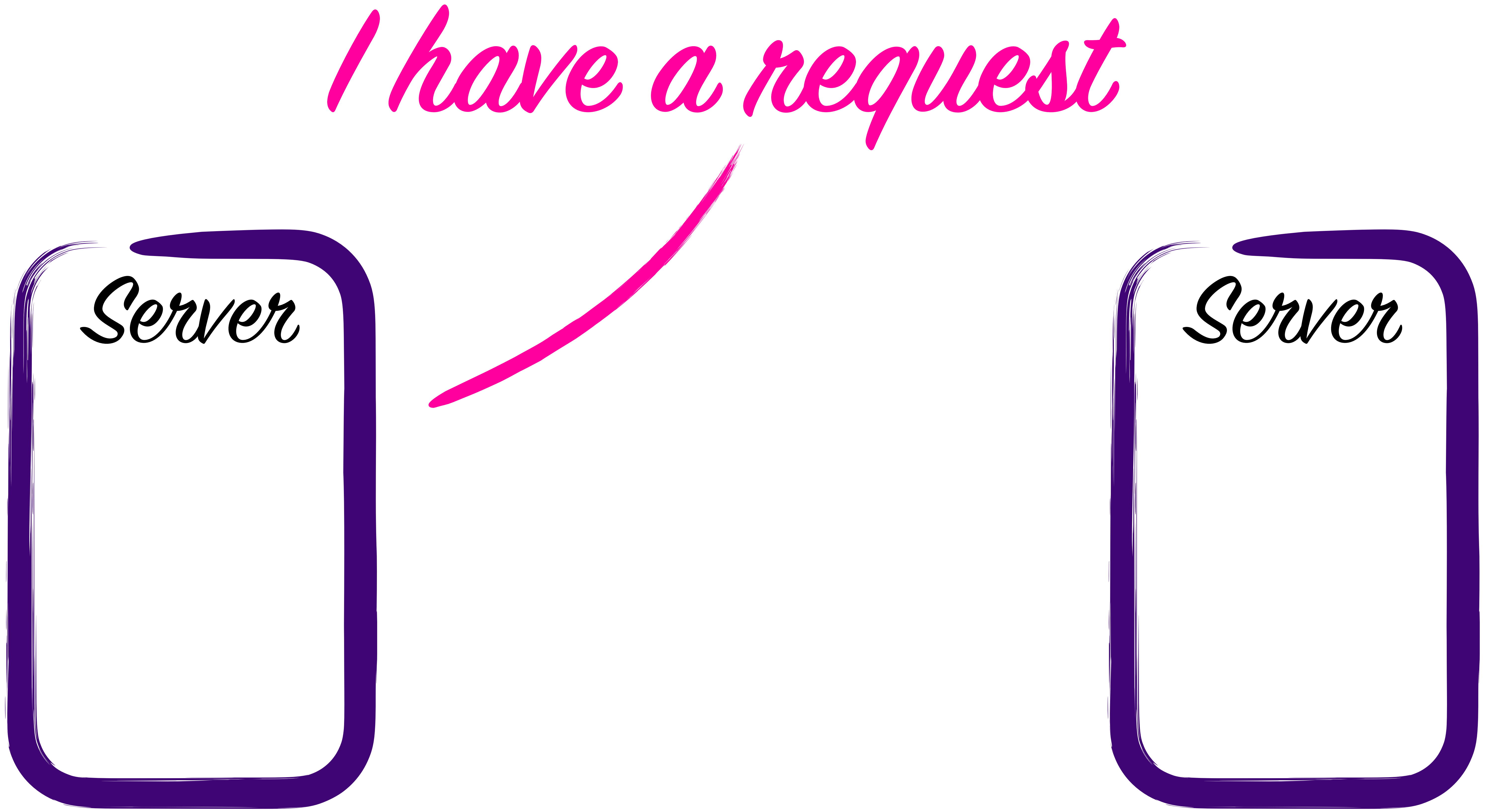
Building Adaptive Systems

Chris Keathley / @ChrisKeathley / c@keathley.io

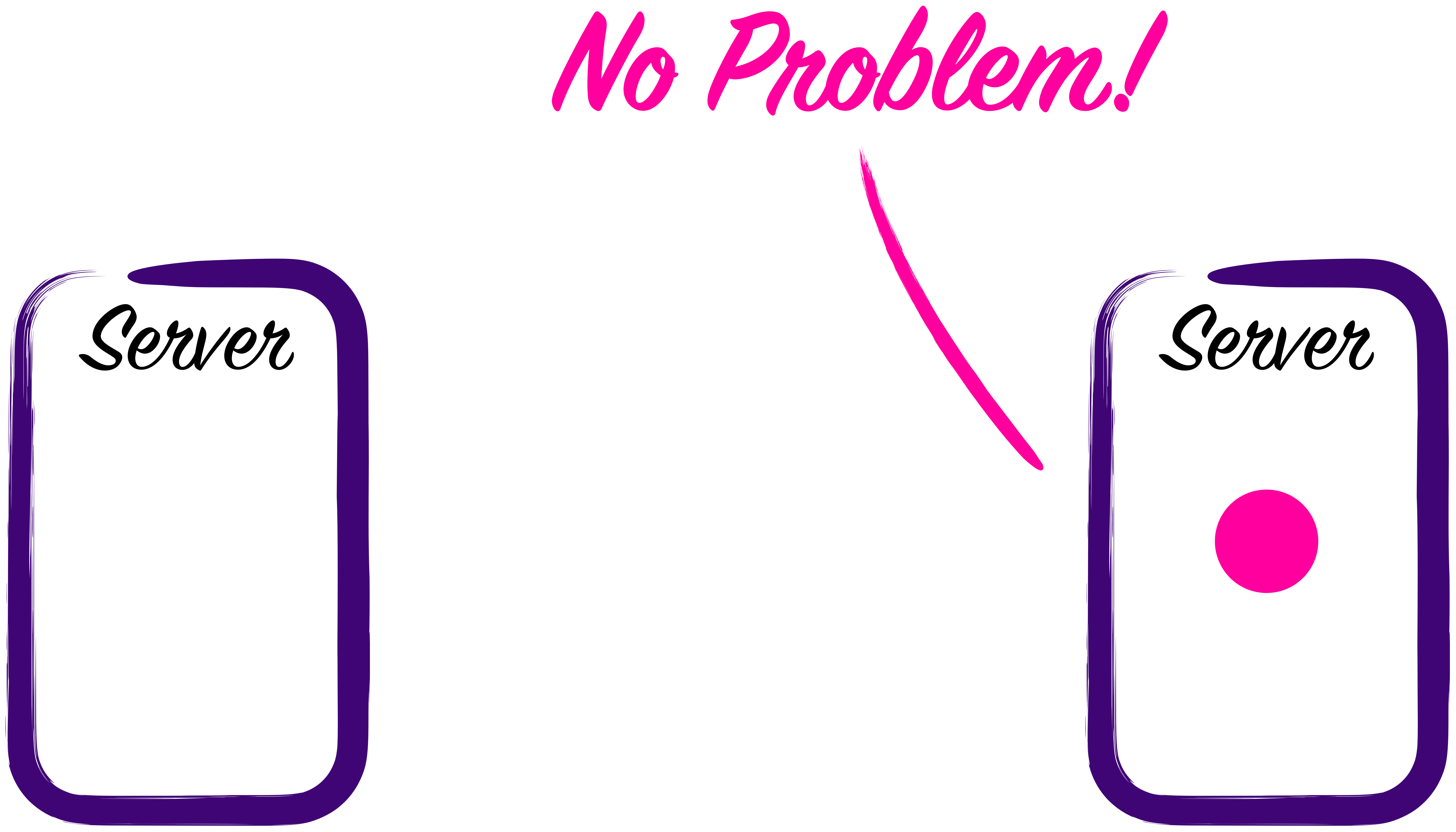

Server


Server

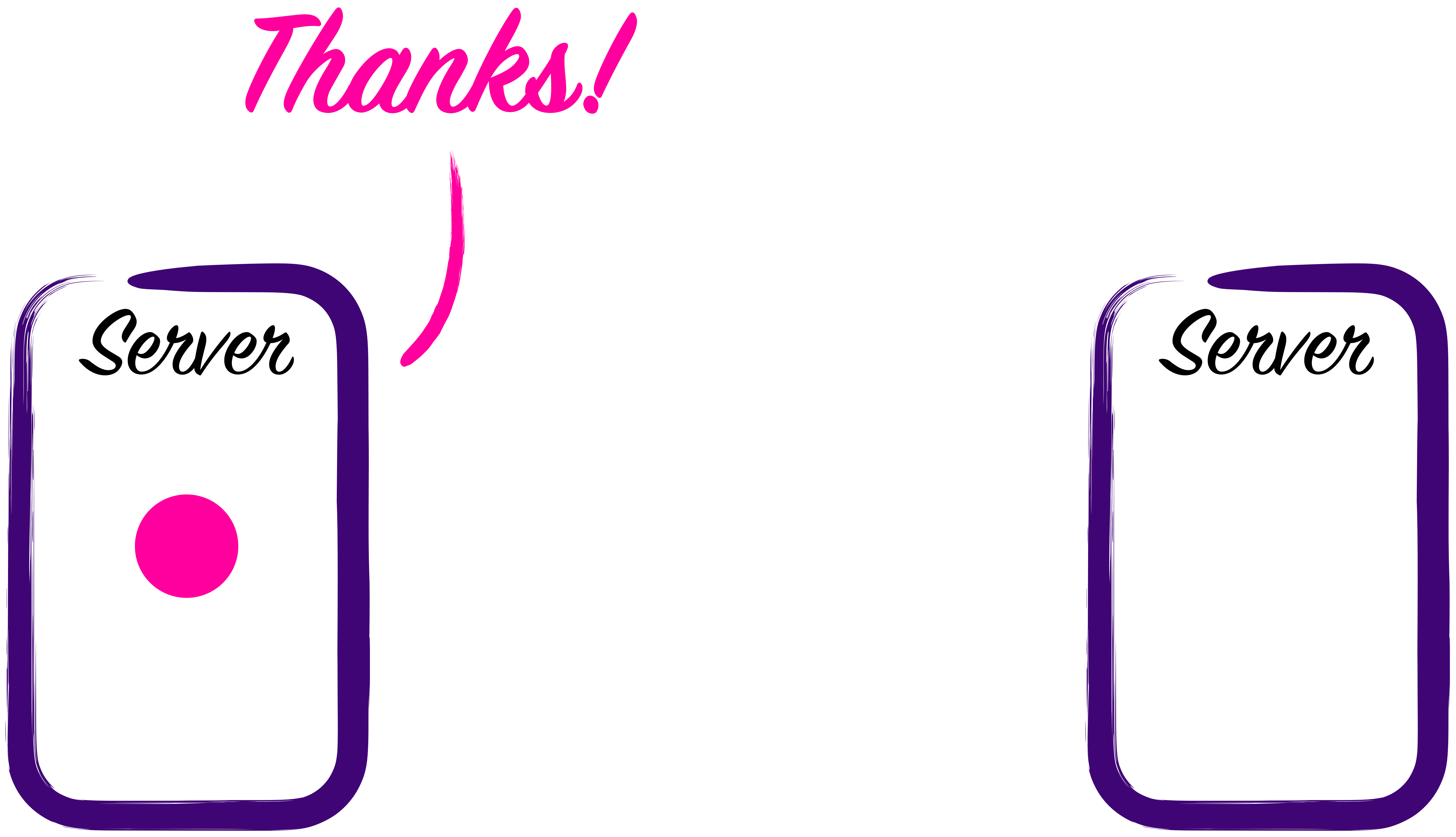

I have a request
Server
Server


Server
Server


Server
Server

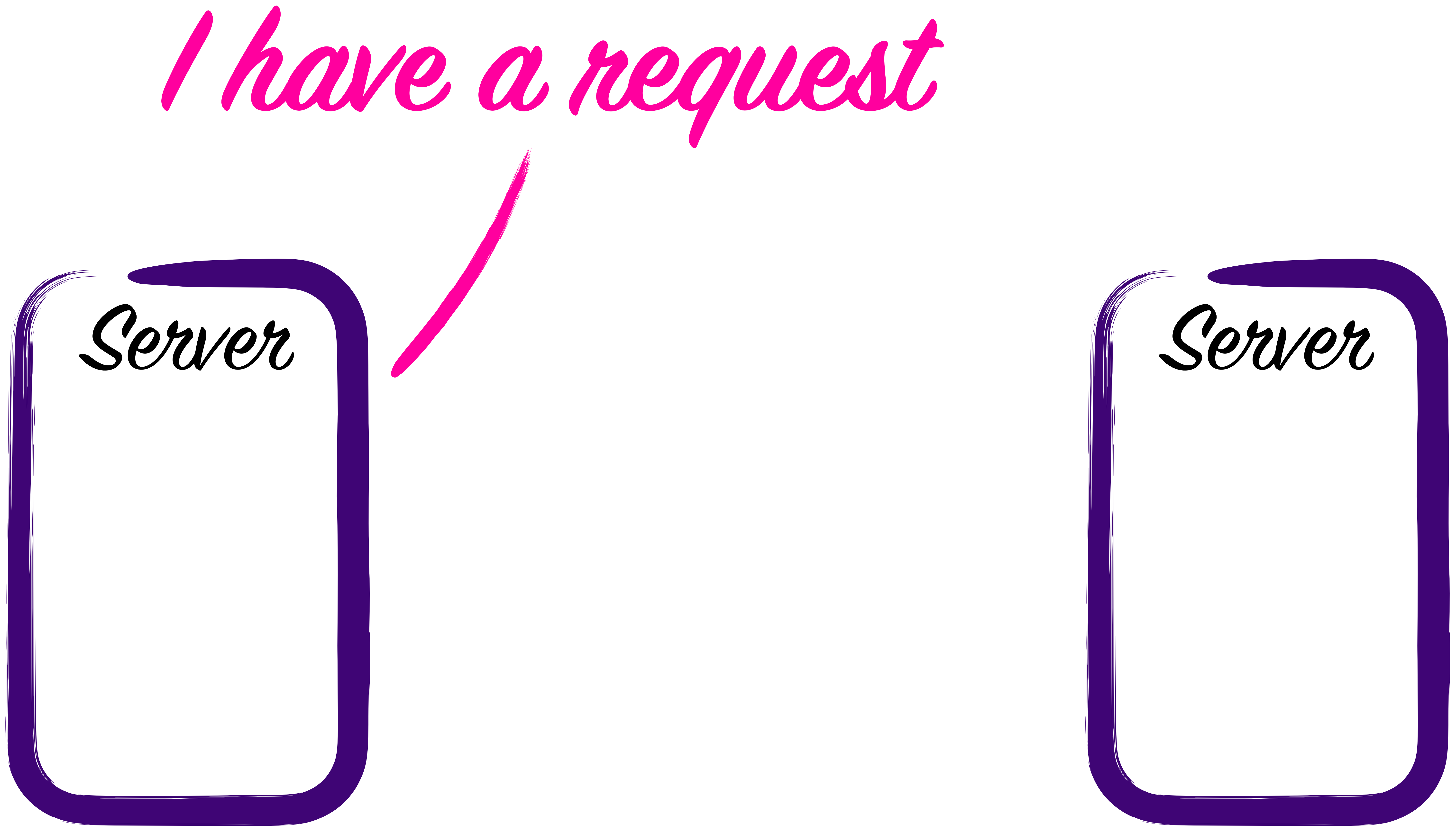

No Problem!
Server
Server


Server
Server


Thanks!
Server
Server

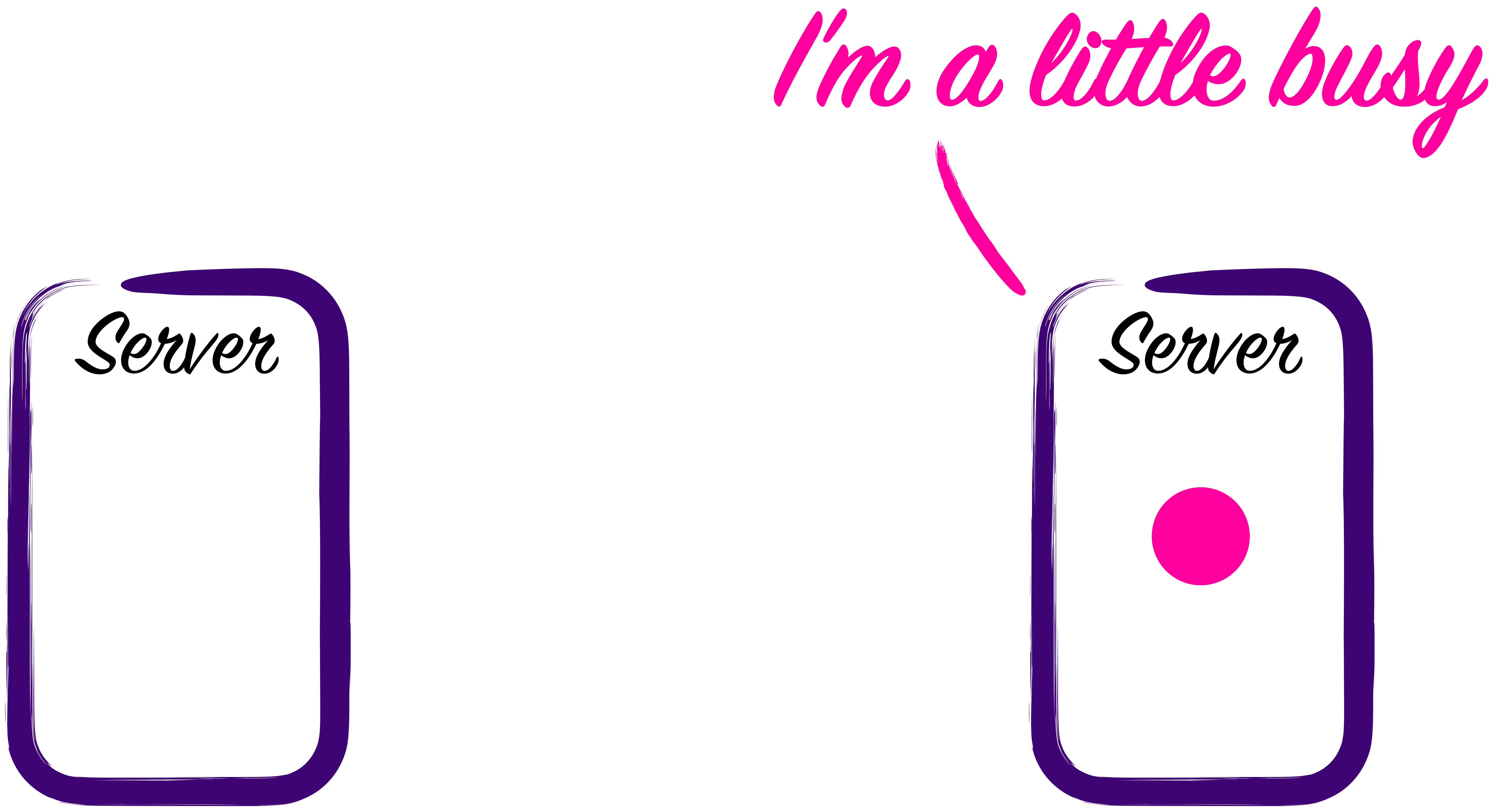

Server

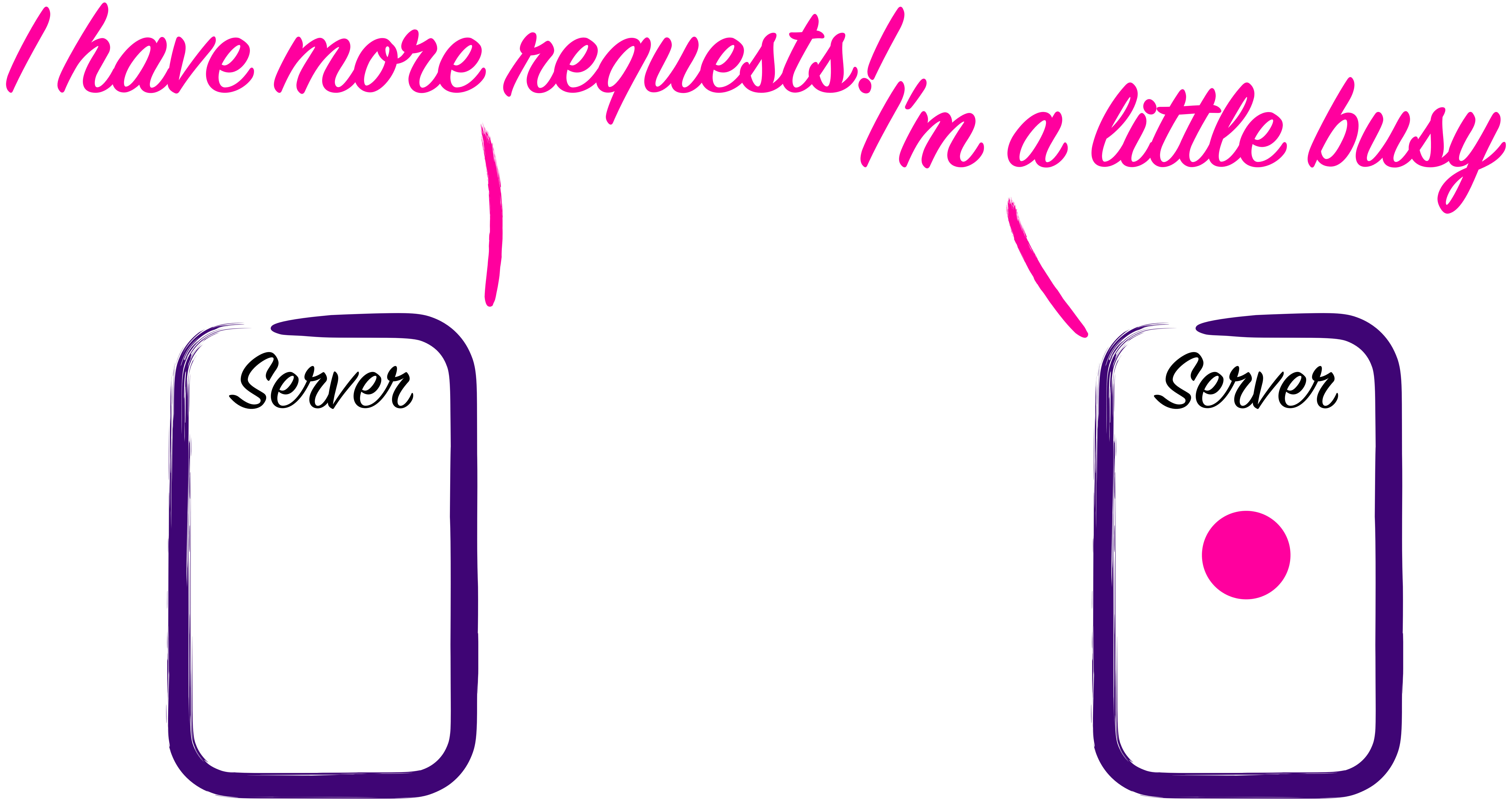

Server

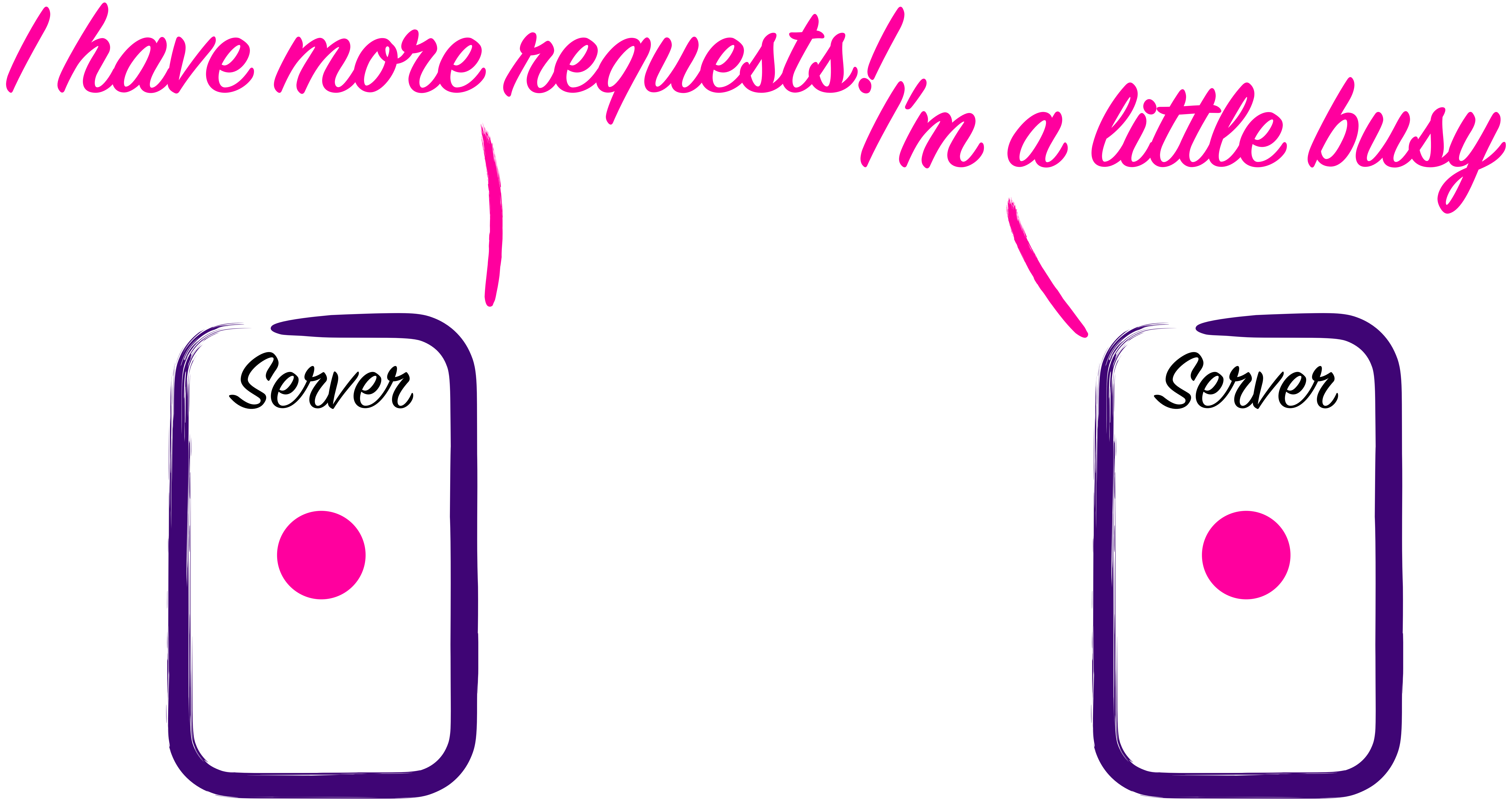

I have a request
Server
Server

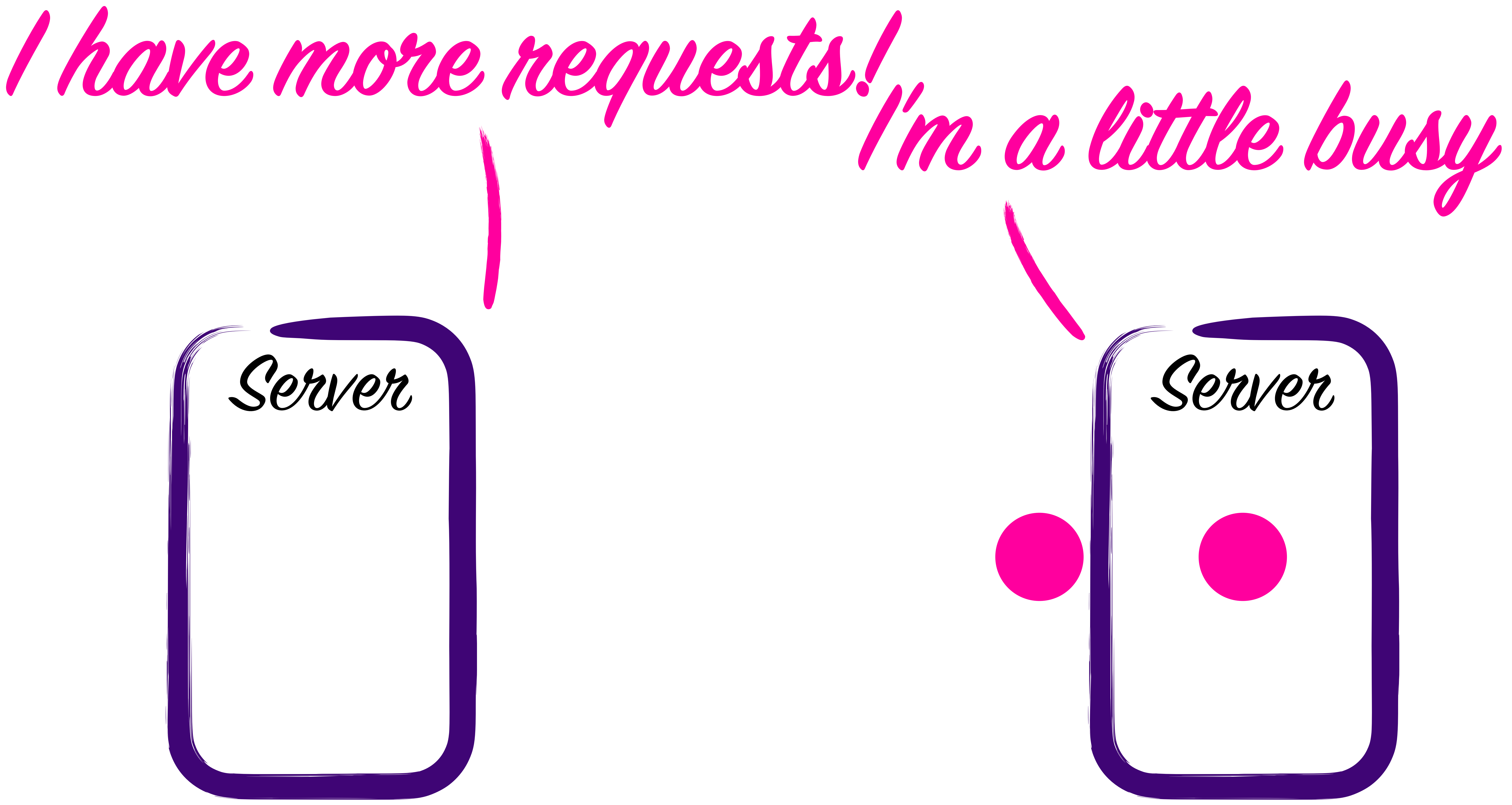

Server
Server

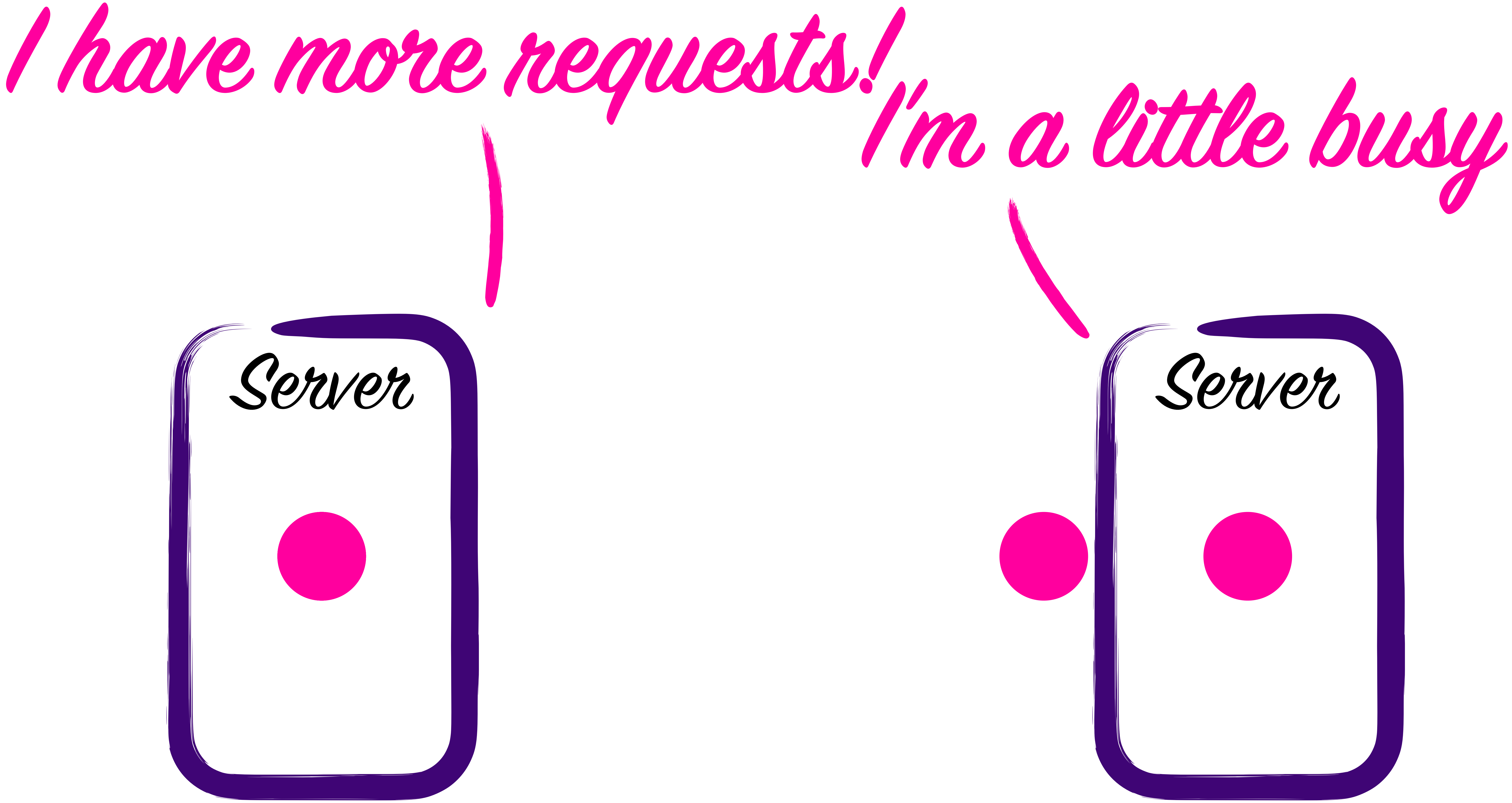

Server
Server

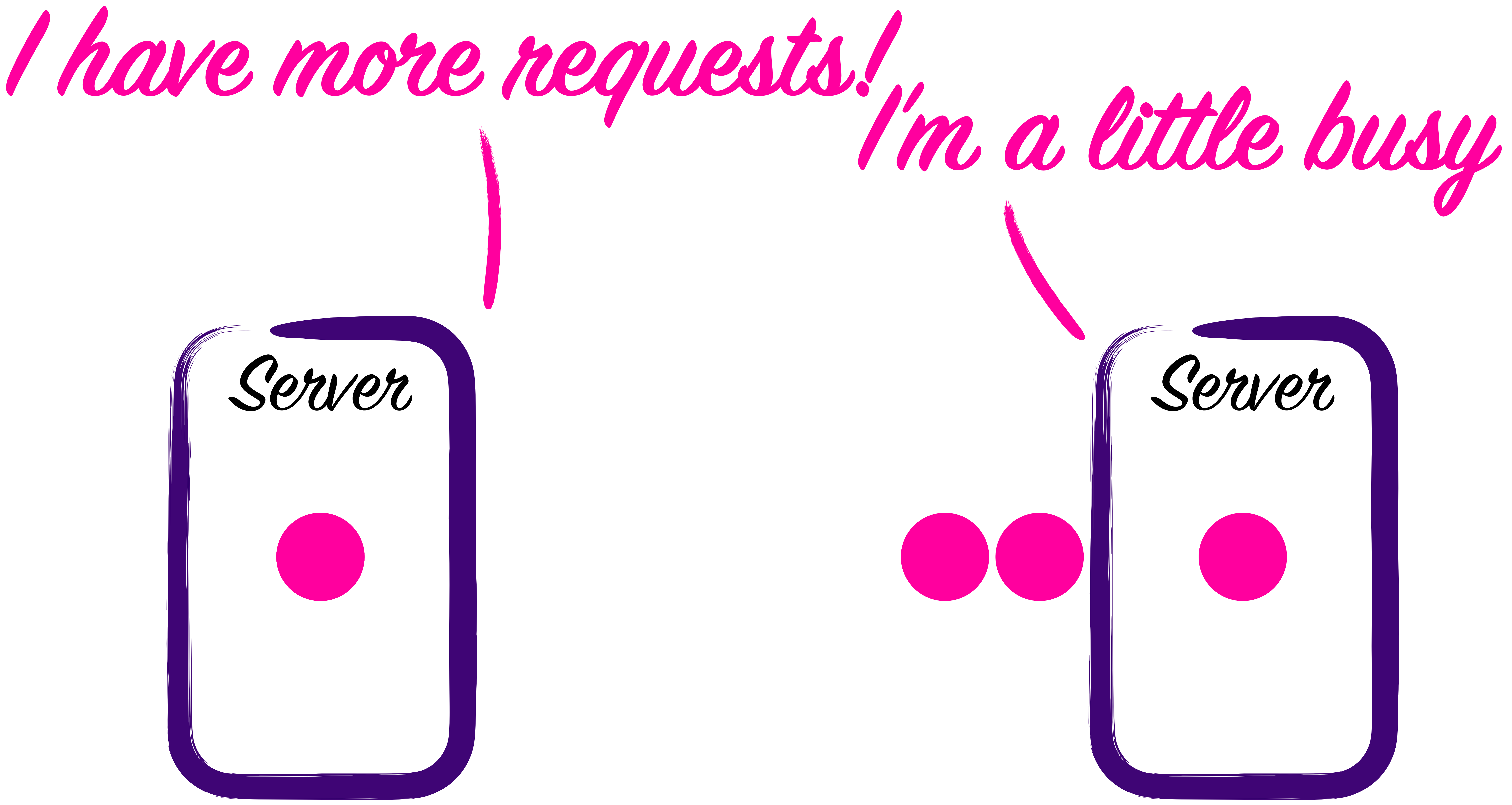

I'm a little busy
Server
Server

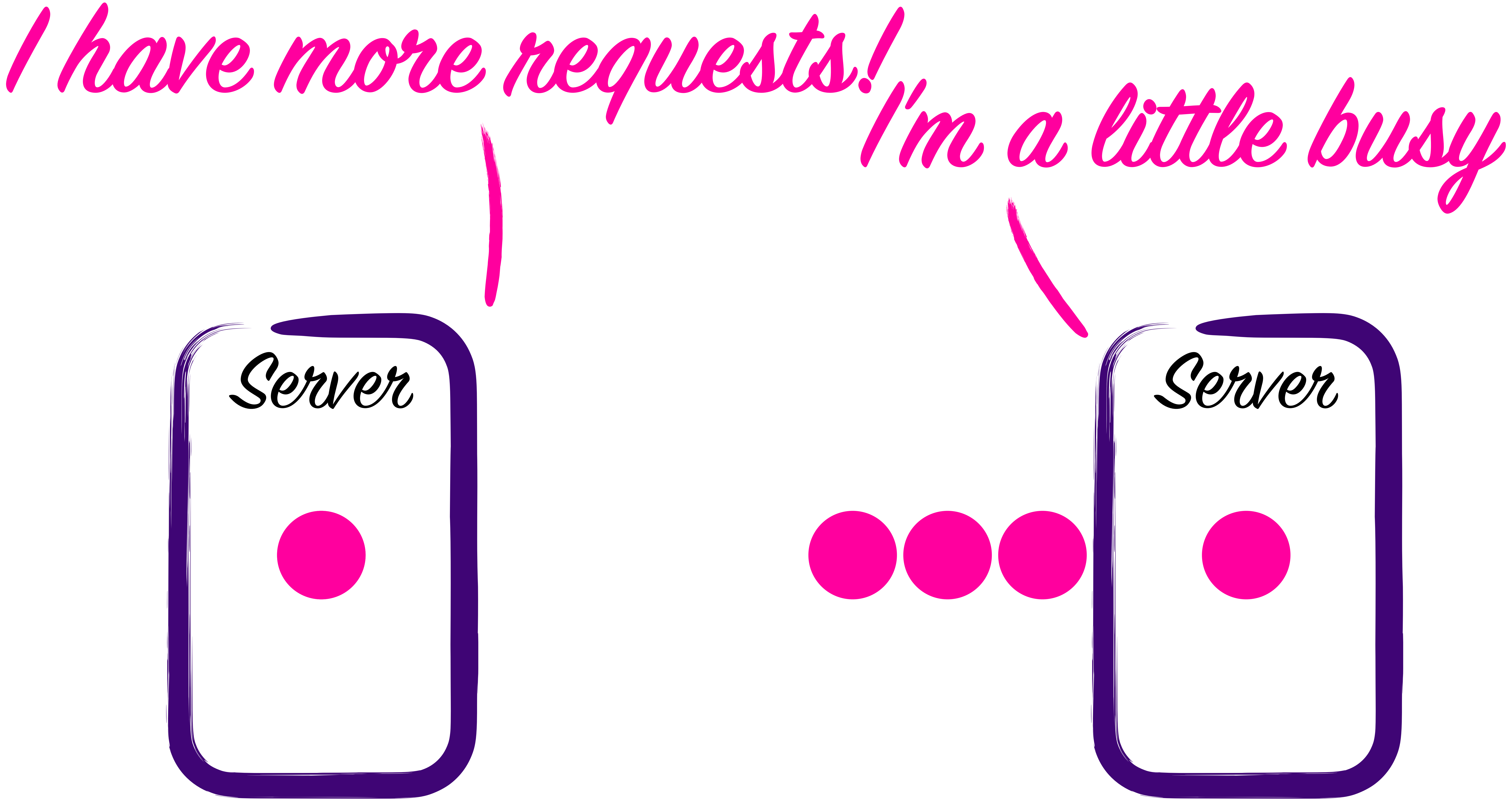

I have more requests!
I'm a little busy
Server
Server

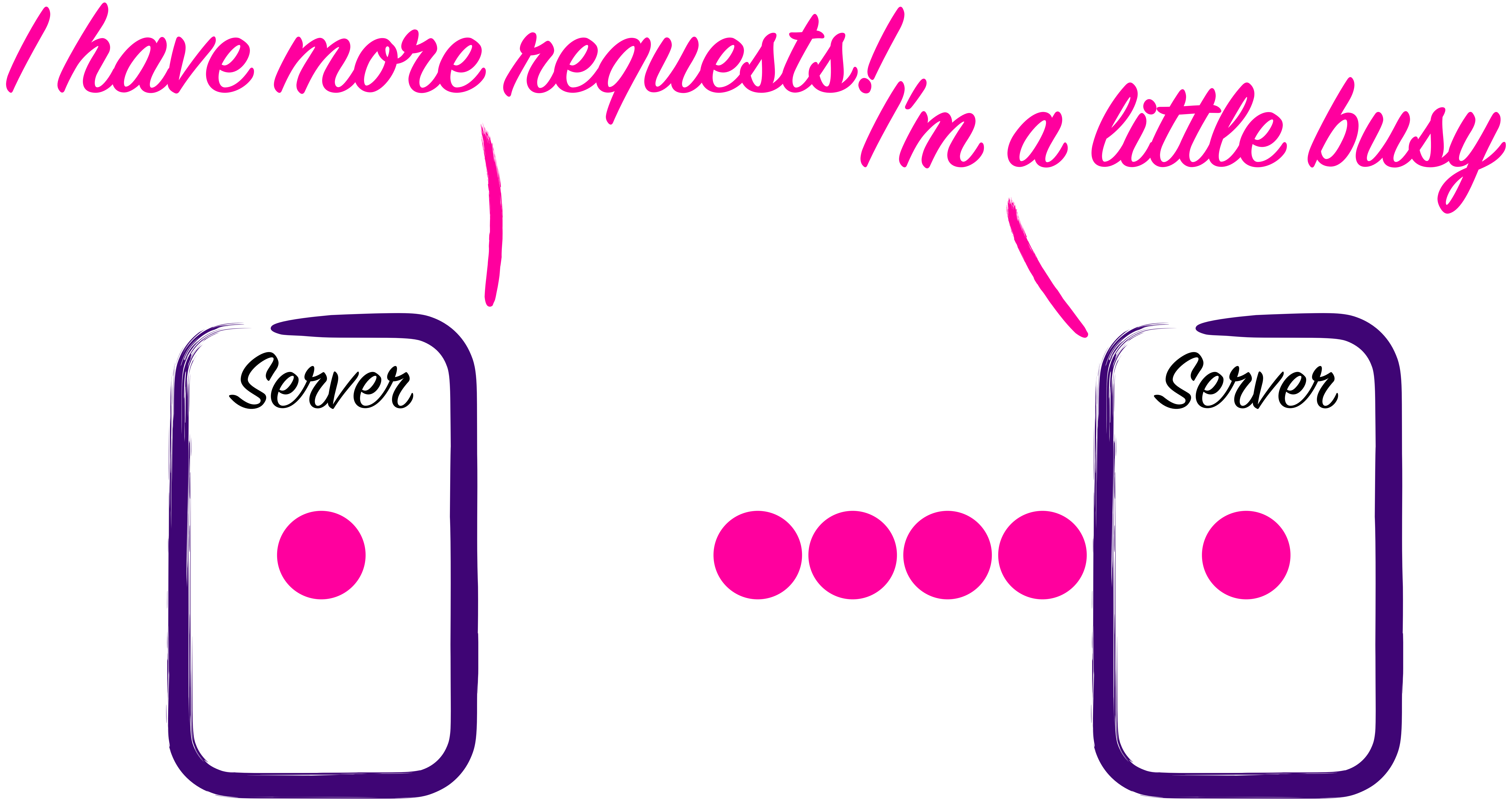

I have more requests!
I'm a little busy
Server
Server

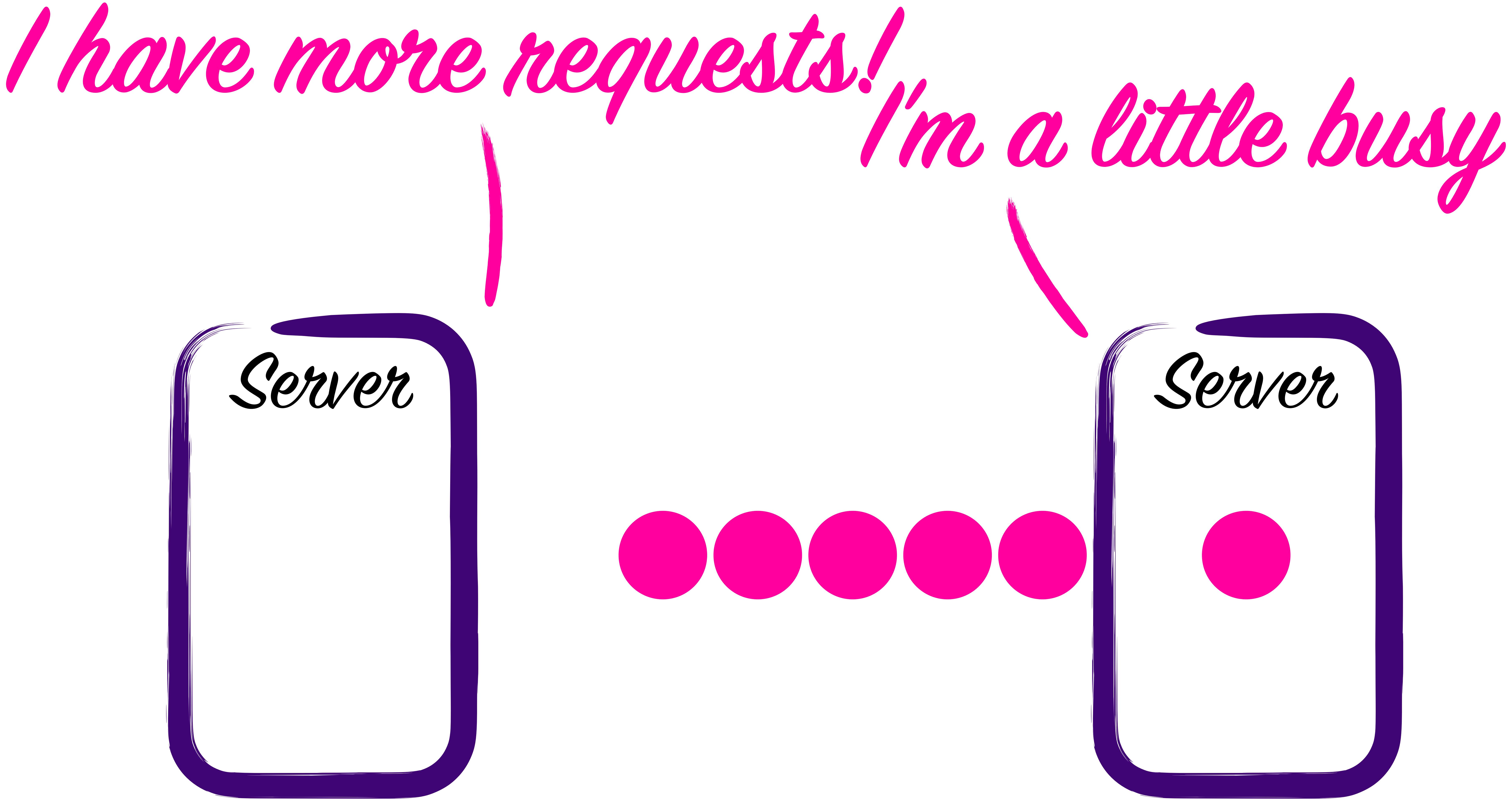

I have more requests!
I'm a little busy
Server
Server

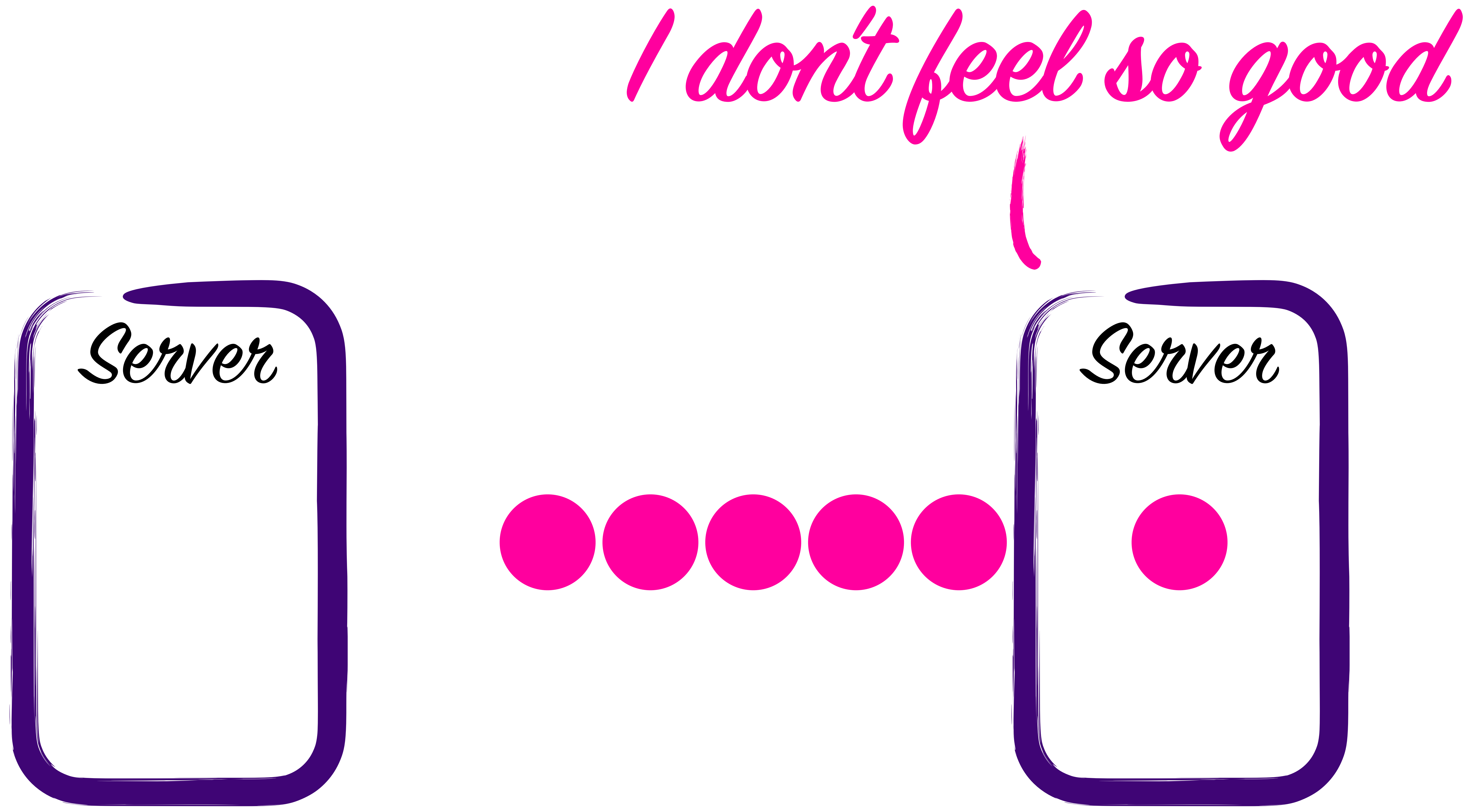

I have more requests!
I'm a little busy
Server
Server

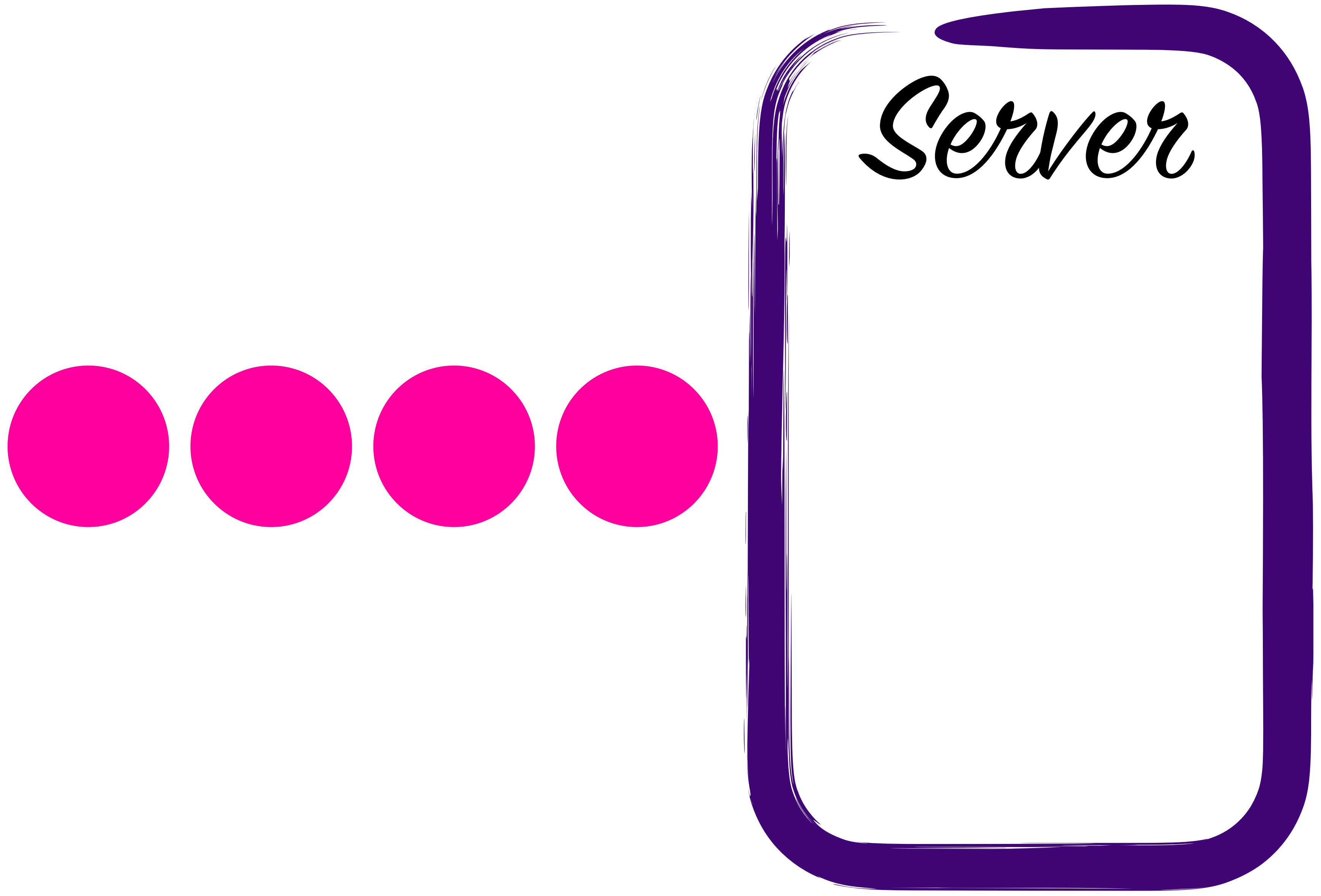

I have more requests!
I'm a little busy
Server
Server

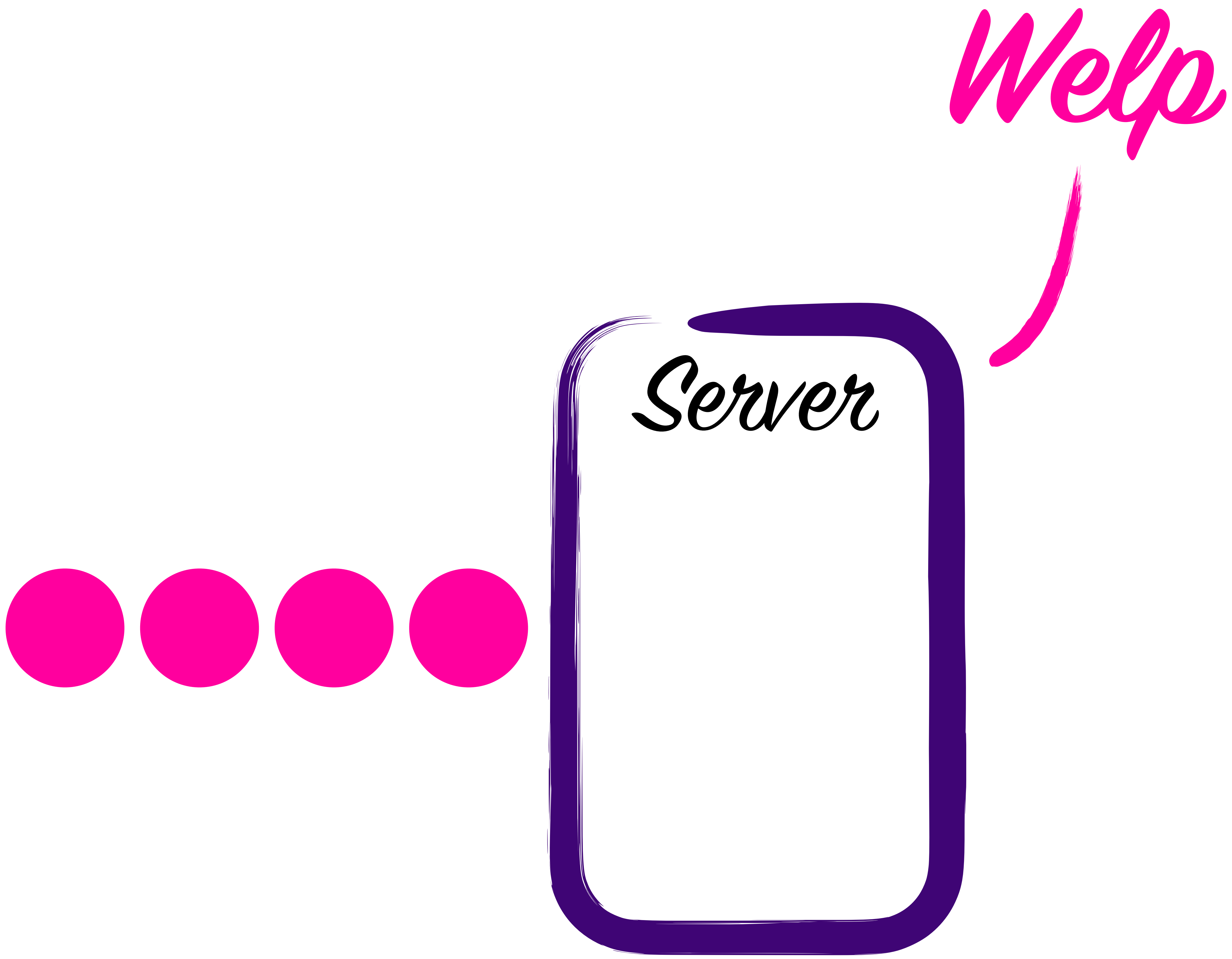

I have more requests!
I'm a little busy
Server
Server

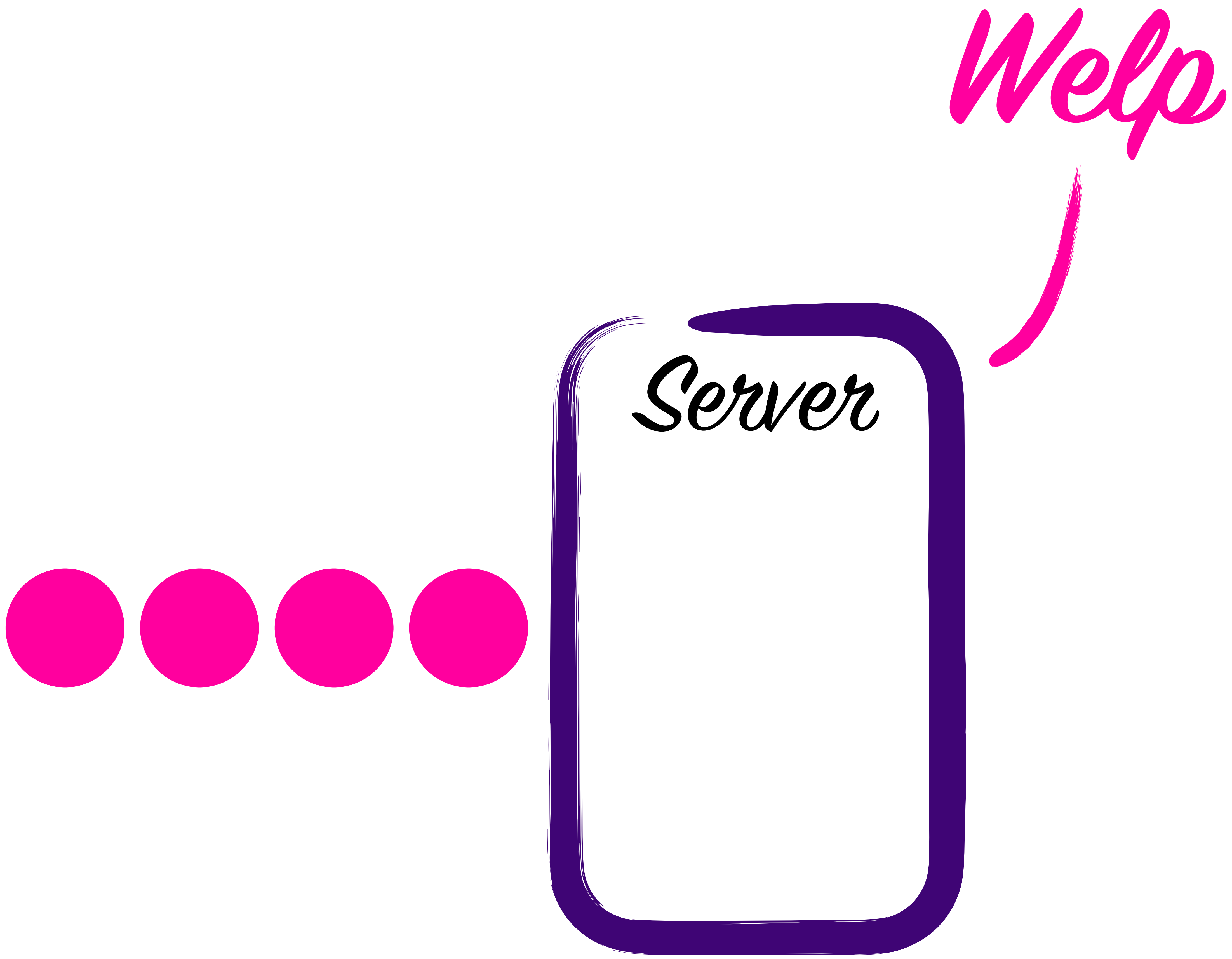

I have more requests!
I'm a little busy
Server
Server


I have more requests!
I'm a little busy
Server
Server


I don't feel so good
Server
Server


Server


Welp
Server


Welp
Server

All services have objectives

A resilient service should be able to withstand a 10x traffic spike and continue to meet those objectives

Lets Talk About…

Queues

Overload Mitigation

Adaptive Concurrency

Lets Talk About…

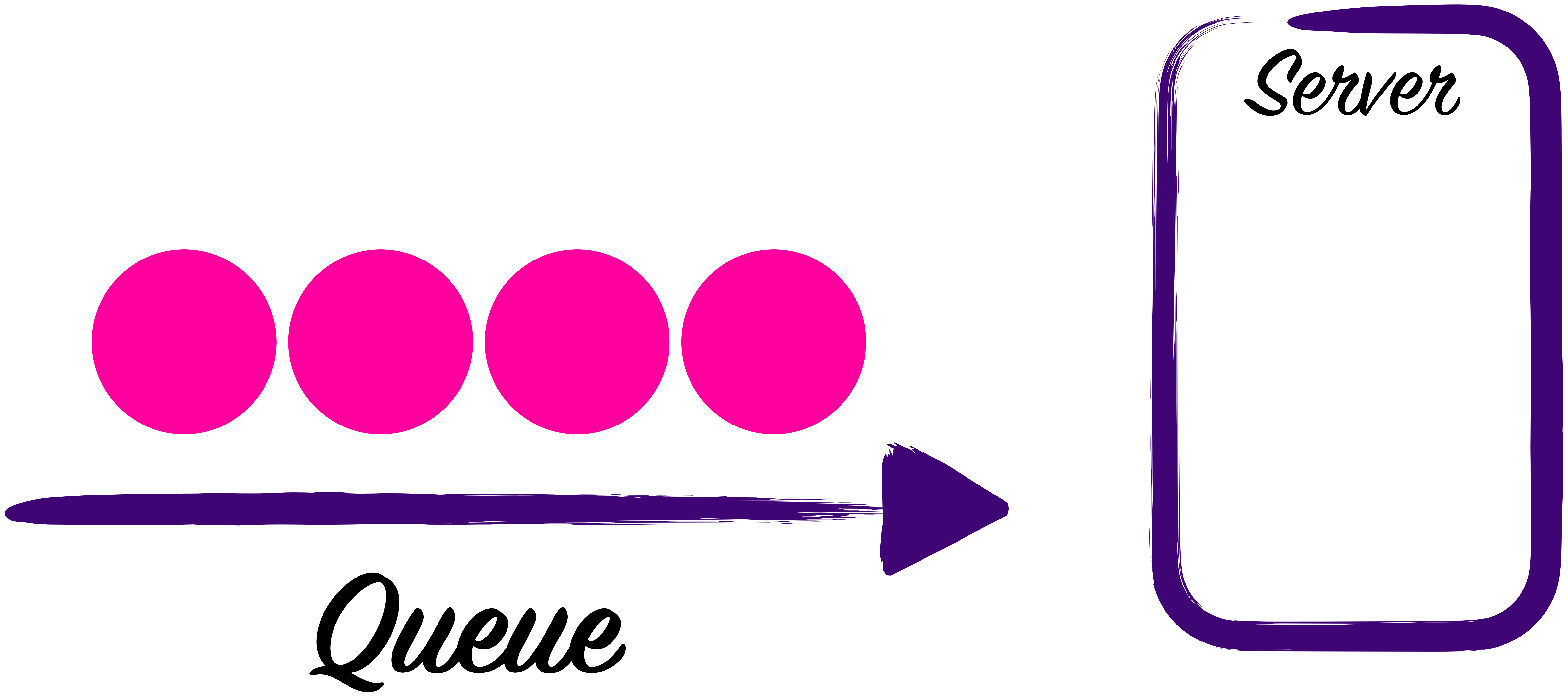
Queues

Overload Mitigation

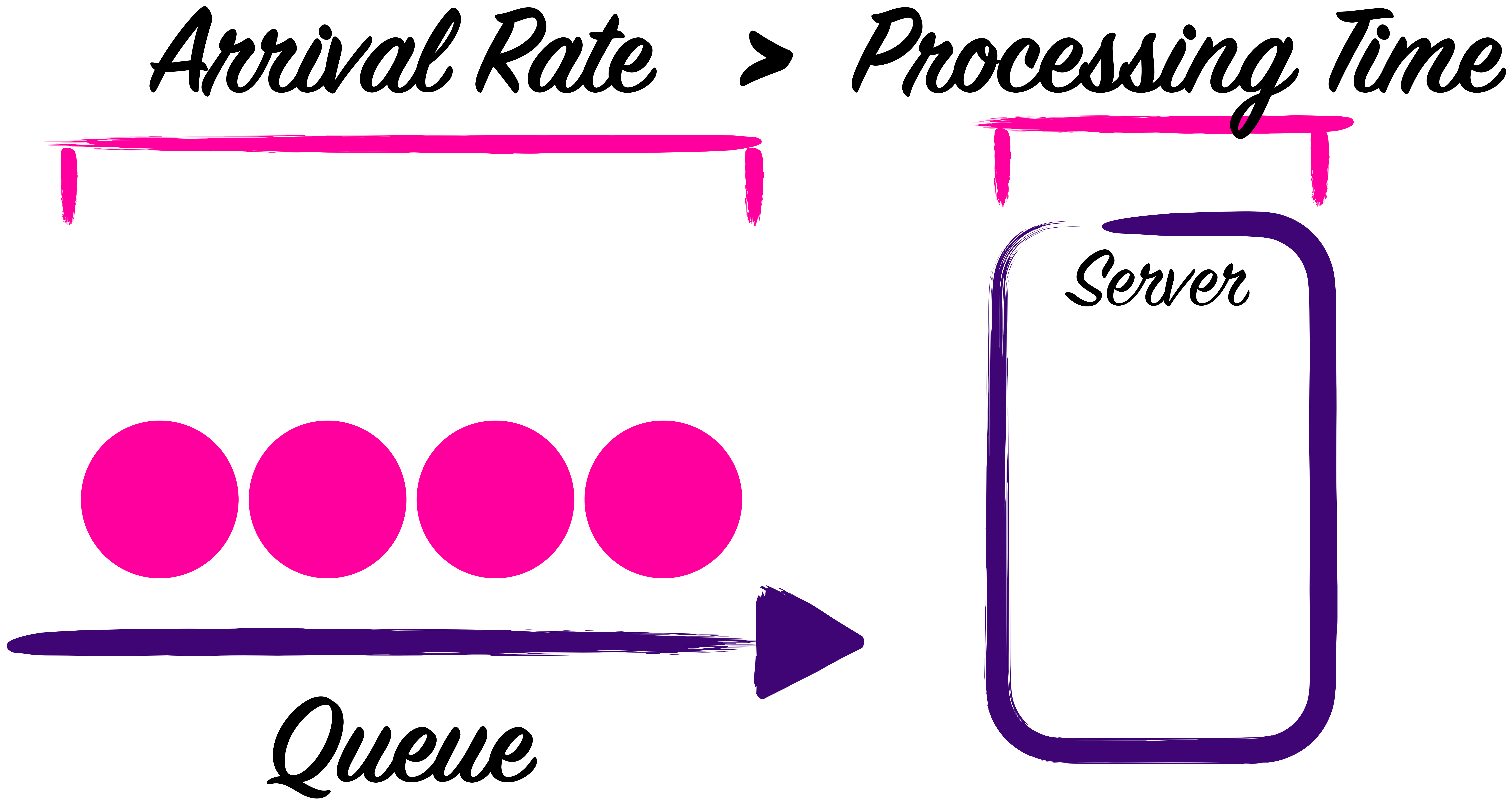
Adaptive Concurrency

What causes overload?

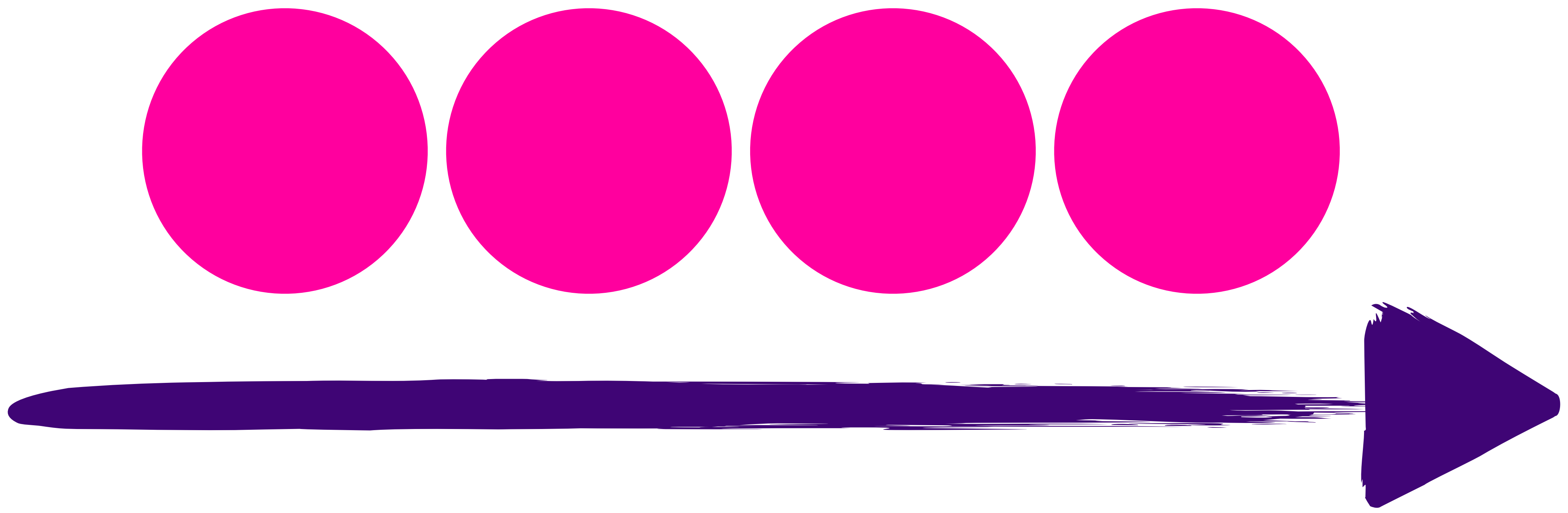
What causes overload?



What causes overload?



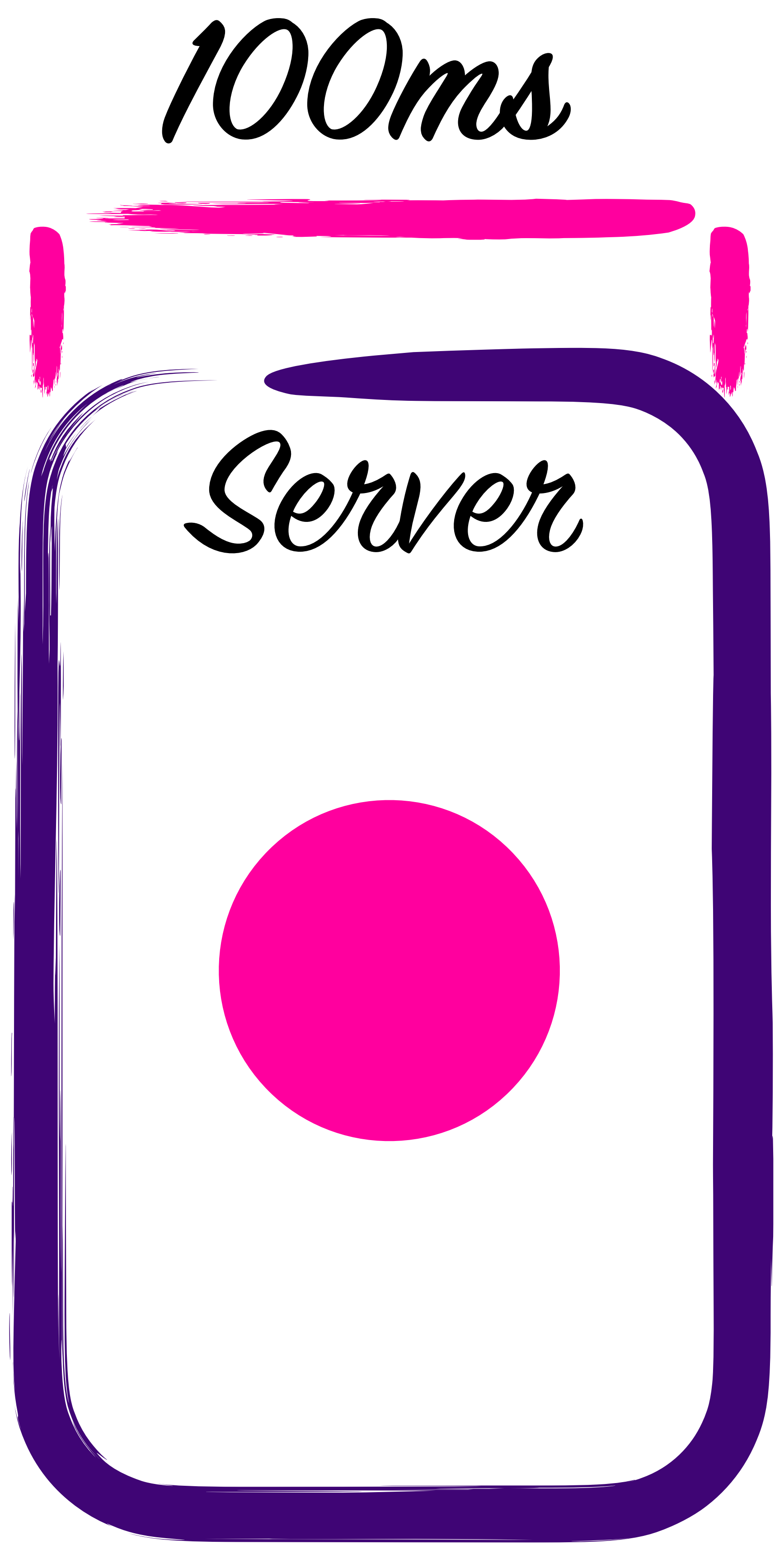
Little's Law



*Elements in the queue = Arrival Rate * Processing Time*

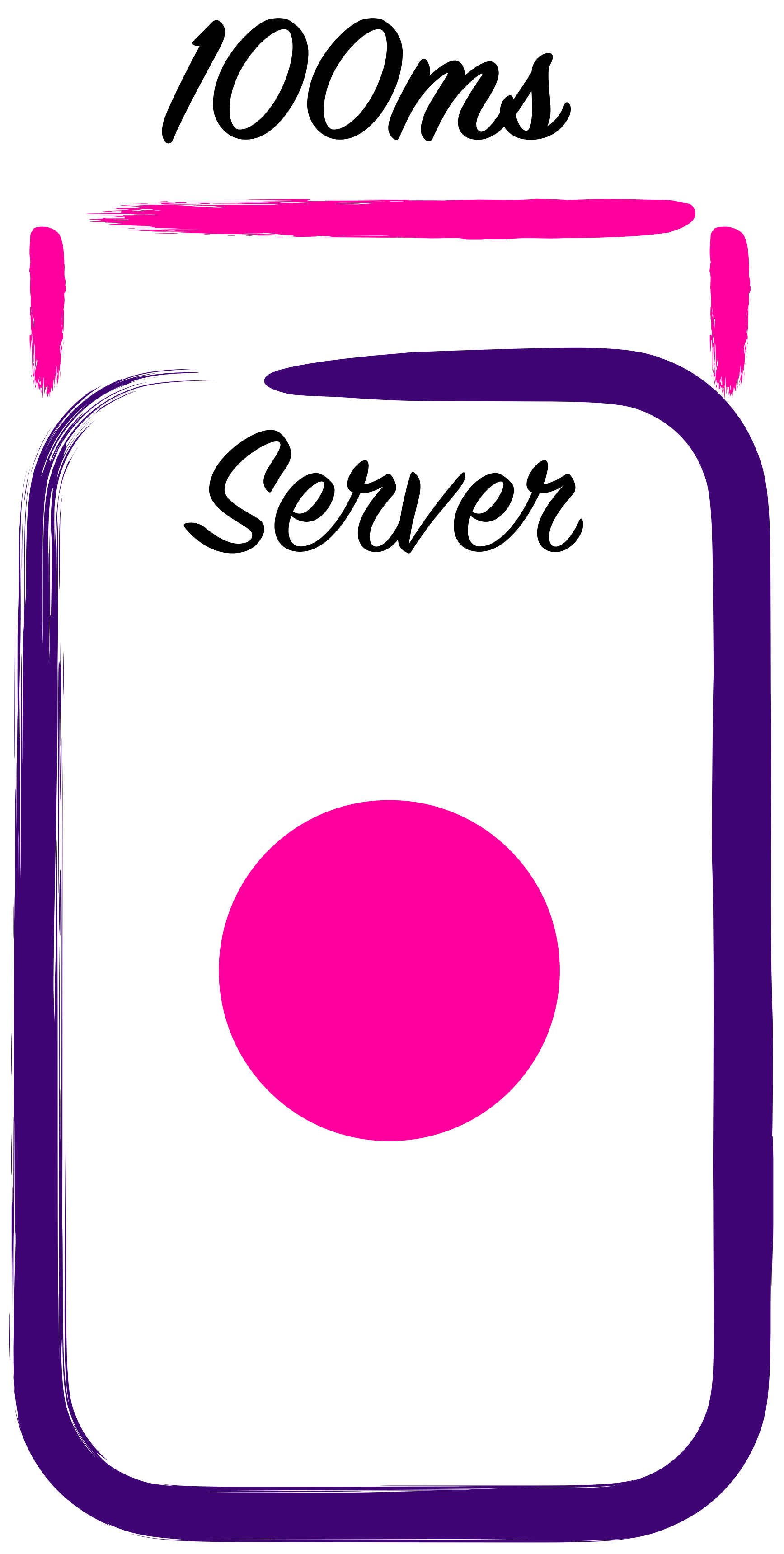
Little's Law

1 requests = 10 rps * 100 ms



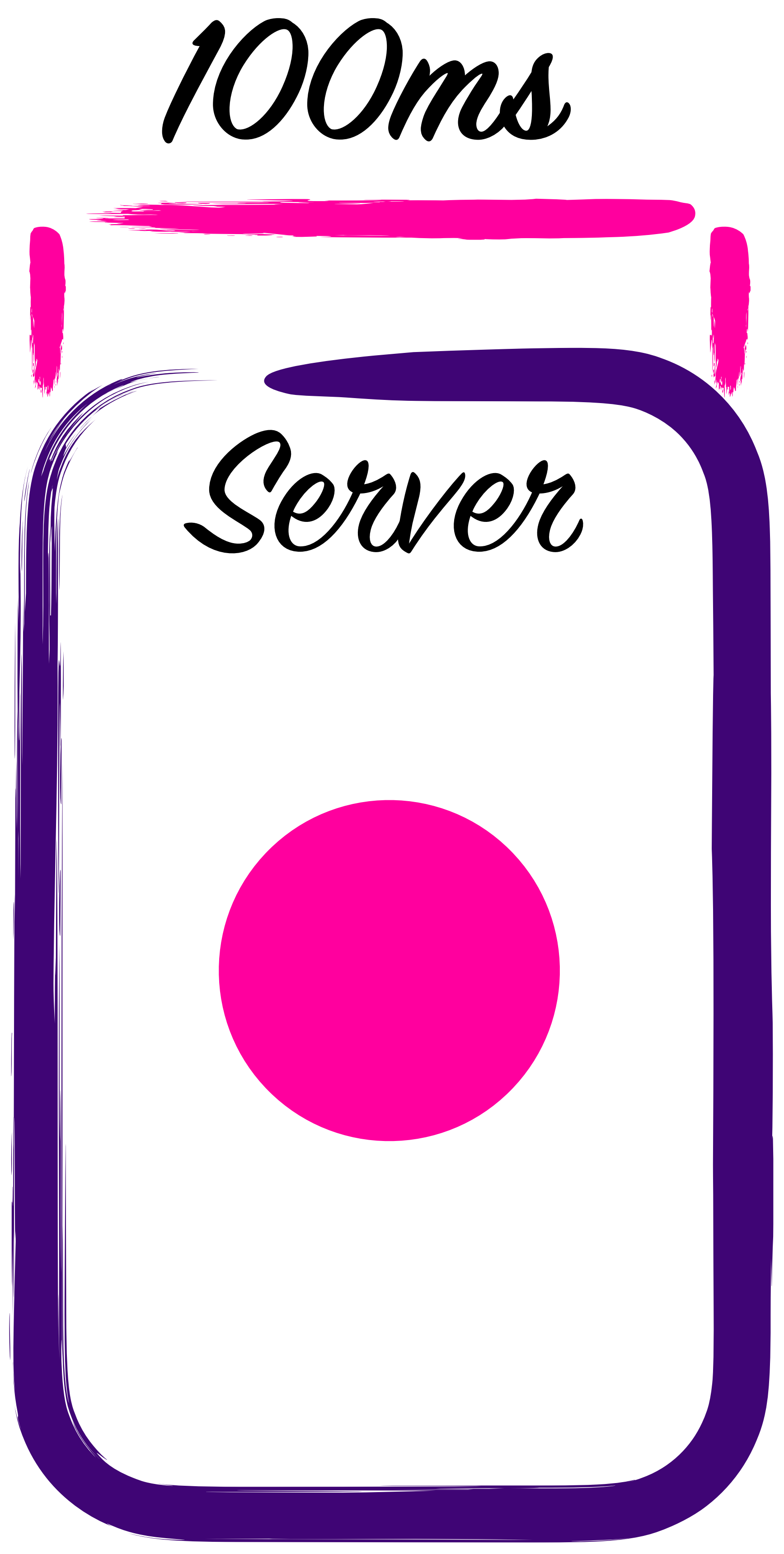
Little's Law

1 requests = 10 rps * 100 ms



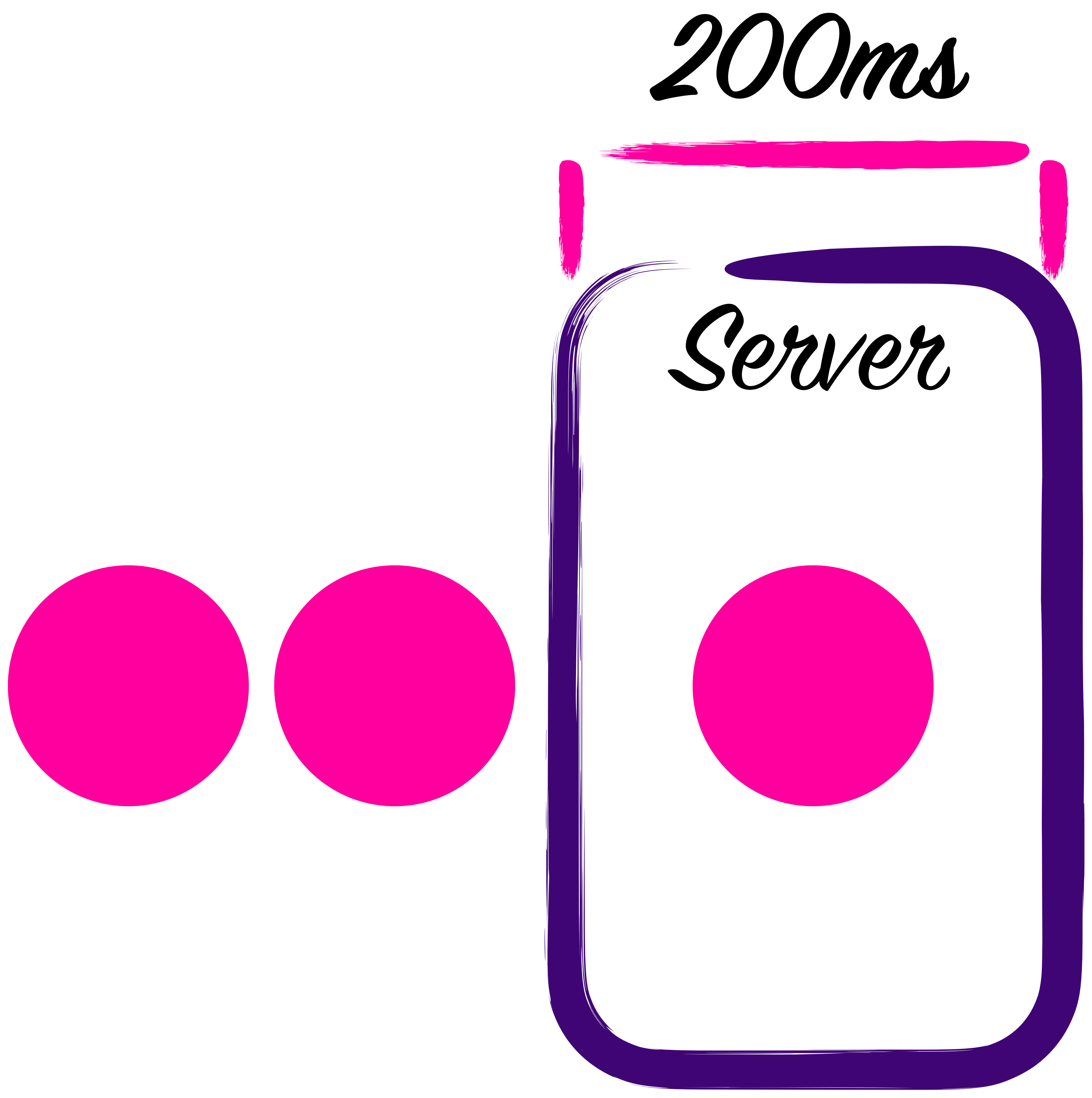
Little's Law

*1 requests = 10 rps * 100 ms*



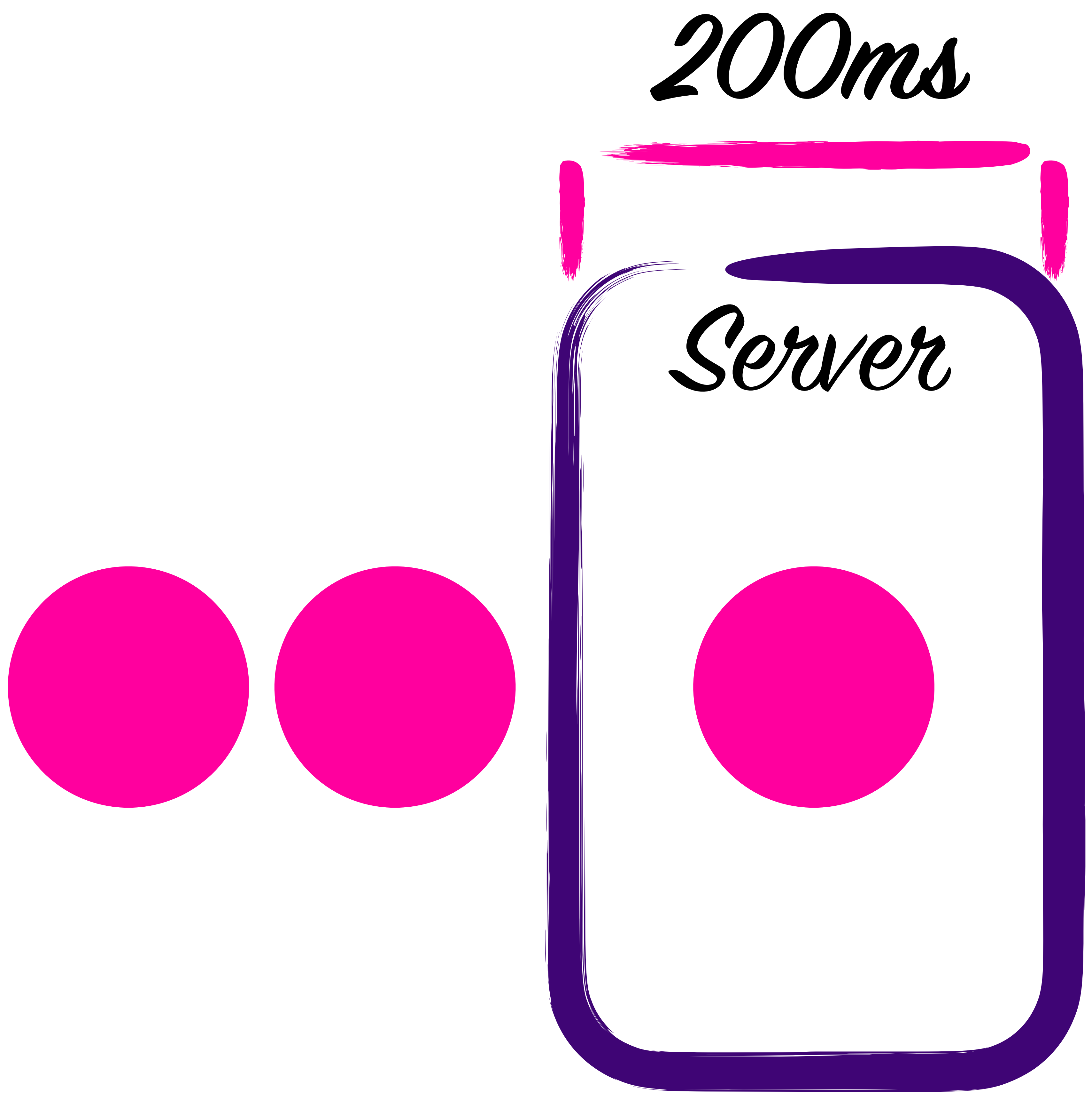
Little's Law

2 requests = 10 rps * 200 ms



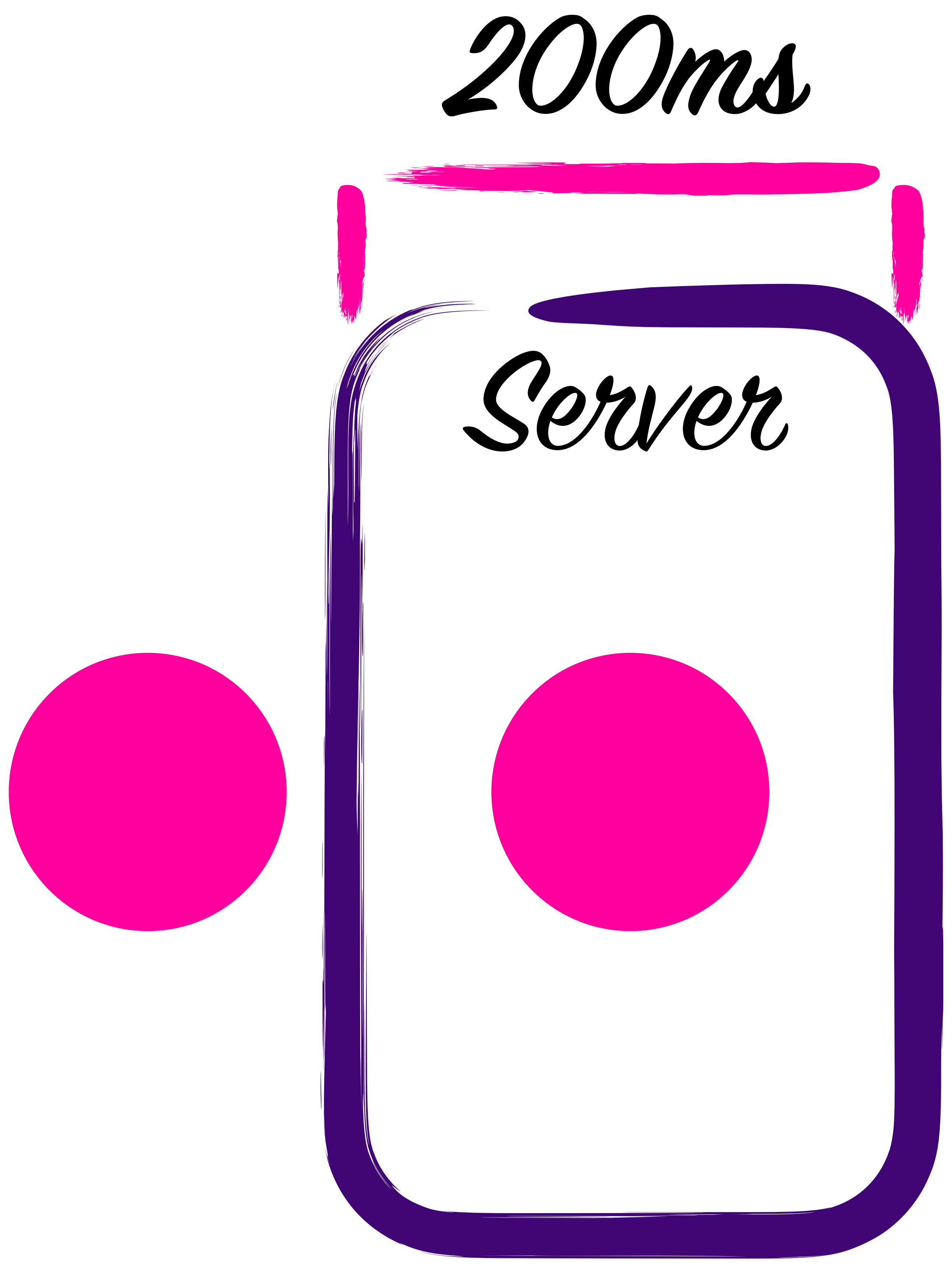
Little's Law

2 requests = 10 rps * 200 ms



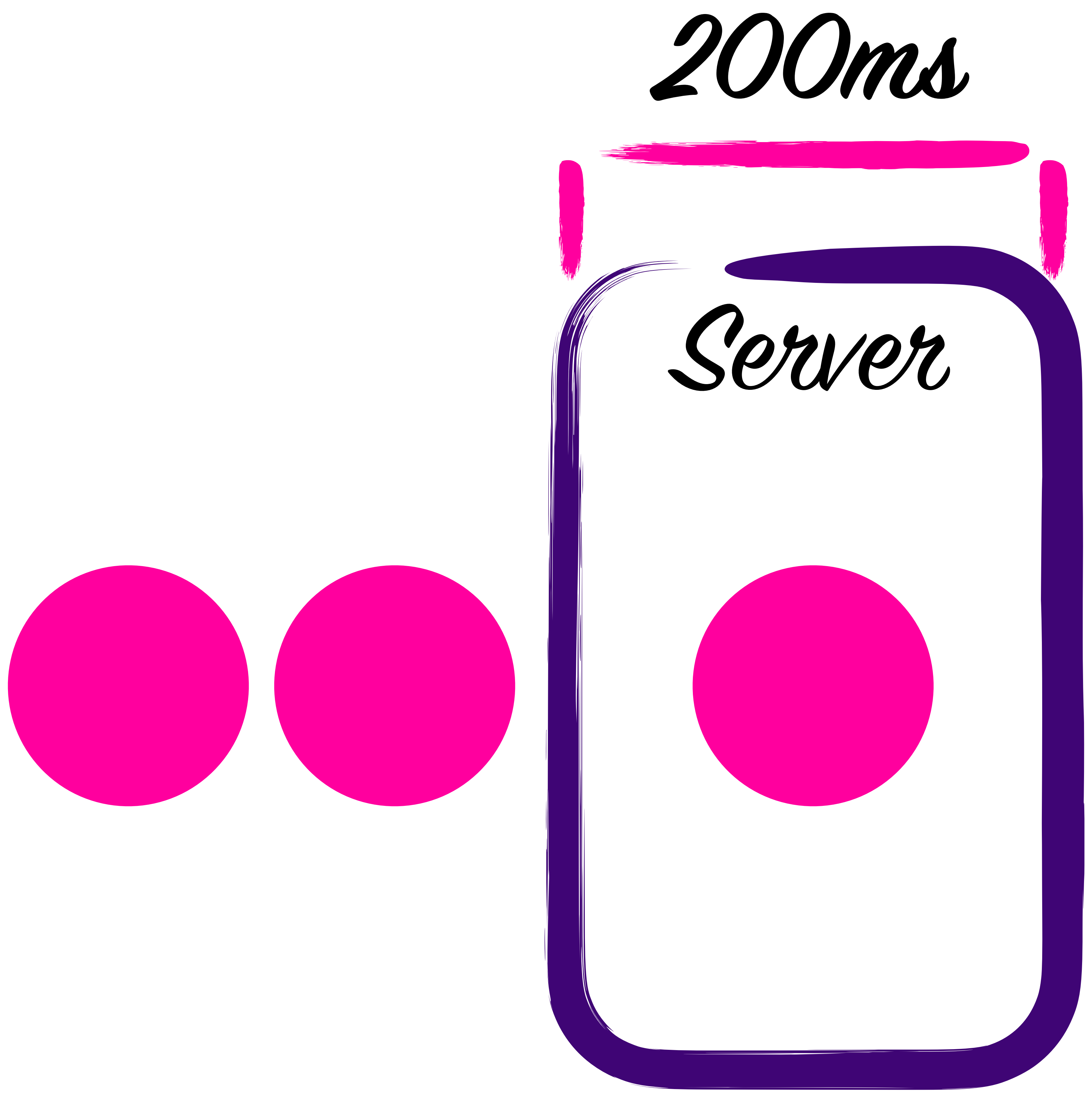
Little's Law

2 requests = 10 rps * 200 ms



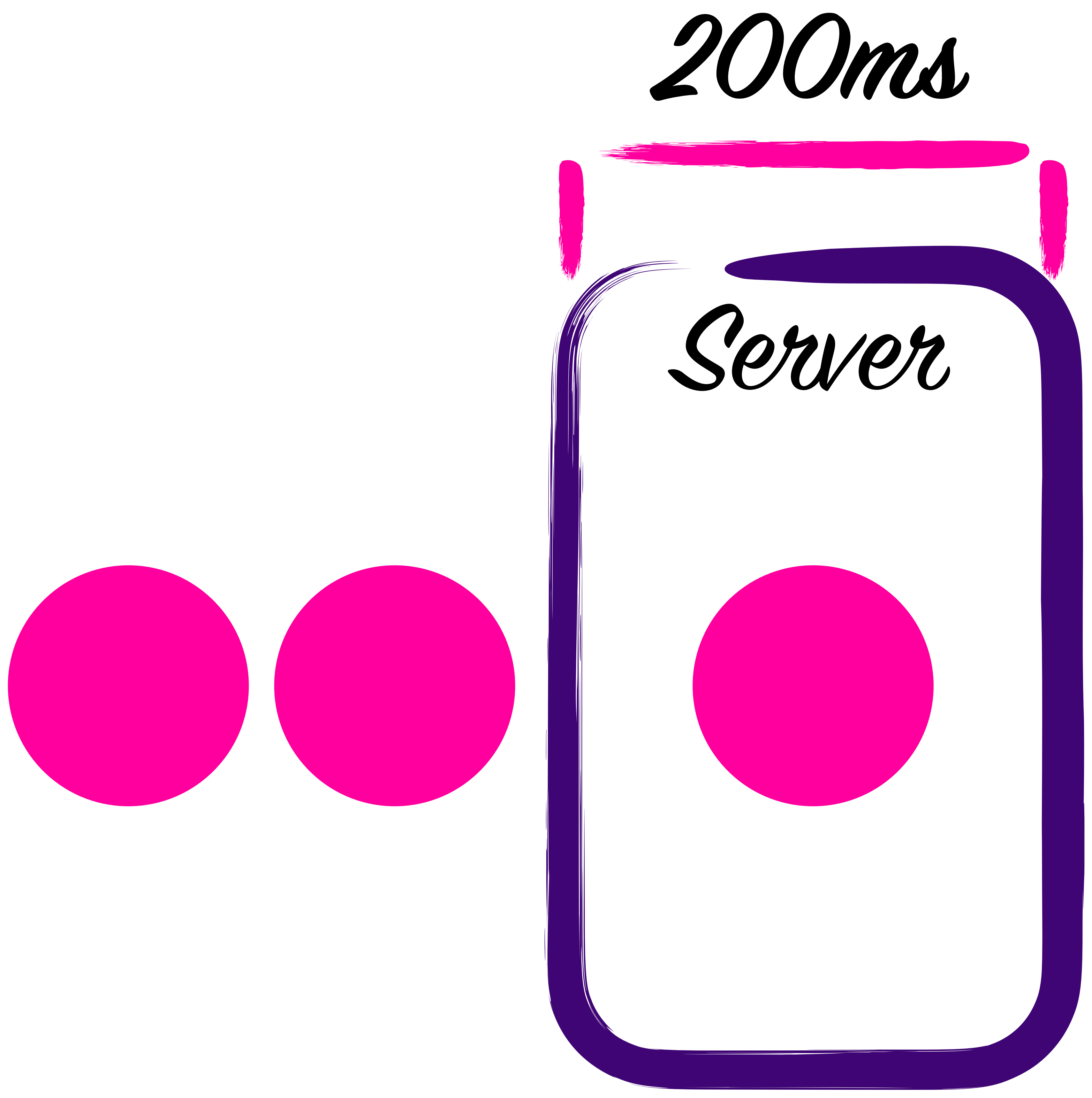
Little's Law

2 requests = 10 rps * 200 ms



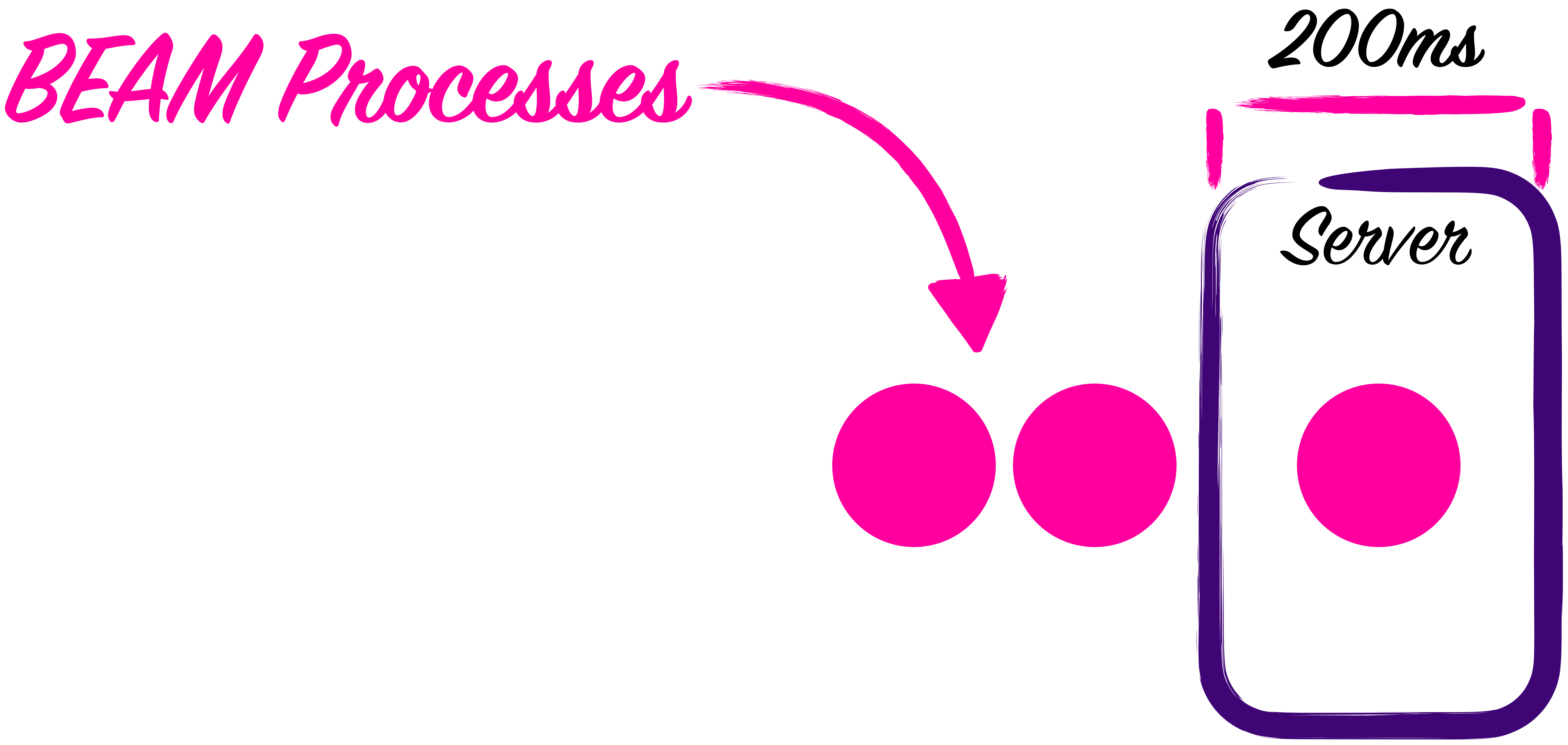
Little's Law

2 requests = 10 rps * 200 ms



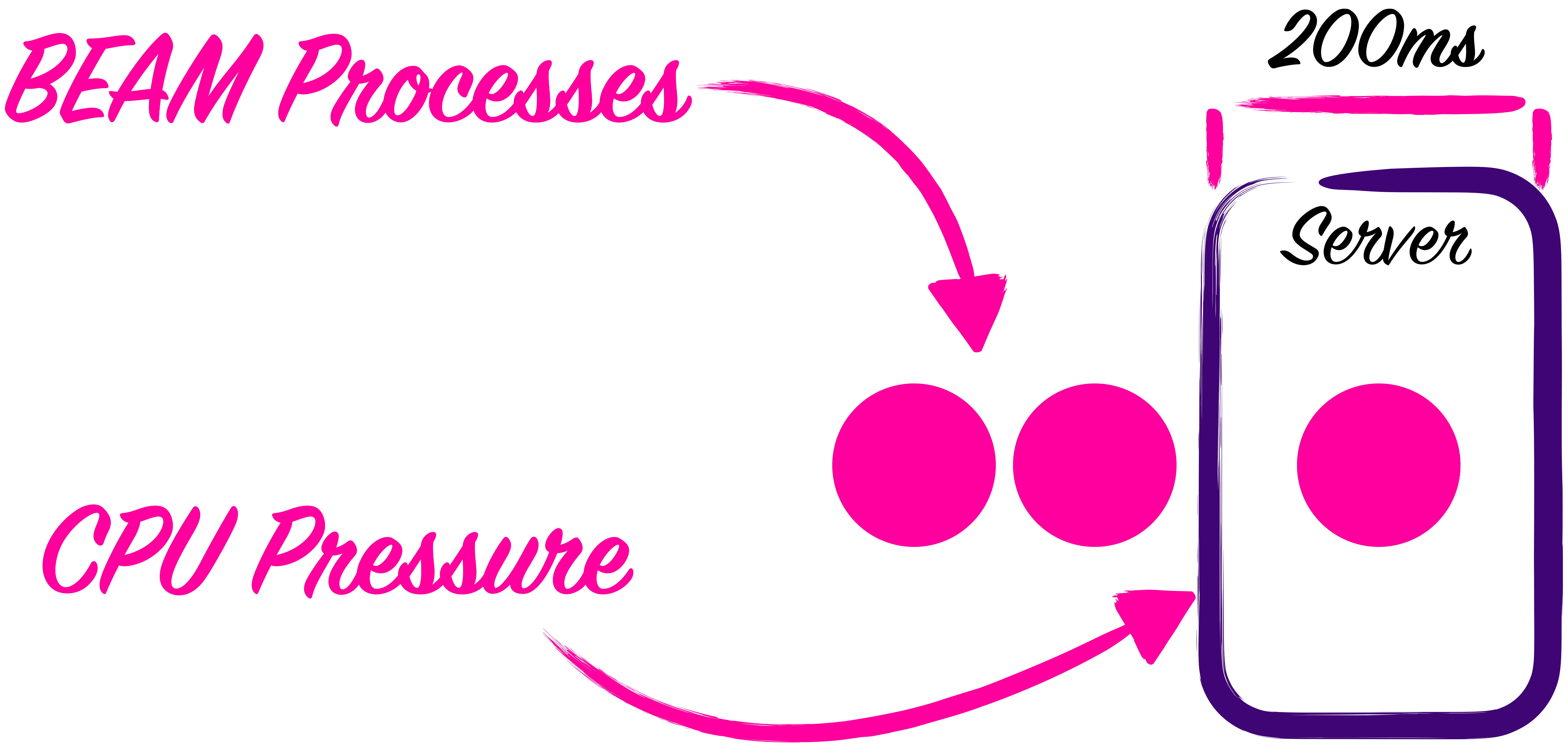
Little's Law

2 requests = 10 rps * 200 ms



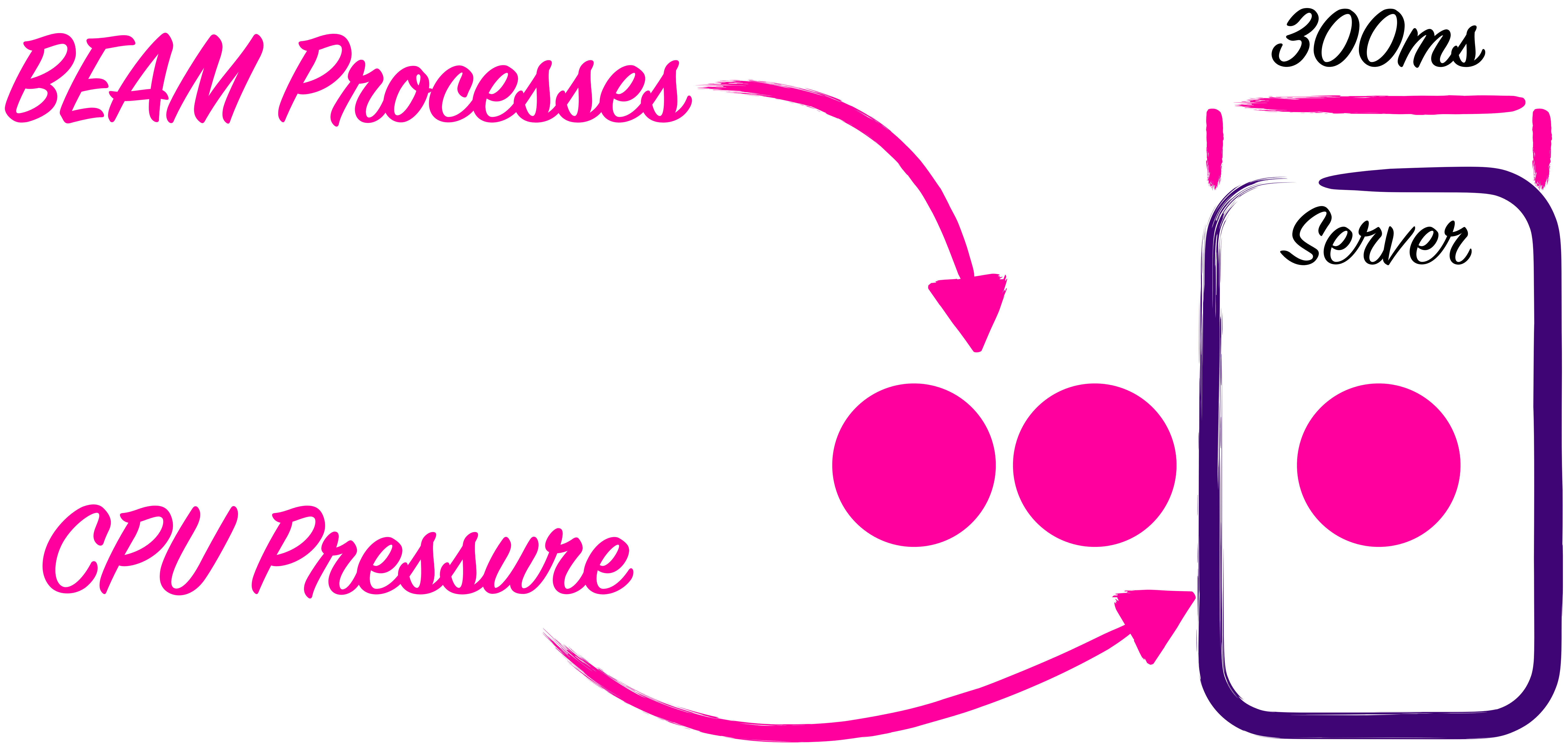
Little's Law

2 requests = 10 rps * 200 ms



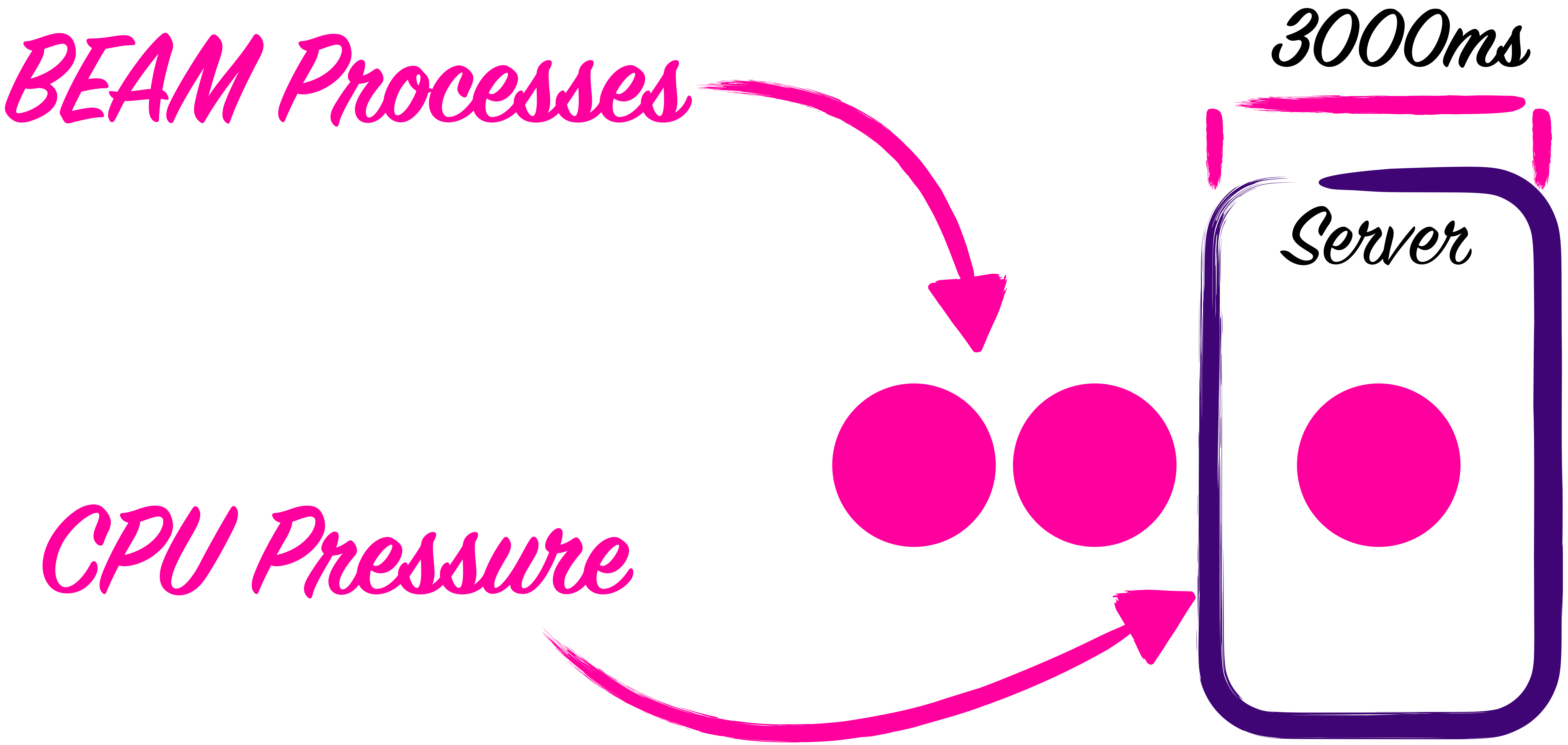
Little's Law

3 requests = 10 rps * 300 ms



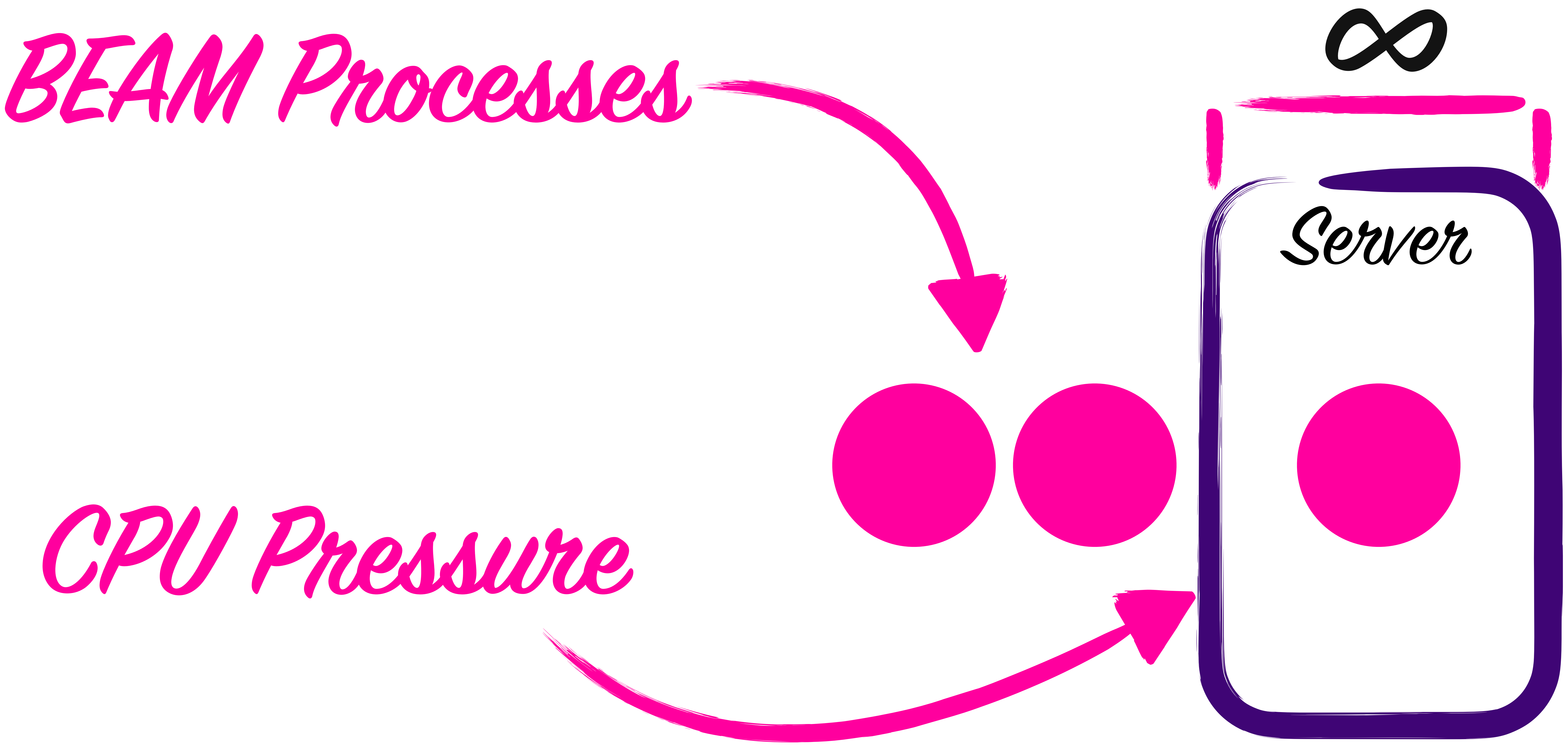
Little's Law

30 requests = 10 rps * 3000 ms



Little's Law

30 requests = 10 rps * ∞ ms



Little's Law

$$30 \text{ requests} = 10 \text{ rps} * \infty \text{ ms}$$

Little’s Law

$$\infty \text{ requests} = 10 \text{ rps} * \infty \text{ ms}$$

Little's Law

This is bad

∞ requests = 10 rps * ∞ ms

Lets Talk About…

Queues

Overload Mitigation

Adaptive Concurrency

Lets Talk About…

Queues

Overload Mitigation

Adaptive Concurrency

Overload

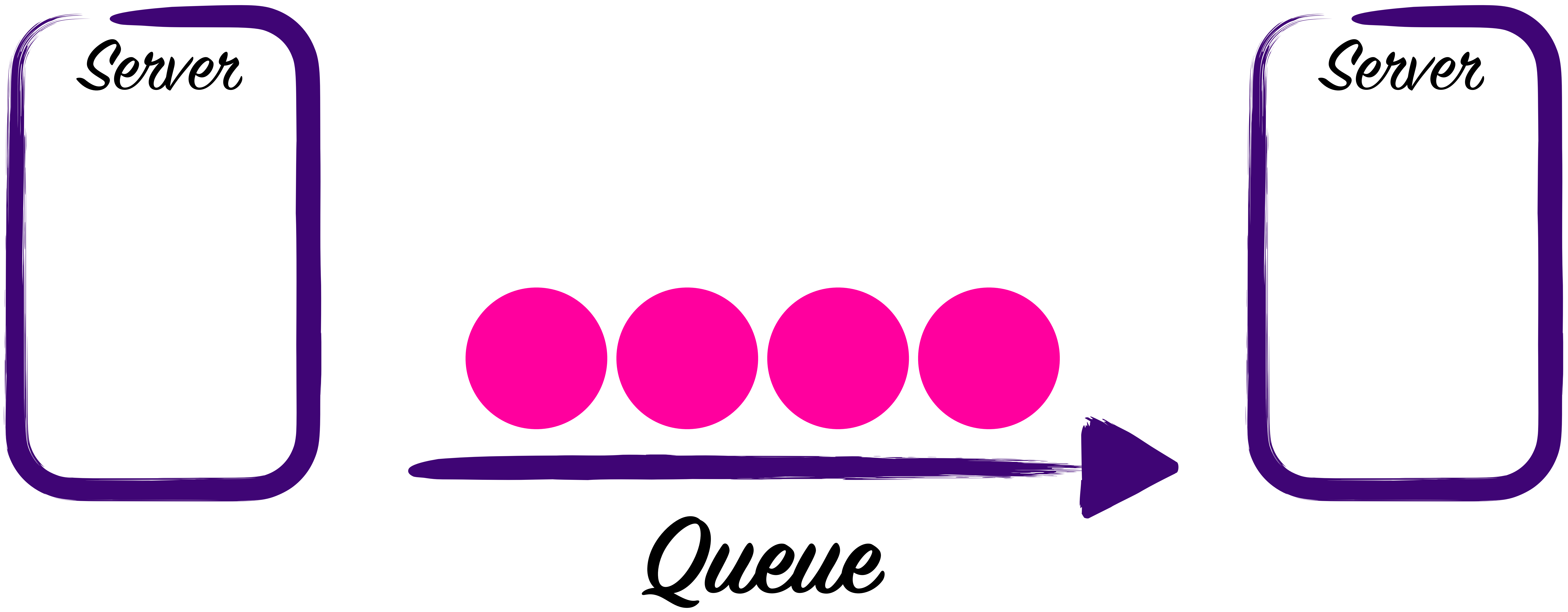
Arrival Rate > Processing Time

Overload

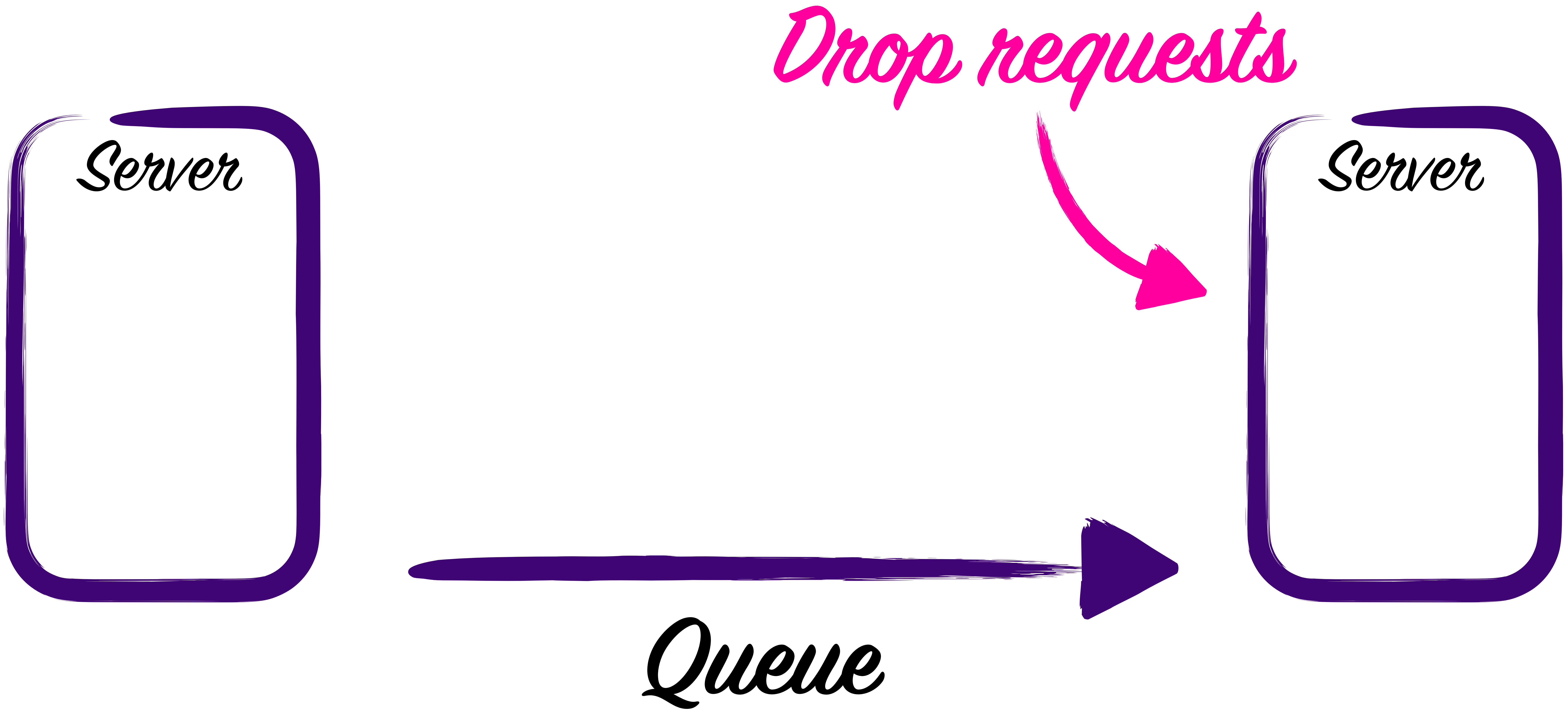
Arrival Rate > Processing Time

We need to get these under control

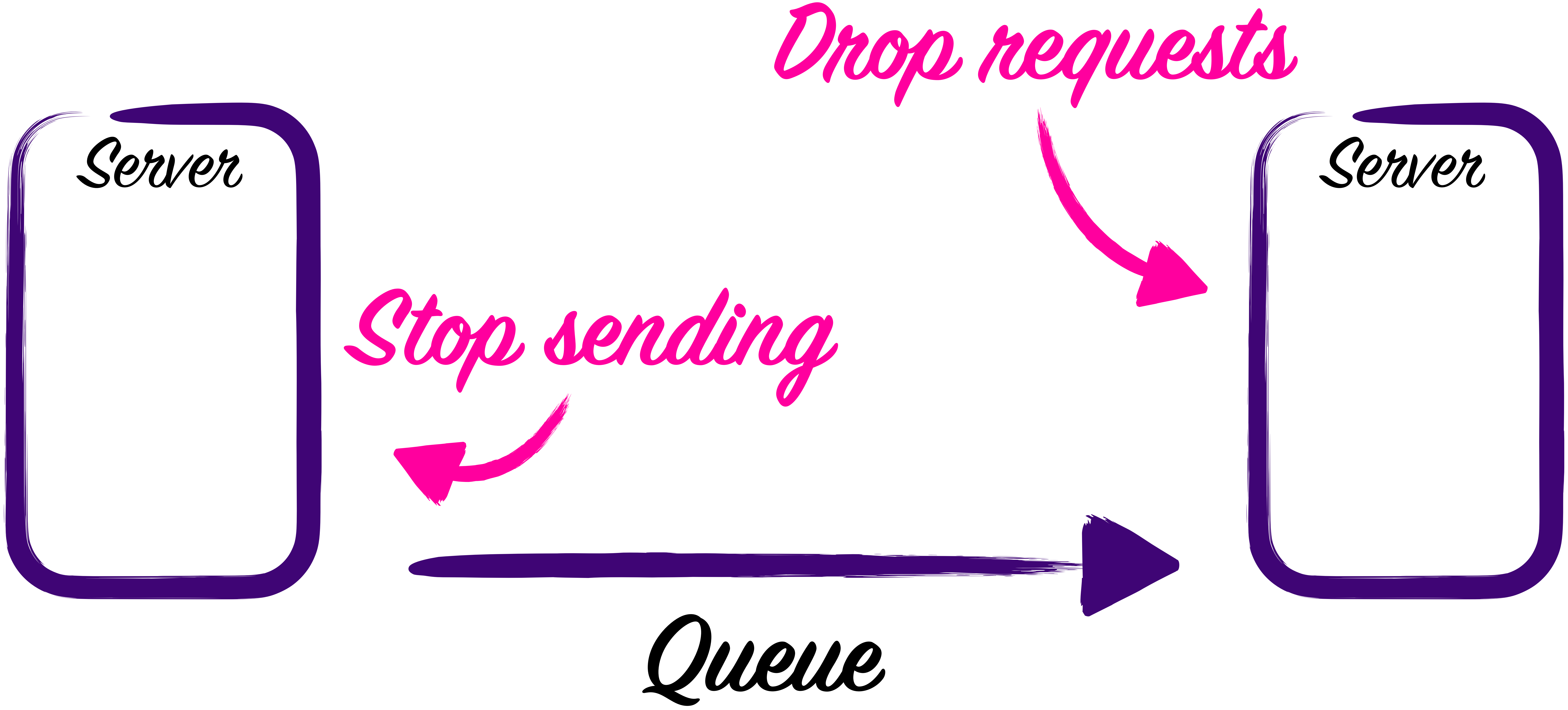
Load Shedding



Load Shedding



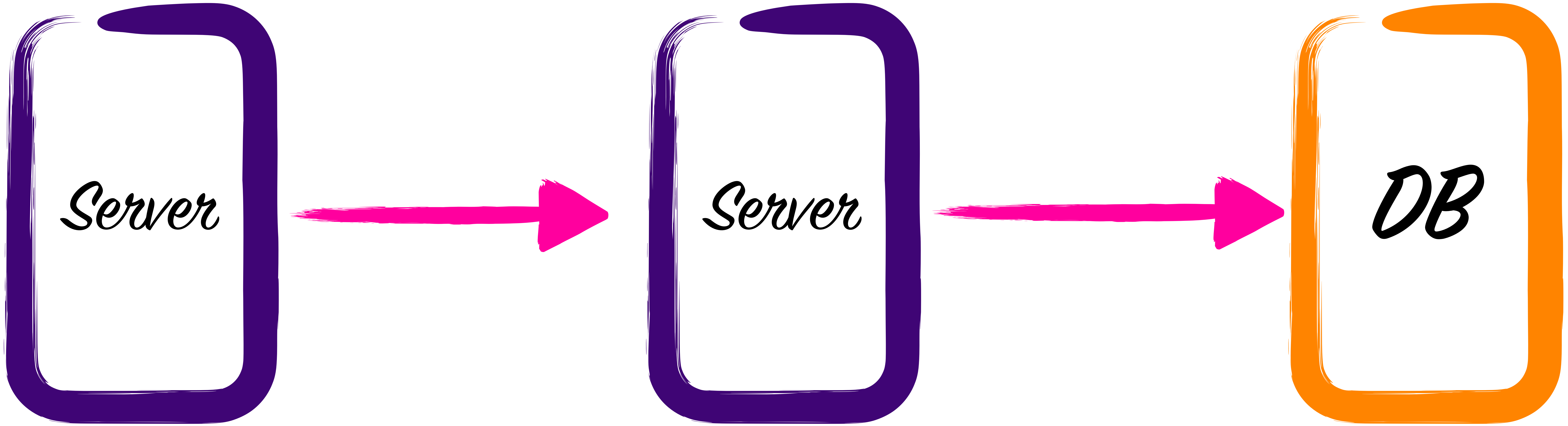
Load Shedding



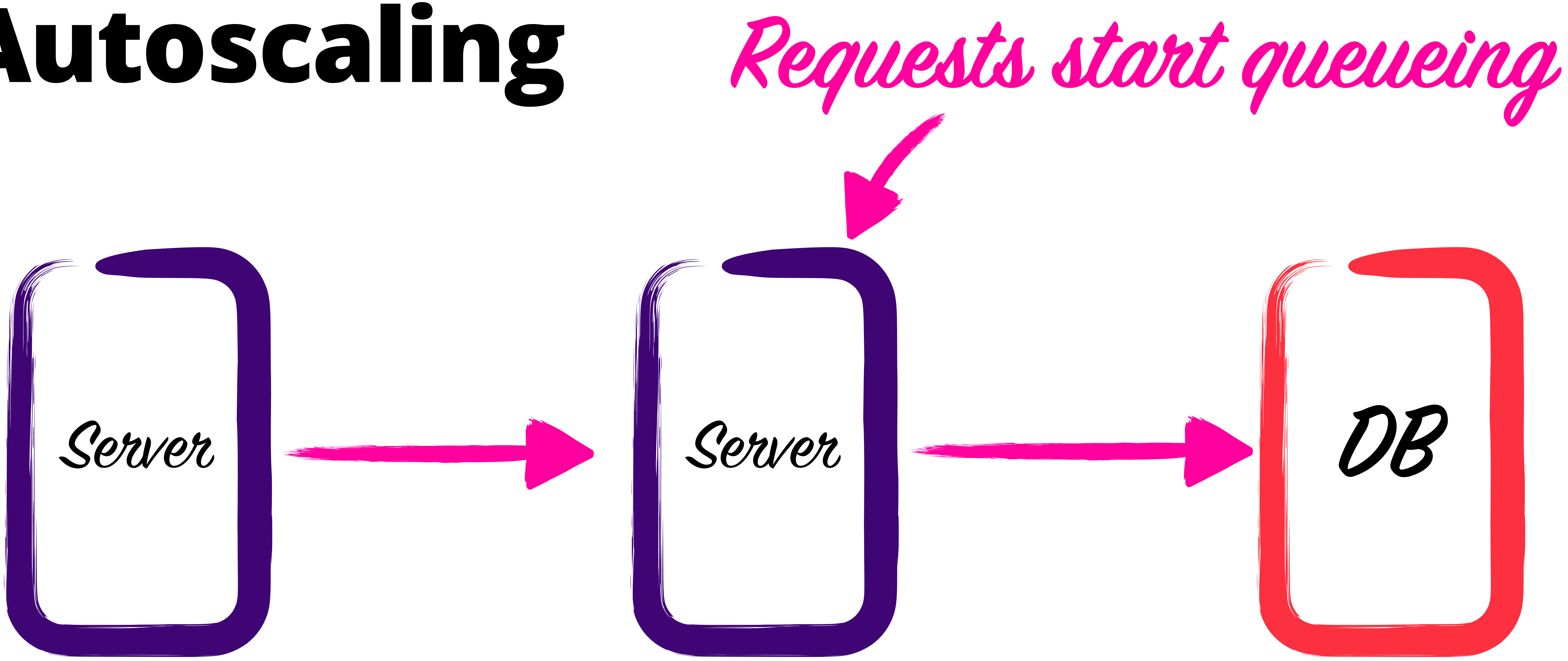
Autoscaling

AutoScaling

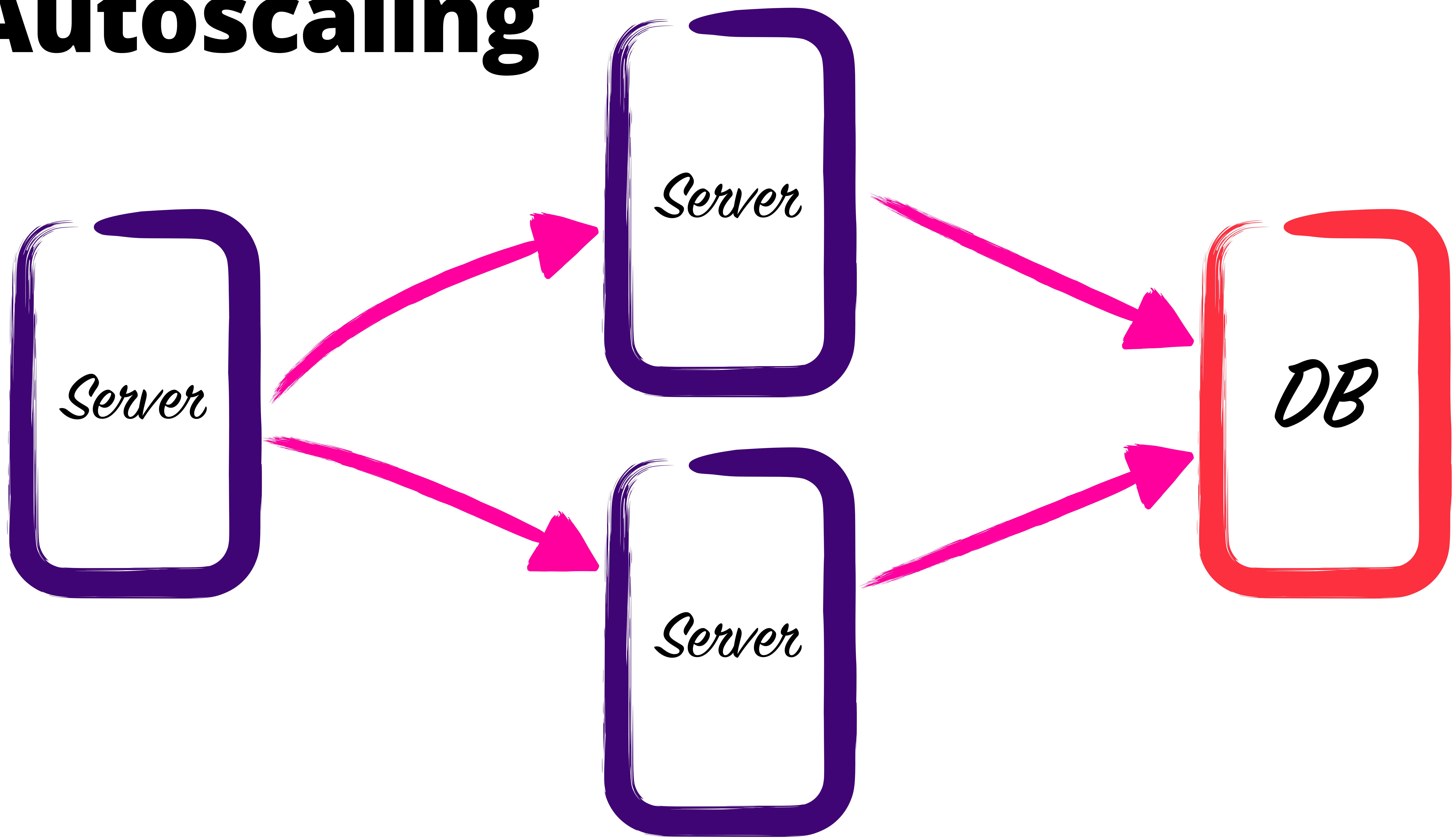
Autoscaling



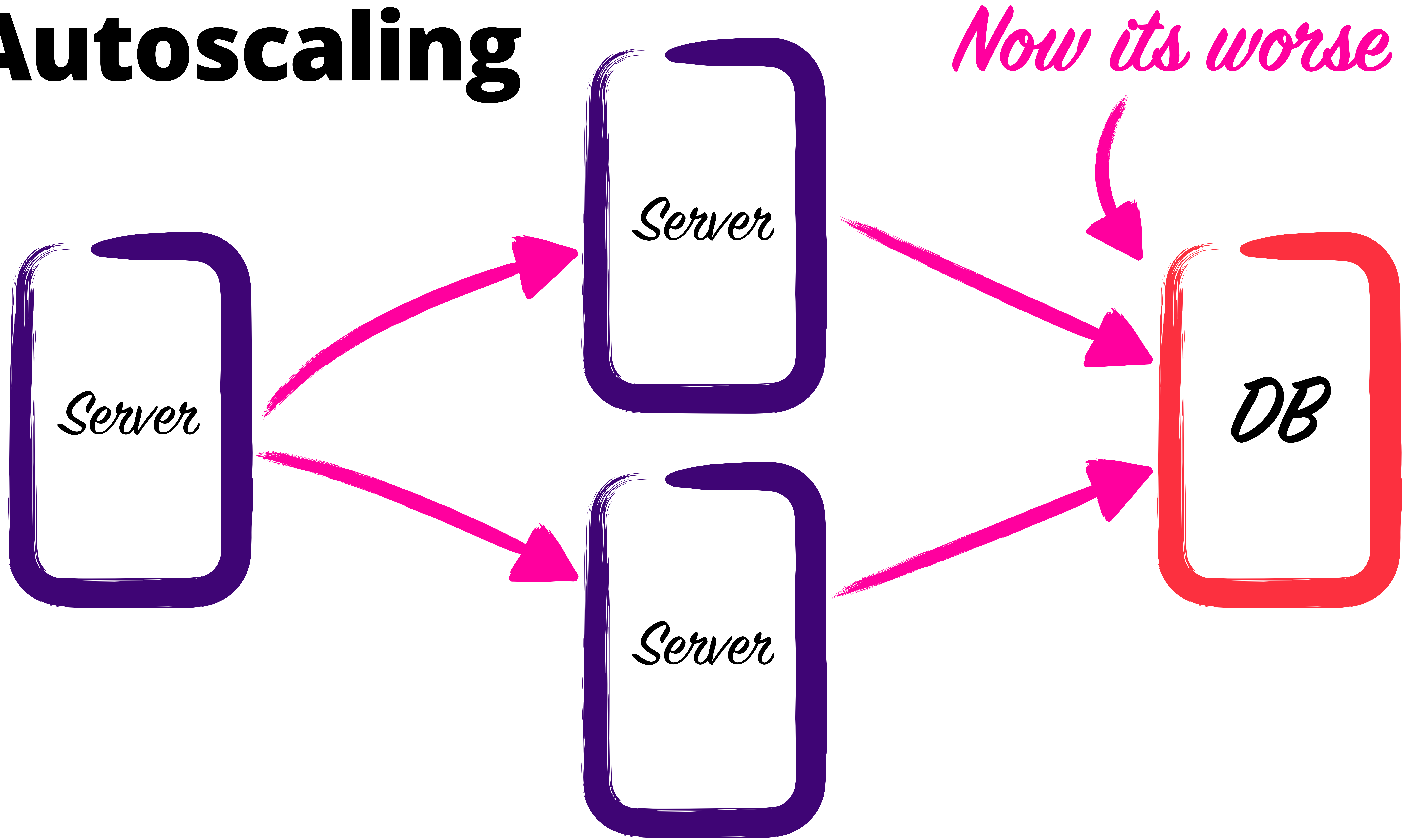
Autoscaling



Autoscaling



Autoscaling

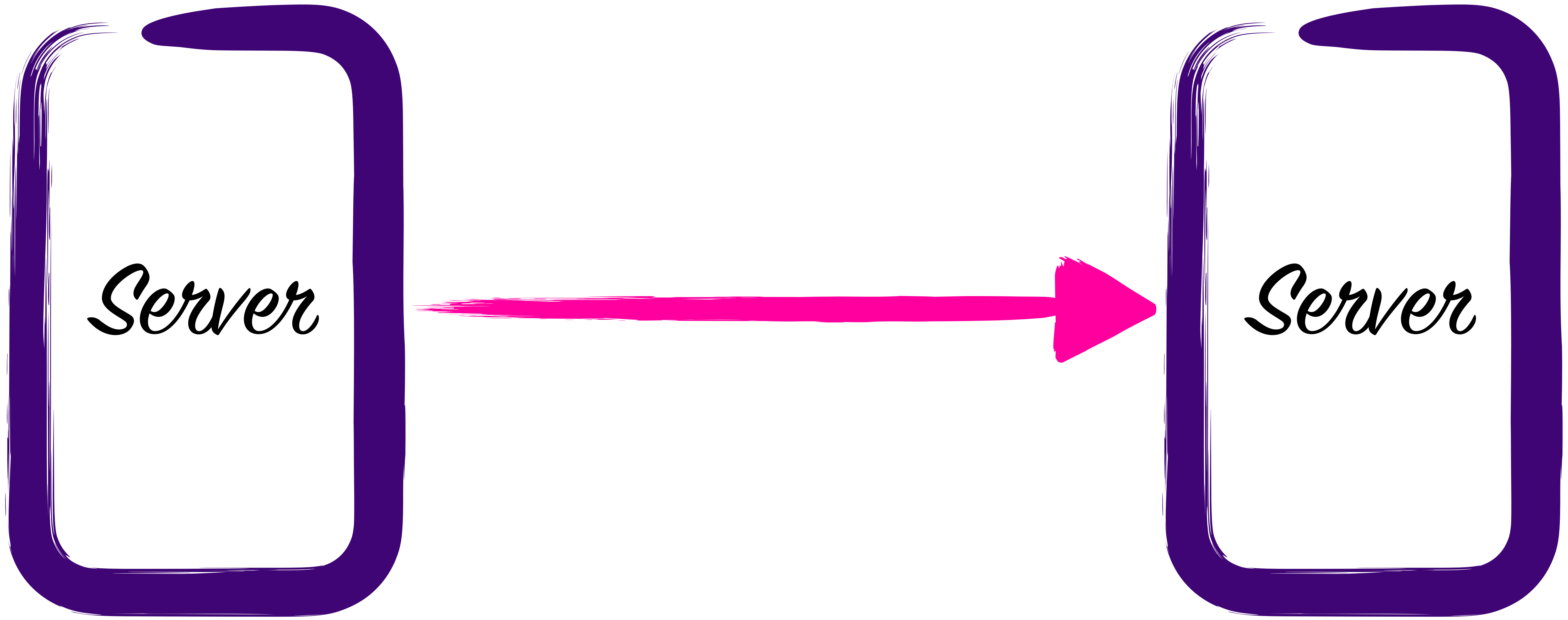


Autoscaling needs to be in response to load shedding

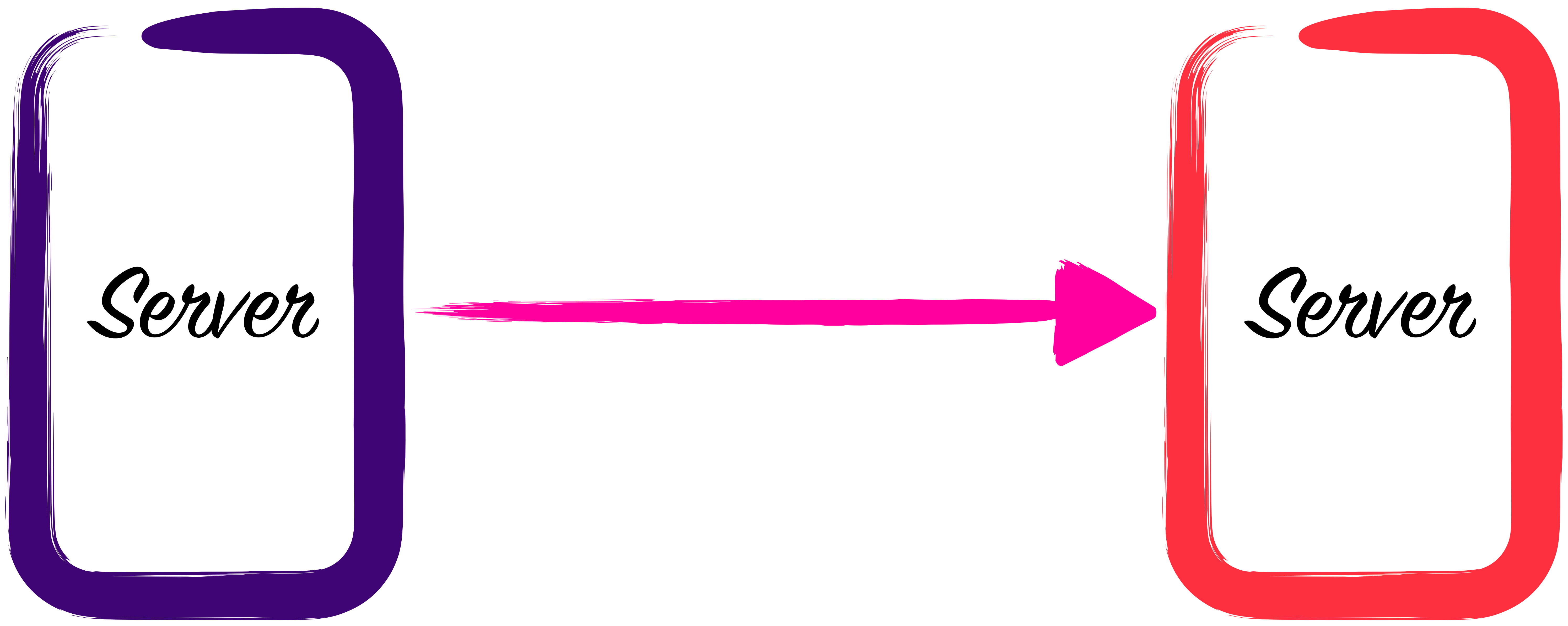
Circuit Breakers

Circuit Breakers

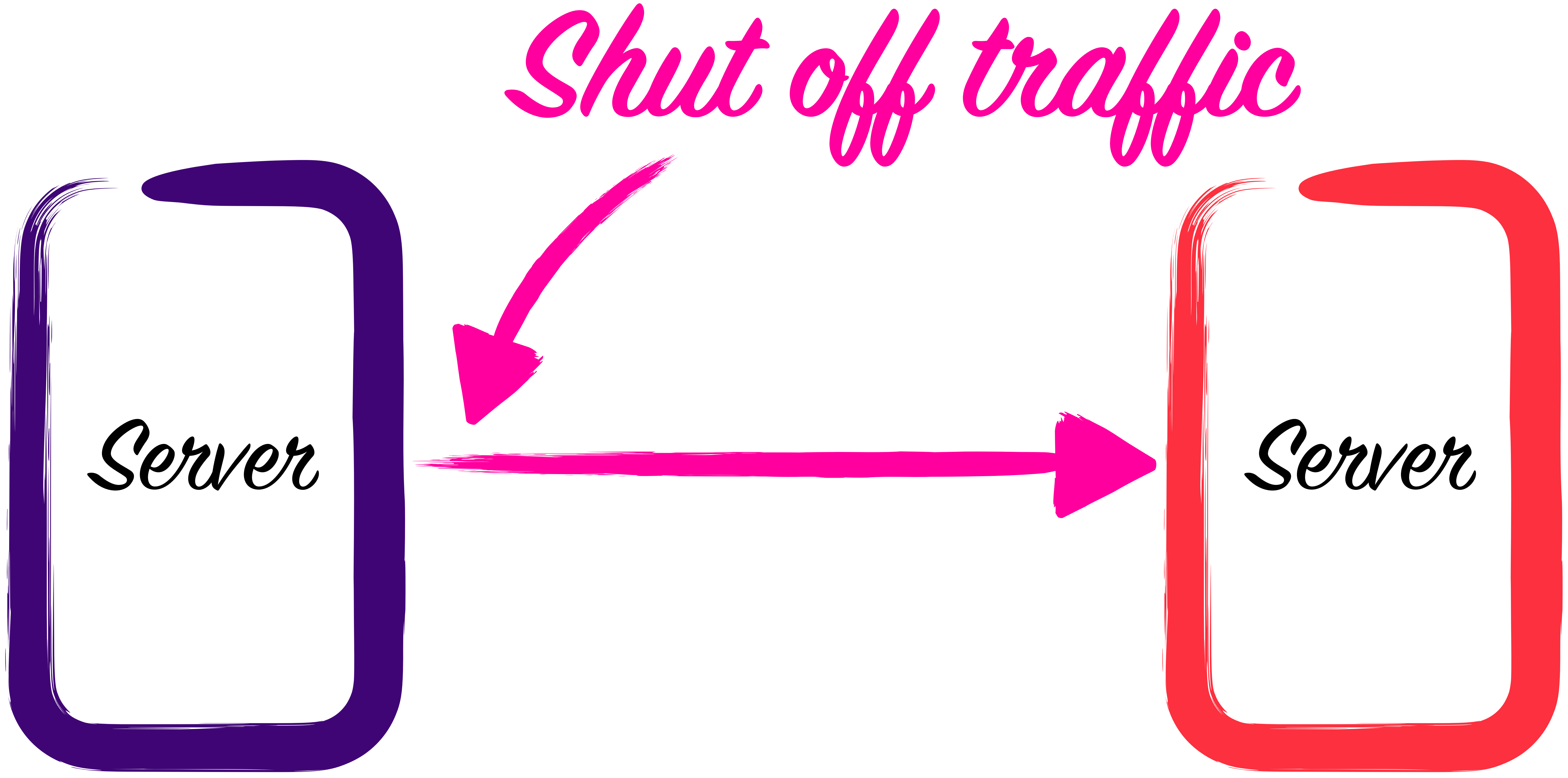
Circuit Breakers



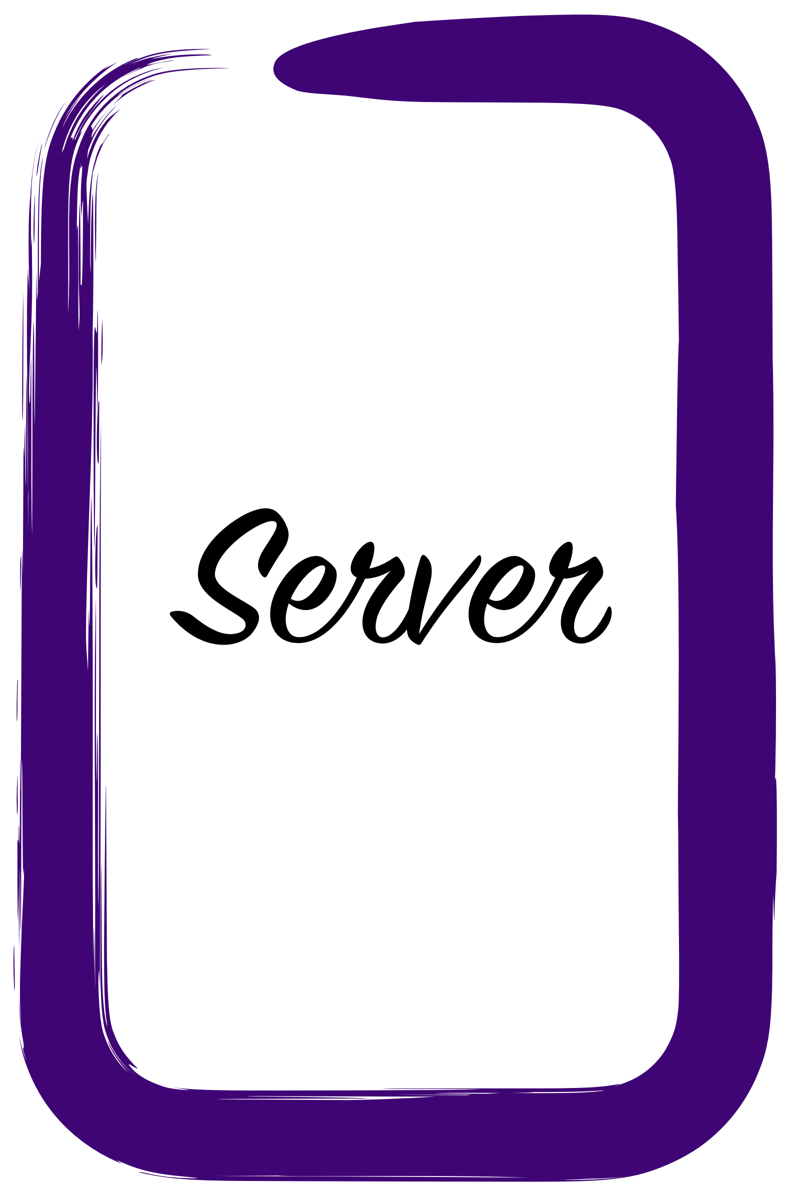
Circuit Breakers

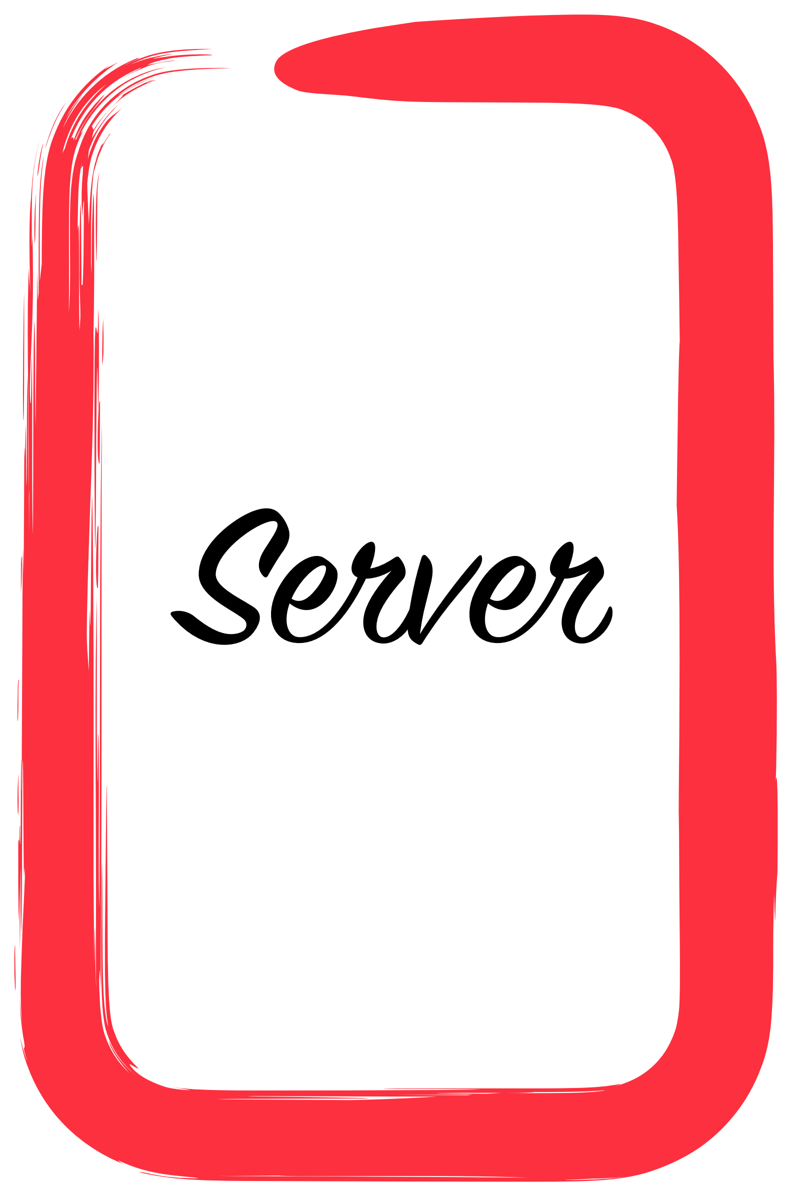


Circuit Breakers



Circuit Breakers





Circuit Breakers

Circuit Breakers are your last line of defense

Lets Talk About…

Queues

Overload Mitigation

Adaptive Concurrency

Lets Talk About…

Queues

Overload Mitigation

Adaptive Concurrency

We want to allow as many requests as we can actually handle

Congestion Avoidance and Control*

Van Jacobson†
Lawrence Berkeley Laboratory

Michael J. Karels‡
University of California at Berkeley

November, 1988

Introduction

Computer networks have experienced an explosive growth over the past few years and with that growth have come severe congestion problems. For example, it is now common to see internet gateways drop 10% of the incoming packets because of local buffer overflows. Our investigation of some of these problems has shown that much of the cause lies in transport protocol implementations (*not* in the protocols themselves): The 'obvious' ways to implement a window-based transport protocol can result in exactly the wrong behavior in response to network congestion. We give examples of 'wrong' behavior and describe some simple algorithms that can be used to make right things happen. The algorithms are rooted in the idea of achieving network stability by forcing the transport connection to obey a 'packet conservation' principle. We show how the algorithms derive from this principle and what effect they have on traffic over congested networks.

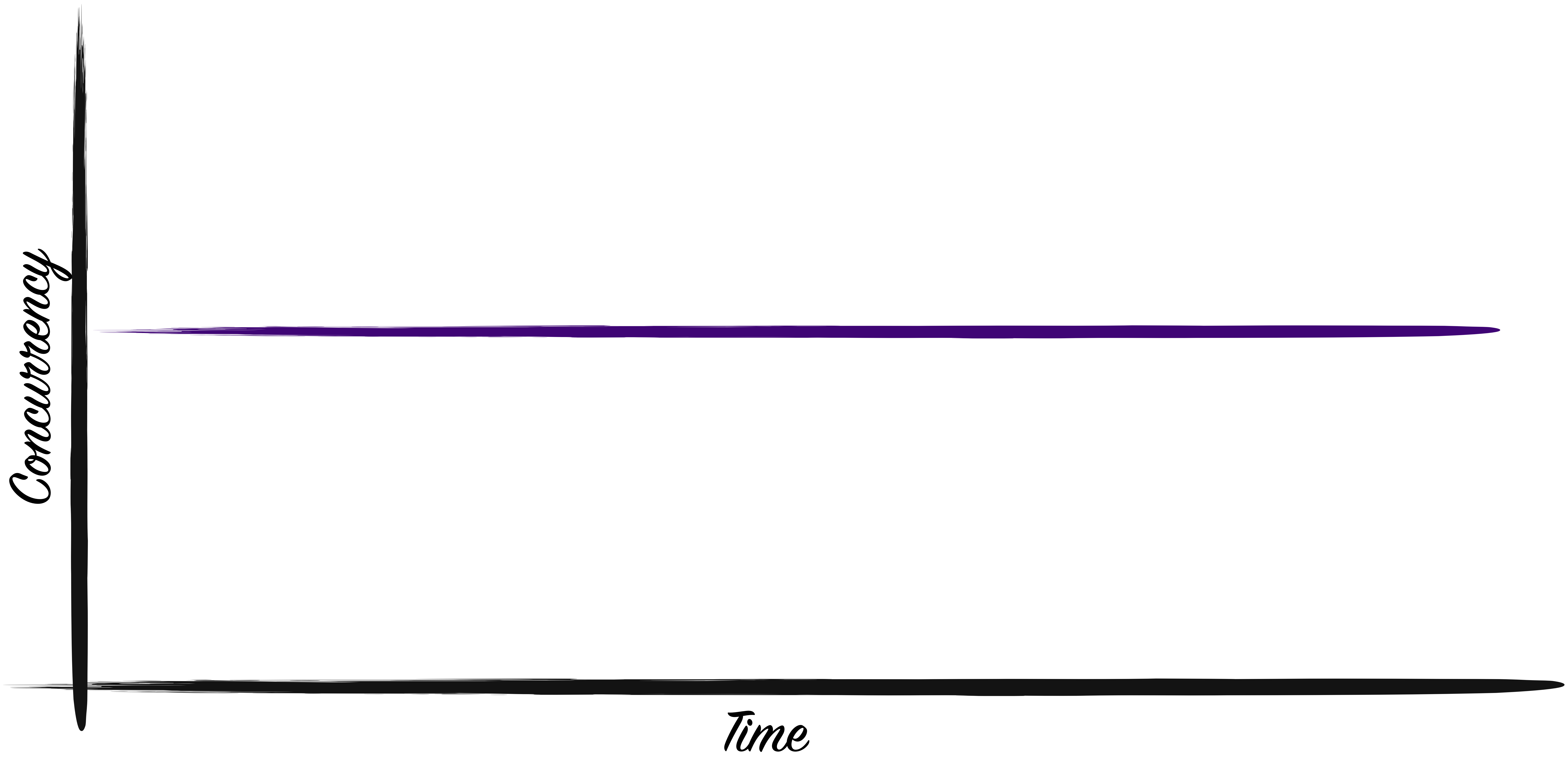
In October of '86, the Internet had the first of what became a series of 'congestion collapses'. During this period, the data throughput from LBL to UC Berkeley (sites separated by 400 yards and two IMP hops) dropped from 32 Kbps to 40 bps. We were fascinated by this sudden factor-of-thousand drop in bandwidth and embarked on an investigation of why things had gotten so bad. In particular, we wondered if the 4.3BSD (Berkeley UNIX) TCP was mis-behaving or if it could be tuned to work better under abysmal network conditions. The answer to both of these questions was "yes".

*__Note:__ This is a very slightly revised version of a paper originally presented at SIGCOMM '88 [12]. If you wish to reference this work, please cite [12].

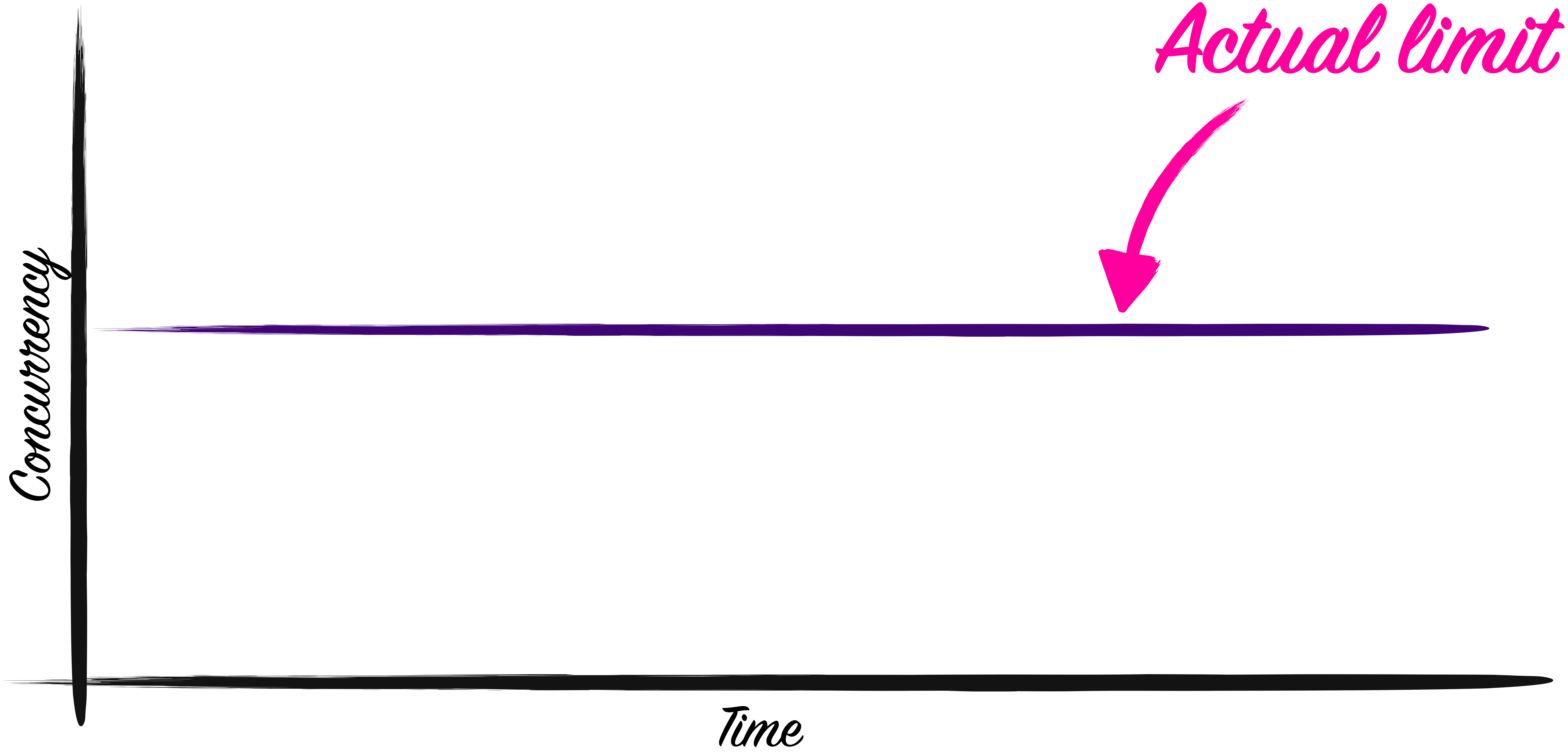
†This work was supported in part by the U.S. Department of Energy under Contract Number DE-AC03-76SF00098.

‡This work was supported by the U.S. Department of Commerce, National Bureau of Standards, under Grant Number 60NANB8D0830.

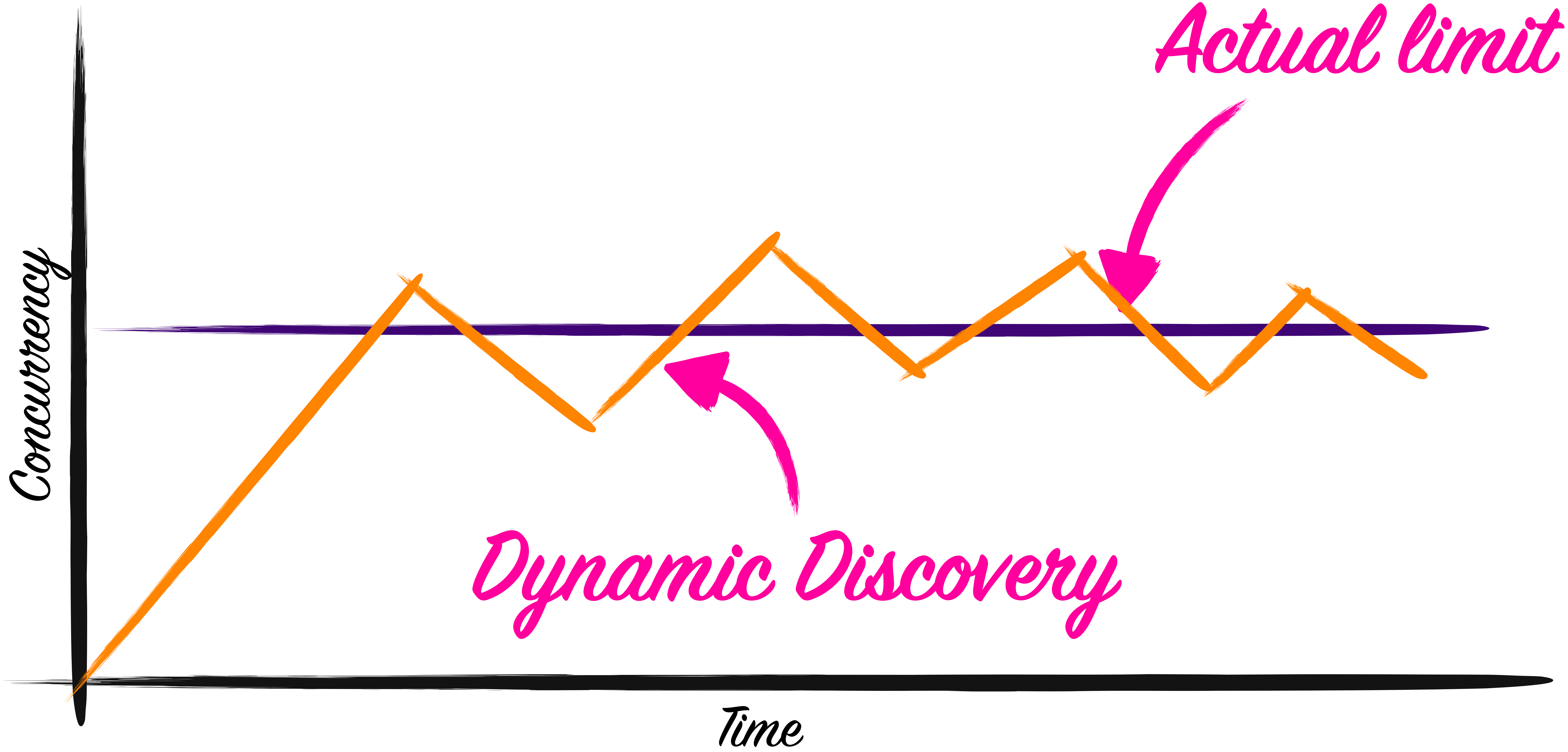
Adaptive Limits



Adaptive Limits



Adaptive Limits



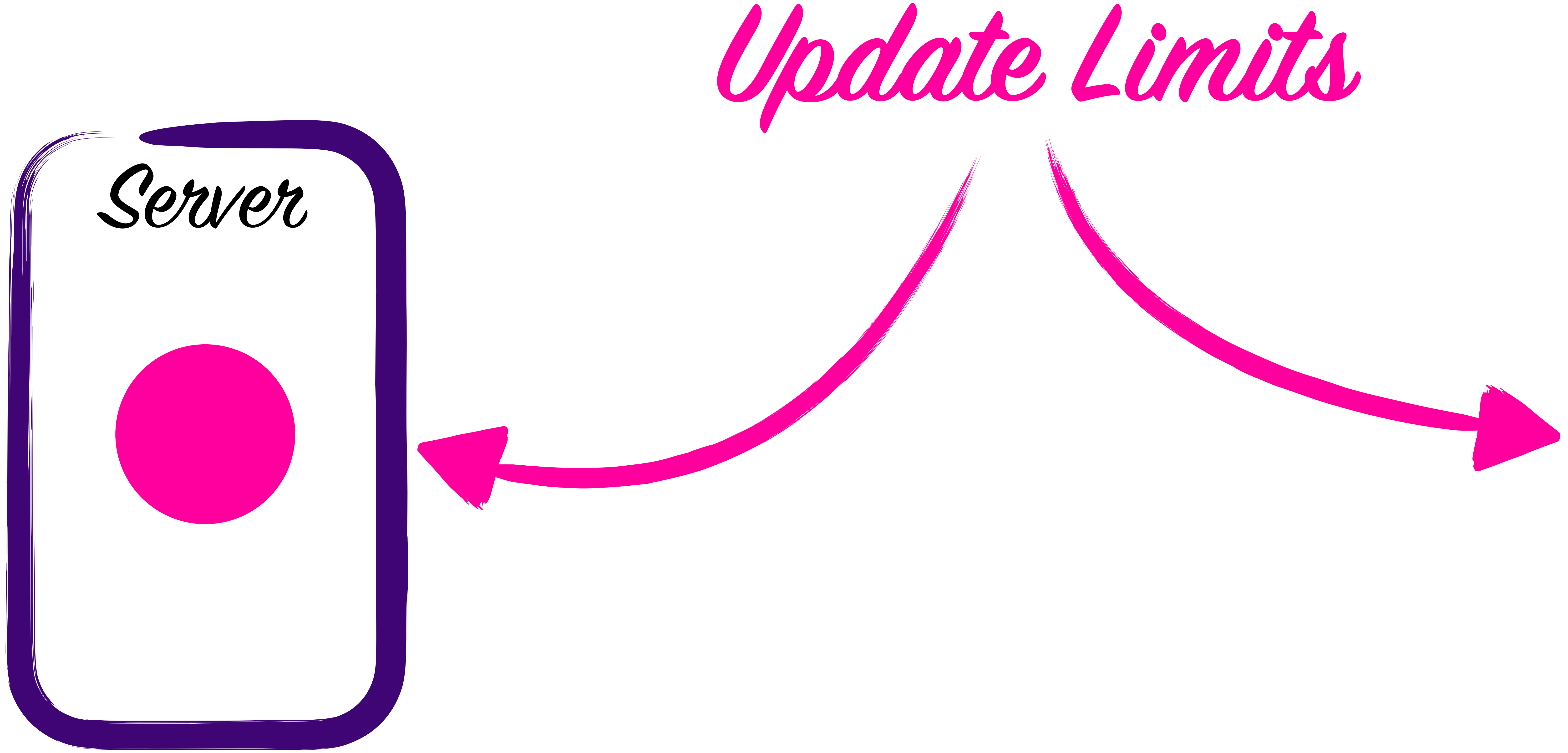
Load Shedding

Load Shedding

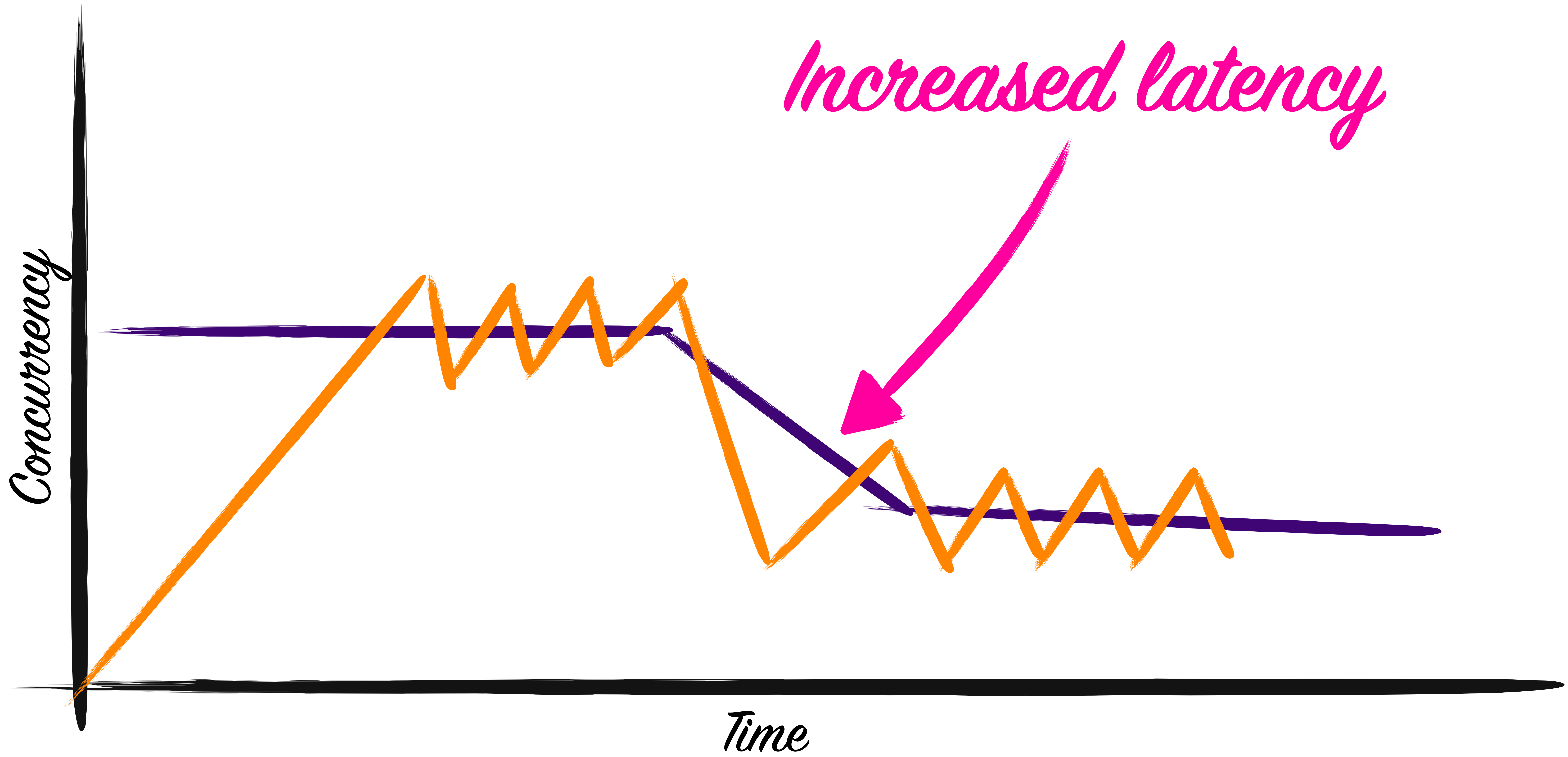
Load Shedding

Load Shedding

Load Shedding



Adaptive Limits



Signals for Adjusting Limits

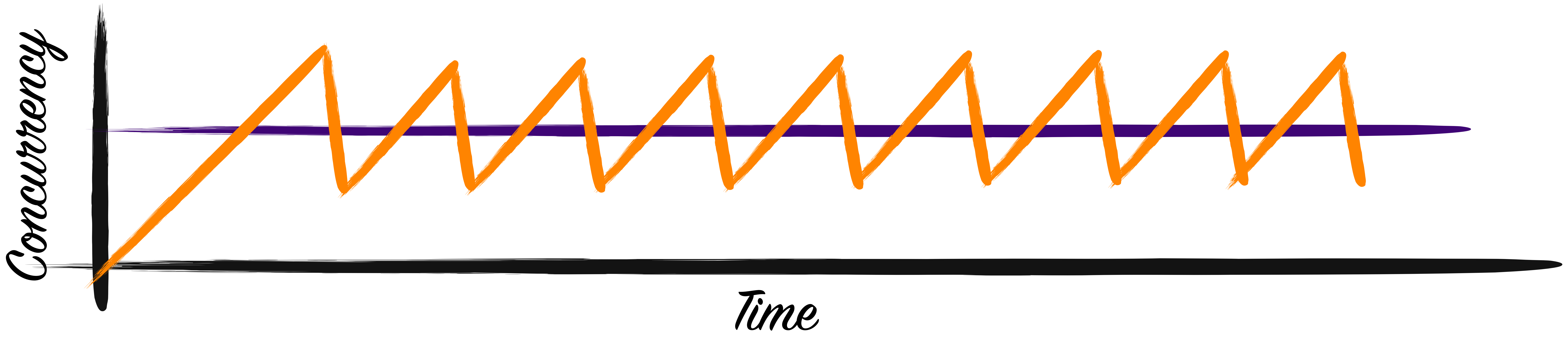
Latency

Successful vs. Failed requests

Additive Increase Multiplicative Decrease

Success state: limit + 1

Backoff state: limit * 0.95



Prior Art/Alternatives

https://github.com/ferd/pobox/

https://github.com/fishcakez/sbroker/

https://github.com/heroku/canal_lock

https://github.com/jlouis/safetyvalve

https://github.com/jlouis/fuse

Regulator

https://github.com/keathley/regulator

Regulator

```
Regulator.install(:service, [
  limit: {Regulator.Limit.AIMD, [timeout: 500]}
])

Regulator.ask(:service, fn ->
  {:ok, Finch.request(:get, "https://keathley.io")}
end)
```

Conclusion

Queues are everywhere

Those queues need to be bounded to avoid overload

If your system is dynamic, your solution will also need to be dynamic

Go and build awesome stuff

Thanks

Chris Keathley / @ChrisKeathley / c@keathley.io