Contention and memory controllers

An introduction to (ideas on) tracking multi-core contention in DRAM

Asier Fernández de Lecea
Francisco J. Cazorla

Presentation outline

In this presentation we will be talking about the following:

- First, some theoretical background on memory controllers (MCs) and some DRAM specifics

Presentation outline

In this presentation we will be talking about the following:

- First, some theoretical background on memory controllers (MCs) and some DRAM specifics
- Motivation for the work

Presentation outline

In this presentation we will be talking about the following:

- First, some theoretical background on memory controllers (MCs) and some DRAM specifics
- Motivation for the work
- How multicore contention takes place within the memory controller

Presentation outline

In this presentation we will be talking about the following:

- First, some theoretical background on memory controllers (MCs) and some DRAM specifics
- Motivation for the work
- How multicore contention takes place within the memory controller
- What we propose VS. what's currently out there

Presentation outline

In this presentation we will be talking about the following:

- First, some theoretical background on memory controllers (MCs) and some DRAM specifics
- Motivation for the work
- How multicore contention takes place within the memory controller
- What we propose VS. what's currently out there
- Status of our work so far

Executive summary

The presentation, in a few sentences:

- We propose an extension of an MC’s timing-constraints table that regulates the execution of DRAM commands

Executive summary

The presentation, in a few sentences:

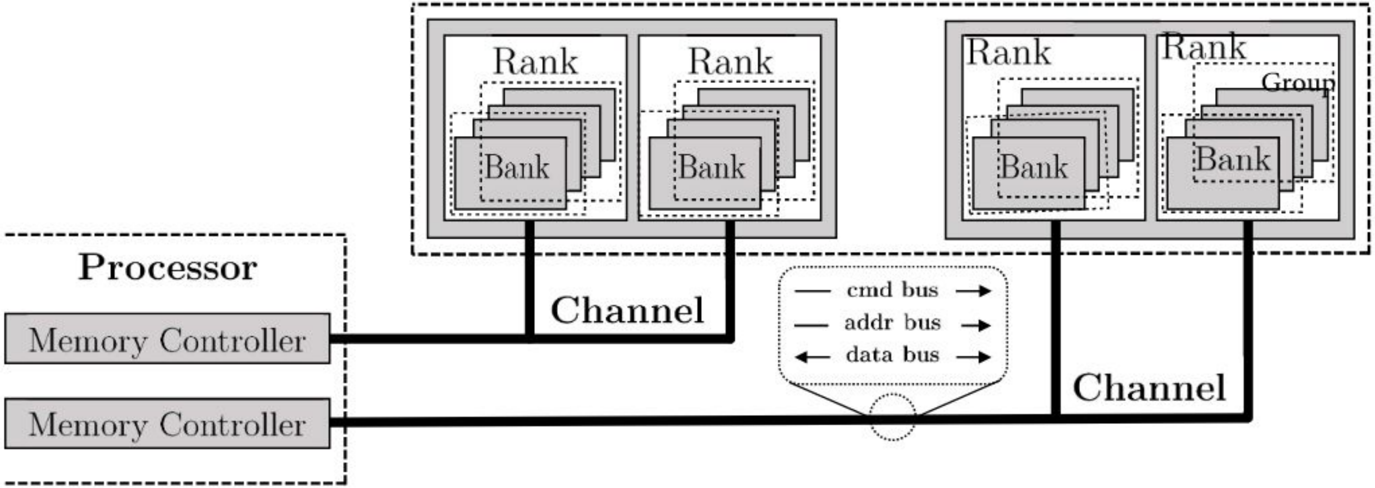
- We propose an extension of an MC’s timing-constraints table that regulates the execution of DRAM commands
- The timing-constraints table can be expanded upon to incorporate different levels of contention specificity

Executive summary

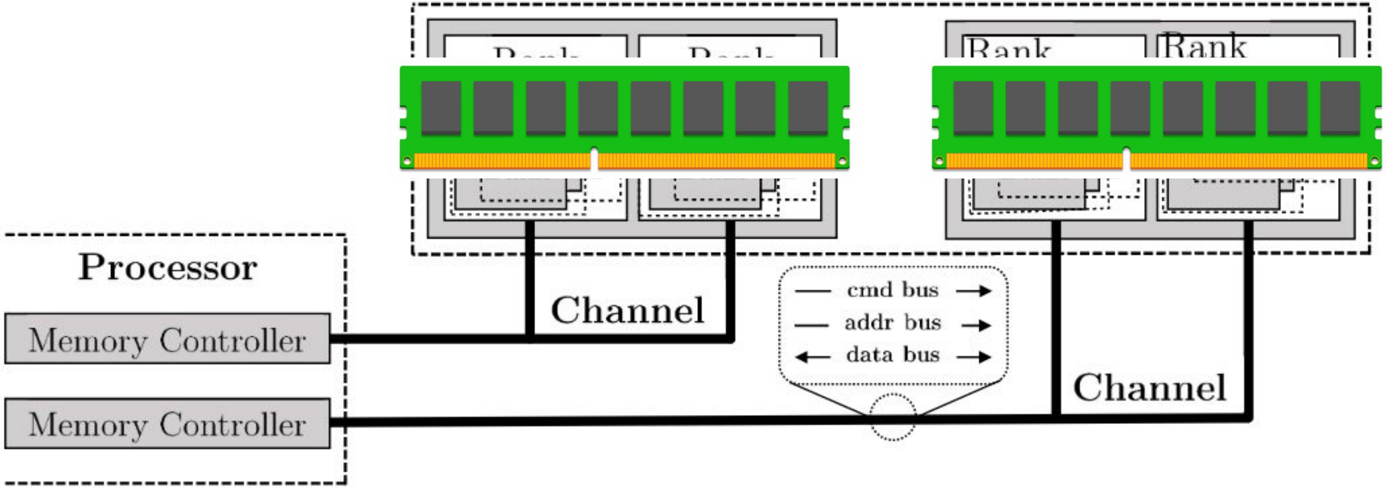
The presentation, in a few sentences:

- We propose an extension of an MC’s timing-constraints table that regulates the execution of DRAM commands
- The timing-constraints table can be expanded-upon to incorporate different levels of contention specificity
- This extension is compatible with static- and dynamic-command-scheduling controllers

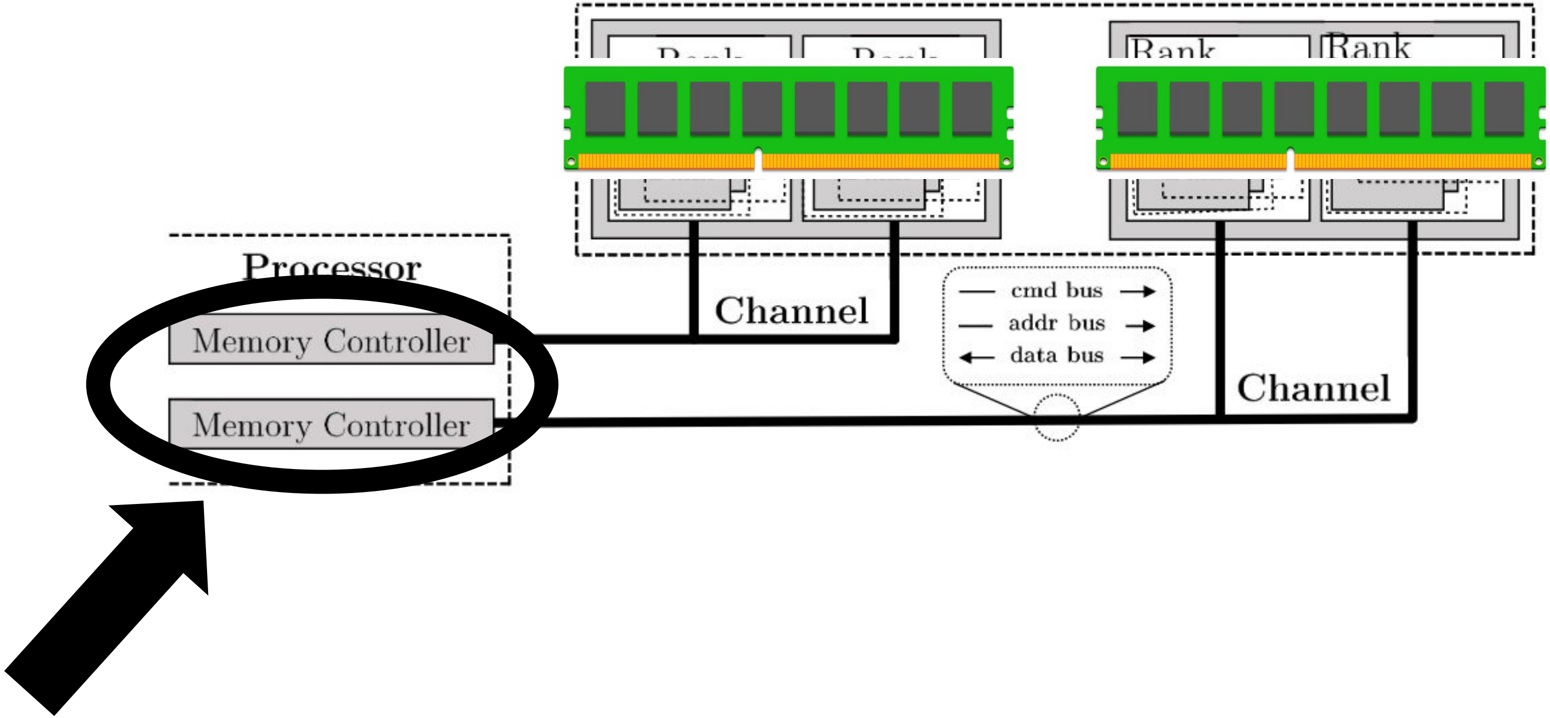
Memory controllers

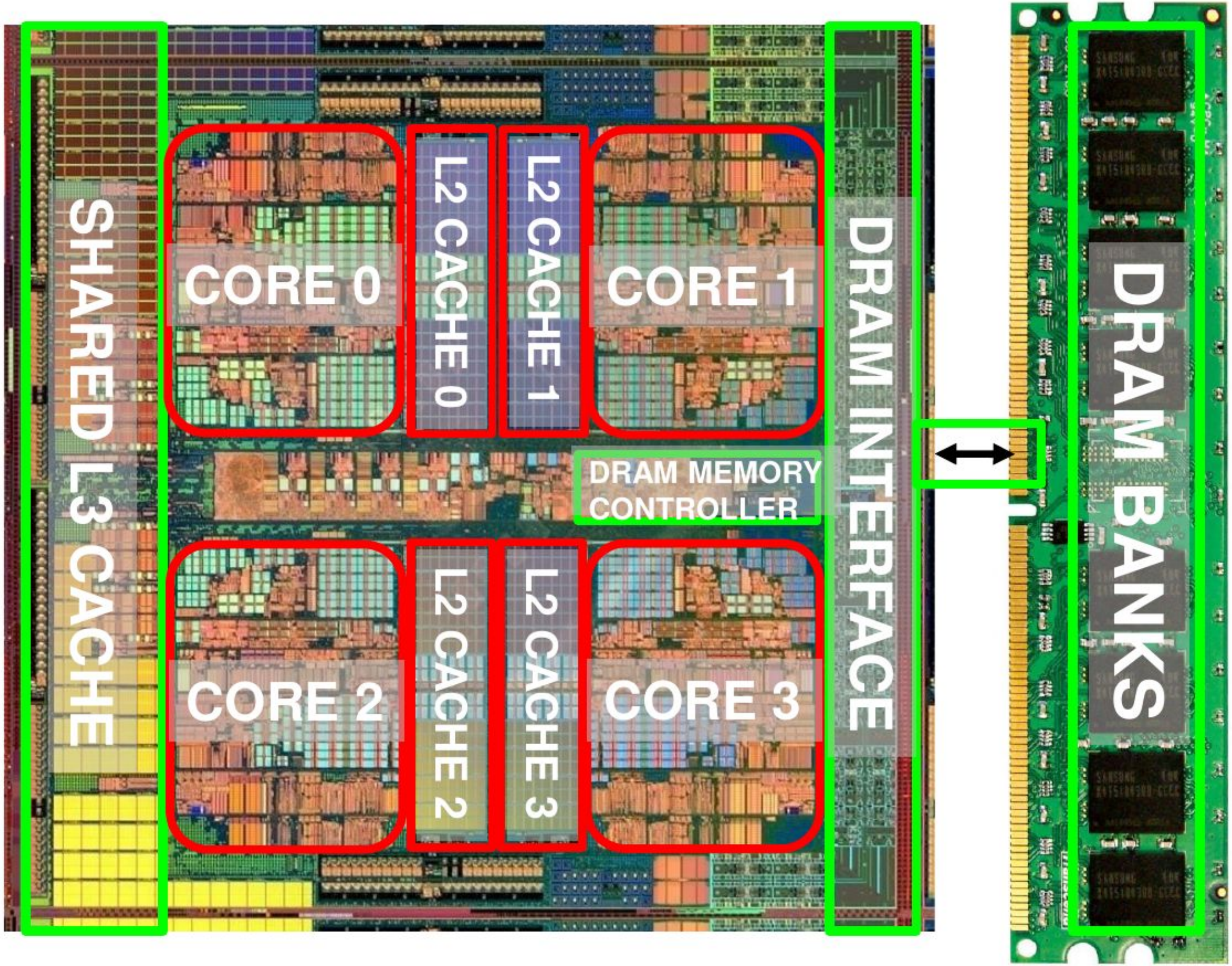


Memory controllers



Memory controllers

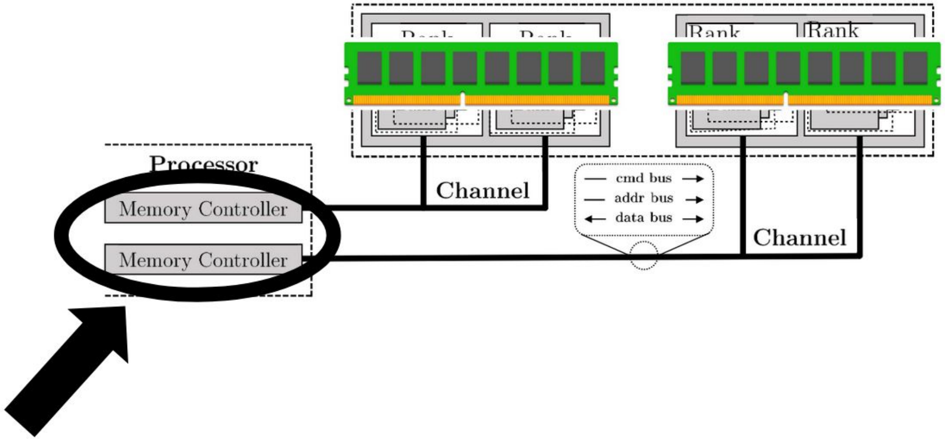




AMD Barcelona, 2006

Memory controllers

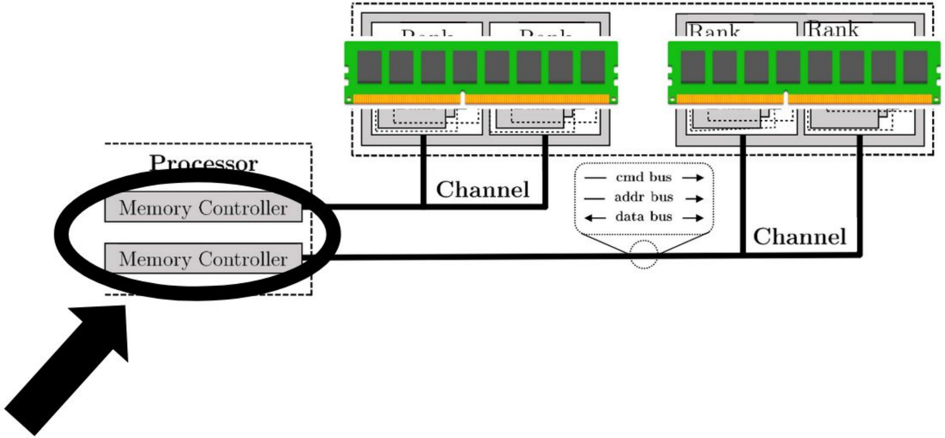
DRAM MCs manage **read** and **write transactions** from CPU to DRAM



Memory controllers

DRAM MCs manage **read** and **write transactions** from CPU to DRAM

- Read transactions → cache misses
- Write transactions → dirty cache lines or data from HDD

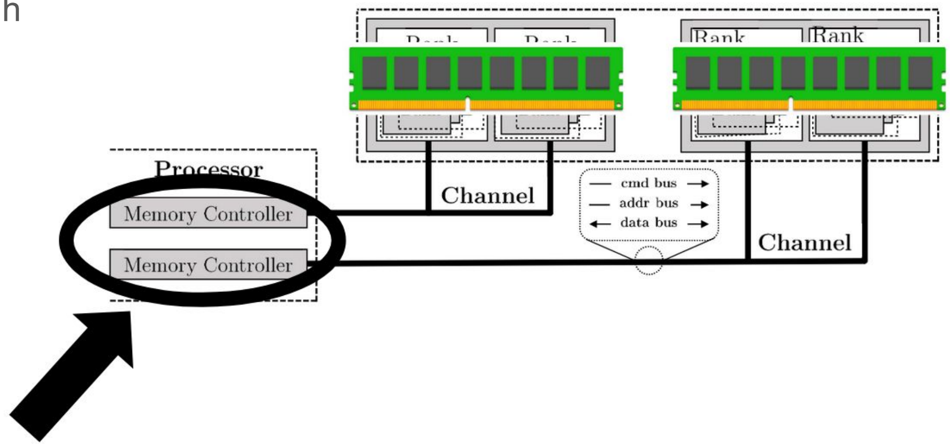


Memory controllers

DRAM MCs manage **read** and **write transactions** from CPU to DRAM

- Read transactions → cache misses
- Write transactions → dirty cache lines or data from HDD

Memory controllers communicate with DRAM through **(1) an address bus**, **(2) a command bus** and **(3) a data bus**

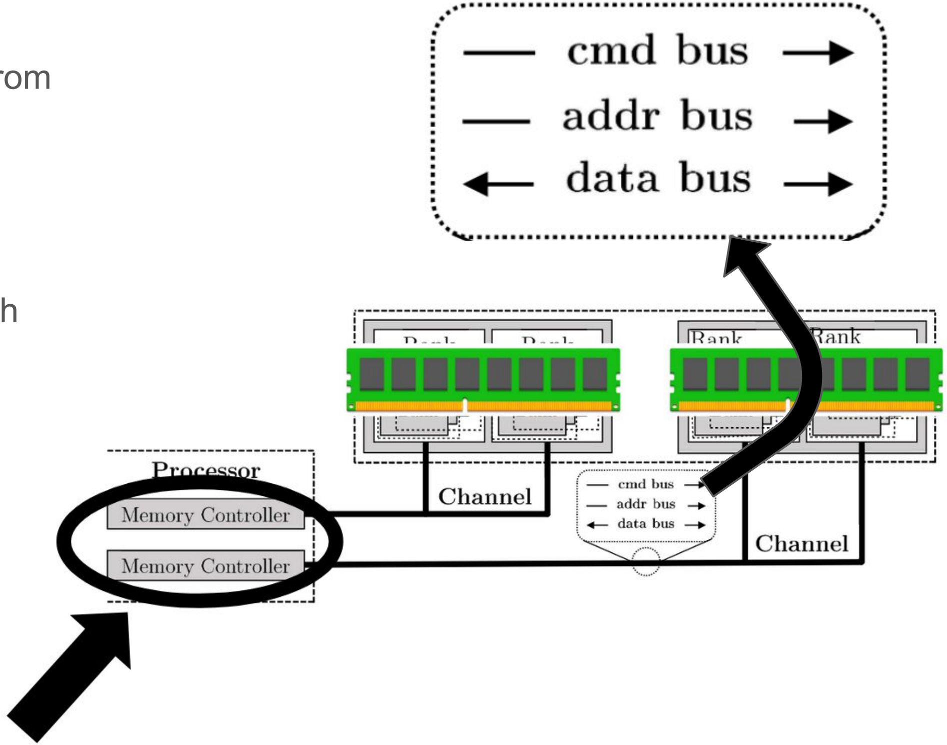


Memory controllers

DRAM MCs manage **read** and **write transactions** from CPU to DRAM

- Read transactions → cache misses
- Write transactions → dirty cache lines or data from HDD

Memory controllers communicate with DRAM through **(1) an address bus**, **(2) a command bus** and **(3) a data bus**



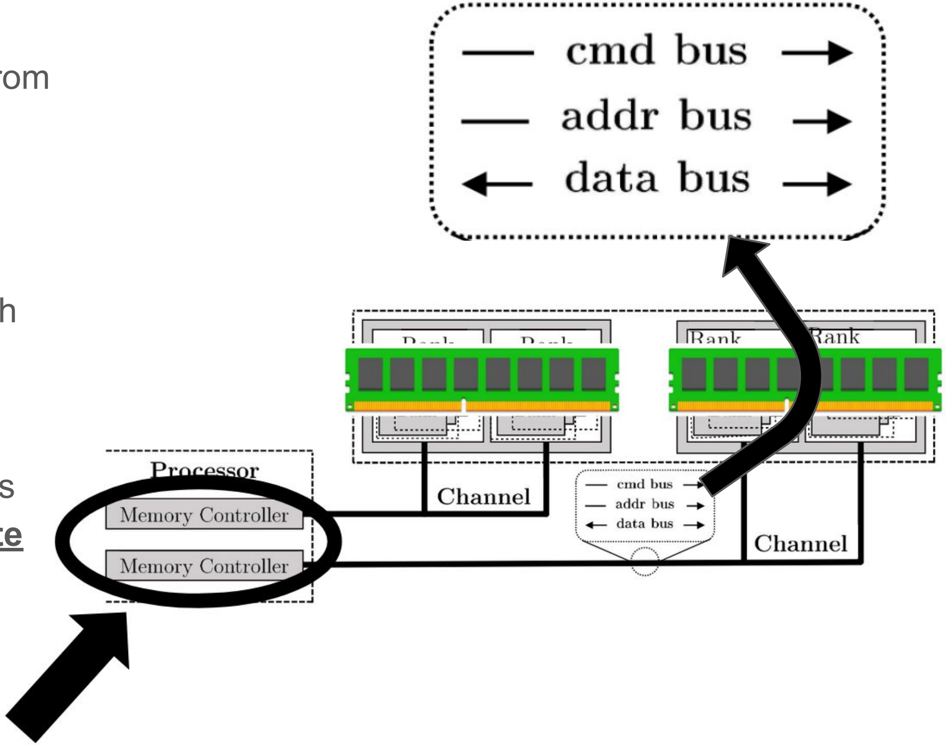
Memory controllers

DRAM MCs manage **read** and **write transactions** from CPU to DRAM

- Read transactions → cache misses
- Write transactions → dirty cache lines or data from HDD

Memory controllers communicate with DRAM through **(1) an address bus**, **(2) a command bus** and **(3) a data bus**

A memory controller communicates to DRAM devices the necessary **DRAM commands** to **fulfill read/write transactions**



Memory controllers

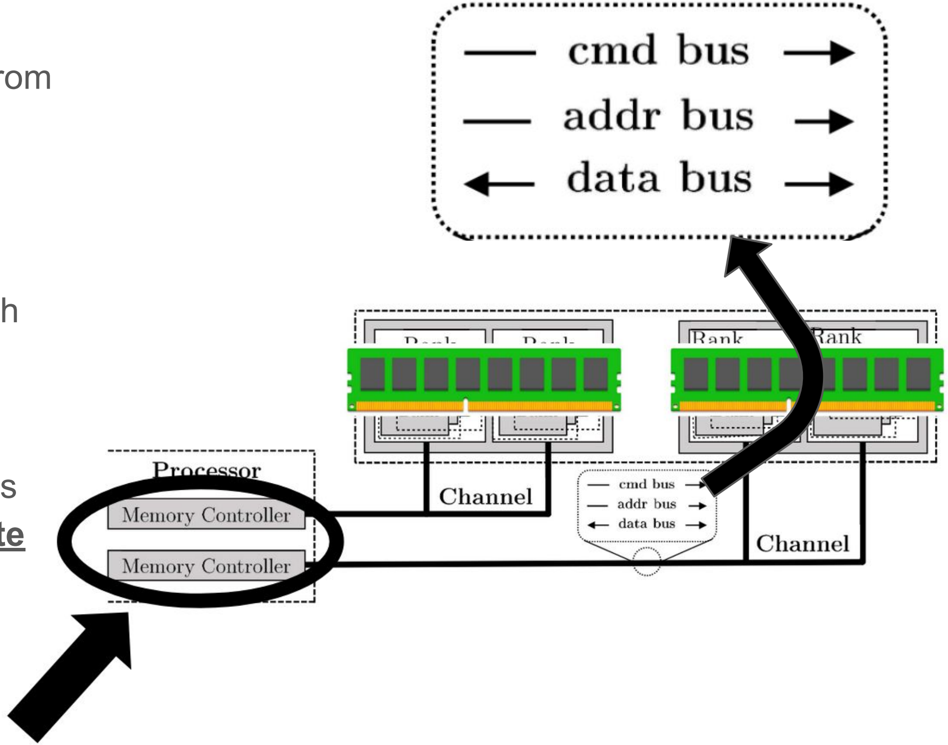
DRAM MCs manage **read** and **write transactions** from CPU to DRAM

- Read transactions → cache misses
- Write transactions → dirty cache lines or data from HDD

Memory controllers communicate with DRAM through **(1) an address bus**, **(2) a command bus** and **(3) a data bus**

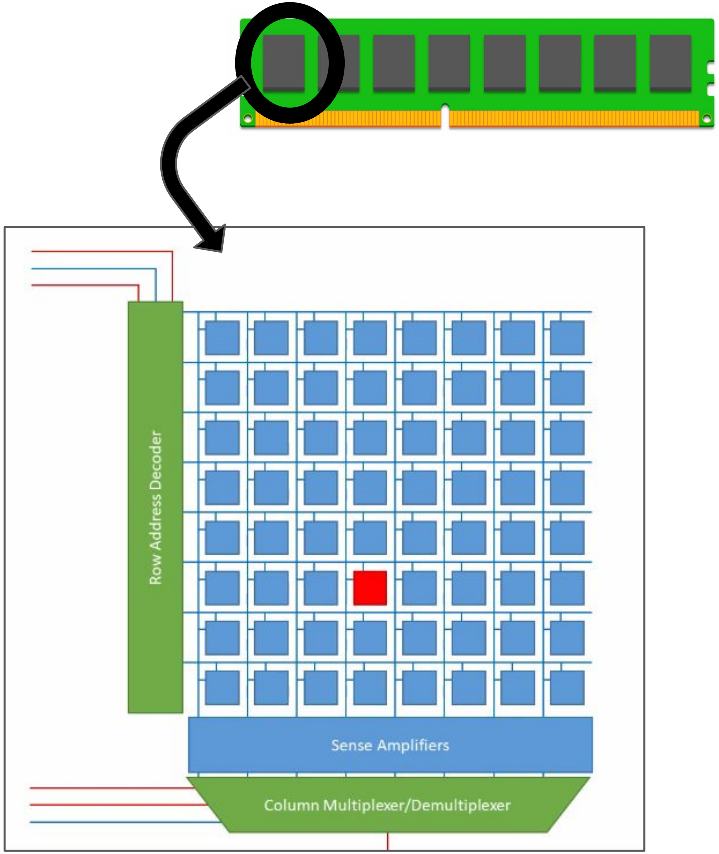
A memory controller communicates to DRAM devices the necessary **DRAM commands** to **fulfill read/write transactions**

Wait… DRAM commands???



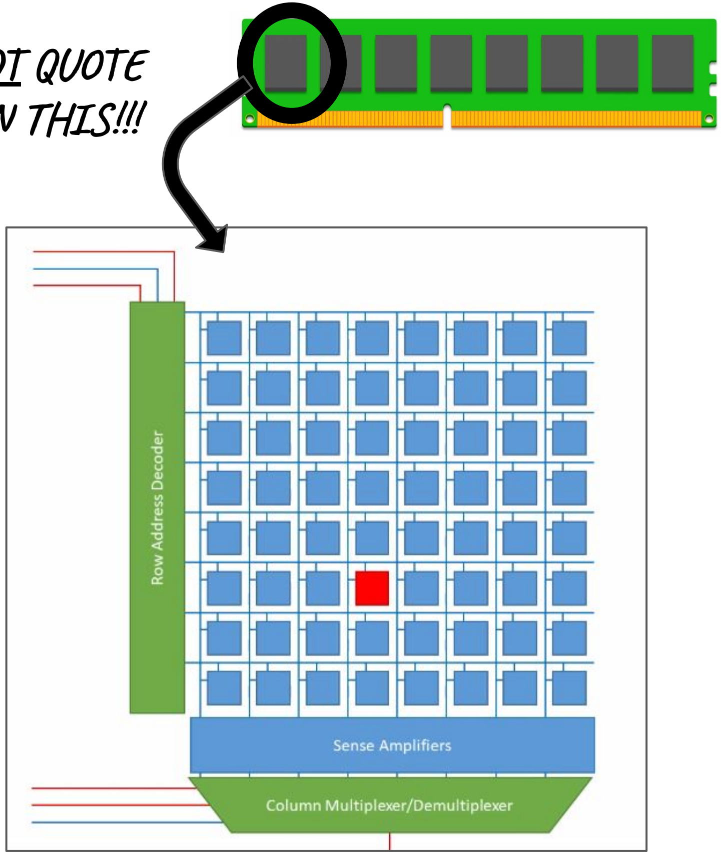
DRAM protocol speedrun

Let’s get into DRAM commands and JEDEC timing constraints!! :)



DRAM protocol speedrun

Let’s get into DRAM commands and JEDEC timing constraints!! :)

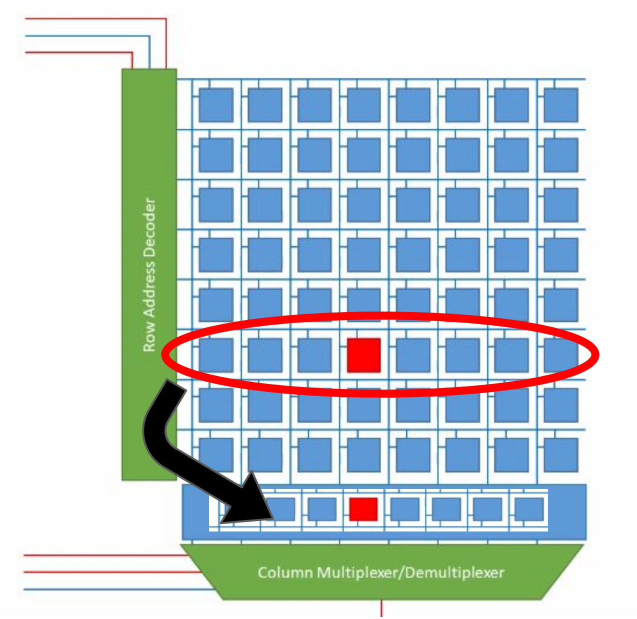


DO NOT QUOTE ME ON THIS!!!

DRAM protocol speedrun

Let’s get into DRAM commands and JEDEC timing constraints!! :)

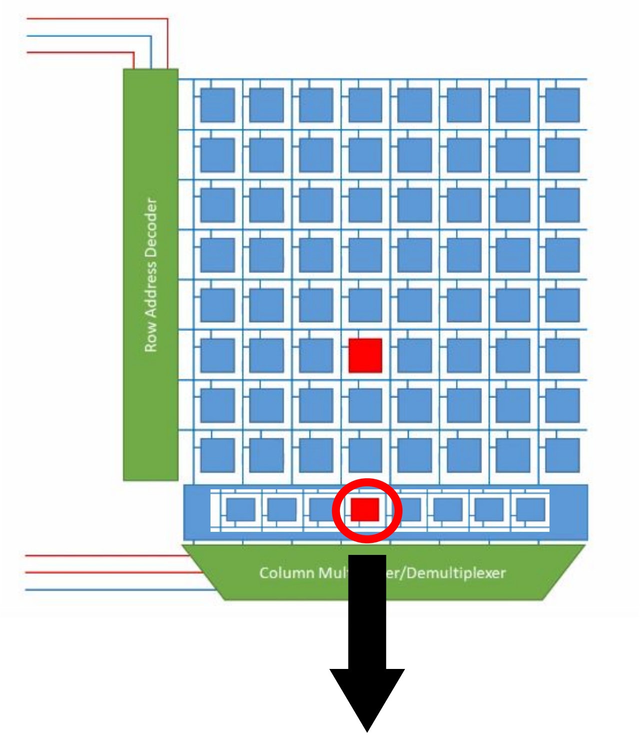
- Moving data from the data array to the sense amplifiers is done through an **ACTIVATE command (ACT)**



DRAM protocol speedrun

Let’s get into DRAM commands and JEDEC timing constraints!! :)

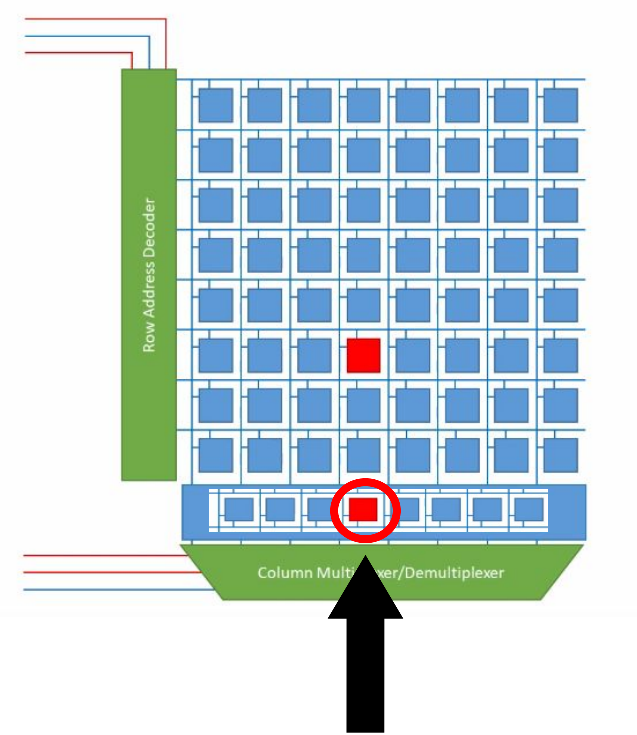
- Moving data from the data array to the sense amplifiers is done through an **ACTIVATE command (ACT)**
- Once data is residing in the sense amplifiers (also called row buffer), data can be either **READ**



DRAM protocol speedrun

Let’s get into DRAM commands and JEDEC timing constraints!! :)

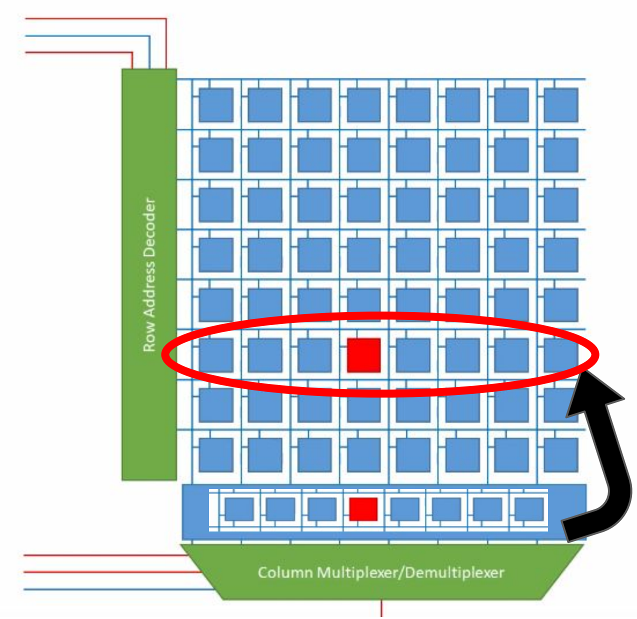
- Moving data from the data array to the sense amplifiers is done through an **ACTIVATE command (ACT)**
- Once data is residing in the sense amplifiers (also called row buffer), data can be either **READ** or **WRITTEN**



DRAM protocol speedrun

Let’s get into DRAM commands and JEDEC timing constraints!! :)

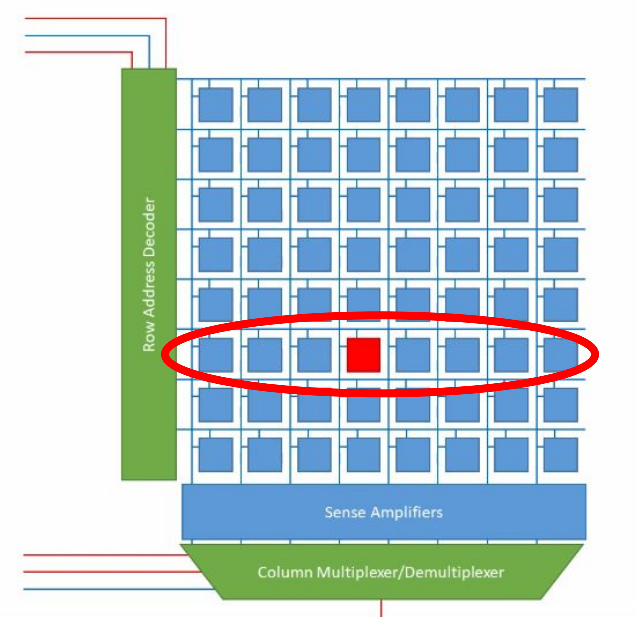
- Moving data from the data array to the sense amplifiers is done through an **ACTIVATE command (ACT)**
- Once data is residing in the sense amplifiers (also called row buffer), data can be either **READ** or **WRITTEN**
- If another row is needed, data residing in the row buffer must be brought back to the data array through a **PRECHARGE** command (reads are destructive)



DRAM protocol speedrun

Let’s get into DRAM commands and JEDEC timing constraints!! :)

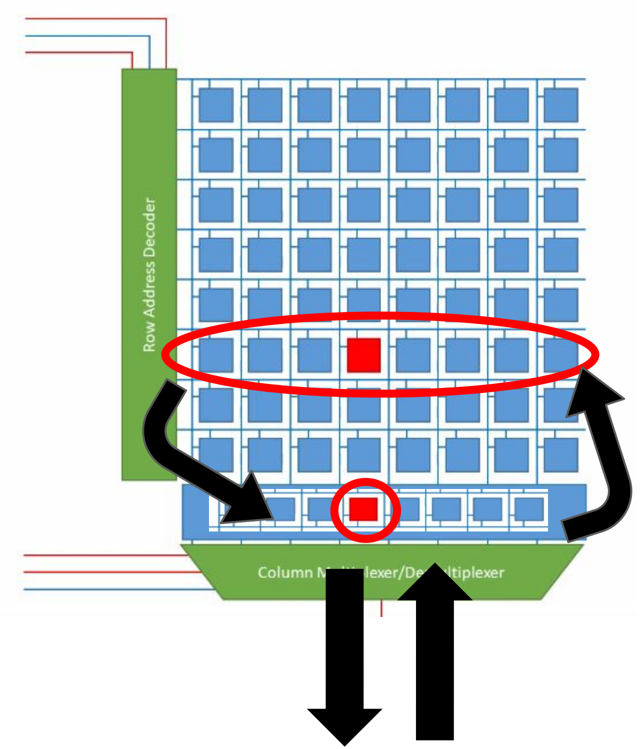
- Moving data from the data array to the sense amplifiers is done through an **ACTIVATE command (ACT)**
- Once data is residing in the sense amplifiers (also called row buffer), data can be either **READ** or **WRITTEN**
- If another row is needed, data residing in the row buffer must be brought back to the data array through a **PRECHARGE** command (reads are destructive)
- After precharging the previous row, **the cycle of ACT-READ/WRITE-PRE** starts all over again…



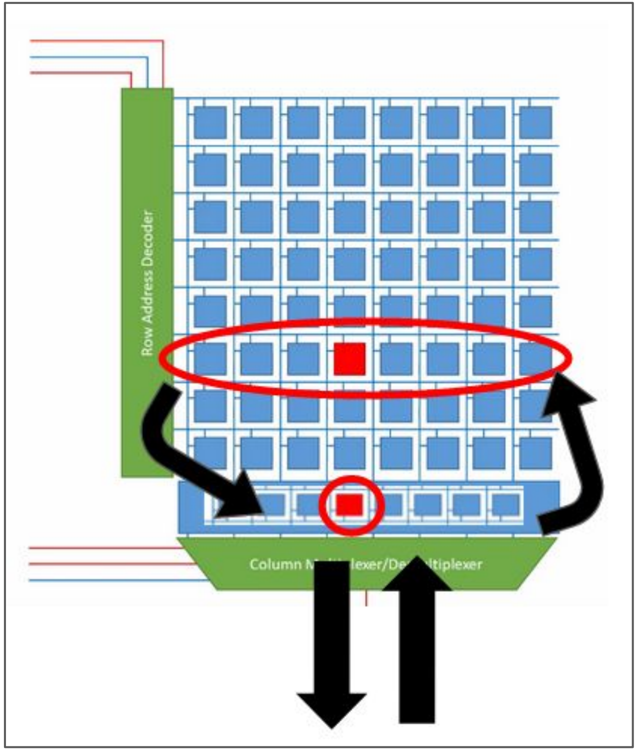
DRAM protocol speedrun

Let’s get into DRAM commands and JEDEC timing constraints!! :)

- Moving data from the data array to the sense amplifiers is done through an **ACTIVATE command (ACT)**
- Once data is residing in the sense amplifiers (also called row buffer), data can be either **READ** or **WRITTEN**
- If another row is needed, data residing in the row buffer must be brought back to the data array through a **PRECHARGE** command (reads are destructive)
- After precharging the previous row, **the cycle of ACT-READ/WRITE-PRE** starts all over again…

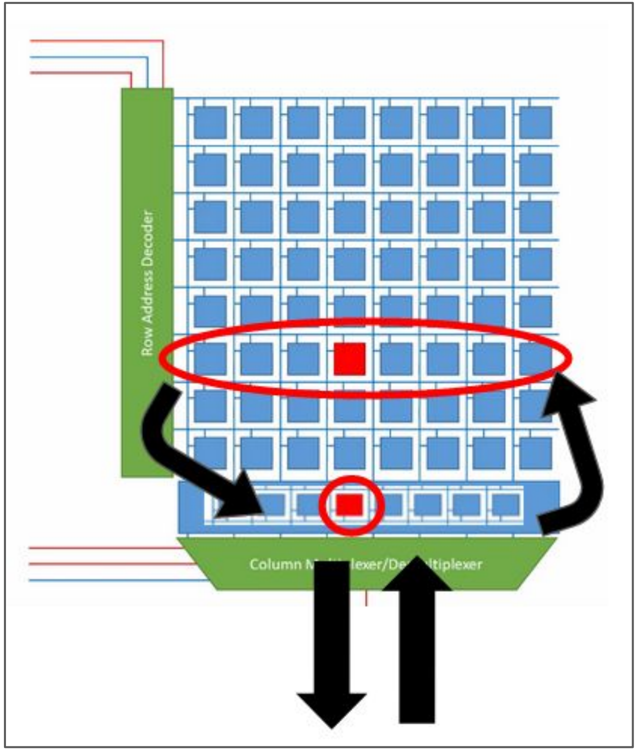


DRAM protocol speedrun



KEY IDEA: DRAM commands must respect certain timing constraints between each other to ensure integrity of data

DRAM protocol speedrun



<u>KEY IDEA:</u> DRAM commands must respect certain <u>timing constraints between each other</u> to ensure integrity of data

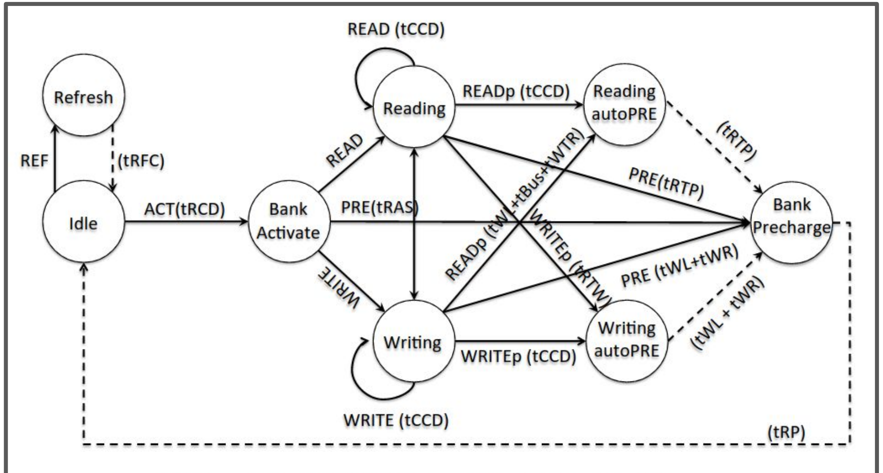
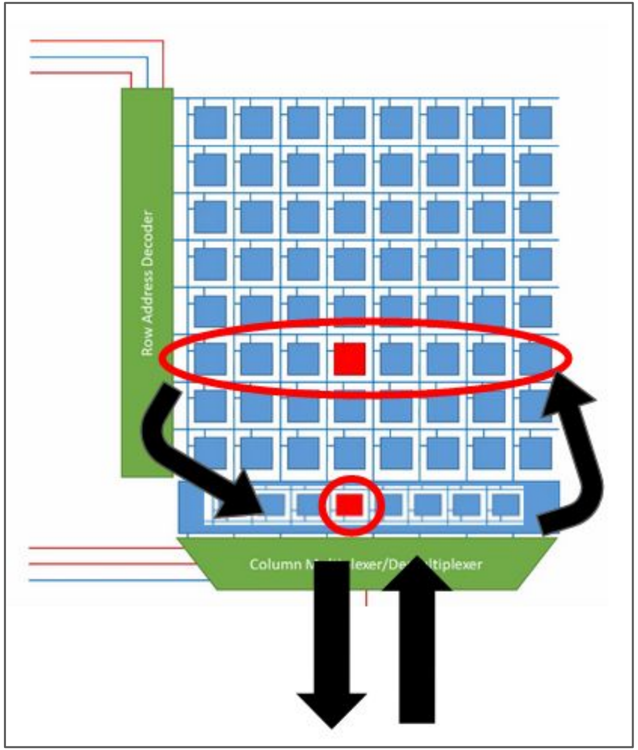
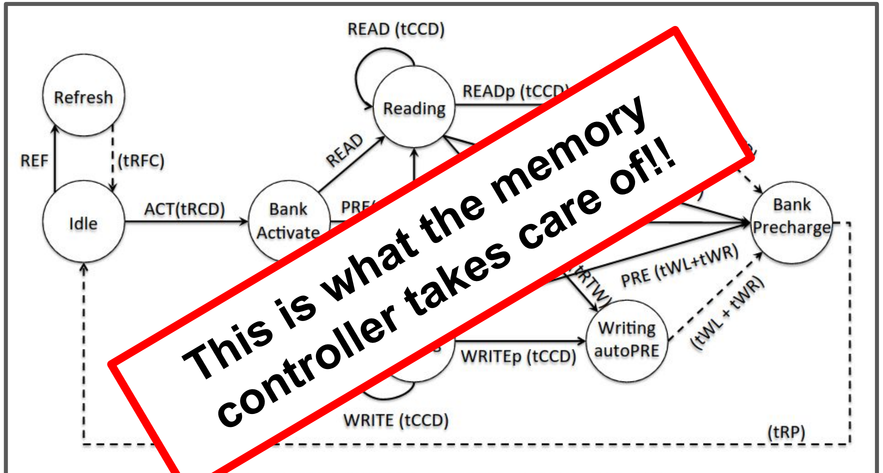


Figure 2.2: DRAM Operation State Machine.

DRAM protocol speedrun



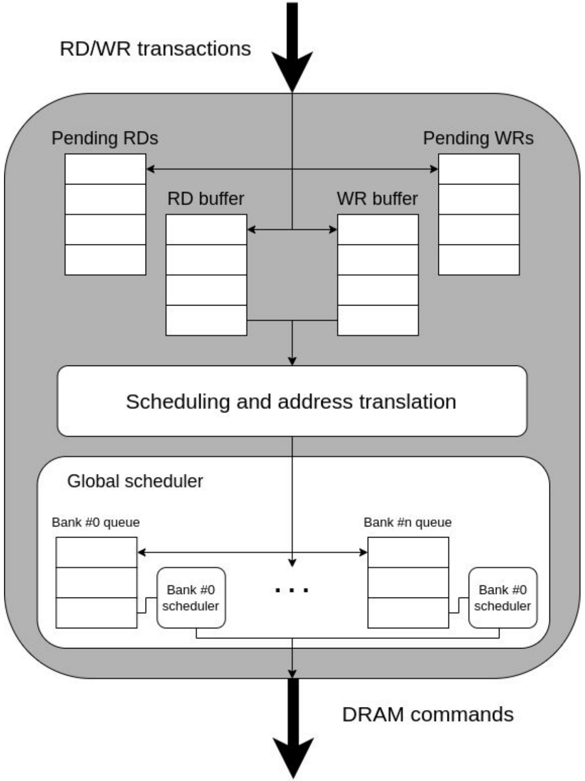
KEY IDEA: DRAM commands must respect certain timing constraints between each other to ensure integrity of data



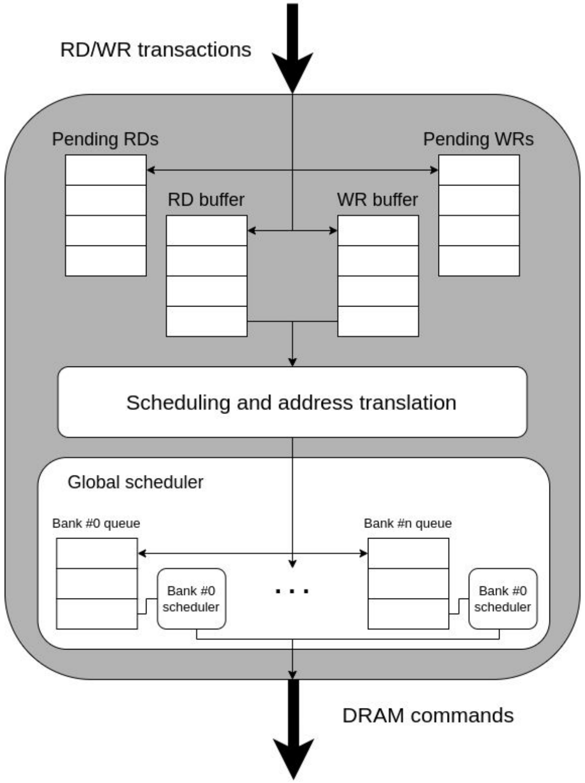
This is what the memory controller takes care of!!

Figure 2.2: DRAM Operation State Machine.

Structure (logical) of a memory controller

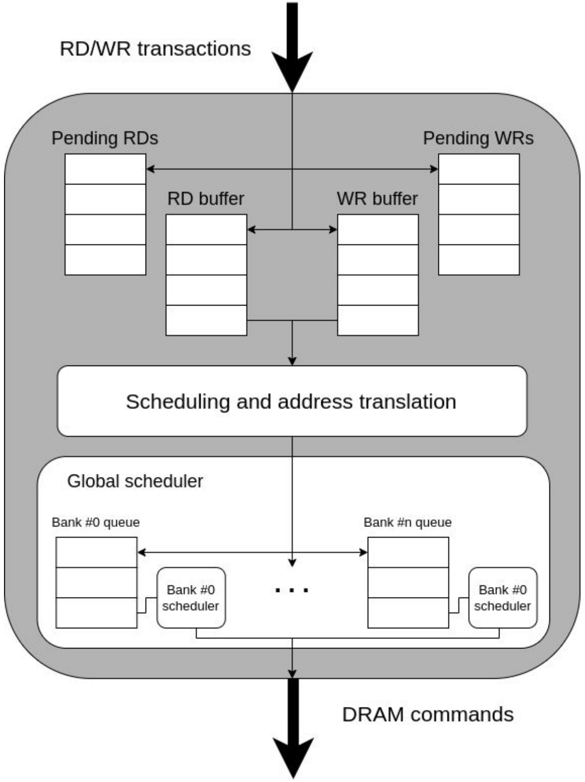


Structure (logical) of a memory controller



Front-end:
transaction-level
management

Structure (logical) of a memory controller

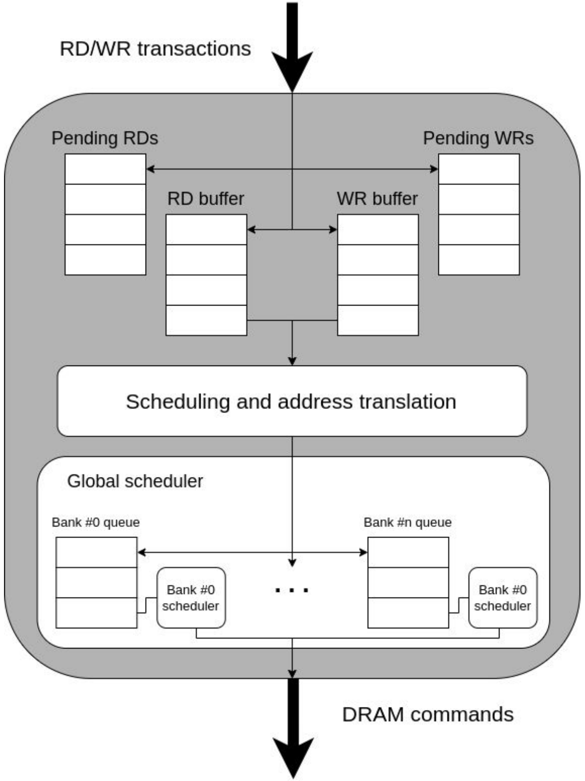


__Front-end:__ transaction-level management

__Back-end:__ DRAM command generation

Structure (logical) of a memory controller

Back-end: DRAM command generation



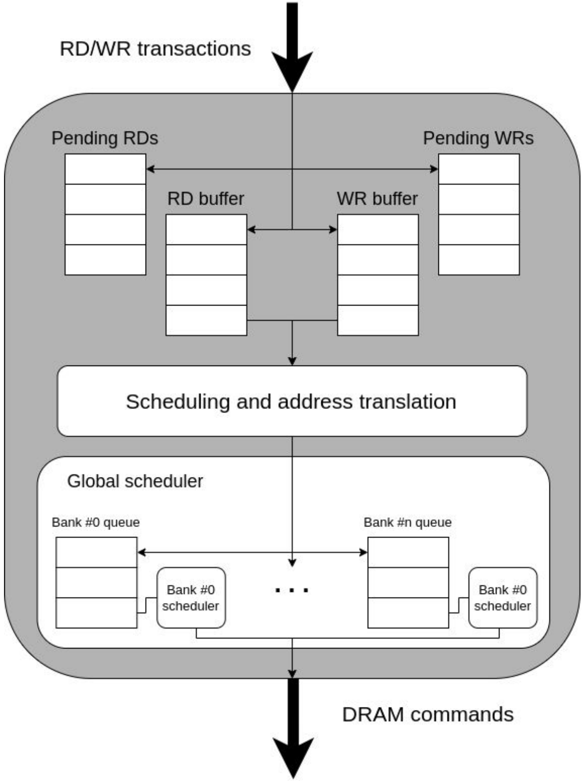
Front-end: transaction-level management

Not so relevant, contention-wise

Structure (logical) of a memory controller

This is where the magic happens!!

Back-end: DRAM command generation

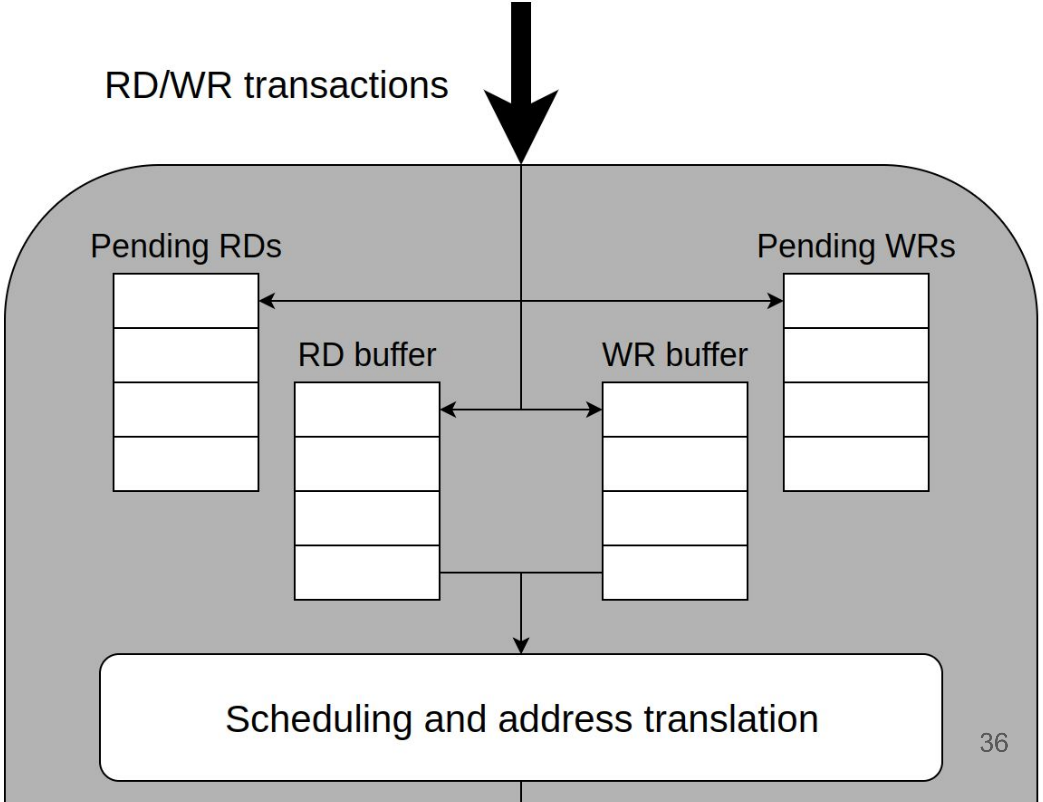


Front-end: transaction-level management

Not so relevant, contention-wise

Structure (logical) of a memory controller

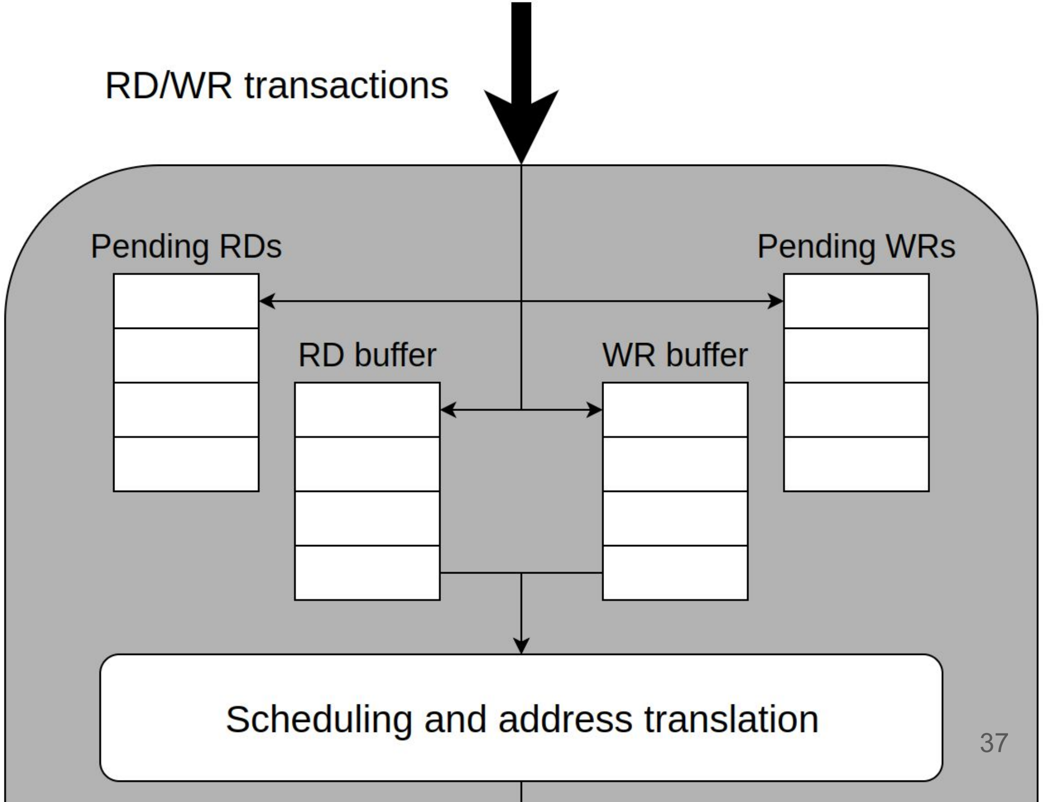
Let's start with transaction management:



Structure (logical) of a memory controller

Let’s start with transaction management:

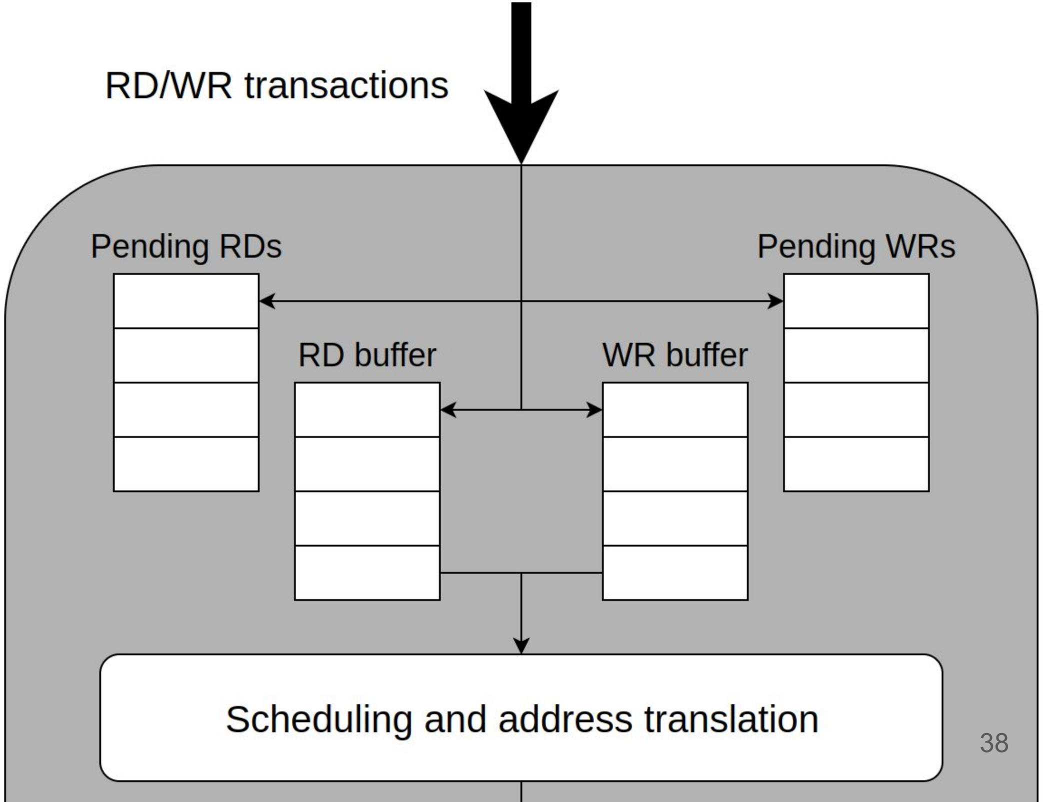
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**



Structure (logical) of a memory controller

Let’s start with transaction management:

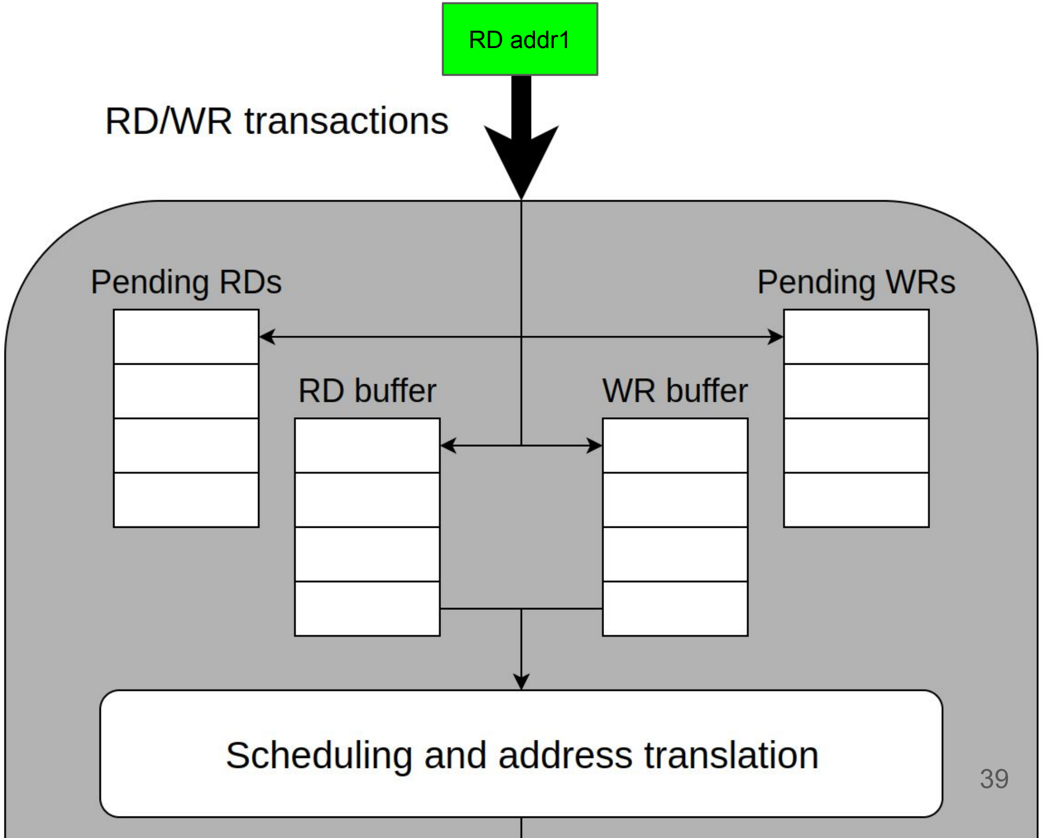
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**



Structure (logical) of a memory controller

Let’s start with transaction management:

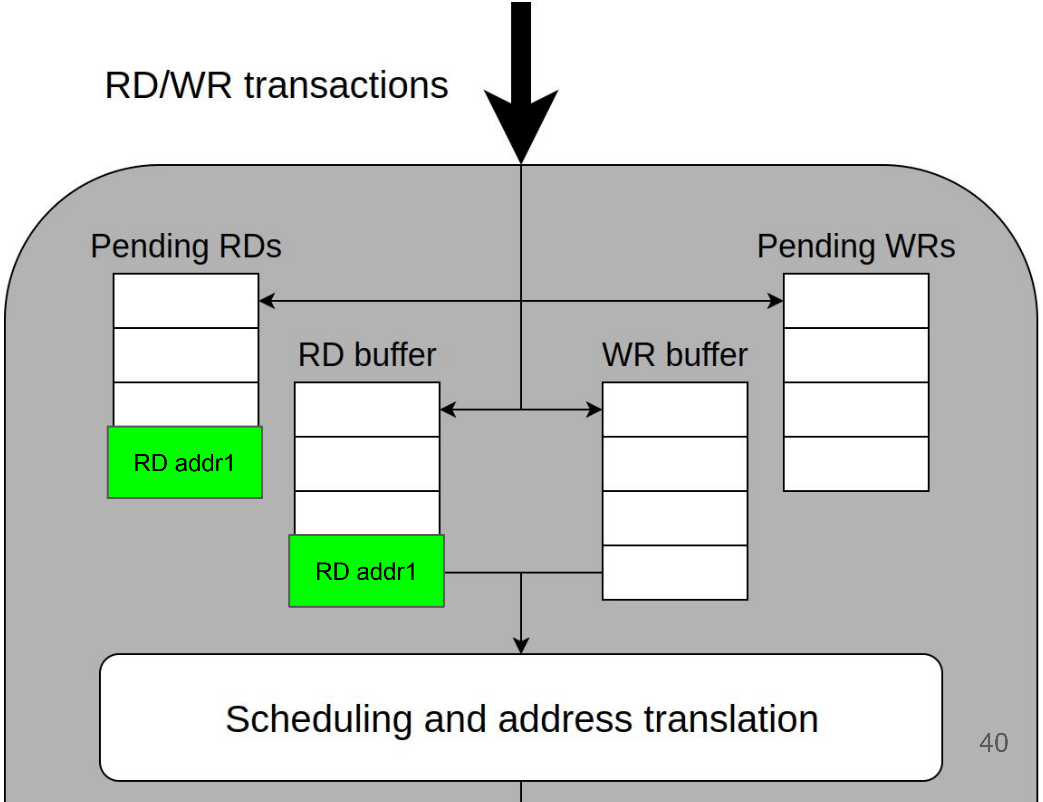
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**



Structure (logical) of a memory controller

Let's start with transaction management:

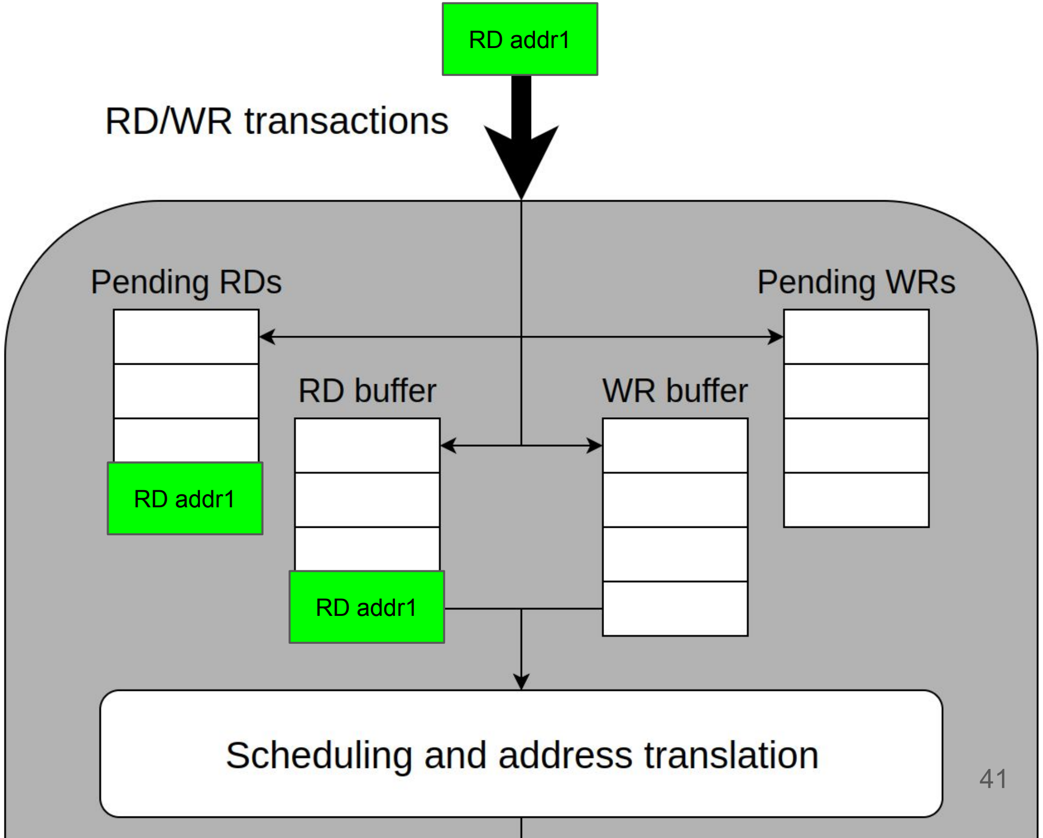
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**



Structure (logical) of a memory controller

Let's start with transaction management:

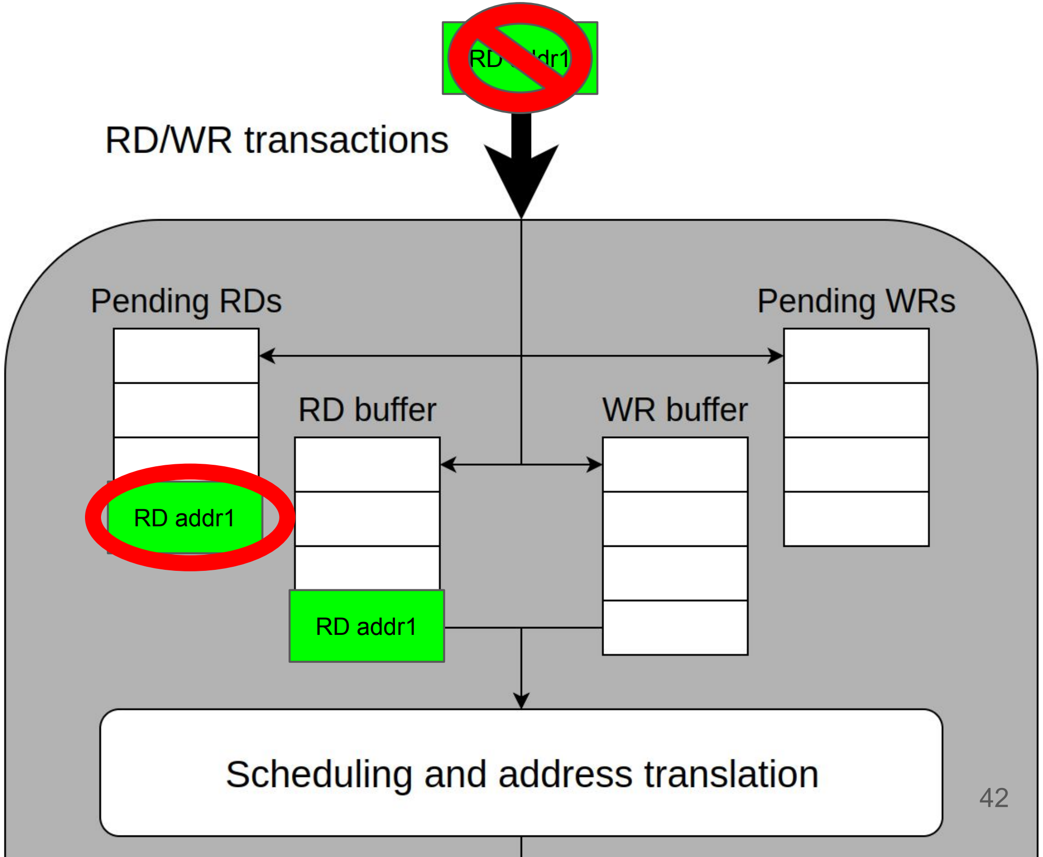
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**



Structure (logical) of a memory controller

Let’s start with transaction management:

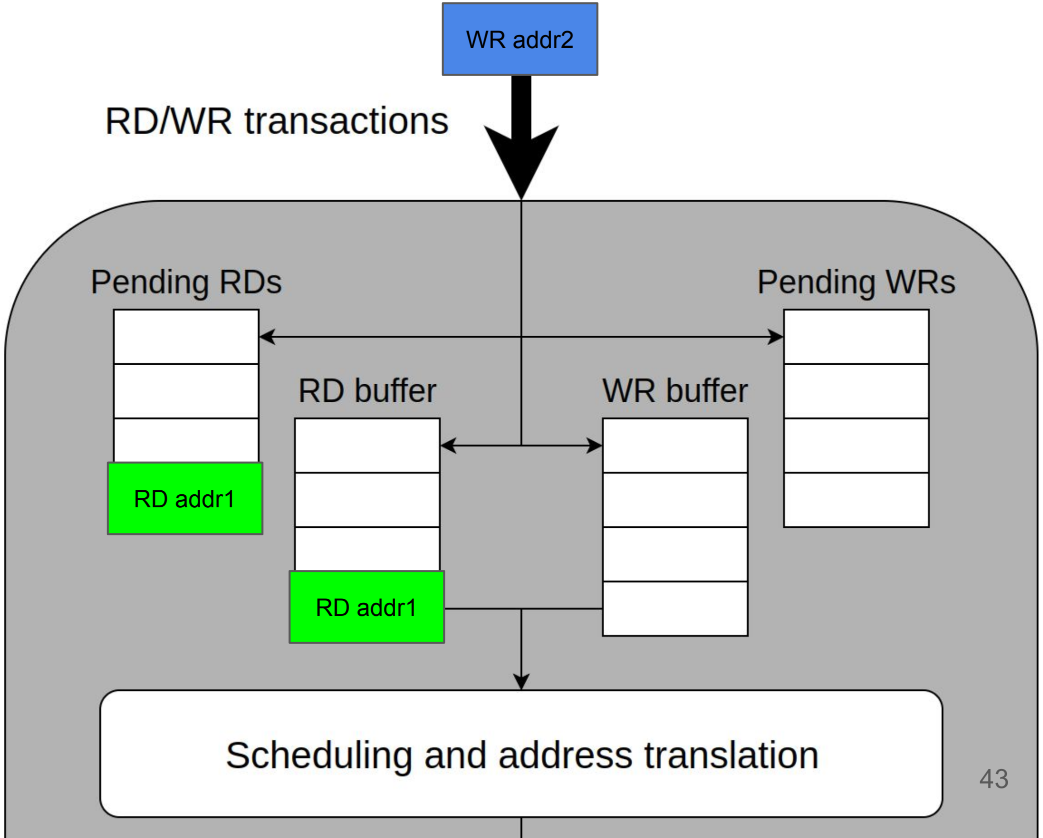
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**



Structure (logical) of a memory controller

Let's start with transaction management:

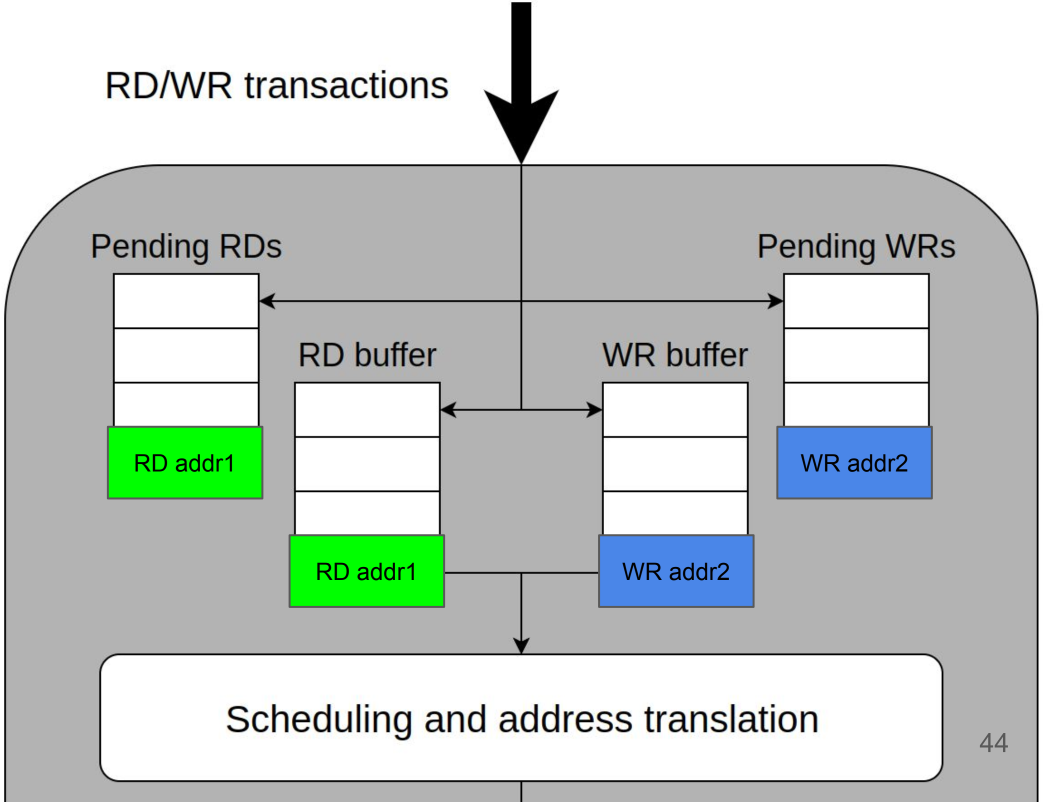
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**



Structure (logical) of a memory controller

Let's start with transaction management:

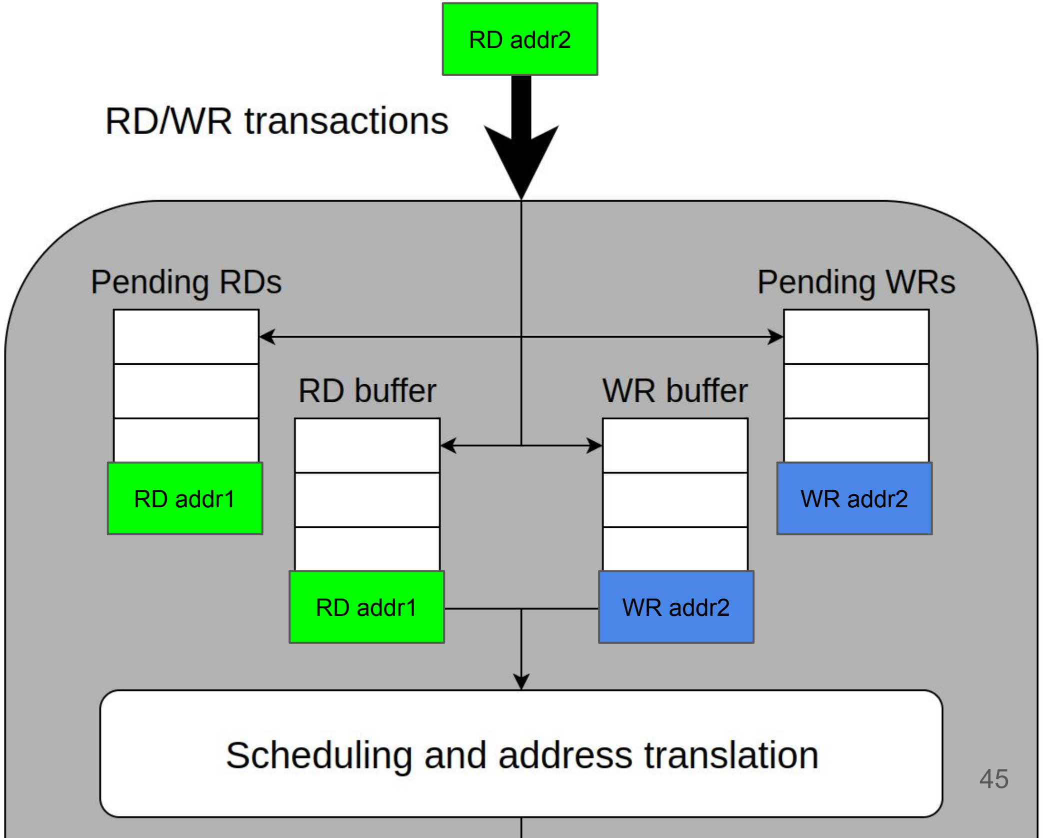
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**



Structure (logical) of a memory controller

Let’s start with transaction management:

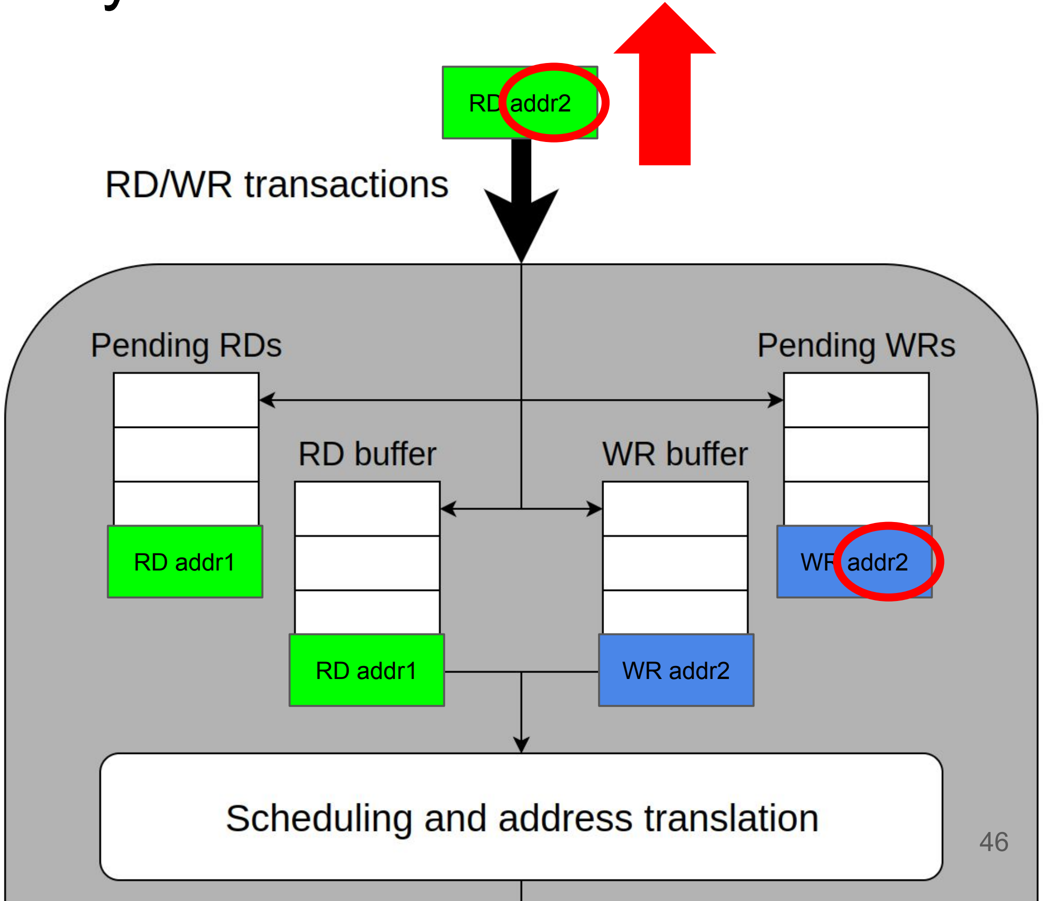
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**



Structure (logical) of a memory controller

Let’s start with transaction management:

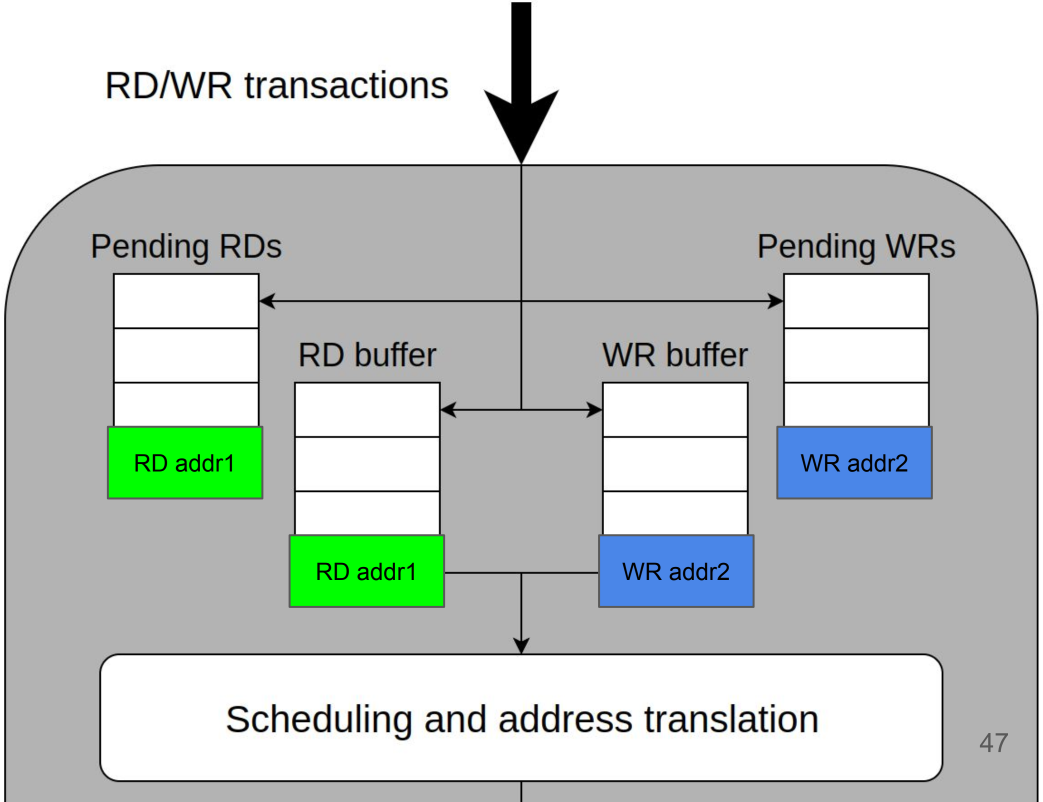
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**



Structure (logical) of a memory controller

Let's start with transaction management:

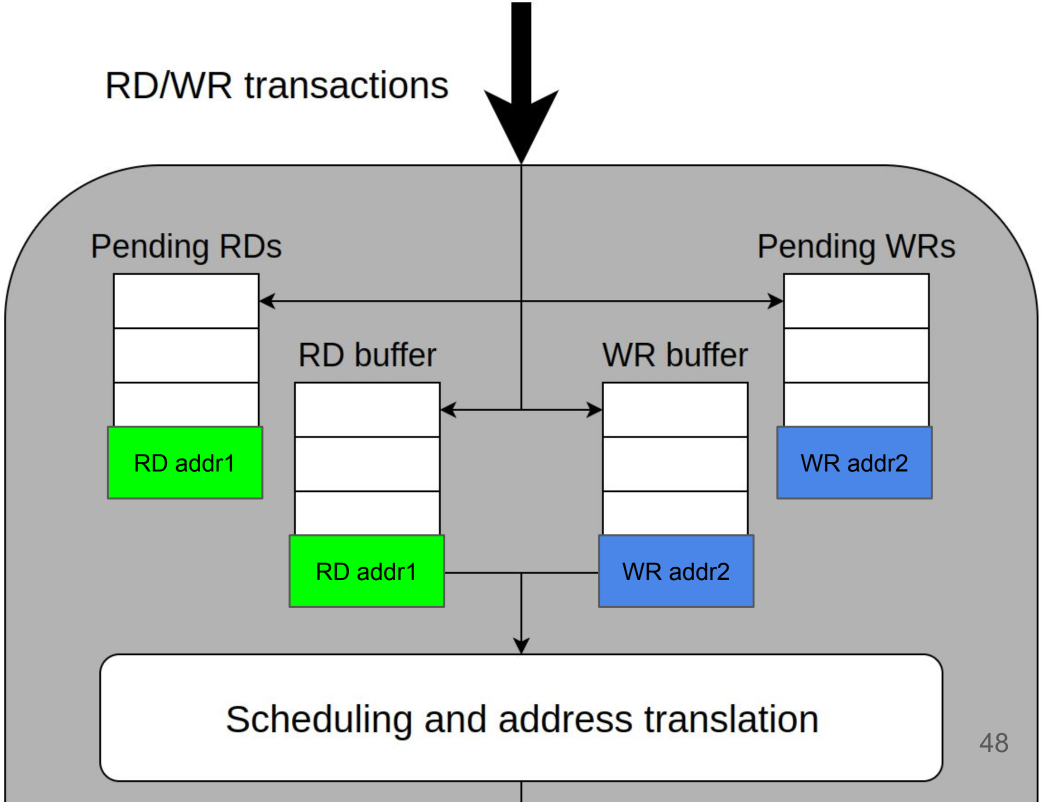
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**
- RD and WR buffers hold the **transactions to be scheduled** to DRAM



Structure (logical) of a memory controller

Let’s start with transaction management:

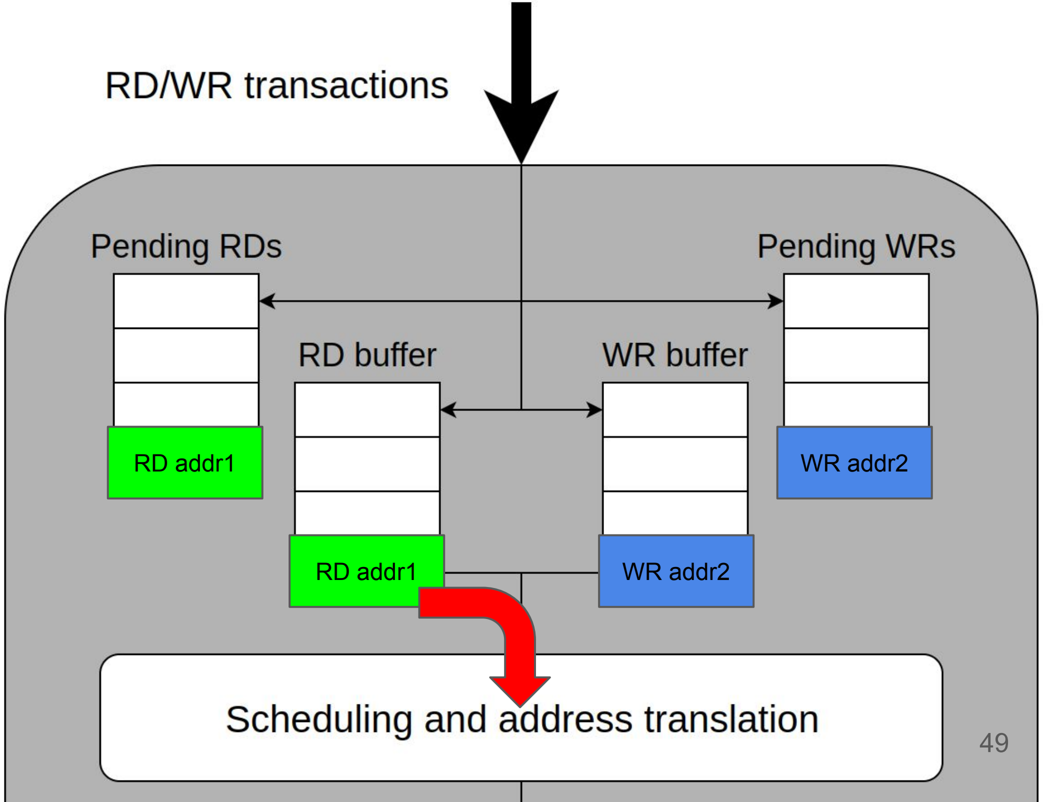
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**
- RD and WR buffers hold the **transactions to be scheduled** to DRAM
- For the purposes of this presentation, scheduling a transaction only implies **moving it from top to bottom for DRAM command generation**



Structure (logical) of a memory controller

Let’s start with transaction management:

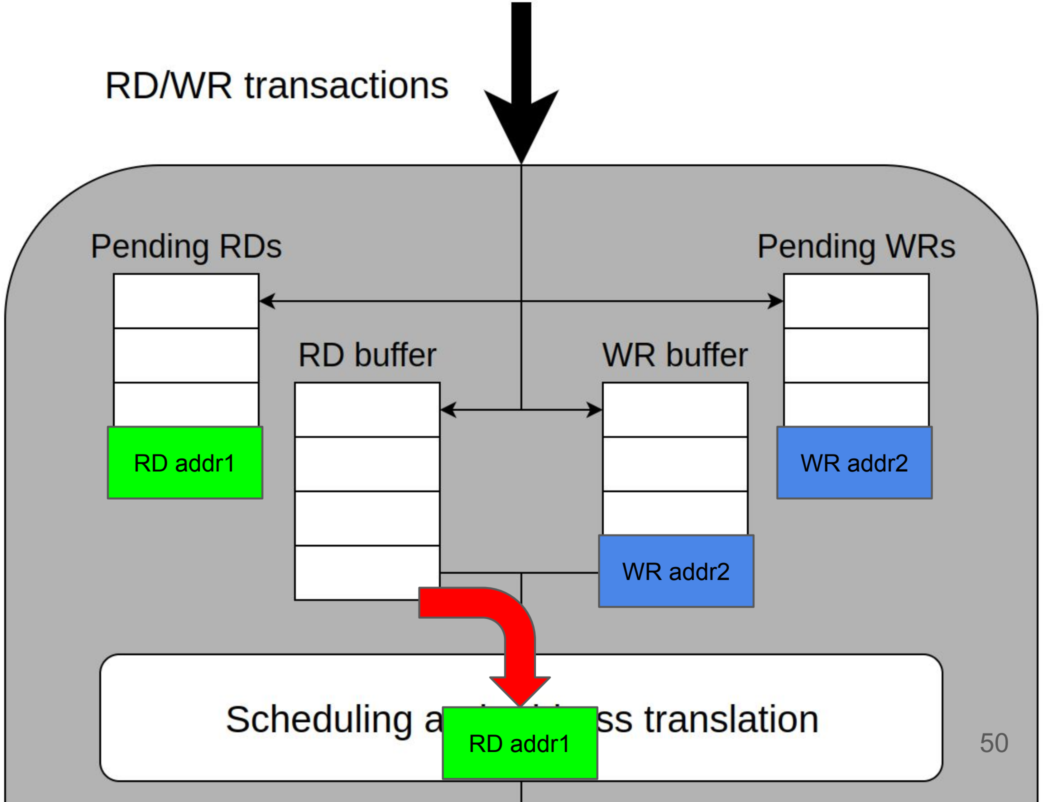
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**
- RD and WR buffers hold the **transactions to be scheduled** to DRAM
- For the purposes of this presentation, scheduling a transaction only implies **moving it from top to bottom for DRAM command generation**



Structure (logical) of a memory controller

Let’s start with transaction management:

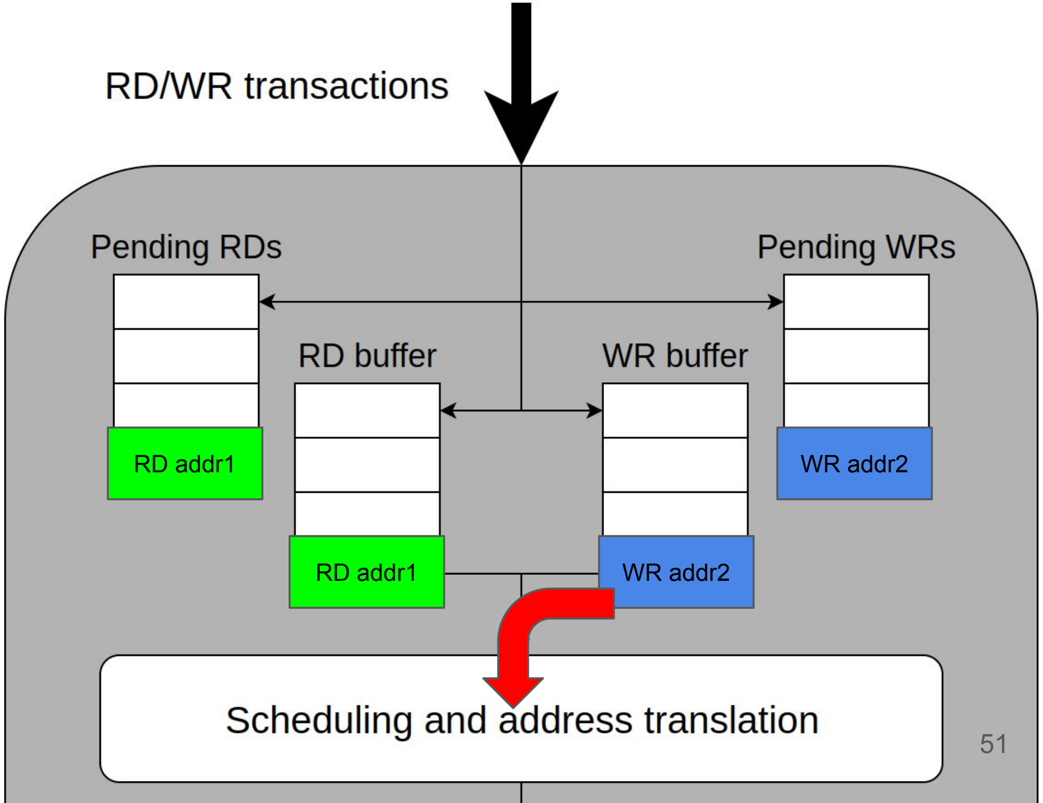
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**
- RD and WR buffers hold the **transactions to be scheduled** to DRAM
- For the purposes of this presentation, scheduling a transaction only implies **moving it from top to bottom for DRAM command generation**



Structure (logical) of a memory controller

Let’s start with transaction management:

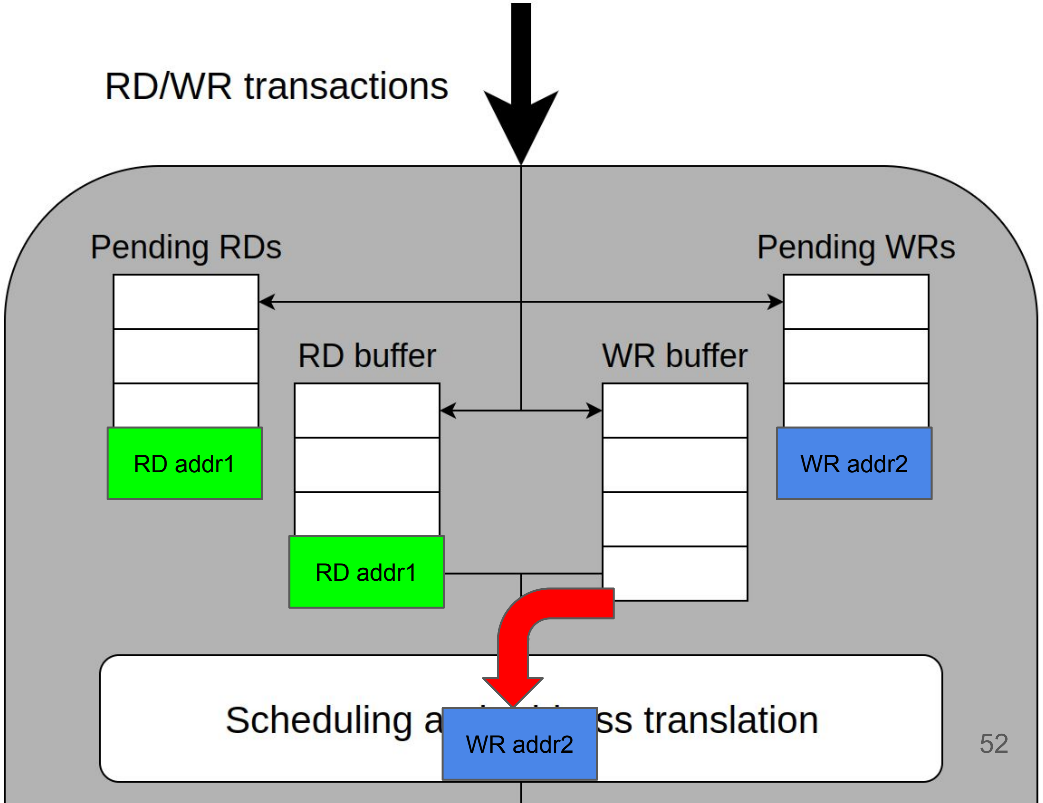
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**
- RD and WR buffers hold the **transactions to be scheduled** to DRAM
- For the purposes of this presentation, scheduling a transaction only implies **moving it from top to bottom for DRAM command generation**



Structure (logical) of a memory controller

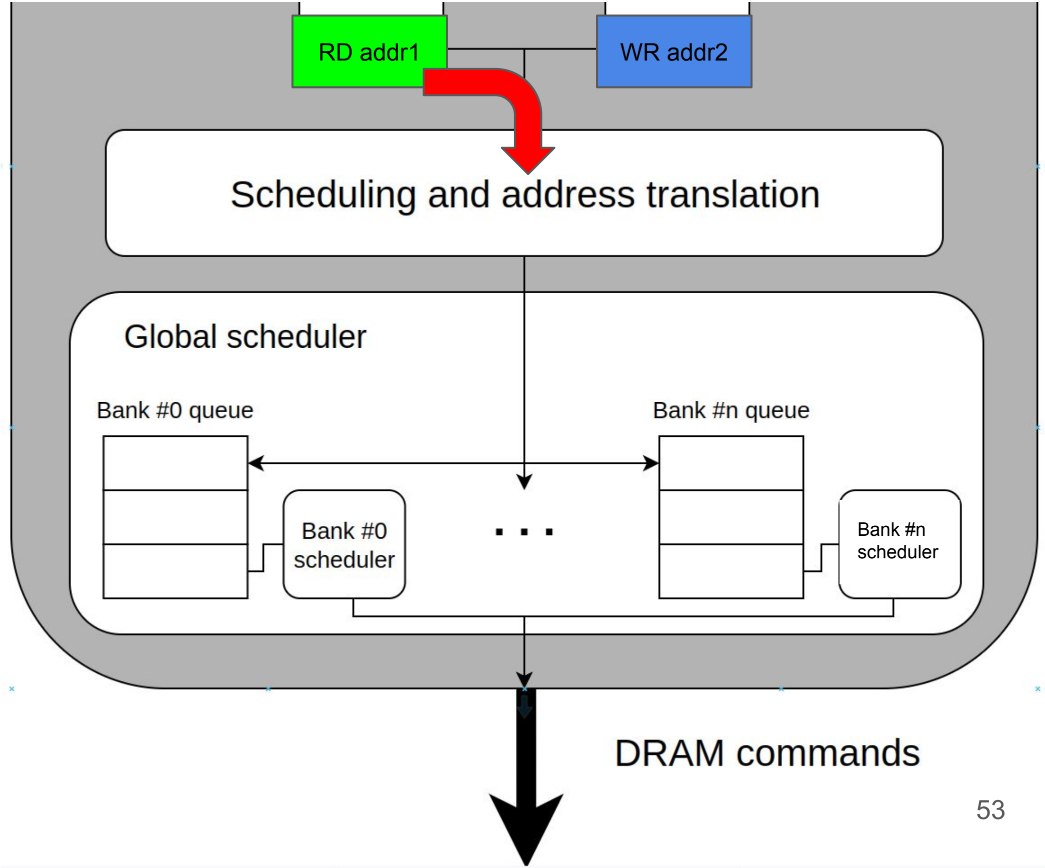
Let’s start with transaction management:

- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
- They **regulate** which transactions **can enter the MC**
- RD and WR buffers hold the **transactions to be scheduled** to DRAM
- For the purposes of this presentation, scheduling a transaction only implies **moving it from top to bottom for DRAM command generation**



Structure (logical) of a memory controller

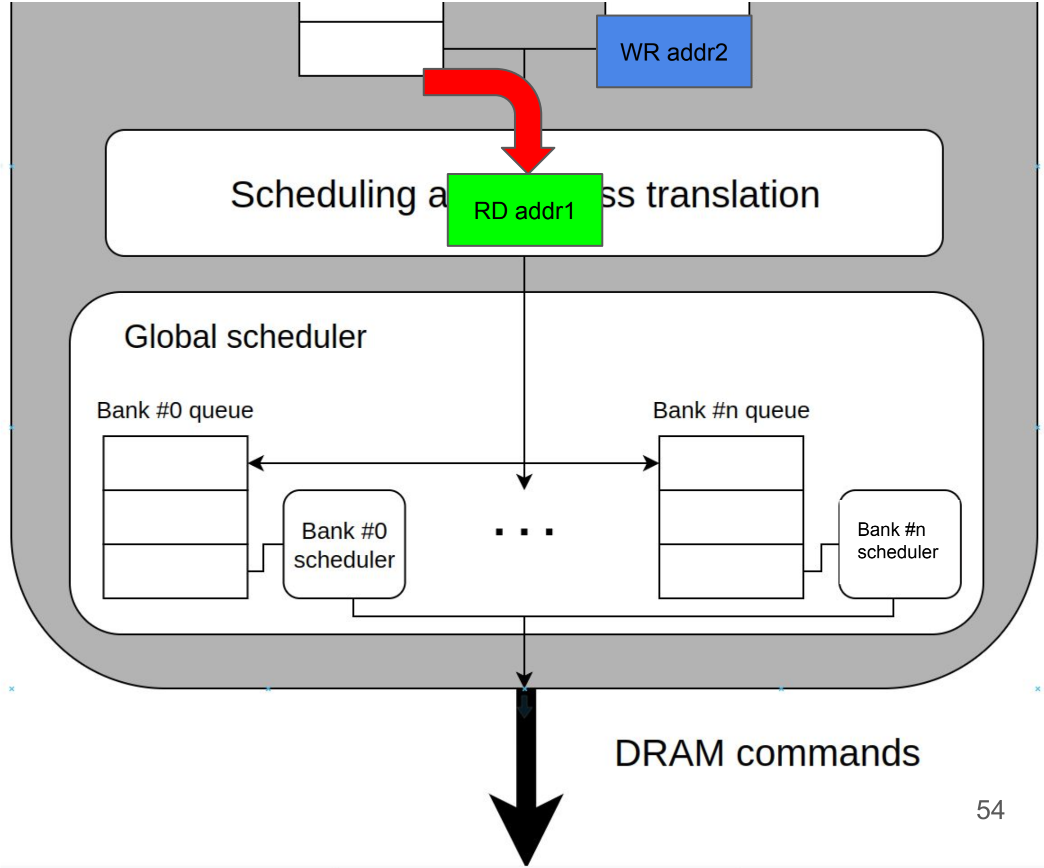
Let’s go down to DRAM command generation:



Structure (logical) of a memory controller

Let’s go down to DRAM command generation:

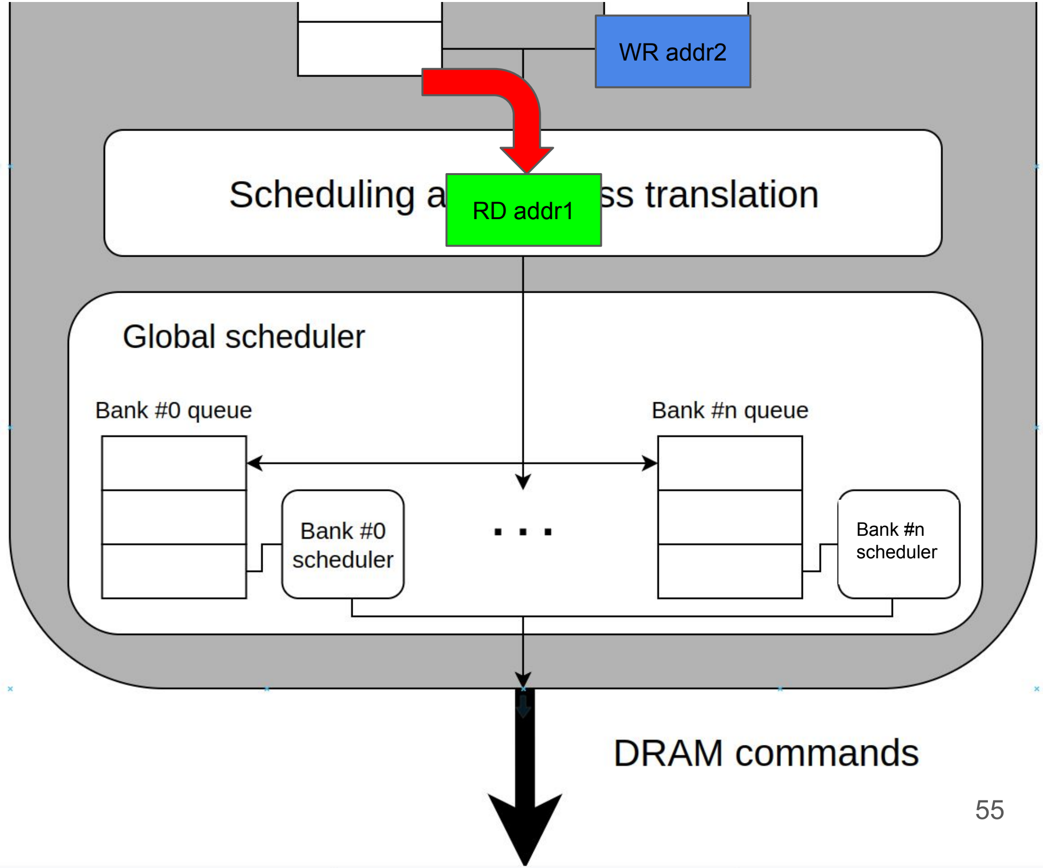
- Scheduling moves stuff from top to bottom, that much is clear



Structure (logical) of a memory controller

Let’s go down to DRAM command generation:

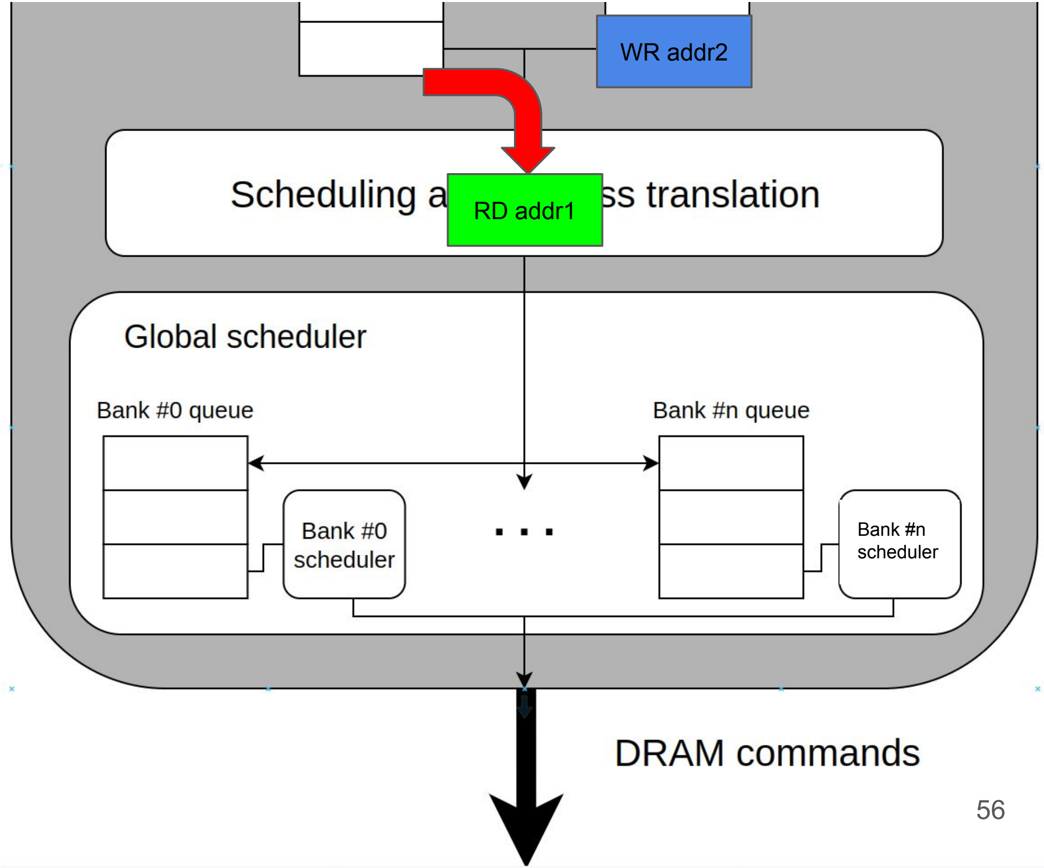
- Scheduling moves stuff from top to bottom, that much is clear
- Address translation **places transactions into bank queues** according to their **target address**



Structure (logical) of a memory controller

Let’s go down to DRAM command generation:

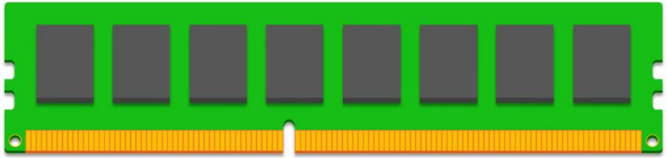
- Scheduling moves stuff from top to bottom, that much is clear
- Address translation **places transactions into bank queues** according to their **target address**
- Addresses are divided into segments defining the **channel, rank, bank, row and column** that the transaction is accessing

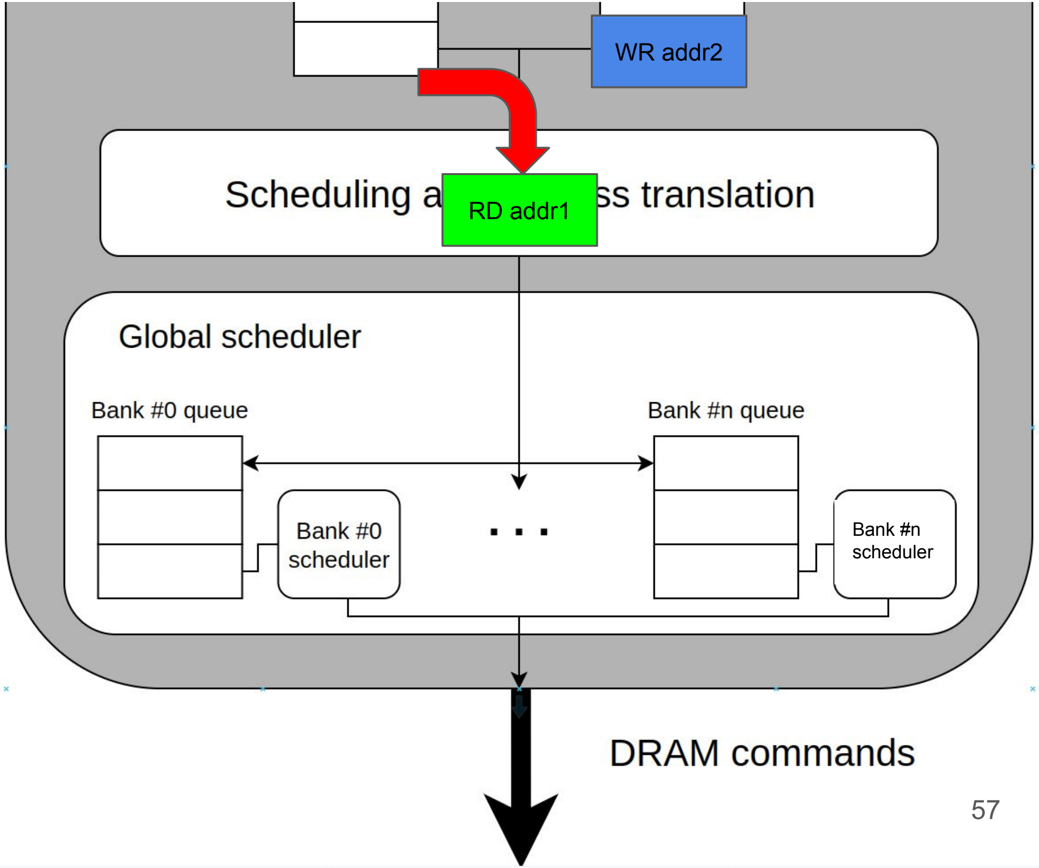


Structure (logical) of a memory controller

Let's go down to DRAM command generation:

- Scheduling moves stuff from top to bottom, that much is clear
- Address translation **places transactions into bank queues** according to their **target address**
- Addresses are divided into segments defining the **channel, rank, bank, row and column** that the transaction is accessing
- This is called the address mapping of the MC

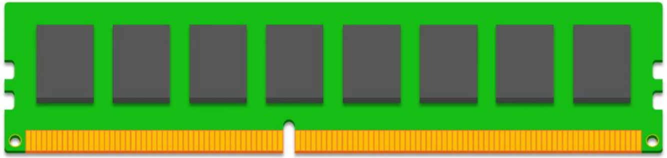


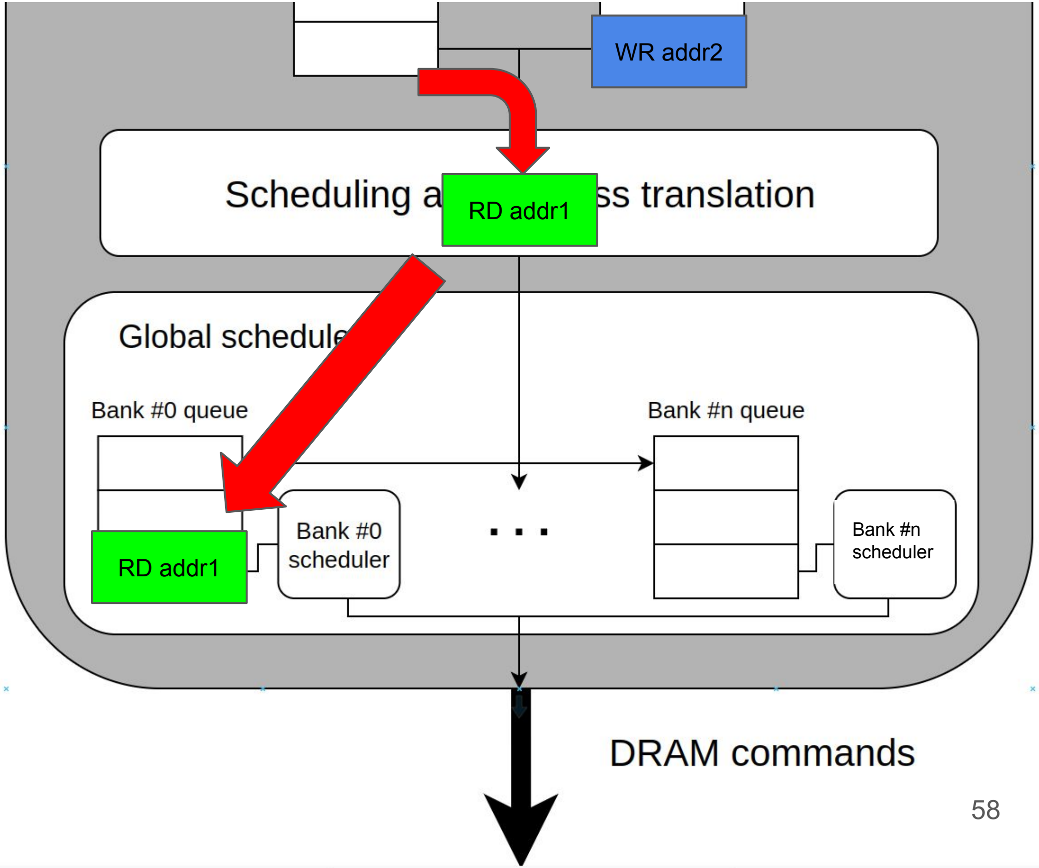


Structure (logical) of a memory controller

Let’s go down to DRAM command generation:

- Scheduling moves stuff from top to bottom, that much is clear
- Address translation **places transactions into bank queues** according to their **target address**
- Addresses are divided into segments defining the **channel, rank, bank, row and column** that the transaction is accessing
- This is called the address mapping of the MC

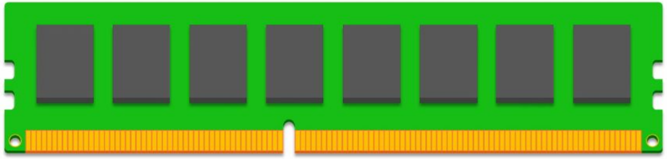


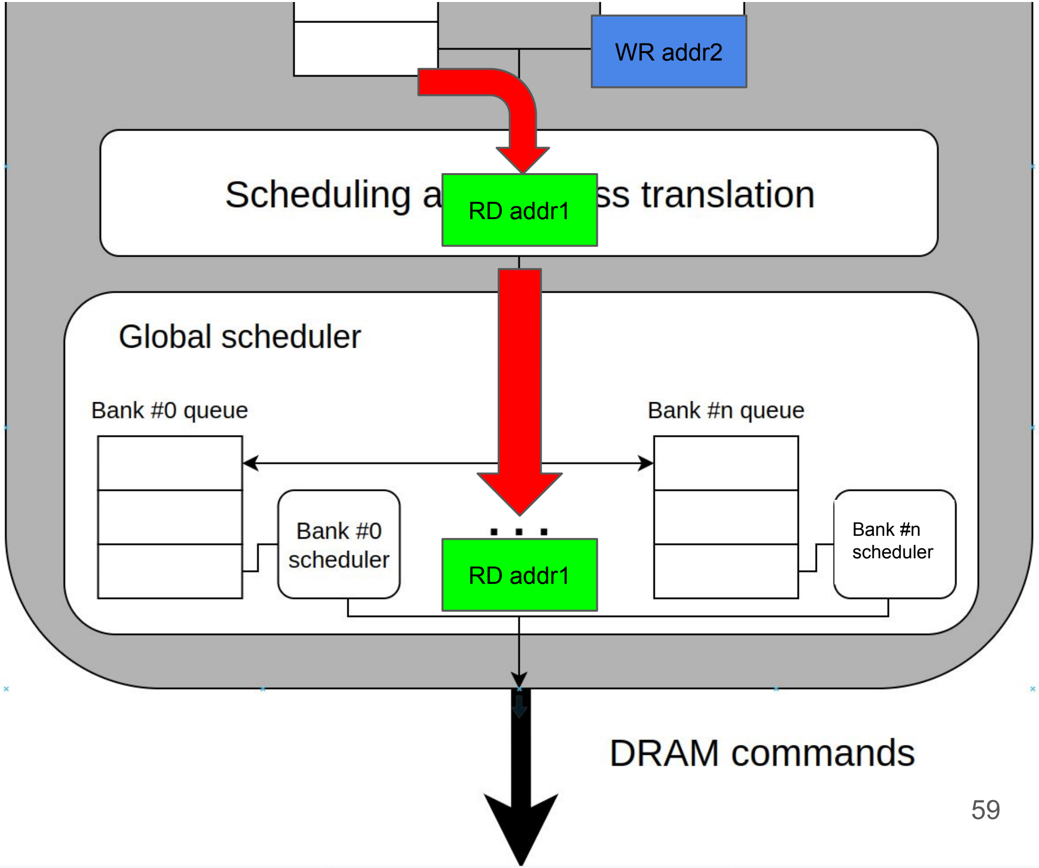


Structure (logical) of a memory controller

Let’s go down to DRAM command generation:

- Scheduling moves stuff from top to bottom, that much is clear
- Address translation **places transactions into bank queues** according to their **target address**
- Addresses are divided into segments defining the **channel, rank, bank, row and column** that the transaction is accessing
- This is called the address mapping of the MC

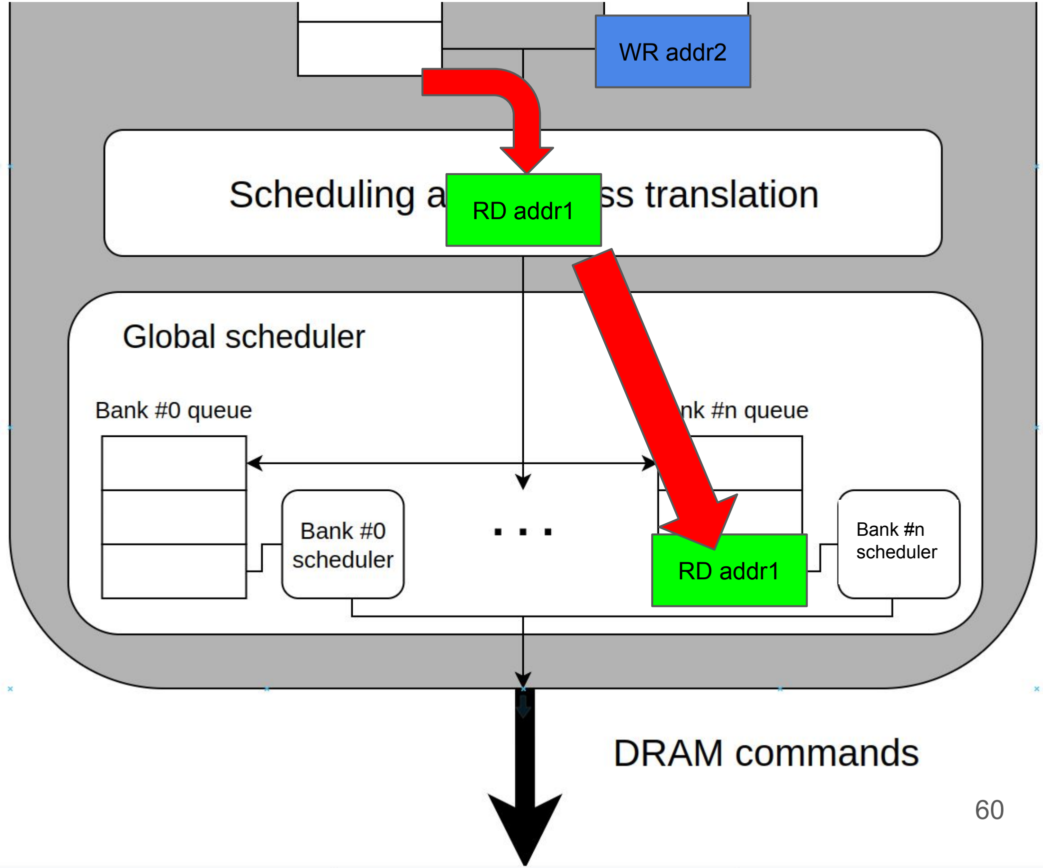


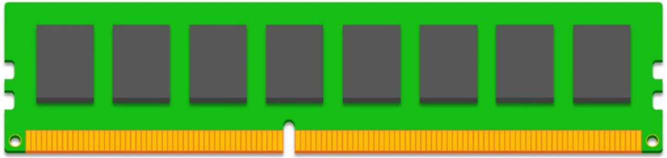


Structure (logical) of a memory controller

Let’s go down to DRAM command generation:

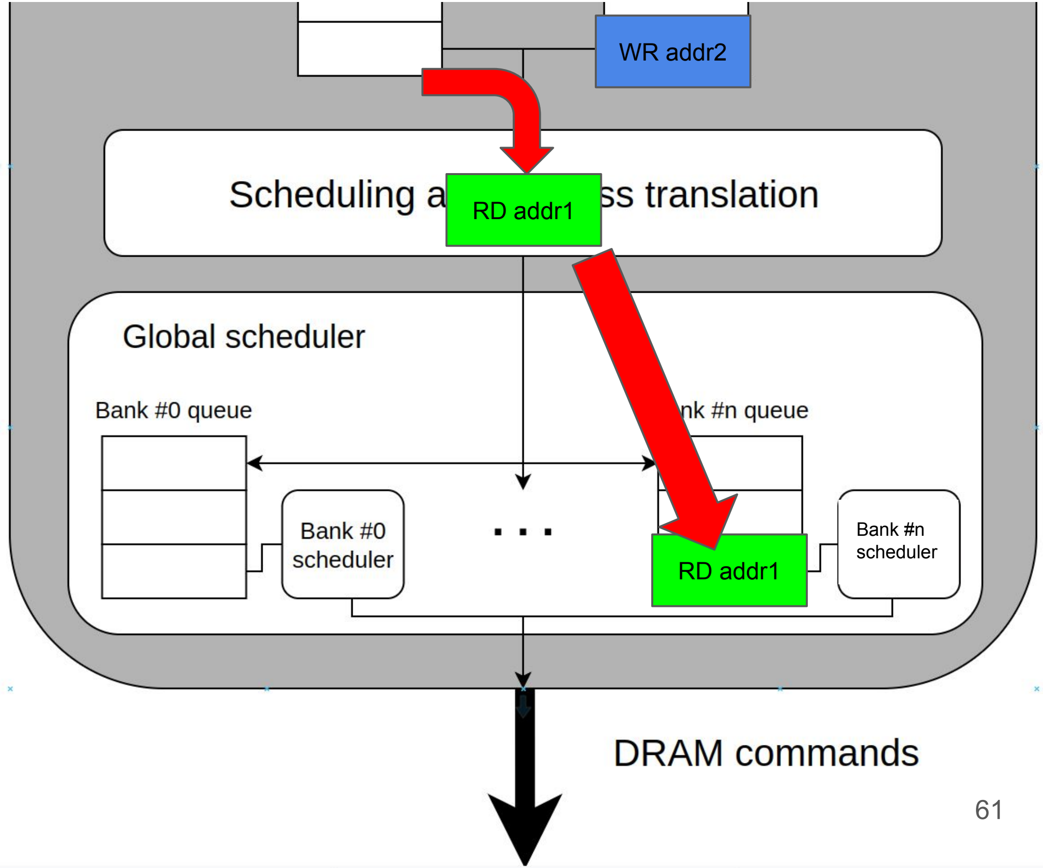
- Scheduling moves stuff from top to bottom, that much is clear
- Address translation **places transactions into bank queues** according to their **target address**
- Addresses are divided into segments defining the **channel, rank, bank, row and column** that the transaction is accessing
- This is called the address mapping of the MC





Structure (logical) of a memory controller

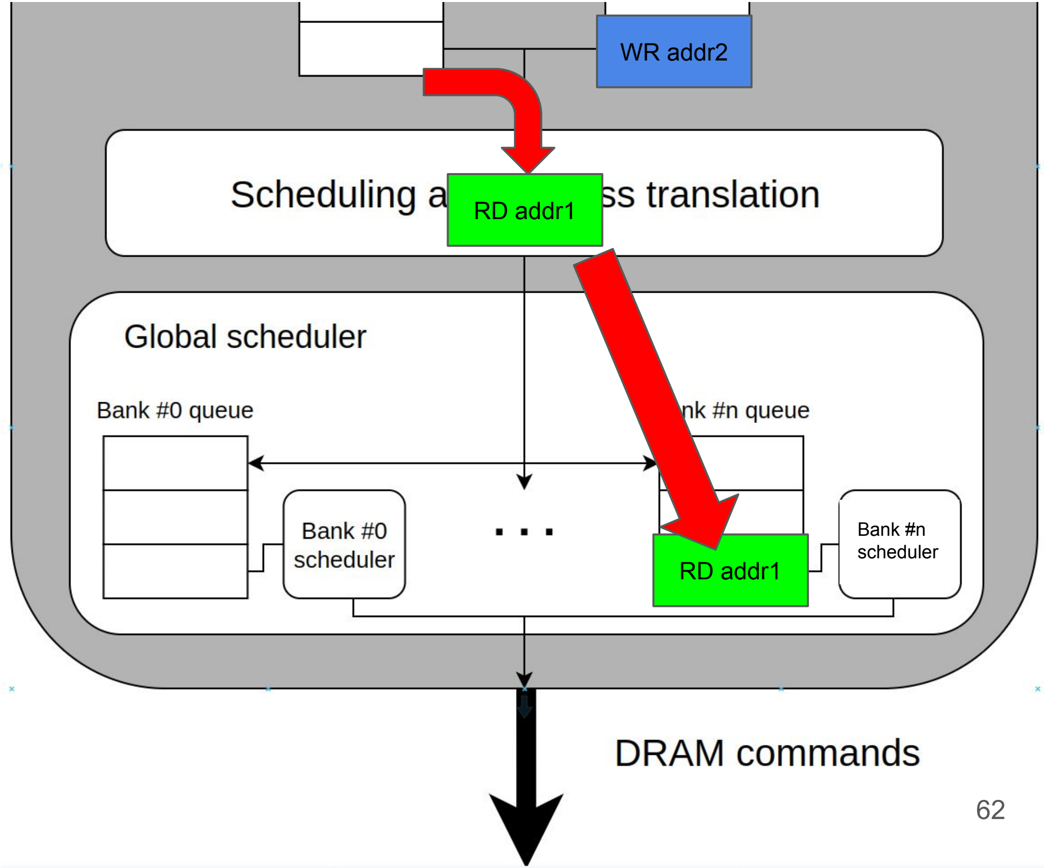
Where are the DRAM commands though??



Structure (logical) of a memory controller

Where are the DRAM commands though??

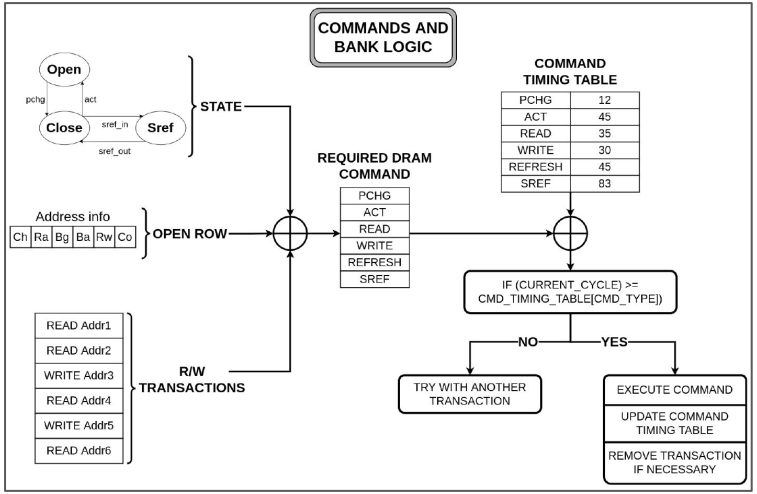
- This is where the bank schedulers come in…



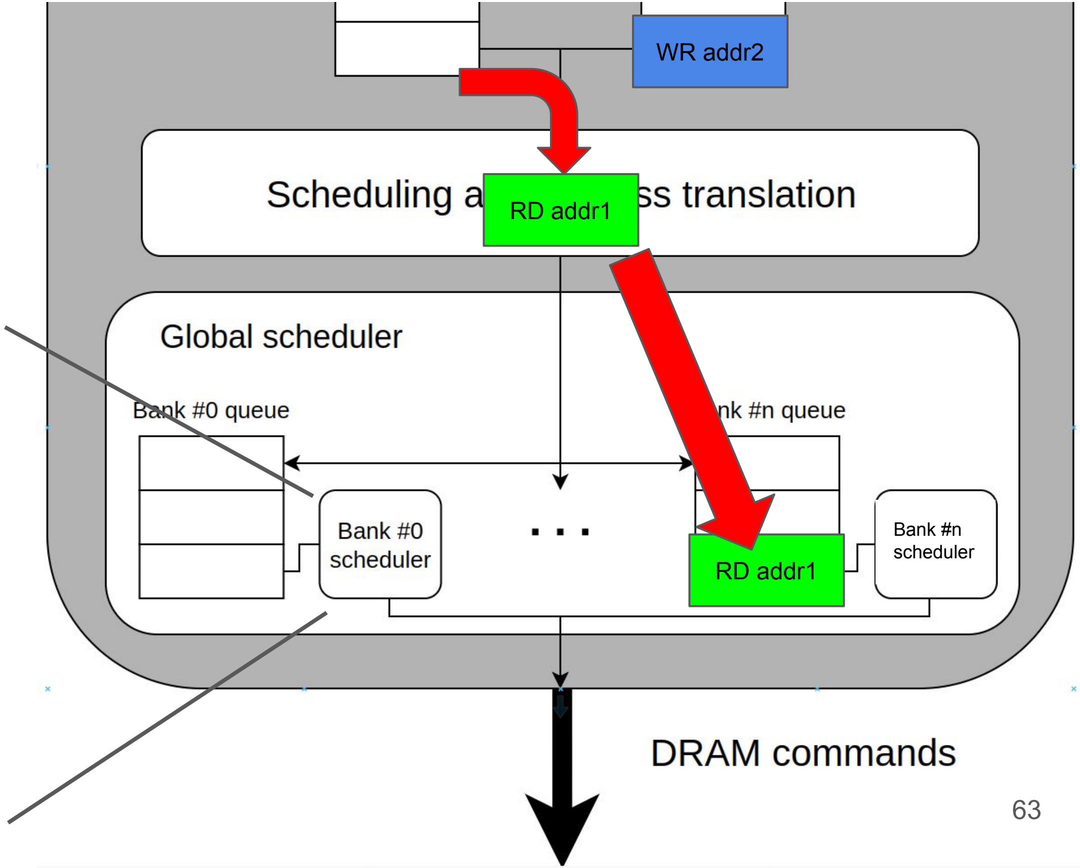
Structure (logical) of a memory controller

Where are the DRAM commands though??

- This is where the bank schedulers come in…



PCHG	12
ACT	45
READ	35
WRITE	30
REFRESH	45
SREF	83

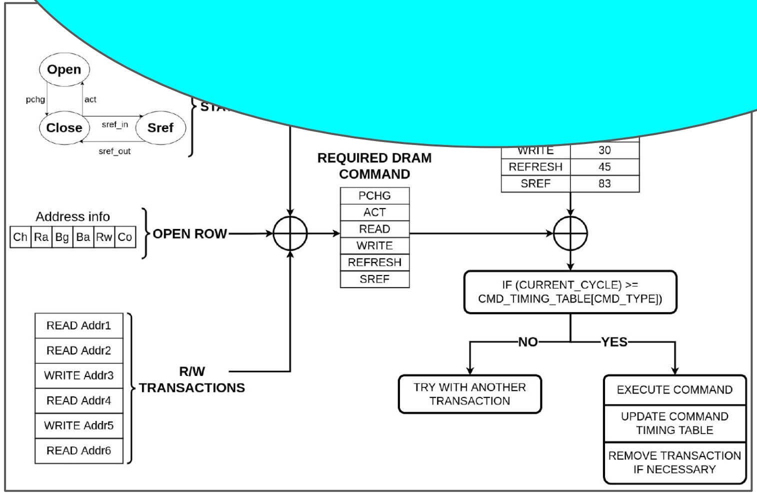


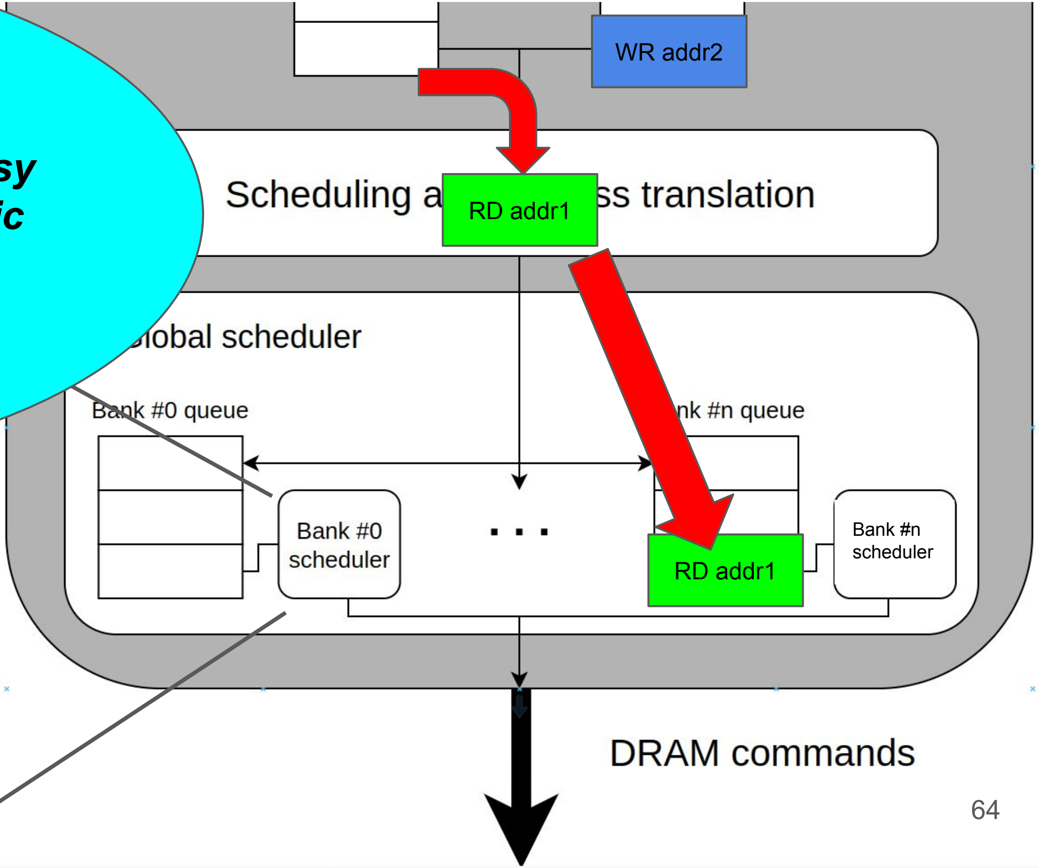
Structure (logical) of a memory controller

Whe

●

We won't be looking into the messy details of the bank scheduler logic today…



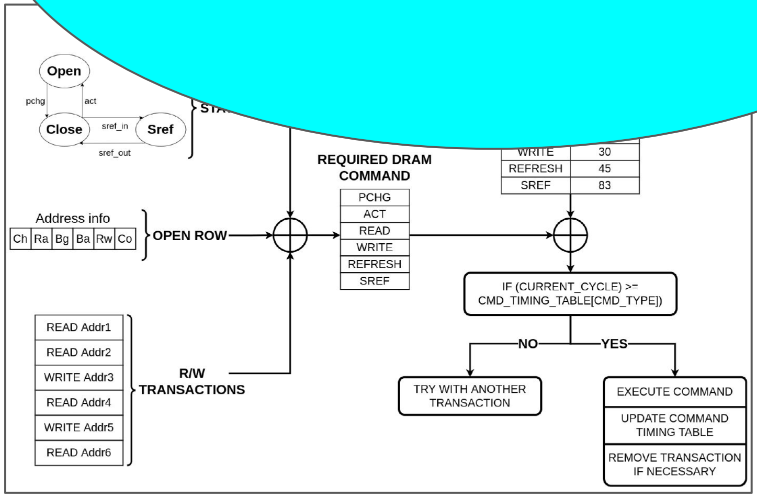


Structure (logical) of a memory controller

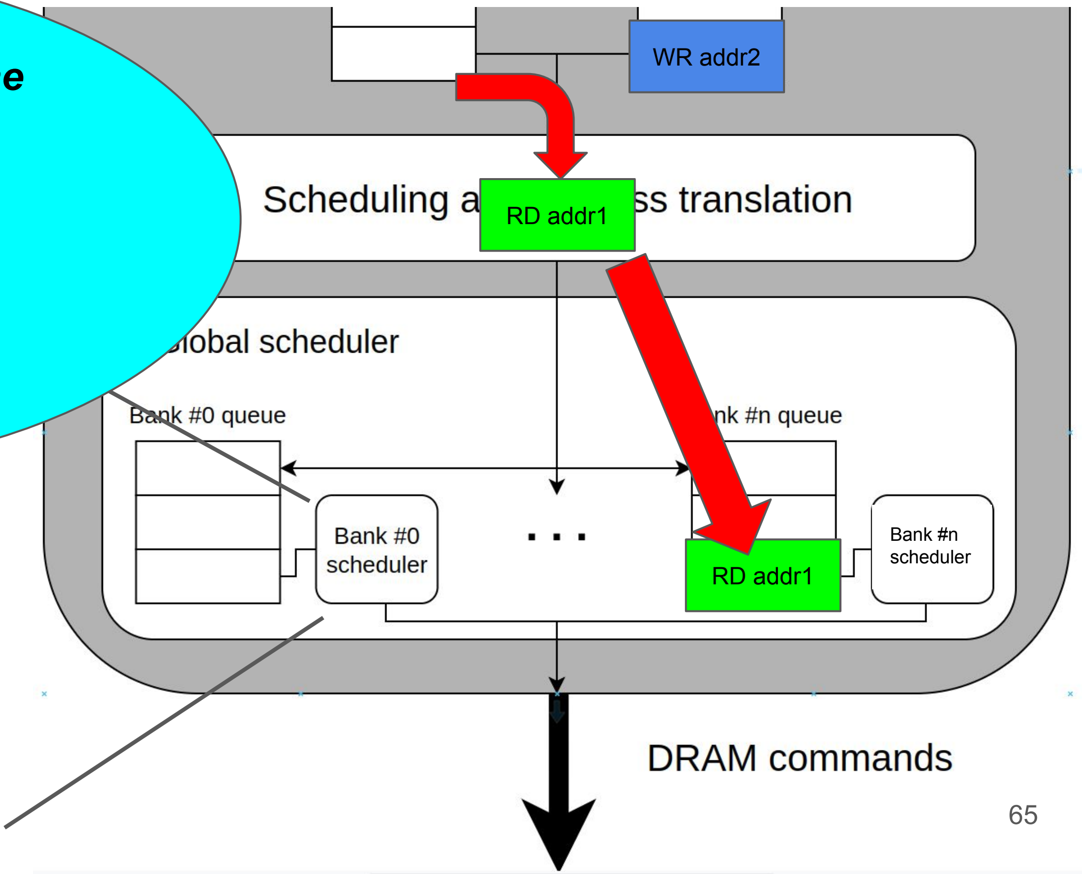
Instead, we'll just consider the following relevant points:

Whe

-



WRITE	30
REFRESH	45
SREF	83



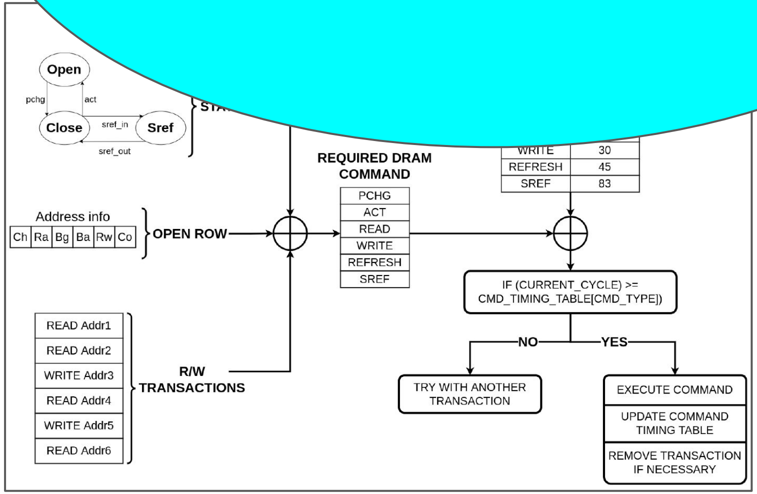
Structure (logical) of a memory controller

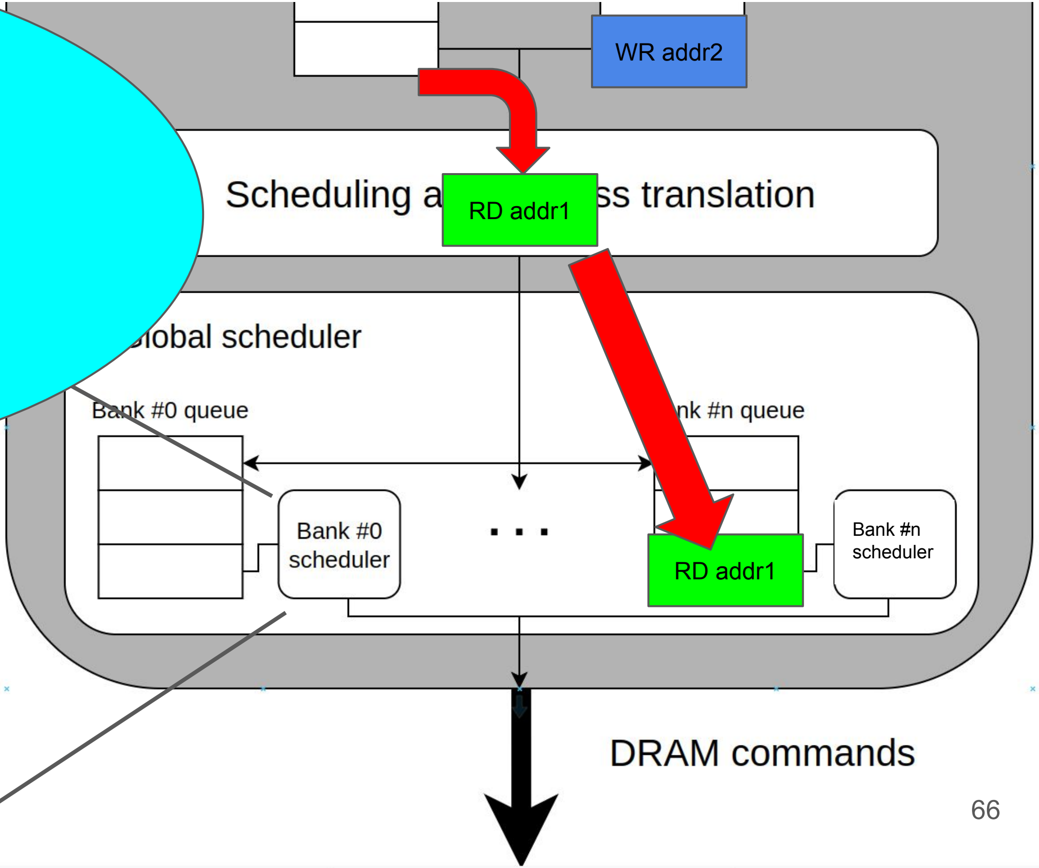
Whe

●

Instead, we'll just consider the following relevant points:

1. *Scheduler transforms requests into DRAM commands*



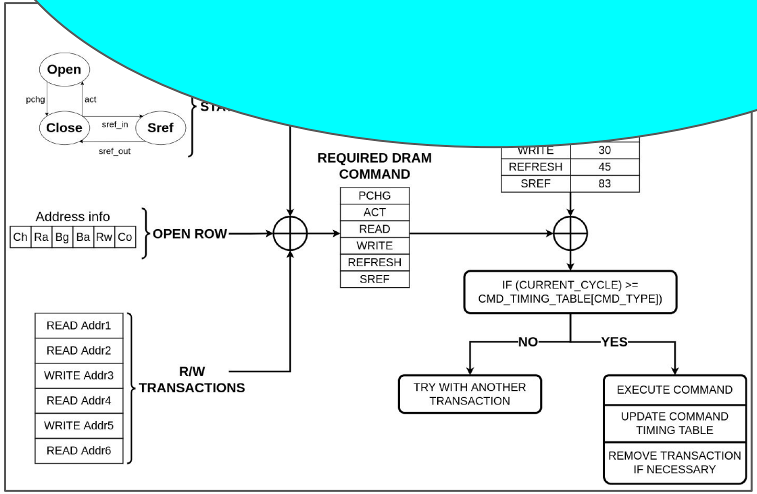


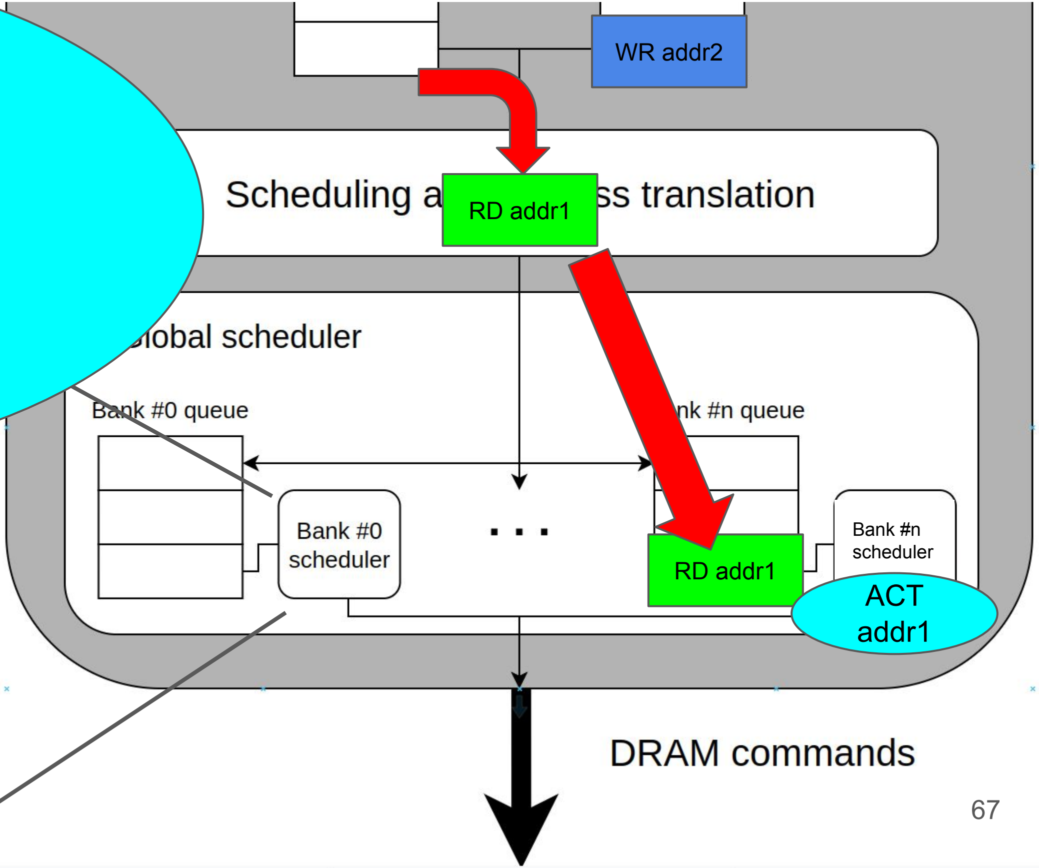
Structure (logical) of a memory controller

Whe

Instead, we'll just consider the following relevant points:

1. *Scheduler transforms requests into DRAM commands*





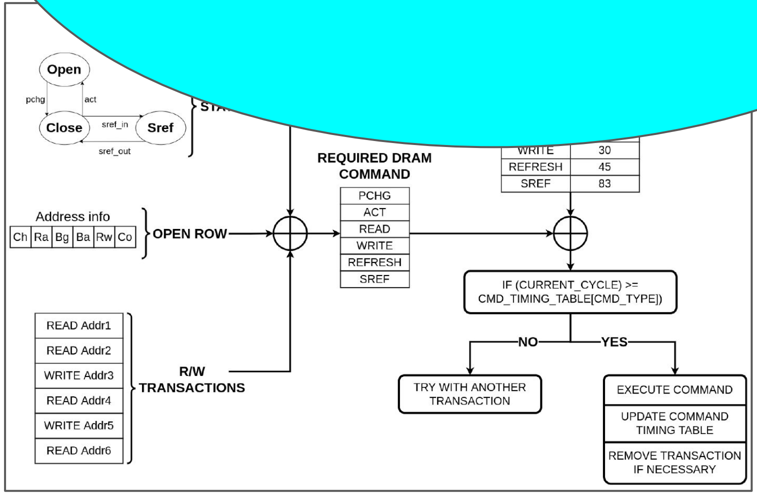
Structure (logical) of a memory controller

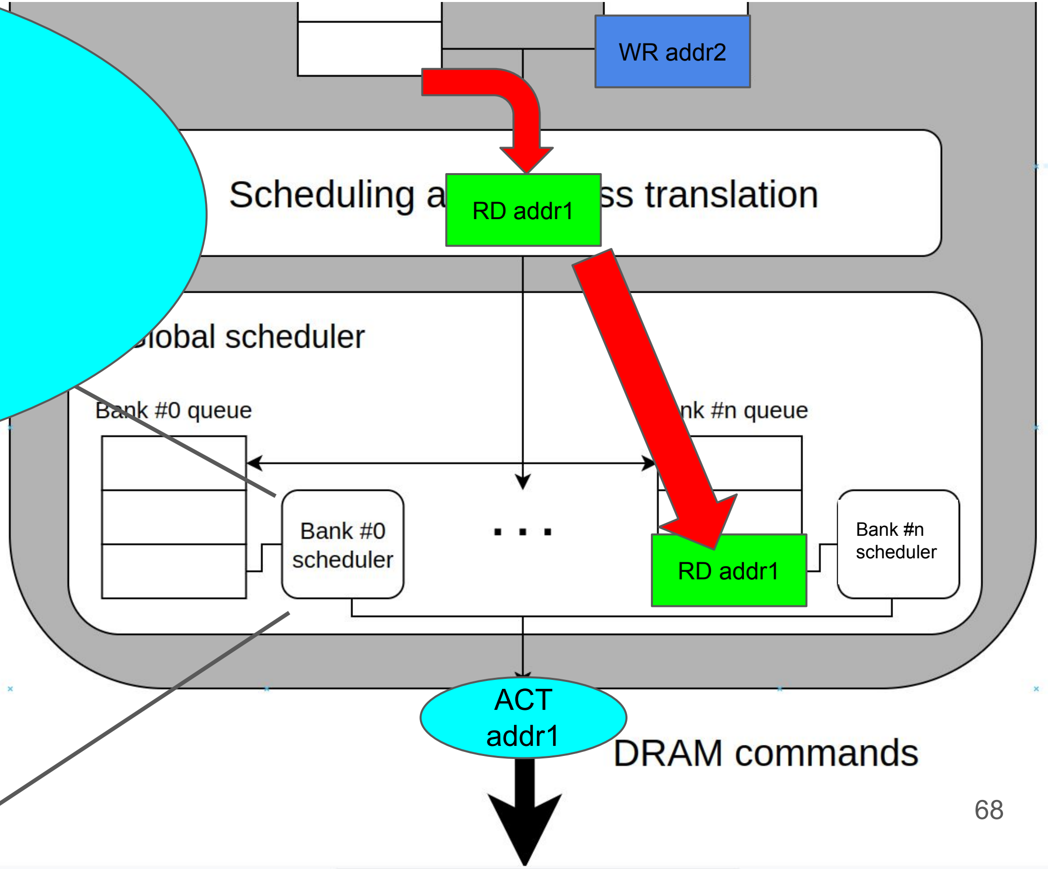
Whe

●

Instead, we'll just consider the following relevant points:

1. *Scheduler transforms requests into DRAM commands*





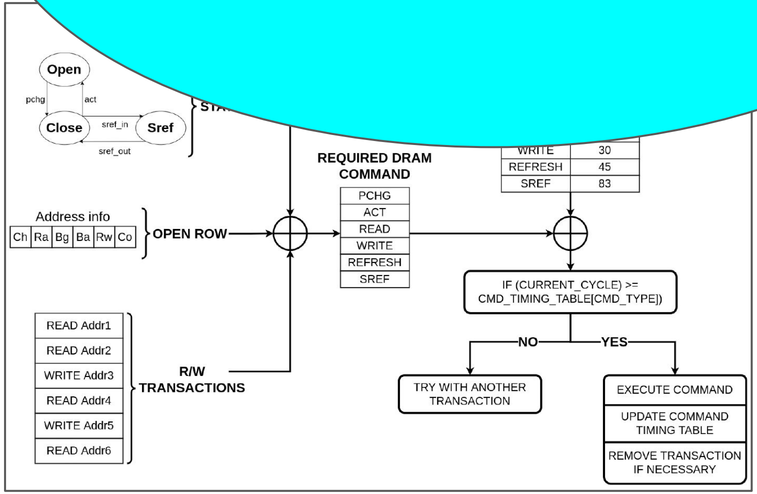
Structure (logical) of a memory controller

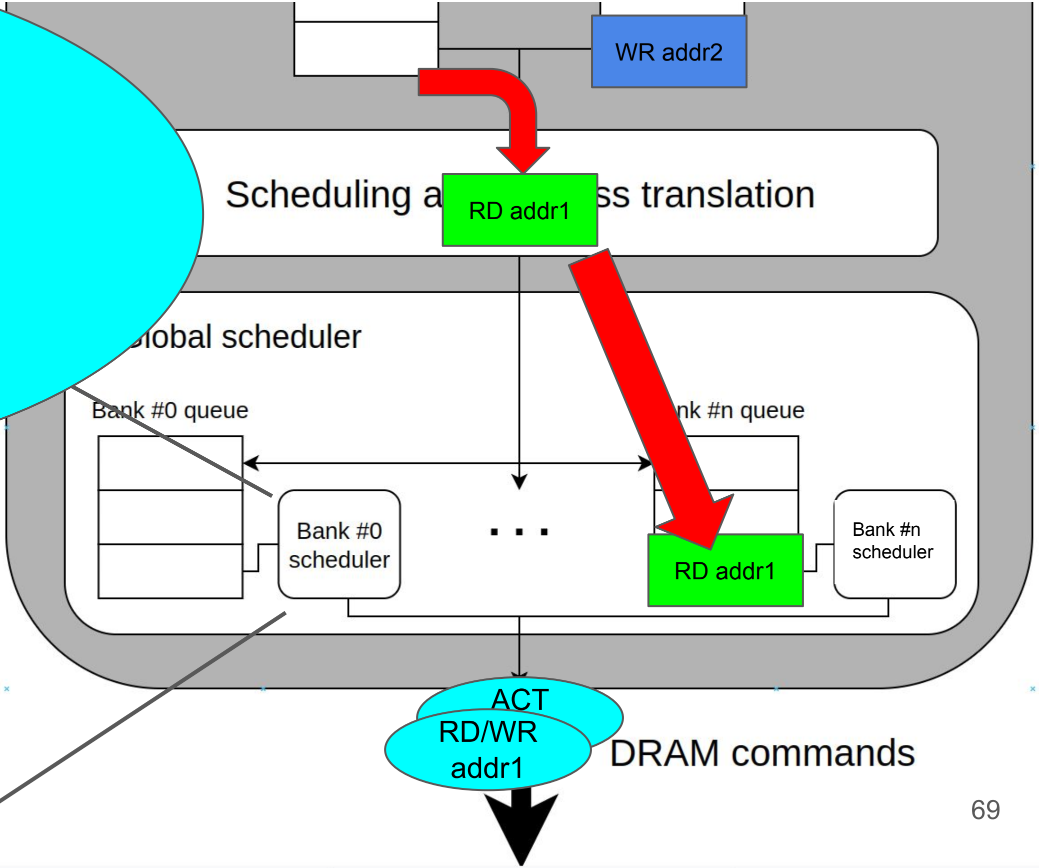
Whe

●

Instead, we'll just consider the following relevant points:

1. *Scheduler transforms requests into DRAM commands*





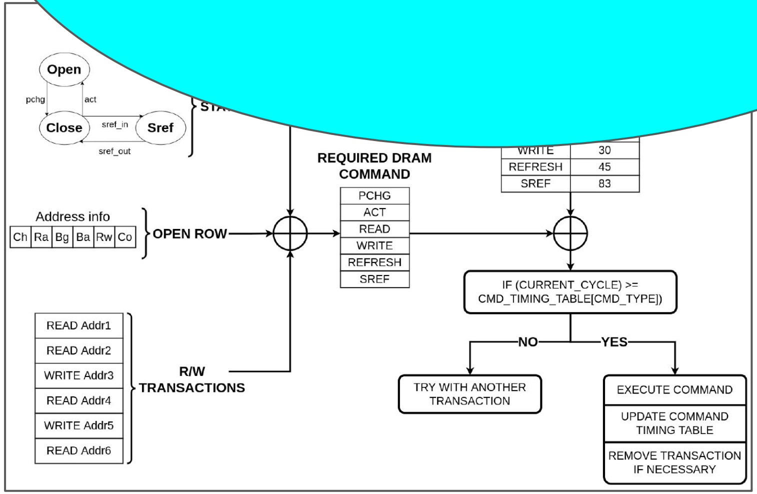
Structure (logical) of a memory controller

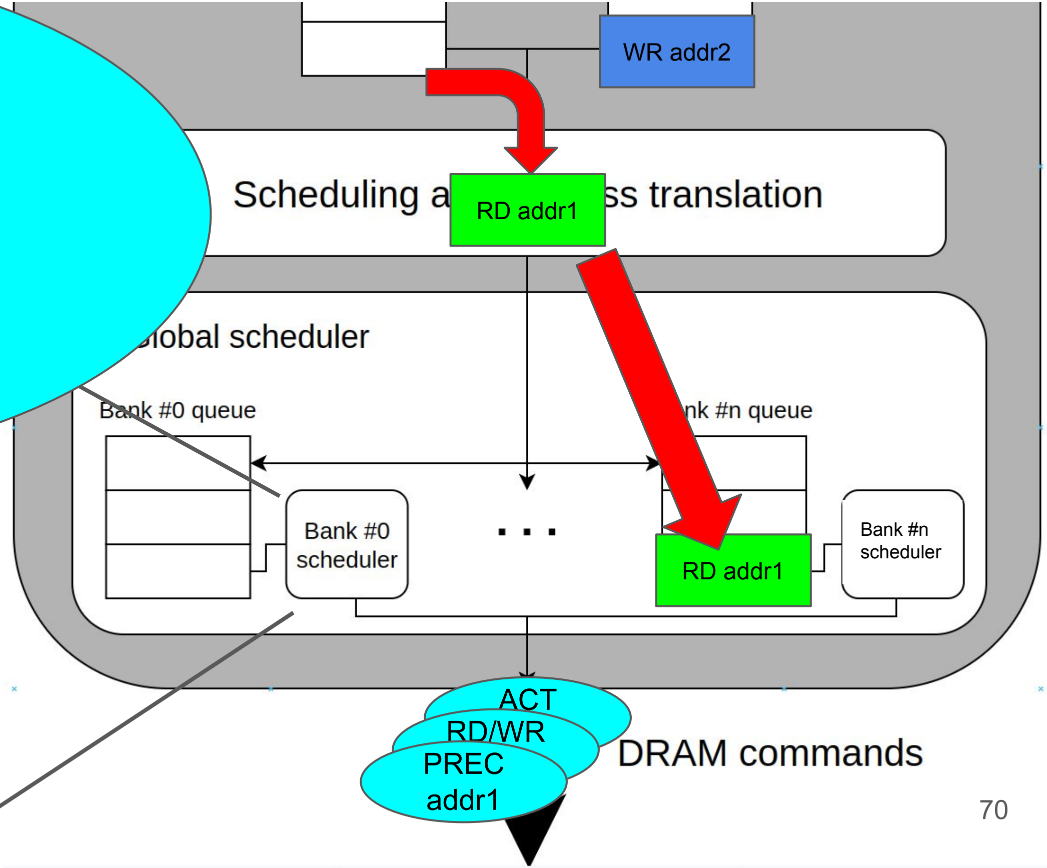
Whe

●

Instead, we'll just consider the following relevant points:

1. *Scheduler transforms requests into DRAM commands*



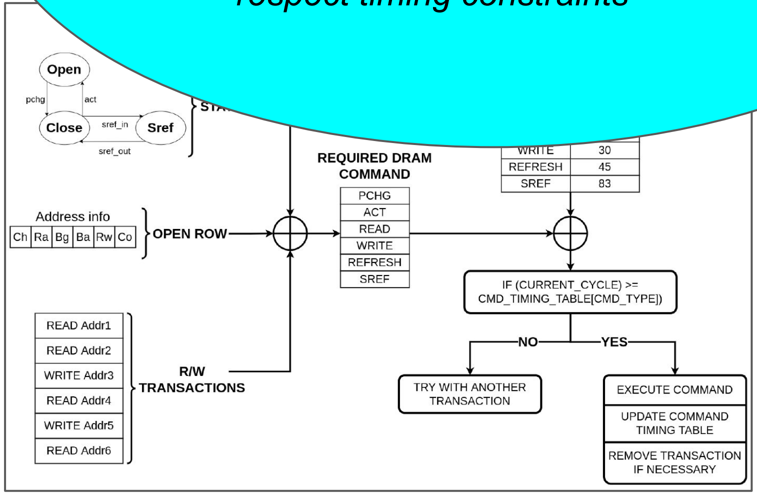


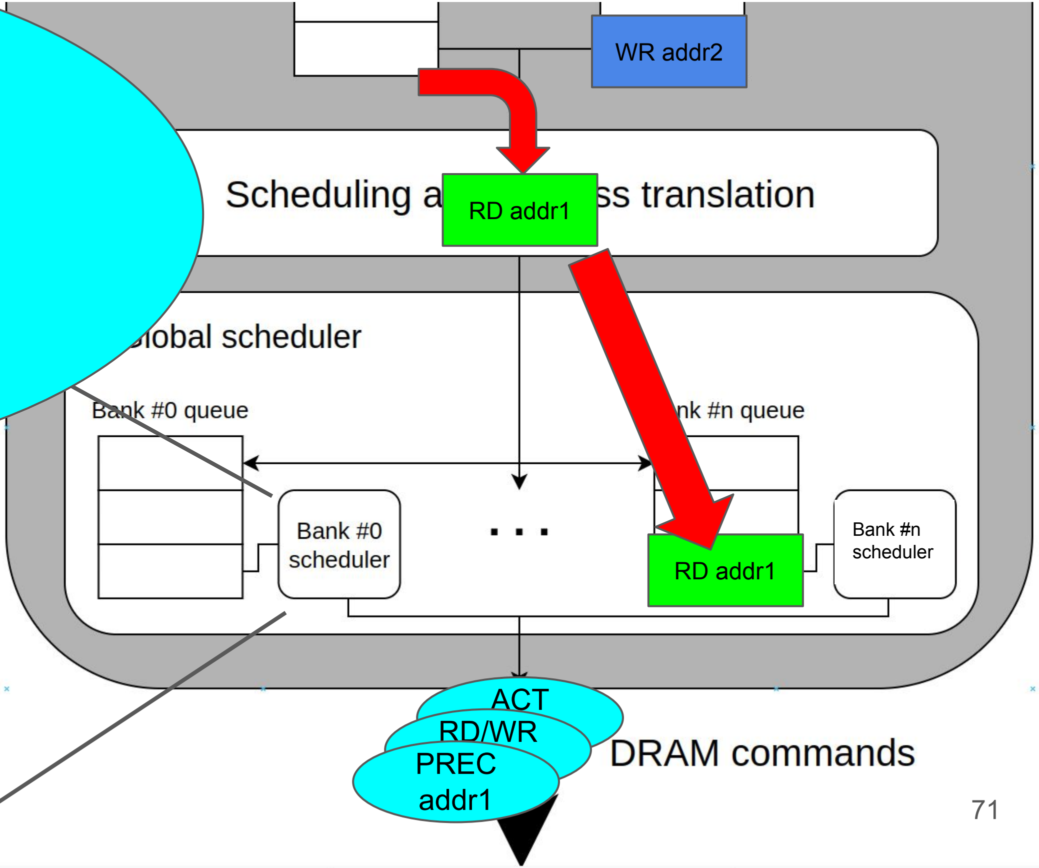
Structure (logical) of a memory controller

Whe

Instead, we'll just consider the following relevant points:

1. *Scheduler transforms requests into DRAM commands*
2. *DRAM command generation must respect timing constraints*





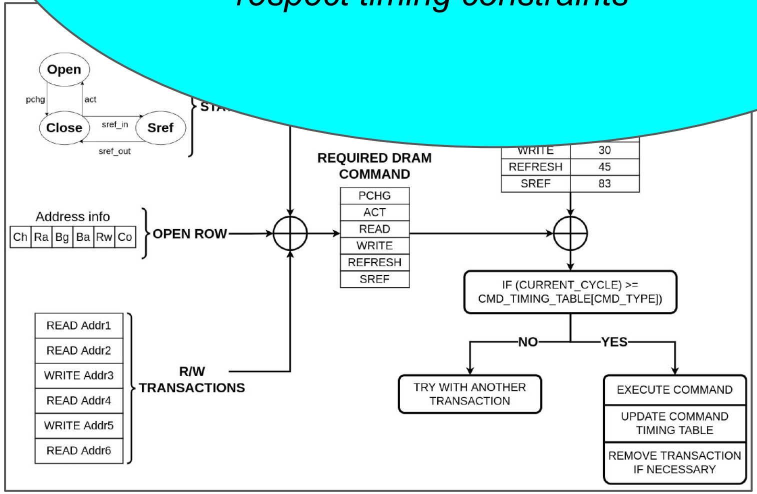
Structure (logical) of a memory controller

Whe

●

Instead, we'll just consider the following relevant points:

1. *Scheduler transforms requests into DRAM commands*
2. *DRAM command generation must respect timing constraints*



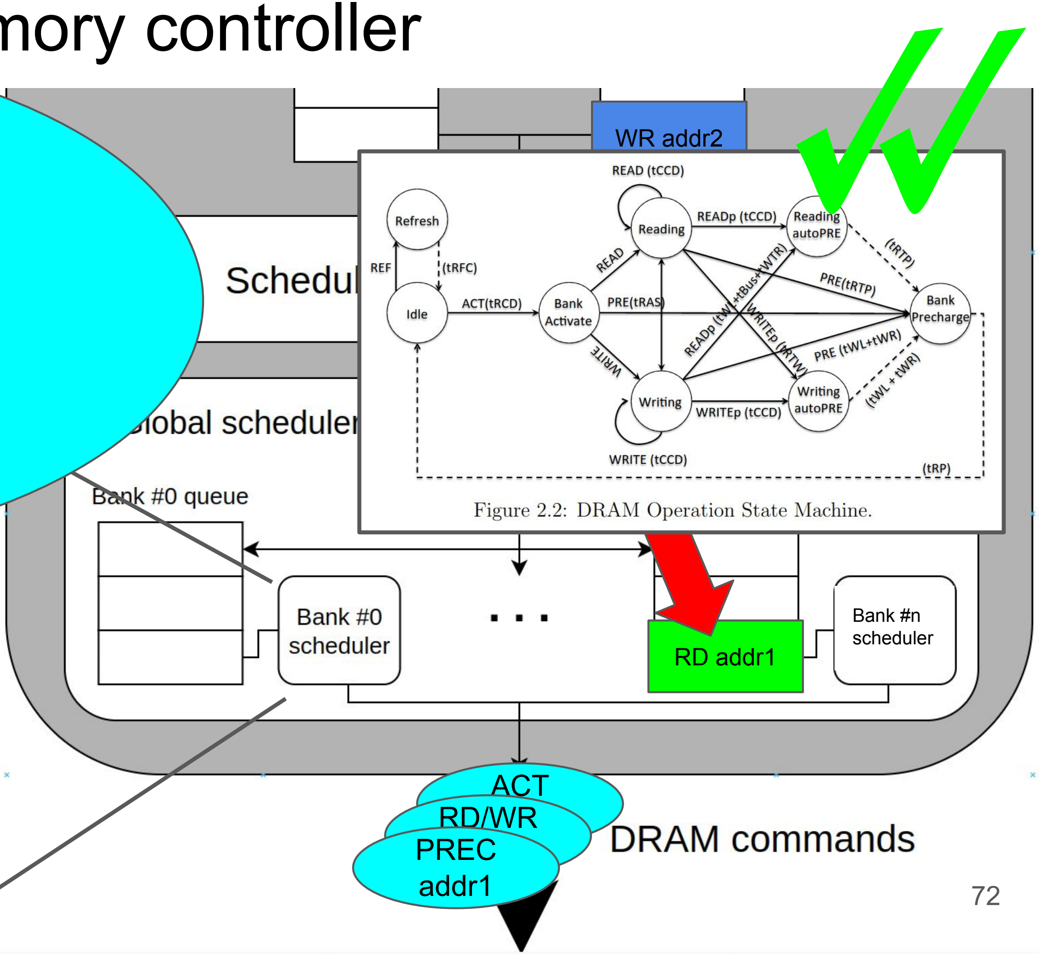


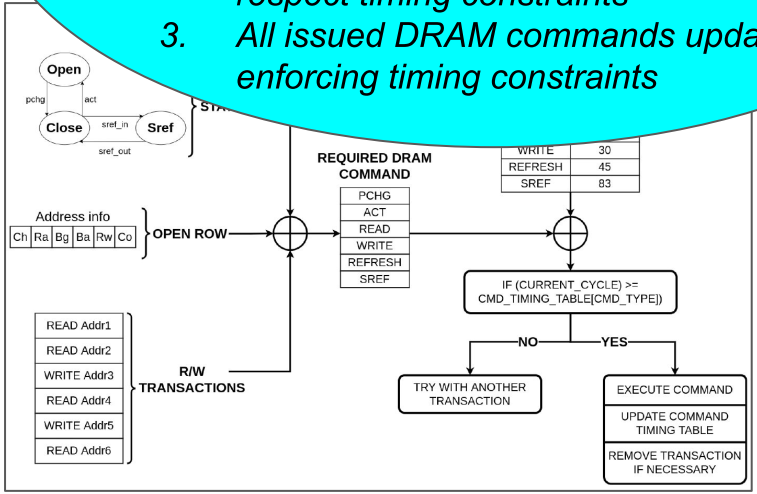
Figure 2.2: DRAM Operation State Machine.

Structure (logical) of a memory controller

Whe

Instead, we'll just consider the following relevant points:

1. *Scheduler transforms requests into DRAM commands*
2. *DRAM command generation must respect timing constraints*
3. *All issued DRAM commands update the enforcing timing constraints*



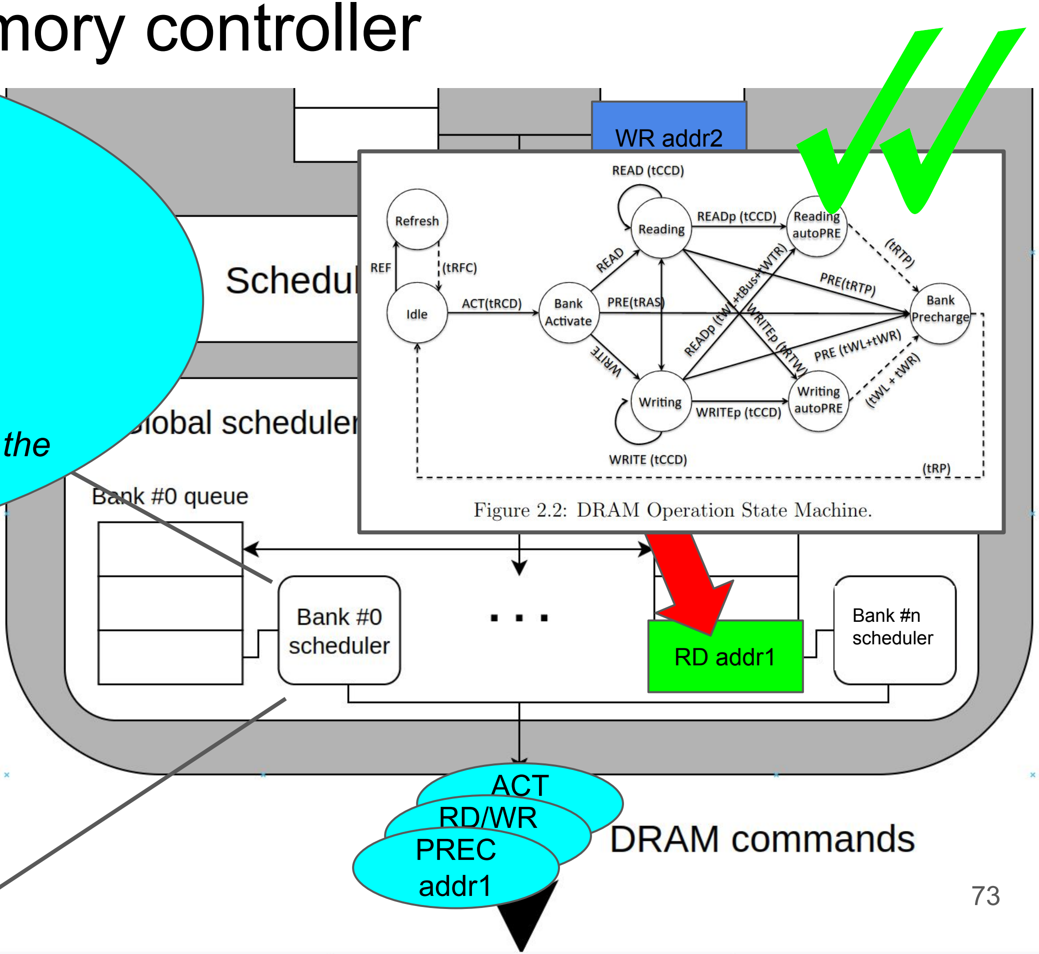
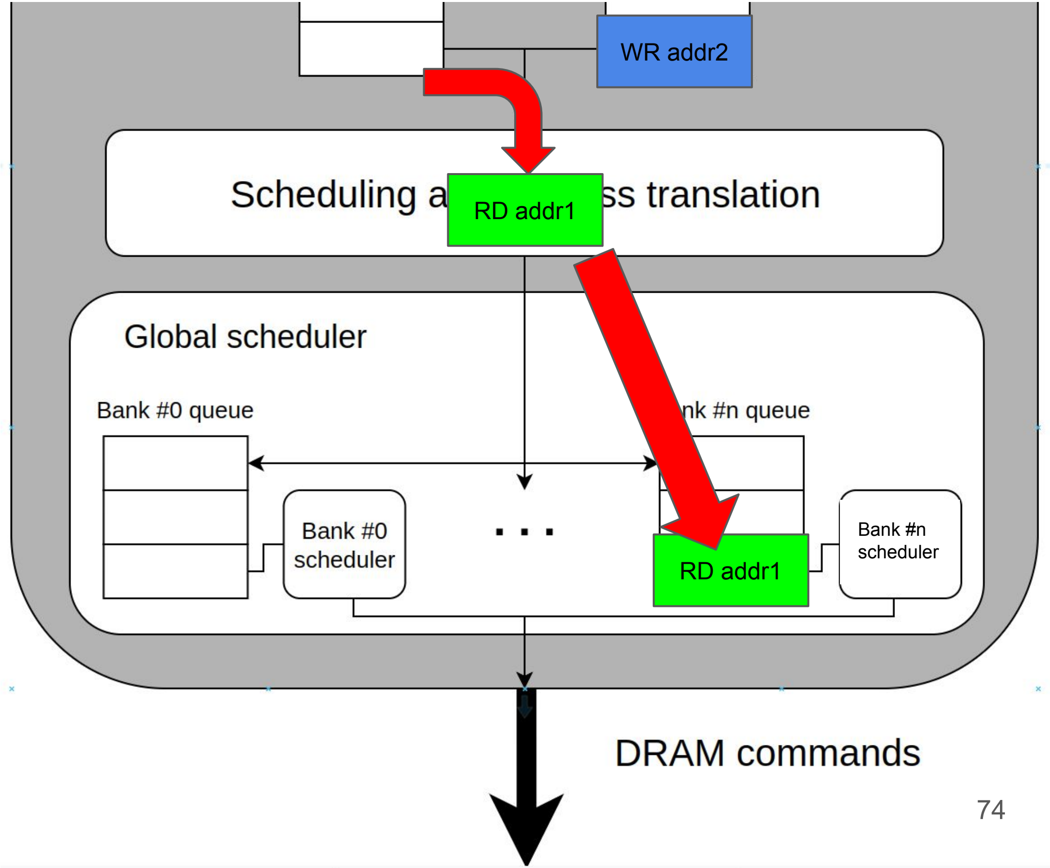


Figure 2.2: DRAM Operation State Machine.

Structure (logical) of a memory controller

Current clock value:
N+1

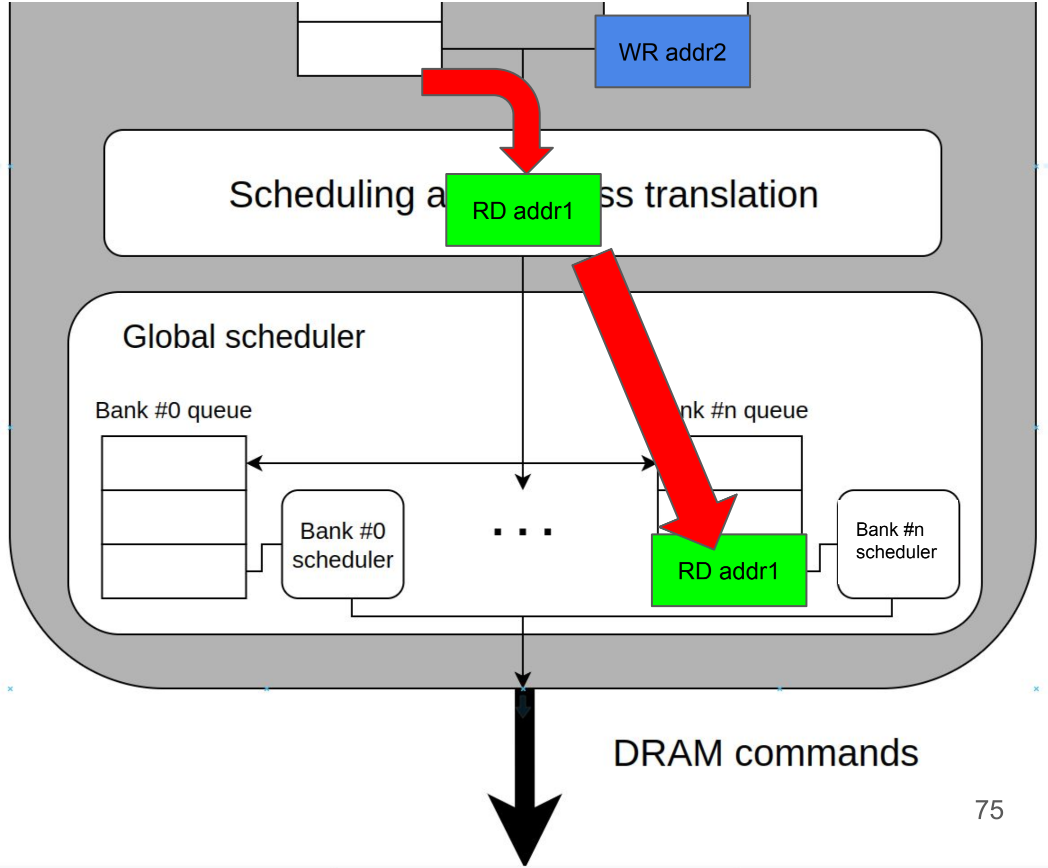
DRAM cmd type	OK clock cycle
PCHG	…
ACT	N
READ	…
WRITE	…
REFRESH	…
SREF	…



Structure (logical) of a memory controller

Current clock value: N+1

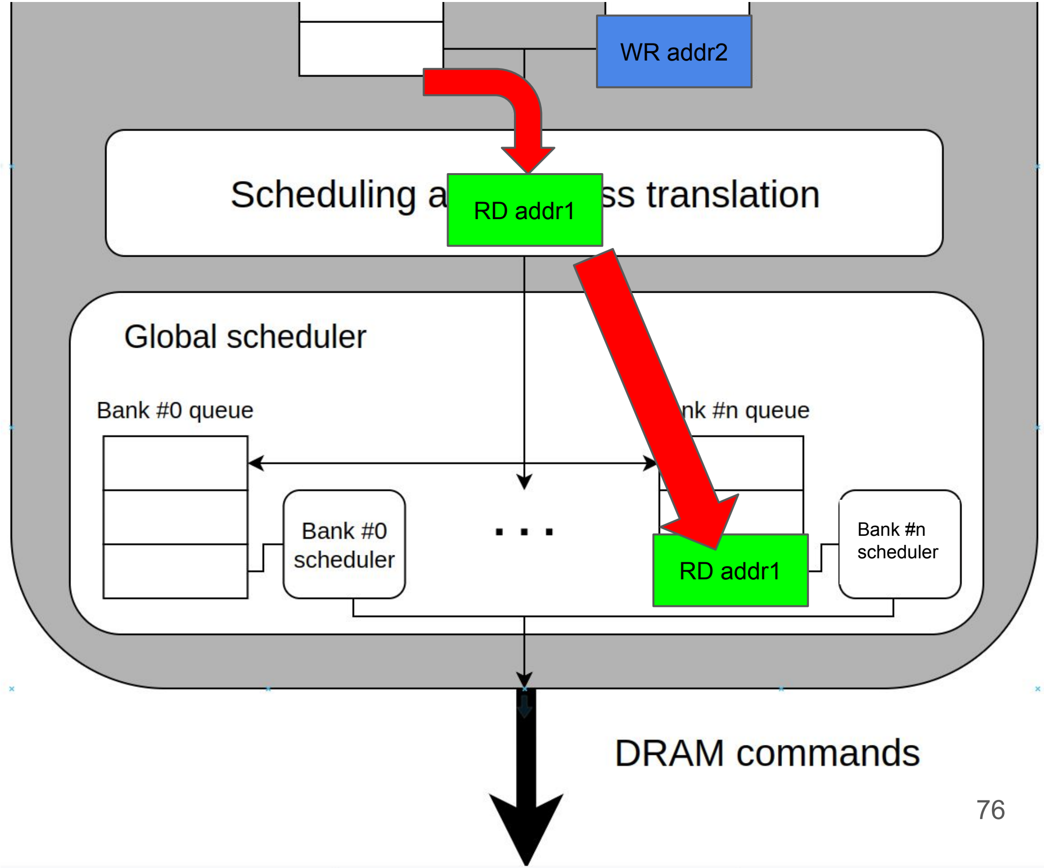
DRAM cmd type	OK clock cycle
PCHG	…
ACT	N
READ	…
WRITE	…
REFRESH	…
SREF	…



Structure (logical) of a memory controller

Current clock value: N+1

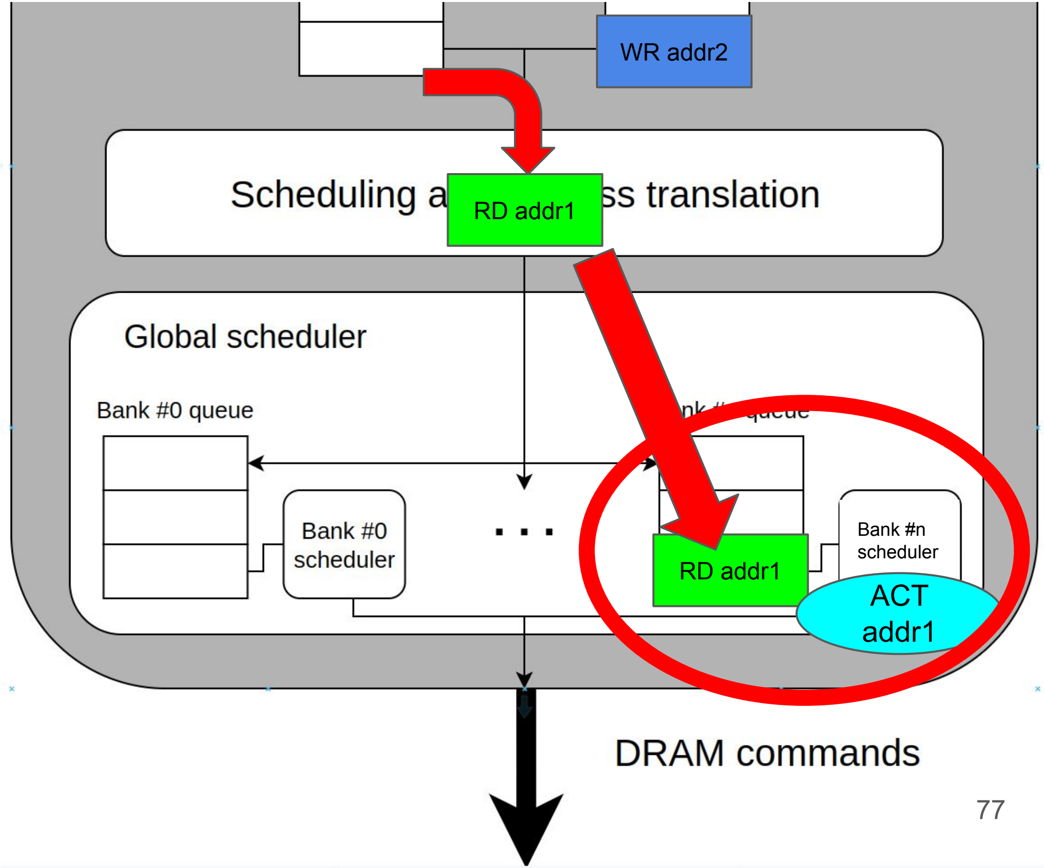
DRAM cmd type	OK clock cycle
PCHG	…
ACT	N
READ	…
WRITE	…
REFRESH	…
SREF	…



Structure (logical) of a memory controller

Current clock value: N+1

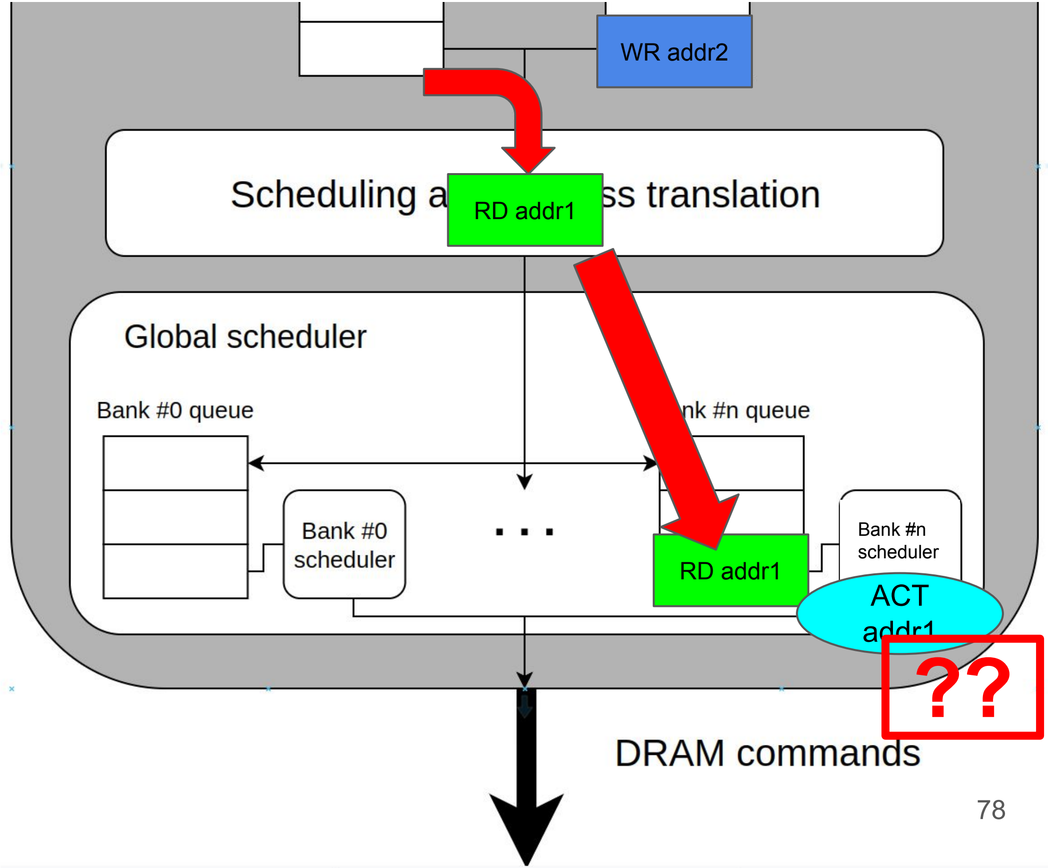
DRAM cmd type	OK clock cycle
PCHG	…
ACT	N
READ	…
WRITE	…
REFRESH	…
SREF	…



Structure (logical) of a memory controller

Current clock value: N+1

DRAM cmd type	OK clock cycle
PCHG	…
ACT	N
READ	…
WRITE	…
REFRESH	…
SREF	…



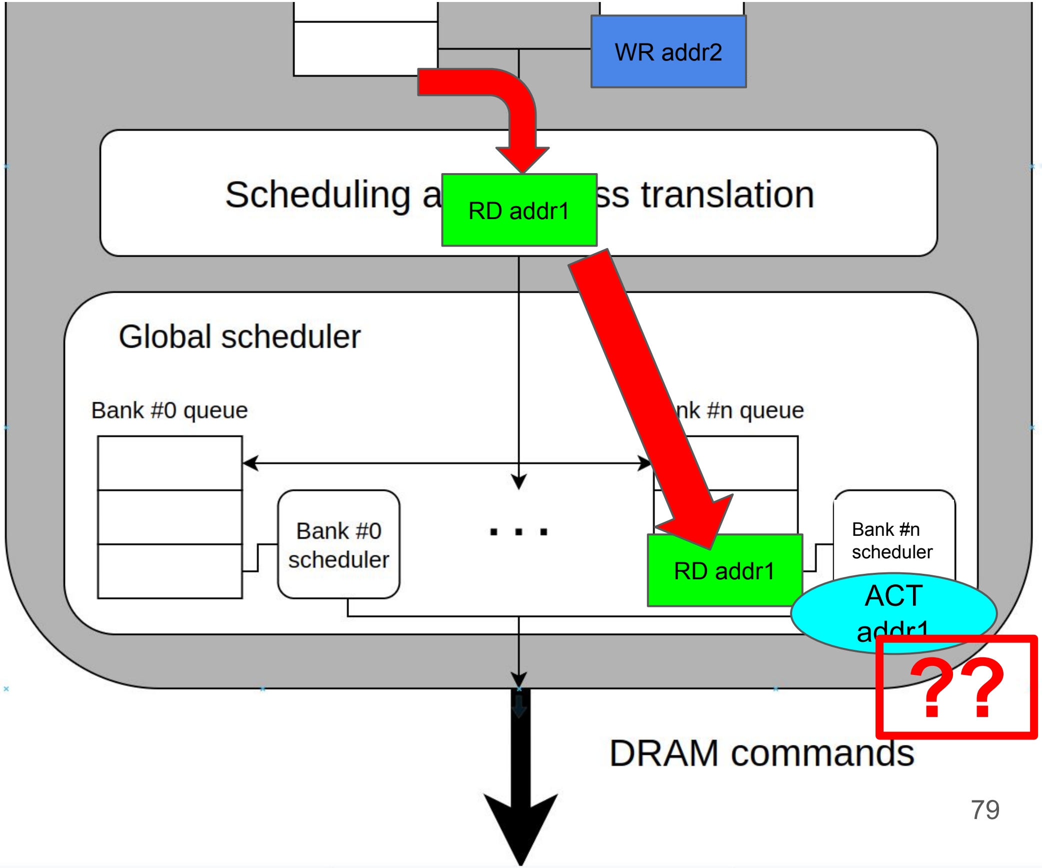
Structure (logical) of a memory controller

Current clock value:
N+1

??

DRAM cmd type	OK clock cycle
PCHG	…
ACT	N
READ	…
WRITE	…
REFRESH	…
SREF	…

??



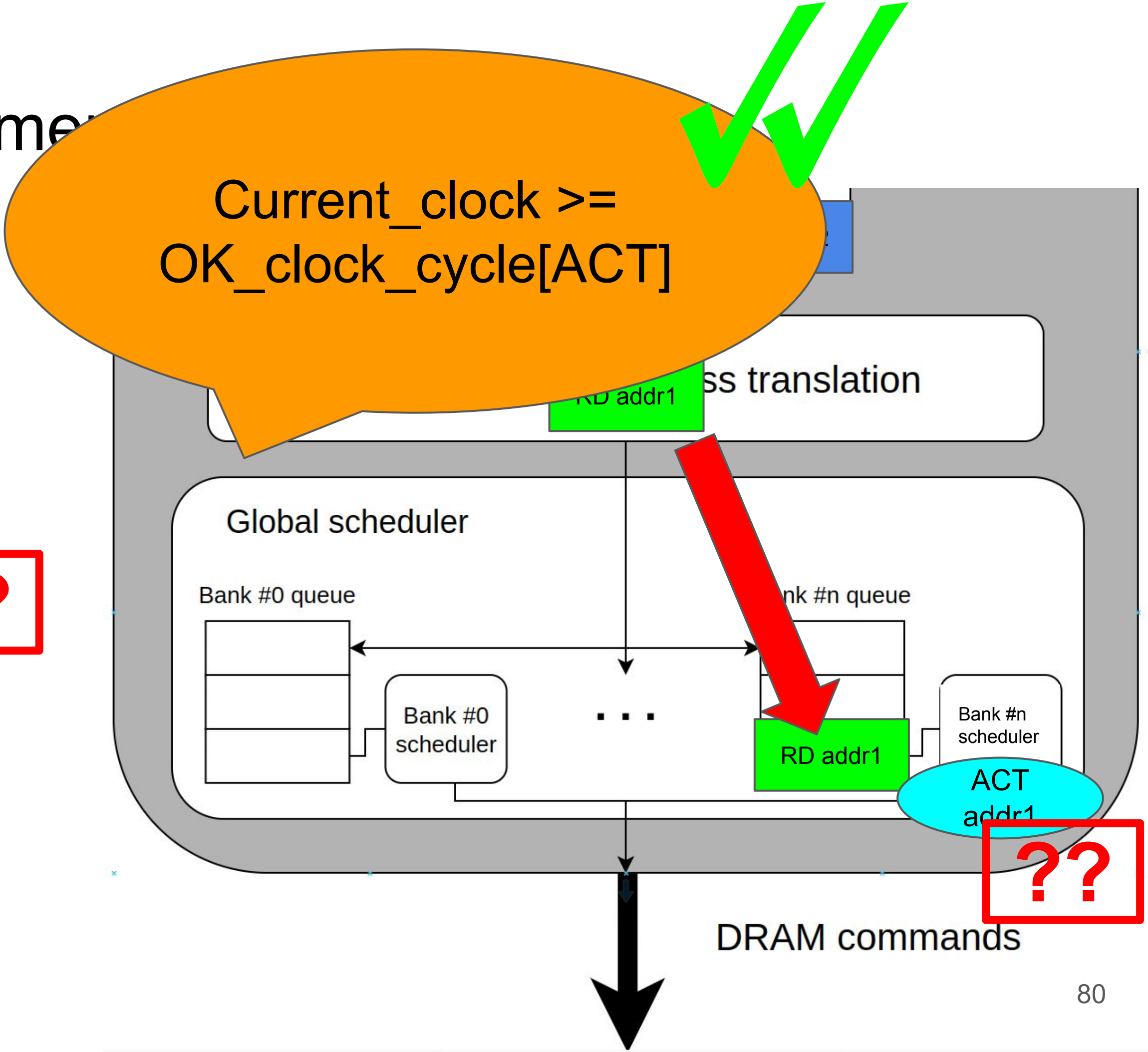
Structure (logical) of a me

Current clock value:
N+1

??

DRAM cmd type	OK clock cycle
PCHG	…
ACT	N
READ	…
WRITE	…
REFRESH	…
SREF	…

??

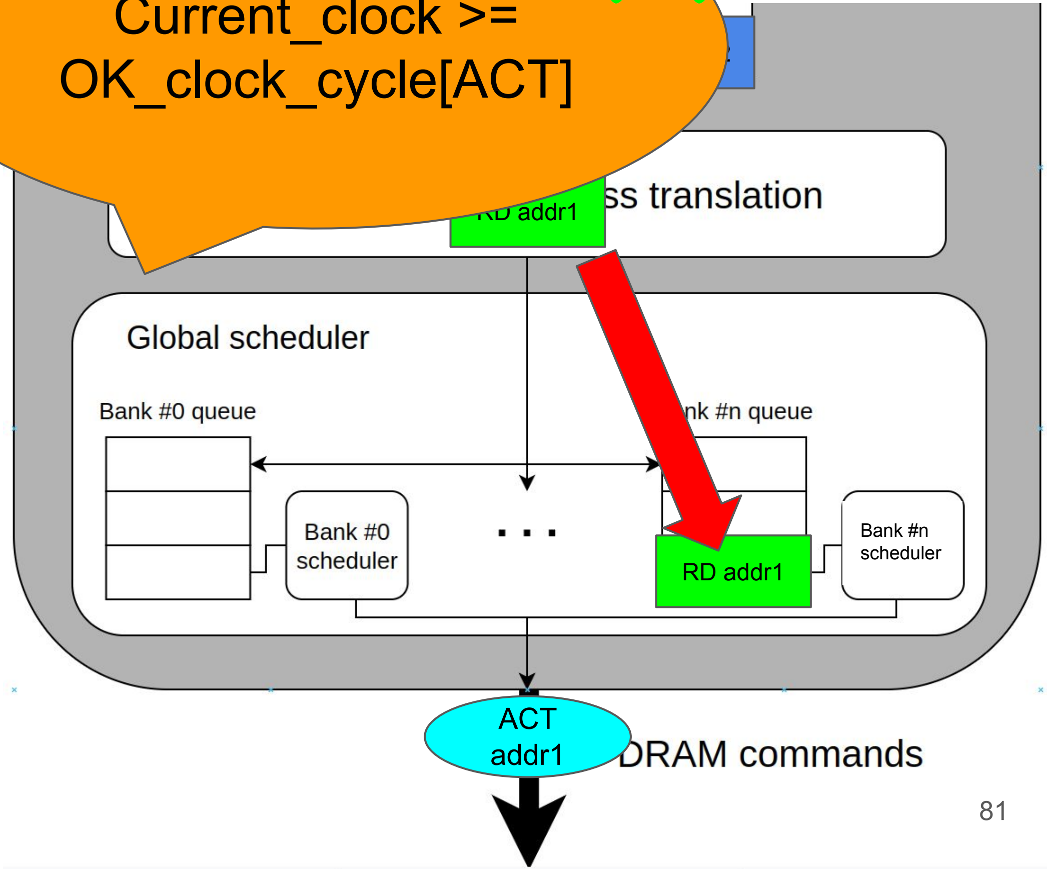


Current_clock >=
OK_clock_cycle[ACT]

Structure (logical) of a me

***Current clock value:
N+1***

DRAM cmd type	OK clock cycle
PCHG	…
ACT	N
READ	…
WRITE	…
REFRESH	…
SREF	…



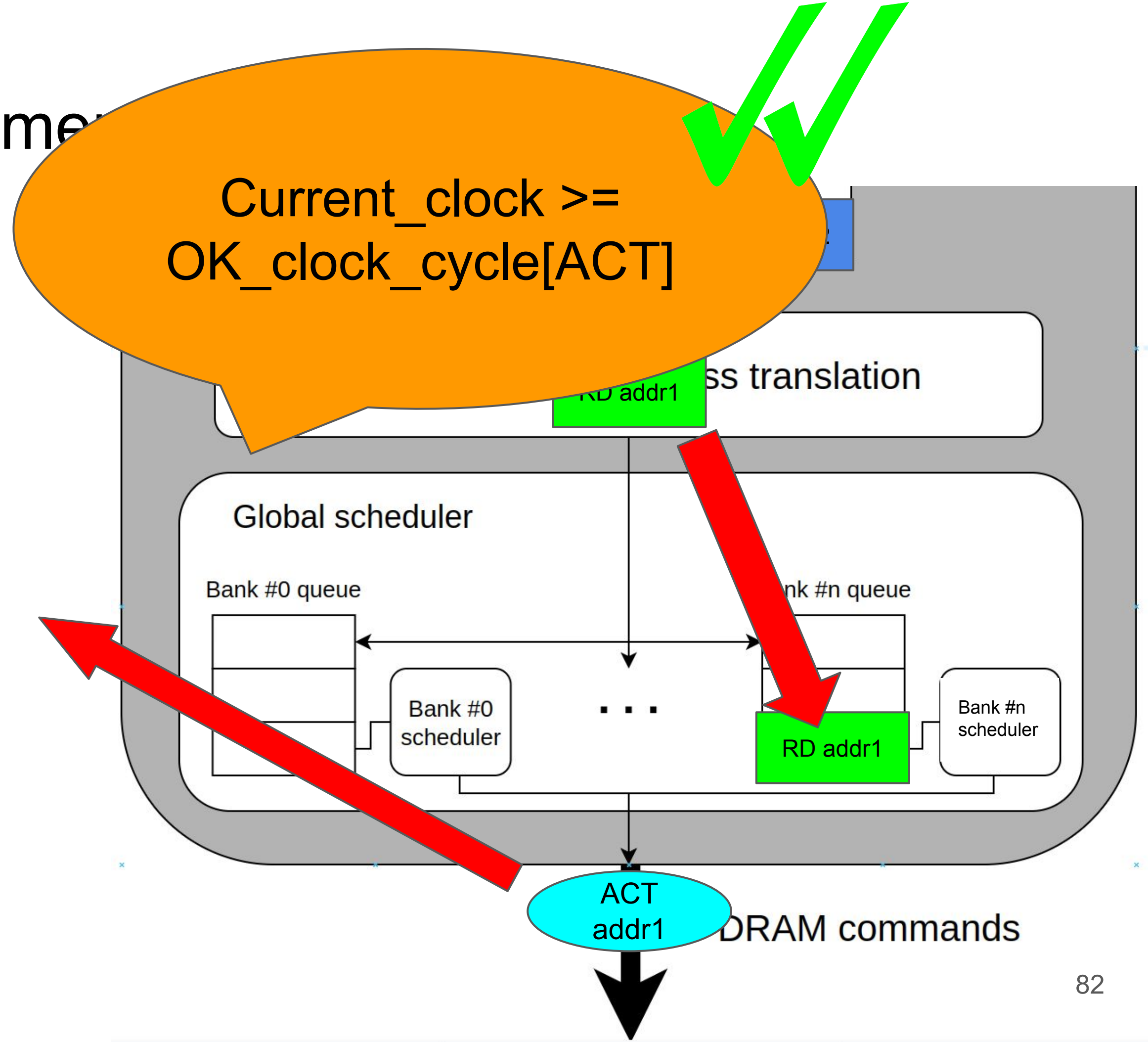
Current_clock >=
OK_clock_cycle[ACT]

✓✓

Structure (logical) of a me

***Current clock value:
N+1***

DRAM cmd type	OK clock cycle
PCHG	…
ACT	N
READ	…
WRITE	…
REFRESH	…
SREF	…

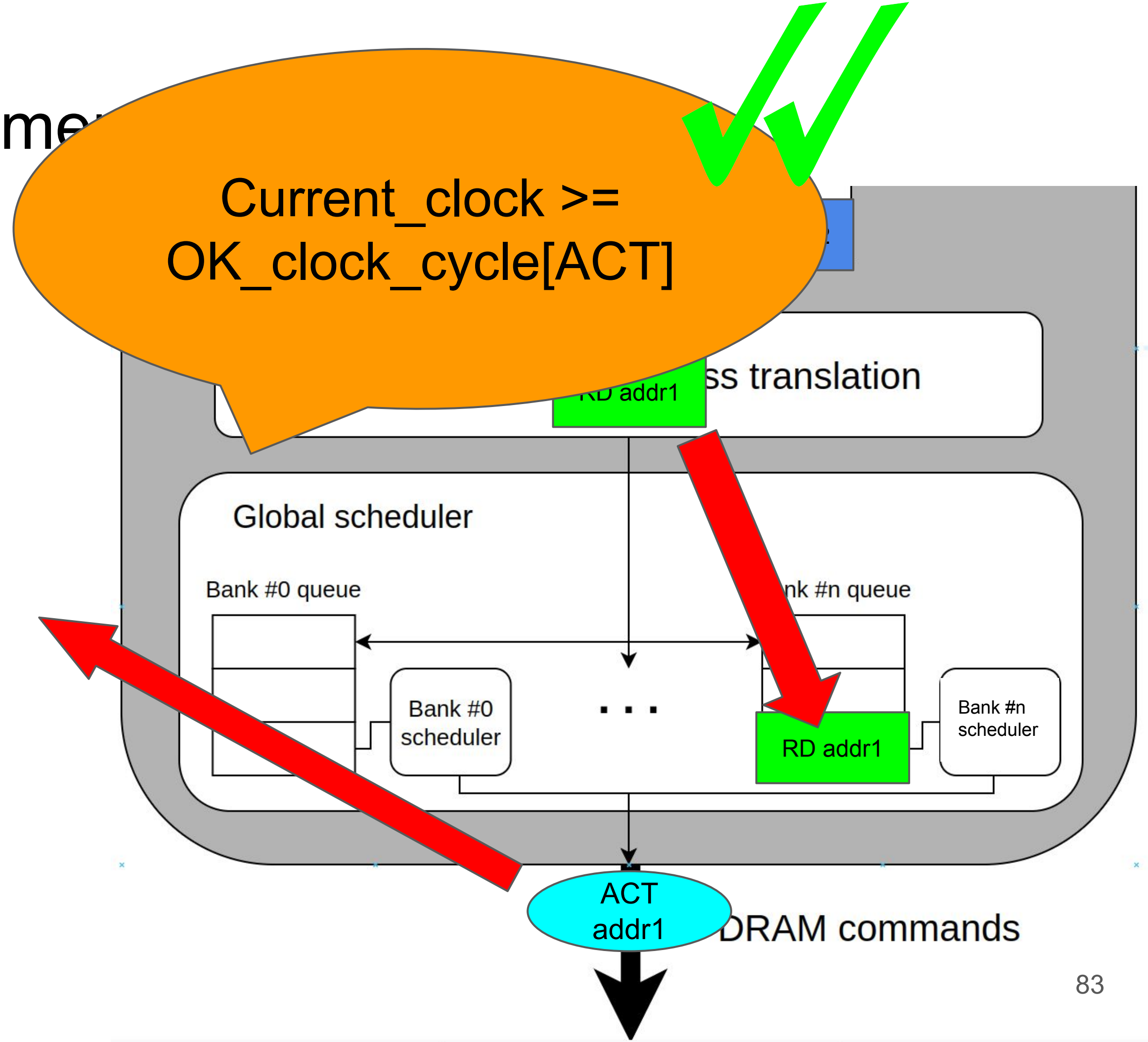


Current_clock >=
OK_clock_cycle[ACT]

Structure (logical) of a me

Current clock value:
N+1

DRAM cmd type	OK clock cycle
PCHG	… + tXXX
ACT	N + tYYY
READ	… + tZZZ
WRITE	… + tAAA
REFRESH	… + tBBB
SREF	… + tCCC

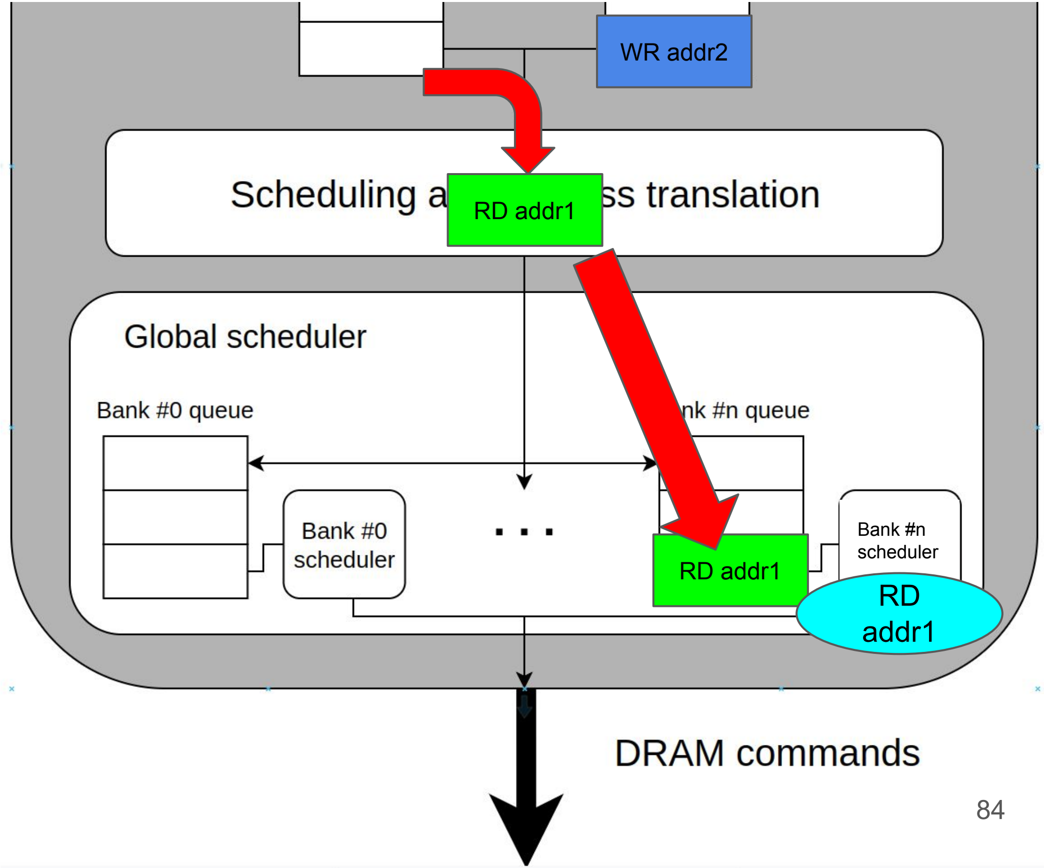


Current_clock >=
OK_clock_cycle[ACT]

Structure (logical) of a memory controller

Current clock value:
N+130

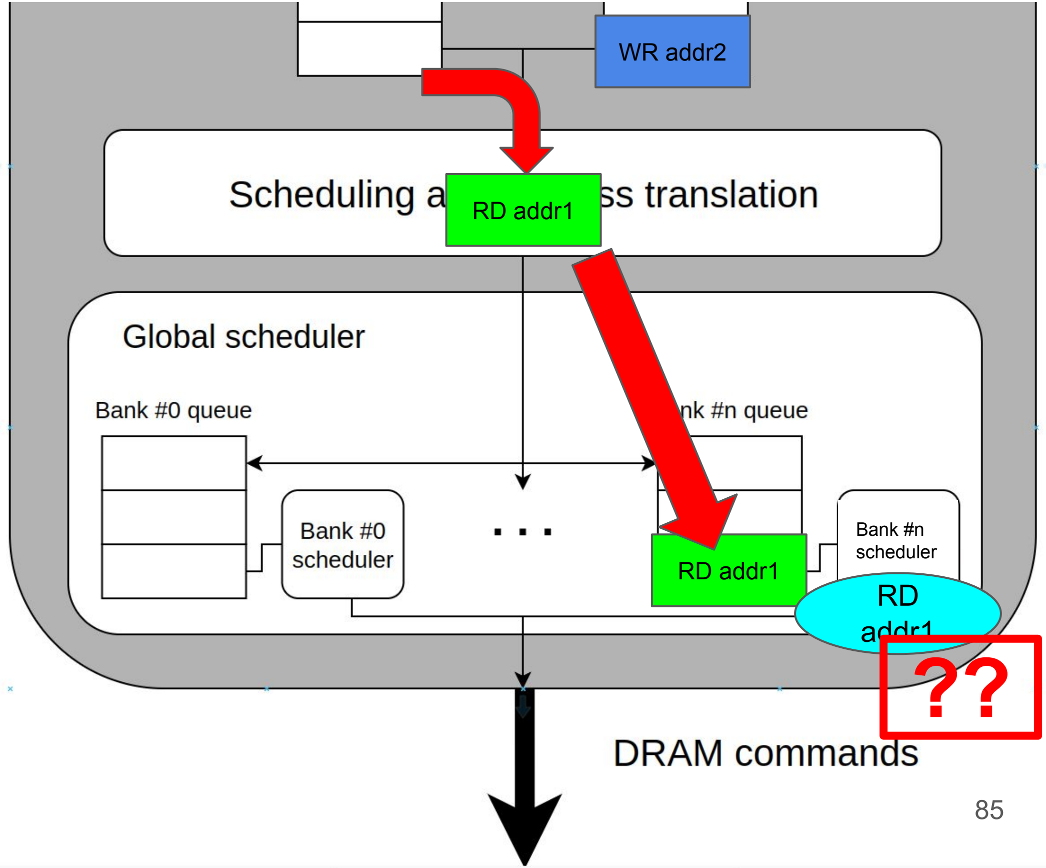
DRAM cmd type	OK clock cycle
PCHG	… + tXXX
ACT	N + tYYY
READ	N + 150
WRITE	… + tAAA
REFRESH	… + tBBB
SREF	… + tCCC



Structure (logical) of a memory controller

Current clock value: N+130

DRAM cmd type	OK clock cycle
PCHG	… + tXXX
ACT	N + tYYY
READ	N + 150
WRITE	… + tAAA
REFRESH	… + tBBB
SREF	… + tCCC



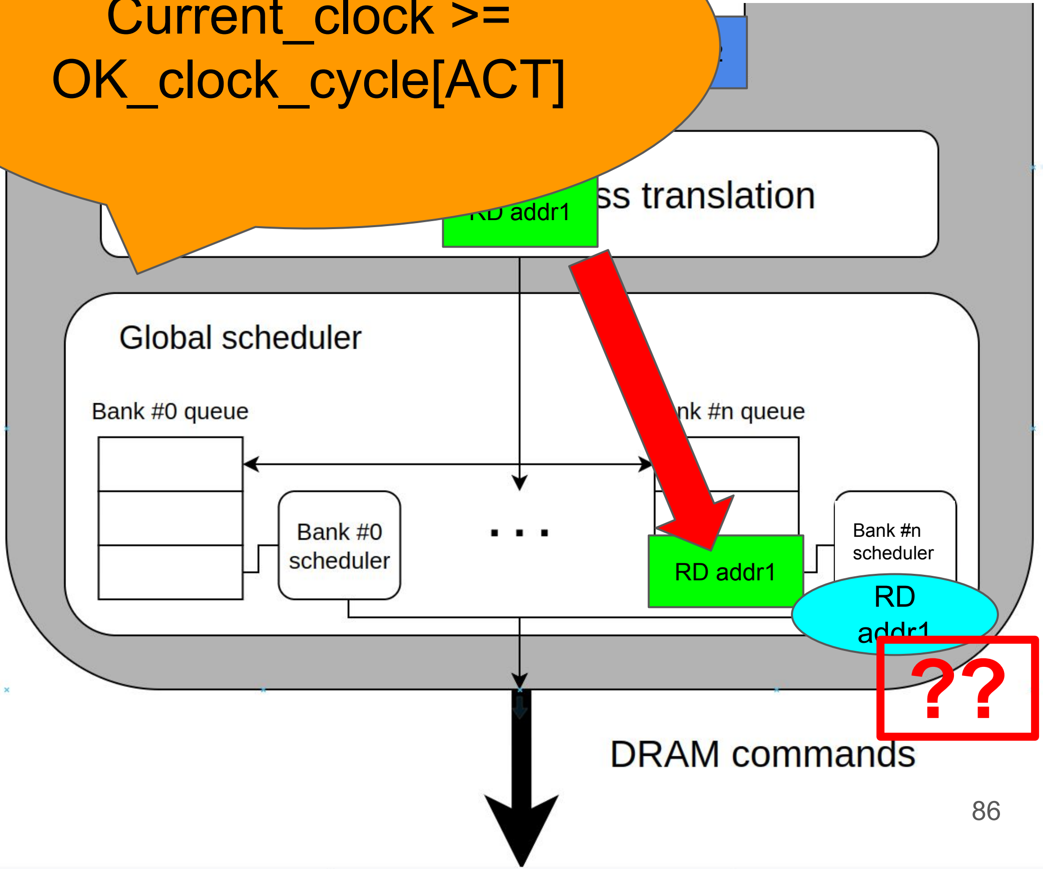
Structure (logical) of a me

Current clock value:
N+130

Current_clock >=
OK_clock_cycle[ACT]

XX

DRAM cmd type	OK clock cycle
PCHG	… + tXXX
ACT	N + tYYY
READ	N + 150
WRITE	… + tAAA
REFRESH	… + tBBB
SREF	… + tCCC



Structure (logical) of a me

Current clock value:
N+130

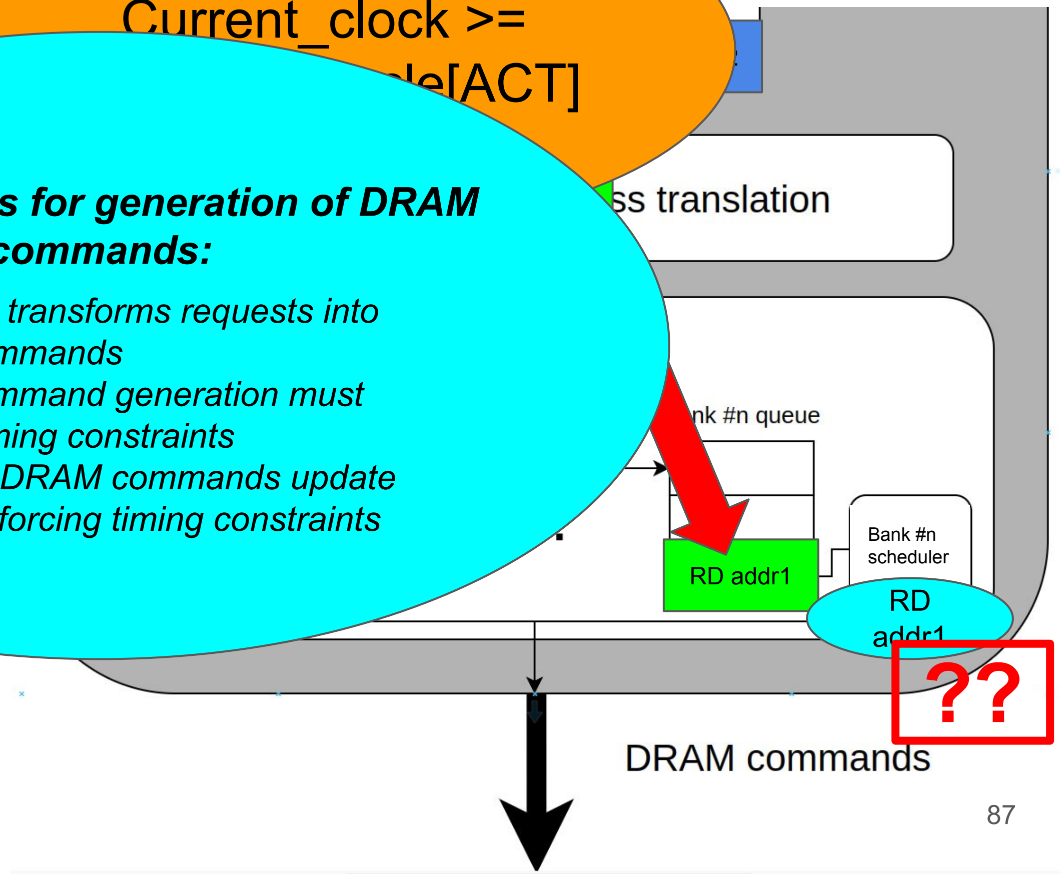
DRAM cmd type	
PCHG	
ACT	
READ	
WRITE	… + tAA
REFRESH	… + tBBB
SREF	… + tCCC

Current_clock >=
le[ACT]

XX

Relevant points for generation of DRAM commands:

1. *Scheduler transforms requests into DRAM commands*
2. *DRAM command generation must respect timing constraints*
3. *All issued DRAM commands update (some) enforcing timing constraints*



This is where our HW proposal comes into full view

Motivation for our proposal (following slides)

- Main memory is an inscrutable black box of contention-induced delays between memory transactions

Motivation for our proposal (following slides)

- Main memory is an inscrutable black box of contention-induced delays between memory transactions
- The closest available thing to contention explainability comes in the form of:

Motivation for our proposal (following slides)

- Main memory is an inscrutable black box of contention-induced delays between memory transactions
- The closest available thing to contention explainability comes in the form of:
    - Page-hit counters
        - The row that needs to be read/written has already been **ACT**ivated

Motivation for our proposal (following slides)

- Main memory is an inscrutable black box of contention-induced delays between memory transactions
- The closest available thing to contention explainability comes in the form of:
    - Page-hit counters
        - The row that needs to be read/written has already been **ACT**ivated
    - Page-miss counters
        - The row has to be **PRE**charged and **ACT**ivated; this is VERY slow…

Motivation for our proposal (following slides)

- Main memory is an inscrutable black box of contention-induced delays between memory transactions
- The closest available thing to contention explainability comes in the form of:
  - Page-hit counters
    - The row that needs to be read/written has already been **ACT**ivated
  - Page-miss counters
    - The row has to be **PRE**charged and **ACT**ivated; this is VERY slow…
  - Page-read counters

Motivation for our proposal (following slides)

- Main memory is an inscrutable black box of contention-induced delays between memory transactions
- The closest available thing to contention explainability comes in the form of:
  - Page-hit counters
    - The row that needs to be read/written has already been **ACT**ivated
  - Page-miss counters
    - The row has to be **PRE**charged and **ACT**ivated; this is VERY slow…
  - Page-read counters
  - Page-write counters

Motivation for our proposal (following slides)

- Main memory is an inscrutable black box of contention-induced delays between memory transactions
- The closest available thing to contention explainability comes in the form of:
  - Page-hit counters
    - The row that needs to be read/written has already been **ACT**ivated
  - Page-miss counters
    - The row has to be **PRE**charged and **ACT**ivated; this is VERY slow…
  - Page-read counters
  - Page-write counters
- More information on interactions and delays between memory transactions could be used for the following:

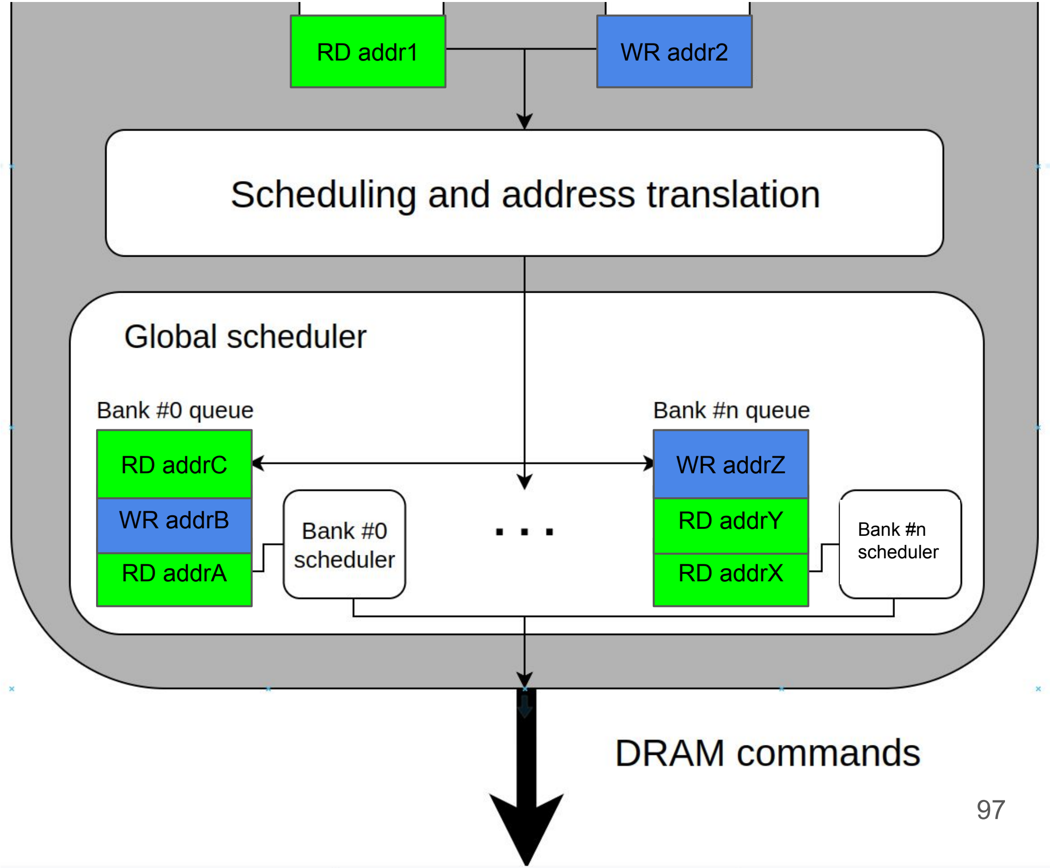
Motivation for our proposal (following slides)

- Main memory is an inscrutable black box of contention-induced delays between memory transactions
- The closest available thing to contention explainability comes in the form of:
    - Page-hit counters
        - The row that needs to be read/written has already been **ACT**ivated
    - Page-miss counters
        - The row has to be **PRE**charged and **ACT**ivated; this is VERY slow…
    - Page-read counters
    - Page-write counters
- More information on interactions and delays between memory transactions could be used for the following:
    - Optimizing core usage/pinning within main memory
    - Verification of stress tests or workloads through dedicated HW counters
    - Utilization metrics of different parts of main memory (bank, bankgroup, rank…)

Back-end view

Current clock value:
N+130

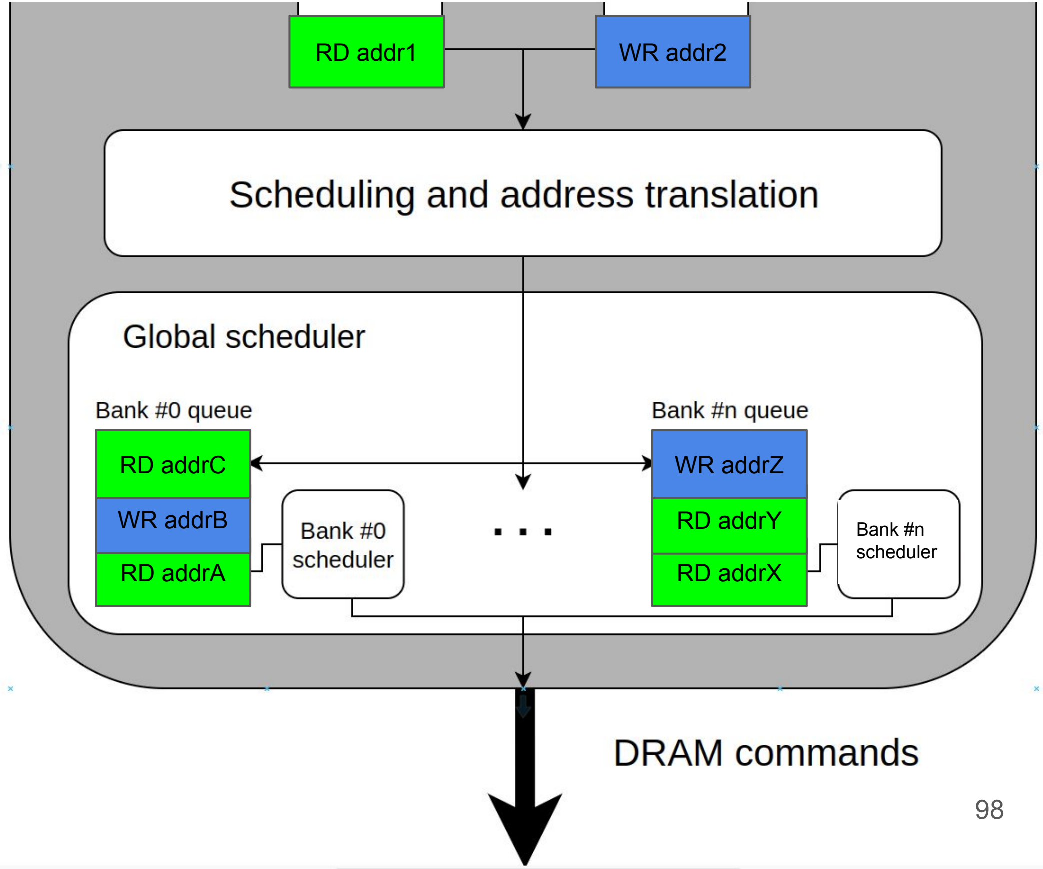
DRAM cmd type	OK clock cycle
PCHG	… + tXXX
ACT	N + tYYY
READ	N + 150
WRITE	… + tAAA
REFRESH	… + tBBB
SREF	… + tCCC



Back-end view, **NOW** with core info

Current clock value:
N+130

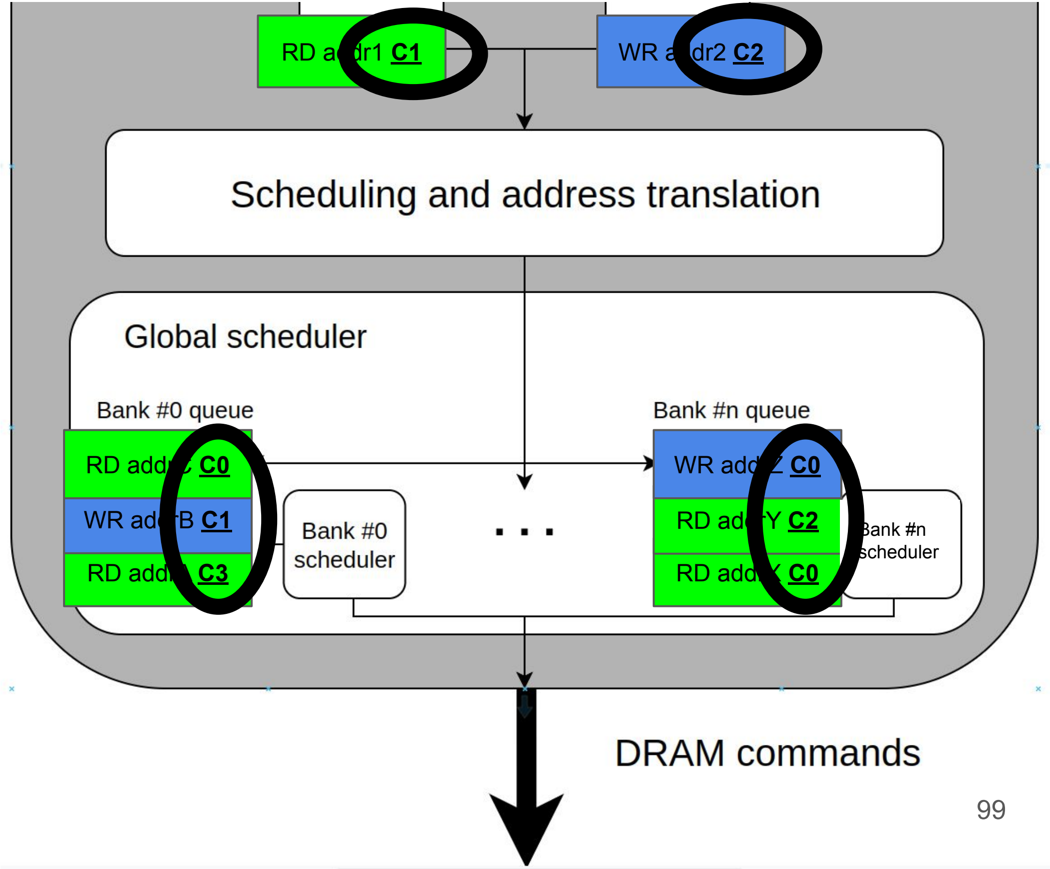
DRAM cmd type	OK clock cycle
PCHG	… + tXXX
ACT	N + tYYY
READ	N + 150
WRITE	… + tAAA
REFRESH	… + tBBB
SREF	… + tCCC



Back-end view, **NOW** with core info

Current clock value:
N+130

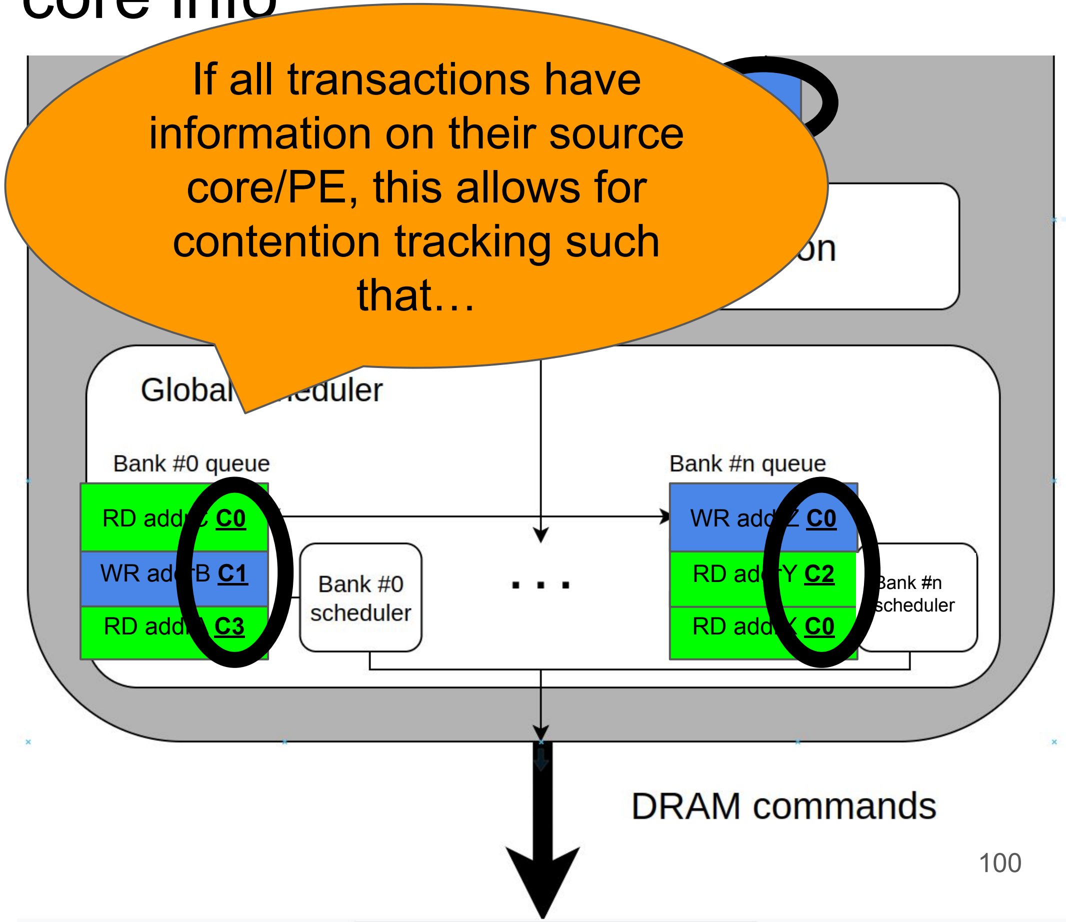
DRAM cmd type	OK clock cycle
PCHG	… + tXXX
ACT	N + tYYY
READ	N + 150
WRITE	… + tAAA
REFRESH	… + tBBB
SREF	… + tCCC



Back-end view, **NOW** with core info

Current clock value: N+130

DRAM cmd type	OK clock cycle
PCHG	… + tXXX
ACT	N + tYYY
READ	N + 150
WRITE	… + tAAA
REFRESH	… + tBBB
SREF	… + tCCC

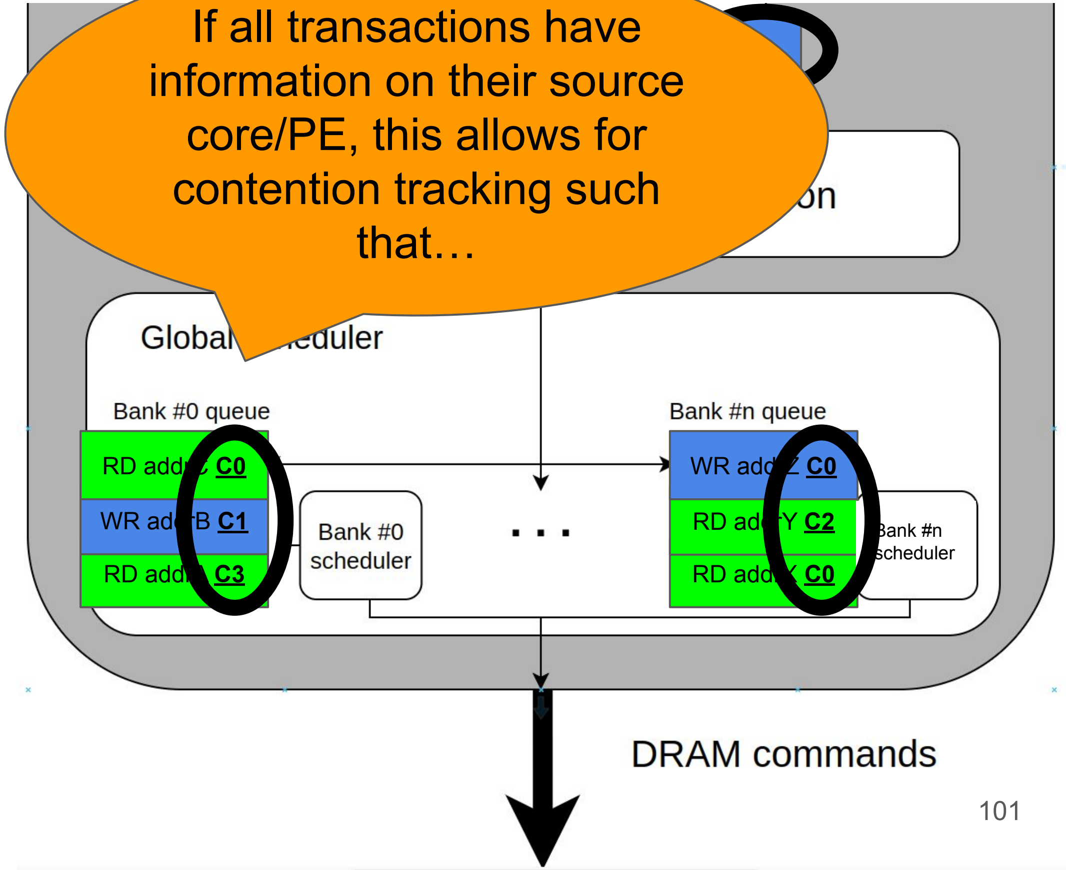


If all transactions have information on their source core/PE, this allows for contention tracking such that…

Back-end view, **NOW** with core info

Current clock value: N+130

DRAM cmd type	**OK clock cycle**	**Last core to update**
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3



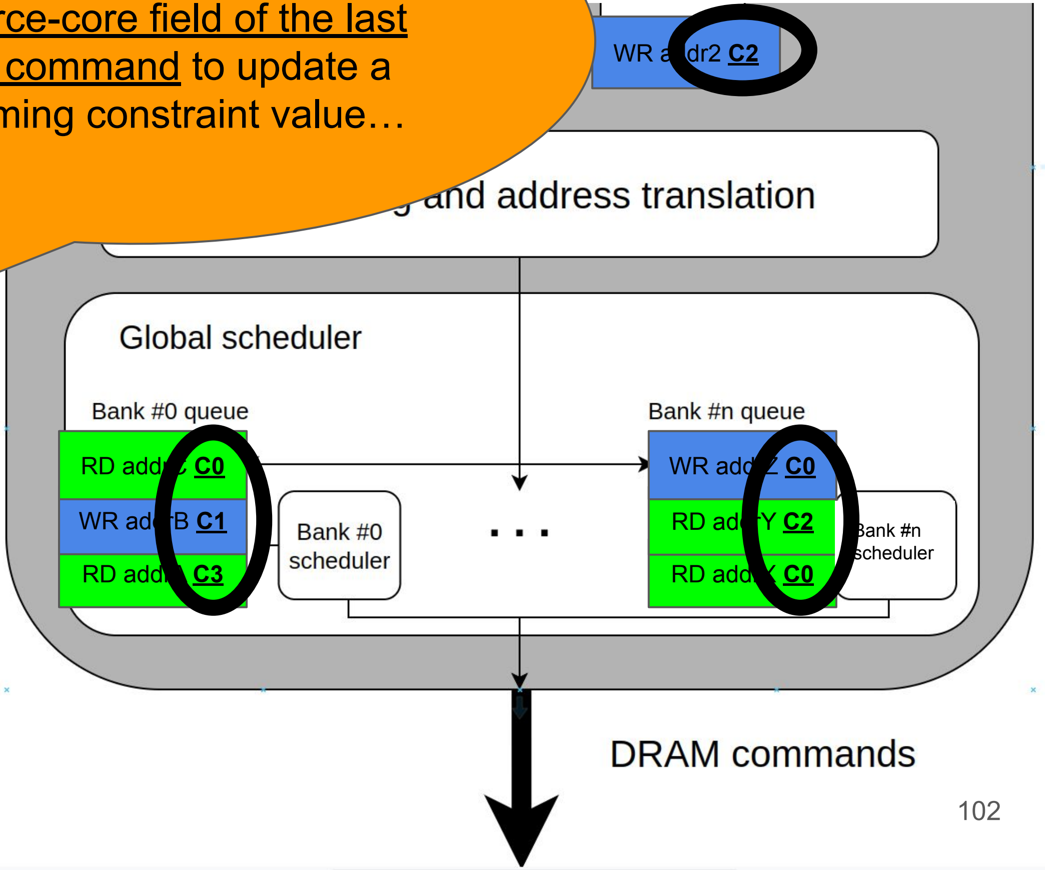
If all transactions have information on their source core/PE, this allows for contention tracking such that…

Back-end view, **N**

Current clock valu
N+130

This column holds information on the <u>source-core field of the last DRAM command</u> to update a given timing constraint value…

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

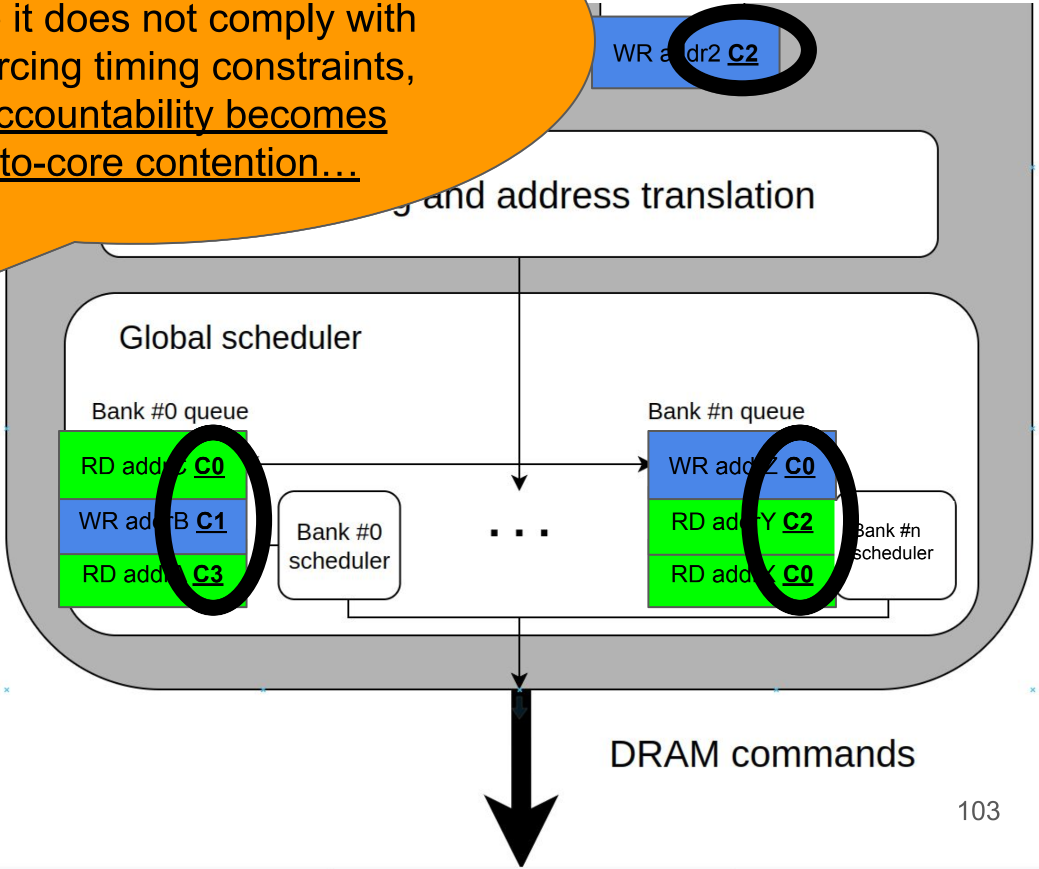


Back-end view, **N**

Current clock valu
N+130

As such, whenever a DRAM command cannot be issued because it does not comply with the enforcing timing constraints, <u>core-accountability becomes core-to-core contention…</u>

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3



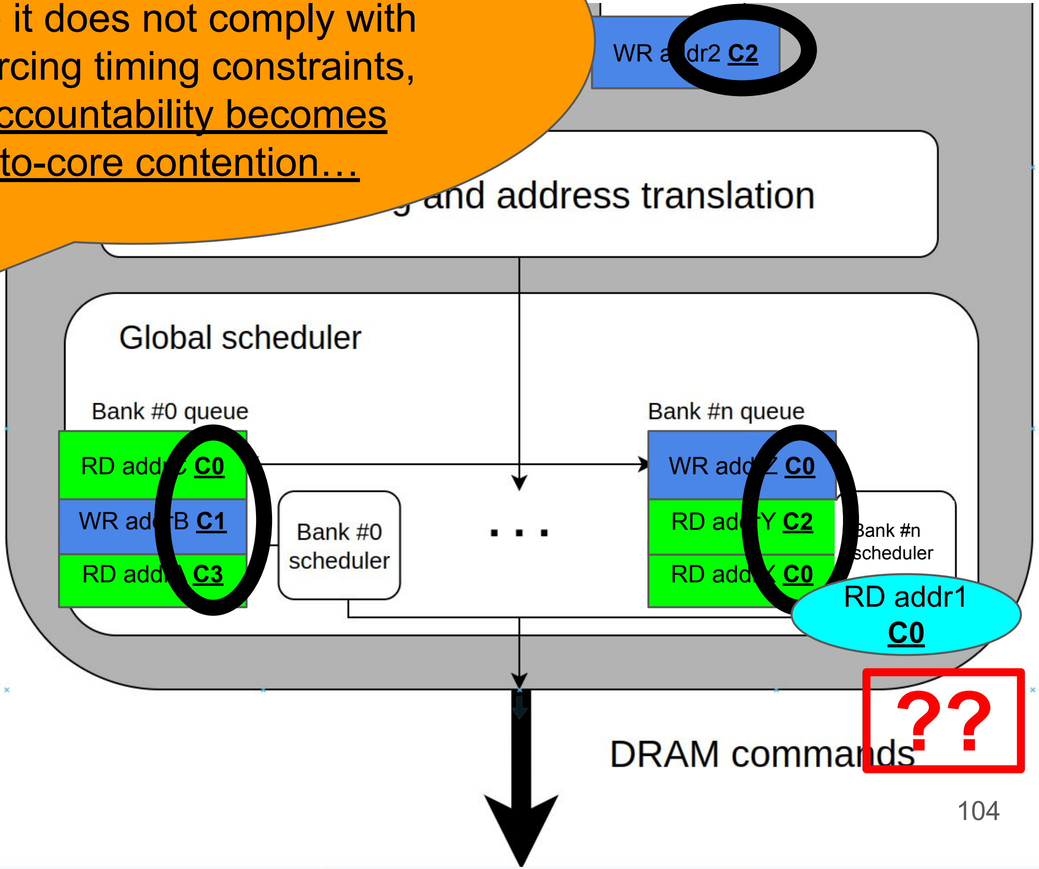
Back-end view, **N**

Current clock valu
N+130

As such, whenever a DRAM command cannot be issued because it does not comply with the enforcing timing constraints, <u>core-accountability becomes core-to-core contention…</u>

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

!!



Source-core info traceability

KEY IDEA: the extension of the timing constraint table with source-core information allows core-to-core contention tracking whenever a memory transaction tries to issue its corresponding DRAM commands

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

Source-core info traceability

KEY IDEA: the extension of the timing constraint table with source-core information allows core-to-core contention tracking whenever a memory transaction tries to issue its corresponding DRAM commands

This holds true for the presented model of the memory controller (MC)

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

Source-core info traceability

KEY IDEA: the extension of the timing constraint table with source-core information allows core-to-core contention tracking whenever a memory transaction tries to issue its corresponding DRAM commands

This holds true for the presented model of the memory controller (MC), which could be interpreted as:

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

Source-core info traceability

KEY IDEA: the extension of the timing constraint table with source-core information allows core-to-core contention tracking whenever a memory transaction tries to issue its corresponding DRAM commands

This holds true for the presented model of the memory controller (MC), which could be interpreted as:

1. MC with a static command scheduling policy, aka closed-page policy

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

Source-core info traceability

KEY IDEA: the extension of the timing constraint table with source-core information allows core-to-core contention tracking whenever a memory transaction tries to issue its corresponding DRAM commands

This holds true for the presented model of the memory controller (MC), which could be interpreted as:

1. MC with a static command scheduling policy, aka closed-page policy
   a. Typical in predictable MCs and the WCET world in general

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

Source-core info traceability

KEY IDEA: the extension of the timing constraint table with source-core information allows core-to-core contention tracking whenever a memory transaction tries to issue its corresponding DRAM commands

This holds true for the presented model of the memory controller (MC), which could be interpreted as:

1. MC with a static command scheduling policy, aka closed-page policy
   a. Typical in predictable MCs and the WCET world in general
2. MC with dynamic command scheduling policy, aka open-page policy

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

Source-core info traceability

KEY IDEA: the extension of the timing constraint table with source-core information allows core-to-core contention tracking whenever a memory transaction tries to issue its corresponding DRAM commands

This holds true for the presented model o memory controller (MC), which could be i as:

1. MC with a static command scheduling policy, aka closed-page policy
   a. Typical in predictable MCs and the WCET world in general
2. MC with dynamic command scheduling policy, aka open-page policy

More on this later…

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
[illegible]	N + 150	C2
[illegible]	… + tAAA	C2
[illegible]	… + tBBB	C3
SREF	… + tCCC	C3

Source-core info traceability

How is any of this relevant?

DRAM cmd type	**OK clock cycle**	**Last core to update**
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

Source-core info traceability

How is any of this relevant?

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

Source-core info traceability

How is any of this relevant?

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

Source-core info traceability

How is any of this relevant?

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

In fact, this “extending-of-the-timing-table” approach can be expanded upon…

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

Source-core info traceability

How is any of this relevant?

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

In fact, this “extending-of-the-timing-table” approach can be expanded upon…

DRAM cmd type	OK clock cycle	Last core to update
PCHG	… + tXXX	C0
ACT	N + tYYY	C1
READ	N + 150	C2
WRITE	… + tAAA	C2
REFRESH	… + tBBB	C3
SREF	… + tCCC	C3

Core-to-core

✓✓

Source-core info traceability

How is any of this relevant?

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

In fact, this “extending-of-the-timing-table” approach can be expanded upon…

DRAM cmd type	OK clock cycle	Last core to update	Address of request
PCHG	… + tXXX	C0	addr1
ACT	N + tYYY	C1	addr2
READ	N + 150	C2	addr3
WRITE	… + tAAA	C2	addr4
REFRESH	… + tBBB	C3	addr5
SREF	… + tCCC	C3	addr6

Core-to-core ✓✓

Type of contention ✓✓

Source-core info traceability

How is any of this relevant?

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

In fact, this “extending-of-the-timing-table” approach can be expanded upon…

DRAM cmd type	OK clock cycle	Last core to update	Address of request	Enforced time constr.
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC	C3	addr6	tCCC

Core-to-core	Type of contention	Time constraint analysis
✓✓	✓✓	✓✓

Source-core info traceability

How is any of this relevant?

Level 0

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

In fact, this “extending-of-the-timing-table” approach can be expanded upon…

DRAM cmd type	OK clock cycle	Last core to update	Address of request	Enforced time constr.
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC	C3	addr6	tCCC

Core-to-core ✓✓

Type of contention ✓✓

Time constraint analysis ✓✓

Source-core info traceability

Level 0 performance counters:

- Page hits
- Page misses
- Page reads
- Page writes

How is any of this rel

The only obtainab
contention is lim
at all, shout ou
into finer-level
requests being

Main memory, in
inscrutable black b
contention we cannot g
right now.

In fact, this “extending-of-the-timing-tab
can be expanded upon…

Address of request	Enforced time constr.
ddr1	tXXX
dr2	tYYY
r3	tZZZ
dr4	tAAA
addr5	tBBB
addr6	tCCC

✓✓

ype of contention

✓✓

Time constraint analysis

✓✓

Source-core info traceability

How is any of this relevant?

Level 0

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

In fact, this “extending-of-the-timing-table” approach can be expanded upon…

DRAM cmd type	OK clock cycle	Last core to update	Address of request	Enforced time constr.
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC	C3	addr6	tCCC

Core-to-core	Type of contention	Time constraint analysis
✓✓	✓✓	✓✓

Source-core info traceability

How is any of this relevant?

Level 0

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

In fact, this "extending-of-the-timing-table" approach can be expanded upon…

Level 1

DRAM cmd type	OK clock cycle	Last core to update	Address of request	Enforced time constr.
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC	C3	addr6	tCCC

Core-to-core ✓✓

Type of contention ✓✓

Time constraint analysis ✓✓

Level 0 performance counters:

Page hits
Page misses
Page reads
Page writes

ceability

Ho

The only obtaina contention is li at all, shout ou into finer-level requests being

Main memory, in inscrutable black b contention we cannot right now.

In fact, this “extending-of-the-timing-tab can be expanded upon…

Level 1 performance counters:

- Level 0’s perf. counters
- Core-to-core delays at MC’s back-end
- Core-to-core delays also at MC’s front-end

Address of request	Enforced time constr.
ddr1	tXXX
dr2	tYYY
r3	tZZZ
dr4	tAAA
addr5	tBBB
addr6	tCCC

ype of contention ✓✓

Time constraint analysis ✓✓

✓✓

Source-core info traceability

How is any of this relevant?

Level 0

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

In fact, this “extending-of-the-timing-table” approach can be expanded upon…

Level 1

DRAM cmd type	OK clock cycle	Last core to update	Address of request	Enforced time constr.
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC	C3	addr6	tCCC

Core-to-core	Type of contention	Time constraint analysis
✓✓	✓✓	✓✓

Source-core info traceability

How is any of this relevant?

Level 0

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

In fact, this “extending-of-the-timing-table” approach can be expanded upon…

Level 2

Level 1

DRAM cmd type	OK clock cycle	Last core to update	Address of request	Enforced time constr.
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC	C3	addr6	tCCC

Core-to-core	Type of contention	Time constraint analysis
✓✓	✓✓	✓✓

ceability

<u>Level 0 performance counters:</u>

Page hits
Page misses
Page reads
Page writes

<u>Level 1 performance counters:</u>

Level 0
CX-CY back-end
CX-CY front-end

<u>Level 2 performance counters:</u>

- Level 1’s perf. counters
- Core-to-core delays with information on their placement in DRAM memory:
  - Within-bank dependencies
  - Within-bankgroup dependencies
  - Within-rank dependencies
  - …

Ho

The only obtainab
contention is lim
at all, shout ou
into finer-level
requests being

Main memory, in
inscrutable black b
contention we cannot g
right now.

In fact, this “extending-of-the-timing-tabl
can be expanded upon…

request	Enforced time constr.
ddr1	tXXX
dr2	tYYY
r3	tZZZ
dr4	tAAA
addr5	tBBB
addr6	tCCC

✓✓

ype of contention

✓✓

Time constraint analysis

✓✓

Source-core info traceability

How is any of this relevant?

Level 0

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

In fact, this “extending-of-the-timing-table” approach can be expanded upon…

Level 2

Level 1

DRAM cmd type	OK clock cycle	Last core to update	Address of request	Enforced time constr.
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC	C3	addr6	tCCC

Core-to-core ✓✓

Type of contention ✓✓

Time constraint analysis ✓✓

Source-core info traceability

How is any of this relevant?

Level 0

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

In fact, this “extending-of-the-timing-table” approach can be expanded upon…

Level 3

Level 2

Level 1

DRAM cmd type	OK clock cycle	Last core to update	Address of request	Enforced time constr.
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC	C3	addr6	tCCC

Core-to-core ✓✓

Type of contention ✓✓

Time constraint analysis ✓✓

Level 0 performance counters:

Page hits
Page misses
Page reads
Page writes

Level 1 performance counters:

Level 0
CX-CY back-end
CX-CY front-end

Level 3 performance counters:

- Level 2’s performance counters
- Core-to-core delays at time-constraint level (finest level of granularity)

Level 2 performance counters:

Level 1
Contention w.r.t. placement

Ho

The only obtai
contention is
at all, shout
into finer-lev
requests bei

Main memory, i
inscrutable black
co

request	Enforced time constr.
addr1	tXXX
ddr2	tYYY
ddr3	tZZZ
addr4	tAAA
addr5	tBBB
addr6	tCCC

ore ✓✓

Type of contention ✓✓

Time constraint analysis ✓✓

ceability

Level 0 performance counters:

Page hits
Page misses
Page reads
Page writes

Level 1 performance counters:

Level 0
CX-CY back-end
CX-CY front-end

Lev

Level 0

Ho

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty con er understanding of

ble” approach

DRAM cmd type	OK clock cycle	Last update	request	Enforced time constr.
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC			tCCC

Level 2 performance counters:

Level 1
Contention w.r.t. placement

Core-to-core

✓✓ ✓✓

Level 3 performance counters:

Level 2
Timing constraint analysis

Source-core info traceability

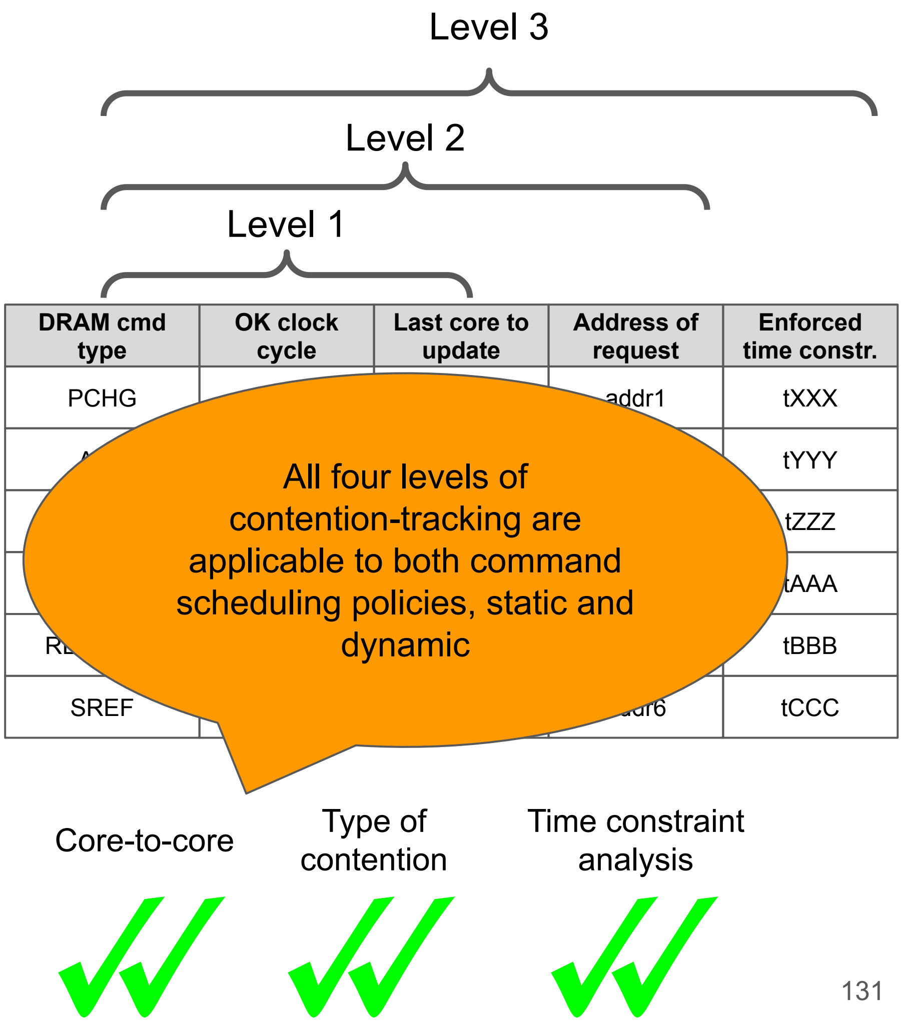
How is any of this relevant?

Level 0

The only obtainable data related to main-memory contention is limited to page-hits and page-misses (if at all, shout out to the T2080), with no sort of insight into finer-level details as to who/why are memory requests being delayed.

Main memory, in terms of contention, is an inscrutable black box of black magic of nasty contention we cannot get a proper understanding of right now.

In fact, this "extending-of-the-timing-table" approach can be expanded upon…

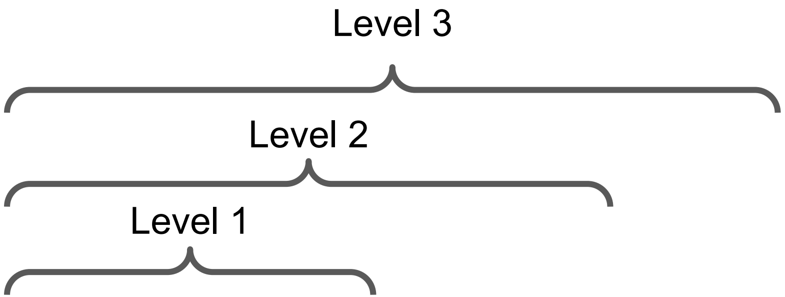


DRAM cmd type	OK clock cycle	Last core to update	Address of request	Enforced time constr.
PCHG			addr1	tXXX
A				tYYY
				tZZZ
				tAAA
RE				tBBB
SREF			ddr6	tCCC

All four levels of contention-tracking are applicable to both command scheduling policies, static and dynamic

Contention proposals

Level 0: page-hits and page-misses
Level 1: core-to-core contention (including inter-queue contention)
Level 2: memory-location-aware contention
Level 3: time constraint analysis



DRAM cmd type	**OK clock cycle**	**Last core to update**	**Address of request**	**Enforced time constr.**
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC	C3	addr6	tCCC

Core-to-core	Type of contention	Time constraint analysis
✓✓	✓✓	✓✓

Contention proposals

Level 0: page-hits and page-misses
Level 1: core-to-core contention (including inter-queue contention)
Level 2: memory-location-aware contention
Level 3: time constraint analysis

The higher the level, the higher the accuracy and specificity of contention information, but also the higher the “hardware”/complexity cost…

Level 3

Level 2

Level 1

DRAM cmd type	OK clock cycle	Last core to update	Address of request	Enforced time constr.
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC	C3	addr6	tCCC

Core-to-core ✓✓

Type of contention ✓✓

Time constraint analysis ✓✓

Contention proposals

Level 0: page-hits and page-misses
Level 1: core-to-core contention (including inter-queue contention)
Level 2: memory-location-aware contention
Level 3: time constraint analysis

The higher the level, the higher the accuracy and specificity of contention information, but also the higher the "hardware"/complexity cost…

This is also true of both command scheduling policies: static command scheduling **almost** behaves like a set of FIFO queues trying to issue DRAM commands

Level 3

Level 2

Level 1

DRAM cmd type	OK clock cycle	Last core to update	Address of request	Enforced time constr.
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC	C3	addr6	tCCC

Core-to-core ✓✓

Type of contention ✓✓

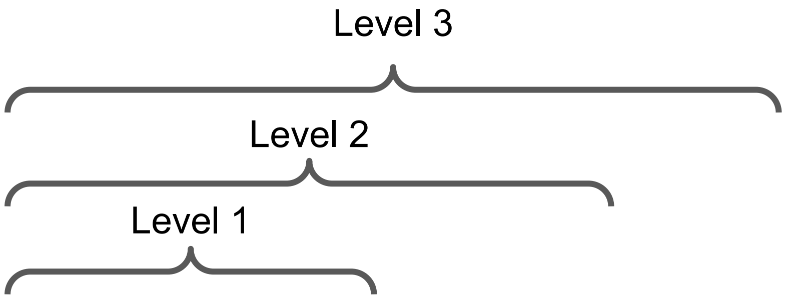
Time constraint analysis ✓✓

Contention proposals

Level 0: page-hits and page-misses
Level 1: core-to-core contention (including inter-queue contention)
Level 2: memory-location-aware contention
Level 3: time constraint analysis

The higher the level, the higher the accuracy and specificity of contention information, but also the higher the “hardware”/complexity cost…

This is also true of both command scheduling policies: static command scheduling **almost** behaves like a set of FIFO queues trying to issue DRAM commands, while dynamic command scheduling is more cumbersome…



DRAM cmd type	OK clock cycle	Last core to update	Address of request	Enforced time constr.
PCHG	… + tXXX	C0	addr1	tXXX
ACT	N + tYYY	C1	addr2	tYYY
READ	N + 150	C2	addr3	tZZZ
WRITE	… + tAAA	C2	addr4	tAAA
REFRESH	… + tBBB	C3	addr5	tBBB
SREF	… + tCCC	C3	addr6	tCCC

Core-to-core	Type of contention	Time constraint analysis
✓✓	✓✓	✓✓

Status of the work

- **The main objective is to present a hardware proposal considering both statically- and dynamically-scheduled MCs, progressing from Level 0 to 3 of complexity**

Status of the work

- **The main objective is to present a hardware proposal considering both statically- and dynamically-scheduled MCs, progressing from Level 0 to 3 of complexity**
    - The value of our proposal also lies in the managing of complex DRAM/MC concepts such as precharge arbitration, four-activation-window enforcements, accounting for the various queue traversal policies within the MC…

Status of the work

- **The main objective is to present a hardware proposal considering both statically- and dynamically-scheduled MCs, progressing from Level 0 to 3 of complexity**
    - The value of our proposal also lies in the managing of complex DRAM/MC concepts such as precharge arbitration, four-activation-window enforcements, accounting for the various queue traversal policies within the MC…

- **DRAMsim3 simulator has been modified to record up to Level-3 contention**

Status of the work

- **The main objective is to present a hardware proposal considering both statically- and dynamically-scheduled MCs, progressing from Level 0 to 3 of complexity**
  - The value of our proposal also lies in the managing of complex DRAM/MC concepts such as precharge arbitration, four-activation-window enforcements, accounting for the various queue traversal policies within the MC…

- **DRAMsim3 simulator has been modified to record up to Level-3 contention**
  - The work performed in the simulator has served to establish the theoretical bases of MCs and how to modify/retouch them for contention transparency

Status of the work

- **The main objective is to present a hardware proposal considering both statically- and dynamically-scheduled MCs, progressing from Level 0 to 3 of complexity**
  - The value of our proposal also lies in the managing of complex DRAM/MC concepts such as precharge arbitration, four-activation-window enforcements, accounting for the various queue traversal policies within the MC…

- **DRAMsim3 simulator has been modified to record up to Level-3 contention**
  - The work performed in the simulator has served to establish the theoretical bases of MCs and how to modify/retouch them for contention transparency
  - DRAMsim3 only supports one command scheduling policy (dynamic)

Status of the work

- **The main objective is to present a hardware proposal considering both statically- and dynamically-scheduled MCs, progressing from Level 0 to 3 of complexity**
  - The value of our proposal also lies in the managing of complex DRAM/MC concepts such as precharge arbitration, four-activation-window enforcements, accounting for the various queue traversal policies within the MC…

- **DRAMsim3 simulator has been modified to record up to Level-3 contention**
  - The work performed in the simulator has served to establish the theoretical bases of MCs and how to modify/retouch them for contention transparency
  - DRAMsim3 only supports one command scheduling policy (dynamic)
  - Results can be obtained from the modified version of the simulator, maybe even presenting the modified simulator itself as a publication

Status of the work

- **The main objective is to present a hardware proposal considering both statically- and dynamically-scheduled MCs, progressing from Level 0 to 3 of complexity**
  - The value of our proposal also lies in the managing of complex DRAM/MC concepts such as precharge arbitration, four-activation-window enforcements, accounting for the various queue traversal policies within the MC…
- **DRAMsim3 simulator has been modified to record up to Level-3 contention**
  - The work performed in the simulator has served to establish the theoretical bases of MCs and how to modify/retouch them for contention transparency
  - DRAMsim3 only supports one command scheduling policy (dynamic)
  - Results can be obtained from the modified version of the simulator, maybe even presenting the modified simulator itself as a publication
- **Writing of paper-worthy stuff**

Status of the work

- **The main objective is to present a hardware proposal considering both statically- and dynamically-scheduled MCs, progressing from Level 0 to 3 of complexity**
  - The value of our proposal also lies in the managing of complex DRAM/MC concepts such as precharge arbitration, four-activation-window enforcements, accounting for the various queue traversal policies within the MC…

- **DRAMsim3 simulator has been modified to record up to Level-3 contention**
  - The work performed in the simulator has served to establish the theoretical bases of MCs and how to modify/retouch them for contention transparency
  - DRAMsim3 only supports one command scheduling policy (dynamic)
  - Results can be obtained from the modified version of the simulator, maybe even presenting the modified simulator itself as a publication

- **Writing of paper-worthy stuff**
  - This presentation and its figures (along with another, more-detailed presentation given for various master's course subjects) are essentially the background and introduction to MC-related concepts to build upon and explain in the intended future publication

Status of the work

- **The main objective is to present a hardware proposal considering both statically- and dynamically-scheduled MCs, progressing from Level 0 to 3 of complexity**
    - The value of our proposal also lies in the managing of complex DRAM/MC concepts such as precharge arbitration, four-activation-window enforcements, accounting for the various queue traversal policies within the MC…

- **DRAMsim3 simulator has been modified to record up to Level-3 contention**
    - The work performed in the simulator has served to establish the theoretical bases of MCs and how to modify/retouch them for contention transparency
    - DRAMsim3 only supports one command scheduling policy (dynamic)
    - Results can be obtained from the modified version of the simulator, maybe even presenting the modified simulator itself as a publication

- **Writing of paper-worthy stuff**
    - This presentation and its figures (along with another, more-detailed presentation given for various master’s course subjects) are essentially the background and introduction to MC-related concepts to build upon and explain in the intended future publication
    - Explanation of contention tracking on a static-command-scheduling MC is almost finished, along with the hardware proposal itself in the form of logical blocks and such