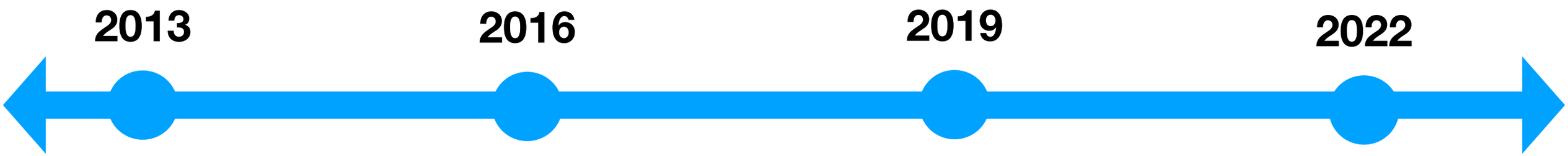
SIGCOMM 2022
Amsterdam

Programmable Multi-Dimensional Table Filters for Line Rate Network Functions

Vishal Shrivastav

PURDUE UNIVERSITY

Evolution of Programmable Data Plane Hardware*



PIFO
[SIGCOMM'16]
Push-In-First-Out Queue

PIEO
[SIGCOMM'19]
Push-In-Extract-Out Queue

Thanos
[SIGCOMM'22]
Multi-Dimensional Table Filtering

* Not an exhaustive list

Data Plane Filtering is Prevalent

Attributes

Paths

	Delay	Utilization

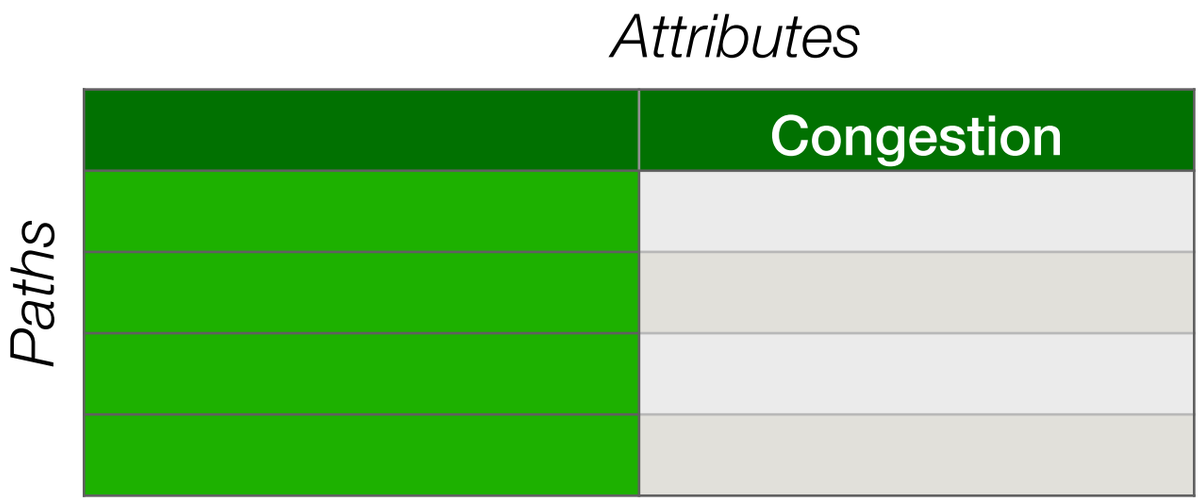
Performance-aware Routing

Filter paths with delay < d and utilization < u.

Choose a random path from the filtered set

Multi-dimensional Filtering

Data Plane Filtering is Prevalent

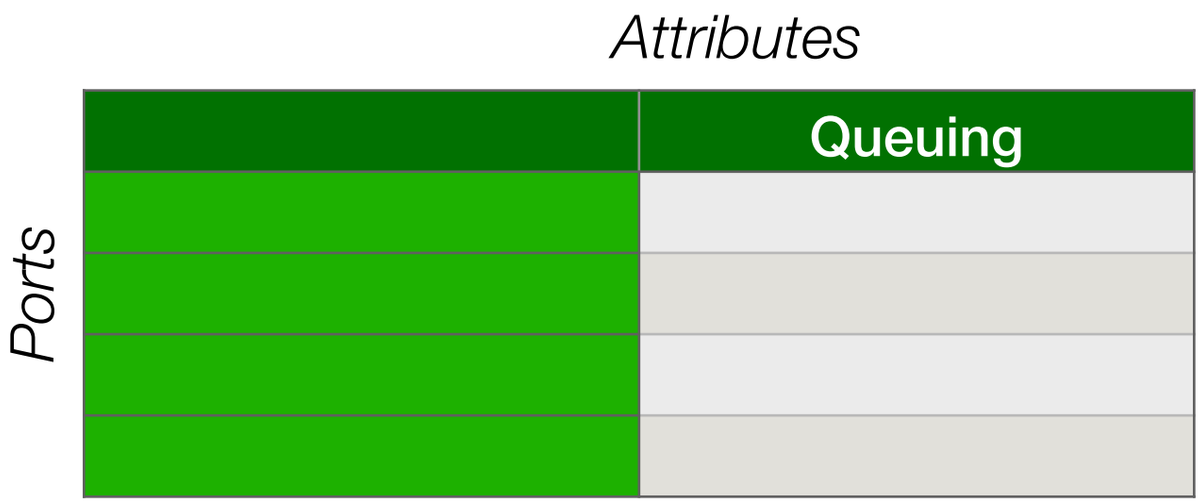


Performance-aware Routing
Filter paths with delay < d and utilization < u.
Choose a random path from the filtered set

Congestion-aware Load Balancing
Filter path with minimum congestion

Multi-dimensional Filtering

Data Plane Filtering is Prevalent



Performance-aware Routing

Filter paths with delay < d and utilization < u.

Choose a random path from the filtered set

Congestion-aware Load Balancing

Filter path with minimum congestion

or

Filter d random egress ports. Choose the least queued port port from those d ports

Multi-dimensional Filtering

Data Plane Filtering is Prevalent

Attributes

Servers

	mem	bw	cpu

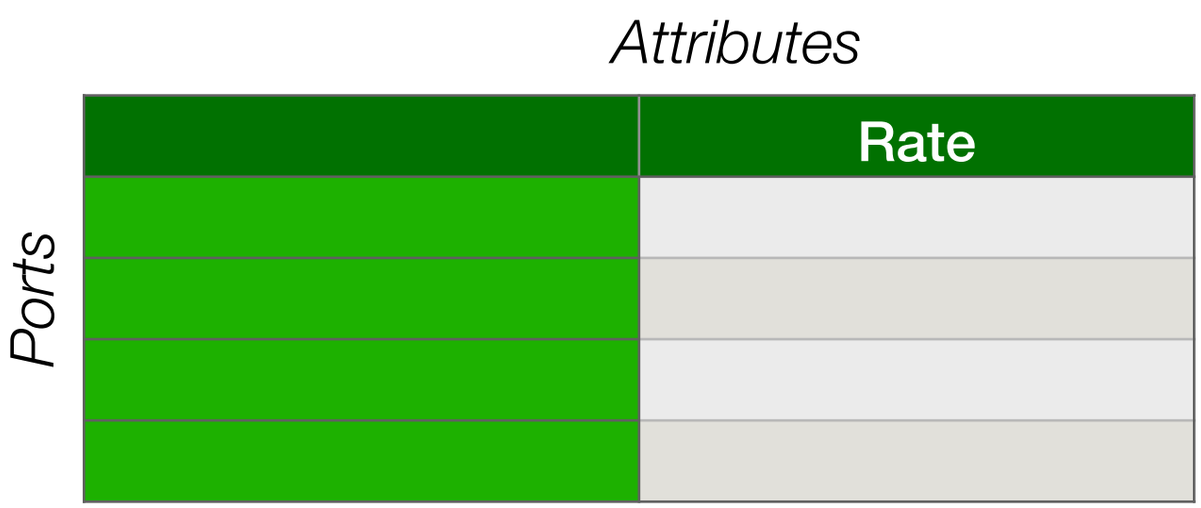
Performance-aware Routing
Filter paths with delay < d and utilization < u.
Choose a random path from the filtered set

Congestion-aware Load Balancing
Filter path with minimum congestion
or
Filter d random egress ports. Choose the least queued port port from those d ports

Resource-aware L4 Load Balancing
Filter servers with avail mem > m and avail bw > b. From the filtered set, choose server with least cpu utilization

Multi-dimensional Filtering

Data Plane Filtering is Prevalent



Performance-aware Routing
Filter paths with delay < d and utilization < u.
Choose a random path from the filtered set

Congestion-aware Load Balancing
Filter path with minimum congestion
or
Filter d random egress ports. Choose the least queued port port from those d ports

Resource-aware L4 Load Balancing
Filter servers with avail mem > m and avail bw > b. From the filtered set, choose server with least cpu utilization

Data Plane Diagnosis
Filter switch ports with packet rate > t

Multi-dimensional Filtering

Data Plane Filtering is Prevalent

Attributes

Paths

	A (0/1)	B (0/1)

Policy Compliance
From all available paths, filter the paths not carrying tenant A's or B's traffic. Choose a path at random from the filtered paths to route a new flow from tenant C.

Performance-aware Routing
Filter paths with delay < d and utilization < u. Choose a random path from the filtered set

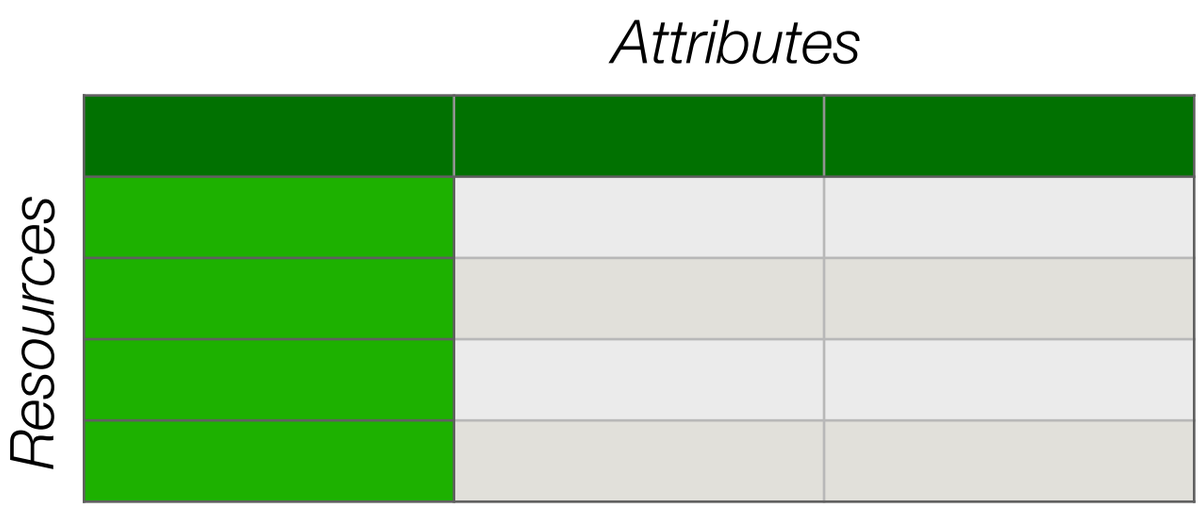
Congestion-aware Load Balancing
Filter path with minimum congestion
or
Filter d random egress ports. Choose the least queued port port from those d ports

Resource-aware L4 Load Balancing
Filter servers with avail mem > m and avail bw > b. From the filtered set, choose server with least cpu utilization

Data Plane Diagnosis
Filter switch ports with packet rate > t

Multi-dimensional Filtering

Data Plane Filtering is Prevalent



Performance-aware Routing
Filter paths with delay < d and utilization < u.
Choose a random path from the filtered set

Congestion-aware Load Balancing
Filter path with minimum congestion
or
Filter d random egress ports. Choose the least queued port port from those d ports

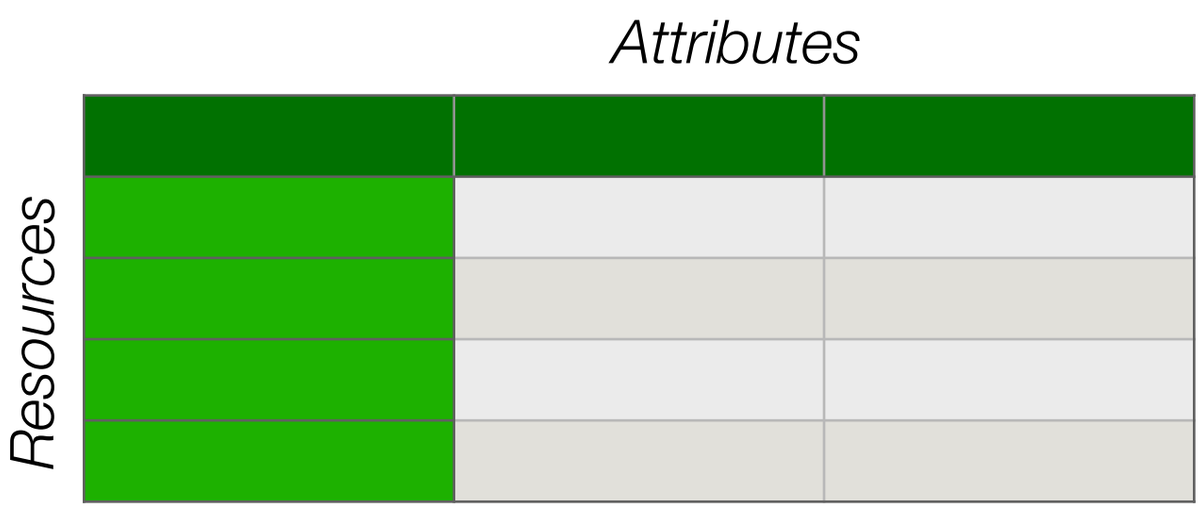
Policy Compliance
From all available paths, filter the paths not carrying tenant A's or B's traffic. Choose a path at random from the filtered paths to route a new flow from tenant C.

Resource-aware L4 Load Balancing
Filter servers with avail mem > m and avail bw > b. From the filtered set, choose server with least cpu utilization

Data Plane Diagnosis
Filter switch ports with packet rate > t

Chained Multi-dimensional Filtering

Data Plane Filtering is Prevalent



Performance-aware Routing
Filter paths with delay < d and utilization < u.
Choose a random path from the filtered set

Congestion-aware Load Balancing
Filter path with minimum congestion
or
Filter d random egress ports. Choose the least queued port port from those d ports

Policy Compliance
From all available paths, filter the paths not carrying tenant A's or B's traffic.
Choose a path at random from the filtered paths to route a new flow from tenant C.

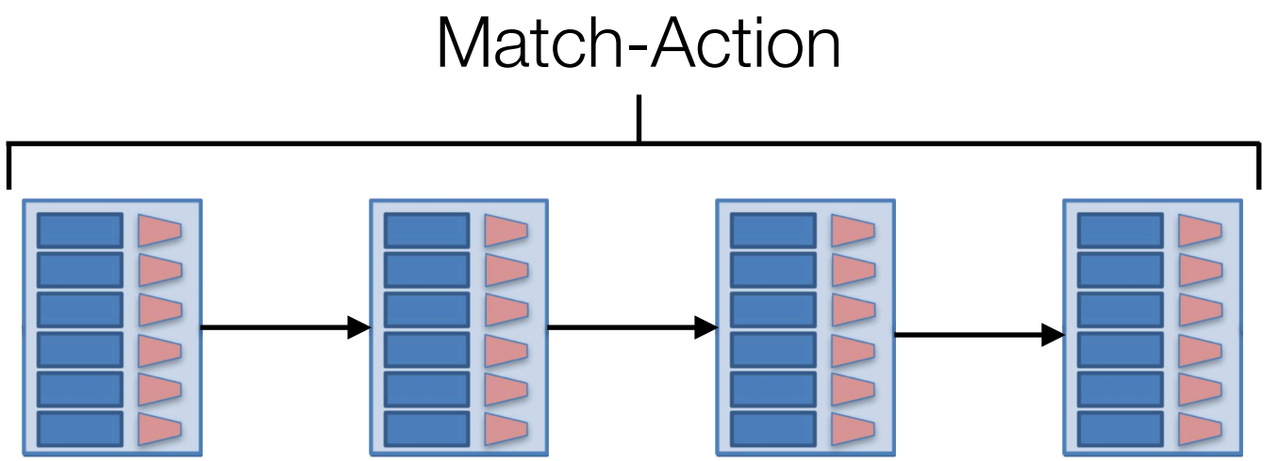
Resource-aware L4 Load Balancing
Filter servers with avail mem > m and avail bw > b. From the filtered set, choose server with least cpu utilization

Data Plane Diagnosis
Filter switch ports with packet rate > t

Chained Multi-dimensional Filtering at Line Rate

State-of-the-Art

Current Programmable Switch Processing Pipeline

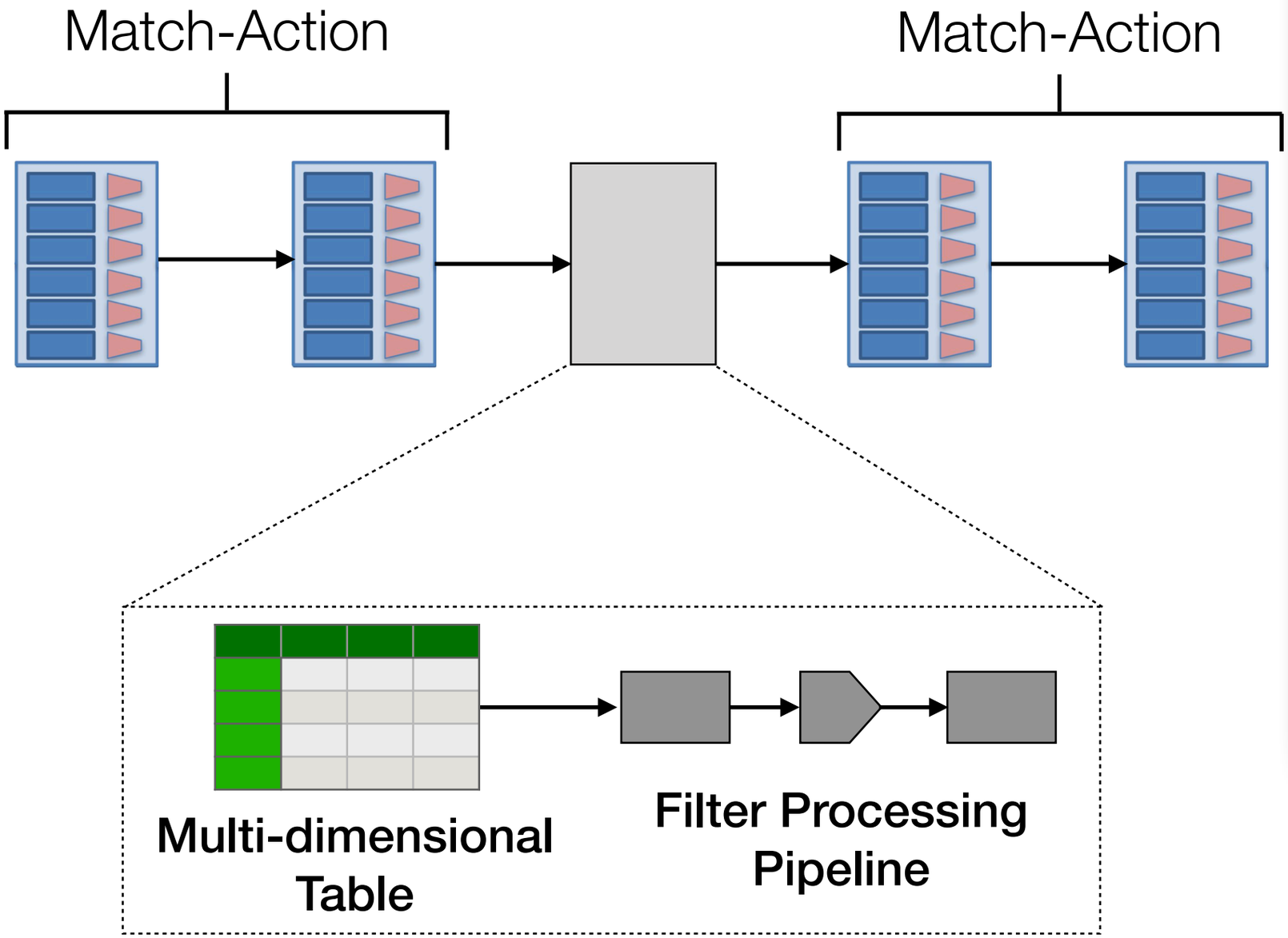


Does not support line rate multi-dimensional filtering

Thanos

Thanos

Thanos Switch Processing Pipeline



Programmable Filter Module

Filter Abstractions and Primitives

Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

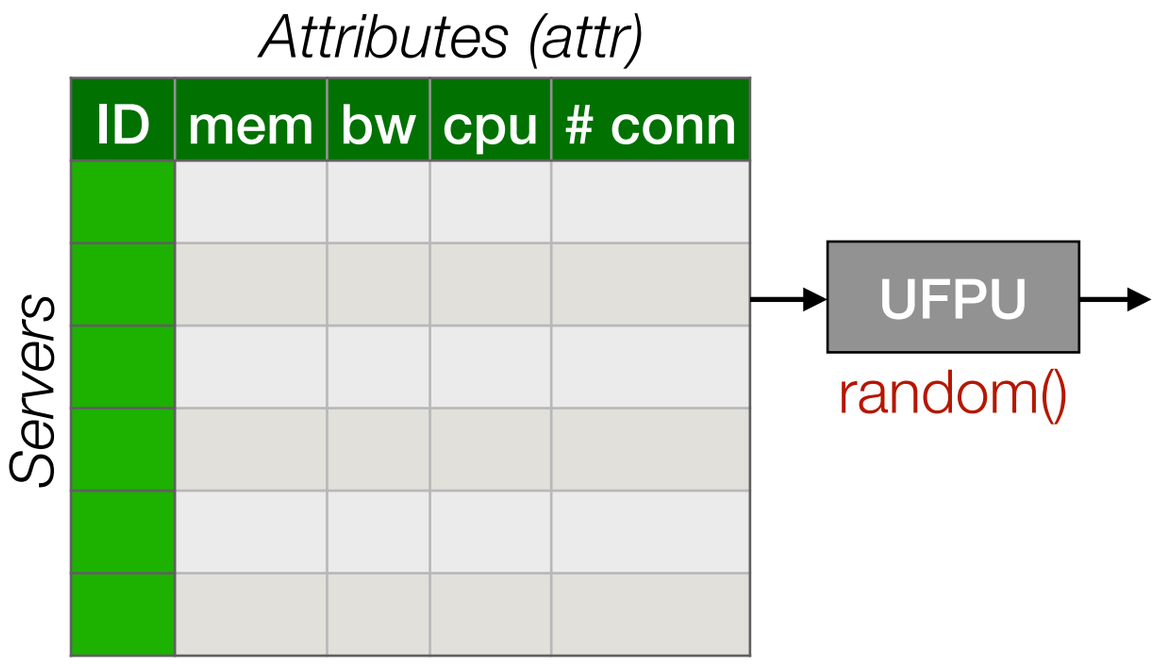
UFPU: table $\mapsto$ table

Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\mapsto$ table { random( )

Filter a load balancing server at random



Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

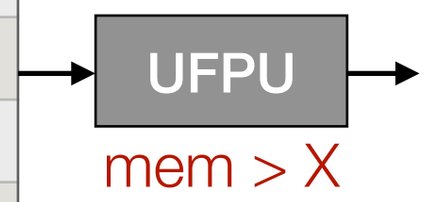
UFPU: table ↦ table {
- random( )
- predicate(attr *relop* X)

Filter load balancing servers with avail mem > X

Attributes (attr)

ID	mem	bw	cpu	# conn

Servers

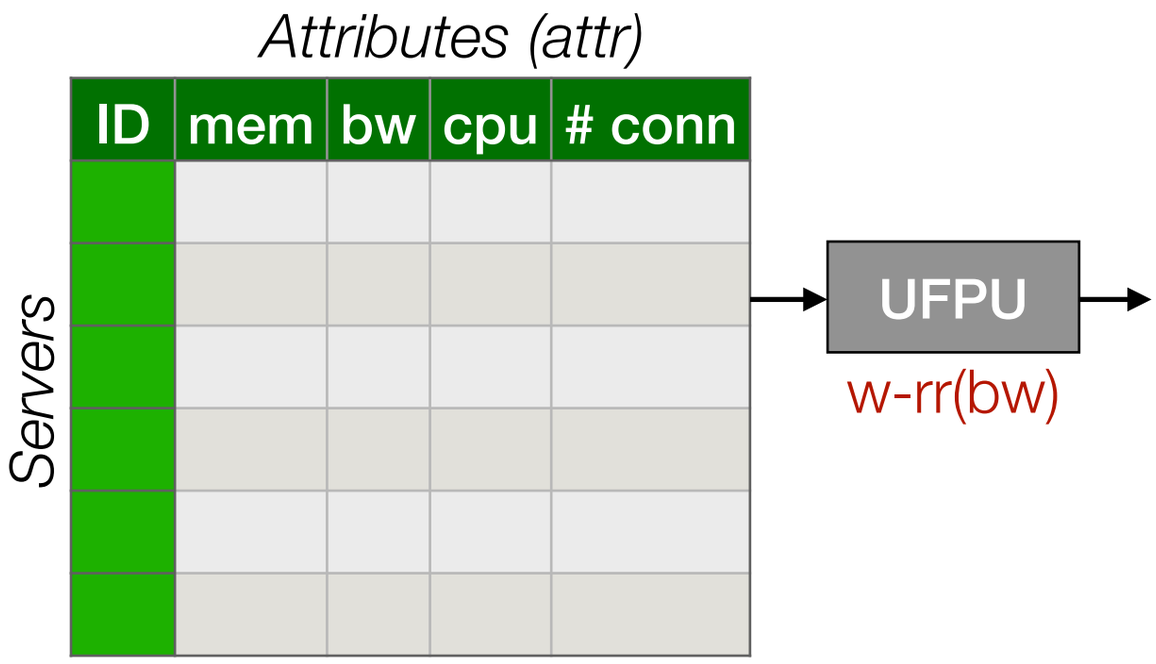


Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table ↦ table {
- random( )
- predicate(attr *relop* X)
- weighted-round-robin(attr)

Filter load balancing servers in a round robin manner weighted by avail bw

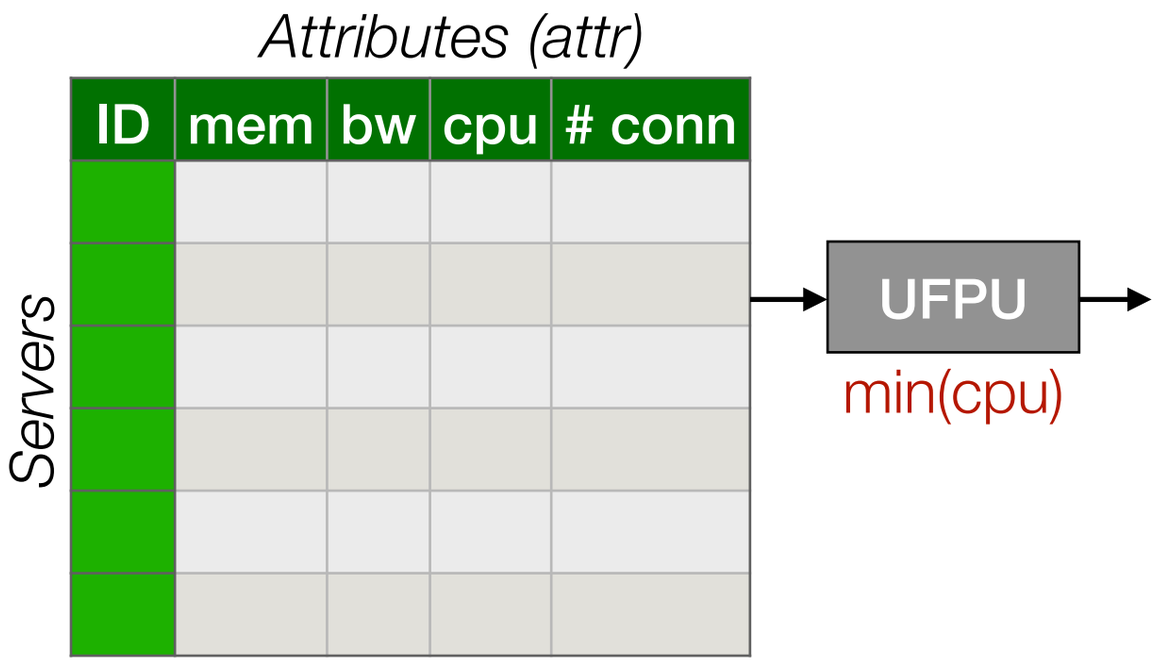


Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table ↦ table {
- random( )
- predicate(attr *relop* X)
- weighted-round-robin(attr)
- min/max(attr)

Filter load balancing server with min cpu utilization



Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\mapsto$ table { random( ) | predicate(attr *relop* X) | weighted-round-robin(attr) | min/max(attr) }

Filter N least cpu utilized load balancing servers

or

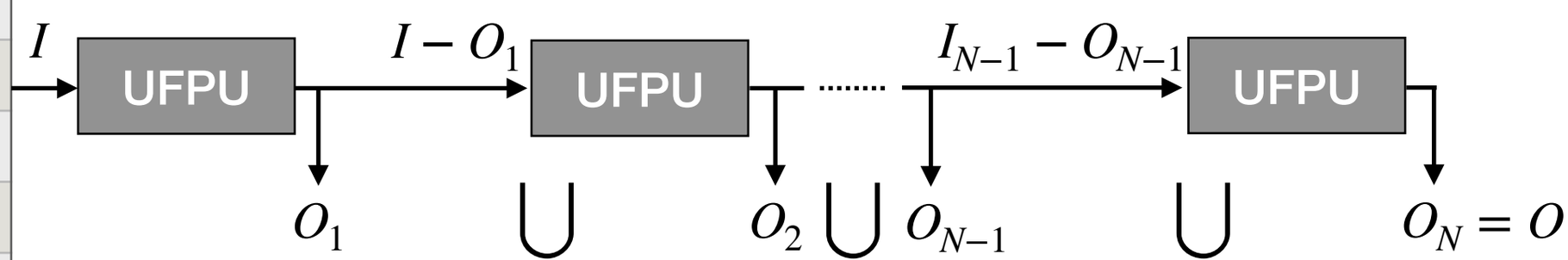
Filter N random load balancing servers

Attributes (attr)

ID	mem	bw	cpu	# conn

Servers

A chain of N UFPUs



Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table $\mapsto$ table {
- random( )
- predicate(attr *relop* X)
- weighted-round-robin(attr)
- min/max(attr)

Filter N least cpu utilized load balancing servers

or

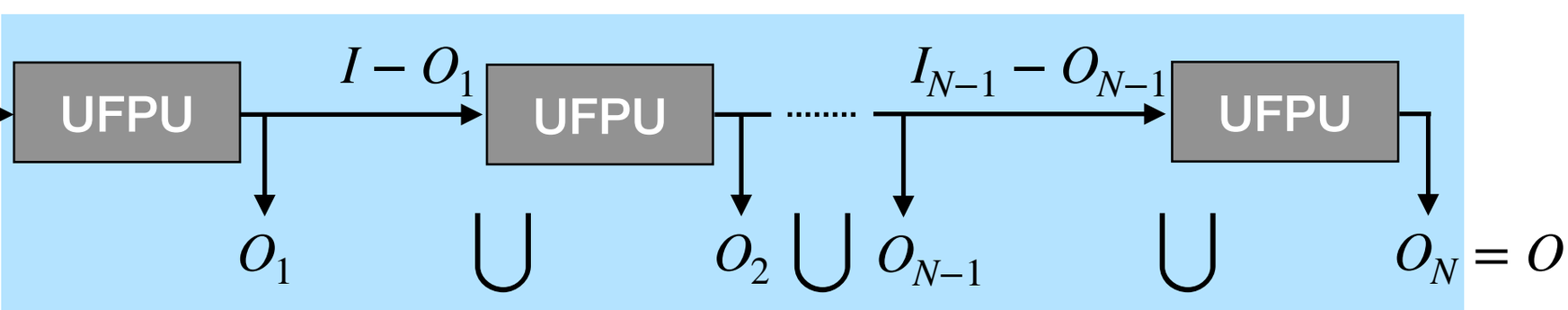
Filter N random load balancing servers

Attributes (attr)

ID	mem	bw	cpu	# conn

Servers

A chain of N UFPUs



Abstractions and Primitives

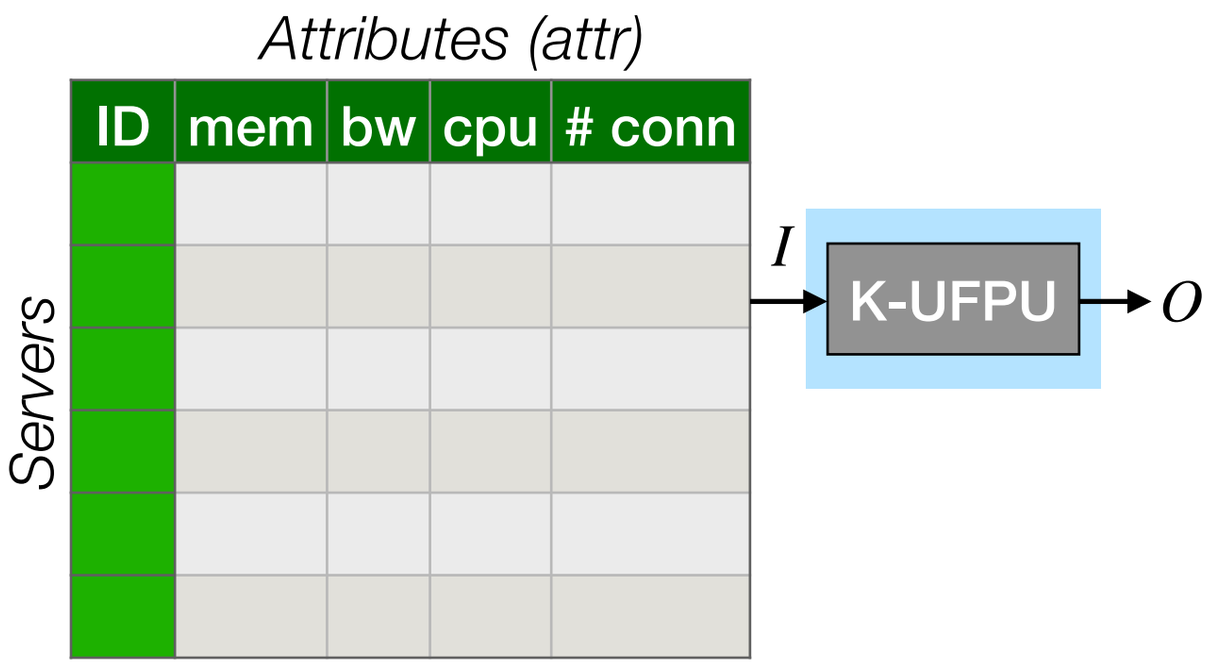
Unary Filter Processing Unit (UFPU)

UFPU: table ↦ table {
- random( )
- predicate(attr *relop* X)
- weighted-round-robin(attr)
- min/max(attr)

Filter N least cpu utilized load balancing servers

or

Filter N random load balancing servers



Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table ↦ table {
- random( )
- predicate(attr *relop* X)
- weighted-round-robin(attr)
- min/max(attr)

Filter N least cpu utilized load balancing servers

or

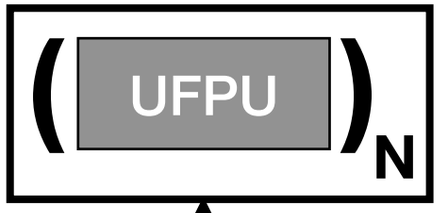
Filter N random load balancing servers

Attributes (attr)

ID	mem	bw	cpu	# conn

Servers

K=N, min(cpu)
or
K=N, random()



K-UFPU

K-UFPU comprises N UFPUs
and adds a new configurable parameter K

K specifies the length of chain (max N)
(by setting K=1, K-UFPU reduces to UFPU)

Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table ↦ table {
- random( )
- predicate(attr *relop* X)
- weighted-round-robin(attr)
- min/max(attr)

Filter N least cpu utilized load balancing servers

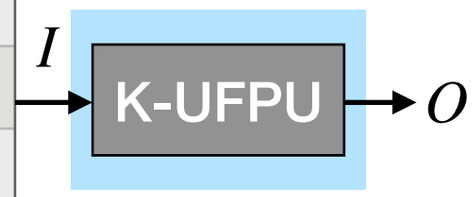
or

Filter N random load balancing servers

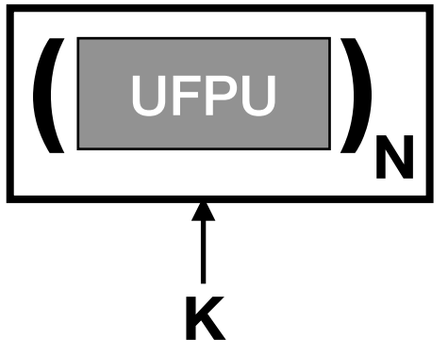
Attributes (attr)

ID	mem	bw	cpu	# conn

Servers



K=N, min(cpu)
or
K=N, random()



K-UFPU

K-UFPU comprises N UFPUs
and adds a new configurable parameter K

K specifies the length of chain (max N)
(by setting K=1, K-UFPU reduces to UFPU)

We use K-UFPU (instead of UFPU)
as the basic computing unit

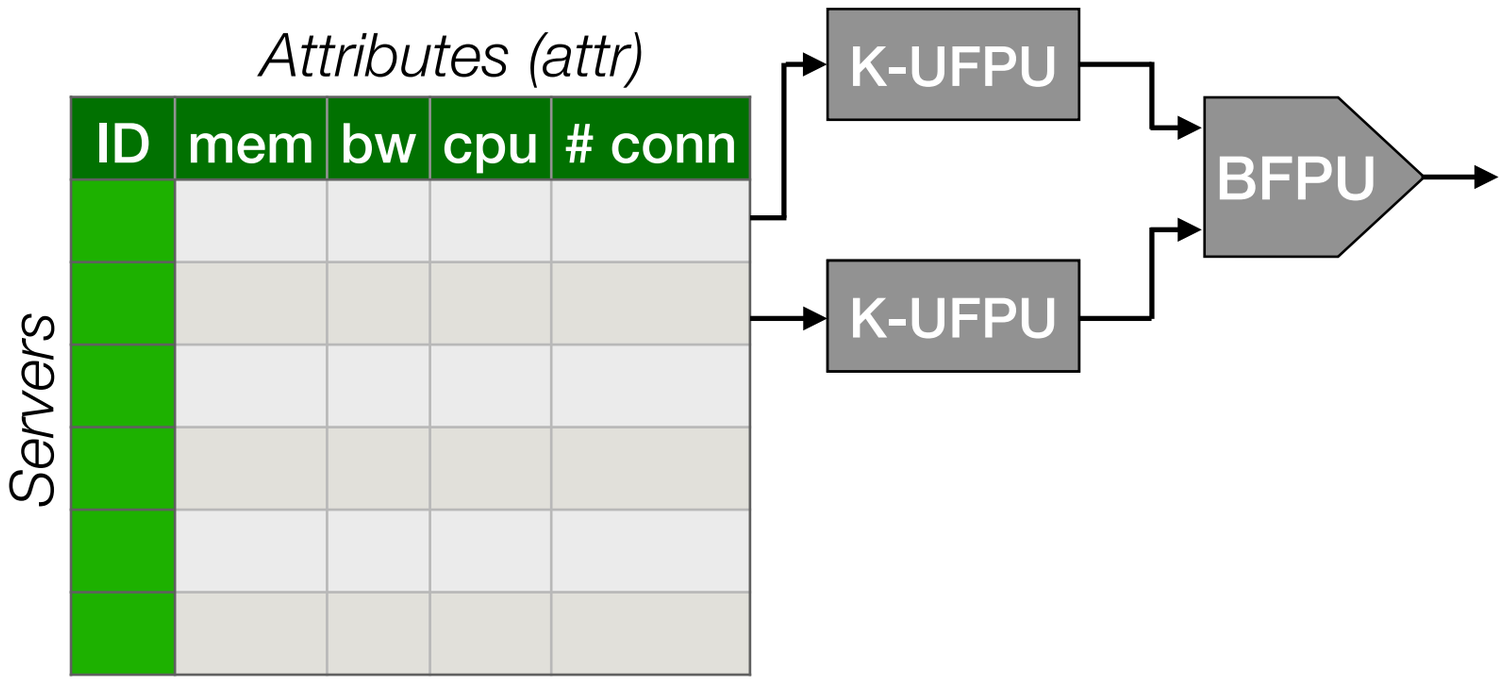
Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table ↦ table {
- random( )
- predicate(attr *relop* X)
- weighted-round-robin(attr)
- min/max(attr)

Binary Filter Processing Unit (BFPU)

BFPU: table, table ↦ table {
- union( )
- intersection( )
- difference( )



Abstractions and Primitives

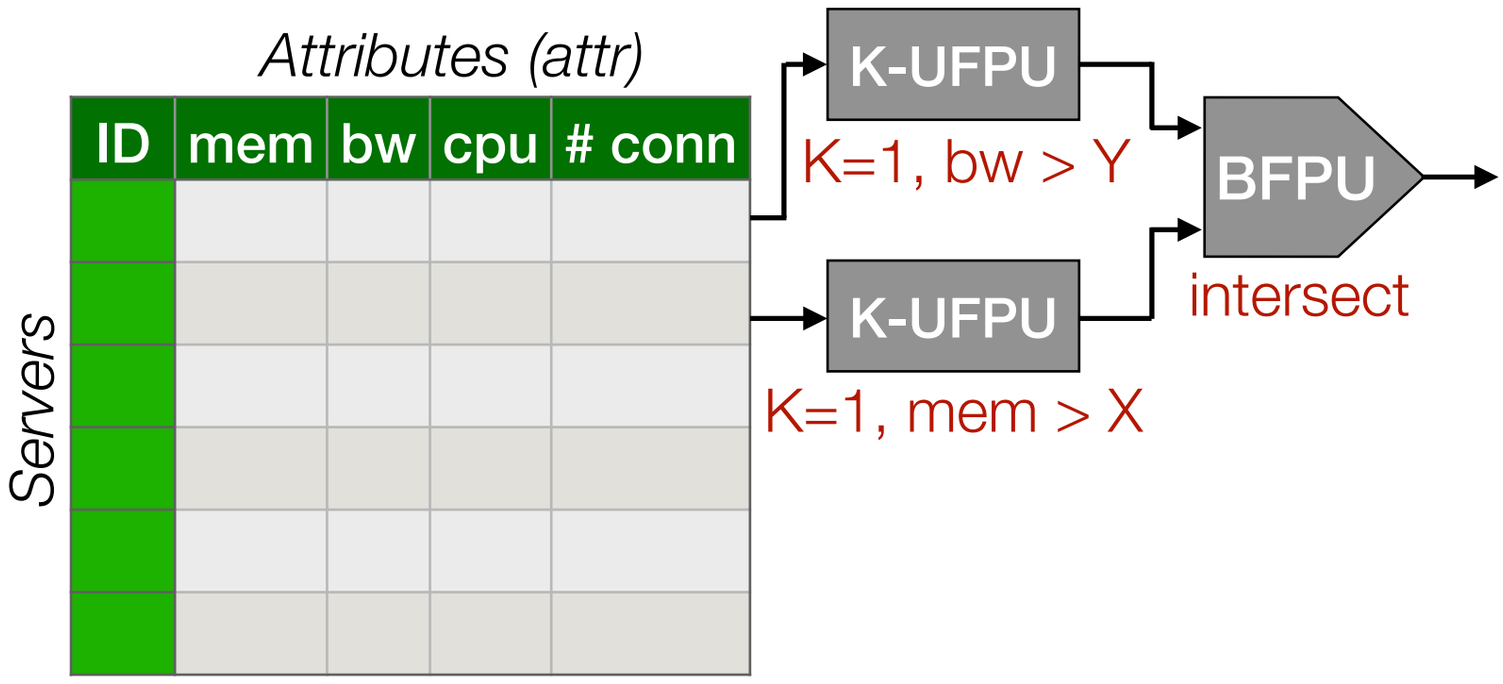
Unary Filter Processing Unit (UFPU)

UFPU: table ↦ table {
- random( )
- predicate(attr *relop* X)
- weighted-round-robin(attr)
- min/max(attr)

Binary Filter Processing Unit (BFPU)

BFPU: table, table ↦ table {
- union( )
- intersection( )
- difference( )

Filter all load balancing servers with avail mem > X and avail bw > Y



Abstractions and Primitives

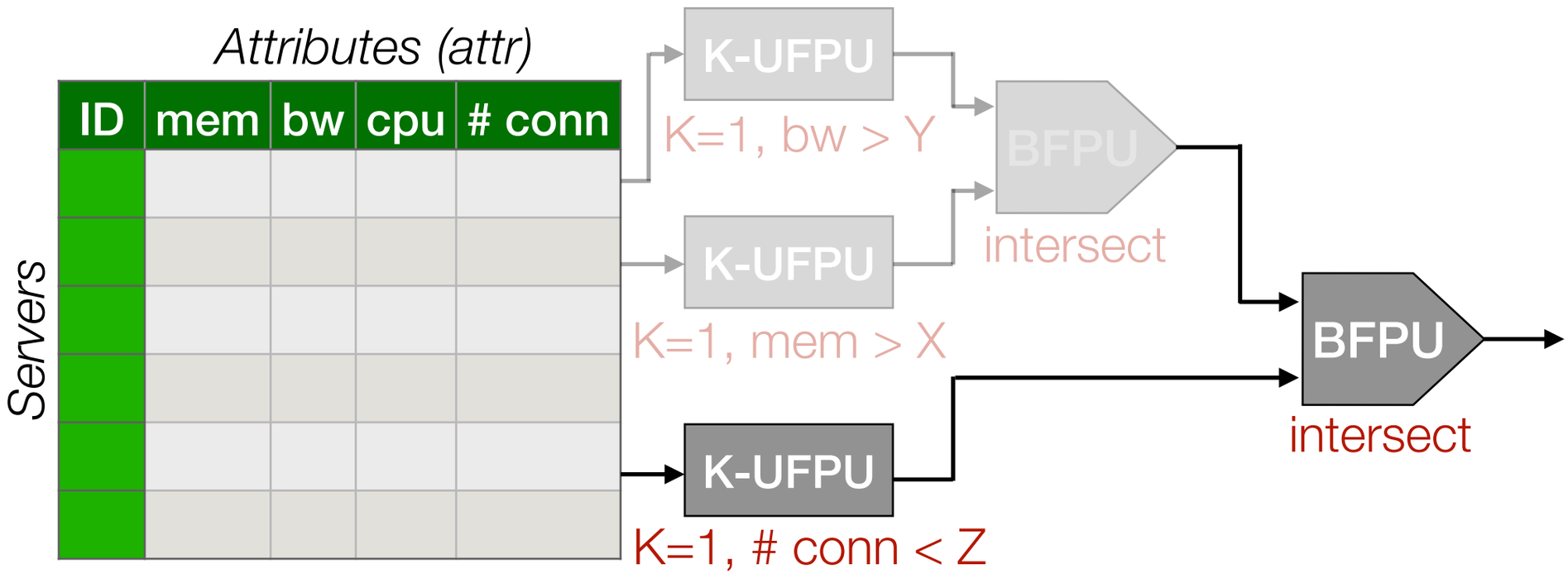
Unary Filter Processing Unit (UFPU)

UFPU: table ↦ table
- random( )
- predicate(attr *relop* X)
- weighted-round-robin(attr)
- min/max(attr)

Binary Filter Processing Unit (BFPU)

BFPU: table, table ↦ table
- union( )
- intersection( )
- difference( )

Filter all load balancing servers with avail mem > X and avail bw > Y and # conn < Z



Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

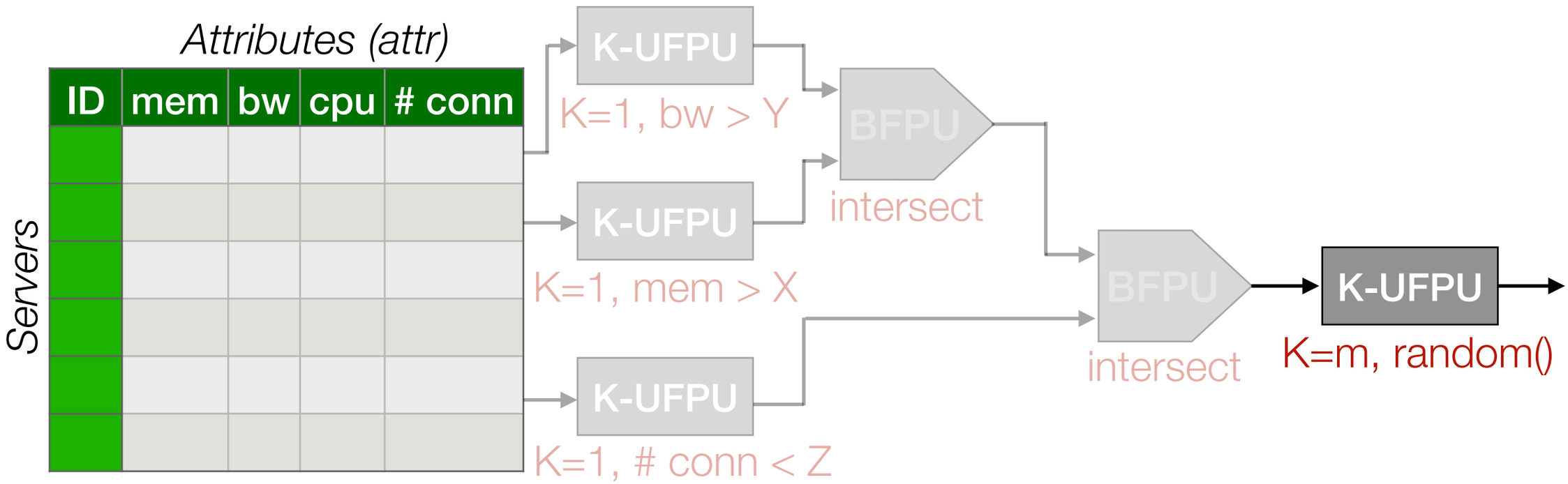
UFPU: table ↦ table { random( ) | predicate(attr *relop* X) | weighted-round-robin(attr) | min/max(attr)

Binary Filter Processing Unit (BFPU)

BFPU: table, table ↦ table { union( ) | intersection( ) | difference( )

Filter all load balancing servers with avail mem > X and avail bw > Y and # conn < Z

Filter m load balancing servers at random



Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

UFPU: table ↦ table
- random( )
- predicate(attr *relop* X)
- weighted-round-robin(attr)
- min/max(attr)

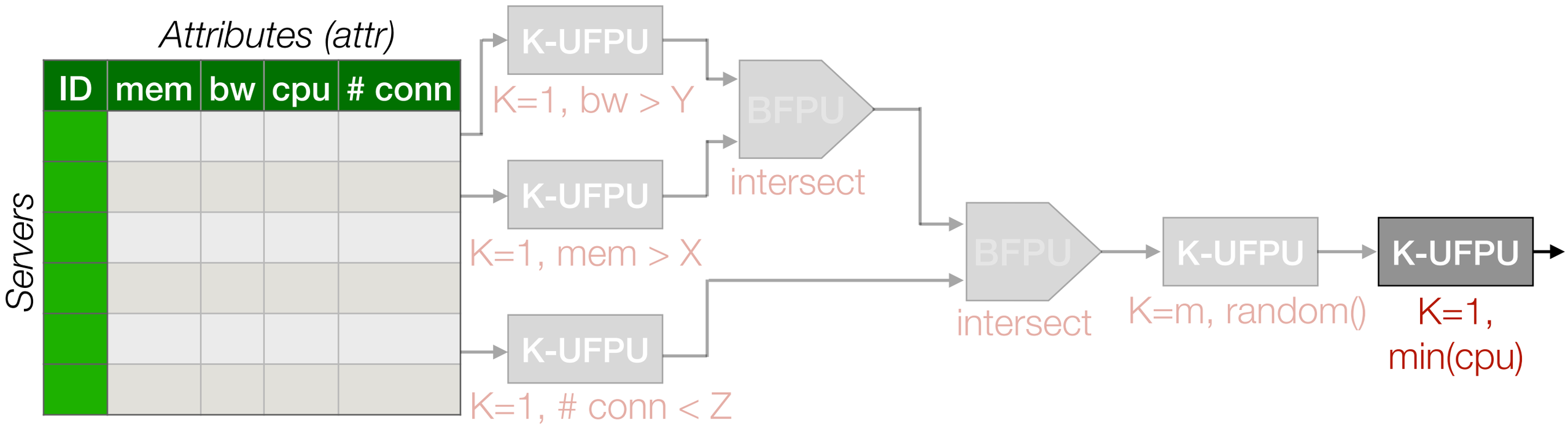
Binary Filter Processing Unit (BFPU)

BFPU: table, table ↦ table
- union( )
- intersection( )
- difference( )

Filter all load balancing servers with avail mem > X and avail bw > Y and # conn < Z

Filter m load balancing servers at random:

Filter load balancing server with min cpu util



Abstractions and Primitives

Unary Filter Processing Unit (UFPU)

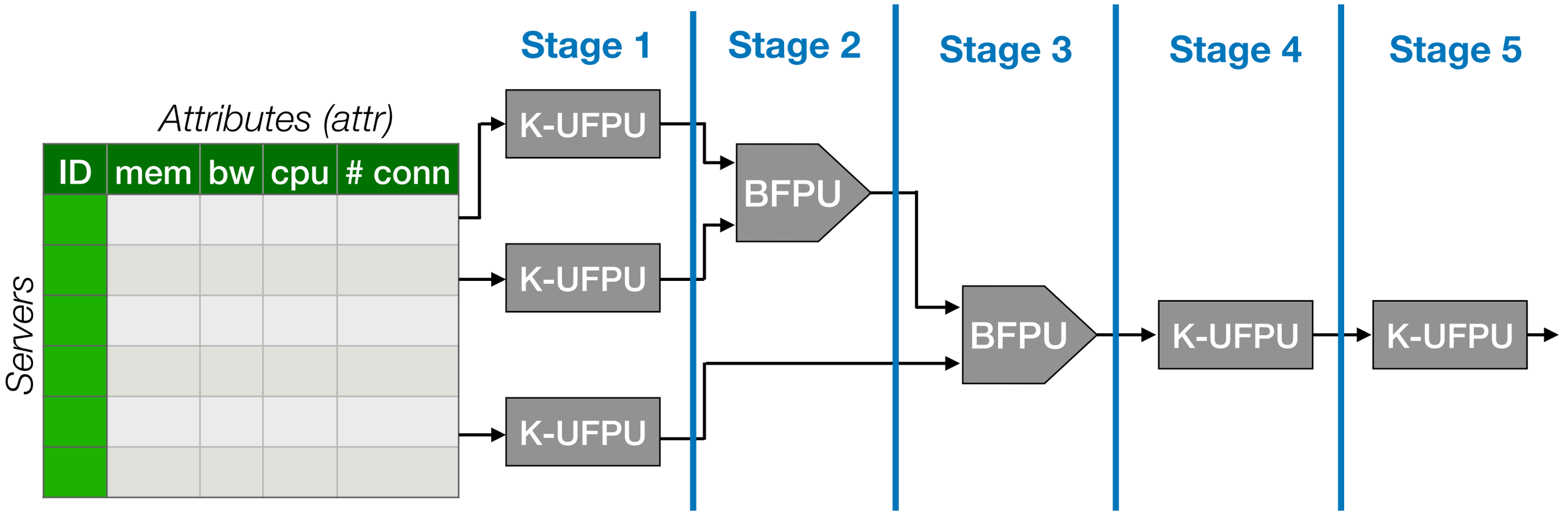
UFPU: table ↦ table {
- random( )
- predicate(attr *relop* X)
- weighted-round-robin(attr)
- min/max(attr)

Binary Filter Processing Unit (BFPU)

BFPU: table, table ↦ table {
- union( )
- intersection( )
- difference( )

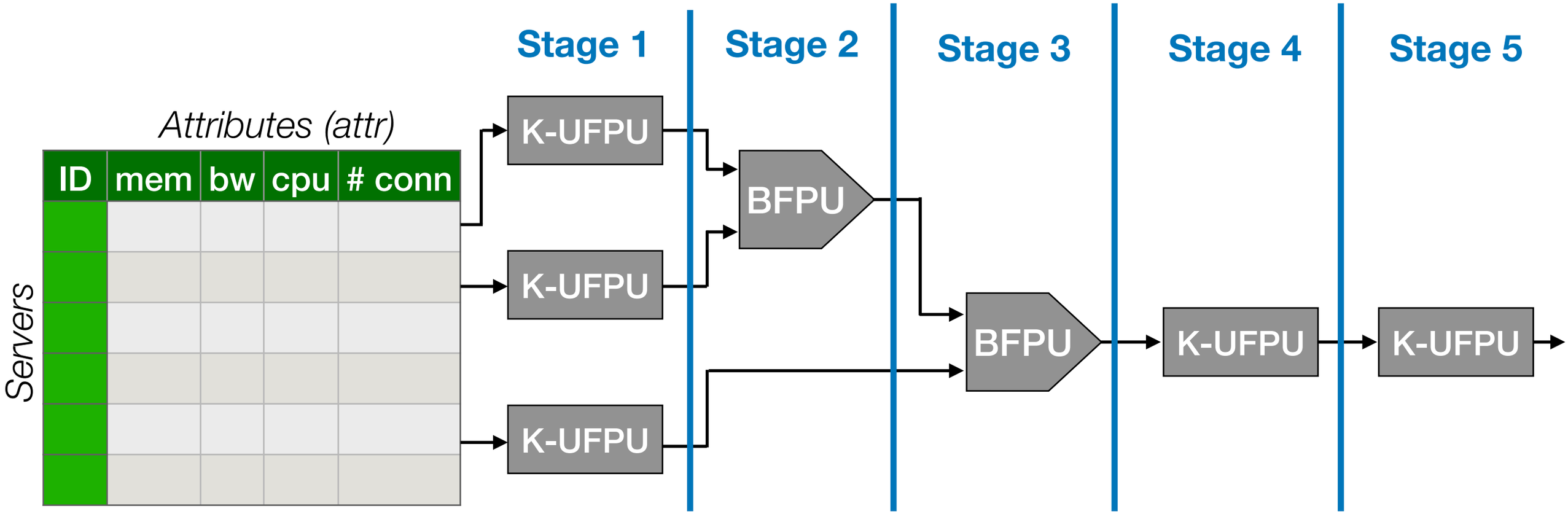
A 5-stage serial chain filter pipeline

(outputs from stage i are inputs to stage i+1)

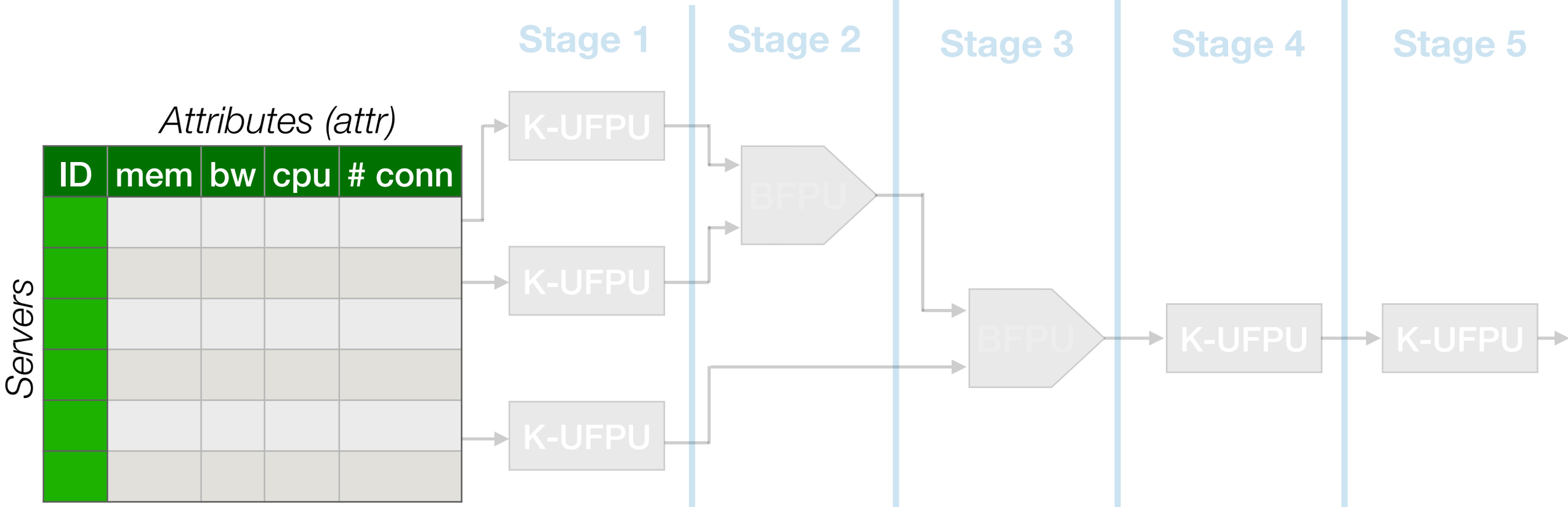


From Abstractions to Hardware Design

Two Hardware Components

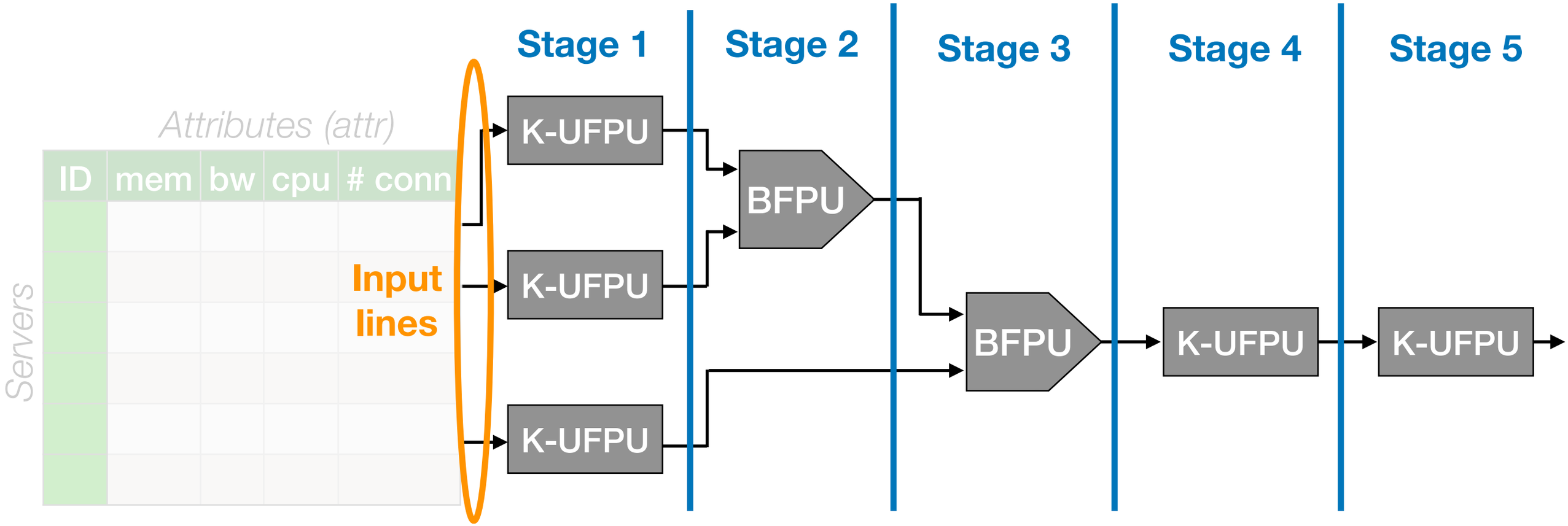


Hardware Component # 1



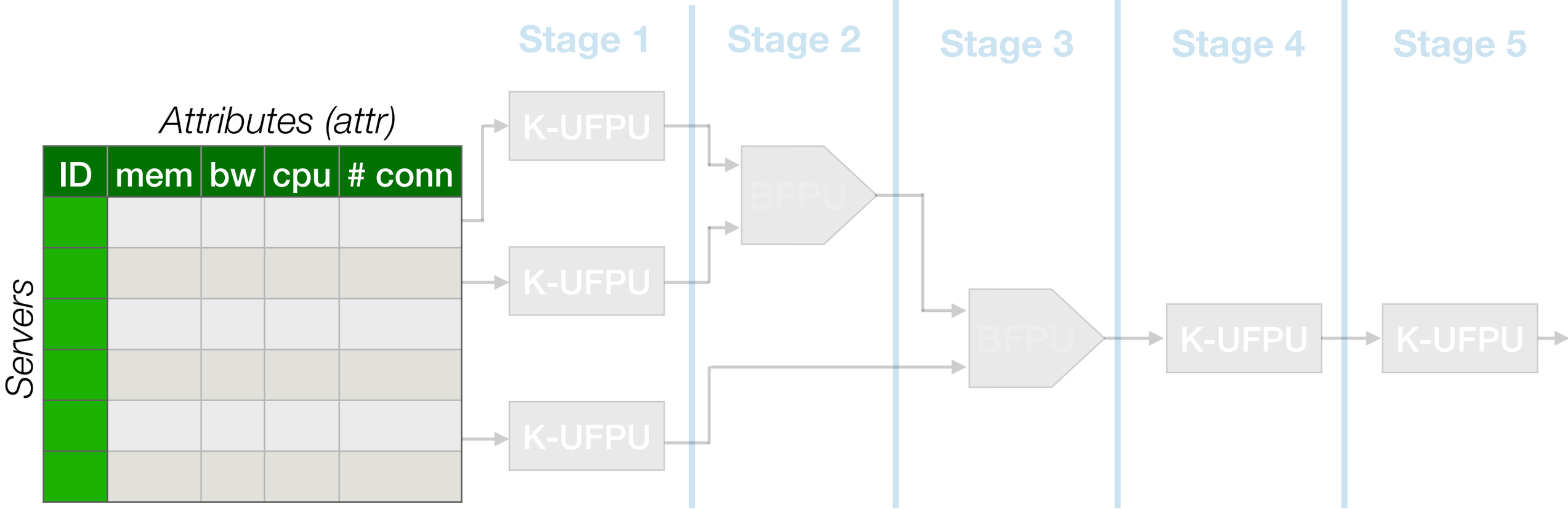
1. Multi-dimensional Table

Hardware Component # 2



2. Programmable Filter Pipeline

Multi-Dimensional Table



How to design an efficient data structure for a multi-dimensional relational table?

Should allow line rate read, write, update!

Multi-Dimensional Table Data Structure

Limitations of Classic Data Structures

No universal data structure

- Range trees / B-Trees for range filtering
- Heap for min/max filtering
- Disjoint Set Data Structure for set operations
- Either compromise on performance of certain operations or pay the cost of maintaining multiple data structures over the same data

Hierarchical Structure

- Fundamental O(log(N)) latency
- Hard to pipeline [1]

[1] Yi-Hua E. Yang and Viktor K. Prasanna. “High Throughput and Large Capacity Pipelined Dynamic Search Tree on FPGA”. In Proceedings of FPGA, 2010

Multi-Dimensional Table Data Structure

Our Solution:

Sorted Multidimensional Bidirectional Map (SMBM)

Multi-Dimensional Table Data Structure

Our Solution:

Sorted Multidimensional Bidirectional Map (SMBM)

ID	X	Y
1	15	6
2	4	19
3	22	19
4	15	8

Example Table

SMBM Property # 1

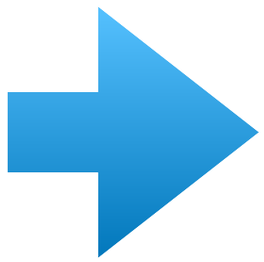
Our Solution:

Sorted Multidimensional Bidirectional Map (SMBM)

ID	X	Y
1	15	6
2	4	19
3	22	19
4	15	8

Example Table

Stored as



ID	X	Y
1	4	6
2	15	8
3	15	19
4	22	19

Store each dimension as *flat* list of flip-flops

Allows parallel access and processing

SMBM Property # 2

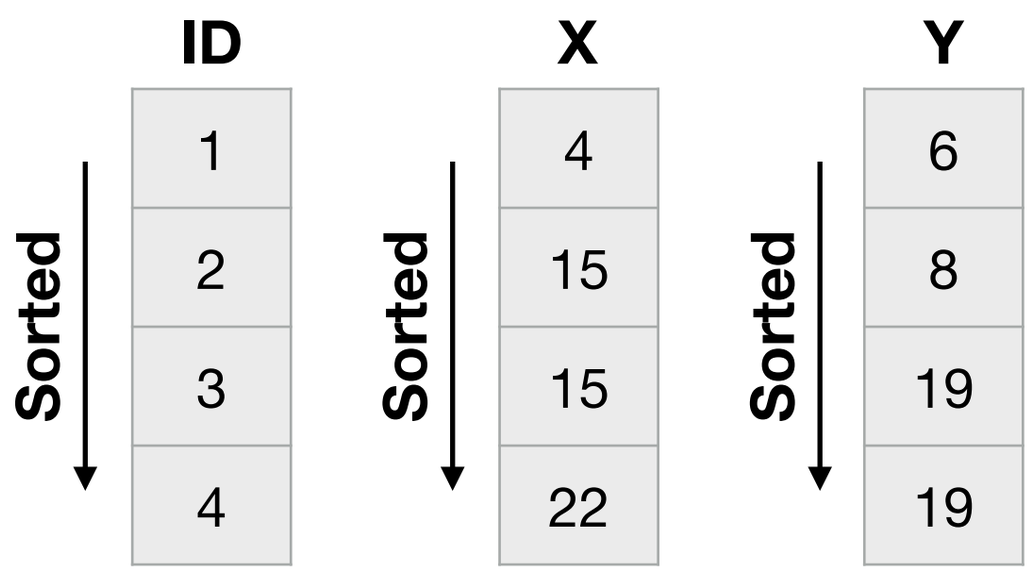
Our Solution:

Sorted Multidimensional Bidirectional Map (SMBM)

ID	X	Y
1	15	6
2	4	19
3	22	19
4	15	8

Example Table

Stored as



Each list is kept sorted

Allows fast max/min filter operations

SMBM Property # 3

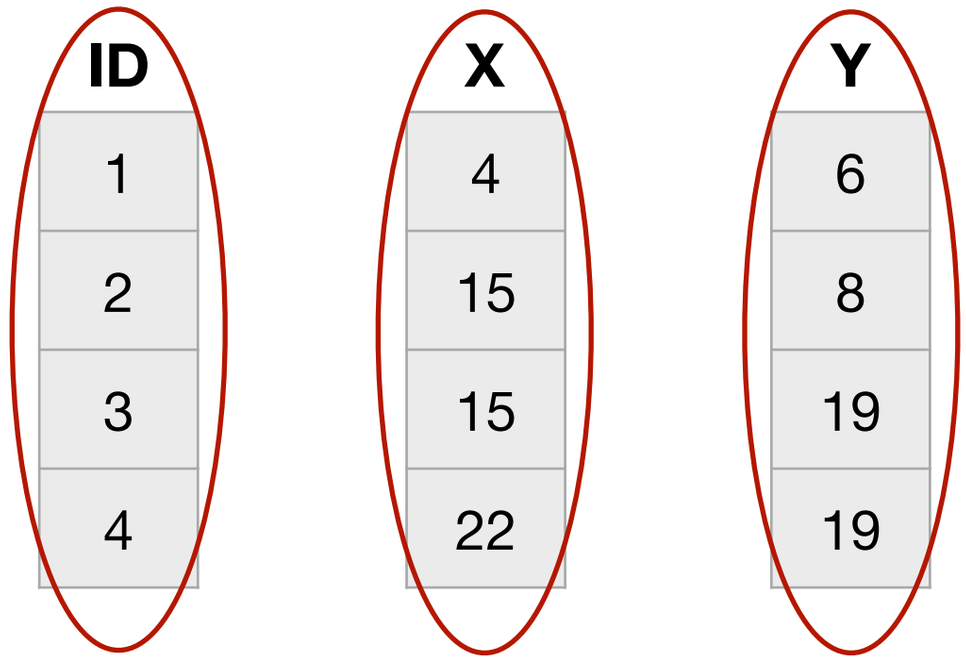
Our Solution:

Sorted Multidimensional Bidirectional Map (SMBM)

ID	X	Y
1	15	6
2	4	19
3	22	19
4	15	8

Example Table

Stored as



Each dimension is stored as an *independent* list

Allows parallel filter operations on multiple dimensions

SMBM Property # 4

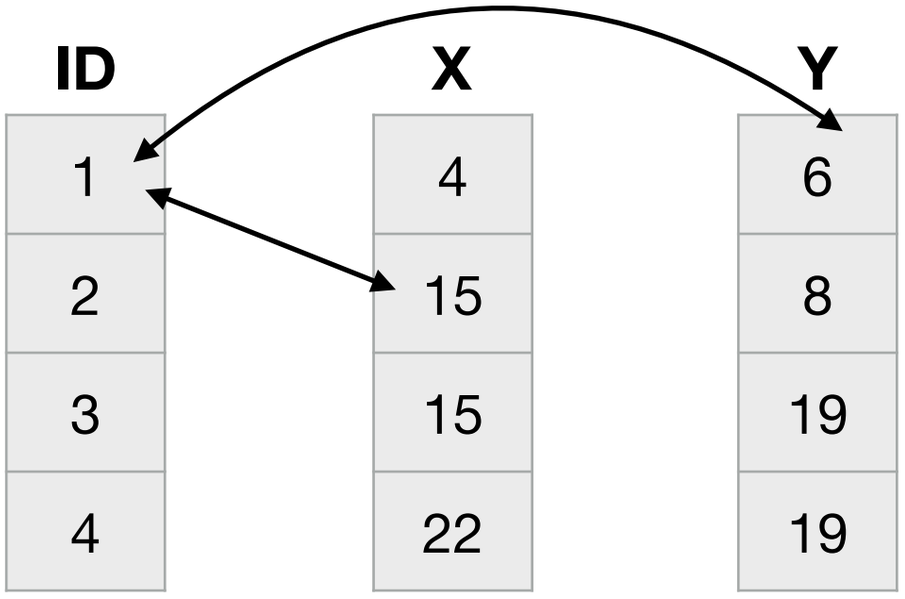
Our Solution:

Sorted Multidimensional Bidirectional Map (SMBM)

ID	X	Y
1	15	6
2	4	19
3	22	19
4	15	8

Example Table

Stored as



Bidirectional mapping between ID and attributes

SMBM Property # 4

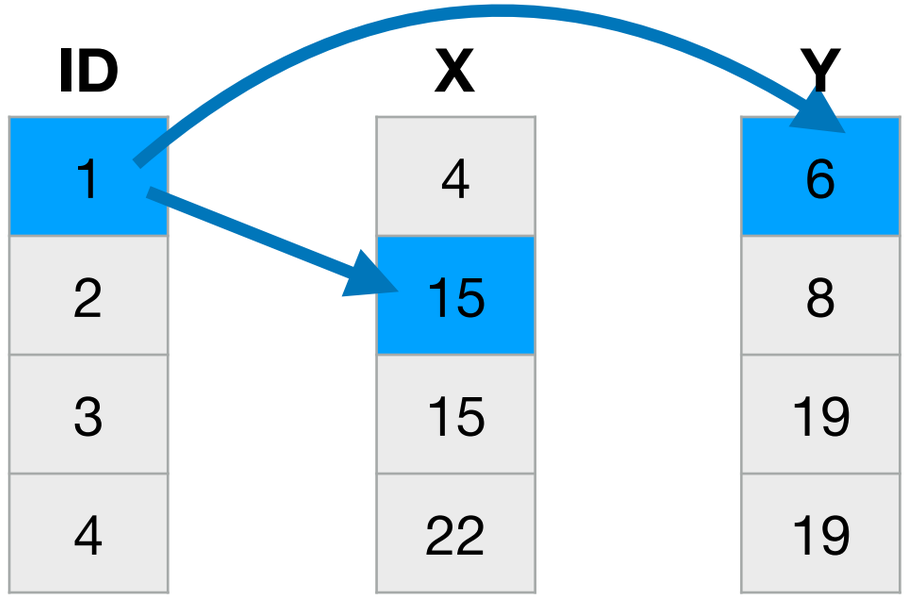
Our Solution:

Sorted Multidimensional Bidirectional Map (SMBM)

ID	X	Y
1	15	6
2	4	19
3	22	19
4	15	8

Example Table

Stored as



Bidirectional mapping between ID and attributes

Forward map keeps track of which attributes belong to which ID

SMBM Property # 4

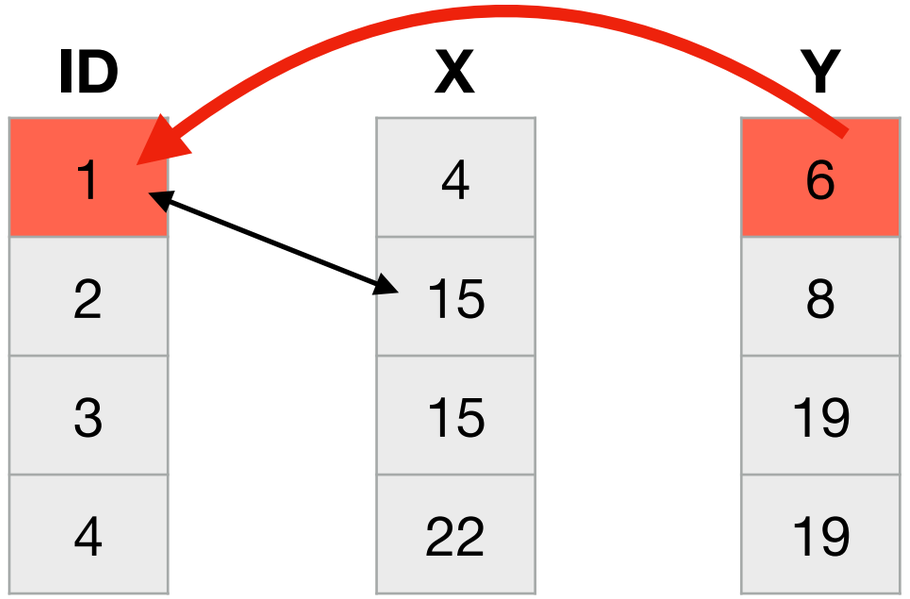
Our Solution:

Sorted Multidimensional Bidirectional Map (SMBM)

ID	X	Y
1	15	6
2	4	19
3	22	19
4	15	8

Example Table

Stored as



Bidirectional mapping between ID and attributes

Forward map keeps track of which attributes belong to which ID

Reverse map allows fast mapping of filtered attributes to their respective IDs

SMBM Performance

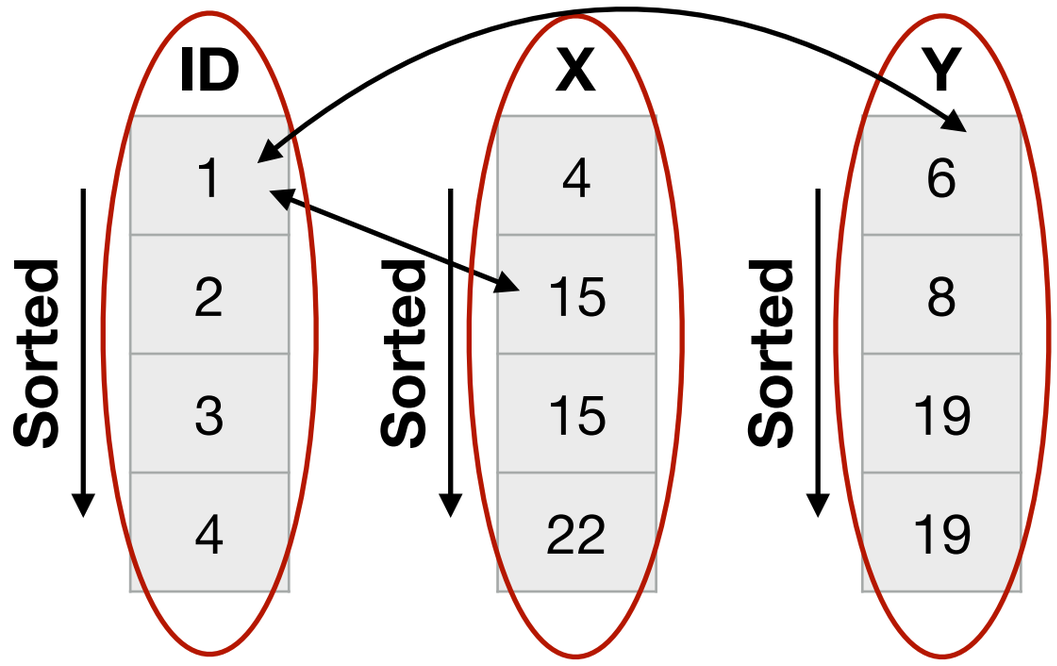
Our Solution:

Sorted Multidimensional Bidirectional Map (SMBM)

ID	X	Y
1	15	6
2	4	19
3	22	19
4	15	8

Example Table

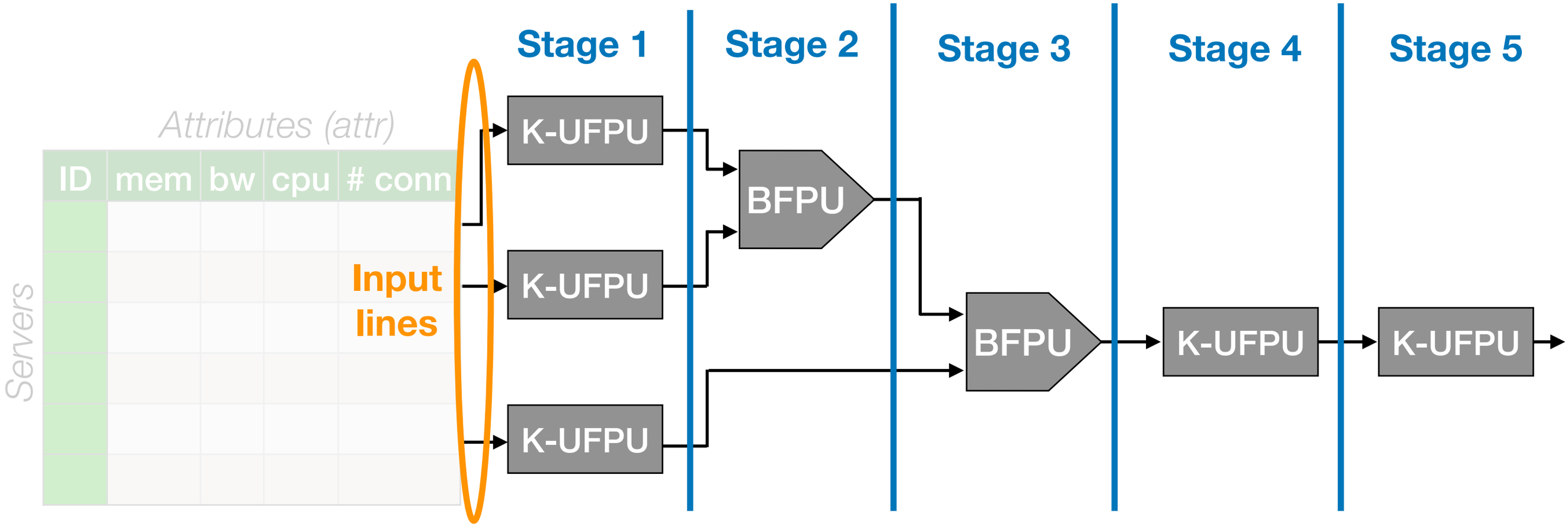
Stored as



Can read the entire data structure in parallel in 1 clock cycle

Add and Delete can be issued every clock cycle with a latency of 2 cycles

Filter Pipeline

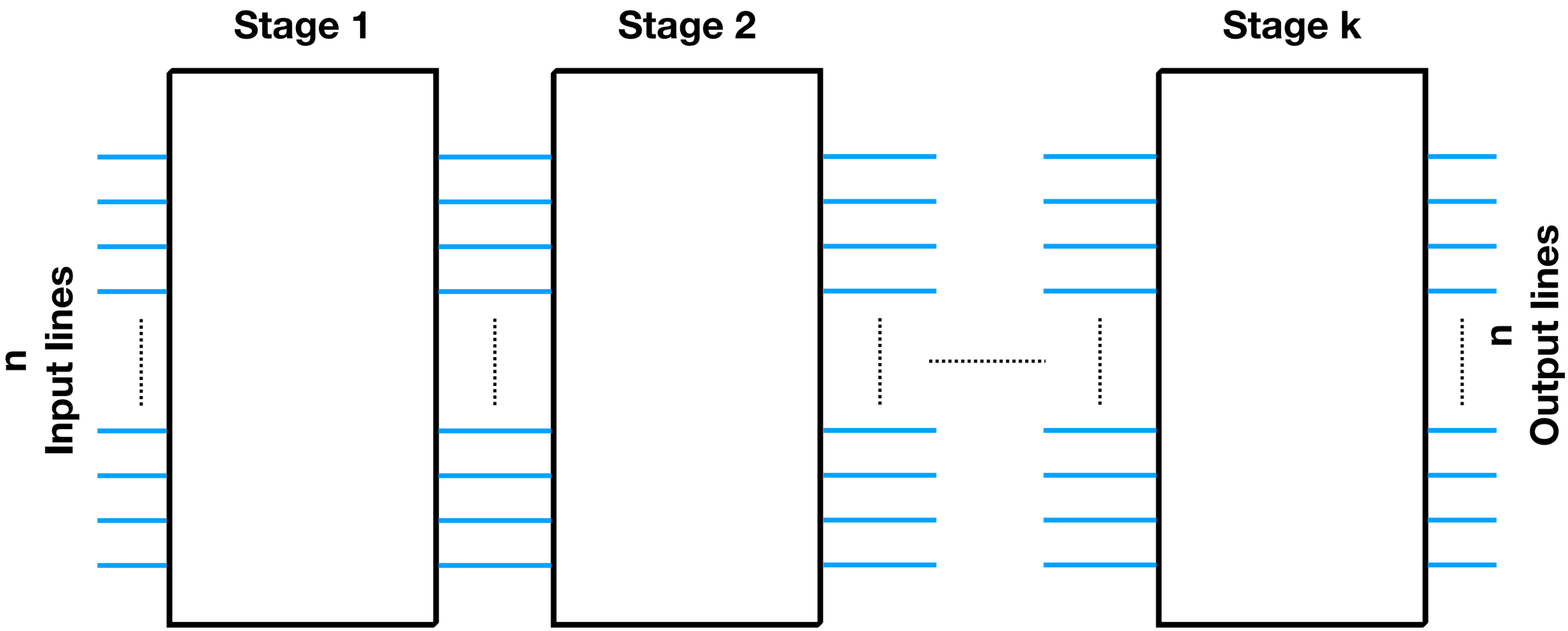


How to design a *fully reconfigurable* and *fast* filter pipeline?

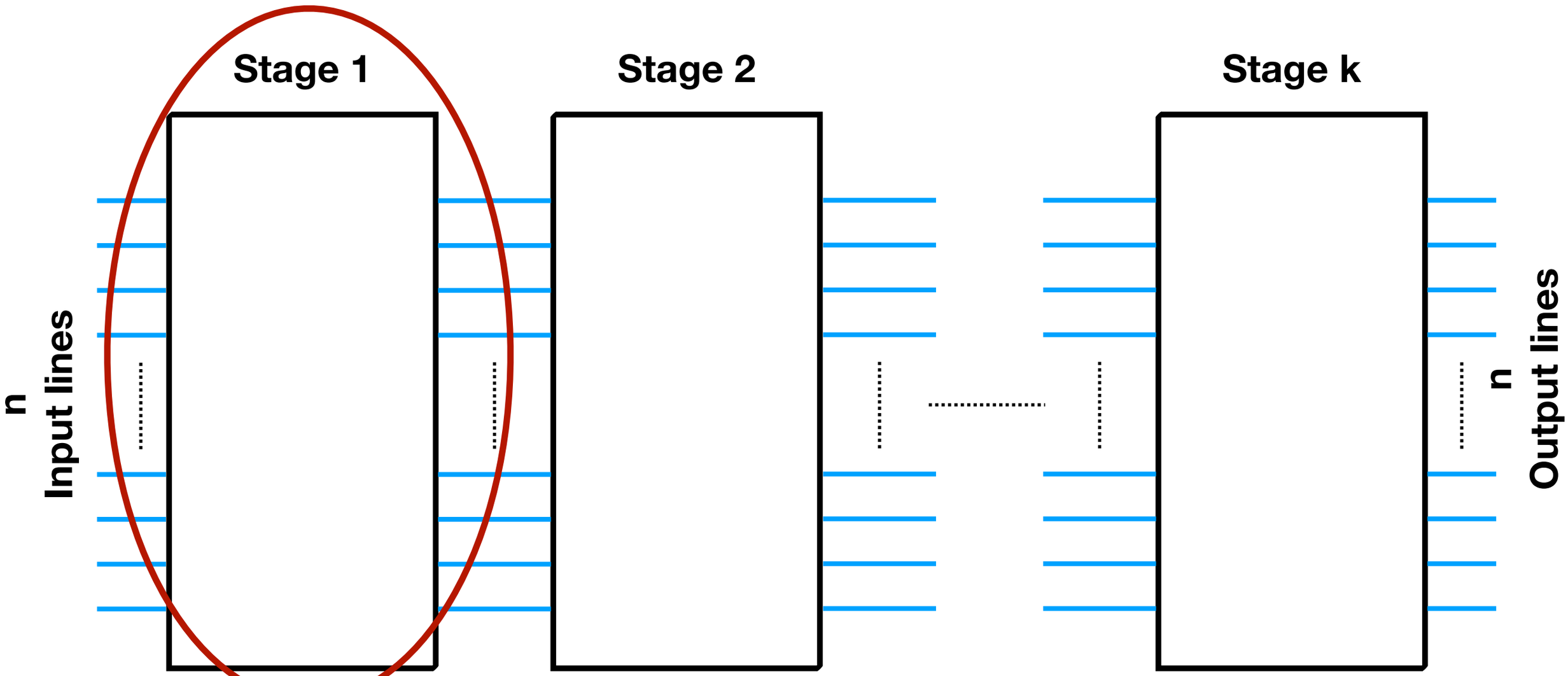
Can express any arbitrary chain of filter operation with chain length <= k on the n input lines

Runs at line rate

Filter Pipeline Layout



Reconfigurable Pipeline

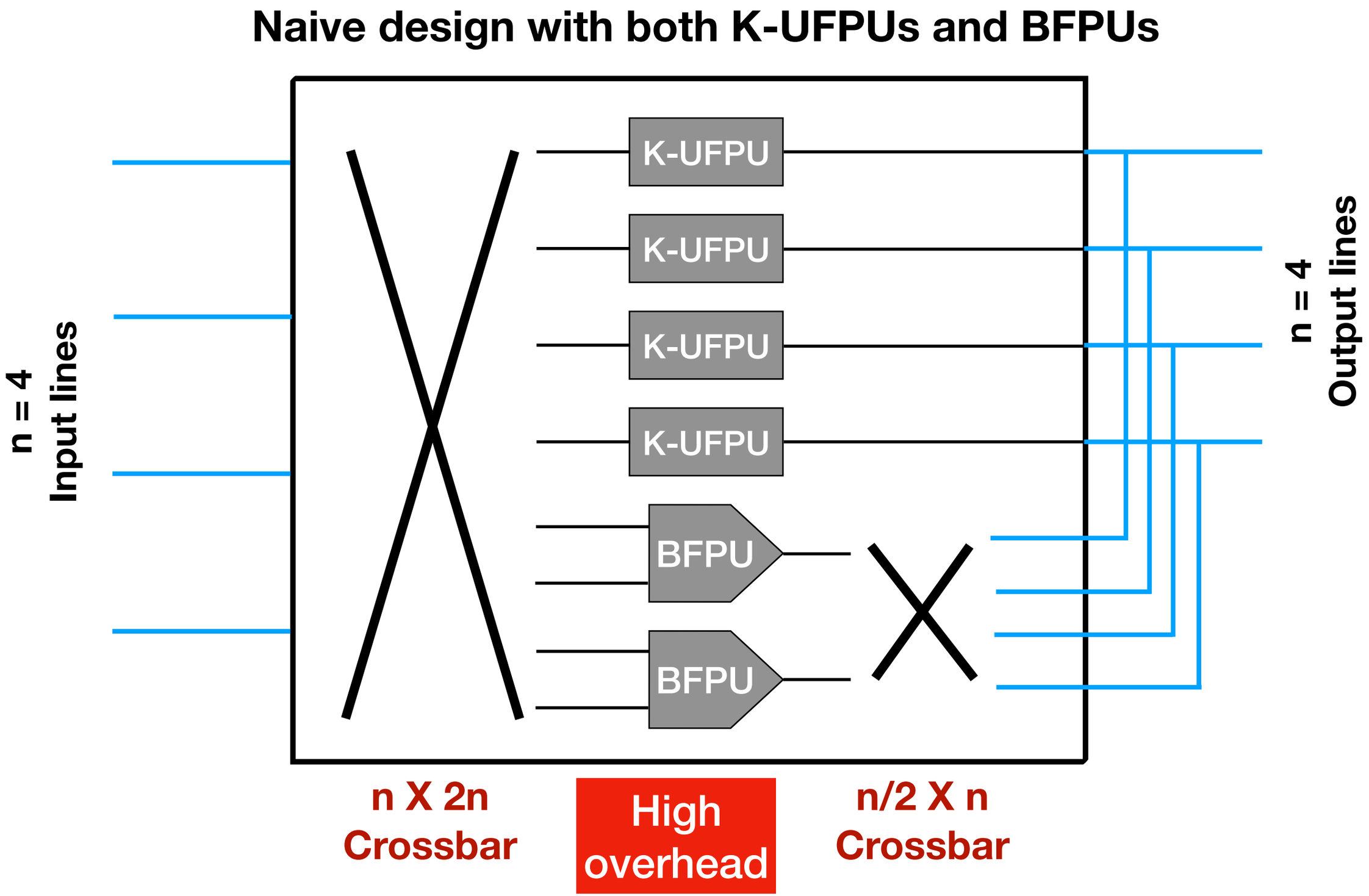


Sufficient condition for full reconfigurability

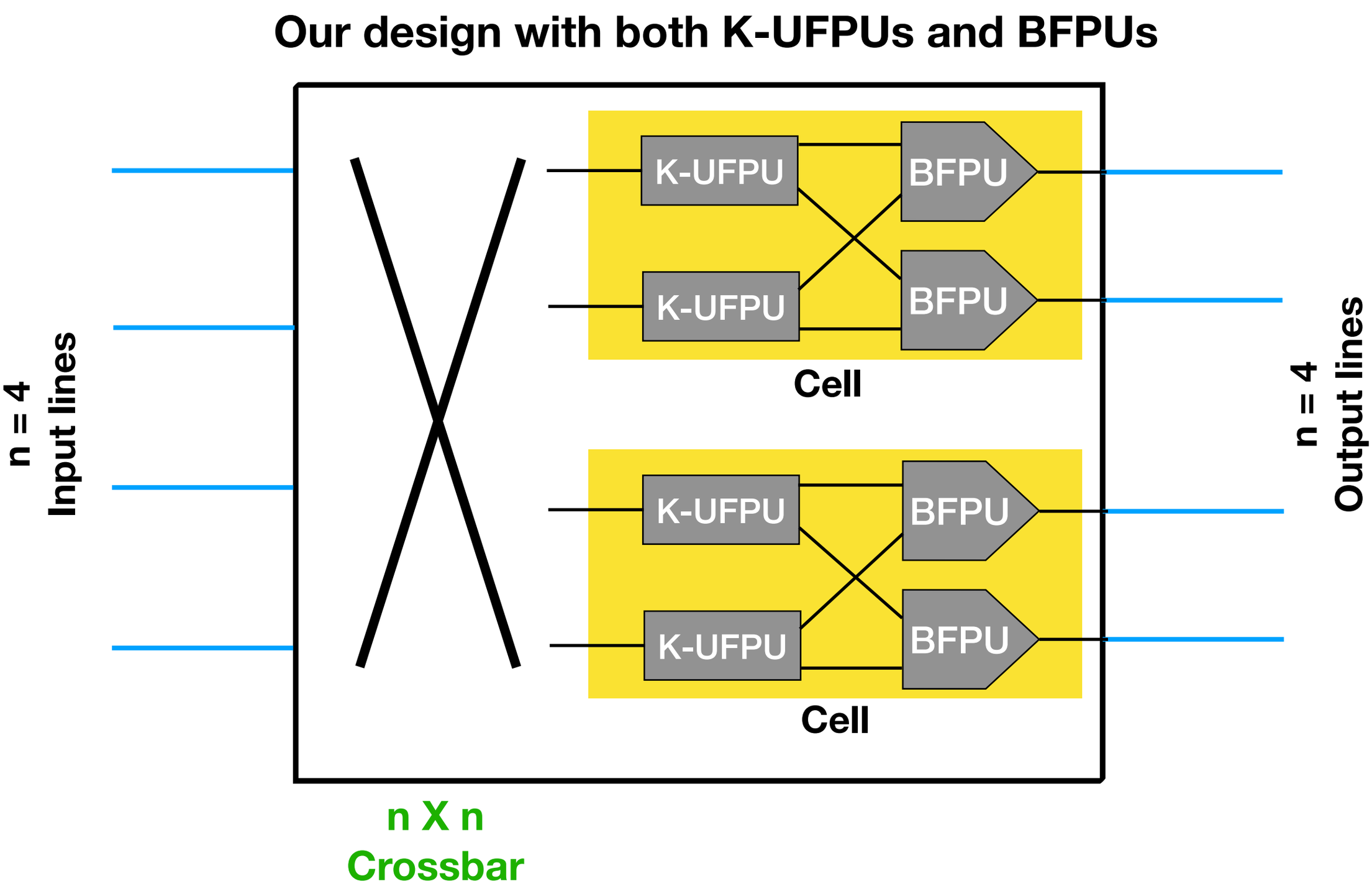
In each stage:

1. Ability to apply any filter operation to an input (or pair of input) line
2. Ability to connect the output of a filter operation to any output line

Single Pipeline Stage Design

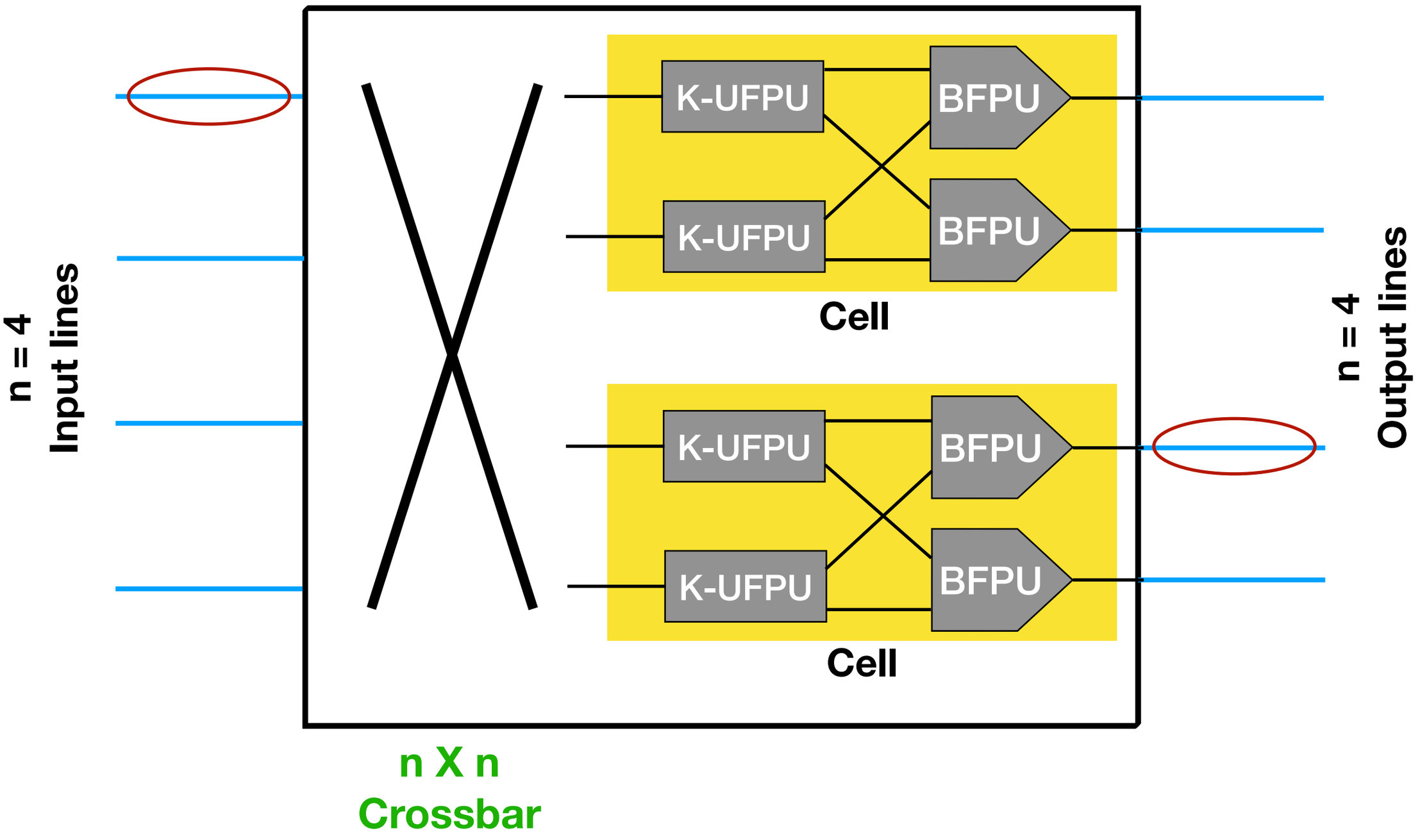


Single Pipeline Stage Design



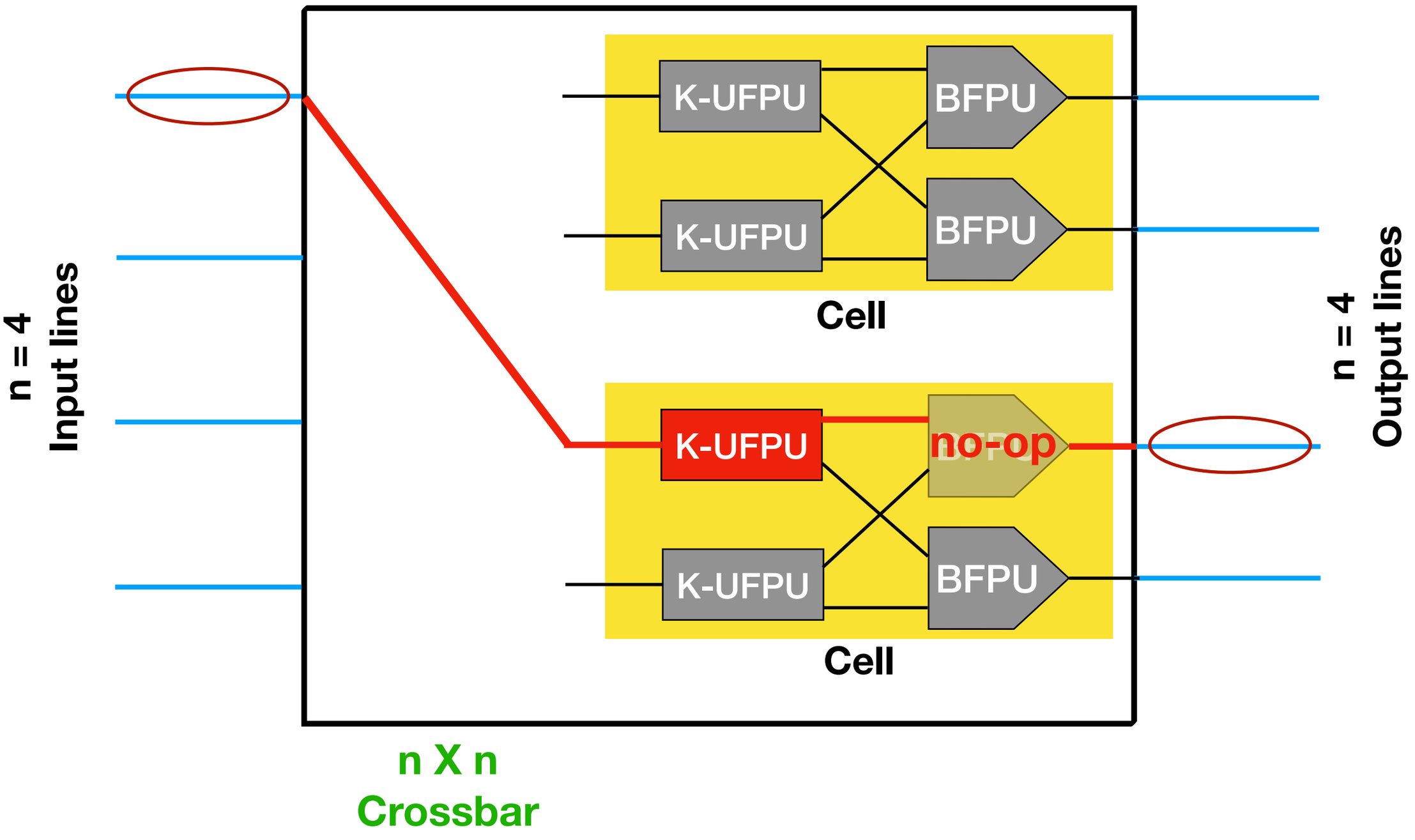
Example Configuration

Apply a UFPU op on Input 1 and connect to Output 3



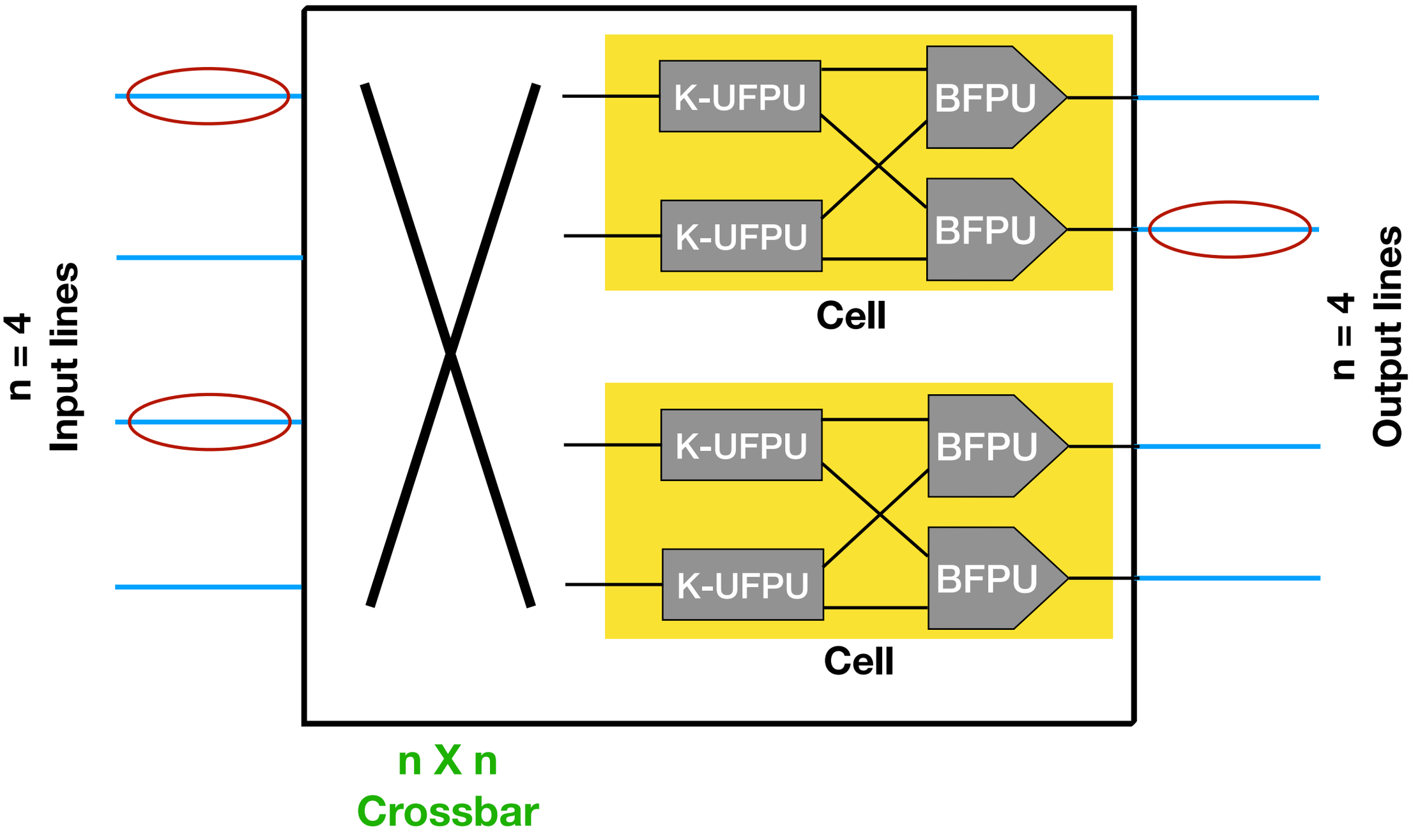
Example Configuration

Apply a UFPU op on Input 1 and connect to Output 3



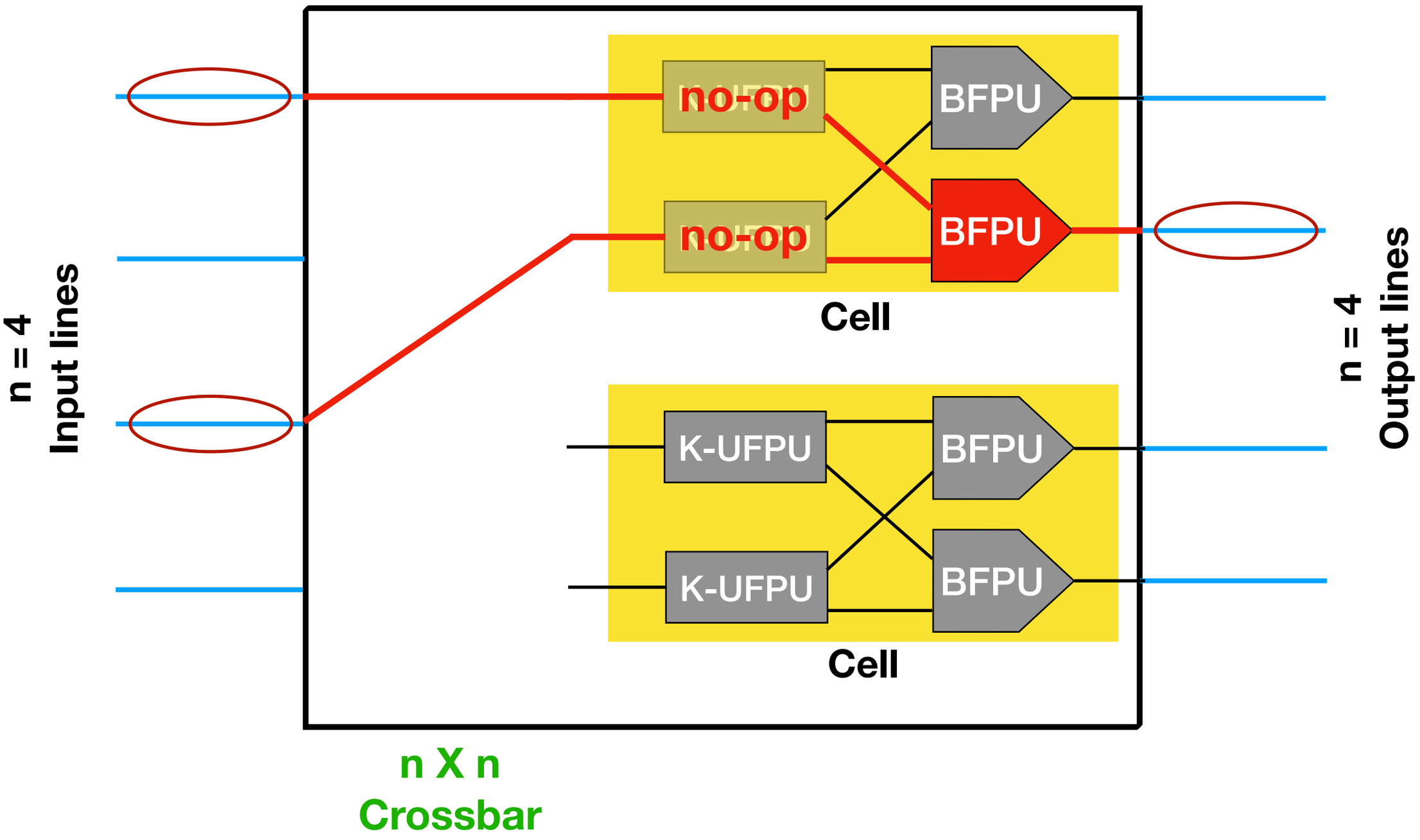
Example Configuration

Apply a BFPU op on Input 1 and 3 connect to Output 2



Example Configuration

Apply a BFPU op on Input 1 and 3 and connect to Output 2



Filter Pipeline Performance

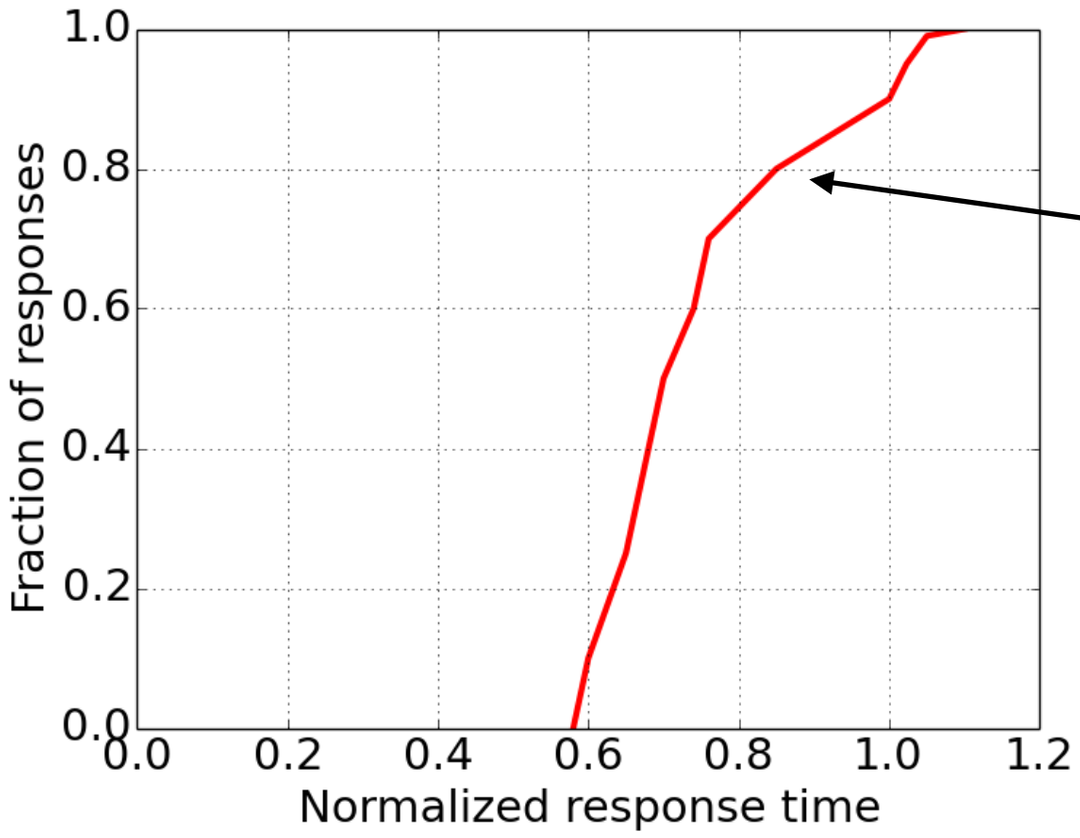
- Filter pipeline can process a new filter request every clock cycle
- Our implementation runs at clock speeds in excess of 1 GHz
  - 1 GHz is the typical clock speed of today's switches
- However, scalability is limited to a few 1000s of table entries
  - …beyond that the clock speed falls below 1 GHz
  - Still sufficient for many applications where table entries include network paths, switch ports, servers in a cluster, etc.

Application Performance

Load balancing client requests

Policy 1. Select a server uniformly at random.

Policy 2. Select a server uniformly at random from the set of servers with CPU utilization < X and available memory > Y and available bandwidth > Z. If the filtered set is empty, select a server uniformly at random from the entire set.

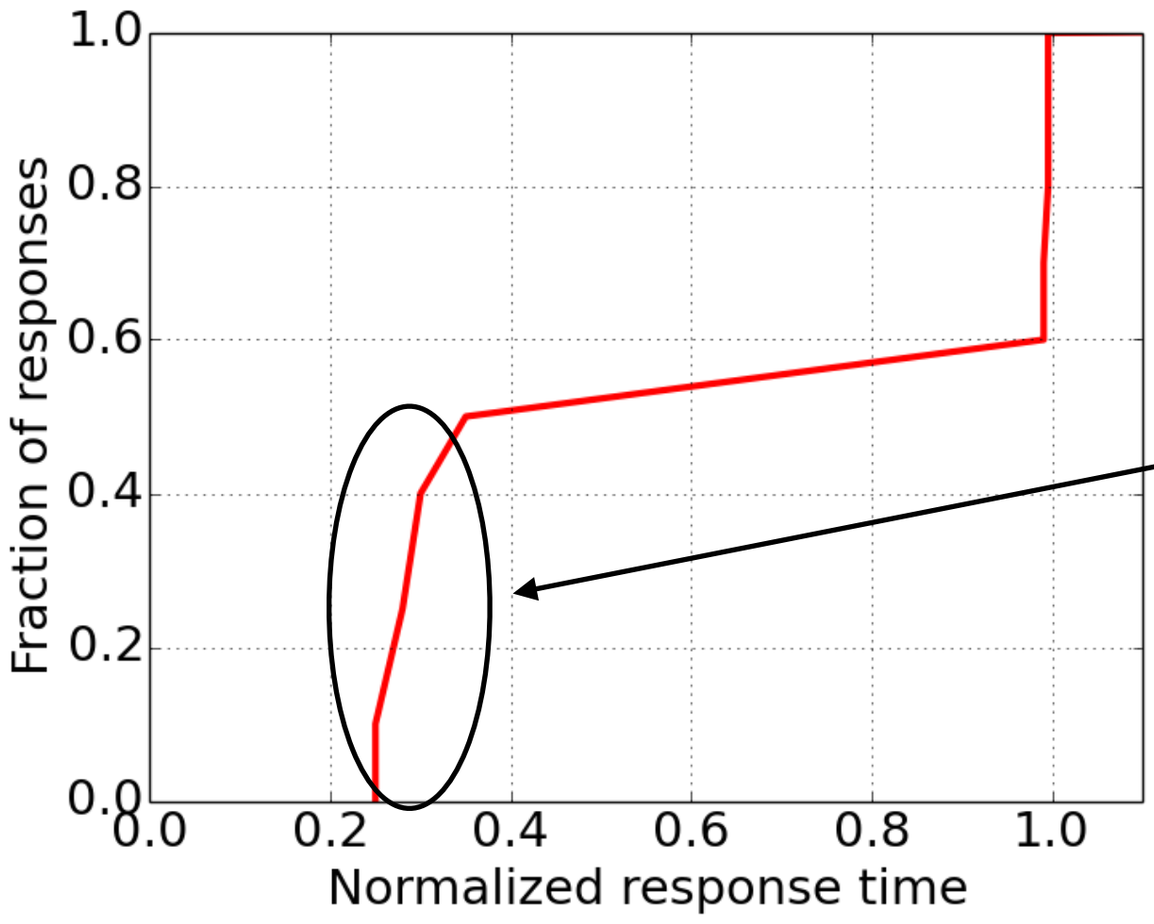


Response time for Policy 2 normalized w.r.t. Policy 1

1.7x—1.3x better response time for 70% of client requests

Application Performance

In-network caching of relational graph filter queries



4x—2.8x better response time for cached filter queries

Response time with caching normalized w.r.t. no caching

Summary

- **Thanos** extends programmable switches with the ability to do programmable line rate filtering over a multi-dimensional table

- Use cases include performance-aware routing, load balancing, network diagnosis, security, firewall…

- The design runs in excess of 1 GHz clock speed and scales to thousands of table entries

- Evaluations show up to 1.7x improvement in performance of key network functions, such as routing and load balancing

- **Overall, advances the current state of in-network computing**
  - enables **richer** algorithms for existing in-network applications
  - enables **new** in-network applications, e.g., caching of relational queries

Thank you!