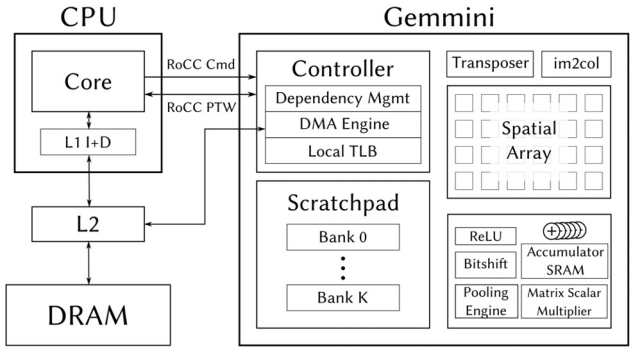
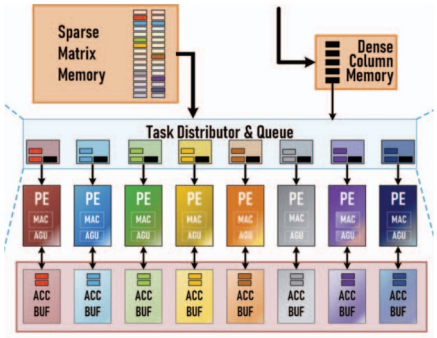
Stellar: An Automated Design Framework for Dense and Sparse Spatial Accelerators

Hasan Genc, Hansung Kim, Prashanth Ganesh, Sophia Shao
University of California, Berkeley (now at NVIDIA)
hgenc@nvidia.com

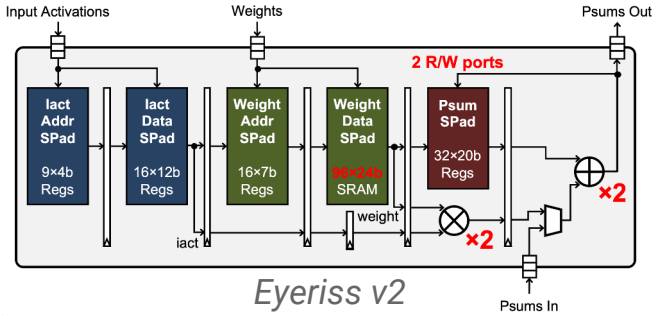
Motivation: Architecture Diversity



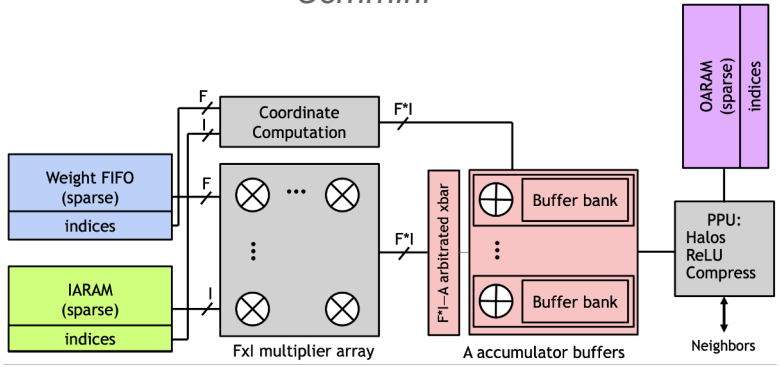
Gemmini



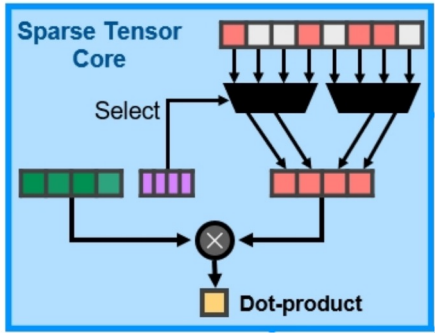
AWB-GCN



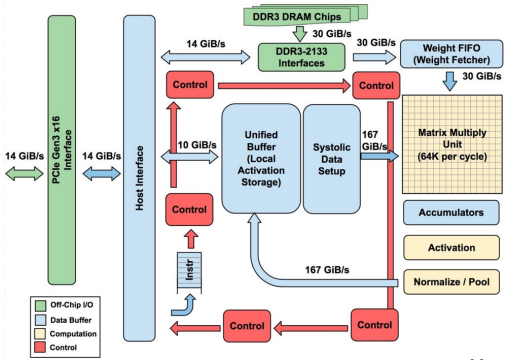
Eyeriss v2



SCNN

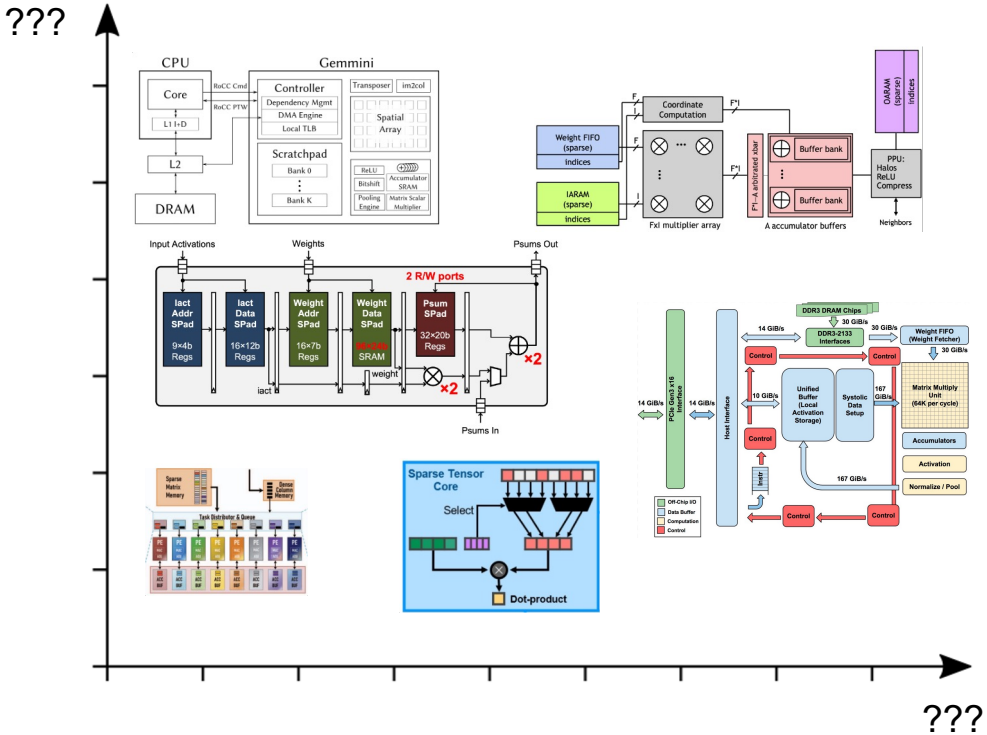


A100

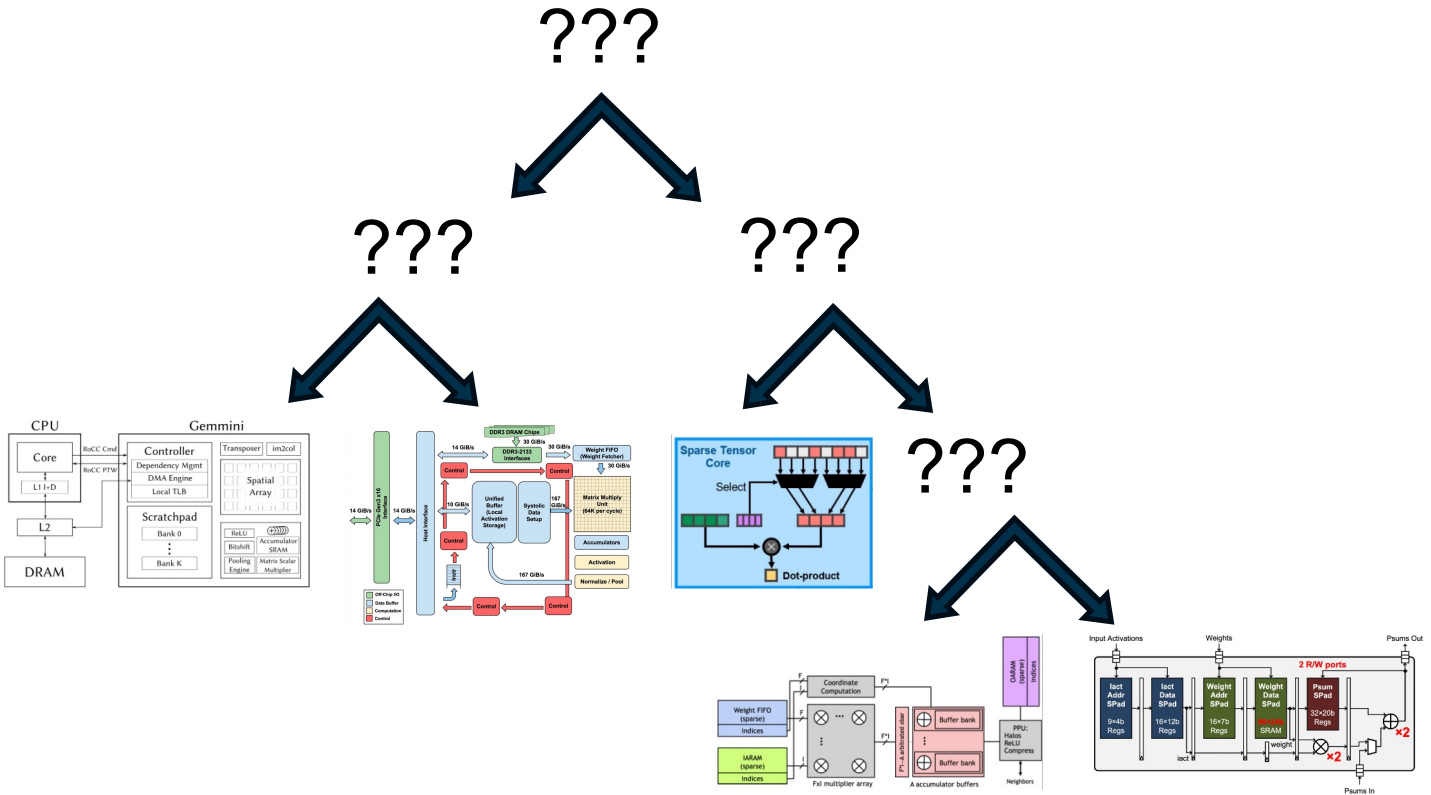


TPU

Motivation: Potential Enumeration?



Motivation: Potential Taxonomy?



Motivation: Separation of Concerns

- We need separation of concerns!

Motivation: Separation of Concerns

- We need separation of concerns!
- Functional behavior
  - E.g. matmul, convolution, sorting, etc.

Matmul

$$C_{ij} = \sum_k A_{ik} B_{kj}$$

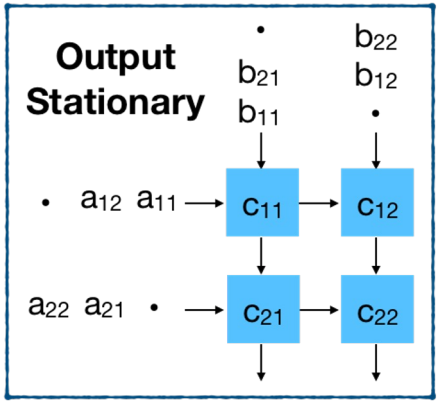
MTTKRP

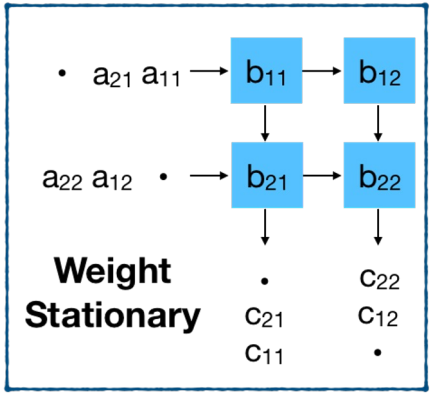
$$A_{ij} = \sum_k \sum_l B_{ikl} D_{lj} C_{kj}$$

Motivation: Separation of Concerns

- We need separation of concerns!

- Functional behavior
  - E.g. matmul, convolution, sorting, etc.
- Dataflow
  - E.g. output-stationary, weight-stationary, etc.

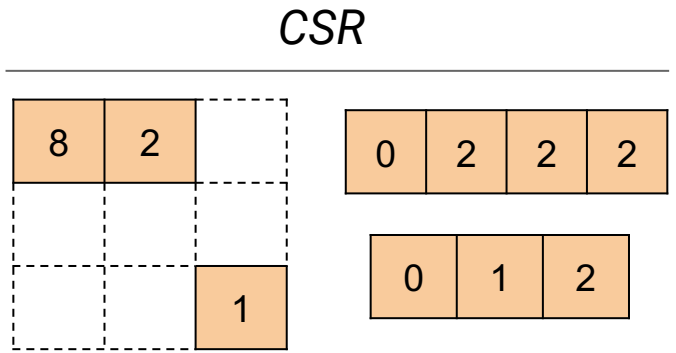


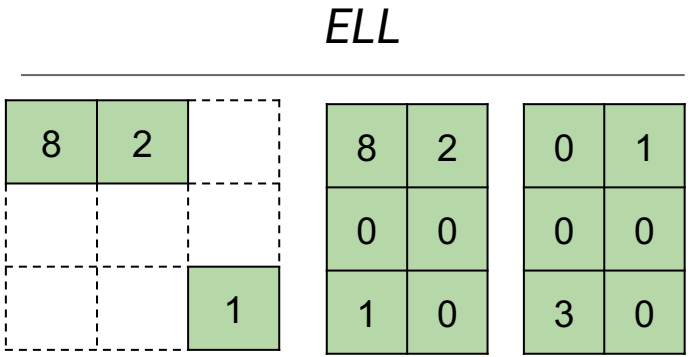


Motivation: Separation of Concerns

- We need separation of concerns!

- Functional behavior
  - E.g. matmul, convolution, sorting, etc.
- Dataflow
  - E.g. output-stationary, weight-stationary, etc.
- Data formats
  - E.g. CSR, ELL, DBB, diagonal

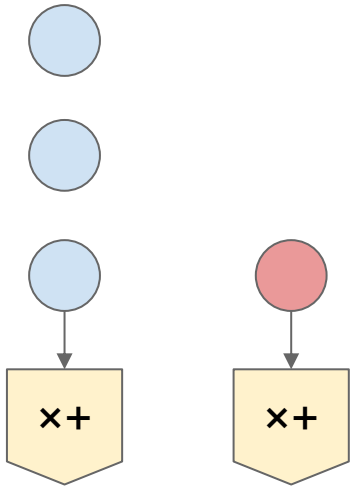




Motivation: Separation of Concerns

- We need separation of concerns!

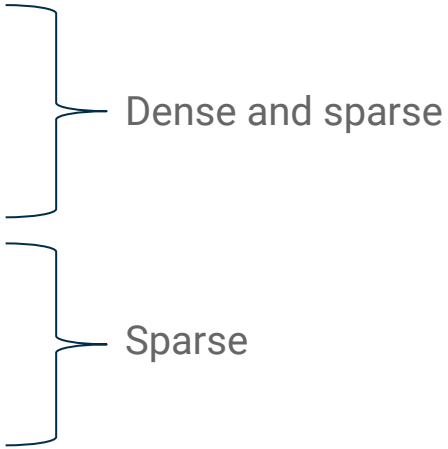
- Functional behavior
  - E.g. matmul, convolution, sorting, etc.
- Dataflow
  - E.g. output-stationary, weight-stationary, etc.
- Data formats
  - E.g. CSR, ELL, DBB, diagonal
- Load-balancing
  - Affects cost of NoC



Motivation: Separation of Concerns

- We need separation of concerns!

- Functional behavior
  - E.g. matmul, convolution, sorting, etc.
- Dataflow
  - E.g. output-stationary, weight-stationary, etc.
- Data formats
  - E.g. CSR, ELL, DBB, diagonal
- Load-balancing
  - Affects cost of NoC



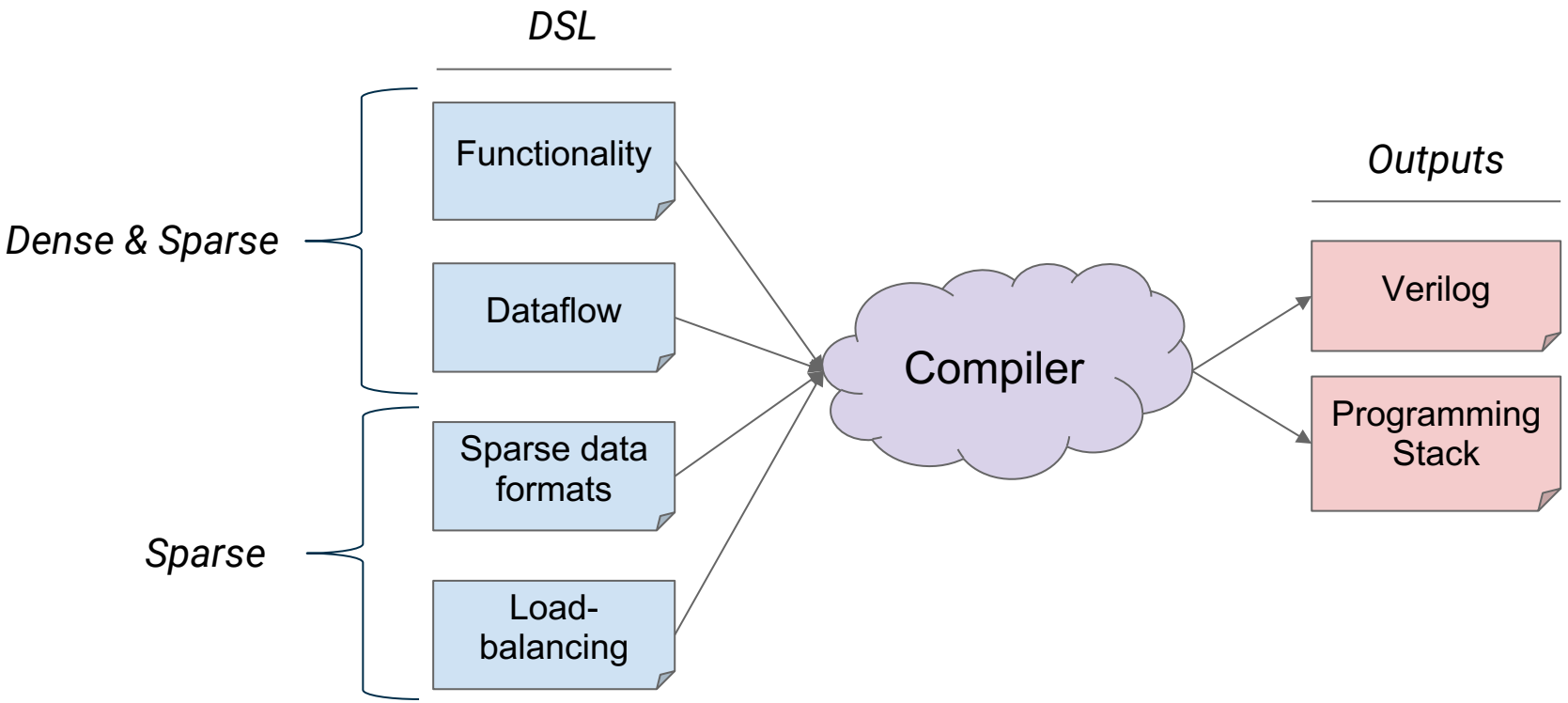
Prior Work

- Dense accelerator design frameworks
  - PolySA
  - Auto
  - Inte
- Sparse a
  framewo
  - TeAAL
  - Sparse Abstract Machine
  - Sparseloop

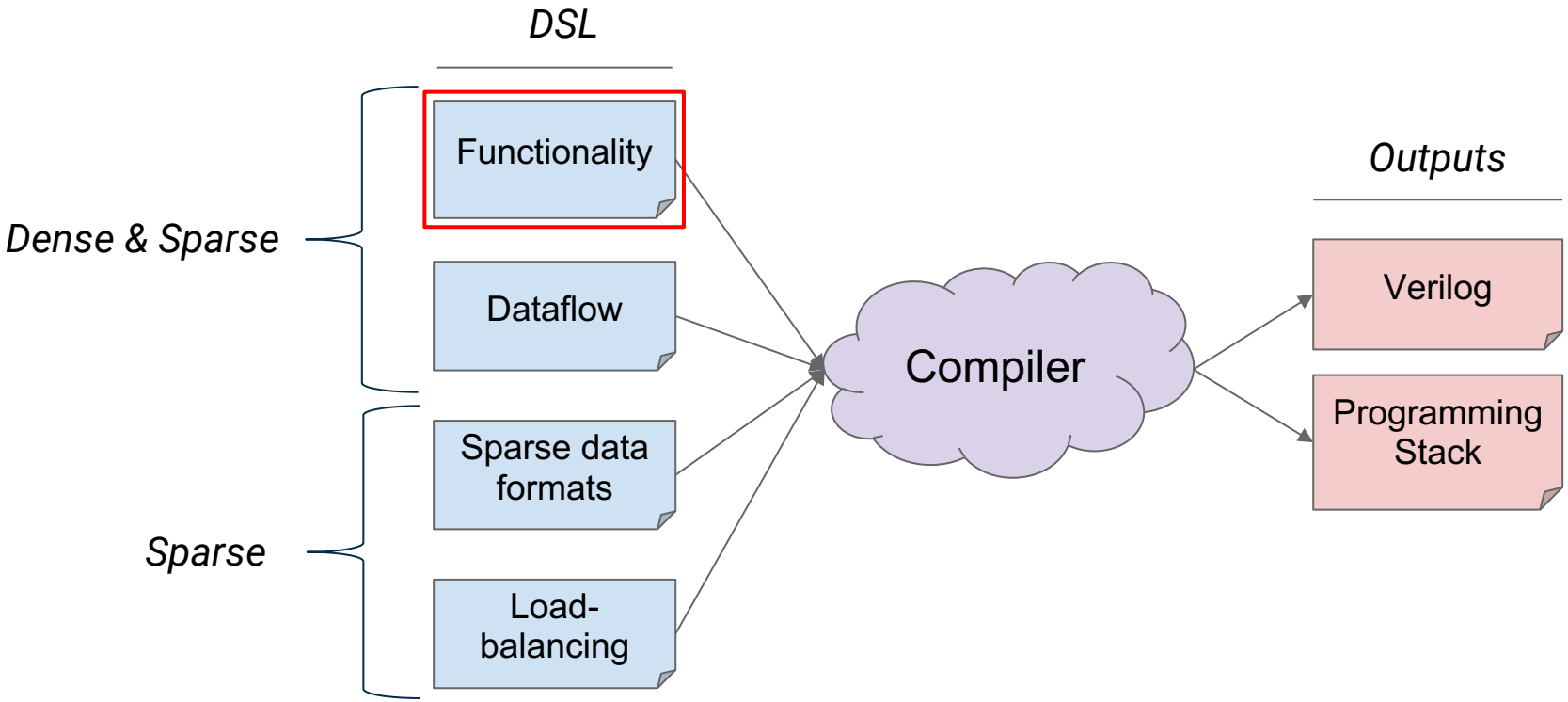
- Limitations of prior work:
  - No sparse RTL generation
    - Primary focus is **simulation**
    - iche
    - ows or
    - ancing
    - schemes

Stellar enables both rapid **specification** and RTL **generation** for **both** dense and sparse accelerators

Overview of Stellar



Expressing Functionality



Expressing Functionality

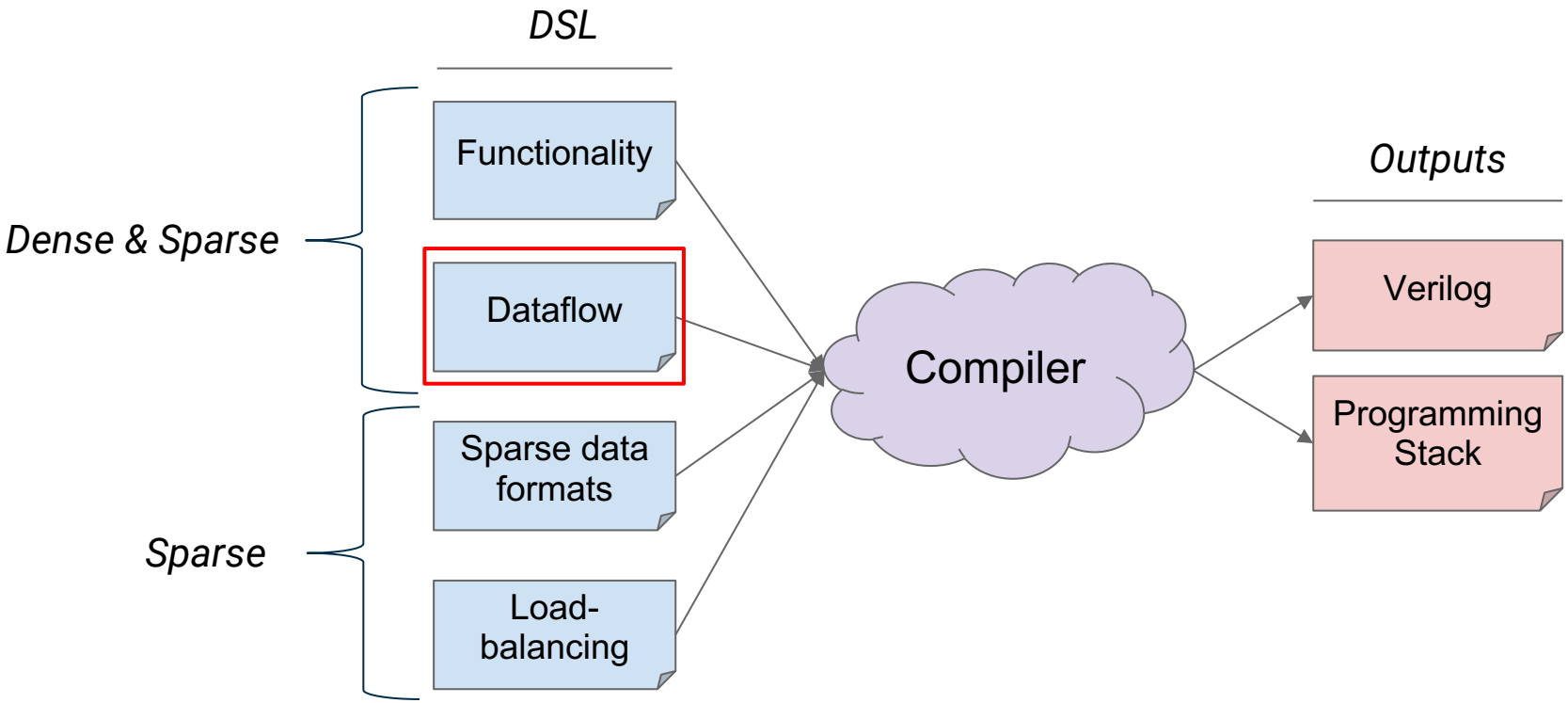
- Similar to prior work on automated systolic array generators
  - E.g. *Algorithms of Informatics*, 2010

- Indirect accesses also supported
  - Helps with merging/sorting algorithms

"Halide-like" Matmul

$$a(i, j, k) = a(i, j - 1, k)$$
$$b(i, j, k) = b(i - 1, j, k)$$
$$c(i, j, k) = c(i, j, k - 1) + a(i, j - 1, k) * b(i - 1, j, k)$$

Overview of Stellar



Expressing Dataflows

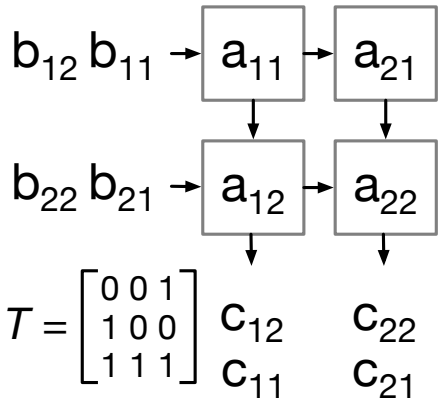
- Defines the placement of PEs
- Defines the order of operations
- Defines the **connections** between PEs in the **ideal, dense** case
    - Some of these PE-to-PE connections will be **broken** and replaced with external IO based on the sparse data format

Space-Time Transform

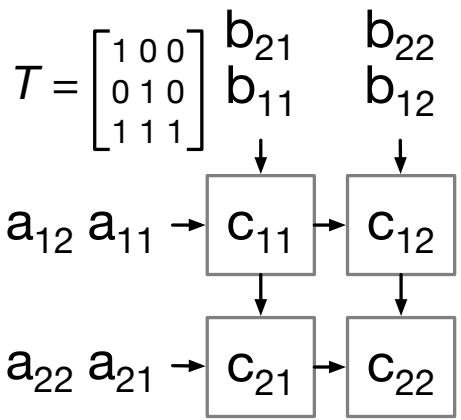
$$f : \begin{bmatrix} i \\ j \\ k \end{bmatrix} \rightarrow \begin{bmatrix} x \\ y \\ t \end{bmatrix}$$

Expressing Dataflows

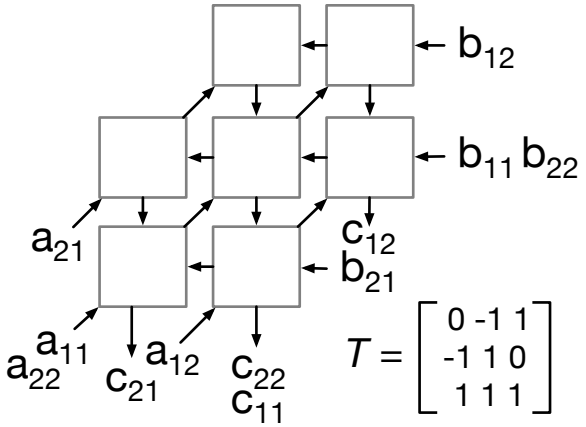
Input-Stationary



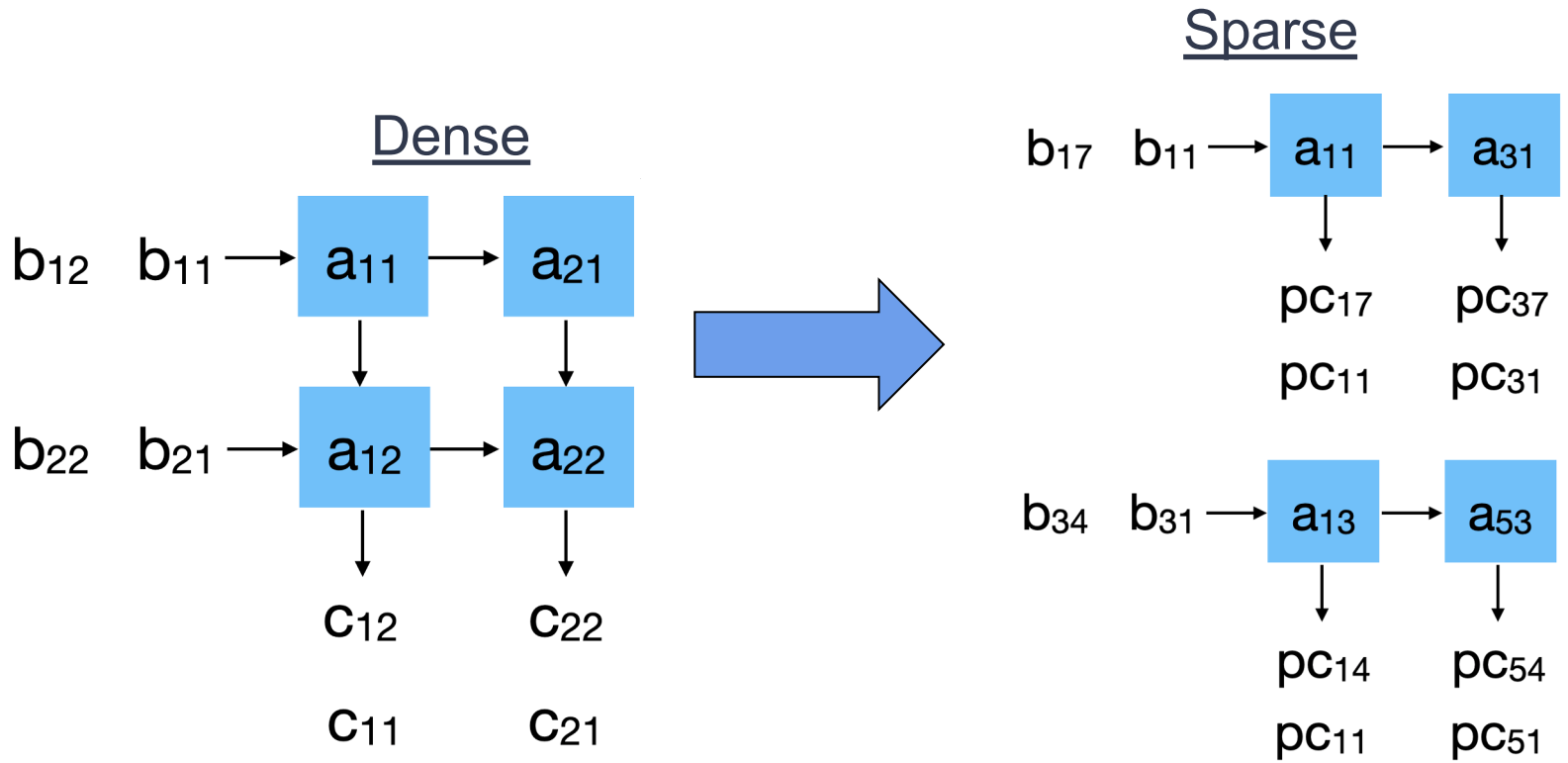
Output-Stationary



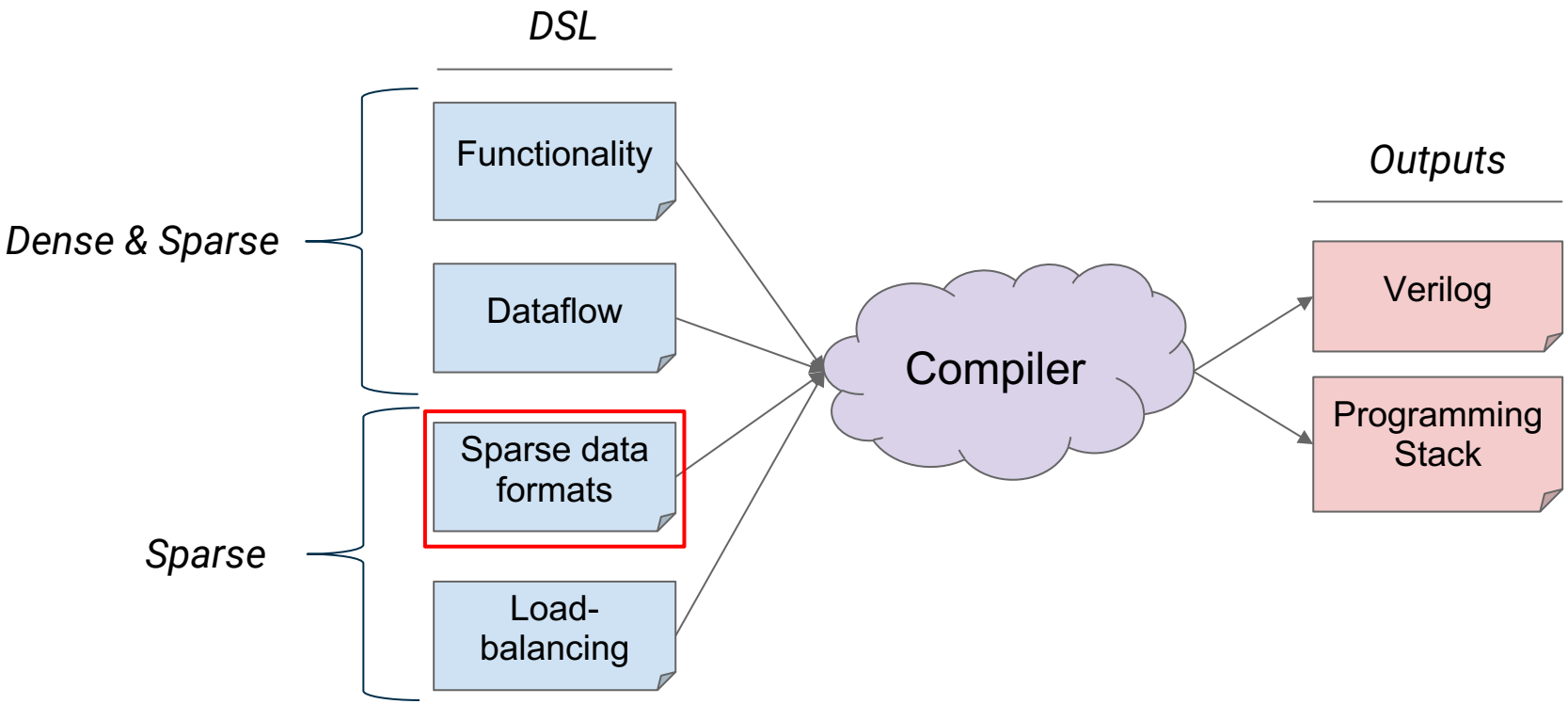
Hexagonal



Expressing Dataflows



Expressing Sparse Data Formats



Expressing Sparse Data Formats

- We can express sparsity in terms of which iterators are “skipped”
  - And what the conditions for the “skip” are

A * B = C where A and B are CSR

Skip *i* if A(*i*,*k*) == 0

Skip *j* if B(*k*,*j*) == 0

A * B = C where A is diagonal

Skip *i* if *i* != *k*

Skip *k* if *i* != *k*

Simple “row-sparsity” example

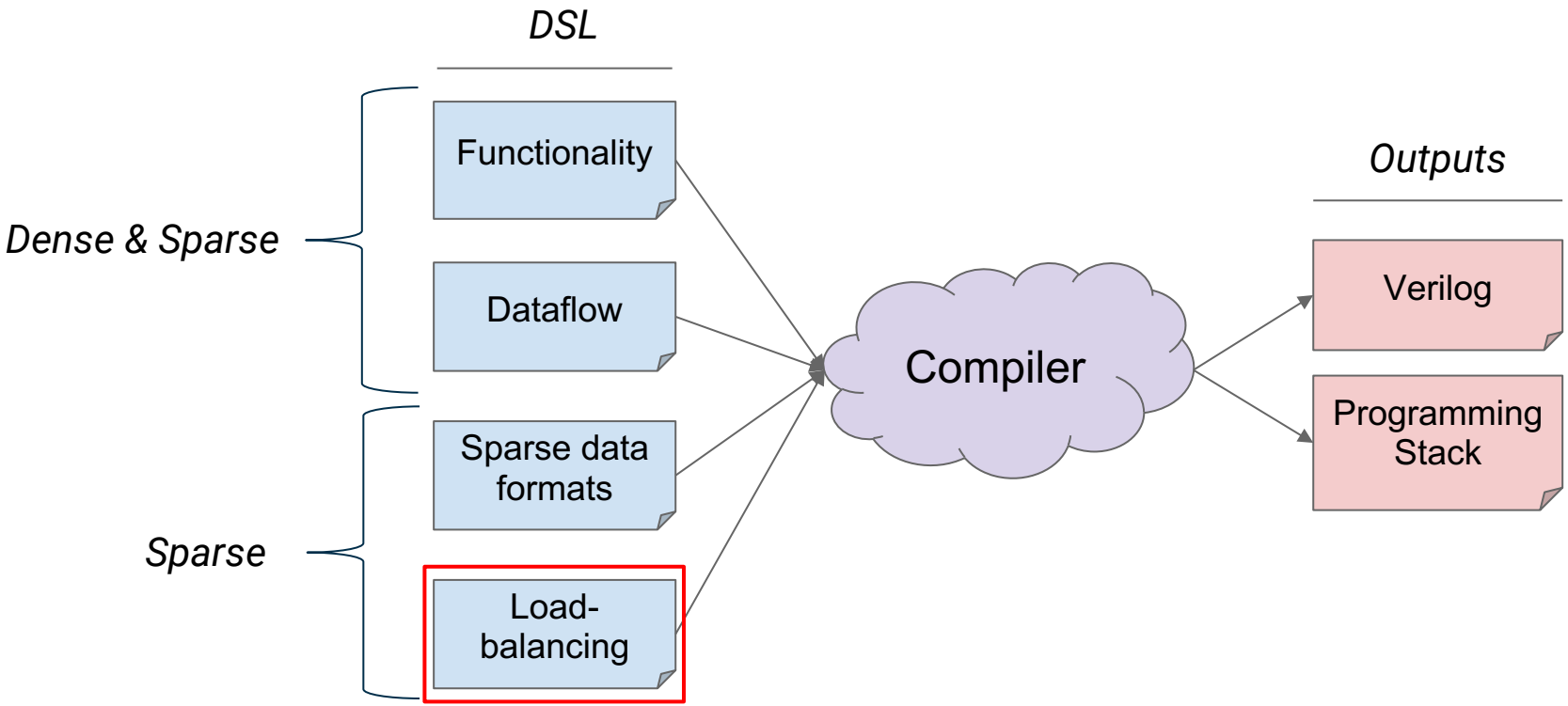
Skip *k* if A(*i*,->) == 0

Expressing Sparse Data Formats

- *Skip* statements are sufficient for spatial array design, but not for memory buffer design
- Memory buffers need to know how metadata is stored, while spatial arrays only need to know which elements to skip
  - *Bitmap* and *CSR* tensors can encode the same matrix, but with very different storage overhead!

- Memory buffers are defined using the *fibertree* notation
  - Every *dimension* in the tensor gets its own sparsity format
- Examples:
  - CSR:
    - *Dense* rows
    - *Compressed* columns
  - Bitmap:
    - *Dense* rows
    - *Bitvector* columns
  - Block-CSR
    - *Dense -> Compressed -> Dense -> Dense*

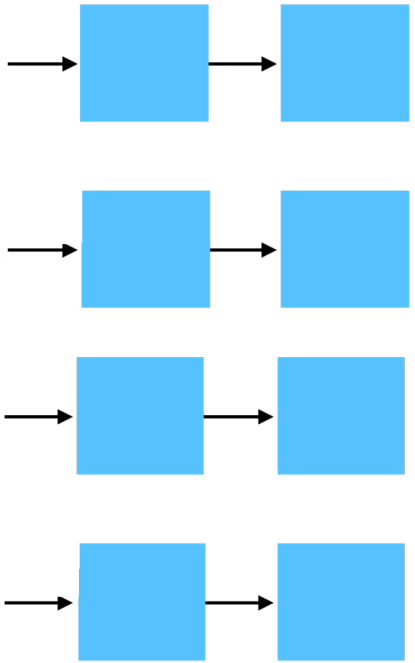
Expressing Load-Balancing



Expressing Load-Balancing

- Idle PEs can perform work that over-utilized PEs would have performed in the future

Spatial Array With No Load-Balancing

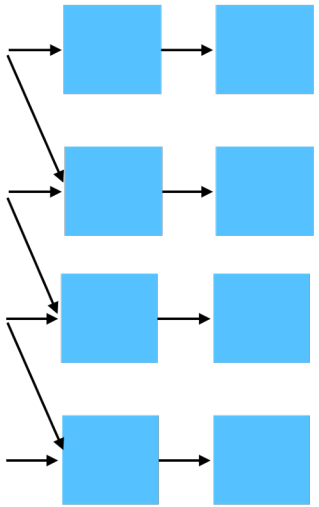


Expressing Load-Balancing

- Idle PEs can perform work that over-utilized PEs would have performed in the future

Remap Over-Utilized Operations to Adjacent Rows

```
Map (i.upperBound->, j, k) to (
    i.lowerBound -> i.upperBound,
    j,
    k + 1
)
```

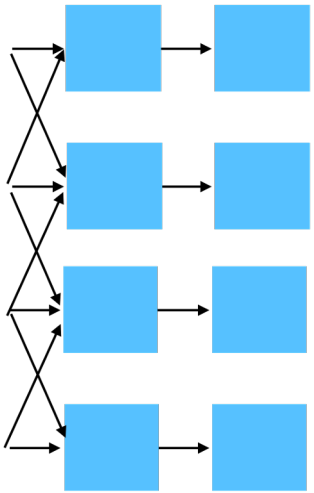


Expressing Load-Balancing

- Idle PEs can perform work that over-utilized PEs would have performed in the future

Remap Over-Utilized Operations to Adjacent Rows

```
for (n <- -1 to 1)
    Map (i.upperBound->, j, k) to (
        i.lowerBound -> i.upperBound,
        j,
        k + n
    )
```

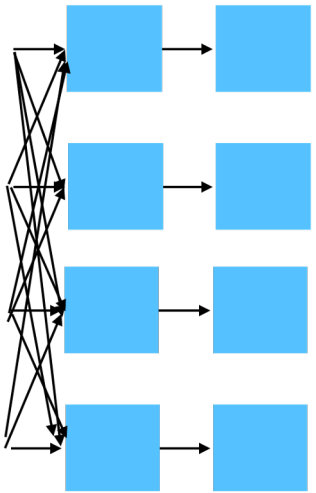


Expressing Load-Balancing

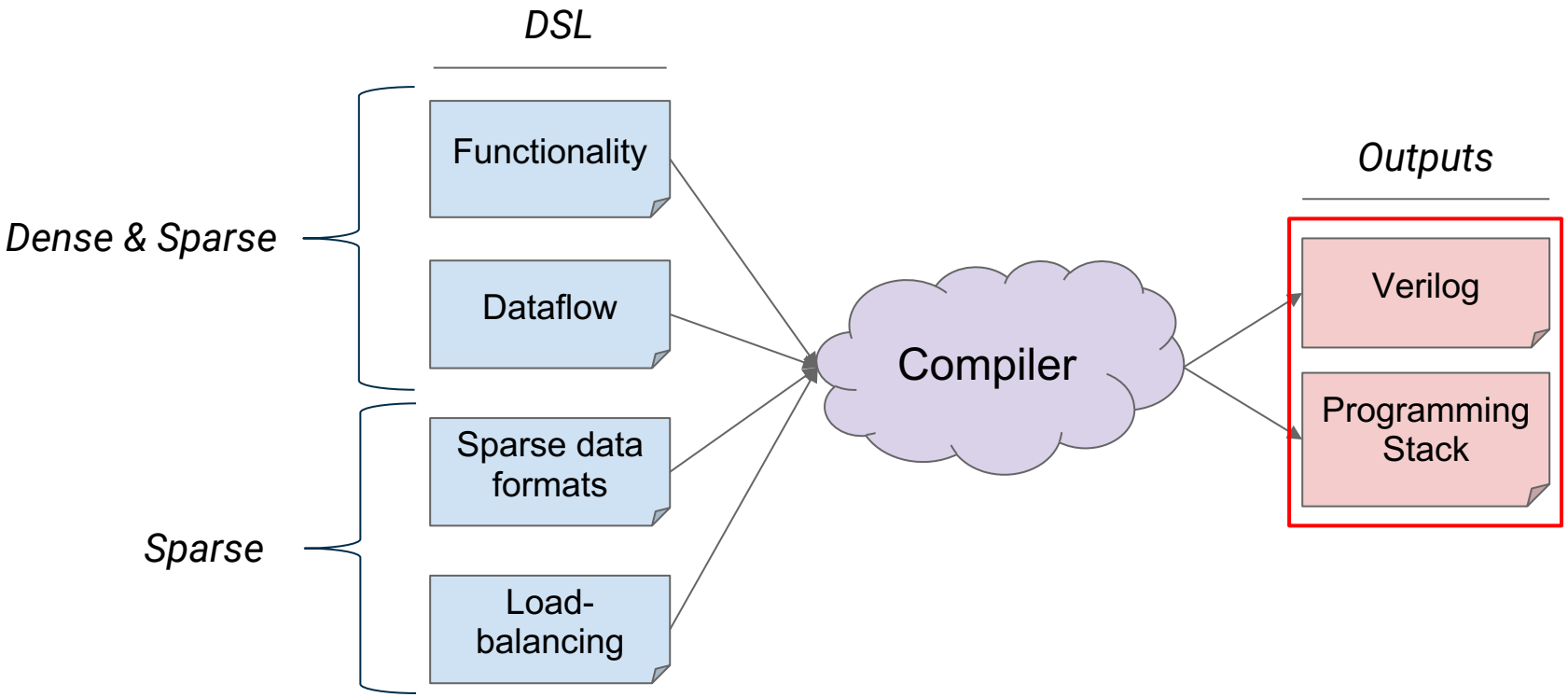
- Idle PEs can perform work that over-utilized PEs would have performed in the future

Remap Over-Utilized Operations to Adjacent Rows

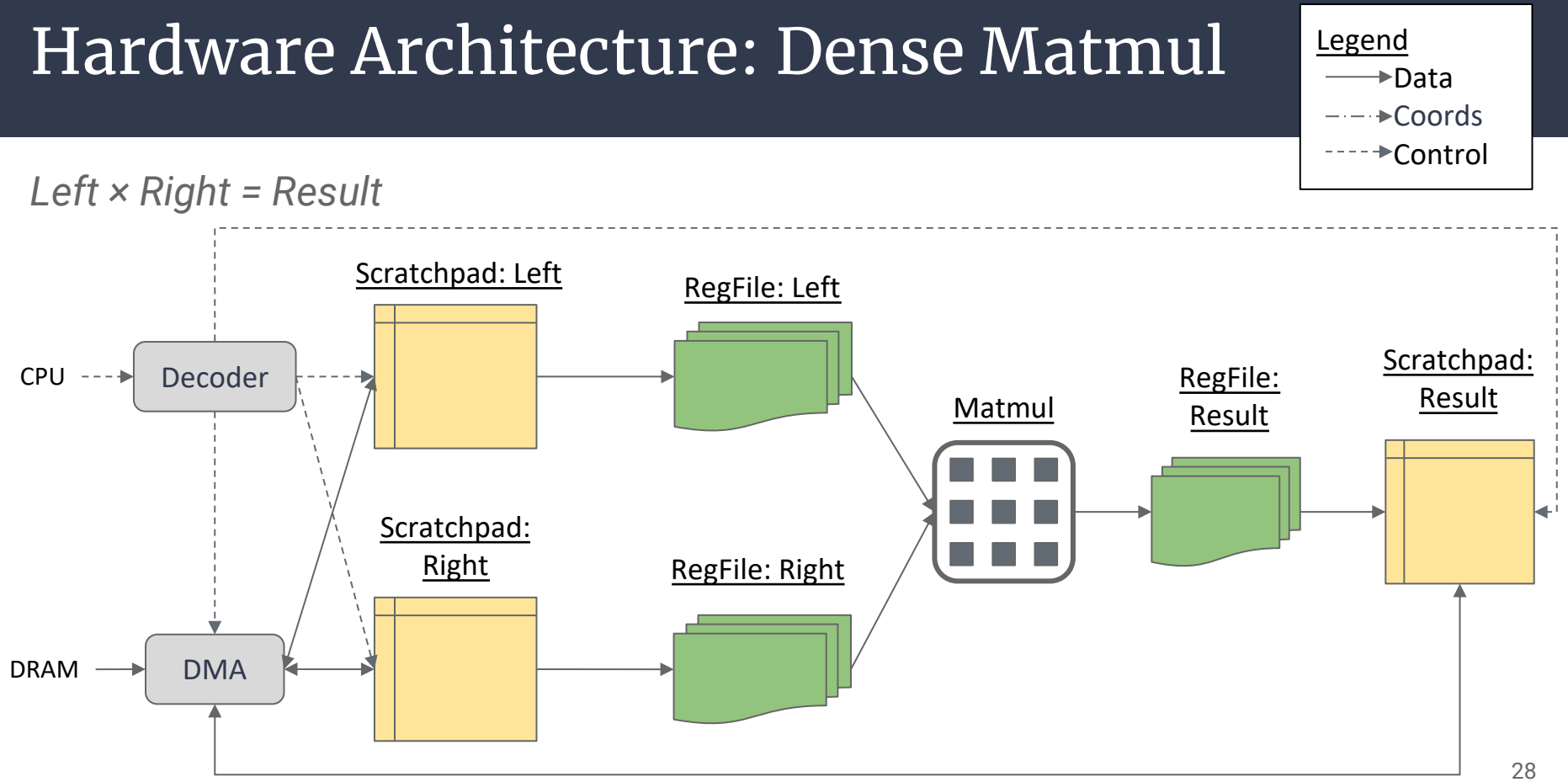
```
for (n <- -3 to 3)
    Map (i.upperBound->, j, k) to (
        i.lowerBound -> i.upperBound,
        j,
        k + n
    )
```



Outputs

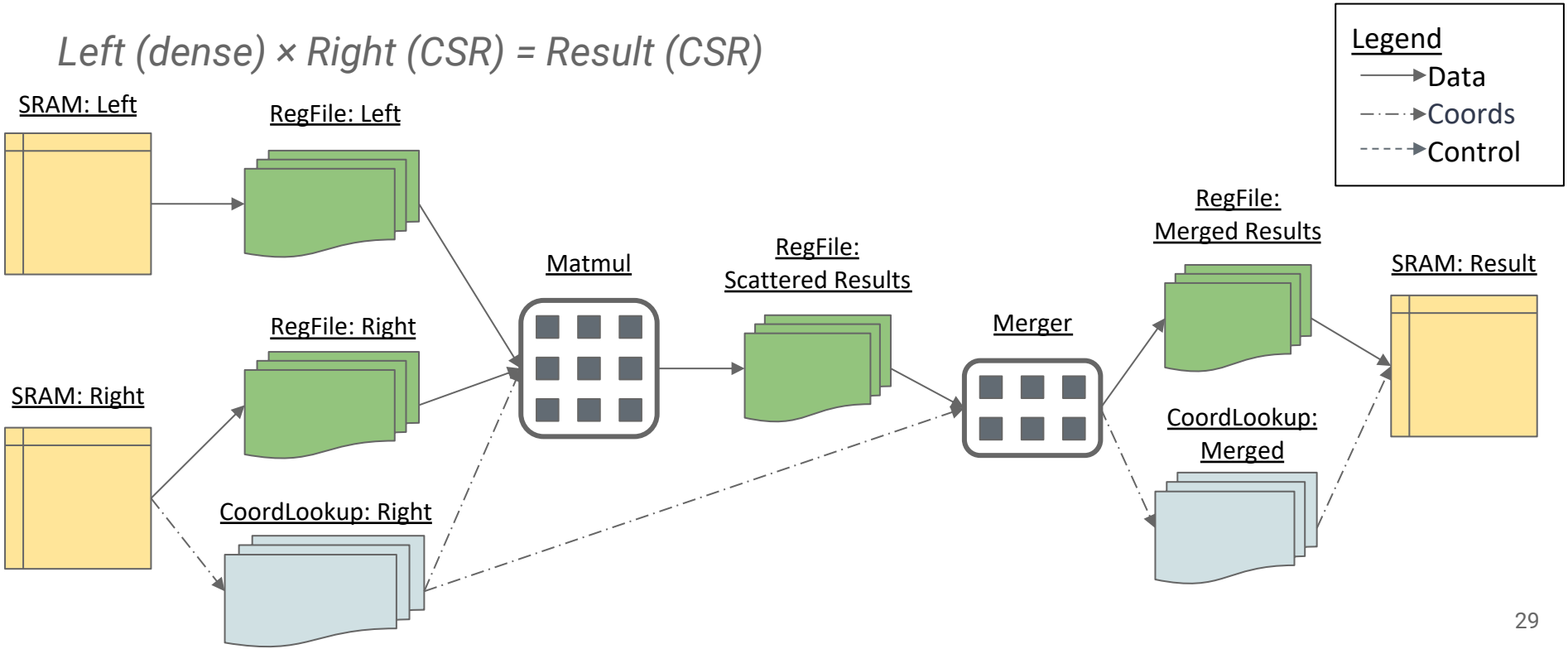

DSL
Functionality
Dataflow
Dense & Sparse
Sparse data formats
Load-balancing
Sparse
Compiler
Outputs
Verilog
Programming Stack

Hardware Architecture: Dense Matmul



Hardware Architecture: Sparse-Dense Matmul

Left (dense) × Right (CSR) = Result (CSR)



Hardware Generation: Spatial Arrays

1. Generate “tensor iteration space”
    - From functional description

Functional Description

// Inputs

$a(i, j.lowerBound, k) = A(i, k)$

$b(i.lowerBound, j, k) = B(i, k)$

// Intermediate calculations

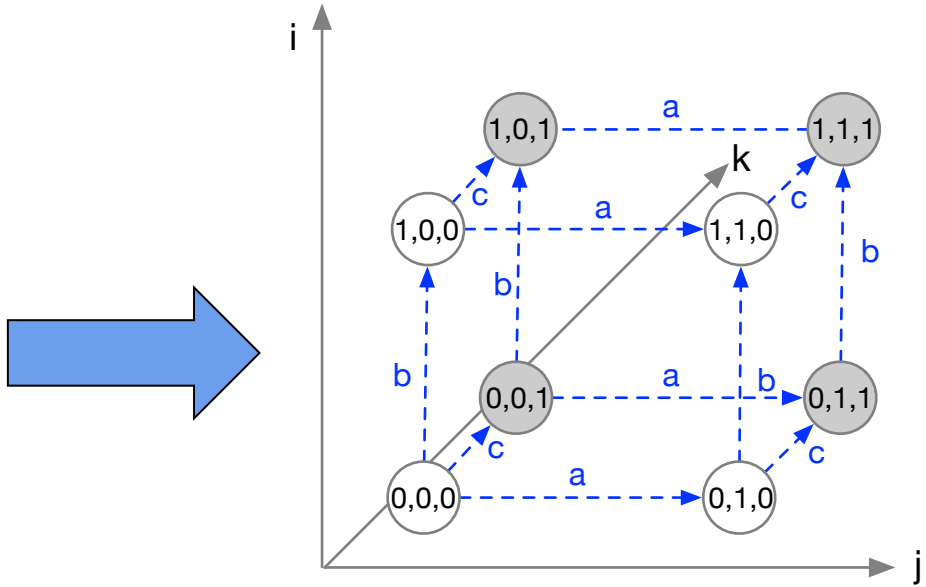
$a(i, j, k) = a(i, j - 1, k)$

$b(i, j, k) = b(i - 1, j, k)$

$c(i, j, k) = c(i, j, k - 1) + a(i, j - 1, k) * b(i - 1, j, k)$

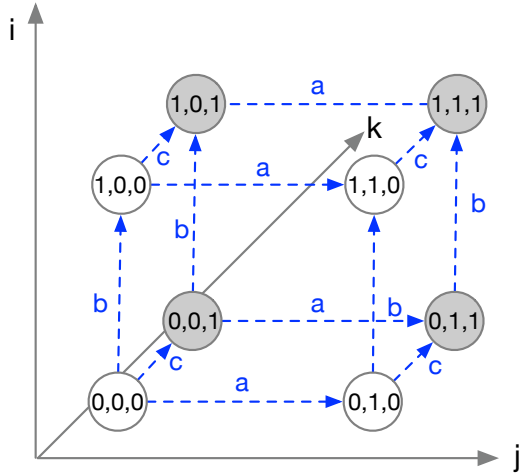
// Outputs

$C(i, j) = c(i, j, k.lowerBound)$

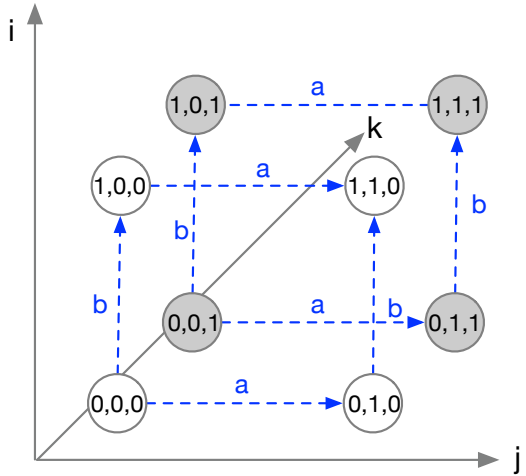


Hardware Generation: Spatial Arrays

1. Generate “tensor iteration space”
   - From functional description
2. Prune PE-to-PE connections
   - Based on sparsity formats and load-balancing

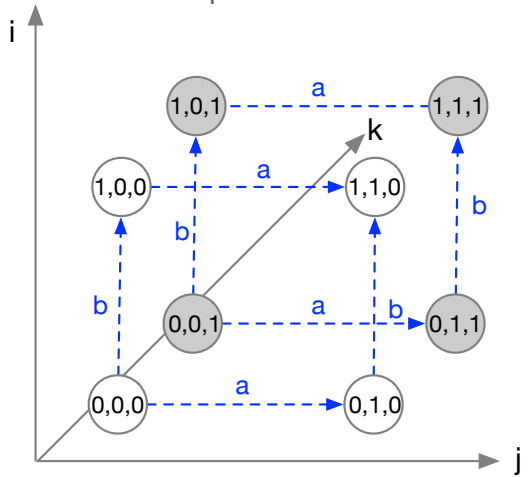


A is CSC and B is CSR
Skip *i* if A(*i*,*k*) == 0
Skip *j* if B(*k*,*j*) == 0

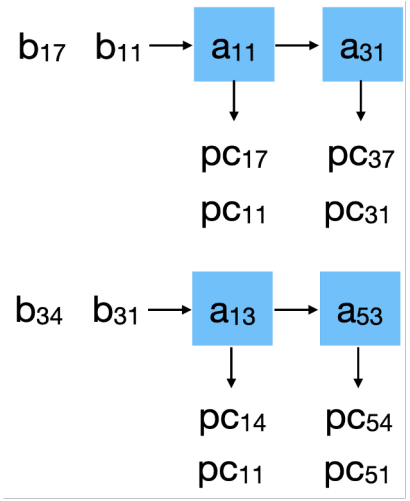


Hardware Generation: Spatial Arrays

1. Generate “tensor iteration space”
   - From functional description
2. Map to dense physical array
   - From space-time transform
3. Map to dense physical array
   - From space-time transform



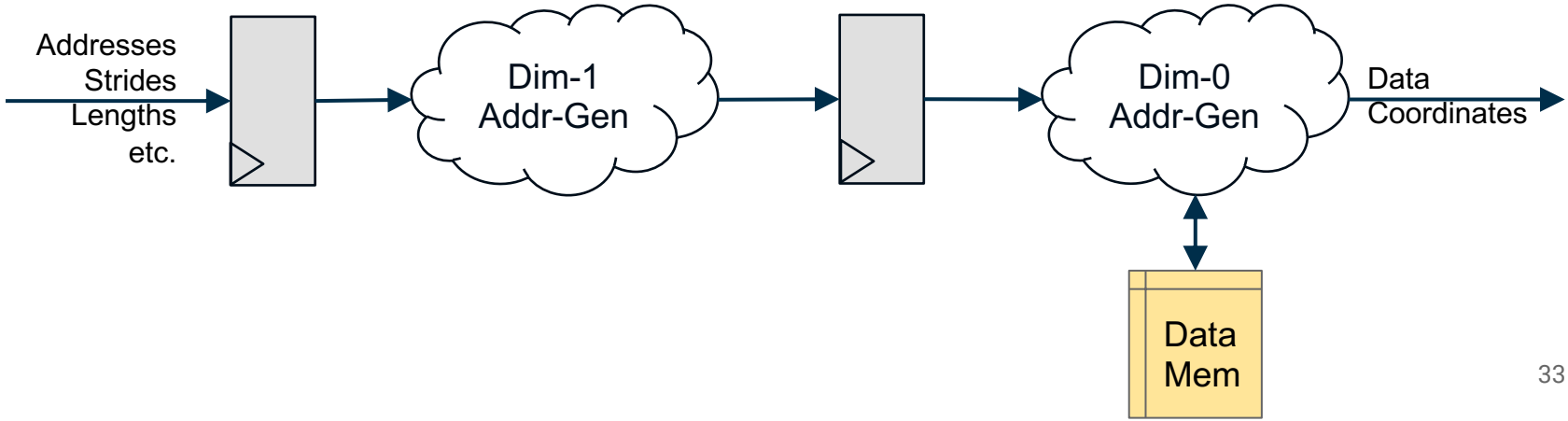
$$T = \begin{bmatrix} 0 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$



Hardware Generation: Memory Buffers

- Handwritten templates for different types of sparsity format
- Template creates different pipeline stages for each *dimension* of a sparse/dense tensor

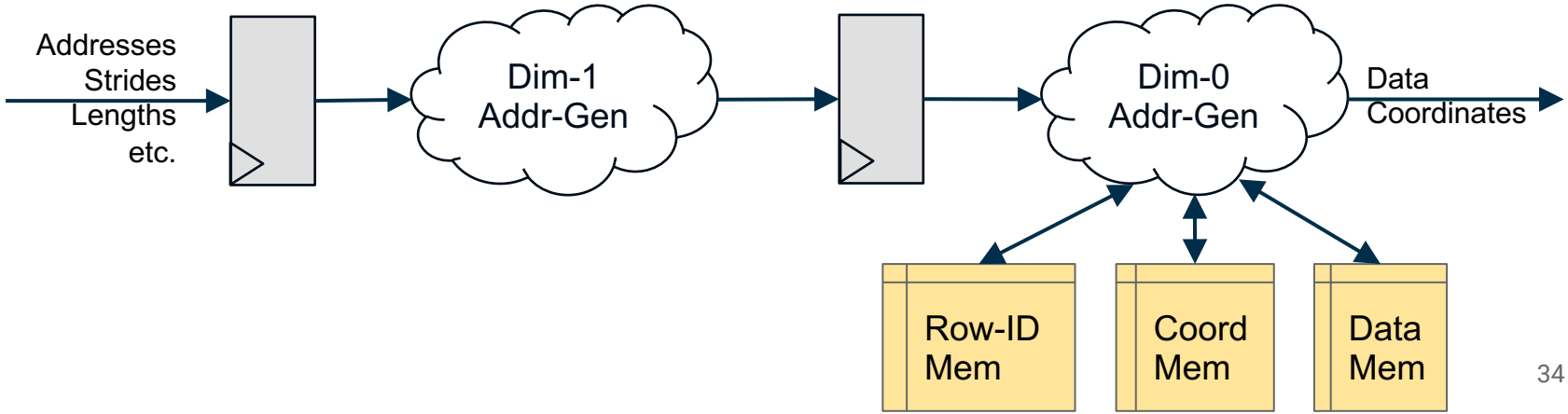
Memory Buffer for Dense Matrices



Hardware Generation: Memory Buffers

- Handwritten templates for different types of sparsity format
- Template creates different pipeline stages for each *dimension* of a sparse/dense tensor

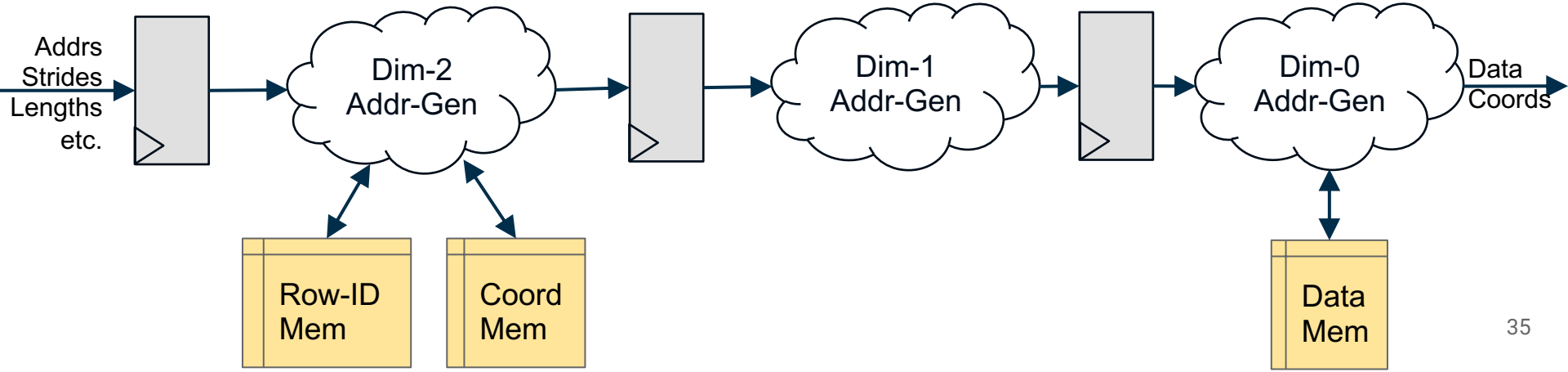
Memory Buffer for CSR Matrices



Hardware Generation: Memory Buffers

- Handwritten templates for different types of sparsity format
- Template creates different pipeline stages for each *dimension* of a sparse/dense tensor

Memory Buffer for Block-CSR Tensors



Hardware Generation: Memory Buffers

- Handwritten templates for different types of sparsity format
- Template creates different pipeline stages for each *dimension* of a sparse/dense tensor

- Memory buffer RTL templates are designed to be pretty generalizable
  - Scatter-gather support, synchronization, interleaving, banking, etc.
- Hardcoding runtime parameters at elaboration time helps a lot

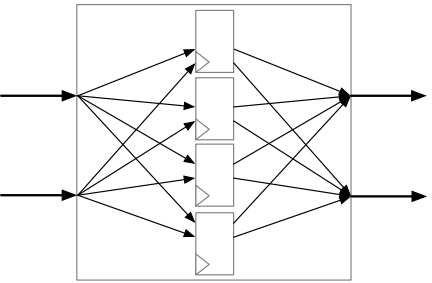
```
1  def hardCoded(x: MemPipeline) = Map(
2    x.read_req.spans(0) -> 4.U,
3    x.read_req.spans(1) -> 4.U,
4    x.read_req.data_strides(0) -> 1.U,
5    x.read_req.data_strides(1) -> 4.U)
```

Example: Hardcoding striding patterns at elaboration-time

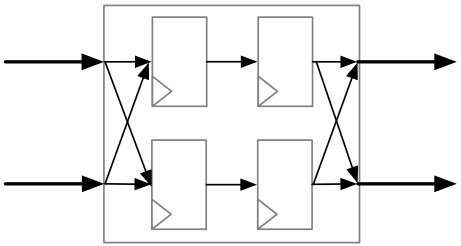
Hardware Generation: Register Files

- Initial hardware output is *flexible* but *expensive*
- Optimization passes will reduce area/power of hardware based on elaboration-time constraints

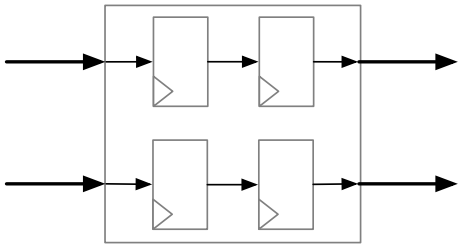
- Regfiles are optimized by “hardening” their “edges” based on **dataflow** and scratchpad access patterns
  - Optimizing edges also allows some data transformation optimizations



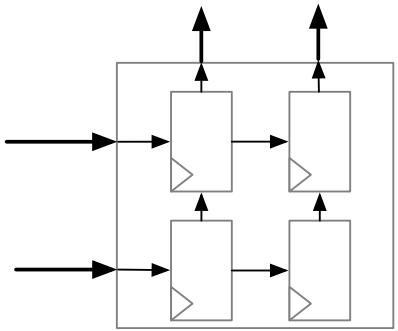
Flexible baseline



IO only at edges



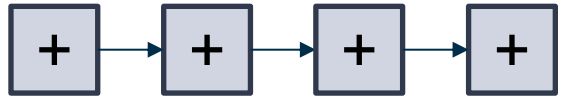
Each IO port connects only to a single RF entry at edge



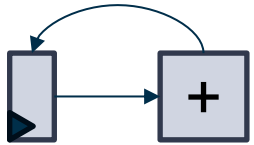
Transpose

Limitations

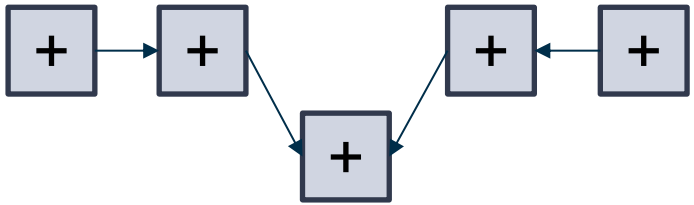
- “Affine” reductions vs “tree-reductions”
- Stellar can easily represent “affine” dataflow transformations
- Stellar isn’t great at representing recursive, or “tree-based” transformations



Chain-reduction



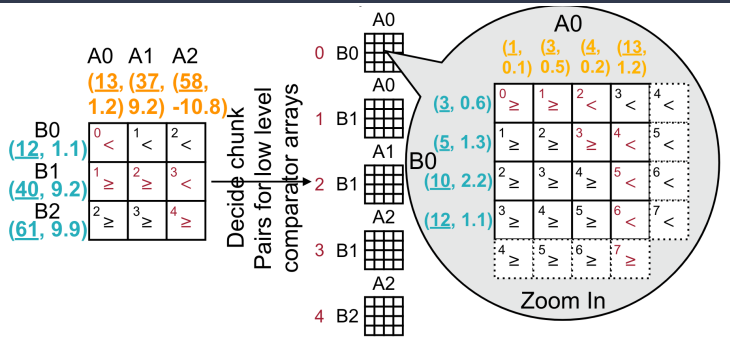
Temporal reduction

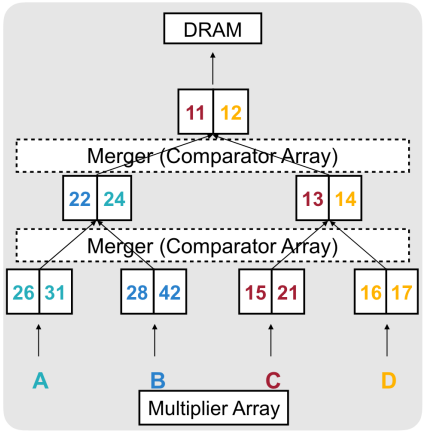


Tree-reduction

Limitations

- “Affine” reductions vs “tree-reductions”
- Stellar can easily represent “affine” dataflow transformations
- Stellar isn’t great at representing recursive, or “tree-based” transformations
  - ...but it *is* possible to represent them in Stellar!





SpArch, 2020

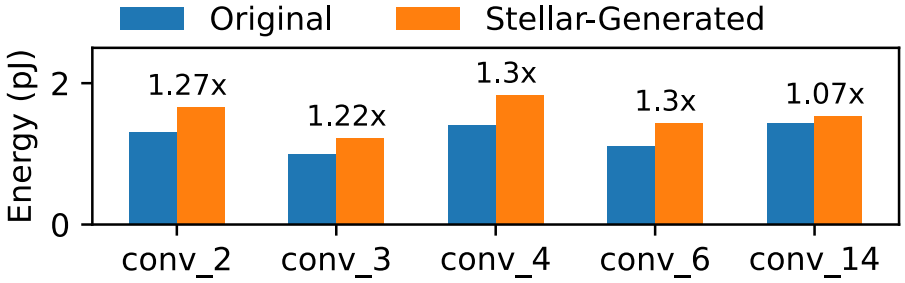
Limitations

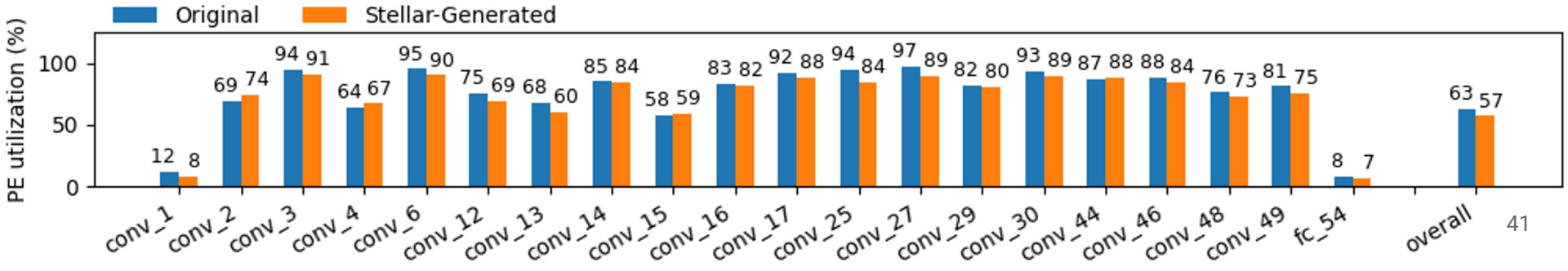
- “Affine” reductions vs “tree-reductions”
- Stellar can easily represent “affine” dataflow transformations
- Stellar isn’t great at representing recursive, or “tree-based” transformations
  - ...but it *is* possible to represent them in Stellar!

- Caches vs scratchpads
- Stellar generates explicitly-managed scratchpads
  - L2 caches backing up the scratchpads and CPU are supported
- But some sparse accelerator works propose new caches with novel eviction/prefetching policies
  - Novel caches not yet supported in Stellar

Results: Performance, Area, Power

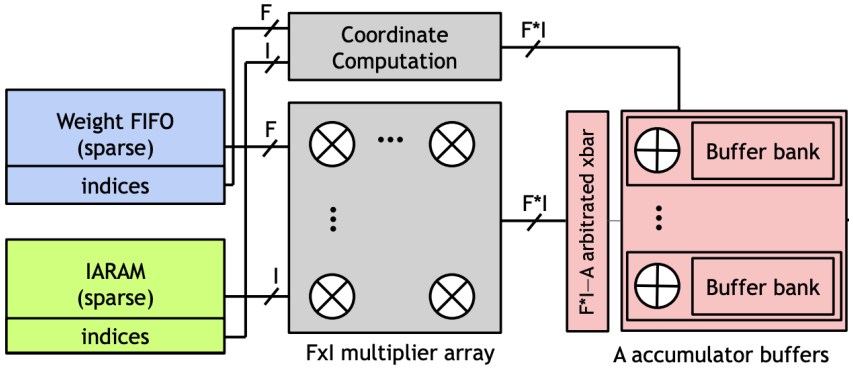
- Stellar-generated accelerators achieve comparable performance and area consumption to hand-designed accelerators
- For example, compared to the dense DNN accelerator Gemmini:
  - 13% more area
  - 90% of performance

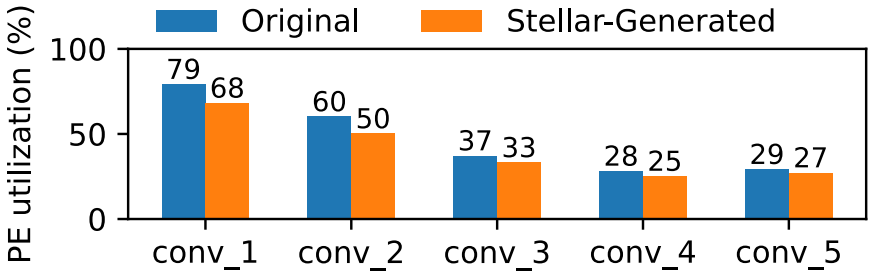




Results: Performance, Area, Power

- Stellar also generates performance sparse accelerators
- For example, SCNN:
  - 4D PE topology
    - 2D spatial array of 2D spatial arrays
  - Both sparse and dense data
    - Inputs, weights, and outputs are sparse
    - Partial sum accumulations are dense
- Achieves 83-94% of handwritten SCNN's performance on various DNN layers





Conclusion and Future Work

- Stellar enables faster accelerator design, evaluation, and RTL generation
  - Attempts to maintain a strong separation of concerns
- Open-sourced:
  - github.com/hngenc/stellar

- Future work:
  - Caches
    - What kinds of abstractions would cover the full space of sparse cache design?
  - Non-affine reductions
    - Including recursion
  - Search strategies over our design-space

Hasan Genc

Prashanth Ganesh

Hansung Kim

Sophia Shao

Questions?