The 5th Workshop on

ACCELERATOR ARCHITECTURE IN COMPUTATIONAL BIOLOGY AND BIOINFORMATICS

June 18th, 2023

In conjunction with 50th IEEE International Symposium on Computer Architecture

https://aacbb-workshop.github.io/

Orlando, Florida, USA

Exploring genomic algorithms on Processing-in-Memory Architecture

Dominique Lavenier

GenScale group, University of Rennes, IRISA-CNRS, Inria

Université de Rennes

UMR IRISA

Inria

Agenda

- Processing-in-Memory (PiM)
  - Why move the calculation close to the data?
  - PiM taxonomy
- Early projects (in Rennes)
  - RDISK (2002-2004)
  - ReMIX (2005-2007)
- UPMEM memory
  - Architecture
  - 2 genomic implementation examples
    - Mapping
    - Protein data base search
- Conclusion

Processing-in-Memory

Main idea

moving computation near the data

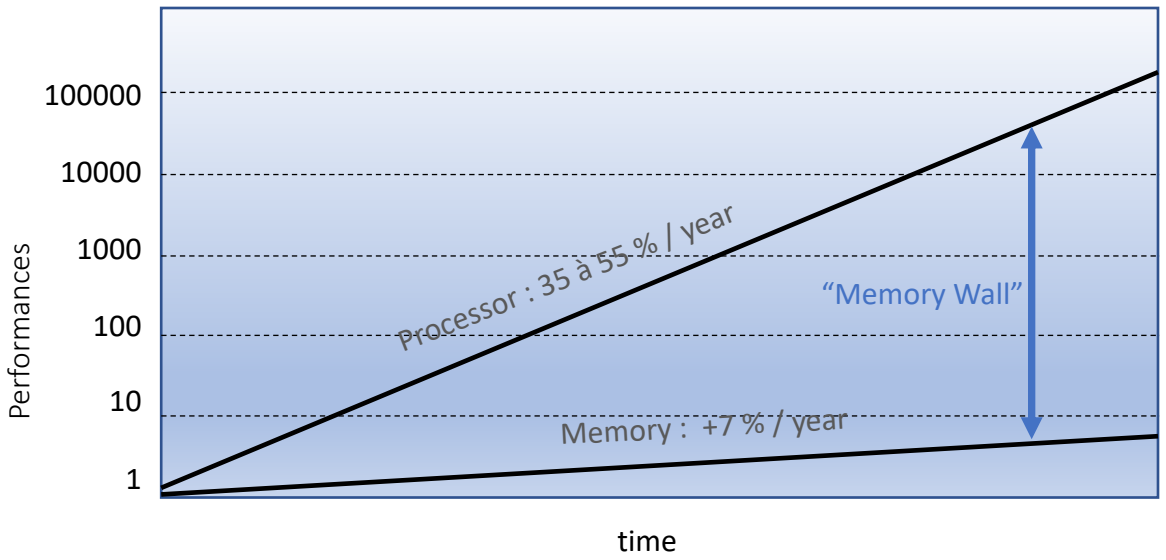
Why ?

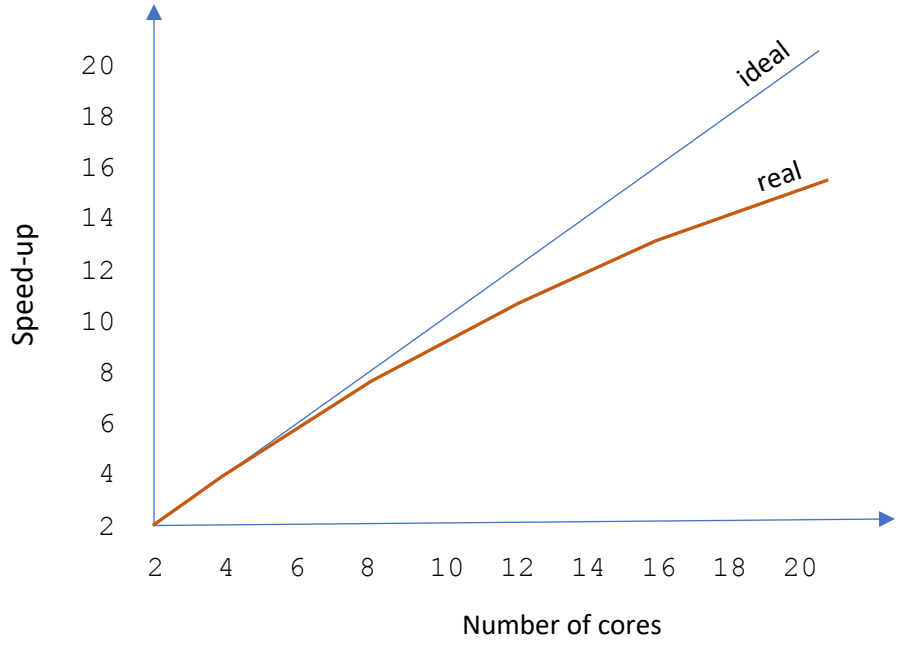
1. To counter the "memory wall" effect
2. To save energy
3. To process efficiently “big-data” applications

Memory Wall

1. Memory wall
2. Energy
3. Big data

Processor and memory technology do not evolve at the same rate





Energy consumption

1. Memory wall
2. Energy
3. Big data

Raw energy cost for various operation (technology – 45nm, 0.9V)

Integer	
Add	
8 bit	0.03 pJ
32 bit	0.1 pJ
Mul	
8 bit	0.2 pJ
32 bit	3.1 pJ

Floating point	
FAdd	
8 bit	0.4 pJ
32 bit	0.9 pJ
FMul	
8 bit	1.1 pJ
32 bit	3.7 pJ

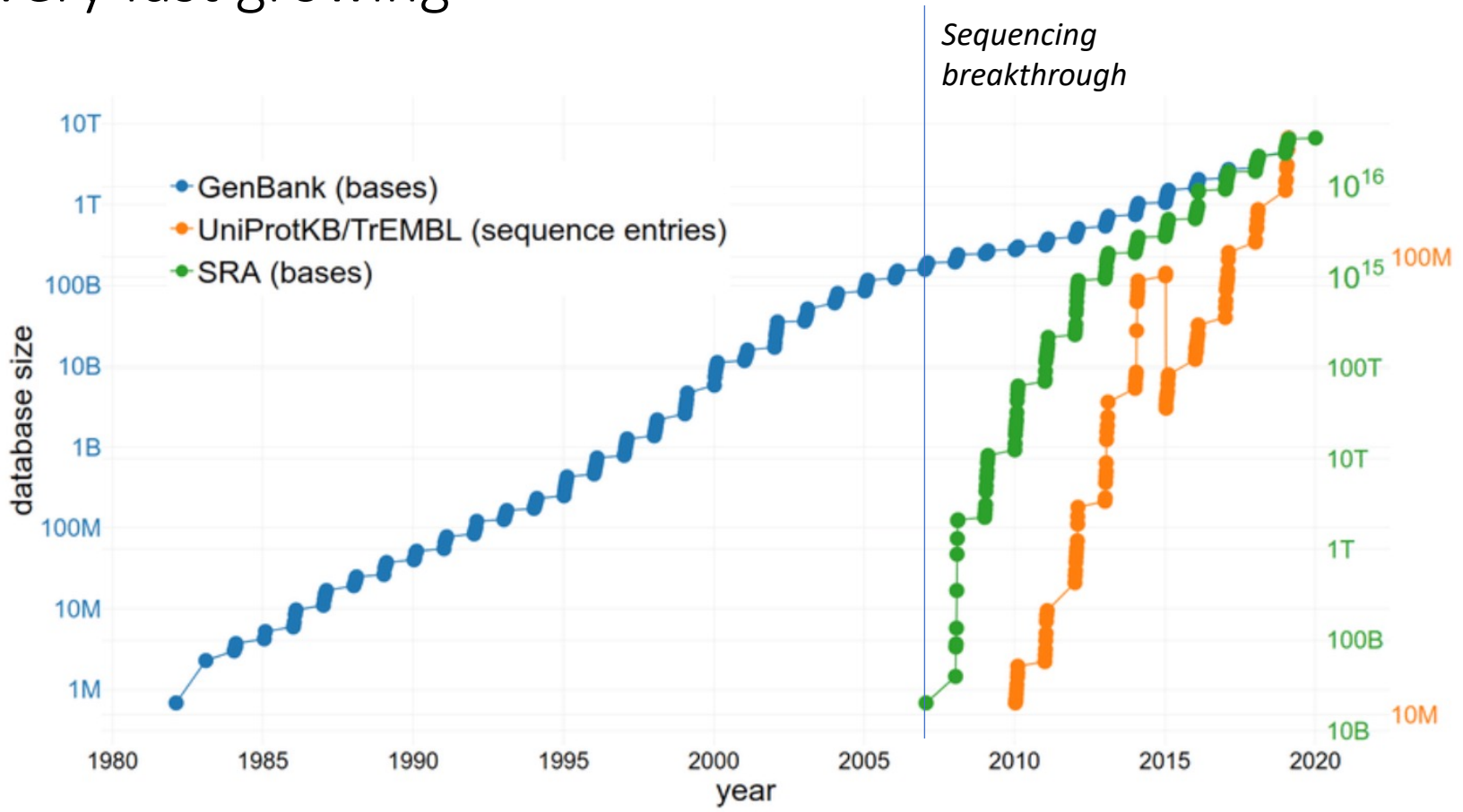
Memory	
Cache (64 bit)	
8 KB	10 pJ
32 KB	20 pJ
1 MB	100 pJ
DRAM	1300 -2600 pJ

From M. Horowitz, ISSCC 2014 (IEEE International Conference on Solid-State Circuit), Computing's Energy Problem (and what we can do about it)

Genomics data

1. Memory wall
2. Energy
3. Big data

Very fast growing



GenBank: 1.2 Tbases (April 2022)

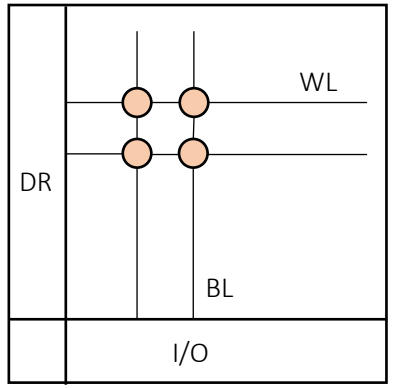
SRA: 100 Pbases

PiM Taxonomy

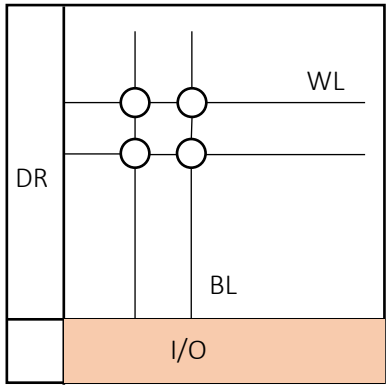
IN computing memory

Data stay in the cell array

memory array



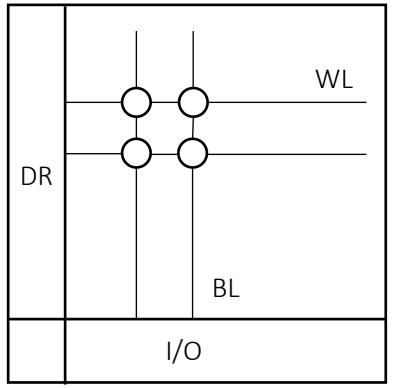
Computing is done inside the memory cells



Computing is done inside the peripheral circuits

NEAR computing memory

Data move outside the cell array

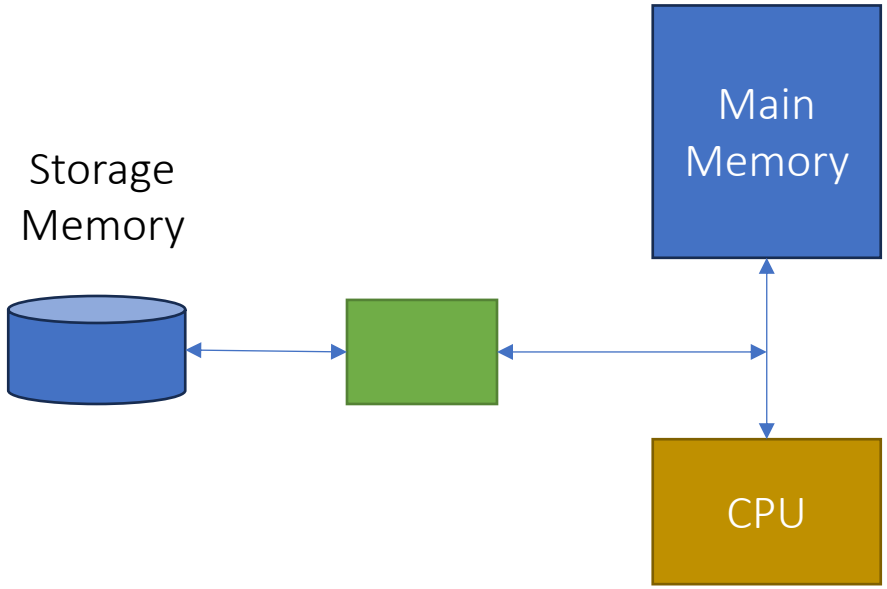


Computing resources

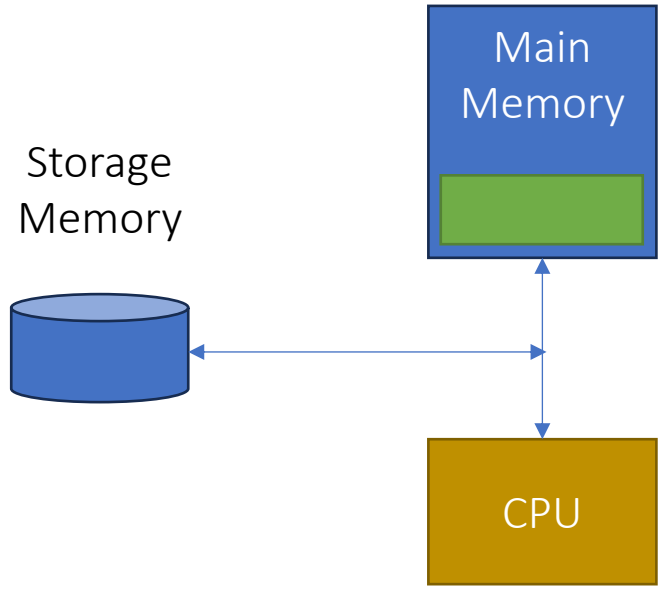
Computing is done outside the memory array

Near Computing Memory Architecture

Where to place extra computing resources ?



RDISK / ReMIX architecture



UPMEM architecture

Agenda

- Processing-in-Memory (PiM)
  - Why move the calculation close to the data?
  - PiM taxonomy
- Early projects (in Rennes)
  - RDISK (2002-2004)
  - ReMIX (2005-2007)
- UPMEM memory
  - Architecture
  - 2 genomic implementation examples
    - Mapping
    - Protein data base search
- Conclusion

RDISK Project

Main focus: data base querying

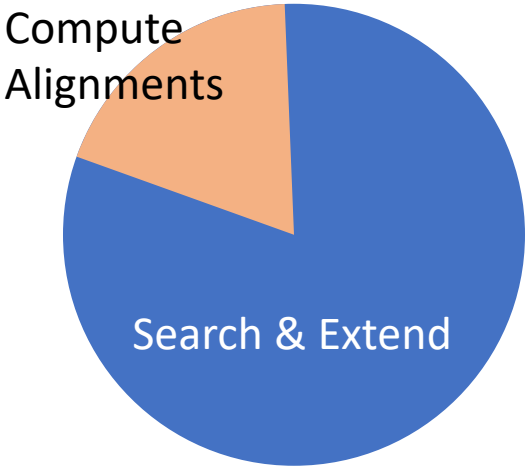
- How to avoid sending large databases into the CPU main memory ?
- Size database >> CPU memory size

Example : blast search

Seed-extend strategy

1. Search hits
2. Extend hits
3. Compute alignments

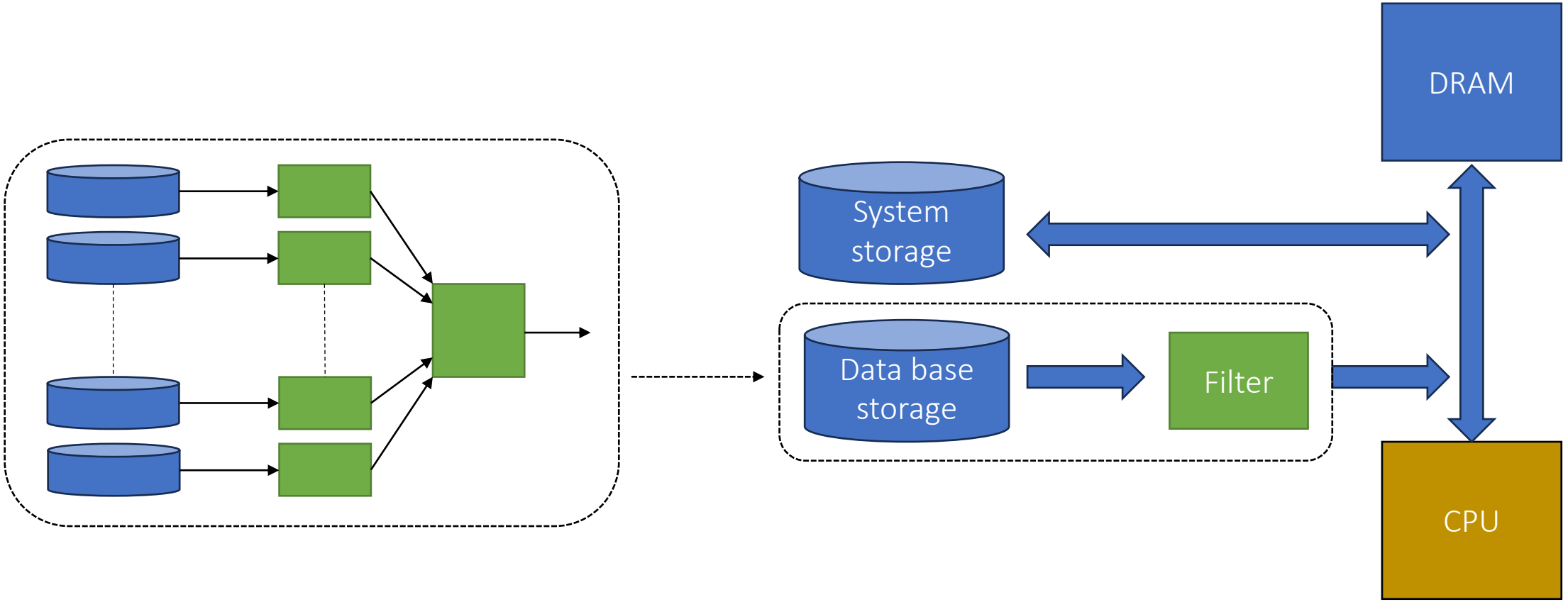
70-80% - Memory bound



Send to CPU only DB sequences that contain potential alignments with the query
Perform steps 1 & 2 outside the CPU

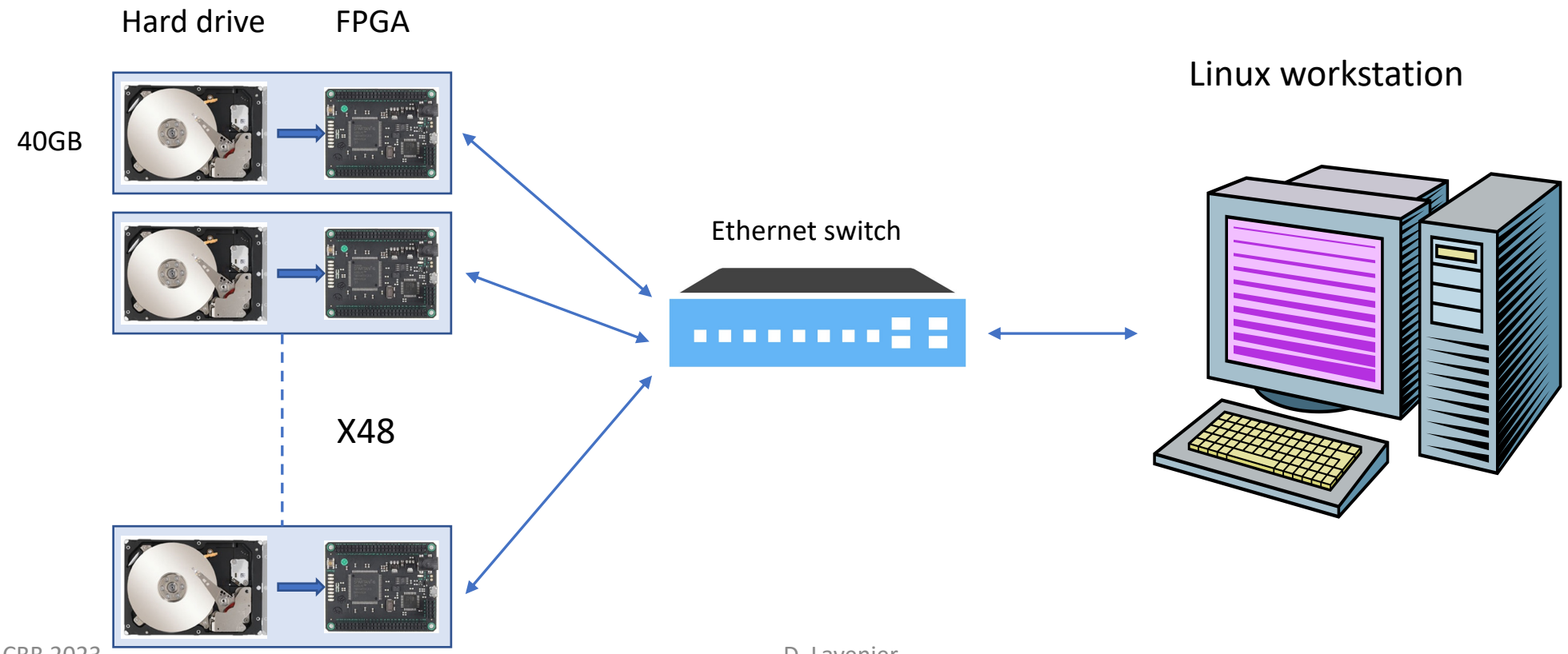
RDISK Architecture

Hardware filter directly connected to database storage devices

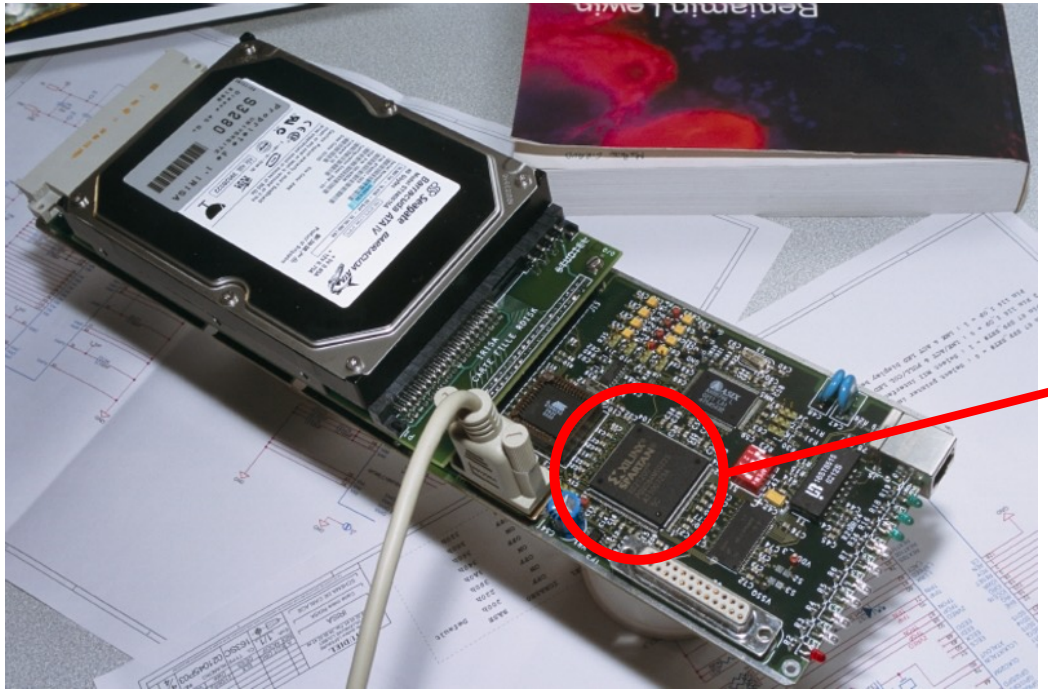


RDISK Implementation

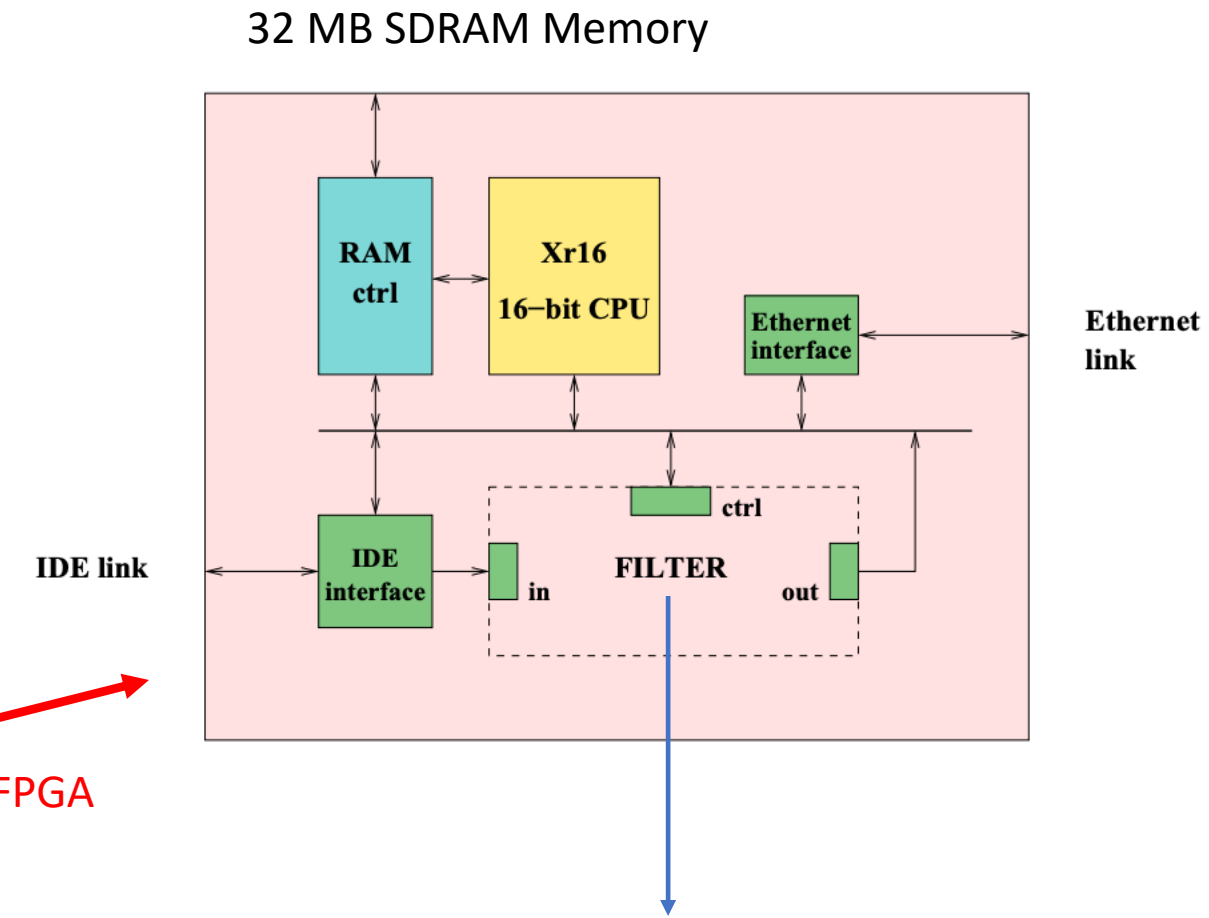
Design in early 2000s



FPGA: SoC architecture



FPGA

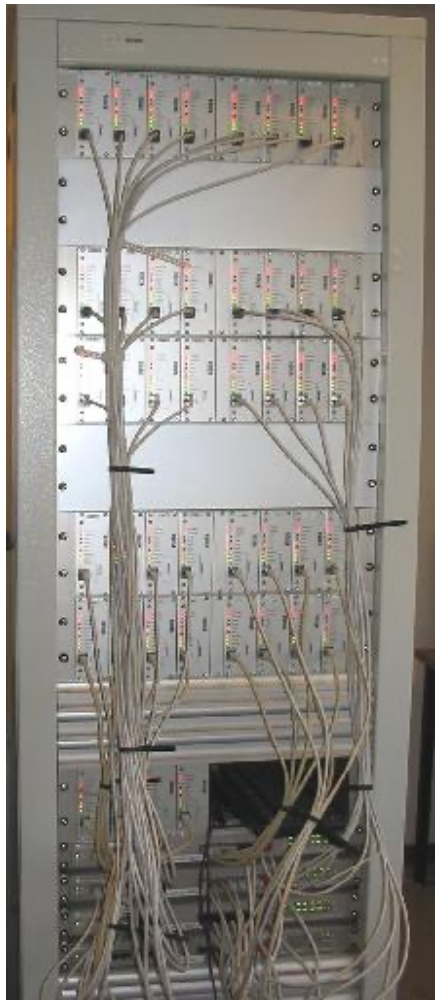


RDISK System

48 nodes system
~2 TBytes

Applications :

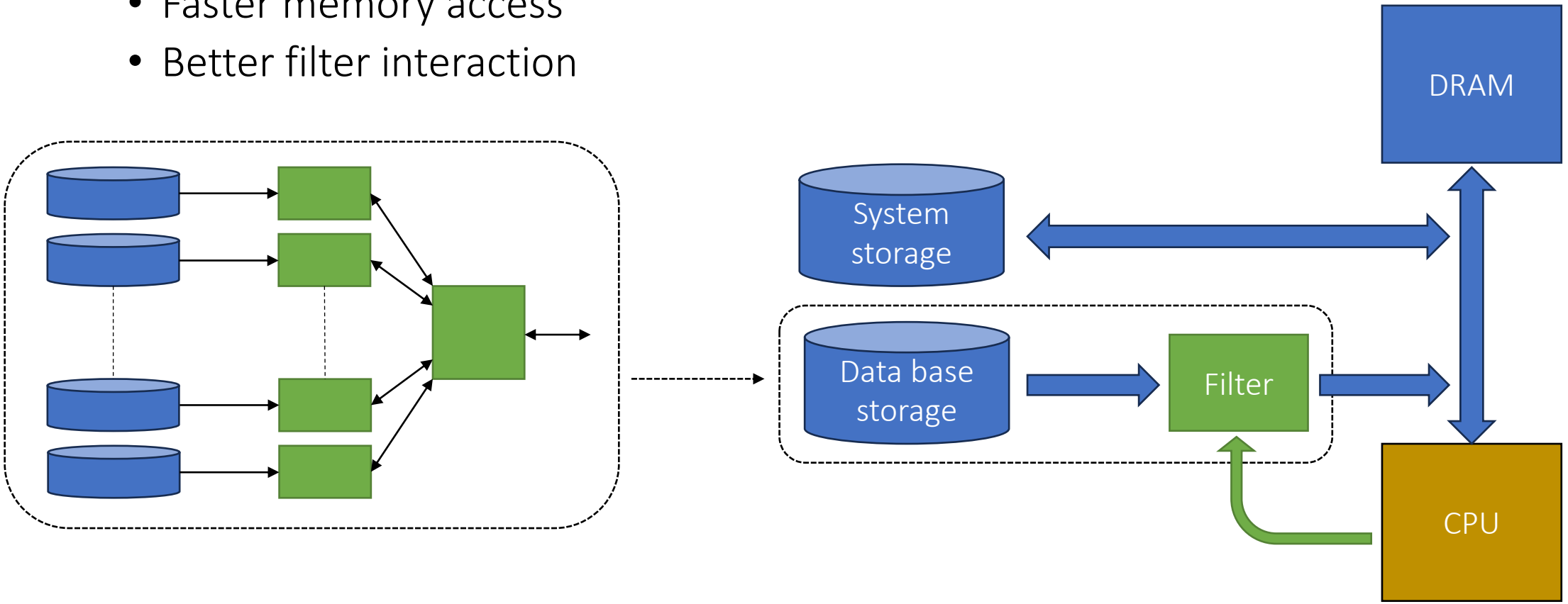
- sequence comparison
- motif search
- data mining
- ...



ReMIX Architecture

Evolution RDISK

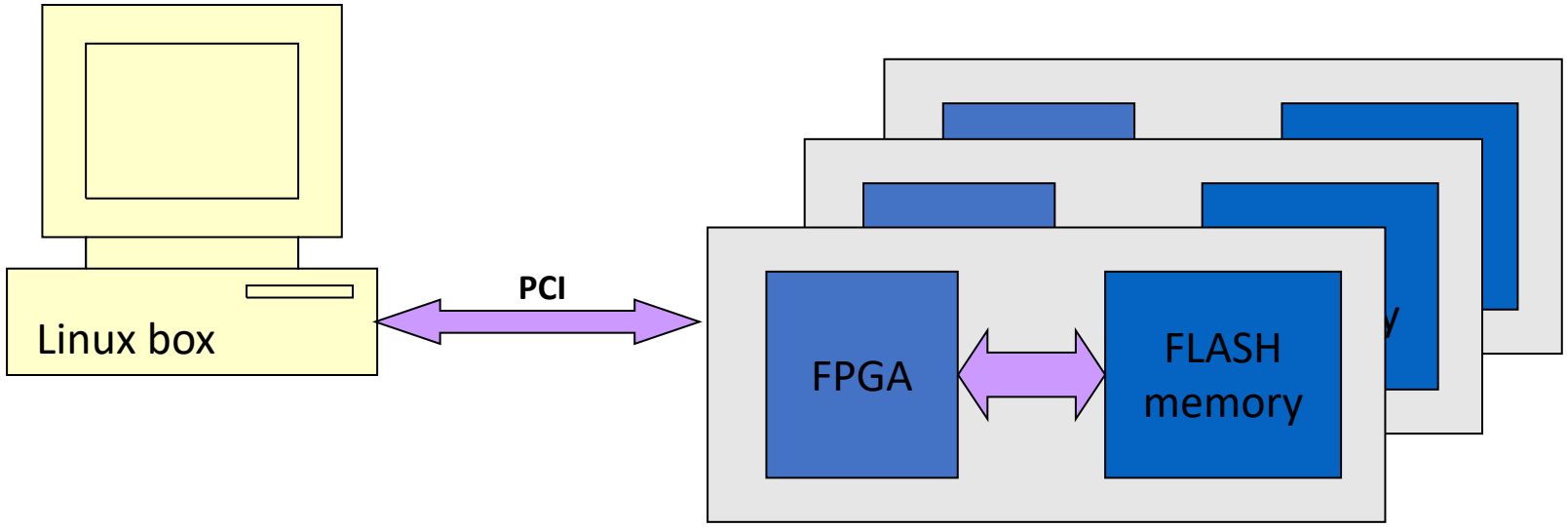
- Faster memory access
- Better filter interaction



ReMIX Implementation

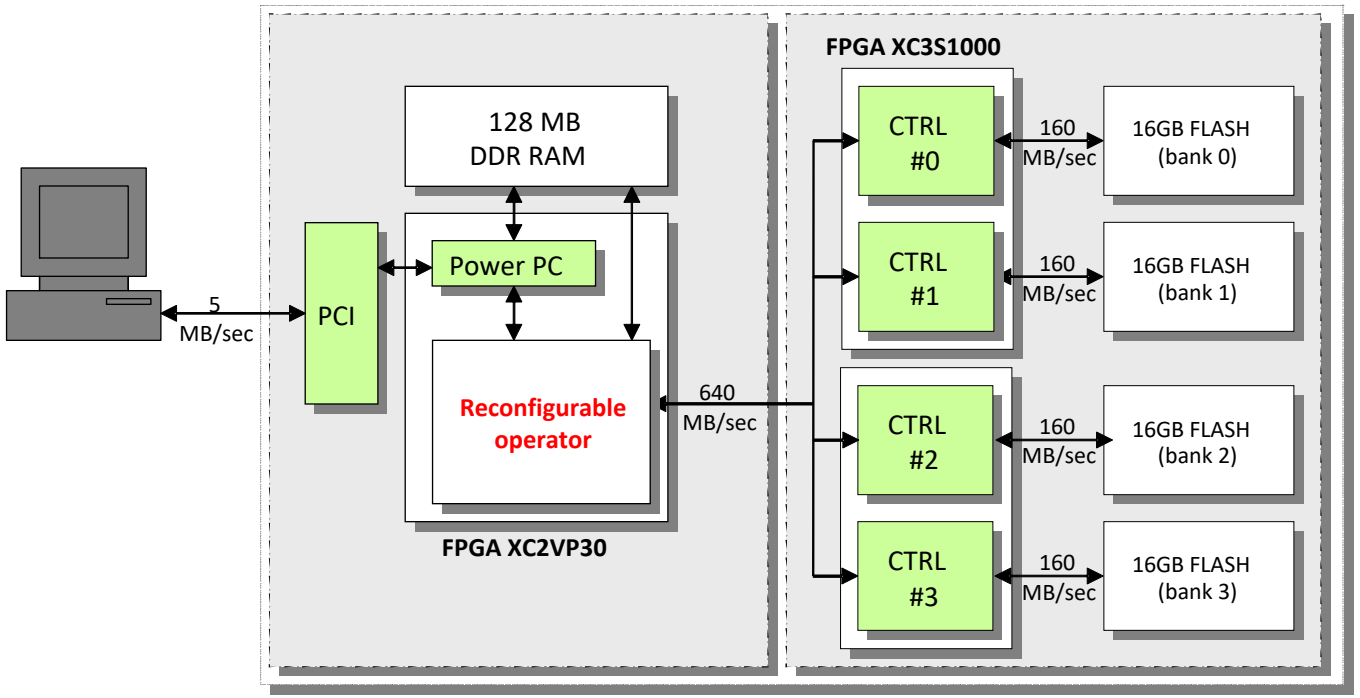
Replace disk by FLASH memory

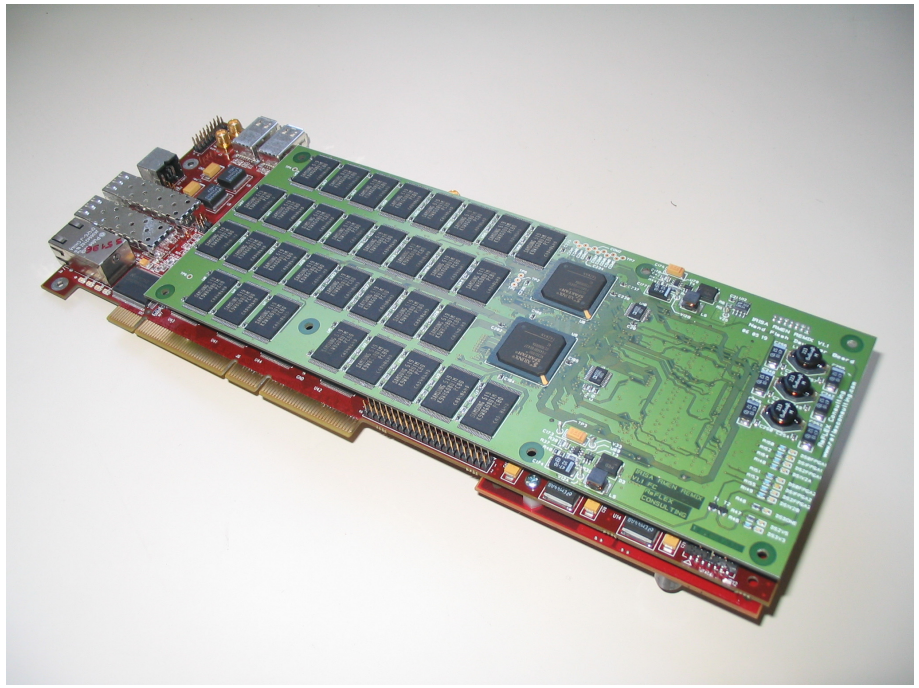
Advanced FLASH memory controller



ReMIX hardware implementation

64 GBytes Flash memory





ReMIX Performances

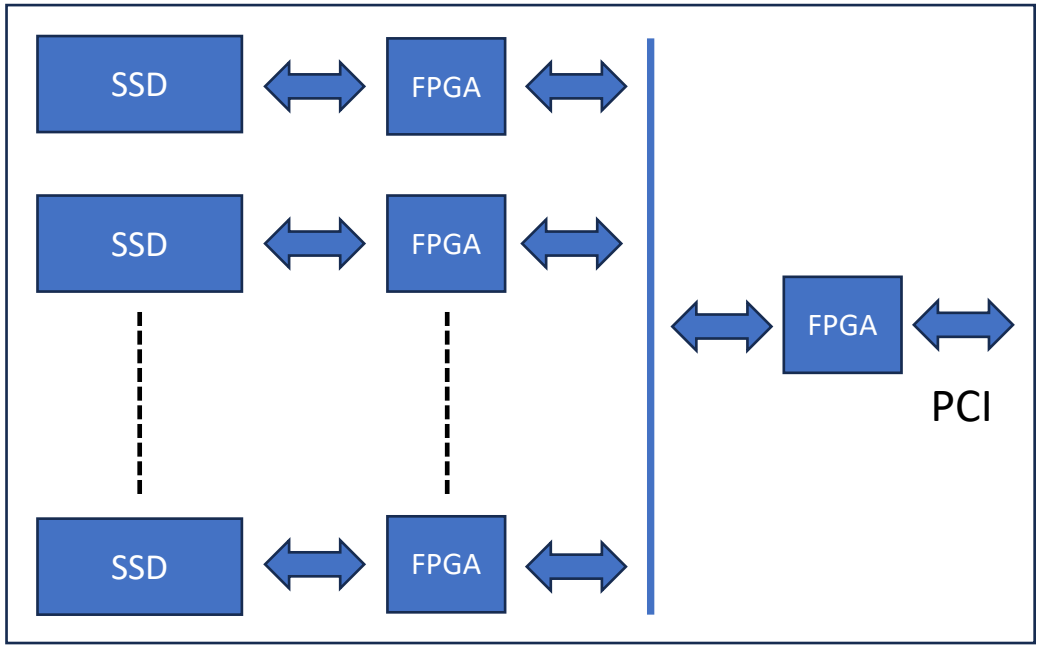
Intensive Protein Sequence Comparison

400 000 proteins vs Human Genome (2006)

- FPGA / FLASH hardware (1 ReMIX node) → 11 days (Blastp-like)
- Bioinfomatics server (64 nodes) → 13 days (Blastp)

Fermat SAS

Industrial version of ReMIX
Solid State Drive + FPGA



Up to 32 TBytes

Road map in genomics

- Base calling
- Sequence alignment
- Variant calling
- Database search

https://fermatinternational.com/

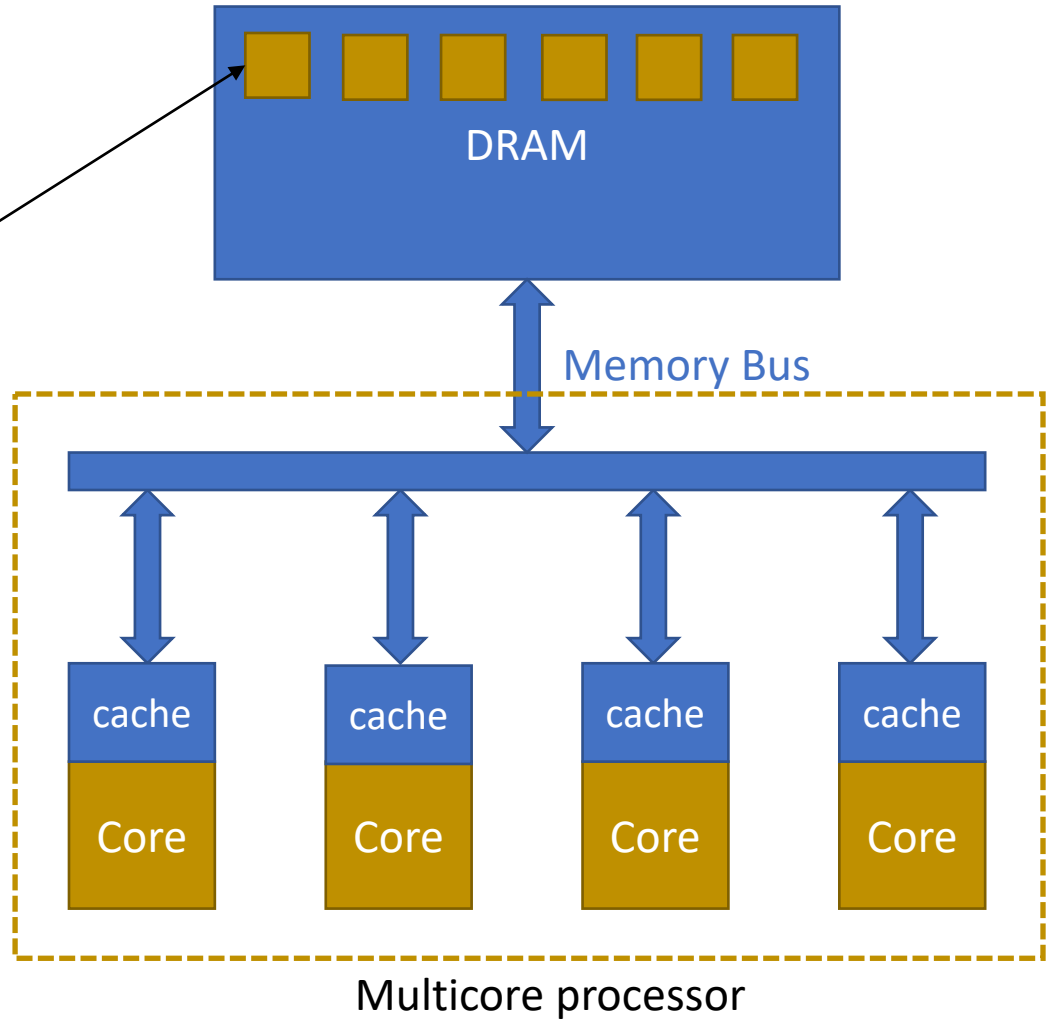
Agenda

- Processing-in-Memory (PiM)
  - Why move the calculation close to the data?
  - PiM taxonomy
- Early projects (in Rennes)
  - RDISK (2002-2004)
  - ReMIX (2005-2007)
- UPMEM memory
  - Architecture
  - 2 genomic implementation examples
    - Mapping
    - Protein data base search
- Conclusion

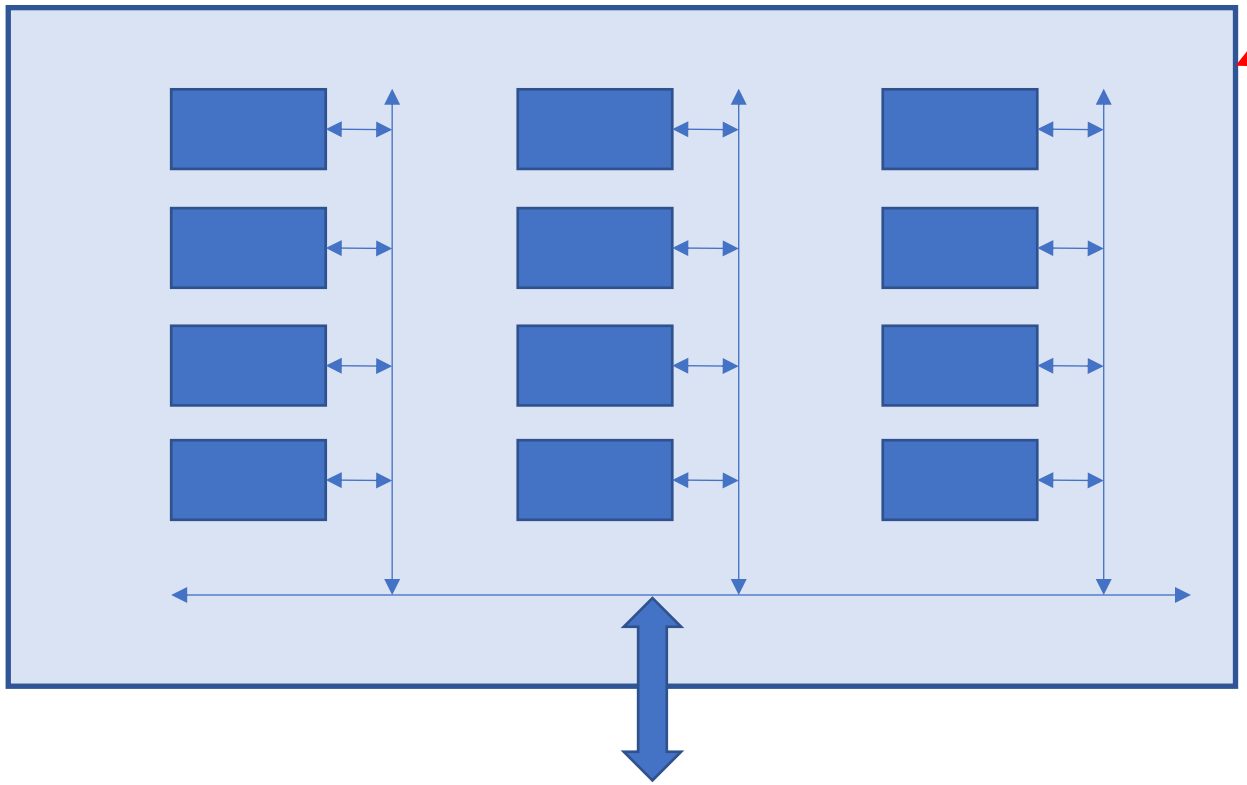
UPMEM Memory

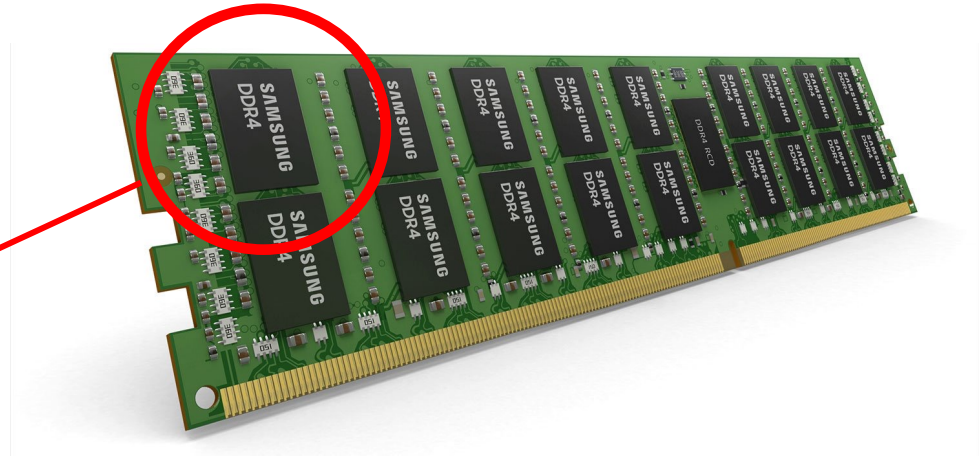
Add computing resources
inside the DRAM memory

DPU : DRAM Processing Unit

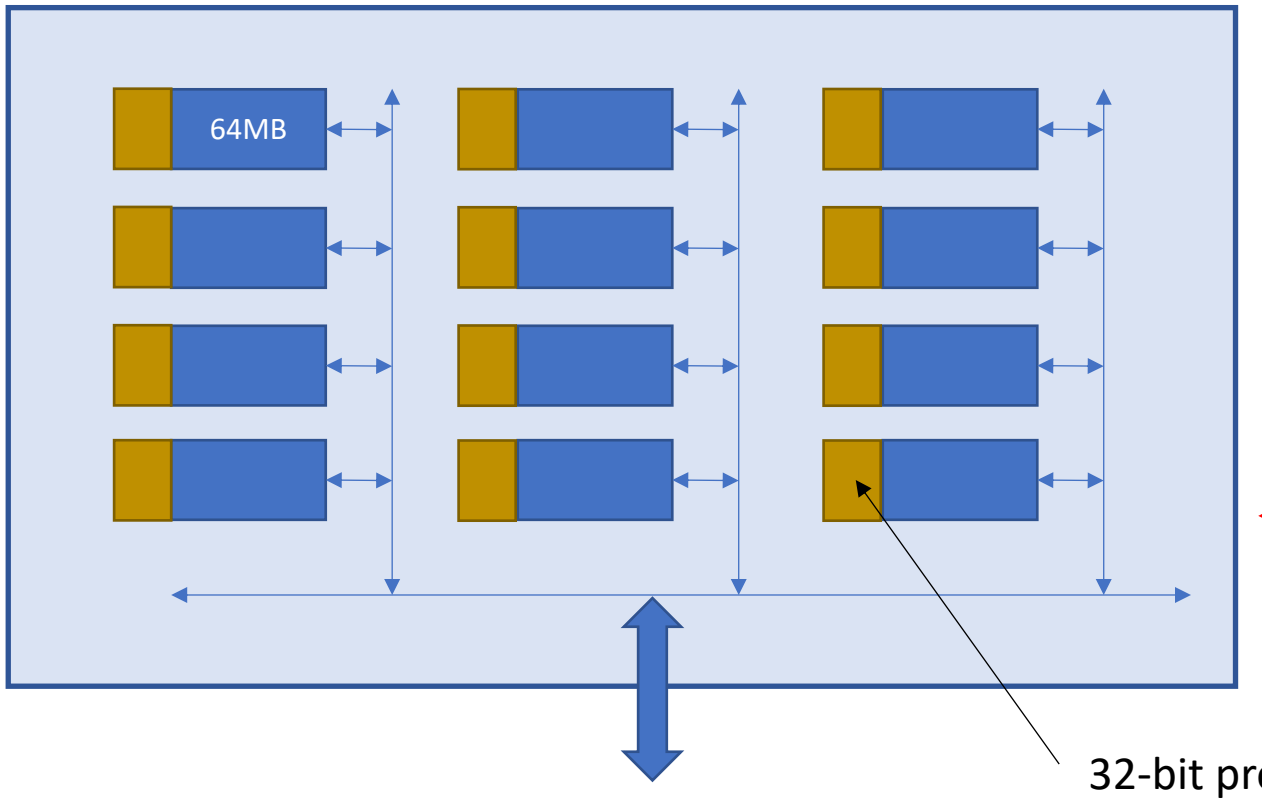


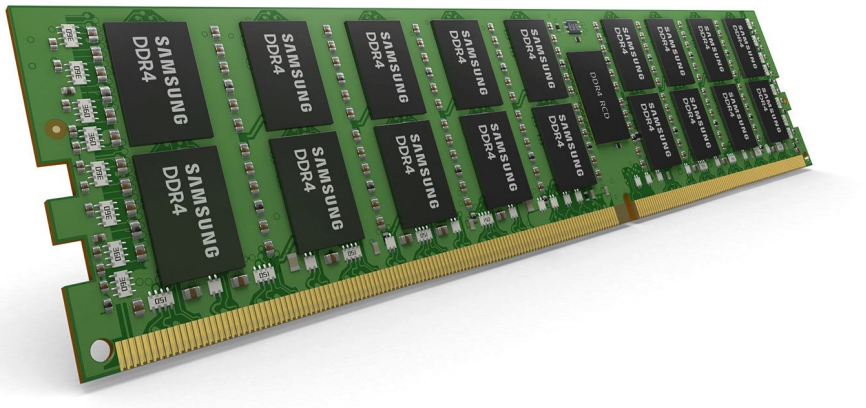
DRAM Architecture





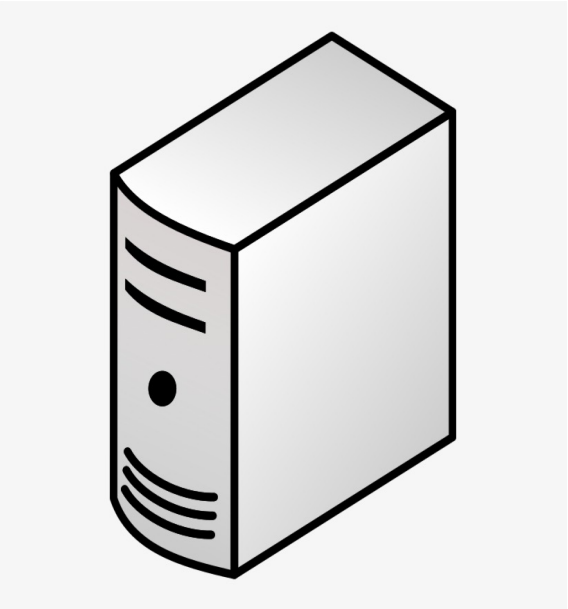
UPMEM DRAM Architecture





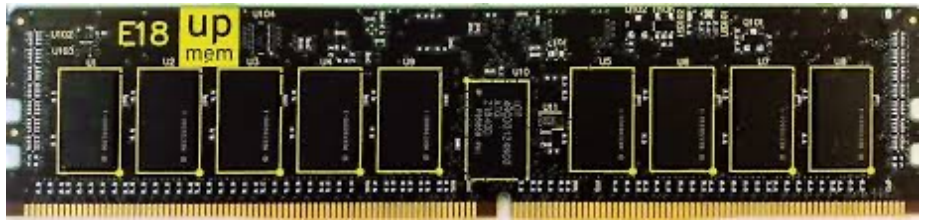
32-bit processor
Multithreaded: up to 24 threads

UPMEM server



16 cores (32 threads)

+



Legacy memory
256 GBytes

PiM memory
160 Gbytes
+
2560 Processors

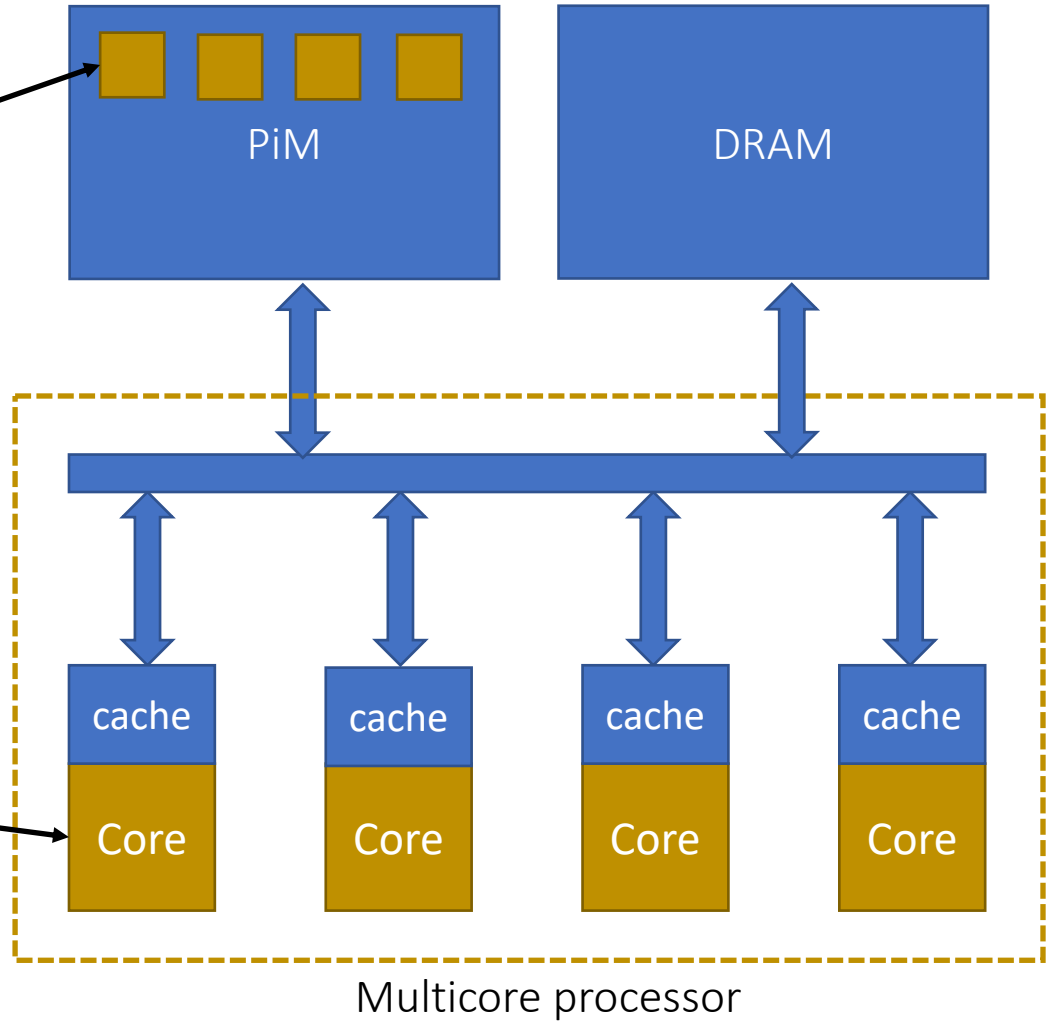
Programming

DPU: 32-bit processor

C Language
1 thread = 1 tasklet

HOST

C, C++, Python
Manage
- Data movements
- DPU synchronization

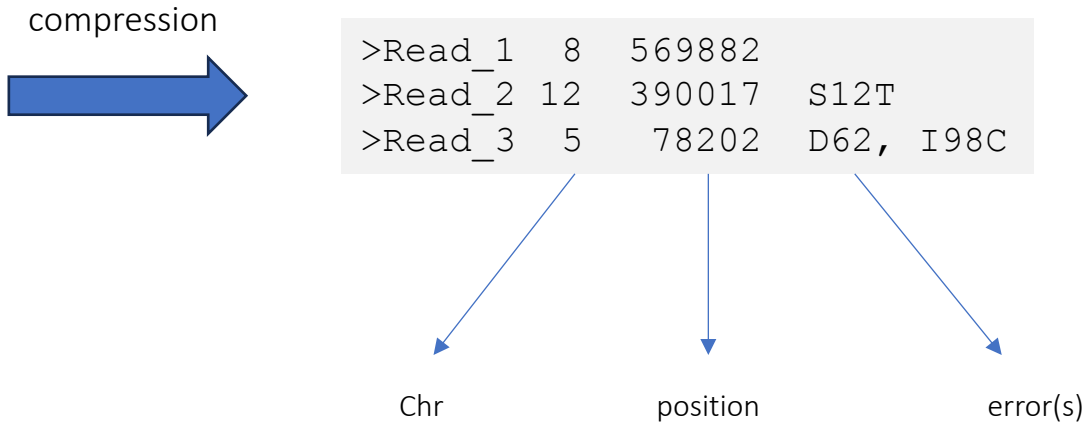


Read mapping for compression

Compression with reference genome (Human Genome)

Input format : Fasta or Fastq

```
>Read_1
GGACACGAGATATTAGGGAGGAGAGATTATTATAGGAGAGGAAA
>Read_2
TTGACACAGAGATAGAGCACCAGGAGATATATGAGAGGACTGGA
>Read_3
CCATAGGACACGATAGGACACACCGGGATATTAGGCCGGAACTT
```



Read mapping for compression

BWA mapping:

Reads
- SRR14724533
- Illimina read
- 40X coverage

Reference
- Human Genome
- GRCh38

```
0 substitution   : 71.9732 %
1 substitution   : 13.6434 %
2 substitutions  : 3.6311 %
3 substitutions  : 1.5432 %
4 substitutions  : 0.8963 %
1 indel 0 sub    : 0.629 %
1 indel 1 sub    : 0.2697 %
1 indel 2 sub    : 0.1264 %
1 indel 3 sub    : 0.0804 %

total read mapped    : 92.7927 %
```

91.6512 %

1.1055 %

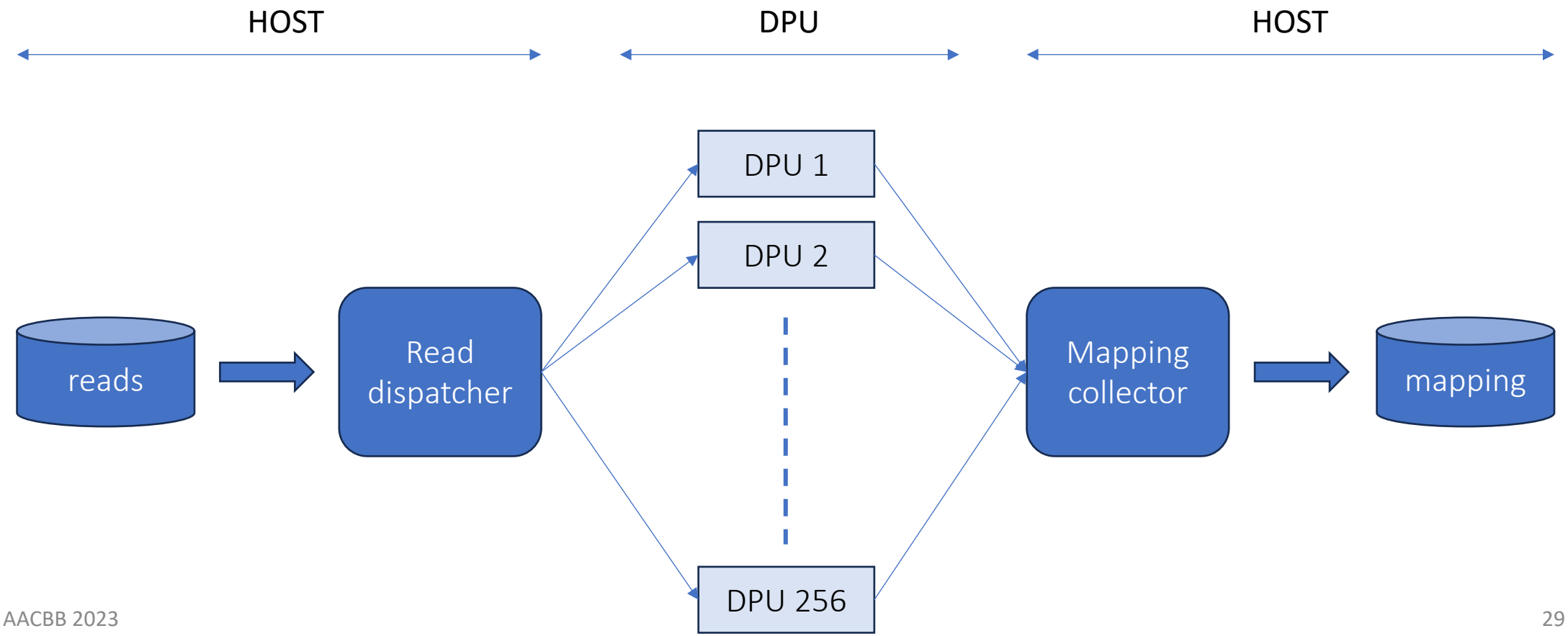
7.2 % not mapped (or partially mapped)

Mapper objectives

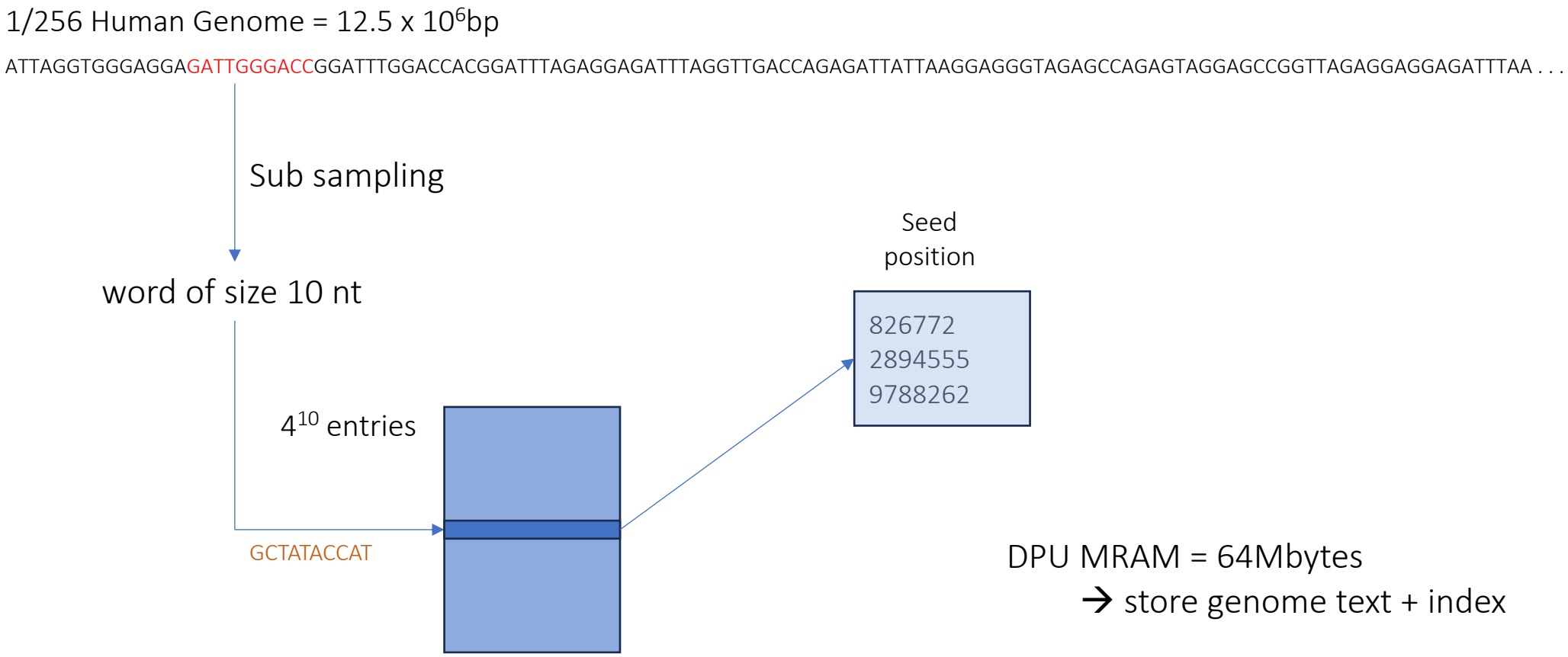
- Keep host processor computing resources to a minimum
- 256 DPUs → Index must fit inside 16GBytes of PiM memory
- Mapping at the disk throughput
- Consider only small gaps
- Reach a minimum mapping percentage of 80 - 85 %
  - Sufficient to obtain a good compression ratio

Implementation principle

Genome index distributed on the DPUs

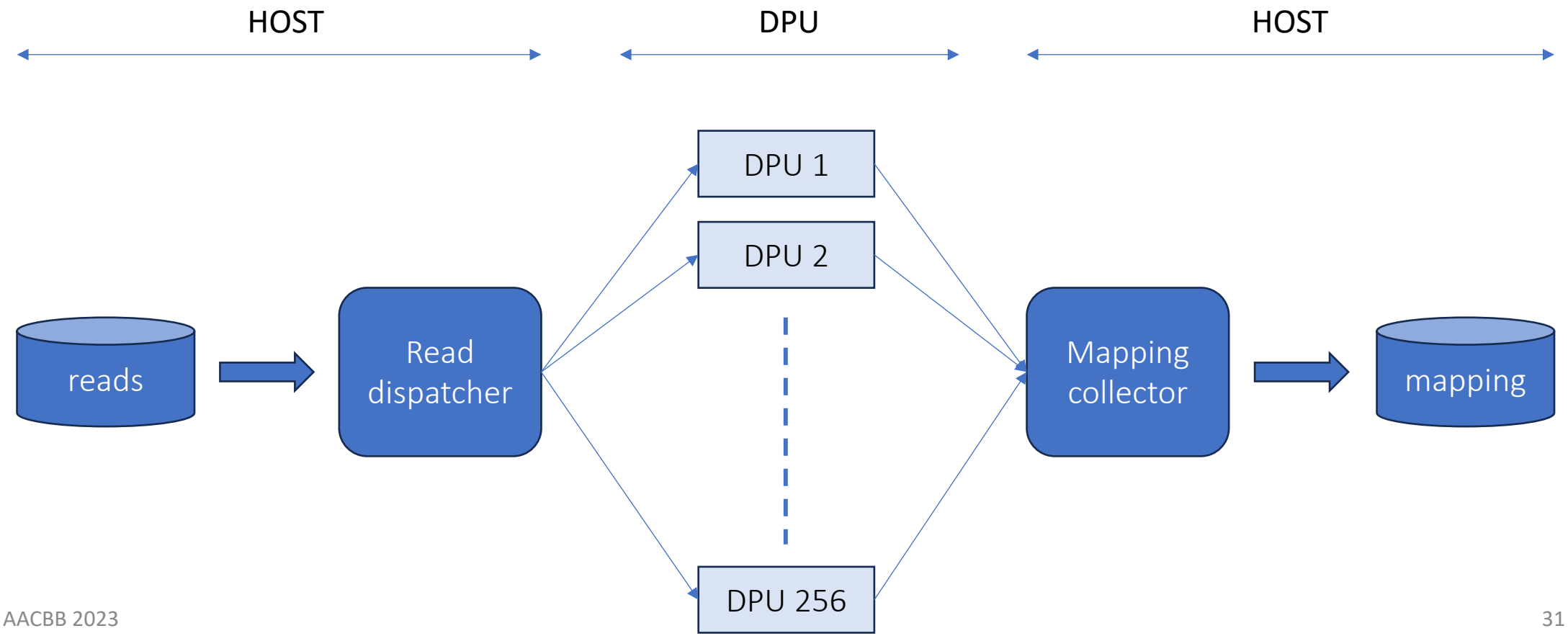


DPU index


1/256 Human Genome = 12.5 x 10^6bp
ATTAGGTGGGAGGAGATTGGGACCGGATTTGGACCACGGATTTAGAGGAGATTTAGGTTGACCAGAGATTATTAAGGAGGGTAGAGCCAGAGTAGGAGCCGGTTAGAGGAGGAGATTTAA . . .
Sub sampling
word of size 10 nt
4^10 entries
GCTATACCAT
Seed position
826772
2894555
9788262
DPU MRAM = 64Mbytes
→ store genome text + index

How to avoid to dispatch reads to all DPUs ?

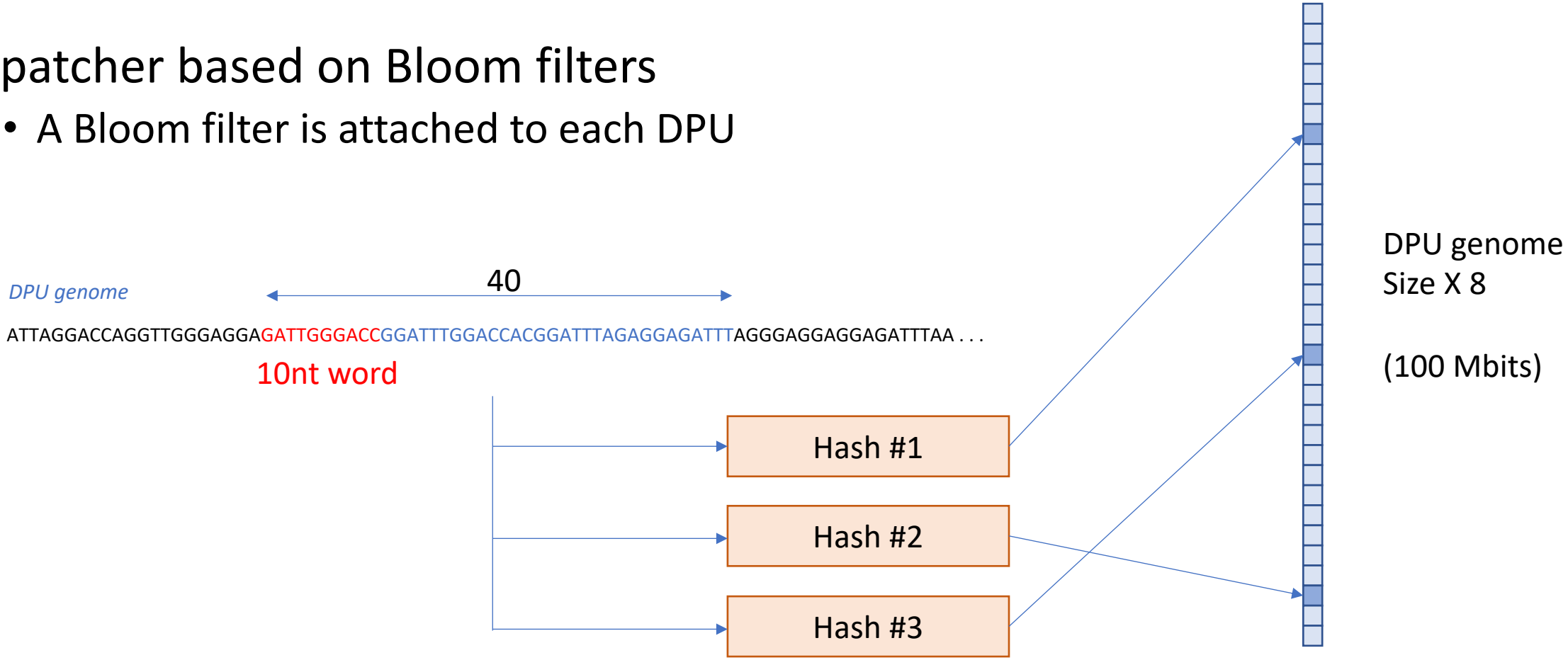
Genome index distributed on the DPUs



Read dispatcher

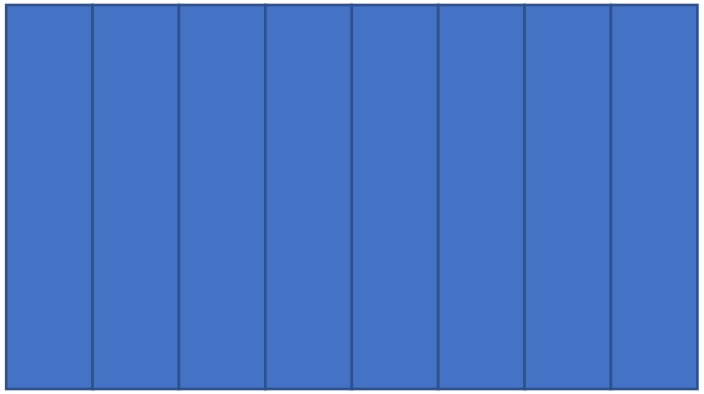
Dispatcher based on Bloom filters

- A Bloom filter is attached to each DPU

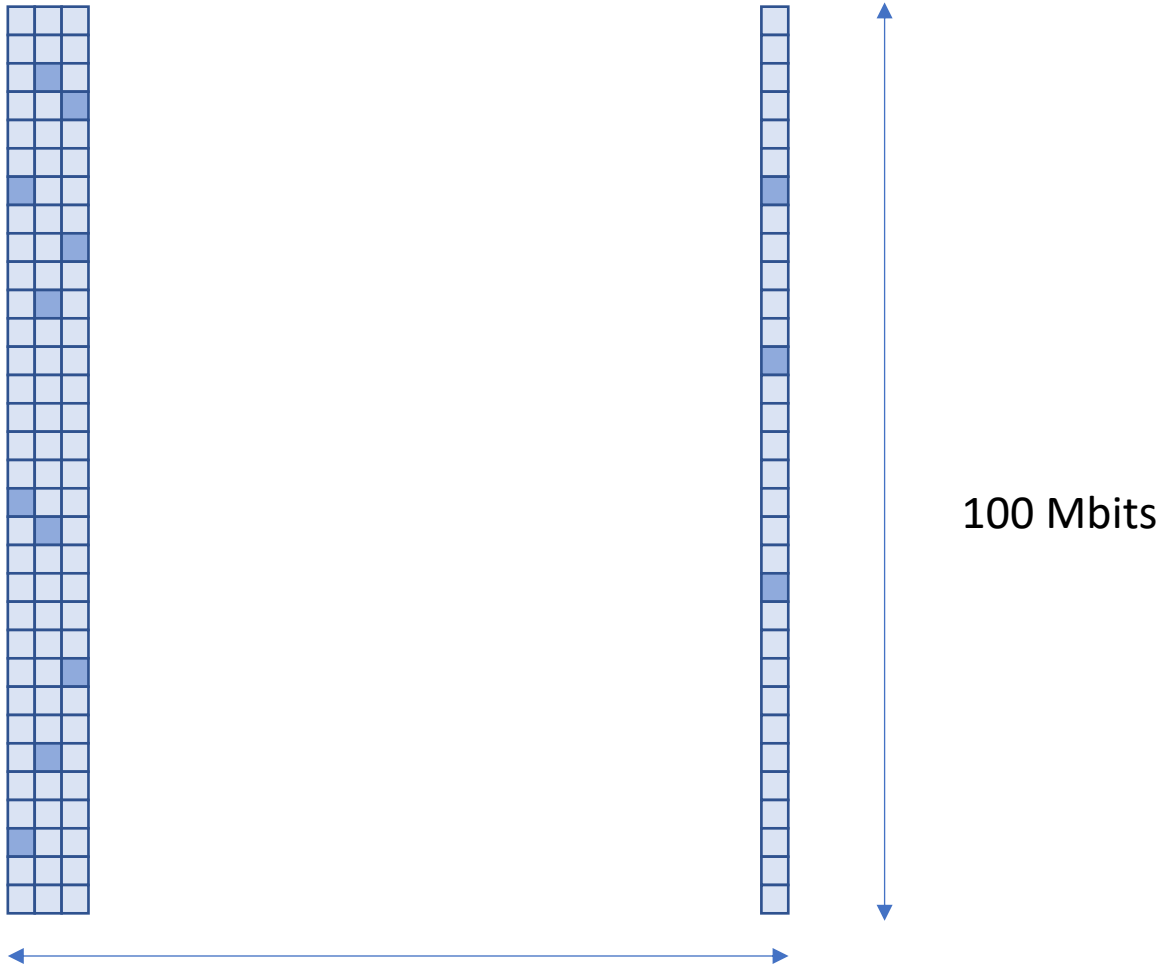


Host data organization

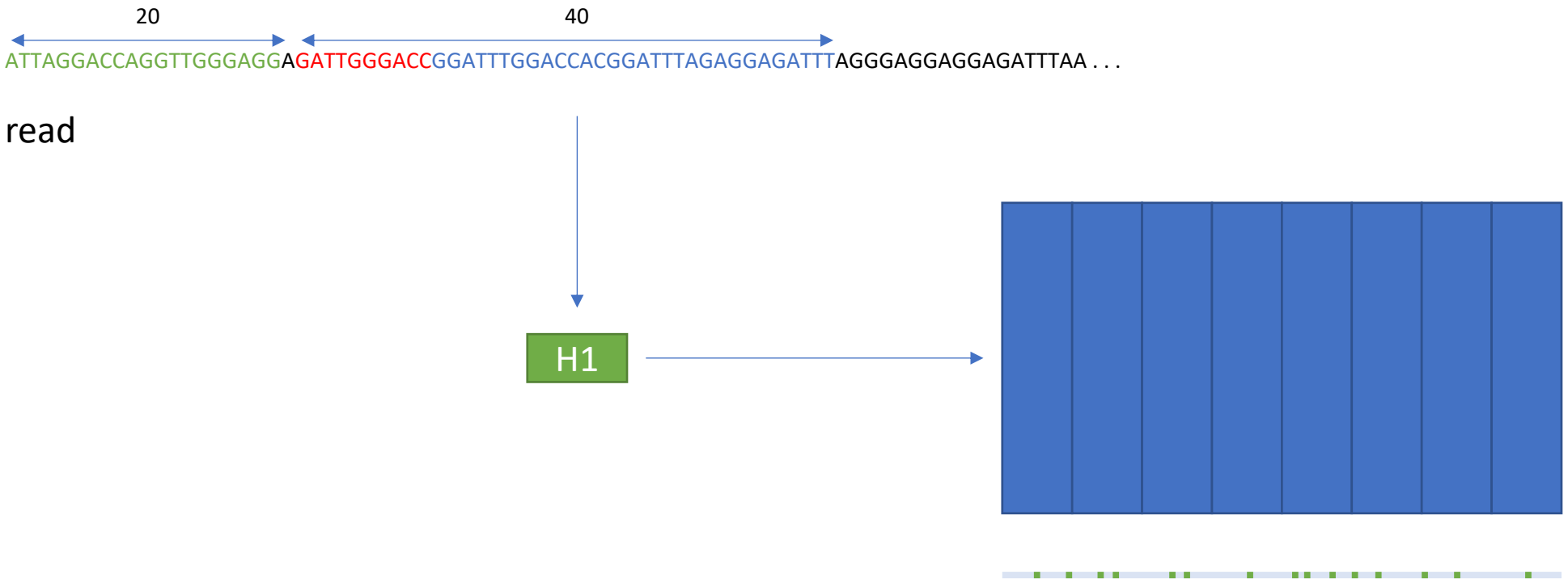
Size of Bloom filters : 3.2 GBytes



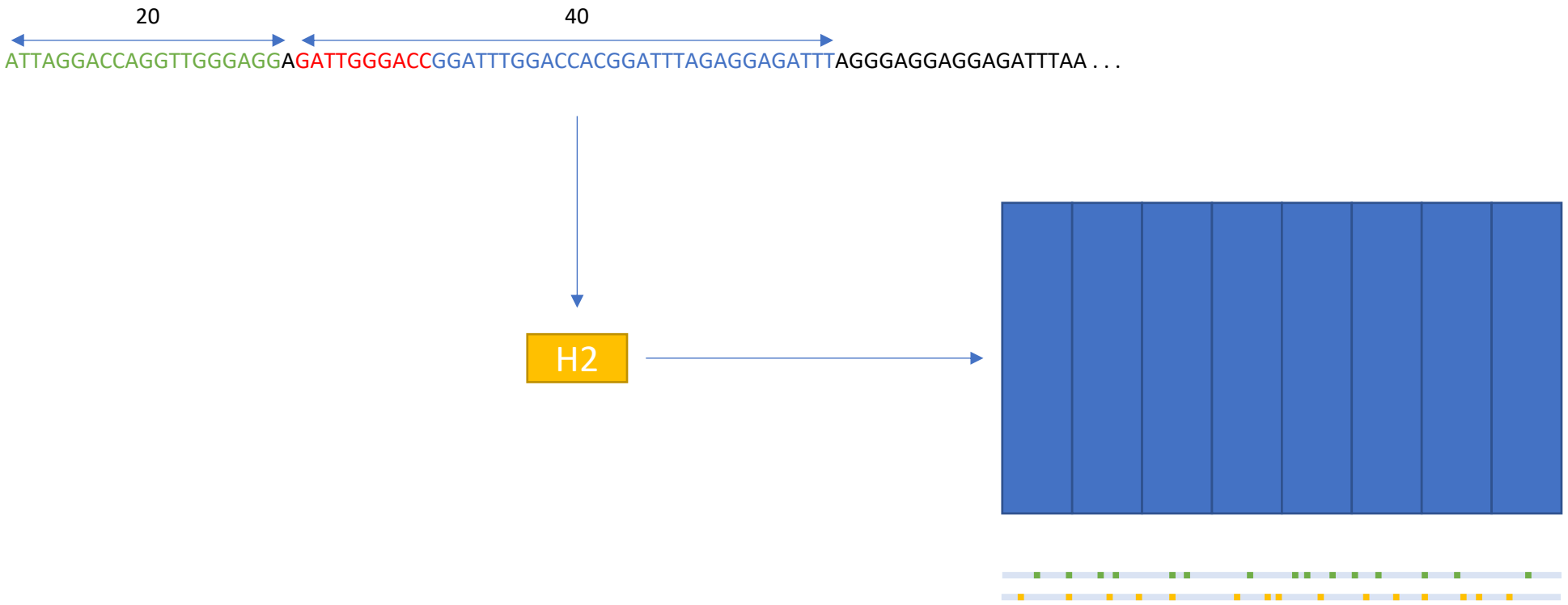
8 x 32bits array of size $100x10^6$



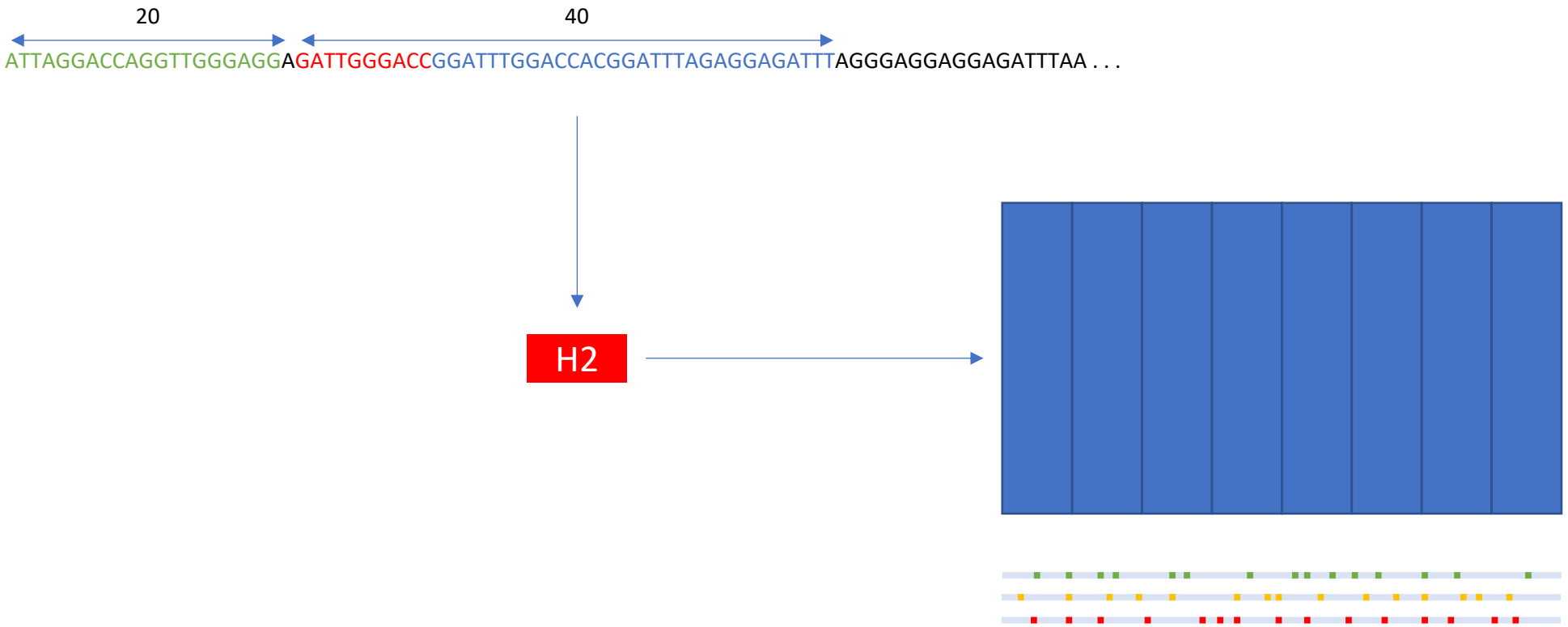
Read dispatching



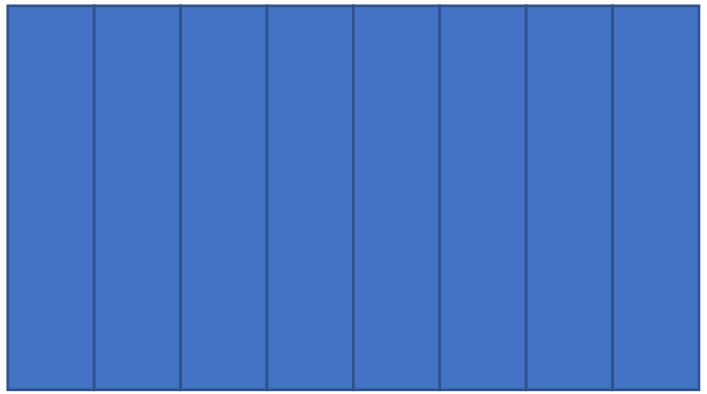
Read dispatching



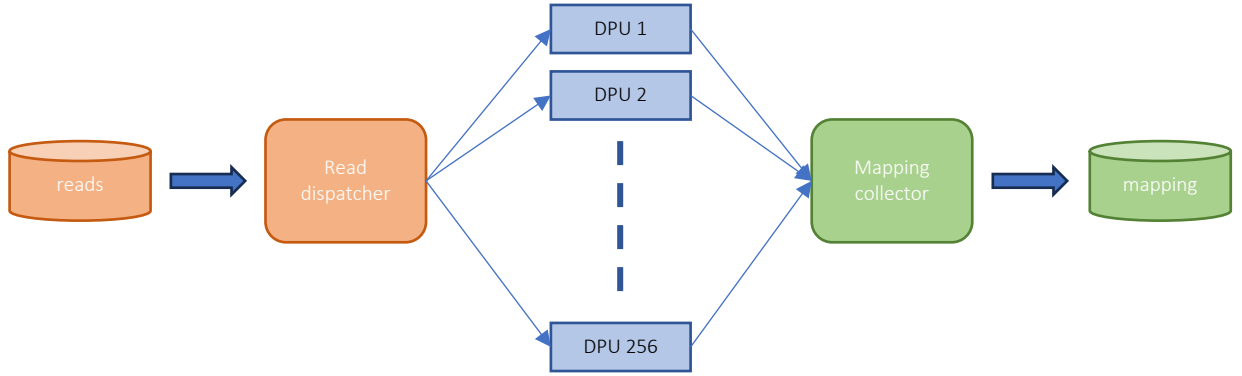
Read dispatching

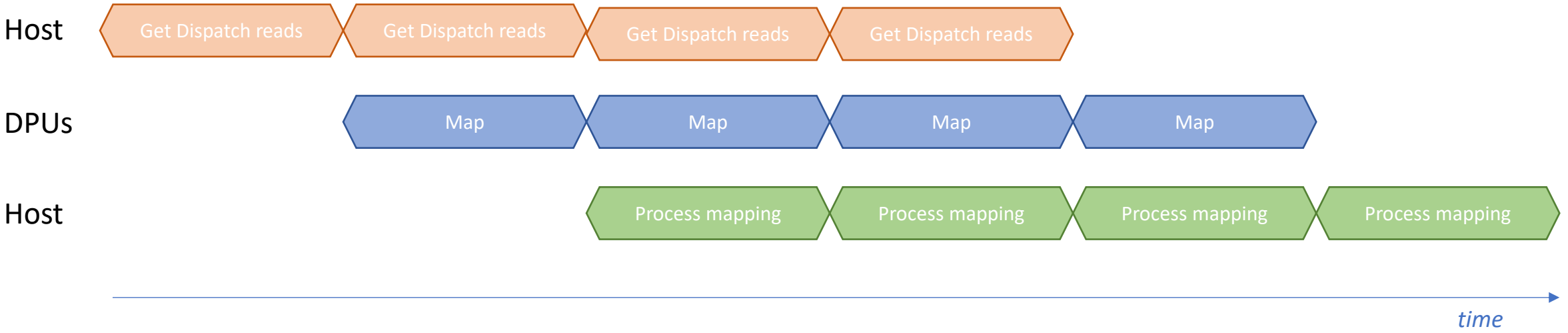


Read dispatching



Pipeline


reads
Read dispatcher
DPU 1
DPU 2
DPU 256
Mapping collector
mapping


Host
Get Dispatch reads
Get Dispatch reads
Get Dispatch reads
Get Dispatch reads
DPUs
Map
Map
Map
Map
Host
Process mapping
Process mapping
Process mapping
Process mapping
time

Results

Throughput: ~ 200 MBytes/sec (Fastq file)

Average CPU Load: 250 %

Mapping quality

N error	mapping error %
0	70.52
1	9.65
2	2.07
3	0.77
4	0.43

83.44 %

Protein database search

Pang: Protein Alignment with No Gap

First version of protein alignment on UPMEM PiM for feasibility study. On going work

Objectives

- Design a parallel implementation using both PiM and the host processor resources
  - PiM → DPU + MRAM
  - Processors → cores + legacy RAM

Seed-extend strategy:

1. Search hits
2. Extend hits
3. Compute alignments

DPU

Host

Hit search heuristics different from Blastp

- Provide similar sensitivity

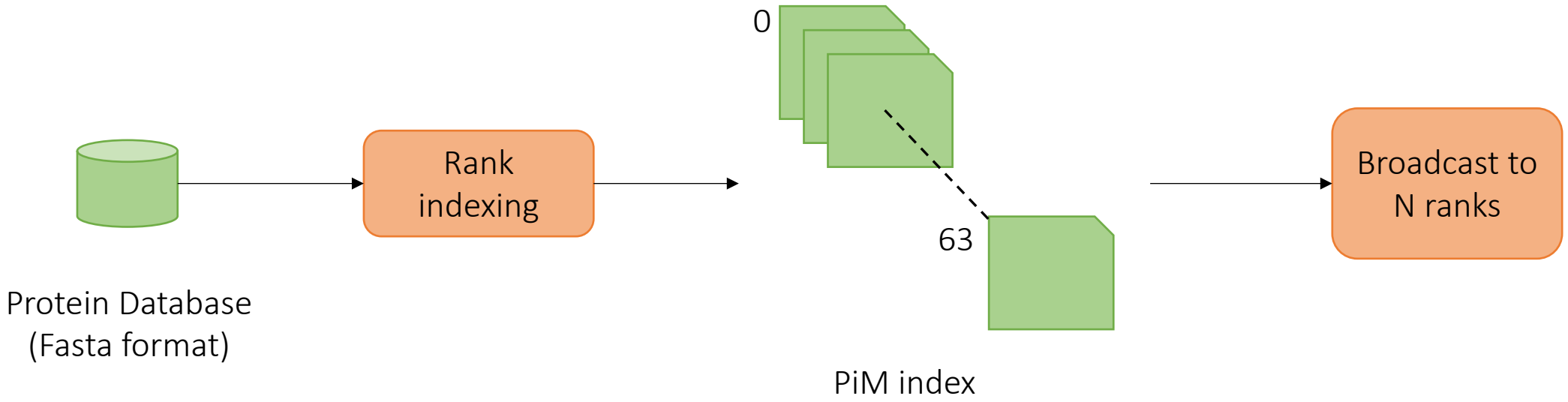
Implementation overview (1)

Step 1 : database indexing

The overall database index fit in a single rank (64 DPUs, 4 Gbytes)
The bank index is split over 64 MRAM DPUs

Rank

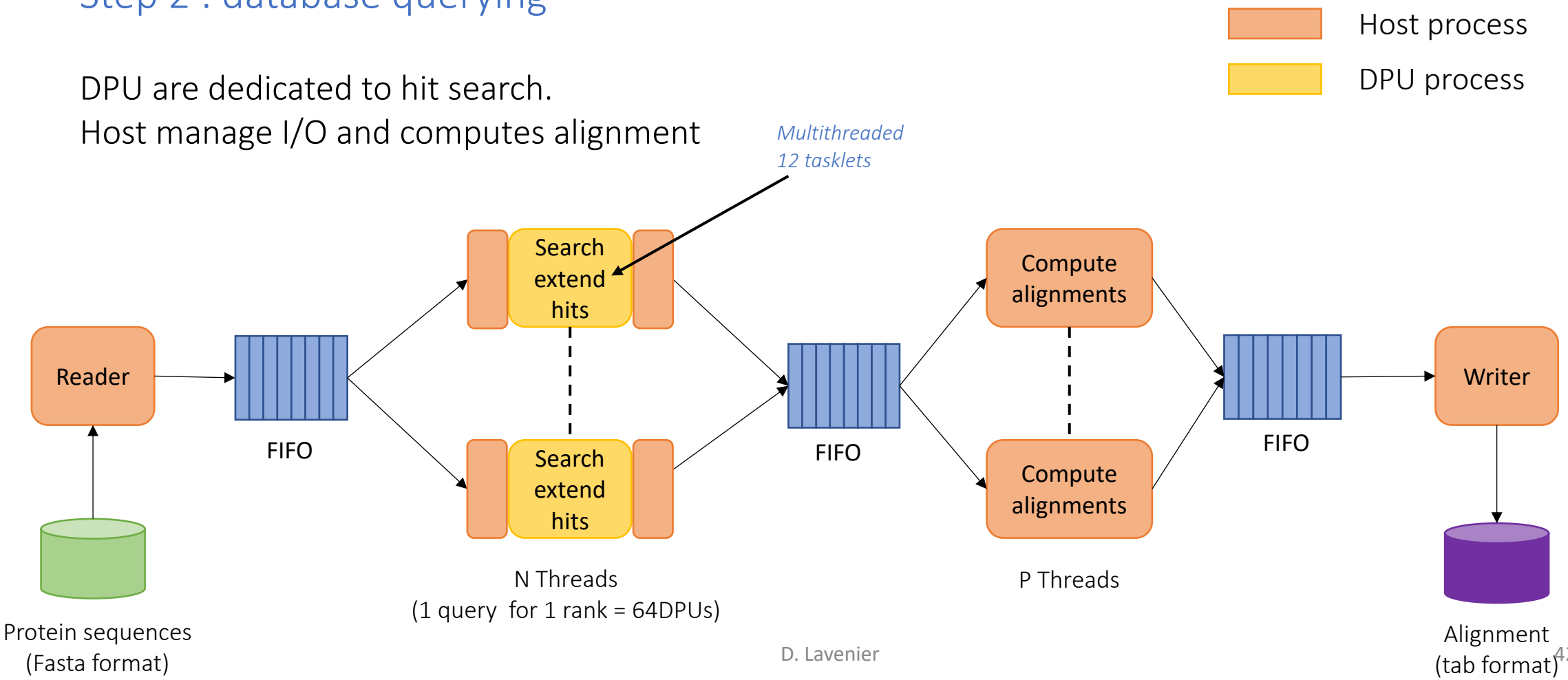
64 DPU on the same die
Optimized transfer



Implementation overview (2)

Step 2 : database querying

DPU are dedicated to hit search.
Host manage I/O and computes alignment



Experiments

Data sets

- Query
  - S1k.fa = 10^3 proteins
  - S10k.fa = 10^4 proteins
- Database = SwissProt, 563 000 proteins → index can fit on a rank (64 DPUs)

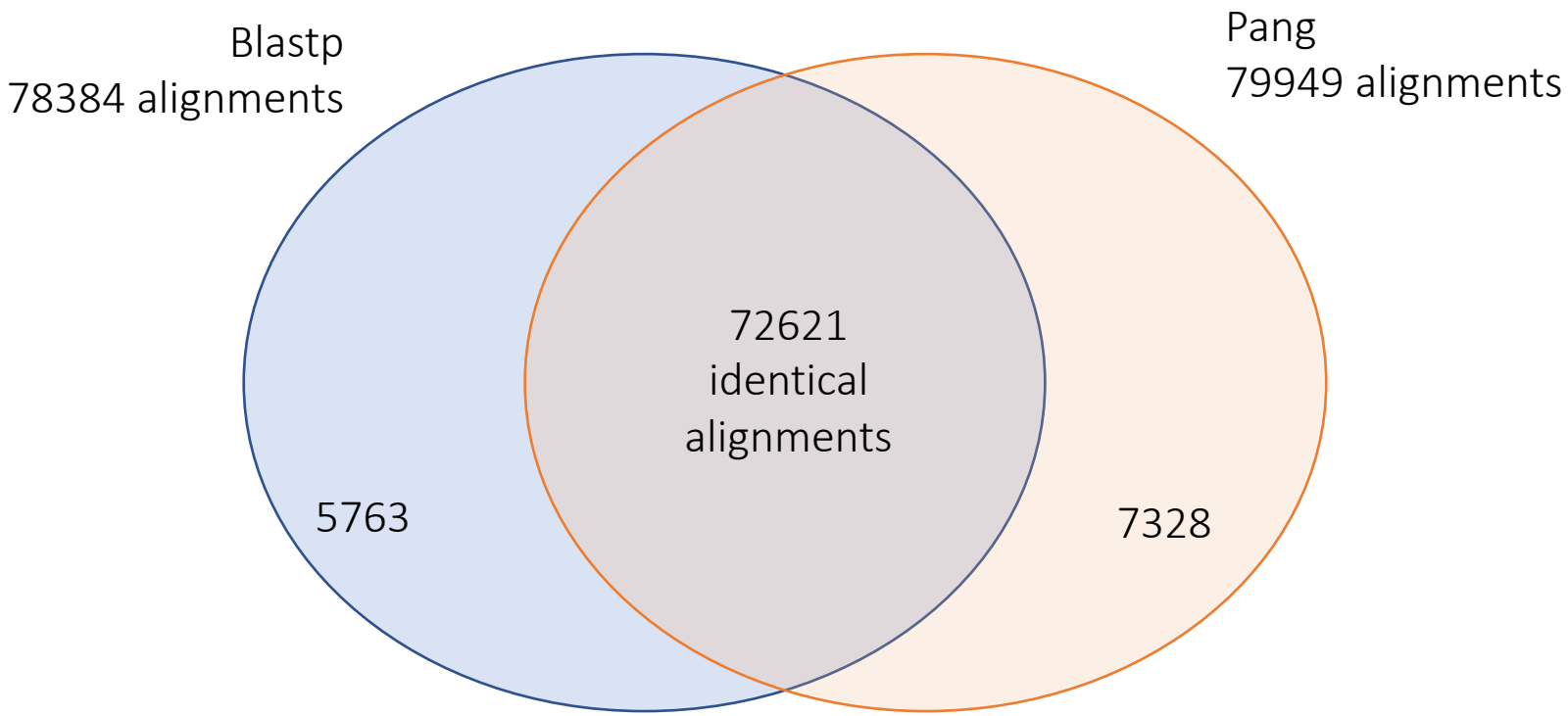
Hardware: UPMEM server

- CPU: 2 × Intel Xeon 4215 processeur, 2.5 GHz (16 cores)
- 256 Gbyes of standard memory
- 160 Gbytes of PiM memory (40 ranks of 64 DPUs)

Comparison with blastp (ungapped)

blastp -query s10k.fa -db sprot -ungapped -out align.blast -comp_based_stats F -evalue 1e-6 -seg yes -outfmt 6 -num_threads X

Pang vs Blastp output



Data : s1k.fa vs SwissProt

Pang speed-up vs Blastp

Rank

64 DPU on the same die
Optimized transfer

Run with 32 PiM ranks

Execution time: Blastp & Pang (in sec.) - 10^4 proteins

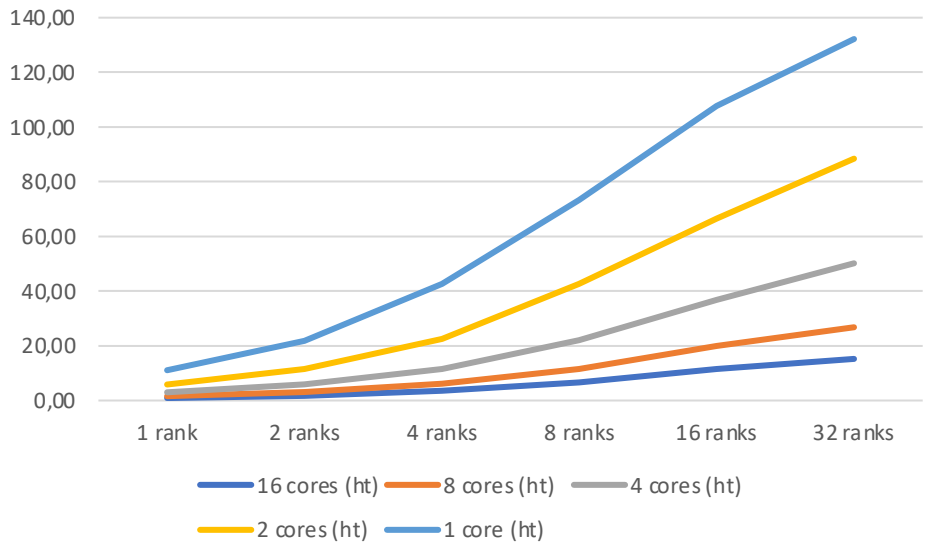
	blastp	Pang(1)	Pang (2)	Pang (4)	Pang (8)	Pang (16)	Pang (24)	Pang (32)
16 cores	844,00	938,14	473,26	240,98	127,15	73,37	62,51	55,46
8 cores	1490,00	939,83	475,31	243,12	128,75	74,42	61,13	55,62
4 cores	2821,00	939,72	475,00	242,66	128,34	76,87	62,06	56,22
2 cores	5495,00	939,05	475,32	242,94	128,63	82,50	67,48	62,13
1 core	10440,00	943,09	477,64	244,97	142,47	96,83	82,23	78,98

Pang time include database indexing
Taskset linux command is used to constrain the number of cores
Hyper threading is active

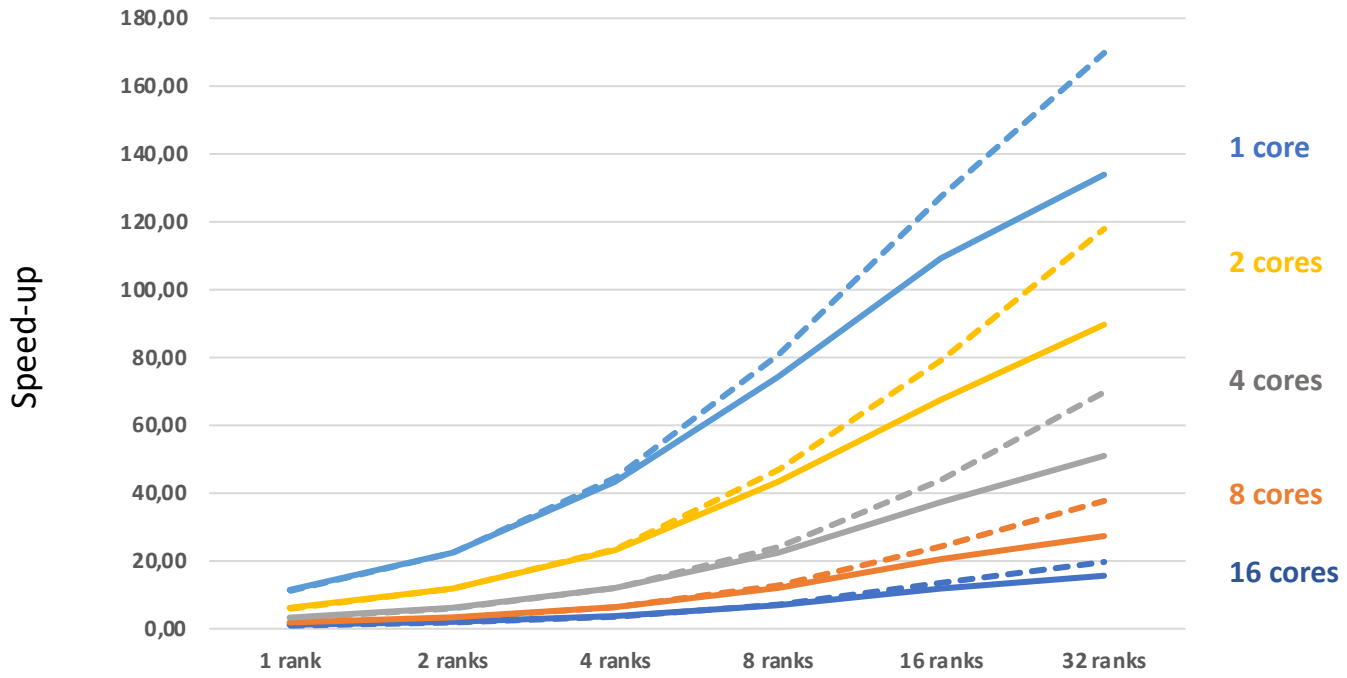
Speed-up = Blastp time / Pang time

Pang speed-up

	1 rank	2 ranks	4 ranks	8 ranks	16 ranks	32 ranks
16 cores	0,90	1,78	3,50	6,64	11,50	15,22
8 cores	1,59	3,13	6,13	11,57	20,02	26,79
4 cores	3,00	5,94	11,63	21,98	36,70	50,18
2 cores	5,85	11,56	22,62	42,72	66,61	88,44
1 core	11,07	21,86	42,62	73,28	107,82	132,19



Pang speed-up vs Blastp



Without DB loading time

Speed-up = blast_time / (Pang_time – load_db_time)

With DB loading time

Speed-up = blast_time / (Pang_time)

Conclusion

Good match between PiM and genomics/bioinformatics applications

- High volume of data
- Independent computation → high potential parallelism
- Rather simple calculation (string processing)

Revisit algorithm for massive parallelism

- UPMEM server: 2560 DPUs, 12 tasklets → 30720 threads

Programming model

- Required for PiM paradigm to be adopted by the community

Not specific to genomics

The 5th Workshop on

ACCELERATOR ARCHITECTURE IN COMPUTATIONAL BIOLOGY AND BIOINFORMATICS

June 18th, 2023

In conjunction with 50th IEEE International Symposium on Computer Architecture

Orlando, Florida, USA

Thank you for your attention

Dominique Lavenier

GenScale group, University of Rennes, IRISA-CNRS, Inria

Université de Rennes

UMR IRISA

Inria