NEXTMOVE SOFTWARE

MANAGING AND SEARCHING ONE TRILLION COMPOUNDS

John Mayfield and Roger Sayle
NextMove Software, Cambridge, UK

"Yes, and you've been playing without an opponent, which is, as you may have guessed... against the rules."
- Anton Ego, Ratatouille

ULTRA-LARGE CHEMICAL DATA?

- 🙈 Ignore
- 🐘 Scale
  - … *more and distributed hardware/solutions*
- 🚀 Optimise
  - … *improved algorithms and data structures*

ARTHOR

Traditional fingerprint based similarity and substructure screening

https://nextmovesoftware.com/arthor

SMALLWORLD

Graph-Edit-Distance (GED) similarity and indexing

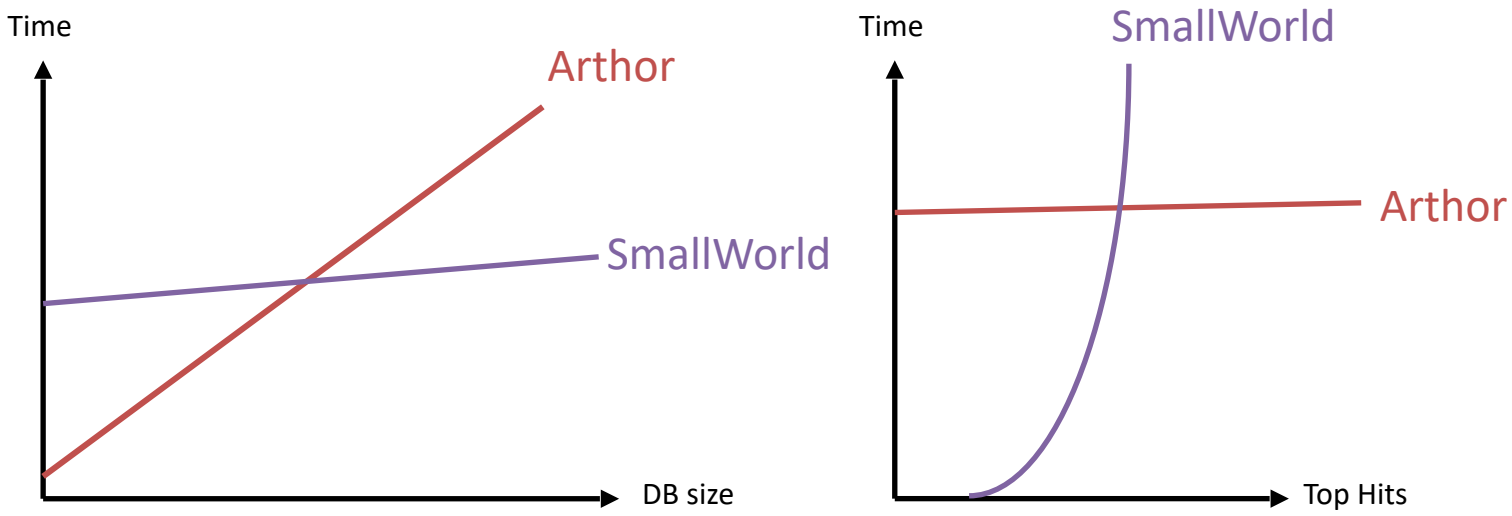
https://nextmovesoftware.com/smallworld

UCSF/Zinc Hosted:

https://arthor.docking.org

https://sw.docking.org

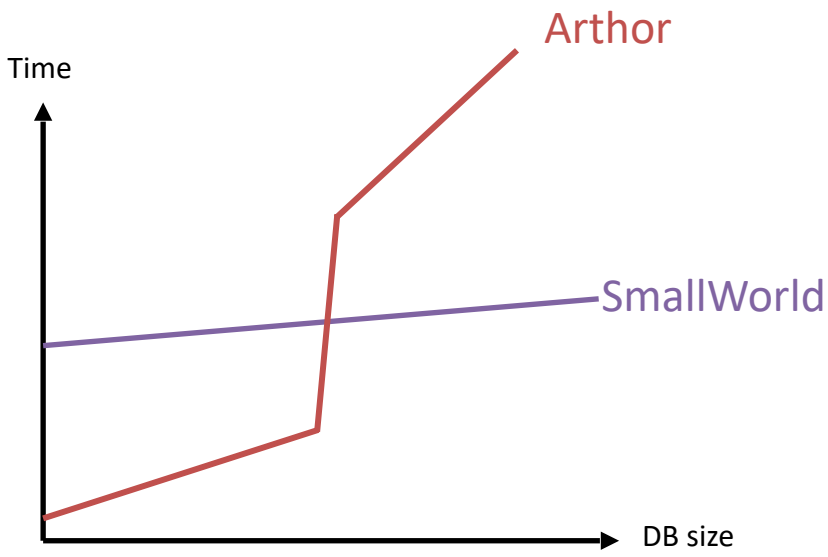
SCALING THEORETICAL

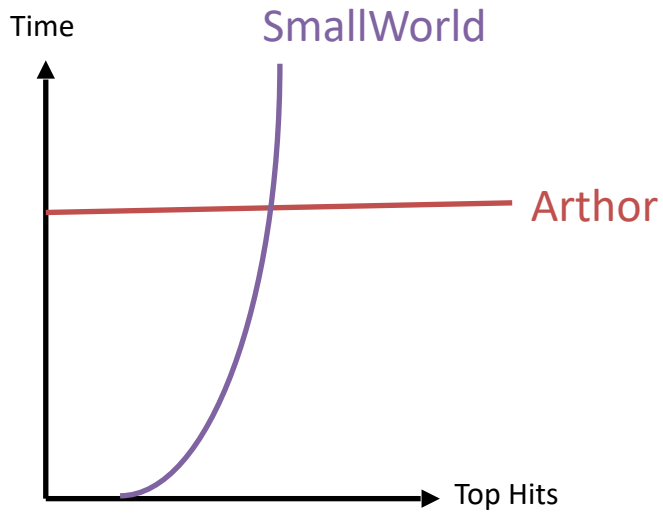


SmallWorld good for close analogues

Arthor good for filtering and/or scoring against everything

SCALING REALITY



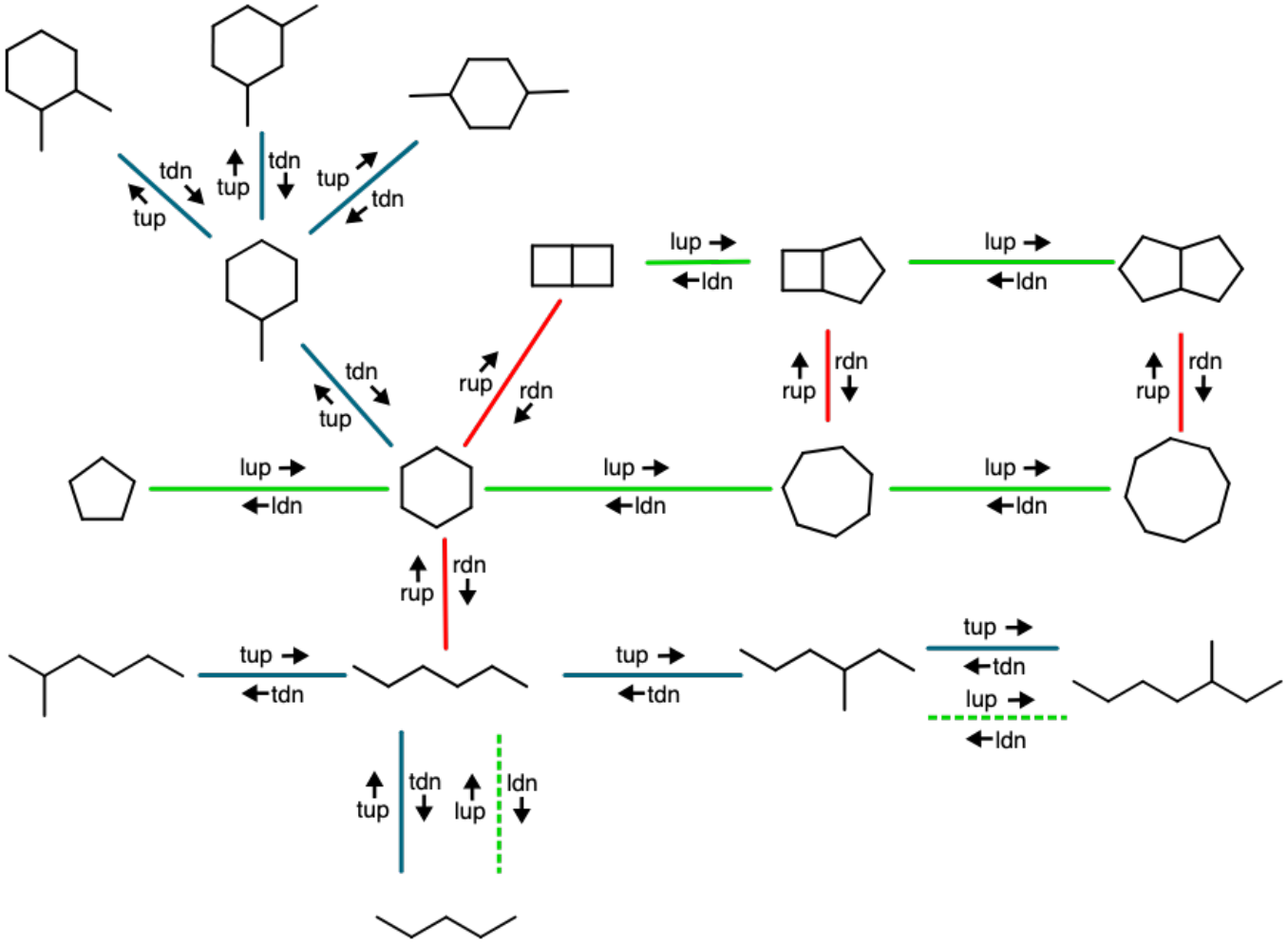


Arthor **much faster** when memory fits in **RAM**

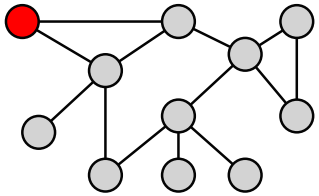
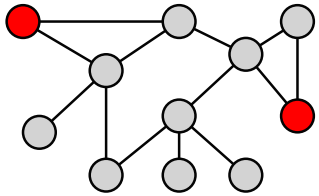
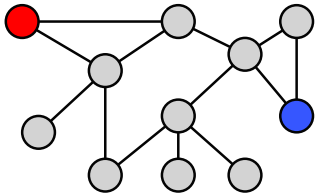
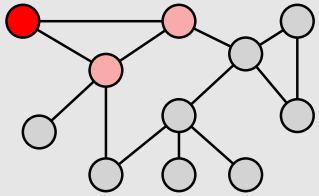
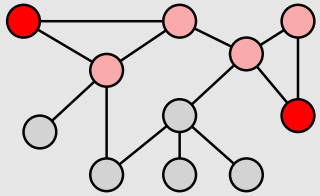
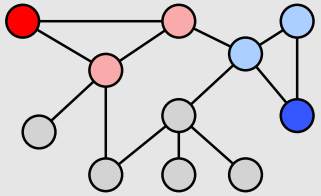
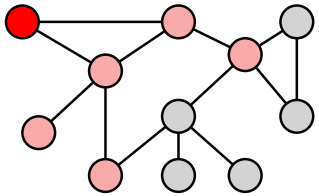
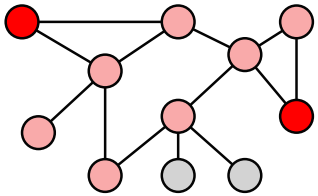
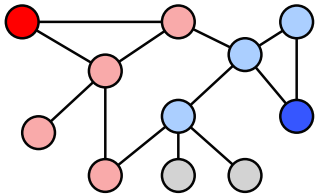
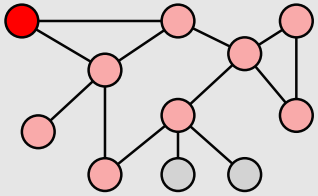
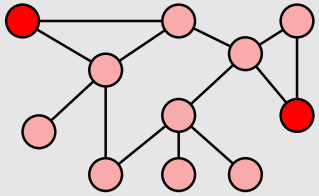
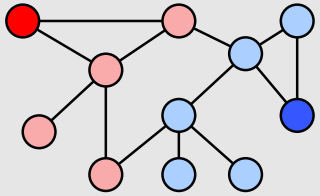
… otherwise must read from disk or worse network!

SMALLWORLD

CHEMICAL SPACE INDEX

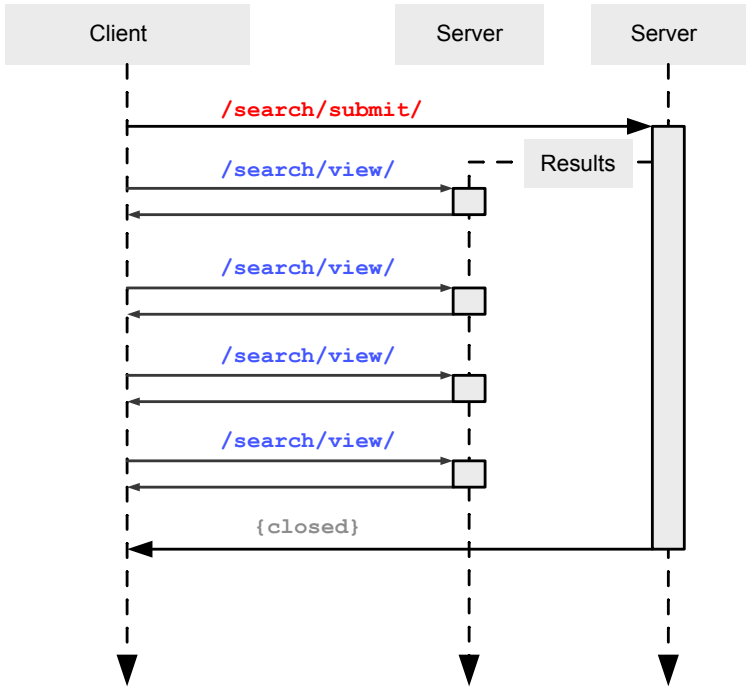


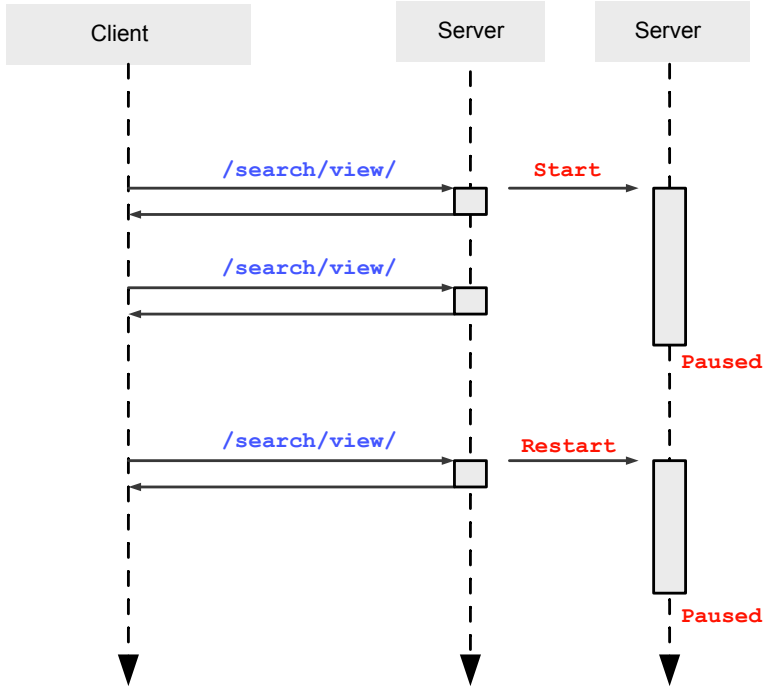
SMALLWORLD SEARCH OPTIONS

	Single Start *Distance from the start location*	Multiple Start ***Closest** distance from the **any** start location*	Multiple Set, Multiple Start *Closer to the **selected** or **rejected** start location*
dist=0			
dist=1			
dist=2			
dist=3			

(future release)

SMALLWORLD NEW WEB API





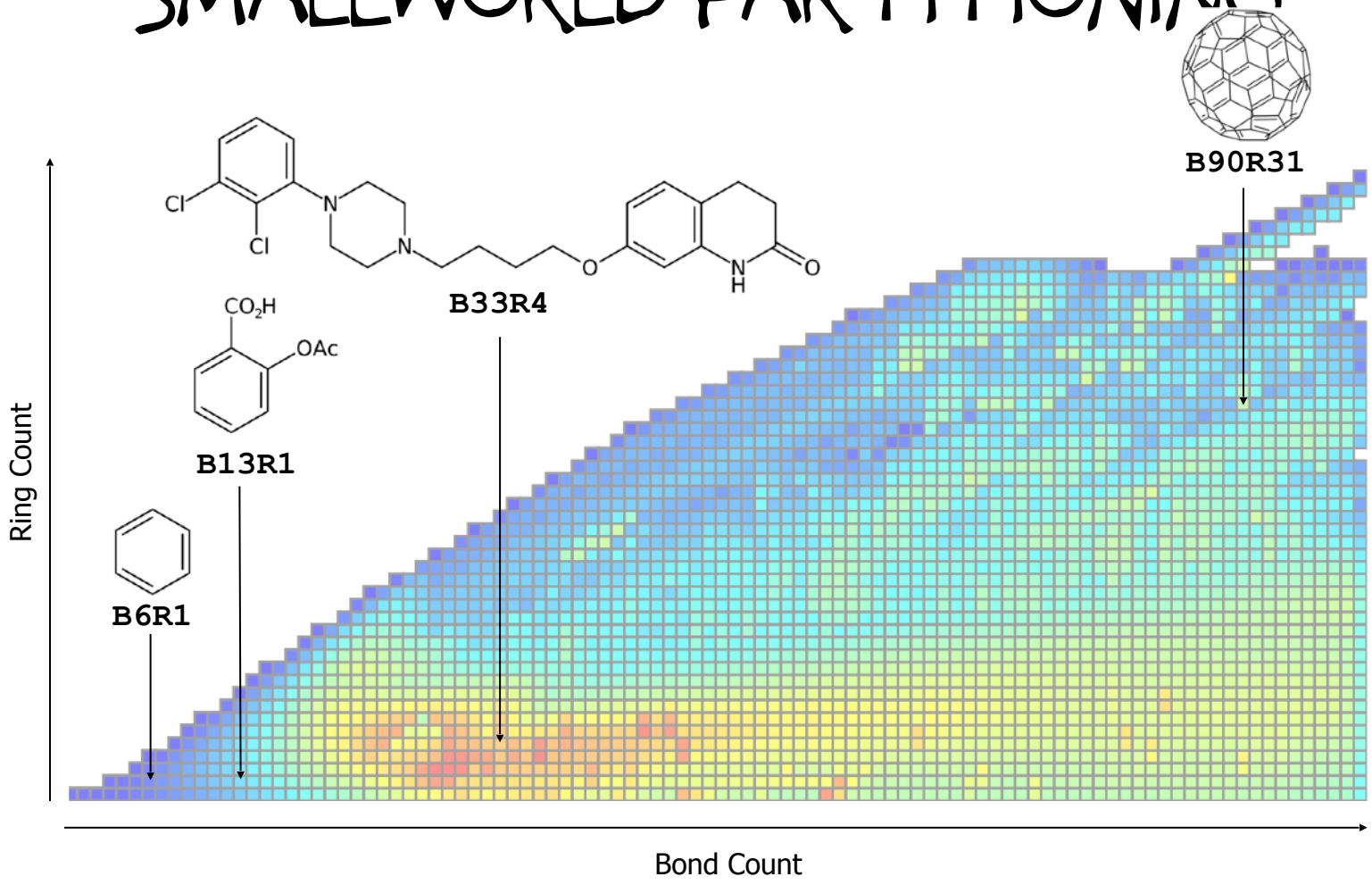
(future release)

- Backwards compatible
- Push (sockets) vs Pull (polling)
- Async vs sync options

SMALLWORLD INDEX GROWTH

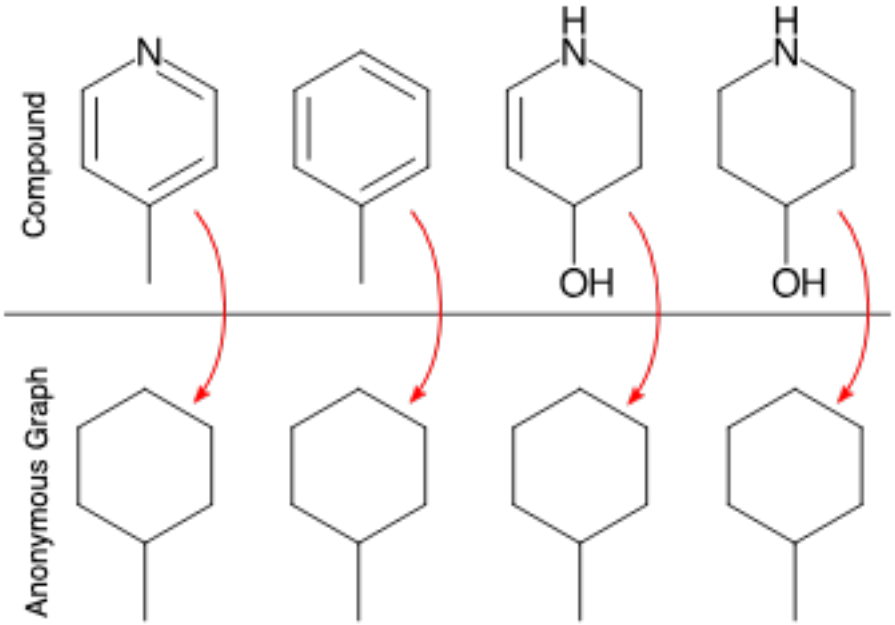
	Nodes (SMILES)			Edges		
	Count (billion)	Space (TB)	B/mol	Count (trillion)	Space (TB)	B/edge
2016	7	-	-	-	-	-
2017	69	-	-	-	-	-
2018	125	-	-	-	-	-
2019	230	2.6	~11.3	2.75	20.6	~7.49
2020	471	6.0	~12.7	6.12	44.6	~7.28
2021	675	8.9	~13.1	9.16	27.6	~3.01
2022	790	9.8	~12.3	9.98	26.8	~2.68
2023	**1,008**	**13.1**	**~13.0**	**11.2**	**30.1**	**~2.69**

SMALLWORLD PARTITIONING



Index **anonymous graphs** of chemical compounds in buckets based on number of bonds and rings (**B#R#**)

SMALLWORLD ANONYMOUS GRAPHS



Many-to-one mapping from *real* compounds to their anonymous graphs (SMILES)

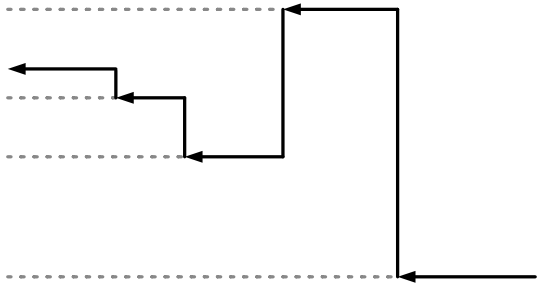
- July 2023 the index reached **1 trillion** graphs
- **13.11 TB** ~ 13.11 bytes per entry

How to efficiently ***map*** compounds to their **B#R#.#**?

****1*****1 B7R1.3**

DB.SMI (2014-2019)

```
**(*)(*)*1**1 1
**(*)**1**1 2
**(*)*1(**1)* 3
**(*)*1***1 4
***(*)*1**1 5
*****1**1 6
****1(**1)* 7
****1***1 8
****1**1* 9
***1(***1)* 10
***1(**1)(*)* 11
***1(**1)** 12
***1(**1*)* 13
***1*(*1*)* 14
***1**(*1)* 15
***1****1 16
***1***1* 17
***1**1(*)* 18
***1**1** 19
**1(****1)* 20
**1(***1)(*)* 21
**1(**1(*)*)* 22
**1(**1)(*)(*)* 23
**1*(*1(*)*)* 24
**1**(*1)(*)* 25
**1**(*1*)* 26
**1***(*1)* 27
**1*****1 28
**1****1* 29
**1***1(*)* 30
**1**1(*)(*)* 31
**1**1*(*)* 32
*1******1 33
```



Looking up B#R#.# id

Lexicographically sorted list **SMILES** and **ID**

Disk/MMAP binary search to find the key/value pair

Variable size lines

SMIZIP

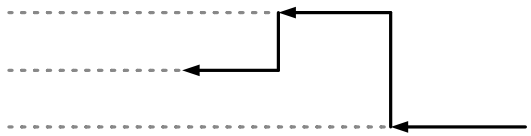
SMILES entries can be compressed with a set of anonymous SMILES specific multi-grams

- Replaces multi-character substrings with a single byte
- Maintain random access to records

https://www.daylight.com/meetings/mug01/Sayle/SmiZip/index.htm

DB.SWIDX - IMPLICIT BINARY TREE (2019)

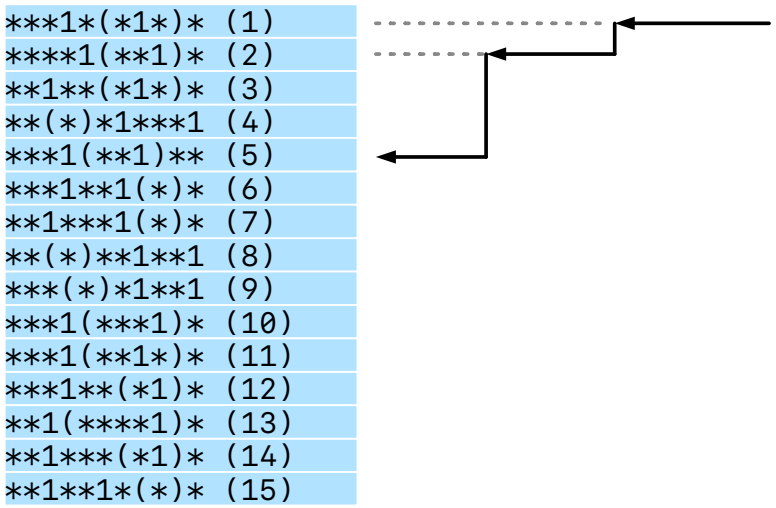
```
*****1**1
****1***1
****1**1*
***1****1
***1***1*
***1**1**
**1*****1
**1****1*
*1******1
**(*)**1**1 (1)
**(*)*1***1 (2)
***(*)*1**1 (3)
****1(**1)* (4)
***1(***1)* (5)
***1(**1)** (6)
***1(**1*)* (7)
***1*(*1*)* (8)
***1**(*1)* (9)
***1**1(*)* (10)
**1(****1)* (11)
**1**(*1*)* (12)
**1***(*1)* (13)
**1***1(*)* (14)
**1**1*(*)* (15)
**(*)(*)*1**1
**(*)*1(**1)*
***1(**1)(*)*
**1(***1)(*)*
**1(**1(*)*)*
**1*(*1(*)*)*
**1**(*1)(*)*
**1**1(*)(*)*
**1(**1)(*)(*)* 23
```



Partitioning by length to binary search on “fixed-size keys”

Fixed size key, **ID** can be calculated by the storage position (byte offset) in the file

DB.SWIDX - IMPLICIT BINARY TREE (2019)



2019: Reorder the keys to be in an implicit binary tree

Multiplicative binary search

Children: 2i+1, 2i+2

Root of the tree stays *hot* in cache

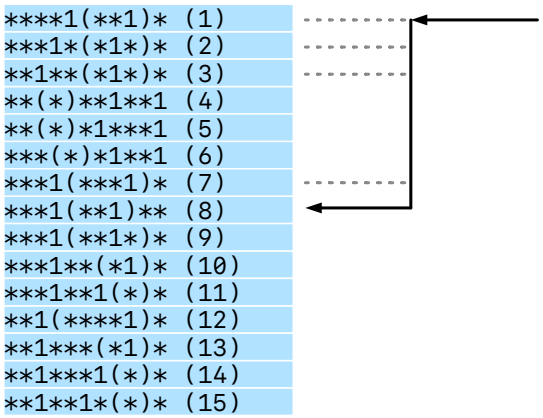
After root, reads become much sparser,

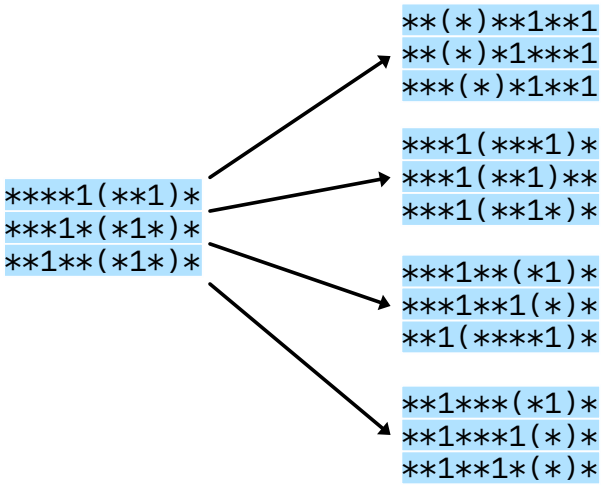
log2(30 billion)

34.8 reads

https://en.wikipedia.org/wiki/Multiplicative_binary_search

DB.SWIDX - IMPLICIT BTREE (2022)





2022: Implicit btree stores blocks of keys and a fanout of more than two (binary tree)

Better disk locality, particularly if we align our “block” and “memory page” size:

page = 8192, keylen = 20

log(30 billion) / log(floor(8192/20)

4.1 reads

Well known in relational databases but as an implicit index 0 overhead (no pointer storage/book keeping)

DB.SWIDX - IMPLICIT BTREE (2022)

Almost **7x** faster and **14%** of the data read

Type	Time	Disk I/O
db.smi	1060s	924 MB
db.swidx (sorted)	333s	682 MB
db.swidx (binary-impl)	528s	610 MB
db.swidx (btree-impl)	148s	136 MB

Benchmark: Lookup 28,000 values in B34R4 from disk (uncached)

DB.SWIDX - IMPLICIT BTREE (2022)

Batch lookup **200x** faster:

- sort and portion query keys based on the current block
- issue I/O advise* to the Kernel before accessing children

Type	One	Batch	Disk I/O
db.smi	1060s	-	924 MB
db.swidx (sorted)	333s	40s	682 MB
db.swidx (binary-impl)	528s	16s	610 MB
db.swidx (btree-impl)	148s	**5s**	136 MB

*posix_fadvise() and/or madvise()

PRACTICAL IMPLICATIONS (SWMAP2)

ChEMBL 29 (2 million structures)

Disk	Cached	Before	After
NVME	Cold	11m 45s	1s
HDD	Cold	5h 19m	15s
NVME	Hot	20s	1s
HDD	Hot	26s	14s

Enamine (1 billion)

Disk	Cached	Before	After
NVME	Cold	107m 5s	927s
NVME	Hot	0h 59m	760s

ARTHOR

ARTHOR SIMILARITY

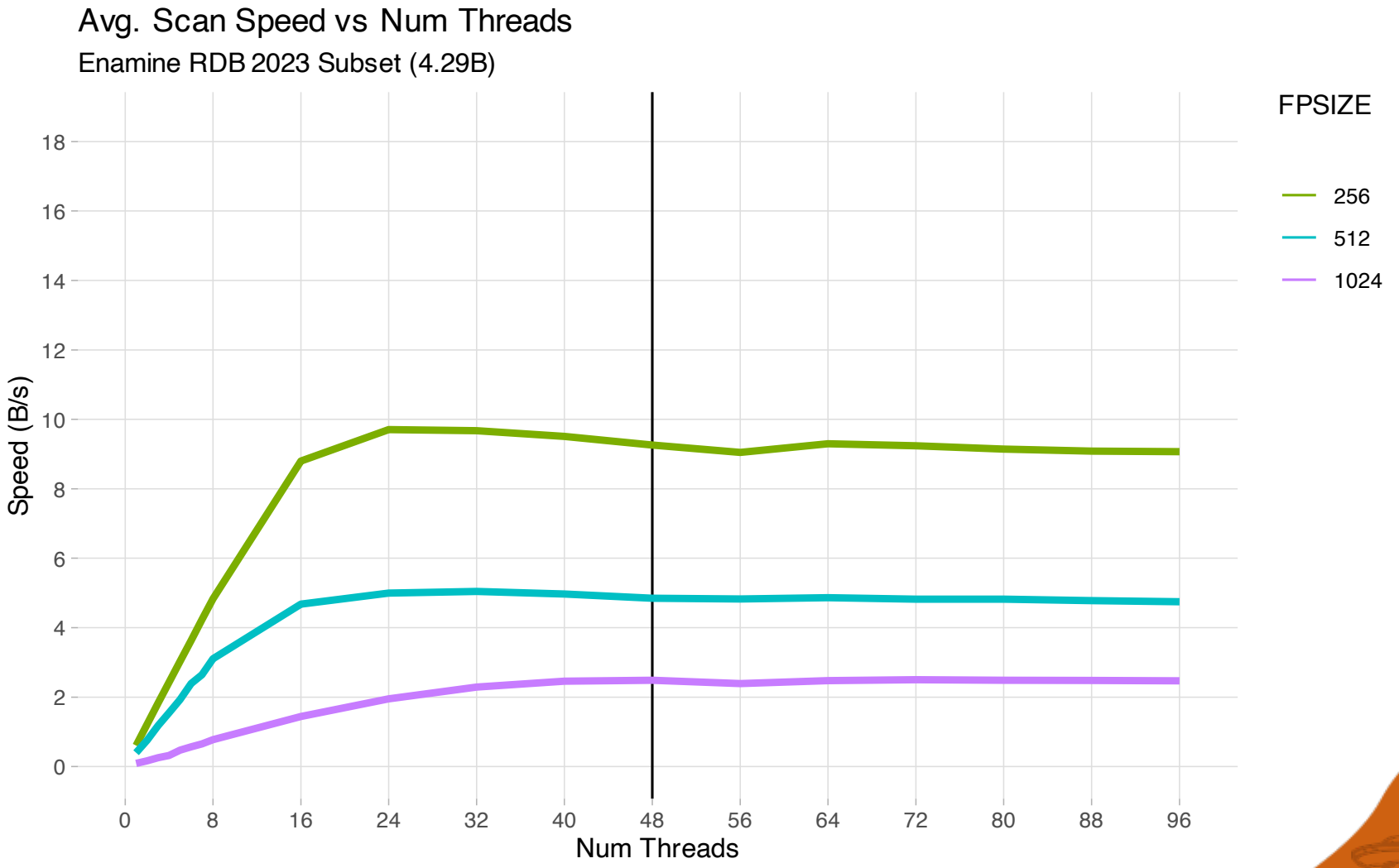
Search Time ≈ Index Size

- 1 billion 256-bit fingerprints: **32GB**
- 2 billion 256-bit fingerprints: **64GB**
- 1 billion 512-bit fingerprints: **64GB**
- 2 billion 1024-bit fingerprints: **256GB**
- 37 billion 256-bit fingerprints: **1184GB**

Ideally bottlenecked by **memory**, **disk**, **network**:

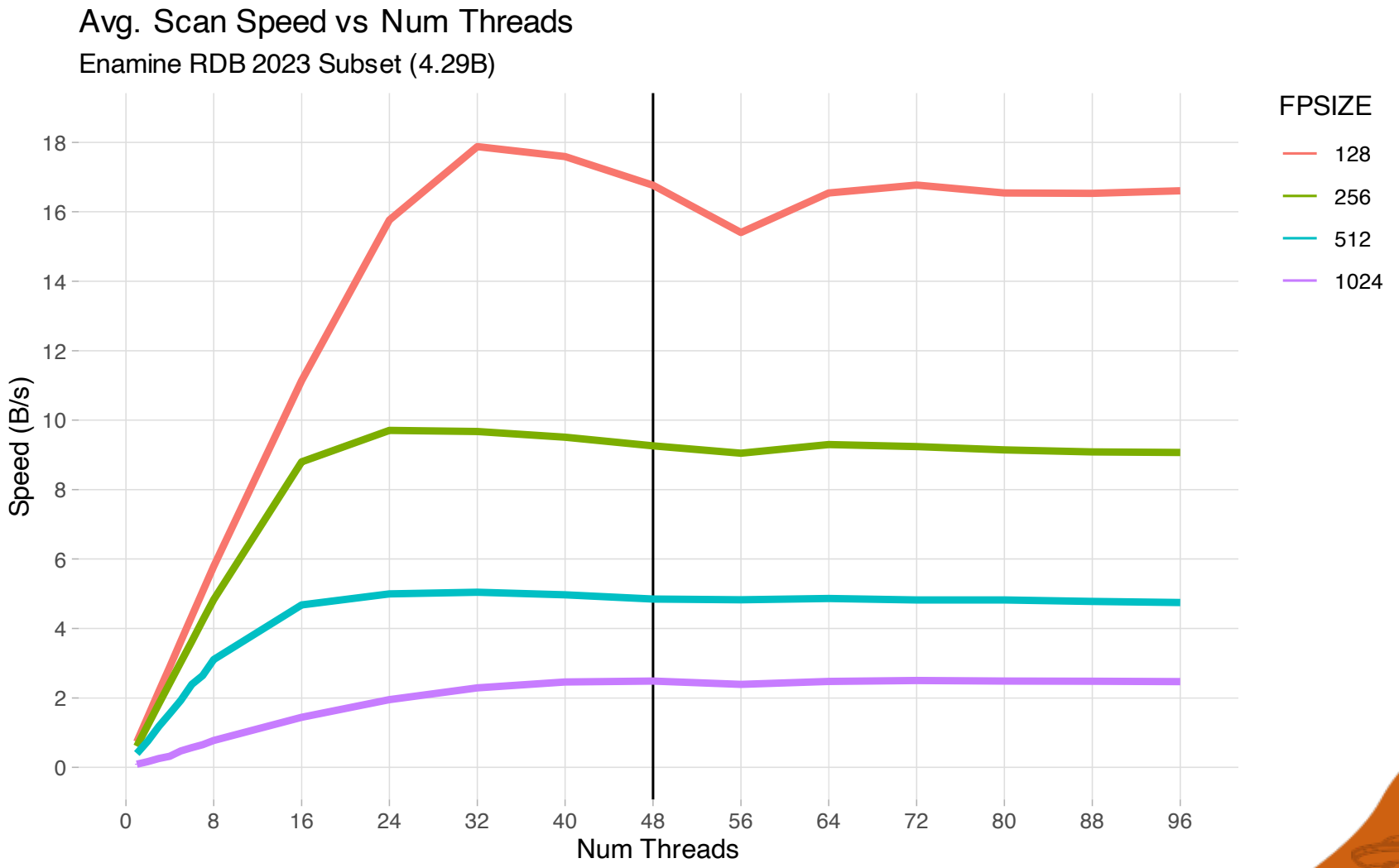
- 1TB @ 1Gb/s Network: 2.2 hours
- 1TB @ 150MB/s HDD disk: 1.85 hours
- 1TB @ 3GB/s NVME: 5.5 mins
- 1TB @ 10Gb/s Network: 13.3 mins

SPEED VS PARALLELISATION VS FPSIZE



Queries: 19 top selling small molecules, ECFP4 256-bit
Hardware: 2x AMD EPYC 7443, 1TB DDDR4 @ 3200 MHz

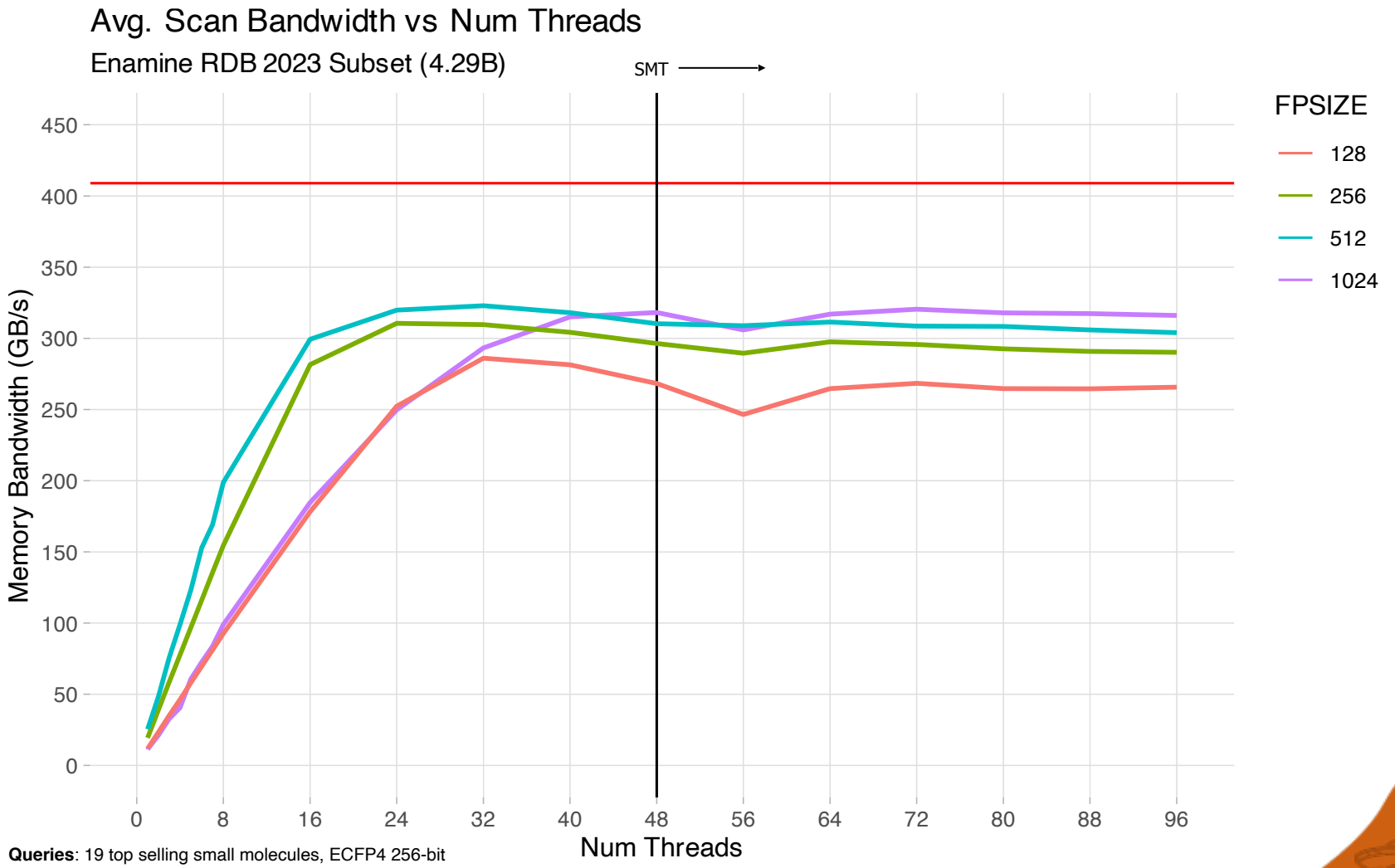
SPEED VS PARALLELISATION VS FPSIZE



Queries: 19 top selling small molecules, ECFP4 256-bit

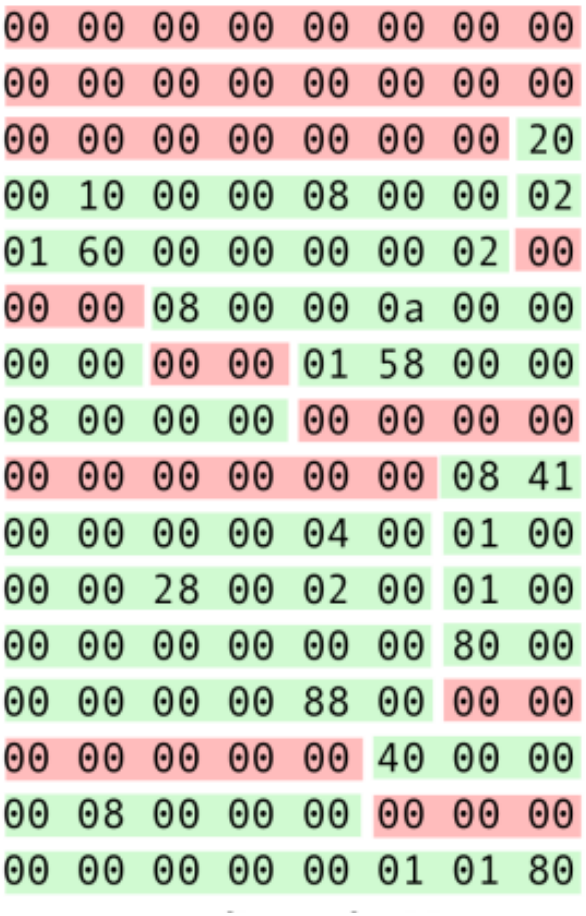
Hardware: 2x AMD EPYC 7443, 1TB DDDR4 @ 3200 MHz

BANDWIDTH VS PARALLELISATION VS FPSIZE



Queries: 19 top selling small molecules, ECFP4 256-bit
Hardware: 2x AMD EPYC 7443, 1TB DDDR4 @ 3200 MHz

ARTHOR 1.0 INSIGHT



Only need to test the 1's set in the query to get the intersect*, skip 0's

- cache/memory paging means they still get read

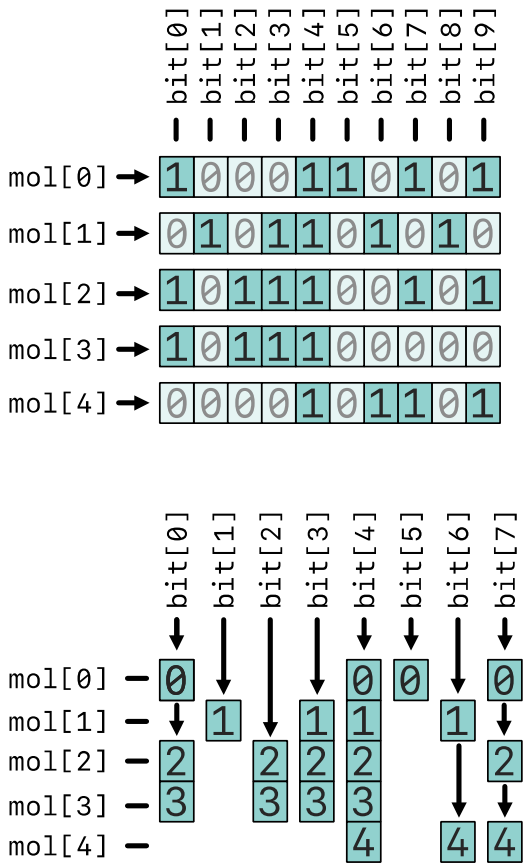
Consider a 1024-bit query with a single bit:

- 16x 64-bit POPC instructions or
- 1x BT (bit-test) instruction

Implemented as x86-64/ARM Just-In-Time (JIT) compiler

*Fingerprints stored in cardinality (POPC) buckets, can compute the Taniomoto with just the intersection

ARTHOR 4.0 INSIGHT



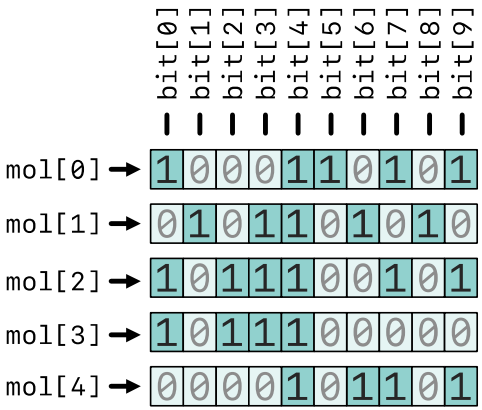
An inverted index:

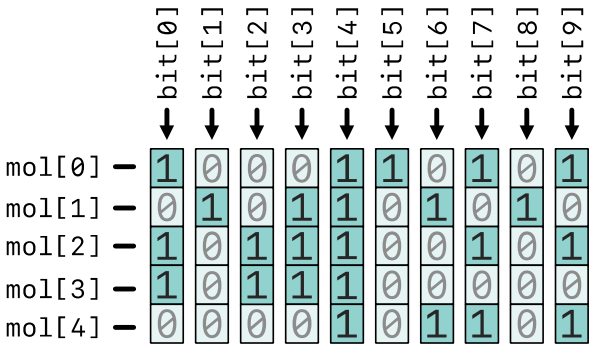
Rather than have the bits for each molecule we can have ***the molecules for each bit***

Compression techniques balance space vs decoding speed:

- list of integers
- list of gaps (varbyte)
- bitmap

ARTHOR 4.0 INSIGHT





Storing as a bitmap takes identical space (the transpose).

However now we only ever access the bits we need to check (those set in the query).

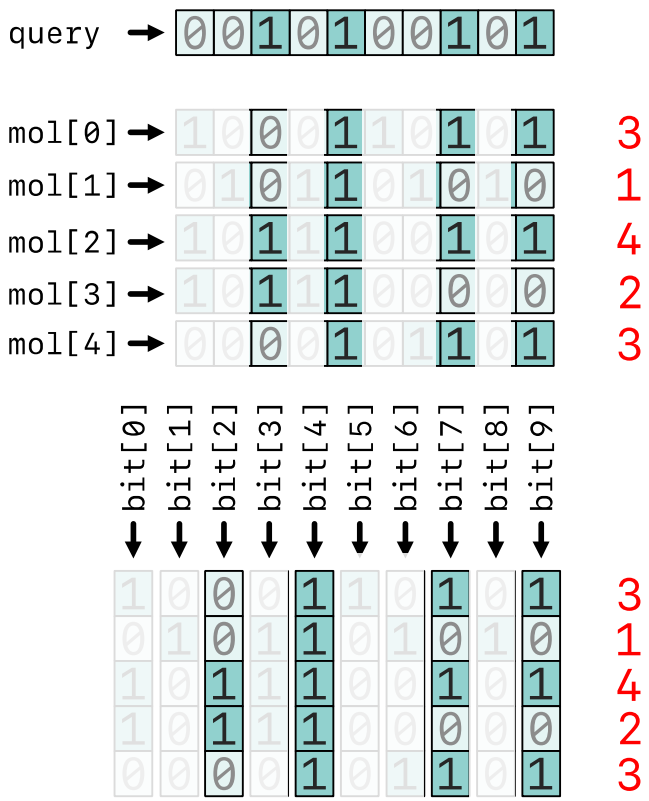
A query with ½ the bits (ρ0.5) in theory does ½ the memory accesses or I/O

HOW MANY BITS IN A QUERY?

Query	256-bit		1024-bit		2048-bit	
	POPC	%	POPC	%	POPC	%
Abilify	50	19.53	53	5.18	55	2.69
Acetaminophen	22	8.59	24	2.34	24	1.17
Androgel	41	16.02	43	4.20	43	2.10
Budesonide	62	24.22	67	6.54	67	3.27

- The number of bits is the query does not scale linearly with FP size
- For circular FP typically (small) fraction of the database

ARTHOR 4.0 INSIGHT



Conventional wisdom:

- Intersect / Union / Not operations
- Non-sequential mem. accesses
- Only v. large/sparse fingerprints
- Use posting-list compression
- Too much overhead

Need efficient method of ***summing*** the occurrences in the inverted bitmaps

BIT TWIDDLING

```
for (unsigned int bitmap : bitmaps) {
  unsigned int *dst = common;
  for (unsigned int w=0; i<len; i++) {
    unsigned int word = bitmap[w];
    *dst++ = word & 0x01010101;   // 0,8,16,24
    *dst++ = (word>>1) & 0x01010101; // 1,9,17,25
    *dst++ = (word>>2) & 0x01010101; // 2,10,18,26
    *dst++ = (word>>3) & 0x01010101; // 3,11,19,27
    *dst++ = (word>>4) & 0x01010101; // 4,12,20,28
    *dst++ = (word>>5) & 0x01010101; // 5,13,21,29
    *dst++ = (word>>6) & 0x01010101; // 6,14,22,30
    *dst++ = (word>>7) & 0x01010101; // 7,15,23,31
  }
}
```

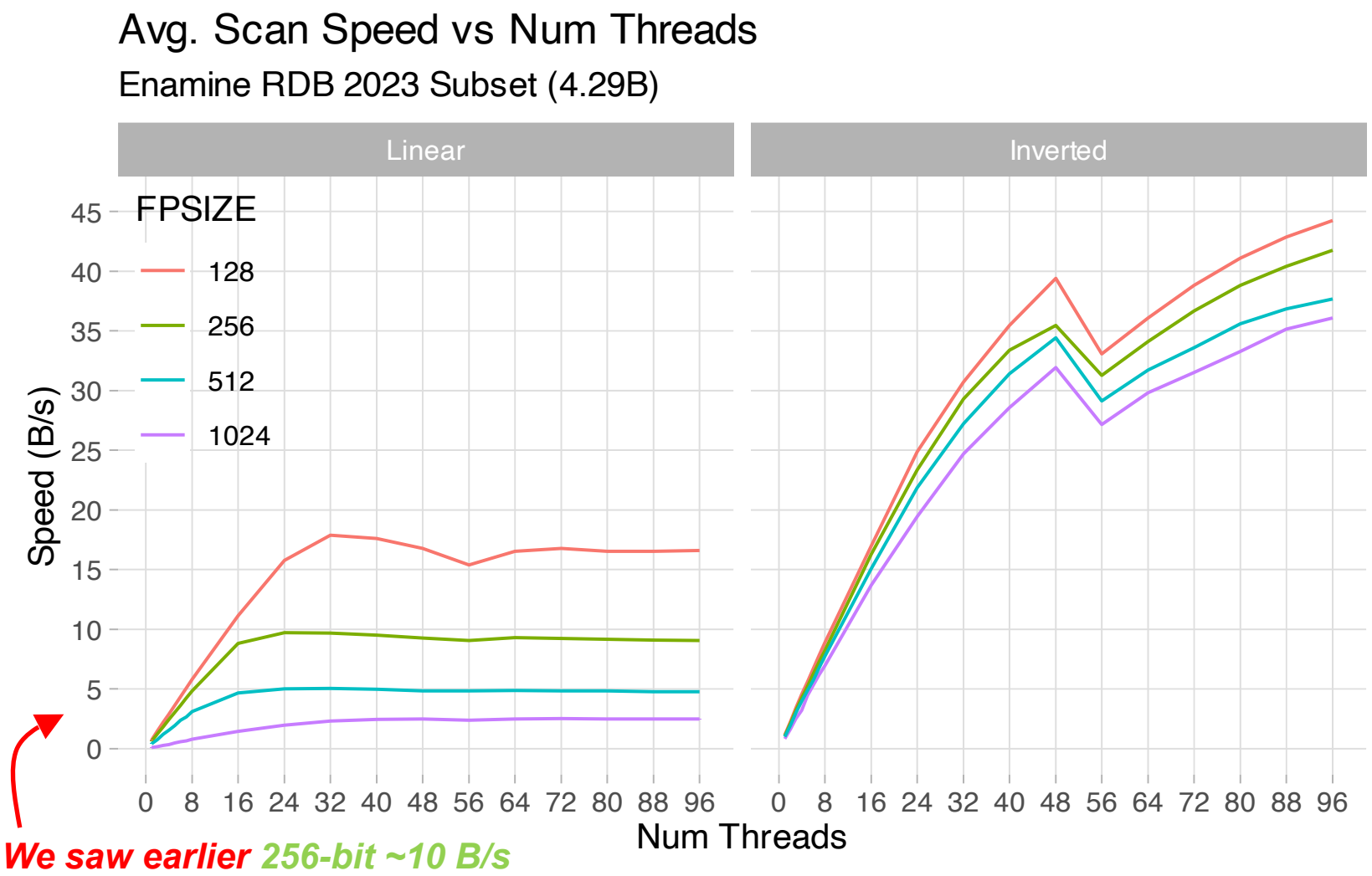
Single instruction, multiple data (**SIMD**) technique can add 4x 8-bits at once in a 32-bit register.

Easy to use 128, 256, and even 512-bit registers

Permute the storage order such that the results are **unshuffled**

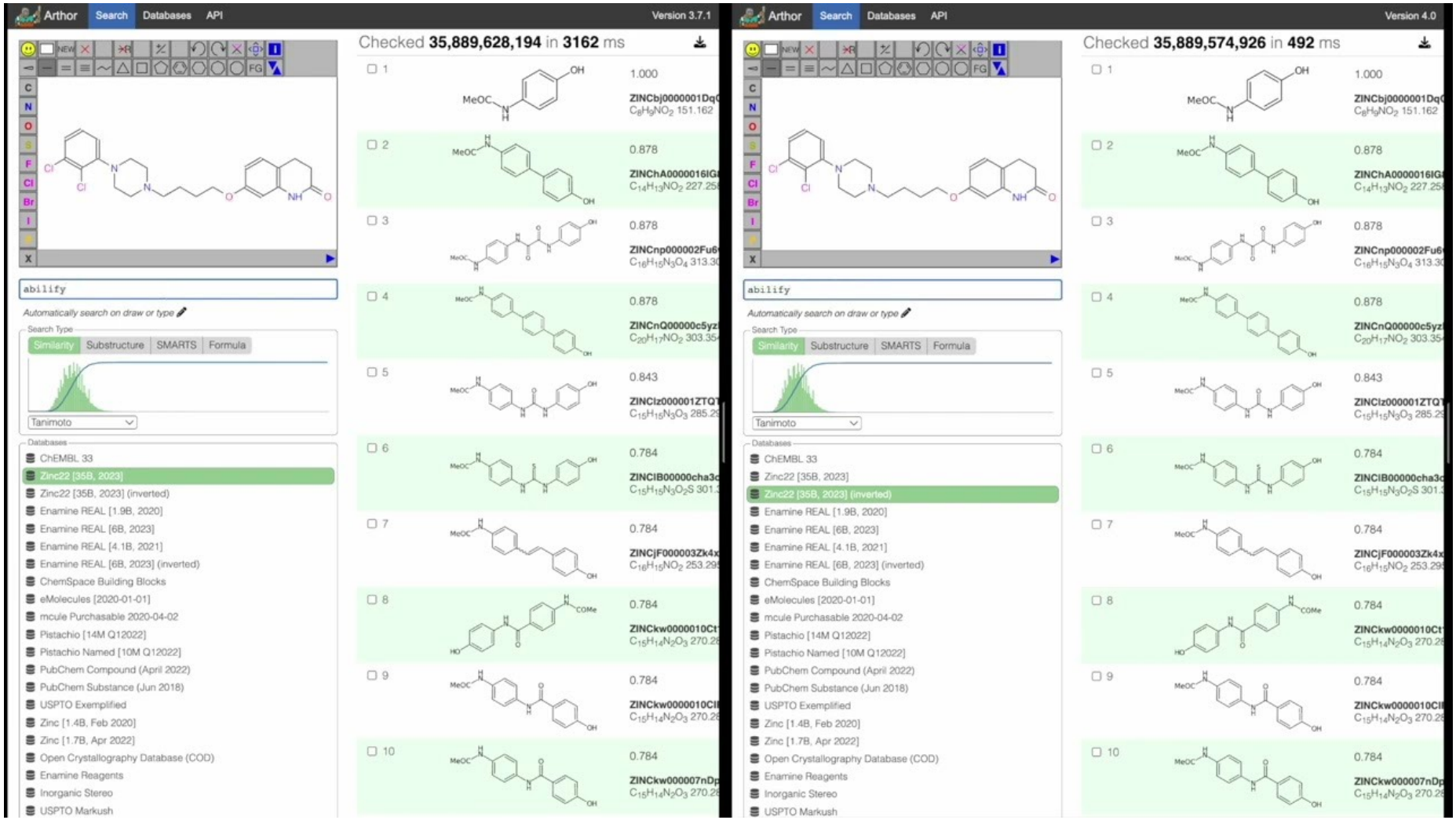
F1NGERPR1NTS: Processing just the bits you need and none of the **1**s you don't (PDF)

THREAD SCALING VS FP SIZE



We saw earlier 256-bit ~10 B/s

Queries: 19 top selling small molecules, ECFP4 256-bit
Hardware: 2x AMD EPYC 7443, 1TB DDDR4 @ 3200 MHz



https://youtu.be/PtKkS6OX8Ks

EXTRA

HIGH PERFORMANCE I/O

"Any sufficiently advanced technology is indistinguishable from magic"
– Arthor C. Clarke

"Any technology distinguishable from magic is insufficiently advanced"
– Roger Sayle's Corollary

Modern hardware is awesome and getting better:

NVME drives:

- PCI 3.0 x4 3500 MB/s Samsung PM1725b
- PCI 4.0 x4 7000 MB/s Intel P5316
- PCI 5.0 x4 14,000 MB/s Kioxia CM7

Theoretical limits

- PCI 3.0 x16 15.754 GB/s
- PCI 4.0 x16 31.508 GB/s
- PCI 5.0 x16 63.015 GB/s

A HOT ROD

Intel 30 TB D5-P5316 7000 MB/s

4x30TB = 120TB

4x7GB/s = 28GB/s

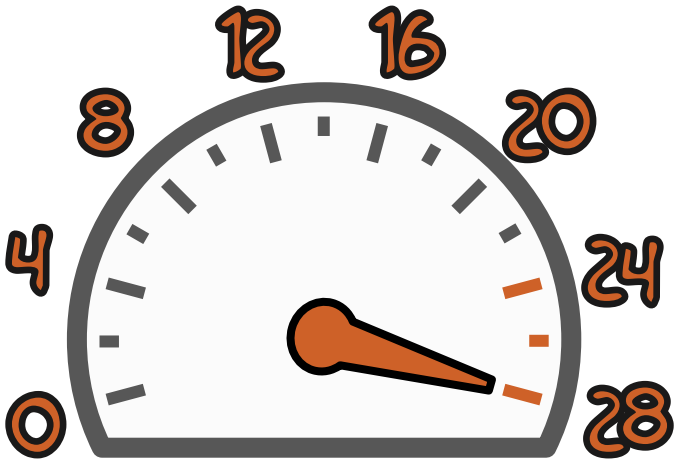
HighPoint SSD7580 HBA/AIC (optional)

Software (mdadm) Raid 0

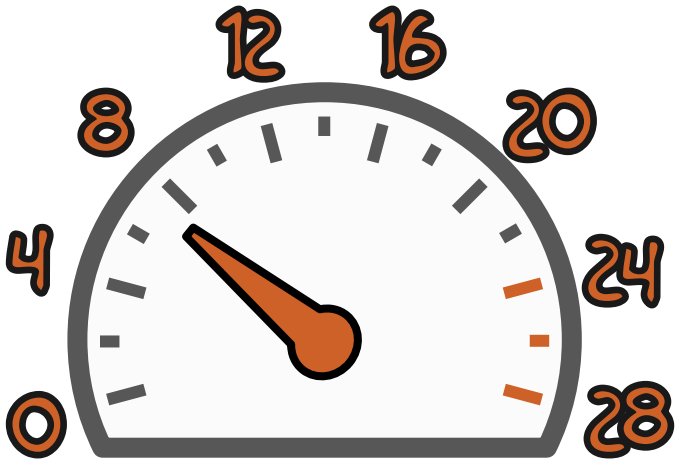
Halved in price since 2022

61TB news recently: D5-P5336

TEETHING ISSUES



Theoretical: **28GB/s**



Observed: **7.8GB/s**

```
$ fio engine=io_uring depth=32 buffer=1MB 60s
IOPS=7490, BW=7491MiB/s (7855MB/s)
```

Flexible IO Tester (fio): https://fio.readthedocs.io/en/latest/fio_doc.html

NUMA NUMA, OH

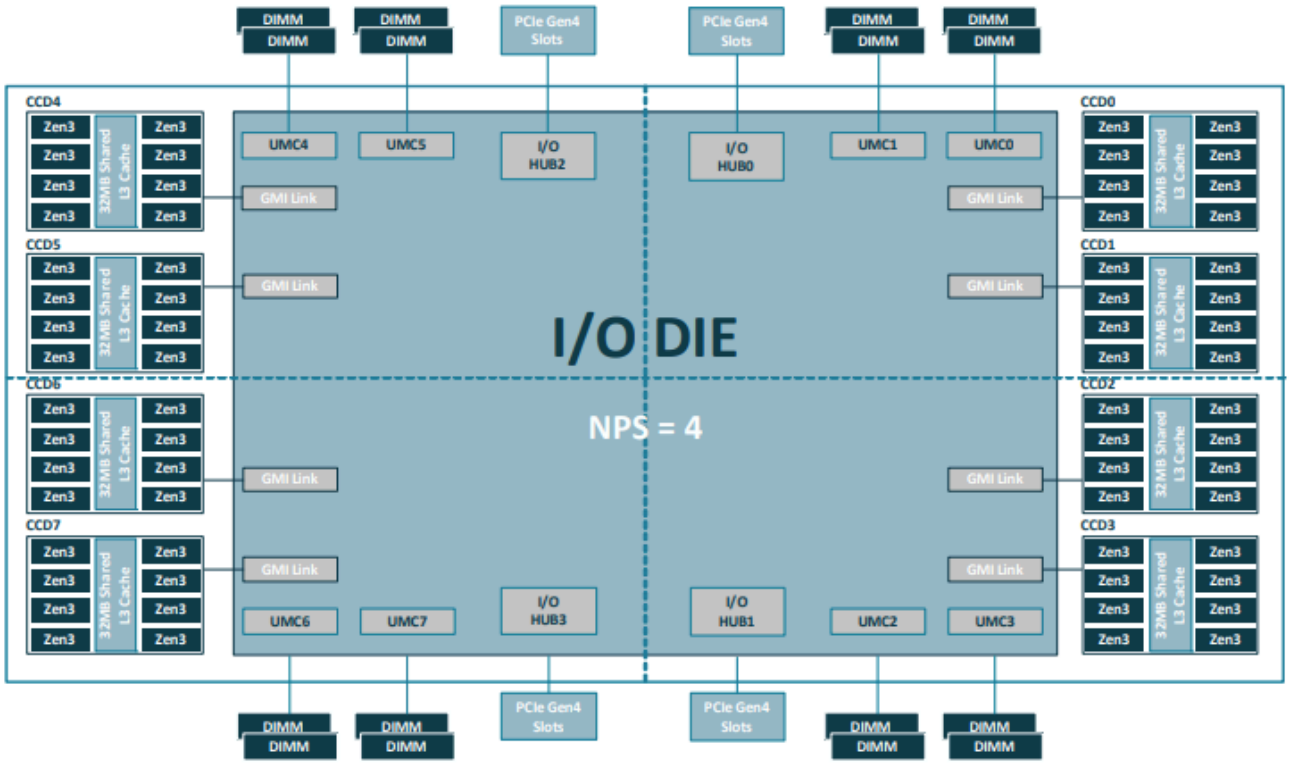
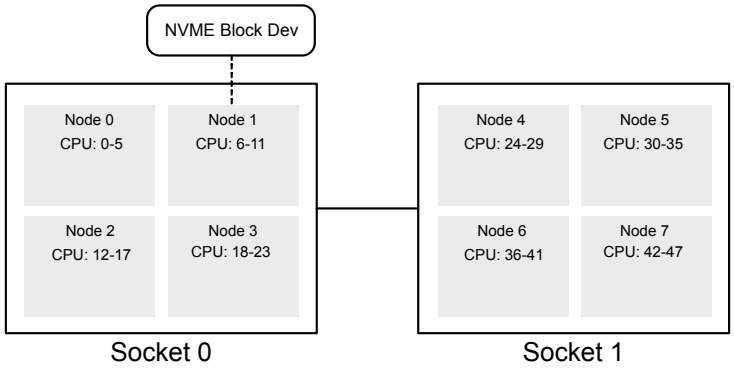
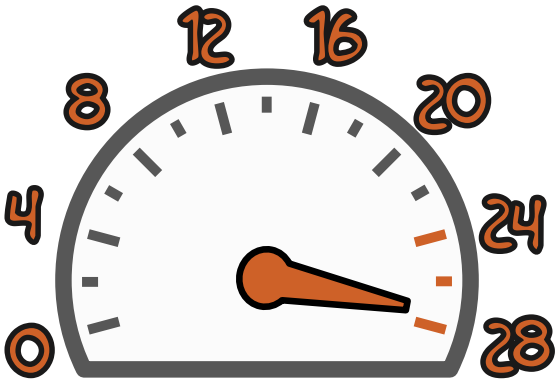


Figure 1-5: EPYC 7003 System on Chip (SoC): 8 CCDs and central IOD

AMD HPC Guide: https://www.amd.com/en/server-docs/high-performance-computing-hpc-tuning-guide-for-amd-epyc-7003-series-processors

NUMA NUMA, OH





Observed: **27.7GB/s**

By default the CPU is configured to appear as a single NUMA node-per-socket (NPS).

NPS=1

8 memory channels per node

Can only pin to socket

NPS=4

2 memory channels per node

Better locality/latency

Can pin to CCD

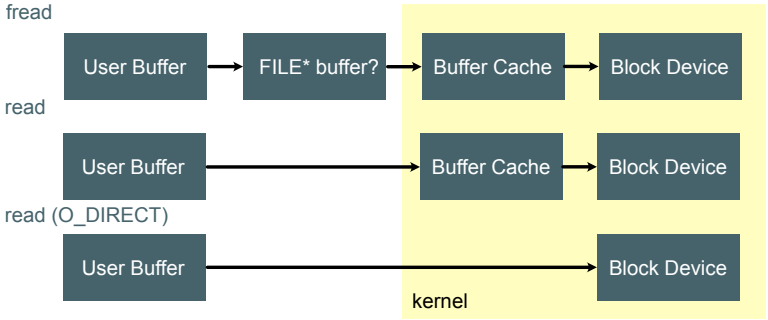
AMD HPC Guide: https://www.amd.com/en/server-docs/high-performance-computing-hpc-tuning-guide-for-amd-epyc-7003-series-processors

Thanks to HighPoint support

FILE DATA CACHES

File data is cached (multiple places).

Cached data is very faster to access.



“Arthor > Preload”

Arthor v3.5: **~3GB/s** (96 thread)

Arthor v3.6: **~14GB/s** (12/96 thread)

Insight - Active I/O on a one NUMA node at a time

“Arthor > Migrate”

NUMA page migration (in Arthor v3.7)

Insight - Normally data is put on a free node rather than where we would like it to go. See also:
vm.zone_reclaim_mode but default is sensible

COUNTING LINES

Word count (wc) is a standard a useful unix utility for counting characters, words, and lines.

Count number of times **\n** (0x10) appears in a stream of bytes.

In a SMILES file it can tell us how many molecules/reactions there are.

```
$ wc -l /data/zinc20/zinc20230806.smi
1922921227 /data/zinc20/zinc20230806.smi
```

OUT-OF-THE-BOX

out-of-cache:

36.49 seconds @ 3.129 GB/s

in-cache:

21.63 seconds @ 5.279 GB/s

```
$> vmtouch -e zinc20230806.smi
$> /usr/bin/time taskset -c 6-11 wc -l zinc20230806.smi
1922925826 zinc20230806.smi
15.06user 21.26system 0:36.49elapsed 99%CPU (... 1928maxresident)k
223018680inputs+0outputs (0major+144minor)pagefaults 0swaps

$> /usr/bin/time taskset -c 6-11 wc -l zinc20230806.smi
1922925826 zinc20230806.smi
14.07user 7.55system 0:21.63elapsed 99%CPU (... 1928maxresident)k
0inputs+0outputs (0major+145minor)pagefaults 0swaps
```

Ubuntu 20.04.6 LTS, 5.15.0-46-generic, Ext4, Raid0, Filesize: 114185563803 bytes

REBUILD FROM SOURCE

Recent versions of **wc -l** can use AVX2 support

GNU coreutils source: git://git.sv.gnu.org/coreutils.git

Roger's 2018 RDKit UGM Talk:

Deceptively Simple: How some cheminformatics problems can be more complicated than they appear (PDF)

out-of-cache:

34.86 seconds @ 3.275 GB/s

in-cache:

8.50 seconds @ 13.433 GB/s

Ubuntu 20.04.6 LTS, 5.15.0-46-generic, Ext4, Raid0, Filesize: 114185563803 bytes

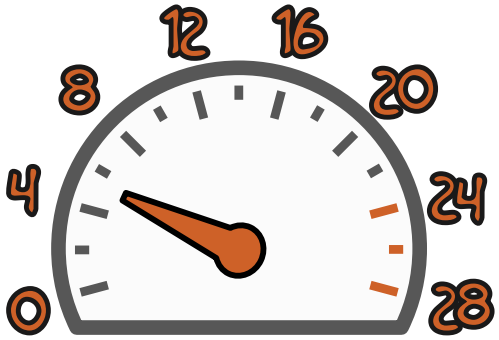
IO_URING MAGIC

```
struct io_uring ring;

if (io_uring_queue_init(opt_iodepth, &ring, 0))
  return fail("io_uring_queue_init()");

// submitting a request (submission queue)
struct io_uring_sqe *sqe = io_uring_get_sqe(&ring);
if (!sqe) break; // sq is full!
io_uring_prep_read(sqe, fd, buffers[i],
                   BUFFER_SIZE, offset);
io_uring_sqe_set_data(sqe, buffers[I]);
if (!io_uring_submit(&ring))
    return fail("io_uring_submit");

// reaping a request (completion queue)
struct io_uring_cqe *cqe;
int ret = io_uring_wait_cqe(&ring, &cqe);
if (ret < 0)
  return fail("io_uring_wait_cqe");
if (cqe->res < 0)
  return fail("IO op failed!");
char *buf = (char*)io_uring_cqe_get_data(cqe);

// process(buf, cqe->res);

io_uring_queue_exit(&ring);
```

With **aio** and **io_uring** it is possible to perform very efficient asynchronous IO

- No copies (O_DIRECT)
  - *cache management overhead*
- Less system calls (io_uring)

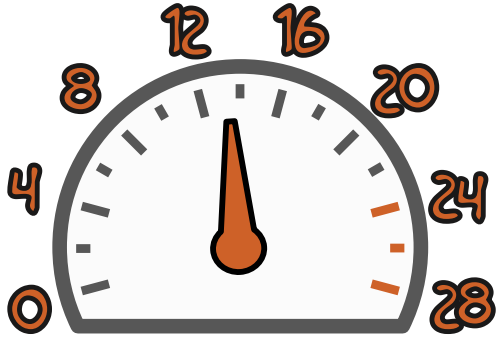
IO_URING MAGIC



wc (default): **5.27GB/s**

21.63 seconds

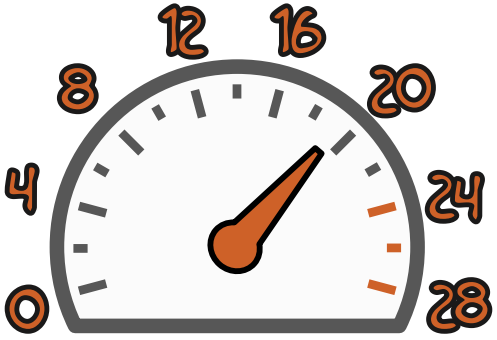
(cached in RAM)



wc (latest): **13.43GB/s**

8.5 seconds

(cached in RAM)



NextMove: **19.32GB/s**

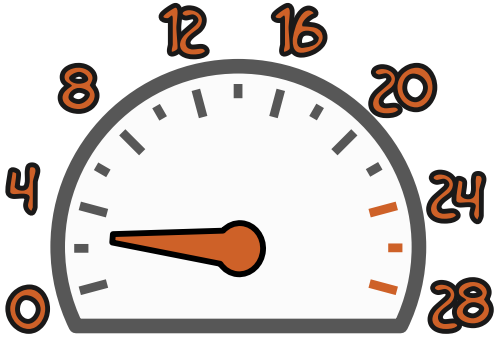
***5.91** seconds*

(read from disk)

Counting lines (Zinc20) faster from disk than in RAM?

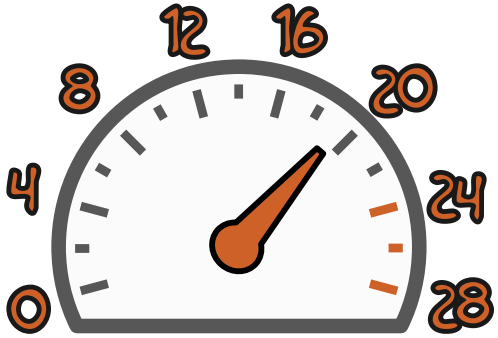
- due to fread() system call overhead, copying

ZINC22



wc (default): **2.75 GB/s**

958.44 seconds



NextMove: **19.37GB/s**

136.21 seconds

NextMove -j2: **21.65GB/s**

121.21 seconds

2.4TB for **35 billion** Zinc22 SMILES from Zinc22, does not fit in memory

CHALLENGES

- EPYC Gen3 DDR4 Maximum possible speed is ~20GB/s
  - Seems to be ~40GB/s limit (read+write: 20GB/s), 42GB/s GMI?
  - Buffers in L3 cache (16MB) did not help
- EPYC Gen4 used 3 channels of DDR5 per node, bottle neck removed?

Programming paradigms:

- Ordering of I/O
- Splitting on line breaks (line-by-line processing) non trivial
- Linux only (equivalent but different API for OS X/Windows)

SUMMARY

SmallWorld

- One trillion (now)
- Faster node lookups (now)
- Improved/simpler Web API (soon)

Arthor

- Faster preload (now) / migrate (imminent)
- Inverted similarity searching (3-5x faster, soon)
- Substructure compression (soon)

Efficient I/O

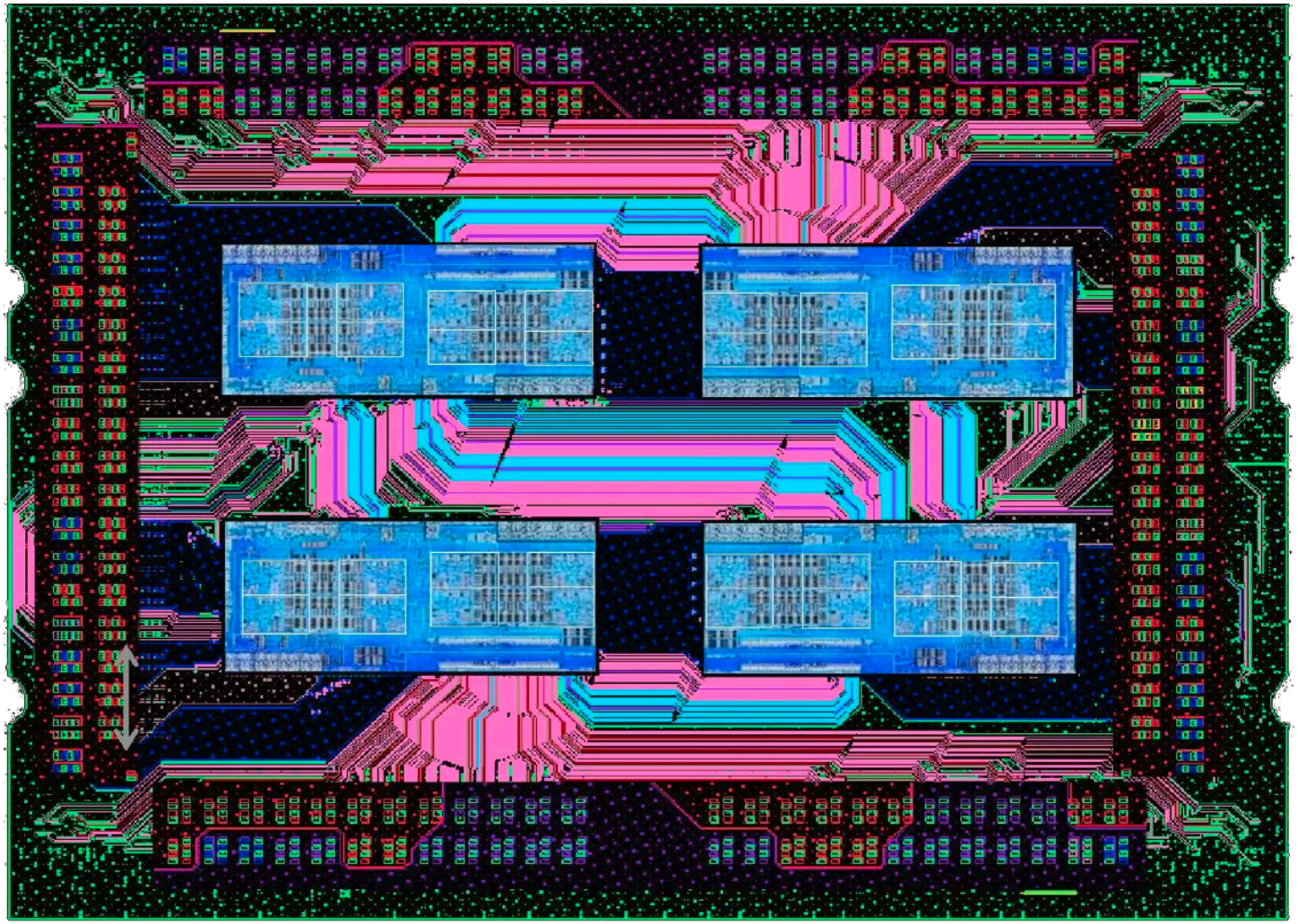
- Async I/O integration SmallWorld and Arthor

ACKNOWLEDGEMENTS

- Ingvar Lagerstedt
- Rachael Pirie
- Richard Gowers
- Noel O'Boyle
- John Irwin
- Yurii Moroz

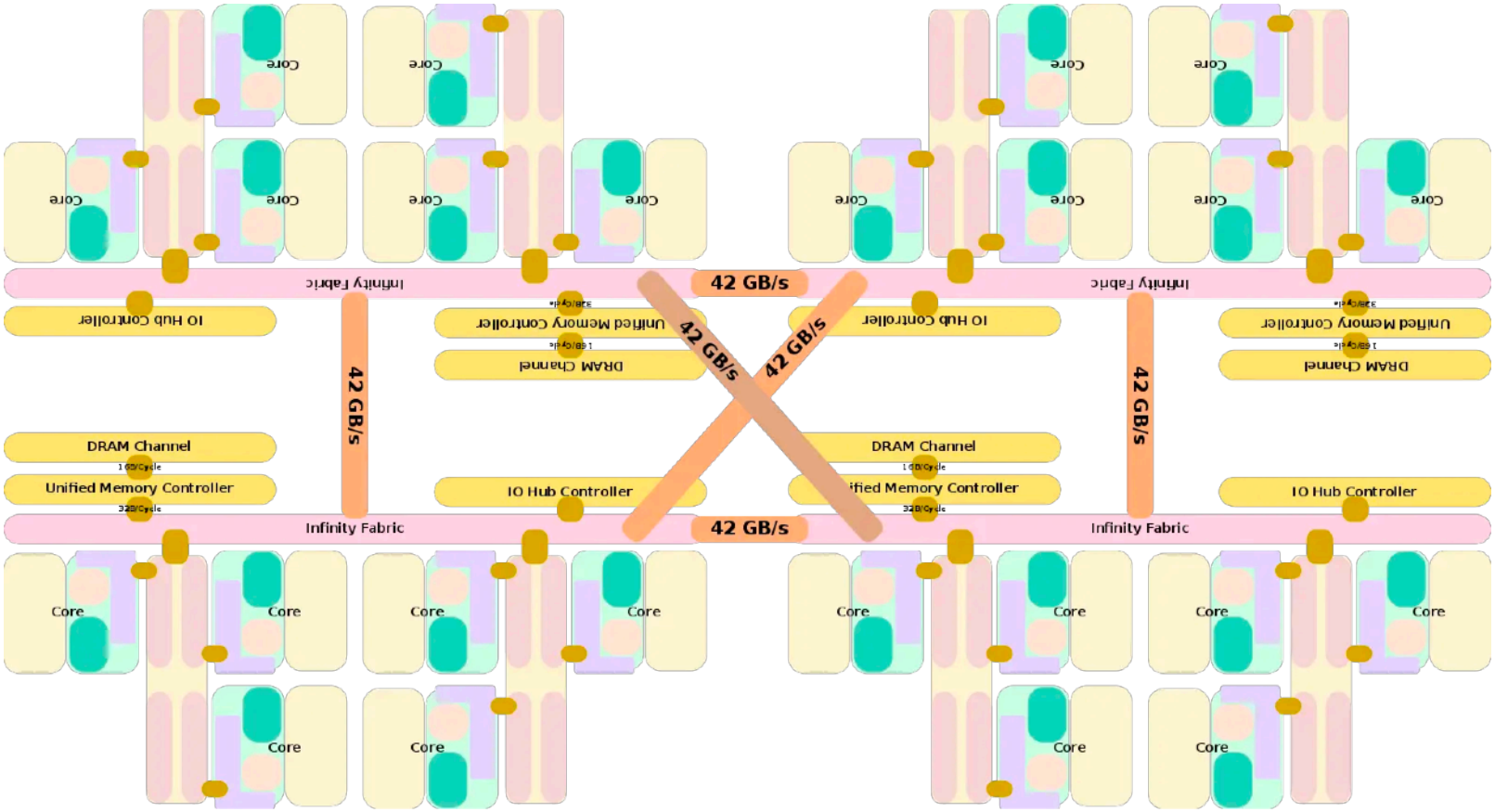
NextMove Software (Aug 2023)

REALITY



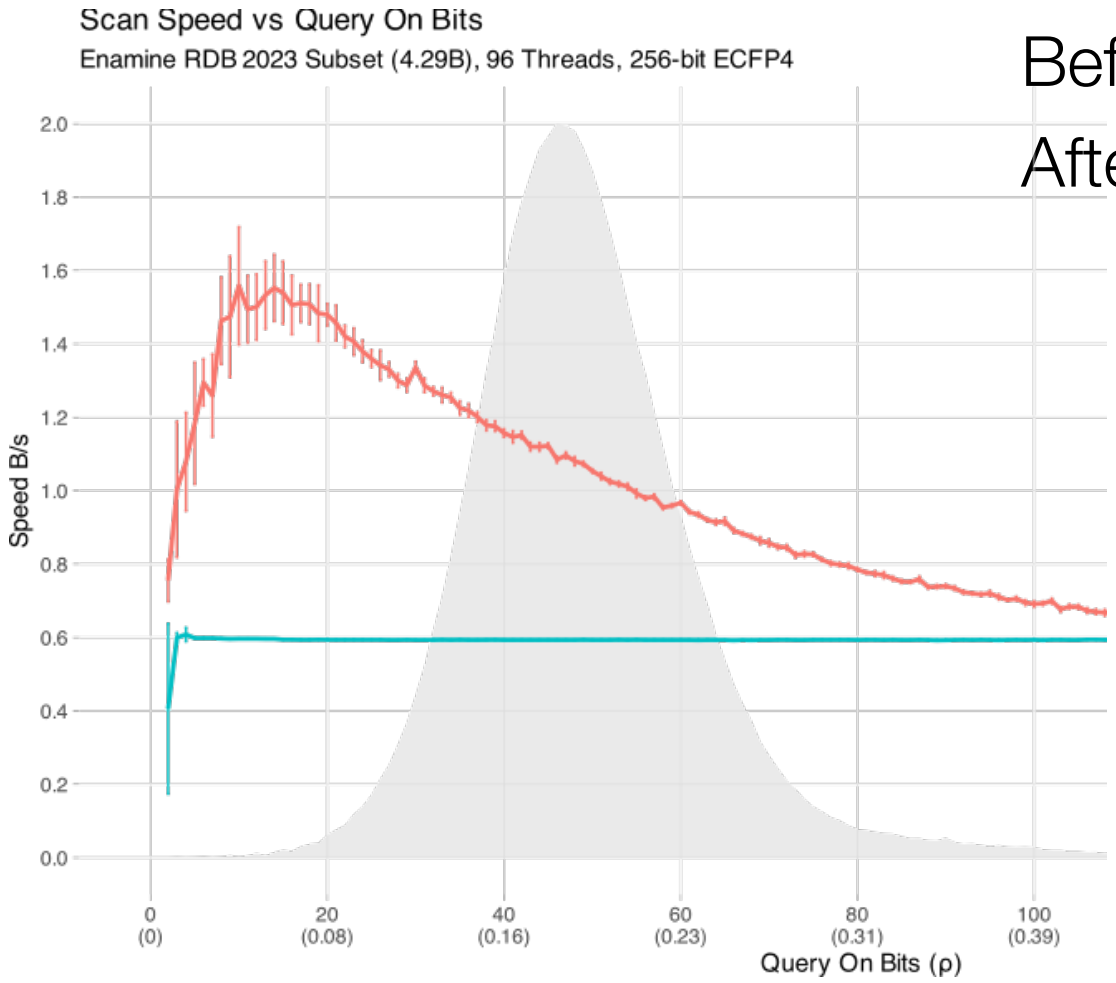
https://en.wikichip.org/wiki/File:amd_epyc_interconnect.png

GMI SPEED?



https://en.wikichip.org/wiki/File:zen_soc_block_(24_cores).svg

SINGLE THREAD REALITY

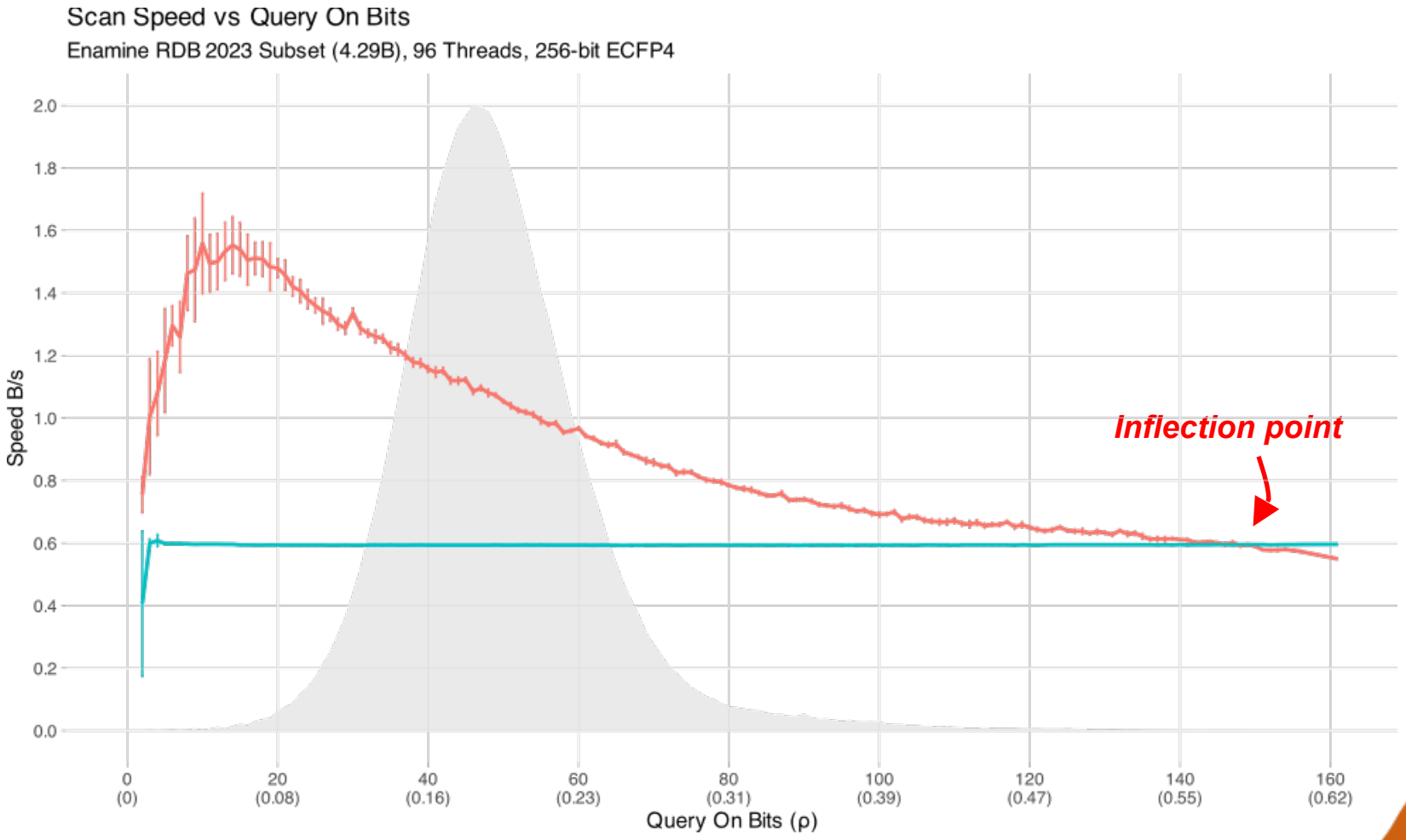


Before: ~0.6 B/s

After: ~0.9-1.2 B/s

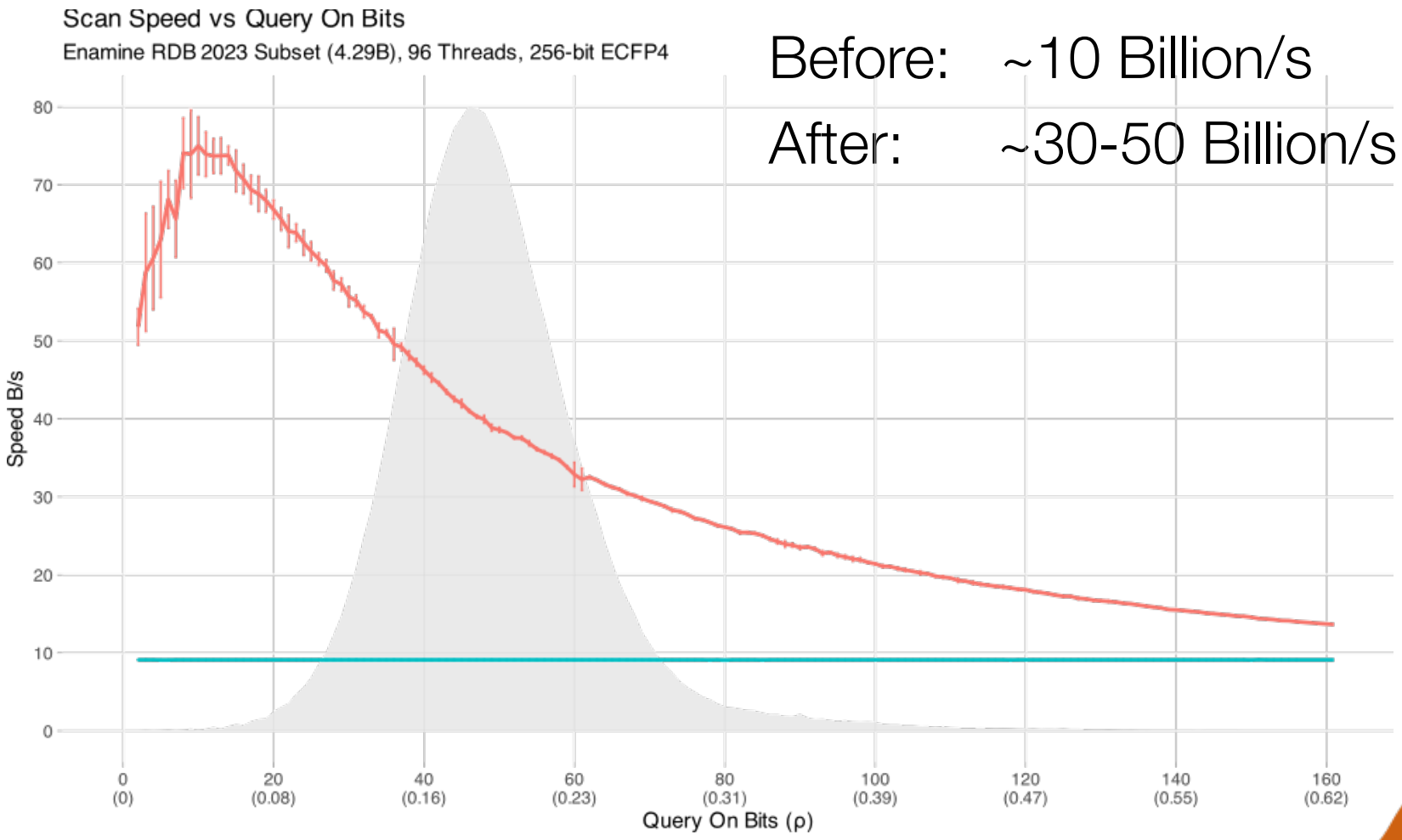
ChEMBL DB POPC distribution shown in grey

SINGLE THREAD REALITY



ChEMBL DB POPC distribution shown in grey

96 THREAD REALITY



Before: ~10 Billion/s

After: ~30-50 Billion/s