TU Graz

SCIENCE
PASSION
TECHNOLOGY

Architecture of ML Systems
03 Size Inference and Rewrites

Matthias Boehm

Graz University of Technology, Austria
Computer Science and Biomedical Engineering
Institute of Interactive Systems and Data Science
BMVIT endowed chair for Data Management

Last update: Mar 20, 2020

ISDS

Announcements/Org

- **#1 Video Recording**
  - Link in TeachCenter & TUbe (lectures will be public)
  - Streaming: https://tugraz.webex.com/meet/m.boehm

- **#2 Course Administration AMLS**
  - COVID-19 precautions **March 11 – April 19**
  - **Project selection** by **Apr 03** (see Lecture 02)
  - Discussion current status project selection

Agenda

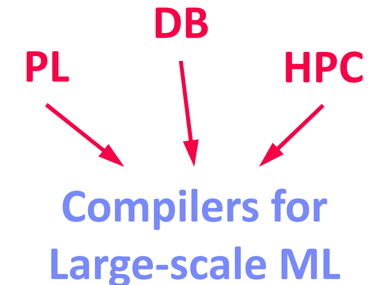
- **Compilation Overview**
- **Size Inference and Cost Estimation**
- **Rewrites (and Operator Selection)**

SystemDS, and several other ML systems

Compilation Overview

Recap: Linear Algebra Systems

- **Comparison Query Optimization**
  - **Rule- and cost-based rewrites and operator ordering**
  - **Physical operator selection and query compilation**
  - Linear algebra / other ML operators, DAGs, control flow, sparse/dense formats



- **#1 Interpretation** (operation at-a-time)
  - Examples: **R**, **PyTorch**, **Morpheus** [PVLDB'17]
- **#2 Lazy Expression Compilation** (DAG at-a-time)
  - Examples: **RIOT** [CIDR'09], **TensorFlow** [OSDI'16] **Mahout Samsara** [MLSystems'16]
  - Examples w/ control structures: **Weld** [CIDR'17], **OptiML** [ICML'11], **Emma** [SIGMOD'15]
- **#3 Program Compilation** (entire program)
  - Examples: **SystemML** [ICDE'11/PVLDB'16], **Julia**, **Cumulon** [SIGMOD'13], **Tupleware** [PVLDB'15]

Optimization Scope

```
1:  X = read($1); # n x m matrix
2:  y = read($2); # n x 1 vector
3:  maxi = 50; lambda = 0.001;
4:  intercept = $3;
5:  ...
6:  r = -(t(X) %*% y);
7:  norm_r2 = sum(r * r); p = -r;
8:  w = matrix(0, ncol(X), 1); i = 0;
9:  while(i<maxi & norm_r2>norm_r2_trgt)
10: {
11:    q = (t(X) %*% X %*% p)+lambda*p;
12:    alpha = norm_r2 / sum(p * q);
13:    w = w + alpha * p;
14:    old_norm_r2 = norm_r2;
15:    r = r + alpha * q;
16:    norm_r2 = sum(r * r);
17:    beta = norm_r2 / old_norm_r2;
18:    p = -r + beta * p; i = i + 1;
19: }
20: write(w, $4, format="text");
```

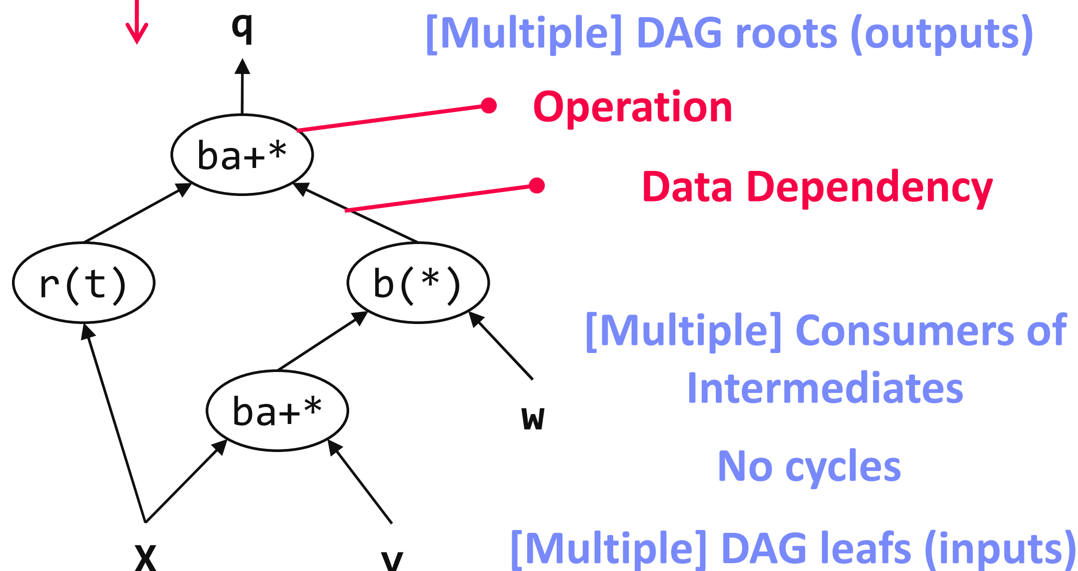
ML Program Compilation / Graphs

- **Script:**

```
while(...) {
    q = t(X) %*% (w * (X %*% v)) ...
}
```

Statement Block Hierarchy

- **Operator DAG** (today's lecture)
  - a.k.a. "graph" (data flow graph)
  - a.k.a. intermediate representation (IR)



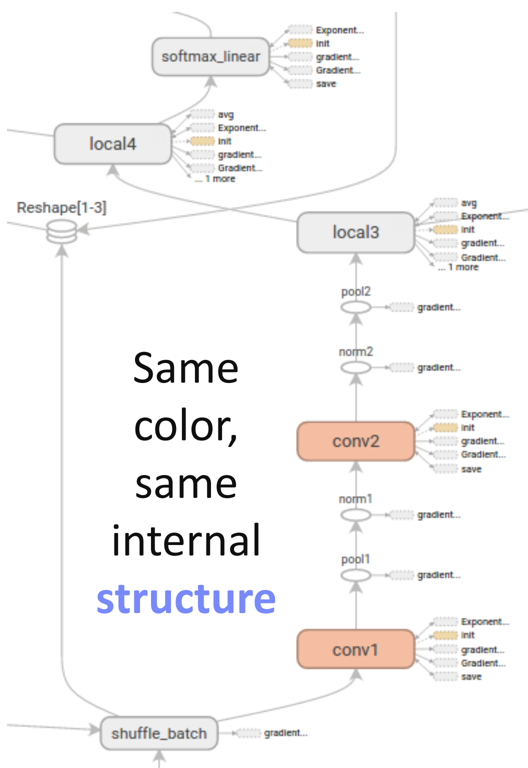
- **Runtime Plan**
  - Compiled runtime plans Interpreted plans

```
SPARK mapmmchain X.MATRIX.DOUBLE w.MATRIX.DOUBLE
  v.MATRIX.DOUBLE _mVar4.MATRIX.DOUBLE XtwXv
```

ML Program Compilation / Graphs, cont.

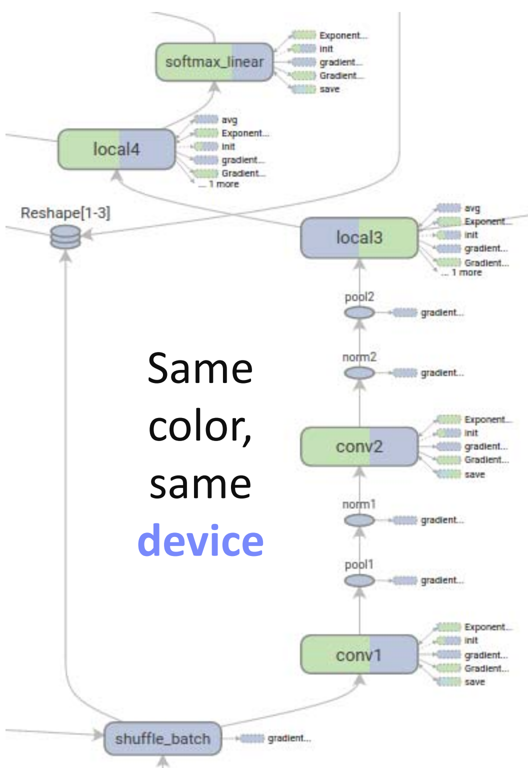
- **Example TF TensorBoard**

(Node) **Structure View**



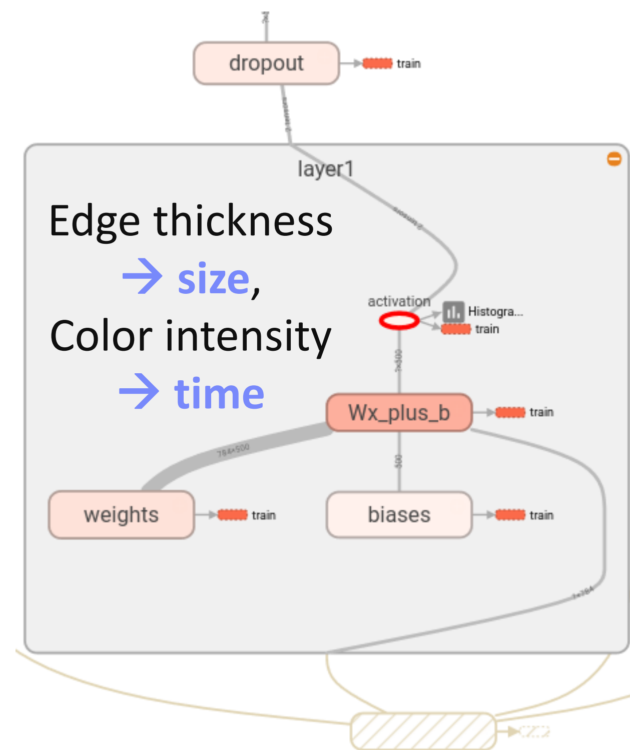
Same color, same internal **structure**

Device View (CPU, GPU)



Same color, same **device**

Tensor Shapes and **Runtime Statistics** (time, mem)



Edge thickness → **size**,
Color intensity → **time**

[https://github.com/tensorflow/tensorboard/blob/master/docs/r1/graphs.md]

Compilation Chain

Script

Language
- **Parsing** (syntactic analysis)
- **Live Variable Analysis**
- **Validate** (semantic analysis)

HOPs
- **Construct HOP DAGs**
- **Multiple Rounds**
  - **Static/Dynamic Rewrites**
  - **Intra-/Inter-Procedural Analysis**
- **Static/Dynamic Rewrites**
- **Compute Memory Estimates**

LOPs
- **Construct LOP DAGs** (incl operator selection, hop-lop rewrites)
- **Generate Runtime Program**

Execution Plan

Dynamic Recompilation (**lecture 04**)

[Matthias Boehm et al: SystemML's Optimizer: Plan Generation for Large-Scale Machine Learning Programs. **IEEE Data Eng. Bull 2014**]

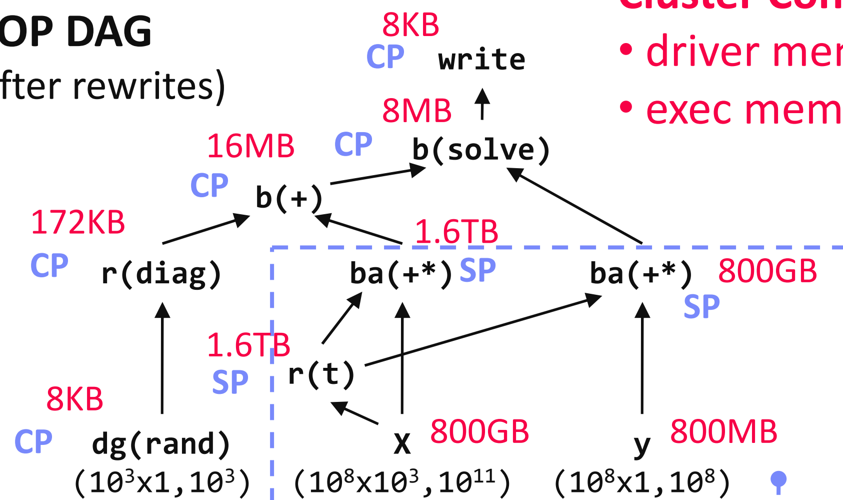
Recap: Basic HOP and LOP DAG Compilation

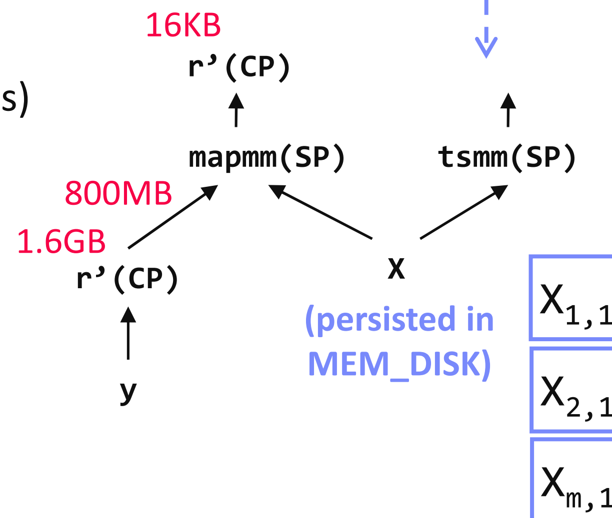
LinregDS (Direct Solve)

```
X = read($1);
y = read($2);
intercept = $3;
lambda = 0.001;
...
if( intercept == 1 ) {
  ones = matrix(1, nrow(X), 1);
  X = append(X, ones);
}
I = matrix(1, ncol(X), 1);
A = t(X) %*% X + diag(I)*lambda;
b = t(X) %*% y;
beta = solve(A, b);
...
write(beta, $4);
```

Scenario:
X: 10^8 x 10^3, 10^{11}
y: 10^8 x 1, 10^8

Cluster Config:
- driver mem: 20 GB
- exec mem: 60 GB





→ **Hybrid Runtime Plans:**
- **Size propagation / memory estimates**
- **Integrated CP / Spark runtime**
- **Dynamic recompilation during runtime**

→ **Distributed Matrices**
- **Fixed-size (squared) matrix blocks**
- **Data-parallel operations**

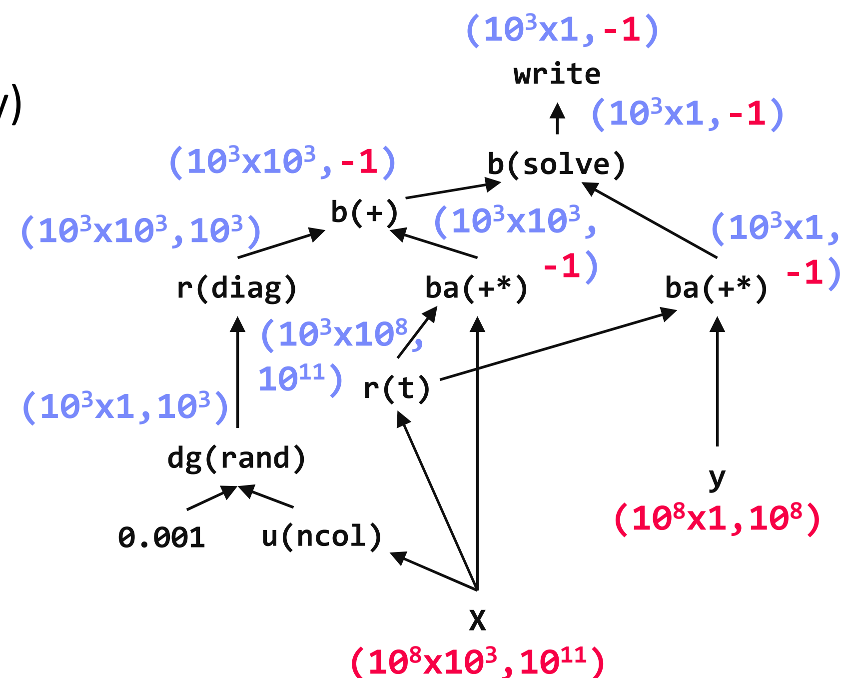
Size Inference and Cost Estimation

Crucial for Generating Valid Execution Plans
& Cost-based Optimization

Constant and Size Propagation

```
X = read($1);
y = read($2);
I = matrix(0.001, ncol(X), 1);
A = t(X) %*% X + diag(I);
b = t(X) %*% y;
beta = solve(A, b);
```

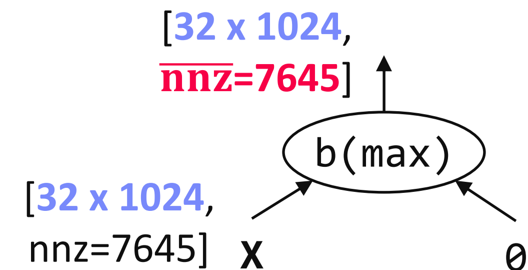
- **Size Information**
  - Dimensions (#rows, #columns)
  - Sparsity (#nnz/(#rows * #columns))
  - ➔ **memory estimates and costs**
- **Principle: Worst-case Assumption**
  - Necessary for guarantees (memory)
- **DAG-level Size Propagation**
  - **Input:** Size information for leaves
  - **Output:** size information for all operators, -1 if still unknown
  - **Propagation based on operation semantics** (single bottom-up pass over DAG)



Constant and Size Propagation, cont.

- **Example SystemDS**
  - Hop `refreshSizeInformation()` (exact)
  - Hop `inferOutputCharacteristics()`
  - Compiler explicitly differentiates between exact and other size information
  - **Note:** ops like aggregate, ctable, rmEmpty

Example Relu
(rectified linear unit)



- **Example TensorFlow**
  - Operator registrations
  - Shape inference functions

```
REGISTER_OP("Relu")
    .Input("features: T")
    .Output("activations: T")
    .Attr("T: {realnumbertype, qint8}")
    .SetShapeFn(
      shape_inference::UnchangedShape)
```

[Alex Passos: Inside TensorFlow – Eager execution runtime, https://www.youtube.com/watch?v=qjx65mD6nrc, Dec 2019]

Constant and Size Propagation, cont.

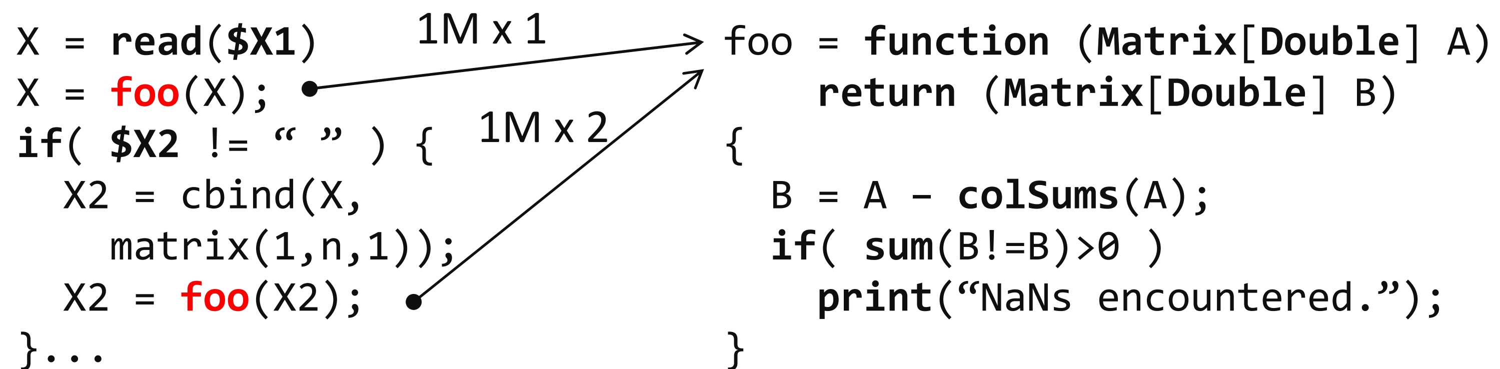
- **Constant Propagation**
  - Relies on live variable analysis
  - Propagate constant literals into read-only statement blocks

- **Program-level Size Propagation**
  - Relies on **constant propagation** and **DAG-level size propagation**
  - **Propagate size information across conditional control flow:** size in leafs, DAG-level prop, extract roots
  - **if:** reconcile if and else branch outputs
  - **while/for:** reconcile pre and post loop, reset if pre/post different

```
X = read($1); # n x m matrix
y = read($2); # n x 1 vector
maxi = 50; lambda = 0.001;
if(...){ }
r = -(t(X) %*% y);
r2 = sum(r * r);
p = -r;                          # m x 1
w = matrix(0, ncol(X), 1);       # m x 1
i = 0;
while(i<maxi & r2>r2_trgt) {
   q = (t(X) %*% X %*% p)+lambda*p;
   alpha = norm_r2 / sum(p * q);
   w = w + alpha * p;            # m x 1
   old_norm_r2 = norm_r2;
   r = r + alpha * q;
   r2 = sum(r * r);
   beta = norm_r2 / old_norm_r2;
   p = -r + beta * p;            # m x 1
   i = i + 1;
}
write(w, $4, format="text");
```

Inter-Procedural Analysis

- **Intra/Inter-Procedural Analysis (IPA)**
  - Integrates all size propagation techniques (**DAG+program**, **size+constants**)
  - Intra-function and inter-function size propagation (**called once**, **consistent sizes**, **consistent literals**)

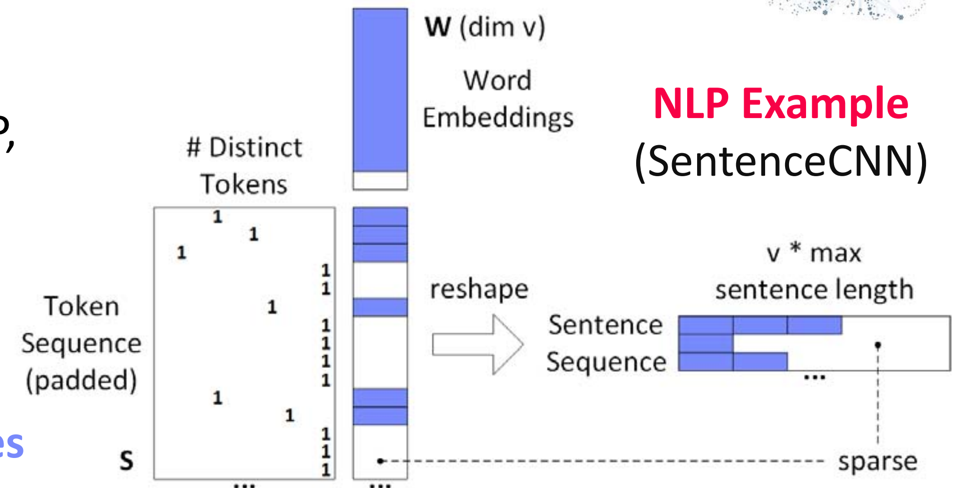


- **Additional IPA Passes (selection)**
  - **Inline functions** (single statement block, small)
  - **Dead code elimination** and simplification rewrites
  - Remove unused functions & flag recompile-once

```
//create order list of IPA passes
_passes = new ArrayList<>();
_passes.add(new IPAPassRemoveUnusedFunctions());
_passes.add(new IPAPassFlagFunctionsRecompileOnce());
_passes.add(new IPAPassRemoveUnnecessaryCheckpoints());
_passes.add(new IPAPassRemoveConstantBinaryOps());
_passes.add(new IPAPassPropagateReplaceLiterals());
_passes.add(new IPAPassInlineFunctions());
_passes.add(new IPAPassEliminateDeadCode());
//note: apply rewrites last because statement block rewrites
//might merge relevant statement blocks in special cases, which
//would require an update of the function call graph
_passes.add(new IPAPassForwardFunctionCalls());
_passes.add(new IPAPassApplyStaticAndDynamicHopRewrites());
```

Sparsity Estimation Overview

- **Motivation**
  - Sparse input matrices from NLP, graph analytics, recommender systems, scientific computing
  - Sparse intermediates (transform, selection, dropout)
  - Selection/permutation matrices

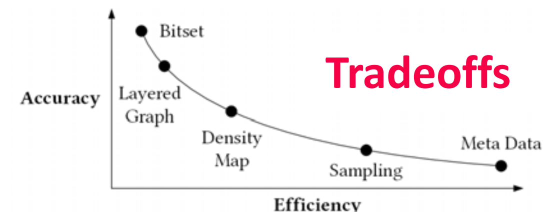


NLP Example (SentenceCNN)

- **Problem Definition**
  - Sparsity estimates used for **format decisions**, **output allocation**, **cost estimates**
  - Matrix A with sparsity s_A = nnz(A)/(mn) and matrix B with s_B = nnz(B)/(nl)
  - Estimate sparsity s_C = nnz(C)/(ml) of matrix product C = A B; d=max(m,n,l)
  - **Assumptions**
    - **A1:** No cancellation errors
    - **A2:** No not-a-number (NaN)

Common assumptions
→ **Boolean matrix product**

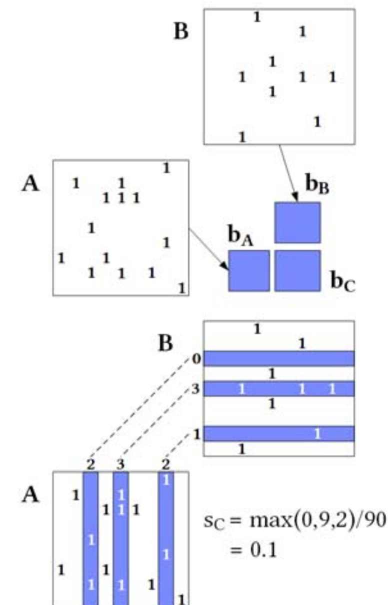
Sparsity Estimation – Estimators



- **#1 Naïve Metadata Estimators**
  - Derive the output sparsity solely from the sparsity of inputs (e.g., SystemML)

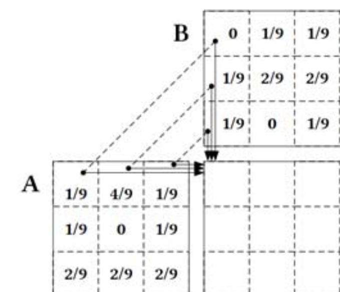
$$\hat{s}_C = 1 - (1 - s_A s_B)^n$$
$$\hat{s}_C = \min(1, s_A n) \cdot \min(1, s_B n)$$

- **#2 Naïve Bitset Estimator**
  - Convert inputs to bitsets and perform Boolean mm
  - Examples: SciDB [SSDBM'11], NVIDIA cuSparse, Intel MKL



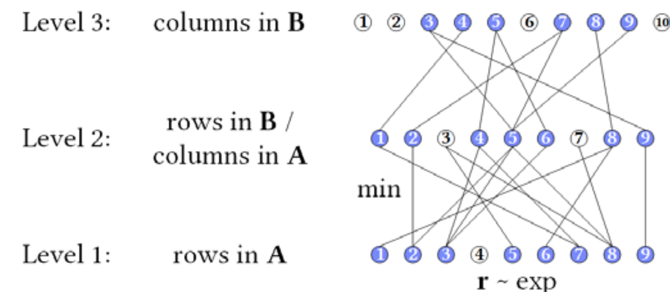
- **#3 Sampling**
  - Take a sample of aligned columns of A and rows of B
  - Sparsity estimated via max of count-products
  - Examples: MatFast [ICDE'17], improvements in paper

- **#4 Density Map**
  - Store sparsity per b x b block (default b = 256)
  - MM-like estimator (average case estimator for *, probabilistic propagation $s_A + s_B - s_A s_B$ for +)
  - Example: SpMacho [EDBT'15], AT Matrix [ICDE'16]



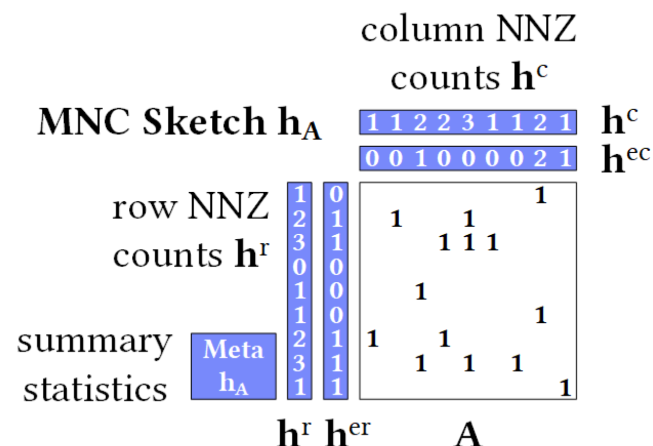
Sparsity Estimation – Estimators, cont.

- **#5 Layered Graph** [J.Comb.Opt.'98]
  - Nodes: rows/columns in mm chain
  - Edges: non-zeros connecting rows/columns
  - Assign r-vectors ~ exp and propagate via min
  - Estimate over roots (output columns)



$$\hat{s}_C = \left(\sum_{v \in \text{roots}} \frac{|\mathbf{r}_v| - 1}{\text{sum}(\mathbf{r}_v)} \right) / (ml),$$

- **#6 MNC Sketch** (Matrix Non-zero Count)
  - Create MNC sketch for inputs A and B
  - Exploitation of structural properties (e.g., 1 non-zero per row, row sparsity)
  - Support for matrix expressions (reorganizations, elementwise ops)
  - Sketch propagation and estimation



$$s_C = \hat{s}_C = h_A^c h_B^r / (ml)$$
if $\max(h_A^r) \leq 1 \vee \max(h_B^c) \leq 1$

[Johanna Sommer, Matthias Boehm, Alexandre V. Evfimievski, Berthold Reinwald, Peter J. Haas: **MNC:** Structure-Exploiting Sparsity Estimation for Matrix Expressions. **SIGMOD 2019**]

Memory Estimates and Costing

- **Memory Estimates**
  - **Matrix memory estimate :=** based on the dimensions and sparsity, decide the format (sparse, dense) and estimate the size in memory
  - **Operation memory estimate :=** input, intermediates, output
  - **Worst-case sparsity estimates** (**upper bound**)
- **#1 Costing at Logical vs Physical Level**
  - Costing at physical level takes physical ops and rewrites into account but is much more costly
- **#2 Costing Operators/Graphs vs Plans**
  - Costing plans requires heuristics for **# iterations**, **branches** in general
- **#3 Analytical vs Trained Cost Models**
  - Analytical: **estimate I/O and compute workload**
  - Training: **build regression models** for individual ops

A Personal War Story

Physical, Plans, **Trained**
[**PVLDB 2014**]

Physical, Plans, Analytical
[**SIGMOD 2015**]

Logical, Graphs, Analytical
[**PVDLB 2018**]

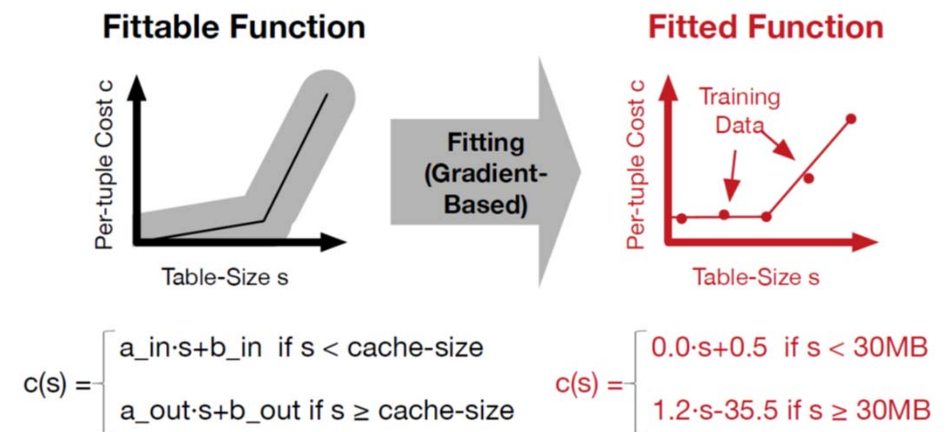
Excursus: Differentiable Programming

- **Overview Differentiable Programming**
  - Adoption of auto differentiation concept from ML systems to PLs
  - Yann LeCun (Jan 2018)

"It's really very much like a regular prog[r]am, except it's parameterized, automatically differentiated, and trainable/optimizable."

- **Example DBMS Fitting**
  - Implement DBMS components as **differentiable functions**
  - E.g.: cost model components
  - Q: **What about guarantees** (memory, size)?

[Benjamin Hilprecht et al: DBMS Fitting: Why should we learn what we already know? **CIDR 2020**]

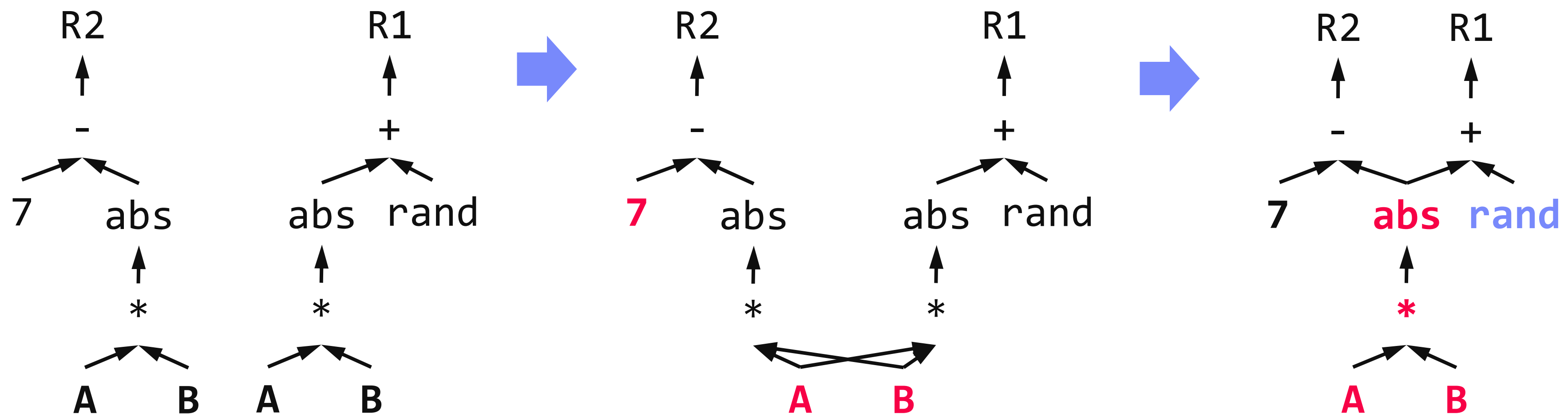


Rewrites and Operator Selection

Traditional PL Rewrites

- **#1 Common Subexpression Elimination (CSE)**
  - **Step 1:** Collect and **replace leaf nodes** (variable reads and literals)
  - **Step 2:** recursively **remove CSEs bottom-up** starting at the leafs by merging nodes with same inputs (**beware non-determinism**)
  - **Example:**

```
R1 = 7 – abs(A * B)
R2 = abs(A * B) + rand()
```



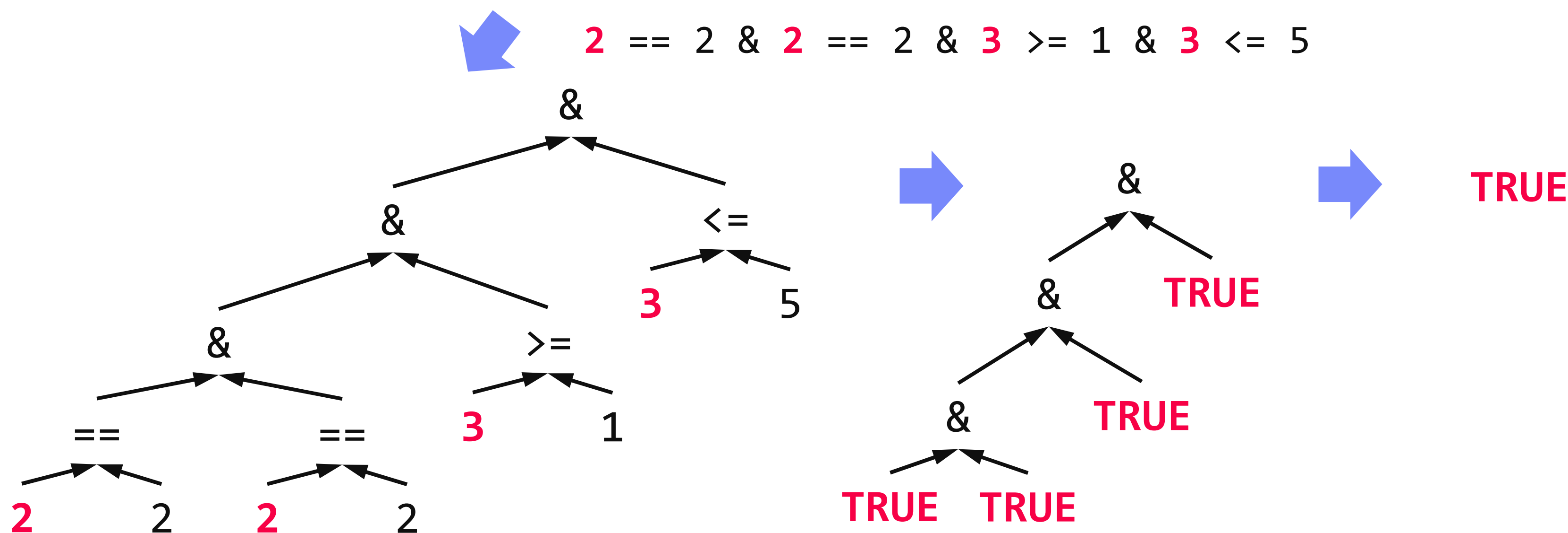
Traditional PL Rewrites, cont.

- **#2 Constant Folding**
  - After constant propagation, fold sub-DAGs over literals into a single literal
  - **Approach:** recursively compile and execute runtime instructions with special handling of one-side constants

[A. V. Aho, M. S. Lam, R. Sethi, and J. D. Ullman. Compilers – Principles, Techniques, & Tools. Addison-Wesley, 2007]

  - **Example** (GLM Binomial probit):

```
ncol_y == 2 & dist_type == 2
  & link_type >= 1 & link_type <= 5
```



Traditional PL Rewrites, cont.

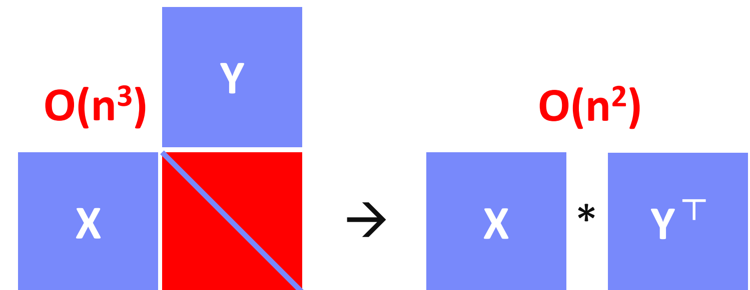
- **#3 Branch Removal**
  - Applied after **constant propagation** and **constant folding**
  - **True predicate:** replace if statement block with if-body blocks
  - **False predicate:** replace if statement block with else-body block, or remove
- **#4 Merge of Statement Blocks**
  - **Merge sequences of unconditional blocks** (s1,s2) into a single block
  - Connect matching DAG roots of s1 with DAG inputs of s2

LinregDS (Direct Solve)

```
X = read($1);
y = read($2);
intercept = 0;
lambda = 0.001;
...          FALSE
if( intercept == 1 ) {
  ones = matrix(1, nrow(X), 1);
  X = cbind(X, ones);
}
I = matrix(1, ncol(X), 1);
A = t(X) %*% X + diag(I)*lambda;
b = t(X) %*% y;
beta = solve(A, b);
...
write(beta, $4);
```

Static/Dynamic Simplification Rewrites

- **Examples of Static Rewrites**
  - `trace(X%*%Y) → sum(X*t(Y))`
  - `sum(X+Y) → sum(X)+sum(Y)`
  - `(X%*%Y)[7,3] → X[7,]%*%Y[,3]`
  - `sum(t(X)) → sum(X)`
  - `rand()*7 → rand(,min=0,max=7)`
  - `sum(lambda*X) → lambda * sum(X);`



[Matthias Boehm et al: SystemML's Optimizer: Plan Generation for Large-Scale Machine Learning Programs. **IEEE Data Eng. Bull 2014**]

- **Examples of Dynamic Rewrites**
  - `t(X) %*% y → t(t(y) %*% X) s.t. costs`
  - `X[a:b,c:d]=Y → X = Y iff dims(X)=dims(Y)`
  - `(...) * X → matrix(0, nrow(X), ncol(X)) iff nnz(X)=0`
  - `sum(X^2) → t(X)%*%X; rowSums(X) → X iff ncol(X)=1`
  - `sum(X%*%Y) → sum(t(colSums(X))*rowSums(Y)) iff ncol(X)>t`

Static/Dynamic Simplification Rewrites, cont

- **TF Constant Push-Down**
  - `Add(c1,Add(x,c2)) → Add(x,c1+c2)`
  - `ConvND(c1*x,c2) → ConvND(x,c1*c2)`

[Rasmus Munk Larsen, Tatiana Shpeisman: TensorFlow Graph Optimizations, **Guest Lecture Stanford 2019**]

Recommended Reading

- **TF Arithmetic Simplifications**
  - Flattening: a+b+c+d → `AddN(a, b, c, d)`
  - Hoisting: `AddN(x * a, b * x, x * c) → x * AddN(a+b+c)`
  - Reduce Nodes Numeric: x+x+x → `3*x`
  - Reduce Nodes Logicial: `!(x > y) → x <= y`

SystemML/SystemDS
`RewriteElementwise-MultChainOptimization`
(orders and collapses matrix, vector, scalar op chains)

- **TF Broadcast minimization**
  - `(M1+s1) + (M2+s2) → (M1+M2) + (s1+s2)`
- **TF Better use of intrinsics**
  - `Matmul(Transpose(X), Y) → Matmul(X, Y, transpose_x=True)`

Vectorization and Incremental Computation

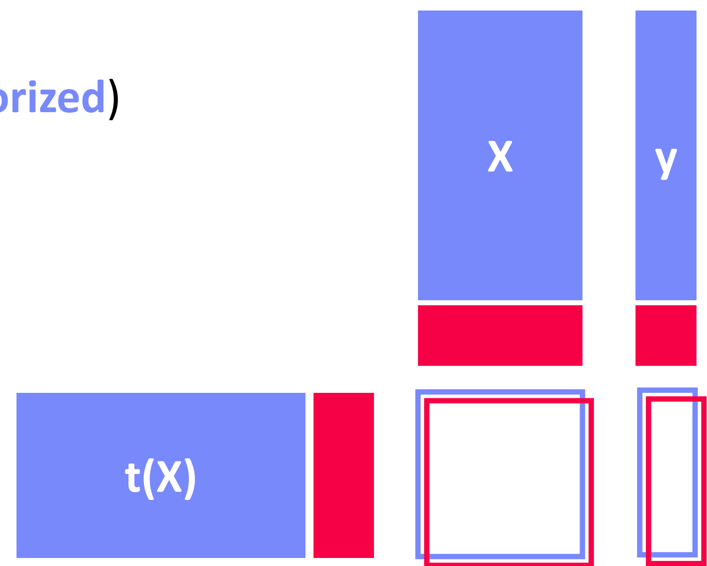
- **Loop Transformations** (e.g., **OptiML**, **SystemML**)
  - **Loop vectorization**
  - Loop hoisting

```
for(i in a:b)
   X[i,1] = Y[i,2] + Z[i,1]

→ X[a:b,1] = Y[a:b,2] + Z[a:b,1]
```

- **Incremental Computations**
  - **Delta update rules** (e.g., **LINVIEW**, **factorized**)
  - Incremental iterations (e.g., **Flink**)

```
A = t(X) %*% X + t(ΔX) %*% ΔX
b = t(X) %*% y + t(ΔX) %*% Δy
```



- **"Decremental"** (e.g., GDPR)

[Sebastian Schelter: "Amnesia" – Machine Learning Models That Can Forget User Data Very Fast. **CIDR 2020**]

Update-in-place

- **Example: Cumulative Aggregate via Strawman Scripts**
  - **But:** R, Julia, Matlab, SystemML, NumPy all provide cumsum(X), etc

```
1: cumsumN2 = function(Matrix[Double] A)
2:   return(Matrix[Double] B)
3: {
4:   B = A; csums = matrix(0,1,ncol(A));
5:   for( i in 1:nrow(A) ) {
6:     csums = csums + A[i,];
7:     B[i,] = csums;
8:   }
9: }
```

copy-on-write → O(n^2)

```
1: cumsumNlogN = function(Matrix[Double] A)
2:   return(Matrix[Double] B)
3: {
4:   B = A; m = nrow(A); k = 1;
5:   while( k < m ) {
6:     B[(k+1):m,] = B[(k+1):m,] + B[1:(m-k),];
7:     k = 2 * k;
8:   }
9: }
```

→ O(n log n)

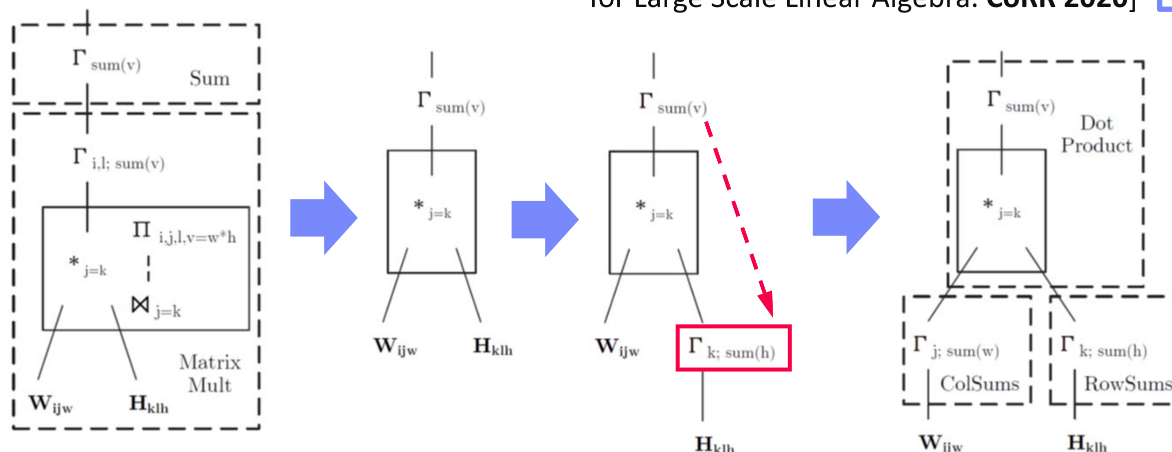
- **Update in place** (w/ O(n))
  - **SystemML:** via rewrites (**why do the above scripts apply?**)
  - **R:** via reference counting
  - **Julia:** by default, otherwise explicit **B = copy(A)** necessary

Excursus: Automatic Rewrite Identification

- **SPOOF (Sum-Product Optimization)**
  - **Break up** LA ops into basic ops (RA)
  - **Elementary sum-product/RA rewrites**
  - **Example:** `sum(W%*%H)`

[Tarek Elgamal et al: SPOOF: Sum-Product Optimization and Operator Fusion for Large-Scale Machine Learning. **CIDR 2017**]

[Yisu Remy Wang et al: SPORES: Sum-Product Optimization via Relational Equality Saturation for Large Scale Linear Algebra. **CoRR 2020**]

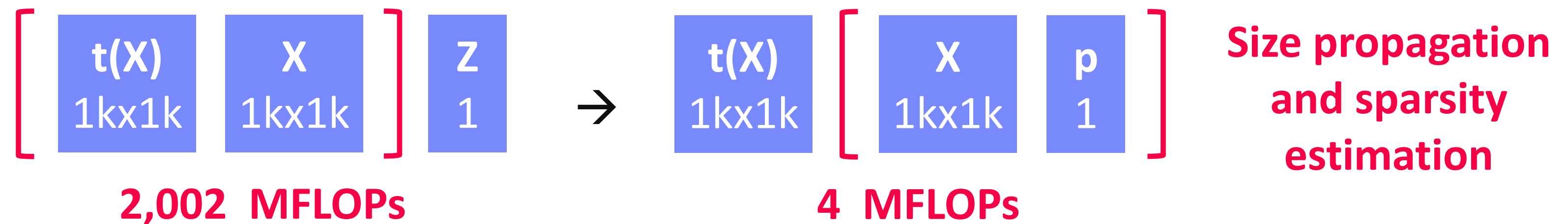


- **TASO (Super Optimization)**
  - List of operator specifications and properties
  - Automatic **generation/verification of graph substitutions**, optimized via backtracking search

[Zhihao Jia et al: TASO: optimizing deep learning computation with automatic generation of graph substitutions. **SOSP 2019**]

Matrix Multiplication Chain Optimization

- **Optimization Problem**
  - Matrix multiplication chain of n matrices $M_1, M_2, \ldots M_n$ (associative)
  - Optimal parenthesization of the product $M_1 M_2 \ldots M_n$



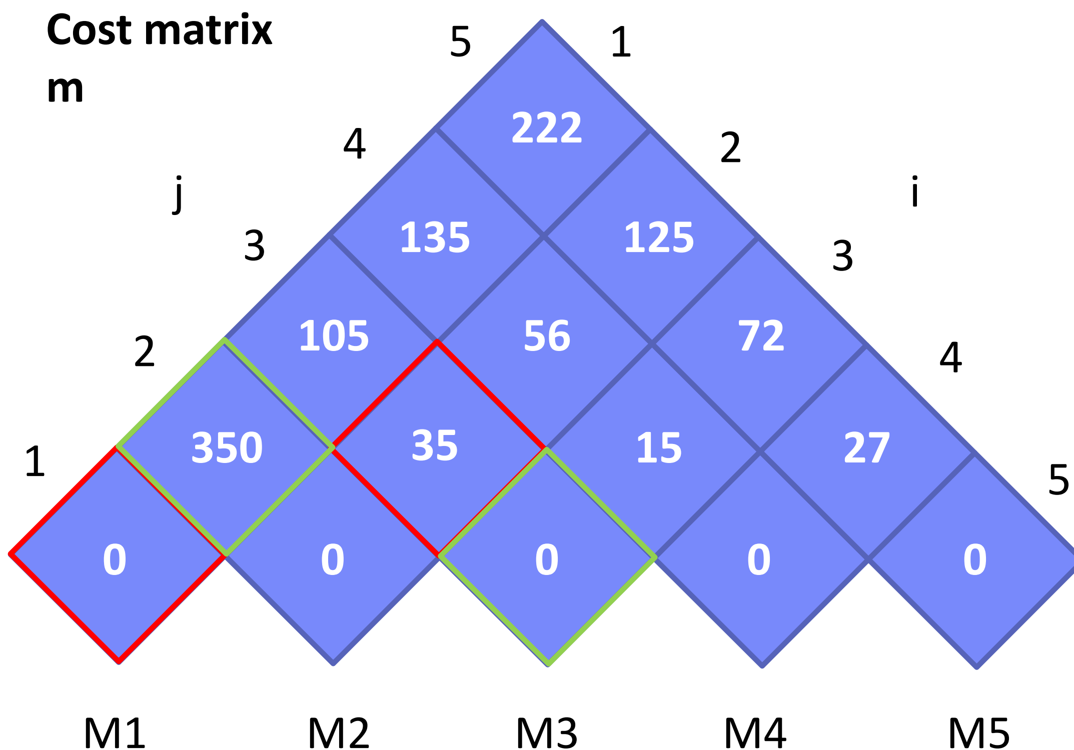
- **Search Space Characteristics**
  - Naïve exhaustive: Catalan numbers → $\Omega(4^n / n^{3/2})$)
  - DP applies: (1) optimal substructure, (2) overlapping subproblems
  - Textbook DP algorithm: $\Theta(n^3)$ time, $\Theta(n^2)$ space
    - Examples: SystemML '14, RIOT ('09 I/O costs), SpMachO ('15 sparsity)
  - Best known: $O(n \log n)$

n	C_{n-1}
5	14
10	4,862
15	2,674,440
20	1,767,263,190
25	1,289,904,147,324

[T. C. Hu, M. T. Shing: Computation of Matrix Chain Products. Part II. **SIAM J. Comput.** 13(2): 228-251, 1984]

Matrix Multiplication Chain Optimization, cont.

M1	M2	M3	M4	M5
10x7	7x5	5x1	1x3	3x9



m[1,3] = min(
m[1,1] + m[2,3] + p1p2p4,
m[1,2] + m[3,3] + p1p3p4)

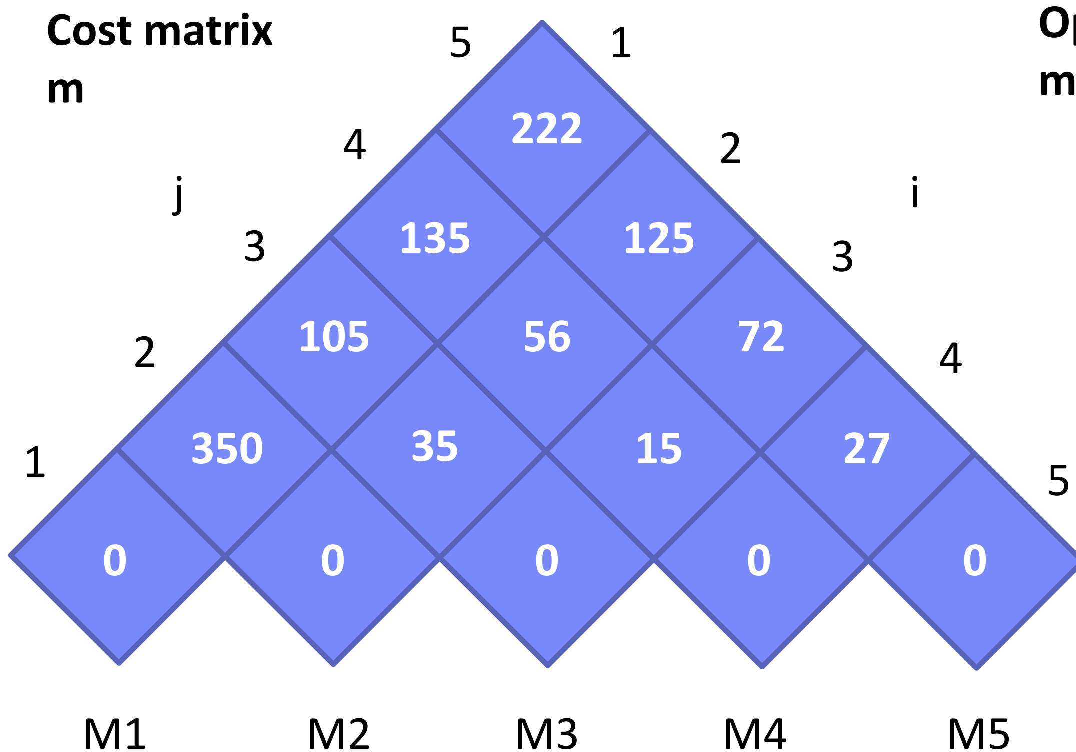
= min(
0 + 35 + 10*7*1,
350 + 0 + 10*5*1)

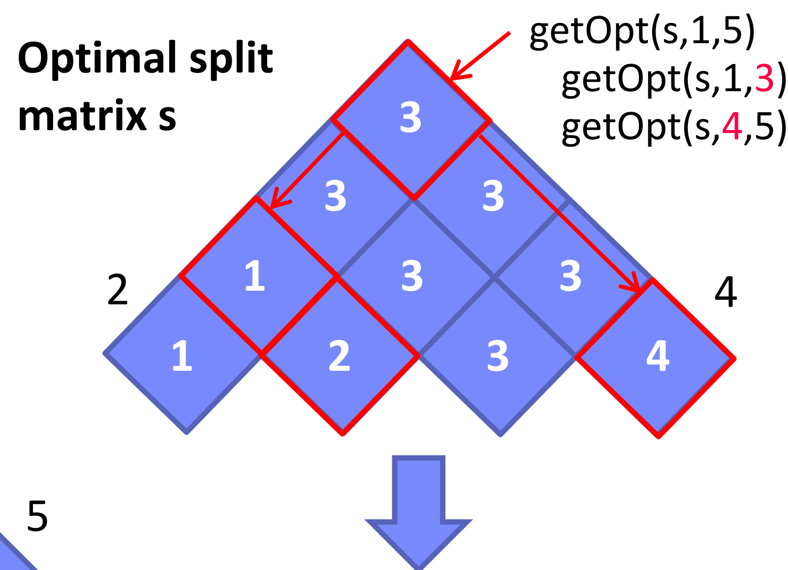
= min(
105,
400)

[T. H. Cormen, C. E. Leiserson, R. L. Rivest, C. Stein: Introduction to Algorithms, Third Edition, **The MIT Press**, pages 370-377, 2009]

Matrix Multiplication Chain Optimization, cont.

M1	M2	M3	M4	M5
10x7	7x5	5x1	1x3	3x9





(M1 M2 M3 M4 M5)

((M1 M2 M3) (M4 M5))

((M1 (M2 M3)) (M4 M5))

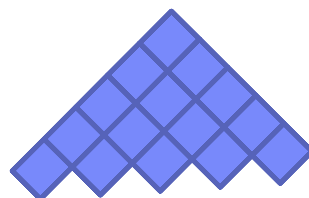
➔ ((M1 (M2 M3)) (M4 M5))

➔ Open questions: DAGs; other operations, sparsity joint opt w/ rewrites, CSE, fusion, and physical operators

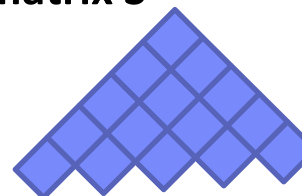
Matrix Multiplication Chain Optimization, cont.

- **Sparsity-aware mmchain Opt**
  - Additional n x n sketch matrix e

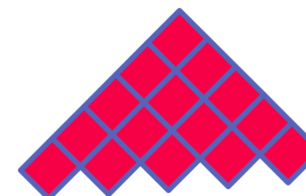
Cost matrix M



Optimal split matrix S



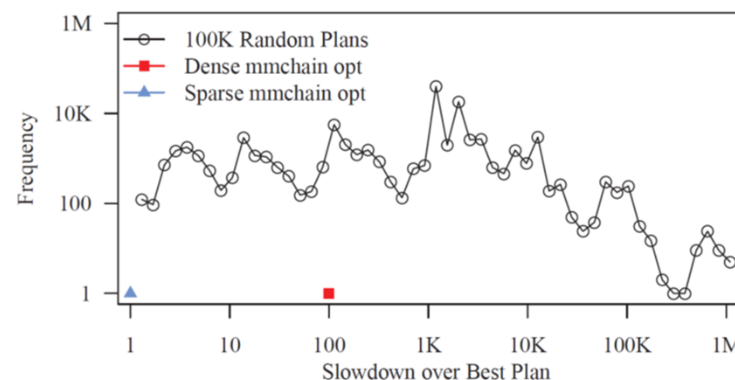
Sketch matrix E



  - Sketch propagation for optimal subchains (currently for all chains)
  - Modified cost computation via MNC sketches (**number FLOPs for sparse** instead of dense mm)

$$C_{i,j} = \min_{k \in [i,j-1]} \begin{array}{c}(C_{i,k} + C_{k+1,j} \\ + \boldsymbol{E_{i,k}.\, h^c E_{k+1,j}.\, h^r})\end{array}$$

Example: n=20 matrices



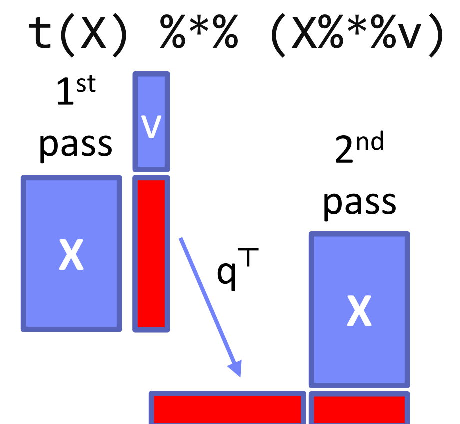
[Johanna Sommer, Matthias Boehm, Alexandre V. Evfimievski, Berthold Reinwald, Peter J. Haas: **MNC:** Structure-Exploiting Sparsity Estimation for Matrix Expressions. **SIGMOD 2019**]

Physical Rewrites and Optimizations

- **Distributed Caching**
  - Redundant compute vs. memory consumption and I/O
  - **#1 Cache intermediates w/ multiple refs** (Emma)
  - **#2 Cache initial read and read-only loop vars** (SystemML)
- **Partitioning**
  - Many frameworks exploit co-partitioning for efficient joins
  - **#1 Partitioning-exploiting operators** (SystemML, Emma, Samsara)
  - **#2 Inject partitioning to avoid shuffle per iteration** (SystemML)
  - **#3 Plan-specific data partitioning** (SystemML, Dmac, Kasen)
- **Other Data Flow Optimizations** (Emma)
  - **#1 Exists unnesting** (e.g., filter w/ broadcast → join)
  - **#2 Fold-group fusion** (e.g., groupByKey → reduceByKey)
- **Physical Operator Selection**

Physical Operator Selection

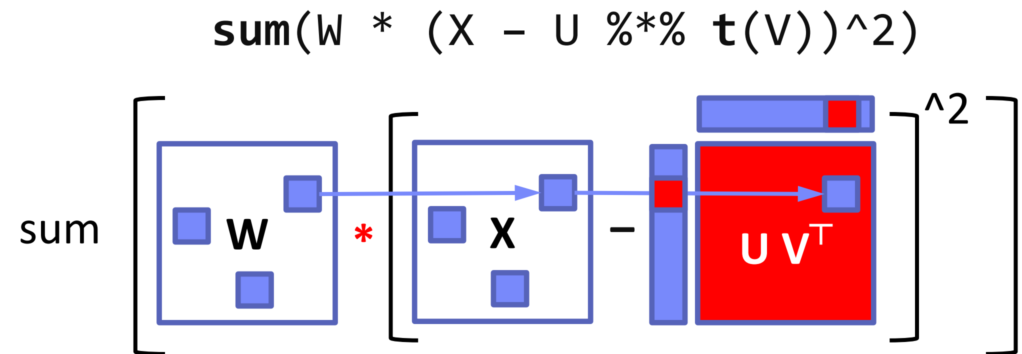
- **Common Selection Criteria**
  - **Data and cluster characteristics** (e.g., data size/shape, memory, parallelism)
  - **Matrix/operation properties** (e.g., diagonal/symmetric, sparse-safe ops)
  - **Data flow properties** (e.g., co-partitioning, co-location, data locality)
- **#0 Local Operators**
  - SystemML mm, tsmm, mmchain; Samsara/Mllib local
- **#1 Special Operators** (special patterns/sparsity)
  - SystemML **tsmm**, **mapmmchain**; Samsara AtA



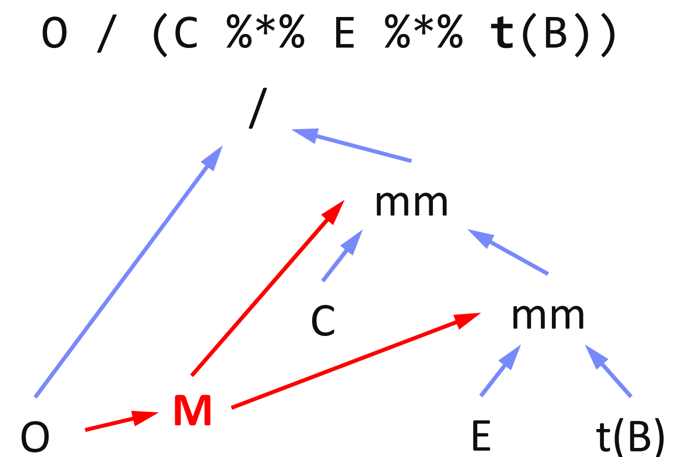
- **#2 Broadcast-Based Operators** (aka broadcast join)
  - SystemML **mapmm**, **mapmmchain**
- **#3 Co-Partitioning-Based Operators** (aka improved repartition join)
  - SystemML **zipmm**; Emma, Samsara OpAtB
- **#4 Shuffle-Based Operators** (aka repartition join)
  - SystemML **cpmm**, **rmm**; Samsara OpAB

Sparsity-Exploiting Operators

- **Goal:** Avoid dense intermediates and unnecessary computation
- **#1 Fused Physical Operators**
  - E.g., SystemML [PVLDB'16] wsloss, wcemm, wdivmm
  - Selective computation over non-zeros of "sparse driver"



- **#2 Masked Physical Operators**
  - E.g., Cumulon MaskMult [SIGMOD'13]
  - Create mask of "sparse driver"
  - Pass mask to single masked matrix multiply operator



Conclusions

- **Summary**
  - Basic compilation overview
  - **Size inference and cost estimation**
  - **Rewrites and operator selection**

Plenty of Open Programming Projects

➔ **Impact of Size Inference and Costs**
  - Advanced optimization of LA programs requires size inference for cost estimation and validity constraints

➔ **Ubiquitous Rewrite Opportunities**
  - Linear algebra programs have plenty of room for optimization
  - Potential for changed asymptotic behavior

- **Next Lectures**
  - **04 Operator Fusion and Runtime Adaptation** [Mar 27] (advanced compilation, operator scheduling, JIT compilation, operator fusion / codegen, MLIR)