http://inst.eecs.berkeley.edu/~ee290-2/sp21/

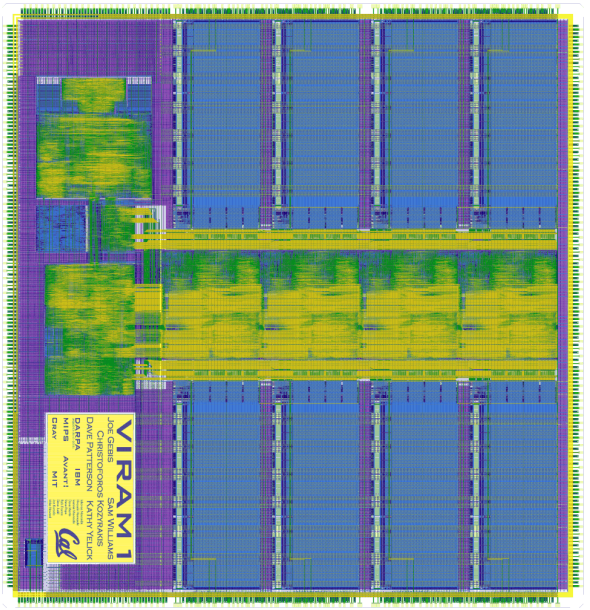
Hardware for Machine Learning

Lecture 16: Other Operators & Near-Data Processing

Sophia Shao

The Berkeley Intelligent RAM (IRAM) Project

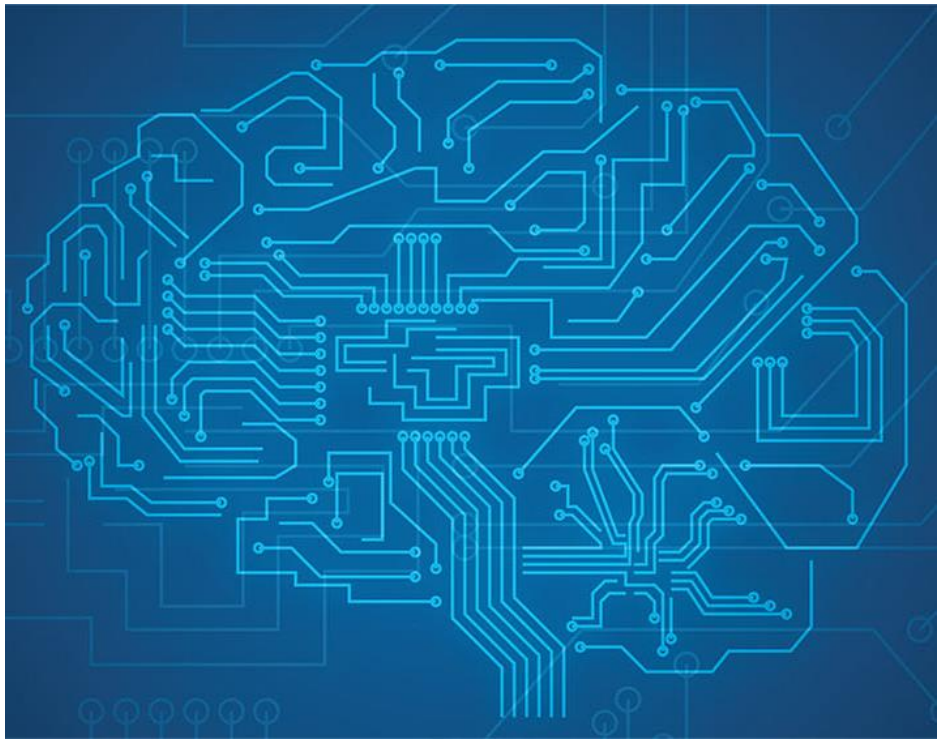
The Berkeley Intelligent RAM (IRAM) project seeks to understand the entire spectrum of issues involved in designing general-purpose computer systems that integrate a processor and DRAM onto a single chip - from circuits, VLSI design and architectures to compilers and operating systems. IRAM should offer several advantages over today's solutions, including considerably reduced latency and dramatically increased bandwidth to main memory, reduced power and energy consumption, and reduced space and weight for embedded, portable, desktop, and parallel computer systems.



http://iram.cs.berkeley.edu/

Review

- Core computation in DNN
- Execution order of the core computation
- Hardware realization of the core computation
- Mapping DNNs to hardware
- Data transfer mechanisms across storage hierarchy
- Sparsity in DNNs
- This Lecture: Codesign example
  - Implications on HW
  - HW-aware design:
    - pruning, compact network design, neural architecture search

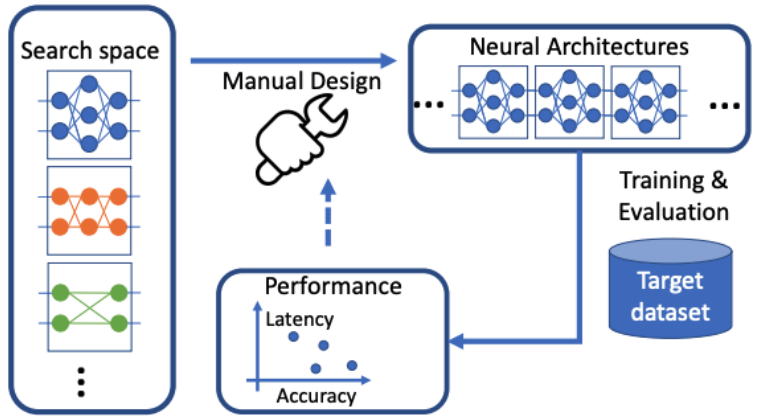


Codesign

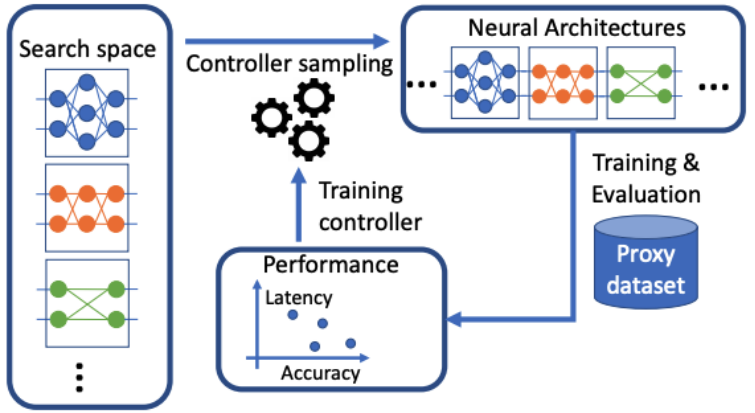
- Implications on HW
- HW-Aware Design
  - Pruning
  - Compact Network
  - Neural Arch. Search

Neural Architecture Search (NAS)

- Manually determine the network architecture is a tedious process.
  - Large number of hyperparameters, e.g., # of layers, connections between layers, and types of layers
- NAS is proposed to automatically search for the optimal architecture at the cost of more computation.



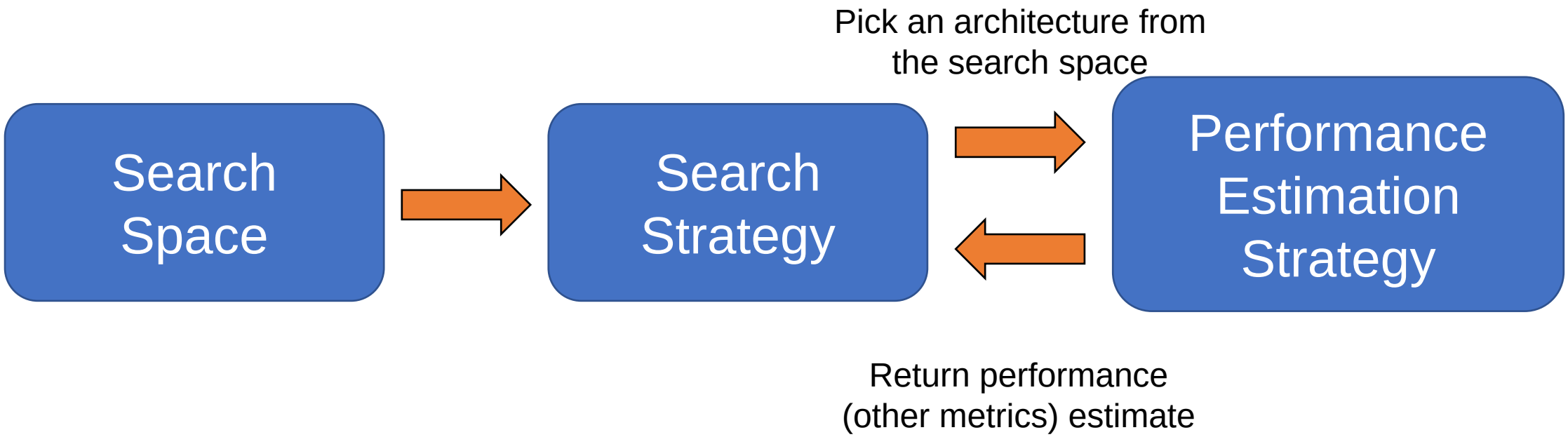
(a) A typical flow of manual ConvNet design.



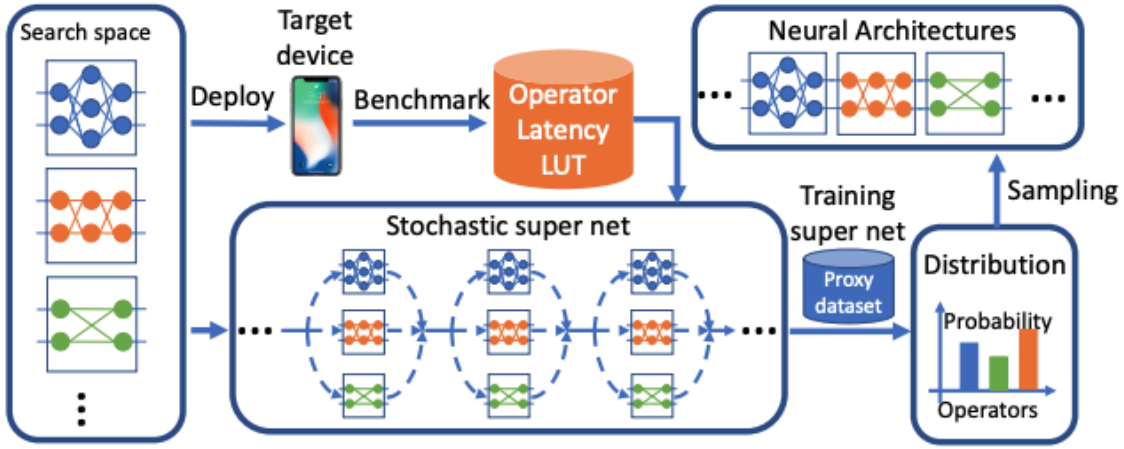
(b) A typical flow of reinforcement learning based neural architecture search.

FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search

Neural Architecture Search (NAS)



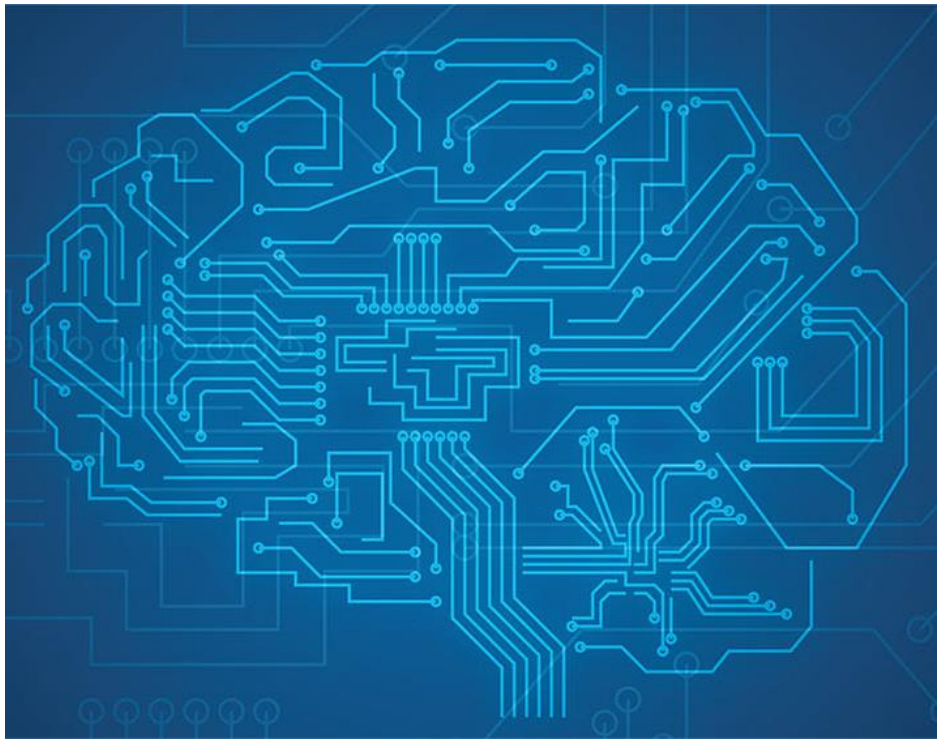
Neural Architecture Search (NAS)



Model	#Parameters	#FLOPs	Latency on iPhone X	Latency on Samsung S8	Top-1 acc (%)
FBNet-iPhoneX	4.47M	322M	19.84 ms (target)	23.33 ms	73.20
FBNet-S8	4.43M	293M	27.53 ms	22.12 ms (target)	73.27

Table 5. FBNets searched for different devices.

FBNet: Hardware-Aware Efficient ConvNet Design via Differentiable Neural Architecture Search



Near-Data Processing

- Other Operators:
  - Element-wise
  - Embedding
- Near-Data Processing
  - Overview
  - Case studies

Beyond convolution and matrix multiplication

- So far our discussion has been focusing on conv and matmul.

TABLE I: Recent Architecture Research in Deep Learning

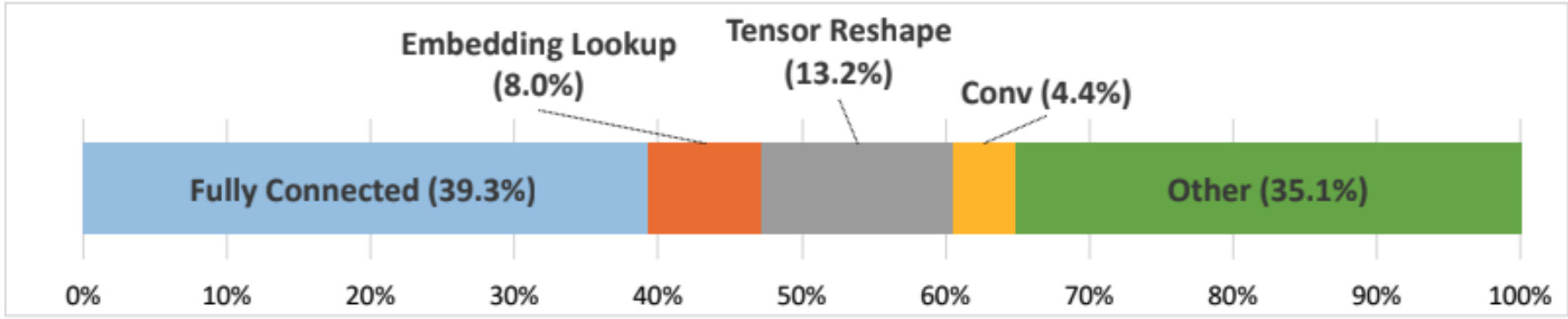
There are a wide variety of deep learning problems which are largely untouched by current work.

Category	Feature	[8]	[9]	[10]	[11]	[12]	[14]	[21]	[24]	[26]	[35]	[38]	[39]	[40]	[44]	[47]	[49]	Fathom
Neuronal Style	Fully-connected	×	×	×			×	×	×	×	×	×	×	×		×		×
	Convolutional	×		×	×	×	×	×		×		×		×			×	×
	Recurrent								×						×			×
Layer Depth	(Maximum)	4	4	3	3	5	16	7	3	13	6	9	4	26	2	5	5	34
Learning Task	Inference	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×	×
	Supervised		×		×					×	×	×	×		×			×
	Unsupervised																	×
	Reinforcement																	×
Application Domain	Vision	×		×	×	×	×	×	×	×	×	×	×	×			×	×
	Speech								×	×								×
	Language Modeling								×	×			×		×			×
	Function Approximation		×													×		×

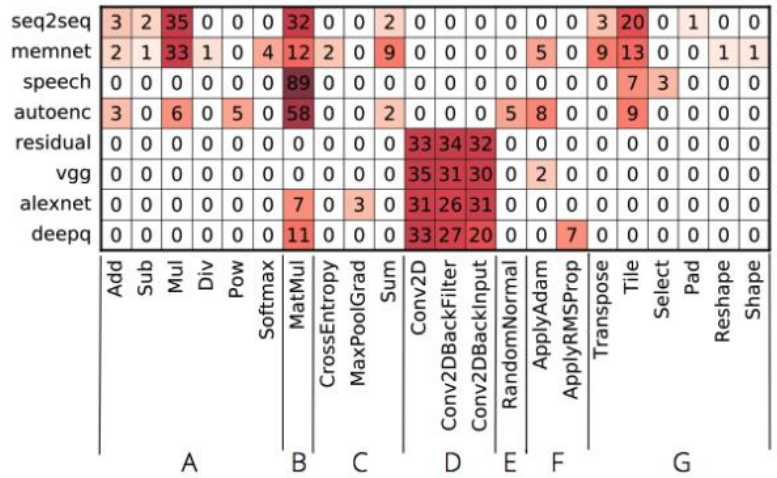
Fathom: Reference Workloads for Modern Deep Learning Methods

Beyond convolution and matrix multiplication

- So far our discussion has been focusing on conv and matmul.



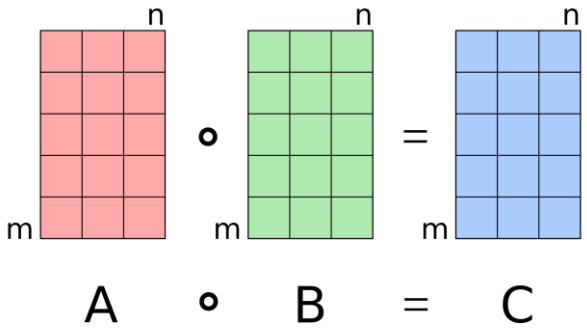
AI at Facebook Datacenter Scale



Group	Operation Class
A	Elementwise Arithmetic
B	Matrix Operations
C	Scatter/Gather
D	Convolution
E	Stochastic Methods
F	Optimization
G	Data Movement

Fathom: Reference Workloads for Modern Deep Learning Methods

Element-wise computation



- A, B, C of the same dimensions.
  - 1 op/output (vs. RSC ops/output in conv.)
- E.g., Long short-term memory
  - Matrix vector multiply
  - Element-wise multiplication/addition

$$
\begin{aligned}
f_t &= \sigma_g(W_f x_t + U_f h_{t-1} + b_f) \\
i_t &= \sigma_g(W_i x_t + U_i h_{t-1} + b_i) \\
o_t &= \sigma_g(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma_h(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \circ c_{t-1} + i_t \circ \tilde{c}_t \\
h_t &= o_t \circ \sigma_h(c_t)
\end{aligned}
$$

Residual Addition

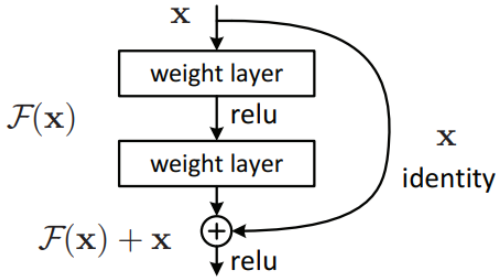
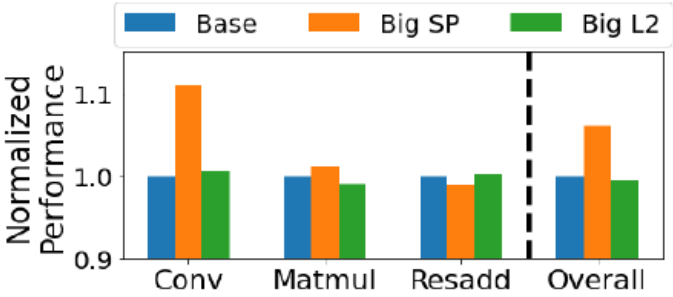


Figure 2. Residual learning: a building block.

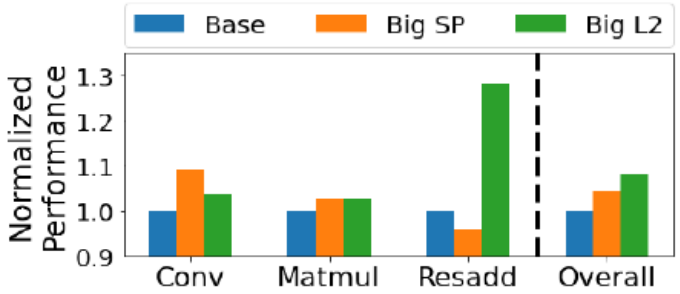
- "Identity shortcut connections add neither extra parameter nor computational complexity."

Config Name	Scratchpad (per core)	Accumulator (per core)	L2 Cache
Base	256 KB	256 KB	1 MB
BigSP	512 KB	512 KB	1 MB
BigL2	256 KB	256 KB	2 MB

(a) Resource contention SoC configurations



(b) Performance of single-core SoCs.



(c) Performance of dual-core SoCs.

- Residual additions, which have virtually no data re-use and are memory-bound operations, exhibit no speedup when increasing the scratchpad memory size. Instead, they exhibit a minor 1%-4% slowdown, due to increased cache thrashing.

Deep Residual Learning for Image Recognition

Embedding Layer

- Embedding: solving representation mapping problem.
  - From a high dimension, i.e., sparse dimension
    - e.g., size of vocabulary, total number of movies
  - To a significantly smaller dimension of meanings, i.e., dense dimension

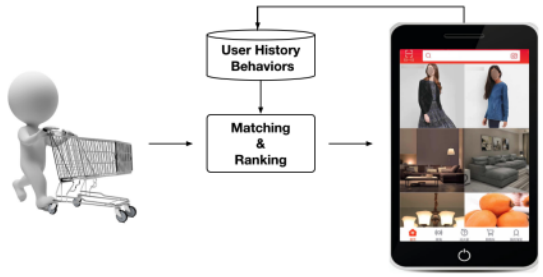


Figure 1: Illustration of running procedure of display advertising system in Alibaba, in which user behavior data plays important roles.

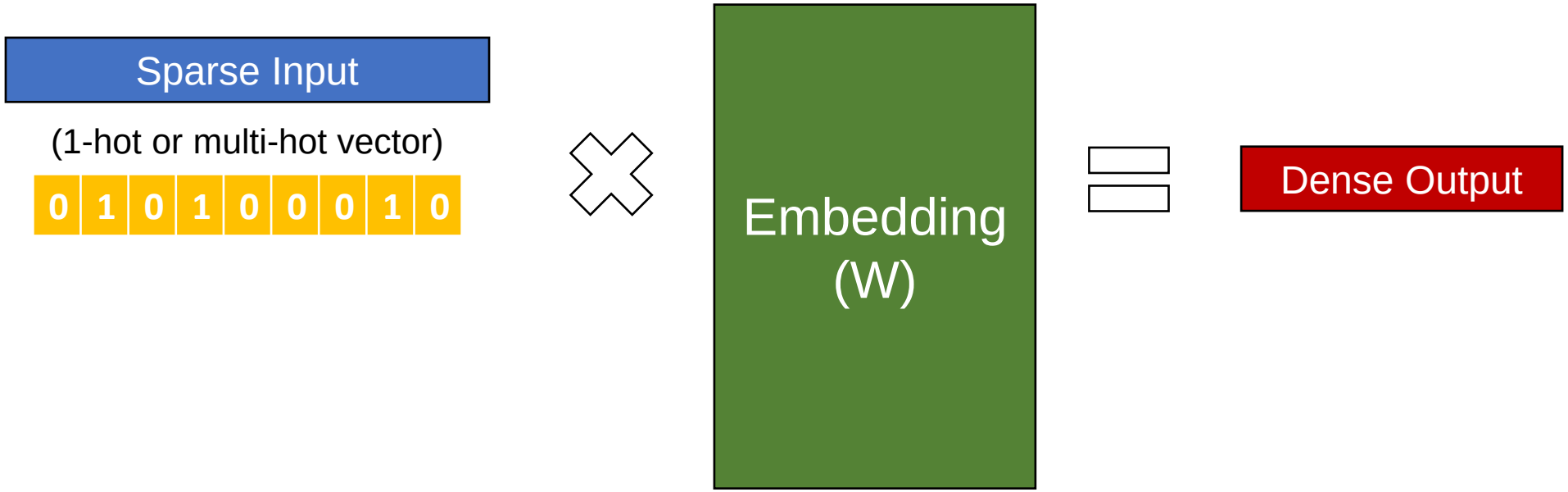
Table 1: Statistics of feature sets used in the display advertising system in Alibaba. Features are composed of sparse binary vectors in the group-wise manner.

Category	Feature Group	Dimemsionality	Type	#Nonzero Ids per Instance
User Profile Features	gender	2	one-hot	1
	age_level	~ 10	one-hot	1
	...	...	...	...
User Behavior Features	visited_goods_ids	~ 10^9	multi-hot	~ 10^3
	visited_shop_ids	~ 10^7	multi-hot	~ 10^3
	visited_cate_ids	~ 10^4	multi-hot	~ 10^2
Ad Features	goods_id	~ 10^7	one-hot	1
	shop_id	~ 10^5	one-hot	1
	cate_id	~ 10^4	one-hot	1
	...	...	...	...
Context Features	pid	~ 10	one-hot	1
	time	~ 10	one-hot	1
	...	...	...	...

Deep Interest Network for Click-Through Rate Prediction

Embedding Layer

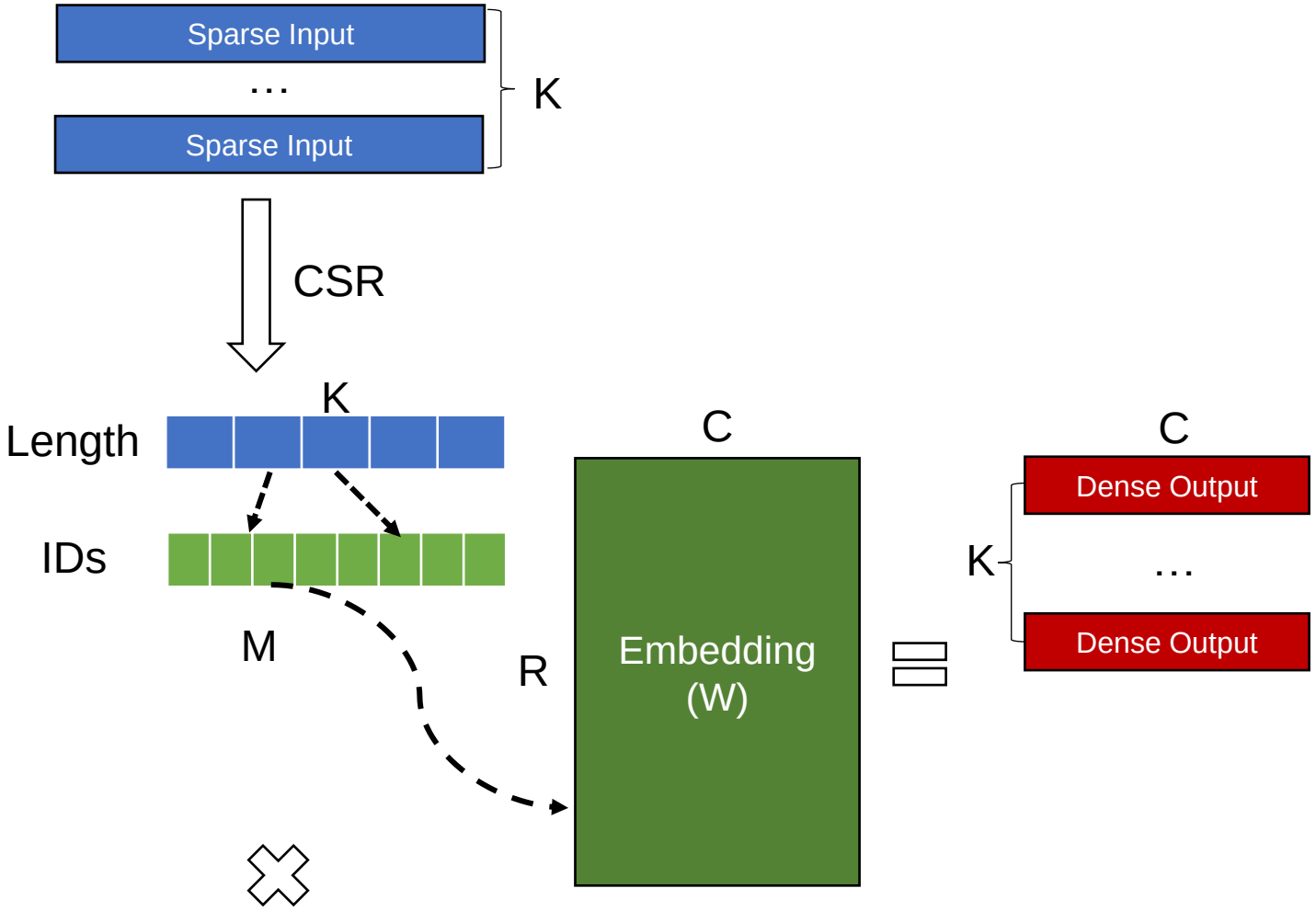
- Core operation: matrix-vector (or matrix-matrix for batched), though not efficient.
- In deployment: memory lookup based on the input vector, extremely low compute density.



Embedding Layer

Algorithm 1 SparseLengthsSum (SLS) pseudo-code

```
1:  Emb ← Embedding Table: R(~millions) x C(~tens)
2:  Lengths ← Vector: K                          ▷ slices of IDs
3:  IDs ← Vector: M (~tens)                      ▷ non-contiguous
4:  Out ← Vector : K × C
5:
6:  CurrentID = 0; OutID = 0
7:  procedure SLS(Emb, Lengths, IDs)
8:      for L in Lengths do
9:          for ID in IDS[CurrentID: CurrentID+L]: do
10:             Emb_vector = Emb[ID]
11:             for i in range(C): do
12:                 Out[OutID][i]+ = Emb_vector[i]
13:             end for
14:         end for
15:         OutID = OutID+1; CurrentID = CurrentID+L
16:     end for
17:     return Out
18: end procedure
```



The Architectural Implications of Facebook's DNN-based Personalized Recommendation

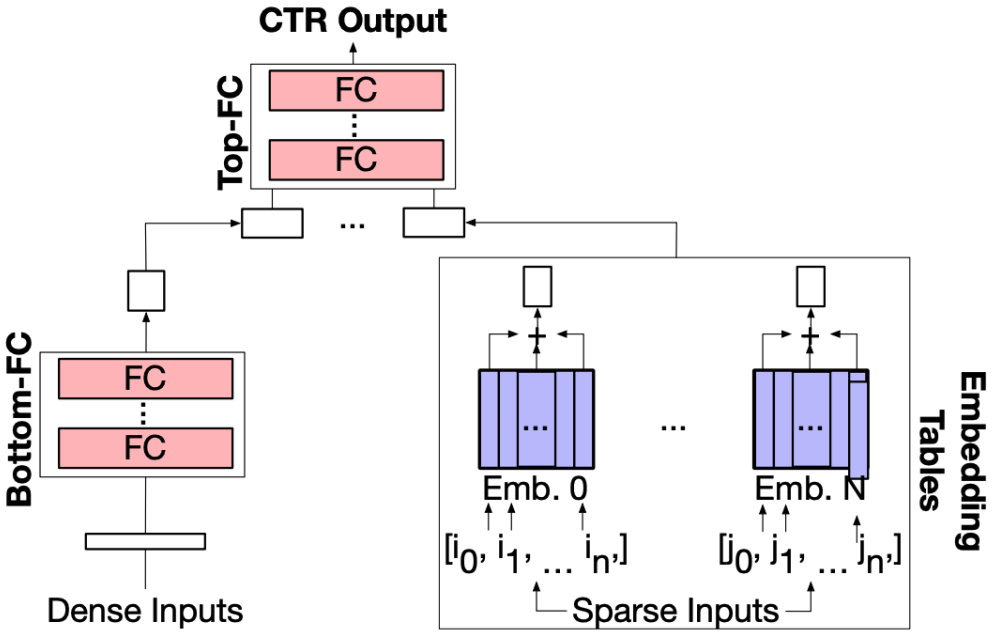
DNN Compute on Sparse Data

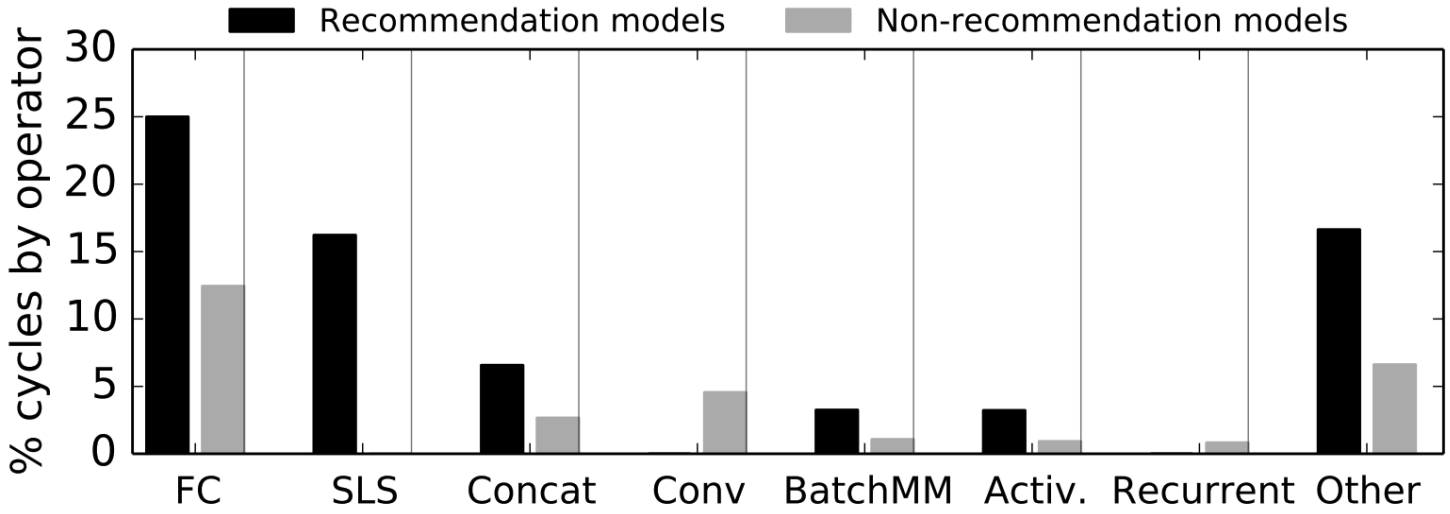
- Skip ineffectual computation when either W or IA is zero.
- Approaches:

	Single-Operand Sparse	Two-Operand Sparse
Align W & IA	Indirection (e.g., Cnvlutin, …)	Intersection (e.g., SNAP, …)
Align OA	Arbitration (e.g., SCNN)	

Deep-Learning Recommendation Model

- A diverse set of operators.





The Architectural Implications of Facebook's DNN-based Personalized Recommendation

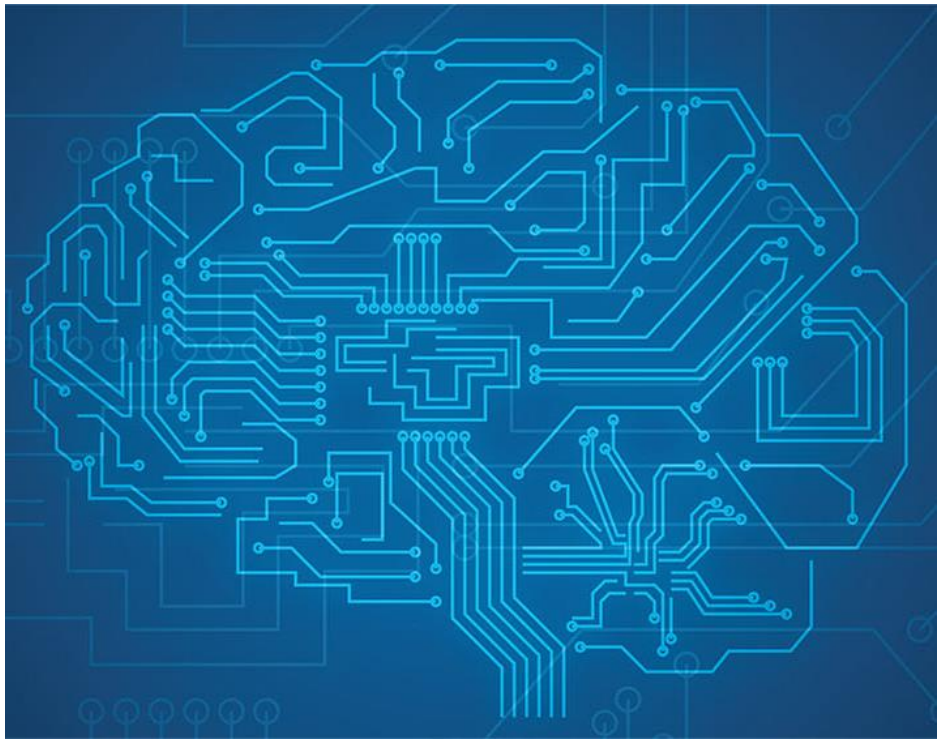
Administrivia

- Spring break next week!
  - Stay safe. Stay healthy.

- Guest lectures coming up:
  - Yaqi Zhang, Principal Engineer, SambaNova Systems
    - SambaNova architecture
  - Vijay Janapa Reddi, Associate Professor, Harvard/MLPerf Inference Co-Chair
    - Deep Learning Benchmarking: MLPerf
  - Cliff Young, Software Engineer, Google
    - TPU and beyond CMOS
  - Brian Zimmer, Senior Research Scientist, NVIDIA
    - Challenges with Analog and In-Memory Computing

Final Project

- Project Proposal (due Friday, 3/19)
  - Find 1-2 relevant research papers of your topic.
  - Write a summary of that research paper.
  - Describe how you hope to see or adapt ideas from it and how you plan to extend or improve it in your final project.
  - Describe infrastructure you plan to use.
  - Project plan: describe milestones to achieve every two weeks:
    - Checkpoint 1 (Week of 4/5)
    - Checkpoint 2 (Week of 4/19)
    - Final presentation (5/3)
    - Final report (5/7)



Near-Data Processing

- Other Operators:
  - Element-wise
  - Embedding
- Near-Data Processing
  - Overview
  - Case studies

Characteristics of non-conv operators:

- Element-wise operations:
  - 1 op for 2 loads and 1 store
- Embedded layer:
  - 1 op (+) for >= 2 loads and 1 store
- These behaviors are similar to other non-DL applications:
  - E.g., database and memory management

Near-Data Processing

- Is an overloaded term ☺
- It could mean:
  - Near-DRAM processing
  - Near-cache processing
  - Logic-in-memory processing
  - …
- In today's discussion, we will present some examples of specialized hardware that
  - Acts as a co-processor in a system
  - Has direct accesses to DRAM without going through the cache hierarchy

Near-Data Processing

- In today's discussion, we will present some examples of specialized hardware that
  - Acts as a co-processor in a system
  - Has direct accesses to DRAM without going through the cache hierarchy
- Benefits:
  - Control: avoid penalty for data-dependent control streams
  - Compute: opportunities to specialized for required operations
  - Data movement: no need to move data through layers of cache hierarchy

LINQits: Big Data on Little Clients, ISCA'13

Eric S. Chung†, John D. Davis†, and Jaewon Lee‡

†*Microsoft Research Silicon Valley*

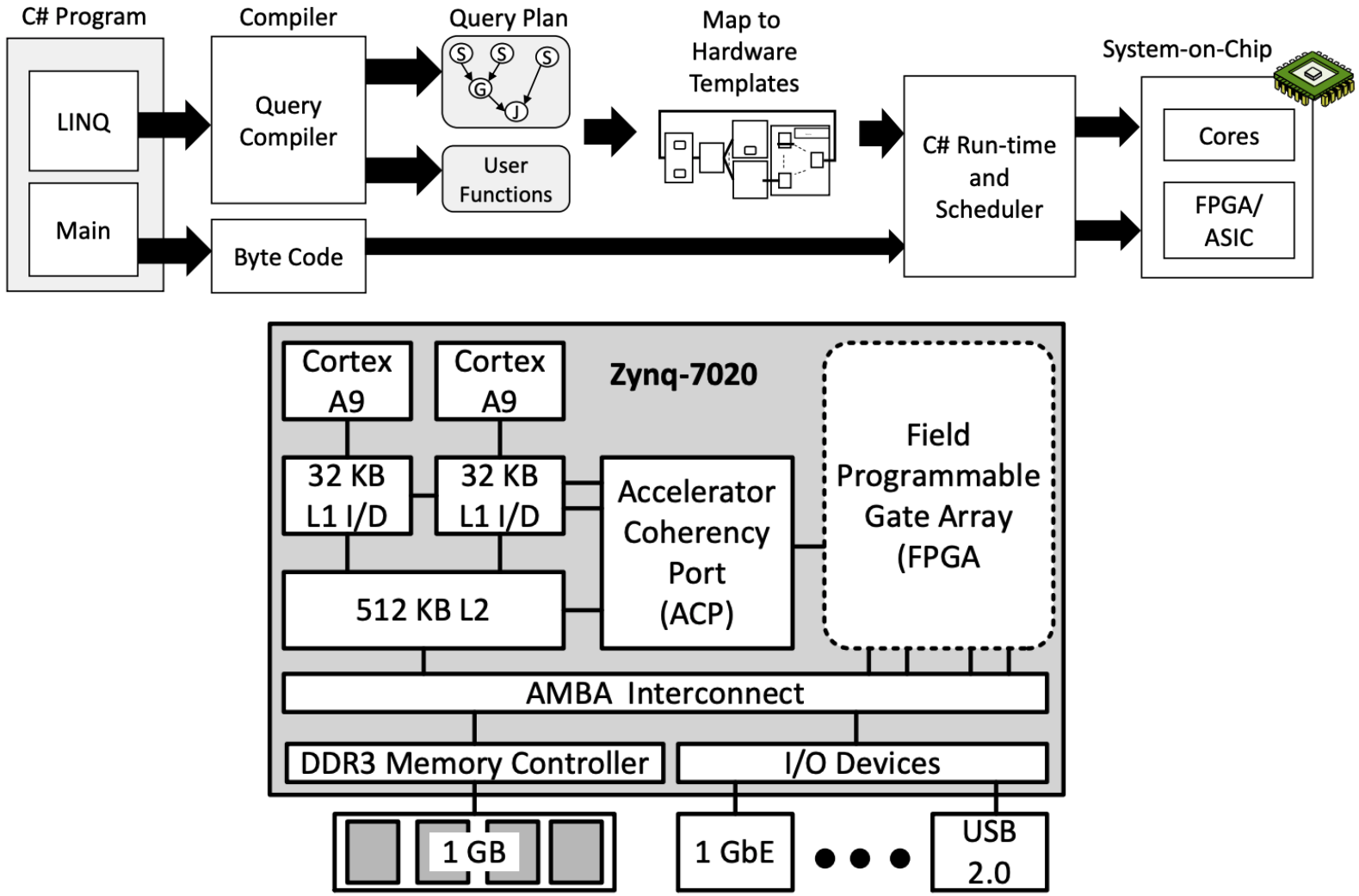
‡*Department of Computer Science and Engineering, POSTECH*

- Accelerates a domain-specific query language, LINQ (Language Integrated Query).

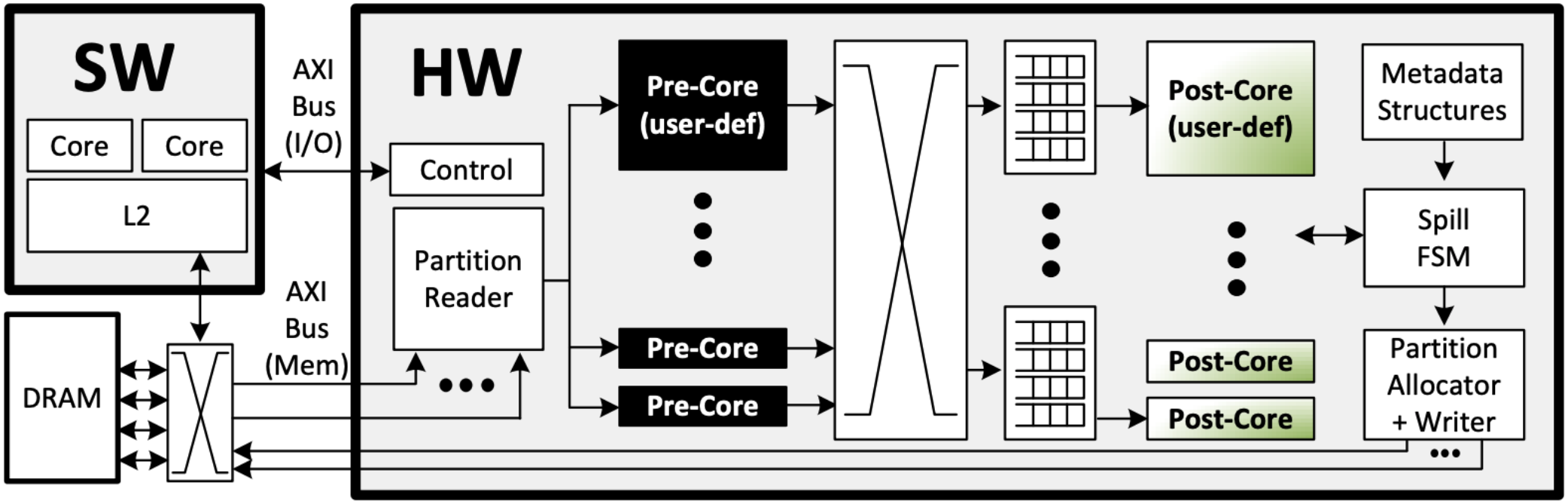
Operation	Meaning
`Where`	(Filter) Keep all values satisfying a given property.
`Select`	(Map) Apply a transformation to each value in the collection.
`Aggregate`	(Fold, Reduce) Combine all values in the collection to produce a single result (e.g., max).
`GroupBy`	Create a collection of collections, where the elements in each inner collection all have a common property (*key*).
`OrderBy`	(Sort) Order the elements in the collection according to some property (*key*).
`SelectMany`	Generates a collection for each element in the input (by applying a function), then concatenates the resulting collections.
`Join`	Combine the values from two collections when they have a common property.

```
1.  // Object-oriented syntax with anonymous functions
2.  var results =
3.    SomeCollection
4.      .Where(cust => cust.sales > 100)
5.      .Select(cust => new { cust.zipCode,
6.                            c.paymentType });

7.  // Equivalent SQL-style syntax
8.  var results = from cust in customer
9.          where cust.sales > 100
10.         select new {cust.zipCode, c.paymentType};

11. foreach(var v in results) ... // do work
```

LINQits: Big Data on Little Clients, ISCA'13



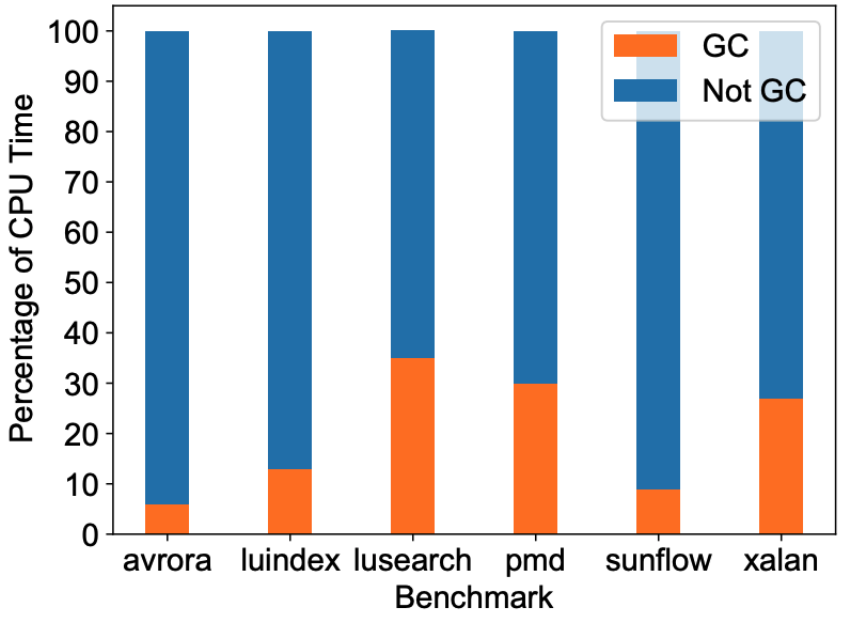
LINQits: Big Data on Little Clients, ISCA'13



Accelerator for Garbage Collection, ISCA'18

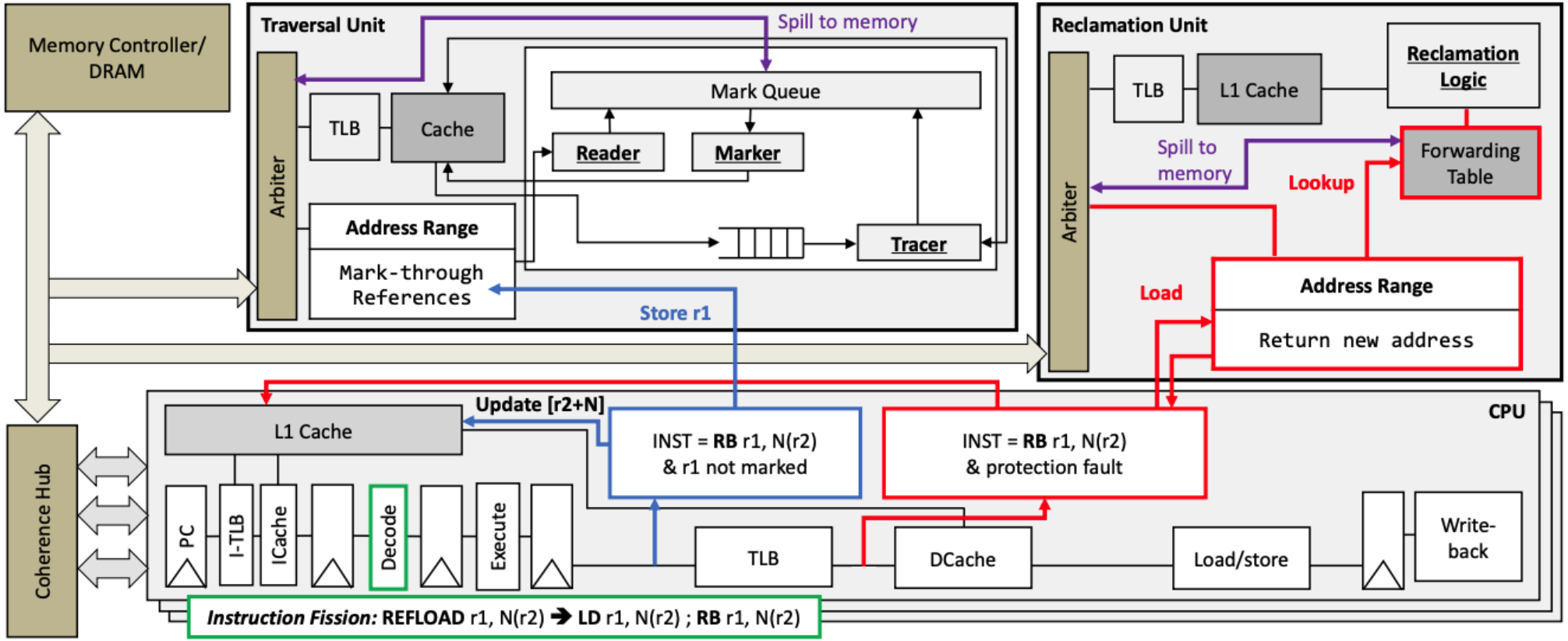
Martin Maas*, Krste Asanović, John Kubiatowicz
University of California, Berkeley
{*maas,krste,kubitron*}*@eecs.berkeley.edu*

- A large number of workloads are written in garbage-collected languages. These applications spend up to 10-35% of their CPU cycles on GC.
- To decrease these overheads by moving GC into a small hardware accelerator that is located close to the memory controller and performs GC more efficiently than a CPU.



Accelerator for Garbage Collection, ISCA'18

- TLB and Cache in accelerators
- Direct interactions with the core.



Accelerator for Range Partitioning, ISCA'2013

Navigating Big Data with High-Throughput, Energy-Efficient Data Partitioning

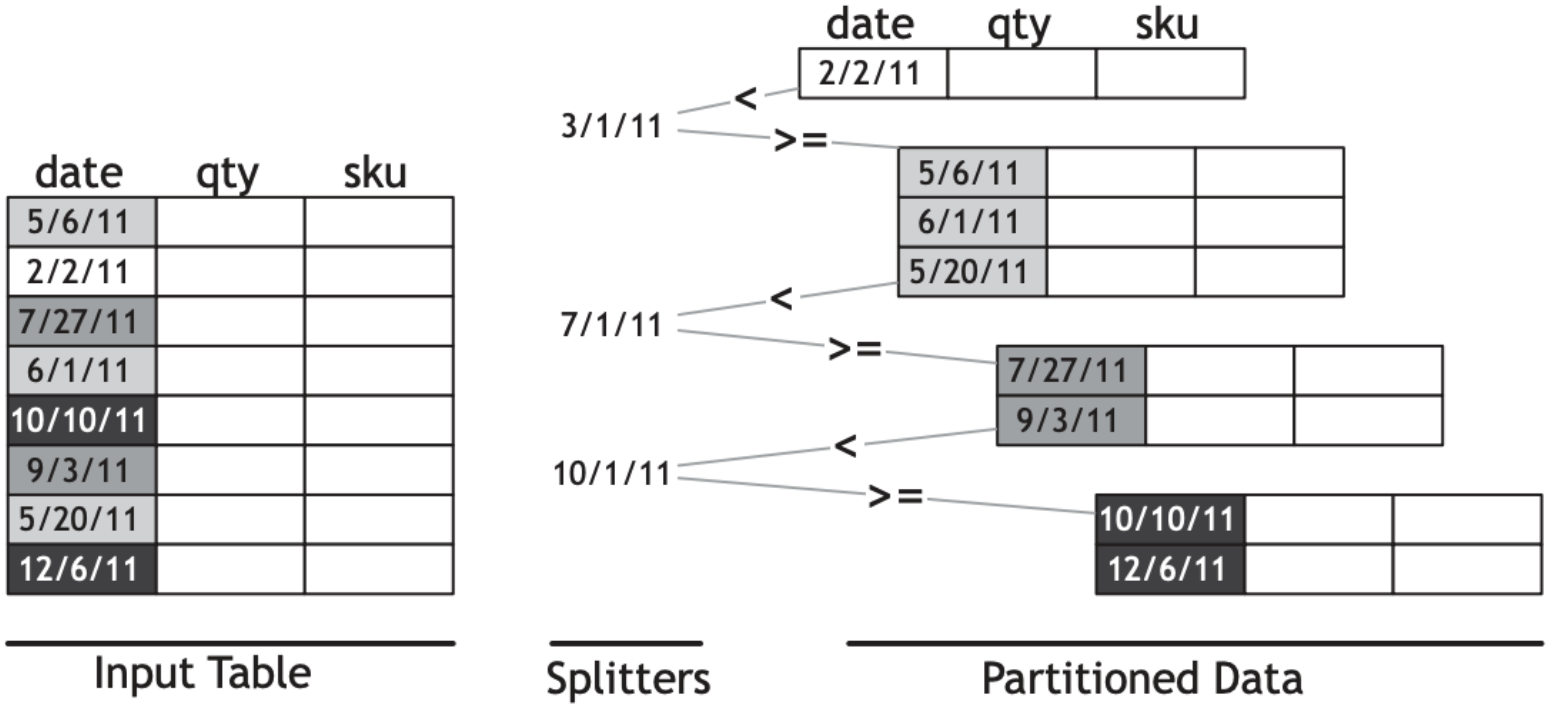
Lisa Wu lisa@cs.columbia.edu
Raymond J. Barker rjb2150@columbia.edu
Martha A. Kim martha@cs.columbia.edu
Kenneth A. Ross kar@cs.columbia.edu

Department of Computer Science
Columbia University
New York, New York

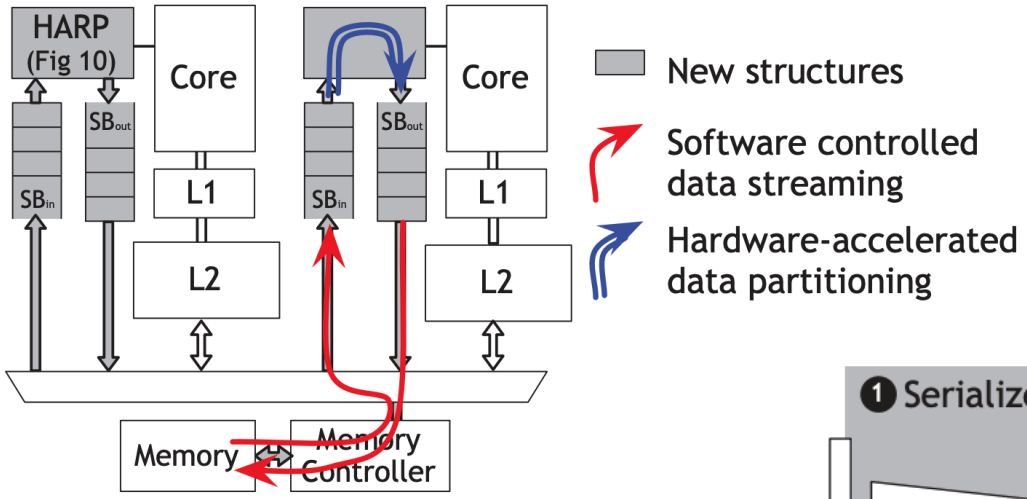
- Another case for database acceleration due to its low compute density.
- Partitioning assigns each record in a large table to a smaller table based on the value of a particular field in the record.

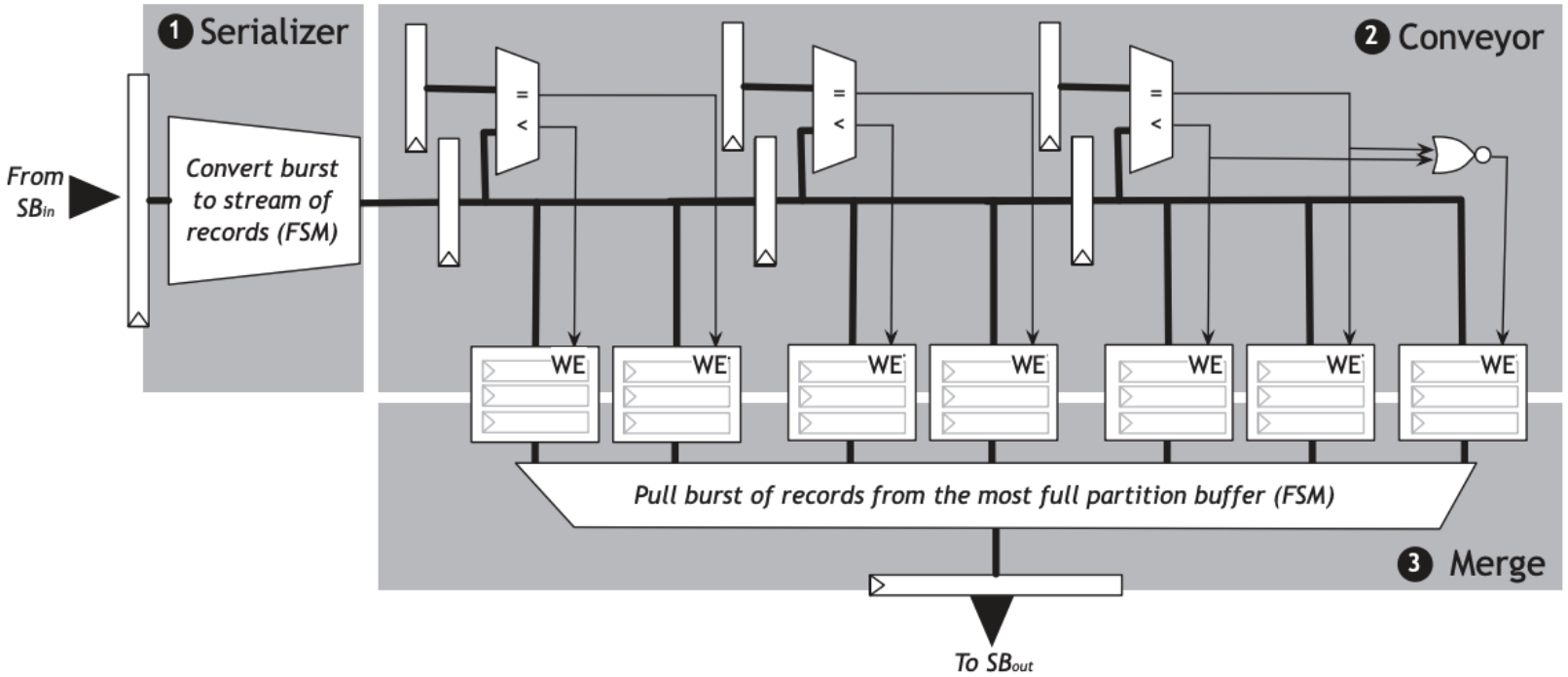
Accelerator for Range Partitioning, ISCA'2013

- An example table of sales records range partitioned by date, into smaller tables. Processing big data one partition at a time makes working sets cache-resident, dramatically improving the overall analysis speed.



Accelerator for Range Partitioning, ISCA'2013





Review

- Core computation in DNN
- Execution order of the core computation
- Hardware realization of the core computation
- Mapping DNNs to hardware
- Data transfer mechanisms across storage hierarchy
- Sparsity in DNNs
- Codesign example
- This Lecture: Other Operators and Near-Data Processing
  - Element-wise, Embedding, DLRM
  - Case studies of near-data processing for low compute density applications.