Sparse Compiler Design

Olivia Hsu

In collaboration with Max Strange, Jaeyeon Won, Kunle Olukotun, Mark Horowitz, Joel Emer, and Fredrik Kjolstad

September 16th, 2021

Motivation

Too many tensor kernels for fixed-function libraries and backends

Linear Algebra

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

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

$C = \sum_{ijkl} M_{ij} P_{jk} \overline{M_{lk}}\, \overline{P_{il}}$ $\tau = \sum_i z_i (\sum_j z_j \theta_{ij})(\sum_k z_k \theta_{ik})$ $a = \sum_{ijklmnop} M_{ij} P_{jk} M_{kl} P_{lm} \overline{M_{nm}} P_{no} \overline{M_{po}}\, \overline{P_{ip}}$

×

Dense Matrix

CSR DCSR BCSR

COO ELLPACK CSB

Blocked COO CSC

DIA Blocked DIA DCSC

Sparse vector Hash Maps

Coordinates

CSF Dense Tensors

Blocked Tensors

Linked Lists Database

Compression Schemes

Cloud Storage

×

CPU

GPUs TPUs

FPGA CGRAs

Sparse Tensor Hardware

Cloud Computers

Supercomputers

×

reorder

precompute

parallelize split

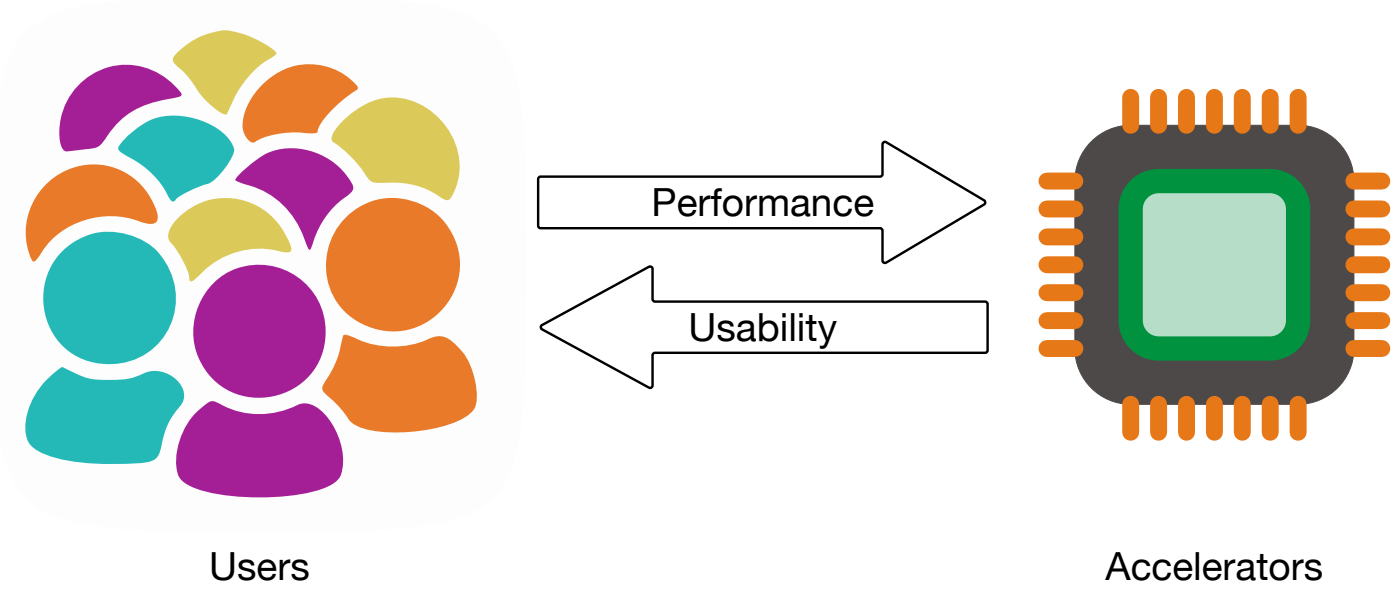
map divide

vectorize unroll

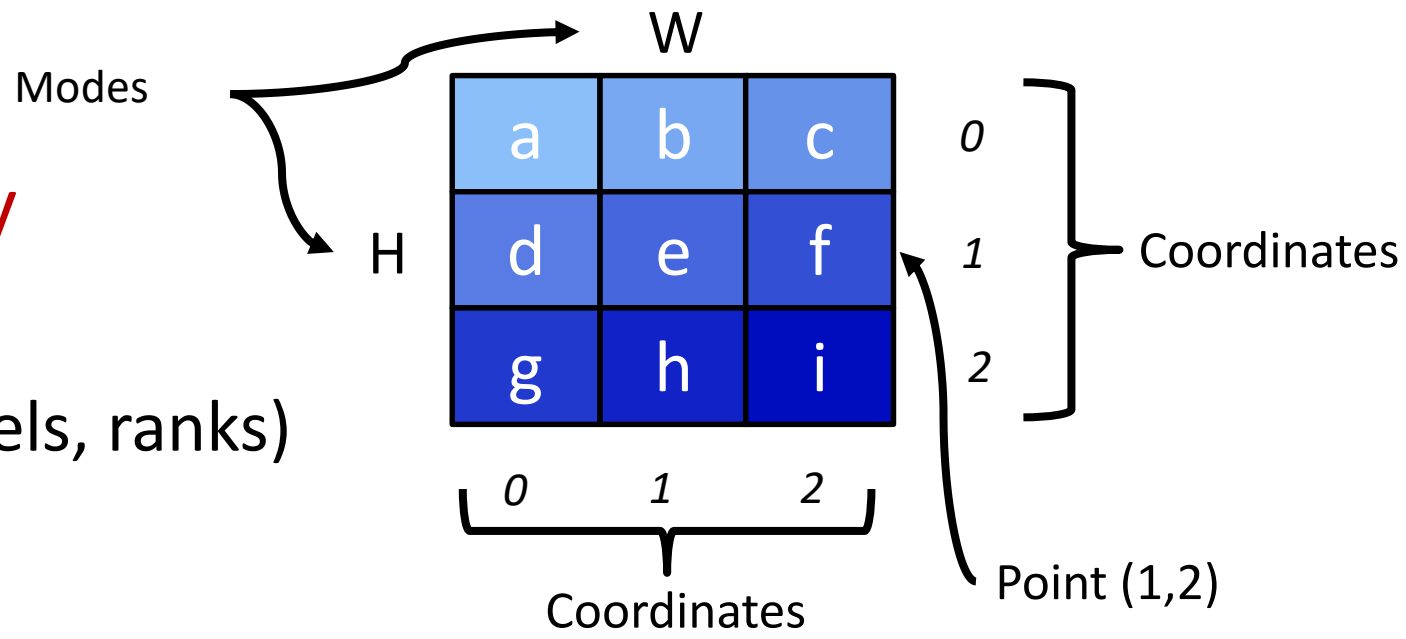
position

Benefits

Long Tail of Expressions (algorithm) × Varying Compression Structures (format) × More Performant Backends (platform) × Backend-Specific Transformations (schedules)



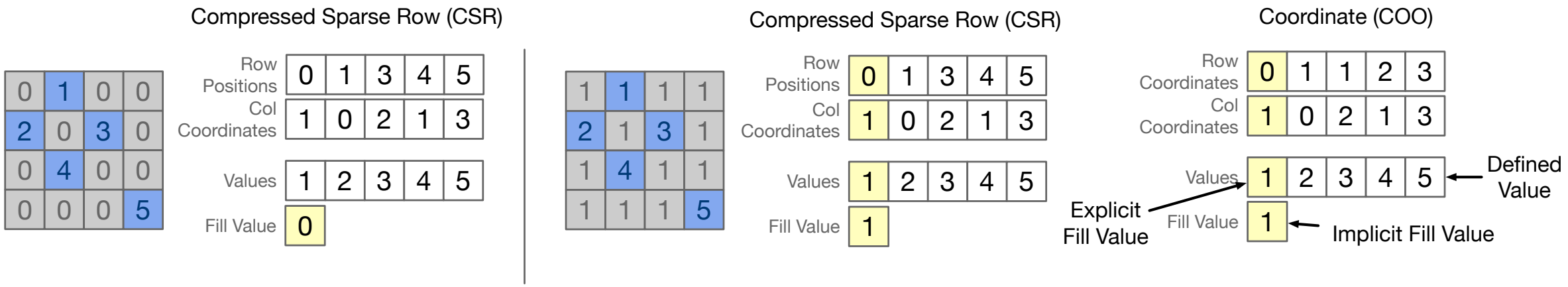
Background: Tensor Terminology



- ***Modes*** (aka dimensions, levels, ranks)
  - The order refers to the number of modes
    - Ex. Matrices are 2nd order
- Each component of a rank is identified by a ***coordinate***
  - Represents the ***data address space***
- Each value element in a tensor is referred to as a "***point***"
  - Points are identified by an ***ordered list*** of coordinates for each rank

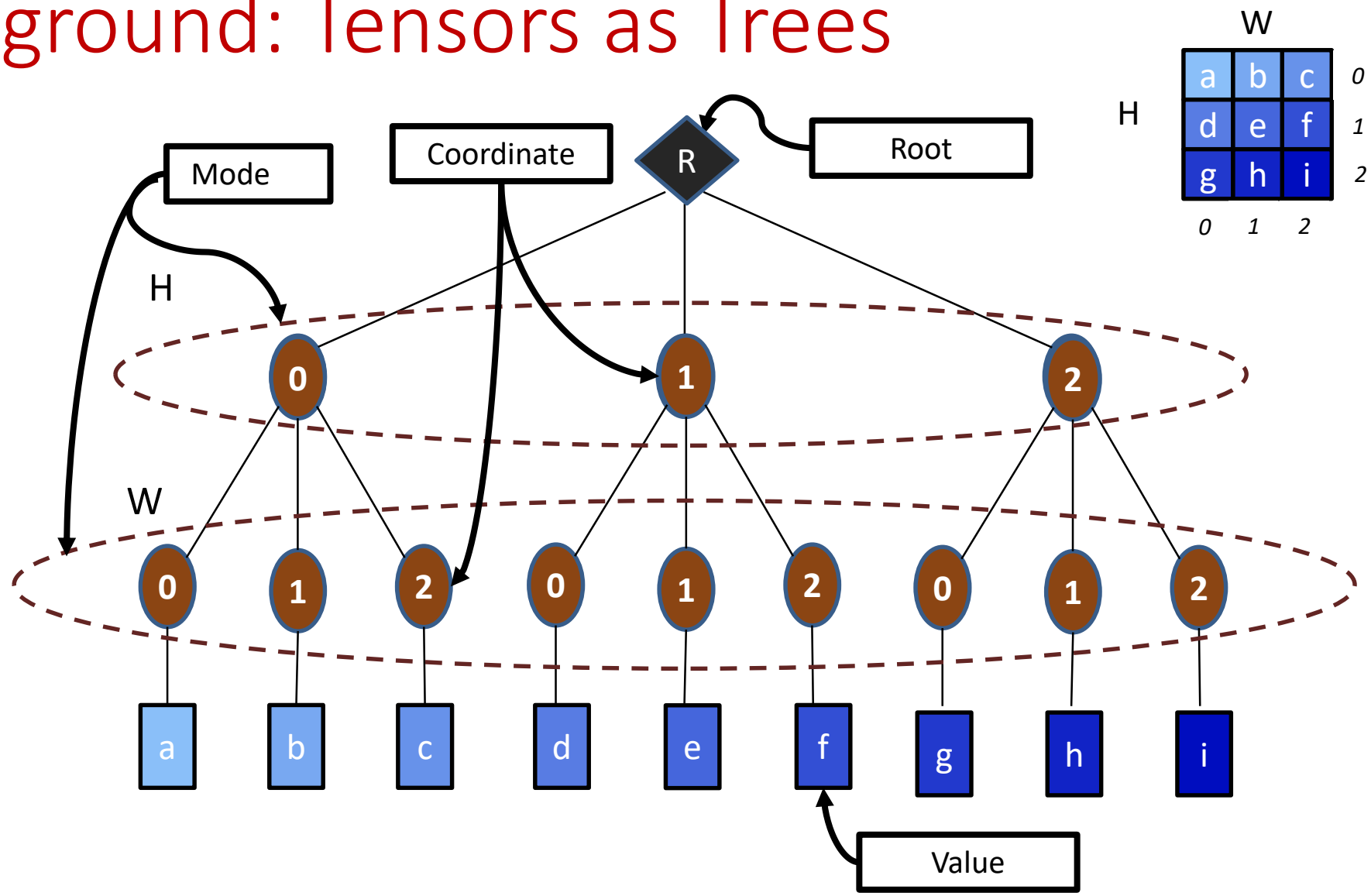
Picture Courtesy of Joel Emer, MIT, NVIDIA

Background: Compression



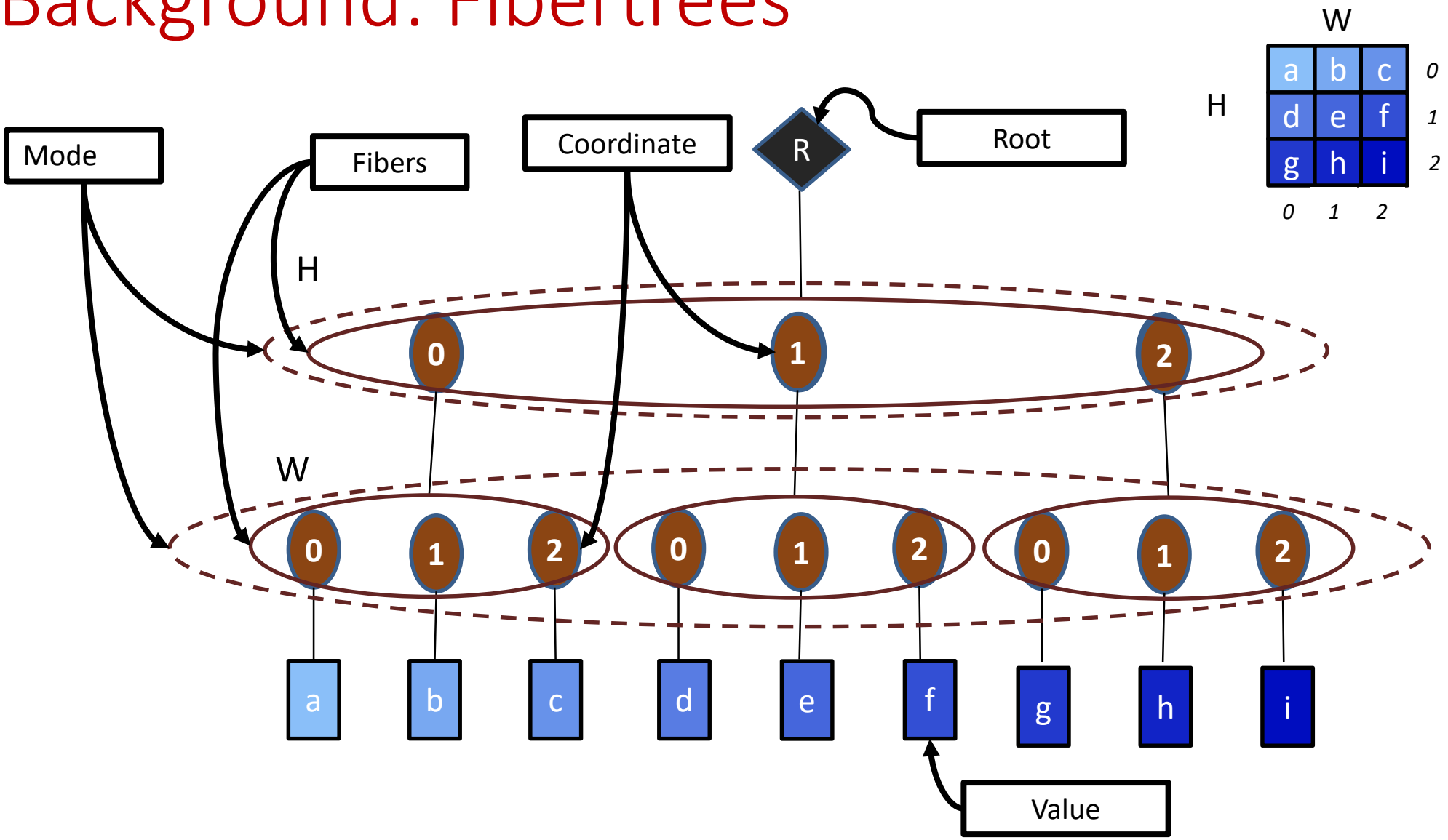
- Each rank (level) has a format attribute that are are ***data structure, representation, and format*** dependent: compressed, uncompressed, etc.
- Uncompressed means the occupancy is ***proportional*** to the max coordinate
- Compressed data has ***metadata overhead*** required to define the scheme
  - Data in a compression array is accessed by ***position***, the ***memory address space***
- ***Sparsity*** refers to the ***data pattern***

Background: Tensors as Trees



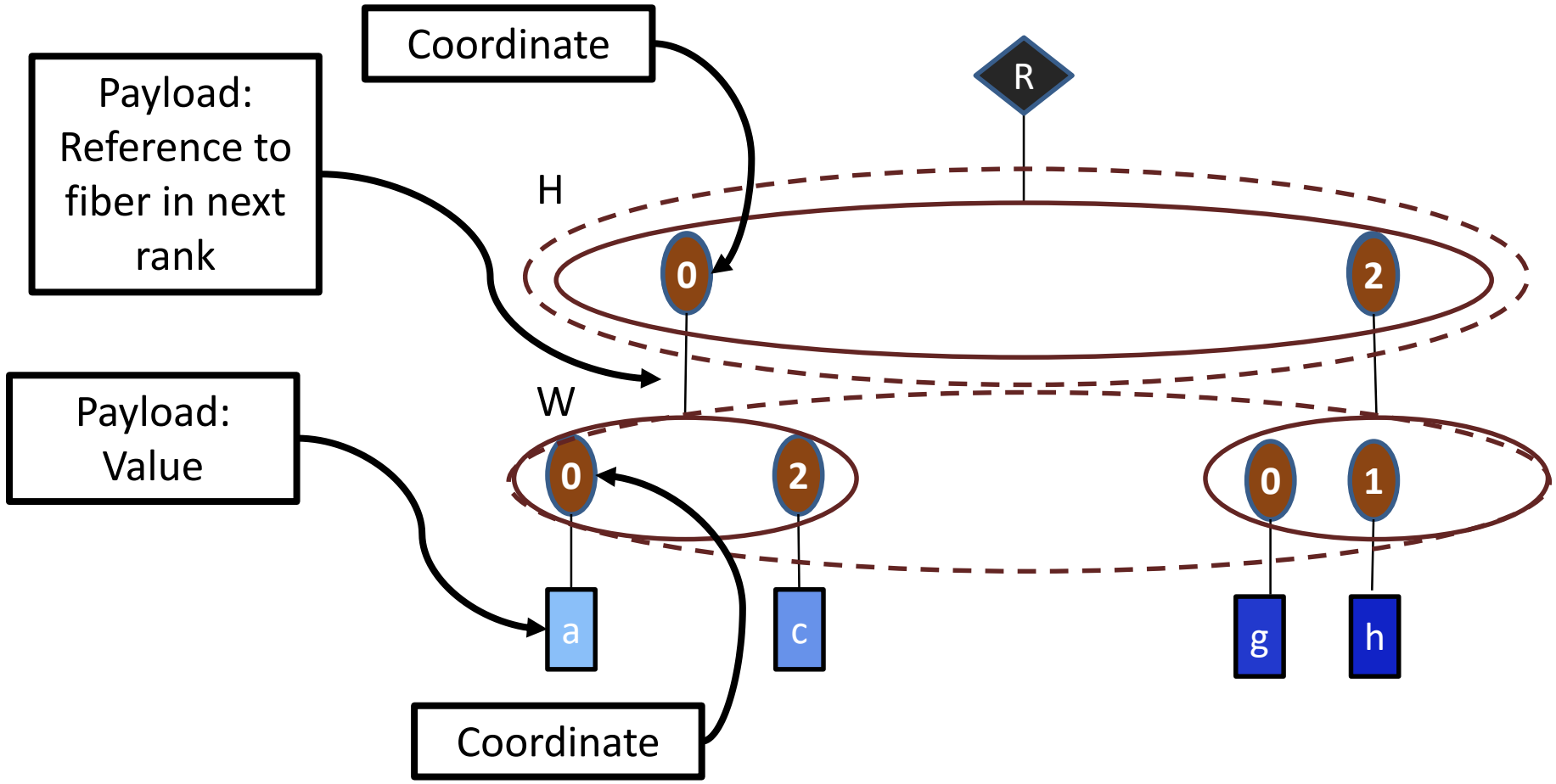
Picture Courtesy of Joel Emer, MIT, NVIDIA

Background: Fibertrees



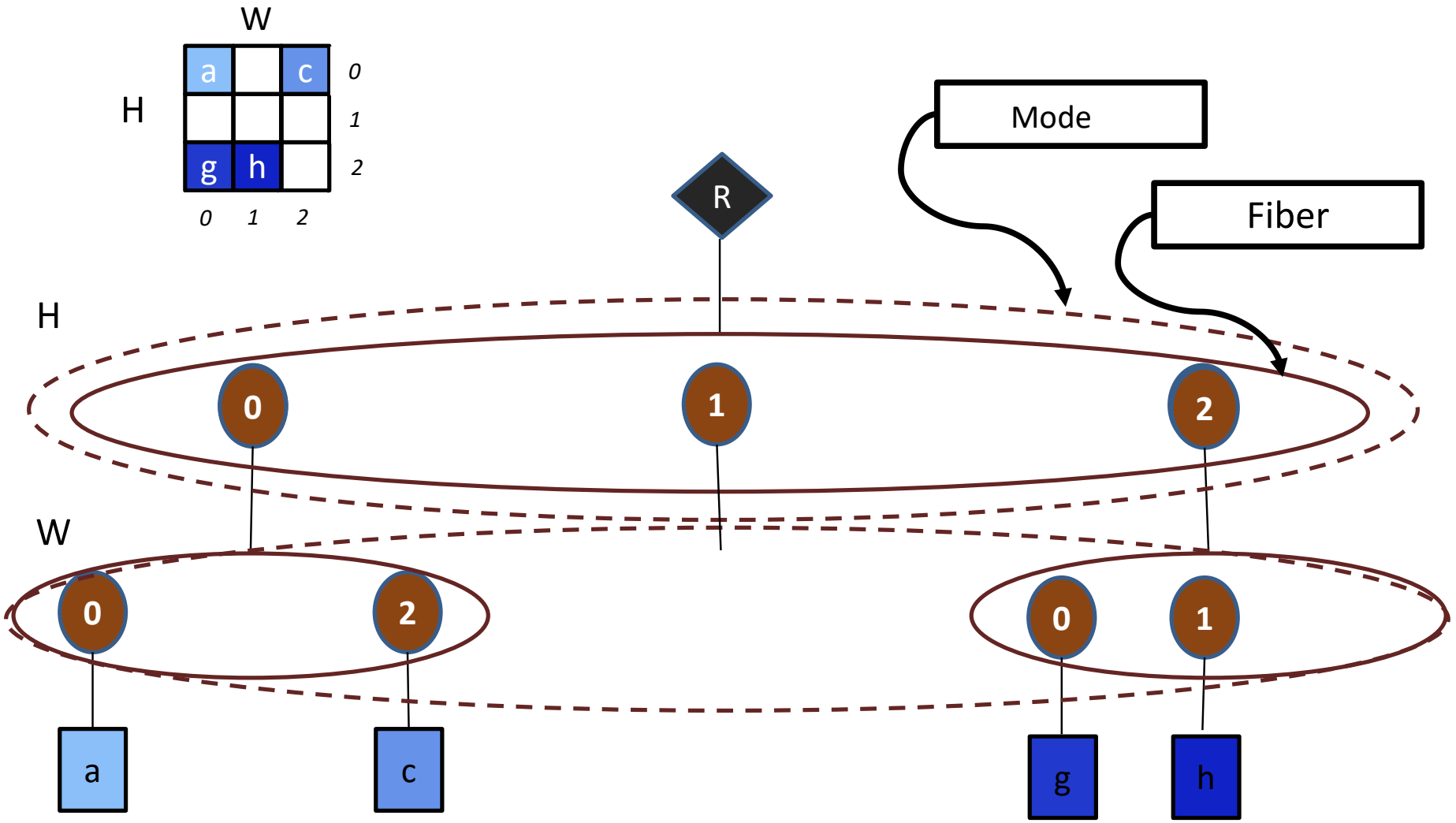
Picture Courtesy of Joel Emer, MIT, NVIDIA

Background: Fiber = (Coordinate, Payload)



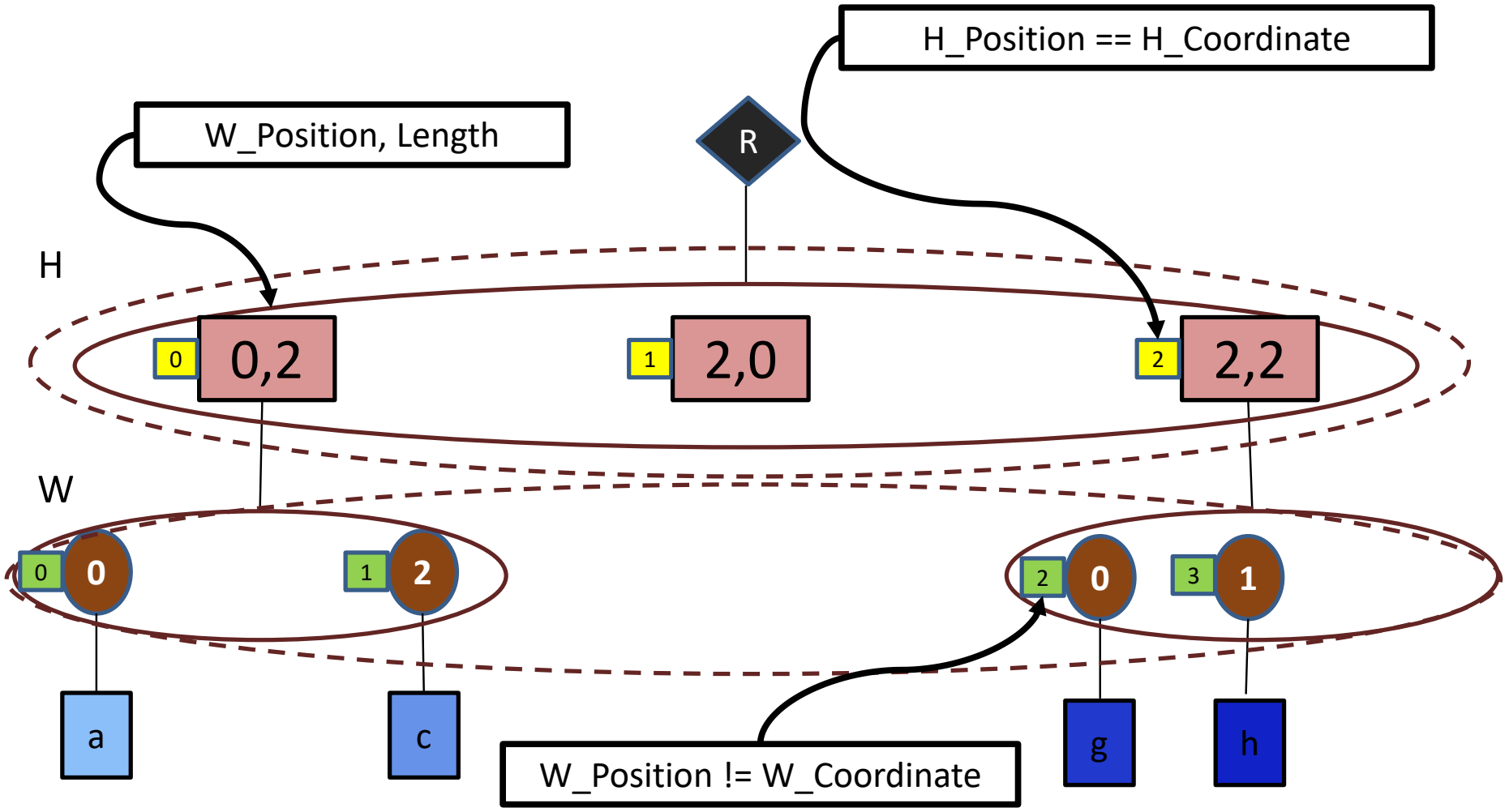
Picture Courtesy of Joel Emer, MIT, NVIDIA

Background: <Uncompressed, Compressed>



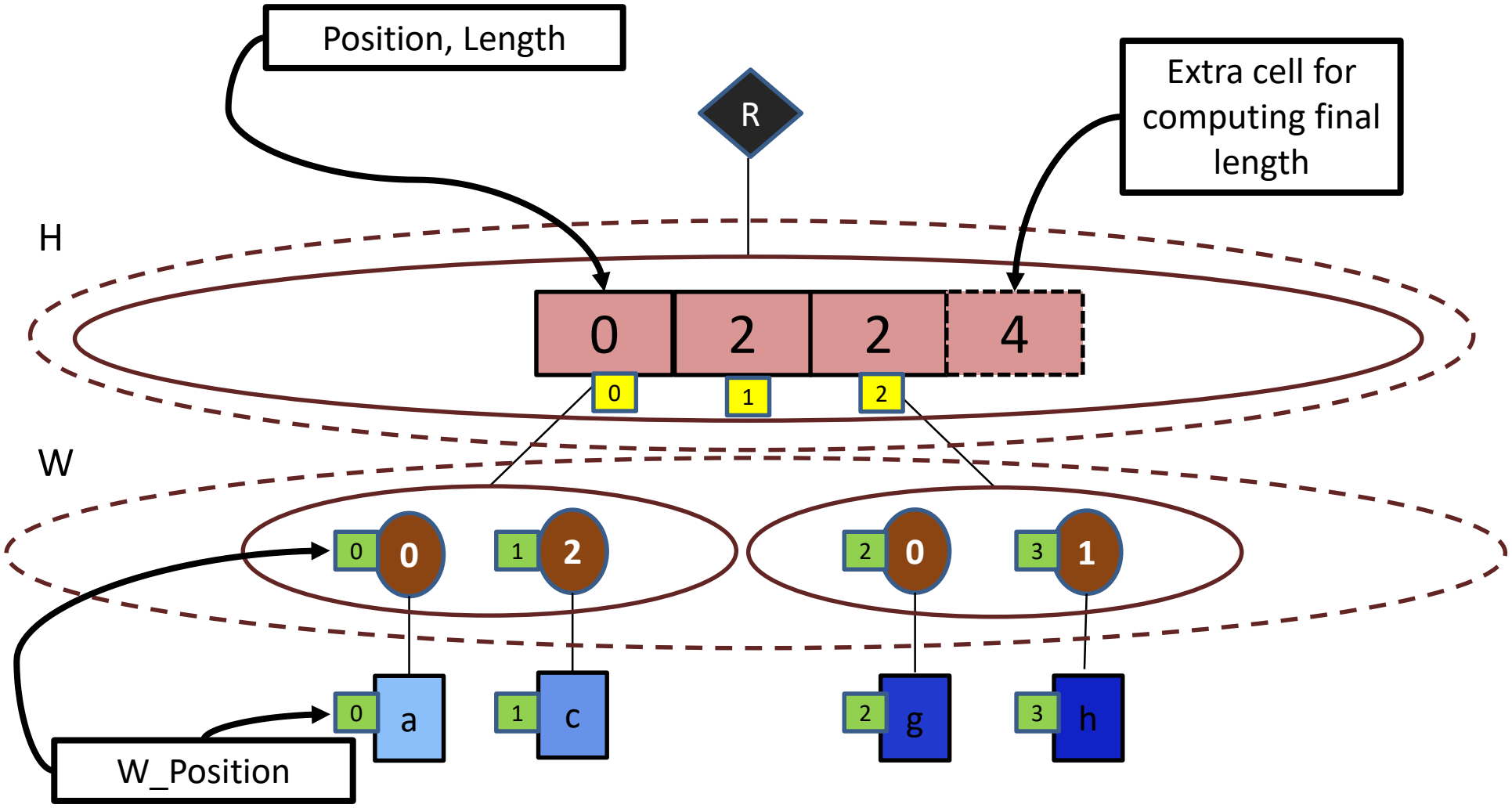
Picture Courtesy of Joel Emer, MIT, NVIDIA

Background: Uncompressed = (Position, Len)



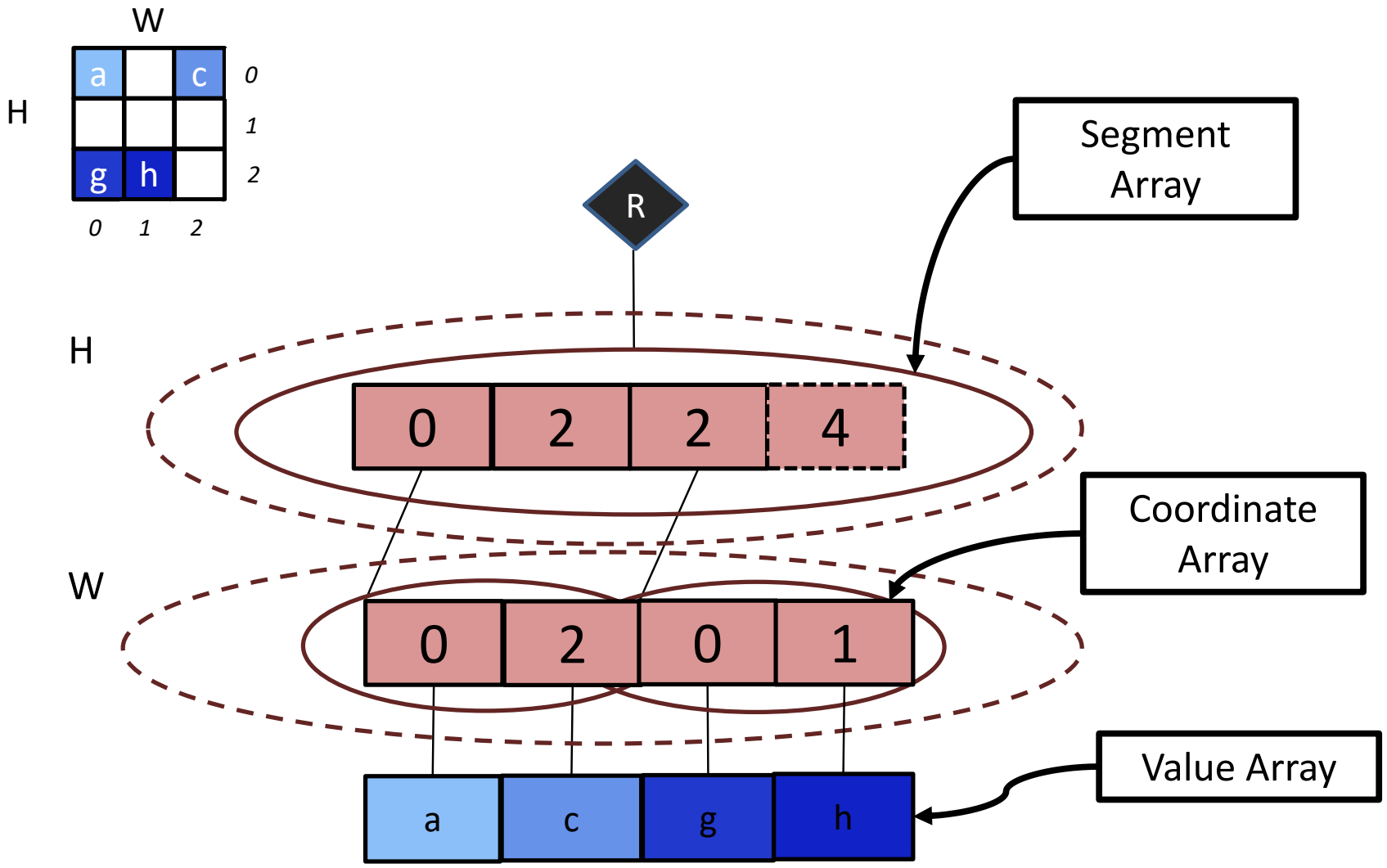
Picture Courtesy of Joel Emer, MIT, NVIDIA

Background: <Position, Len> Array



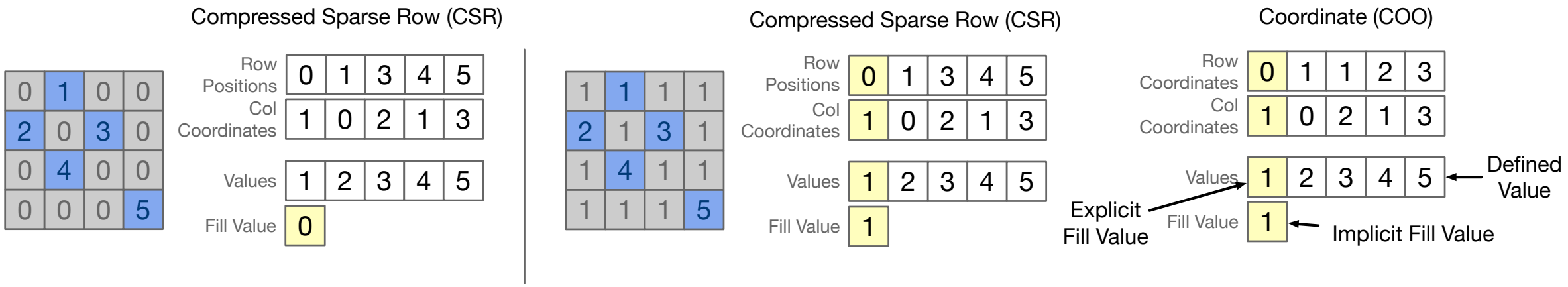
Picture Courtesy of Joel Emer, MIT, NVIDIA

Background: Compressed Sparse Row



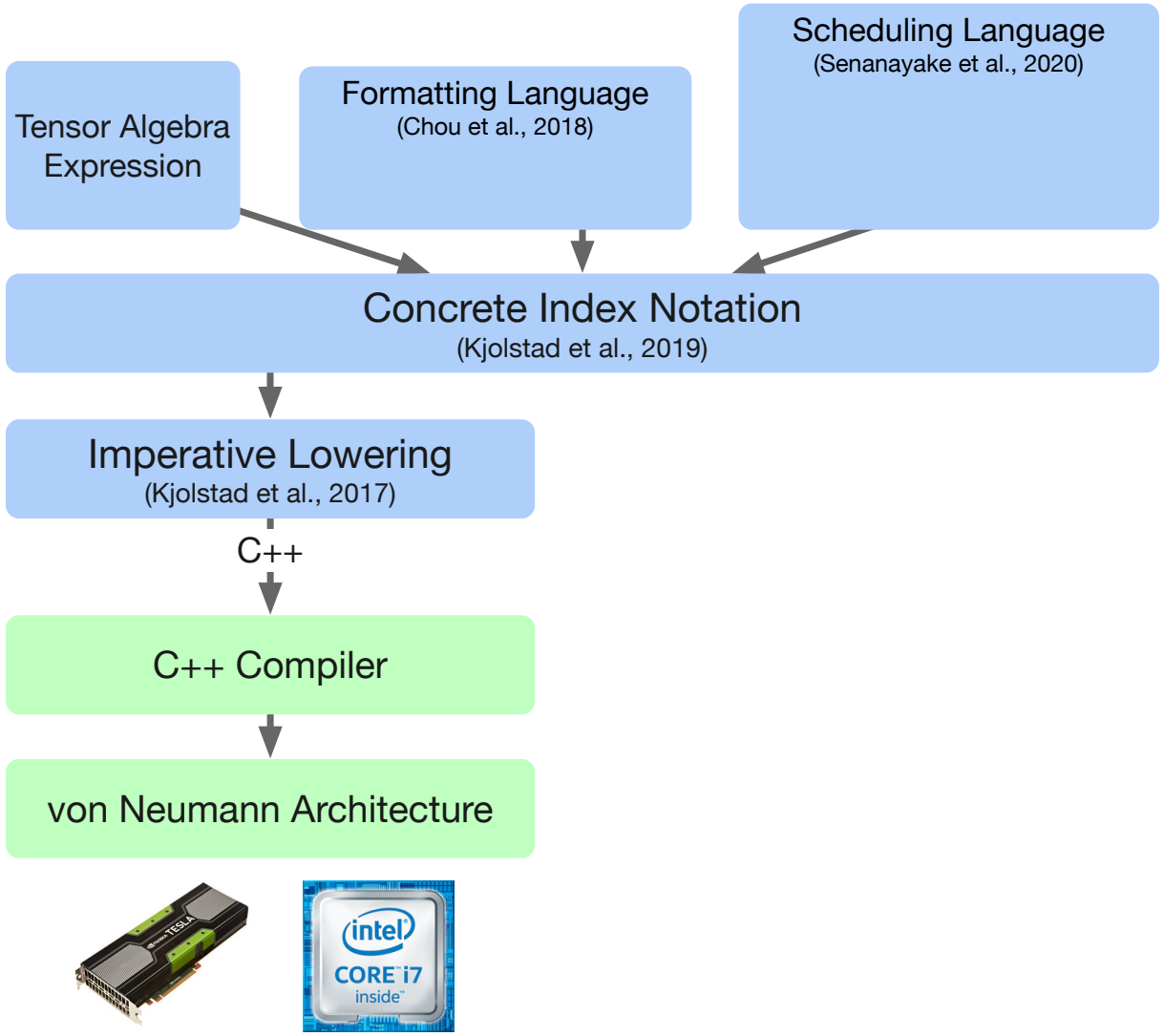
Picture Courtesy of Joel Emer, MIT, NVIDIA

Background: Compression Again



- Each rank (level) has a format attribute that are are ***data structure, representation, and format*** dependent: compressed, uncompressed, etc.
- Uncompressed means the occupancy is ***proportional*** to the max coordinate
- Compressed data has ***metadata overhead*** required to define the scheme
  - Data in a compression array is accessed by ***position***, the ***memory address space***
- ***Sparsity*** refers to the ***data pattern***

TACO Overview



Legend

Other

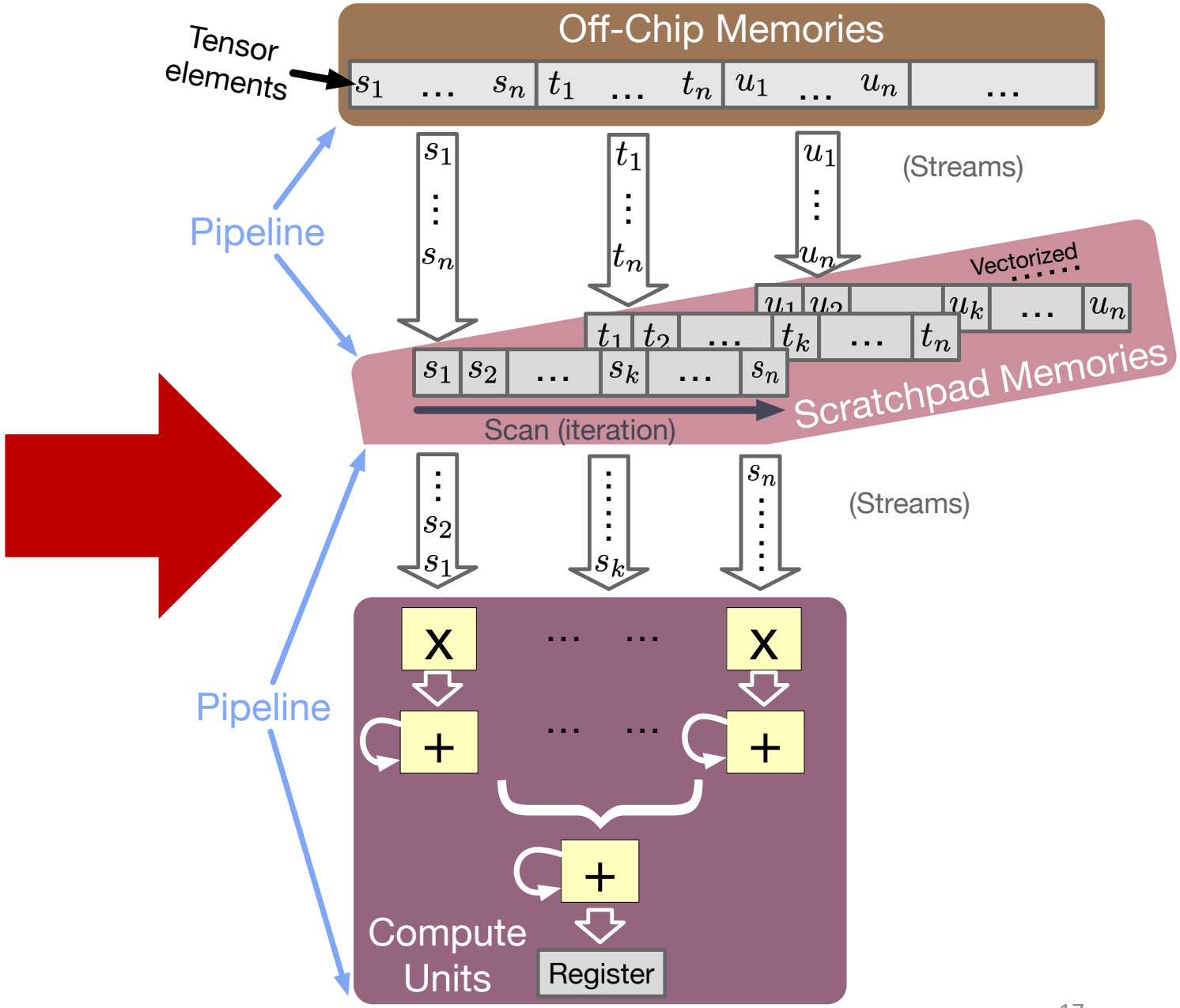
TACO Literature

Main Challenge

Imperative Representation

$$\forall_i (x_i = b_i * c_i)$$

Concrete Index Notation



Goal

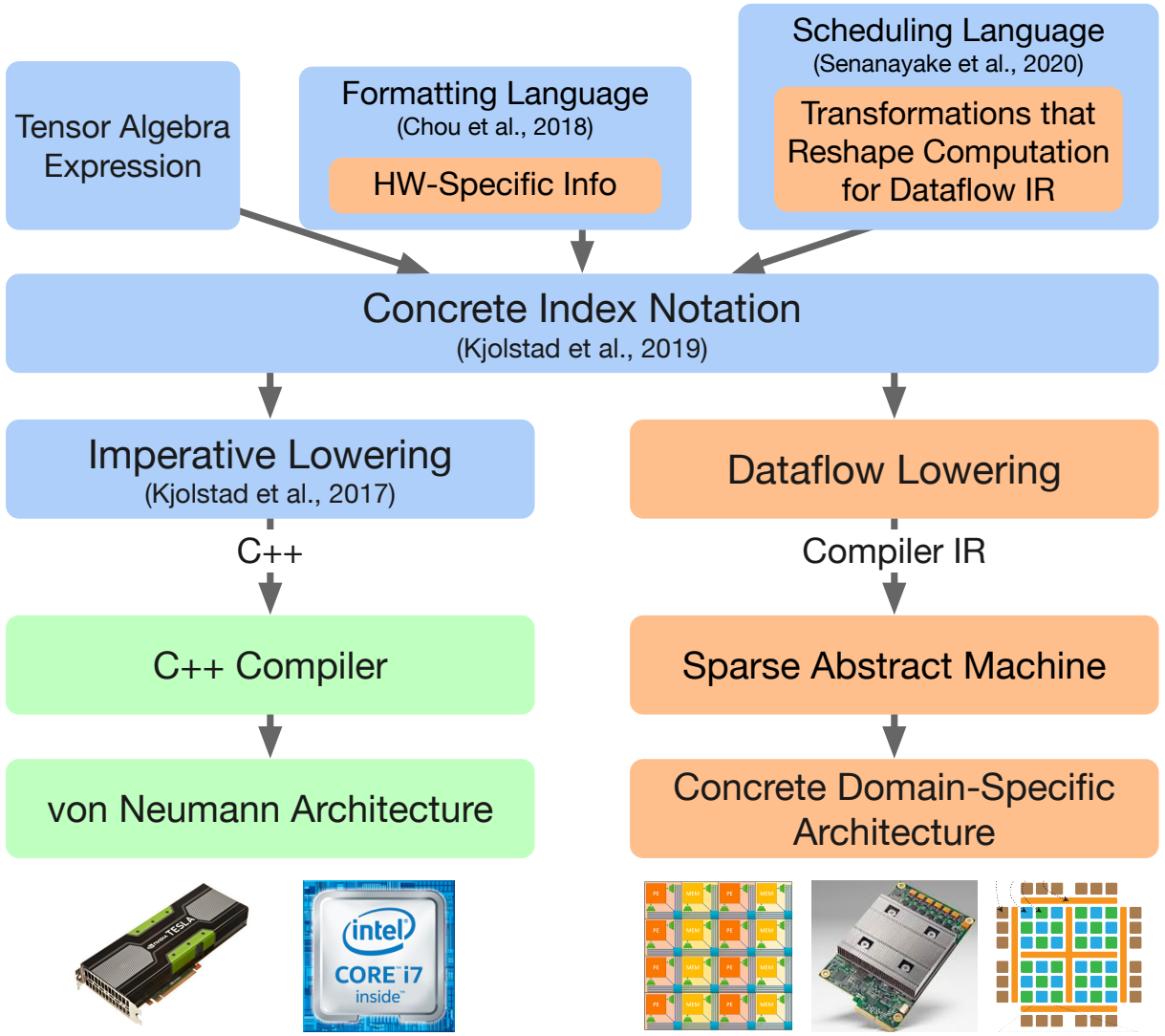
Create a ***logical abstraction*** that can represent ***sparse dataflow*** applications

Attempt to describe the ***entire*** space of sparse tensor algebra kernels spatially

Ultimately, use this representation within the sparse compiler as an ***IR***

Develop ***efficient hardware implementations*** of the logical blocks in conjunction

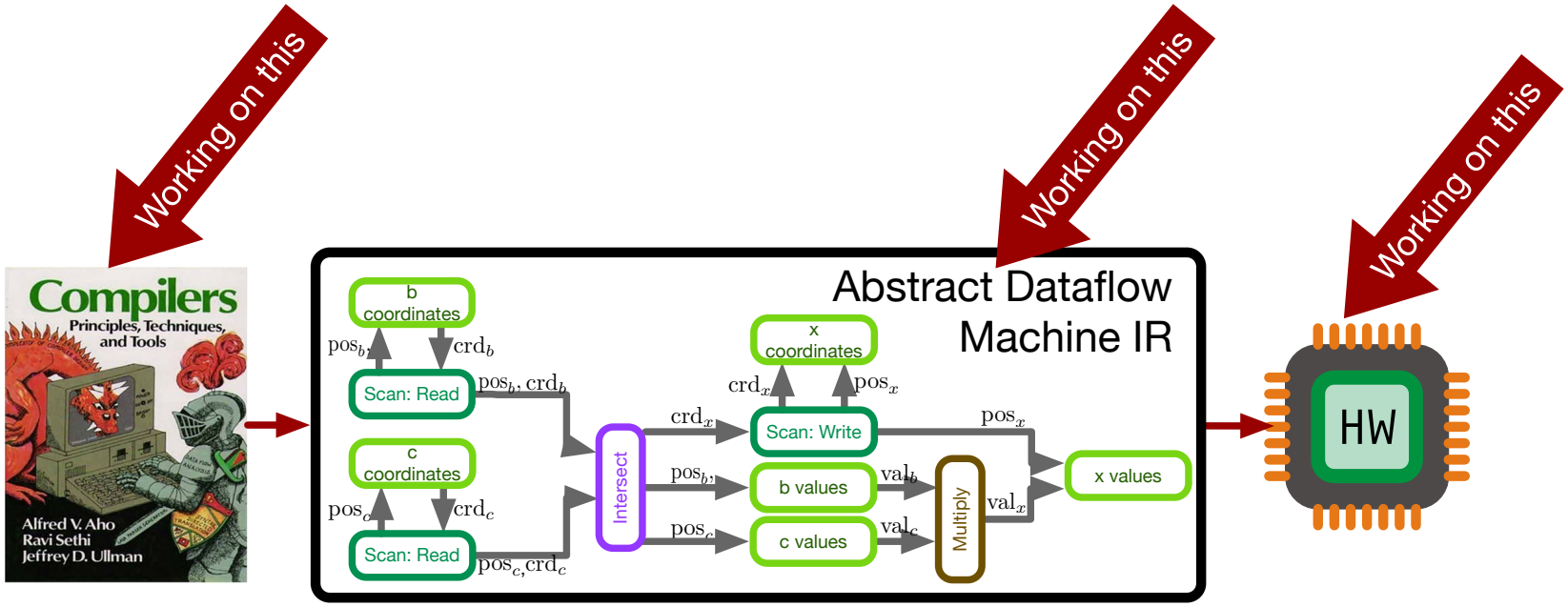
Sparse Dataflow Compiler Overview



Legend

- Sparse Dataflow Compiler
- Other
- TACO Literature

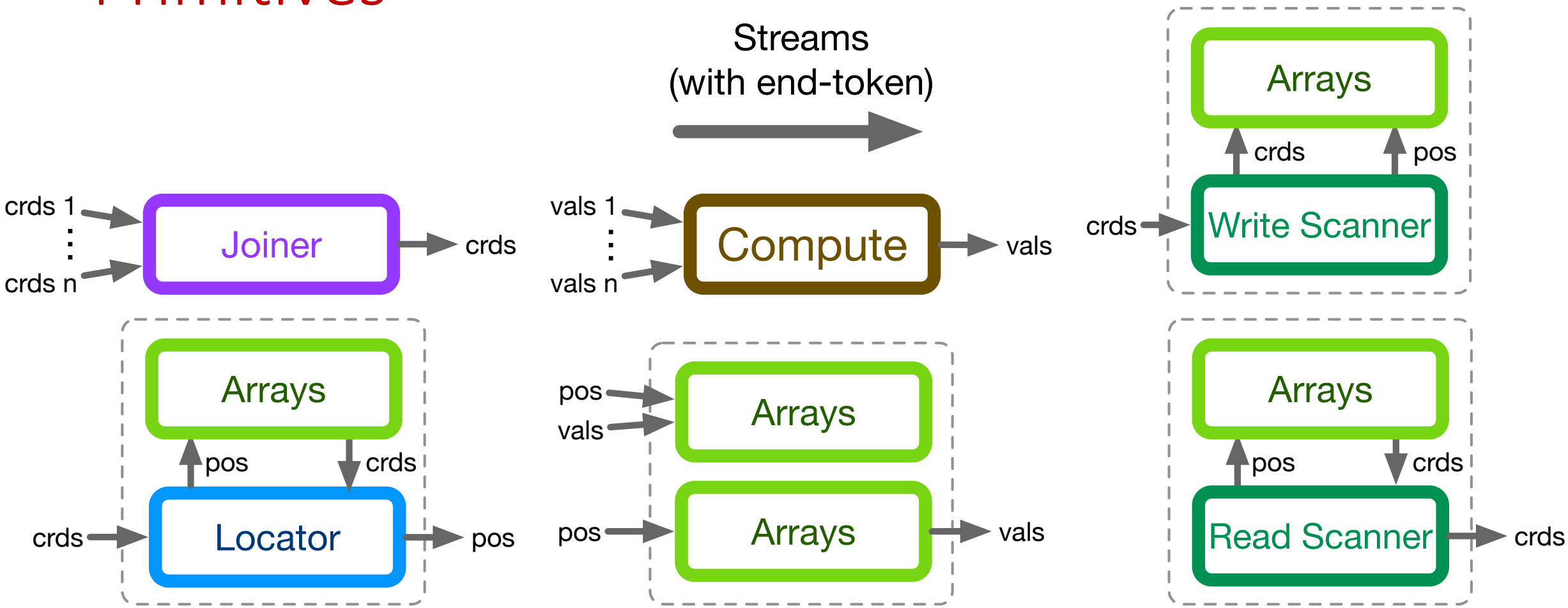
Current Work



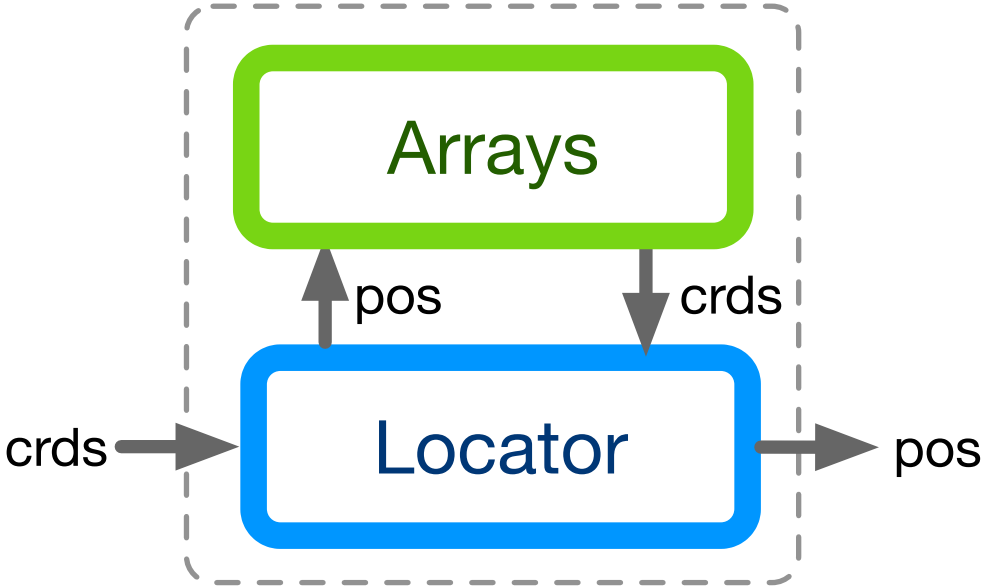
Approach

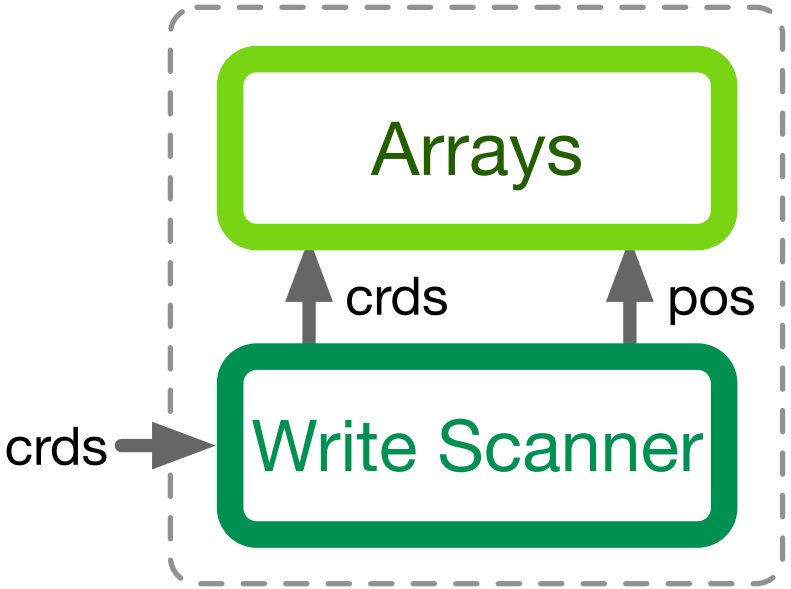
- Draw sparse kernels spatially
- Start Simple, and increase in degrees of freedom
  - Started out with SpV * V
  - Change vector compression formats: SpV * SpV, V * V
  - Change parallelization (x2): SpV * V, SpV * SpV
  - Change operator: SpV + SpV
  - Increase dimensions and add reductions: SpM * V
- Enumerate dataflow abstraction options to make granularity and abstraction decisions

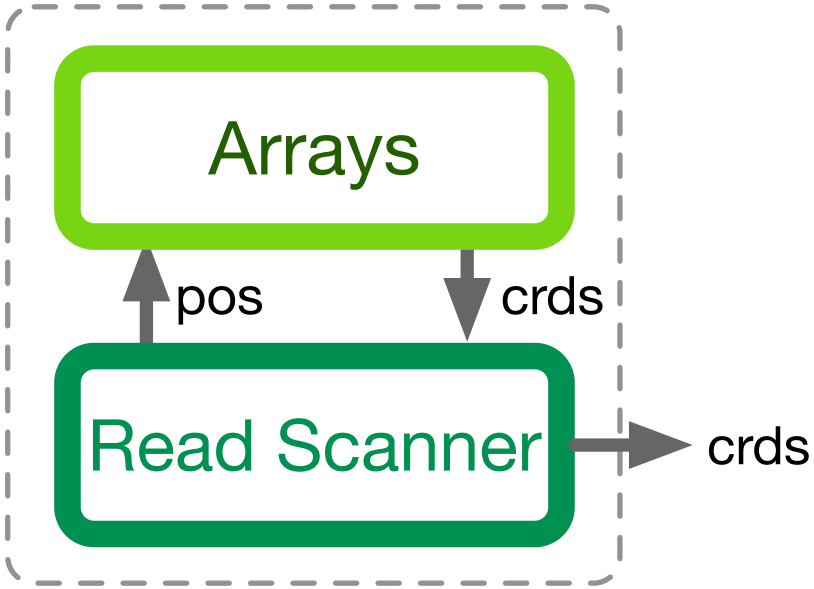
Primitives


Streams
(with end-token)
crds 1
crds n
Joiner
crds
vals 1
vals n
Compute
vals
Arrays
crds
pos
Write Scanner
crds
Arrays
pos
crds
Locator
crds
pos
pos
vals
Arrays
Arrays
pos
vals
Arrays
pos
crds
Read Scanner
crds

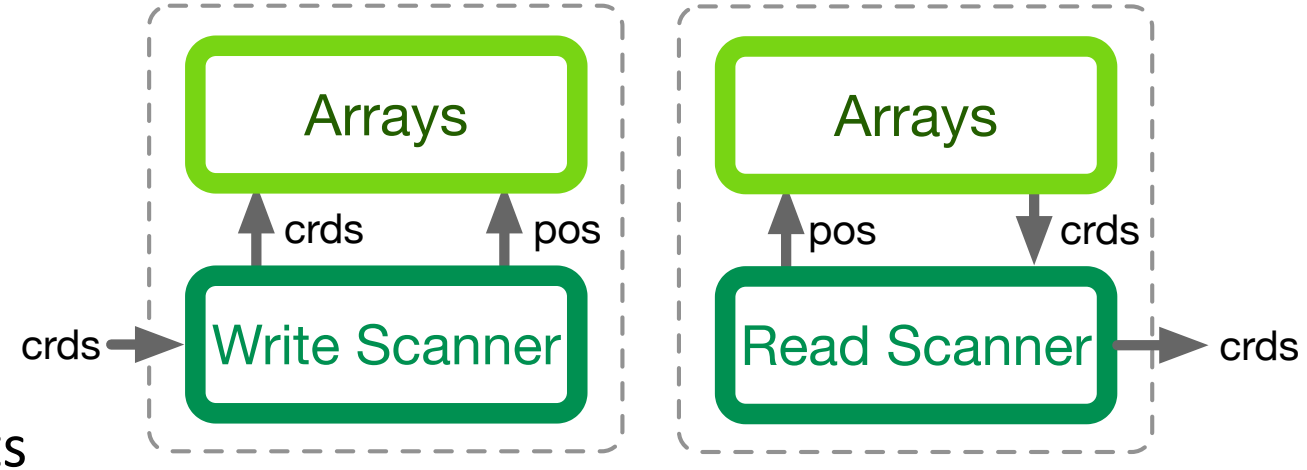
Primitives


Arrays
pos
crds
crds
Locator
pos


Arrays
crds
pos
crds
Write Scanner


Arrays
pos
crds
Read Scanner
crds

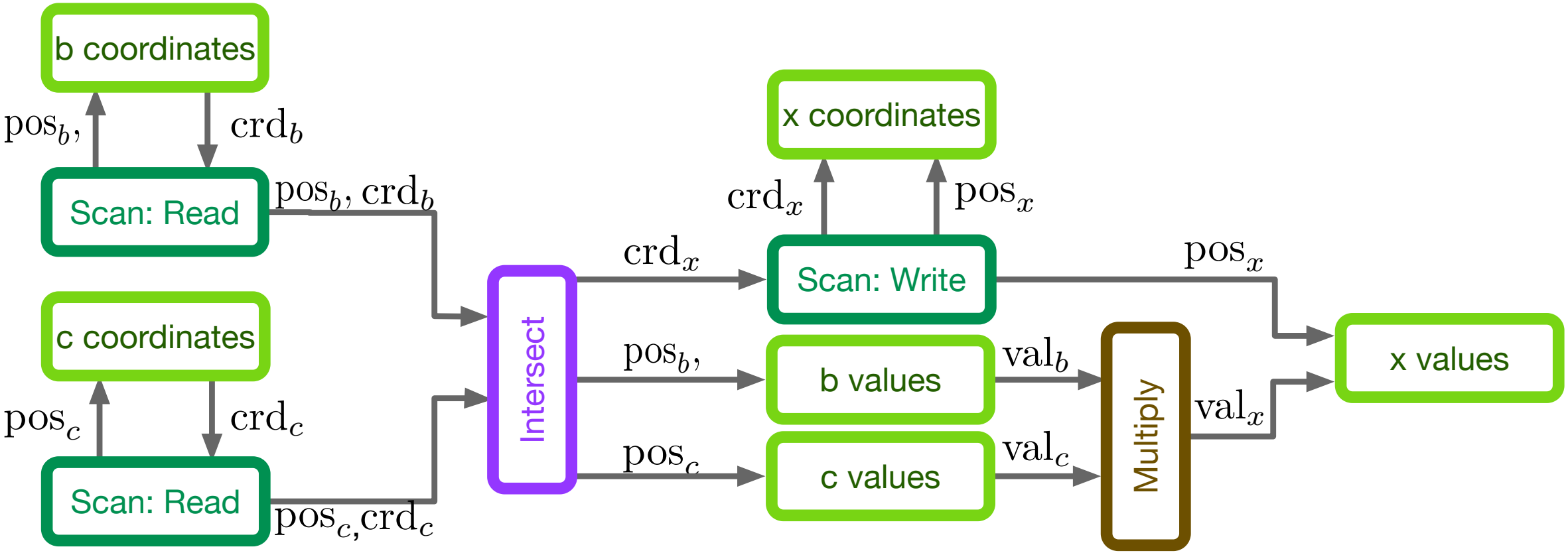
Typing



- Define Interfaces with types
  - All streams are of valued elements
  - Positions and Coordinates are special types with properties
- Streams
  - Have implicit end-token that denotes the end of a rank
  - Could potentially generalize to streams of n-valued tuples (vectorization)
- Need a systematic way of checking type compatibility between modules

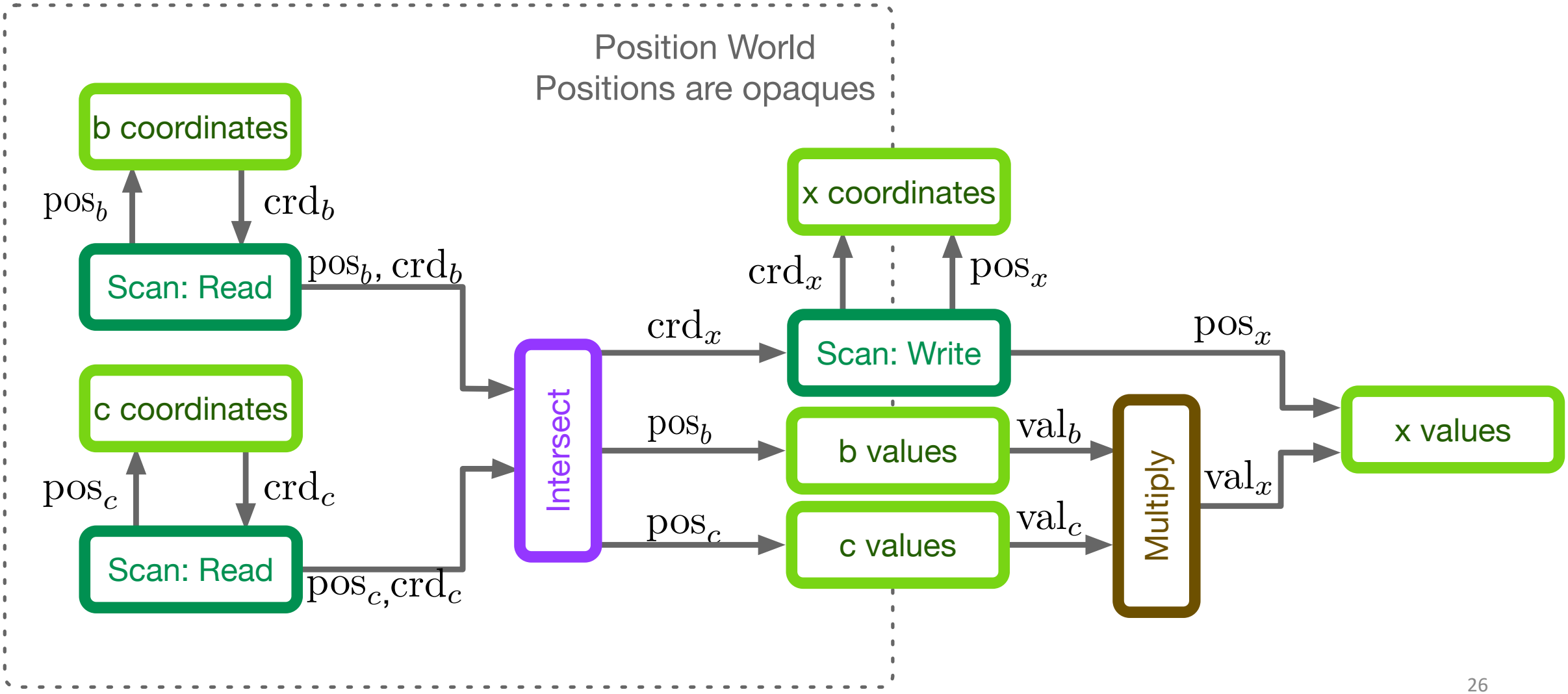
SpV = SpV * SpV

$$x_i = b_i * c_i$$



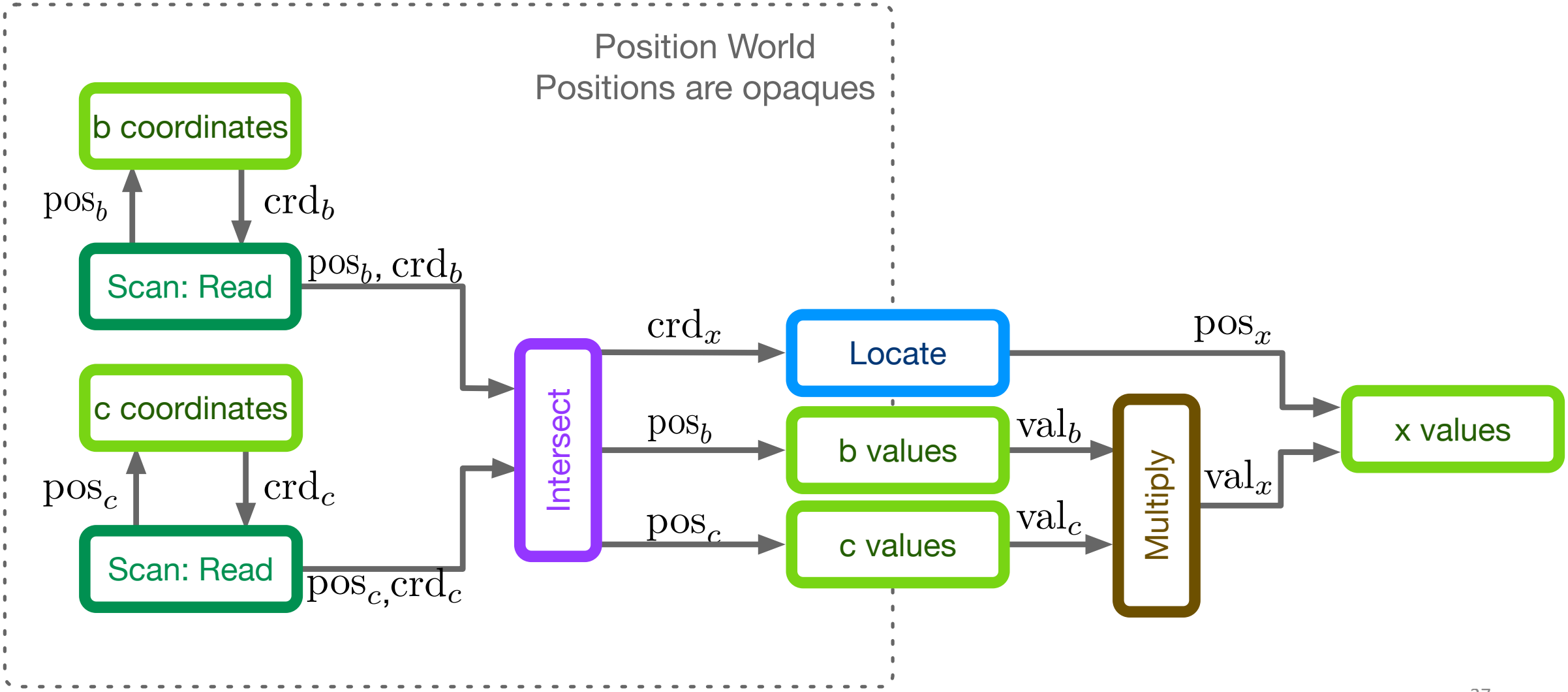
SpV = SpV * SpV Continued

$$x_i = b_i * c_i$$



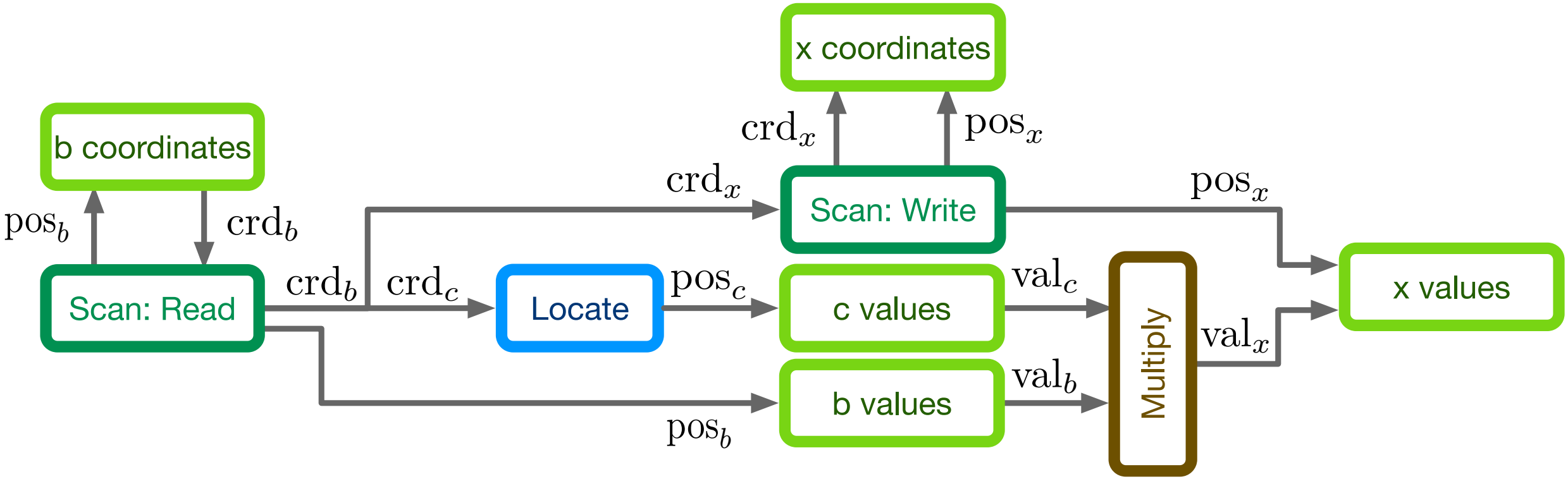
V = SpV * SpV

$$x_i = b_i * c_i$$



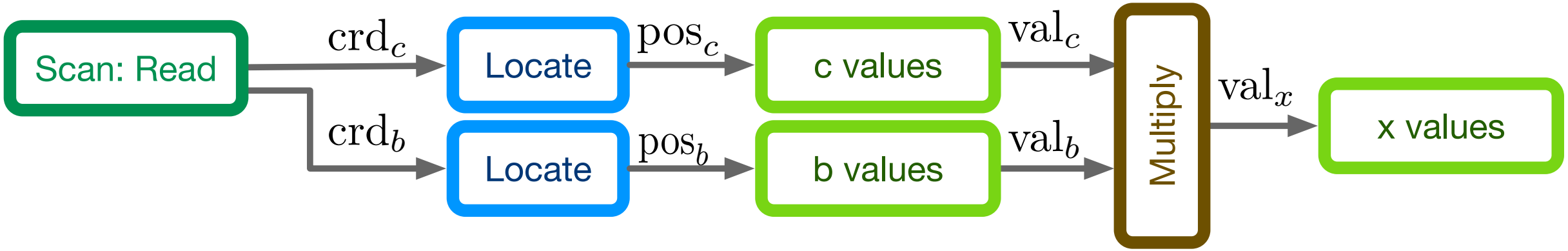
SpV = SpV * V

$$x_i = b_i * c_i$$



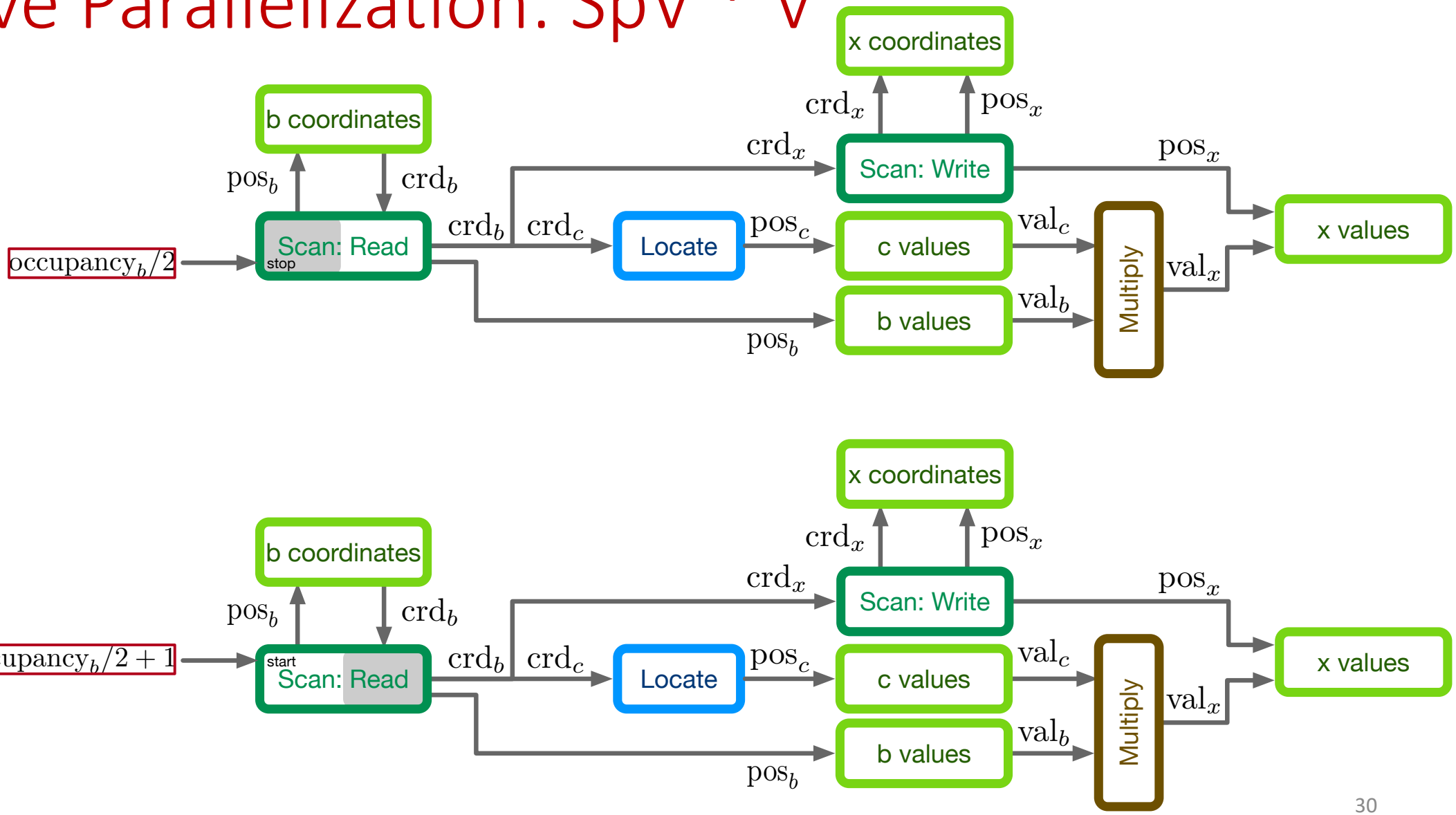
V = V * V

$$x_i = b_i * c_i$$


Scan: Read
crd_c
crd_b
Locate
Locate
pos_c
pos_b
c values
b values
val_c
val_b
Multiply
val_x
x values

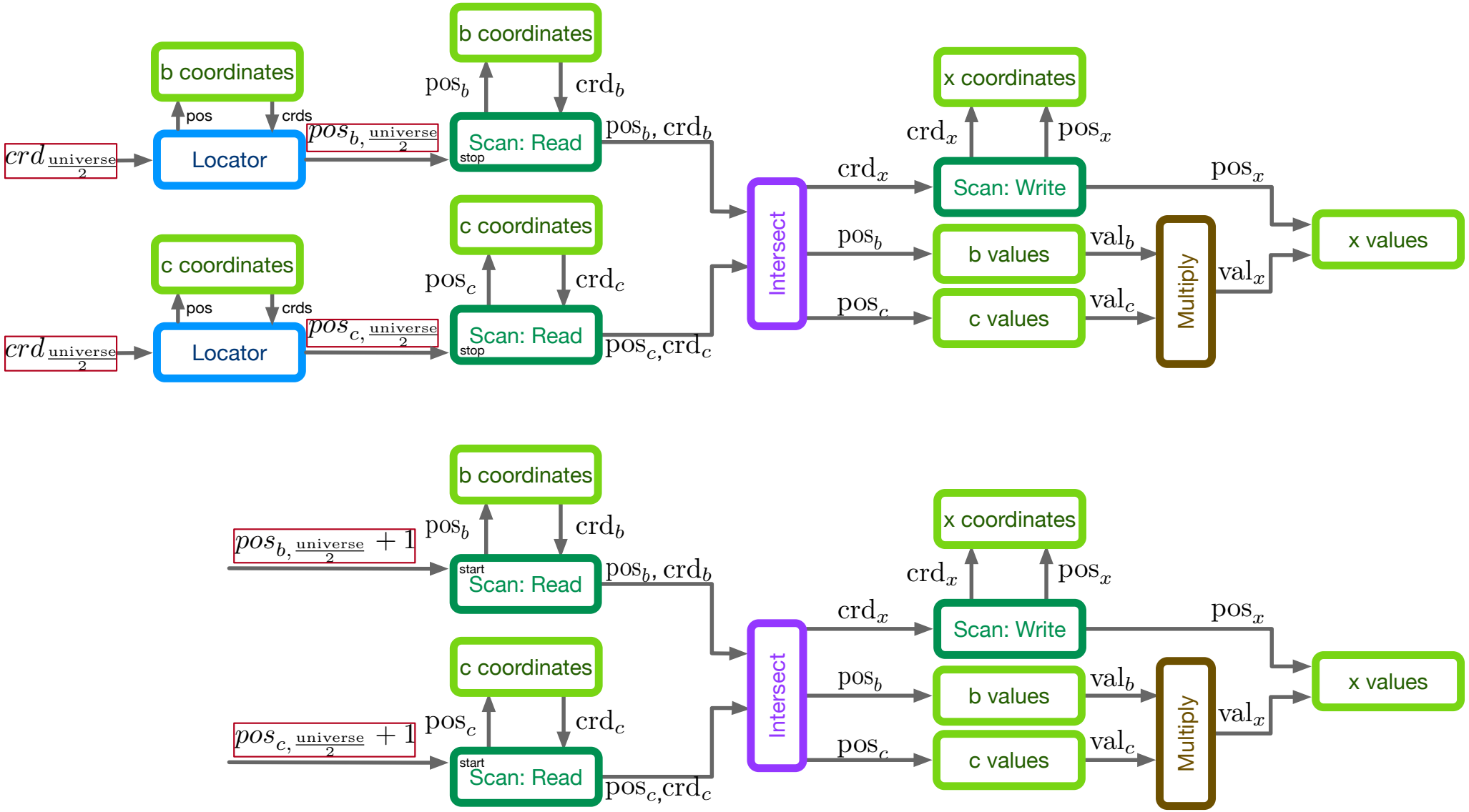
$$x_i = b_i * c_i$$

Naïve Parallelization: SpV * V



$$x_i = b_i * c_i$$

Parallelization Coord Space: SpV * SpV



Parallelization Things to Think About

- Parallelize by other split methods (more complicated)
  - SpV * SpV → Cut one vector by occupancy/2 and locate into other input
  - Split both by occupancy/2 and then take the average of the 2 coordinates
  - Goal is to get the most balanced workload for both halves
  - Can always find sparsity pattern that is pathological for each method
- Parallelizing spatially is basically duplicating the same graph
- Missing compaction step of the resulting tensor at the end for readout/future use in cascaded kernels

Next Steps

- Look into combining our current abstraction to include the fibertree abstraction in array storage
- Expanding kernels to other degrees of freedom: SpMV
- Analyzing more parallelization techniques (including vectorized streams)
- A compiler-prompted sparse hardware acceleration structure for coordinate skipping on intersections

Preliminary Conclusions

- Working on a logical abstraction for sparse dataflow kernels
- Fibertree + fiber argument stream abstraction
- Parallelization is simple in hardware
- Typing and interfaces are important
- Expanding kernels one degree of freedom (complexity) at a time