Self-Optimzing Memory Controllers: A Reinforcement Learning Approach

Engin Ipek[1,2] Onur Mutlu[2] José F. Martinez[1] Rich Carauana[1]

[1] Cornell University, Ithaca, NY 14850 USA
[2] Microsoft Research, Redmond, WA 98052 USA

International Symposium on Computer Architecture (ISCA) 2008

Presented by Valery Fischer

Summary

Problem & Goal

Key Ideas

Novelty

Mechanisms & Implementation

Results & Evaluation

Main Takeaways

Critique and Discussion

Strengths

Weaknesses

Thoughts

Discussion

Summary

Executive Summary

Executive Summary

Motivation: Efficiently utilizing off-chip bandwidth is critical in the design of Chip Multiprocessors (CMPs)

Executive Summary

Motivation: Efficiently utilizing off-chip bandwidth is critical in the design of Chip Multiprocessors (CMPs)

Problem: Conventional memory controllers deliver relatively low performance and off-chip bandwidth presents an impediment to CMP scalability, also memory controllers are difficult to optimize with a fixed, rigid policy.

Executive Summary

Motivation: Efficiently utilizing off-chip bandwidth is critical in the design of Chip Multiprocessors (CMPs)

Problem: Conventional memory controllers deliver relatively low performance and off-chip bandwidth presents an impediment to CMP scalability, also memory controllers are difficult to optimize with a fixed, rigid policy.

Goal: Improve off-chip bandwidth by utilizing a dynamic, self-optimizing memory controller

Executive Summary

Motivation: Efficiently utilizing off-chip bandwidth is critical in the design of Chip Multiprocessors (CMPs)

Problem: Conventional memory controllers deliver relatively low performance and off-chip bandwidth presents an impediment to CMP scalability, also memory controllers are difficult to optimize with a fixed, rigid policy.

Goal: Improve off-chip bandwidth by utilizing a dynamic, self-optimizing memory controller

Key Ideas: Transforming the memory controller to a reinforcement learning (RL) agent, which finds the best scheduling policy for long-term performance

Executive Summary

Motivation: Efficiently utilizing off-chip bandwidth is critical in the design of Chip Multiprocessors (CMPs)

Problem: Conventional memory controllers deliver relatively low performance and off-chip bandwidth presents an impediment to CMP scalability, also memory controllers are difficult to optimize with a fixed, rigid policy.

Goal: Improve off-chip bandwidth by utilizing a dynamic, self-optimizing memory controller

Key Ideas: Transforming the memory controller to a reinforcement learning (RL) agent, which finds the best scheduling policy for long-term performance

Evaluation: Test the new memory controller in different environments against the state-of-the-art best average access scheduling policy FR-FCFS

Executive Summary

Motivation: Efficiently utilizing off-chip bandwidth is critical in the design of Chip Multiprocessors (CMPs)

Problem: Conventional memory controllers deliver relatively low performance and off-chip bandwidth presents an impediment to CMP scalability, also memory controllers are difficult to optimize with a fixed, rigid policy.

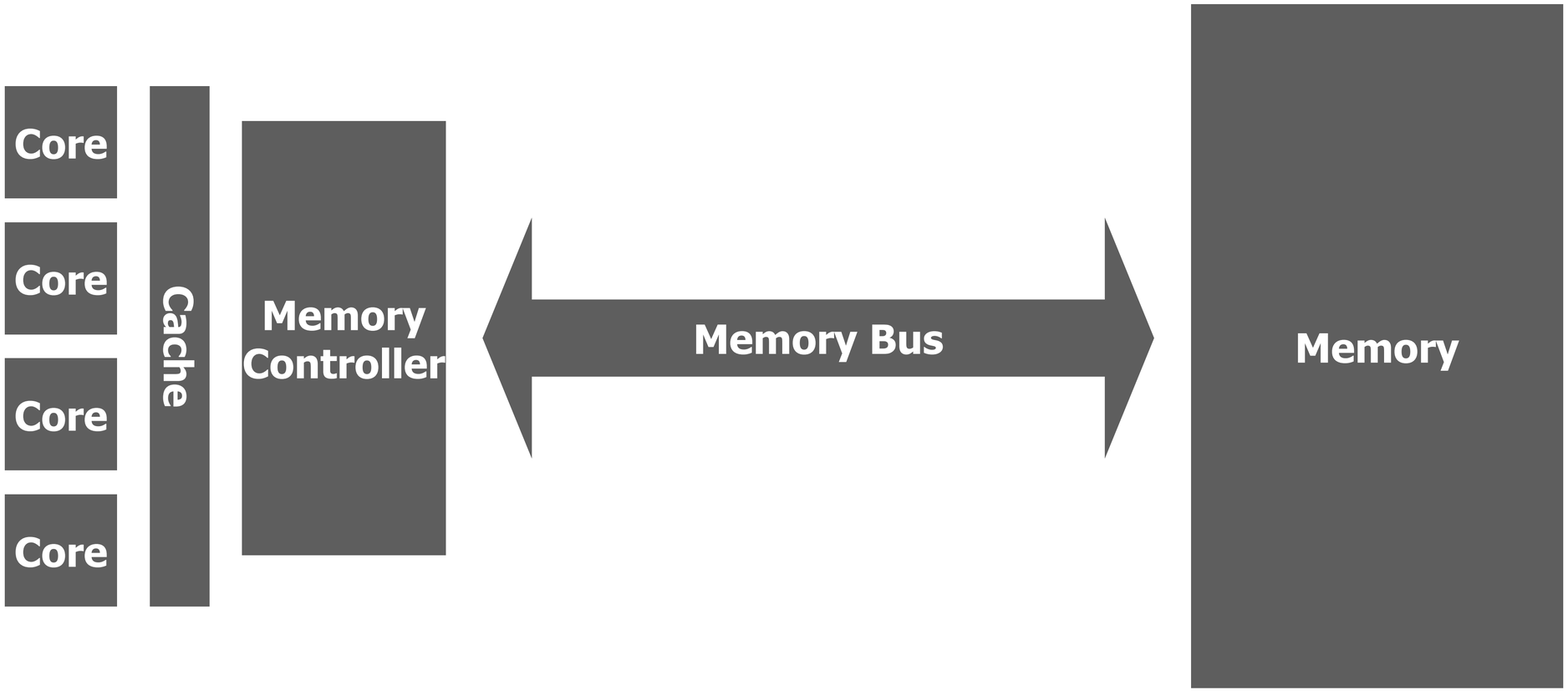
Goal: Improve off-chip bandwidth by utilizing a dynamic, self-optimizing memory controller

Key Ideas: Transforming the memory controller to a reinforcement learning (RL) agent, which finds the best scheduling policy for long-term performance

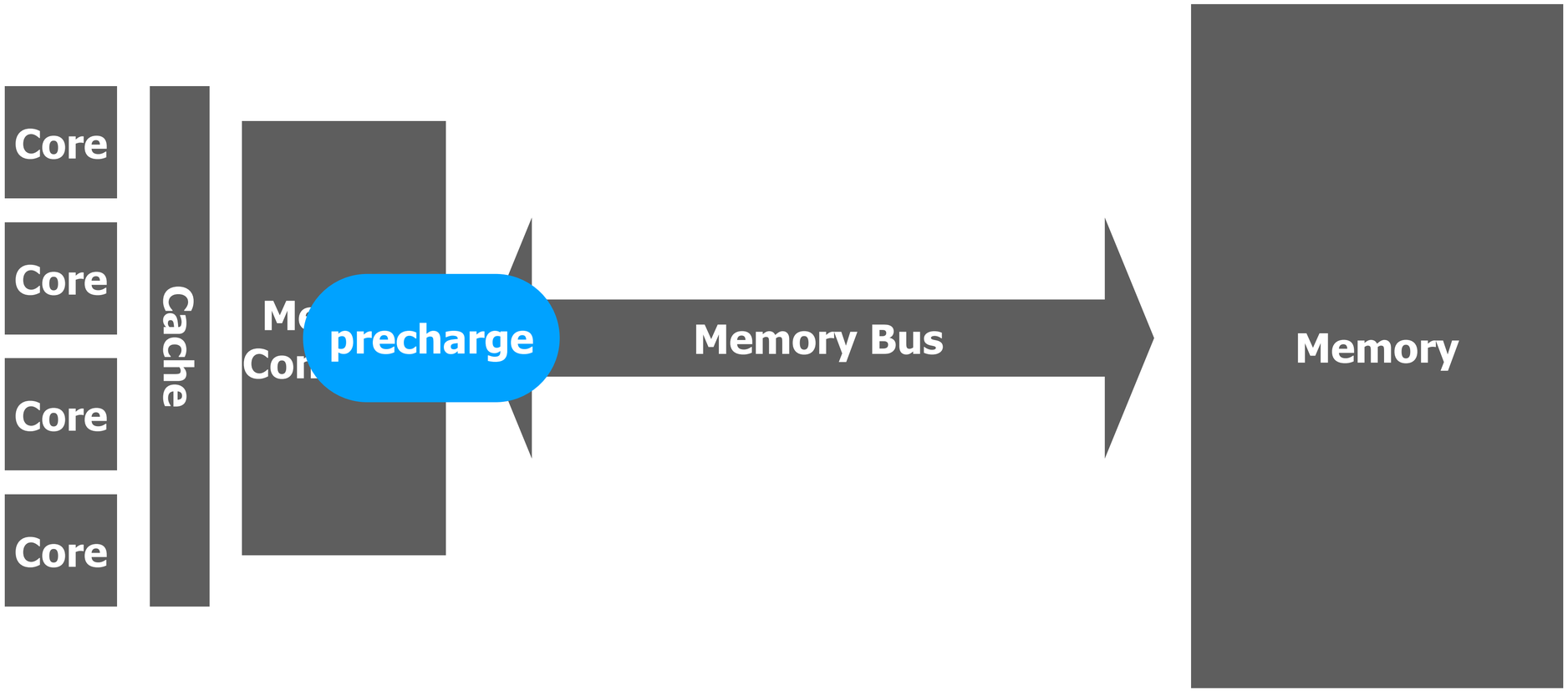
Evaluation: Test the new memory controller in different environments against the state-of-the-art best average access scheduling policy FR-FCFS

Results: An RL-based memory controller improves performance of parallel applications on a 4-core CMP by 19% on average and DRAM bandwidth utilization by 22% compared to FR-FCFS

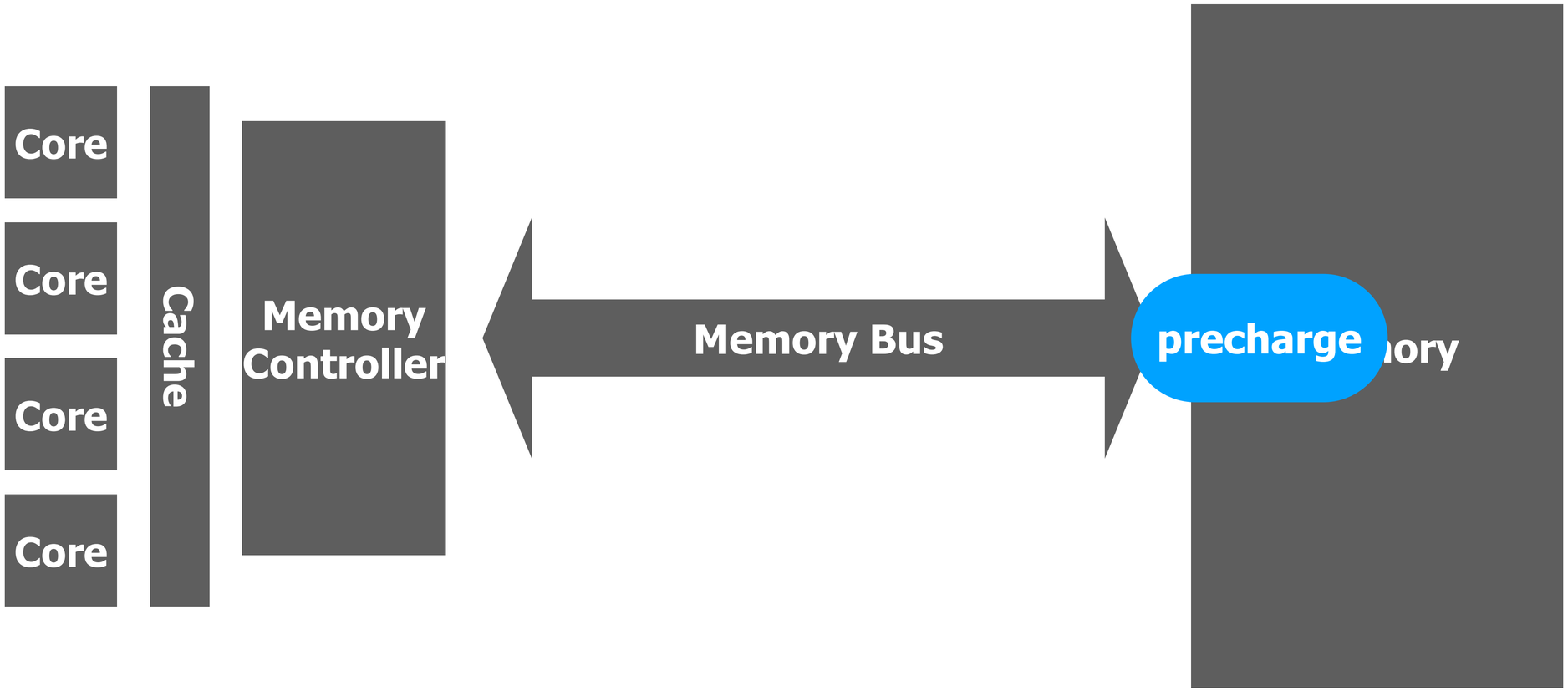
Memory Controllers



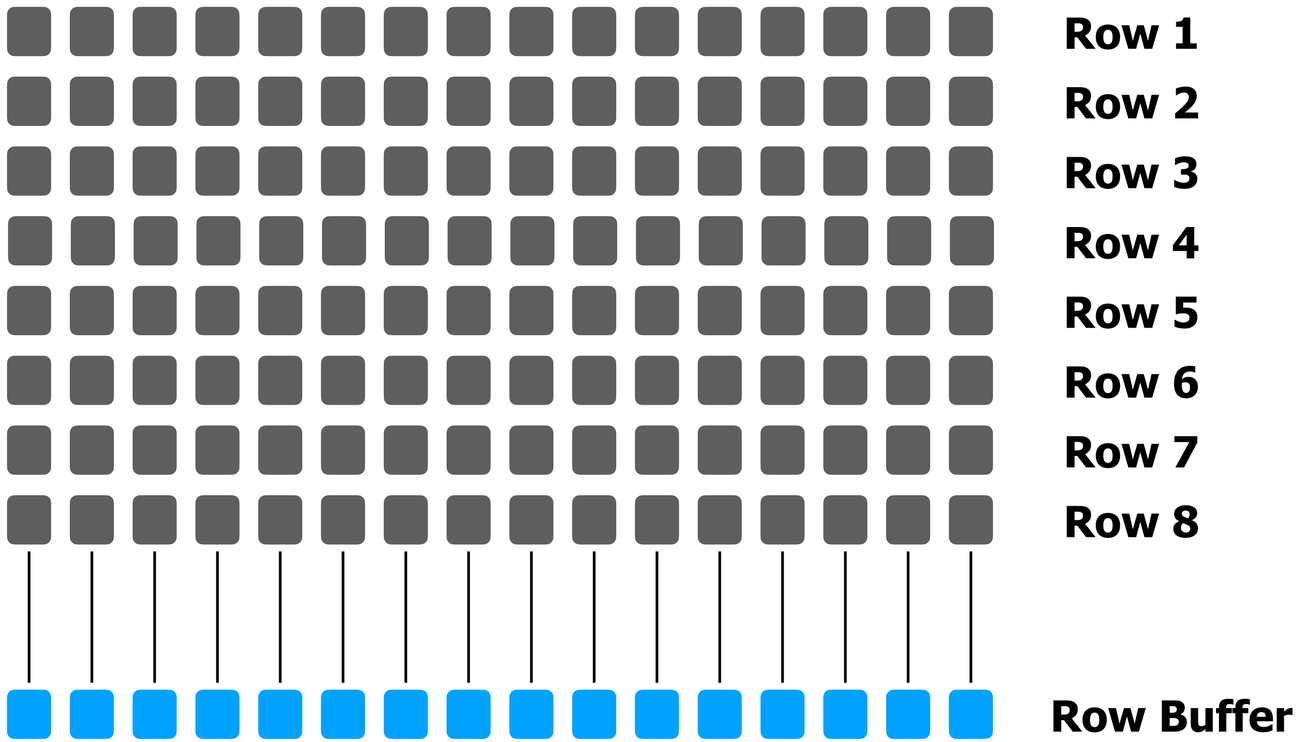
Memory Controllers



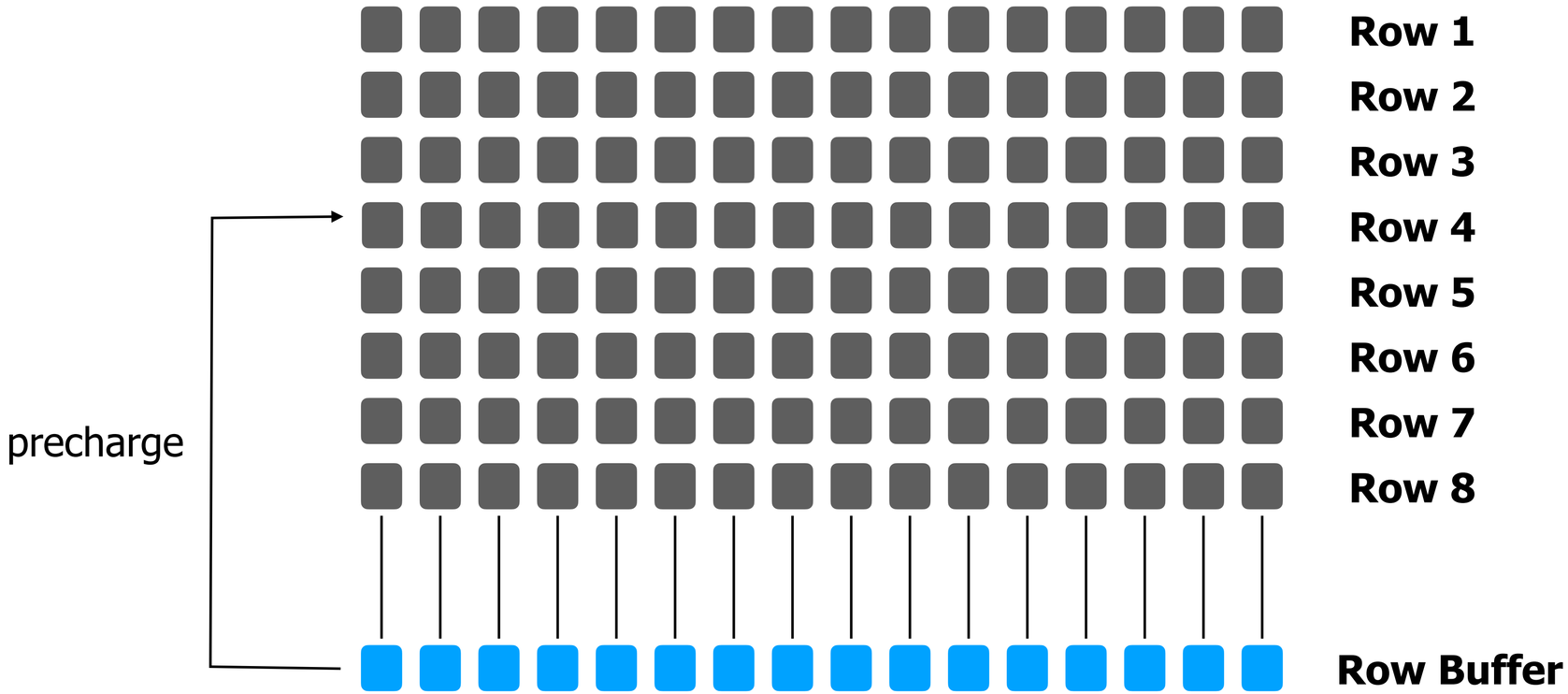
Memory Controllers



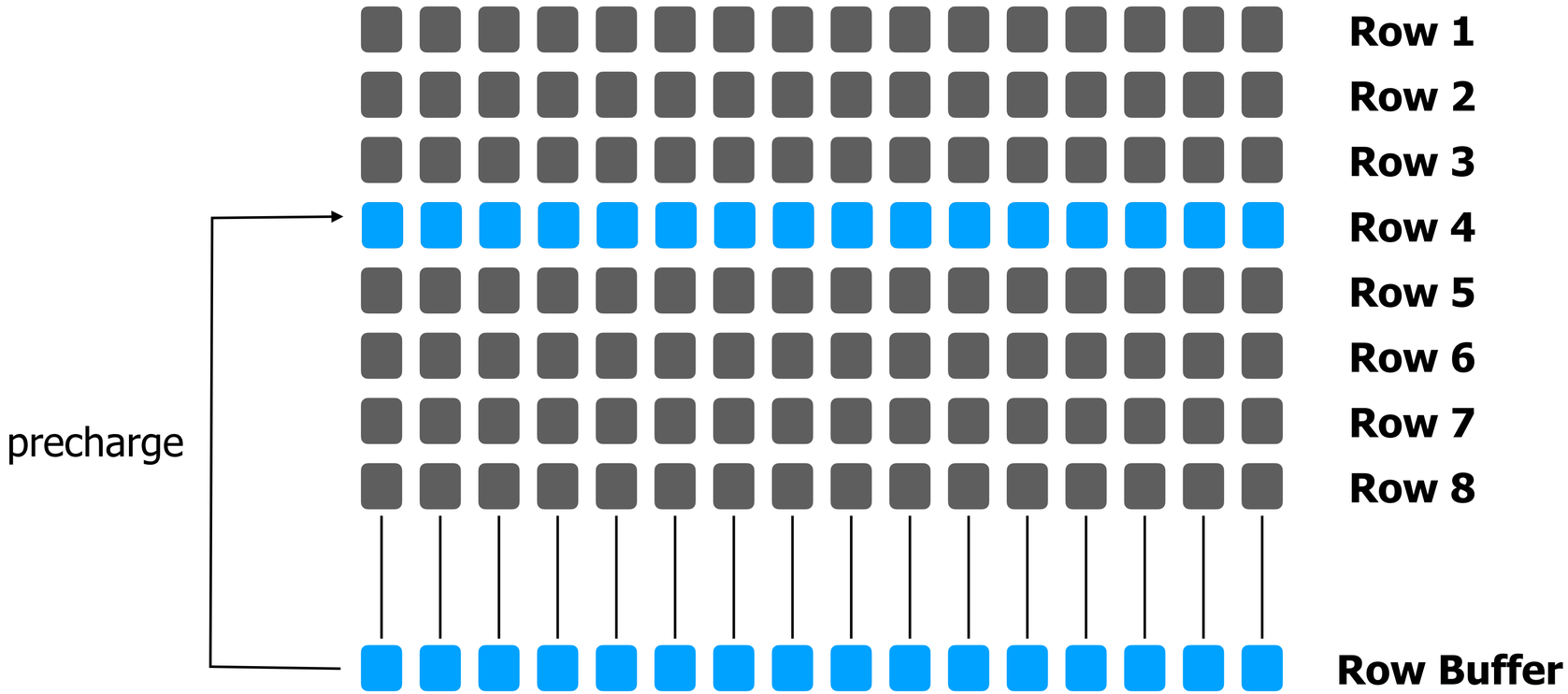
DRAM Bank (Precharge)



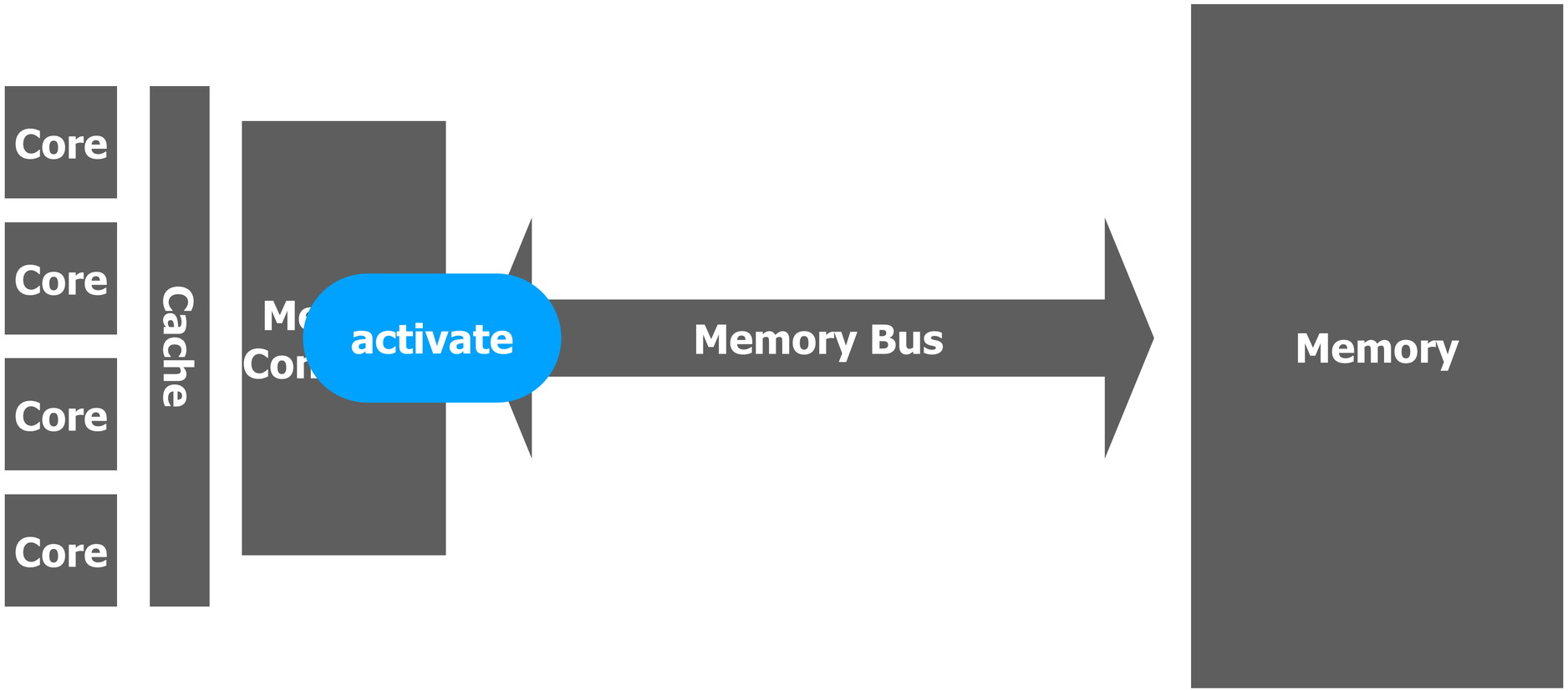
DRAM Bank (Precharge)



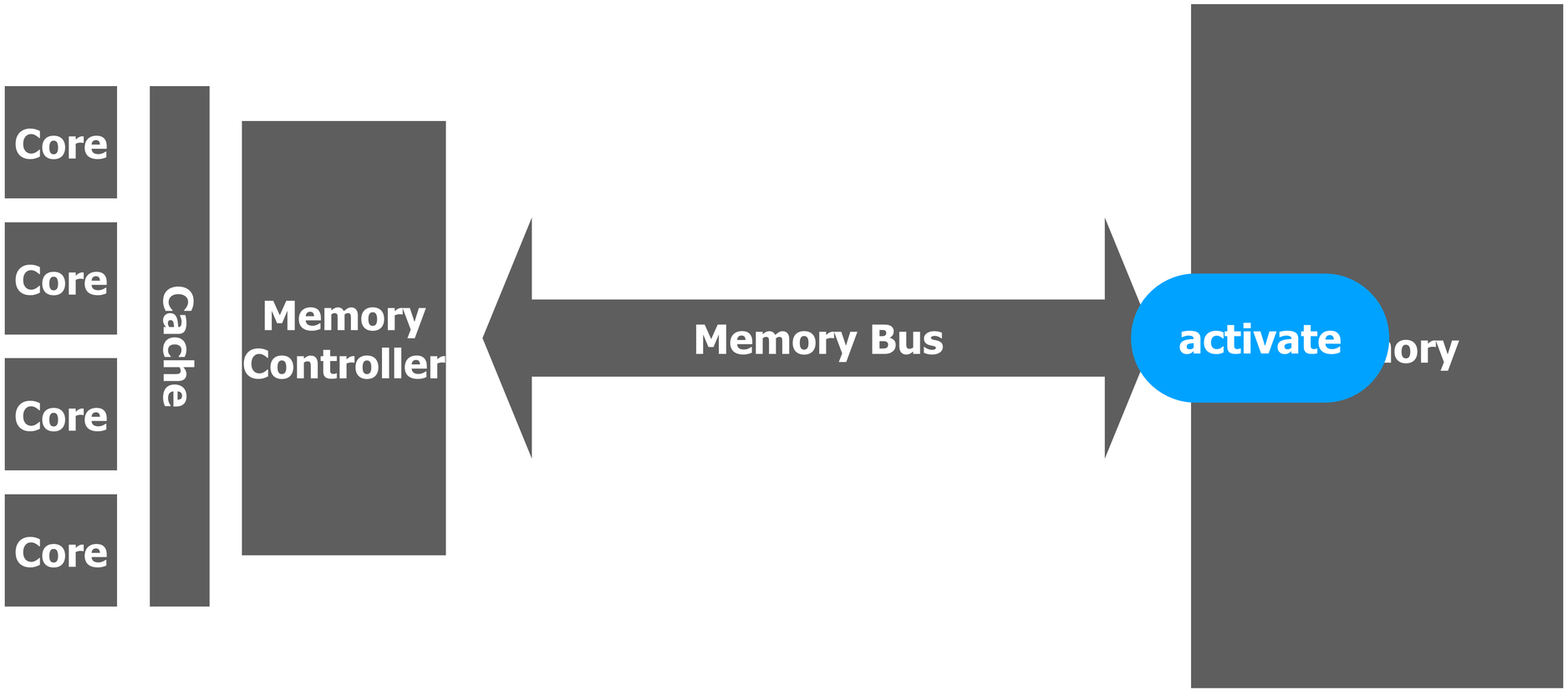
DRAM Bank (Precharge)



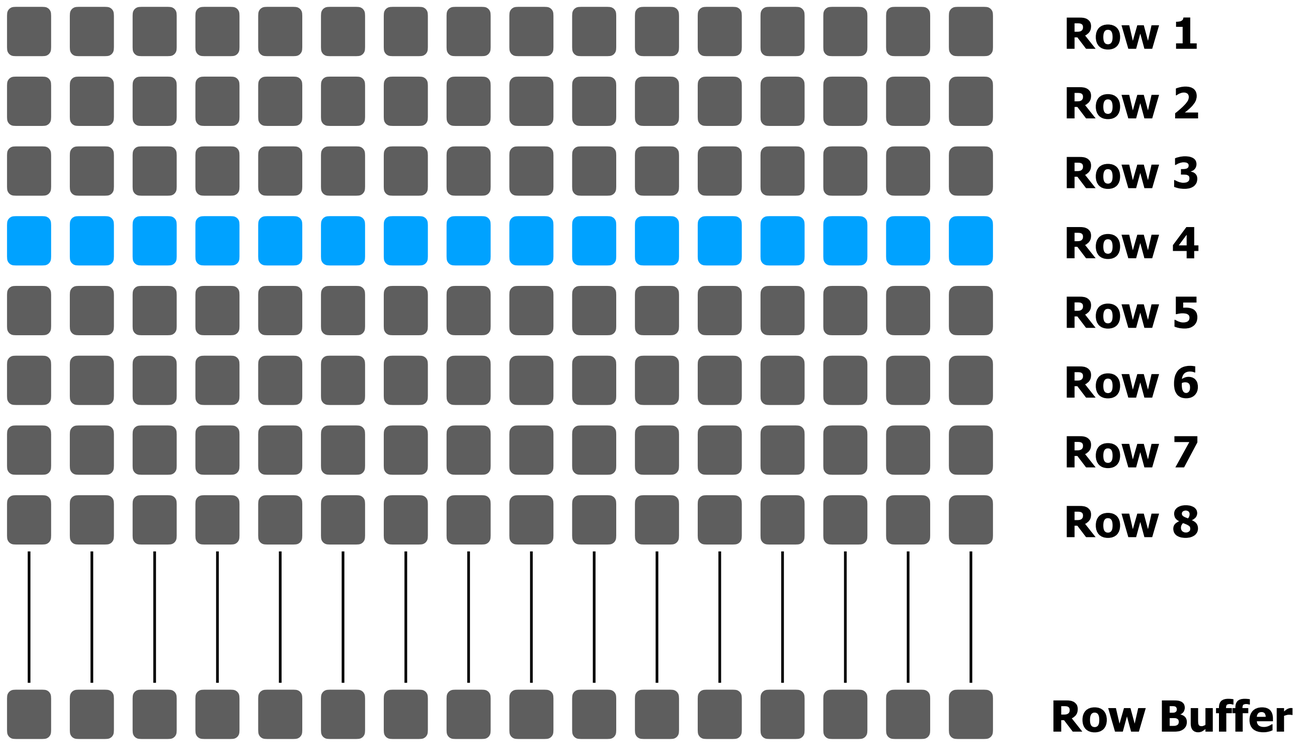
Memory Controllers



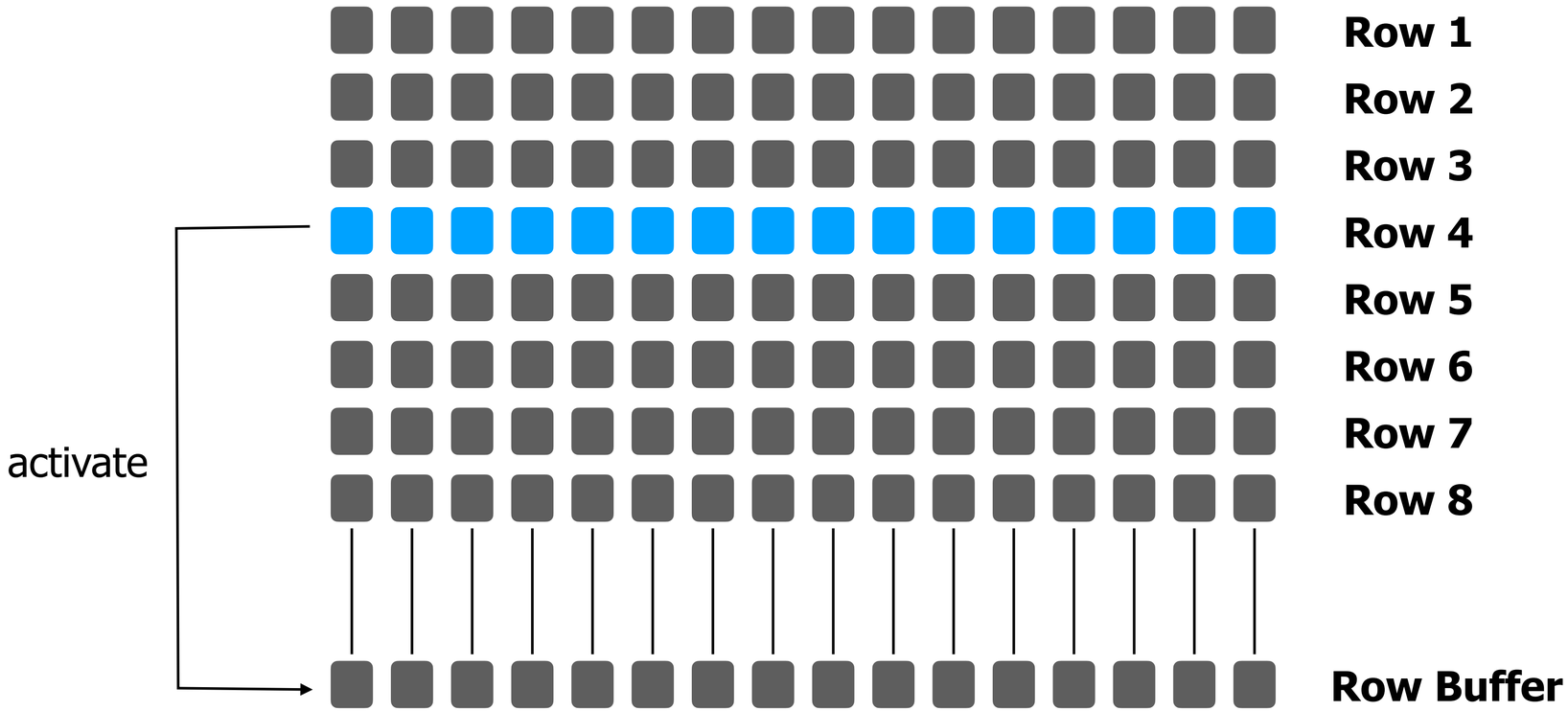
Memory Controllers



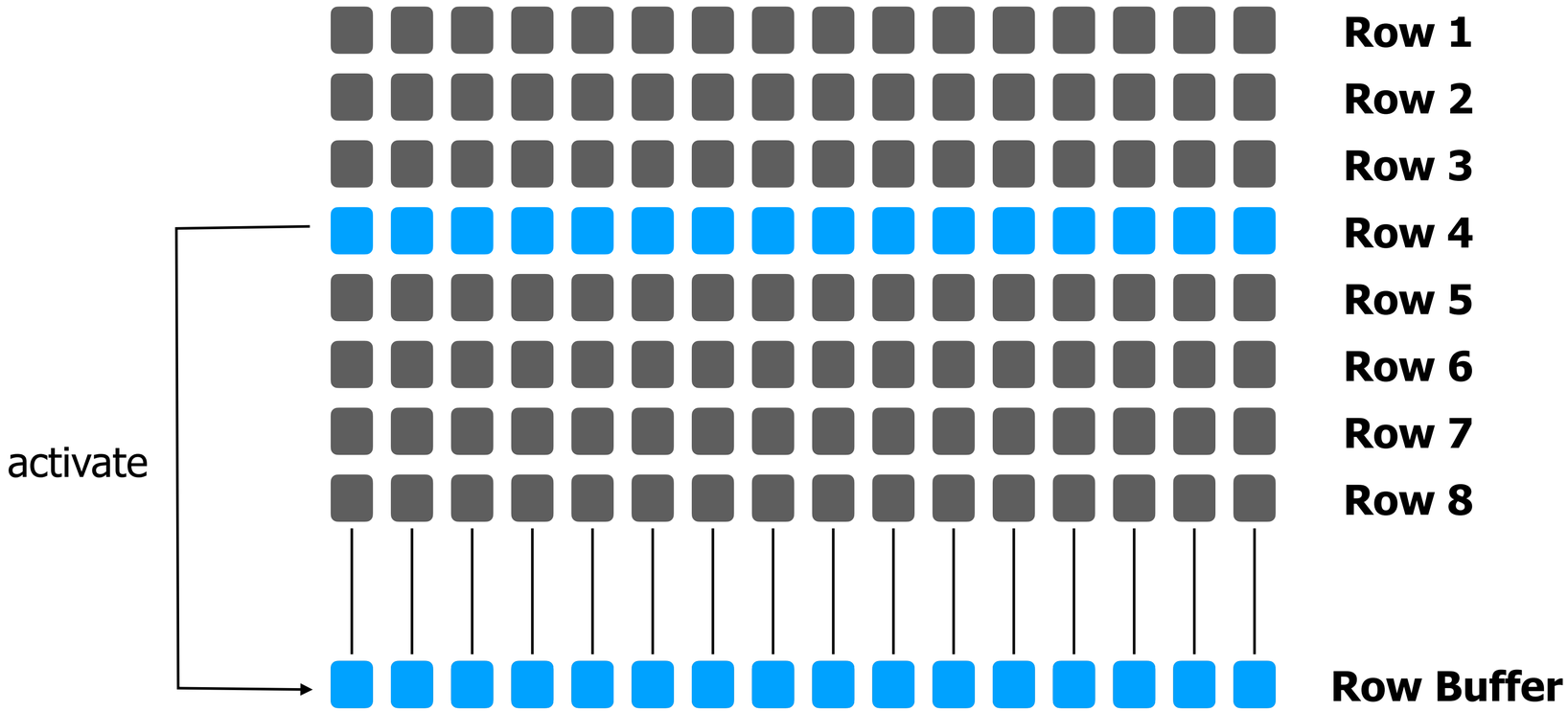
DRAM Bank (activate)



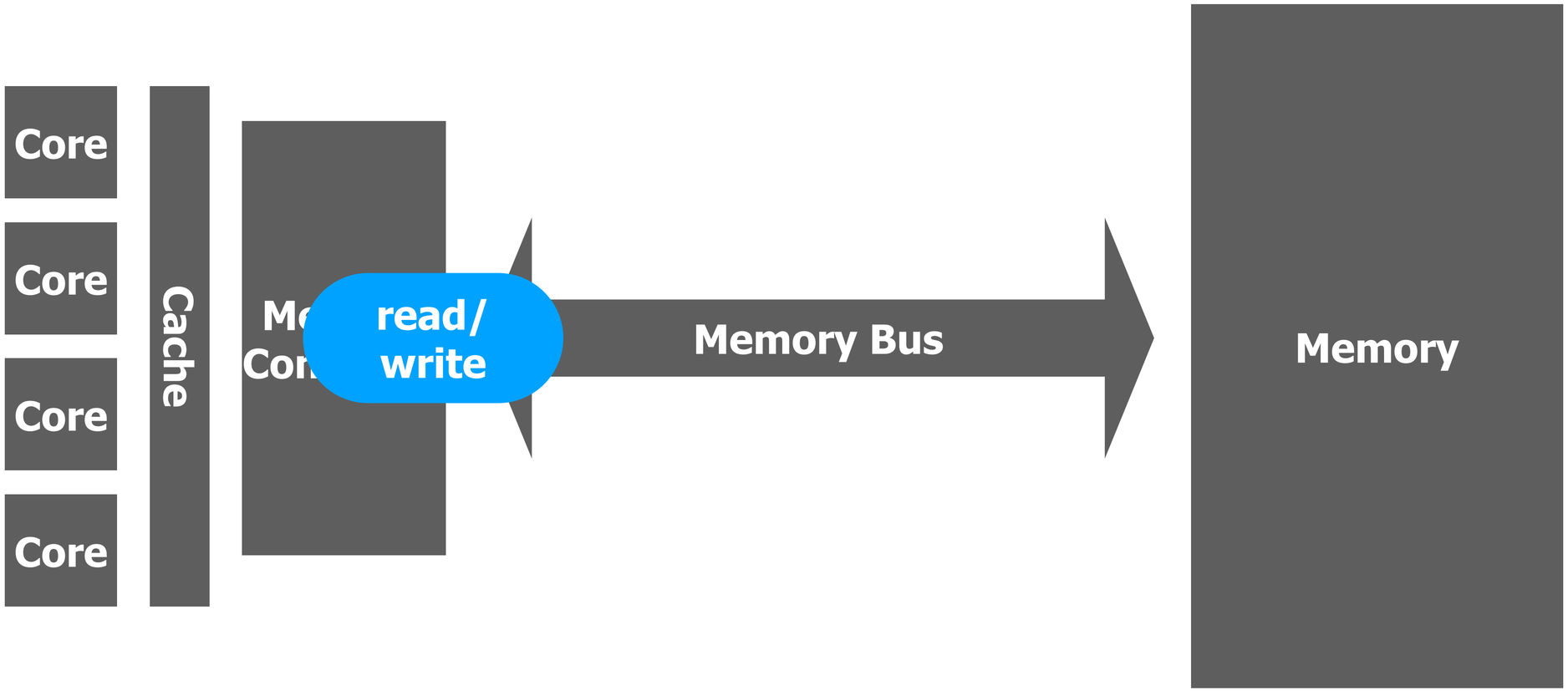
DRAM Bank (activate)



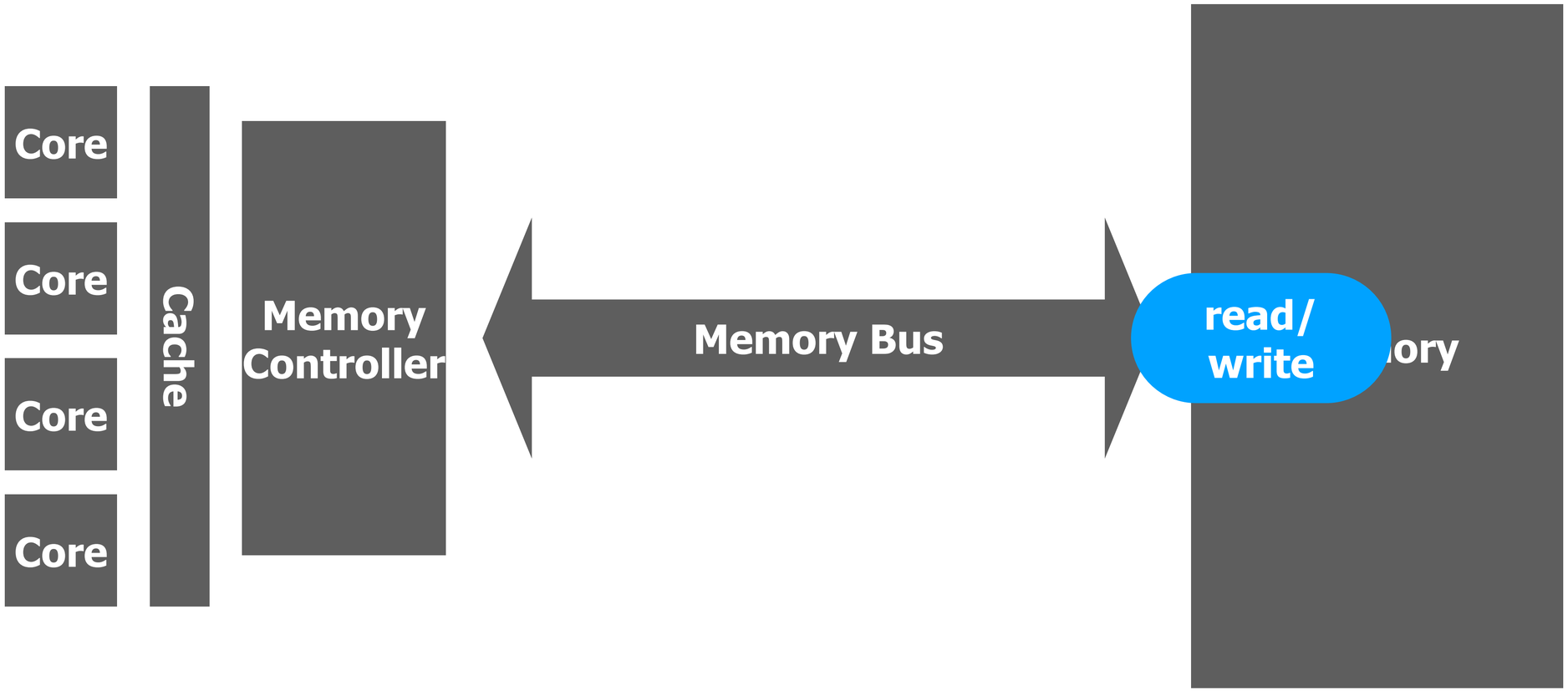
DRAM Bank (activate)



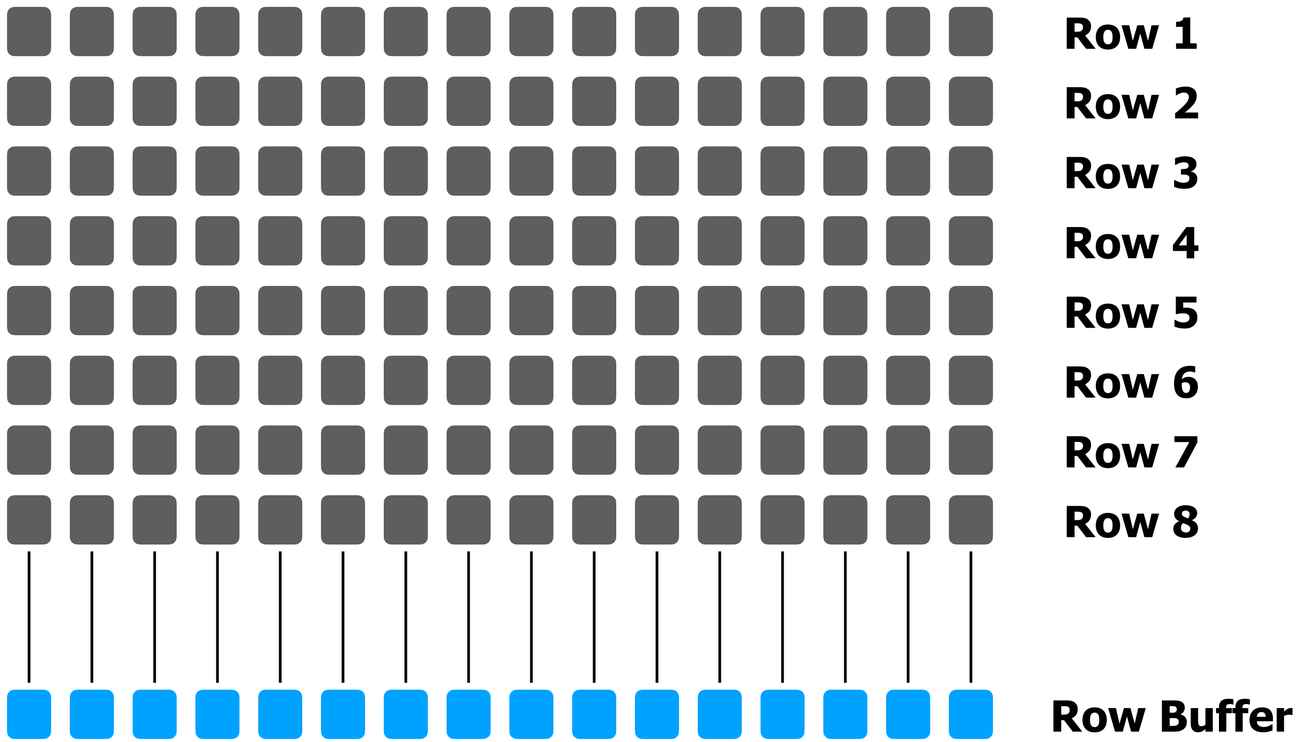
Memory Controllers



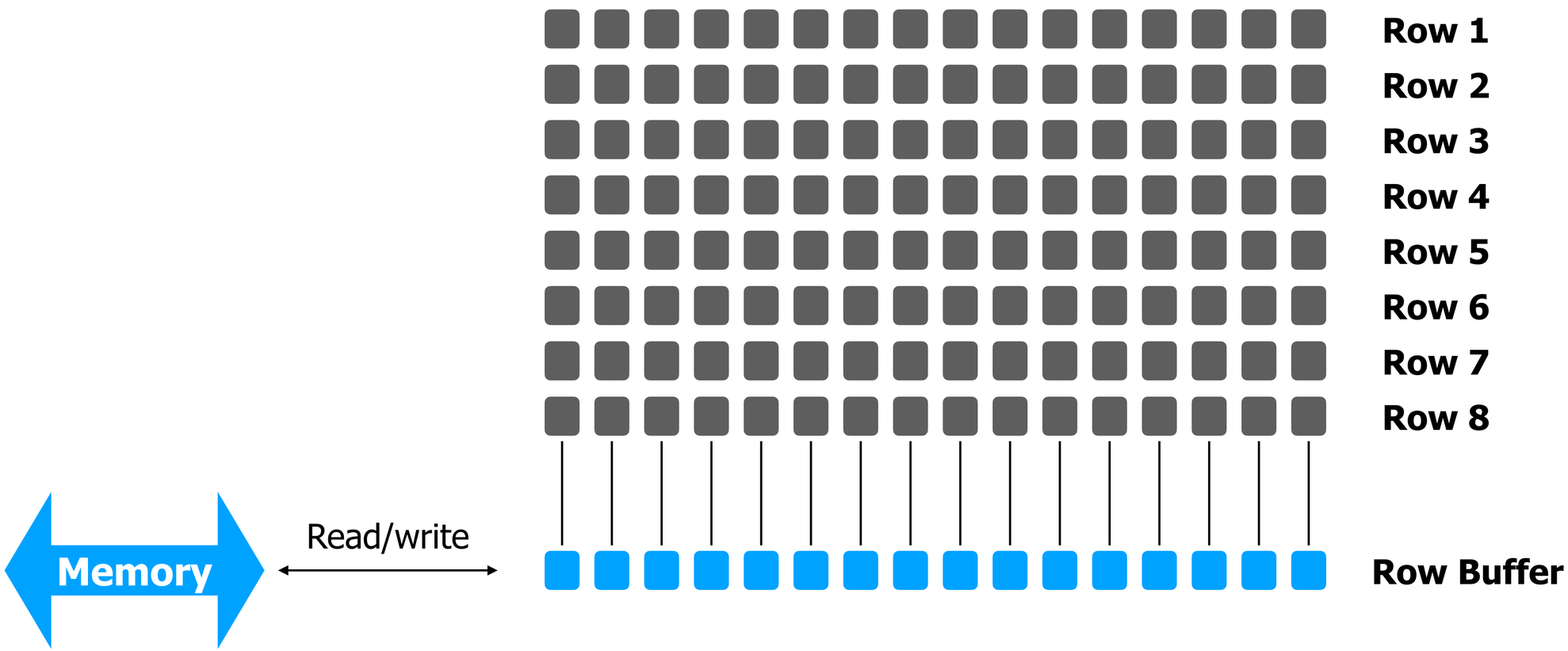
Memory Controllers



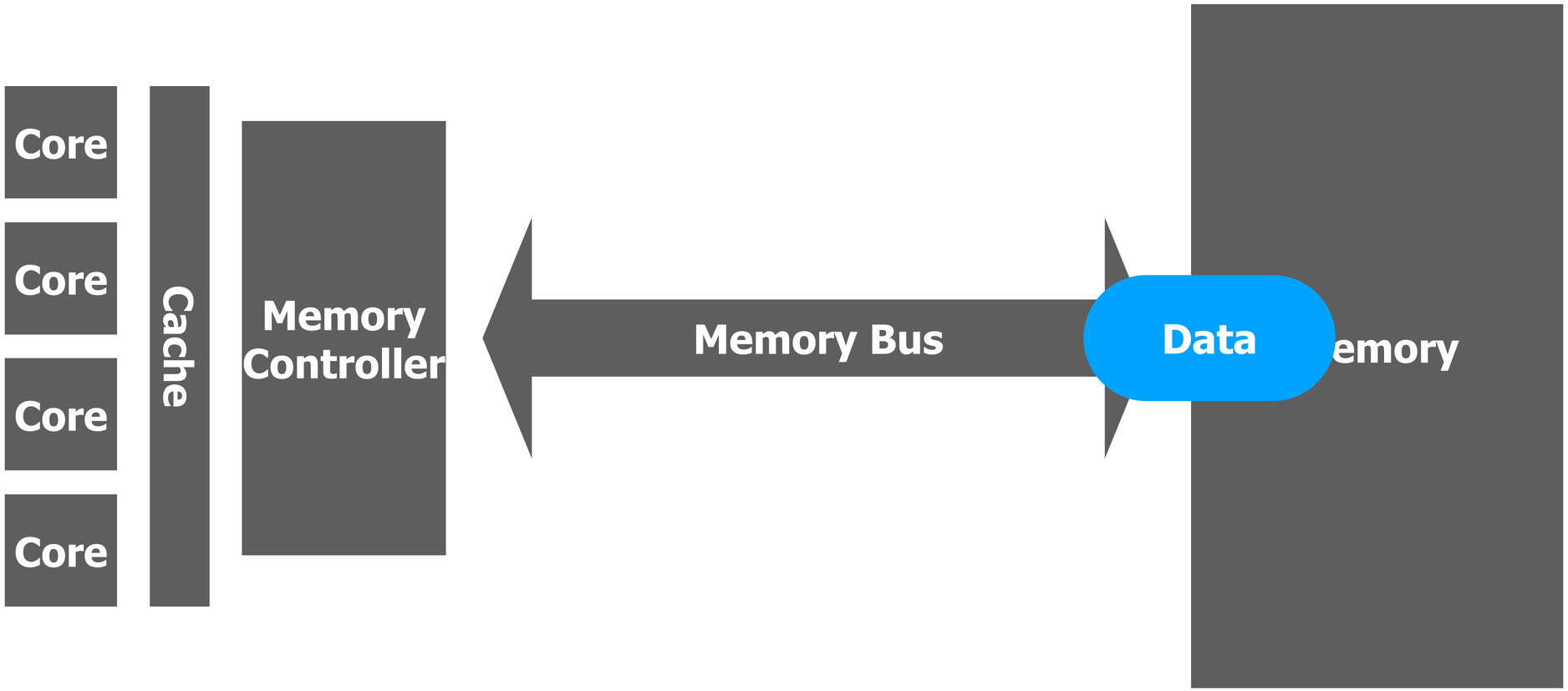
DRAM Bank (read/write)



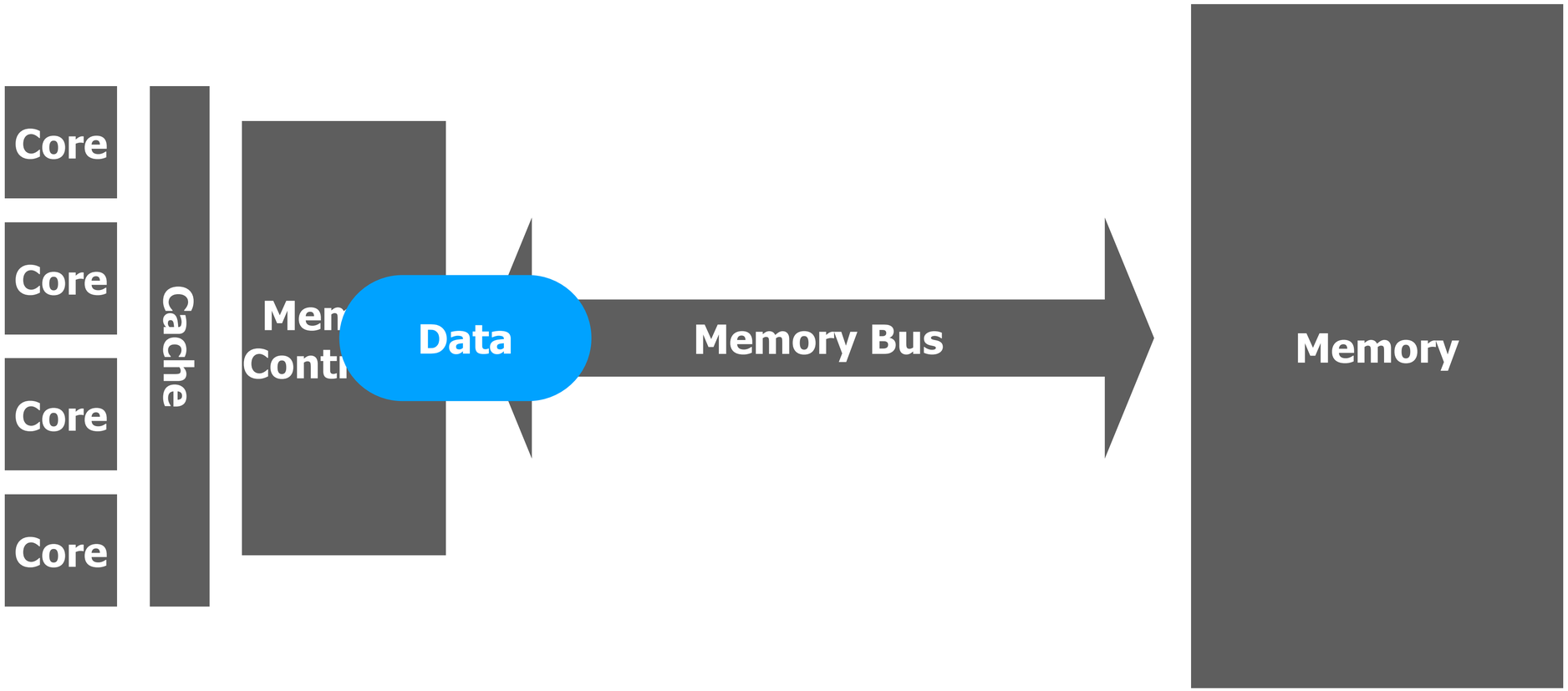
DRAM Bank (read/write)



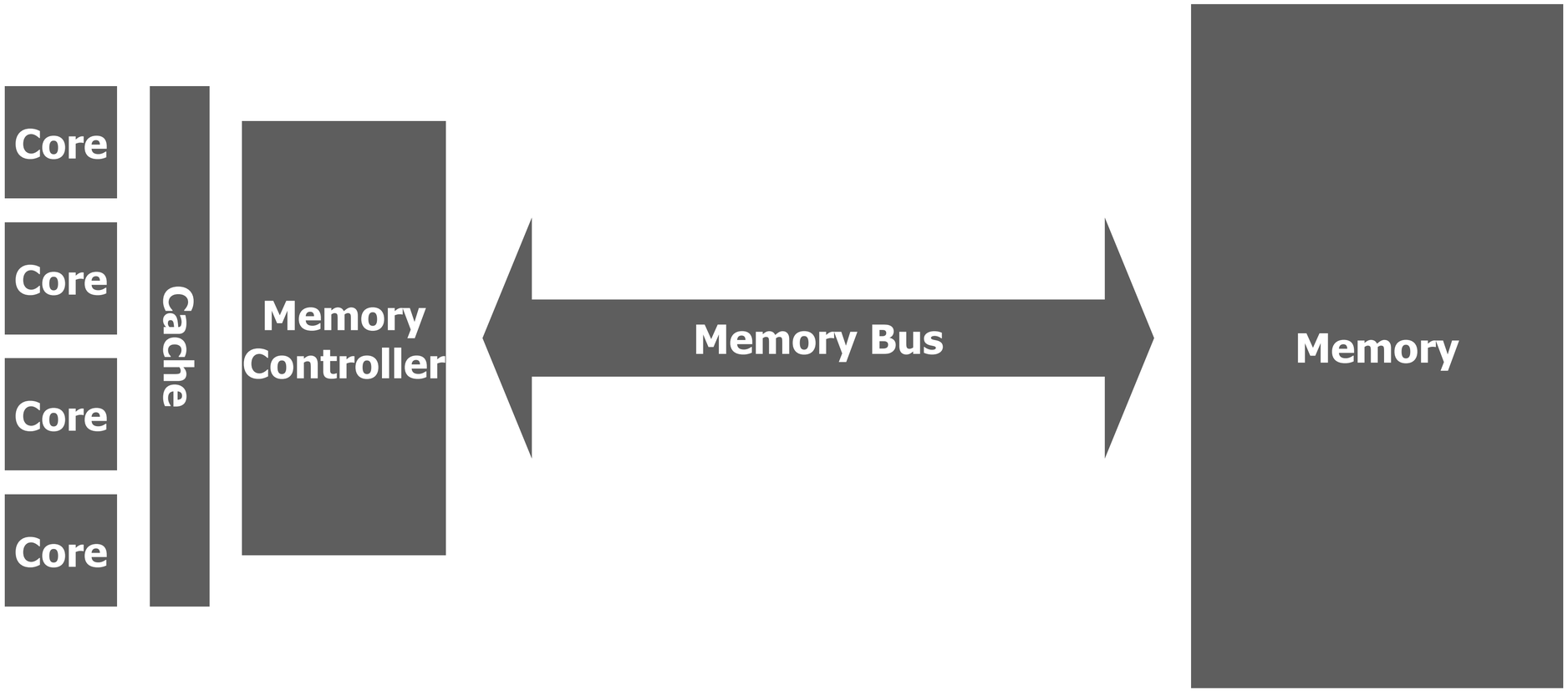
Memory Controllers



Memory Controllers



Memory Controllers



Scalability Issue

Scalability Issue

Off-chip bandwidth scalability is limited

Scalability Issue

Off-chip bandwidth scalability is limited

Off-chip bandwidth presents a serious impediment to CMP scalability

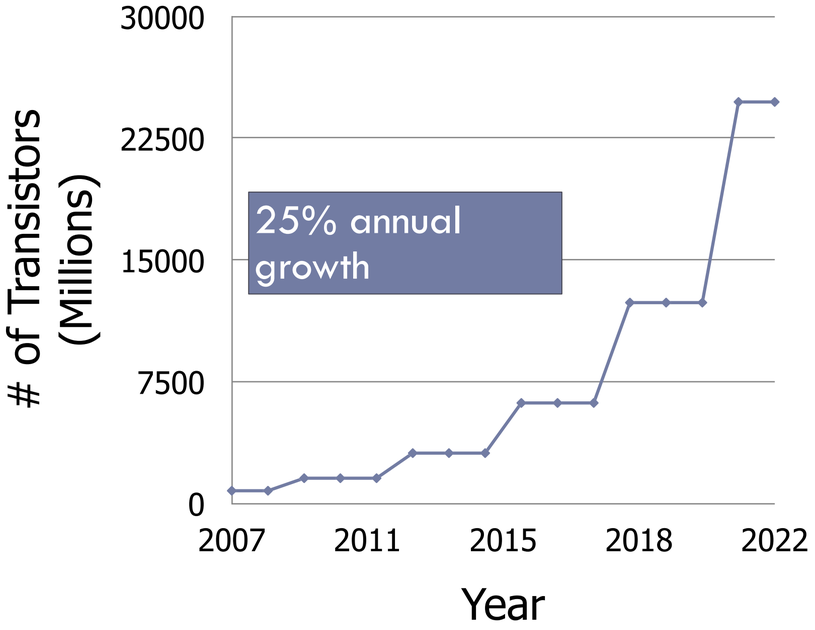
Scalability Issue

Off-chip bandwidth scalability is limited

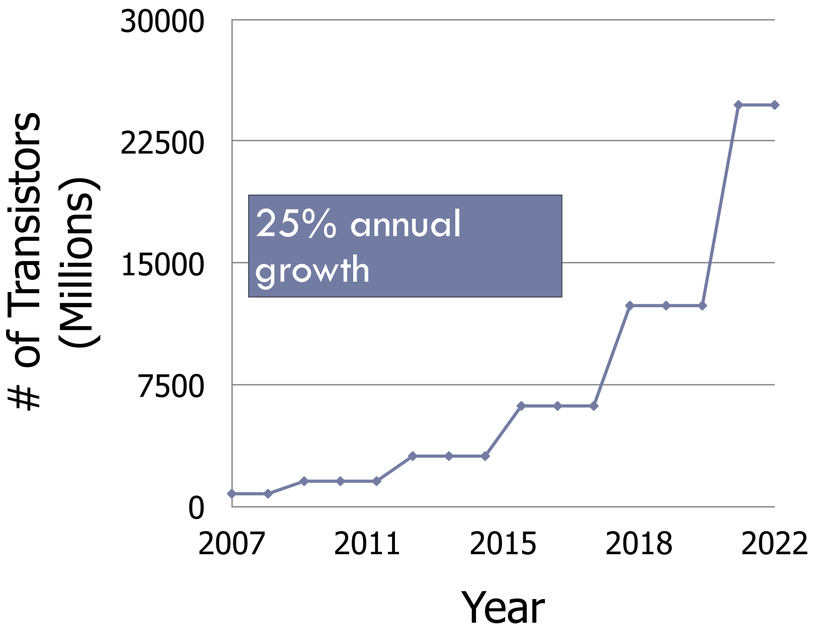
Off-chip bandwidth presents a serious impediment to CMP scalability

In practice only a fraction of the bandwidth can be used

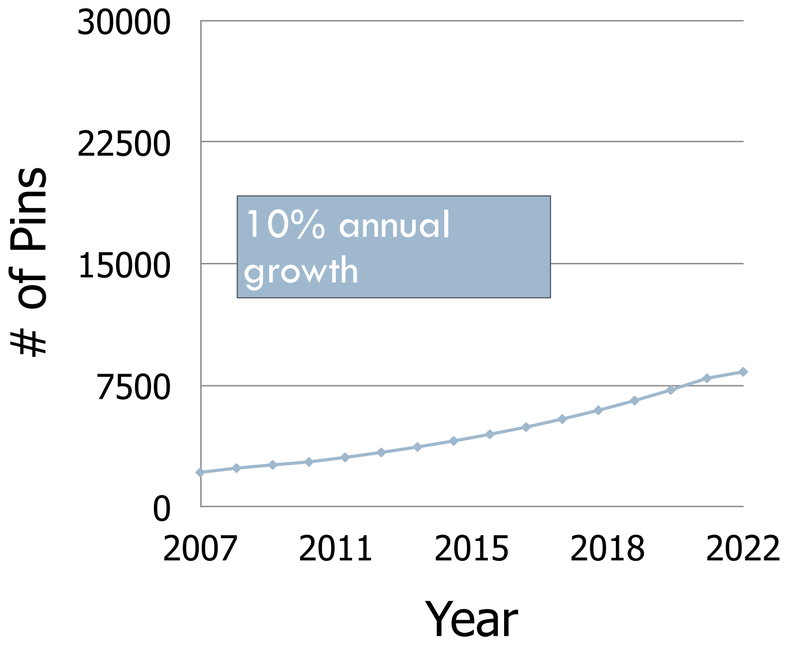
Transistor Count



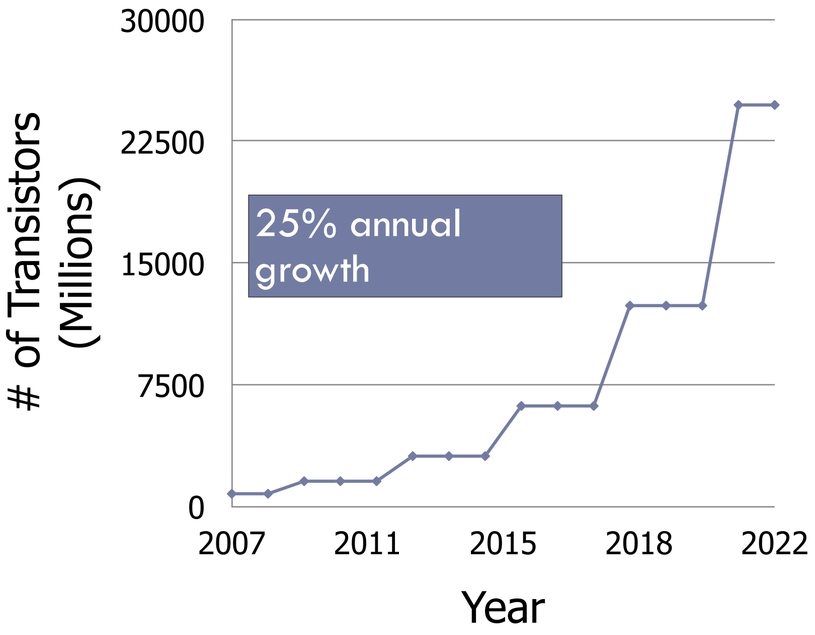
Transistor Count



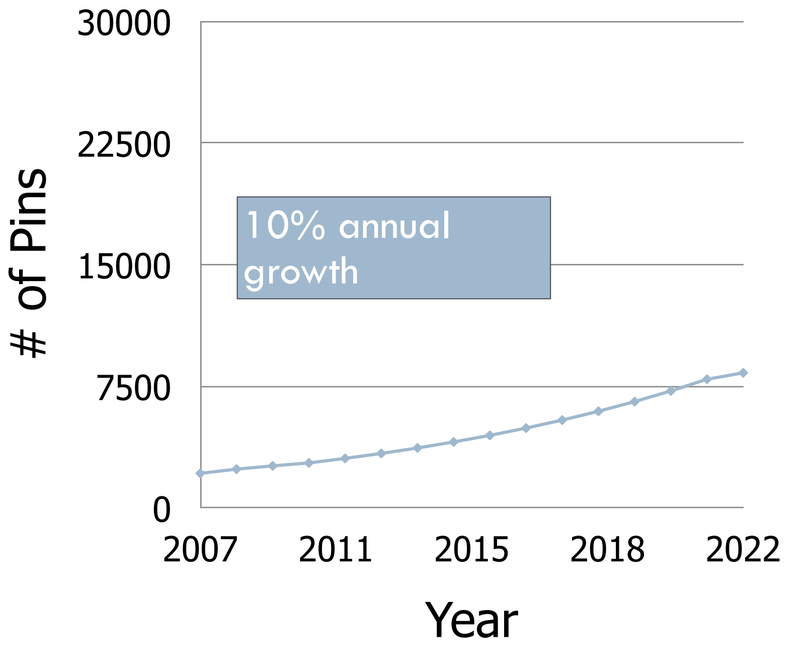
Pin Count



Transistor Count



Pin Count



Memory bandwidth, dictated by the pin count, poses a bottleneck for CMP performance

ITRS 2007 Executive Summary

Problem & Goal

Key Ideas

Novelty

Mechanisms & Implementation

Evaluation & Results

Main Takeaways

Problem & Goal

Key Ideas

Novelty

Mechanisms & Implementation

Evaluation & Results

Main Takeaways

Problem & Goal

Problem & Goal

State-of-the-art memory controllers deliver low performance due to their fixed, rigid access scheduling policies designed for average-case behaviour

Problem & Goal

State-of-the-art memory controllers deliver low performance due to their fixed, rigid access scheduling policies designed for average-case behaviour

For improvements in CMP architectures, off-chip bandwidth of the memory bus presents a serious impediment to its scalability

Problem & Goal

State-of-the-art memory controllers deliver low performance due to their fixed, rigid access scheduling policies designed for average-case behaviour

For improvements in CMP architectures, off-chip bandwidth of the memory bus presents a serious impediment to its scalability

DRAM scheduling is a complex problem, as workloads demand differing scheduling policies

Problem & Goal

State-of-the-art memory controllers deliver low performance due to their fixed, rigid access scheduling policies designed for average-case behaviour

For improvements in CMP architectures, off-chip bandwidth of the memory bus presents a serious impediment to its scalability

DRAM scheduling is a complex problem, as workloads demand differing scheduling policies

Goal: Improve Performance of off-chip bandwidth by designing a better memory controller.

Problem & Goal

Key Ideas

Novelty

Mechanisms & Implementation

Evaluation & Results

Main Takeaways

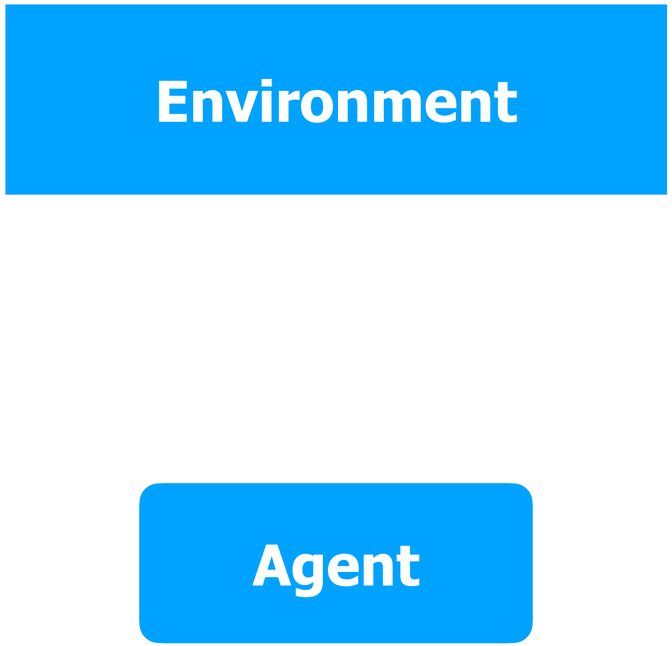
Key Ideas

Reinforcement Learning as a self-optimizing agent maps to a flexible, self-optimizing scheduler

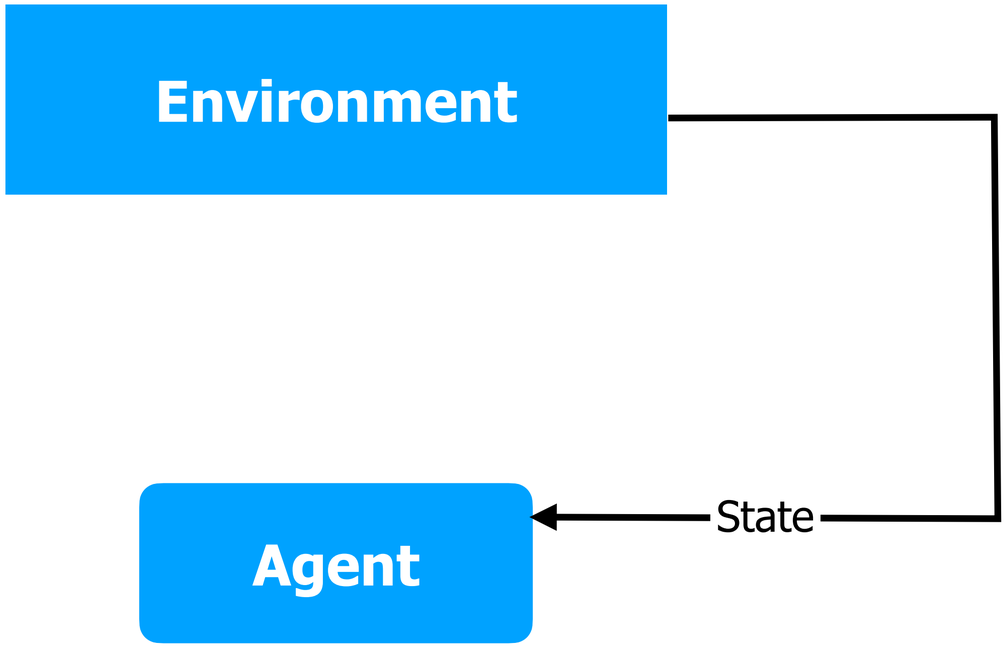
Make memory scheduling easier and more efficient

Allows for flexible scheduling in multiple different workloads

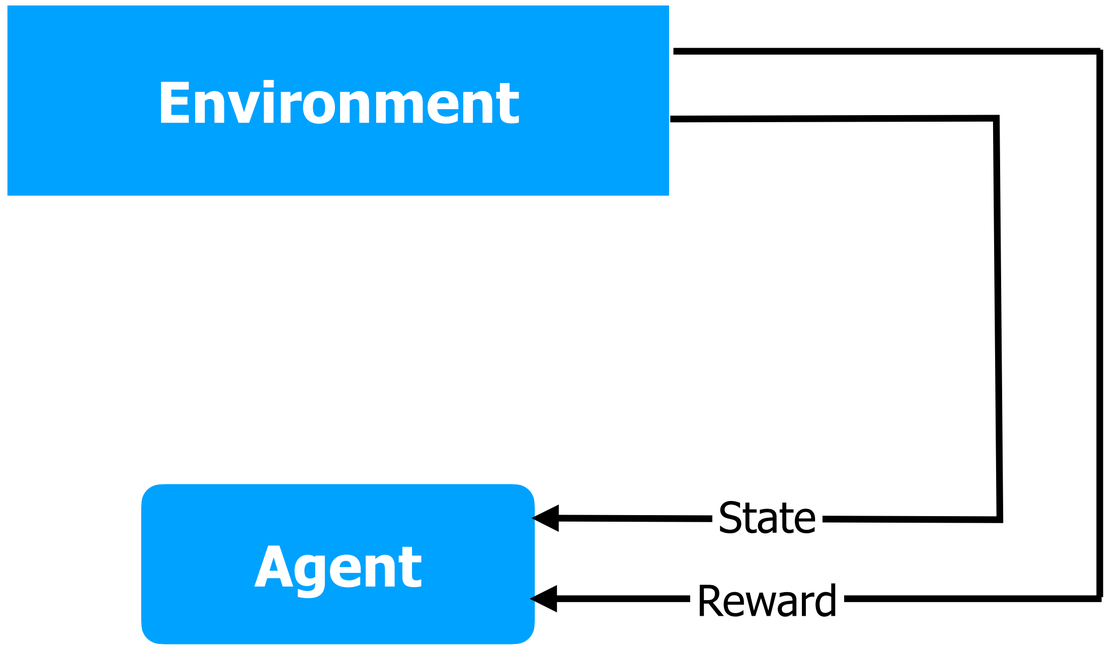
Reinforcement Learning



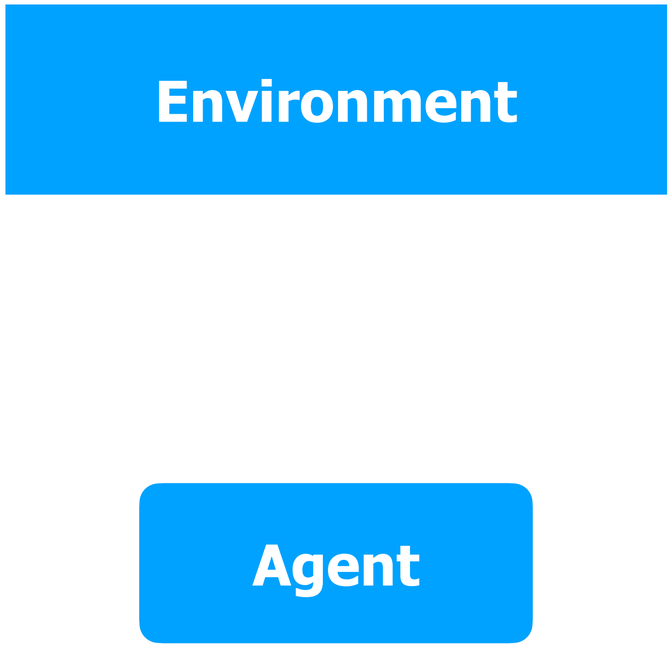
Reinforcement Learning



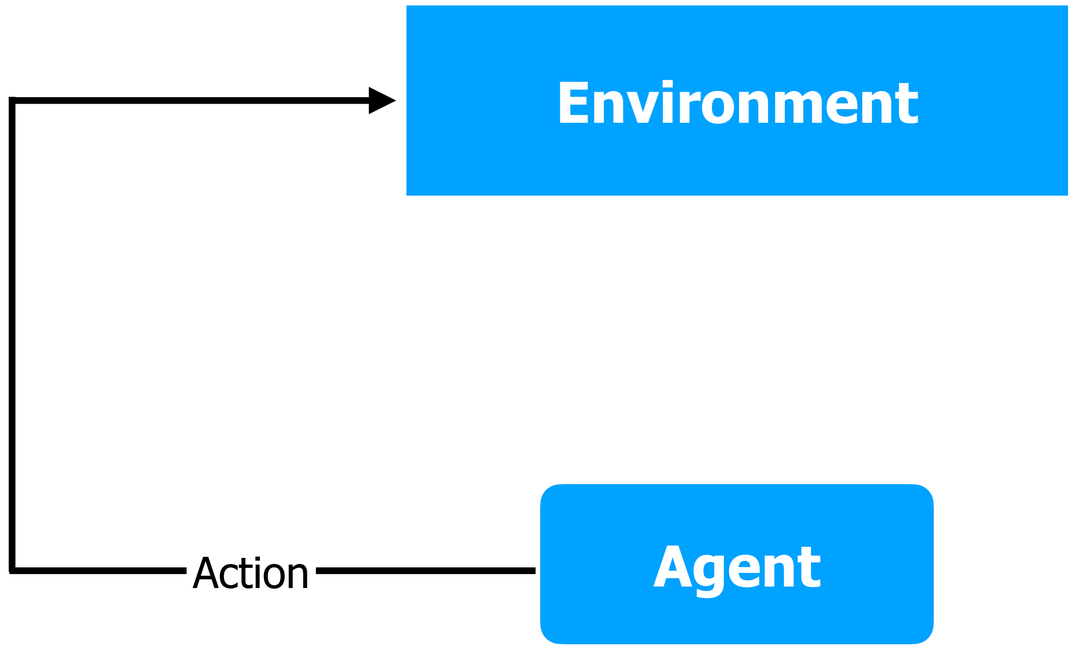
Reinforcement Learning



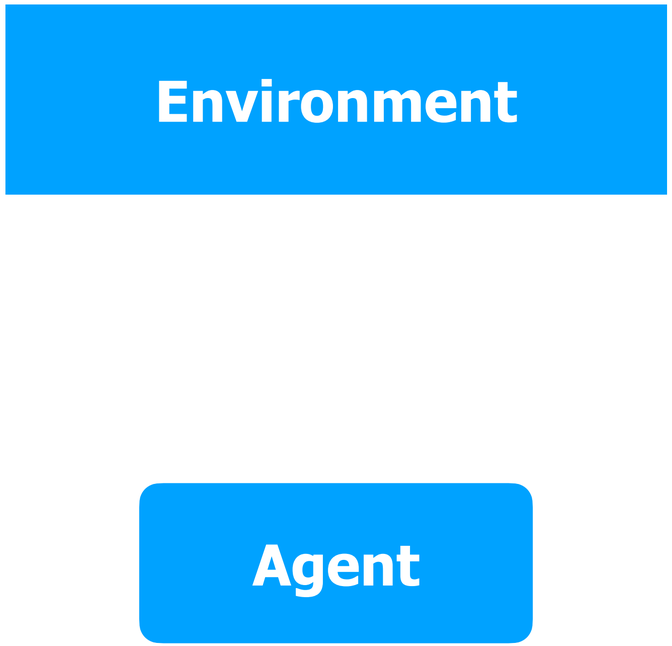
Reinforcement Learning



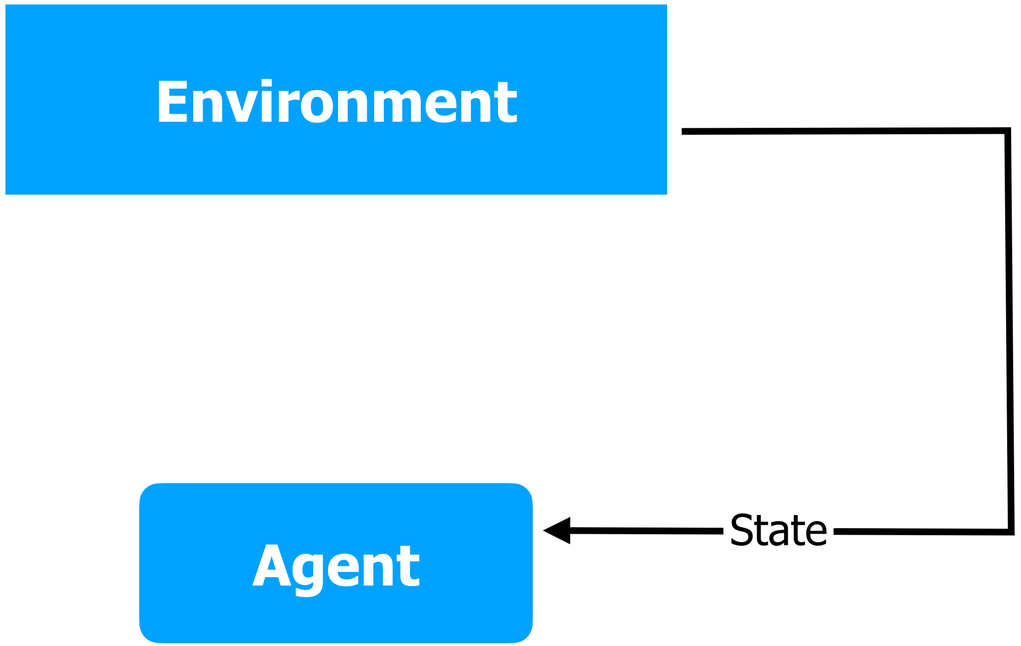
Reinforcement Learning



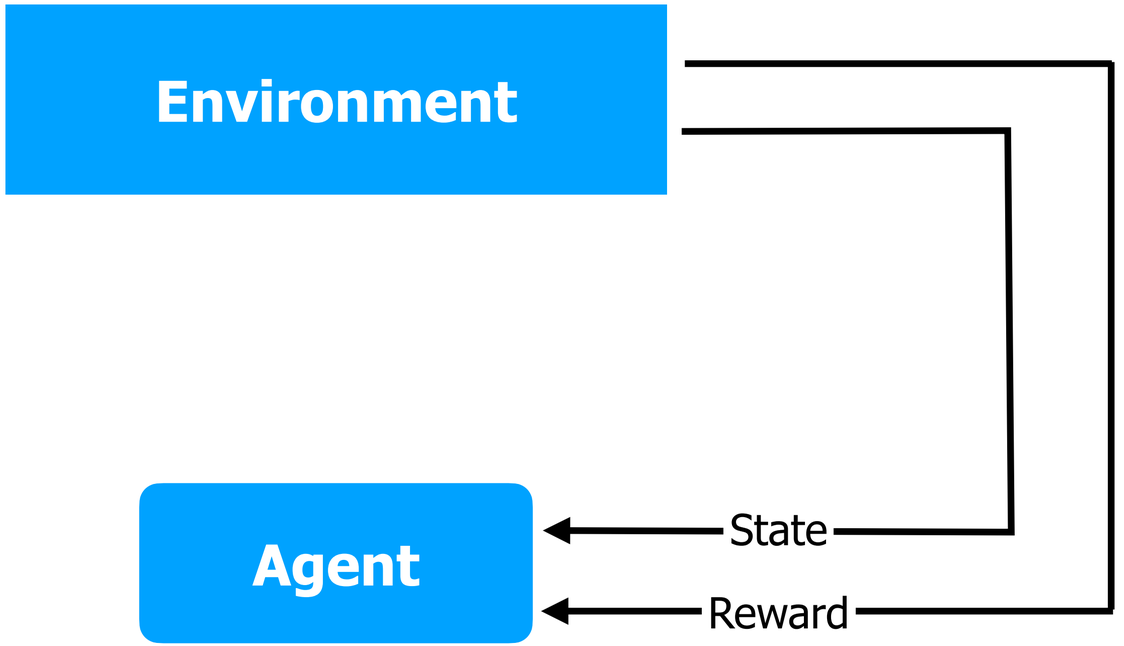
Reinforcement Learning



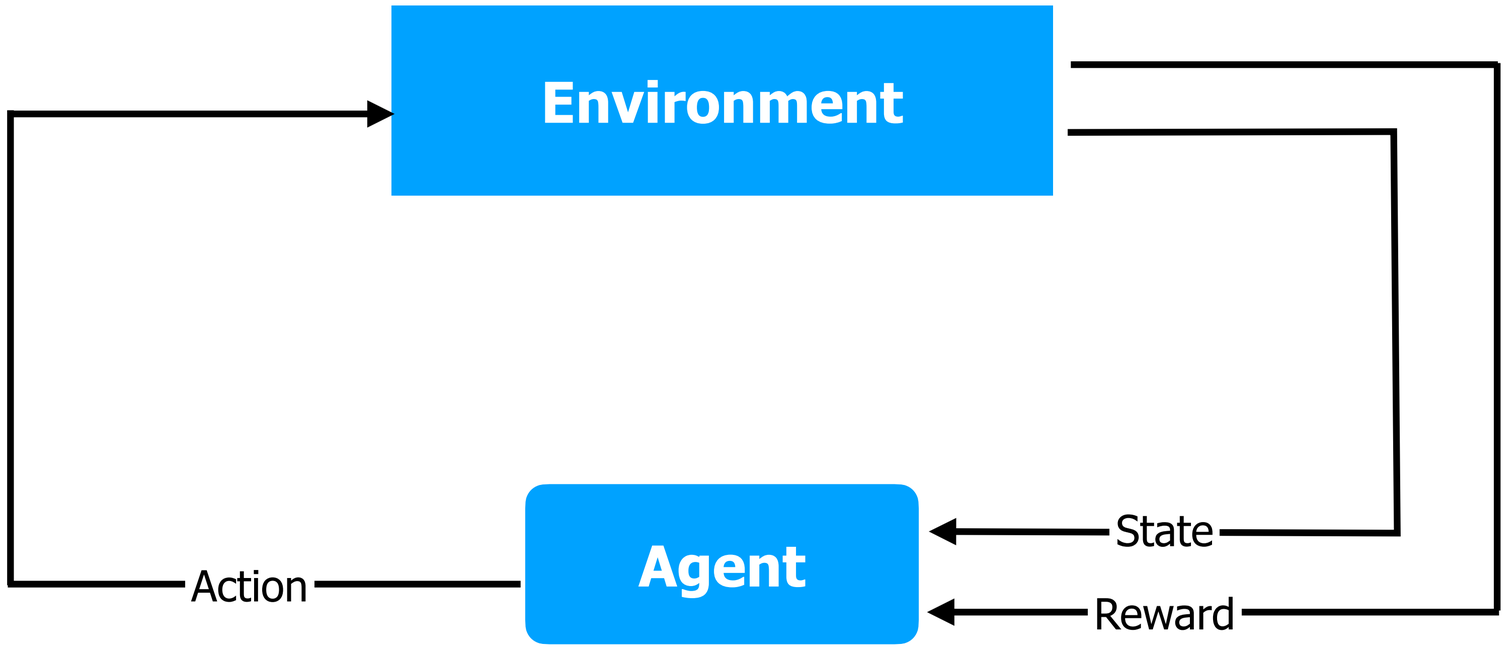
Reinforcement Learning



Reinforcement Learning



Reinforcement Learning



Applicability to Memory Controllers

Applicability to Memory Controllers

Applicability to Memory Controllers

RL applies well to Memory Controller design

Applicability to Memory Controllers

RL applies well to Memory Controller design

Easy translation of Environment, Agent, Action, Reward, and State to System, Scheduler, Command, Bus Utilization, and State Attributes

Applicability to Memory Controllers

RL applies well to Memory Controller design

Easy translation of Environment, Agent, Action, Reward, and State to System, Scheduler, Command, Bus Utilization, and State Attributes

Applicability to Memory Controllers

RL applies well to Memory Controller design

Easy translation of Environment, Agent, Action, Reward, and State to System, Scheduler, Command, Bus Utilization, and State Attributes

Applicability to Memory Controllers

RL applies well to Memory Controller design

Easy translation of Environment, Agent, Action, Reward, and State to System, Scheduler, Command, Bus Utilization, and State Attributes

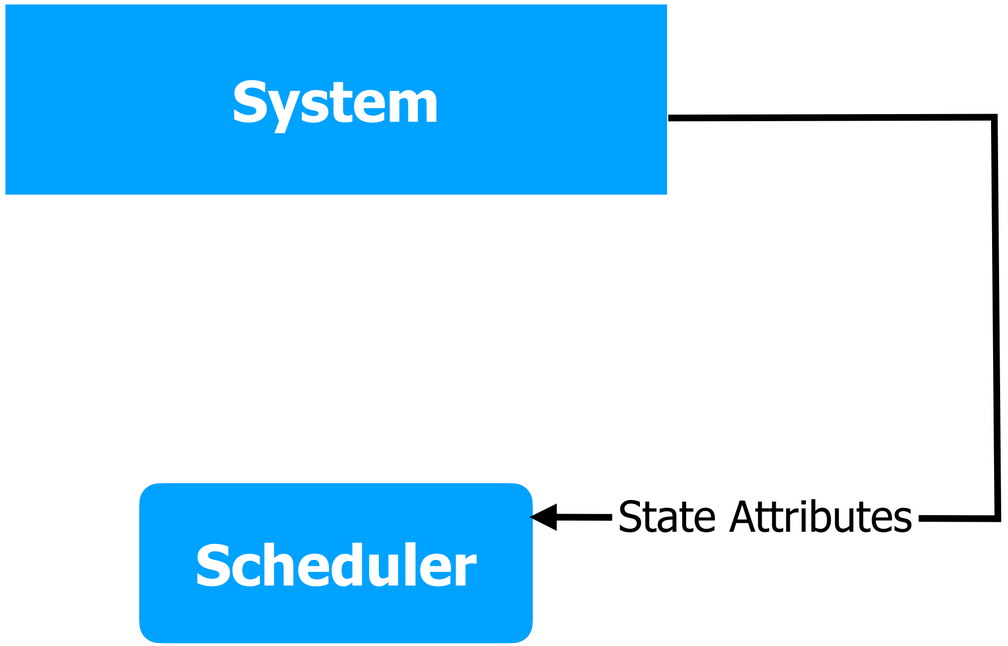
Key Idea: Design the memory controller as an RL agent whose goal is to learn an optimal memory scheduling policy via interaction with the system

Applicability to Memory Controllers

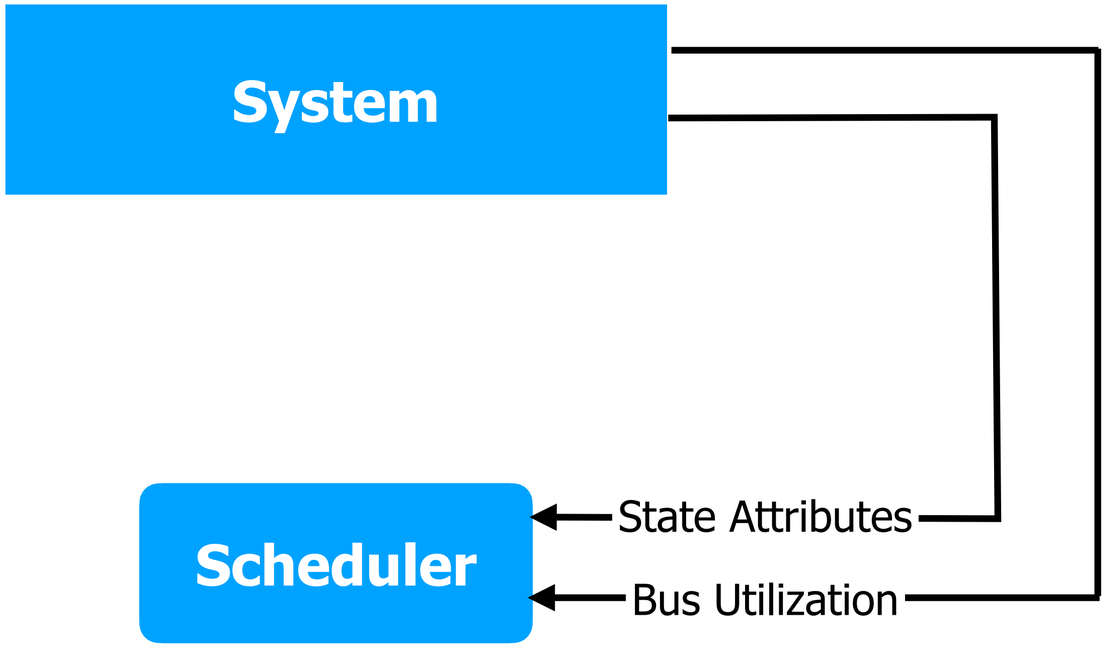
System

Scheduler

Applicability to Memory Controllers



Applicability to Memory Controllers

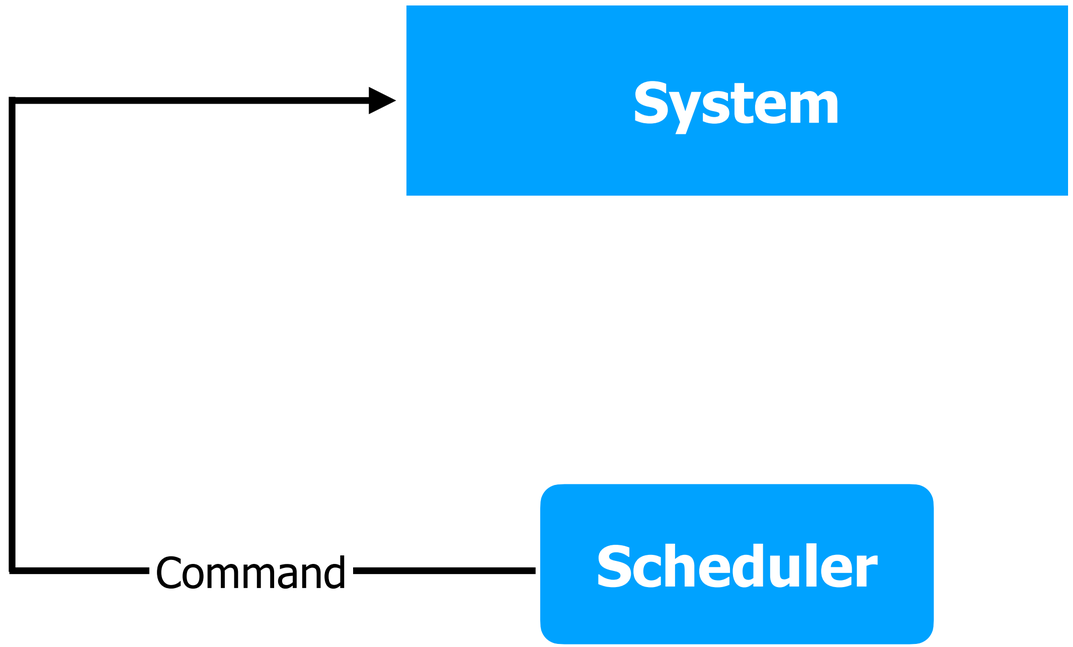


Applicability to Memory Controllers

System

Scheduler

Applicability to Memory Controllers

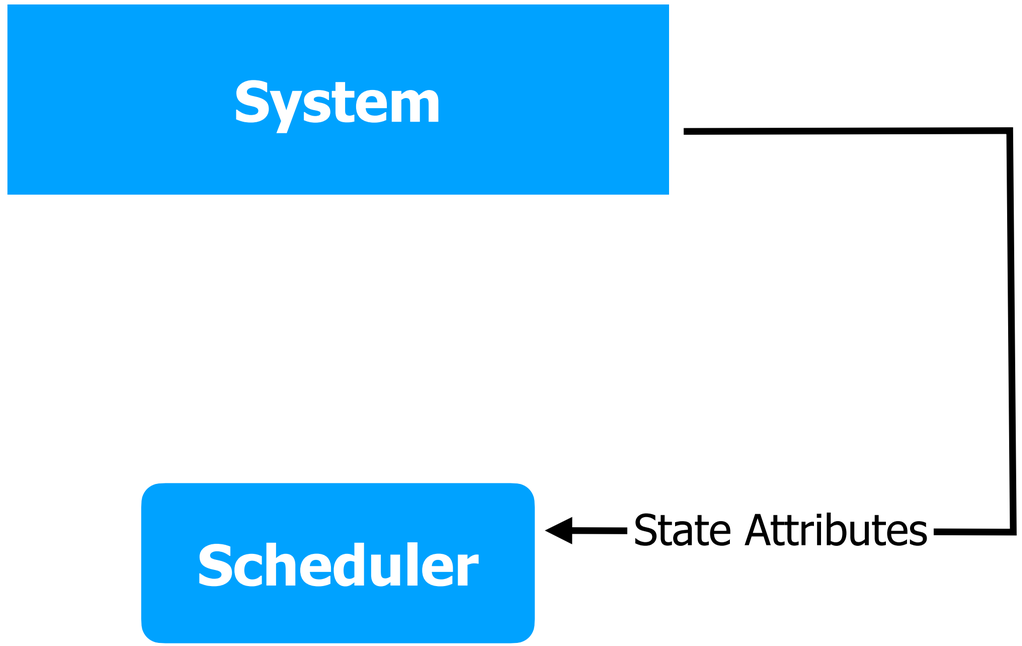


Applicability to Memory Controllers

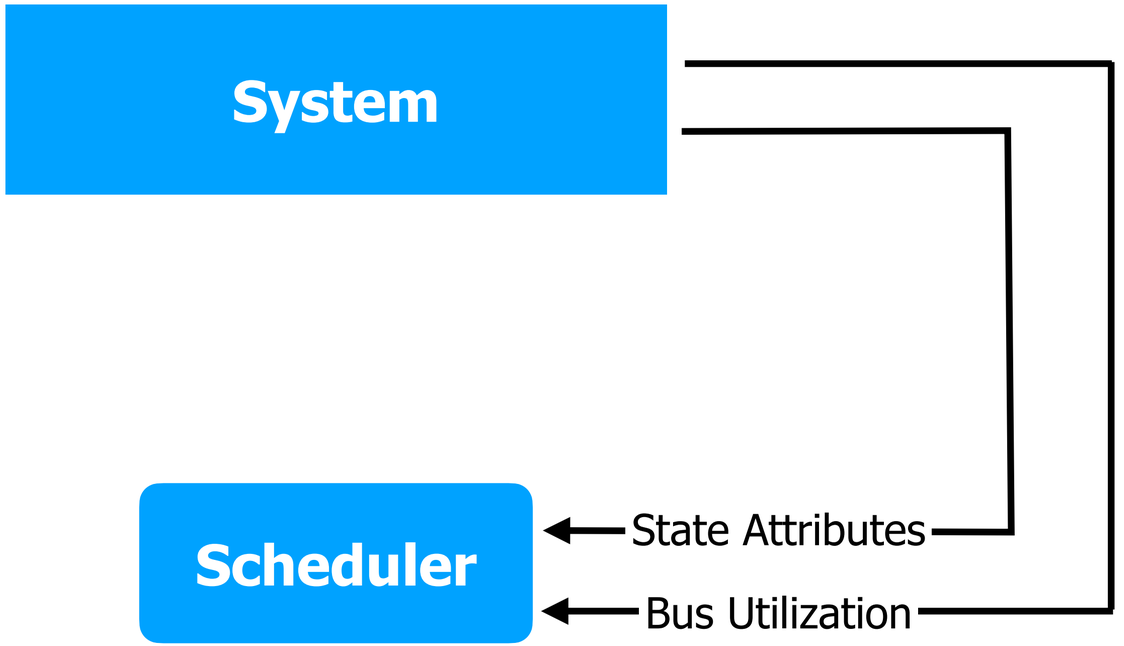
System

Scheduler

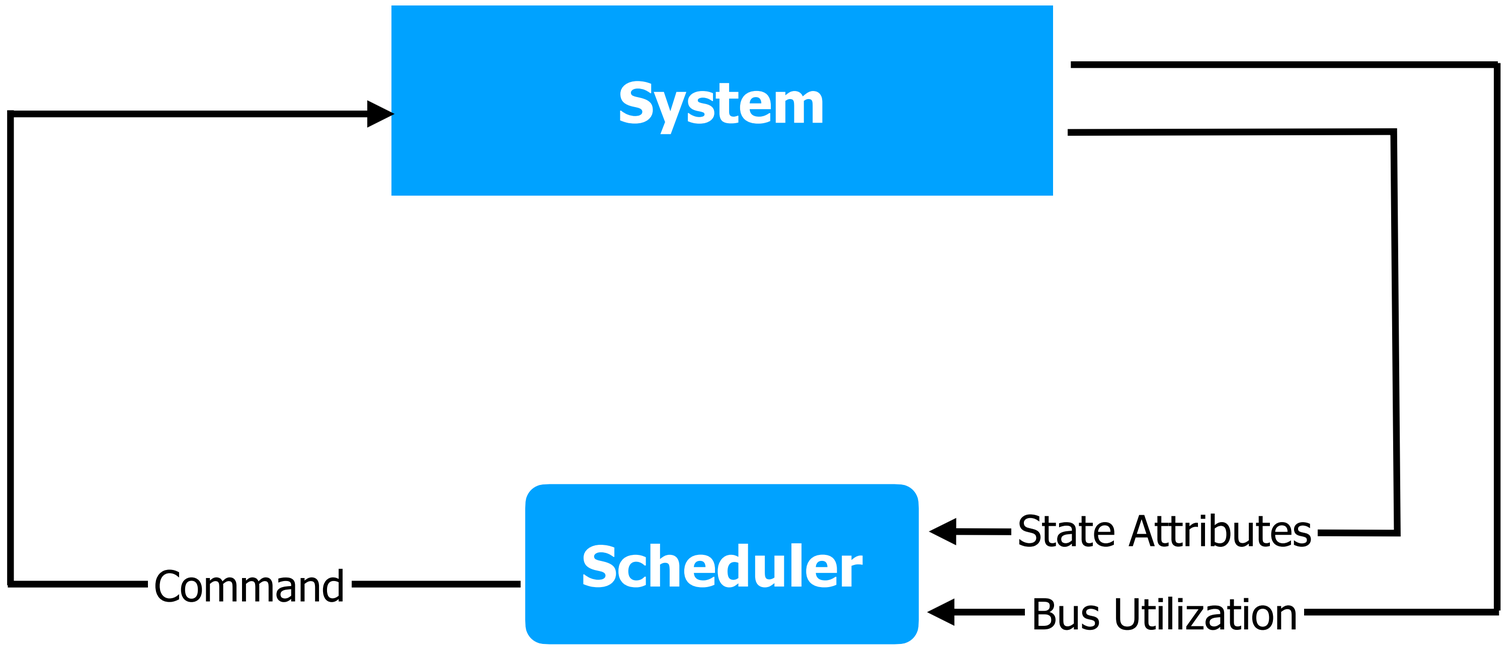
Applicability to Memory Controllers



Applicability to Memory Controllers



Applicability to Memory Controllers



Challenges [1/2]

Challenges [1/2]

Temporal Credit Assignment: Agent must be able to anticipate long-term consequences of its actions

Challenges [1/2]

Temporal Credit Assignment: Agent must be able to anticipate long-term consequences of its actions

Exploration vs Exploitation: Agent must explore the environment enough to be able to make good decisions (exploration), but also follow and exploit a found strategy (exploitation)

Challenges [1/2]

Temporal Credit Assignment: Agent must be able to anticipate long-term consequences of its actions

Exploration vs Exploitation: Agent must explore the environment enough to be able to make good decisions (exploration), but also follow and exploit a found strategy (exploitation)

Generalization: Agent must be able to generalize to act on such a big space of possible configurations

Challenges [2/2]

Challenges [2/2]

Implementation of the agent should not incur additional latency

Challenges [2/2]

Implementation of the agent should not incur additional latency

Needs to run below DRAM cycle time

Challenges [2/2]

Implementation of the agent should not incur additional latency

Needs to run below DRAM cycle time

Proposed Solution: Use dedicated hardware

Problem & Goal

Key Ideas

Novelty

Mechanisms & Implementation

Evaluation & Results

Main Takeaways

Novelty

Novelty

Reinforcement Learning (RL) based self-optimizing memory controller

Novelty

Reinforcement Learning (RL) based self-optimizing memory controller

Allows hardware designers to define a performance target instead of a fixed policy

Novelty

Reinforcement Learning (RL) based self-optimizing memory controller

Allows hardware designers to define a performance target instead of a fixed policy

First to apply Machine Learning in memory controllers

Problem & Goal

Key Ideas

Novelty

Mechanisms & Implementation

Evaluation & Results

Main Takeaways

Rewards

Rewards

Rewards represent the goal towards which is being optimized

Rewards

Rewards represent the goal towards which is being optimized

Current Bus Utilization is the reward for this agent

Feature Selection

Feature Selection

The features describe the system state to the agent

Feature Selection

The features describe the system state to the agent

Selected Features:

Feature Selection

The features describe the system state to the agent

Selected Features:

Feature Selection

The features describe the system state to the agent

Selected Features:

1. Number of Reads

Feature Selection

The features describe the system state to the agent

Selected Features:

1. Number of Reads
2. Number of Writes

Feature Selection

The features describe the system state to the agent

Selected Features:

1. Number of Reads
2. Number of Writes
3. Number of reads, that are load misses

Feature Selection

The features describe the system state to the agent

Selected Features:

1. Number of Reads
2. Number of Writes
3. Number of reads, that are load misses
4. Relation of command to a load miss in core C

Feature Selection

The features describe the system state to the agent

Selected Features:

1. Number of Reads
2. Number of Writes
3. Number of reads, that are load misses
4. Relation of command to a load miss in core C
5. Number of writes waiting for row referenced by command considered

Feature Selection

The features describe the system state to the agent

Selected Features:

1. Number of Reads
2. Number of Writes
3. Number of reads, that are load misses
4. Relation of command to a load miss in core C
5. Number of writes waiting for row referenced by command considered
6. Number of oldest load misses waiting for row referenced by command considered per core

Feature Selection

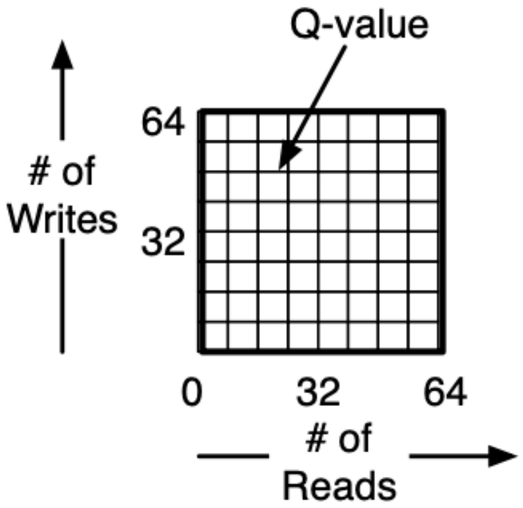
The features describe the system state to the agent

Selected Features:

1. Number of Reads
2. Number of Writes
3. Number of reads, that are load misses
4. Relation of command to a load miss in core C
5. Number of writes waiting for row referenced by command considered
6. Number of oldest load misses waiting for row referenced by command considered per core

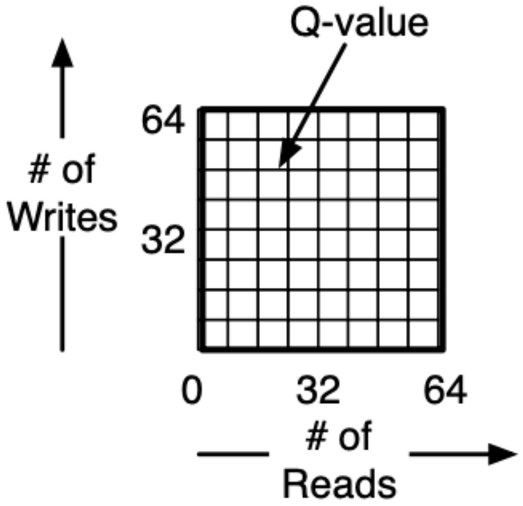
Features were selected through an automated feature selection process

Q-Values



Q-Values

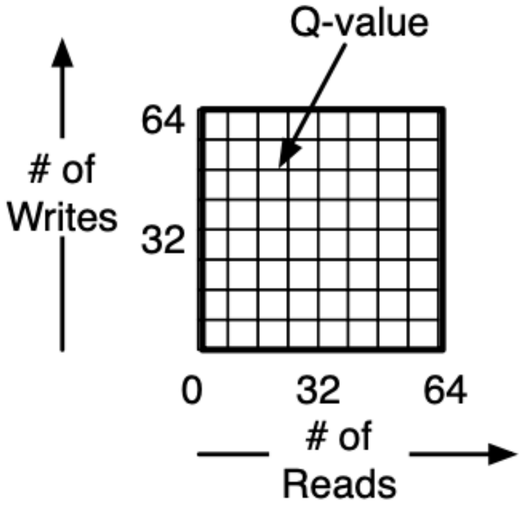
Q-values approximate the cumulative long-term reward for each state action pair



Q-Values

Q-values approximate the cumulative long-term reward for each state action pair

Will update itself using a function of the reward, and previous Q-values, and the new arrived state

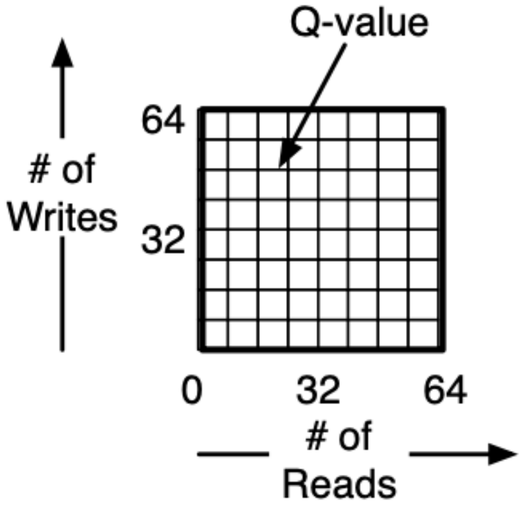


Q-Values

Q-values approximate the cumulative long-term reward for each state action pair

Will update itself using a function of the reward, and previous Q-values, and the new arrived state

The values are updated through a bellman equation:

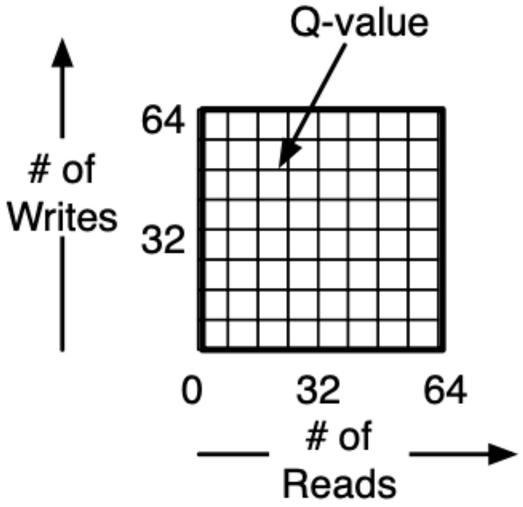


Q-Values

Q-values approximate the cumulative long-term reward for each state action pair

Will update itself using a function of the reward, and previous Q-values, and the new arrived state

The values are updated through a bellman equation:



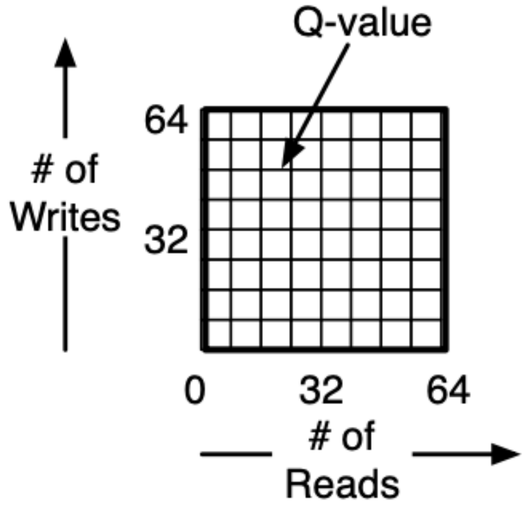
Q-Values

Q-values approximate the cumulative long-term reward for each state action pair

Will update itself using a function of the reward, and previous Q-values, and the new arrived state

The values are updated through a bellman equation:

$$Q(s_{prev}, a_{prev}) \leftarrow (1-\alpha)Q(s_{prev}, a_{prev}) + \alpha[r + \gamma Q(s_{current}, a_{current})]$$



CMAC representation

CMAC representation

There are too many Q-values to represent, takes up a lot of space:

CMAC representation

There are too many Q-values to represent, takes up a lot of space:

$$O(NumberOfStates * NumberOfActions) = O(TransactionQueueEntriesNumberOfAttributes * NumberOfActions)$$

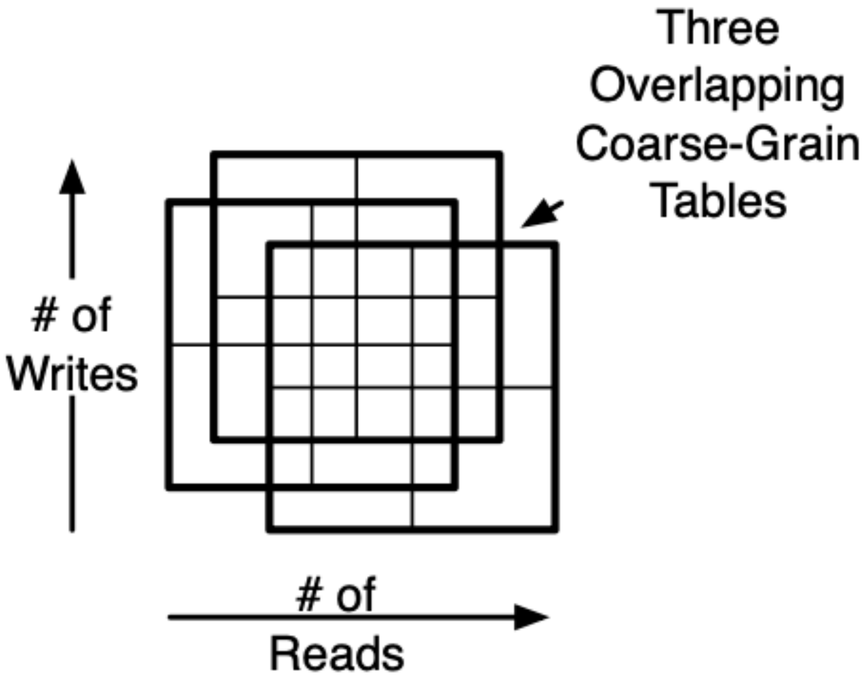
CMAC representation

There are too many Q-values to represent, takes up a lot of space:

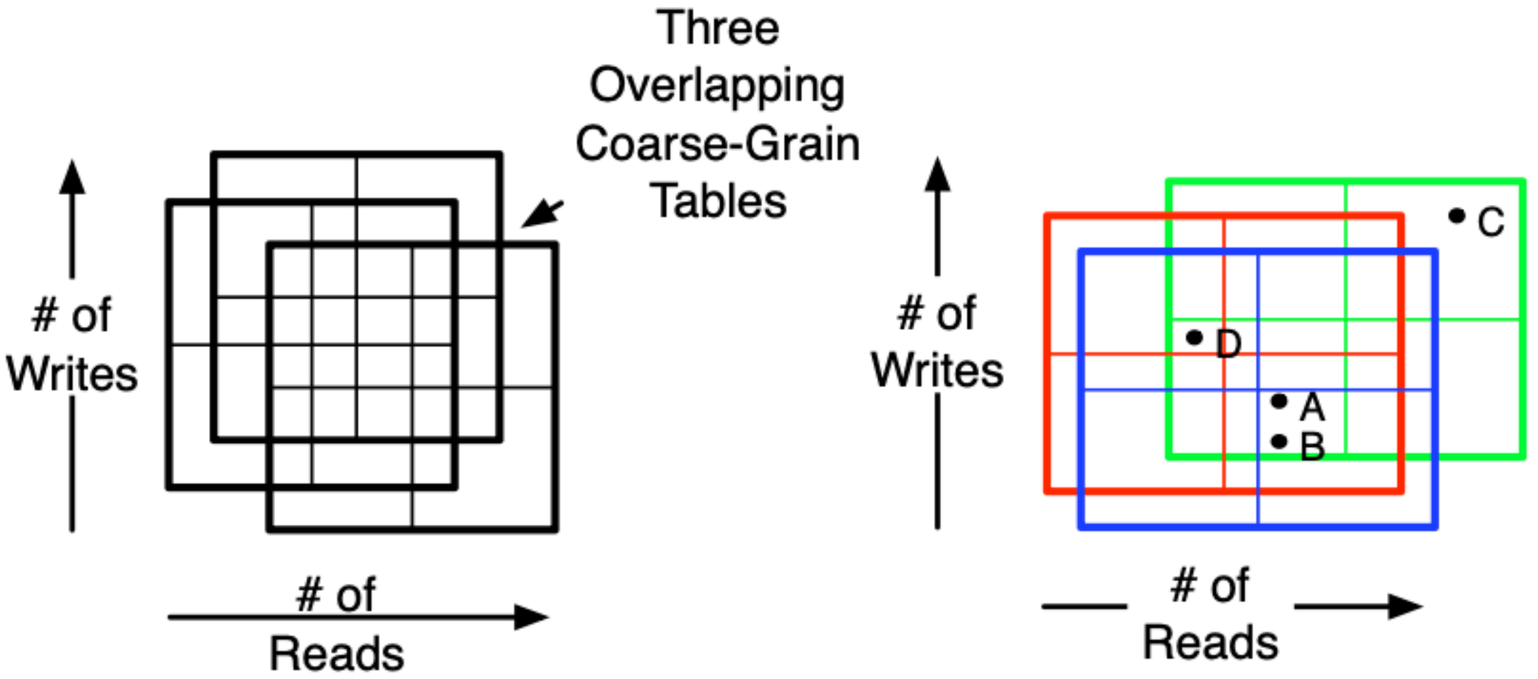
$$O(NumberOfStates * NumberOfActions) = O(TransactionQueueEntriesNumberOfAttributes * NumberOfActions)$$

CMAC representation is used for generalization and resolution:
Using overlapping coarse grained tables for adaptive resolution

CMAC representation

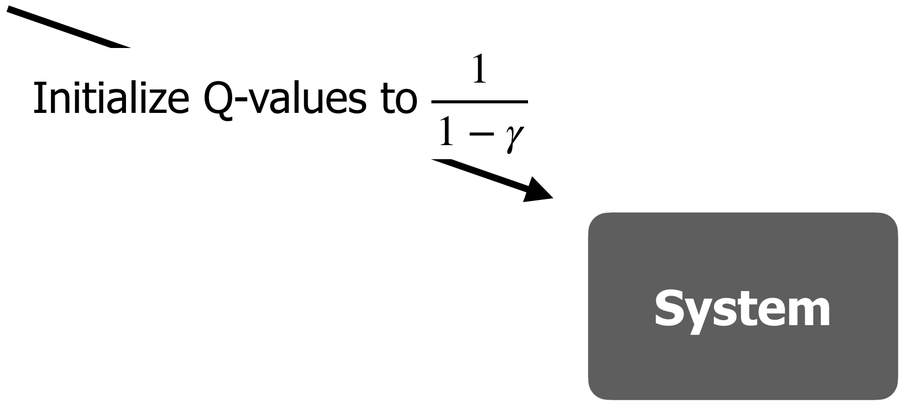


CMAC representation

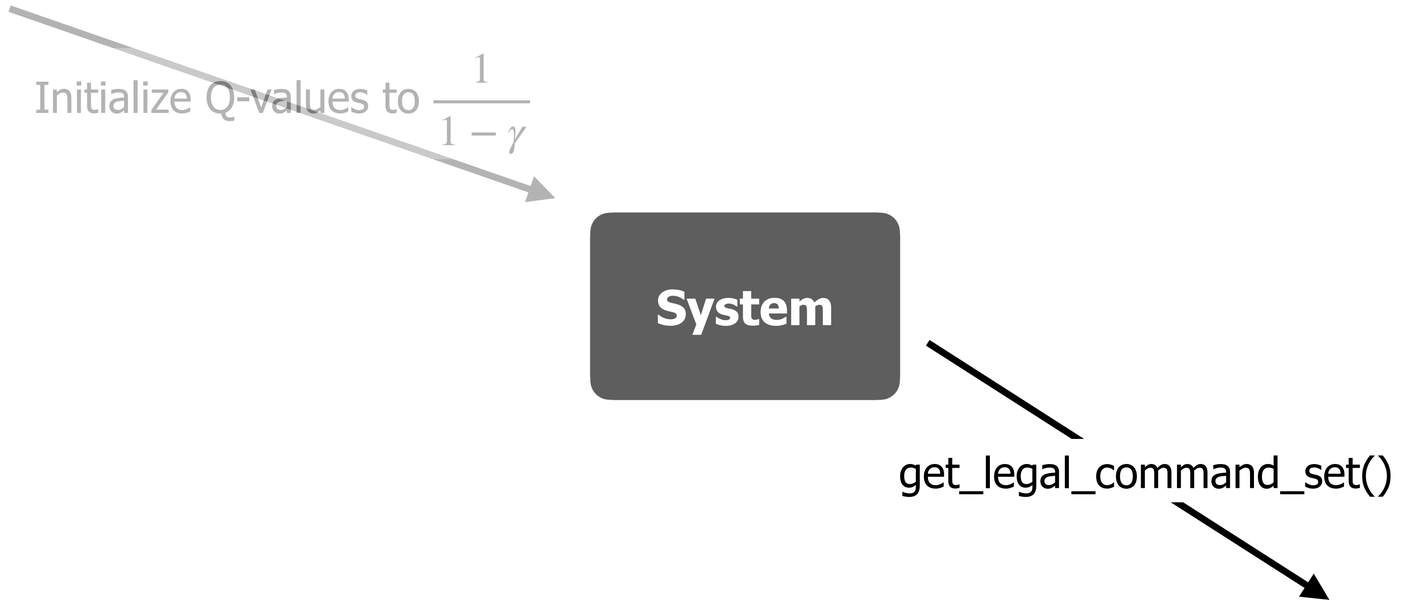


Algorithm (Initialization)

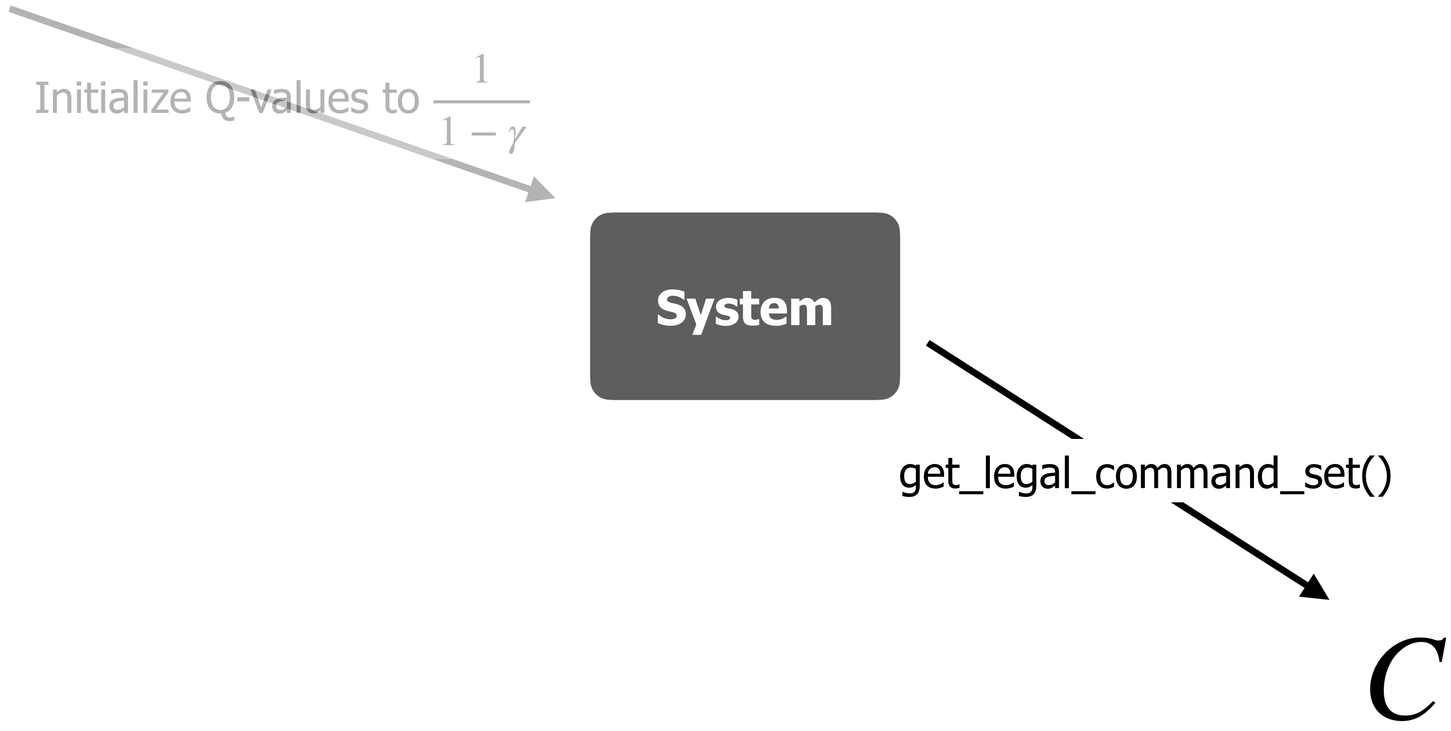
Algorithm (Initialization)



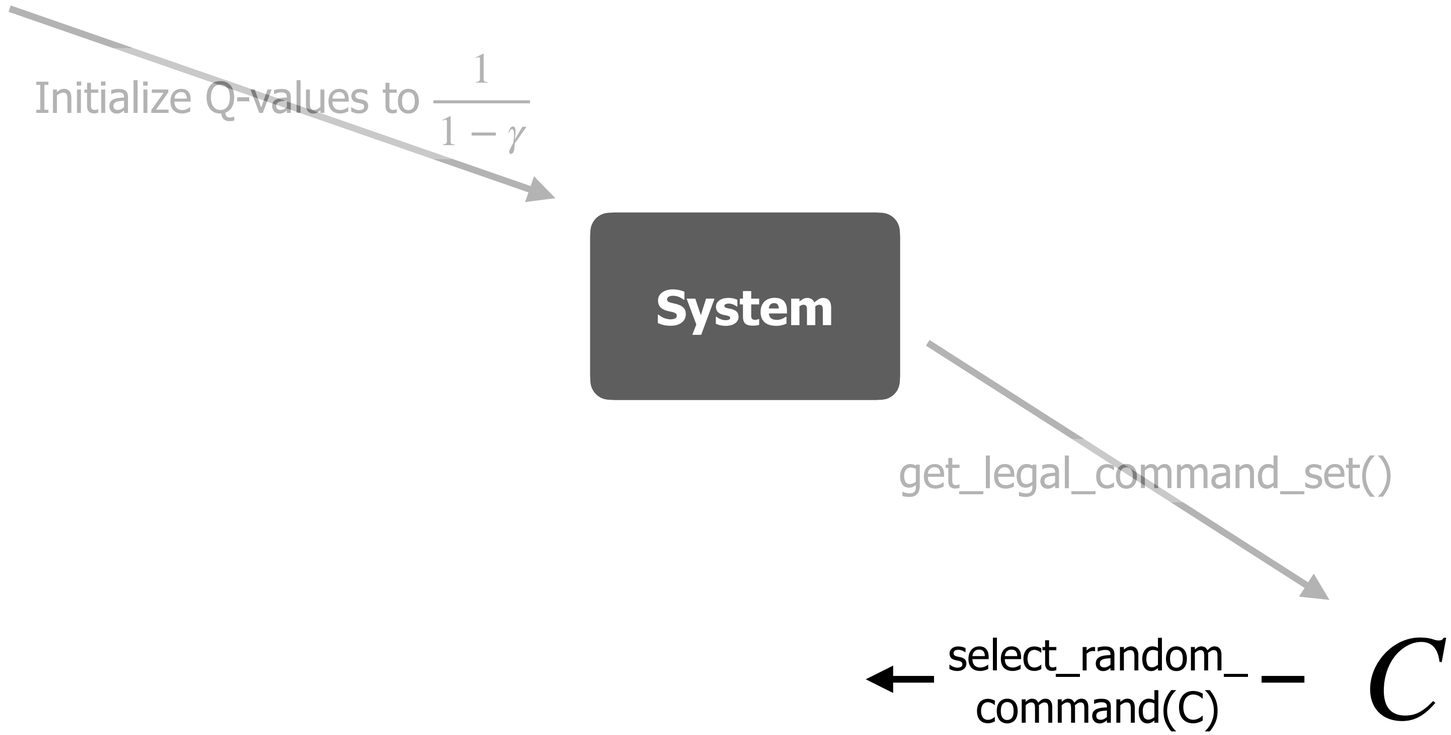
Algorithm (Initialization)



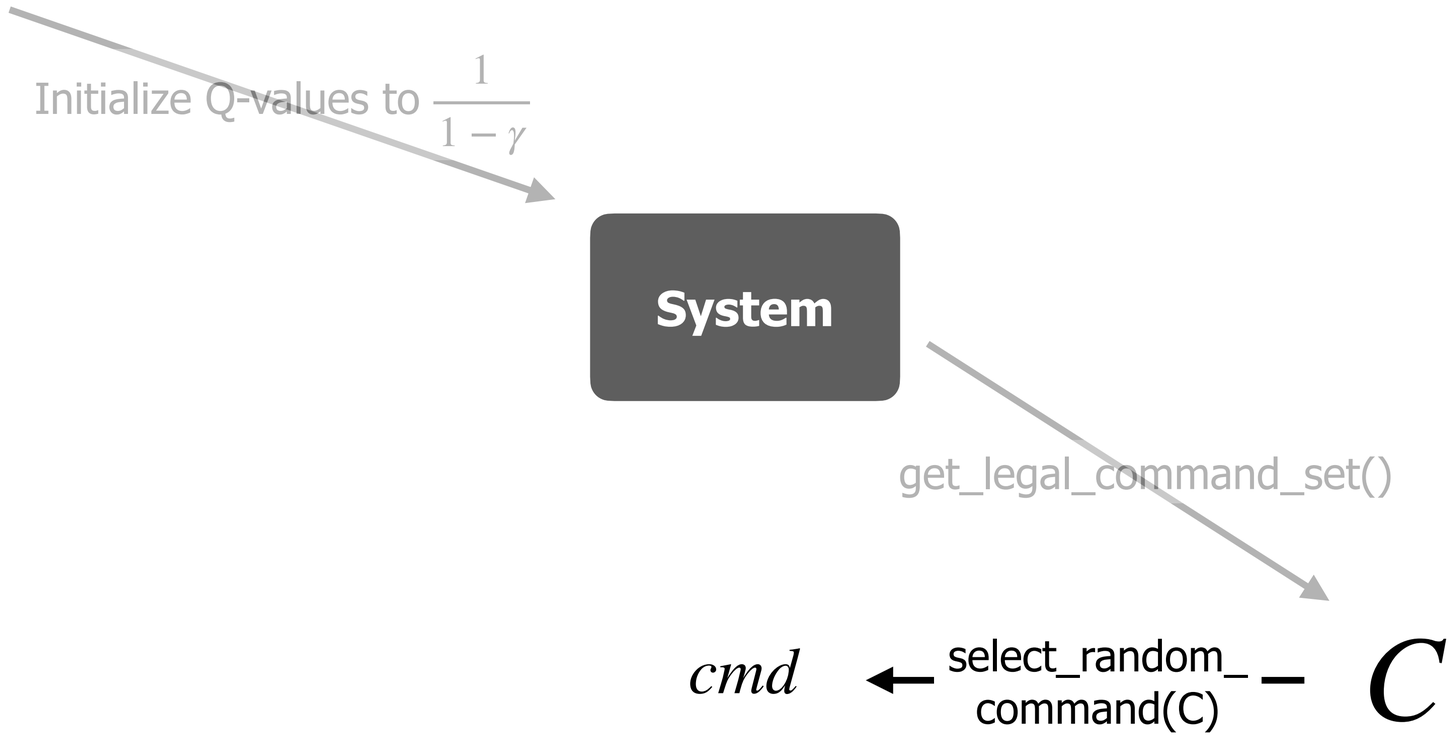
Algorithm (Initialization)



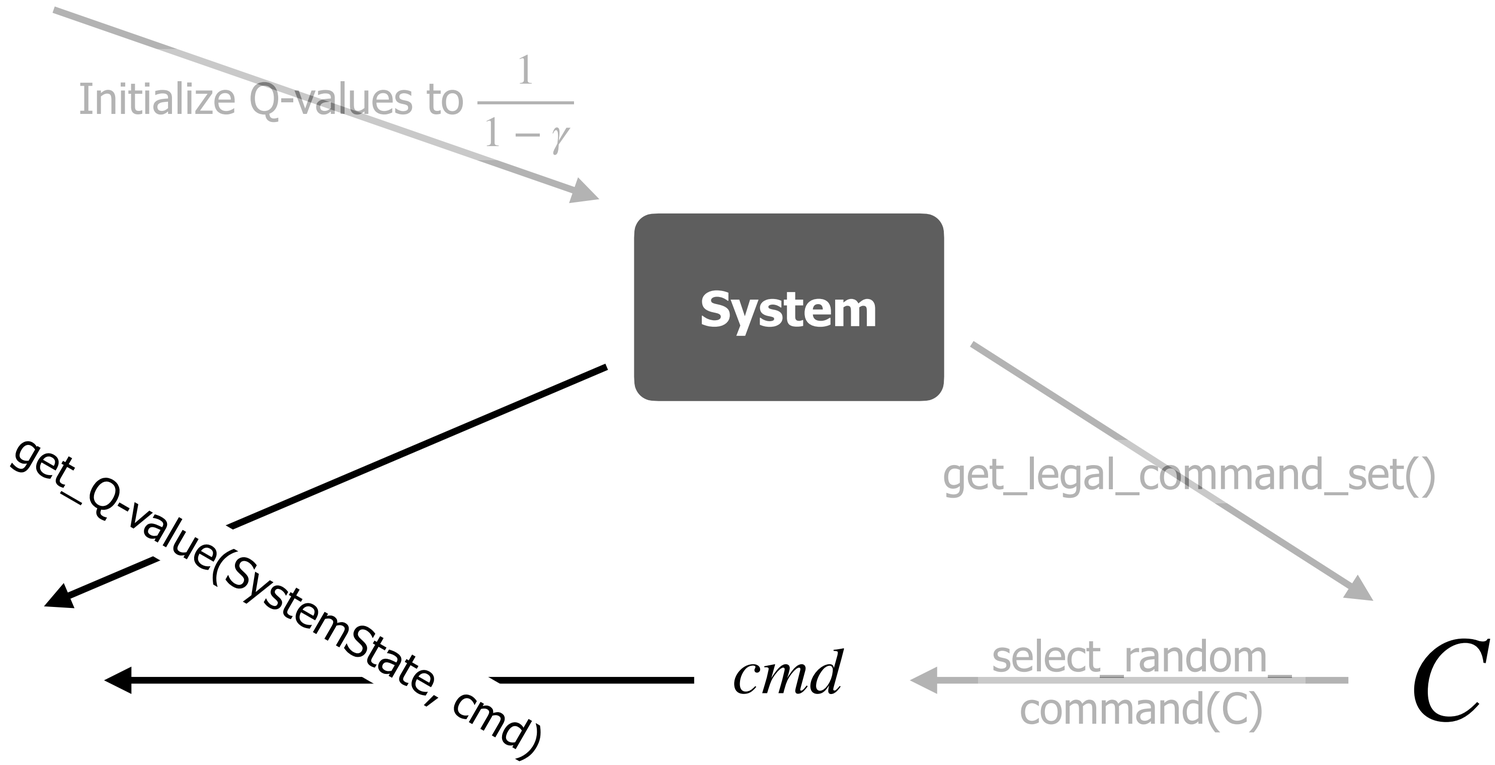
Algorithm (Initialization)



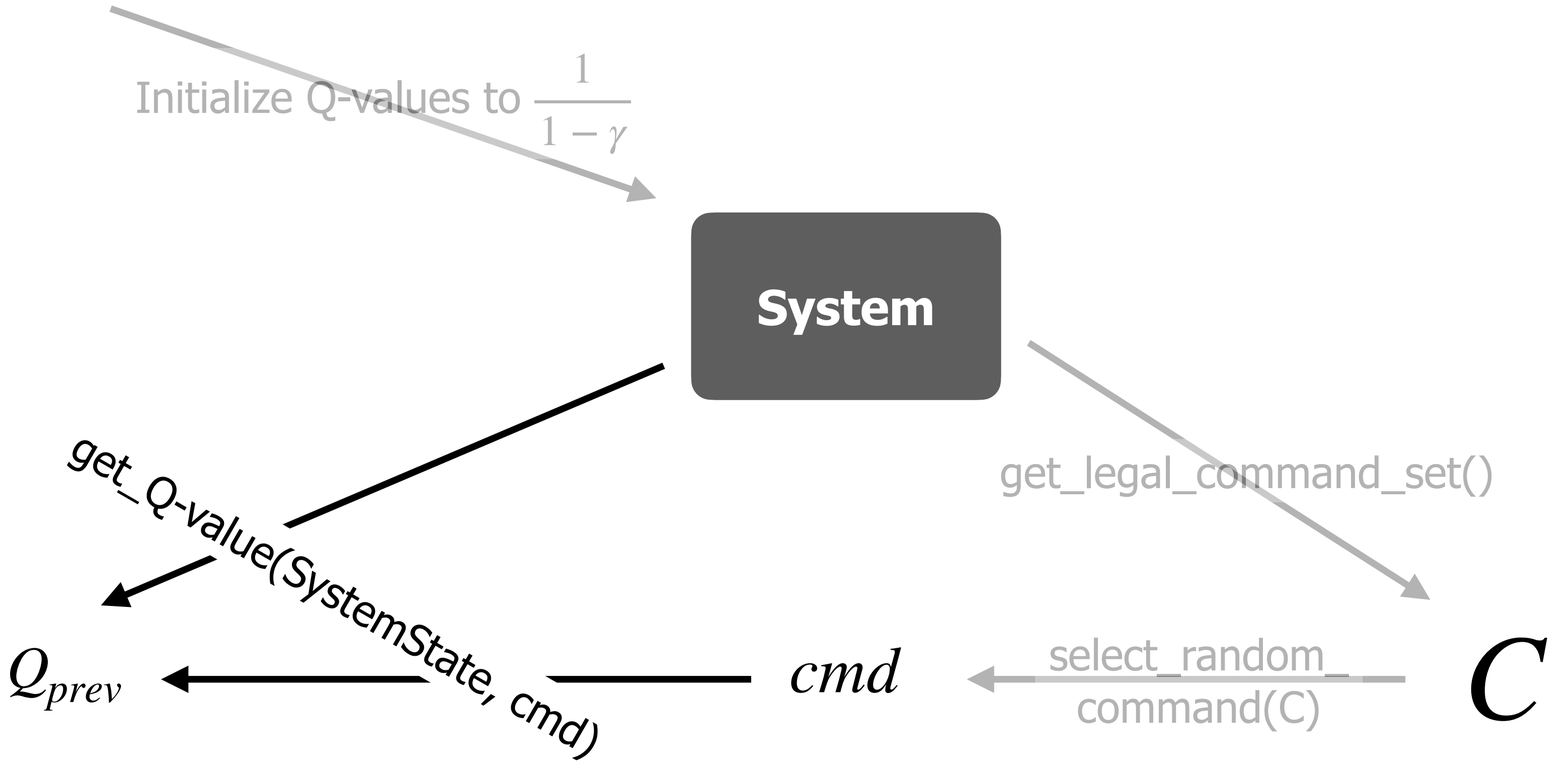
Algorithm (Initialization)



Algorithm (Initialization)



Algorithm (Initialization)



Algorithm

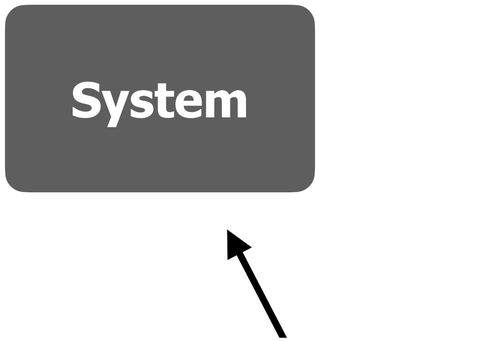
for all DRAM cycles

System

Q_{prev} cmd C

Algorithm

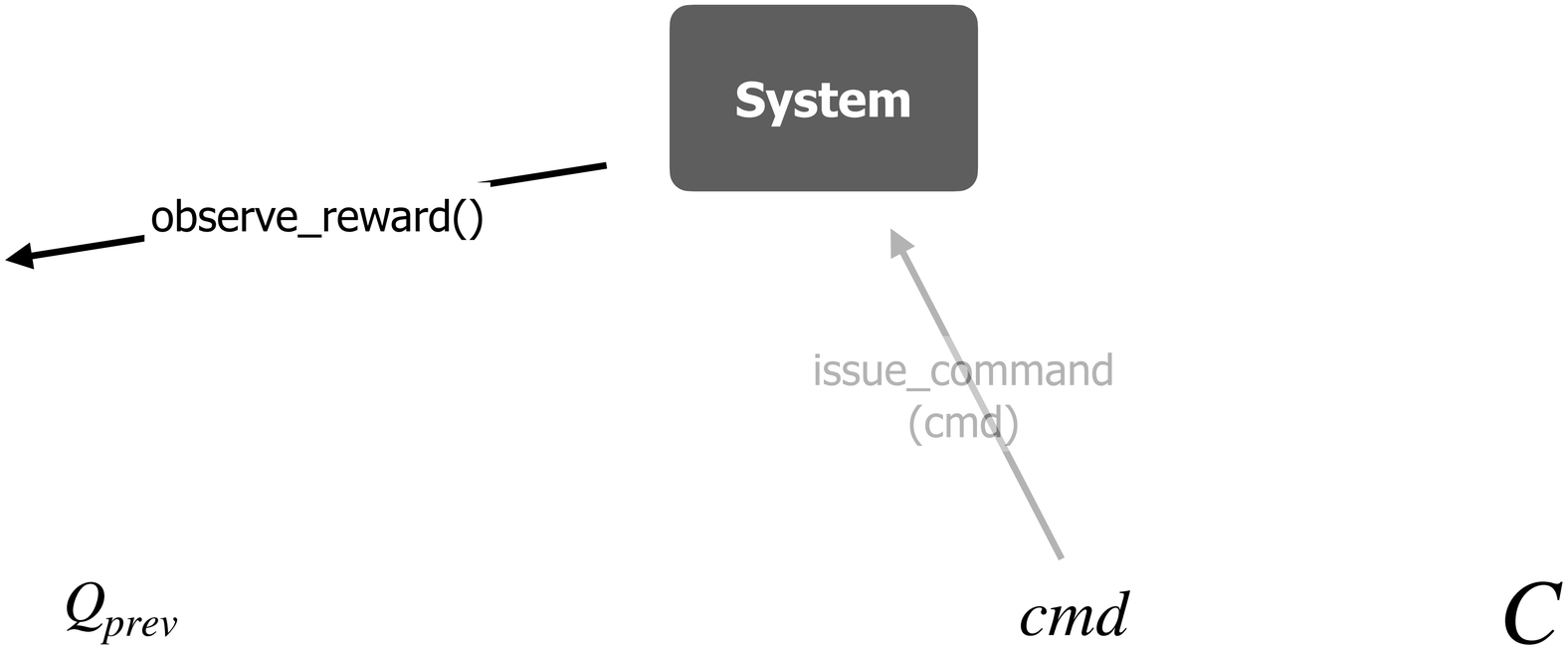
for all DRAM cycles



Q_{prev} cmd C

Algorithm

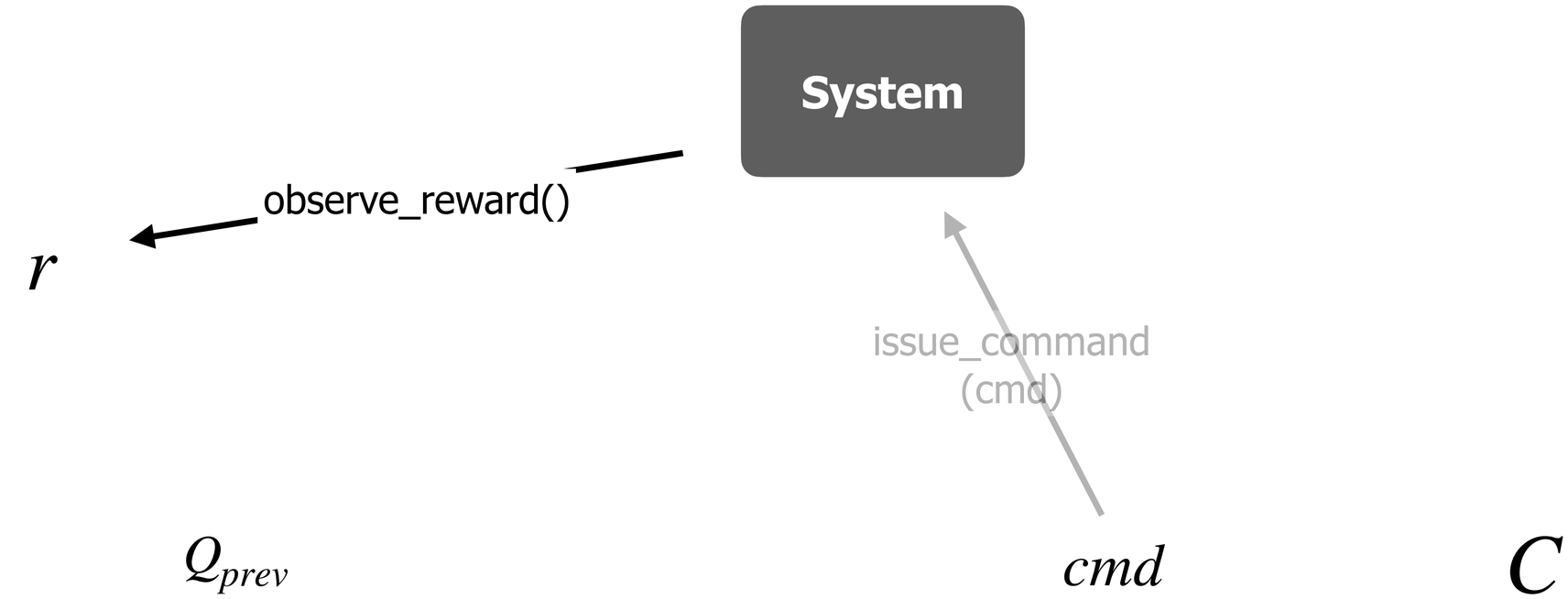
for all DRAM cycles



Q_{prev} cmd C

Algorithm

for all DRAM cycles



r

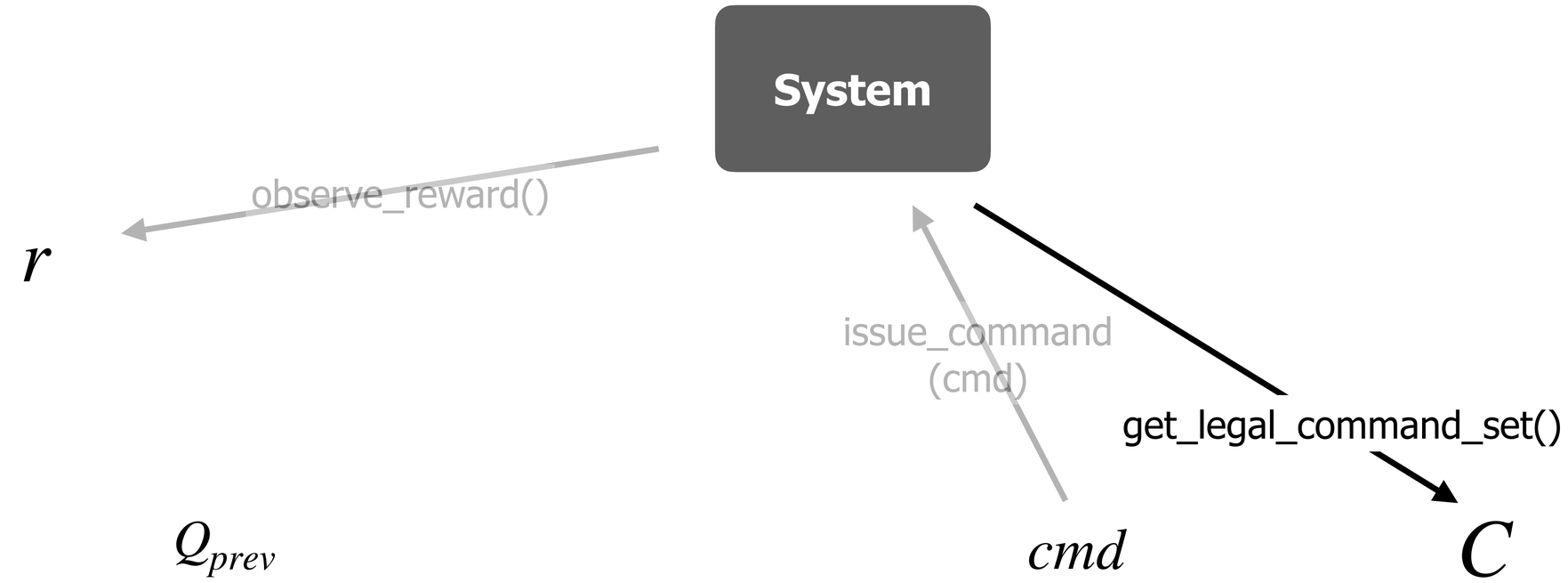
Q_{prev}

cmd

C

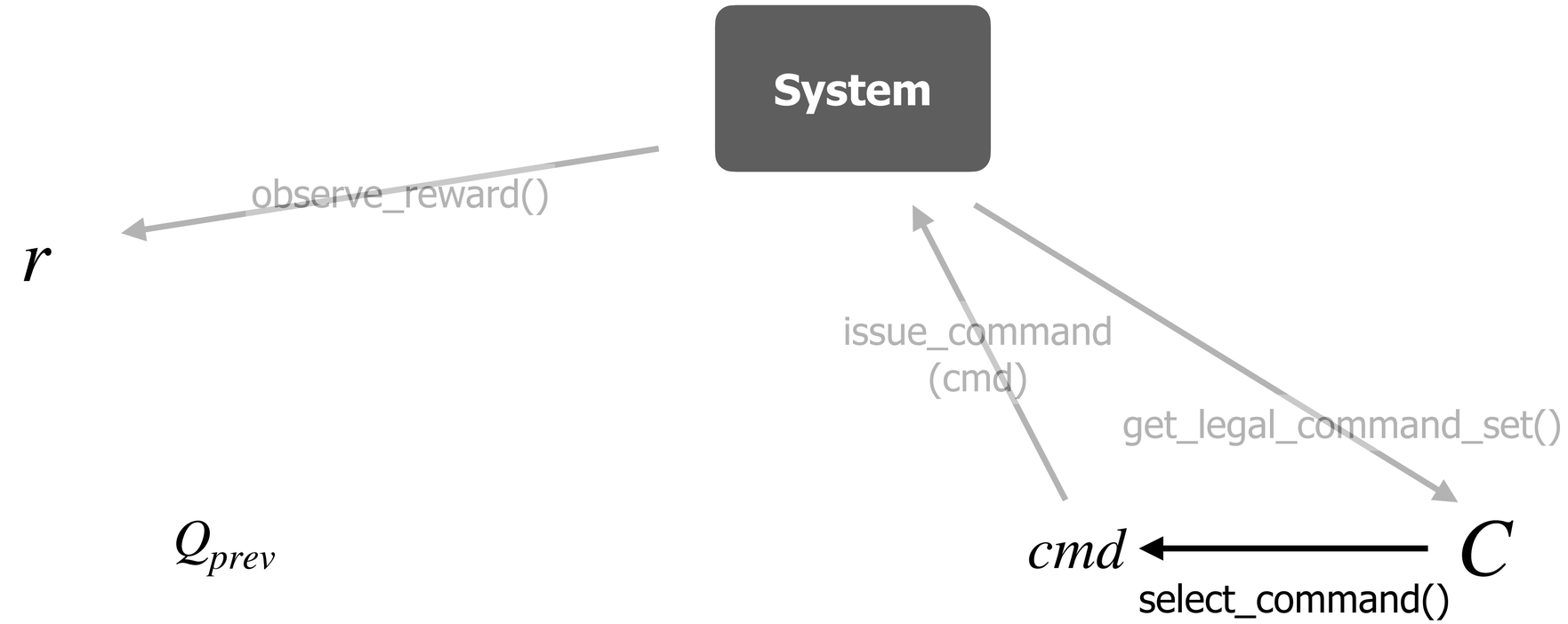
Algorithm

for all DRAM cycles



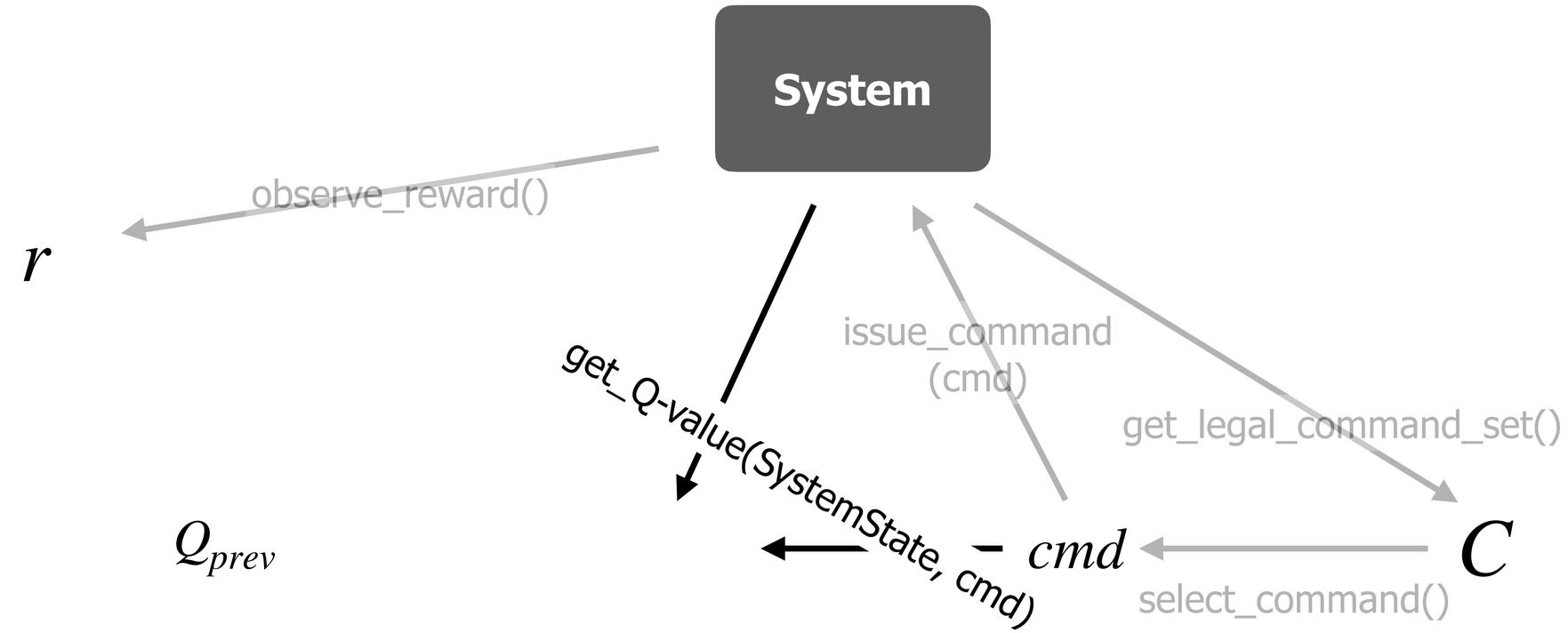
Algorithm

for all DRAM cycles



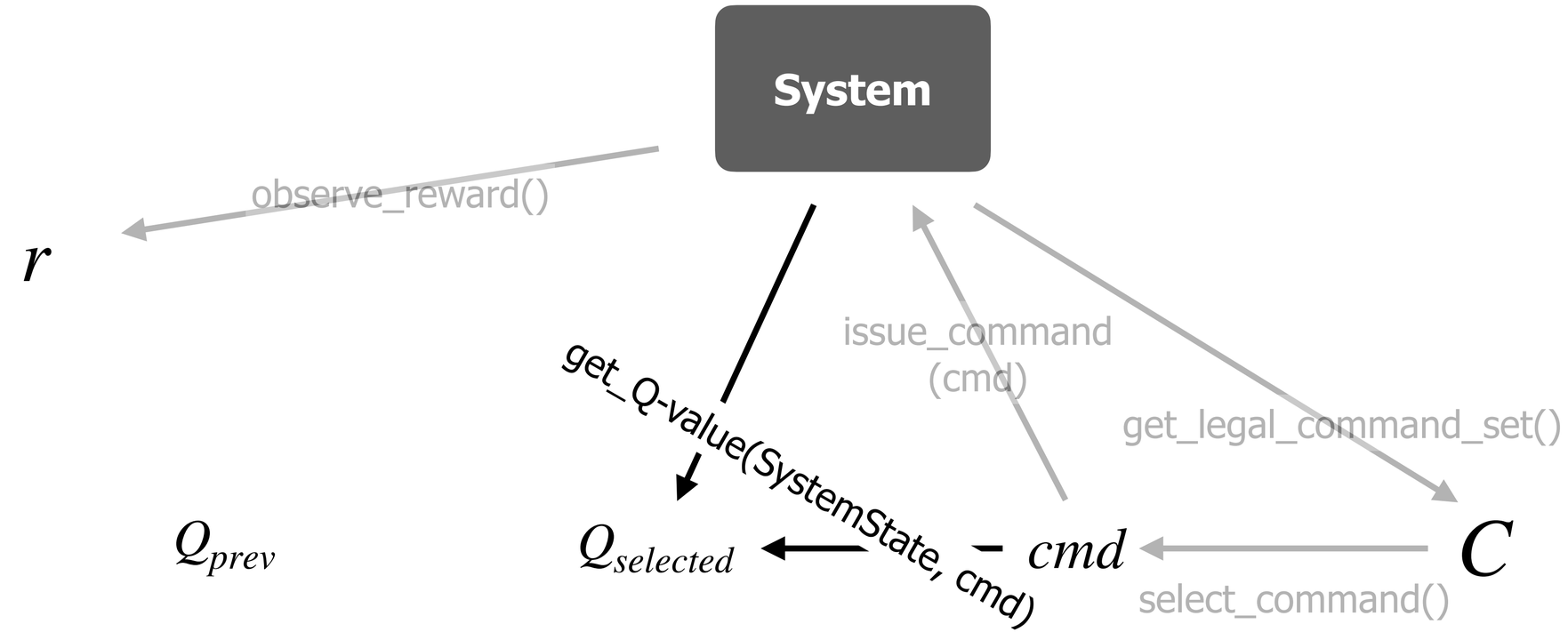
Algorithm

for all DRAM cycles



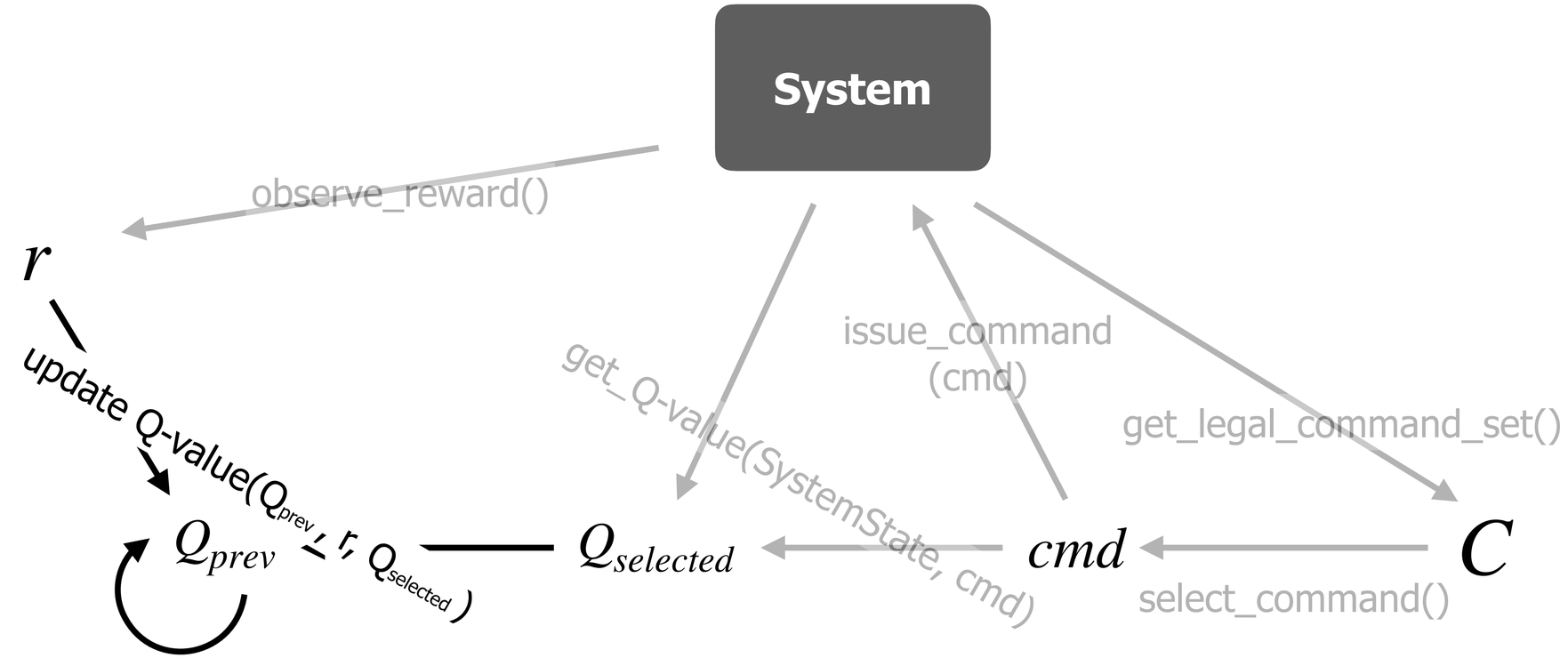
Algorithm

for all DRAM cycles

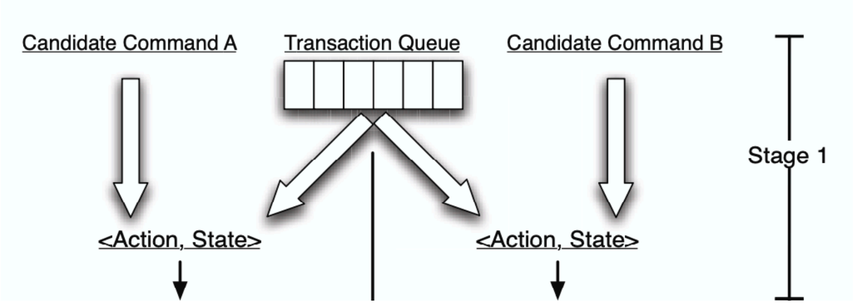


Algorithm

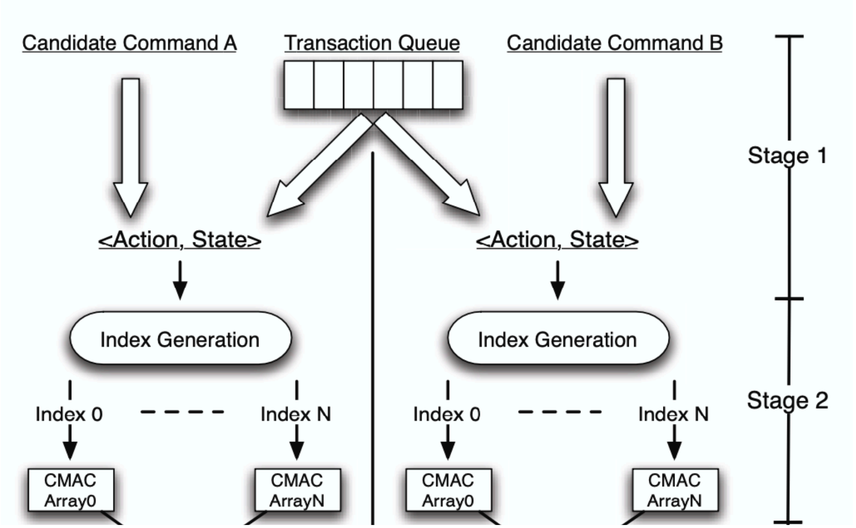
for all DRAM cycles



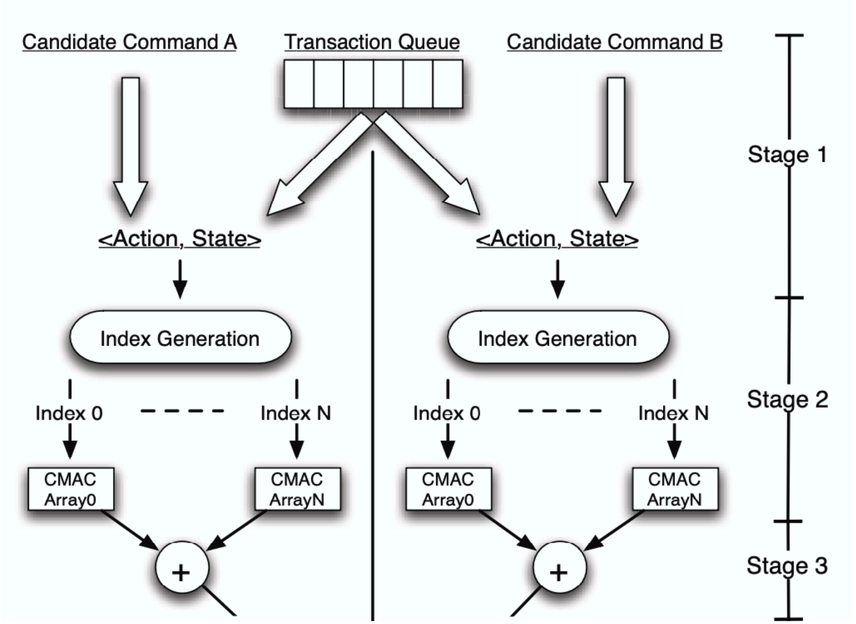
Implementation



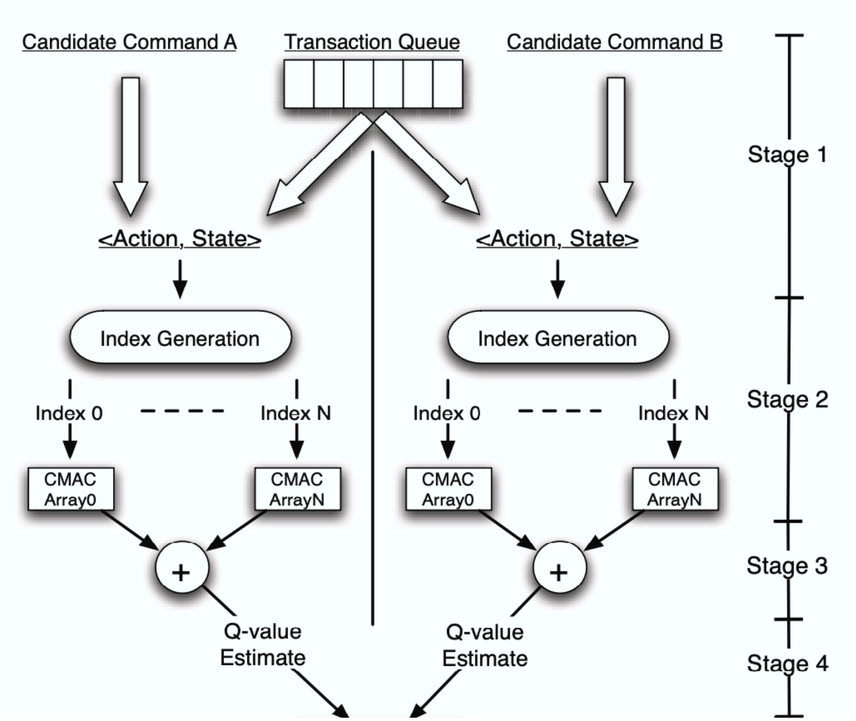
Implementation



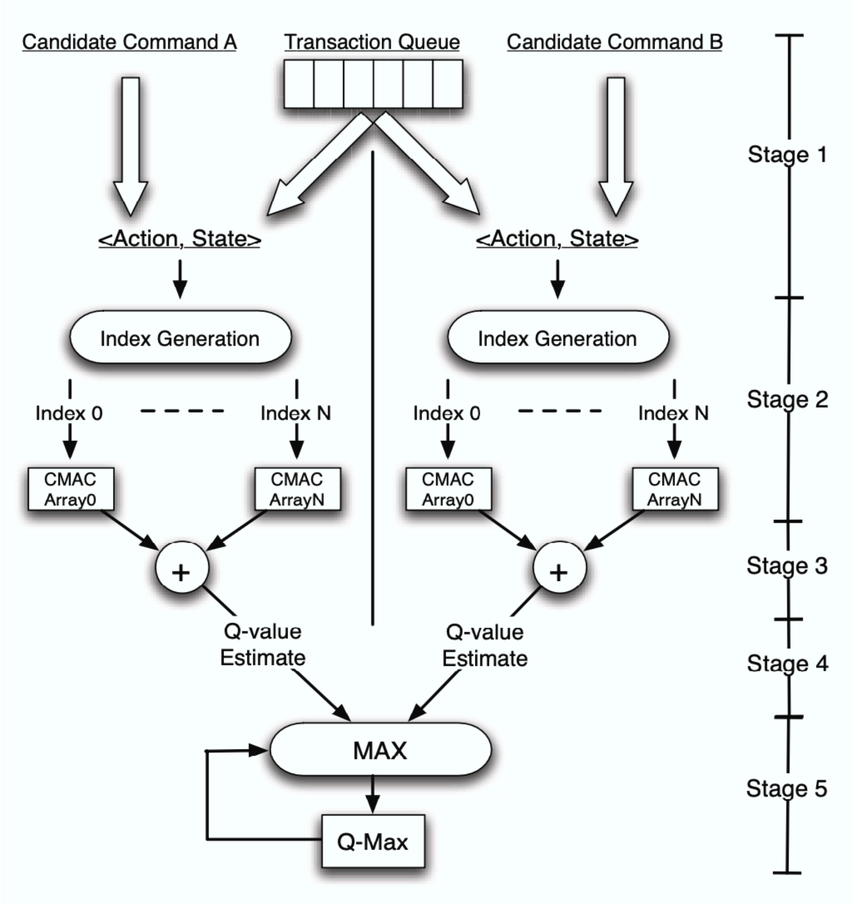
Implementation



Implementation



Implementation



Implementation

Implementation

Does not pose a large processing overhead

Implementation

Does not pose a large processing overhead

The 5 stages run below DRAM cycle time

Implementation

Does not pose a large processing overhead

The 5 stages run below DRAM cycle time

Does not create additional latencies to any instruction it executes

Implementation

Does not pose a large processing overhead

The 5 stages run below DRAM cycle time

Does not create additional latencies to any instruction it executes

Storage Overhead = 32 kB

Problem & Goal

Key Ideas

Novelty

Mechanisms & Implementation

Evaluation & Results

Main Takeaways

Experimental Setup

Experimental Setup

Comparison: FR-FCFS, in-order memory controller, optimistic scheduler with 100% peak DRAM throughput

Experimental Setup

Comparison: FR-FCFS, in-order memory controller, optimistic scheduler with 100% peak DRAM throughput

Benchmarking Workloads: Mix of scalable parallel scientific applications (from the SPLASH-2 suite, SPEC OpenMP suite, and parallel NAS benchmarks) and a parallelized data mining application (SCALPARC from Nu-MineBench)

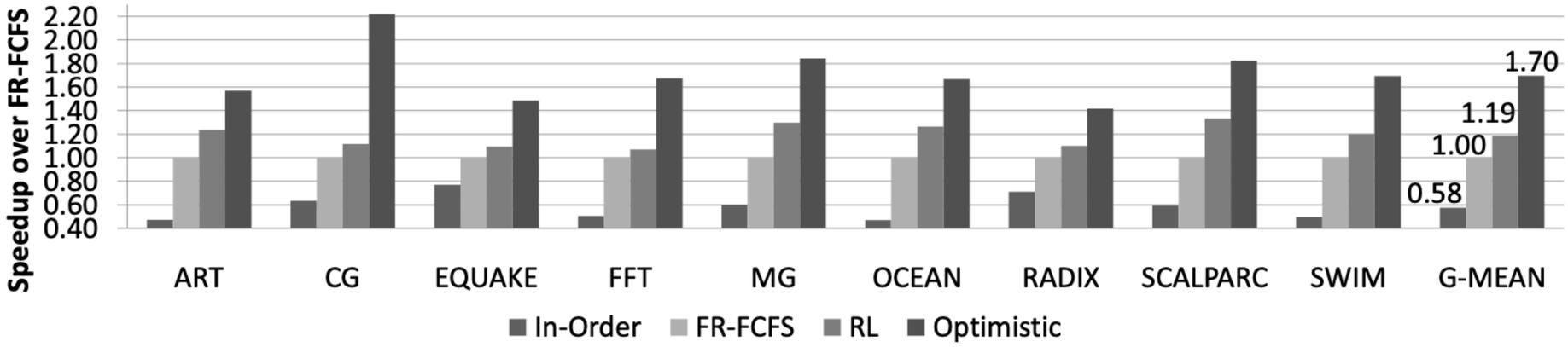
Experimental Setup

Comparison: FR-FCFS, in-order memory controller, optimistic scheduler with 100% peak DRAM throughput

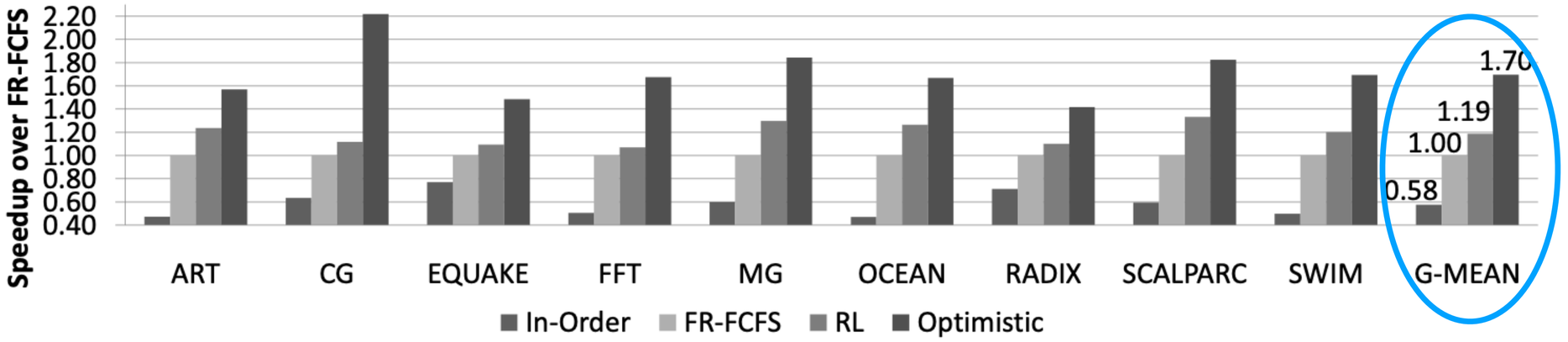
Benchmarking Workloads: Mix of scalable parallel scientific applications (from the SPLASH-2 suite, SPEC OpenMP suite, and parallel NAS benchmarks) and a parallelized data mining application (SCALPARC from Nu-MineBench)

The parallel workloads were simulated on a CMP with four two-way simultaneously multithreaded (SMT) cores, 4MB of L2 cache, and a DDR2-800 memory system

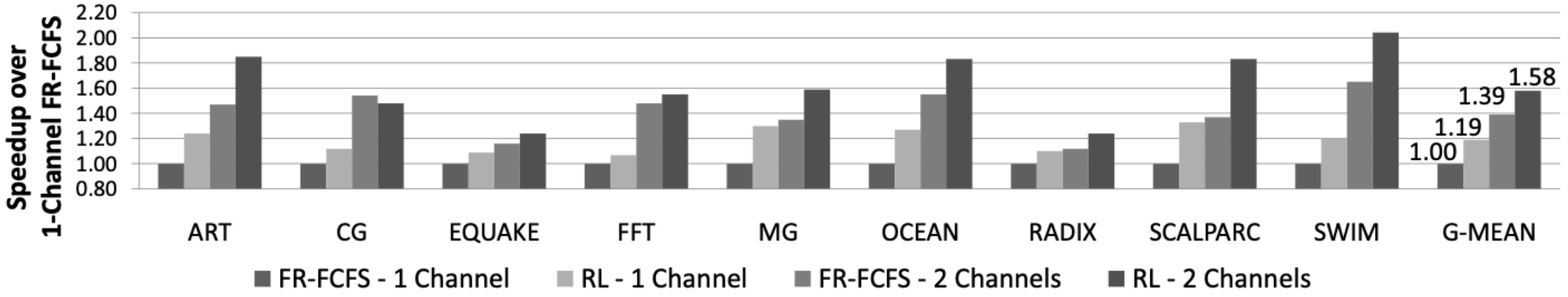
Evaluation: Speedup



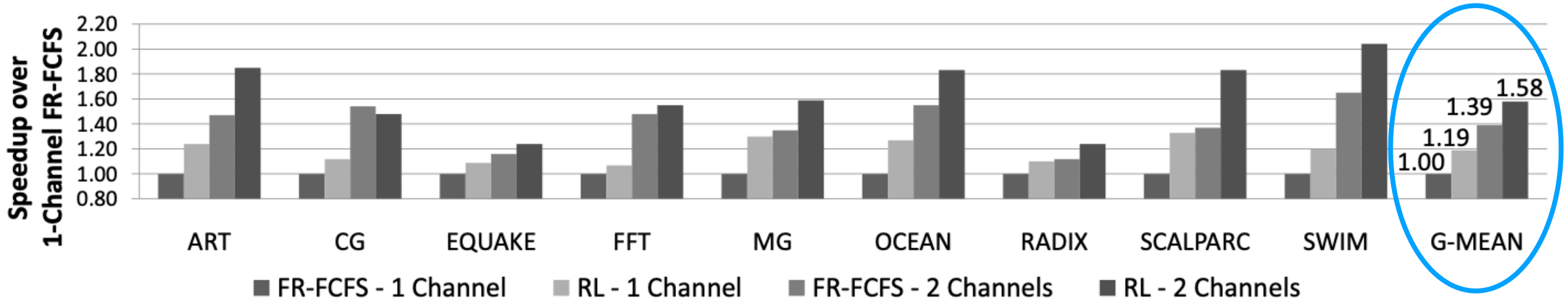
Evaluation: Speedup



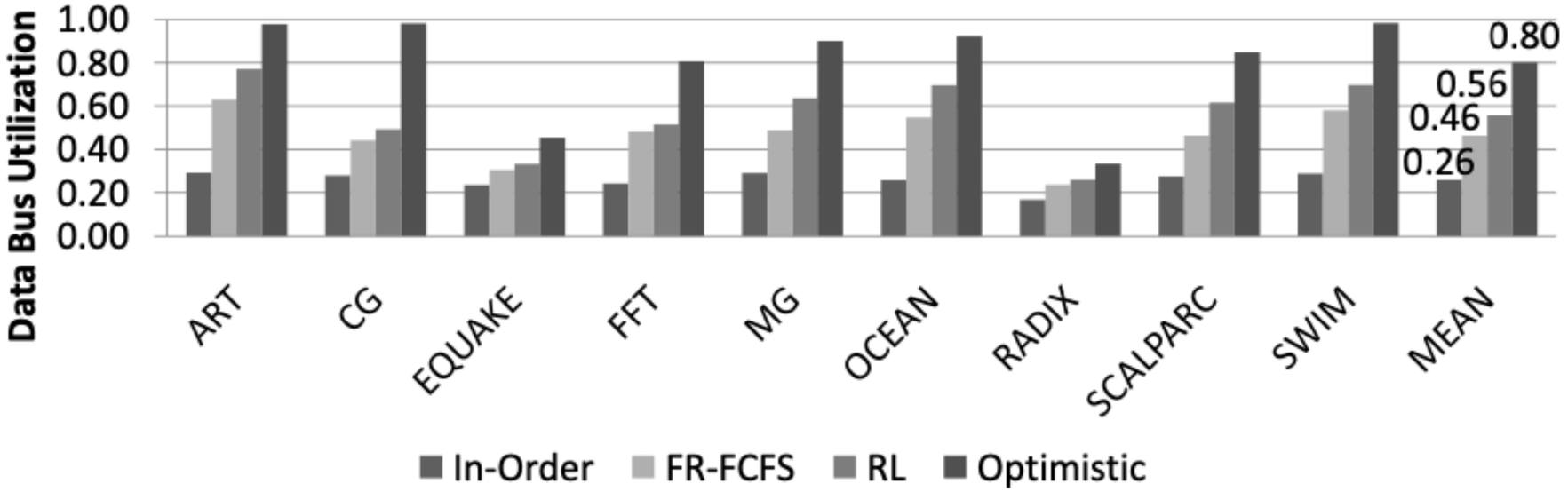
Evaluation: Speedup



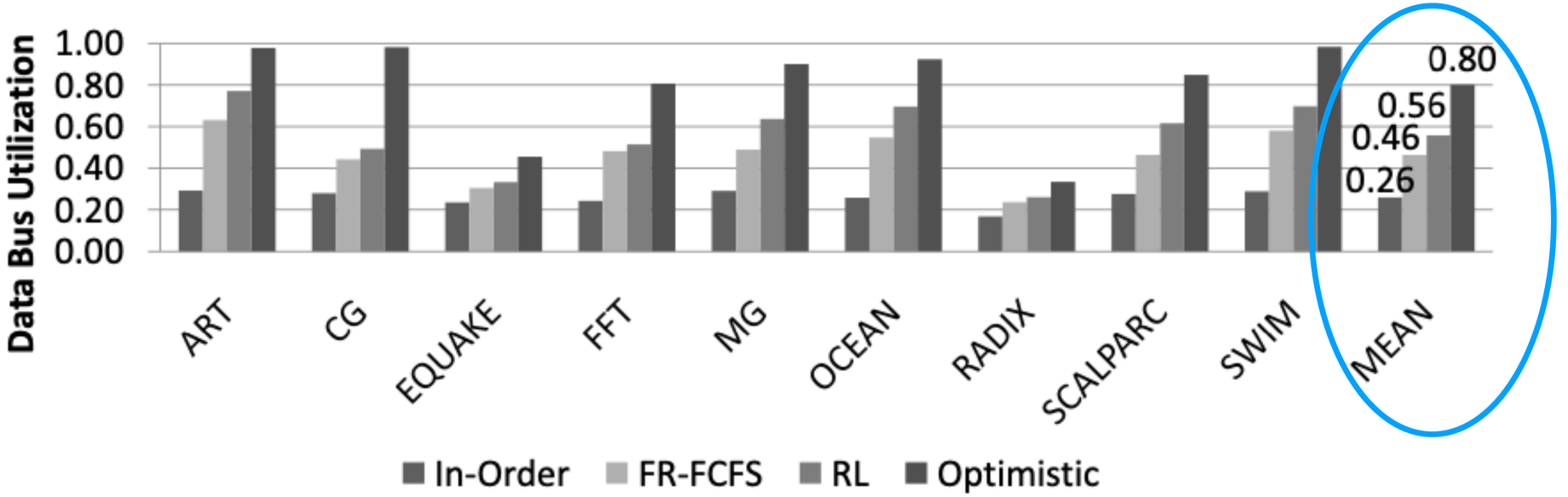
Evaluation: Speedup



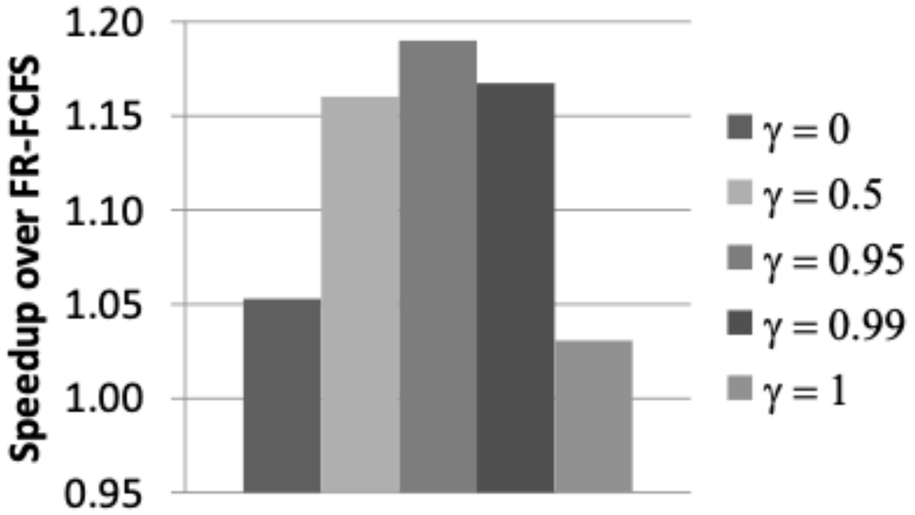
Evaluation: Data Bus Utilization



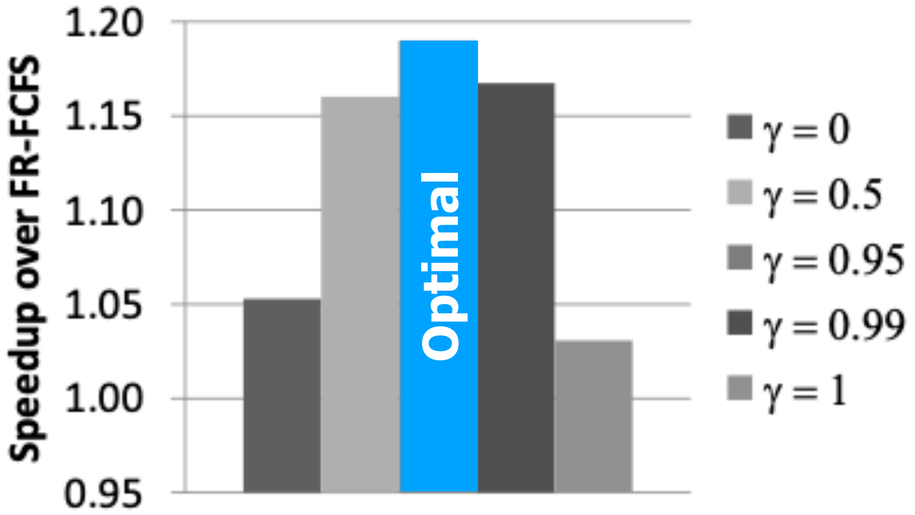
Evaluation: Data Bus Utilization



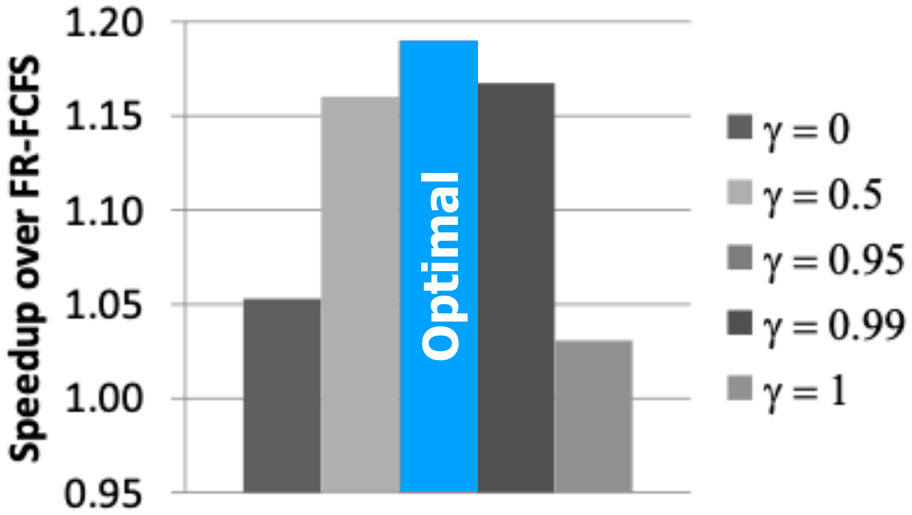
Evaluation: Sensitivity Parameters



Evaluation: Sensitivity Parameters

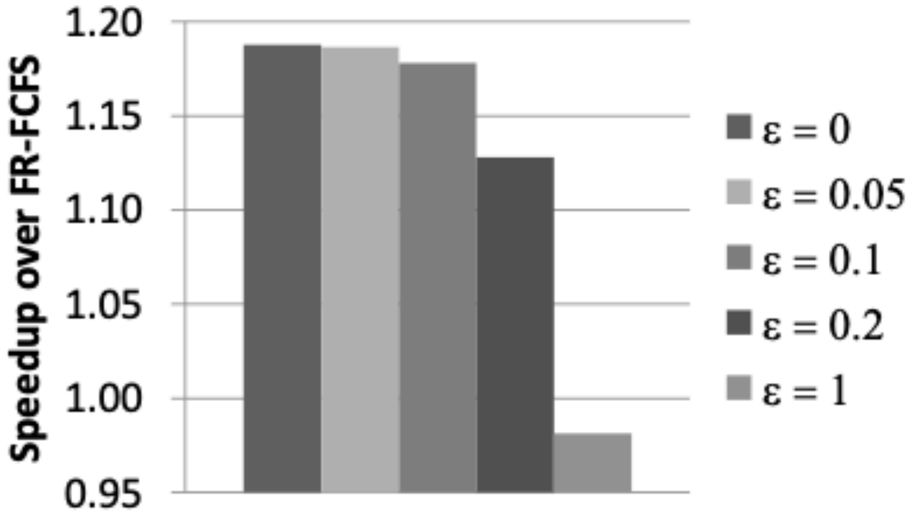


Evaluation: Sensitivity Parameters



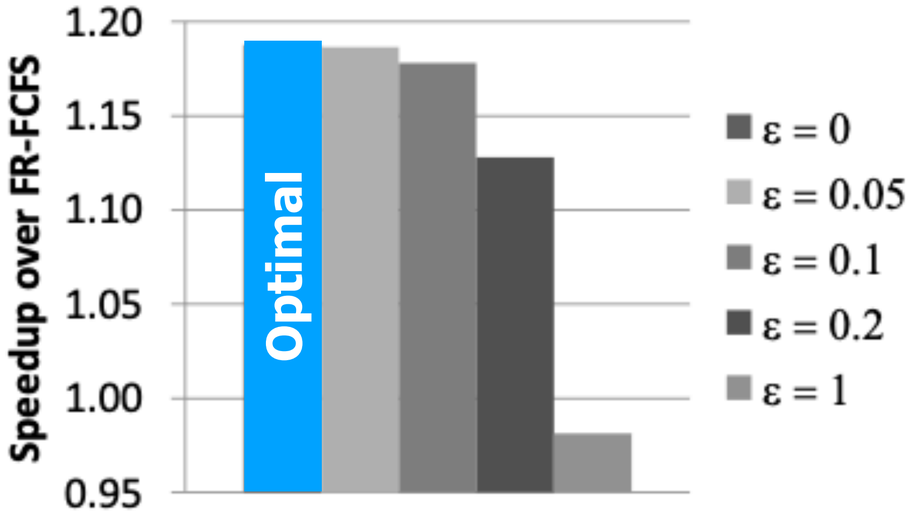
$$Q(s_{prev}, a_{prev}) \leftarrow (1-\alpha)Q(s_{prev}, a_{prev}) + \alpha[r + \gamma Q(s_{current}, a_{current})]$$

Evaluation: Sensitivity Parameters



ϵ decides the amount of random commands chosen

Evaluation: Sensitivity Parameters



ϵ decides the amount of random commands chosen

Evaluation Summary

Evaluation Summary

RL-based memory scheduler improves performance over state-of-the-art memory scheduling policies by 19% on average

Evaluation Summary

RL-based memory scheduler improves performance over state-of-the-art memory scheduling policies by 19% on average

Improvements are seen for **all** tested workloads and architectures

Evaluation Summary

RL-based memory scheduler improves performance over state-of-the-art memory scheduling policies by 19% on average

Improvements are seen for **all** tested workloads and architectures

Shows a promising way of improving memory schedulers

Problem & Goal

Key Ideas

Novelty

Mechanisms & Implementation

Evaluation & Results

Main Takeways

Executive Summary

Executive Summary

Problem: Conventional memory controllers deliver relatively low performance and off-chip bandwidth presents an impediment to Chip Multiprocessor (CMP) scalability, also memory controllers are difficult to optimize with a fixed, rigid policy.

Executive Summary

Problem: Conventional memory controllers deliver relatively low performance and off-chip bandwidth presents an impediment to Chip Multiprocessor (CMP) scalability, also memory controllers are difficult to optimize with a fixed, rigid policy.

Goal: Improve off-chip bandwidth by utilizing a dynamic, self-optimizing memory controller

Executive Summary

Problem: Conventional memory controllers deliver relatively low performance and off-chip bandwidth presents an impediment to Chip Multiprocessor (CMP) scalability, also memory controllers are difficult to optimize with a fixed, rigid policy.

Goal: Improve off-chip bandwidth by utilizing a dynamic, self-optimizing memory controller

Key Ideas: Transforming the memory controller to a reinforcement learning (RL) agent, which finds the best scheduling policy for long-term performance

Executive Summary

Problem: Conventional memory controllers deliver relatively low performance and off-chip bandwidth presents an impediment to Chip Multiprocessor (CMP) scalability, also memory controllers are difficult to optimize with a fixed, rigid policy.

Goal: Improve off-chip bandwidth by utilizing a dynamic, self-optimizing memory controller

Key Ideas: Transforming the memory controller to a reinforcement learning (RL) agent, which finds the best scheduling policy for long-term performance

Evaluation: Test the new memory controller in different environments against the state-of-the-art best average access scheduling policy FR-FCFS

Executive Summary

Problem: Conventional memory controllers deliver relatively low performance and off-chip bandwidth presents an impediment to Chip Multiprocessor (CMP) scalability, also memory controllers are difficult to optimize with a fixed, rigid policy.

Goal: Improve off-chip bandwidth by utilizing a dynamic, self-optimizing memory controller

Key Ideas: Transforming the memory controller to a reinforcement learning (RL) agent, which finds the best scheduling policy for long-term performance

Evaluation: Test the new memory controller in different environments against the state-of-the-art best average access scheduling policy FR-FCFS

Results: An RL-based memory controller improves performance of parallel applications on a 4-core CMP by 19% on average and DRAM bandwidth utilization by 22% compared to FR-FCFS

Executive Summary

Motivation: Efficiently utilizing off-chip bandwidth is critical in the design of Chip Multiprocessors (CMPs)

Problem: Conventional memory controllers deliver relatively low performance and off-chip bandwidth presents an impediment to Chip Multiprocessor (CMP) scalability, also memory controllers are difficult to optimize with a fixed, rigid policy.

Goal: Improve off-chip bandwidth by utilizing a dynamic, self-optimizing memory controller

Key Ideas: Transforming the memory controller to a reinforcement learning (RL) agent, which finds the best scheduling policy for long-term performance

Evaluation: Test the new memory controller in different environments against the state-of-the-art best average access scheduling policy FR-FCFS

Results: An RL-based memory controller improves performance of parallel applications on a 4-core CMP by 19% on average and DRAM bandwidth utilization by 22% compared to FR-FCFS

Main Takeaways

Main Takeaways

An RL-based self-optimizing memory scheduler adapts to its environment and shows promising results compared to state-of-the-art technologies

Main Takeaways

An RL-based self-optimizing memory scheduler adapts to its environment and shows promising results compared to state-of-the-art technologies

It reduces human design effort

Main Takeaways

An RL-based self-optimizing memory scheduler adapts to its environment and shows promising results compared to state-of-the-art technologies

It reduces human design effort

Utilizes the memory bus more efficiently

Main Takeaways

An RL-based self-optimizing memory scheduler adapts to its environment and shows promising results compared to state-of-the-art technologies

It reduces human design effort

Utilizes the memory bus more efficiently

Promising way for future technologies

Critique & Discussion

Strengths

Weaknesses

Thoughts

Discussion

Strengths

Weaknesses

Thoughts

Discussion

Strengths

Strengths

1. The paper shows the first application of machine learning to memory controllers

Strengths

1. The paper shows the first application of machine learning to memory controllers

2. Leads to a significant performance improvement in contrast to fixed state-of-the-art memory controllers

Strengths

1. The paper shows the first application of machine learning to memory controllers

2. Leads to a significant performance improvement in contrast to fixed state-of-the-art memory controllers

3. The proposed solution is self-optimizing and dynamic, and can thus improve performance for many different workloads

Strengths

1. The paper shows the first application of machine learning to memory controllers

2. Leads to a significant performance improvement in contrast to fixed state-of-the-art memory controllers

3. The proposed solution is self-optimizing and dynamic, and can thus improve performance for many different workloads

4. Thorough evaluation with many different workloads and comparisons

Strengths

1. The paper shows the first application of machine learning to memory controllers
2. Leads to a significant performance improvement in contrast to fixed state-of-the-art memory controllers
3. The proposed solution is self-optimizing and dynamic, and can thus improve performance for many different workloads
4. Thorough evaluation with many different workloads and comparisons
5. Well written: Clearly explains the mechanism behind the implementation

Strengths

Weaknesses

Thoughts

Discussion

Weaknesses

Weaknesses

1. The paper makes specific assumptions and uses parameters which could be explained more thoroughly for better understanding

Weaknesses

1. The paper makes specific assumptions and uses parameters which could be explained more thoroughly for better understanding

2. The specific implementation/how exactly this is implemented in hardware and its costs are not discussed in a lot of detail

Weaknesses

1. The paper makes specific assumptions and uses parameters which could be explained more thoroughly for better understanding

2. The specific implementation/how exactly this is implemented in hardware and its costs are not discussed in a lot of detail

3. The paper limits itself to a scope and does not fully consider what could be done outside of the assumed constraints

Strengths

Weaknesses

Thoughts

Discussion

Thoughts

Thoughts

Cool and novel idea with really good applicability

Thoughts

Cool and novel idea with really good applicability

Starting grounds of optimizing difficult problems with ML

Thoughts

Cool and novel idea with really good applicability

Starting grounds of optimizing difficult problems with ML

Opens up new possibilities in ML and HW interplay

Thoughts

Cool and novel idea with really good applicability

Starting grounds of optimizing difficult problems with ML

Opens up new possibilities in ML and HW interplay

Encourages the use of ML in places, where dynamic and adaptable solutions are needed

Strengths

Weaknesses

Thoughts

Discussion

Discussion

What are potential reasons for or against implementing this in a real-world system?

Discussion

Are there other applications similar to this, which you can see in other Computer Architecture designs?

Discussion

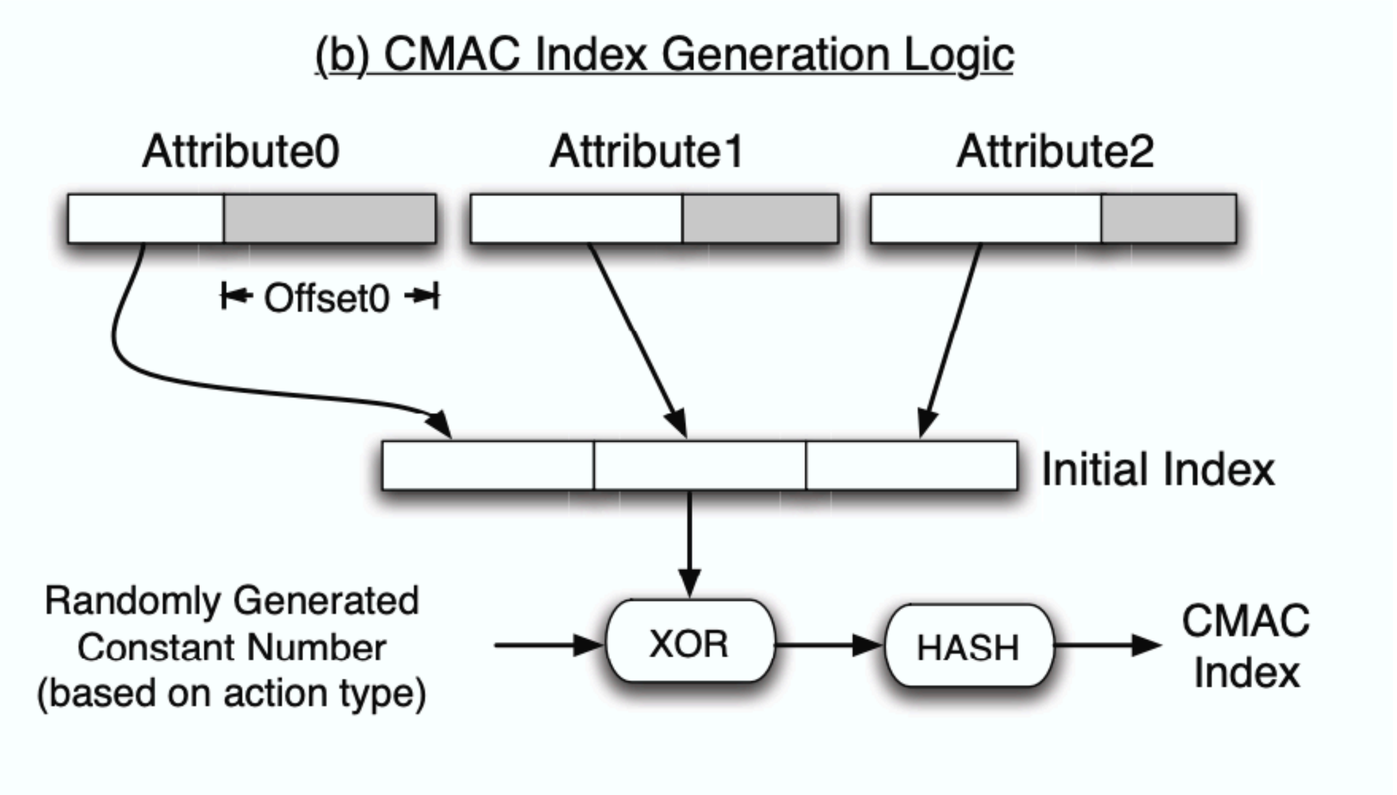
Is this worth exploring further now 13 years after the publishing of this paper?

Discussion

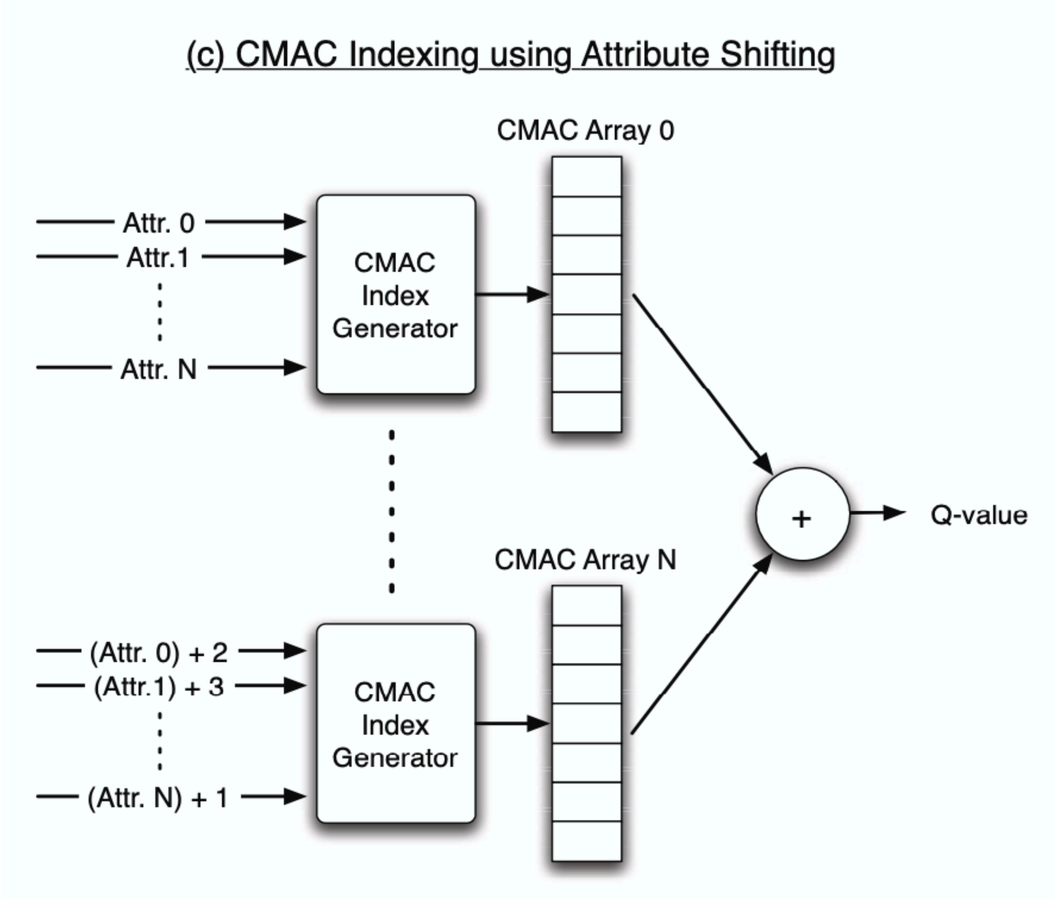
Can you see other criteria, other than only latency overhead, which could also be considered for an implementation of this technology?

Backup sildes

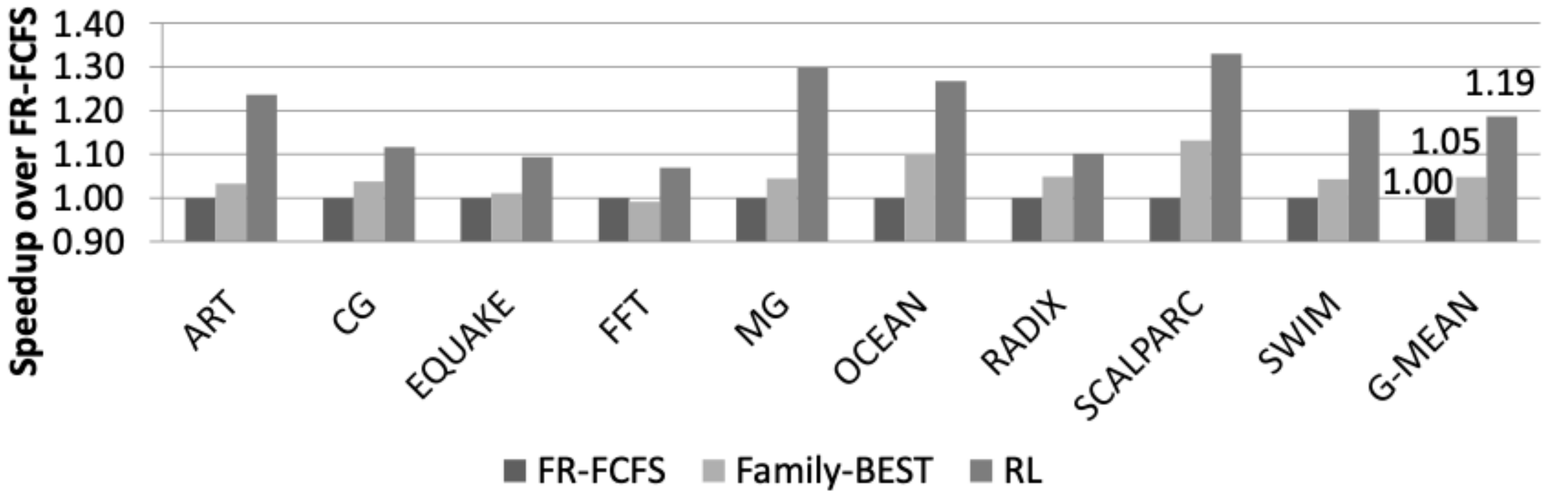
Implementation



Implementation

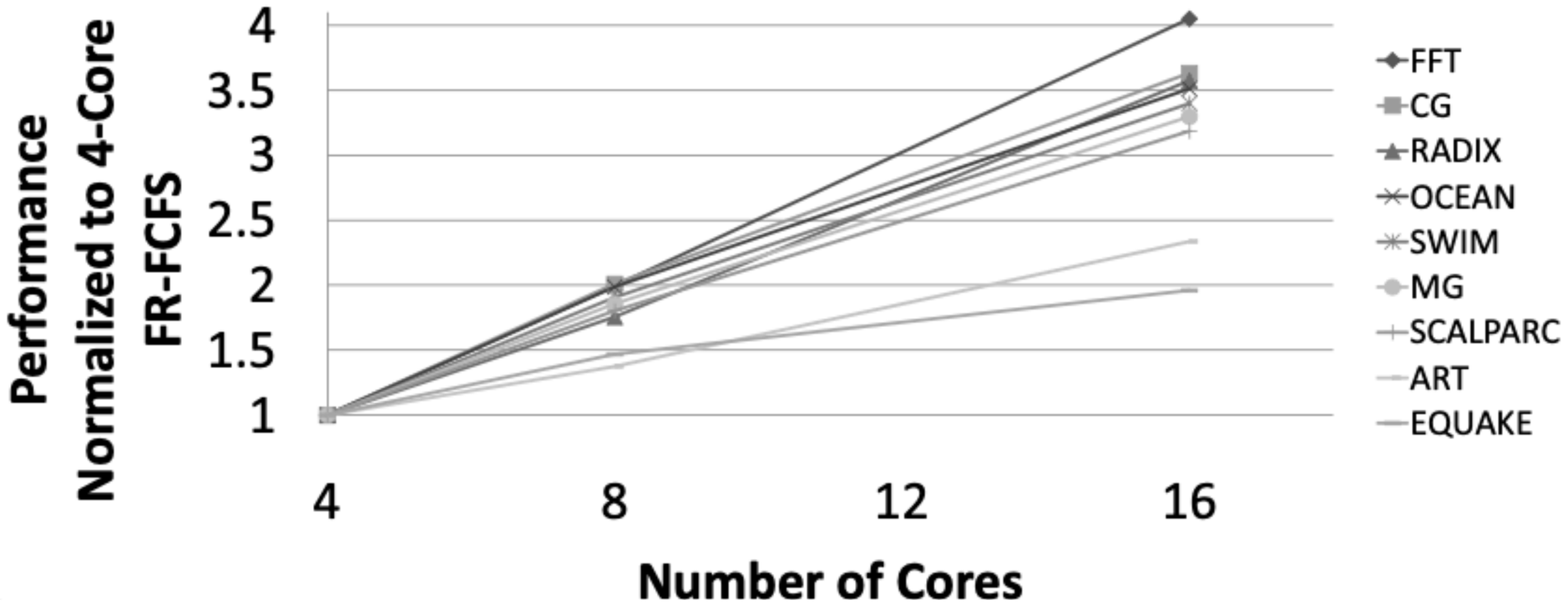


Evaluation: Speedup

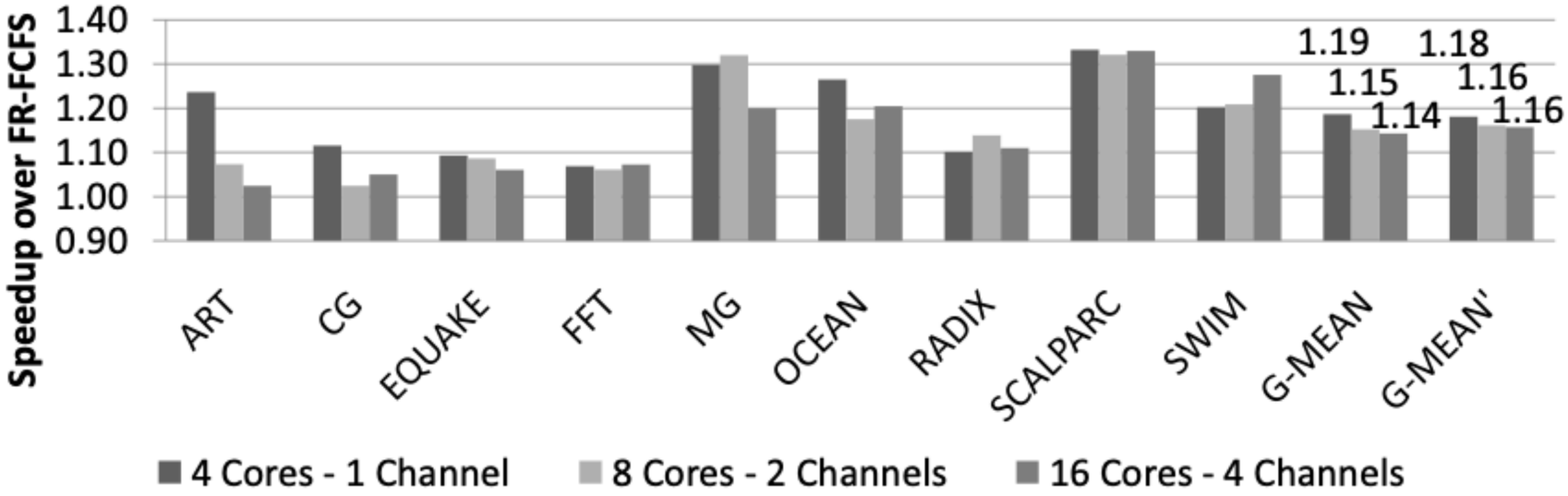


Same information is given to the FR-FCFS memory scheduler

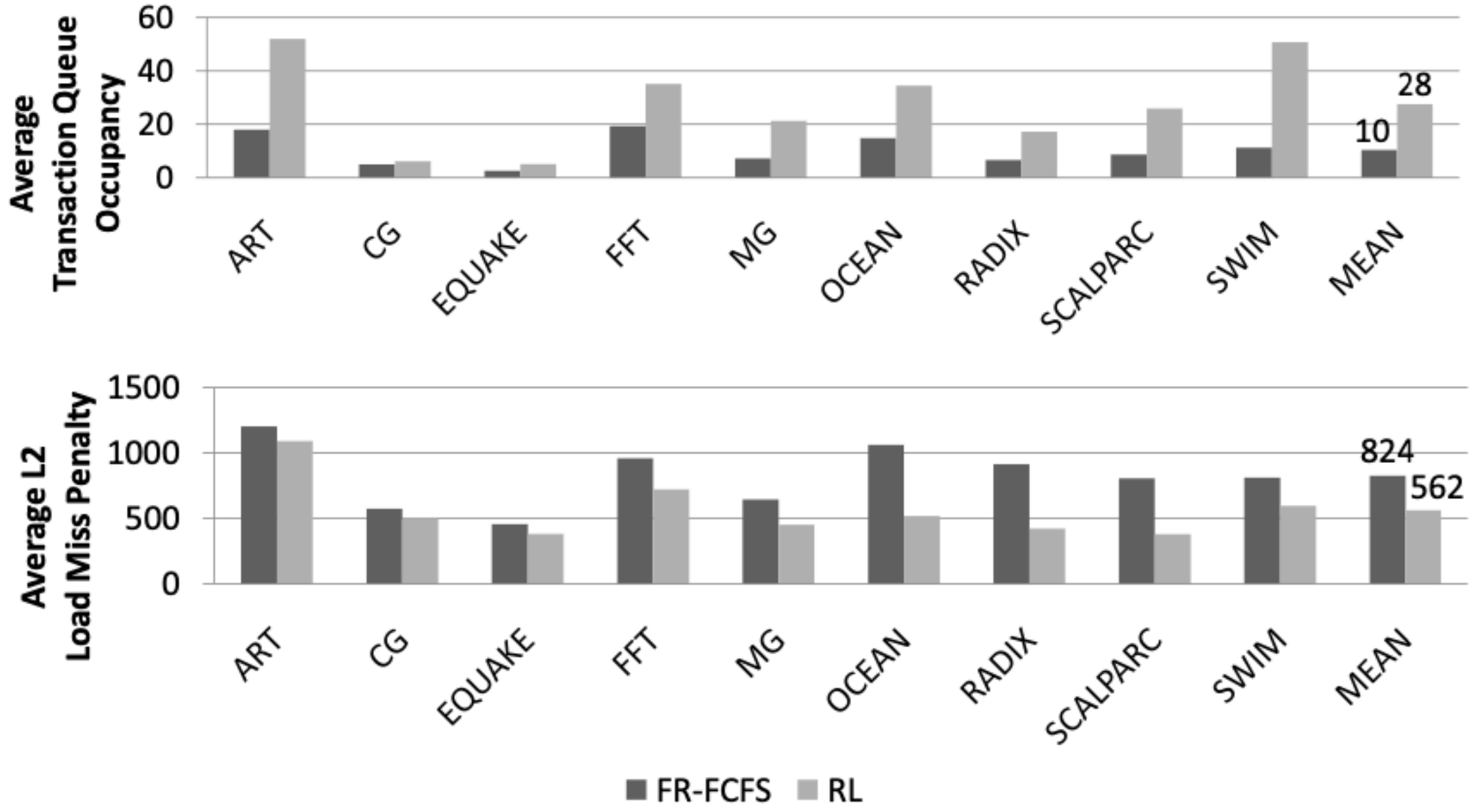
Evaluation: Performance



Evaluation: Speedup



Evaluation: System



Evaluation: static vs. dynamic