Hyperscan Progress on ICX and Data Analytics Use Case

Yang Hong

HYPERSCAN.IO

intel®

Legal Disclaimer

General Disclaimer:

© Copyright 2015 Intel Corporation. All rights reserved. Intel, the Intel logo, Intel Inside, the Intel Inside logo, Intel. Experience What's Inside are trademarks of Intel. Corporation in the U.S. and/or other countries. *Other names and brands may be claimed as the property of others.

Technology Disclaimer:

Intel technologies' features and benefits depend on system configuration and may require enabled hardware, software or service activation. Performance varies depending on system configuration. No computer system can be absolutely secure. Check with your system manufacturer or retailer or learn more at [intel.com].

Performance Disclaimers:

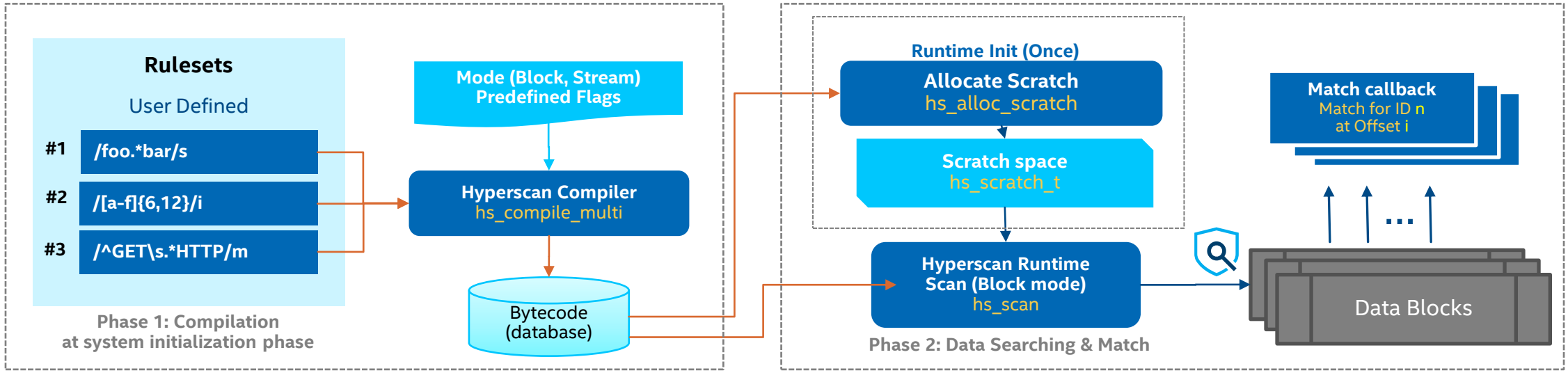
Results have been estimated or simulated using internal Intel analysis or architecture simulation or modeling, and provided to you for informational purposes. Any differences in your system hardware, software or configuration may affect your actual performance.

Agenda

- Overview
- 5.3/5.4 Release update
- ICX Platform Benchmark
- 5.5 Features and Future
- Case study: Clickhouse
- Case study: textual analysis

Hyperscan Overview

- Hyperscan is a regular expression matching library
  - Software-only, IA specific (requires SSSE3 as a baseline!)
  - Support majority syntax of PCRE (~~zero width~~, ~~capturing~~)
  - Match "Rulesets" on data blocks or packet streaming
  - Callback if match found. Flexible and powerful



Hyperscan:

An industry fastest Regular Expression, Literal Matching Algorithm on Intel platform

Applications

Integrated Open Source Solutions and User Cases（40+ Customers, 37 Open Source Projects)

IDS/IPS: SNORT, SURICATA

SD-WAN/DPI: ntop

Web Application Firewall: modsecurity Open Source Web Application Firewall, NGINX, NAXSI

Language Bindings

golang, Java, python

Spam Filtering System: RSPAMD spam filtering system

Github

Clickhouse Database

Operating Systems

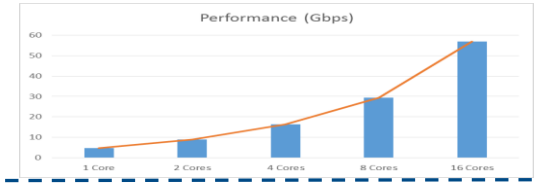
FreeBSD, fedora, ubuntu, debian, gentoo linux, Windows, OS X

Intel Architectures

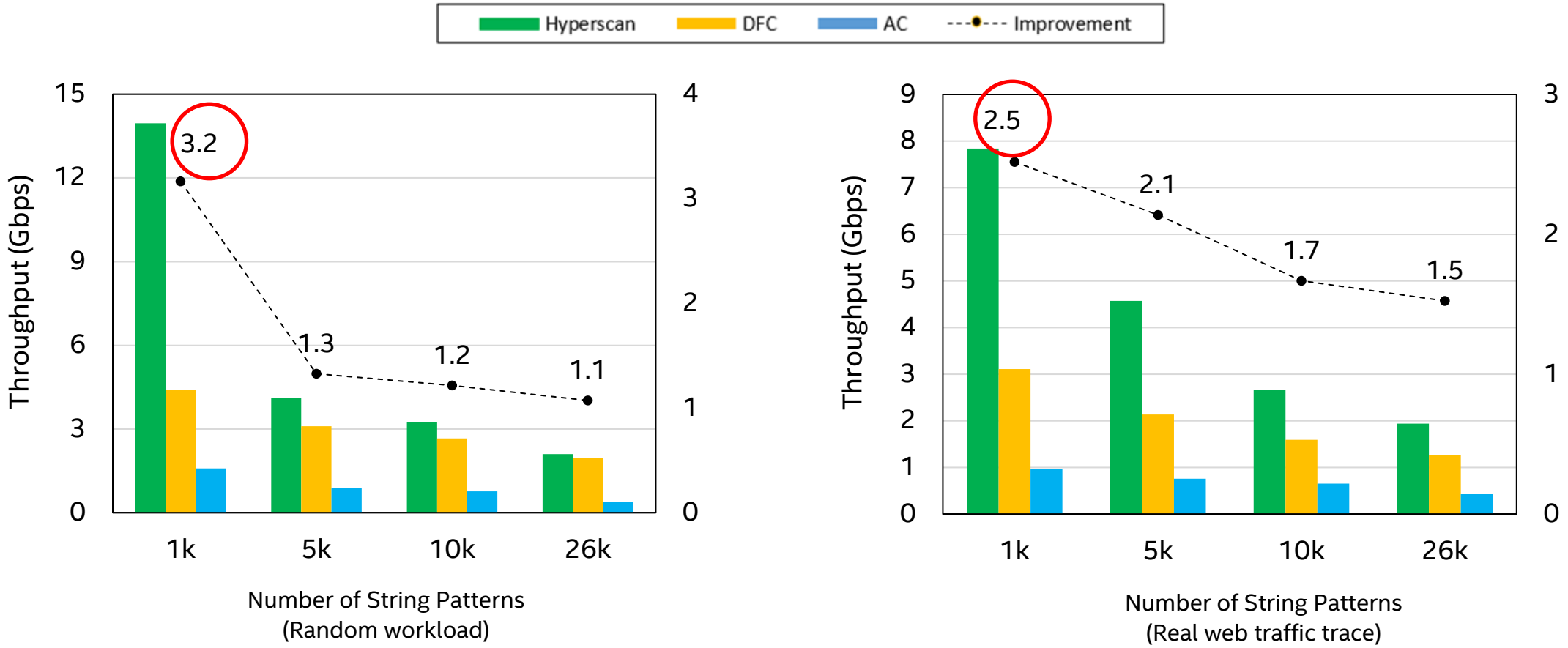
Seamless Support from Atom to Xeon processor

intel inside ATOM ... intel inside CORE i5 ... intel inside XEON

Linear Core Scalability, Intel Optimized



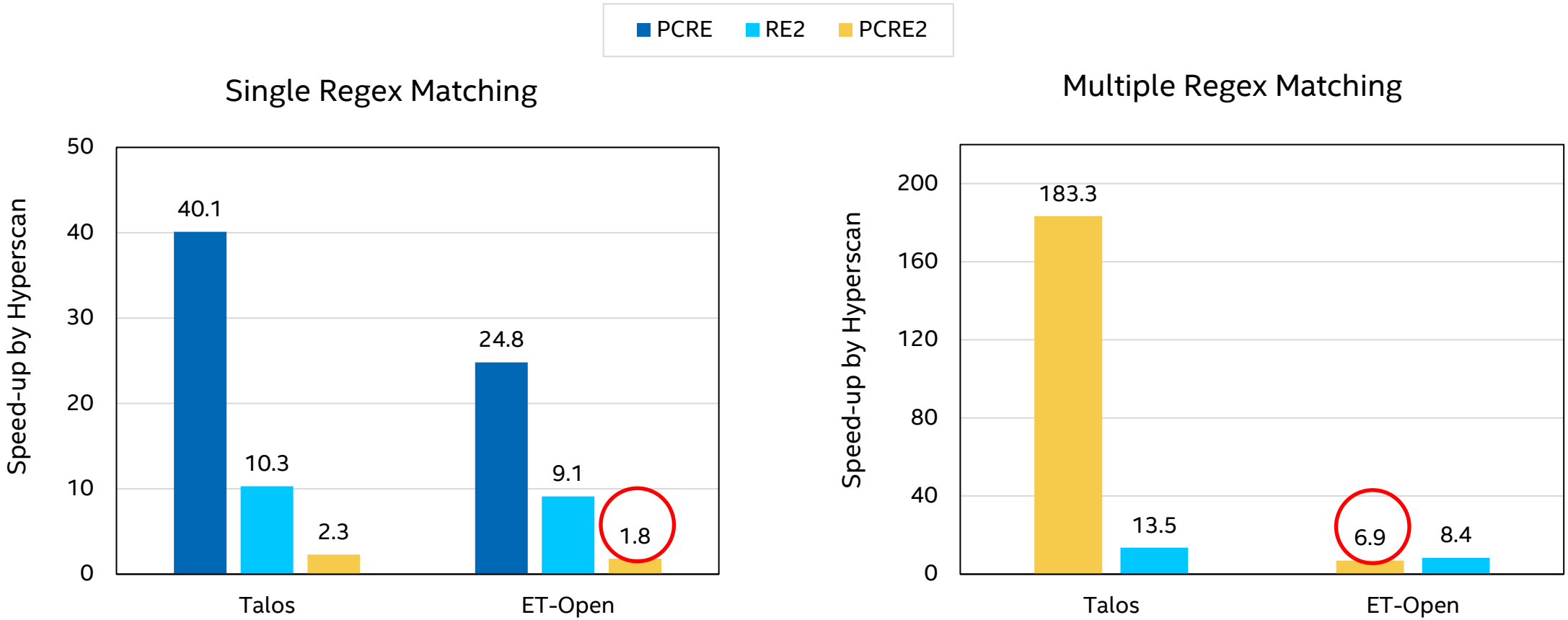
Multi-literal Matching Performance with Snort ET-Open



- Hyperscan 5.0.0
- Intel Xeon Platinum 8180 CPU @ 2.50GHz
- Single core

https://www.usenix.org/system/files/nsdi19-wang-xiang.pdf

Regex Matching Performance

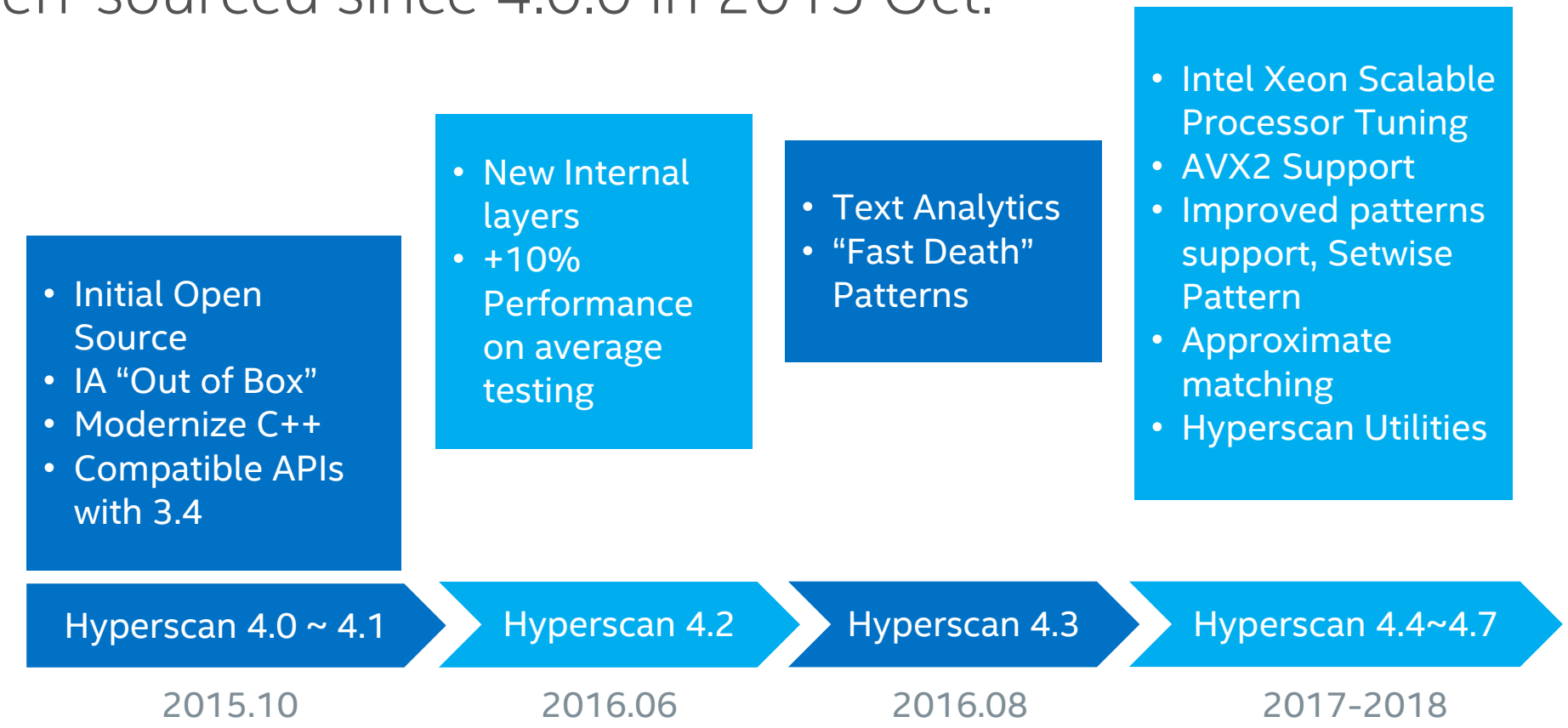


- Snort Talos: 1,300 regexes
- ET-Open: 2,800 regexes
- Real Web traffic trace.

https://www.usenix.org/system/files/nsdi19-wang-xiang.pdf

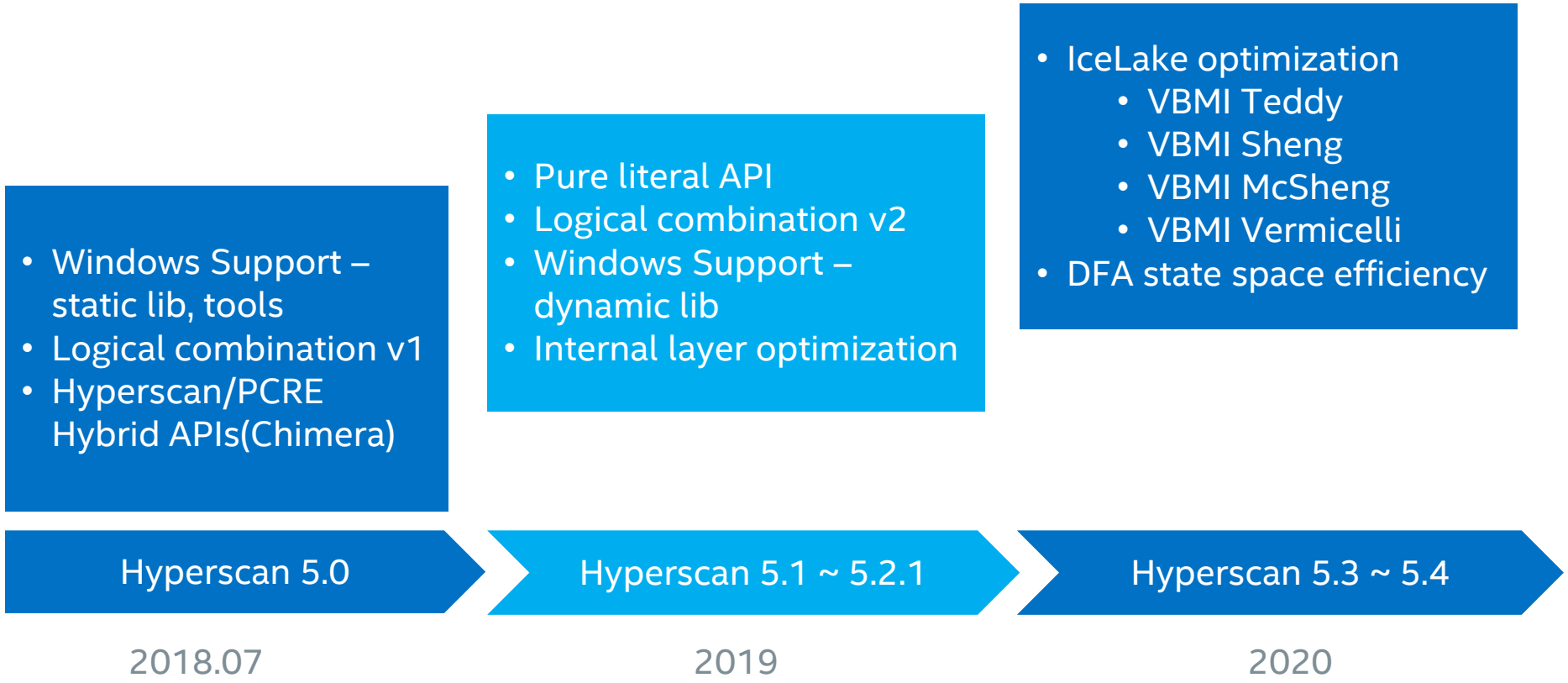
Hyperscan Footprint

- Open-sourced since 4.0.0 in 2015 Oct.



Hyperscan Footprint

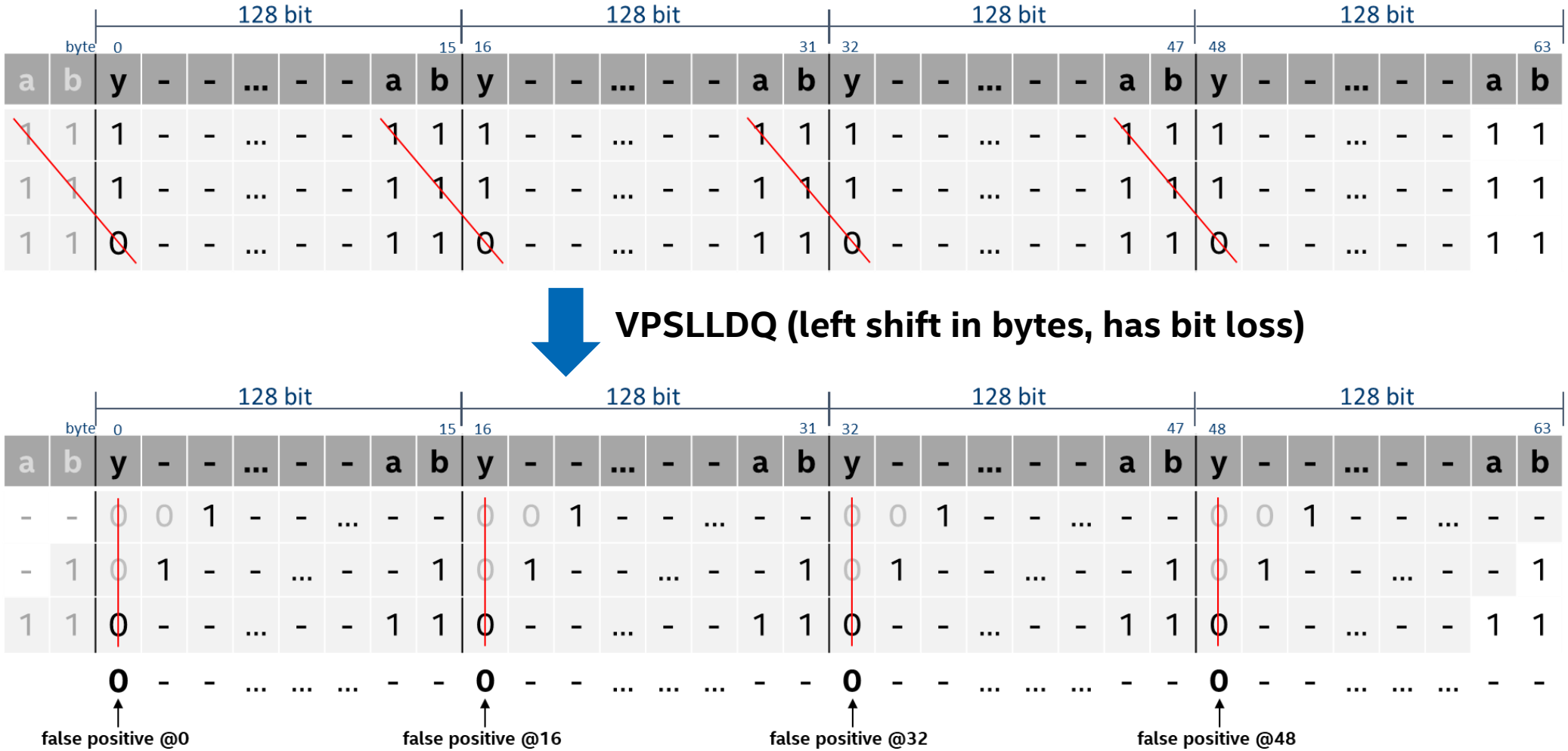
- Latest release: 5.4.0 (2020.Dec)



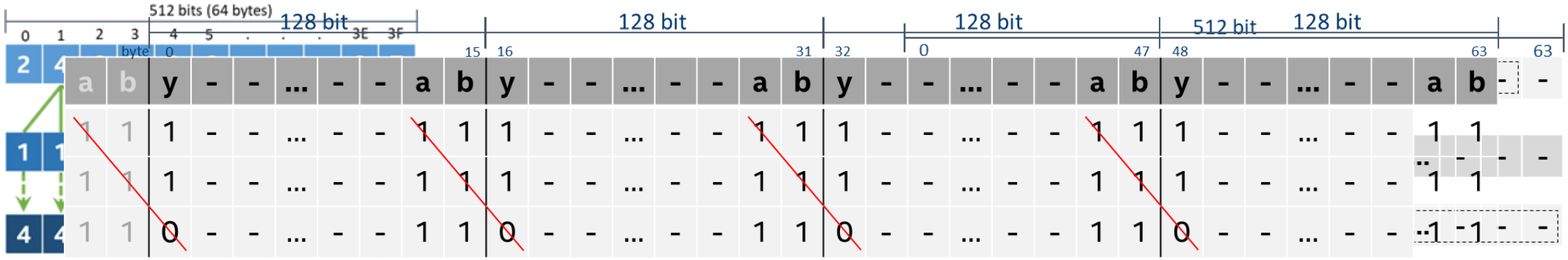
5.3/5.4 Release Update

AVX512/AVX512VBMI optimization for components

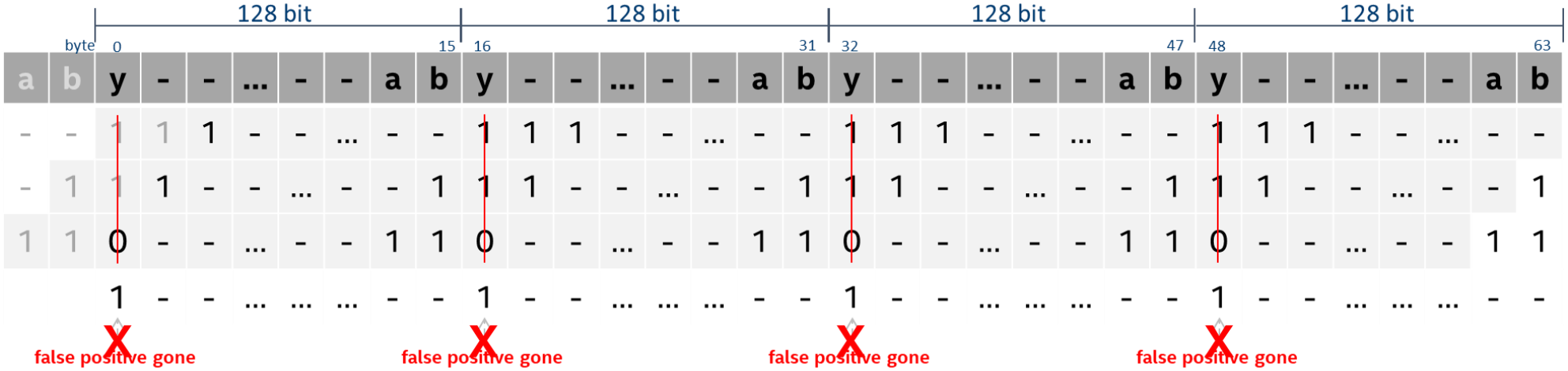
AVX512VBMI Optimization: Literal Matching (Teddy)



AVX512VBMI Optimization: Literal Matching (Teddy)



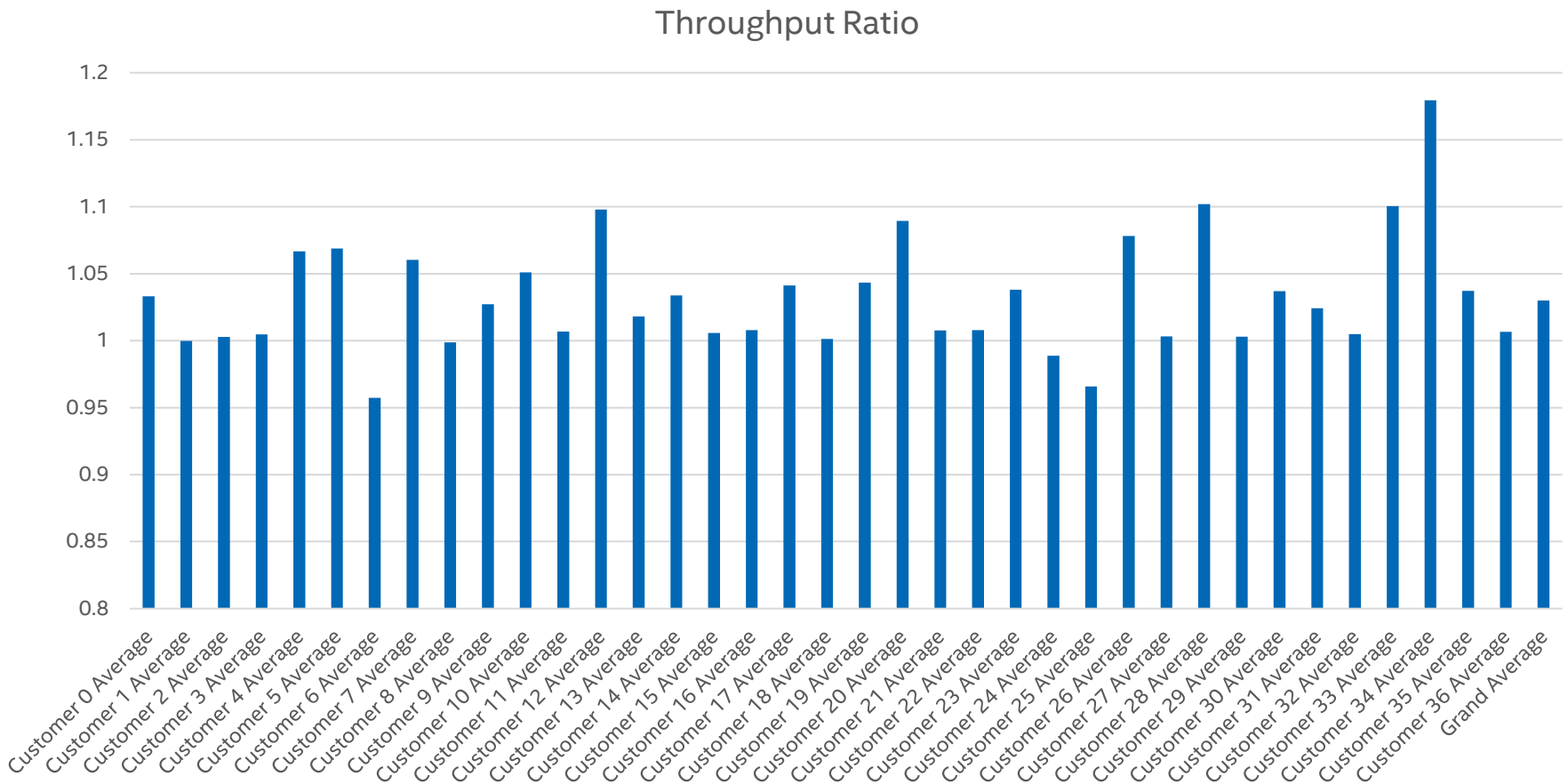
VPERMB (left shift in bytes, no bit loss)



AVX512VBMI Acceleration: Teddy

Small scale literal matching Teddy:

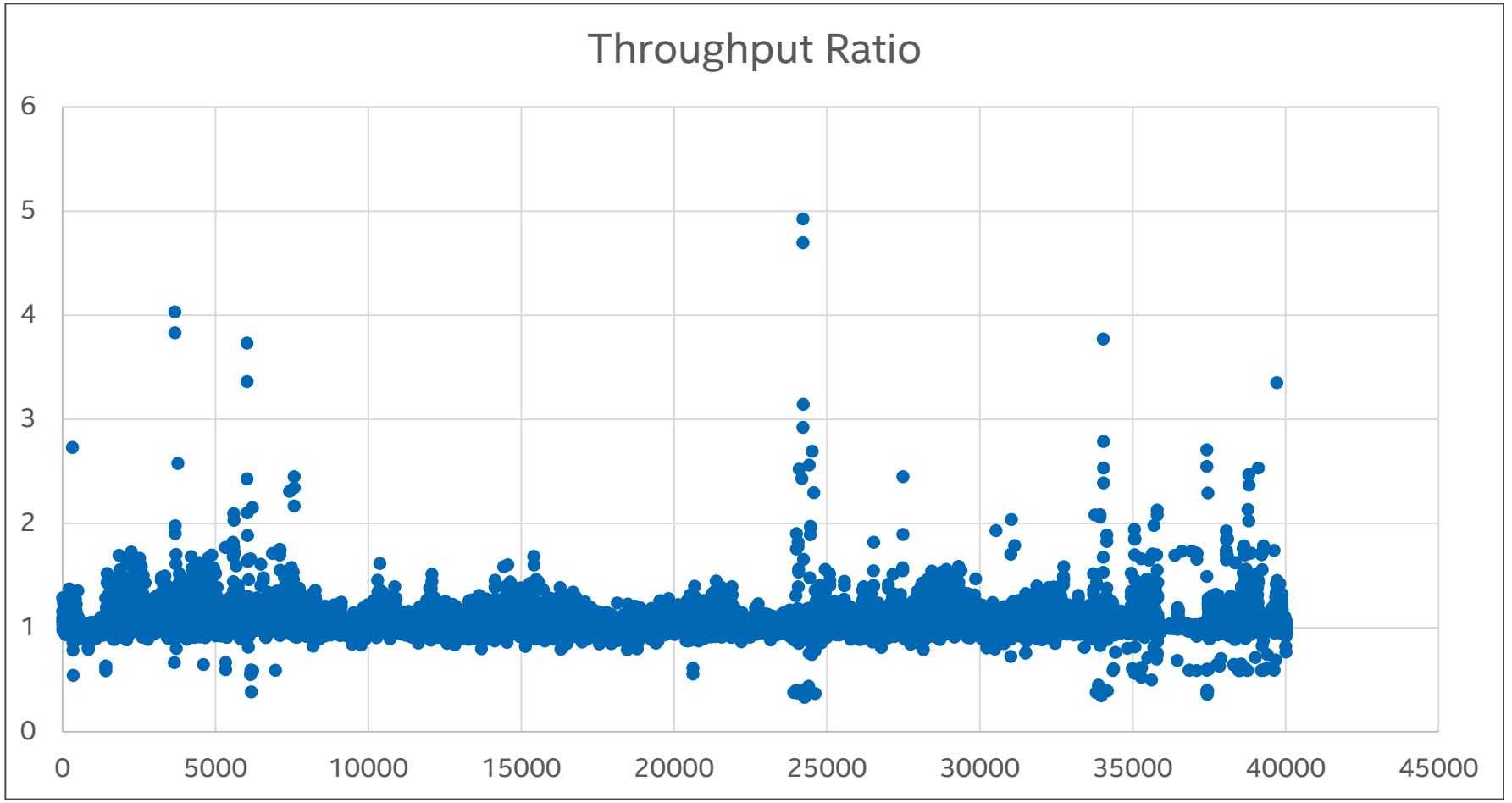
- Accelerated with VPERMB from AVX512VBMI
- Best case: 17.9%↑ for a single customer ruleset



AVX512VBMI Acceleration: Teddy

Small scale literal matching Teddy:

- Average of 16,800 cases optimized: ~6.5% ↑
- Best case: ~4.92x ↑



AVX512VBMI Optimization - DFA (Sheng)

256 columns

State	...	a	b	c	d	...
0	...	1	0	0	0	...
1	...	1	2	0	0	...
2	...	2	2	3	2	...
3	...	2	2	3	4	...
4	...	2	2	3	2	...

DFA transition table

512 bits (64 states)

TRANS_MASK[0x00]	0	0	0	0	0	1	.....	0	0
⋮	.....								
TRANS_MASK[a]	2	3	4	5	6	10	.....	61	62
TRANS_MASK[b]	6	35	2	0	0	15	.....	7	23
⋮	.....								
TRANS_MASK[0xff]	45	36	35	45	61	4	.....	0	1

pos := state << 8 + char_input

state := TRANS_TABLE[pos]

Critical path: **6~7 cycles**

mask512 = TRANS_MASK[char_input]

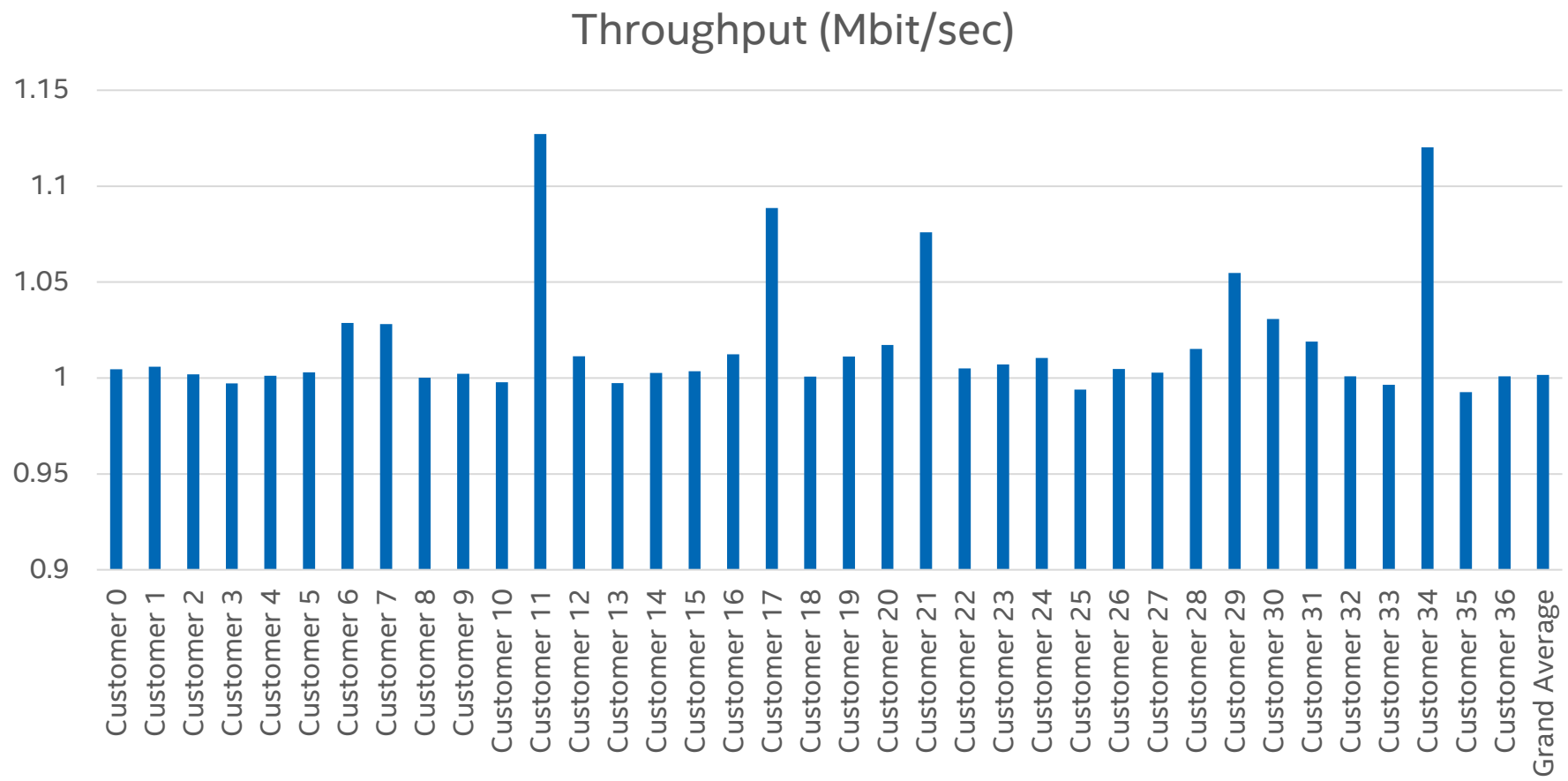
state512 = VPERMB(state512, mask512)

Critical path: **3 cycles**

AVX512 Acceleration - Sheng

Shuffled DFA Engine – Sheng:

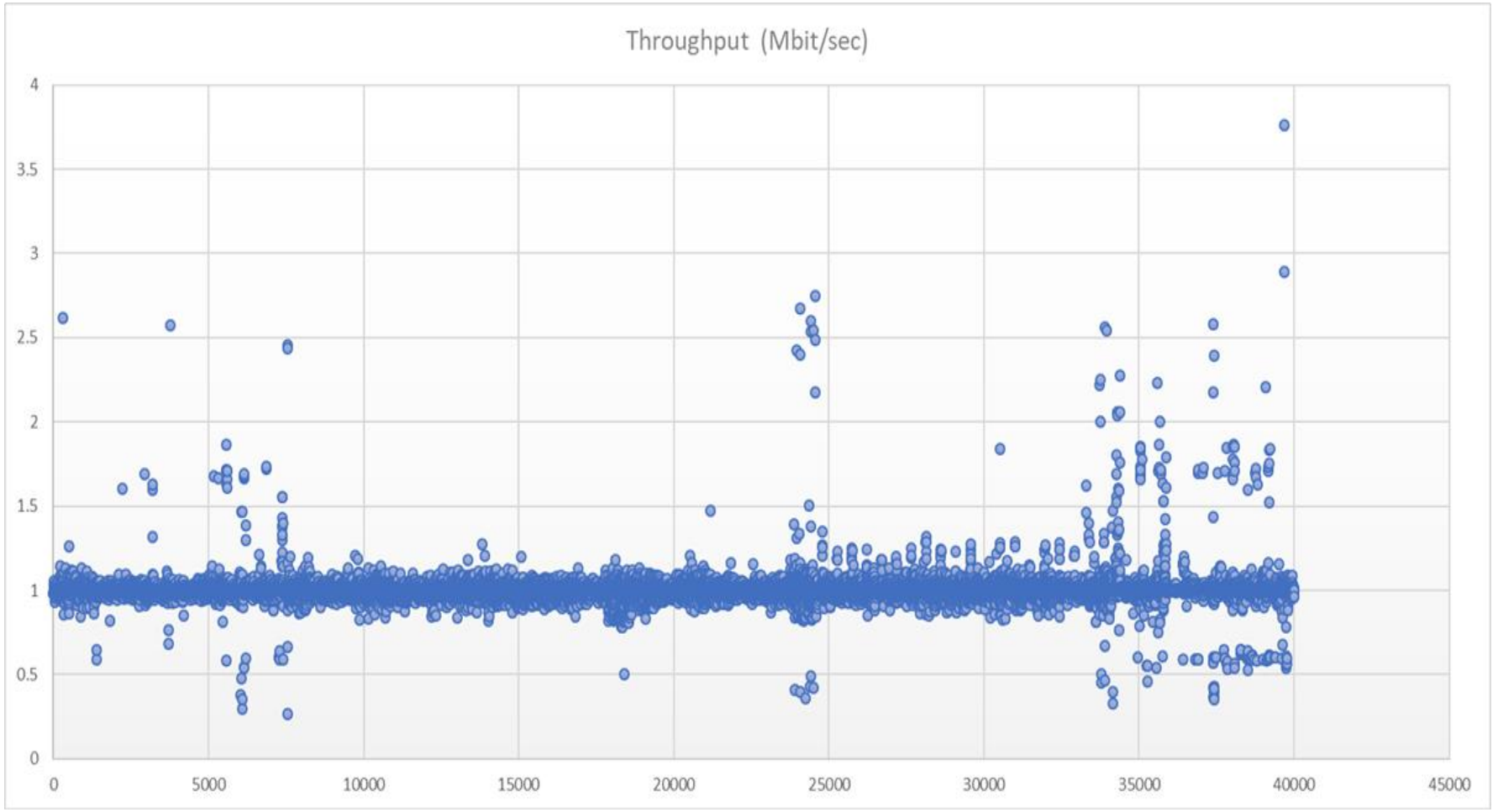
- Extend state to 64 with VPERMB from AVX512VBMI
- Best case: 12.72%↑ for a single customer signature set



AVX512 Acceleration - Sheng

Shuffled DFA Engine – Sheng :

- Average of 4021 cases optimized: ~1.1% ↑
- Best case: ~2.08x ↑



AVX512 Optimization: 1/2-byte Matching (Vermicelli)

```
1.  chars1 = VPBROADCASTB(c1);      // 16B 广播
2.  chars2 = VPBROADCASTB(c2);      // 16B 广播
3.  ……
4.  LOOP till buf_end:
5.      data = MOVDQA(buf);         // 16B 加载
6.      t0 = PCMPEQB(chars1, data); // 16B 比较
7.      t1 = PCMPEQB(chars2, data); // 16B 比较
8.      t1 = PSRLDQ(t1, 1);         // 16B 右移位
9.      t = AND(t0, t1);            // 16B 按位与
10.     z = PMOVMSKB(T);            // 16B 最高位提取
11.     pos = ctz32(z);
12. ……
```

SSSE3 implementation

1. Direct replacement (wider length)
   - 64B 广播 64B 加载 64B 按位与
   - **VPBROADCASTB, VMOMDQA32, VPAND**
2. Instruction combination
   - 16B 比较 16B 最高位提取
   - **PCMPEQB + PMOVMSKB**

   64B 比较+最高位提取

   ➔ **VPCMPB** (return 64-bit value)
3. Special case (byte shift)
   - 16B 右移位 64B 右移位
   - **PSRLDQ➔VPSRLDQ:** shift for 64 bytes, bit loss at 128-bit boundary

     比较 > 移位 > 最高位提取

     for 512-bit

   64B 比较+最高位提取
   - **VPCMPB:** >> for 64-bit value, no bit loss

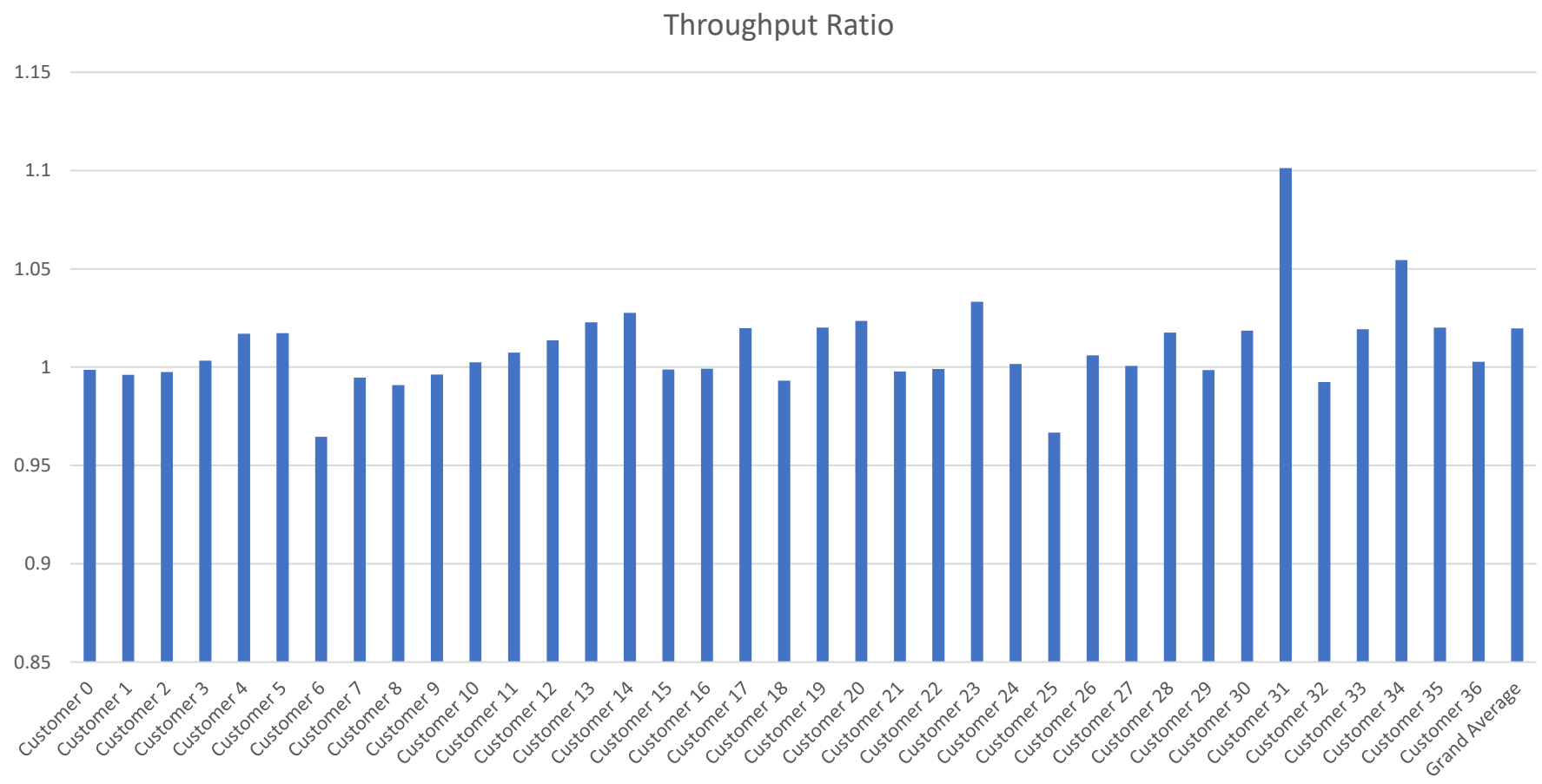
     比较 > 最高位提取 > 移位

     for 64-bit

AVX512 Acceleration: Vermicelli

Acceleration scheme Vermicelli:

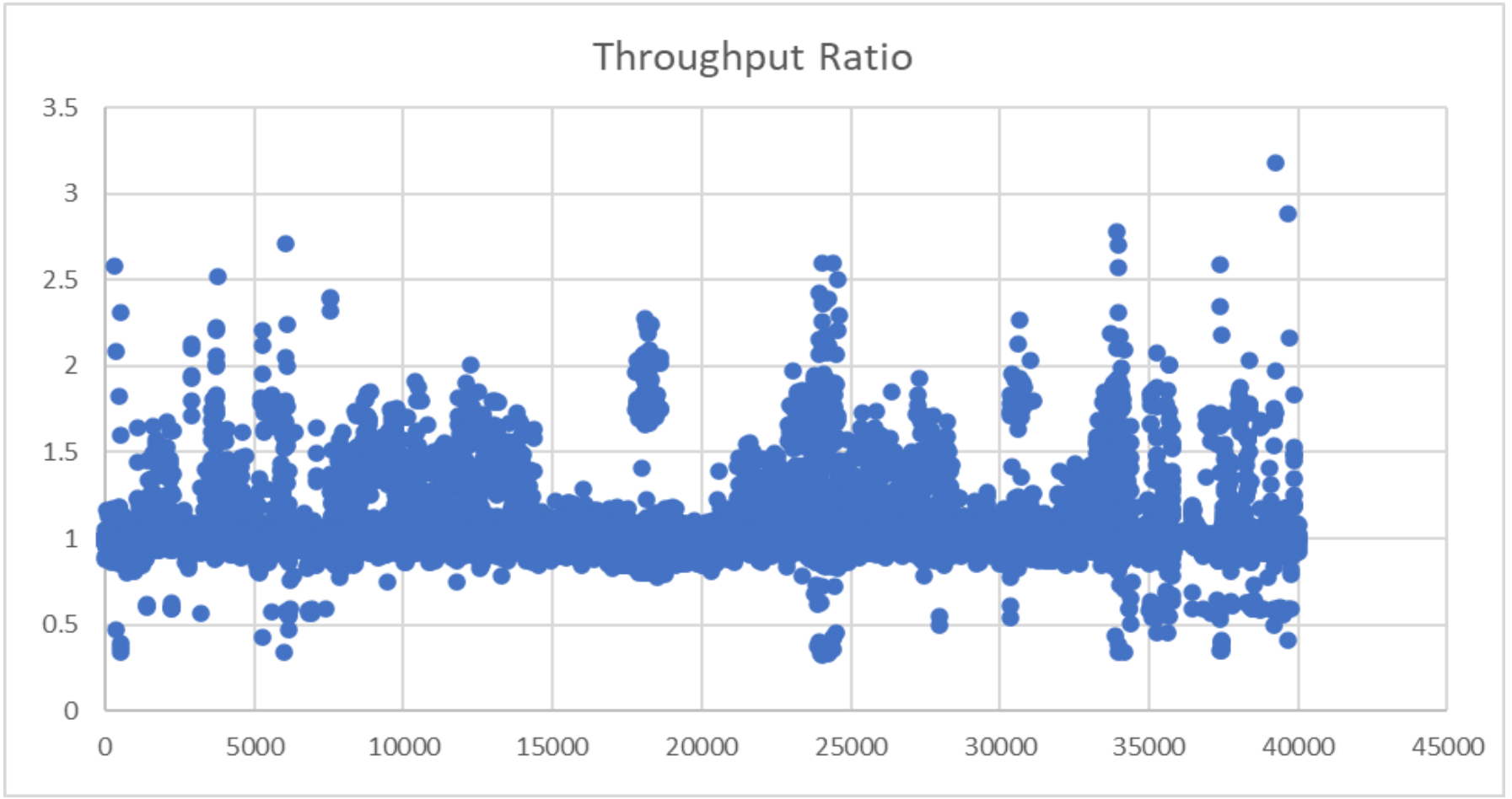
- Accelerated with VPCMPB from AVX512VL + AVX512BW
- Best case: 10.1%↑ for a single customer ruleset



AVX512 Acceleration: Vermicelli

Acceleration scheme Vermicelli :

- Average of total 40,000 cases: ~2.0% ↑
- Best case: ~2.2x ↑

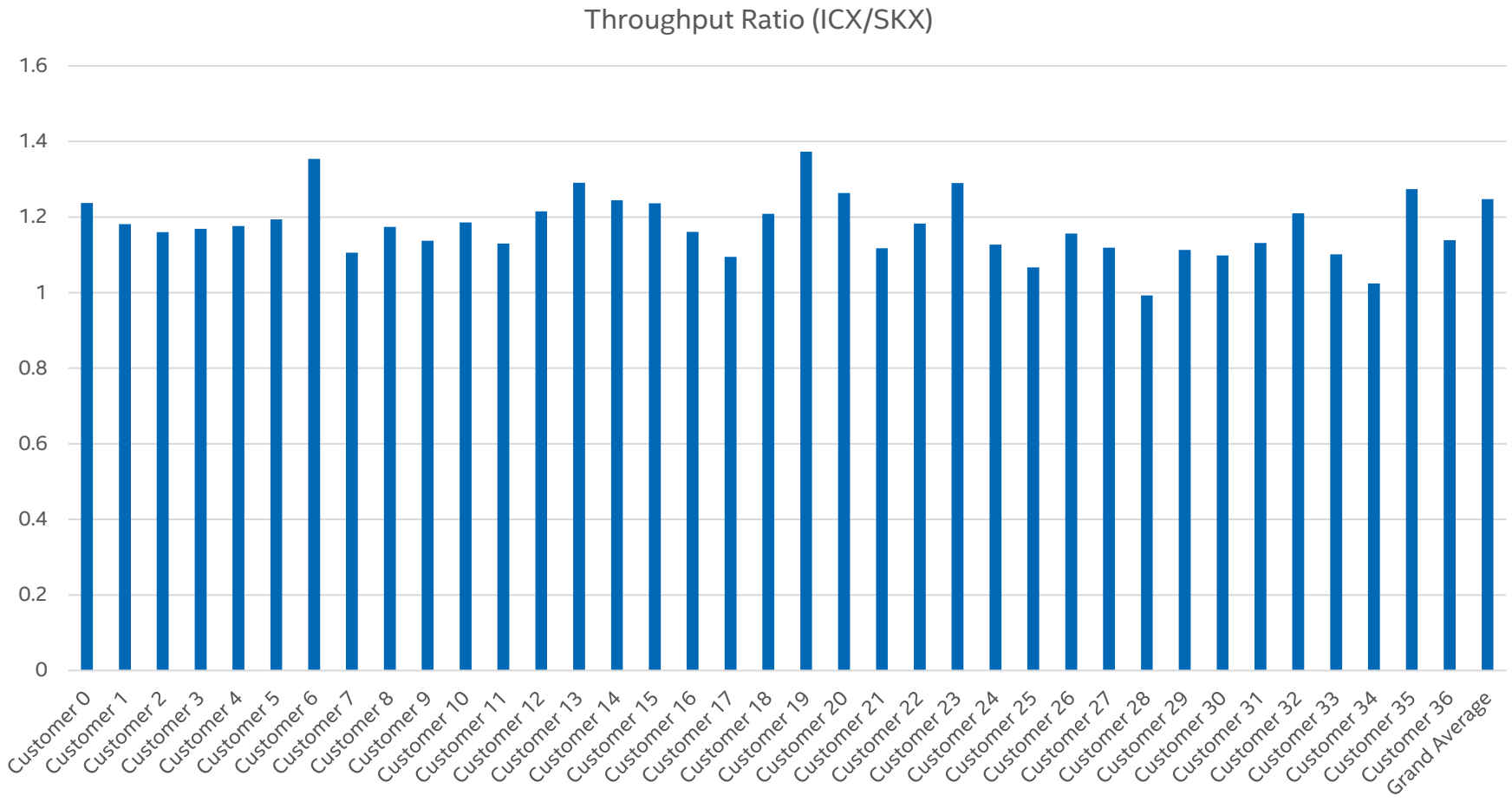


IceLake Benckmark - Overview

https://github.com/intel/hyperscan (5.4.0, 5.2.0)

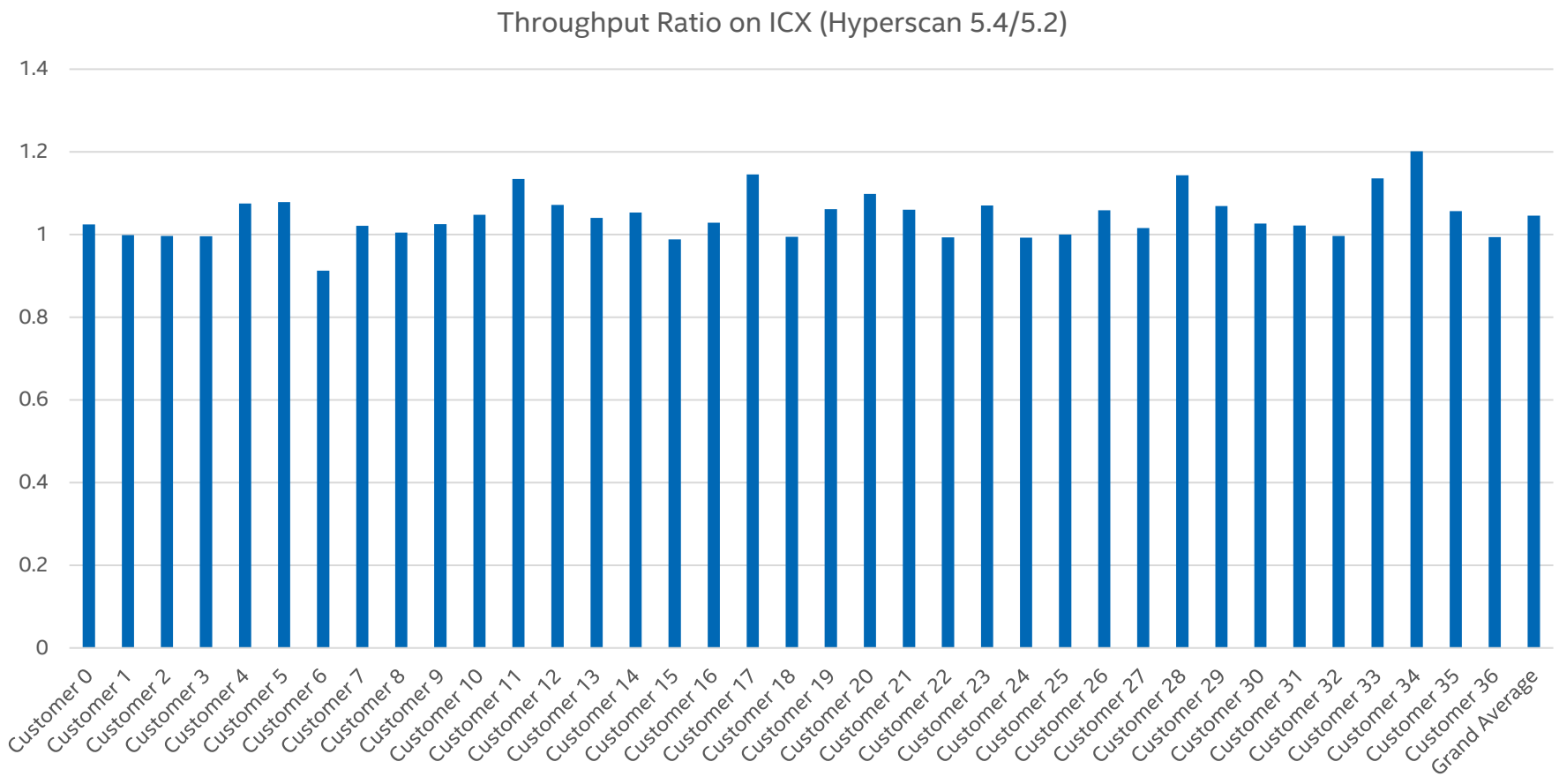
Hyperscan Performance – ICX v.s. SKX

- Hyperscan 5.2 performance on ICX and SKX
- Hsbench with 37 customer rulesets on captured network traffic
- Average 24.7%↑ for all customer rulesets
- Best case: 37.3%↑ for a single customer ruleset



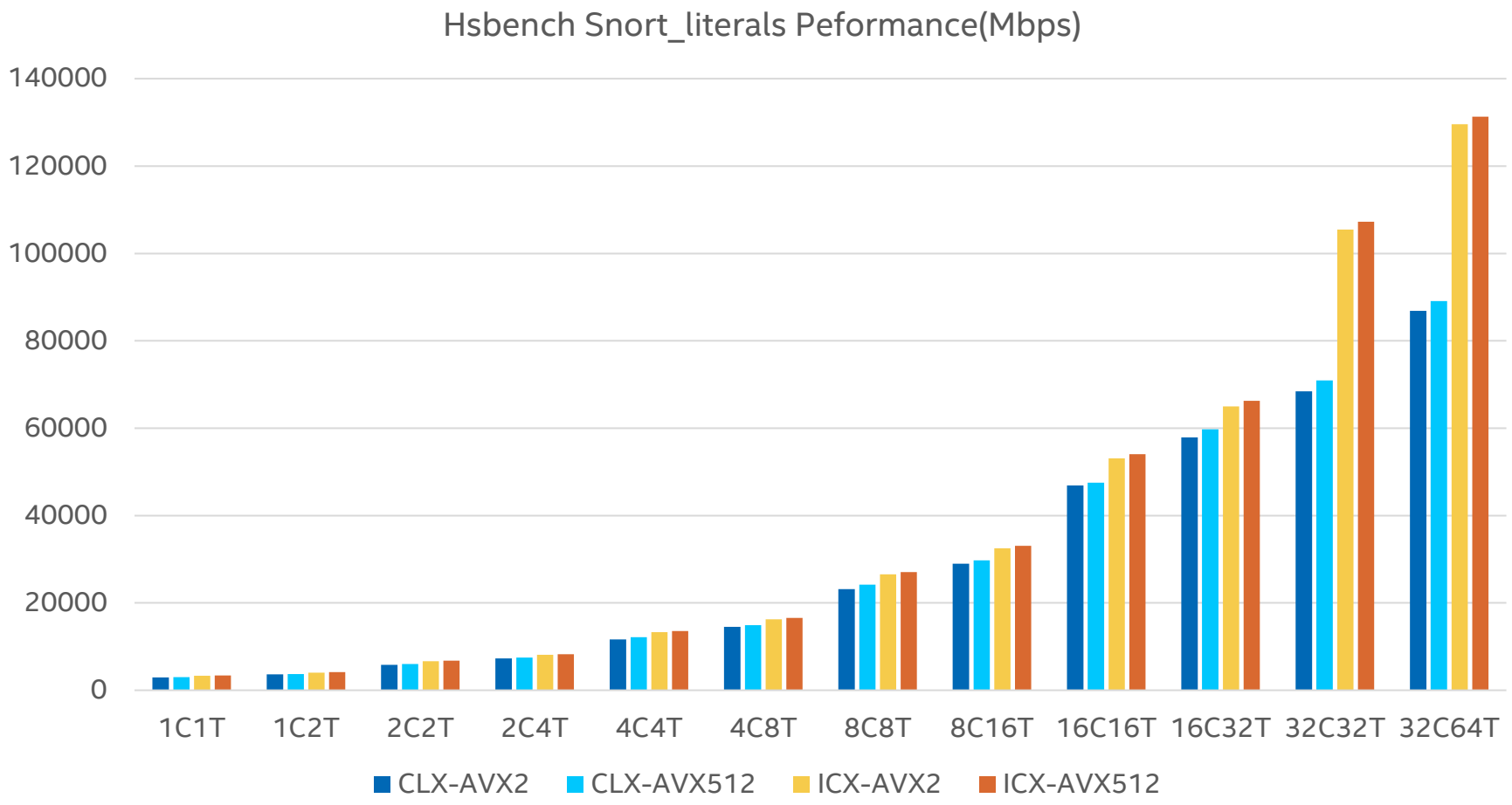
Hyperscan Performance on ICX – 5.4 v.s. 5.2

- Hyperscan 5.4 is AVX512(VBMI) optimized compared to 5.2
- Hsbench with 37 customer rulesets on captured network traffic
- Average 4.6%↑ for all customer rulesets
- Best case: 20.1%↑ for a single customer ruleset



AVX512 Performance on CLX and ICX

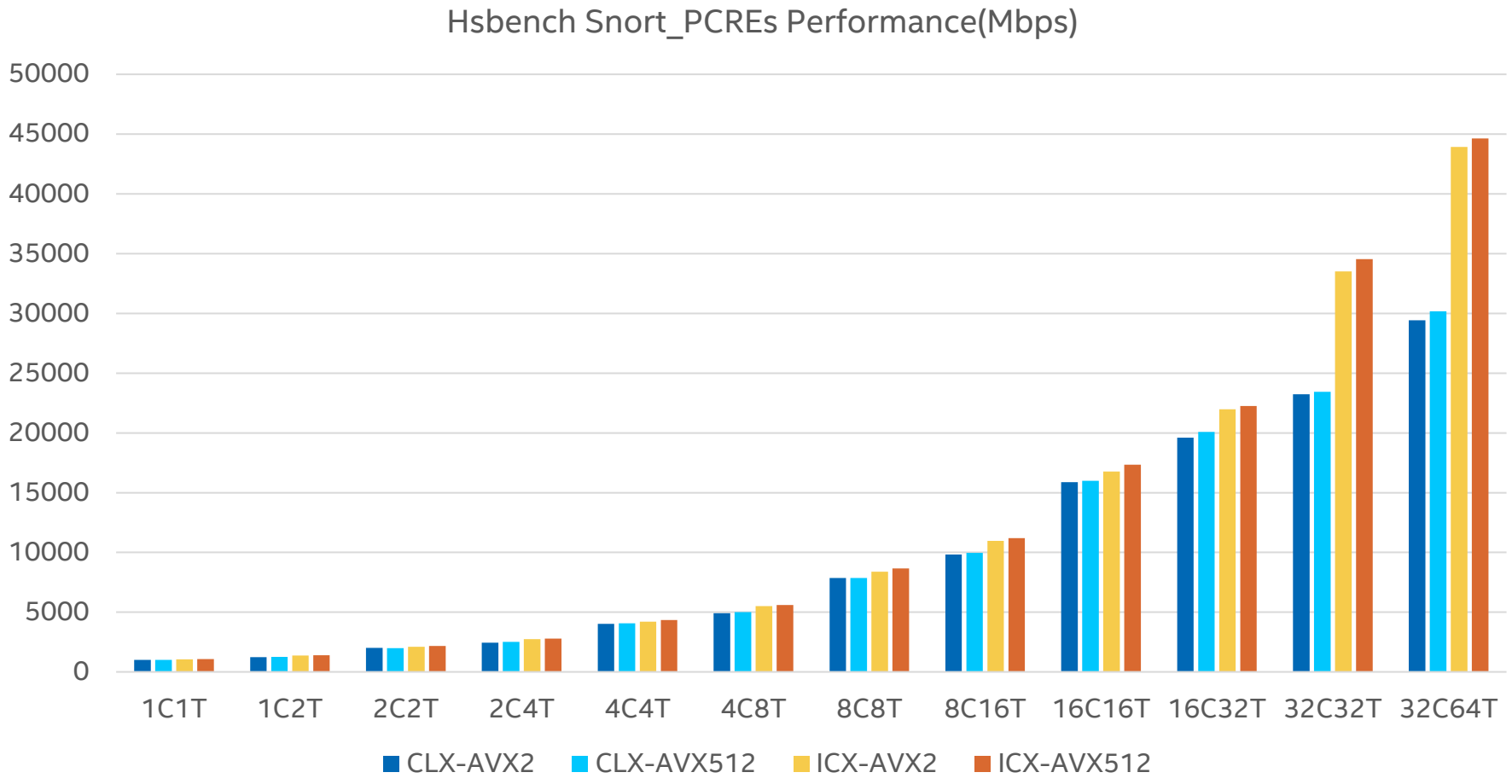
- Hyperscan 5.4
- Hsbench with open ruleset snort_literals on http network traffic



https://01.org/hyperscan/downloads/sample-data-hyperscan-hsbench-performance-measurement

AVX512 Performance on CLX and ICX

- Hyperscan 5.4
- Hsbench with open ruleset snort_pcres on http network traffic

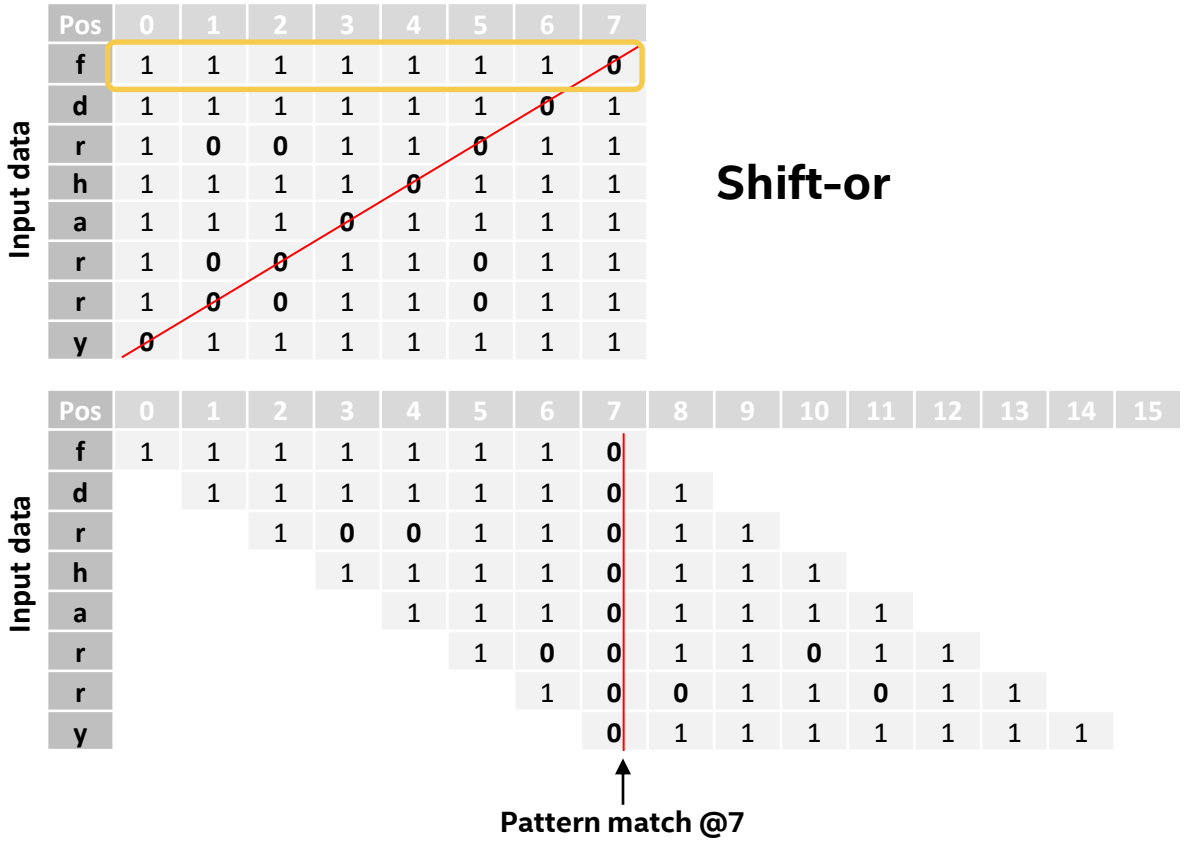


https://01.org/hyperscan/downloads/sample-data-hyperscan-hsbench-performance-measurement

5.5 Features and Future

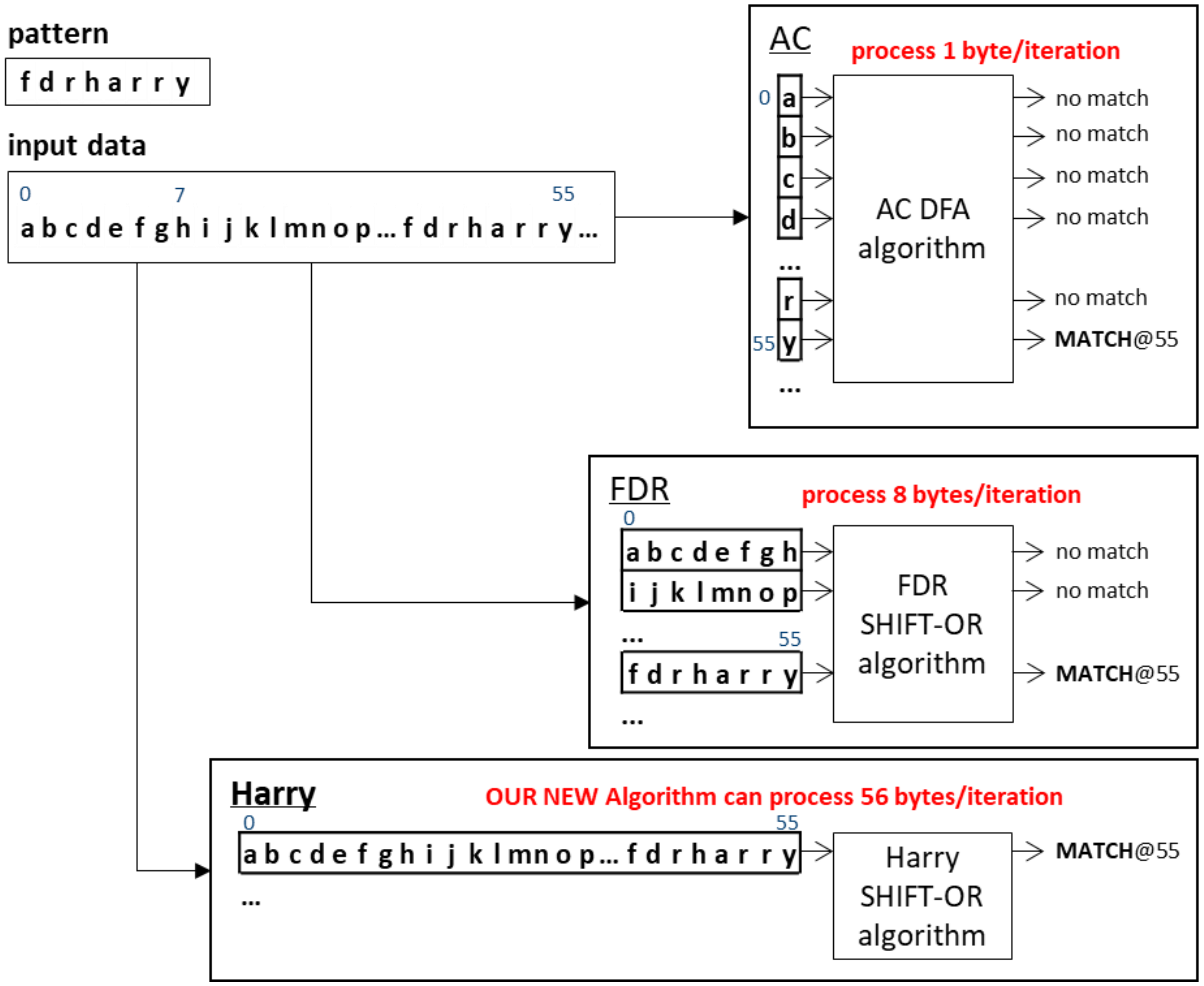
Faster multi-literal matcher (than FDR)

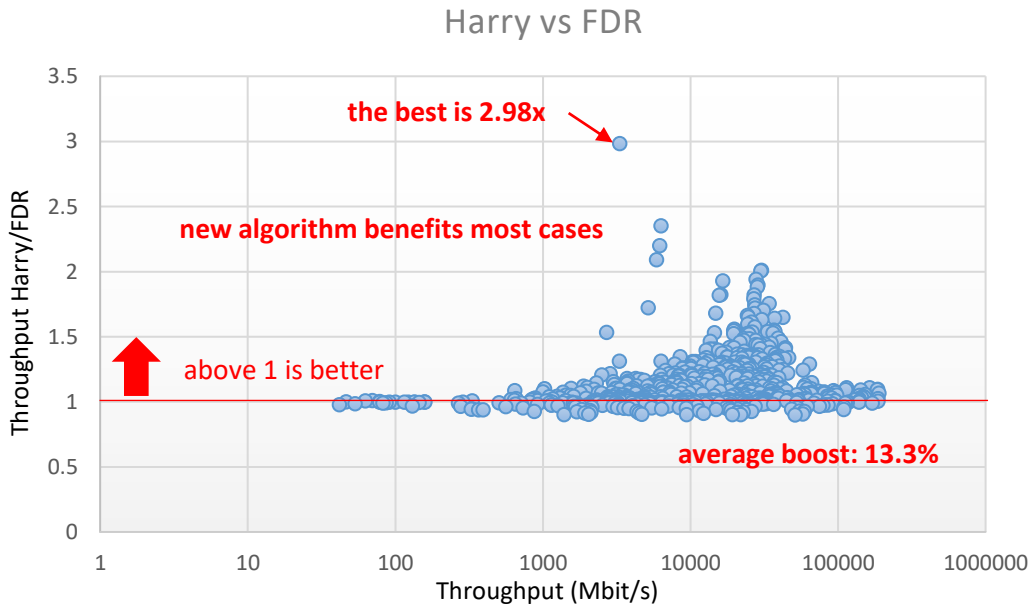
Pattern	y	r	r	a	h	r	d	f
Offset	0	1	2	3	4	5	6	7
0x00	1	1	1	1	1	1	1	1
…	…	…	…	…	…	…	…	…
a	1	1	1	0	1	1	1	1
…	…	…	…	…	…	…	…	…
d	1	1	1	1	1	1	0	1
e	1	1	1	1	1	1	1	1
f	1	1	1	1	1	1	1	0
g	1	1	1	1	1	1	1	1
h	1	1	1	1	0	1	1	1
…	…	…	…	…	…	…	…	…
r	1	0	0	1	1	0	1	1
…	…	…	…	…	…	…	…	…
y	0	1	1	1	1	1	1	1
…	…	…	…	…	…	…	…	…
0xff	1	1	1	1	1	1	1	1



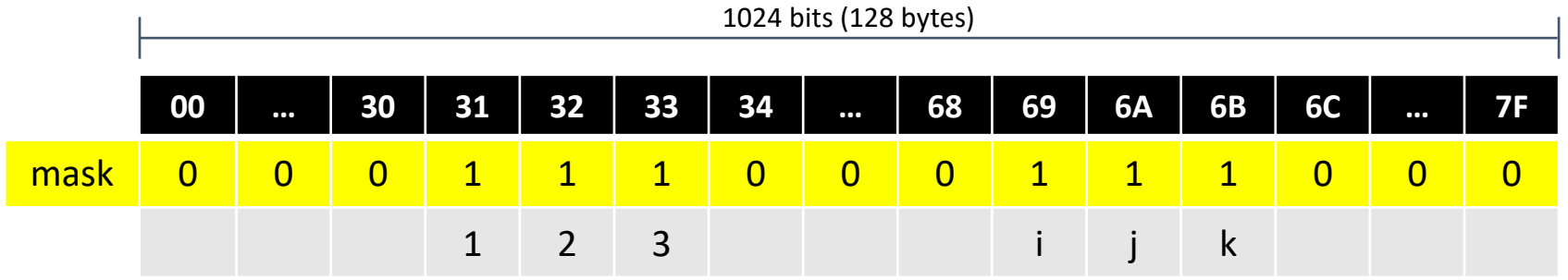
https://www.usenix.org/system/files/nsdi19-wang-xiang.pdf

Multi-literal matchers

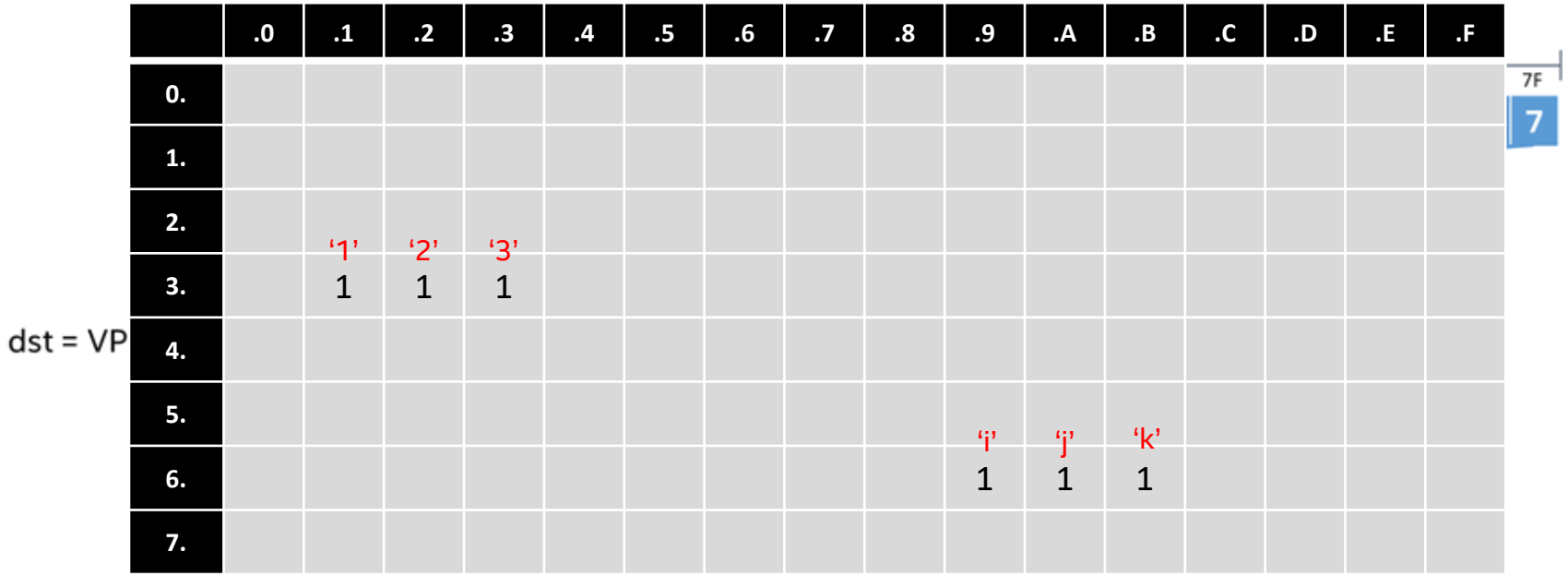




Faster Character class matcher

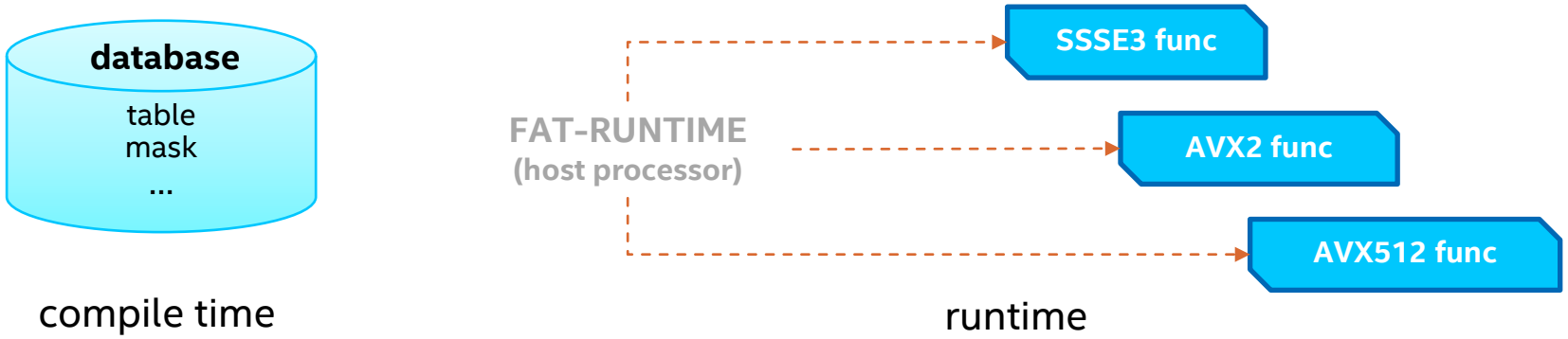


Example: [1-3i-k], (support characters with ASCII value < 128)

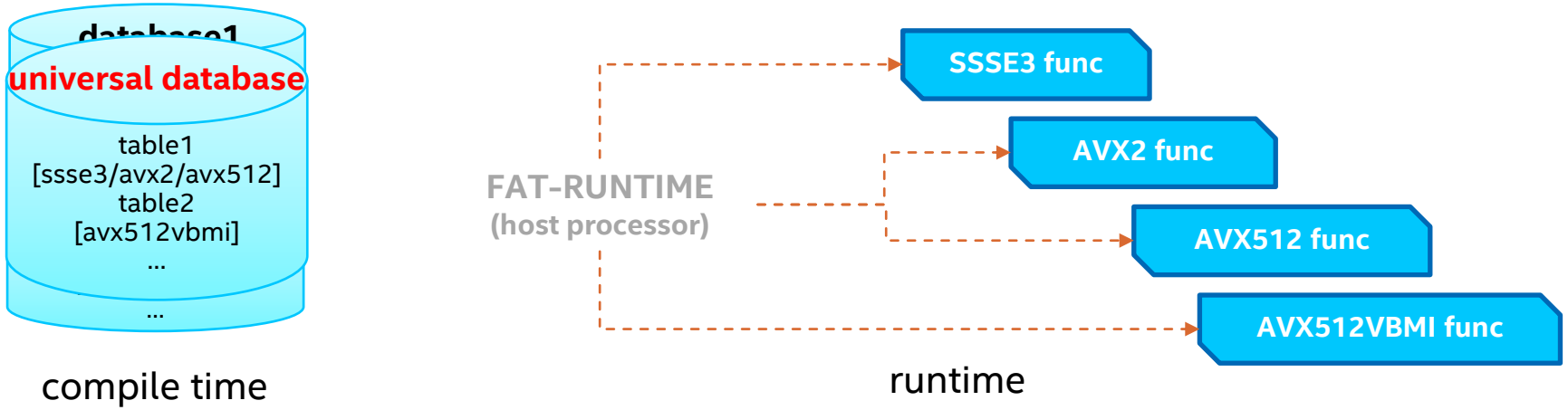


Universal Database – Background

- Early stage

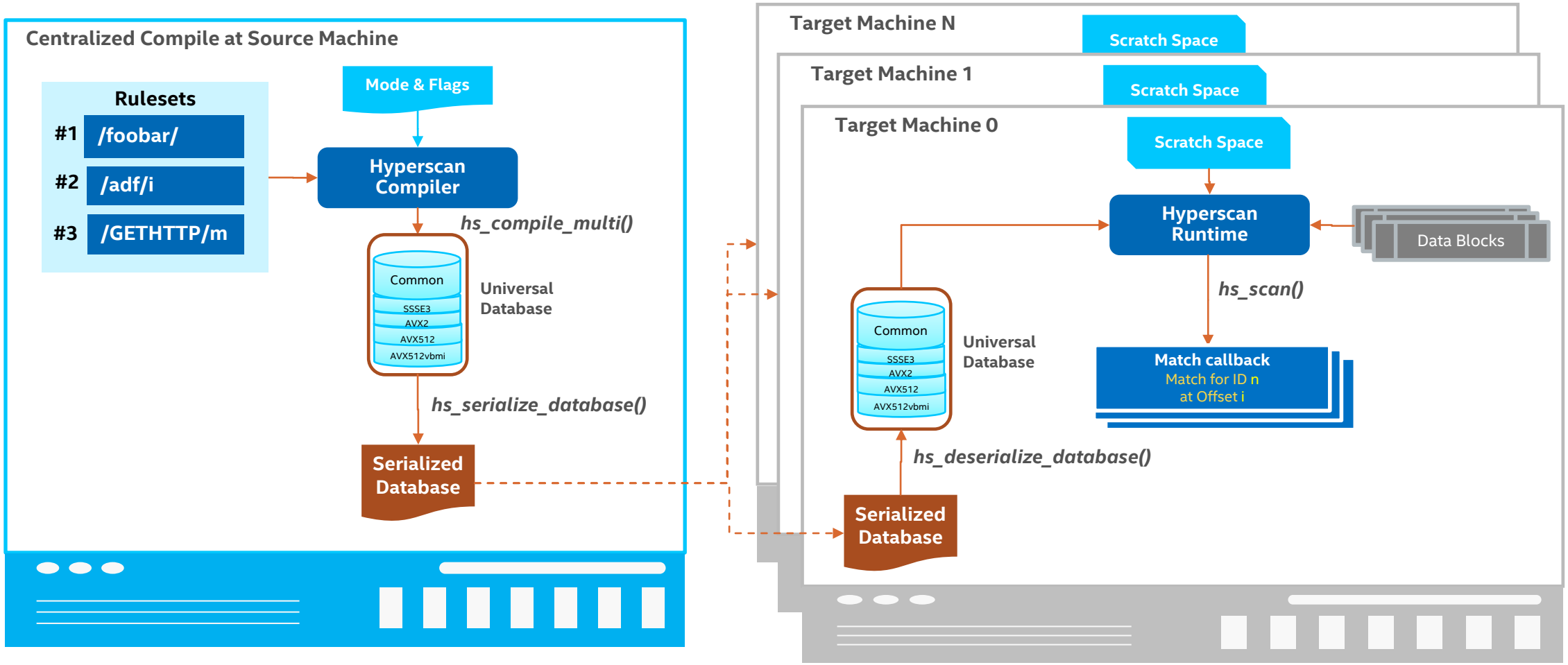


- Late stage



Universal Database 2.0

Keep traditional APIs



Long-term Hyperscan

- Sustained optimization for new platform (IceLake, Sapphire Rapids)
- Hyperscan decoupling

Not every customer needs "all of Hyperscan"

  - Separate literal matcher
    - More general: allow character class

[Hh][Yy][Pp][Ee][Rr][Ss][Cc][Aa][Nn]

  - Other engines
    - Looser coupling
    - Keep integrity and performance

Case Study: Clickhouse

Hyperscan integration with Database

Clickhouse

- Open source database solution from Yandex under Apache 2 license license (https://github.com/ClickHouse/ClickHouse)
- Column-oriented DBMS (columnar database management system) for online analytical processing (OLAP)
- Stable, sustainable and linearly scalable up to store and process trillions of rows and petabytes of data.
- Integrate Hyperscan for multiple regex matching functions in v19.5.2.6 (2019) (https://clickhouse.tech/docs/en/query_language/functions/string_search_functions/)
- Replacing default solution RE2 with Hyperscan shows significant boost for database queries with regular expressions and add support of approximate multiple regex matching (https://presentations.clickhouse.tech/hse_2019/string_algorithms.pdf)

ClickHouse Performance

Regex matching: RE2 vs Hyperscan v5.1.1(corei7)

ID	Query (MB/s)	RE2	Hyperscan
1-1	select sum(multiMatchAny(URL, ['/t[0-9]+-', '/questions/7{9}[0-9]+'])) from tutorial.hits_100m_v1 settings max_threads=1	376.91	540.2
1-2	select sum(multiMatchAnyIndex(URL, ['/t[0-9]+-', '/questions/7{9}[0-9]+']))	406.02	541.98
2-1	select sum(multiMatchAny(URL, ['ножниц.*вырубн', 'ножниц.*рычажн', 'ножниц.*гильотин']))	1010	703.1
2-2	select sum(multiMatchAnyIndex(URL, ['ножниц.*вырубн', 'ножниц.*рычажн', 'ножниц.*гильотин']))	967.77	703.12
3-1	select sum(multiMatchAny(URL, ['f[ae]b[ei]rl', 'ф[иаэе]б[еэи][рпл]', 'афиукд', 'a[ft],th','^ф[аиеэ]?б?[еэи]?$', 'берлик', 'fab', 'фа[беьв]+е?[рлко]']))	264.7	542.35
3-2	select sum(multiMatchAnyIndex(URL, ['f[ae]b[ei]rl', 'ф[иаэе]б[еэи][рпл]', 'афиукд', 'a[ft],th','^ф[аиеэ]?б?[еэи]?$', 'берлик', 'fab', 'фа[беьв]+е?[рлко]']))	255.65	523.6
4-1	select sum(multiMatchAny(URL, ['/questions/q*', '/q[0-9]*/', '/questions/[0-9]*']))	389.44	689.47
4-2	select sum(multiMatchAnyIndex(URL, ['/questions/q*', '/q[0-9]*/', '/questions/[0-9]*']))	382.26	692.55
5-1	select sum(multiMatchAny(URL, ['//ngs.ru/$', '//m.ngs.ru/$', '//news.ngs.ru/$','//m.news.ngs.ru/$','//ngs.ru/\\?', '//m.ngs.ru/\\?','//news.ngs.ru/\\?', '//m.news.ngs.ru/\\?']))	226.73	483.29
5-2	select sum(multiMatchAnyIndex(URL, ['//ngs.ru/$', '//m.ngs.ru/$', '//news.ngs.ru/$','//m.news.ngs.ru/$','//ngs.ru/\\?', '//m.ngs.ru/\\?','//news.ngs.ru/\\?', '//m.news.ngs.ru/\\?']))	220.64	444.58
6-1	select sum(multiMatchAny(URL, ['[ми][аеэпви][нм][асзи][иус]*', '[mn][aeauo][nm]s[yyi]*','ru', 'v[ft\']v[cp][be]', 'www', 'ьфььын', 'маиси', 'mam','amsy', 'маммси', 'амси', 'vfvc']))	99.37	514.63
6-2	select sum(multiMatchAnyIndex(URL, ['[ми][аеэпви][нм][асзи][иус]*', '[mn][aeauo][nm]s[yyi]*','ru', 'v[ft\']v[cp][be]', 'www', 'ьфььын', 'маиси', 'mam','amsy', 'маммси', 'амси', 'vfvc']))	97.51	513.18

(v20.11.3.3-stable, SKL)

Tip: To use Hyperscan literal matcher, pass strings to multiMatchxxx()

ClickHouse Performance

String matching: Multi Volnitsky vs Hyperscan v5.1.1(corei7)

ID	Query (MB/s)	Volnitsky	Hyperscan
1	select sum(multiSearchAny(URL, ['yandex', 'google'])) from tutorial.hits_100m_v1 settings max_threads=1	974.8	616.95
2	select sum(multiSearchAny(URL, ['yahoo', 'pikabu']))	901.43	634.24
3	select sum(multiSearchAny(URL, ['yandex', 'google', 'yahoo']))	893.58	643.33
4	select sum(multiSearchAny(URL, ['yandex', 'google', 'yahoo', 'pikabu']))	875.59	602.62
5	select sum(multiSearchAny(URL, ['yandex', 'google', 'yahoo', 'pikabu', 'facebook', 'wikipedia', 'reddit']))	794.51	575.28
6	select sum(multiSearchAny(URL, ['yandex', 'google', 'yahoo', 'pikabu', 'facebook', 'wikipedia', 'reddit','yandex', 'google', 'yahoo', 'pikabu', 'facebook', 'wikipedia', 'reddit']))	632.96	576.45
7	select sum(multiSearchAny(URL, ['kvartiry', 'nedvizhimost','kommercheskaya_nedvizhimost','garazhi_i_mashinomesta','doma_dachi_kottedzhi','zemelnye_uchastki', 'komnaty','nedvizhimost_za_rubezhom']))	589.21	575.87
8	select sum(multiSearchAny(URL, ['ут', 'утк', 'утко', 'утконос','enrjyjc', 'utkonos', 'enrjyjc','www', 'http', 'enrfyjc', 'гелщтщы']))	780.46	539.92
9	select sum(multiSearchAny(URL, ['бэбиблок', 'бэбиблог', 'бебиблок', 'бебиблог', 'blog', 'беби блог', 'бэбиблог', 'бб', ',t,b,kju', 'бейбиблог','бейбиблог.ру', ',\',b,kju', 'бэйбиблог', 'бейби блог', 'бэби блок','бэбибл', 'бебиб', 'бебибло']))	355.3	567.27
10	select sum(multiSearchAny(URL, ['fitnes-kluby','sportivnoe-oborudovanie-atributika','krytye-sportivnye-ploshchadki','pejntbol-strajk–i-hard-bol','strelkovye-kluby-tiry','sportivnye-organizatsii','basseyny-plavatelnye','otkrytye-sportivnye-ploshchadki-bazy', 'gornolyzhnye-sklony','pryzhki-s-parashyutom','sportivnye-sektsii', 'yakht-kluby','joga-centry-i-instruktory','bukmekerskie-kontory','skalolazanie-voskhozhdenie-v-gory','fekhtovalnye-kluby','drugoj-sport-i-fitnes','boevye-iskusstva']))	370.97	606.32
11	select sum(multiSearchAny(URL, ['chelyabinsk.74.ru', 'doctor.74.ru','transport.74.ru', 'm.74.ru', '//74.ru/','chel.74.ru', 'afisha.74.ru','diplom.74.ru', 'chelfin.ru','//chel.ru', 'chelyabinsk.ru','cheldoctor.ru', '//mychel.ru','cheldiplom.ru', '74.ru/video', 'market','poll', 'mail', 'conference', 'consult','contest', 'tags', 'feedback', 'pages','text']))	247.18	449.5
12	select sum(multiSearchAny(URL, ['p17266p66989p97b7', 'p17266p66988p285b','p17266p66986pa4e8', 'p15926p65809pbab6','p15926p65810p2672', 'p15926p65811p9afa','p15926p65812p97e3', 'p15926p65813pd214','p15926p65813pd214', 'p15926p65815p350b','p15926p65816pe52c', 'p15926p65814p0cc9','p15926p65817p4cea', 'p15926p65818p9b20','p15926p65860p4435', 'p15926p65861p1f1e','p15926p65862p6f5b', 'p15926p65864pef1c','p15926p64433p9fca', 'p15926p64435p9eb7','p15926p64436p5e2c', 'p14762p59496p5de8','p14762p67782pfc4a']))	754.21	814.96
13	select sum(multiSearchAny(URL, ['вуман','вумен ру женский журнал','вуменфорум', 'вуманжурнал ру','воман','www.woman.ru','вуман ру', 'womanru','женский журнал','форумвумен','devty', 'вумен.ру','вумен руфорум','вуманжурнал','женский форум', 'журнал вумен','цщьфт','женский журналвумен','women', 'woman форум','devtyhe','вуман.ру','женский форум вумен','women.ru женский сайт','форум вуменру','вум','сайт вумен', 'воменс.ру','devfy','вомен','woman.ruжурнал','woman.ru форум', 'вуменру','вуман форум','цщьфтюкг','devfy he','wom','вумен форум новое', 'вумен форумновые','вумэн','форум вуман']))	171.9	489.3

(v20.11.3.3-stable, SKL)

Case Study: textual analytics

SIMD techniques in textual processing

Tokenize problem

- Example:

Text	Delimiter
https://github.com/intel	: / .
regular expression\nmatching library	\s

- Delimiter + Matching
  1. Delimiter Pre-processing

     Normal: [\s], [xxx]

     Conditional: a[xxx]b

     Grouping: “xxx”, (xxx), {xxx}
  2. Word Matching

Hyperscan SIMD usage

1. Multiple literals

   parallel shift-or algorithm
2. One/Two-byte sequence

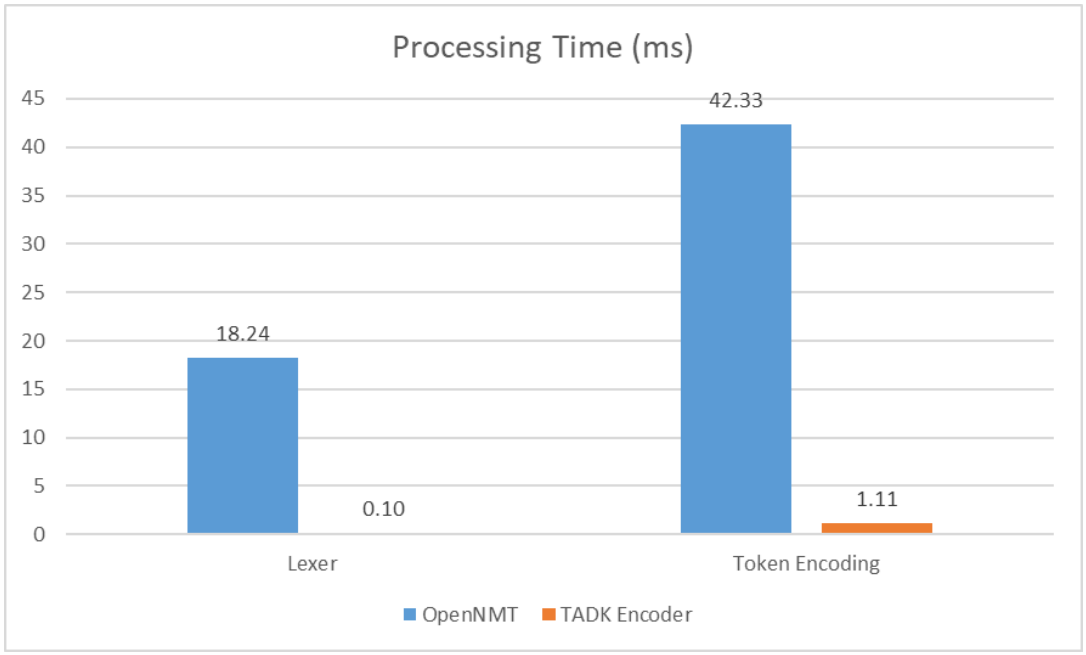
   “f”, “na”
3. Character class

   [\dAEIOUaeiou],
   [\sAEIOUaeiou\x87\x88\x96\x97]
4. Character class at certain positions

   -1: [ABCD], 0: [BCD], 1: [CD], 2: [abcd]
5. NFA/DFA

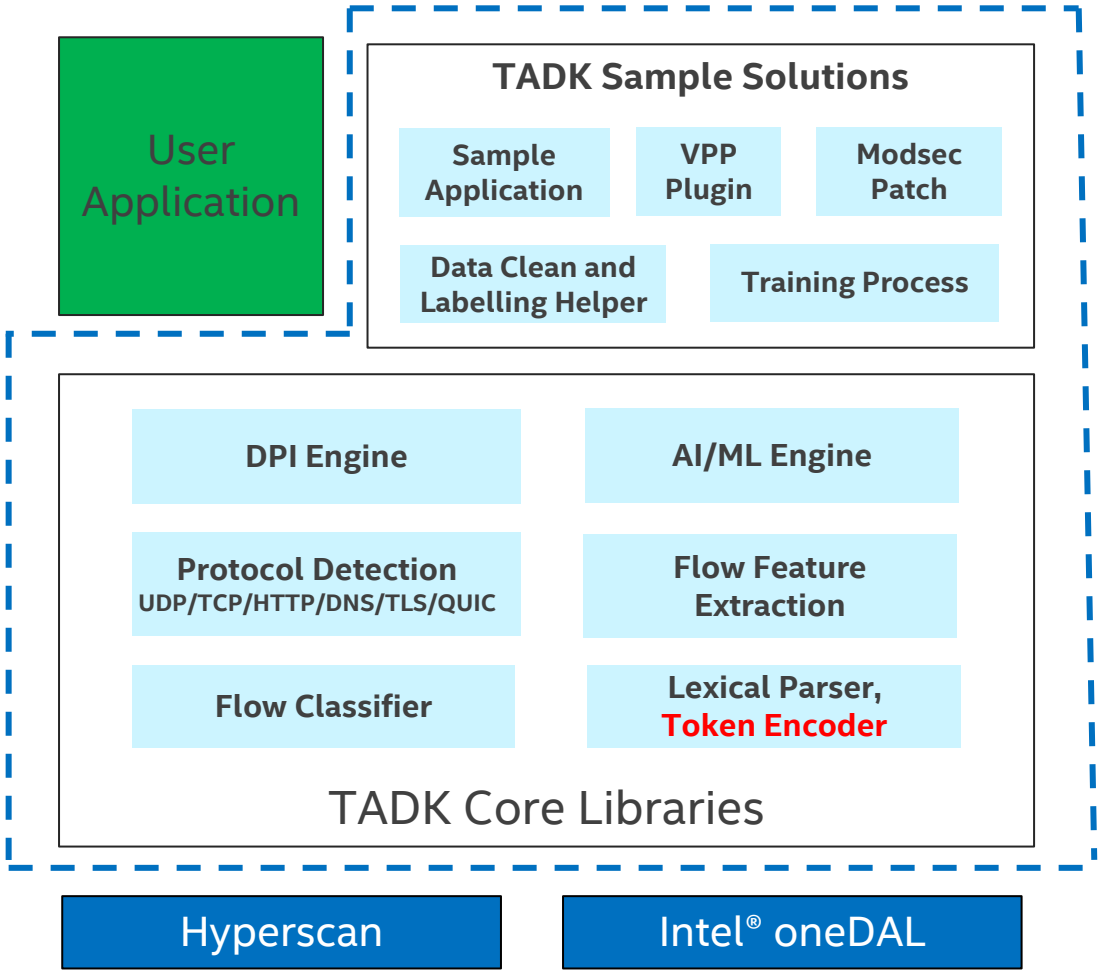
Tokenize problem (1)

- TADK Encoder



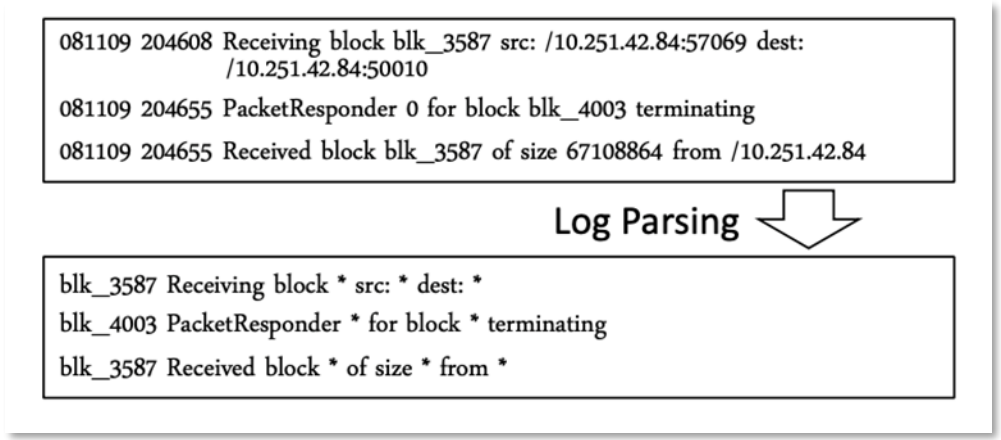
TADK Encoder vs OpenNMT

50.1x boost in Token Encoding
98.5% accuracy (original 99.1%)



Tokenize problem (2)

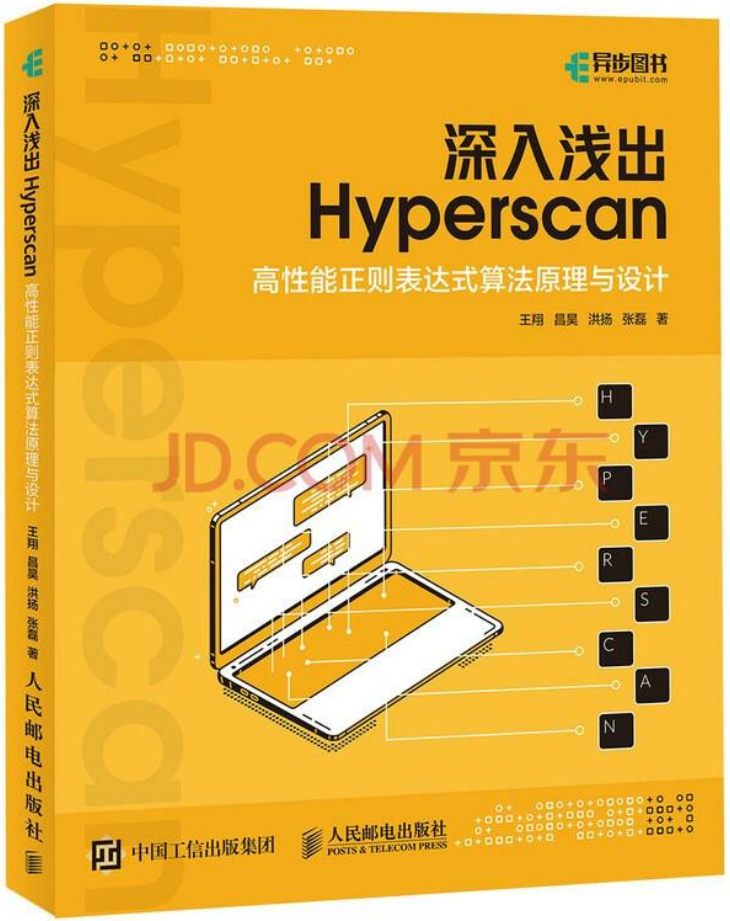
- Log Template Generator



  - Offline: Machine Learning-Clustering
  - Online: Prefix Tree-based (Drain)

    https://github.com/IBM/Drain3 (python-based)

- Our prototype of Drain
  - Dual prefix-tree
  - C-based
  - ~10x boost against Drain3 on a 20MB log



深入浅出 Hyperscan：高性能正则表达式算法原理与设计
(人民邮电出版社)

https://item.jd.com/13387744.html

- 介绍正则表达式的背景知识以及字符串匹配和正则匹配的各类常规算法；
- 探索Hyperscan算法库的功能特性和与业界广泛使用的较为成熟的正则匹配算法库的比较；
- 阐释Hyperscan总体设计原则，并详细描述了对正则表达式的全新解构思路；
- 展现经过解构后的正则表达式模型的实现方法，并详细描述了优化手段；
- 介绍Hyperscan使用过程中性能调优的若干原则与技巧；
- 展示Hyperscan与多种现实应用的整合案例；

Contact us

- Community:
  - https://github.com/intel/hyperscan
- Email:
  - hyperscan@lists.01.org


intel