The 11th IEEE Non-Volatile Memory Systems and Applications Symposium, IEEE NVMSA 2022

RNA-seq Quantification on Processing in memory Architecture: Observation and Characterization

Liang-Chi Chen[1,4], Shu-Qi Yu[2,4], Chien-Chung Ho[1], Yuan-Hao Chang[4], Da-Wei Chang[1], Wei-Chen Wang[2,3], Yu-Ming Chang[5]

[1] CSIE, National Cheng Kung University, Taiwan

[2] CSIE, National Taiwan University, Taiwan

[3] EECS, Massachusetts Institute of Technology, United States

[4] IIS, Academia Sinica, Taiwan

[5] Wolley Inc., Taiwan

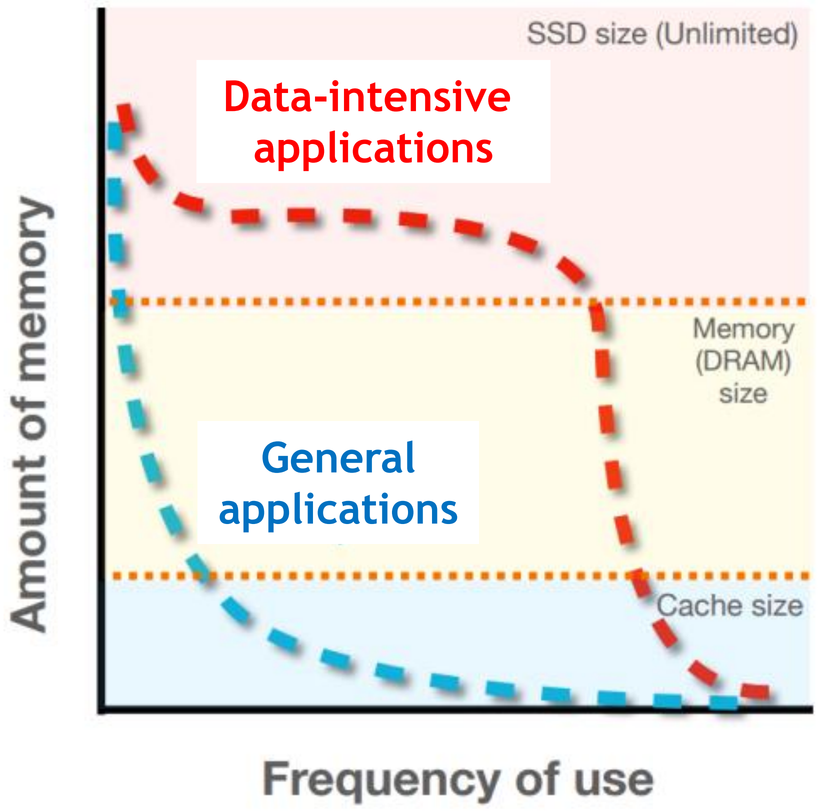
Aug. 23, 2022, Taipei, Taiwan

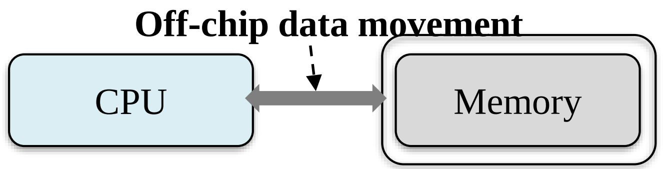
Outline

Outline

Introduction

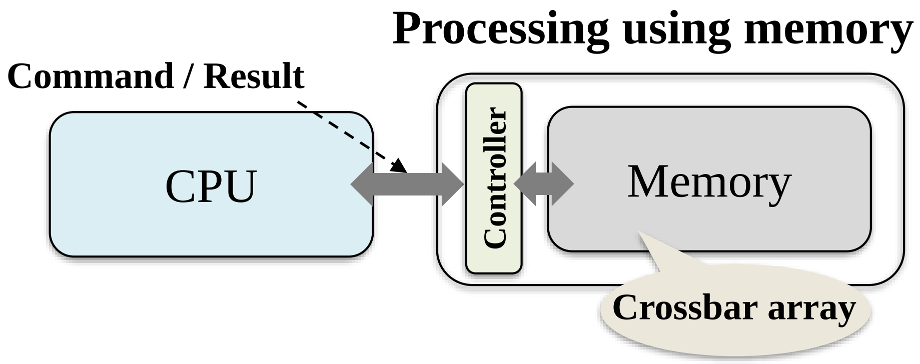
- In the past, most of applications perform very well on traditional architecture. (blue line)
- However, modern applications need lots of data for computation during run time. (red line)
  - Neural network
  - Genome sequencing
  - Graph processing
- Because of small size of cache, CPU has to spend a lot of time to access data in memory/storage.
- Processing-in-memory (PIM)

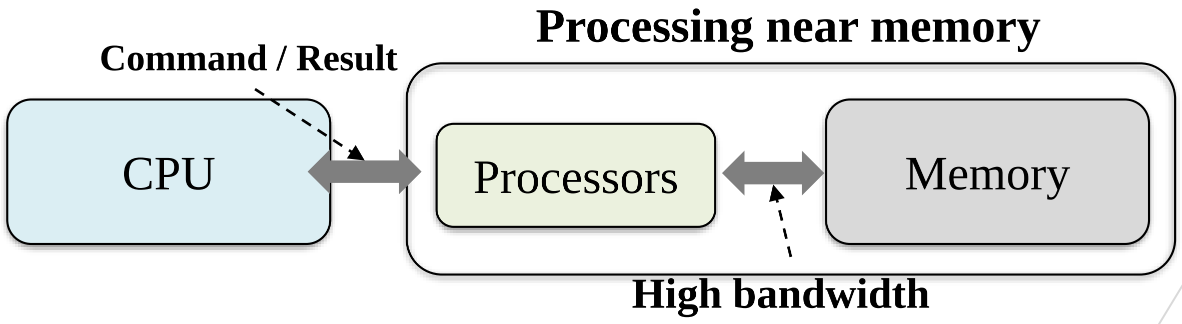




Introduction

- Processing-in-memory (PIM)
- We need to process data near/in the memory
- To support PIM in modern architectures
  - processing using memory : E.g., Neural network computation on ReRAM.
  - processing near memory : E.g., UPMEM DPU.
- The objective of this work is to enable a strong understanding of the characterizations and design tradeoffs of UPMEM DPU.



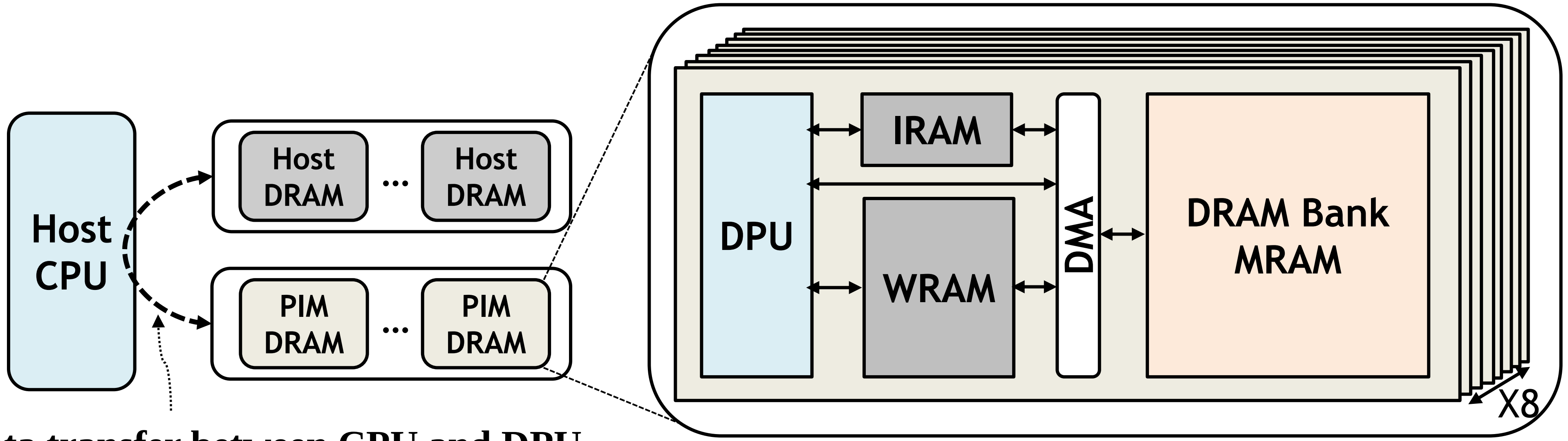


Outline

Background: UPMEM DPU[1]

- **UPMEM DRAM Processing Units (DPU)**
  - A DPU is a 32-bit RISC core with 24 hardware threads (tasklets)
  - 24KB IRAM (instruction memory)
  - 64KB WRAM (working memory , i.e. cache)
  - 64MB MRAM (main memory , i.e. DRAM bank)

PIM DRAM Chip

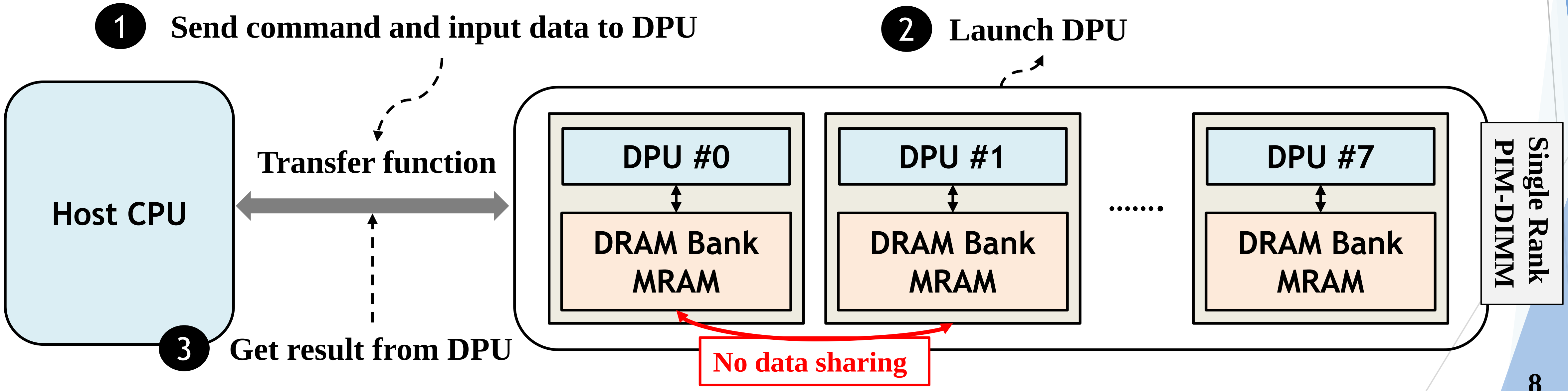


Data transfer between CPU and DPU

[1] Devaux, Fabrice. "The true processing in memory accelerator." 2019 IEEE Hot Chips 31 Symposium (HCS). IEEE Computer Society, 2019.

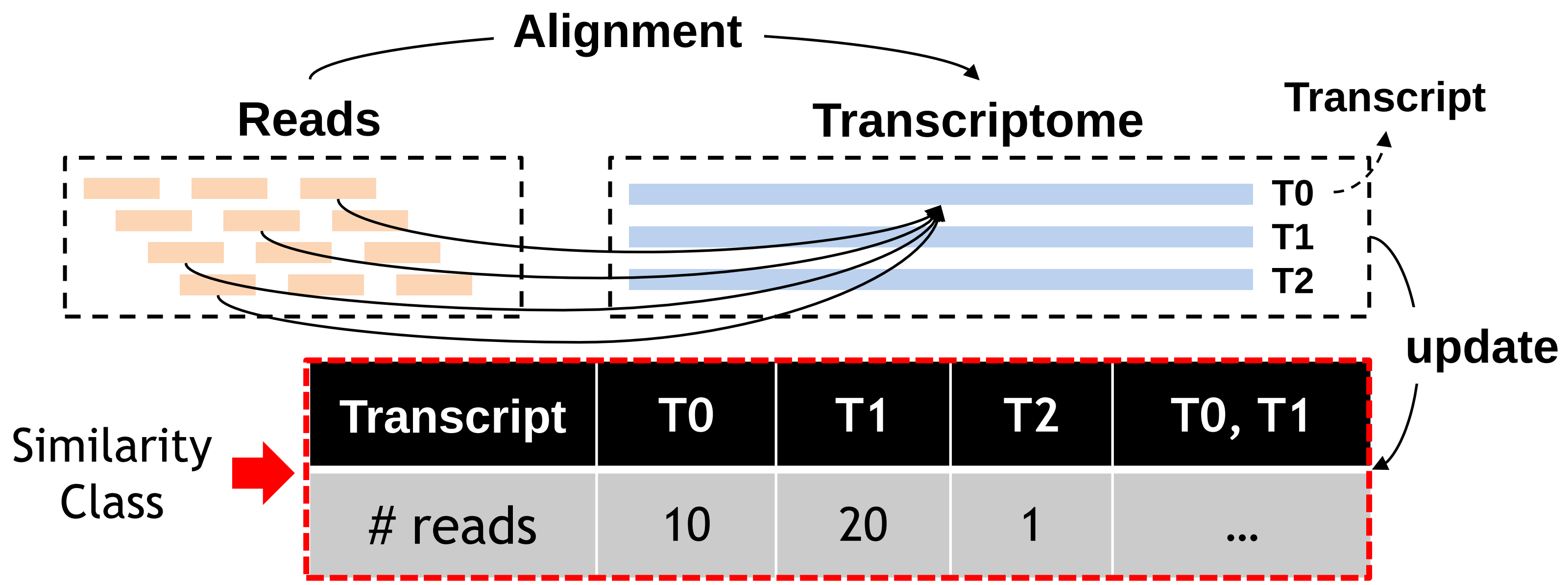
Background: UPMEM DPU[1]

- DPU constraints and limitation
  - CPU access MRAM only by transfer APIs
  - Granularity of communication: Rank
  - Data transfer and DPU execution can't work concurrently
  - No data sharing among DPUs



Background: RNA-seq Quantification

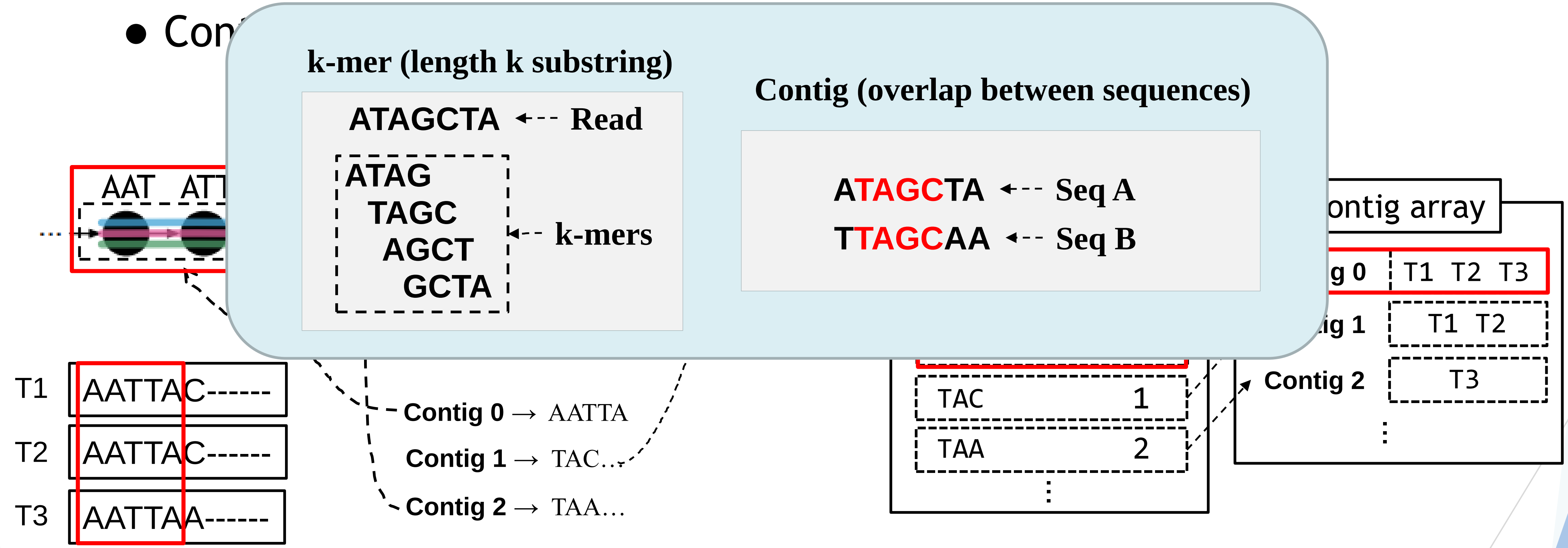
- RNA sequences(RNA-seq) quantification
  - To know the quality of a set of sequences
  - The most time-consuming step is read alignment step
    - The inputs are reads and transcriptome
    - The output is a similarity class



Transcript	T0	T1	T2	T0, T1
# reads	10	20	1	…

Background: kallisto [2]

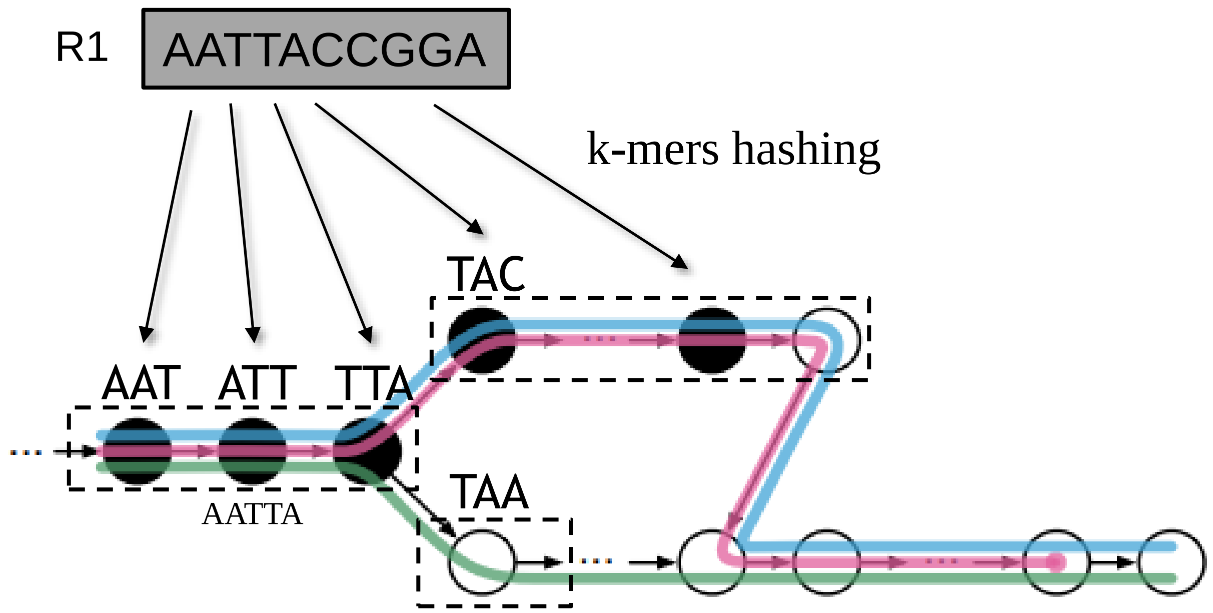
- kallisto implementation
  - de Bruijn Graph → hash table (k-mer, contig)
  - Con

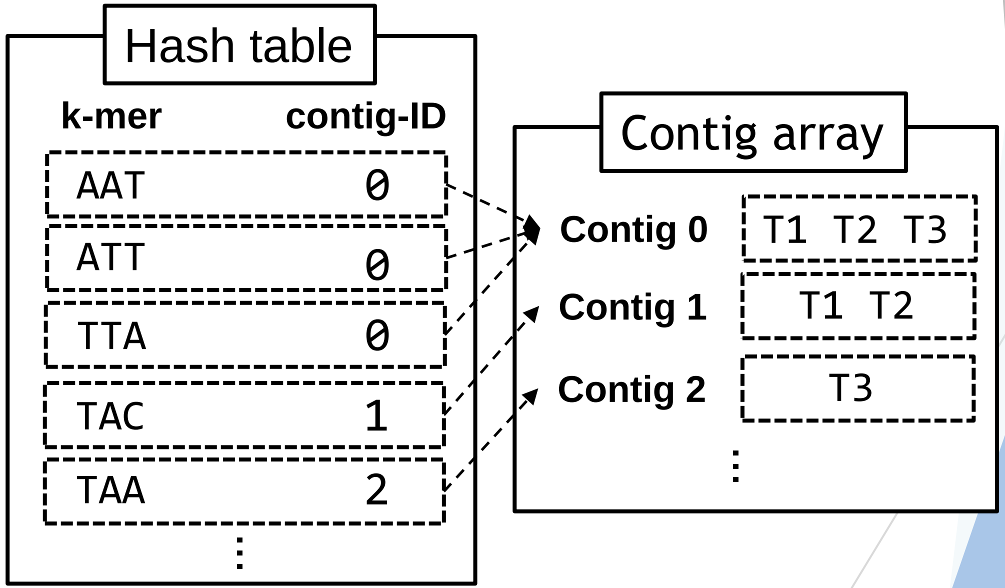


[2] Bray, Nicolas L., et al. "Near-optimal probabilistic RNA-seq quantification." Nature biotechnology 34.5 (2016): 525-527.

Background: kallisto

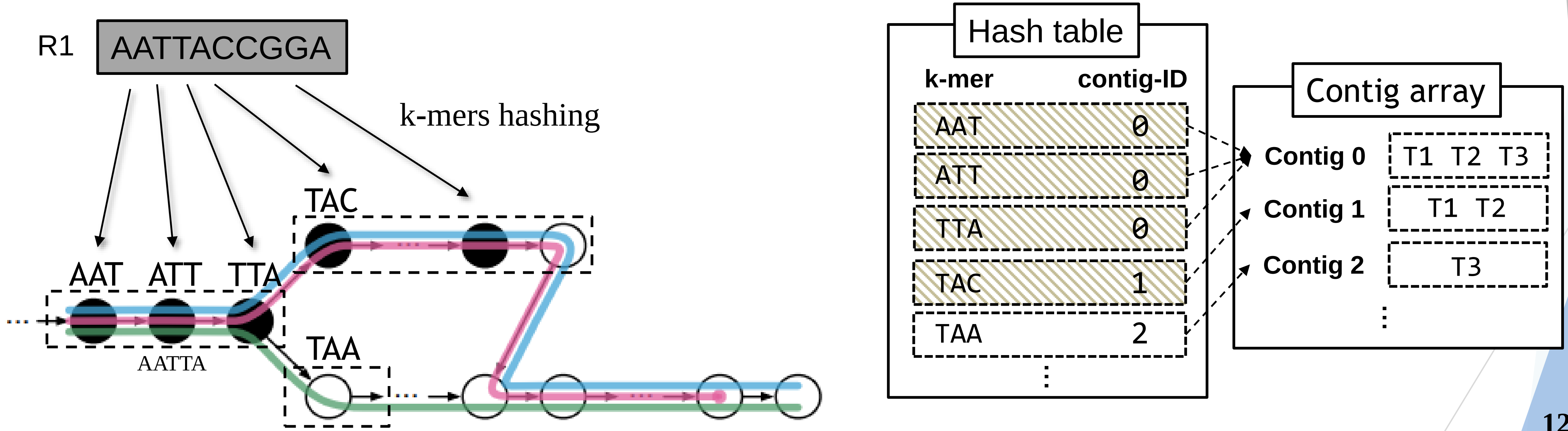
- Read mapping
  - Hash k-mers in read to nodes in the graph





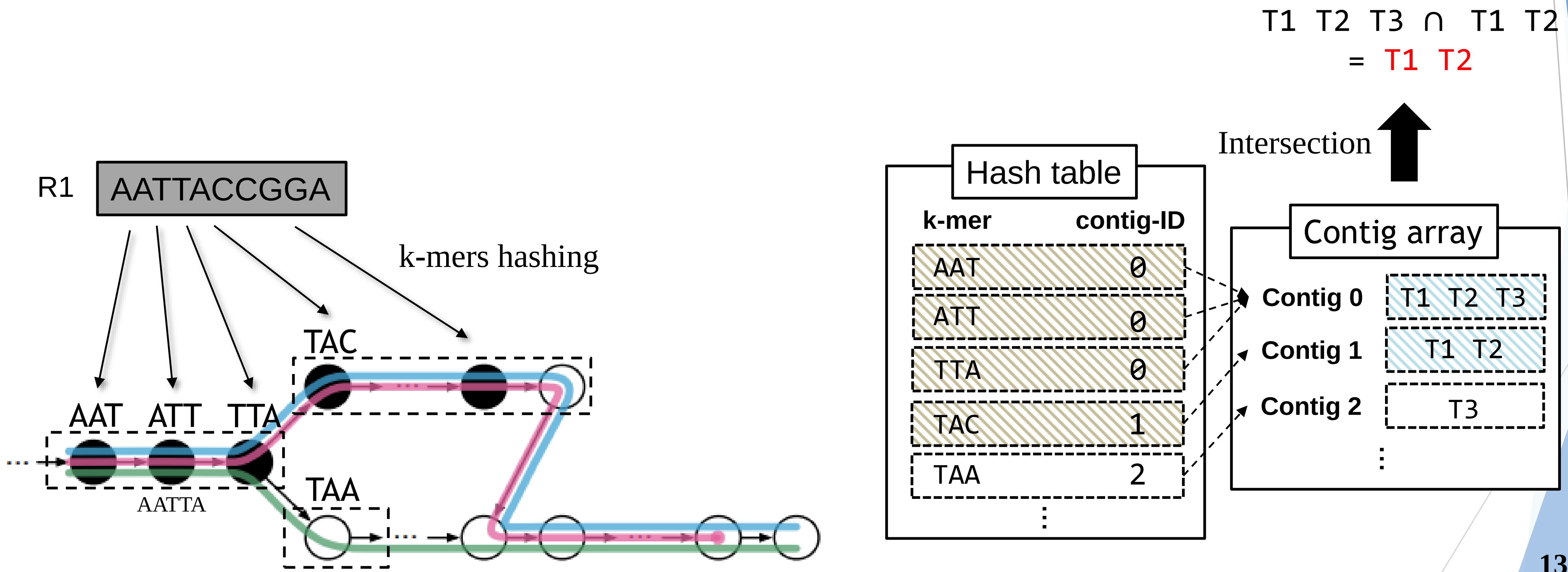
Background: kallisto

- Read mapping
  - Hash k-mers in read to nodes in the graph



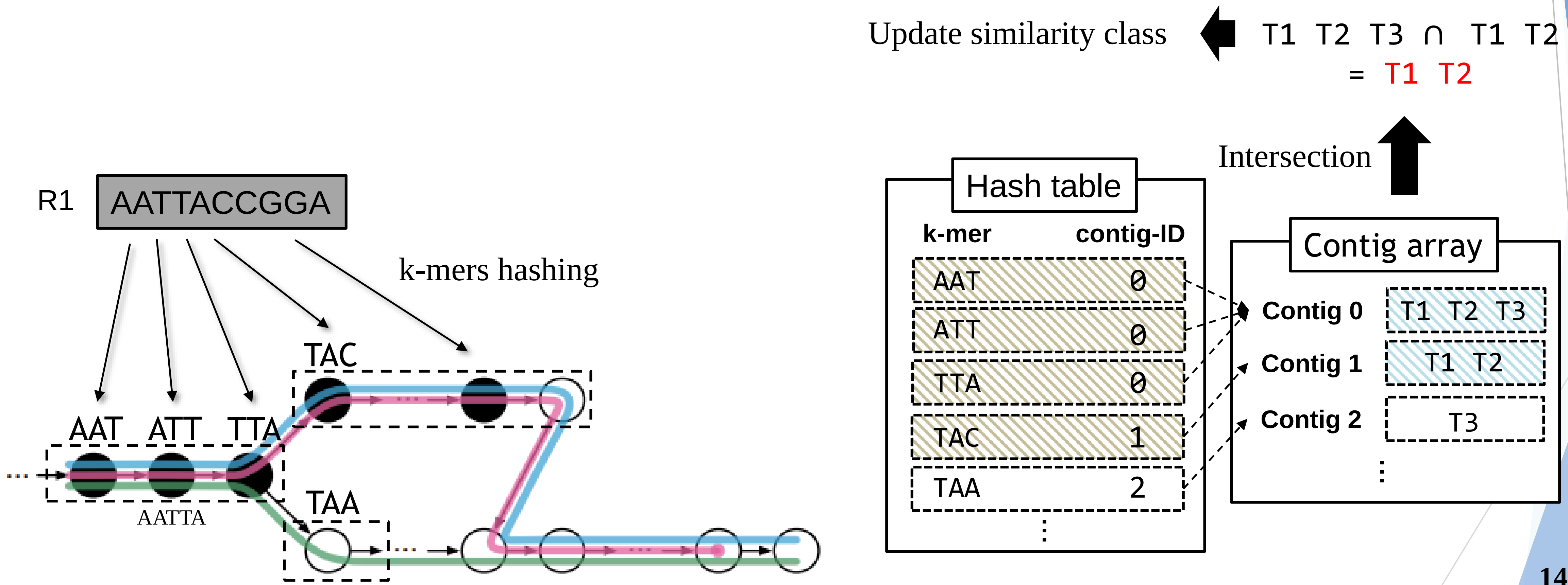
Background: kallisto

- Read mapping
  - Hash k-mers in read to nodes in the graph



Background: kallisto

- Read mapping
  - Hash k-mers in read to nodes in the graph



Outline

Motivation

- Sequencing usually needs large number of reads (10M-100M).
  - PIM is a possible way to increase application throughput

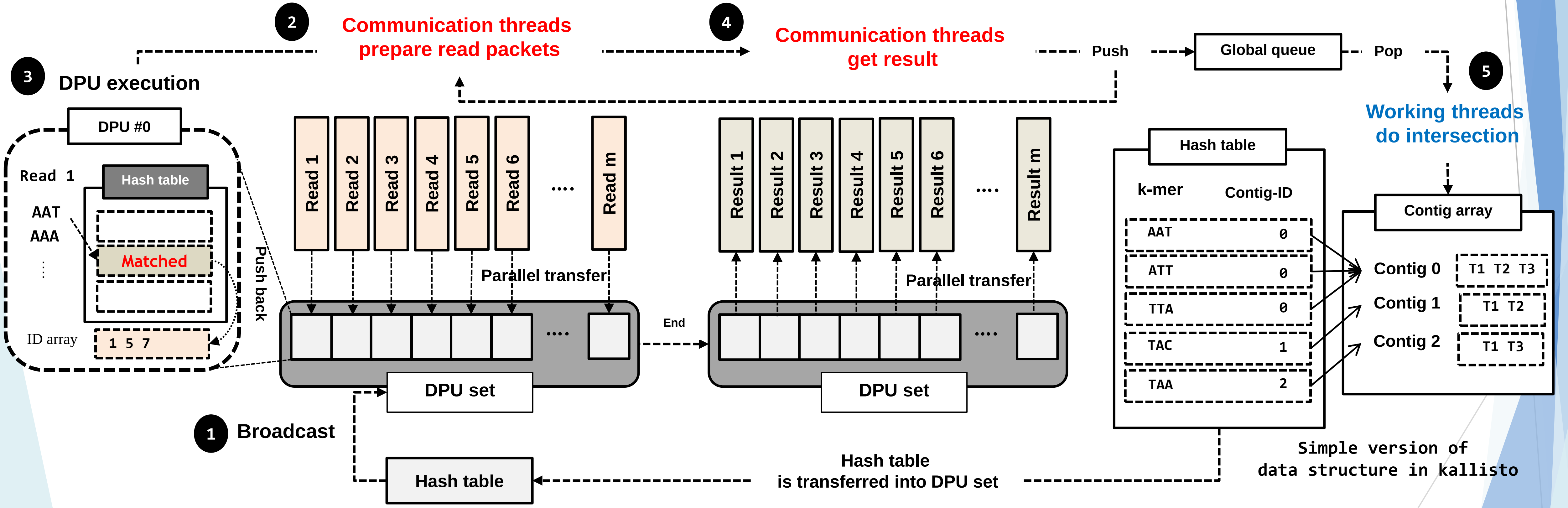
- Sequencing on UPMEM system may have some concerns
  - DPU constrains: e.g. no data sharing between DPUs
  - Frequent data movement between CPU and DPU
  - DPU-based software design

- We choose RNA-seq quantification to be a case study, understanding the characteristics of sequencing on UPMEM.

Outline

RNA-seq Quantification on DPU

- Design overview of DPU-based kallisto(D_kallisto)
  - The communication thread is responsible for reads transfer and waiting for the result.
  - The working thread is responsible for intersection task.

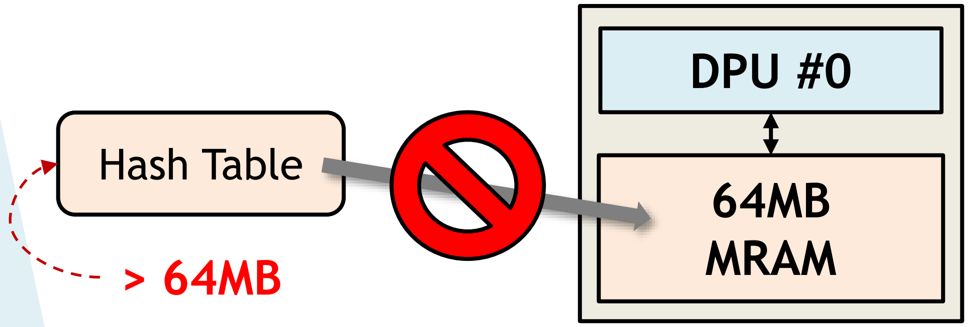


Outline

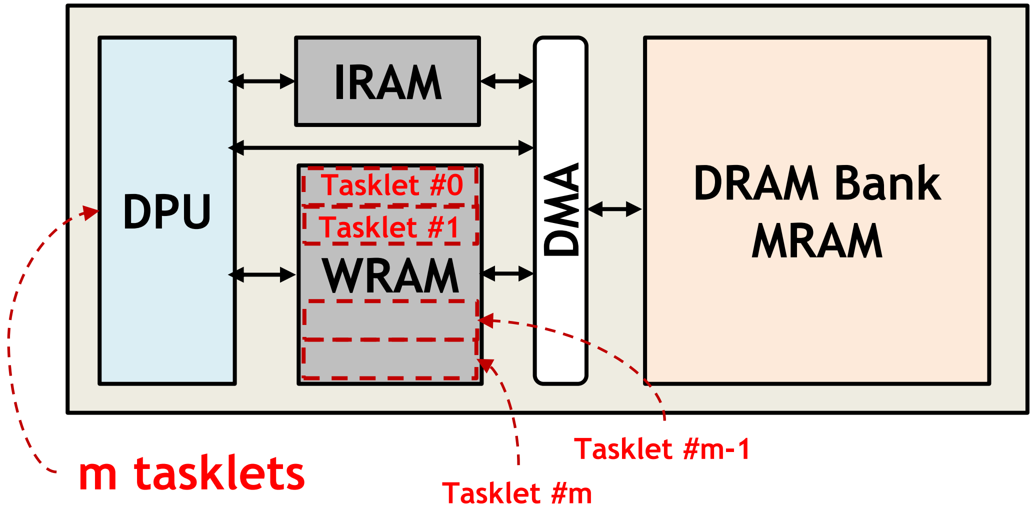
- Introduction
- Background
- Motivation
- RNA-seq Quantification on DPU
- **Design Tradeoffs and Evaluations**
  - Large Hash Table
  - DPU Programming issues
- Conclusion

Design Challenges and Tradeoffs

- **Challenges**
  1. Hash tables can be larger than 64MB (DPU MRAM size).
     A. Replacement policy
     B. Split policy
  2. Programming issues caused by limited size of DPU WRAM.
     A. long cache
     B. 128-byte cache



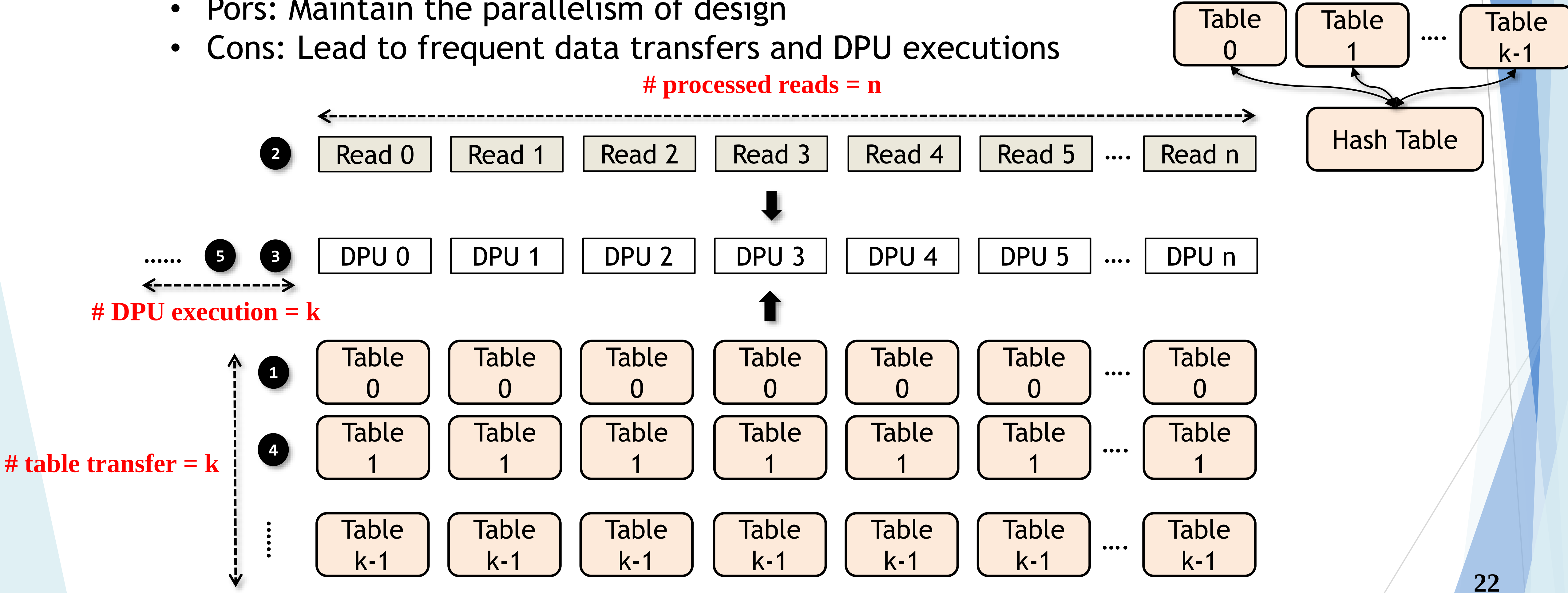
More number of tasklets in execution,
the **less WRAM space each tasklet** can be allocated.



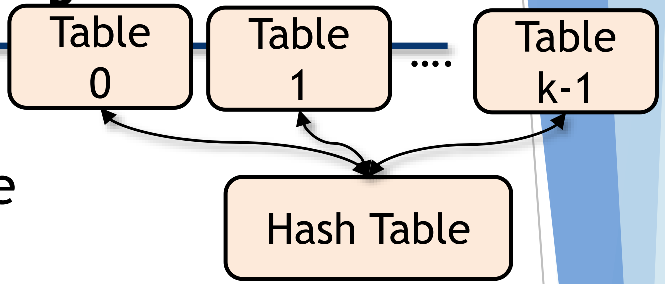
Outline

Large Hash table: Replacement Policy

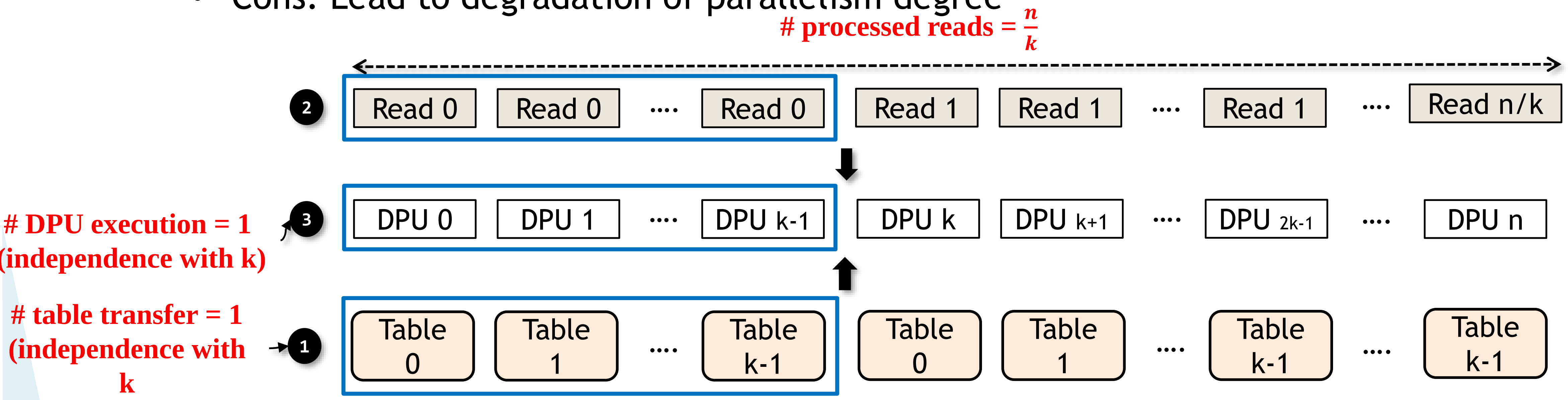
- Replacement policy
  - All the available DPUs to obtain the same sub-table at the same time
  - Pors: Maintain the parallelism of design
  - Cons: Lead to frequent data transfers and DPU executions



Large Hash table: Split Policy

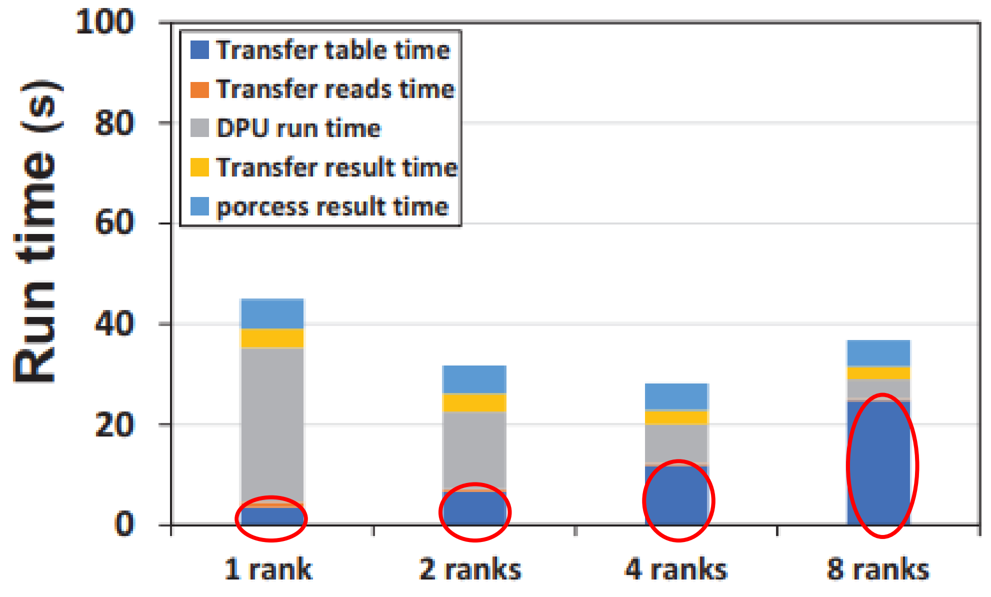


- Split policy
  - Every k DPUs to obtain the different sub-tables at the same time
  - Pors: Only one DPU execution and table transfer
  - Cons: Lead to degradation of parallelism degree

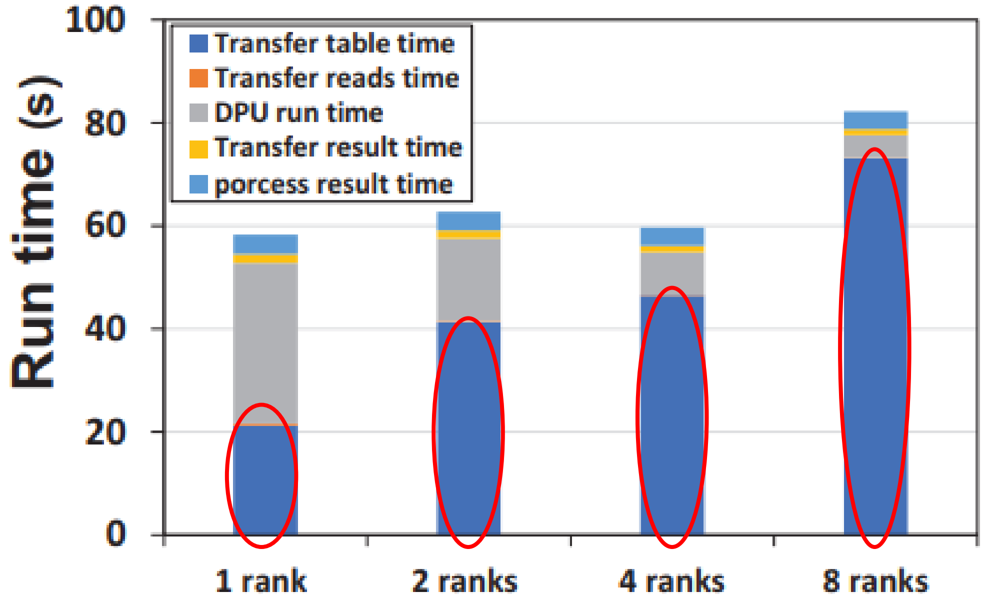


Experimental Result: Split Policy vs Replacement Policy

- Execution time breakdown.
- Replacement policy incurs lots of data transfer between CPU and DPU.



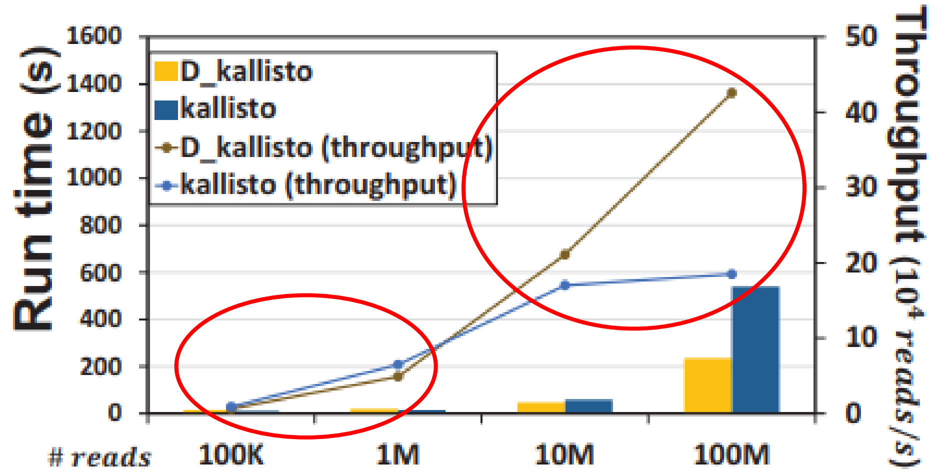
(a) Split policy.



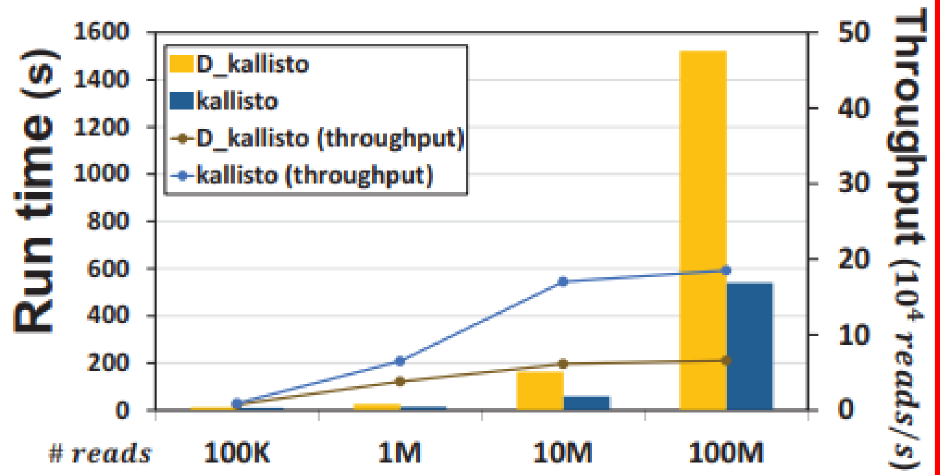
(b) Replacement policy

Experimental Result: Split Policy vs Replacement Policy

- D_kallisto with replacement policy is less efficient than CPU-based kallisto.
- Split policy performs much better with large data size.



(a) Split policy



(b) Replacement policy

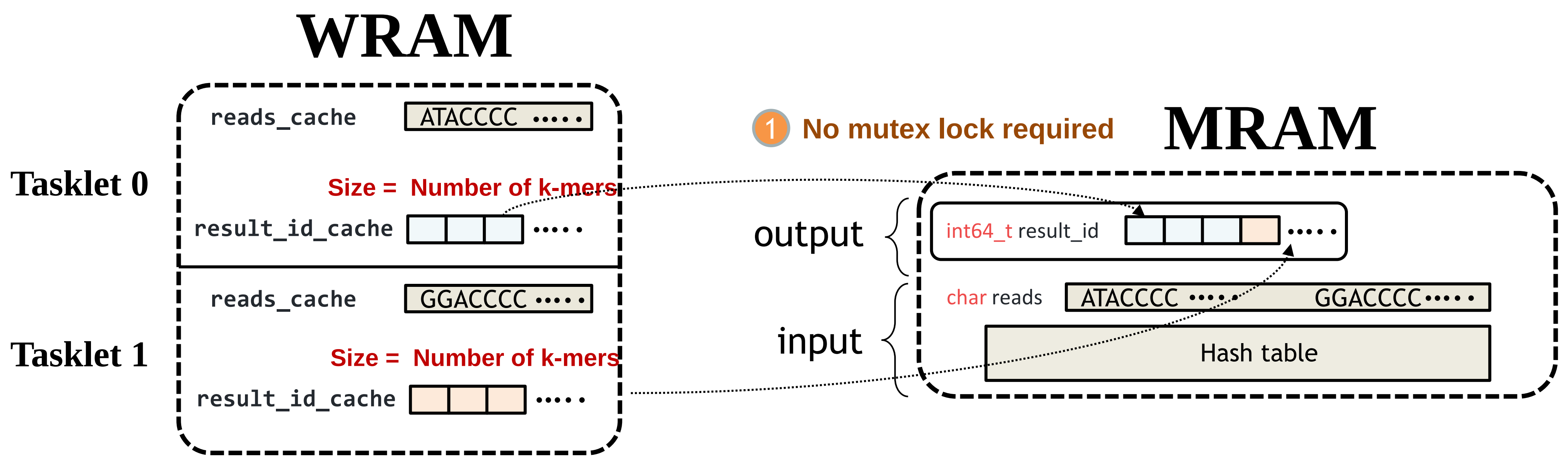
Frequently data transfers is an overhead in DPU system.

UPMEM DPU system is a material suitable for large data size workloads.

Outline

DPU Programming Issues: Long Cache

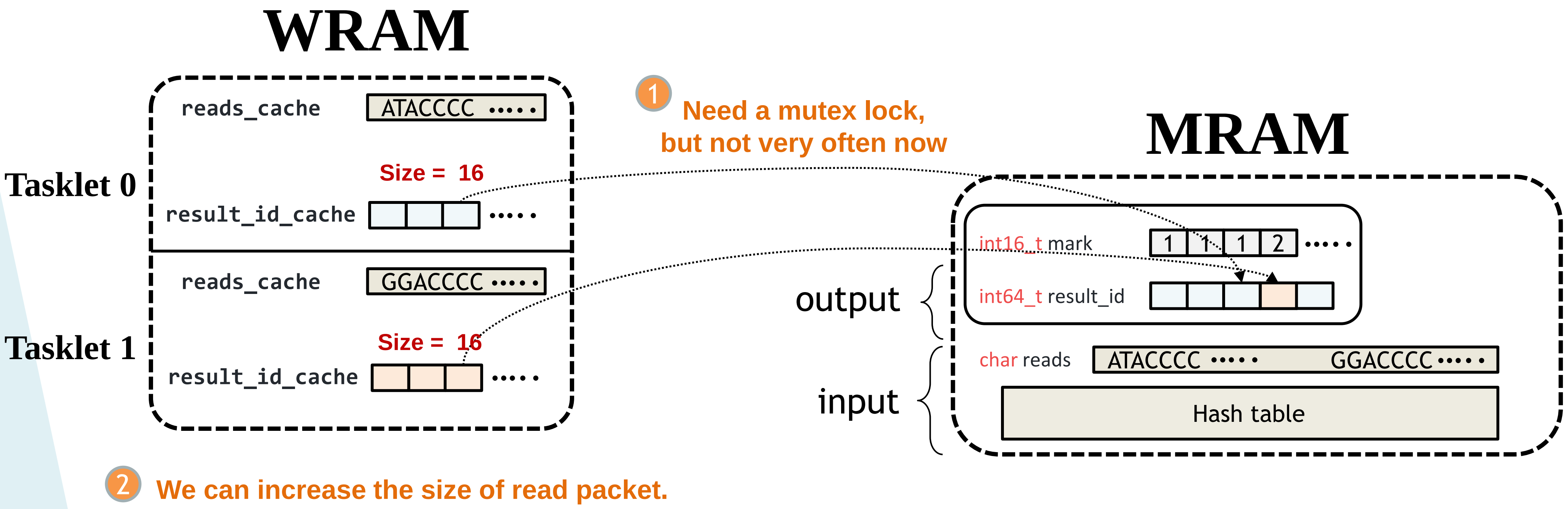
- Long cache
  - Avoid mutex locks to maintain the program efficiency.
  - Long cache keeps all results in WRAM until all reads are processed.



2 This implementation results in limited numbers of reads and tasklets .

DPU Programming Issues: 128-byte Cache

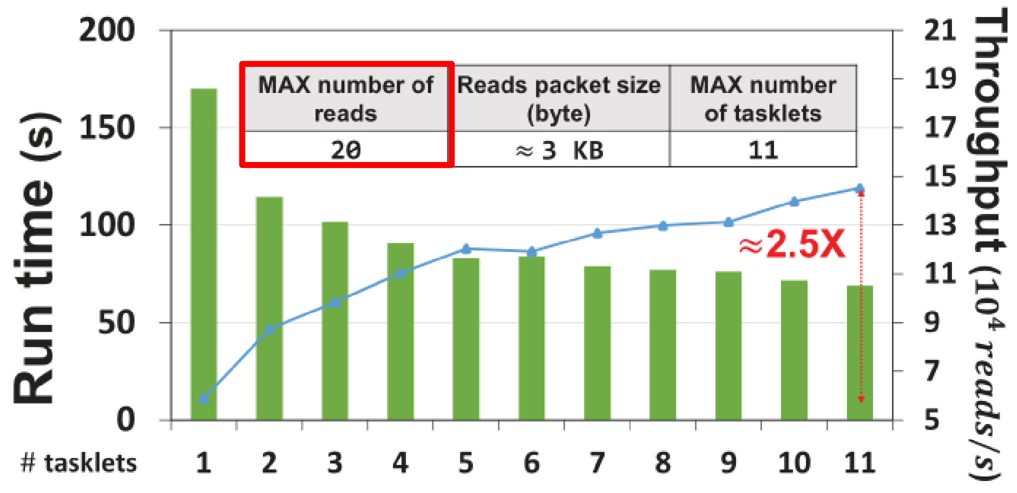
- 128-byte cache and mutex lock
  - We reduce cache size to 128 bytes (16 × 64 bits) [3].
  - Although it needs a mutex lock in MRAM, it increases the size of read packet.



[3] Gómez-Luna, Juan, et al. "Benchmarking a new paradigm: An experimental analysis of a real processing-in-memory architecture." arXiv preprint arXiv:2105.03814 (2021).

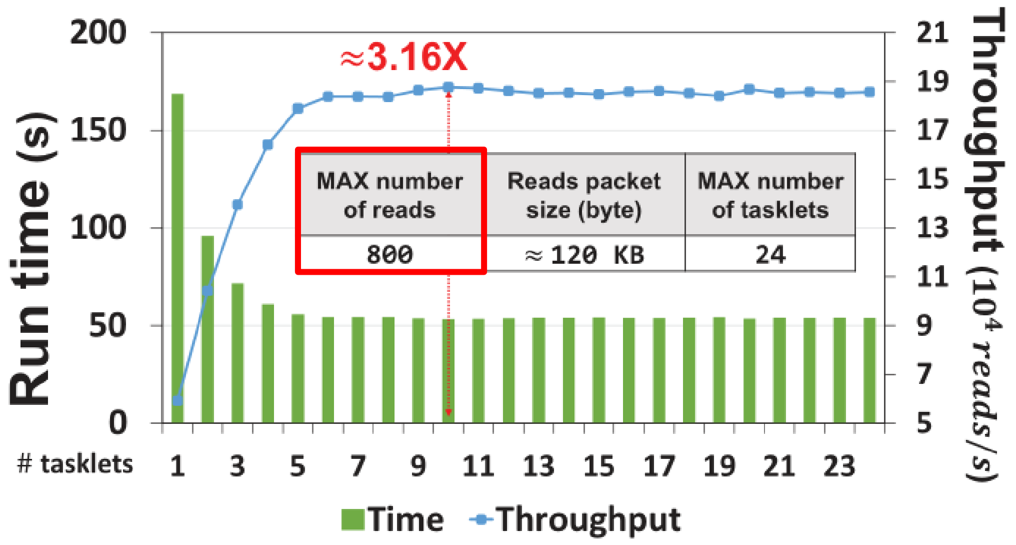
Experimental Result: Long Cache vs 128-byte Cache

- Long cache: The highest speedup is 2.5X with 11 tasklets
- 128-byte cache: The highest speedup is 3.16X with 10 tasklets



MAX number of reads	Reads packet size (byte)	MAX number of tasklets
20	≈ 3 KB	11

(a) Performance of *D_kallisto* with long cache



MAX number of reads	Reads packet size (byte)	MAX number of tasklets
800	≈ 120 KB	24

(b) Performance of *D_kallisto* with 128-byte cache and mutex lock

WRAM is expensive, we have to design the data flow carefully during DPU-based programming.

Outline

- Introduction
- Background
- Motivation
- RNA-seq Quantification on DPU
- Design Tradeoffs and Evaluations
  - Large Hash Table
  - DPU Programming Issues
- **Conclusion**

Conclusion

- We implement D_kallisto to help communities more understand the DPU-based programming

- A series of experiments is built and conducted to characterize the tradeoff between the overall performance and the hardware/software constraints of DPUs

- Suggestions and consideration of the DPU programming are also presented in this work
  - Frequently data transfers will become an overhead
  - We should use WRAM with fine-grained design in DPU programming

Thanks for your attention

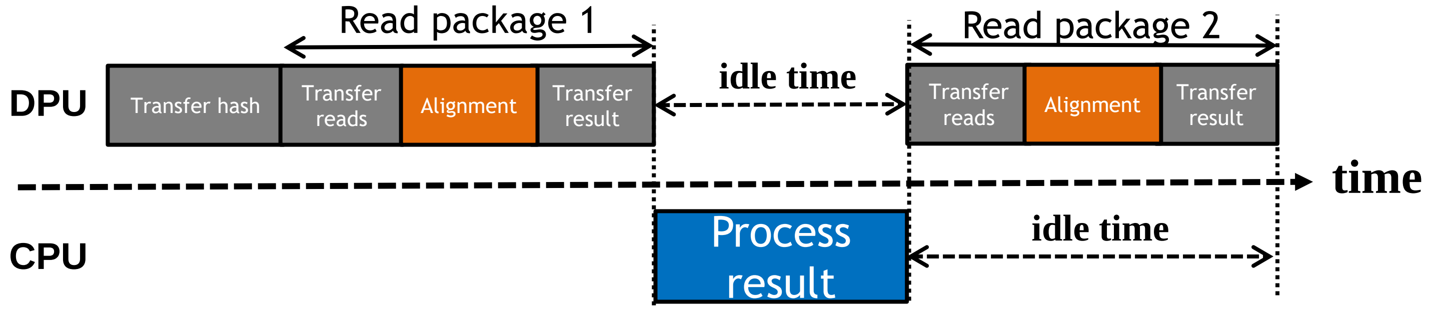
Question & Answer

Appendix

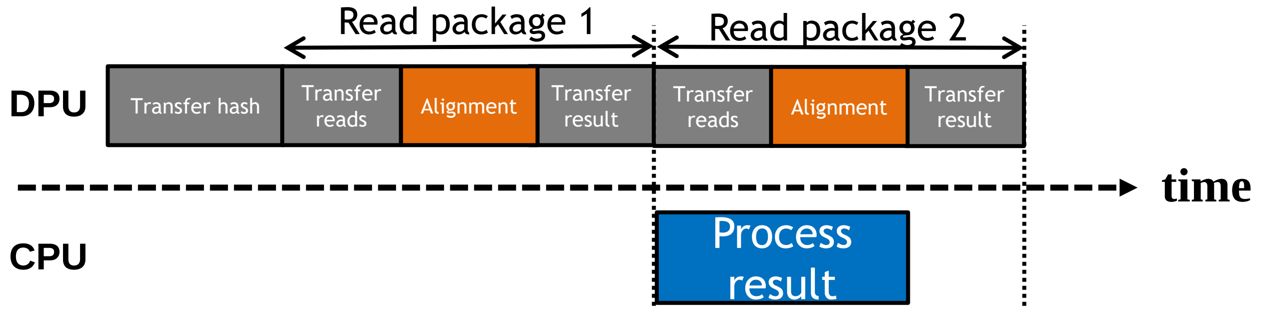
CPU & DPU Task Overlap

- Design overview : Enable concurrency between CPU and DPU.
  - Reduce idle time of CPU and DPU.

w/o communication thread and working thread



w/ communication thread and working thread



Read package 1
Read package 2
DPU
Transfer hash
Transfer reads
Alignment
Transfer result
Transfer reads
Alignment
Transfer result
time
CPU
Process result