark 2.0+
df3 = df2.select(df2.y, df2.y.alias("z"))
with tf.Graph().as_default() as g:
    # The placeholders. Note the special name that end with '_input':
    y_input = tfs.block(df3, 'y', tf_name="y_input")
    z_input = tfs.block(df3, 'z', tf_name="z_input")
    y = tf.reduce_sum(y_input, [0], name='y')
    z = tf.reduce_min(z_input, [0], name='z')
    # The resulting dataframe
    (data_sum, data_min) = tfs.reduce_blocks([y, z], df3)
```

Compute Node

Executor JVM

Executor JVM

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Executor JVM

inter-node parallelism

Romeo Kienzler

Parameter Server on Driver

TensorSpark

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

Executor JVM

Compute Node

Executor JVM

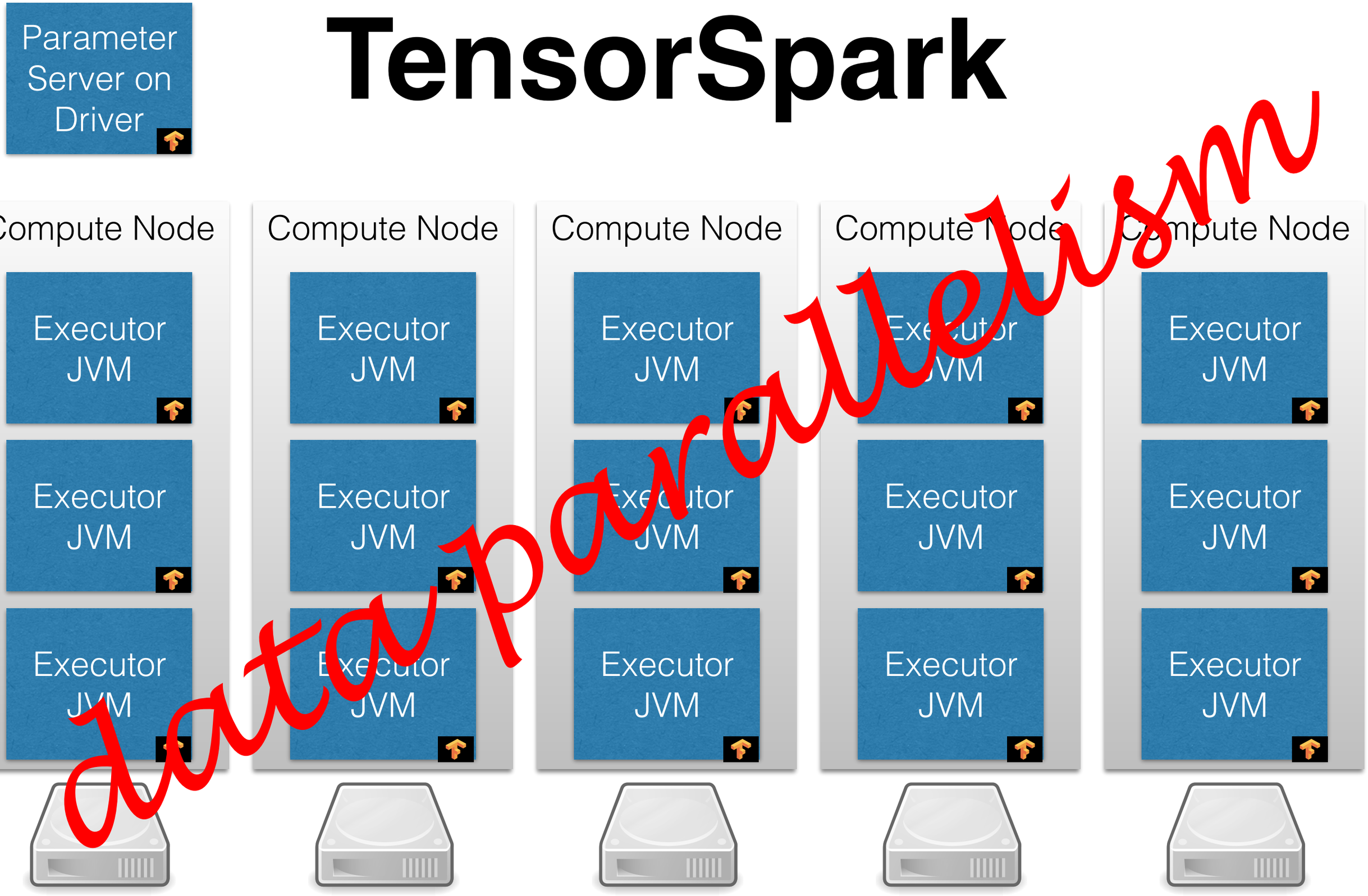
Executor JVM

Compute Node

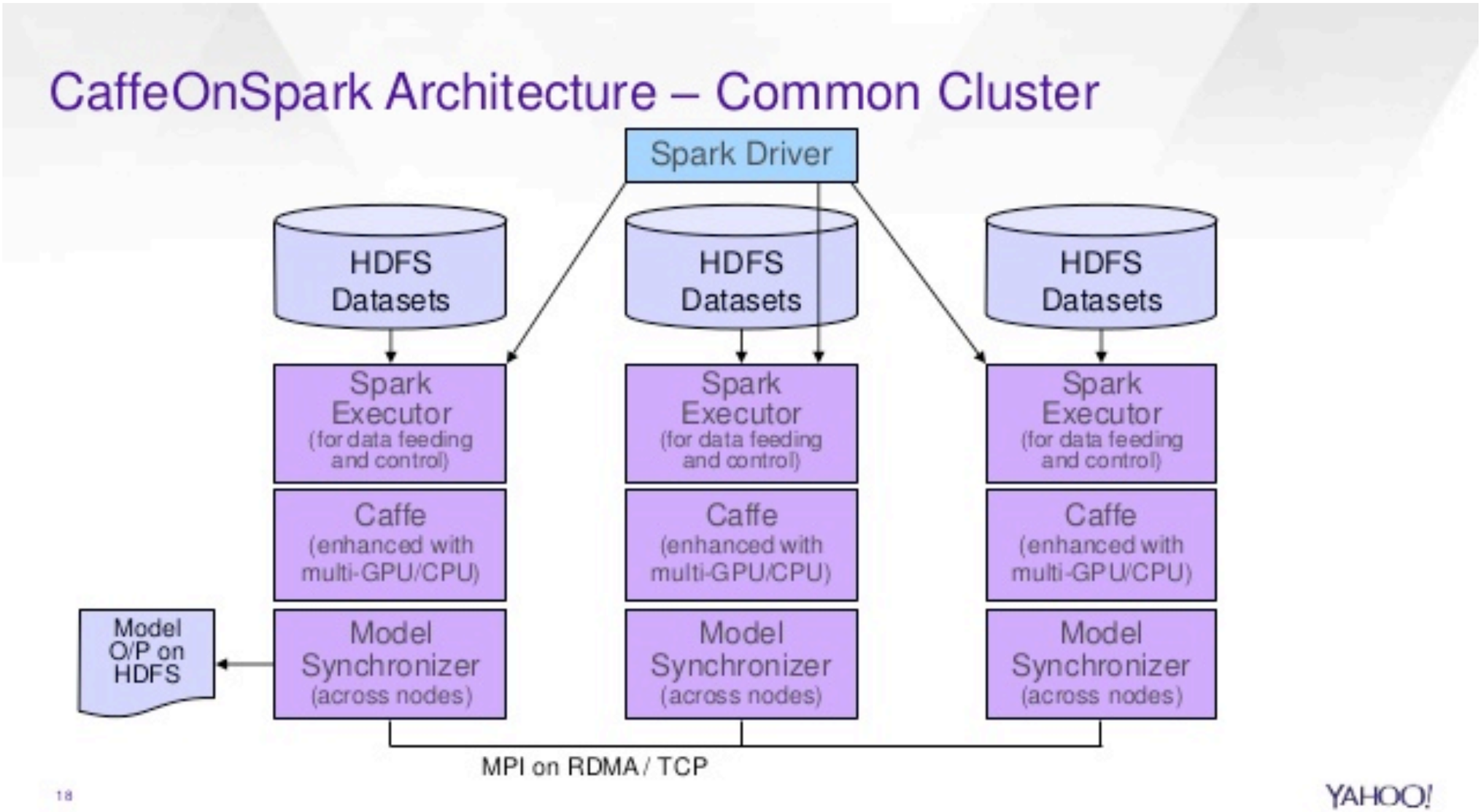
Executor JVM

Executor JVM

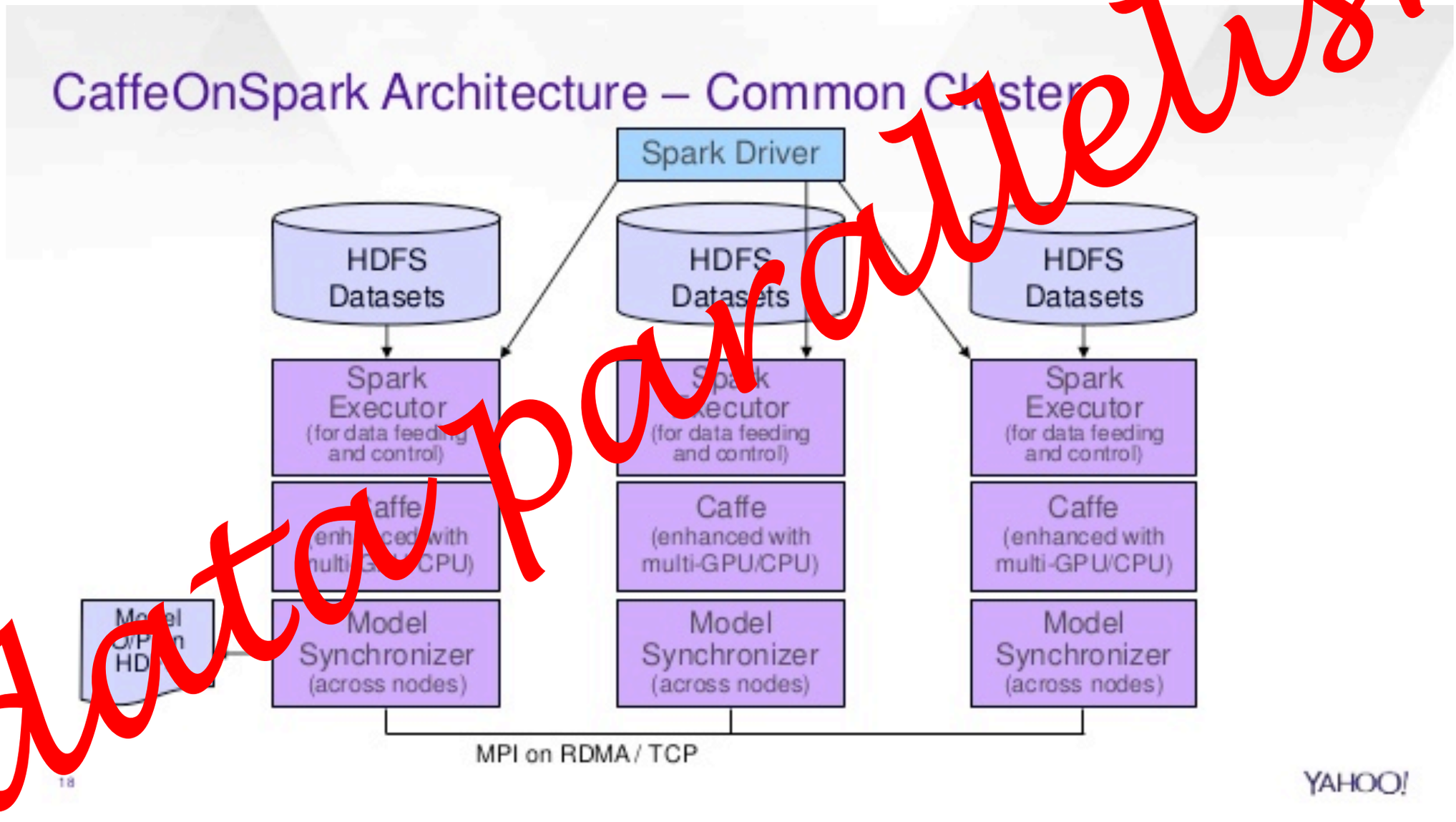
TensorSpark



CaffeOnSpark



CaffeOnSpark



data parallelism

?