Custard and The Sparse Abstract Machine: Compiling Sparse Applications to Coarse-Grained Reconfigurable Arrays

Olivia Hsu

Stanford University

Dataflow hardware can accelerate sparse tensor algebra

Dataflow hardware can accelerate sparse tensor algebra

$a = Bc + a$ $\quad a = Bc$ $\quad A = B + C$

$a = B^Tc + d$ $\quad A = \alpha B$ $\quad a = Bc + b$

$a = B^Tc$ $\quad a = b \odot c$ $\quad a = B(c + d)$

$A = B + C + D$ $\quad A = BC$ $\quad A = B \odot (CD)$

$A = B \odot C$ $\quad A = 0$ $\quad A = BCd$ $\quad A = B^T$ $\quad a = B^TBc$

$a = b + c$ $\quad A = B$ $\quad K = A^TCA$ $\quad a = \alpha Bc + \beta a$

$$A_{ij} = \sum_{kl} B_{ikl}C_{lj}D_{kj} \quad A_{kj} = \sum_{il} B_{ikl}C_{lj}D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl}C_{ij}D_{kj} \quad A_{ij} = \sum_{k} B_{ijk}c_k$$

$$A_{ijk} = \sum_{l} B_{ikl}C_{lj} \quad A_{ik} = \sum_{j} B_{ijk}c_j$$

$$A_{jk} = \sum_{i} B_{ijk}c_i \quad A_{ijl} = \sum_{k} B_{ikl}C_{kj}$$

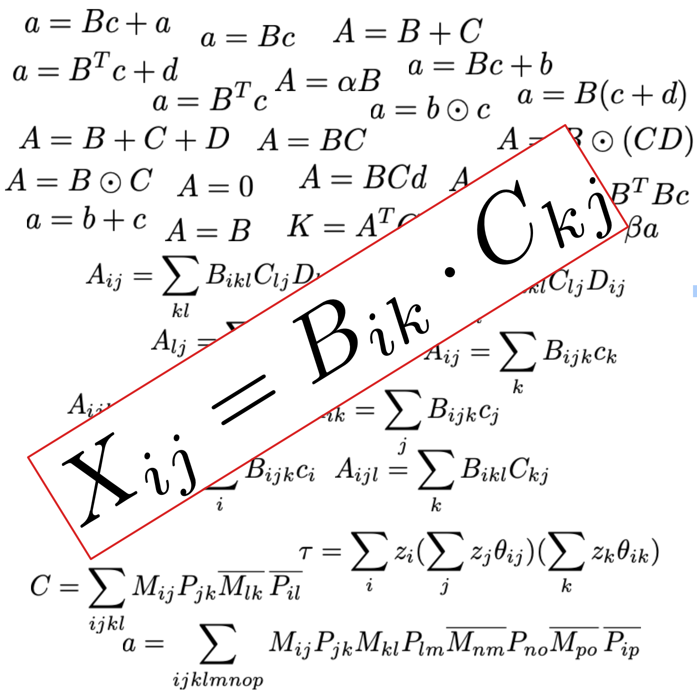
$$\tau = \sum_{i} z_i(\sum_{j} z_j\theta_{ij})(\sum_{k} z_k\theta_{ik})$$

$$C = \sum_{ijkl} M_{ij}P_{jk}\overline{M_{lk}}\,\overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij}P_{jk}M_{kl}P_{lm}\overline{M_{nm}}P_{no}\overline{M_{po}}\,\overline{P_{ip}}$$

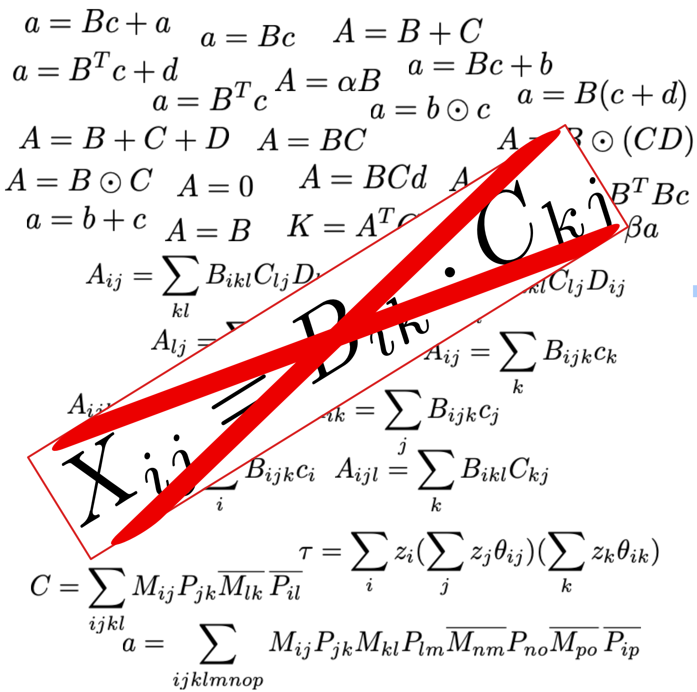
Need Generality to Handle This…

Dataflow hardware can accelerate sparse tensor algebra



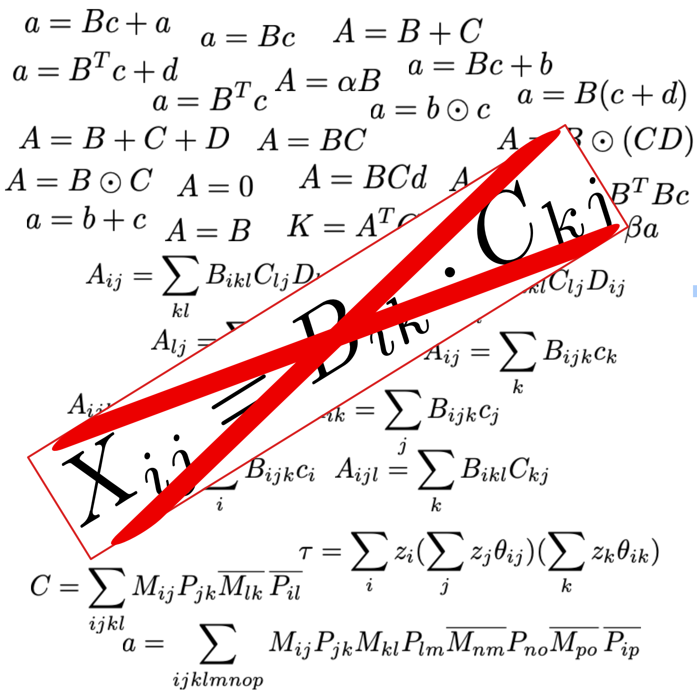
Need Generality to Handle This…

Dataflow hardware can accelerate sparse tensor algebra

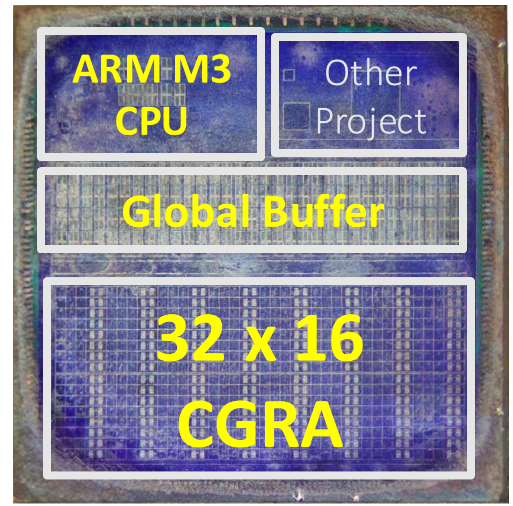


Need Generality to Handle This…

Dataflow hardware can accelerate sparse tensor algebra



Need Generality to Handle This…



Onyx CGRA

[Koul et al. VLSI, HotChips 2024]

but really any sparse accelerator…

Performance requires generality in schedules

Performance requires generality in schedules

Fusion

Dataflow Ordering

Performance requires generality in schedules

$$A = B \odot (CD)$$

SDDMM

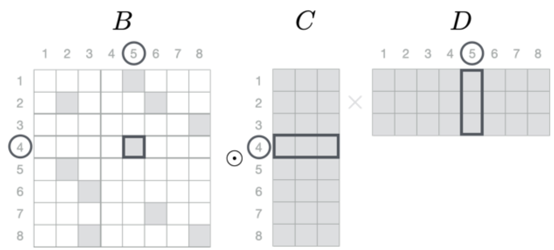
Fusion

Dataflow Ordering

Performance requires generality in schedules

$$A = B \odot (CD)$$

SDDMM



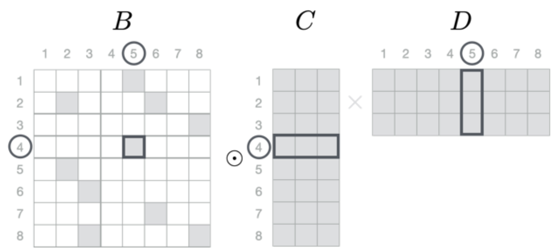
Fusion

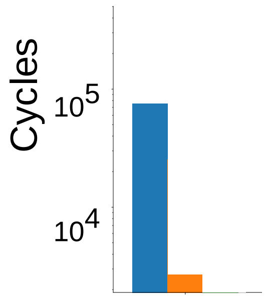
Dataflow Ordering

Performance requires generality in schedules

$$A = B \odot (CD)$$

SDDMM





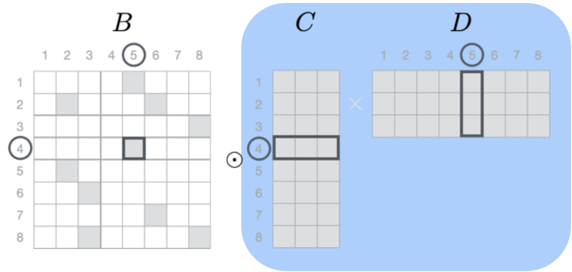
Fusion

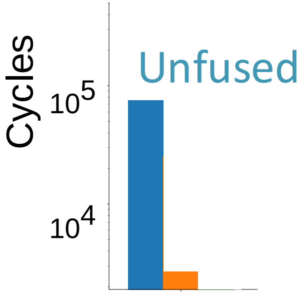
Dataflow Ordering

Performance requires generality in schedules

$$A = B \odot (CD)$$

SDDMM





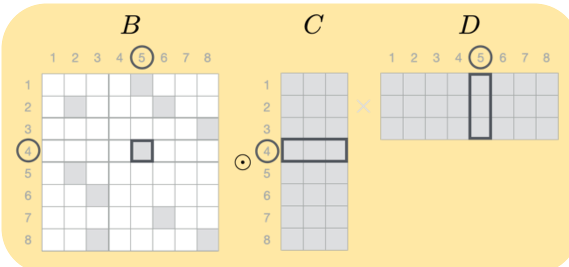
Fusion

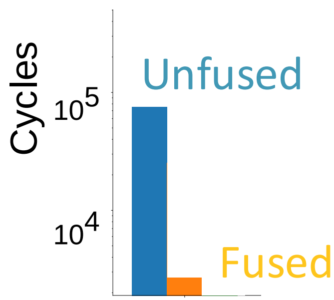
Dataflow Ordering

Performance requires generality in schedules

$$A = B \odot (CD)$$

SDDMM





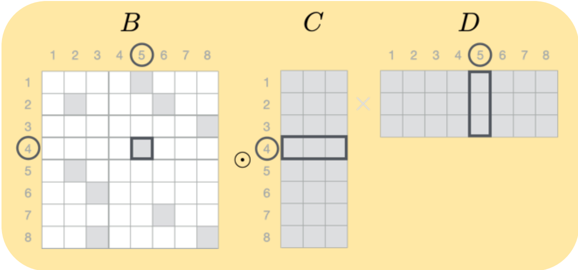
Fusion

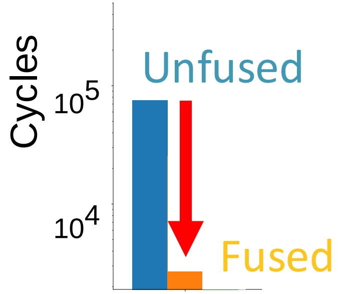
Dataflow Ordering

Performance requires generality in schedules

$$A = B \odot (CD)$$

SDDMM





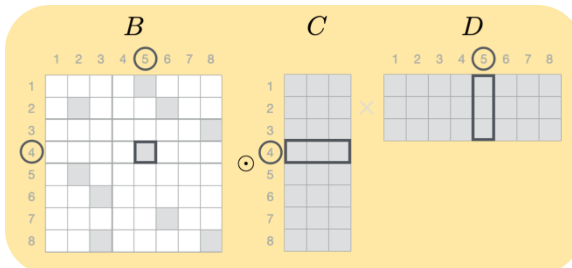
Fusion

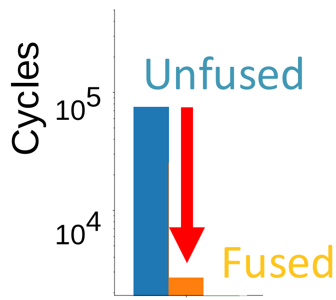
Dataflow Ordering

Performance requires generality in schedules

$$A = B \odot (CD)$$

SDDMM

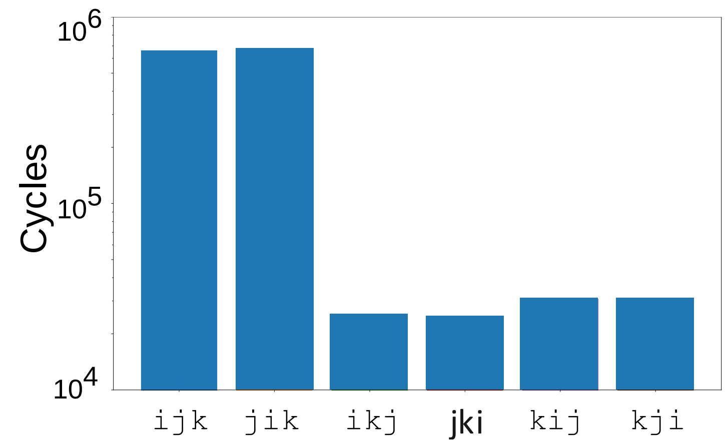




Fusion

$$X_{ij} = B_{ik} \cdot C_{kj}$$

SpMSpM

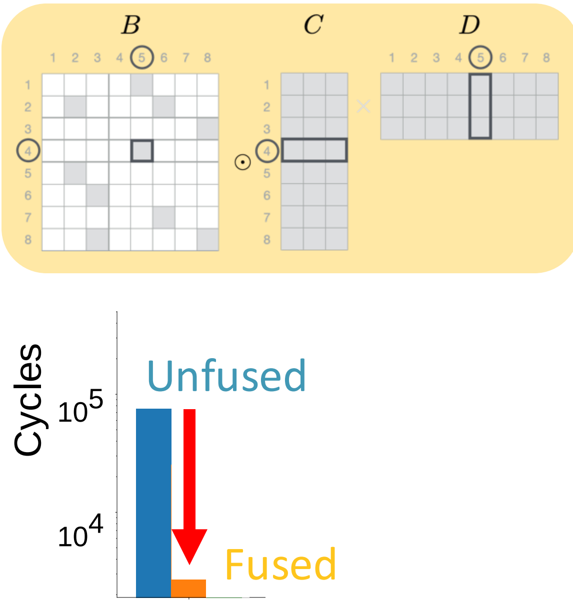


Dataflow Ordering

Performance requires generality in schedules

$$A = B \odot (CD)$$

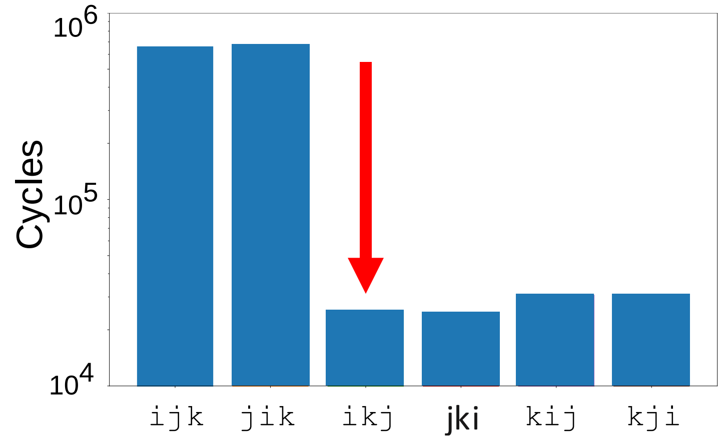
SDDMM



Fusion

$$X_{ij} = B_{ik} \cdot C_{kj}$$

SpMSpM



Dataflow Ordering

Efficient mapping requires a compiler

Efficient mapping requires a compiler

$a = Bc + a$ $a = Bc$ $A = B + C$ Linear Algebra

$a = B^T c + d$ $A = \alpha B$ $a = Bc + b$

$a = B^T c$ $a = b \odot c$ $a = B(c + d)$

$A = B + C + D$ $A = BC$ $A = B \odot (CD)$

$A = B \odot C$ $A = 0$ $A = BCd$ $A = B^T$ $a = B^T Bc$

$a = b + c$ $A = B$ $K = A^T C A$ $a = \alpha Bc + \beta a$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

Data analytics (tensor factorization)

$$\tau = \sum_{i} z_i \left(\sum_{j} z_j \theta_{ij}\right)\left(\sum_{k} z_k \theta_{ik}\right)$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

Quantum Chromodynamics

Algorithm (expression)

Efficient mapping requires a compiler

$a = Bc + a$ $a = Bc$ $A = B + C$ Linear Algebra

$a = B^T c + d$ $A = \alpha B$ $a = Bc + b$

$a = B^T c$ $a = b \odot c$ $a = B(c + d)$

$A = B + C + D$ $A = BC$ $A = B \odot (CD)$

$A = B \odot C$ $A = 0$ $A = BCd$ $A = B^T$ $a = B^T Bc$

$a = b + c$ $A = B$ $K = A^T CA$ $a = \alpha Bc + \beta a$

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj}$ $A_{ij} = \sum_{k} B_{ijk} c_k$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $A_{ik} = \sum_{j} B_{ijk} c_j$

$A_{jk} = \sum_{i} B_{ijk} c_i$ $A_{ijl} = \sum_{k} B_{ikl} C_{kj}$

Data analytics (tensor factorization)

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$ $\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$

$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$

Quantum Chromodynamics

Algorithm (expression)

×

reorder

precompute

parallelize split

map divide

vectorize unroll

position

Schedule

Efficient mapping requires a compiler

$a = Bc + a$ $\quad a = Bc$ $\quad A = B + C$ Linear Algebra

$a = B^T c + d$ $\quad a = B^T c$ $\quad A = \alpha B$ $\quad a = Bc + b$ $\quad a = b \odot c$ $\quad a = B(c + d)$

$A = B + C + D$ $\quad A = BC$ $\quad A = B \odot (CD)$

$A = B \odot C$ $\quad A = 0$ $\quad A = BCd$ $\quad A = B^T$ $\quad a = B^T Bc$

$a = b + c$ $\quad A = B$ $\quad K = A^T C A$ $\quad a = \alpha Bc + \beta a$

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

Data analytics (tensor factorization)

$$\tau = \sum_i z_i \left(\sum_j z_j \theta_{ij}\right)\left(\sum_k z_k \theta_{ik}\right)$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

Quantum Chromodynamics

Algorithm (expression)

$\times$

reorder

precompute

parallelize split

map divide

vectorize unroll

position

Schedule

→

Efficient mapping requires a compiler

Linear Algebra

$a = Bc + a$ $a = Bc$ $A = B + C$ $a = B^T c + d$ $A = \alpha B$ $a = Bc + b$ $a = B^T c$ $a = b \odot c$ $a = B(c + d)$ $A = B + C + D$ $A = BC$ $A = B \odot (CD)$ $A = B \odot C$ $A = 0$ $A = BCd$ $A = B^T$ $a = B^T Bc$ $a = b + c$ $A = B$ $K = A^T C A$ $a = \alpha Bc + \beta a$

Data analytics (tensor factorization)

$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj}$ $A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$

$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj}$ $A_{ij} = \sum_{k} B_{ijk} c_k$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $A_{ik} = \sum_{j} B_{ijk} c_j$

$A_{jk} = \sum_{i} B_{ijk} c_i$ $A_{ijl} = \sum_{k} B_{ikl} C_{kj}$

Quantum Chromodynamics

$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$

$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$

Algorithm (expression)

$\times$

reorder

precompute

parallelize split

map divide

vectorize unroll

position

Schedule

→ Onyx

Efficient mapping requires a compiler

Linear Algebra

$a = Bc + a$ $\quad a = Bc$ $\quad A = B + C$ $\quad a = B^T c + d$ $\quad A = \alpha B$ $\quad a = Bc + b$ $\quad a = B^T c$ $\quad a = b \odot c$ $\quad a = B(c + d)$

$A = B + C + D$ $\quad A = BC$ $\quad A = B \odot (CD)$ $\quad A = B \odot C$ $\quad A = 0$ $\quad A = BCd$ $\quad A = B^T$ $\quad a = B^T Bc$

$a = b + c$ $\quad A = B$ $\quad K = A^T C A$ $\quad a = \alpha Bc + \beta a$

Data analytics (tensor factorization)

$$A_{ij} = \sum_{kl} B_{ikl} C_{lj} D_{kj} \qquad A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$$

$$A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj} \qquad A_{ij} = \sum_{k} B_{ijk} c_k$$

$$A_{ijk} = \sum_{l} B_{ikl} C_{lj} \qquad A_{ik} = \sum_{j} B_{ijk} c_j$$

$$A_{jk} = \sum_{i} B_{ijk} c_i \qquad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$$

Quantum Chromodynamics

$$\tau = \sum_{i} z_i \left(\sum_{j} z_j \theta_{ij}\right)\left(\sum_{k} z_k \theta_{ik}\right)$$

$$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$$

$$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$$

Algorithm (expression)

×

reorder

precompute

parallelize split

map divide

vectorize unroll

position

Schedule

→

EIE

SpArch OuterSPACE

Gamma MatRaptor

Plasticine

Sigma

Tensaurus

Onyx Sparse TPU

SCNN

Eyeriss V2

SpaceA

UCNN

Spada

Fifer Sparse CGRAs

SPU

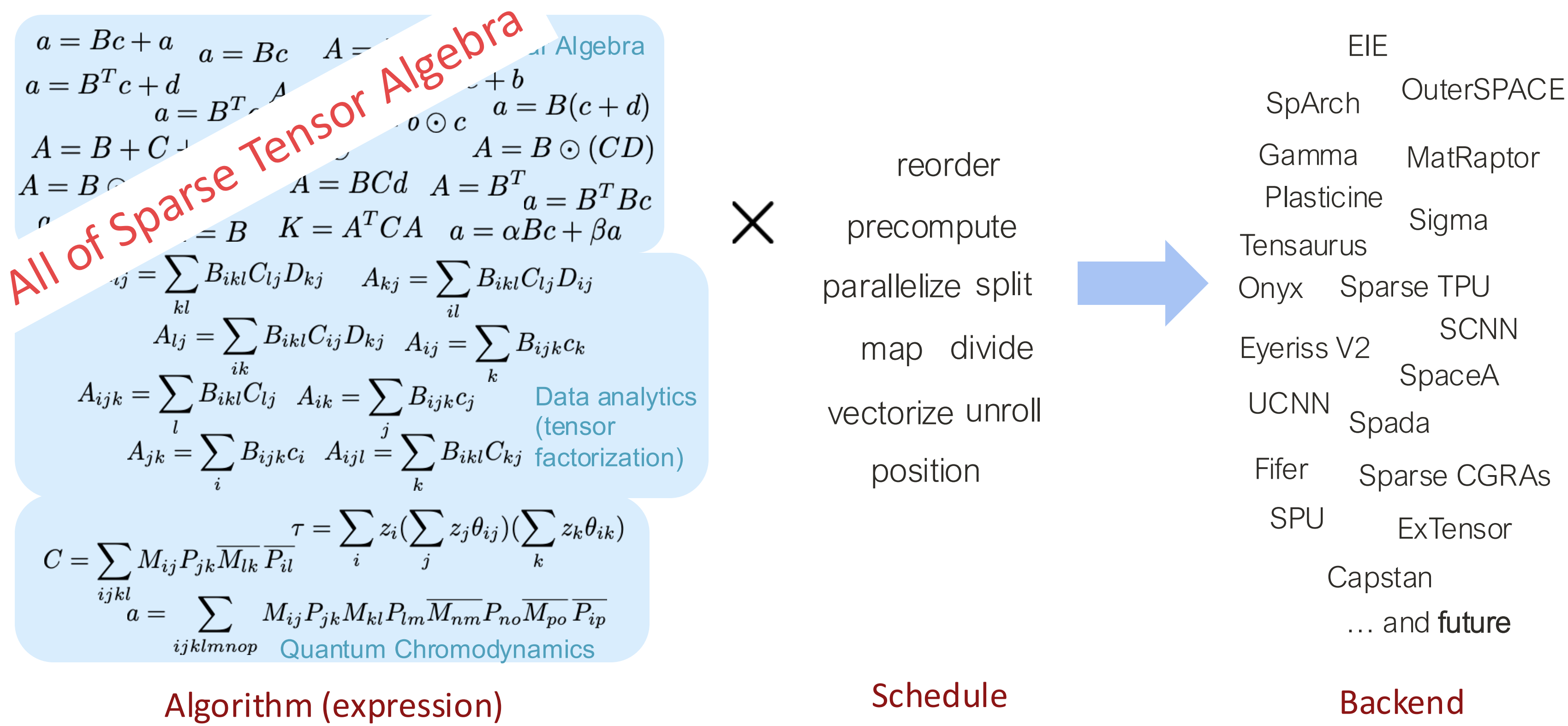
ExTensor

Capstan

… and future

Backend

Efficient mapping requires a compiler



Efficient mapping requires a compiler

All of Sparse Tensor Algebra

$a = Bc + a$ $\quad a = Bc$ $\quad a = B^T c + d$ $\quad a = B(c + d)$ $\quad A = B \odot (CD)$ $\quad A = BCd$ $\quad a = B^T Bc$ $\quad K = A^T C A$ $\quad a = \alpha Bc + \beta a$

…ar Algebra

$A_{kj} = \sum_{il} B_{ikl} C_{lj} D_{ij}$ $\quad A_{lj} = \sum_{ik} B_{ikl} C_{ij} D_{kj}$ $\quad A_{ij} = \sum_{k} B_{ijk} c_k$

$A_{ijk} = \sum_{l} B_{ikl} C_{lj}$ $\quad A_{ik} = \sum_{j} B_{ijk} c_j$

$A_{jk} = \sum_{i} B_{ijk} c_i$ $\quad A_{ijl} = \sum_{k} B_{ikl} C_{kj}$

Data analytics (tensor factorization)

$\tau = \sum_{i} z_i (\sum_{j} z_j \theta_{ij})(\sum_{k} z_k \theta_{ik})$

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$

$a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$

Quantum Chromodynamics

Algorithm (expression)

×

reorder
precompute
parallelize split
map divide
vectorize unroll
position

Schedule

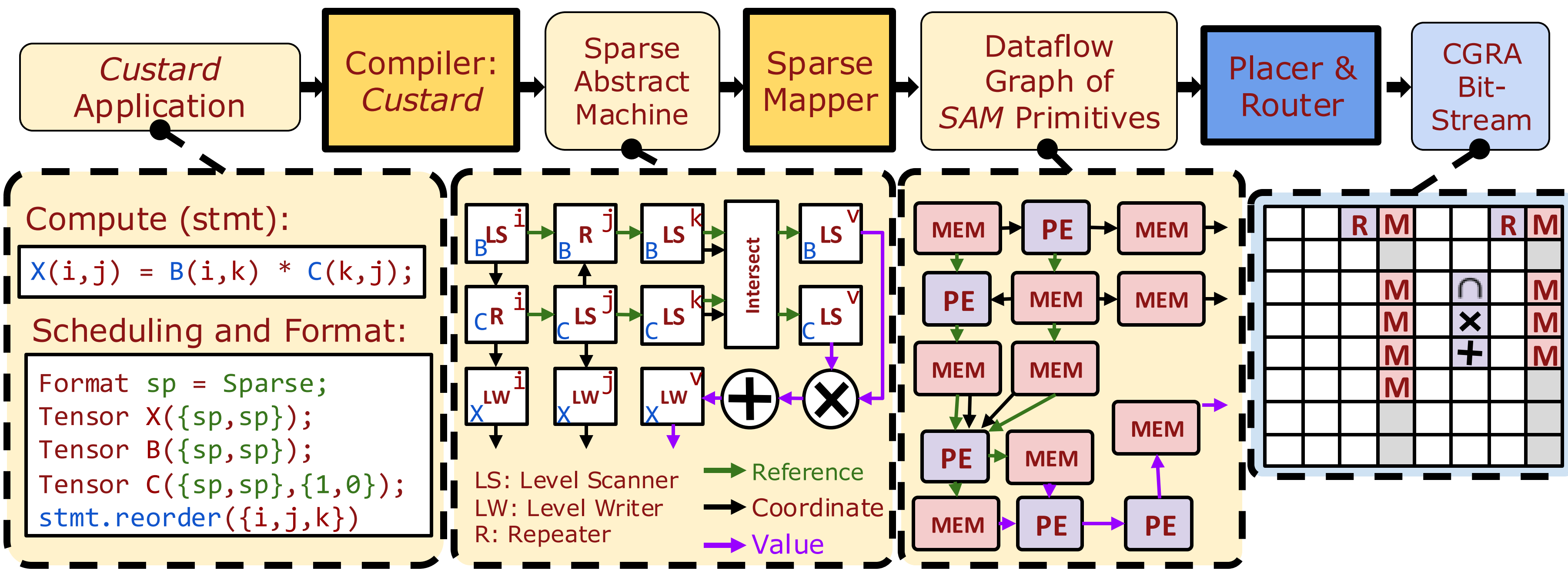
→

Lends itself to multiple backends

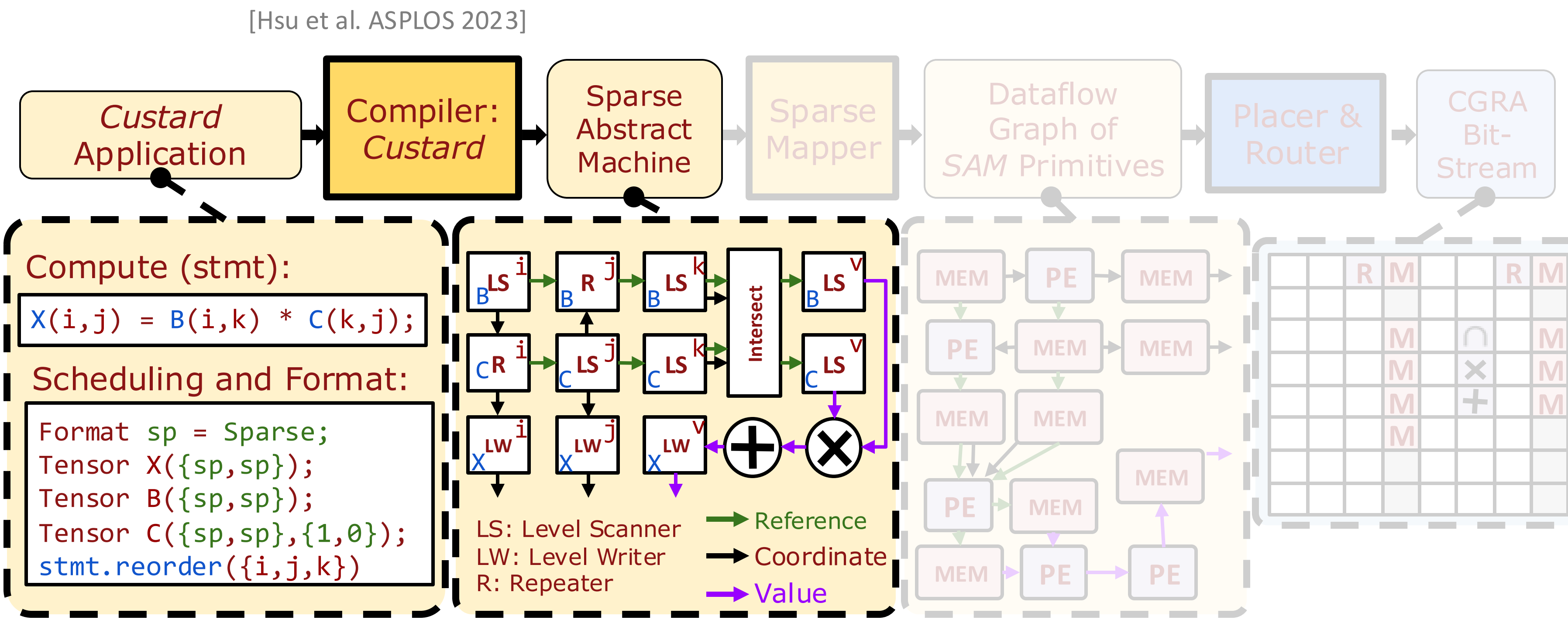
…CE
MatRaptor
…cine
Sigma
Tensaurus
Onyx
Sparse TPU
SCNN
Eyeriss V2
SpaceA
UCNN
Spada
Fifer
Sparse CGRAs
SPU
ExTensor
Capstan
… and future

Backend

This work in the DSL-based CGRA flow

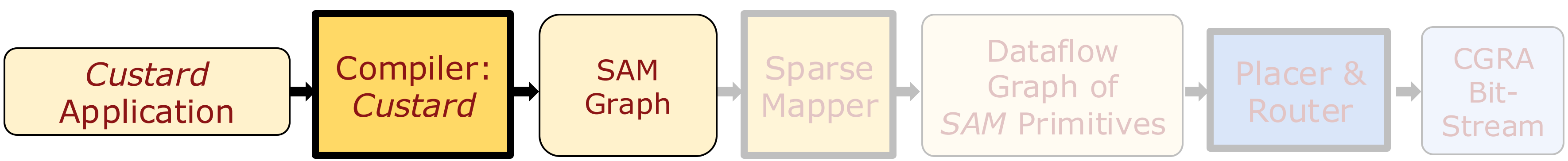


This work in the DSL-based CGRA flow



SAM leverages domain-specific languages and comes with the Custard compiler

Custard
Application

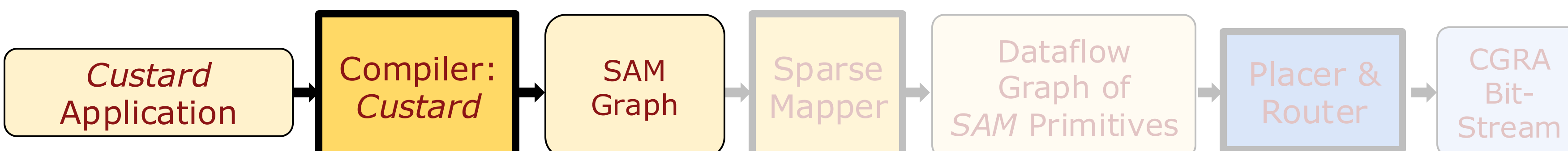


SAM leverages domain-specific languages and comes with the Custard compiler

Custard Application

```
X(i,j) = B(i,k) * C(k,j);
```

Tensor Index Notation



SAM leverages domain-specific languages and comes with the Custard compiler

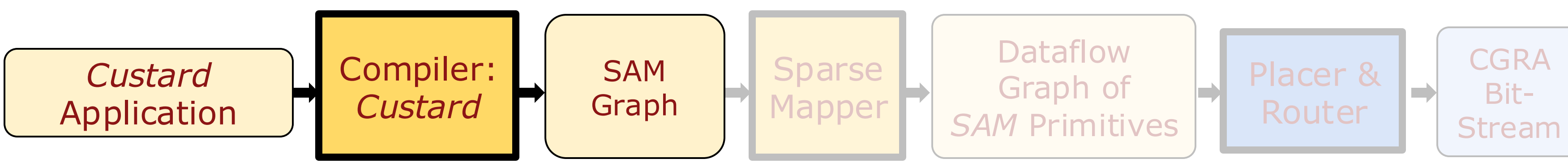
Custard Application

```
Format sp = Sparse;
Tensor X({sp,sp});
Tensor B({sp,sp});
Tensor C({sp,sp},{1,0});
```

```
X(i,j) = B(i,k) * C(k,j);
```

Formats

Tensor Index Notation



SAM leverages domain-specific languages and comes with the Custard compiler

Custard Application

```
Format sp = Sparse;
Tensor X({sp,sp});
Tensor B({sp,sp});
Tensor C({sp,sp},{1,0});
```

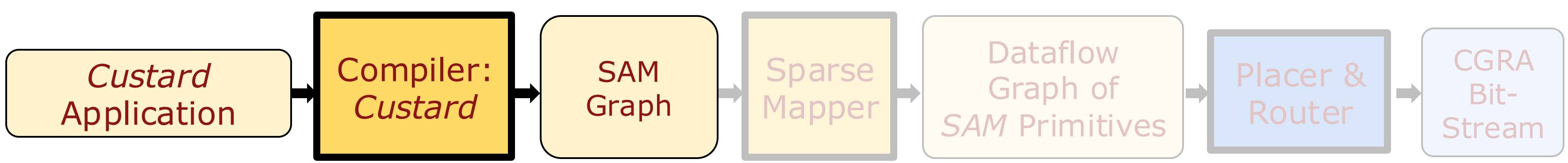
```
X(i,j) = B(i,k) * C(k,j);
```

```
stmt.reorder({i,j,k});
```

Formats

Tensor Index Notation

Schedule



SAM leverages domain-specific languages and comes with the Custard compiler

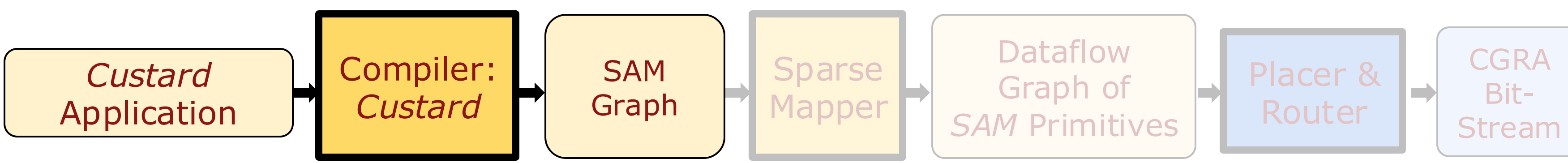
Custard Application

```
Format sp = Sparse;
Tensor X({sp,sp});
Tensor B({sp,sp});
Tensor C({sp,sp},{1,0});
```

```
X(i,j) = B(i,k) * C(k,j);
```

```
stmt.reorder({i,j,k});
```

High-Level DSLs	Formats	Tensor Index Notation	Schedule



SAM leverages domain-specific languages and comes with the Custard compiler

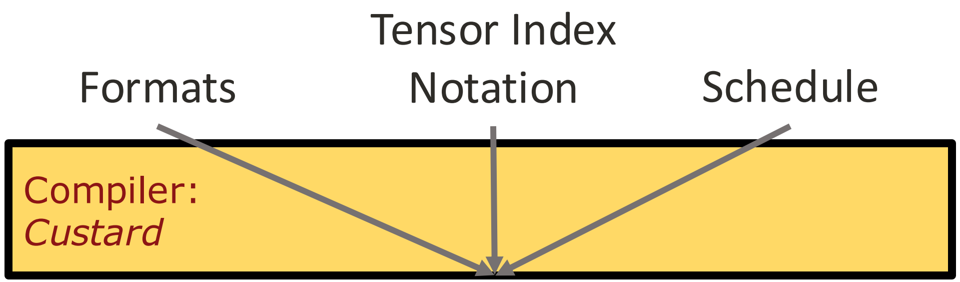
Custard Application

```
Format sp = Sparse;
Tensor X({sp,sp});
Tensor B({sp,sp});
Tensor C({sp,sp},{1,0});
```

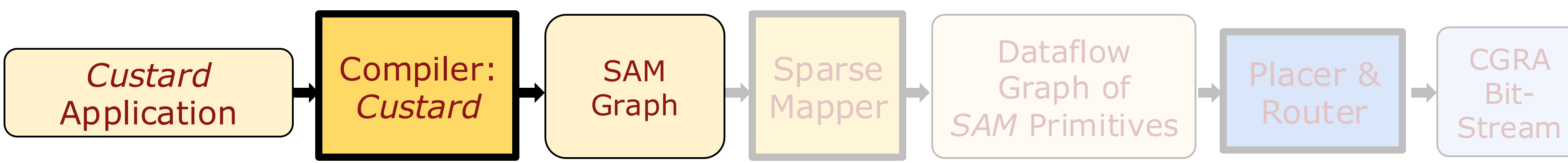
```
X(i,j) = B(i,k) * C(k,j);
```

```
stmt.reorder({i,j,k});
```

High-Level DSLs



The Sparse Abstract Machine



Representing dataflow in SAM

SAM Graph

SAM represents:

1. Wires carrying data through streams
2. Modules that compute on the data through primitives

Representing dataflow in SAM

SAM Graph

SAM represents:

1. Wires carrying data through streams
2. Modules that compute on the data through primitives

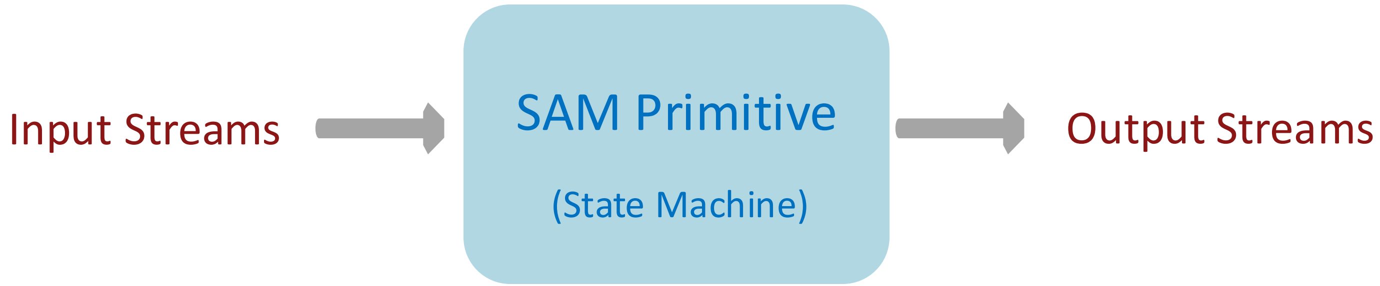
Input Streams → → Output Streams

Representing dataflow in SAM

SAM Graph

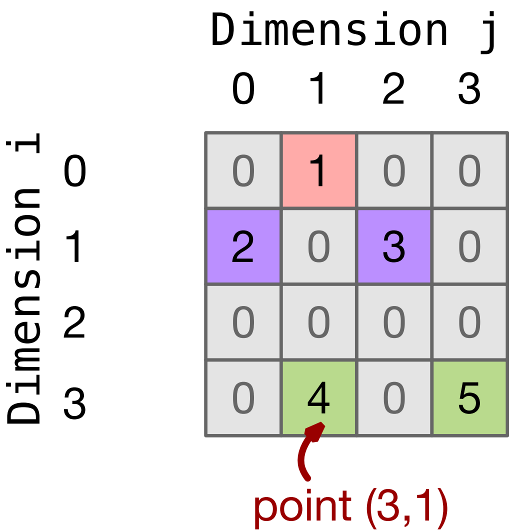
SAM represents:

1. Wires carrying data through streams
2. Modules that compute on the data through primitives



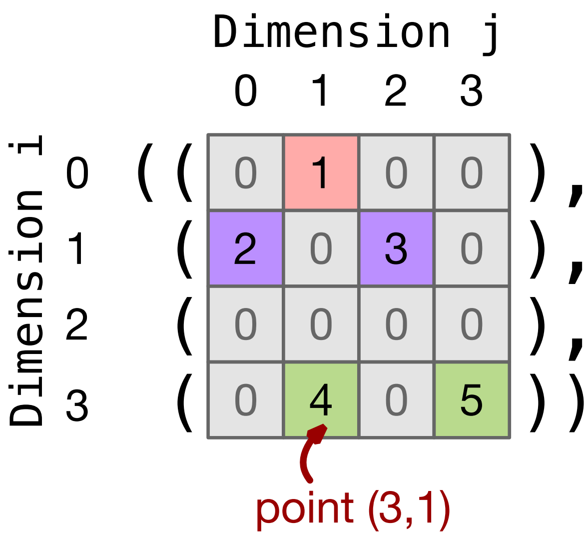
Representing tensors in SAM

SAM Graph



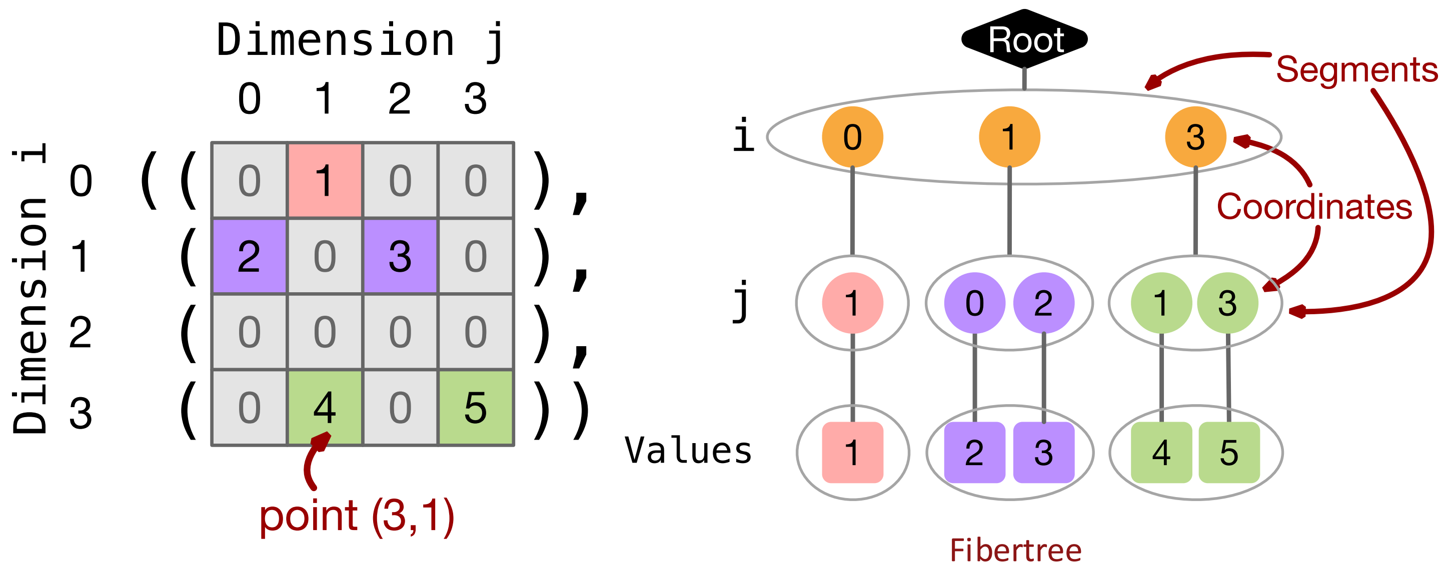
Representing tensors in SAM

SAM Graph



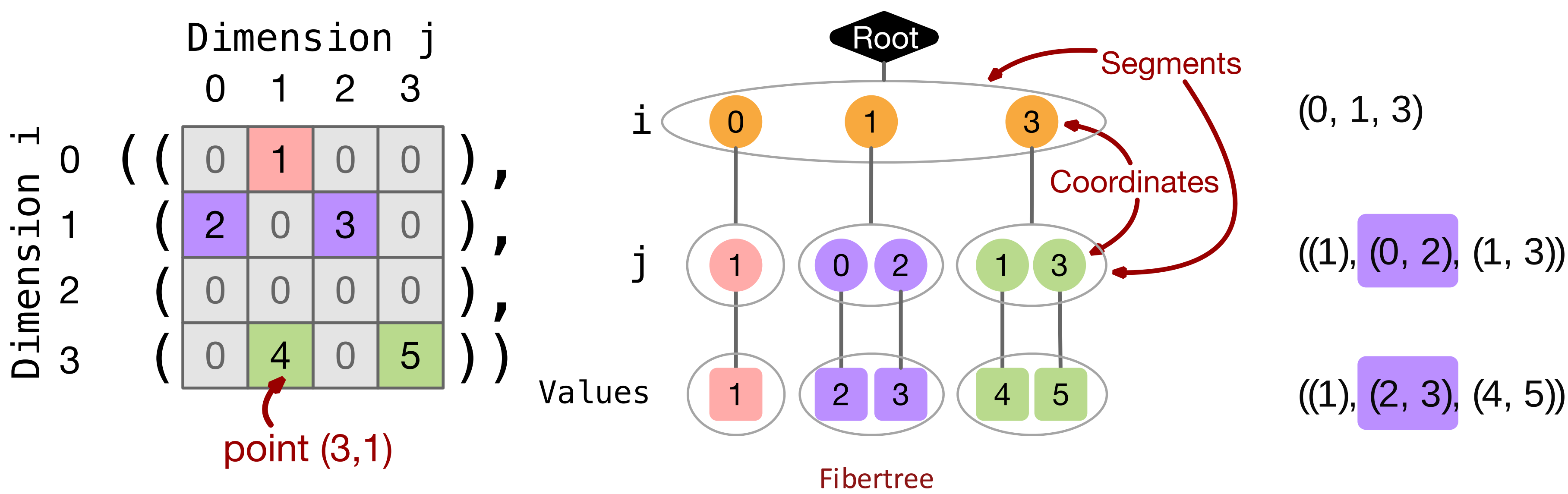
Representing tensors in SAM

SAM Graph



Representing tensors in SAM

SAM Graph



As data structures and flattened streams

SAM Graph

(0, 1, 3)

((1), (0, 2), (1, 3))

((1), (2, 3), (4, 5))

As data structures and flattened streams

SAM Graph

(0, 1, 3)

((1), (0, 2), (1, 3))

((1), (2, 3), (4, 5))

Arrays (Space)

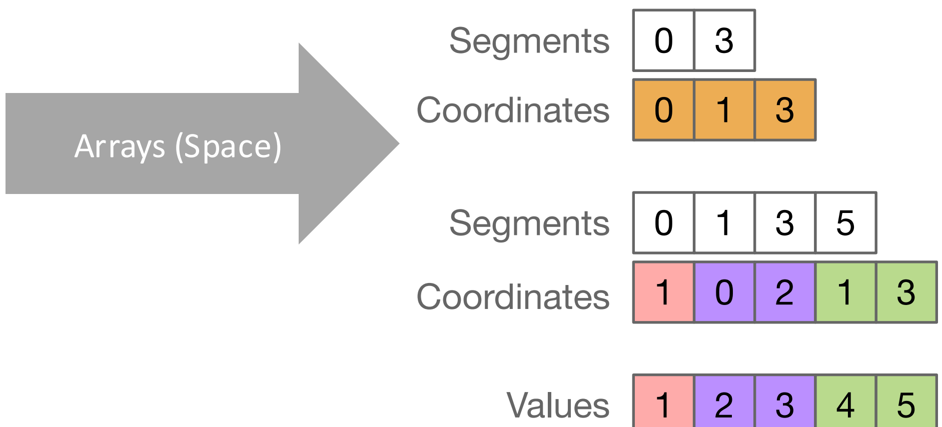
As data structures and flattened streams

SAM Graph

(0, 1, 3)

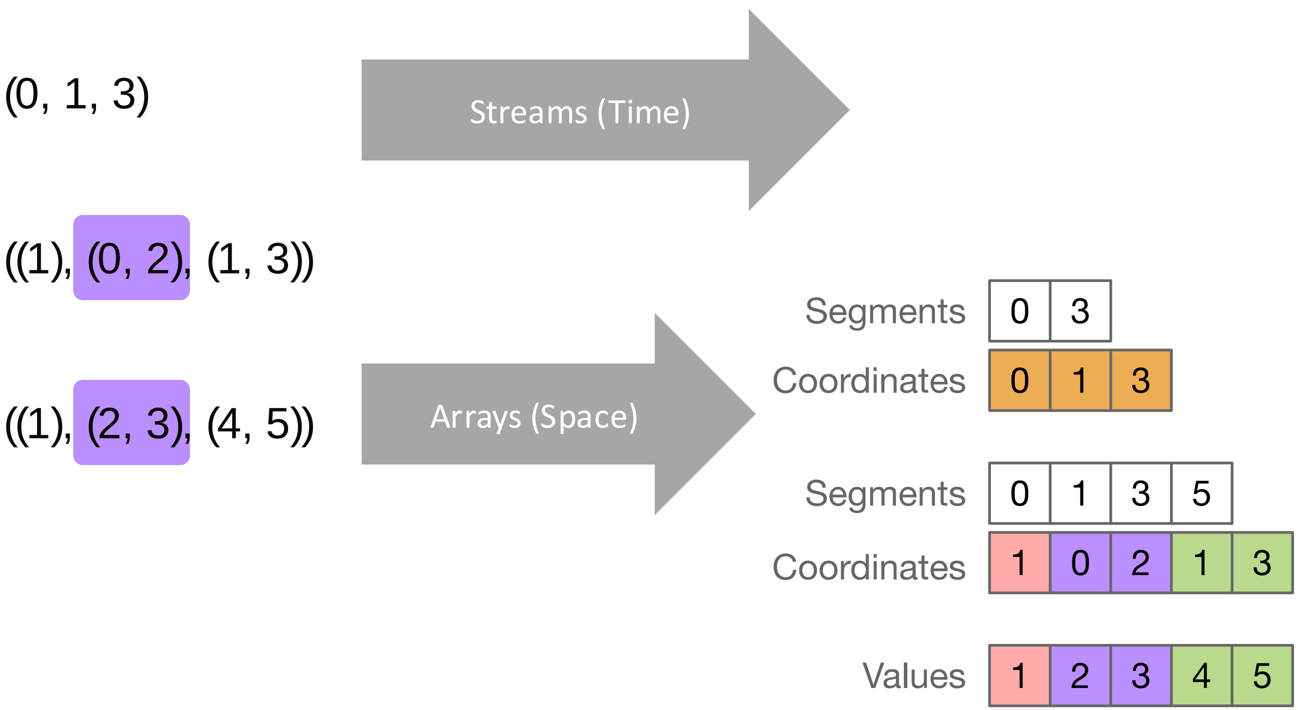
((1), (0, 2), (1, 3))

((1), (2, 3), (4, 5))



As data structures and flattened streams

SAM Graph



As data structures and flattened streams

SAM Graph

$D, S_0, 3, 1, 0$

(0, 1, 3)

((1), (0, 2), (1, 3))

((1), (2, 3), (4, 5))

Streams (Time)

Arrays (Space)

Segments	0	3	
Coordinates	0	1	3

Segments	0	1	3	5	
Coordinates	1	0	2	1	3
Values	1	2	3	4	5

As data structures and flattened streams

SAM Graph

$D, S_0, 3, 1, 0$

$D, S_1, 3, 1, S_0, 2, 0, S_0, 1$

(0, 1, 3)

Streams (Time)

((1), (0, 2), (1, 3))

((1), (2, 3), (4, 5))

Arrays (Space)

Segments	0	3

Coordinates	0	1	3

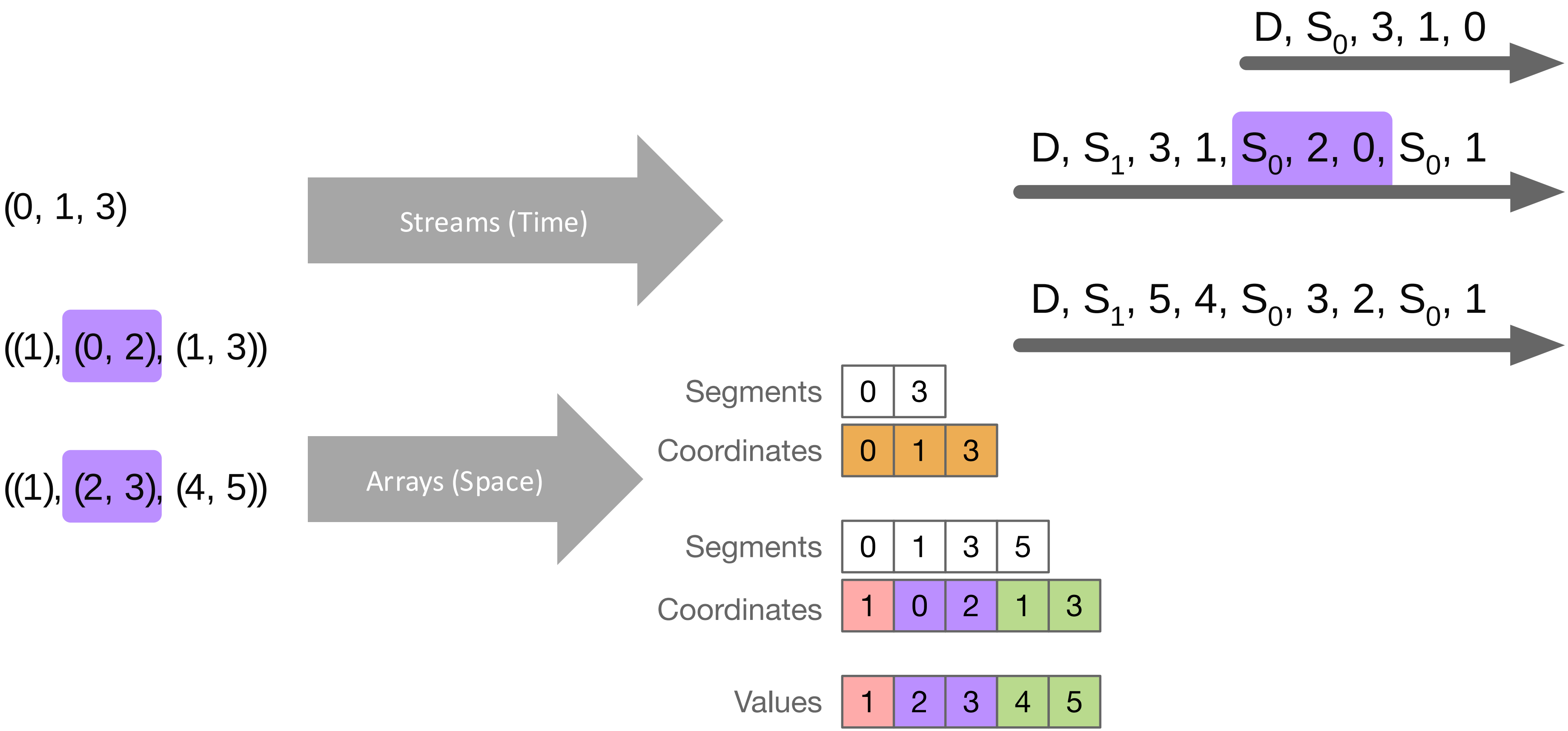
Segments	0	1	3	5

Coordinates	1	0	2	1	3

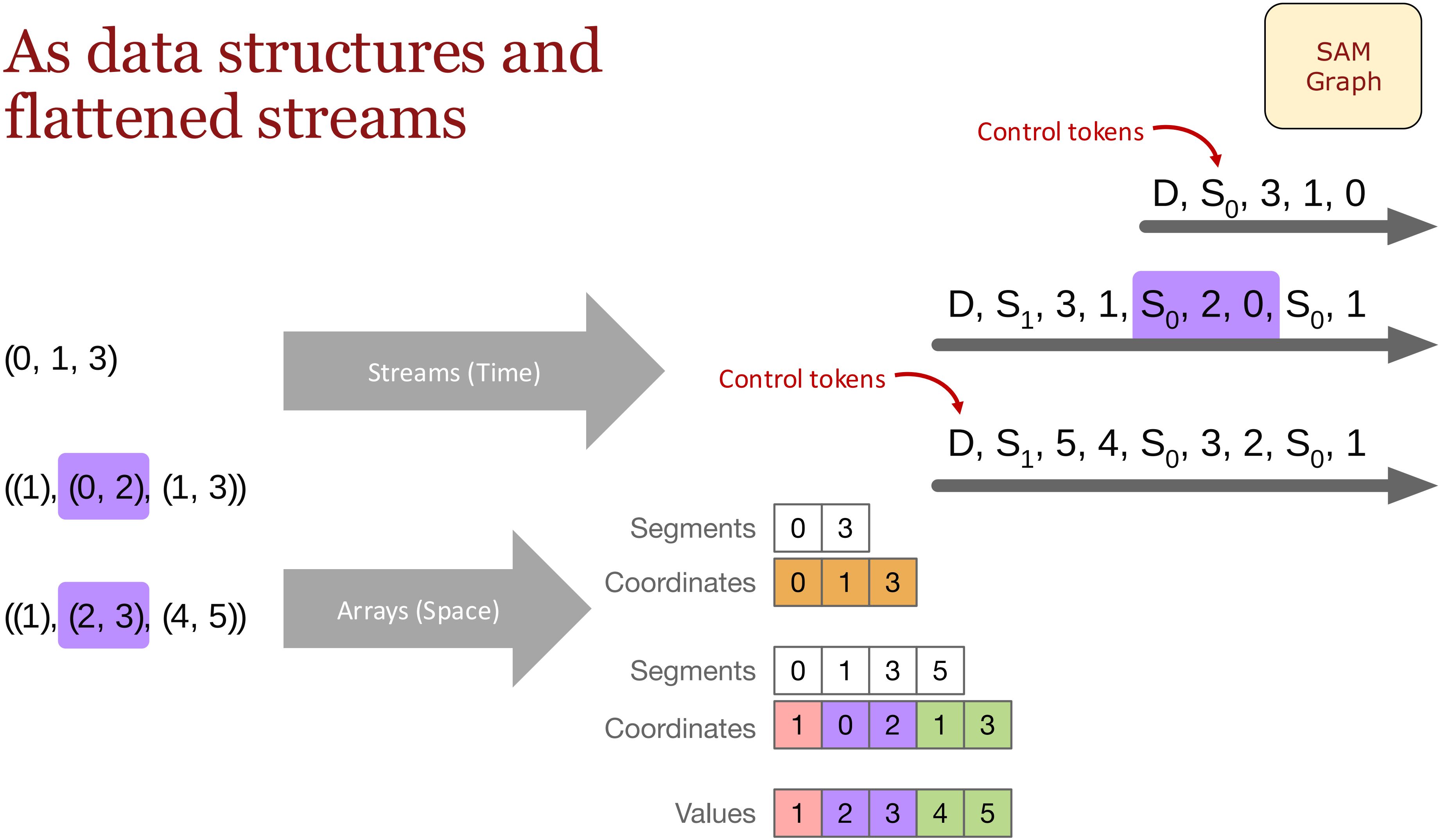
Values	1	2	3	4	5

As data structures and flattened streams

SAM Graph



As data structures and flattened streams



SAM supports all of tensor algebra

SAM
Graph

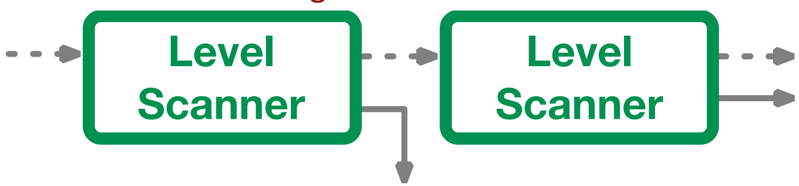
The sparse abstract machine has clean interfaces defined for each feature of sparse tensor algebra, called primitives

SAM supports all of tensor algebra

SAM Graph

The sparse abstract machine has clean interfaces defined for each feature of sparse tensor algebra, called primitives

1. Iterate through multidimensional tensors

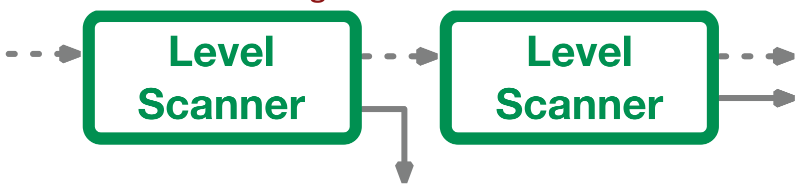


SAM supports all of tensor algebra

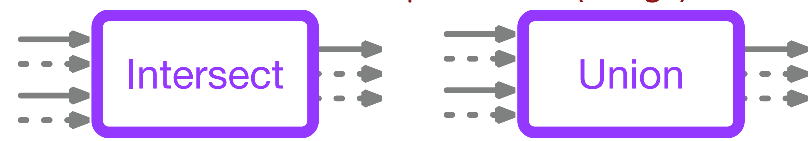
SAM Graph

The sparse abstract machine has clean interfaces defined for each feature of sparse tensor algebra, called primitives

1. Iterate through multidimensional tensors



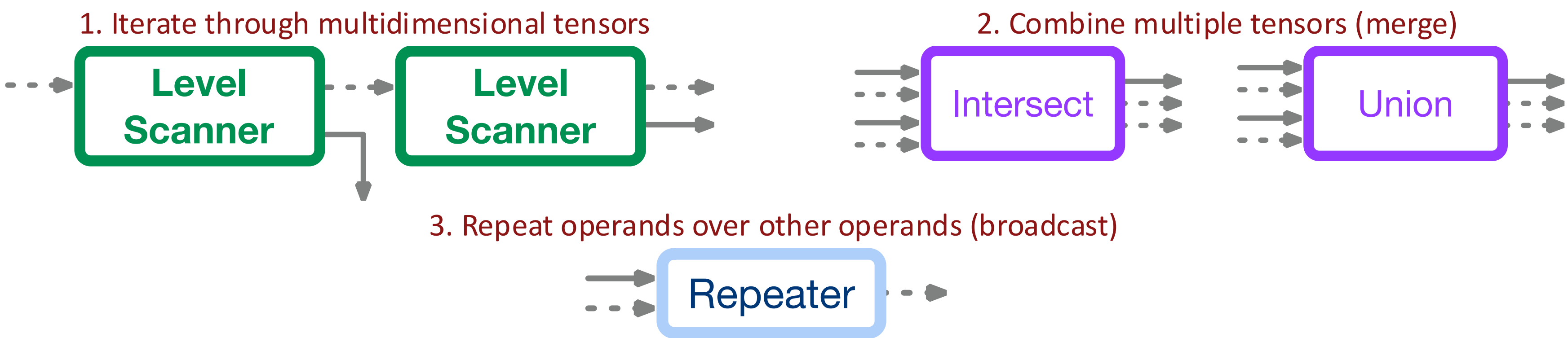
2. Combine multiple tensors (merge)



SAM supports all of tensor algebra

SAM Graph

The sparse abstract machine has clean interfaces defined for each feature of sparse tensor algebra, called primitives

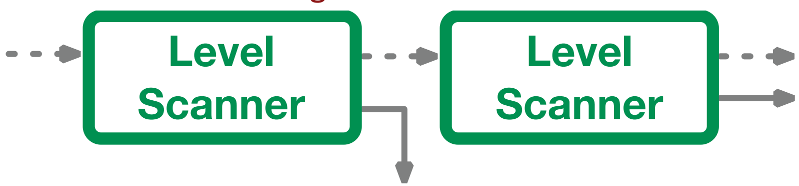


SAM supports all of tensor algebra

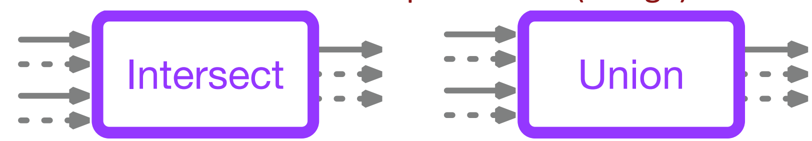
SAM Graph

The sparse abstract machine has clean interfaces defined for each feature of sparse tensor algebra, called primitives

1. Iterate through multidimensional tensors

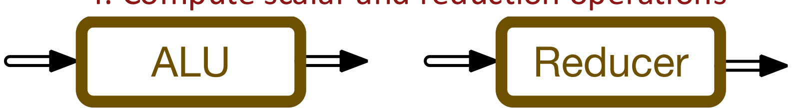


2. Combine multiple tensors (merge)



3. Repeat operands over other operands (broadcast)

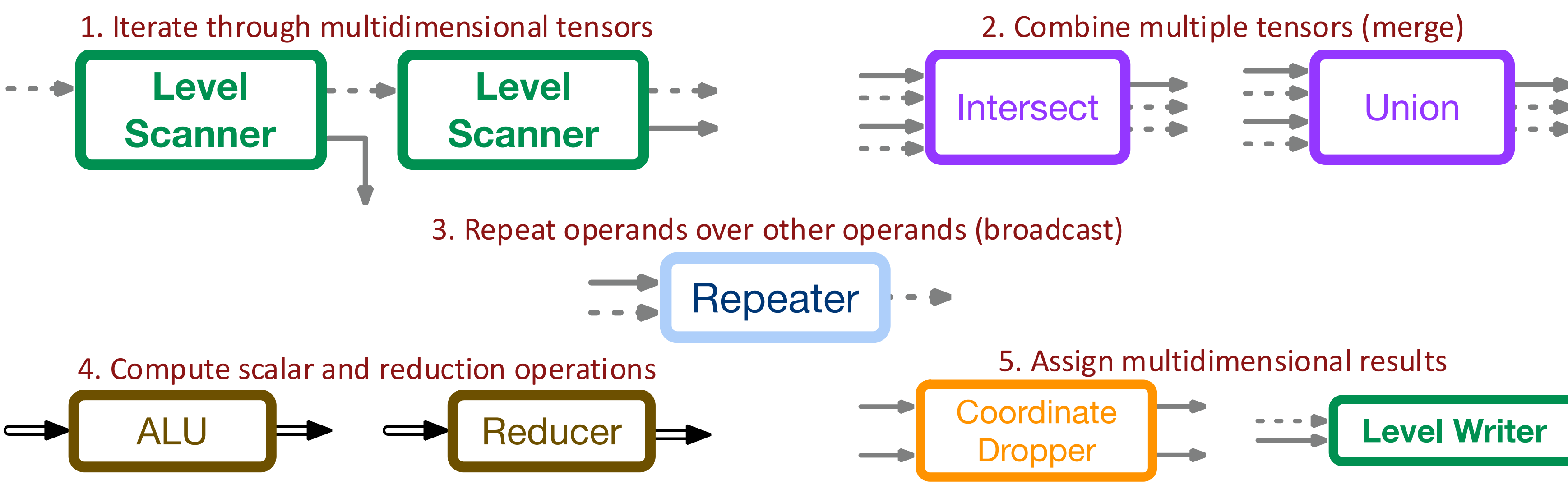
4. Compute scalar and reduction operations



SAM supports all of tensor algebra

SAM Graph

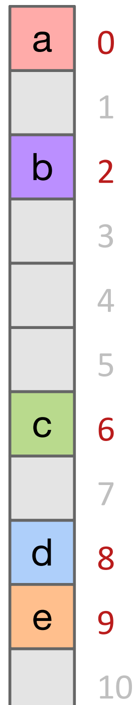
The sparse abstract machine has clean interfaces defined for each feature of sparse tensor algebra, called primitives



Vector scaling example with repeaters

SAM Graph

$$x_i = \underline{b_i} \cdot c$$

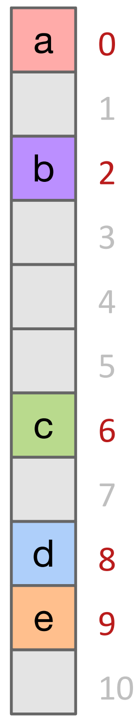


b_i

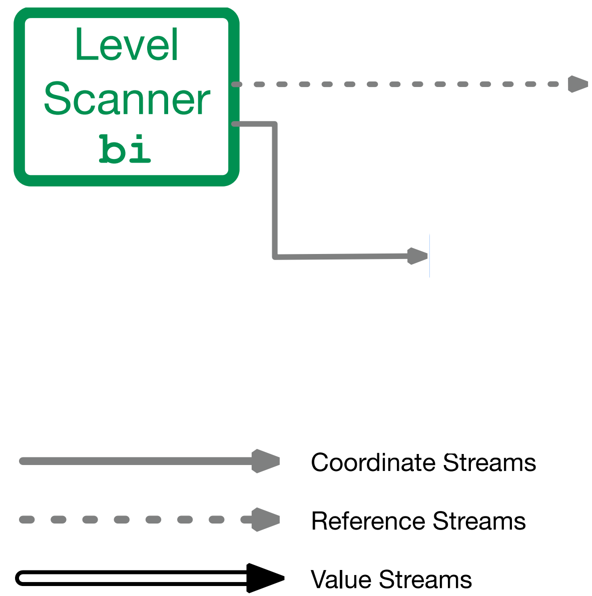
Vector scaling example with repeaters

SAM Graph

$$x_i = \underline{b_i} \cdot c$$



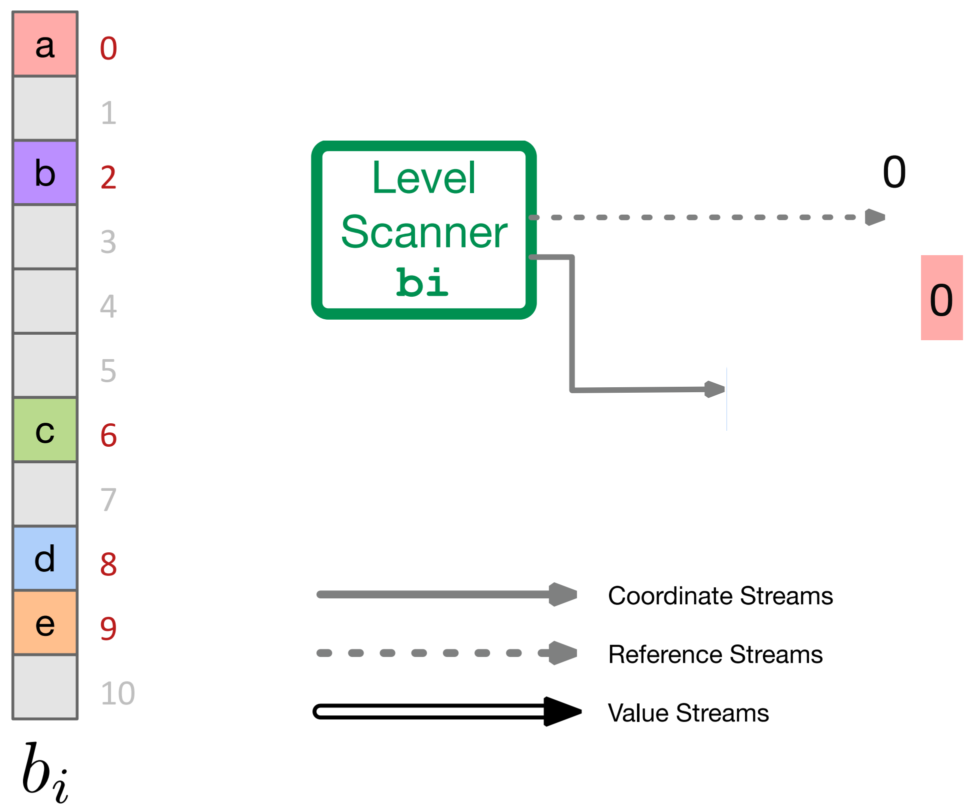
b_i



Vector scaling example with repeaters

SAM Graph

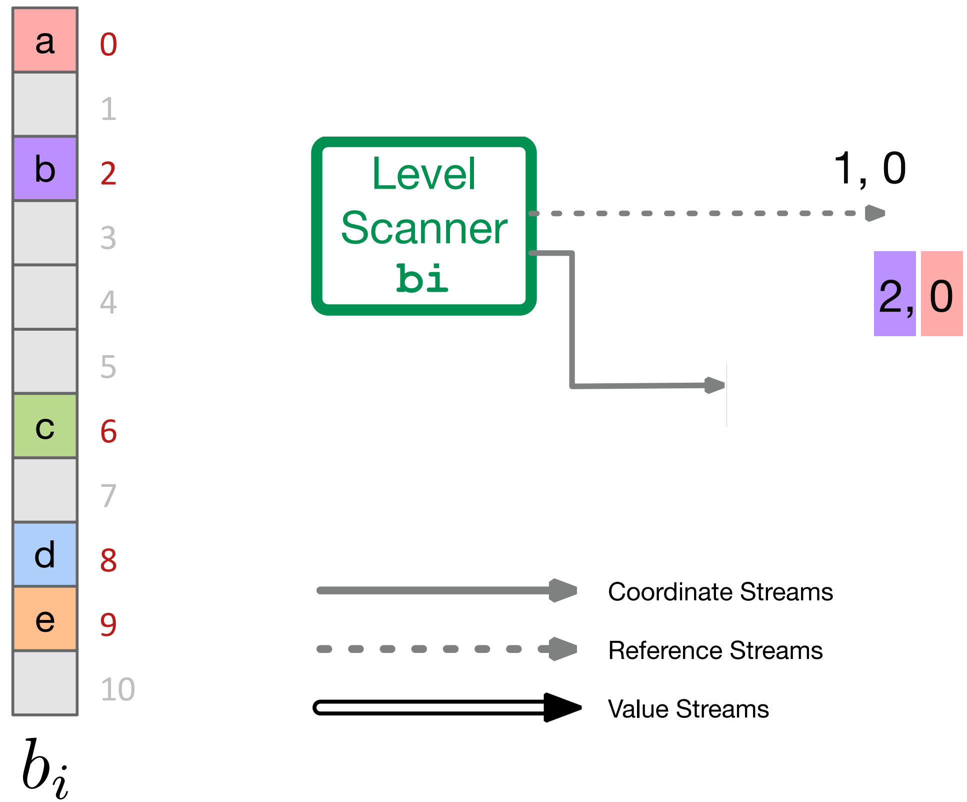
$$x_i = \underline{b_i} \cdot c$$



Vector scaling example with repeaters

SAM Graph

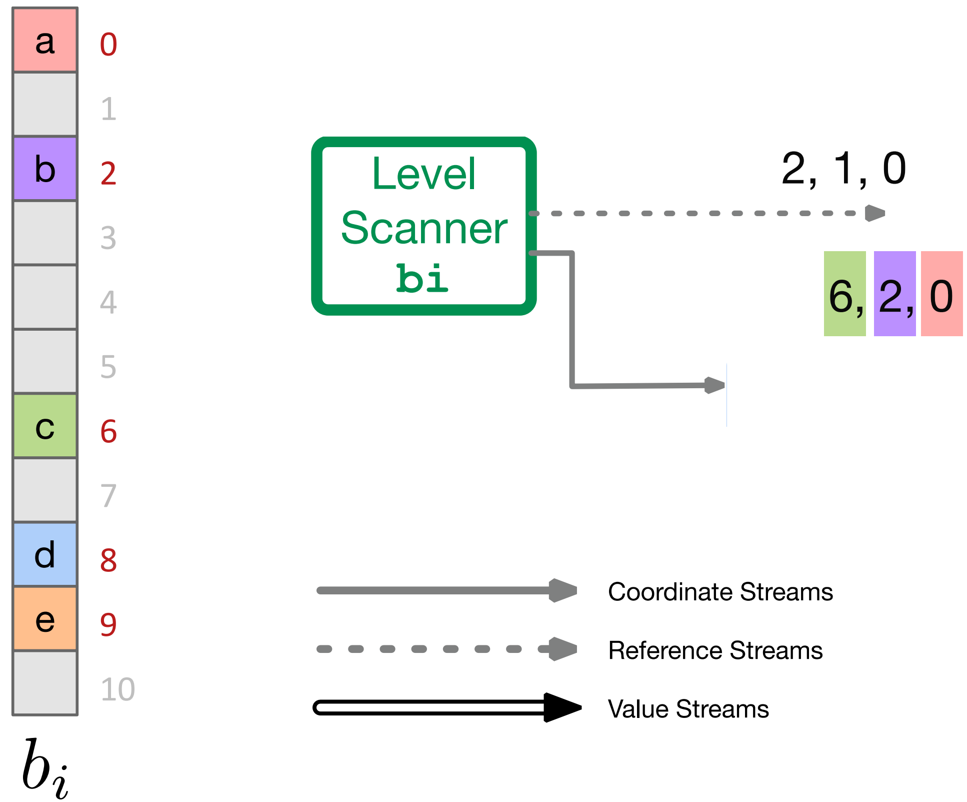
$$x_i = b_i \cdot c$$



Vector scaling example with repeaters

SAM Graph

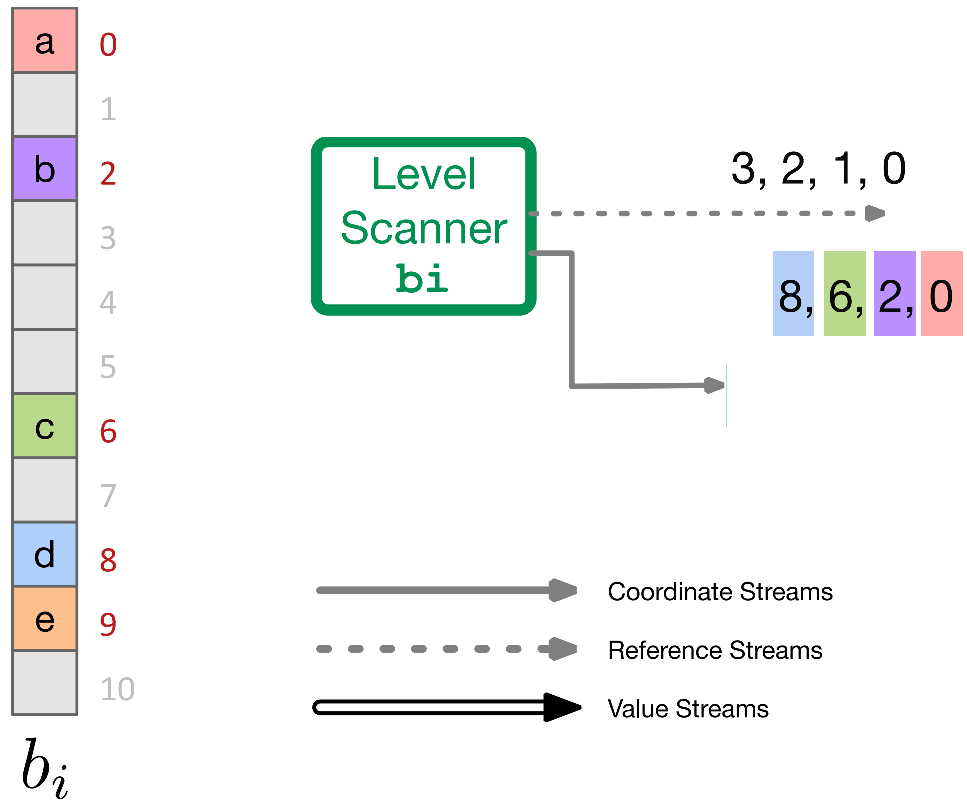
$$x_i = \underline{b_i} \cdot c$$



Vector scaling example with repeaters

SAM Graph

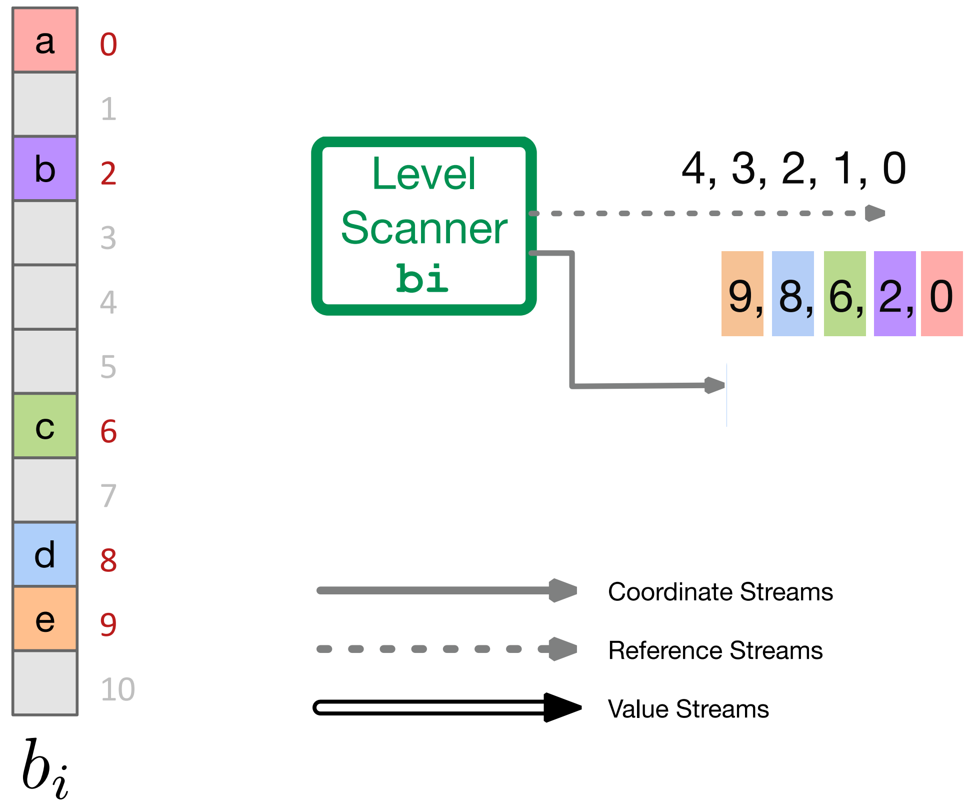
$$x_i = \underline{b_i} \cdot c$$



Vector scaling example with repeaters

SAM Graph

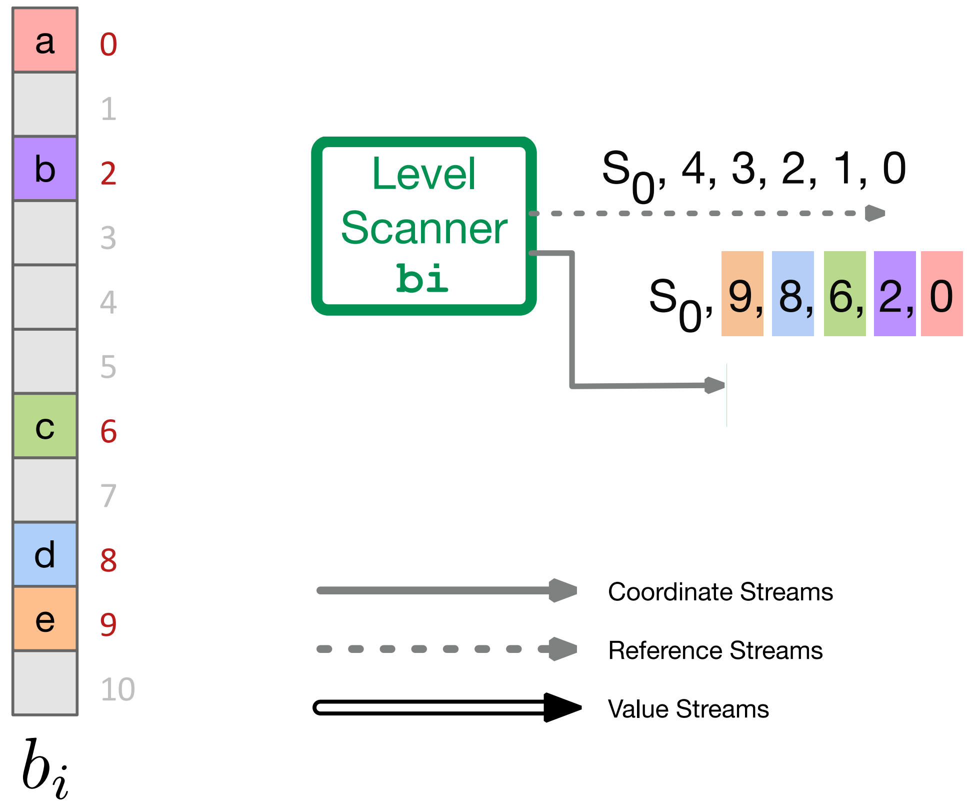
$$x_i = \underline{b_i} \cdot c$$



Vector scaling example with repeaters

SAM Graph

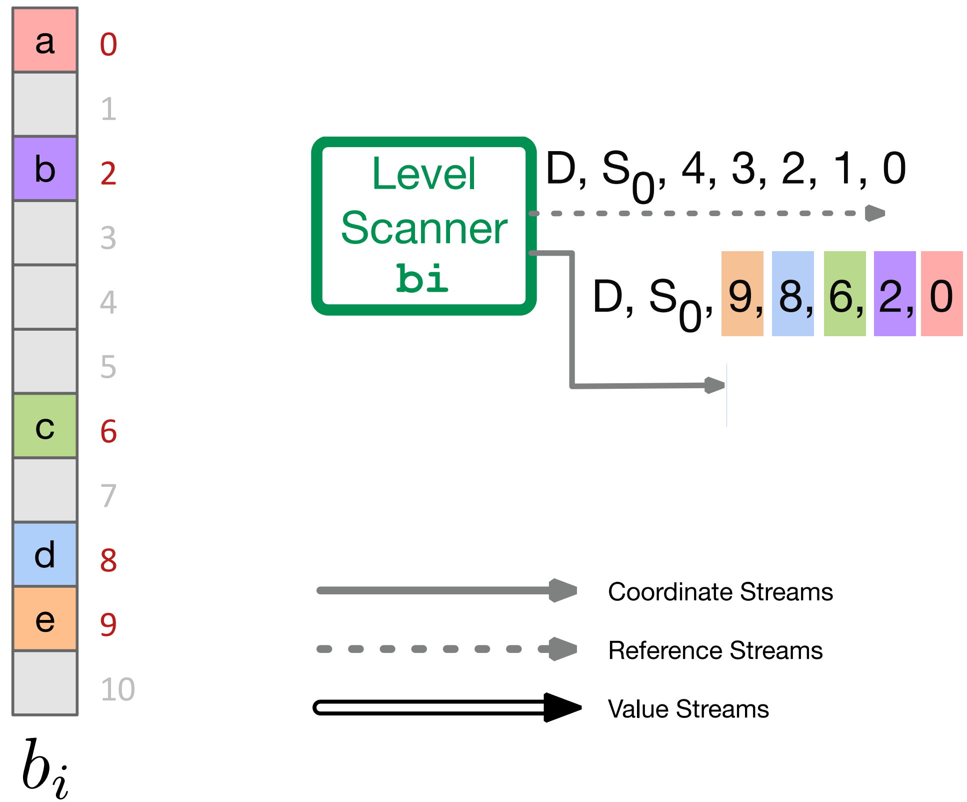
$$x_i = \underline{b_i} \cdot c$$



Vector scaling example with repeaters

SAM Graph

$$x_i = \underline{b_i} \cdot c$$

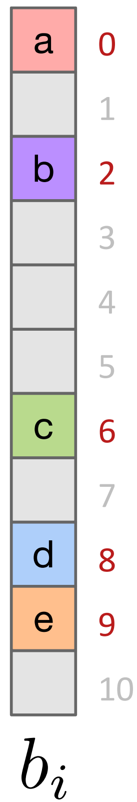


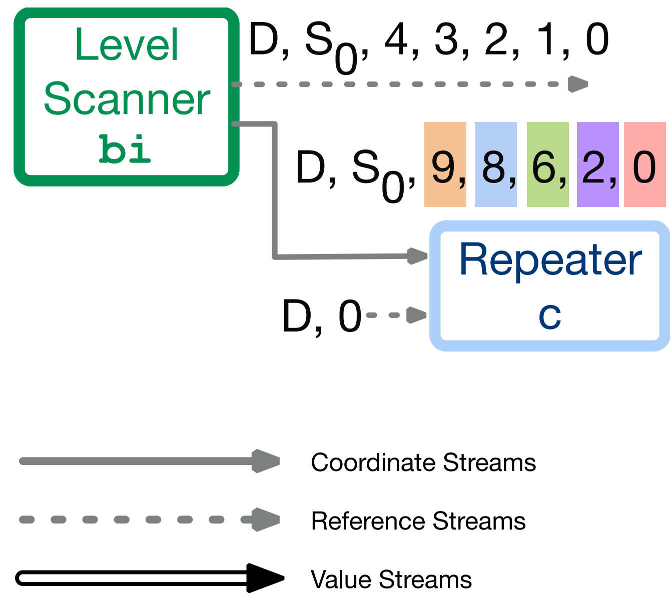
Vector scaling example with repeaters

SAM Graph

$$x_i = \underline{b_i \cdot c}$$

$$c \cdot$$

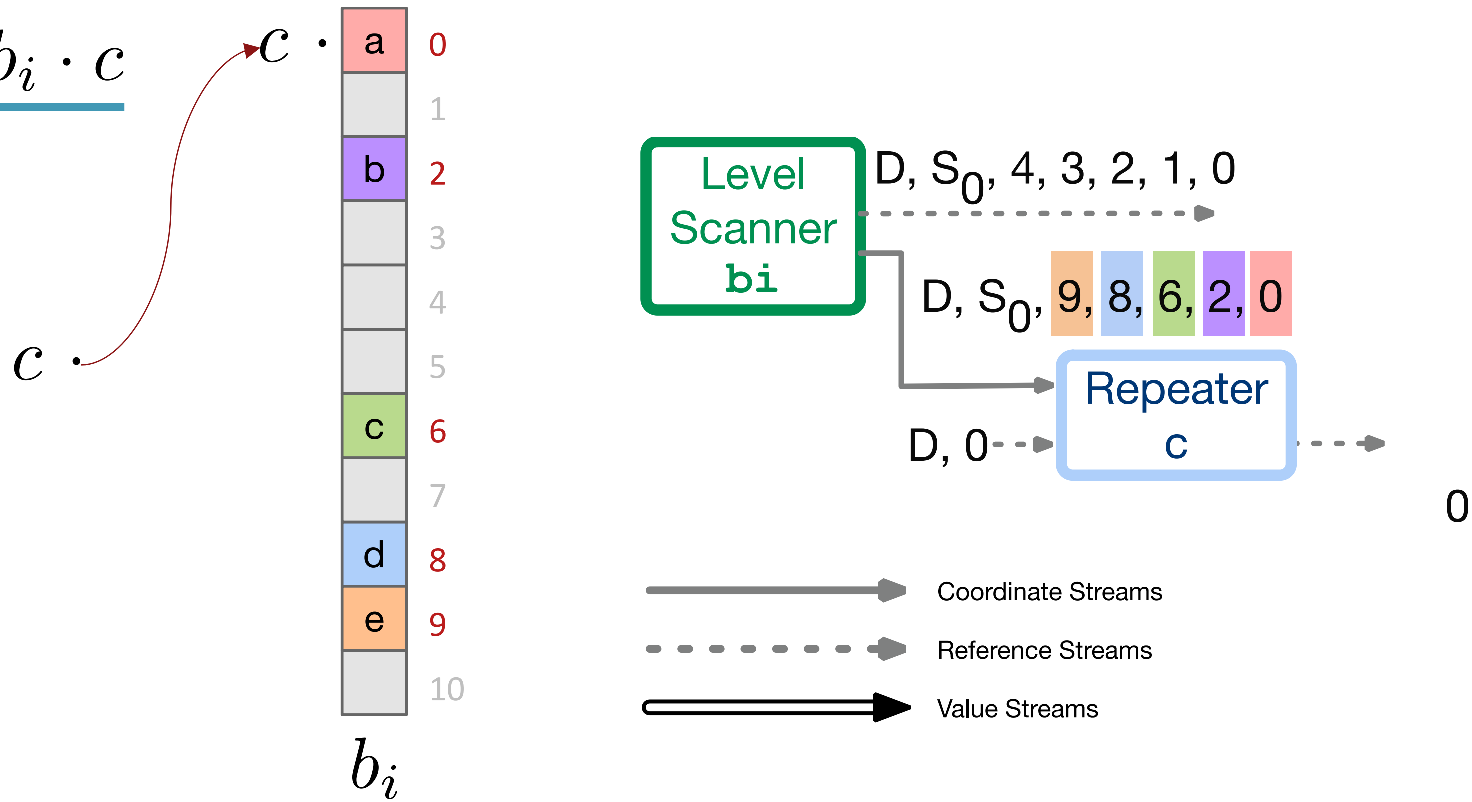




Vector scaling example with repeaters

SAM Graph

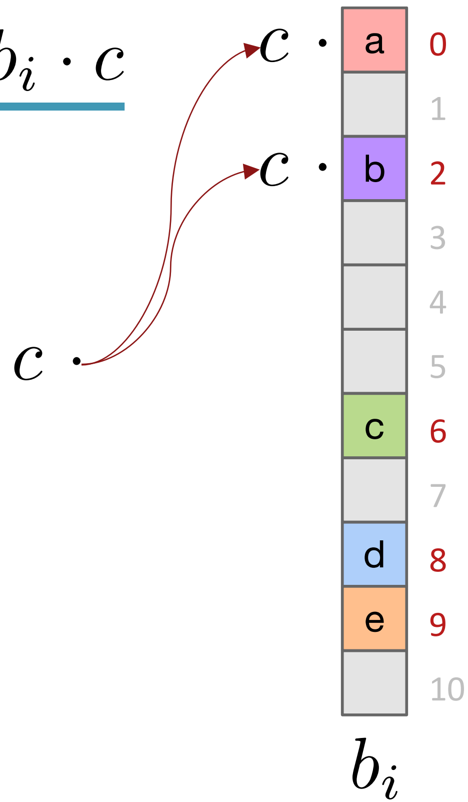
$$x_i = b_i \cdot c$$

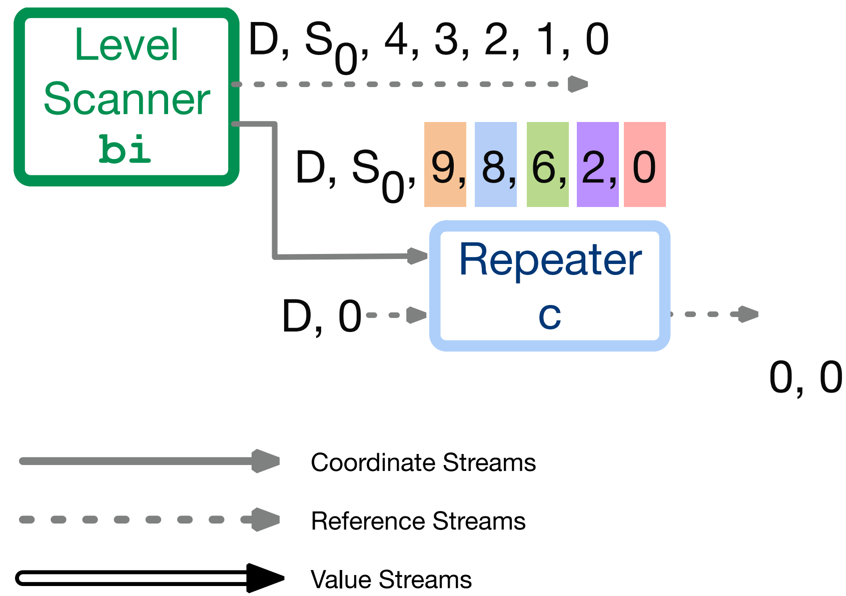


Vector scaling example with repeaters

SAM Graph

$$x_i = b_i \cdot c$$

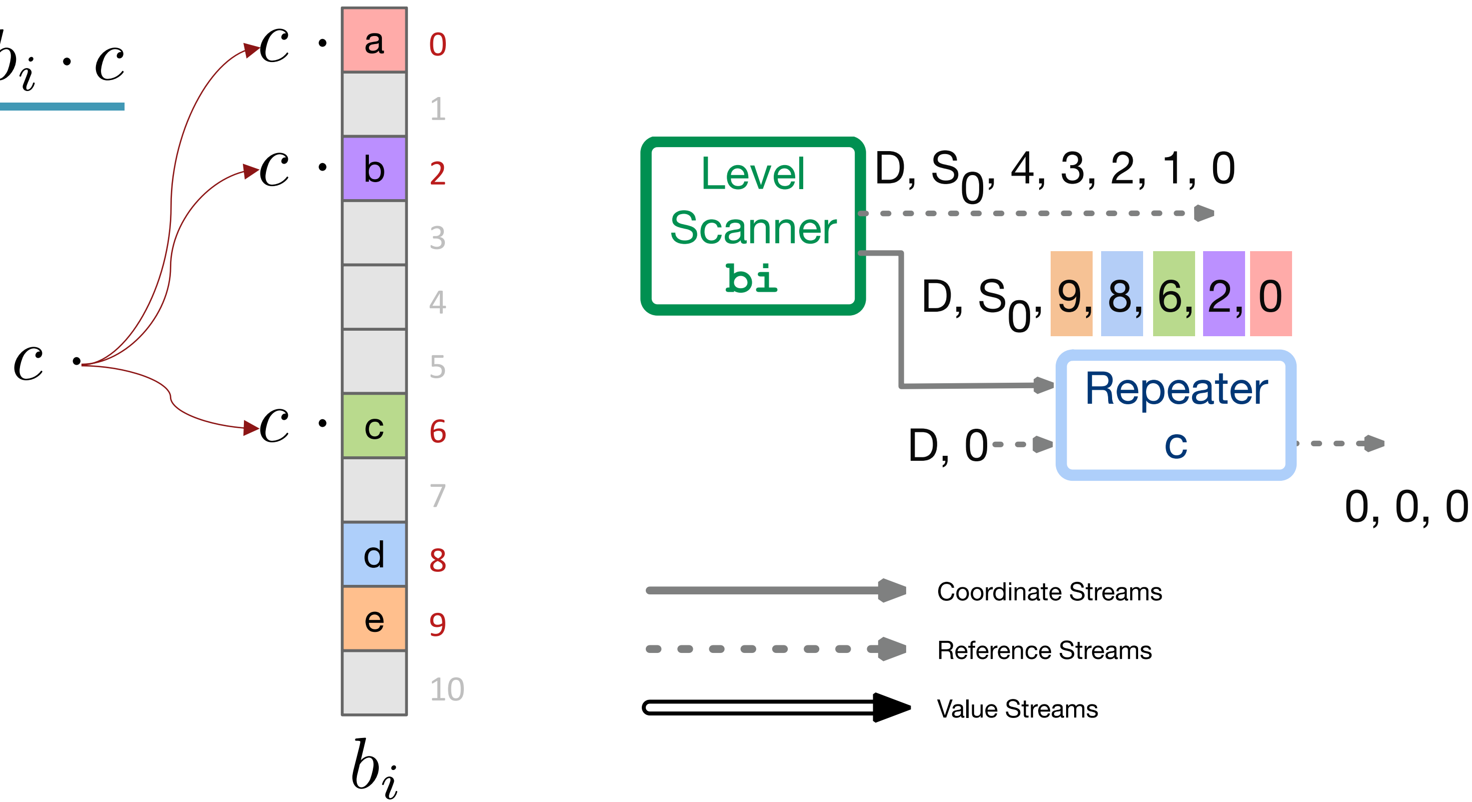




Vector scaling example with repeaters

SAM Graph

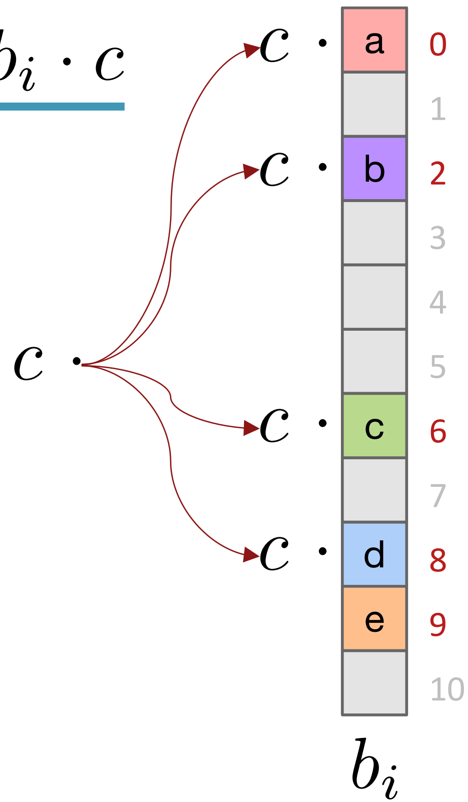
$$x_i = b_i \cdot c$$

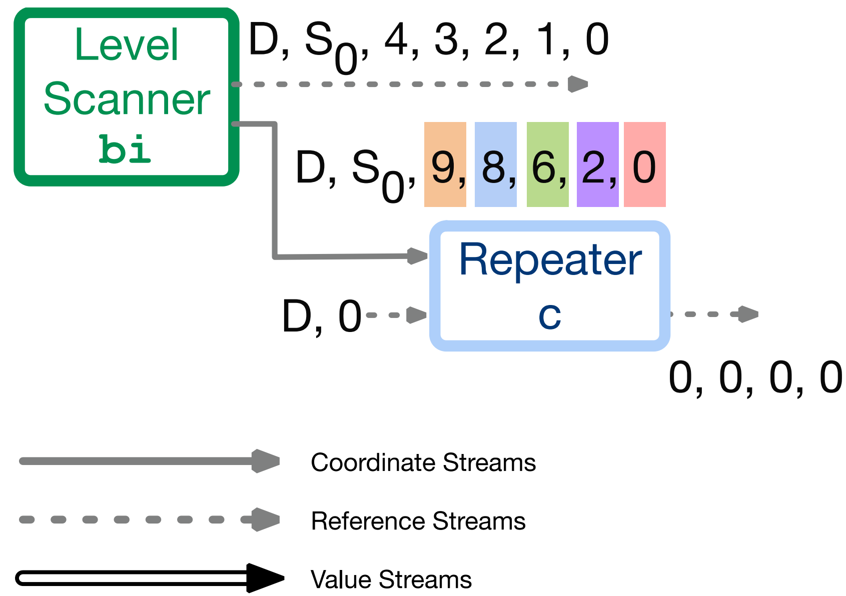


Vector scaling example with repeaters

SAM Graph

$$x_i = b_i \cdot c$$

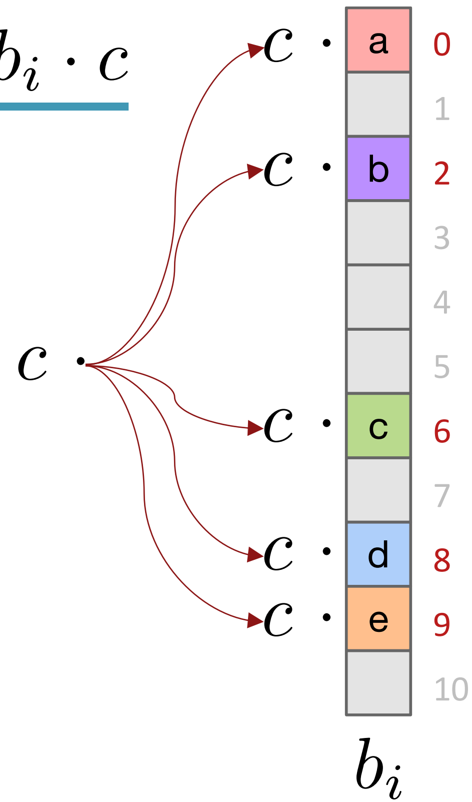


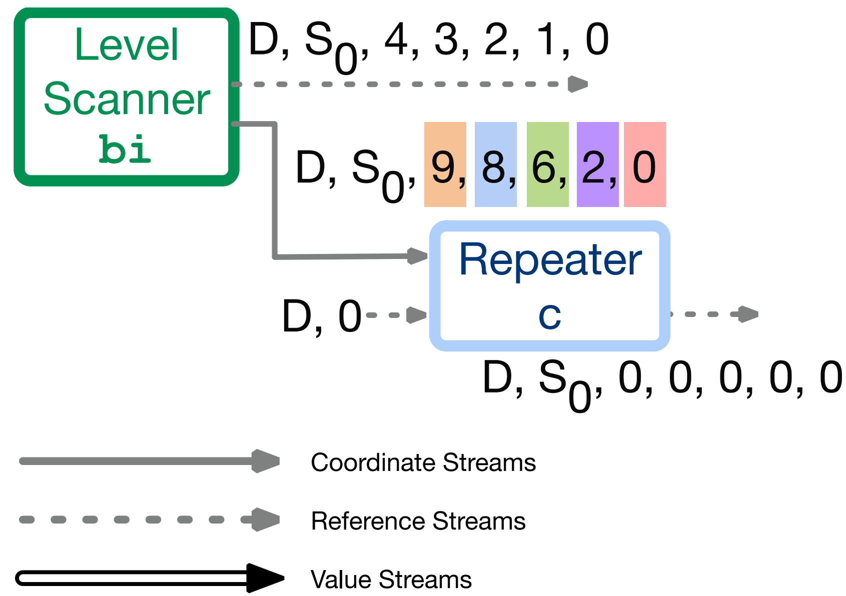


Vector scaling example with repeaters

SAM Graph

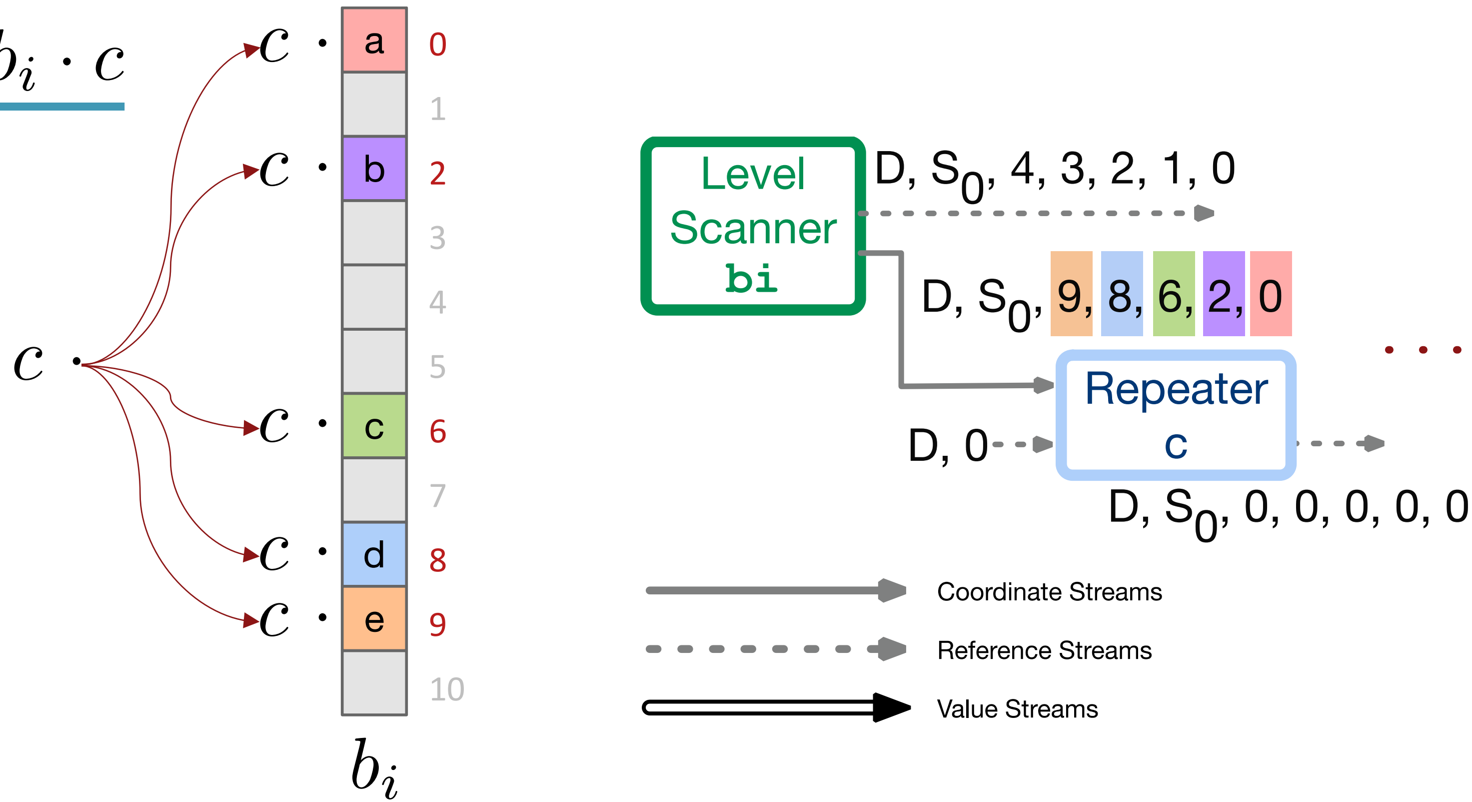
$$x_i = b_i \cdot c$$





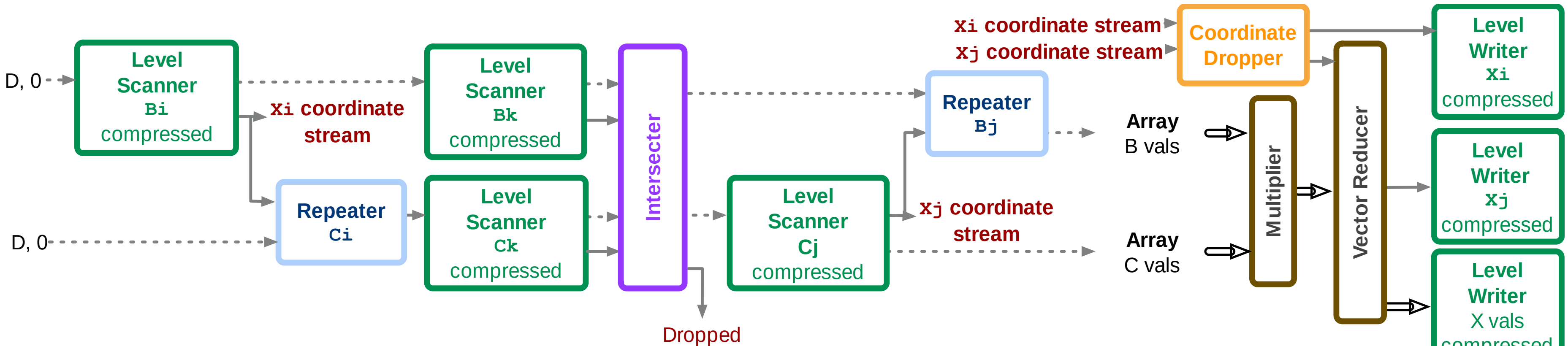
Vector scaling example with repeaters

$$x_i = b_i \cdot c$$



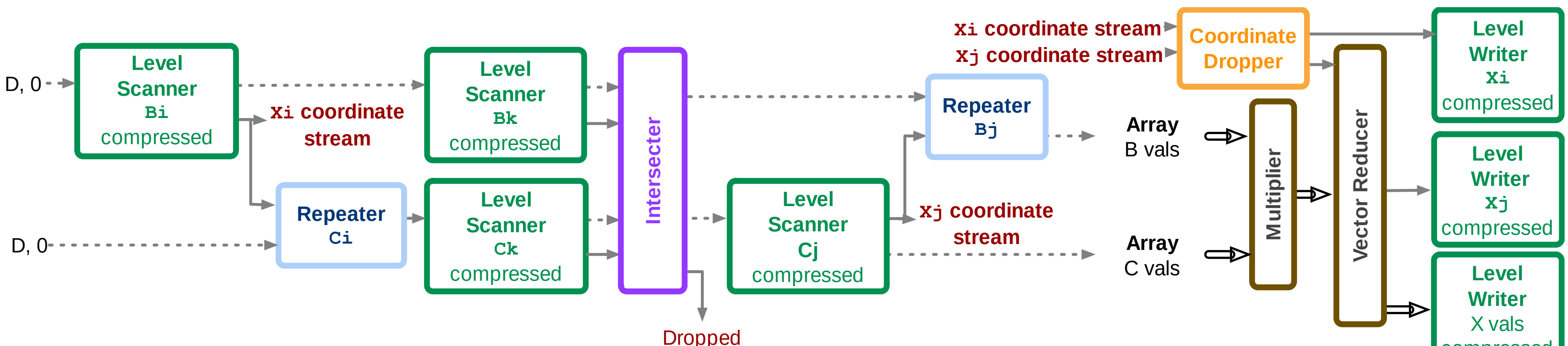
Primitives compose to compute expressions: SpM*SpM

SAM Graph



Primitives compose to compute expressions: SpM*SpM

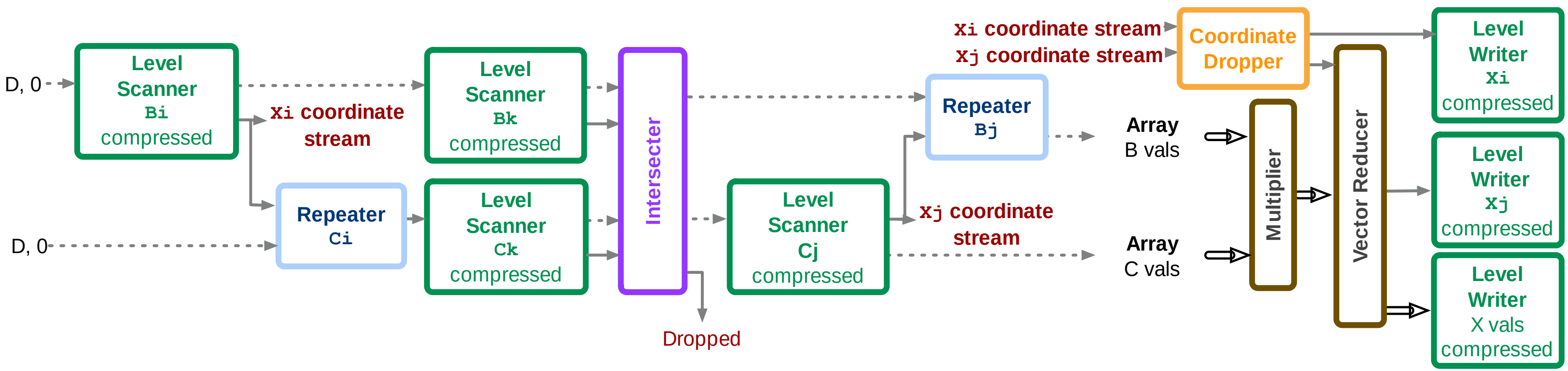
SAM Graph



$$\forall_i \forall_k \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

Primitives compose to compute expressions: SpM*SpM

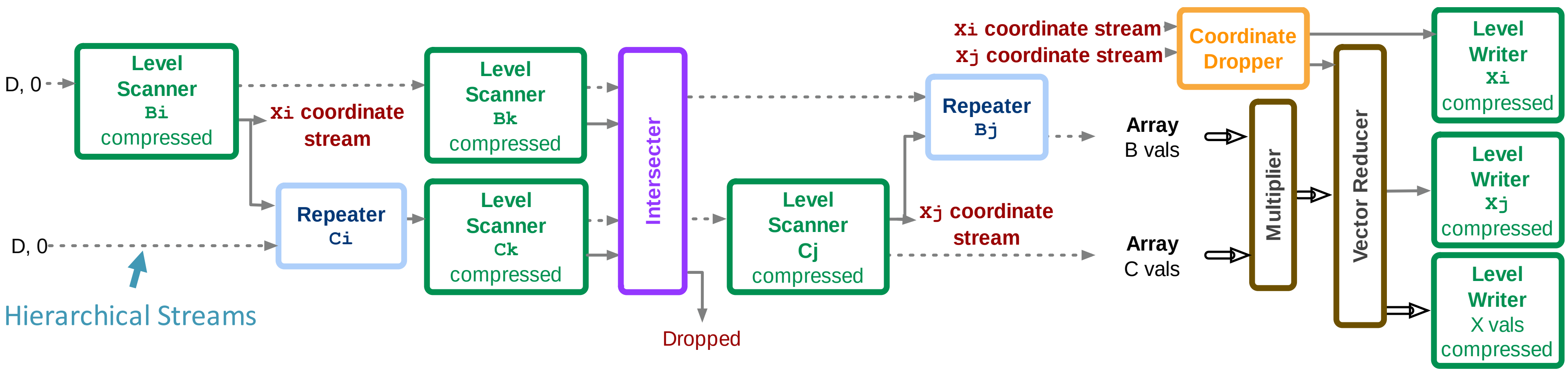
SAM Graph



$$\forall_i \forall_k \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

Primitives compose to compute expressions: SpM*SpM

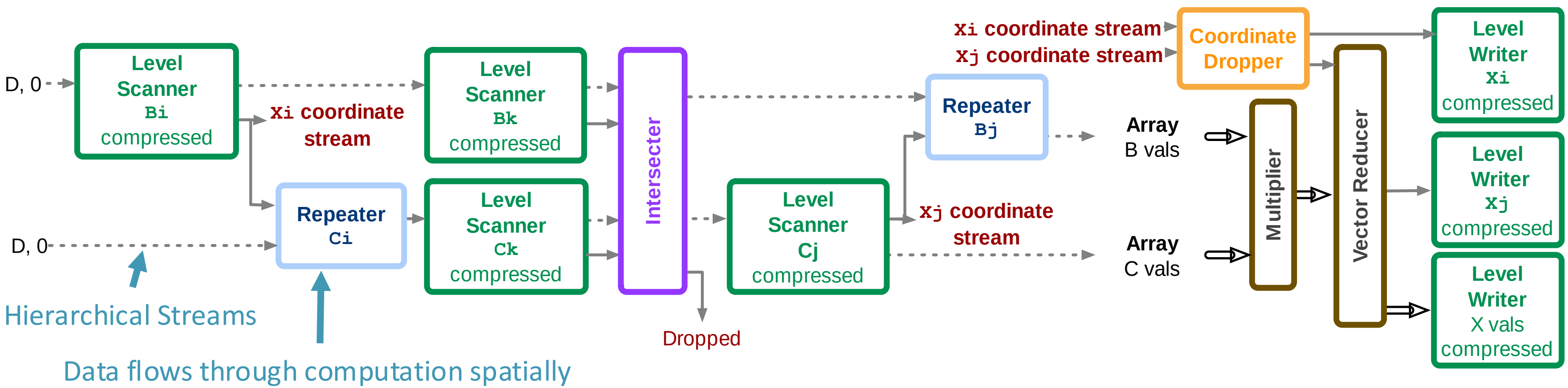
SAM Graph



$$\forall_i \underline{\forall_k} \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

Primitives compose to compute expressions: SpM*SpM

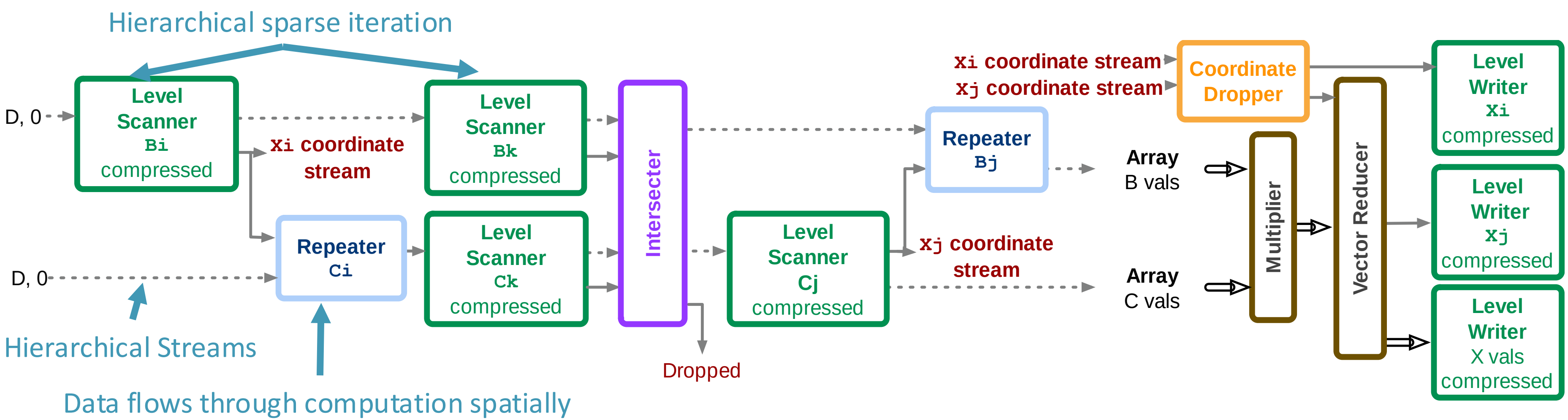
SAM Graph



$$\forall_i \forall_k \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

Primitives compose to compute expressions: SpM*SpM

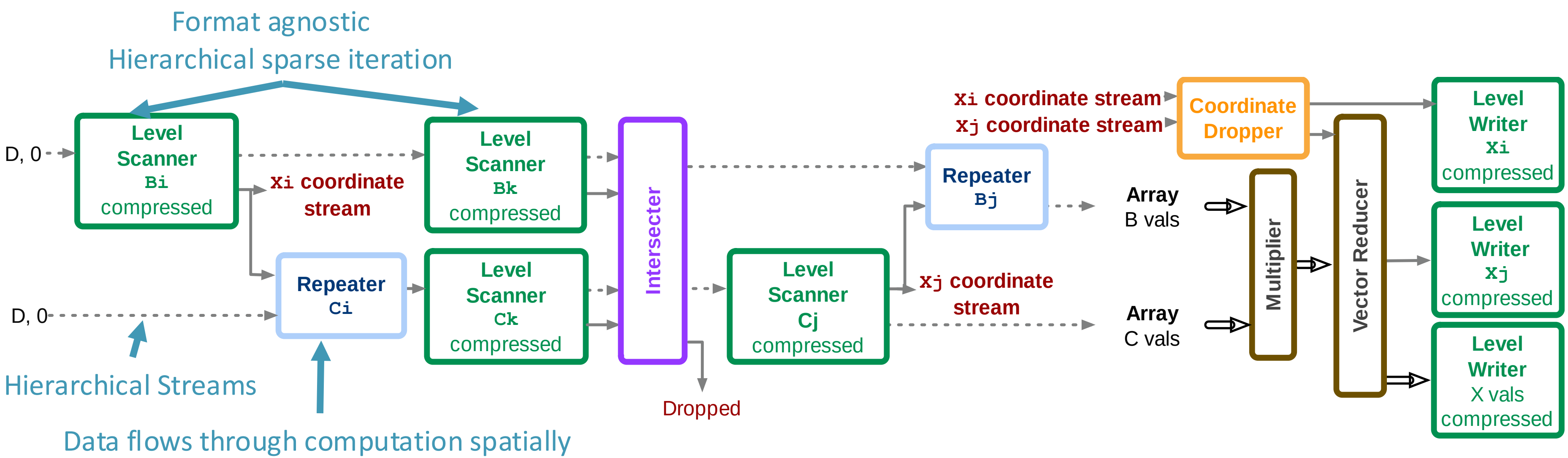
SAM Graph



$$\forall_i \forall_k \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

Primitives compose to compute expressions: SpM*SpM

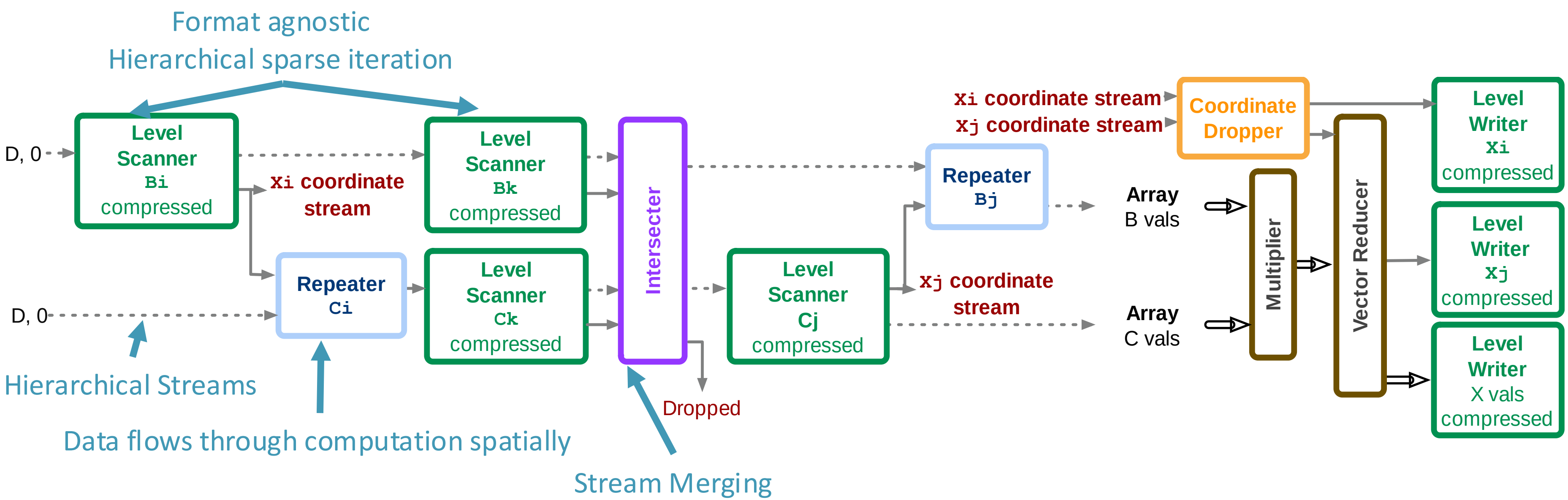
SAM Graph



$$\forall_i \forall_k \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

SAM Graph

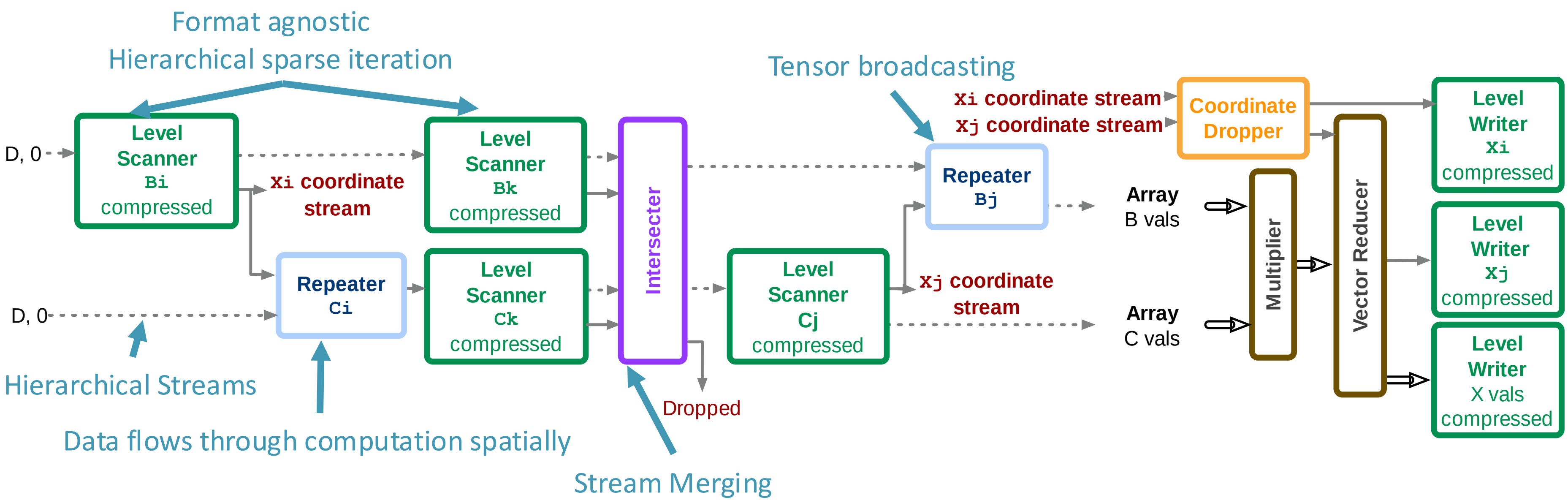
Primitives compose to compute expressions: SpM*SpM



$$\forall_i \forall_k \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

Primitives compose to compute expressions: SpM*SpM

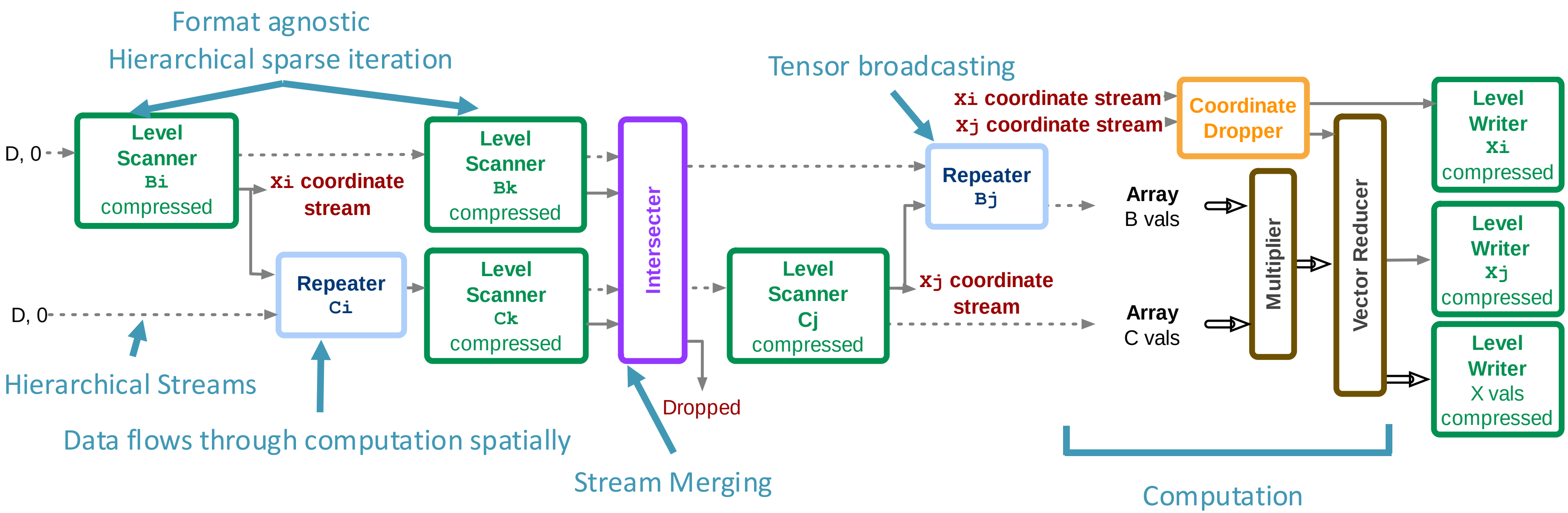
SAM Graph



$$\forall_i \forall_k \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

Primitives compose to compute expressions: SpM*SpM

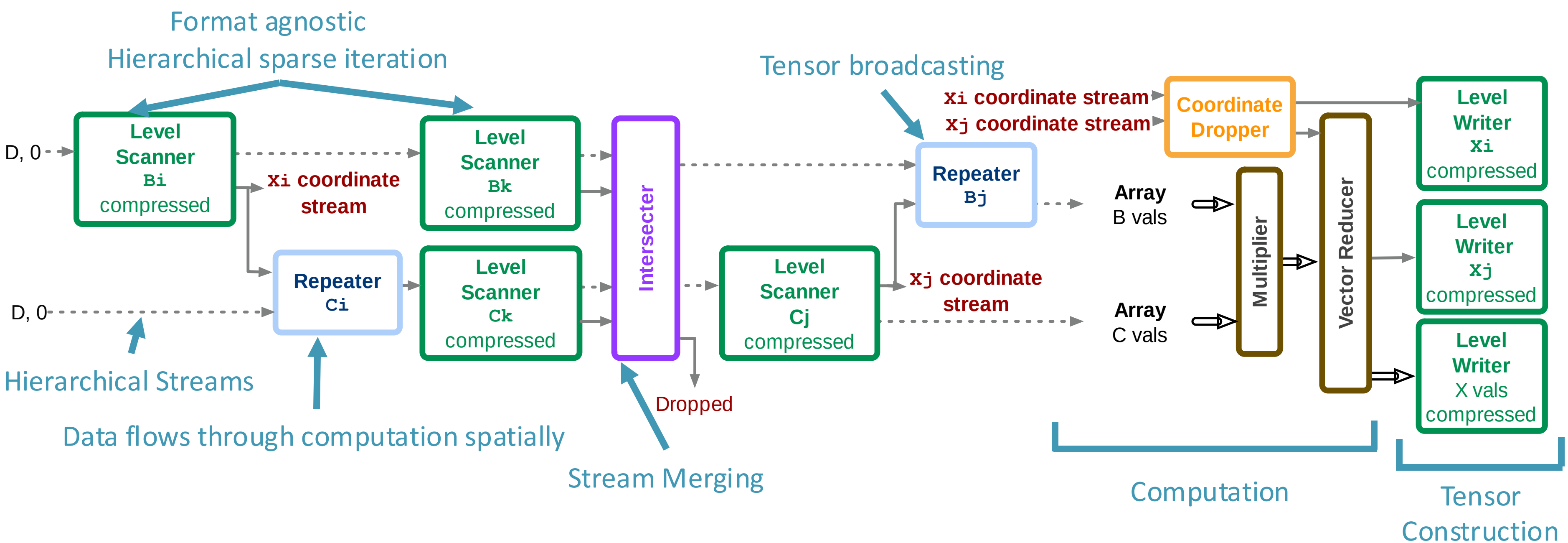
SAM Graph



$$\forall_i \forall_k \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

Primitives compose to compute expressions: SpM*SpM

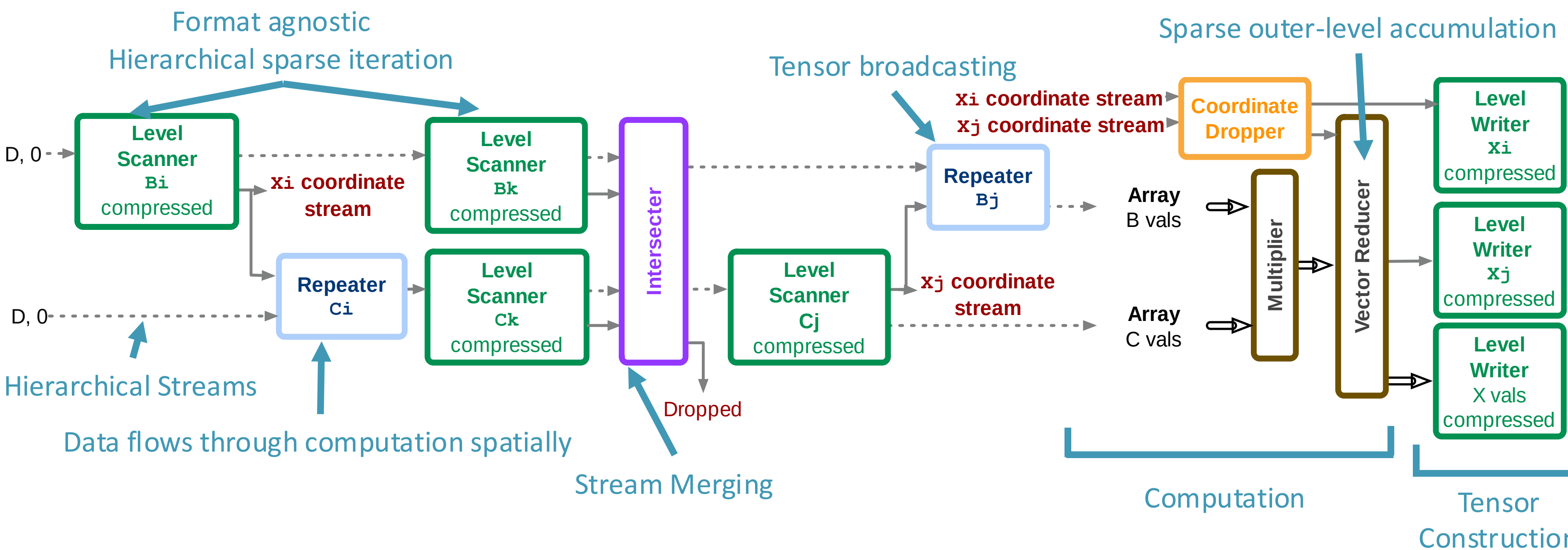
SAM Graph



$$\forall_i \forall_k \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

Primitives compose to compute expressions: SpM*SpM

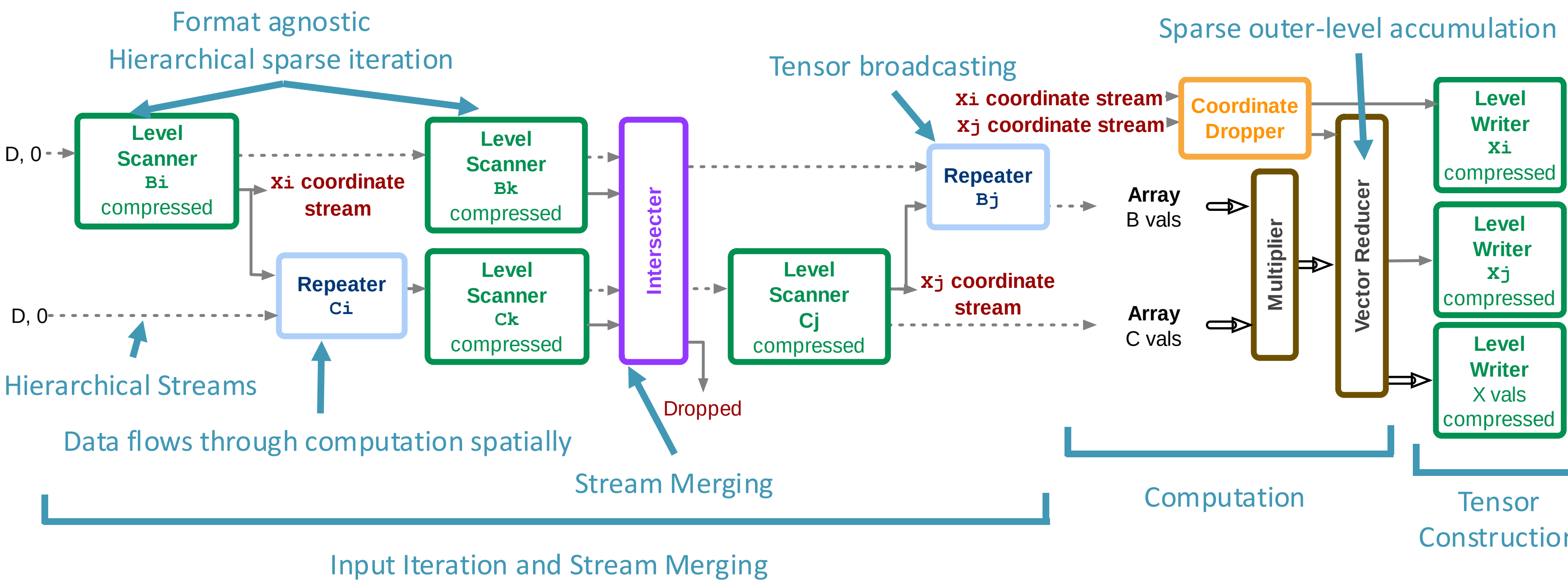
SAM Graph



$$\forall_i \forall_k \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

Primitives compose to compute expressions: SpM*SpM

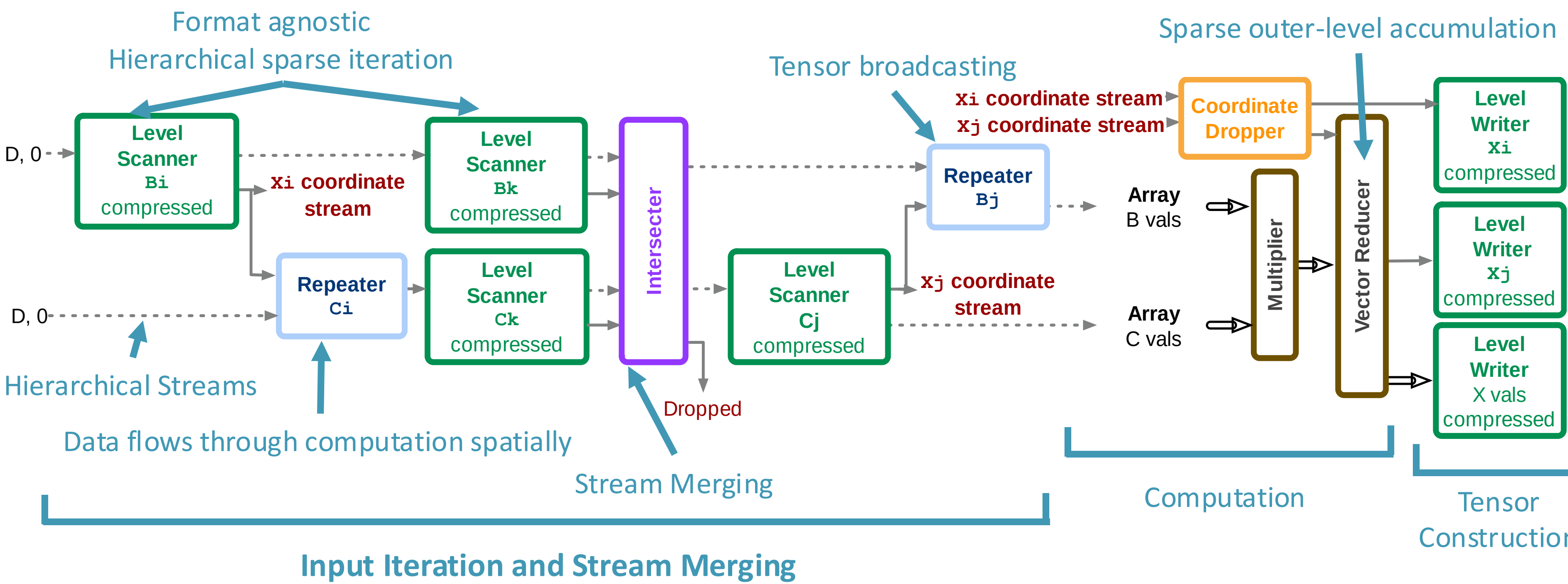
SAM Graph



$$\forall_i \forall_k \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

Primitives compose to compute expressions: SpM*SpM

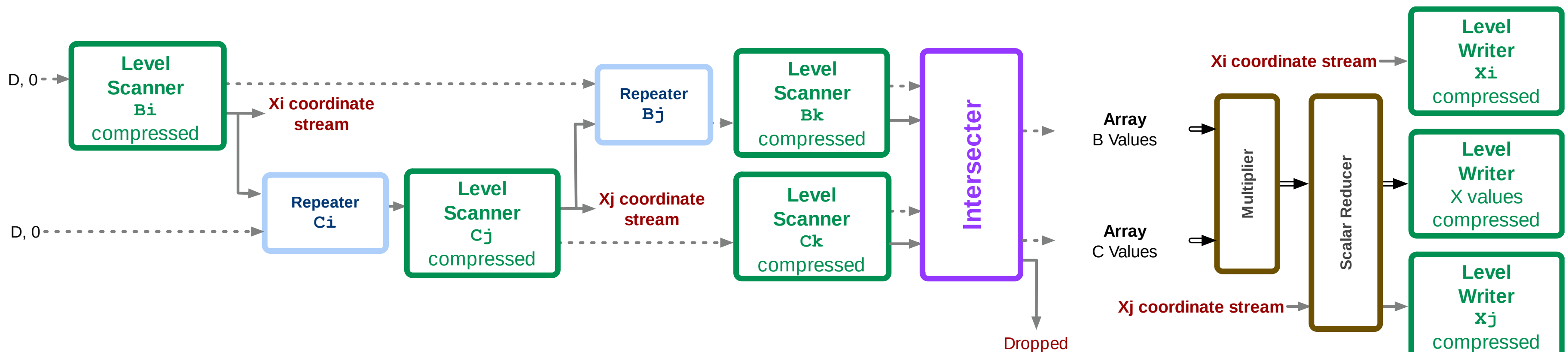
SAM Graph



$$\forall_i \forall_k \forall_j X_{ij} = B_{ik} \cdot C_{kj}$$

SAM Graph

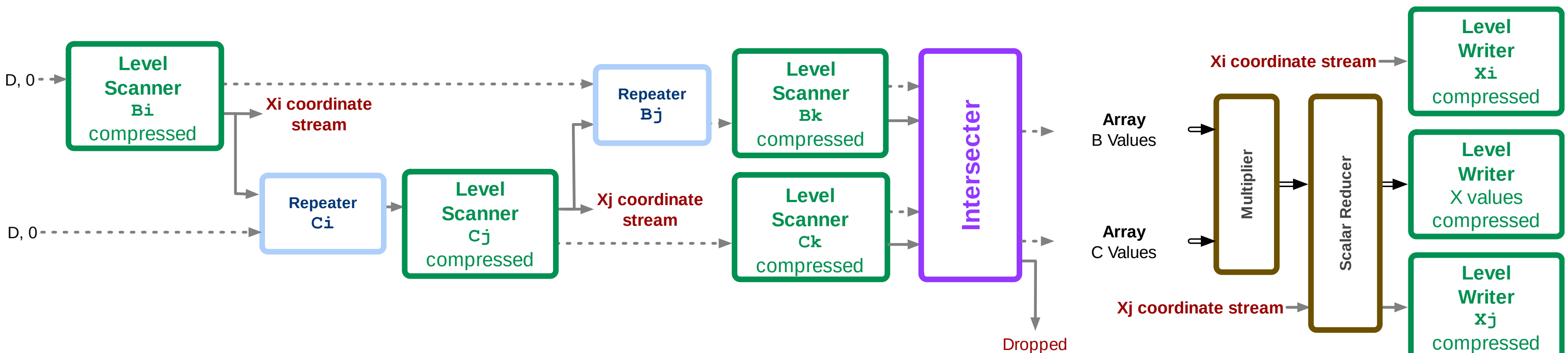
Inner-product algorithm in SAM



$$\forall_i \forall_j \forall_k X_{ij} = B_{ik} * C_{kj}$$

SAM Graph

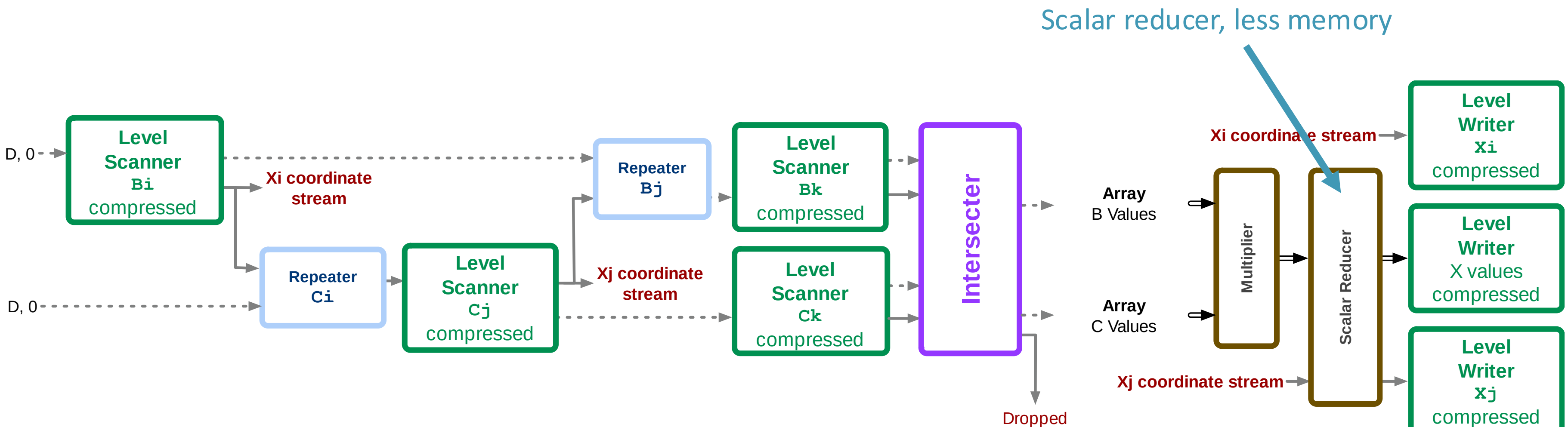
Inner-product algorithm in SAM



$$\forall_i \forall_j \forall_k X_{ij} = B_{ik} * C_{kj}$$

SAM Graph

Inner-product algorithm in SAM

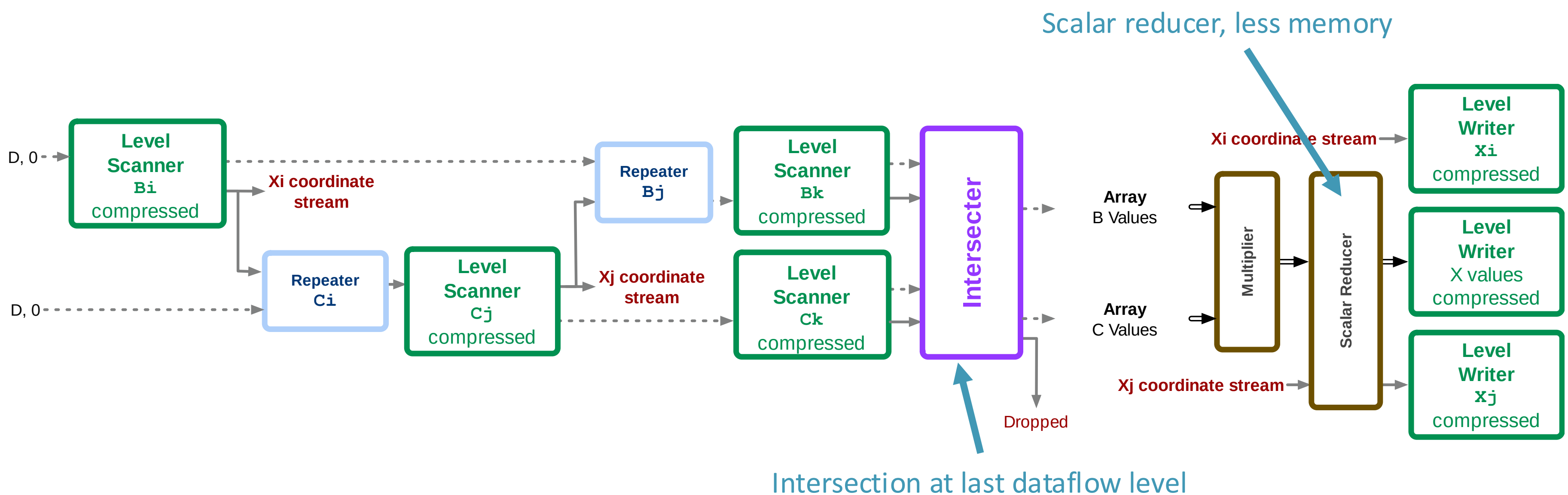


$$\forall_i \forall_j \forall_k X_{ij} = B_{ik} * C_{kj}$$

Stanford University

SAM Graph

Inner-product algorithm in SAM



$$\forall_i \forall_j \forall_k X_{ij} = B_{ik} * C_{kj}$$

Custard's compiler algorithm to SAM

Compiler: *Custard*

Expression

$$X_{ij} = B_{ik} * C_{kj}$$

Format Language

```
B=({sparse,sparse},
   {mode0,mode1})
C=({sparse, sparse},
   {mode1, mode0})
```

Schedule

```
reorder(i, k, j)
```

Custard's compiler algorithm to SAM

Compiler: *Custard*

Expression

$$X_{ij} = B_{ik} * C_{kj}$$

Format Language

```
B=({sparse,sparse},
   {mode0,mode1})
C=({sparse, sparse},
   {mode1, mode0})
```

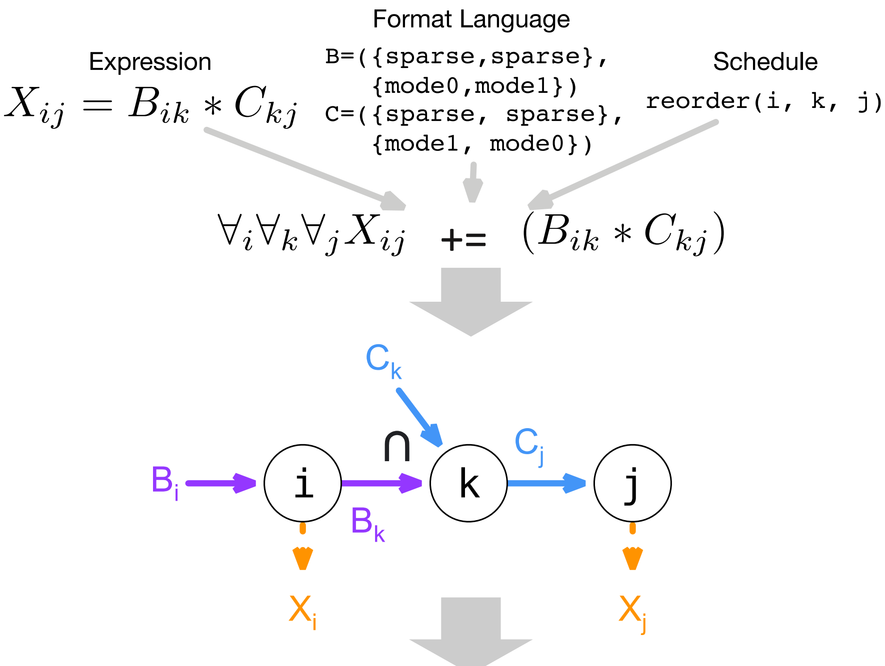
Schedule

```
reorder(i, k, j)
```

$$\forall_i \forall_k \forall_j X_{ij} \mathrel{+}= (B_{ik} * C_{kj})$$

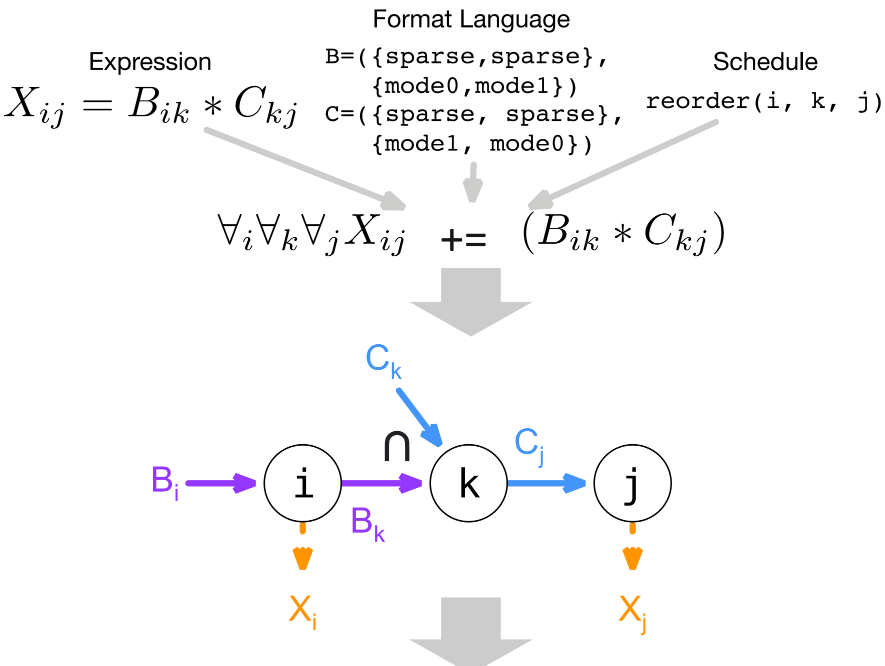
Custard's compiler algorithm to SAM

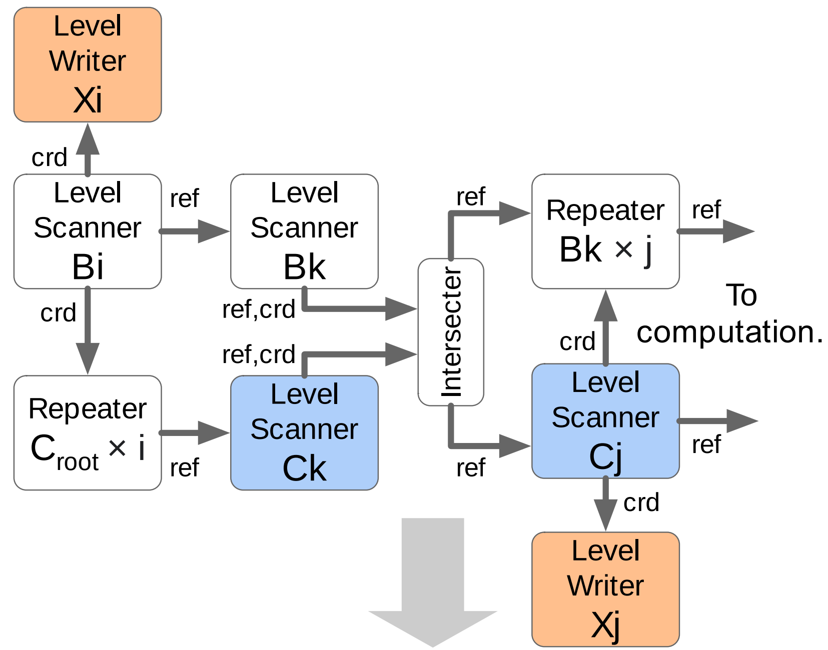
Compiler: *Custard*



Custard's compiler algorithm to SAM

Compiler: *Custard*





Lower to Example Implementation

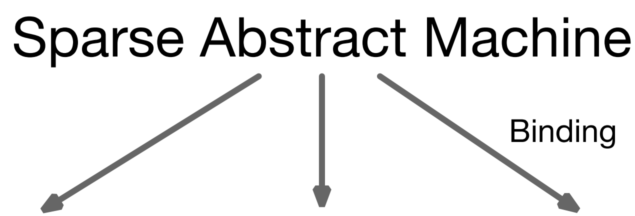
Demo: Generating SAM graphs with Custard

> `./sparse_demo.sh compile`

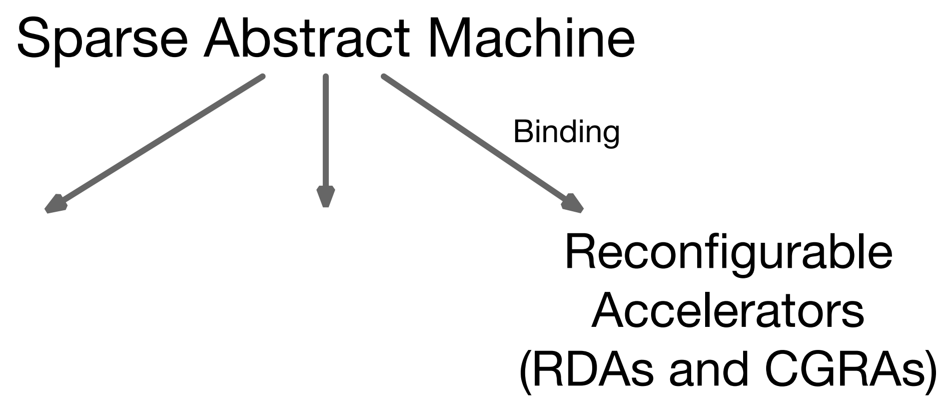
- This runs the applications in `./sam/compiler/sam-kernels.sh` through the Custard compiler
- All SAM graphs generated in `./sam/compiler/sam-outputs/`

- View the SpMSpM kernel `matmul_ijk` in `./sam/compiler/sam-outputs/png/matmul_ijk.png` in VSCode or using `docker cp`
- We will also view a smaller kernel, `mat_elemadd` in `./sam/compiler/sam-outputs/png/mat_elemadd.png`, which we will be using for the rest of the demo

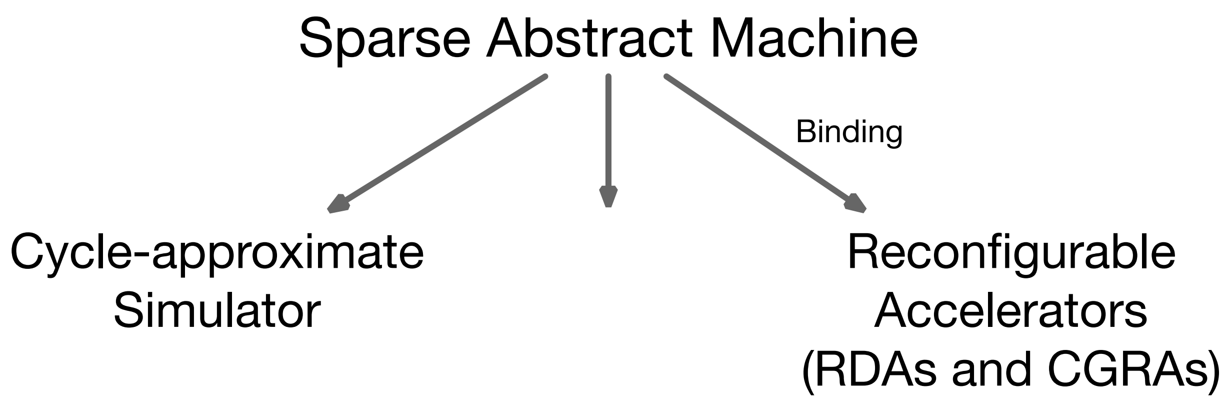
Although abstract, SAM binds to real hardware



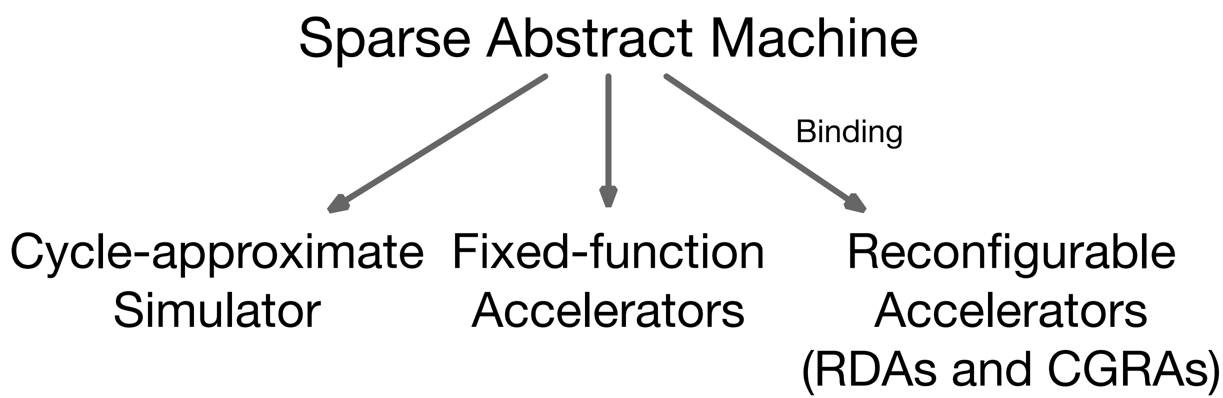
Although abstract, SAM binds to real hardware



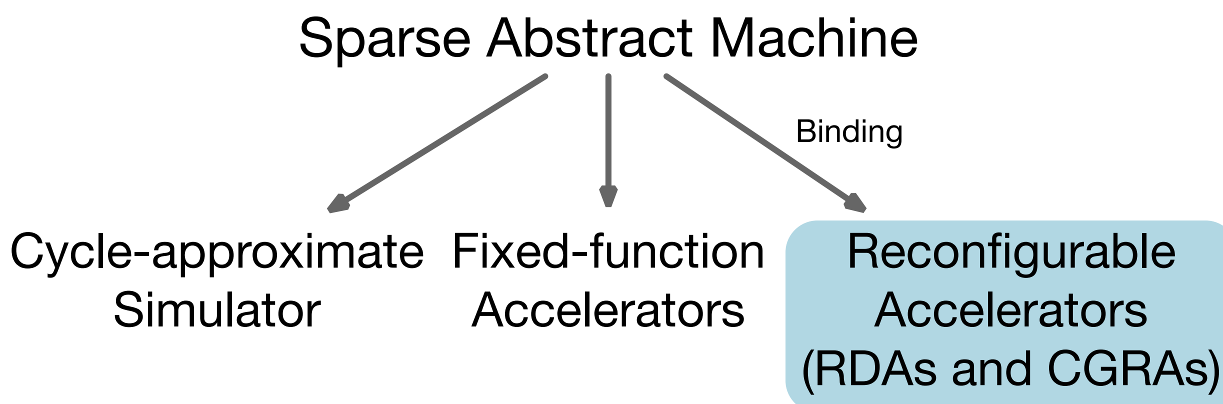
Although abstract, SAM binds to real hardware



Although abstract, SAM binds to real hardware

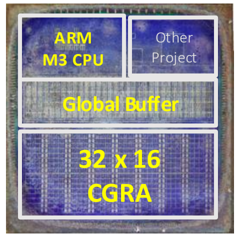


Although abstract, SAM binds to real hardware



Programming dataflow requires an abstract machine that a compiler can target

High-Level
Input Languages
(APIs)

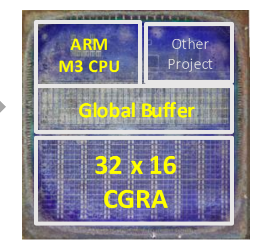


Sparse Accelerator

Programming dataflow requires an abstract machine that a compiler can target

High-Level
Input Languages
(APIs)

Compile

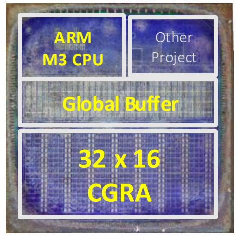


Sparse Accelerator

Programming dataflow requires an abstract machine that a compiler can target

High-Level Input Languages (APIs)

Compile



Sparse Accelerator

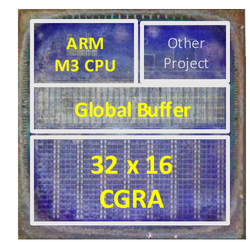
A clean interface decouples the compiler from specific hardware implementations

Programming dataflow requires an abstract machine that a compiler can target

High-Level Input Languages (APIs)

Compile

Map



Sparse Accelerator

A clean interface decouples the compiler from specific hardware implementations

Programming dataflow requires an abstract machine that a compiler can target

[Hsu et al. ASPLOS 2023]

High-Level Input Languages (APIs)

Compile

The Sparse Abstract Machine (SAM)

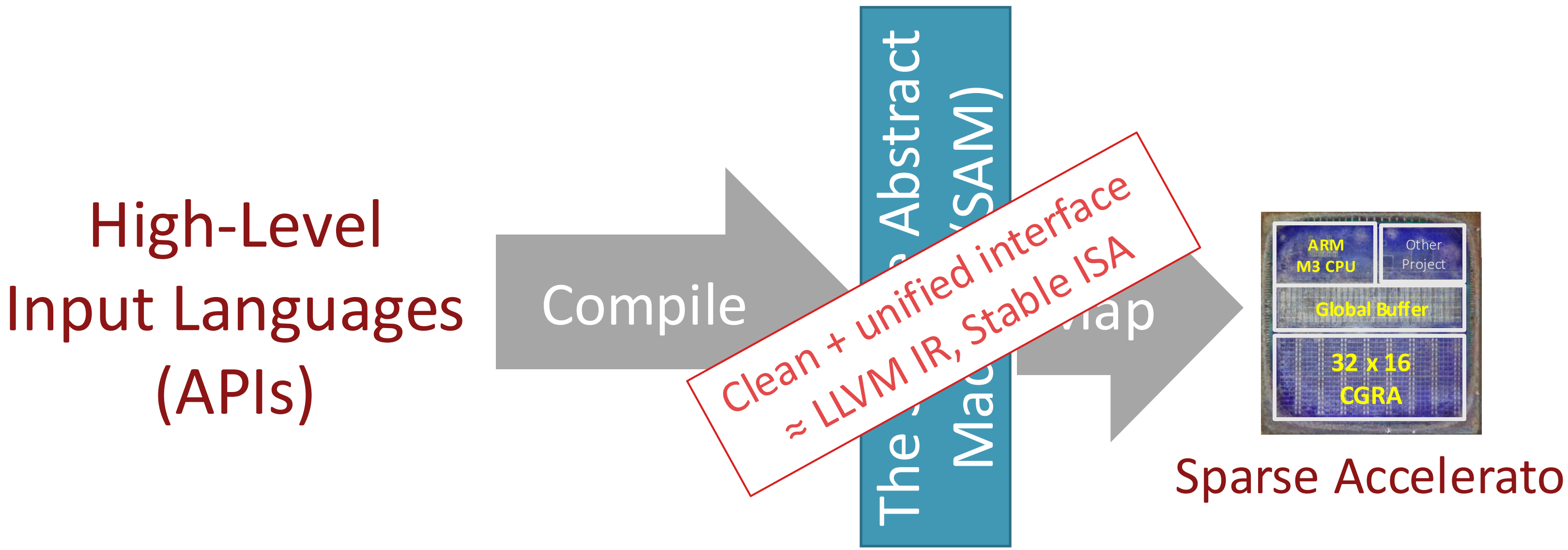
Map

Sparse Accelerator

A clean interface decouples the compiler from specific hardware implementations

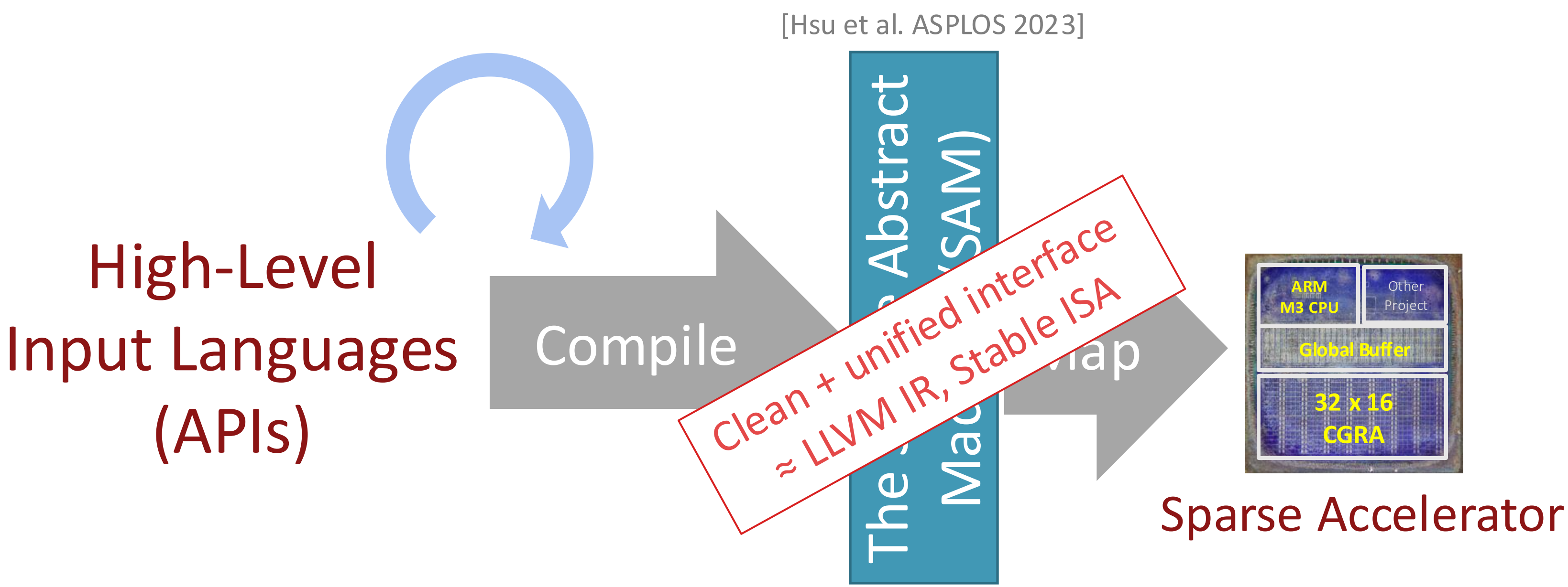
Programming dataflow requires an abstract machine that a compiler can target

[Hsu et al. ASPLOS 2023]



A clean interface decouples the compiler from specific hardware implementations

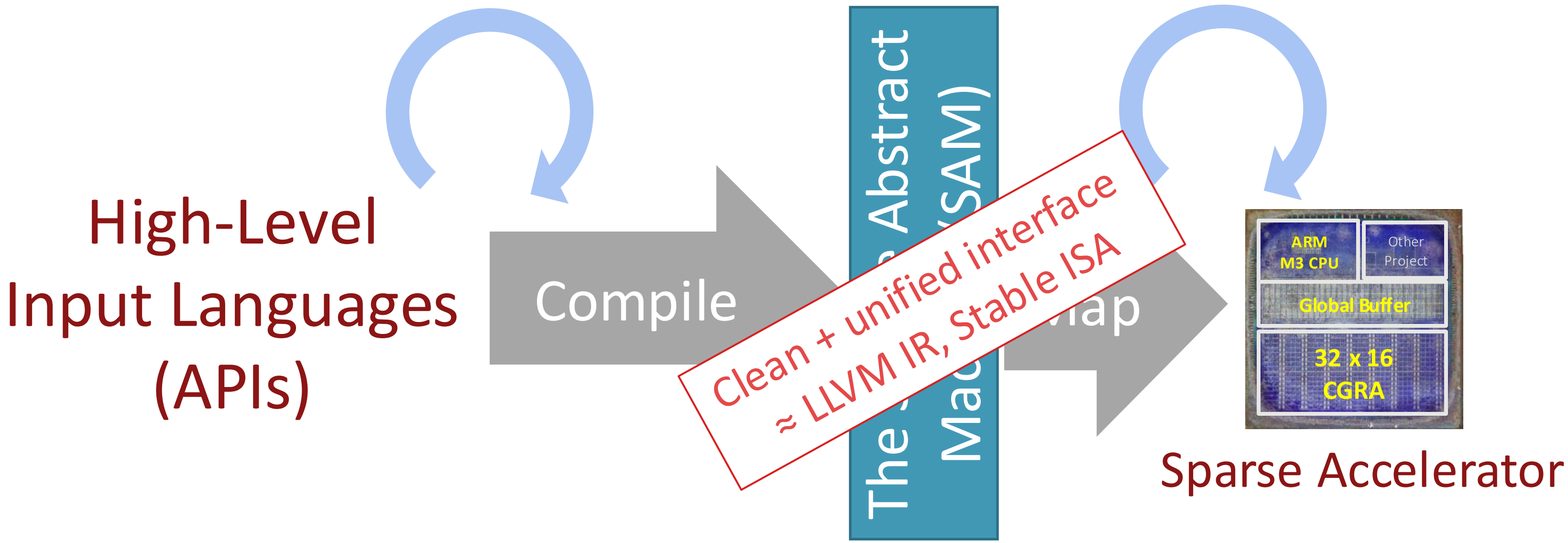
Programming dataflow requires an abstract machine that a compiler can target



A clean interface decouples the compiler from specific hardware implementations

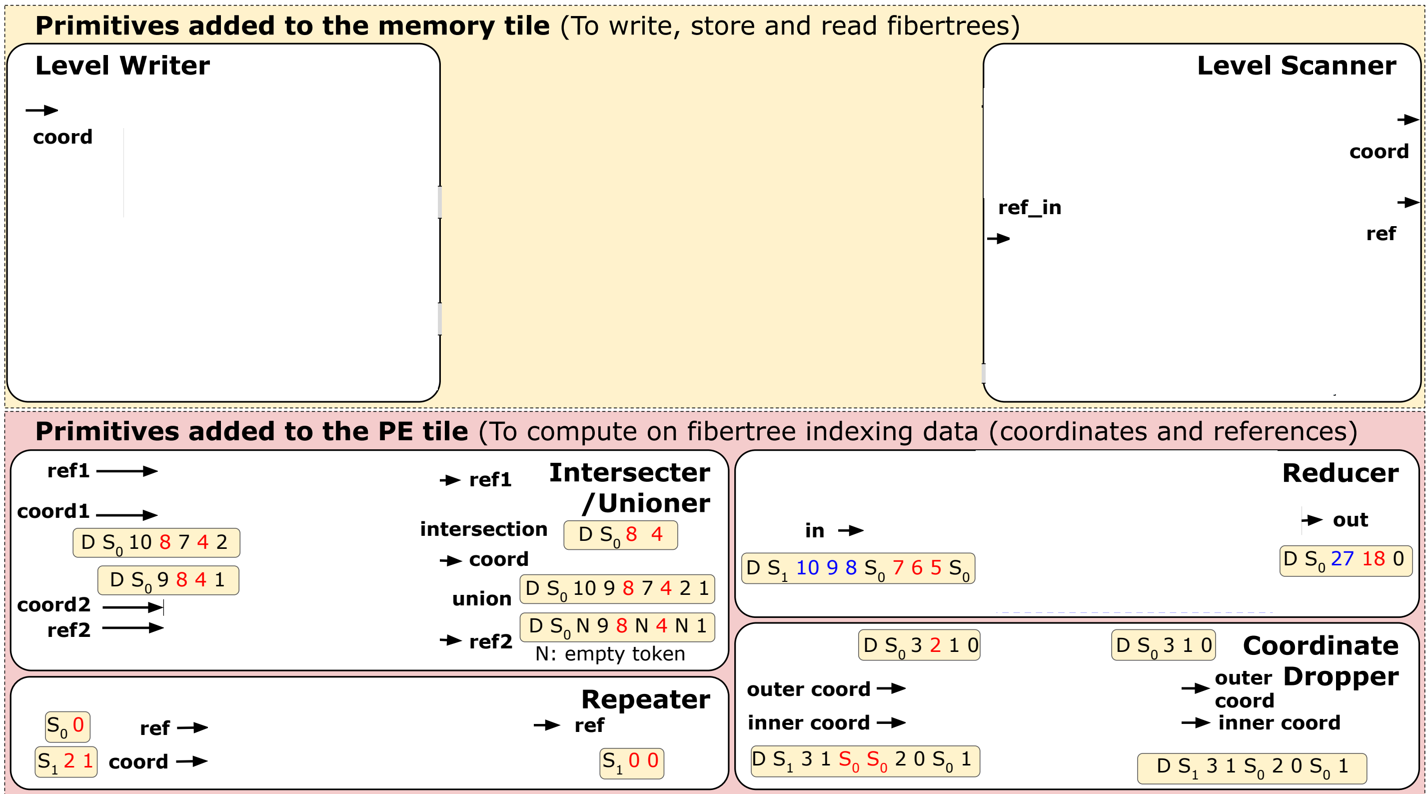
Programming dataflow requires an abstract machine that a compiler can target

[Hsu et al. ASPLOS 2023]

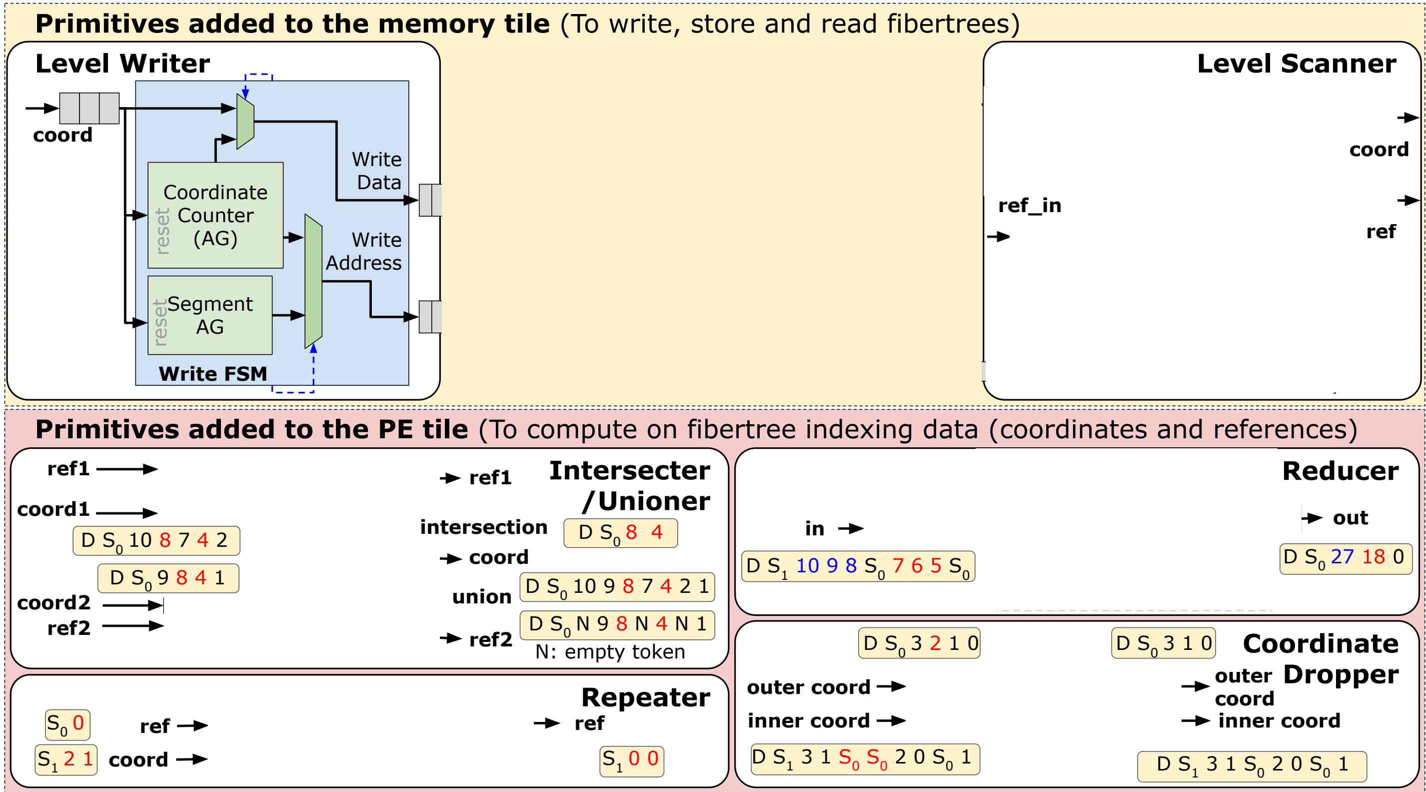


A clean interface decouples the compiler from specific hardware implementations

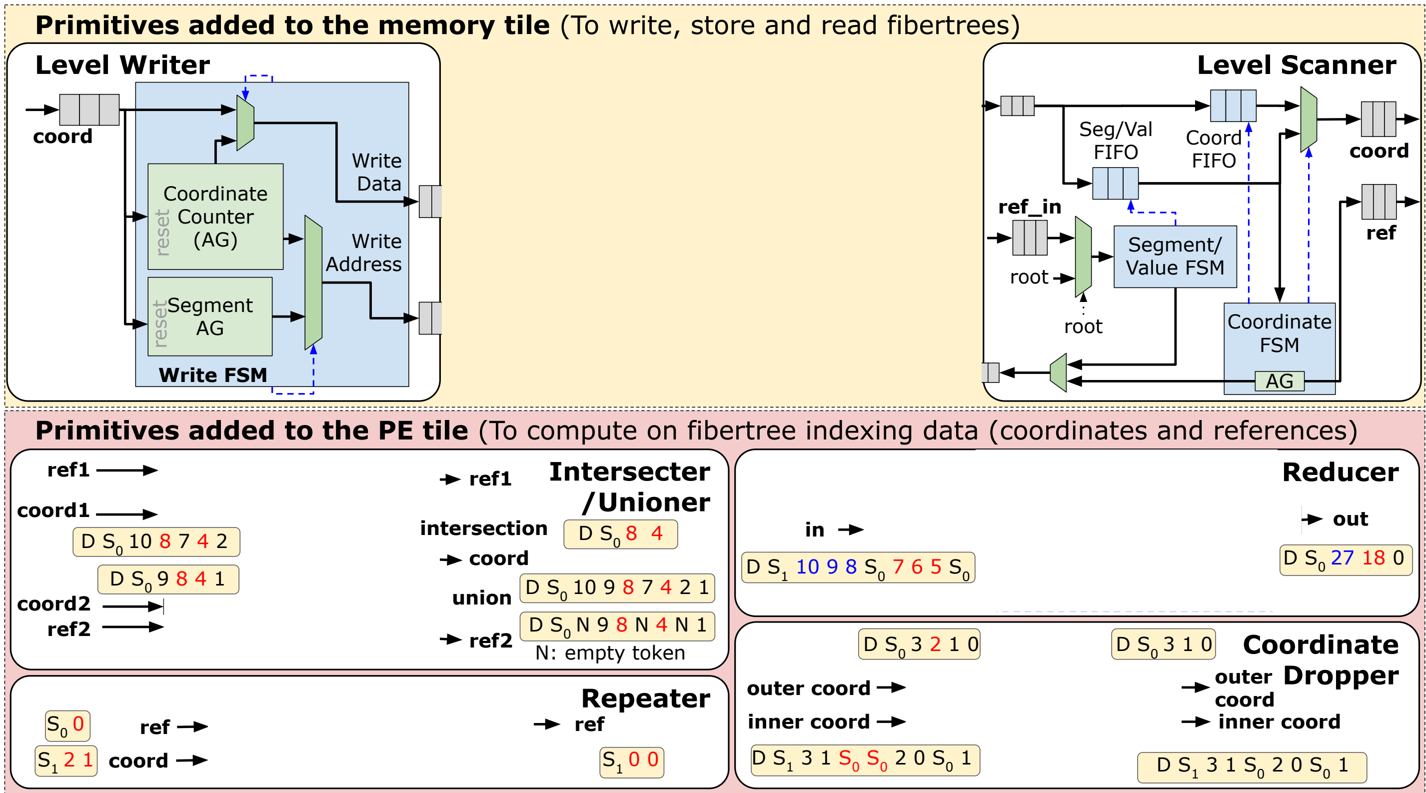
SAM as the architectural specification of our sparse CGRA fabric



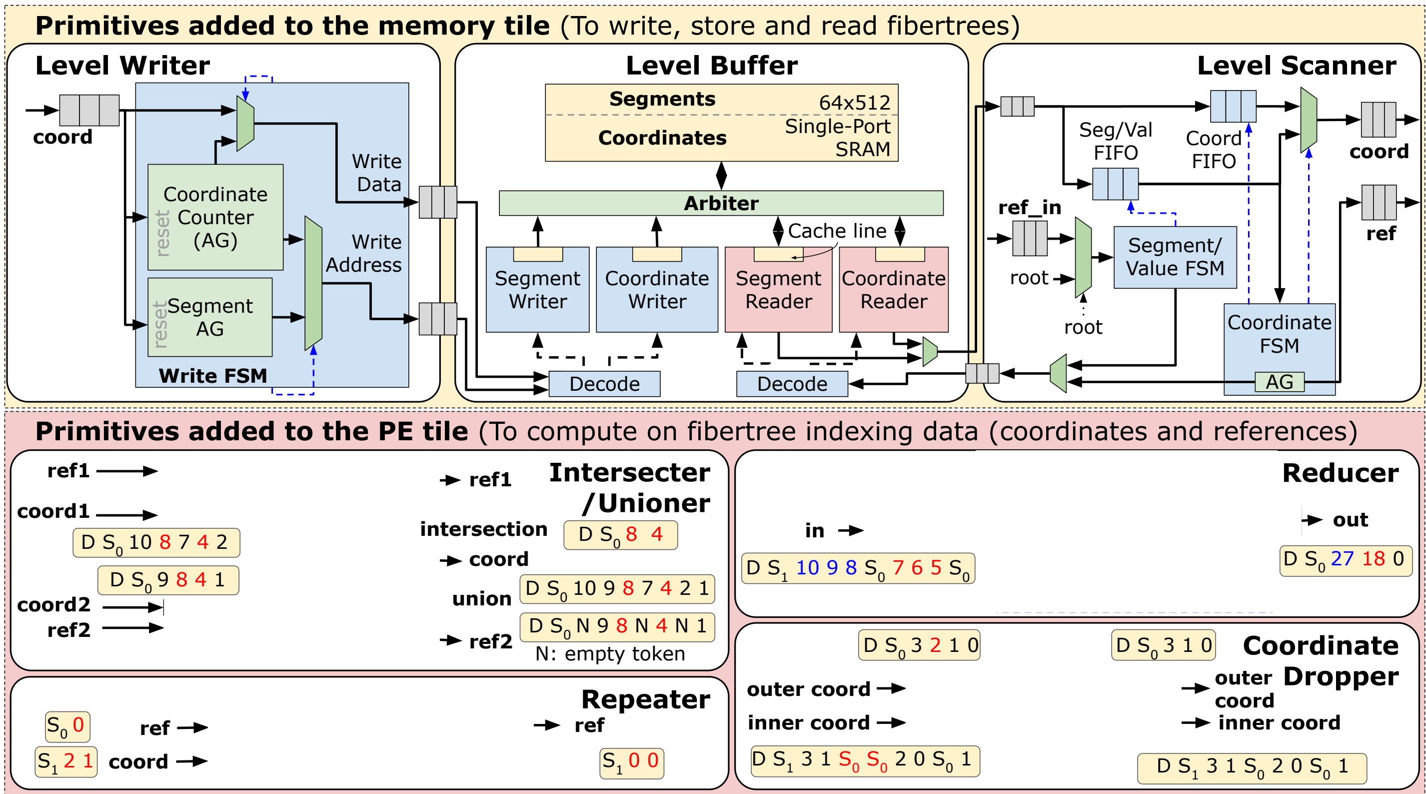
SAM as the architectural specification of our sparse CGRA fabric



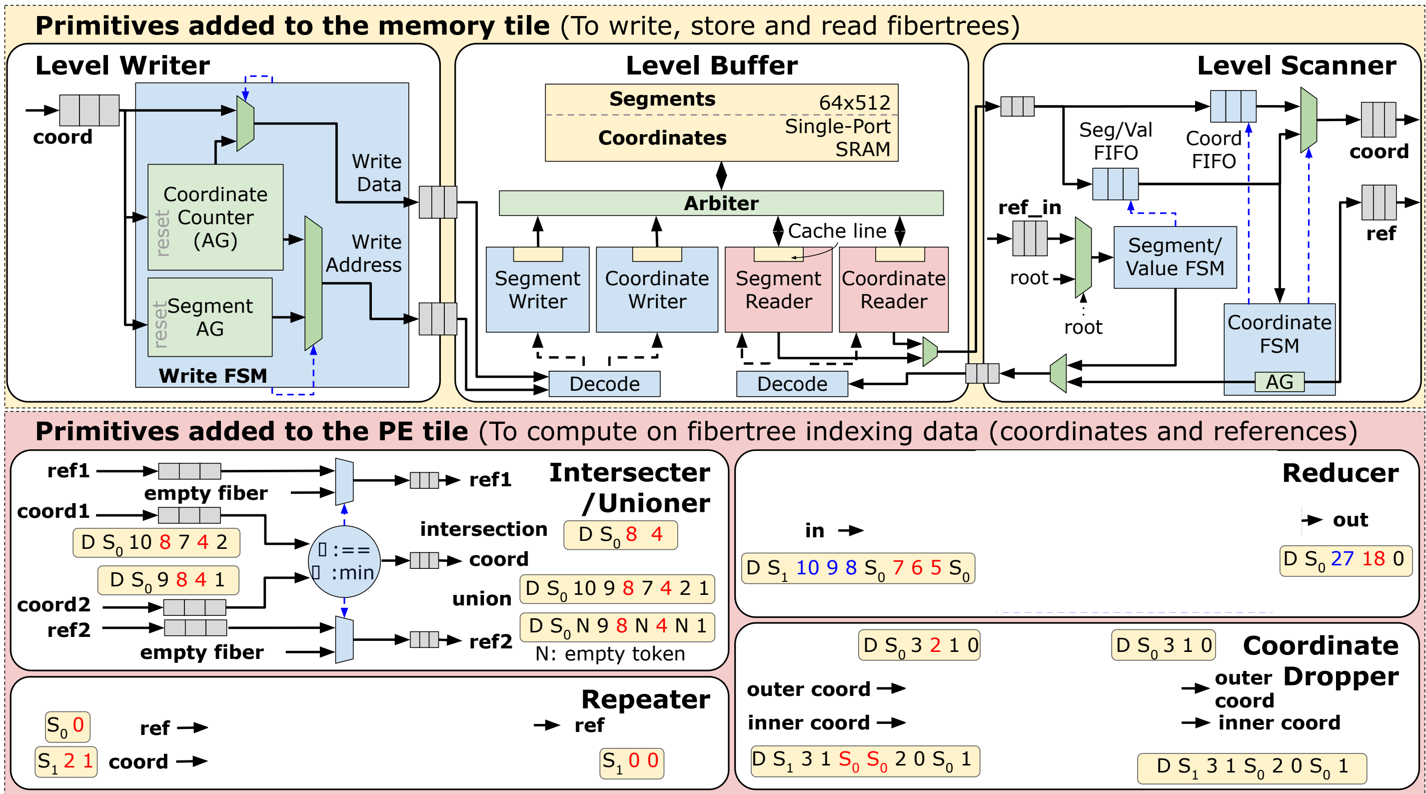
SAM as the architectural specification of our sparse CGRA fabric



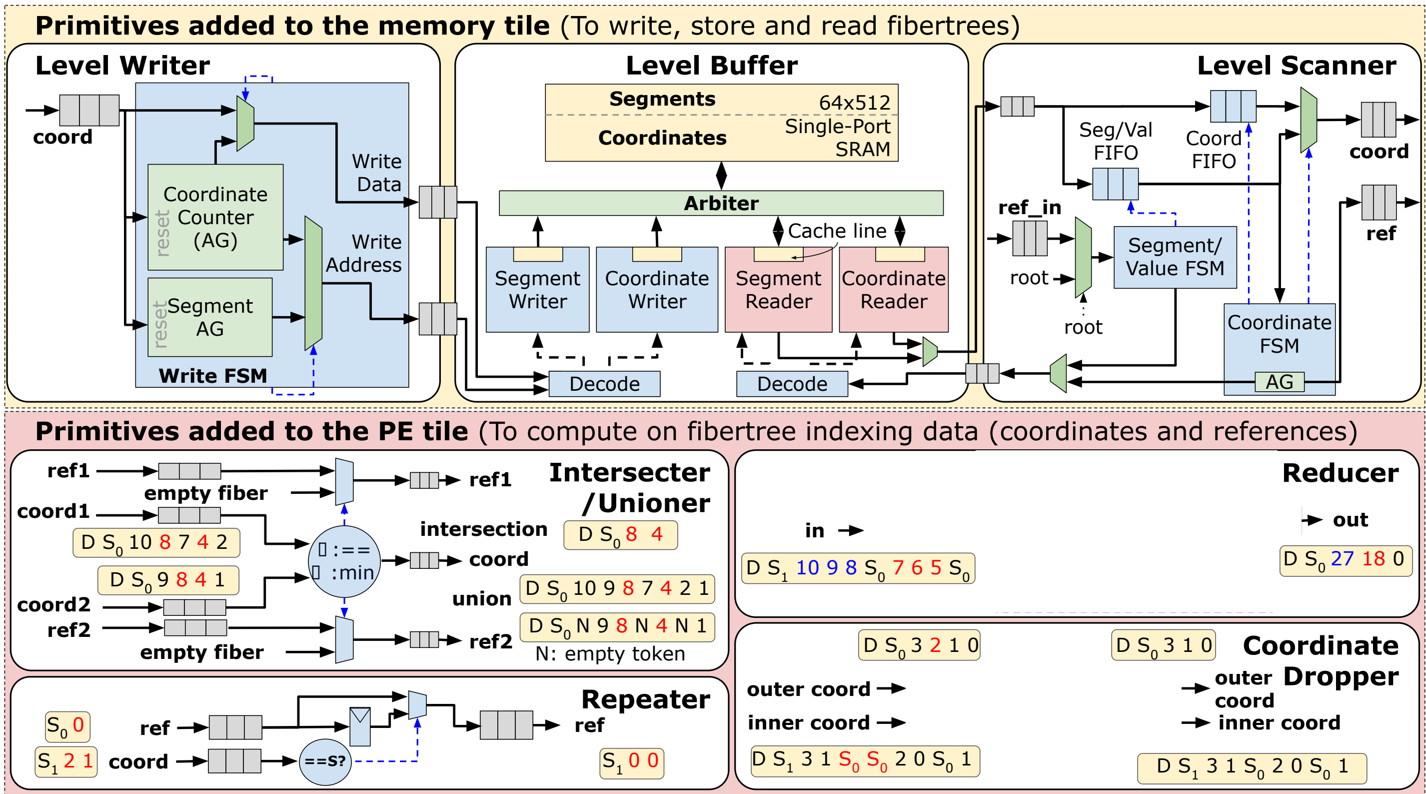
SAM as the architectural specification of our sparse CGRA fabric



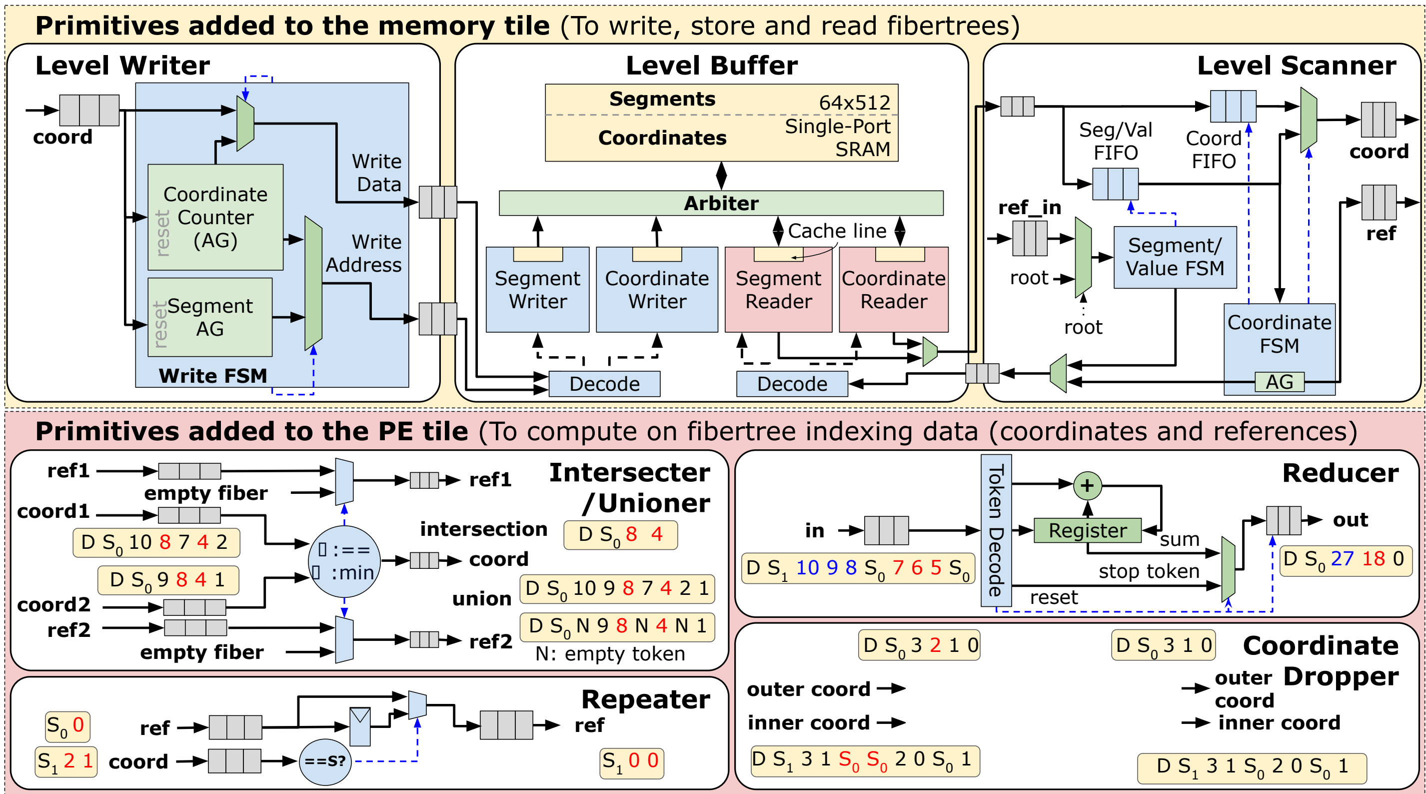
SAM as the architectural specification of our sparse CGRA fabric



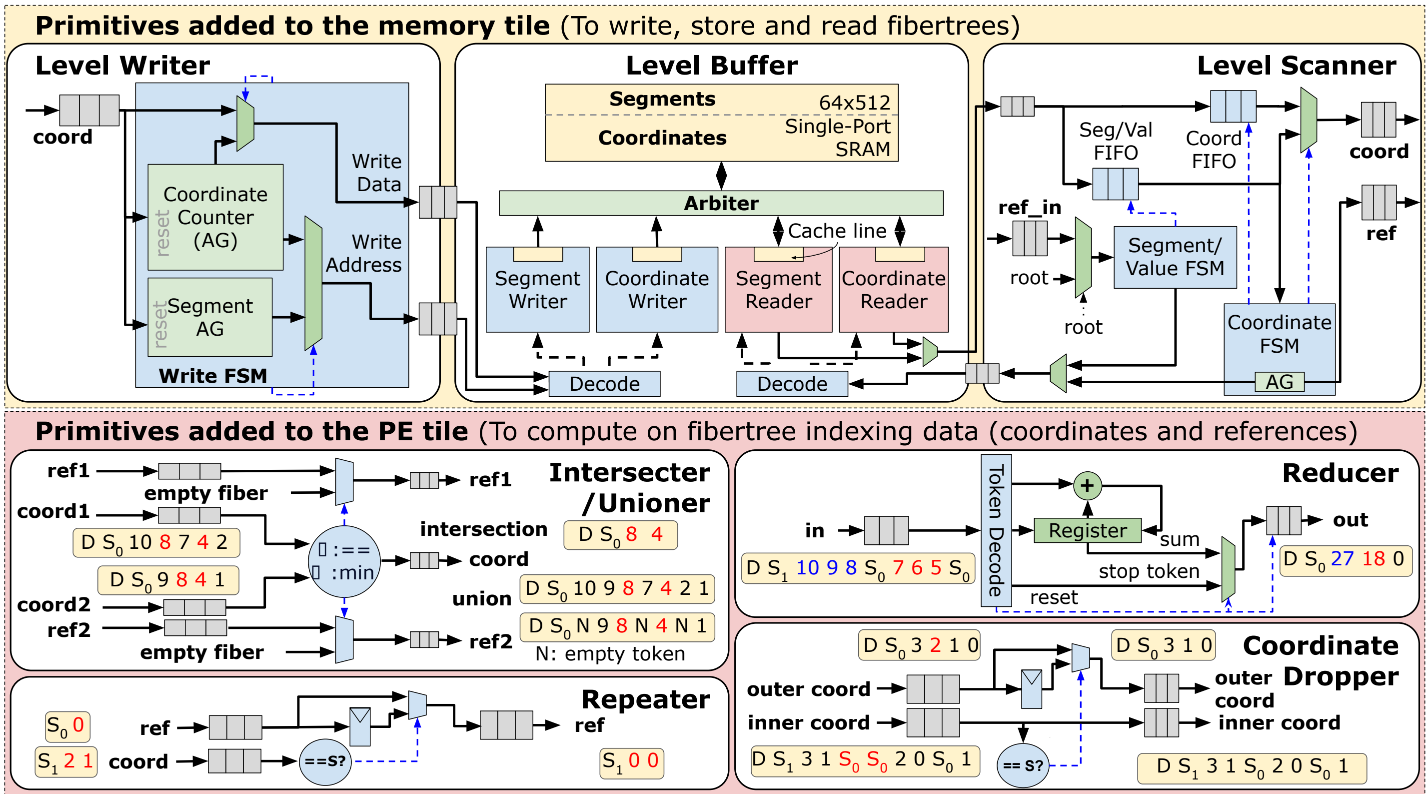
SAM as the architectural specification of our sparse CGRA fabric



SAM as the architectural specification of our sparse CGRA fabric



SAM as the architectural specification of our sparse CGRA fabric



Hardware-aware sparse dataflow graph

- CGRA architecture and microarchitecture requires more transformations during binding
- Introduce the concept of a hardware-aware SAM graph
- Performs transformations like:
  - Broadcast removal
  - Decomposition of N-joiners to binary joiners
  - Merges Level Scanners and Level Writers
  - Inserts Level Buffers

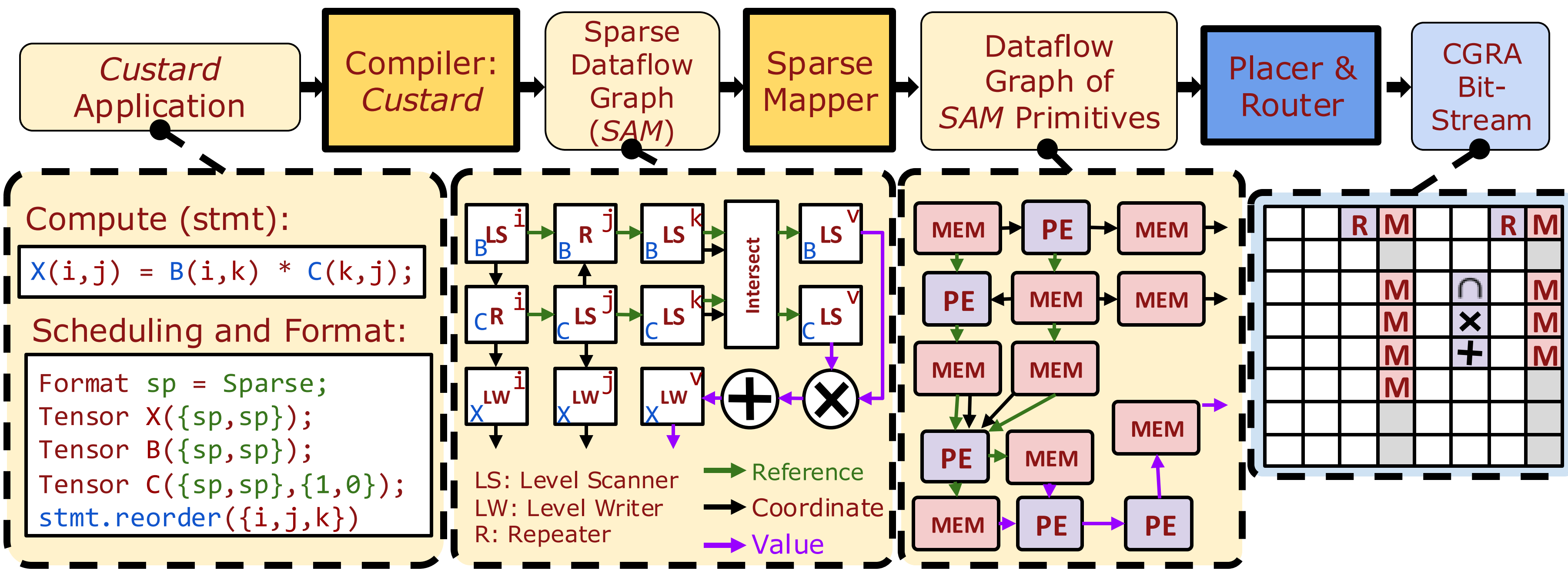
Demo: Mapping to CGRA Microarchitecture

`>./sparse_demo.sh lower`

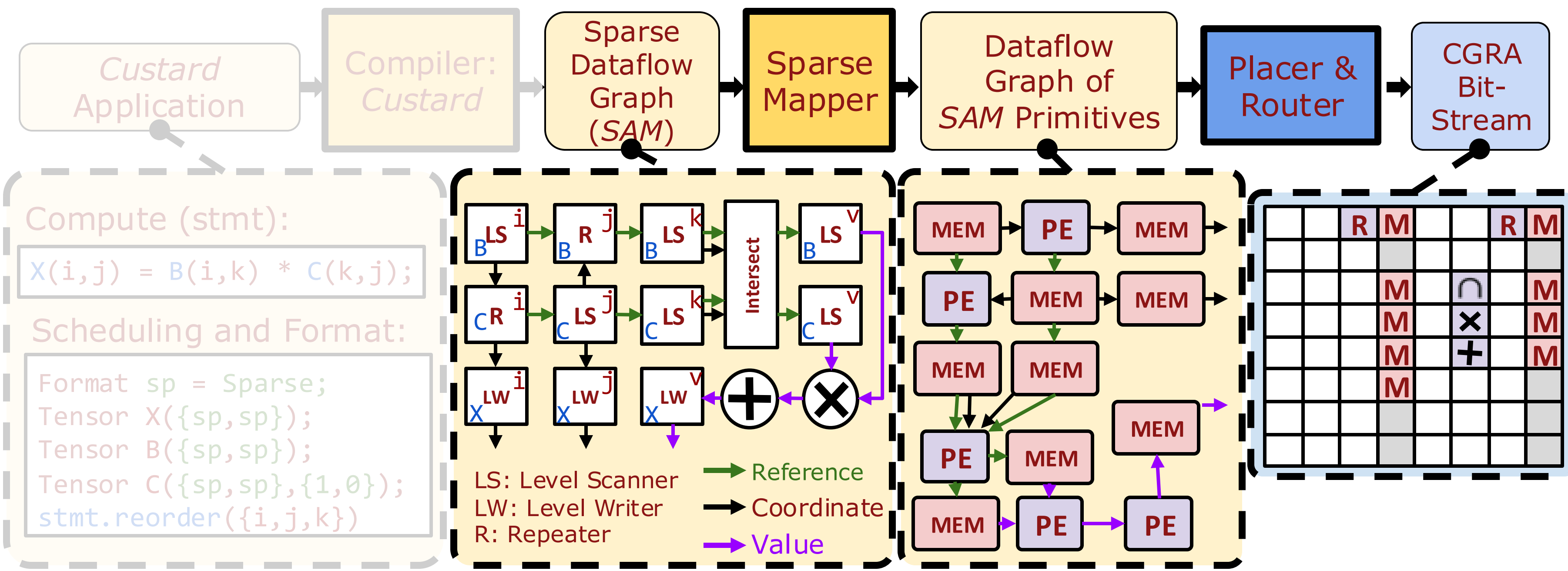
- This runs the SAM graph through the lowering process to produce a hardware-aware sparse dataflow graph

- We can visualize that graph in `/aha/sam/hw_aware_mat_elemadd.png`

Tool flow that maps SAM to a CGRA



Tool flow that maps SAM to a CGRA



Demo: Generating CGRA Bitstream for sparse applications

Run the following command:

```
> ./sparse_demo.sh gen
```

- This generates a CGRA bitstream from the hardware-aware graph using tools introduced later
- It also generates a testbench that runs an example matrix through, checking it with gold (written in Numpy)
- Explore output files generated in `/aha/garnet/SPARSE_TESTS/mat_elemadd_0/`

- Dataflow hardware, like CGRAs, can speed up sparse computation
- Presented ideas from the Sparse Abstract Machine and Onyx
  - SAM is an abstract IR that represents sparse tensor algebra as dataflow graphs
  - SAM comes with the Custard front-end compiler
- Introduced the AHA flow for sparse applications

Conclusion