Memory controllers

A short, conceptual introduction

Disclaimer!!

- This presentation will be laying out a logical overview of how memory controllers (MCs) work based on previous analysis of the DRAMsim3 simulator (https://github.com/umd-memsys/DRAMsim3)

Disclaimer!!

- This presentation will be laying out a logical overview of how memory controllers (MCs) work based on previous analysis of the DRAMsim3 simulator (https://github.com/umd-memsys/DRAMsim3)
- As such, take this presentation as what it is: an abstracted introduction to a software-defined memory controller

Disclaimer!!

- This presentation will be laying out a logical overview of how memory controllers (MCs) work based on previous analysis of the DRAMsim3 simulator (https://github.com/umd-memsys/DRAMsim3)
- As such, take this presentation as what it is: an abstracted introduction to a software-defined memory controller
- If interested, the closest I have managed to find on real-world MC implementation is the following paper: https://www.cs.utah.edu/~bojnordi/data/tocs13.pdf

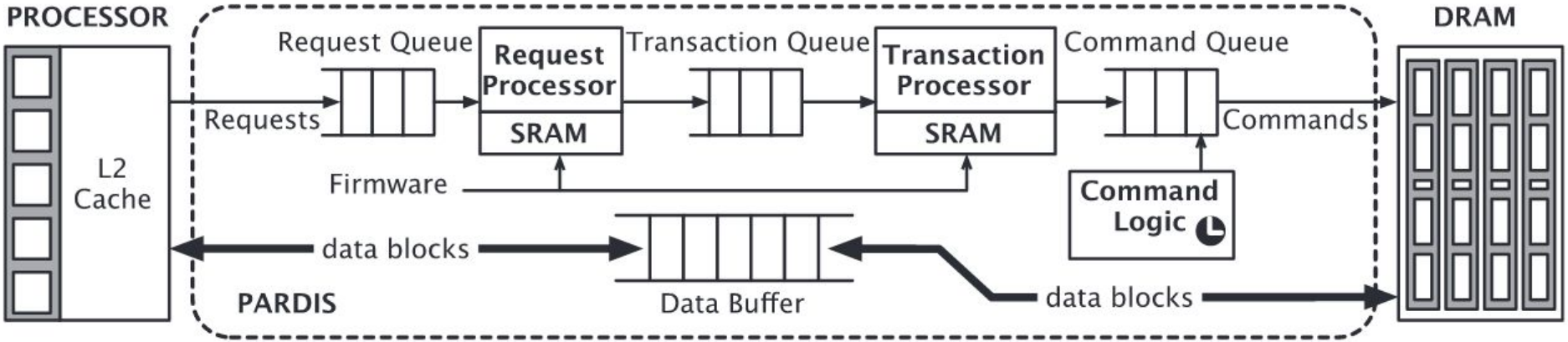


Fig. 1. Illustrative example of PARDIS in a computer system.

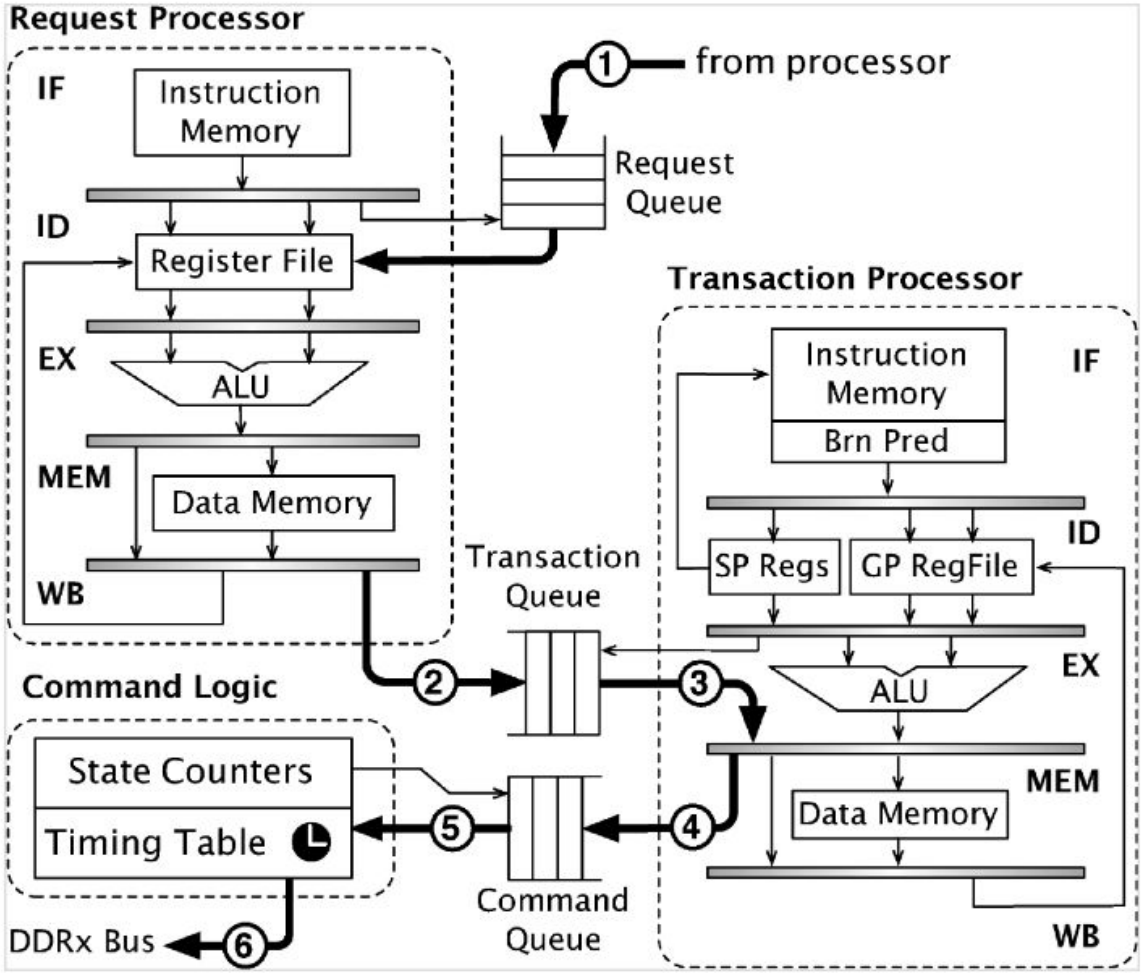


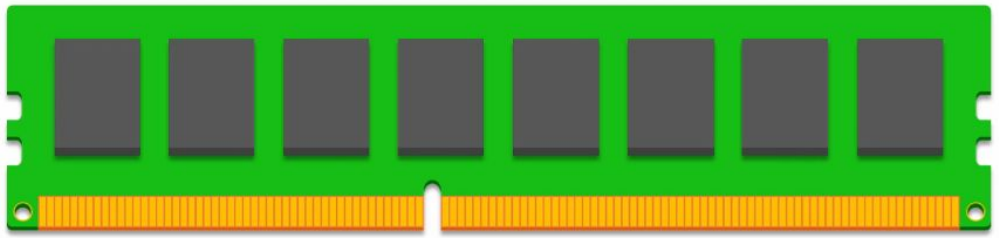
Fig. 15. Illustrative example of the proposed PARDIS implementation.

DRAM basics: a speedrun

DRAM memory is organized as follows:

- Each “stick” of RAM corresponds (typically) to one **channel** (the “stick” is what it’s called the Dual Inline Memory Module, the **DIMM**)

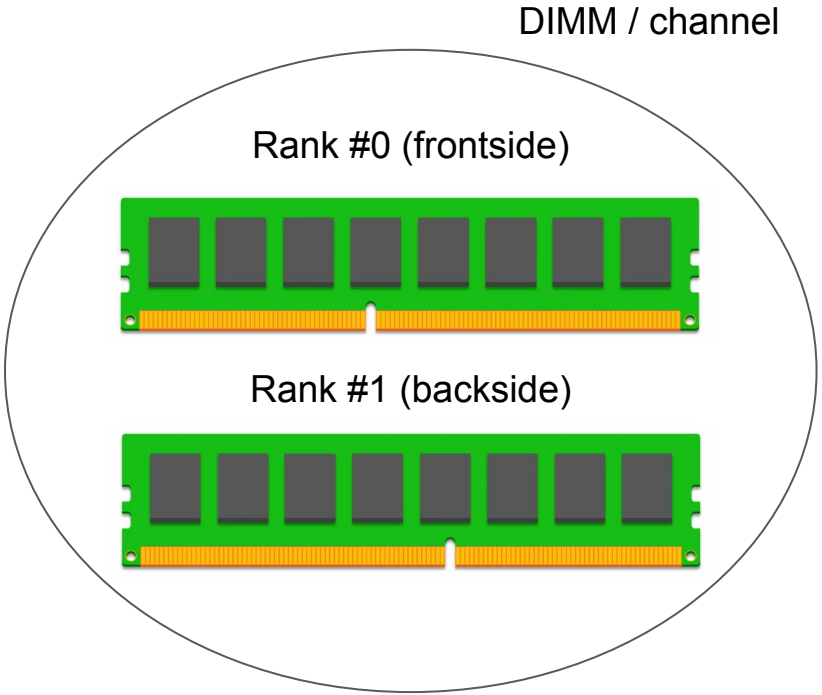
DIMM / channel



DRAM basics: a speedrun

DRAM memory is organized as follows:

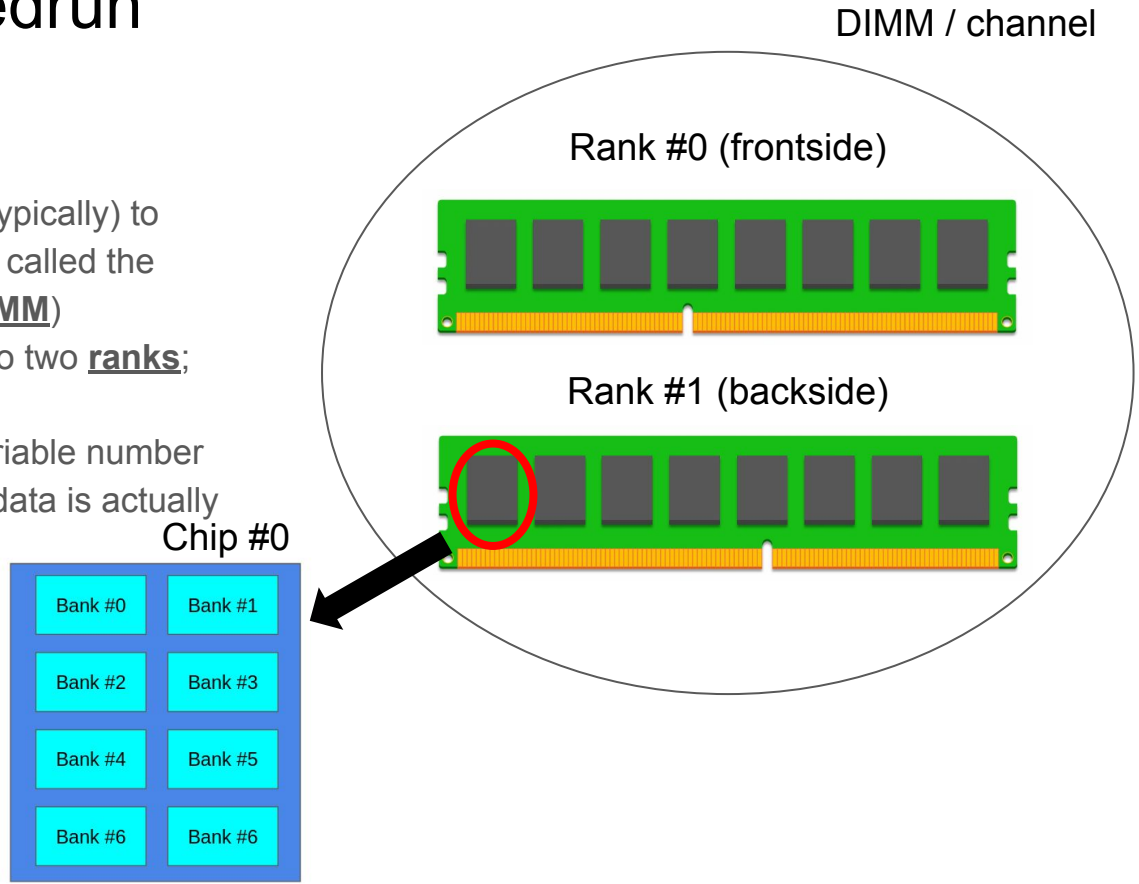
- Each “stick” of RAM corresponds (typically) to one **channel** (the “stick” is what it’s called the Dual Inline Memory Module, the **DIMM**)
- Each DIMM is divided (typically) into two **ranks**; each side of the DIMM is a rank



DRAM basics: a speedrun

DRAM memory is organized as follows:

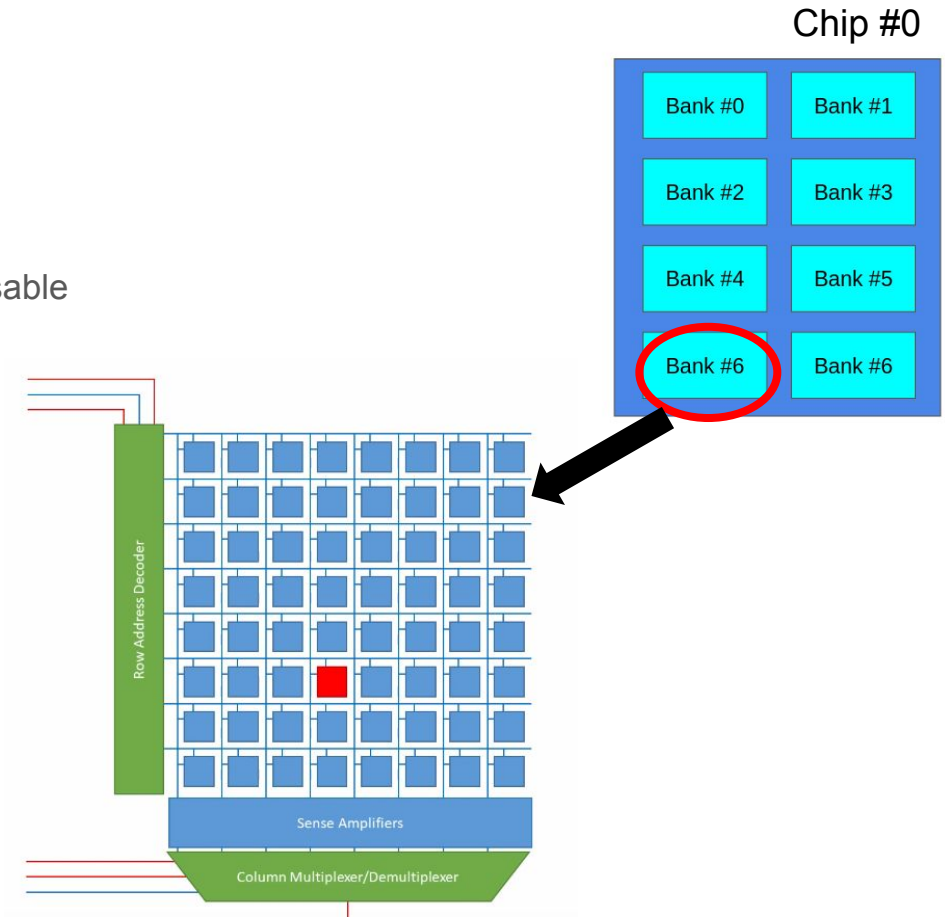
- Each “stick” of RAM corresponds (typically) to one **channel** (the “stick” is what it’s called the Dual Inline Memory Module, the **DIMM**)
- Each DIMM is divided (typically) into two **ranks**; each side of the DIMM is a rank
- Each rank contains (in DDR3) a variable number of **banks**, typically 8; this is where data is actually stored



DRAM basics: a speedrun

DRAM memory is organized as follows:

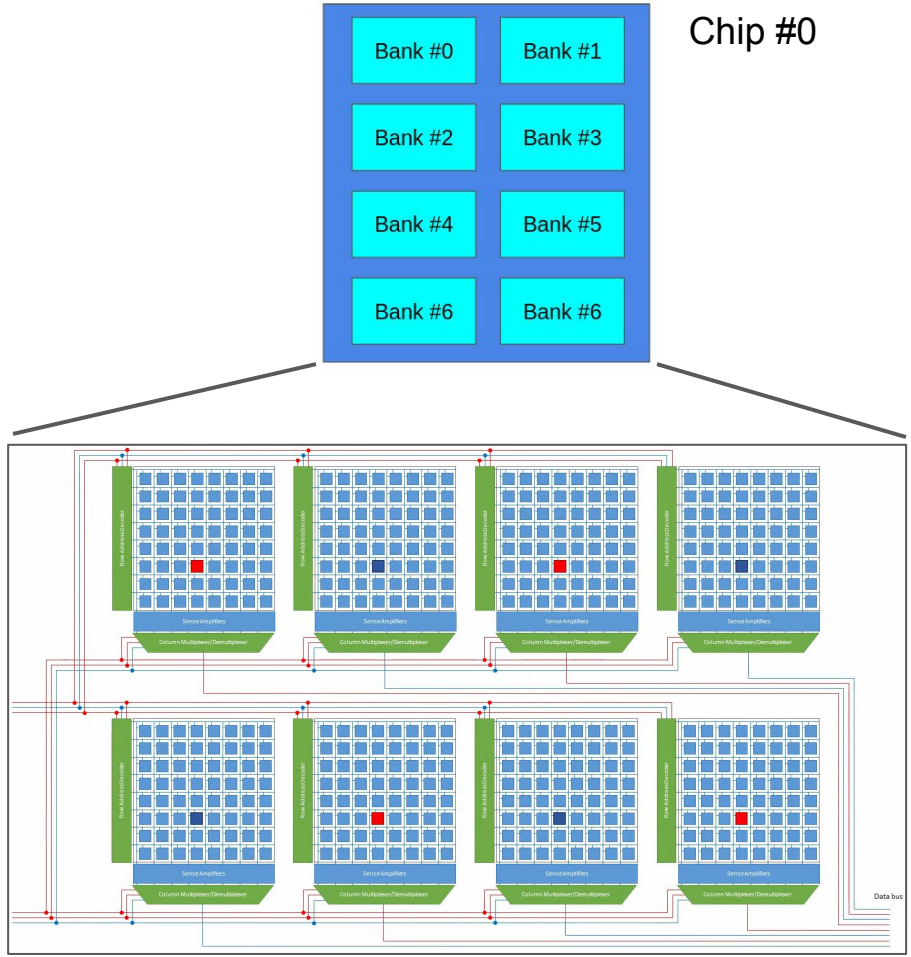
- …
- Banks are **data arrays** of **1T1C** cells addressable by **row and column decoders**



DRAM basics: a speedrun

DRAM memory is organized as follows:

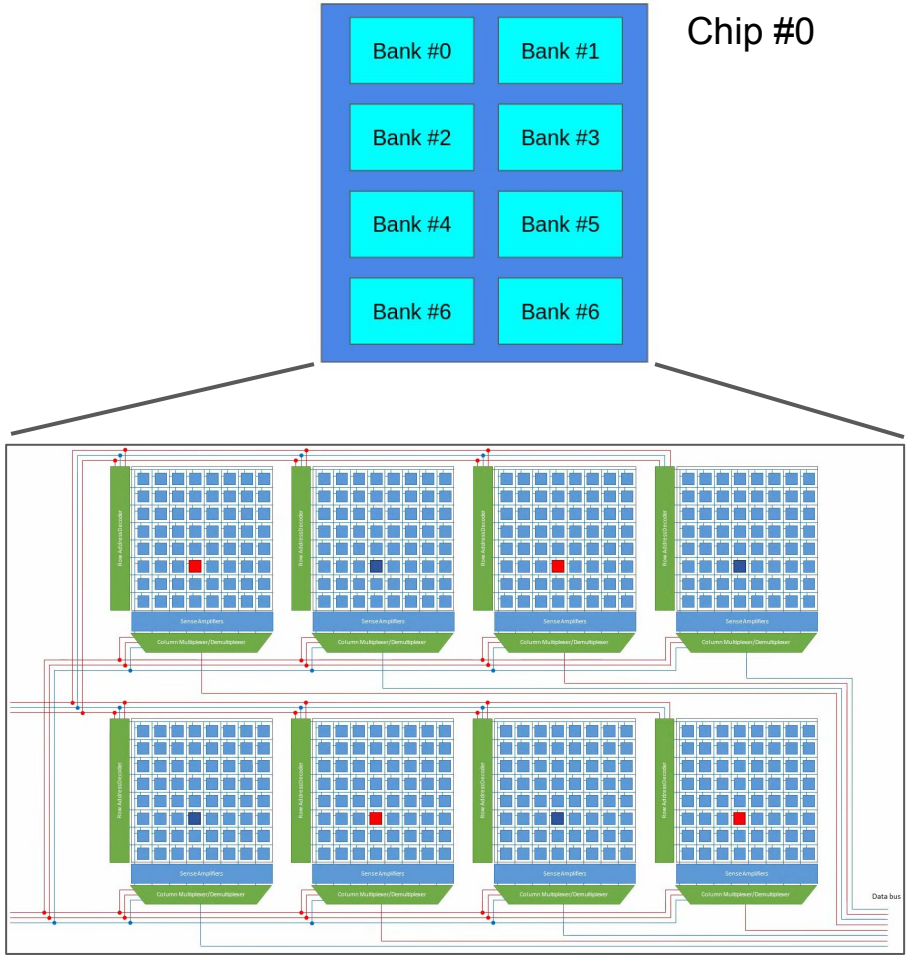
- …
- Banks are **data arrays** of **1T1C** cells addressable by **row and column decoders**
- Every chip #0-7 of one rank has 8 banks in it…



DRAM basics: a speedrun

DRAM memory is organized as follows:

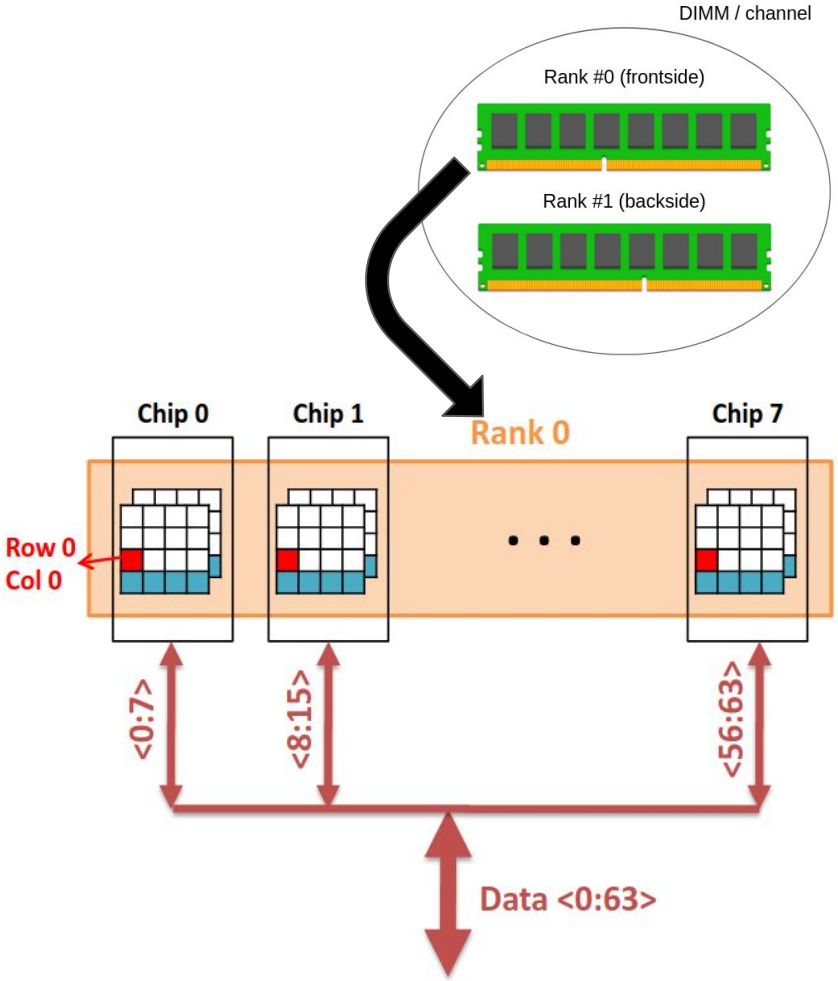
- …
- Banks are **data arrays** of **1T1C** cells addressable by **row and column decoders**
- Every chip #0-7 of one rank has 8 banks in it…
- Accesses to column #0 of row #0 of bank #0 of rank #0 of channel #0 implies accessing:



DRAM basics: a speedrun

DRAM memory is organized as follows:

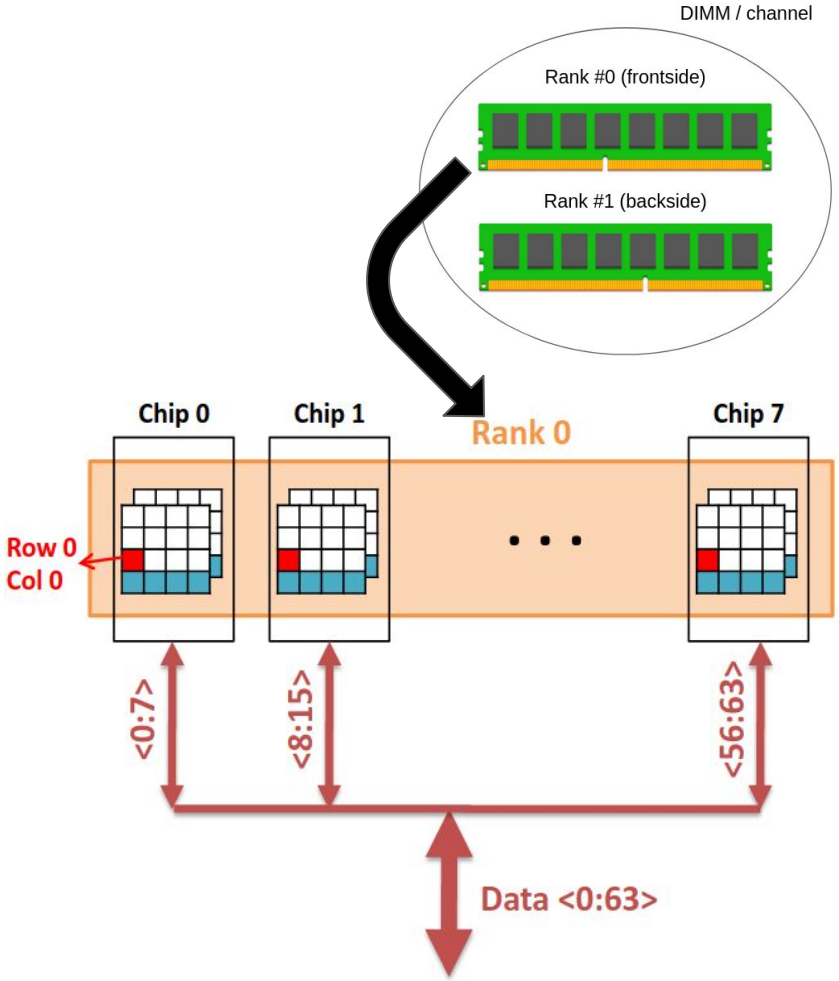
- …
- Banks are **data arrays** of **1T1C** cells addressable by **row and column decoders**
- Every chip #0-7 of one rank has 8 banks in it…
- Accesses to column #0 of row #0 of bank #0 of rank #0 of channel #0 implies accessing:



DRAM basics: a speedrun

DRAM memory is organized as follows:

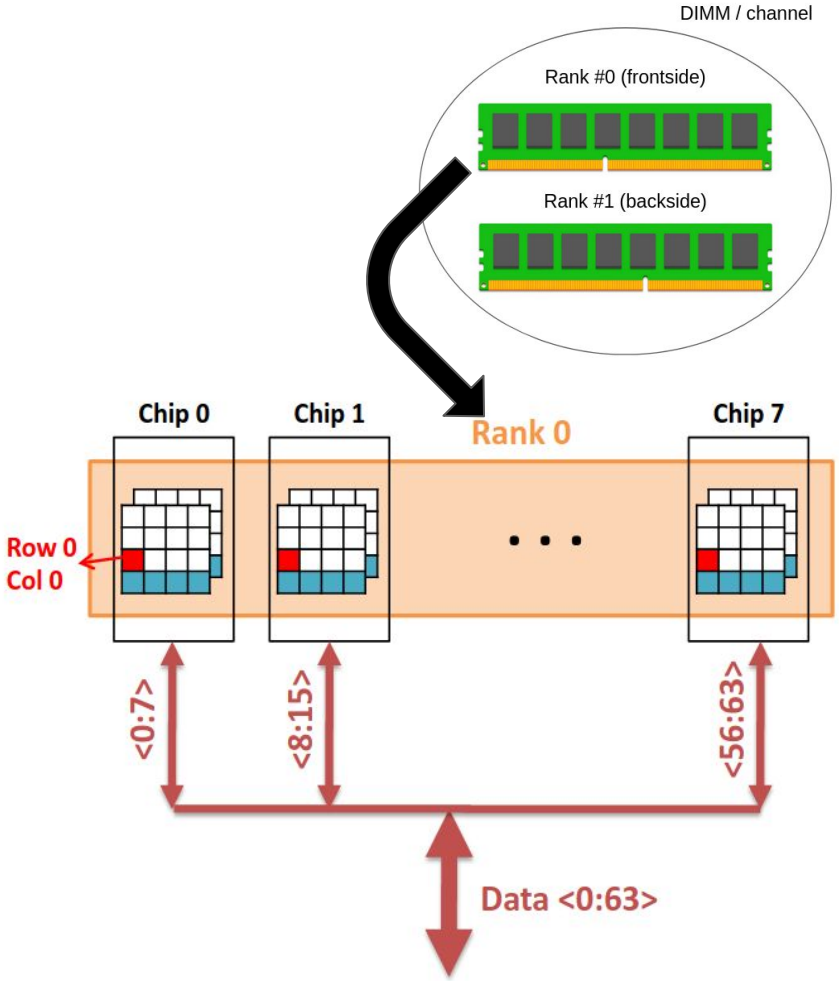
- …
- Banks are **data arrays** of **1T1C** cells addressable by **row and column decoders**
- Every chip #0-7 of one rank has 8 banks in it…
- Accesses to column #0 of row #0 of bank #0 of rank #0 of channel #0 implies accessing:
    - Every column #0



DRAM basics: a speedrun

DRAM memory is organized as follows:

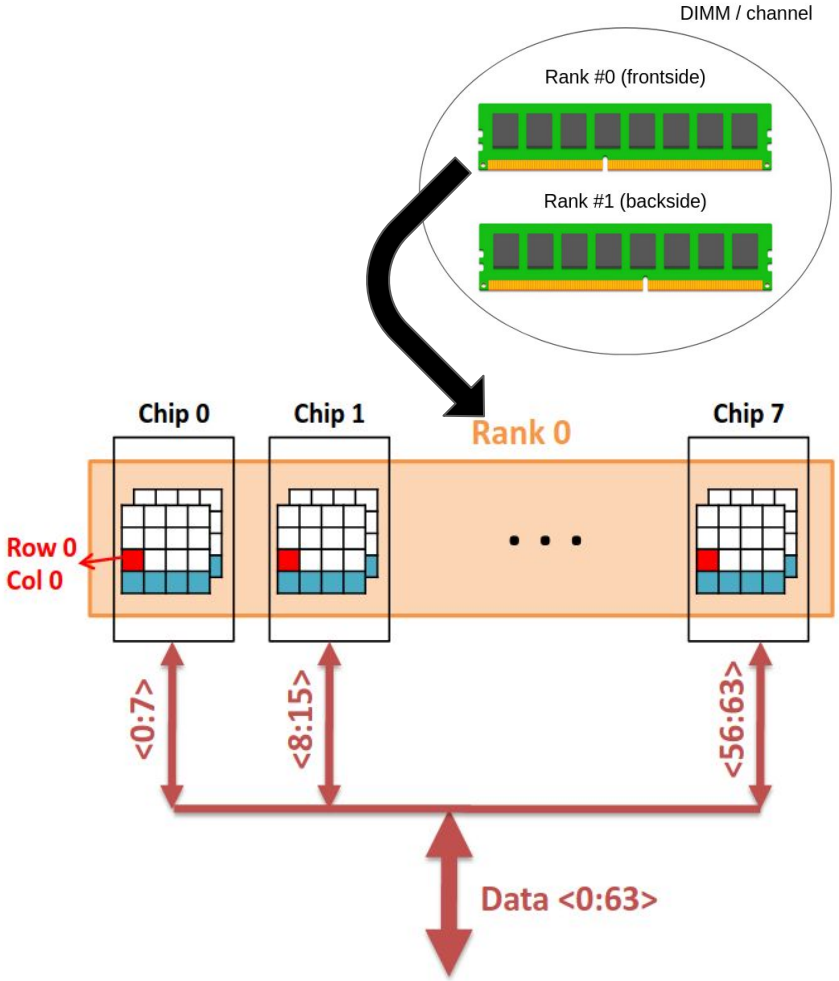
- …
- Banks are **data arrays** of **1T1C** cells addressable by **row and column decoders**
- Every chip #0-7 of one rank has 8 banks in it…
- Accesses to column #0 of row #0 of bank #0 of rank #0 of channel #0 implies accessing:
  - Every column #0
  - Of every row #0



DRAM basics: a speedrun

DRAM memory is organized as follows:

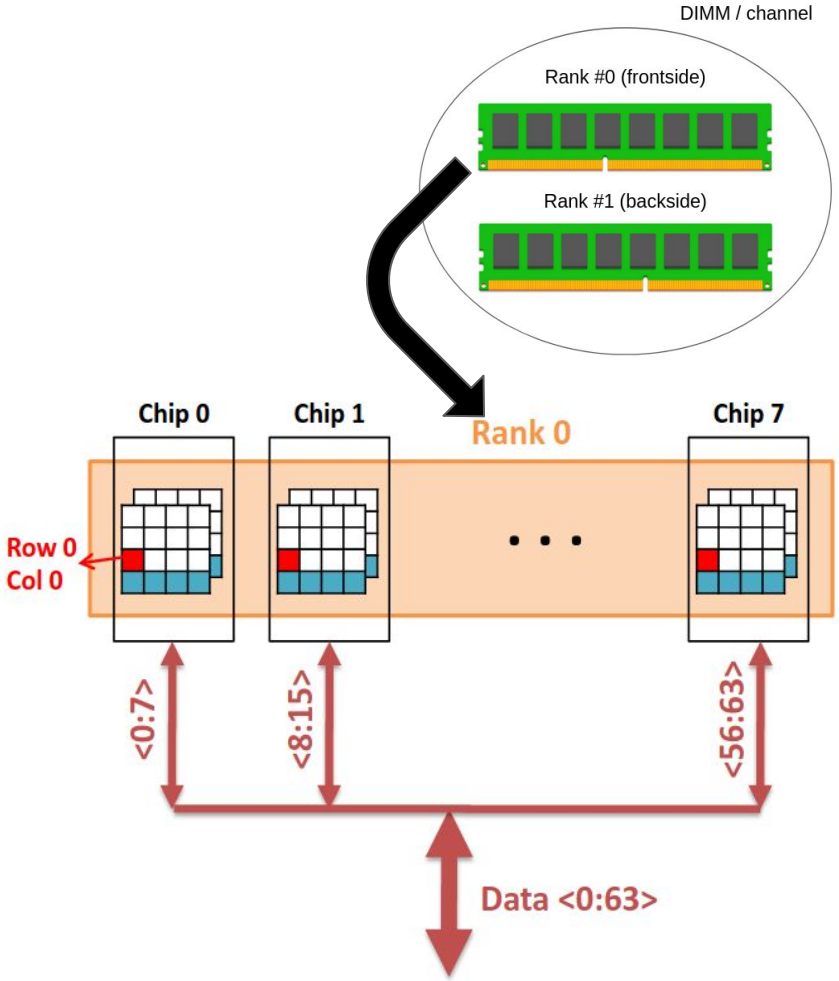
- …
- Banks are **data arrays** of **1T1C** cells addressable by **row and column decoders**
- Every chip #0-7 of one rank has 8 banks in it…
- Accesses to column #0 of row #0 of bank #0 of rank #0 of channel #0 implies accessing:
  - Every column #0
  - Of every row #0
  - Of every bank #0



DRAM basics: a speedrun

DRAM memory is organized as follows:

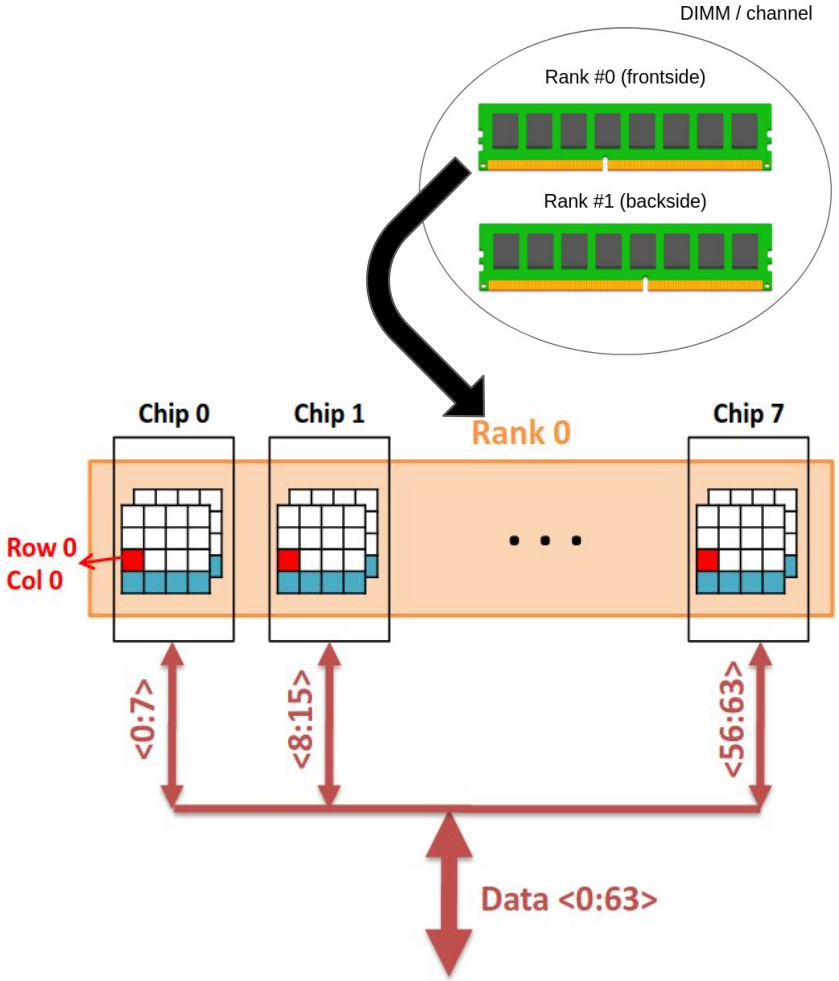
- …
- Banks are **data arrays** of **1T1C** cells addressable by **row and column decoders**
- Every chip #0-7 of one rank has 8 banks in it…
- Accesses to column #0 of row #0 of bank #0 of rank #0 of channel #0 implies accessing:
  - Every column #0
  - Of every row #0
  - Of every bank #0
  - Of every chip



DRAM basics: a speedrun

DRAM memory is organized as follows:

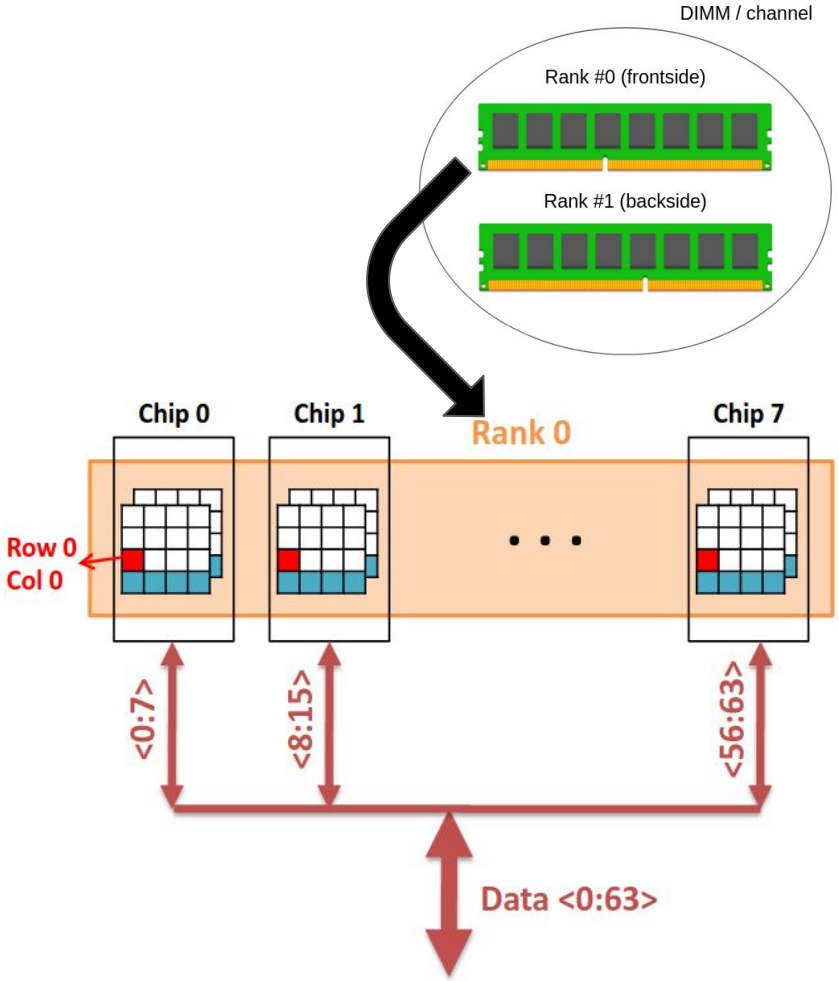
- …
- Banks are **data arrays** of **1T1C** cells addressable by **row and column decoders**
- Every chip #0-7 of one rank has 8 banks in it…
- Accesses to column #0 of row #0 of bank #0 of rank #0 of channel #0 implies accessing:
  - Every column #0
  - Of every row #0
  - Of every bank #0
  - Of every chip in rank #0



DRAM basics: a speedrun

DRAM memory is organized as follows:

- …
- Banks are **data arrays** of **1T1C** cells addressable by **row and column decoders**
- Every chip #0-7 of one rank has 8 banks in it…
- Accesses to column #0 of row #0 of bank #0 of rank #0 of channel #0 implies accessing:
  - Every column #0
  - Of every row #0
  - Of every bank #0
  - Of every chip in rank #0
  - In channel #0

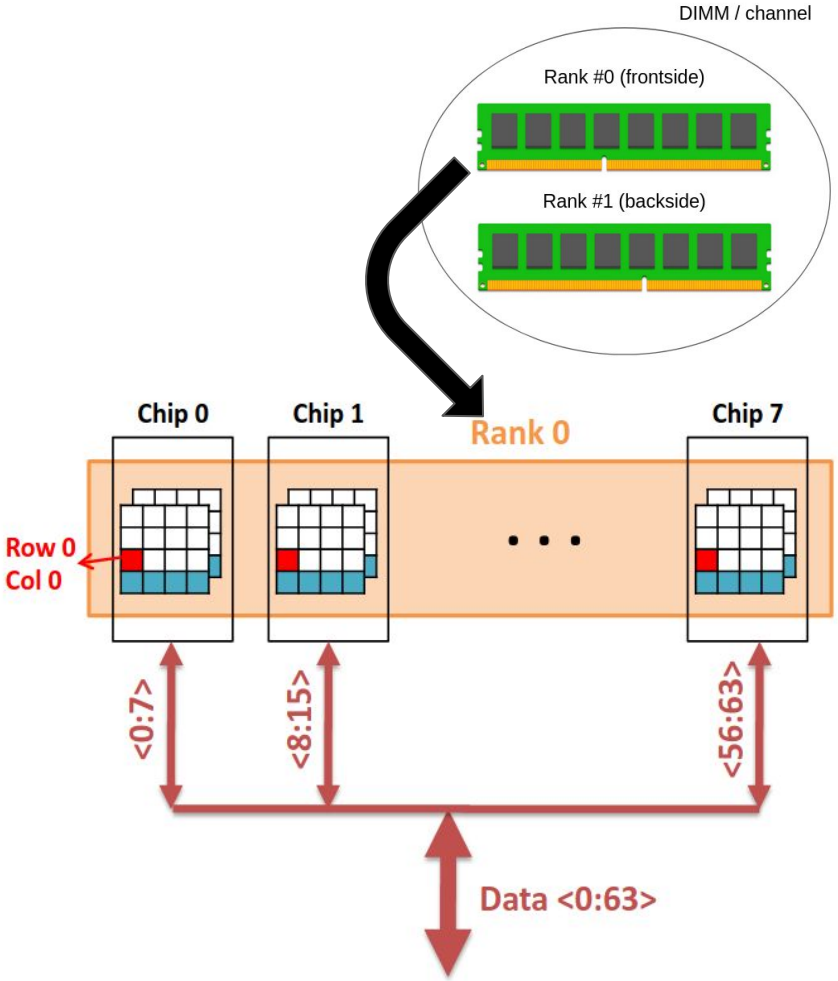


DRAM basics: a speedrun

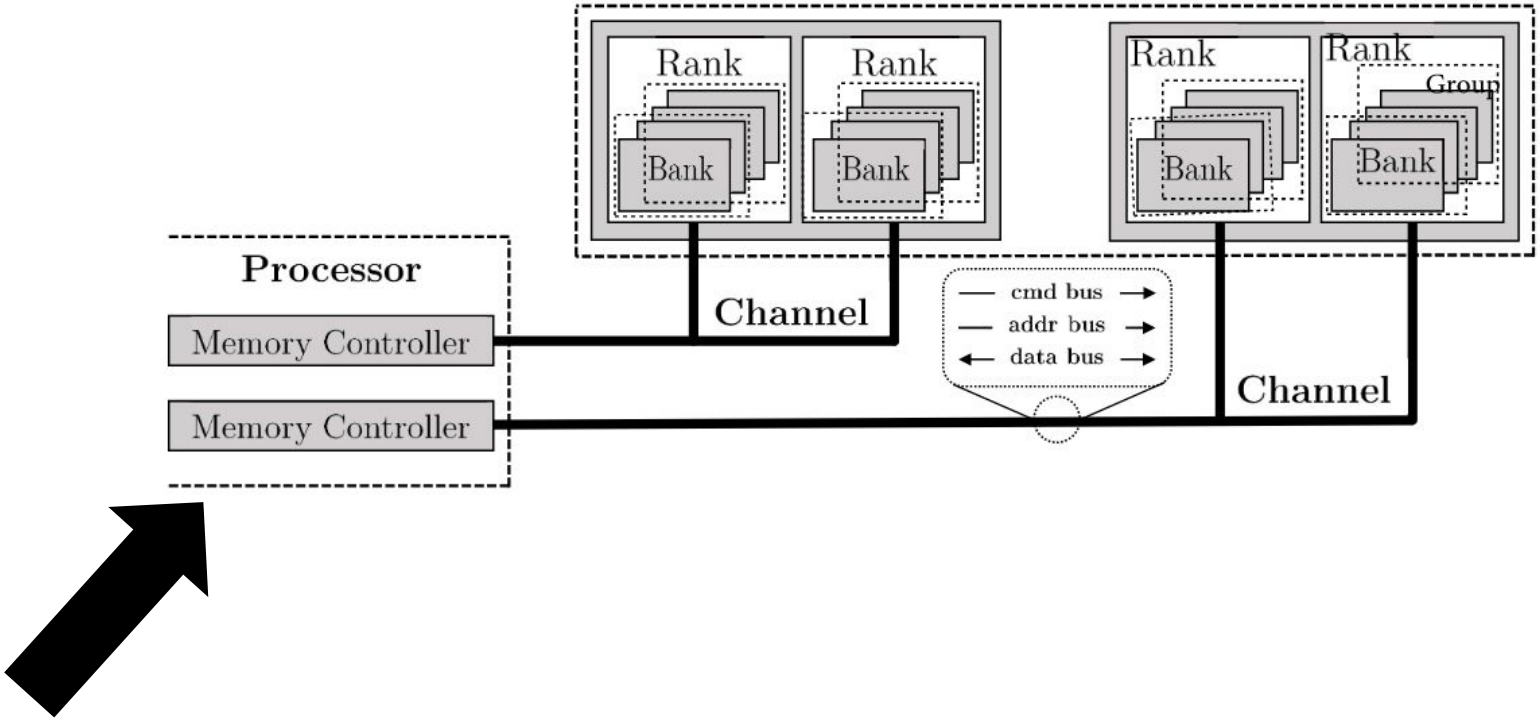
DRAM memory is organized as follows:

- …
- In summary:

The outcoming 64-bit value is interleaved between the 8 column #0's (8-bit wide) of the 8 bank #0's of the 8 chips of rank #0 in the DIMM of channel #0!!!

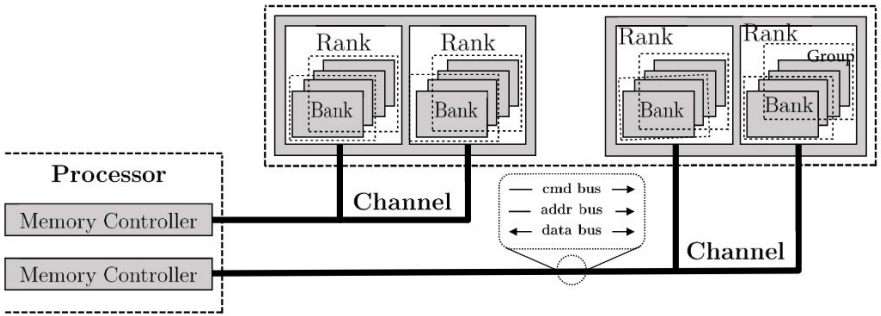


Memory controllers



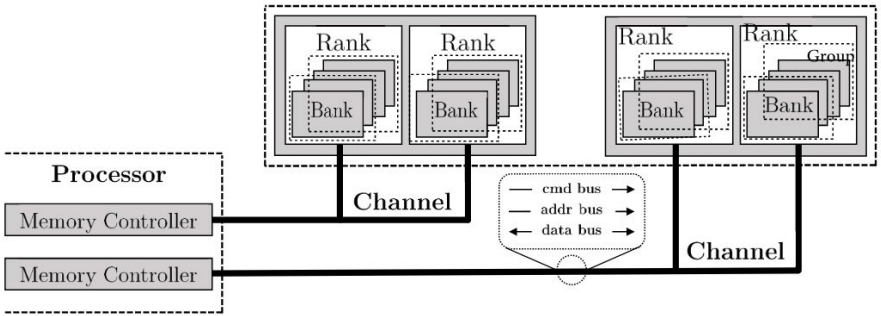
Memory controllers

- DRAM MCs manage **read** and **write transactions** from CPU to DRAM
  - Read transactions → cache misses
  - Write transactions → dirty cache lines or data from HDD



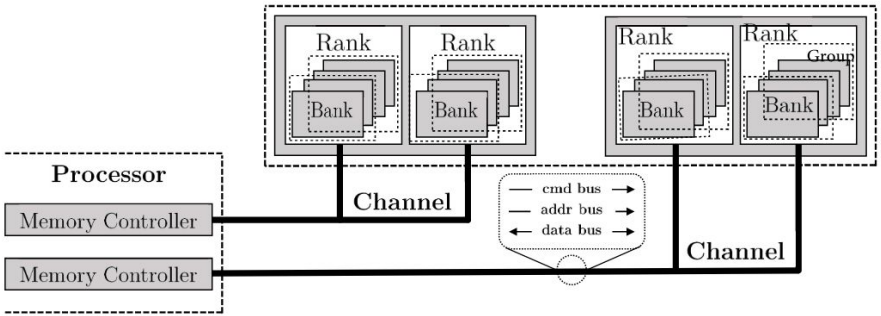
Memory controllers

- 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**



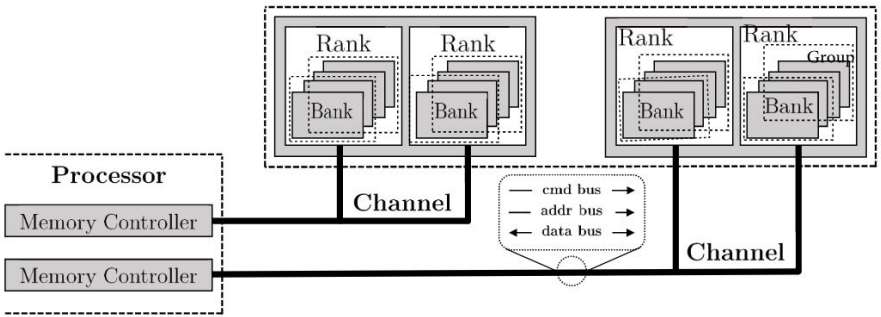
Memory controllers

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



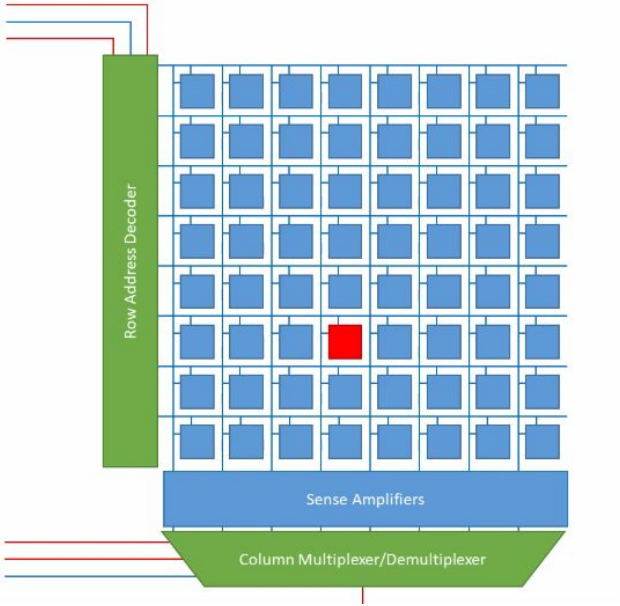
Memory controllers

- 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???



Memory controllers

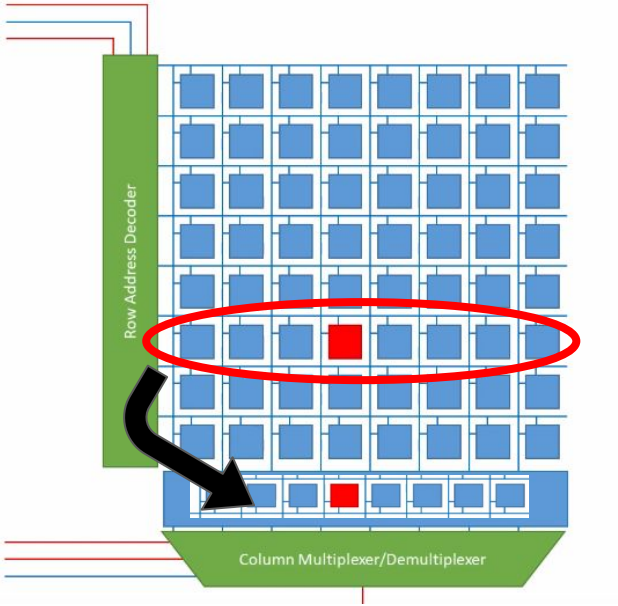
Let’s get into DRAM commands and JEDEC timing constraints!! :)



Memory controllers

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

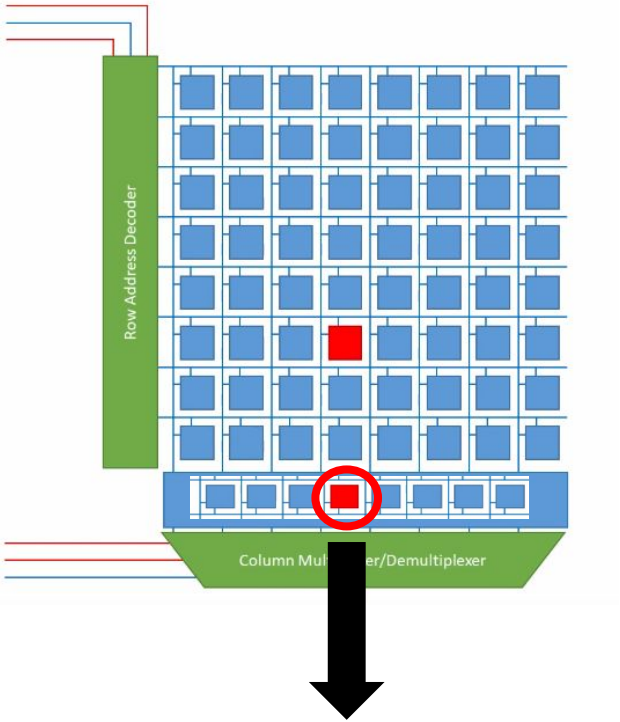
- Moving data from the data array to the sense amplifiers is done through an **ACTIVATE command (ACT)**



Memory controllers

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

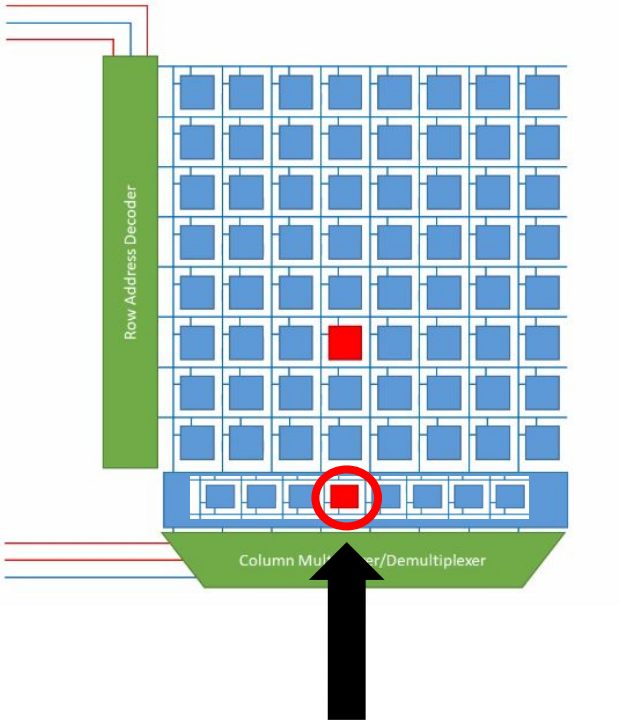
- Moving data from the data array to the sense amplifiers is done through an **ACTIVATE command (ACT)**
- Once data is residing in the sense amplifiers (also called row buffer), data can be either **READ**



Memory controllers

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

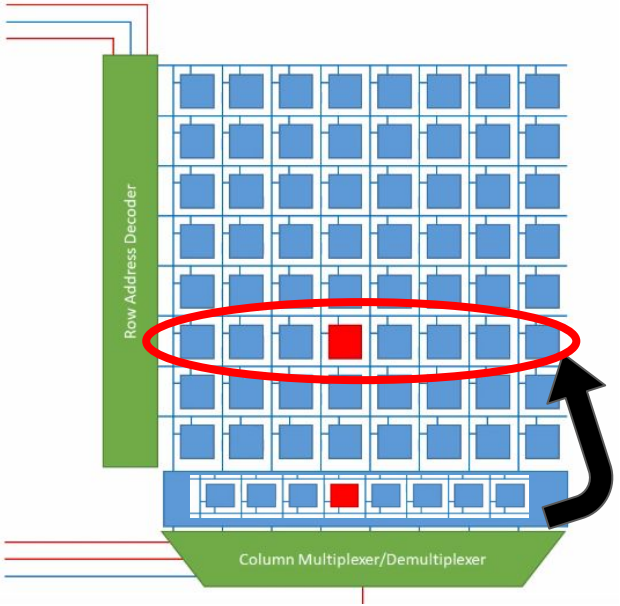
- Moving data from the data array to the sense amplifiers is done through an **ACTIVATE command (ACT)**
- Once data is residing in the sense amplifiers (also called row buffer), data can be either **READ** or **WRITTEN**



Memory controllers

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

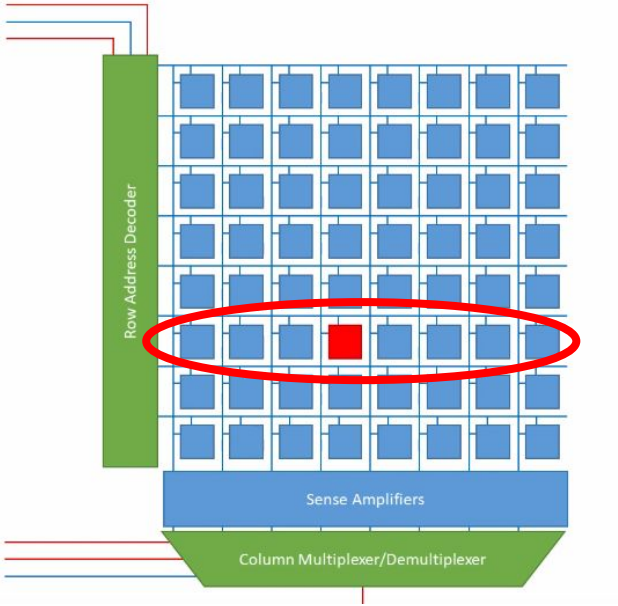
- Moving data from the data array to the sense amplifiers is done through an **ACTIVATE command (ACT)**
- Once data is residing in the sense amplifiers (also called row buffer), data can be either **READ** or **WRITTEN**
- If another row is needed, data residing in the row buffer must be brought back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



Memory controllers

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

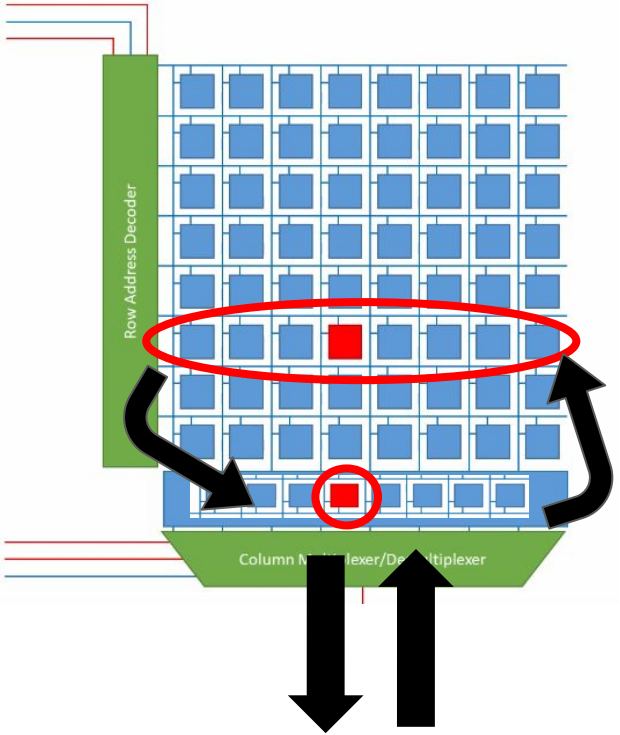
- Moving data from the data array to the sense amplifiers is done through an **ACTIVATE command (ACT)**
- Once data is residing in the sense amplifiers (also called row buffer), data can be either **READ** or **WRITTEN**
- If another row is needed, data residing in the row buffer must be brought back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



Memory controllers

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…



Memory controllers

DRAM commands must respect certain **timing constraints** to ensure integrity of data:

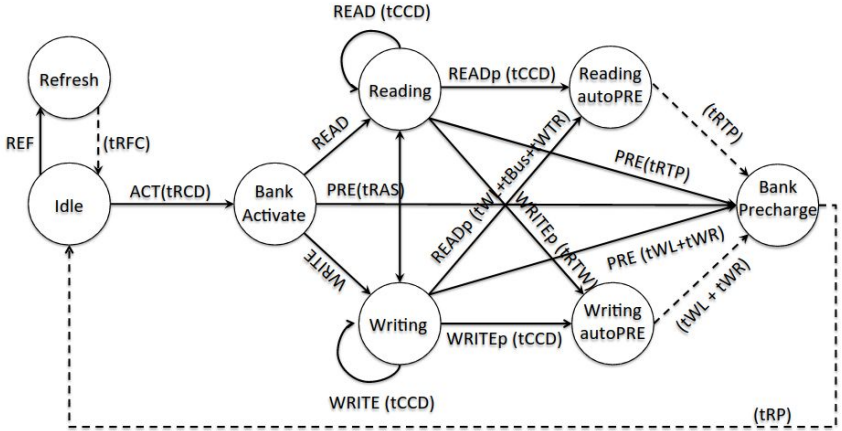


Figure 2.2: DRAM Operation State Machine.

Memory controllers

DRAM commands must respect certain **timing constraints** to ensure integrity of data:

- This has to do with the ability of DRAM's internal HW components (1T1C cells, sense amplifiers, I/O gatings, latches…) to **charge/discharge at certain speeds** or with the **capacity to hold their values** for extended periods of time

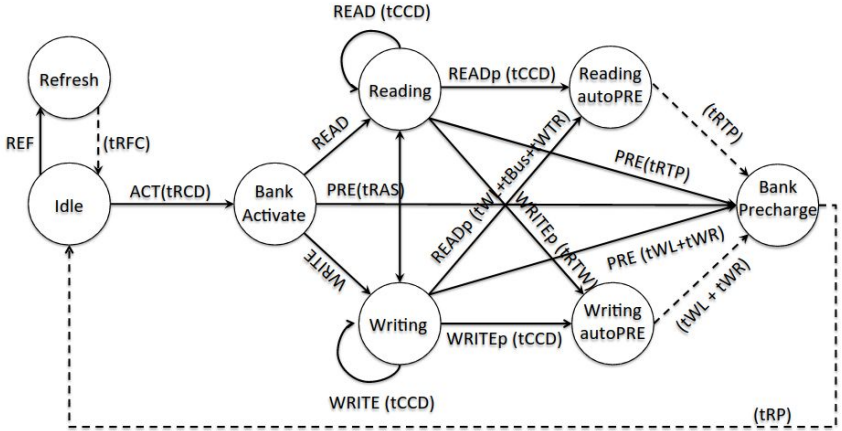


Figure 2.2: DRAM Operation State Machine.

Memory controllers

DRAM commands must respect certain **timing constraints** to ensure integrity of data:

- This has to do with the ability of DRAM's internal HW components (1T1C cells, sense amplifiers, I/O gatings, latches…) to **charge/discharge at certain speeds** or with the **capacity to hold their values** for extended periods of time
- The physical characteristics of components **require the application of certain timing constraints between DRAM commands**

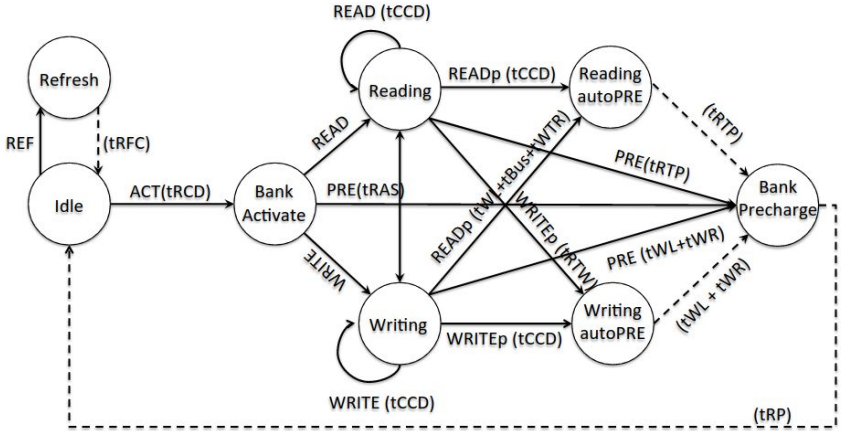


Figure 2.2: DRAM Operation State Machine.

Memory controllers

DRAM commands must respect certain **timing constraints** to ensure integrity of data:

- This has to do with the ability of DRAM's internal HW components (1T1C cells, sense amplifiers, I/O gatings, latches…) to **charge/discharge at certain speeds** or with the **capacity to hold their values** for extended periods of time
- The physical characteristics of components **require the application of certain timing constraints between DRAM commands**
    - Imagine the sense amplifiers requiring some time to "stabilize the charge" of its values before they can be read…

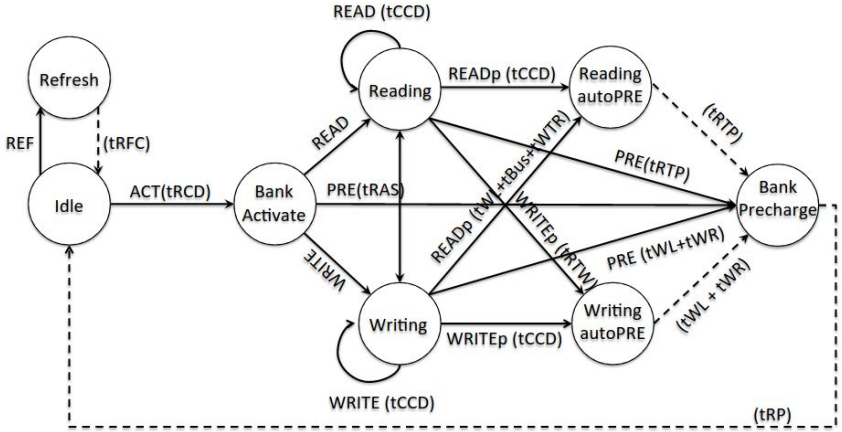
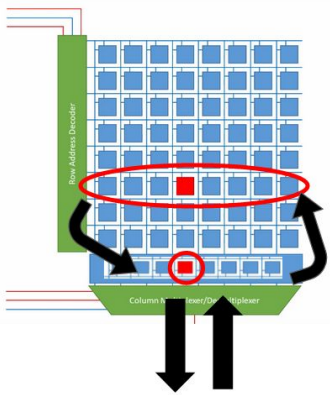


Figure 2.2: DRAM Operation State Machine.

Memory controllers

DRAM commands must respect certain **timing constraints** to ensure integrity of data:

- This has to do with the ability of DRAM's internal HW components (1T1C cells, sense amplifiers, I/O gatings, latches…) to **charge/discharge at certain speeds** or with the **capacity to hold their values** for extended periods of time
- The physical characteristics of components **require the application of certain timing constraints between DRAM commands**
  - Imagine the sense amplifiers requiring some time to "stabilize the charge" of its values before they can be read…
- These timing values are **measured in cycles**



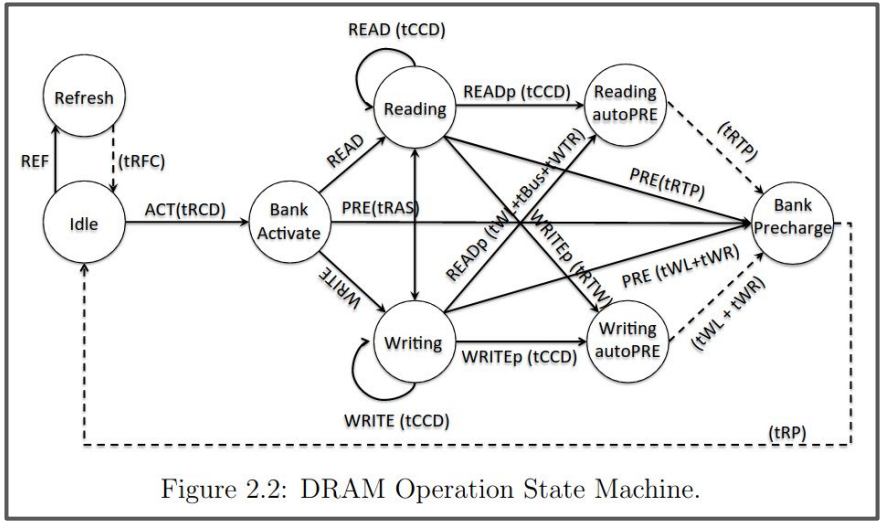


Figure 2.2: DRAM Operation State Machine.

Memory controllers

DRA
con

-
-

Timing constraints			
tCK	1	tRRD_L	7
AL	0	tWTR_S	3
CL	15	tWTR_L	8
CWL	11	tFAW	32
tRCD	15	tWR	16
tRP	15	tWR2	17
tRAS	36	tRTP	8
tRFC	374	tCCD_S	4
tRFC2	278	tCCD_L	6
tRFC4	171	tCKE	6
tREFI	8328	tCKESR	7
tRPRE	1	tXS	385
tWPRE	1	tXP	7
tRRD_S	6	tRTRS	1
read_delay	RL + tBURST = AL + CL + BL/2	write_delay	WL + tBURST = AL + CWL + BL/2

values before they can be read…

- These timing values are **measured in cycles**

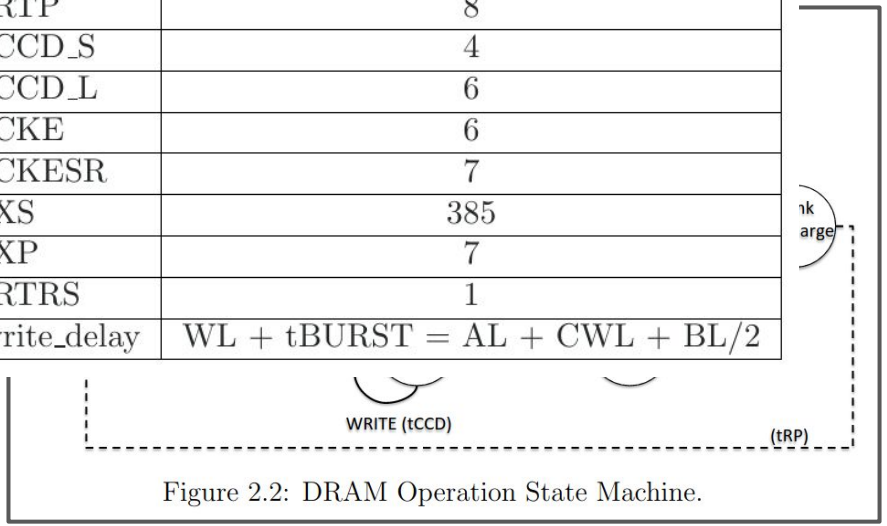
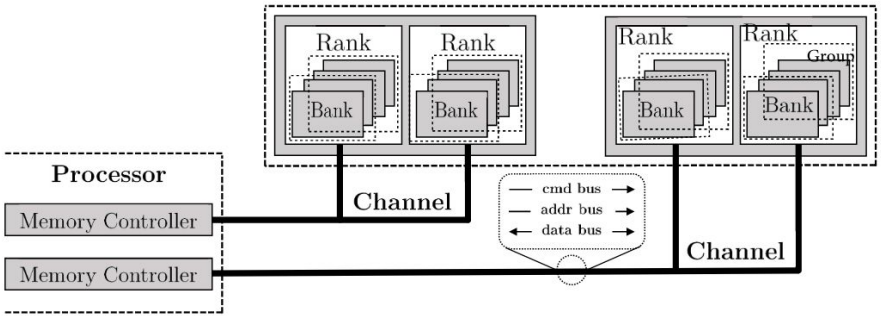


Figure 2.2: DRAM Operation State Machine.

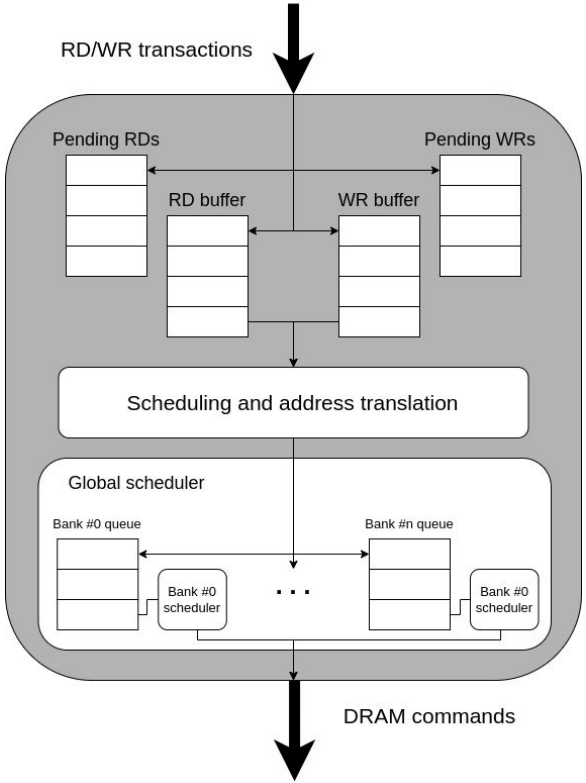
Memory controllers

- 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 the CPU 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…**

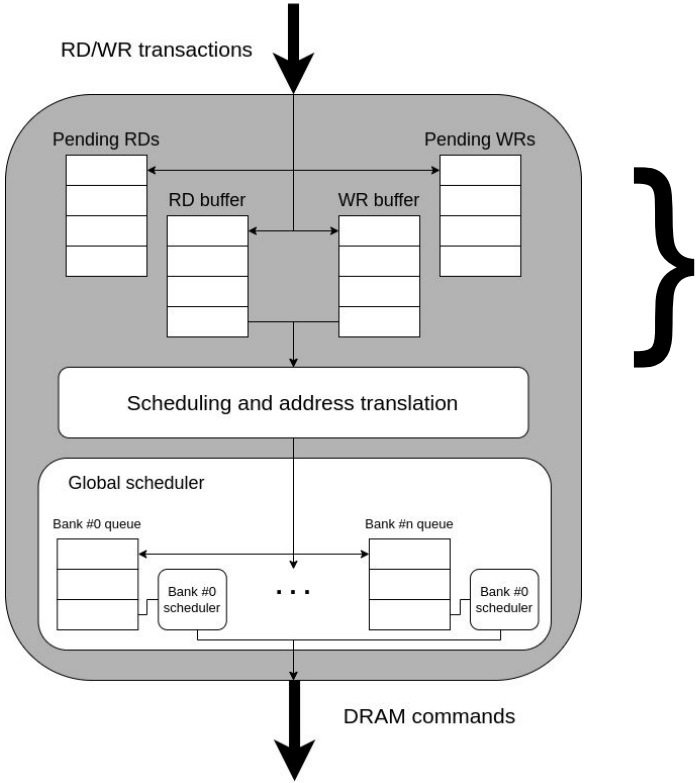
…while adhering to JEDEC timing constraints and respecting data and device integrity (refreshes, activation windows, device temperature, ZQ calibration…)!!!



Structure (logical) of a memory controller

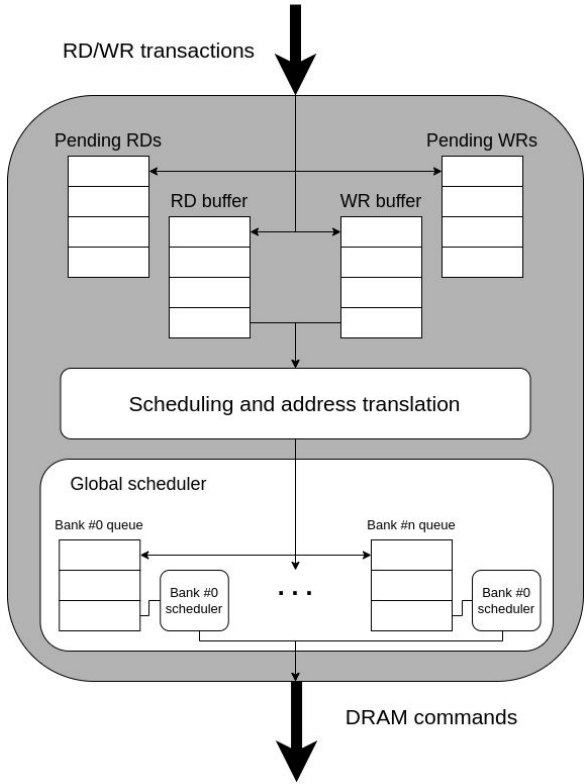


Structure (logical) of a memory controller



Transaction-level management

Structure (logical) of a memory controller

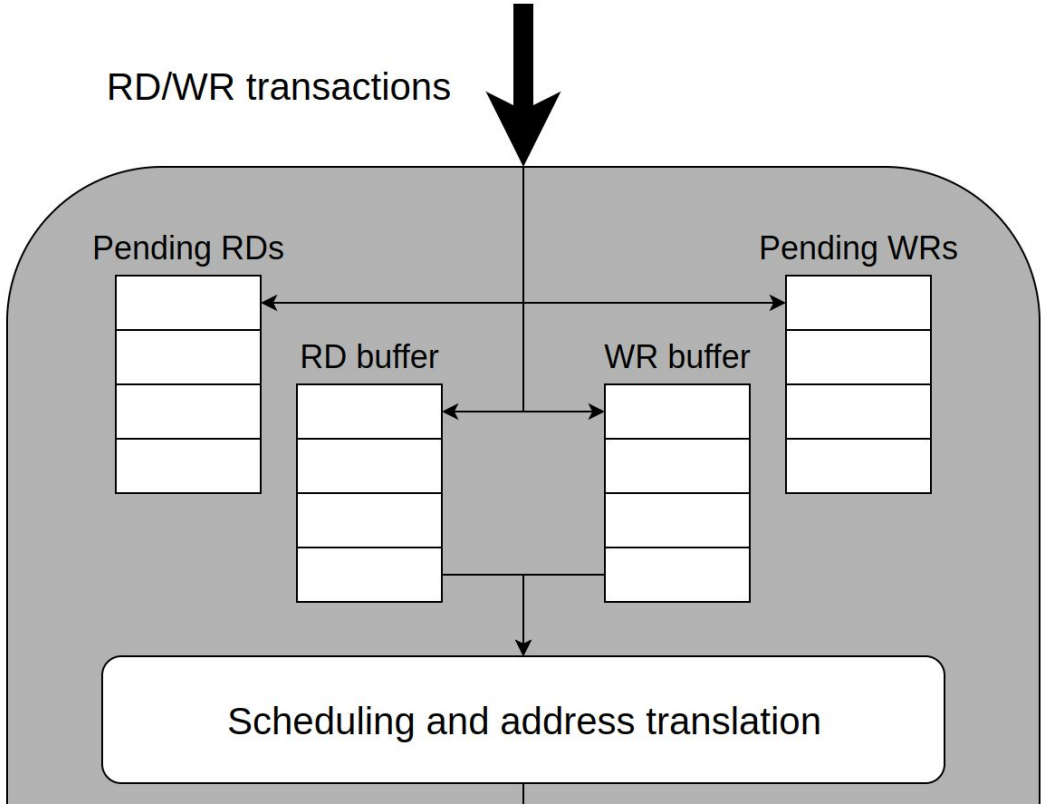


Transaction-level management

DRAM command generation

Structure (logical) of a memory controller

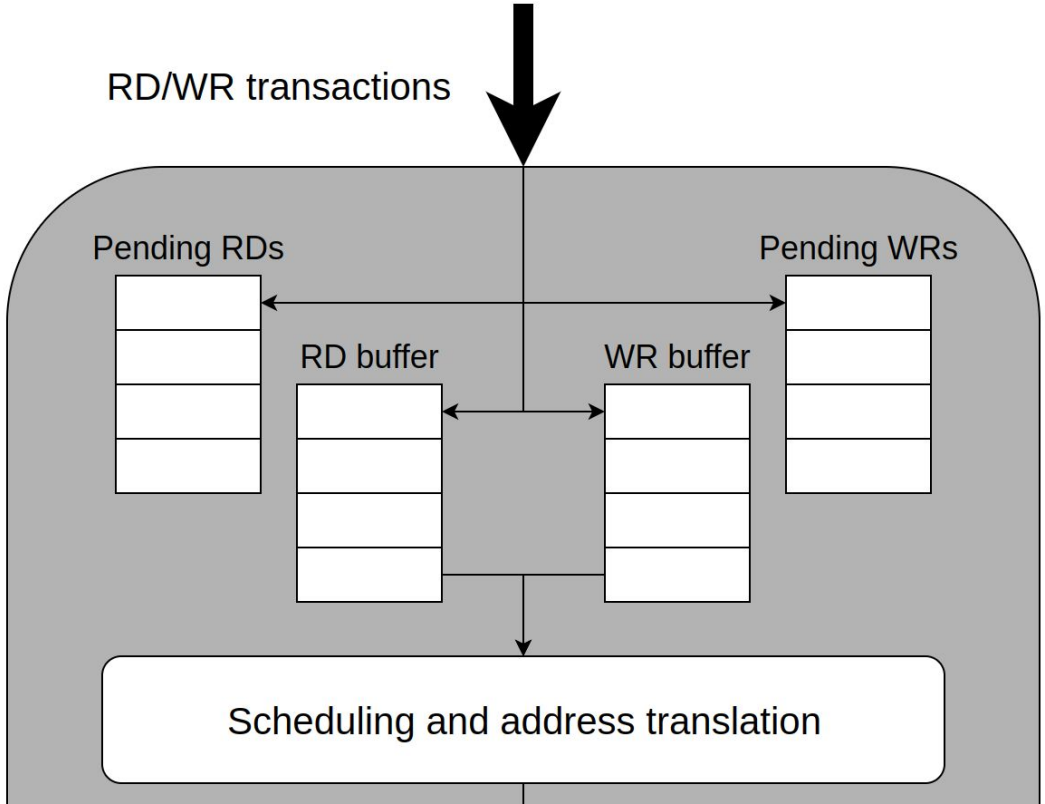
Let’s start with transaction management:



Structure (logical) of a memory controller

Let’s start with transaction management:

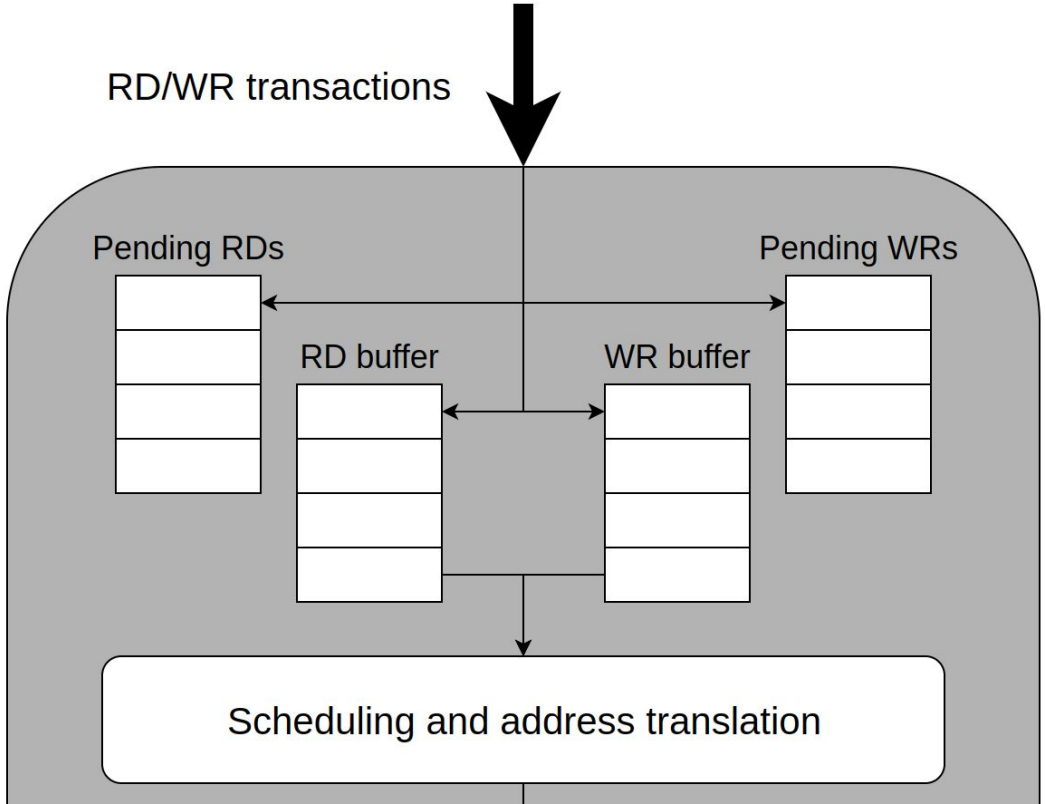
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**



Structure (logical) of a memory controller

Let’s start with transaction management:

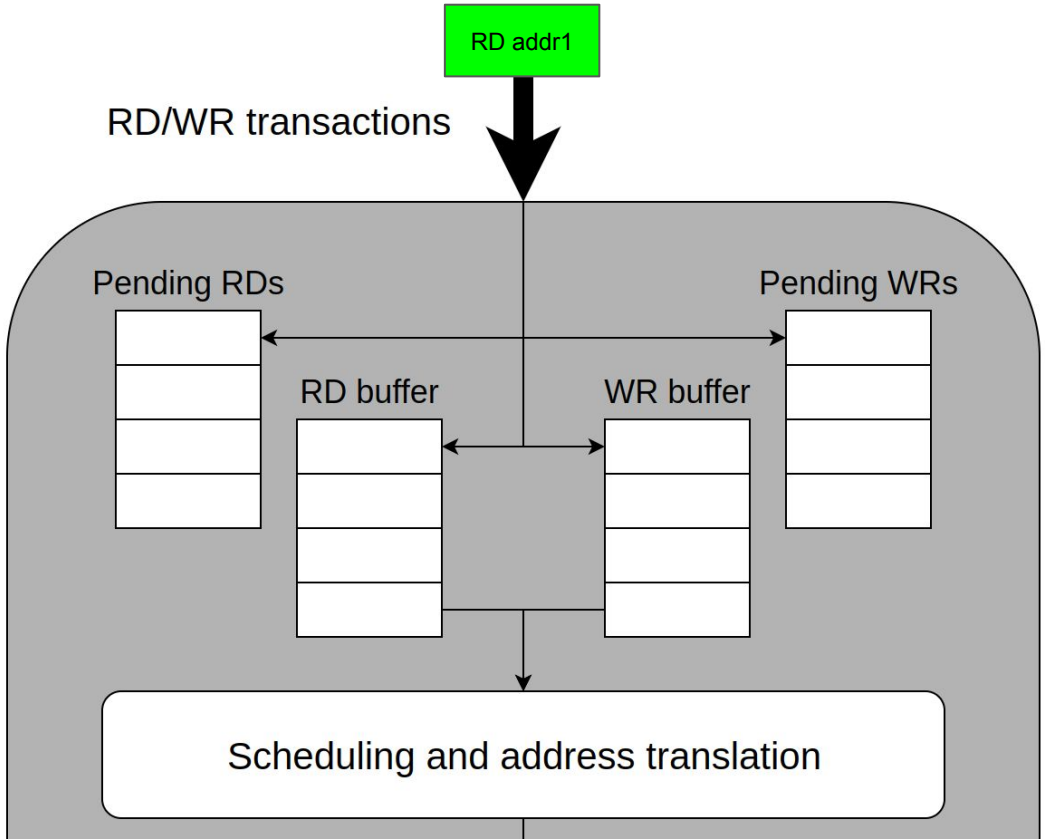
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
  - They keep a “state” of sorts, ensuring that no incoming transactions are repeated or to send back RD operations for which there is a write pending



Structure (logical) of a memory controller

Let's start with transaction management:

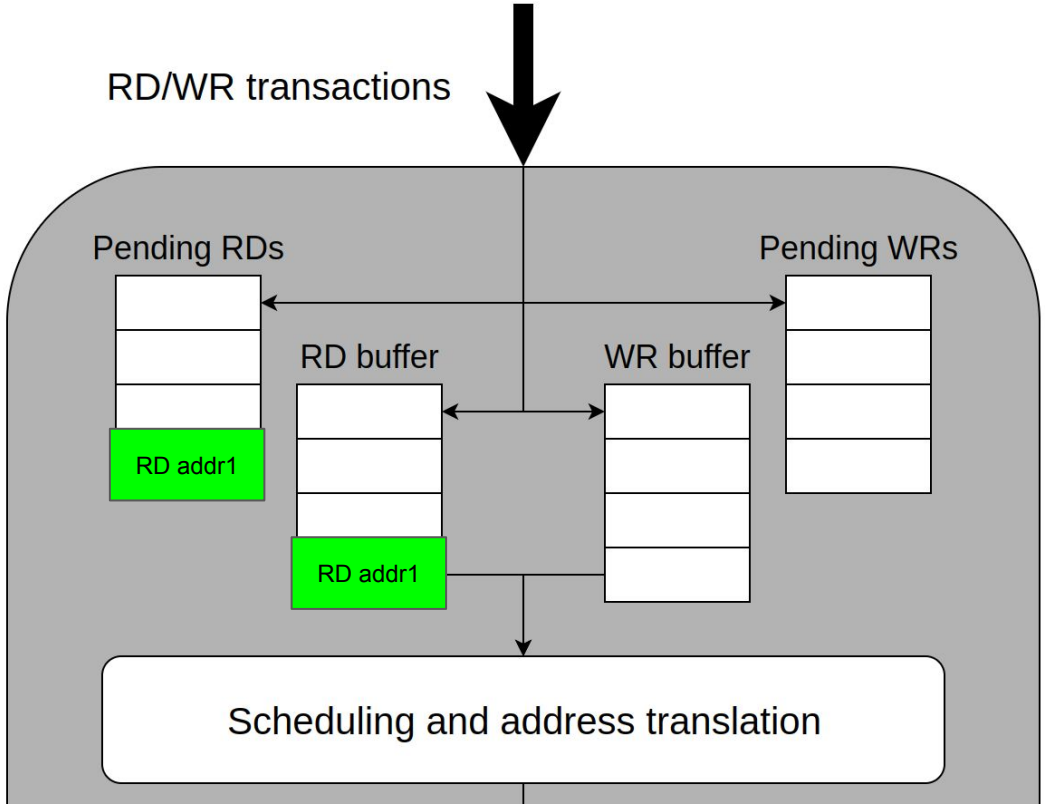
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
  - They keep a "state" of sorts, ensuring that no incoming transactions are repeated or to send back RD operations for which there is a write pending



Structure (logical) of a memory controller

Let’s start with transaction management:

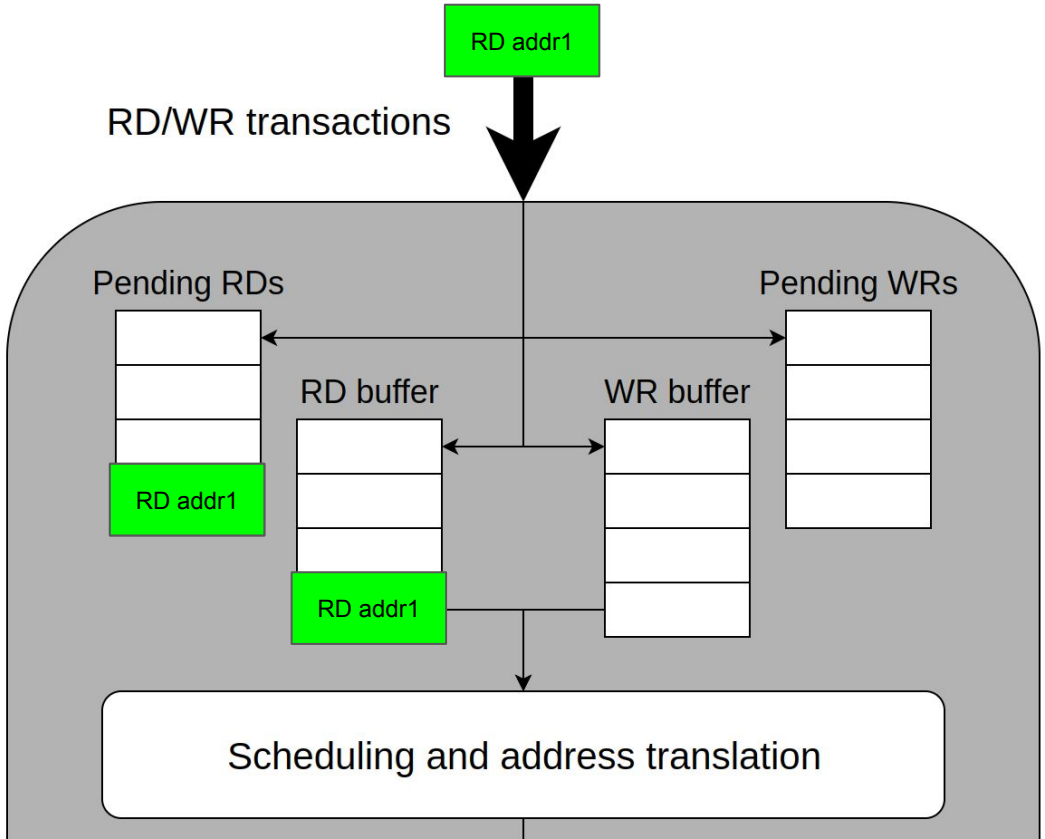
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
  - They keep a “state” of sorts, ensuring that no incoming transactions are repeated or to send back RD operations for which there is a write pending



Structure (logical) of a memory controller

Let’s start with transaction management:

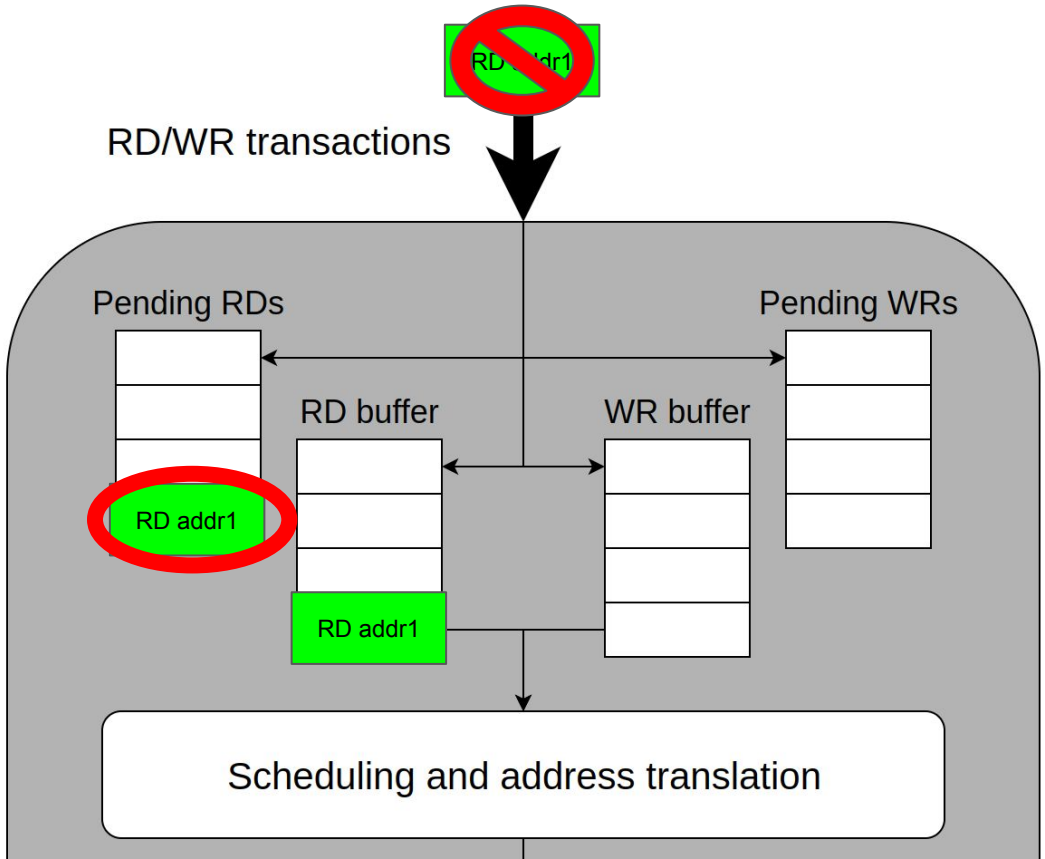
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
  - They keep a “state” of sorts, ensuring that no incoming transactions are repeated or to send back RD operations for which there is a write pending



Structure (logical) of a memory controller

Let’s start with transaction management:

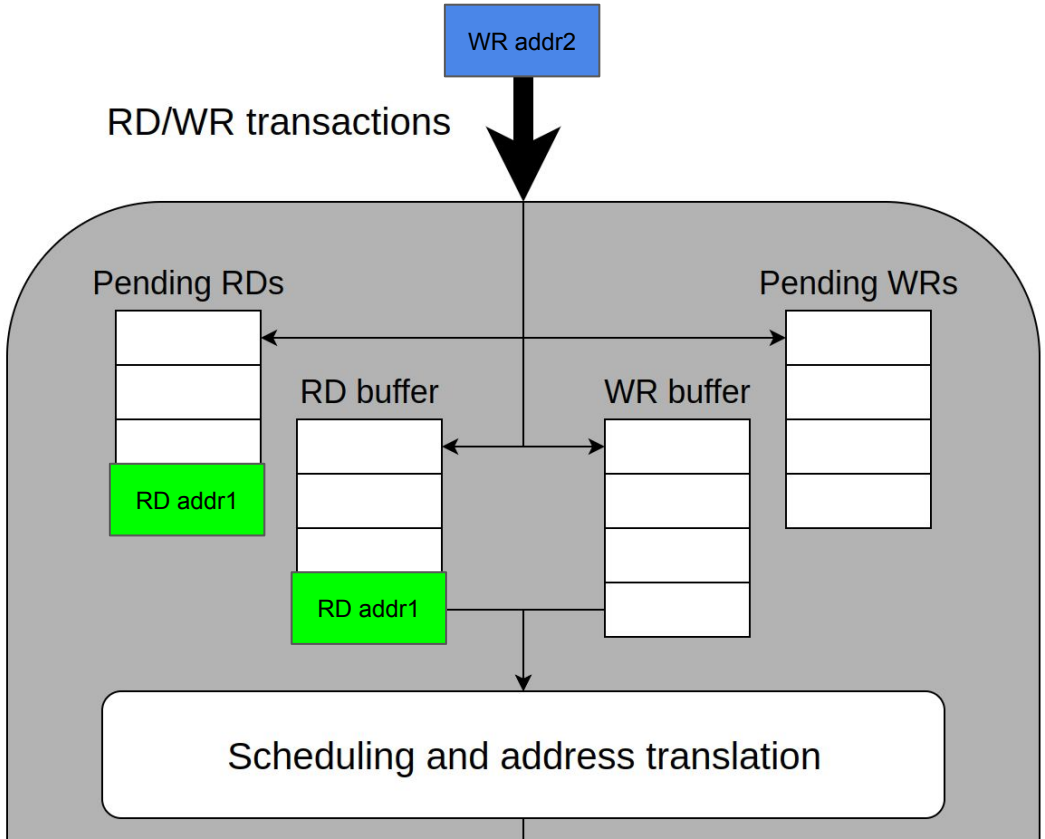
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
  - They keep a “state” of sorts, ensuring that no incoming transactions are repeated or to send back RD operations for which there is a write pending



Structure (logical) of a memory controller

Let’s start with transaction management:

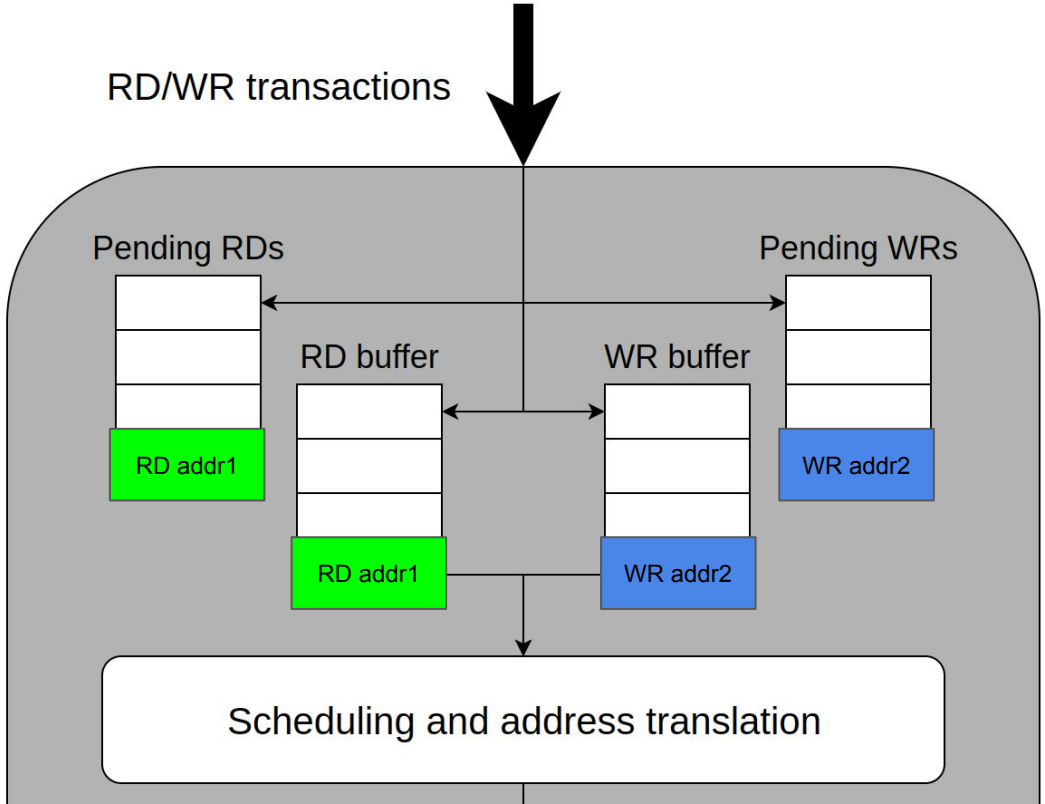
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
  - They keep a “state” of sorts, ensuring that no incoming transactions are repeated or to send back RD operations for which there is a write pending



Structure (logical) of a memory controller

Let’s start with transaction management:

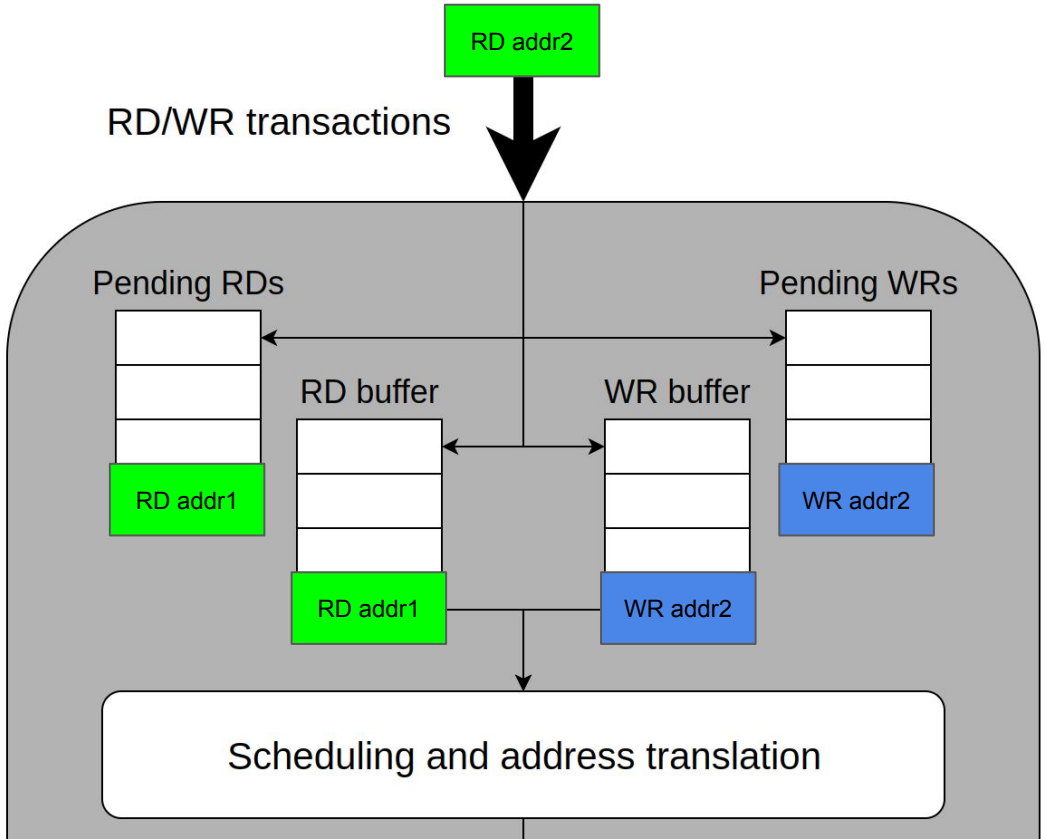
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
  - They keep a “state” of sorts, ensuring that no incoming transactions are repeated or to send back RD operations for which there is a write pending



Structure (logical) of a memory controller

Let’s start with transaction management:

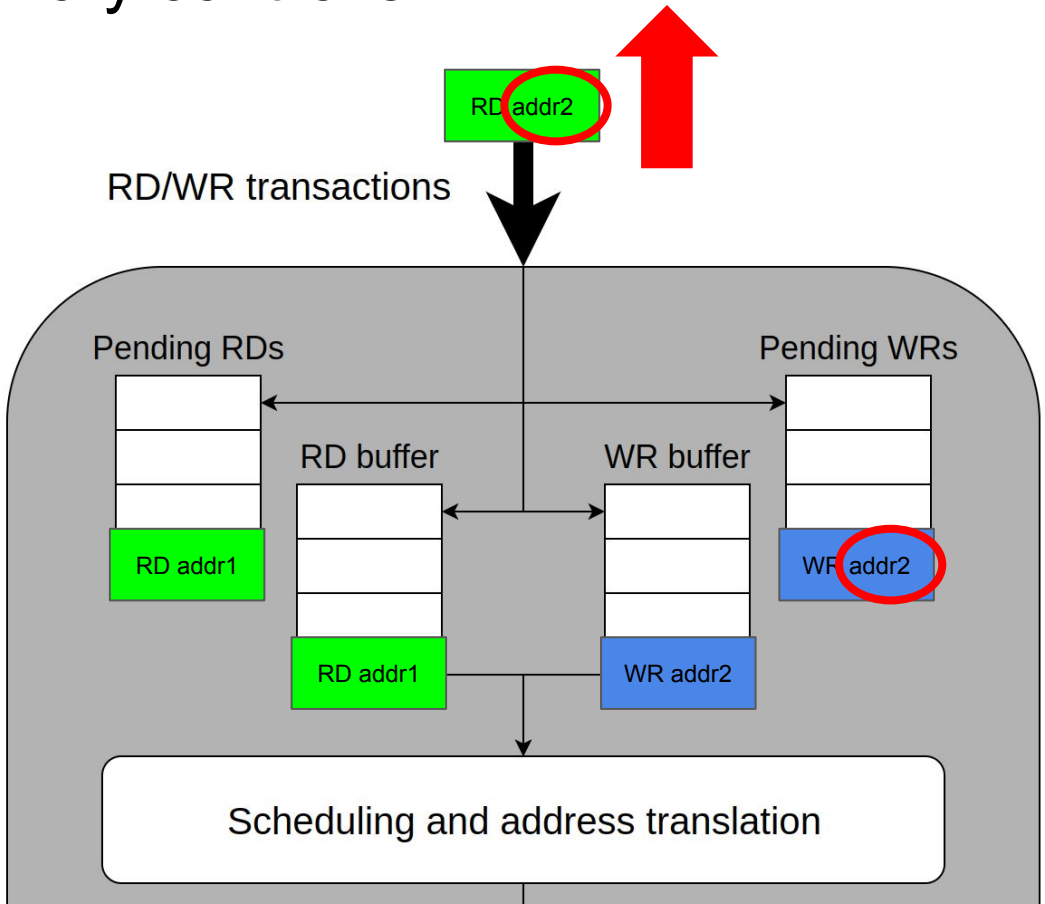
- The Pending RDs/WRs queues hold information on **<u>transactions that have not finished yet</u>**
  - They keep a “state” of sorts, ensuring that no incoming transactions are repeated or to send back RD operations for which there is a write pending



Structure (logical) of a memory controller

Let's start with transaction management:

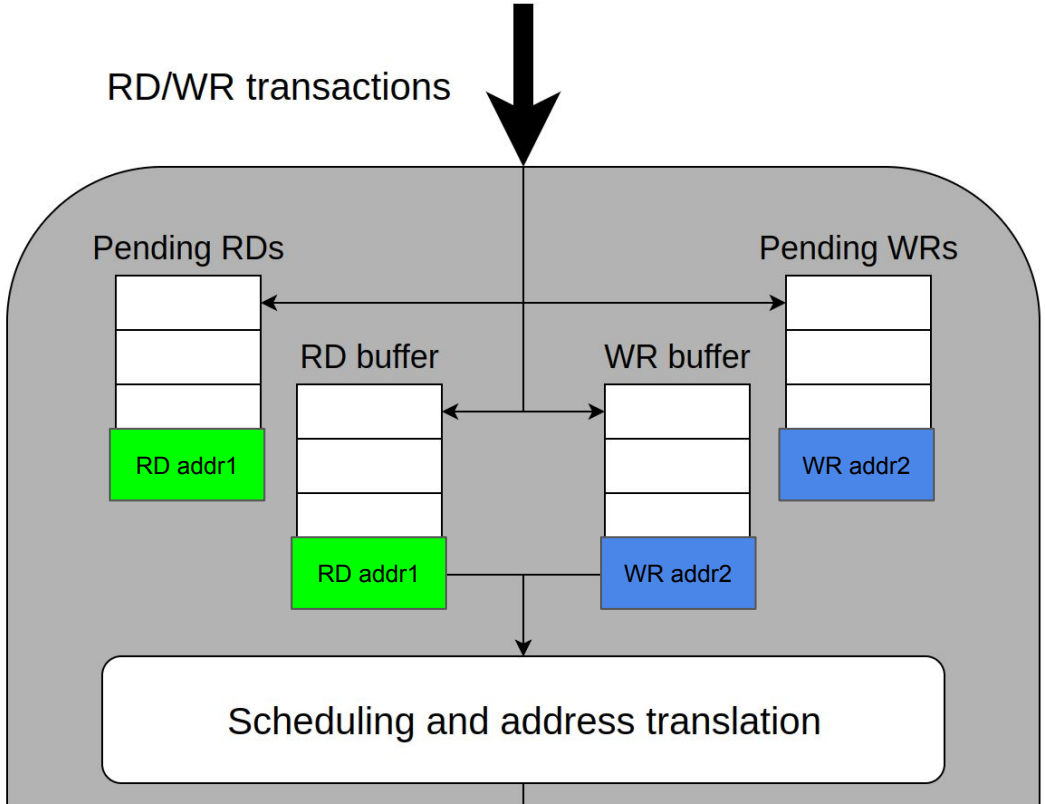
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
  - They keep a "state" of sorts, ensuring that no incoming transactions are repeated or to send back RD operations for which there is a write pending



Structure (logical) of a memory controller

Let's start with transaction management:

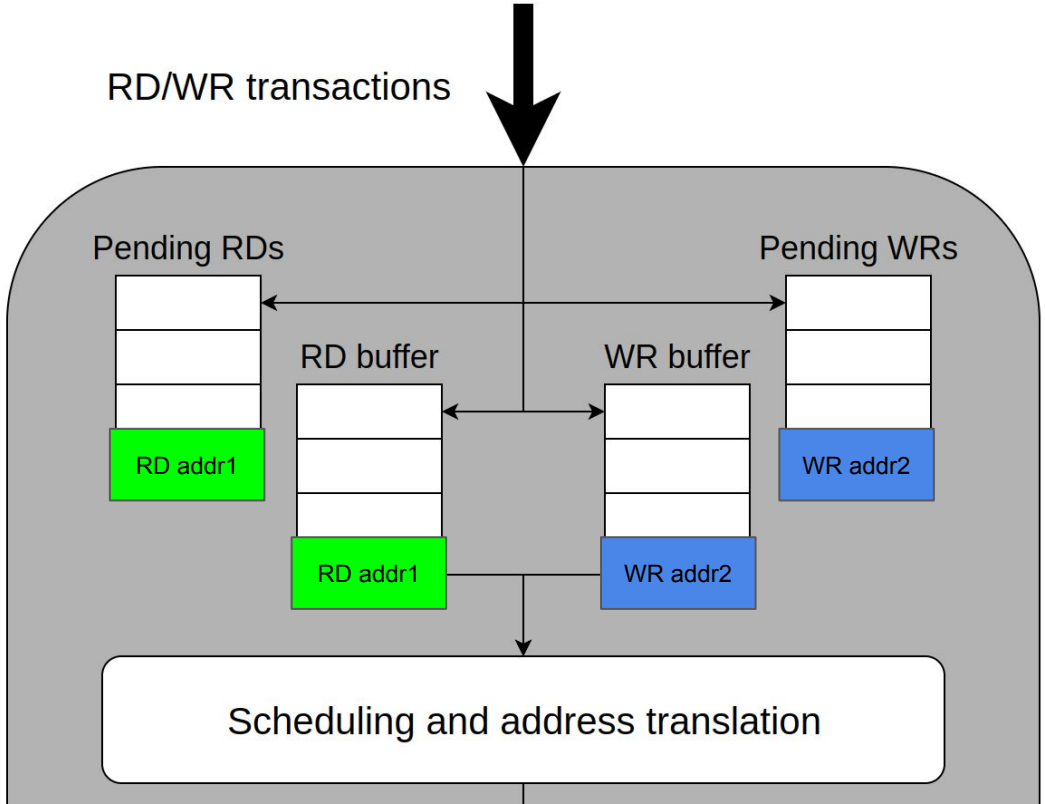
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
  - They keep a "state" of sorts, ensuring that no incoming transactions are repeated or to send back RD operations for which there is a write pending
- RD and WR buffers hold the **transactions to be scheduled** to DRAM



Structure (logical) of a memory controller

Let's start with transaction management:

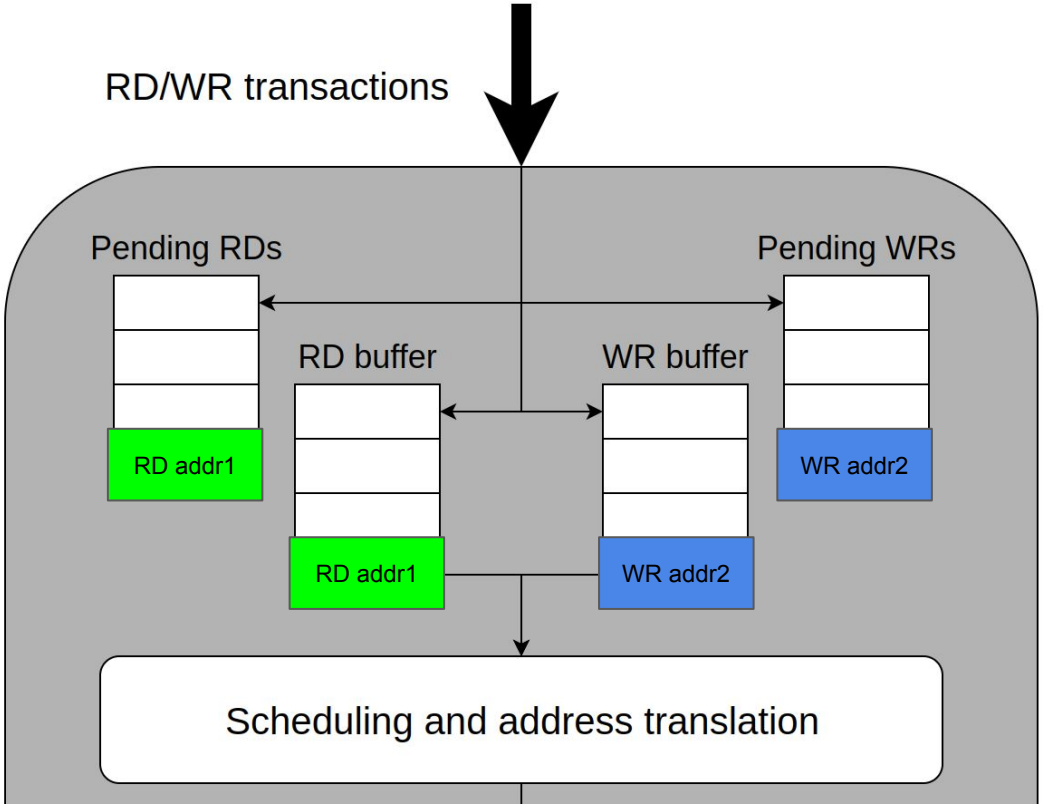
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
  - They keep a "state" of sorts, ensuring that no incoming transactions are repeated or to send back RD operations for which there is a write pending
- RD and WR buffers hold the **transactions to be scheduled** to DRAM
  - If their max capacity is reached, no more transactions are accepted (buffers control the incoming flow of transactions, not the Pending queues…)



Structure (logical) of a memory controller

Let's start with transaction management:

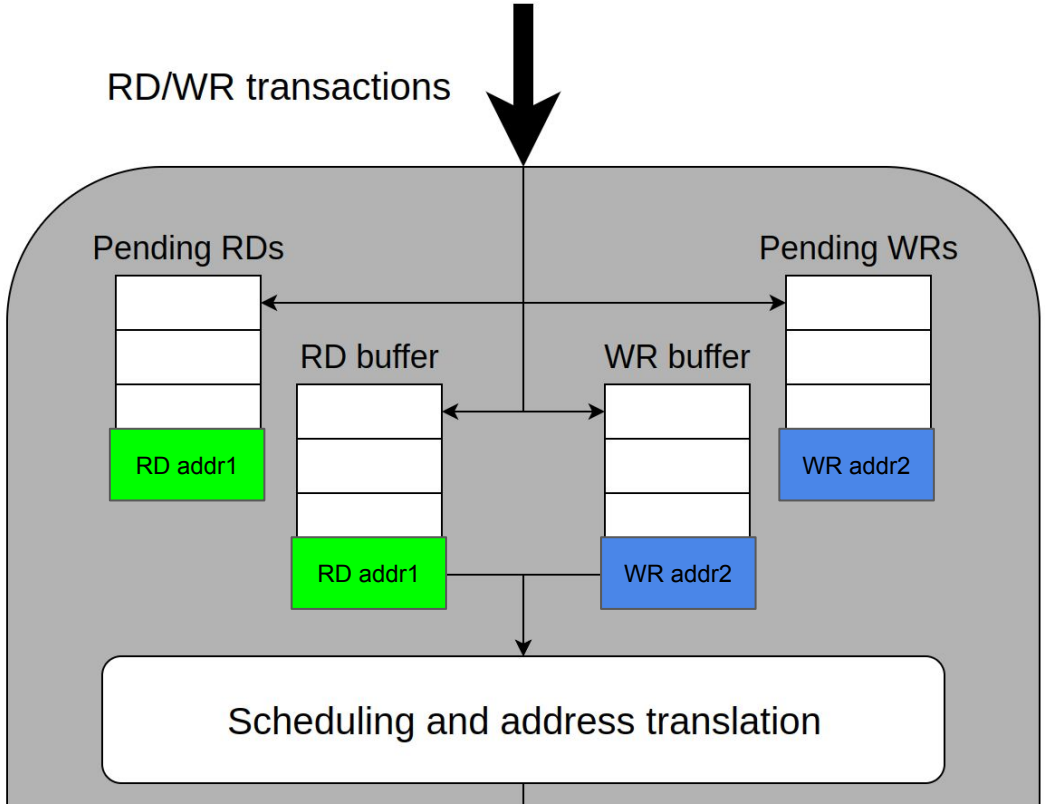
- The Pending RDs/WRs queues hold information on **transactions that have not finished yet**
  - They keep a "state" of sorts, ensuring that no incoming transactions are repeated or to send back RD operations for which there is a write pending
- RD and WR buffers hold the **transactions to be scheduled** to DRAM
  - If their max capacity is reached, no more transactions are accepted (buffers control the incoming flow of transactions, not the Pending queues…)
  - Transactions are removed from the buffers when they are scheduled



Structure (logical) of a memory controller

Let's start with transaction management:

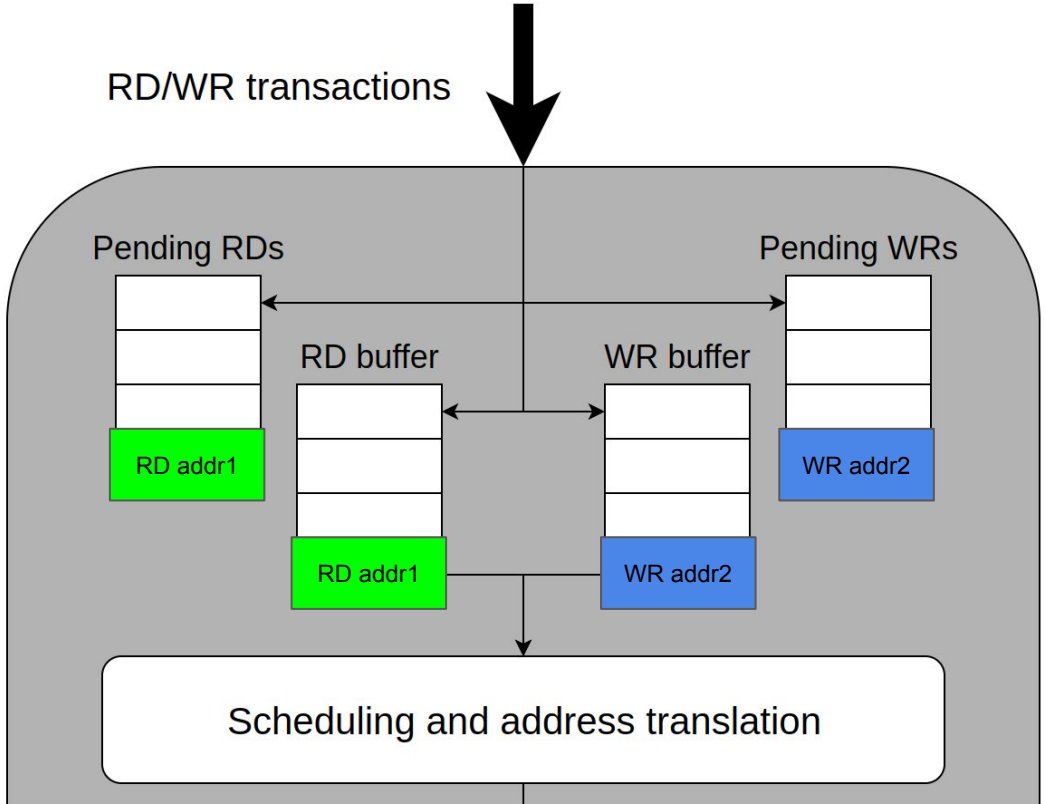
- …ok, but, what is *"scheduling a transaction"*?



Structure (logical) of a memory controller

Let's start with transaction management:

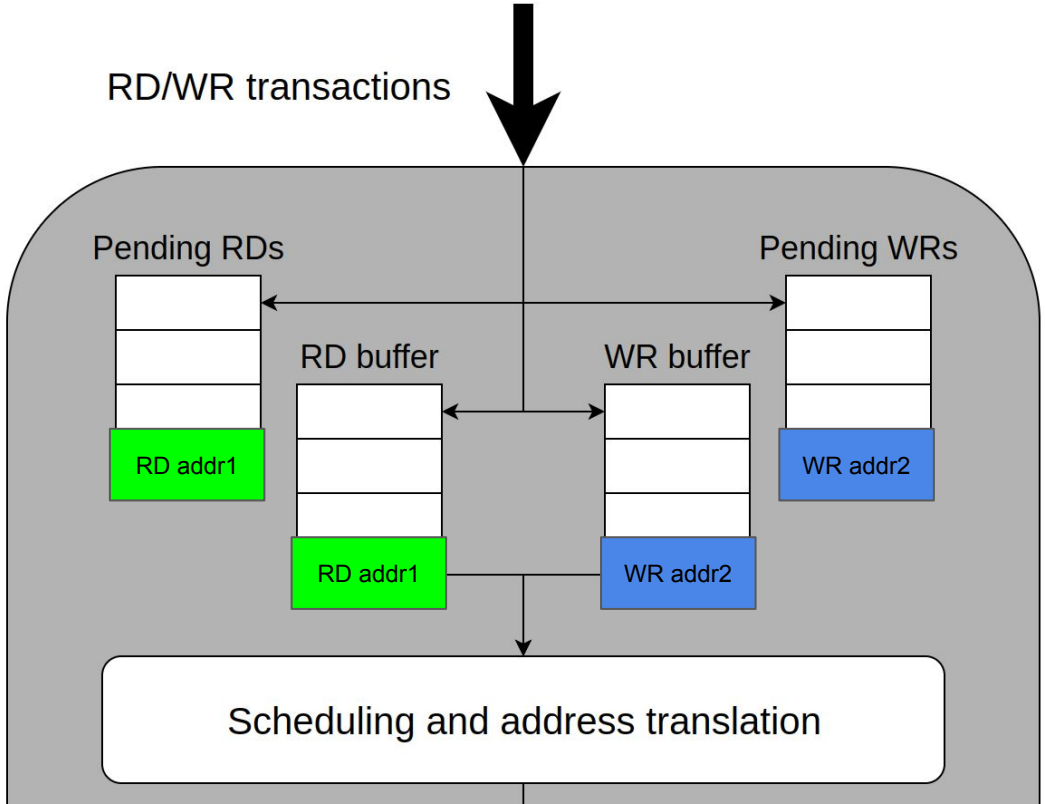
- …ok, but, what is *"scheduling a transaction"*?
- Scheduling: **moving R/W transactions** from their buffers **to bank schedulers** for DRAM command generation



Structure (logical) of a memory controller

Let's start with transaction management:

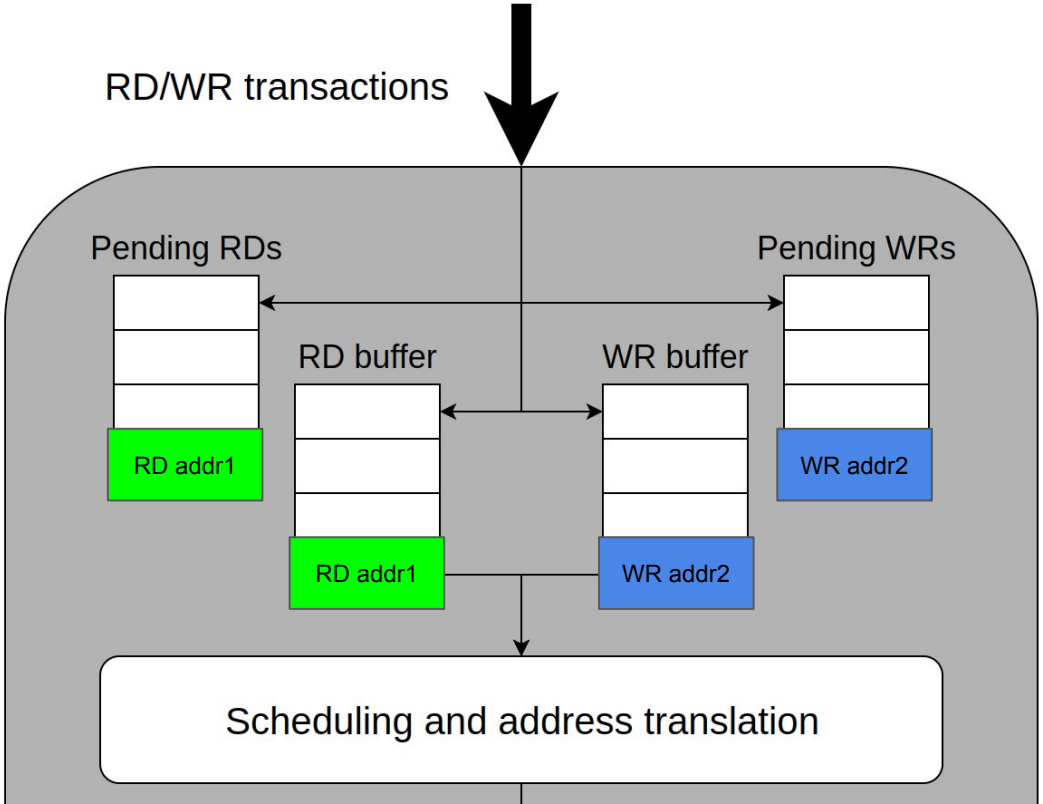
- …ok, but, what is *"scheduling a transaction"*?
- Scheduling: **moving R/W transactions** from their buffers **to bank schedulers** for DRAM command generation
  - So, what's so special about scheduling?



Structure (logical) of a memory controller

Let's start with transaction management:

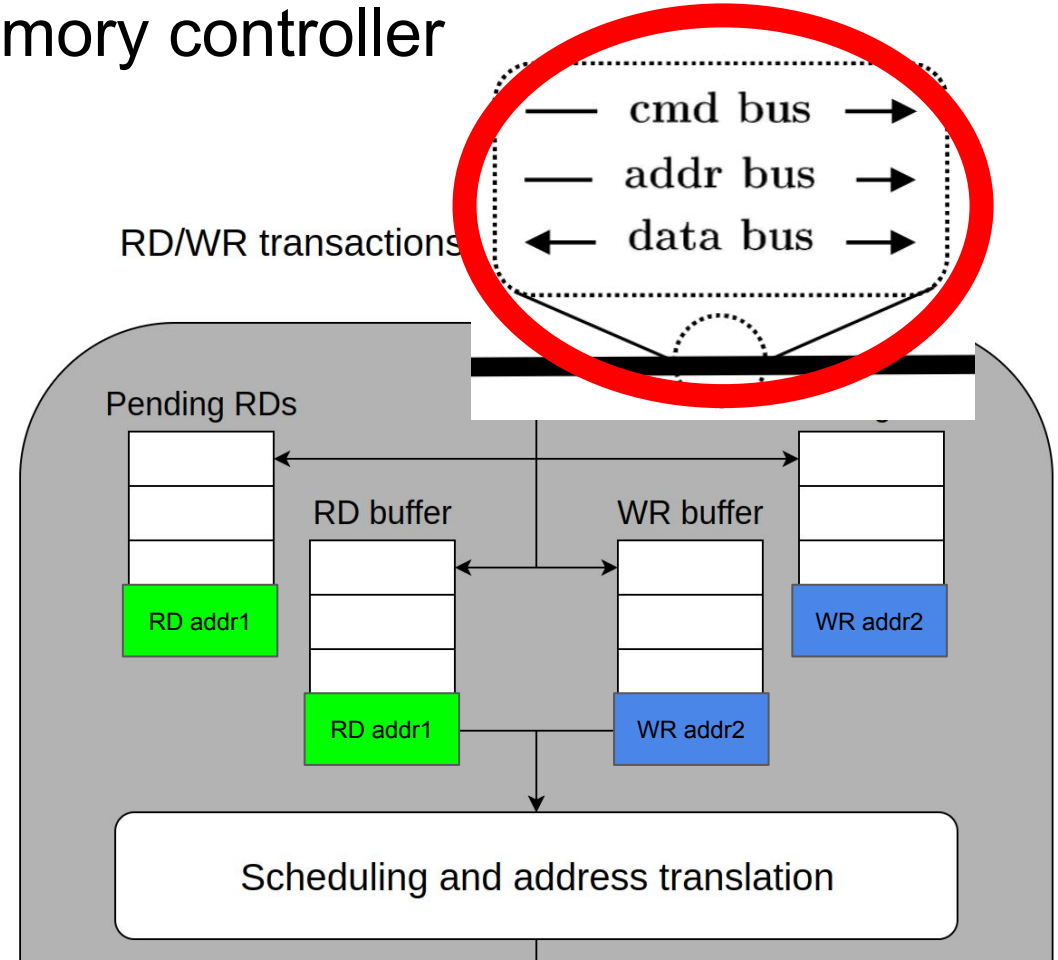
- …ok, but, what is *"scheduling a transaction"*?
- Scheduling: **moving R/W transactions** from their buffers **to bank schedulers** for DRAM command generation
  - So, what's so special about scheduling?
- Scheduling governs the **logic of which buffer to schedule transactions out of**, either the RD buffer or the WR buffer



Structure (logical) of a memory controller

Let's start with transaction management:

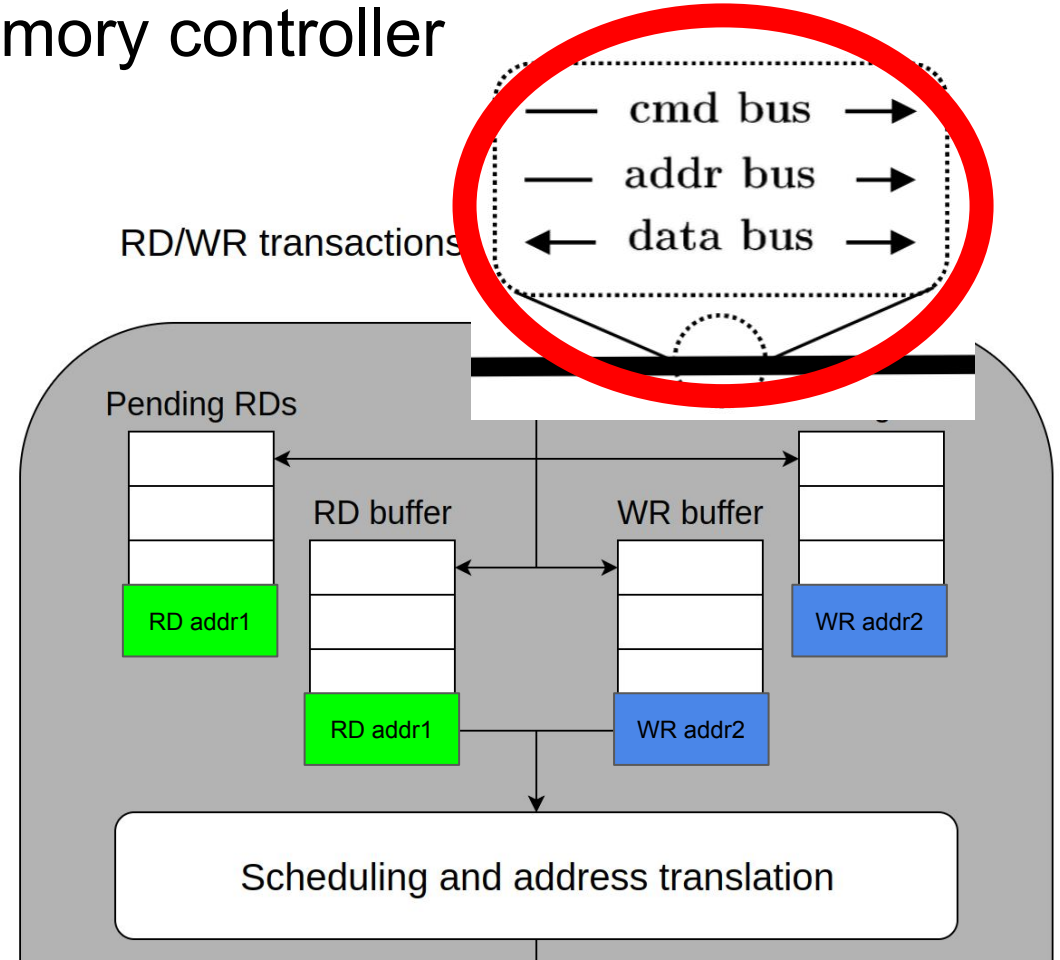
- …ok, but, what is *"scheduling a transaction"*?
- Scheduling: **moving R/W transactions** from their buffers **to bank schedulers** for DRAM command generation
  - So, what's so special about scheduling?
- Scheduling governs the **logic of which buffer to schedule transactions out of**, either the RD buffer or the WR buffer
  - Remember the three buses? Well, to minimize the latency effect of switching the direction of the data bus for reading or writing, write transactions are scheduled in batches, thus keeping them separate from reads



Structure (logical) of a memory controller

Let's start with transaction management:

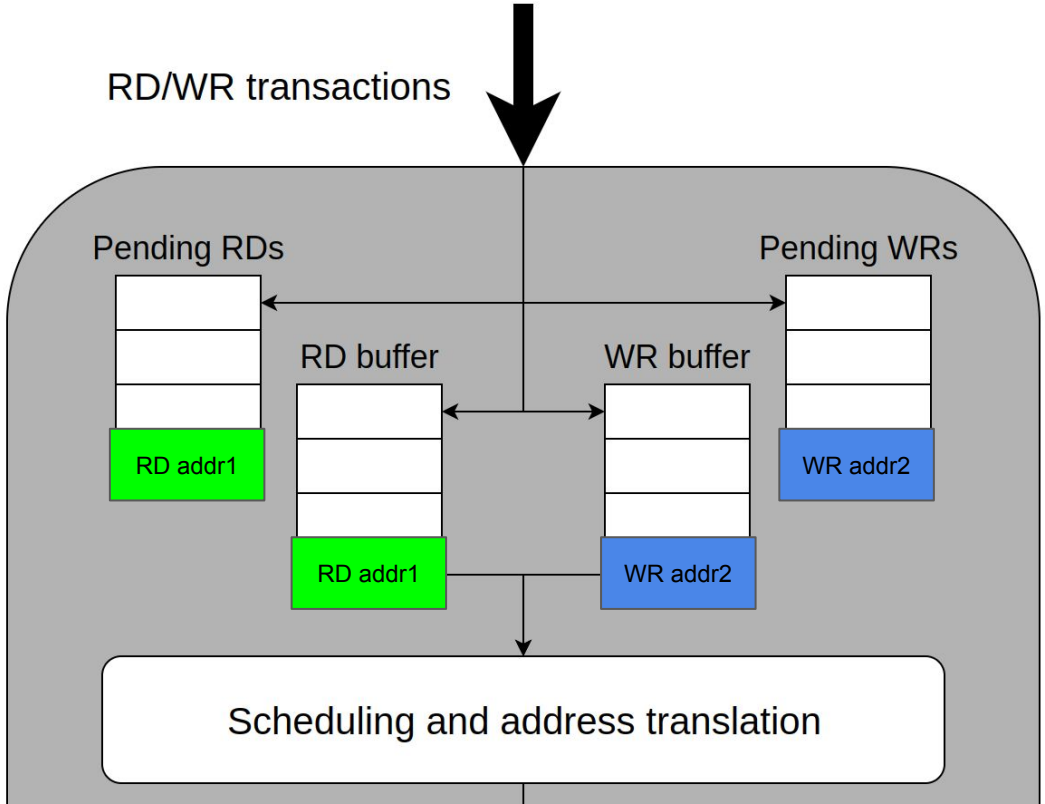
- …ok, but, what is *"scheduling a transaction"*?
- Scheduling: **moving R/W transactions** from their buffers **to bank schedulers** for DRAM command generation
  - So, what's so special about scheduling?
- Scheduling governs the **logic of which buffer to schedule transactions out of**, either the RD buffer or the WR buffer
  - Remember the three buses? Well, to minimize the latency effect of switching the direction of the data bus for reading or writing, write transactions are scheduled in batches, thus keeping them separate from reads
  - This is known as a write batching policy



Structure (logical) of a memory controller

Let's start with transaction management:

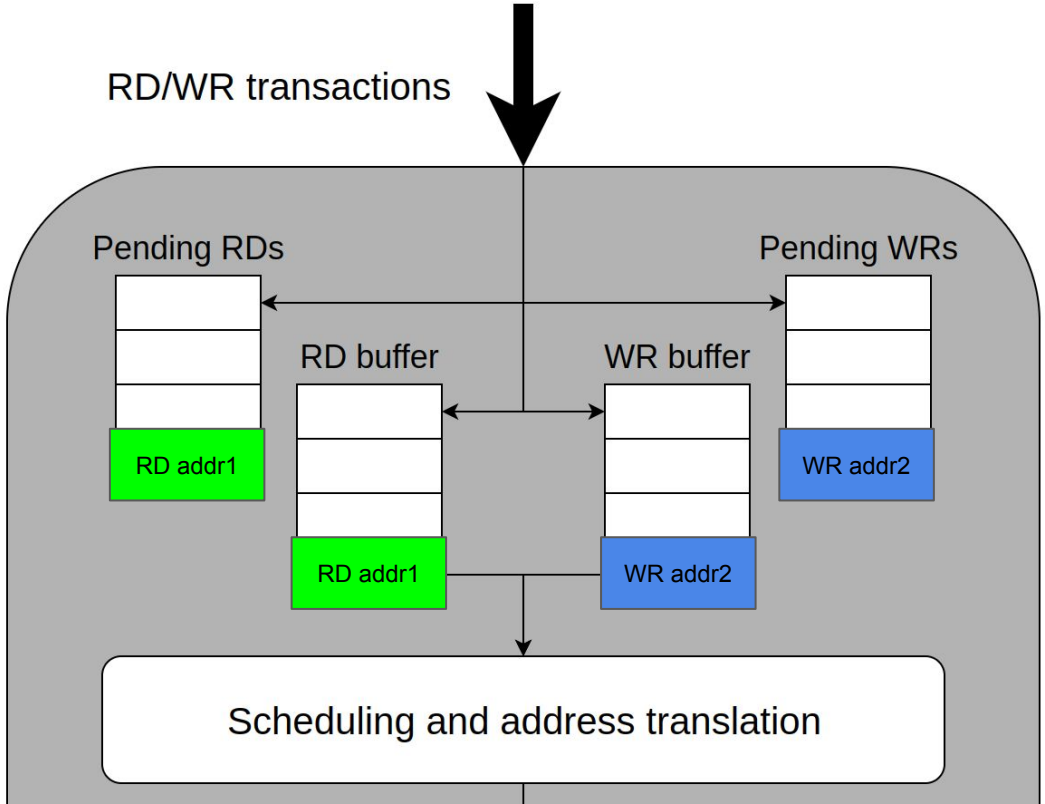
- …
- (from some slides ago) *"Transactions are removed from the buffers when they are scheduled"*



Structure (logical) of a memory controller

Let's start with transaction management:

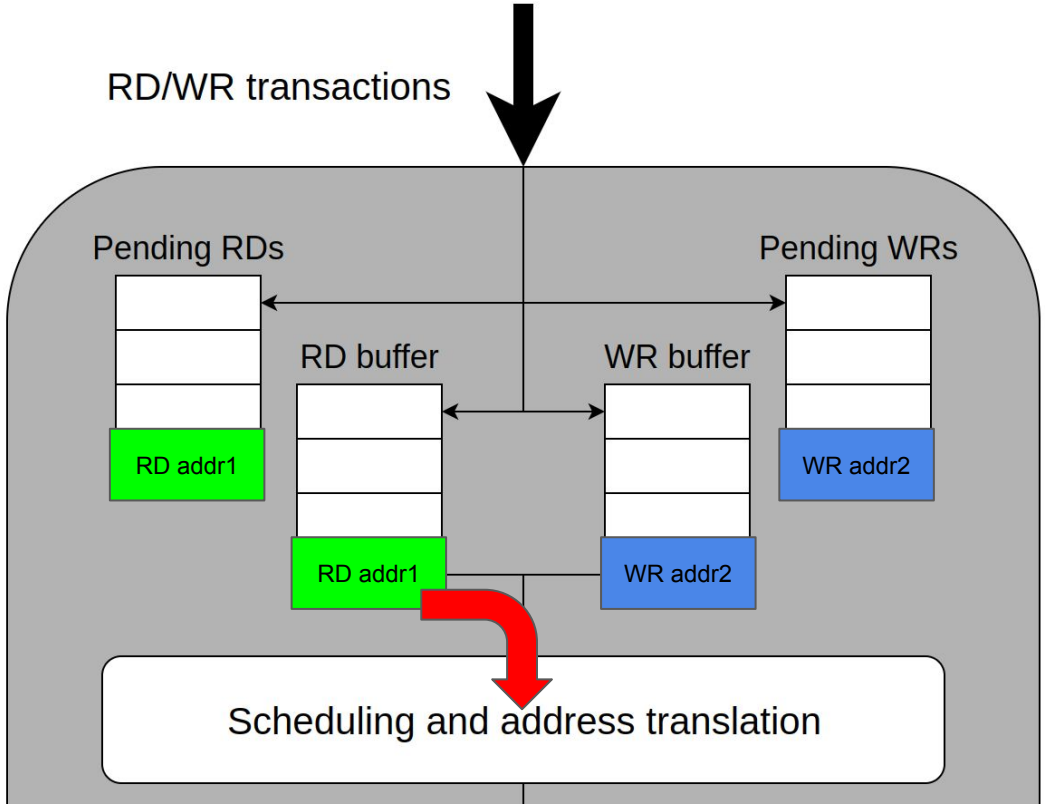
- …
- (from some slides ago) *"Transactions are removed from the buffers when they are scheduled"*
  - If the write batching policy has not been activated, transactions in the RD buffer are continuously being scheduled



Structure (logical) of a memory controller

Let's start with transaction management:

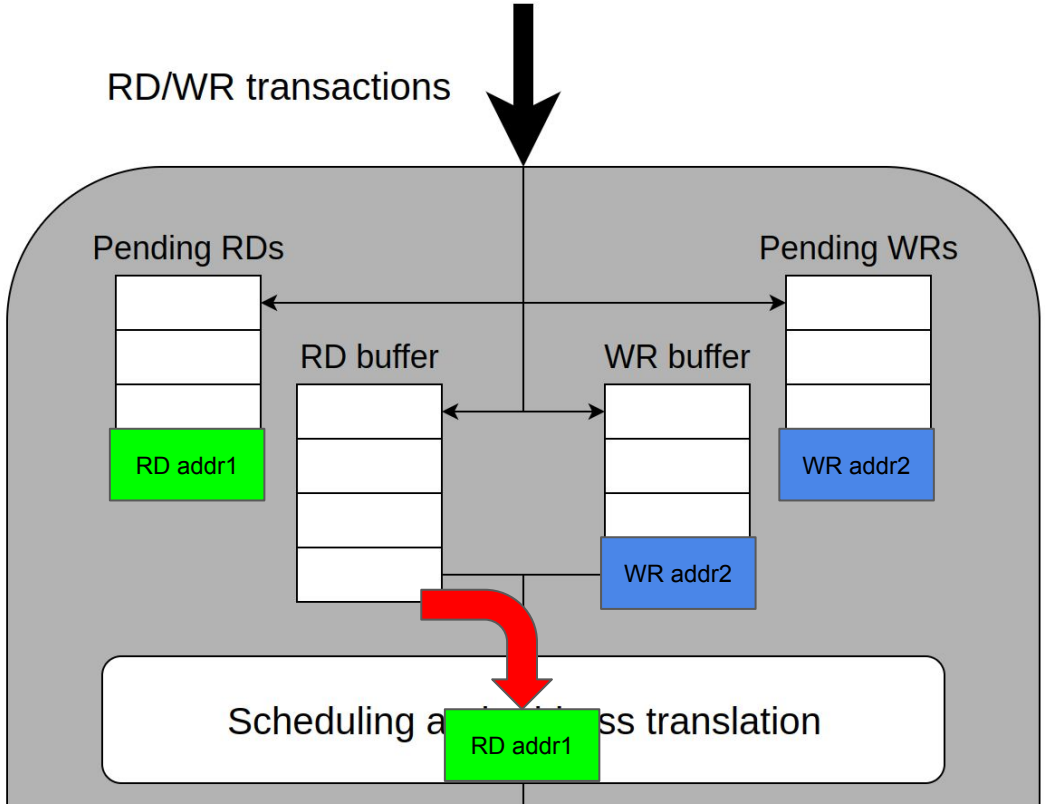
- …
- (from some slides ago) *"Transactions are removed from the buffers when they are scheduled"*
  - If the write batching policy has not been activated, transactions in the RD buffer are continuously being scheduled



Structure (logical) of a memory controller

Let's start with transaction management:

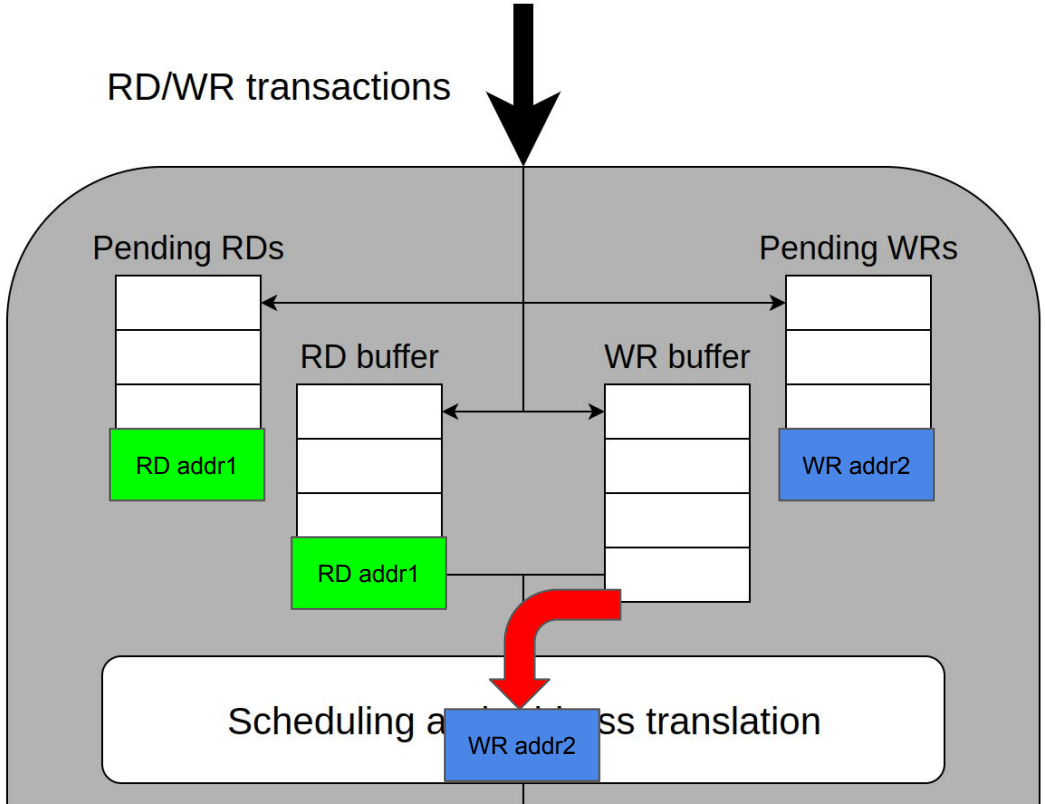
- …
- (from some slides ago) *"Transactions are removed from the buffers when they are scheduled"*
  - If the write batching policy has not been activated, transactions in the RD buffer are continuously being scheduled



Structure (logical) of a memory controller

Let's start with transaction management:

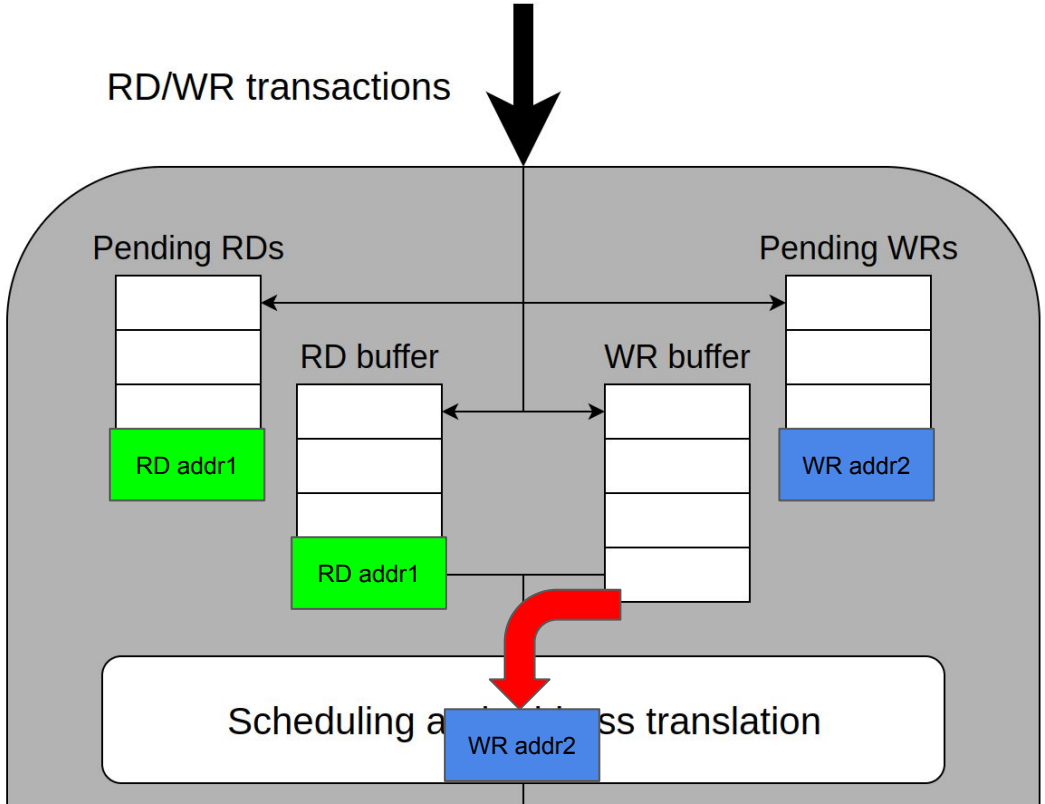
- …
- (from some slides ago) *"Transactions are removed from the buffers when they are scheduled"*
  - If the write batching policy has not been activated, transactions in the RD buffer are continuously being scheduled
  - If write batching is activated, write transactions start being scheduled until all the WR transactions with which write batching are activated are removed from the WR buffer



Structure (logical) of a memory controller

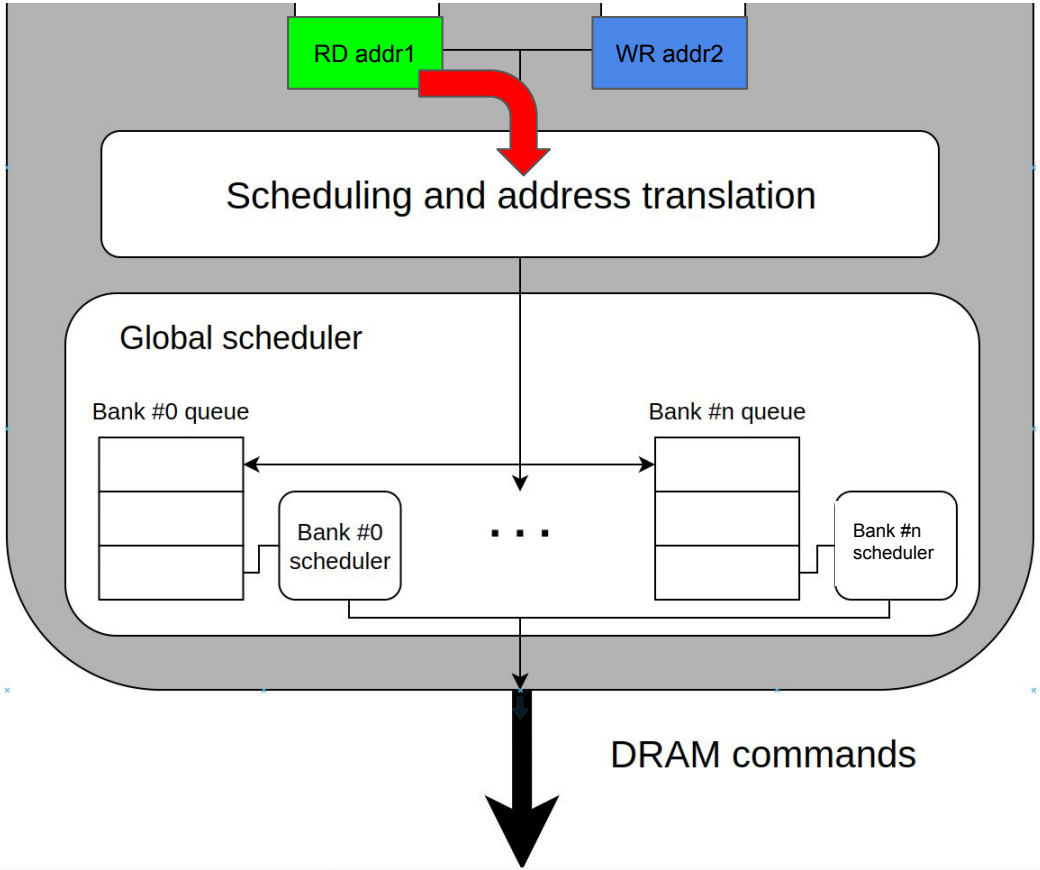
Let's start with transaction management:

- …
- (from some slides ago) *"Transactions are removed from the buffers when they are scheduled"*
  - If the write batching policy has not been activated, transactions in the RD buffer are continuously being scheduled
  - If write batching is activated, write transactions start being scheduled until all the WR transactions with which write batching are activated are removed from the WR buffer
- **REMINDER:** transactions are kept at the Pending queues to keep a record of what is being processed in memory at the moment…



Structure (logical) of a memory controller

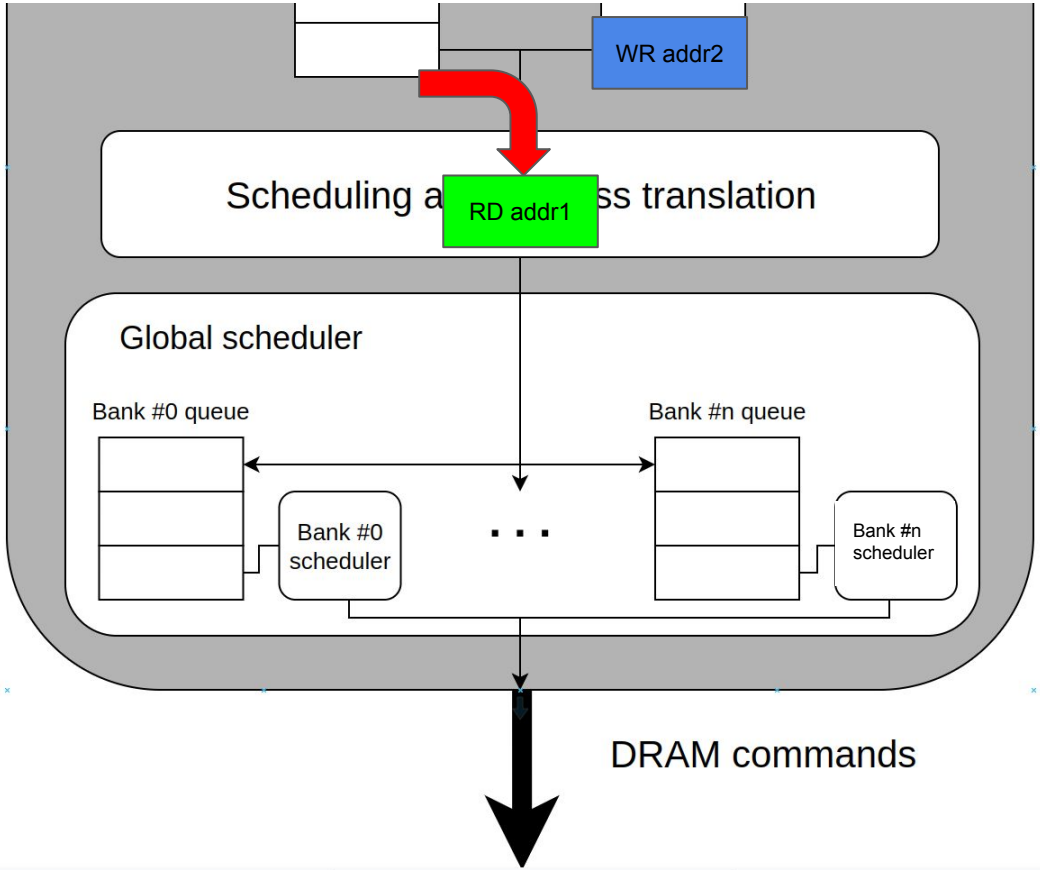
Let’s go down to DRAM command generation:



Structure (logical) of a memory controller

Let’s go down to DRAM command generation:

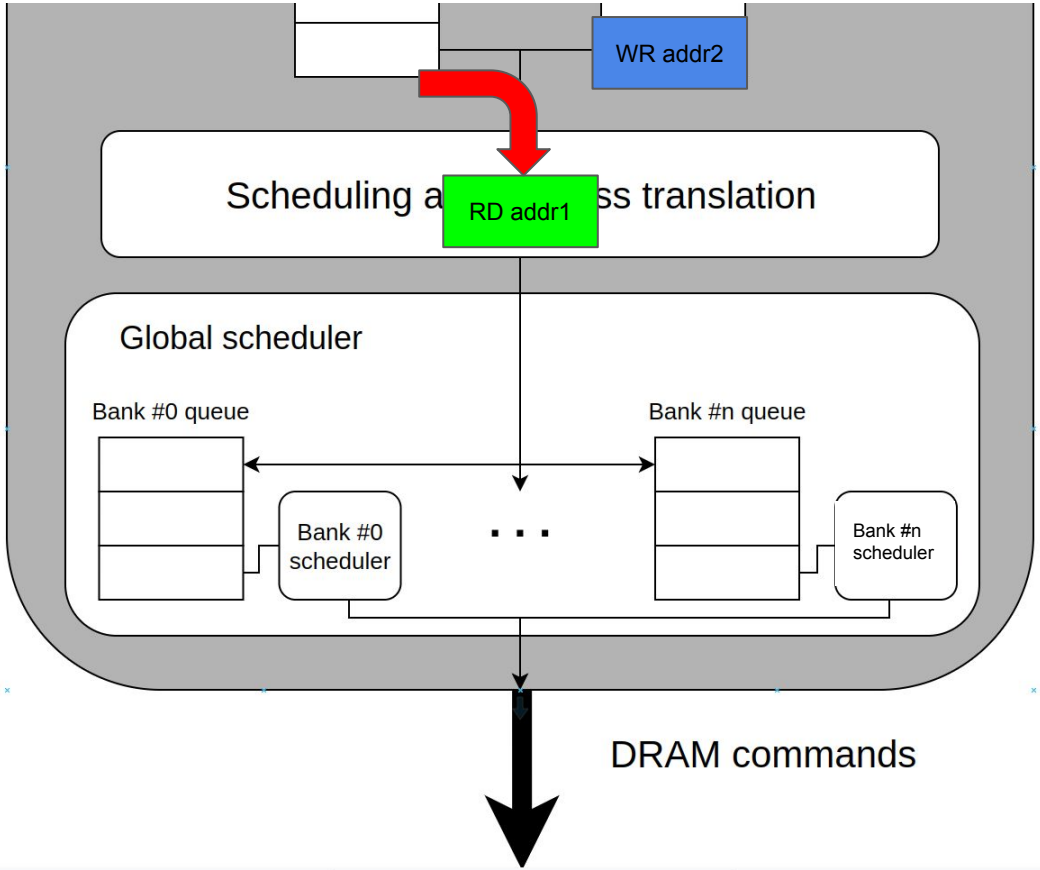
- Scheduling moves stuff from top to bottom, that much is clear



Structure (logical) of a memory controller

Let's go down to DRAM command generation:

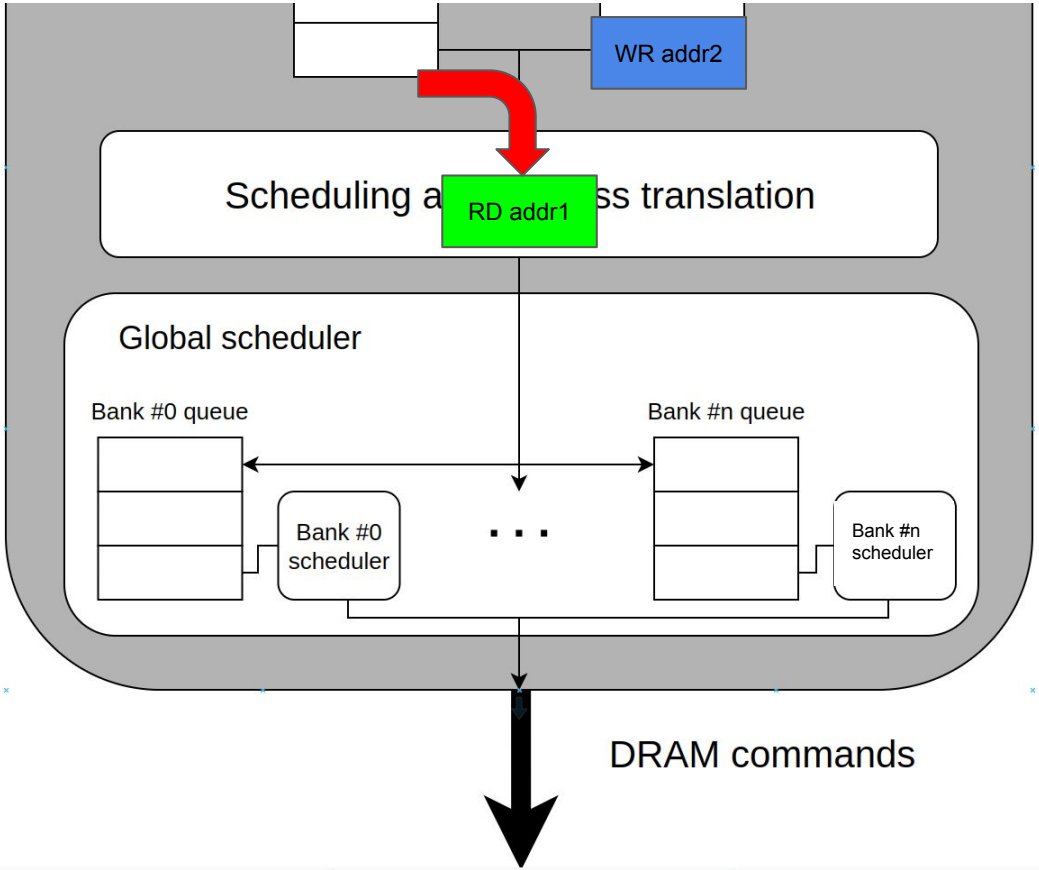
- Scheduling moves stuff from top to bottom, that much is clear
- Address translation **places transactions into bank queues** according to their **target address**



Structure (logical) of a memory controller

Let's go down to DRAM command generation:

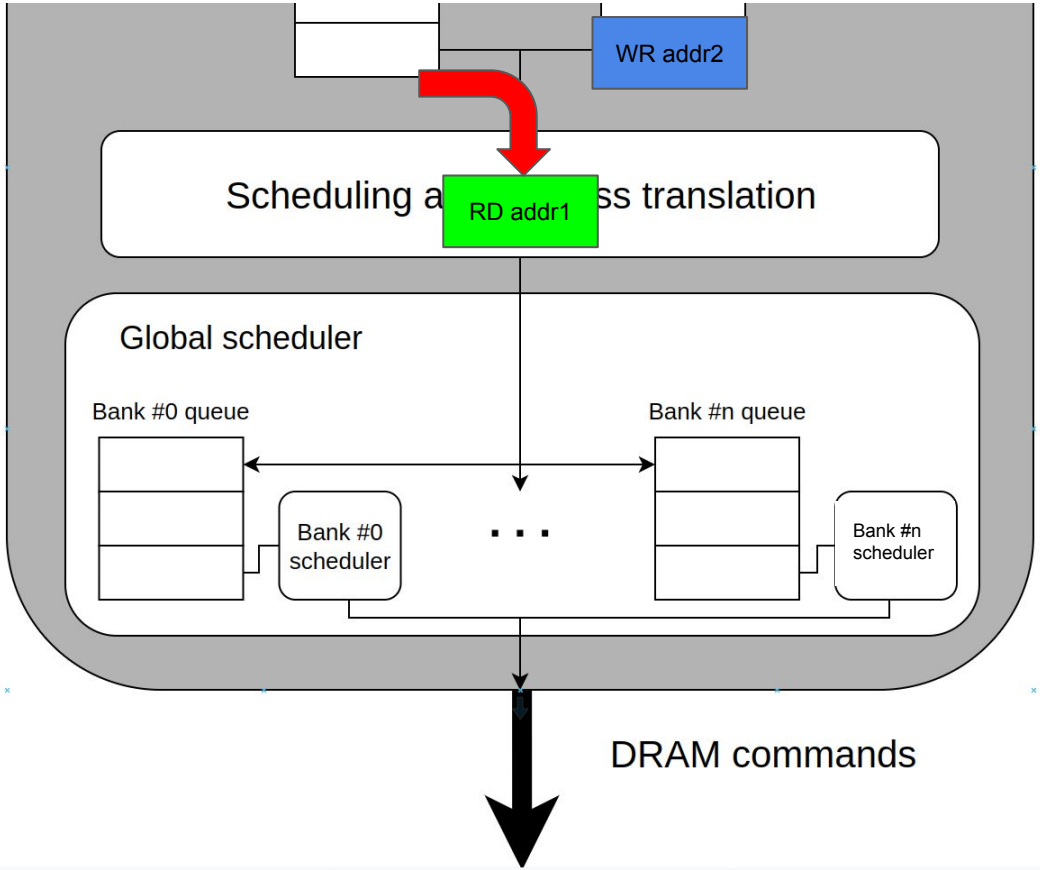
- Scheduling moves stuff from top to bottom, that much is clear
- Address translation **places transactions into bank queues** according to their **target address**
  - Addresses are divided into segments defining the channel, rank, bank, row and column that the transaction is accessing



Structure (logical) of a memory controller

Let's go down to DRAM command generation:

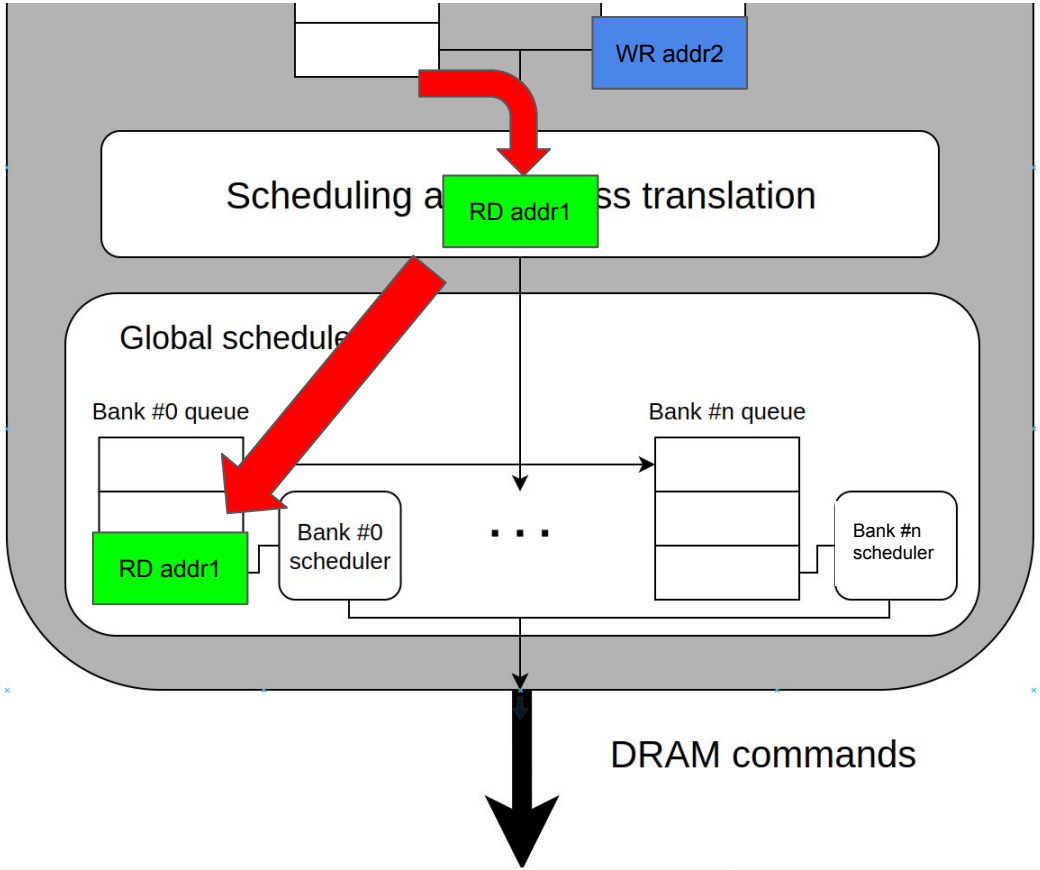
- Scheduling moves stuff from top to bottom, that much is clear
- Address translation **places transactions into bank queues** according to their **target address**
  - Addresses are divided into segments defining the channel, rank, bank, row and column that the transaction is accessing
  - This is called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



Structure (logical) of a memory controller

Let's go down to DRAM command generation:

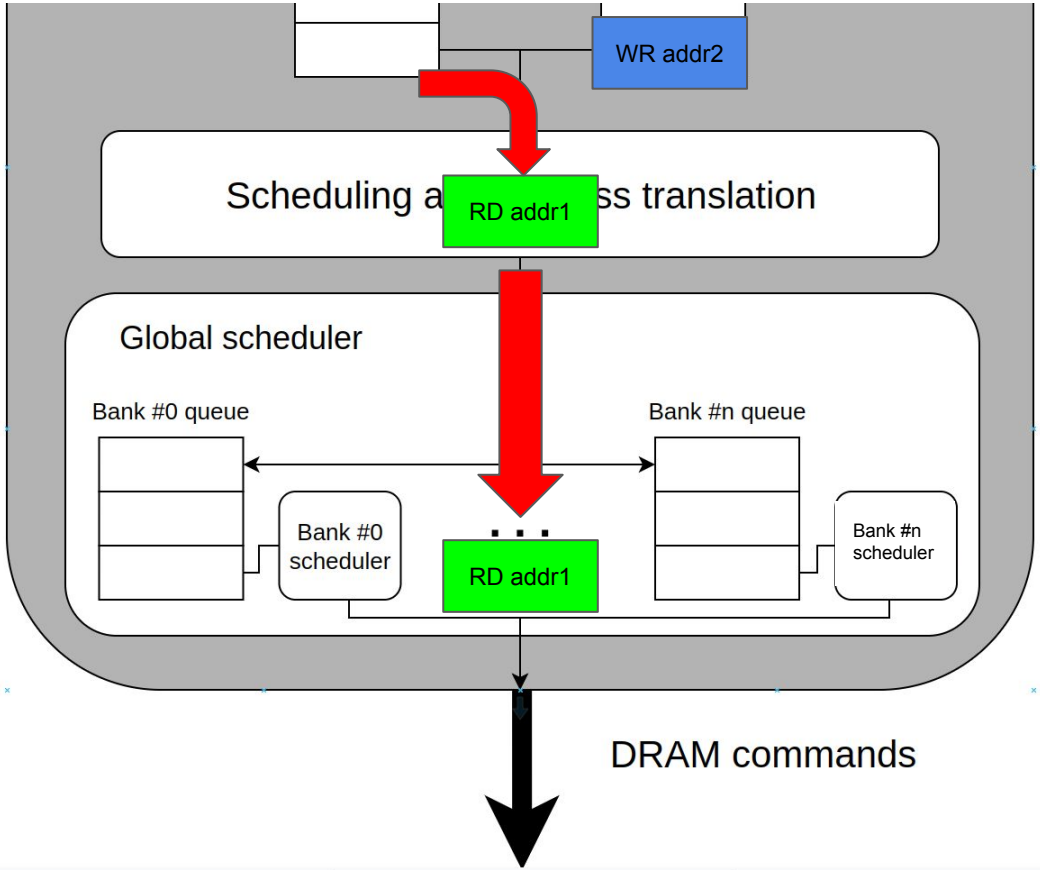
- Scheduling moves stuff from top to bottom, that much is clear
- Address translation **places transactions into bank queues** according to their **target address**
  - Addresses are divided into segments defining the channel, rank, bank, row and column that the transaction is accessing
  - This is called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



Structure (logical) of a memory controller

Let's go down to DRAM command generation:

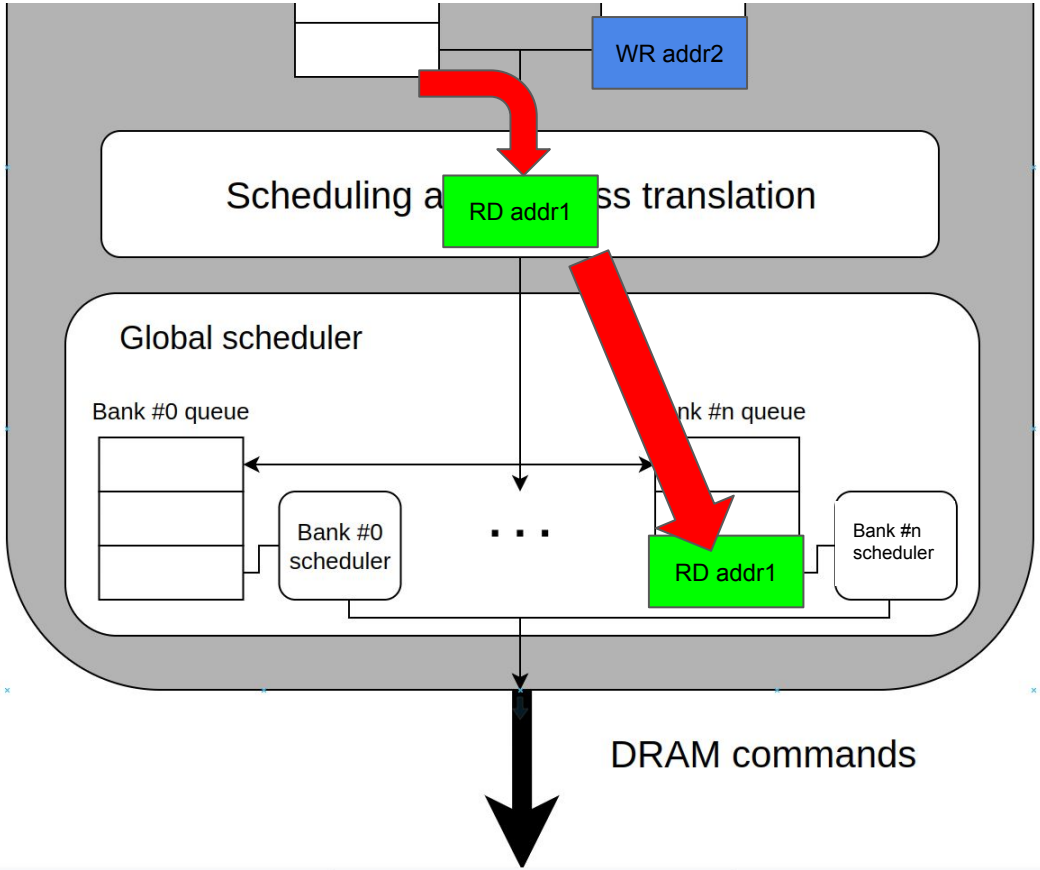
- Scheduling moves stuff from top to bottom, that much is clear
- Address translation **places transactions into bank queues** according to their **target address**
  - Addresses are divided into segments defining the channel, rank, bank, row and column that the transaction is accessing
  - This is called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



Structure (logical) of a memory controller

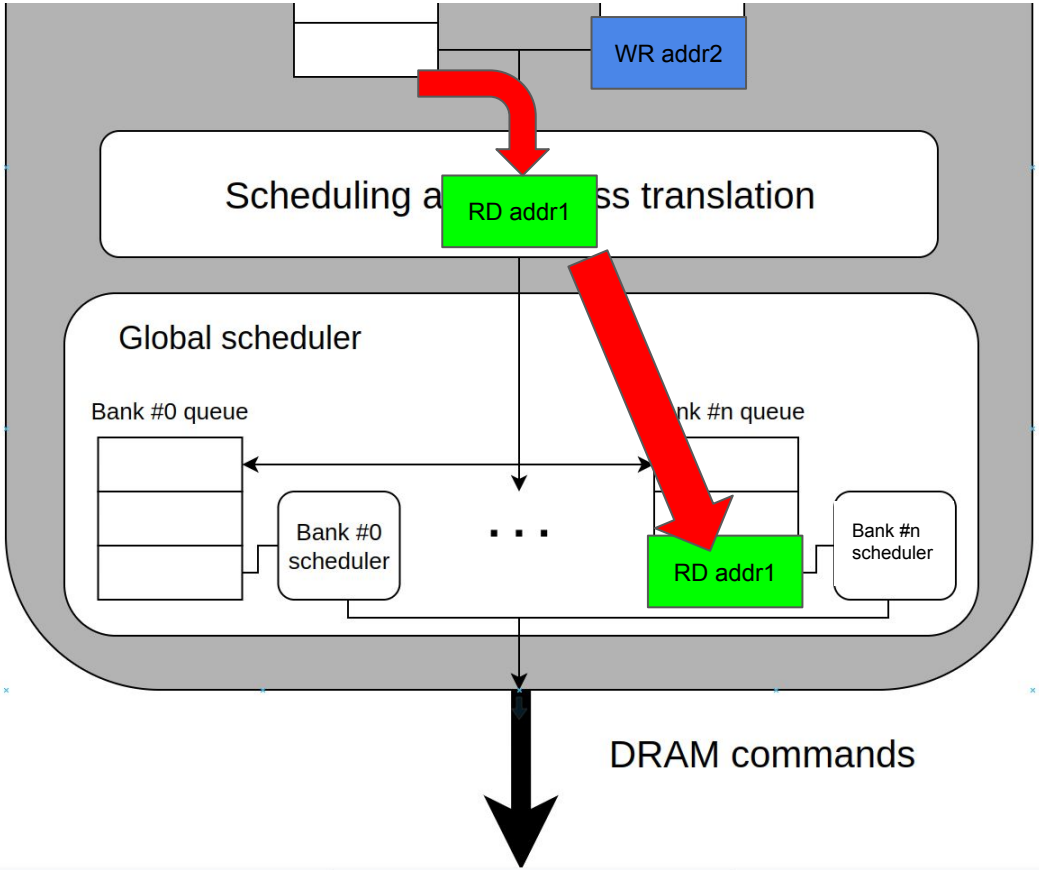
Let's go down to DRAM command generation:

- 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

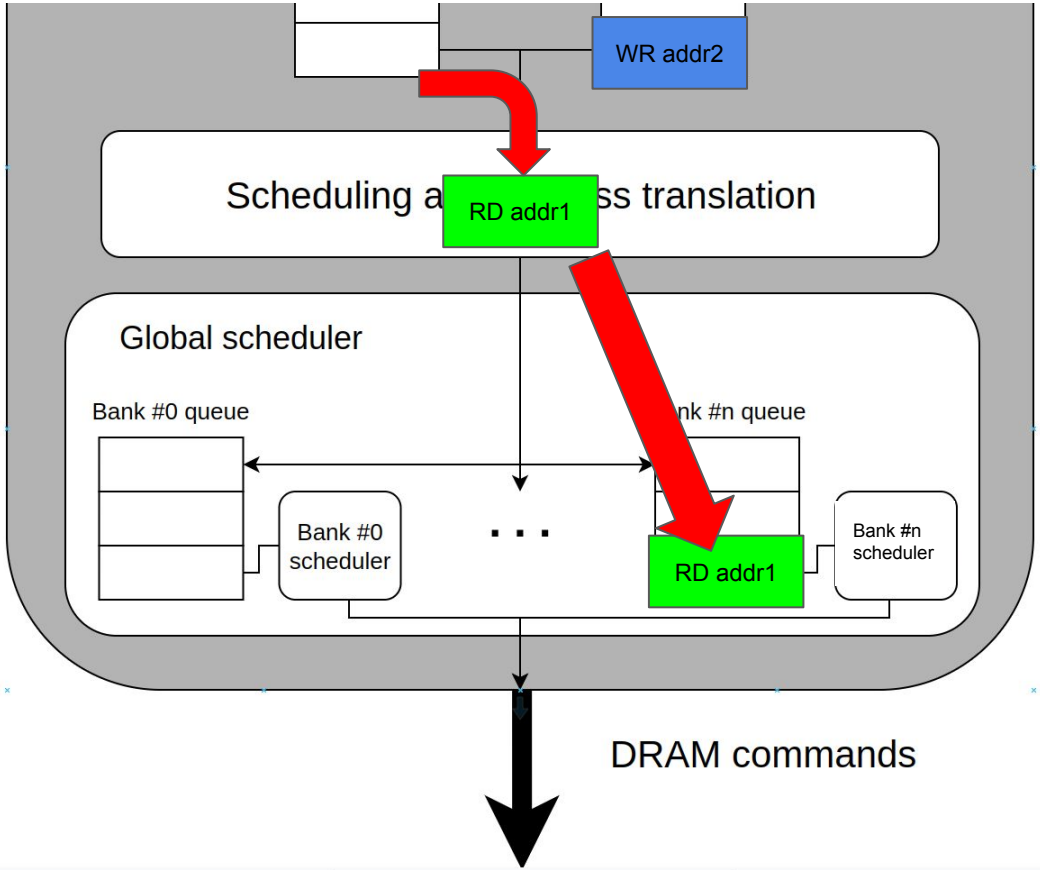
Where are the DRAM commands though??



Structure (logical) of a memory controller

Where are the DRAM commands though??

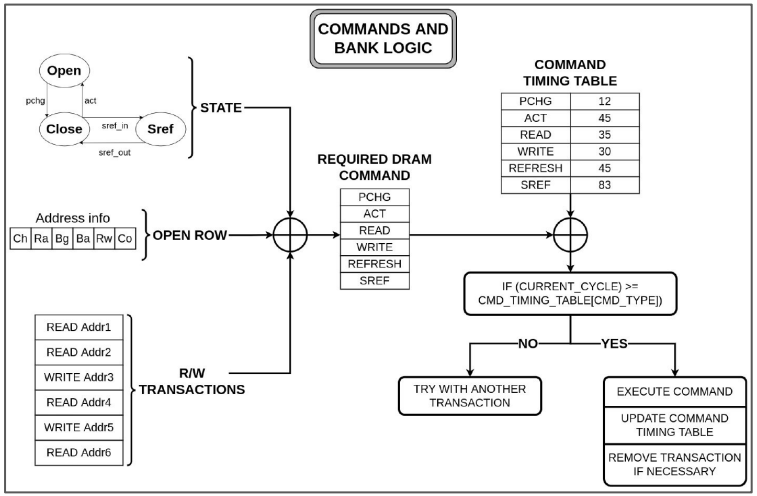
- This is where the bank schedulers come in…

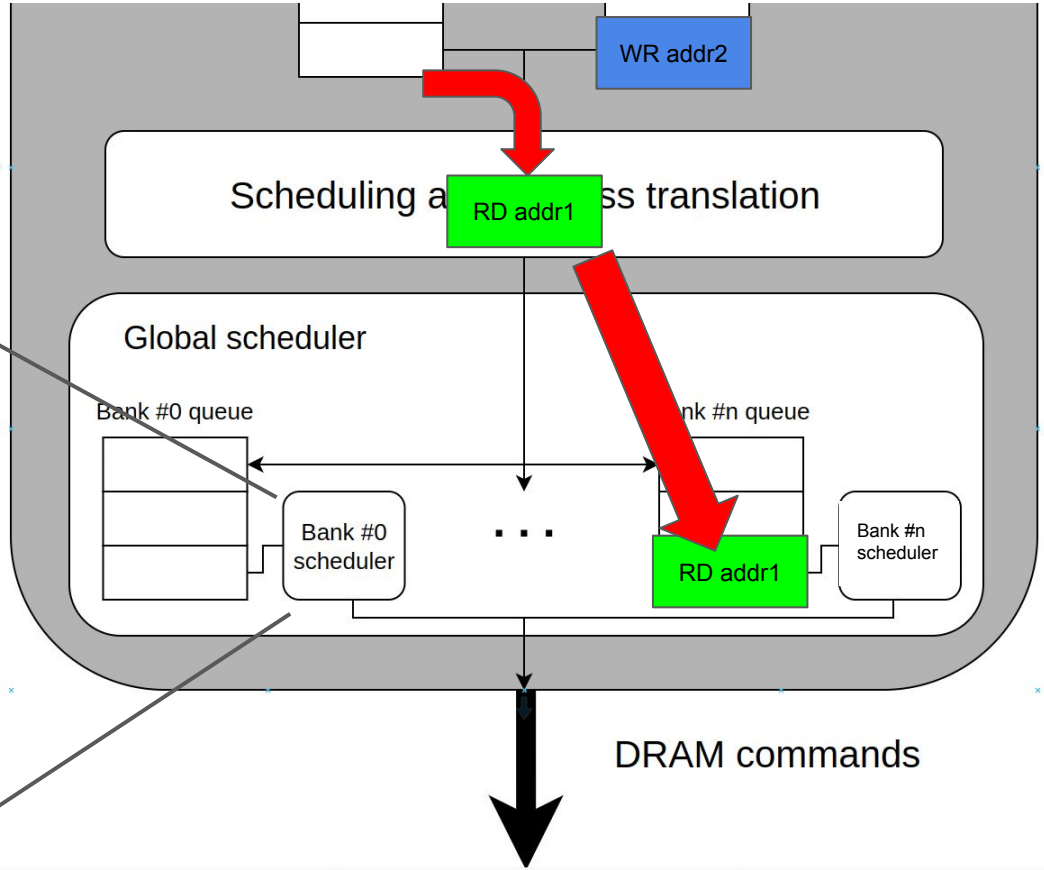


Structure (logical) of a memory controller

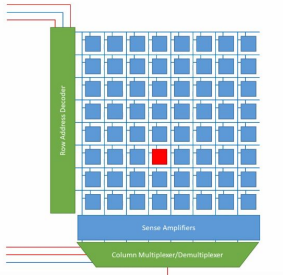
Where are the DRAM commands though??

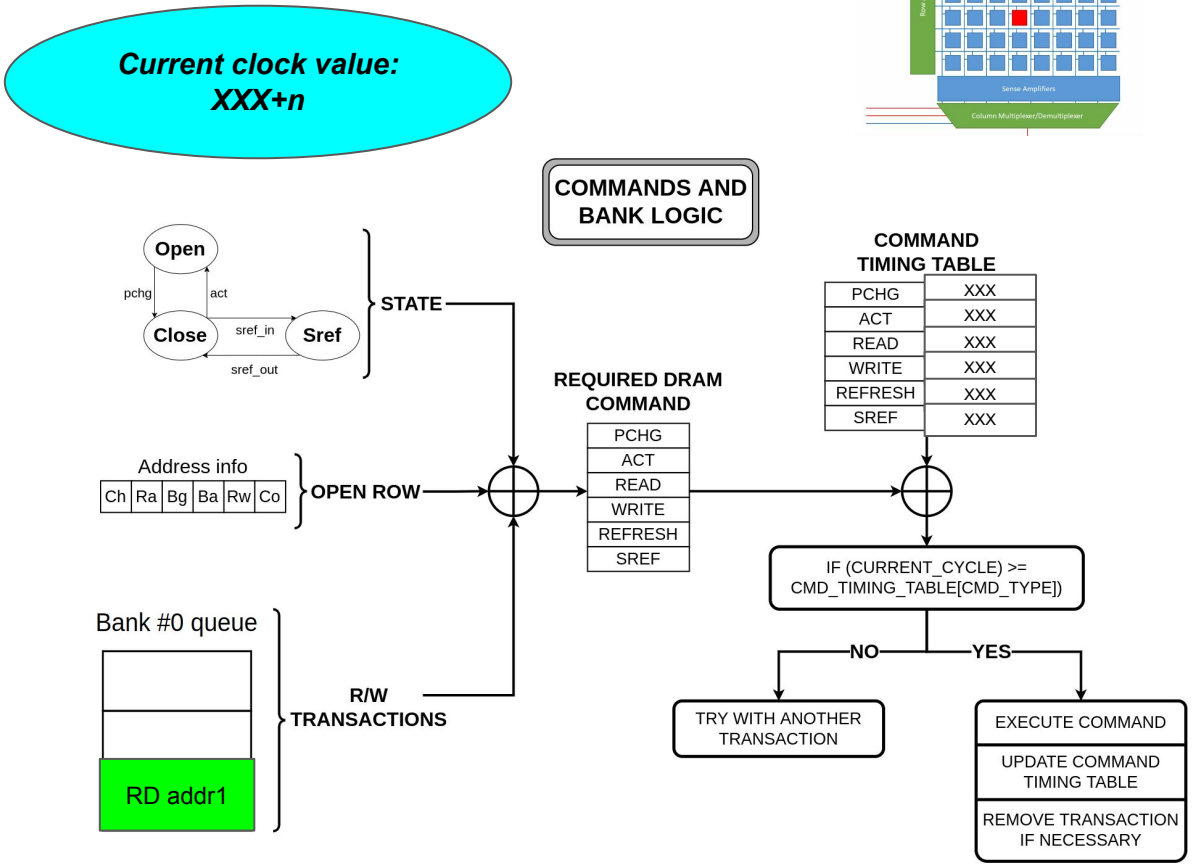
- This is where the bank schedulers come in…





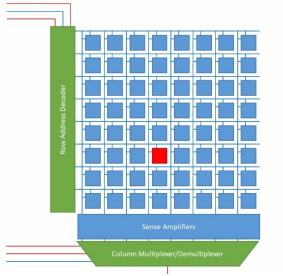
Structure (logical) of a memory controller

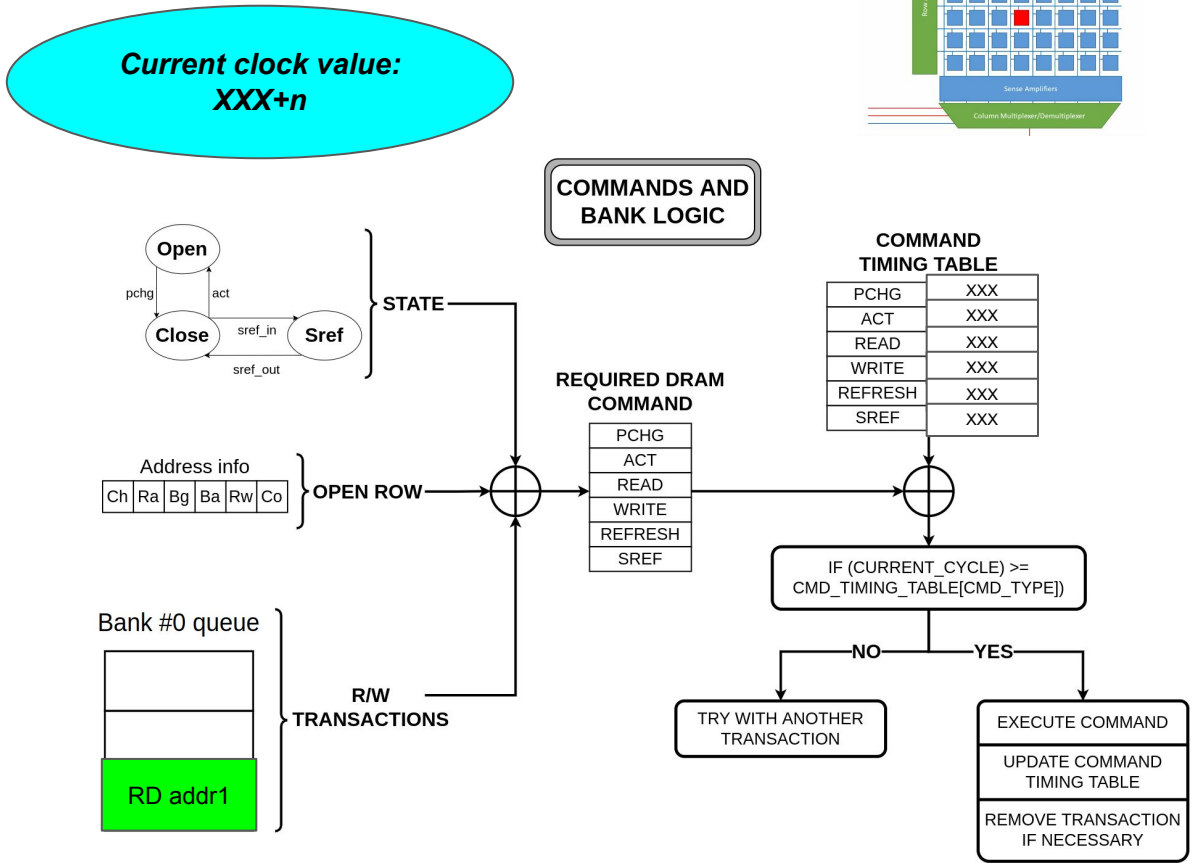




Structure (logical) of a memory controller

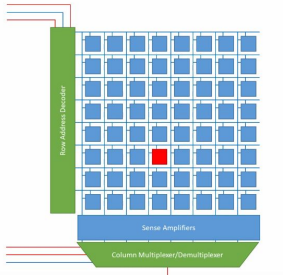
Let’s assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

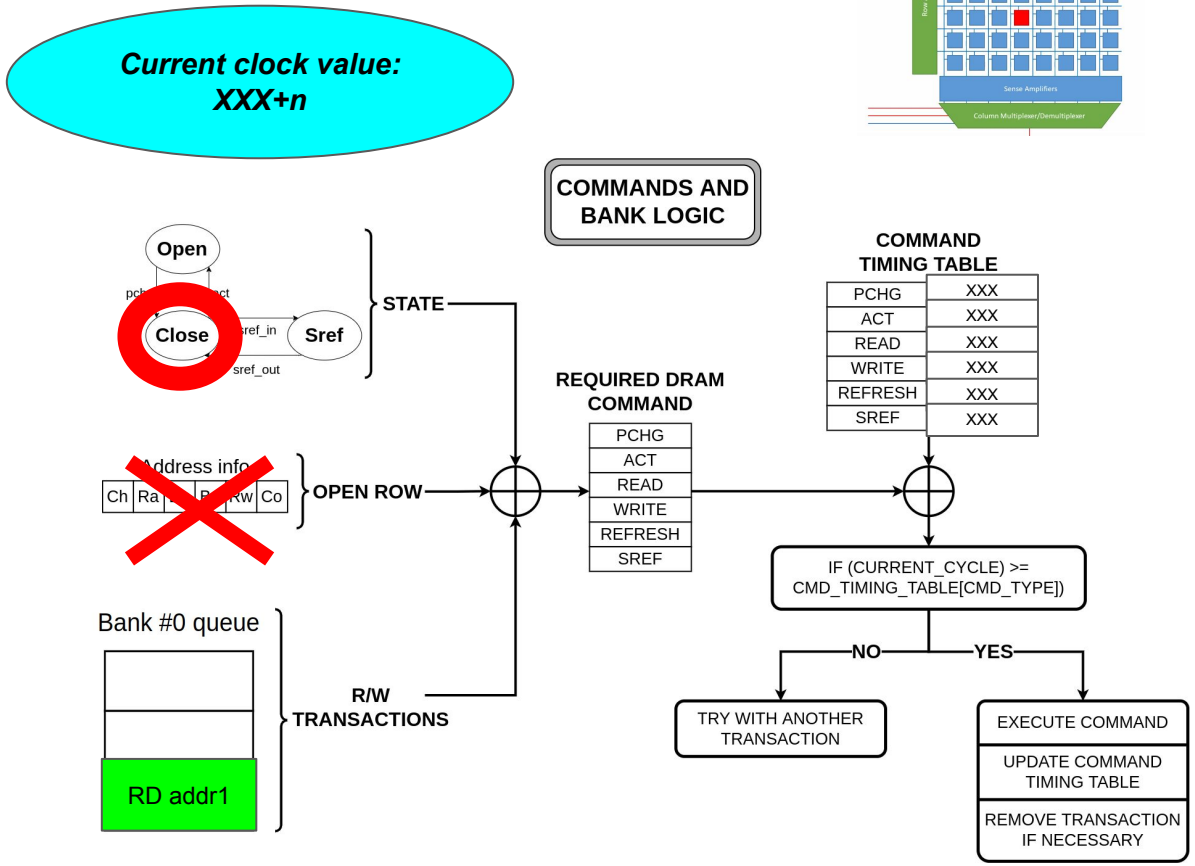




Structure (logical) of a memory controller

Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

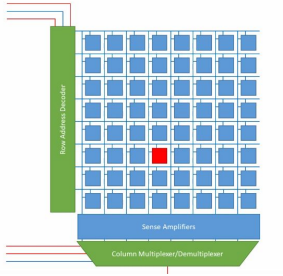


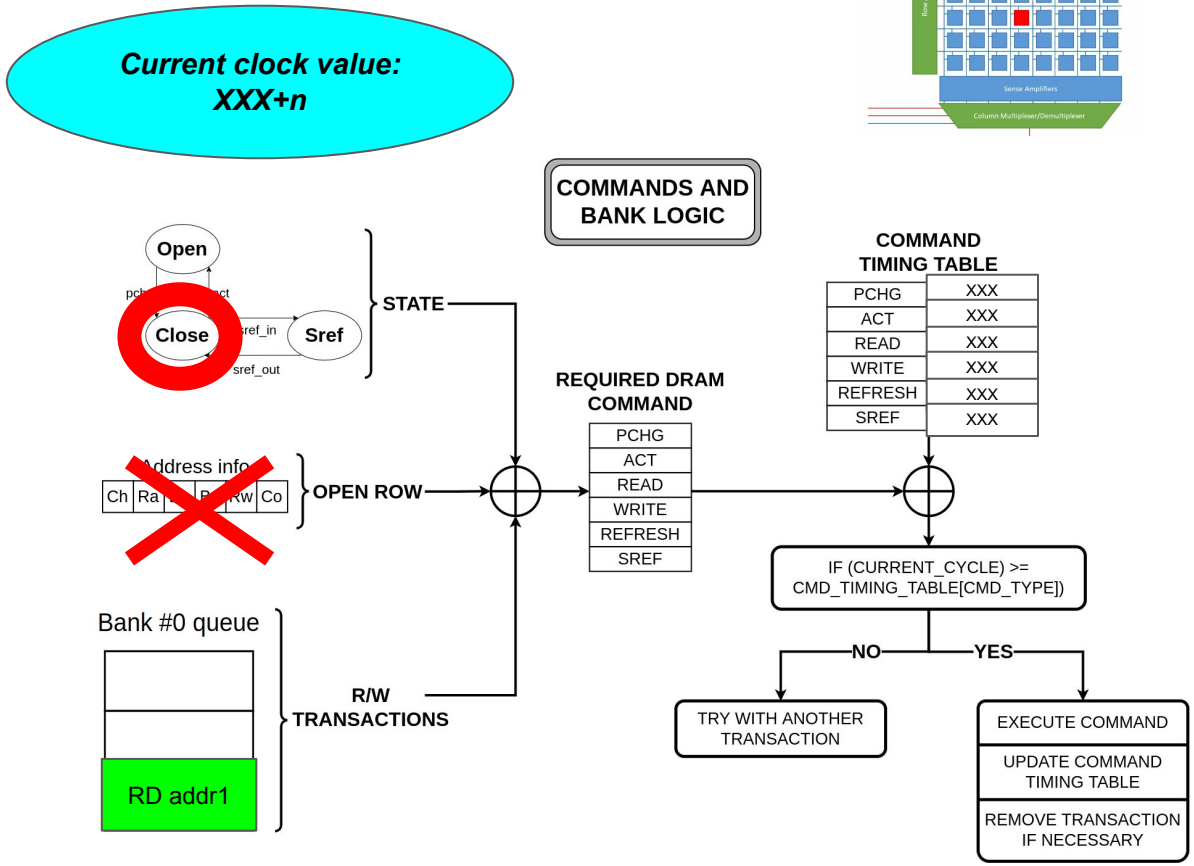


Structure (logical) of a memory controller

Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

Let's follow a FR-FCFS policy:



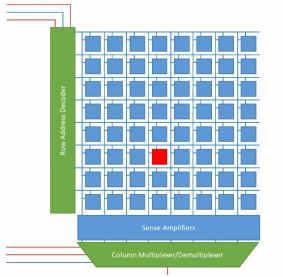


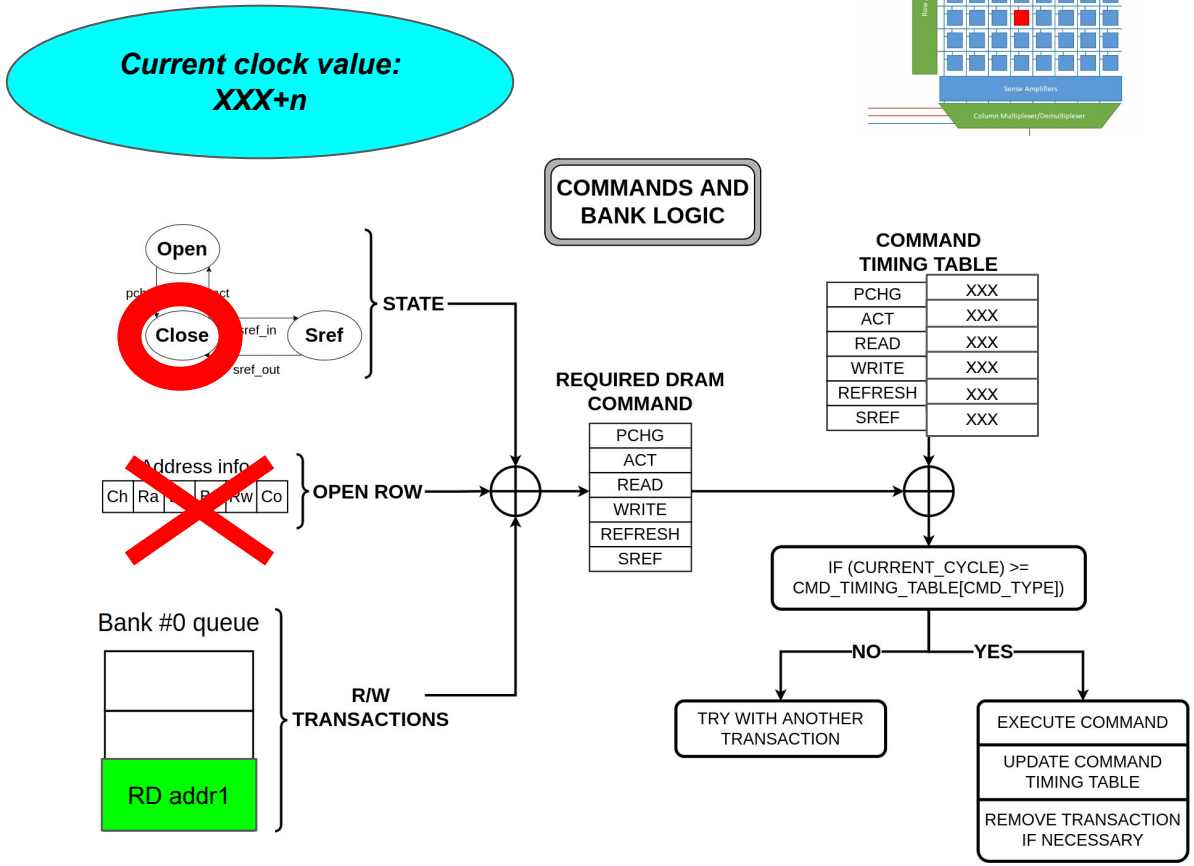
Structure (logical) of a memory controller

Let’s assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

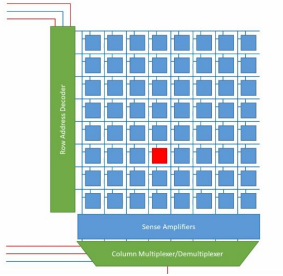
Let’s follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…





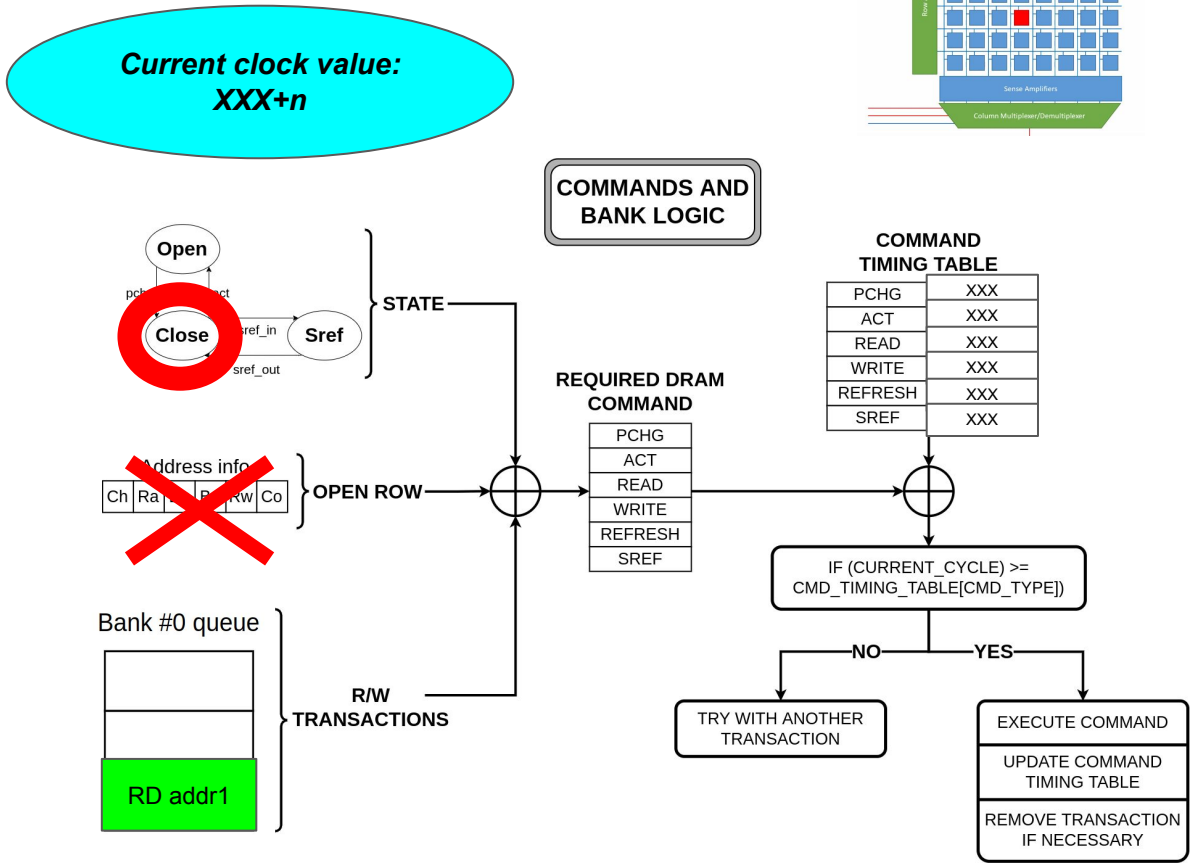
Structure (logical) of a memory controller



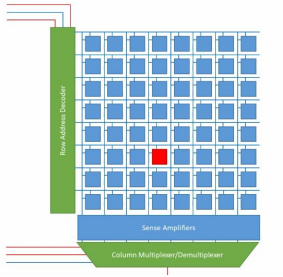
Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent



Structure (logical) of a memory controller

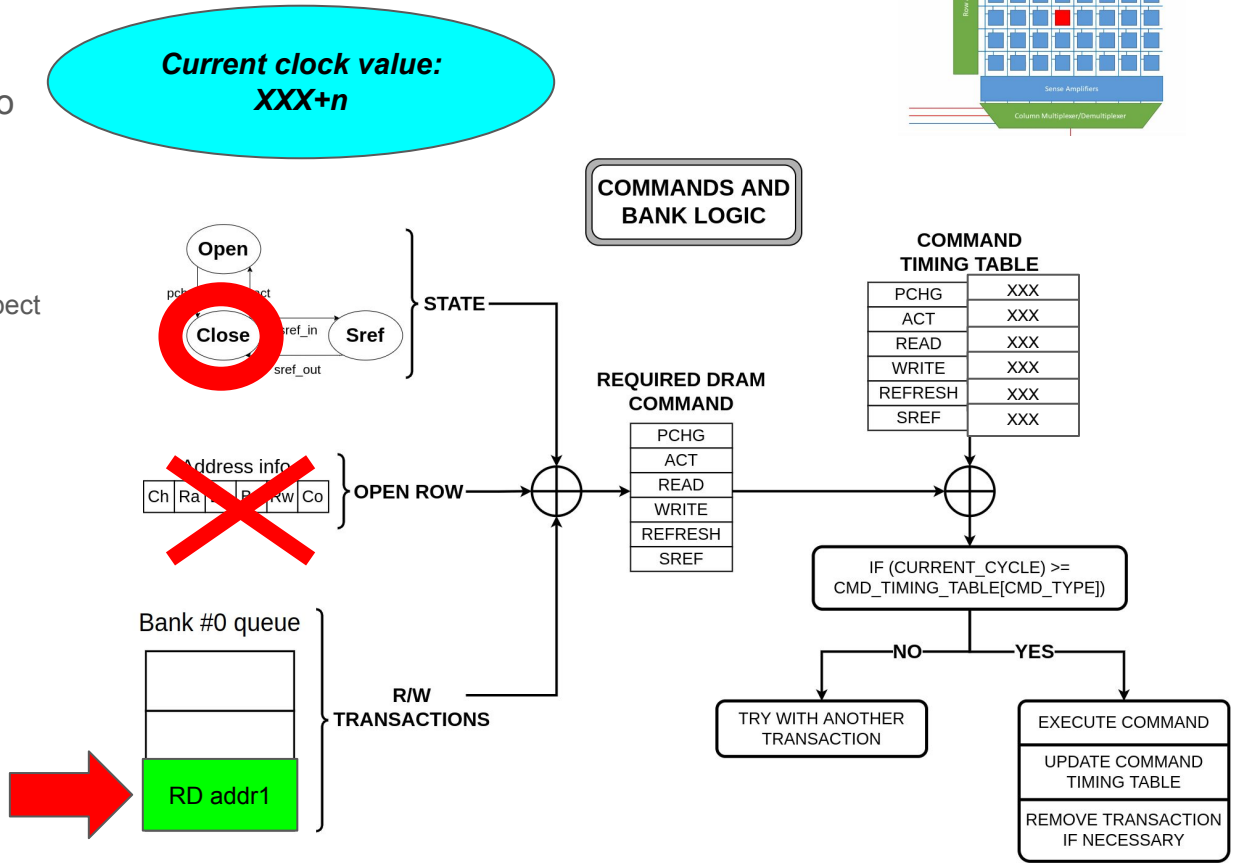


Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

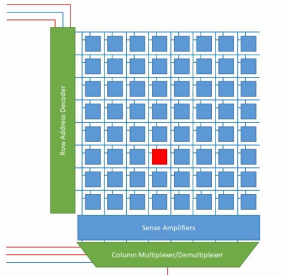
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

"RD addr1" needs an ACT:



Structure (logical) of a memory controller



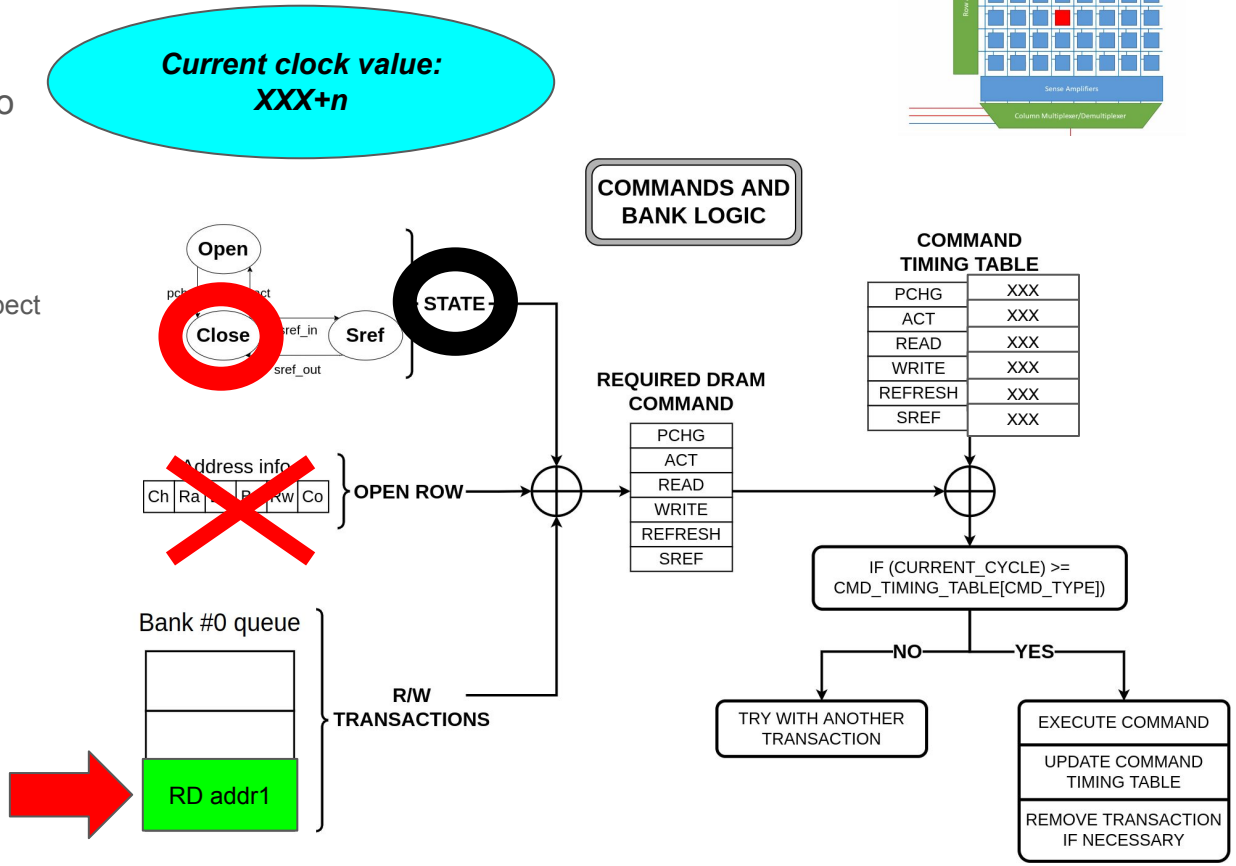
Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

Let's follow a FR-FCFS policy:

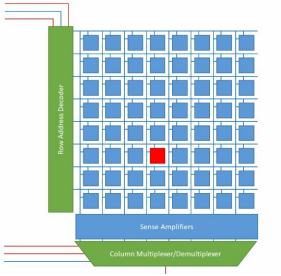
- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

"RD addr1" needs an ACT:

- Consider the state of the bank



Structure (logical) of a memory controller



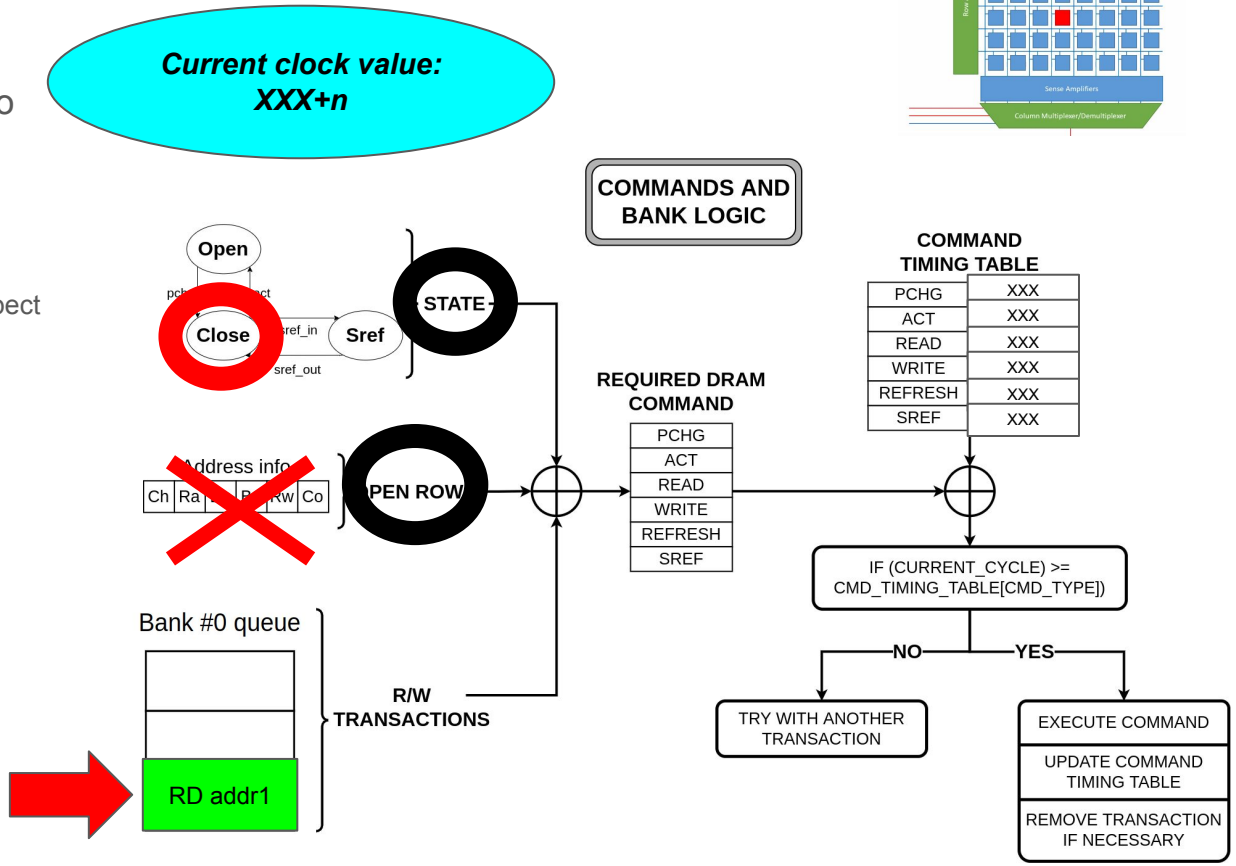
Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened



Structure (logical) of a memory controller

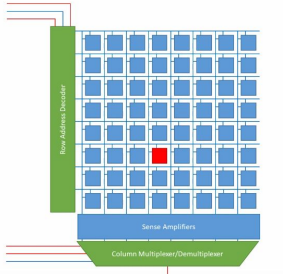
Let’s assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

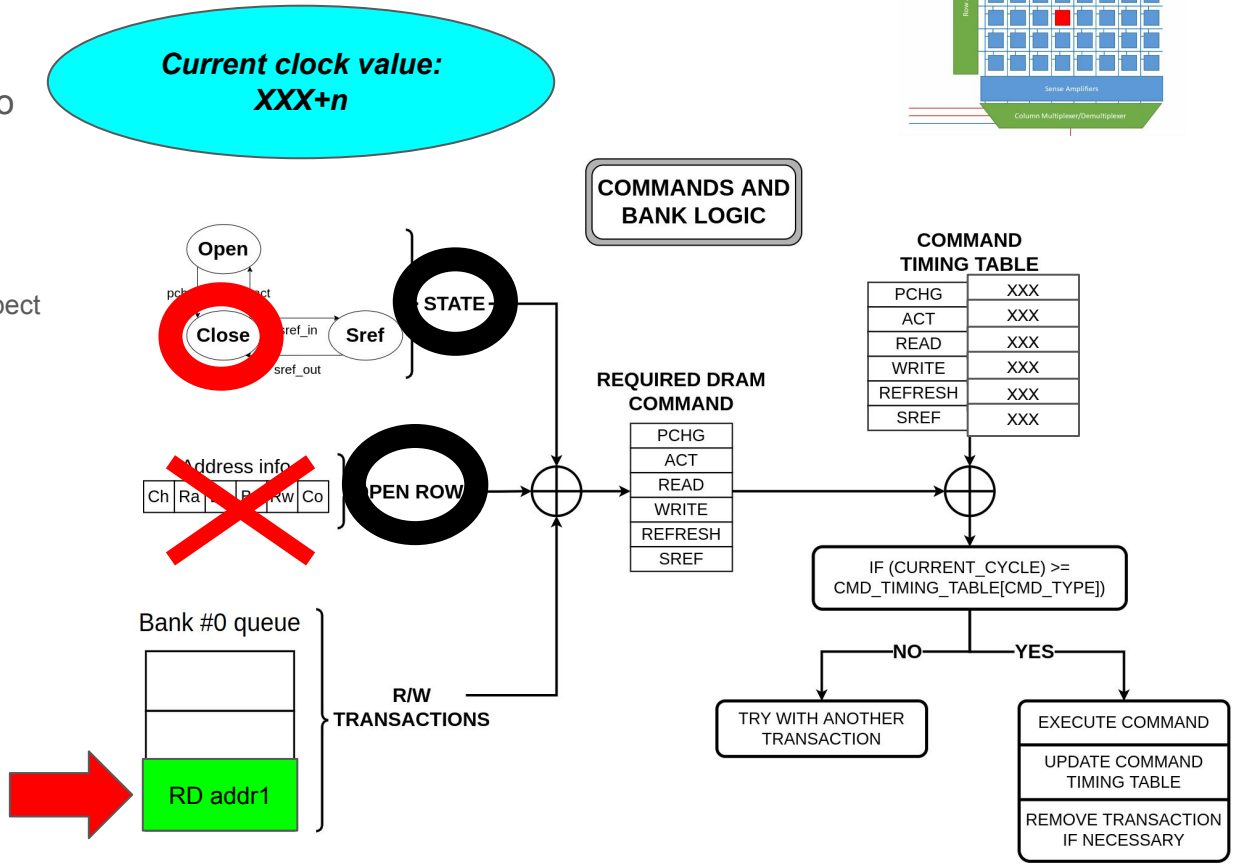
Let’s follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

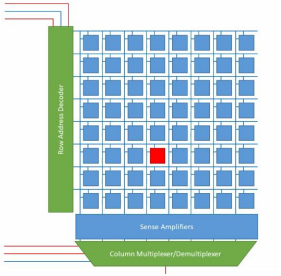
“RD addr1” needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the “current” transaction needs





Structure (logical) of a memory controller



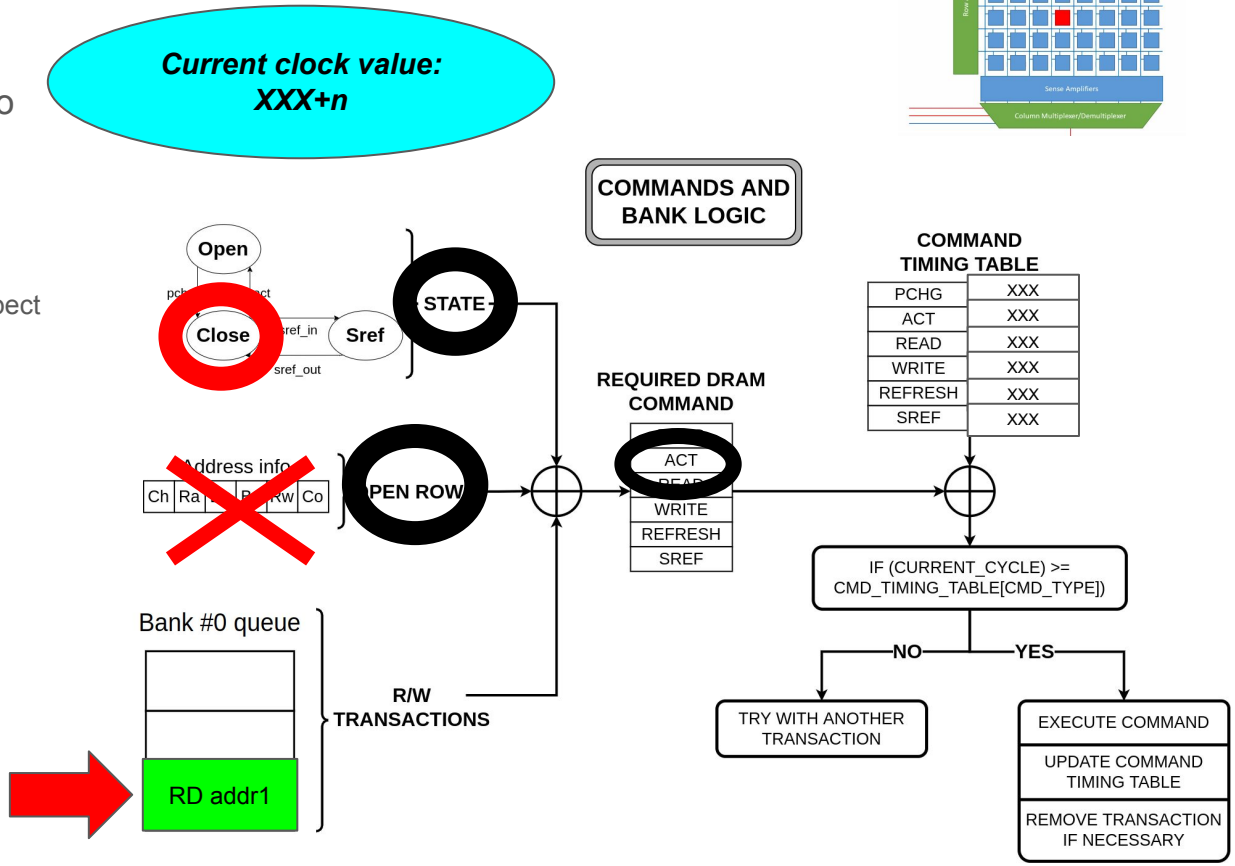
Let’s assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

Let’s follow a FR-FCFS policy:

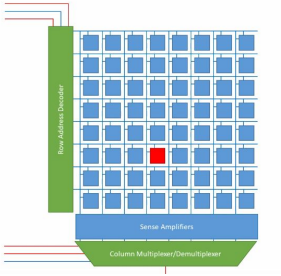
- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

“RD addr1” needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the “current” transaction needs



Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

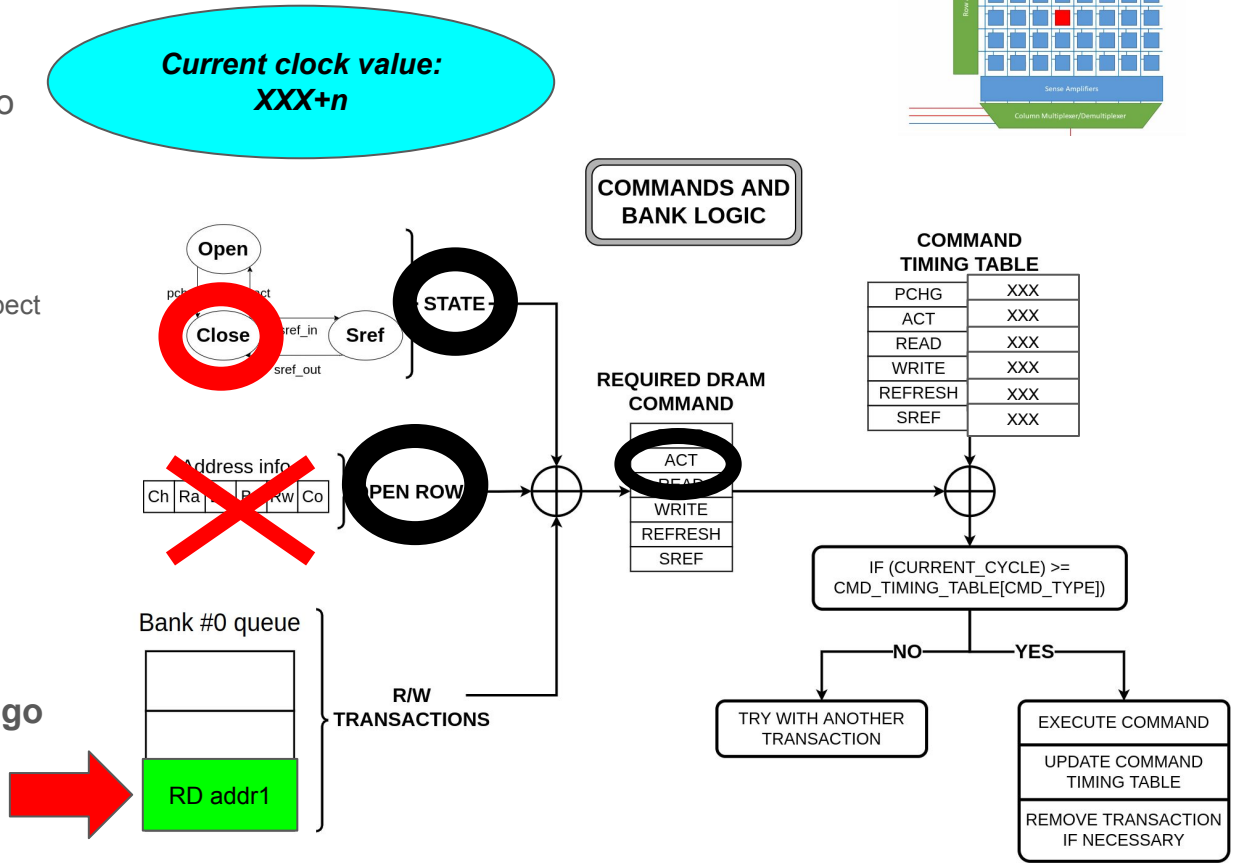
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

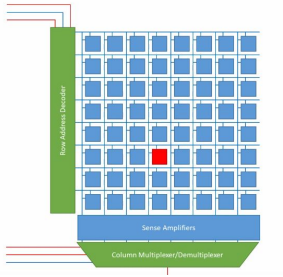
"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller



Let’s assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

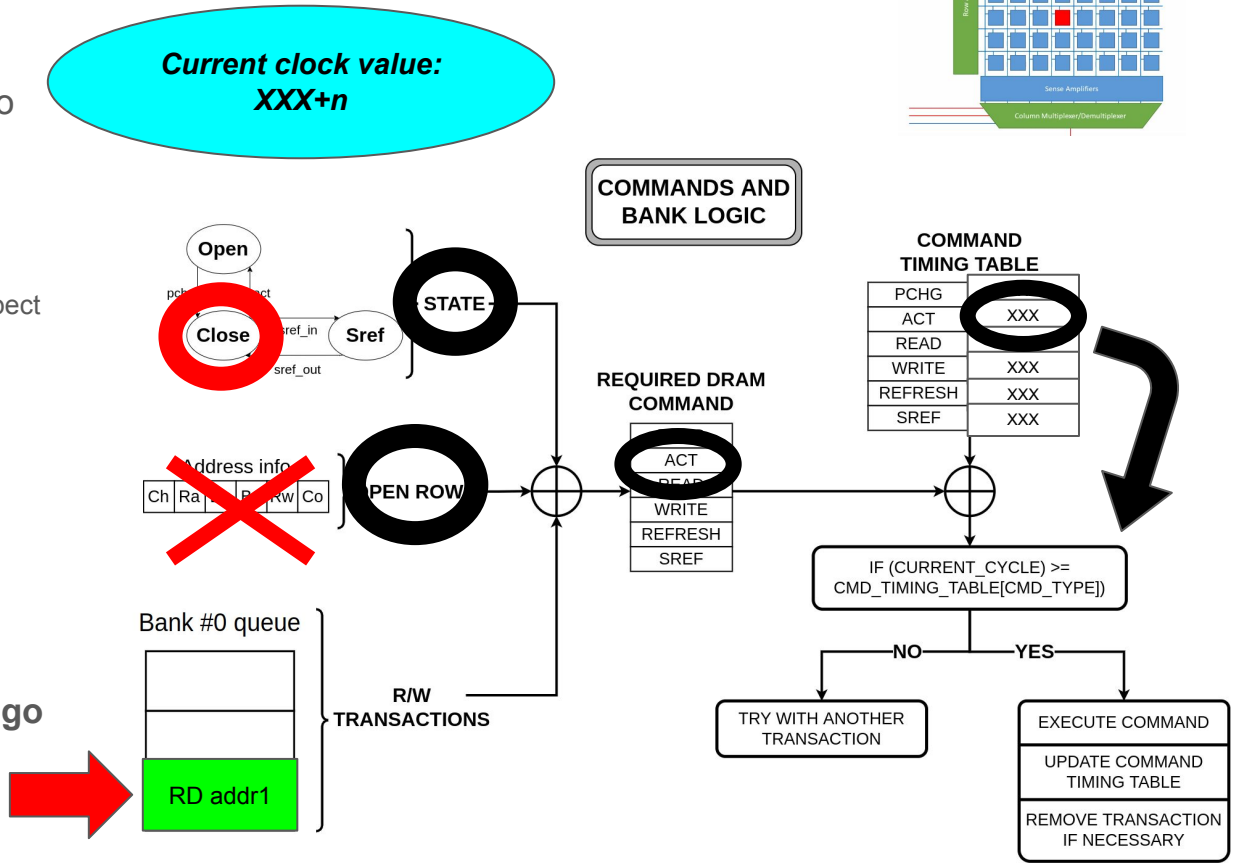
Let’s follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

“RD addr1” needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the “current” transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller

Let’s assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

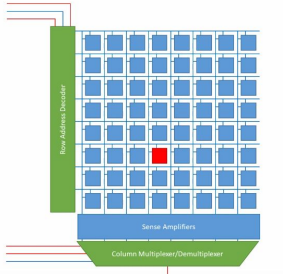
Let’s follow a FR-FCFS policy:

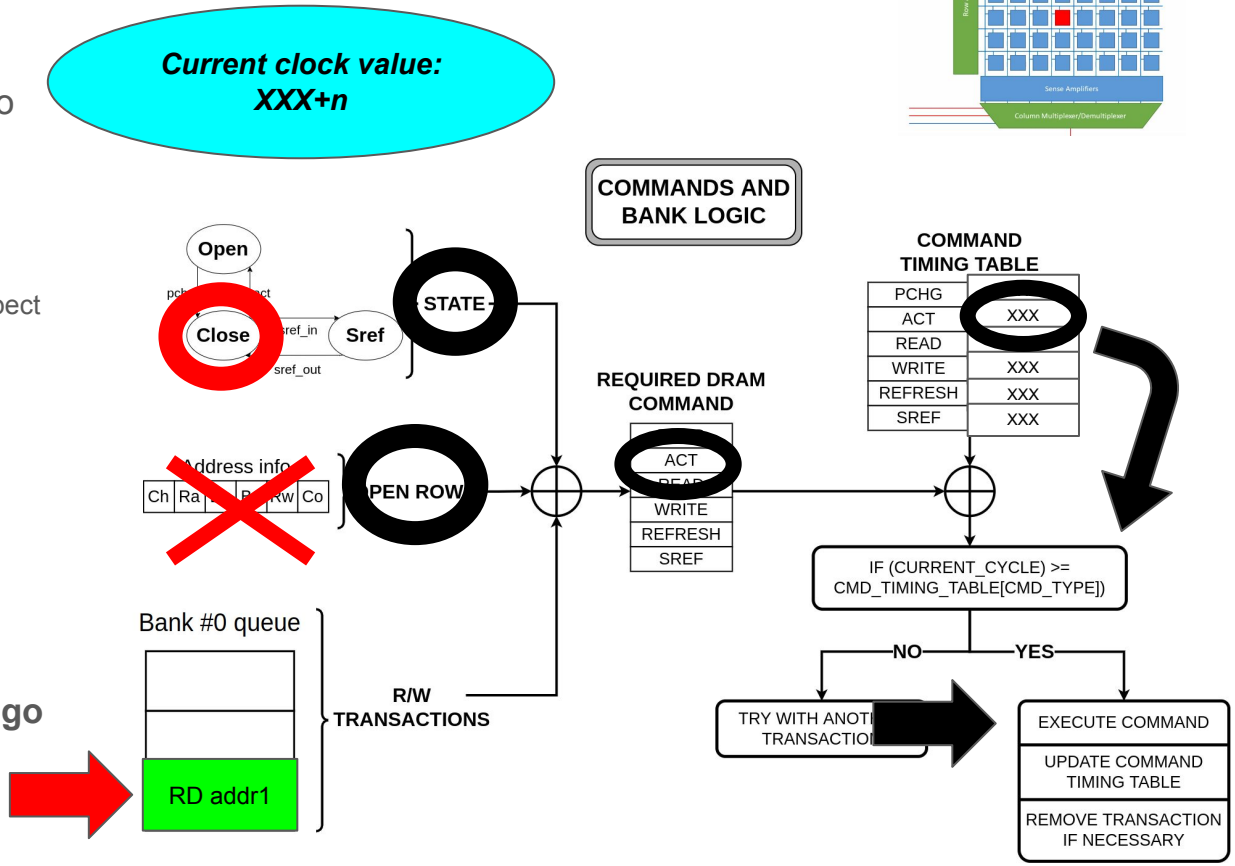
- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

“RD addr1” needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the “current” transaction needs

Does the execution of this command go against previous timing constraints?





Structure (logical) of a memory controller

Let’s assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

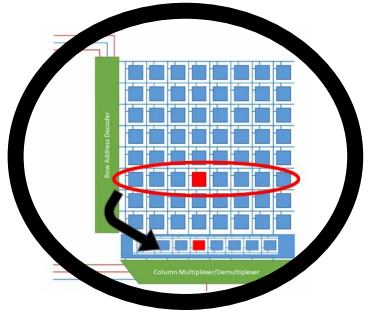
Let’s follow a FR-FCFS policy:

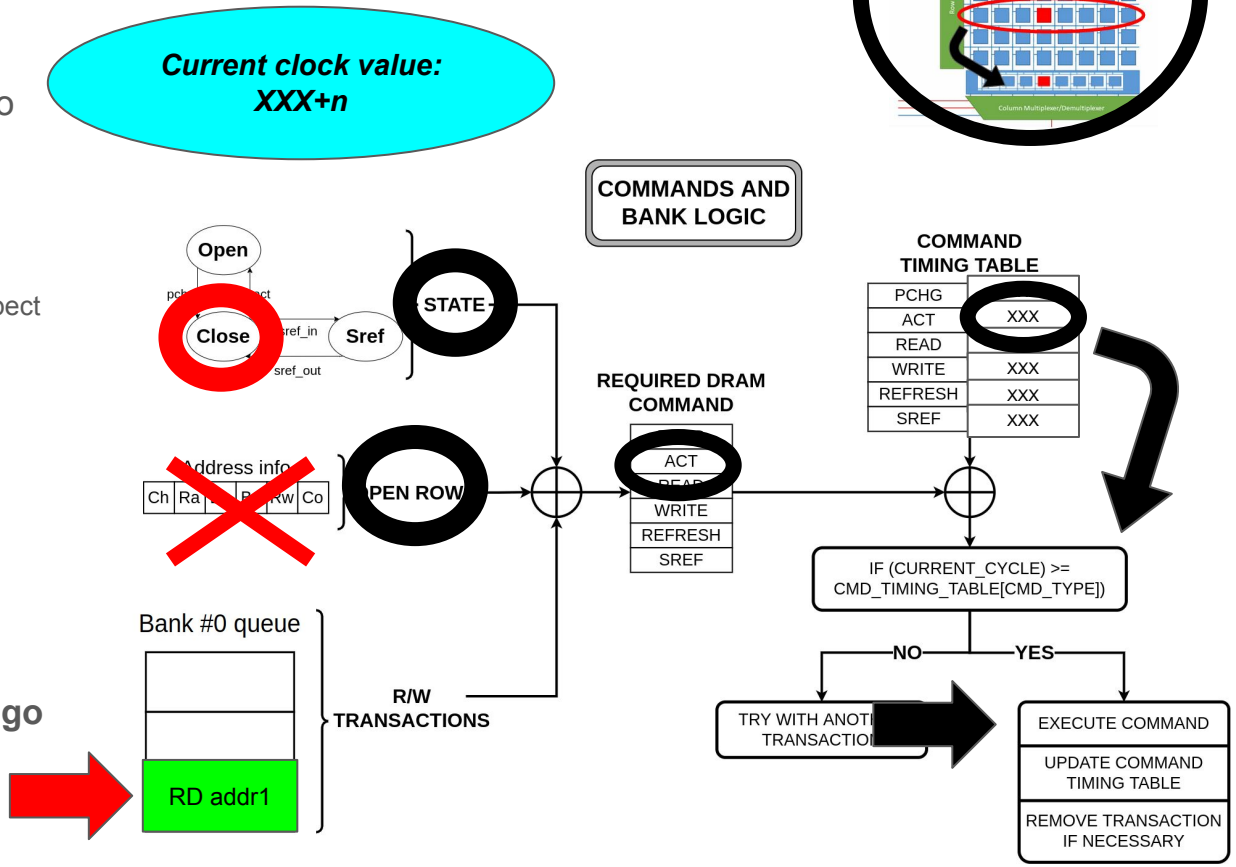
- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

“RD addr1” needs an ACT:

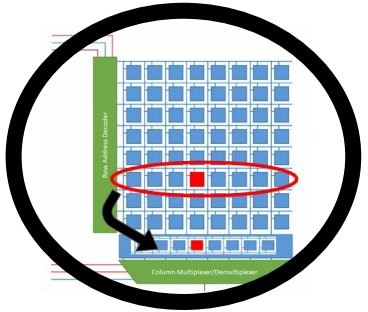
- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the “current” transaction needs

Does the execution of this command go against previous timing constraints?





Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

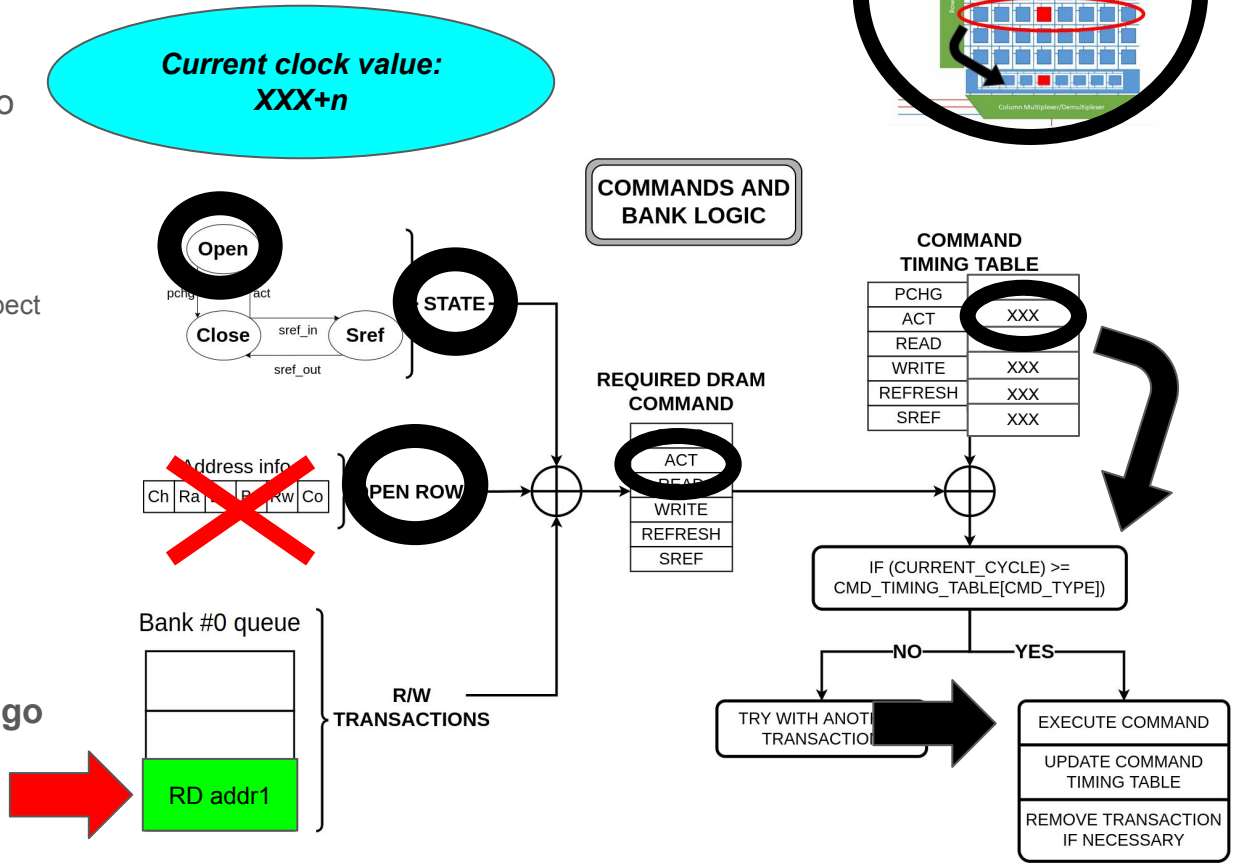
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

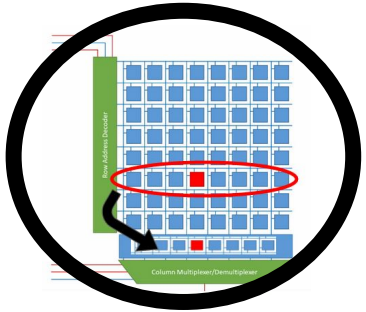
"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

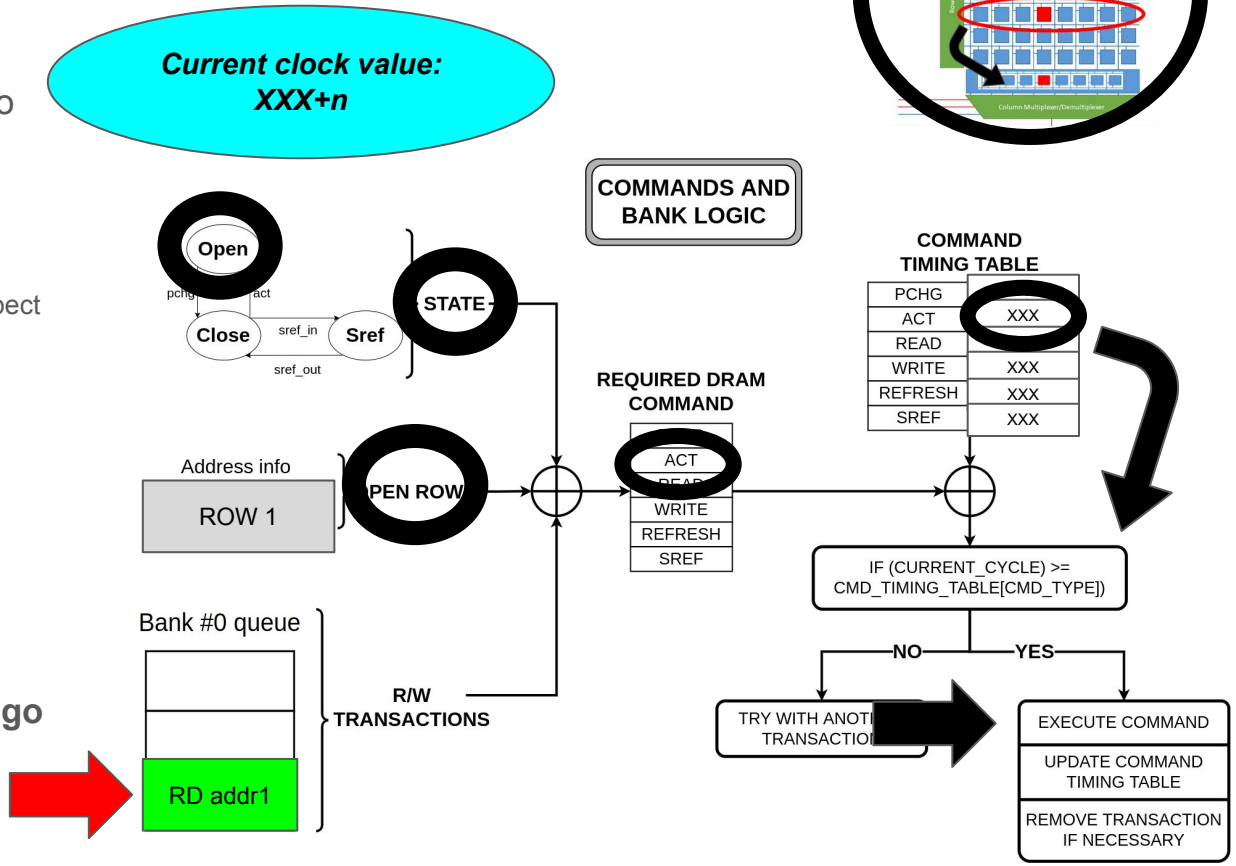
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller

Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

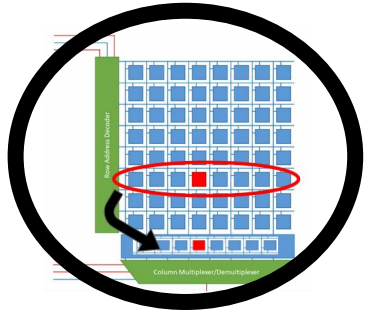
Let's follow a FR-FCFS policy:

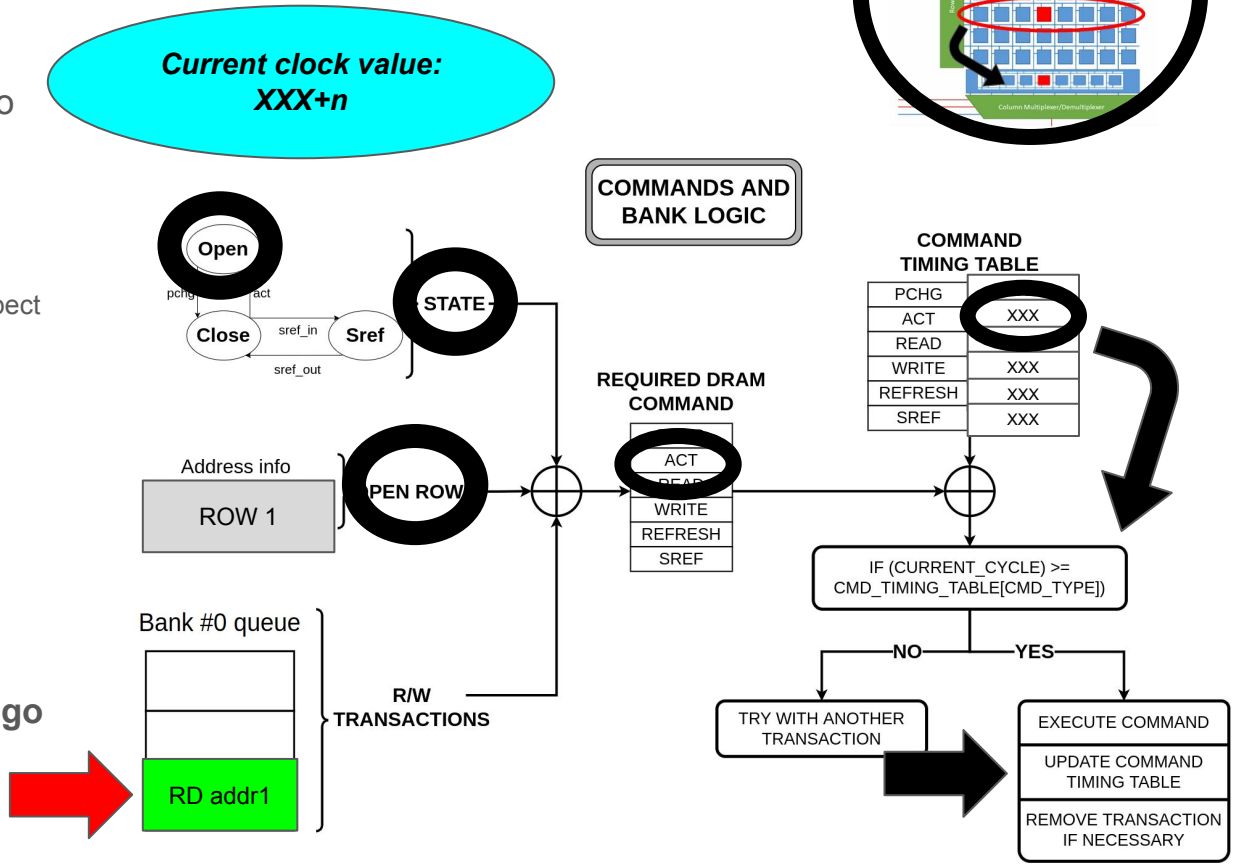
- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?





Structure (logical) of a memory controller

Let’s assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

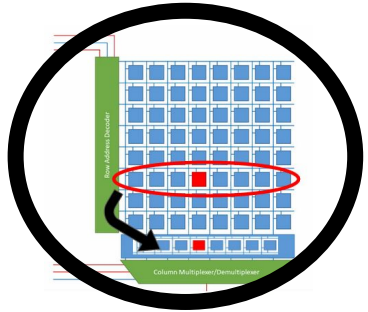
Let’s follow a FR-FCFS policy:

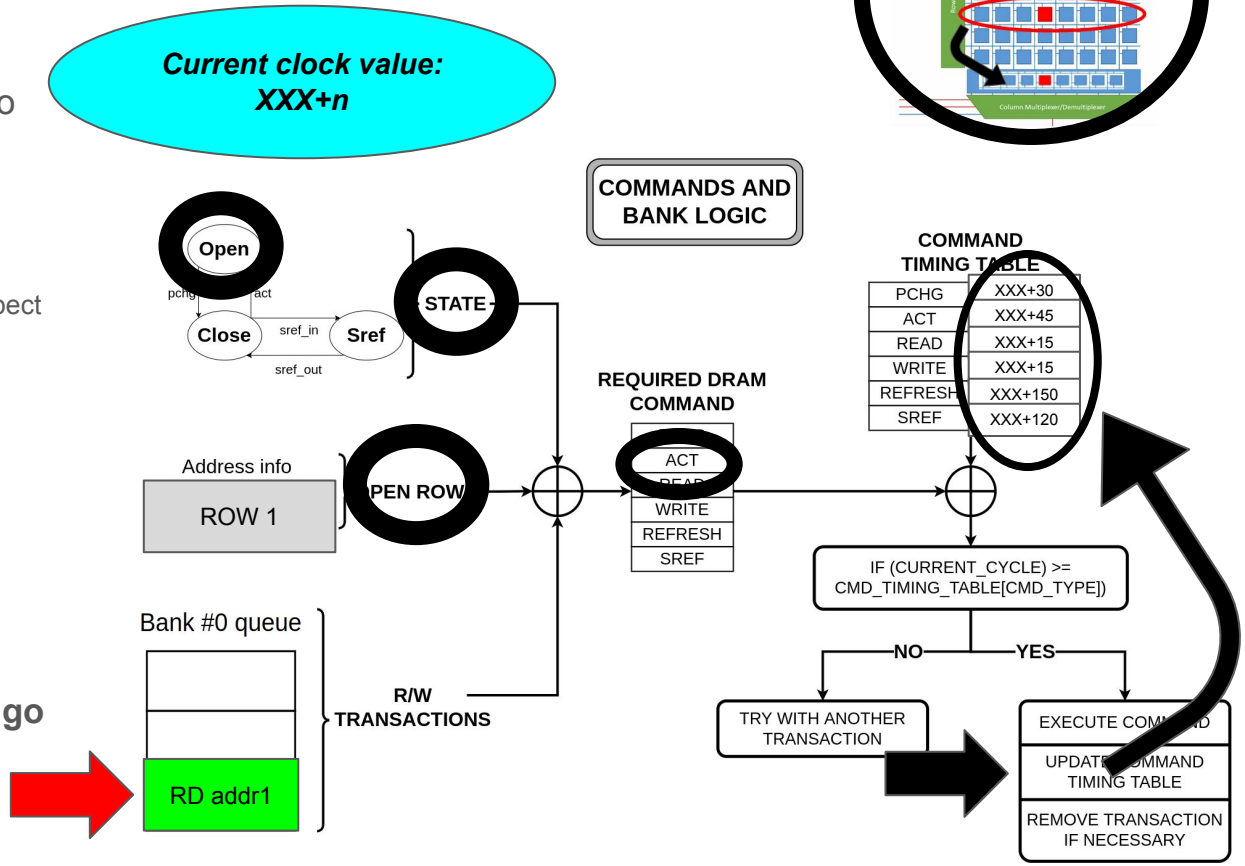
- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

“RD addr1” needs an ACT:

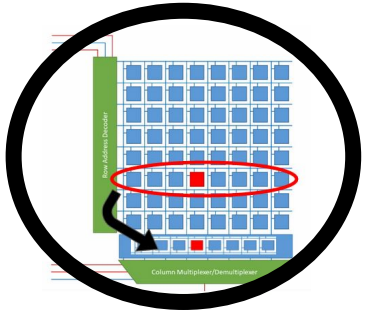
- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the “current” transaction needs

Does the execution of this command go against previous timing constraints?





Structure (logical) of a memory controller



Let’s assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

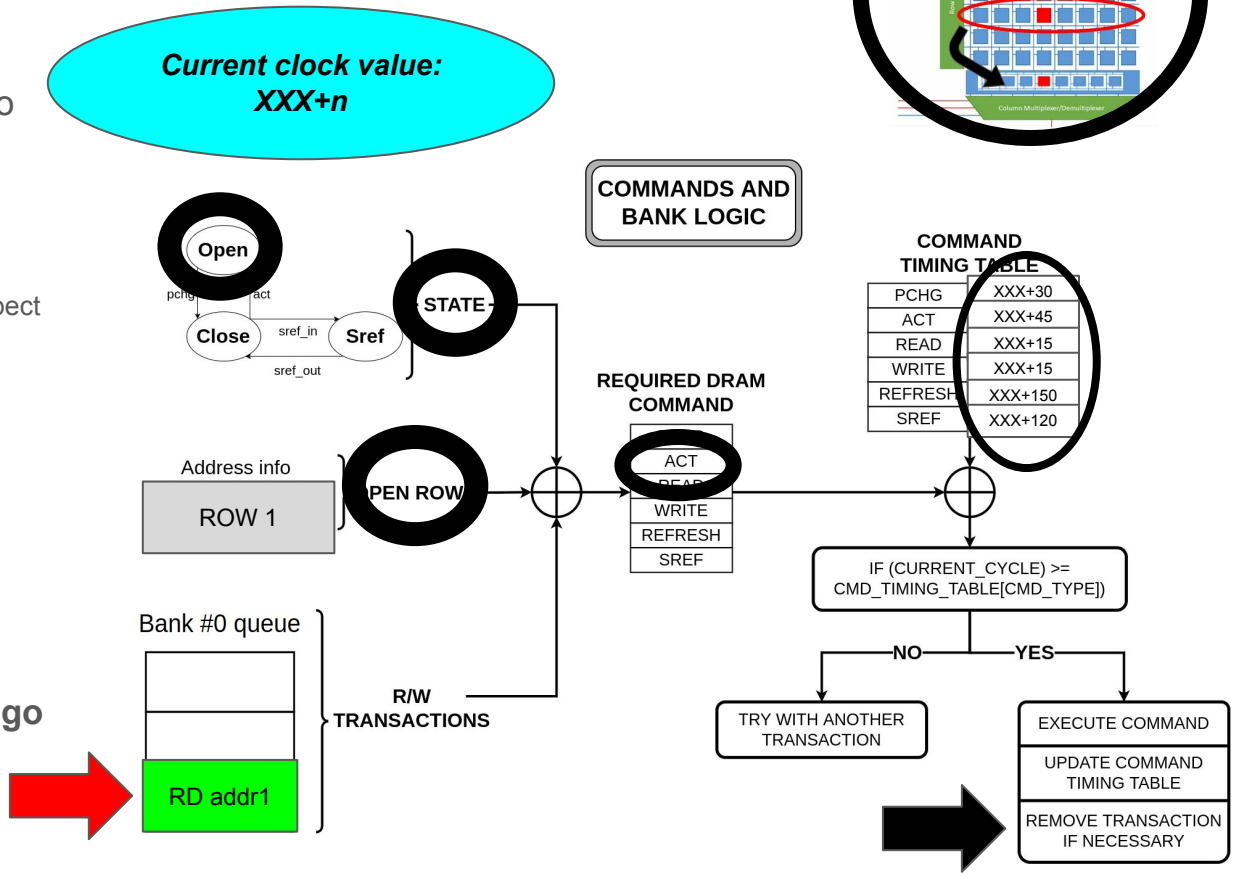
Let’s follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

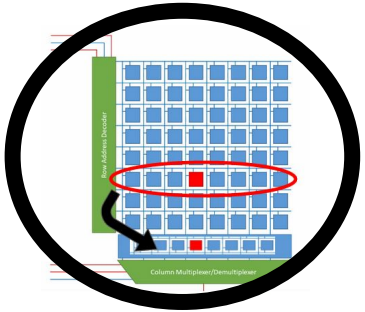
“RD addr1” needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the “current” transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

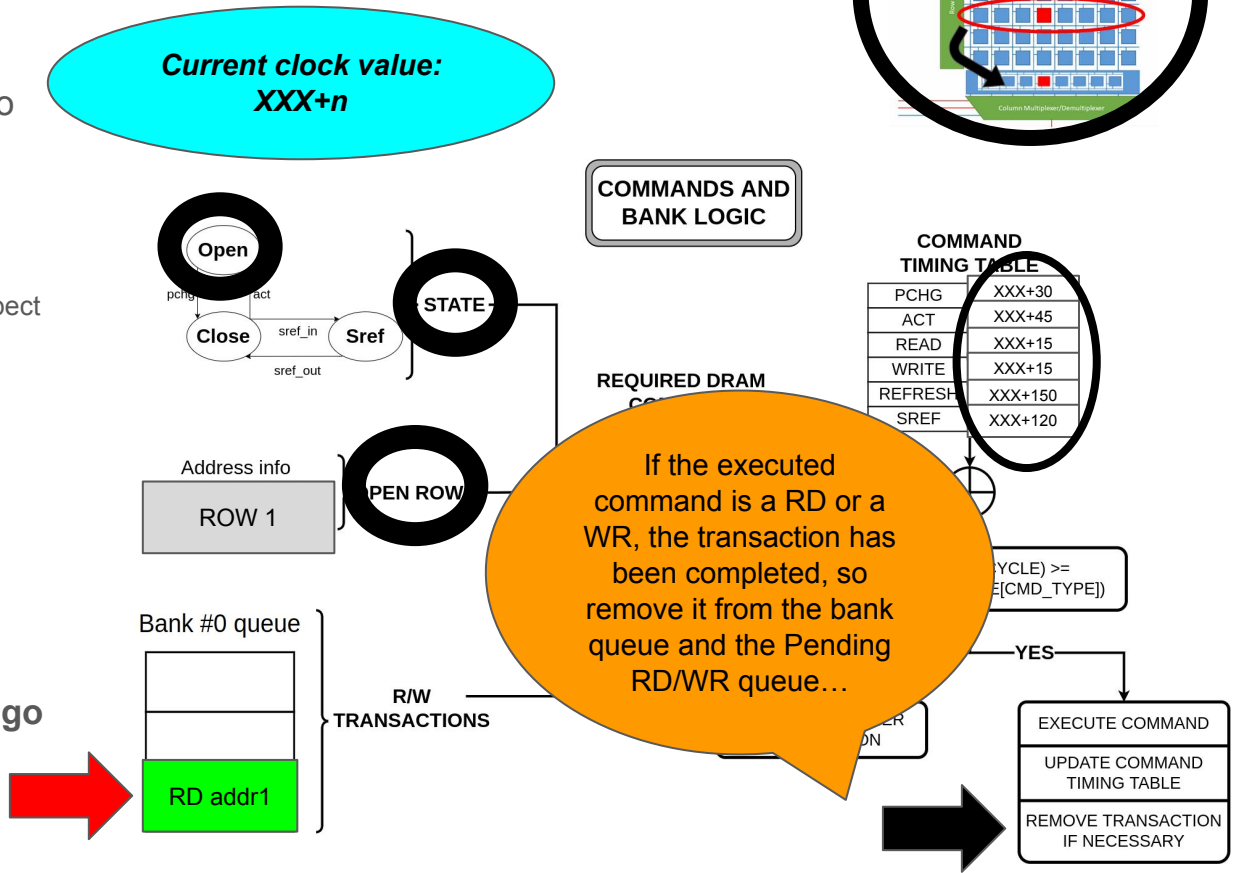
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

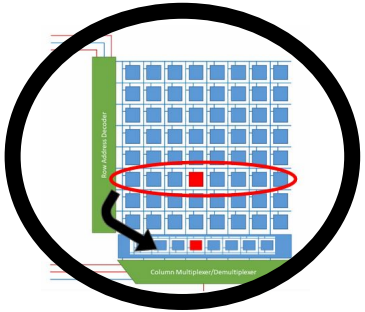
"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller



Let's assume an idle (**CLOSE
nothing in the row buffe

Let's follow a FR-F

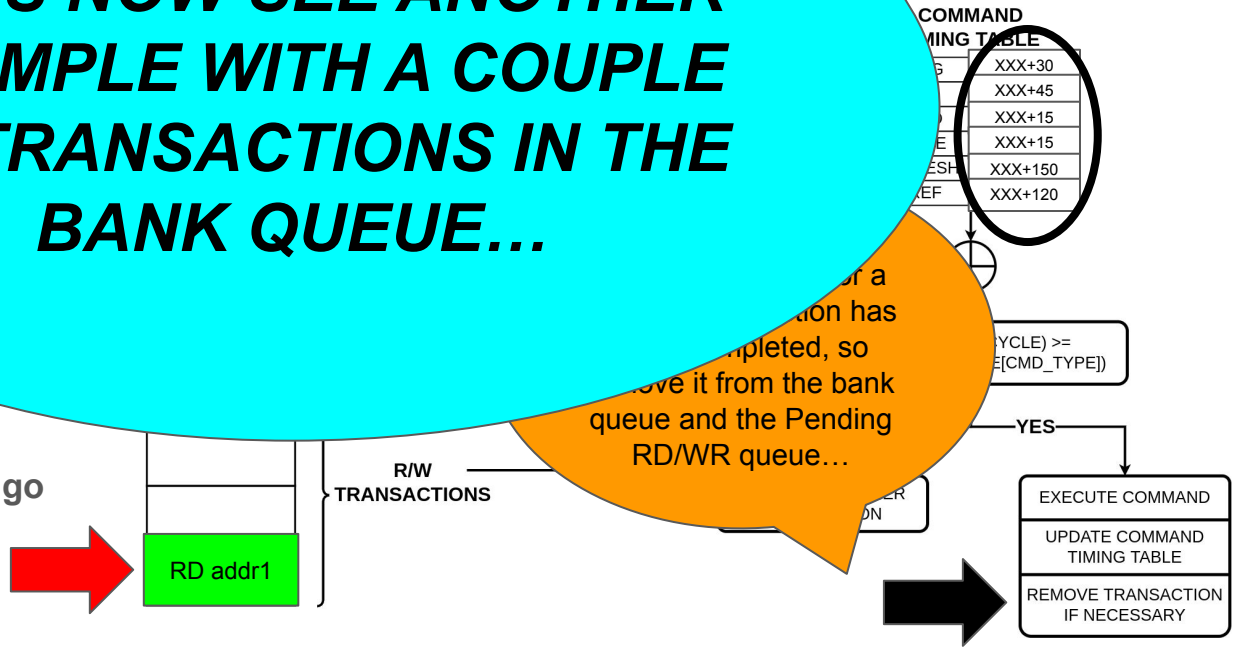
- Try to issue fi
  the timing cor
- …from oldest

"RD addr1" needs a

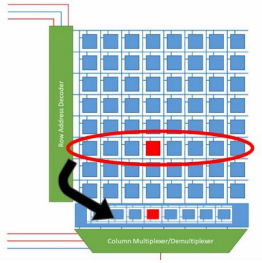
- Consider the state of the
- Consider which row is current
- Consider what row the "current" trans
  needs

Does the execution of this command go against previous timing constraints?

LET'S NOW SEE ANOTHER EXAMPLE WITH A COUPLE OF TRANSACTIONS IN THE BANK QUEUE…



Structure (logical) of a memory controller



Let’s assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

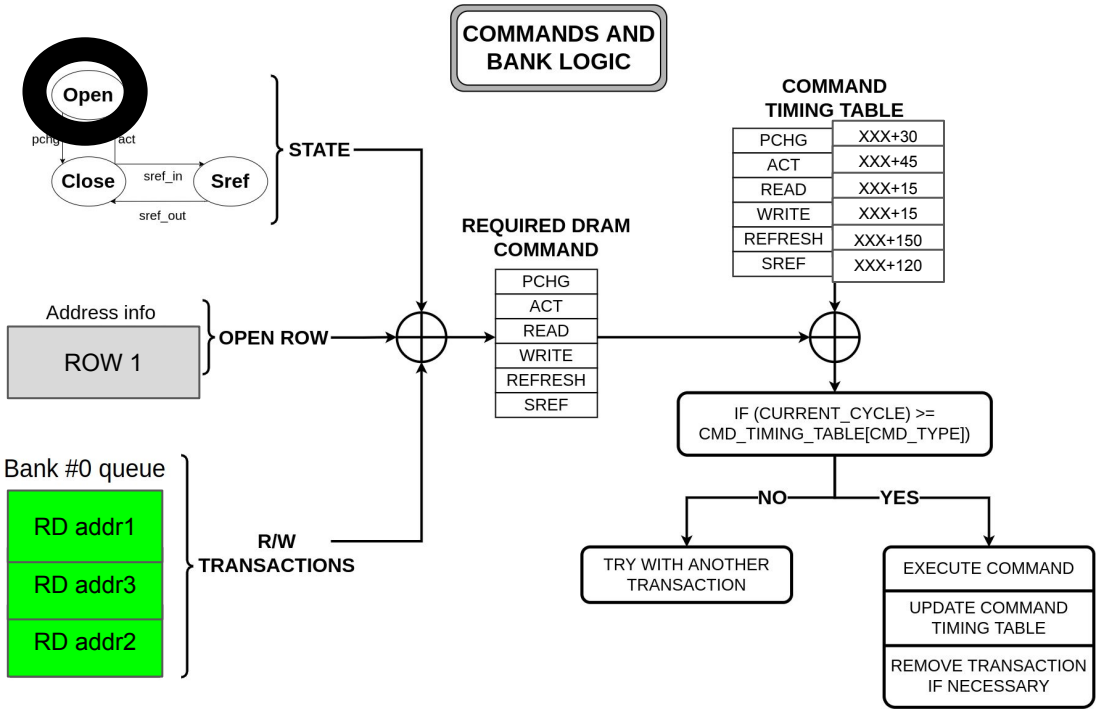
Let’s follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

“RD addr1” needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the “current” transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller

Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

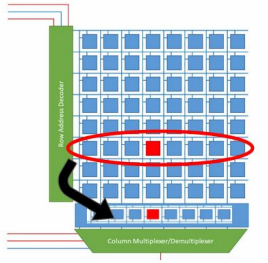
Let's follow a FR-FCFS policy:

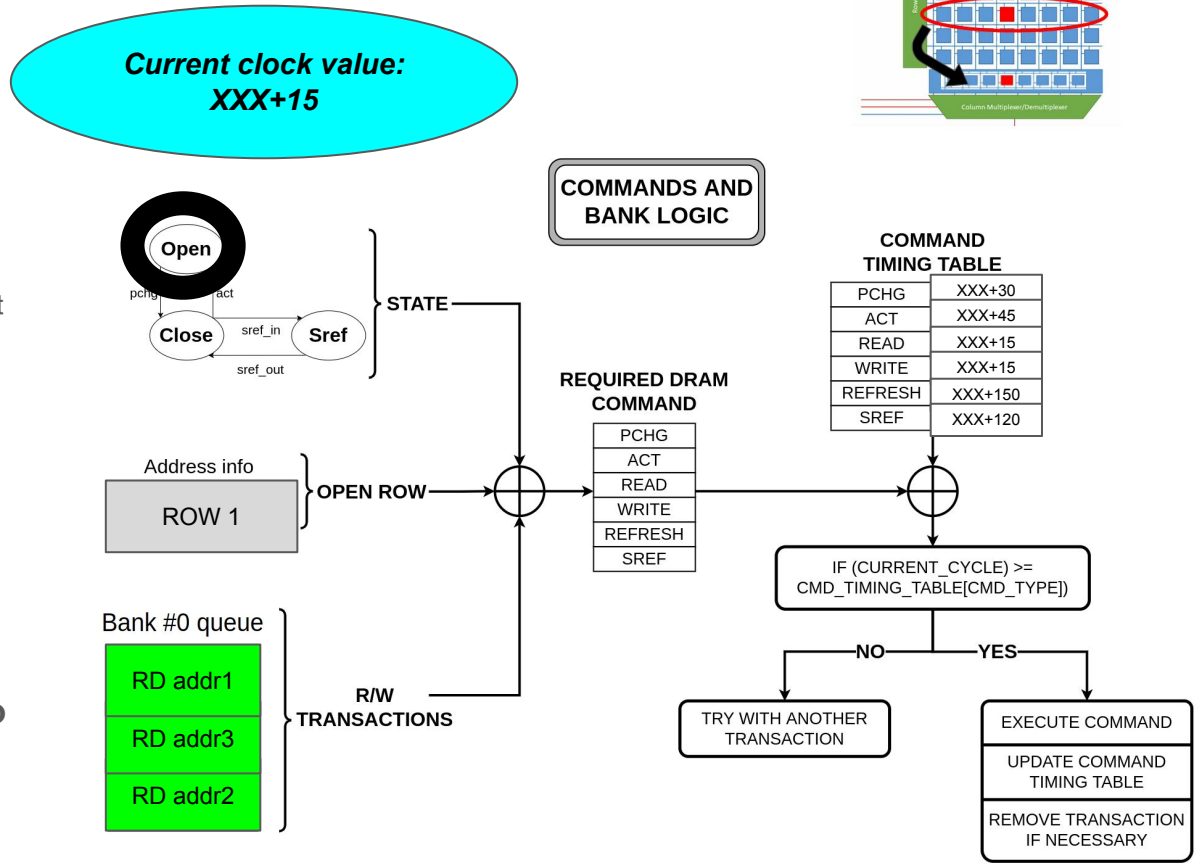
- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

"RD addr1" needs an ACT:

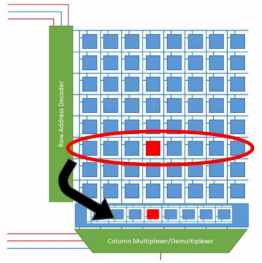
- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?





Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

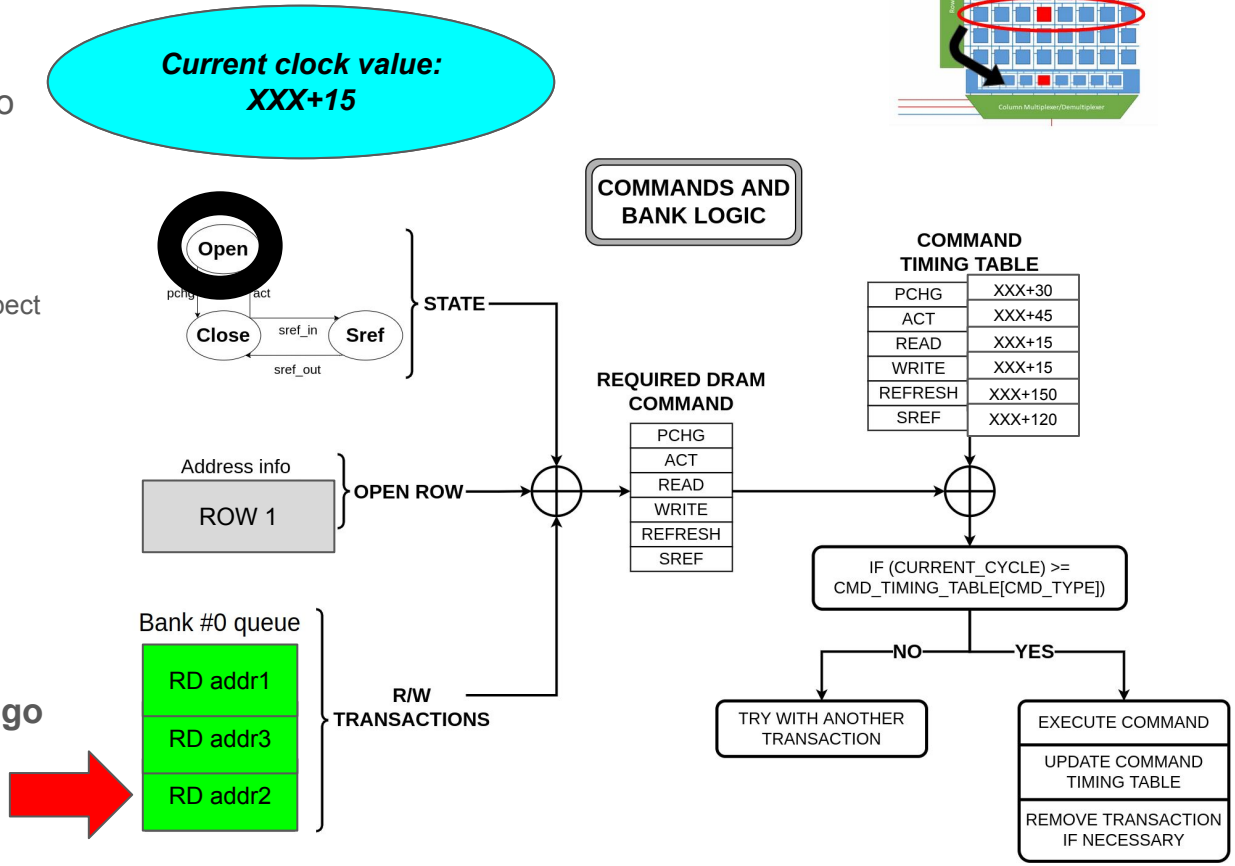
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

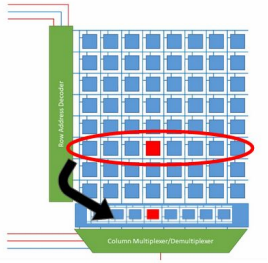
"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

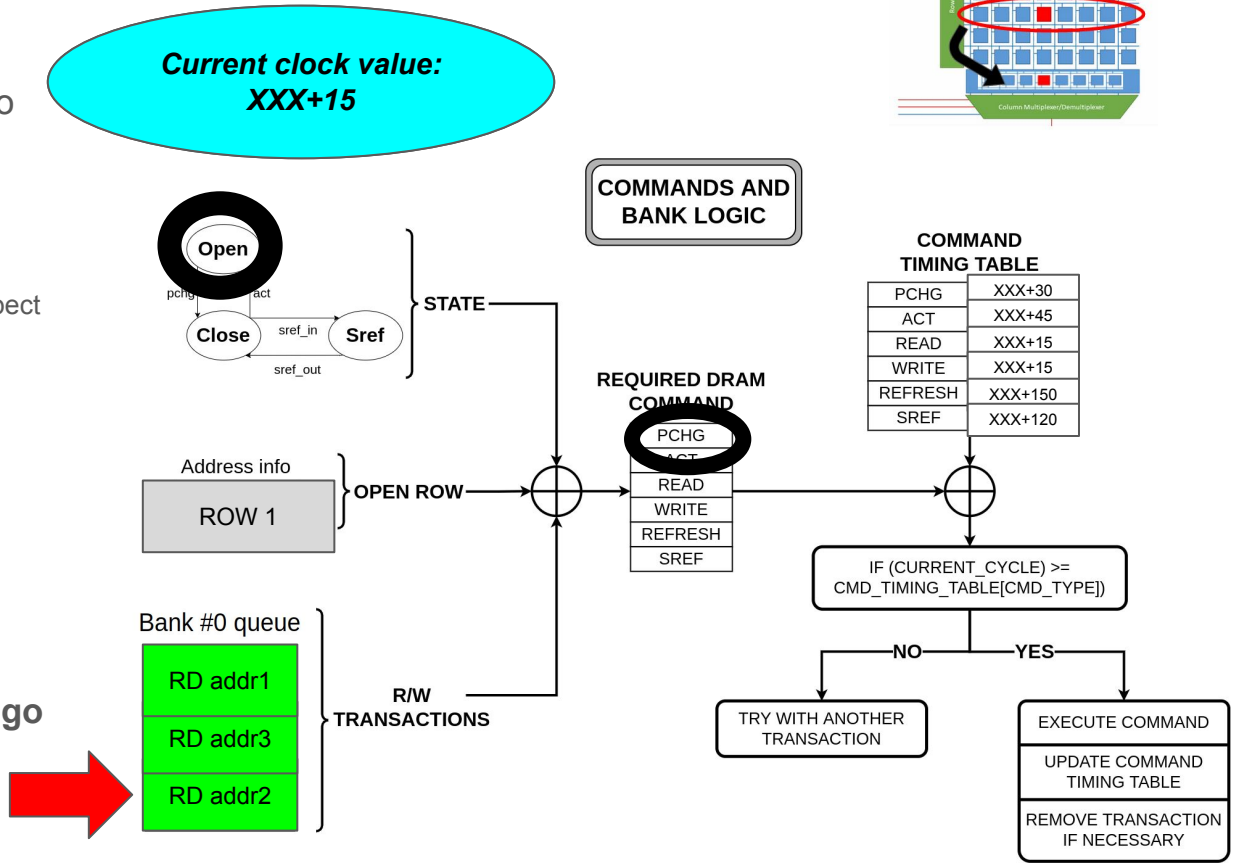
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

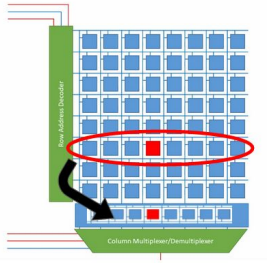
"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

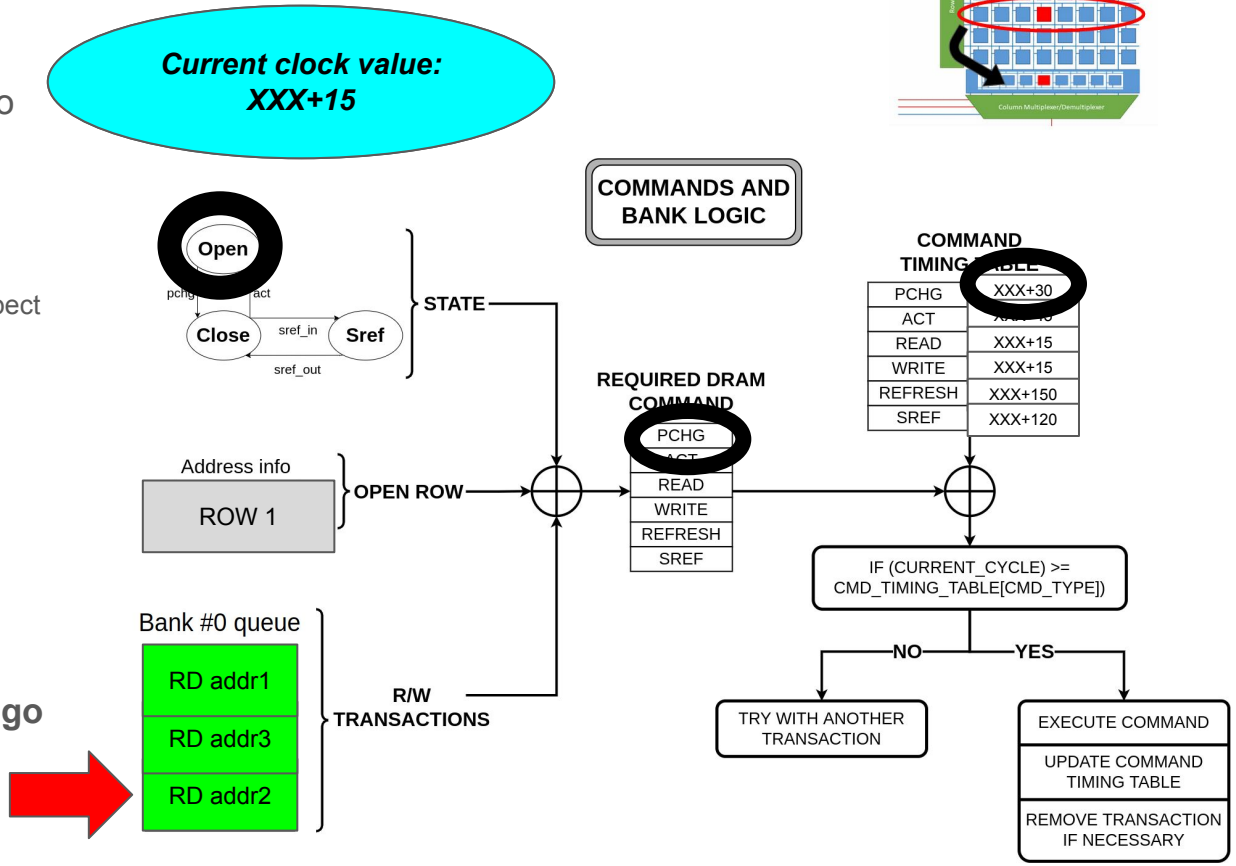
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

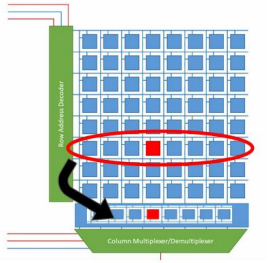
"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

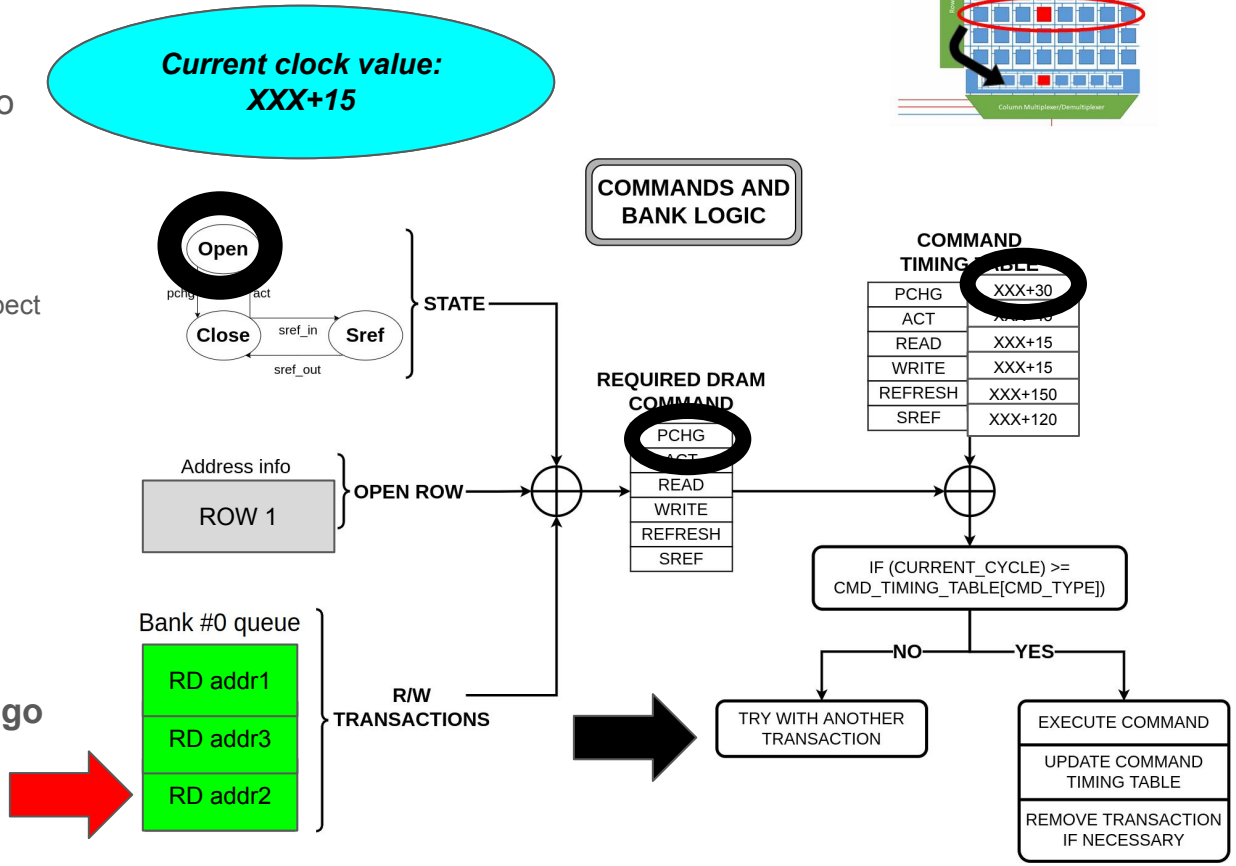
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

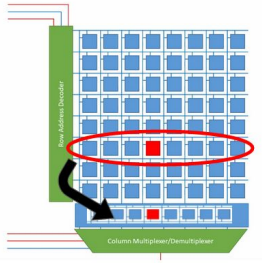
"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

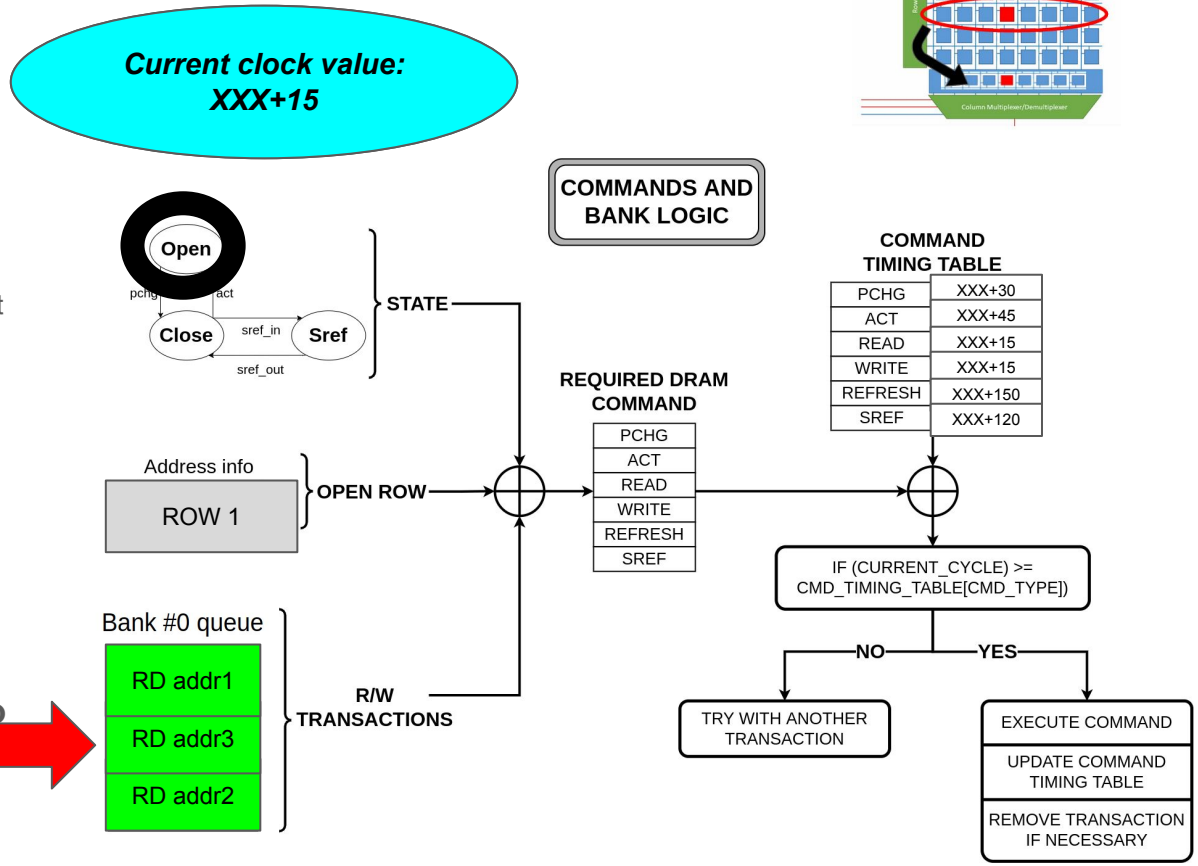
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

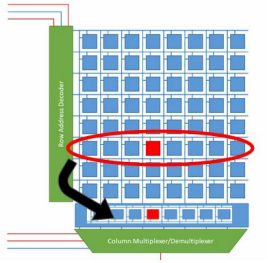
"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

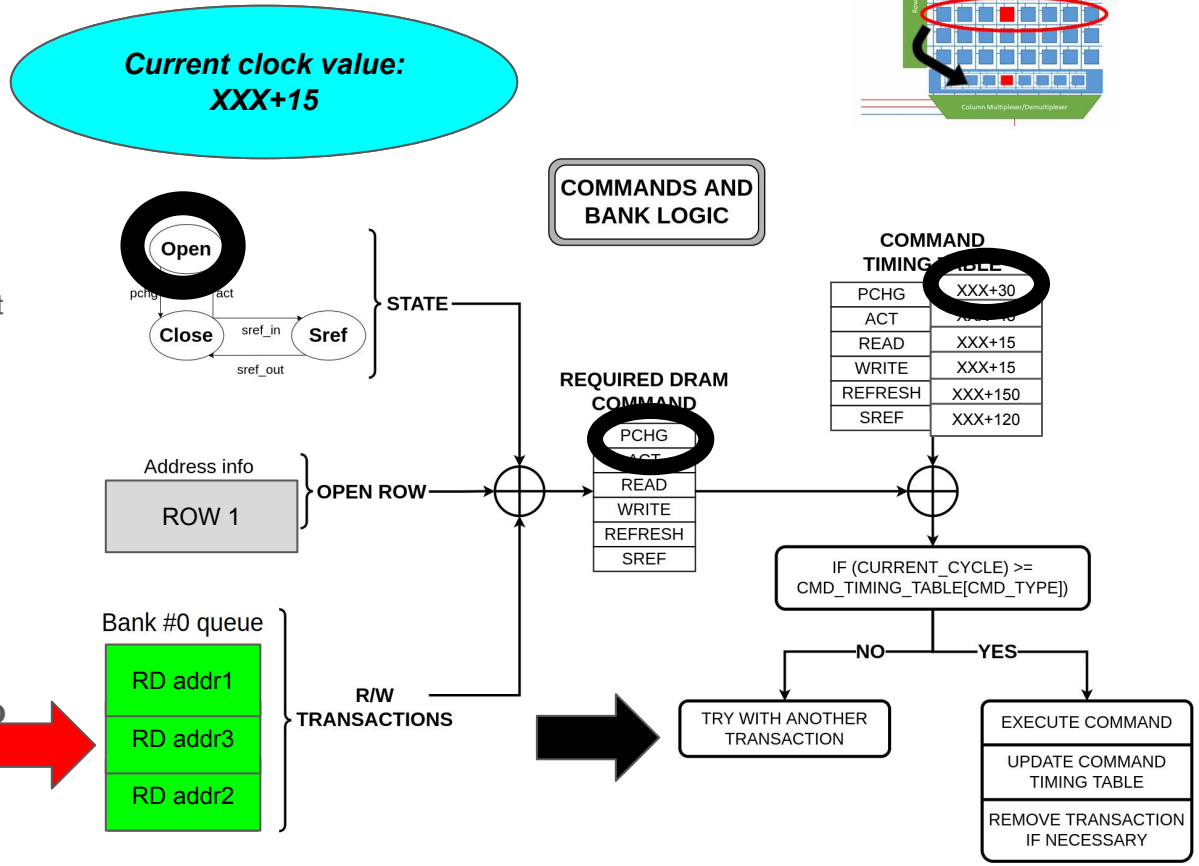
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

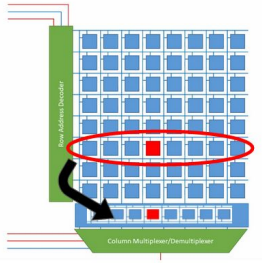
"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

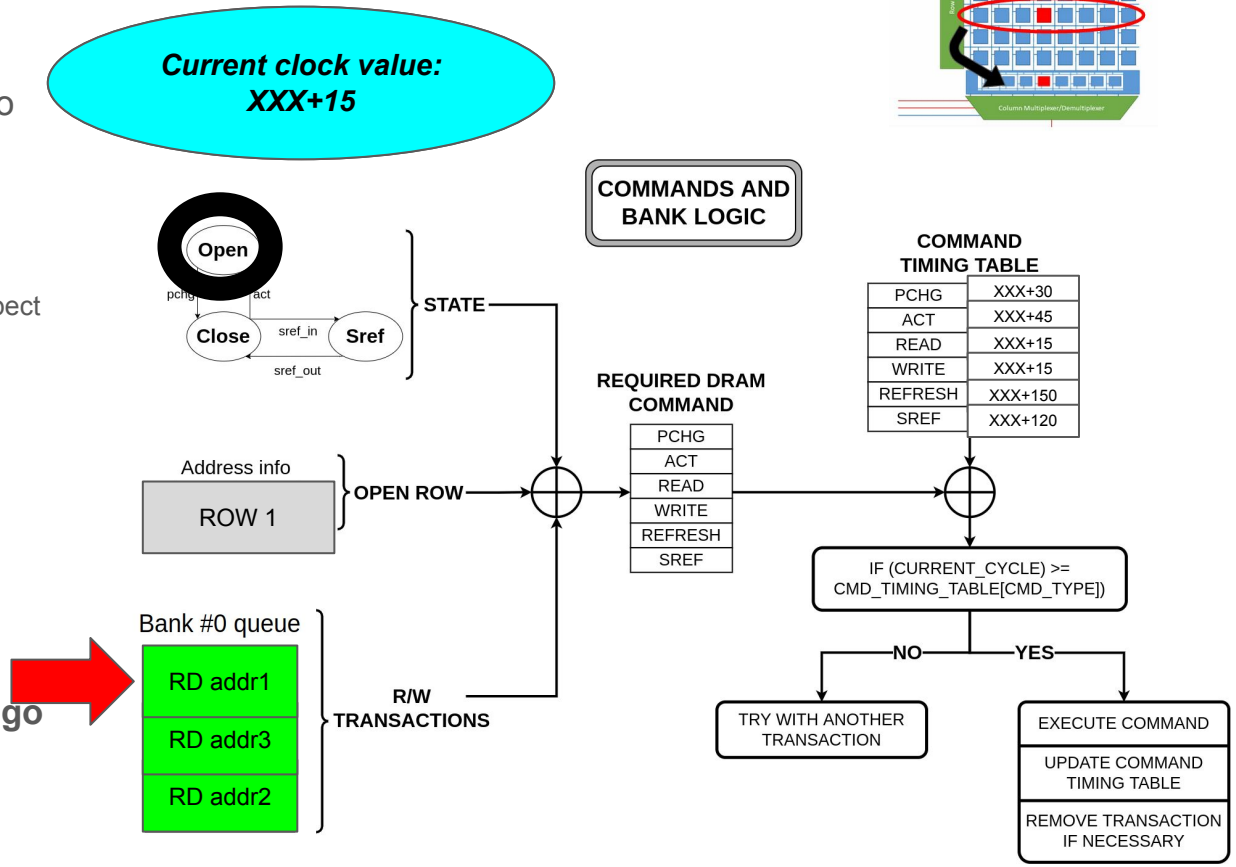
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

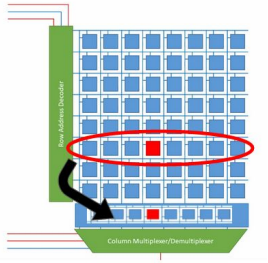
"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

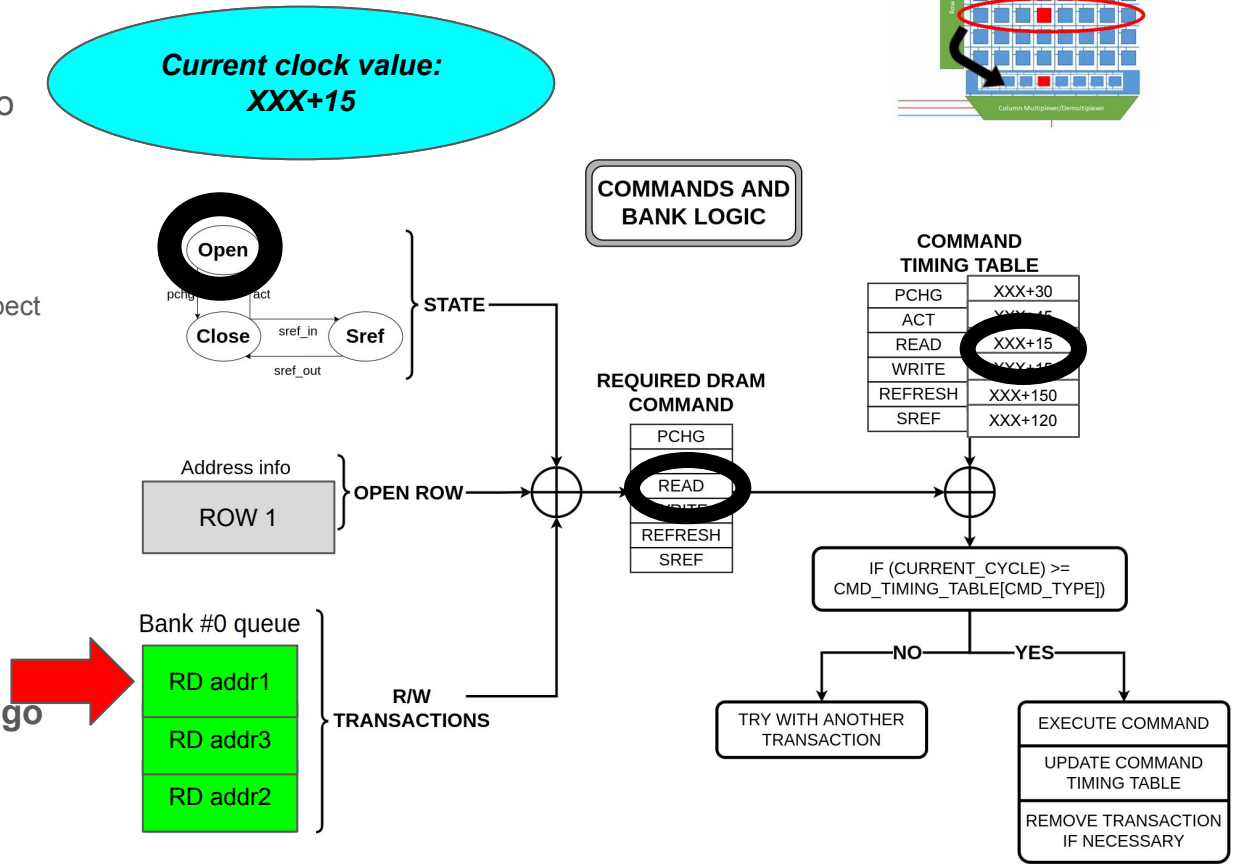
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

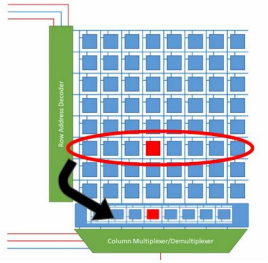
"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller



Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

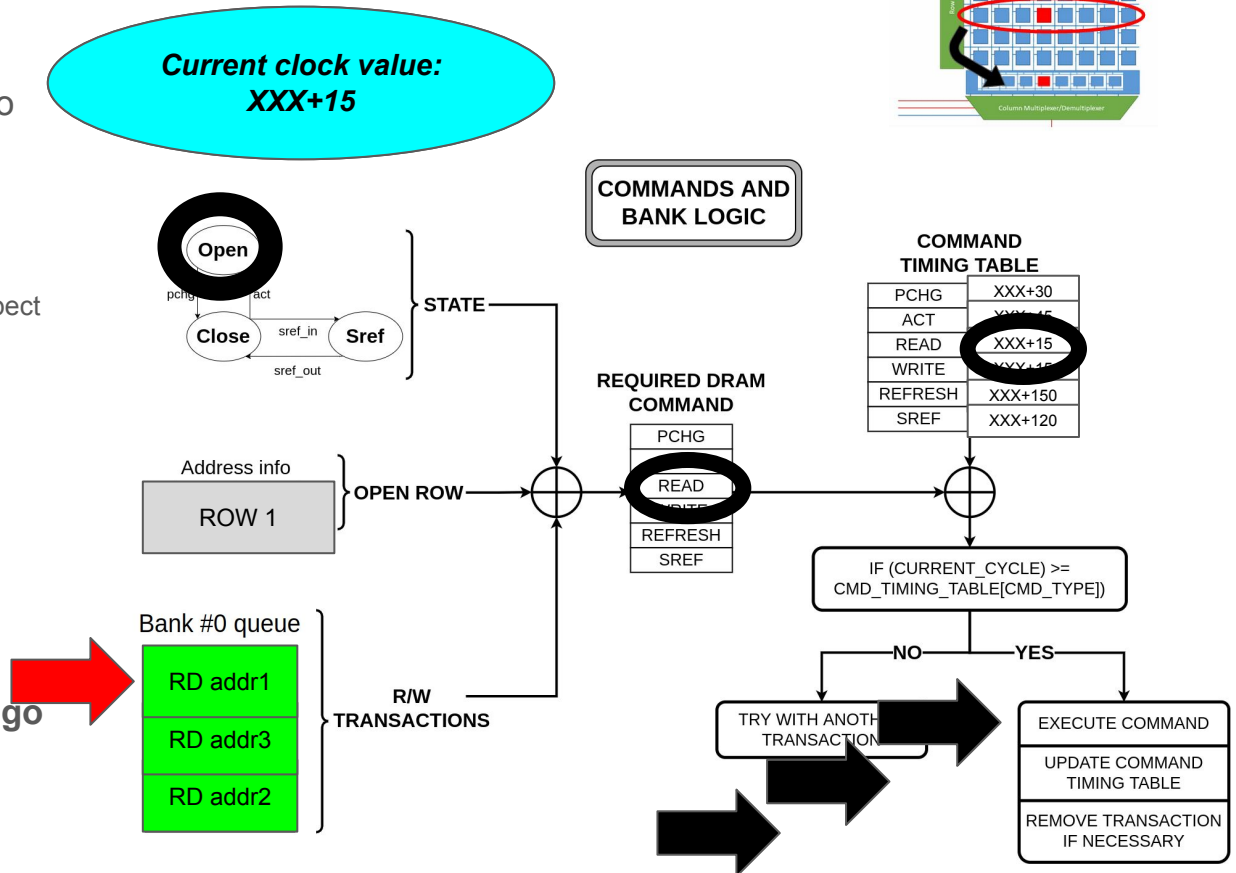
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller

Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

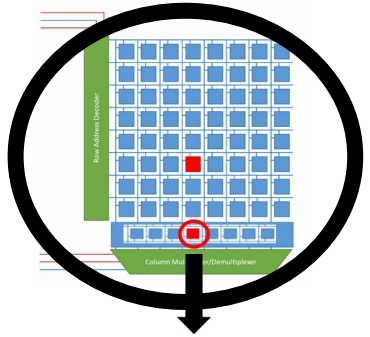
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller

Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

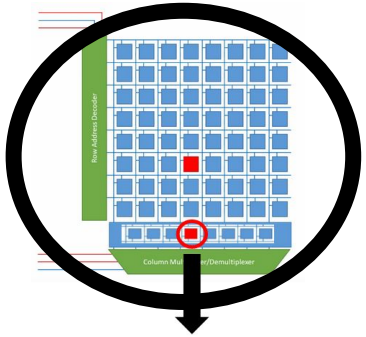
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller

Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

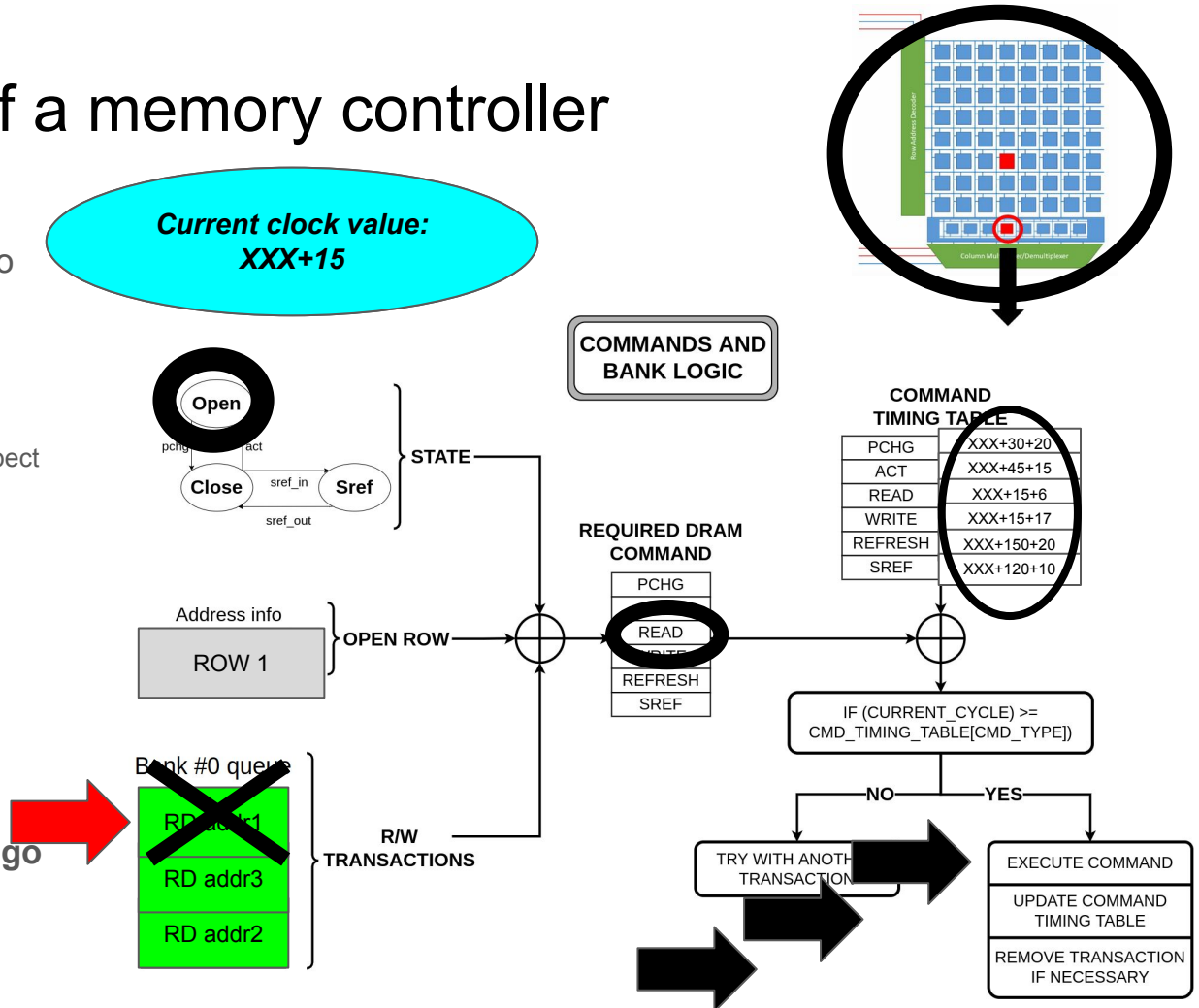
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

"RD addr1" needs an ACT:

- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller

Let's assume an idle (**CLOSE**d) bank, so nothing in the row buffer…

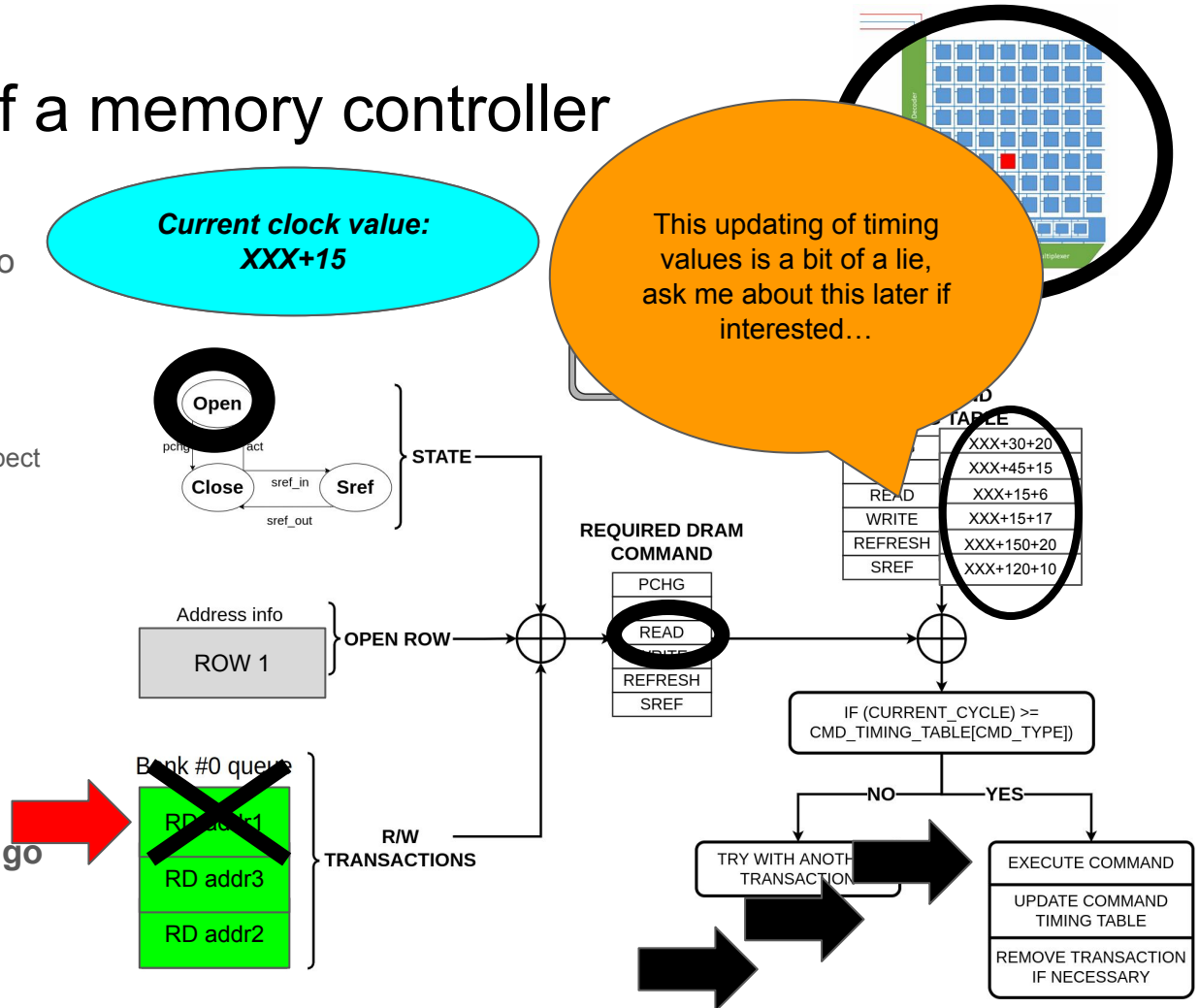
Let's follow a FR-FCFS policy:

- Try to issue first DRAM commands that respect the timing constraints…
- …from oldest transaction to most recent

"RD addr1" needs an ACT:

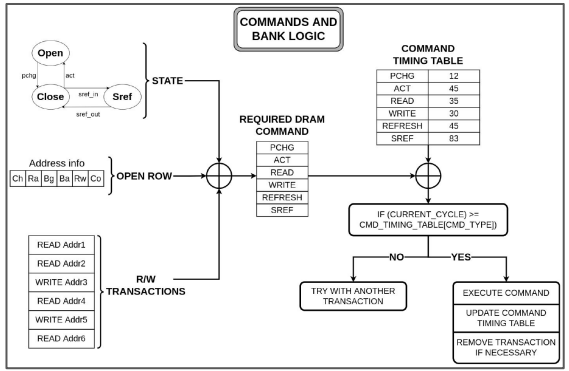
- Consider the state of the bank
- Consider which row is currently opened
- Consider what row the "current" transaction needs

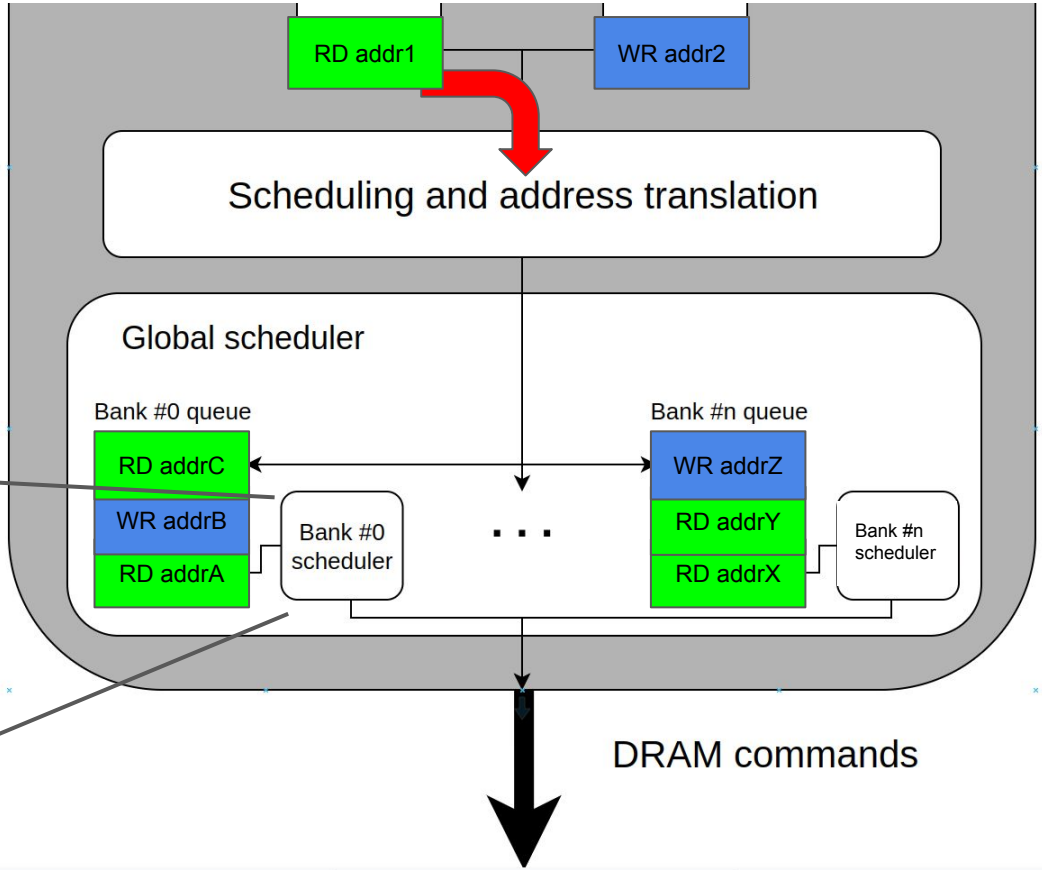
Does the execution of this command go against previous timing constraints?



Structure (logical) of a memory controller

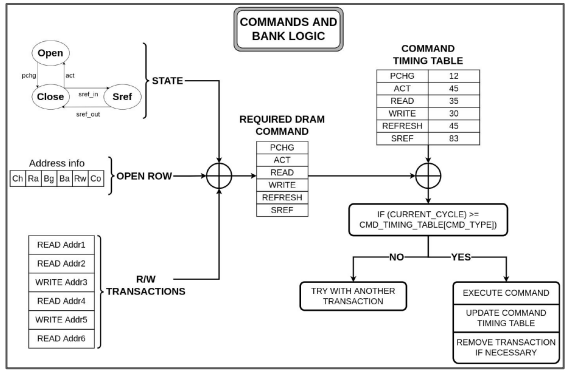
Okay, one last thing…

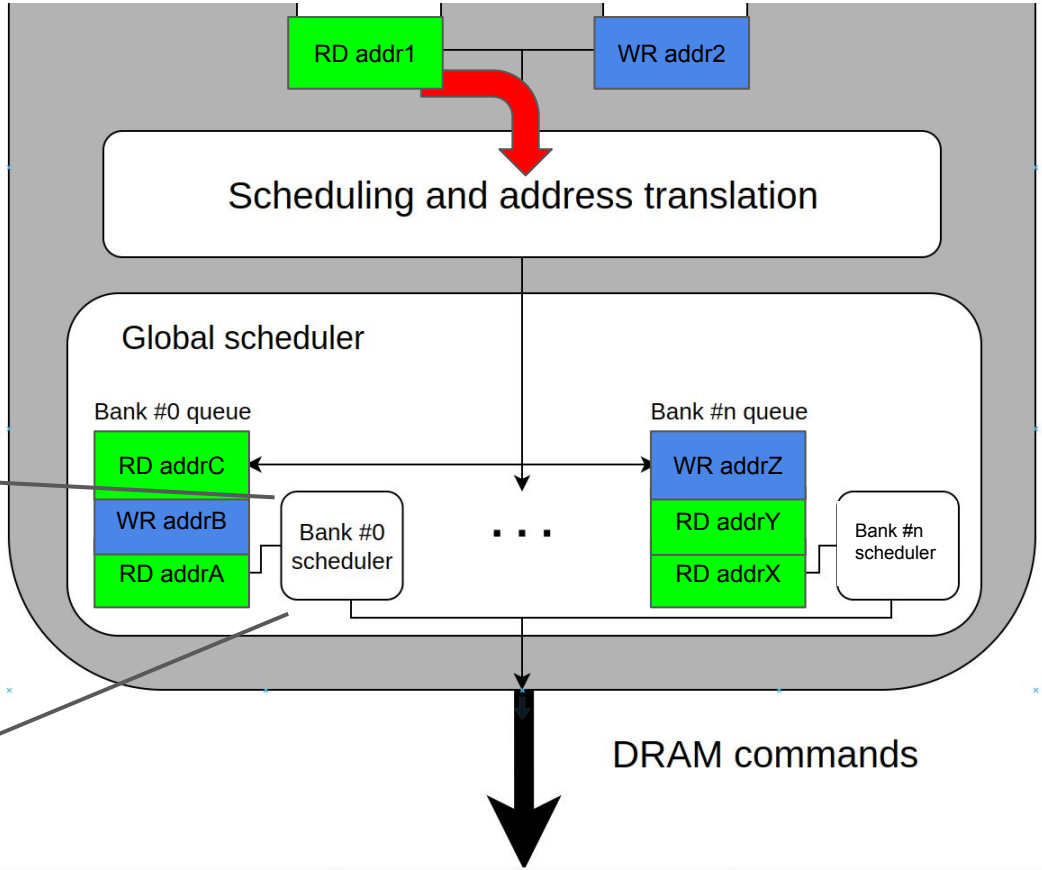




Structure (logical) of a memory controller

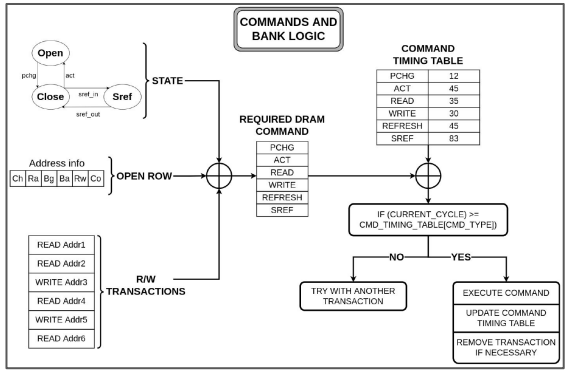
All bank schedulers generate DRAM commands to finish their transactions in FR-FCFS fashion

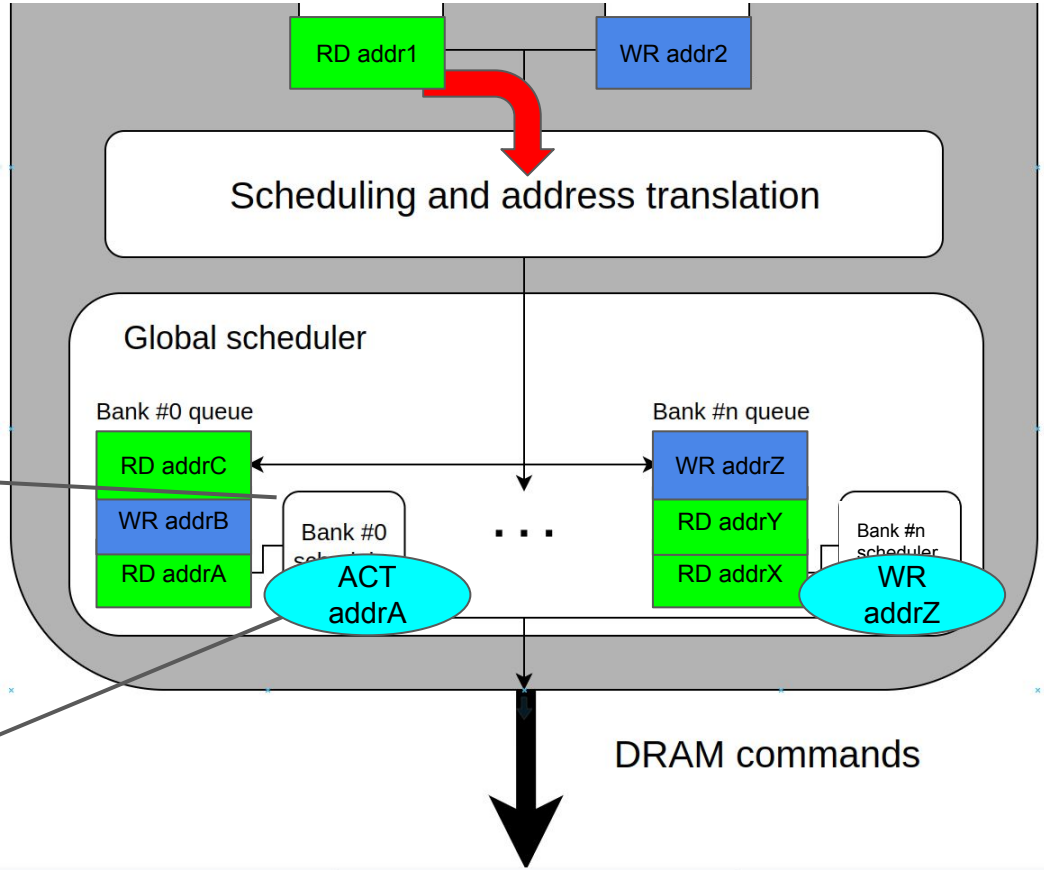




Structure (logical) of a memory controller

All bank schedulers generate DRAM commands to finish their transactions in FR-FCFS fashion

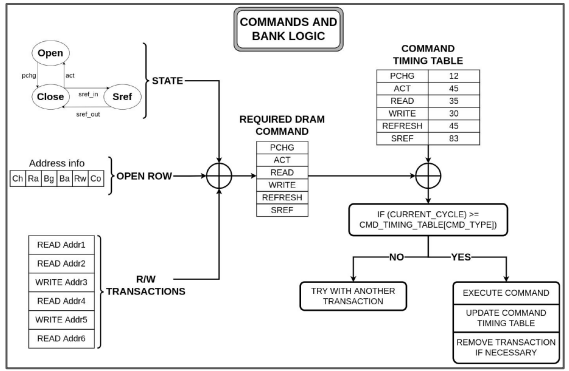


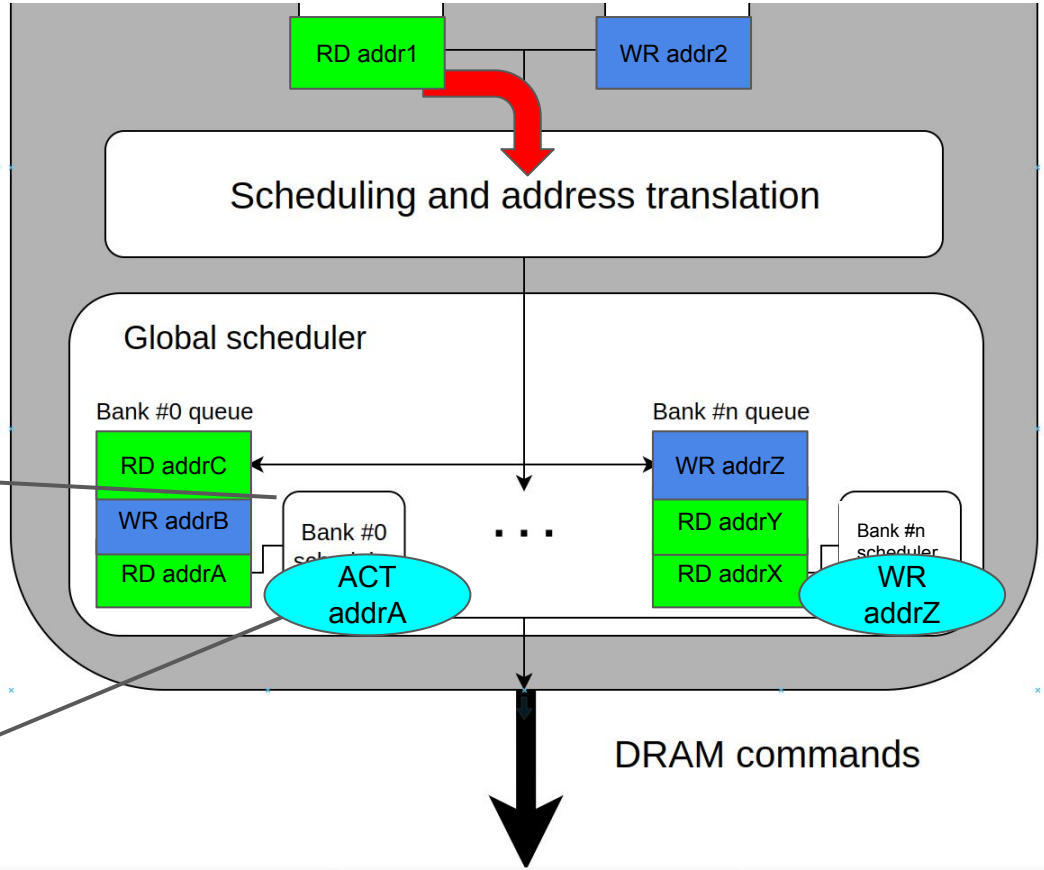


Structure (logical) of a memory controller

All bank schedulers generate DRAM commands to finish their transactions in FR-FCFS fashion

However, only one DRAM command can be issued at a time, so bank schedulers are scheduled in Round-Robin fashion

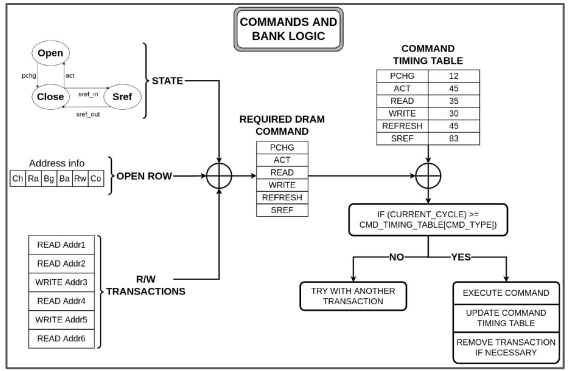


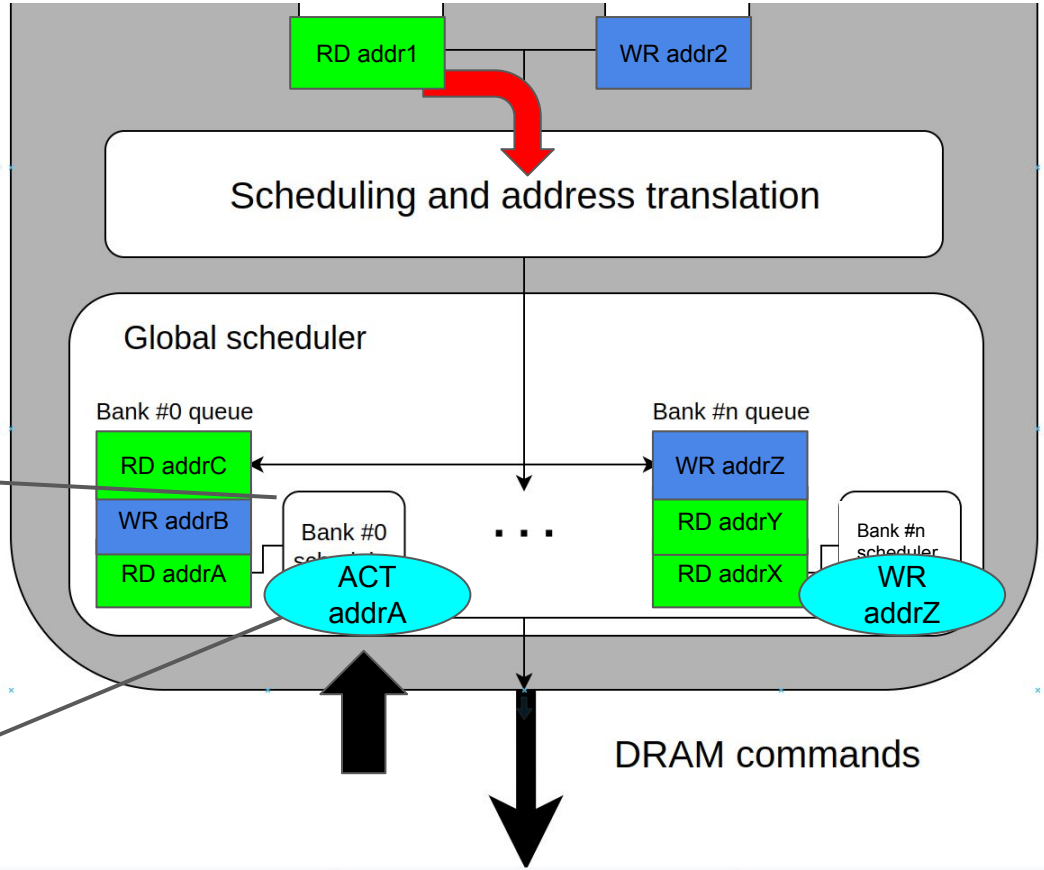


Structure (logical) of a memory controller

All bank schedulers generate DRAM commands to finish their transactions in FR-FCFS fashion

However, only one DRAM command can be issued at a time, so bank schedulers are scheduled in Round-Robin fashion

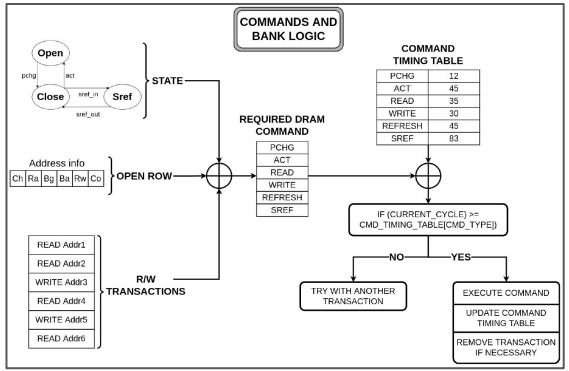


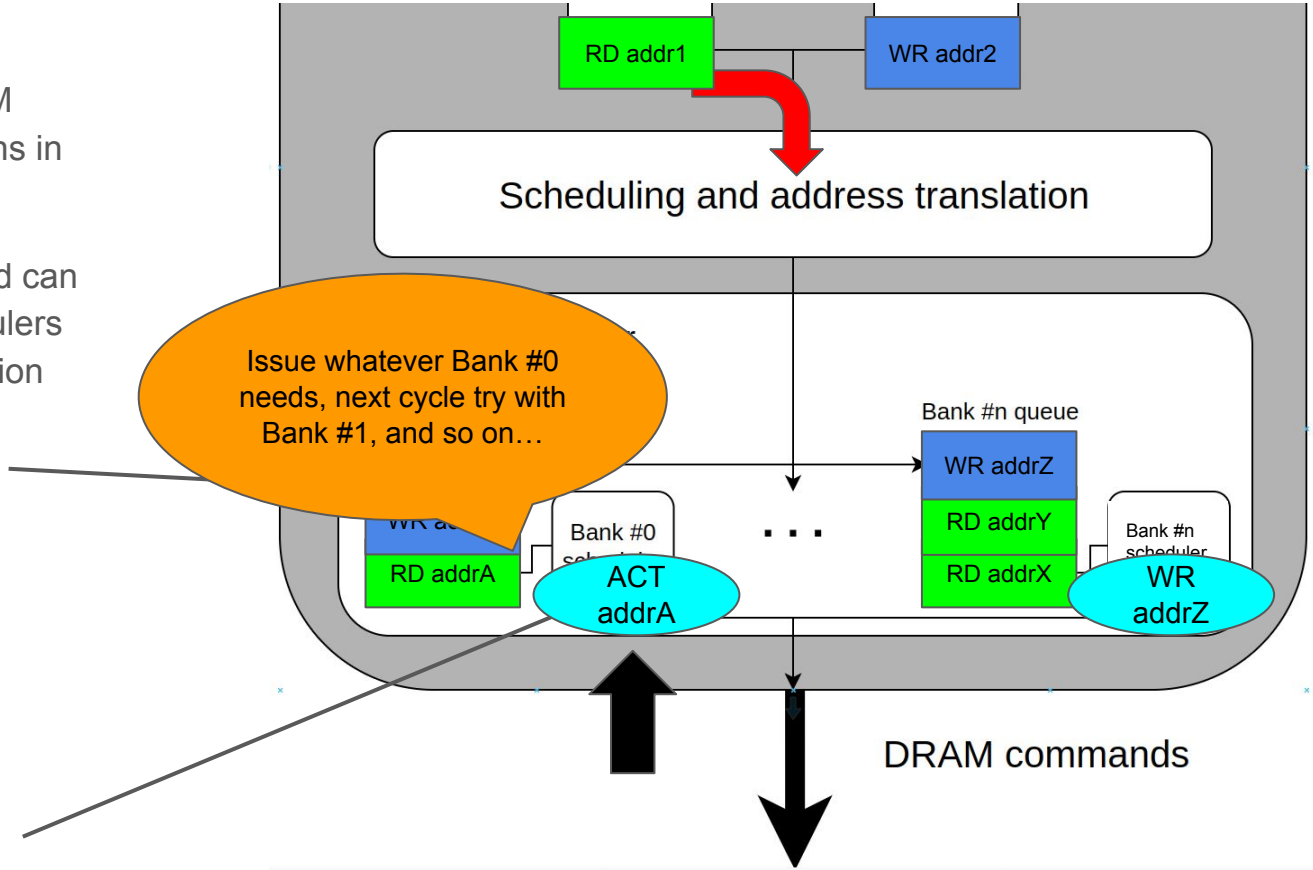


Structure (logical) of a memory controller

All bank schedulers generate DRAM commands to finish their transactions in FR-FCFS fashion

However, only one DRAM command can be issued at a time, so bank schedulers are scheduled in Round-Robin fashion

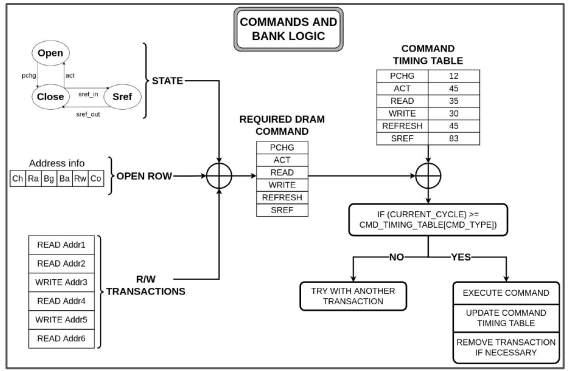




Structure (logical) of a memory controller

All bank schedulers generate DRAM commands to finish their transactions in FR-FCFS fashion

However, only one DRAM command can be issued at a time, so bank schedulers are scheduled in Round-Robin fashion



If no commands can be issued due to timing constraints, try with n+1 and next cycle try with with n+2 and so on…

Issue whatever Bank #0 needs, next cycle try with Bank #1, and so on…

Bank #n queue

WR addrZ

RD addrY

RD addrX

RD addrA

Bank #0

Bank #n scheduler

ACT addrA

WR addrZ

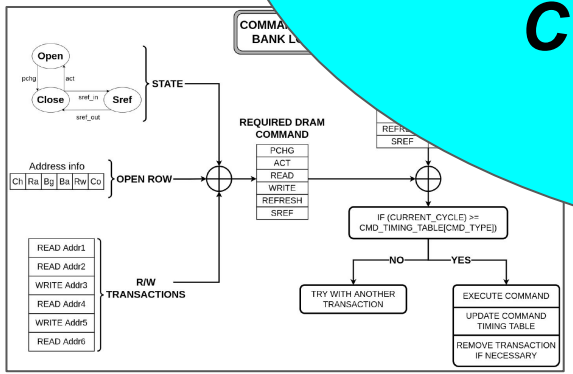
DRAM commands

Structure (logical) of a memory controller

All bank schedulers g
commands to finis
FR-FCFS fashio

However, onl
be issued at
are schedule

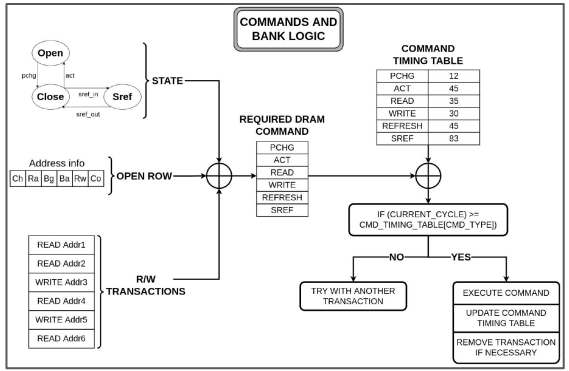
ALSO, THE EXECUTION OF ONE COMMAND OF BANK QUEUE X IMPLIES THE UPDATING OF THE REST OF THE BANK QUEUES' TIMING CONSTRAINTS AS WELL, NOT JUST ITS OWN…

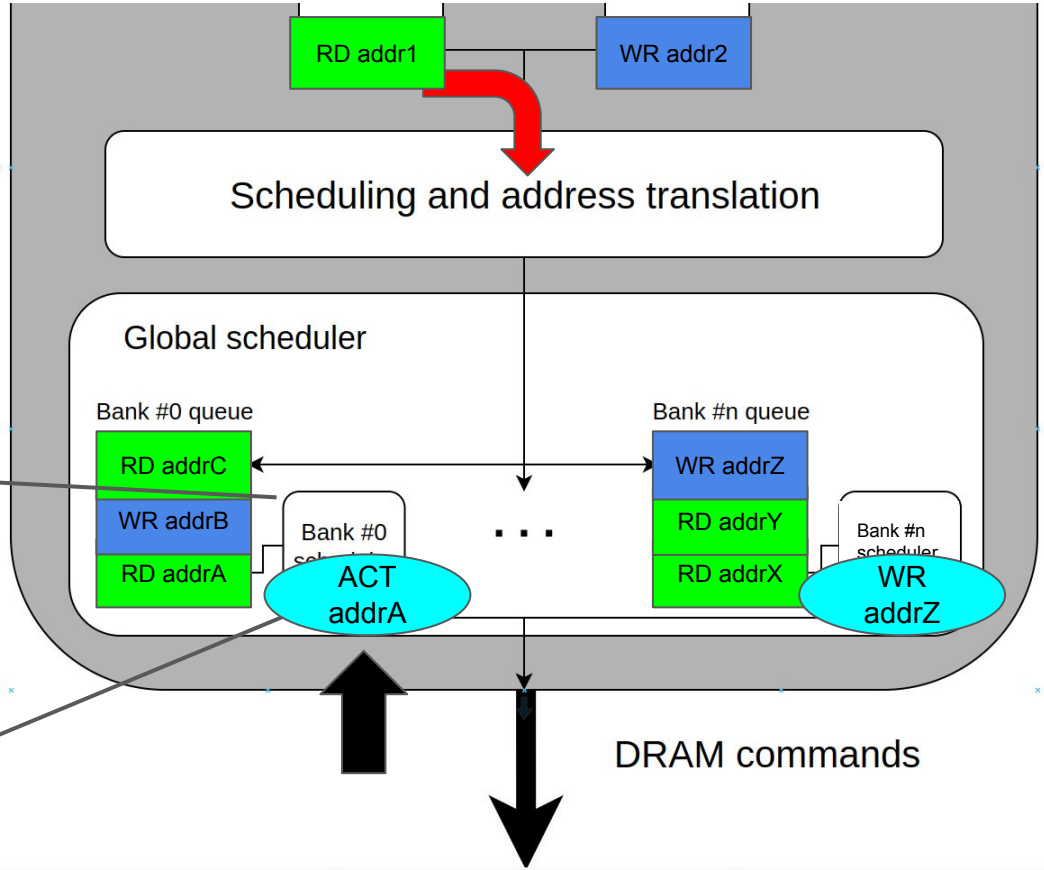


Structure (logical) of a memory controller

All bank schedulers generate DRAM commands to finish their transactions in FR-FCFS fashion

However, only one DRAM command can be issued at a time, so bank schedulers are scheduled in Round-Robin fashion

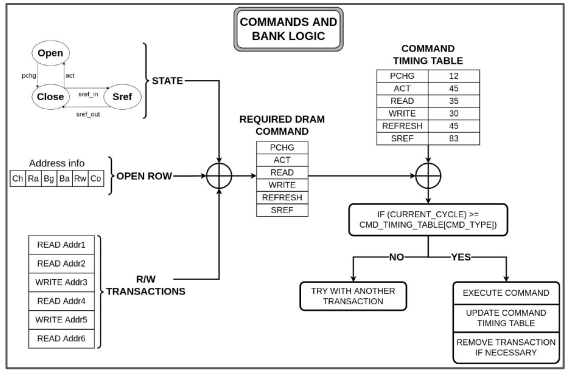


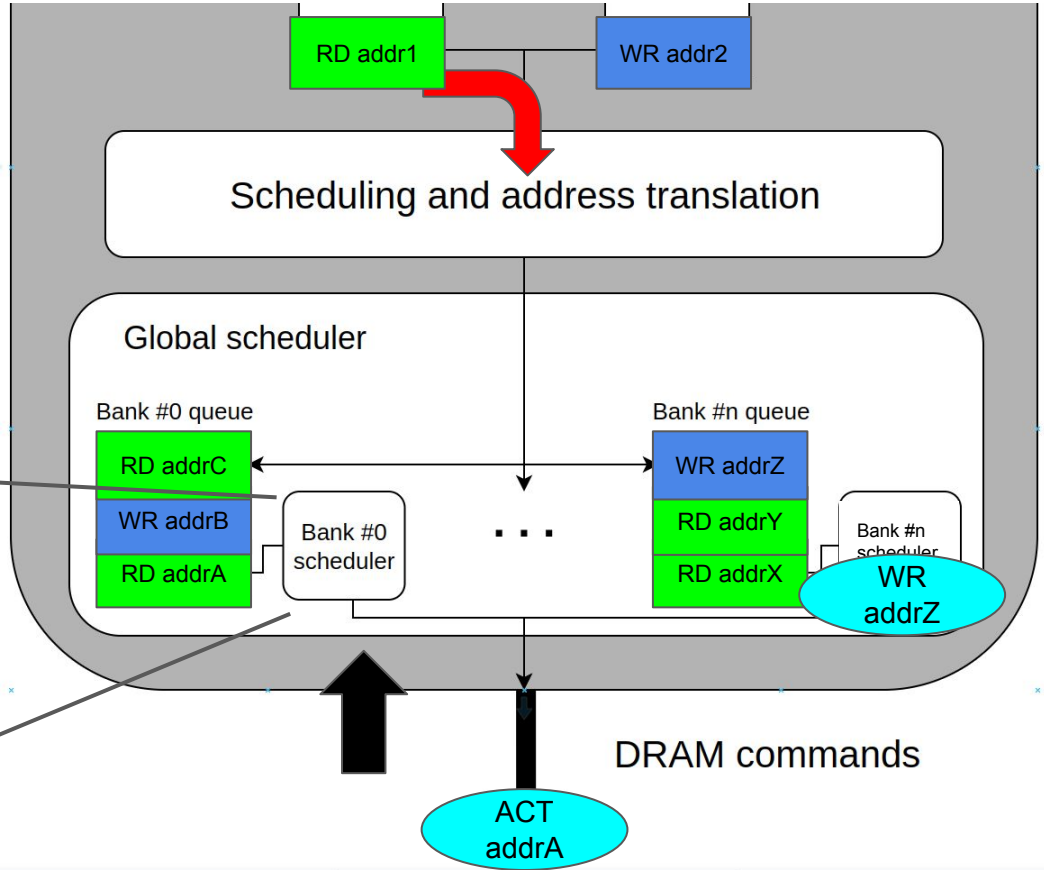


Structure (logical) of a memory controller

All bank schedulers generate DRAM commands to finish their transactions in FR-FCFS fashion

However, only one DRAM command can be issued at a time, so bank schedulers are scheduled in Round-Robin fashion

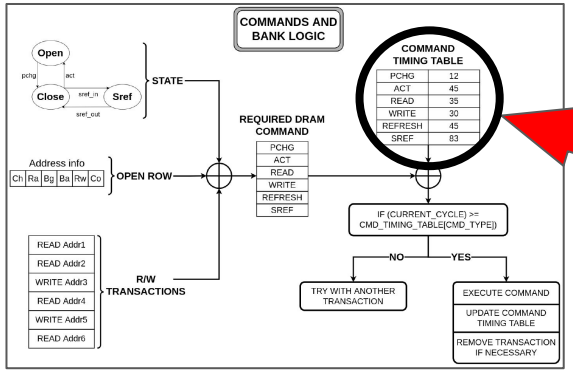


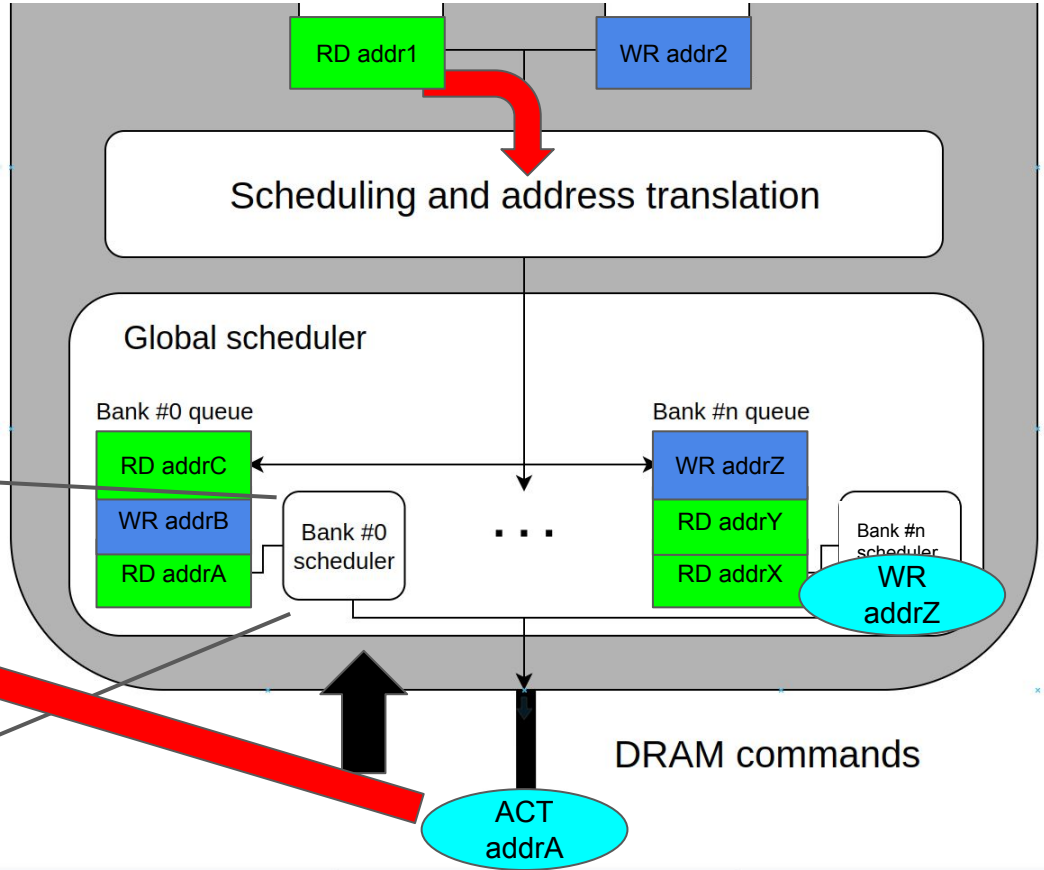


Structure (logical) of a memory controller

All bank schedulers generate DRAM commands to finish their transactions in FR-FCFS fashion

However, only one DRAM command can be issued at a time, so bank schedulers are scheduled in Round-Robin fashion

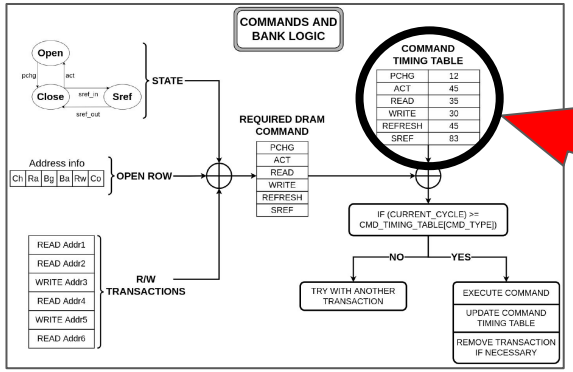


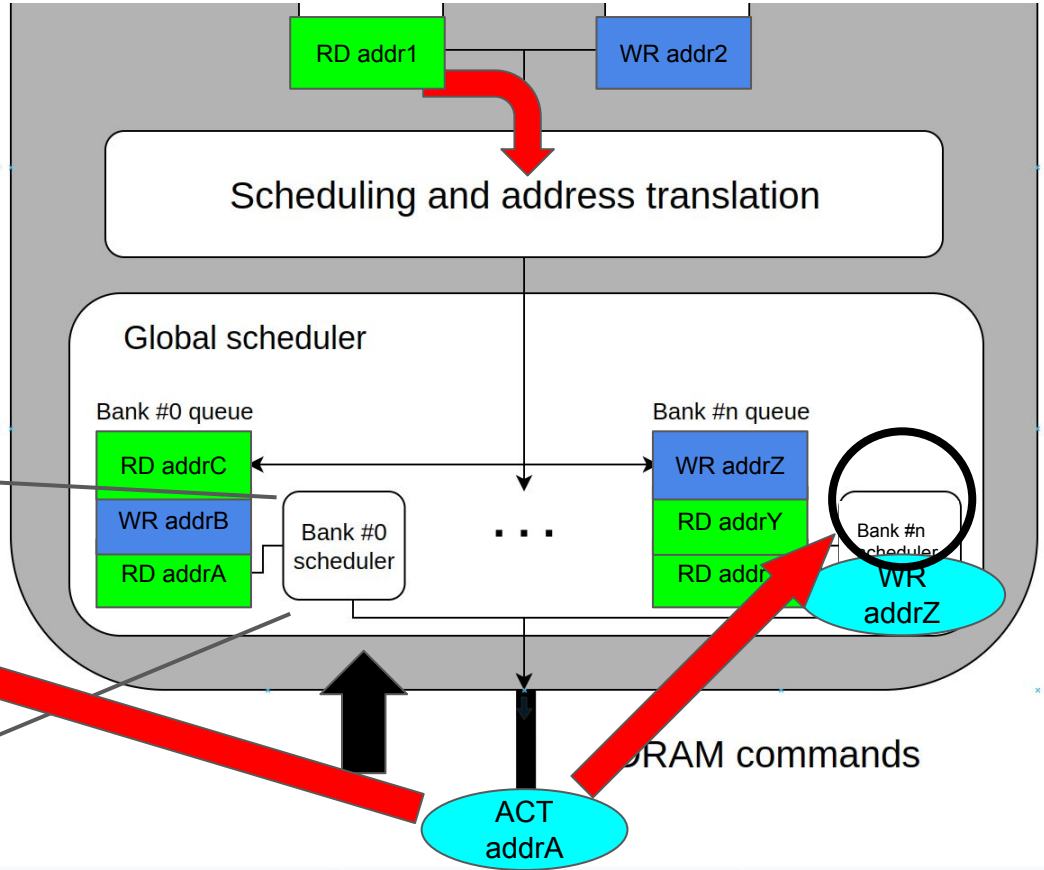


Structure (logical) of a memory controller

All bank schedulers generate DRAM commands to finish their transactions in FR-FCFS fashion

However, only one DRAM command can be issued at a time, so bank schedulers are scheduled in Round-Robin fashion

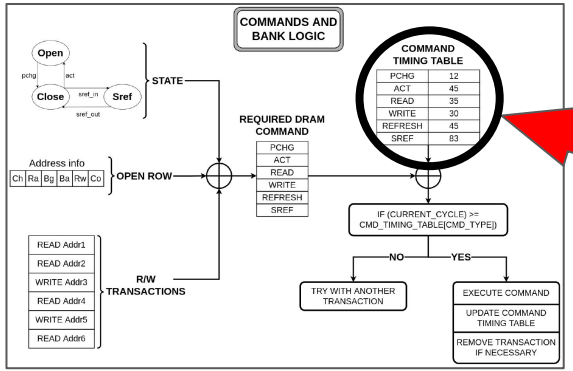


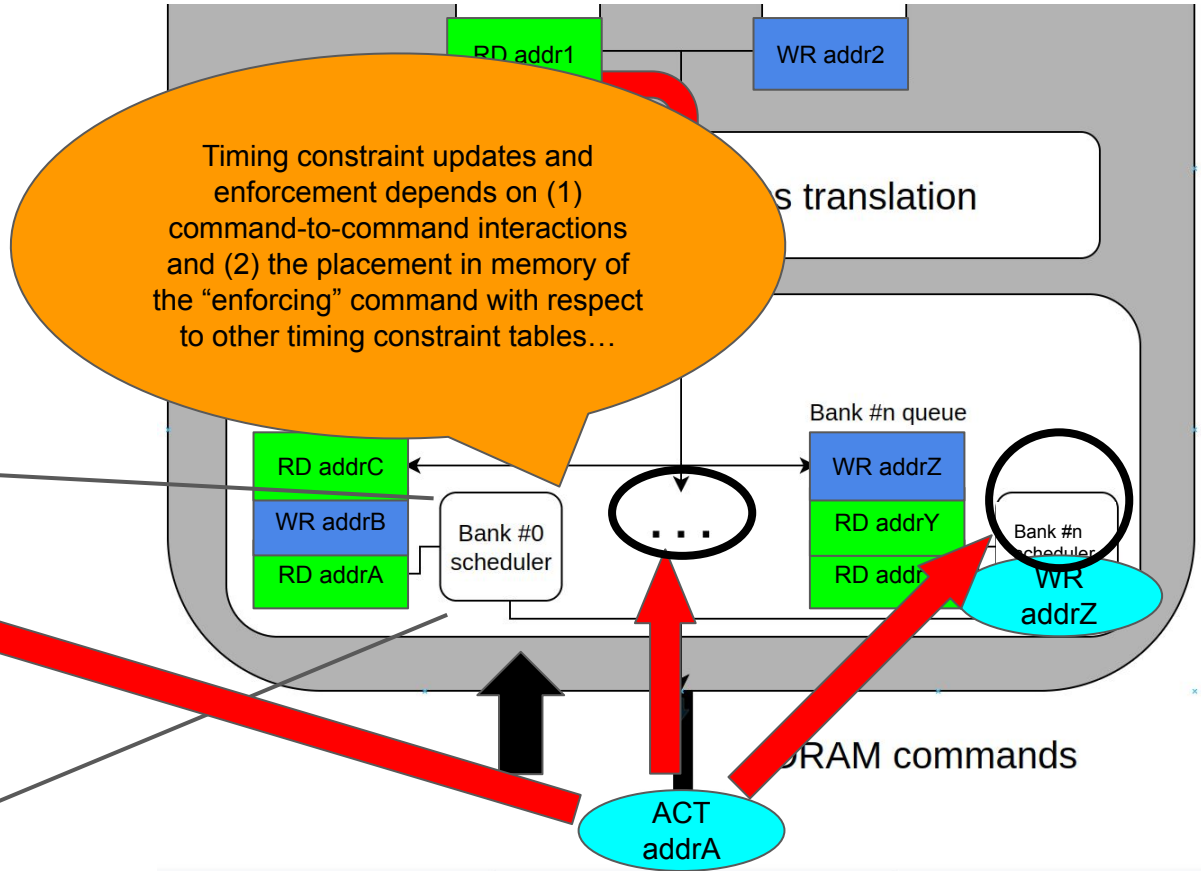


Structure (logical) of a memory controller

All bank schedulers generate DRAM commands to finish their transactions in FR-FCFS fashion

However, only one DRAM command can be issued at a time, so bank schedulers are scheduled in Round-Robin fashion

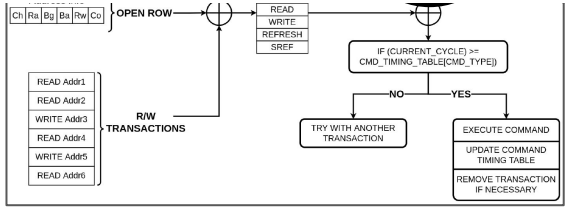


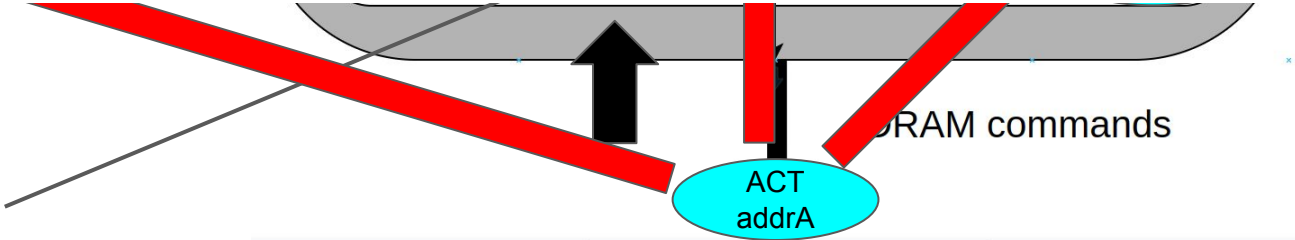


Structure (logical) of a memory controller

		Read	Write
Read	sb	tCCD_L: 6	RL+tBURST-WL+tRTRS: 9
	sbg	tCCD_S: 4	RL+tBURST-WL+tRTRS: 9
	sr	tBURST: 4	RL+tBURST-WL+tRTRS: 9
	sch	tBURST+tRTRS: 5	read_delay+tBURST+tRTRS-write_delay: 9
Write	sb	write_delay+tWTR_L: 23	tCCD_L: 6
	sbg	write_delay+tWTR_L: 23	tCCD_S: 4
	sr	write_delay+tWTR_S: 18	tBURST: 4
	sch	write_delay+tBURST+tRTRS-read_delay: 1	tBURST: 4

Table 5: Expected timing constraints for ever read/write combination for all two-core placements in memory.

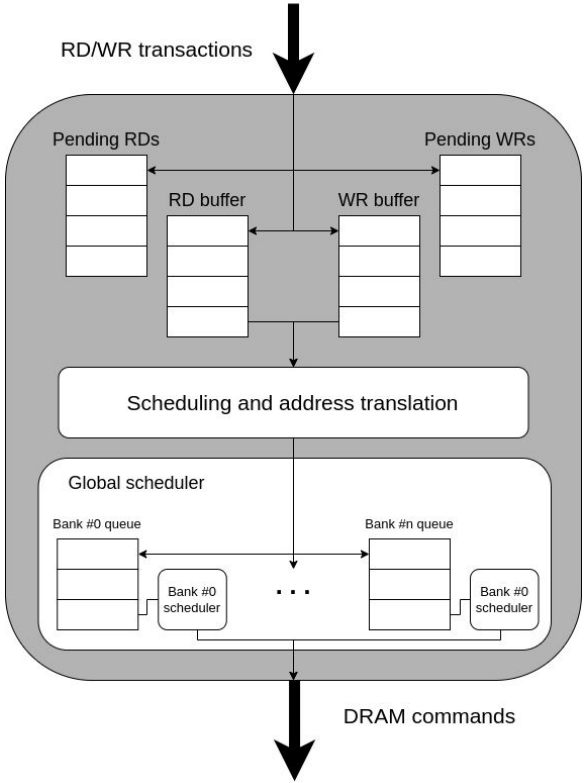




Summary

Address mapping
FR-FCFS
Round-Robin
TIming updates

DRAM command generation



Transaction-level management

Pending transactions
Read-your-writes
Transaction scheduling
Write batching

Other interesting, undiscussed MC topics

Refreshing policies
Page policies (open, close, adaptable)
Precharge arbitration
Four-activation window
ZQ calibration
Priorities and criticalities
Bank partitioning and/or isolation
Predictable MCs for high criticality systems
…

Sources of figures

If a figure has no identifying reference or caption, this means it is either original or pulled from a confidential paper obtained for conference review.

Programmable controller:
https://www.cs.utah.edu/~bojnordi/data/tocs13.pdf

DRAM blocks:

- https://www.youtube.com/watch?v=Mhqi70OPW0o
- https://course.ece.cmu.edu/~ece740/f13/lib/exe/fetch.php?media=onur-740-fall13-module3.5-main-memory-part1-version1.pdf

DRAM commands interaction diagram:
https://uwspace.uwaterloo.ca/bitstream/handle/10012/11138/Guo_Danlu.pdf

MC figures (from Bruce Jacob's book "Memory Systems: Cache, DRAM, Disk")

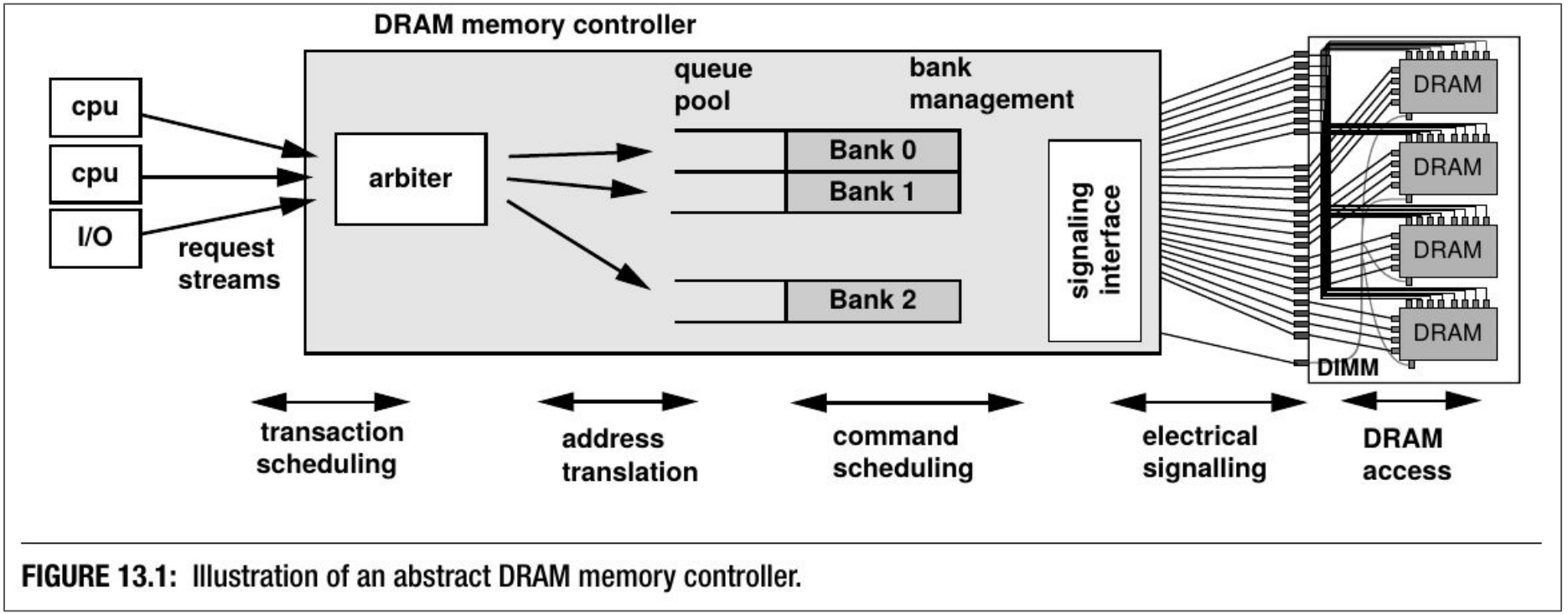


FIGURE 13.1: Illustration of an abstract DRAM memory controller.