Architecture of ML Systems (AMLS)
04 Compilation – Adaptation, Fusion, and JIT

Prof. Dr. Matthias Boehm
Technische Universität Berlin
Berlin Institute for the Foundations of Learning and Data
Big Data Engineering (DAMS Lab)

Last update: May 11, 2023

Announcements / Org

- **#1 Hybrid & Video Recording**
  - Hybrid lectures (in-person, zoom) with optional attendance
    https://tu-berlin.zoom.us/j/9529634787?pwd=R1ZsN1M3SC9BOU1OcFdmem9zT202UT09
  - Zoom **video recordings**, links from website
    https://mboehm7.github.io/teaching/ss23_amls/index.htm

- **#2 Reminder Project / Exercise Selection**
  - **Task description** and updated projects on course website
  - **Project Selection by May 10**, Submission by **July 04**
  - Submission for TU Berlin / TU Graz / external students

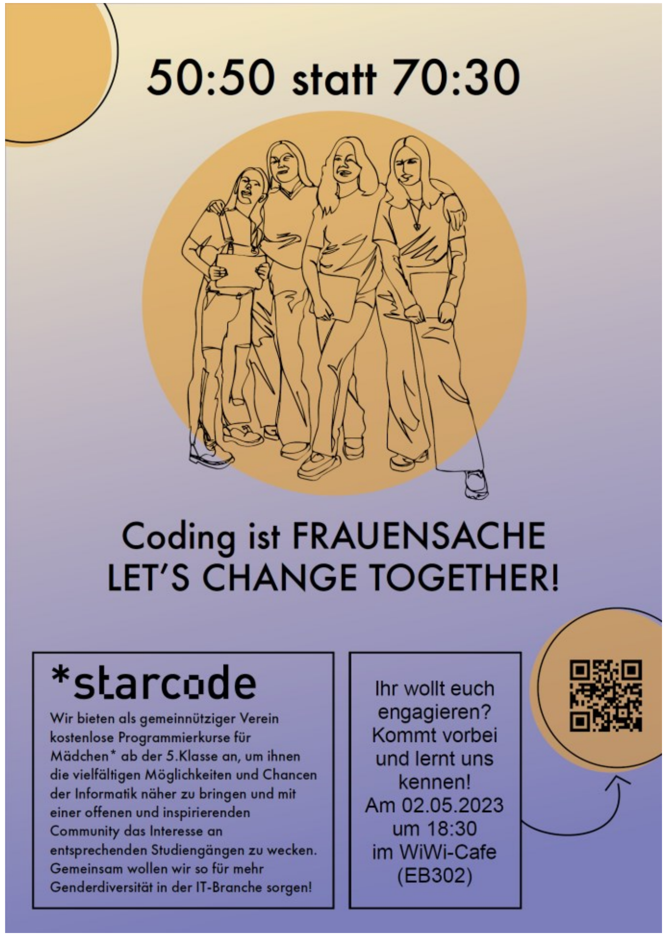
~263 course regs

~80 project regs

- **#3 Feedback**
  - Both positive and negative, both public (Q&A forum) or private (email/DM)
  - Missing background, too fast-paced

Announcements / Org, cont.

- **#4 `*starcode` – Help Wanted**
  - Student initiative, **offering free programming workshops** for female students (school-level, but beyond Scratch)
  - Looking for **students to get involved as mentors**
  - https://www.starcode.de/



- **#5 Three SHK Positions @ DAMS**
  - Calls for applications out soon
  - **2x 60h/month** for up to 5 years
  - **1x 40h/month** for up to 5 years

Agenda

- **Motivation and Terminology**
- **Runtime Adaptation**
- **Operator Fusion & JIT Compilation**

Motivation and Terminology

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

PL

DB

HPC

Compilers for Large-scale ML

- **#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** [PVLDB'16], **Julia** **Cumulon** [SIGMOD'13], **Tupleware** [PVLDB'15]

Optimization Scope

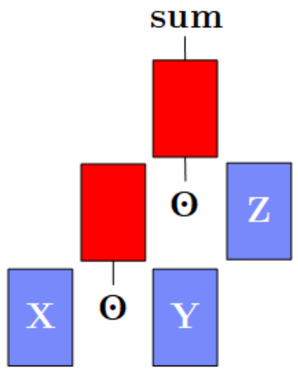
```
X = read($1); # n x m matrix
y = read($2); # n x 1 vector
maxi = 50; lambda = 0.001;
intercept = $3;
...
r = -(t(X) %*% y);
norm_r2 = sum(r * r); p = -r;
w = matrix(0, ncol(X), 1); i = 0;
while(i<maxi & norm_r2>norm_r2_trgt)
{
   q = (t(X) %*% X %*% p)+lambda*p;
   alpha = norm_r2 / sum(p * q);
   w = w + alpha * p;
   old_norm_r2 = norm_r2;
   r = r + alpha * q;
   norm_r2 = sum(r * r);
   beta = norm_r2 / old_norm_r2;
   p = -r + beta * p; i = i + 1;
}
write(w, $4, format="text");
```

Major Compilation/Runtime Challenges (aka Why You Should Care)

- **#1 Unknown/Changing Sizes**
  - Sizes inference crucial for cost-estimation and validity constraints (e.g., rewrites)
  - **Tradeoff:** optimization scope vs size inference effort
  - **Challenge:** Unknowns ➔ conservative fallback plans

```
Y = foo(X)
Z = Y[Ix,]
# nrow(Z)?
```

- **#2 Operator Runtime Overhead**
  - Operators great for **programmability**, size inference, simple compilation, and **efficient kernel implementations** (sparse, dense, compressed)
  - **Tradeoff:** general-purpose vs specialization
  - **Challenges:** intermediates, parallelization, complexity of operator combinations



➔ **Resource-efficient Training and Inference** (runtime, energy, costs)

Terminology Ahead-of-Time / Just-in-Time

- **Ahead-of-Time Compilation**
  - Originating from compiled languages like C, C++
  - **#1** **Program compilation** at different abstraction levels
  - **#2** **Inference program compilation** & packaging

Apache SystemML™
TensorFlow Lite
PYTORCH

- **Just-In-Time Compilation** (at runtime for specific data/HW)
  - Originating from JIT-compiled languages like Java, C#
  - **#1** **Lazy expression evaluation** + optimization
  - **#2** Program/function compilation **with recompilation**

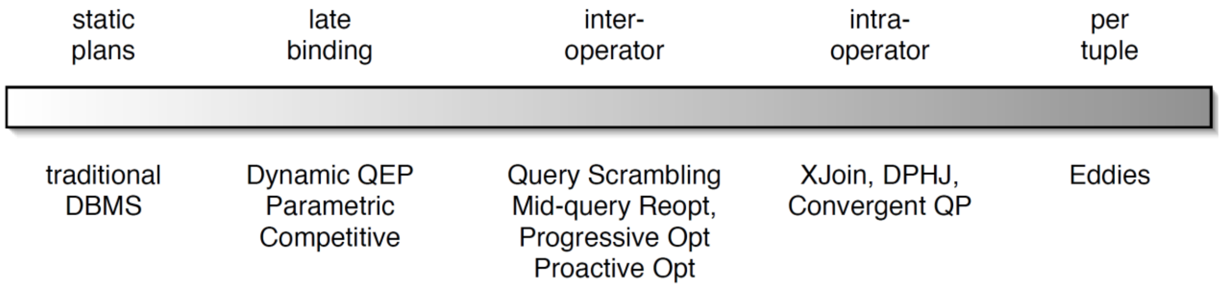
MAHOUT
TensorFlow
Apache SystemML™
julia
(LLVM)

- **Excursus: Java JIT**
  - **#1** Start w/ Java bytecode interpretation by JVM ➔ **fast startup**
  - **#2** **Tiered JIT compile** (cold, warm, hot, very hot, scorching) ➔ **performance**
  - Trace statistics (frequency, time) at method granularity
  - Note: `-XX:+PrintCompilation`

PL

Terminology Runtime Adaptation & JIT

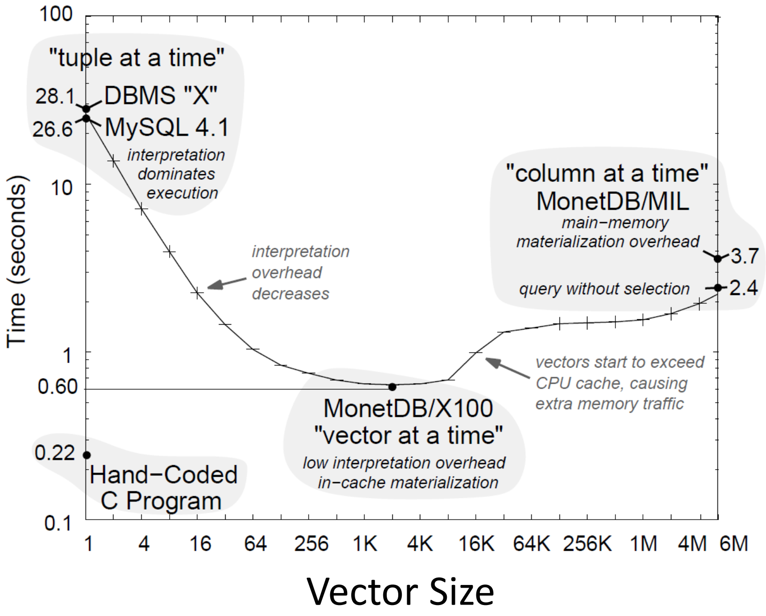
- **Excursus: Adaptive Query Processing**
  - DB **Spectrum of Adaptivity**



[Amol Deshpande, Joseph M. Hellerstein, Shankar Raman: Adaptive query proc-essing: why, how, when, what next. **SIGMOD 2006**]

- **Excursus: Query Execution Strategies**
  - DB
  - **#1 Volcano Iterator Model**
  - **#2 Materialized Intermediates**
  - **#3 Vectorized (Batched) Execution**
  - **#4 Query Compilation**
  - HPC Similar: Loop fusion, fission, tiling

[Peter A. Boncz, Marcin Zukowski, Niels Nes: MonetDB/X100: Hyper-Pipelining Query Execution. **CIDR 2005**]



Runtime Adaptation

ML Systems w/ Optimizing Compiler

Apache SystemML™

Issues of Unknown or Changing Sizes

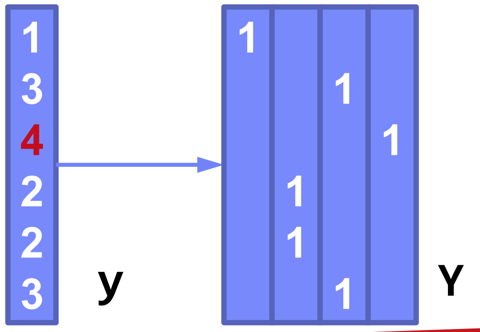
- **Problem of unknown/changing sizes**
  - Unknown or changing sizes and sparsity of intermediates
    These unknowns lead to very conservative fallback plans (distributed ops)

- **#1 Control Flow**
  - Branches and loops
  - Complex function call graphs
  - User-Defined Functions

```
X = read('/tmp/X.csv');
if( intercept )
  X = cbind(X, matrix(1,nrow(X),1));

Z = foo(X) + X; # size of + and Z?
```

- **#2 Data-Dependencies**
  - Data-dependent operators
    (e.g., table, rmEmpty, aggregate)
  - Computed size expressions

```
Y = table(seq(1,nrow(X)), y);
grad = t(X) %*% (P - Y);

d = dout[,(t-2)*M+1:(t-1)*M];

cur_Q = matrix (0, 1, 2*ncur);
cur_S = matrix (0, 1, ncur*dist);
```



Issues of Unknown or Changing Sizes, cont.

- **#3 Changing Dims and Sparsity**
  - Iterative feature selection workloads
  - Changing dimensions or sparsity
  - ➔ Same code with different data

```
# Example Stepwise Linear Regression
while( continue ) {
   parfor( i in 1:n ) {
      if( !fixed[1,i] ) {
         Xi = cbind(Xg, X[,i])
         B[,i] = lm(Xi,y)
      }
   }
   # add best to Xg
   # (AIC .. Akaike Information Criterion)
}
```

- **#4 API Limitations**
  - Precompiled scripts/programs (inputs unavailable)

- **(#5 Compiler Limitations)**

➔ **Dynamic recompilation techniques** as robust fallback strategy
- Shares goals and challenges with adaptive query processing
- However, ML domain-specific techniques and rewrites

Recompilation

Script

~100 ms — Language
- **Parsing** (syntactic analysis)
- **Live Variable Analysis**
- **Validate** (semantic analysis)

~10 ms — HOPs — **Multiple Rounds**
- **Construct HOP DAGs**
- **Static/Dynamic Rewrites**
- **Intra-/Inter-Procedural Analysis**

~1 ms
- **Static/Dynamic Rewrites**
- **Compute Memory Estimates**

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

Execution Plan

Dynamic Recompilation

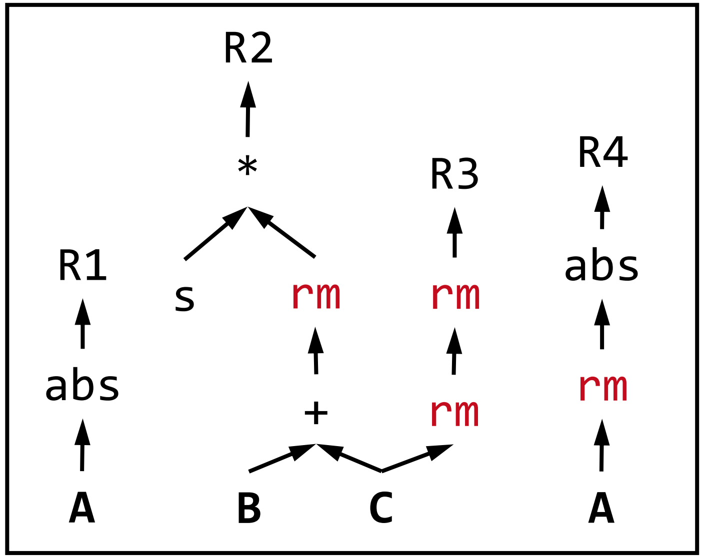
Other systems w/ recompile: **SciDB**, **MatFast**

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

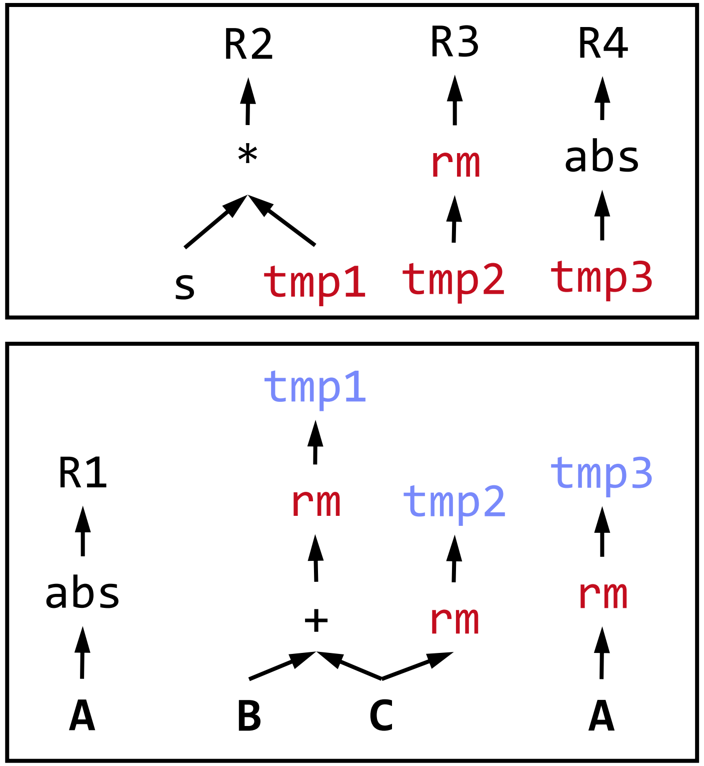
Dynamic Recompilation

- **Compile-time Decisions**
  - **Split HOP DAGs for recompilation:** prevent unknowns but keep DAGs as large as possible; split after reads w/ unknown sizes and specific operators
  - **Mark HOP DAGs for recompilation:** Spark due to unknown sizes / sparsity

Control flow → statement blocks
→ **initial recompilation granularity**



(recursive rewrite)



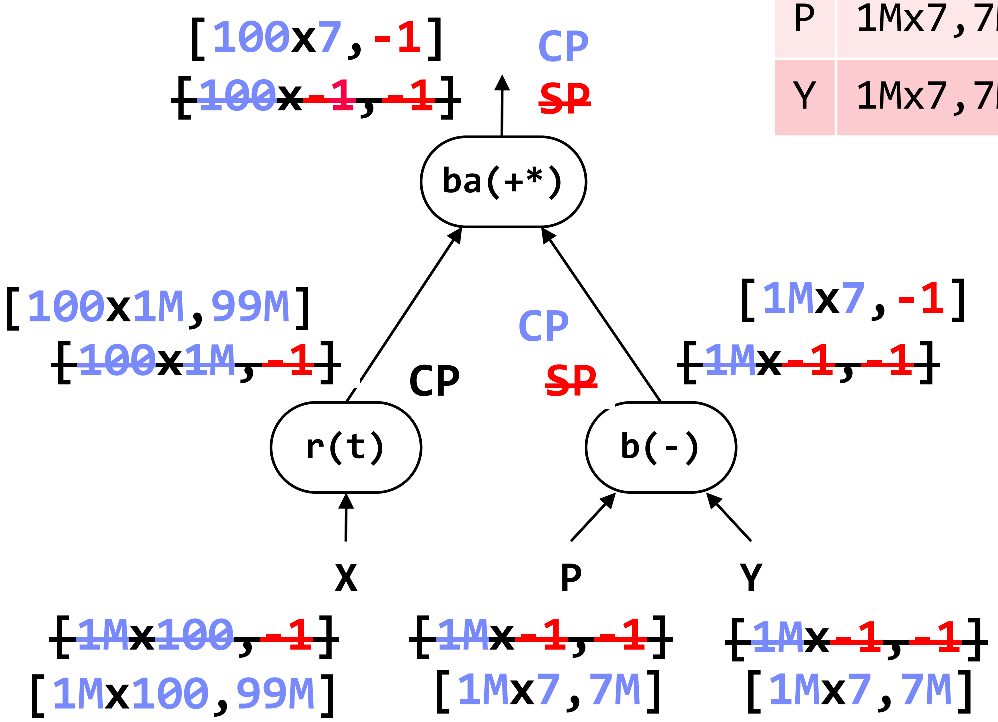

```
rm .. removeEmpty(X,
          [margin="rows",select=I])
```

Dynamic Recompilation, cont.

- **Dynamic Recompilation at Runtime** on recompilation hooks (last level program blocks, predicates, recompile once functions)
  - Deep Copy DAG
  - Replace Literals
  - Update DAG Statistics
  - Dynamic Rewrites
  - Recompute Memory Estimates
  - Execution Type Selection
  - [Codegen]
  - Generate Runtime Instructions

Symbol Table

X	1Mx100,99M
P	1Mx7,7M
Y	1Mx7,7M



Dynamic Recompilation, cont.

- **Recompile Once Functions**
  - Unknowns due to inconsistent or unknown call size information
  - IPA marks functions as "recompile once", if it contains loops
  - **Recompile the entire function on entry + disable unnecessary recompile**

```
foo = function(Matrix[Double] A)
    # recompiled w/ size of A
    return (Matrix[Double] C)
{
    C = rand(nrow(A),1) + A;
    while(...)
          C = C / rowSums(C) * s
}
```

- **Recompile parfor Loops**
  - Unknown sizes and iterations
  - **Recompile parfor loop on entry + disable unnecessary recompile**
  - Create independent DAGs for individual parfor workers

```
while( continue ) {
   parfor( i in 1:n ) {
      if( !fixed[1,i] ) {
         Xi = cbind(Xg, X[,i])
         B[,i] = lm(Xi,y)
      }
   }
}
```

Operator Fusion & JIT Compilation

(aka Code Generation)

Many State-of-the-Art ML Systems,
especially for DNNs and numerical computation

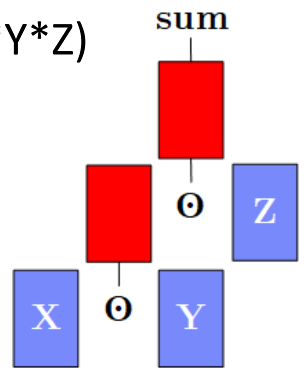
Motivation: Fusion

[Matthias Boehm et al.: On Optimizing Operator Fusion Plans for Large-Scale ML in SystemML. **PVLDB 2018**]

- **Data Flow Graphs** (**better data access**)
  - DAGs of linear algebra (LA) operations and statistical functions
  - Materialized intermediates → **ubiquitous fusion opportunities**

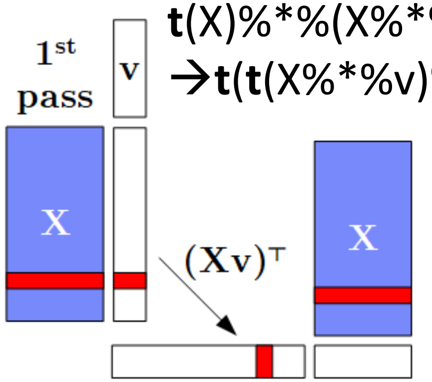
a) Intermediates

sum(X*Y*Z)

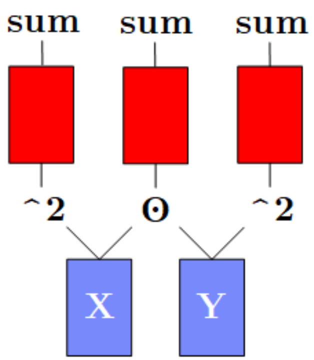


b) Single-Pass

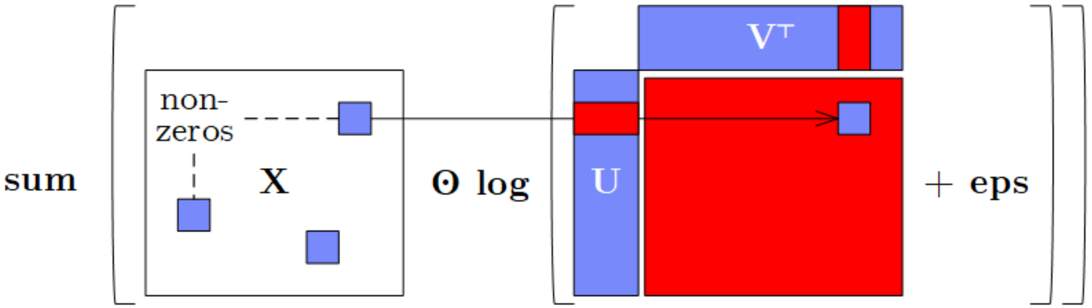
t(X)%*%(X%*%v)
→t(t(X%*%v)%*%X)



c) Multi-Aggregates



d) Sparsity Exploitation

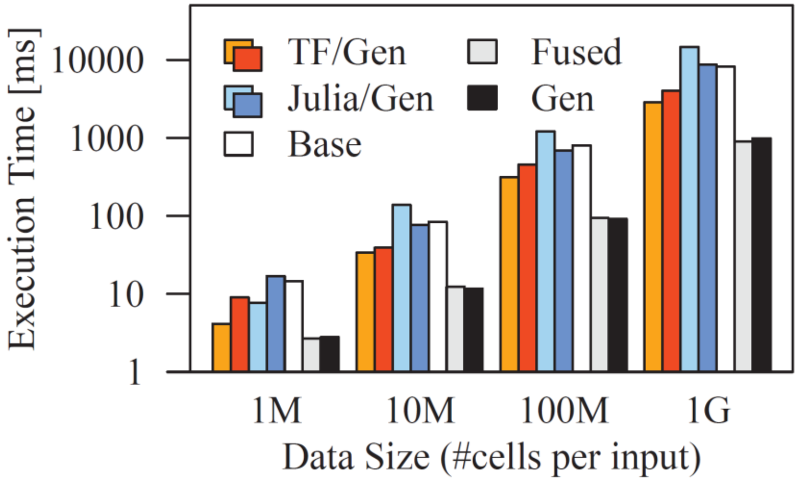


Motivation: Fusion, cont.

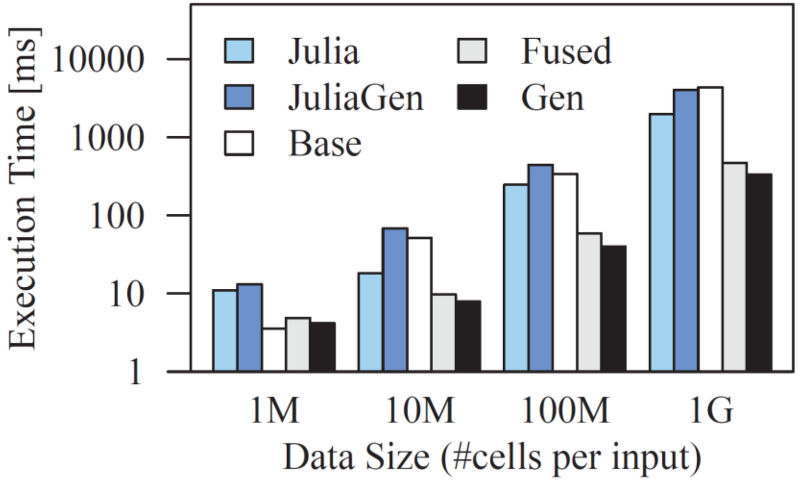
Beware: SystemML 1.0, Julia 0.6.2, TensorFlow 1.5

Cell Template: `sum(X*Y*Z)`

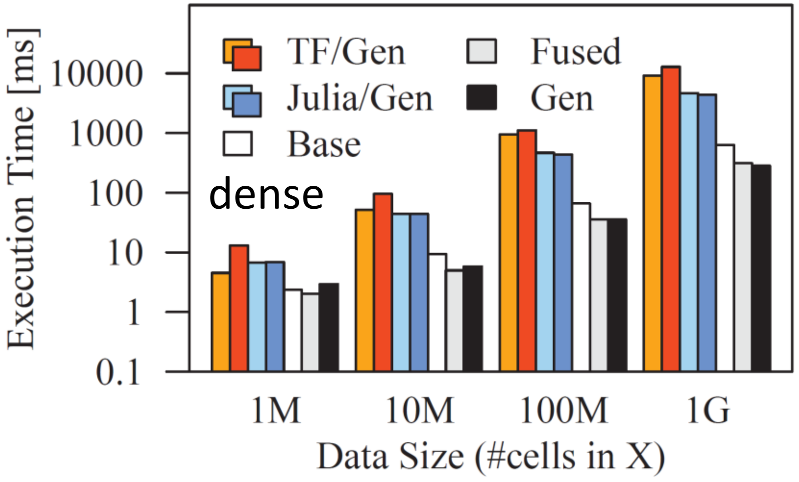
dense



sparse (0.1)



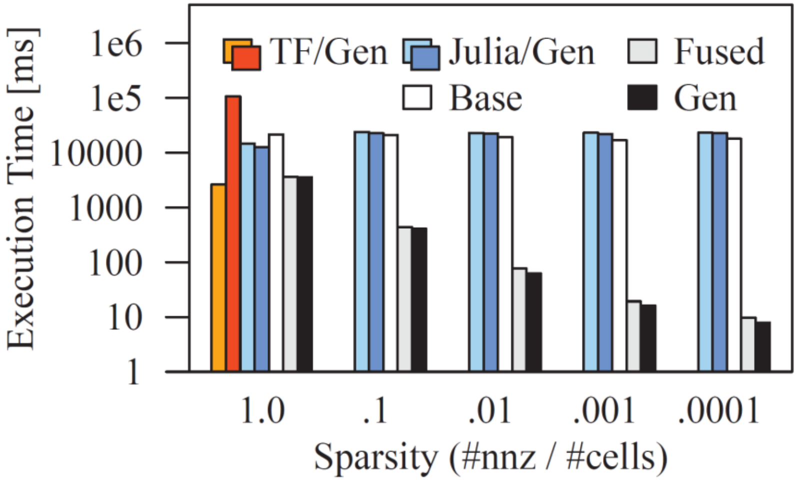
Row: `t(X)%*%(w*(X%*%v))`



TF w/ manual rewrite
→ **t**(**t**(w*(X%*%v))%*%X):
9.2 s to 1.6 s
(compared to Gen 283ms)

Outer: `sum(X*log(U%*%t(V)+1e-15))`

20K x 20K,
rank 100



Motivation: Just-In-Time Compilation

- **Operator Kernels** (**better code**)
  - Specialization opportunities: data types, shapes, and operator graphs
  - Heterogeneous hardware: CPUs, GPUs, FPGAs, ASICs x architectures

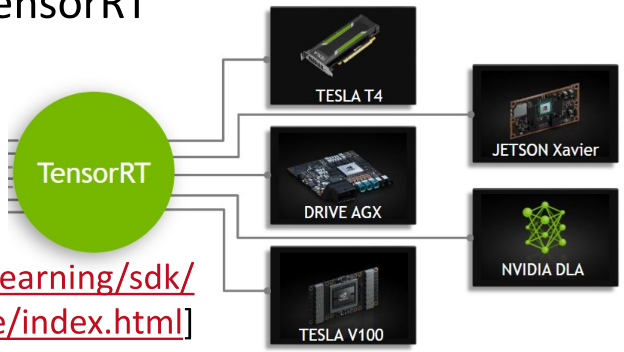
- **#1 CPU Architecture**
  - Specialize to available instructions sets
  - Register allocation and assignment, etc

Examples: x86-64, sparc, amd64, arm, ppc

- **#2 Heterogeneous Hardware**
  - JIT compilation for custom-build ASICs with HW support for ML ops
  - Different architectures of devices

Example: NVIDIA TensorRT

GPU Platforms



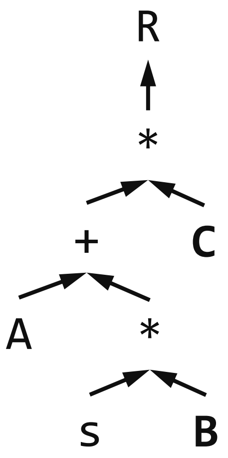
[https://docs.nvidia.com/deeplearning/sdk/tensorrt-developer-guide/index.html]

- **#3 Custom ML Program**
  - Operator graphs and sizes

Operator Fusion Overview – Basic Concept

- **Related Research Areas**
  - HPC: loop fusion, tiling, and distribution (**NP complete**)
  - DB: query compilation / ML: operator fusion (dependencies given by data flow graph)

- **Example Operator Fusion**




```
for( i in 1:n )
  tmp1[i,1] = s * B[i,1];
for( i in 1:n )
  tmp2[i,1] = A[i,1] + tmp1[i,1];
for( i in 1:n )
  R[i,1] = tmp2[i,1] * C[i,1];
```

```
for( i in 1:n )
  R[i,1] = (A[i,1] + s*B[i,1]) * C[i,1];
```

Memory Bandwidth:
L1 core: 1TB/s
L3 socket: 400GB/s
Mem: 100 GB/s

[https://software.intel.com/en-us/articles/memory-performance-in-a-nutshell]

- **Sparse Codegen** [**Credit:** Fredrik Kjolstad]
  - https://cs343d.github.io/lectures/lecture7.iteration1.pdf
  - https://cs343d.github.io/lectures/lecture8.iteration2.pdf

Excursus: Operator Fusion in Large Language Models (LLMs)

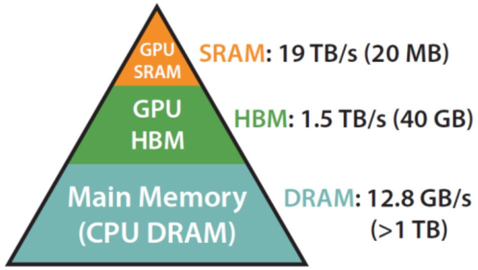
- **Transformer Architecture**
  - Key component: attention mechanisms
  - **Slow/large memory req on long sequences**

[Ashish Vaswani et al: Attention is All you Need. **NeurIPS 2017**]

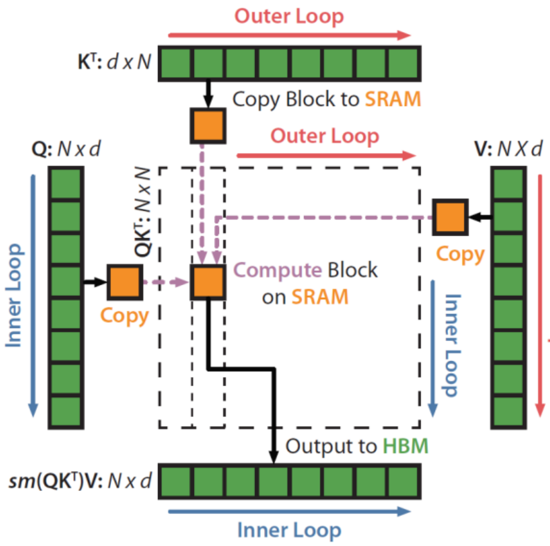
- **FlashAttention**
  - Fused kernel w/ blocking avoids materializing NxN attention matrix

[Tri Dao et al: FlashAttention: Fast and Memory-Efficient Exact Attention with IO-Awareness. **NeurIPS 2022**]

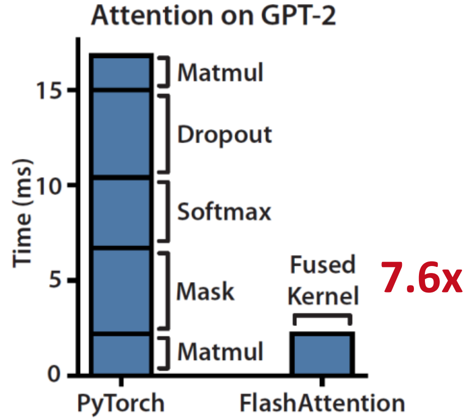
- **Large Language Models**
  - GPT-2 based on Transformer
  - GPT-3 based on SparseTransformer (dense/sparse)
  - MPT-7B (May 05, 2023) based on FlashAttention
    https://www.mosaicml.com/blog/mpt-7b



Memory Hierarchy with Bandwidth & Memory Size



FlashAttention



BS/MS/SHK Projects: Fused Kernels for SparseTransformers

Evolution of Operator Fusion in ML Systems

- **1st Gen: Handwritten Fused Operators**
  - [BLAS (since 1979): e.g., `alpha * X + Y` → `AXPY`]
  - Rewrites: e.g., A+B+C → `AddN(A, B, C)`, `t(X) %*% (w * (X %*% v))` → `MMCHAIN`
  - Sparsity exploiting fused ops: e.g., `sum(X*log(U%*%t(V)+eps))`

[Arash Ashari: On optimizing machine learning workloads via kernel fusion. **PPOPP 2015**]

[Matthias Boehm: SystemML: Declarative Machine Learning on Spark. **PVLDB 2016**]

- **2nd Gen: Fusion Heuristics**
  - Automatic operator fusion via elementary ops
  - Heuristics for replacing sub-DAGs w/ fused ops

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

- **3rd Gen: Optimized Fusion Plans**
  - Greedy/exact fusion plan (sub-DAG) selection
  - [Greedy/evolutionary kernel implementations]

[Matthias Boehm et al.: On Optimizing Operator Fusion Plans for Large-Scale ML in SystemML. **PVLDB 2018**]

Automatic Operator Fusion System Landscape

System	Year	Approach	Sparse	Distr.	Optimization
BTO	2009	Loop Fusion	No	No	k-Greedy, cost-based
Tupleware	2015	Loop Fusion	No	Yes	Heuristic
Kasen	2016	Templates	(Yes)	Yes	Greedy, cost-based
SystemML	2017	Templates	Yes	Yes	Exact, cost-based
Weld	2017	Templates	(Yes)	Yes	Heuristic
Taco	2017	Loop Fusion	Yes	No	Manuel
Julia	2017	Loop Fusion	Yes	No	Manuel
Tensorflow XLA	2017	Loop Fusion	No	No	Manuel/Heuristic
Tensor Comprehensions	2018	Loop Fusion	No	No	Evolutionary, cost-based
TVM	2018	Loop Fusion	No	No	ML/cost-based
PyTorch	2019	Loop Fusion	No	No	Manual/Heuristic
JAX	2019	N/A	No	No	See TF XLA
OpenAI Triton	2021	Loop Fusion	(Yes)	Yes	Manuel/Heuristic

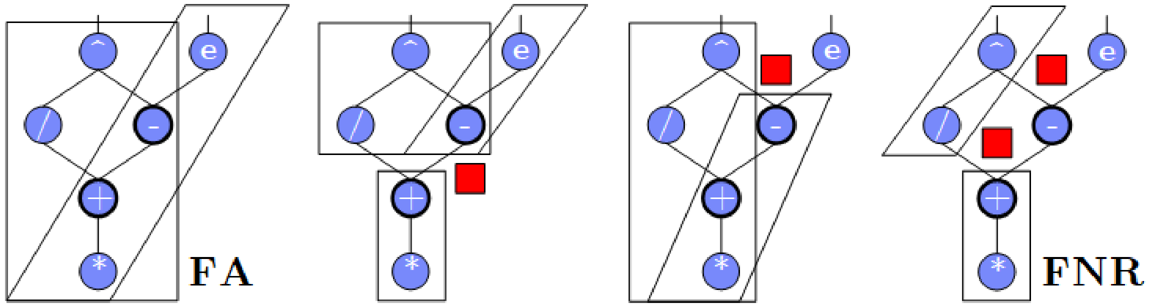
JIT

Use Case SystemDS – A Case for Optimizing Fusion Plans

- **Problem:** Fusion heuristics → **poor plans** for complex DAGs (cost/structure), sparsity exploitation, and local/distributed operations
- **Goal:** **Principled approach for optimizing fusion plans**

$$C = A + s * B$$
$$D = (C/2)\hat{}(C\text{-}1)$$
$$E = \exp(C\text{-}1)$$

- **#1 Materialization Points** (e.g., for multiple consumers)
- **#2 Sparsity Exploitation** (and ordering of sparse inputs)
- **#3 Decisions on Fusion Patterns** (e.g., template types)
- **#4 Constraints** (e.g., memory budget and block sizes)

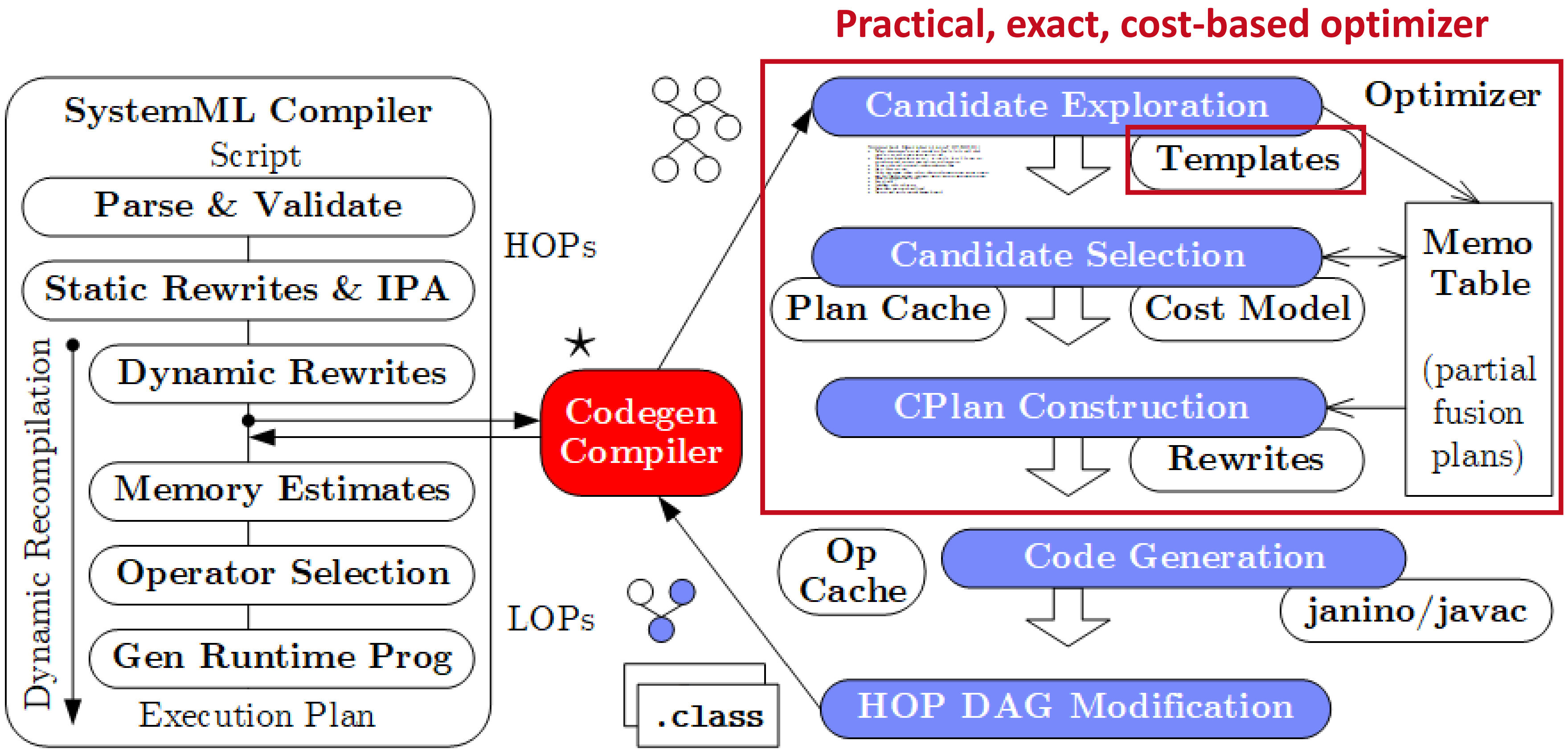



```
Y + X * (U %*% t(V))
```

sparse-safe over X

→ Search Space that requires optimization

Use Case SystemDS – System Architecture



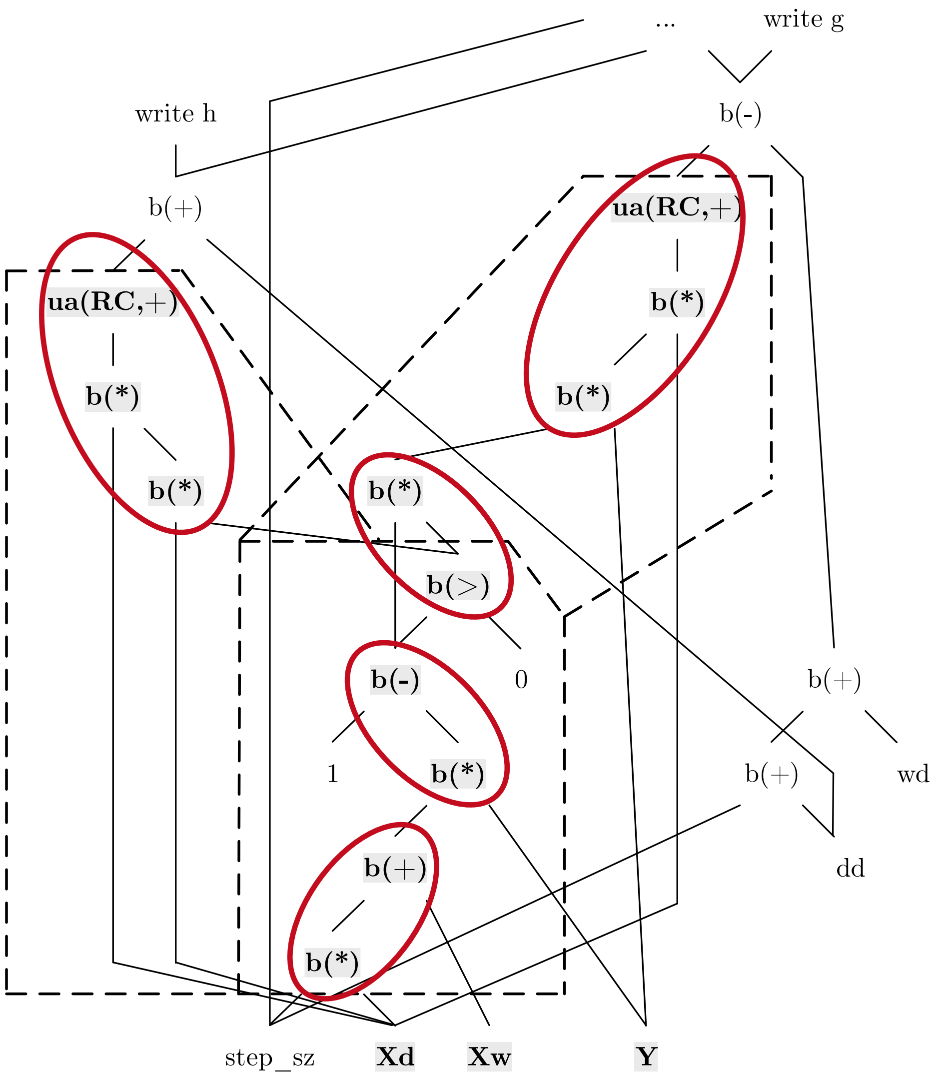
Templates:
Cell, Row, MAgg, Outer
w/ different data bindings

CPlan representation and codegen similar in TF XLA (HLO primitives, pre-clustering of nodes, caching, LLVM codegen)

Use Case SystemDS – Codegen Example L2SVM (Cell/MAgg)

- **L2SVM Inner Loop**

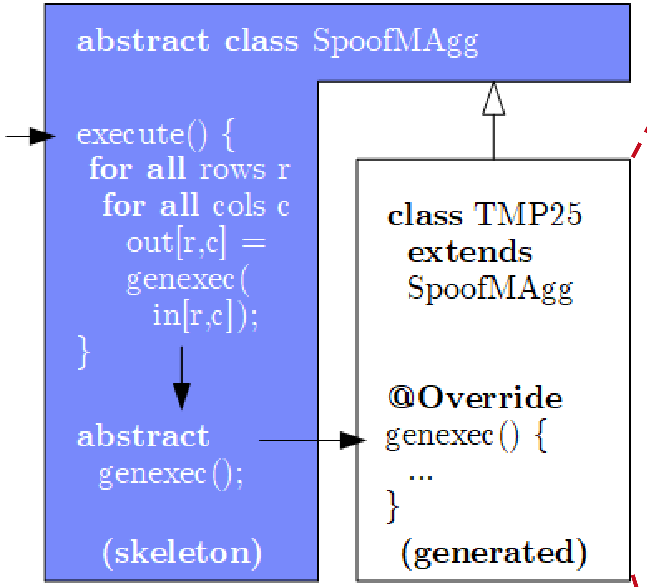
```
while(continueOuter & iter < maxi) {
  #...
  while(continueInner) {
    out = 1-Y* (Xw+step_sz*Xd);
    sv = (out > 0);
    out = out * sv;
    g = wd + step_sz*dd
      - sum(out * Y * Xd);
    h = dd + sum(Xd * sv * Xd);
    step_sz = step_sz - g/h;
}} ...
```



- **# of Vector Intermediates**
  - Base (w/o fused ops): 10
  - Fused (w/ fused ops): 4

Use Case SystemDS – Codegen Example L2SVM, cont. (Cell/MAgg)

- **Template Skeleton**
  - Data access (dense, sparse, compressed)
  - Cache blocking
  - Multi-threading
  - Final aggregation




```
public final class TMP25 extends SpoofMAgg {
  public TMP25() {
    super(false, AggOp.SUM, AggOp.SUM);
  }
  protected void genexec(double a, SideInput[] b,
   double[] scalars, double[] c, ...) {
    double TMP11 = getValue(b[0], rowIndex);
    double TMP12 = getValue(b[1], rowIndex);
    double TMP13 = a * scalars[0];
    double TMP14 = TMP12 + TMP13;
    double TMP15 = TMP11 * TMP14;
    double TMP16 = 1 - TMP15;
    double TMP17 = (TMP16 > 0) ? 1 : 0;
    double TMP18 = a * TMP17;
    double TMP19 = TMP18 * a;
    double TMP20 = TMP16 * TMP17;
    double TMP21 = TMP20 * TMP11;
    double TMP22 = TMP21 * a;
    c[0] += TMP19;
    c[1] += TMP22;
} }
```

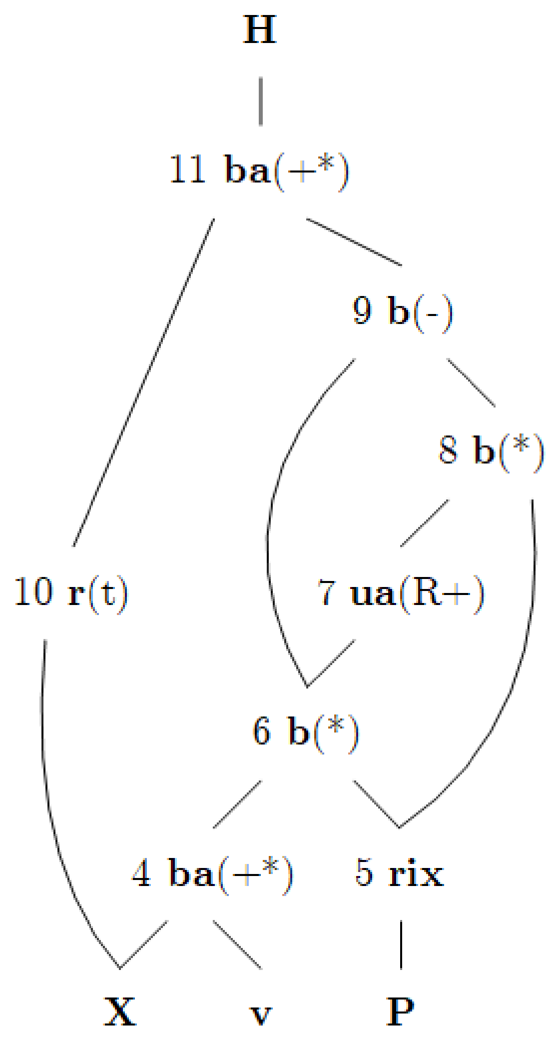
- **# of Vector Intermediates**
  - Gen (codegen ops): 0

Use Case SystemDS – Codegen Example MLogreg (Row)

- **MLogreg Inner Loop** (main expr on feature matrix X)

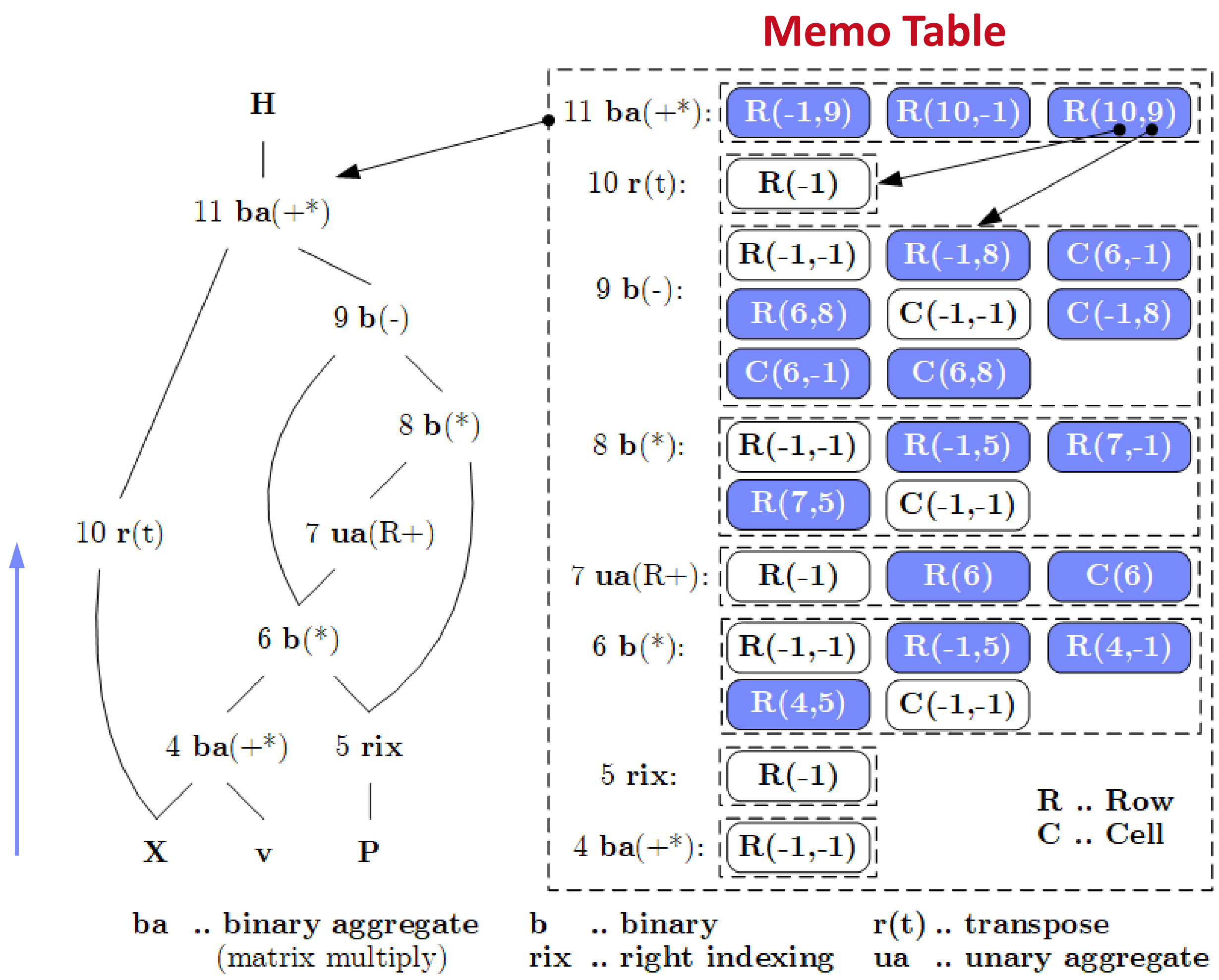
```
1: Q = P[, 1:k] * (X %*% v)
2: H = t(X) %*% (Q - P[, 1:k] * rowSums(Q))
```

```
public final class TMP25 extends SpoofRow {
  public TMP25() {
    super(RowType.COL_AGG_B1_T, true, 5);
  }
  protected void genexecDense(double[] a, int ai,
   SideInput[] b, double[] c,..., int len) {
    double[] TMP11 = getVector(b[1].vals(rix),...);
    double[] TMP12 = vectMatMult(a, b[0].vals(rix),...);
    double[] TMP13 = vectMult(TMP11, TMP12, 0, 0,...);
    double TMP14 = vectSum(TMP13, 0, TMP13.length);
    double[] TMP15 = vectMult(TMP11, TMP14, 0,...);
    double[] TMP16 = vectMinus(TMP13, TMP15, 0, 0,...);
    vectOuterMultAdd(a, TMP16, c, ai, 0, 0,...); }
  protected void genexecSparse(double[] avals, int[] aix,
   int ai, SideInput[] b, ..., int len) {...}
}
```



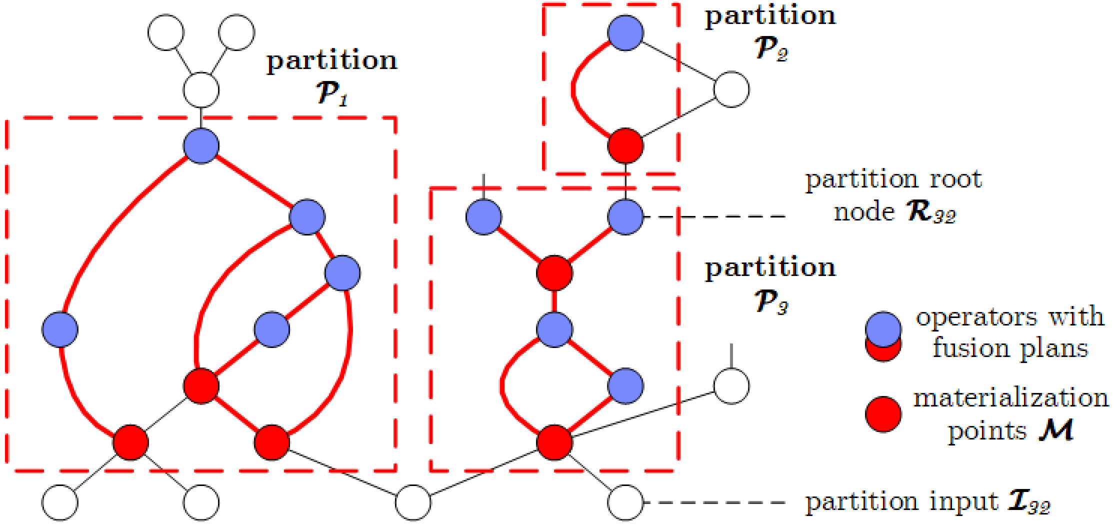
Use Case SystemDS – Candidate Exploration (by example MLogreg)

- **Memo Table for partial fusion plans** (candidates)
- **OFMC Template Fusion API**
  - Open
  - Fuse, Merge
  - Close
- **OFMC Algorithm**
  - **Bottom-up Exploration** (single-pass, template-agnostic)
  - Linear space and time



Use Case SystemDS – Candidate Selection (Partitions and Interesting Points)

- **#1 Determine Plan Partitions**
  - Materialization Points M
  - Connected components of fusion references (in memo)
  - Root and input nodes
  - → Optimize partitions independently



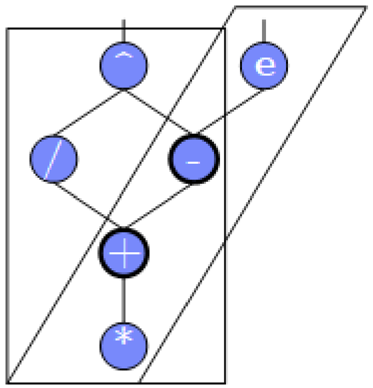
- **#2 Determine Interesting Points**
  - **Materialization Point Consumers:** Each data dependency on materialization points considered separately
  - **Template / Sparse Switches:** Data dependencies where producer has templates non-applicable for consumers
  - → **Optimizer considers all $2^{|M'i|}$ plans** (with $|M'_i| \geq |M_i|$) per partition

Use Case SystemDS – Candidate Selection, cont. (Costs and Constraints)

- **Overview Cost Model**
  - Cost partition with **analytical cost model** (see 03 Compilation)
  - Plan comparisons / fusion errors don't propagate / dynamic recompilation

$$C(\mathcal{P}_i|\mathbf{q}) = \sum_{p \in \mathcal{P}_i|\mathbf{q}} \left(\hat{T}_p^w + \max\left(\hat{T}_p^r, \hat{T}_p^c\right)\right)$$

- **#3 Evaluate Costs**
  - #1: Memoization of already processed sub-DAGs
  - #2: Account for shared reads and CSEs within operators
  - #3: Account for redundant computation (overlap)
  - ➔ **DAG traversal** and **cost vectors** per fused operator (with memoization of pairs of operators and cost vectors)

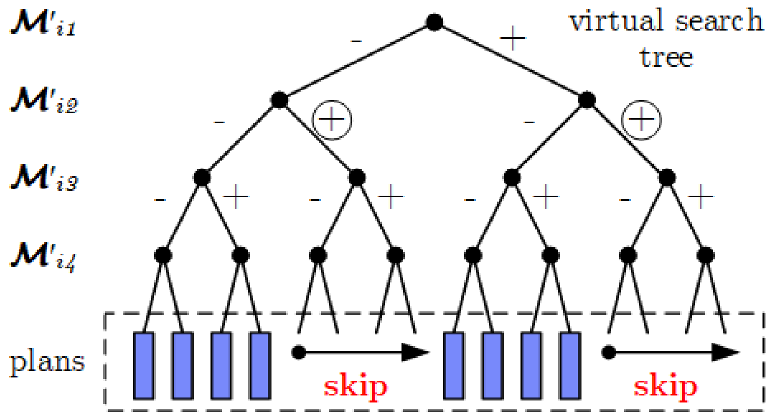


- **#4 Handle Constraints**
  - **Prefiltering** violated constraints (e.g., row template in distributed ops)
  - Assign **infinite costs for violated constraints** during costing

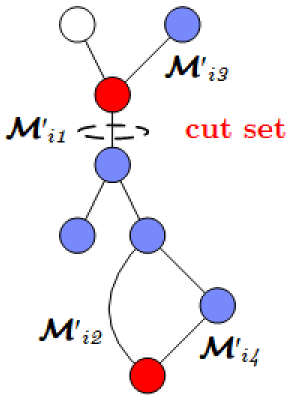
Use Case SystemDS – Candidate Selection, cont. (MPSkipEnum and Pruning)

- **#5 Basic Enumeration**
  - Linearized search space: from - to *

```
for( j in 1:pow(2,|M'_i|) )
   q = createAssignment(j)
   C = getPlanCost(P_i, q)
   maintainBest(q, C)
```



- **#6 Cost-Based Pruning**
  - **Upper bound:** cost C^U of best plan q* (monotonically decreasing)
  - **Opening heuristic:** evaluate FA and FNR heuristics first
  - **Lower bound:** C^{LS} (read input, write output, min compute) + dynamic C^{LD} (materialize intermediates q) ➔ **skip subspace** if $C^U \leq C^{LS} + C^{LD}$



- **#7 Structural Pruning**
  - **Observation:** Assignments can create independent sub problems
  - Build **reachability graph** to determine **cut sets**
  - During enum: probe cut sets, recursive enum, combine, and skip

Ahead-of-Time Compilation

- **TensorFlow `tf.compile`**
  - Compile entire TF graph into binary function w/ low footprint
  - **Input:** Graph, config (feeds+fetches w/ fixes shape sizes)
  - **Output:** x86 binary and C++ header (e.g., inference)
  - **Specialization for frozen model and sizes**

[Chris Leary, Todd Wang: XLA – TensorFlow, Compiled!, **TF Dev Summit 2017**]

- **PyTorch Compile**
  - Compile Python functions into ScriptModule/ScriptFunction
  - Lazily collect operations, optimize, and JIT compile
  - Explicit `jit.script` call or `@torch.jit.script`

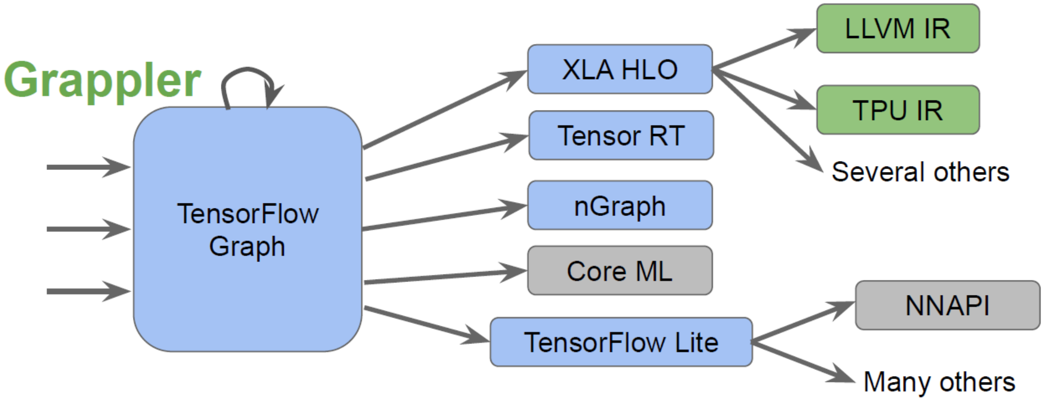
[Vincent Quenneville-Bélair: How PyTorch Optimizes Deep Learning Computations, **Guest Lecture Stanford 2020**]

```
a = torch.rand(5)
def func(x):
  for i in range(10):
    x = x * x # unrolled into graph
  return x

jitfunc = torch.jit.script(func) # JIT
jitfunc.save("func.pt")
```

Excursus: MLIR

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

- **Motivation TF Compiler Ecosystem**
  - Different IRs and compilation chains for runtime backends
  - **Duplication of infrastructure** and fragile error handling
  - **Adoption:** [https://github.com/llvm/torch-mlir]



- **MLIR (Multi-level, Machine Learning IR)**
  - SSA-based IR, similar to LLVM
  - Hierarchy of modules, functions, regions, blocks, and operations
  - **Dialects for different backends** (defined ops, customization)
  - **Systematic lowering**

[Chris Lattner et al.: MLIR: Scaling Compiler Infrastructure for Domain Specific Computation. **CGO 2021**, https://arxiv.org/pdf/2002.11054.pdf]

```
func @testFunction(%arg0: i32) {
  %x = call @thingToCall(%arg0)
    : (i32) -> i32
  br ^bb1
^bb1:
  %y = addi %x, %x : i32
  return %y : i32
}
```

Excursus: MLIR, cont.

(DAPHNE pre-project prototype)

```
while(i < max_iter) { # PageRank
  p = alpha*(G%*%p) + (1-alpha)*(e%*%u%*%p);
  i += 1;
}
```

After Several Optimization Passes

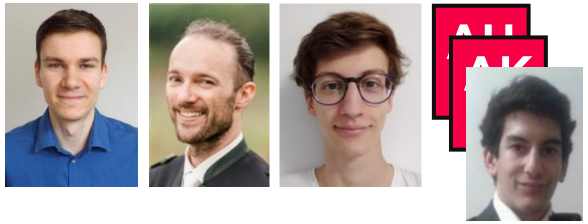
```
module  {
  func @main() {
    %0 = daphne.constant 5.000000e-01 : f64
    %1 = daphne.constant 0 : i64
    %2 = daphne.constant 1.000000e+00 : f64
    %3 = daphne.constant 1 : i64
    %4 = daphne.constant 10 : i64
    %5 = daphne.rand  {cols = 50 : i64, rows = 50 : i64, seed = -1 : i64, sparsity = 7.000000e-02 : f64} : () -> ...
    %6, %7, %8 = ...
    %9 = daphne.sub %2, %0 : (f64, f64) -> f64
    %10:2 = daphne.while (%arg0 = %6, %arg1 = %1) : (!daphne.matrix<50x1xf64>, i64) -> (same) condition: {
      %11 = cmpi "ult", %arg1, %4 : i64
      daphne.yield %11 : i1
    } body: {
      %11 = daphne.mat_mul %5, %arg0 : (!daphne.matrix<50x50xf64>, !daphne.matrix<50x1xf64>) -> !daphne.matrix<50x1xf64>
      %12 = daphne.mul %11, %0 : (!daphne.matrix<50x1xf64>, f64) -> !daphne.matrix<50x1xf64>
      %13 = daphne.mat_mul %8, %arg0 : (!daphne.matrix<1x50xf64>, !daphne.matrix<50x1xf64>) -> !daphne.matrix<1x1xf64>
      %14 = daphne.mat_mul %7, %13 : (!daphne.matrix<50x1xf64>, !daphne.matrix<1x1xf64>) -> !daphne.matrix<50x1xf64>
      %15 = daphne.mul %9, %14 : (f64, !daphne.matrix<50x1xf64>) -> !daphne.matrix<50x1xf64>
      %16 = daphne.add %12, %15 : (!daphne.matrix<50x1xf64>, !daphne.matrix<50x1xf64>) -> !daphne.matrix<50x1xf64>
      %17 = daphne.add %arg1, %3 : (i64, i64) -> i64
      daphne.yield %16, %17 : !daphne.matrix<50x1xf64>, i64
    }
    daphne.print %10#0 : !daphne.matrix<50x1xf64>
    daphne.return
  }
}
```

3) Code motion outside loop

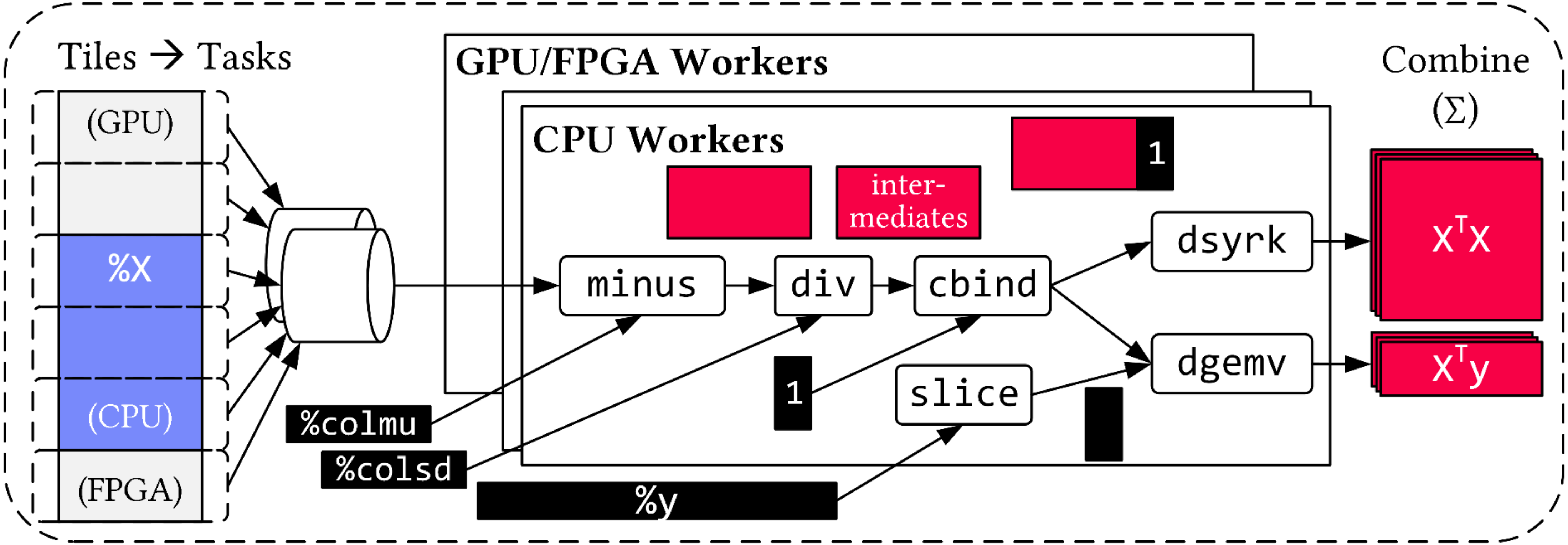
1) Shape inference of dimensions

2) Matrix multiplication chain reordered

Use Case DAPHNE – Vectorized (Tiled) Execution



```
(%9, %10) = fusedPipeline1(%X, %y, %colmu, %colsd) {
```



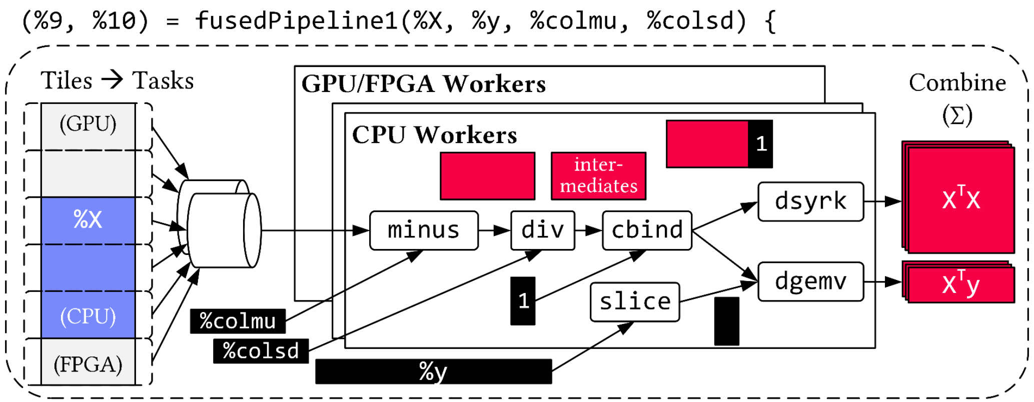
Default Parallelization Frame & Matrix Ops

Locality-aware, Multi-device Scheduling

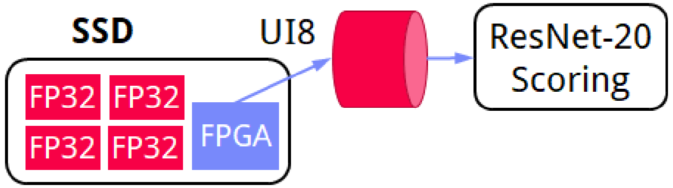
Fused Operator Pipelines on Tiles/Scalars + Codegen

Use Case DAPHNE – Vectorized Execution

- **#1 Zero-copy Input Slicing**
  - Create view on sliced input (no-op)
  - All kernels work on views
- **#2 Sparse Intermediates**
  - Reuse dense/sparse kernels
  - Sparse pipeline intermediates for free
- **#3 Fine-grained Control**
  - Task sizes (dequeue, data access) vs data binding (cache-conscious ops)
  - Scheduling for load balance (e.g., sparse operations)



- **#4 Computational Storage**
  - Task queues connect eBPF programs, async I/O into buffers, and op pipelines



Summary & QA

- **Motivation and Terminology**
- **Runtime Adaptation**
- **Operator Fusion & JIT**

Recommended Reading

[Chris Leary, Todd Wang: XLA – TensorFlow, Compiled!, **TF Dev Summit 2017,** https://www.youtube.com/watch?v=kAOanJczHA0]

➔ **Impact of Size Inference and Costs (lecture 03)**

➔ **Ubiquitous Rewrite, Fusion, and Codegen/JIT Opportunities**

- **Next Lectures** (Runtime Aspects)
  - **Holiday May 18** (Christi Himmelfahrt/Ascension Day)
  - **05 Data- and Task-Parallel Execution** (batch/prog) [May 25]
  - **06 Parameter Servers** (mini-batch) [Jun 01]
  - **07 Hybrid Execution and HW Accelerators** [Jun 08]
  - **08 Caching, Partitioning, Indexing and Compression** [Jun 15]