cloudera

Estimating Financial Risk with Spark

Sandy Ryza | Senior Data Scientist

cloudera

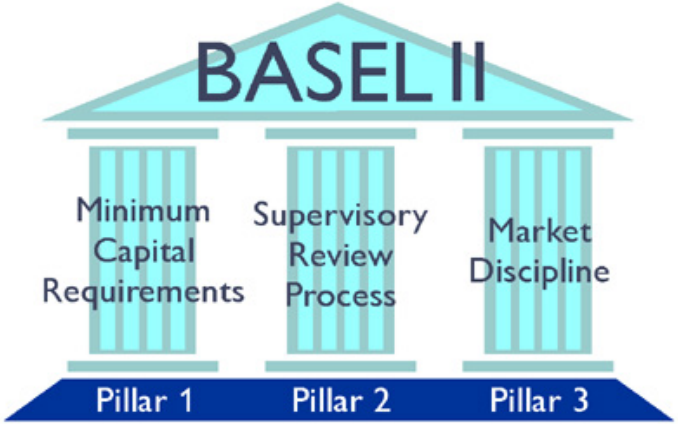
© Cloudera, Inc. All rights reserved.

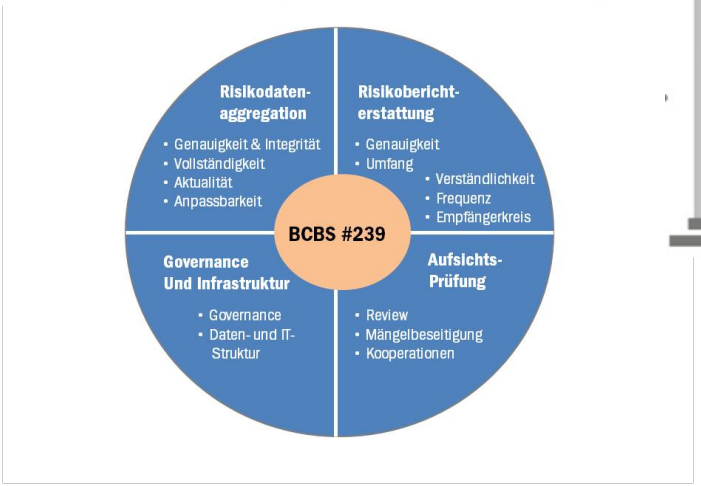
cloudera

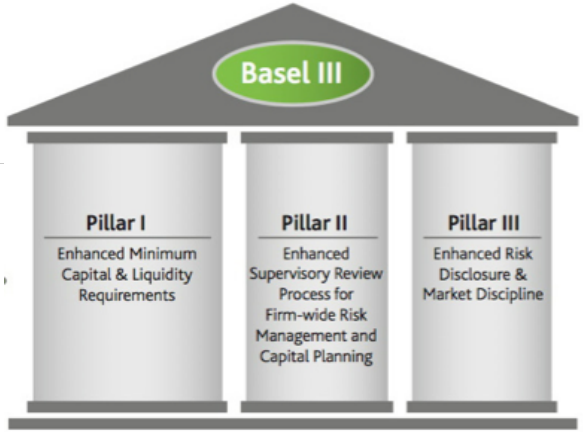
© Cloudera, Inc. All rights reserved.

In reasonable circumstances, what's the most you can expect to lose?

© Cloudera, Inc. All rights reserved.







cloudera

© Cloudera, Inc. All rights reserved.

```
def valueAtRisk(
  portfolio,
  timePeriod,
  pValue
): Double = { ... }
```

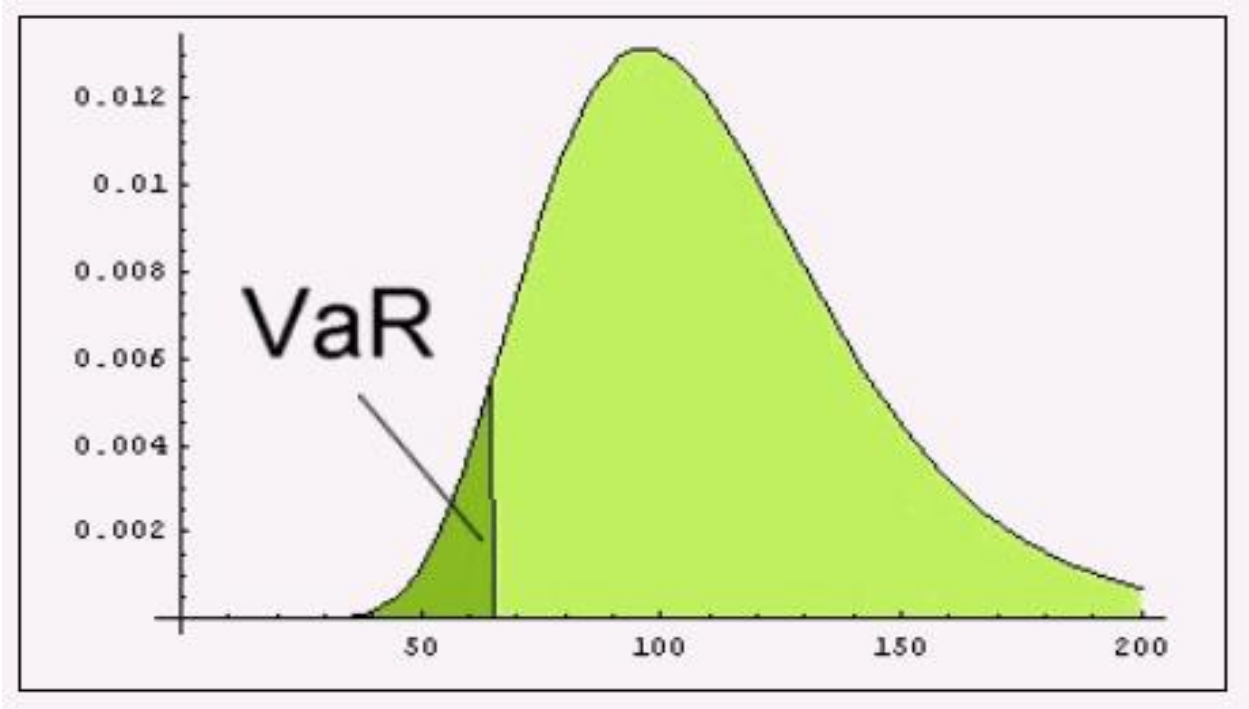

© Cloudera, Inc. All rights reserved.

```
def valueAtRisk(
  portfolio,
  2 weeks,
  0.05
) = $1,000,000
```

cloudera

© Cloudera, Inc. All rights reserved.

Probability density



Portfolio return ($) over the time period

cloudera

© Cloudera, Inc. All rights reserved.

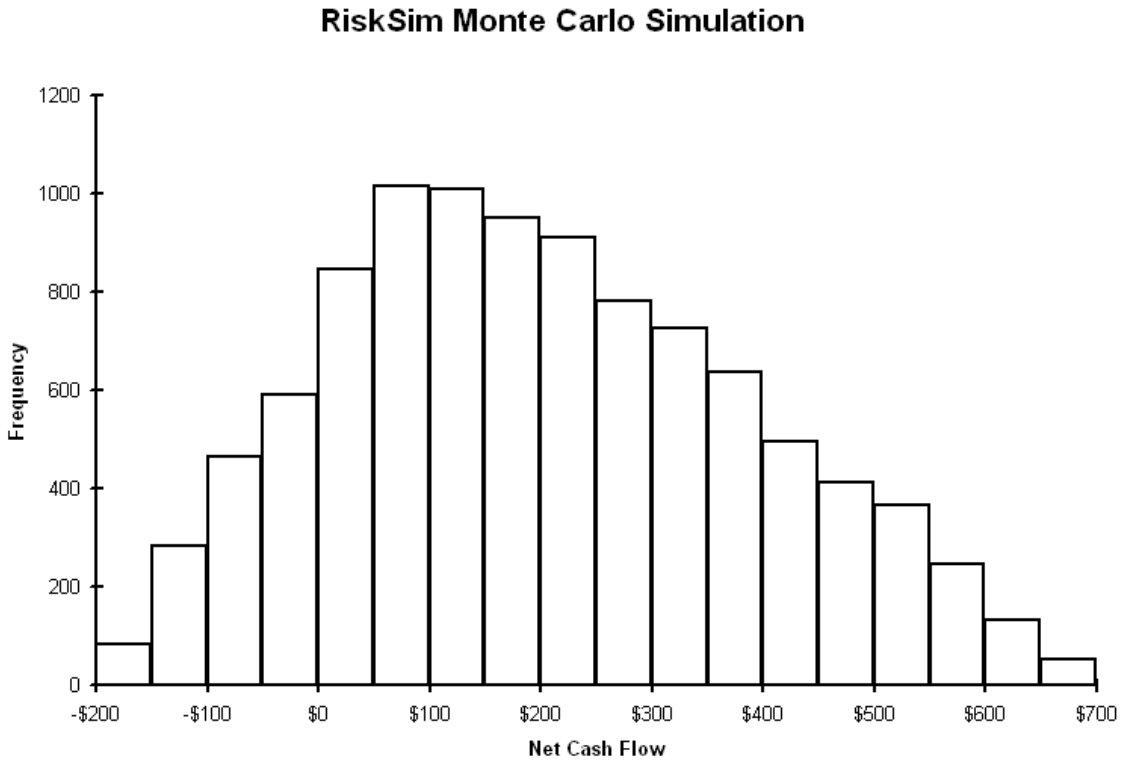
VaR estimation approaches

- Variance-covariance
- Historical
- Monte Carlo

© Cloudera, Inc. All rights reserved.

cloudera

© Cloudera, Inc. All rights reserved.



cloudera

© Cloudera, Inc. All rights reserved.

Market Risk Factors

- Indexes (S&P 500, NASDAQ)
- Prices of commodities
- Currency exchange rates
- Treasury bonds

© Cloudera, Inc. All rights reserved.

Predicting Instrument Returns from Factor Returns

- Train a linear model on the factors for each instrument

$$r_{it} = c_i + \sum_{j=1}^{|w_i|} w_{ij} \cdot m_{tj}$$

© Cloudera, Inc. All rights reserved.

Fancier

- Add features that are non-linear transformations of the market risk factors
- Decision trees
- For options, use Black-Scholes

© Cloudera, Inc. All rights reserved.

```
import org.apache.commons.math3.stat.regression.OLSMultipleLinearRegression

// Load the instruments and factors
val factorReturns: Array[Array[Double]] = ...
val instrumentReturns: RDD[Array[Double]] = ...

// Fit a model to each instrument
val models: Array[Array[Double]] =
  instrumentReturns.map { instrument =>
    val regression = new OLSMultipleLinearRegression()
    regression.newSampleData(instrument, factorReturns)
    regression.estimateRegressionParameters()
  }.collect()
```


© Cloudera, Inc. All rights reserved.

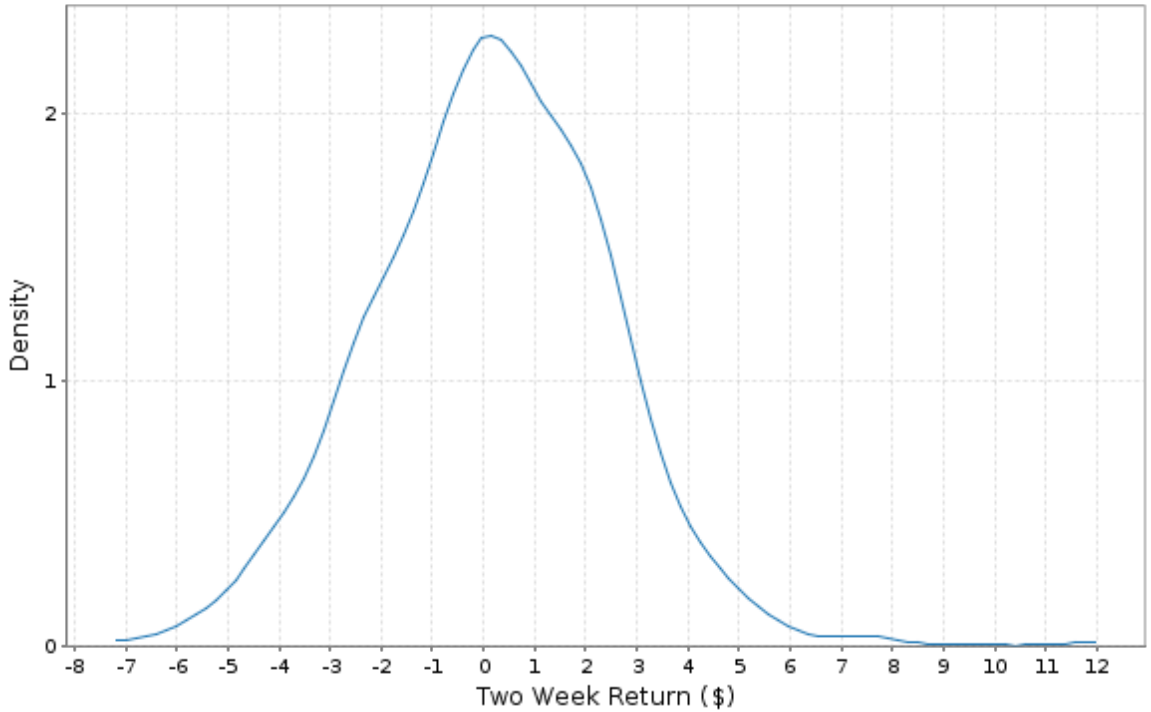
How to sample factor returns?

- Need to be able to generate sample vectors where each component is a factor return.

- Factors returns are usually correlated.

© Cloudera, Inc. All rights reserved.

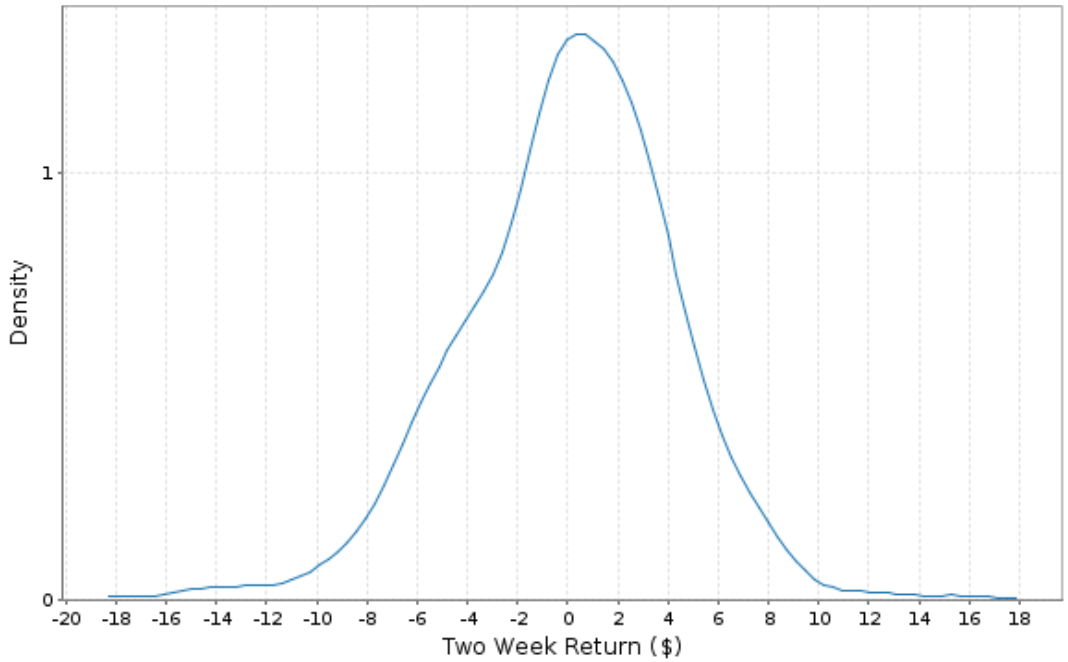
Distribution of US treasury bond two-week returns



cloudera

© Cloudera, Inc. All rights reserved.

Distribution of crude oil two-week returns



cloudera

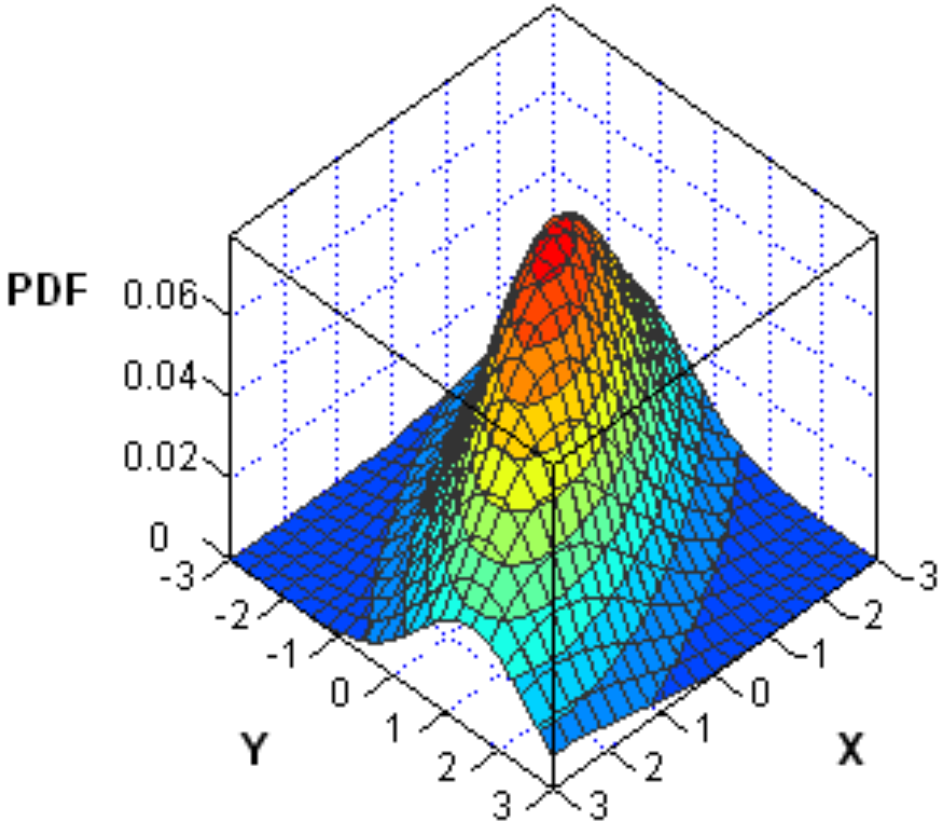
© Cloudera, Inc. All rights reserved.

The Multivariate Normal Distribution

$$m_t \sim \mathcal{N}(\mu, \Sigma)$$

- Probability distribution over vectors of length N
- Given all the variables but one, that variable is distributed according to a univariate normal distribution
- Models correlations between variables

© Cloudera, Inc. All rights reserved.



© Cloudera, Inc. All rights reserved.

```
import org.apache.commons.math3.stat.correlation.Covariance

// Compute means
val factorMeans: Array[Double] = transpose(factorReturns)
  .map(factor => factor.sum / factor.size)

// Compute covariances
val factorCovs: Array[Array[Double]] = new Covariance(factorReturns)
  .getCovarianceMatrix().getData()
```


© Cloudera, Inc. All rights reserved.

Fancier

- Multivariate normal often a poor choice compared to more sophisticated options
- Fatter tails: Multivariate T Distribution
- Filtered historical simulation
  - ARMA
  - GARCH

© Cloudera, Inc. All rights reserved.

Running the simulations

- Create an RDD of seeds
- Use each seed to generate a set of simulations
- Aggregate results

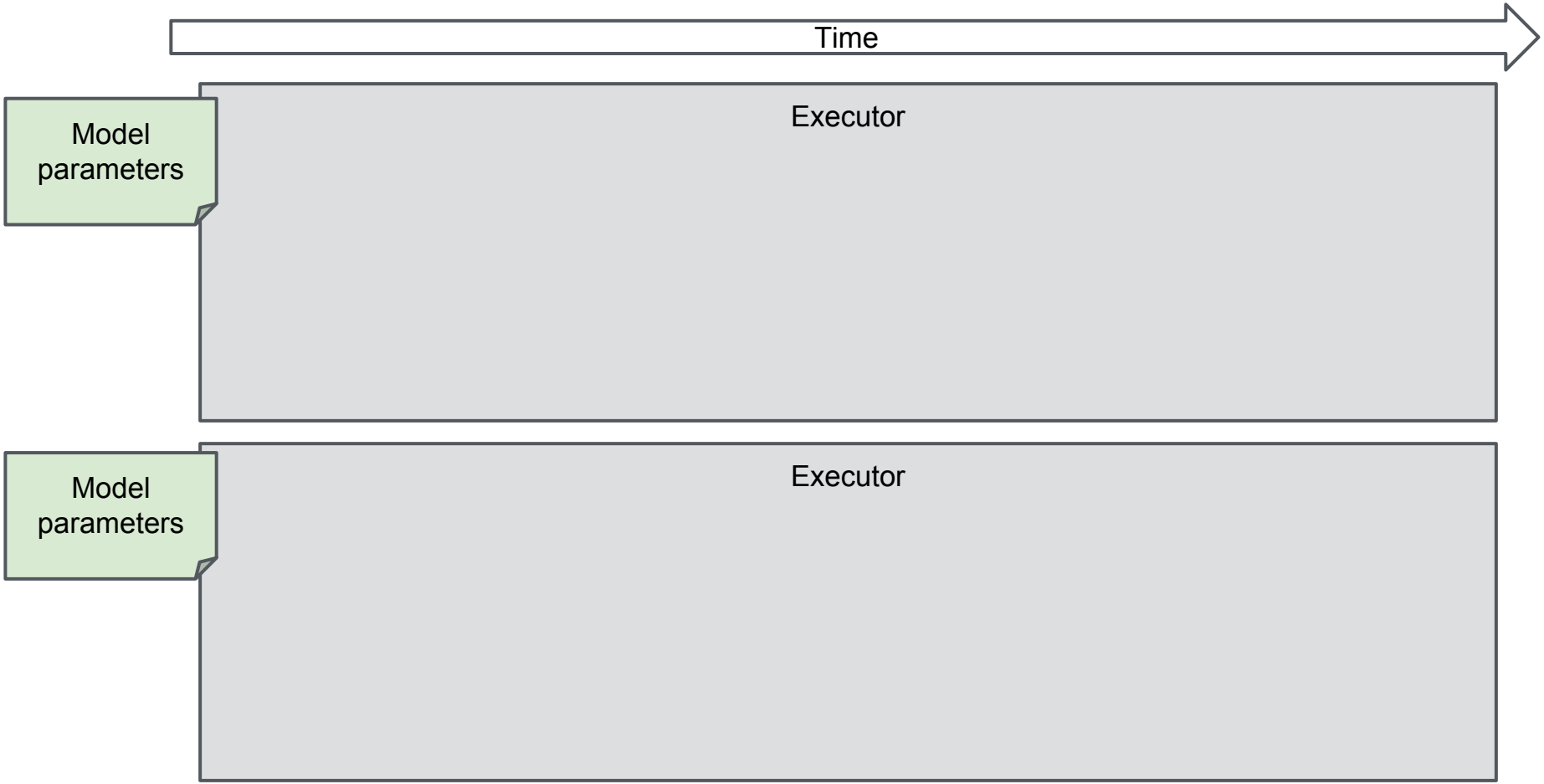
© Cloudera, Inc. All rights reserved.

```
// Broadcast the factor return -> instrument return models
val bModels = sc.broadcast(models)

// Generate a seed for each task
val seeds = (baseSeed until baseSeed + parallelism)
val seedRdd = sc.parallelize(seeds, parallelism)

// Create an RDD of trials
val trialReturns: RDD[Double] = seedRdd.flatMap { seed =>
  trialReturns(seed, trialsPerTask, bModels.value, factorMeans, factorCovs)
}
```


© Cloudera, Inc. All rights reserved.

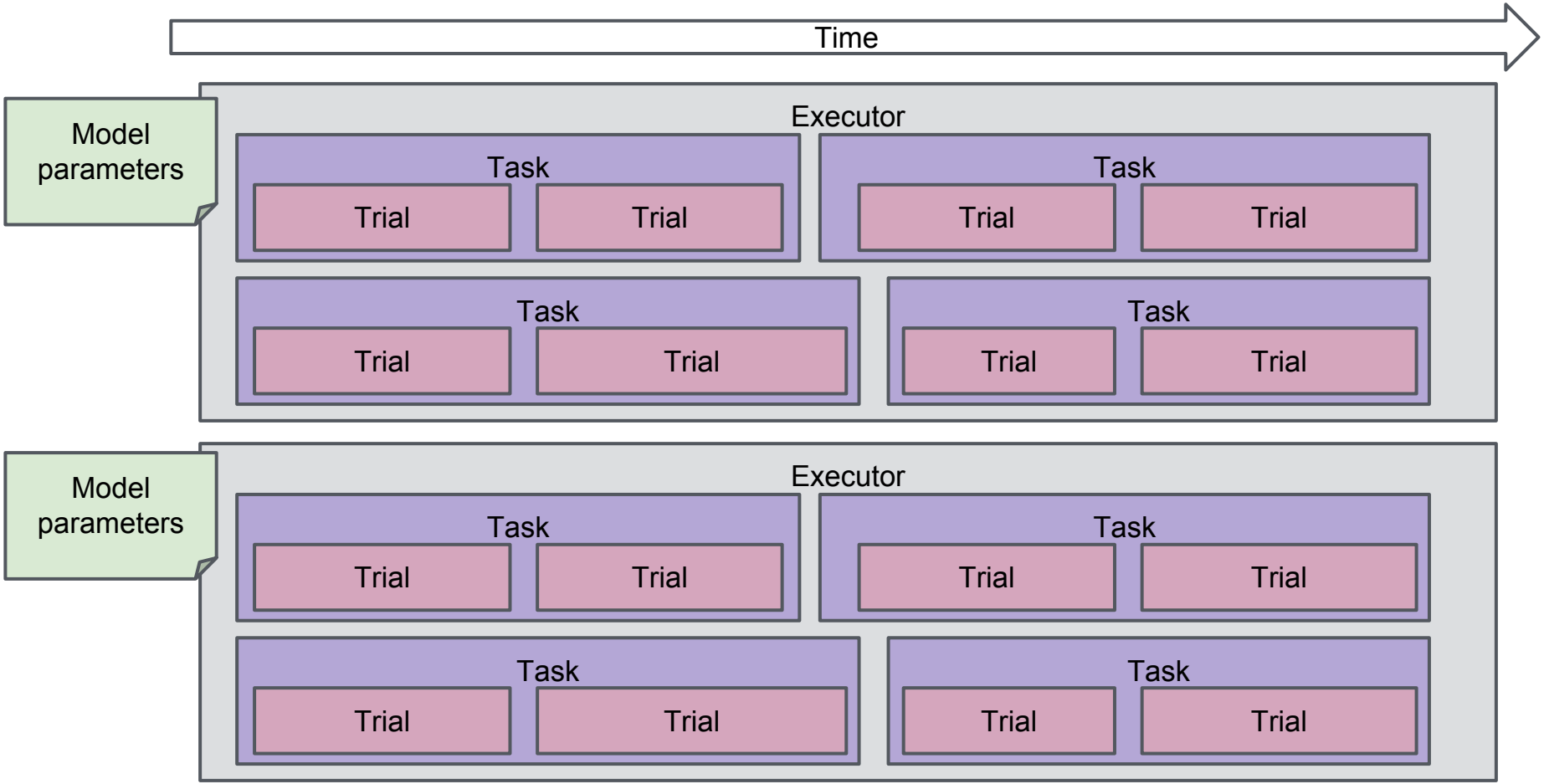
```
def trialReturn(factorDist: MultivariateNormalDistribution, models: Seq[Array[Double]]): Double = {
  val trialFactorReturns = factorDist.sample()
  var totalReturn = 0.0

  for (model <- models) {
    // Add the returns from the instrument to the total trial return
    for (i <- until trialFactorsReturns.length) {
      totalReturn += trialFactorReturns(i) * model(i)
    }
  }
  totalReturn
}
```


© Cloudera, Inc. All rights reserved.



cloudera

© Cloudera, Inc. All rights reserved.

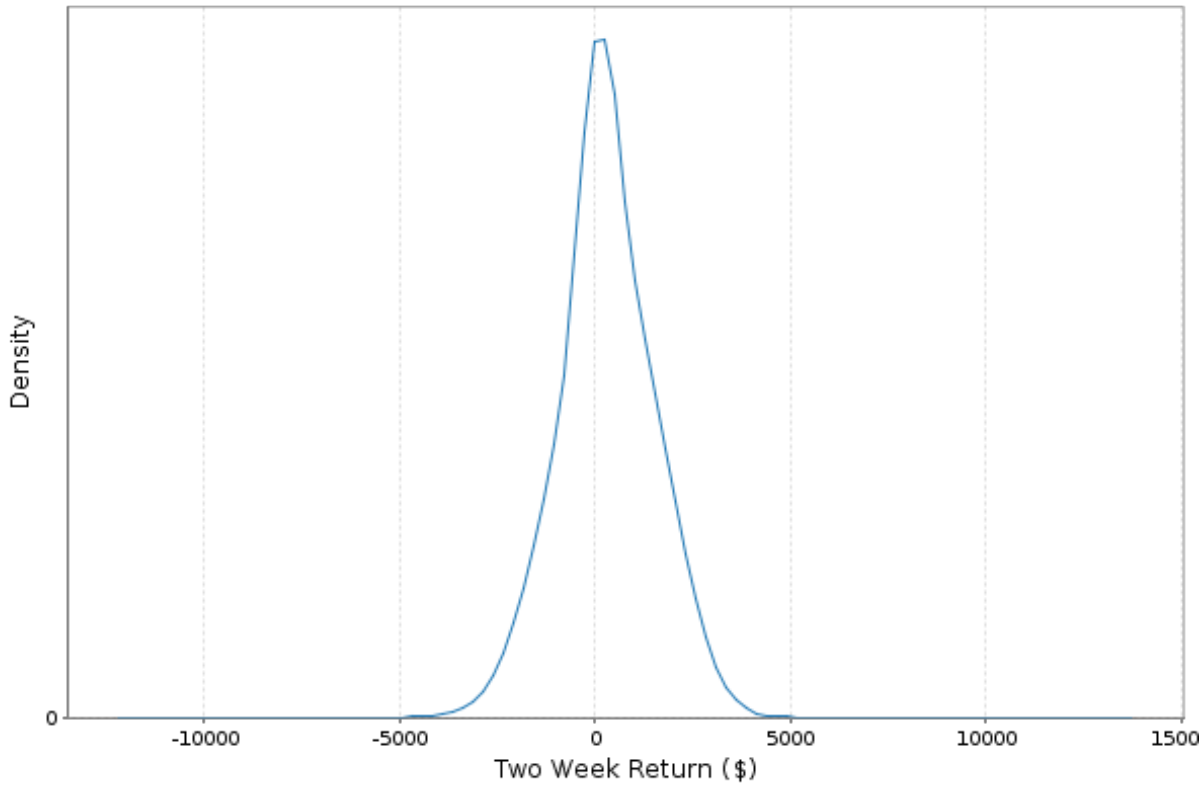


cloudera

© Cloudera, Inc. All rights reserved.

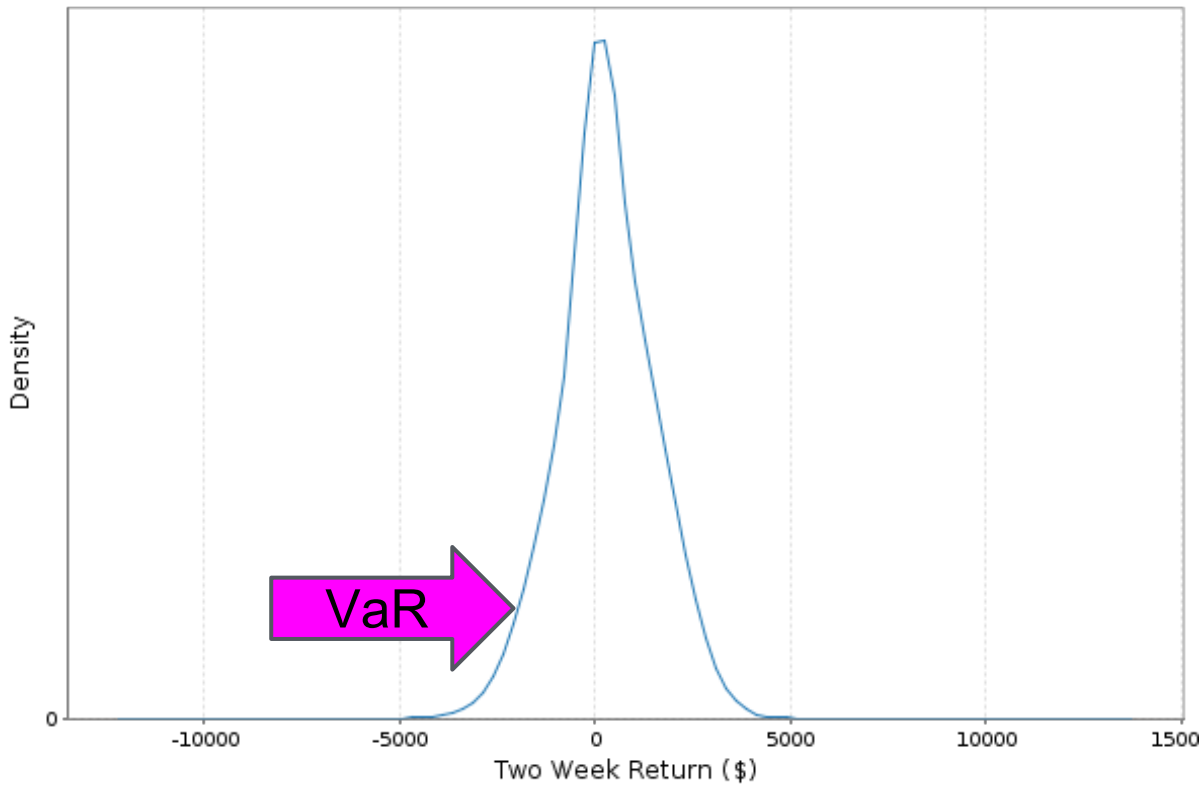
```
// Cache for reuse
trialReturns.cache()

val numTrialReturns = trialReturns.count().toInt

// Compute value at risk
val valueAtRisk = trials.takeOrdered(numTrialReturns / 20).last

// Compute expected shortfall
val expectedShortfall =
  trials.takeOrdered(numTrialReturns / 20).sum / (numTrialReturns / 20)
```


© Cloudera, Inc. All rights reserved.



cloudera

© Cloudera, Inc. All rights reserved.



cloudera

© Cloudera, Inc. All rights reserved.

So why Spark?

© Cloudera, Inc. All rights reserved.

Ease of use

- Parallel computing for 5-year olds
- Scala, Python, and R REPLs

© Cloudera, Inc. All rights reserved.

Single platform for

- Cleaning data
- Fitting models
- Running simulations
- Storing results
- Analyzing results

cloudera

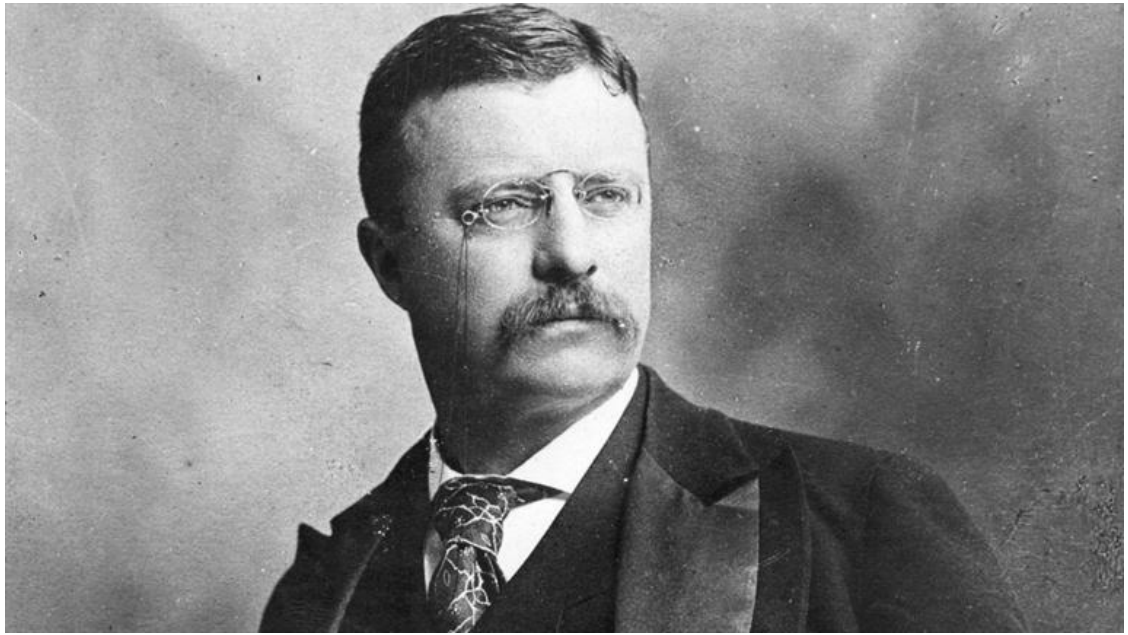
© Cloudera, Inc. All rights reserved.

But it’s CPU-bound and we’re using Java?

- Computational bottlenecks are normally in matrix operations, which can be BLAS-ified
- Can call out to GPUs just like in C++
- Memory access patterns aren’t high-GC inducing

© Cloudera, Inc. All rights reserved.

Want to do this yourself?



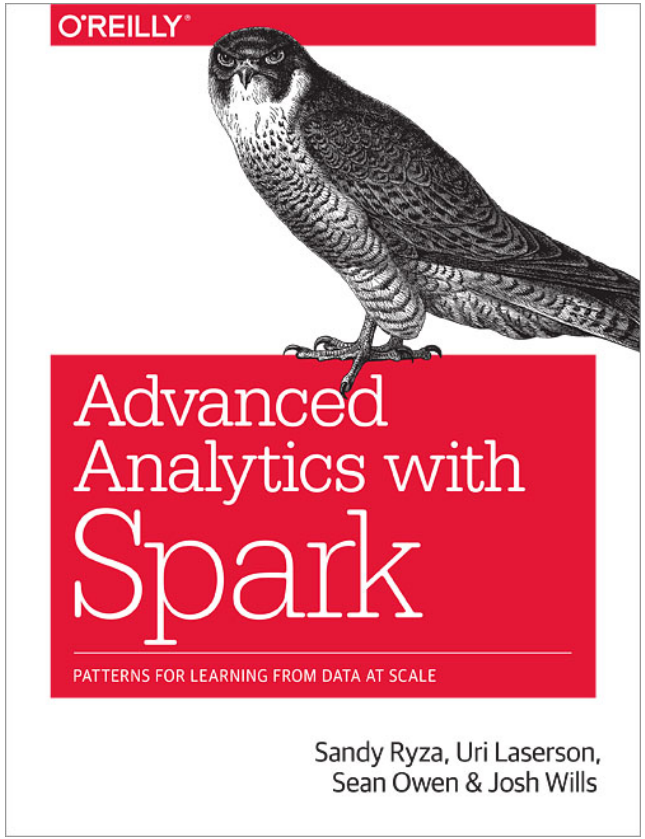
cloudera

© Cloudera, Inc. All rights reserved.

spark-timeseries

- https://github.com/cloudera/spark-timeseries
- Everything here + some fancier stuff
- Patches welcome!

© Cloudera, Inc. All rights reserved.



cloudera

© Cloudera, Inc. All rights reserved.