ASN 1 – X – Plain - Did

yea catch that pun

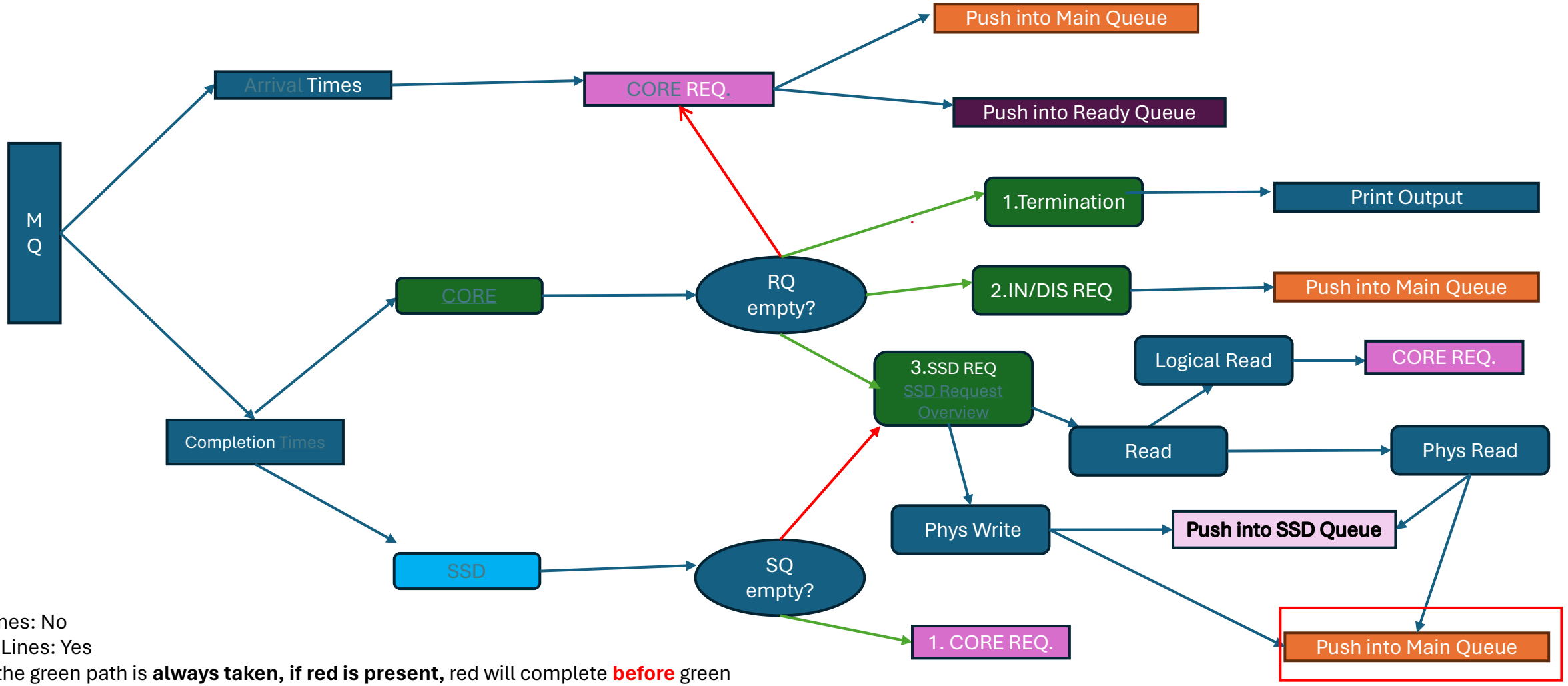
Sooo before we begin

Dis-claimer – The following content *could* be **misleading** and has the *possibility* of being …..**wrong.** Please be confident in **your approach** and use this mainly as **advice**, I want to help but use with your own discernment. Most of this advice comes from the slides itself that Dr. Paris gave.

If helpful great!

If not or confusing, no worries please find seek other resources to help. All love.

Overview (click on each for reference)



Red Lines: No
Green Lines: Yes
Note: the green path is **always taken, if red is present,** red will complete **before** green

Correction: Each Process Has their Own Buffer

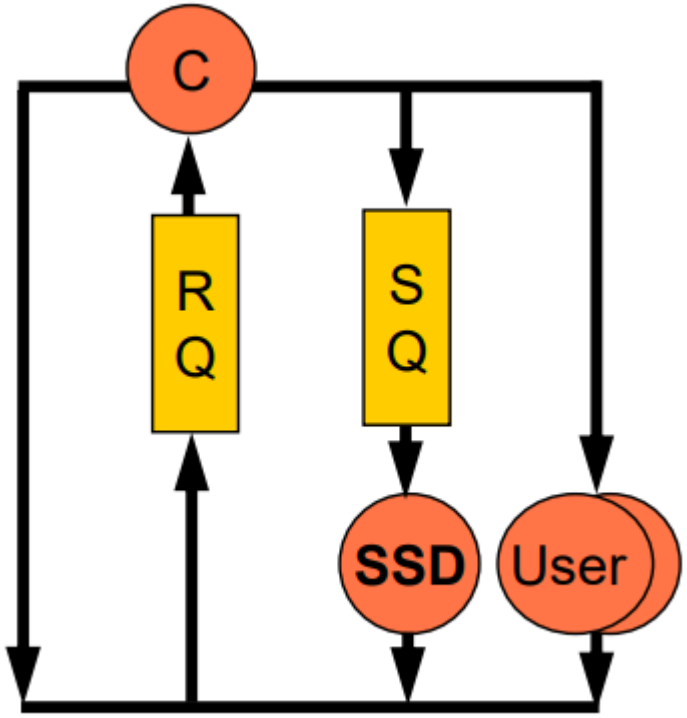
- Follow the links:

- Code for the function to calculate
- Final explanation

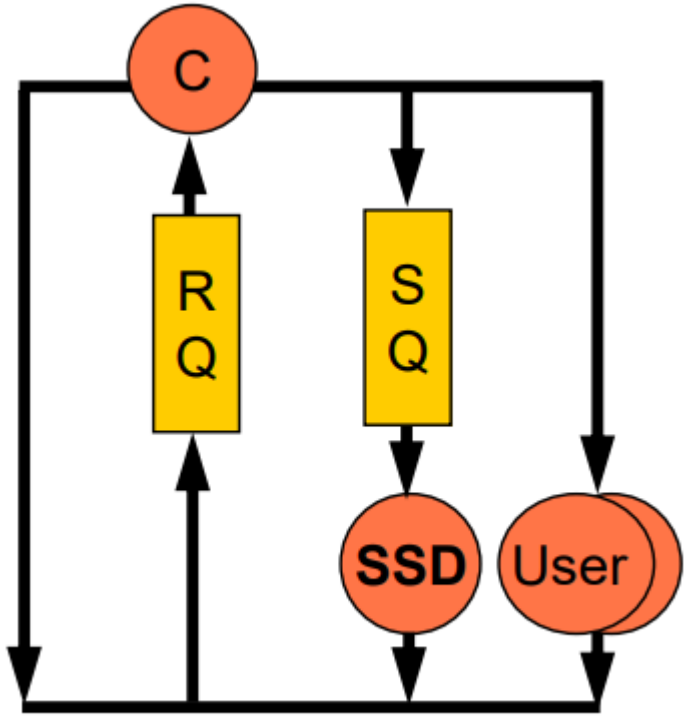
What is the point of this assignment

- To hate yourself ? Maybe
- But its not as bad as it looks!
- Just break it down and focus on one element at a time.

Let's revisit a simple picture:



Let's revisit a simple picture:



Where is the CPU ? → C

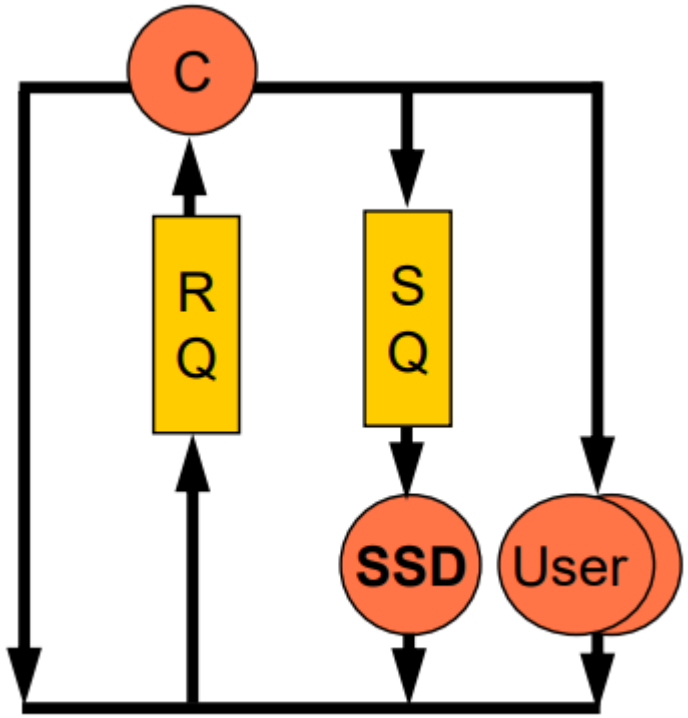
Where is the SSD → SSD

Where is the ready queue → RQ

Where is the SSD queue → SQ

Where is the input/display → User

Where is the **scheduler** ?.... → Not here
....why not....



We will make the main scheduler **in our program!**

How? Think of using a way to schedule things in order of **time!**

We can do this using a **priority queue:**

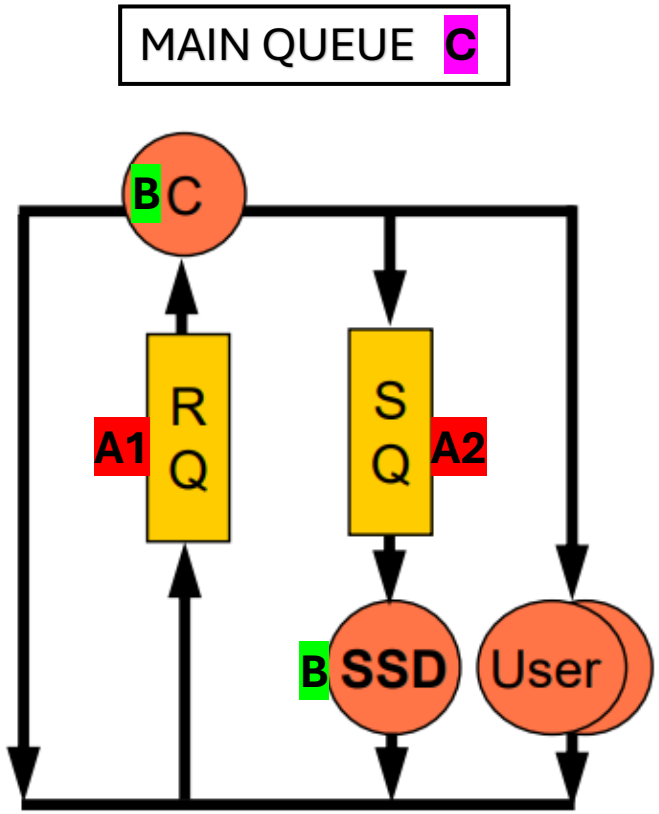
Priority	**1**	**2**	**3**	**4**
Process Time	P3 Time: 300	P1 Time:400	P2 Time: 450	P4 Time: 500

Here:

1 → Means lowest time (highest priority)

Why? The ones with the **least time** need to be served **first!**

Define Suggested Data Structures



A → Normal Queues (A1,A2) of **Processes**
1. SQ Queue
2. RQ Queue

B → Boolean Values (Is Empty)
1. CpuisEmpty
2. SSDisEmpty

C → Priority Queue of **Processes**

Define a process using struct:

struct **Processes** {
1. PID (process id num)
2. Time Completion/Arrival Time (float)
3. Instruction
4. Num Log Reads
5. Num Phy Writes
6. Num Phy Reads
7. **Buffer Size (initially 0)**

}

Speaking on that buffer....

```
if cur_command == "WRITE":
    if is_SSD_empty:
        is_SSD_empty = False
        current.time += 0.1
        current.num_phy_writes += 1
        pq.push(current)
    else:
        SQ.push(current)
elif cur_command == "READ":
    if value > current.buffer_size:
        if is_SSD_empty:
            current.buffer_size = handleBuffer(value, current.buffer_size, BSIZE
            is_SSD_empty = False
            current.time += 0.1
            current.num_phy_reads += 1
            pq.push(current)
        else:
            SQ.push(current)
    else:
        current.buffer_size = handleBuffer(value, current.buffer_size, BSIZE)
        table_needs.Process_Table[pid].curr_line += 1
        current.num_log_reads += 1
        CoreRequest(current)
```

Function to Calculate new Buffer size

```
int handleBuffer(int bytesNeeded, int curBSIZE, int BSIZE) {
  int finalBufferSize;

  if (bytesNeeded <= curBSIZE) {
              finalBufferSize = curBSIZE - bytesNeeded;
  }

  else {
    int bytesMissing = bytesNeeded - curBSIZE;
    int bytesBrought;
    if (bytesMissing % BSIZE == 0) {
      bytesBrought = bytesMissing;
    }

    else {
      bytesBrought = (bytesMissing / BSIZE + 1) * BSIZE;
    }

    finalBufferSize = bytesBrought - bytesMissing;
  }

  return finalBufferSize;
}
```

So what now??

Think about the variables that need to be global (there was nothing against it) → generally in programing this is a bad habit but since all functions will share these variables/data structures this will make our life so much easier!!

Definity want these global!

A → Normal Queues (A1,A2) of **Processes**
1. SQ Queue
2. RQ Queue

B → Boolean Values (Is Empty)
1. CpuisEmpty
2. SSDisEmpty

C → Priority Queue of **Processes**

And these (will talk about why later)
- Clock Time = 0
- Map

Now let's focus on the thing that will drive the program forward

It a “simple” while loop….

But really it is. Your main action is happening in the functions.

What is the concept behind the while loop. While there are process that need to get something done, we keep it going. In other words, while our main queue **is not empty**, we keep it going.

The Main is Mainly this…

1	2
Process 3 Time = 0 START	Process 1 Time = 50 CORE

While the main queue is not empty:

1. Read the top value and save it . → Saved_Top = [Process 3 ,0,Start]
2. Pop the top from the main queue
3. Set the clock time = time completion or arrival time → clock time = 0
4. Figure out what to do based on the command
   1. Arrival (Start) Event → this is the one we execute in this **example**
   2. Completion Event

MQ After pop →

1
PROCESS 1 TIME = 50 CORE

So, what do we put into our MainQueue?

Process(es) with only either its **completion times** or **arrival (start)**

Where do completion times come from? **After** the while loop starts

- Note: Completion Times are associated with operations: CORE, READ, WRITE, INPUT, DISPLAY

Where do arrival times come from?? **Before** the while loop starts

- Note: Arrival Times are associated with only one operation: START

How to read in input to create this mainqueue **before** the while loop starts using an Example (Input 12.txt)

First read in the input via input redirection (make sure you use WSL Ubuntu BASH (if not replit works to via shell) both will work!) in the main.

```
#include <iostream>
#include <vector>
using namespace std;
struct input{
 string command;
  int time;
};


int main()
{
    vector <input> inputs;
```

Suggestion use a vector of type **input**.

```
BSIZE 4096
START  0
CORE   200
WRITE 4096
CORE   30
START  100
CORE   80
WRITE 4096
CORE   40
```

struct input :

- command
- time

Note: At this point we should know how many process we have, we have two in the example above

Now we need to create a process table….

Why this will allow us to be organized in our program!

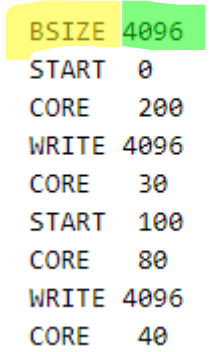
PID	Start Line	End Line	Current Line	State
0				1
1				2
k				2

This will always have 5 columns, with **k** (process amount) number of rows!
Therefore, we can always use a list of lists of integers (or vector of vectors)

Note our state: Can only be numbers! So, use a map to define a number mapped to a state:
1 → “Running”
2→ “Ready”

Global Variable

Use the input table to fill the process table



```
BSIZE 4096
START  0
CORE   200
WRITE 4096
CORE   30
START  100
CORE   80
WRITE 4096
CORE   40
```

Index	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

PID	Start Line	End Line	Current Line	State
0	1	4	-1	0
1	5	8	-1	0

Here:
- Current Line → -1 means has not started
- State → 0 means has not started

For indexes 0,3,7 those are **not** time values. We are just using the structure to hold the value of what the command has (READ,WRITES,BSIZE)

Now we have the input: We can create the mainQueue

The order is now we use the process table → to get the index from the input table → to get the commands/times (or values)

PID	Start Line	End Line	Current Line	State
0	1	4	-1	0
1	5	8	-1	0

Index	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

For each **PID**, add the start line index command and time to the main queue:

Main Queue	0	1
MQ	Process 0 Time = 0 START Log reads = 0 Phys writes =0 Phys reads =0	Process 1 Time = 100 START Log reads = 0 Phys writes =0 Phys reads =0

Now start that while loop up

Main Queue	0	1
MQ	Process 0 Time = 0 START Log reads = 0 Phys writes =0 Phys reads =0	Process 1 Time = 100 START Log reads = 0 Phys writes =0 Phys reads =0

While MainQueue is not empty:

1. Save the top

0
Process 0 Time = 0 START Log reads = 0 Phys writes =0 Phys reads =0

2. Pop the top

(MQ after Pop)

Main Queue	1
MQ	Process 1 Time = 100 START Log reads = 0 Phys writes =0 Phys reads =0

3. Save top time to **clock time**

Clock Time = 0

4. Figure out the command

START

Understanding the functions of the main

While the main queue is not empty:

1. Read the top value and save it .
2. Pop the top from the main queue
3. Set the clock time = time completion or arrival time
4. Figure out what to do based on the command

if (command == “START”) → it **must** be an **arrival function**

else: → it **must** be a **completion function** event

Arrival Function

For this function we need to pass a couple of things as **parameters** minimum :

1. The process (as a struct with its values respectively)
2. the process table
3. the input table
4. (all the other variables are global for us)

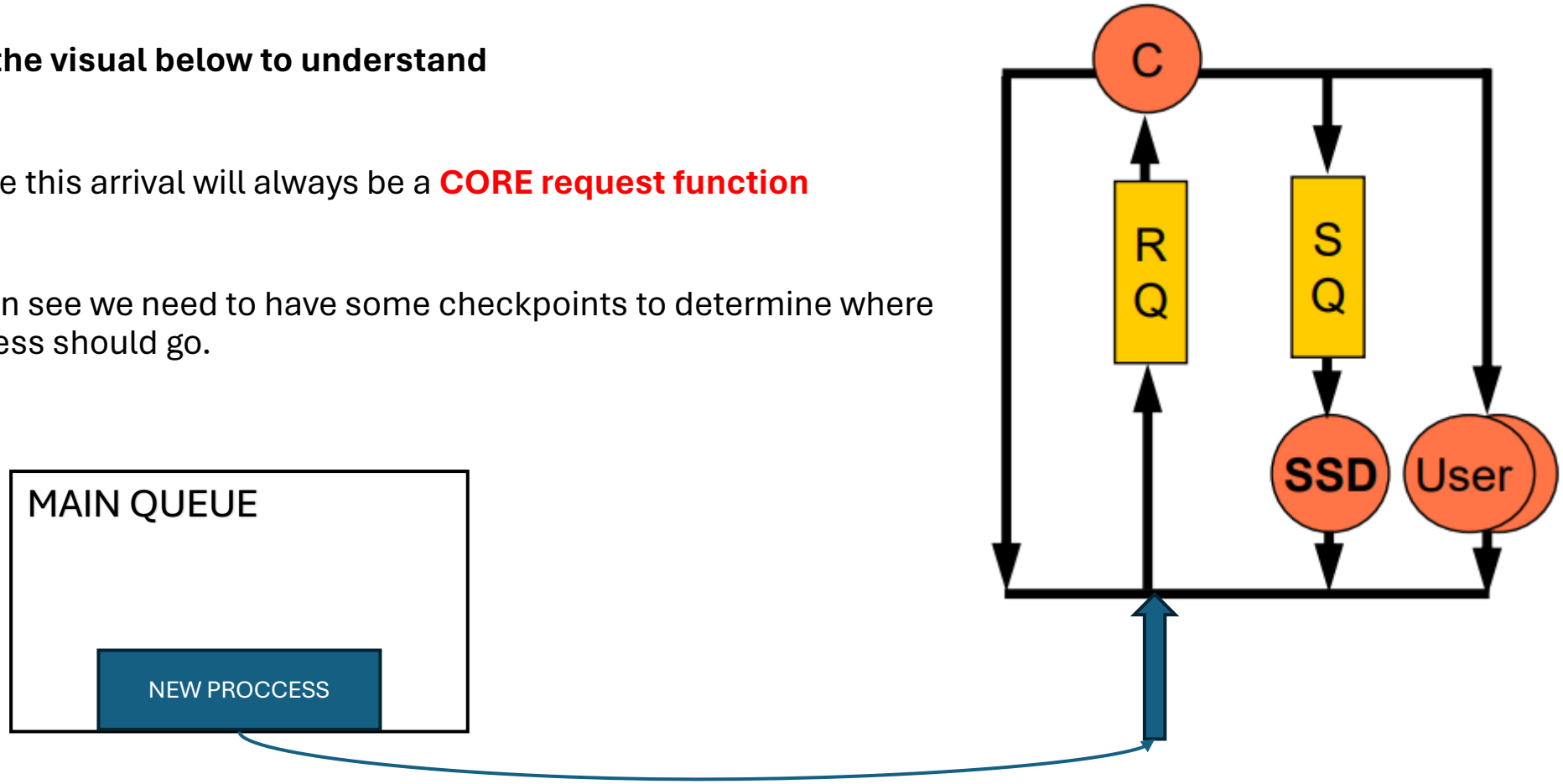
Arrival Cont.

We need to think the arrival of a new process will **always** go to the **core.**

Look at the visual below to understand

There fore this arrival will always be a **CORE request function**

So we can see we need to have some checkpoints to determine where the process should go.



ARRIVAL FUNCTION

Before we create the Core Request Function:

We need the arrival function to do something.

1. Change the process table current line **(Current Line = Start Line +1)**
2. Get the command from the input table (which will be core)

We send these **updated** parameters to the core Request:

1. Process Table (you'll want to pass this by **reference** everywhere you go, since you're updating in each function)
2. Process (change the command and time)

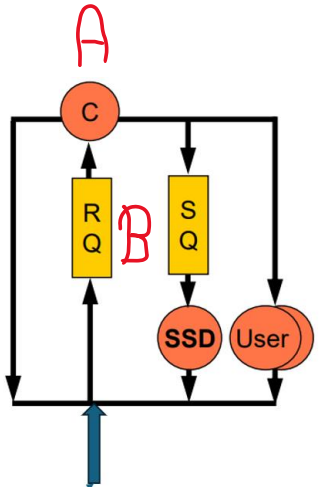
We send these unmodified parameters to the core Request:

1. Input Table (this will never change)

PID	Start Line	End Line	Current Line	State
0	1	4	2	0
1	5	8	-1	0

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

CORE REQUEST FUNCTION



PID	Start Line	End Line	Curr ent Line	State
0	1	4	**2**	1
1	5	8	-1	0

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

Incoming Parameters (Process, Process Table, Input Table)

This function will do two things:

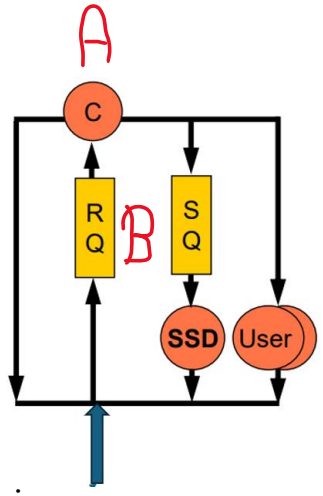
1. If the CORE is empty [A]
    1. set the CPU to full
    2. Change ptable status (not current line) (1 → Running)
    3. Find the completion Time = **clock time** + time needed (0 + 200)
    4. Push this process back into the **main queue** using its **completion time**

2. If the Core is Full (we did not do this one this time) **[B]**
    1. Change the status of the event to ready
    2. Put this process in the **READY QUEUE** with its **time needed**

This function **does not** return anything (void), we are just to handle an event and change our process table.

Action with our Example

Incoming Parameters (Process, Process Table, Input Table)



PID	Start Line	End Line	Curr ent Line	State
0	1	4	**2**	1
1	5	8	-1	0

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

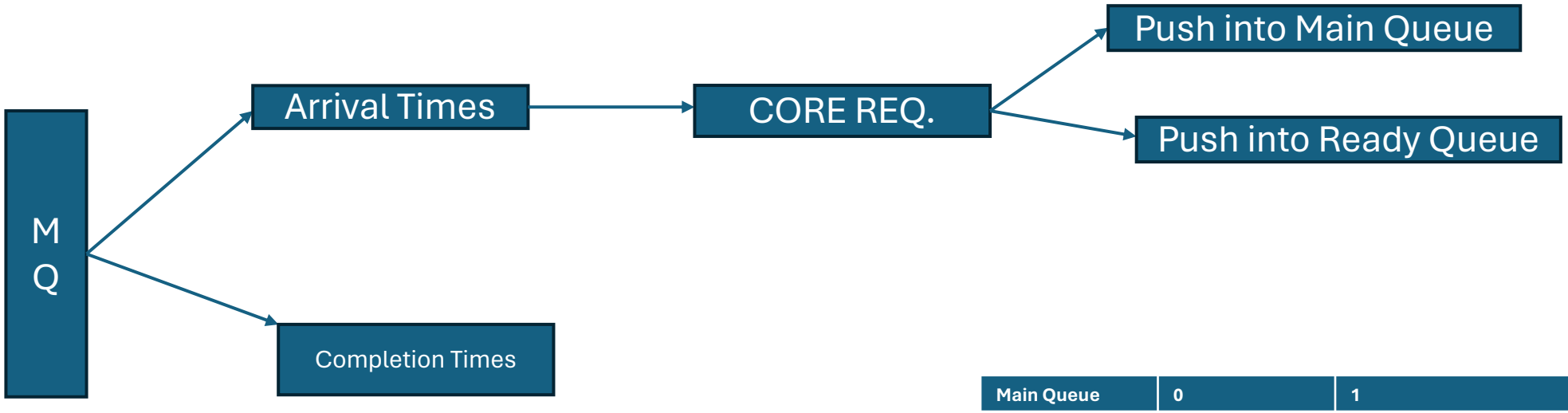
If the CORE is empty [A]

1. set the CPU to full
2. Change ptable status (not current line) (1 → Running)
3. Find the completion Time = **clock time** + time needed (0 + 200)
4. Push this process back into the **main queue** using its **completion time**
5. Our main queue now looks like this, I would like to point out there is only again

Completion Times and **Arrival Times** in our main queue

Main Queue	0	1
MQ	Process 1 Time = 100 START Log reads = 0 Phys writes =0 Phys reads =0	Process 0 Time = 200 CORE Log reads = 0 Phys writes =0 Phys reads =0

Let's create a flow chart of where we are now



Main Queue	0	1
MQ	Process 1 Time = 100 START Log reads = 0 Phys writes =0 Phys reads =0	Process 0 Time = 200 CORE Log reads = 0 Phys writes =0 Phys reads =0

Now start that while loop up

Main Queue	0	1
MQ	Process 1 Time = 100 START Log reads = 0 Phys writes =0 Phys reads =0	Process 0 Time = 200 CORE Log reads = 0 Phys writes =0 Phys reads =0

While MainQueue is not empty:

1. Save the top

0
Process 1 Time = 100 START Log reads = 0 Phys writes =0 Phys reads =0

2. Pop the top

(MQ after Pop)

Main Queue	0
MQ	Process 0 Time = 200 CORE Log reads = 0 Phys writes =0 Phys reads =0

3. Save top time to **clock time**

Clock Time = 100

4. Figure out the command

START → arrival event

ARRIVAL FUNCTION

0
Process 1 Time = 100 START Log reads = 0 Phys writes =0 Phys reads =0

Before we use the Core Request Function:

We need the arrival function to do something.

PID	Start Line	End Line	Current Line	State
0	1	4	2	0
1	5	8	6	0

1. Change the process table current line **(Current Line = Start Line +1)**
2. Get the command from the input table (which will be core)

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

We send these **updated** parameters to the core Request:

1. Process Table (you'll want to pass this by **reference** everywhere you go, since you're updating in each function)
2. Process (change the command and time)

We send these unmodified parameters to the core Request:

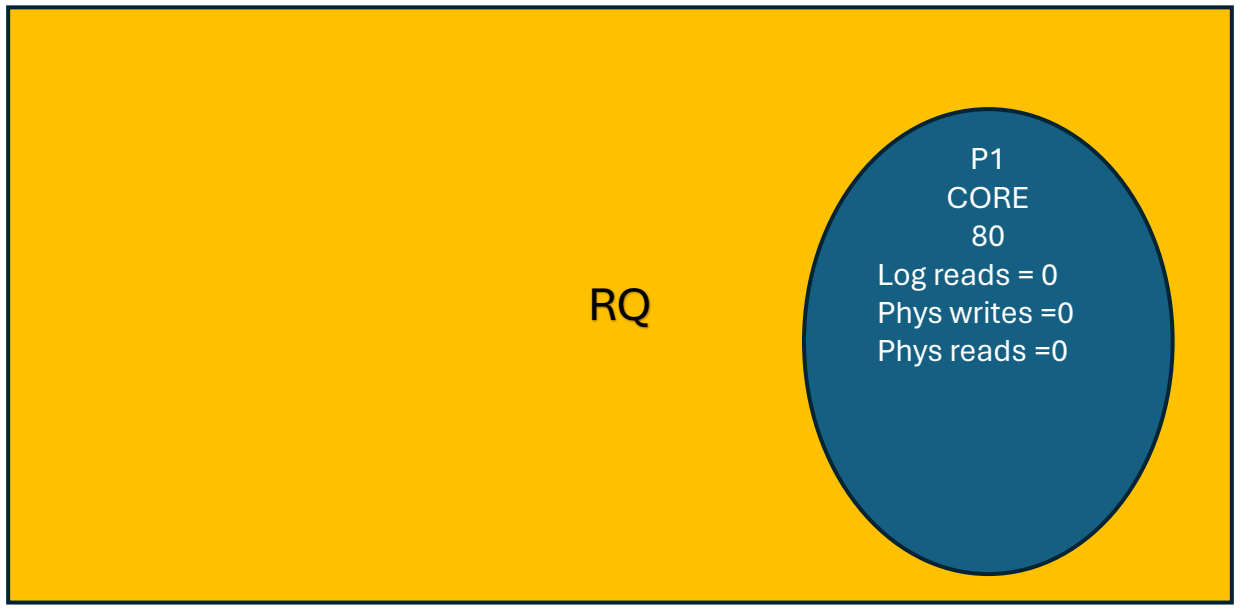
1. Input Table (this will never change)

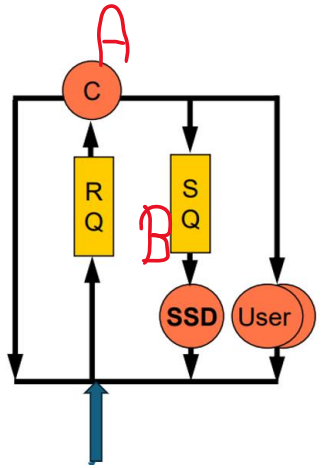
Action with our example

Incoming Parameters (Process, Process Table, Input Table)

CLOCK TIME = 100

2. If the Core is Full **[B]**
    1. Change the status of the event to ready (2)
    2. Put this process in the **READY QUEUE** with its **time needed**

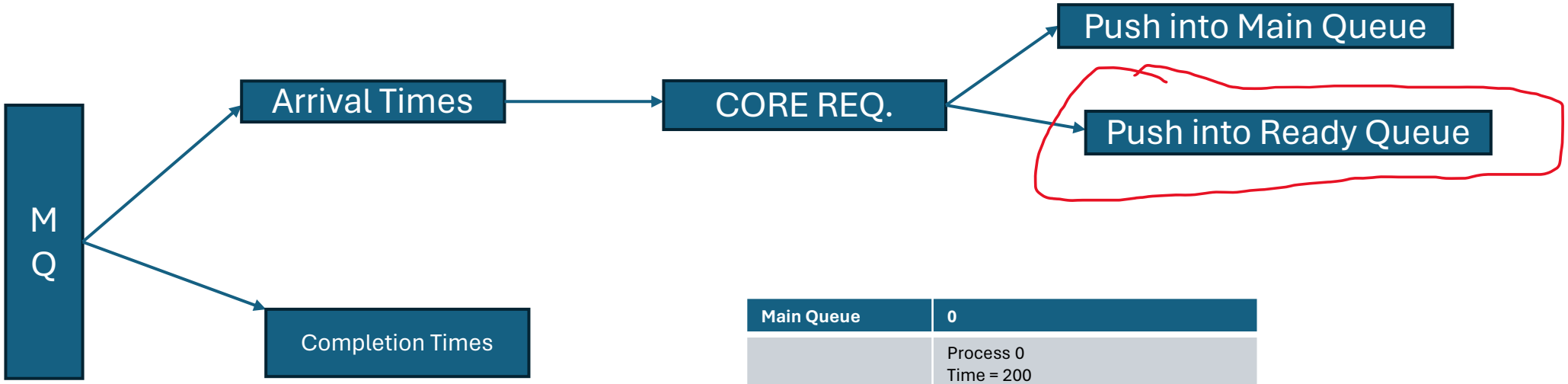




PID	Start Line	End Line	Current Line	State
0	1	4	2	1
1	5	8	6	2

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

Let's create a flow chart of where we are now



Main Queue	0
MQ	Process 0 Time = 200 CORE Log reads = 0 Phys writes =0 Phys reads =0

Now start that while loop up

Main Queue	0
MQ	Process 0 Time = 200 CORE Log reads = 0 Phys writes =0 Phys reads =0

While MainQueue is not empty:

1. Save the top

0
Process 0 Time = 200 CORE Log reads = 0 Phys writes =0 Phys reads =0

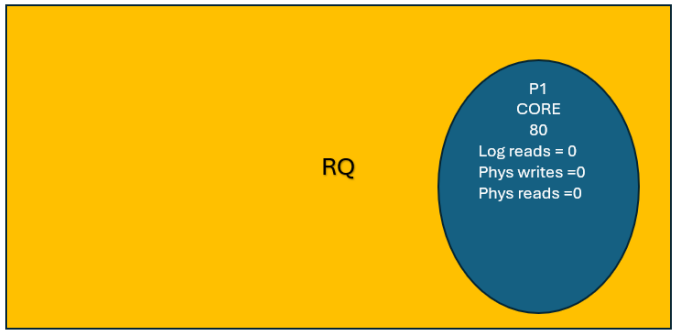
2. Pop the top

(MQ after Pop)

Main Queue
MQ

3. Save top time to **clock time**

Clock Time = 200



4. Figure out the command

CORE → completion event

Overview Completion Events

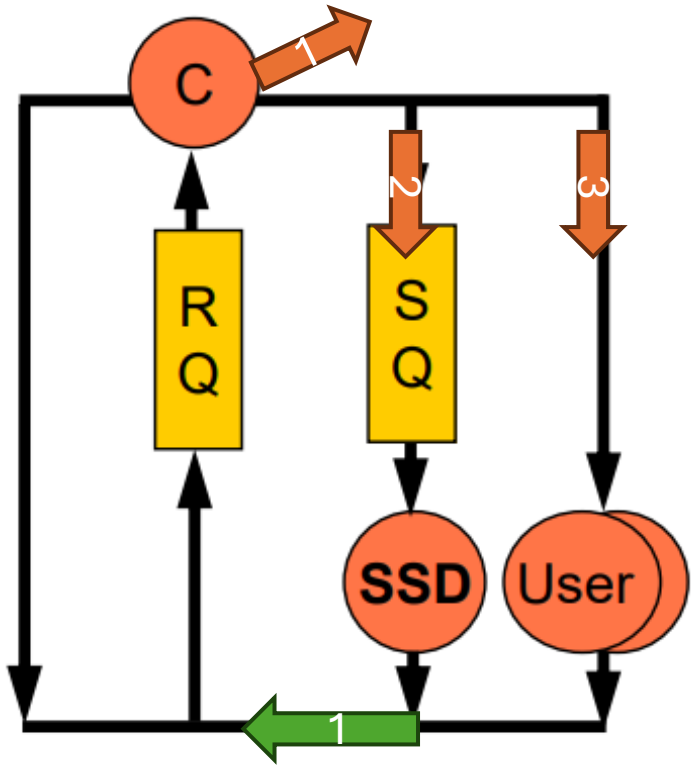
There are **2 main types** of completion events with different sub events:

Core Completion

1. Terminate or
2. Go to the SSD
3. Go to the Input/Display

SSD Completion

1. Go to the CORE



Core/SSD completion

This is a general principle for both:

Before we decide the next step, we need to take care of anything that was waiting for the core or SSD.

Analogy: This makes sense if there is room I am using, before I leave the room I will check if there was anyone waiting

1. If there was someone, I let them have the room, but the room is still full
2. If there was none, I make sure to note the room is empty

Overview Completion Events Finalized

There are **2 main types** of completion events with different sub events:

Core Completion

Set the Core to empty

Schedule the process (if) waiting (RQ) then:

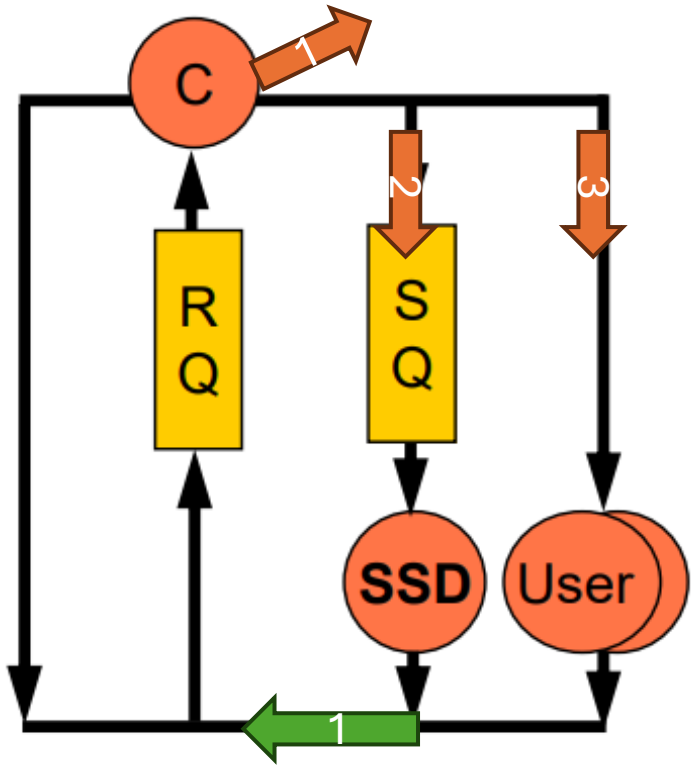
1. Terminate (print output) or
2. Go to the SSD
3. Go to the Input/Display

SSD Completion

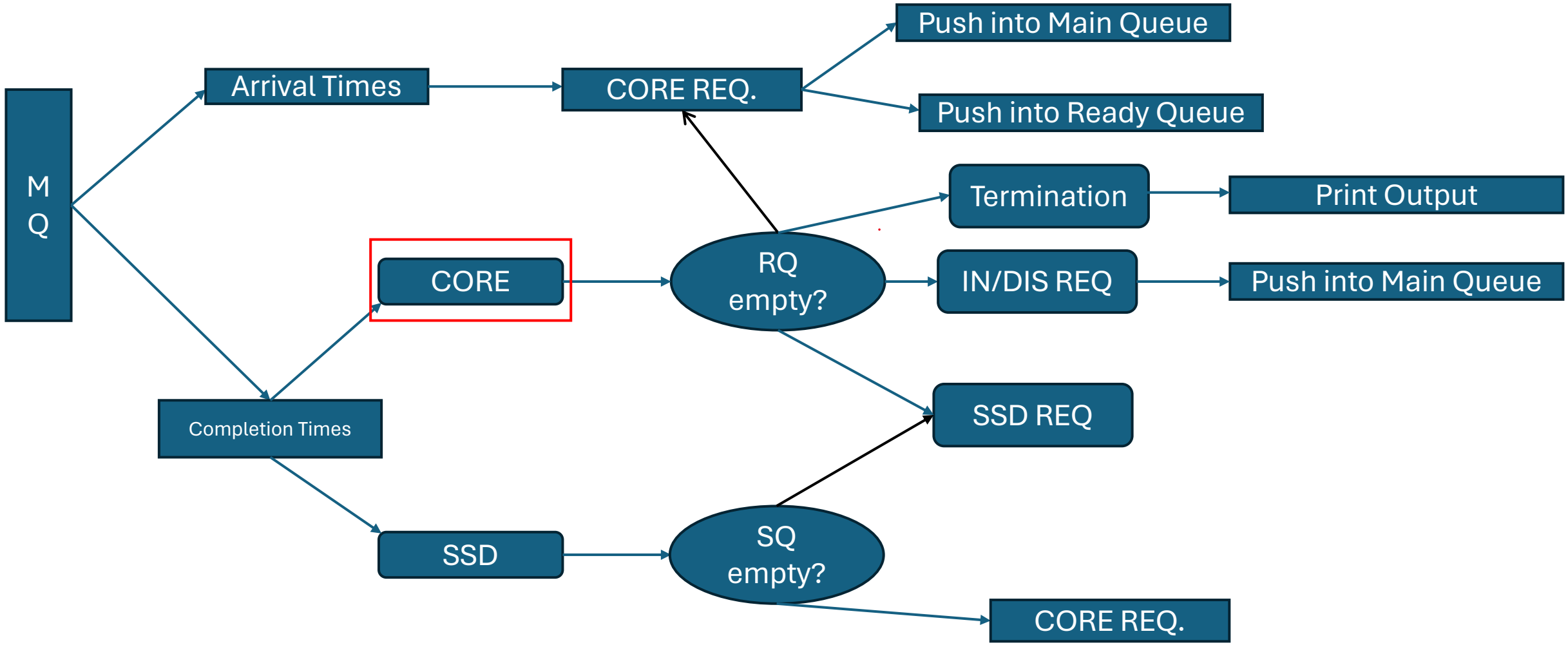
Set the SSD to empty

Schedule the process (if) waiting (SQ) then:

1. Go to the CORE



Let's create a flow chart of where we are now



Refresh where we are

Main Queue	0
MQ	Process 0 Time = 200 CORE Log reads = 0 Phys writes =0 Phys reads =0

While MainQueue is not empty:

1. Save the top

0
Process 0 Time = 200 CORE Log reads = 0 Phys writes =0 Phys reads =0

2. Pop the top

(MQ after Pop)

Main Queue
MQ

3. Save top time to **clock time**

Clock Time = 200

4. Figure out the command

CORE

Core Completion Function outline

Core = Empty

If(RQ is not empty):

Top = Rq.top(); Rq.pop();

CoreRequest(Top…)

If(End Line == Current Line): We Terminate → end the function/print output

Else:
Increment the current line by 1

If(Command == Read or Command == Write) : SSD request

If (Command == Input) : Push it back into the MQ with completion time

Core Completion w our Example

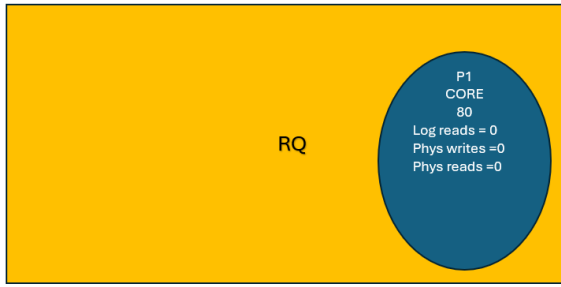
Core Completion

Set the Core to empty → Core is empty (global)

Schedule the process waiting (RQ) then:

1. Terminate (print output) or
2. Go to the SSD
3. Go to the Input/Display

Now to the core request function again



P1
CORE
80
Log reads = 0
Phys writes =0
Phys reads =0

PID	Start Line	End Line	Current Line	State
0	1	4	2	1
1	5	8	6	1

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

If the CORE is empty [A]

1. set the CPU to full
2. Change ptable status (not current line) (1 → Running)
3. Find the completion Time = **clock time** + time needed (200 + 80) = 280
4. Push this process back into the **main queue** using its **completion time**
5. Our main queue now looks like this, I would like to point out there is only again

Completion Times and **Arrival Times** in our main queue

Main Queue	0
MQ	Process 1 Time = 280 CORE Log reads = 0 Phys writes =0 Phys reads =0

Core Completion w our Example

PID	Start Line	End Line	Curr ent Line	State
0	1	4	3	1
1	5	8	6	1

Core Completion

Set the Core to empty → Core is empty (global)

Schedule the process (if one) waiting (RQ) then: (Done)

1. Terminate (print output) or

Increment Current Line by 1

1. Go to the SSD
2. Go to the Input/Display

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

Core Completion Function with Example

0
Process 0 Time = 200 CORE Log reads = 0 Phys writes =0 Phys reads =0

Core = Empty

If(RQ is not empty):

Top = Rq.top(); Rq.pop();

CoreRequest(Top…) → we already did this (Slide 36)

If(End Line == Current Line): We Terminate → end the function/print output

Else:
Increment the current line by 1 / (2+1) = 3

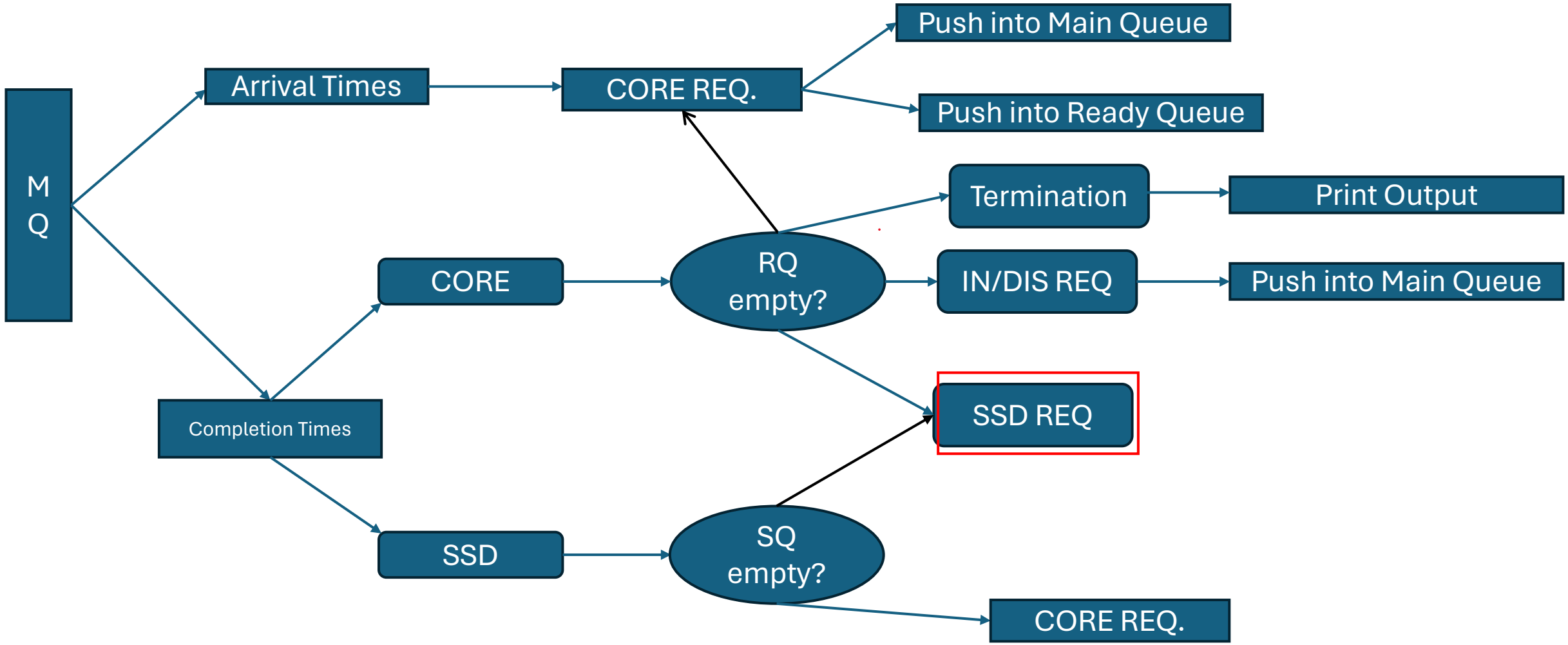
If(Command == Read or Command == Write) : SSD request

If (Command == Input) : Push it back into the MQ with completion time

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

PID	Start Line	End Line	Current Line	State
0	1	4	**3**	1
1	5	8	6	1

Let's create a flow chart of where we are now



SSD Request Overview

SSD request can be used by two commands :

1. Read

Reads are either **physical or logical:**

1. If physical cost us 0.1ms and we use the SSD
2. If logical cost us 0ms and we move to our next command (CoreRequest)

2. Writes

Writes are **always physical** and cost 0.1ms

SSD REQUEST FUNCTION

Let's simply this flow:

If the command is a **read** command

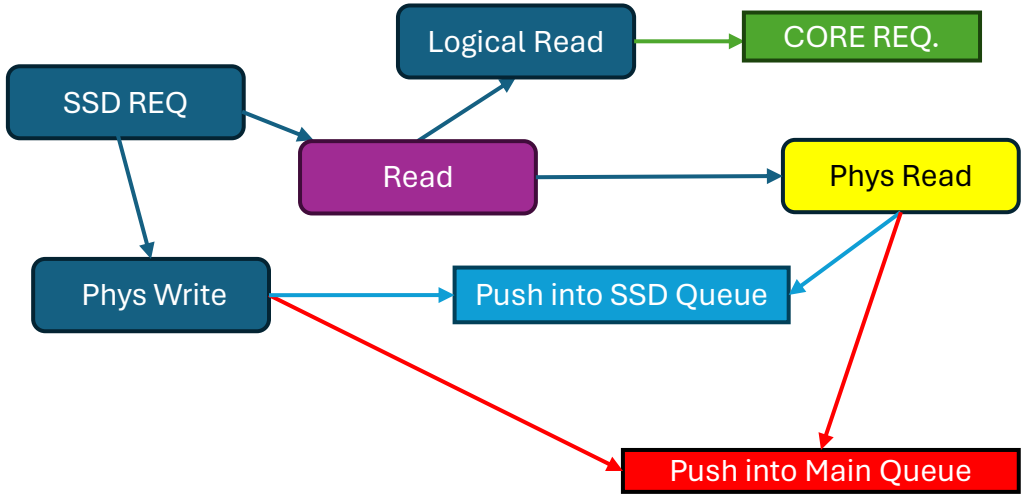
1. If the buffer is too small for the request → **Physical Read** → Push into **MainQueue (SSD empty)** or **SSD queue (SSD full)**

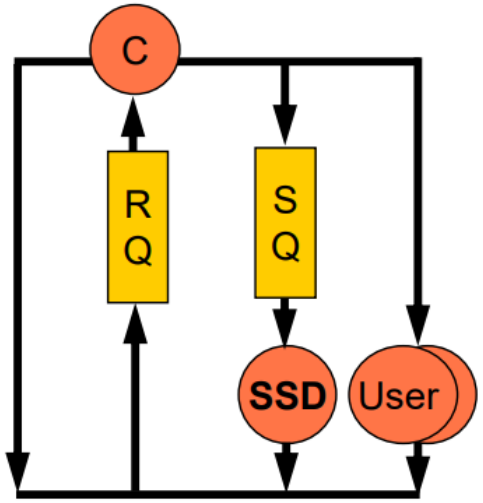
2. If the buffer has been read and the size is large enough for the request → Logical Read → Process next instruction for a **CoreRequest**

Look for the slides in explanation pdf (DR.Paris) to understand how to do the math for this.

If the command is a write command:

Always physical write (no BSIZE to check) →**Physical WRITE**→ Push into **MainQueue (SSD empty)** or **SSD queue (SSD full)**

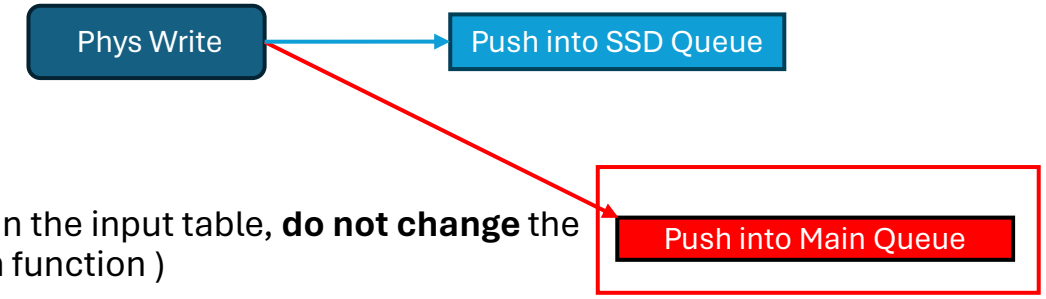




Revisit the Core Completion → SSD Request

The process 0 is at command write:

Is the SSD full? No



So, we update the status of the state to **blocked (3)** in the input table, **do not change** the current line (this will be done in the SSD completion function)

Pop this process 0 into the mainqueue with its completion time (clock time + completion = 200 + 0.1)

Remind you have to update the **physical writes** by 1!

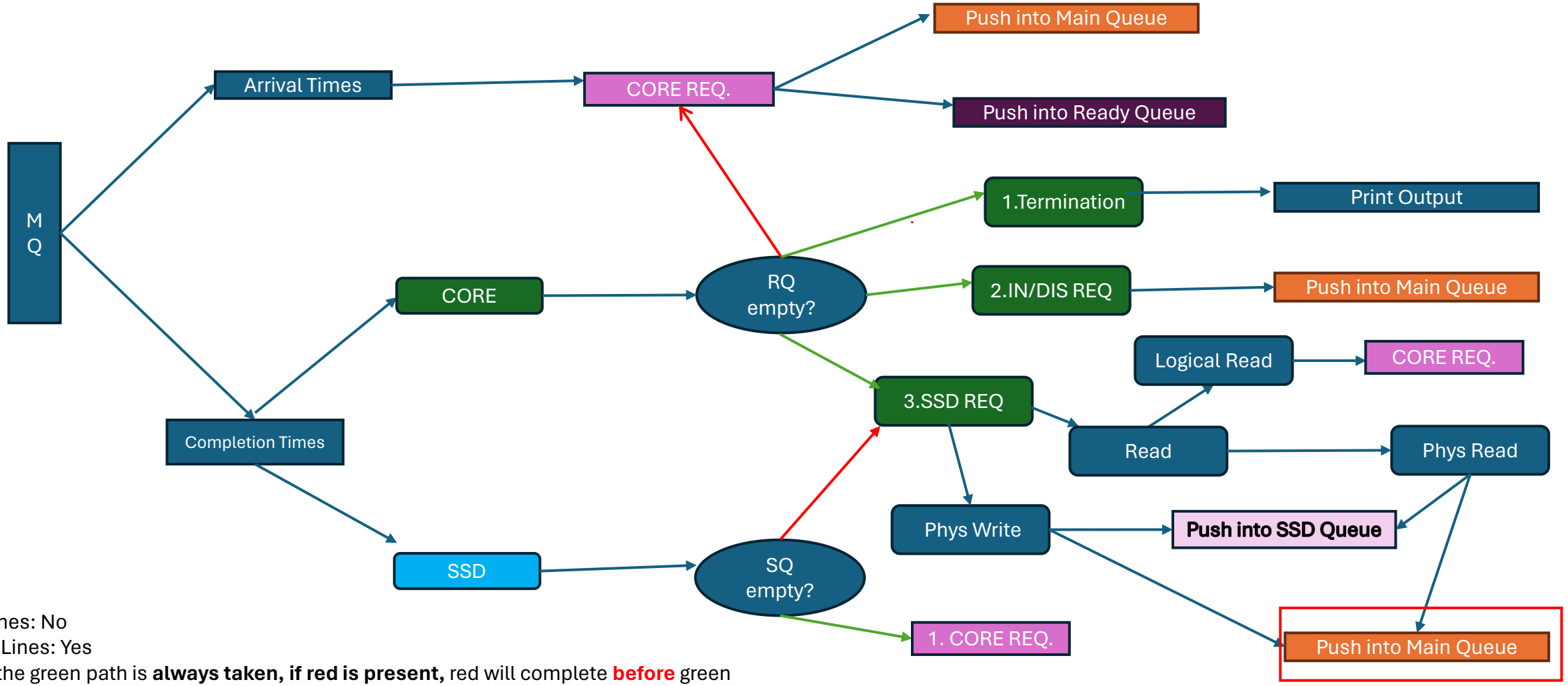
Remember the clock time was from when we popped out this process on **slide 33**

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

Main Queue	0	1
MQ	Process 0 Time = 280.1 WRITE Log reads = 0 Phys writes =1 Phys reads =0	Process 1 Time = 280 CORE Log reads = 0 Phys writes =0 Phys reads =0

PID	Start Line	End Line	Current Line	State
0	1	4	3	3
1	5	8	6	1

Final flow chart of where we are now



Red Lines: No
Green Lines: Yes
Note: the green path is **always taken, if red is present,** red will complete **before** green

Now start that while loop up

Main Queue	0	1
MQ	Process 0 Time = 280.1 WRITE Log reads = 0 Phys writes =0 Phys reads =0	Process 1 Time = 280 CORE Log reads = 0 Phys writes =0 Phys reads =0

While MainQueue is not empty:

1. Save the top

0
Process 0 Time = 280.1 WRITE Log reads = 0 Phys writes =0 Phys reads =0

2. Pop the top

(MQ after Pop)

Main Queue	1
MQ	Process 1 Time = 280 CORE Log reads = 0 Phys writes =0 Phys reads =0

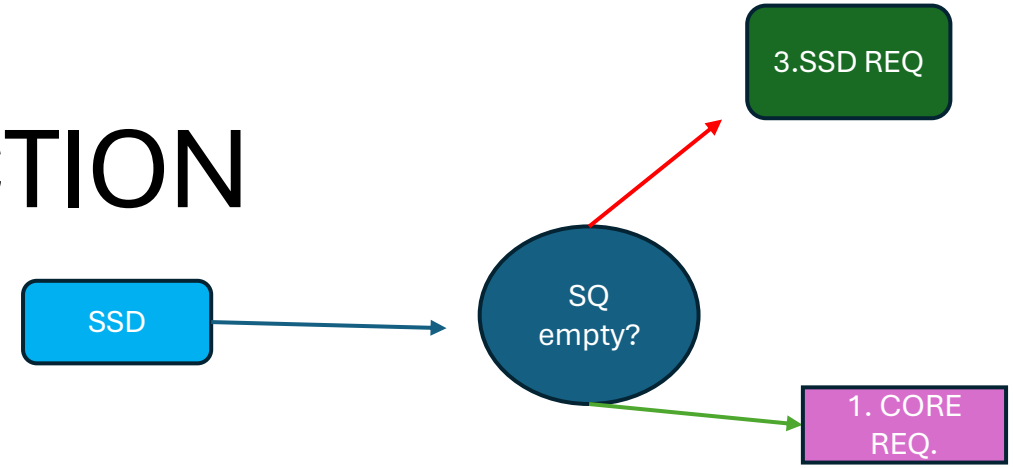
3. Save top time to **clock time**

Clock Time = 200.1

4. Figure out the command

WRITE→ **completion event**

SSD COMPLETION FUNCTION

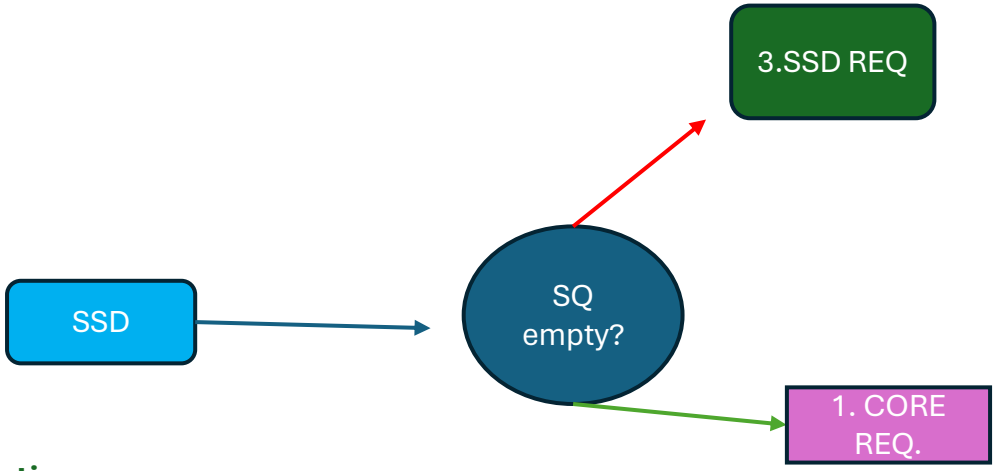


Same as core,

Set the SSD to empty then:

1. If any process was waiting in the SQ, send it to the SSD request function
2. Else: Look at its next command (will be core BTW)

Our Example SSD



Set the SSD to empty then:

1. **If any process was waiting in the SQ, send it to the SSD request function**
2. **Else:** Look at its **next command {increment current line}** (will be core BTW) → **this is the one we have**

Update the Process Table

Well think about it, we already made a **core request function**, so just send the process there

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

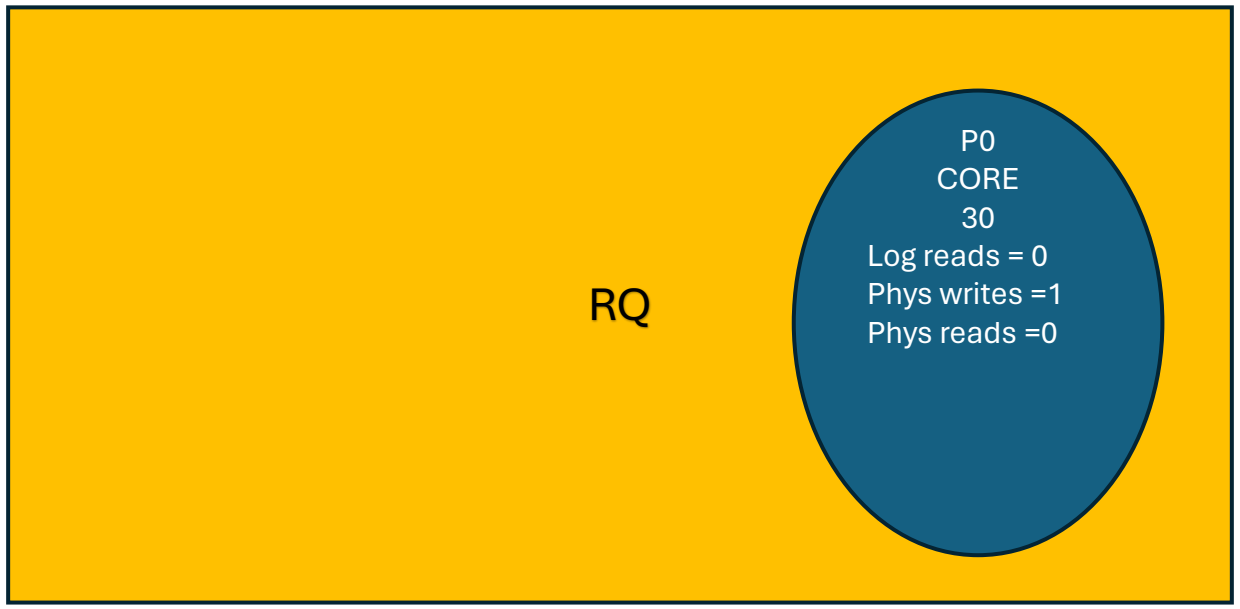
PID	Start Line	End Line	Current Line	State
0	1	4	4	3
1	5	8	6	1

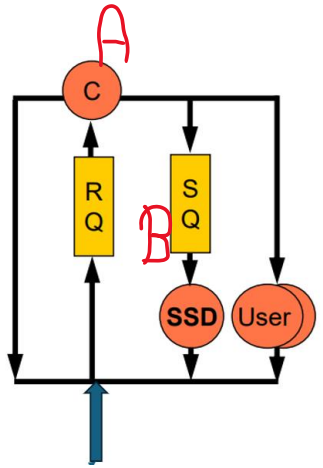
Action with our example

Incoming Parameters (Process, Process Table, Input Table)

CLOCK TIME = 200.1

2. If the Core is Full **[B]**
    1. Change the status of the event to ready (2)
    2. Put this process in the **READY QUEUE** with its **time needed**





PID	Start Line	End Line	Current Line	State
0	1	4	4	2
1	5	8	6	1

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

Now start that while loop up

Main Queue	0
MQ	Process 1 Time = 280 CORE Log reads = 0 Phys writes =0 Phys reads =0

While MainQueue is not empty:

1. Save the top

0
Process 1 Time = 280 CORE Log reads = 0 Phys writes =0 Phys reads =0

2. Pop the top

(MQ after Pop)

Main Queue
MQ

3. Save top time to **clock time**

Clock Time = 280

4. Figure out the command

CORE→ **completion event**

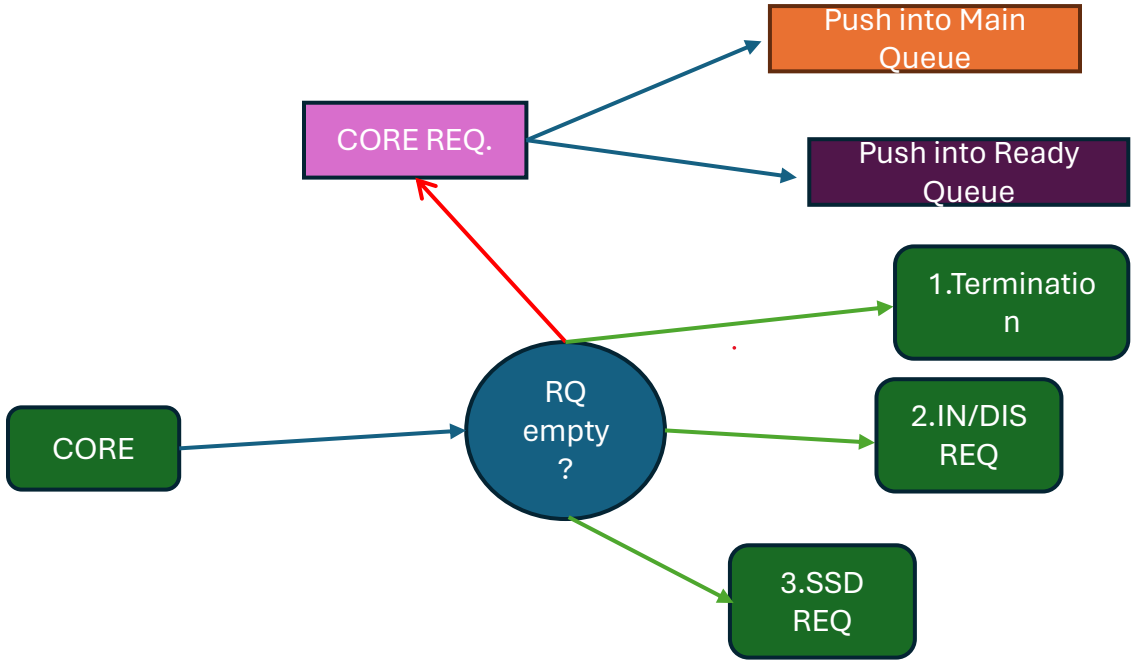
Core Completion w our Example

Core Completion

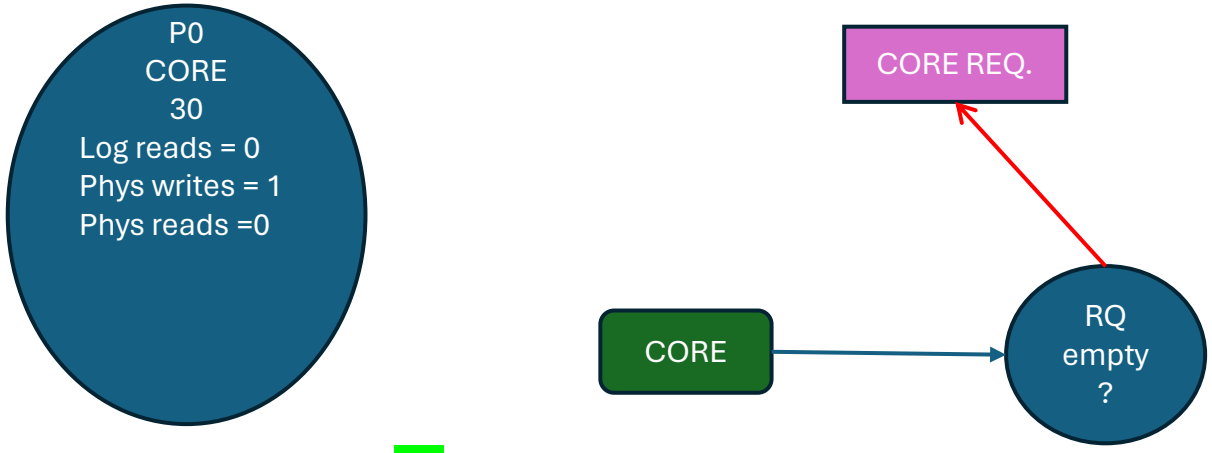
Set the Core to empty → Core is empty (global)

Schedule the process waiting (RQ) then:

1. Terminate (print output) or
2. Go to the SSD
3. Go to the Input/Display



Now to the core request function again



PID	Start Line	End Line	Curr ent Line	State
0	1	4	4	1
1	5	8	6	1

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

If the CORE is empty [A]

1. set the CPU to full
2. Change ptable status (not current line) (1 → Running)
3. Find the completion Time = **clock time** + time needed (280 + 30) = 310
4. Push this process back into the **main queue** using its **completion time**
5. Our main queue now looks like this, I would like to point out there is only again

Completion Times and **Arrival Times** in our main queue

Main Queue	0
MQ	Process 0 Time = 310 CORE Log reads = 0 Phys writes =0 Phys reads =0

Core Completion w our Example

PID	Start Line	End Line	Current Line	State
0	1	4	4	1
1	5	8	7	1

Core Completion

Set the Core to empty → Core is empty (global)

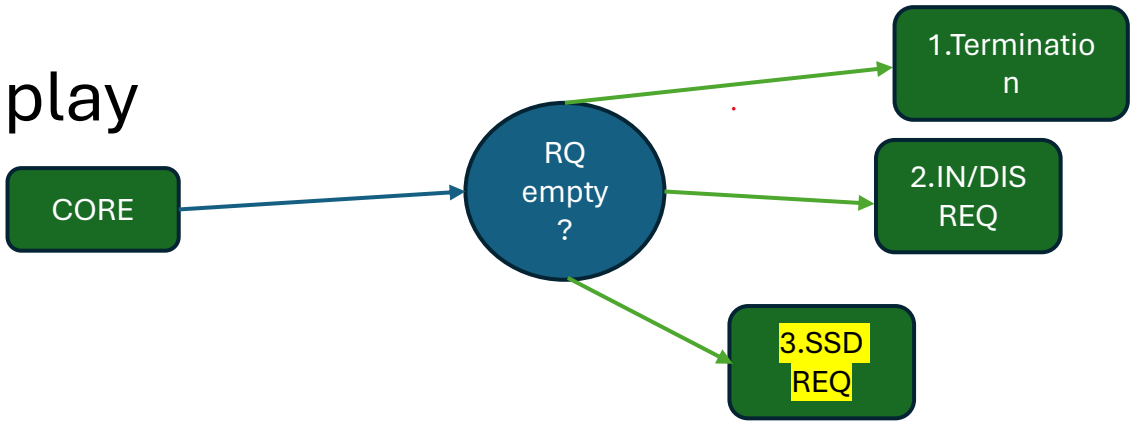
Schedule the process (if one) waiting (RQ) then: (Done)

1. Terminate (print output) or

Increment Current Line by 1

1. Go to the SSD
2. Go to the Input/Display

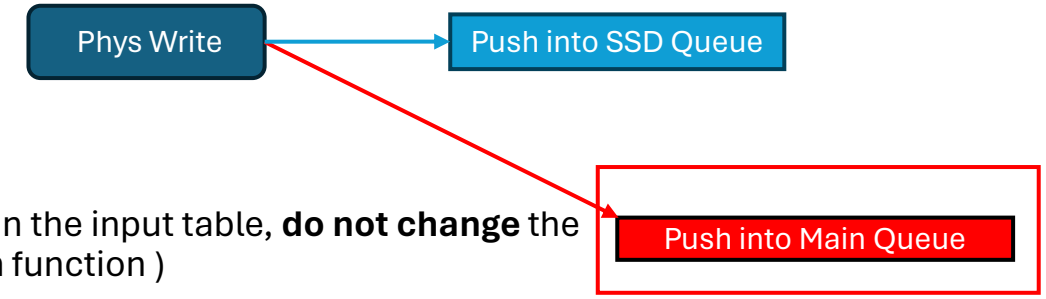
Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40



Revisit the Core Completion → SSD Request

The process 1 is at command **write:**

Is the SSD full? **No**



So, we update the status of the state to **blocked (3)** in the input table, **do not change** the current line (this will be done in the SSD completion function)

Pop this process 0 into the mainqueue with its completion time (clock time + completion = 280 + 0.1 = 280.1)

Remind you have to update the **physical writes** by 1!

Remember the clock time was from when we popped out this process on **slide 49**

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

Main Queue	0	1
MQ	Process 1 Time = 280.1 WRITE Log reads = 0 Phys writes =1 Phys reads =0	Process 0 Time = 310 CORE Log reads = 0 Phys writes =1 Phys reads =0

PID	Start Line	End Line	Current Line	State
0	1	4	4	1
1	5	8	7	3

Now start that while loop up

Main Queue	0
MQ	Process 0 Time = 310 CORE Log reads = 0 Phys writes =1 Phys reads =0

While MainQueue is not empty:

1. Save the top

0
Process 1 Time = 280.1 WRITE Log reads = 0 Phys writes =1 Phys reads =0

2. Pop the top

(MQ after Pop)

Main Queue	0
MQ	Process 0 Time = 310 CORE Log reads = 0 Phys writes =1 Phys reads =0

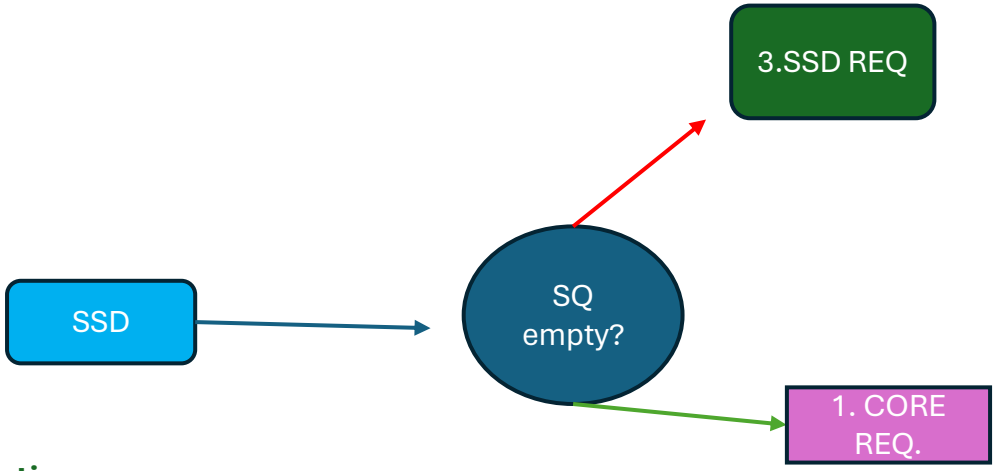
3. Save top time to **clock time**

Clock Time = 280.1

4. Figure out the command

WRITE→ **completion event**

Our Example SSD



Set the SSD to empty then:

1. **If any process was waiting in the SQ, send it to the SSD request function**
2. **Else:** Look at its **next command {increment current line}** (will be core BTW) → **this is the one we have**

Update the Process Table

Well think about it, we already made a **core request function**, so just send the process there

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

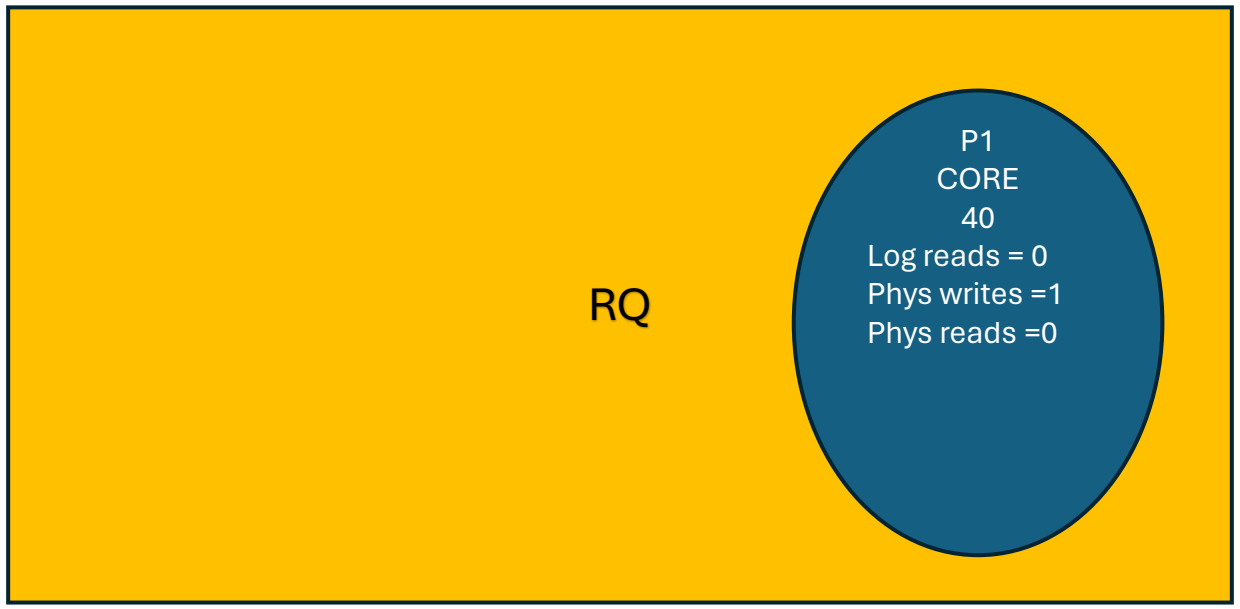
PID	Start Line	End Line	Current Line	State
0	1	4	4	3
1	5	8	8	1

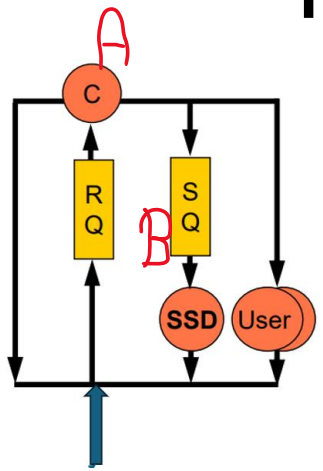
Action with our example (CoreRequest)

Incoming Parameters (Process, Process Table, Input Table)

CLOCK TIME = 280.1

2. If the Core is Full **[B]**
    1. Change the status of the event to ready (2)
    2. Put this process in the **READY QUEUE** with its **time needed**





PID	Start Line	End Line	Curr ent Line	State
0	1	4	4	1
1	5	8	8	2

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

Main lOOP

Main Queue	0
MQ	Process 0 Time = 310 CORE Log reads = 0 Phys writes =1 Phys reads =0

While MainQueue is not empty:

1. Save the top

0
Process 0 Time = 310 CORE Log reads = 0 Phys writes =1 Phys reads =0

2. Pop the top

(MQ after Pop)

Main Queue
MQ

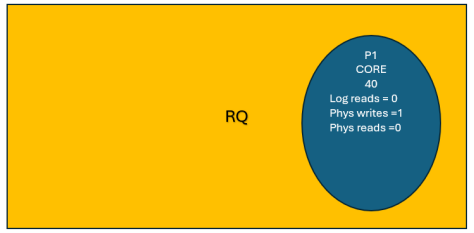
3. Save top time to **clock time**

Clock Time = 310

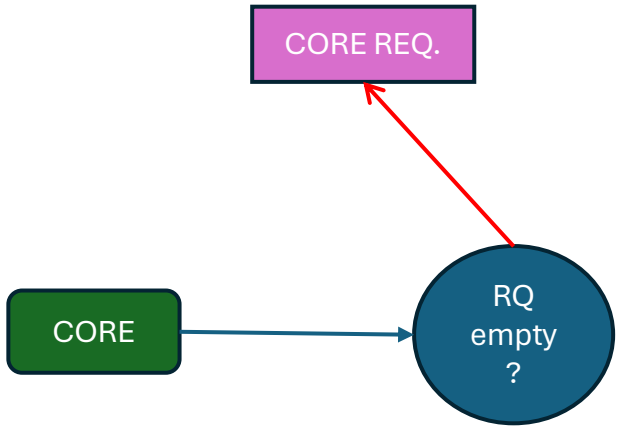
4. Figure out the command

CORE

Now to the core request function again



P1
CORE
40
Log reads = 0
Phys writes =1
Phys reads =0



PID	Start Line	End Line	Curr ent Line	State
0	1	4	4	1
1	5	8	8	1

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40

If the CORE is empty [A]

1. set the CPU to full
2. Change ptable status (not current line) (1 → Running)
3. Find the completion Time = **clock time** + time needed (310 + 40) = 350
4. Push this process back into the **main queue** using its **completion time**
5. Our main queue now looks like this, I would like to point out there is only again

Completion Times and **Arrival Times** in our main queue

Main Queue	0
MQ	Process 1 Time = 350 CORE Log reads = 0 Phys writes =1 Phys reads =0

Core Completion w our Example

PID	Start Line	End Line	Current Line	State
0	1	4	4	-1
1	5	8	8	1

Core Completion

Set the Core to empty → Core is empty (global)

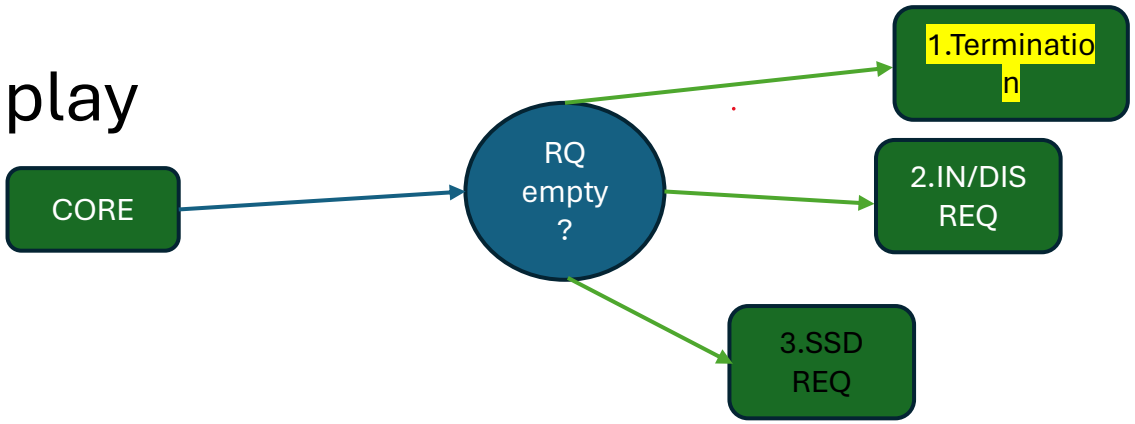
Schedule the process (if one) waiting (RQ) then: (Done)

1. Terminate (print output) or (Current Line = Endline)

Increment Current Line by 1

1. Go to the SSD
2. Go to the Input/Display

Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40



Here we print the output

Process 0 terminates at t = 310ms. It performed 0 physical read(s), 0 logical read(s), and 1 physical write(s). Process states: ----------
0 TERMINATED 1 RUNNING

Main lOOP

Main Queue	0
MQ	Process 1 Time = 350 CORE Log reads = 0 Phys writes =1 Phys reads =0

While MainQueue is not empty:

1. Save the top

0
Process 1 Time = 350 CORE Log reads = 0 Phys writes =1 Phys reads =0

2. Pop the top

(MQ after Pop)

Main Queue
MQ

3. Save top time to **clock time**

Clock Time = 350

4. Figure out the command

CORE

Core Completion w our Example

PID	Start Line	End Line	Current Line	State
0	1	4	4	-1
1	5	8	8	-1

Core Completion

Set the Core to empty → Core is empty (global)

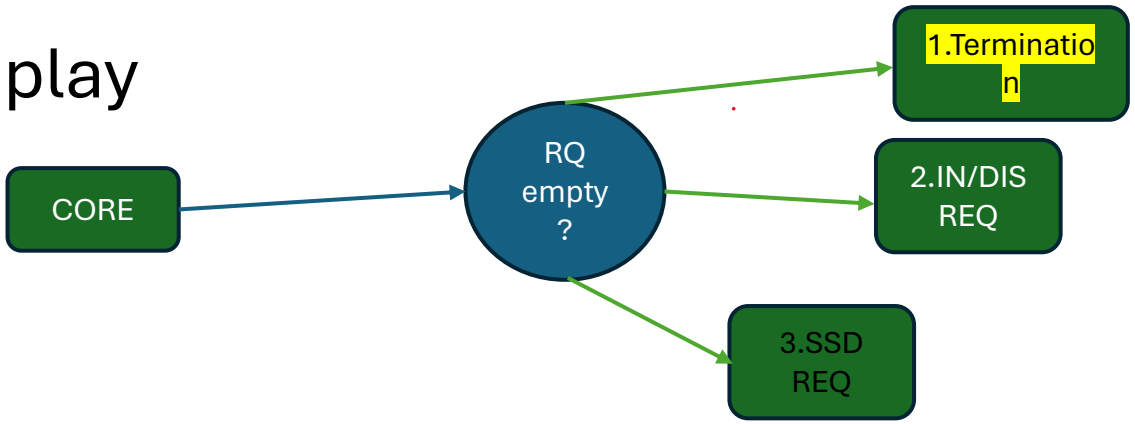
Schedule the process (if one) waiting (RQ) then: (None/Done)

1. Terminate (print output) or (Current Line = Endline)

Increment Current Line by 1

1. Go to the SSD
2. Go to the Input/Display

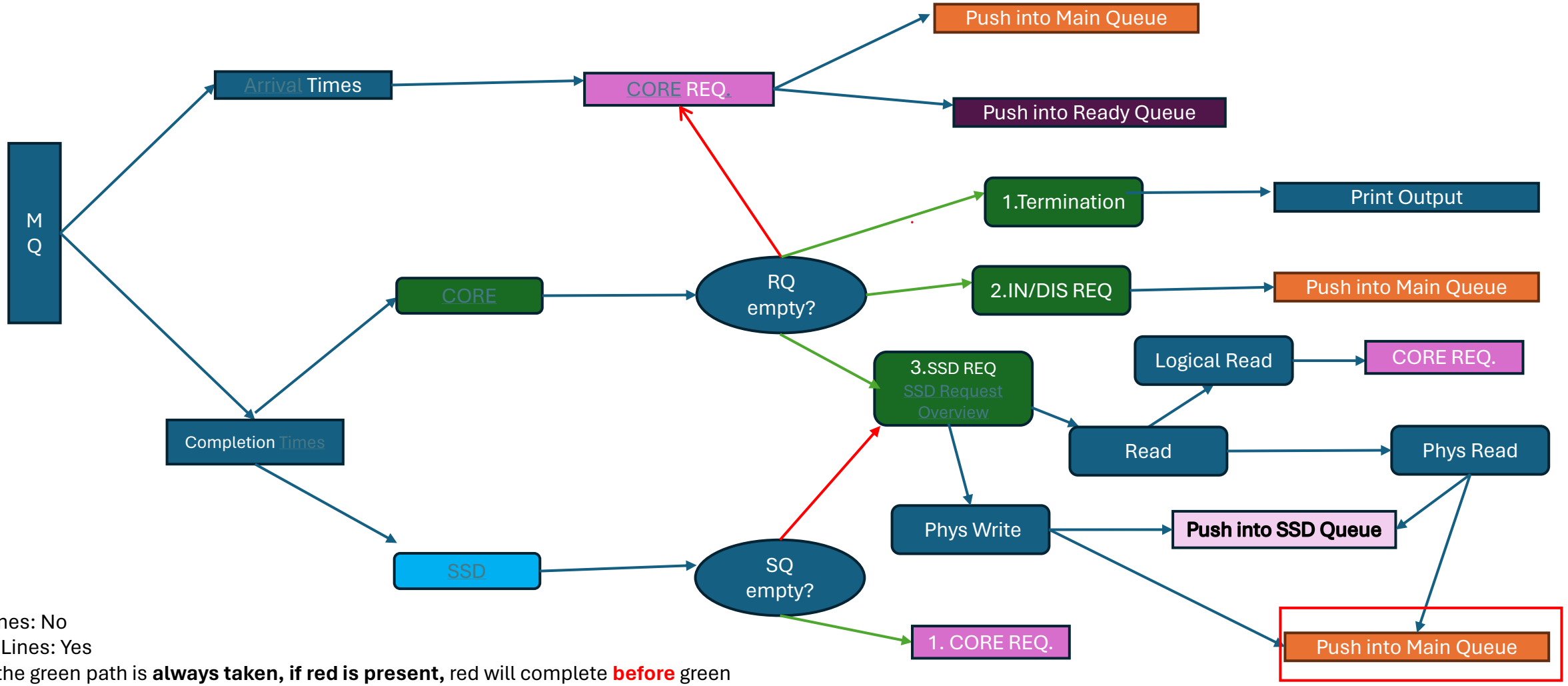
Index/PID	Com.	Time
0	BSIZE	4096
1	START	0
2	CORE	200
3	WRITE	4096
4	CORE	30
5	START	100
6	CORE	80
7	WRITE	4096
8	CORE	40



Here we print the output

Process 1 terminates at t = 350ms. It performed 0 physical read(s), 0 logical read(s), and 1 physical write(s). Process states: ----------
1 TERMINATED

Overview (click on each for reference)



Red Lines: No
Green Lines: Yes
Note: the green path is **always taken, if red is present,** red will complete **before** green