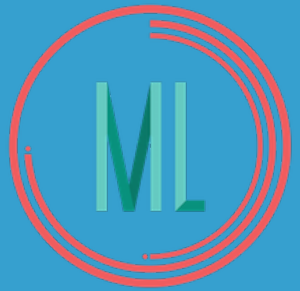


Inside Apache SystemML

Fred Reiss

Chief Architect, IBM Spark Technology Center

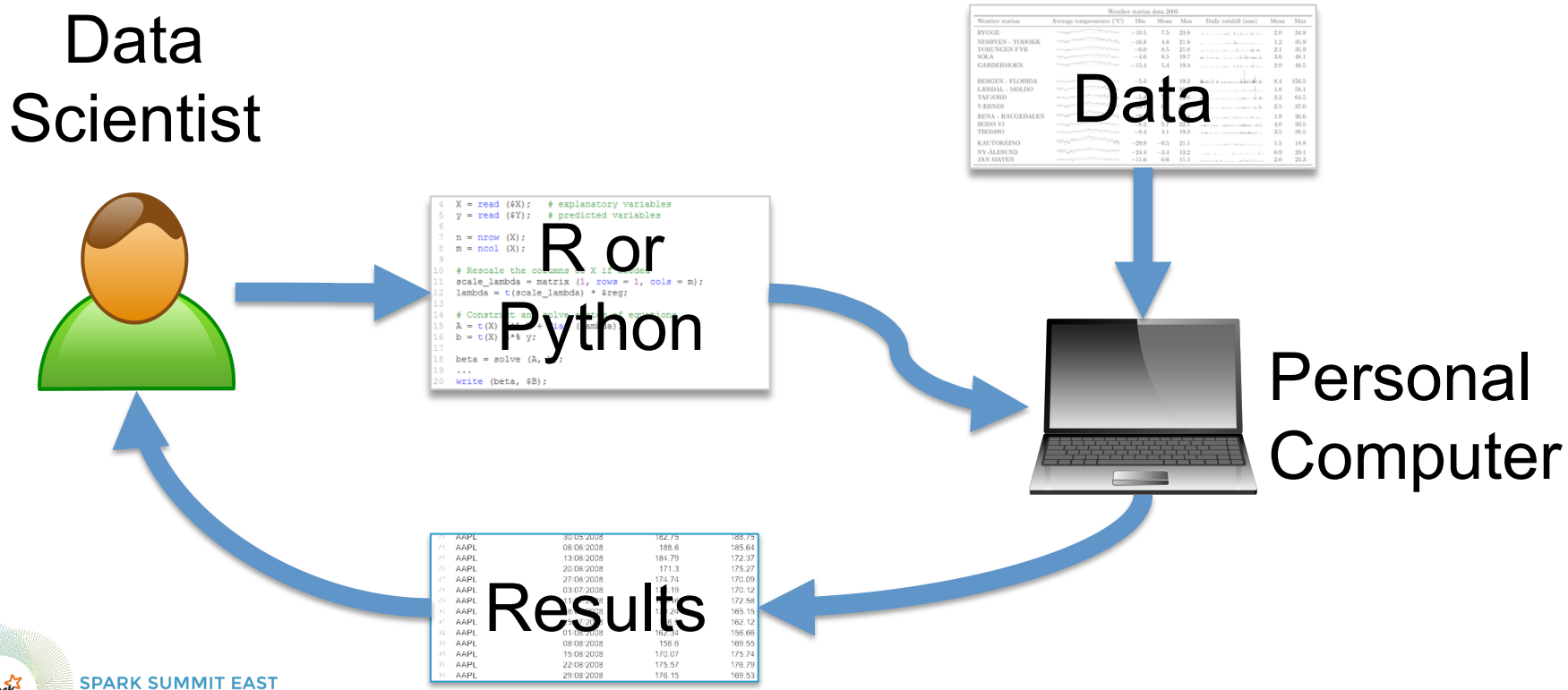
Member of the IBM Academy of Technology

SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY

Origins of the SystemML Project

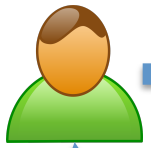
- **2007-2008:** Multiple projects at IBM Research – Almaden involving machine learning on Hadoop.
- **2009:** We create a dedicated team for scalable ML.
- **2009-2010:** Through engagements with customers, we observe how data scientists create machine learning algorithms.

State-of-the-Art: Small Data



State-of-the-Art: Big Data

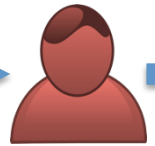
Data Scientist



```
X = read (&X);   # explanatory variables
y = read (&Y);   # predicted variables

n = nrow (X);
m = ncol (X);

# Rescale the col
scale_lambda = matrix (1, rows = 1, cols = m);
lambda = t(scale_lambda) * lreg;

# Construct
A = t(X)
b = t(X)

beta = solve (A,
...
write (beta, &B);
```

R or Python

Systems Programmer



```
var health:   Int = 100,
var inventory:  Seq[InventoryItem] = Nil)

def main(args: Array[String]) {

  object BusPass extends InventoryItem("a bus pass")
  case class Money(dollars: Int) extends InventoryItem("$" + dollars)
  object CoolDrinkingAge extends InventoryItem("a book on code breaking")

  val livingRoom = new Place("the living room", prop = "lot")
  val gs = GameState(
  ...
  val closet   = new Place("a closet")
  val road     = new Place("a road")
  val bus      = new Place("a bus")
  gs.inventory = gs.inventory.filterNot(_ == BusPass)
  }}
  val library   = Place("the library")
  val kiosk     = Place("an information kiosk with a strange code written on it")
  val ladder    = Place("a ladder hidden inside the kiosk", prop = "lot")
  val treasureRoom= Place("a room full of treasure", prop = "lot", goal = true)

  case class Transition(from: Place, mustHave: Seq[InventoryItem] = Nil)
  val travelsandPlace = {
    implicit def place2Tr(place: Place) = Transition(place)
```

Scala

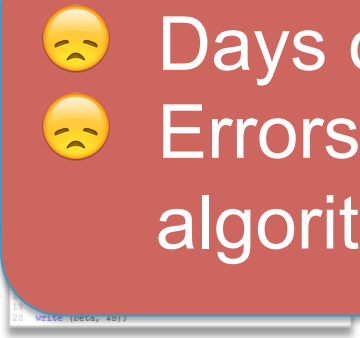
24	AAPL	30/05/2008	182.75	188.75
25	AAPL	06/06/2008	188.6	185.64
26	AAPL	13/06/2008	184.79	172.37
27	AAPL	20/06/2008	171.3	175.27
28	AAPL	27/06/2008	174.74	170.09
29	AAPL	03/07/2008	170.19	170.12
30	AAPL	11/07/2008	170.56	172.58
31	AAPL	18/07/2008	172.54	165.15
32	AAPL	25/07/2008	166.9	162.12
33	AAPL	01/08/2008	162.34	156.66
34	AAPL	08/08/2008	158.6	169.55
35	AAPL	15/08/2008	170.07	175.74
36	AAPL	22/08/2008	175.57	176.79
37	AAPL	29/08/2008	176.15	169.53

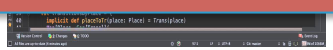
Results

State-of-the-Art: Big Data

Data Scientist

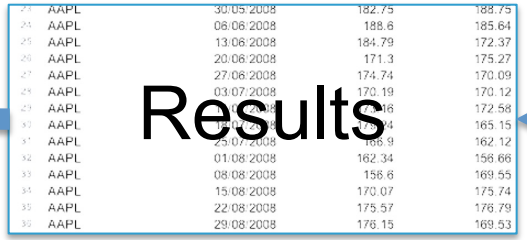
- Days or weeks per iteration
- Errors while translating algorithms





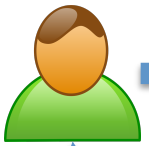
Spark

Results



The SystemML Vision

Data Scientist



```
X = read ($X);   # explanatory variables
y = read ($Y);   # predicted variables

n = nrow (X);
m = ncol (X);

# Rescale the columns of X
scale_lambda = matrix (1, rows = 1, cols = m);
lambda = t(scale_lambda) * $reg;

# Construct and solve the linear system
A = t(X) %*% X + diag (lambda);
b = t(X) %*% y;

beta = solve (A,
...
write (beta, $B);
```

R or Python

SystemML

Spark

24	AAPL	30/05/2008	182.75	188.75
25	AAPL	06/06/2008	188.6	185.64
26	AAPL	13/06/2008	184.79	172.37
27	AAPL	20/06/2008	171.3	175.27
28	AAPL	27/06/2008	174.74	170.09
29	AAPL	03/07/2008	170.19	170.12
30	AAPL	11/07/2008	166	172.58
31	AAPL	18/07/2008	174	165.15
32	AAPL	25/07/2008	166.9	162.12
33	AAPL	01/08/2008	162.34	156.66
34	AAPL	08/08/2008	156.6	169.55
35	AAPL	15/08/2008	170.07	175.74
36	AAPL	22/08/2008	175.57	176.79
37	AAPL	29/08/2008	176.15	169.53

Results

The SystemML Vision

Data Scientist

```
X = read ($X);   # explanatory variables
y = read ($Y);   # predicted variables

n = nrow (X);
m = ncol (X);

# Rescale the columns in X
scale_lambda = matrix (1, rows = 1, cols = m);
lambda = t(scale_lambda) * $reg;

# Construct
A = t(X)
b = t(X)

beta = solve (A,
...
write (beta, $B);
```

R or Python

Fast iteration
Same answer

SystemML

Spark

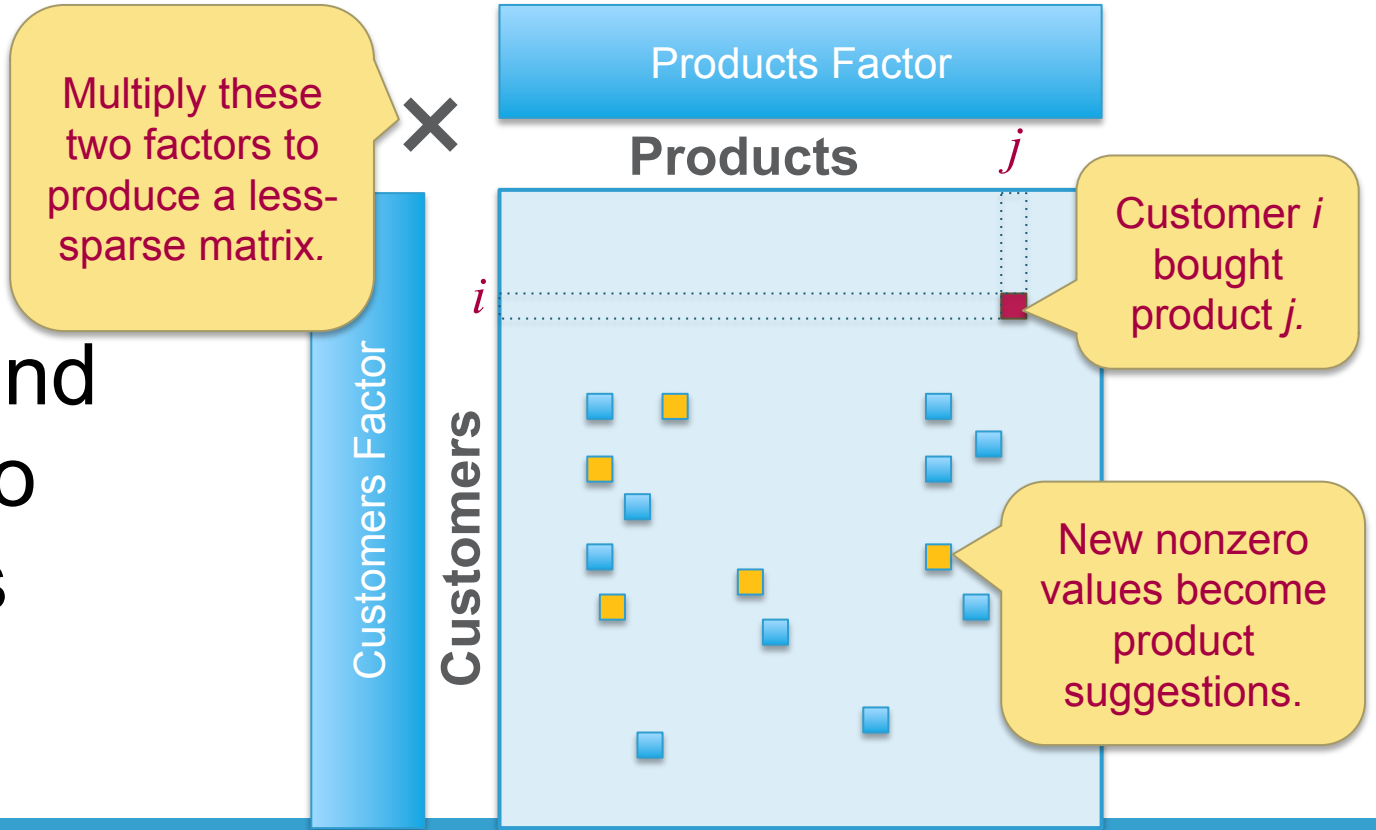
24	AAPL	30/05/2008	182.75	188.75
25	AAPL	06/06/2008	188.6	185.64
26	AAPL	13/06/2008	184.79	172.37
27	AAPL	20/06/2008	171.3	175.27
28	AAPL	27/06/2008	174.74	170.09
29	AAPL	03/07/2008	170.19	170.12
30	AAPL	11/07/2008	166	172.58
31	AAPL	18/07/2008	164	165.15
32	AAPL	25/07/2008	168.9	162.12
33	AAPL	01/08/2008	162.34	156.66
34	AAPL	08/08/2008	156.6	169.55
35	AAPL	15/08/2008	170.07	175.74
36	AAPL	22/08/2008	175.57	176.79
37	AAPL	29/08/2008	176.15	169.53

Results

Running Example:

Alternating Least Squares

- Problem: Recommend products to customers



Alternating Least Squares (in R)

```
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;
}
```

Alternating Least Squares (in R)

```
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;
}
```

1. Start with random factors.
2. Hold the Products factor constant and find the best value for the Customers factor.
   (Value that most closely approximates the original matrix)
3. Hold the Customers factor constant and find the best value for the Products factor.
4. Repeat steps 2-3 until convergence.

Every line has a clear purpose!

Alternating Least Squares (spark.ml)

```scala
/**
 * :: DeveloperApi ::
 * Implementation of the ALS algorithm.
 */
@DeveloperApi
def train[ID: ClassTag]( // scalastyle:ignore
    ratings: RDD[Rating[ID]],
    rank: Int = 10,
    numUserBlocks: Int = 10,
    numItemBlocks: Int = 10,
    maxIter: Int = 10,
    regParam: Double = 1.0,
    implicitPrefs: Boolean = false,
    alpha: Double = 1.0,
    nonnegative: Boolean = false,
    intermediateRDDStorageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK,
    finalRDDStorageLevel: StorageLevel = StorageLevel.MEMORY_AND_DISK,
    checkpointInterval: Int = 10,
    seed: Long = 0L)(
    implicit ord: Ordering[ID]): (RDD[(ID, Array[Float])], RDD[(ID, Array[Float])]) = {
  require(intermediateRDDStorageLevel != StorageLevel.NONE,
    "ALS is not designed to run without persisting intermediate RDDs.")
  val sc = ratings.sparkContext
  val userPart = new ALSPartitioner(numUserBlocks)
  val itemPart = new ALSPartitioner(numItemBlocks)
  val userLocalIndexEncoder = new LocalIndexEncoder(userPart.numPartitions)
  val itemLocalIndexEncoder = new LocalIndexEncoder(itemPart.numPartitions)
  val solver = if (nonnegative) new NNLSSolver else new CholeskySolver
  val blockRatings = partitionRatings(ratings, userPart, itemPart)
    .persist(intermediateRDDStorageLevel)
  val (userInBlocks, userOutBlocks) =
    makeBlocks("user", blockRatings, userPart, itemPart, intermediateRDDStorageLevel)
  // materialize blockRatings and user blocks
  userOutBlocks.count()
  val swappedBlockRatings = blockRatings.map {
    case ((userBlockId, itemBlockId), RatingBlock(userIds, itemIds, localRatings)) =>
      ((itemBlockId, userBlockId), RatingBlock(itemIds, userIds, localRatings))
  }
  val (itemInBlocks, itemOutBlocks) =
    makeBlocks("item", swappedBlockRatings, itemPart, userPart, intermediateRDDStorageLevel)
  // materialize item blocks
  itemOutBlocks.count()
  val seedGen = new XORShiftRandom(seed)
  var userFactors = initialize(userInBlocks, rank, seedGen.nextLong())
  var itemFactors = initialize(itemInBlocks, rank, seedGen.nextLong())
  var previousCheckpointFile: Option[String] = None
  val shouldCheckpoint: Int => Boolean = (iter) =>
    sc.checkpointDir.isDefined && checkpointInterval != -1 && (iter % checkpointInterval == 0)
  val deletePreviousCheckpointFile: () => Unit = () =>
    previousCheckpointFile.foreach { file =>
      try {
        FileSystem.get(sc.hadoopConfiguration).delete(new Path(file), true)
      } catch {
        case e: IOException =>
          logWarning(s"Cannot delete checkpoint file $file:", e)
      }
    }
```

```scala
      srcIds += r.user
      dstIds += r.item
      ratings += r.rating
      this
    }

    /** Merges another [[RatingBlockBuilder]]. */
    def merge(other: RatingBlock[ID]): this.type = {
      size += other.srcIds.length
      srcIds ++= other.srcIds
      dstIds ++= other.dstIds
      ratings ++= other.ratings
      this
    }

    /** Builds a [[RatingBlock]]. */
    def build(): RatingBlock[ID] = {
      RatingBlock[ID](srcIds.result(), dstIds.result(), ratings.result())
    }
  }

  /**
   * Partitions raw ratings into blocks.
   *
   * @param ratings raw ratings
   * @param srcPart partitioner for src IDs
   * @param dstPart partitioner for dst IDs
   *
   * @return an RDD of rating blocks in the form of ((srcBlockId, dstBlockId), ratingBlock)
   */
  private def partitionRatings[ID: ClassTag](
      ratings: RDD[Rating[ID]],
      srcPart: Partitioner,
      dstPart: Partitioner): RDD[((Int, Int), RatingBlock[ID])] = {

    /* The implementation produces the same result as the following but generates less objec

    ratings.map { r =>
      ((srcPart.getPartition(r.user), dstPart.getPartition(r.item)), r)
    }.aggregateByKey(new RatingBlockBuilder)(
        seqOp = (b, r) => b.add(r),
        combOp = (b0, b1) => b0.merge(b1.build()))
      .mapValues(_.build())
    */

    val numPartitions = srcPart.numPartitions * dstPart.numPartitions
    ratings.mapPartitions { iter =>
      val builders = Array.fill(numPartitions)(new RatingBlockBuilder[ID])
      iter.flatMap { r =>
        val srcBlockId = srcPart.getPartition(r.user)
        val dstBlockId = dstPart.getPartition(r.item)
        val idx = srcBlockId + srcPart.numPartitions * dstBlockId
        val builder = builders(idx)
        builder.add(r)
        if (builder.size >= 2048) { // 2048 * (3 * 4) = 24k
          builders(idx) = new RatingBlockBuilder
          Iterator.single(((srcBlockId, dstBlockId), builder.build()))
```

```scala
        } else {
          Iterator.empty
        }
      } ++ {
        builders.view.zipWithIndex.filter(_._1.size > 0).map { case (block, idx) =>
          val srcBlockId = idx % srcPart.numPartitions
          val dstBlockId = idx / srcPart.numPartitions
          ((srcBlockId, dstBlockId), block.build())
        }
      }
    }.groupByKey().mapValues { blocks =>
      val builder = new RatingBlockBuilder[ID]
      blocks.foreach(builder.merge)
      builder.build()
    }.setName("ratingBlocks")
  }

  /**
   * Builder for uncompressed in-blocks of (srcId, dstEncodedIndex, rating) tuples.
   * @param encoder encoder for dst indices
   */
  private[recommendation] class UncompressedInBlockBuilder[@specialized(Int, Long) ID: ClassTag](
      encoder: LocalIndexEncoder)(
      implicit ord: Ordering[ID]) {

    private val srcIds = mutable.ArrayBuilder.make[ID]
    private val dstEncodedIndices = mutable.ArrayBuilder.make[Int]
    private val ratings = mutable.ArrayBuilder.make[Float]

    /**
     * Adds a dst block of (srcId, dstLocalIndex, rating) tuples.
     *
     * @param dstBlockId dst block ID
     * @param srcIds original src IDs
     * @param dstLocalIndices dst local indices
     * @param ratings ratings
     */
    def add(
        dstBlockId: Int,
        srcIds: Array[ID],
        dstLocalIndices: Array[Int],
        ratings: Array[Float]): this.type = {
      val sz = srcIds.length
      require(dstLocalIndices.length == sz)
      require(ratings.length == sz)
      this.srcIds ++= srcIds
      this.ratings ++= ratings
      var j = 0
      while (j < sz) {
        this.dstEncodedIndices += encoder.encode(dstBlockId, dstLocalIndices(j))
        j += 1
      }
      this
    }

    /** Builds a [[UncompressedInBlock]]. */
    def build(): UncompressedInBlock[ID] = {
```

Alternating Least Squares (spark.ml)

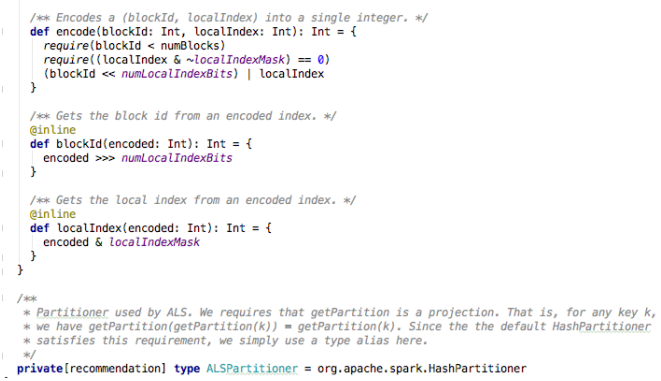
```
if (implicitPrefs) {
  for (iter <- 1 to maxIter) {
    userFactors.setName(s"userFactors-$iter").persist(intermediateRDDStorageLevel)
    val previousItemFactors = itemFactors
    itemFactors = computeFactors(userFactors, userOutBlocks, itemInBlocks, rank, regPara
      userLocalIndexEncoder, implicitPrefs, alpha, solver)
    previousItemFactors.unpersist()
    itemFactors.setName(s"itemFactors-$iter").persist(intermediateRDDStorageLevel)
    // TODO: Generalize PeriodicGraphCheckpointer and use it here.
    if (shouldCheckpoint(iter)) {
      itemFactors.checkpoint() // itemFactors gets materialized in computeFactors.
    }
    val previousUserFactors = userFactors
    userFactors = computeFactors(itemFactors, itemOutBlocks, userInBlocks, rank, regPara
      itemLocalIndexEncoder, implicitPrefs, alpha, solver)
    if (shouldCheckpoint(iter)) {
      deletePreviousCheckpointFile()
      previousCheckpointFile = itemFactors.getCheckpointFile
    }
    previousUserFactors.unpersist()
  }
} else {
  for (iter <- 0 until maxIter) {
    itemFactors = computeFactors(userFactors, userOutBlocks, itemInBlocks, rank, regPara
      userLocalIndexEncoder, solver = solver)
    if (shouldCheckpoint(iter)) {
      itemFactors.checkpoint()
      itemFactors.count() // checkpoint item factors and cut lineage
      deletePreviousCheckpointFile()
      previousCheckpointFile = itemFactors.getCheckpointFile
    }
    userFactors = computeFactors(itemFactors, itemOutBlocks, userInBlocks, rank, regPara
      itemLocalIndexEncoder, solver = solver)
  }
}
val userIdAndFactors = userInBlocks
  .mapValues(_.srcIds)
  .join(userFactors)
  .mapPartitions({ items =>
    items.flatMap { case (_, (ids, factors)) =>
      ids.view.zip(factors)
    }
  // Preserve the partitioning because IDs are consistent with the partitioners in userI
  // and userFactors.
  }, preservesPartitioning = true)
  .setName("userFactors")
  .persist(finalRDDStorageLevel)
val itemIdAndFactors = itemInBlocks
  .mapValues(_.srcIds)
  .join(itemFactors)
  .mapPartitions({ items =>
    items.flatMap { case (_, (ids, factors)) =>
      ids.view.zip(factors)
    }
  }, preservesPartitioning = true)
  .setName("itemFactors")
  .persist(finalRDDStorageLevel)
if (finalRDDStorageLevel != StorageLevel.NONE) {
  userIdAndFactors.count()
  itemFactors.unpersist()
  itemIdAndFactors.count()
  userInBlocks.unpersist()
  userOutBlocks.unpersist()
  itemInBlocks.unpersist()
  itemOutBlocks.unpersist()
  blockRatings.unpersist()
}
(userIdAndFactors, itemIdAndFactors)
}

/**
 * Factor block that stores factors (Array[Float]) in an Array.
 */
private type FactorBlock = Array[Array[Float]]

/**
 * Out-link block that stores, for each dst (item/user) block, which src (user/item) factors to
 * send. For example, outLinkBlock(0) contains the local indices (not the original src IDs) of t
 * src factors in this block to send to dst block 0.
 */
private type OutBlock = Array[Array[Int]]

/**
 * In-link block for computing src (user/item) factors. This includes the original src IDs
 * of the elements within this block as well as encoded dst (item/user) indices and correspondi
 * ratings. The dst indices are in the form of (blockId, localIndex), which are not the original
 * dst IDs. To compute src factors, we expect receiving dst factors that match the dst indices.
 * For example, if we have an in-link record
 *
 * {srcId: 0, dstBlockId: 2, dstLocalIndex: 3, rating: 5.0},
 *
 * and assume that the dst factors are stored as dstFactors: Map[Int, Array[Array[Float]]], whic
 * is a blockId to dst factors map, the corresponding dst factor of the record is dstFactor(2)(3
 *
 * We use a CSC-like (compressed sparse column) format to store the in-link information. So we c
 * compute src factors one after another using only one normal equation instance.
 *
 * @param srcIds src ids (ordered)
 * @param dstPtrs dst pointers. Elements in range [dstPtrs(i), dstPtrs(i+1)) of dst indices and
 *                ratings are associated with srcIds(i).
 * @param dstEncodedIndices encoded dst indices
 * @param ratings ratings
 *
 * @see [[LocalIndexEncoder]]
 */
private[recommendation] case class InBlock[@specialized(Int, Long) ID: ClassTag](
    srcIds: Array[ID],
    dstPtrs: Array[Int],
    dstEncodedIndices: Array[Int],
    ratings: Array[Float]) {
  /** Size of the block. */
  def size: Int = ratings.length
  require(dstEncodedIndices.length == size)
  require(dstPtrs.length == srcIds.length + 1)
}

/**
 * Initializes factors randomly given the in-link blocks.
 *
 * @param inBlocks in-link blocks
 * @param rank rank
 * @return initialized factor blocks
 */
private def initialize[ID](
    inBlocks: RDD[(Int, InBlock[ID])],
    rank: Int,
    seed: Long): RDD[(Int, FactorBlock)] = {
  // Choose a unit vector uniformly at random from the unit sphere, but from the
  // "first quadrant" where all elements are nonnegative. This can be done by choosing
  // elements distributed as Normal(0,1) and taking the absolute value, and then normalizing.
  // This appears to create factorizations that have a slightly better reconstruction
  // (<1%) compared picking elements uniformly at random in [0,1].
  inBlocks.map { case (srcBlockId, inBlock) =>
    val random = new XORShiftRandom(byteswap64(seed ^ srcBlockId))
    val factors = Array.fill(inBlock.srcIds.length) {
      val factor = Array.fill(rank)(random.nextGaussian().toFloat)
      val nrm = blas.snrm2(rank, factor, 1)
      blas.sscal(rank, 1.0f / nrm, factor, 1)
      factor
    }
    (srcBlockId, factors)
  }
}

/**
 * A rating block that contains src IDs, dst IDs, and ratings, stored in primitive arrays.
 */
private[recommendation] case class RatingBlock[@specialized(Int, Long) ID: ClassTag](
    srcIds: Array[ID],
    dstIds: Array[ID],
    ratings: Array[Float]) {
  /** Size of the block. */
  def size: Int = srcIds.length
  require(dstIds.length == srcIds.length)
  require(ratings.length == srcIds.length)
}

/**
 * Builder for [[RatingBlock]]. [[mutable.ArrayBuilder]] is used to avoid boxing/unboxing.
 */
private[recommendation] class RatingBlockBuilder[@specialized(Int, Long) ID: ClassTag]
  extends Serializable {

  private val srcIds = mutable.ArrayBuilder.make[ID]
  private val dstIds = mutable.ArrayBuilder.make[ID]
  private val ratings = mutable.ArrayBuilder.make[Float]
  var size = 0

  /** Adds a rating. */
  def add(r: Rating[ID]): this.type = {
    size += 1
```

Alternating Least Squares (spark.ml)

```
    new UncompressedInBlock(srcIds.result(), dstEncodedIndices.result(), ratings.result())
  }
}

/**
 * A block of (srcId, dstEncodedIndex, rating) tuples stored in primitive arrays.
 */
private[recommendation] class UncompressedInBlock[@specialized(Int, Long) ID: ClassTag](
    val srcIds: Array[ID],
    val dstEncodedIndices: Array[Int],
    val ratings: Array[Float])(
    implicit ord: Ordering[ID]) {

  /** Size the of block. */
  def length: Int = srcIds.length

  /**
   * Compresses the block into an [[InBlock]]. The algorithm is the same as converting a
   * sparse matrix from coordinate list (COO) format into compressed sparse column (CSC) format.
   * Sorting is done using Spark's built-in Timsort to avoid generating too many objects.
   */
  def compress(): InBlock[ID] = {
    val sz = length
    assert(sz > 0, "Empty in-link block should not exist.")
    sort()
    val uniqueSrcIdsBuilder = mutable.ArrayBuilder.make[ID]
    val dstCountsBuilder = mutable.ArrayBuilder.make[Int]
    var preSrcId = srcIds(0)
    uniqueSrcIdsBuilder += preSrcId
    var curCount = 1
    var i = 1
    var j = 0
    while (i < sz) {
      val srcId = srcIds(i)
      if (srcId != preSrcId) {
        uniqueSrcIdsBuilder += srcId
        dstCountsBuilder += curCount
        preSrcId = srcId
        j += 1
        curCount = 0
      }
      curCount += 1
      i += 1
    }
    dstCountsBuilder += curCount
    val uniqueSrcIds = uniqueSrcIdsBuilder.result()
    val numUniqueSrdIds = uniqueSrcIds.length
    val dstCounts = dstCountsBuilder.result()
    val dstPtrs = new Array[Int](numUniqueSrdIds + 1)
    var sum = 0
    i = 0
    while (i < numUniqueSrdIds) {
      sum += dstCounts(i)
      i += 1
      dstPtrs(i) = sum
    }
    InBlock(uniqueSrcIds, dstPtrs, dstEncodedIndices, ratings)
  }

  private def sort(): Unit = {
    val sz = length
    // Since there might be interleaved log messages, we insert a unique id for easy pairing.
    val sortId = Utils.random.nextInt()
    logDebug(s"Start sorting an uncompressed in-block of size $sz. (sortId = $sortId)")
    val start = System.nanoTime()
    val sorter = new Sorter(new UncompressedInBlockSort[ID])
    sorter.sort(this, 0, length, Ordering[KeyWrapper[ID]])
    val duration = (System.nanoTime() - start) / 1e9
    logDebug(s"Sorting took $duration seconds. (sortId = $sortId)")
  }
}

/**
 * A wrapper that holds a primitive key.
 *
 * @see [[UncompressedInBlockSort]]
 */
private class KeyWrapper[@specialized(Int, Long) ID: ClassTag](
    implicit ord: Ordering[ID]) extends Ordered[KeyWrapper[ID]] {

  var key: ID = _

  override def compare(that: KeyWrapper[ID]): Int = {
    ord.compare(key, that.key)
  }

  def setKey(key: ID): this.type = {
    this.key = key
    this
  }
}

/**
 * [[SortDataFormat]] of [[UncompressedInBlock]] used by [[Sorter]].
 */
private class UncompressedInBlockSort[@specialized(Int, Long) ID: ClassTag](
    implicit ord: Ordering[ID])
  extends SortDataFormat[KeyWrapper[ID], UncompressedInBlock[ID]] {

  override def newKey(): KeyWrapper[ID] = new KeyWrapper()

  override def getKey(
      data: UncompressedInBlock[ID],
      pos: Int,
      reuse: KeyWrapper[ID]): KeyWrapper[ID] = {
    if (reuse == null) {
      new KeyWrapper().setKey(data.srcIds(pos))
    } else {
      reuse.setKey(data.srcIds(pos))
    }
  }

  override def getKey(
      data: UncompressedInBlock[ID],
      pos: Int): KeyWrapper[ID] = {
    getKey(data, pos, null)
  }

  private def swapElements[@specialized(Int, Float) T](
      data: Array[T],
      pos0: Int,
      pos1: Int): Unit = {
    val tmp = data(pos0)
    data(pos0) = data(pos1)
    data(pos1) = tmp
  }

  override def swap(data: UncompressedInBlock[ID], pos0: Int, pos1: Int): Unit = {
    swapElements(data.srcIds, pos0, pos1)
    swapElements(data.dstEncodedIndices, pos0, pos1)
    swapElements(data.ratings, pos0, pos1)
  }

  override def copyRange(
      src: UncompressedInBlock[ID],
      srcPos: Int,
      dst: UncompressedInBlock[ID],
      dstPos: Int,
      length: Int): Unit = {
    System.arraycopy(src.srcIds, srcPos, dst.srcIds, dstPos, length)
    System.arraycopy(src.dstEncodedIndices, srcPos, dst.dstEncodedIndices, dstPos, length)
    System.arraycopy(src.ratings, srcPos, dst.ratings, dstPos, length)
  }

  override def allocate(length: Int): UncompressedInBlock[ID] = {
    new UncompressedInBlock(
      new Array[ID](length), new Array[Int](length), new Array[Float](length))
  }

  override def copyElement(
      src: UncompressedInBlock[ID],
      srcPos: Int,
      dst: UncompressedInBlock[ID],
      dstPos: Int): Unit = {
    dst.srcIds(dstPos) = src.srcIds(srcPos)
    dst.dstEncodedIndices(dstPos) = src.dstEncodedIndices(srcPos)
    dst.ratings(dstPos) = src.ratings(srcPos)
  }
}

/**
 * Creates in-blocks and out-blocks from rating blocks.
 * @param prefix prefix for in/out-block names
 * @param ratingBlocks rating blocks
 * @param srcPart partitioner for src IDs
 * @param dstPart partitioner for dst IDs
 * @return (in-blocks, out-blocks)
 */
private def makeBlocks[ID: ClassTag](
    prefix: String,
    ratingBlocks: RDD[((Int, Int), RatingBlock[ID])],
```

Alternating Least Squares (spark.ml)

```
    srcPart: Partitioner,
    dstPart: Partitioner,
    storageLevel: StorageLevel)(
    implicit srcOrd: Ordering[ID]): (RDD[(Int, InBlock[ID])], RDD[(Int, OutBlock)]) = {
  val inBlocks = ratingBlocks.map {
    case ((srcBlockId, dstBlockId), RatingBlock(srcIds, dstIds, ratings)) =>
      // The implementation is a faster version of
      // val dstIdToLocalIndex = dstIds.toSet.toSeq.sorted.zipWithIndex.toMap
      val start = System.nanoTime()
      val dstIdSet = new OpenHashSet[ID](1 << 20)
      dstIds.foreach(dstIdSet.add)
      val sortedDstIds = new Array[ID](dstIdSet.size)
      var i = 0
      var pos = dstIdSet.nextPos(0)
      while (pos != -1) {
        sortedDstIds(i) = dstIdSet.getValue(pos)
        pos = dstIdSet.nextPos(pos + 1)
        i += 1
      }
      assert(i == dstIdSet.size)
      Sorting.quickSort(sortedDstIds)
      val dstIdToLocalIndex = new OpenHashMap[ID, Int](sortedDstIds.length)
      i = 0
      while (i < sortedDstIds.length) {
        dstIdToLocalIndex.update(sortedDstIds(i), i)
        i += 1
      }
      logDebug(
        "Converting to local indices took " + (System.nanoTime() - start) / 1e9 + " secon
      val dstLocalIndices = dstIds.map(dstIdToLocalIndex.apply)
      (srcBlockId, (dstBlockId, srcIds, dstLocalIndices, ratings))
  }.groupByKey(new ALSPartitioner(srcPart.numPartitions))
    .mapValues { iter =>
      val builder =
        new UncompressedInBlockBuilder[ID](new LocalIndexEncoder(dstPart.numPartitions))
      iter.foreach { case (dstBlockId, srcIds, dstLocalIndices, ratings) =>
        builder.add(dstBlockId, srcIds, dstLocalIndices, ratings)
      }
      builder.build().compress()
    }.setName(prefix + "InBlocks")
    .persist(storageLevel)
  val outBlocks = inBlocks.mapValues { case InBlock(srcIds, dstPtrs, dstEncodedIndices, _
    val encoder = new LocalIndexEncoder(dstPart.numPartitions)
    val activeIds = Array.fill(dstPart.numPartitions)(mutable.ArrayBuilder.make[Int])
    var i = 0
    val seen = new Array[Boolean](dstPart.numPartitions)
    while (i < srcIds.length) {
      var j = dstPtrs(i)
      ju.Arrays.fill(seen, false)
      while (j < dstPtrs(i + 1)) {
        val dstBlockId = encoder.blockId(dstEncodedIndices(j))
        if (!seen(dstBlockId)) {
          activeIds(dstBlockId) += i // add the local index in this out-block
          seen(dstBlockId) = true
        }
        j += 1
      }
      i += 1
    }
    activeIds.map { x =>
      x.result()
    }
  }.setName(prefix + "OutBlocks")
    .persist(storageLevel)
  (inBlocks, outBlocks)
}

/**
 * Compute dst factors by constructing and solving least square problems.
 *
 * @param srcFactorBlocks src factors
 * @param srcOutBlocks src out-blocks
 * @param dstInBlocks dst in-blocks
 * @param rank rank
 * @param regParam regularization constant
 * @param srcEncoder encoder for src local indices
 * @param implicitPrefs whether to use implicit preference
 * @param alpha the alpha constant in the implicit preference formulation
 * @param solver solver for least squares problems
 *
 * @return dst factors
 */
private def computeFactors[ID](
    srcFactorBlocks: RDD[(Int, FactorBlock)],
    srcOutBlocks: RDD[(Int, OutBlock)],
    dstInBlocks: RDD[(Int, InBlock[ID])],
    rank: Int,
    regParam: Double,
    srcEncoder: LocalIndexEncoder,
    implicitPrefs: Boolean = false,
    alpha: Double = 1.0,
    solver: LeastSquaresNESolver): RDD[(Int, FactorBlock)] = {
  val numSrcBlocks = srcFactorBlocks.partitions.length
  val YtY = if (implicitPrefs) Some(computeYtY(srcFactorBlocks, rank)) else None
  val srcOut = srcOutBlocks.join(srcFactorBlocks).flatMap {
    case (srcBlockId, (srcOutBlock, srcFactors)) =>
      srcOutBlock.view.zipWithIndex.map { case (activeIndices, dstBlockId) =>
        (dstBlockId, (srcBlockId, activeIndices.map(idx => srcFactors(idx))))
      }
  }
  val merged = srcOut.groupByKey(new ALSPartitioner(dstInBlocks.partitions.length))
  dstInBlocks.join(merged).mapValues {
    case (InBlock(dstIds, srcPtrs, srcEncodedIndices, ratings), srcFactors) =>
      val sortedSrcFactors = new Array[FactorBlock](numSrcBlocks)
      srcFactors.foreach { case (srcBlockId, factors) =>
        sortedSrcFactors(srcBlockId) = factors
      }
      val dstFactors = new Array[Array[Float]](dstIds.length)
      var j = 0
      val ls = new NormalEquation(rank)
      while (j < dstIds.length) {
        ls.reset()
        if (implicitPrefs) {
          ls.merge(YtY.get)
        }
        var i = srcPtrs(j)
        var numExplicits = 0
        while (i < srcPtrs(j + 1)) {
          val encoded = srcEncodedIndices(i)
          val blockId = srcEncoder.blockId(encoded)
          val localIndex = srcEncoder.localIndex(encoded)
          val srcFactor = sortedSrcFactors(blockId)(localIndex)
          val rating = ratings(i)
          if (implicitPrefs) {
            // Extension to the original paper to handle b < 0. confidence is a function of |b|
            // instead so that it is never negative. c1 is confidence - 1.0.
            val c1 = alpha * math.abs(rating)
            // For rating <= 0, the corresponding preference is 0. So the term below is only added
            // for rating > 0. Because YtY is already added, we need to adjust the scaling here.
            if (rating > 0) {
              numExplicits += 1
              ls.add(srcFactor, (c1 + 1.0) / c1, c1)
            }
          } else {
            ls.add(srcFactor, rating)
            numExplicits += 1
          }
          i += 1
        }
        // Weight lambda by the number of explicit ratings based on the ALS-WR paper.
        dstFactors(j) = solver.solve(ls, numExplicits * regParam)
        j += 1
      }
      dstFactors
  }
}

/**
 * Computes the Gramian matrix of user or item factors, which is only used in implicit preference.
 * Caching of the input factors is handled in [[ALS#train]].
 */
private def computeYtY(factorBlocks: RDD[(Int, FactorBlock)], rank: Int): NormalEquation = {
  factorBlocks.values.aggregate(new NormalEquation(rank))(
    seqOp = (ne, factors) => {
      factors.foreach(ne.add(_, 0.0))
      ne
    },
    combOp = (ne1, ne2) => ne1.merge(ne2))
}

/**
 * Encoder for storing (blockId, localIndex) into a single integer.
 *
 * We use the leading bits (including the sign bit) to store the block id and the rest to store
 * the local index. This is based on the assumption that users/items are approximately evenly
 * partitioned. With this assumption, we should be able to encode two billion distinct values.
 *
 * @param numBlocks number of blocks
 */
private[recommendation] class LocalIndexEncoder(numBlocks: Int) extends Serializable {
  require(numBlocks > 0, s"numBlocks must be positive but found $numBlocks.")

  private[this] final val numLocalIndexBits =
    math.min(java.lang.Integer.numberOfLeadingZeros(numBlocks - 1), 31)
  private[this] final val localIndexMask = (1 << numLocalIndexBits) - 1
```



```
  /** Encodes a (blockId, localIndex) into a single integer. */
  def encode(blockId: Int, localIndex: Int): Int = {
    require(blockId < numBlocks)
    require((localIndex & ~localIndexMask) == 0)
    (blockId << numLocalIndexBits) | localIndex
  }

  /** Gets the block id from an encoded index. */
  @inline
  def blockId(encoded: Int): Int = {
    encoded >>> numLocalIndexBits
  }

  /** Gets the local index from an encoded index. */
  @inline
  def localIndex(encoded: Int): Int = {
    encoded & localIndexMask
  }
}

/**
 * Partitioner used by ALS. We requires that getPartition is a projection. That is, for any key k,
 * we have getPartition(getPartition(k)) = getPartition(k). Since the the default HashPartitioner
 * satisfies this requirement, we simply use a type alias here.
 */
private[recommendation] type ALSPartitioner = org.apache.spark.HashPartitioner
```

- 25 lines' worth of algorithm…
- …mixed with 800 lines of performance code

Alternating Least Squares (in R)

```
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;
}
```

Alternating Least Squares (in ~~R~~)

(in SystemML's subset of R)

```
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 can compile and run this algorithm at scale
- No additional performance code needed!

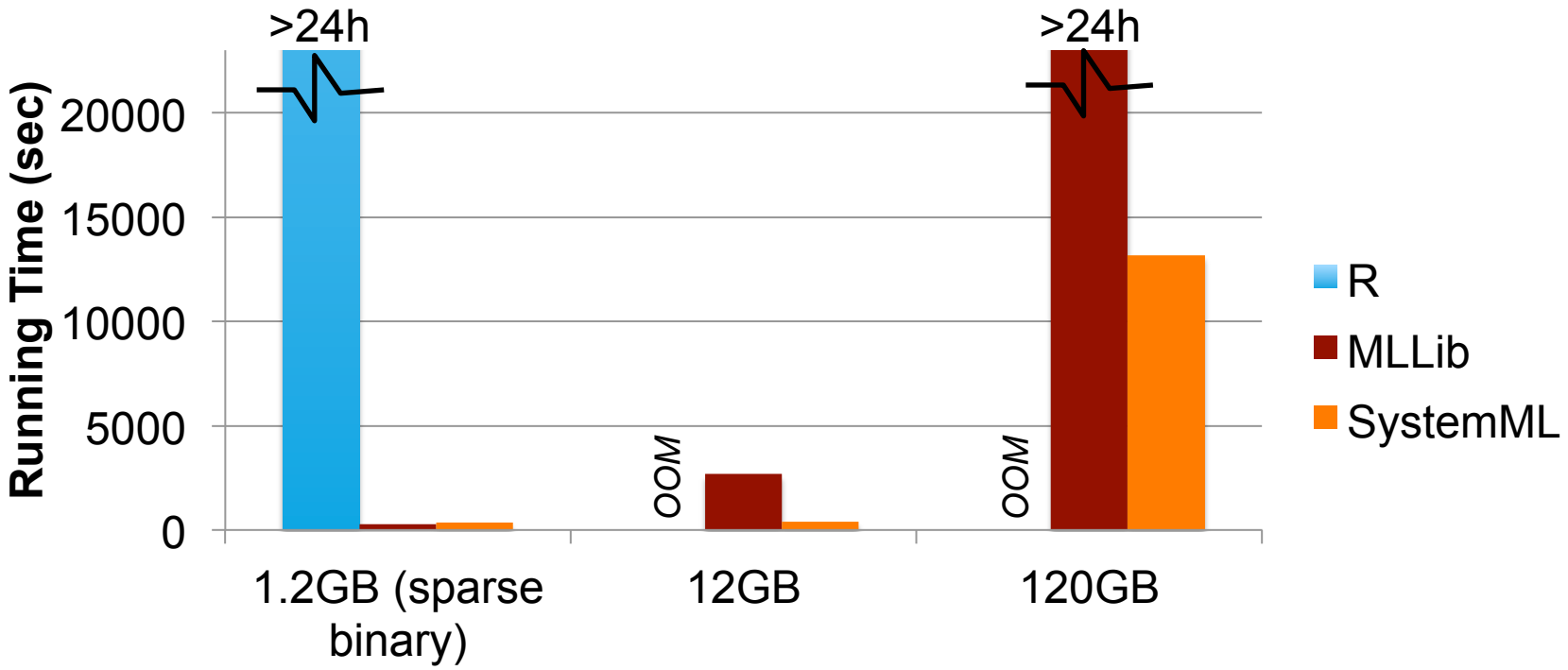
How fast does it run?

Running time comparisons between machine learning algorithms are problematic

- Different, equally-valid answers
- Different convergence rates on different data

- But we'll do one anyway

Performance Comparison: ALS



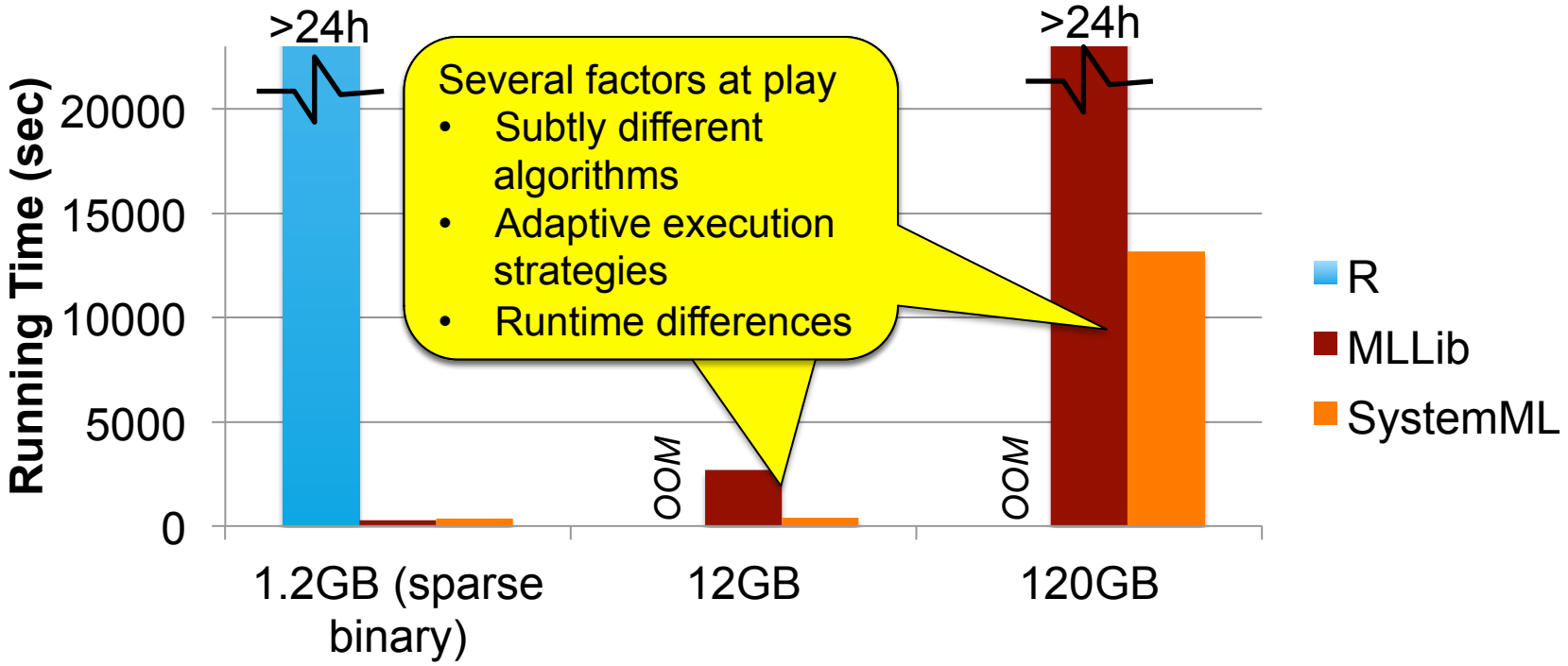
Details: Synthetic data, 0.01 sparsity, 10^5 products × {10^5,10^6,10^7} users. Data generated by multiplying two rank-50 matrices of normally-distributed data, sampling from the resulting product, then adding Gaussian noise. Cluster of 6 servers with 12 cores and 96GB of memory per server. Number of iterations tuned so that all algorithms produce comparable result quality.

Takeaway Points

- SystemML runs the R script in parallel
  - Same answer as original R script
  - Performance is comparable to a low-level RDD-based implementation

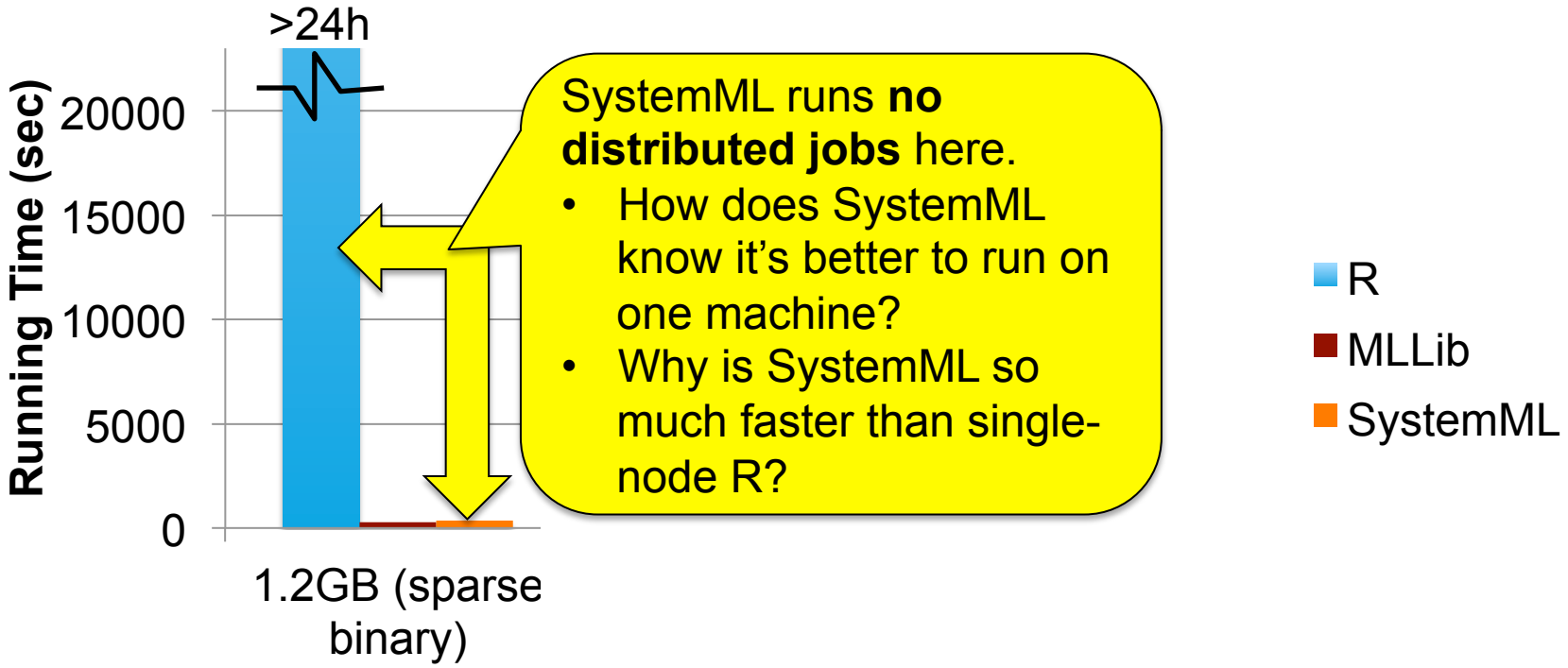
- How does SystemML achieve this result?

Performance Comparison: ALS

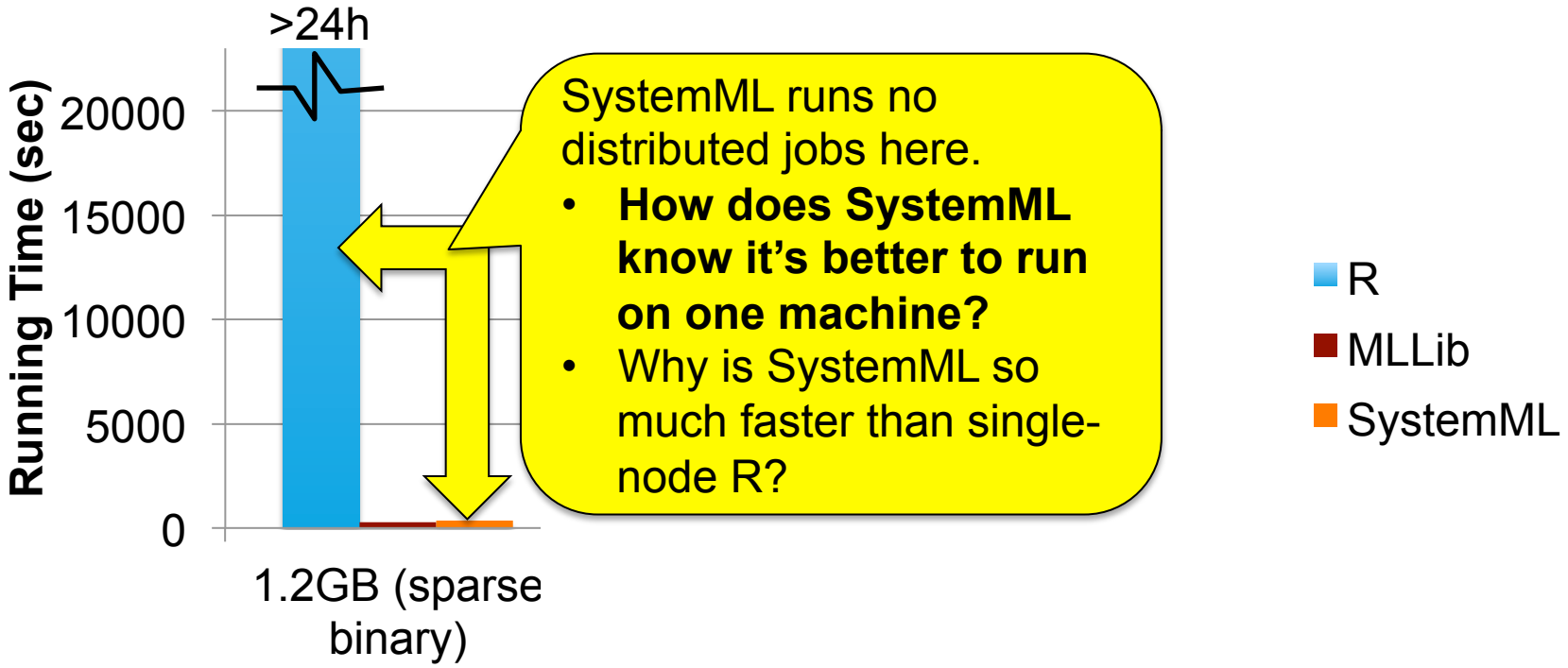


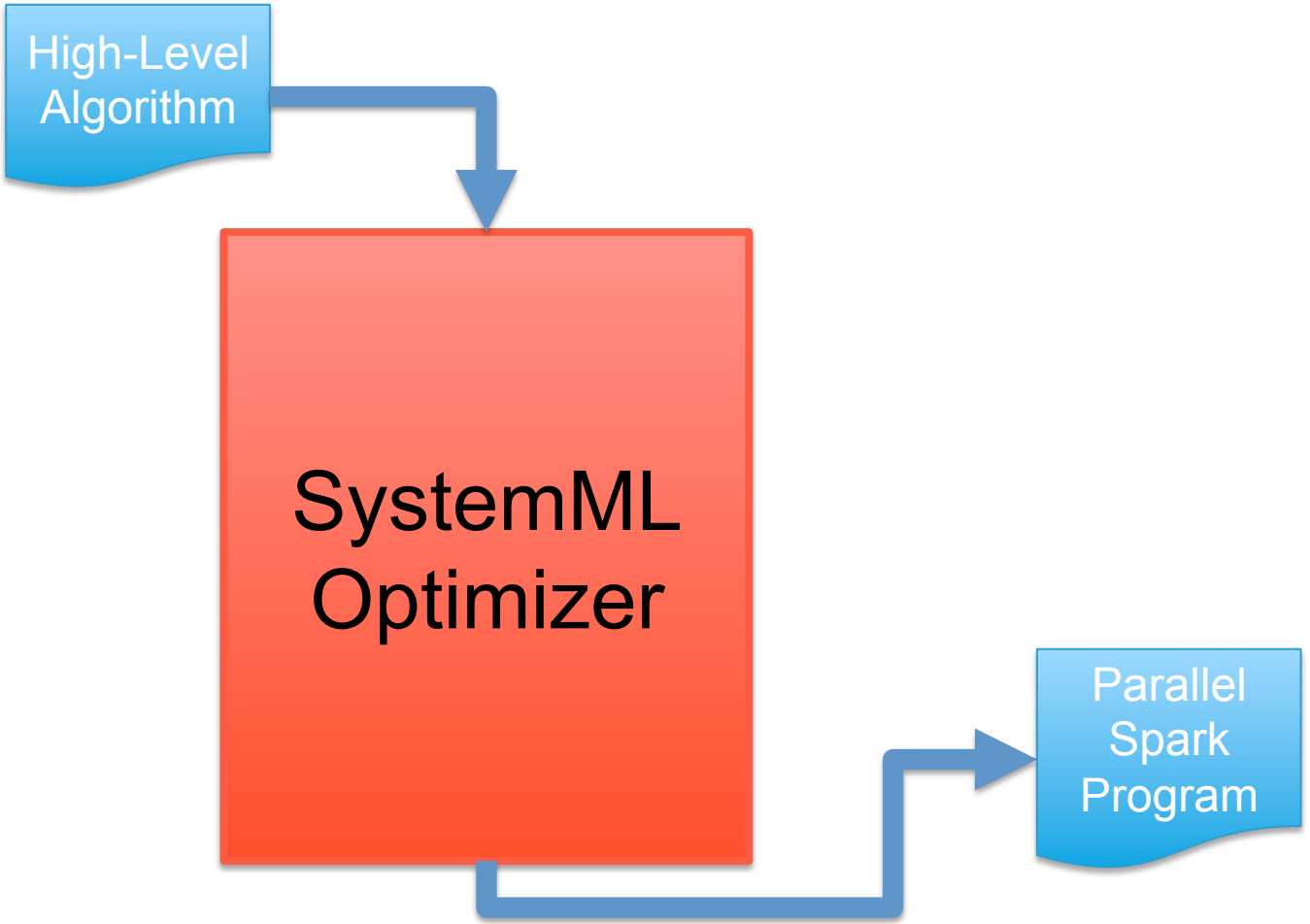
Details: Synthetic data, 0.01 sparsity, 10^5 products × {10^5,10^6,10^7} users. Data generated by multiplying two rank-50 matrices of normally-distributed data, sampling from the resulting product, then adding Gaussian noise. Cluster of 6 servers with 12 cores and 96GB of memory per server. Number of iterations tuned so that all algorithms produce comparable result quality.

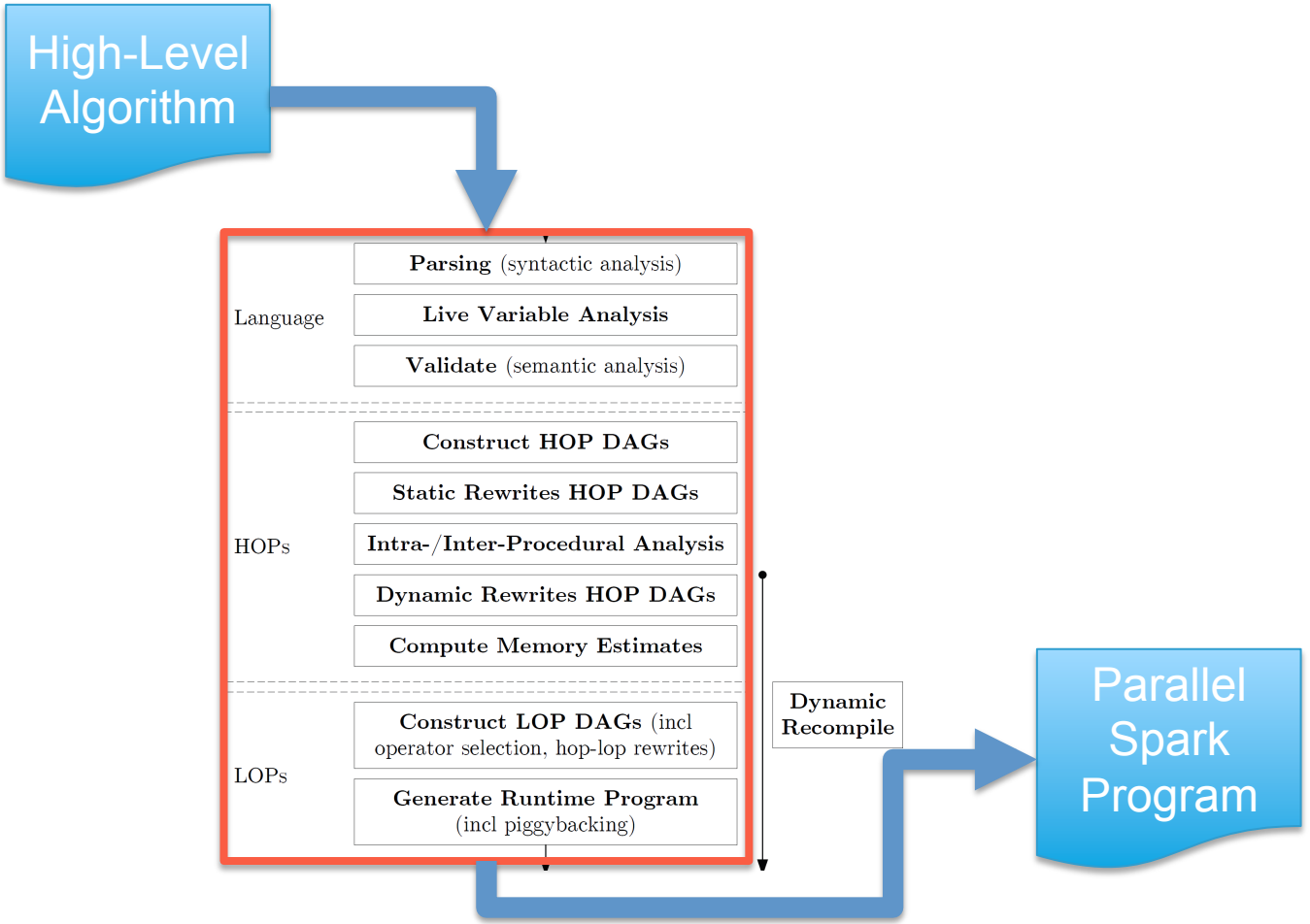
Questions We'll Focus On



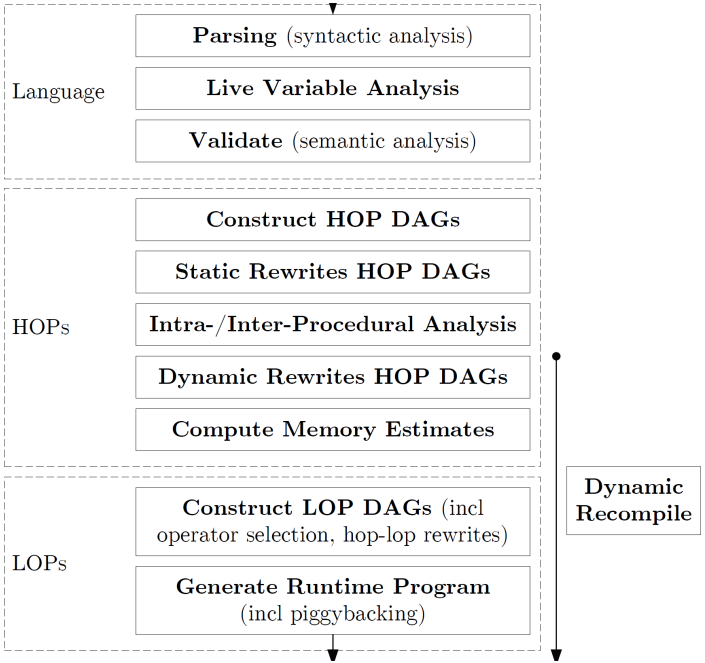
Questions We'll Focus On



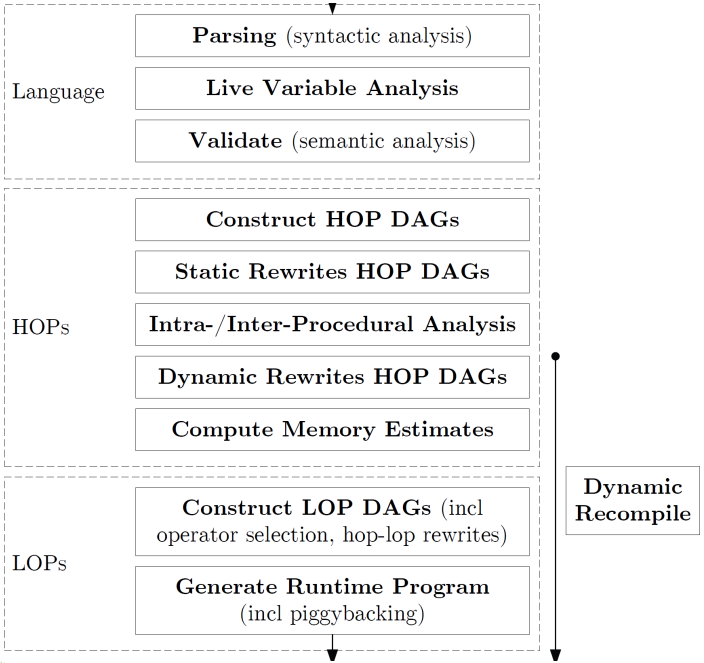

High-Level Algorithm
SystemML Optimizer
Parallel Spark Program


High-Level Algorithm
Language
Parsing (syntactic analysis)
Live Variable Analysis
Validate (semantic analysis)
HOPs
Construct HOP DAGs
Static Rewrites HOP DAGs
Intra-/Inter-Procedural Analysis
Dynamic Rewrites HOP DAGs
Compute Memory Estimates
LOPs
Construct LOP DAGs (incl operator selection, hop-lop rewrites)
Generate Runtime Program (incl piggybacking)
Dynamic Recompile
Parallel Spark Program

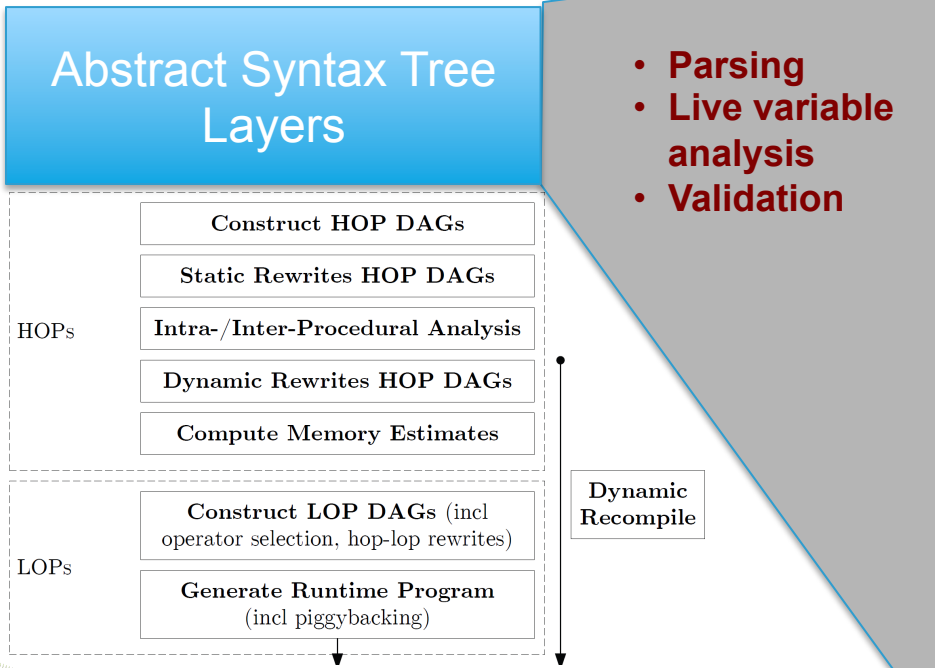
The SystemML Optimizer Stack



The SystemML Optimizer Stack



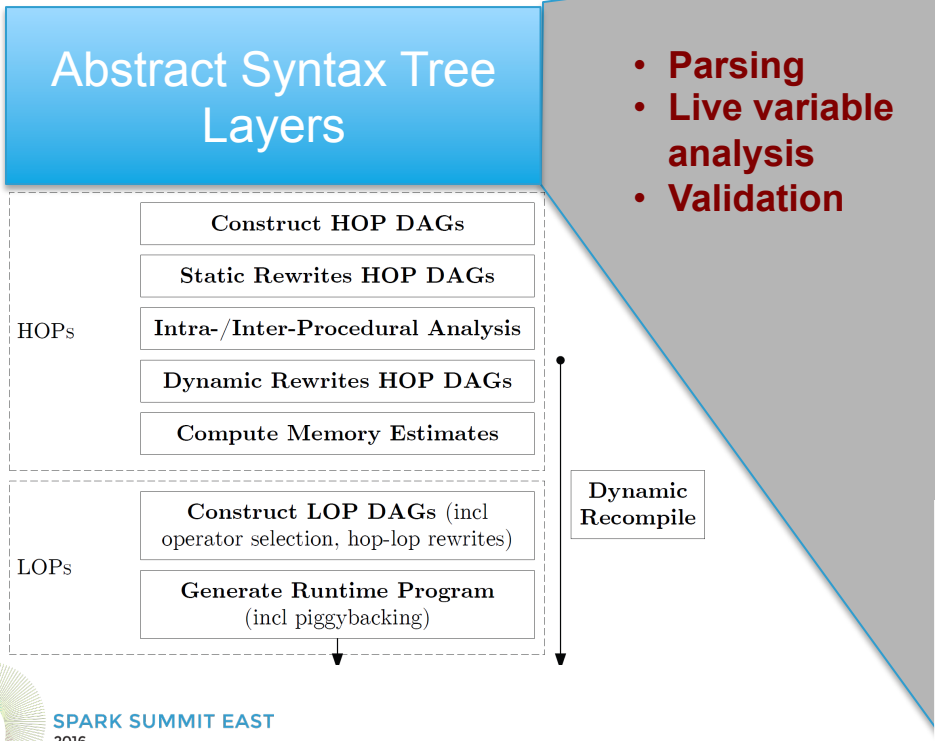
The SystemML Optimizer Stack



- **Parsing**
- **Live variable analysis**
- **Validation**

```
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;
}
```

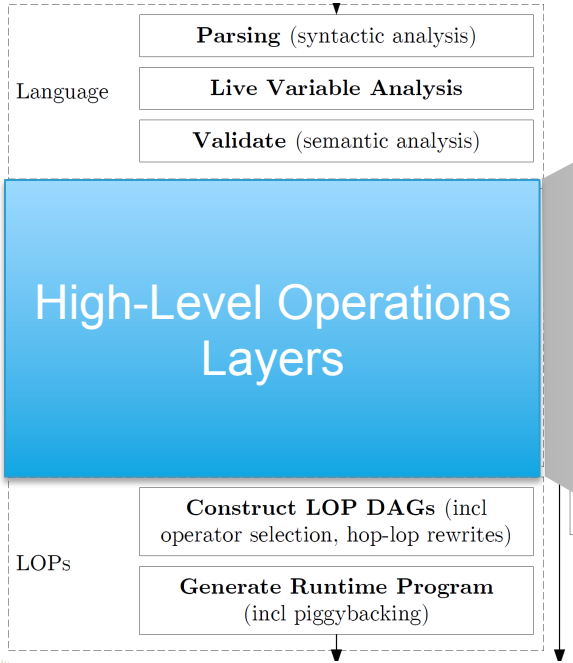
The SystemML Optimizer Stack



- **Parsing**
- **Live variable analysis**
- **Validation**

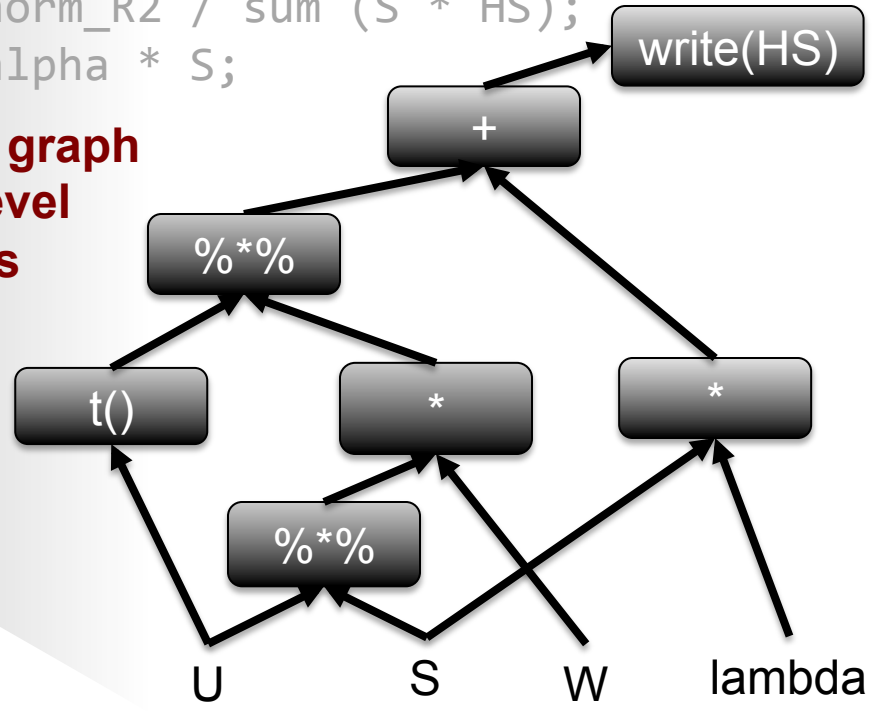
```
U = rand(nrow(X), r, min = -1.0, max = 1.0);
V = rand(r, ncol(X), min = -1.0, max = 1.0);
while(i < mi) {
  i = i + 1; ii = 1;
  if (is_U)
    G = (W * (U %*% V - X)) %*% t(V) + lambda * U;
  else
    G = t(U) %*% (W * (U %*% V - X)) + lambda * V;
  norm_G2 = sum(G ^ 2); norm_R2 = norm_G2;
  R = -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;
}
```

The SystemML Optimizer Stack




```
HS = t(U) %*% (W * (U %*% S)) + lambda * S;
alpha = norm_R2 / sum (S * HS);
V = V + alpha * S;
```

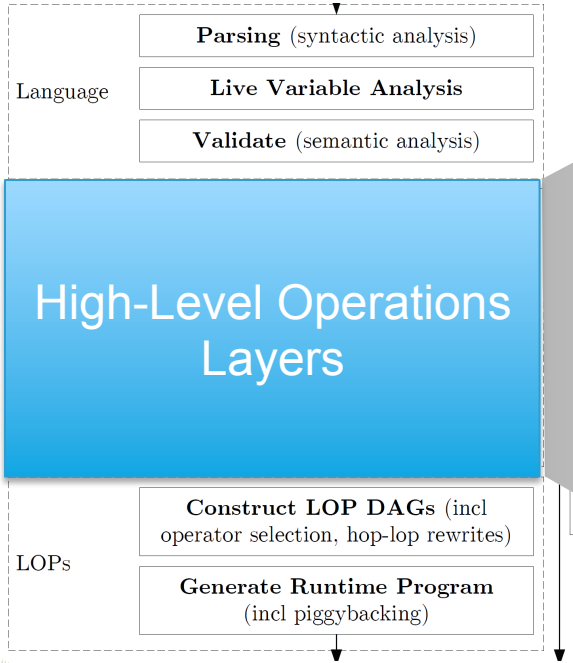
Construct graph of High-Level Operations (HOPs)

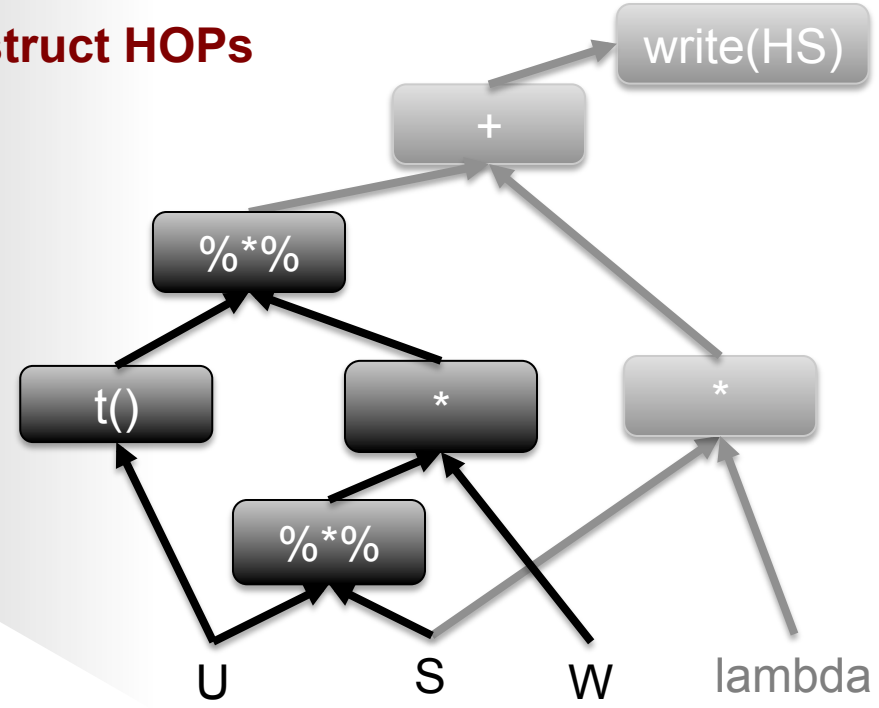


The SystemML Optimizer Stack

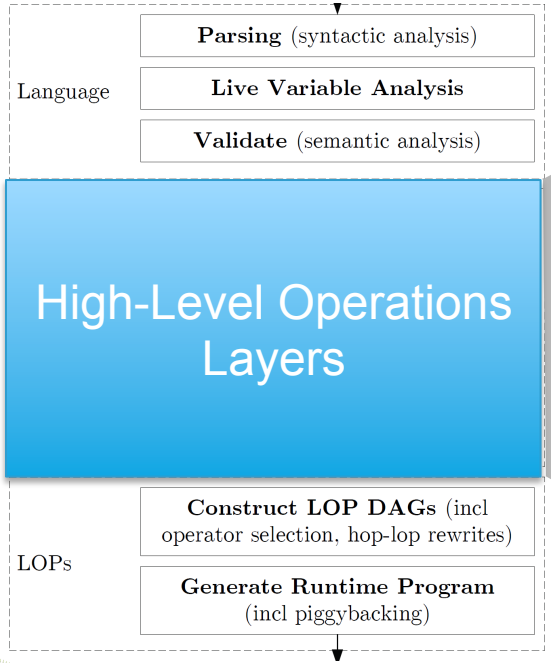
```
HS = t(U) %*% (W * (U %*% S)) + lambda * S;
```

- **Construct HOPs**



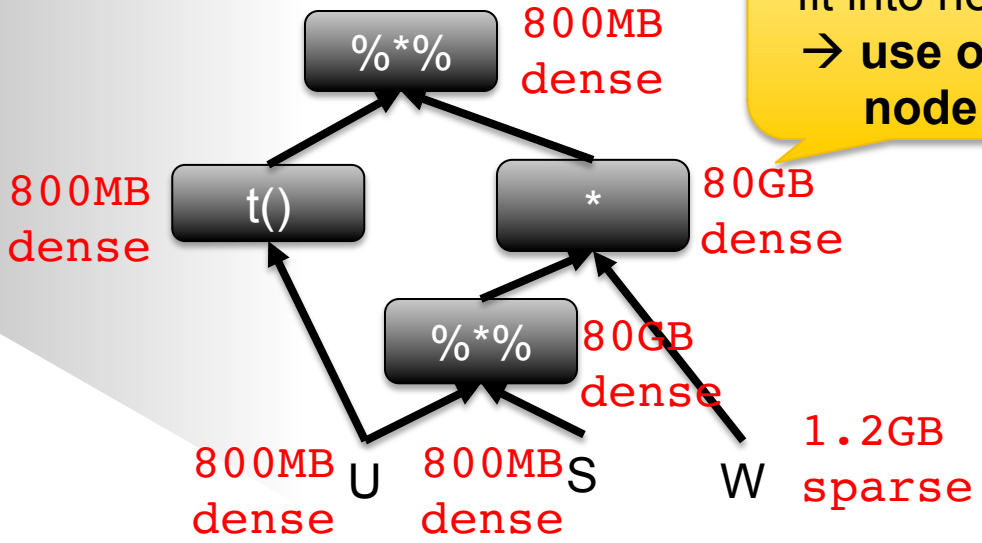


The SystemML Optimizer Stack

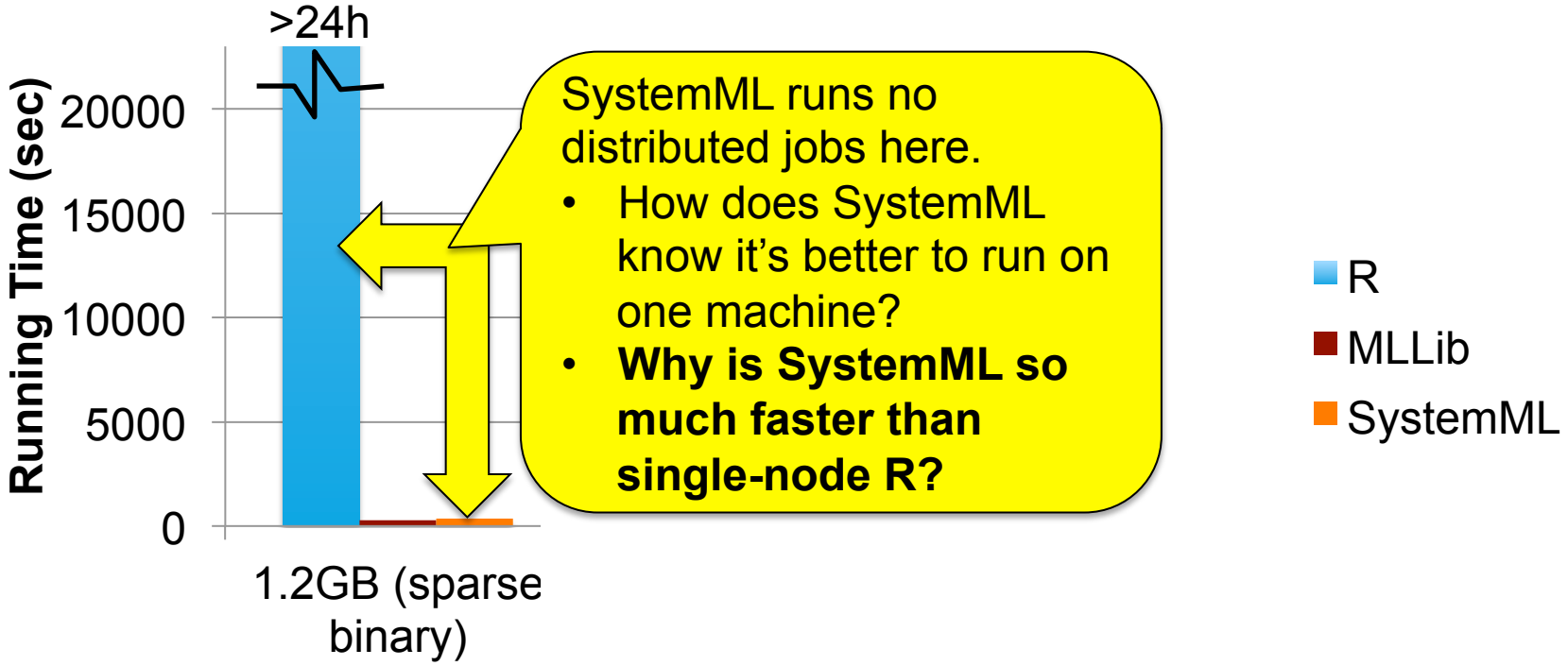



```
t(U) %*% (W * (U %*% S))
```

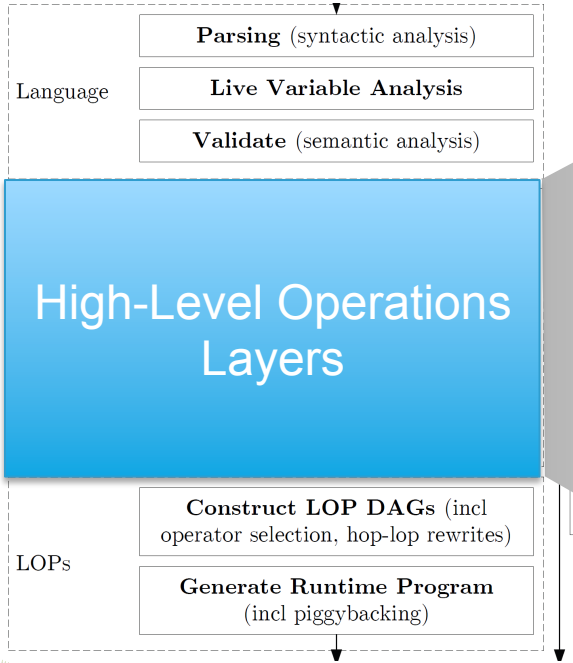
- **Construct HOPs**
- **Propagate statistics**
- **Determine distributed operation**



Questions We'll Focus On

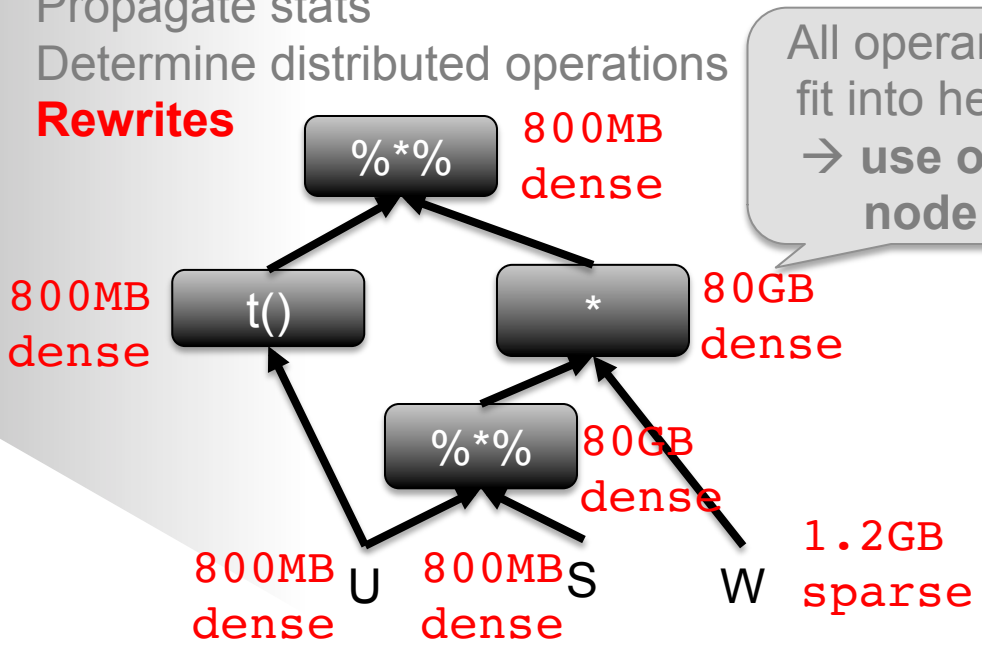


The SystemML Optimizer Stack



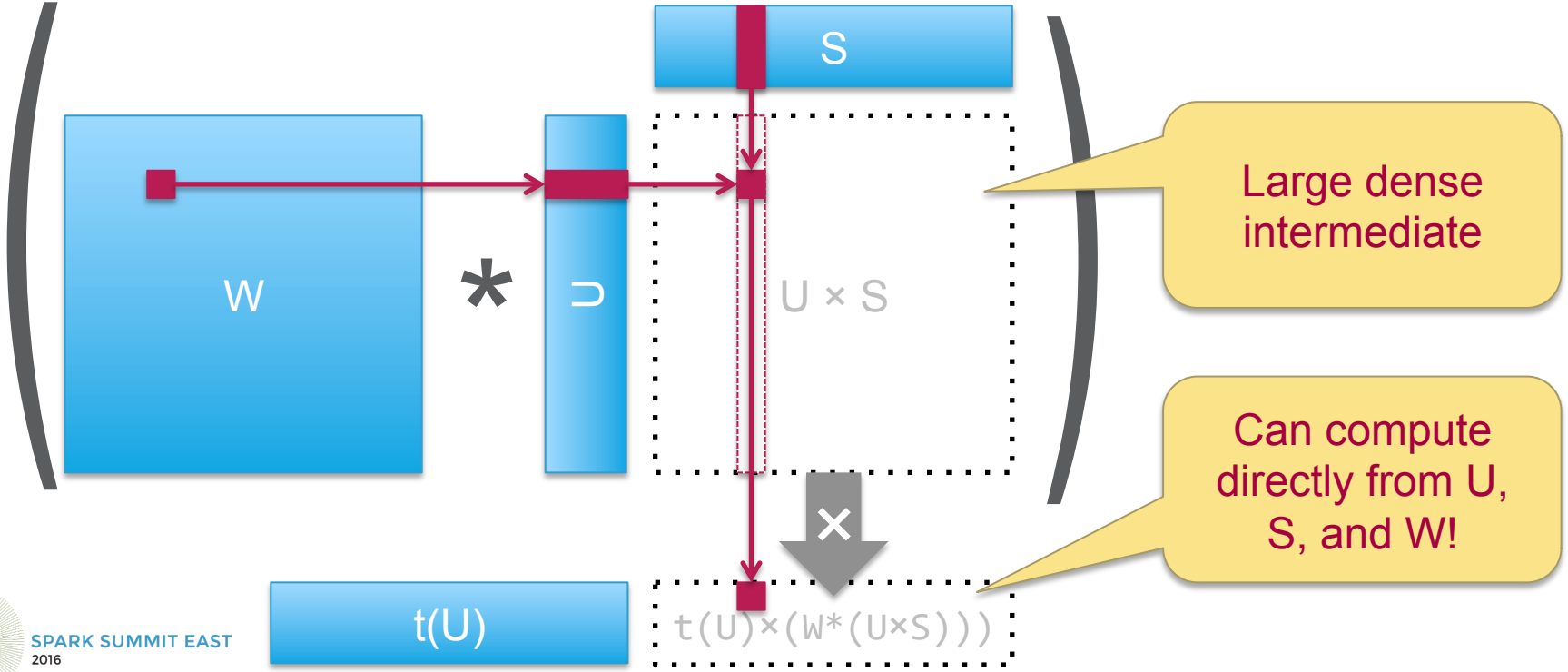

```
t(U) %*% (W * (U %*% S))
```

- Construct HOPs
- Propagate stats
- Determine distributed operations
- **Rewrites**

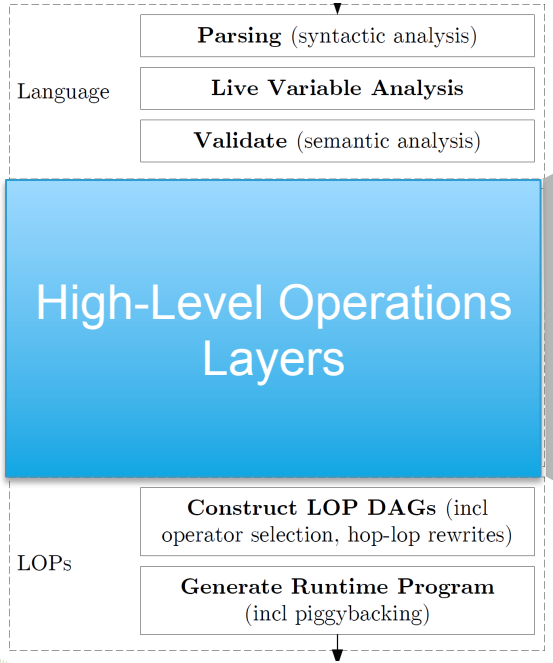


Example Rewrite: wdivmm

```
t(U) %*% (W * (U %*% S))
```

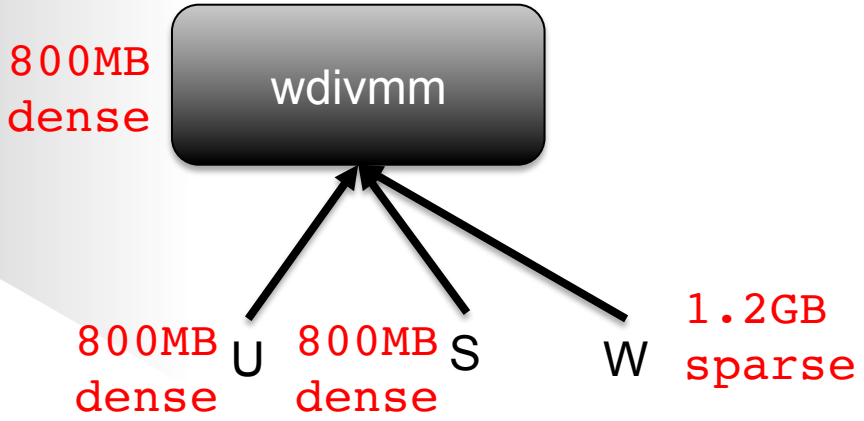


The SystemML Optimizer Stack



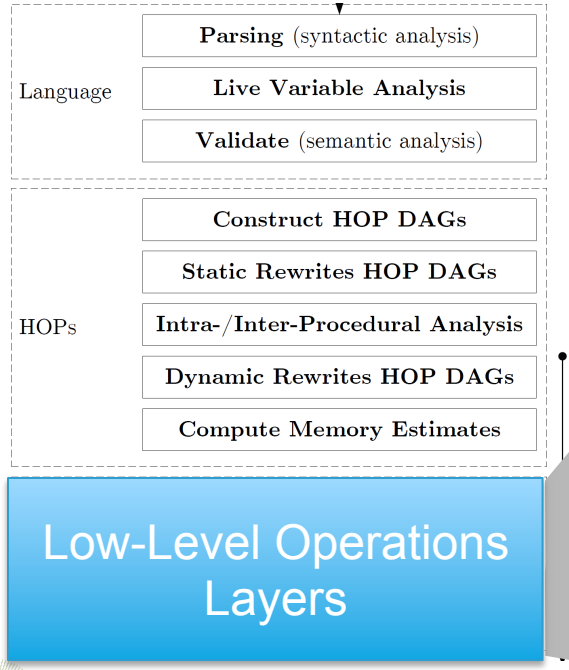

```
t(U) %*% (W * (U %*% S)
```

- Construct HOPs
- Propagate stats
- Determine distributed operations
- **Rewrites**

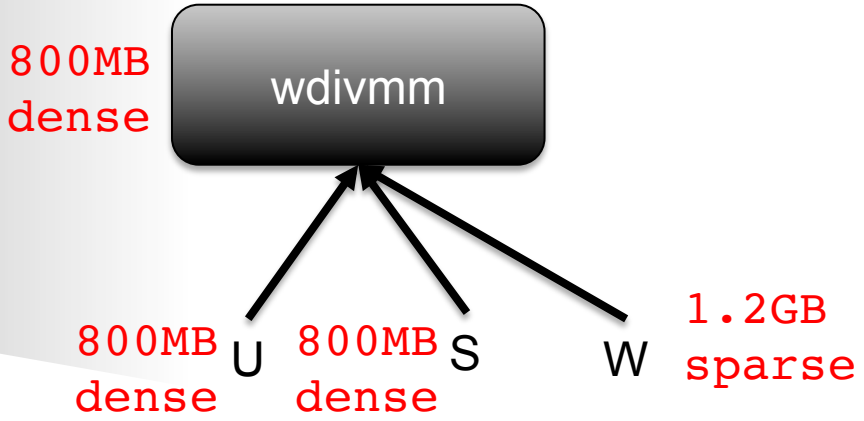


The SystemML Optimizer Stack

```
t(U) %*% (W * (U %*% S)
```



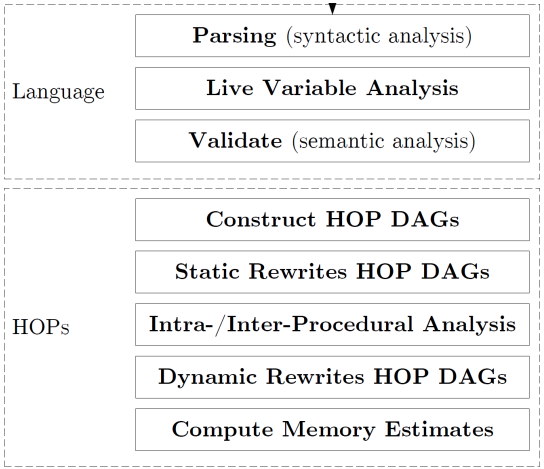
- **Convert HOPs to Low-Level Operations (LOPs)**



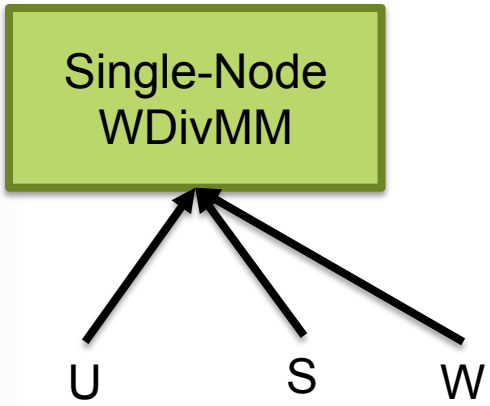
The SystemML Optimizer Stack

```
t(U) %*% (W * (U %*% S)
```

- **Convert HOPs to Low-Level Operations (LOPs)**

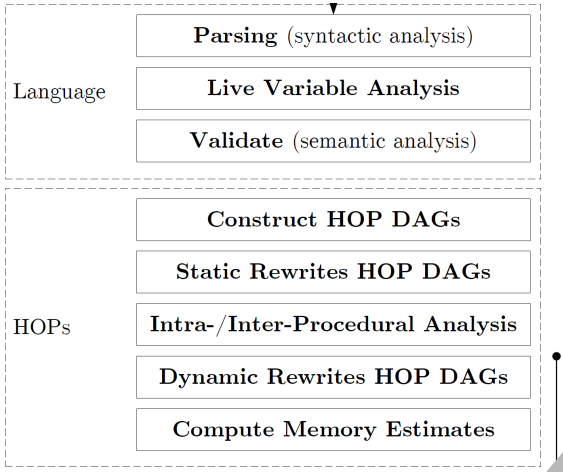


Low-Level Operations Layers

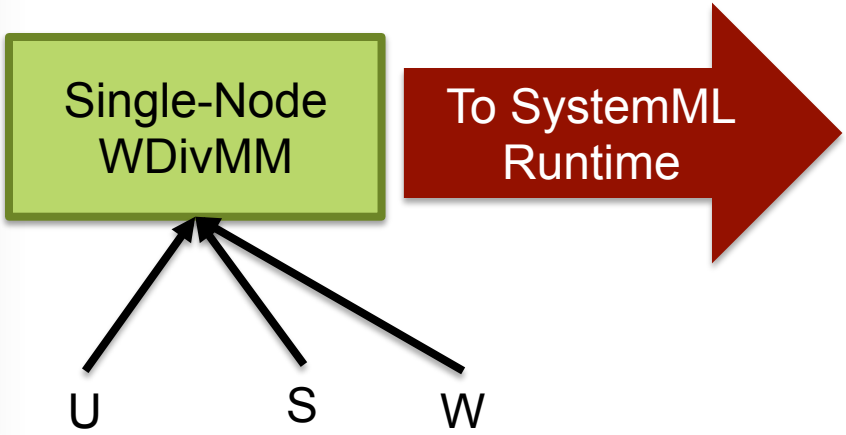


The SystemML Optimizer Stack

```
t(U) %*% (W * (U %*% S)
```



- Convert HOPs to Low-Level Operations (LOPs)
- **Generate runtime instructions**



The SystemML Runtime for Spark

- Automates critical performance decisions
  - *Distributed or local computation?*
  - *How to partition the data?*
  - *To persist or not to persist?*

The SystemML Runtime for Spark

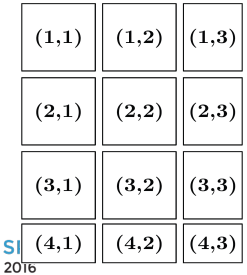
- Distributed vs local: Hybrid runtime
  - Multithreaded computation in Spark Driver
  - Distributed computation in Spark Executors
  - Optimizer makes a cost-based choice

The SystemML Runtime for Spark

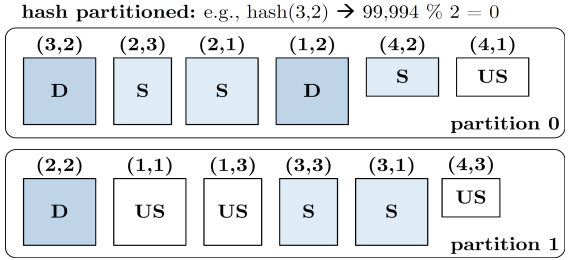
Efficient Linear Algebra

- Binary block matrices (JavaPairRDD<MatrixIndexes, MatrixBlock>)
- Adaptive block storage formats: Dense, Sparse, Ultra-Sparse, Empty
- Efficient kernels for all combinations of block types

Logical Blocking (w/ B_c=1,000)

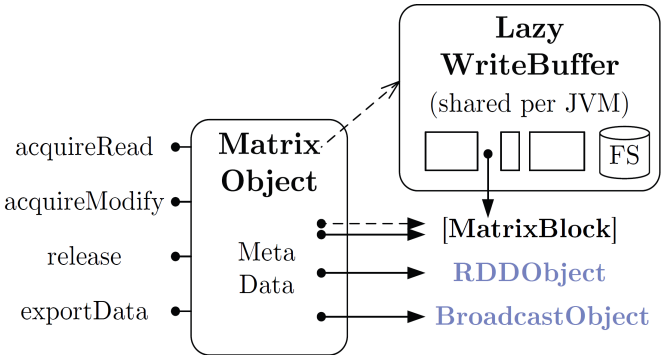


Physical Blocking and Partitioning (w/ B_c=1,000)



Automated RDD Caching

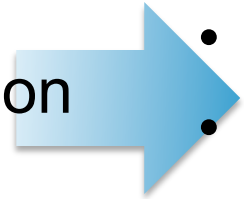
- Lineage tracking for RDDs/ broadcasts
- Guarded RDD collect/parallelize
- Partitioned Broadcast variables



Recap

Questions

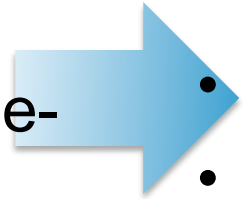
- How does SystemML know it's better to run on one machine?



Answers

- Live variable analysis
- Propagation of statistics

- Why is SystemML so much faster than single-node R?



- Advanced rewrites
- Efficient runtime

But wait, there's more!

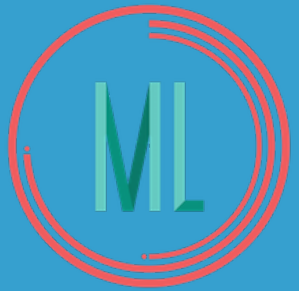
- Many other rewrites
- Cost-based selection of physical operators
- Dynamic recompilation for accurate stats
- Parallel FOR (ParFor) optimizer
- Direct operations on RDD partitions
- YARN and MapReduce support

Open-Sourcing SystemML

- SystemML is open source!
  - Announced in June 2015
  - Available on Github since September 1
  - First open-source binary release (0.8.0) in October 2015
  - Entered Apache incubation in November 2015
  - First Apache open-source binary release (0.9) available now

- We are actively seeking contributors and users!

http://systemml.apache.org/



THANK YOU.

For more information, go to
http://systemml.apache.org/

SPARK SUMMIT EAST
DATA SCIENCE AND ENGINEERING AT SCALE
FEBRUARY 16-18, 2016 NEW YORK CITY