Scaling Up Clustered Network Appliances with ScaleBricks

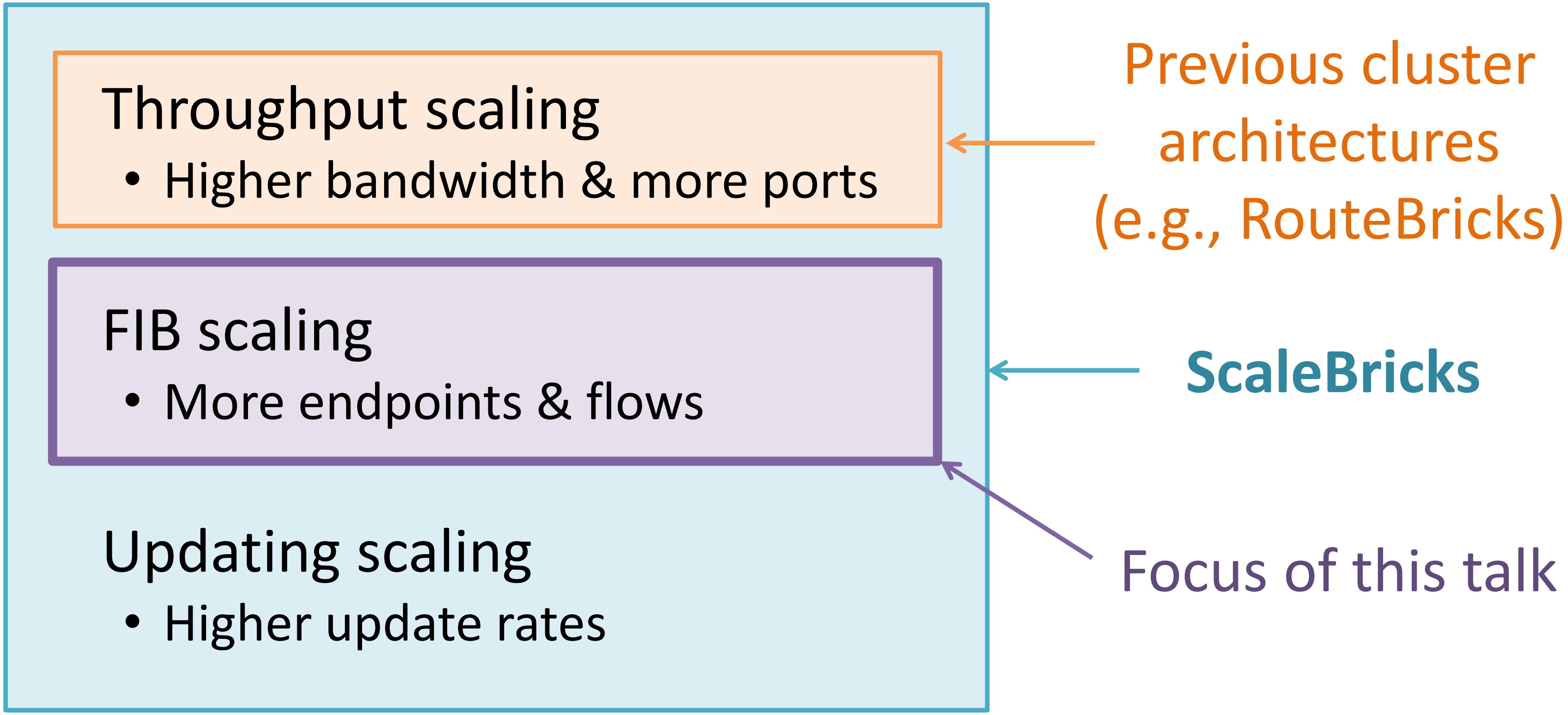
Dong Zhou, Bin Fan, **Hyeontaek Lim**, David Andersen
Michael Kaminsky†, Michael Mitzenmacher**
Ren Wang†, Ajaypal Singh‡

Carnegie Mellon University, †Intel Labs
***Harvard University, ‡Connectem, Inc.*

http://www.istc-cc.cmu.edu/

Intel Science & Technology
Center for Cloud Computing

Scaling Up Clustered Network Appliances



Throughput scaling

- Higher bandwidth & more ports

FIB scaling

- More endpoints & flows

Updating scaling

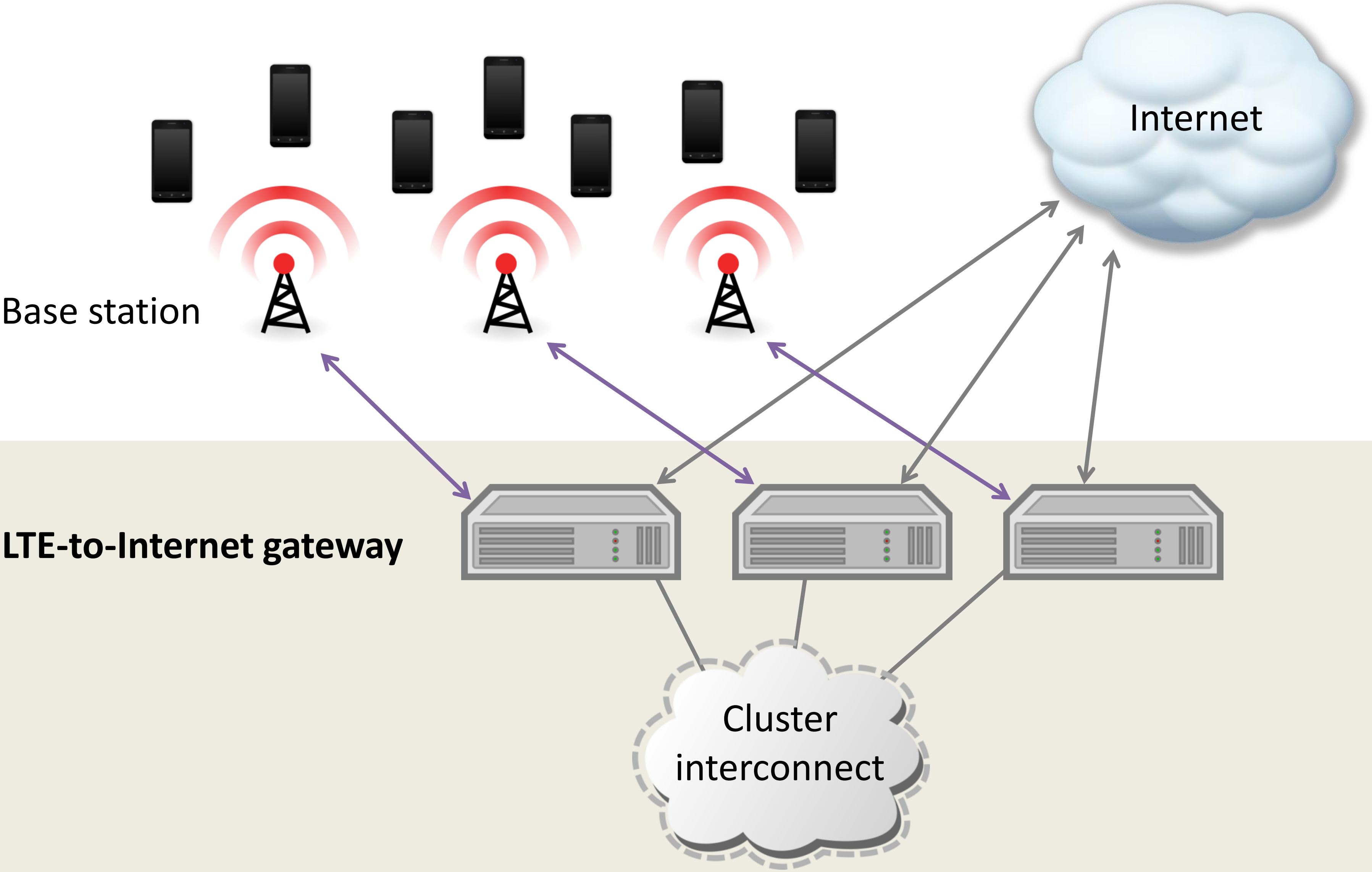
- Higher update rates

Previous cluster architectures (e.g., RouteBricks)

ScaleBricks

Focus of this talk

Motivation: Network Appliance in LTE



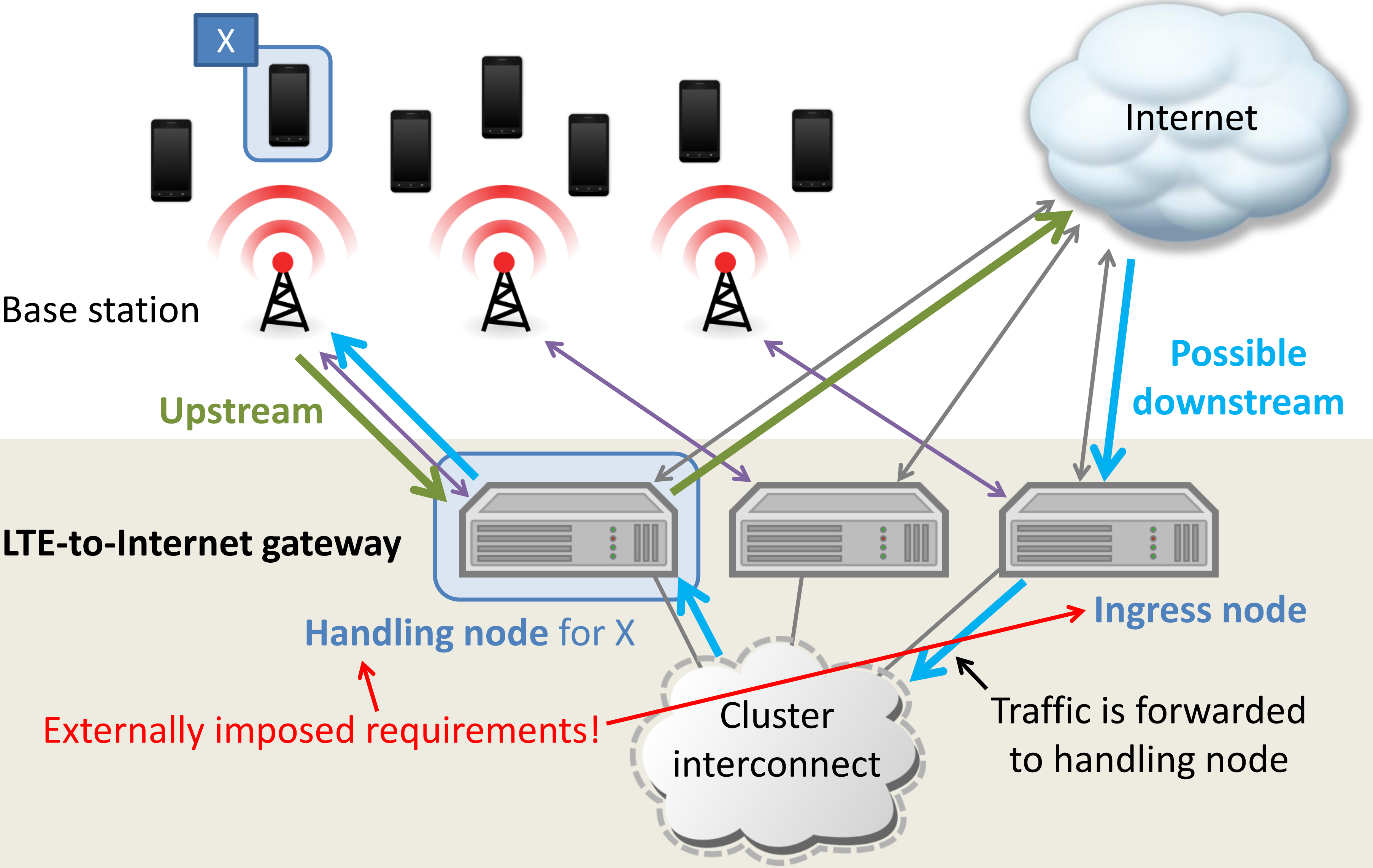
Internet

Base station

LTE-to-Internet gateway

Cluster interconnect

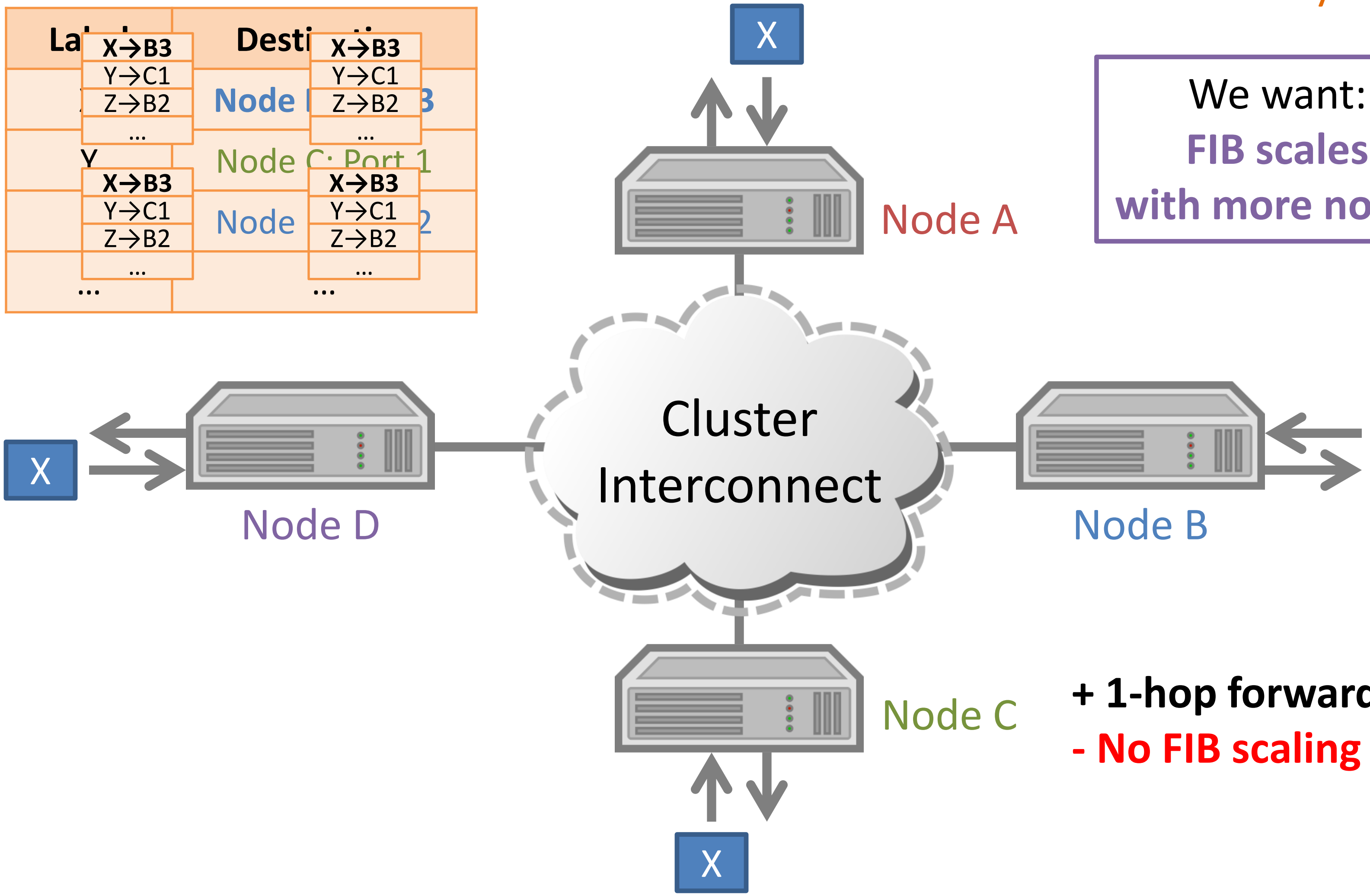
Motivation: Network Appliance in LTE



Fully Duplicated FIB

FIB (forwarding table)

Full FIB on every node



We want:
FIB scales with more nodes

+ 1-hop forwarding
- No FIB scaling

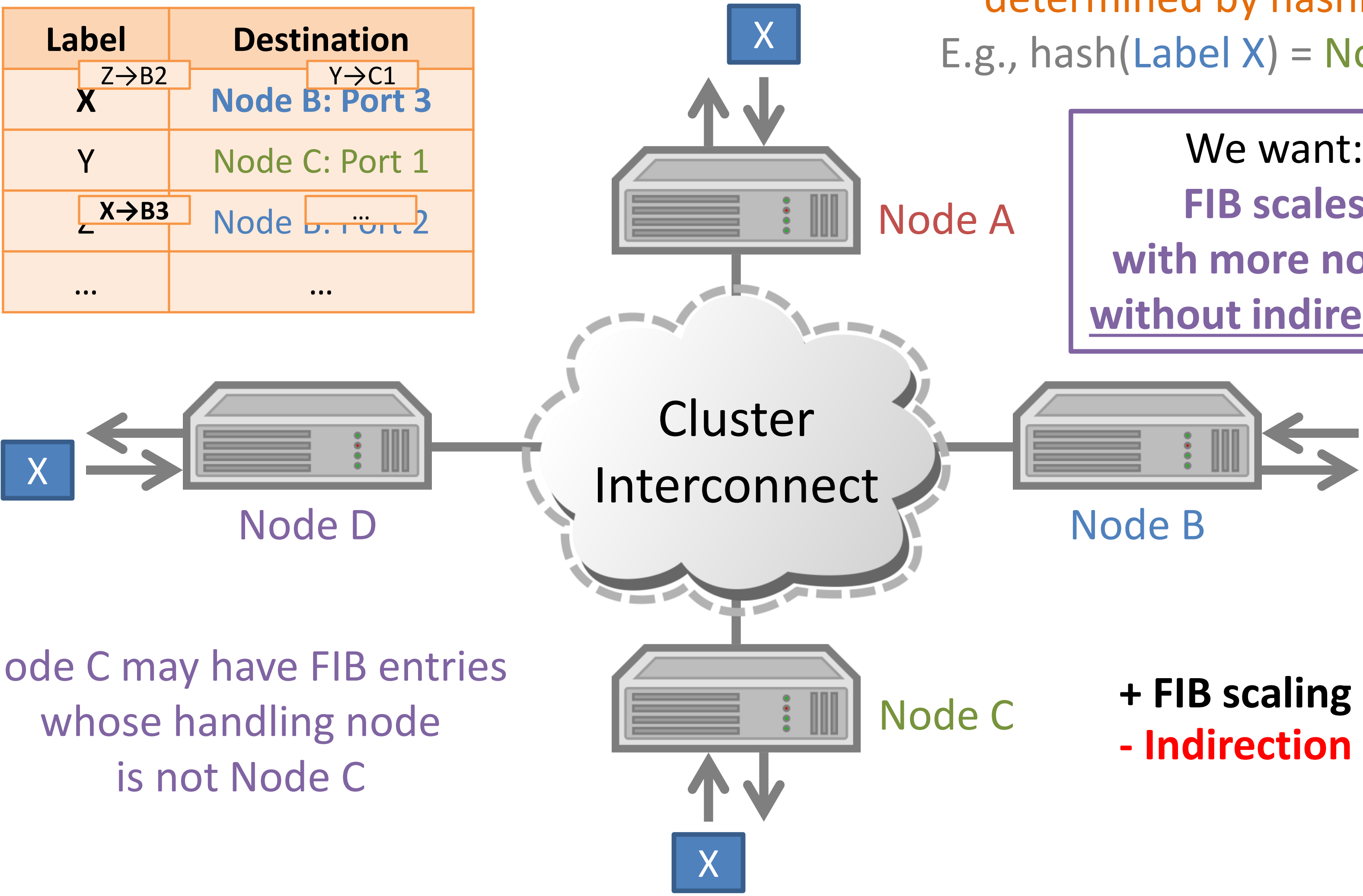
Hash Partitioned FIB

FIB (forwarding table)

Label	**Destination**
Z→B2 **X**	Y→C1 **Node B: Port 3**
Y	Node C: Port 1
X→B3 Z	… Node B: Port 2
…	…

Random partition of FIB determined by hashing

E.g., hash(Label X) = Node C



We want:

FIB scales
with more nodes
without indirection

Node C may have FIB entries whose handling node is not Node C

+ FIB scaling
- Indirection

FIB Scale-Out on Cluster Architectures

- Does the cluster provide **FIB scaling** with more nodes?
- Does the cluster require **indirection** that adds overhead?

Architecture	FIB Scaling	Indirection
Full Duplication	No	No
Hash Partitioning	Yes	Yes
ScaleBricks	**Yes**	**No**

34% lower latency

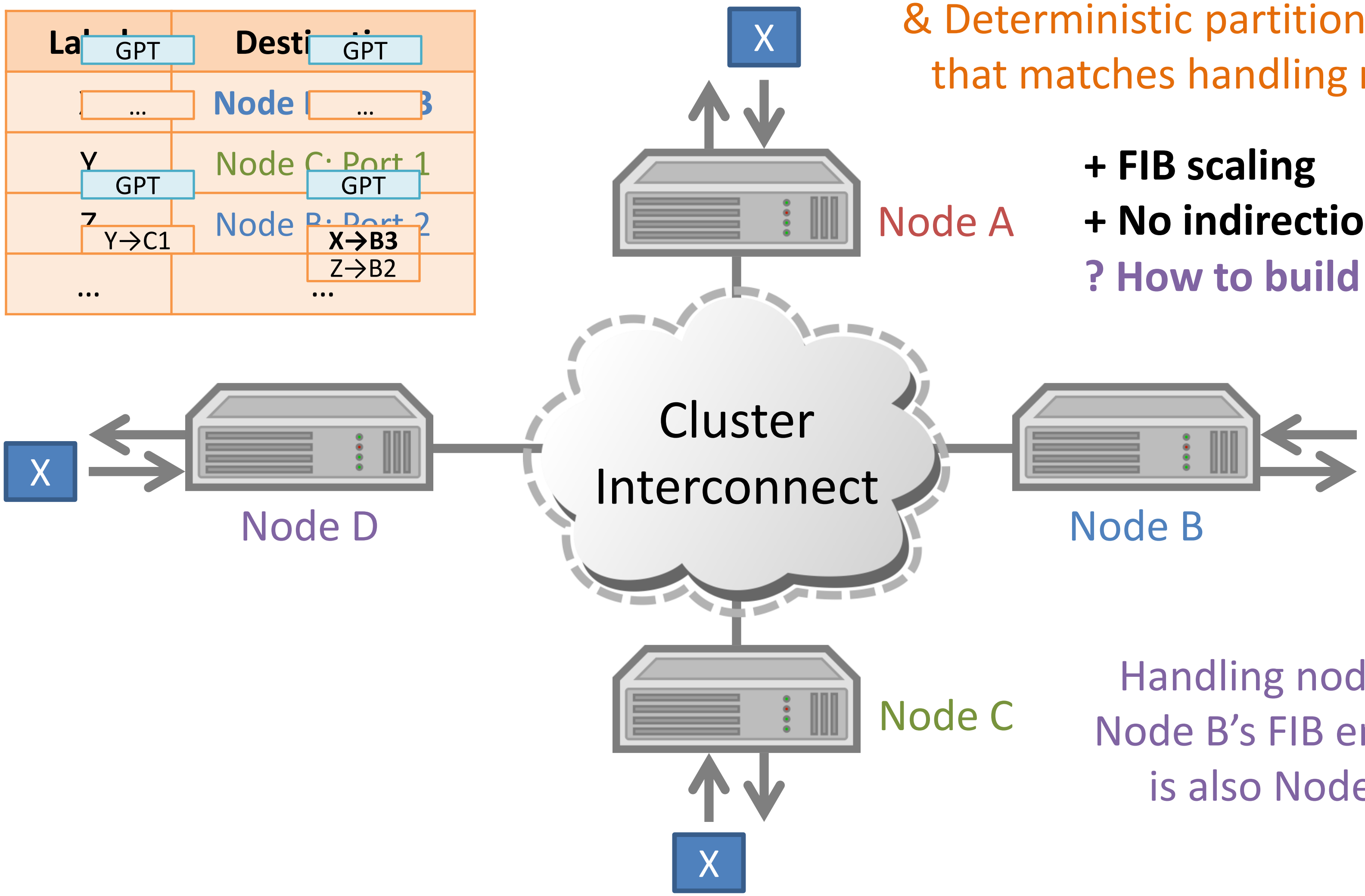
Scaling through 4-32 nodes
10% lower latency & 23% higher throughput

ScaleBricks

X→B
Y→C
Z→B
…

Global Partition Table (GPT)
& Deterministic partition of FIB that matches handling node

FIB (forwarding table)



- **+ FIB scaling**
- **+ No indirection**
- **? How to build GPT**

Handling node of Node B's FIB entries is also Node B

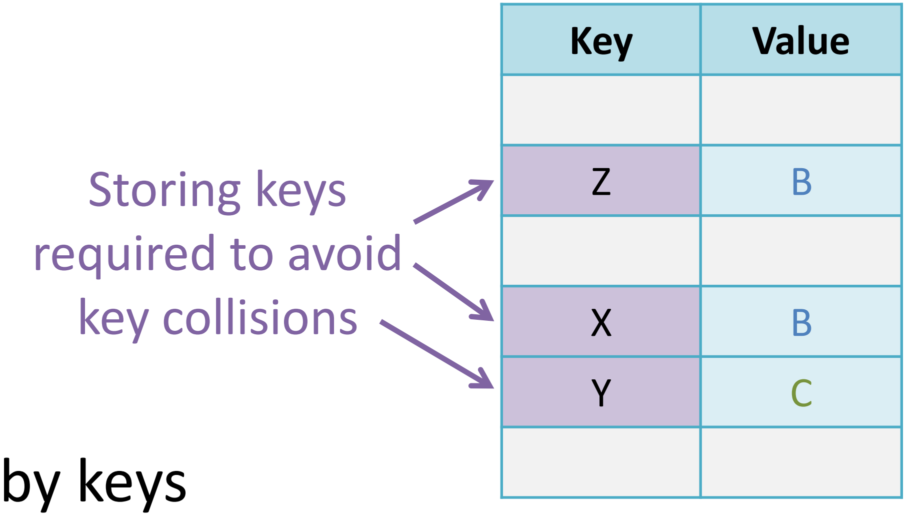
Designing Global Partition Table (GPT)

- GPT should be very small
  - Every node has GPT containing every FIB entry's handling node info.
- Strawman solution: Hash table

FIB (forwarding table)

Label	Destination
X	Node B: Port 3
Y	Node C: Port 1
Z	Node B: Port 2
…	…

Hash table-based GPT



Key	Value
Z	B
X	B
Y	C

Storing keys required to avoid key collisions

- Most table space is taken by keys
  - E.g., 104-bit keys (5-tuple labels) vs. 2-bit values (4 cluster nodes)
- Is there a way to remove keys while avoiding collisions?

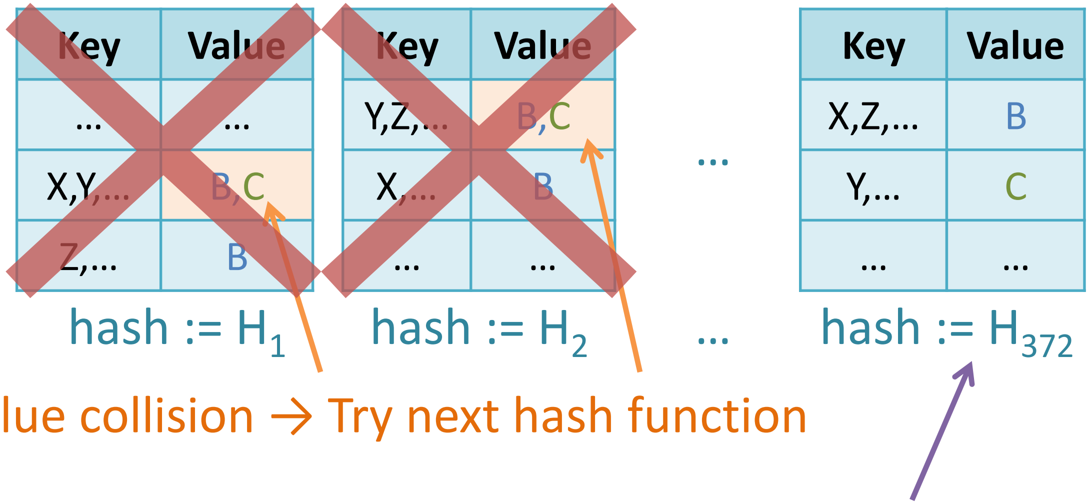
Our Solution for GPT: SetSep

- Practical set separation data structure
  - Do not store keys
  - Brute force to avoid “value” collisions (instead of key collisions)

FIB (forwarding table)

Label	Destination
X	Node B: Port 3
Y	Node C: Port 1
Z	Node B: Port 2
…	…

SetSep-based GPT



~~Key~~	~~Value~~
~~…~~	~~…~~
~~X,Y,…~~	~~B,C~~
~~Z,…~~	~~B~~

hash := H_1

~~Key~~	~~Value~~
~~Y,Z,…~~	~~B,C~~
~~X,…~~	~~B~~
~~…~~	~~…~~

hash := H_2

…

Key	Value
X,Z,…	B
Y,…	C
…	…

hash := H_{372}

Value collision → Try next hash function

No value collision → Use this hash function (“H_{372}”) & value array as GPT

No Key Existence Test in SetSep

Data structure	Existent key	Nonexistent key
Hash table	Correct value	“Key not found”
SetSep	**Correct value**	**Arbitrary value**

- Mitigating arbitrary return values
  - Tolerate arbitrary values for nonexistent keys; or
  - Use additional data structures to detect nonexistent keys

- ScaleBricks uses partial FIB to detect nonexistent keys

Making SetSep Fast

- Construction time problem
  - Exponentially increasing # of trials with more entries and wider values
  - 16→32 entries, 1-bit values: Up to 2^{16} times slower
  - 16 entries, 1→2-bit values: Up to 2^{16} times slower

- SetSep solutions to achieve linear construction time
  - Two-level hashing to divide entries into small, evenly-sized sets
  - Separate hash functions to encode individual value bits

See our SIGCOMM 2015 paper for more details

- Trading space for faster construction by using sparser value array
- Fast generation of many hash functions
- Fast batched lookups with memory prefetching

Main Properties of SetSep

- Compact size
  - $0.5+1.5\lceil \log_2(\text{node count}) \rceil$ bits/entry
  - E.g., 3.5 bits/entry for 4 nodes

- Reasonably fast construction
  - 0.24 million entries/sec (1 thread)

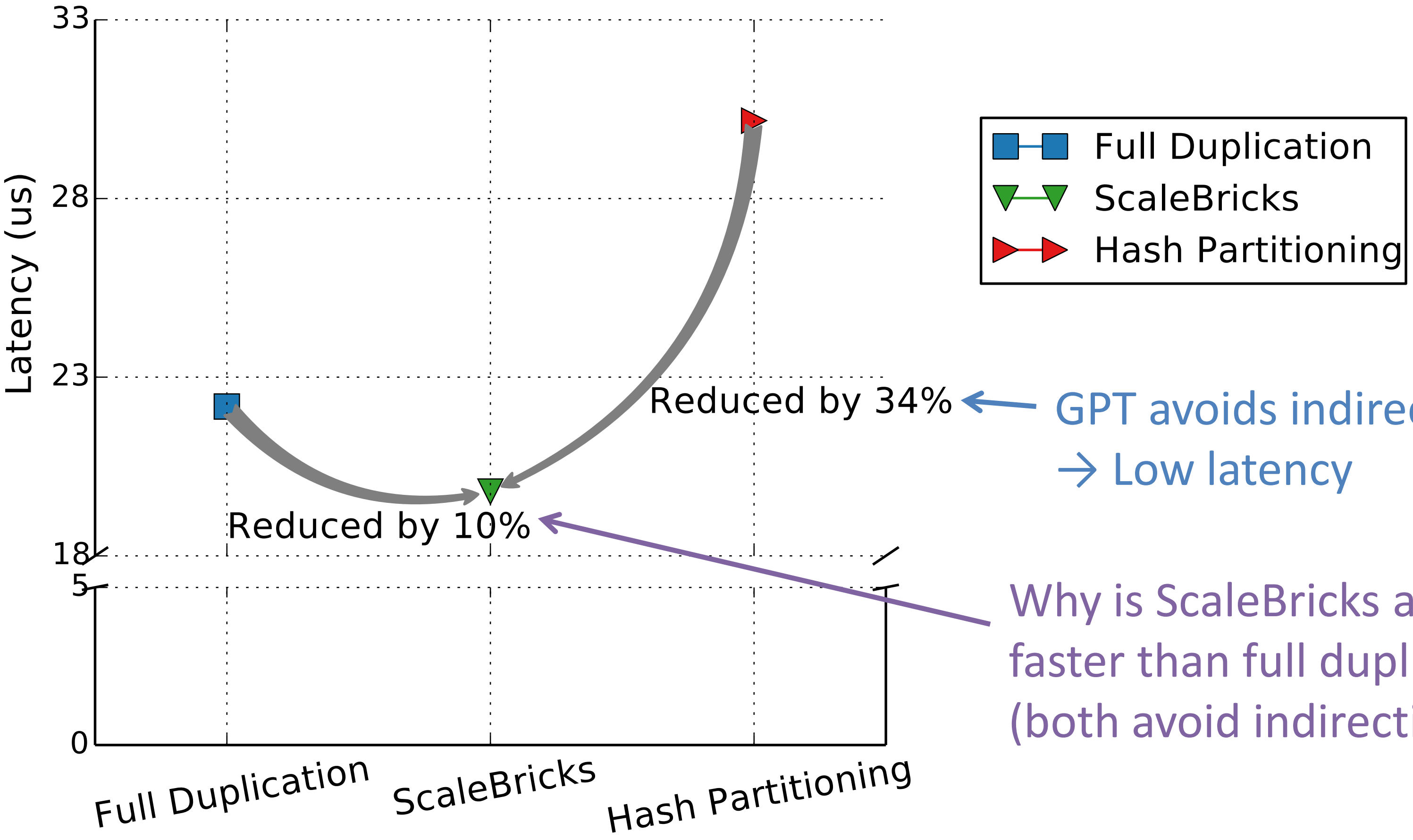
- Fast lookup
  - 520 million lookups/sec (16 threads)

Evaluation Overview

- Full-system forwarding performance
- Scalability analysis

- Setup
  - Modified Connectem’s LTE Evolved Packet Core stack
    - Using Intel DPDK
  - Traffic generated by Spirent SPT-N11U Ethernet testing platform
  - 4x commodity server nodes
    - 2x Intel Xeon E5-2697 v2 (30 MiB L3 cache)
    - 2x Intel 82599ES (dual-port 10 GbE NIC)
  - 10 GbE hardware switch as cluster interconnect

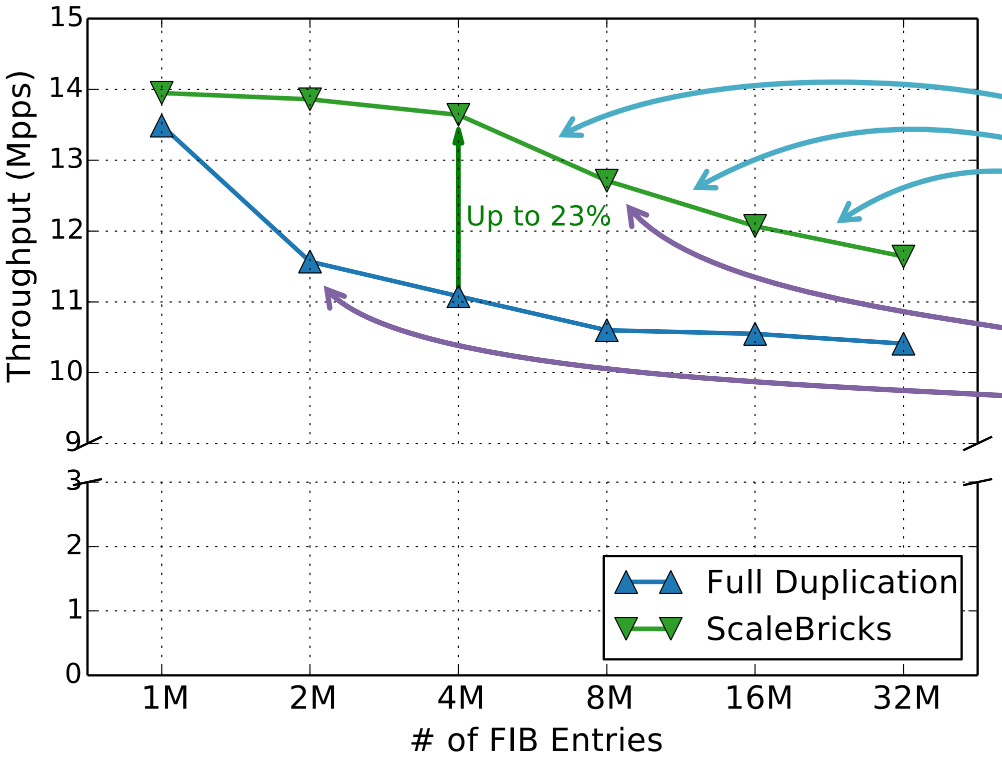
End-to-End Latency with 4 Nodes



GPT avoids indirection
→ Low latency

Why is ScaleBricks also faster than full duplication? (both avoid indirection)

Per-Node Throughput



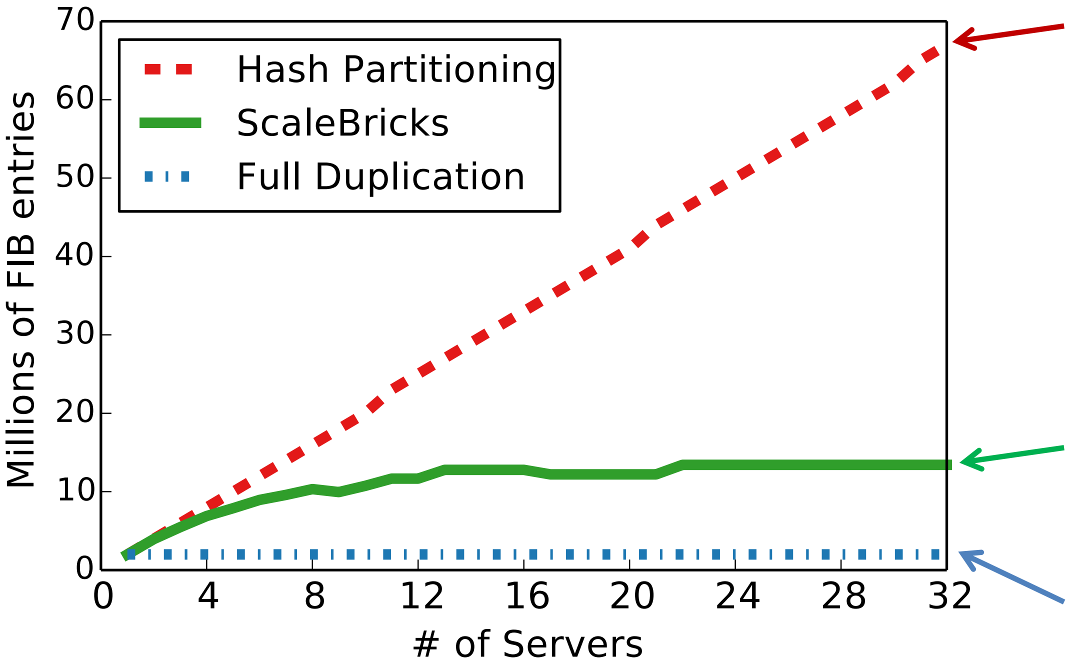
ScaleBricks exhibits graceful tput decrease

ScaleBricks begins to slow with larger FIB than full duplication does

ScaleBricks’s partial FIB is smaller than full FIB
→ More FIB entries fit in CPU L3 cache
→ Higher throughput & lower latency

Scalability Analysis

Aggregate FIB size when each node uses **16 MiB** of memory



Hash partitioning shows very good FIB scaling (at the cost of latency & throughput)

ScaleBricks offers FIB scaling through 4-32 nodes

Full duplication provides no FIB scaling

Conclusion

- ScaleBricks: Scalable cluster architecture for middleboxes
  - Global Partition Table + Partial FIB: FIB scaling without indirection
  - 23% higher tput, 34% lower latency, FIB scaling through 4-32 nodes
- SetSep: Compact key-value mapping for small value space
  - Skip storing keys, brute force to avoid value collisions
  - Small memory overhead, fast lookup, good construction speed
- Applications
  - Clustered network appliances with flow pinning
  - We are looking for other cool applications of ScaleBricks and SetSep!