Authors: Robert Le, Aashil Ahmad, Ashwin Anil

**Table of Contents**

* Notes/Graphs
  + Round Robin
    - Time Quantum of 10
    - Time Quantum of 50
    - Time Quantum of 100
    - Comparison of the TQs related to total time and CPU efficiency and more
  + FCFS
    - Process Arrival Time vs TAT
    - Service Time vs TAT
* Comparison of both Algorithms
* Conclusions

**Notes/Graphs**

ROUND ROBIN

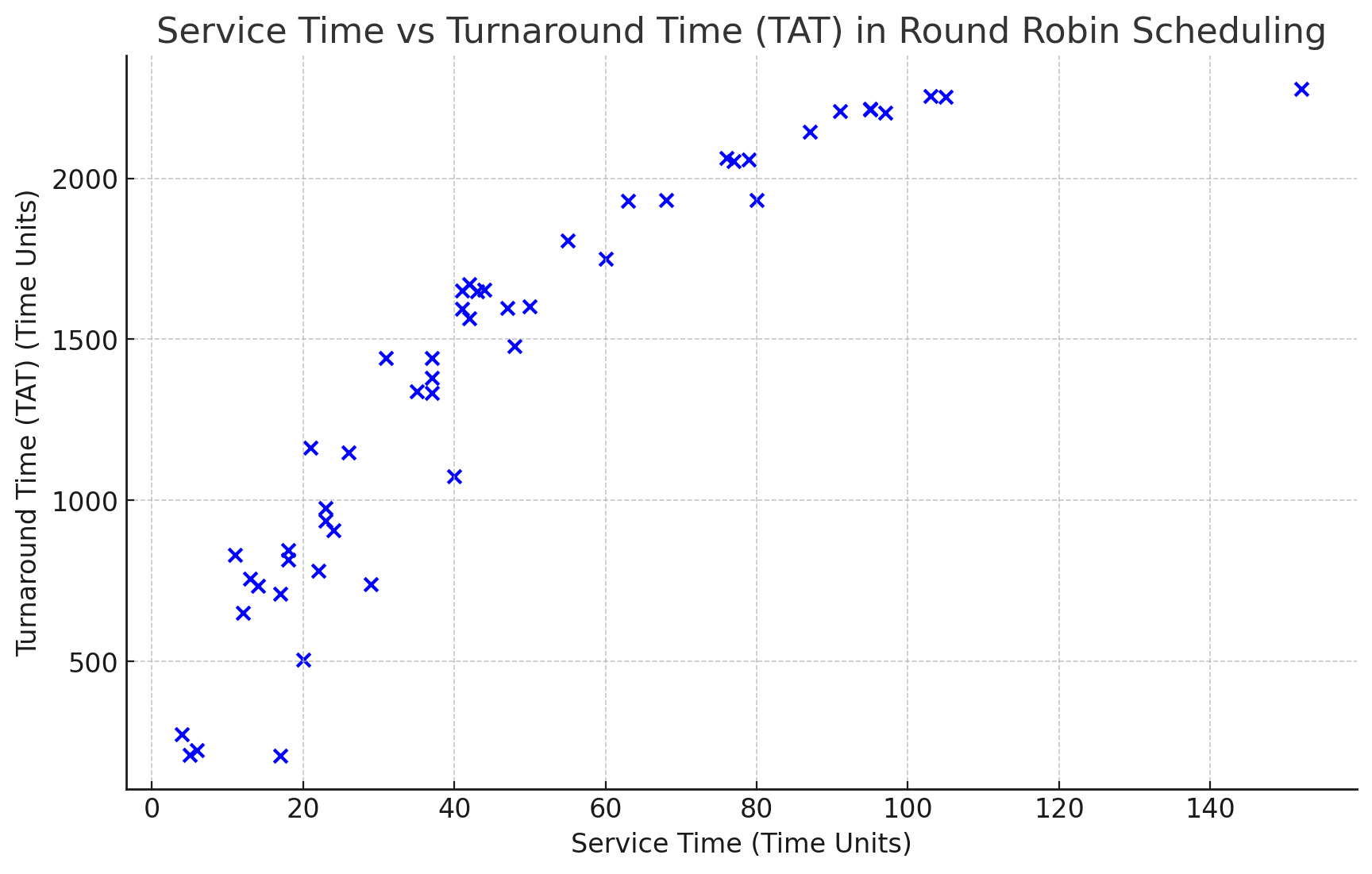
* There is a direct linear relationship between Service time and Turnaround time/Waiting time. The smaller processes that require less execution time will ultimately have less turnaround time/waiting time. Due to the equal sharing of execution in RR with time slices, small processes will not have to wait for big processes to be finished executing (such as FCFS) and instead be given their time slice to do their work without waiting for other processes to finish. Also more beneficial with our small Time Quantum, the smaller processes will need to wait less.
* We used a time quantum of 2, which impacts scheduling since there is more context switching due to the short amount of time slice each process can execute, (maybe causing overhead), also could potentially increase response times due to increased switching.
* Maybe we should use a time quantum of 10, 50, and 100 instead.
* If we had mostly longer processes, a bigger TQ would be optimal due to less context switching = less overhead, and the long processes could finish or almost finish in their long time slices undisturbed. Vice versa for mostly small processes.
* The goal is to have an optimal TQ so that processes in the queue are finished fast and efficiently with reduced context switching, allowing fewer response times in the waiting processes.
* FCFS beats RR with lower TAT (if RR TQ is 2), but ultimately RR won with the lowest Total Time. Let's analyze it.

**If Round Robin’s Time Quantum is 2:**

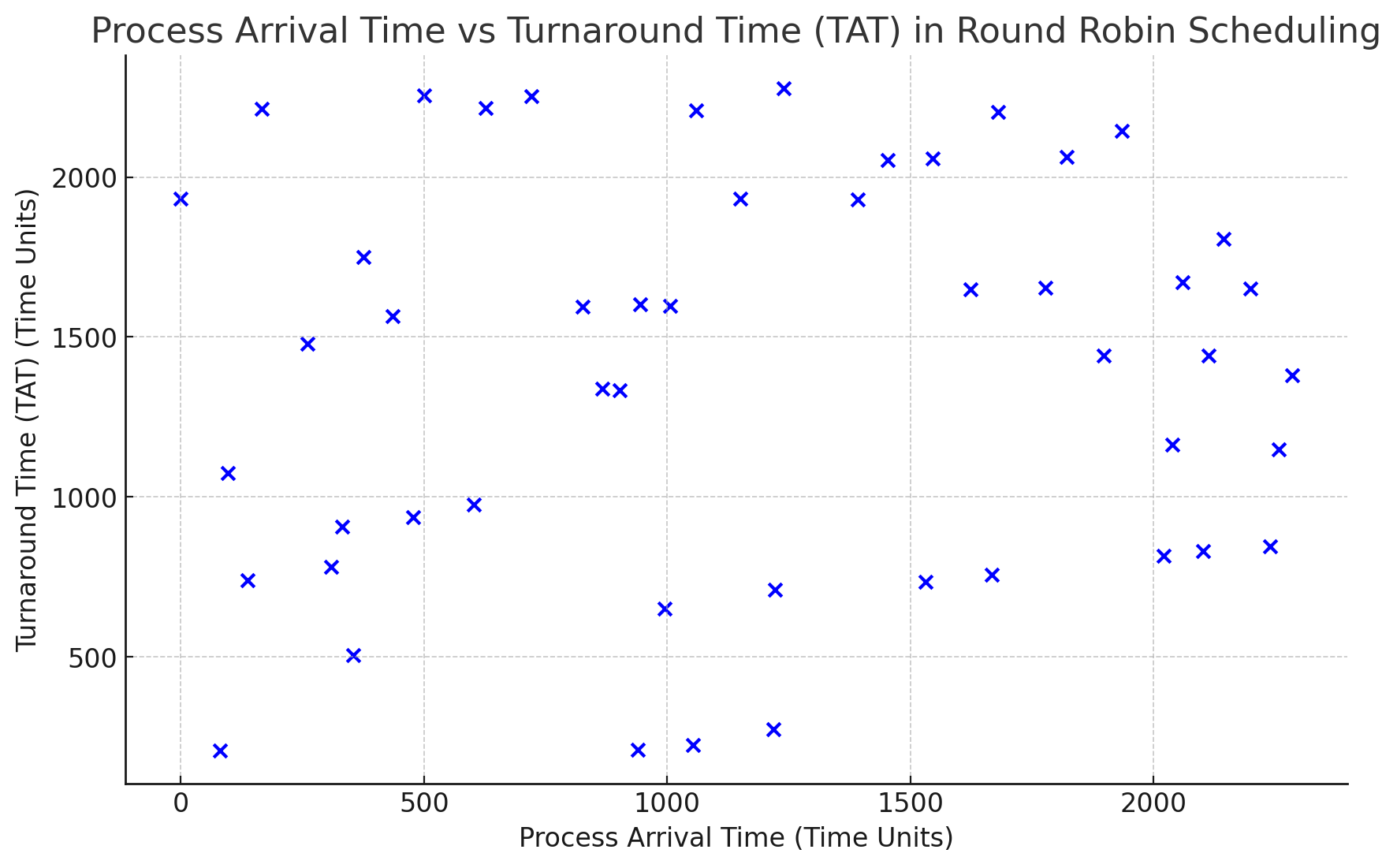
* **Why RR has lowest total time:** fairness with the Time Quantum, ensures efficient utilization of CPU time, thus meaning smaller processes can finish first.
* **Why RR has a higher average TAT:** an increase in context switching due to Time Quantum, causes overhead where it includes the time to save state in the current process and load the next process.
* Also, account for if processes don't finish in their TQ, they will have to wait for other processes to complete their share, thus increasing their TAT
* **Why FCFS has a lower average TAT (if RR has a TQ of 2):** once a process comes in and goes out once it's done executing, there is no context switching and, therefore no added overhead time. However, FCFS can have high TAT if the long processes come before the short ones.
* **Why FCFS has higher total time:** due to less efficient CPU utilization of processes since it is just a simple rule of taking the process and finishing it. Let's say We have a million small processes that will take 1 second to finish execution, but if a 10-year-long process was the first in the queue, we would have to wait 10 years for that one process to finish before executing the others, causing a lot of inefficiencies

**Now, instead of a Time Quantum of 2, let’s test with 10, 50, and 100.**

**Time Quantum of 10:**

****

* As stated before, there is a direct relationship between Service Time and TAT in Round Robin. Due to the fair share of execution, and no convoy effect of waiting for other processes to finish, the small processes are able to finish executing first no matter what time they arrive.
* With a time quantum of 10 for 50 processes, we feel it seems to be more balanced given the pool. It may seem a little low if we have mostly large processes, but if we had mostly small it would probably be very good.
* As stated before, too low of a TQ would make too many turns/context switching causing a lot of CPU overhead, with the possibility of processes not finishing in their first turn, and too high would still cause more time due to, let's say a big process is given a huge chunk of time to finish, the small processes will still have to wait with this huge chunk of time



* This will go for all Round Robin simulations regardless of time quantum, in Round Robin, process arrival time does not matter on what finishes executing first. Compared to FCFS, process arrival time matters.
* Let’s also log additional data such as Time Quantum and Total Time, so we can compare the Total Time for all 3 TQs (10,50,100) and compare.

*Additional Data:*

The Total Time required is 1638 time units

Average Turn Around Time: 1383.9 time units

Average Waiting Time: 1337.48 time units

Average Response Time: 1378.68 time units

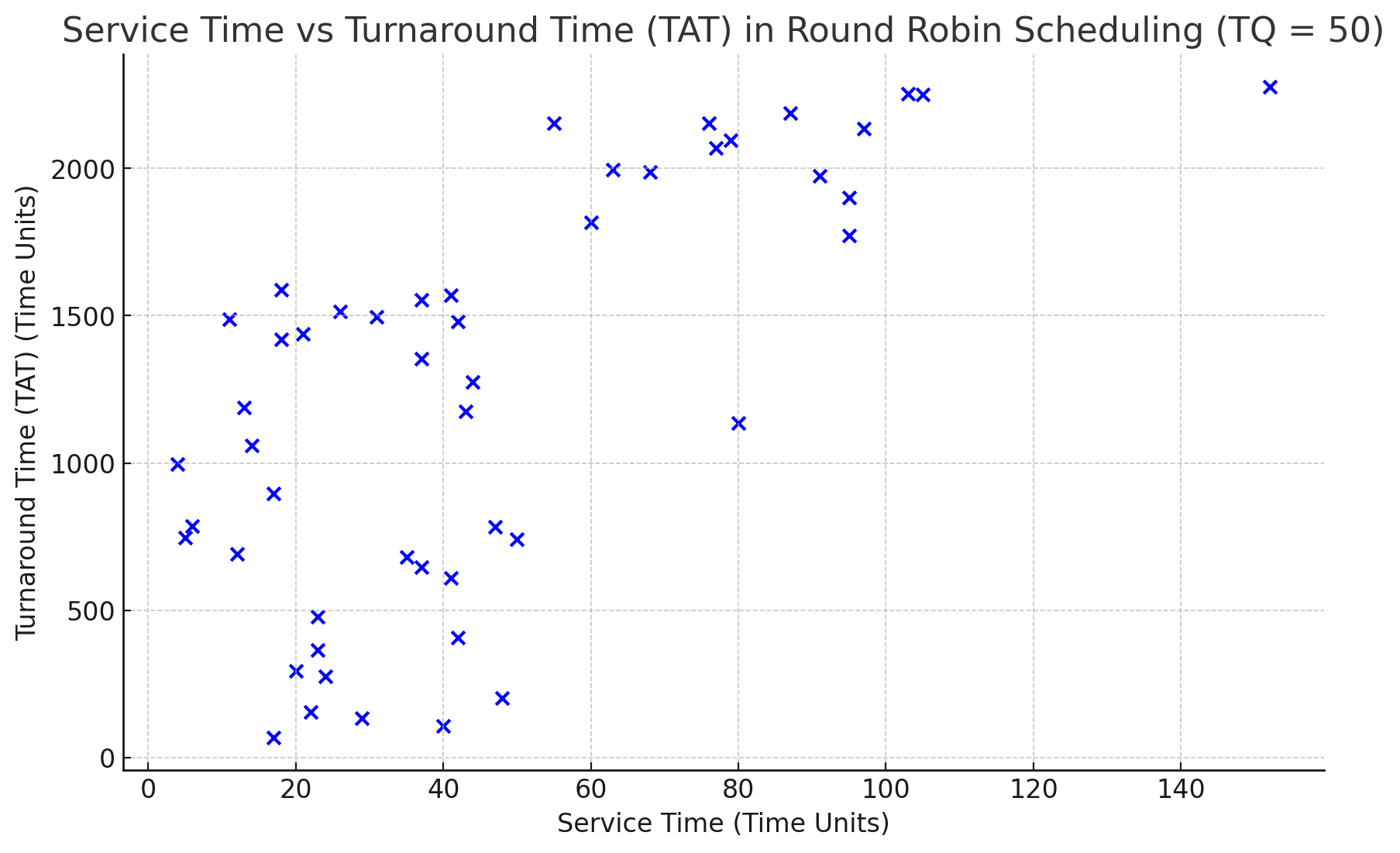
CPU Efficiency: 96.2406%

—---------------------------------------------------------------------------------------------------------------------------------------------------

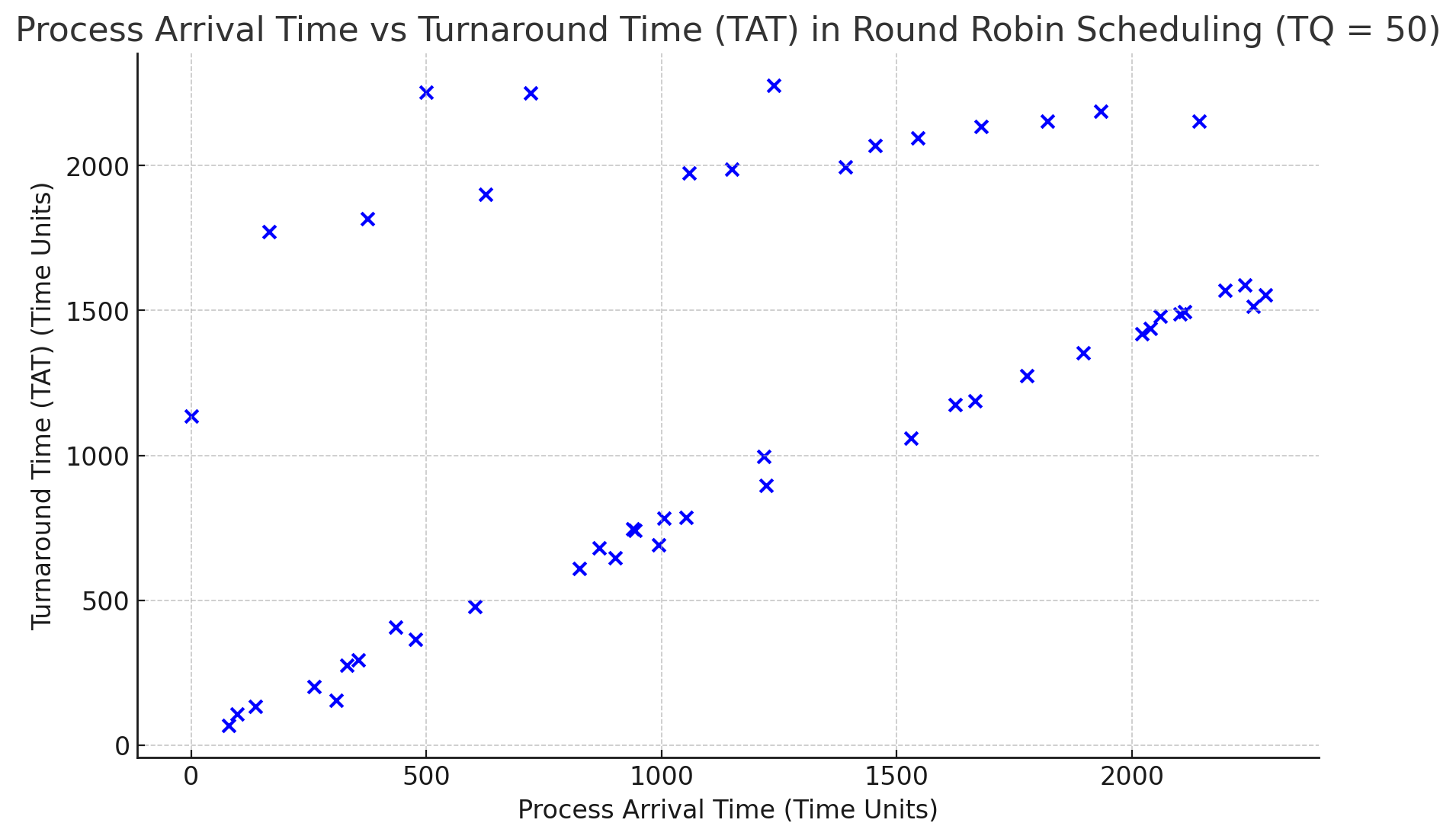
**Time Quantum of 50:**

Initial thoughts:

* With a TQ of 50 for 50 processes we think it is more balanced since we have a mix of both big and small processes. We think this is a fair amount of time for each process to finish or mostly finish in its turn time.



* There seems to be a direct relationship again between service time and TAT. Most can conclude that small processes will still finish first.
* However, the data seems to be way more sparse and spread out.
* Probably this is due to the higher TQ, leading to the small processes having to wait longer times to finish execution.
* The larger processes shouldn’t have trouble since they are given a larger amount of time to finish, which must be why the graph becomes a little clearer when service time increases.
* But due to more shared time for large processes, the small processes will have to wait longer



* Interestingly, it seems that process arrival time is starting to matter, with a few outliers. Maybe it is due to the fact that small processes waiting for the bigger processes to finish with their higher share of time.

*Additional Data:*

Total Time required is 1813 time units

Average Turn Around Time: 1236.48 time units

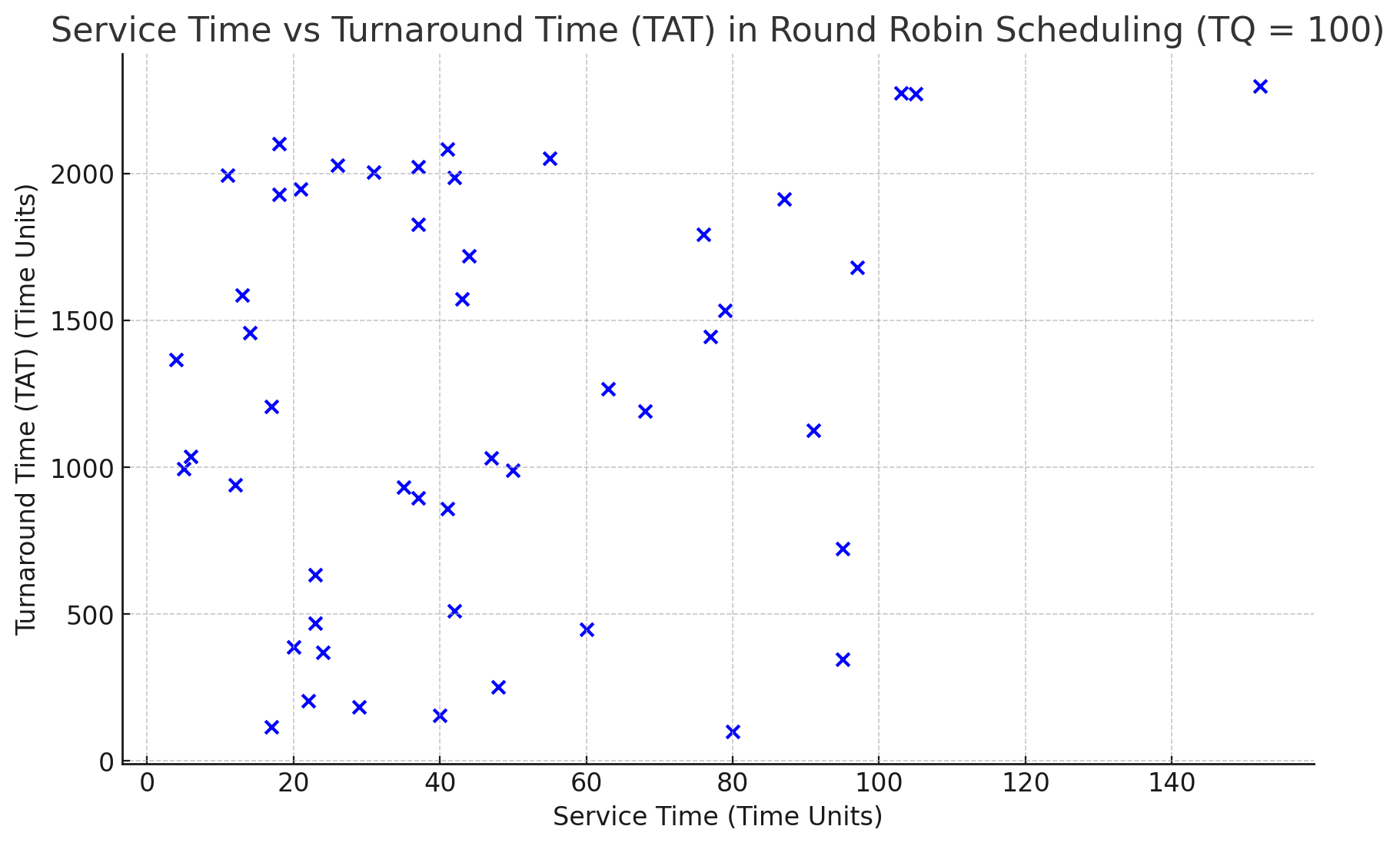
Average Waiting Time: 1190.06 time units

Average Response Time: 1210.06-time units

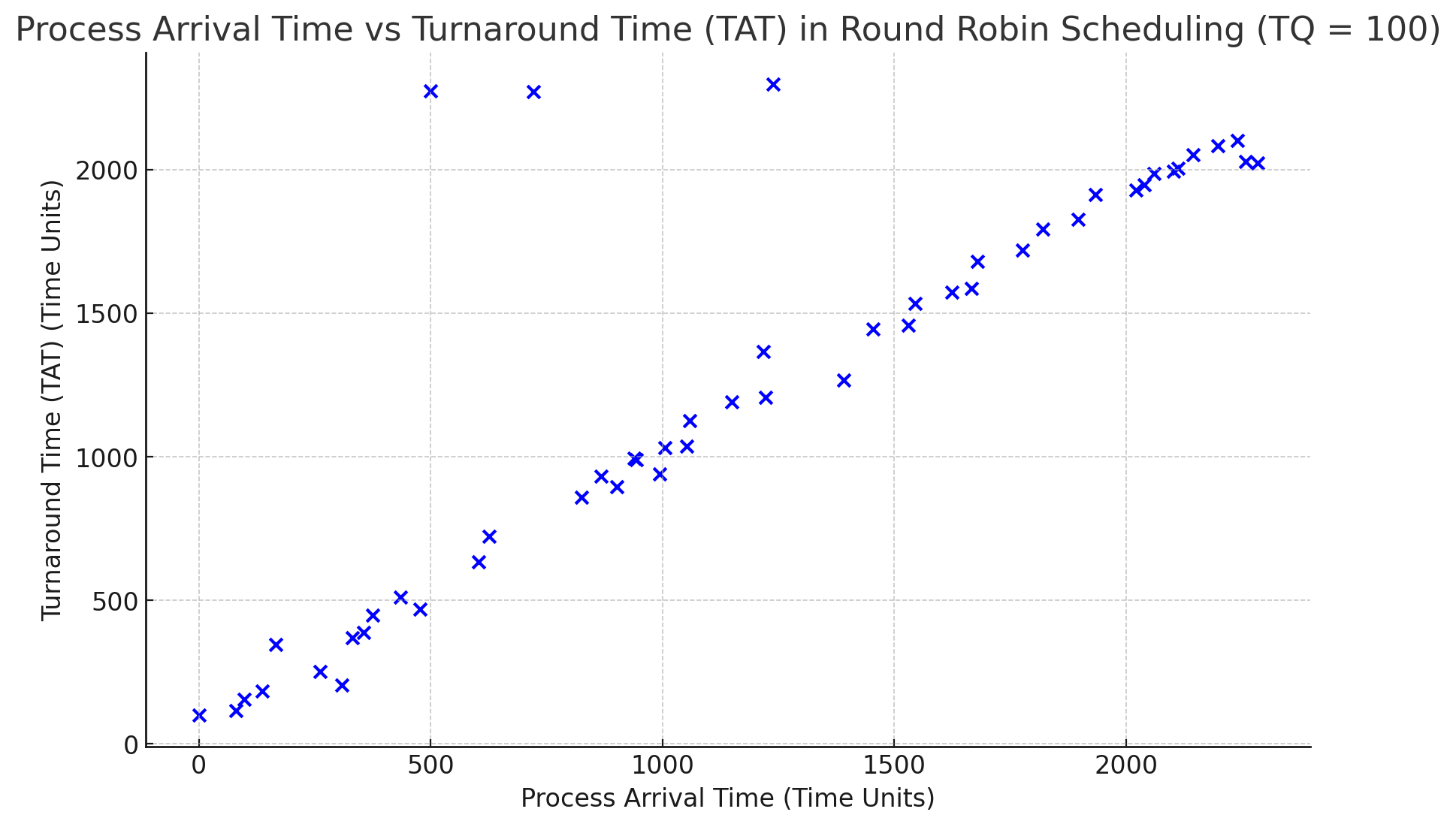
CPU Efficiency: 97.2222%

—-----------------------------------------------------------------------------------------------------------------------------------------------------

**Time Quantum of 100:**



* Seems our hypothesis was correct. Looks like small processes are getting a taste of FCFS, and are suffering due to the large processes hogging their timeshare to finish their execution (Convoy Effect)
* Now process arrival order definitely matters for fast execution time, not service time, since the small processes if arriving first would be able to finish first since they had their turn first and didn’t have to wait for a gigantic time quantum



* Wow, look at that. This looks like FCFS qualities.
* Contradicting our earlier point that process arrival doesn’t matter in RR regardless of TQ, this TQ matters
* We can conclude that a higher TQ would make process arrival more important/relevant

*Additional Data:*

Total Time required is 2281 time units

Average Turn Around Time: 1245.24 time units

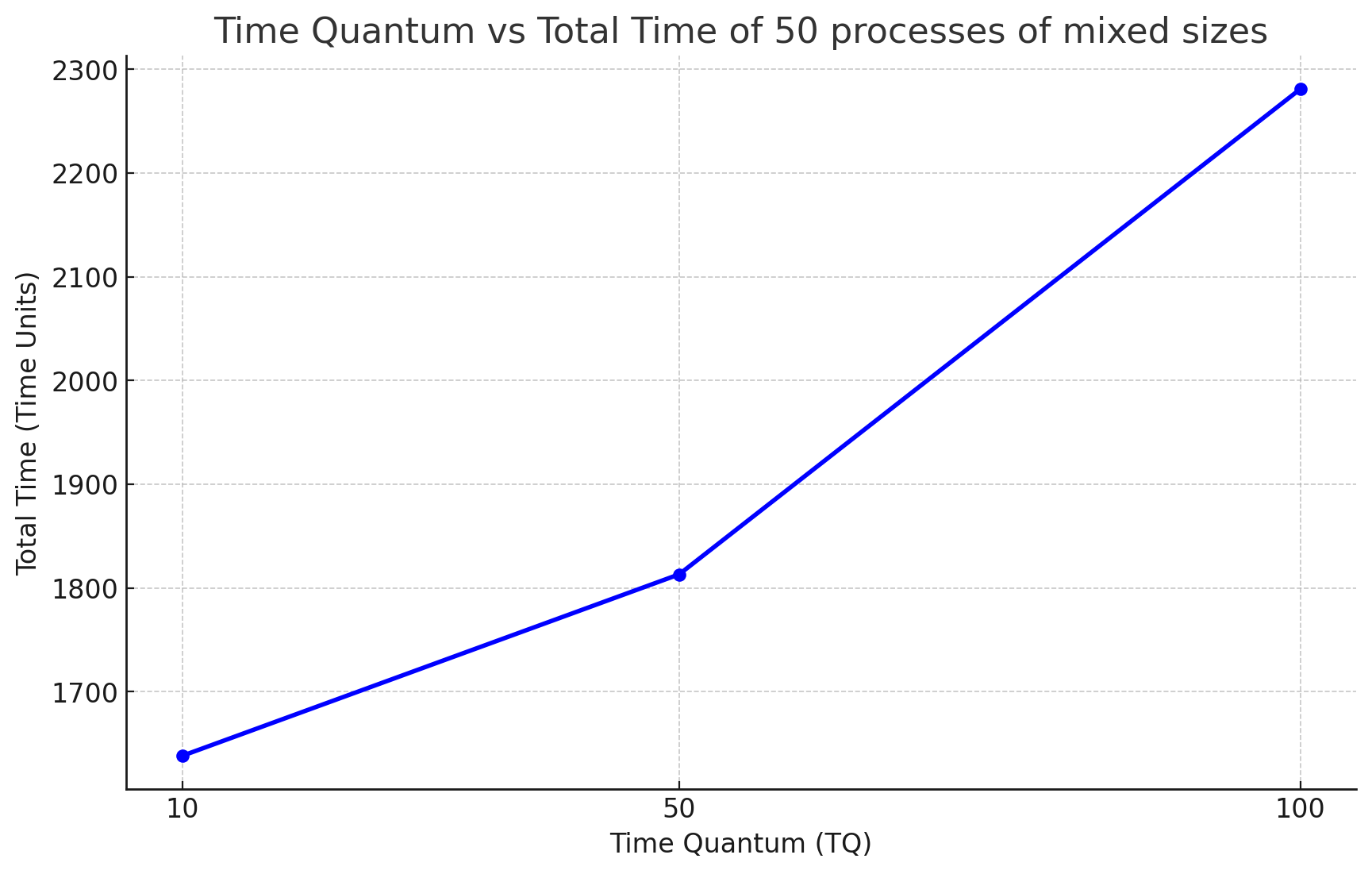
Average Waiting Time: 1198.82 time units

Average Response Time: 1204.42 time units

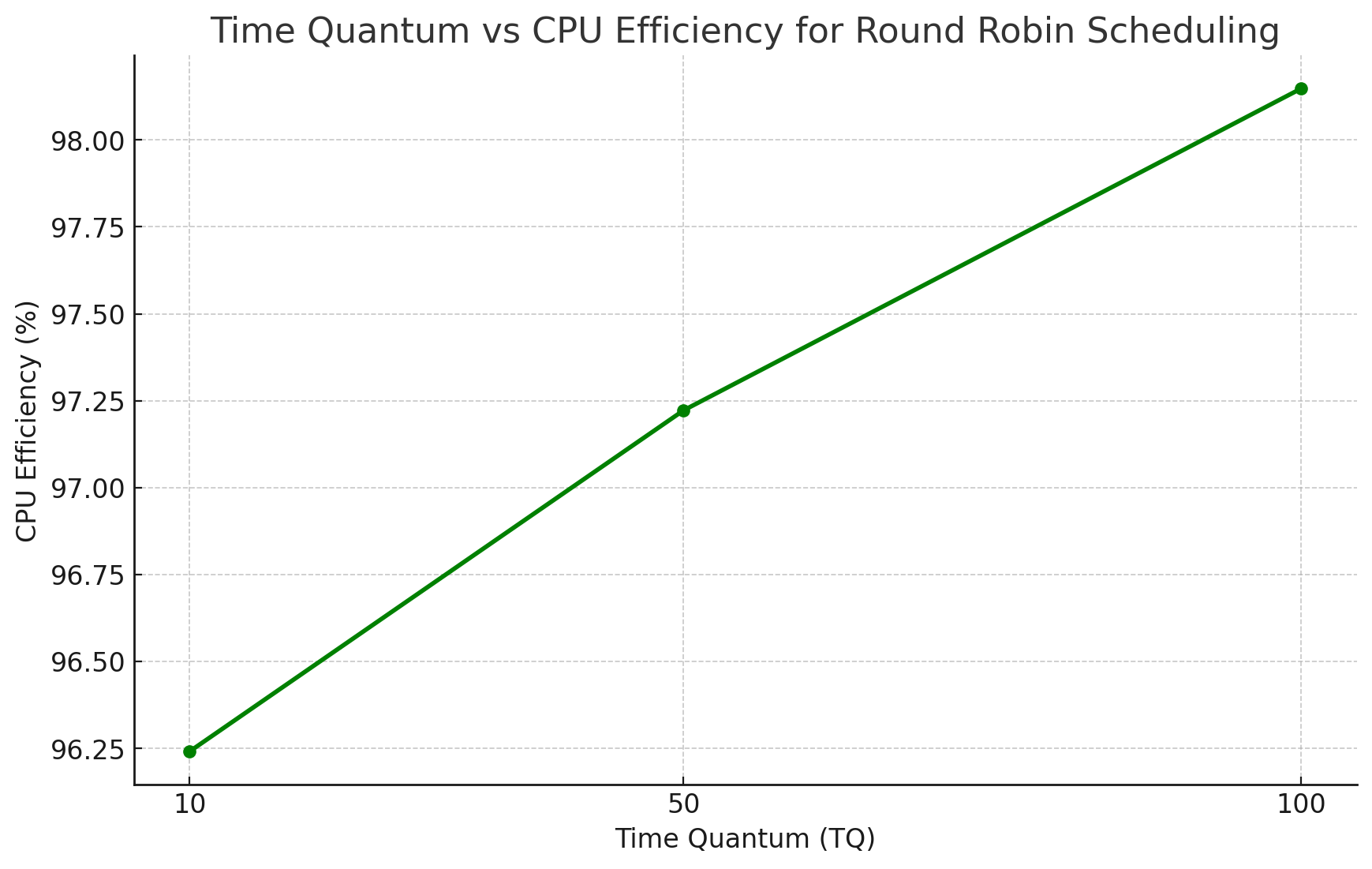
CPU Efficiency: 98.1481%

—--------------------------------------------------------------------------------------------------------------------------------------------------

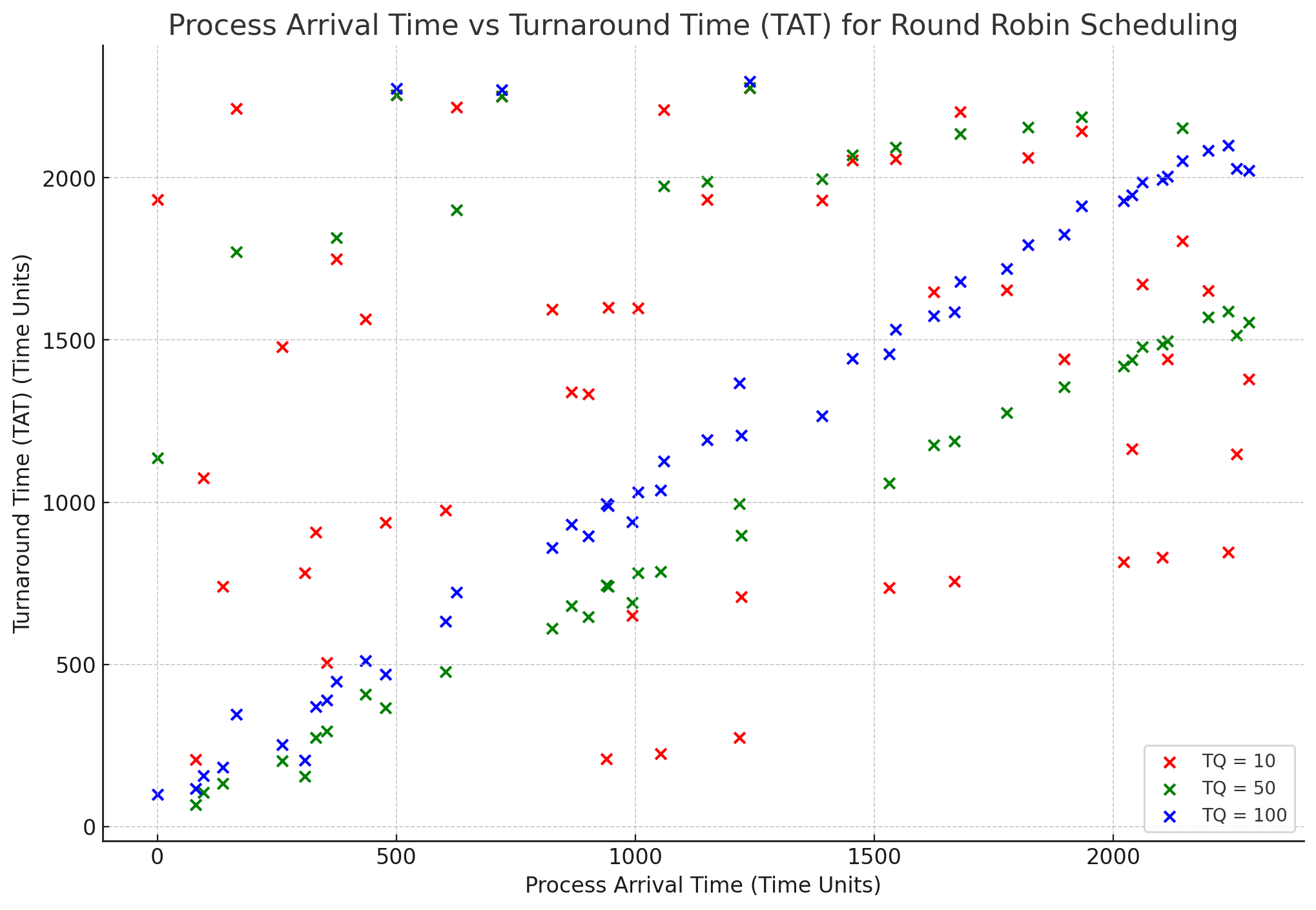
**Comparing All Three TIme Quantums of 10, 50, 100**

****

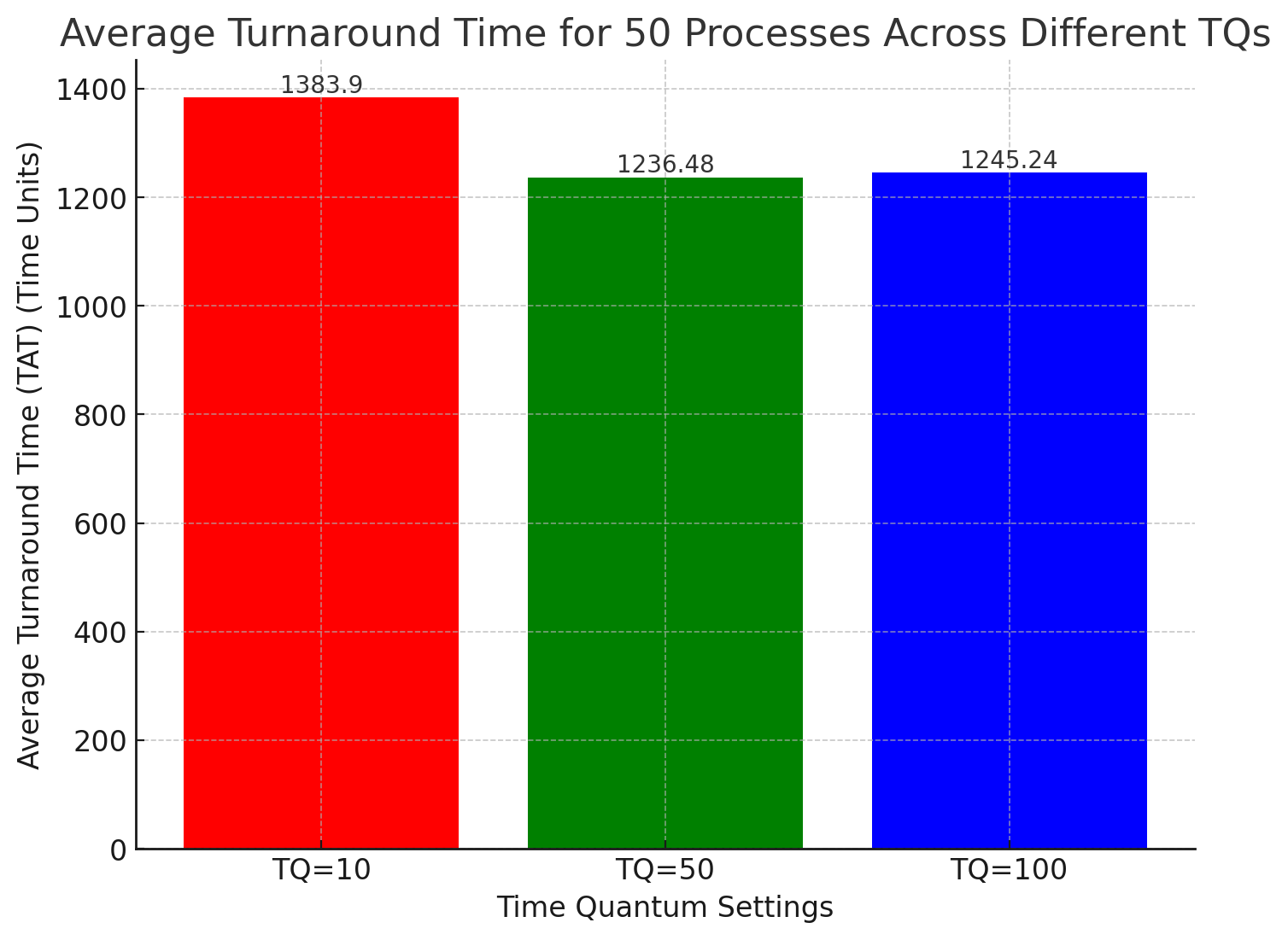
* Seems that particularly for this pool of 50 processes, a higher TQ leads to a higher total time.
* This could be attributed to smaller processes having to wait longer if bigger processes get their shared time first, and having more time to finish execution due to higher TQ, leading to more process arrival importance.



* However, even with slower times, it seemed that CPU efficiency increased as Time Quantum increased.
* This could probably be because, with the lower TQ of 10, there was more context switching/overhead. Maybe we had mostly large processes in our pool and they couldn’t finish before being context switched.
* Compared to a higher TQ they could finish in their first turn without context switching, causing less overhead and more efficiency



* We saw that process arrival time started to have more importance when TQ increased. This is due to the convoy effect increasing as wait times will be longer. However, if the pool was sorted from smallest to largest, this graph would be different and process arrival time would have less importance.



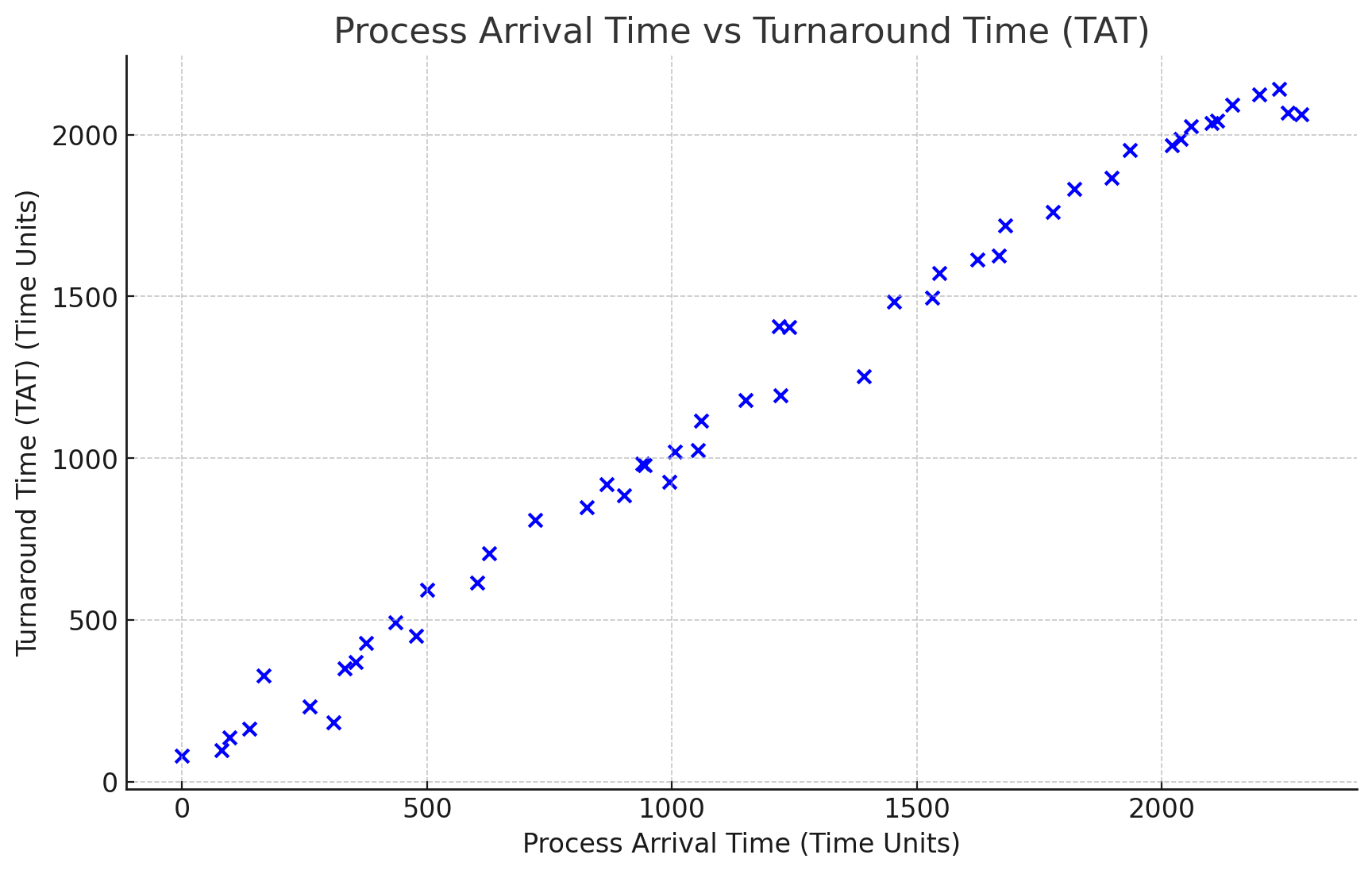
* TQ = 10 had the highest due to more context switching due to the low time frame each process is given to finish. Thus large processes will need more turns to finish their process
* TQ = 100 had the lowest due to lowest context switching, due to processes with so much time to finish, needing less context switching due to the ability to finish processes in their first turn. But with such huge times, process arrival order would matter since if big processes come first, small will have to wait a long time, compared to if small came first it would finish first and the big can execute next.

-----------------------------------------------------------------------------------------------------------------------

**FCFS**

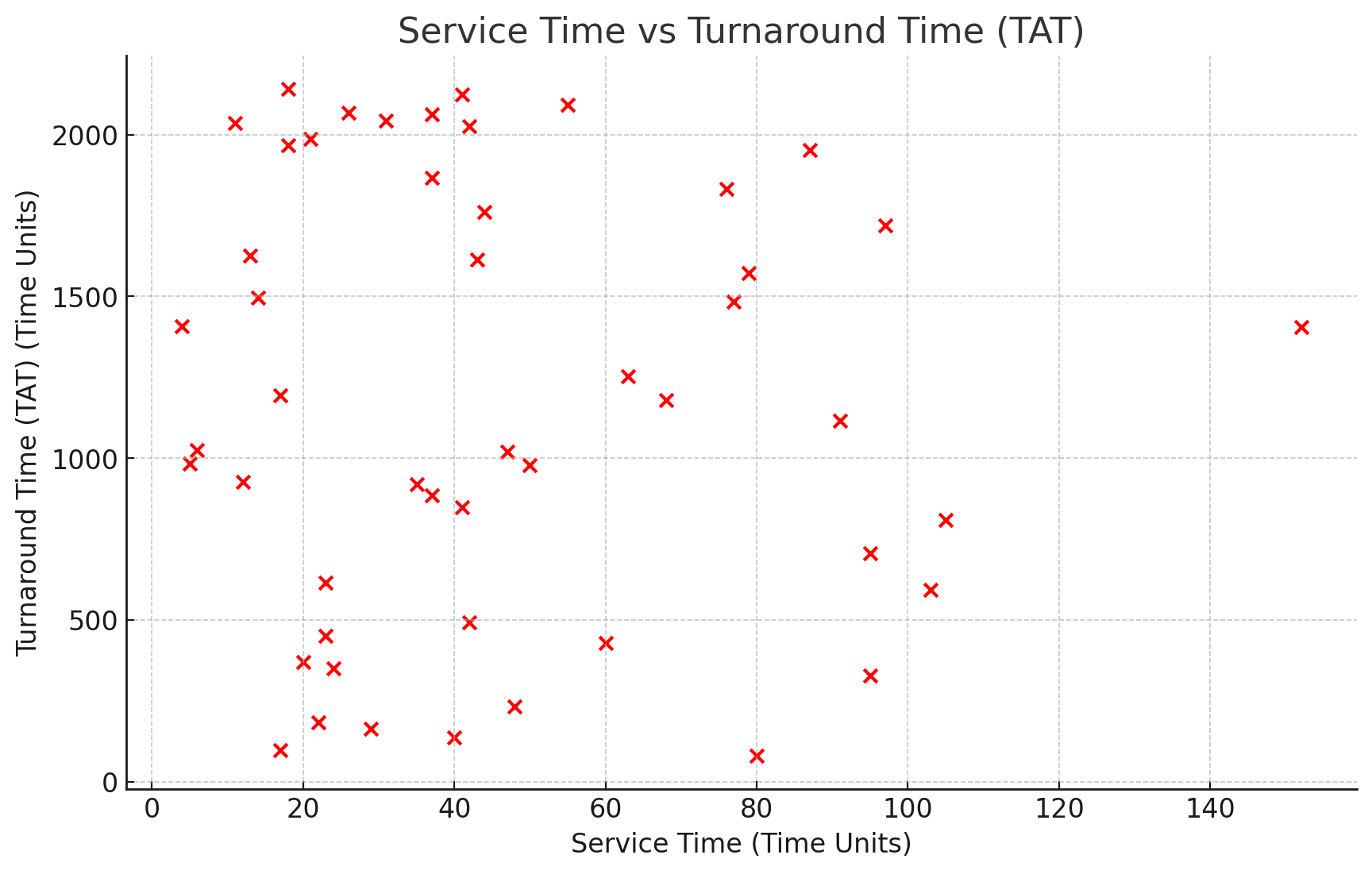
* The direct relationship between process arrival time and TAT. if 3 processes arrived at the arrival time queue at time 0, the last process would have a very high( NOT HIGHEST) TAT no matter the service time due to waiting for other processes to finish.
* There is a less average TAT compared to when RR has a Time Quantum of 2 due to no/minimal context switching/overhead, however, there are exceptions to the rule. For example, if all the biggest processes came first, avg TAT would significantly increase since the small processes waiting cannot be executed until the big processes finish executing.
* If you got lucky and your process pool was sorted from shortest jobs first to biggest last, then FCFS would have more optimal times, due to reduced convoy effect
* The convoy effect of increasing waiting times for subsequent processes due to big processes taking up time to finish executing first is a downside.
* Now let’s look at some graphs:

**Process Arrival Time vs Turnaround Time**



* As shown above, the later a process arrives, the longer it takes to finish executing. This is compliant with FCFS standards. First come first serve.

**Service Time vs Turnaround Time**



* As shown above, there is definitely no relationship between the size of a process and turnaround time. From this, we can see the downsides of FCFS, where small processes might have too large of a turnaround time and cannot finish first, due to waiting for larger processes that arrive before to finish execution. (Convoy Effect)

*Additional Data:*

Total Time required is 2321 time units

Average Turn Around Time: 1172.98-time units

Average Waiting Time: 1126.56 time units

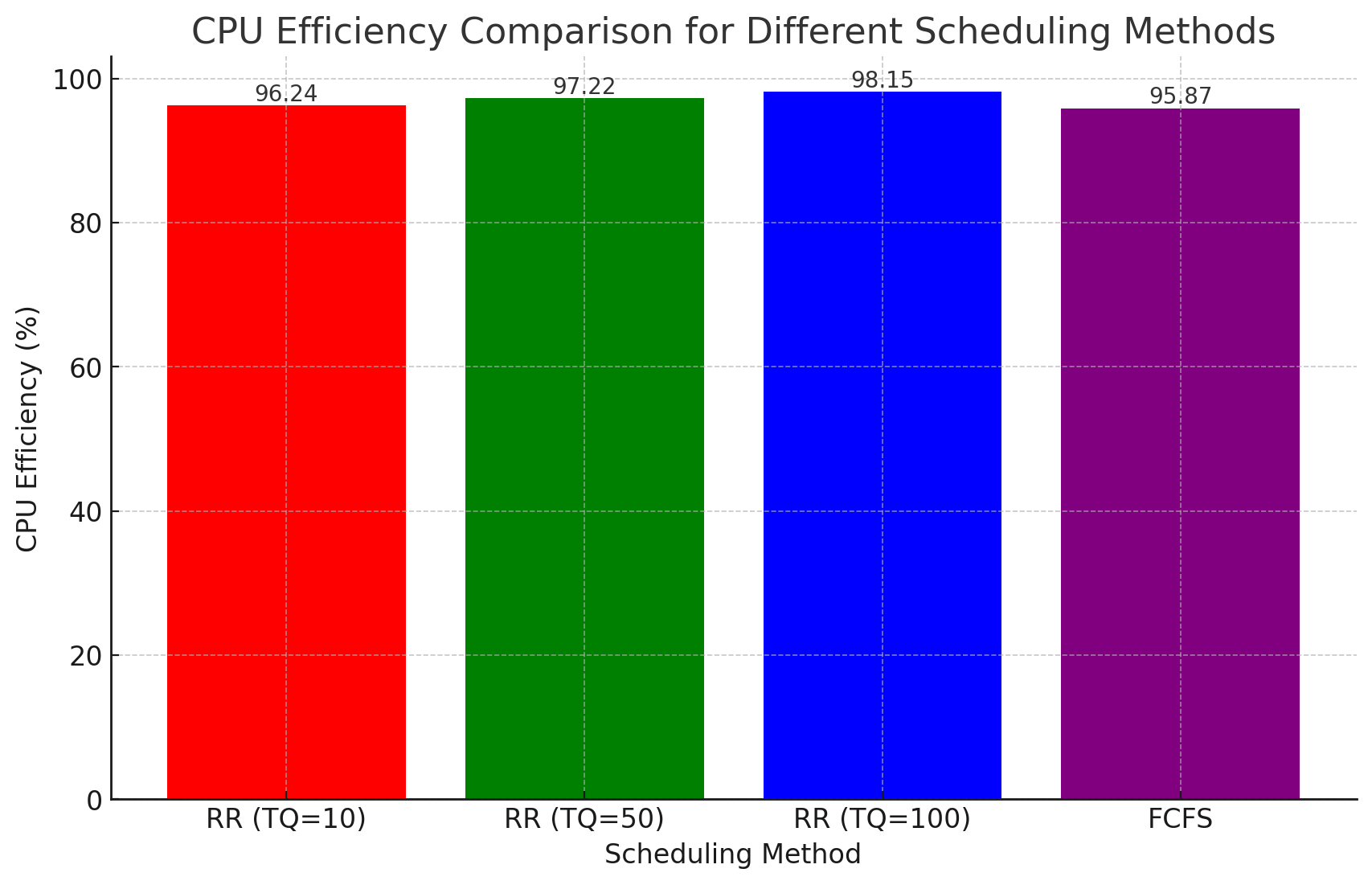
Average Response Time: 1126.56 time units

CPU Efficiency: 95.8695%

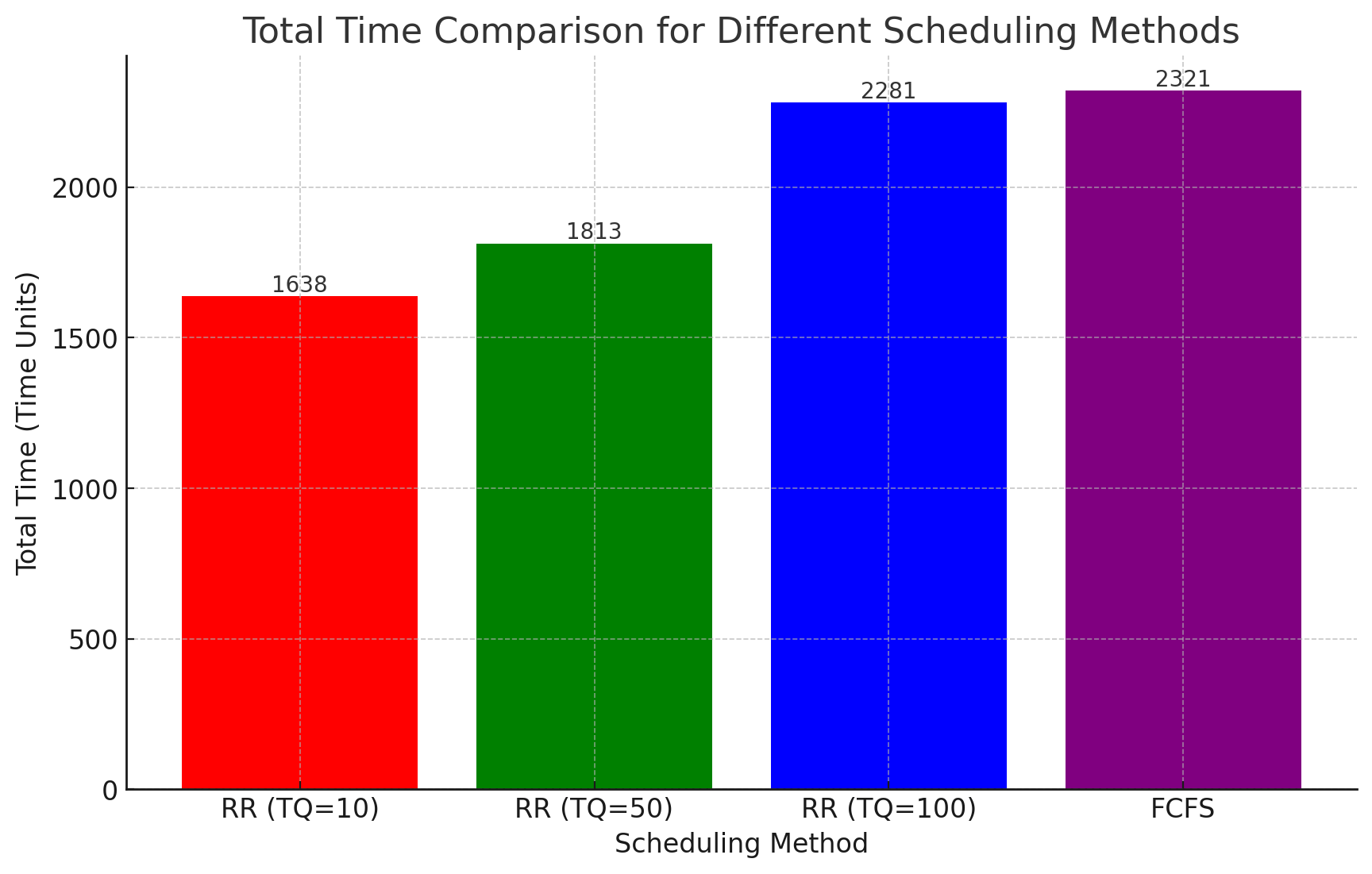
—------------------------------------------------------------------------------------------------------------------------------------------------------

**COMPARISON OF BOTH METHODS**

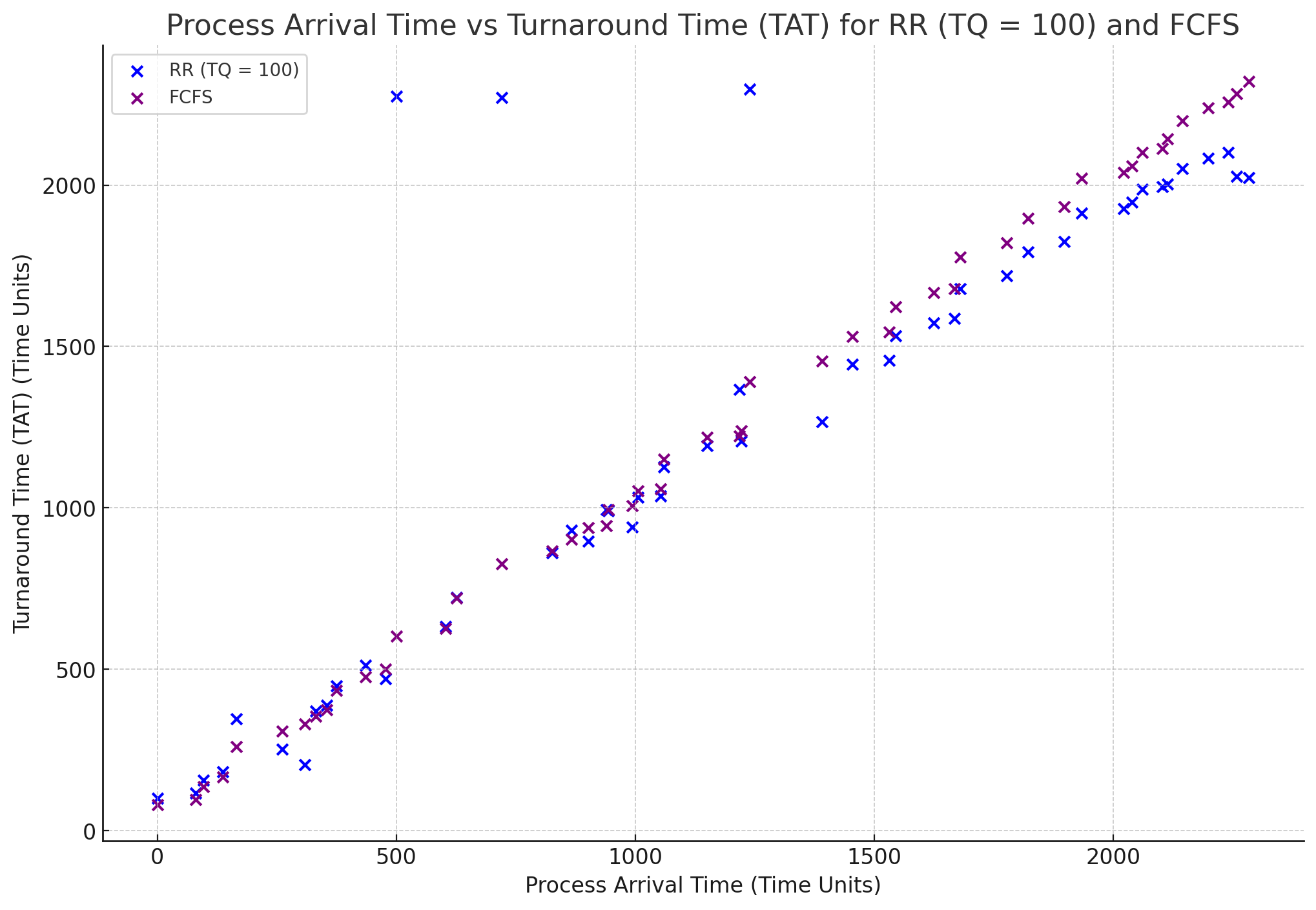
* Now, we will compare both algorithms



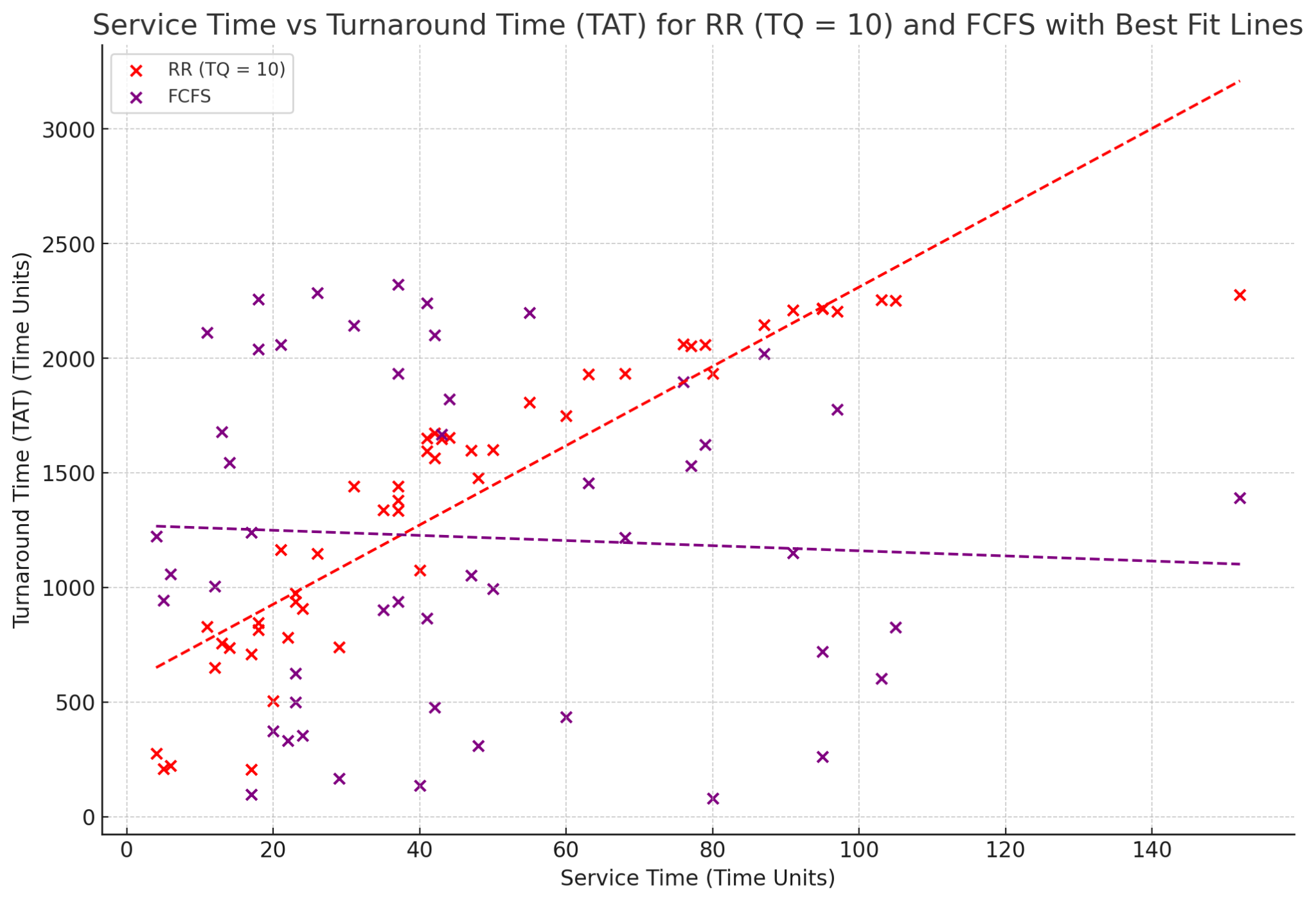
* Seems that RR beats FCFS in terms of CPU efficiency.
* Fairness and not having to wait for big processes to fully finish, and having small processes finish execution first could account for that.
* RR TQ = 100 had the highest efficiency due to the fact larger processes could finish their tasks in their first turn due to a large TQ, causing less context switching instead of not being able to finish and having to context switch and then wait to perform tasks again.



* Seems like TQ is the golden number for this pool. It is beating everything else in terms of times
* Even with the RR TQ of 100 being egregious, causing slower times and making process arrival more important like FCFS, it still beat FCFS by a little bit
* FCFS is the loser



* Going back to what we said about how TQ=100 is similar to FCFS in terms of process arrival time importance, this proves it



* I chose the best RR TQ of 10 and compared it to FCFS
* FCFS has inconsistent service times with TAT, showing the inefficiency of FCFS, not allowing the smaller processes to finish first, compared to RR TQ of 10, showing that smaller processes are able to finish first, and large ones last.

**Conclusions**

* Seemed that RR would beat FCFS no matter what, even with an unbalanced TQ
* The TQ of 10, the lowest, seemed to have the fastest TAT times, probably due to the smaller tasks being able to finish faster, but also the lowest CPU efficiency of the 3 TQs, probably due to higher context switching, and the larger tasks might not be able to finish their process in the first turn
* A larger TQ = more process arrival importance. Due to small tasks being unlucky and having to wait for big processes to finish execution due to the large amount of time they are given, this causes an increase in total time and an increase in the importance of process arrival. However, CPU efficiency is increased due to less context switching, and processes are likely to finish in their first turn.