Reliable Generation of High-Performance Matrix Algebra

Jeremy G. Siek, University of Colorado at Boulder

Joint work with
Liz Jessup,
Boyana Norris,
Geoffrey Belter,
Thomas Nelson

Lake Isabelle, Colorado

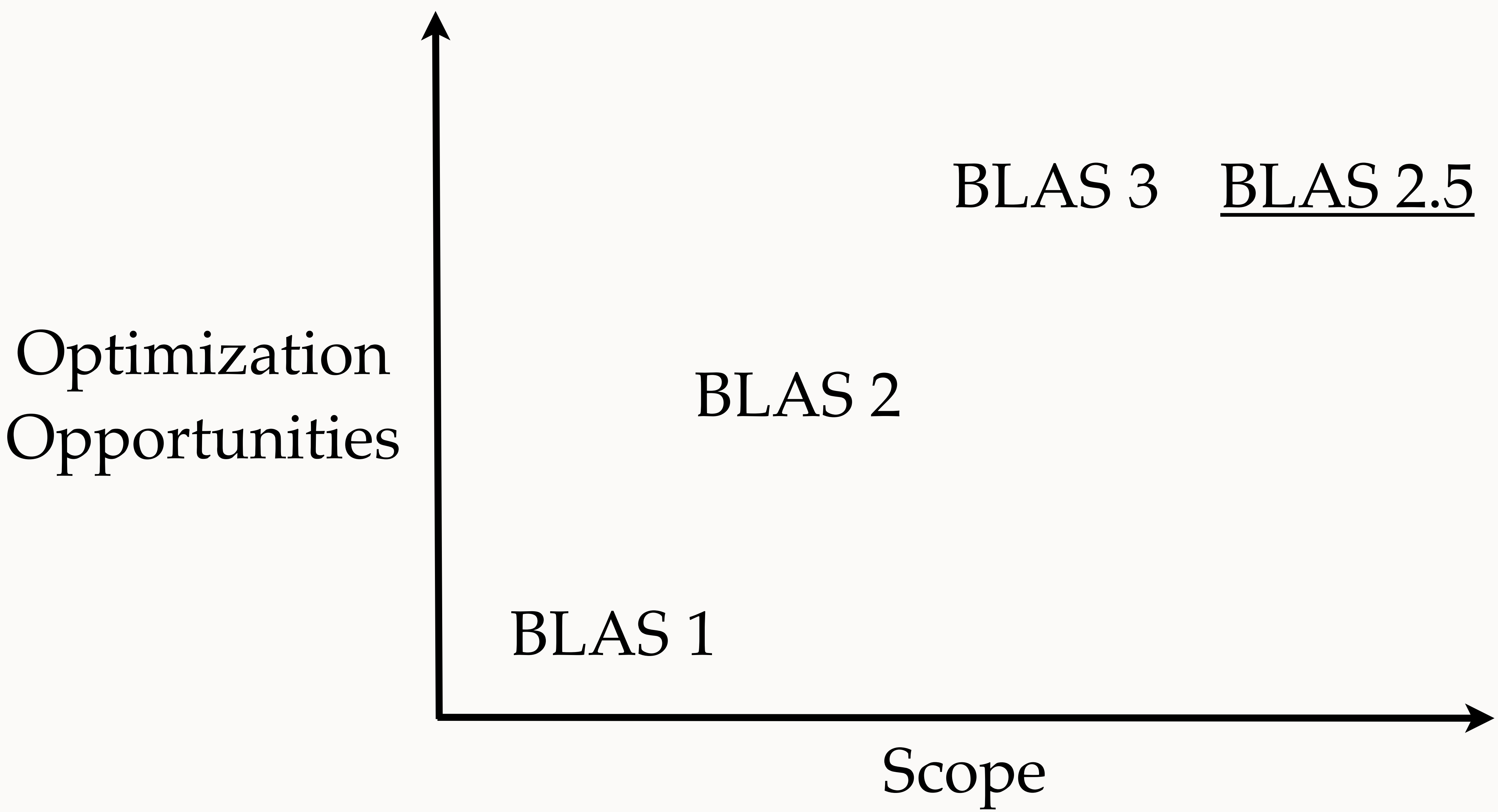
Our SC'12 submission is available on my web page.

ABSTRACTION VS. PERFORMANCE

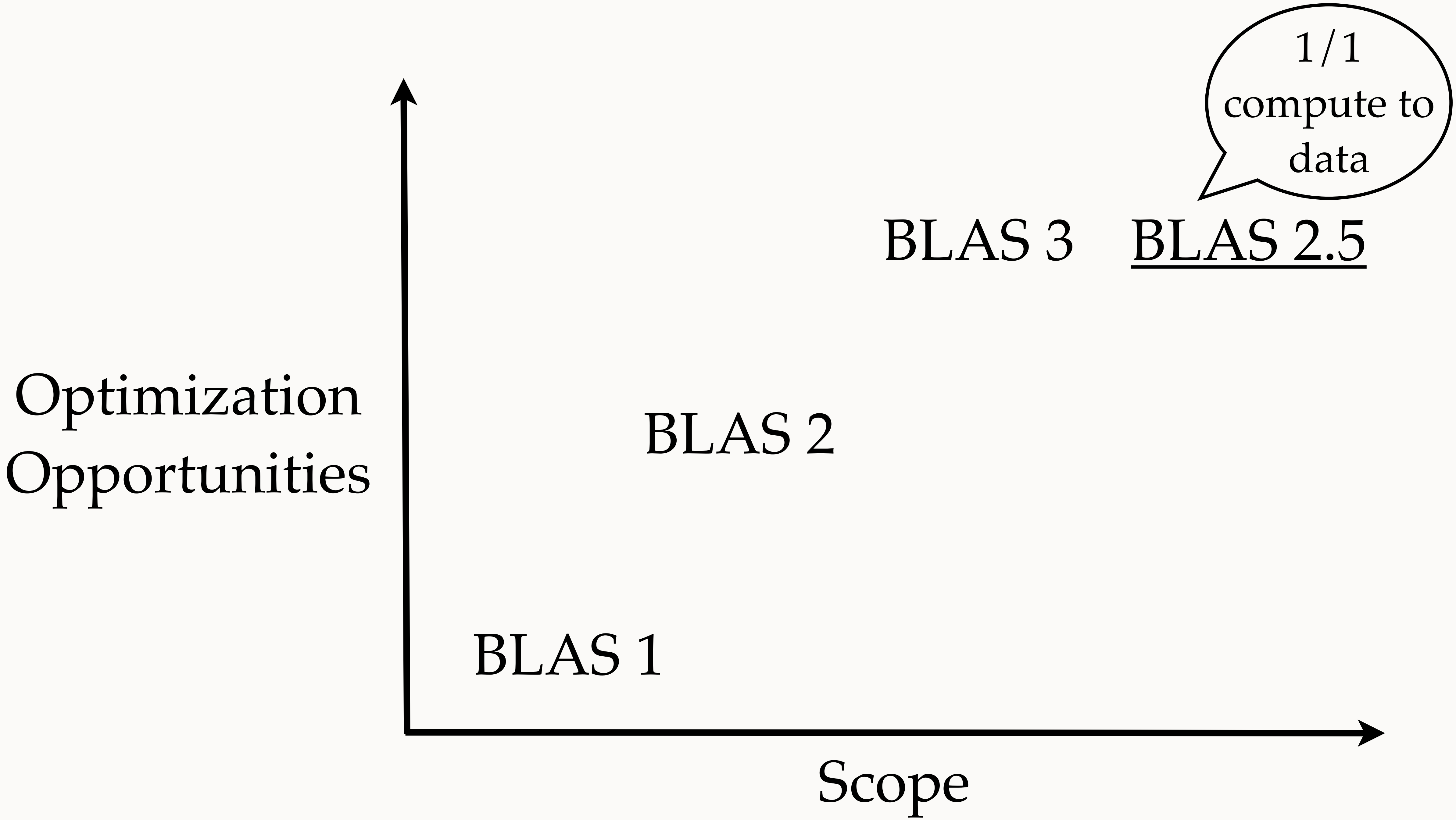
A few lines from the PETSc Biconjugate gradient method.

```
      betaold = beta;
      KSP_MatMult(ksp,Amat,Pr,Zr); /*     z <- Kp    */
      VecConjugate(Pl);
      KSP_MatMultTranspose(ksp,Amat,Pl,Zl);
      VecConjugate(Pl);
      VecConjugate(Zl);
      VecDot(Zr,Pl,&dpi);     /*     dpi <- z'p     */
      a = beta/dpi;           /*     a = beta/p'z   */
      VecAXPY(X,a,Pr);        /*     x <- x + ap    */
```

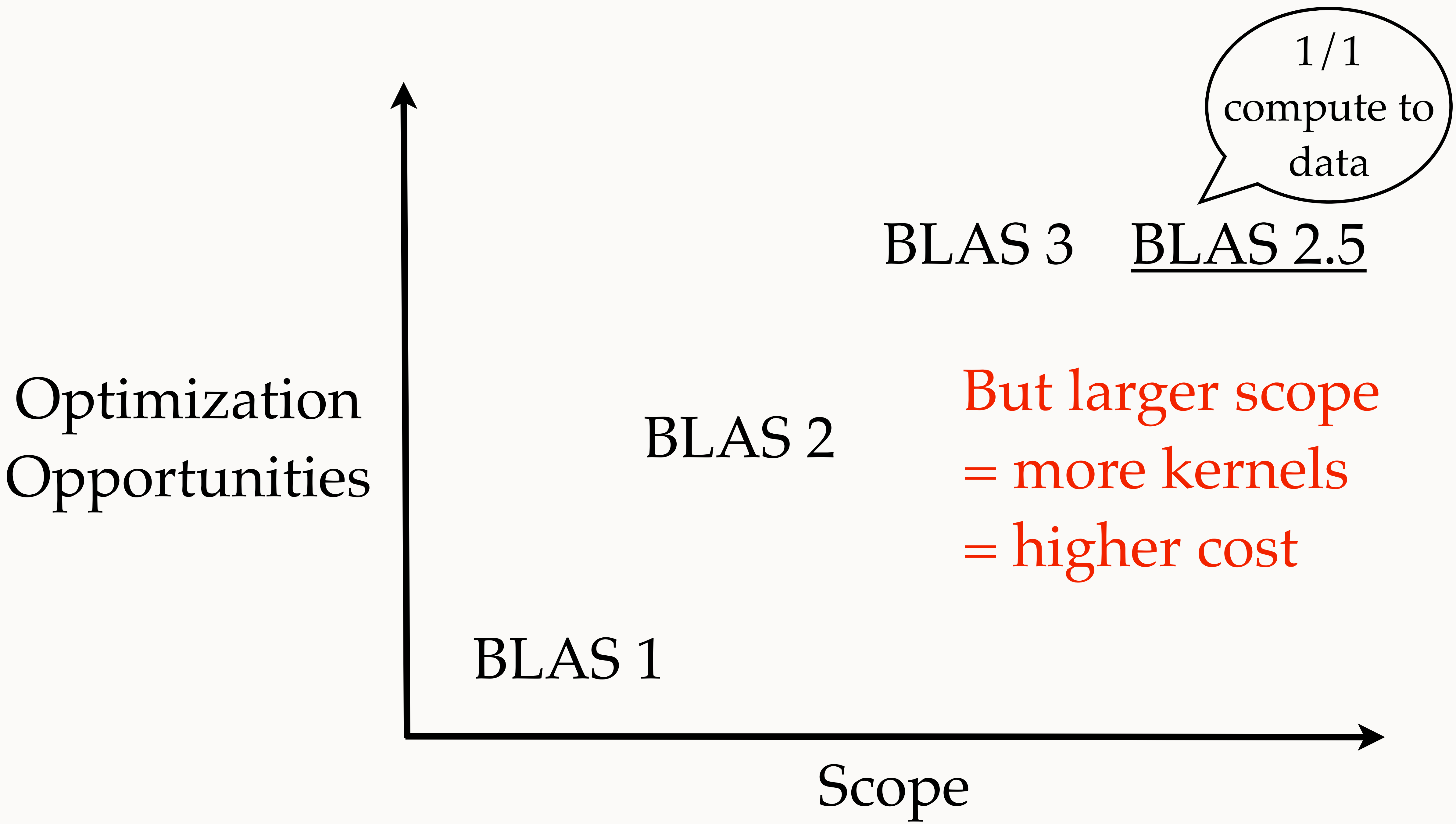
LARGER SCOPE = BETTER PERFORMANCE



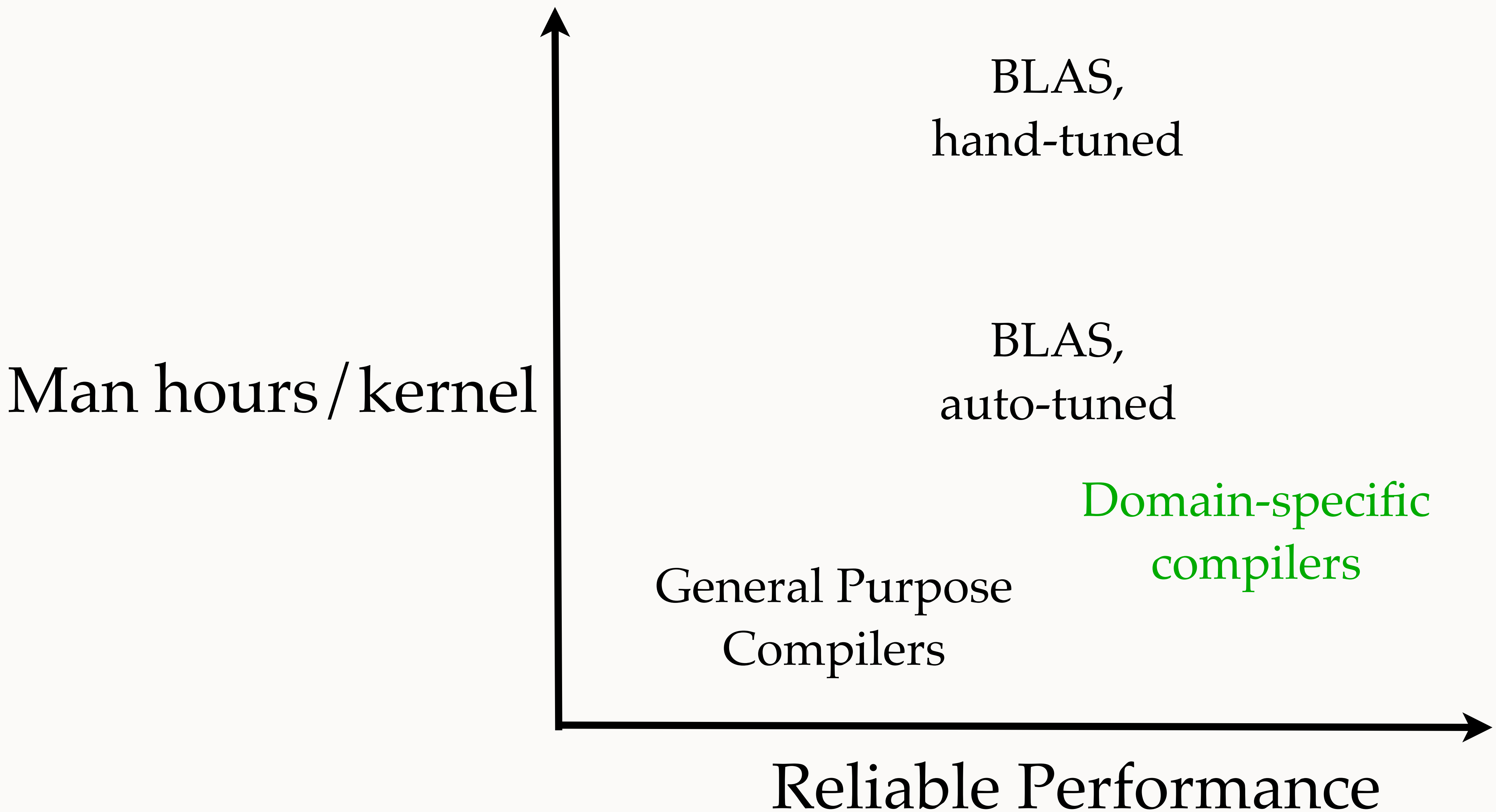
LARGER SCOPE = BETTER PERFORMANCE



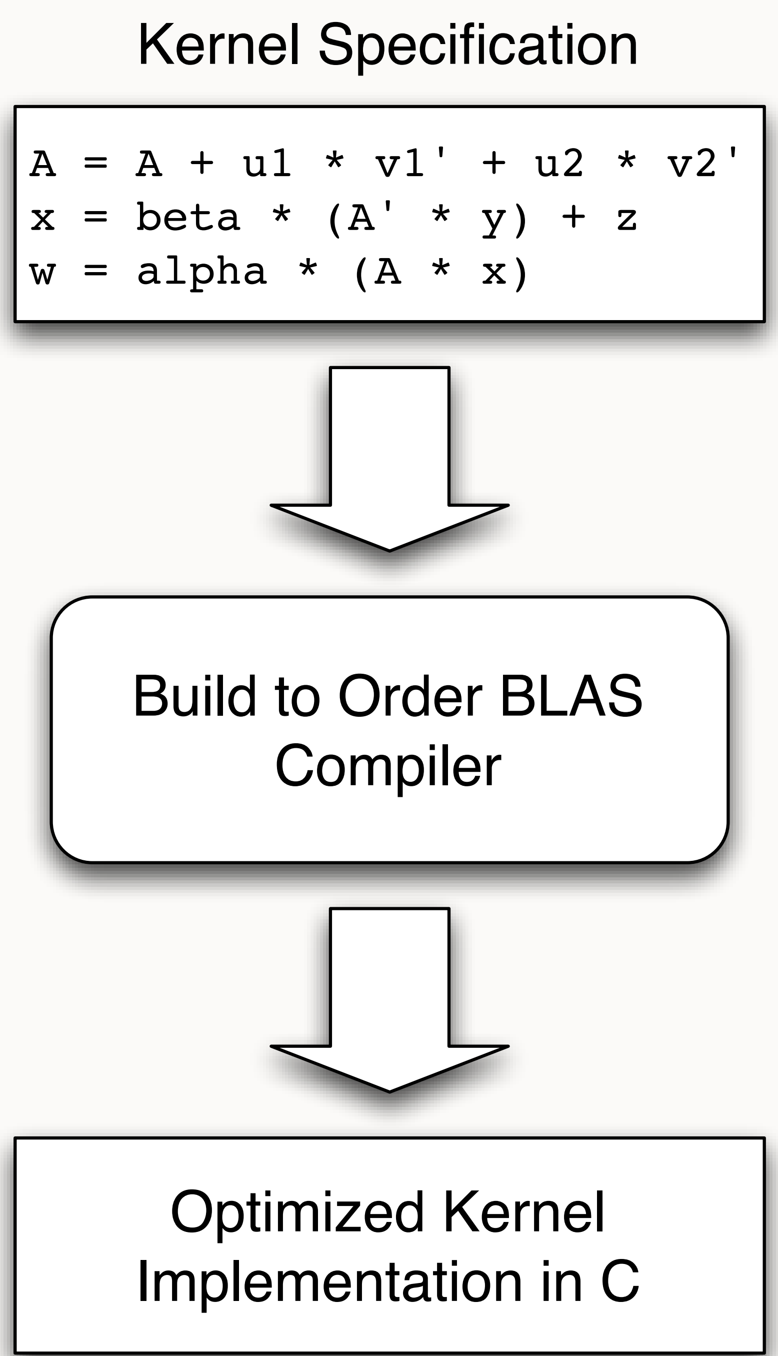
LARGER SCOPE = BETTER PERFORMANCE



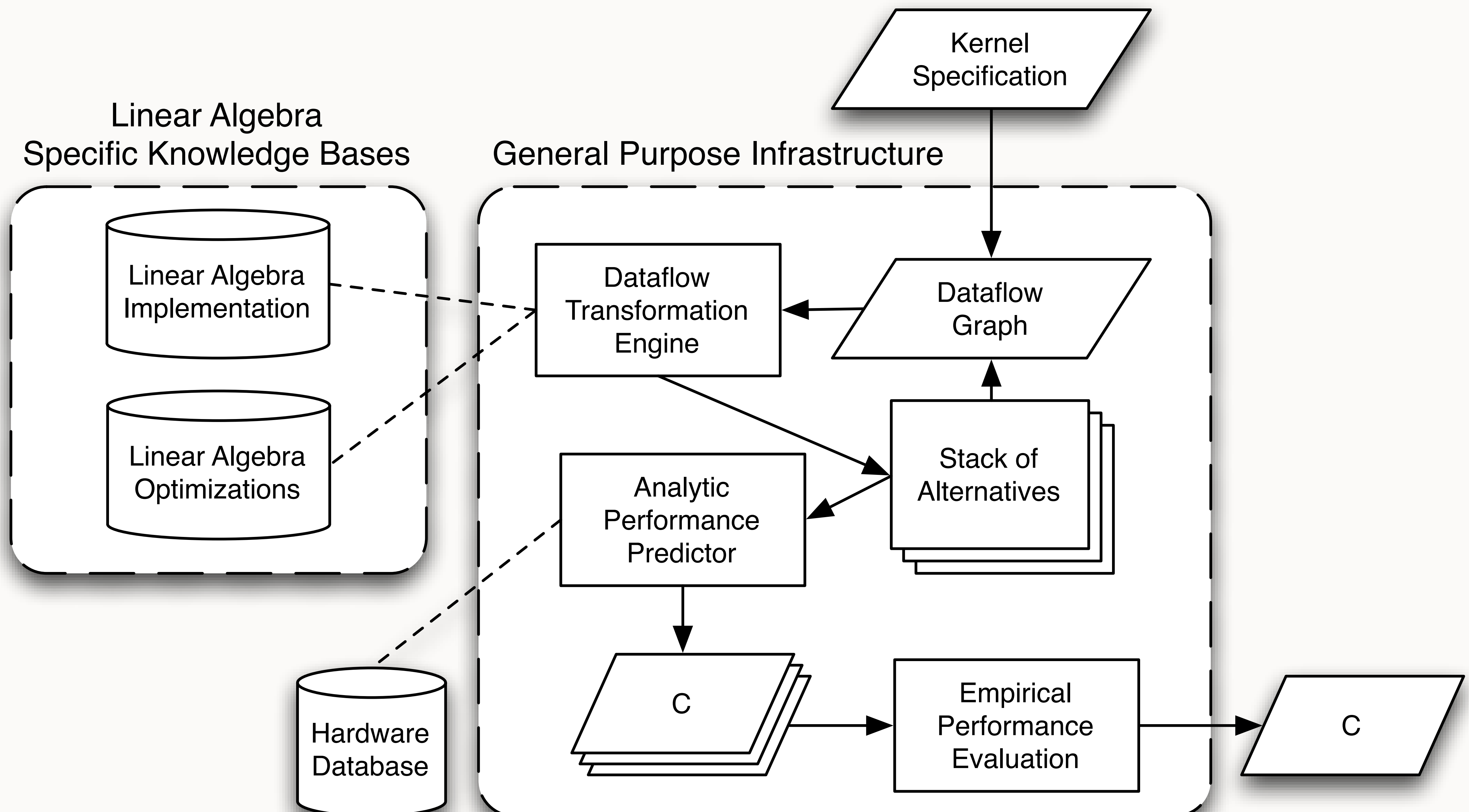
COST & PERFORMANCE



BUILD TO ORDER BLAS

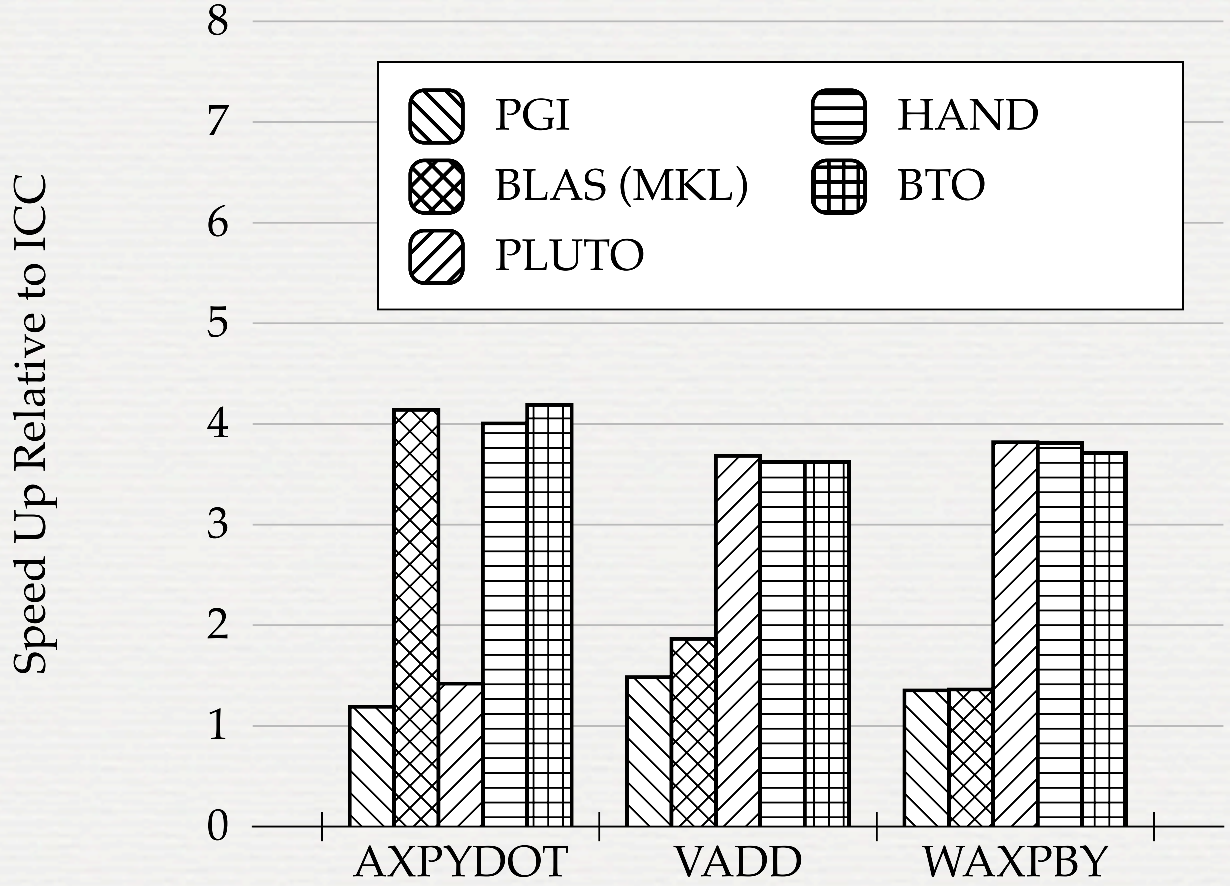


BUILD TO ORDER BLAS



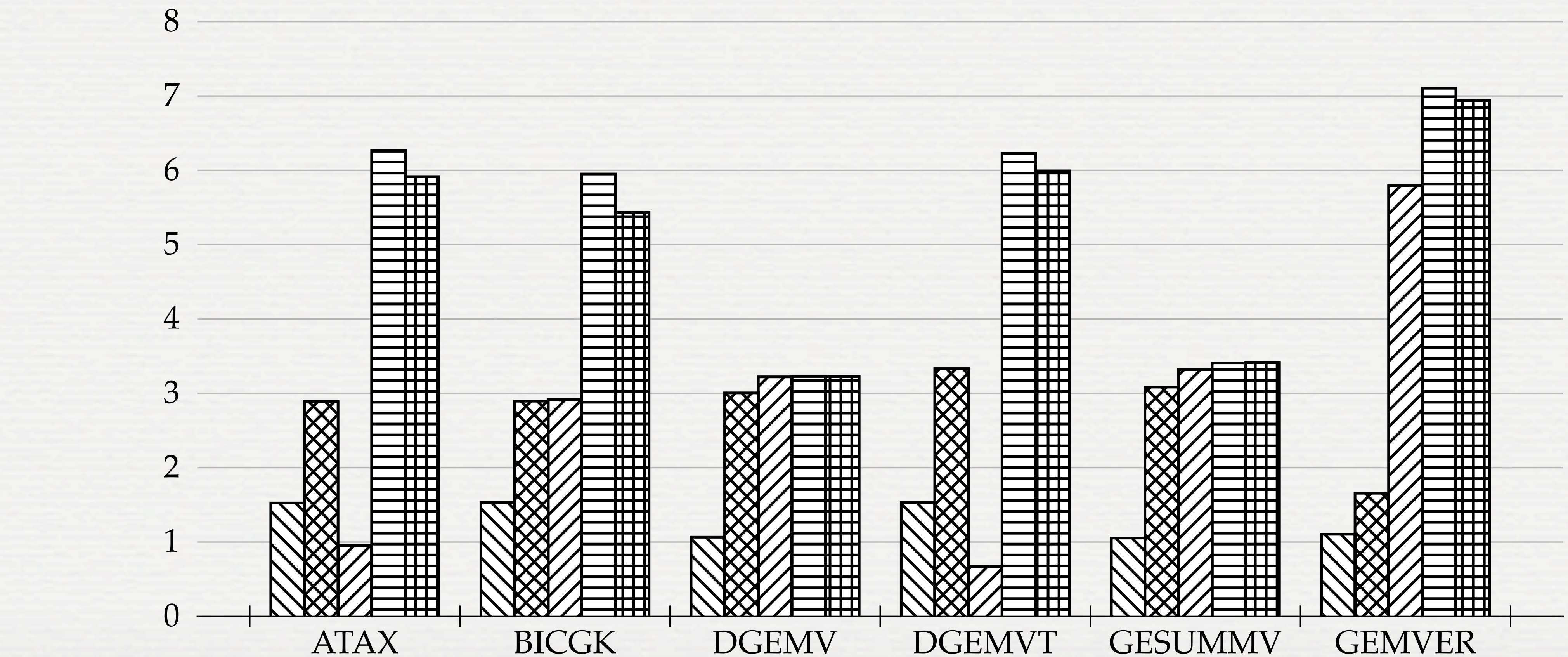
KERNEL SPECIFICATION

```
GEMVER
in u1 : vector, u2 : vector, v1 : vector,
   v2 : vector, alpha : scalar,
   beta : scalar, y : vector, z : vector
inout A : dense column matrix
out x : vector, w : vector
{
   A = A + u1 * v1' + u2 * v2'
   x = beta * (A' * y) + z
   w = alpha * (A * x)
}
```



Kernel	Operation
AXPYDOT	$z \leftarrow w - \alpha v$ $\beta \leftarrow z^T u$
VADD	$x \leftarrow w + y + z$
WAXPBY	$w \leftarrow \alpha x + \beta y$
ATAX	$y \leftarrow A^T A x$
BICGK	$q \leftarrow Ap$ $s \leftarrow A^T r$
DGEMV	$z \leftarrow \alpha A x + \beta y$
DGEMVT	$x \leftarrow \beta A^T y + z$ $w \leftarrow \alpha A x$
GEMVER	$B \leftarrow A + u_1 v_1^T + u_2 v_2^T$ $x \leftarrow \beta B^T y + z$ $w \leftarrow \alpha B x$
GESUMMV	$y \leftarrow \alpha A x + \beta B x$

Intel Westmere, 24 core, 2.66 GHz



RESULTS ON AMD

Speedups relative to PGI

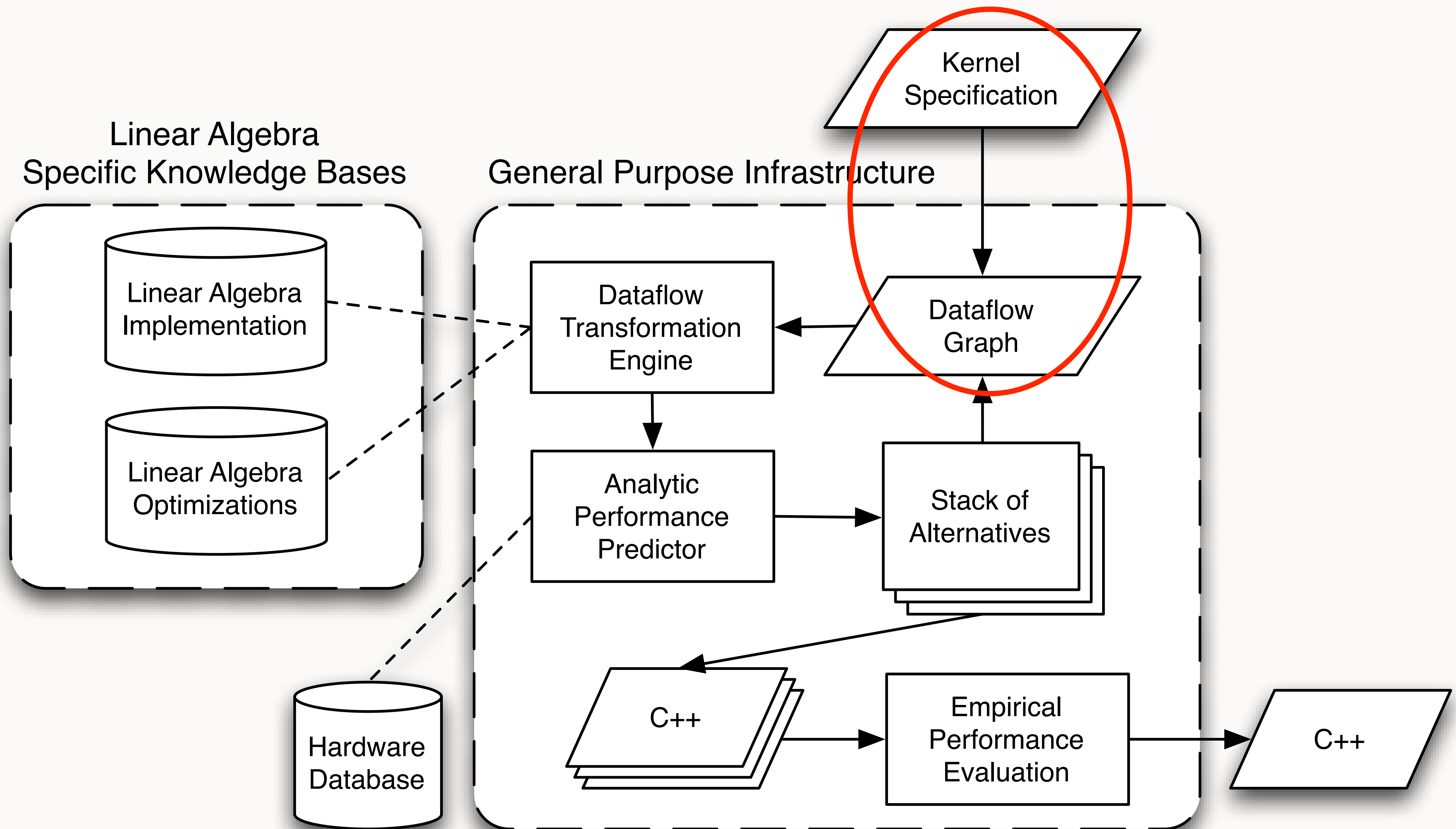
Kernel	BLAS	Pluto	HAND	BTO
AXPYDOT	0.97	1.81	1.58	**1.86**
VADD	0.84	1.33	1.50	**1.83**
WAXPBY	0.79	1.40	1.68	**1.91**
ATAX	1.27	0.69	**2.92**	**2.92**
BICGK	1.27	0.80	2.80	**2.84**
DGEMV	1.67	0.71	1.85	**1.89**
DGEMVT	1.67	0.71	1.85	**1.89**
GEMVER	1.04	1.61	**2.61**	2.34
GESUMMV	1.63	0.63	1.74	**1.75**

AMD Phenom, 6 core, 3.3 GHz

Kernel	BLAS	Pluto	HAND	BTO
AXPYDOT	0.82	1.60	1.73	**1.61**
VADD	0.43	1.05	1.14	**1.15**
WAXPBY	0.34	1.06	**1.16**	1.11
ATAX	2.49	0.43	4.09	**4.28**
BICGK	2.35	1.60	3.03	**4.22**
DGEMV	**2.45**	0.89	1.66	2.07
DGEMVT	2.43	0.43	**4.08**	4.03
GEMVER	1.70	2.00	**4.15**	4.05
GESUMMV	**2.36**	0.37	1.65	2.03

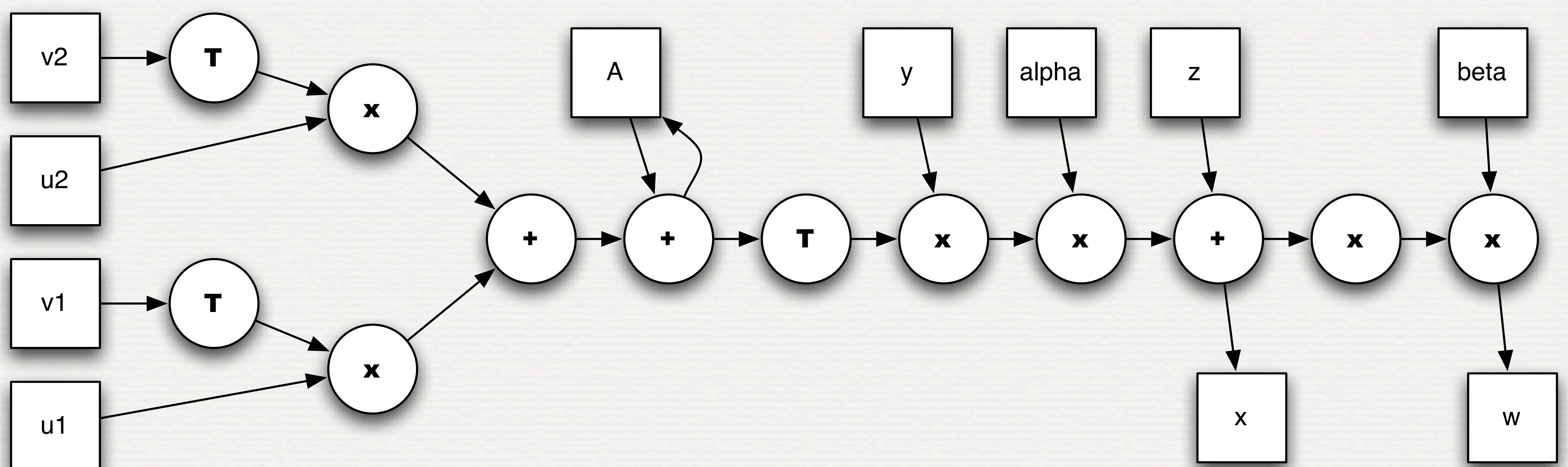
AMD Interlagos, 64 core, 2.2 GHz

BUILD TO ORDER BLAS



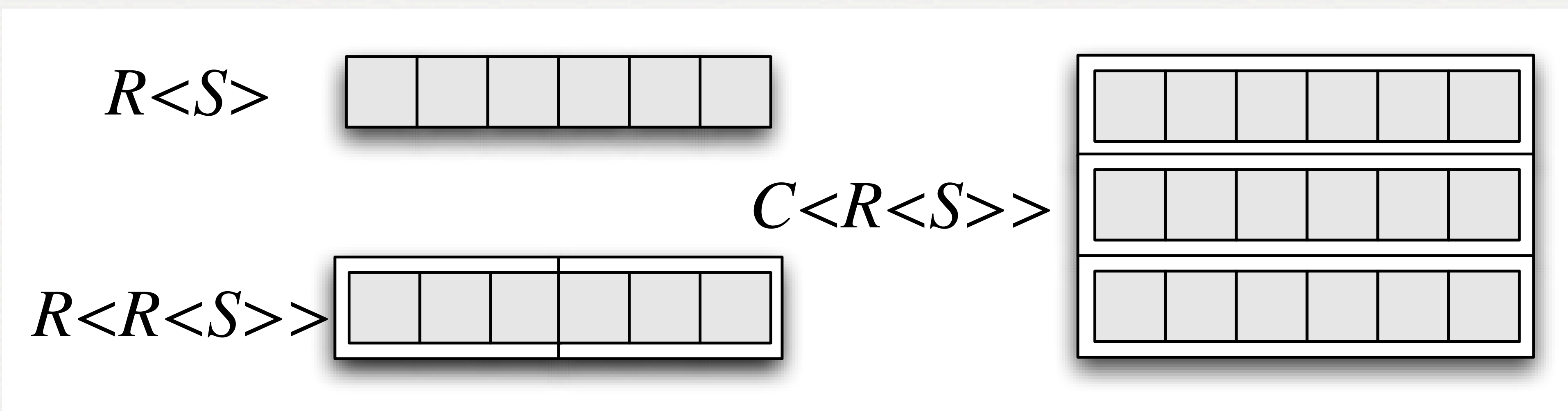
DATAFLOW GRAPH

```
A = A + u1 * v1' + u2 * v2'
x = beta * (A' * y) + z
w = alpha * (A * x)
```



TRAVERSAL PATTERNS

orientations	O	$::=$	$C \mid R$
types	τ	$::=$	$O{<}\tau{>} \mid S.$

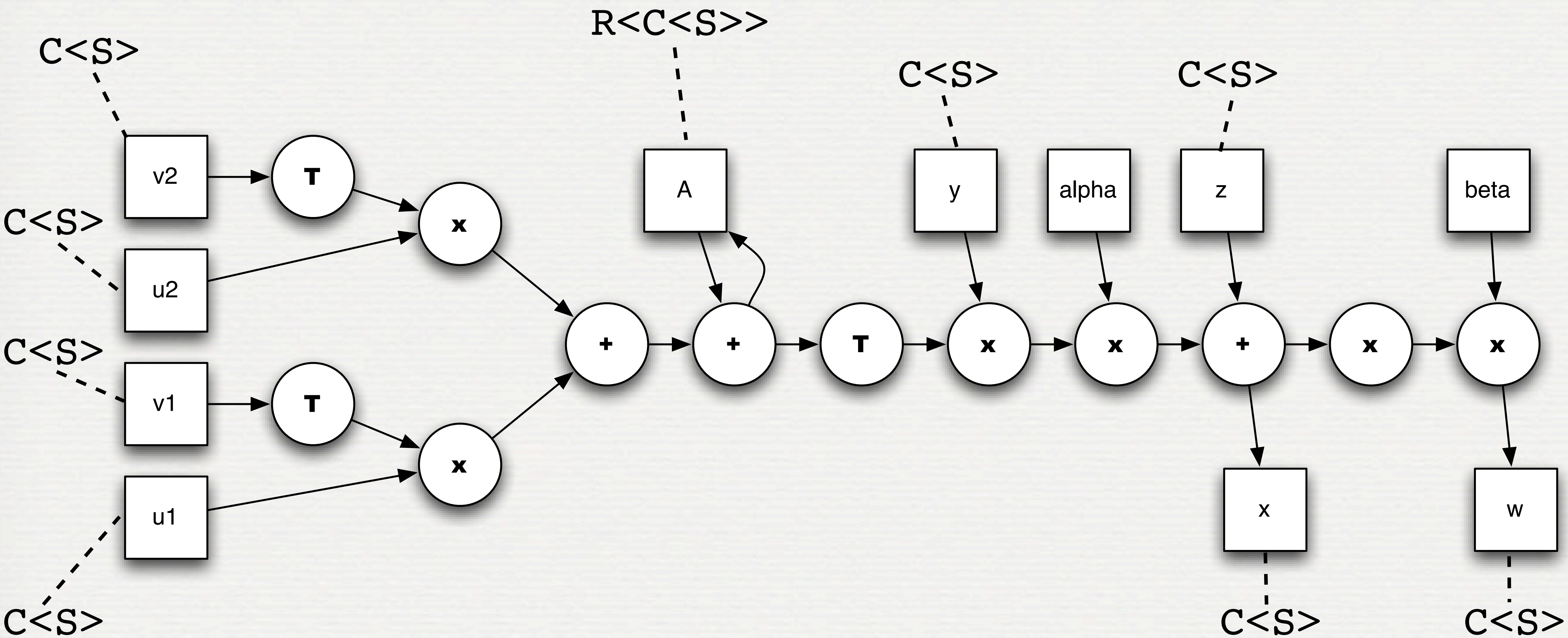


LINEAR ALGEBRA DB

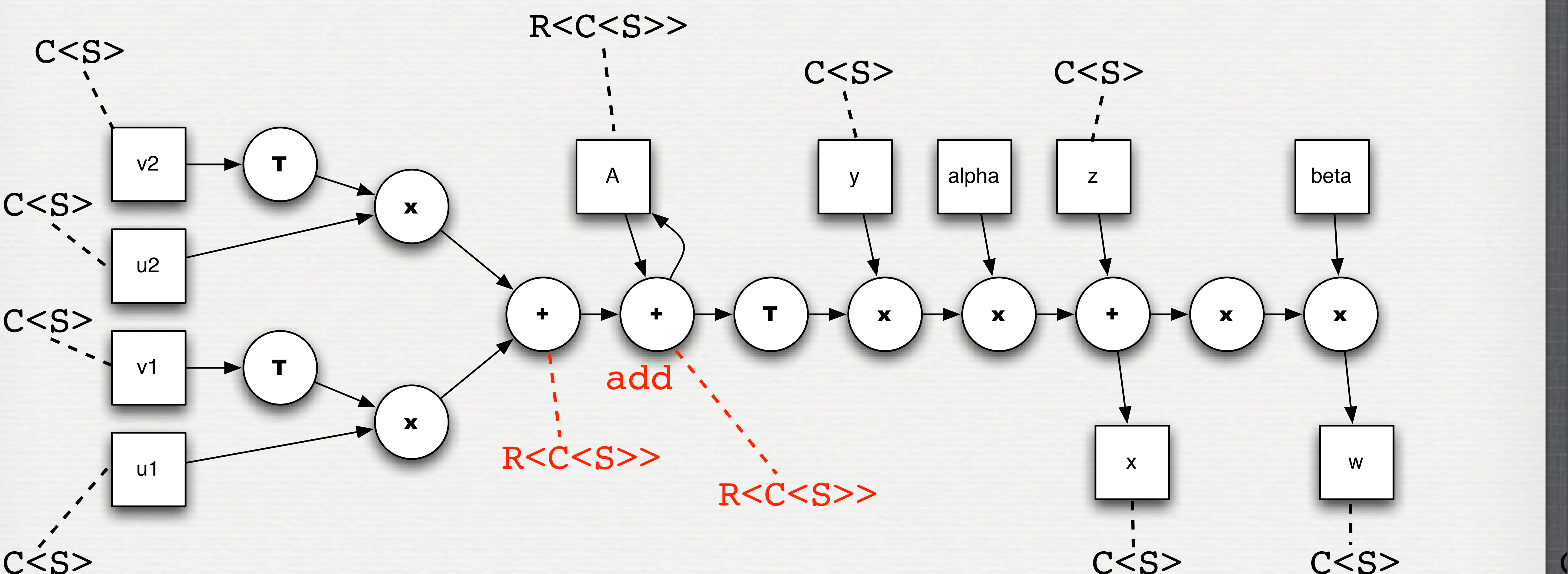
Algo	Op and Operands	Result Type	Pipe
`add`	$O<\tau_l> + \ O<\tau_r>$	$O<\tau_l + \tau_r>$	yes
`s-add`	$S + S$	S	no
`trans`	$O<\tau>^T$	$O^T<\tau^T>$	yes
`s-mult`	$S \times S$	S	no
`rr-mult`	$R<\tau_l> \times \ R<\tau_r>$	$R<R<\tau_l> \times \ \tau_r>$	yes
`cc-mult`	$C<\tau_l> \times \ C<\tau_r>$	$C<\tau_l \times \ C<\tau_r>>$	yes
`dot`	$R<\tau_l> \times \ C<\tau_r>$	$\sum(\tau_l \times \ \tau_r)$	no
`outer1`	$C<\tau_l> \times \ R<\tau_r>$	$C<\tau_l \times \ R<\tau_r>>$	yes
`outer2`	$C<\tau_l> \times \ R<\tau_r>$	$R<C<\tau_l> \times \ \tau_r>$	yes
`scale`	$S \times \ O<\tau>$	$O<S \times \ \tau>$	yes

Table 1: Sample of the linear algebra knowledge base.

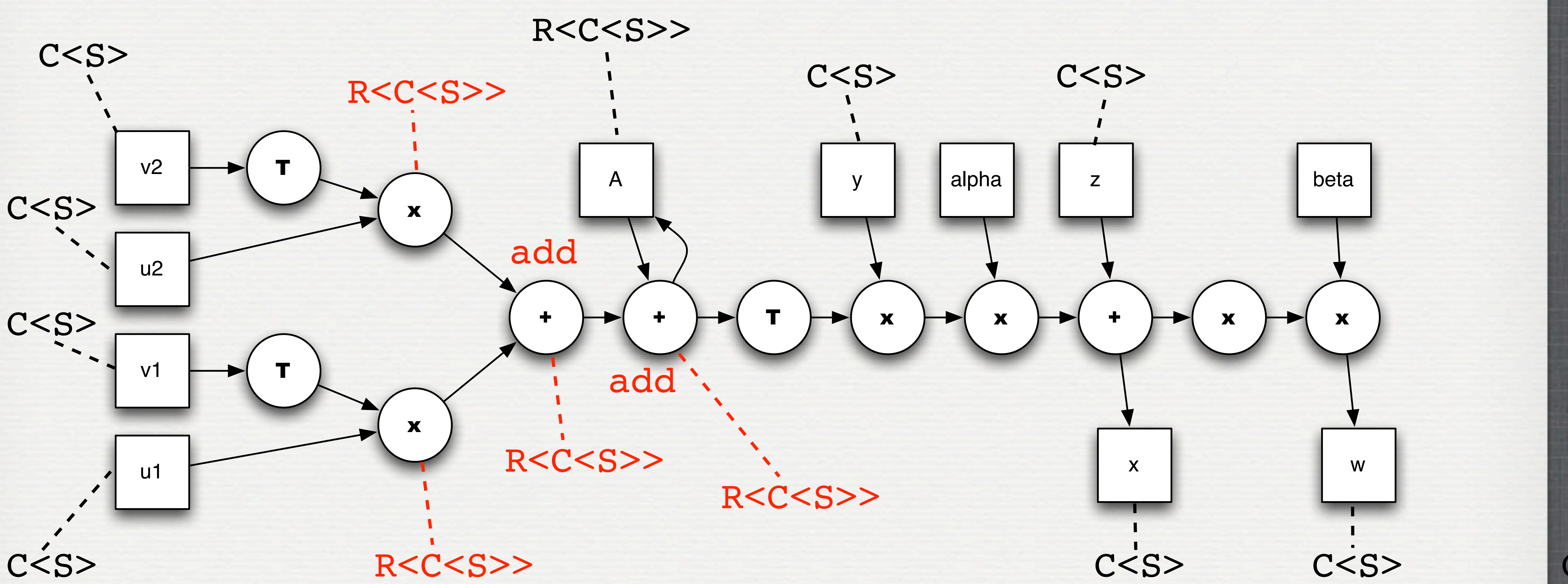
TYPE/ALGO INFERENCE



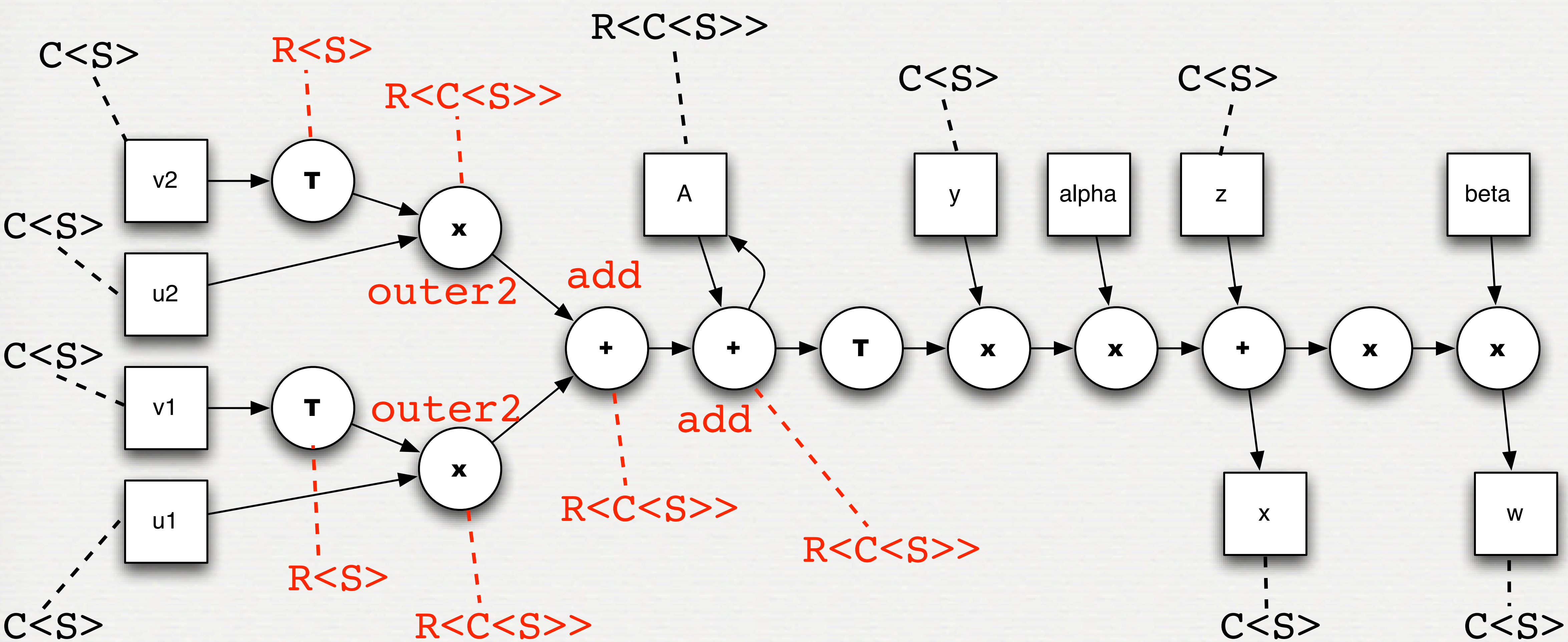
TYPE/ALGO INFERENCE



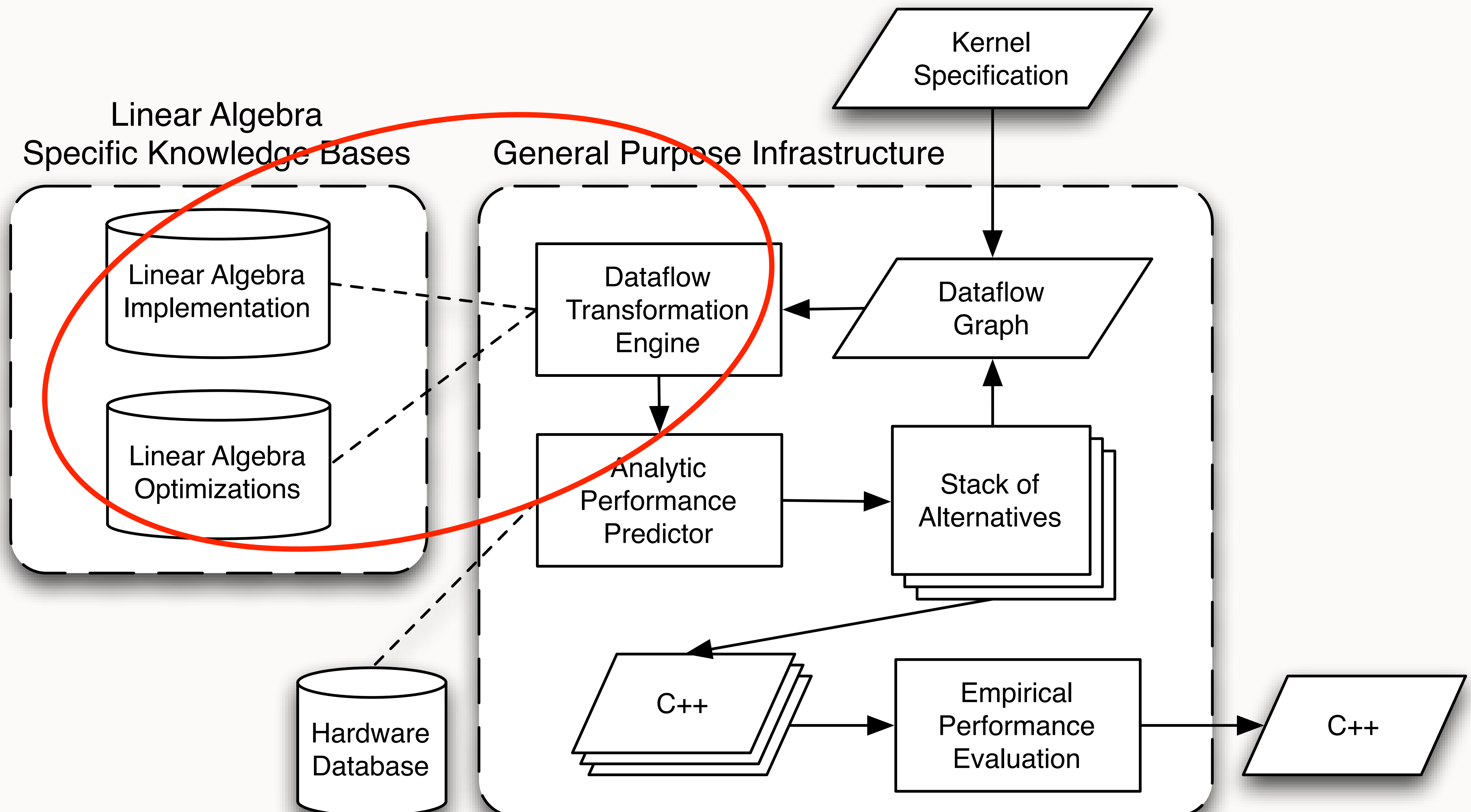
TYPE/ALGO INFERENCE



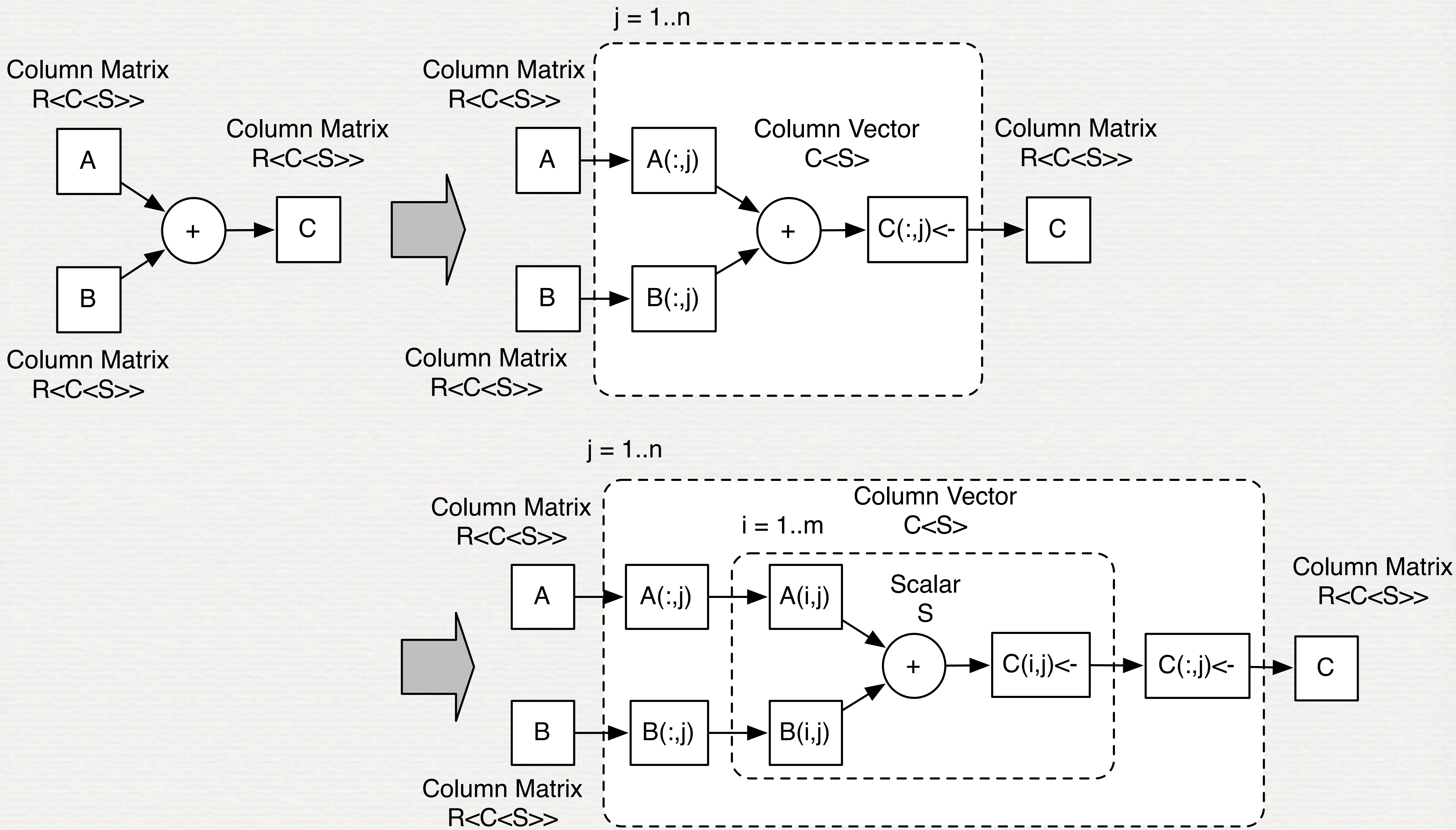
TYPE/ALGO INFERENCE



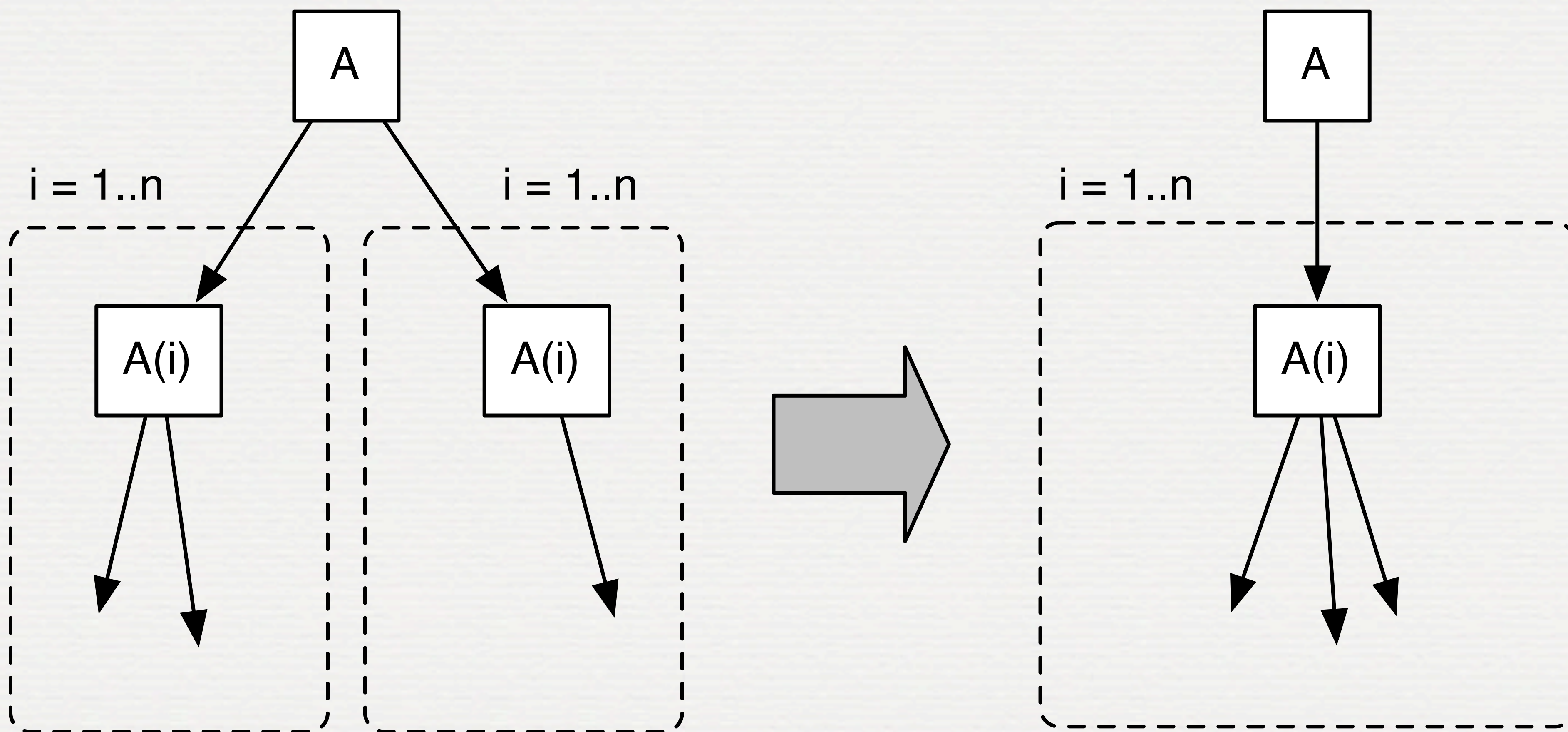
BUILD TO ORDER BLAS



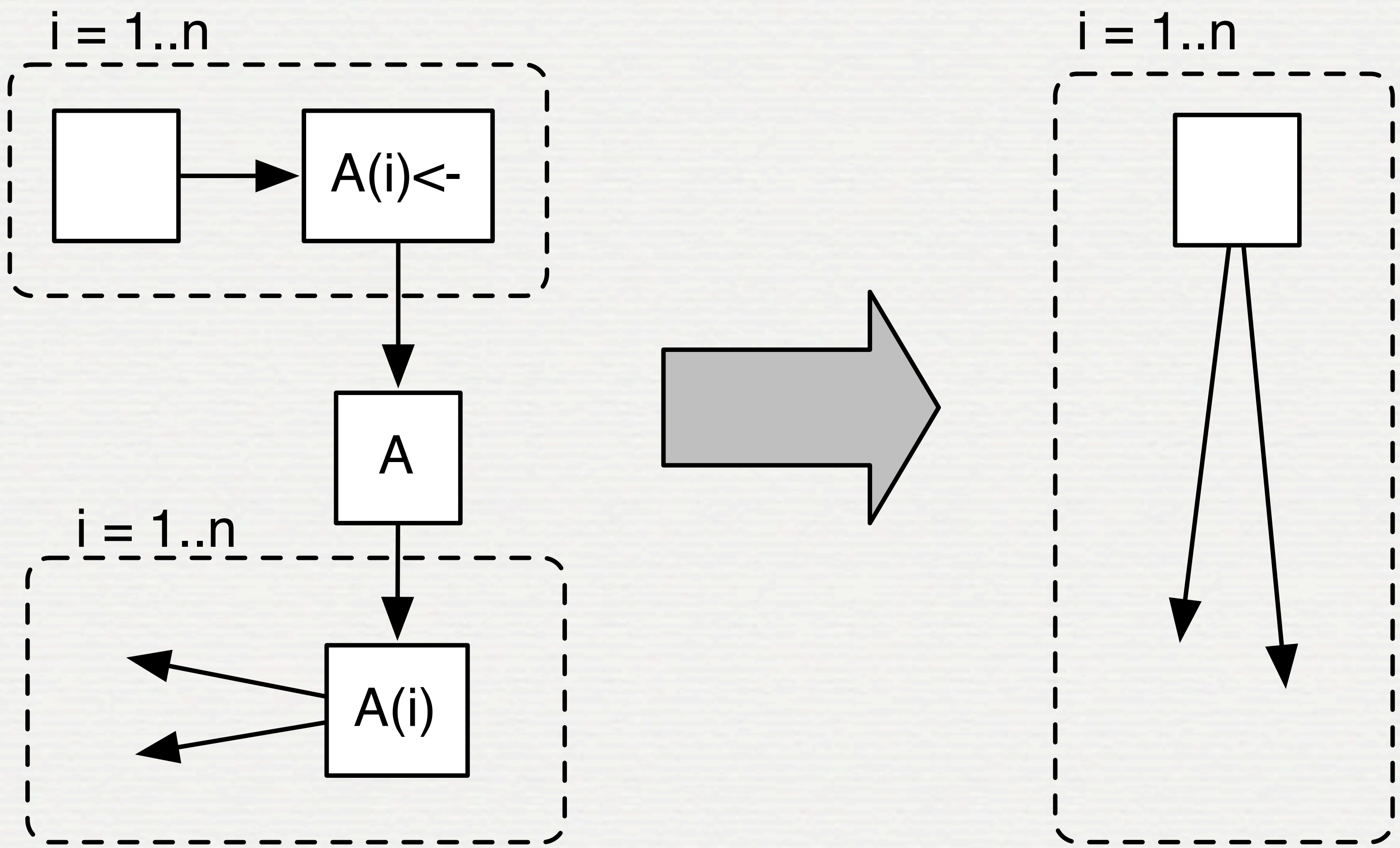
DATAFLOW REFINEMENT



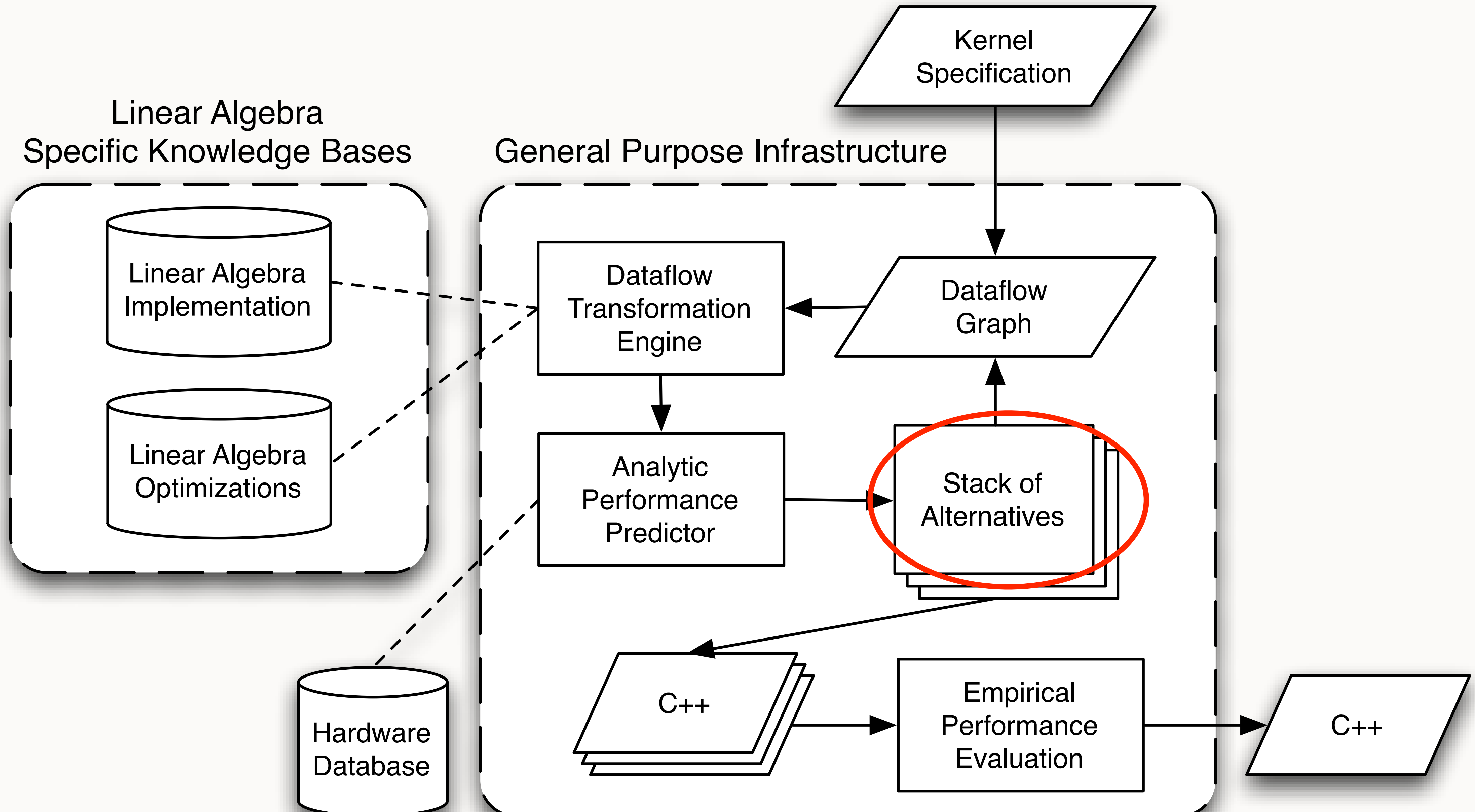
OPTIMIZATION FUSION (1)



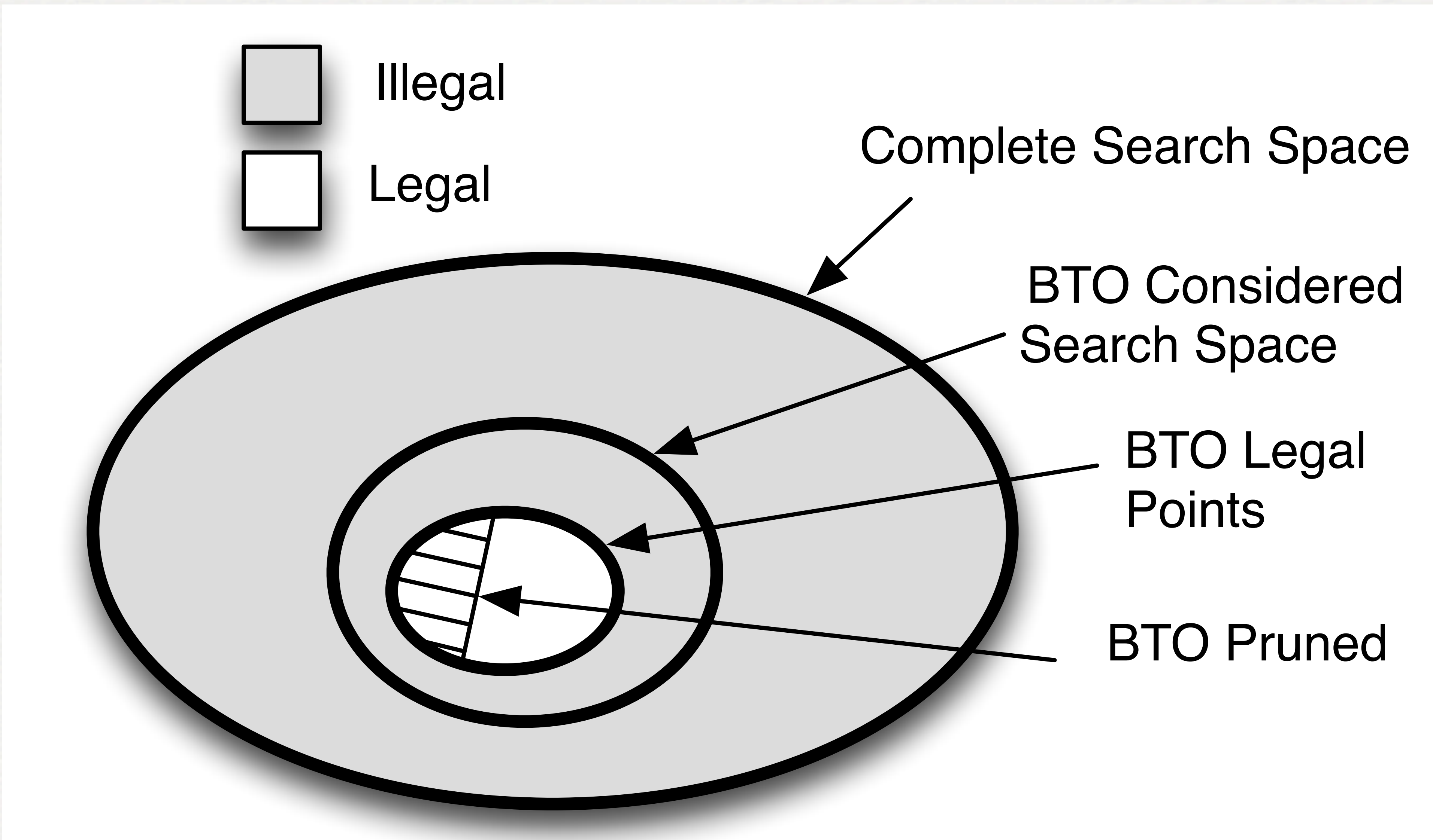
OPTIMIZATION FUSION (2)



BUILD TO ORDER BLAS



SEARCH SPACE



ENUMERATING THE SPACE

- We try to avoid even considering illegal points
- Loop fusion is an equivalence relation
- Can't fuse inner loops if you haven't already fused their outer loops.

$$y \leftarrow \beta A^T A x$$

$$\begin{array}{rc} a: & \{\{1\}\}\{\{2\}\}\{\{3\}\} \\ b: & \{\{1\}\{2\}\}\{\{3\}\} \\ c: & \{\{12\}\}\{\{3\}\} \\ d: & \{\{1\}\{3\}\}\{\{2\}\} \\ e: & \{\{1\}\{2\}\{3\}\} \\ f: & \{\{123\}\} \end{array}$$

$a:$

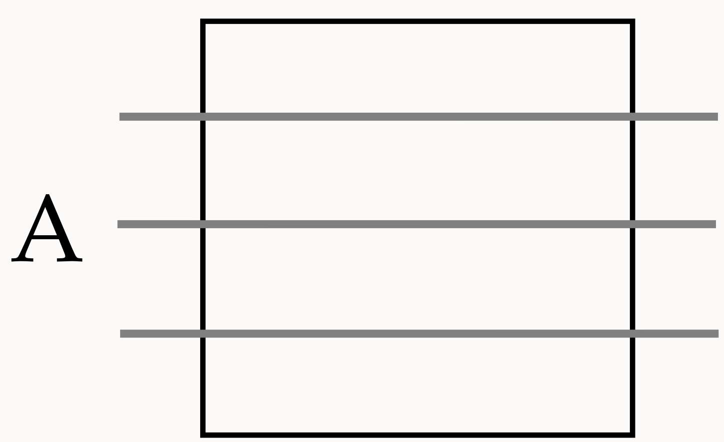
```
for i in 1 to M
    for j in 1 to N
        t0[i] += A[i,j] * x[j]
for i in 1 to M
    for j in 1 to N
        t1[j] += A[i,j] * t0[i]
for j in 1 to N
    y[j] = t1[j] * beta
```

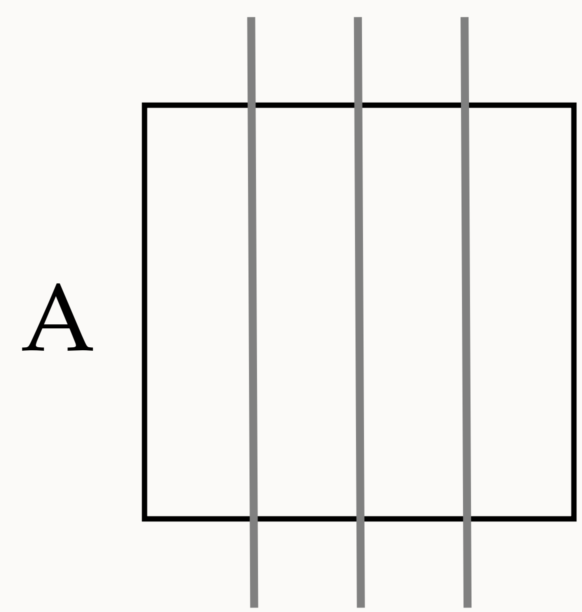
Sunday, May 20, 2012

PARTITIONING

```
for i in 1 to M
    for j in 1 to N
        t0[i] += A[i,j] * x[j]
```

$$\{_i\{_j 1\}\}$$





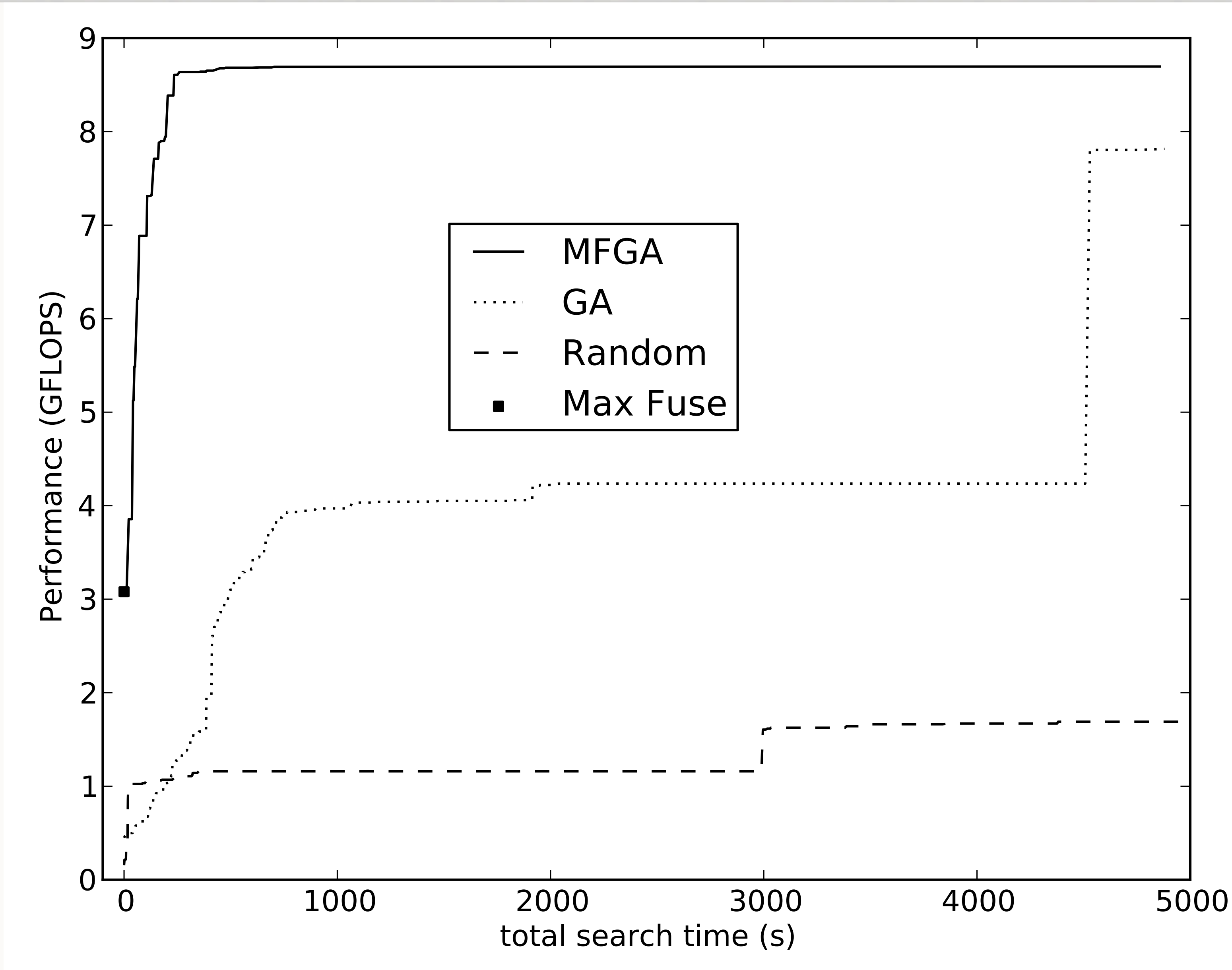
$$\{_{p(i)}\{_i\{_j 1\}\}\}$$

$$\{_{p(j)}\{_i\{_j 1\}\}\}$$

MFGA SEARCH ALGORITHM

- We start with a greedy search technique that we call max-fuse (MF).
- Then we mutate to seed a genetic algorithm (GA).
  - add or remove fusions
  - add or remove partitions
  - change direction of partition (horizontal/vertical)
  - increment/decrement number of threads assigned to a partition

SEARCH TIME VS. PERFORMANCE



For GEMVER on Intel Westmere, 24 core

FUTURE WORK/ CONCLUSIONS

- We obtain reliable, high-performance matrix algebra
  1. high-level specification language
  2. careful enumeration of optimization choices
  3. search algorithm: max-fuse + genetic
- Future work:
  - More parallelism using MPI, GPUs
  - More matrix formats: banded, triangular, **sparse**