Computer Architecture
Final Project
Part -1

Accelerate RNA Sequence Quantification on RISC-V Distributed Cluster on Near-DRAM Platform

Background

RNA Sequence Quantification: Indexing

Transcripts	K-mers (k = 3)
T0: ACATGGA	(ACA, CAT, ATG, TGG, GGA)
T1: CATA	(CAT, ATA)
T2: GGGATACAT	(GGG, GGA, GAT, ATA, TAC, ACA, CAT)

Step 1:

Transcriptome Indexing

Hash-map

kmer	Idxs	kmer	Idxs
ACA	T0, T2	**ATA**	T1, T2
CAT	T0, T1, T2	**GGG**	T2
ATG	T0	**GAT**	T2
TGG	T0	**TAC**	T2
GGA	T0, T2		

RNA Sequence Quantification: Quantification

Step 2:

Quantification

RNA Seq.		**K-mers (k = 3)**
R0: GGACT	→	(GGA, GAC, ACT)
R1: CATAC	→	(CAT, ATA, TAC)
R2: TACAT	→	(TAC, ACA, CAT)

2.1 Index Querying

kmer	Idxs	kmer	Idxs
ACA	T0, T2	ATA	T1, T2
CAT	T0, T1, T2	GGG	T2
ATG	T0	GAT	T2
TGG	T0	TAC	T2
GGA	T0, T2		

Hash-map

2.2 Max Transcript Identification

R0: MAX((T0, T2),(),())

R1: MAX((T0, T1, T2),(T1, T2),(T2))

R2: MAX((T2),(T0, T2),(T0, T1, T2))

R0: (T0, T2)

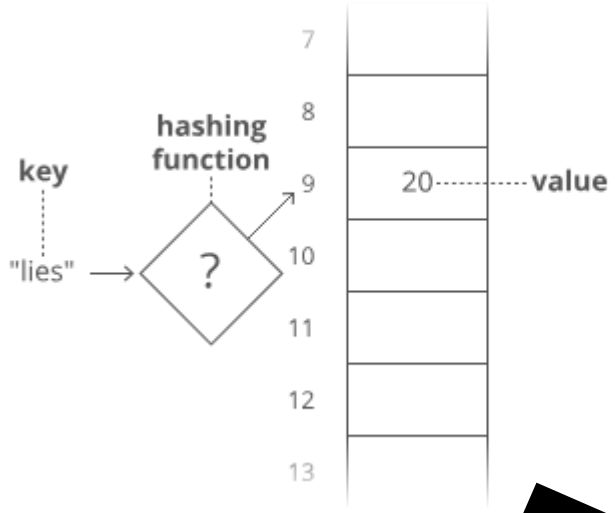
R1: (T2)

R2: (T2)

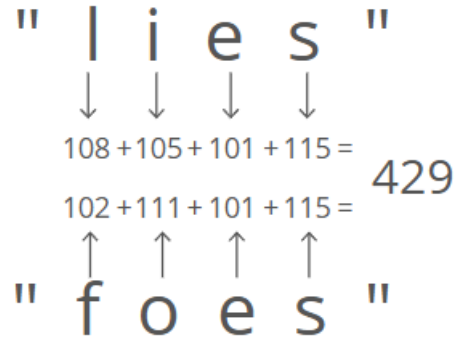
Final Result

T0: 1 **T1**: 0 **T2**: 3

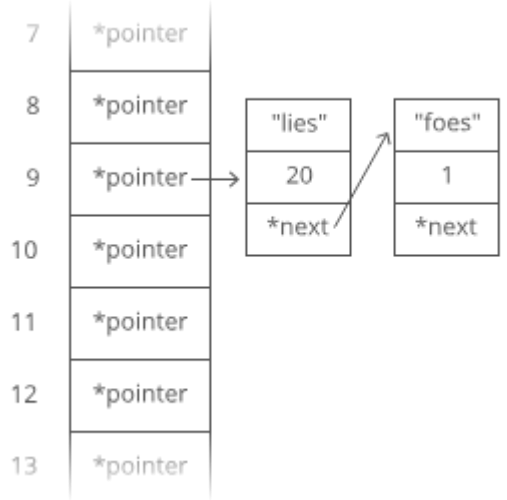
Hash Maps & Performance Issues



1. **Compute-intensive Hash-function**

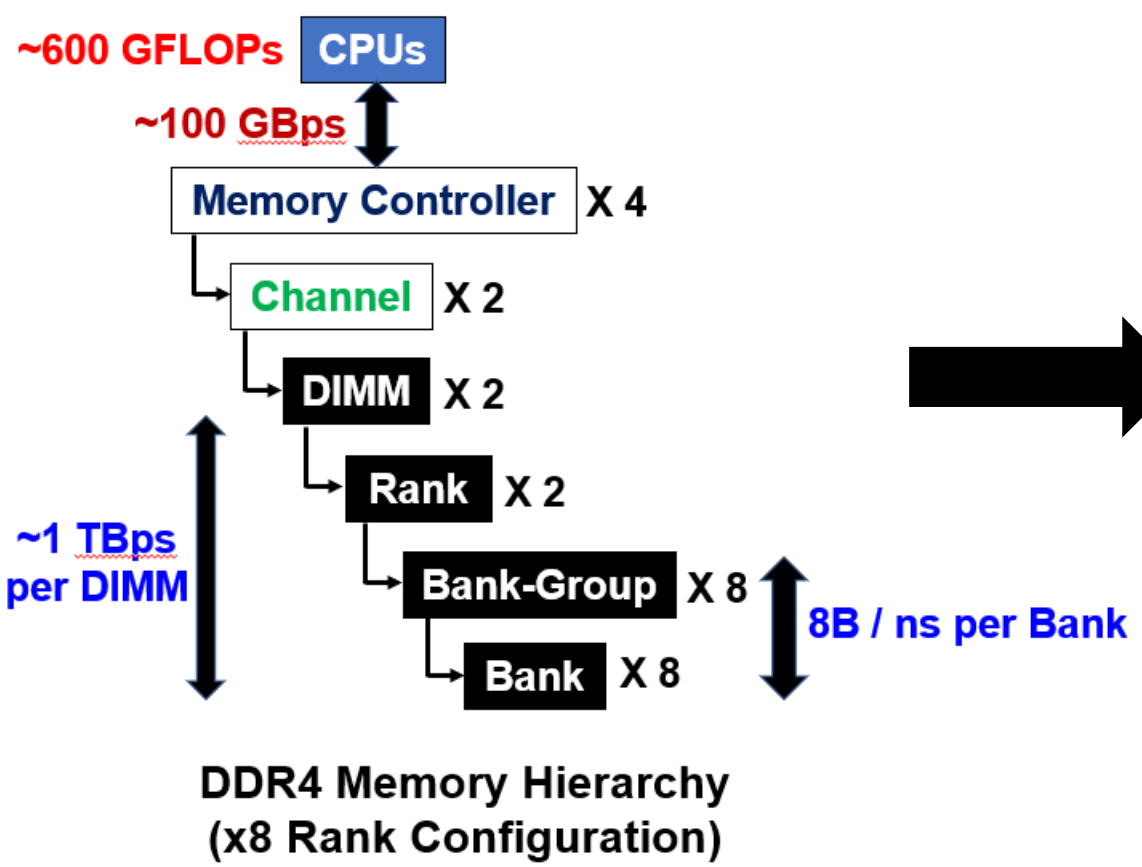


2. **Hash-collisions among keys**

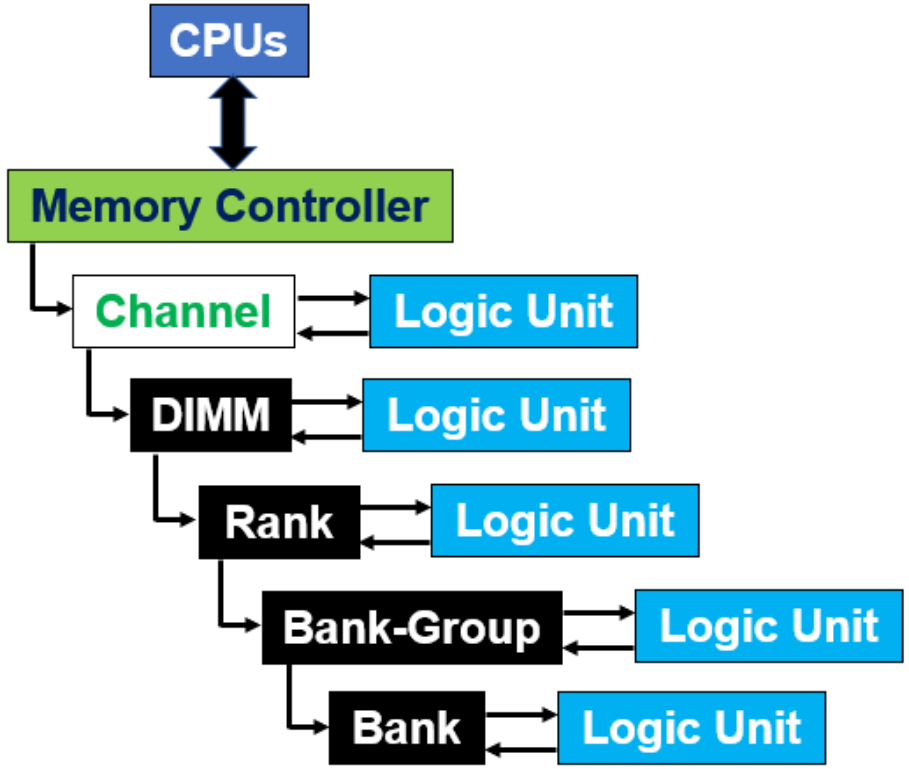


3. **Random data accesses**
   - 3.1. **DRAM**
   - 3.2. **Storage**
   - 3.3. **Network**

Near-Memory Computing: General DIMMs



DDR4 Memory Hierarchy
(x8 Rank Configuration)



Near-Memory Computing

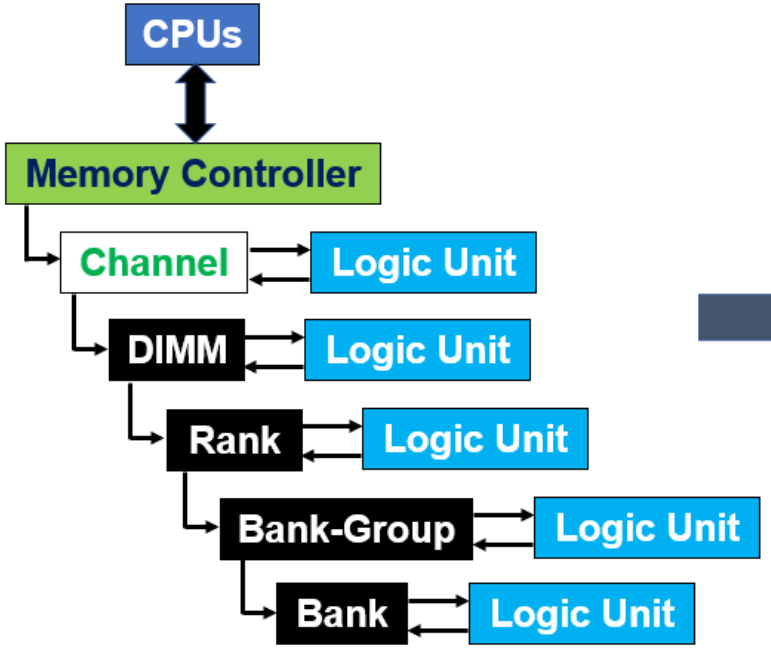
Pros:

1. Reduced latency
2. Reduced energy consumption
3. Massive parallelism

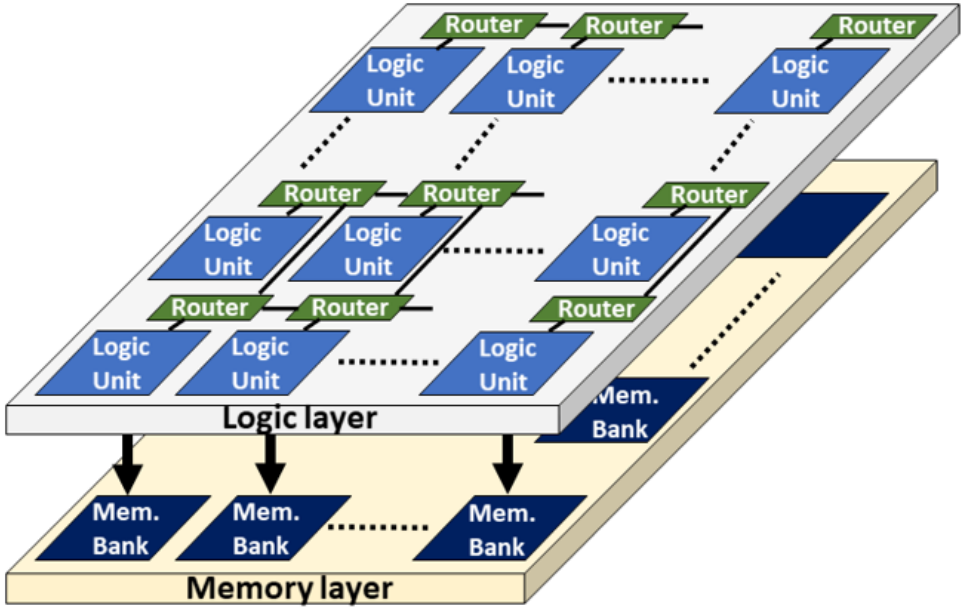
Cons:

1. Data access scope: Small DRAM capacity
2. Data share restrictions: Shared data buses
3. Logic Unit's Process technology constrained by DRAM

Near-Memory Computing: Stacked DRAM (SEDRAM)



Near-Memory Computing using general DIMMs



Near-Memory Computing using Stacked Memory

Cons:

1. Data access scope: Small DRAM capacity
2. Data share restrictions: Shared data buses
3. Logic Unit's Process technology constrained by DRAM

Pros:

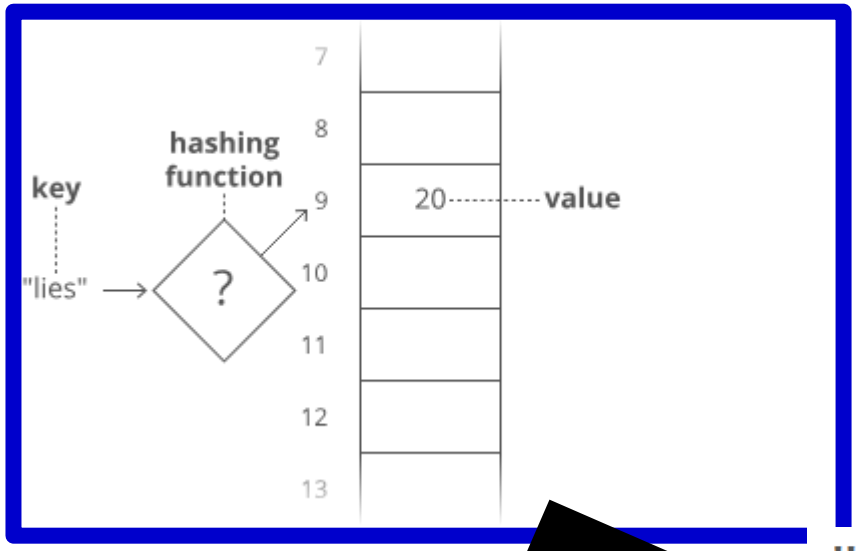
1. 1 Logic Unit == 1 DRAM Bank
2. NoC-facilitated data shares on Logic Layer
3. Independent Logic Unit Process technology

Problem Description

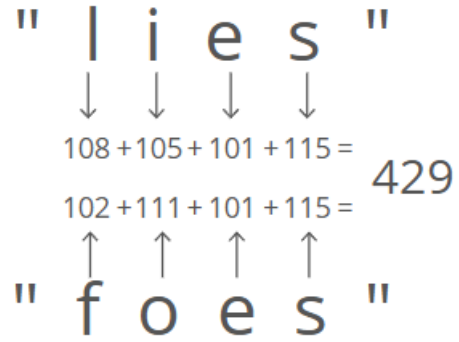
Problem Statement

- Objective: Improve RNA sequence Quantification performance on the Stacked DRAM (SEDRAM) platform
  1. Indexing: Hash-map generation (slide 3)
  2. Quantification: Query Hash-map and generate final results (slide 4)
- Methodology:
  - Improve Indexing and Quantification with better algorithms for:
    1. k-mer Hashing: Reduce computational latency
    2. Memory layout of Hash-map: Reduce memory-access latency
    3. Distribution of input transcript and RNA sequences (*query*) across Logic Units: Reduce NoC transfer latency
    4. Most-common Transcript identification for each read: Reduce computational latency

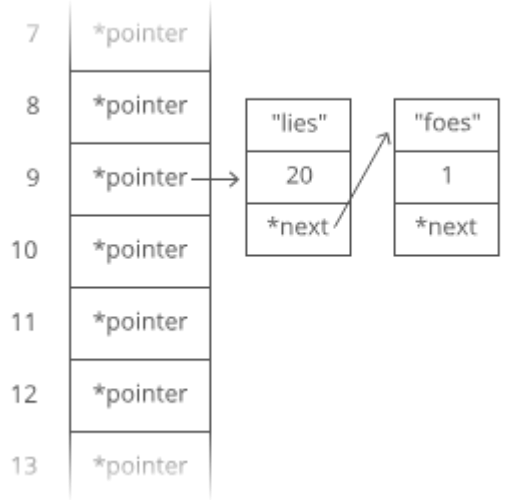
Hash Maps & Performance Issues



1. **Compute-intensive Hash-function**



2. **Hash-collisions among keys**

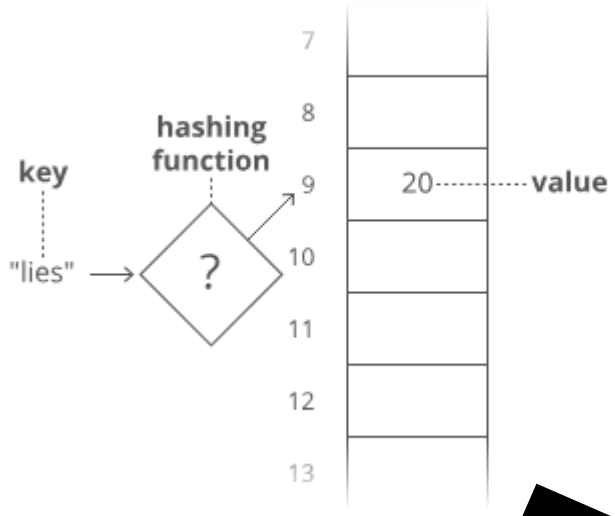


3. **Random data accesses**
   - 3.1. **DRAM**
   - 3.2. **Storage**
   - 3.3. **Network**

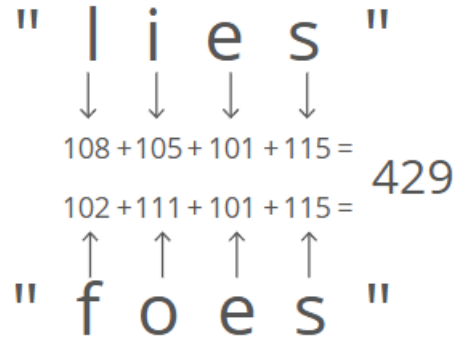
Problem Statement

- Objective: Improve RNA sequence Quantification performance on the Stacked DRAM (SEDRAM) platform
  1. Indexing: Hash-map generation (slide 3)
  2. Quantification: Query Hash-map and generate final results (slide 4)
- Methodology:
  - Improve Indexing and Quantification with better algorithms for:
    1. k-mer Hashing: Reduce computational latency
    2. Memory layout of Hash-map: Reduce memory-access latency
    3. Distribution of input transcript and RNA sequences (*query*) across Logic Units: Reduce NoC transfer latency
    4. Most-common Transcript identification for each read: Reduce computational latency

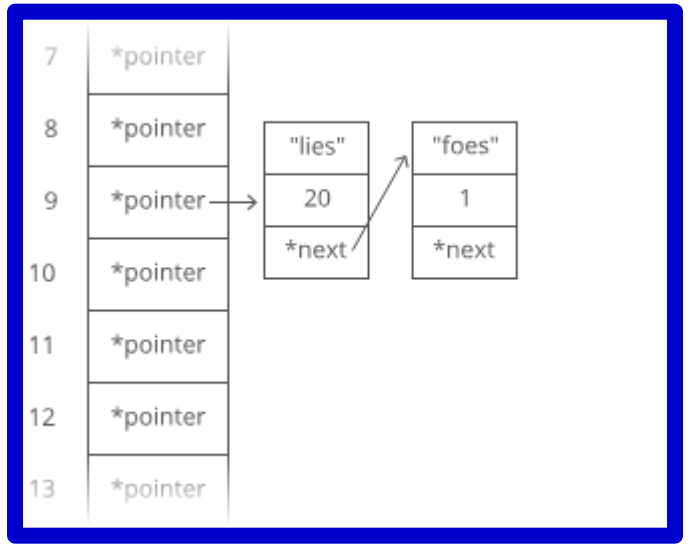
Hash Maps & Performance Issues



1. **Compute-intensive Hash-function**



2. **Hash-collisions among keys**

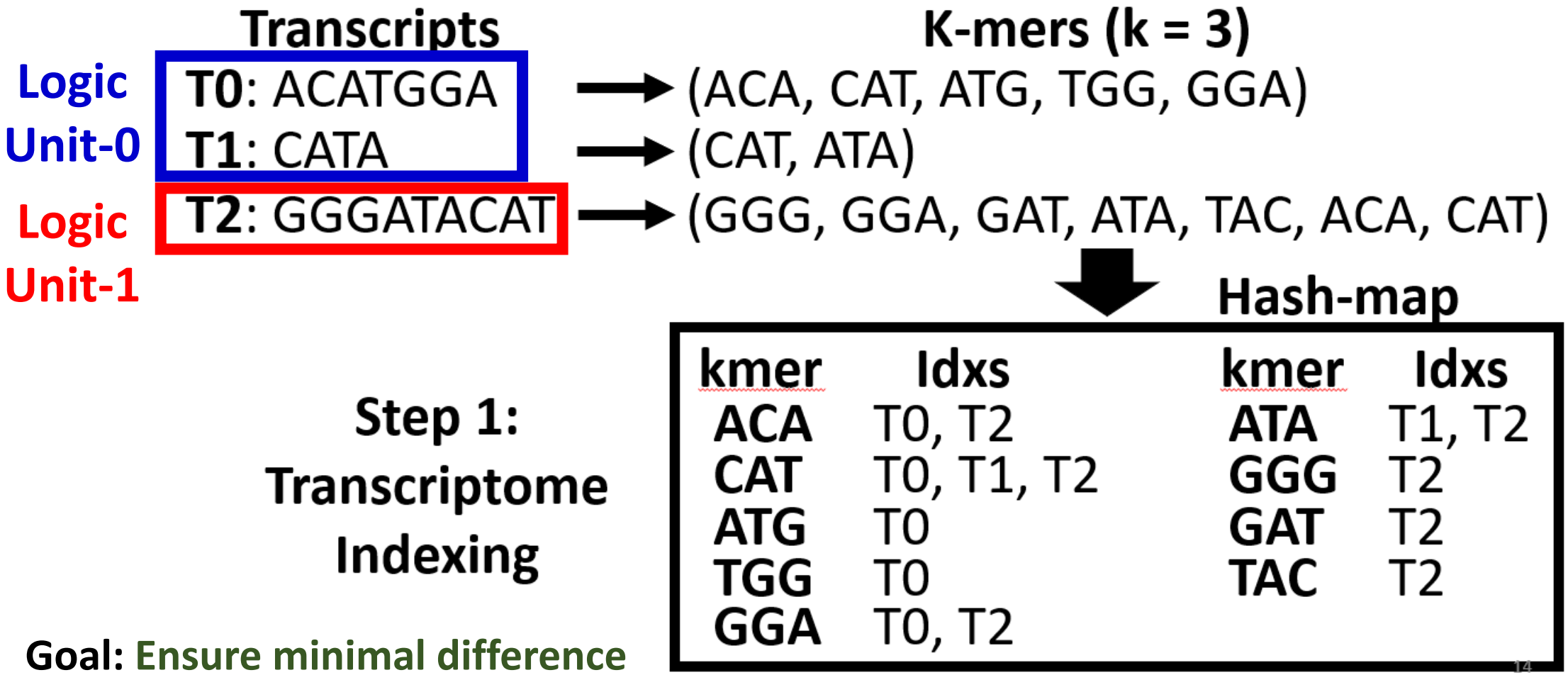


3. **Random data accesses**
   - 3.1. **DRAM**
   - 3.2. **Storage**
   - 3.3. **Network**

Problem Statement

- Objective: Improve RNA sequence Quantification performance on the Stacked DRAM (SEDRAM) platform
  1. Indexing: Hash-map generation (slide 3)
  2. Quantification: Query Hash-map and generate final results (slide 4)
- Methodology:
  - Improve Indexing and Quantification with better algorithms for:
    1. k-mer Hashing: Reduce computational latency
    2. Memory layout of Hash-map: Reduce memory-access latency
    3. Distribution of input transcript and RNA sequences (*query*) across Logic Units: Reduce NoC transfer latency
    4. Most-common Transcript identification for each read: Reduce computational latency

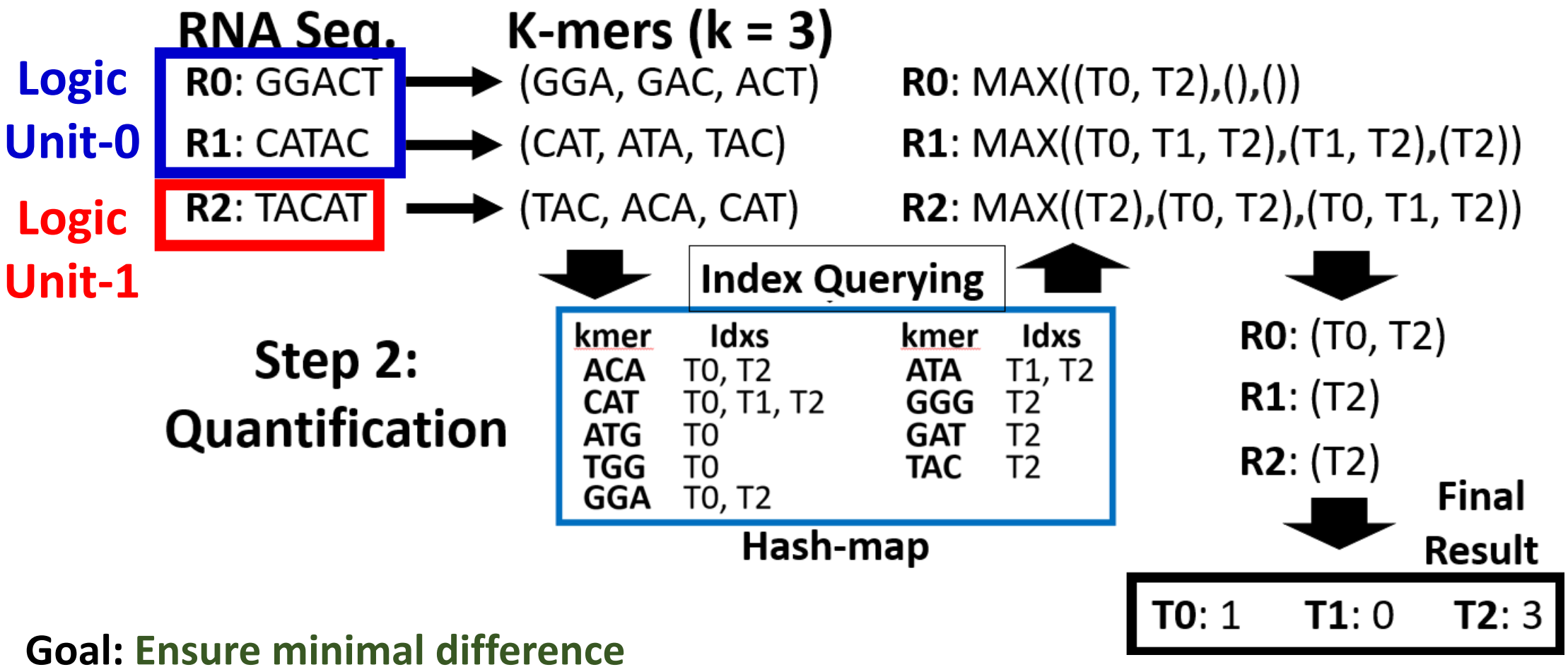
Distribution for Indexing



Step 1: Transcriptome Indexing

Goal: **Ensure minimal difference between Logic Units 0 and 1 Indexing execution latency**

Distribution for Quantification



Goal: **Ensure minimal difference between Logic Units 0 and 1 Quantification execution latency**

Problem Statement

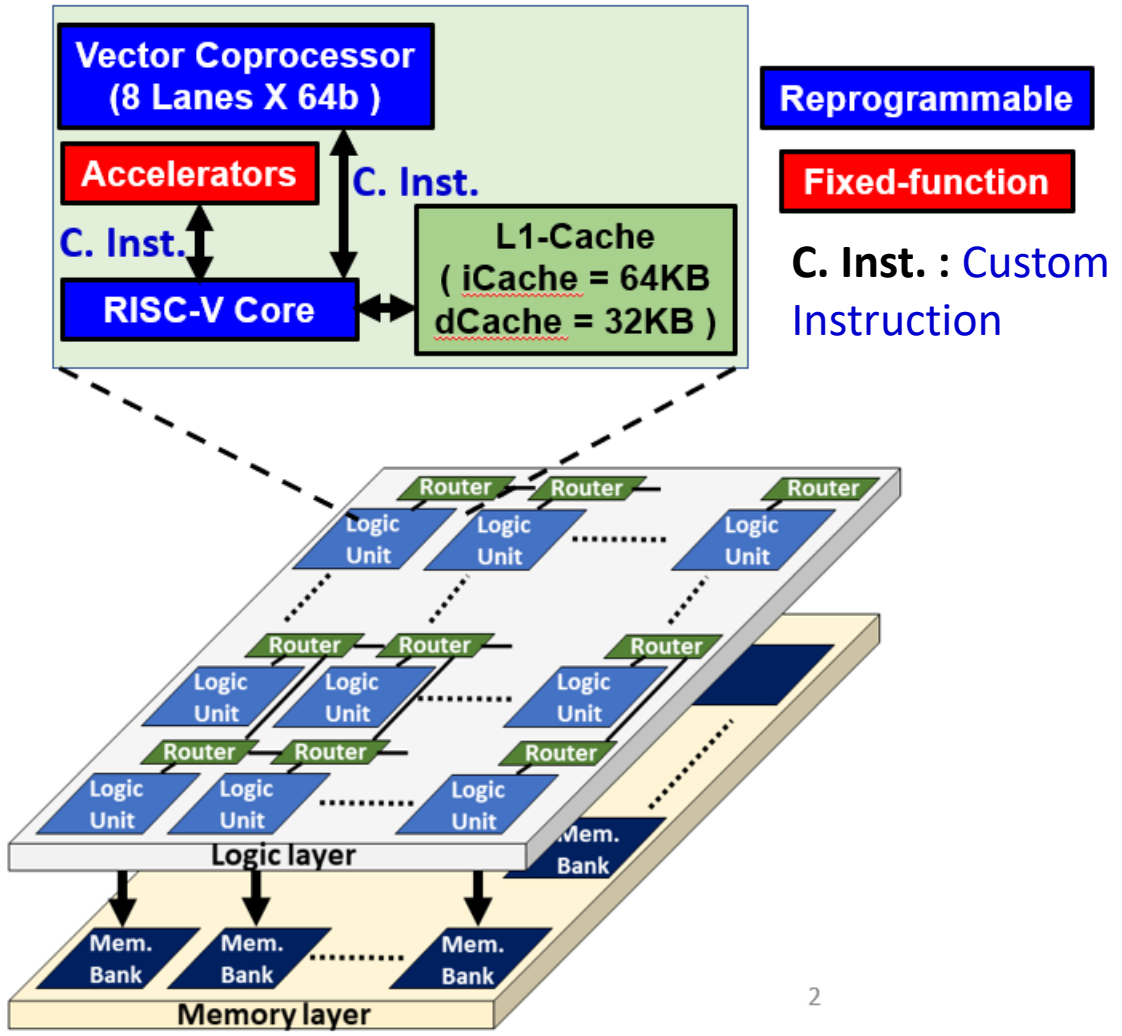
- Objective: Improve RNA sequence Quantification performance on the Stacked DRAM (SEDRAM) platform
  1. Indexing: Hash-map generation (slide 3)
  2. Quantification: Query Hash-map and generate final results (slide 4)
- Methodology:
  - Improve Indexing and Quantification with better algorithms for:
    1. k-mer Hashing: Reduce computational latency
    2. Memory layout of Hash-map: Reduce memory-access latency
    3. Distribution of input transcript and RNA sequences (*query*) across Logic Units: Reduce NoC transfer latency
    4. Most-common Transcript identification for each read: Reduce computational latency

Note: 1. You can interchange Transcripts and RNA Sequences for Indexing.

2. Assume kmer length to be variable while designing algorithms.

3. You can design distribution of Hash-Map instead of only the sequences to be indexed

Detailed Target Architecture



- **RISC-V Core:** In-Order @800 MHz
- **Vector Coprocessors:** 8 lanes X 64b, Controlled by RISC-V Custom Instructions
- **Accelerators:** Controlled by RISC-V Custom Instructions
- **DRAM Bank:** 256MB @4266 MBPS per pin

Program only RISC-V Cores using pre-existing instructions in FP-1

Custom Instructions for Vector Coprocessors & Accelerators and NoC Simulations to be implemented in FP-2

Simulation Setup & Evaluations

Simulation Tools & Methodology

- Tools (Pre-installed in Docker Image):
  - Gem5: RISC-V Core + Vector Coprocessors + Accelerator
  - Ramulator: DRAM Bank (Pre-integrated with Gem5)
  - Garnet: NoC traffic
- Methodology
  1. Design distribution scheme for Indexing and Quantification (slides 14-15) across the SEDRAM module with 64 Logic Units. Identify Logic Unit with largest workload.
  2. Simulate a single Logic Unit-DRAM Bank pair (identified above) for your proposed Indexing and Quantification algorithms.
  3. Estimate data-share needs and latency using provided NoC Bandwidth.
  4. Bonus:
     1. Analyze performance-resource tradeoff for using Logic Units compared to the Baseline on CPU Cores

Setup (Prerequisite: Install Docker)

- Steps:
  1. Pull Docker Image
     - sudo docker pull amansinhaatnycu/ca-fp:v2
  2. Launch Docker Container from Image
     - sudo docker run -ti --name ca-fp amansinhaatnycu/ca-fp:v2
  3. Change to Baseline CPU program directory, understand code, and run
     - cd /home/CA-FP1/baseline/ ; make run

Simulation of Code to be Evaluated

1. Change to the directory with code for task distribution, implement CPU C++ program, run and find largest workload for a single Logic Unit
   - cd /home/CA-FP1/evaluation/ ; vi distribute.cpp ; make make_distribute ; make run_distribute
2. Change to the directory with RISC-V code (single Logic Unit + DRAM Bank) to be evaluated, implement your Indexing and Quantification algorithm
   - cd /home/CA-FP1/evaluation/
   - vi index.c ; vi quantify.c
3. Compile Indexing and Quantification implementations
   - make make_index ; make make_quantify
4. Simulate Indexing and Quantification implementations
   - make run_index ; make run_quantify

Zip & Submit:
/home/CA-FP1/evaluation/

Instructions available in the .cpp, .c & .h files.
DO NOT MODIFY MARKED REGIONS

Evaluation Methodology & Scoring

- Total Score: 100
  1. Distribution Algorithm (distribute.cpp): 10
  2. Indexing Algorithm (index.c): 20
  3. Quantification Algorithm (quantify.c): 25
  4. Total Runtime (Ranking of Distribution + Indexing + Quantification): 30
  5. Report (Details of proposed algorithms 1-3 & their performance results): 15
- Bonus: 10
  - Report (Analyze performance-area* tradeoff for Logic Units compared to the Baseline execution on CPU Cores#)

***Assume Logic Unit Area from the paper: "Near-DRAM Accelerated Matrix Multiplications", MCSoC-2024.**

#To be provided on 04/30.

Docker Installation (Ubuntu Link)

- # Add Docker's official GPG key:

```
sudo apt-get update
sudo apt-get install ca-certificates curl
sudo install -m 0755 -d /etc/apt/keyrings
sudo curl -fsSL https://download.docker.com/linux/ubuntu/gpg -o /etc/apt/keyrings/docker.asc
sudo chmod a+r /etc/apt/keyrings/docker.asc
```

- # Add the repository to Apt sources:

```
echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.asc] https://download.docker.com/linux/ubuntu \
  $(. /etc/os-release && echo "${UBUNTU_CODENAME:-$VERSION_CODENAME}") stable" | \
  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
sudo apt-get update
```

- # Install latest version of Docker

```
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin
```

- # Verify installation

```
sudo docker run hello-world
```

You can check the above link for other Linux Distributions.
For Windows, use Ubuntu Subsystem and install on top of it.