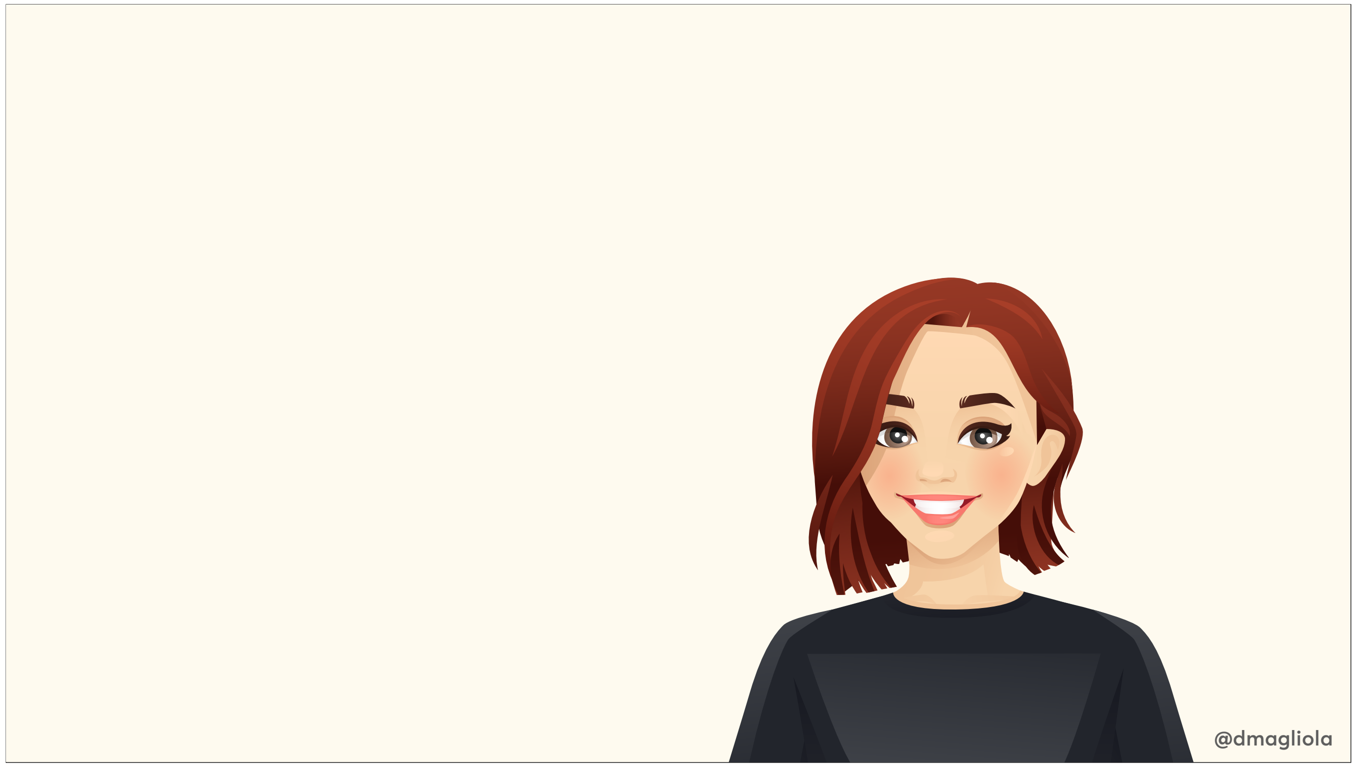
The secret to happy queues

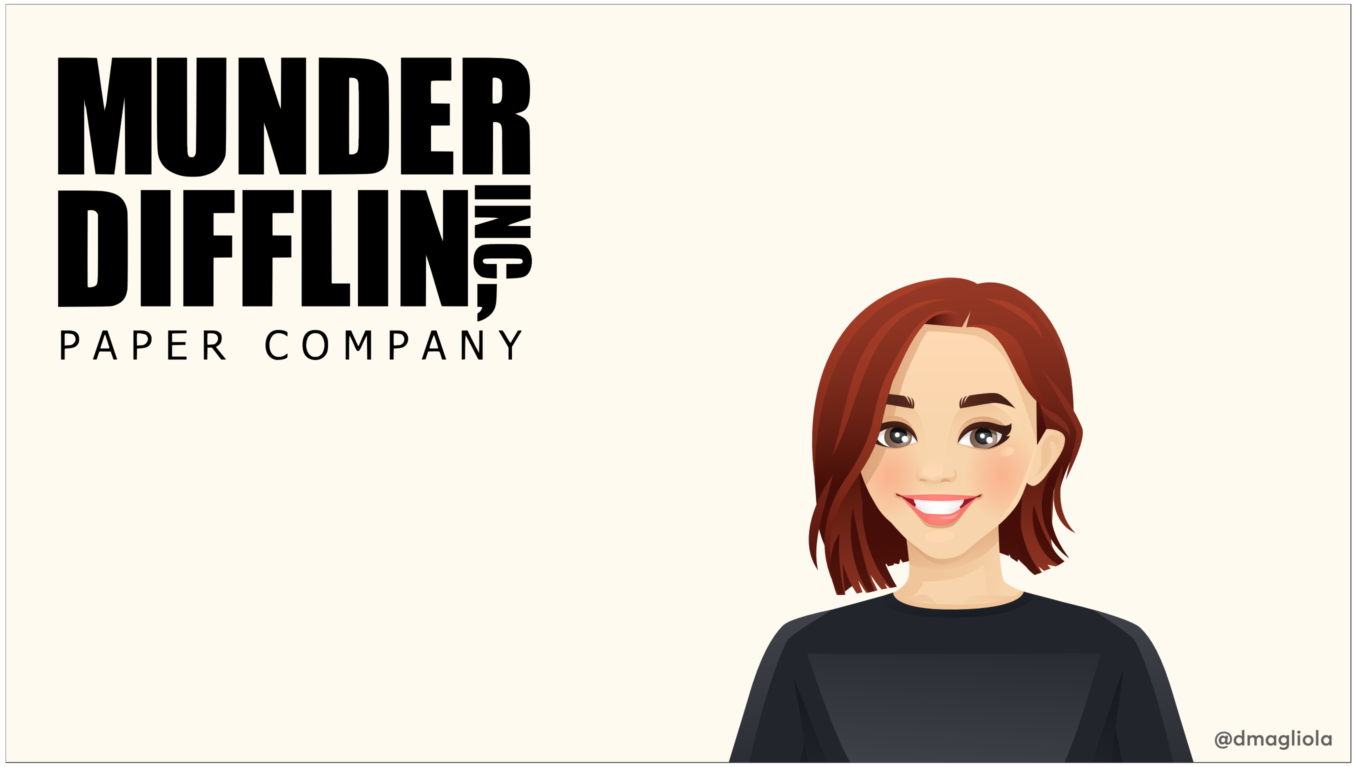
Daniel Magliola @dmagliola indeedflex™

RUBYCONF



@dmagliola

I want you to meet Sally. Sally is a lead engineer at Munder Difflin,



MUNDER DIFFLIN, INC.

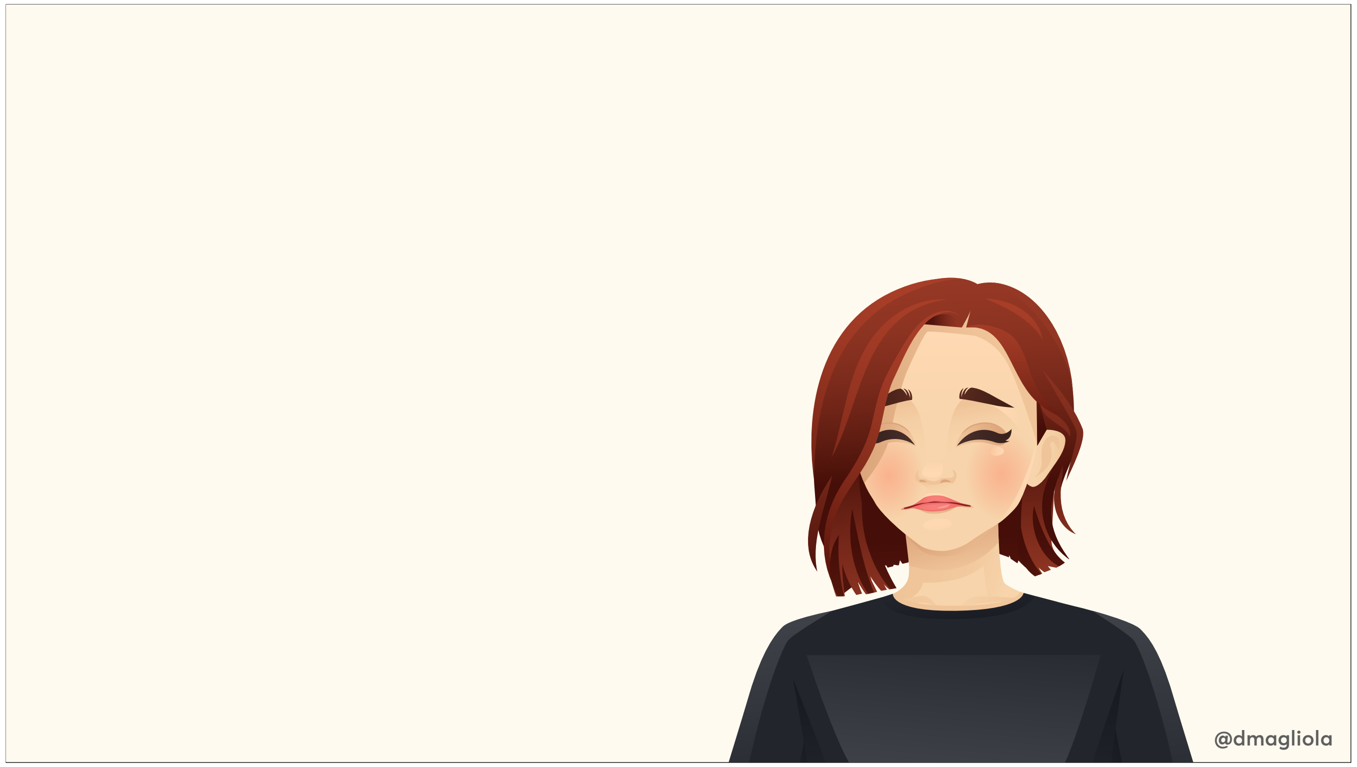
PAPER COMPANY

@dmagliola

a leading supplier of paper and other stationery products.

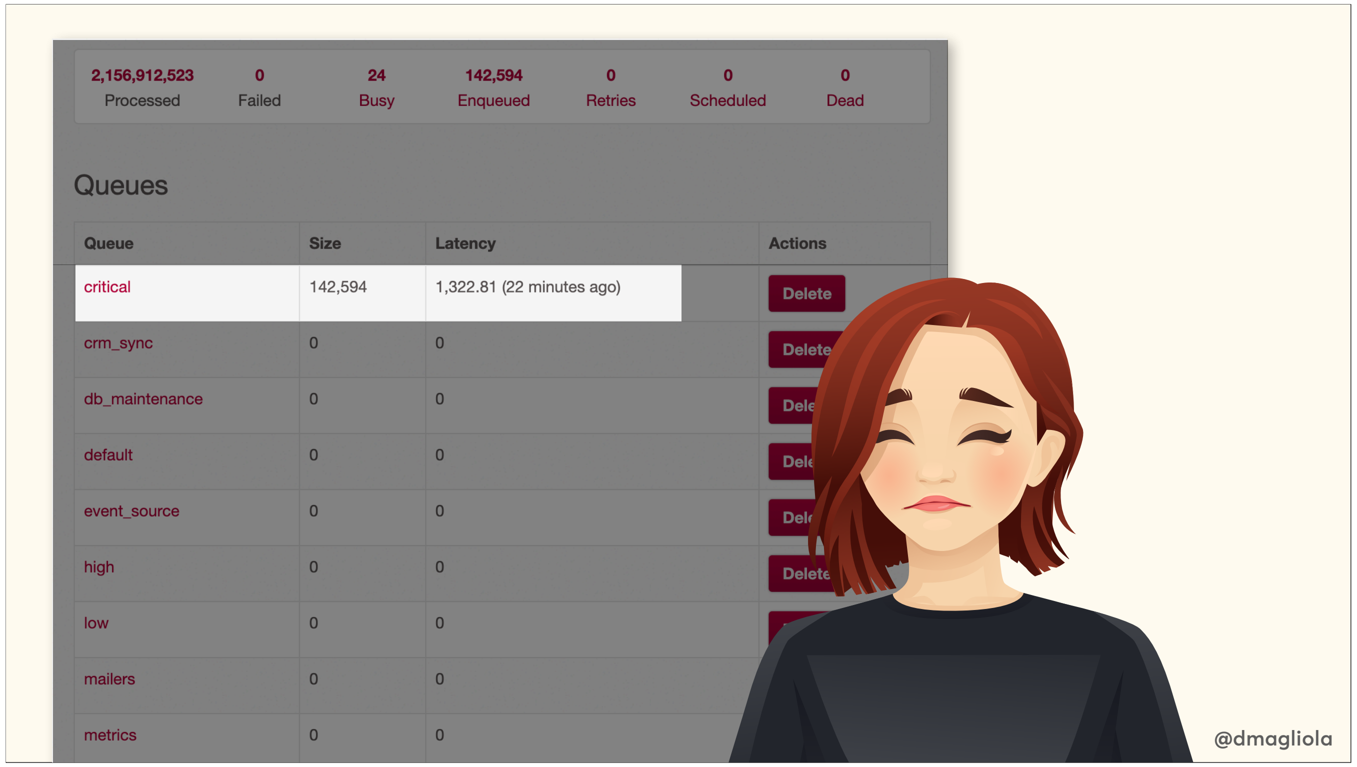
She’s been there since the very beginning, one of the first engineers they hired. Because of this, she knows the codebase inside and out, and she’s extremely familiar with the issues of running their app in production.

And Sally…



Is sad today.
Again.

And she is sad today,



2,156,912,523	0	24	142,594	0	0	0
Processed	Failed	Busy	Enqueued	Retries	Scheduled	Dead

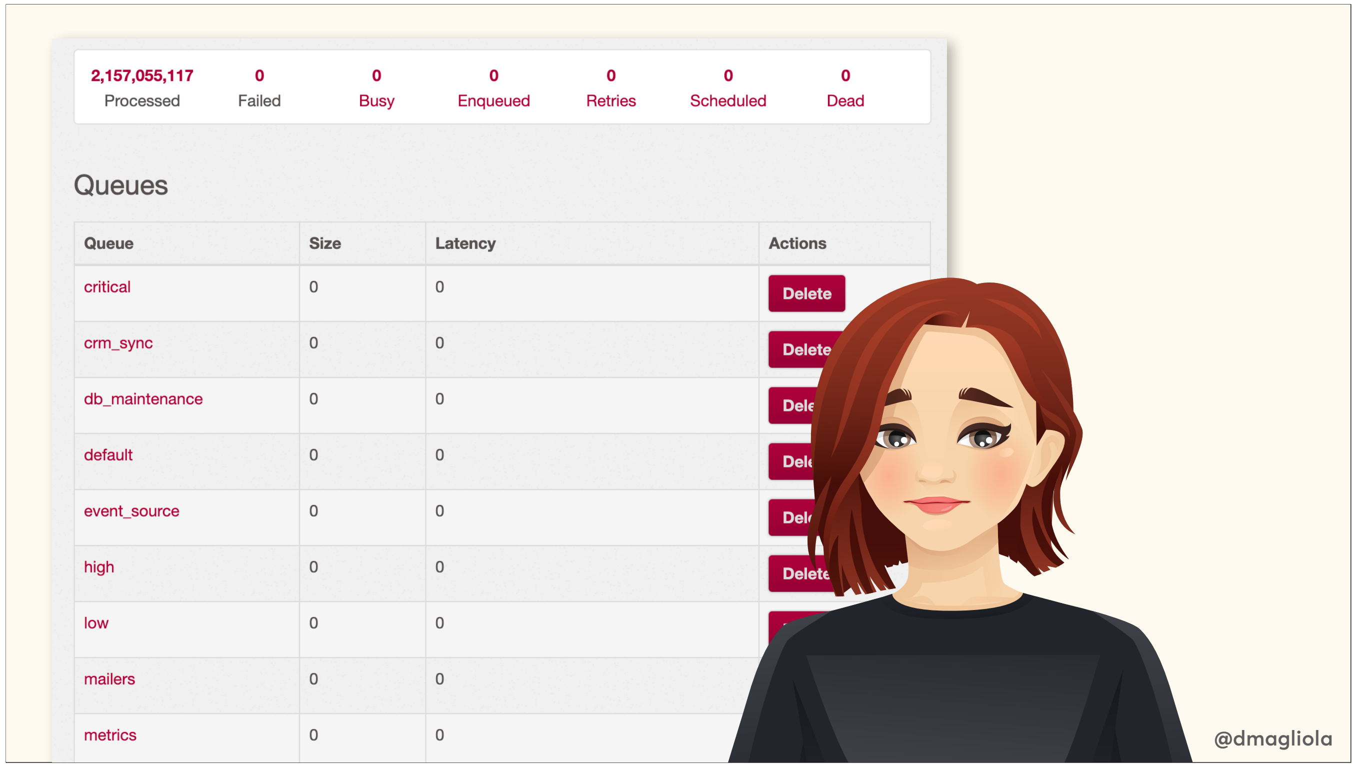
Queues

Queue	Size	Latency	Actions
critical	142,594	1,322.81 (22 minutes ago)	Delete
crm_sync	0	0	Delete
db_maintenance	0	0	Dele
default	0	0	Dele
event_source	0	0	Dele
high	0	0	Delete
low	0	0	
mailers	0	0	
metrics	0	0	

@dmagliola

because their queues are sad today.
Again.

So she deals with the problem, again,



2,157,055,117	0	0	0	0	0	0
Processed	Failed	Busy	Enqueued	Retries	Scheduled	Dead

Queues

Queue	Size	Latency	Actions
critical	0	0	Delete
crm_sync	0	0	Delete
db_maintenance	0	0	Dele
default	0	0	Dele
event_source	0	0	Del
high	0	0	Delete
low	0	0	
mailers	0	0	
metrics	0	0	

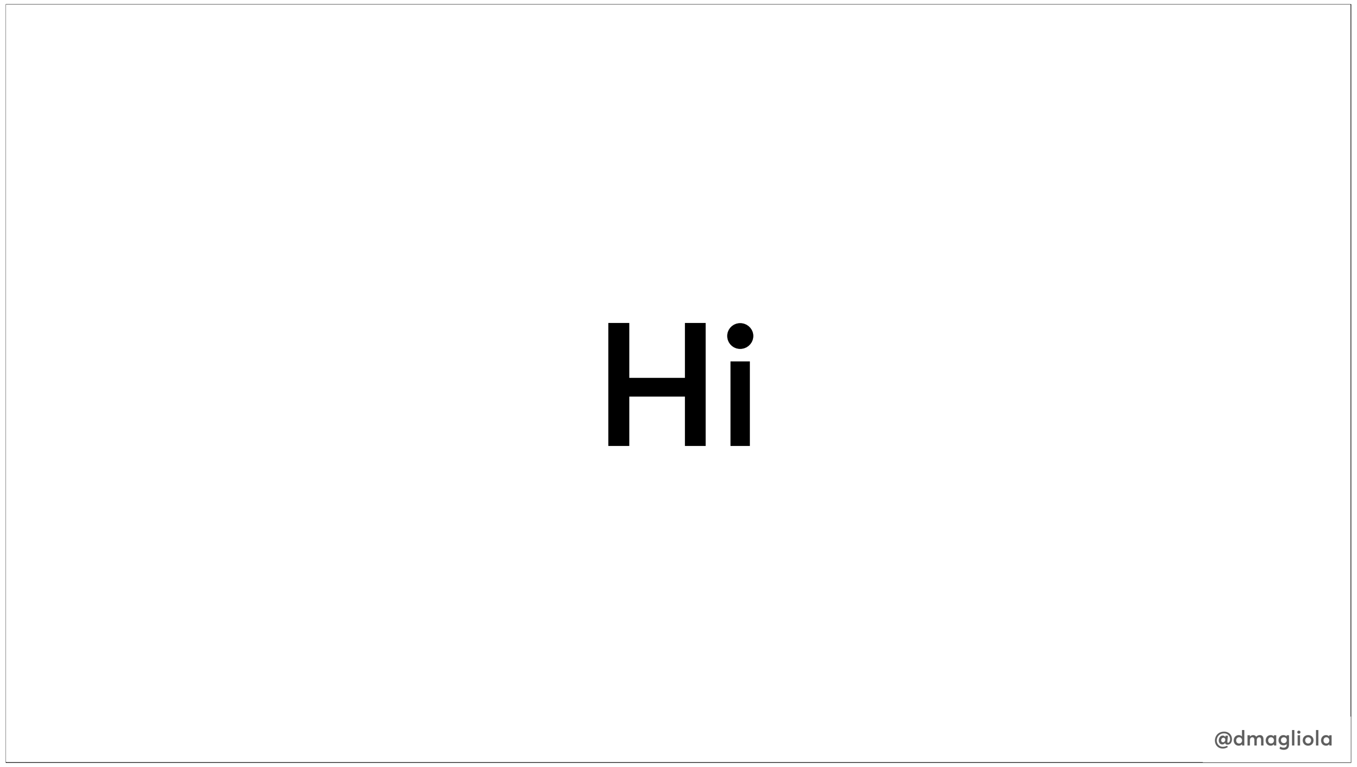
makes the system happy, then goes back to her normal work…

She’s still thinking about it, though. This has been a problem for years, and no matter what they’ve tried, they never seem to be able to fix it.

The next morning, after a good night’s sleep,

Sally wakes up with the solution. A radical new idea that will fix this problem once and for all.

But to understand how that will fix their problems, we're going to need a bit of history, and to understand how Munder Difflin got here in the first place.



Hi

@dmagliola

First of all though, hi.

My name is Daniel, and

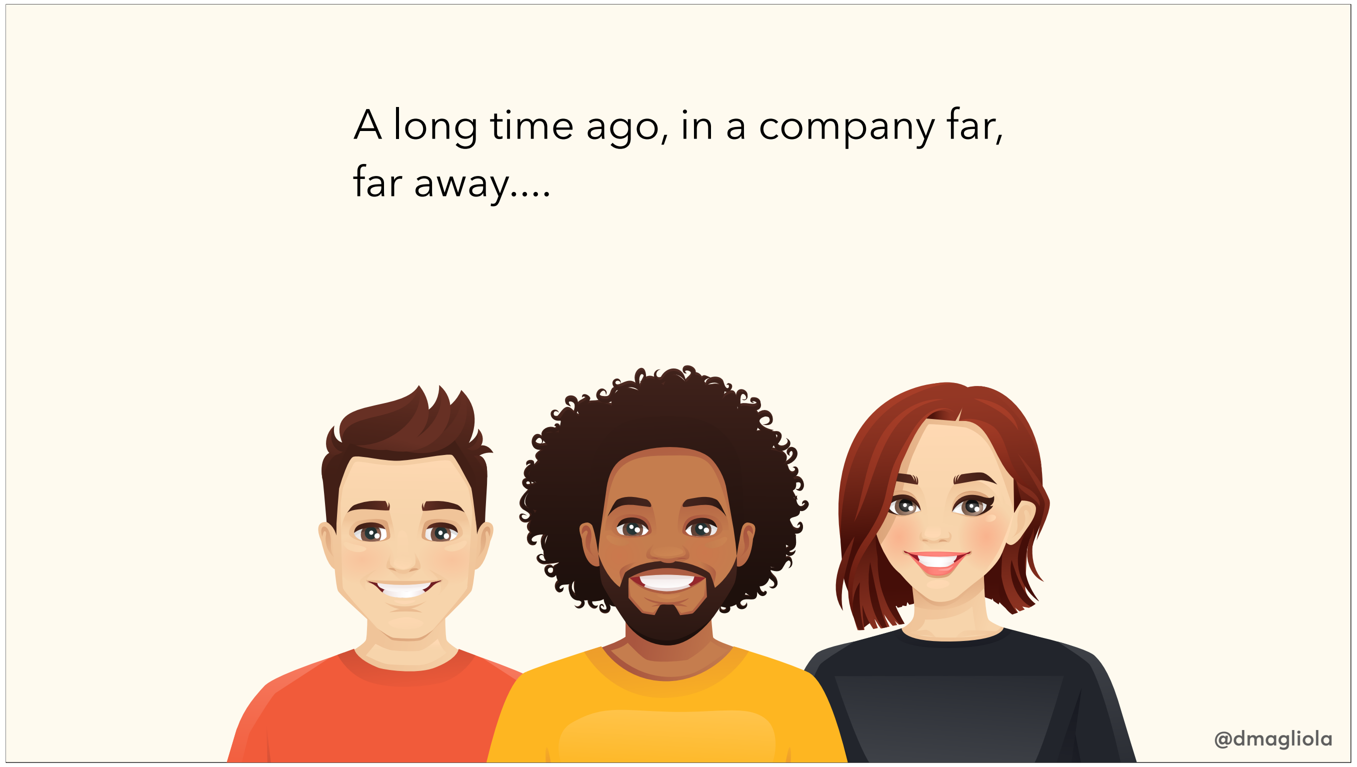
indeed flex

@dmagliola

I work at IndeedFlex, we are a flexible staffing platform based in London.

As you probably noticed, though, i’m not originally from London, I come from Argentina, so in case you were wondering, that’s the accent.

And now, let’s go back to our little paper company.

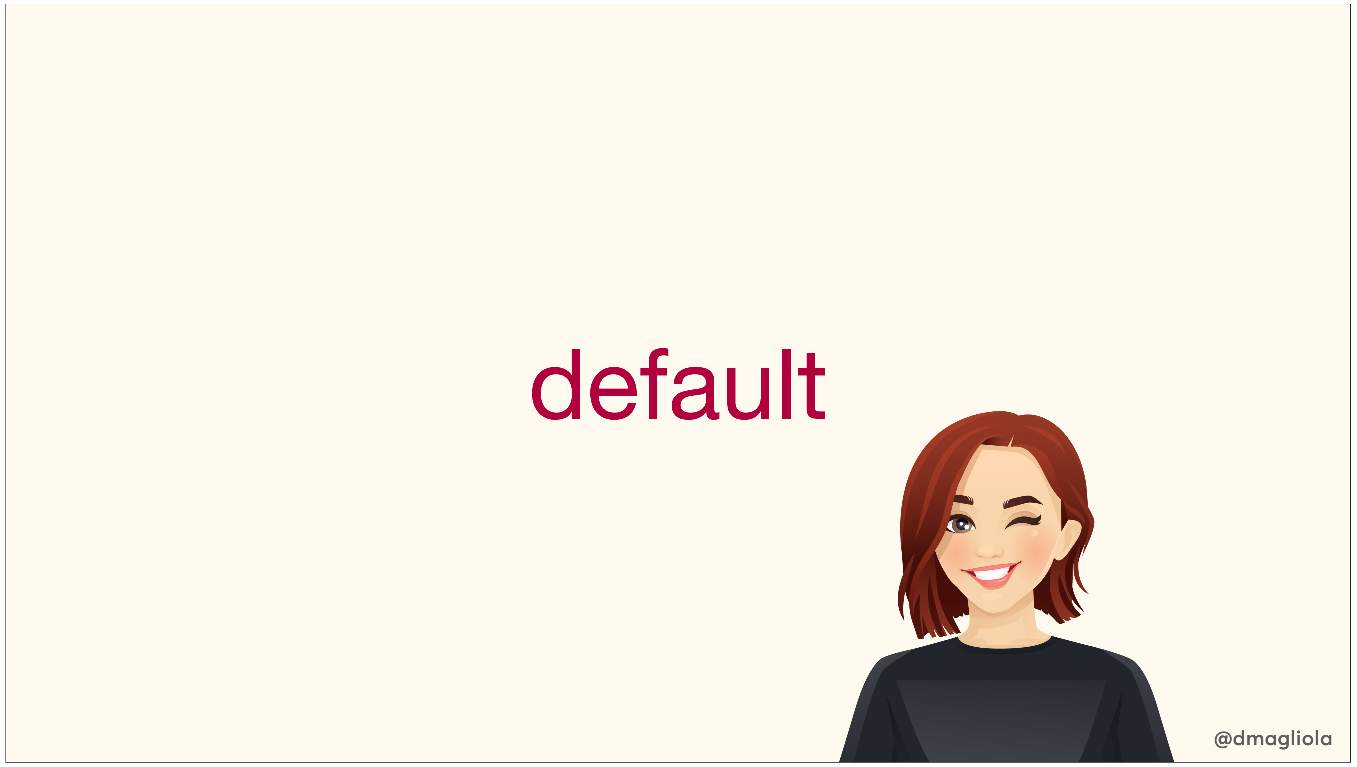


A long time ago, in a company far, far away....

@dmagliola

You see, when Sally joined Munder Difflin, they were a tiny team with only 3 developers; and they wrote the app that was running the entire company. Buying paper, keeping track of their inventory, selling it, delivering it, everything.

And at first everything was just running inside their one little web server and that was fine for a while, but they started having some trouble with that, so Sally decided to add a background job system.



default

@dmagliola

And then, there was a queue.

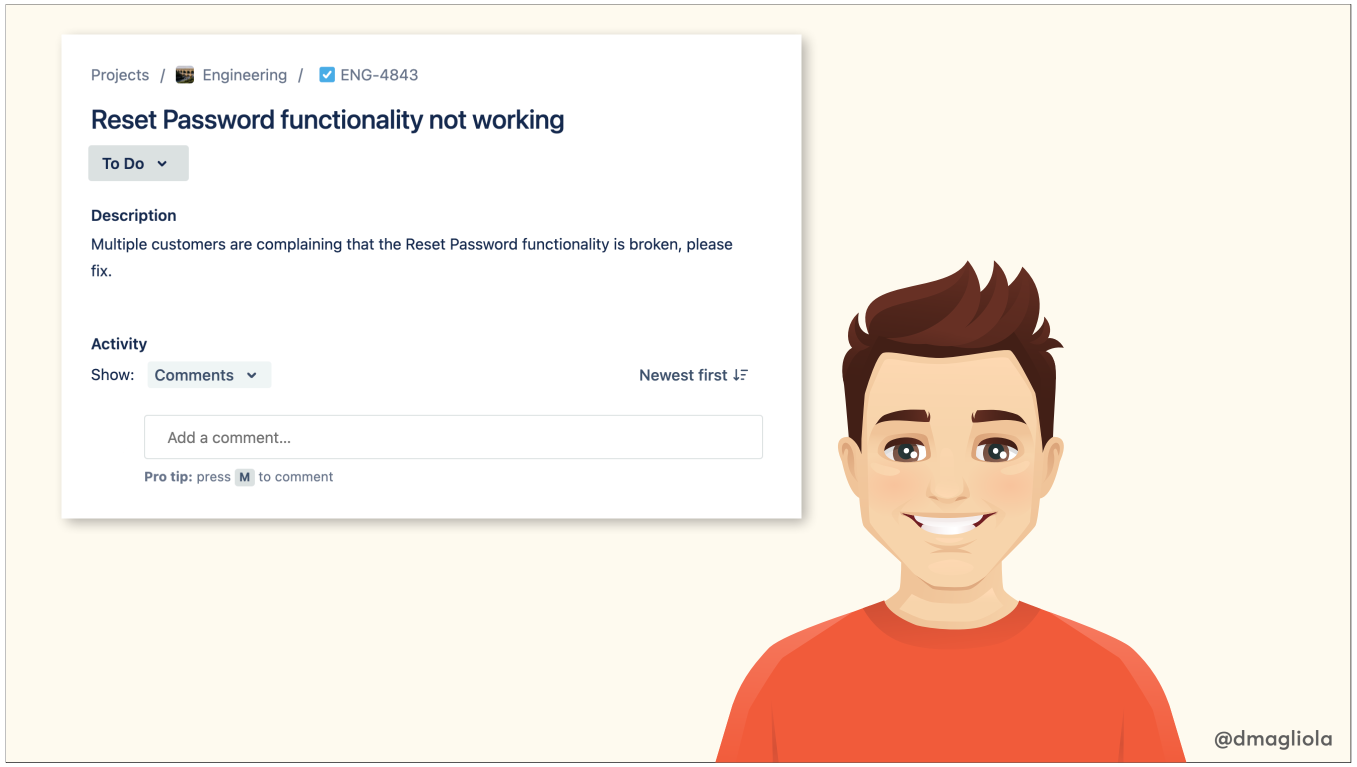
And the queue was good, it solved a lot of problems for the team, so they started adding more… and more jobs to the queue.

And the queue… was still good. So more and more jobs got added to it.

ACT 1

Paved with good intentions

And so our story begins…



Projects / Engineering / ☑ ENG-4843

Reset Password functionality not working

Description

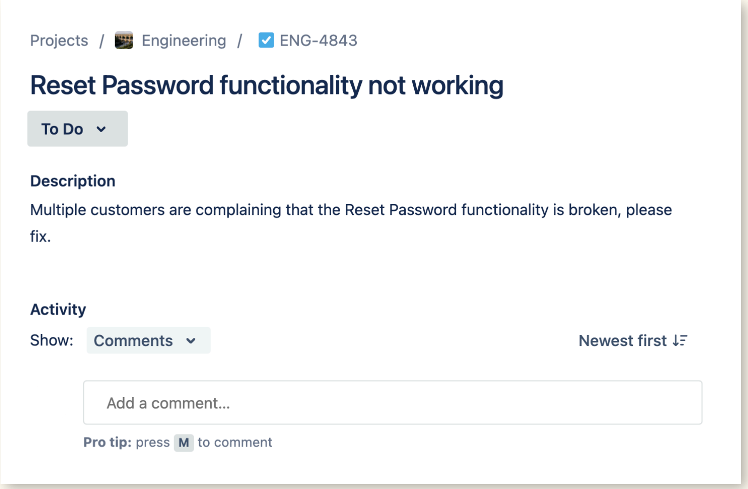
Multiple customers are complaining that the Reset Password functionality is broken, please fix.

Activity

@dmagliola

Now a few months later, Joey receives a Support ticket: users are reporting the Reset Password functionality is broken.

Joey takes the ticket, [loud clacking on keyboard],



Projects / Engineering / ENG-4843

Reset Password functionality not working

To Do

Description

Multiple customers are complaining that the Reset Password functionality is broken, please fix.

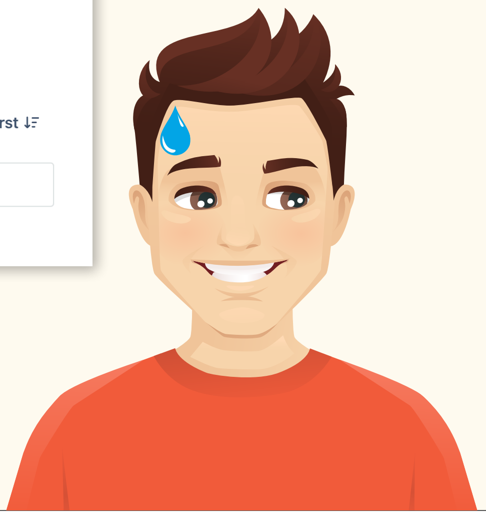
Activity

Show: Comments

Newest first

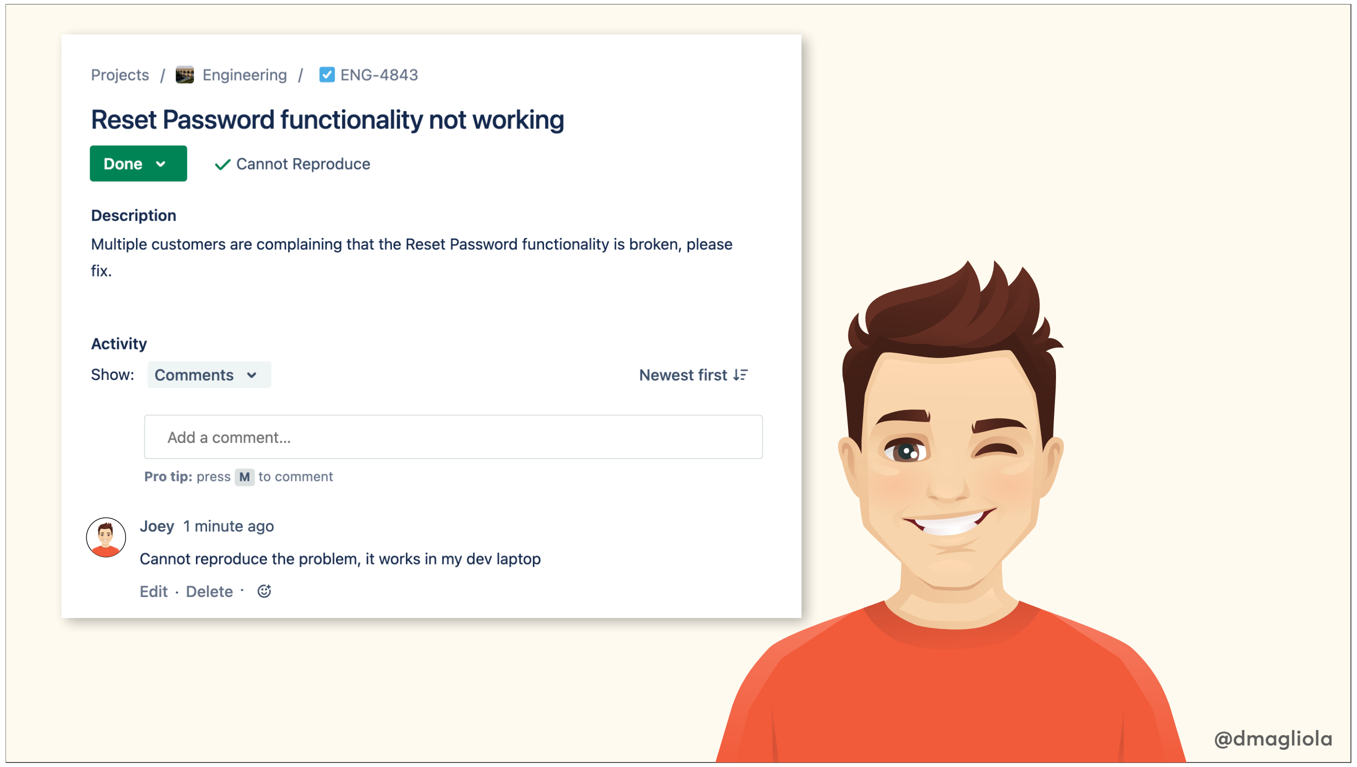
Add a comment...

Pro tip: press M to comment



@dmagliola

"but it works on my machine!", he says,



Projects / Engineering / ENG-4843

Reset Password functionality not working

✓ Cannot Reproduce

Description

Multiple customers are complaining that the Reset Password functionality is broken, please fix.

Activity

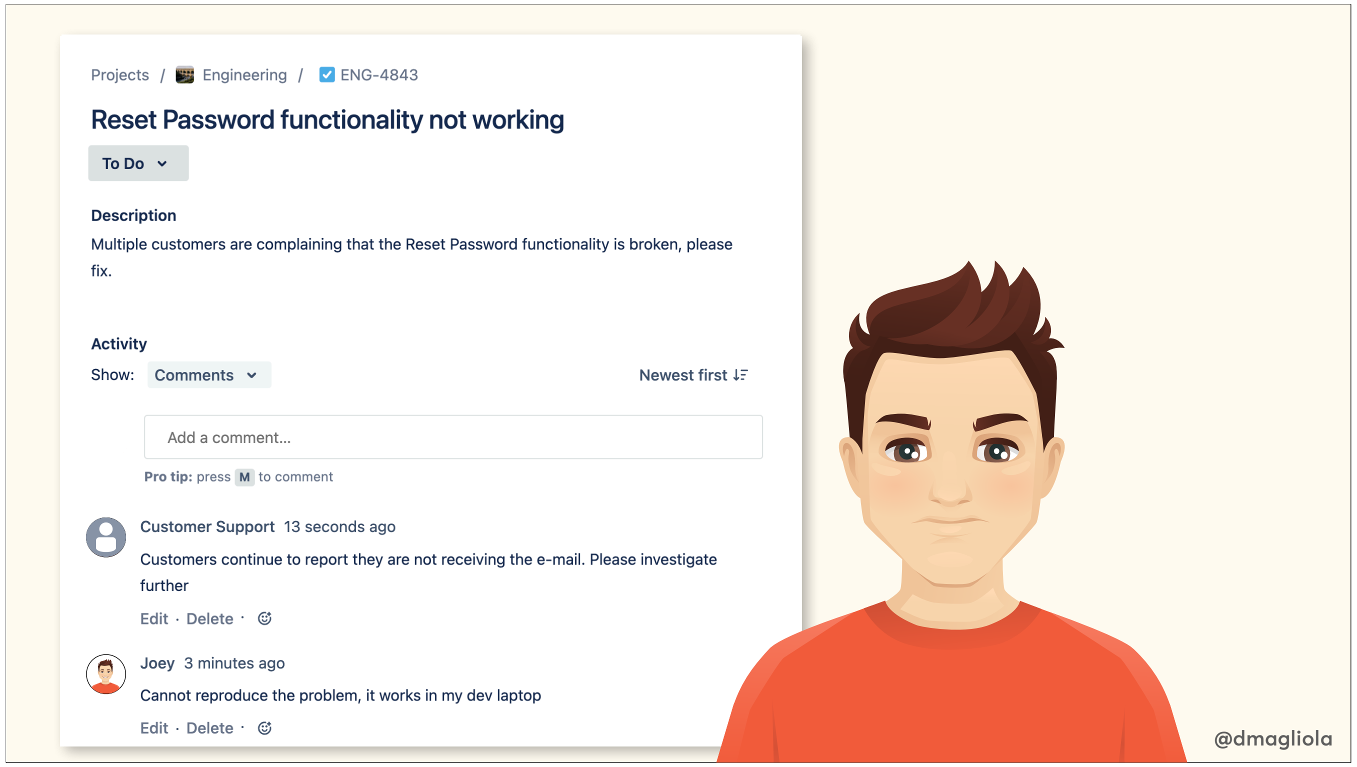
Show: Comments

Joey 1 minute ago

Cannot reproduce the problem, it works in my dev laptop

and closes as "Can't Reproduce".

Of course, the ticket comes back with more details



Projects / Engineering / ☑ ENG-4843

Reset Password functionality not working

To Do ⌄

Description

Multiple customers are complaining that the Reset Password functionality is broken, please fix.

Activity

Show: Comments ⌄ Newest first

Add a comment...

Pro tip: press M to comment

Customer Support 13 seconds ago

Customers continue to report they are not receiving the e-mail. Please investigate further

Edit · Delete ·

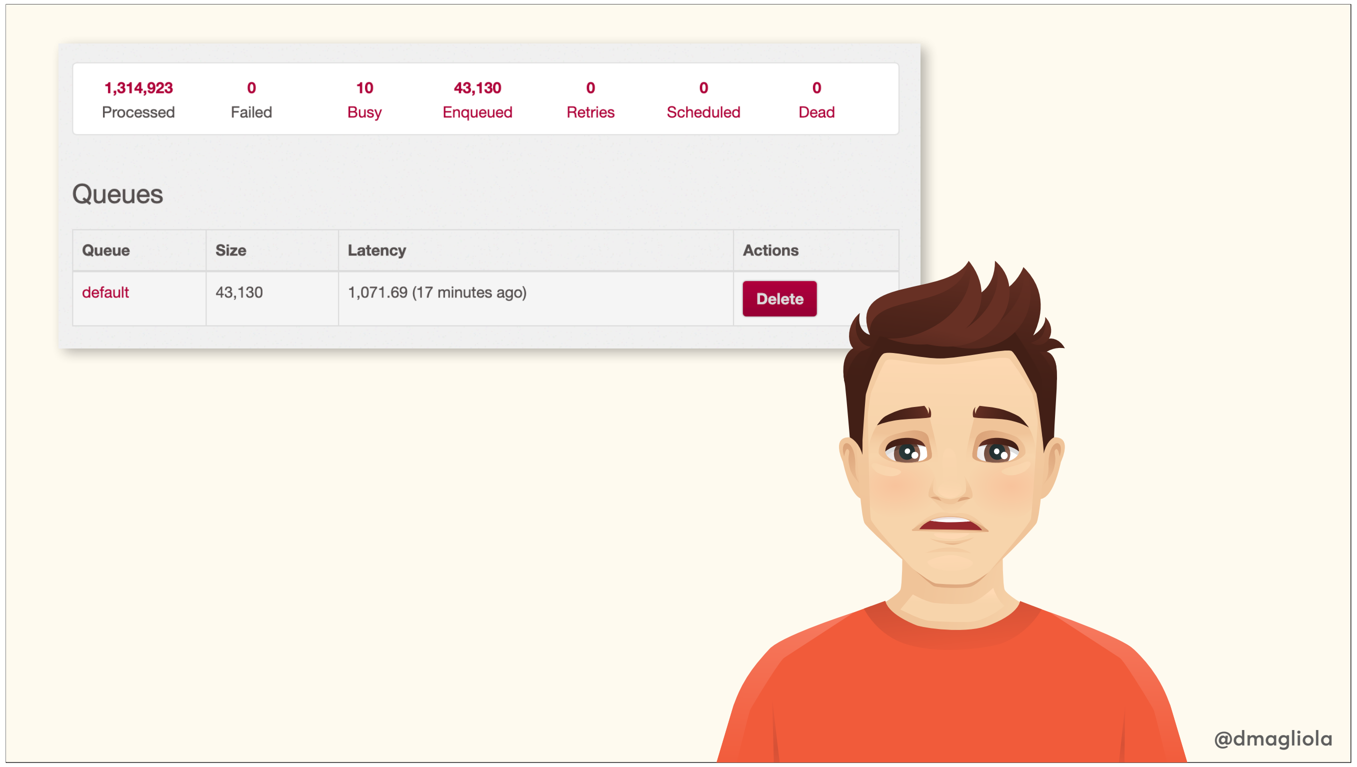
Joey 3 minutes ago

Cannot reproduce the problem, it works in my dev laptop

Edit · Delete ·

@dmagliola

"Users say they are still not receiving the email". And sure enough, when Joey tests this again in Production, he indeed does not get the email.



1,314,923	0	10	43,130	0	0	0
Processed	Failed	Busy	Enqueued	Retries	Scheduled	Dead

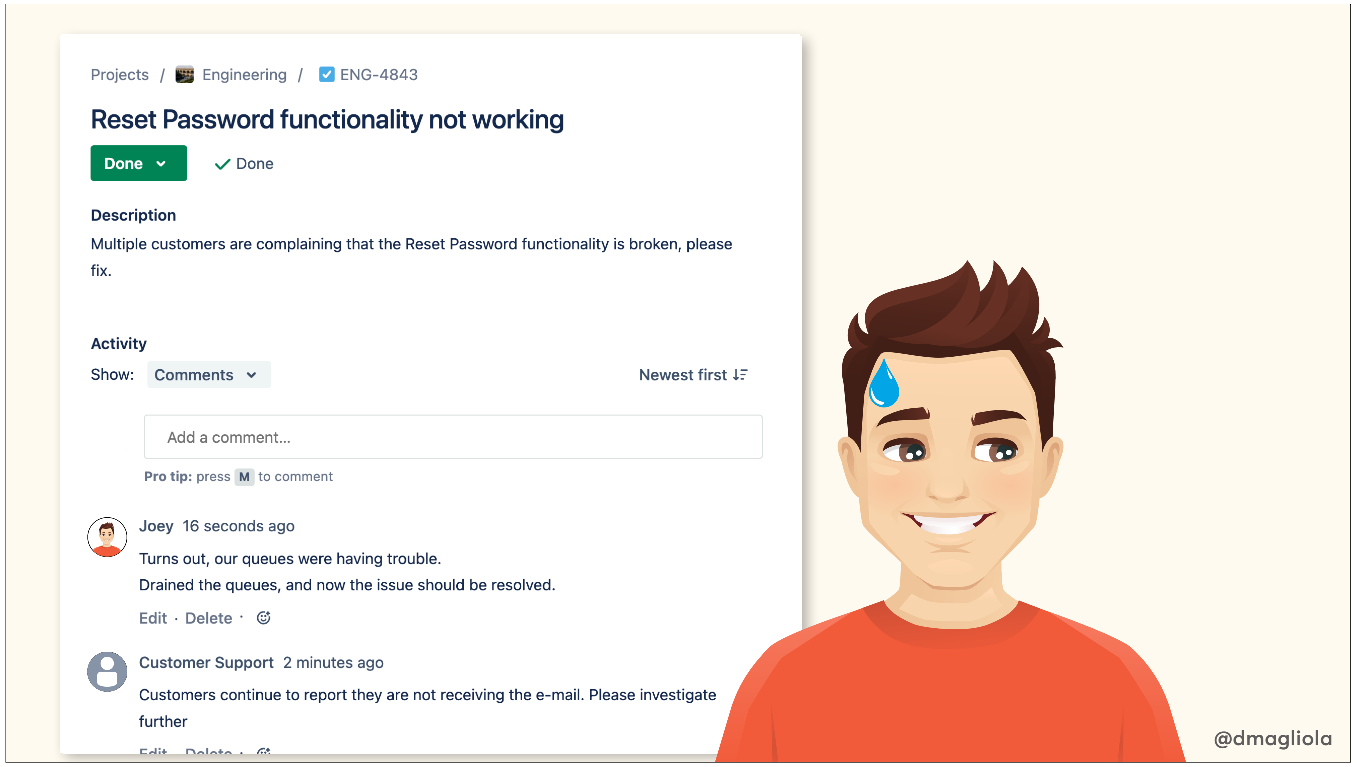
Queues

Queue	Size	Latency	Actions
default	43,130	1,071.69 (17 minutes ago)	Delete

@dmagliola

After a bit of investigation, it turns out that their queue has 40,000 jobs in it, and emails are getting stuck there and going out late.

Joey spins up some extra workers so the queue drains faster, and marks the ticket as Resolved.



Reset Password functionality not working

Description

Multiple customers are complaining that the Reset Password functionality is broken, please fix.

Joey 16 seconds ago

Turns out, our queues were having trouble.
Drained the queues, and now the issue should be resolved.

Customer Support 2 minutes ago

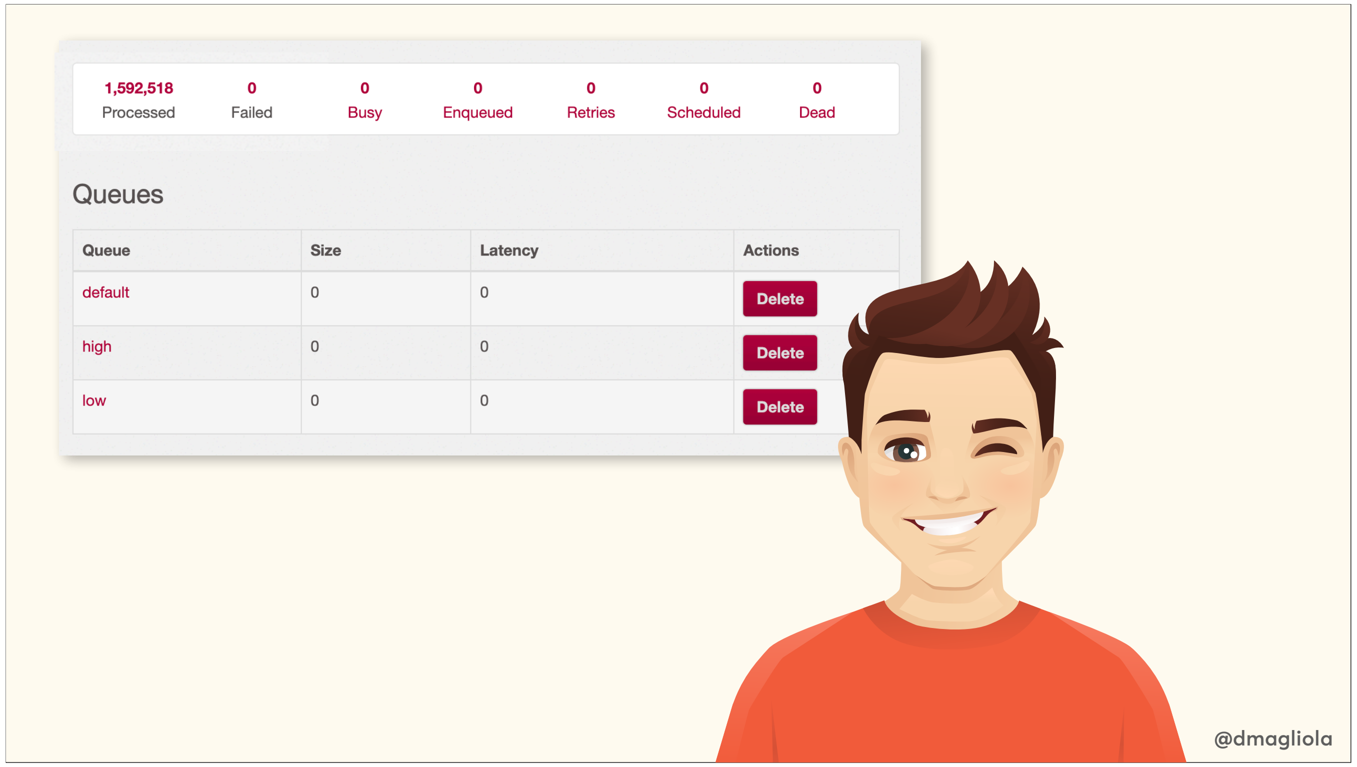
Customers continue to report they are not receiving the e-mail. Please investigate further

But… this had customer impact, so he decides to call it an incident; and while he's writing the post-mortem, he starts thinking "how can we prevent this from happening again?"

Now Joey knows that when a job gets stuck in a queue, it's because other types of jobs are getting in its way, and one way to fix that is to try isolate those loads so they don't interfere with each other. "We can't have all jobs in one big bag and pick them in order, right? some of them are more important than others"

Now in some queueing systems, you can set priorities for your jobs, and this lets some jobs "skip the queue" and run faster. So Joey thought maybe that would be a good idea, but it turns out their queues don't support priorities. Jobs are picked up in the same order they are put in the queue.
(As an aside, by the way, that's a good thing, priorities won't really solve the problem, we'll see why later)

Instead, in this system, what you need to do to isolate loads is create separate queues.



1,592,518	0	0	0	0	0	0
Processed	Failed	Busy	Enqueued	Retries	Scheduled	Dead

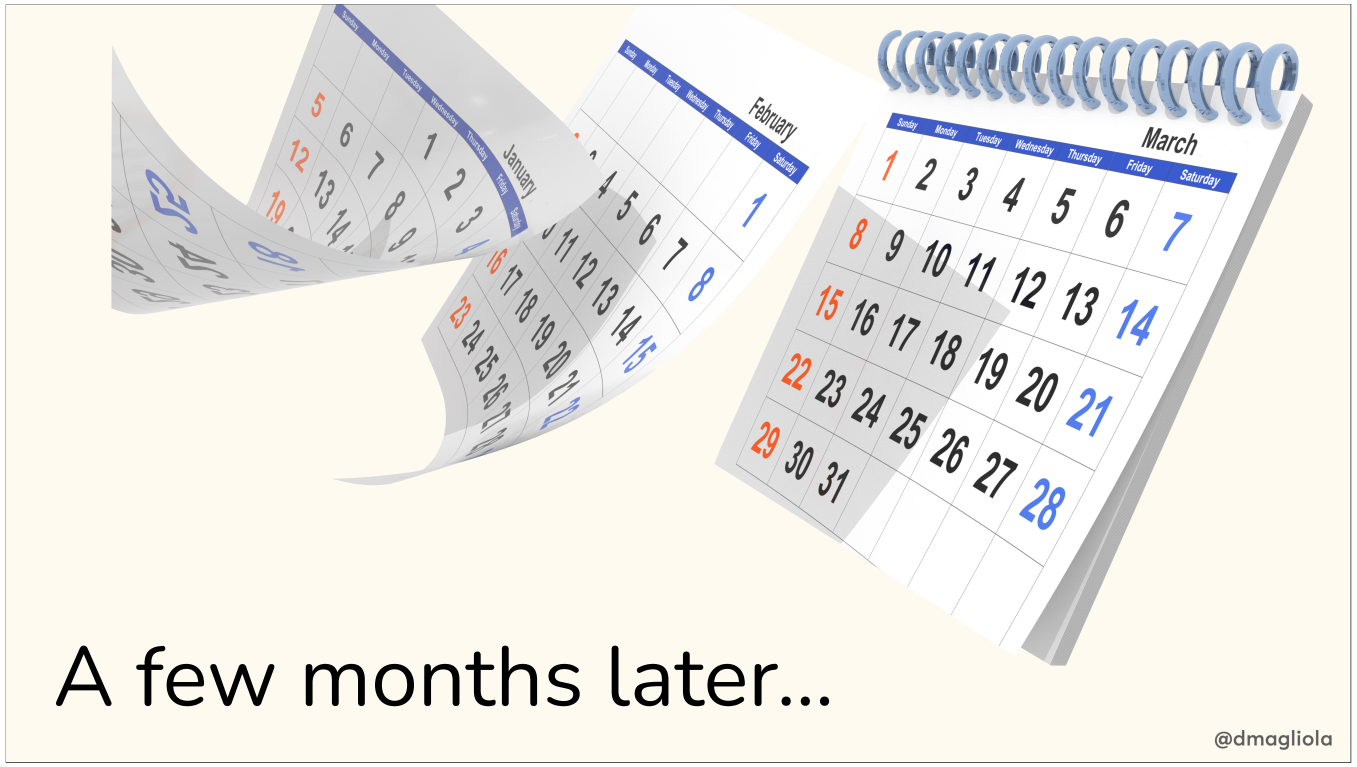
Queues

Queue	Size	Latency	Actions
default	0	0	Delete
high	0	0	Delete
low	0	0	Delete

@dmagliola

So Joey decides to create the "high" priority queue, for important jobs. And because programmers love us some symmetry, he also creates a "low" queue for less important stuff.

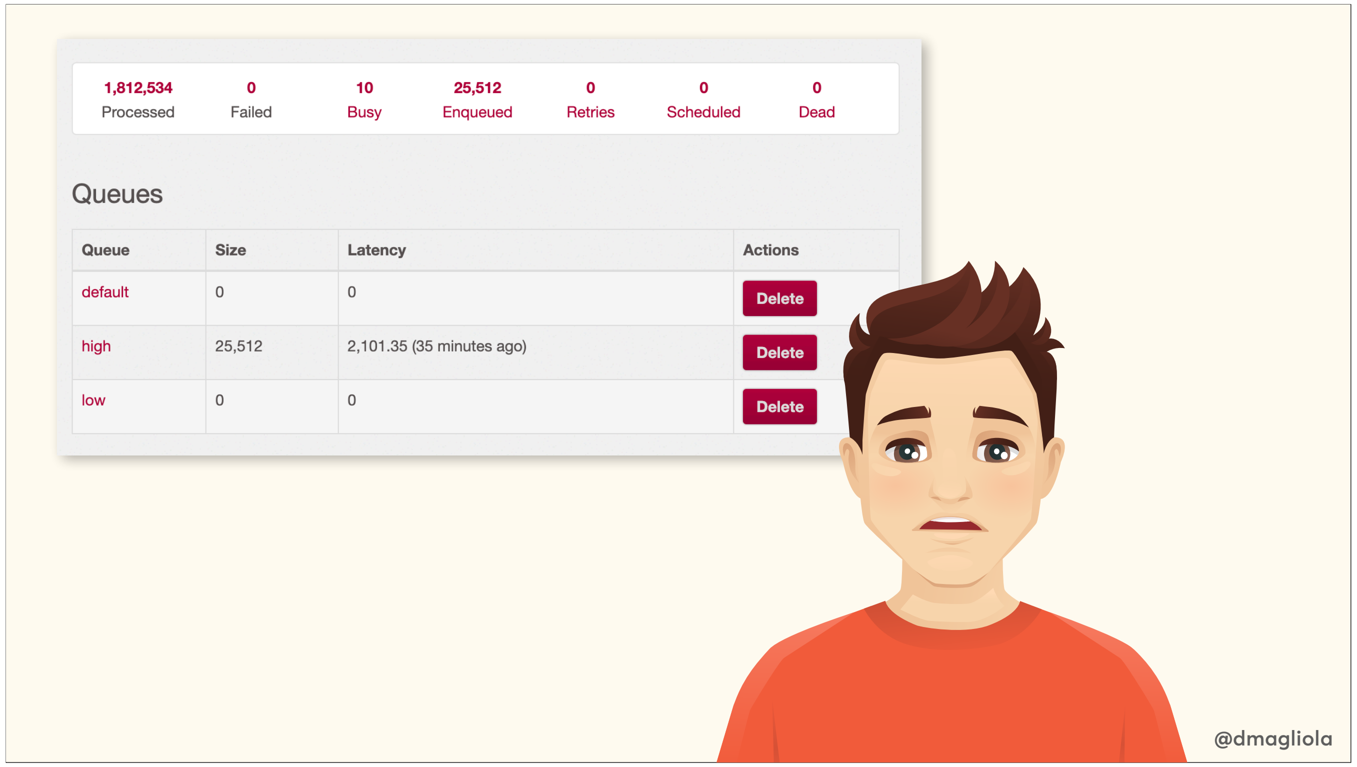
A few days later he finds some jobs that need to happen every now and then but they're not urgent; and they take a bit long to run.
It'd be a shame if something important was late because of them, so he puts them in that low queue.



A few months later...

@dmagliola

A few months down the road, Joey is dealing with another incident related to queues.



1,812,534	0	10	25,512	0	0	0
Processed	Failed	Busy	Enqueued	Retries	Scheduled	Dead

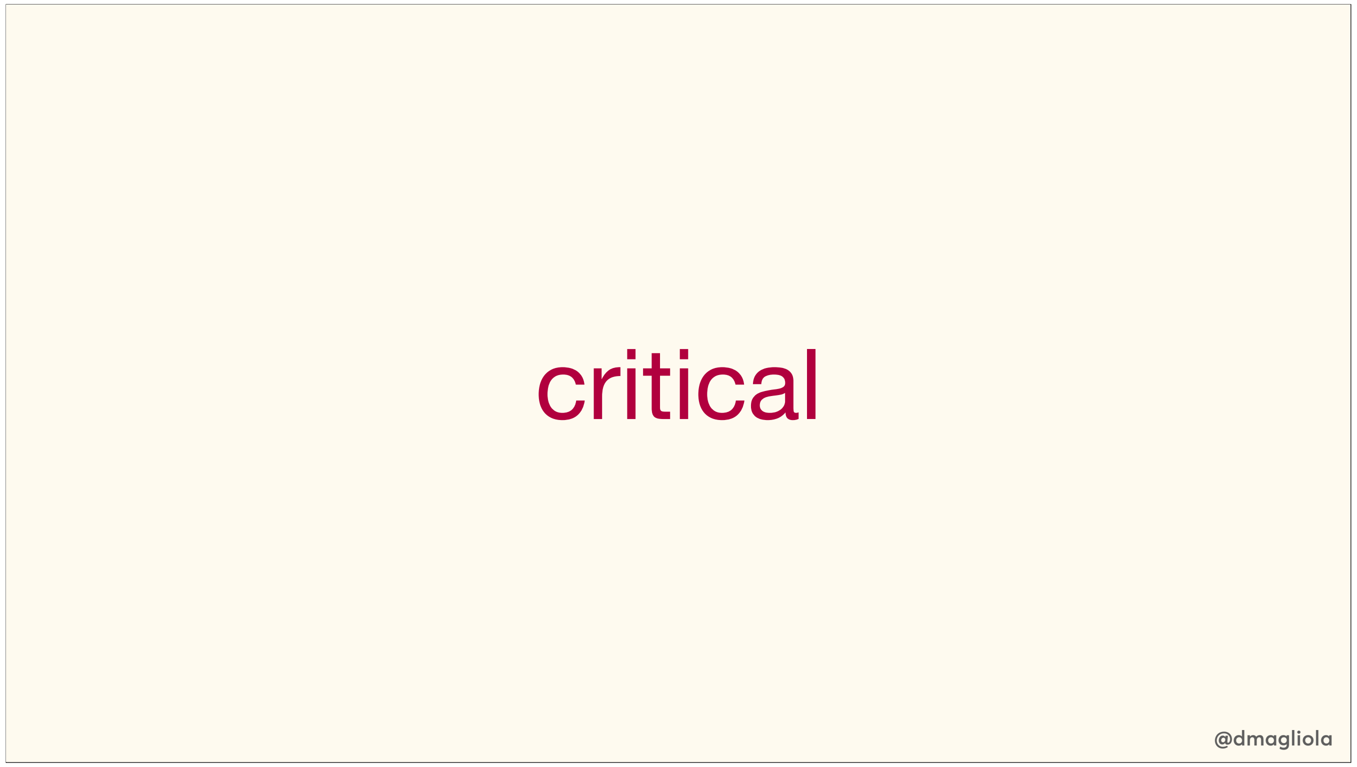
Queues

Queue	Size	Latency	Actions
default	0	0	Delete
high	25,512	2,101.35 (35 minutes ago)	Delete
low	0	0	Delete

@dmagliola

Turns out, everyone's jobs are "important", so the "high" queue has tons of different jobs, and now, Credit Card transactions are not being processed fast enough because of some other new job that's taking too long, and we're losing sales.

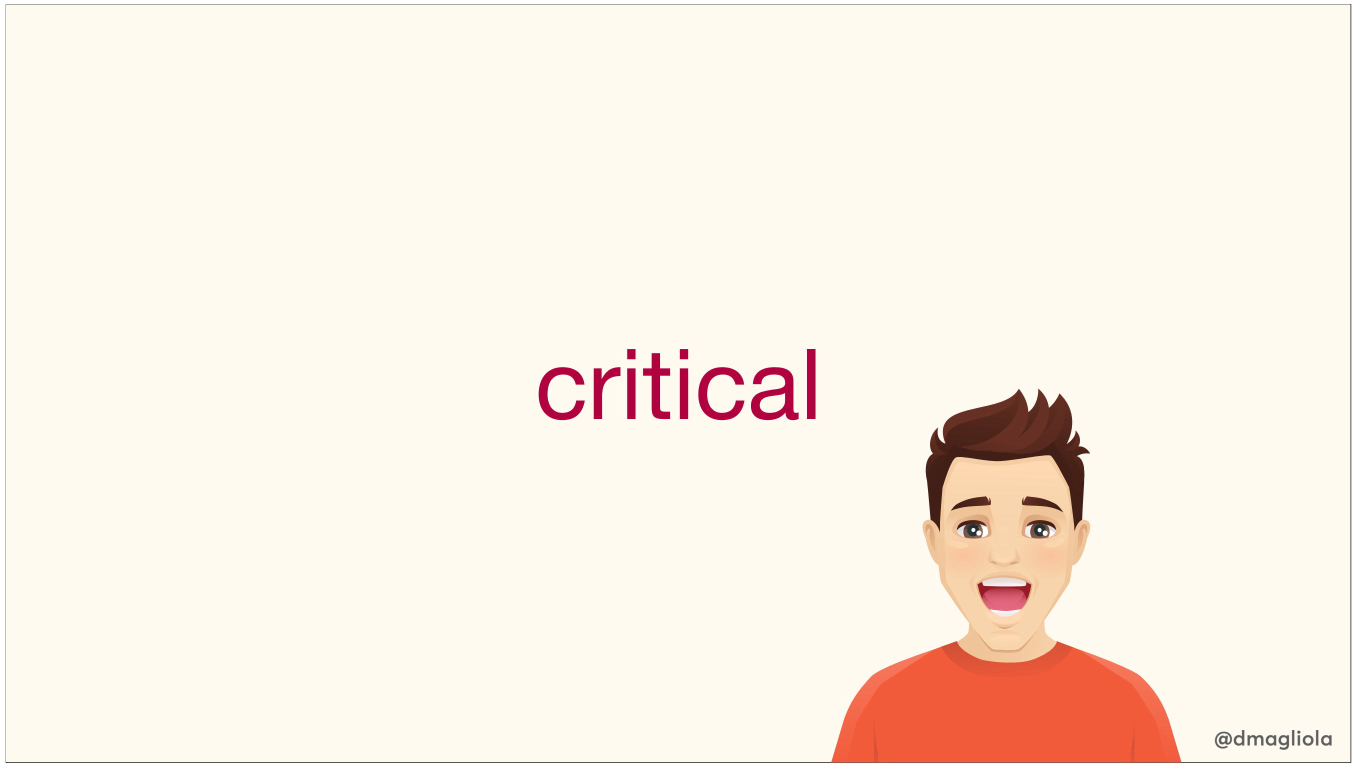
And so, after another post-mortem,



critical

@dmagliola

Joey christens the "critical" queue… And he makes it very clear to everyone.



critical

@dmagliola

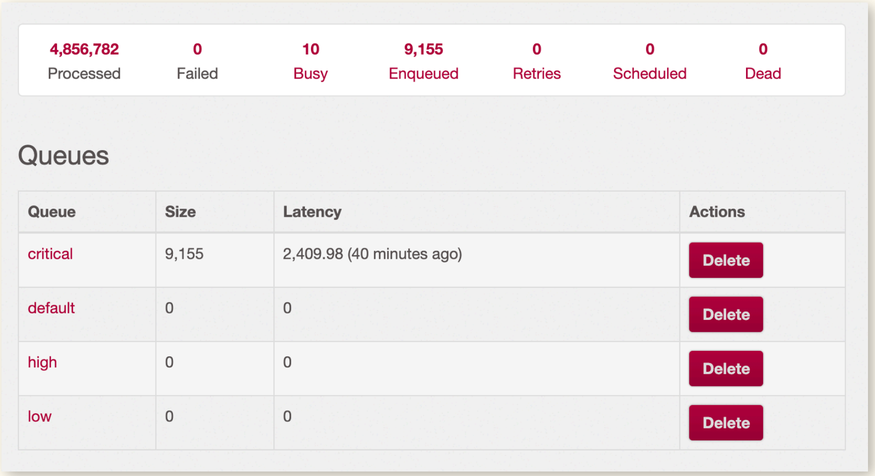
"I know your jobs are important, but only things that are very, very critical can go here".

We cannot let this happen again!

ACT 2

It happened again

I'm guessing none of you are surprised here…



4,856,782	0	10	9,155	0	0	0
Processed	Failed	Busy	Enqueued	Retries	Scheduled	Dead

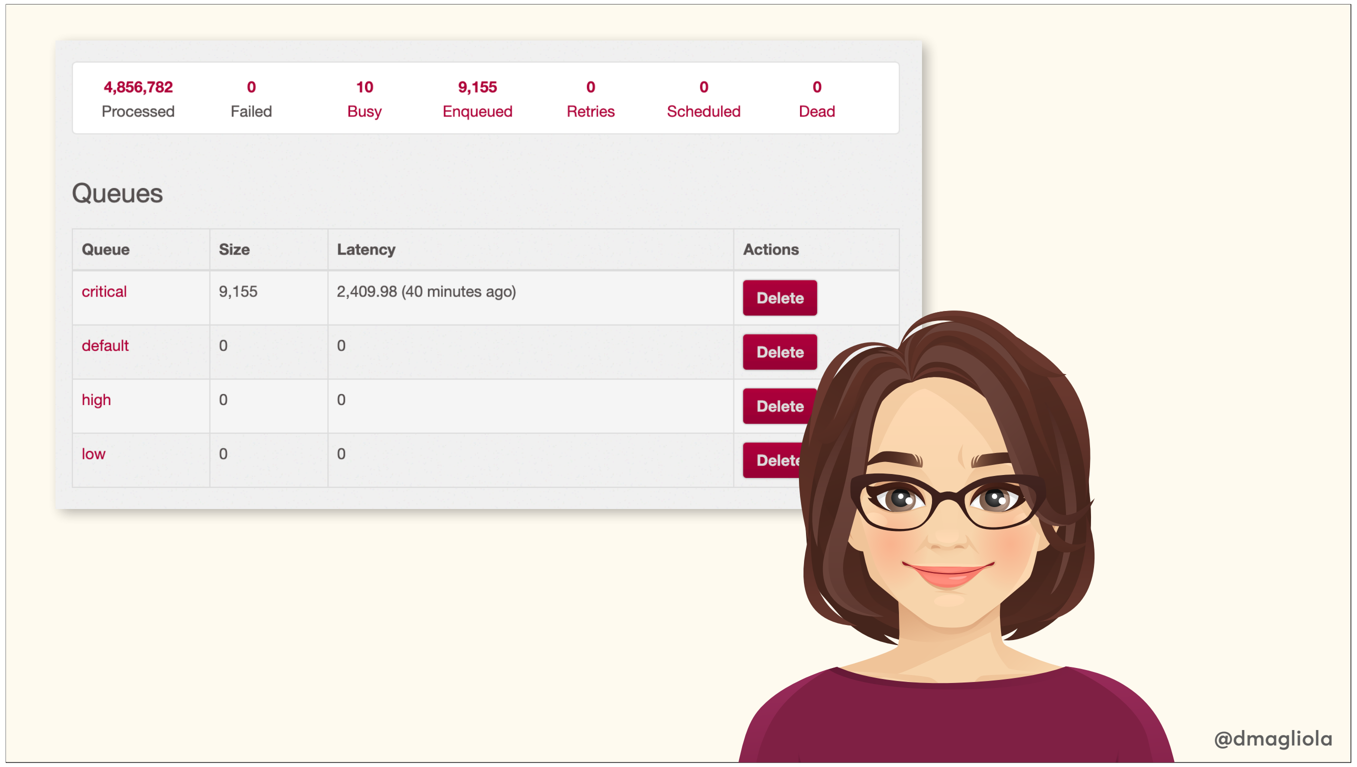
Queues

Queue	Size	Latency	Actions
critical	9,155	2,409.98 (40 minutes ago)	Delete
default	0	0	Delete
high	0	0	Delete
low	0	0	Delete

@dmagliola

A few months later, the Credit Card company had an outage and Credit Card jobs, which are "critical" and normally finish in about a second, were taking a whole minute until they timed out. This started backing up the queue, and 2FA text messages, which are also "critical", started going out late. Again.

But by this point the company had hired a very senior engineer from a much bigger company, and she brought a lot of hard-won experience with her.



Processed	Failed	Busy	Enqueued	Retries	Scheduled	Dead
4,856,782	0	10	9,155	0	0	0

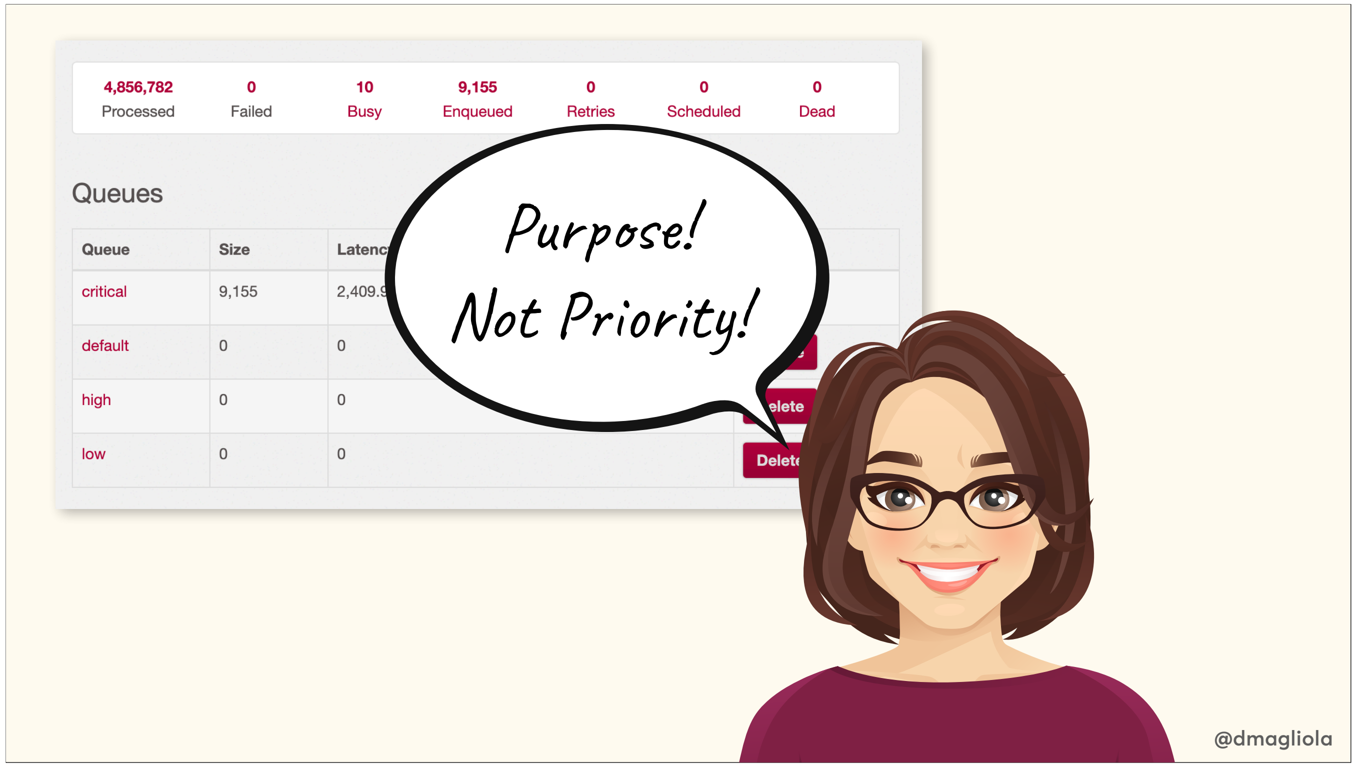
Queues

Queue	Size	Latency	Actions
critical	9,155	2,409.98 (40 minutes ago)	Delete
default	0	0	Delete
high	0	0	Delete
low	0	0	Delete

This is Marianne, and when she noticed this incident, she recognized the problem immediately, and got involved.

Marianne has seen this exact pattern before and she knows what the problem is. Organizing queues based on "priority" or "importance" is doomed to fail. First of all, there is no clear meaning of what is "high priority" or "low priority", and yes, there are docs with examples of what should go where, but they can never predict everything that we'll need to throw at our queues. Some of these jobs will also be very high volume or long-running and it's really hard to predict how all of them will interact.

But she's seen this before, and she has the answer:



4,856,782	0	10	9,155	0	0	0
Processed	Failed	Busy	Enqueued	Retries	Scheduled	Dead

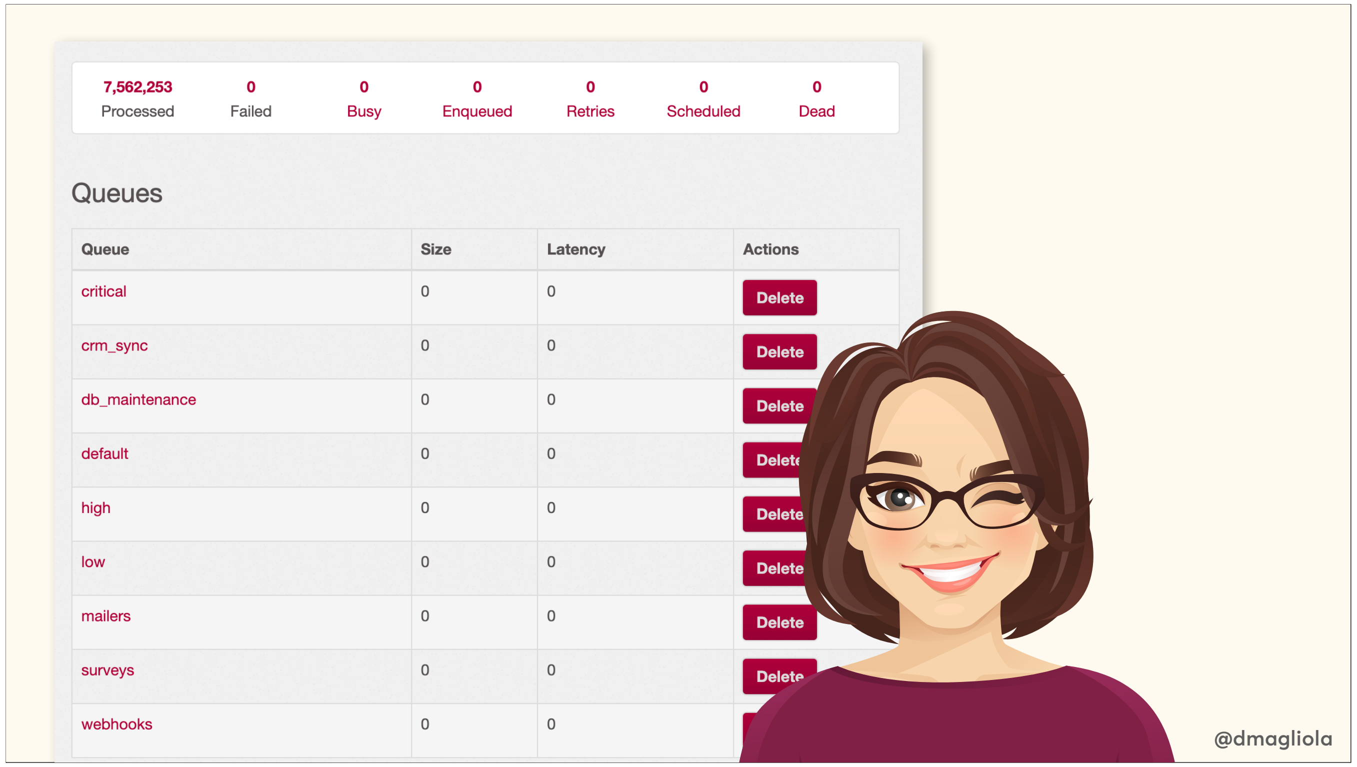
Queues

Queue	Size	Latenc	
critical	9,155	2,409.9	
default	0	0	
high	0	0	elete
low	0	0	Delet

@dmagliola

we need to have separate different queues for the different purposes, not "priorities". That way, jobs that run together all look similar, making performance more predictable, and it's also going to be much clearer what goes where. That way, if Credit Cards start having trouble, they don't interfere with unrelated stuff like 2FA.

Now, of course, you can't have a queue for every purpose (although some companies have tried), but you can define a few for your most common things. Marianne sets up a queue for "mailers", which are generally pretty important. She also notices we send millions and millions of Customer Satisfaction Survey emails and those are NOT important, so she adds a "surveys" queue.



7,562,253	0	0	0	0	0	0
Processed	Failed	Busy	Enqueued	Retries	Scheduled	Dead

Queues

Queue	Size	Latency	Actions
critical	0	0	Delete
crm_sync	0	0	Delete
db_maintenance	0	0	Delete
default	0	0	Delete
high	0	0	Delete
low	0	0	Delete
mailers	0	0	Delete
surveys	0	0	Delete
webhooks	0	0	

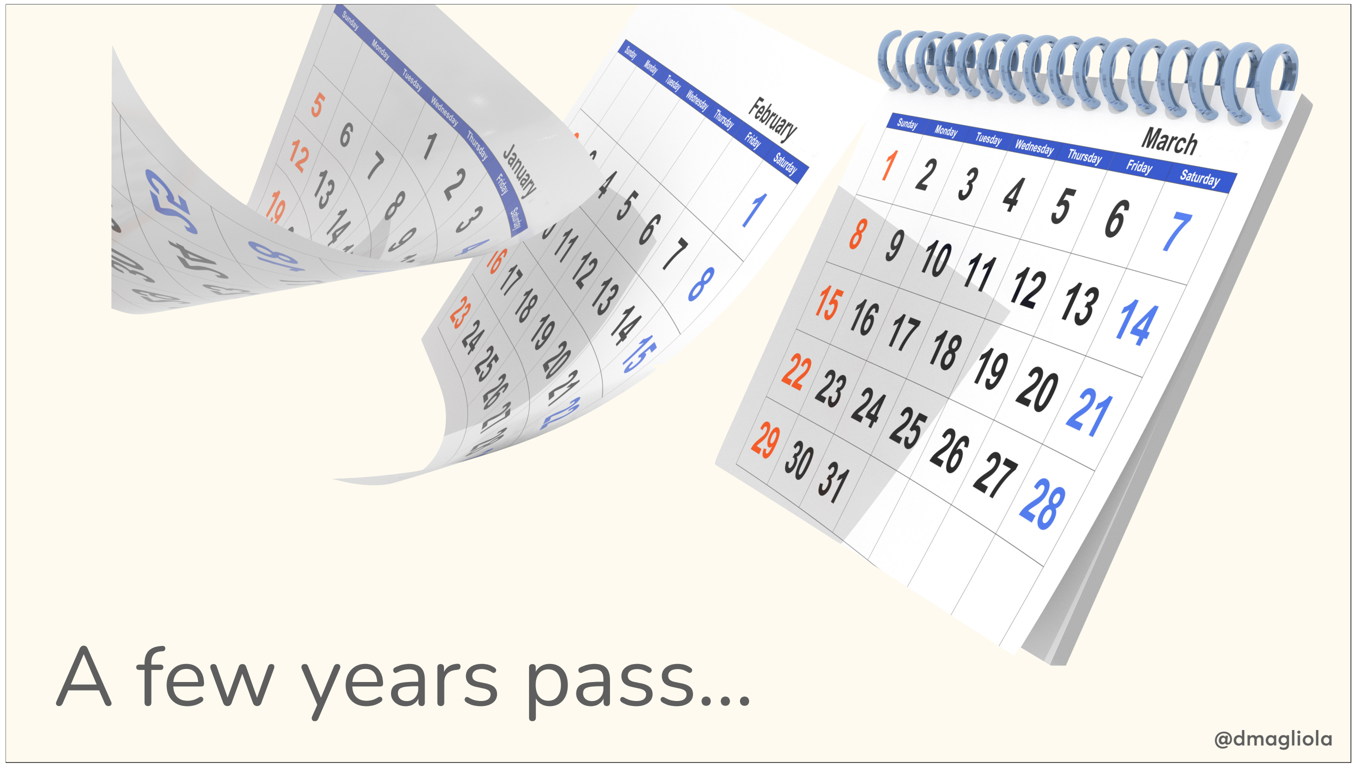
@dmagliola

A few months later, you have something like this, and jobs are humming along just fine.
Every now and then a new queue is born, and everyone is happy with this.

ACT 3

Too many cooks

for a while…

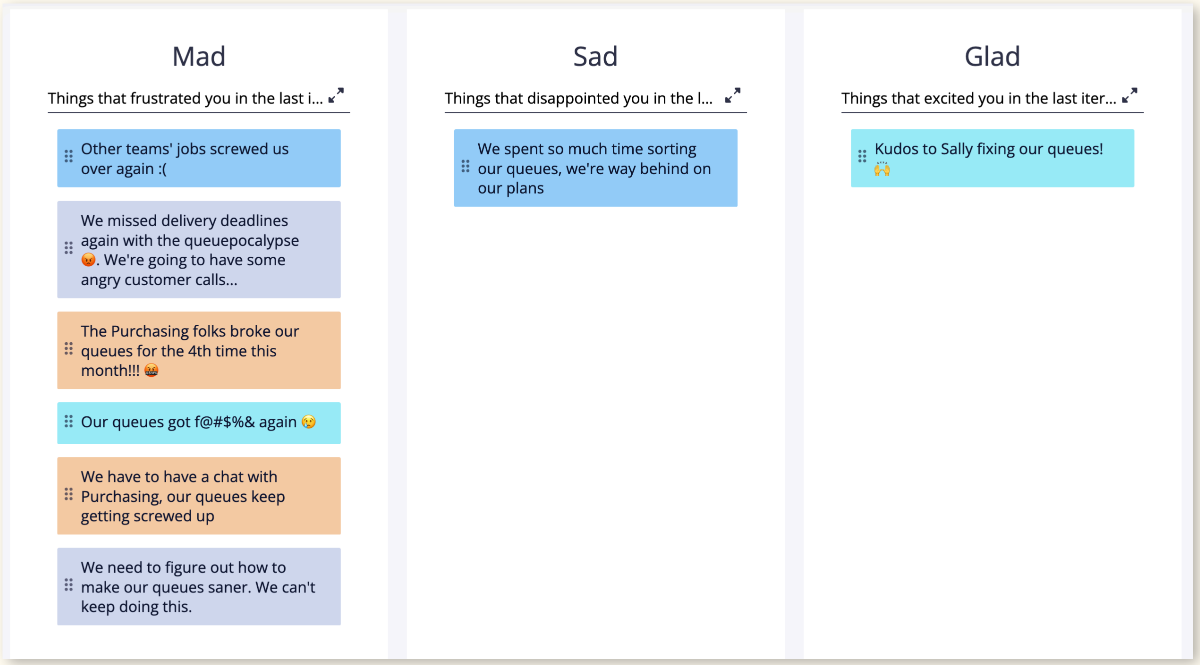


A few years pass…

@dmagliola

Fast forward a few years, and our little company has grown quite a bit by now, we have a couple dozen developers, and as usual they are split into teams, based on the different functionalities of their app. You have the Purchasing folks buying the paper, the Inventory people keeping track of it, the Logistics team, etc.

One day, the Logistics team is having a little retro,



Mad

Things that frustrated you in the last i...

Other teams' jobs screwed us over again :(

We missed delivery deadlines again with the queuepocalypse 😡. We're going to have some angry customer calls...

The Purchasing folks broke our queues for the 4th time this month!!! 🤬

Our queues got f@#$%& again 😢

We have to have a chat with Purchasing, our queues keep getting screwed up

We need to figure out how to make our queues saner. We can't keep doing this.

Sad

Things that disappointed you in the l...

We spent so much time sorting our queues, we're way behind on our plans

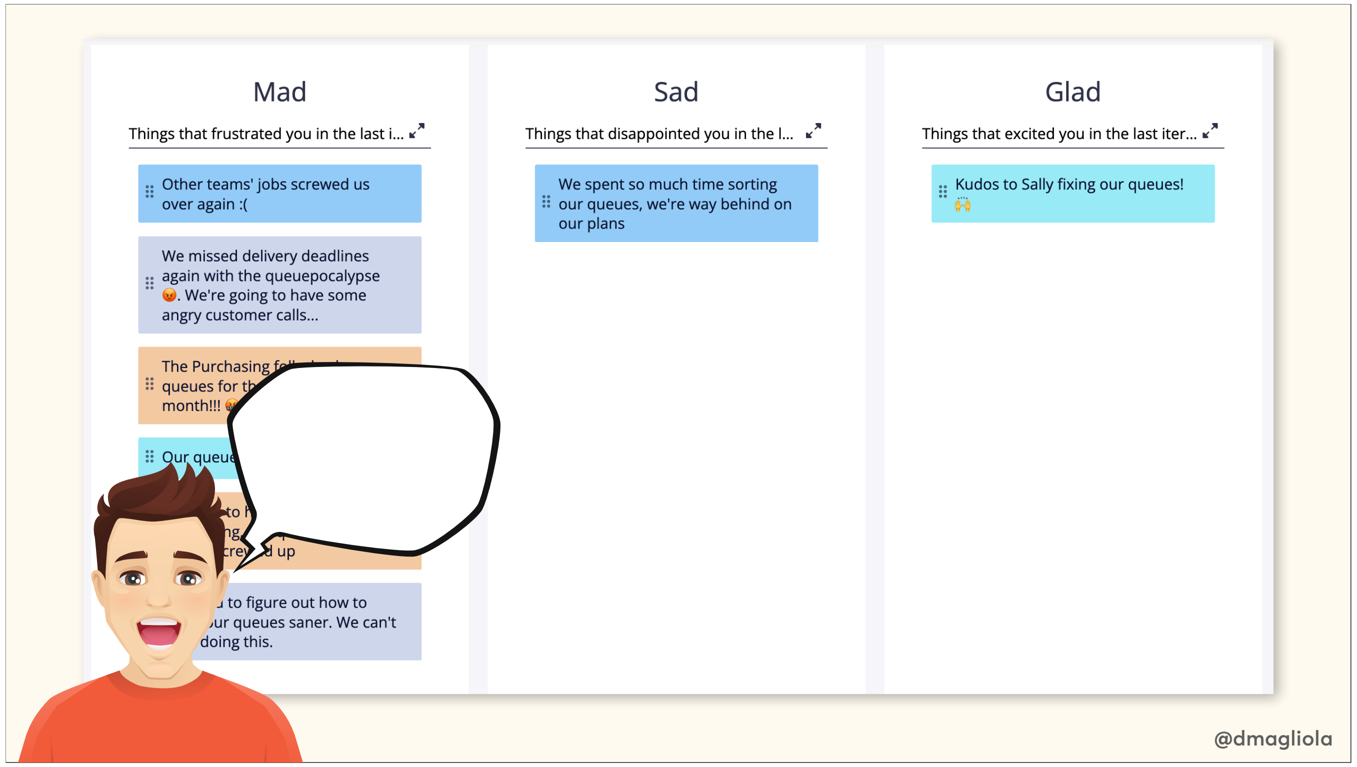
Glad

Things that excited you in the last iter...

Kudos to Sally fixing our queues! 🙌

@dmagliola

and the "Mad" column is pretty long.



Mad

Things that frustrated you in the last i...

- Other teams' jobs screwed us over again :(
- We missed delivery deadlines again with the queuepocalypse 😡. We're going to have some angry customer calls...
- The Purchasing f queues for th month!!! 🤬
- Our queue
- to h ng, crey d up
- to figure out how to our queues saner. We can't doing this.

Sad

Things that disappointed you in the l...

- We spent so much time sorting our queues, we're way behind on our plans

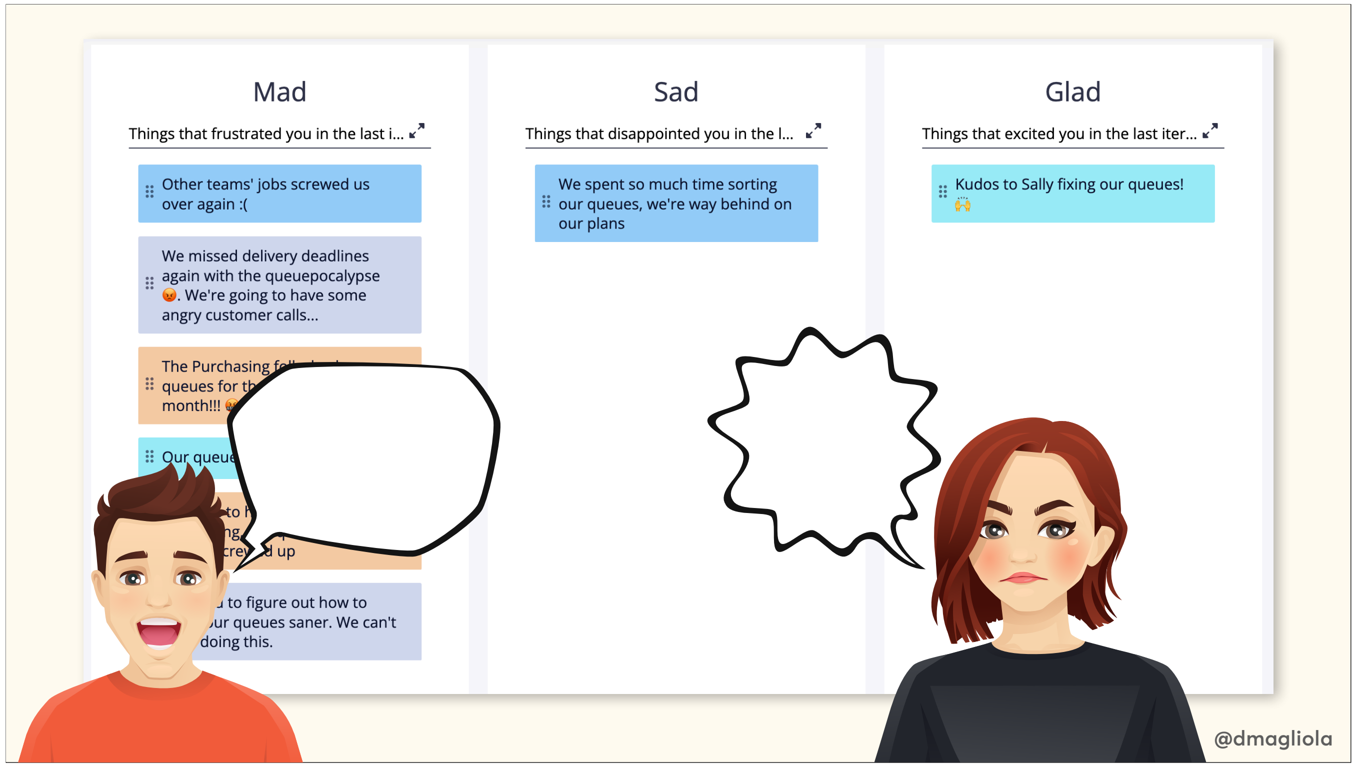
Glad

Things that excited you in the last iter...

- Kudos to Sally fixing our queues! 🙌

@dmagliola

"The Purchasing folks have done it again"



Mad

Things that frustrated you in the last i...

Other teams' jobs screwed us over again :(

We missed delivery deadlines again with the queuepocalypse 😡. We're going to have some angry customer calls...

The Purchasing f queues for th month!!! 🤬

Our queue

to h ng, crey d up

to figure out how to our queues saner. We can't doing this.

Sad

Things that disappointed you in the l...

We spent so much time sorting our queues, we're way behind on our plans

Glad

Things that excited you in the last iter...

Kudos to Sally fixing our queues! 🙌

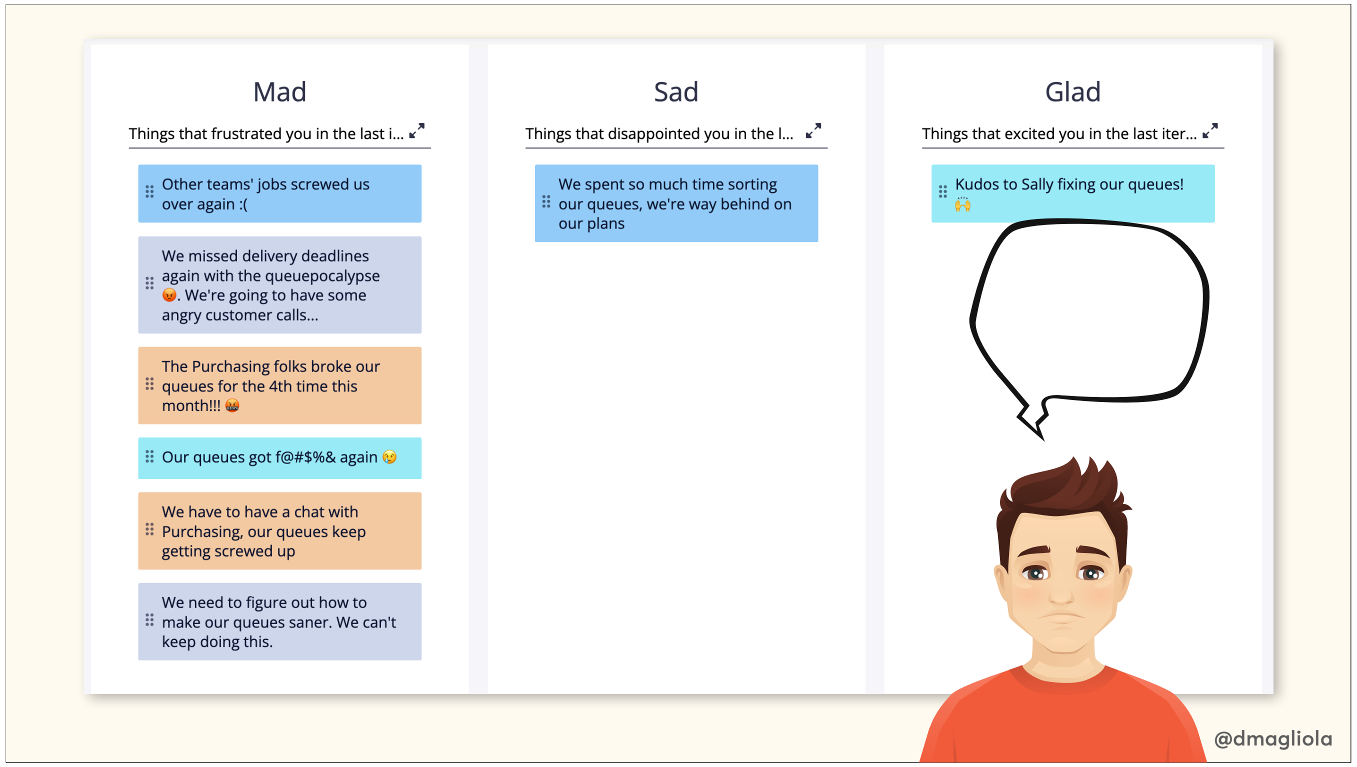
@dmagliola

"This is the 4th time this happens this quarter"

Mad
Things that frustrated you in the last i...
Other teams' jobs screwed us over again :(
We missed delivery deadlines again with the queuepocalypse 😡. We're going to have some angry customer calls...
The Purchasing f
queues for th
month!!! 🤬
Our queue
to h
ng
crey
d up
u to figure out how to
our queues saner. We can't
doing this.
Sad
you in the l...
rting
Glad
Things that excited you in the last iter...
Kudos to Sally fixing our queues! 🙌
@dmagliola

"How many times do we need to tell them"

Turns out, Purchasing had this genius idea. Instead of calling vendors and asking them for better prices, they could just auto-email everyone and let the computer do the haggling. The feature was a little aggressive, and there were a lot of emails; the trucking company didn't get the notifications they needed, and Shipments went out late that day. Again.



Mad

Things that frustrated you in the last i...

- Other teams' jobs screwed us over again :(
- We missed delivery deadlines again with the queuepocalypse 😡. We're going to have some angry customer calls...
- The Purchasing folks broke our queues for the 4th time this month!!! 🤬
- Our queues got f@#$%& again 😢
- We have to have a chat with Purchasing, our queues keep getting screwed up
- We need to figure out how to make our queues saner. We can't keep doing this.

Sad

Things that disappointed you in the l...

- We spent so much time sorting our queues, we're way behind on our plans

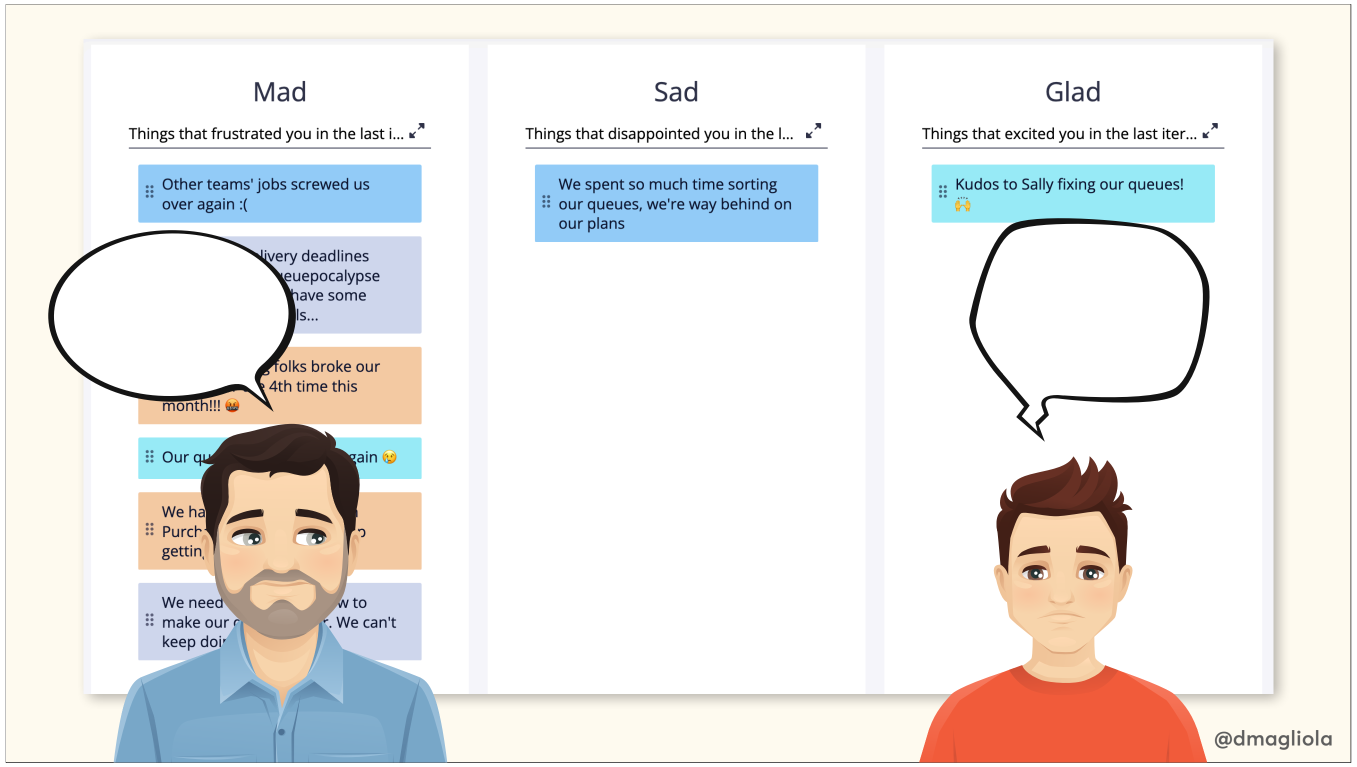
Glad

Things that excited you in the last iter...

- Kudos to Sally fixing our queues! 🙌

@dmagliola

"It's not always Purchasing though. We've had trouble with the Sales team doing the same thing too".



Mad

Things that frustrated you in the last i...

Other teams' jobs screwed us over again :(

...livery deadlines ...euepocalypse ...have some ...ls...

...folks broke our ...4th time this month!!! 😡

Our qu... ...gain 😢

We ha... Purch... gettin... ...o

We need... ...w to make our ... . We can't keep doi...

Sad

Things that disappointed you in the l...

We spent so much time sorting our queues, we're way behind on our plans

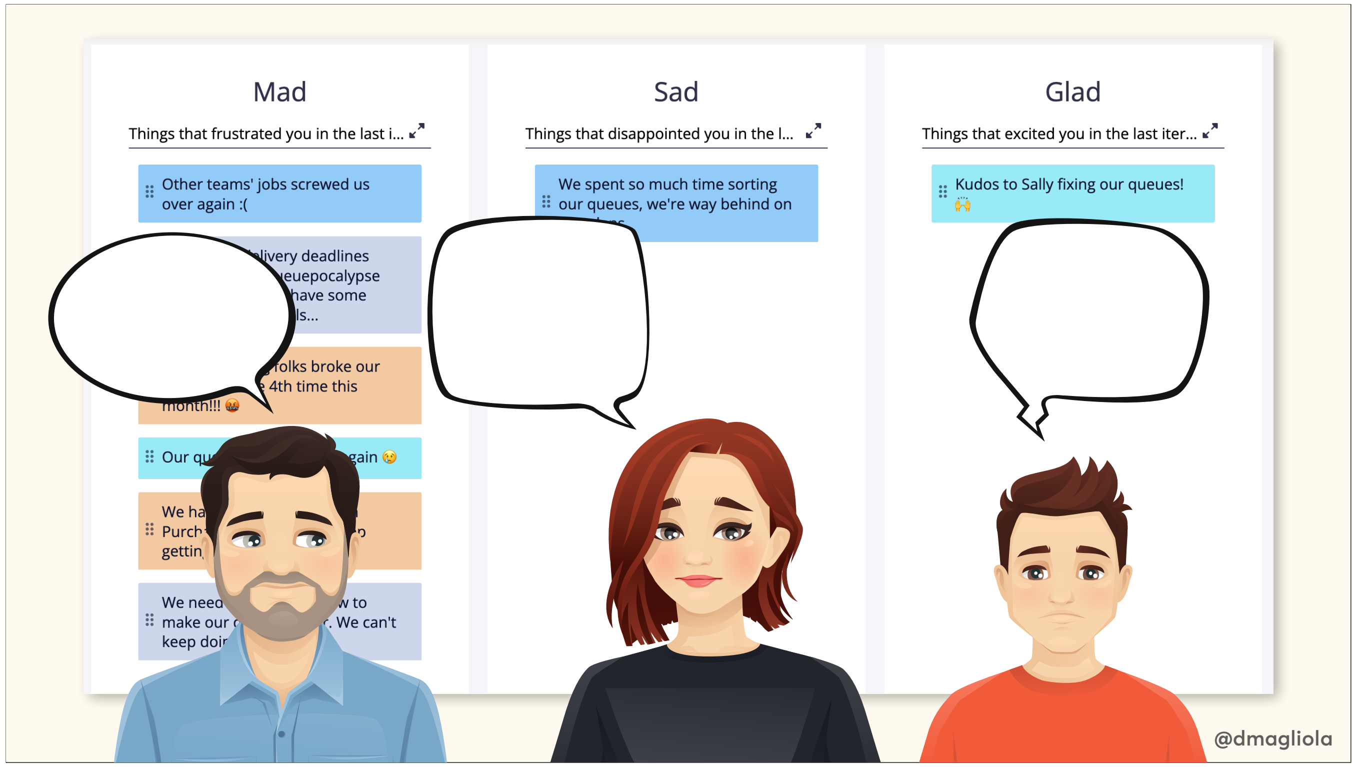
Glad

Things that excited you in the last iter...

Kudos to Sally fixing our queues! 🙌

"And to be fair, we've also done this. Remember the ShipPocalypse last Cyber Monday, where we clogged every queue in the system and ruined everyone's day?"

The conclusion seems clear...



Mad

Things that frustrated you in the last i...

Other teams' jobs screwed us over again :(

...livery deadlines ...eueopocalypse ...have some ...ls...

...folks broke our ...4th time this month!!! 😡

Our qu... ...gain 😢

We ha... Purch... gettin...

We need... ...w to make our ... . We can't keep doir...

Sad

Things that disappointed you in the l...

We spent so much time sorting our queues, we're way behind on

Glad

Things that excited you in the last iter...

Kudos to Sally fixing our queues! 🙌

@dmagliola

"We have all these queues, but we have no control over what other teams are putting in them. And they don't know what our jobs are and what our requirements are for those jobs."

Mad

Things that frustrated you in the last i...

Other teams' jobs screwed us over again :(

We missed delivery d again with the queu 😡. We're going to h angry customer calls

The Purchasing folks brok ur queues for the 4th time this month!!! 🤬

Our qu gain 😢

We ha Purch gettin

We need w to make our r. We can't keep doir

Sad

Things that disappointed you in the l...

We spent so much time sorting our queues, we're way behind on our plans

Glad

Things that excited you in the last iter...

Kudos to Sally fixing our queues! 🙌

"There's only one way to fix this".

Teams need to be able to act independently of each other, that's the point of teams.

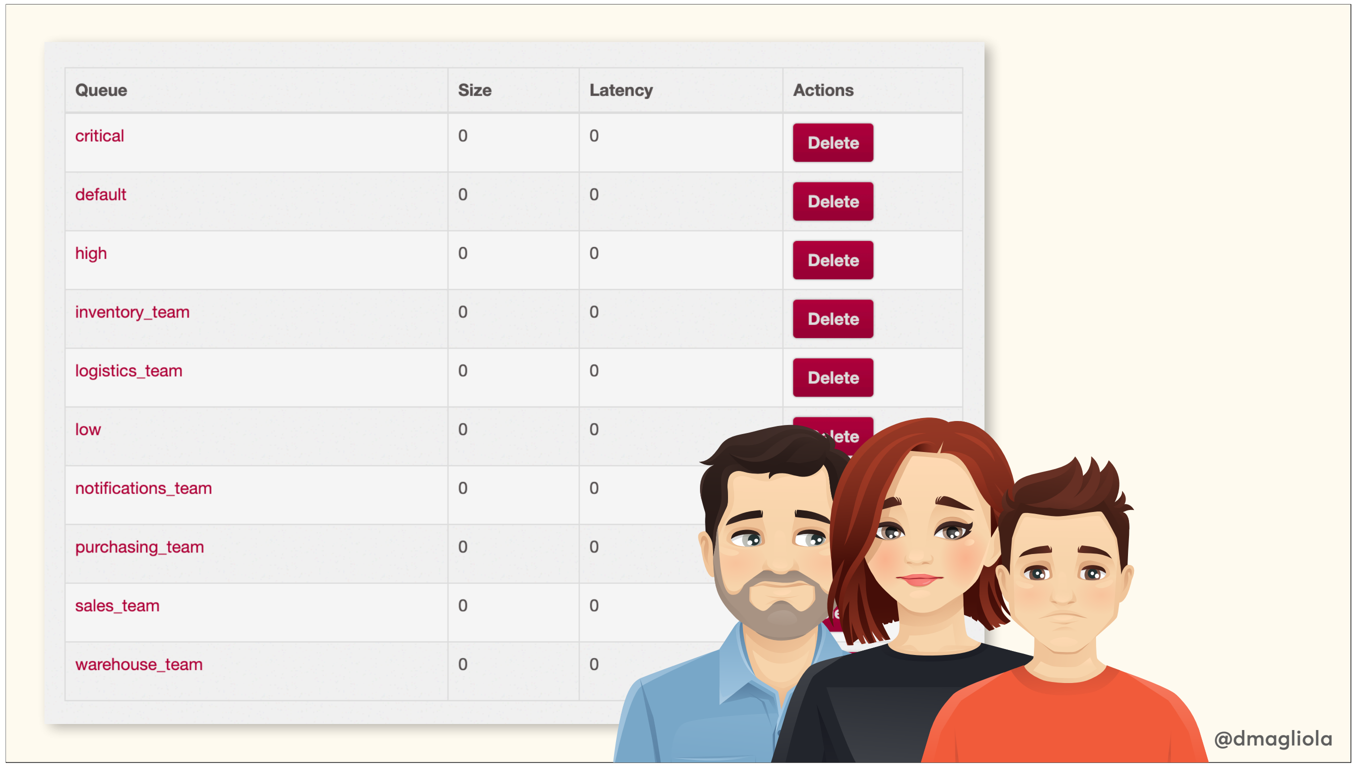
logistics_team

@dmagliola

And just like that, the "logistics_team" queue is born.

Look… at least they didn't go for microservices!

Now of course, it's not just the Logistics team that gets their queue.

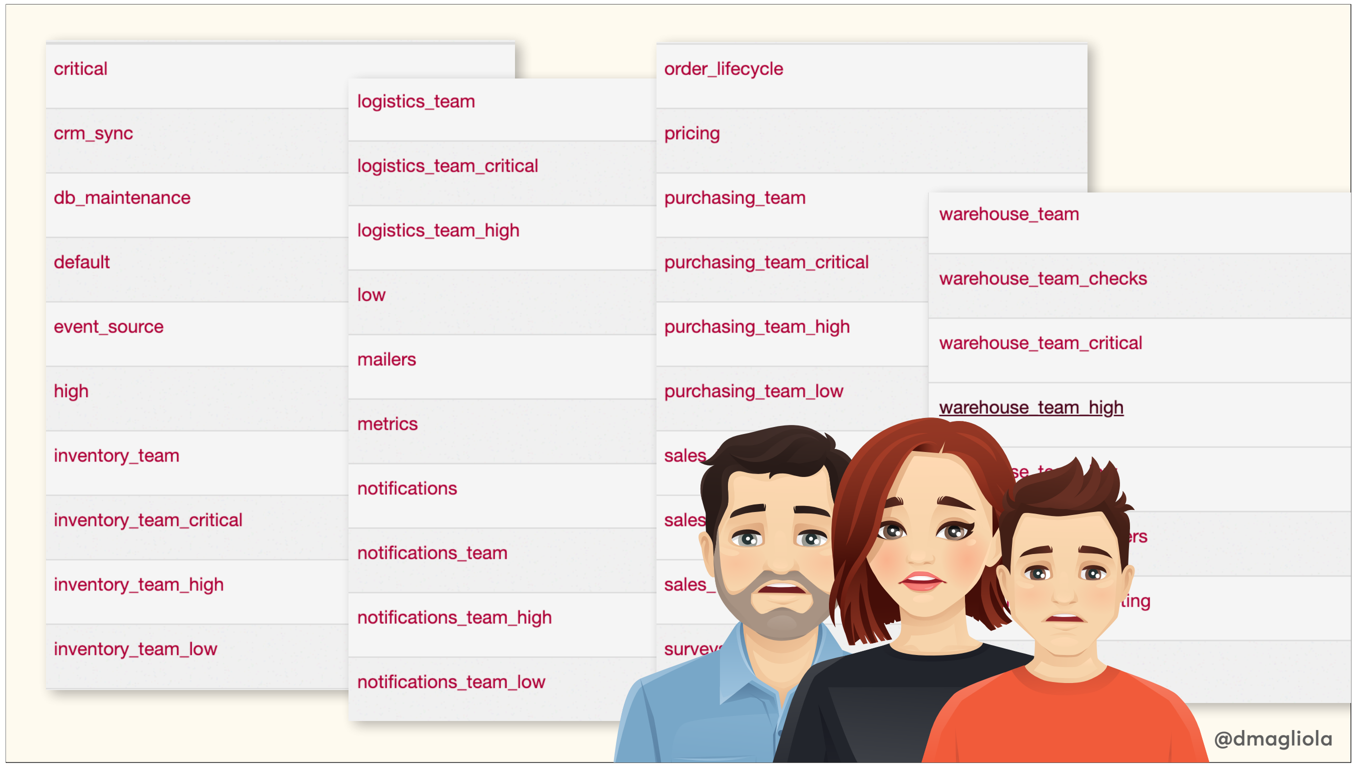


Queue	Size	Latency	Actions
critical	0	0	Delete
default	0	0	Delete
high	0	0	Delete
inventory_team	0	0	Delete
logistics_team	0	0	Delete
low	0	0	lete
notifications_team	0	0	
purchasing_team	0	0	
sales_team	0	0	
warehouse_team	0	0	

@dmagliola

They obviously share this idea with the other teams, and ask them to do the same.

And guess what.. Not everything that the logistics team does has the same urgency...



critical
crm_sync
db_maintenance
default
event_source
high
inventory_team
inventory_team_critical
inventory_team_high
inventory_team_low

logistics_team
logistics_team_critical
logistics_team_high
low
mailers
metrics
notifications
notifications_team
notifications_team_high
notifications_team_low

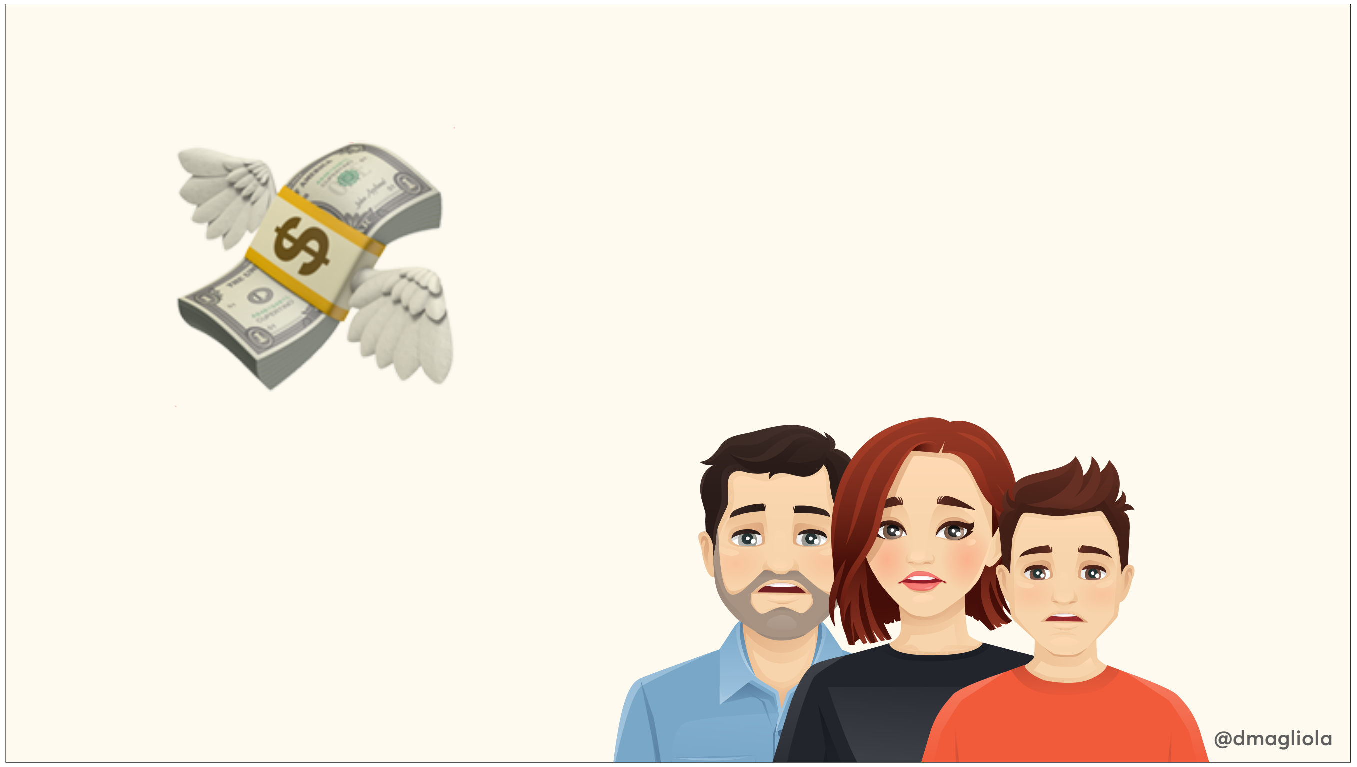
order_lifecycle
pricing
purchasing_team
purchasing_team_critical
purchasing_team_high
purchasing_team_low
sales
sales
sales_
survey

warehouse_team
warehouse_team_checks
warehouse_team_critical
warehouse_team_high

@dmagliola

So 3 months later we end up with "logistics_high" and "purchasing_low"… and of course, our original "high" and "low" and "mailers" are still there, because we have hundreds of different jobs in there and no one knows what they do or where they should go.

This is starting to get hairy…



And also it's a good thing that we have VC money behind this company, but even then, at some point someone does notice that we're spending WAY more in servers than we should, and they start asking questions. Turns out, we have 60 queues now, which means 60 servers. Most of these servers are not doing anything the vast majority of the time… But someone has to serve those queues.

So an obvious decision is made: we can configure processes to work several queues at once, so we will group some of them together and reduce the server count.

And guess what

Mad
Things that frustrated you in the last i...
Other teams' jobs screwed us over again :(
We missed delivery deadlines again with the queuepocalypse 😡. We're going to have some angry customer calls...
The Purchasing fo
queues for th
month!!! 😡
Our queue
to f
ng
crew d up
u to figure out how to
ur queues saner. We can't
doing this.
ad
you in the l...
rting
on
Glad
Things that excited you in the last iter...
Kudos to Sally fixing our queues! 🙌

@dmagliola

@dmagliola

Now, you may be asking why I'm telling you this clearly facetious story about people making very obvious mistakes, but the truth is, this isn't fiction at all. I've renamed the queues and the teams to protect the innocent, but I've seen this exact story develop in pretty much exactly this way in every company i've worked in. Hell, i've been half of these characters proposing things that obviously weren't going to work! This was me with the bad ideas.

And I’ve seen this enough that I think this is a common progression that many companies go through. You may have seen this too, and the rest of this talk will hopefully help with these issues. Or if you haven’t gone down this path yet... Hopefully it can prevent you some headaches...

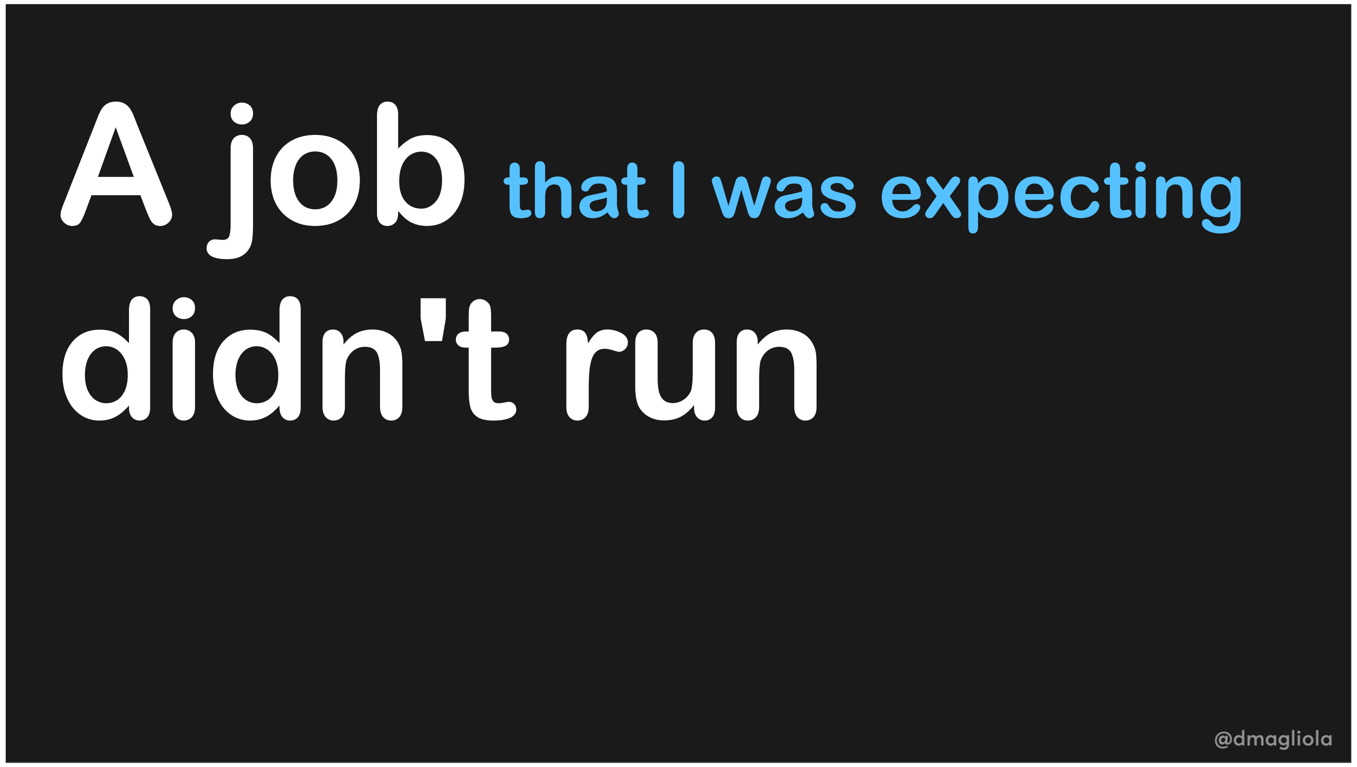
The reason I think this is a common progression, by the way, is not that these characters are bad engineers. The thing is, when faced with these challenges, these steps do seem like the obvious solution. They make sense! The problem is that queues tend to have interactions and behaviours that are very hard to reason about, so when we propose one of these changes to fix a problem, it’s really hard to predict the next problem that they will cause.

So how do we get out of this?

Well, we first need to think about what actual problem we’re trying to solve when we start making new queues. Because the problem is deceptive, and we tend to attack the wrong issue. The problem isn’t queues. It’s not jobs or priorities or threads.

The problem is the language we use to talk about this. It... often is.

If you think about it, you have jobs running in the background all the time, and no one notices. If someone notices, that’s when you have a problem. But what is it that people notice?

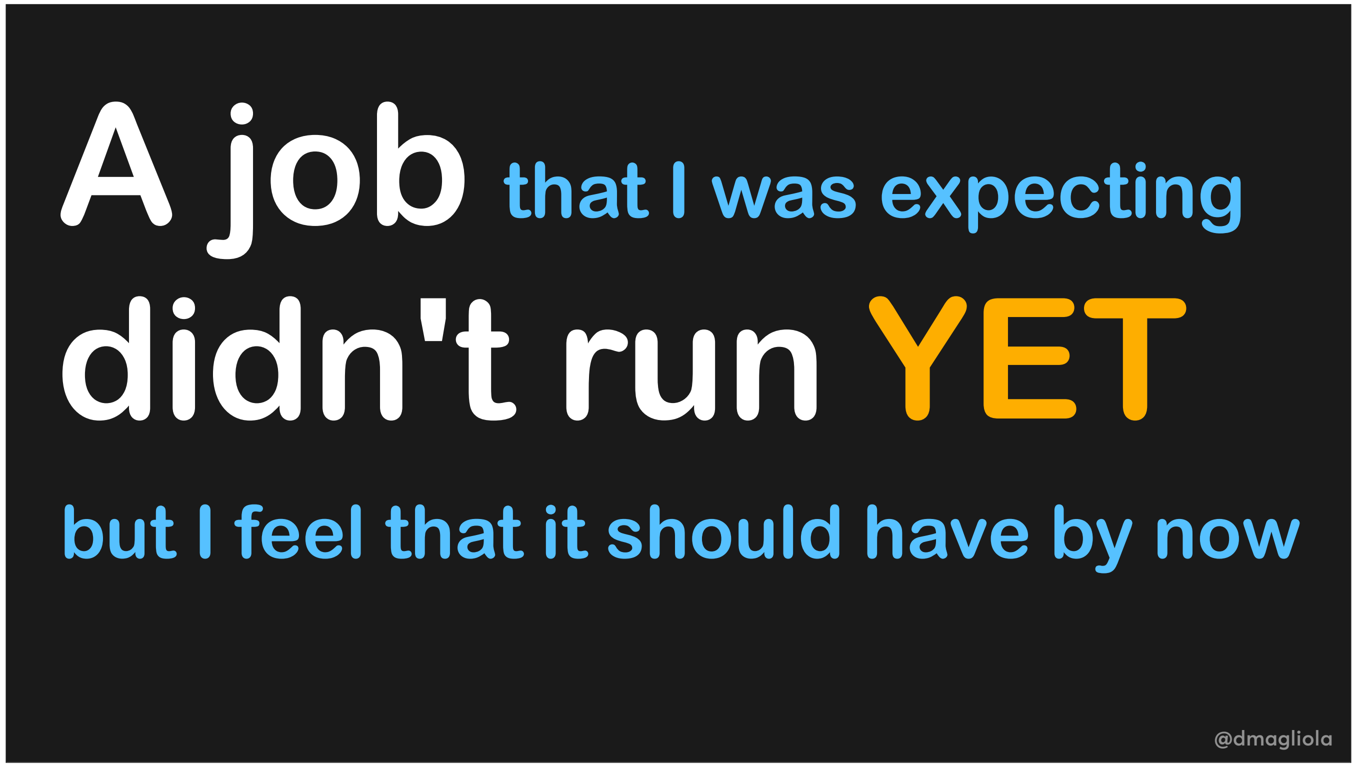


A job that I was expecting didn't run

@dmagliola

"A job didn't run"

Which generally actually means



A job that I was expecting

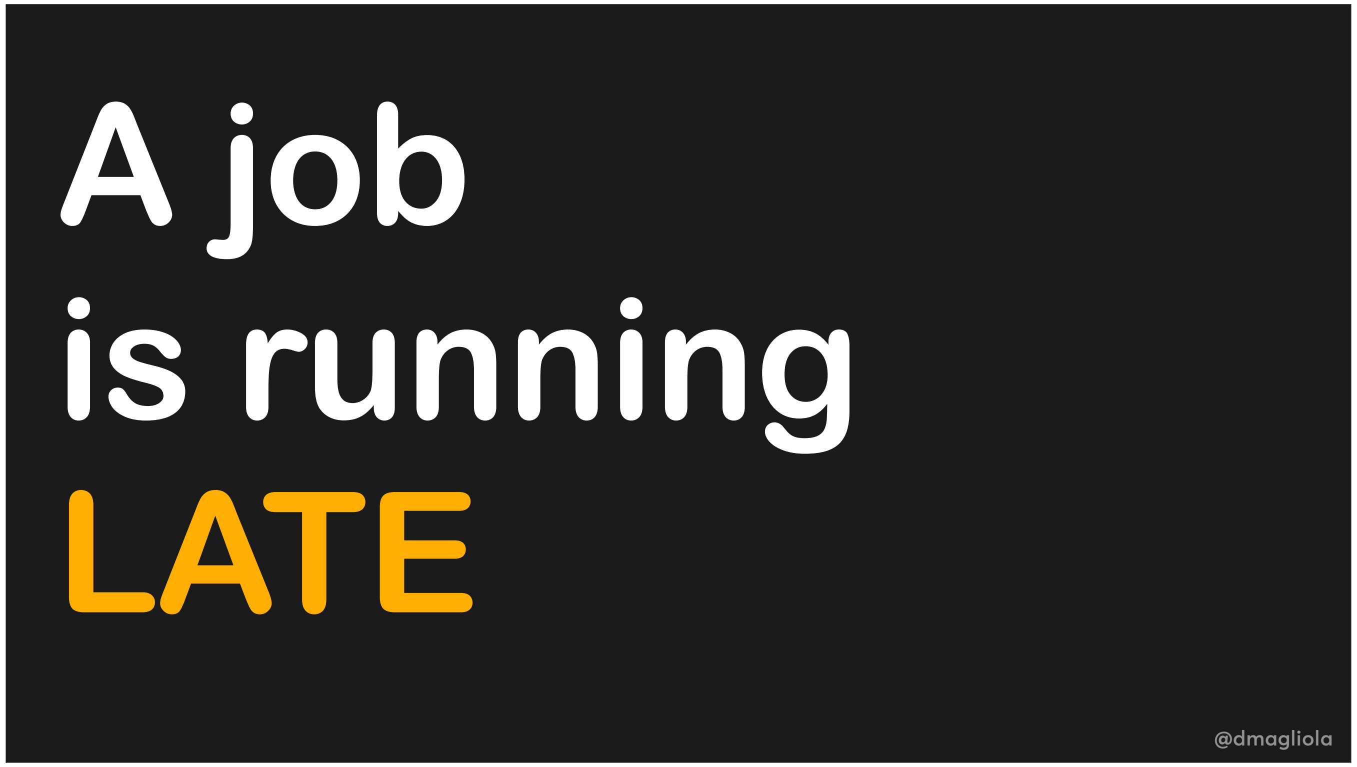
didn't run YET

but I feel that it should have by now

@dmagliola

"It didn't run yet, but I feel like it should have by now"

Or put another way



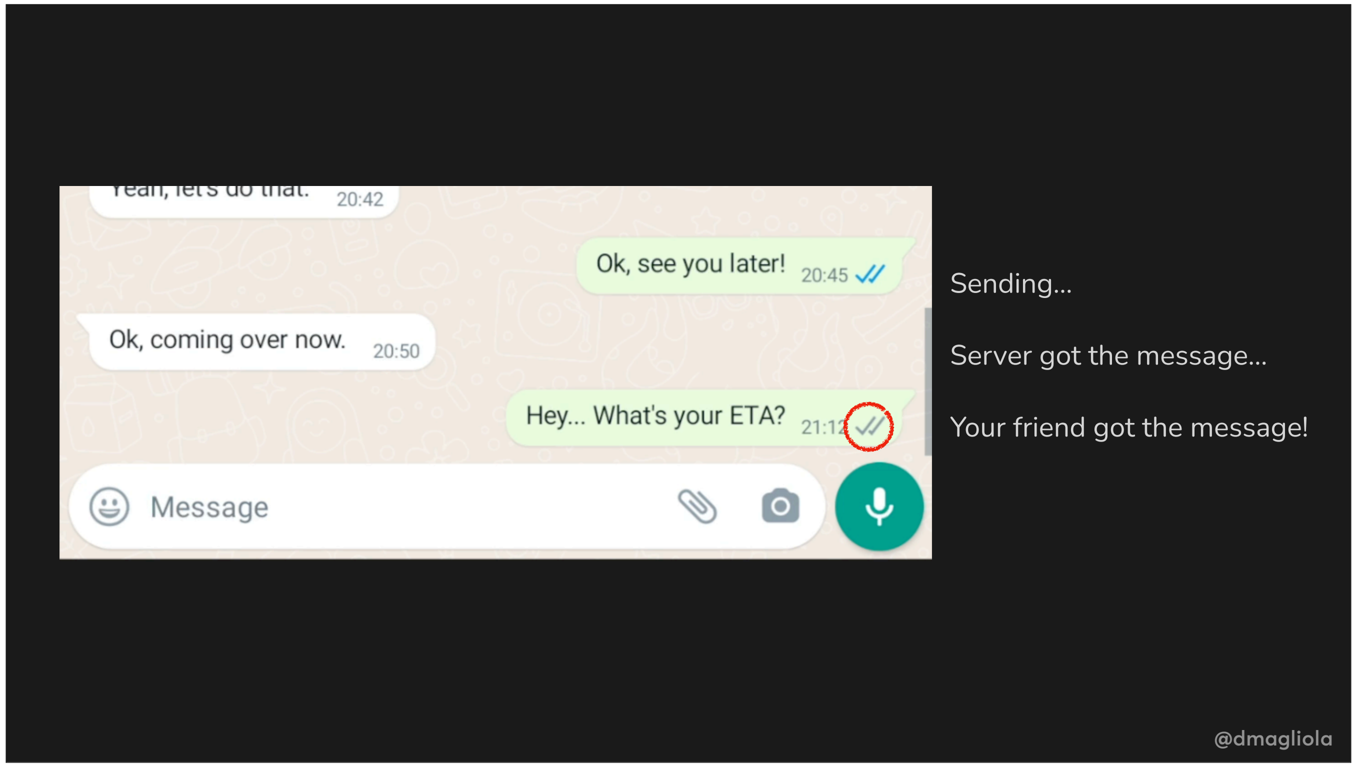
A job
is running
LATE

@dmagliola

"A job is running LATE"

And here's where the trouble starts. Because what does "late" mean here?

Many, many jobs every day don't get to start immediately once they get enqueued, they tend to wait in the queue a bit. And no one notices. Some may wait for minutes, and no one notices. Some others, it's very obvious if they're late.



Yeah, let's do that. 20:42

Ok, see you later! 20:45

Ok, coming over now. 20:50

Hey... What's your ETA? 21:12

Sending...

Server got the message...

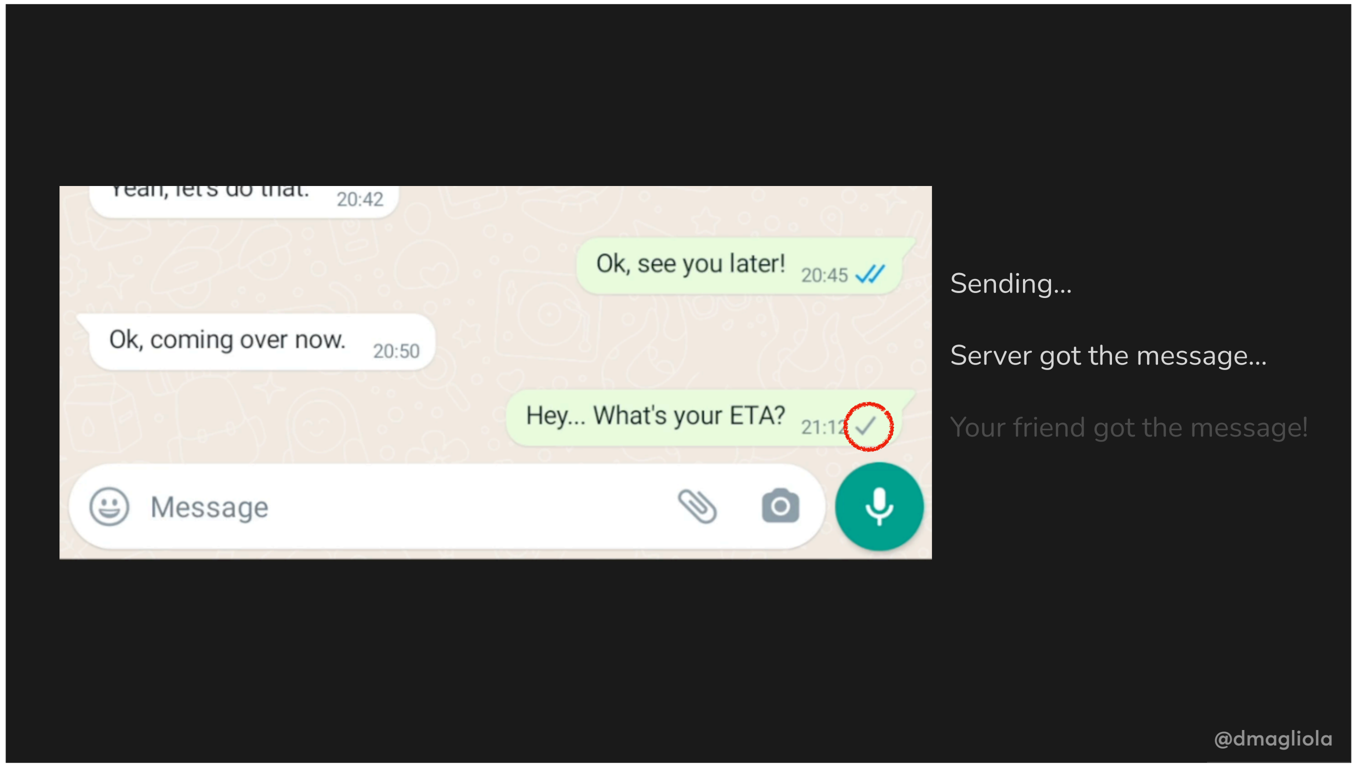
Your friend got the message!

@dmagliola

For example, you probably know this one:
You send a message to a friend, it tells you when it reached their phone.

That took a second, and you didn't even notice.

Now let's try a different scenario...



Yeah, let's do that. 20:42

Ok, see you later! 20:45 ✓✓

Ok, coming over now. 20:50

Hey... What's your ETA? 21:12 ✓

Sending...

Server got the message...

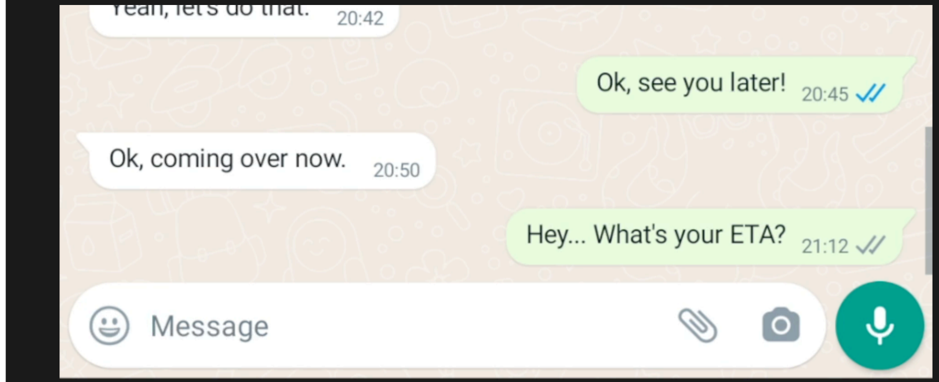
Your friend got the message!

@dmagliola

How anxious are you getting right now?

Is he in a tunnel? Did his phone run out of battery?

Should I be worried? Is he dead?



Yeah, let's do that. 20:42

Ok, see you later! 20:45 ✓✓

Ok, coming over now. 20:50

Hey... What's your ETA? 21:12 ✓✓

Sending...

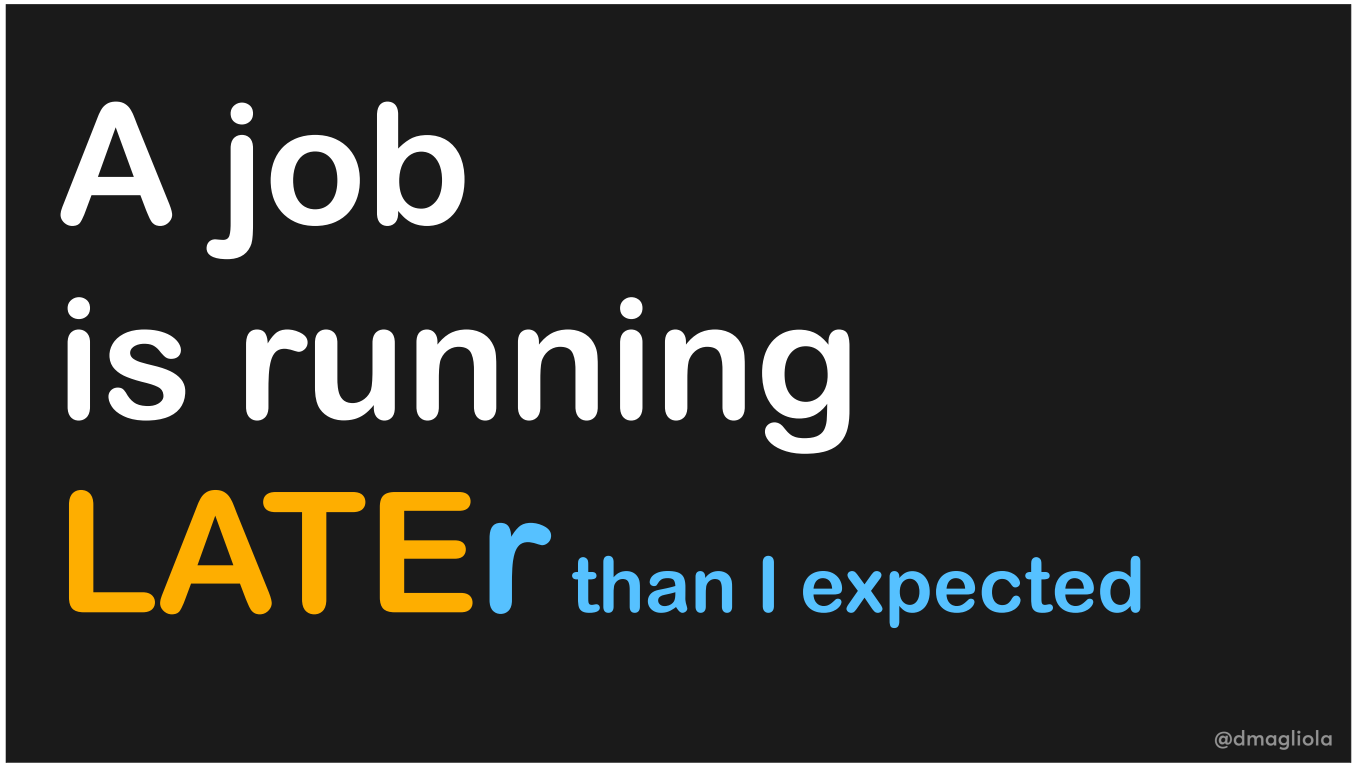
Server got the message...

Your friend got the message!

@dmagliola

Oh, thank god, he's ok.

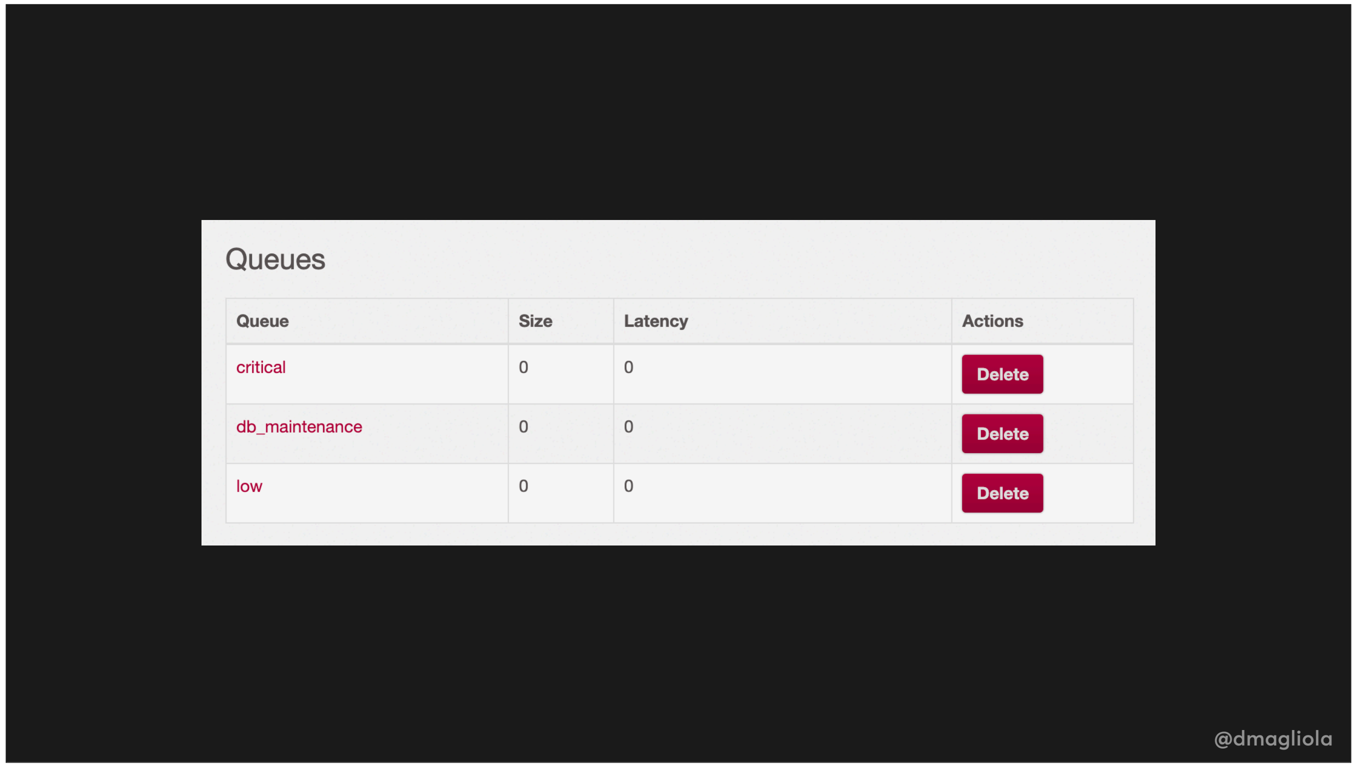
Now THAT job was late!



A job
is running
LATEr than I expected

@dmagliola

So here's the issue. There's an expectation you have for your jobs, of how long it'll be until they run. But that expectation is implicit. It's one of those "I'll know it when i'll see it" kind of things, where you know if it's running late, but if I asked you when you think it's late... You probably can't tell me.

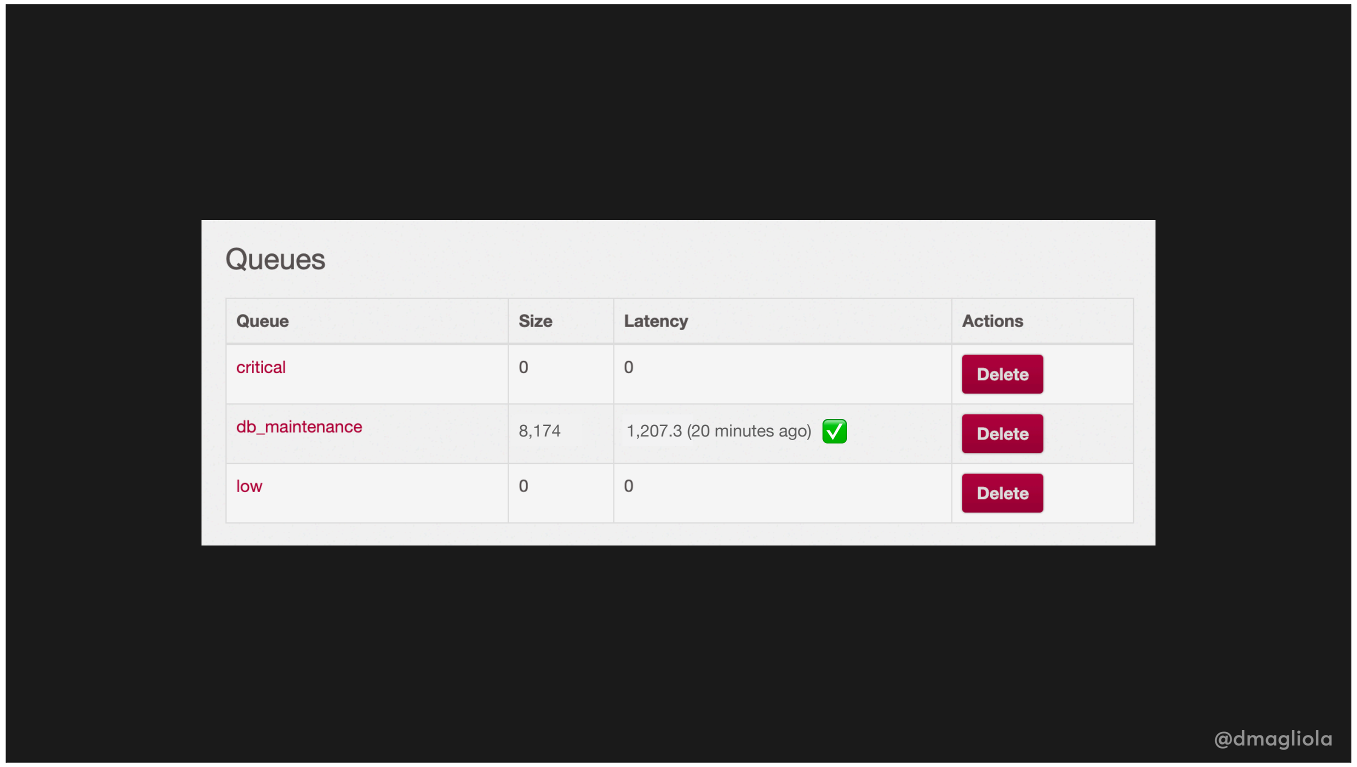


Queues

Queue	Size	Latency	Actions
critical	0	0	Delete
db_maintenance	0	0	Delete
low	0	0	Delete

@dmagliola

Like, you got the "database maintenance" queue, where you run a "VACUUM ANALYZE" on every table every night because, there was a problem with statistics once 2 years ago and we will never trust the database again….

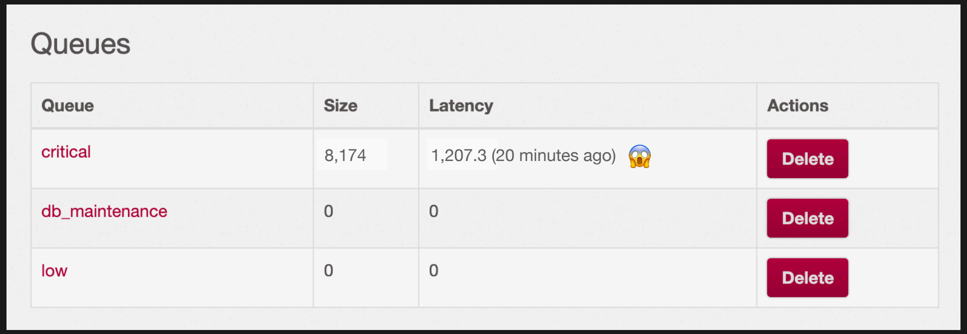


Queues

Queue	Size	Latency	Actions
critical	0	0	Delete
db_maintenance	8,174	1,207.3 (20 minutes ago) ✅	Delete
low	0	0	Delete

@dmagliola

That's running 20 minutes behind today… Is that a problem? Almost certainly not!

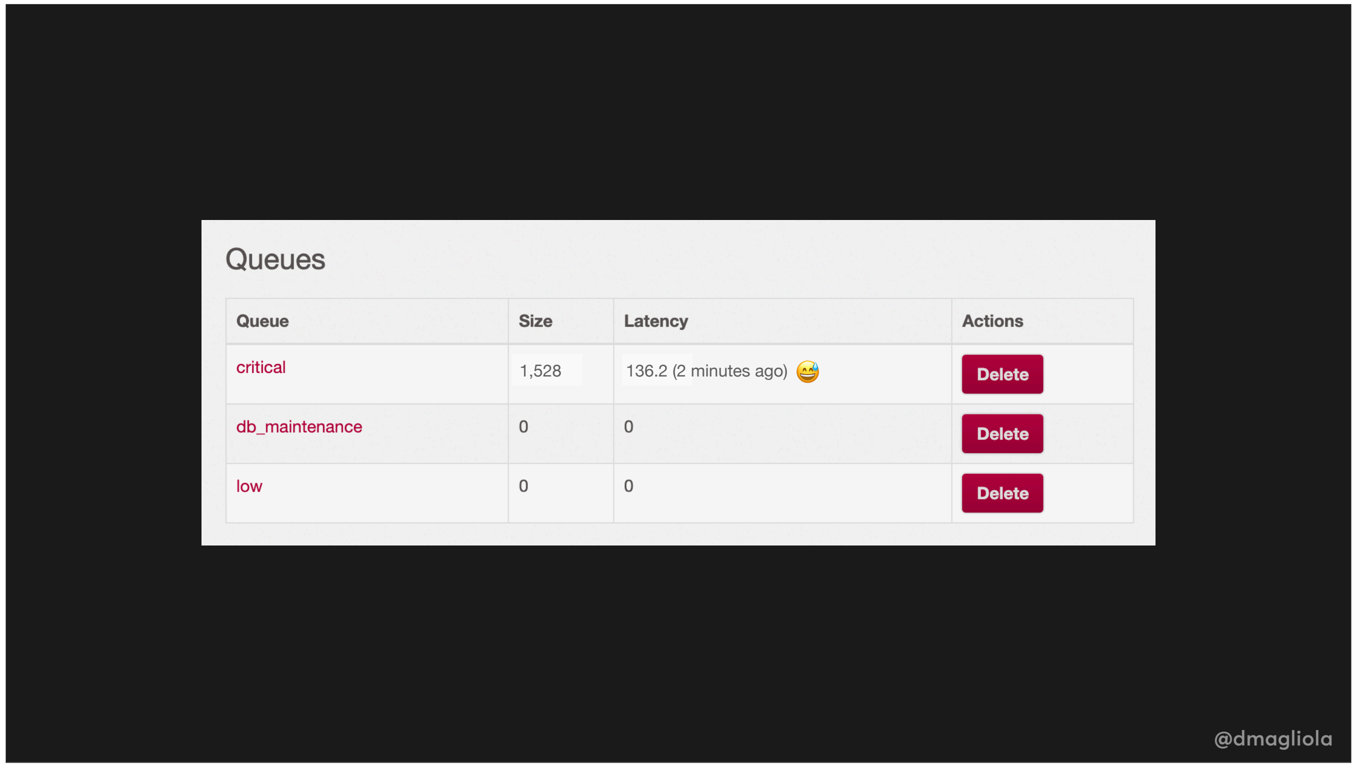


Queues

Queue	Size	Latency	Actions
critical	8,174	1,207.3 (20 minutes ago) 😱	Delete
db_maintenance	0	0	Delete
low	0	0	Delete

@dmagliola

What if your "critical" queue was 20 minutes behind? Almost definitely a problem!

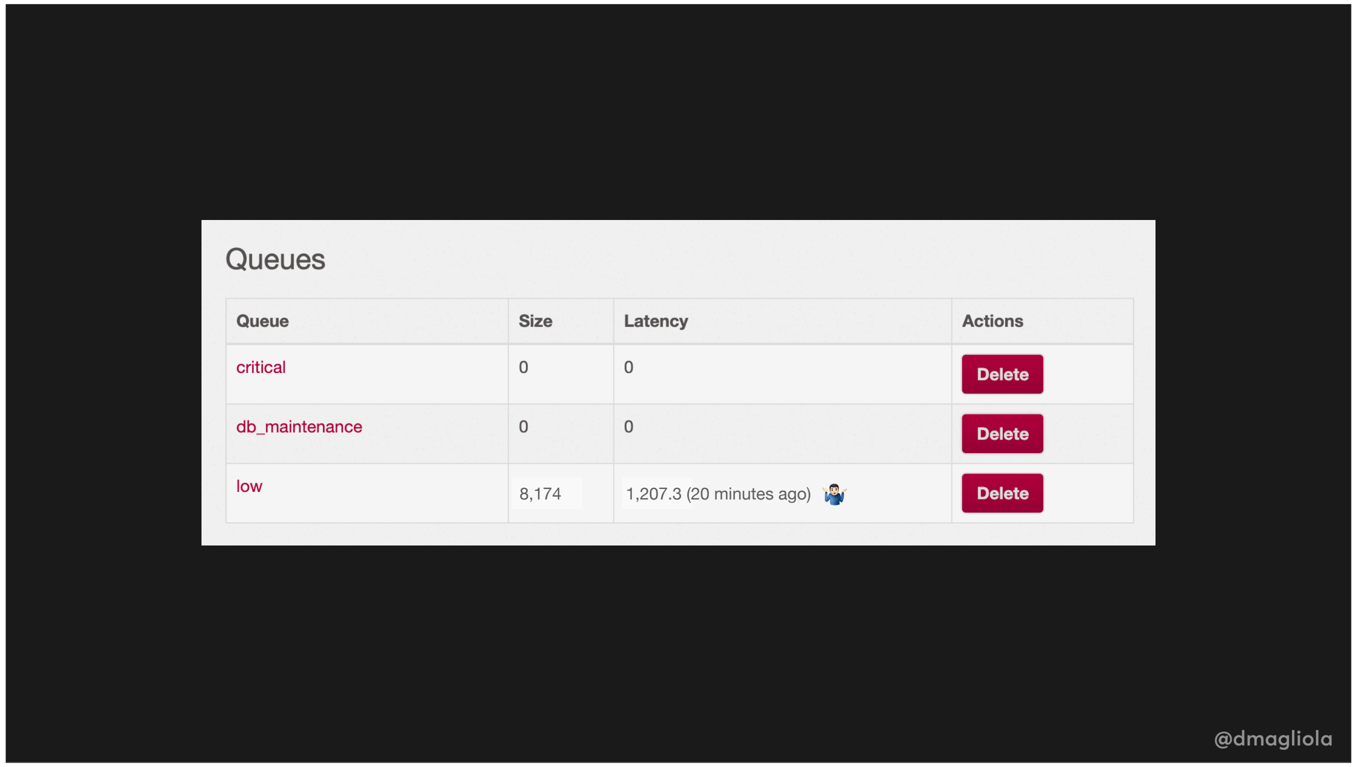


Queues

Queue	Size	Latency	Actions
critical	1,528	136.2 (2 minutes ago) 😅	Delete
db_maintenance	0	0	Delete
low	0	0	Delete

@dmagliola

What if it's 2 minutes behind though? Is that ok?



Queues

Queue	Size	Latency	Actions
critical	0	0	Delete
db_maintenance	0	0	Delete
low	8,174	1,207.3 (20 minutes ago) 🤷	Delete

@dmagliola

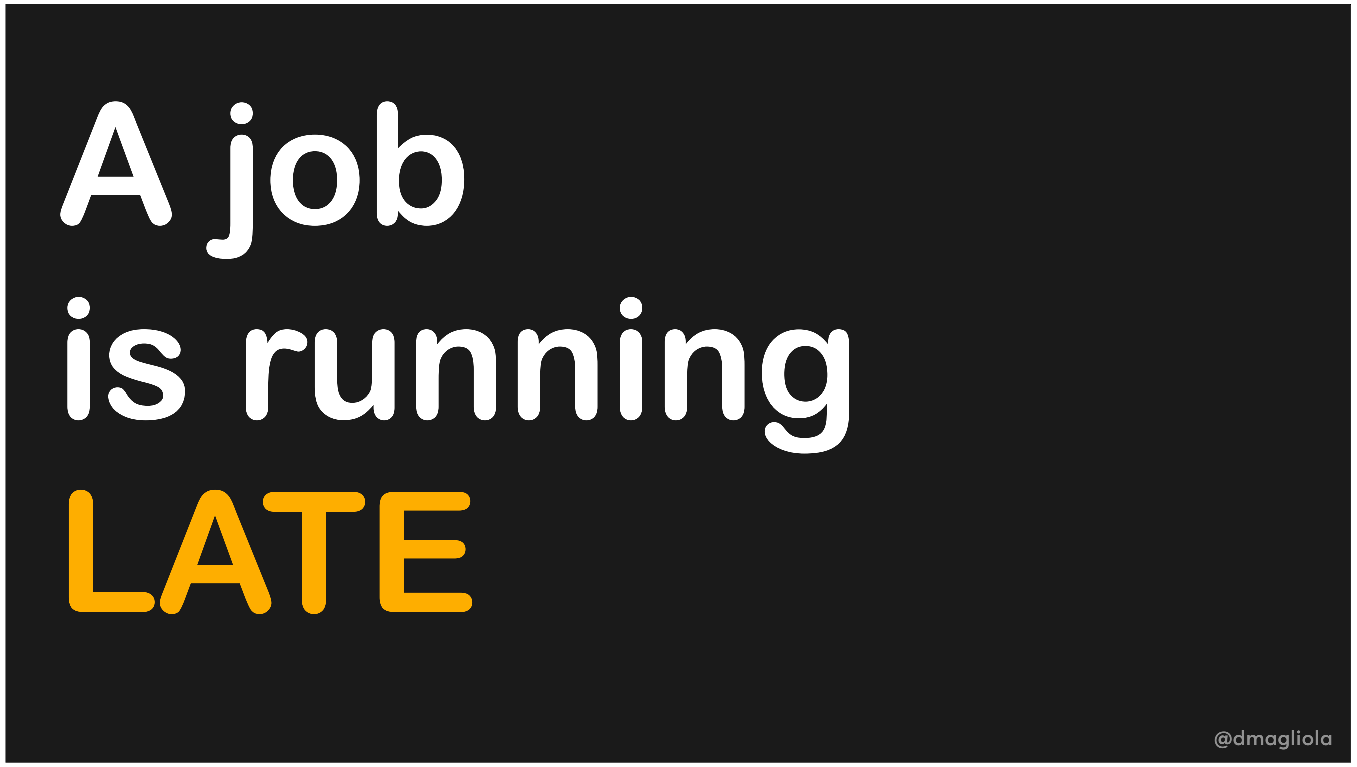
What if the "low priority" queue is 20 minutes behind? Is that bad? [shrugs]

The problem is the language that we use to define these queues and the jobs we put in them. We give them these vague names, and it gets us in trouble.

@dmagliola

″High priority” or ″low priority” gives you some relative idea, ″this is more important or urgent than that”, but it gives you no insight as to whether your queues are healthy or not at any given point, until someone is missing some job and starts shouting.

And as an engineer, it also gives you no indication as to where you should put things. So what do you do when you’re writing a new job? You try to look around the queues for other jobs that seem similar to yours, and follow that. Or you just guess? You listen to your gut, about how ″important” this job ″feels”.



A job is running LATE

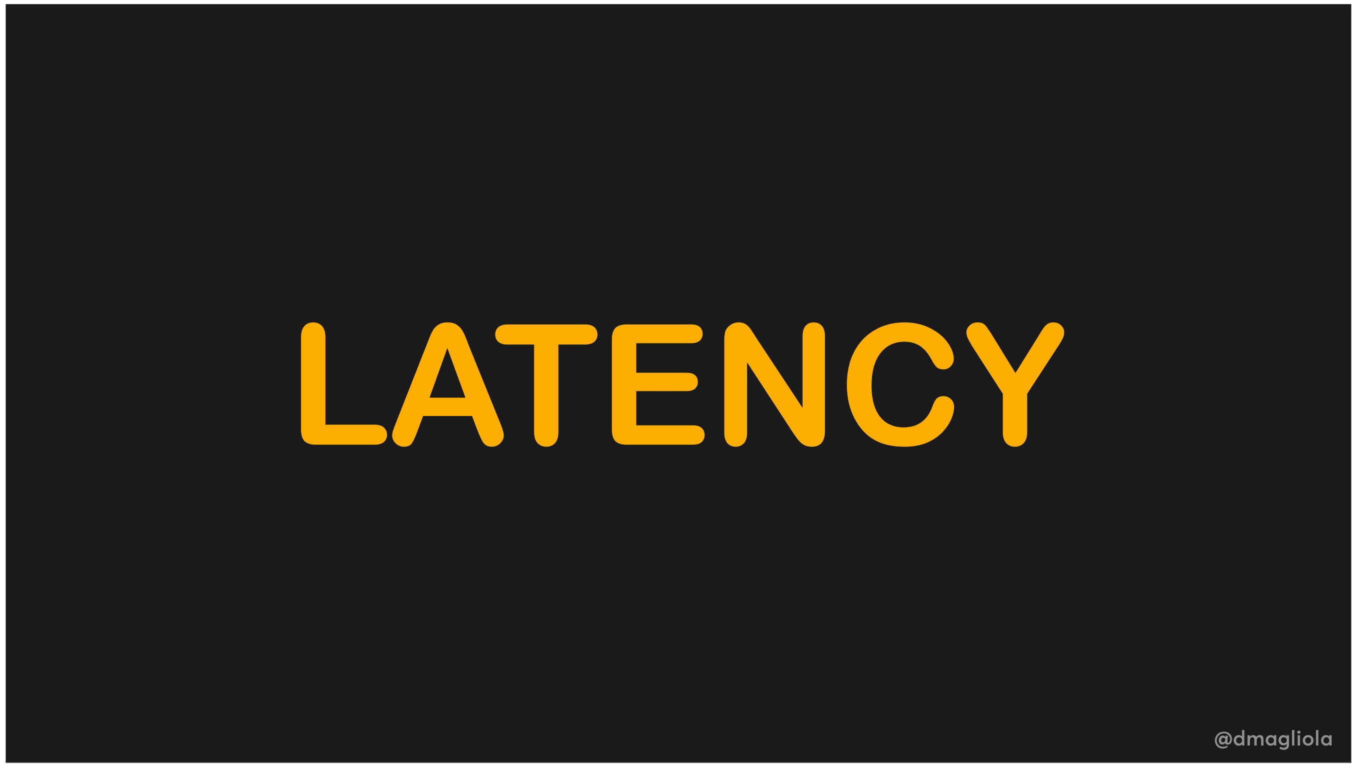
@dmagliola

The problem is the words we use. "High and low priority" doesn't give you anything very useful. They're very vague names.

But the good thing is the answer is also in this language…

Because what we care about is not priority, or "importance". It's not the "type of jobs" or "which team owns them"

THIS is what we care about. A job is running LATE.



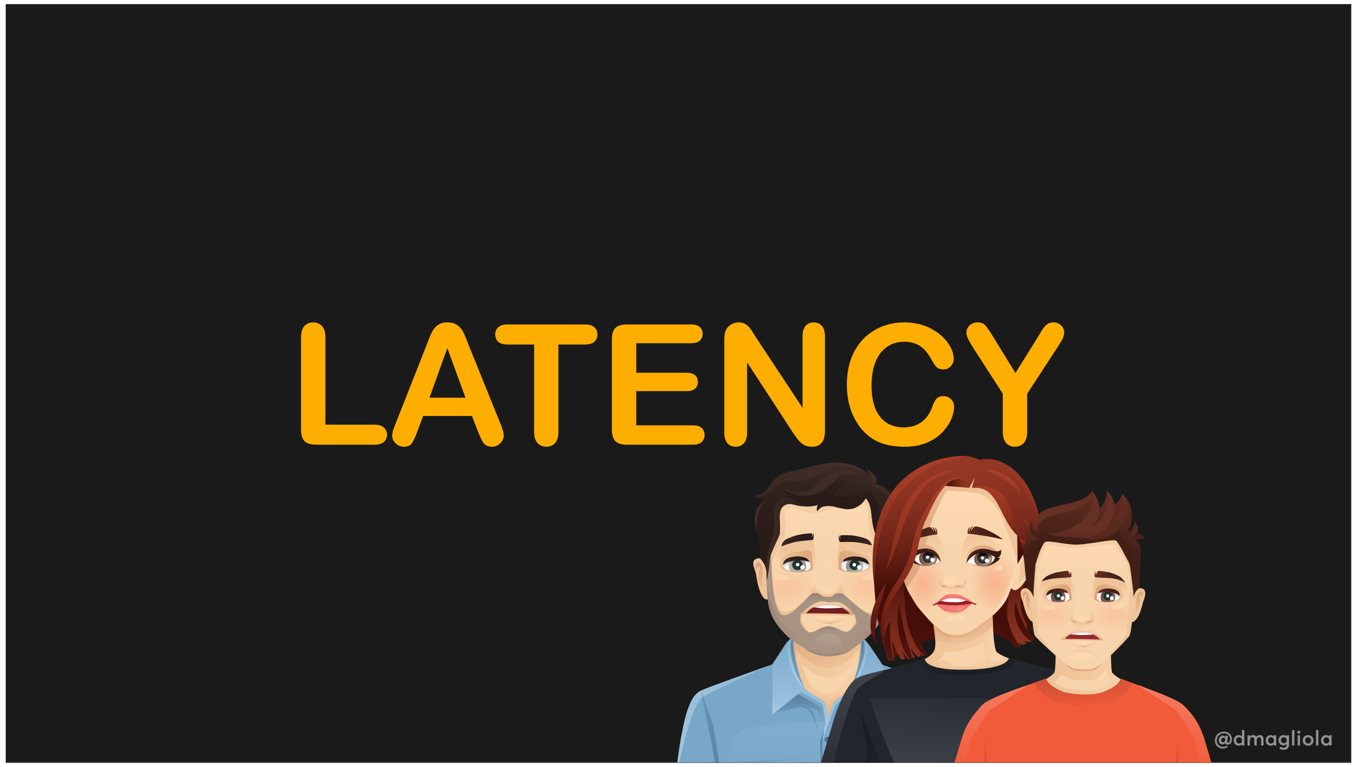
LATENCY

@dmagliola

The one thing we care about, is LATENCY.

And that’s how we should structure our queues.

We should assign jobs to queues looking ONLY at how much latency they can tolerate, before it feels like they are "late”.



LATENCY

@dmagliola

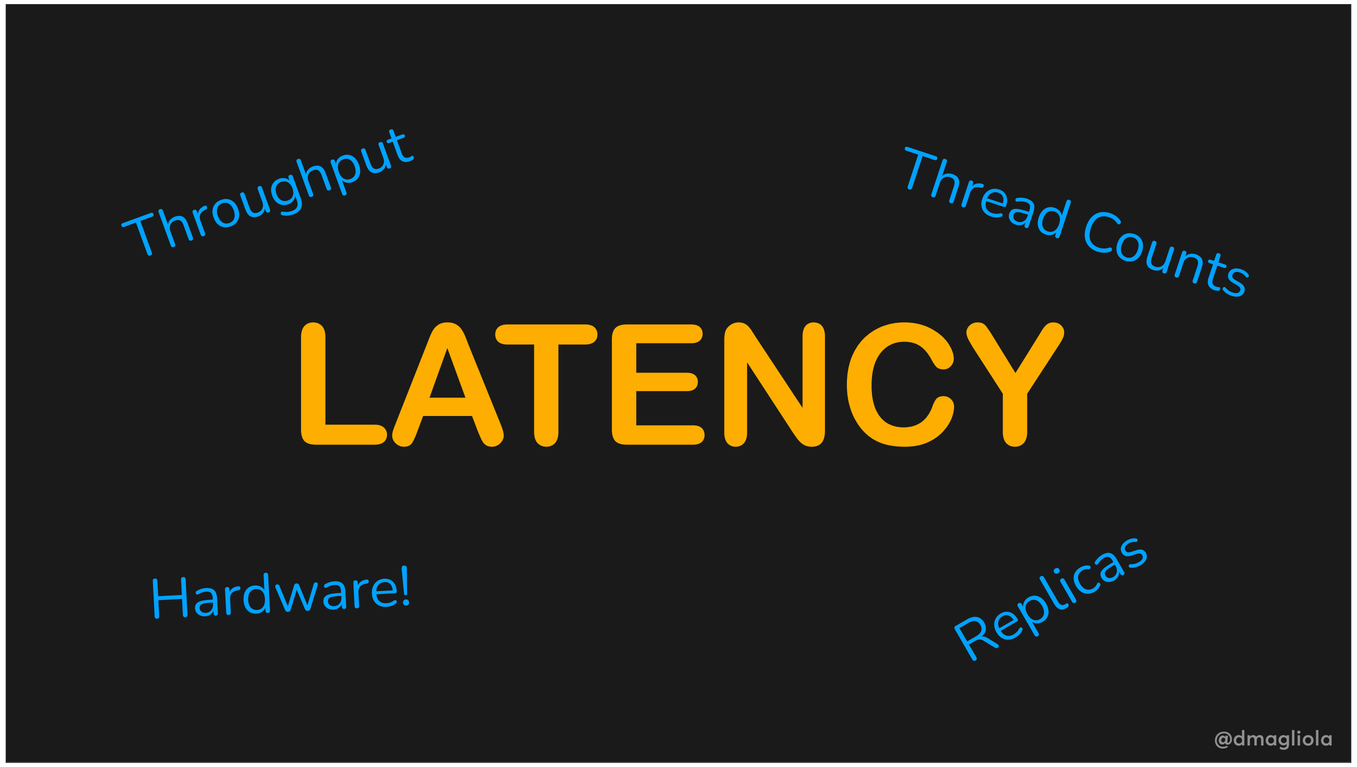
And this is the problem our friends are having. They are trying to isolate workloads from each other, but their queues were not designed around the one thing we care about.

A job
is running
LATE

@dmagliola

Remember the symptom is always "this thing didn't happen as quickly as it needed to".

The ONE thing we care about is latency. This sounds simplistic, but i'm dying on this hill. Latency is the ONLY thing you really care about for your queues. Everything else is gravy.



Throughput

Thread Counts

LATENCY

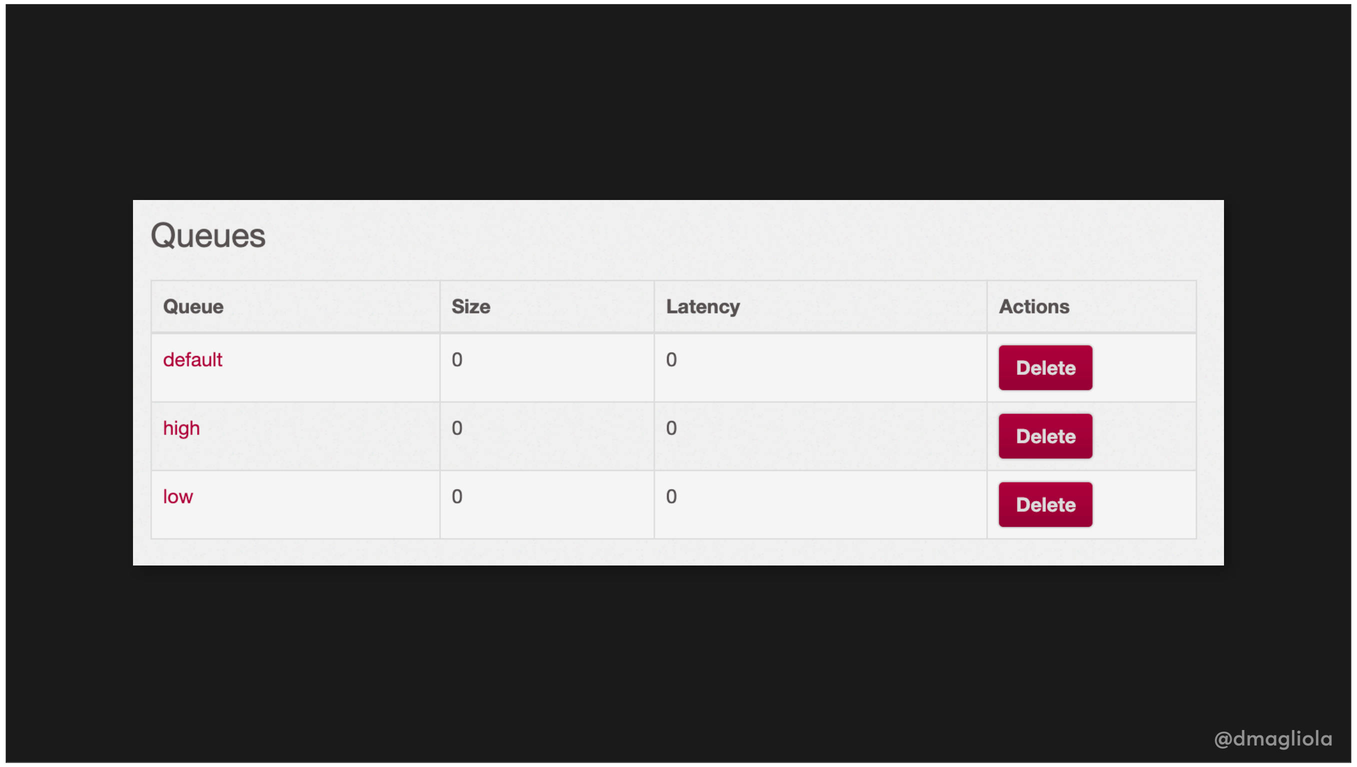
Hardware!

Replicas

@dmagliola

Now, there are other things you'll care about in order to achieve that latency: Throughput, thread counts, hardware resources, etc, etc. But those are a means to an end. You don't really care about throughput. What you care about is that you put a job in the queue and it gets done "soon enough". Throughput is just how you cope with the volume to make sure that happens.

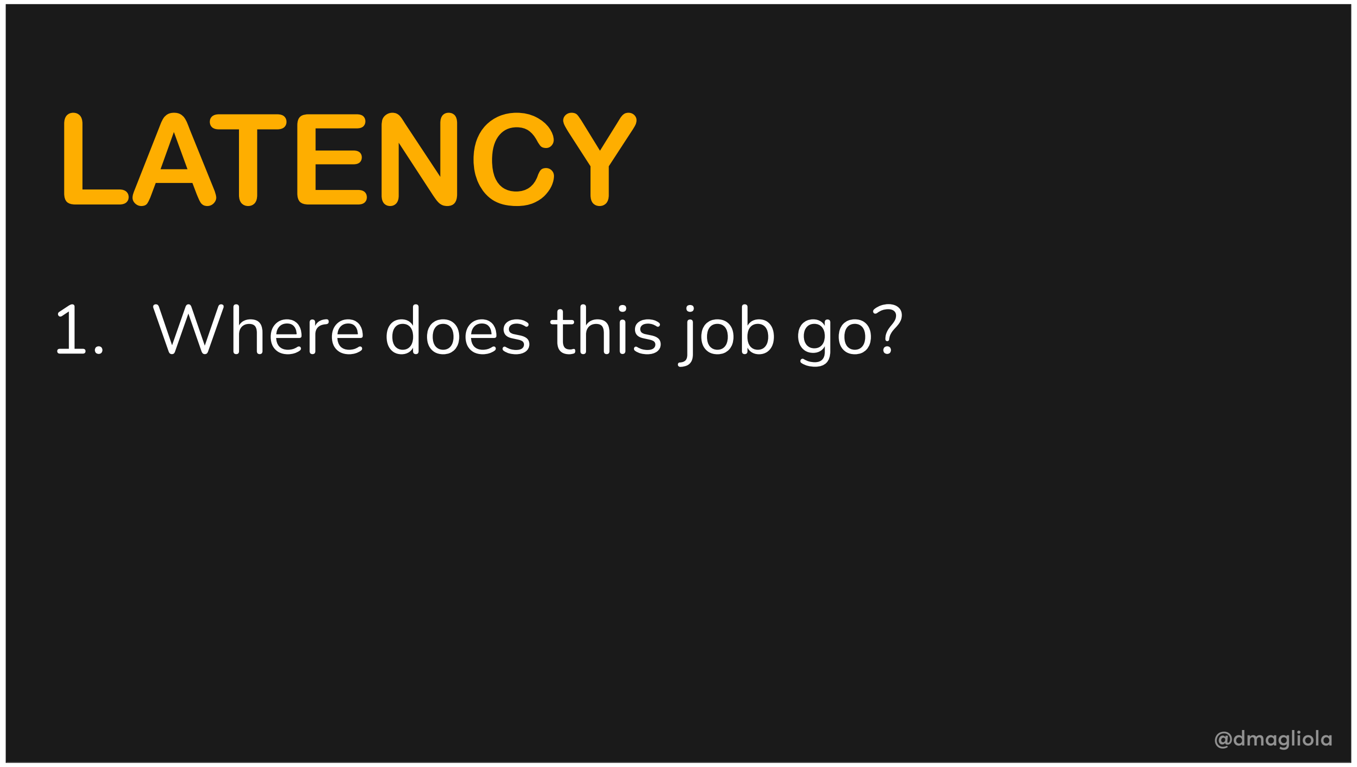
So separating your queues based on purpose, or per team, they are just roundabout ways to try and get that latency under control, but the issue they all have is they don't really center on that latency. They are "roundabout" ways, and we want to attack the problem directly.



Queues

Queue	Size	Latency	Actions
default	0	0	Delete
high	0	0	Delete
low	0	0	Delete

Now, to be fair, the very first approach WAS doing roughly the right thing. The "high" and "low" queues were trying to specify the latency tolerance for those jobs. The problem is that while that decision comes from the right instinct, having "high" and "low" and "super really extra high" is extremely ambiguous.



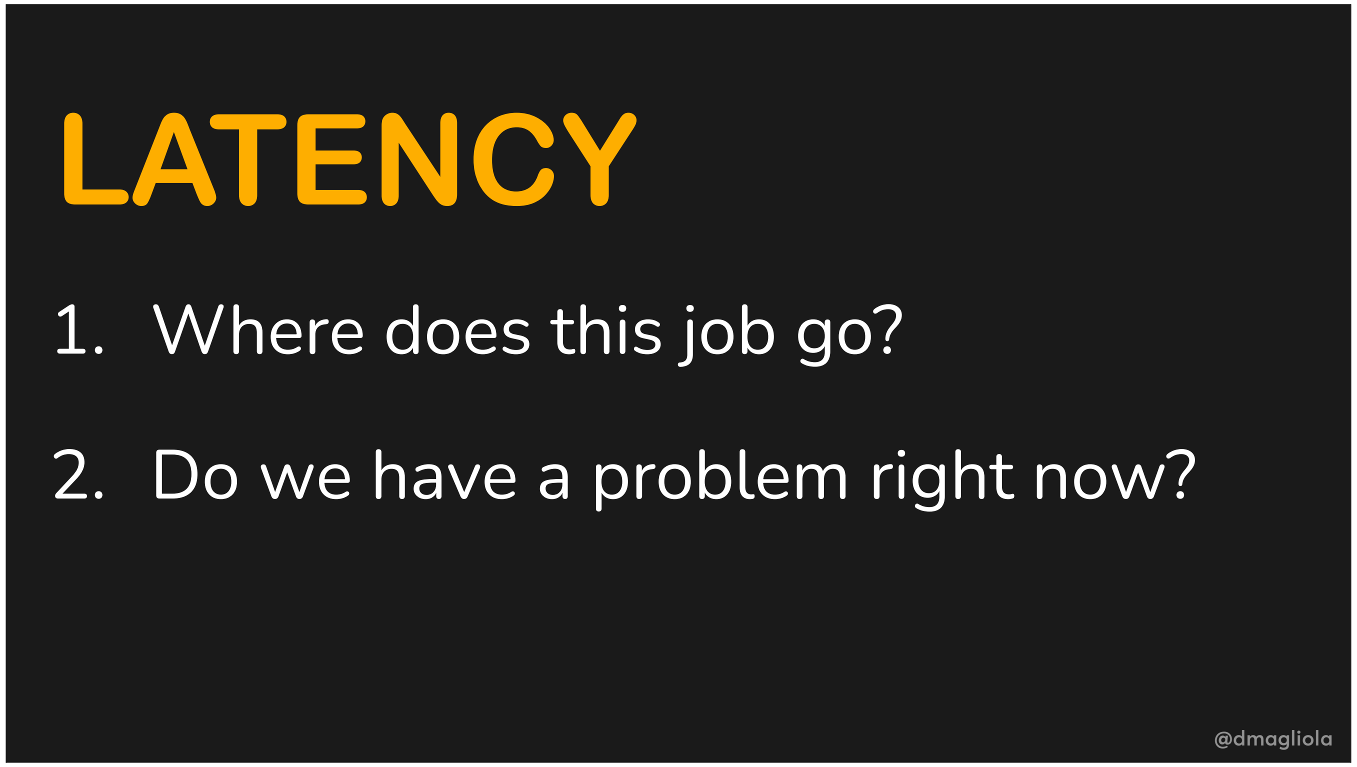
LATENCY

1. Where does this job go?

@dmagliola

For 2 reasons:

First, I have this new job that I just wrote, and it's OK if it takes 1 minute to run, but not 10 minutes. Where does that go? Is that "high"? Is that "critical"?

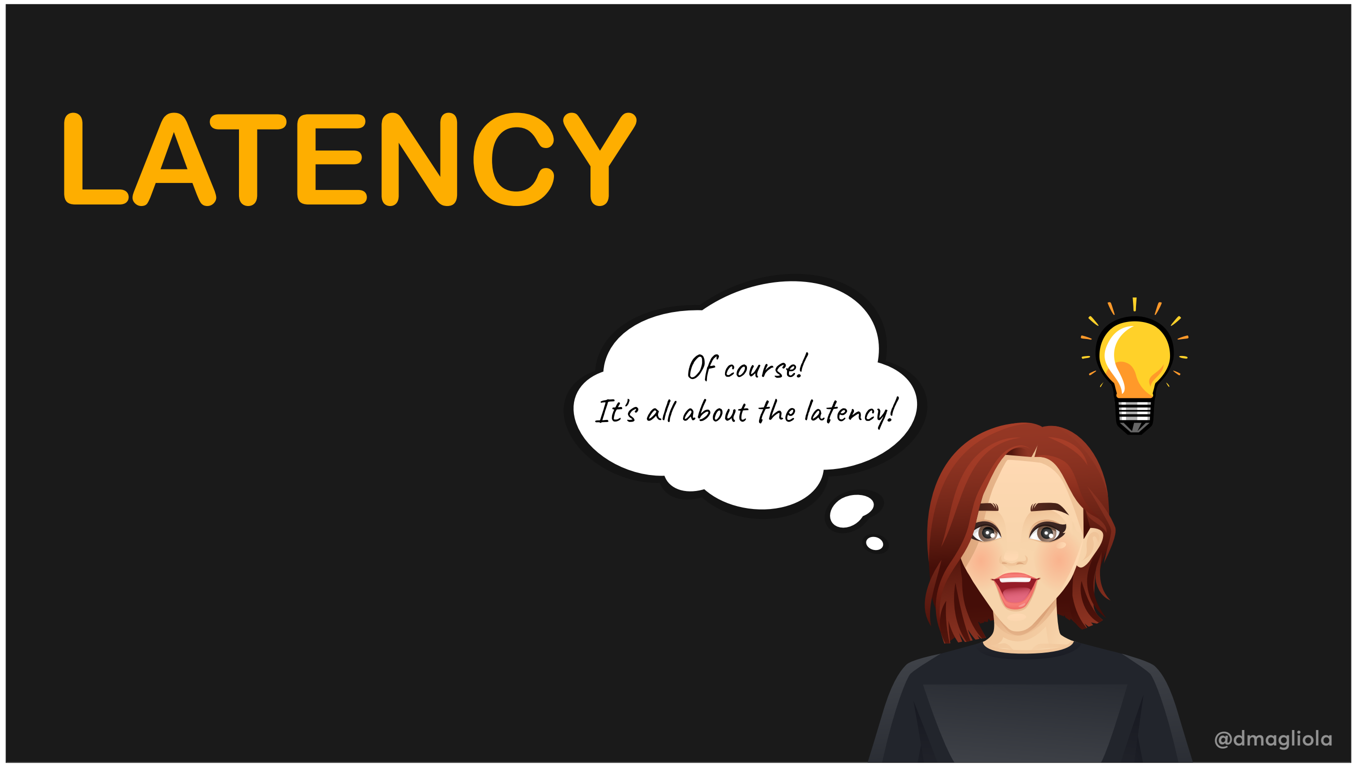


LATENCY

1. Where does this job go?
2. Do we have a problem right now?

@dmagliola

Second, When things go wrong, we want to find out there's a problem before our customers start calling us... So we want to set up some alerts... But at what point do we alert? How much latency is too much for the "high" queue? What about the "low" queue?



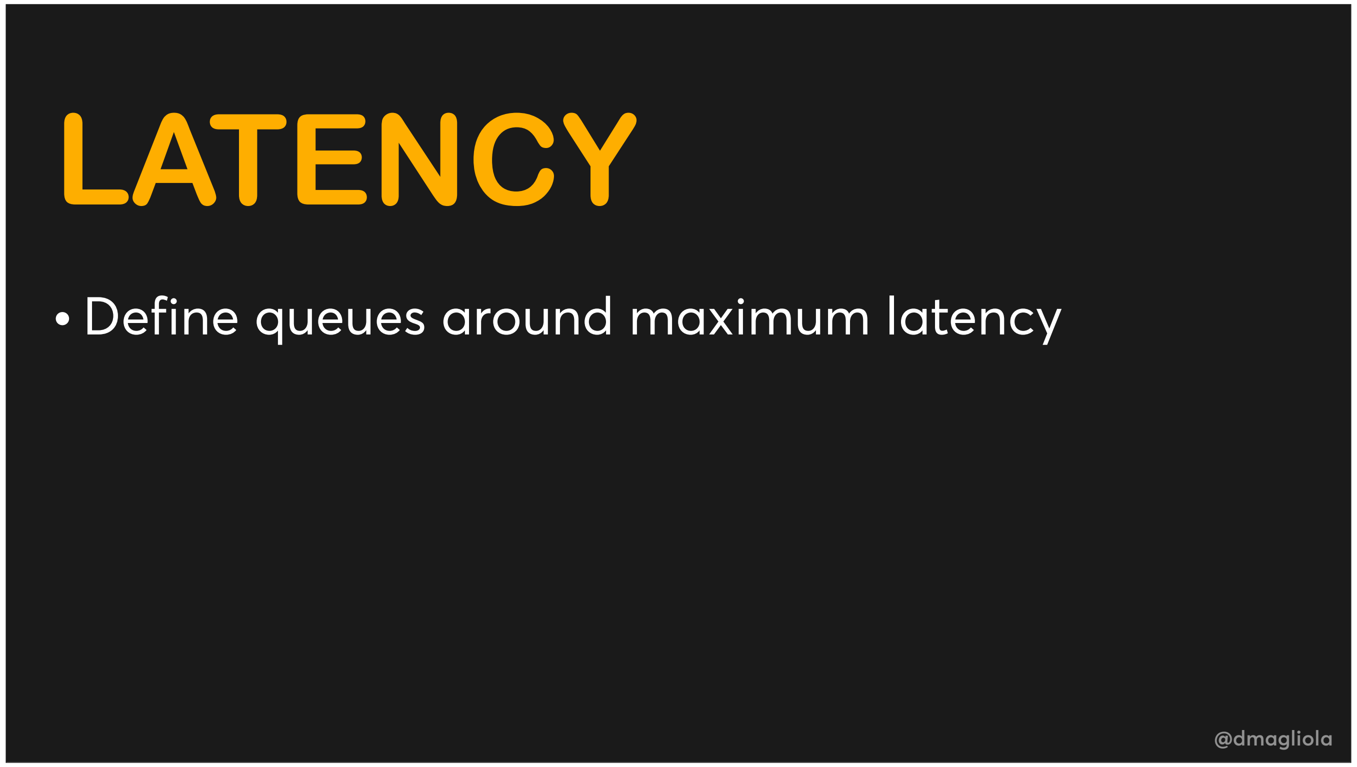
LATENCY

@dmagliola

And here's where our hero, Sally, comes in with her brilliant idea:

We will organize our queues around different allowed Latencies, with clear guarantees and strict constraints.

Now, let's unpack that because there's a lot there:



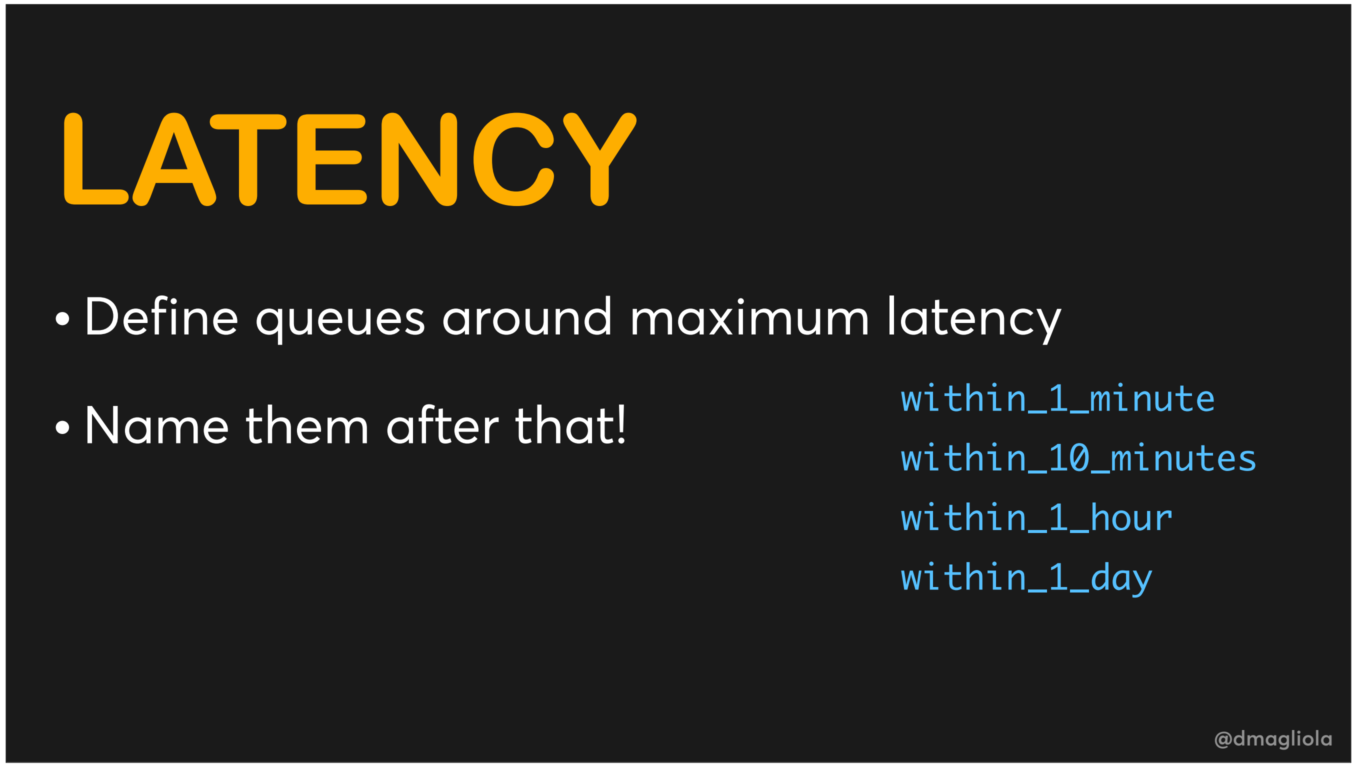
LATENCY

- Define queues around maximum latency

@dmagliola

We will define queues around how much latency a job in those queues might experience. This is the maximum latency we will allow in these queues. Now, "allow" may sound naive, but you'll see what I mean in a second.

What's important is that you know that if you put a job in those queues, it'll start running in no more time than this advertised maximum latency.



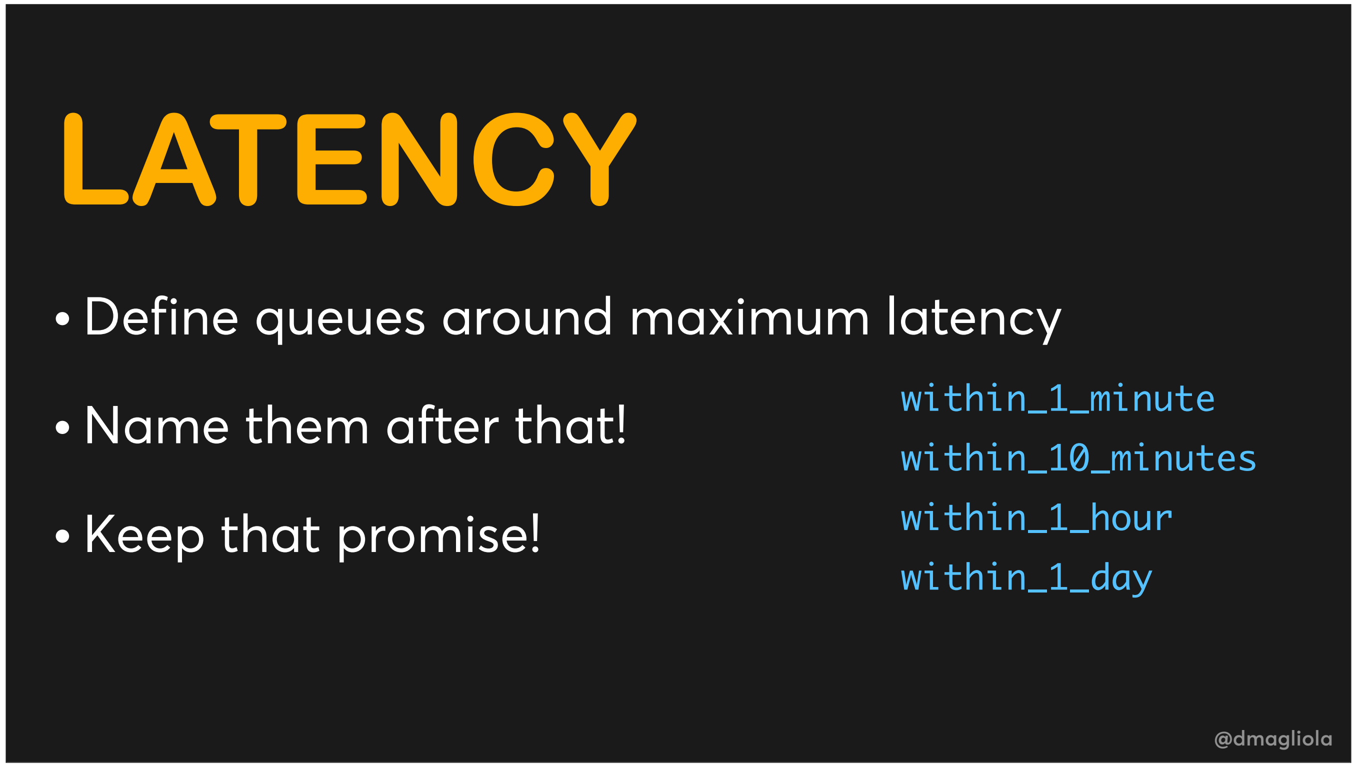
LATENCY

- Define queues around maximum latency
- Name them after that!

```
within_1_minute
within_10_minutes
within_1_hour
within_1_day
```

@dmagliola

We will name these queues based on that, clearly and unambiguously. Anyone that's looking at a queue needs to know exactly what they can expect from it.



LATENCY

- Define queues around maximum latency
- Name them after that!
- Keep that promise!

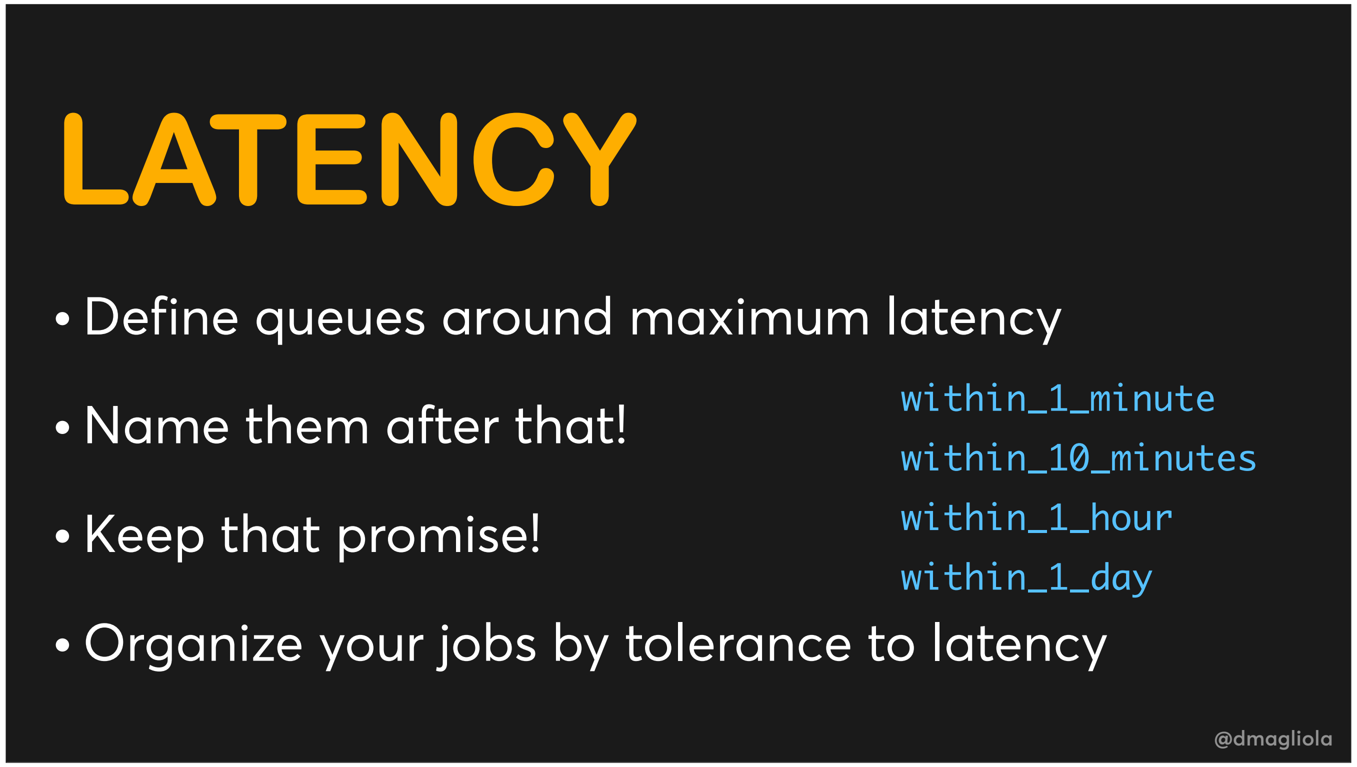
```
within_1_minute
within_10_minutes
within_1_hour
within_1_day
```

@dmagliola

Now here's the important part… That name?

That's a GUARANTEE. It's a CONTRACT. If you put a job in one of these queues, it will start running before that much time elapses.

If a queue's latency goes over what it says in the name… Alerts go off. Fire departments get called. On-call engineers get woken up, and we fix it. That name is a promise, and you can count on it. And that part is the key.



LATENCY

- Define queues around maximum latency
- Name them after that!
- Keep that promise!
- Organize your jobs by tolerance to latency

```
within_1_minute
within_10_minutes
within_1_hour
within_1_day
```

@dmagliola

So as an engineer, whenever you're writing a job, you will choose what queue to put it in based on how much latency it can tolerate. And what that means is.... What is the threshold where, if your job doesn't run by that time, someone will notice and think that the job is late, or it didn't run, or the system is broken. This makes it much less ambiguous where to put things.

What happens if this job runs an hour later? Will anyone notice? No? Great, you go to the "within1hour" queue. 1 hour is too much? How about 10 minutes? Is it ok if it waits for up to 10 minutes? Cool. If not, "within1minute".

Now, granted, finding this threshold will be easier for some jobs than others, but you want to try and map out what would be the consequences of running at different levels of "late", and figure out what would go wrong.

And... Be realistic, there's always a temptation to think everything must happen now, but remember that it can't all happen instantly, and we need to be thorough in figuring out how much time we can allow our job to wait, and always choose the "slowest" queue possible.

LATENCY

The answer to life, the universe, and everything?

@dmagliola

Now I sense maybe you're not convinced yet, so let's look at why this solves the problems we were seeing, in a bit more detail.

The key thing here is that the names we give these queues are clear promises, they are a contract, and they set clear expectations for our developers. And by putting a given job in a given queue, a developer is declaring what that job's needs are.

Remember we said earlier that the problem is that jobs have an implicit amount of delay they can tolerate. This is a requirement those jobs have, which is not documented anywhere. And it's not actionable in any way.

When we set up our queues like this, we turn that implicit requirement into an explicit one. If a job is in the "within10minutes" queue, you know it needs to start running in 10 minutes or less. Or else.

LATENCY

Not all jobs are created equal

@dmagliola

Another side of this is that when you have things like "high and low priority" queues, you will be unintentionally mixing jobs that can tolerate different levels of latency. If your "high" queue is running, say, 4 minutes behind, that will be fine for some jobs, but maybe not for others, and it's not very clear what we can do with that.

When you set clear expectations, this is much easier. You already know that all the jobs in the "within10minutes" queue can tolerate up to 10 minutes of delay. If your queue is running 5 minutes behind, you unequivocally know it's fine, because the authors of those jobs explicitly told you so, by putting them in that queue.

Metrics

The first step is knowing you have a problem

@dmagliola

This solves another one of our problems. If our queues are having trouble, we want to know before our customers find out, and fix it as fast as possible. So you want alerts for when things go wrong. If you have folks on-call, you want them to be paged if queues breach their latency guarantees.

When you have vague names for your queues, how do you set the threshold for those alerts? How do you know that any threshold you set is good for every job in that queue, that exists now or that will get added in the future?

With clear queue names, the alert pretty much sets itself. "within10minutes"? Cool, you alert if the latency hits 10 minutes. Done.

Prevention

@dmagliola

Now of course if you are alerting, it is too late. Things have already gone wrong. Ideally you want to try and prevent this from happening. You also don't want to waste too much money by having too many servers that are idle all the time.

These clear promises make it easier to do auto-scaling, and set adequate thresholds for it. You know precisely what level of latency is acceptable. The queues tell you! You have an idea of how long your servers take to boot up and start helping with the load. You add a bit of margin, and that's your auto-scaling threshold. When you hit that much latency, you start adding servers.

At IndeedFlex for example, we use 50% for most of our queues. If you hit 50% of the allowed latency, we start adding servers, 1 more every few minutes, until the latency goes back down, below 50%. Now 50% was an easy number to choose and a bit arbitrary, we can probably tune that up for the "slower" queues, but it's high enough that most of our queues are running on the minimum number of servers most of the time, and when we get spikes, the queues just deal with it, and we almost never get paged for breaches.

Oh, and by the way... If you do auto-scaling, always do it based on queue latency. Never the number of jobs pending, or anything else. Always based on latency.

> “The best things in life ain’t free
>
> -- Roger Taylor

@dmagliola

Now there’s a flip side to this… These guarantees don’t come for free.

Contracts always go both ways.

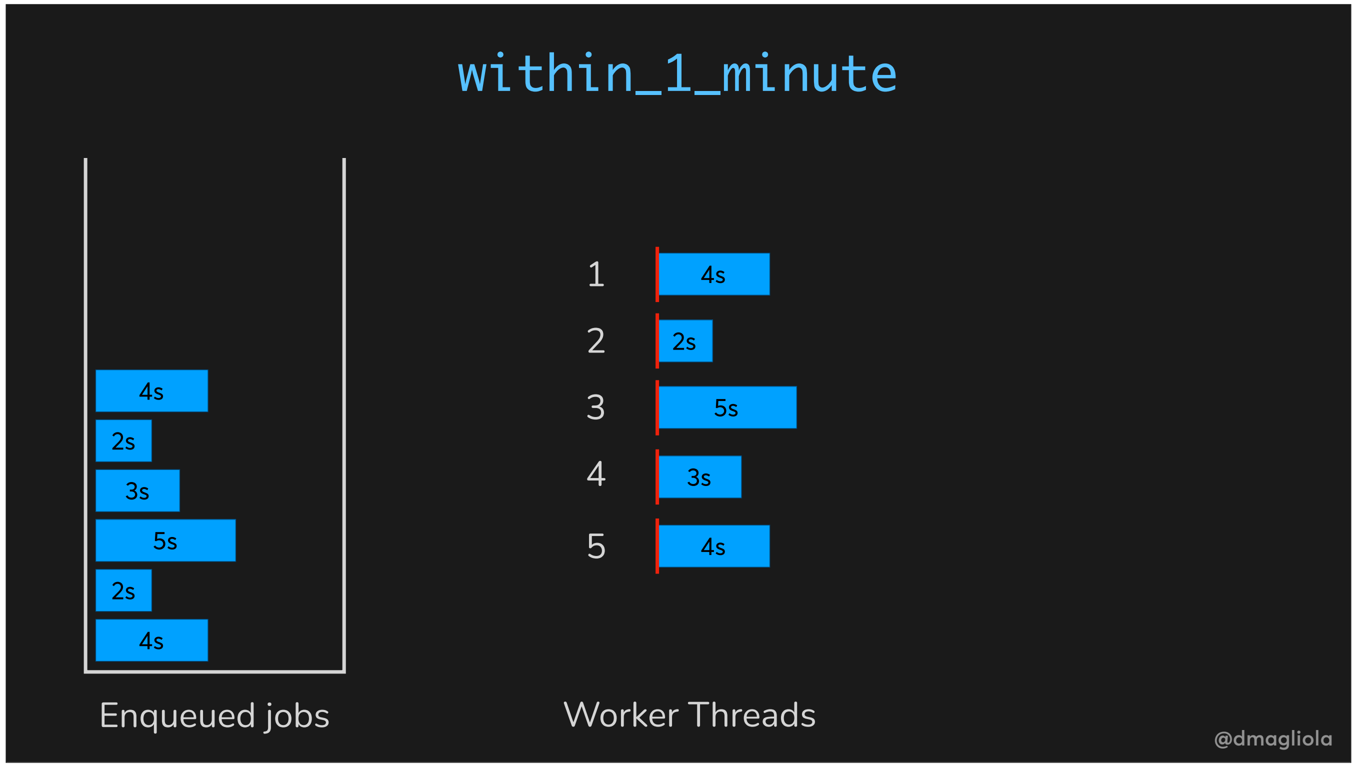
For us, that means just one simple rule: If you are on a queue that can only tolerate very small amounts of latency, you need to finish running pretty quickly. That is the flip side to this contract.

I call it the swimming pool principle. In a swimming pool, you normally have fast, medium and slow lanes, right? If you want to be in this lane, you better swim at least this fast, or you need to go to the slower lanes, so you don't get in the way.

In practice, this applies to our jobs too. If you are on a "slow" queue, one where jobs can tolerate a lot of latency, you can run for a while and it's fine. But if you are on a "fast" queue, where jobs need to start quickly, you need to get out of the way fairly fast, or you'll ruin the party for everyone.

When you put your job in a given queue, you're signing into that contract. If you break the contract, you'll get kicked out of the queue.

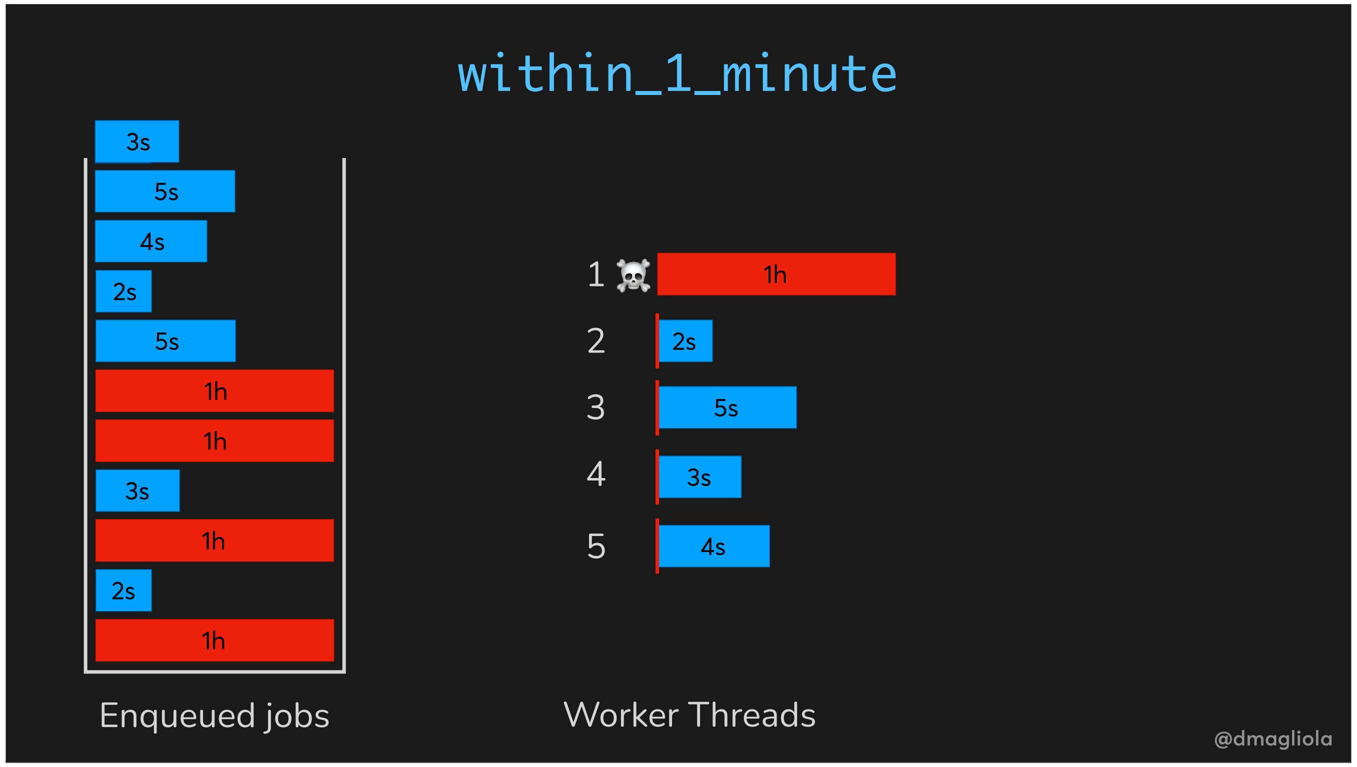
So we need to set a time limit for jobs in each of these queues. That is, if you want to run in the "within1minute" queue, you need to finish in "these many" seconds, or we can't take you in here, you need to go to a slower queue, sorry.



within_1_minute

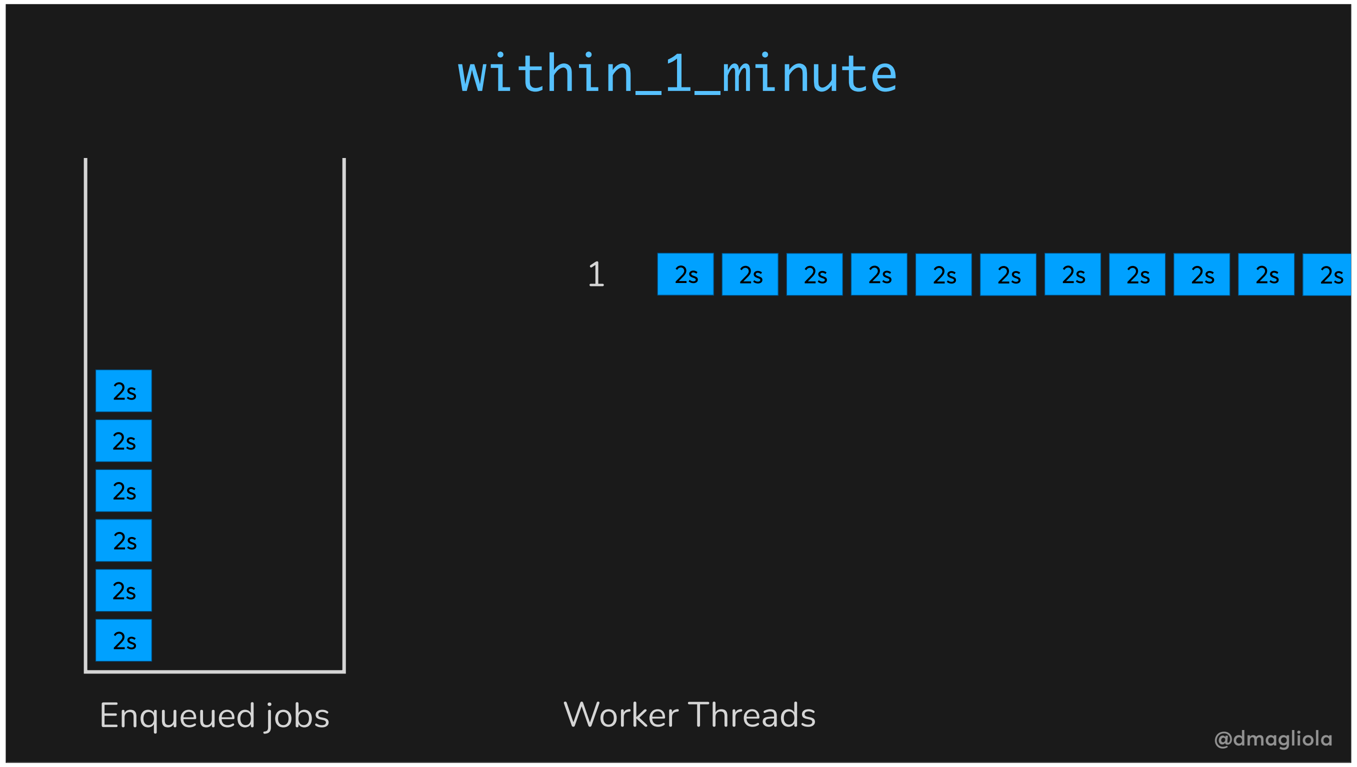
Now to see why, there's a bunch of "queuing theory formulas" to calculate this, but that's kinda boring, so just picture this instead…
You have a queue here on the left where all of the jobs need to start within 1 minute…

And like any queue, here on the right it has a number of threads working those jobs. And the queue is running its little jobs, and they get out of the way pretty quickly, new jobs arrive, and everyone is happy.



Now you suddenly start putting jobs in that queue that EACH take 1 hour from start to finish… What happens to that queue?

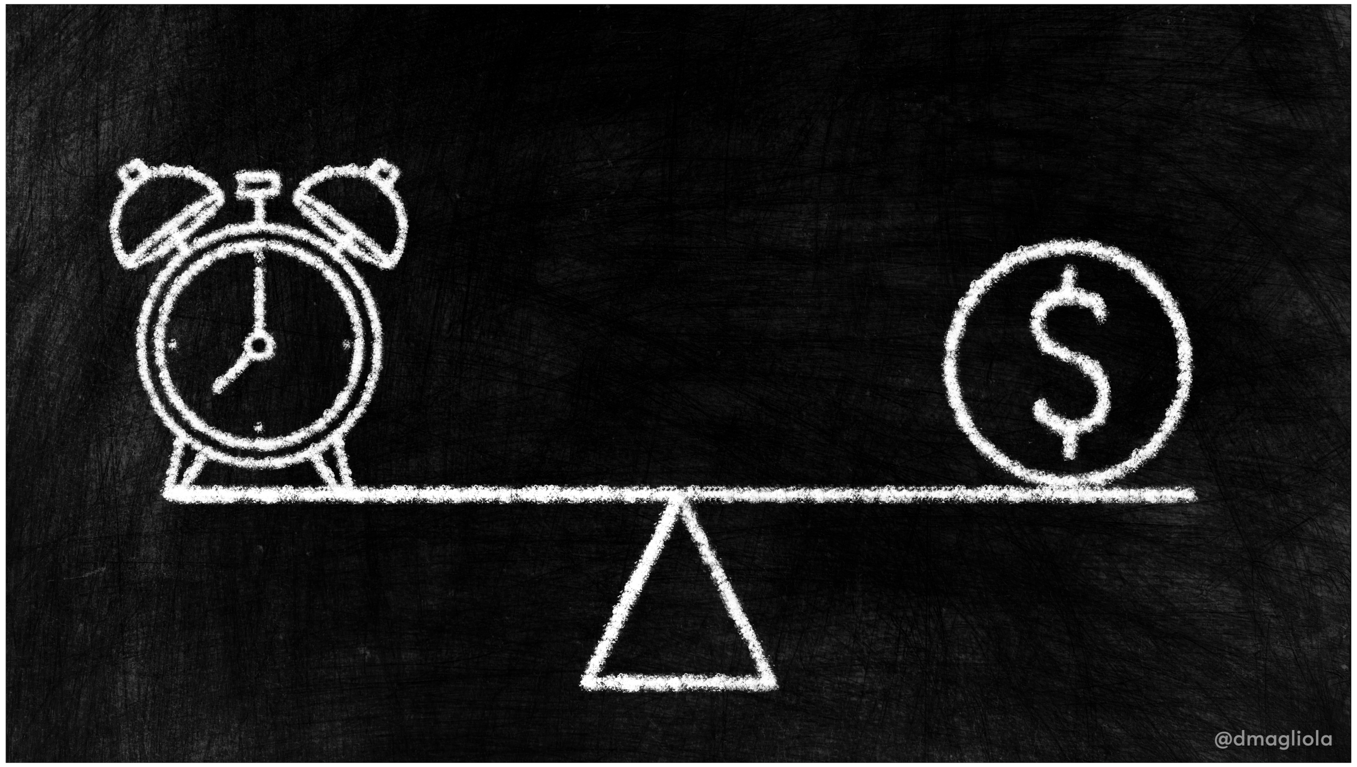
As soon as one of these threads gets one of those long jobs, that thread is now gone for an hour, it can't pick up any new jobs. So a small number of these long jobs will clog all the threads in that queue really quickly, and no work will get done anymore.
If you do autoscaling… You're going to add a new thread for each job that clogs one. That is a lot of threads.



within_1_minute

Now imagine instead that every job in that queue finishes in 2 seconds instead… Now one single thread could handle a lot of jobs, before you breach your promise. You can now run a lot more jobs with a lot fewer threads. It’s going to be much easier to keep your latency low.

This is also by the way why ”priorities” don’t work. Your low priority, long running jobs can easily clog all the threads, and there’s no one left to pick up your high priority ones.



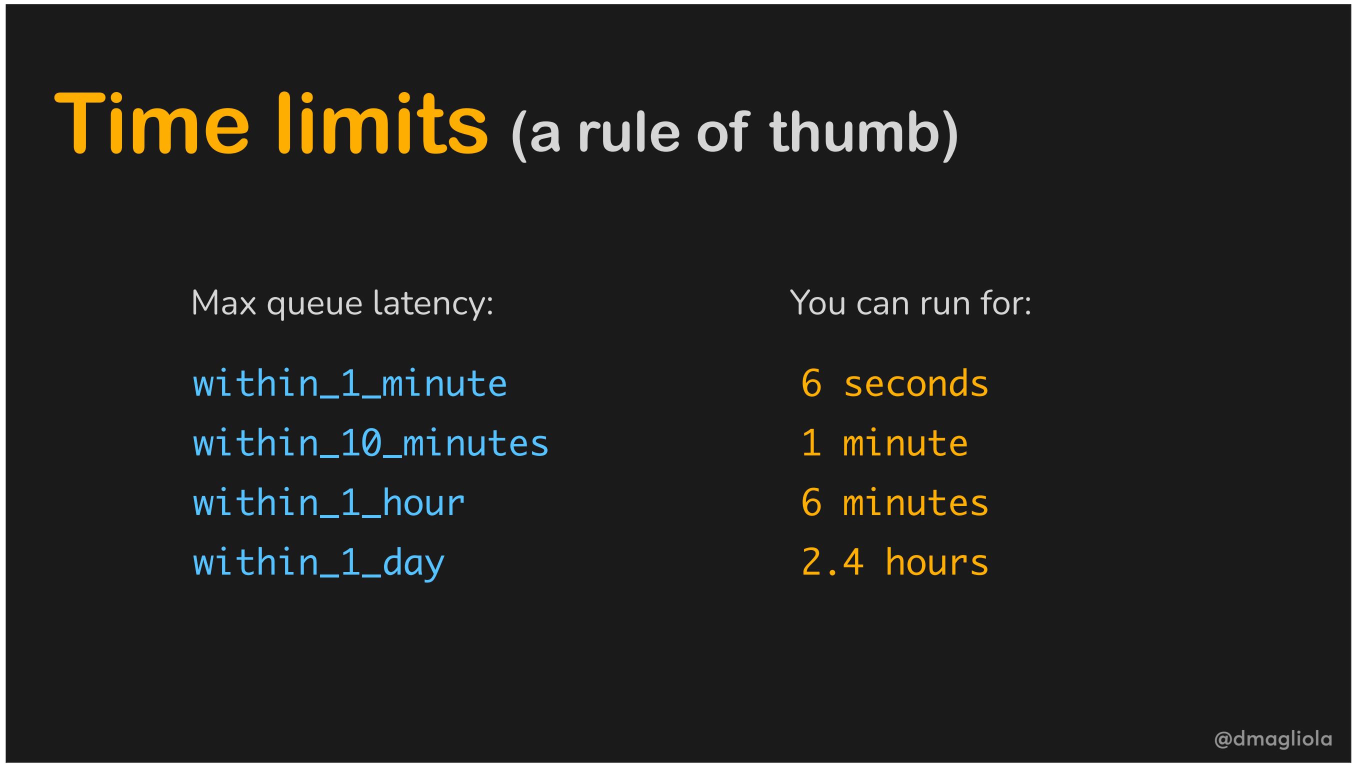
So you need to set time limits for each queue. The faster the queue, the faster you need to finish.

Now where you choose to set these depends pretty much on how much money you're willing to spend on servers… and on developers. It's a trade-off, like everything else in life.

If you set these limits too high, if you let jobs run for a long time… That's going to make your developers lives easier… But you'll need more threads, and you'll spend more money in servers.

If you set them too low, jobs that need to run with low latency need to be optimized more, maybe refactored and split into different parts… It's going to be more annoying for your developers, and you're going to spend more time doing that and so you'll spend more money in developers.

It's a balancing act, and it depends on your priorities.



Time limits (a rule of thumb)

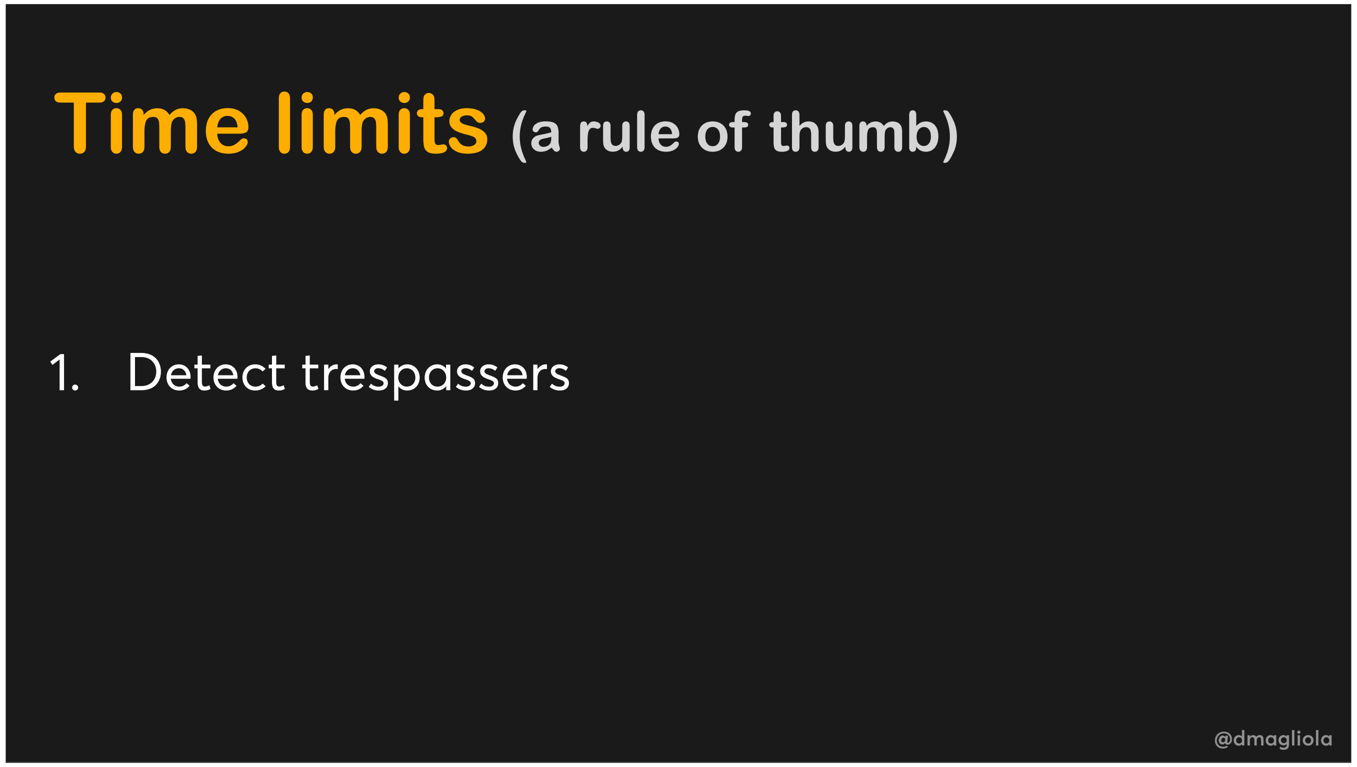
Max queue latency:	You can run for:
`within_1_minute`	`6 seconds`
`within_10_minutes`	`1 minute`
`within_1_hour`	`6 minutes`
`within_1_day`	`2.4 hours`

@dmagliola

I like the rule of thumb of setting the target at "1/10th the latency limit".

So, a job in "within1minute" can run for up to 6 seconds, one in "within10minutes" can run for up to 1 minute, etc.

In reality, this is kind of aspirational, you're not going to start here, and it's also going to vary a lot based on the current state of your app and your codebase. It's quite possible you'll have to set it quite a lot higher at first and spend more on servers.



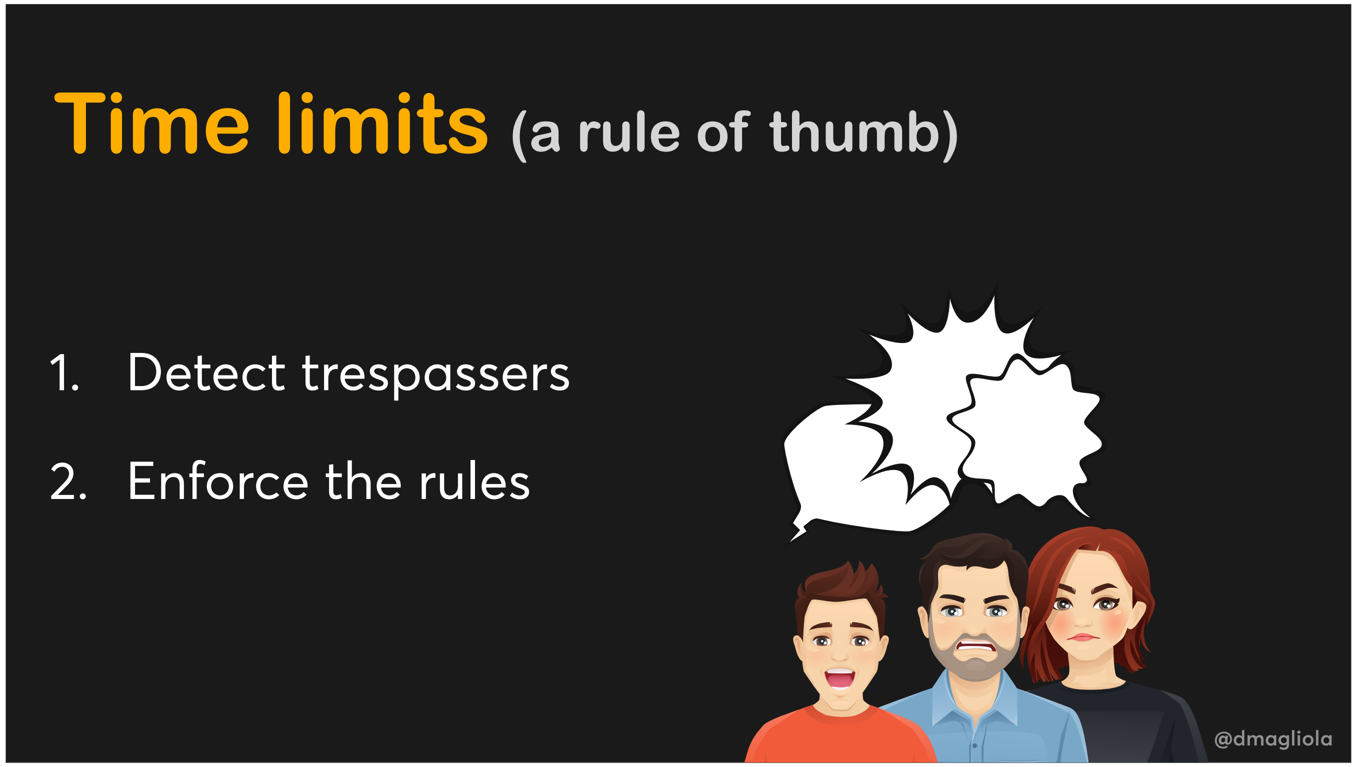
Time limits (a rule of thumb)

1. Detect trespassers

@dmagliola

Now there are 3 important things to keep in mind, that are going to be key to this implementation:

#1: The latency promise in the queue name, AAAAND these time limits are a contract between the infrastructure and the developers. And like any contract, we need to know if it's broken. If a job is going over these limits, a warning should go out to the respective team to either improve this or move it to a slower queue. This is not a page, it shouldn't wake anyone up in the middle of the night, but it should make it clear that it needs to be changed.



Time limits (a rule of thumb)

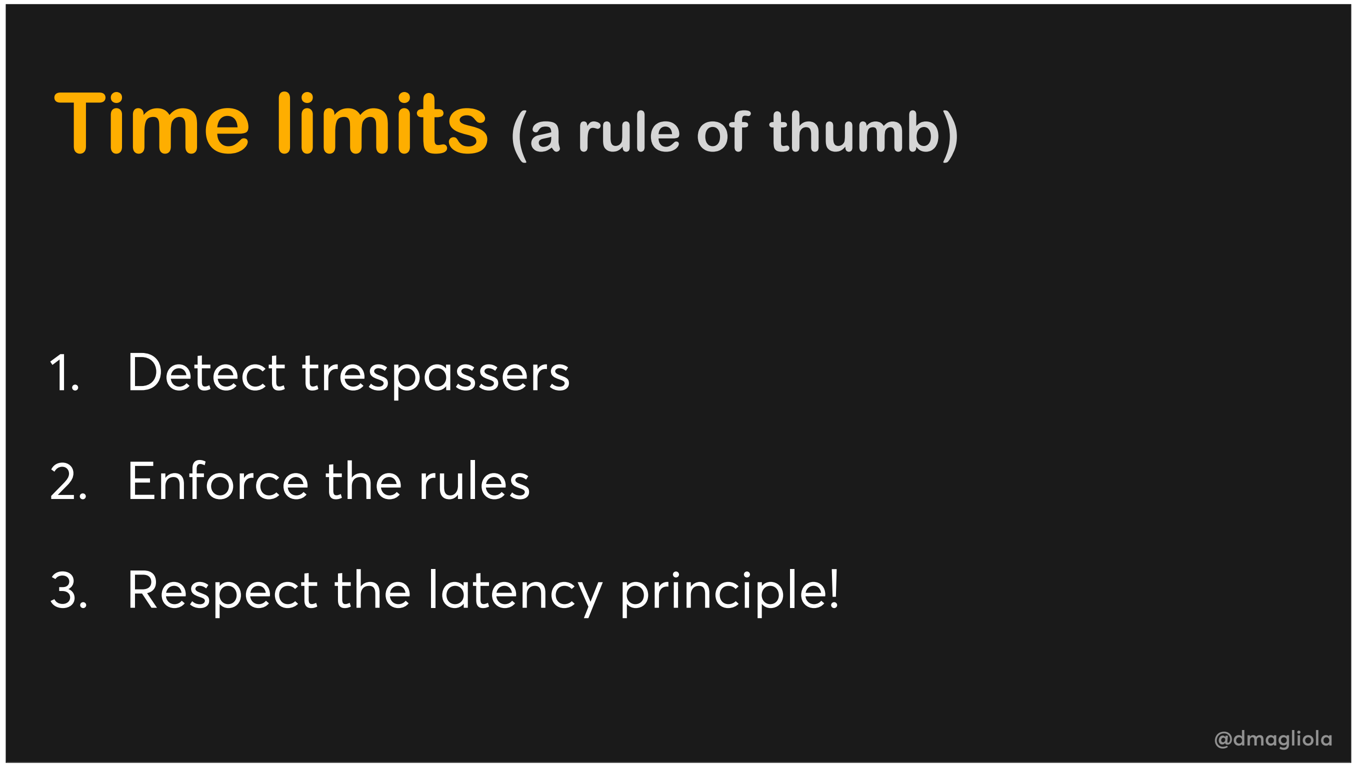
1. Detect trespassers
2. Enforce the rules

@dmagliola

#2: There's an implicit social contract here.

Remember the inter-team fights that resulted in "one queue per team"? This is how you prevent that. If a job is breaching the time limit, there need to be an explicit permission for folks to move it. Of course, we'll all talk to the respective people involved, and try to get an idea of what the best course of action is…

But if a given job is making queues sad for everyone, you need to have that explicit permission for other teams to move it out,
or you WILL end up back in a situation where "each team wants to control their own queue".



Time limits (a rule of thumb)

1. Detect trespassers
2. Enforce the rules
3. Respect the latency principle!

@dmagliola

#3: Respect the principle, and focus on latency tolerance. Just because a job finishes quickly and can run in the fast queues, doesn't mean it should.

There's going to be a temptation to put jobs in the fastest queue that'll accept them. Don't do that! You may have jobs that run very quickly, but nobody cares if they're late. Put them in the slow queues, don't let them get in the way of jobs that are actually sensitive to latency!

And this… is how you fix the problems that you might be having with your queues.

@dmagliola

"But wait, Daniel", I hear you say…

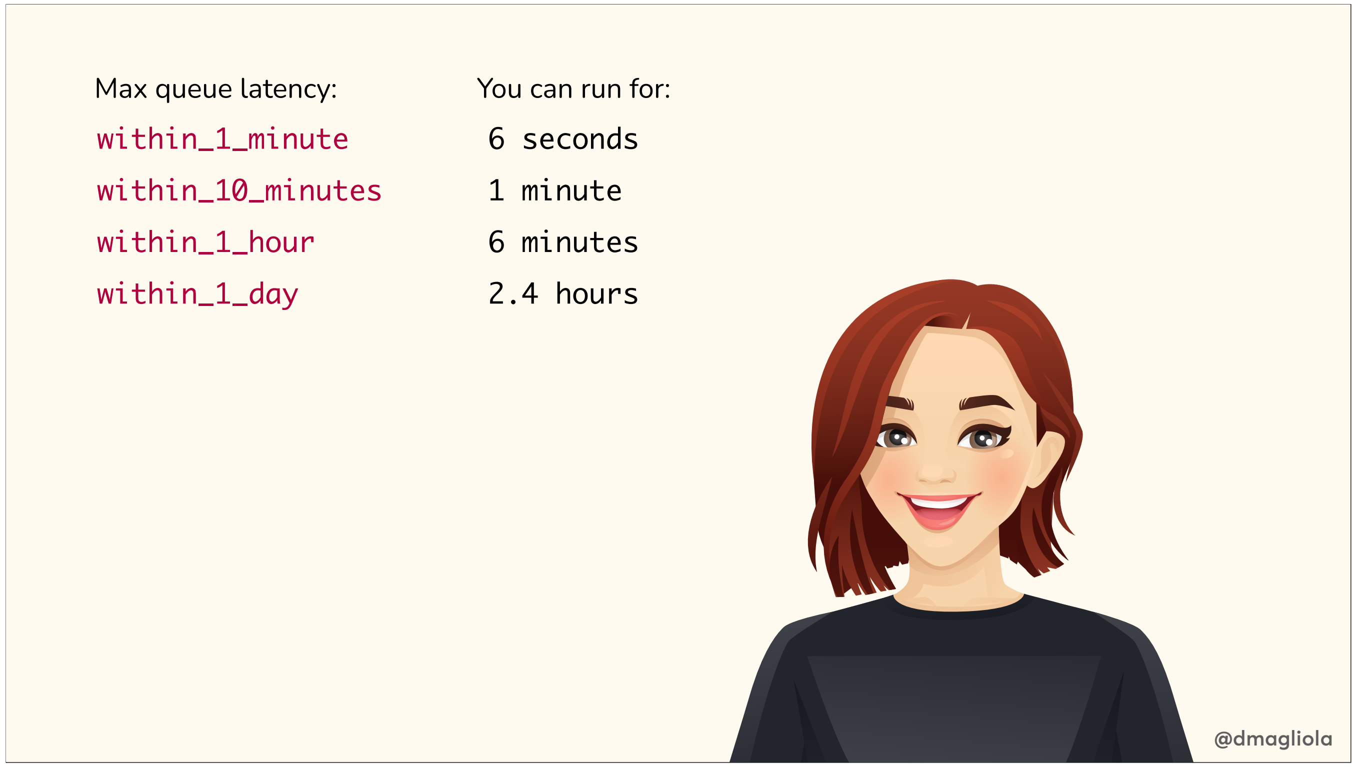
ACT 4

Speedily ever after

What happened to our merry band of software developers?
Well,

after her epiphany, Sally proposed this new structure to the team, and they liked it!, and they started work on adopting it.

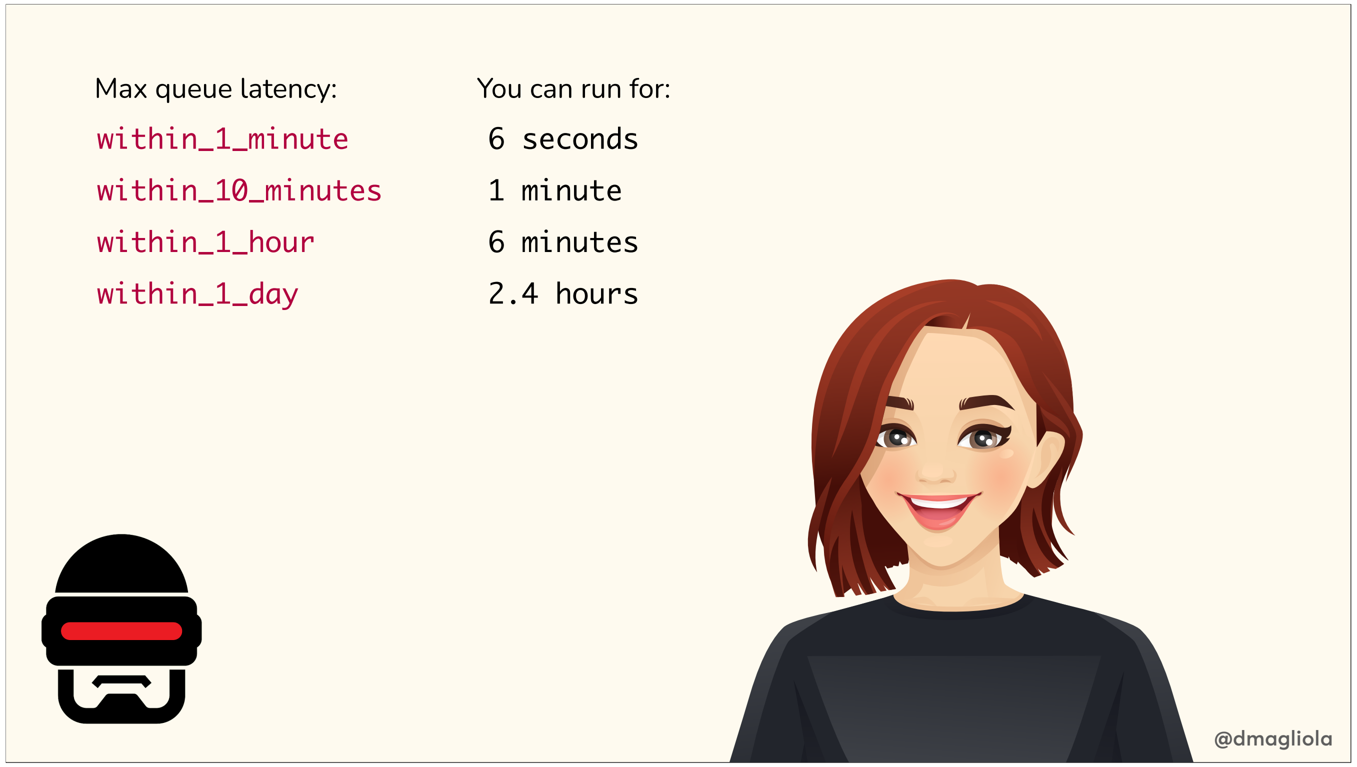
It wasn't easy, and it certainly wasn't quick, but they got there in the end.
And there are useful lessons in what they did…



Max queue latency:	You can run for:
`within_1_minute`	6 seconds
`within_10_minutes`	1 minute
`within_1_hour`	6 minutes
`within_1_day`	2.4 hours

@dmagliola

First of course they created the new queues, with their alerts!,
and the rule was that every new job from now on can only go into those new queues.

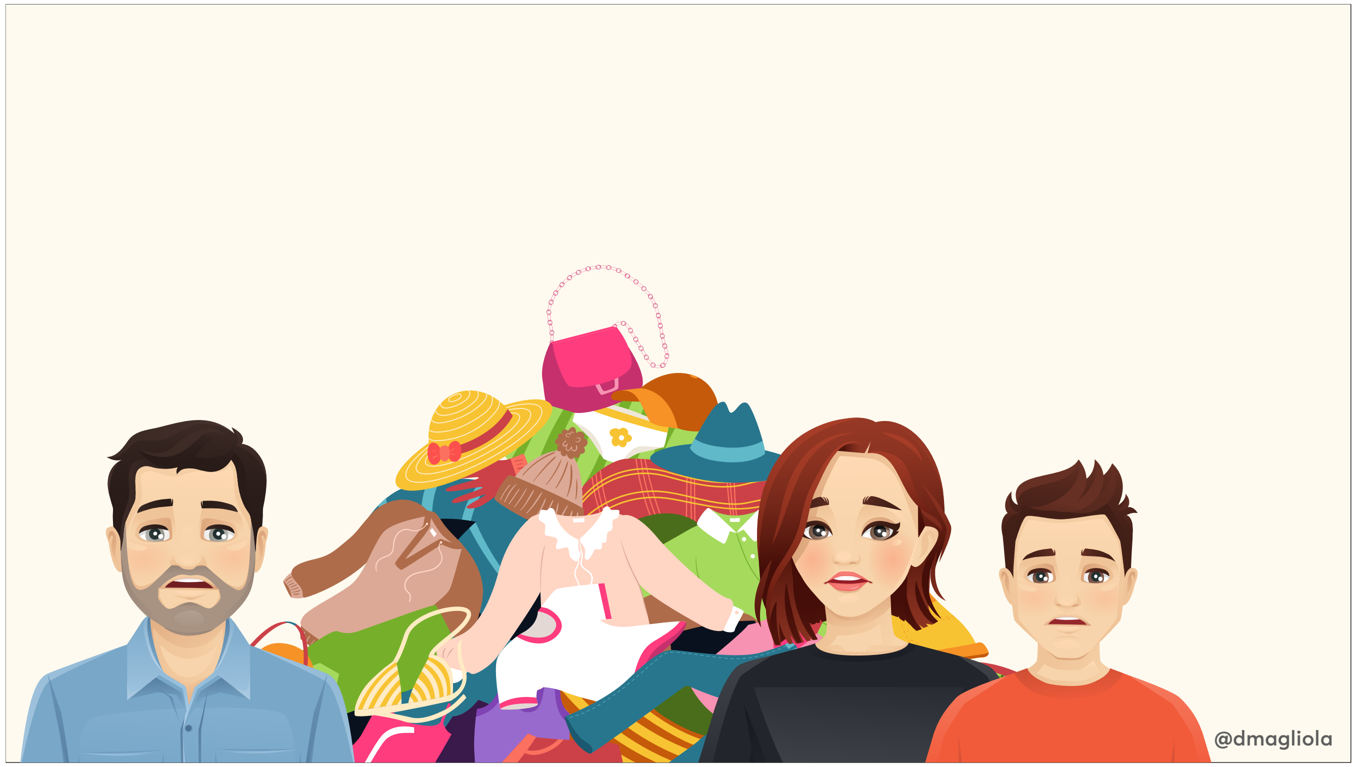


Max queue latency:	You can run for:
`within_1_minute`	6 seconds
`within_10_minutes`	1 minute
`within_1_hour`	6 minutes
`within_1_day`	2.4 hours

@dmagliola

They actually made a custom Rubocop rule, to make sure no one would forget!

And then, they started the huge task of



figuring out where all the existing jobs should go.

Which… is a lot, and it can be a bit daunting, but they found a very clever strategy.
First, they started not with the most important things, but with the easiest.

critical
crm_sync
notifications
db_maintenance
pricing
purchasing_high
logistics
release_team
nightly_checkout
@dmagliola

within_1_minute

within_10_minutes

within_1_hour

within_1_day

For example, some queues can be pretty obvious.
They had 2 queues that run "overnight maintence jobs". Easy! No one's in a rush for those.

critical
crm_sync
notifications
sources
event_
pricing
purchasing_high
logistics_team
release_team
warehouse
@dmagliola

within_1_minute

within_10_minutes

within_1_hour

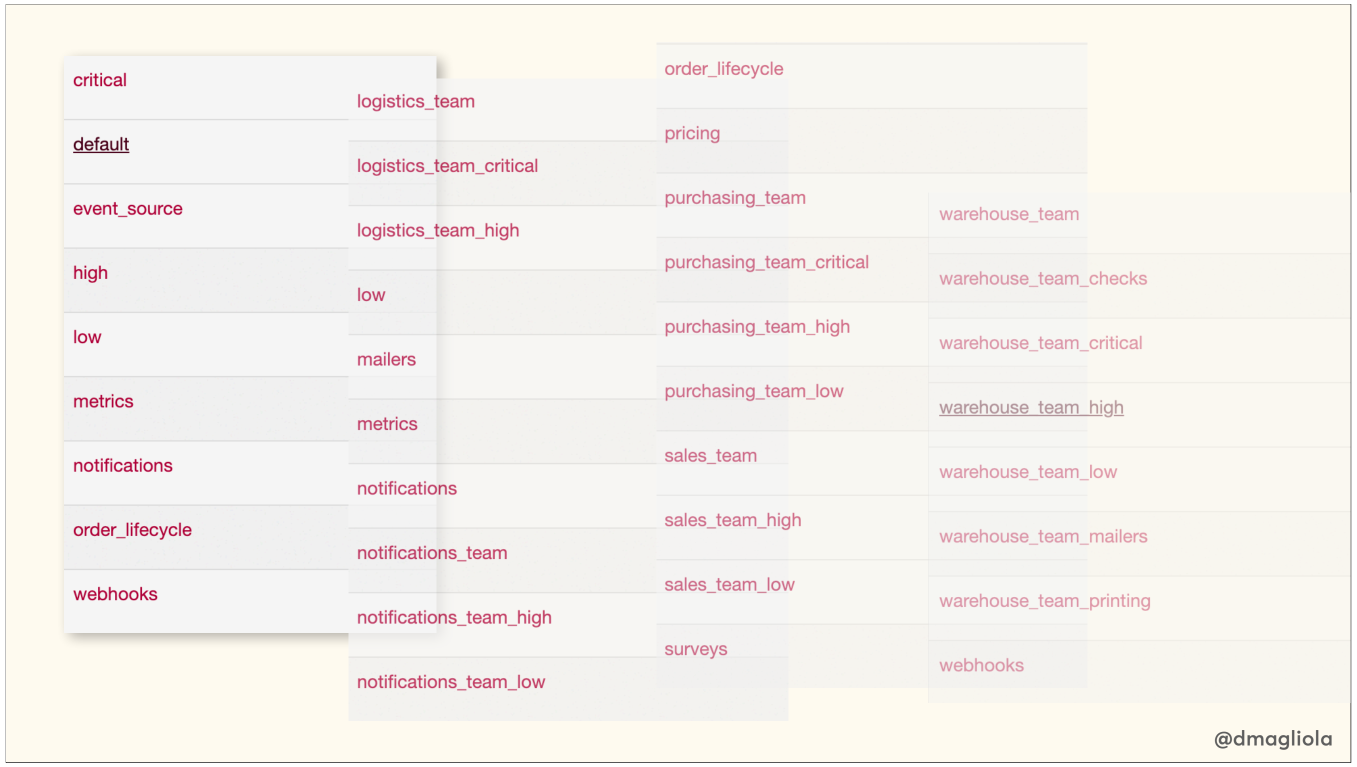
within_1_day

nightly_checks

db_maintenance

That goes into "within1day".

Jobs that took really long to run, for the most part, also got sent to the slow queues. No one expected those to run quickly anyway.
And then, once the easy stuff is moved out, there's no point in having the queues for each team anymore, we can just collapse them all back into "priorities", and maybe add more servers to them if necessary, you're still coming out ahead.



critical
default
event_source
high
low
metrics
notifications
order_lifecycle
webhooks

logistics_team
logistics_team_critical
logistics_team_high
low
mailers
metrics
notifications
notifications_team
notifications_team_high
notifications_team_low

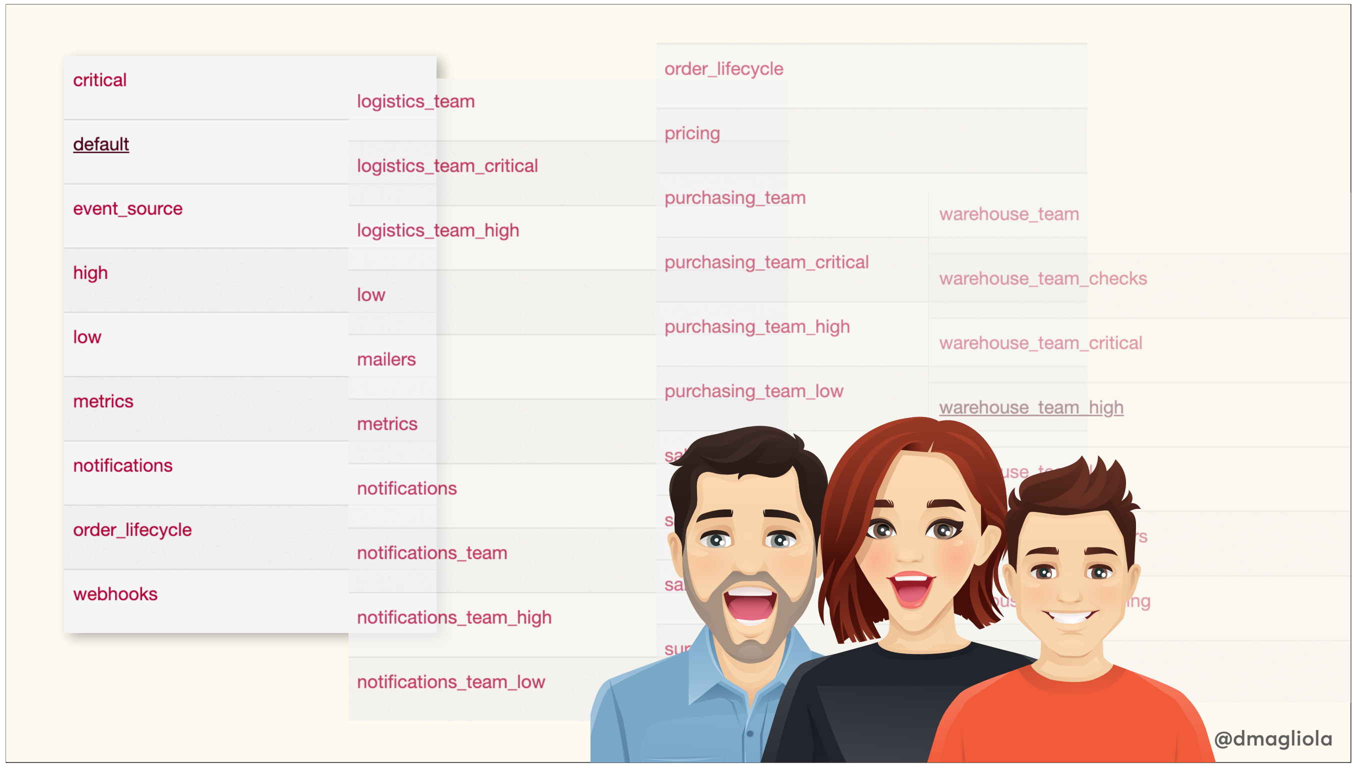
order_lifecycle
pricing
purchasing_team
purchasing_team_critical
purchasing_team_high
purchasing_team_low
sales_team
sales_team_high
sales_team_low
surveys

warehouse_team
warehouse_team_checks
warehouse_team_critical
warehouse_team_high
warehouse_team_low
warehouse_team_mailers
warehouse_team_printing
webhooks

@dmagliola

And pretty quickly, those 60 queues turned into 9, and this was a massive motivation push.

It was SO MUCH EASIER to keep the system running well with all those fewer queues.



critical
default
event_source
high
low
metrics
notifications
order_lifecycle
webhooks

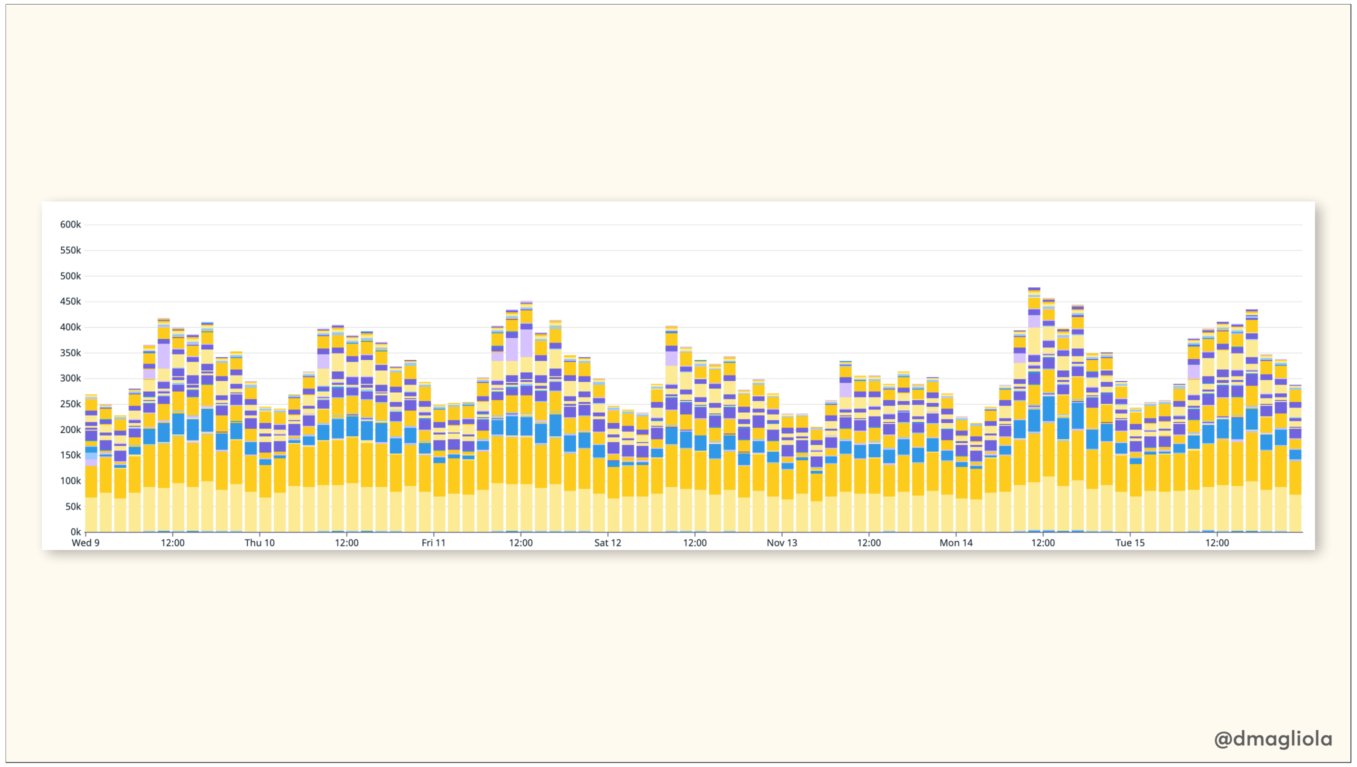
logistics_team
logistics_team_critical
logistics_team_high
low
mailers
metrics
notifications
notifications_team
notifications_team_high
notifications_team_low

order_lifecycle
pricing
purchasing_team
purchasing_team_critical
purchasing_team_high
purchasing_team_low

warehouse_team
warehouse_team_checks
warehouse_team_critical
warehouse_team_high

@dmagliola

And they could see real progress! They were so eager to do the rest!



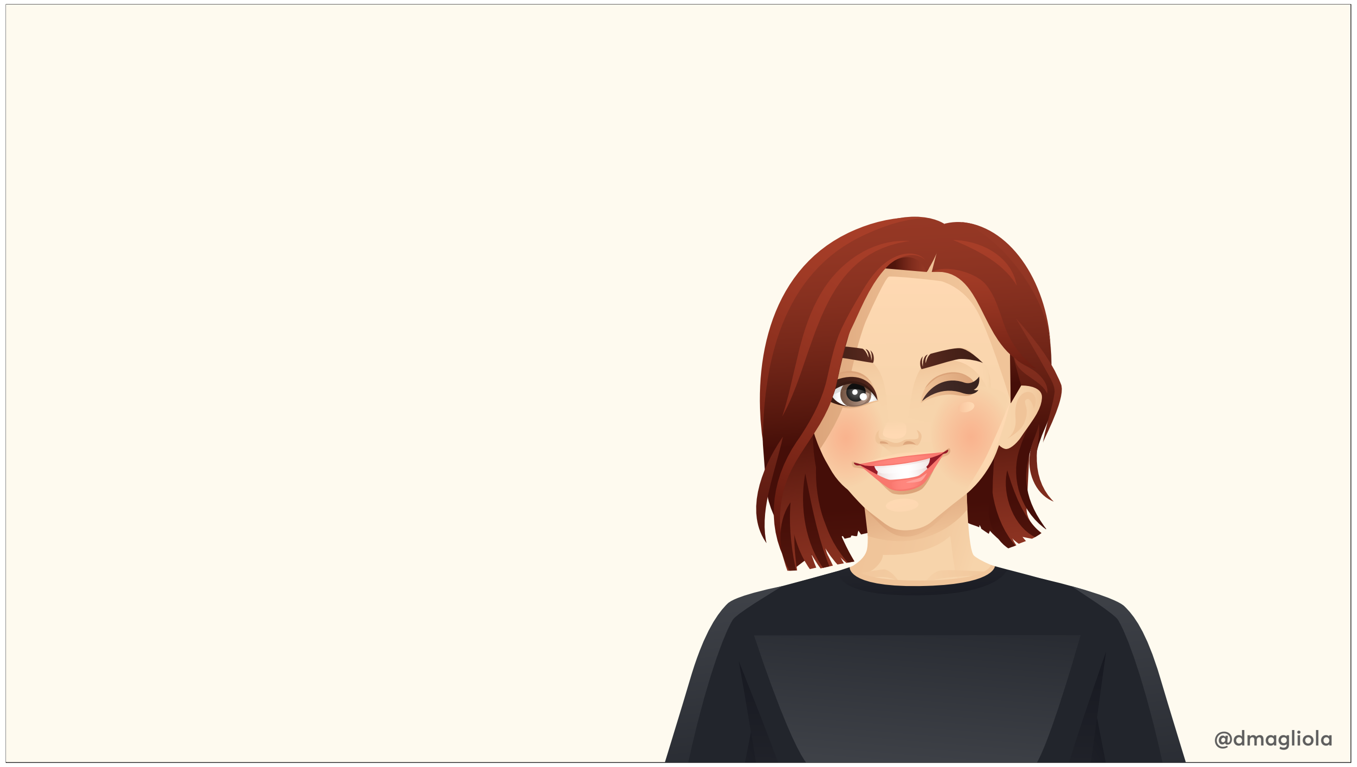
@dmagliola

The next trick? That was really clever.

They went for the jobs that ran the most often, the highest volume ones. Those are always a candidate for ruining your day, so you want to make sure they're in the right queues… Those are sometimes hard to categorize, but normally there's not that many of them!

And as they kept going down this list, at one point they realized that they had a lot of different jobs left… There was a lot of work left to do. But their total volume was tiny compared to the rest…

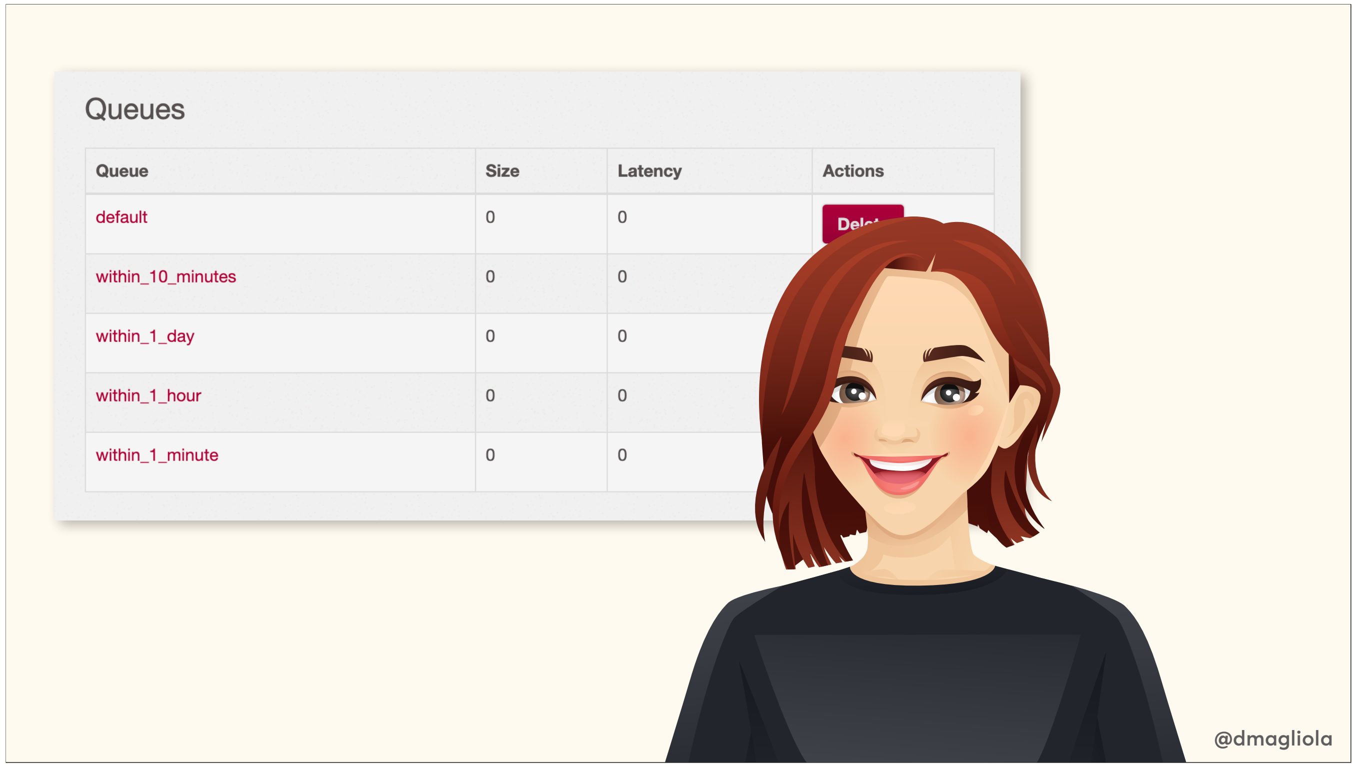
And here's a little secret…



@dmagliola

They never actually finished…

After a while, the jobs they had left were so few and so quick, that there was very little chance they would cause problems.



Queues

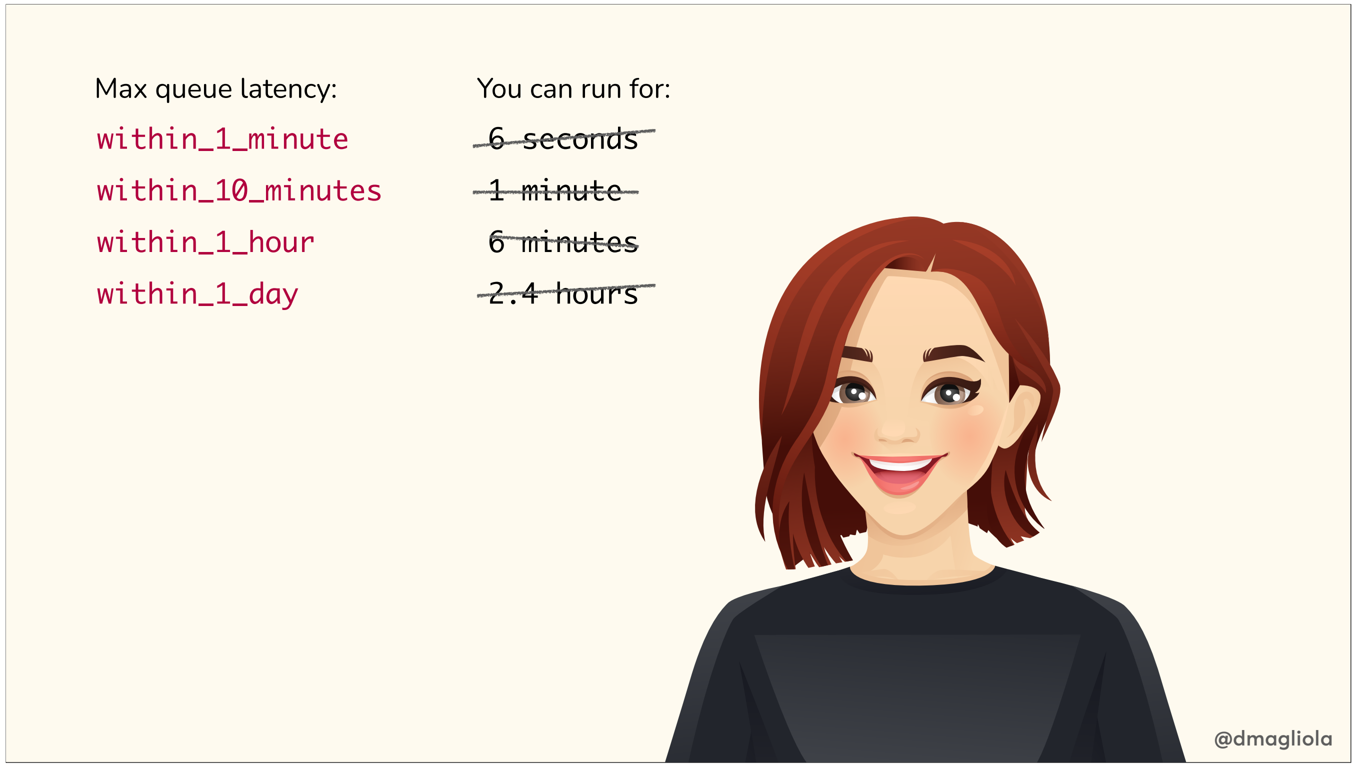
Queue	Size	Latency	Actions
default	0	0	
within_10_minutes	0	0	
within_1_day	0	0	
within_1_hour	0	0	
within_1_minute	0	0	

@dmagliola

So they just merged them all into the "default" queue, and left them there. It wasn't worth cataloguing each one.
Which left them with this.

This was bliss. It was a dream to keep it running day-to-day, but they weren't done.

You see, to move faster and keep momentum, they did take a couple of shortcuts.

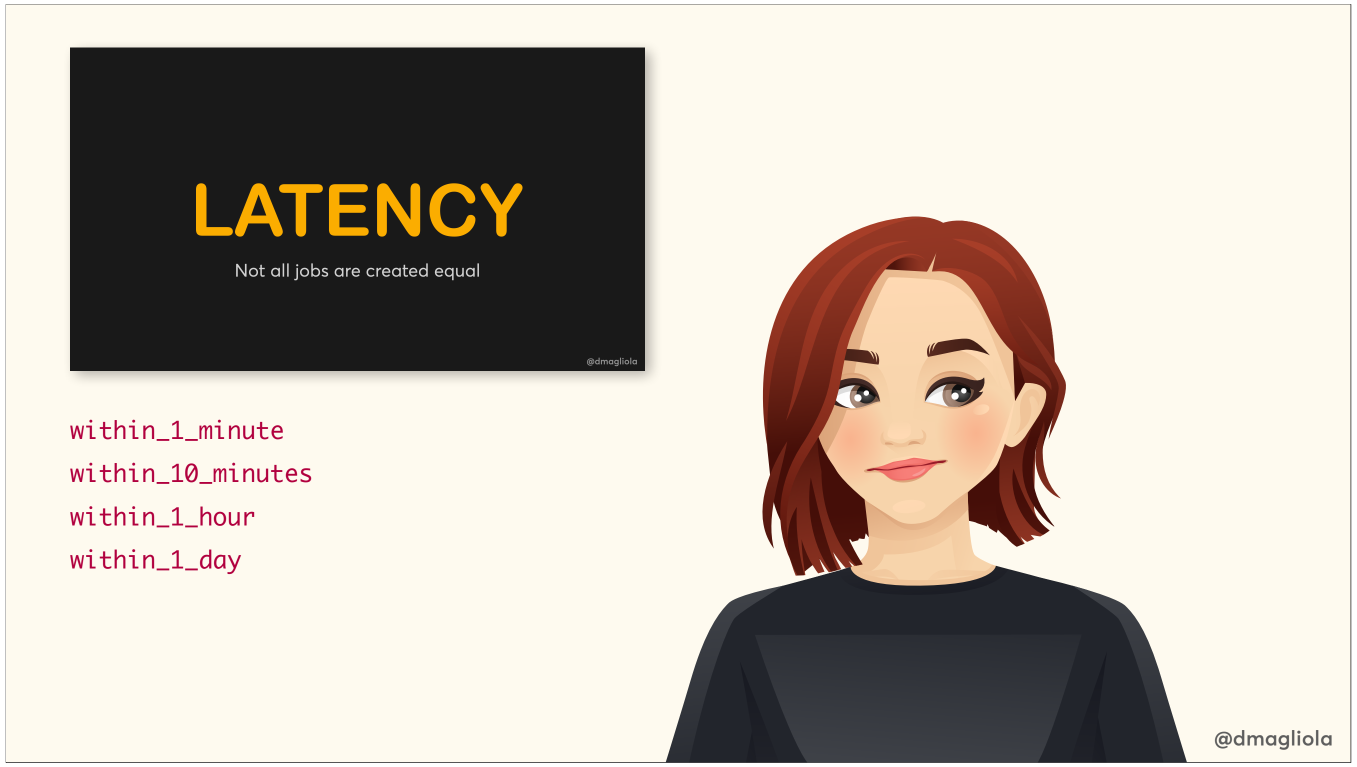


Max queue latency:	You can run for:
within_1_minute	~~6 seconds~~
within_10_minutes	~~1 minute~~
within_1_hour	~~6 minutes~~
within_1_day	~~2.4 hours~~

@dmagliola

First of all, those speed limits? They set them WAY higher at first, and just threw more servers at the problem.

This made it much easier to start putting jobs in the right queues, without having to worry too much about their performance. But now they had a clear ordered list of "trespassers"... Jobs that were too slow for their queues, which they then work on, gradually making them faster, and then gradually lowering these time limits until they got to their sweet spot.



LATENCY

Not all jobs are created equal

@dmagliola

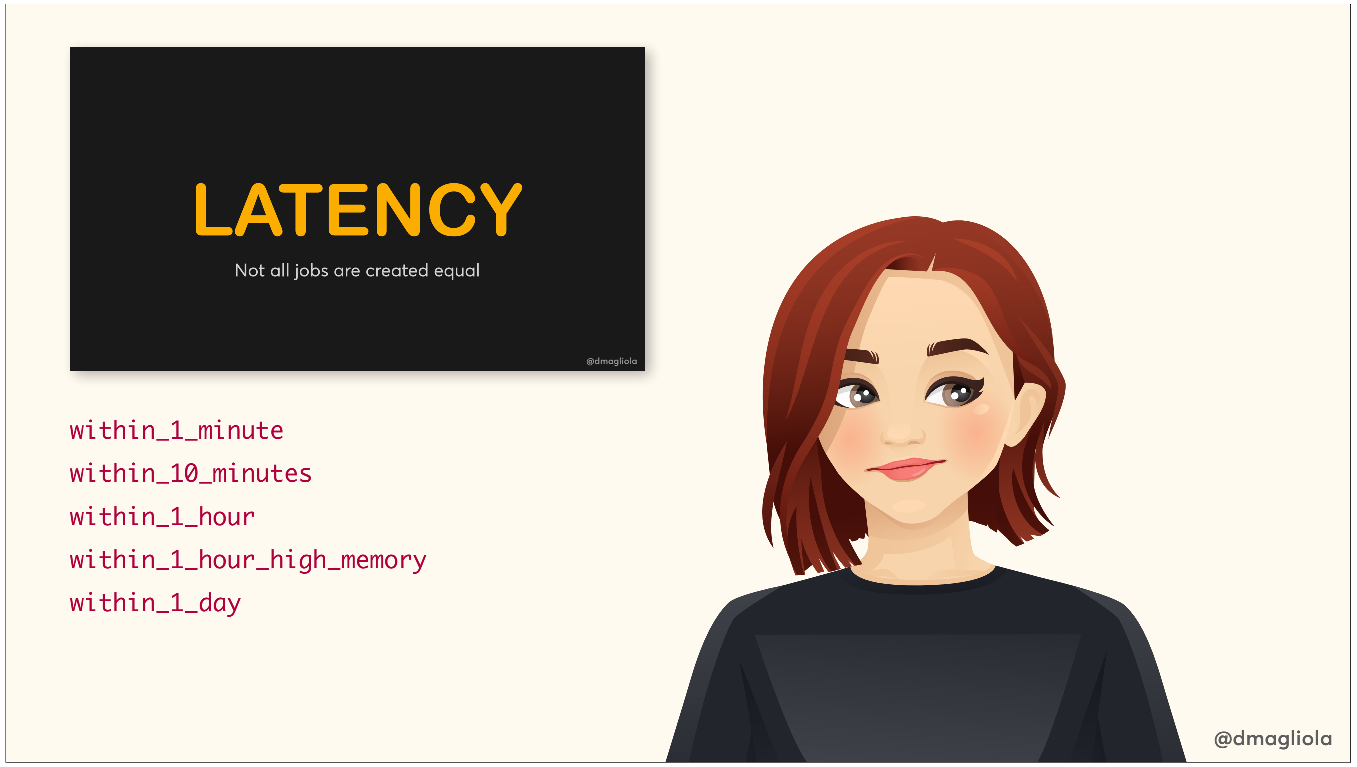
```
within_1_minute
within_10_minutes
within_1_hour
within_1_day
```

@dmagliola

They also learned an extremely important lesson… When to break the one rule…

You know how I said latency is the ONE thing you care about? Well that’s true, but sometimes practical considerations mean it makes sense to stray from the path a little bit…

For example: Some jobs need special hardware, generally a lot more RAM.



LATENCY

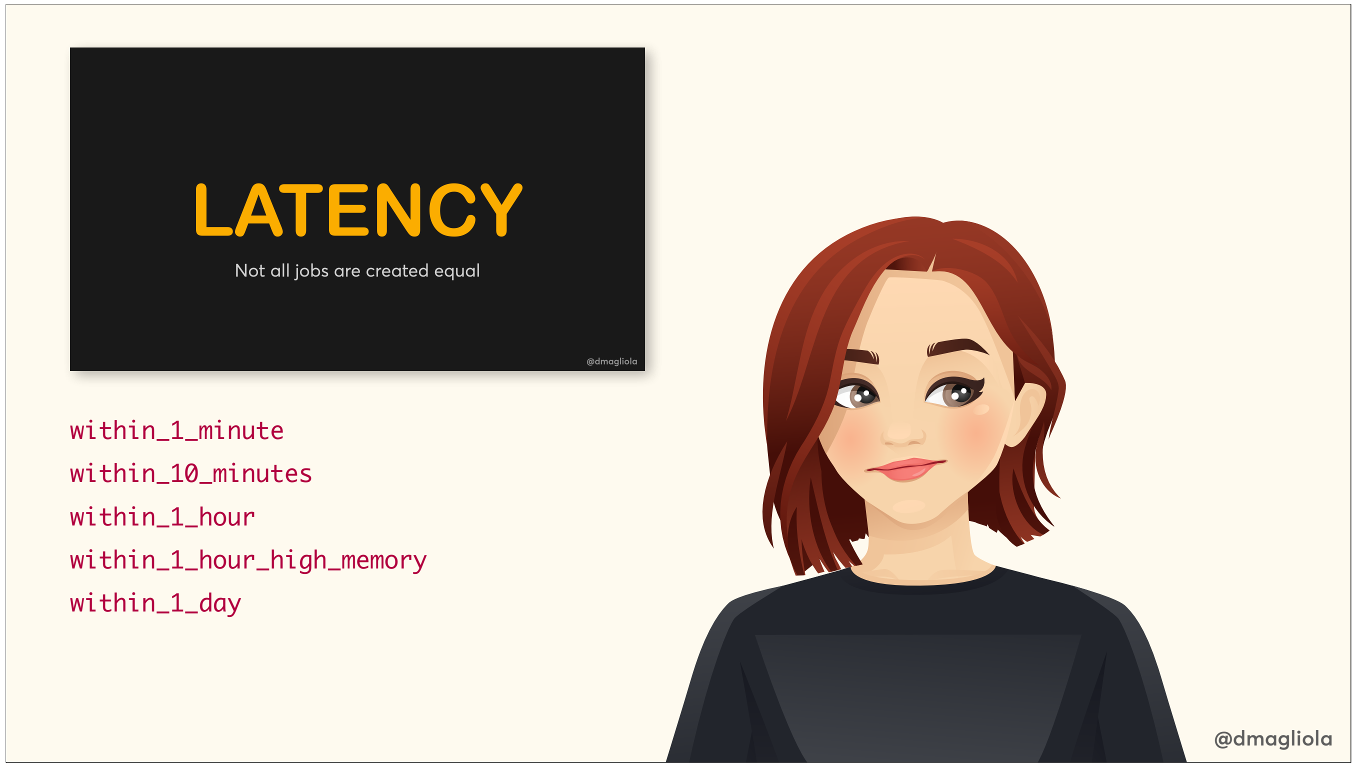
Not all jobs are created equal

@dmagliola

```
within_1_minute
within_10_minutes
within_1_hour
within_1_hour_high_memory
within_1_day
```

@dmagliola

If you can put them in their own queue, then you can have a few large servers for them, instead of making all the servers for that queue more expensive.



LATENCY

Not all jobs are created equal

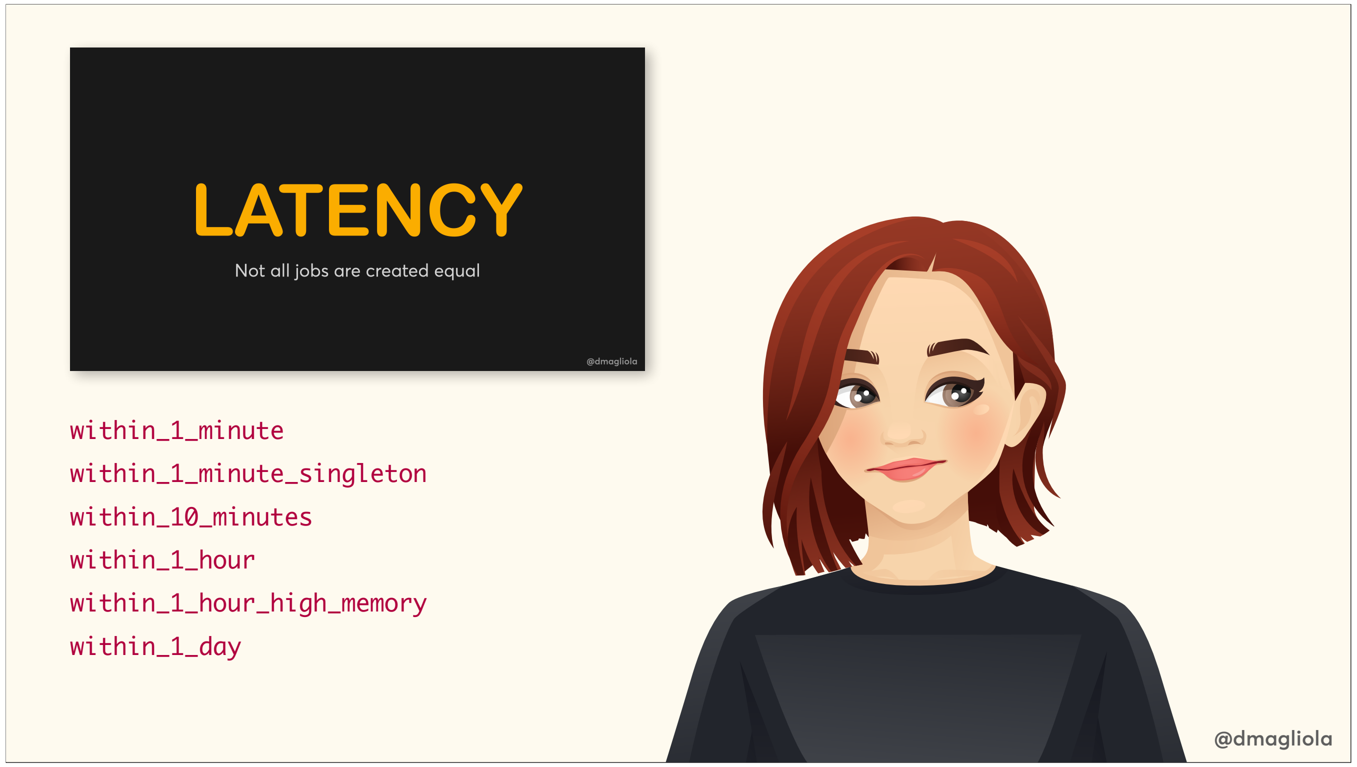
@dmagliola

```
within_1_minute
within_10_minutes
within_1_hour
within_1_hour_high_memory
within_1_day
```

@dmagliola

Sometimes you have jobs that want to run one at a time…

This is generally an anti-pattern… But there are actually some legitimate reasons to do this.



LATENCY

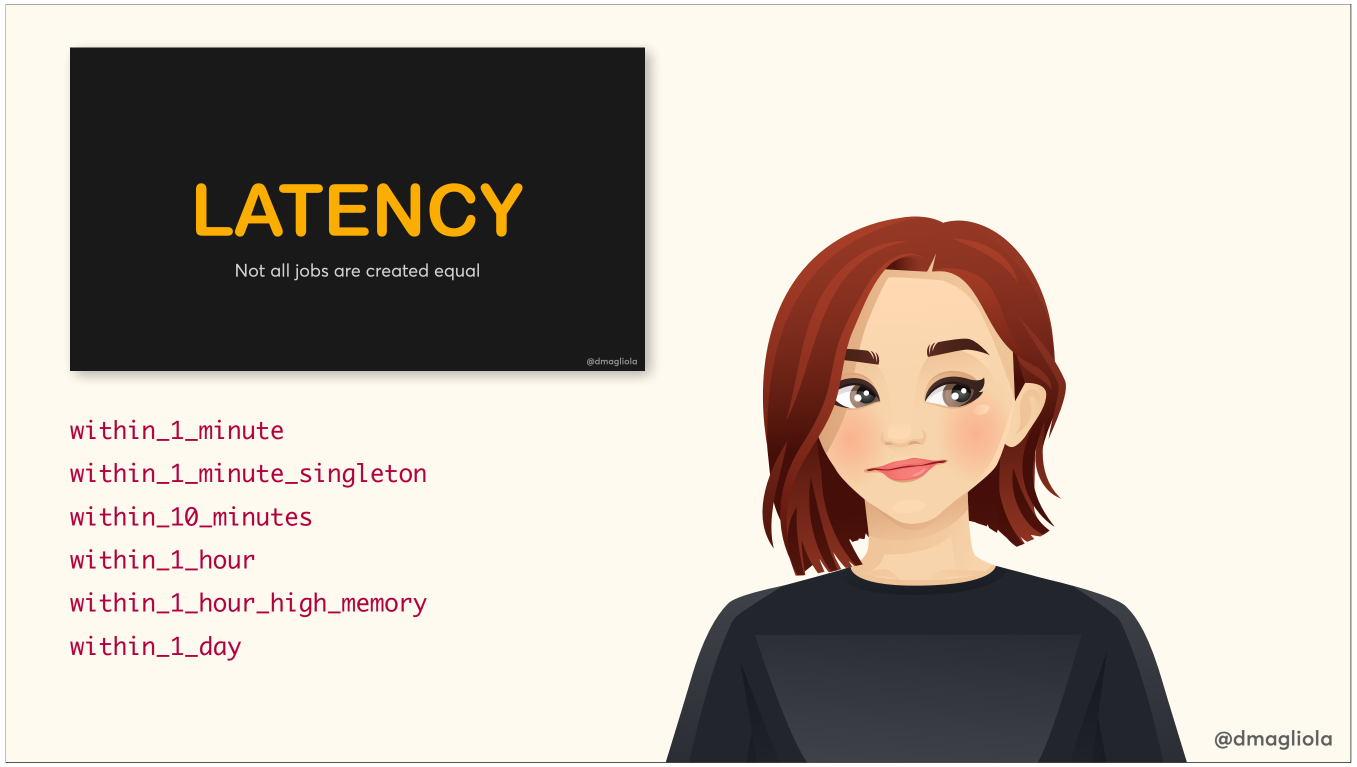
Not all jobs are created equal

@dmagliola

```
within_1_minute
within_1_minute_singleton
within_10_minutes
within_1_hour
within_1_hour_high_memory
within_1_day
```

@dmagliola

Having their own queue, with 1 thread and never more than 1 server lets you do that. And the name makes it clear you should never give that more than one instance.



LATENCY

Not all jobs are created equal

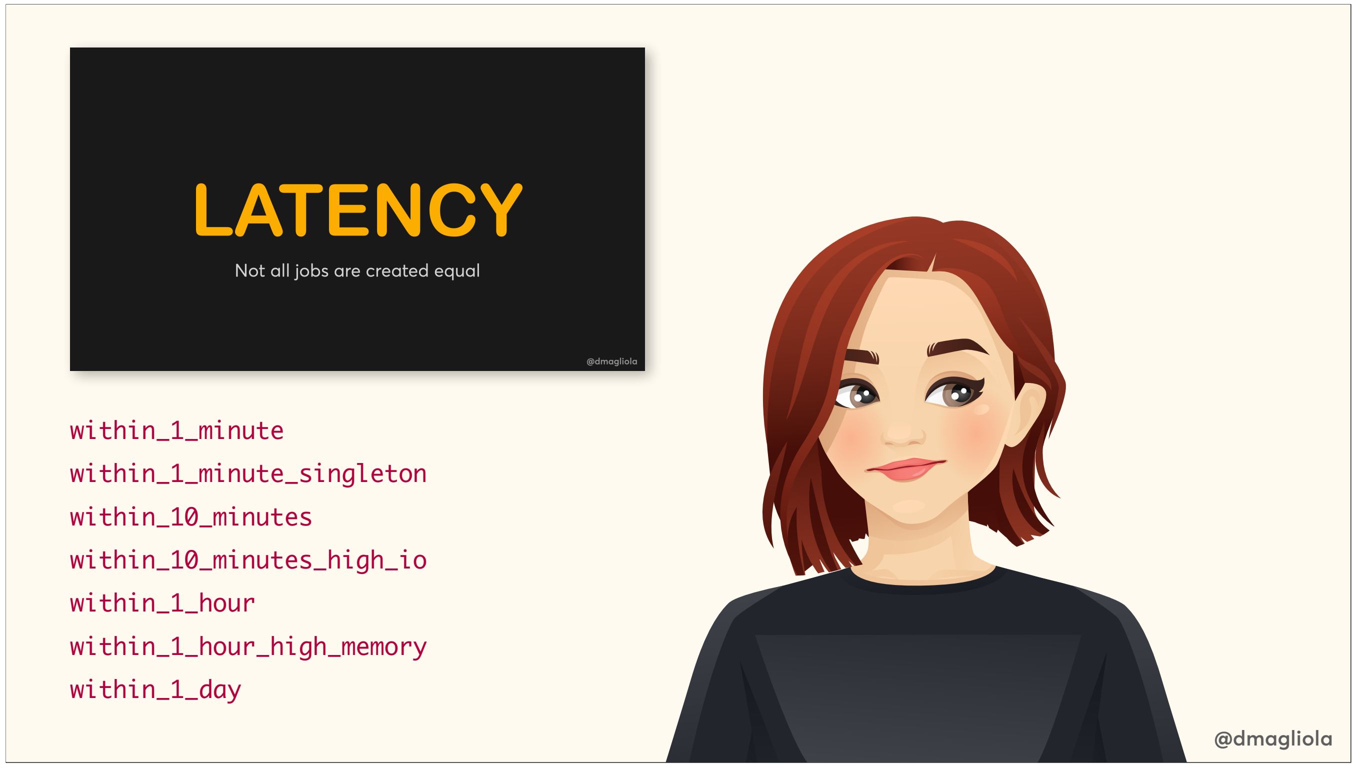
@dmagliola

```
within_1_minute
within_1_minute_singleton
within_10_minutes
within_1_hour
within_1_hour_high_memory
within_1_day
```

@dmagliola

Some jobs do A LOT of waiting on I/O, and use almost no CPU. Mailers are like that.

You could have really high thread counts for those, and save on servers, but that would hurt the other jobs…



LATENCY

Not all jobs are created equal

@dmagliola

```
within_1_minute
within_1_minute_singleton
within_10_minutes
within_10_minutes_high_io
within_1_hour
within_1_hour_high_memory
within_1_day
```

@dmagliola

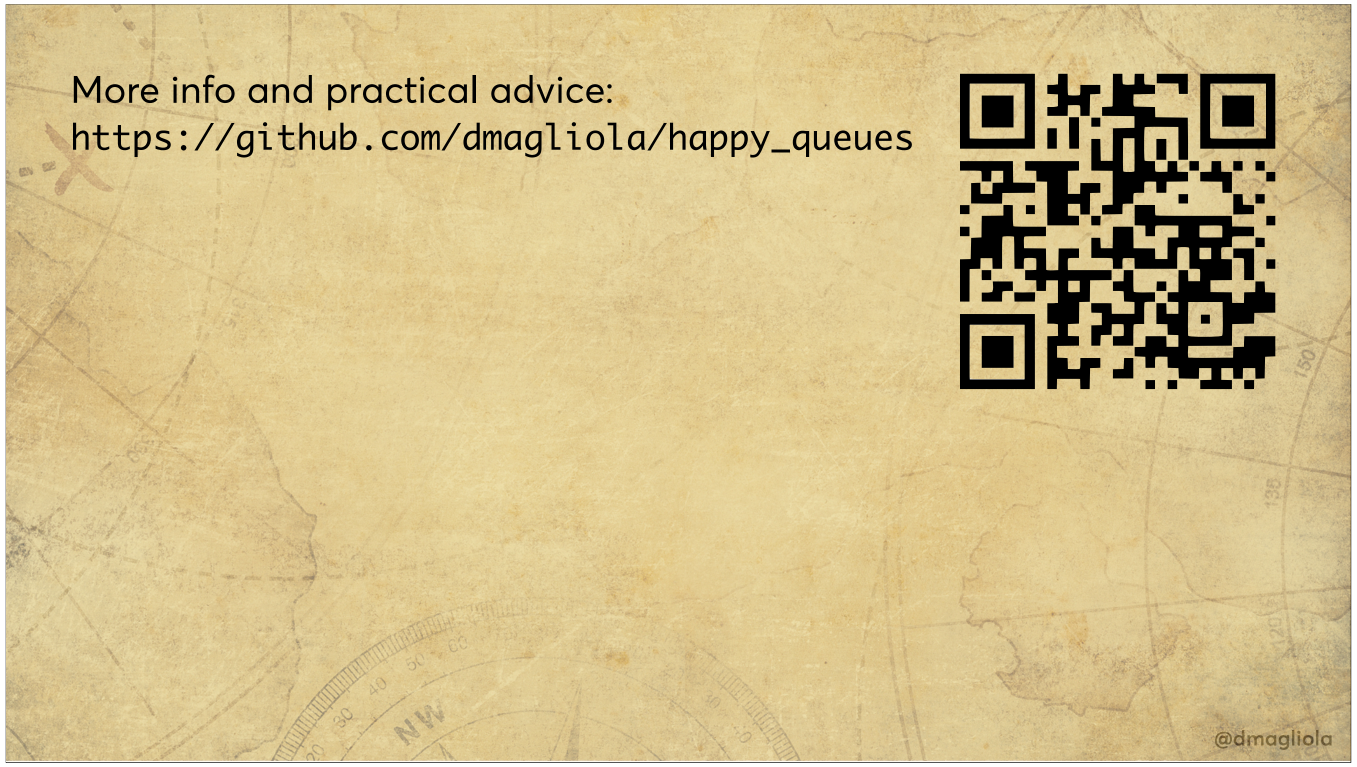
Giving them their own queue lets you make that distinction, and save on servers.

You see that these are all the usual trade-offs of money vs complexity. And as always, YMMV, but it's sometimes useful to stray from the path.

Just make sure all of your queues still have a clear latency guarantee in their name, and the usual alerts, and you're golden.

And that's how our brave team of developers solved all their queueing woes,
reached pure queue bliss,
and lived speedily ever after.

fin



More info and practical advice:
`https://github.com/dmagliola/happy_queues`

@dmagliola

Now before I go, I want to leave you with a couple of quick notes…

First, I just mentioned some of the most important lessons our friends learned while implementing this, but there are a lot more, and they don’t fit in half an hour. You can get more practical advice on how to do this on this repo.

More info and practical advice:
`https://github.com/dmagliola/happy_queues`

Thanks to:

Tekin Süleyman (@tekin)

Branch in Time

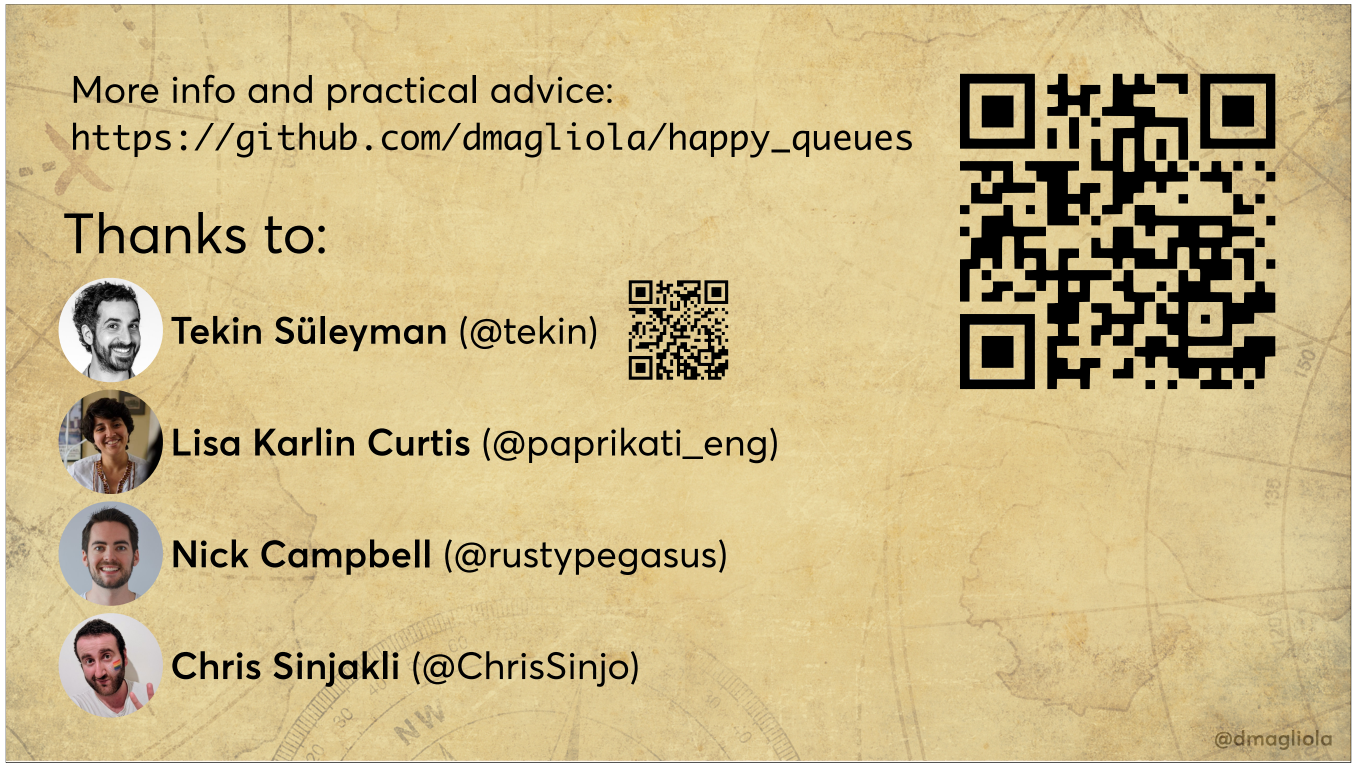
A story about revision histories

@tekin

@dmagliola

I also want to do some thanking:

First, to Tekin suleyman, who wrote one of the best talks I've ever seen, and whose style massively inspired this talk. If you haven't seen this, you should definitely watch it.



More info and practical advice:
https://github.com/dmagliola/happy_queues

Thanks to:

Tekin Süleyman (@tekin)

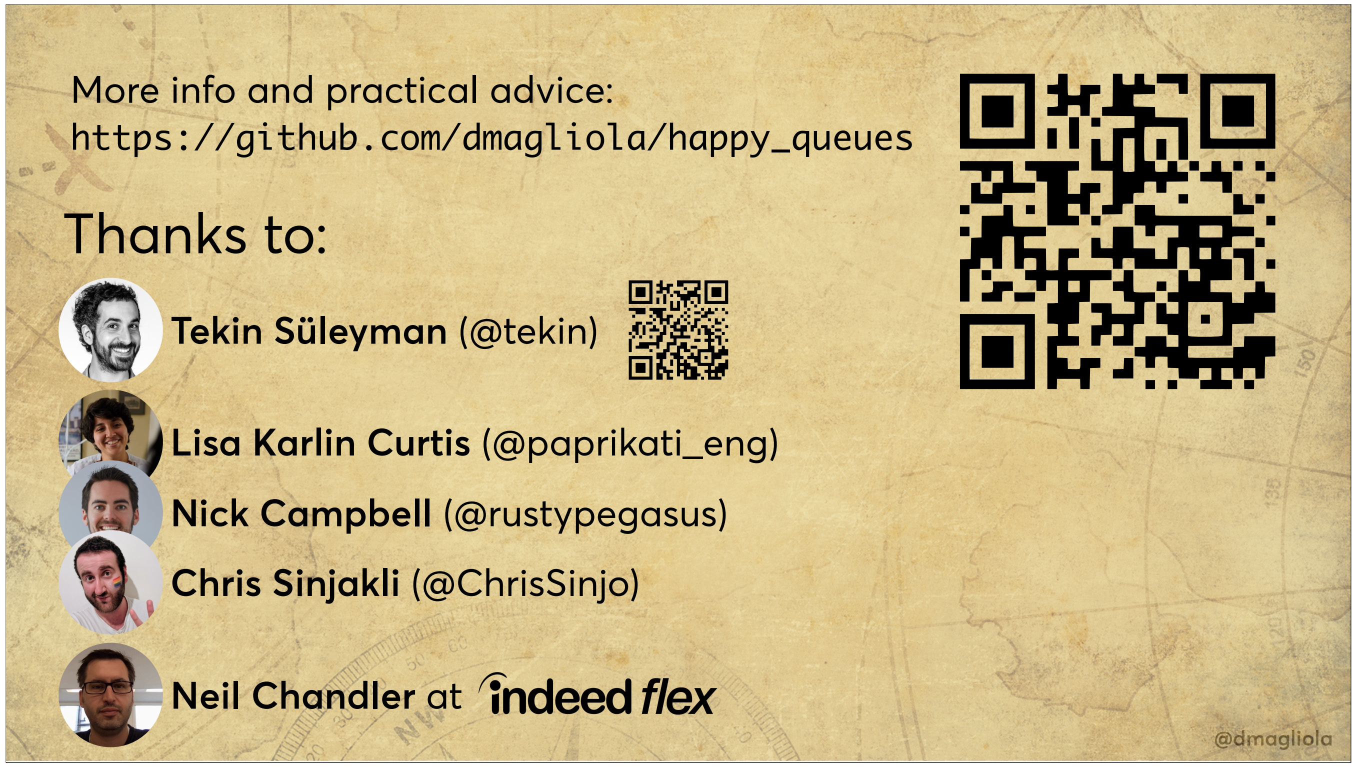
Lisa Karlin Curtis (@paprikati_eng)

Nick Campbell (@rustypegasus)

Chris Sinjakli (@ChrisSinjo)

@dmagliola

Also to my friends Nick, Lisa and Chris, for watching early versions of this talk and every other talk, and making them much better.



More info and practical advice:
`https://github.com/dmagliola/happy_queues`

Thanks to:

Tekin Süleyman (@tekin)

Lisa Karlin Curtis (@paprikati_eng)

Nick Campbell (@rustypegasus)

Chris Sinjakli (@ChrisSinjo)

Neil Chandler at indeed flex

@dmagliola

And last but not least to Neil Chandler, one of my teammates, who came up with most of those clever ideas on how to make this implementation tolerable.

And that's all from me, thank you, and i'll see you all in the hallway.

The secret to happy queues

Daniel Magliola @dmagliola indeed flex

RUBYCONF