Pauses/throughput/latency/aggressiveness

Garbage collection concerns

Garbage collection concerns

- as mentioned earlier, it is (almost) impossible to have a single, *one-size-fits-all* garbage collector suitable for all scenarios

Garbage collection concerns

- as mentioned earlier, it is (almost) impossible to have a single, *one-size-fits-all* garbage collector suitable for all scenarios
- we have different applications - different workloads, different expectations, ...

Garbage collection concerns

- as mentioned earlier, it is (almost) impossible to have a single, *one-size-fits-all* garbage collector suitable for all scenarios
- we have different applications - different workloads, different expectations, ...
- we can look at the GC from various points/expectations:

Garbage collection concerns

- as mentioned earlier, it is (almost) impossible to have a single, *one-size-fits-all* garbage collector suitable for all scenarios
- we have different applications - different workloads, different expectations, ...
- we can look at the GC from various points/expectations:
  - memory overhead - is it reclaiming memory aggressively or pretty leazily?

Garbage collection concerns

- as mentioned earlier, it is (almost) impossible to have a single, *one-size-fits-all* garbage collector suitable for all scenarios
- we have different applications - different workloads, different expectations, ...
- we can look at the GC from various points/expectations:
  - memory overhead - is it reclaiming memory aggressively or pretty leazily?
  - CPU overhead - does it consume a lot of CPU (💲)?

Garbage collection concerns

- as mentioned earlier, it is (almost) impossible to have a single, *one-size-fits-all* garbage collector suitable for all scenarios
- we have different applications - different workloads, different expectations, ...
- we can look at the GC from various points/expectations:
    - memory overhead - is it reclaiming memory aggressively or pretty leazily?
    - CPU overhead - does it consume a lot of CPU (💲)?
    - pauses - how long pauses are introduced? how often they happen?

Garbage collection concerns

- as mentioned earlier, it is (almost) impossible to have a single, *one-size-fits-all* garbage collector suitable for all scenarios
- we have different applications - different workloads, different expectations, …
- we can look at the GC from various points/expectations:
  - memory overhead - is it reclaiming memory aggressively or pretty leazily?
  - CPU overhead - does it consume a lot of CPU (💲)?
  - pauses - how long pauses are introduced? how often they happen?
  - throughput - how much requests per second we can process?

Garbage collection concerns

- as mentioned earlier, it is (almost) impossible to have a single, *one-size-fits-all* garbage collector suitable for all scenarios
- we have different applications - different workloads, different expectations, …
- we can look at the GC from various points/expectations:
  - memory overhead - is it reclaiming memory aggressively or pretty leazily?
  - CPU overhead - does it consume a lot of CPU (💲)?
  - pauses - how long pauses are introduced? how often they happen?
  - throughput - how much requests per second we can process?
  - scalability - how does GC scale with the number of CPU/RAM/requests/…?

Garbage collection concerns

- as mentioned earlier, it is (almost) impossible to have a single, *one-size-fits-all* garbage collector suitable for all scenarios
- we have different applications - different workloads, different expectations, ...
- we can look at the GC from various points/expectations:
    - memory overhead - is it reclaiming memory aggressively or pretty leazily?
    - CPU overhead - does it consume a lot of CPU (💲)?
    - pauses - how long pauses are introduced? how often they happen?
    - throughput - how much requests per second we can process?
    - scalability - how does GC scale with the number of CPU/RAM/requests/...?
    - diagnostics - is it simple to monitor what's going on?

Garbage collection concerns

- as mentioned earlier, it is (almost) impossible to have a single, *one-size-fits-all* garbage collector suitable for all scenarios
- we have different applications - different workloads, different expectations, ...
- we can look at the GC from various points/expectations:
    - memory overhead - is it reclaiming memory aggressively or pretty leazily?
    - CPU overhead - does it consume a lot of CPU (💲)?
    - pauses - how long pauses are introduced? how often they happen?
    - throughput - how much requests per second we can process?
    - scalability - how does GC scale with the number of CPU/RAM/requests/...?
    - diagnostics - is it simple to monitor what's going on?
    - customization - are we able to customize/tune it?

Garbage collection concerns

- as mentioned earlier, it is (almost) impossible to have a single, *one-size-fits-all* garbage collector suitable for all scenarios
- we have different applications - different workloads, different expectations, ...
- we can look at the GC from various points/expectations:
  - **memory overhead - is it reclaiming memory aggressively or pretty lazily?**
  - **CPU overhead - does it consume a lot of CPU (💲)?**
  - **pauses - how long pauses are introduced? how often they happen?**
  - **throughput - how much requests per second we can process?**
  - scalability - how does GC scale with the number of CPU/RAM/requests/...?
  - diagnostics - is it simple to monitor what's going on?
  - customization - are we able to customize/tune it?
- and that's what this lesson (an the whole course) is about 😎

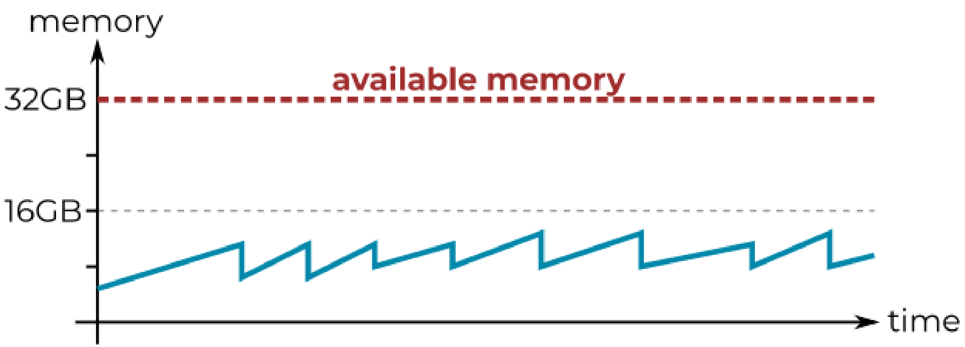
Memory overhead

An obvious metric - how much memory is being consumed when running **the same** program:

Memory overhead

An obvious metric - how much memory is being consumed when running **the same** program:

"Good" GC

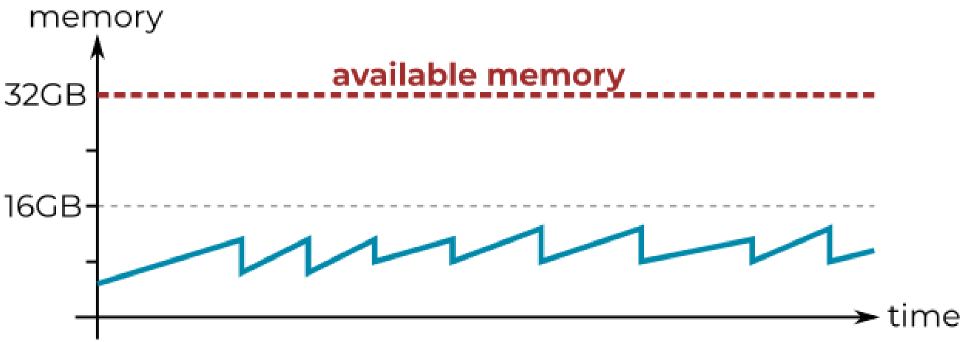


It keeps memory usage constantly small

Memory overhead

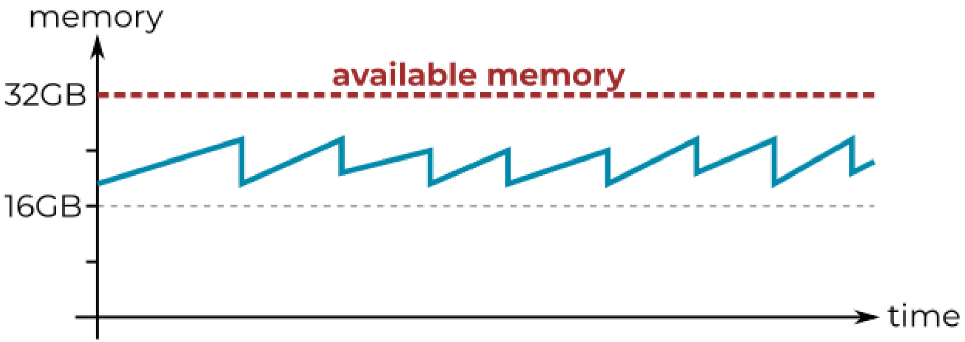
An obvious metric - how much memory is being consumed when running **the same** program:

"Good" GC



It keeps memory usage constantly small

"Bad" GC

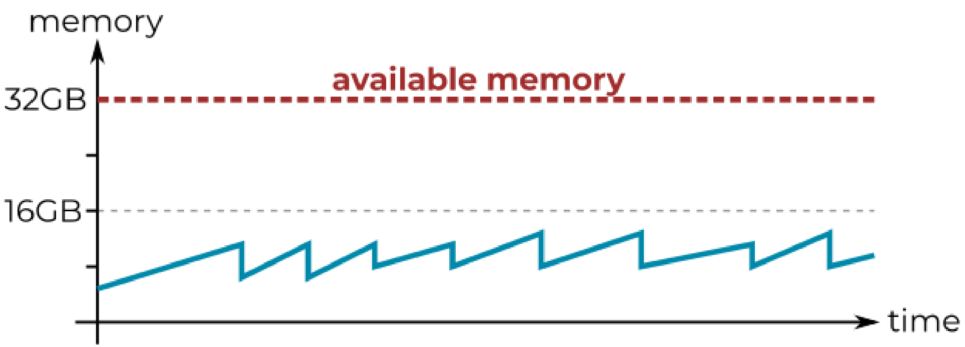


It keeps memory usage significantly bigger all the time

Memory overhead

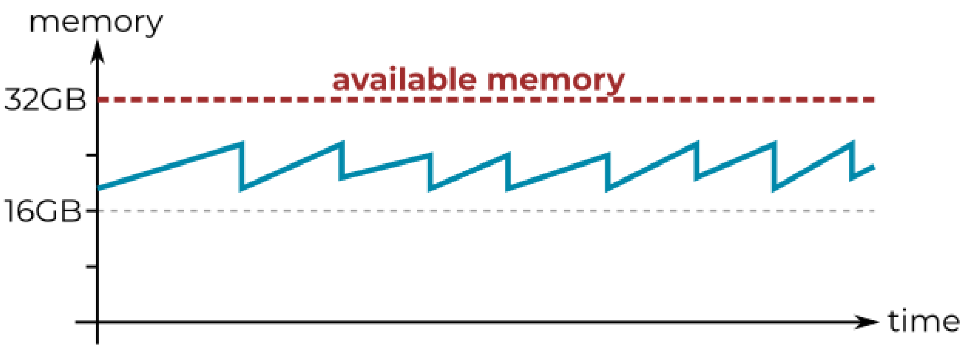
An obvious metric - how much memory is being consumed when running **the same** program:

"Good" GC



It keeps memory usage constantly small

"Bad" GC



It keeps memory usage significantly bigger all the time

- but... it is not so easy to say "good" or "bad" - what's **the cost** of smaller memory usage? CPU? application slowdown?
- if and only if all is the same except memory usage - we have better or worse GC

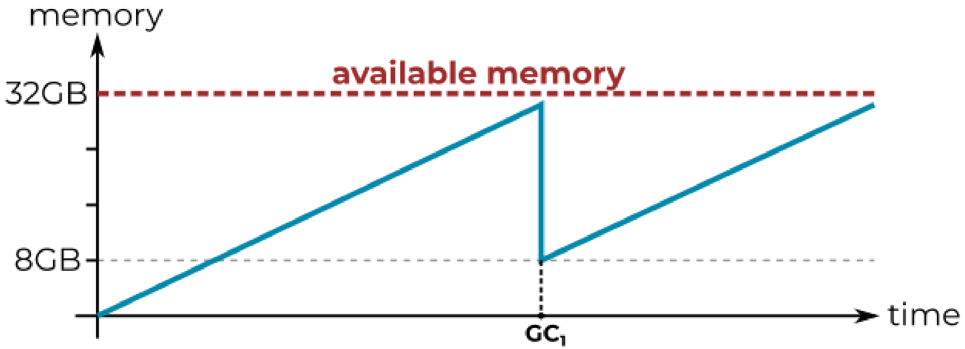
Memory overhead

More often, we talk about GC's **aggressiveness** - how often and how much memory it is reclaiming:

Memory overhead

More often, we talk about GC's **aggressiveness** - how often and how much memory it is reclaiming:

Less aggressive GC

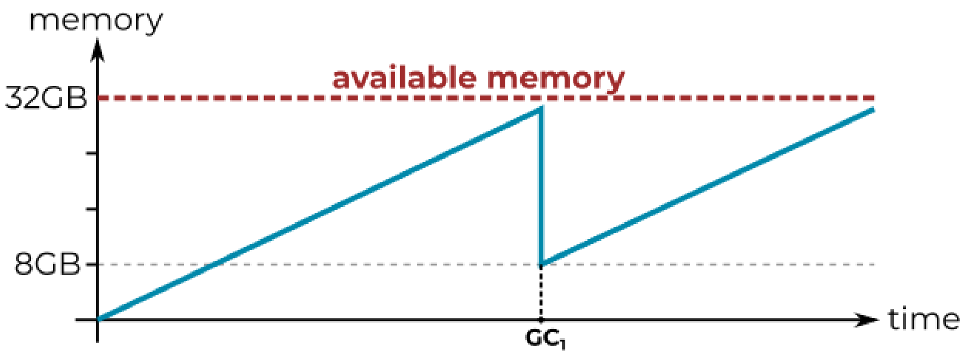


- allows to accumulate a lot of garbage - often close to the limits
- does GC very rarely - reclaiming a lot of memory at once (it may take a lot of time)

Memory overhead

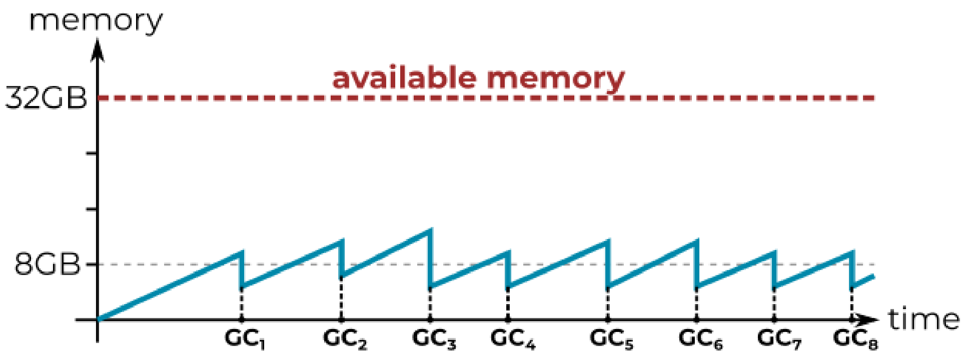
More often, we talk about GC's **aggressiveness** - how often and how much memory it is reclaiming:

Less aggressive GC



- allows to accumulate a lot of garbage - often close to the limits
- does GC very rarely - reclaiming a lot of memory at once (it may take a lot of time)

More aggressive GC

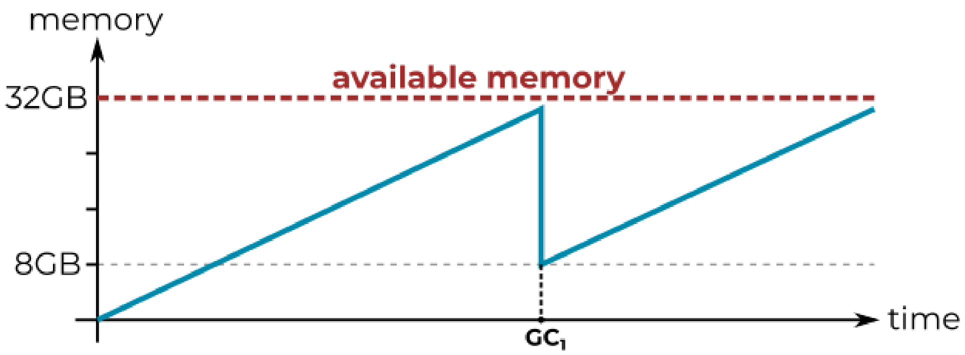


- does GC very often - reclaiming memory in small batches (it is rather fast)
- memory footprint is significantly smaller

Memory overhead

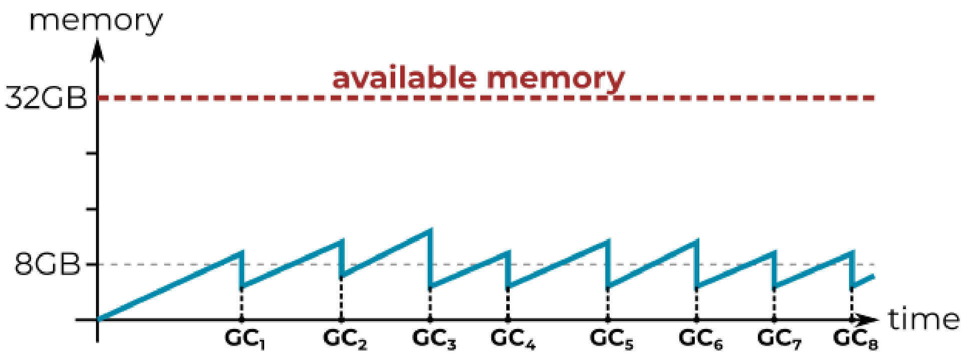
More often, we talk about GC's **aggressiveness** - how often and how much memory it is reclaiming:

Less aggressive GC



- allows to accumulate a lot of garbage - often close to the limits
- does GC very rarely - reclaiming a lot of memory at once (it may take a lot of time)

More aggressive GC



- does GC very often - reclaiming memory in small batches (it is rather fast)
- memory footprint is significantly smaller

Aggressiveness influences other metrics - the less frequent GCs, the bigger overhead they introduce (longer pauses, bigger CPU spikes).

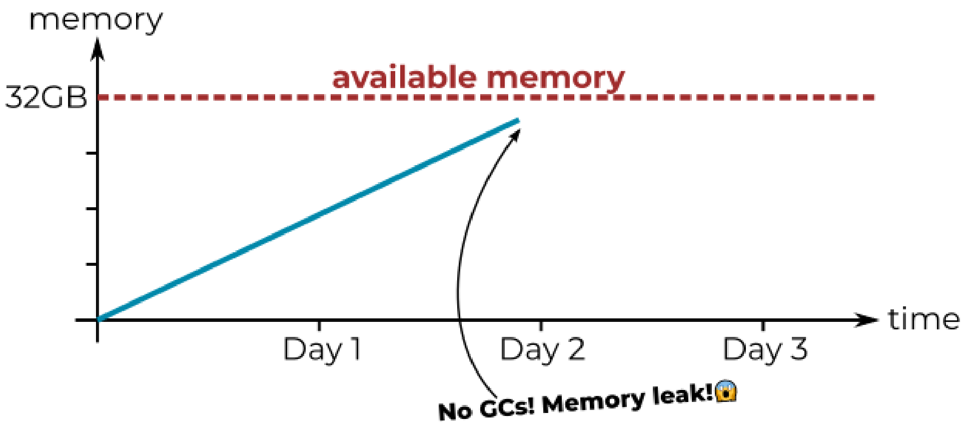
Memory overhead

Aggressiveness is a pretty frequent source of confusion:

Memory overhead

Aggressiveness is a pretty frequent source of confusion:

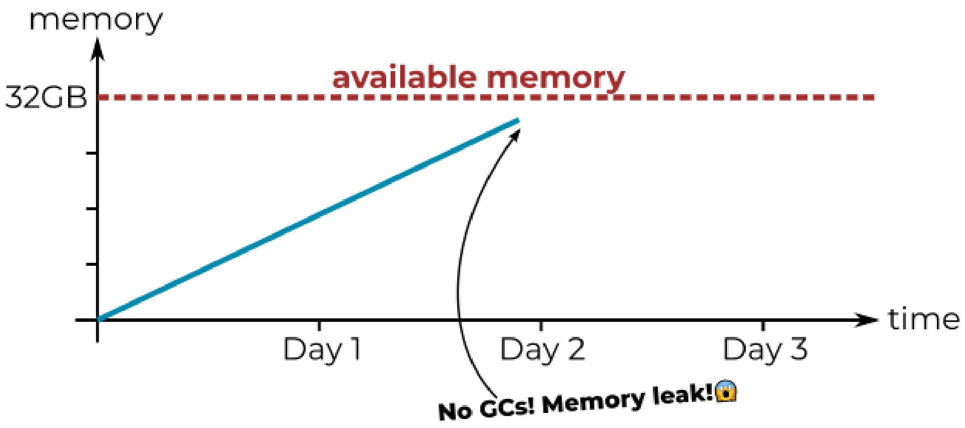
1. We see a lot of memory growing up to GBs (and sometimes even no GCs at all) and treat it as a memory leak!



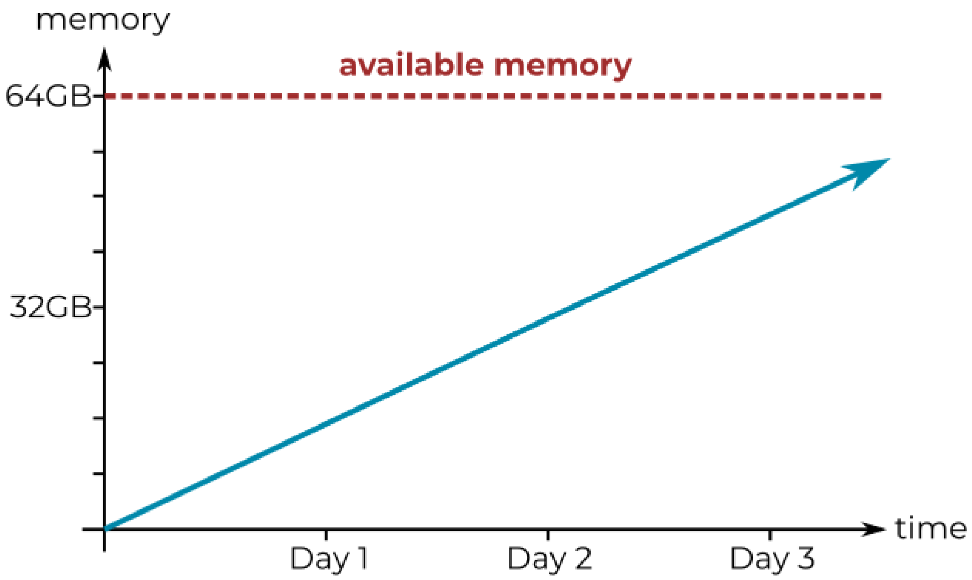
Memory overhead

Aggressiveness is a pretty frequent source of confusion:

1. We see a lot of memory growing up to GBs (and sometimes even no GCs at all) and treat it as a memory leak!



2. In fear, we add more RAM to avoid killing the process and "the leak" becomes even bigger!



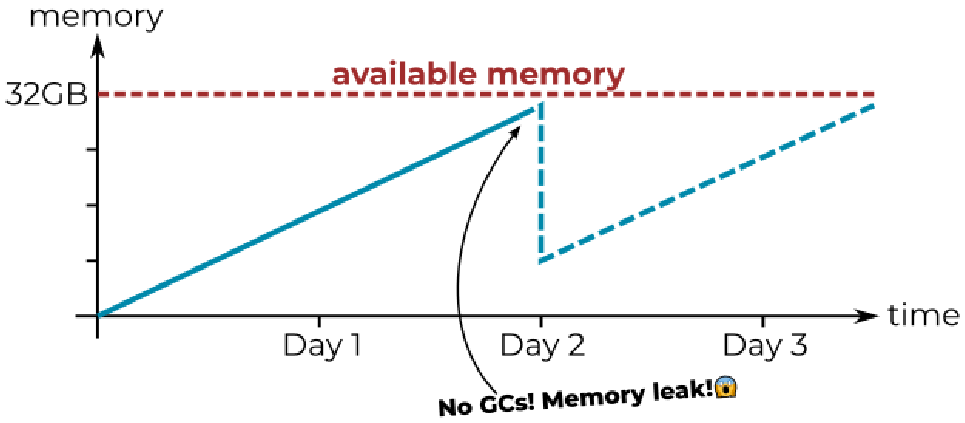
Memory overhead

Aggressiveness is a pretty frequent source of confusion (cont.):

Memory overhead

Aggressiveness is a pretty frequent source of confusion (cont.):

3. If we waited a little longer, the GC would be eventually called (*):

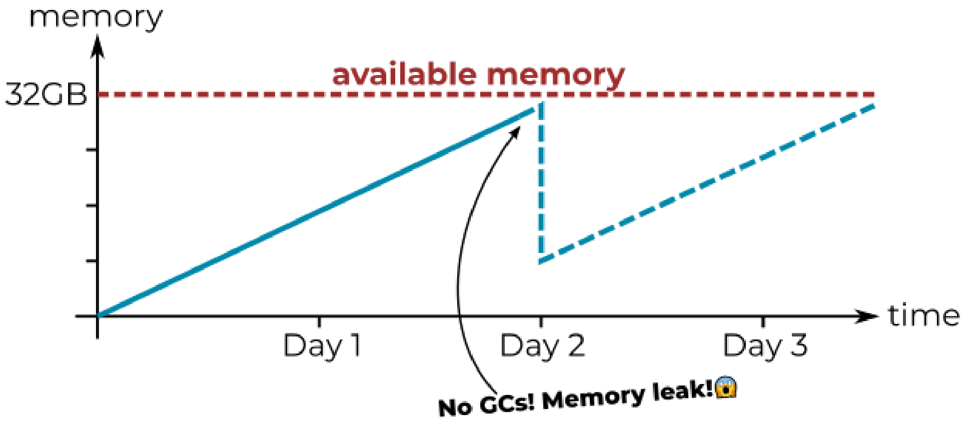


* unless we really have a memory leak, not aggressiveness misunderstanding

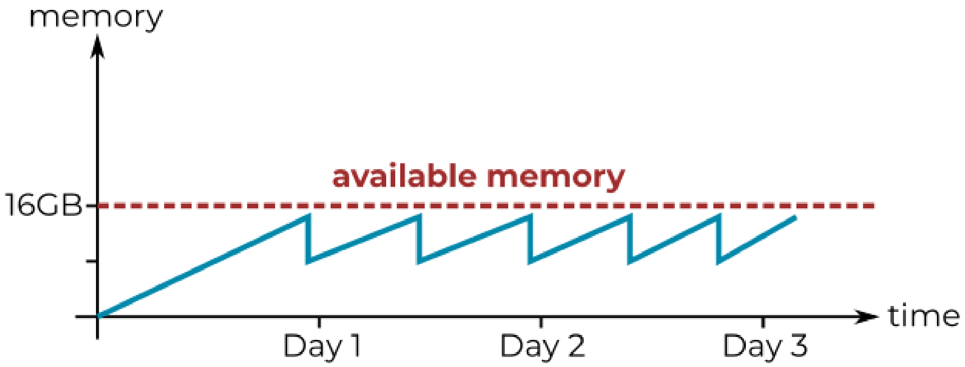
Memory overhead

Aggressiveness is a pretty frequent source of confusion (cont.):

3. If we waited a little longer, the GC would be eventually called (*):



4. Or we could solve "the memory leak" by reducing available RAM:

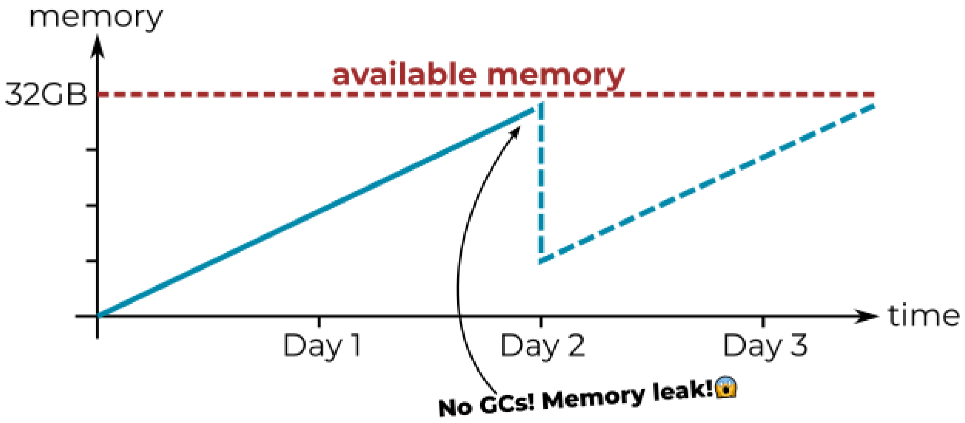


* unless we really have a memory leak, not aggressiveness misunderstanding

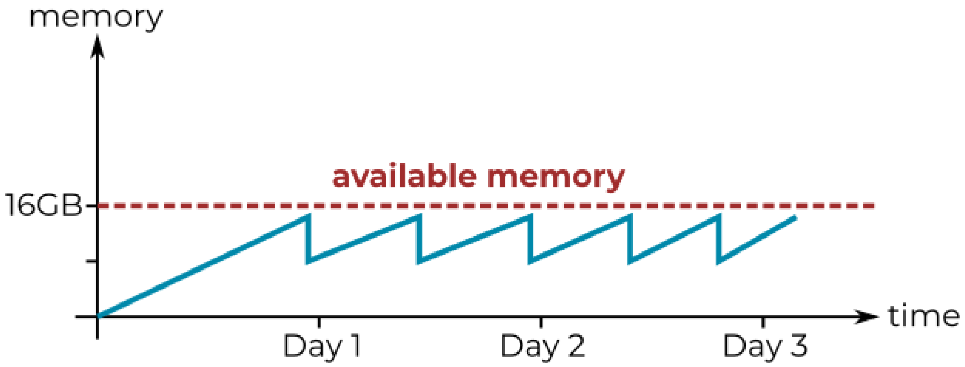
Memory overhead

Aggressiveness is a pretty frequent source of confusion (cont.):

3. If we waited a little longer, the GC would be eventually called (*):



4. Or we could solve "the memory leak" by reducing available RAM:



* unless we really have a memory leak, not aggressiveness misunderstanding

GC's aggressiveness may be its **implementation detail**, but we can have some **control** over it - we will return to that when understanding the GC a little more 😇

Memory overhead

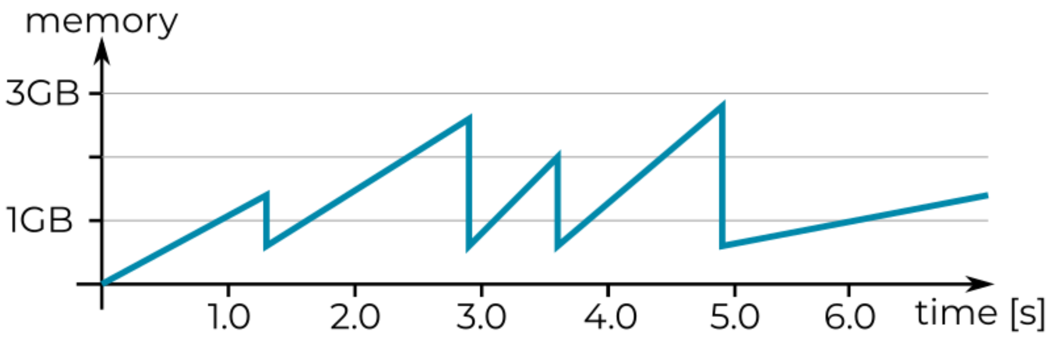
How to measure?

Memory overhead

How to measure? We will return to that when understanding GC more. In general - it is *"how much memory does this process use?"* question.

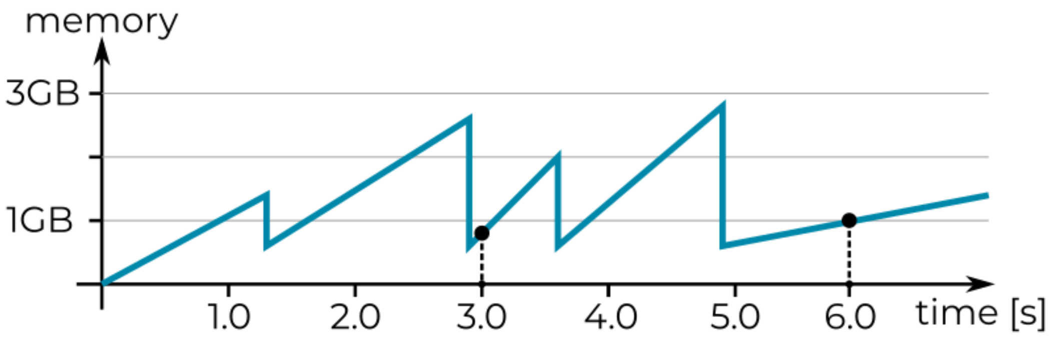
Memory overhead

But, remember about one little fact. Memory usage change in time due to allocations & GC:



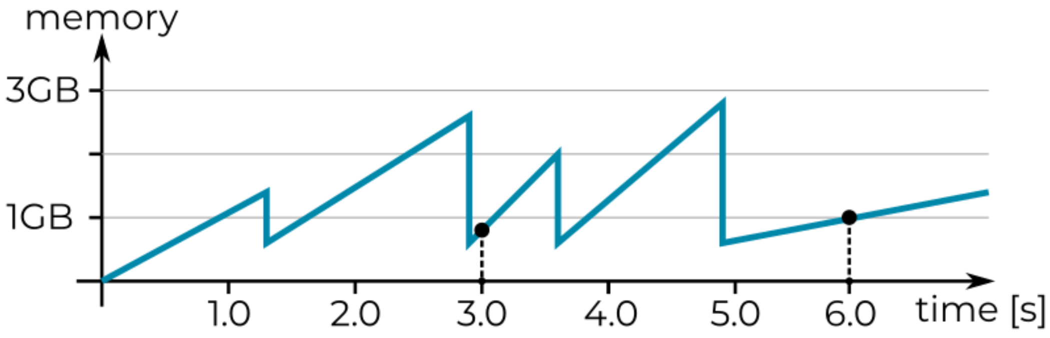
Memory overhead

If we take a memory dumps at some random times:



Memory overhead

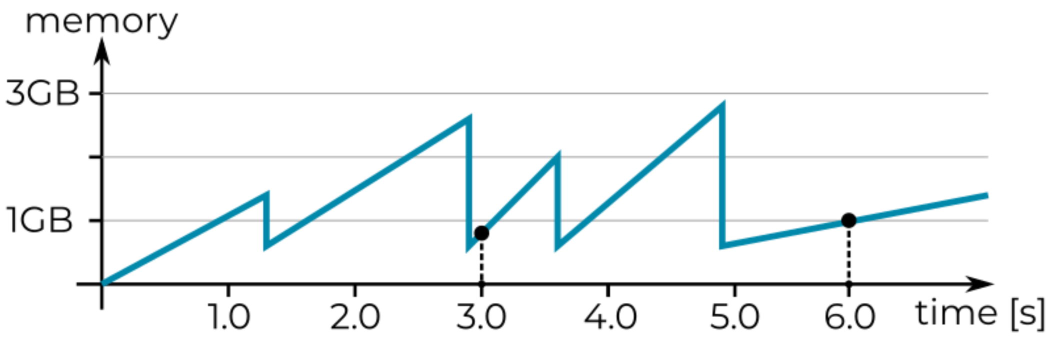
If we take a memory dumps at some random times:



.. we will see memory usage of around 1GB, not considering the dynamic nature of the GC!

Memory overhead

If we take a memory dumps at some random times:



.. we will see memory usage of around 1GB, not considering the dynamic nature of the GC!

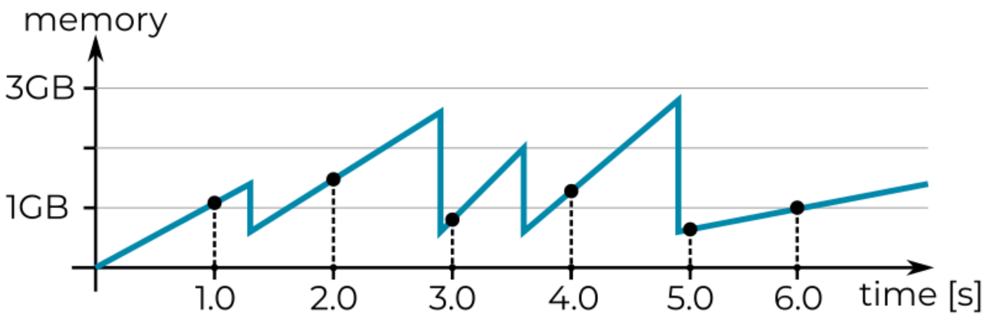
(and I see it surprisingly often to analyze memory dump without knowledge how it relates to the surrounding GCs)

Memory overhead

If we measure on some regular, timely basis:

Memory overhead

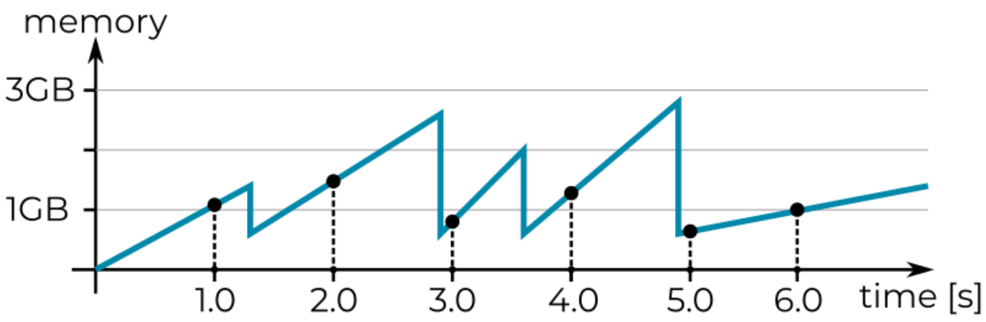
If we measure on some regular, timely basis:

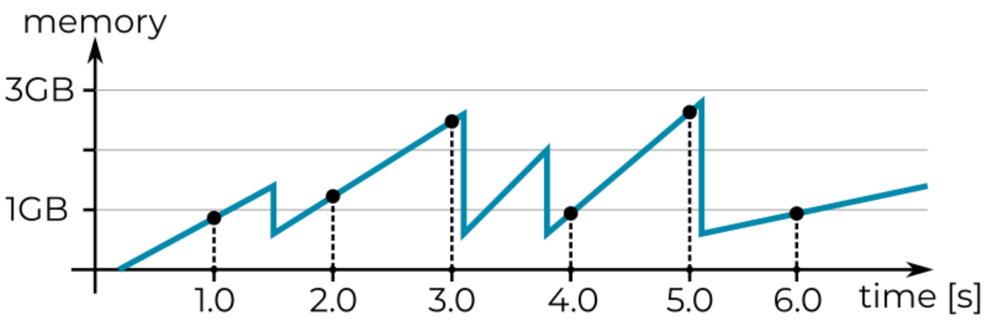


we will see 1, 1.5, 0.8, 1.2, 0.6, 1 GB memory usage

Memory overhead

If we measure on some regular, timely basis:



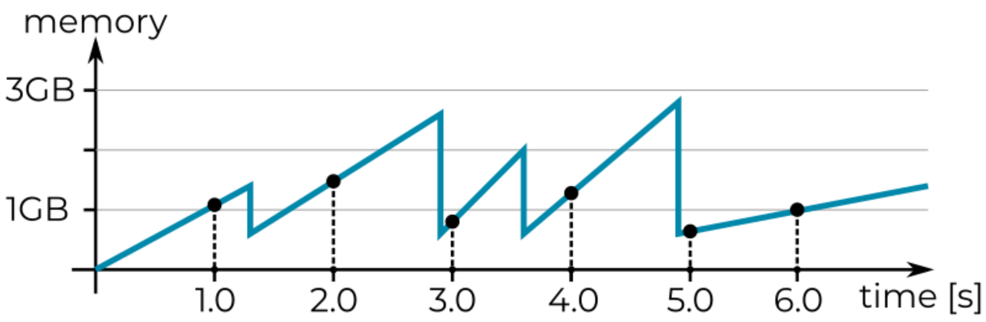


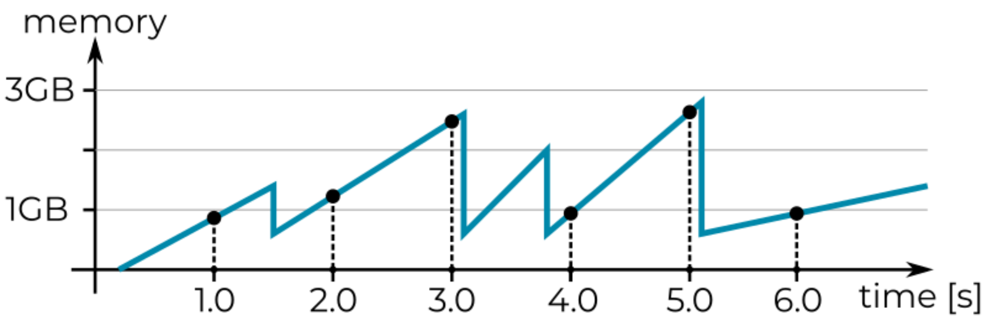
we will see 1, 1.5, 0.8, 1.2, 0.6, 1 GB memory usage

or 0.9, 1.2, **2.5**, 1.0, **2.6**, 1.0 GB. It really depends how "lucky" we are.

Memory overhead

If we measure on some regular, timely basis:





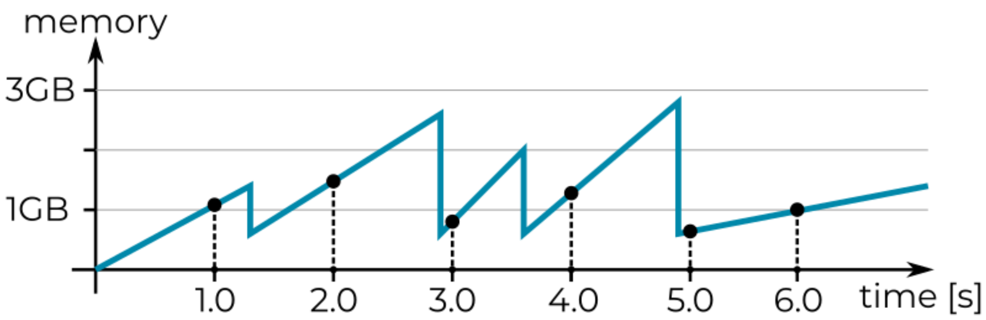
we will see 1, 1.5, 0.8, 1.2, 0.6, 1 GB memory usage

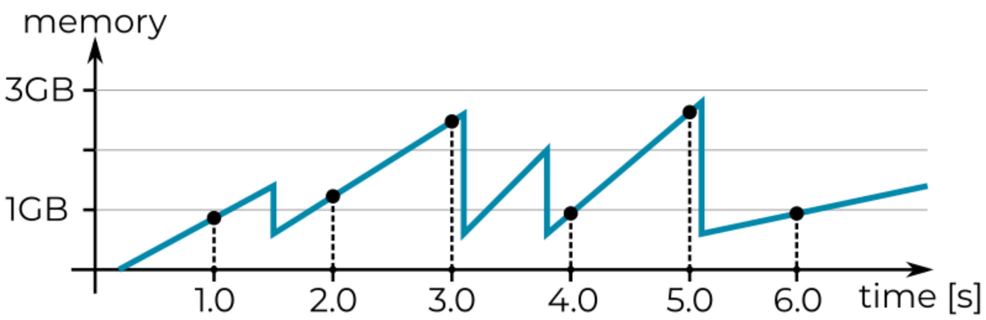
or 0.9, 1.2, **2.5**, 1.0, **2.6**, 1.0 GB. It really depends how "lucky" we are.

- this is a typical *sampling problem* - we may sample very often...

Memory overhead

If we measure on some regular, timely basis:

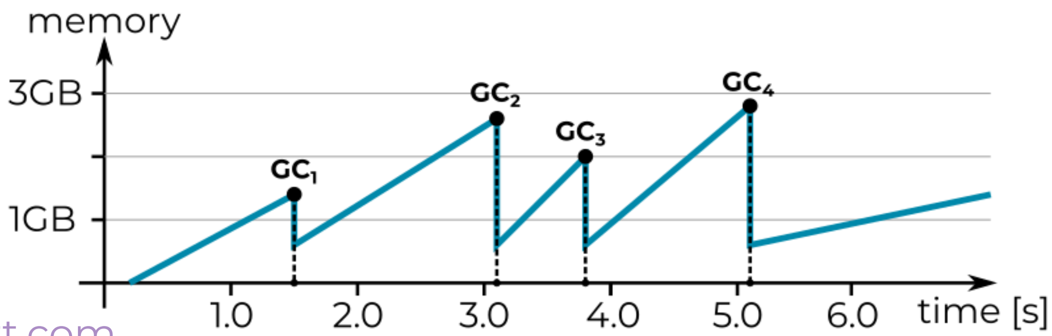




we will see 1, 1.5, 0.8, 1.2, 0.6, 1 GB memory usage

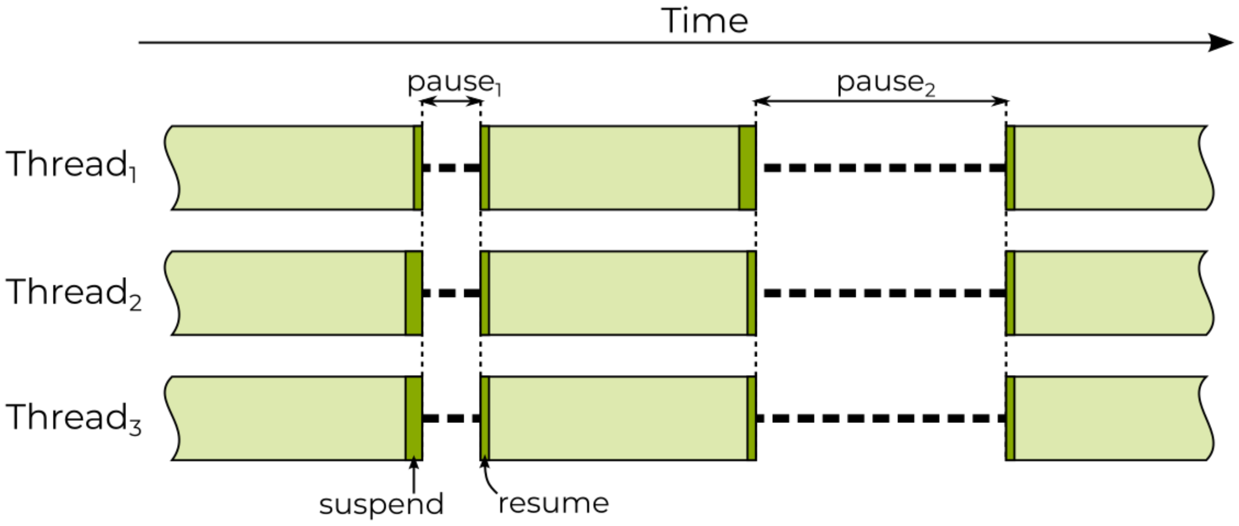
or 0.9, 1.2, **2.5**, 1.0, **2.6**, 1.0 GB. It really depends how "lucky" we are.

- this is a typical *sampling problem* - we may sample very often...
- or even better - measure with respect to the GC to know exact values **before** and **after**:



Pauses

All GCs need to occasionally pause an application (threads) for shorter or longer amount of time:

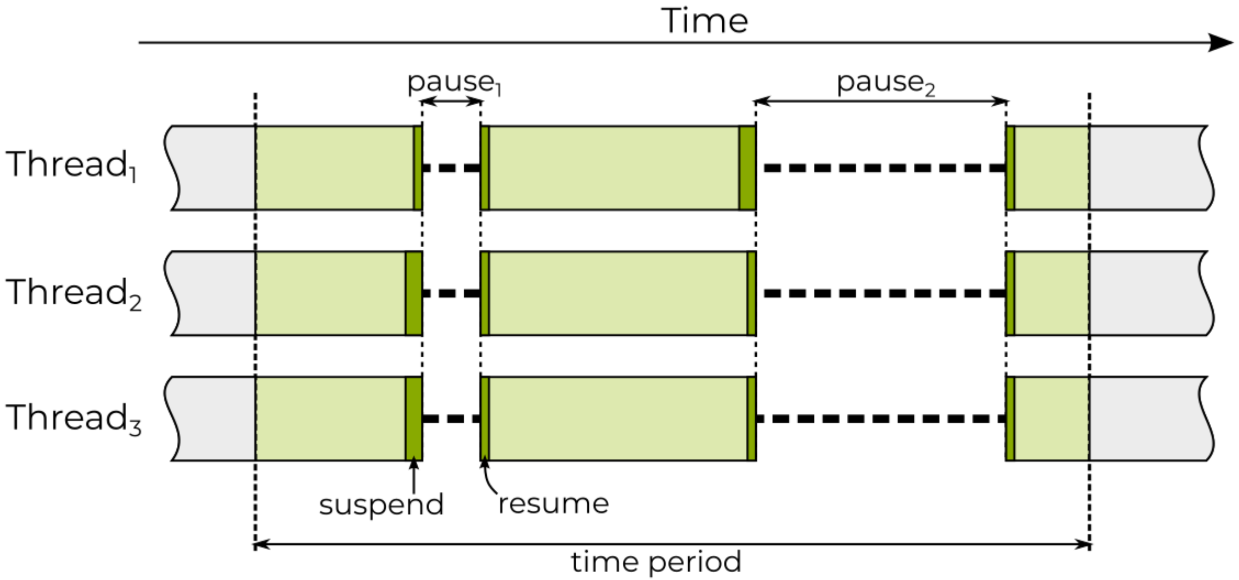


This is typically not-so-good:

- in interactive application - it affects the user who may notice worse responsiveness
- in request processing application (like web) - it affects some requests processing times

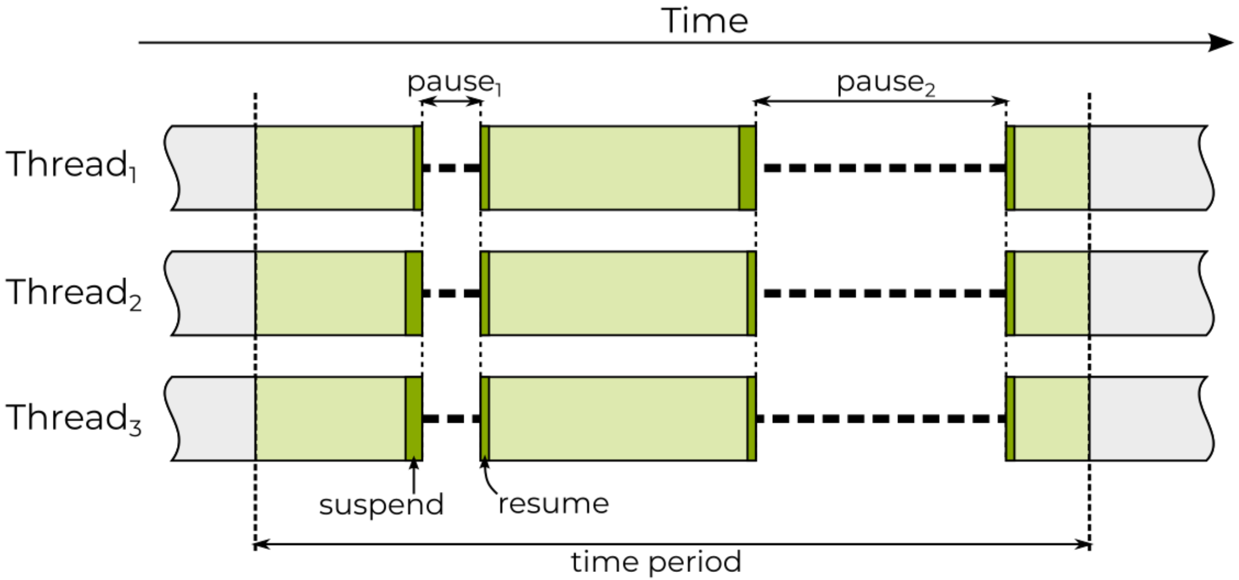
Pauses

We can intuitively reflect it by relative pause time of the application:



Pauses

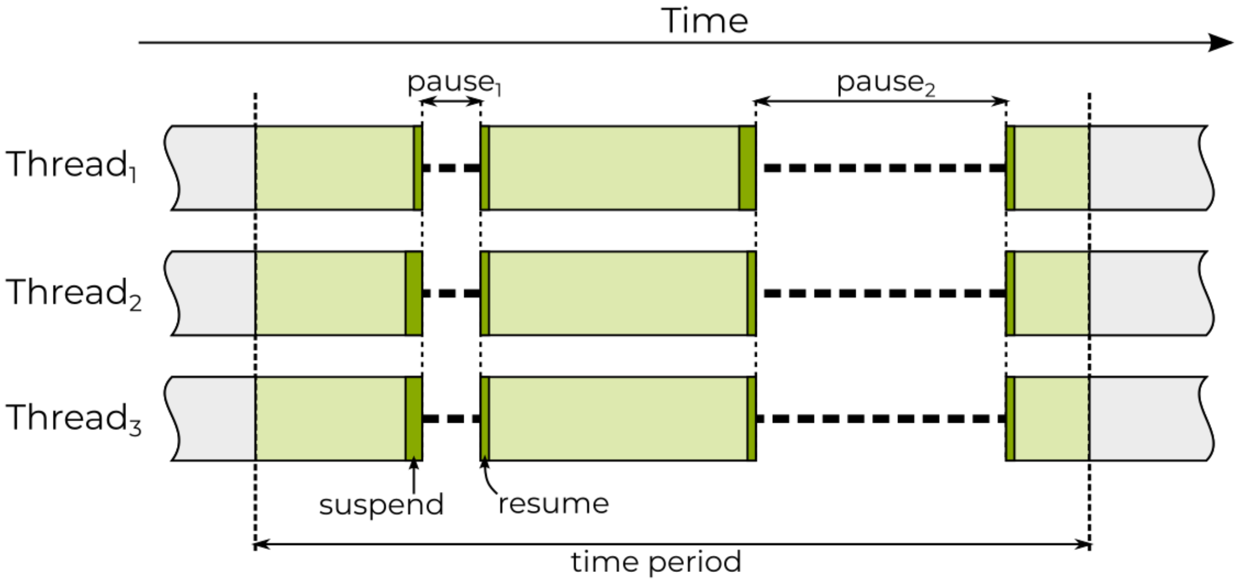
We can intuitively reflect it by relative pause time of the application:



$$\% \text{ Pause time in GC} = \frac{\text{pause}_1 + \text{pause}_2}{\text{time period}}$$

Pauses

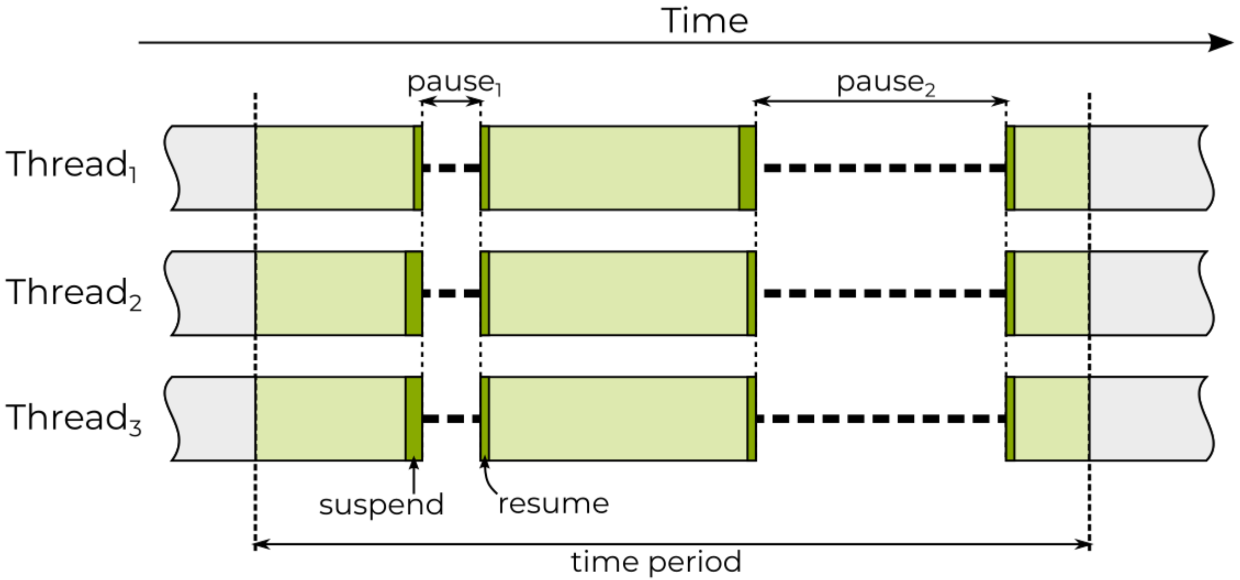
We can intuitively reflect it by relative pause time of the application:



$$\% \text{ Pause time in GC} = \frac{\text{pause}_1 + \text{pause}_2}{\text{time period}} = \frac{0.7 + 3.0}{10.0} = 37\%$$

Pauses

We can intuitively reflect it by relative pause time of the application:

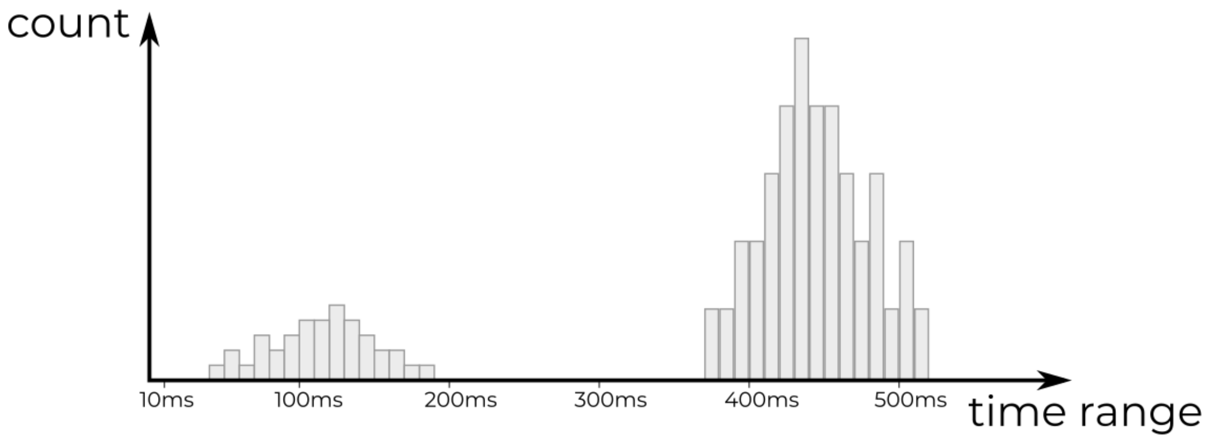


$$\% \text{ Pause time in GC} = \frac{pause_1 + pause_2}{\text{time period}} = \frac{0.7 + 3.0}{10.0} = 37\%$$

"Usually a well behaved app has < 5% Pause time in GC while it's actively handling workloads."

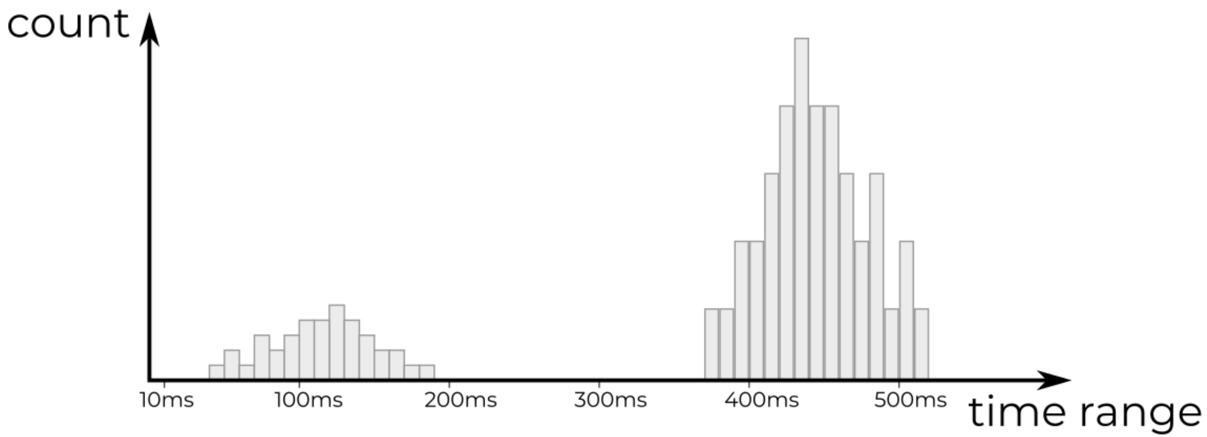
Pauses

An interesting point of view is to see **pauses histogram** - it may reveal a little bit more complex (and useful) image that a single *% Pause Time in GC*:



Pauses

An interesting point of view is to see **pauses histogram** - it may reveal a little bit more complex (and useful) image that a single *% Pause Time in GC*:

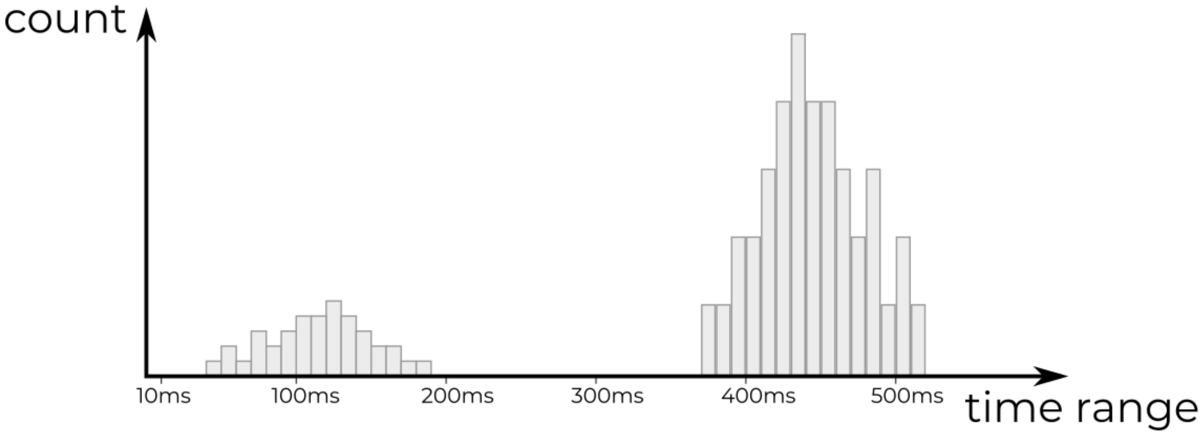


Here, for example, we see that there are two common, separate "types" of pauses:

- shorter pauses between 50-200 ms
- pretty a lot of big pauses around 400-500 ms

Pauses

An interesting point of view is to see **pauses histogram** - it may reveal a little bit more complex (and useful) image that a single *% Pause Time in GC*:



Here, for example, we see that there are two common, separate "types" of pauses:

- shorter pauses between 50-200 ms
- pretty a lot of big pauses around 400-500 ms

Obviously, we may be interested in reducing those longer ones. Especially since they affect so-called **tail latency**.

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:
  - request processing latency - in apps processing requests like web apps, it is the time between issuing a request and getting an answer. In our GC-related case, without network latency

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:
  - request processing latency - in apps processing requests like web apps, it is the time between issuing a request and getting an answer. In our GC-related case, without network latency
  - user interface latency - in desktop/mobile apps, it is the time between interacting with the UI (clicking something) and seeing a result

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:
  - request processing latency - in apps processing requests like web apps, it is the time between issuing a request and getting an answer. In our GC-related case, without network latency
  - user interface latency - in desktop/mobile apps, it is the time between interacting with the UI (clicking something) and seeing a result

    (well, in the end it also pretty often includes server-side and network latency to talk with the backend)

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:
    - request processing latency - in apps processing requests like web apps, it is the time between issuing a request and getting an answer. In our GC-related case, without network latency
    - user interface latency - in desktop/mobile apps, it is the time between interacting with the UI (clicking something) and seeing a result

      (well, in the end it also pretty often includes server-side and network latency to talk with the backend)
- GC pauses obviously affect latency - we need to wait longer for something

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:
    - request processing latency - in apps processing requests like web apps, it is the time between issuing a request and getting an answer. In our GC-related case, without network latency
    - user interface latency - in desktop/mobile apps, it is the time between interacting with the UI (clicking something) and seeing a result

      (well, in the end it also pretty often includes server-side and network latency to talk with the backend)
- GC pauses obviously affect latency - we need to wait longer for something
- latency is pretty often measured in **percentiles**

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:
    - request processing latency - in apps processing requests like web apps, it is the time between issuing a request and getting an answer. In our GC-related case, without network latency
    - user interface latency - in desktop/mobile apps, it is the time between interacting with the UI (clicking something) and seeing a result

      (well, in the end it also pretty often includes server-side and network latency to talk with the backend)
- GC pauses obviously affect latency - we need to wait longer for something
- latency is pretty often measured in **percentiles**
- percentile - *"the value below which a percentage of data falls"* - MathIsFun

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:
  - request processing latency - in apps processing requests like web apps, it is the time between issuing a request and getting an answer. In our GC-related case, without network latency
  - user interface latency - in desktop/mobile apps, it is the time between interacting with the UI (clicking something) and seeing a result

    (well, in the end it also pretty often includes server-side and network latency to talk with the backend)
- GC pauses obviously affect latency - we need to wait longer for something
- latency is pretty often measured in **percentiles**
- percentile - *"the value below which a percentage of data falls"* - MathIsFun
- they perfectly correspond to the business measures/goals:

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:
  - request processing latency - in apps processing requests like web apps, it is the time between issuing a request and getting an answer. In our GC-related case, without network latency
  - user interface latency - in desktop/mobile apps, it is the time between interacting with the UI (clicking something) and seeing a result

    (well, in the end it also pretty often includes server-side and network latency to talk with the backend)
- GC pauses obviously affect latency - we need to wait longer for something
- latency is pretty often measured in **percentiles**
- percentile - *"the value below which a percentage of data falls"* - MathIsFun
- they perfectly correspond to the business measures/goals:
  - *"what percent of responses is faster than given time?"* - so, you measure latency in percentiles

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:
    - request processing latency - in apps processing requests like web apps, it is the time between issuing a request and getting an answer. In our GC-related case, without network latency
    - user interface latency - in desktop/mobile apps, it is the time between interacting with the UI (clicking something) and seeing a result

      (well, in the end it also pretty often includes server-side and network latency to talk with the backend)
- GC pauses obviously affect latency - we need to wait longer for something
- latency is pretty often measured in **percentiles**
- percentile - *"the value below which a percentage of data falls"* - MathIsFun
- they perfectly correspond to the business measures/goals:
    - *"what percent of responses is faster than given time?"* - so, you measure latency in percentiles
    - *"we want 80% of our users is handled faster than 1 second"* - so, you optimize for 80-th percentile

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:
  - request processing latency - in apps processing requests like web apps, it is the time between issuing a request and getting an answer. In our GC-related case, without network latency
  - user interface latency - in desktop/mobile apps, it is the time between interacting with the UI (clicking something) and seeing a result

    (well, in the end it also pretty often includes server-side and network latency to talk with the backend)
- GC pauses obviously affect latency - we need to wait longer for something
- latency is pretty often measured in **percentiles**
- percentile - *"the value below which a percentage of data falls"* - MathIsFun
- they perfectly correspond to the business measures/goals:
  - *"what percent of responses is faster than given time?"* - so, you measure latency in percentiles
  - *"we want 80% of our users is handled faster than 1 second"* - so, you optimize for 80-th percentile
- we may be interested in high 95-th (P95) or 99-th percentile (P99):

Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:
  - request processing latency - in apps processing requests like web apps, it is the time between issuing a request and getting an answer. In our GC-related case, without network latency
  - user interface latency - in desktop/mobile apps, it is the time between interacting with the UI (clicking something) and seeing a result

    (well, in the end it also pretty often includes server-side and network latency to talk with the backend)
- GC pauses obviously affect latency - we need to wait longer for something
- latency is pretty often measured in **percentiles**
- percentile - *"the value below which a percentage of data falls"* - MathIsFun
- they perfectly correspond to the business measures/goals:
  - *"what percent of responses is faster than given time?"* - so, you measure latency in percentiles
  - *"we want 80% of our users is handled faster than 1 second"* - so, you optimize for 80-th percentile
- we may be interested in high 95-th (P95) or 99-th percentile (P99):
  - it is fine that 95% or 99% of customers are satisfied by fast response times

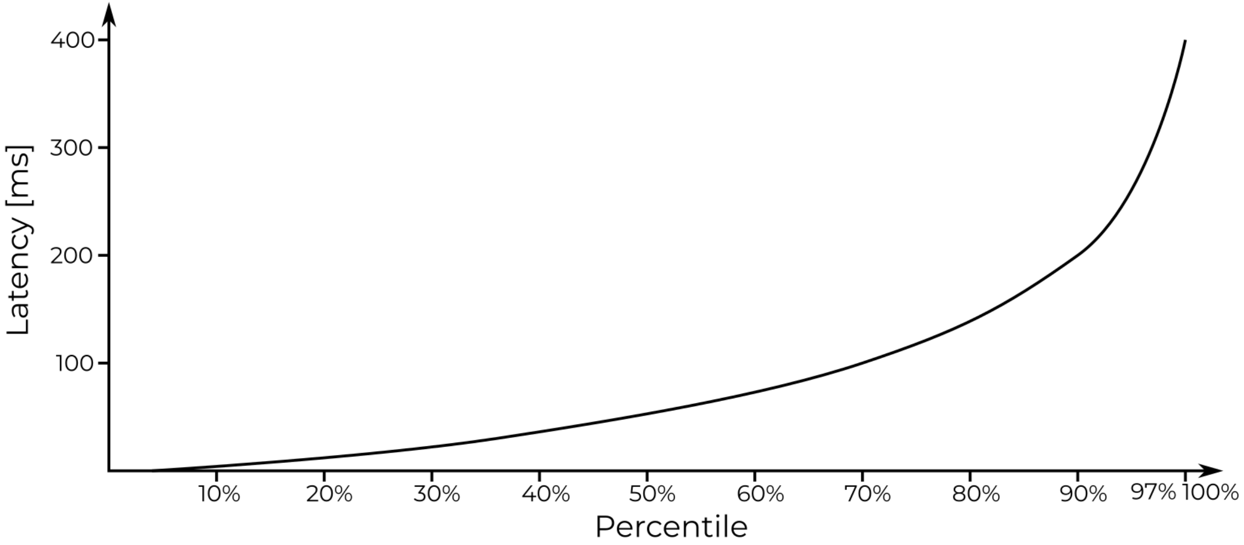
Pauses vs Latency

- *"Latency - a time delay between the cause and the effect"* - Wikipedia
- for example:
    - request processing latency - in apps processing requests like web apps, it is the time between issuing a request and getting an answer. In our GC-related case, without network latency
    - user interface latency - in desktop/mobile apps, it is the time between interacting with the UI (clicking something) and seeing a result

      (well, in the end it also pretty often includes server-side and network latency to talk with the backend)
- GC pauses obviously affect latency - we need to wait longer for something
- latency is pretty often measured in **percentiles**
- percentile - *"the value below which a percentage of data falls"* - MathIsFun
- they perfectly correspond to the business measures/goals:
    - *"what percent of responses is faster than given time?"* - so, you measure latency in percentiles
    - *"we want 80% of our users is handled faster than 1 second"* - so, you optimize for 80-th percentile
- we may be interested in high 95-th (P95) or 99-th percentile (P99):
    - it is fine that 95% or 99% of customers are satisfied by fast response times
    - but... still there is this 5% or 1% which may be hit be the long latency (and leave with their 💸)

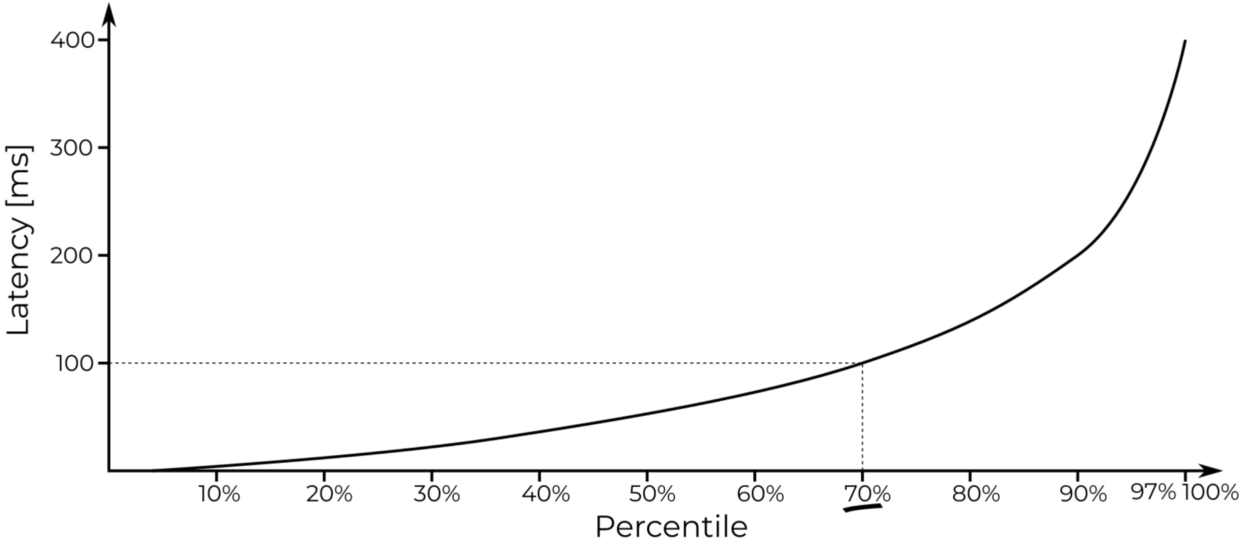
Pauses vs Latency

We can get a broad picture by visualizing latency in spectrum of percentiles:



Pauses vs Latency

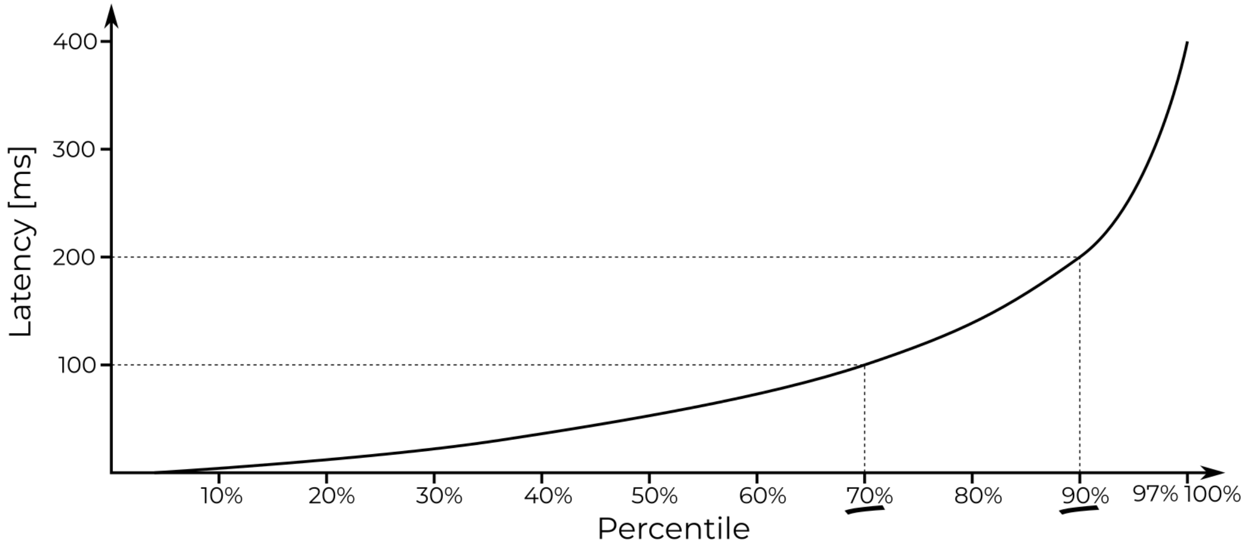
We can get a broad picture by visualizing latency in spectrum of percentiles:



- **70-th percentile** - 70% of responses are below 100 ms

Pauses vs Latency

We can get a broad picture by visualizing latency in spectrum of percentiles:

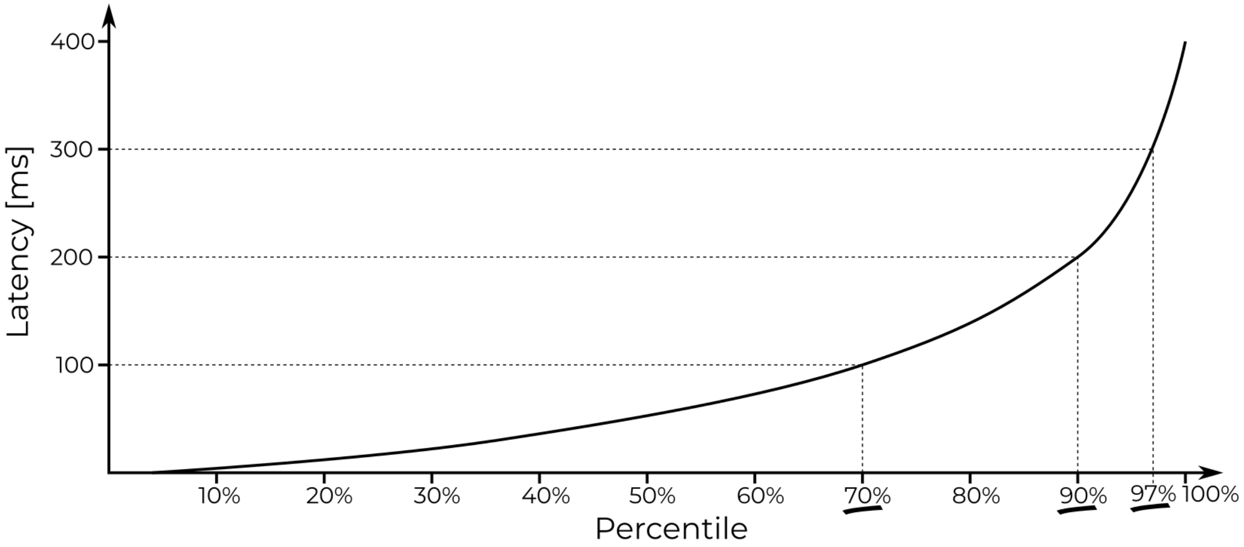


- **70-th percentile** - 70% of responses are below 100 ms
- **90-th percentile** - 90% of responses are below 200 ms

  (which leaves 10% people observing times slower than 200ms)

Pauses vs Latency

We can get a broad picture by visualizing latency in spectrum of percentiles:



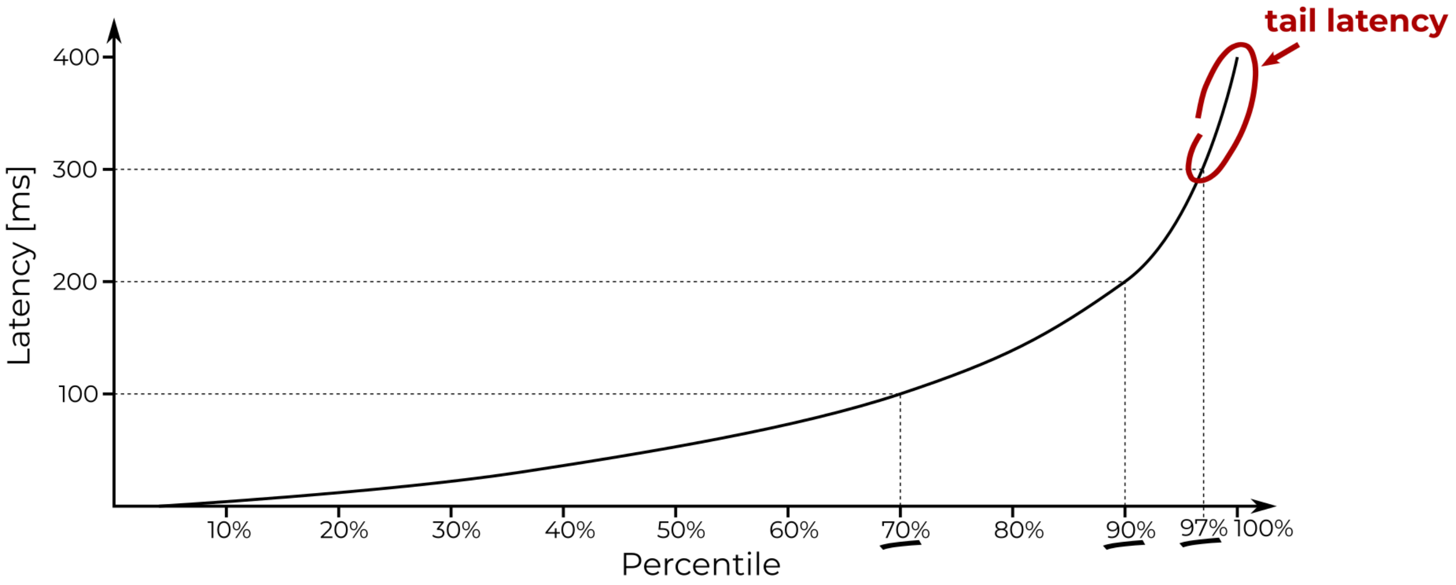
- **70-th percentile** - 70% of responses are below 100 ms
- **90-th percentile** - 90% of responses are below 200 ms

  (which leaves 10% people observing times slower than 200ms)
- **97-th percentile** - 97% of responses are below 300 ms

  (which leaves 3% people observing times slower than 300ms)

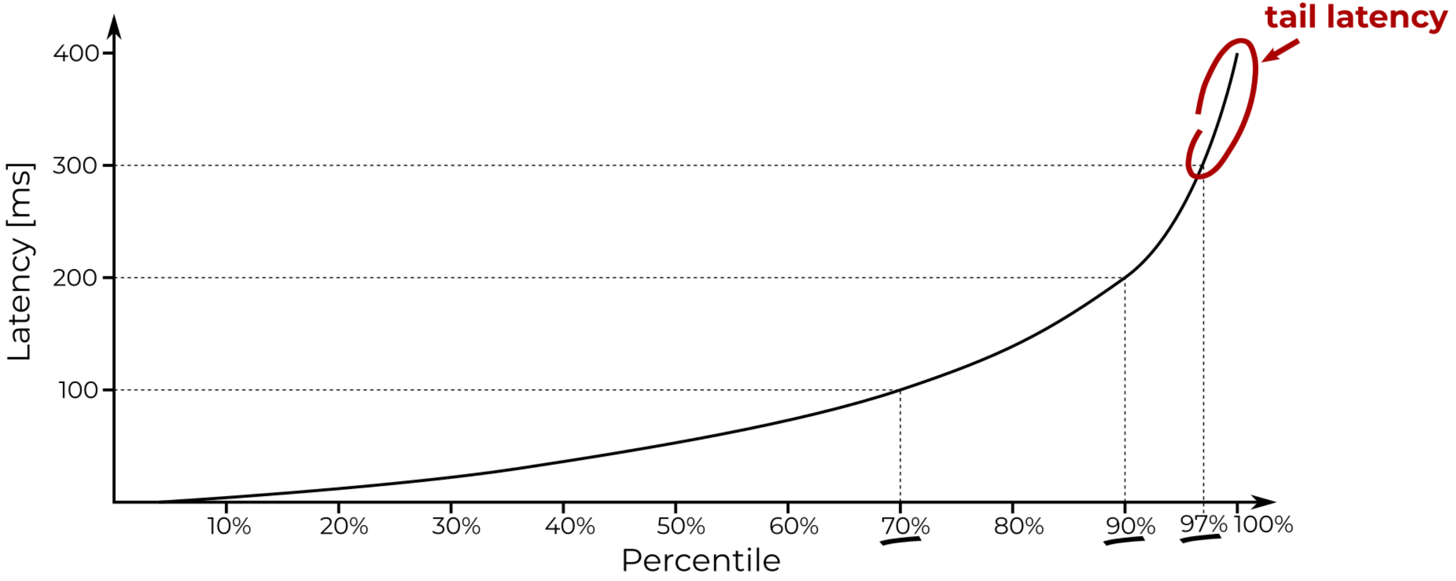
Pauses vs Latency

Tail latency - some high-percentile latency, observed by users rarely, but probably really severe:



Pauses vs Latency

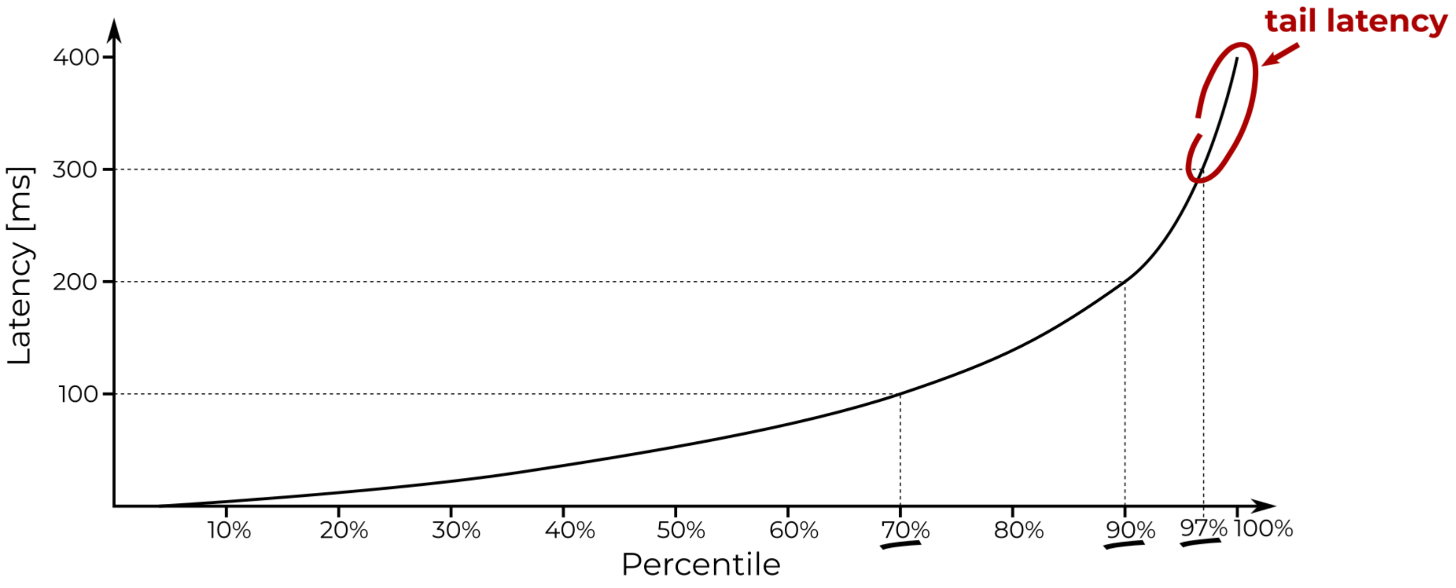
Tail latency - some high-percentile latency, observed by users rarely, but probably really severe:



- optimizing tail latency is trying to reduce latency in some high percentile like P95 or P99

Pauses vs Latency

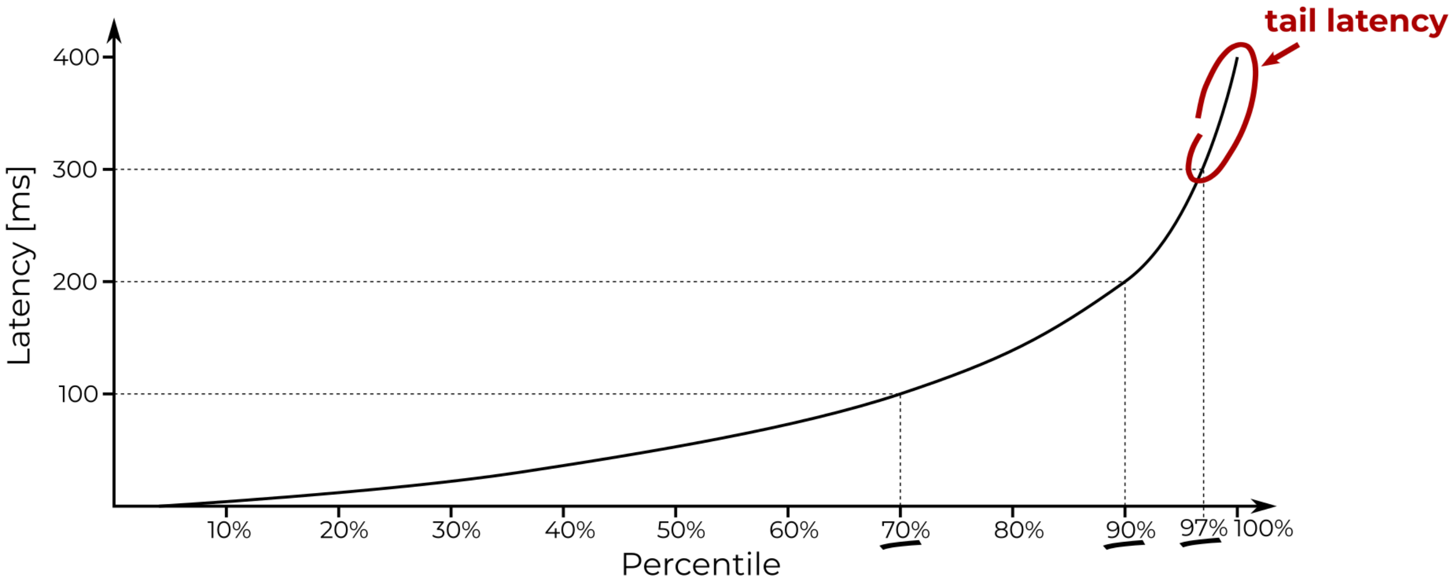
Tail latency - some high-percentile latency, observed by users rarely, but probably really severe:



- optimizing tail latency is trying to reduce latency in some high percentile like P95 or P99
- for example, if you want to reduce P97 latency, look at responses <300 ms

Pauses vs Latency

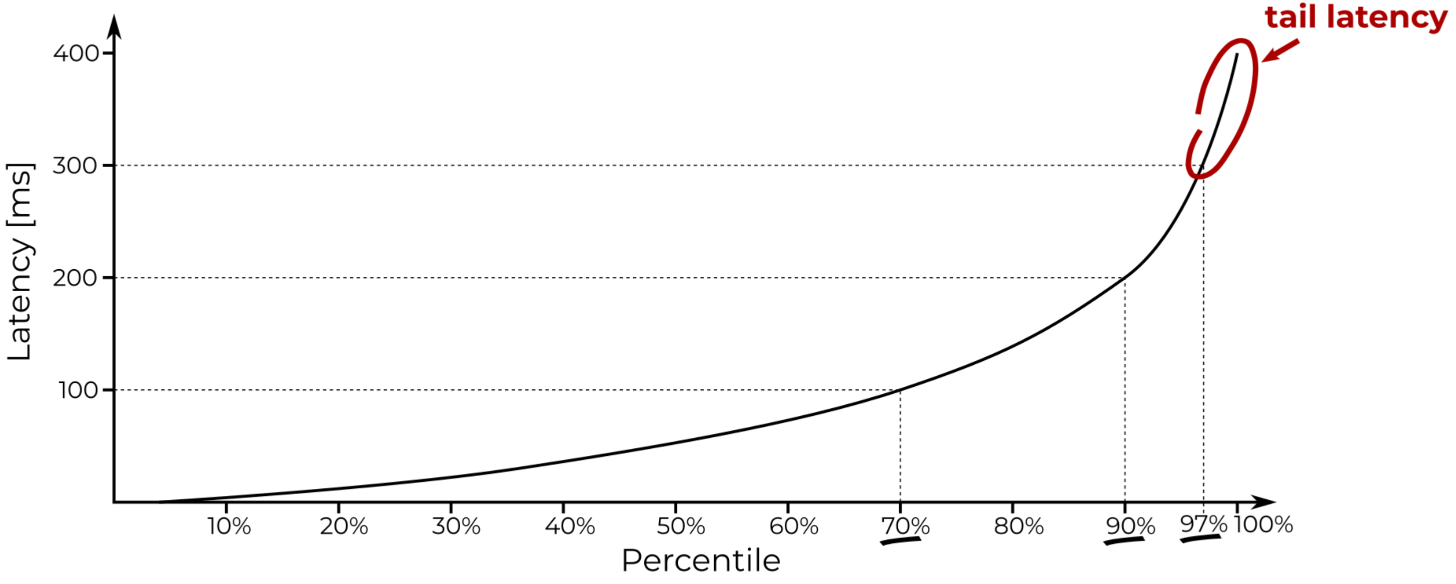
Tail latency - some high-percentile latency, observed by users rarely, but probably really severe:



- optimizing tail latency is trying to reduce latency in some high percentile like P95 or P99
- for example, if you want to reduce P97 latency, look at responses <300 ms
    - we investigate reasons of responses slower than 300ms

Pauses vs Latency

Tail latency - some high-percentile latency, observed by users rarely, but probably really severe:



- optimizing tail latency is trying to reduce latency in some high percentile like P95 or P99
- for example, if you want to reduce P97 latency, look at responses <300 ms
  - we investigate reasons of responses slower than 300ms
  - ...and GC pauses may be one of them!

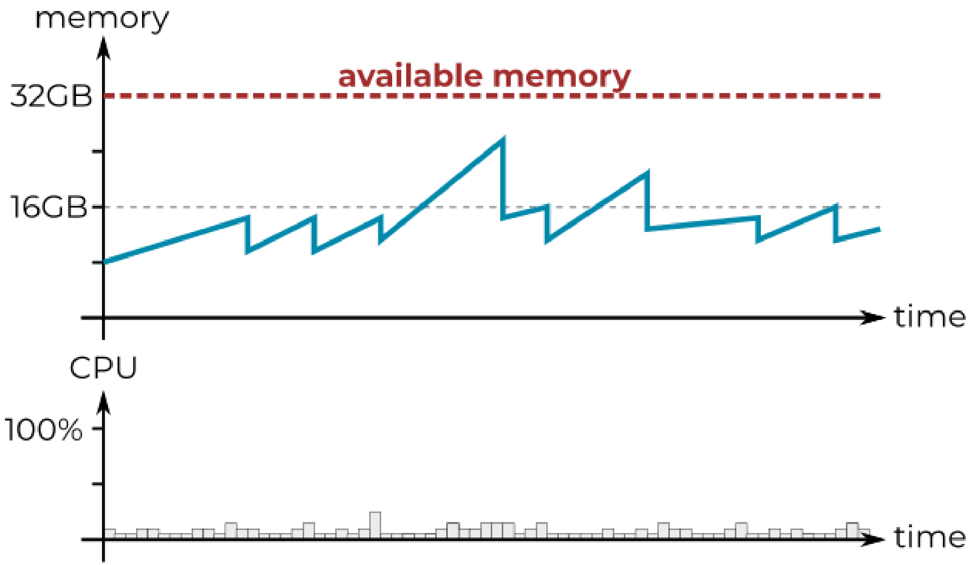
CPU overhead

We simply need to control how much CPU it takes to do the GC work. Again, typically it is a compromise with some other metrics like memory footprint, pauses, …

CPU overhead

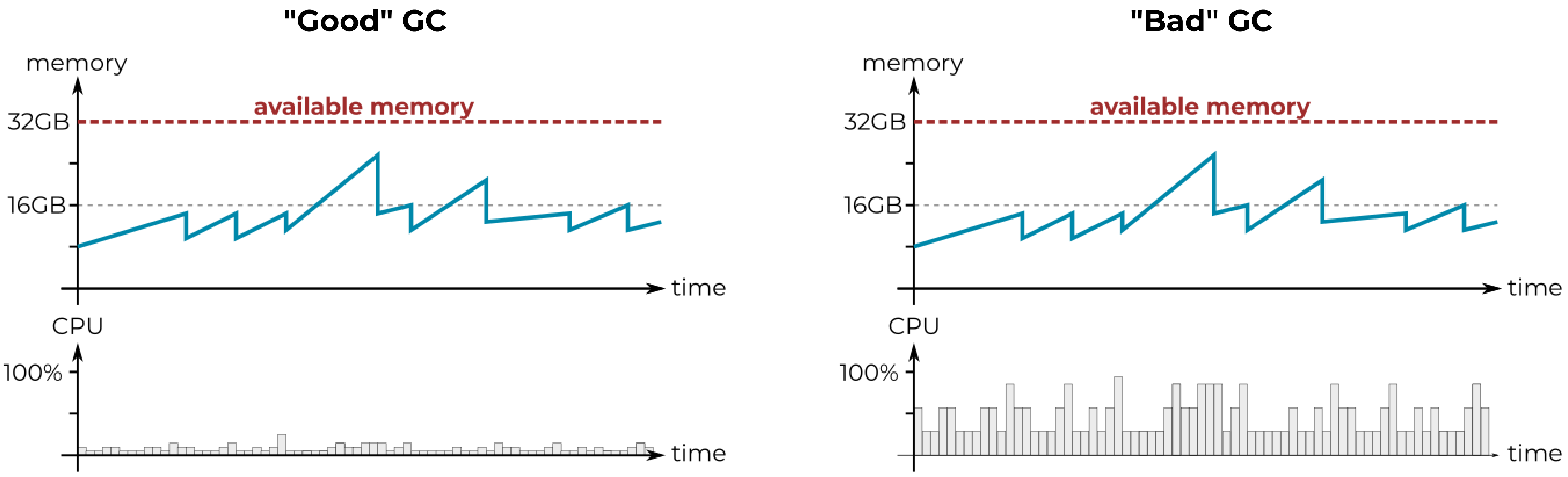
We simply need to control how much CPU it takes to do the GC work. Again, typically it is a compromise with some other metrics like memory footprint, pauses, ...

"Good" GC



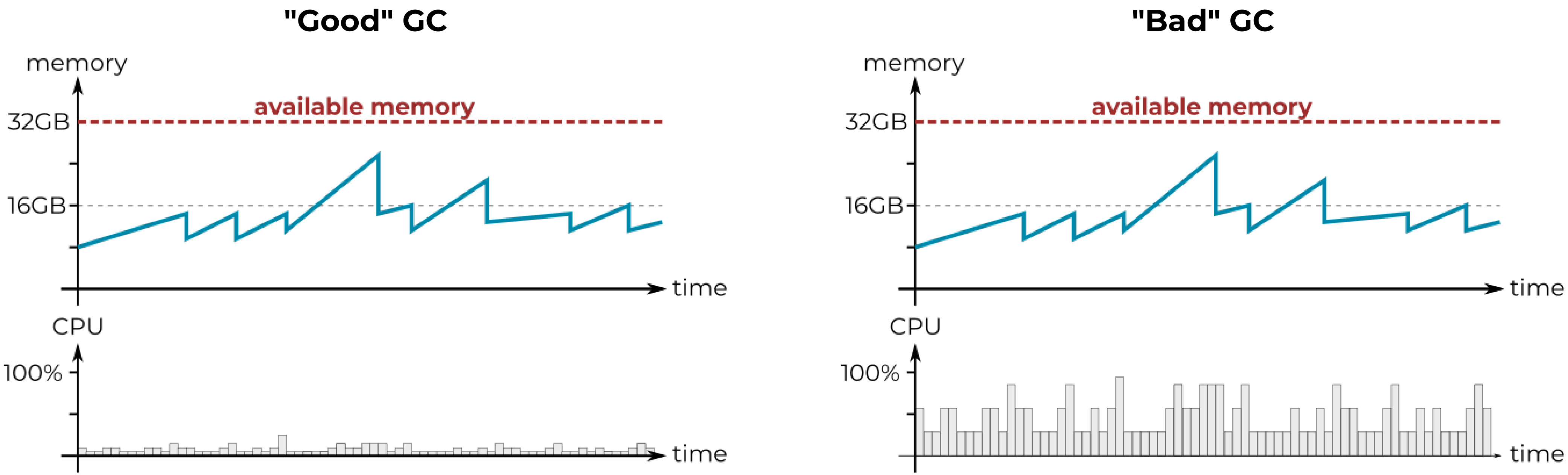
CPU overhead

We simply need to control how much CPU it takes to do the GC work. Again, typically it is a compromise with some other metrics like memory footprint, pauses, ...



CPU overhead

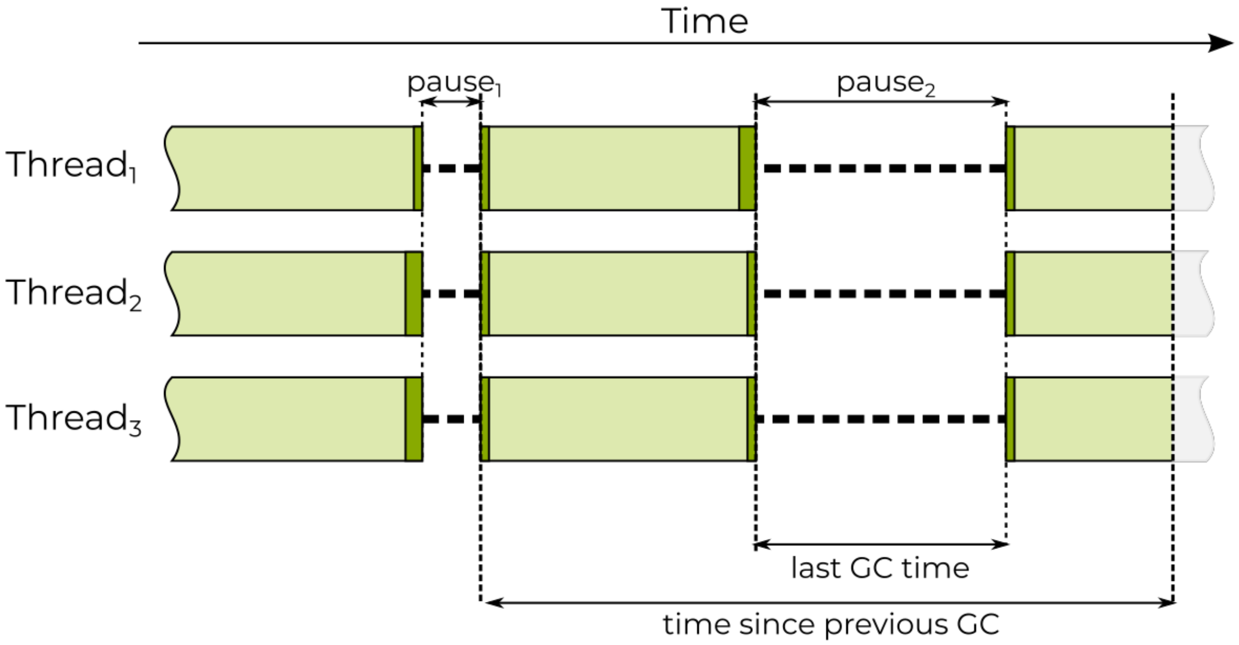
We simply need to control how much CPU it takes to do the GC work. Again, typically it is a compromise with some other metrics like memory footprint, pauses, ...



We could measure **the whole machine CPU usage** or **our process CPU usage**.

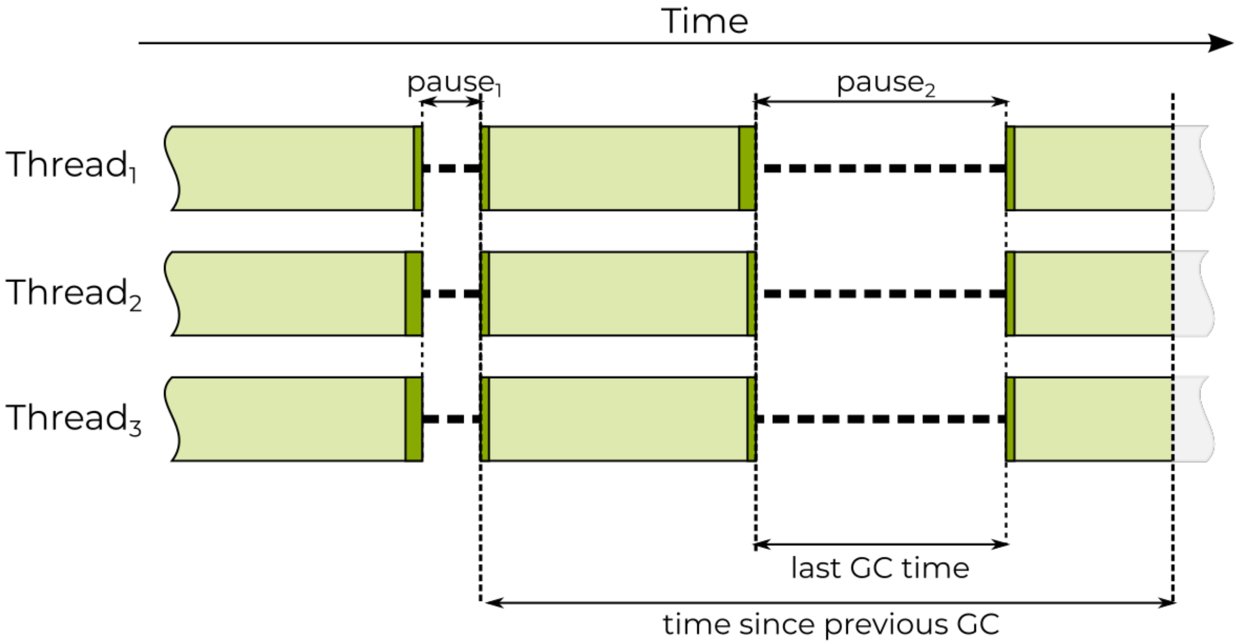
CPU overhead

The simplest measurement is the relative time spent in the CC to the application time:



CPU overhead

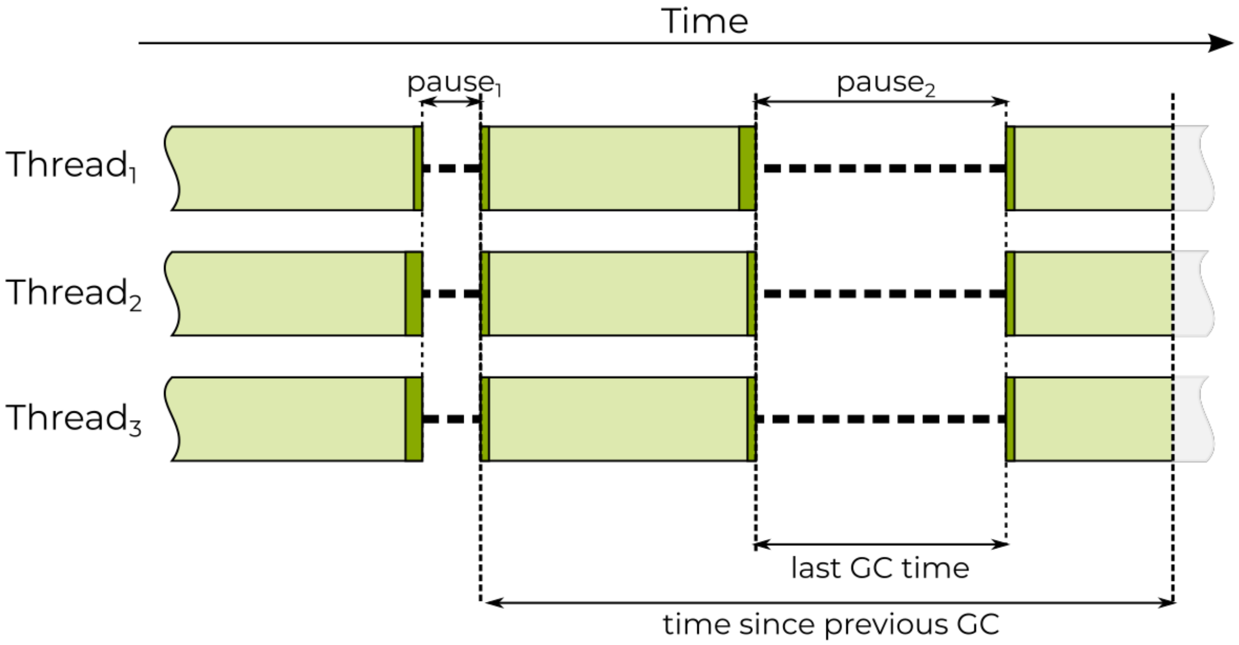
The simplest measurement is the relative time spent in the CC to the application time:



$$\% \text{ Time in GC} = \frac{\text{last GC time}}{\text{time since previous GC}}$$

CPU overhead

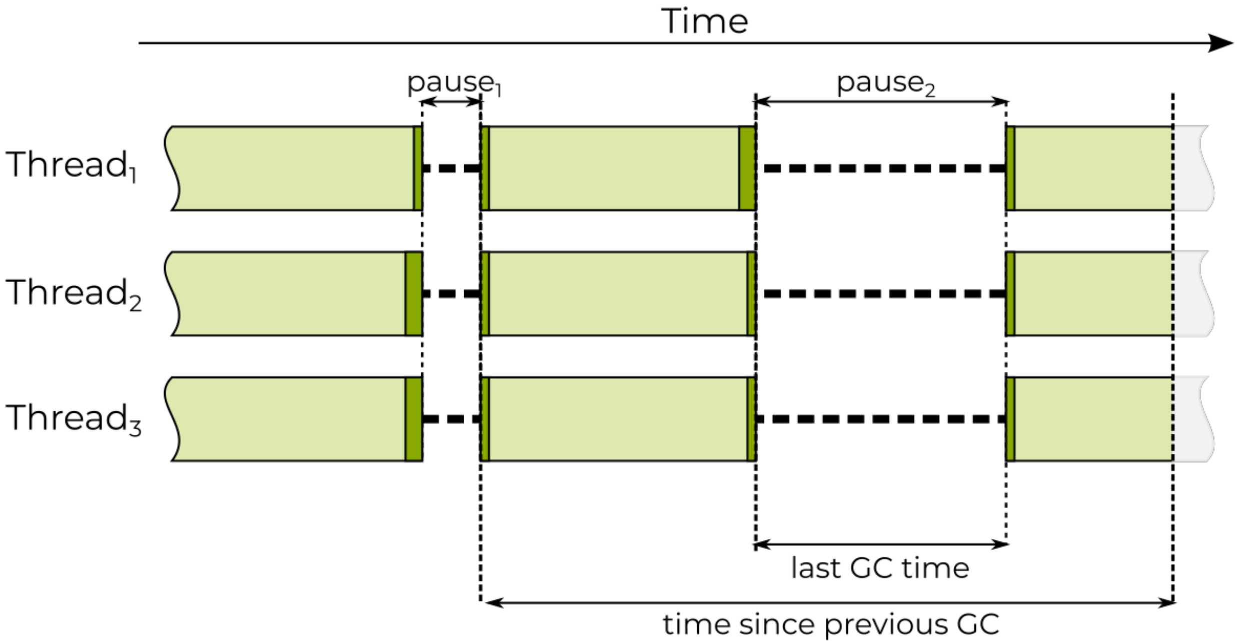
The simplest measurement is the relative time spent in the CC to the application time:



$$\% \text{ Time in GC} = \frac{\text{last GC time}}{\text{time since previous GC}} = \frac{3.0}{8.3} \approx 36\%$$

CPU overhead

The simplest measurement is the relative time spent in the CC to the application time:

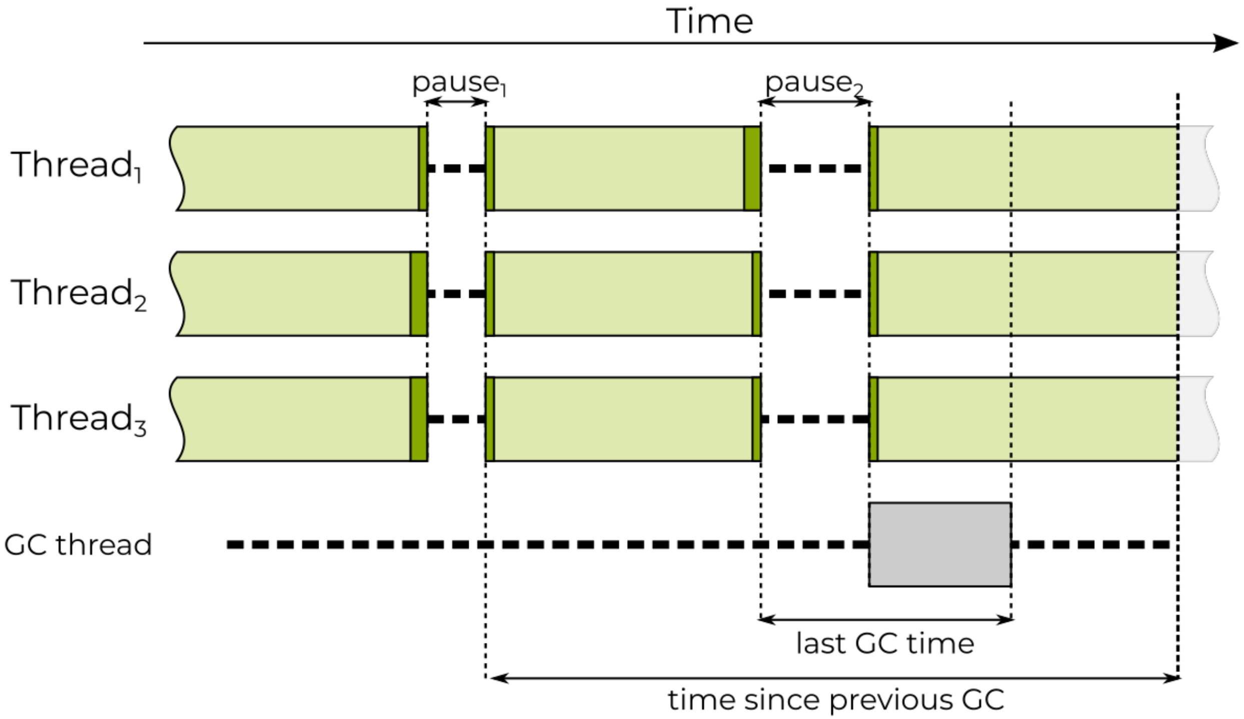


$$\% \text{ Time in GC} = \frac{\text{last GC time}}{\text{time since previous GC}} = \frac{3.0}{8.3} \approx 36\%$$

In case of GC work done only during pauses - it's the same as *% Pause time in GC* (as above).

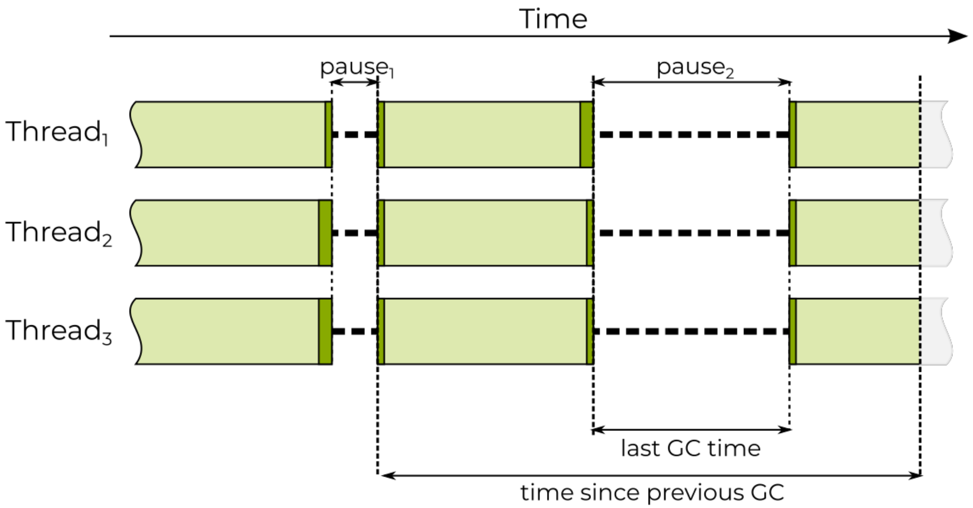
CPU overhead

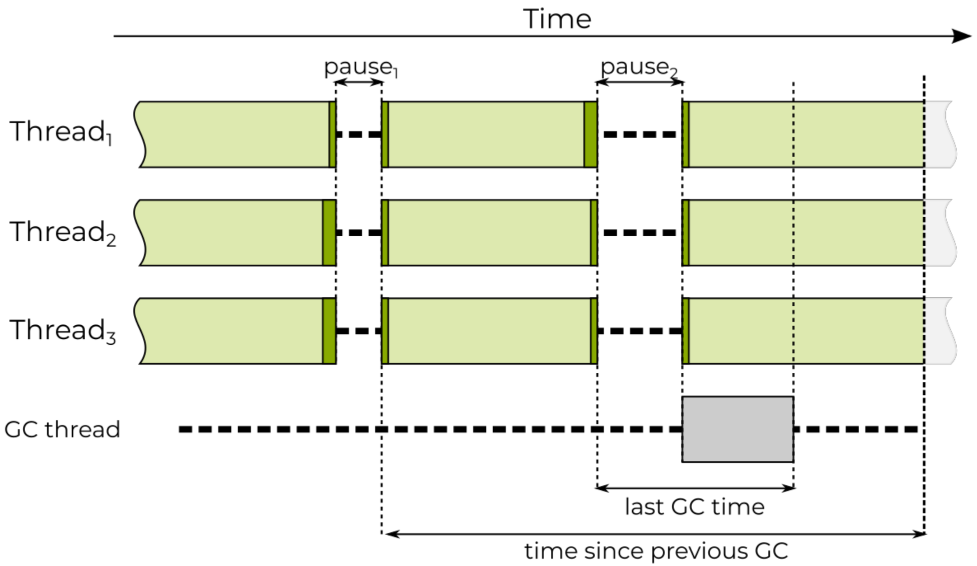
If part of the GC work is done **concurrently**, it also counts:



CPU overhead

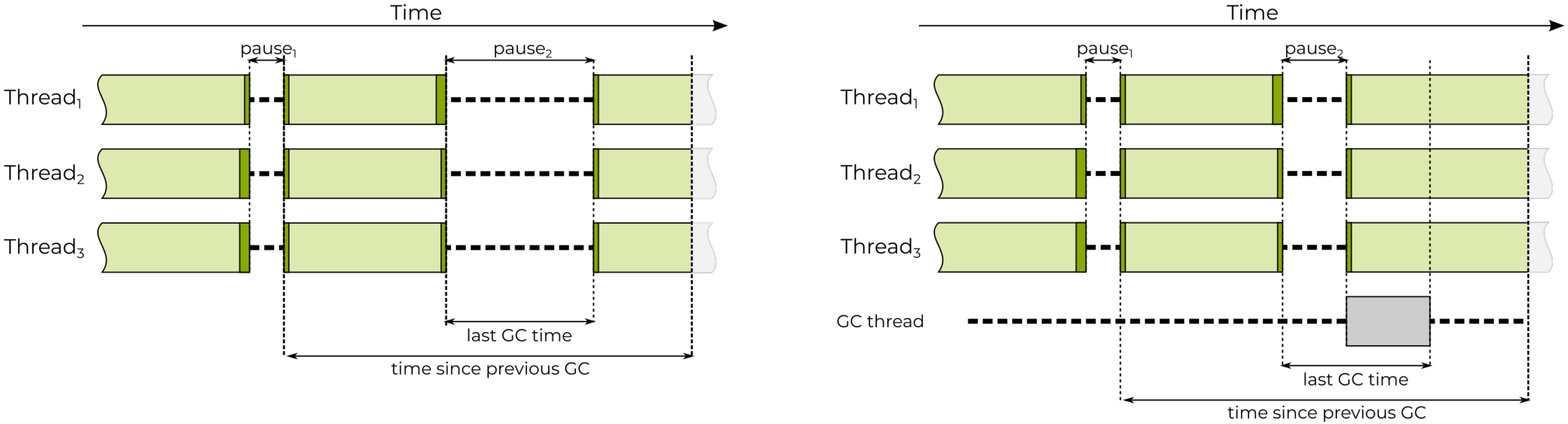
% Time in GC may be a little misleading - it produces the same values for concurrent and non-concurrent case:





CPU overhead

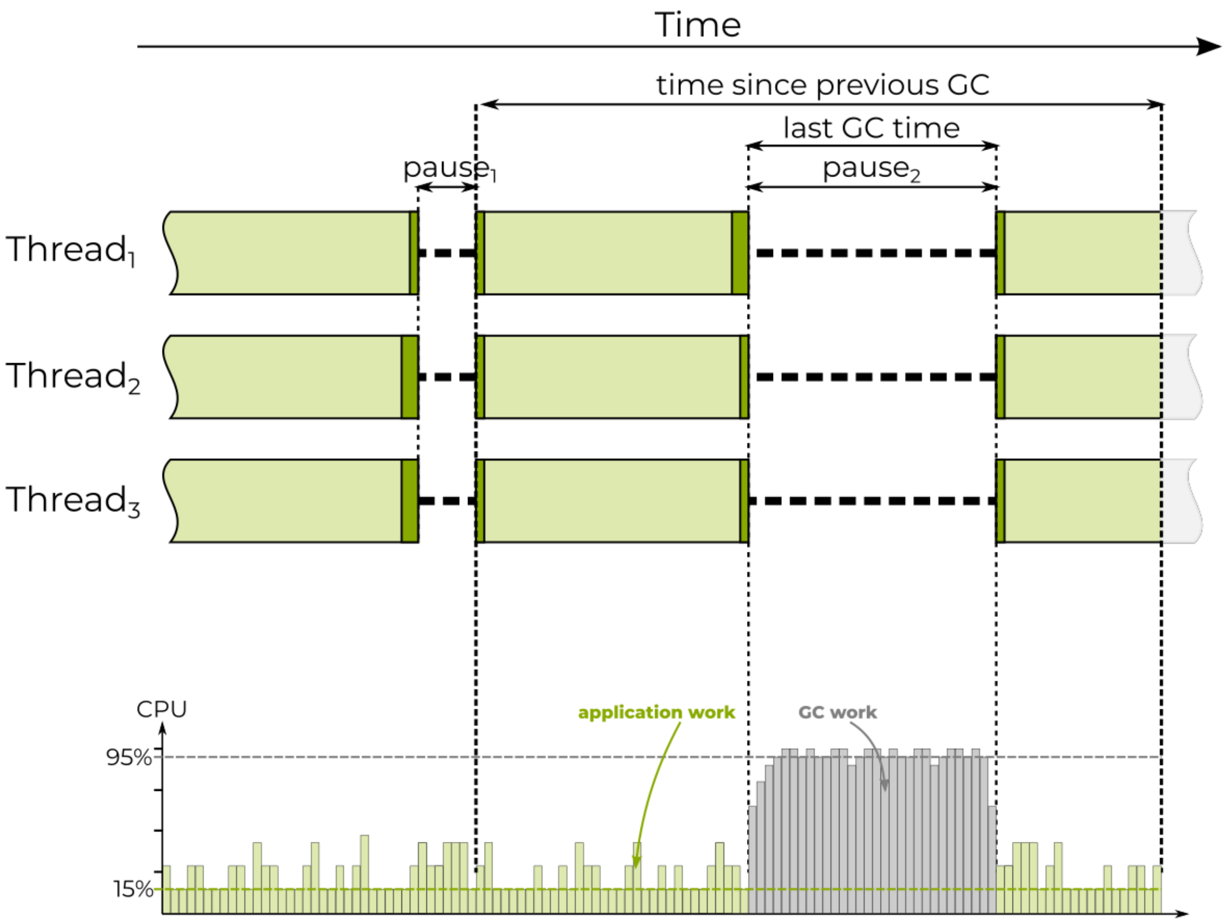
% Time in GC may be a little misleading - it produces the same values for concurrent and non-concurrent case:



But we don't know which case consumes more CPU - it depends how CPU-intensive is the GC work during pause and concurrent phase.

CPU overhead

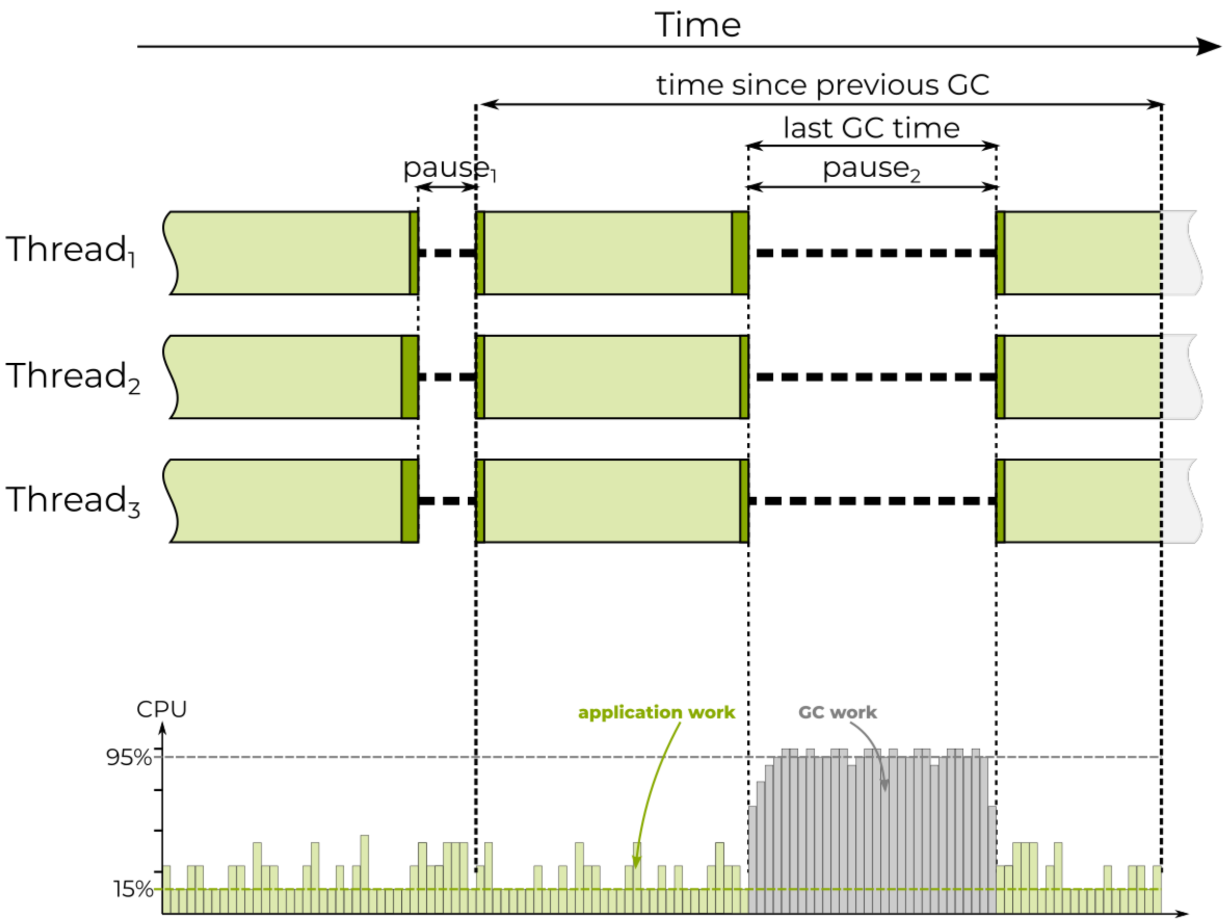
% CPU time in GC is much more precise - instead of time, measure real **CPU samples** done by the CPU:



- very CPU-intensive work during the GC pause - that's good, **we want to end it as fast as possible**!

CPU overhead

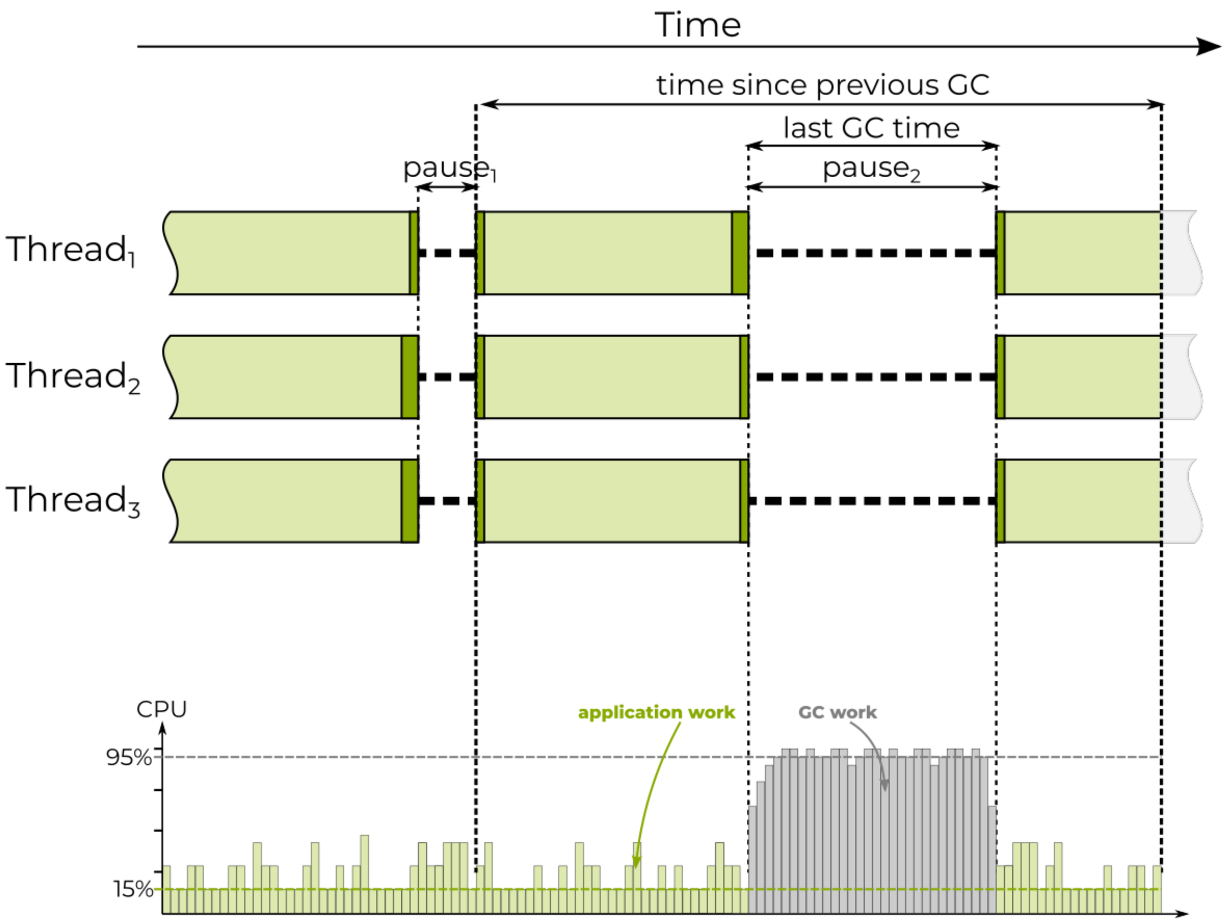
% CPU time in GC is much more precise - instead of time, measure real **CPU samples** done by the CPU:



- very CPU-intensive work during the GC pause - that's good, **we want to end it as fast as possible**!
- expensive to monitor - requires CPU samples...

CPU overhead

% CPU time in GC is much more precise - instead of time, measure real **CPU samples** done by the CPU:



- very CPU-intensive work during the GC pause - that's good, **we want to end it as fast as possible**!
- expensive to monitor - requires CPU samples...
- easy to calculate:

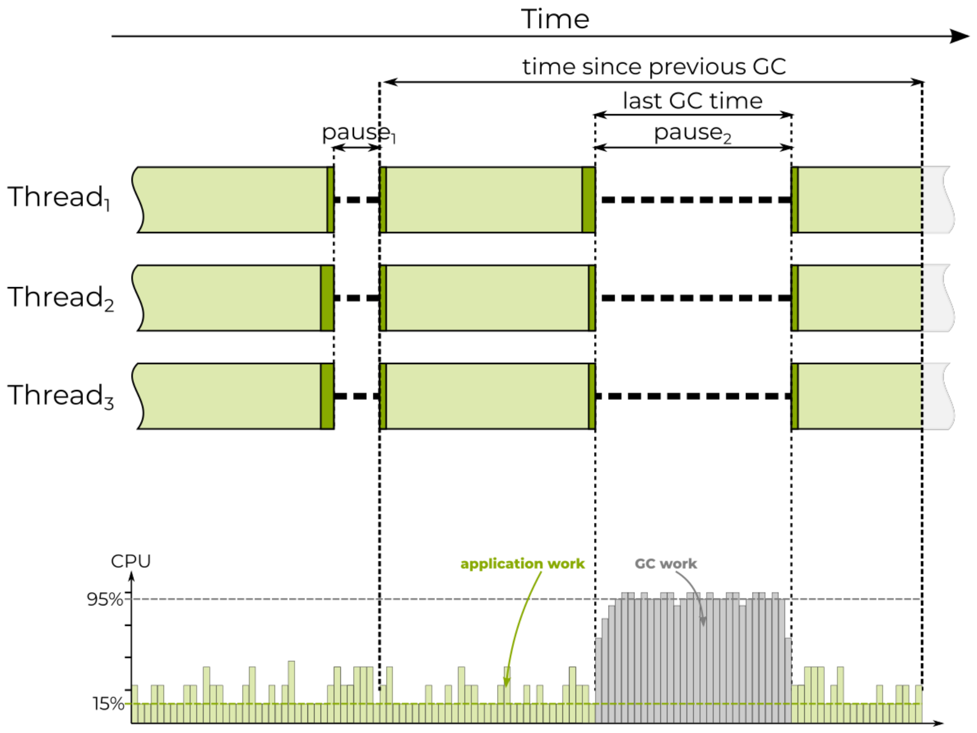
$$\%\text{ CPU time in GC} = \frac{\text{CPU samples in GC}}{\text{total CPU samples}} \approx 80\%$$

CPU overhead

% CPU time in GC takes into account concurrent GC much better:

CPU overhead

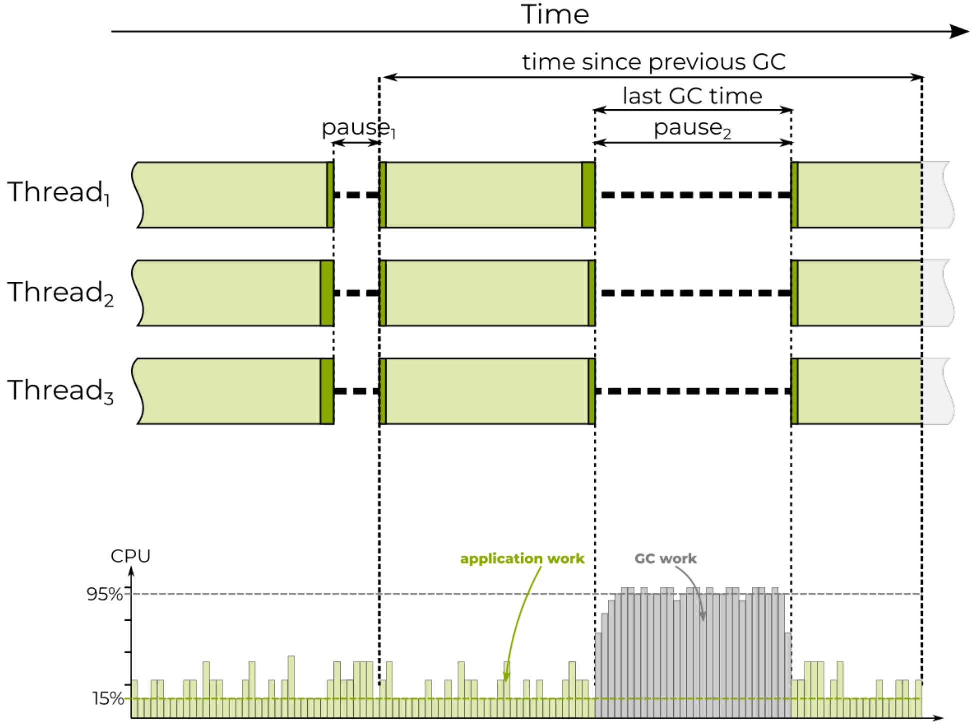
% CPU time in GC takes into account concurrent GC much better:



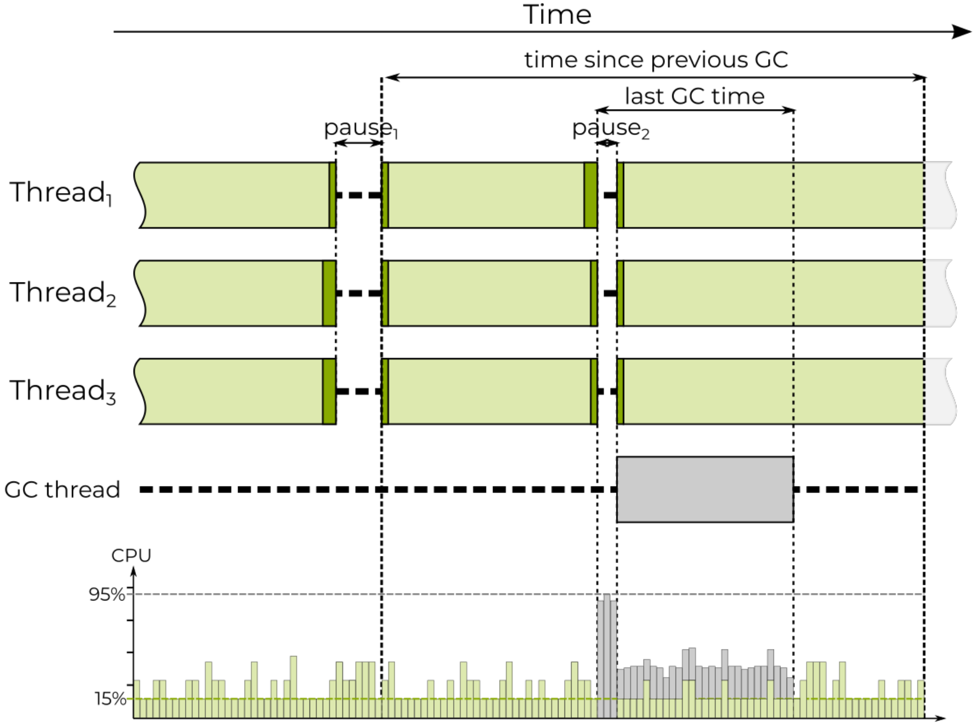
≈ 80%

CPU overhead

% CPU time in GC takes into account concurrent GC much better:



≈ **80%**



≈ **50%**

Throughput

- measurement of how many requests (units of work) we are able to process in a unit of time

Throughput

- measurement of how many requests (units of work) we are able to process in a unit of time
- for example, we are able to process 100 requests per second

Throughput

- measurement of how many requests (units of work) we are able to process in a unit of time
- for example, we are able to process 100 requests per second
- the more, the better!

Throughput

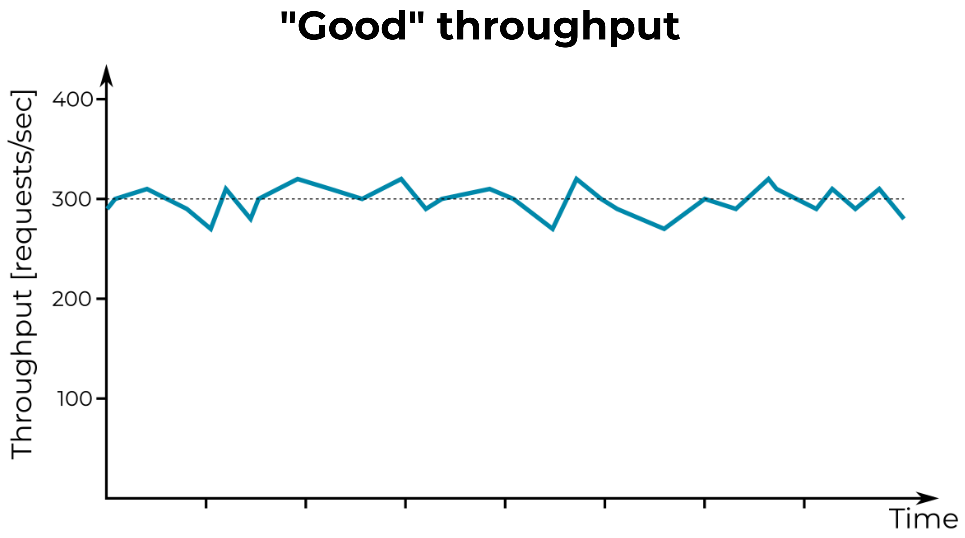
- measurement of how many requests (units of work) we are able to process in a unit of time
- for example, we are able to process 100 requests per second
- the more, the better!
- but is has its cost like resource (CPU) consumption etc.

Throughput

Typically it is not super stable, so it is good to measure it over time:

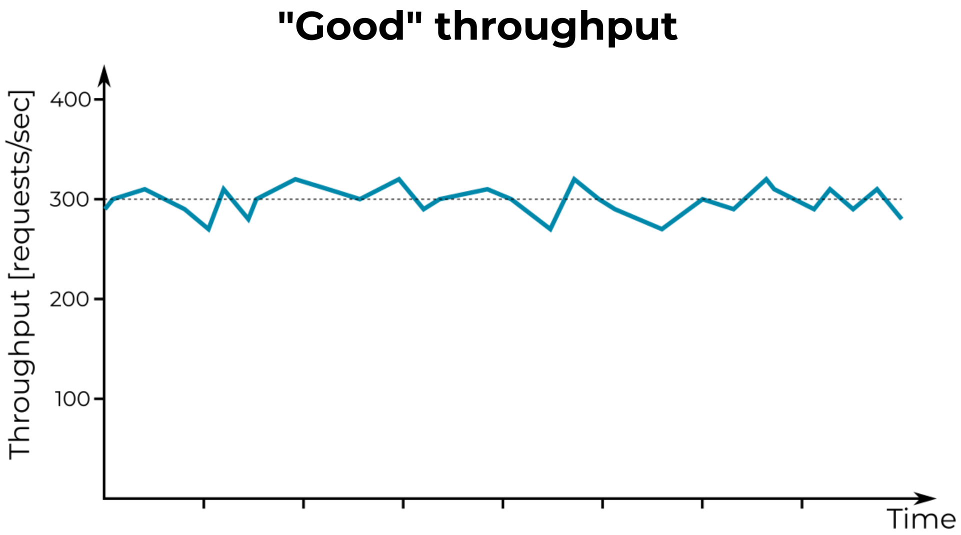
Throughput

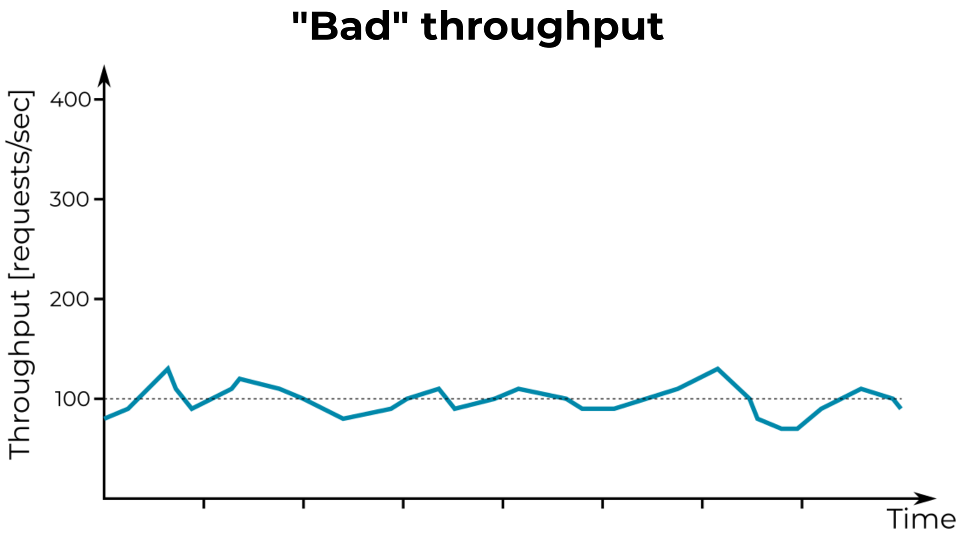
Typically it is not super stable, so it is good to measure it over time:



Throughput

Typically it is not super stable, so it is good to measure it over time:





Summary

- now we know all the features of an excellent GC:

Summary

- now we know all the features of an excellent GC:
  - aggressive, so it results in **low memory overhead**

Summary

- now we know all the features of an excellent GC:
  - aggressive, so it results in **low memory overhead**
  - **low CPU usage**

Summary

- now we know all the features of an excellent GC:
  - aggressive, so it results in **low memory overhead**
  - **low CPU usage**
  - super **short pauses** not impacting latency

Summary

- now we know all the features of an excellent GC:
    - aggressive, so it results in **low memory overhead**
    - **low CPU usage**
    - super **short pauses** not impacting latency
    - **high throughput** stable over time

Summary

- now we know all the features of an excellent GC:
  - aggressive, so it results in **low memory overhead**
  - **low CPU usage**
  - super **short pauses** not impacting latency
  - **high throughput** stable over time
- the problem is... there is no such a perfect GC

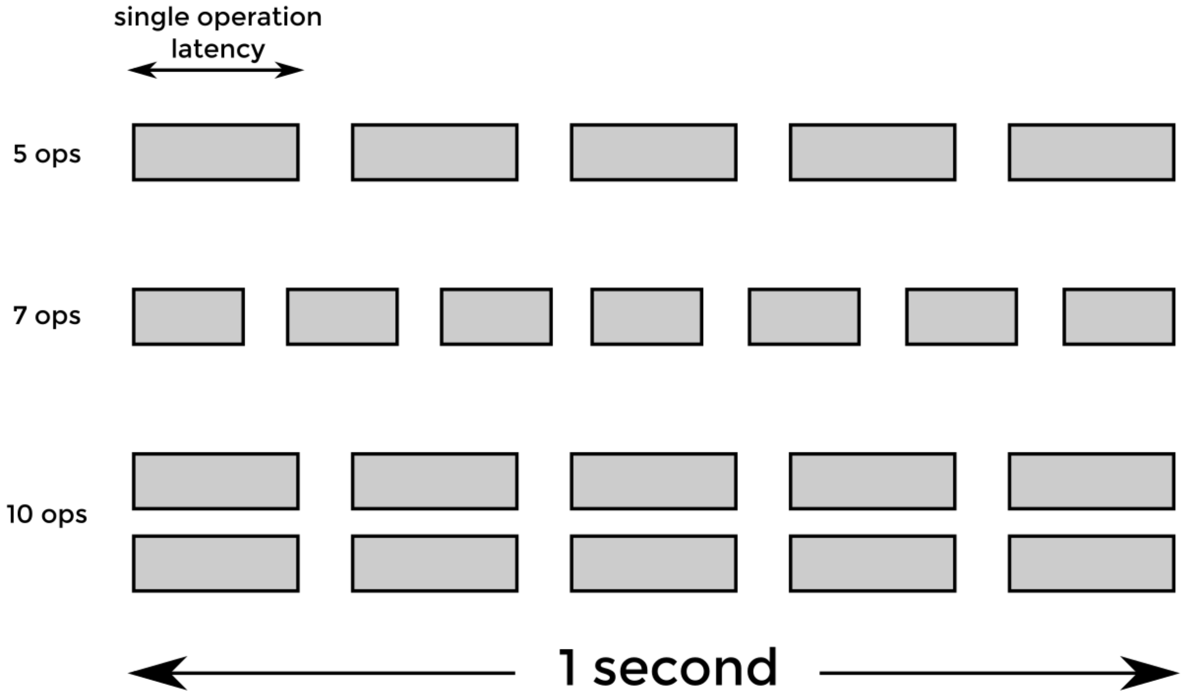
Summary

- now we know all the features of an excellent GC:
  - aggressive, so it results in **low memory overhead**
  - **low CPU usage**
  - super **short pauses** not impacting latency
  - **high throughput** stable over time
- the problem is... there is no such a perfect GC
  - obviously we can optimize algorithms to make it more efficient

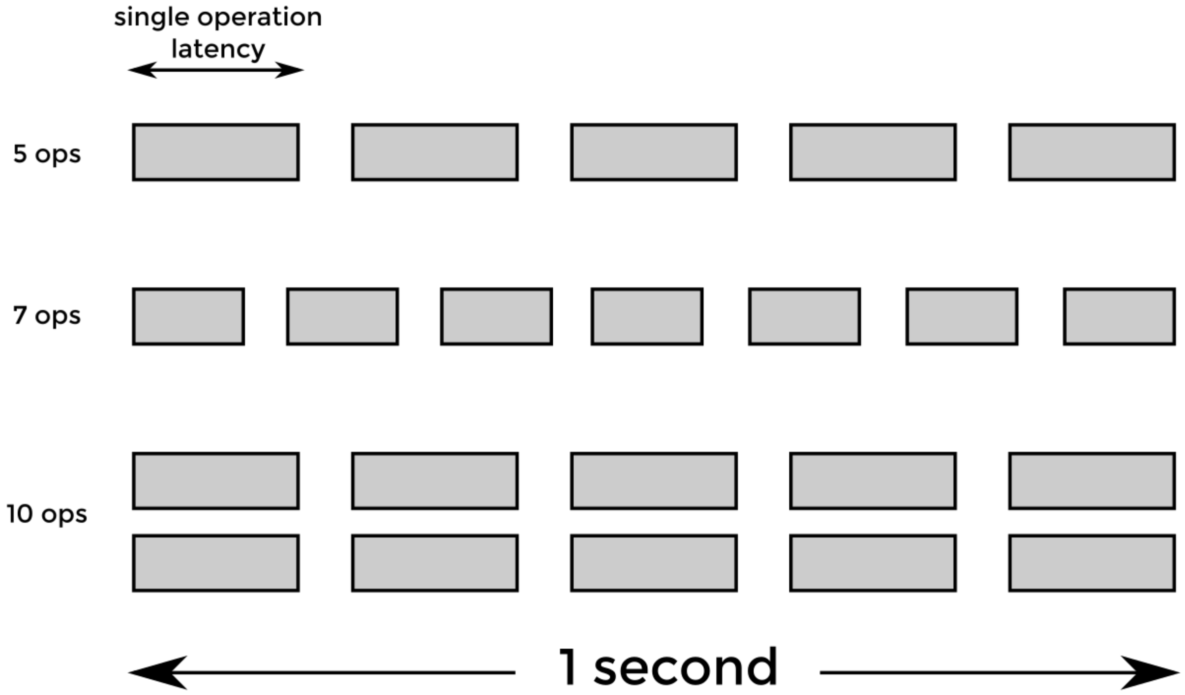
Summary

- now we know all the features of an excellent GC:
  - aggressive, so it results in **low memory overhead**
  - **low CPU usage**
  - super **short pauses** not impacting latency
  - **high throughput** stable over time
- the problem is... there is no such a perfect GC
  - obviously we can optimize algorithms to make it more efficient
  - but in the end, pretty often we optimize one aspect as a trade off with the others

Summary

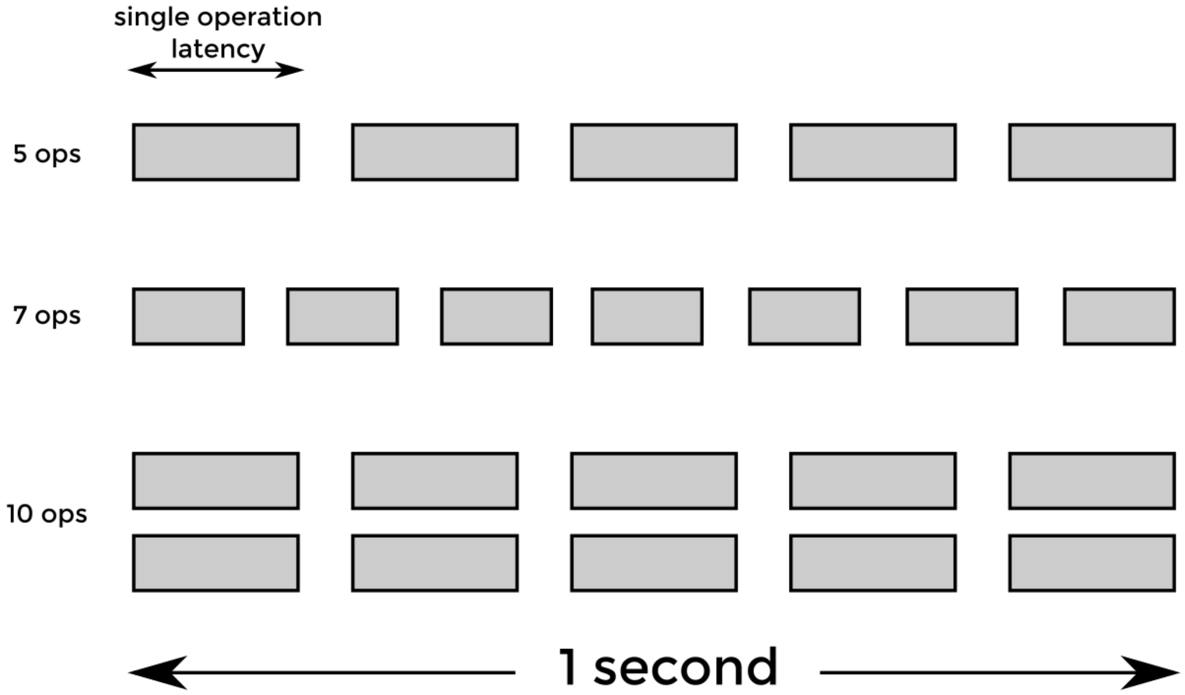


Summary



- we can increase **throughput** by reducing **latency** (e.g. **GC pauses**)

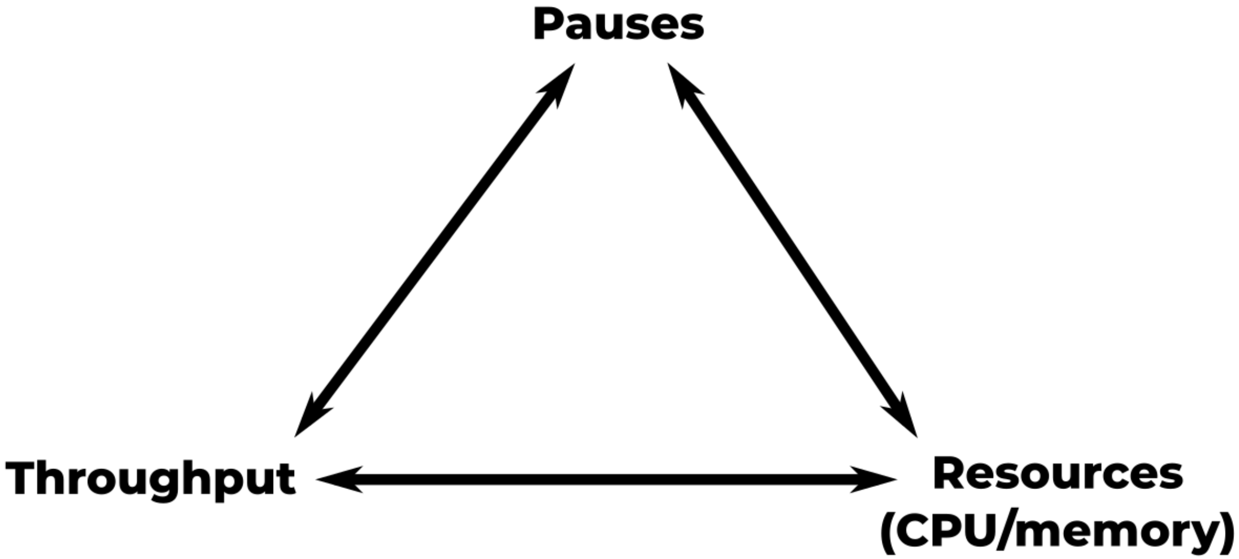
Summary



- we can increase **throughput** by reducing **latency** (e.g. **GC pauses**)
- we can increase **throughput** by parallelizing work (e.g. consuming more CPU cores)

Garbage Collection Trilemma

The "impossible triangle" of the GC features:



Materials

- .NET Memory Performance Analysis