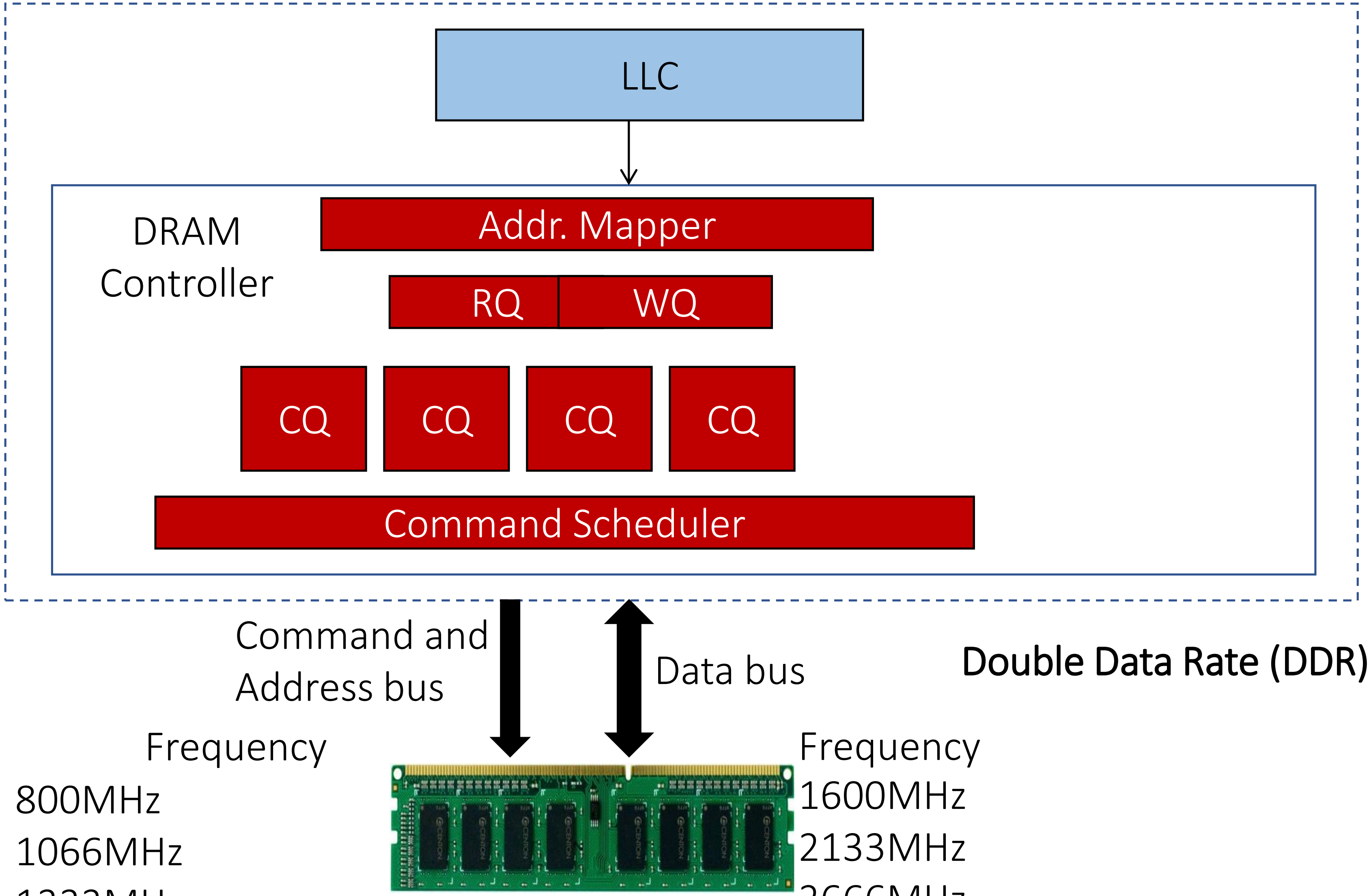
CS698Y: Modern Memory Systems
Lecture-17 (DRAM Controller)

Biswabandan Panda

biswap@cse.iitk.ac.in

https://www.cse.iitk.ac.in/users/biswap/CS698Y.html

An Overview



LLC

DRAM Controller

Addr. Mapper

RQ

WQ

CQ

CQ

CQ

CQ

Command Scheduler

Command and Address bus

Data bus

Double Data Rate (DDR)

Frequency

800MHz

1066MHz

1333MHz

Frequency

1600MHz

2133MHz

2666MHz

Reads vs Writes

Reads are critical to performance

Write Queue stores writes and the writes are serviced after # writes reach a threshold

Why?

The direction of the data bus changes from reads to writes. So ??

DRAM controller creates DRAM commands from based on the requests at read Q and write Q

DRAM Scheduling

Based on
Row-buffer locality,
Source of the request,
Loads/Stores
Load criticality

Satisfy all the timing constraints. Around 60

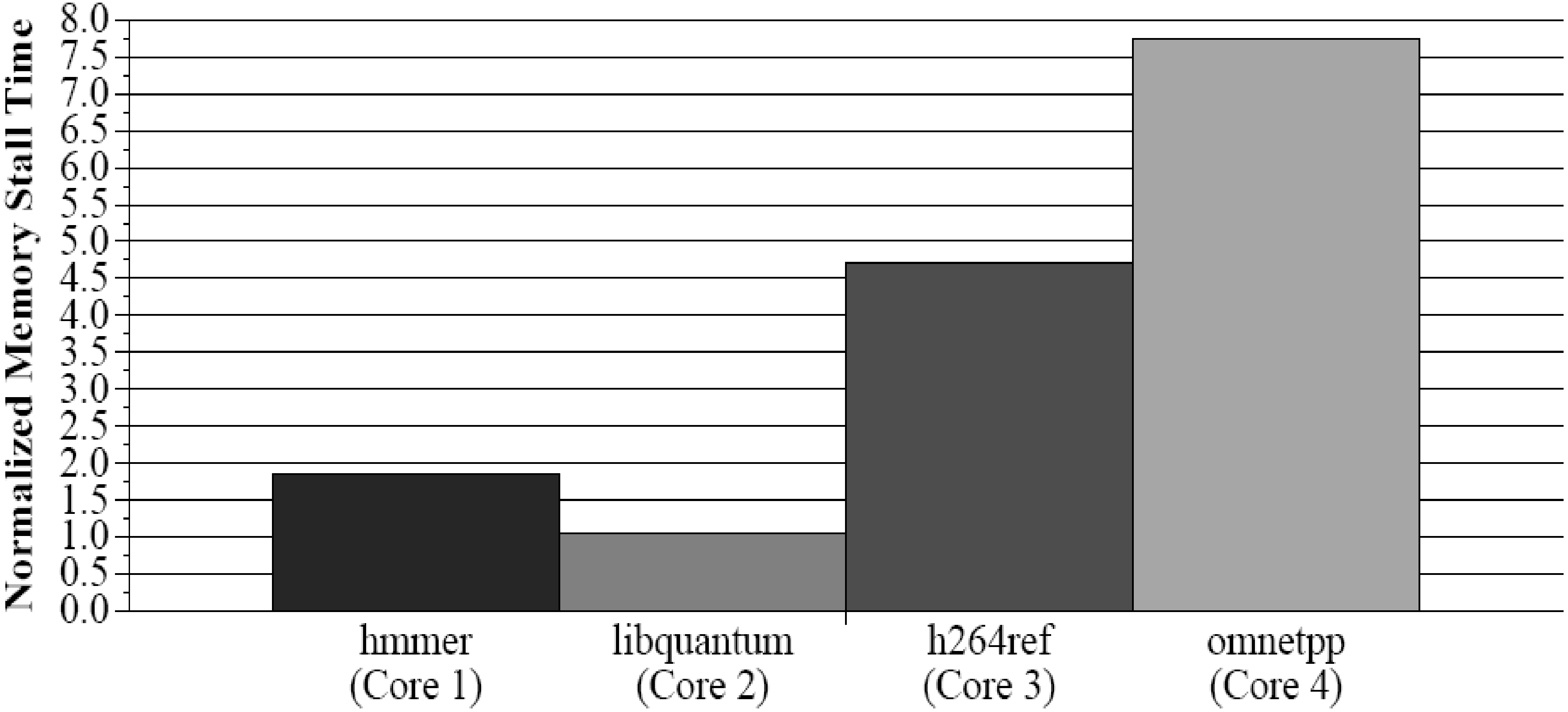
FCFS?

FR-FCFS [ISCA 00]

Prefers requests with Row hits (column-first) FR: First Ready

FR-FCFS for Multi-core Systems

Inter-core Conflicts between prefetch and demand requests



Usenix Security ‘07

Non-deterministic Slowdown

Memory Performance Hog

```
// initialize large arrays A,
B
     streaming
for (j=0; j<N; j++) {
     index = j*linesize;
     A[index] = B[index];
     …
}          STREAM
```

```
// initialize large arrays A,
B
          random
for (j=0; j<N; j++) {
     index = rand();
     A[index] = B[index];
     …
}          RANDOM
```

Sequential memory access

high row buffer locality (96% hit rate)

Memory Intensive

Random

Low row buffer locality (3% hit rate)

Memory Intensive

https://github.com/CMU-SAFARI/Cache-Memory-Hog

Let’s Revisit This

Option-I:

30 = (3 × 10) = 3 programming assignments

40 = (2 × 20) = Quiz 1.0 and Quiz 2.0 (Optional Quiz 1.1 and Quiz 2.1) = max (Quiz 1.x) + max (Quiz 2.x)

20 = (2 × 10) = 2 paper reviews

10 = Classroom and Piazza participation

Bonus points for finding typos in slides

Option-II:

30 = (3 × 10) = 3 programming assignments

20 = (1 × 20) = max (Quiz 1.0, Quiz 1.1)

30 = (1 × 30) = 1 research project (weekly meetings)

10 = (2 × 5) = 2 paper reviews

10 = Classroom and Piazza participation

Iterative Assessment

Choose your option wisely ☺

Let’s Discuss about the following (DRAM Management)

Bandwidth Management

Scaling the bandwidth wall [ISCA ‘09] Bandwidth Partitioning [SIGMETRICS ‘11]
Bandwidth allocation [PABST, HPCA ‘17]

Capacity Management

Compressed DRAM [LCP, MICRO 13] [MBZIP, TACO ‘17]

Power Management

DVFS at the Memory
[Memory DVFS, ICAC ‘11][Coscale, MICRO ‘12][MultiScale, ISLPED ‘12]

Metrics of Interest (Let’s spend some quality time)

Application *i* running on an N-core system

Throughput = ∑ IPC (i)

Individual Slowdown (i) = CPI-together (i) / CPI-alone (i)

Weighted Speedup = ∑ (IPC-together(i) / IPC-alone (i))

Harmonic Mean of Speedups = N/∑ (IPC-alone(i)/IPC-together (i))

Unfairness =
Max-Slowdown/Min-Slowdown =
max(Individual slowdowns)/min(individual slowdowns)

STFM [MICRO 07]

- During each time interval, for each thread, DRAM controller
  - Tracks T_{shared}
  - Estimates T_{alone}

- At the beginning of a scheduling cycle, DRAM controller
  - Computes Slowdown = T_{shared}/T_{alone} for each thread with an outstanding legal request
  - Computes unfairness = MAX Slowdown / MIN Slowdown

- If unfairness < α
  - Use DRAM throughput oriented baseline scheduling policy
    - (1) row-hit first
    - (2) oldest-first

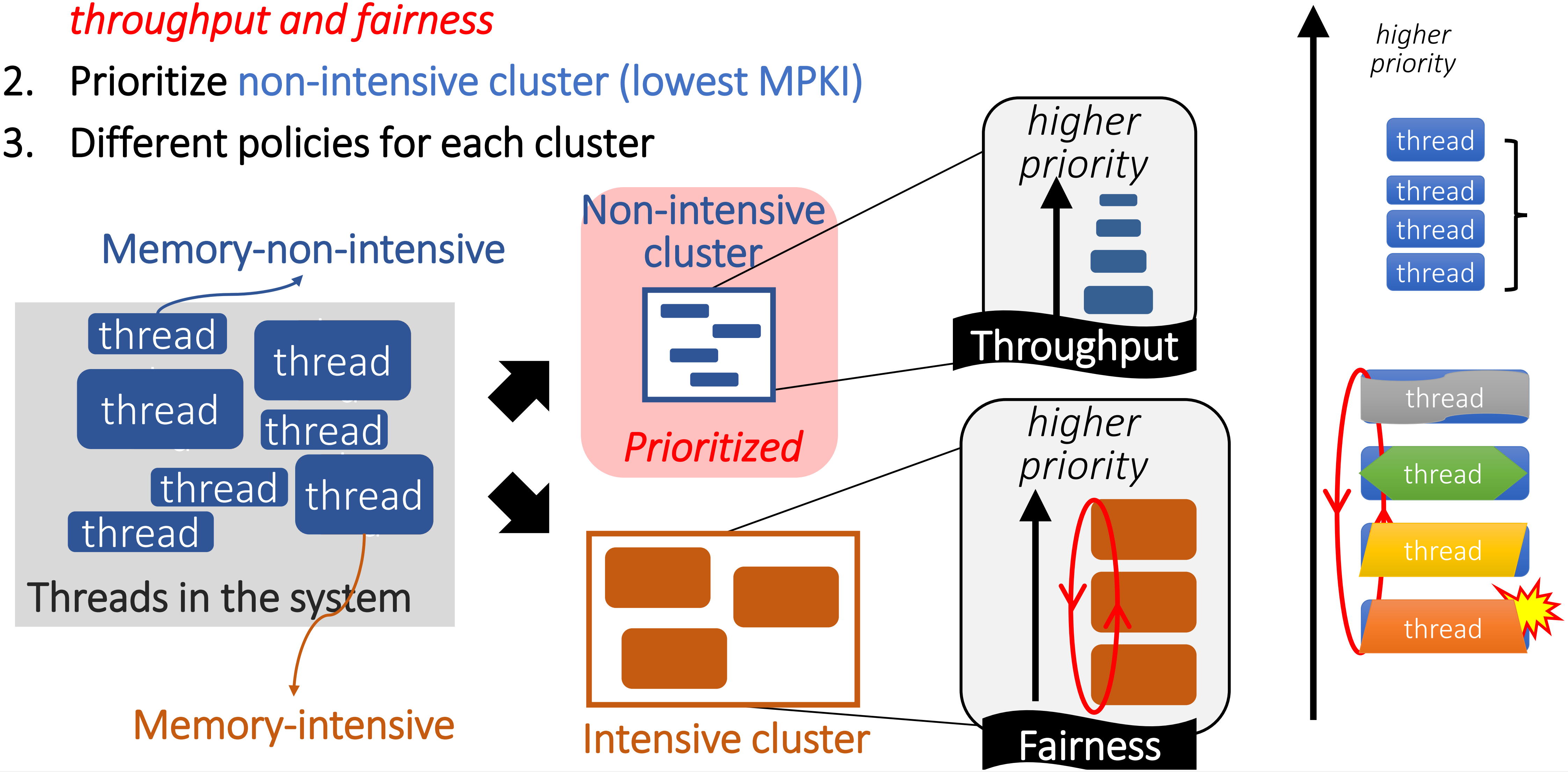
STFM [MICRO 07]

- If unfairness ≥ α
  - Use fairness-oriented scheduling policy
    - (1) requests from thread with MAX Slowdown first
    - (2) row-hit first
    - (3) oldest-first

- Maximizes DRAM throughput if it cannot improve fairness
- Does NOT waste useful bandwidth to improve fairness
  - If a request does not interfere with any other, it is scheduled

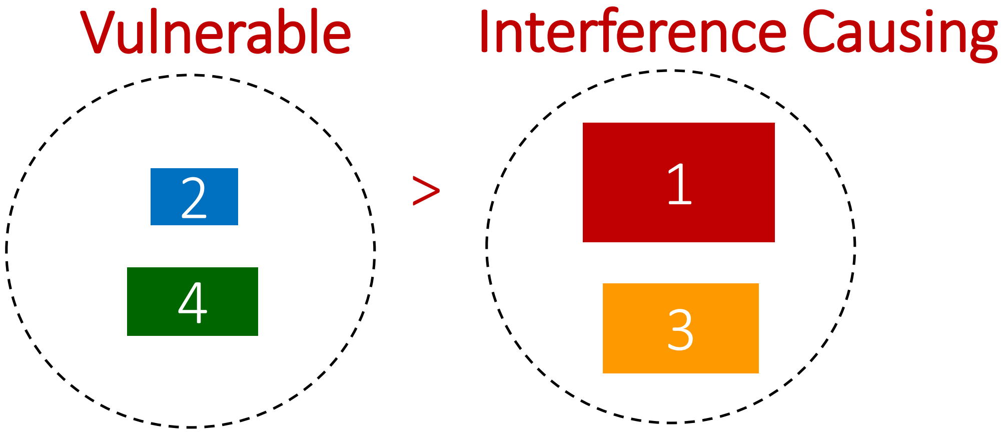
TCM – Thread Cluster memory Scheduling [MICRO 10]

1. Group threads into two *clusters to balance throughput and fairness*
2. Prioritize non-intensive cluster (lowest MPKI)
3. Different policies for each cluster



BLISS [ICCD 14]

Group instead of rank as ranking adds complexity



Basic Idea:

- *Group* applications with a large number of consecutive requests as *interference-causing* → *Blacklisting*
- *Deprioritize* blacklisted applications
- *Clear* blacklist periodically (1000s of cycles)