IBM Research

Accelerating Database Workloads by Software-Hardware-System Co-design

Rajesh Bordawekar and Mohammad Sadoghi

IBM T. J. Watson Research Center

ICDE 2016 Tutorial

Outline

- Acceleration Landscape
  – Answering Why, What, and How to Accelerate?
- Acceleration Opportunities in Database Workloads
  – FPGAs (e.g., Data Streams)
    – System Model
    – Programming Model
    – Representational Model
    – Algorithmic Model
  – GPUs (e.g., Disk-based Databases & Database Utilities)
- Conclusions and Future Directions

Why Hardware Accelerators?

- **Large & complex control units**
  - Roughly 95% of chip resources (transistors) are dedicated to control units
  - **Solution:** Introduce simplified custom processors for specialized tasks
- **Memory wall & Von Neumann bottleneck**
  - Limited bandwidth between CPUs and memory (bandwidth mismatch)
  - Shared physical memory and system bus for both data and code
  - **Solution:** Couple custom processors with private local memory and embedding code as logic
- **Redundant memory accesses**
  - Incoming data is forced to be written to main memory before it is read again by CPUs for processing
  - **Solution:** Channel data directly to custom processors
- **Power Consumption**
  - Higher transistor density together with higher clock speed results in superlinear increase in power consumption and a greater need for heat dissipation
  - **Solution:** Use many low-frequency, but specialized, chips

Caveat: Most existing custom hardware lack flexibility and simplicity granted when using general-purpose processors

How to Accelerate?

- **Data Parallelism**
  - Single instruction, multiple data (SIMD): DB2 BLU, SAP Hana, Oracle Dual-format, Microsoft Columnstore Indexes, MonetDB
  - Data Partitioning
- **Task Parallelism**
  - Executing many concurrent and independent threads over the data
  - Data Replication
- **Pipeline Parallelism**
  - Decomposing a task into sequence of subtasks
- **Co-processor Design**
  - Offloading computation to accelerators (a co-operative computational model)
- **Co-placement Design**
  - Placing accelerator on the path of data (partial computation or best effort computation)

Accelerator Landscape

Graphics Processing Unit (GPU)
Nvidia and AMD GPUs

Commodity

Hardware
Multi-threading

Application-Specific
Integrated Circuits

Specialized

Single-Instruction Multiple Data (SIMD)
SSE,AVX,VSX,
Oracle SPARC DAX

Field-programmable Gate Array (FPGA)
Xilinx and Altera FPGAs

Acceleration Technology Map

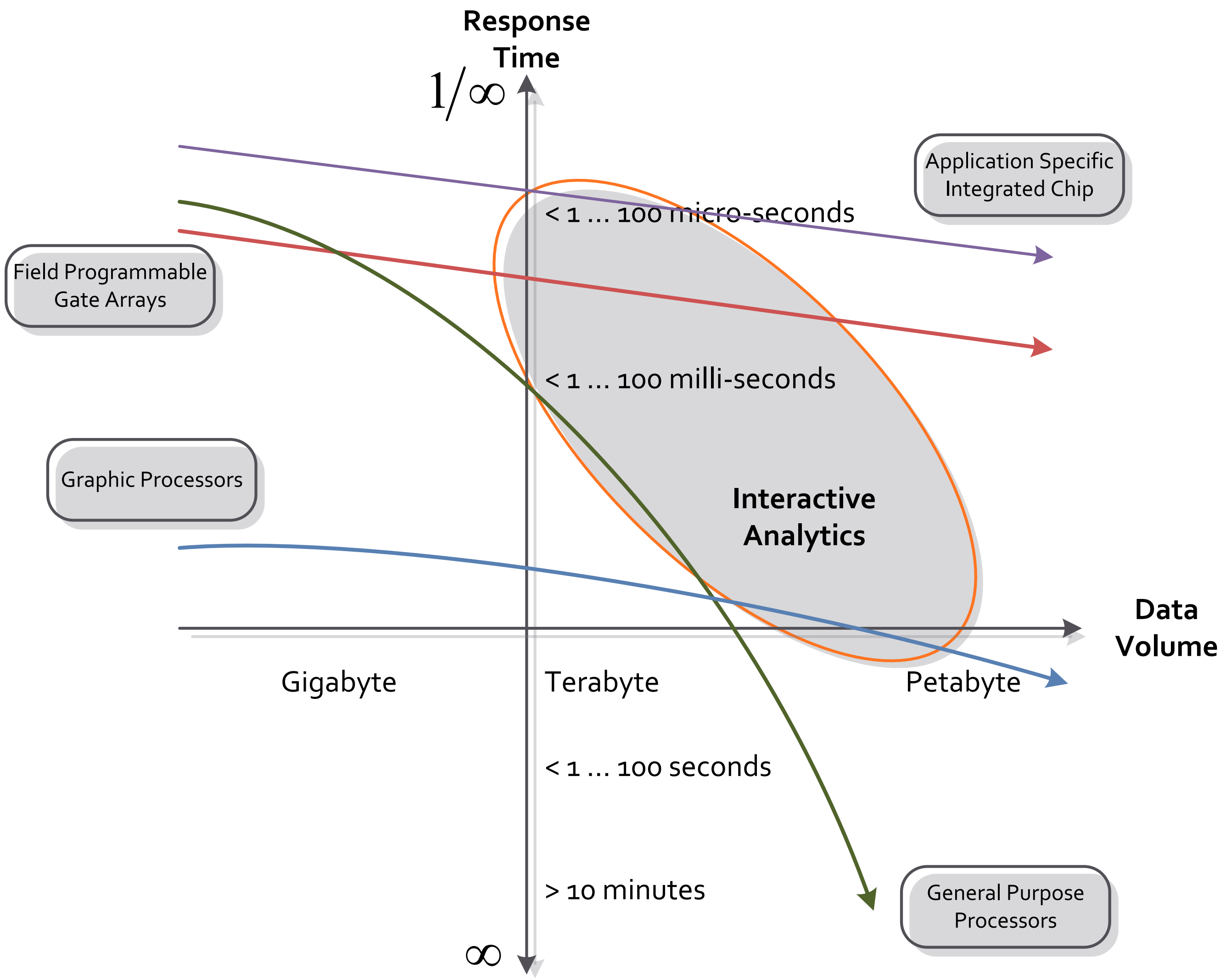


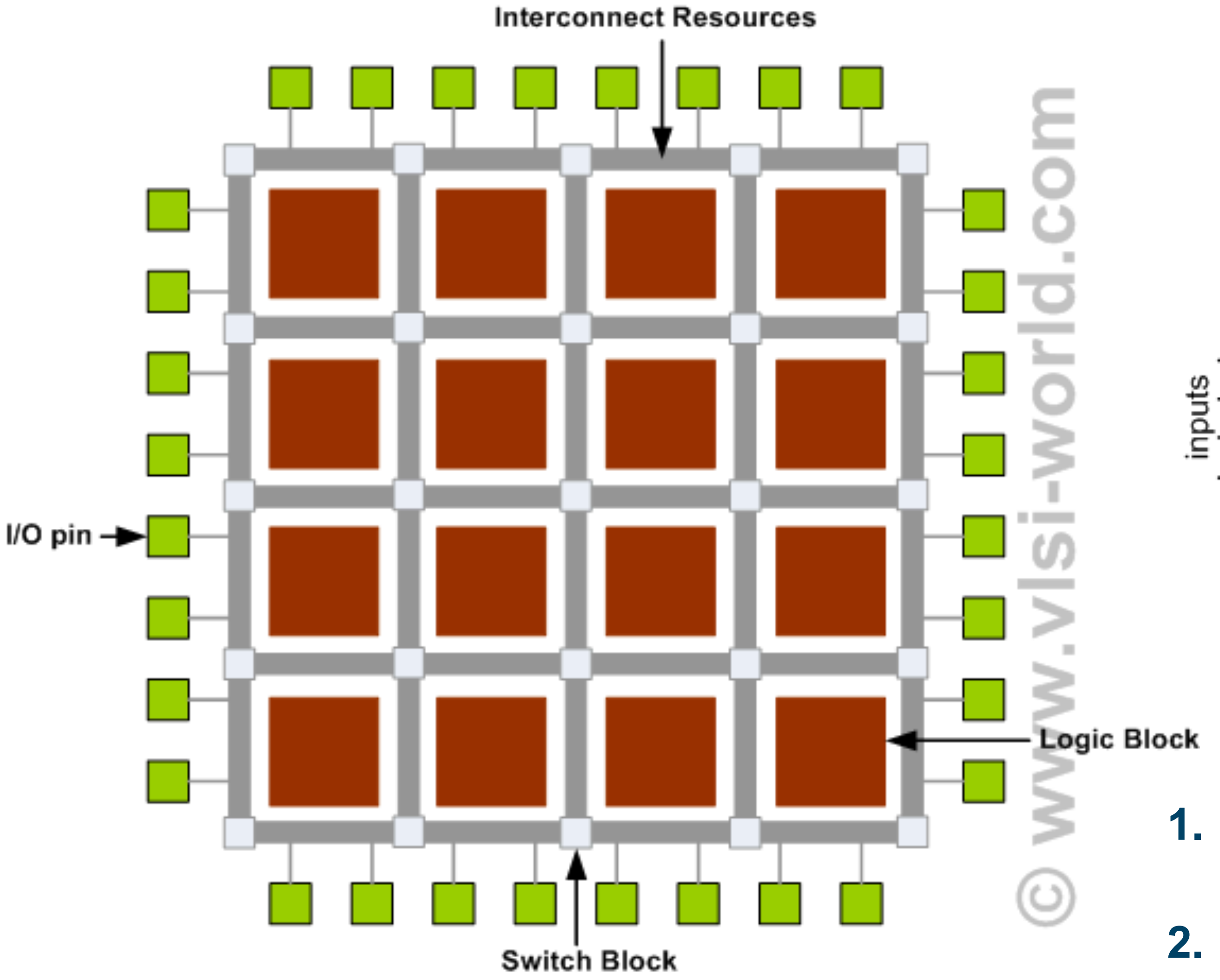
Figure Credits: Najafi, Sadoghi, Jacobsen

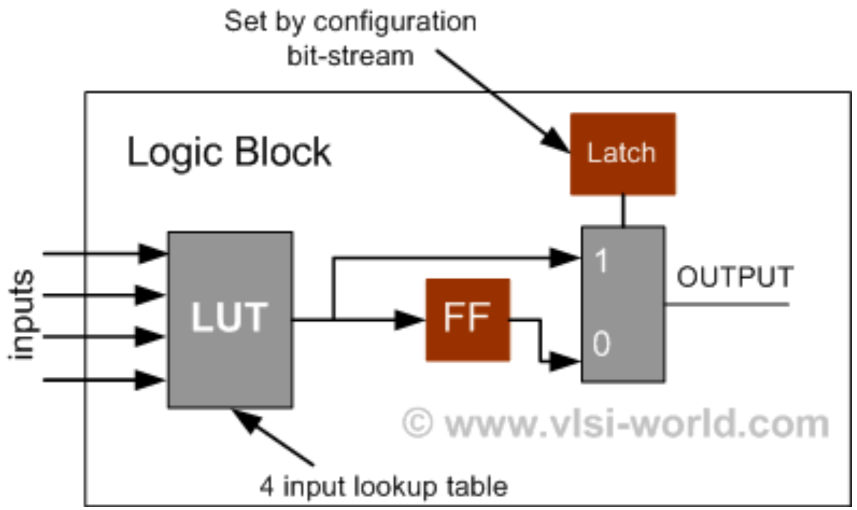
What to Accelerate?

- **Disk-based Databases**
  - OLTP and OLAP
  - Relational, MOLAP, NoSQL (e.g., Graph, RDF/XML/JSON Databases)
- **In-memory Databases**
  - Integrated (OLAP and OLTP) Systems, Embedded Databases
  - Pure OLAP Systems, NoSQL (e.g., Graph, RDF/XML Databases)
- **Streaming Data Processing Systems**
  - SQL and Complex-Event Processing
- **Database Utilities**
  - Compression, Encoding/Decoding, Statistics Generation, Query Plan Generation
- Database Extensions
  - Analytics (e.g., Top-K Queries, Time-series, Clustering, Association Rules, Spatial)
- Distributed Data Processing Systems (Disk-based and In-memory)
  - Distributed Key-Value Middleware (e.g., Memcached)
  - MapReduce Systems (e.g., Hadoop with HDFS)

FPGA Acceleration
(Module I)

What is an FPGA?





1. **Compute:** Configurable logic blocks (CLBs) consisting of Lookup Tables (LUTs) + SRAM
2. **Memory:** Registers, Distributed RAMs, and Block RAMs
3. **Communication:** Configurable interconnects

FPGA Design Flow

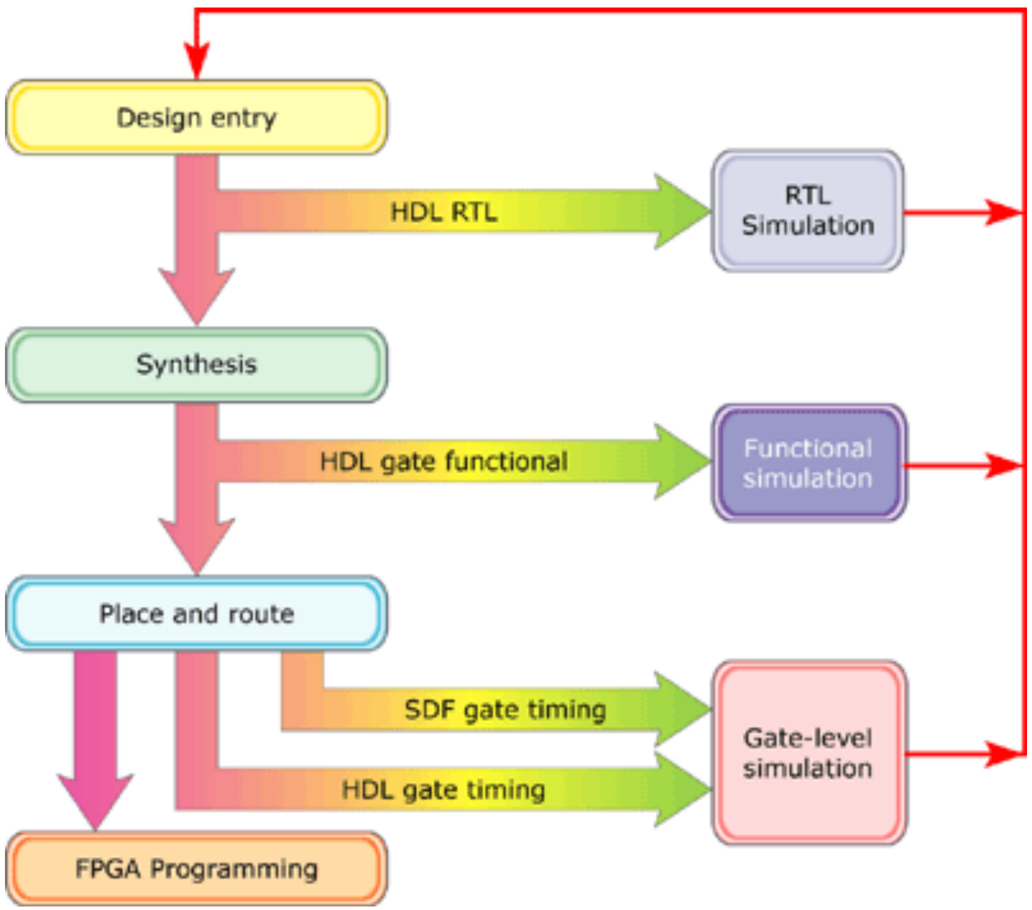


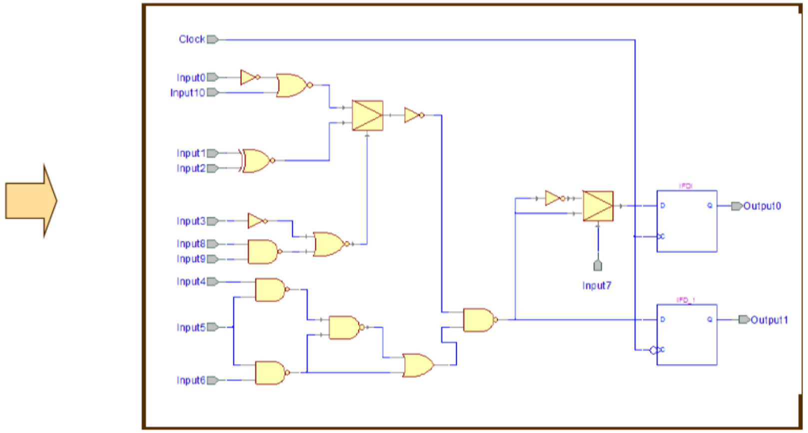
Figure Credits: http://www.design-reuse.com

FPGA Programming Process

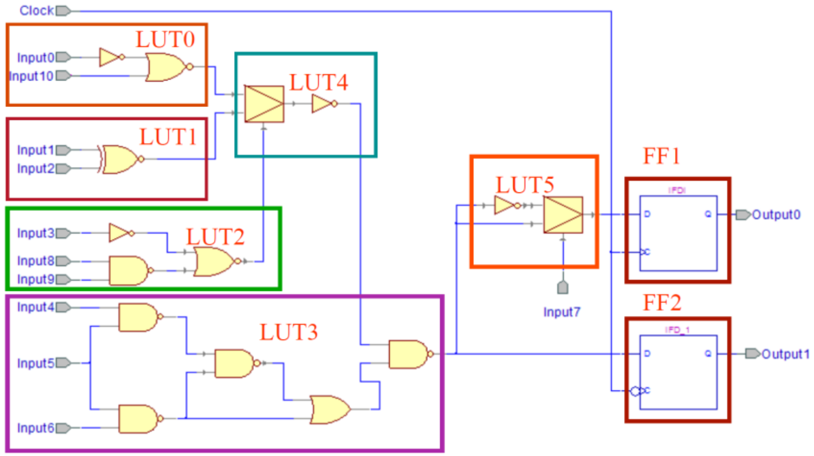
(1) Logic Synthesis

VHDL description

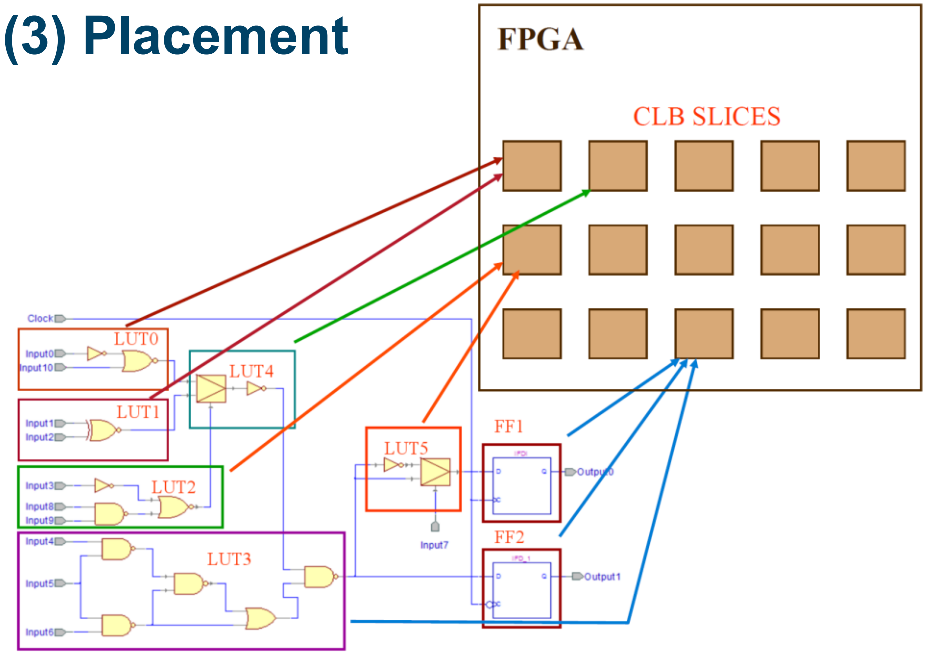
```
architecture MLU_DATAFLOW of MLU is

signal A1:STD_LOGIC;
signal B1:STD_LOGIC;
signal Y1:STD_LOGIC;
signal MUX_0, MUX_1, MUX_2, MUX_3: STD_LOGIC;

begin
        A1<=A when (NEG_A='0') else
                not A;
        B1<=B when (NEG_B='0') else
                not B;
        Y<=Y1 when (NEG_Y='0') else
                not Y1;

        MUX_0<=A1 and B1;
        MUX_1<=A1 or B1;
        MUX_2<=A1 xor B1;
        MUX_3<=A1 xnor B1;

        with (L1 & L0) select
                Y1<=MUX_0 when "00",
                        MUX_1 when "01",
                        MUX_2 when "10",
                        MUX_3 when others;

end MLU_DATAFLOW;
```



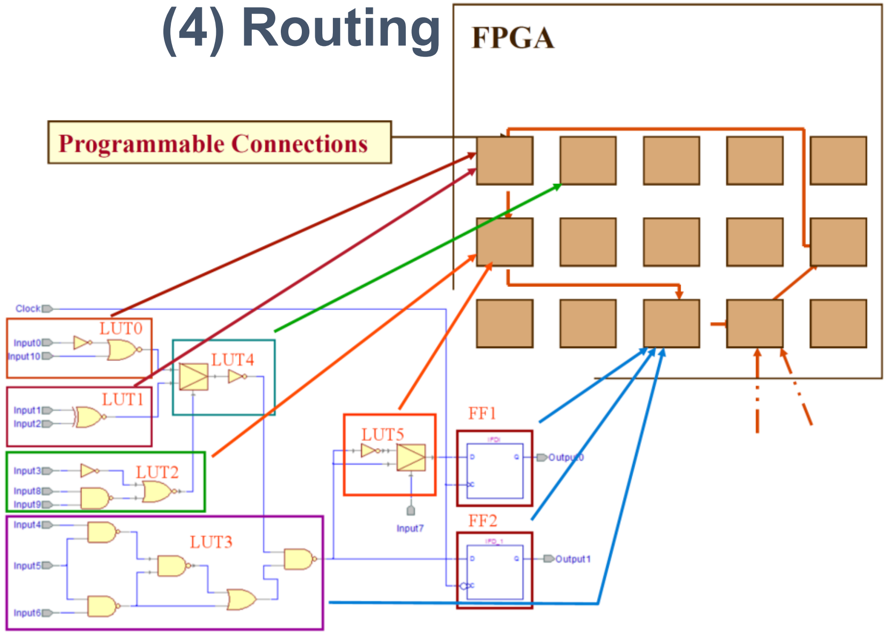
(2) Mapping



(3) Placement



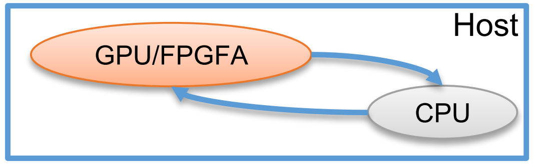
(4) Routing



Acceleration Design Space

System Model

Co-placement Design



Co-processor Design

Programming Model

Hardware Description Language (VHDL, Verilog)

Higher-level Language (OpenCL & Streams-C)

Static Compiler (Glacier)

Dynamic Compiler (FQP)

Representational Model

Static Circuits

Parametrized Circuits (Skeleton Automata, fpga-ToPSS, OPB, FQP, Ibex, IBM Netezza)

Parameterized Segments (FQP)

Temporal/Spatial Instructions (Q100)

Parametrized Topology (FQP)

Algorithmic Model

Indexing (Propagation)

Bi-directional Flow (Handshake Join)

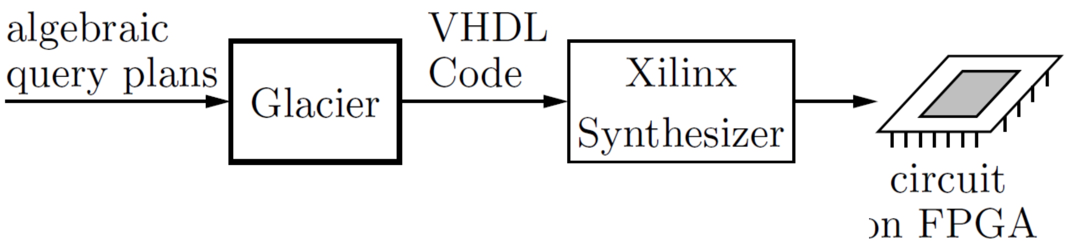
Multi-query Optimization (Rete-like Network)

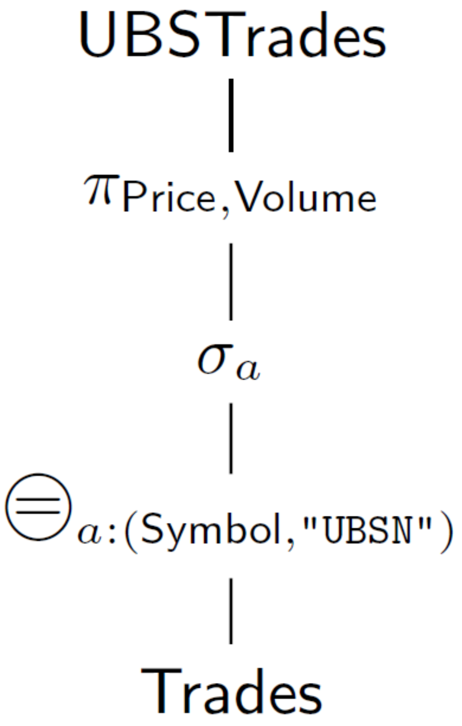
Top-down Flow (SplitJoin)

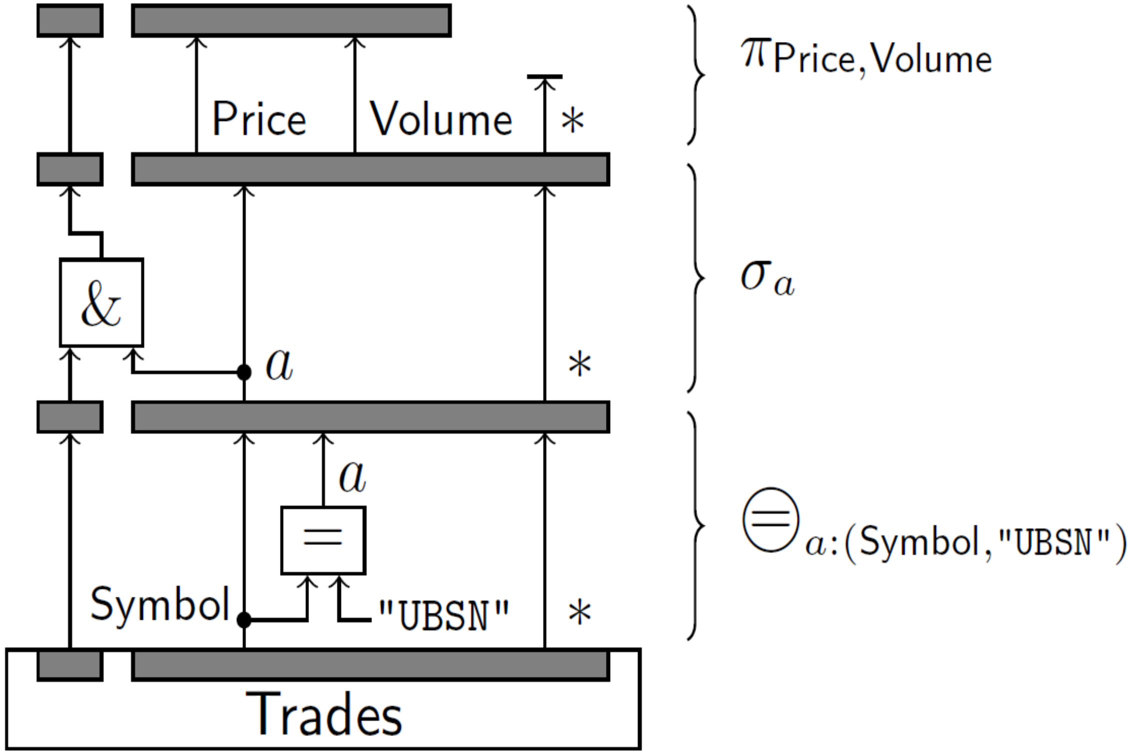
Boolean Formula Precomputation (Ibex)

Programming Models (Compilation)

Static Compiler: SQL-to-FPGA Mapping







constructing complex queries from basic composable (but static) logic blocks

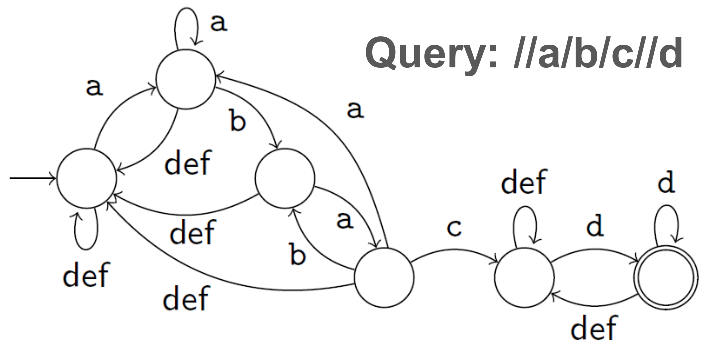
Mueller, Teubner, Alonso Streams on wires: a query compiler for FPGAs. VLDB'09
Mueller, Teubner, Alonso. Glacier: a query-to-hardware compiler. SIGMOD '10

Figure Credits: Mueller, Teubner, Alonso

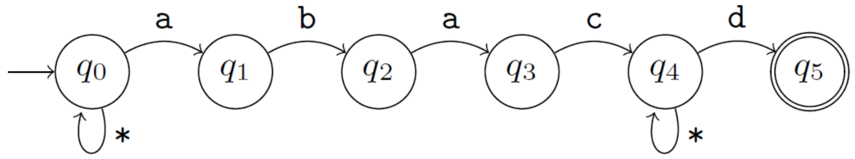
Representational Models (Parameterization)

Parametrized Circuits: Skeleton Automata

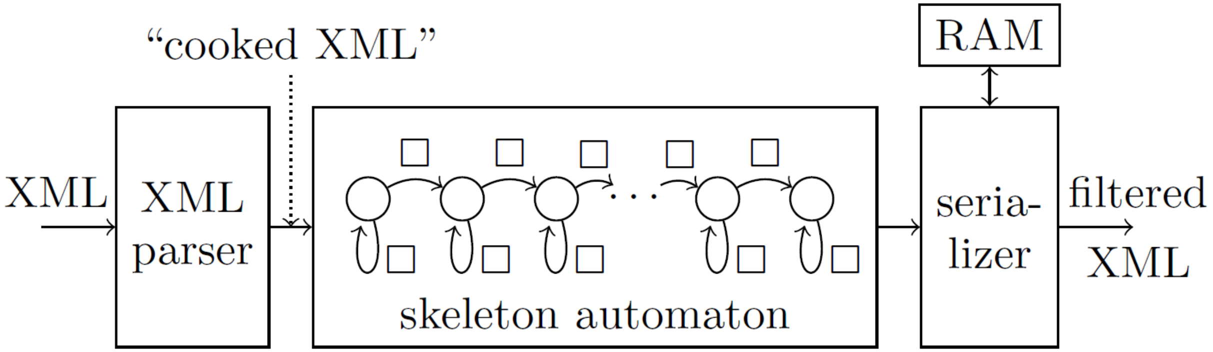
Query: //a/b/c//d

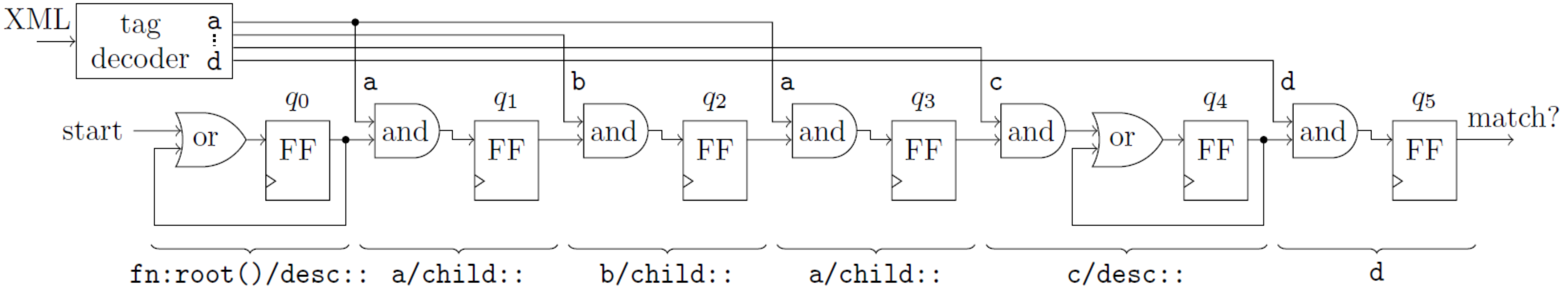


(a) Deterministic finite-state automaton (DFA).



(b) Non-deterministic finite-state automaton (NFA).



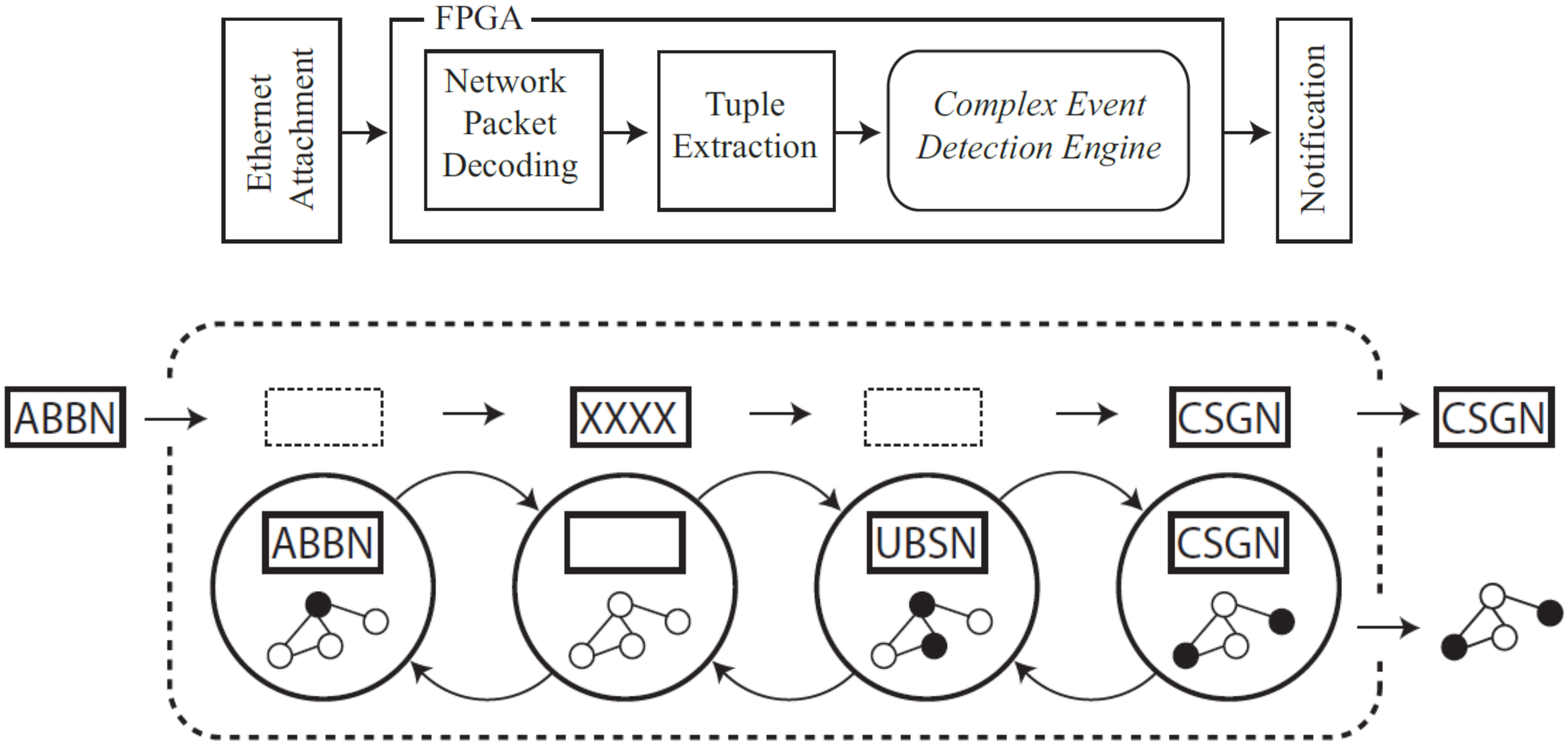


decomposing non-deterministic finite-state automata into (i) structural skeleton (logic) and (ii) reconfigurable conditions (memory)

Teubner, Woods, Nie. Skeleton automata for FPGAs: reconfiguring without reconstructing. SIGMOD '12

Figure Credits: Teubner, Woods, Nie

Parametrized Circuits: Flexible, Pipelined Partitioning

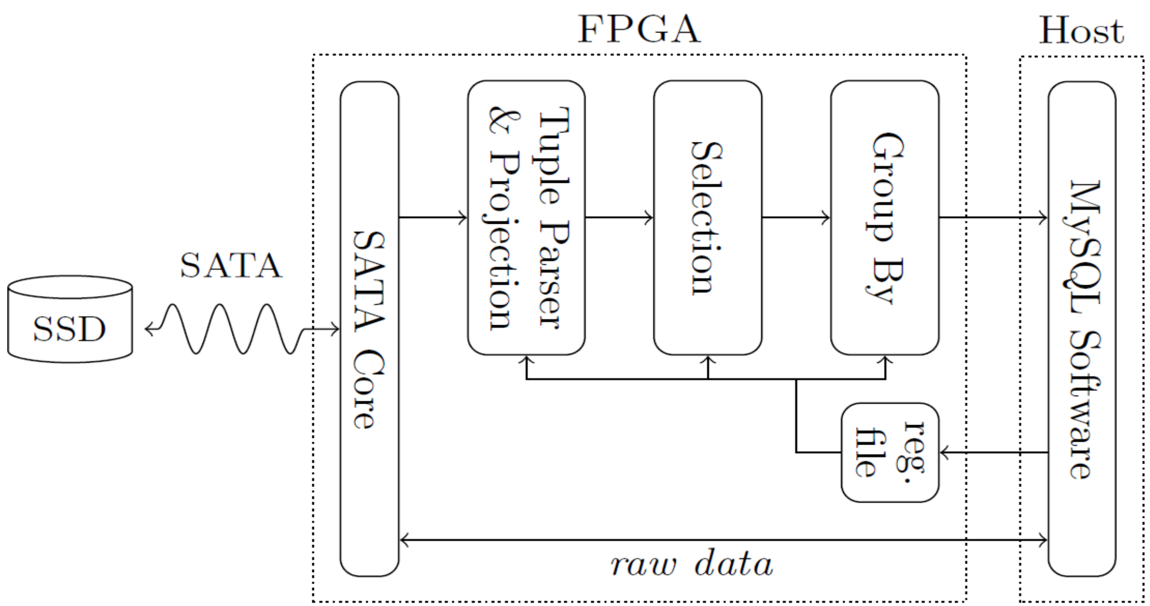


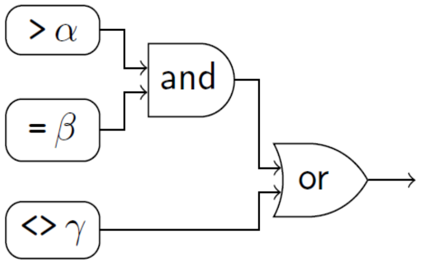
generalized pipeline design to support arbitrary-size partitioning using only neighbor-to-neighbor communication (states swap)

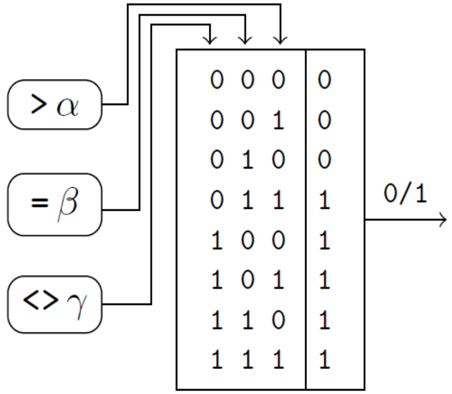
Woods, Teubner, Alonso. Complex event detection at wire speed with FPGAs.. VLDB'10

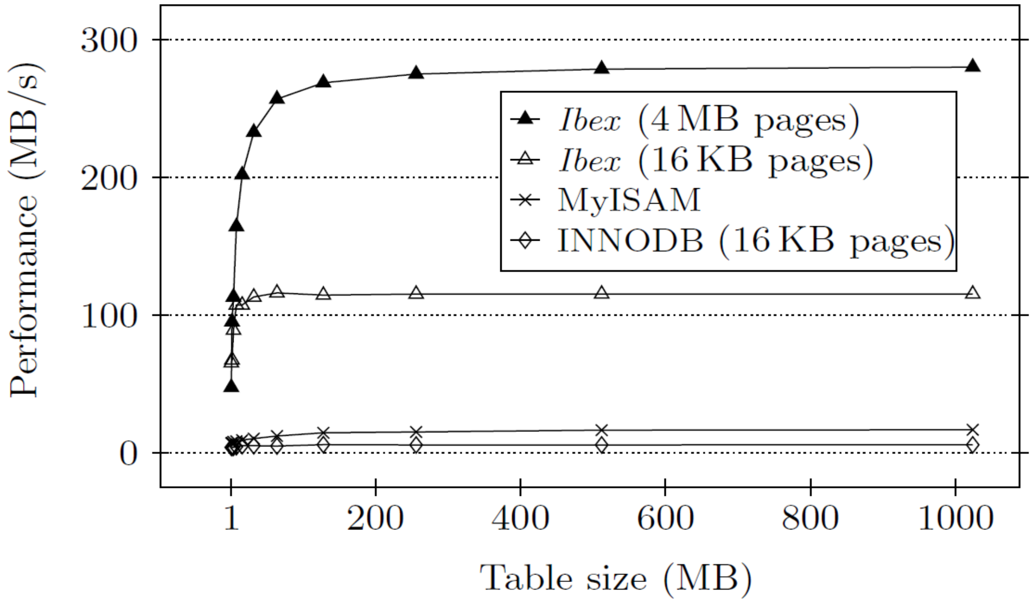
Figure Credits: Woods, Woods, Teubner, Alonso

Parametrized Circuits: Off-loading Partial Computation







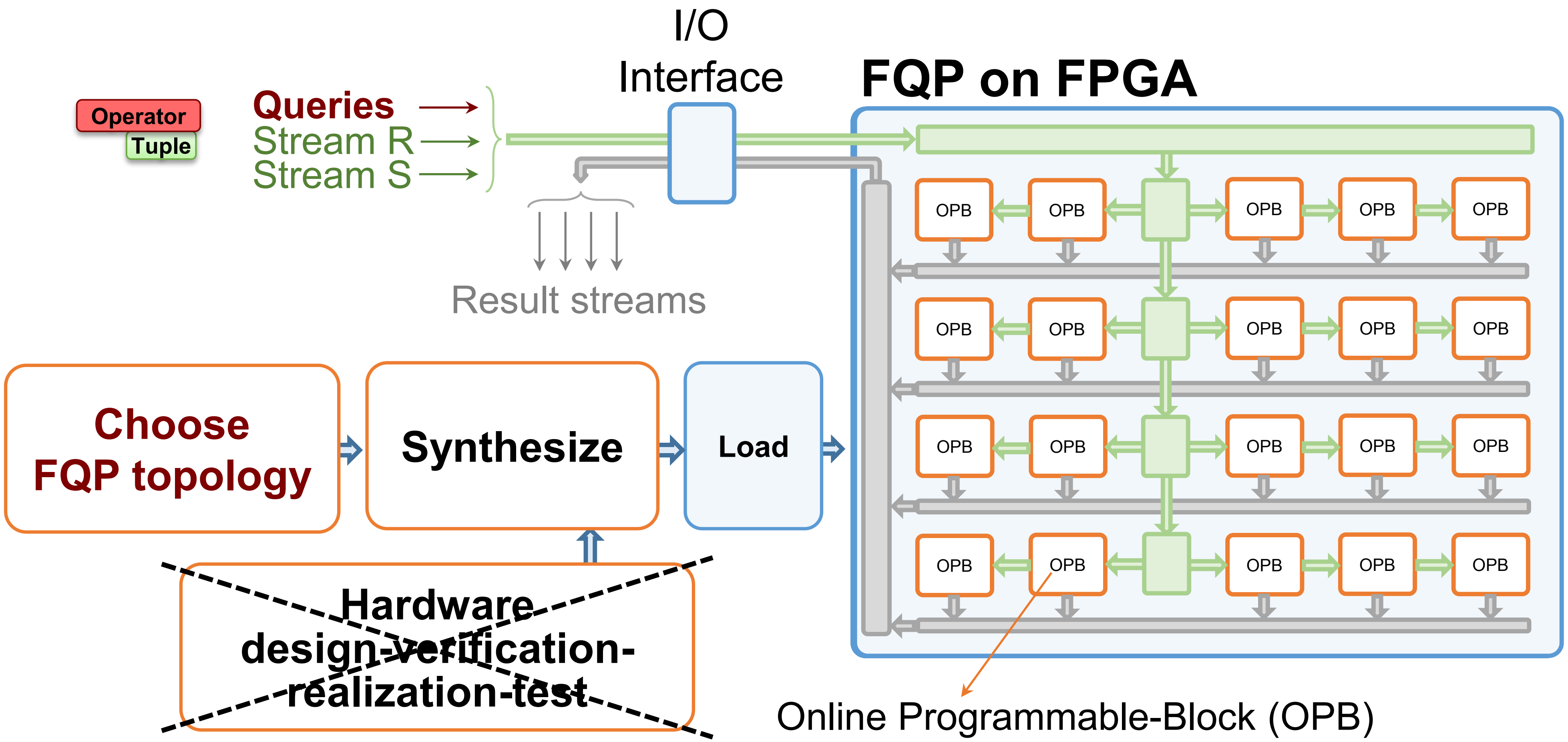


co-operative hardware (best-effort computation) and software (pre-computing arbitrary Boolean selection conditions) model

Woods, Teubner, Alonso, Less watts, more performance: An intelligent storage engine for data appliances, SIGMOD’13
Woods, Istvan, Alonso, Ibex - an intelligent storage engine with support for advanced SQL off-loading, PVLDB’14

Figure Credits: Woods, Alonso, Teubner

Parametrized Circuits & Topology: Flexible Query Processor (FQP)

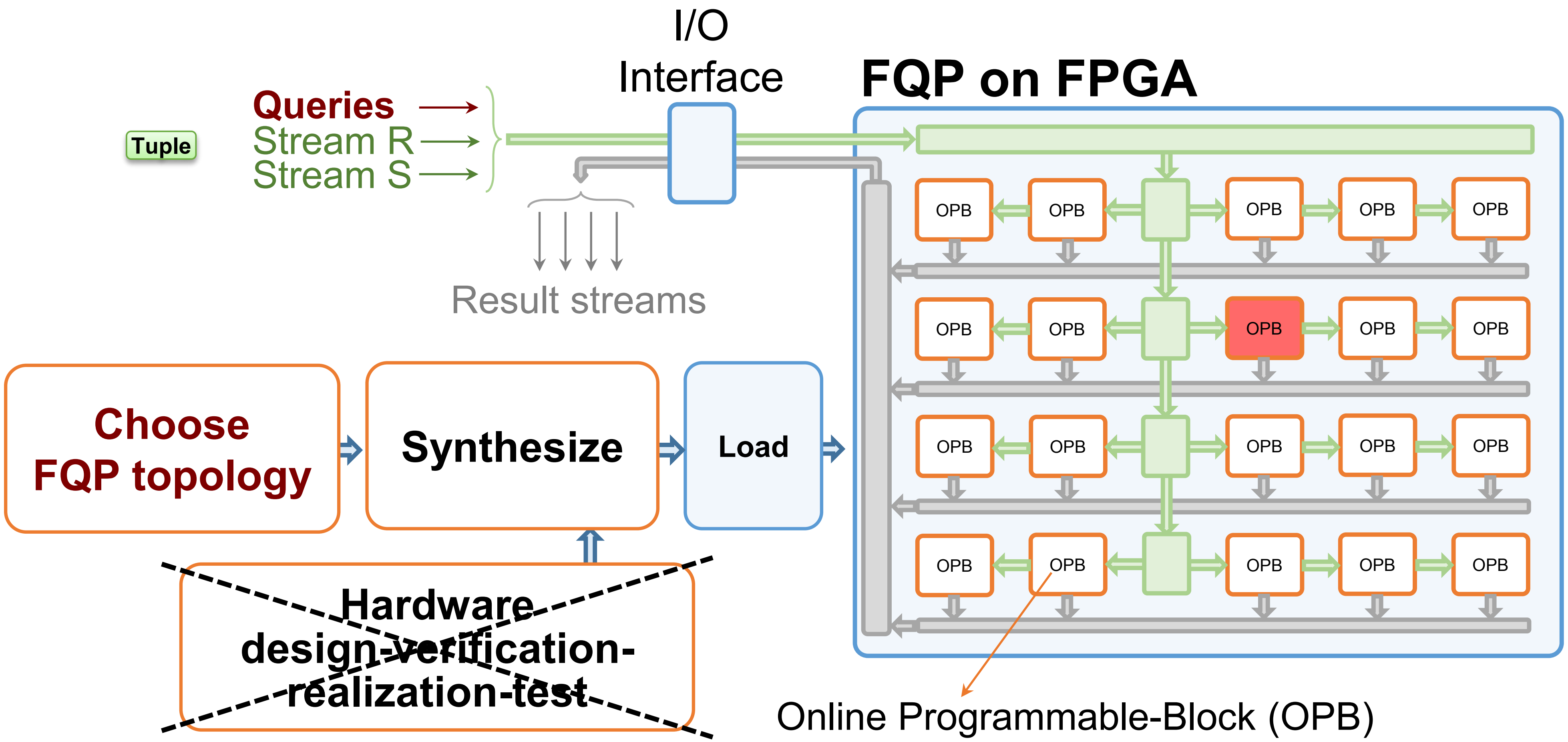


constructing complex queries from composable reconfigurable logic blocks, where composition itself is reconfigurable

Najafi, Sadoghi, Jacobsen. Flexible query processor on FPGAs. PVLDB'13
Najafi, Sadoghi, Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks,. ICDE'15

Figure Credits: Najafi, Sadoghi, Jacobsen

Parametrized Circuits & Topology: Flexible Query Processor (FQP)



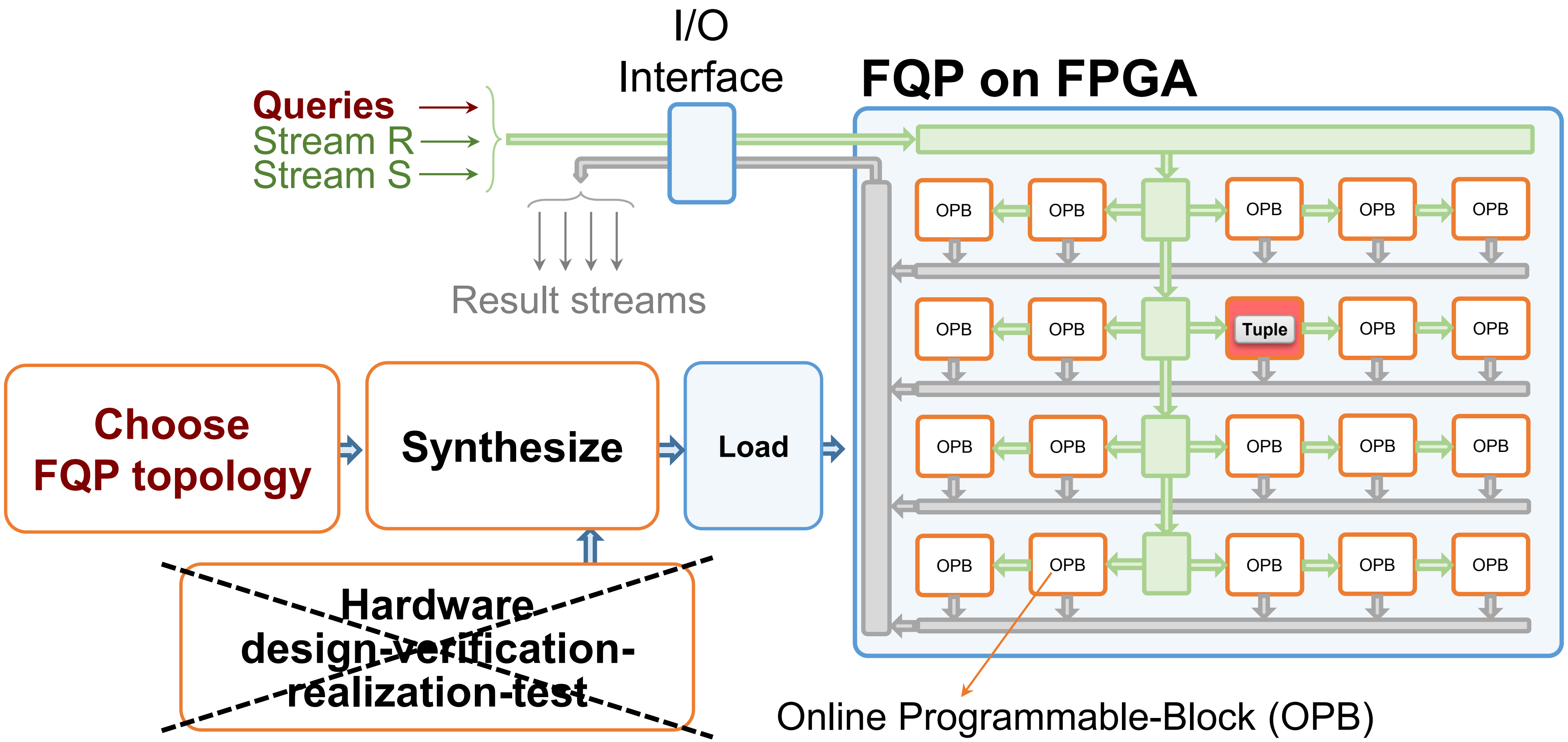
constructing complex queries from composable reconfigurable logic blocks, where composition itself is reconfigurable

Najafi, Sadoghi, Jacobsen. Flexible query processor on FPGAs. PVLDB'13

Najafi, Sadoghi, Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks,. ICDE'15

Figure Credits: Najafi, Sadoghi, Jacobsen

Parametrized Circuits & Topology: Flexible Query Processor (FQP)

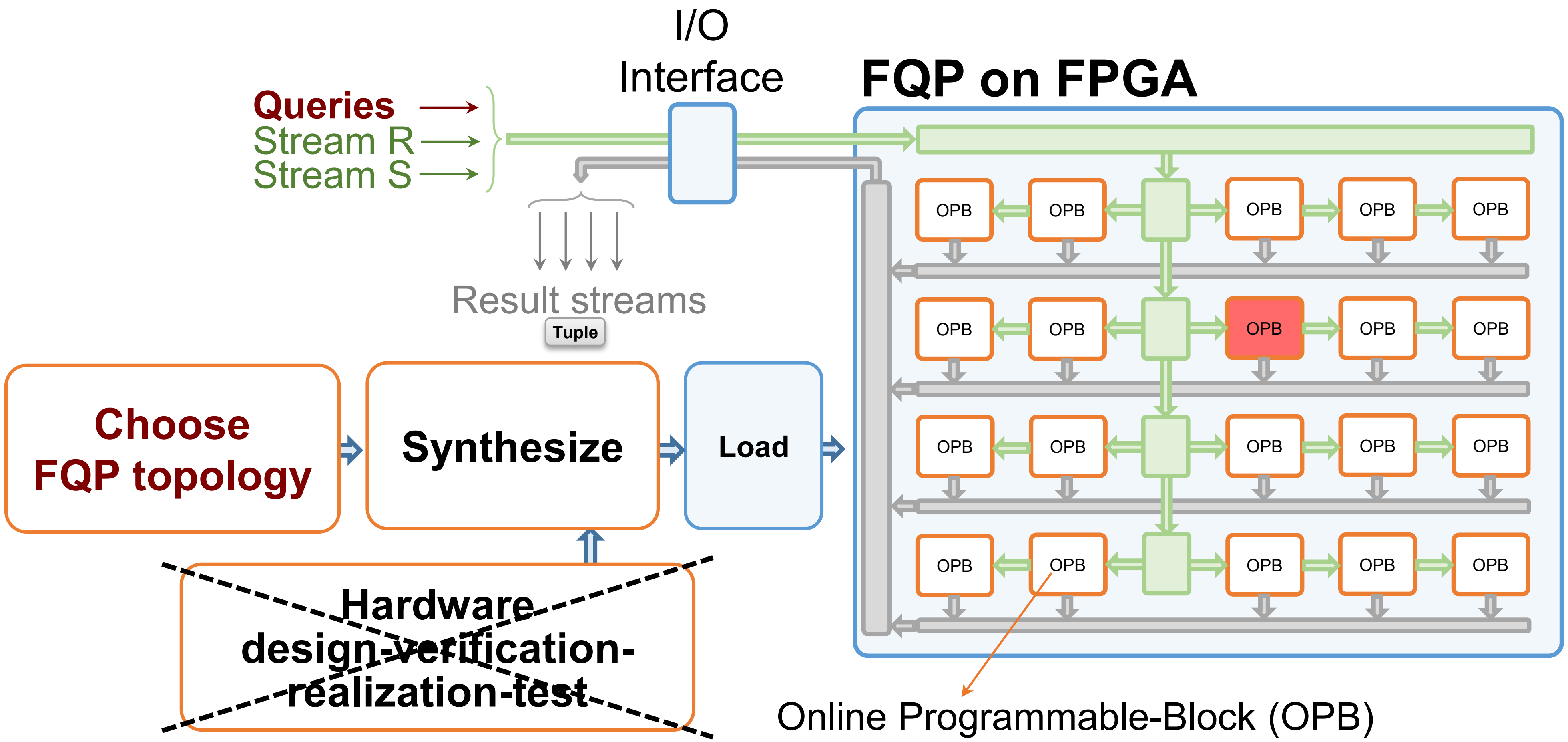


constructing complex queries from composable reconfigurable logic blocks, where composition itself is reconfigurable

Najafi, Sadoghi, Jacobsen. Flexible query processor on FPGAs. PVLDB'13
Najafi, Sadoghi, Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks,. ICDE'15

Figure Credits: Najafi, Sadoghi, Jacobsen

Parametrized Circuits & Topology: Flexible Query Processor (FQP)

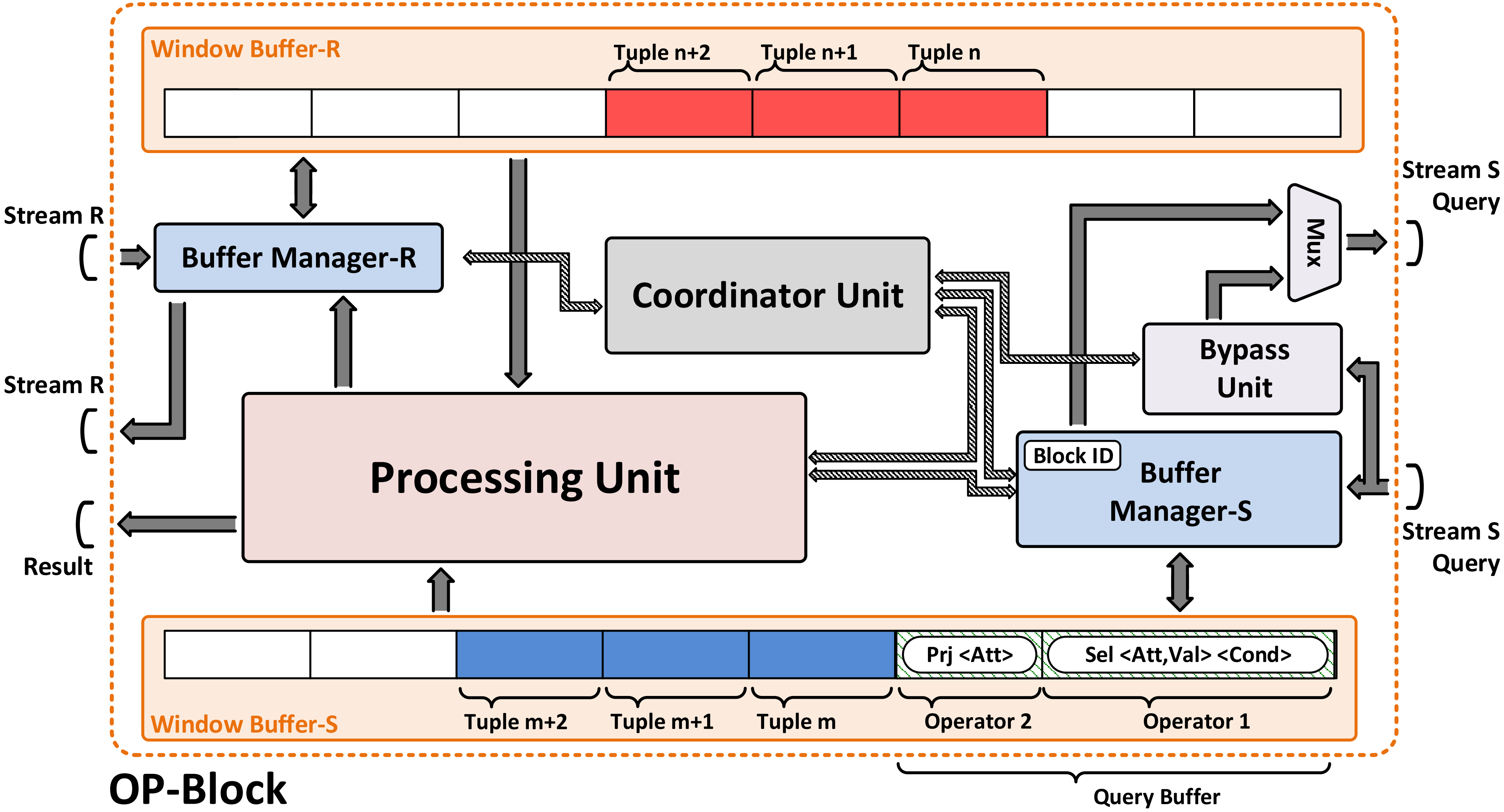


constructing complex queries from composable reconfigurable logic blocks, where composition itself is reconfigurable

Najafi, Sadoghi, Jacobsen. Flexible query processor on FPGAs. PVLDB'13
Najafi, Sadoghi, Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks,. ICDE'15

Figure Credits: Najafi, Sadoghi, Jacobsen

Parametrized Circuits: Online Programmable-blocks (OPB) Internals



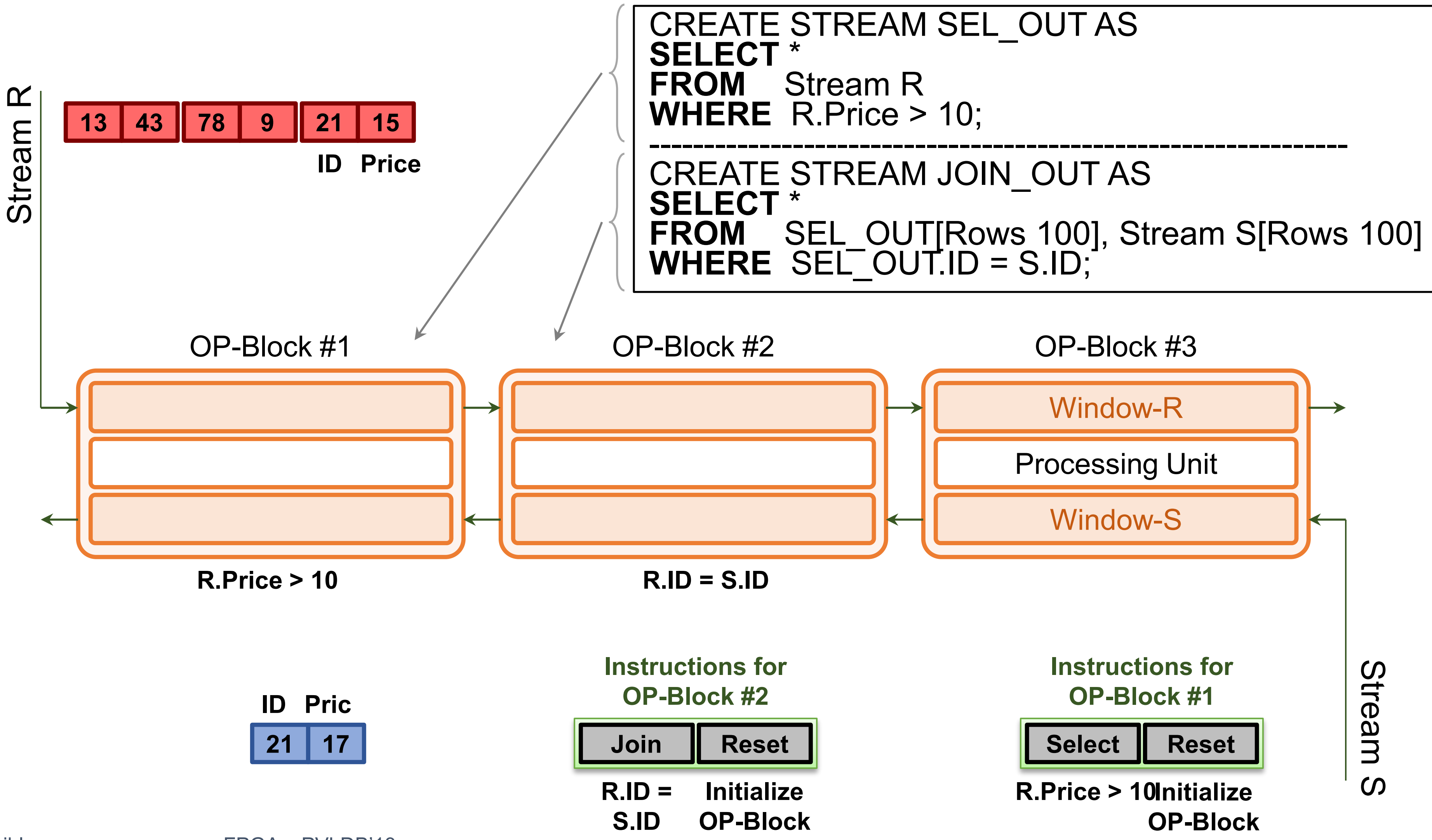
a general reconfigurable logic blocks supporting selection, project, and join operations

Najafi, Sadoghi, Jacobsen. Flexible query processor on FPGAs. PVLDB'13

Najafi, Sadoghi, Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks,. ICDE'15

Figure Credits: Najafi, Sadoghi, Jacobsen

FQP: Query Programming

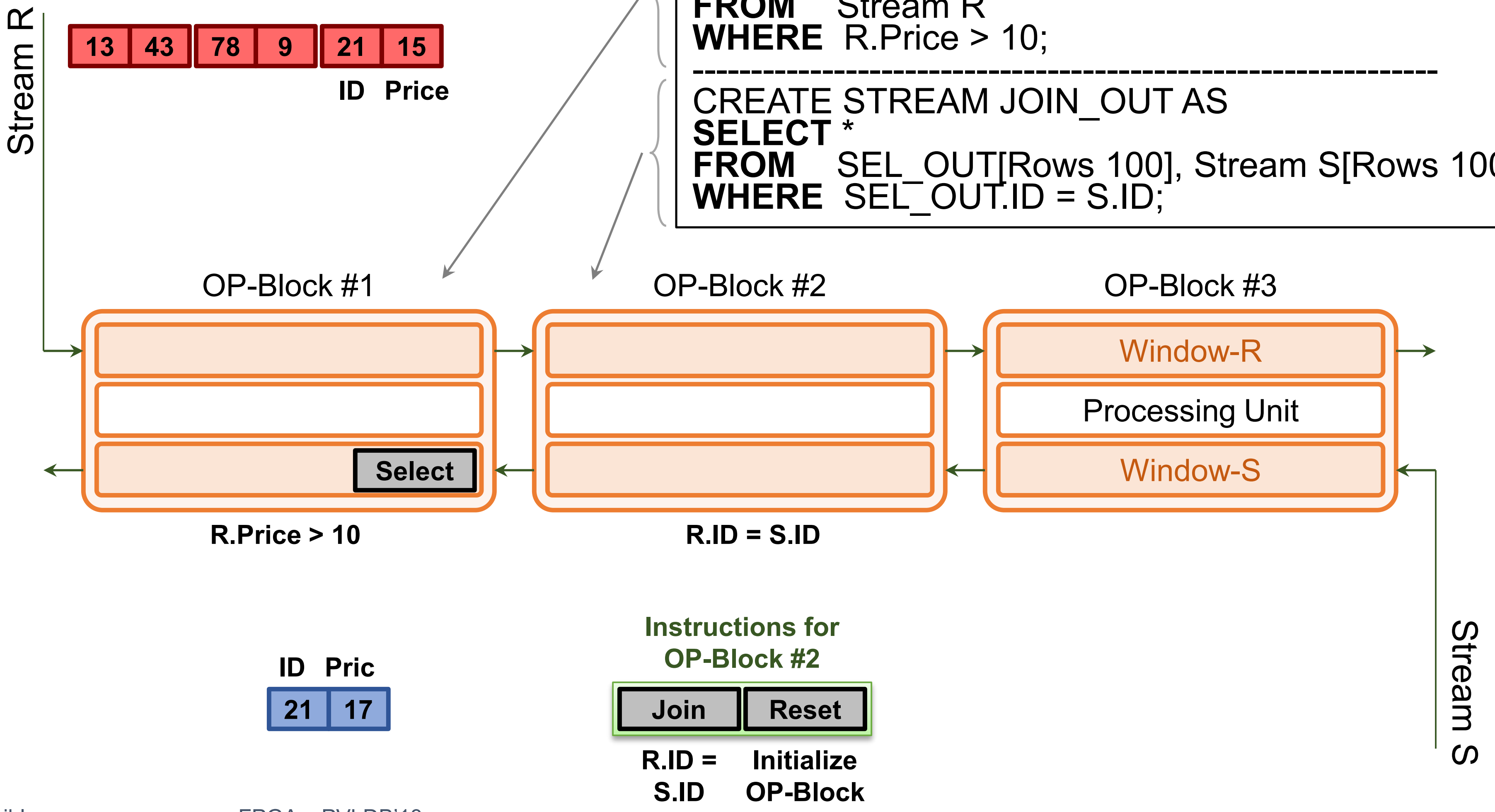


Najafi, Sadoghi, Jacobsen. Flexible query processor on FPGAs. PVLDB'13
Najafi, Sadoghi, Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks,. ICDE'15

Figure Credits: Najafi, Sadoghi, Jacobsen

FQP: Query Programming

```
CREATE STREAM SEL_OUT AS
SELECT *
FROM   Stream R
WHERE  R.Price > 10;
----------------------------------------------------------------
CREATE STREAM JOIN_OUT AS
SELECT *
FROM   SEL_OUT[Rows 100], Stream S[Rows 100]
WHERE  SEL_OUT.ID = S.ID;
```

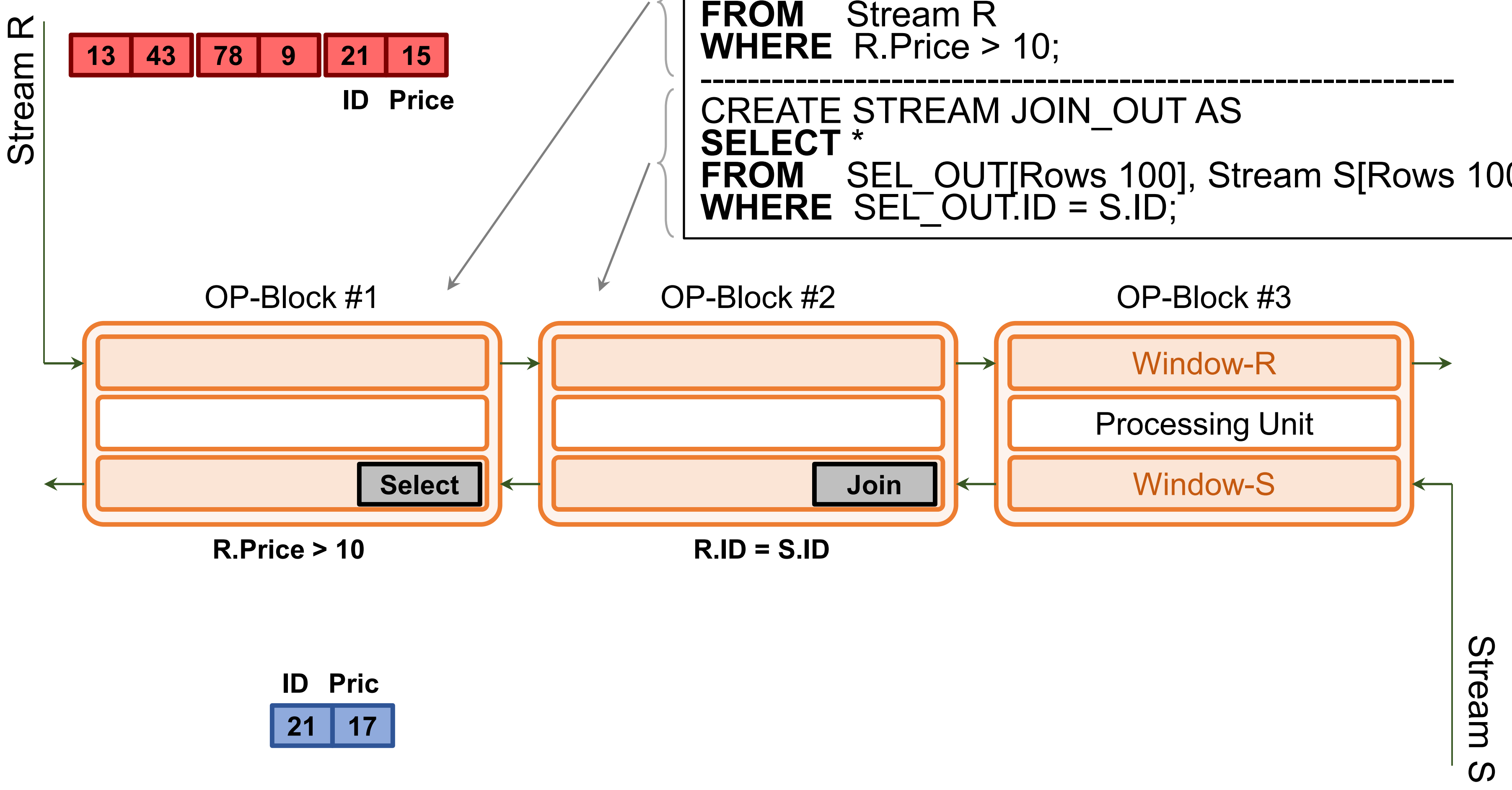


Najafi, Sadoghi, Jacobsen. Flexible query processor on FPGAs. PVLDB'13
Najafi, Sadoghi, Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks,. ICDE'15

Figure Credits: Najafi, Sadoghi, Jacobsen

FQP: Query Programming

```
CREATE STREAM SEL_OUT AS
SELECT *
FROM   Stream R
WHERE  R.Price > 10;
----------------------------------------------------------------
CREATE STREAM JOIN_OUT AS
SELECT *
FROM   SEL_OUT[Rows 100], Stream S[Rows 100]
WHERE  SEL_OUT.ID = S.ID;
```

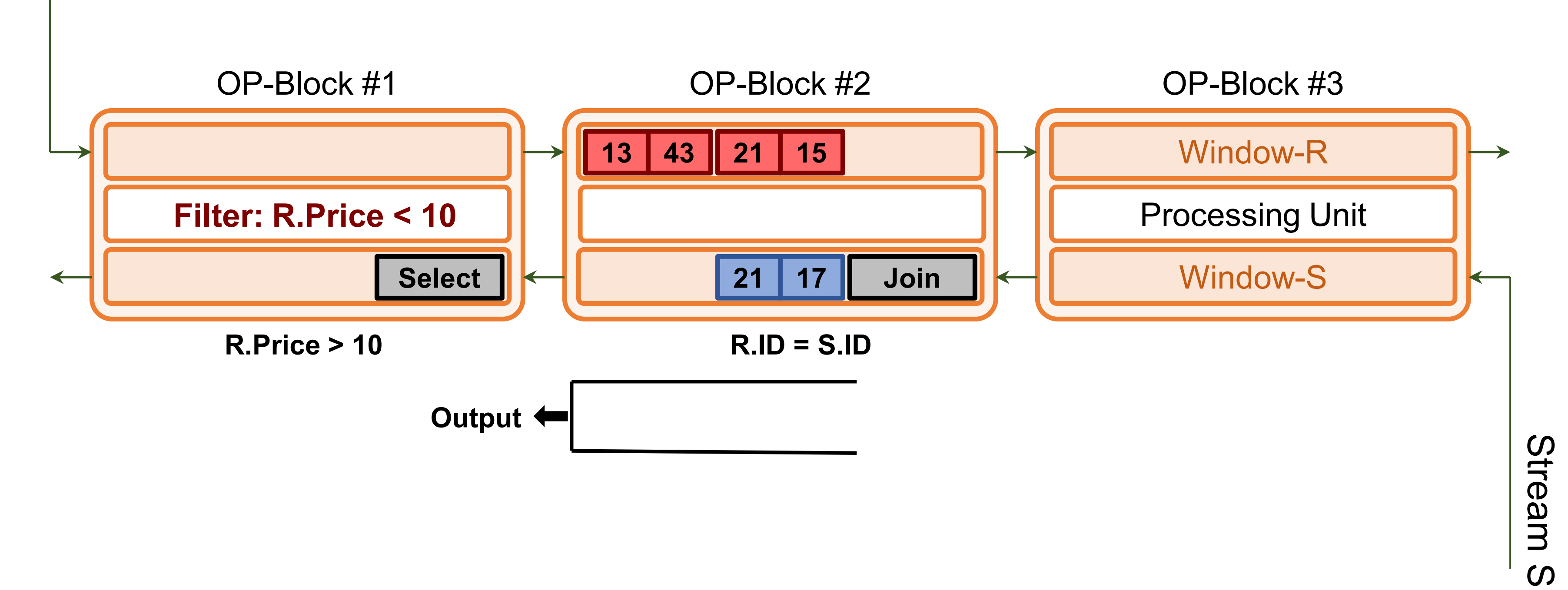


Najafi, Sadoghi, Jacobsen. Flexible query processor on FPGAs. PVLDB’13
Najafi, Sadoghi, Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks,. ICDE’15

Figure Credits: Najafi, Sadoghi, Jacobsen

FQP: Query Execution

Operation	OPB Instruction Programming Latency	
Projection	5 cycles	40ns
Join	5 cycles	40ns
Reset	4 cycles	32ns
Bypass	2+2 cycles	32ns

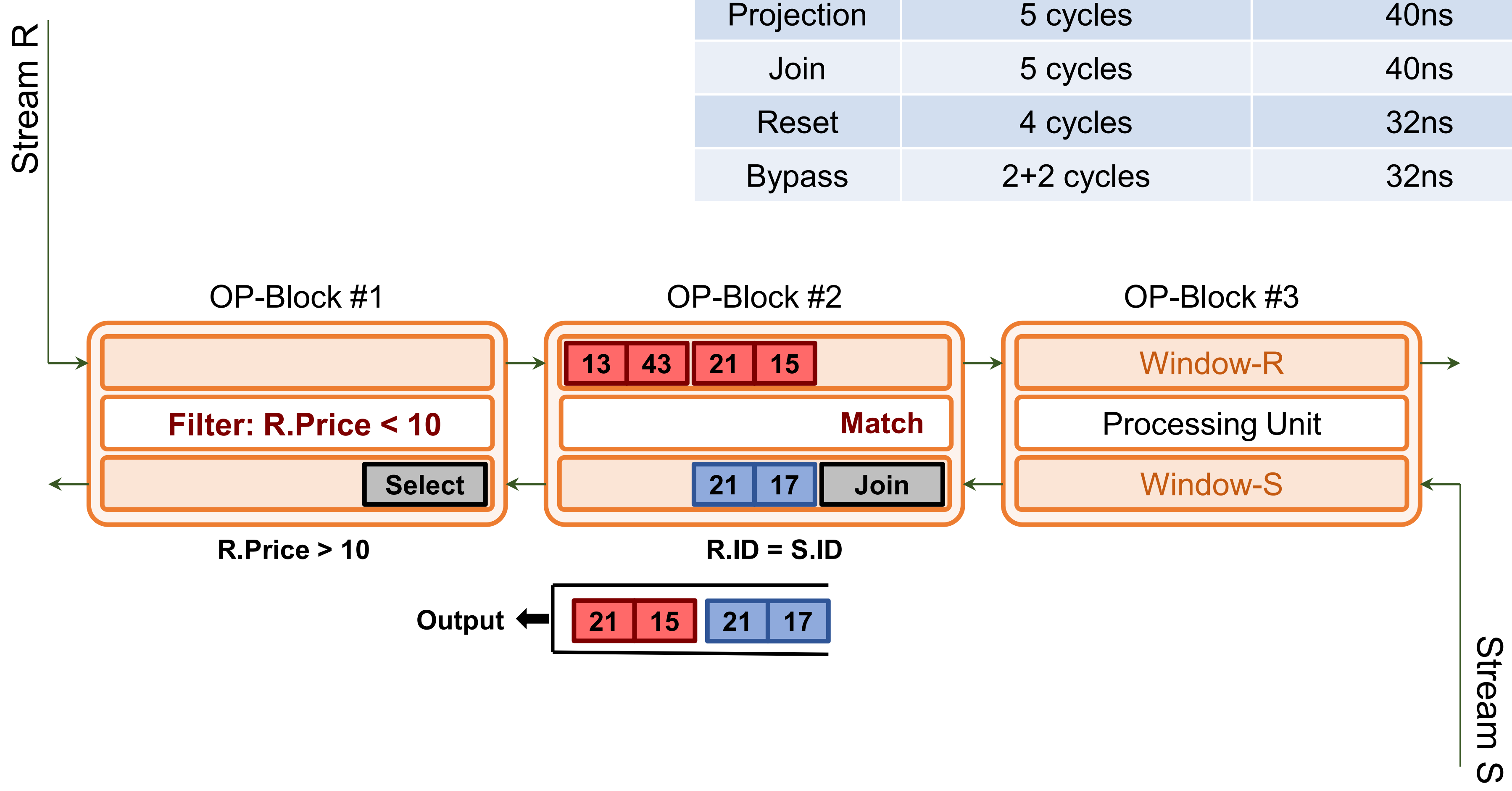


Najafi, Sadoghi, Jacobsen. Flexible query processor on FPGAs. PVLDB'13
Najafi, Sadoghi, Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks,. ICDE'15

Figure Credits: Najafi, Sadoghi, Jacobsen

FQP: Query Execution

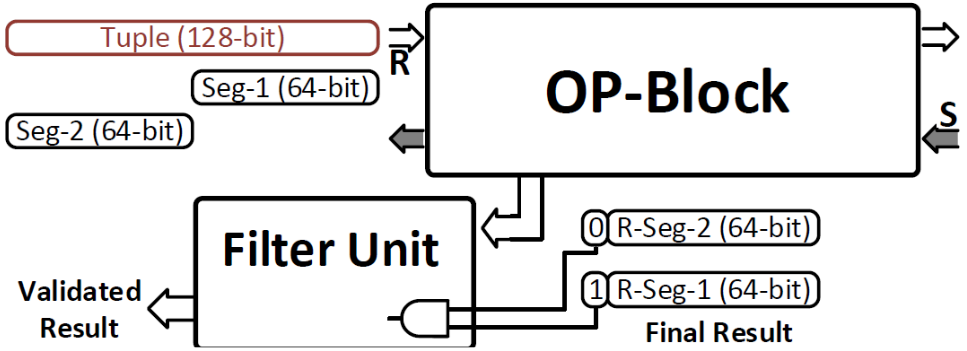
Operation	OPB Instruction Programming Latency	
Projection	5 cycles	40ns
Join	5 cycles	40ns
Reset	4 cycles	32ns
Bypass	2+2 cycles	32ns



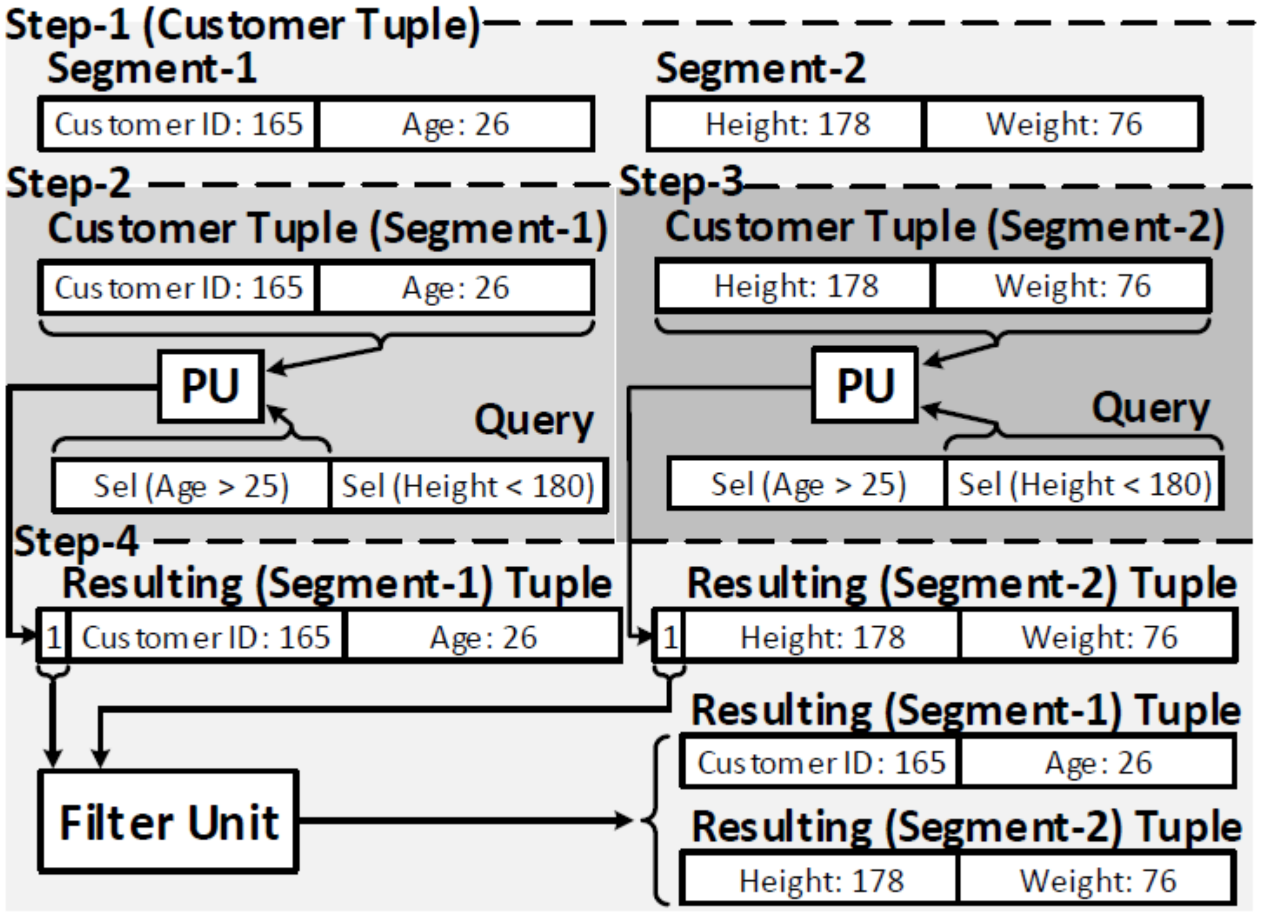
Najafi, Sadoghi, Jacobsen. Flexible query processor on FPGAs. PVLDB'13
Najafi, Sadoghi, Jacobsen. Configurable hardware-based streaming architecture using online programmable-blocks,. ICDE'15

Figure Credits: Najafi, Sadoghi, Jacobsen

Parametrized Segments: Vertical Query & Data Partitioning




```
CREATE STREAM CS_SEL AS
SELECT *
  FROM Customer_Stream
 WHERE Age > 25, Height < 180
```

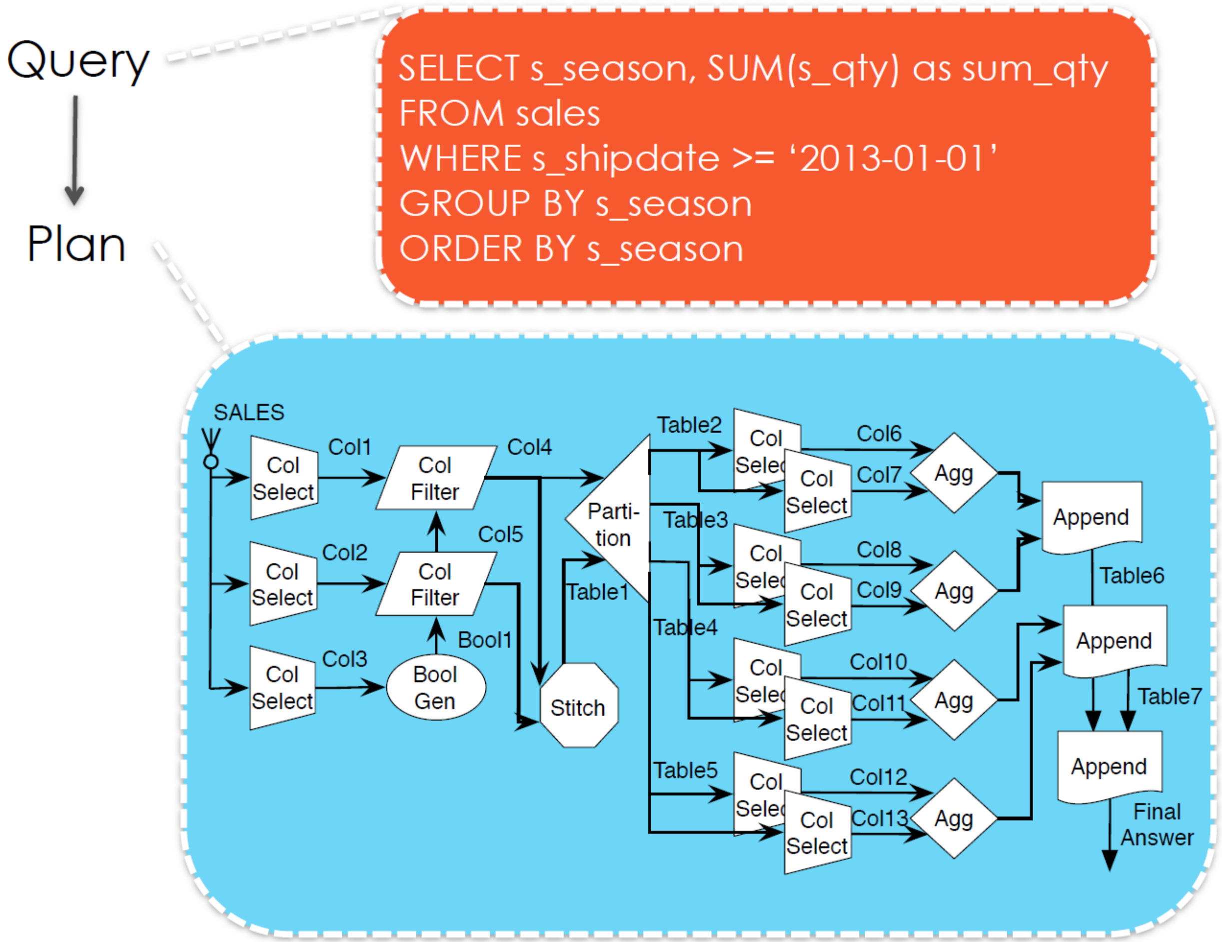


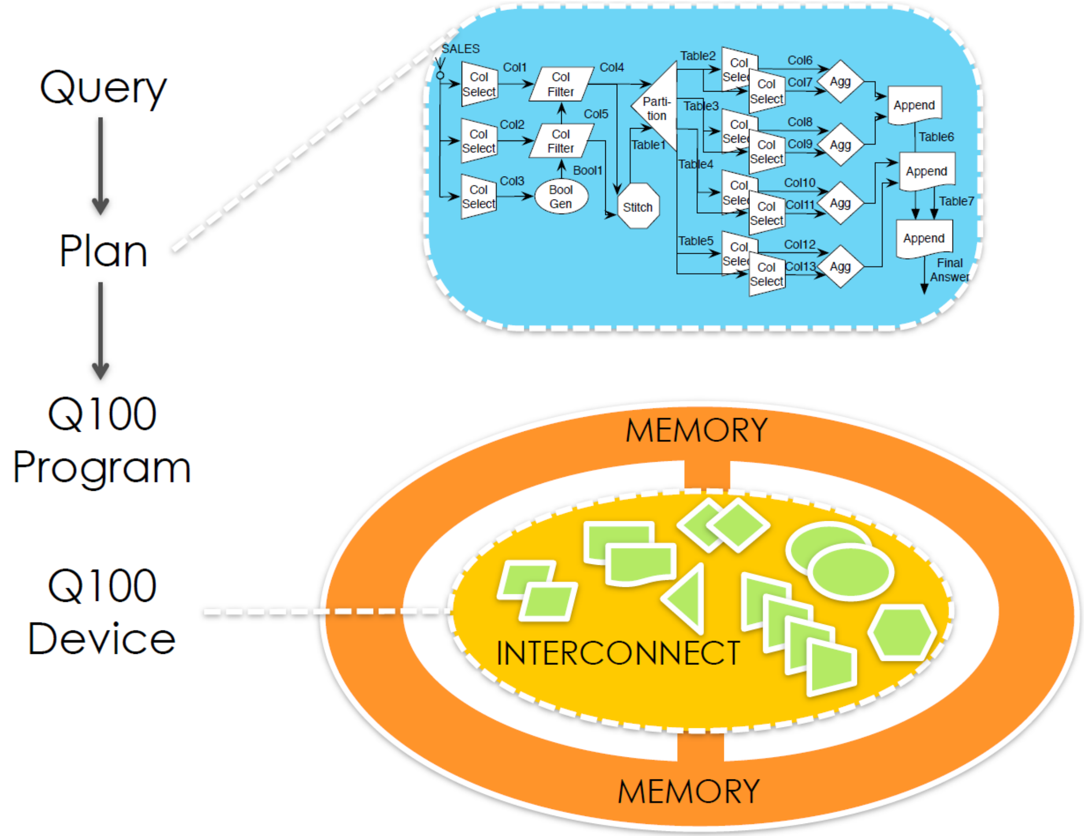
supporting arbitrary-size schema given a fixed wiring/routing-budget through a vertically partitioning of query and data

Najafi, Sadoghi, Jacobsen. The FQP vision: Flexible query processing on a reconfigurable computing fabric, SIGMOD Record'15

Figure Credits: Najafi, Sadoghi, Jacobsen

Specialized Processing Unit: Horizontal Query Partitioning



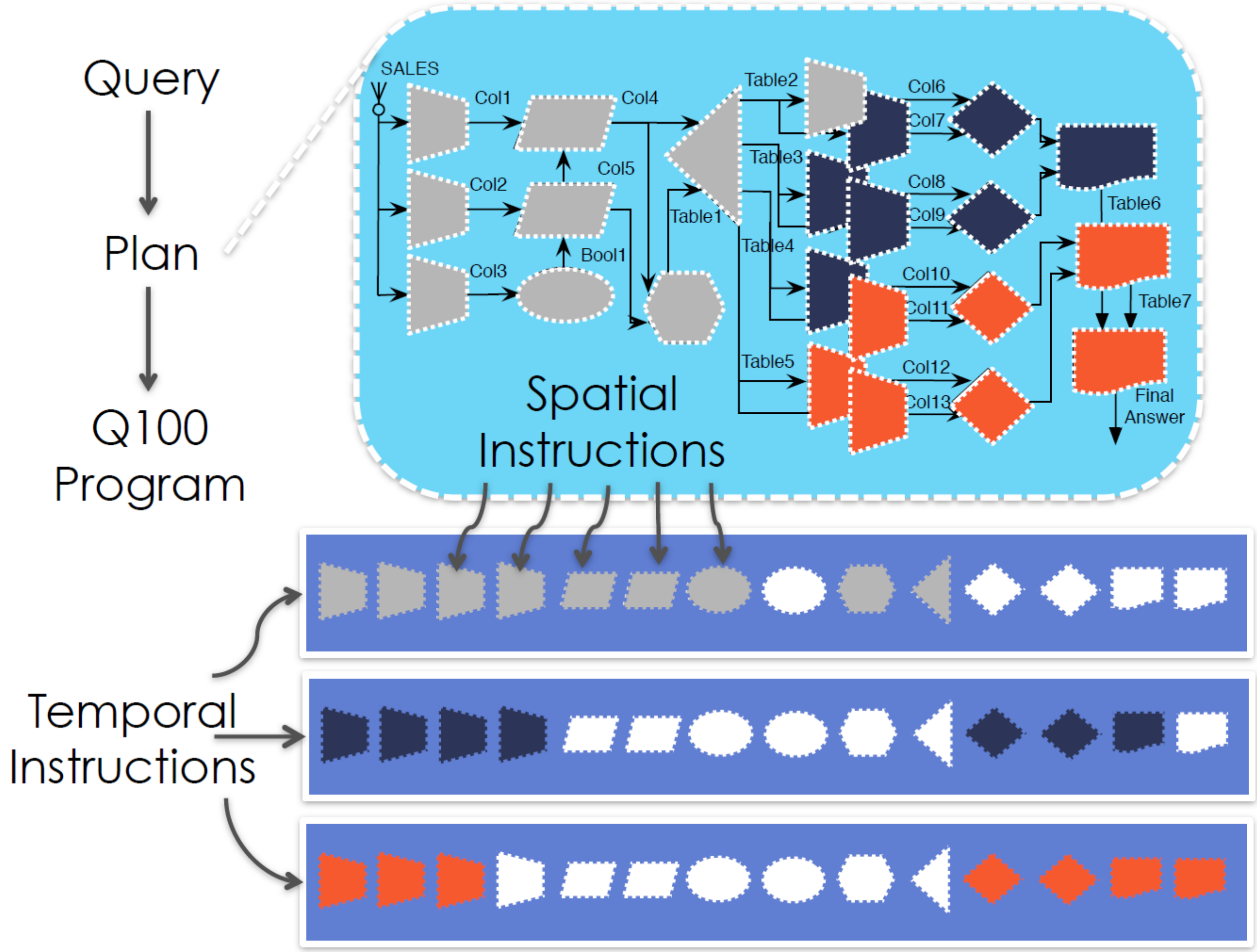


supporting arbitrary-size query given a fixed logic-budget through horizontally partitioning of query into pipelined stages

Wu, Andrea, Timothy, Kim, Ross. Q100: The Architecture and Design of a Database Processing Unit. ASPLOS'14
Wu, Lottarini, Paine, Kim, Ross. The Q100 Database Processing Unit. IEEE Micro'15

Figure Credits: Wu, Lottarini, Paine, Kim, Ross

Specialized Processing Unit: Horizontal Query Partitioning

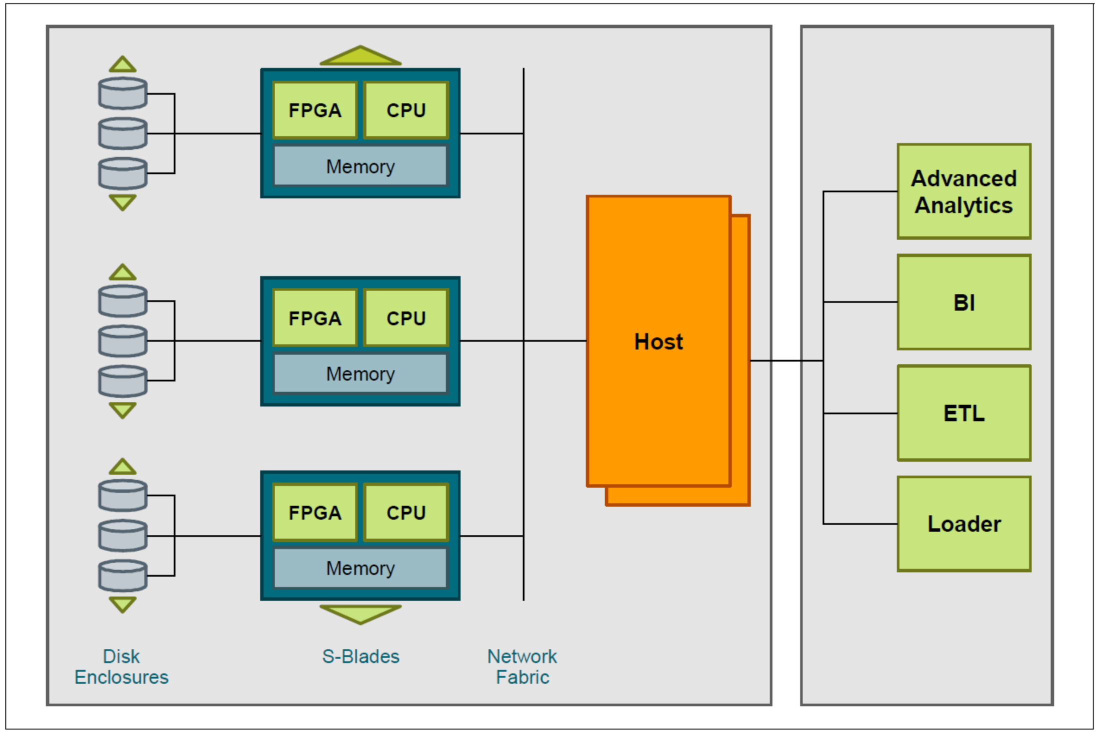


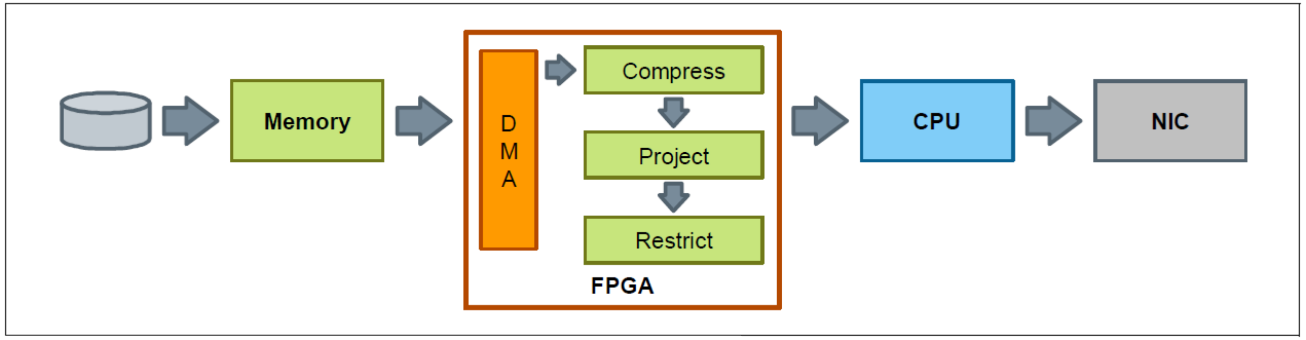
supporting arbitrary-size query given a fixed logic-budget through horizontally partitioning of query into pipelined stages

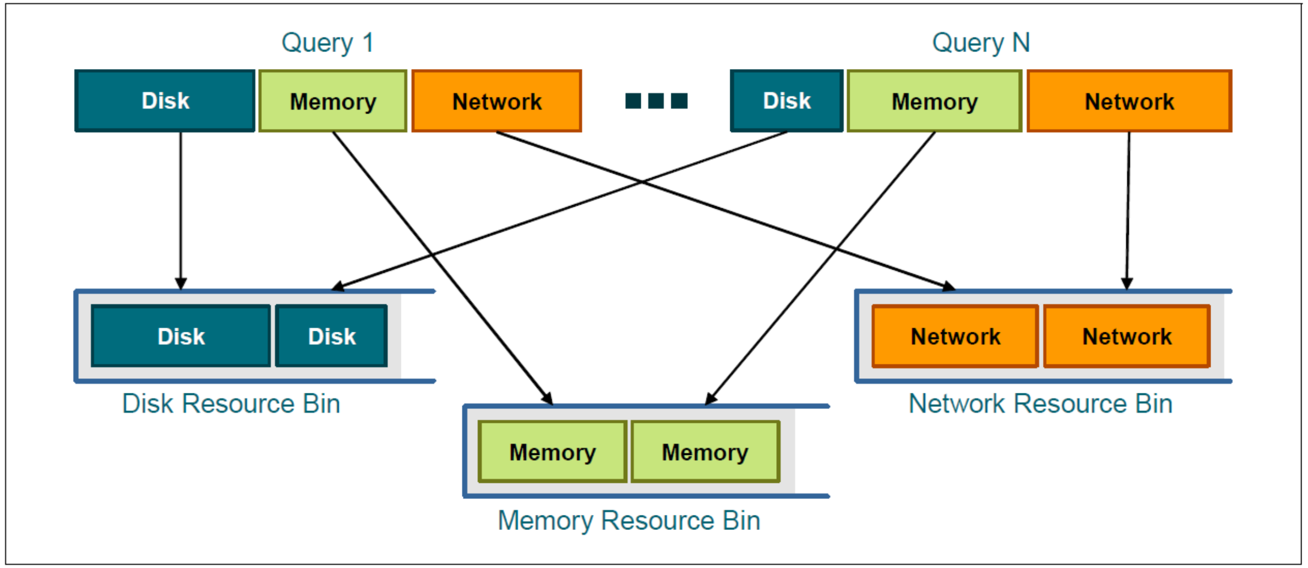
Wu, Andrea, Timothy, Kim, Ross. Q100: The Architecture and Design of a Database Processing Unit. ASPLOS'14
Wu, Lottarini, Paine, Kim, Ross. The Q100 Database Processing Unit. IEEE Micro'15

Figure Credits: Wu, Lottarini, Paine, Kim, Ross

IBM Netezza: FPGA-Accelerated Streaming Technology (FAST) Engines™





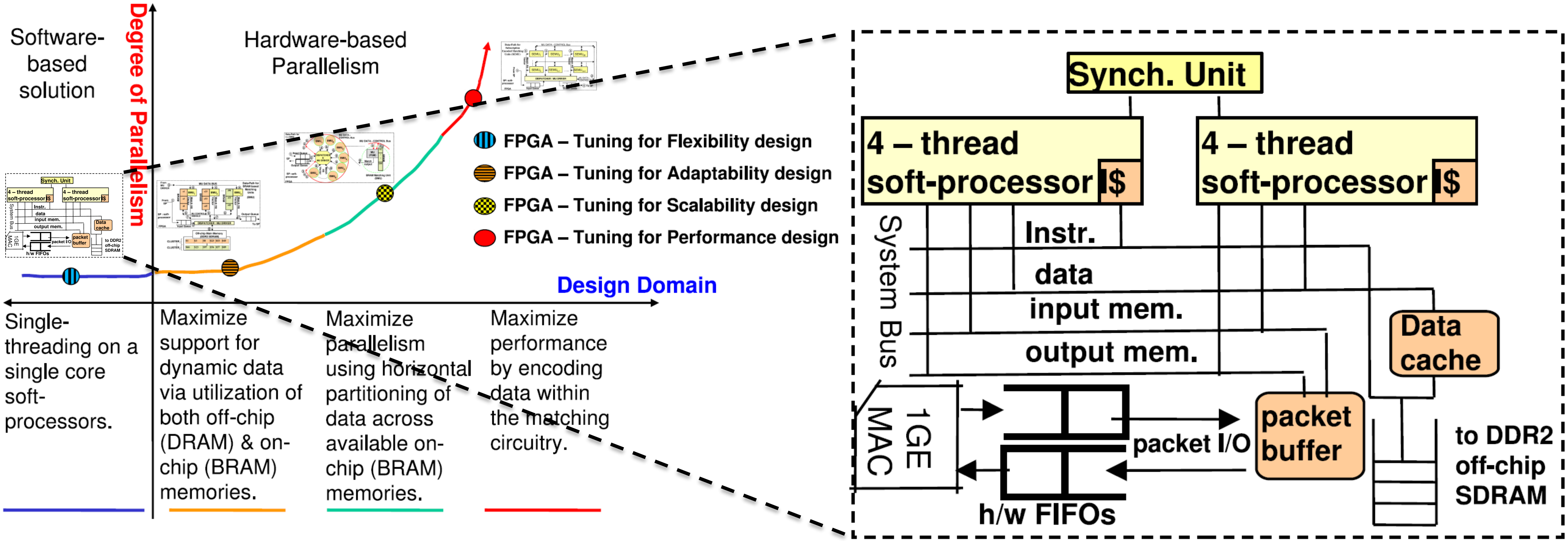


a commercial success of using parameterizable circuit design for offloading query computation within IBM Netezza appliance

Francisco: IBM PureData System for Analytics Architecture A Platform for High Performance Data Warehousing and Analytics, IBM Redbook'14
The Netezza FAST Engines™ Framework A Powerful Framework for High-Performance Analytics. Netezza White Paper'08

Algorithmic Models
(Balancing Data Flow vs. Control Flow)

fpga-ToPSS: Event Processing Design Landscape

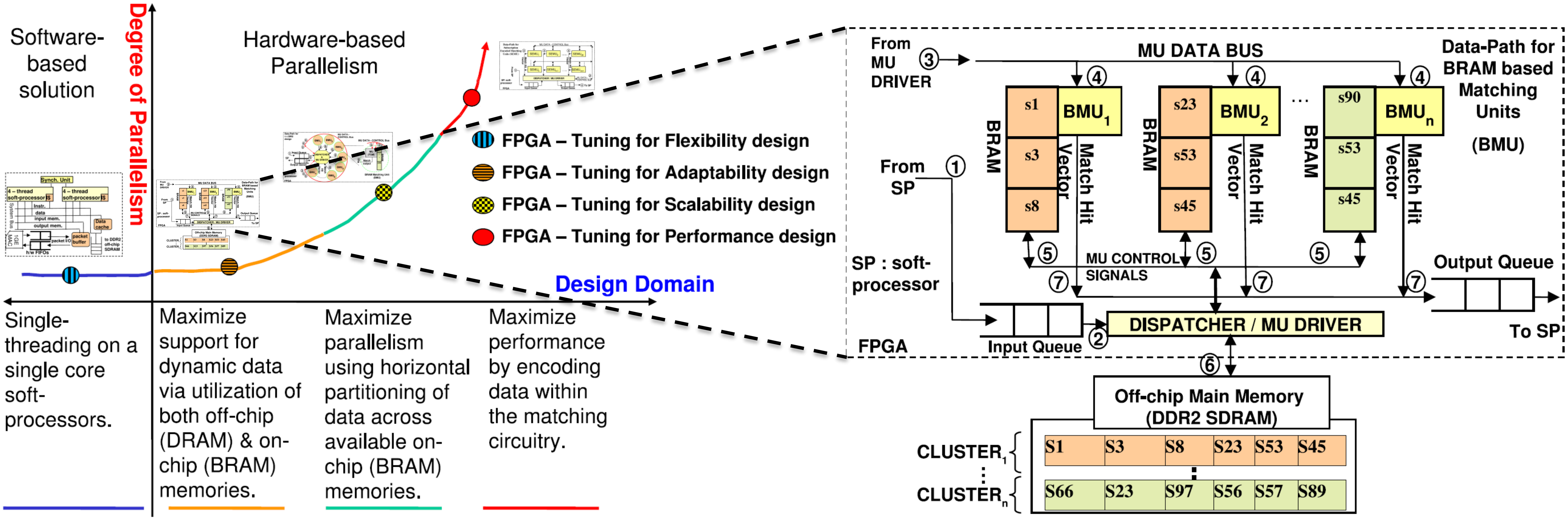


providing ease of the development and deployment cycle by running C program on soft-cores

Sadoghi, Labrecque, Singh, Shum, Jacobsen, Efficient event processing through reconfigurable hardware for algorithmic trading, PVLDB'10
Sadoghi, Singh, Jacobsen. Towards highly parallel event processing through reconfigurable hardware. DaMoN'11
Sadoghi, Singh, Jacobsen. fpga-ToPSS: line-speed event processing on FPGAs. DEBS'11

Figure Credits: Sadoghi, Labrecque, Singh, Shum, Jacobsen

fpga-ToPSS: Event Processing Design Landscape

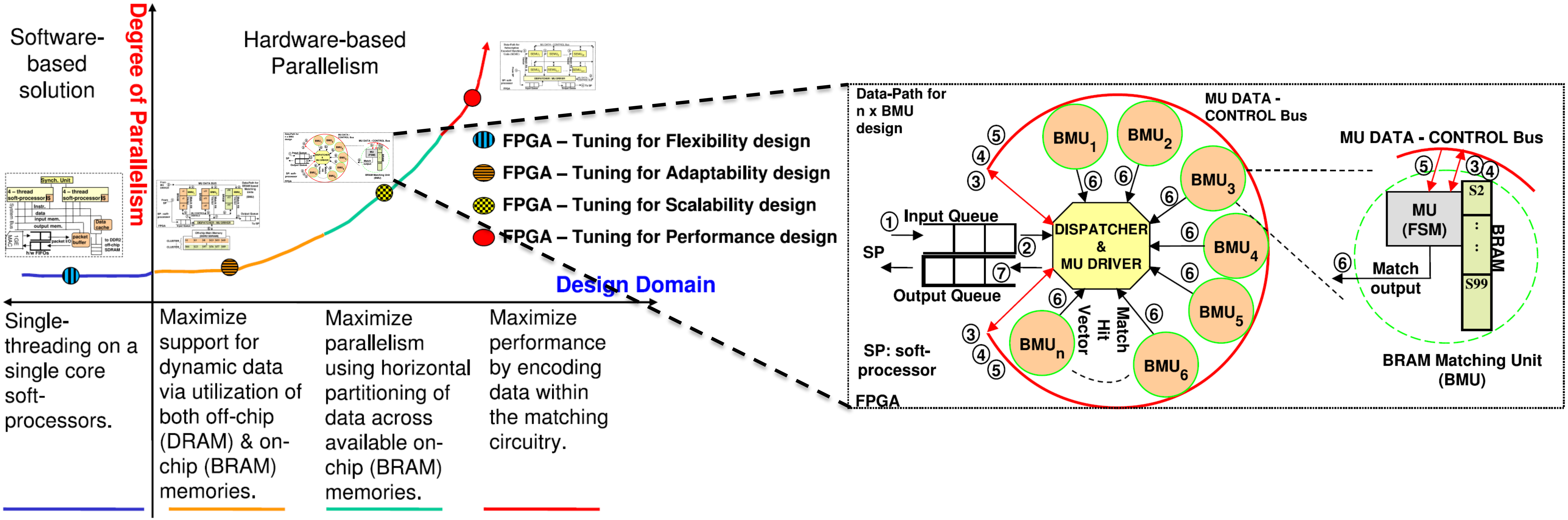


supporting changes to queries by storing it in off-chip memory while hiding latency by storing static queries in on-chip memory

Sadoghi, Labrecque, Singh, Shum, Jacobsen, Efficient event processing through reconfigurable hardware for algorithmic trading, PVLDB'10
Sadoghi, Singh, Jacobsen. Towards highly parallel event processing through reconfigurable hardware. DaMoN'11
Sadoghi, Singh, Jacobsen. fpga-ToPSS: line-speed event processing on FPGAs. DEBS'11

Figure Credits: Sadoghi, Labrecque, Singh, Shum, Jacobsen

fpga-ToPSS: Event Processing Design Landscape

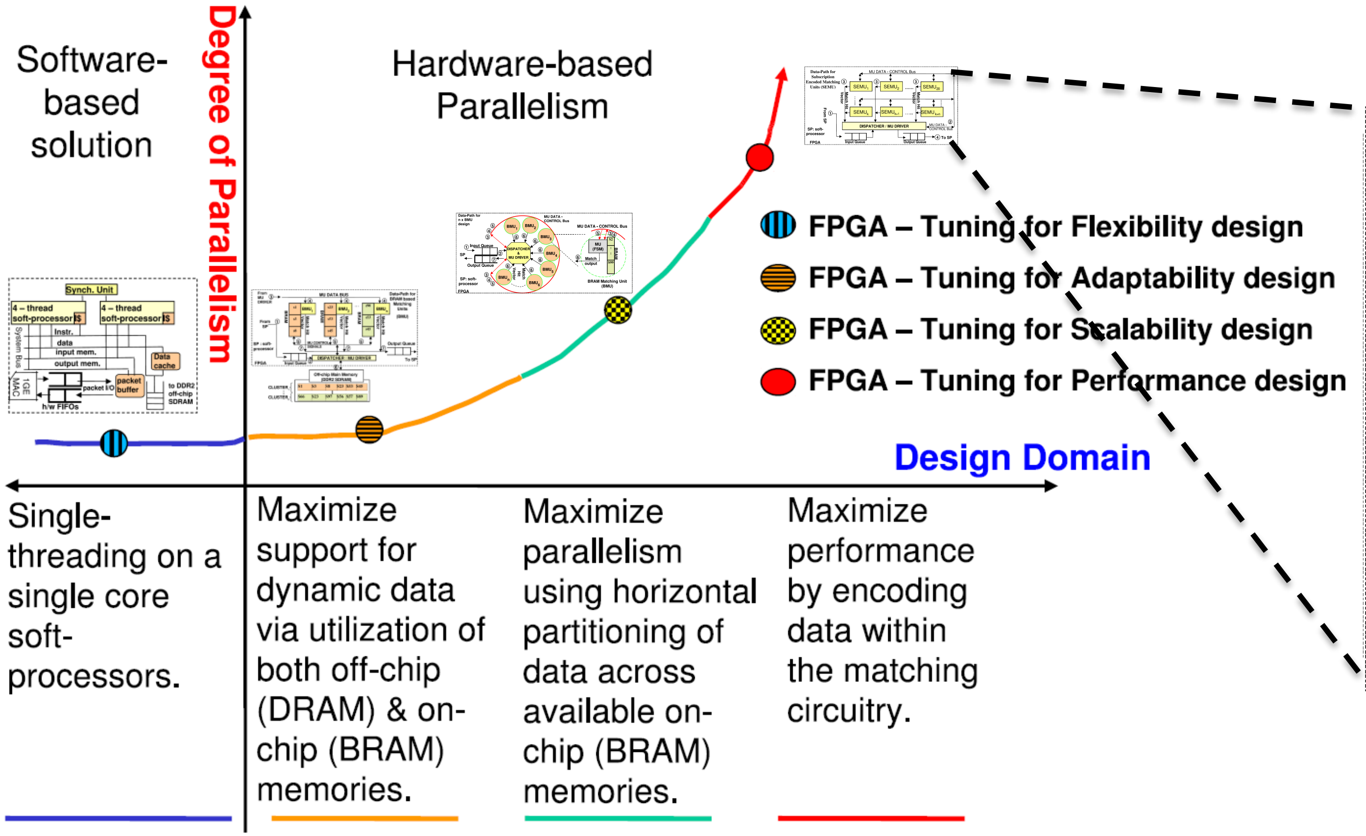


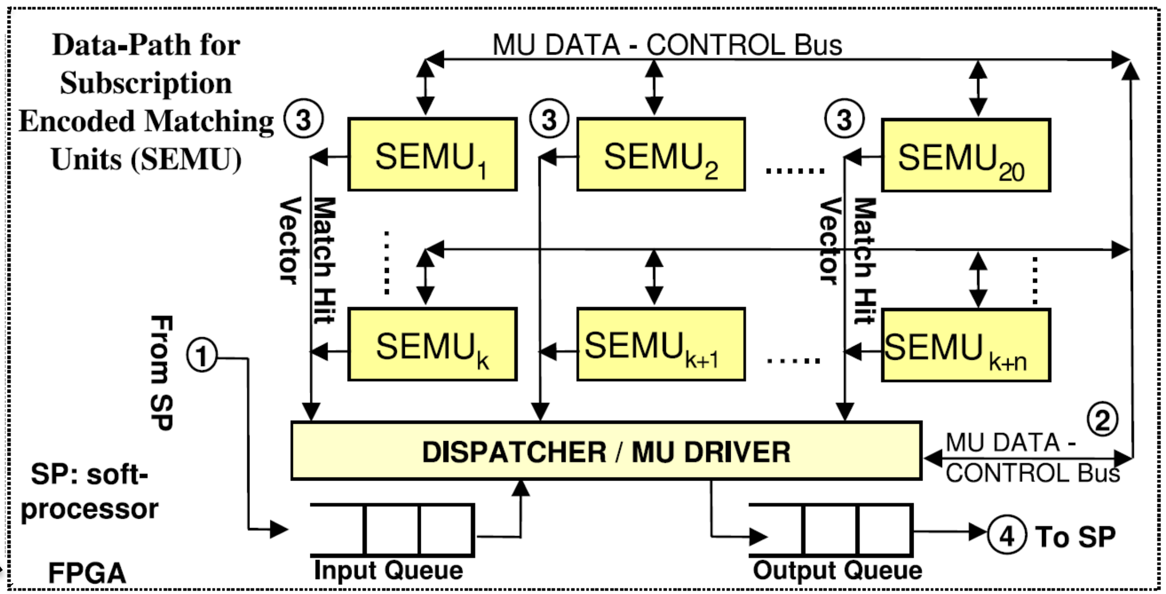
coupling logic & dedicated on-chip memory and eliminating global communication by horizontal data partitioning

Sadoghi, Labrecque, Singh, Shum, Jacobsen, Efficient event processing through reconfigurable hardware for algorithmic trading, PVLDB’10
Sadoghi, Singh, Jacobsen. Towards highly parallel event processing through reconfigurable hardware. DaMoN’11
Sadoghi, Singh, Jacobsen. fpga-ToPSS: line-speed event processing on FPGAs. DEBS’11

Figure Credits: Sadoghi, Labrecque, Singh, Shum, Jacobsen

fpga-ToPSS: Event Processing Design Landscape



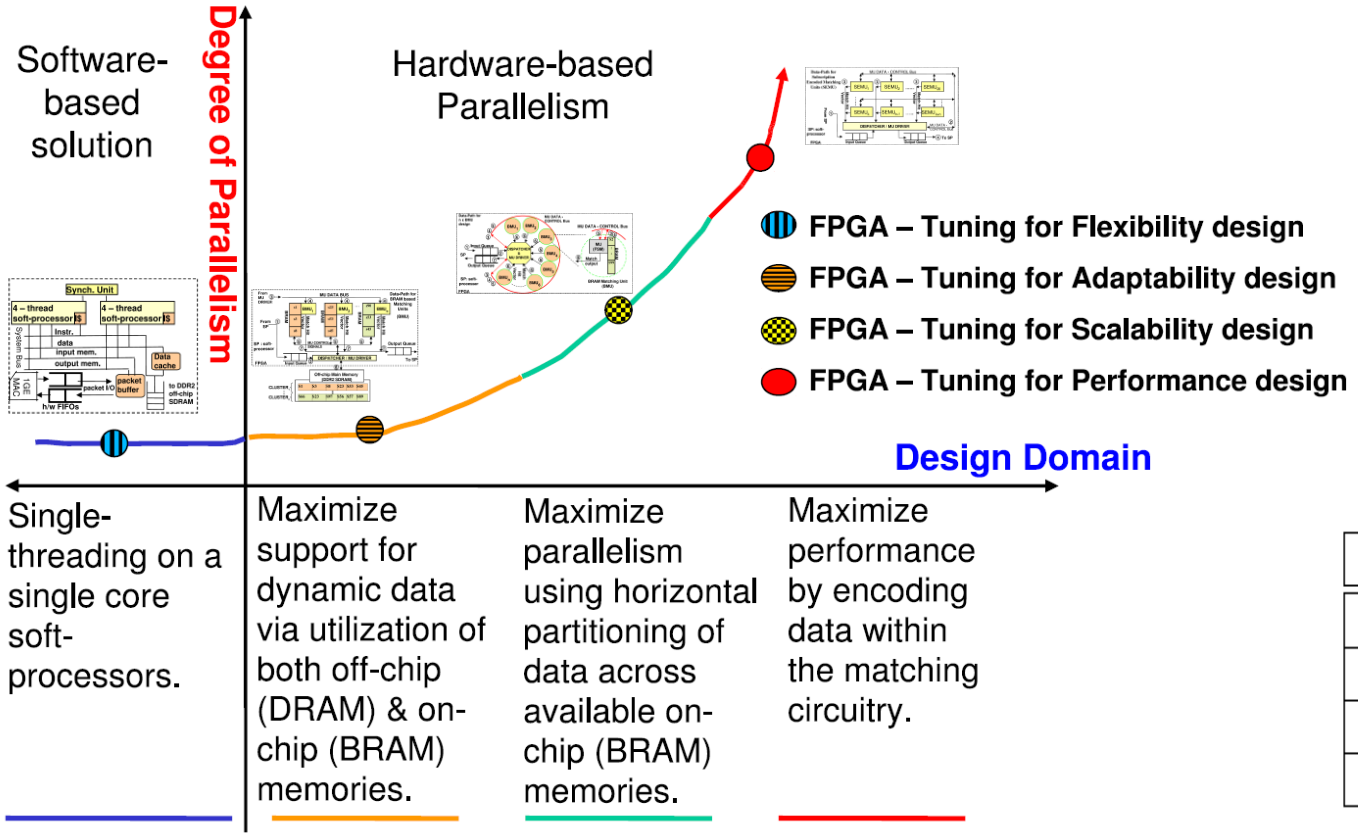


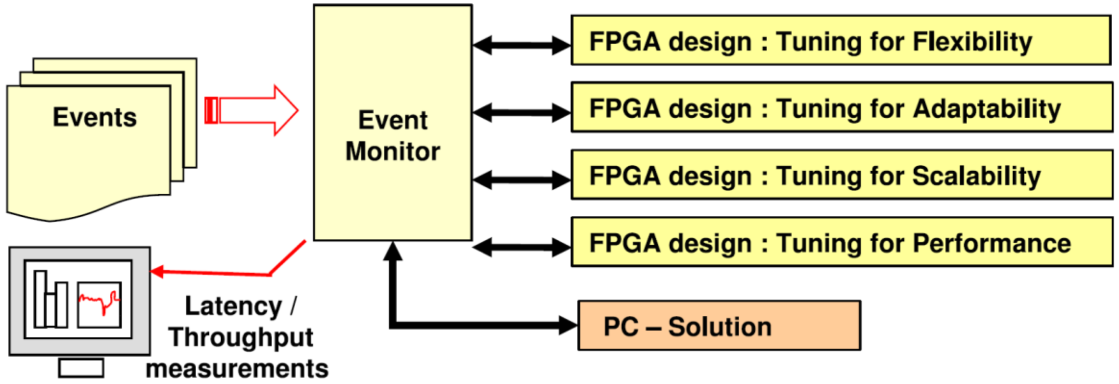
encoding each query as a custom logic block for maximum parallelism

Sadoghi, Labrecque, Singh, Shum, Jacobsen, Efficient event processing through reconfigurable hardware for algorithmic trading, PVLDB'10
Sadoghi, Singh, Jacobsen. Towards highly parallel event processing through reconfigurable hardware. DaMoN'11
Sadoghi, Singh, Jacobsen. fpga-ToPSS: line-speed event processing on FPGAs. DEBS'11

Figure Credits: Sadoghi, Labrecque, Singh, Shum, Jacobsen

fpga-ToPSS: Event Processing Design Landscape





Workload	PC	Flexibility	Adaptability	Scalability	Performance
250	**7.45**	0.89	17.15	**45.04**	**62.29**
1K	**4.01**	3.09	12.31	**29.66**	N/A
10K	**0.58**	0.04	0.72	**19.30**	N/A
100K	0.031	0.01	0.05	N/A	N/A

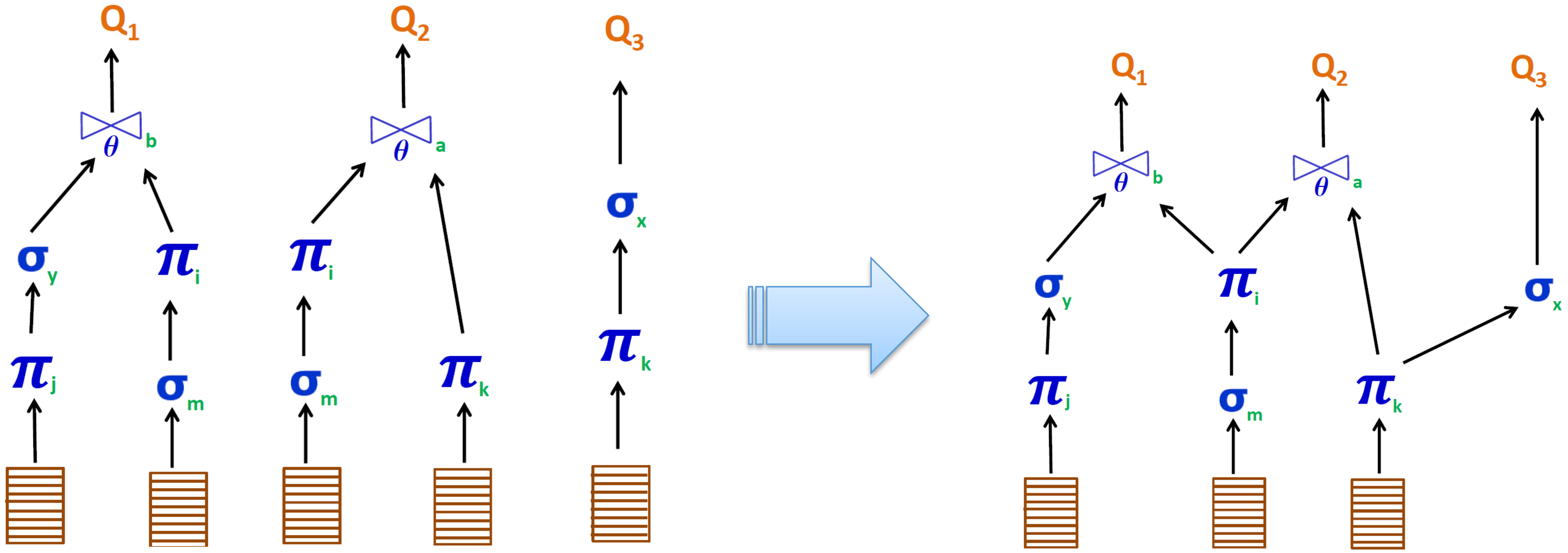
Percentage of Line-rate Utilization

achieving line-rate processing

Sadoghi, Labrecque, Singh, Shum, Jacobsen, Efficient event processing through reconfigurable hardware for algorithmic trading, PVLDB'10
Sadoghi, Singh, Jacobsen. Towards highly parallel event processing through reconfigurable hardware. DaMoN'11
Sadoghi, Singh, Jacobsen. fpga-ToPSS: line-speed event processing on FPGAs. DEBS'11

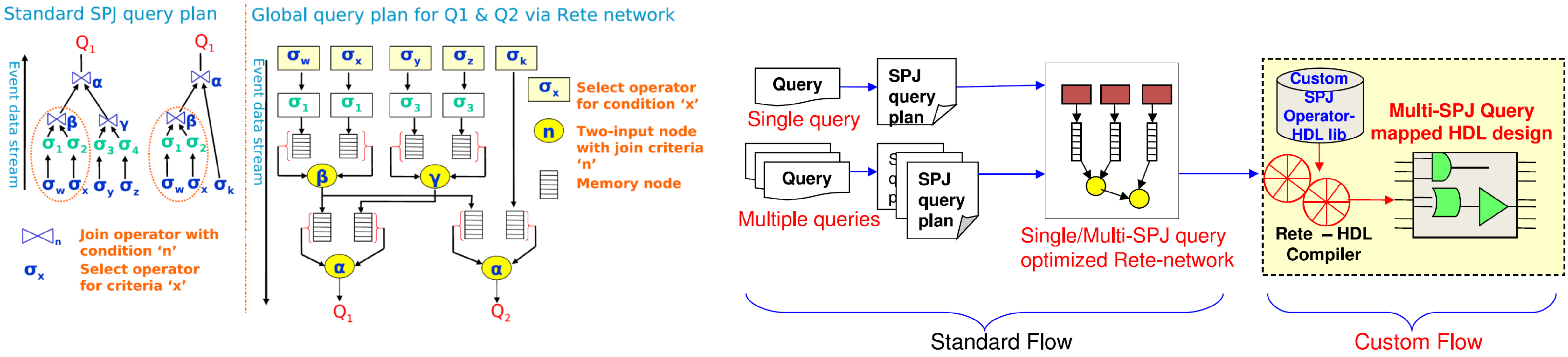
Figure Credits: Sadoghi, Labrecque, Singh, Shum, Jacobsen

Global Query Plan: Rete-like Operator Network



exploiting inter- and intra-parallelism by constructing a global query plan

Global Query Plan: Rete-like Operator Network

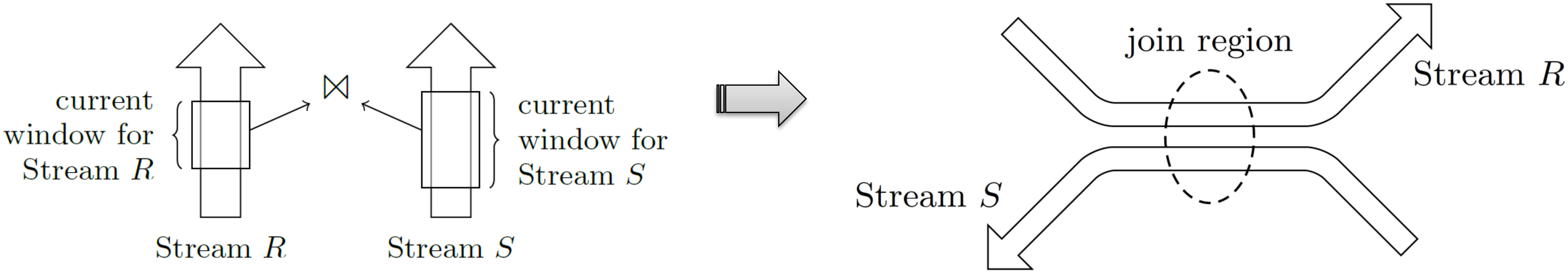


compiling multiple queries into a global query plan on FPGAs

Sadoghi, Javed, Tarafdar, Singh, Palaniappan, Jacobsen, Multi-query stream processing on FPGAs,, ICDE'12

Figure Credits: Sadoghi, Javed, Tarafdar, Singh, Palaniappan, Jacobsen

Redefining Joins Using Data Flow

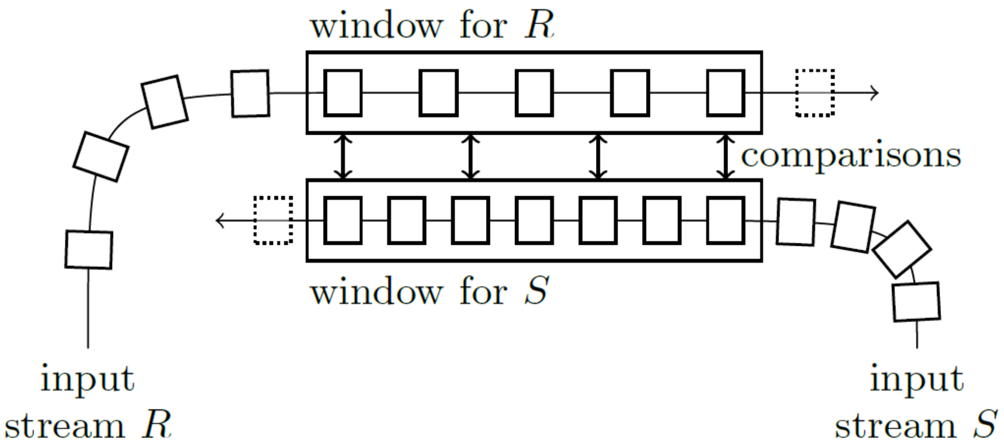


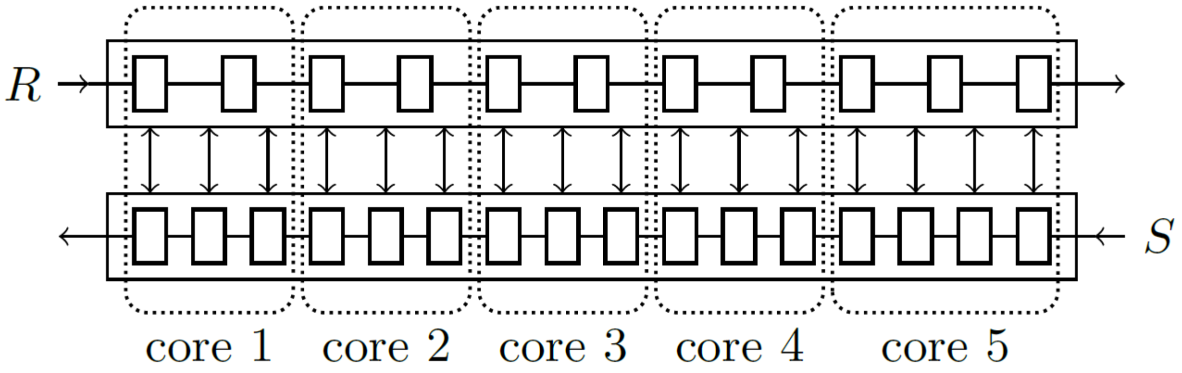
introducing bi-directional data flow to naturally direct streams in opposite directions to eliminate complex control flow

Teubner Mueller. How soccer players would do stream joins. SIGMOD'11
Roy, Teubner, Gemulla, Low-latency handshake join, PVLDB'14

Figure Credits: Teubner Mueller, Roy, Teubner, Gemulla

Redefining Joins Using Data Flow



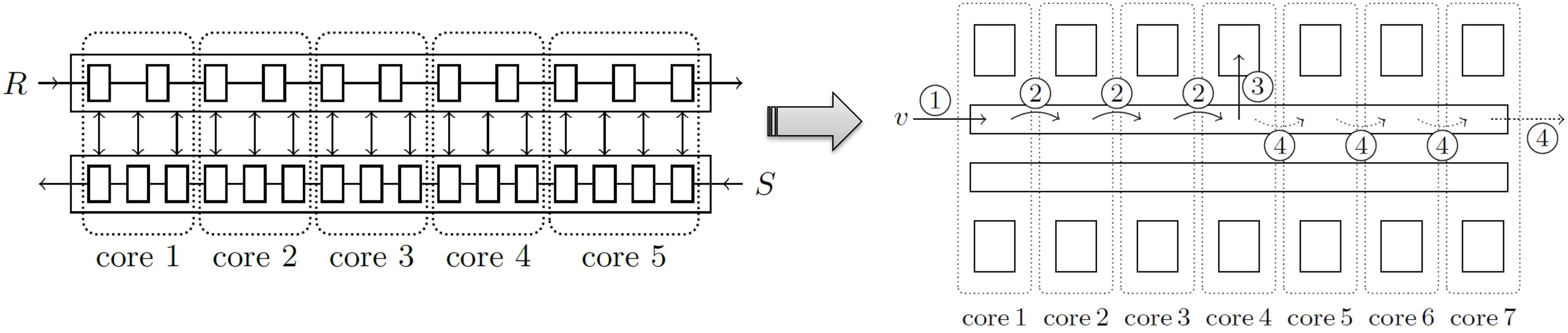


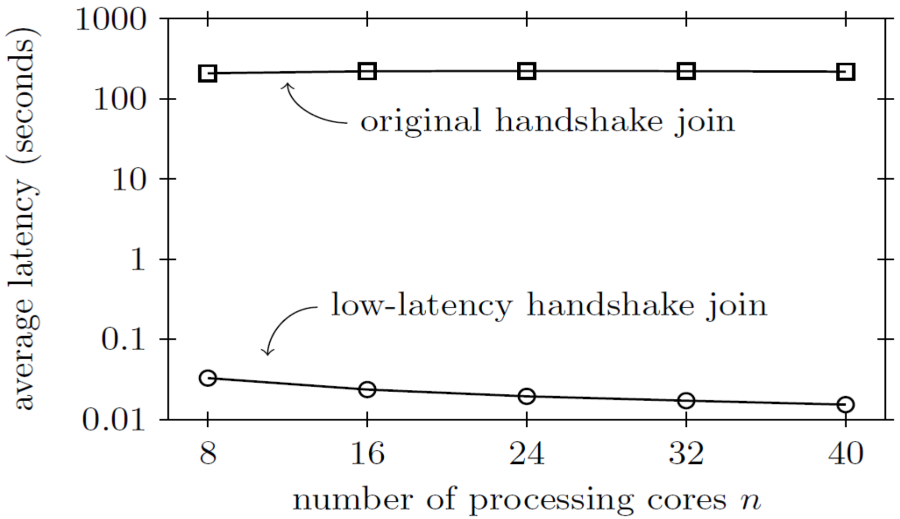
introducing bi-directional data flow to naturally direct streams in opposite directions to eliminate complex control flow

Teubner Mueller. How soccer players would do stream joins. SIGMOD'11
Roy, Teubner, Gemulla, Low-latency handshake join, PVLDB'14

Figure Credits: Teubner Mueller, Roy, Teubner, Gemulla

Redefining Joins Using Data Flow



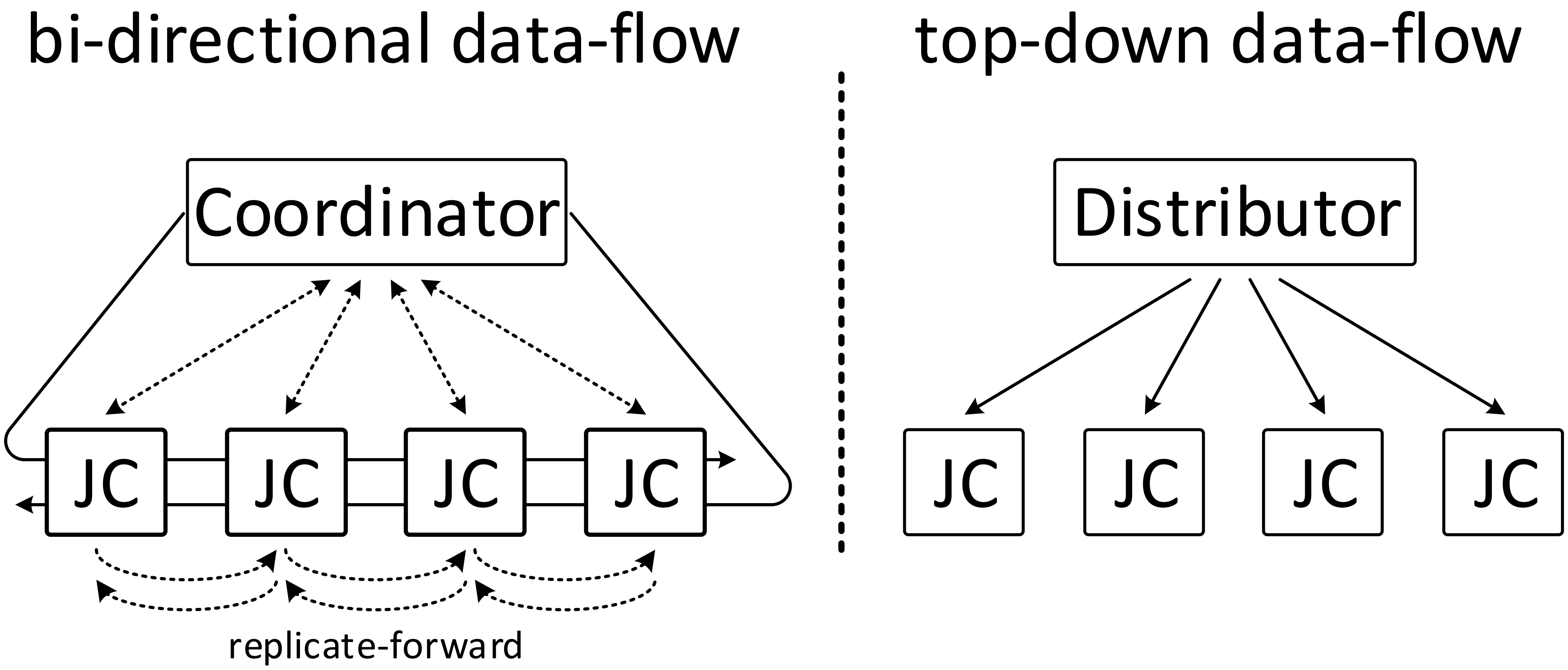


introducing bi-directional data flow to naturally direct streams in opposite directions to eliminate complex control flow

Teubner Mueller. How soccer players would do stream joins. SIGMOD'11
Roy, Teubner, Gemulla, Low-latency handshake join, PVLDB'14

Figure Credits: Teubner Mueller, Roy, Teubner, Gemulla

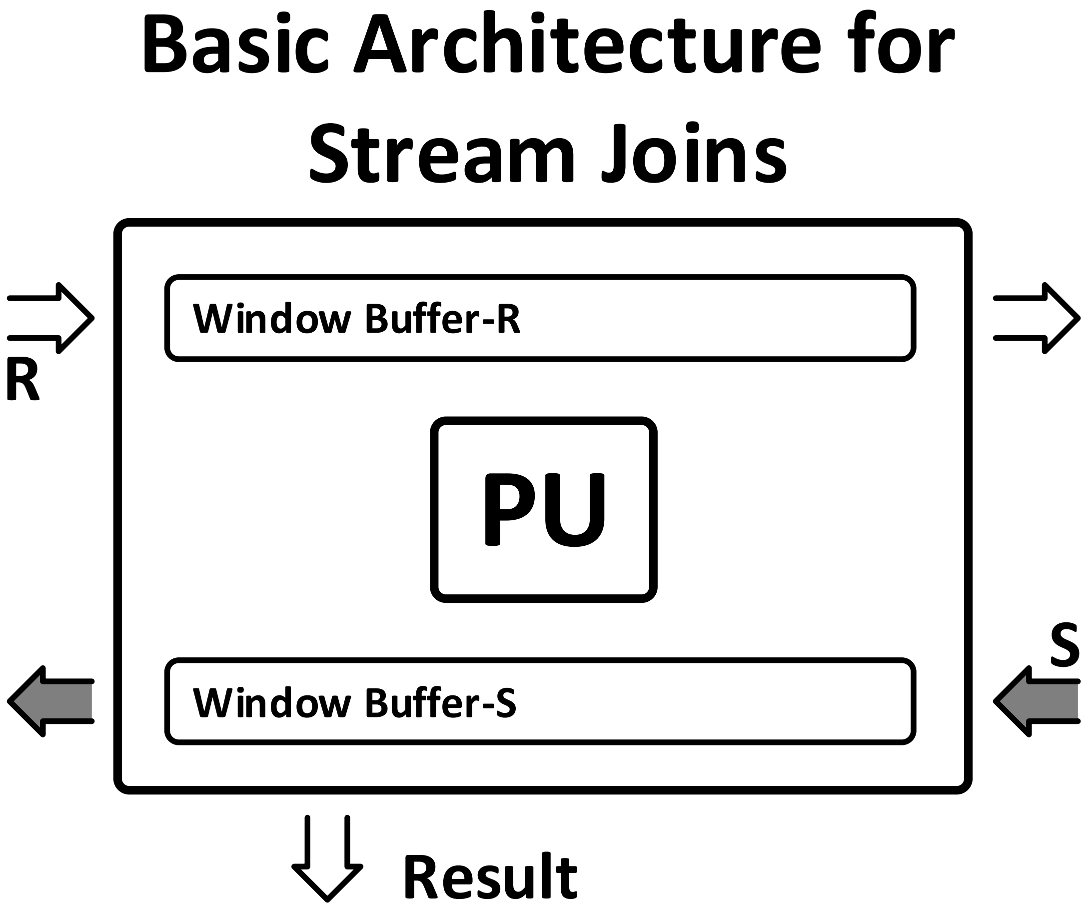
Revisiting Data Flow Model



rethinking the data flow to eliminate the control flow

Figure Credits: Najafi, Sadoghi, Jacobsen

SplitJoin: Introducing Top-Down Flow Architecture

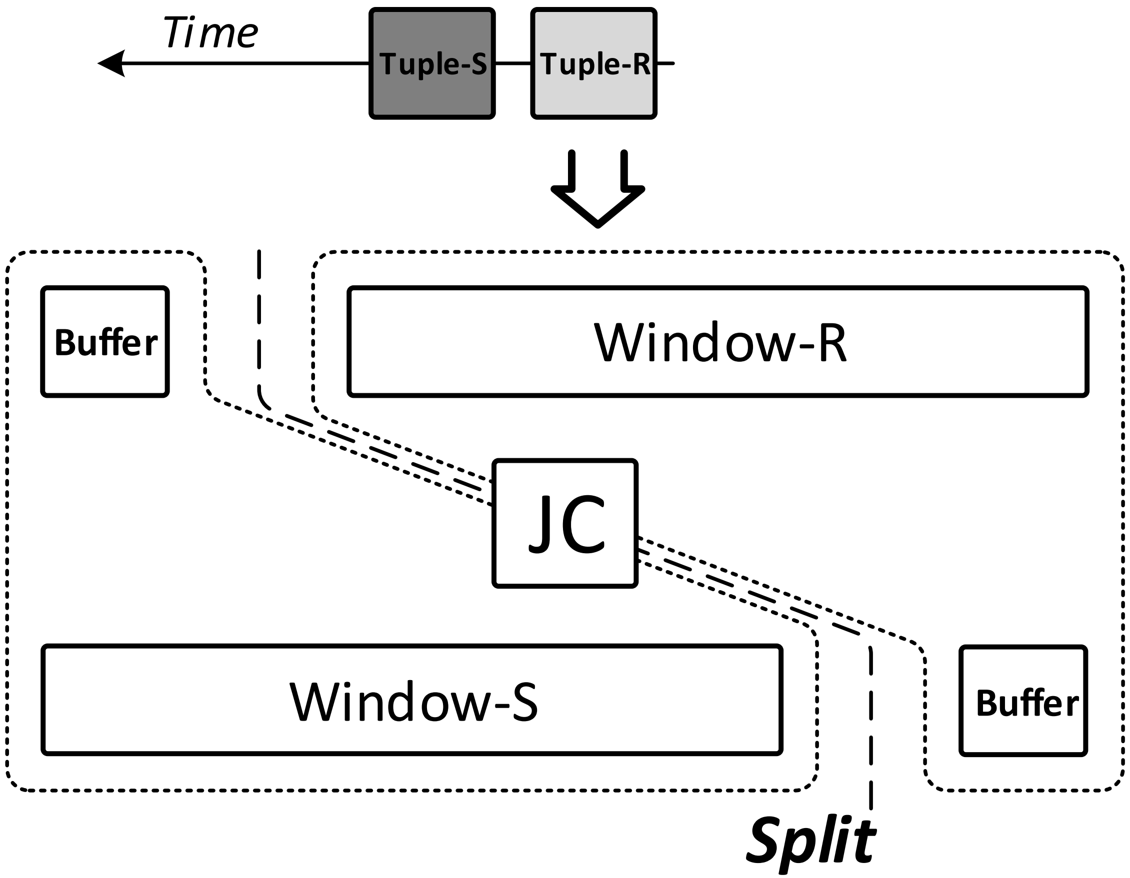


rethinking the data flow to eliminate the control flow & split the join into concurrent/independent "store" & "process" steps

Najafi, Sadoghi, Jacobsen. SplitJoin: A Scalable, Low-latency Stream Join Architecture with Adjustable Ordering Precision. USENIX ATC'16

Figure Credits: Najafi, Sadoghi, Jacobsen

SplitJoin: Introducing Top-Down Flow Architecture

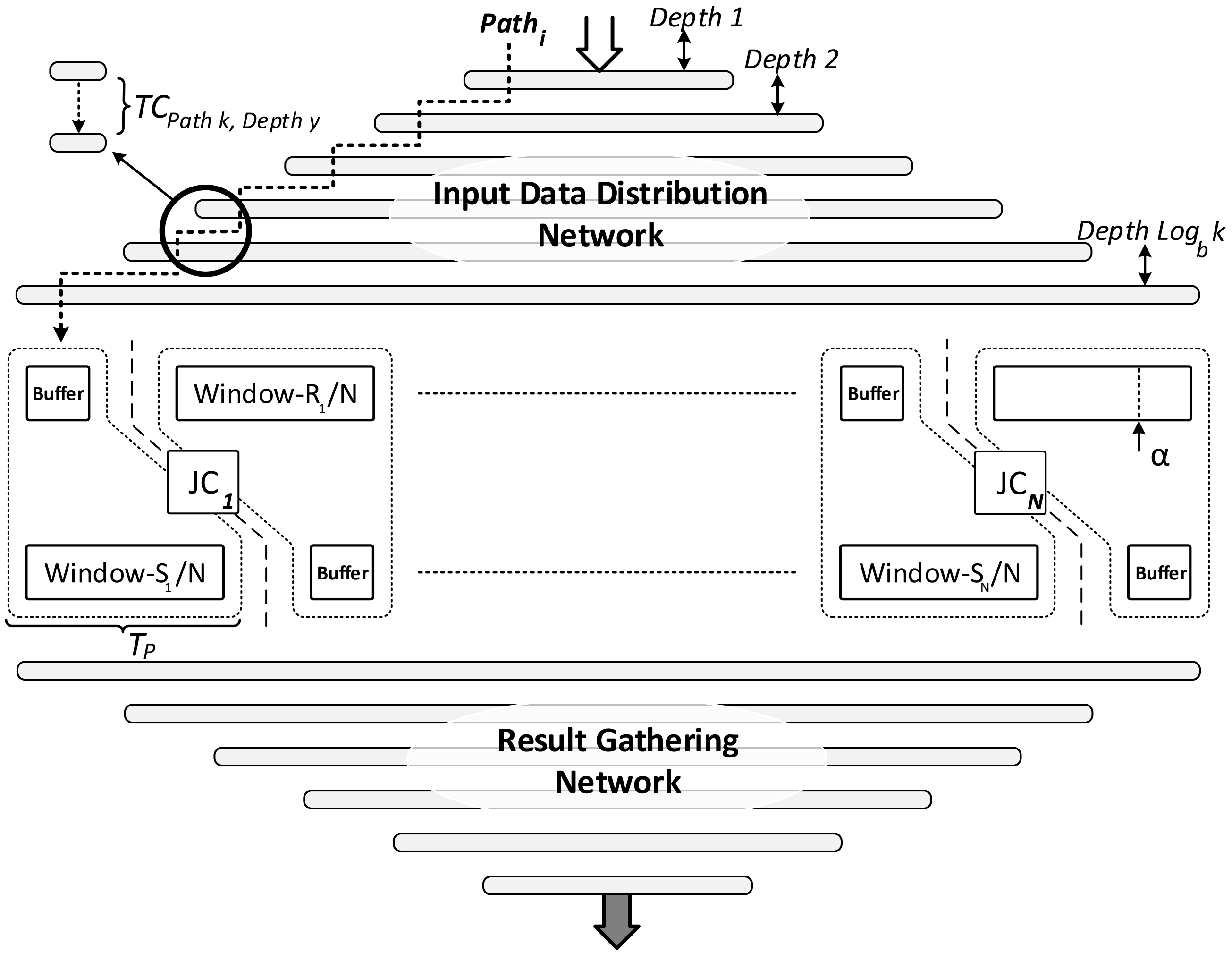


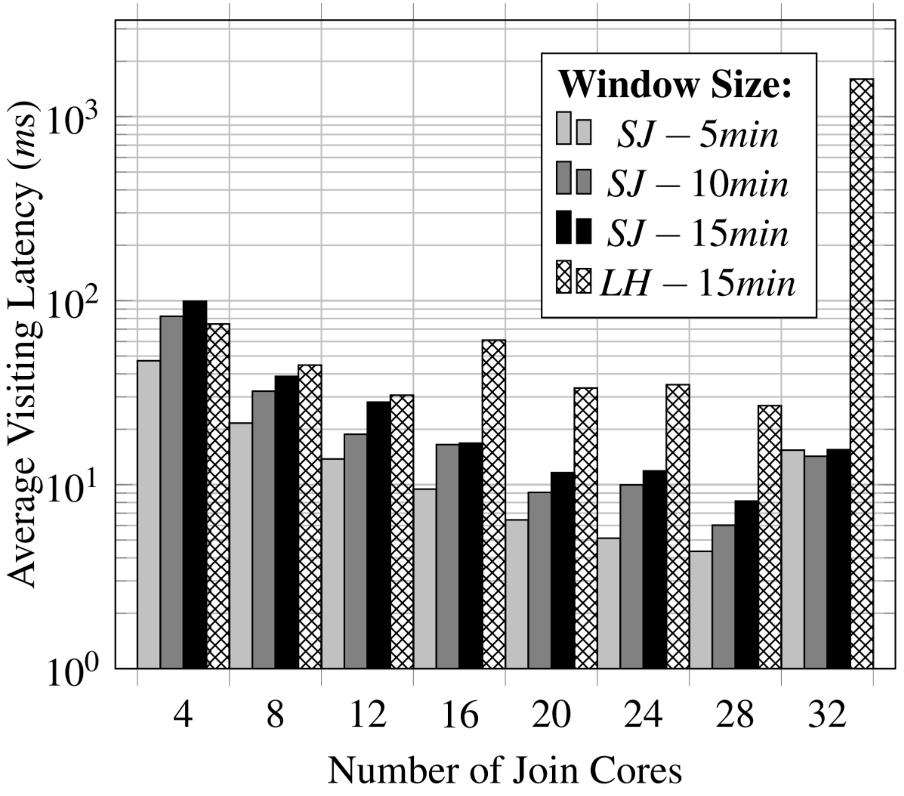
rethinking the data flow to eliminate the control flow & split the join into concurrent/independent “store” & “process” steps

Najafi, Sadoghi, Jacobsen. SplitJoin: A Scalable, Low-latency Stream Join Architecture with Adjustable Ordering Precision. USENIX ATC’16

Figure Credits: Najafi, Sadoghi, Jacobsen

SplitJoin: Introducing Top-Down Flow Architecture





rethinking the data flow to eliminate the control flow & split the join into concurrent/independent "store" & "process" steps

Najafi, Sadoghi, Jacobsen. SplitJoin: A Scalable, Low-latency Stream Join Architecture with Adjustable Ordering Precision. USENIX ATC'16

Figure Credits: Najafi, Sadoghi, Jacobsen

Future Directions/Open Questions

- What is the best initial topology given a query workload as a prior?
  - Can we construct a topology in order to reduce routing (i.e., to reduce the wiring complexity) or to minimize chip area overhead (i.e., to reduce the number of logic blocks).
- What is the complexity of query assignment to a set of custom hardware blocks?
  - A poorly chosen query assignment may increase query execution time, leave some blocks unutilized, negatively affect energy use, and degrade the overall processing performance.
- How to formalize query assignment algorithmically (e.g., developing cost models)?
  - Going beyond classical join reordering and physical plan selections, there is a whole new perspective on how to apply instruction-level and fine-level memory-access optimization.
  - What is the most efficient method for wiring custom operators to minimize the routing distance?
  - How to collect statistics during query execution, and how to introduce dynamic re-wiring and movement of data given a fixed hardware topology?
- Given the topology and the query assignment formalism, how do we generalize from single-query optimization to multi-query optimization?
- How do we extend query execution on hardware to co-processor & co-placement designs by distributing and orchestration query execution over heterogeneous hardware (e.g., CPUs, FPGAs, and GPUs) by exploring different placement arrangement on the path of data?

Najafi, Sadoghi, Jacobsen. The FQP vision: Flexible query processing on a reconfigurable computing fabric, SIGMOD Record'15

GPU Acceleration (Module II)

Graphics Processing Units (GPUs): Background

- Designed primarily as specialized processors for accelerating key graphics workloads (e.g., computer graphics, animation, virtual reality, games, etc.)
  - Wide-spread usage from enterprise servers, embedded (e.g., drones), to mobile (cell phones)
- Implements various core graphics operators in hardware, e.g., resterization, texture mapping, or shader processing
  - Most core operations involve matrix and vector calculations over floating point numbers
- GPUs built as a powerful parallel processing system for floating point calculations
  - Support large memory bandwidth to match the real-time graphics constraints
- Increasing usage in non-graphics applications (general purpose/GP) due to significant compute and memory capabilities
  - Scientific computing, deep learning, drones, self-driving cars….
- Examples: Nvidia Pascal (Tesla P100), AMD Radeon Polaris, Nvidia TX1,..

Graphics Processing Units: System Architecture

- Modern GPUs are, in reality, massively parallel computers on a chip
- Key design factors: Power consumption and compute/memory capabilities
- Compute cores and memory packaged as a card and connected to a host system (CPU and memory) via a CPU-GPU interface (usually, PCI-e)
- GPU cards have additional ports to connect with each other
- GPU memory (device memory) connected the compute cores via high speed interconnect
  - Device memory size limits the amount of ”on-device” data. Current device memory size 24 GB.
  - GPU internal memory bandwidth very high (700 GB+ for the Nvidia Pascal GPU)

Nvidia Pascal (Tesla P100) Card

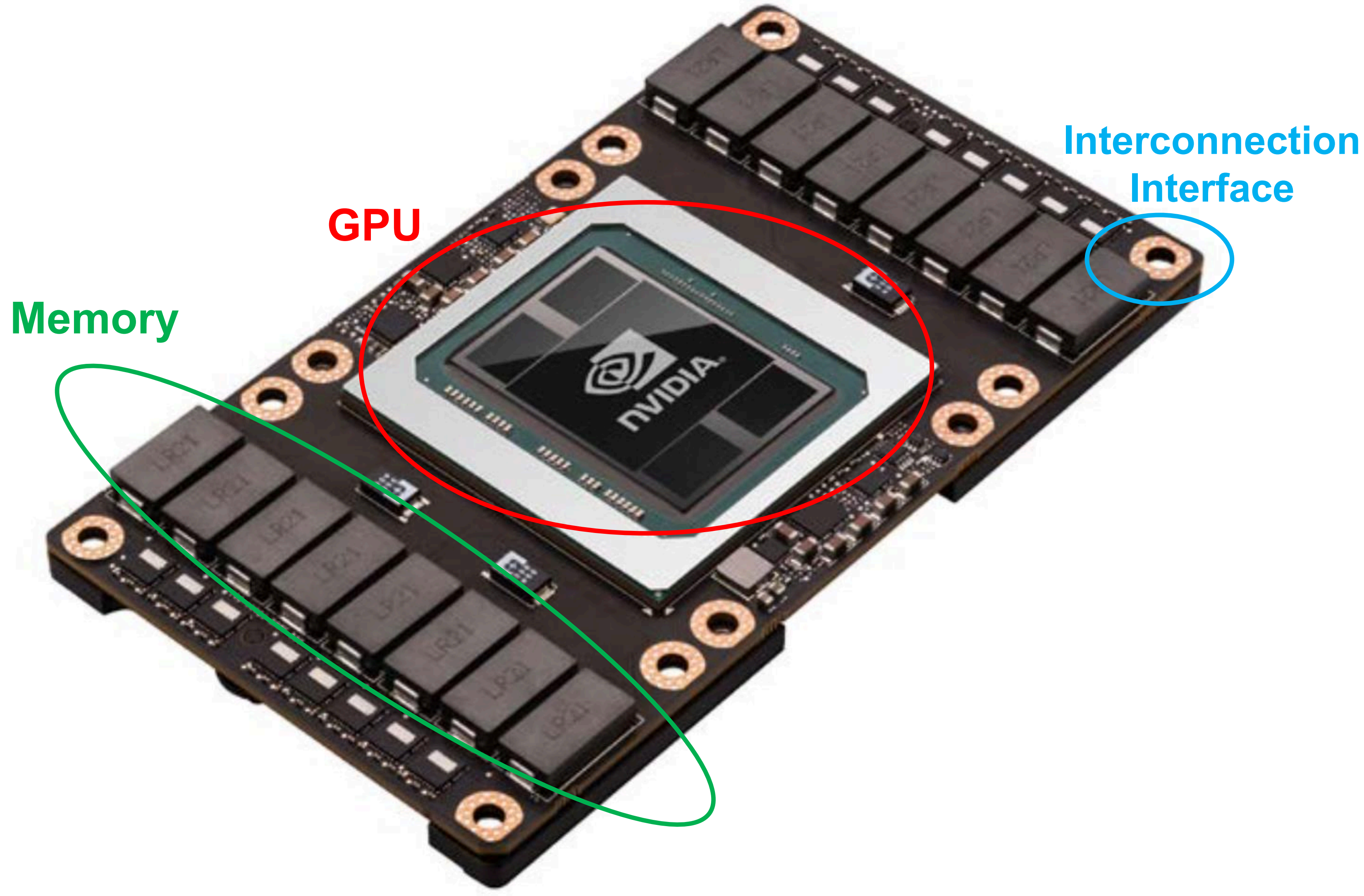
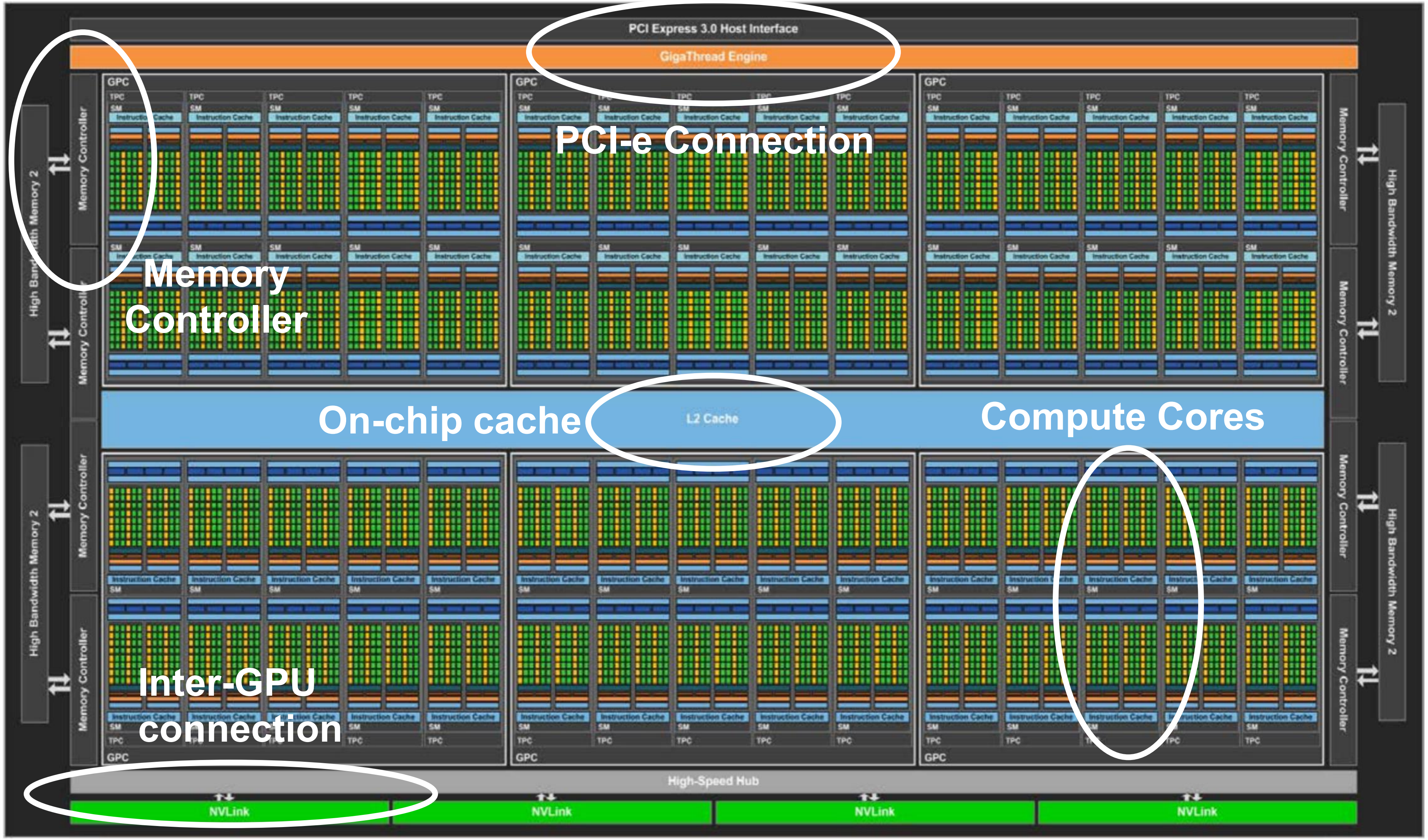


Figure Credit: https://devblogs.nvidia.com/parallelforall/inside-pascal/

Graphics Processing Units: Processor Architecture

- Massively multi-threaded data-parallel processor.
- Follows the Single-instruction Multiple Thread (SIMT) programming model
- Built using computational blocks: Symmetric Multiprocessors, Texture processing Clusters and Graphics Processing Clusters
- Supports multiple types of memories
  - Off-chip device memory
  - Off-chip texture memory
  - On-chip local memory, and constant memory
  - On-chip L1 and L2 caches, and registers
- Supports FP64, FP32, and FP16 types (in addition to the integer family)
- Peak FP32 GFLOPs 10.6 TF, FP64 5.3 GFLOPs
- Memory Bandwidth: 700 GB/s+
- Power consumption: 300 W

Nvidia Pascal (Tesla P100) Processor Architecture



Nvidia Pascal Micro-architecture

Single-precision Compute Unit

Texture Memory

Graphics Processing Units: Usage

- GPUs are hybrid accelerators: need CPUs to control the execution of functions
  - GPU-based code invoked as kernels from the CPU host
  - Data is usually copied to and from the host
- GPU usually connected to the CPU hosts via PCI-e connection (peak 16 GB/s unidirectional)
- New interconnection fabric, called NVLINK, will be used for both CPU-GPU and inter-GPU connections (peak 40 GB/s bidirectional)
- GPUs can be connected together to form a multi-GPU system, that can be managed by a group of host CPUs
  - GPUDirect provides fast P2P data transfer across GPUs
- Factors affecting GPU system configurations (choice of GPU)
  - Type of computation: Compute-bound (Single or Double precision) or memory bound
  - Memory footprint of the workload
  - Compute intensity

General Purpose GPUs (GPGPUs): Execution Model

- GPU execution model is a hybrid accelerator model
  - Host system and GPU resources (compute and memory) are managed and used separately
  - GPU functions executed as non-blocking kernels
  - Nvidia's unified programming model (UVM) enables operations on the global memory address space that spans host and device memories
- GPU device execution model supports multiple types of parallelism
  - Massive data parallelism via light-weight threads
  - Shared address space programming
  - Distributed memory programming via using thread-blocks
- GPU is a throughput-oriented processor
  - Massive data parallelism is needed to hide memory access costs

GPGPU Programming Models

- GPGPU programming models support massive data parallelism using the single-instruction multiple-thread (SIMT) approach
- Two widely used programming languages (C/C++ extensions): CUDA and OpenCL
- The programming languages provide user abstractions:
  - to partition the computations over multiple threads using different parallelization approaches
  - to allocate data in different memory regions
  - to manage the mapping and scheduling of threads over underlying hardware
- Additional workload-specific libraries (e.g., CUBLAS, CUSPARSE, CUDNN..)

Key GPGPU Performance and Functionality Issues

- GPU is a throughput machine
  - Need to use a large of threads to hide memory latency
- Memory access performance depends on many factors
  - Best performance when logically consecutive threads access physically closer location (GPU hardware coalesces multiple thread accesses)
  - Read-only memory accesses perform much better than update accesses
    - Random reads optimized using texture memory
  - Accesses to host memory via Unified Virtual Memory is not fast
- Conditional execution between threads leads to thread serialization
  - Warps share the instruction counter
- Unaligned and non-uniform memory access degrade performance due to un-coalesced memory accesses
- Atomic operations restricted to 32 and 64 bit variables
- For performance, most data structures need to be array-based
- Limited support for dynamic memory allocation

Issues in exploiting GPUs for Database Processing

- Database processing not compute (FLOPS) intensive
  - Traditional database execution I/O bound
  - Most in-memory calculations involve memory traversals and comparisons
    - Non-iterative. Calculate-to-access ratio very low (usually 1)
    - Many situations involve non-contiguous memory accesses (e.g., hash tables)
  - Only numerically-intensive tasks include OLAP reductions, statistics calculations, and classical analytics extensions
  - Transaction processing update-intensive
- Data being processed usually larger than the GPU device memory
- Data stored in specialized formats
  - Row-wise records, Columnar stores, Log files, Tree-based indices
- A variety of data types: Need to process data for GPU operations
  - Variable-length character arrays (Varchars), Double-precision, Date, BLOBs
- Fast access to storage sub-systems

Potential Exploitation of GPUs in Database Processing

- Exploitation of GPU's High internal Bandwidth
  - Sorting (for ordering/grouping elements, Join operations)
  - Hashing (for Joins, Indexing)
  - Predicate Evaluations (fast comparisons)
  - Dynamic programming for selecting query plans
- Exploitation of GPU's Numerical Capabilities
  - OLAP Reductions
  - Utilities: Compression, Encryption, Statistical Calculations
  - Analytical Extensions: Top-K, Text, Spatial, and Time-series analytics

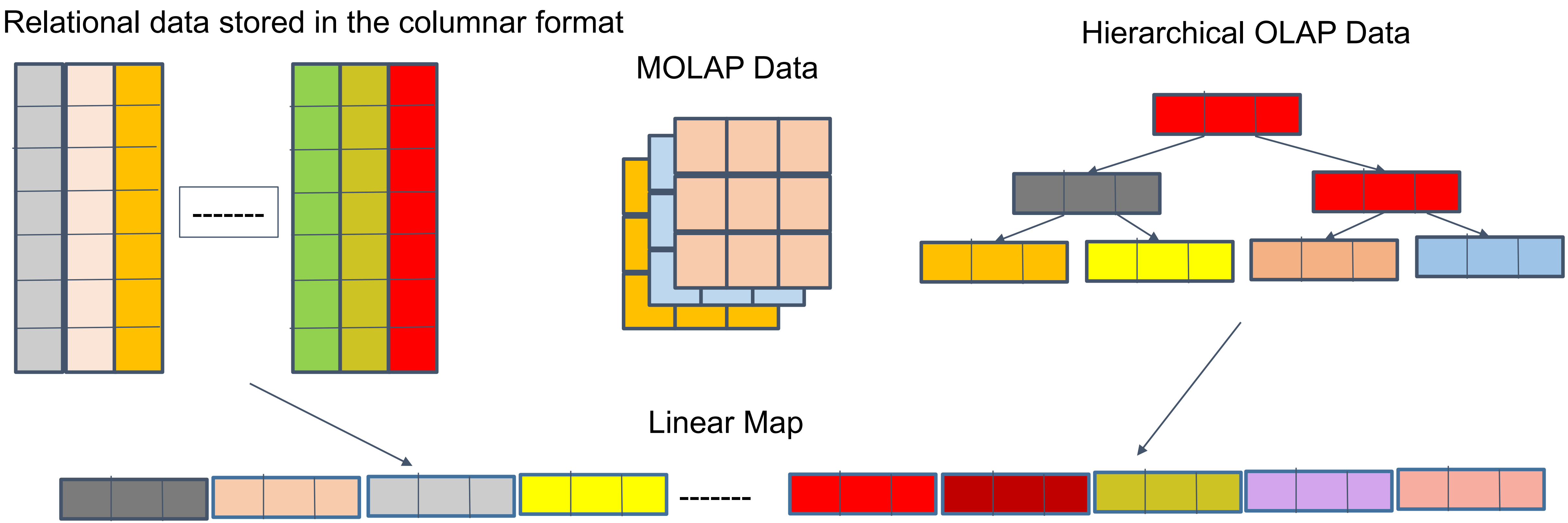
GPU-Accelerated Database in Practice

- OLAP Acceleration
  - Jedox
- Graph Analytics
  - MapD, gpudb, BlazeGraph
- Database functional acceleration (hashing, sorting)
- Relational query execution on the hybrid CPU+GPU systems
  - Sqream, DeepCloud Whirlwind
- GPU Support for key database infrastructures, e.g., index
- GPU acceleration of XML workloads

GPU exploitation for Database Processing

- OLAP (Both relational and MOLAP)
  - Exploiting GPU's memory bandwidth for data access and grouping
  - Exploiting GPU's numerical capabilities for data aggregation
- Optimized GPU libraries for key database primitives (Need to consider multiple types, sizes, and other constraints)
  - Sorting
  - Hash functions (e.g., bloom filters) and its applications for joins, grouping
  - Joins, Selection and Projection
    - Hash-Join vs. Sorted-Join
    - Revisit Nested-loop Joins- may perform better on GPUs due to regular access patterns
  - Graph Analytics: Trees, graphs, DAGs (for NoSQL data processing)
  - Statistical libraries: Histograms, Frequency counting, Unique items, Sampling,
  - Analytical libraries: String processing, Top-K,..

OLAP Acceleration: Fast Data Access



- Linear array can be accessed in parallel by very large numbers of threads (in millions)
- For data in the device memory, read-only data access bandwidth can exceed 300 GB/s
- For random reads, data can be mapped to texture memory
- For very large data sets, GPUs can access data directly from the host main memory using unified virtual memory

Hash-Based Group By/Aggregate

SELECT C1, SUM(C2) FROM Simple_Table GROUP BY C1

Simple_Table

KEY	Value
93	5
23	2
93	1
23	5
93	0
93	1000

Parallel HM Creation

Hash Map

Key	Aggregated Value
23	7
93	1006

Parallel Probe

Result

23	7
93	1006

Towards a Hybrid Design for Fast Query Processing in DB2 with BLU Acceleration Using Graphical Processing Units: A Technology Demonstration, S. Meraji, B. Schiefer, L. Pham, L. Chu, P. Kokosielis, A. Storm, W. Young, C. Ge, G. Ng, K. Kanagaratnam, SIGMOD16 (To appear)

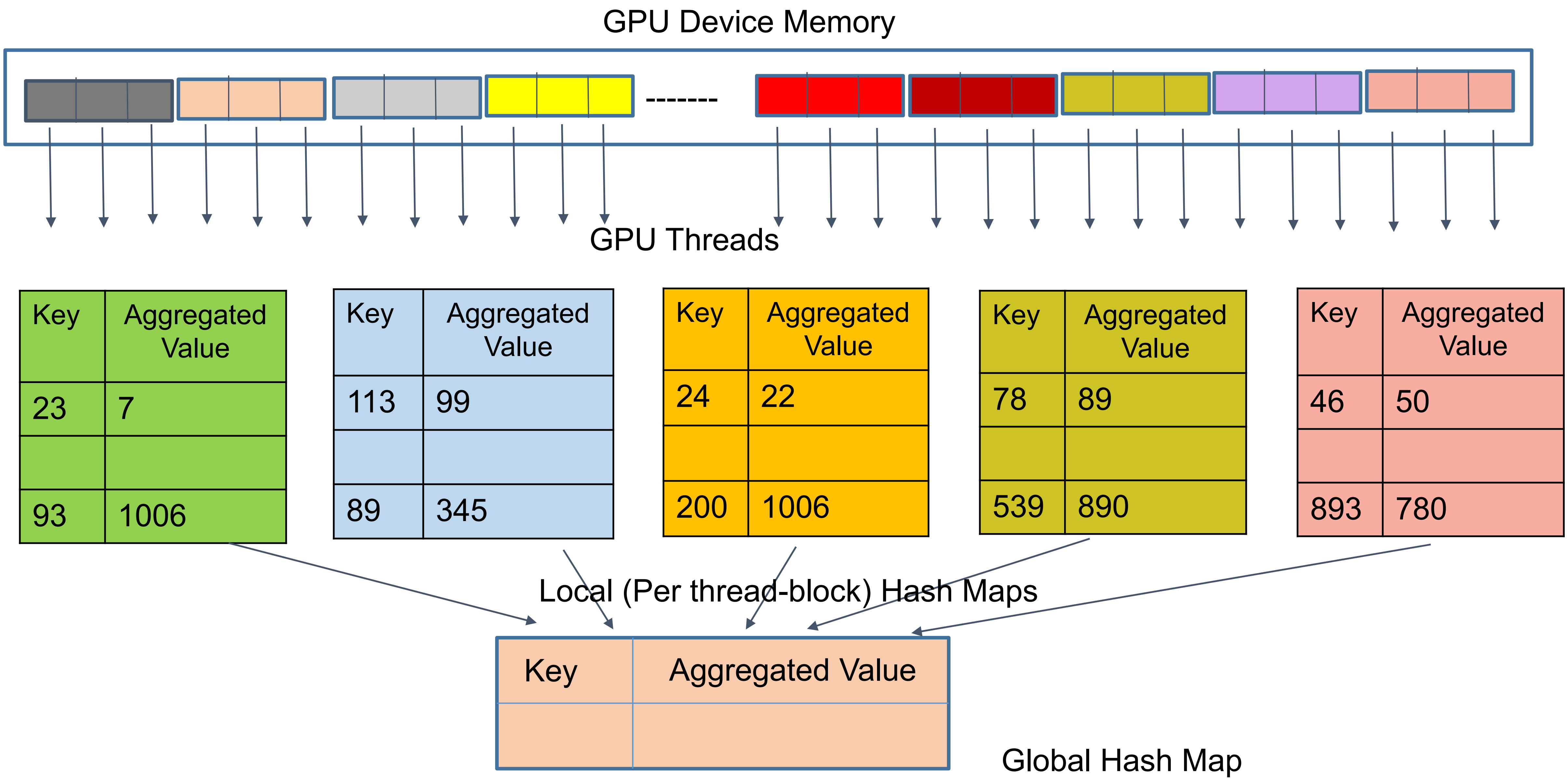
OLAP Acceleration: Parallel Grouping and Aggregation

- Grouping involves creating hash maps in the GPU's memories
- Use Murmur or mod hash functions
- Each insertion into the hash map simultaneously performs the reduction operation (e.g., add or max)
- The GPU kernel can use thousands of threads. Each thread
  - Accesses the input data
  - Computes hash value
  - Inserts into the target location and uses atomic operation to execute the reduction operation
- Multi-level hash maps used to compute final result

- GPU's massive data parallelism enables extremely fast reduction operations
  - Sum, Min, Max
  - Average, Median, etc.
- Nvidia GPUs provide fast atomic operations on 32-bit values
  - atomicAdd(), atomicCAS()
  - Can be extended to support 64-bit variables
- For smaller number of groups, hash maps can be created in the shared memory

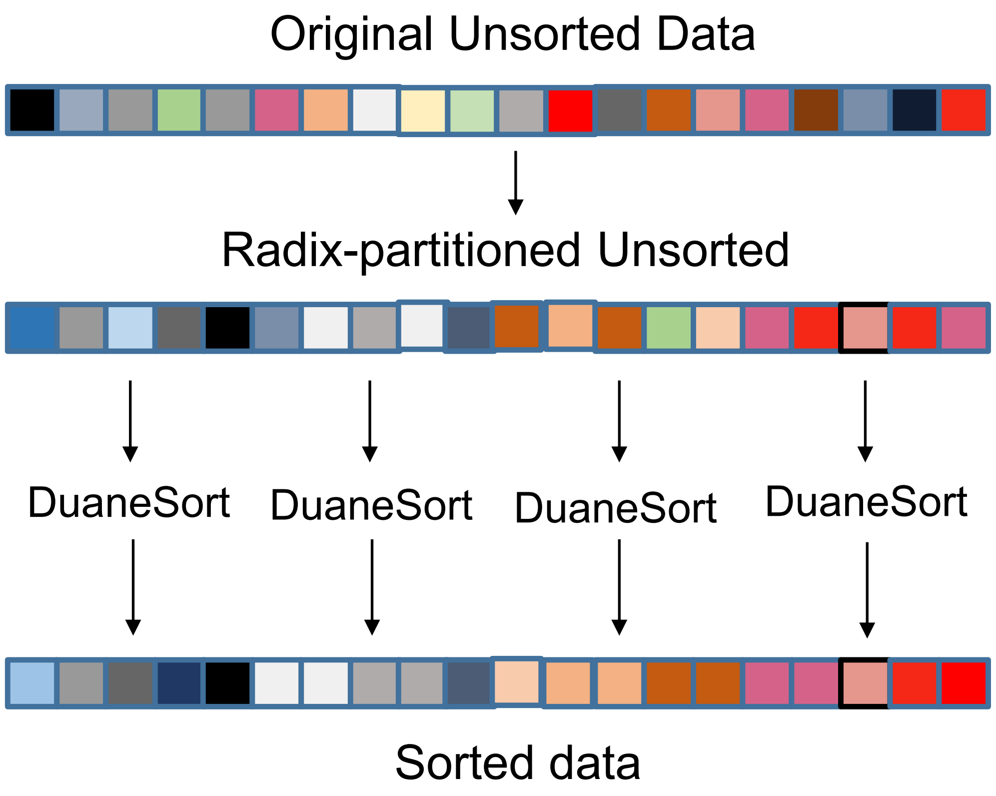
Towards a Hybrid Design for Fast Query Processing in DB2 with BLU Acceleration Using Graphical Processing Units: A Technology Demonstration, S. Meraji, B. Schiefer, L. Pham, L. Chu, P. Kokosielis, A. Storm, W. Young, C. Ge, G. Ng, K. Kanagaratnam, SIGMOD16 (To appear)

End-to-end OLAP Acceleration via Group-By Aggregation



GPU-Accelerated Utilities: Sorting

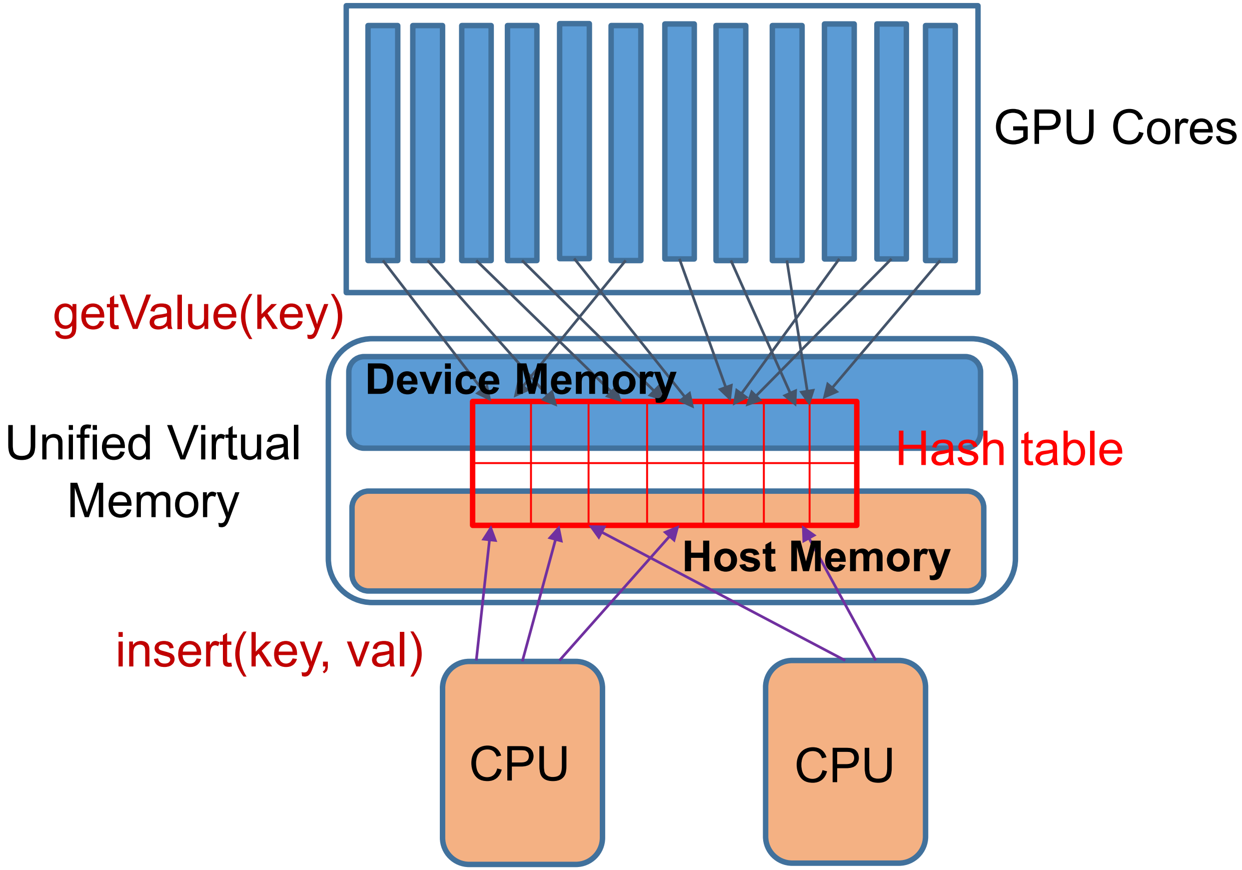
- GPU can exploit its memory capabilities for sorting datasets that can fit in the device memory (16 GB)
- GPU-based sorting works very well for aligned data sets (key and values)
  - The best GPU sort is based on the radix sort algorithm (Duane Sort)
- Sorting large datasets on GPUs uses hybrid approaches
  - GPU sort is used as a kernel by the CPU sorting algorithm
  - Aligned versions of the original key/values used for the GPU sort
  - Can operate on very large datasets



Merrill, Grimshaw. High Performance and Scalable Radix Sorting: A Case Study of Implementing Dynamic Parallelism for GPU Computing. Parallel Computing Letters'11
Cho, Brand, Bordawekar, Finkler, Kulandaisamy, Puri. PARADIS: An Efficient Parallel Algorithm for In-place Radix Sort., PVLDB'15

GPU-Accelerated Utilities: Hashing

- Hashing can exploit GPU's high read-only bandwidth (300 GB/s+)
- GPU acceleration better suited for hash probing, rather than insertion which results in random updates
- Performance also affected by atomic operation supports
  - Only supports aligned keys
- For data larger than device memory, hash table built using unified virtual memory
- Examples: Cuckoo Hash, Hybrid CPU-GPU hash, and Stadium Hash



Alcantara, Sharf, Abbasinejad, Sengupta, Mitzenmacher, Owens, Amenta. Real-time Parallel Hashing on the GPU. ACM TOG'09
R. Bordawekar, Nvidia.Evaluation of Parallel Hashing Techniques. GTC'14
Khorasani, Belviranli, Gupta, Bhuyan. Stadium Hashing: Scalable and Flexible Hashing on GPUs. PACT'15

Novel Data Management Applications of GPU

- Multi-GPU In-memory Multi-media Database
  - Host CPU acts simply as a controller and request broker
  - GPU device memory used as the primary storage subsystem, share the host memory as a secondary persistent memory
  - GPUs interact with each other via GPUDirect without involving the host
- GPU-accelerated Relational Scientific Database
  - Can be built on any columnar storage (e.g., monetDB is being used for processing astronomical data.)
  - Structured region-based accesses suitable for GPU execution.

GPU Exploitation for Database Workloads: Summary

- Most data management workloads are not compute (FLOP) bound. Exception being reduction operations in the OLAP scenario.
  - GPU's high internal bandwidth is the selling point
  - Aggregation performance on GPU at least 4/5 faster than CPU (SIMD+MT codes)
- Data management workload process large quantities of data
  - Bandwidth of the host-GPU connection is critical (PCI-e or NVLINK)
  - Device memory size key for usability
  - Direct connection with the storage infrastructure necessary
- Data layout optimizations required for GPU usage
  - Columnar layout more suited than row store
- Multiple-GPU scenario more suitable for data management workloads
  - Performance and capabilities of the GPUDirect functionality very relevant
- Closer connection with underlying network fabric required for using GPUs in latency-sensitive workloads
- GPU power consumption a key issue in database system building (e.g., appliances)
  - Clusters of Low-power GPUs clusters may be an option

Designing a GPU-accelerated Database System

- Based on the co-processor model of acceleration
- Suitable workloads: Data Warehousing, OLAP, Graph Analytics, ETL Systems
- Factors to consider:
  - Characteristics of the workloads: data size, type of data, runtime constraints
  - System configuration issues
    - Power limits
    - Hardware configuration factors: Packaging and Cooling issues, Power supply
    - Single node or multi-node scenarios
  - Deployment model: On premise or service-based

Multi-core CPU vs. GPU: Data Management Perspective

- CPUs have deeper and larger memory hierarchies than GPU
  - Up to 3-level caches and can host multi-TB of main memory
- Direct access to persistent storage subsystem (e.g., disks, and SSDs)
- Multiple cores with multiple SMT threads, support for short-vector SIMD parallelism (256-bit AVX2, 128-bit VSX). Very high execution frequency (2.5 GHz+)
- Limited Main-Memory Bandwidth (100+ GB/s)
- Reasonable FLOPS performance
- Supports very fast atomic operations
- Can support any general data structure (e.g., linked lists)
- Can handle high degree of concurrent update operations
- **Strengths: Support for generalized data structures, High concurrent update performance, Direct connection to memory and storage subsystems, Large memory and disk capacities (in TBs)**
- **Weakness: Low main memory bandwidth, low computation capacity and degree of parallelism**

- GPUs have much smaller device memory (currently, 6 GB max) and need to pass through PCI link to access the host memory
- GPUs need to go via the PCI-E link, and the CPU host to access persistent storage subsystem
- Thousands of low frequency (730 MHz) cores, SIMT execution. Very limited SIMD support (warp-level)
- High device-memory bandwidth (300+ GB/s, as high as 700 GB/s)
- High FLOP performance for single- and double-precision performance
- Fast single-precision atomic operations, slow double-precision atomic operations
- Data-structures need to be array-based. No support for pointer-based traversals.
- Irregular memory accesses (in particular, writes) slower. Performance degrades when thread count increases.
- **Strengths: High internal memory bandwidth, compute capability, asynchronous execution**
- **Weaknesses: Access via PCI link, limited memory capability, Low concurrent update performance, constraints on data structures**

Future Directions/Open Questions

- What are the emerging workloads?
  - Internet of Things (IoT), BlockChain, Analytics,..
- What are the novel and distributive deployment models?
  - Cloud, mobile, and appliances
- What are new constraints
  - Power, cost, and real-time processing
- What are the untapped architecture opportunities?
  - NVRAM, active memory (e.g., Micron Automata Processor), active networking

- How to achieve line-rate data processing (what level of parallelism can be attained)?
- How to overcome the hardware inflexibility and development cost challenges?
- How/Where to place hardware accelerators in query execution pipelines in practice?
- What are the power and energy consumption benefits of hardware acceleration?

Thank You!
QA