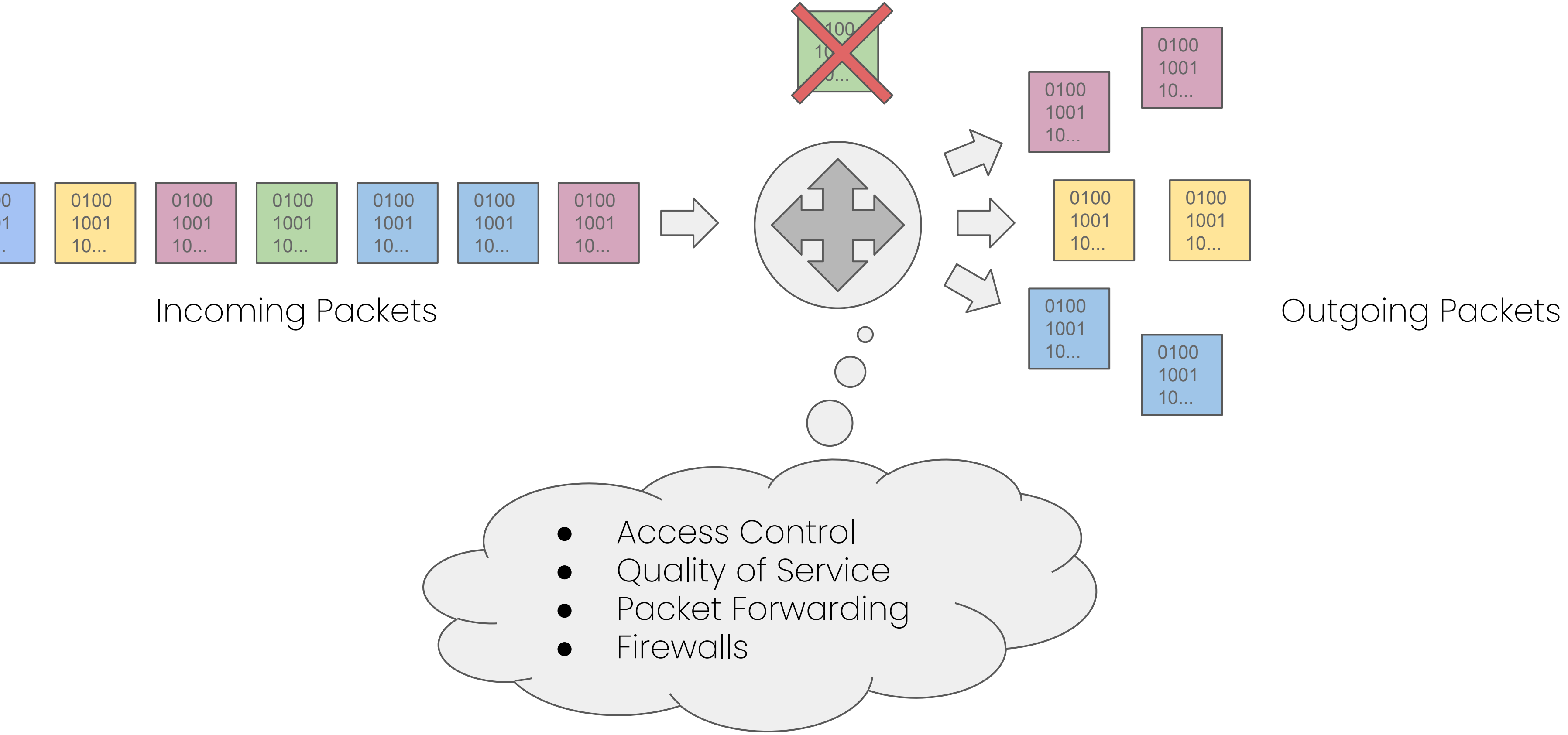
A Computational Approach to Packet Classification

Alon Rashelbach Ori Rottenstreich Mark Silberstein

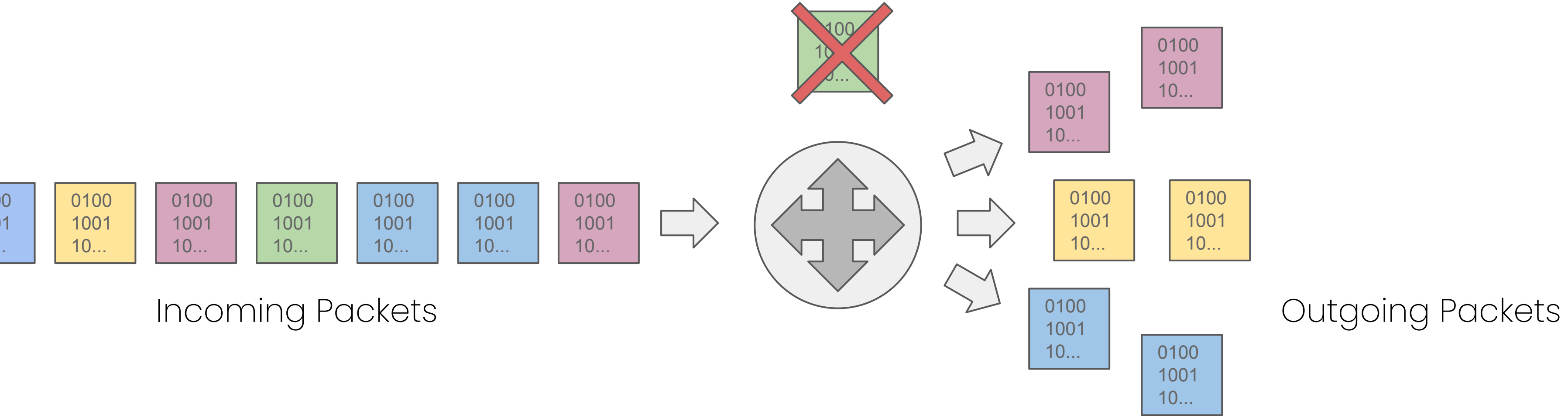
Technion, Israel

SIGCOMM 2020

A Brief about Packet Classification (1/3)

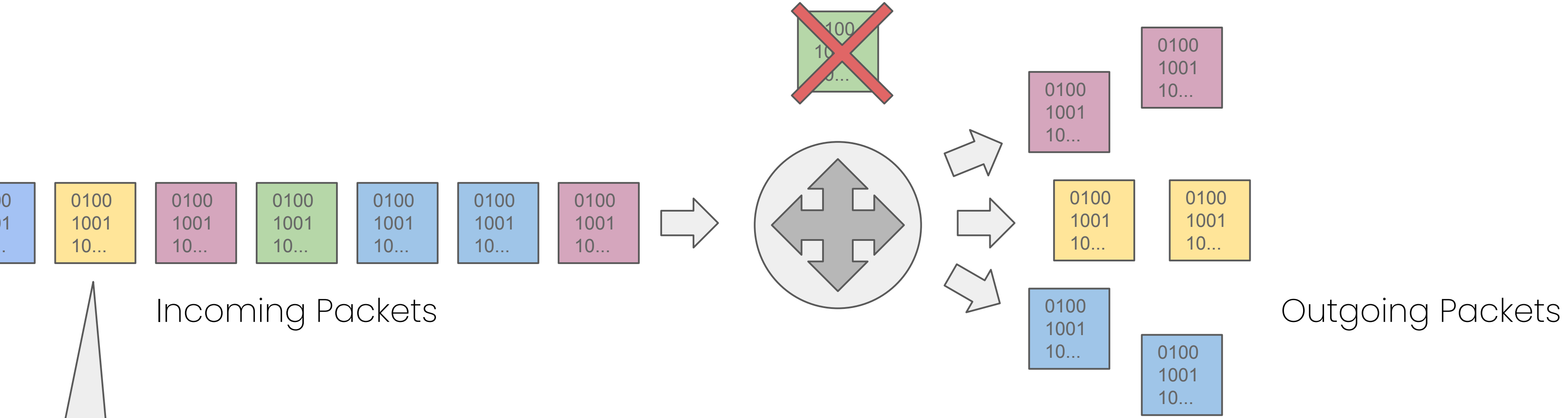


A Brief about Packet Classification (2/3)



Src IP	Dst IP	Src Port	Dst Port	Action	Priority
10.0.10.*	8.8.*.*	0-65535	80	Port 1	3
10.0.20.*	8.8.7.*	0-65535	443	Port 2	2
10.0.*.*	8.8.7.1	0-65535	0-65535	Drop	1

A Brief about Packet Classification (3/3)



src-ip: 10.0.10.1
dst-ip: 8.8.7.1
src-port: 7870
dst-port: 80

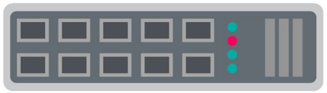
Src IP	Dst IP	Src Port	Dst Port	Action	Priority
10.0.10.*	8.8.*.*	0-65535	80	Port 1	3
10.0.20.*	8.8.7.*	0-65535	443	Port 2	2
10.0.*.*	8.8.7.1	0-65535	0-65535	Drop	1

Hardware vs. Software

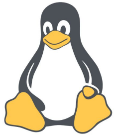
Ternary Content Addressable Memory (TCAM)

Various Algorithms

Hardware ⟵⟶ Software

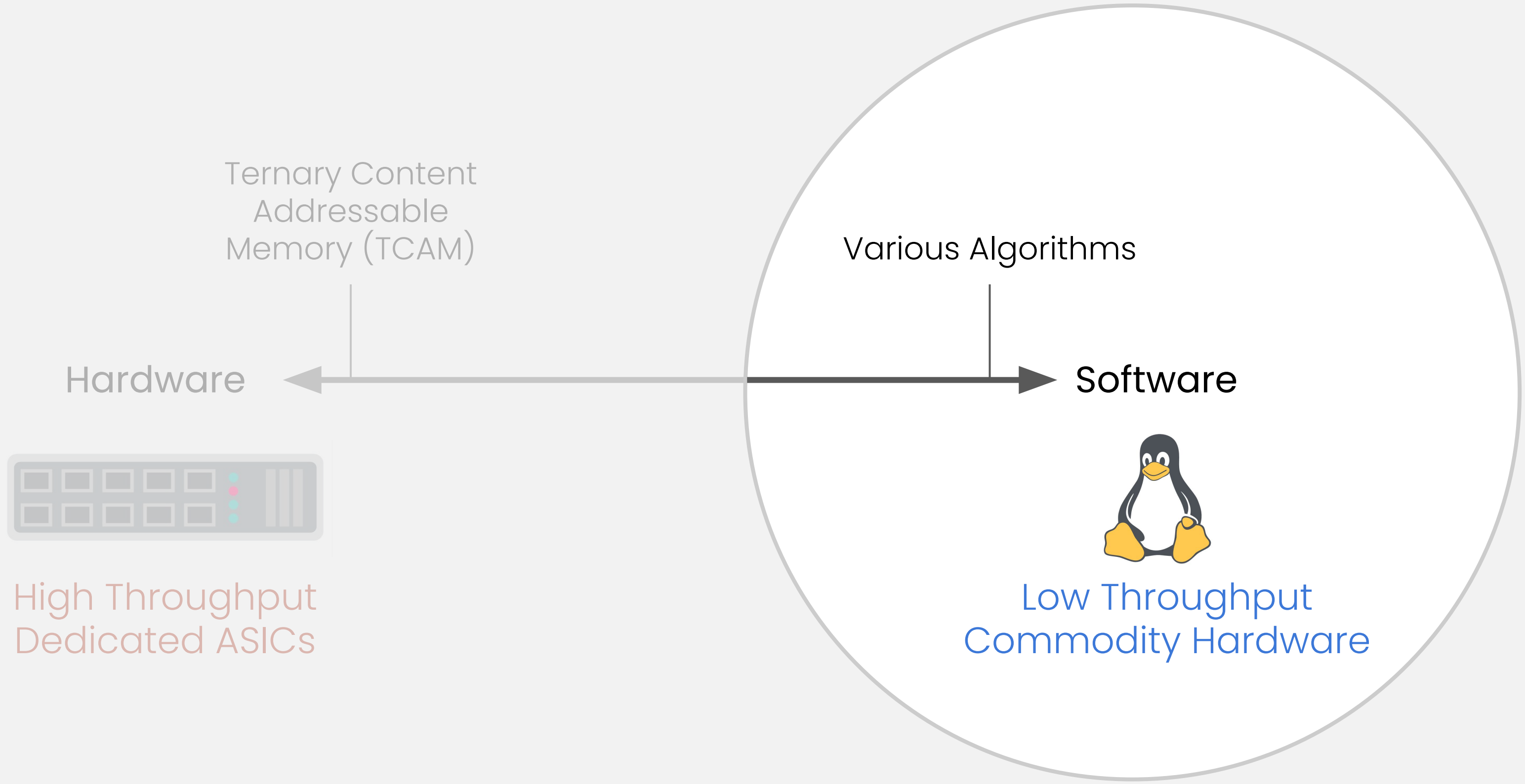


High Throughput
Dedicated ASICs

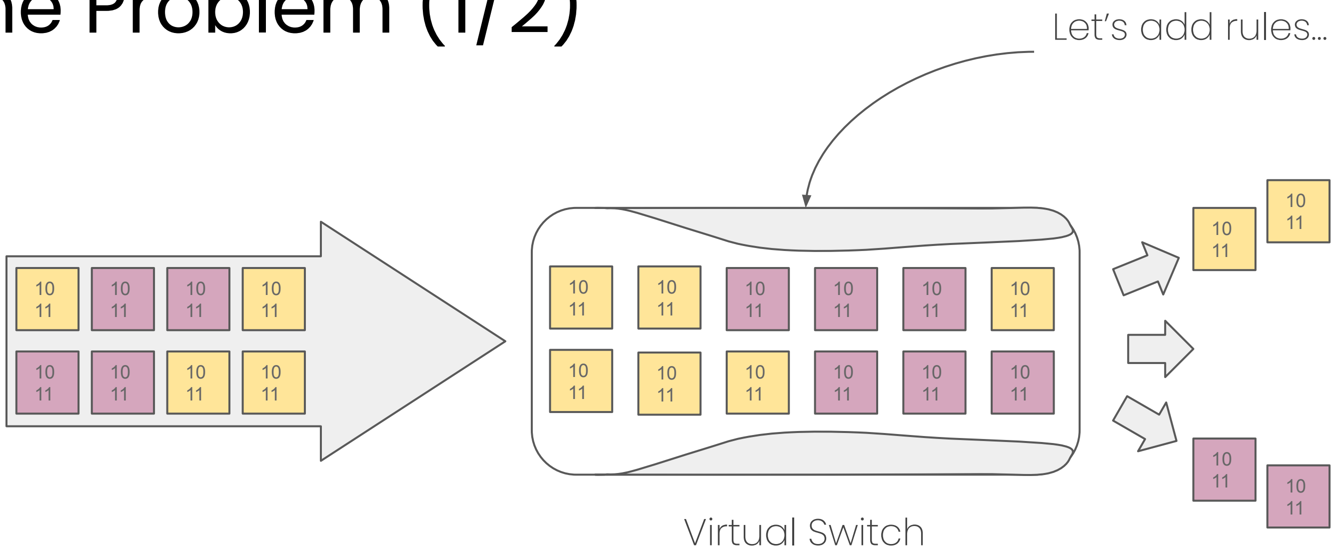


Low Throughput
Commodity Hardware

Hardware vs. Software



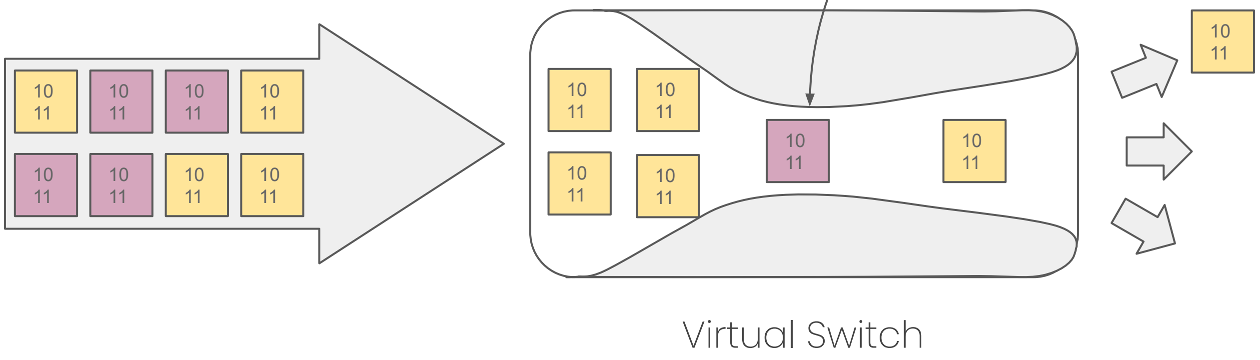
The Problem (1/2)



As we add more rules to virtual switches...

The Problem (2/2)

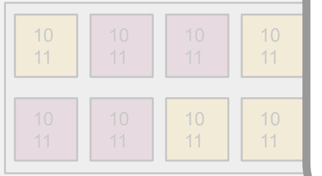
Let's add rules...
Bottleneck!



...we lower their throughput!

The Problem (2/2)

Let's add rules...
Bottleneck!

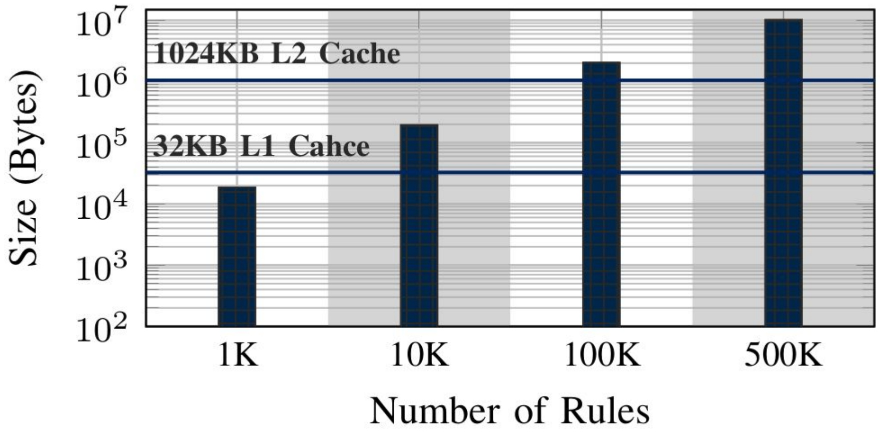


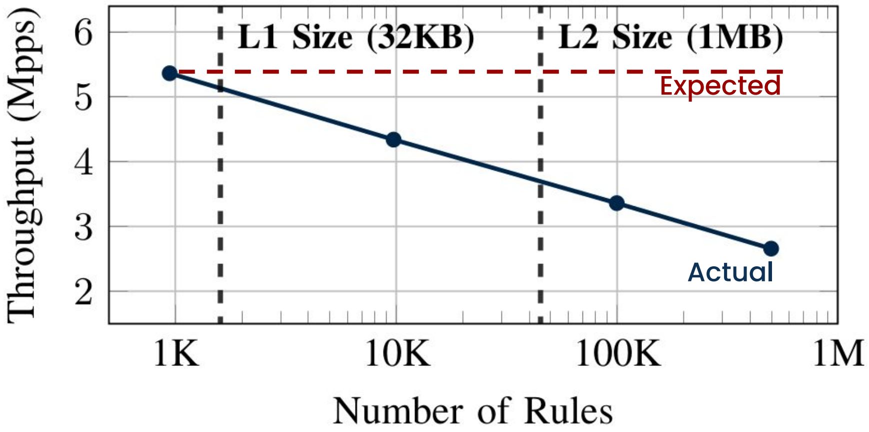
Large Rule Sets
Spill Out of CPU Core Cache!

Virtual Switch

...we lower their throughput!

Large Rule Sets Spill Out of CPU Cache (1/2)

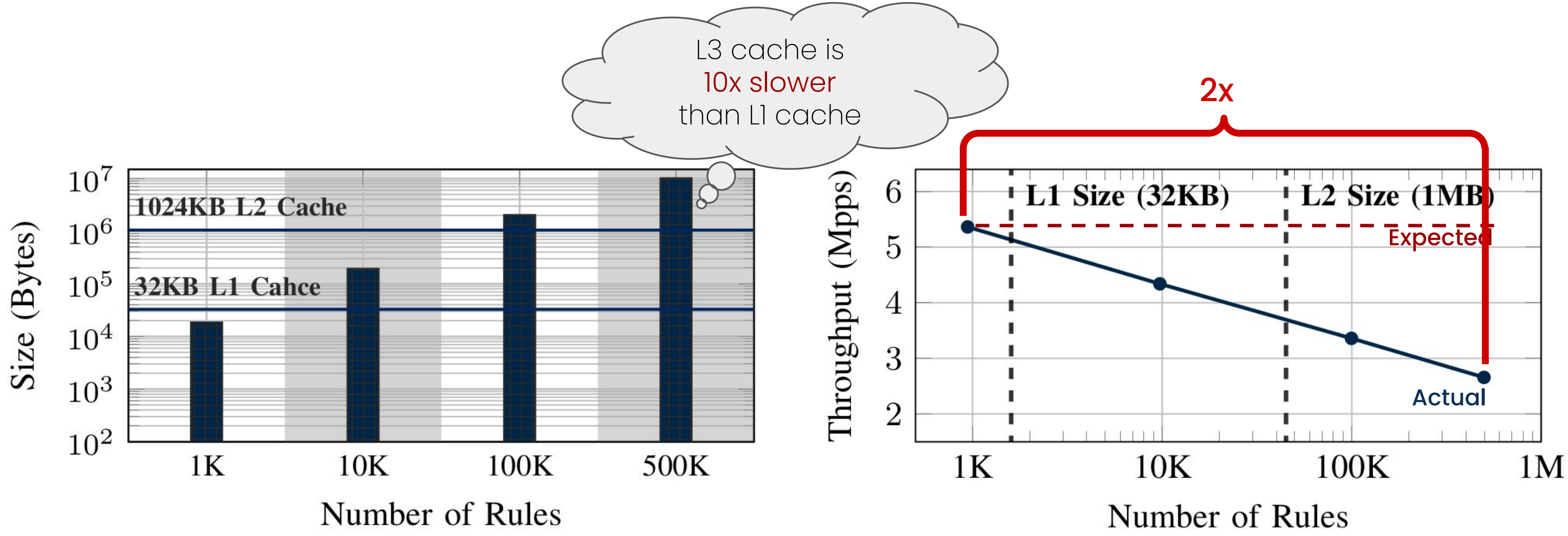




TupleMerge:
State-of-the-art

(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*

Large Rule Sets Spill Out of CPU Cache (2/2)



TupleMerge:
State-of-the-art

(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Taylor et al., “Classbench: A Packet Classification Benchmark”. *TON 2007.*

Observations

1. The rule space is sparse

2. L3 Cache and DRAM access time is the major bottleneck

3. Hardware trends towards commodity accelerators

Observations

1. The ru

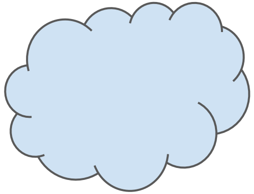
2. L3 Ca

3. Hardware trends towards commodity accelerators

Let's trade
memory accesses for computations

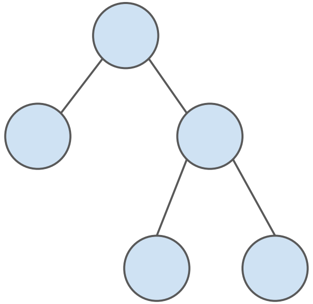
Recursive Model Index (RMI) (1/5)

Key	Value
56	value0
60	value1
68	value2
71	value3
80	value4



Model

≠

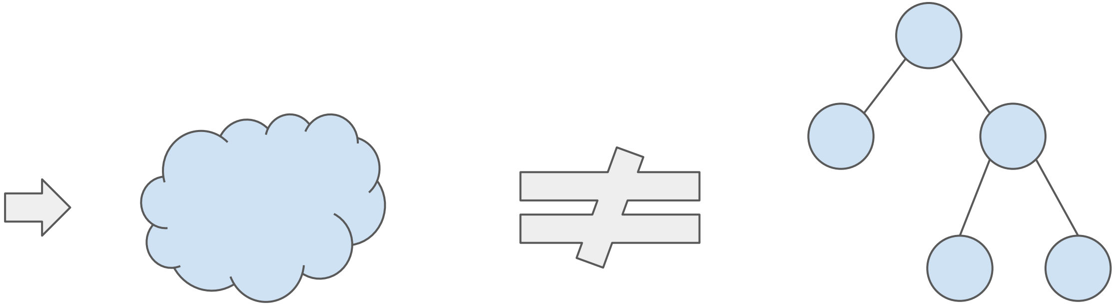


Btree

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

Recursive Model Index (RMI) (1/5)

Key	Value
56	value0
60	value1
68	value2
71	value3
80	value4



Model

0.15 MB
Neural Network Inference
98ns per lookup

Btree

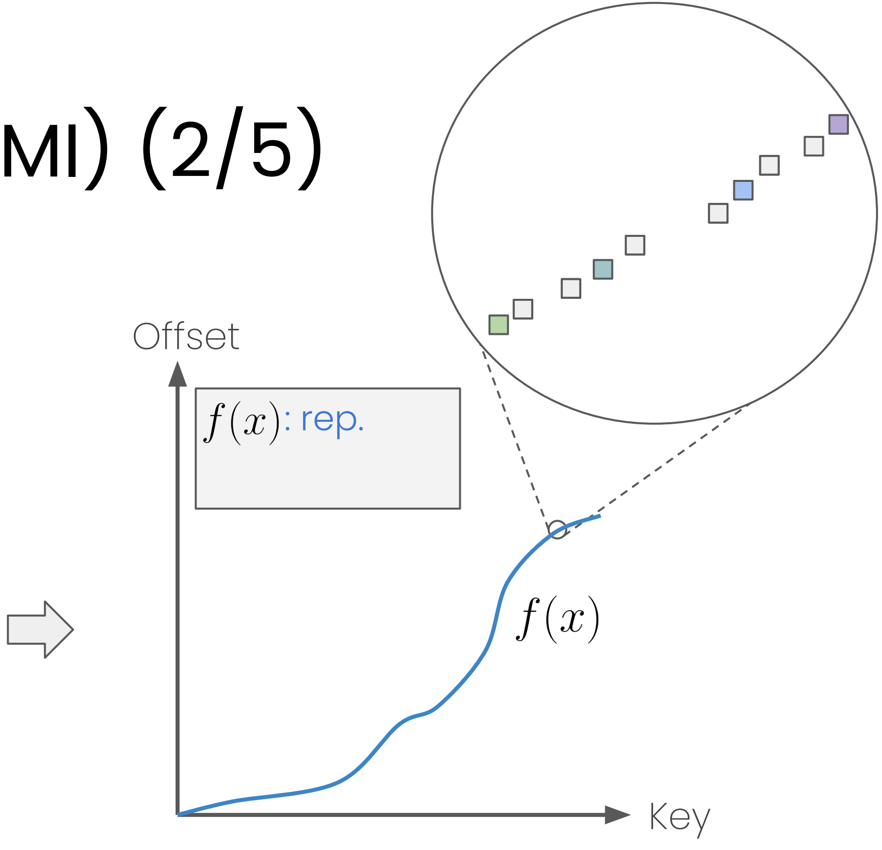
13 MB
Tree Traversal
256ns per lookup

2.7x lookup performance in databases

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

Recursive Model Index (RMI) (2/5)

Mem Offset	Key	Value
100	56	value0
101	60	value1
102	68	value2
103	71	value3
104	80	value4
105	87	value5
106	93	value6
107	100	value7
108	101	value8
109	117	value9



$$IndexedKeys = \{ \quad \}$$

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

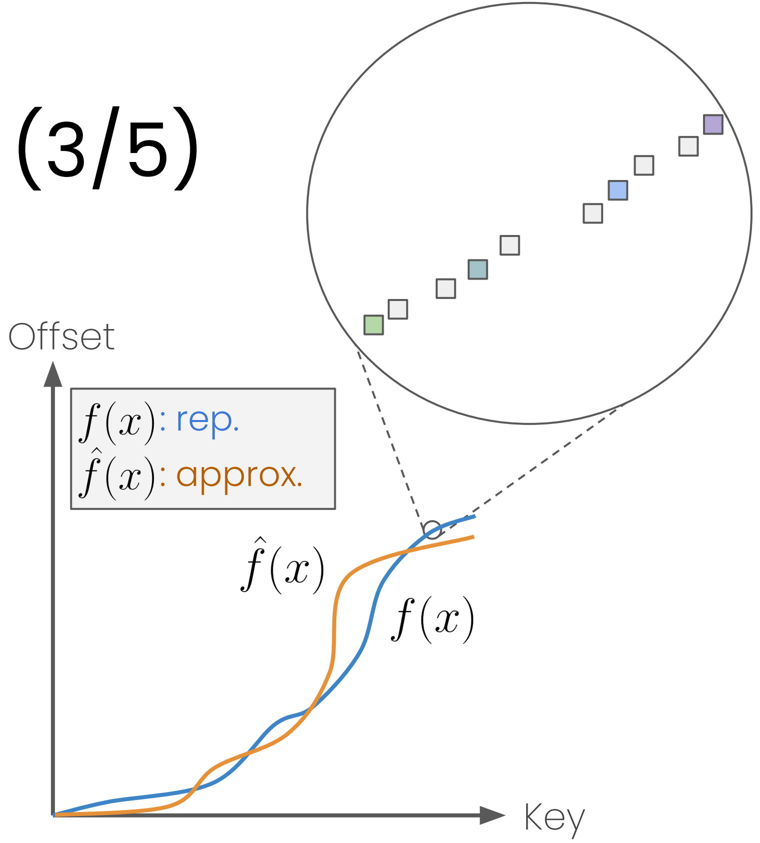
Recursive Model Index (RMI) (3/5)

x ⇨ Model ⇨ $\hat{f}(x)$

Max Approximation Error

$$|f(x) - \hat{f}(x)| < \epsilon$$
$$x \in Keys$$

Exhaustive Scan Over All Keys

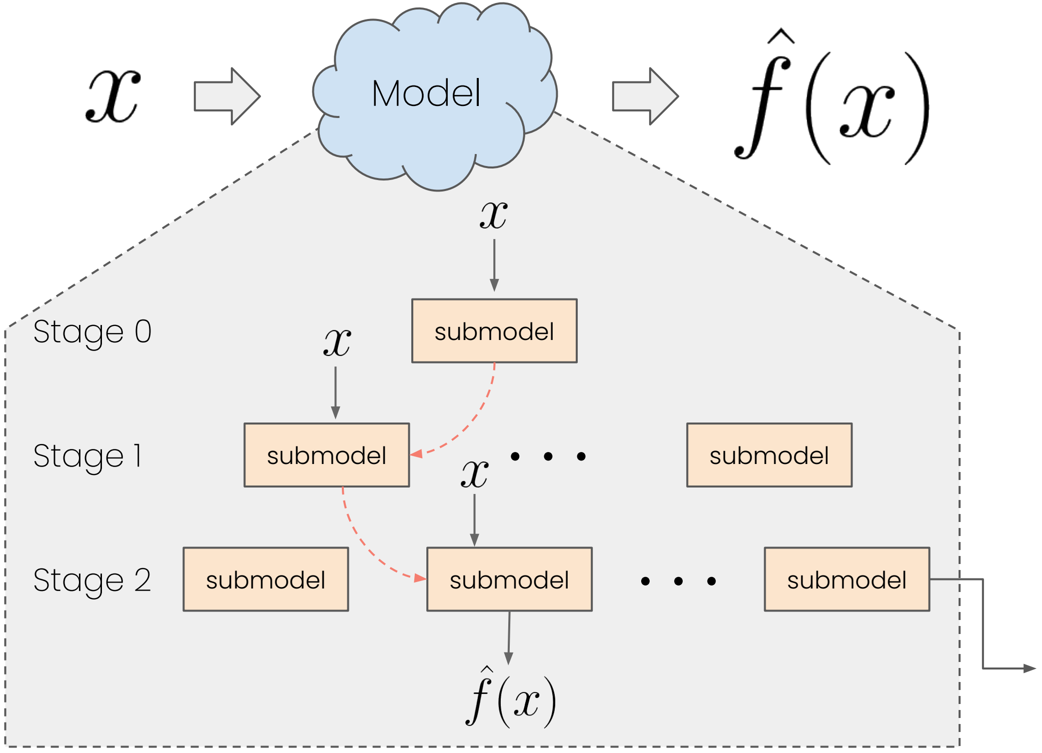


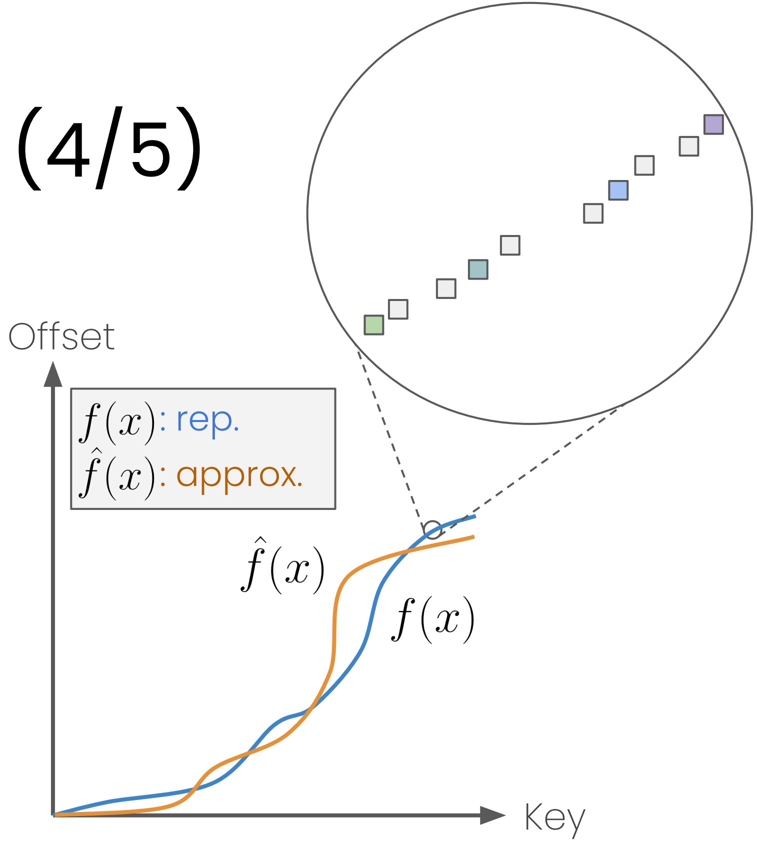
Train model using

$$IndexedKeys = \{ \qquad \}$$

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

Recursive Model Index (RMI) (4/5)

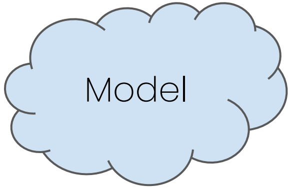




Configurable:
1. Neural Network
2. B-Tree

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

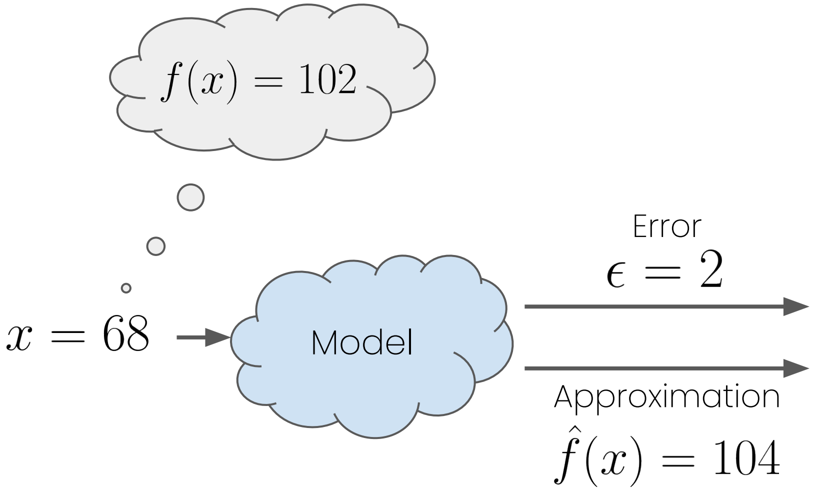
Recursive Model Index (RMI) (5/5)



Mem Offset	Key	Value
101	60	value1
102	68	value2
103	71	value3
104	80	value4
105	87	value5
106	93	value6
107	100	value7

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

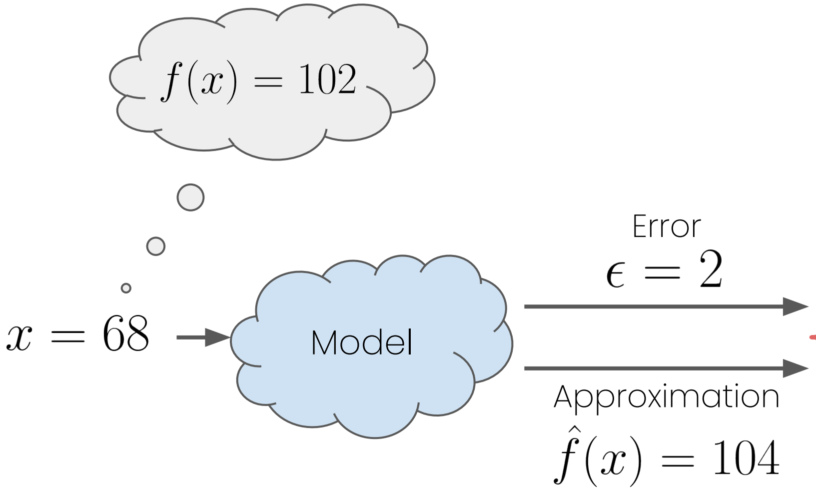
Recursive Model Index (RMI) (5/5)



Mem Offset	Key	Value
101	60	value1
102	68	value2
103	71	value3
104	80	value4
105	87	value5
106	93	value6
107	100	value7

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

Recursive Model Index (RMI) (5/5)



Mem Offset	Key	Value
101	60	value1
102	**68**	**value2**
103	71	value3
104	80	value4
105	87	value5
106	93	value6
107	100	value7

(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

RMI is Not Enough for Packet Classification!

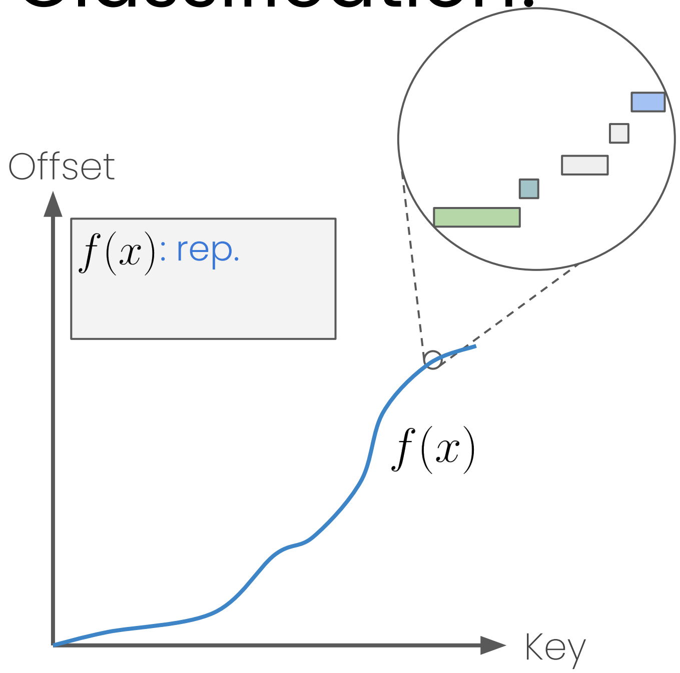
Mem Offset	dst-ip	Action
0	127.0.0.1	value0
1	127.0.0.*	value1
2	8.8.*.*	value2
⋮	⋮	⋮
100	3.*.*.*	value3

RMI does not support wildcards

RMI is Not Enough for Packet Classification!

Mem Offset	dst-ip	Action
0	127.0.0.1	value0
1	127.0.0.*	value1
2	8.8.*.*	value2
.	:	
100	3.*.*.*	value3

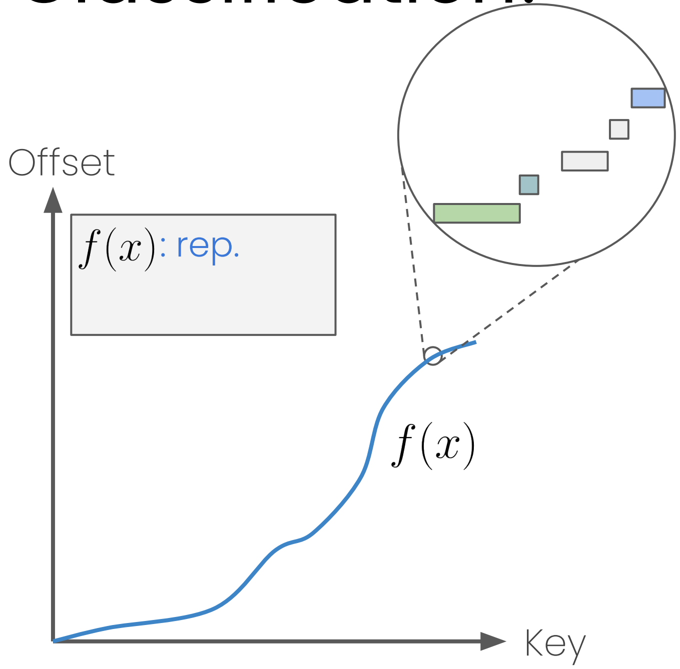
RMI does not support wildcards



RMI is Not Enough for Packet Classification!

Mem Offset	dst-ip	Action
0	127.0.0.1	value0
1	127.0.0.*	value1
2	8.8.*.*	value2
⋮	⋮	⋮
100	3.*.*.*	value3

RMI does not support wildcards



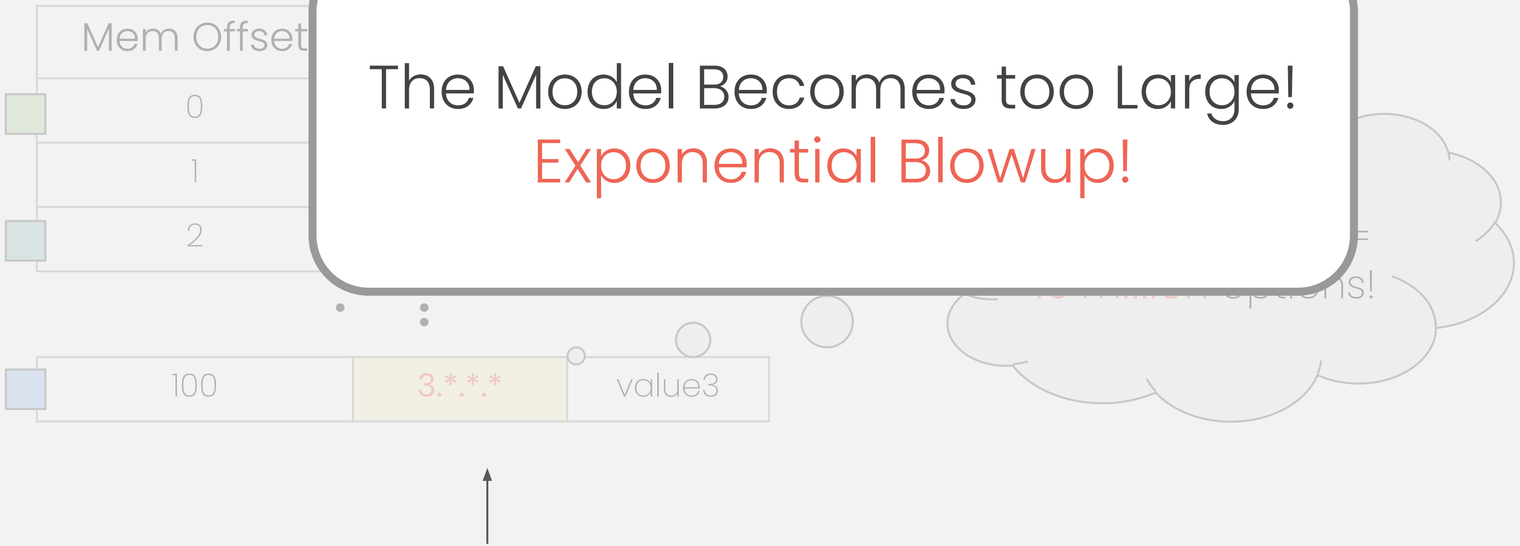
RMI is Not Enough for Packet Classification!

Mem Offset	dst-ip	Action
0	127.0.0.1	value0
1	127.0.0.*	value1
2	8.8.*.*	value2
⋮	⋮	⋮
100	3.*.*.*	value3

RMI does not support wildcards

256 * 256 * 256 =
16 million options!

RMI is Not Enough for Packet Classification!



The Model Becomes too Large!

Exponential Blowup!

RMI does not support wildcards

Our Solution - …

1. Fields contain wildcards

2. Rules may overlap in one or more fields

3. Rules have high dimensionality

Our Solution - Range-Query RMI Models

1. ~~Fields contain wildcards~~ ← Approximate ranges instead of keys

2. Rules may overlap in one or more fields

3. Rules have high dimensionality

Our Solution - Range-Query RMI Models

1. ~~Fields contain wildcards~~ ← Approximate ranges instead of keys

2. ~~Rules may overlap in one or more fields~~

3. ~~Rules have high dimensionality~~

Efficient rule-partitioning

Our Solution - Range-Query RMI Models

1. ~~Fields contain wildcards~~ ← Approximate ranges instead of keys

2. ~~Rules may overlap in one or more fields~~

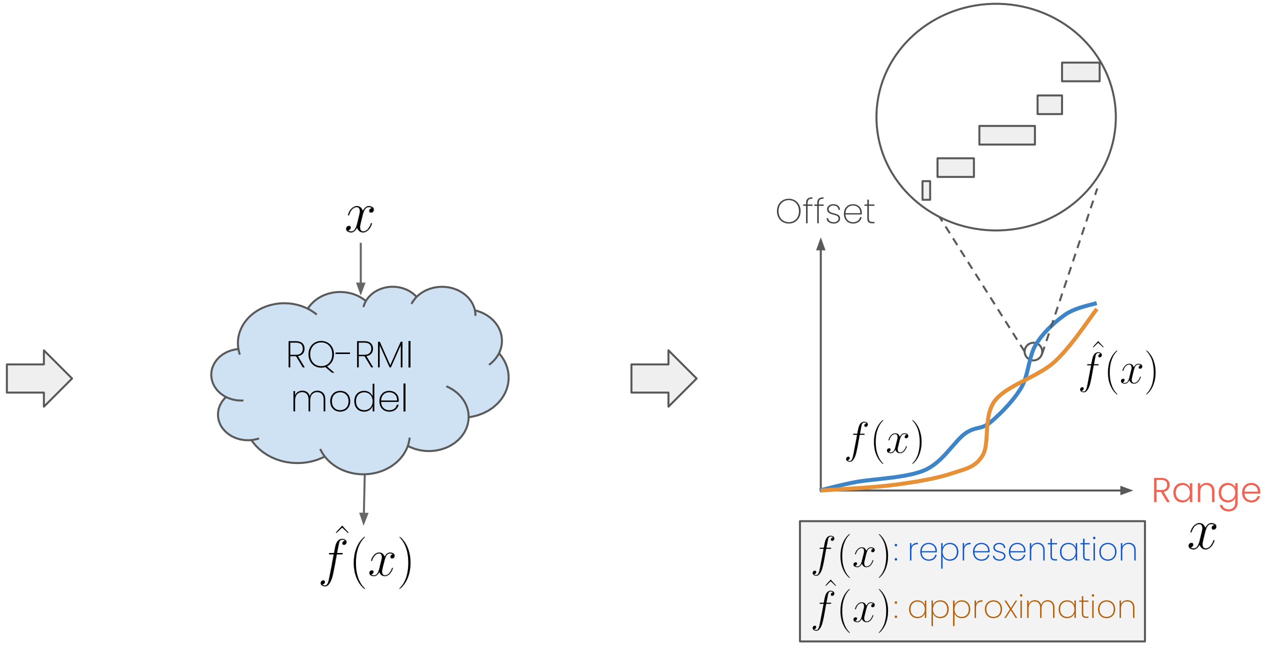
3. ~~Rules have high dimensionality~~

Efficient rule-partitioning

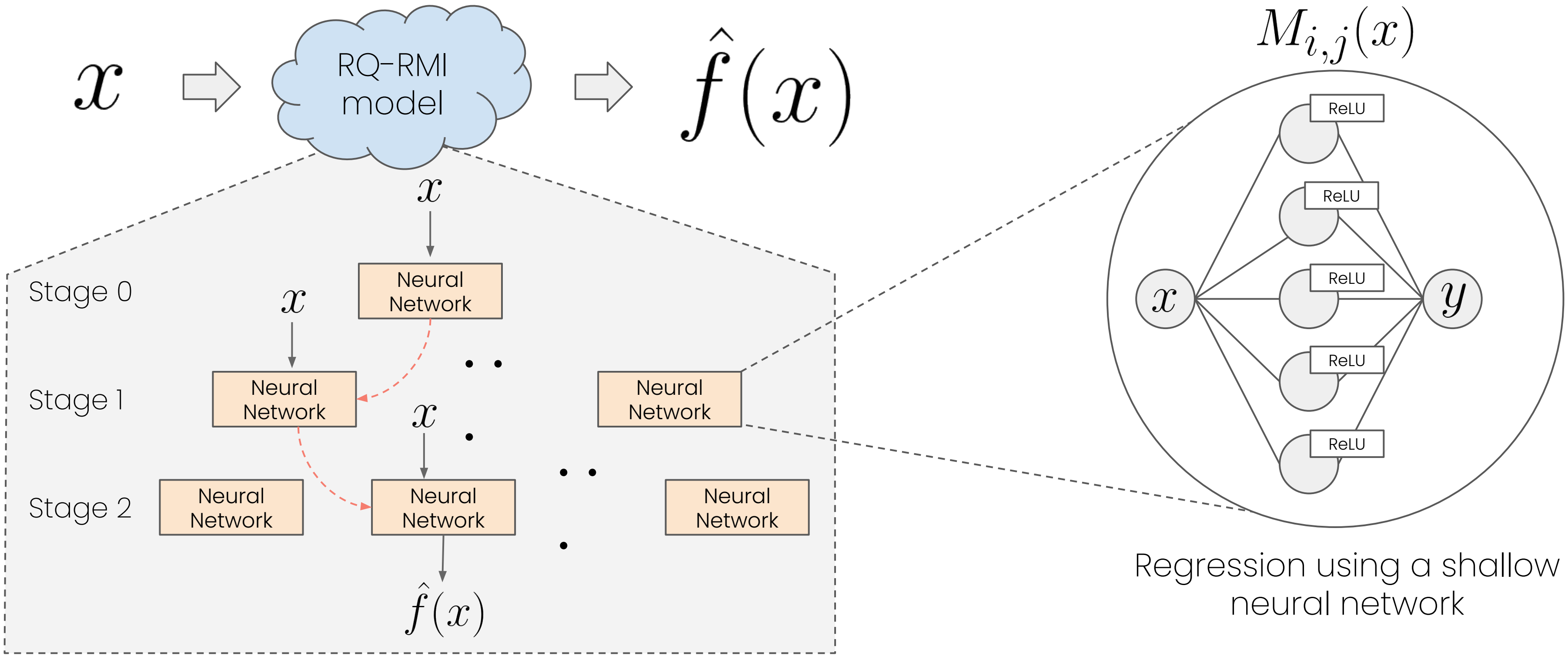
Training and lookup are efficient and fast

Handling Wildcards: Range-Query RMI (1/4)

Range	Value
0-70	600
⋮	⋮
80-104	125

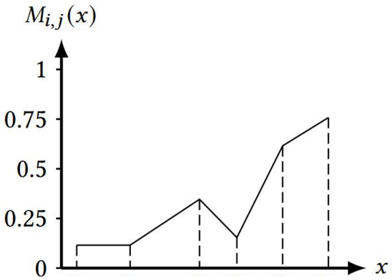


Handling Wildcards: Range-Query RMI (2/4)



Regression using a shallow neural network

Handling Wildcards: Range-Query RMI (3/4)



A bounded approximation error
Is guaranteed

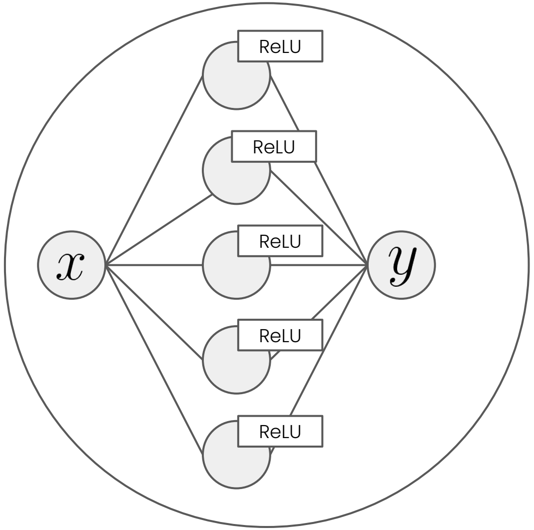
$$|f(x) - \hat{f}(x)| < \epsilon$$

For the entire input domain!

$$\forall x$$

Neural networks with ReLU activations are piecewise linear functions

$M_{i,j}(x)$



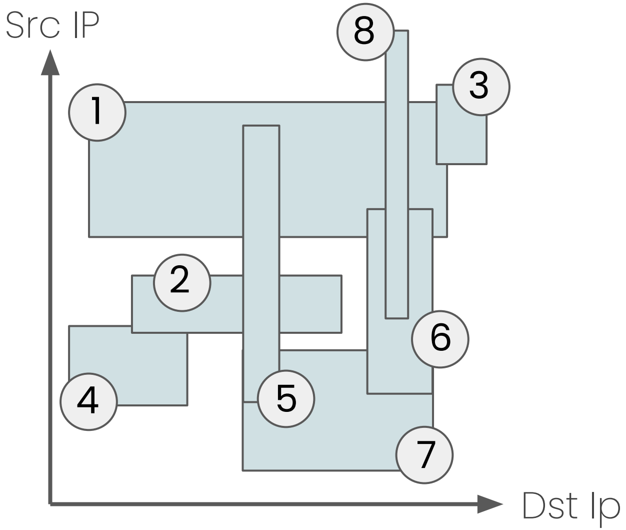
Handling Wildcards: Range-Query RMI (4/4)

1. Enables wildcard matching!
2. Effective training technique!
3. Correctness guarantees!

See paper for details

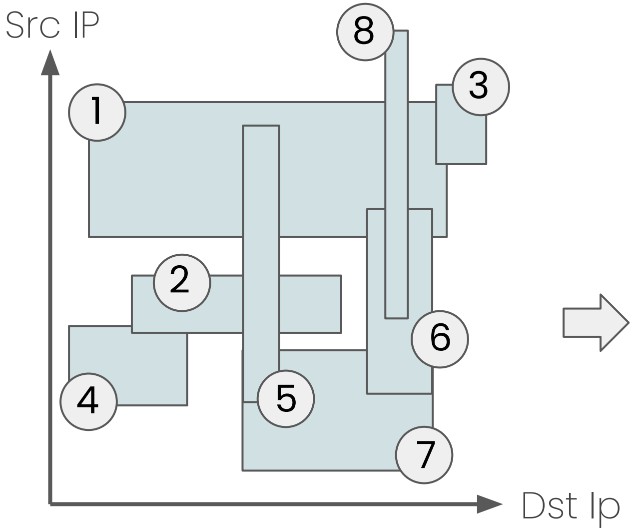
See paper for proofs

Handling Overlaps + Dimensionality: iSets (1/5)

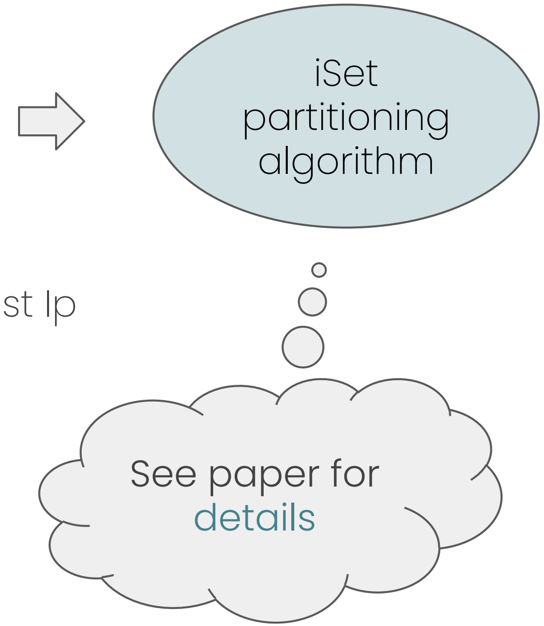


Ruleset geometrical representation

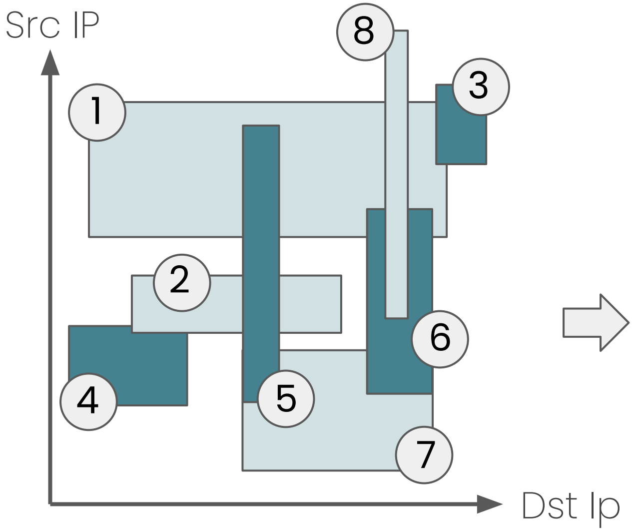
Handling Overlaps + Dimensionality: iSets (1/5)



Ruleset geometrical representation



Handling Overlaps + Dimensionality: iSets (1/5)



Ruleset geometrical representation

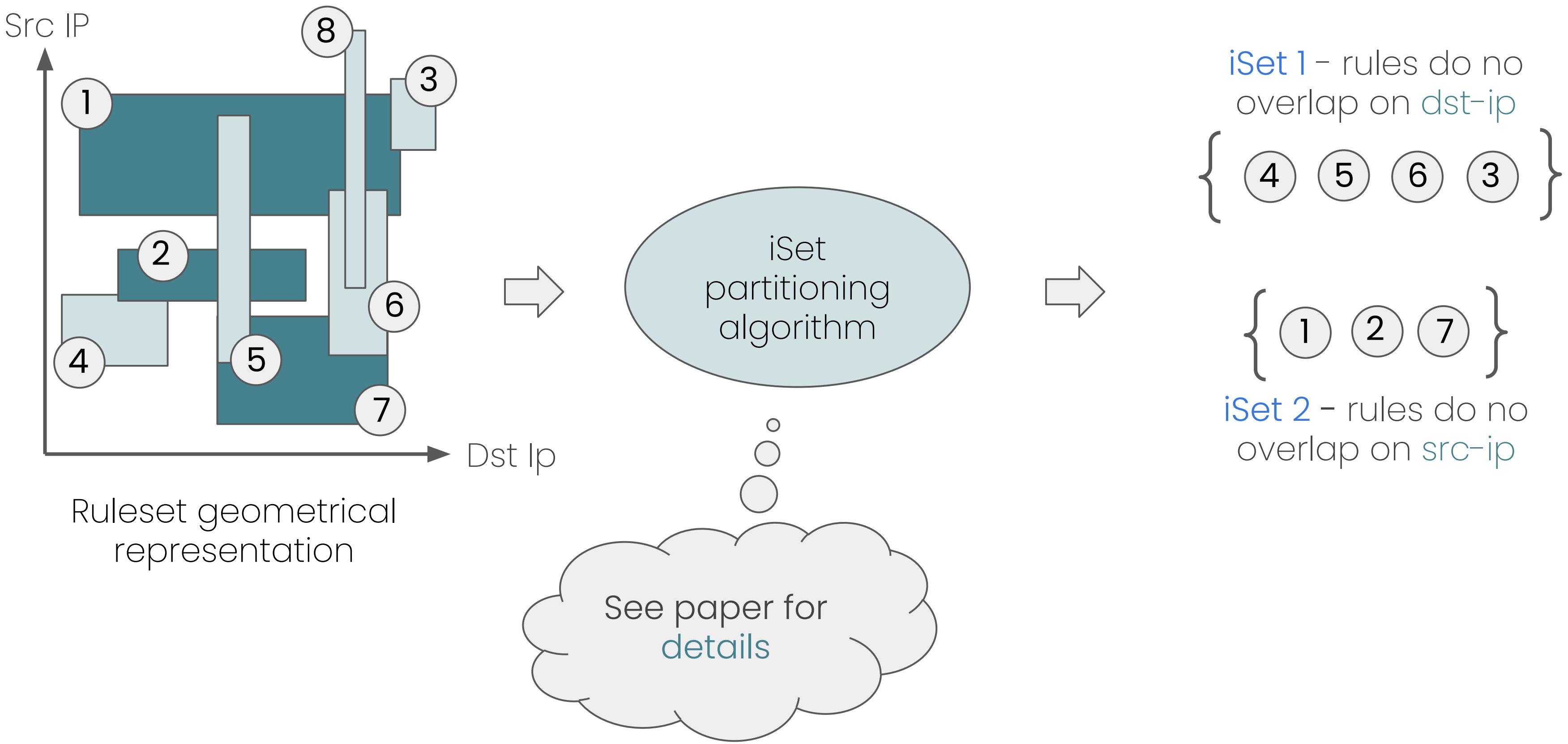
iSet partitioning algorithm

See paper for details

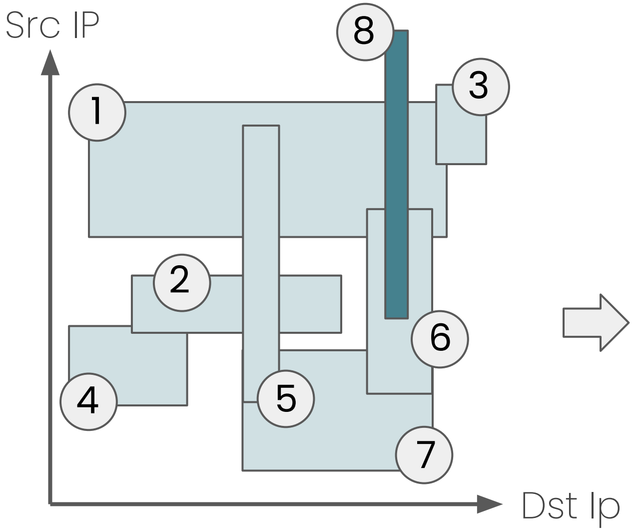
iSet 1 - rules do no overlap on dst-ip

{ 4 5 6 3 }

Handling Overlaps + Dimensionality: iSets (1/5)



Handling Overlaps + Dimensionality: iSets (1/5)



Ruleset geometrical representation

iSet partitioning algorithm

See paper for details

iSet 1 - rules do no overlap on dst-ip

{ 4 5 6 3 }

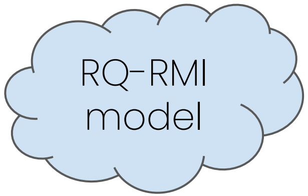
{ 1 2 7 }

iSet 2 - rules do no overlap on src-ip

{ 8 }

Remainder - rules that do not fit in any iSet

Handling Overlaps + Dimensionality: iSets (2/5)



dst-IP	Index within iSet
0-60	1
⋮	⋮
80-670	4

Each iSet can be approximated using an RQ-RMI model

iSet 1 - rules do no overlap on dst-ip

{ 4 5 6 3 }

{ 1 2 7 }

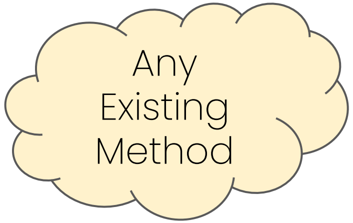
iSet 2 - rules do no overlap on src-ip

{ 8 }

Remainder - rules that do not fit in any iSet

Handling Overlaps + Dimensionality: iSets (3/5)

The remainder set is handled by any classification method



iSet 1 - rules do no overlap on dst-ip

{ 4 5 6 3 }

{ 1 2 7 }

iSet 2 - rules do no overlap on src-ip

{ 8 }

Remainder - rules that do not fit in any iSet

Handling Overlaps + Dimensionality: iSets (4/5)

RQ-RMI solves the problem for a single field.
What about all other fields?

iSet 1 - rules do no overlap on dst-ip

{ 4 5 6 3 }

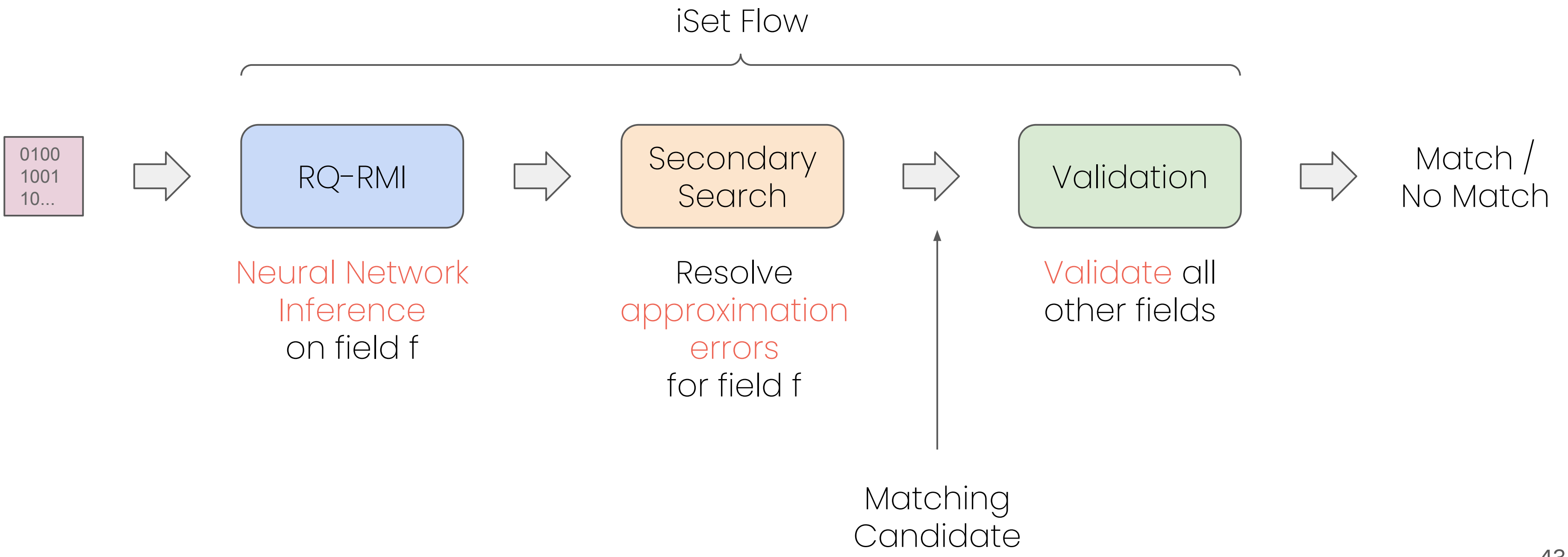
{ 1 2 7 }

iSet 2 - rules do no overlap on src-ip

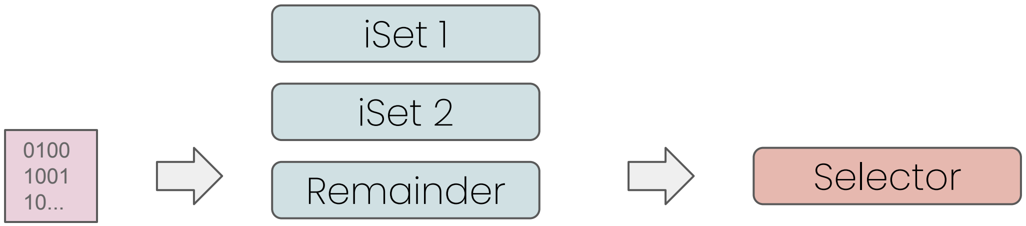
{ 8 }

Remainder - rules that do not fit in any iSet

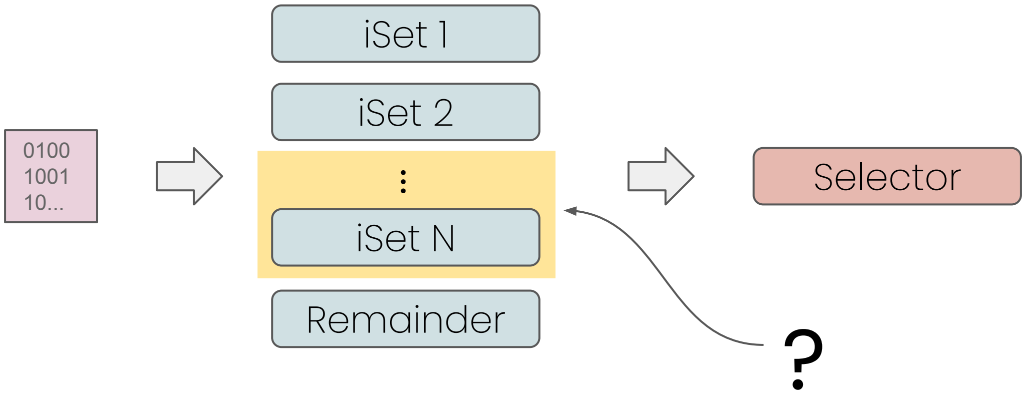
Handling Overlaps + Dimensionality: iSets (4/5)



Handling Overlaps + Dimensionality: iSets (5/5)



Handling Overlaps + Dimensionality: iSets (5/5)



The iSet Tradeoff (1/2)

iSet coverage is high for large rulesets!

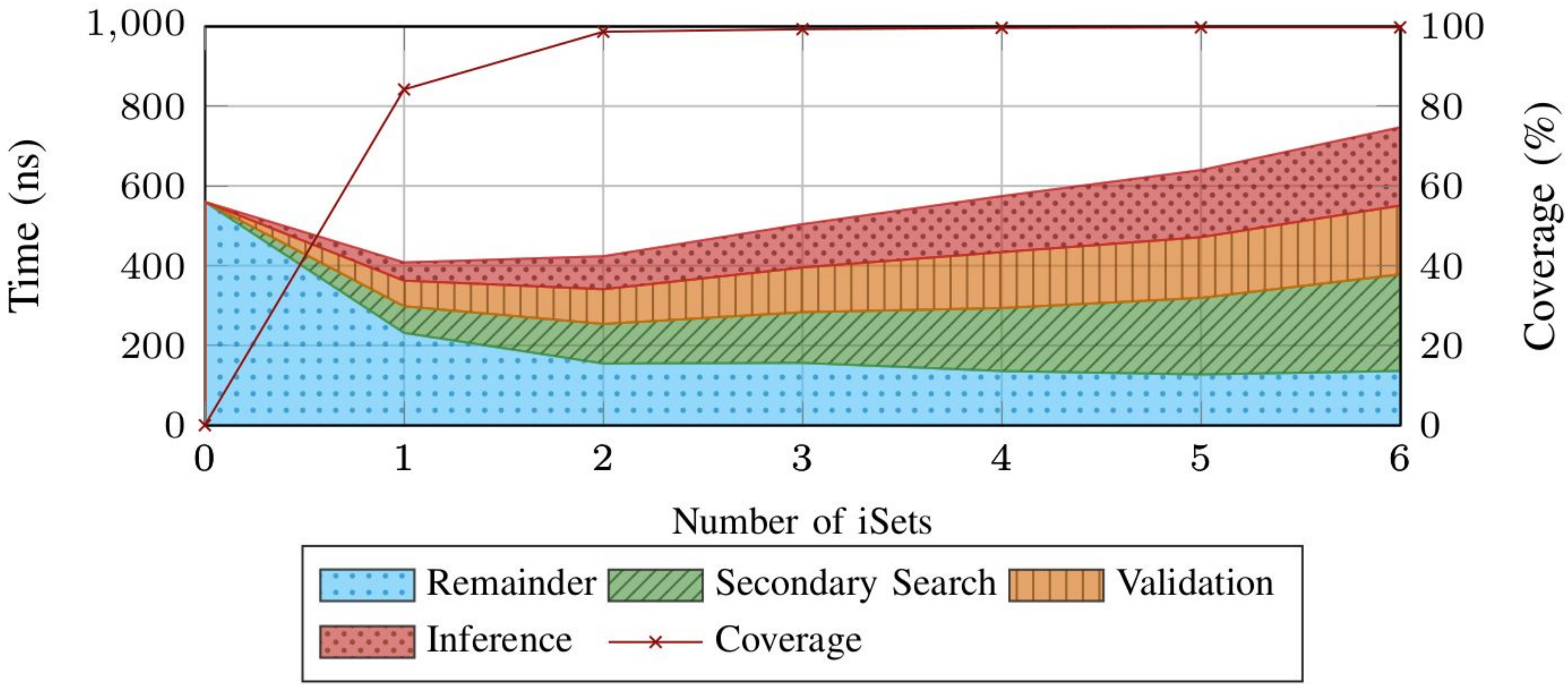
Ruleset Size	1 iSet	2 iSets	3 iSets	4 iSets
1K	20.2 ± 18.6	28.9 ± 22.3	34.6 ± 25.6	38.7 ± 27.2
10K	45.1 ± 31.6	59.6 ± 38.9	62.6 ± 37.1	65.1 ± 35.7
100K	80.0 ± 14.5	96.5 ± 8.3	98.1 ± 4.8	98.8 ± 2.7
500K	84.2 ± 10.5	98.8 ± 1.5	99.4 ± 0.6	99.7 ± 0.2
183,376	57.8	91.6	96.5	98.2

The iSet Tradeoff (1/2)

iSet coverage is high for large rulesets!

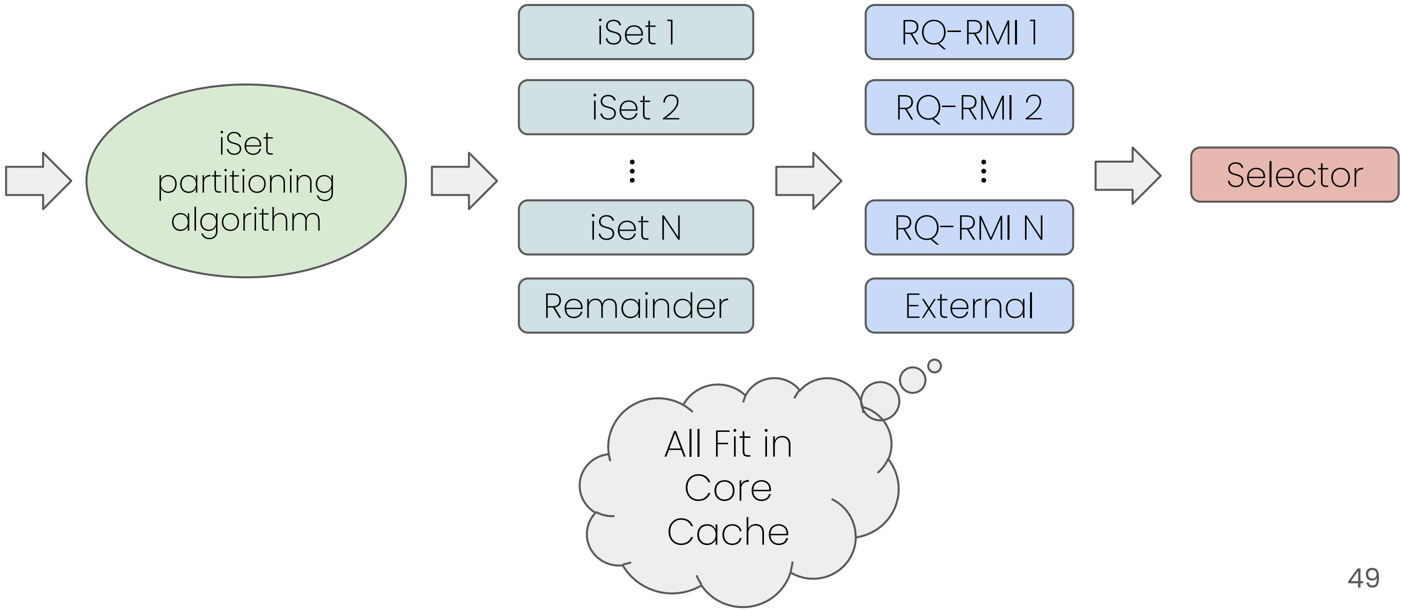
Ruleset Size	1 iSet	2 iSets	3 iSets	4 iSets
1K	20.2 ± 18.6	28.9 ± 22.3	34.6 ± 25.6	38.7 ± 27.2
10K	45.1 ± 31.6	59.6 ± 38.9	62.6 ± 37.1	65.1 ± 35.7
100K	80.0 ± 14.5	96.5 ± 8.3	98.1 ± 4.8	98.8 ± 2.7
500K	84.2 ± 10.5	98.8 ± 1.5	99.4 ± 0.6	99.7 ± 0.2
183,376	57.8	91.6	96.5	98.2

The iSet Tradeoff (2/2)

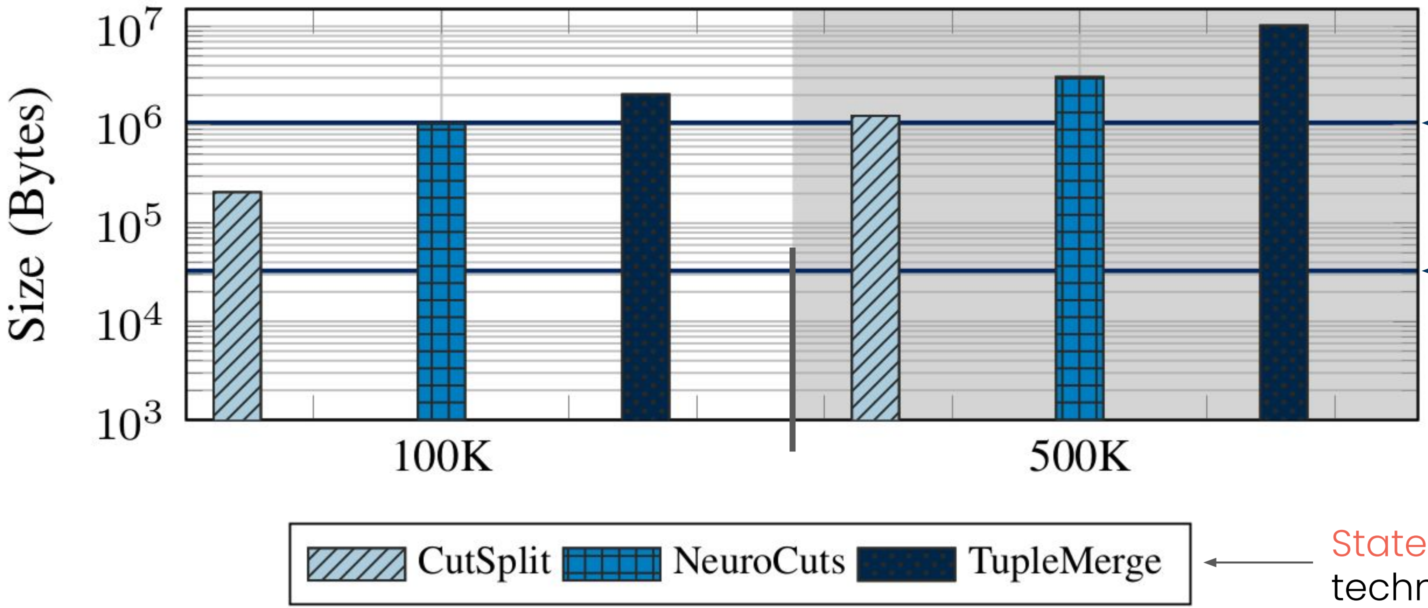


NuevoMatch: Putting it All Together

Src IP	Dst IP	Port
10.0.10.*	8.8.*.*	0-80
10.0.20.*	8.8.7.*	443
10.0.*.*	8.8.7.1	*
55.0.0.*	3.3.*.*	22



Can We Fit in L2 Cache? (1/2)



1024 KB
L2 Cache Size

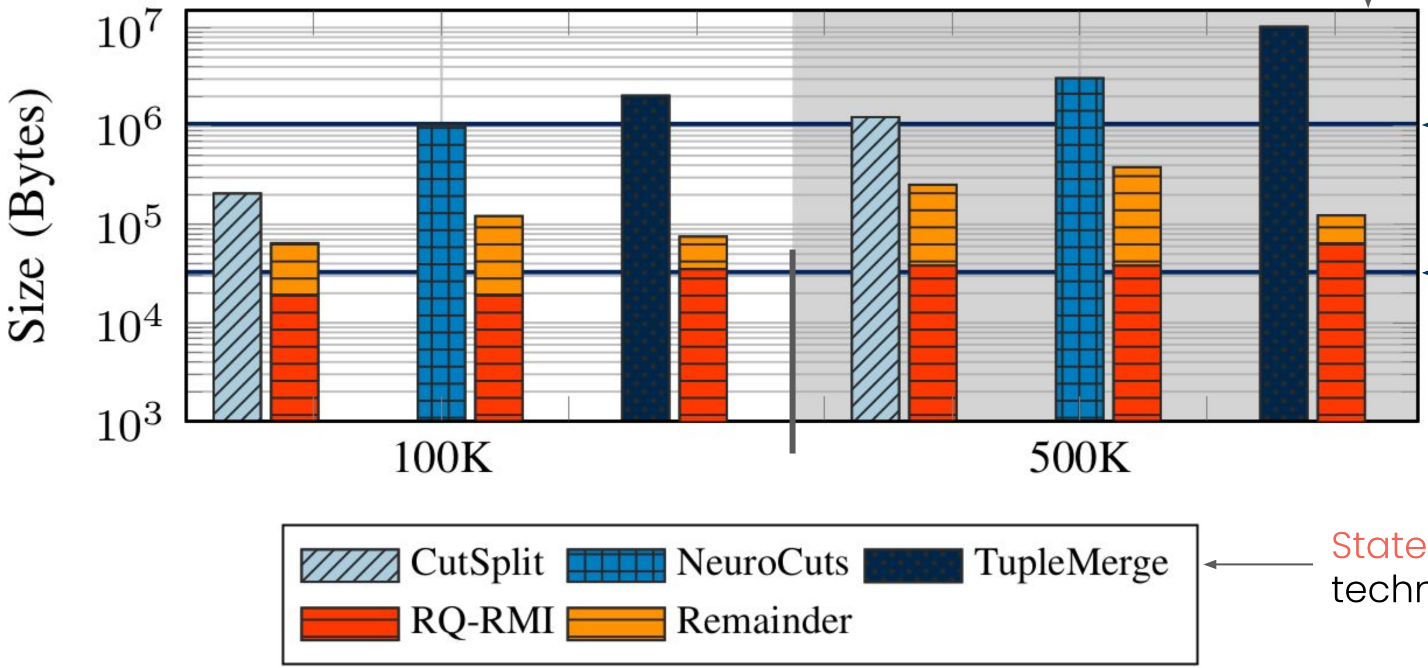
32 KB
L1 Cache Size

State of the art
techniques

(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Liang et al., "Neural Packet Classification". *SIGCOMM 2019.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*

Can We Fit in L2 Cache? (2/2)

Geomean compression factor of 4.9x, 8x, 82x



1024 KB
L2 Cache Size

32 KB
L1 Cache Size

State of the art techniques

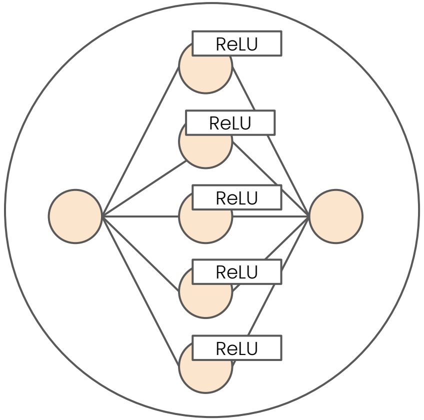
(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Liang et al., "Neural Packet Classification". *SIGCOMM 2019.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*

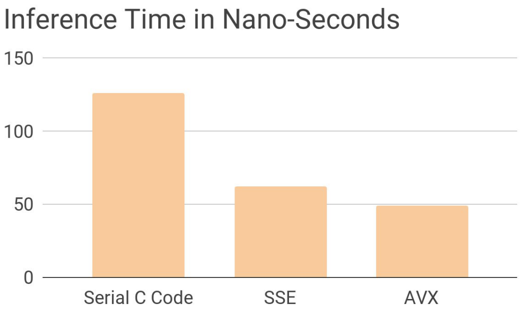
Isn't Neural Network Inference Slow?

Not if you…

Use Small and Shallow Neural Networks

Use Wide Vector Instructions



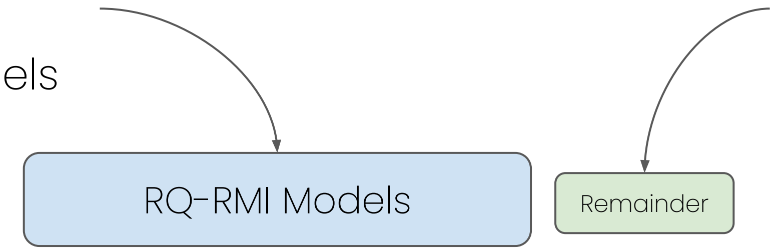


40–50 ns on a CPU!

What About Updates? (1/2)

We easily can remove rules from the models

New rules get into the remainder



We retrain the models after a certain threshold in the number of new rules.

What About Updates? (2/2)

Training with TensorFlow takes between
10-40 min on a CPU
TensorFlow has many overheads

In our ongoing work we currently reach
1-20 seconds on a CPU
using native implementation

Evaluation

End-to-end performance

- Single-core vs. multi-core settings
- Small vs. large rule sets
- Traffic with uniform / skewed temporal locality
- Memory footprint comparison
- Performance under L3 cache contention
- Real-world forwarding rules

Performance analysis

- iSet Coverage
- Impact of the number of iSets
- Partitioning effectiveness
- Training time and search bounds
- Performance with many fields

Evaluation

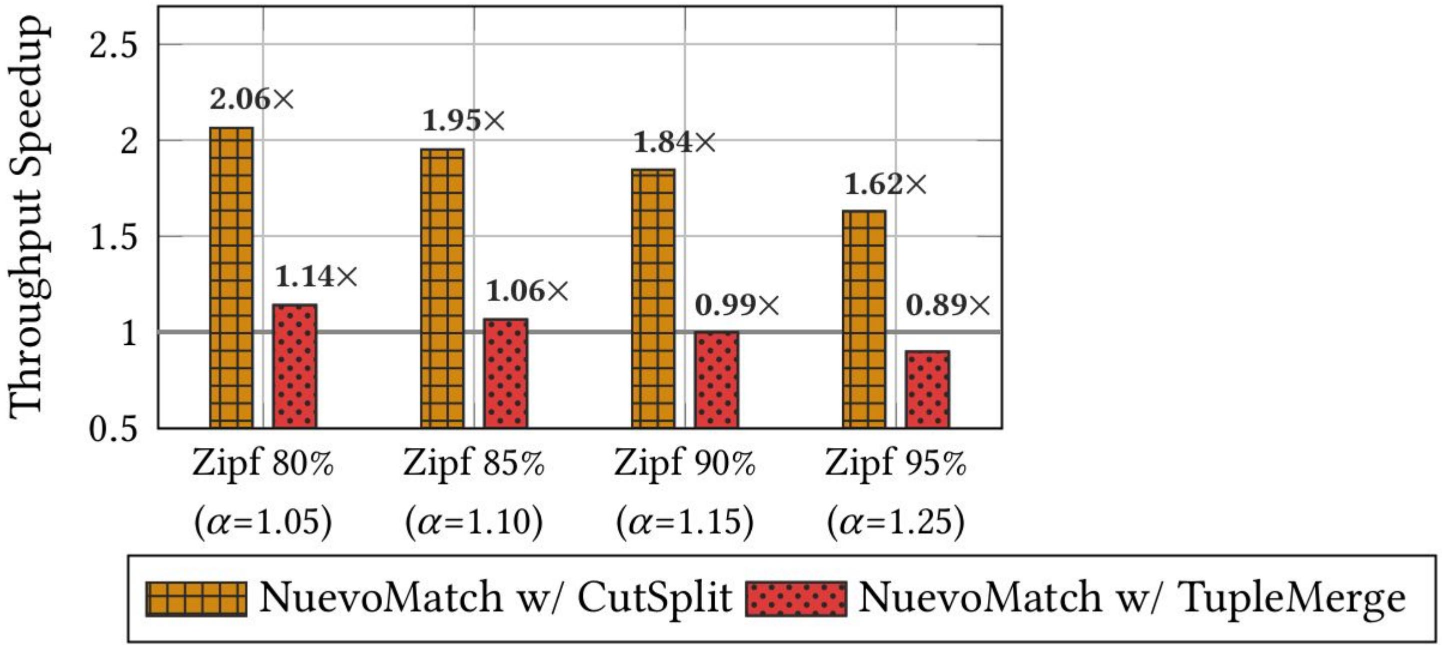
End-to-end performance

- Single-core vs. multi-core settings
- Small vs. large rule sets
- Traffic with uniform / skewed temporal locality
- Memory footprint comparison
- Performance under L3 cache contention
- Real-world forwarding rules

Performance analysis

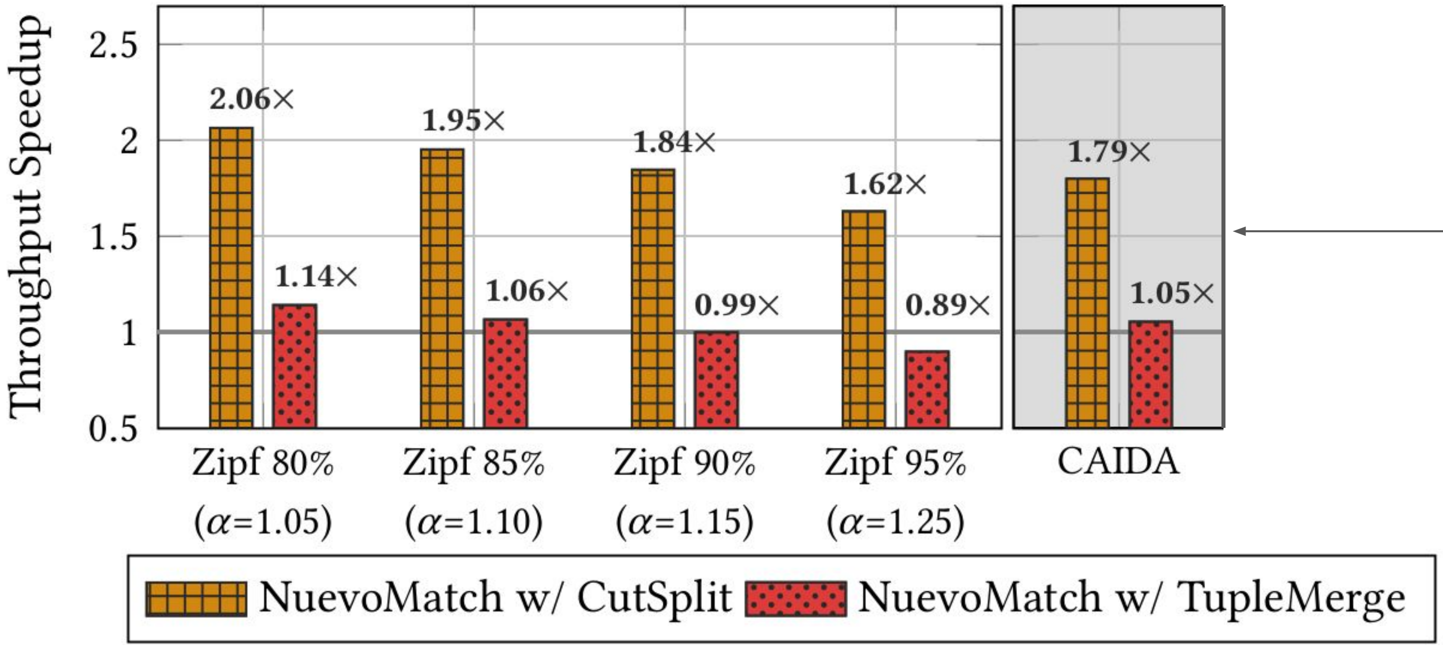
- iSet Coverage
- Impact of the number of iSets
- Partitioning effectiveness
- Training time and search bounds
- Performance with many fields

Evaluation - Skewed Traces



(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Katta et al., “CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks”. *SOSR 2016*.
(*) Taylor et al., “Classbench: A Packet Classification Benchmark”. *TON 2007.*
(*) The CAIDA UCSD Anonymized Internet Traces 2019 (www.caida.org/data/passive/passive_dataset.xml).

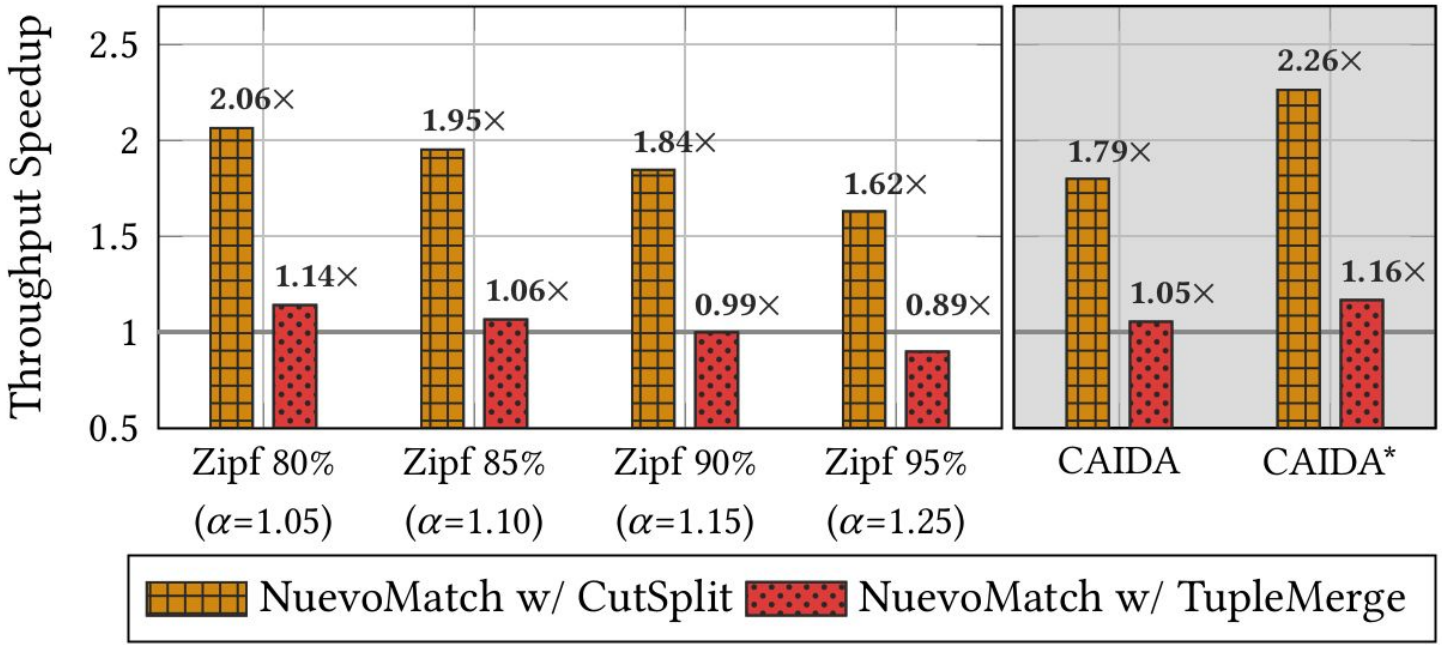
Evaluation - Skewed Traces



Real-world temporal locality

(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Katta et al., "CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks". *SOSR 2016*.
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*
(*) The CAIDA UCSD Anonymized Internet Traces 2019 (www.caida.org/data/passive/passive_dataset.xml).

Evaluation - Skewed Traces

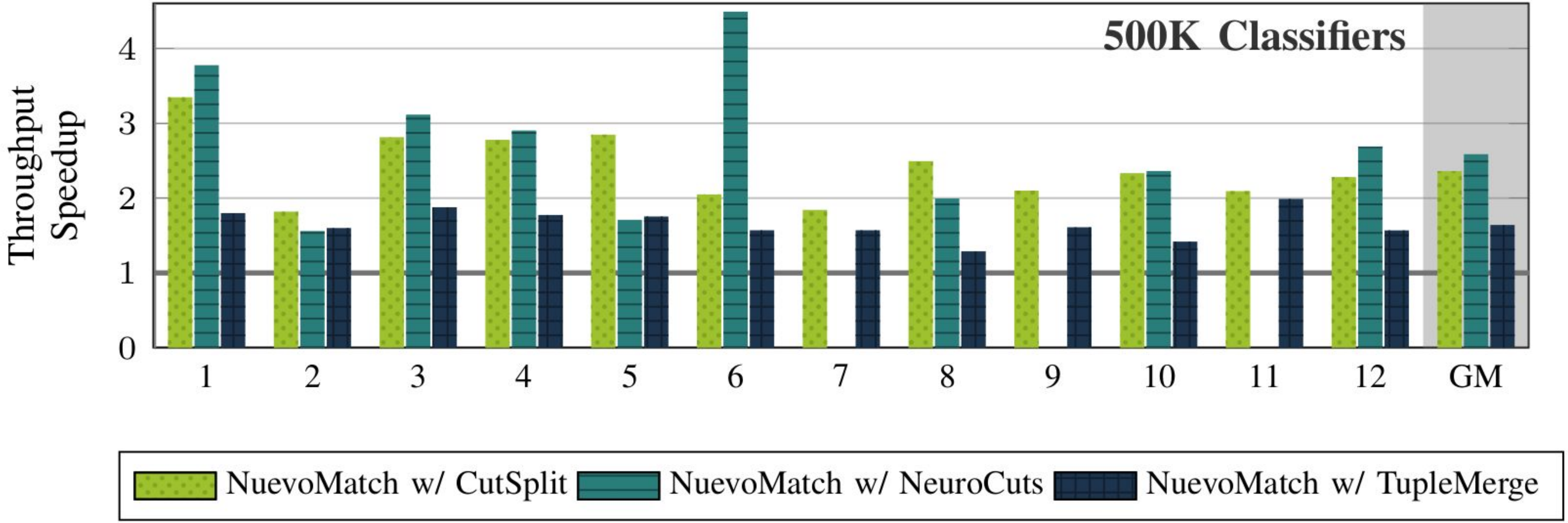


L3 Cache Contention

(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Katta et al., "CacheFlow: Dependency-Aware Rule-Caching for Software-Defined Networks". *SOSR 2016.*
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*
(*) The CAIDA UCSD Anonymized Internet Traces 2019 (www.caida.org/data/passive/passive_dataset.xml).

Evaluation - Uniform Access

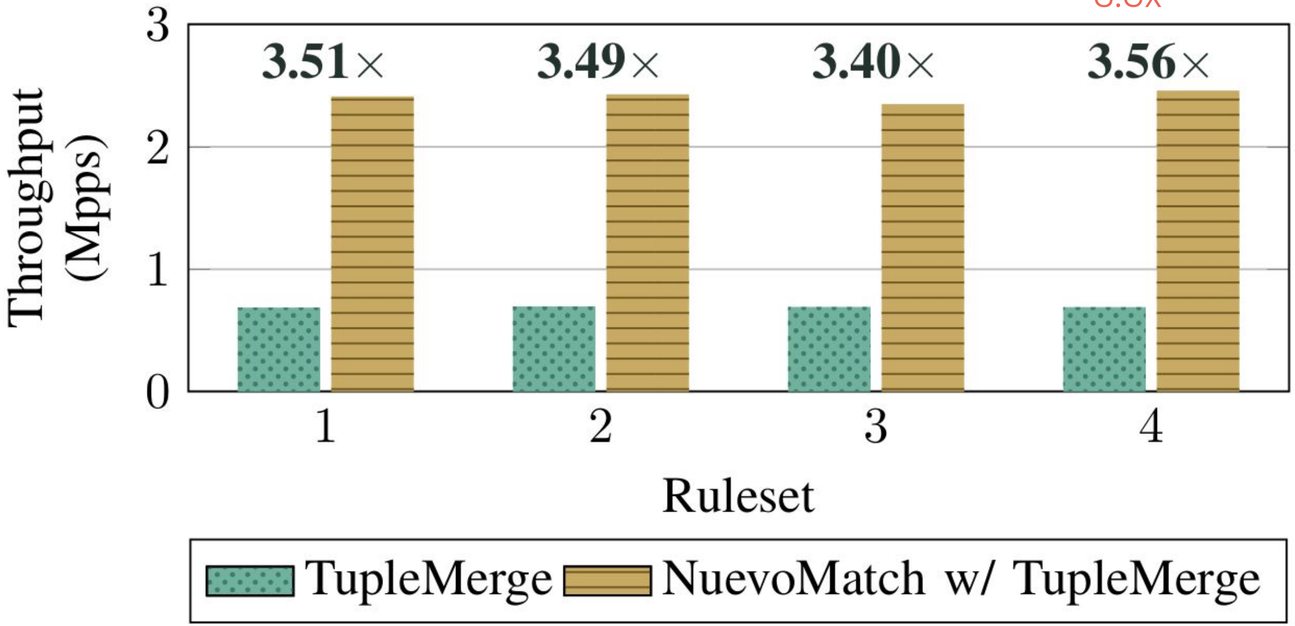
Geomean throughput
speedup of
2.4x, 2.6x, 1.6x



(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Liang et al., "Neural Packet Classification". *SIGCOMM 2019.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Taylor et al., "Classbench: A Packet Classification Benchmark". *TON 2007.*

Evaluation - Real-World Rule Sets

Geomean throughput speedup of 3.5x



(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*
(*) Zong et al., "Automatic Test Packet Generation". *CoNEXT 2012.*

Conclusions

1. NuevoMatch: a new point in the design space of packet classification
2. RQ-RMI for range-value queries
3. Error bound guarantees using piecewise-linear functions

Thank You

See More On
https://acsl.group/publications/

Accelerated Computing Systems Lab

Technion Israel Institute of Technology

Alon Rashelbach
alonrs@campus.technion.ac.il

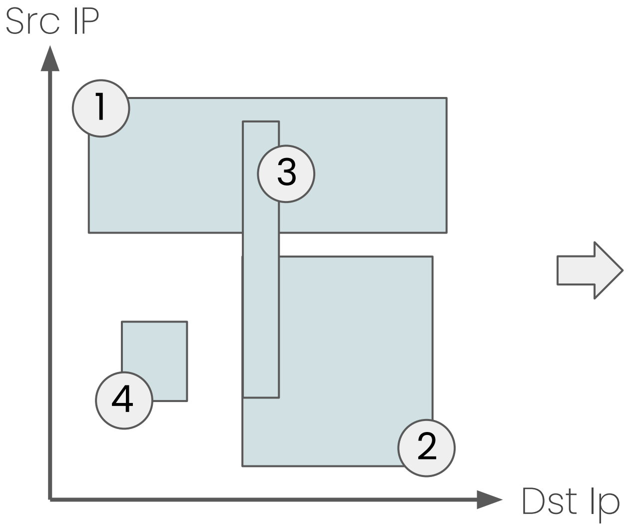
Ori Rottenstreich
or@technion.ac.il

Mark Silberstein
mark@ee.technion.ac.il

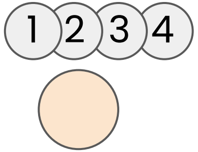
Extra Slides

Decision Tree Approaches (1/3)

	Src IP	Dst IP	Action
1	10.0.10.*	8.8.*.*	Port 1
2	10.0.20.*	8.8.7.*	Port 2
3	10.0.*.*	8.8.7.1	Drop
4	55.0.0.*	3.3.*.*	Port 1



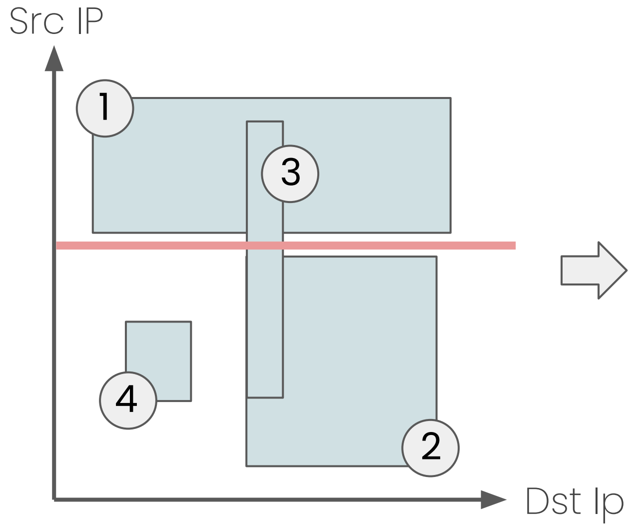
Ruleset geometrical representation



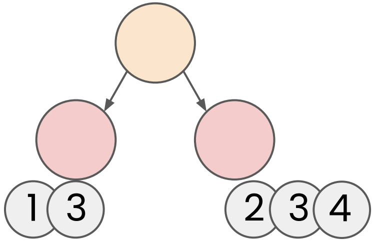
(*) Gupta et al., “Classifying Packets with Hierarchical Intelligent Cuttings”. *IEEE Micro 2000.*
(*) Singh et al., "Packet Classification Using Multidimensional Cutting". *SIGCOMM 2003.*
(*) Vamanan et al., "EffiCuts: Optimizing Packet Classification for Memory and Throughput". *SIGCOMM 2010.*
(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Liang et al., "Neural Packet Classification". *SIGCOMM 2019.*

Decision Tree Approaches (2/3)

	Src IP	Dst IP	Action
1	10.0.10.*	8.8.*.*	Port 1
2	10.0.20.*	8.8.7.*	Port 2
3	10.0.*.*	8.8.7.1	Drop
4	55.0.0.*	3.3.*.*	Port 1



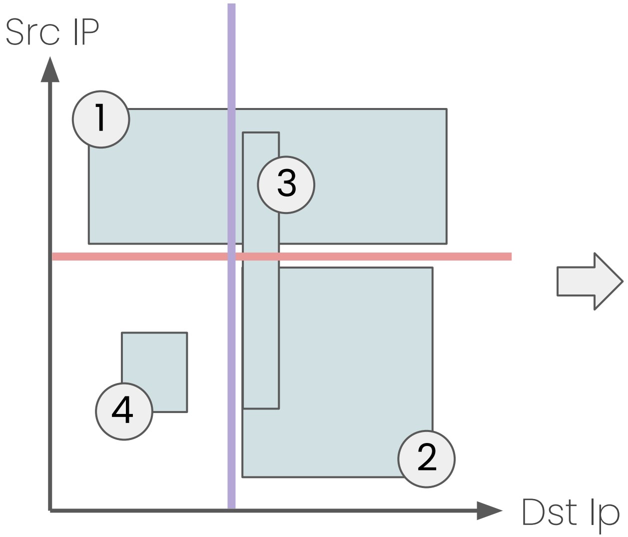
Ruleset geometrical representation



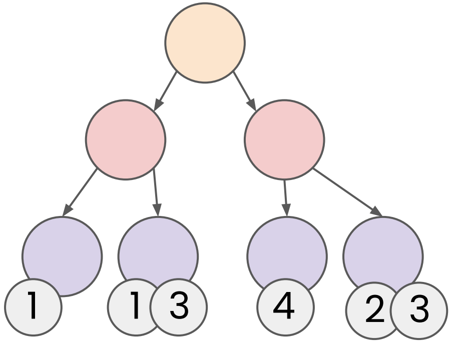
(*) Gupta et al., "Classifying Packets with Hierarchical Intelligent Cuttings". *IEEE Micro 2000.*
(*) Singh et al., "Packet Classification Using Multidimensional Cutting". *SIGCOMM 2003.*
(*) Vamanan et al., "EffiCuts: Optimizing Packet Classification for Memory and Throughput". *SIGCOMM 2010.*
(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Liang et al., "Neural Packet Classification". *SIGCOMM 2019.*

Decision Tree Approaches (3/3)

	Src IP	Dst IP	Action
1	10.0.10.*	8.8.*.*	Port 1
2	10.0.20.*	8.8.7.*	Port 2
3	10.0.*.*	8.8.7.1	Drop
4	55.0.0.*	3.3.*.*	Port 1



Ruleset geometrical representation



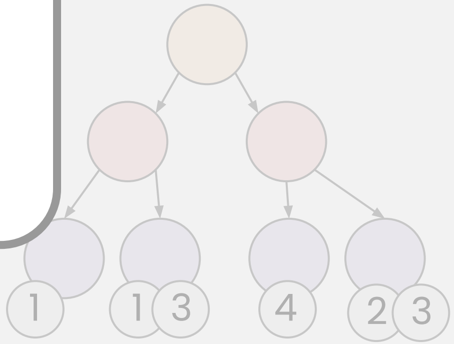
(*) Gupta et al., “Classifying Packets with Hierarchical Intelligent Cuttings”. *IEEE Micro 2000.*
(*) Singh et al., "Packet Classification Using Multidimensional Cutting". *SIGCOMM 2003.*
(*) Vamanan et al., "EffiCuts: Optimizing Packet Classification for Memory and Throughput". *SIGCOMM 2010.*
(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Liang et al., "Neural Packet Classification". *SIGCOMM 2019.*

Decision Tree Approaches (3/3)

	Src IP	D	
1	10.0.10.*	8	
2	10.0.20.*	8	
3	10.0.*.*	8.8.7.1	Drop
4	55.0.0.*	3.3.*.*	Port 1

Rule replication causes memory growth!

Ruleset geometrical representation



(*) Gupta et al., “Classifying Packets with Hierarchical Intelligent Cuttings”. *IEEE Micro 2000.*
(*) Singh et al., "Packet Classification Using Multidimensional Cutting". *SIGCOMM 2003.*
(*) Vamanan et al., "EffiCuts: Optimizing Packet Classification for Memory and Throughput". *SIGCOMM 2010.*
(*) Li et al., "CutSplit: A Decision Tree Combining Cutting and Splitting for Scalable Packet Classification". *INFOCOM 2018.*
(*) Liang et al., "Neural Packet Classification". *SIGCOMM 2019.*

Hash-Table Approaches (1/3)

	Src IP	Dst IP	Action
1	10.0.10.*	8.8.*.*	Port 1
2	10.0.20.*	8.8.7.*	Port 2
3	10.0.*.*	8.8.7.1	Drop
4	55.0.0.*	3.3.*.*	Port 1

(*) Srinivasan et al., "Packet Classification Using Tuple Space Search". *SIGCOMM 1999.*
(*) Pfaff et al., "The Design and Implementation of Open vSwitch". *NSDI 2015.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*

Hash-Table Approaches (2/3)

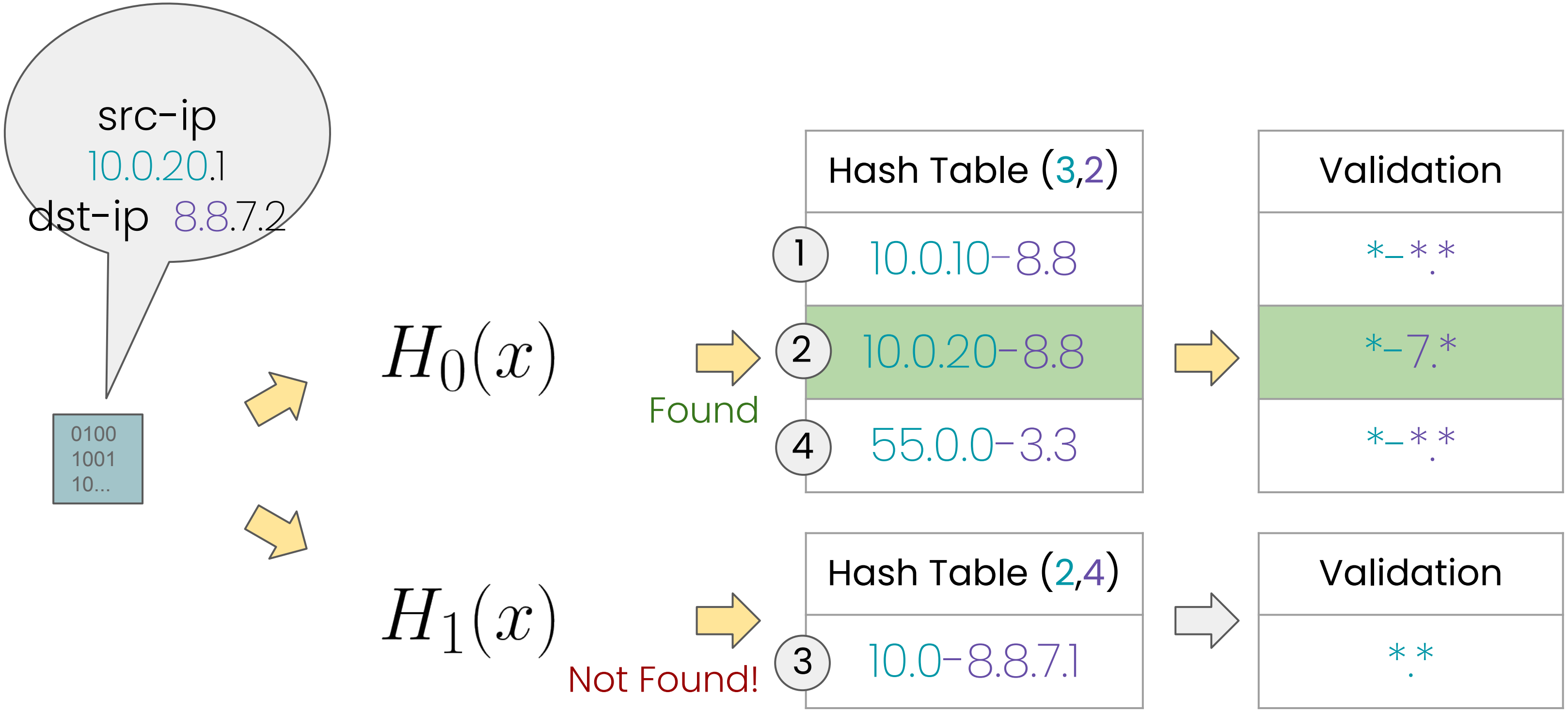
	Src IP	Dst IP	Action
1	10.0.10.*	8.8.*.*	Port 1
2	10.0.20.*	8.8.7.*	Port 2
3	10.0.*.*	8.8.7.1	Drop
4	55.0.0.*	3.3.*.*	Port 1

	Hash Table (3,2)	Validation
1	10.0.10–8.8	*–*.*
2	10.0.20–8.8	*–7.*
4	55.0.0–3.3	*–*.*

	Hash Table (2,4)	Validation
3	10.0–8.8.7.1	*.*

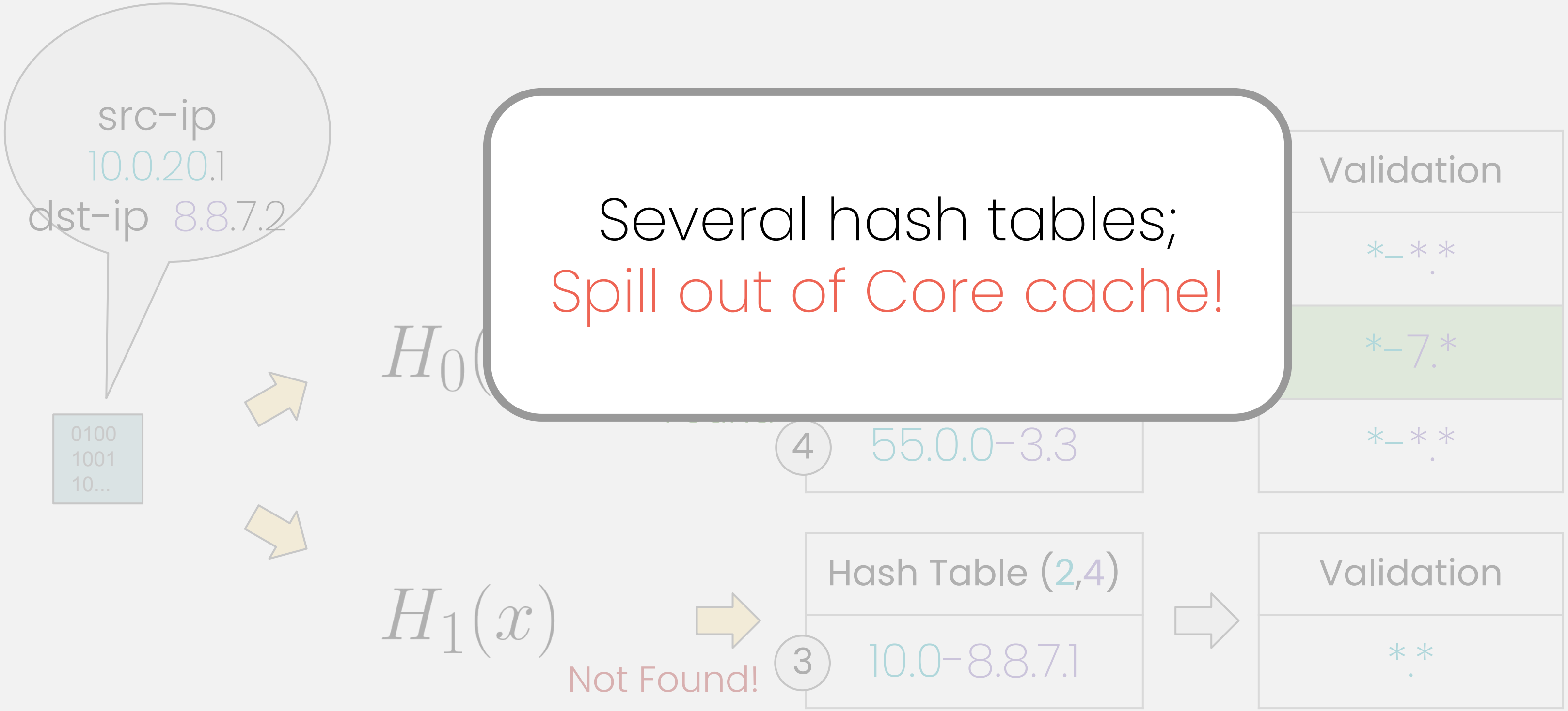
(*) Srinivasan et al., "Packet Classification Using Tuple Space Search". *SIGCOMM 1999.*
(*) Pfaff et al., "The Design and Implementation of Open vSwitch". *NSDI 2015.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*

Hash-Table Approaches (3/3)



(*) Srinivasan et al., "Packet Classification Using Tuple Space Search". *SIGCOMM 1999.*
(*) Pfaff et al., "The Design and Implementation of Open vSwitch". *NSDI 2015.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*

Hash-Table Approaches (3/3)

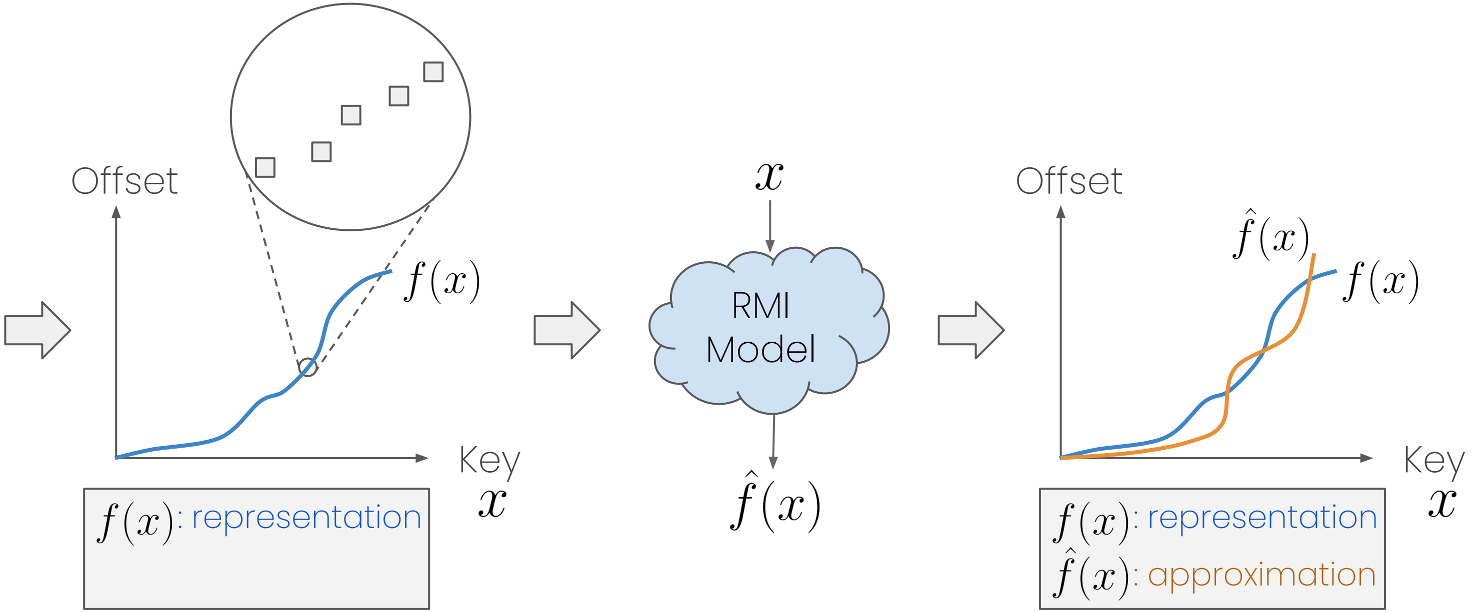


Several hash tables;
Spill out of Core cache!

(*) Srinivasan et al., "Packet Classification Using Tuple Space Search". *SIGCOMM 1999.*
(*) Pfaff et al., "The Design and Implementation of Open vSwitch". *NSDI 2015.*
(*) Daly et al., "TupleMerge: Fast Software Packet Processing for online Packet Classification". *TON 2019.*

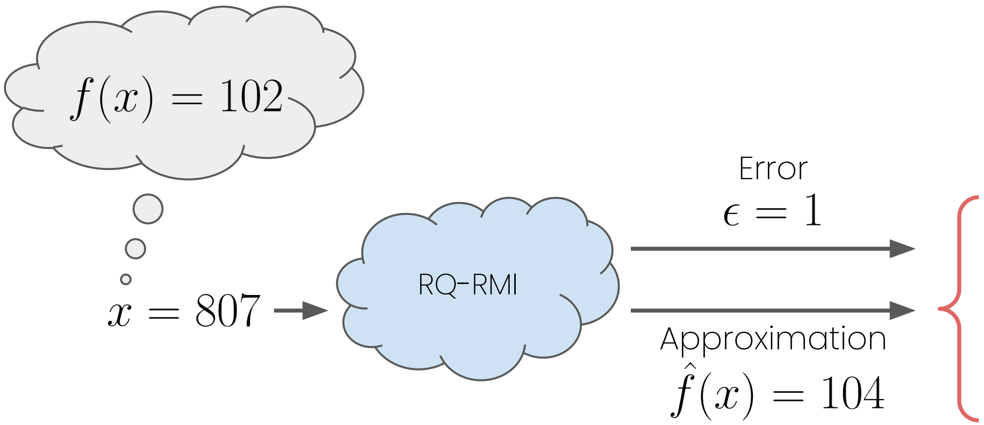
Recursive Model Index (RMI)

Key	Value
56	600
⋮	⋮
104	125



(*) Kraska et al., "The Case for Learned Index Structures". *SIGMOD 2018.*

Handling Wildcards: Range-Query RMI (4/4)

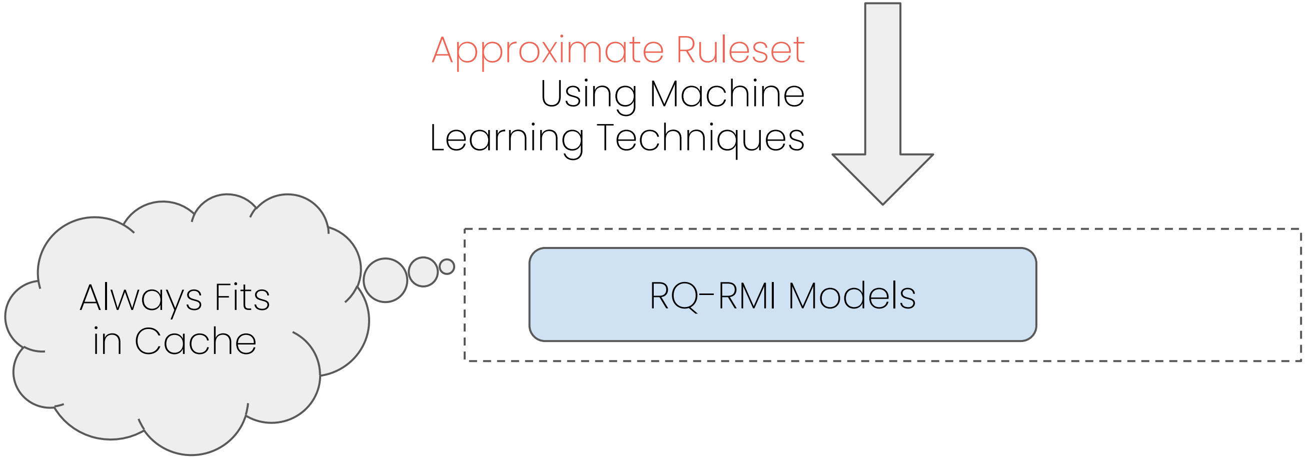


Mem Offset	Range
98	0-100
100	130-700
102	**800-801**
104	807-890
106	970-990
108	991-1000

Introducing NuevoMatch (1/4)

Rule Set

Src IP	Dst IP	Src Port	Dst Port	Action
10.0.10.*	8.8.*.*	0-65535	80	Port 1
10.0.*.*	8.8.7.1	0-65535	0-6	Drop



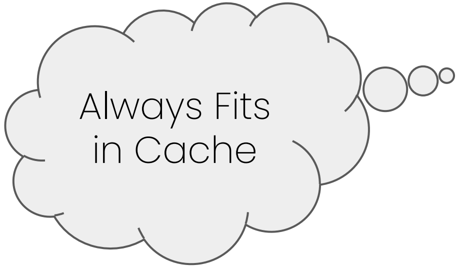
Introducing NuevoMatch (2/4)

Rule Set

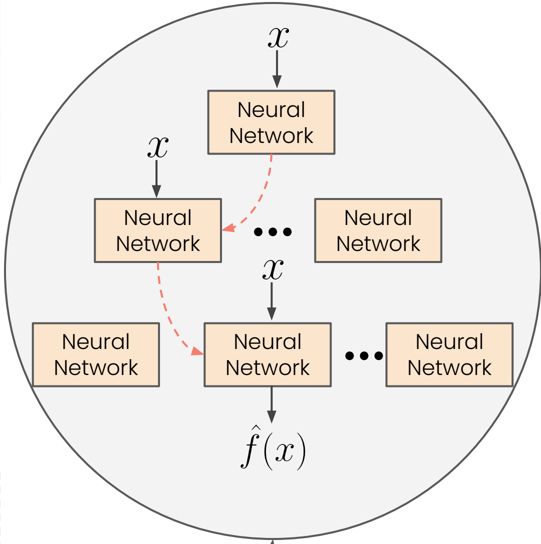
Src IP	Dst IP	Src Port	Dst Port	Action
10.0.10.*	8.8.*.*	0-65535	80	Port 1
10.0.*.*	8.8.7.1	0-65535	0-6	Drop

Approximate Ruleset
Using Machine
Learning Techniques

RQ-RMI Models



RQ-RMI model

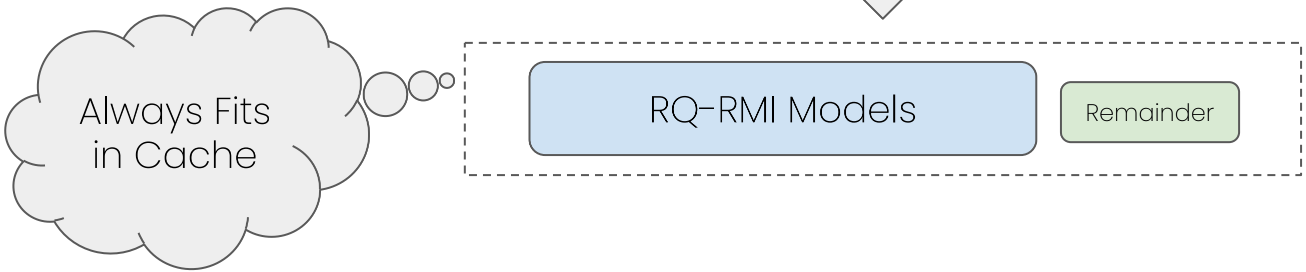


Introducing NuevoMatch (3/4)

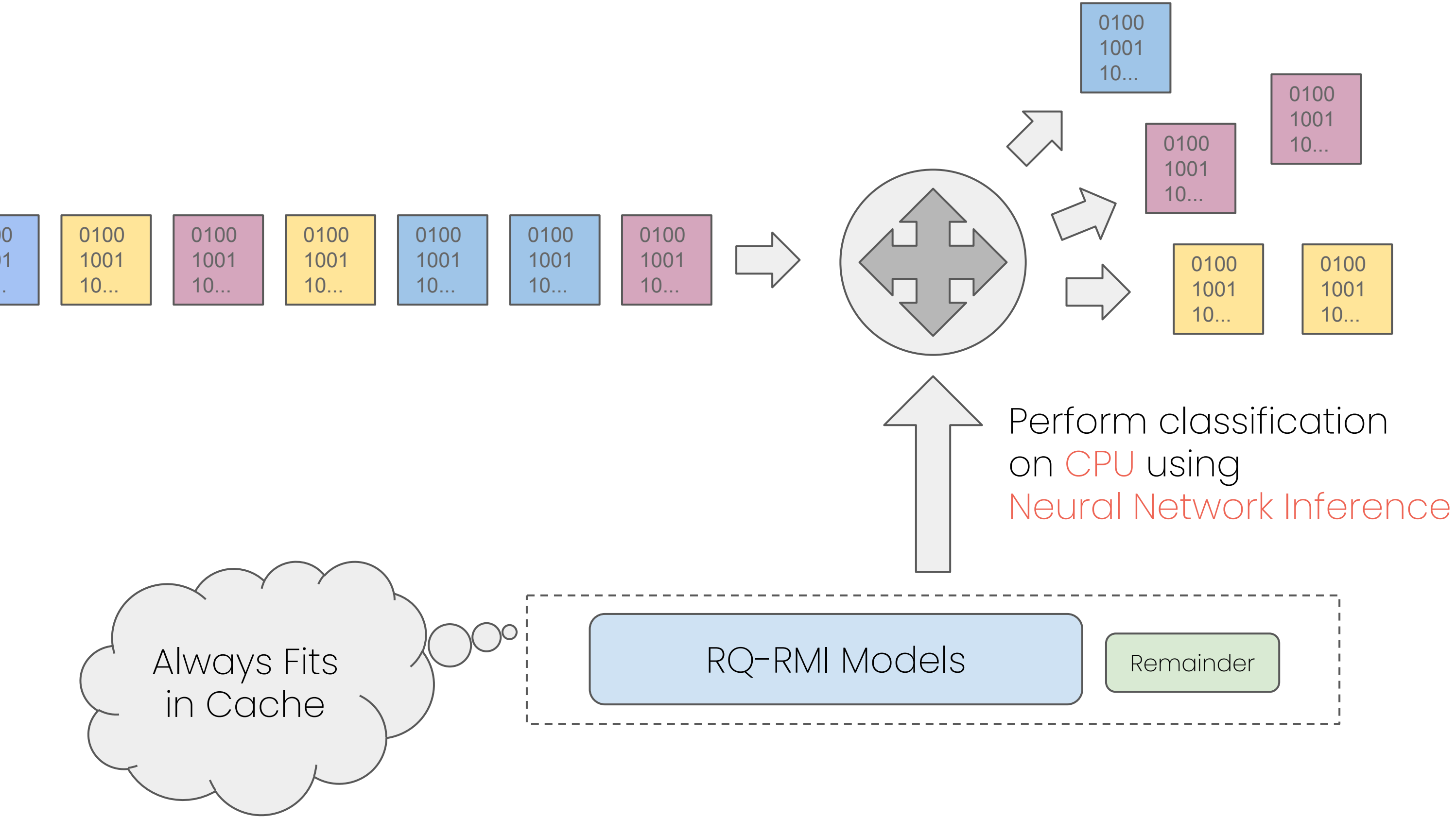
Rule Set

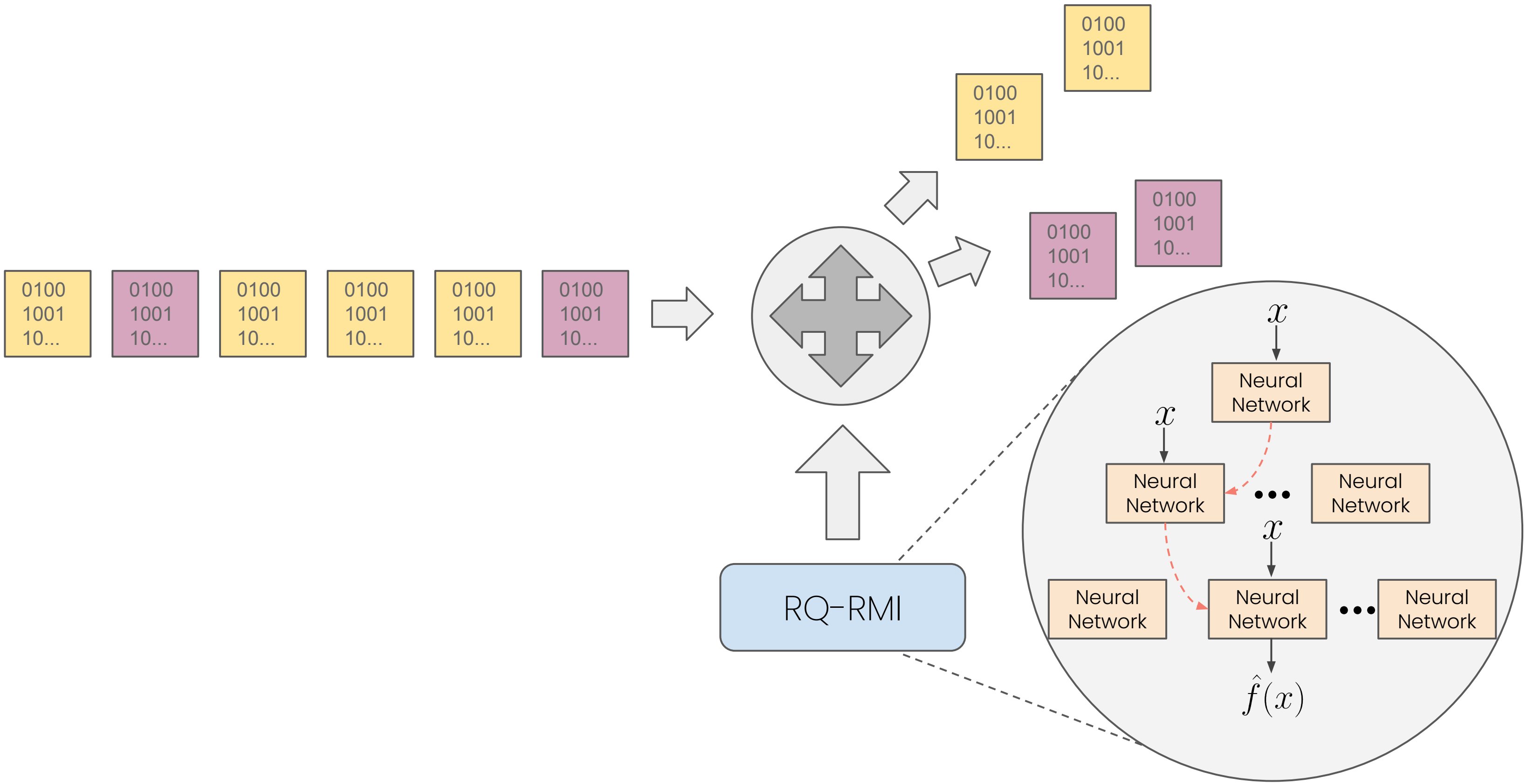
Src IP	Dst IP	Src Port	Dst Port	Action
10.0.10.*	8.8.*.*	0-65535	80	Port 1
10.0.*.*	8.8.7.1	0-65535	0-6	Drop

Some part cannot be approximated, so we keep it as a remainder



Introducing NuevoMatch (4/4)




0100
1001
10...
RQ-RMI
Neural Network
x
$\hat{f}(x)$