The journey from (not) REST to GraphQL at scale

GraphQL meetup 22-10-2025

DEEZER

DEEZER

~10M subscribers

550 employees spread over France, Germany, United Kingdom, Brazil and USA

Deezer is available in 180+ countries and 26 languages

~800 servers in Paris + Google Cloud

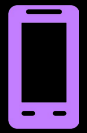
~200 network equipments (switches, routers, etc.)

More than 6 PB of storage needs

6 To of daily logs and 45M active metrics series

~4B requests per day

Desktop



Mobile

TV

Car

Connected Speakers

Wearables

~❯whoami_

Loïc Doubinine

Backend Developer

@Deezer since 2016

@ztec.fr

@ztec@mamot.fr

@ztec6

www.ztec.fr/en/social

blog.ztec.fr

DEEZER

The journey from (not) REST to GraphQL at scale

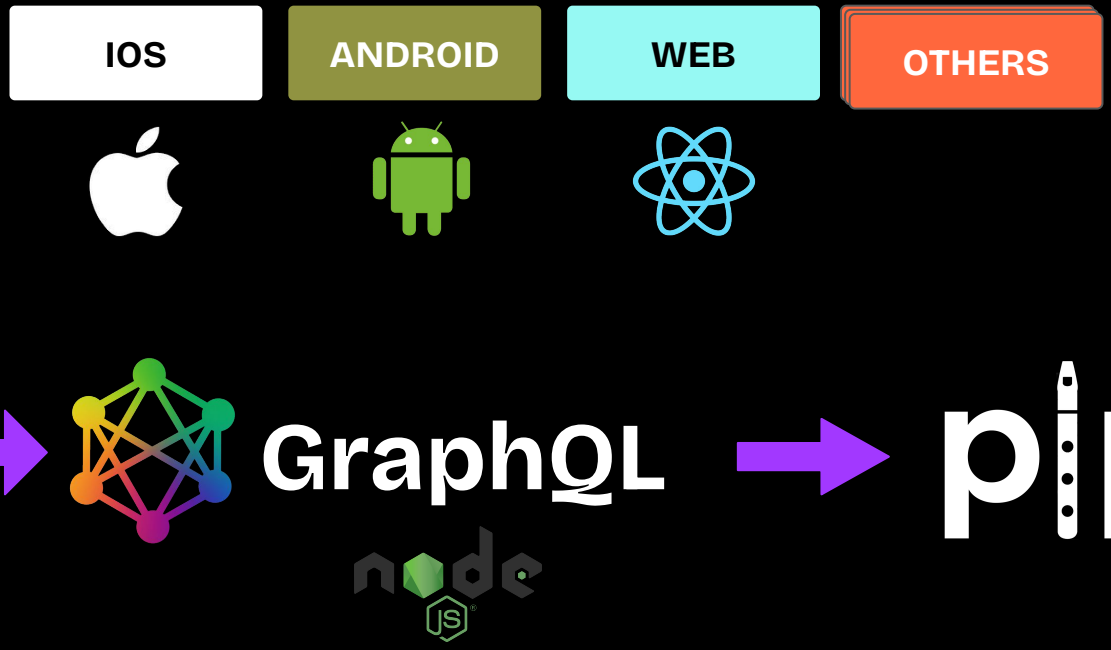
GraphQL meetup 22-10-2025

DEEZER

DEEZER

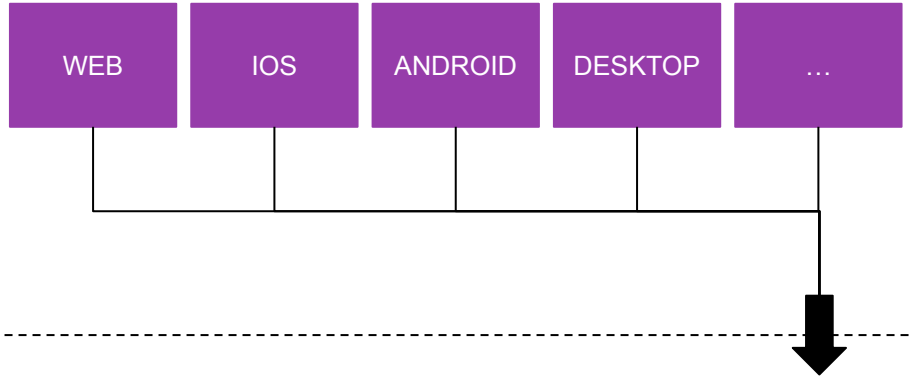
Context

Modernizing old APIs

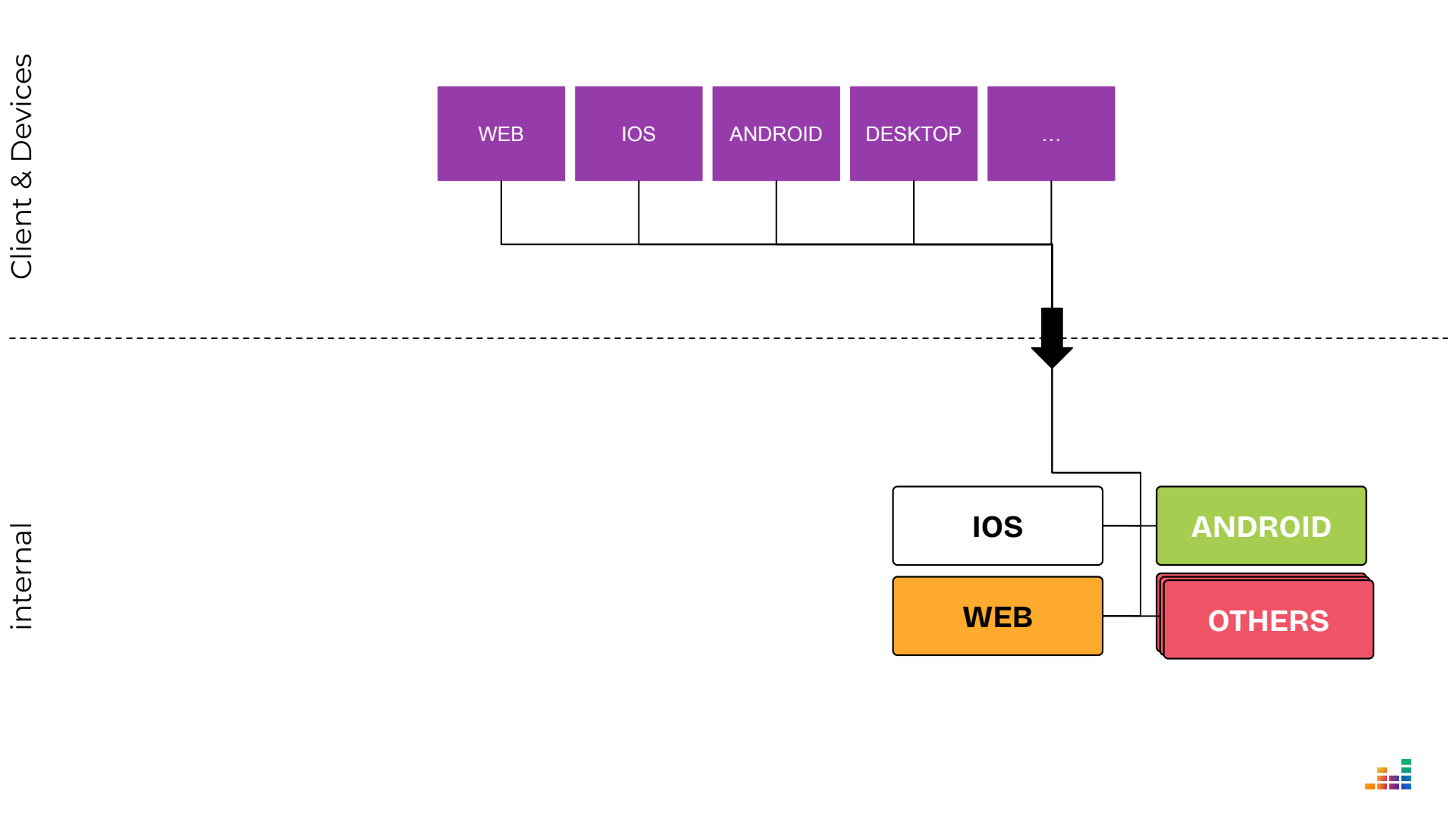


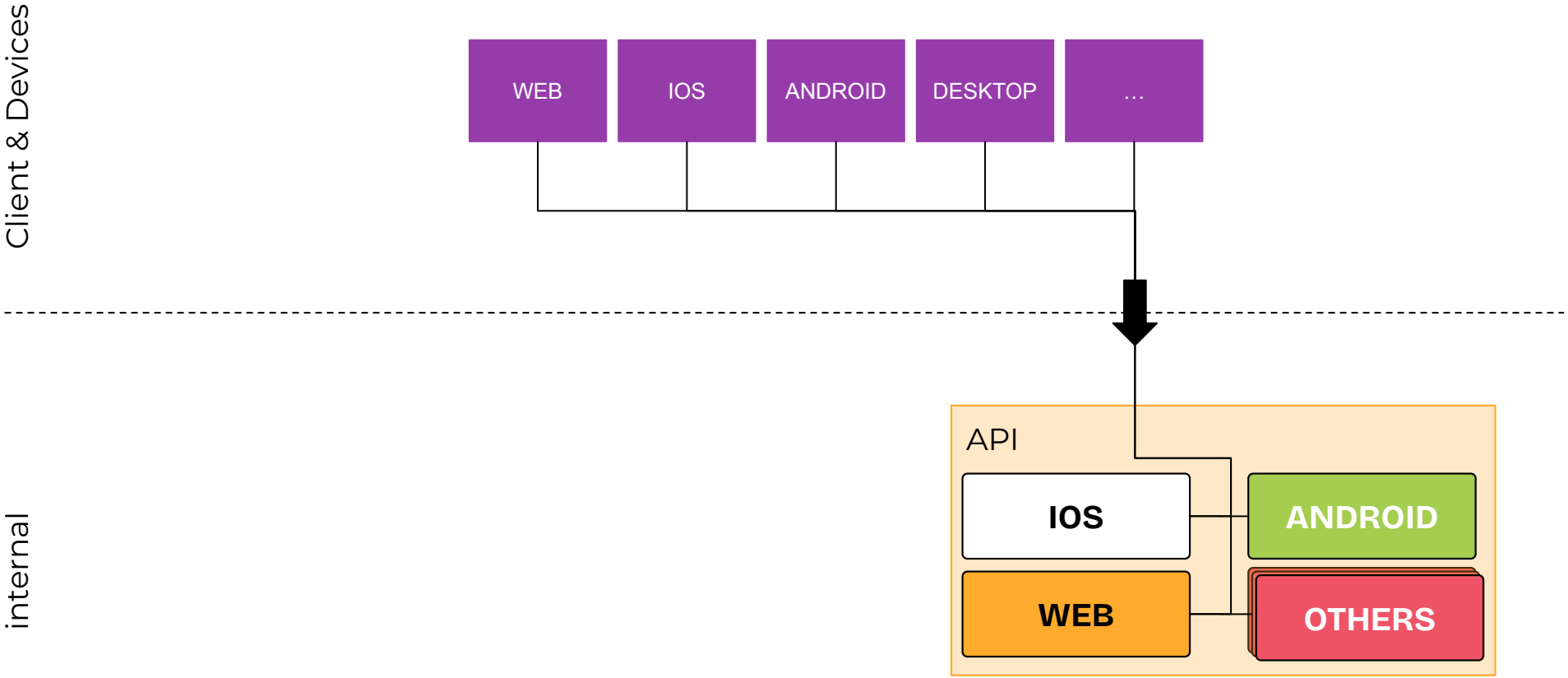

WEB
IOS
ANDROID
DESKTOP
…

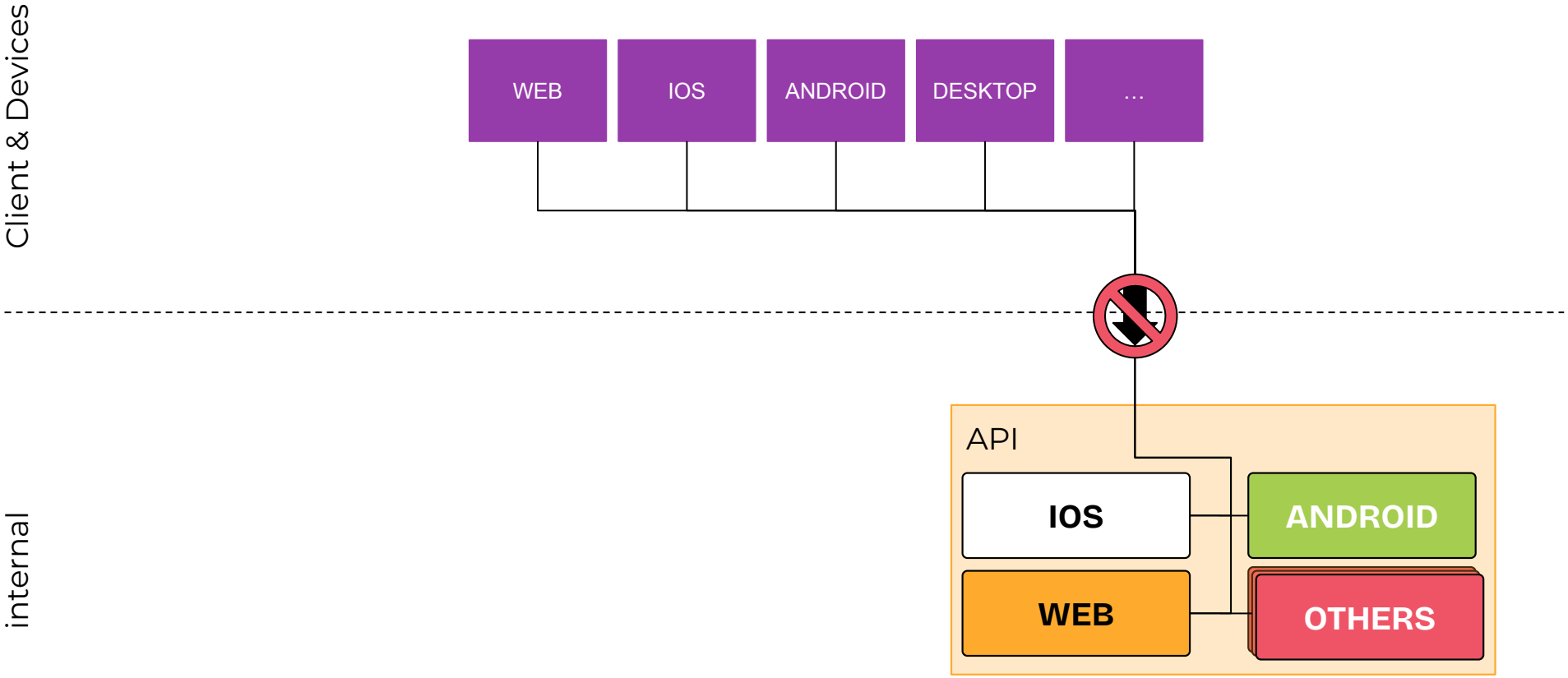
Client & Devices

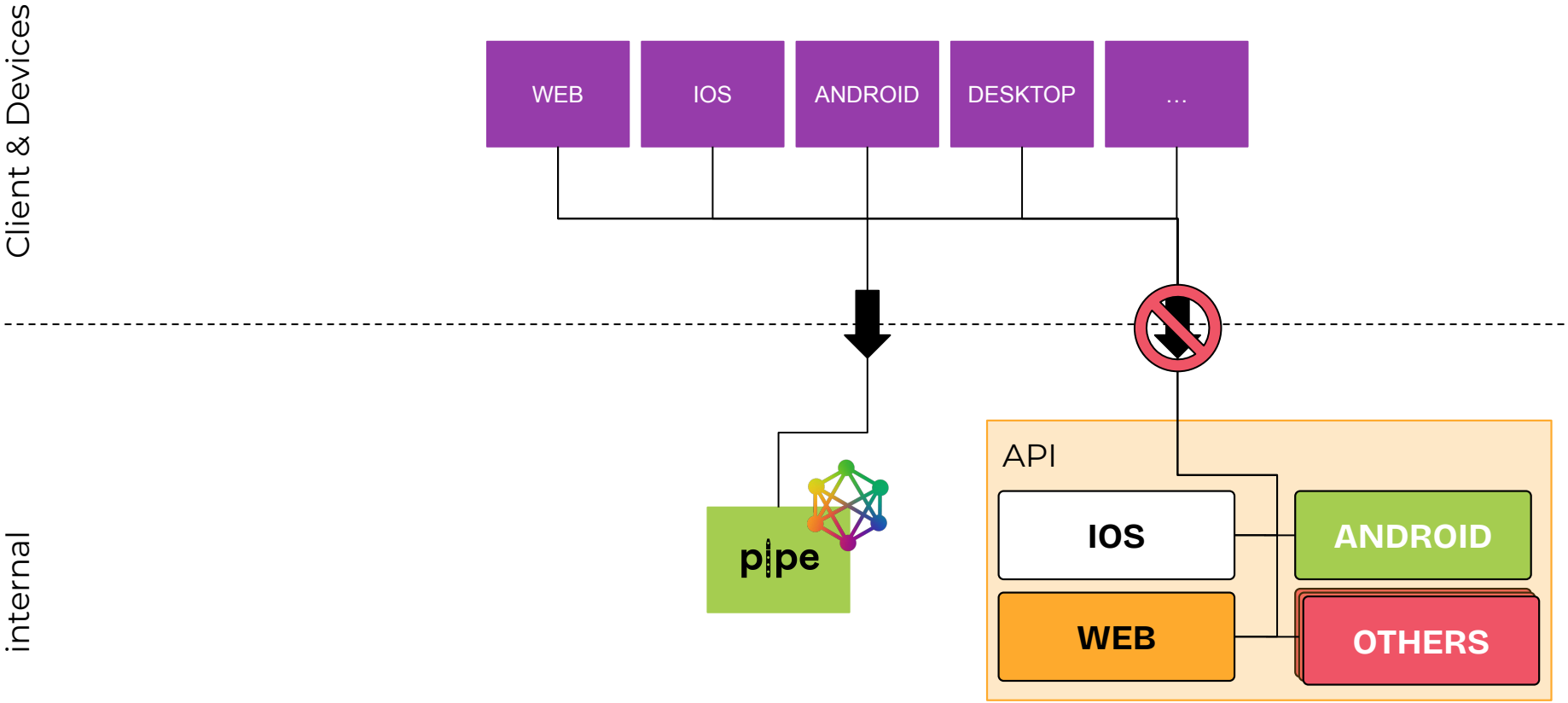


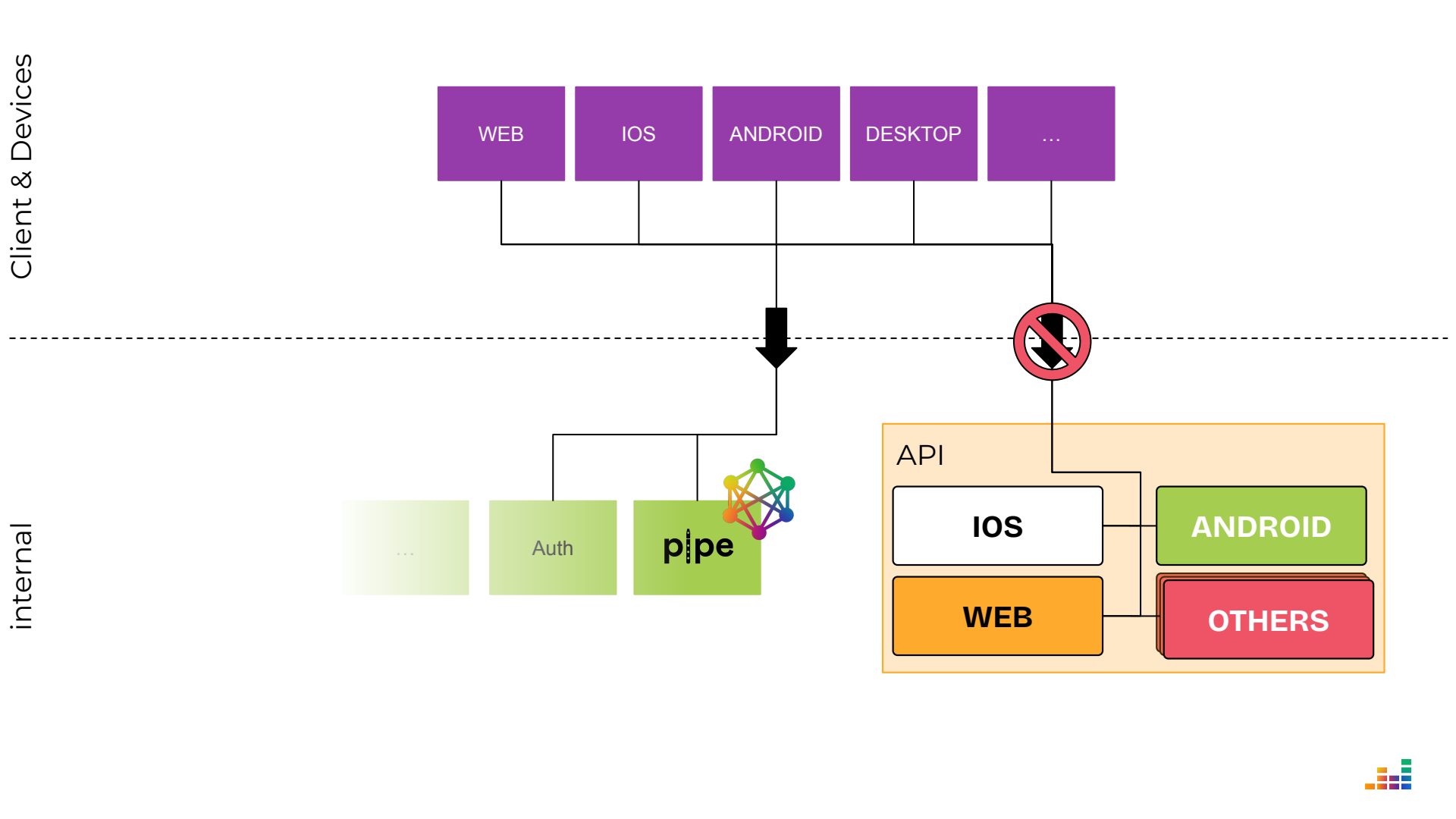
internal

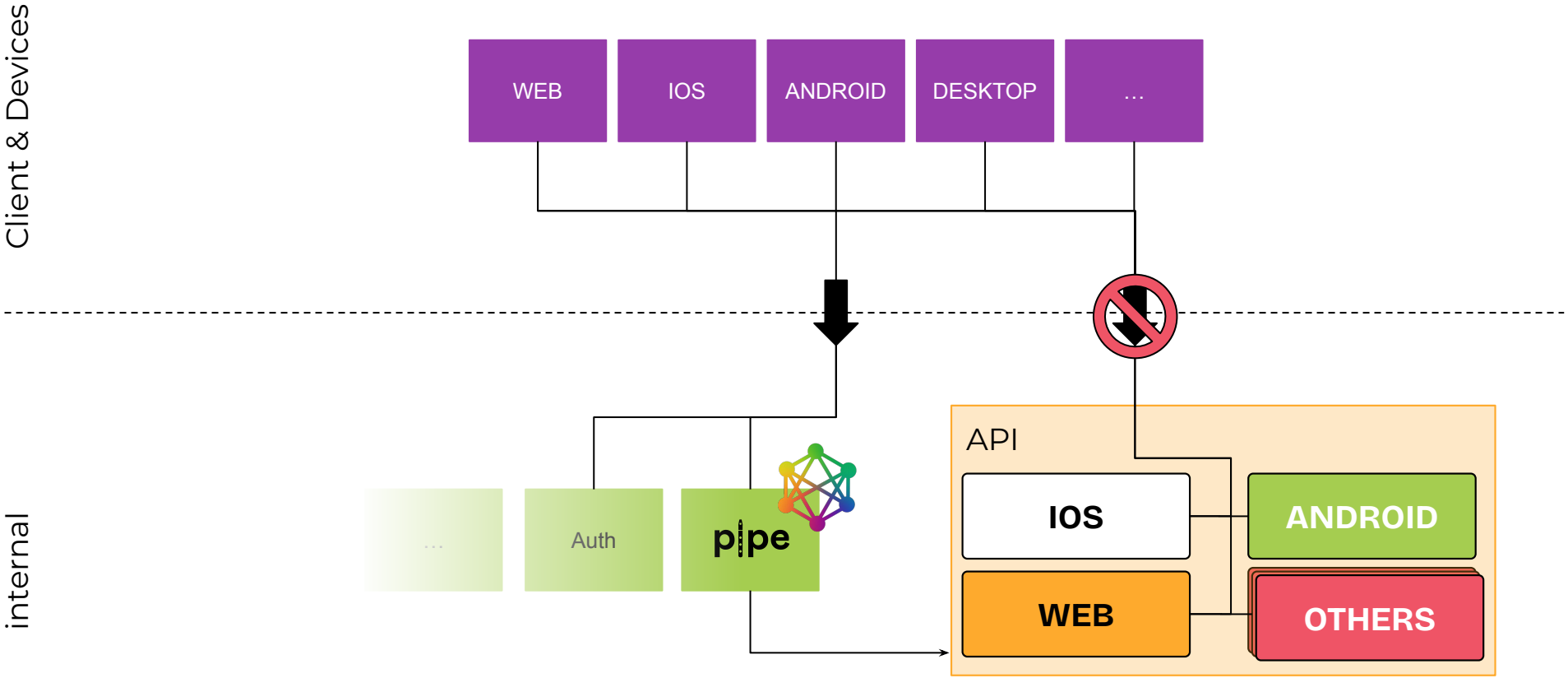

Client & Devices
WEB
IOS
ANDROID
DESKTOP
...
internal
IOS
ANDROID
WEB
OTHERS

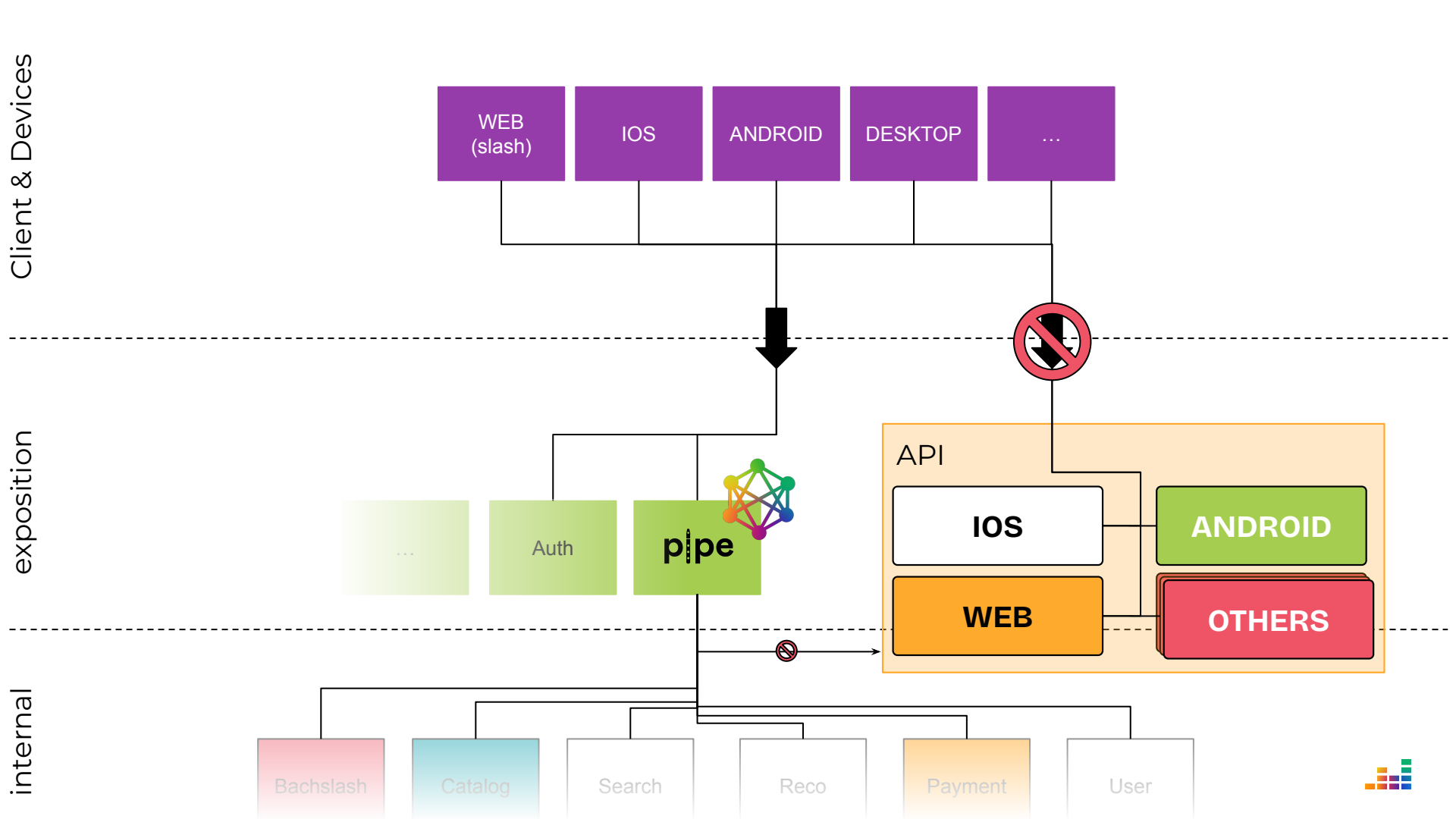

Client & Devices
WEB
IOS
ANDROID
DESKTOP
...
internal
API
IOS
ANDROID
WEB
OTHERS

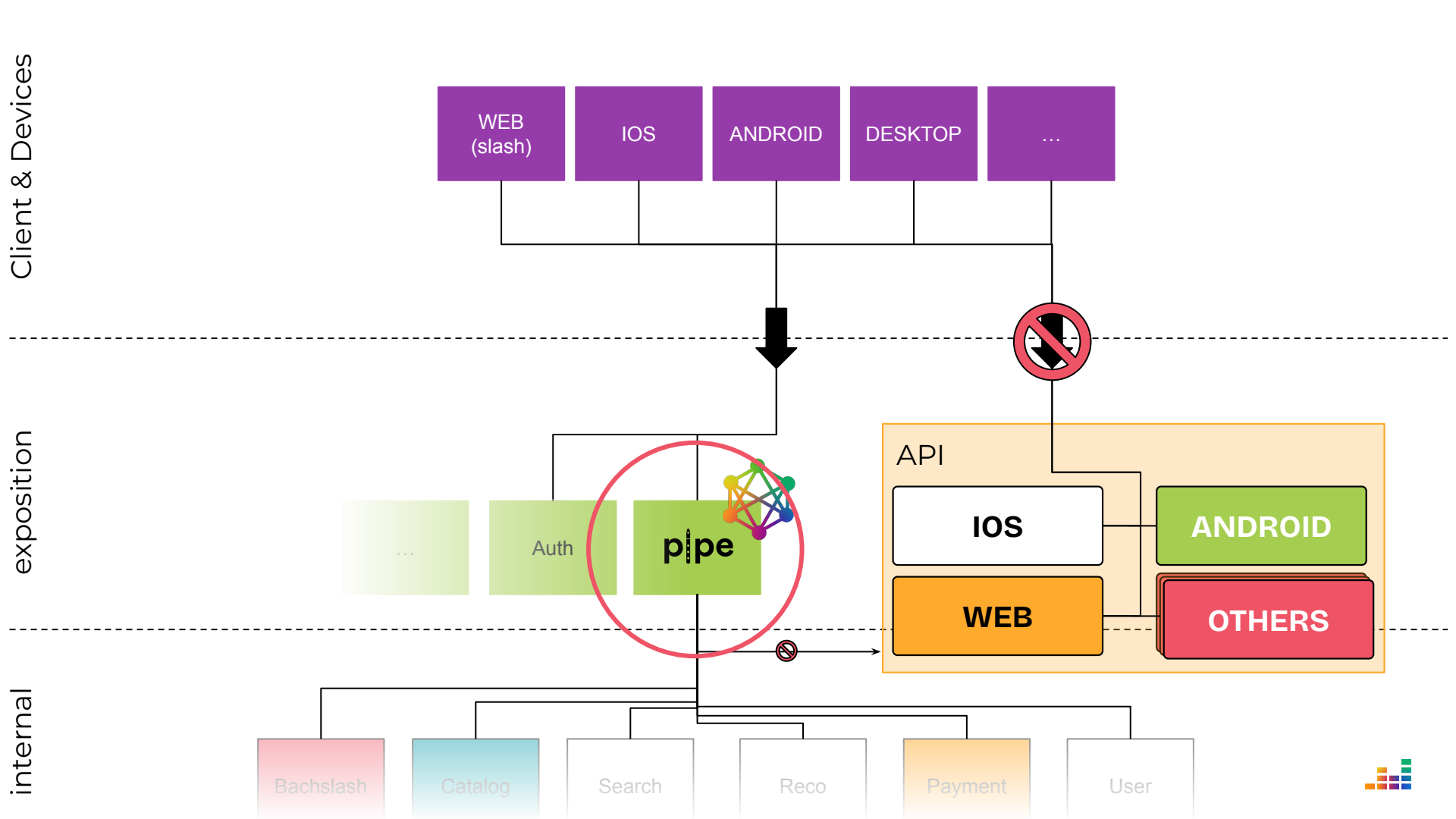

Client & Devices
WEB
IOS
ANDROID
DESKTOP
...
internal
API
IOS
ANDROID
WEB
OTHERS


Client & Devices
internal
WEB
IOS
ANDROID
DESKTOP
...
pipe
API
IOS
ANDROID
WEB
OTHERS


Client & Devices
internal
WEB
IOS
ANDROID
DESKTOP
...
Auth
pipe
API
IOS
ANDROID
WEB
OTHERS


Client & Devices
WEB
IOS
ANDROID
DESKTOP
...
internal
...
Auth
pipe
API
IOS
ANDROID
WEB
OTHERS

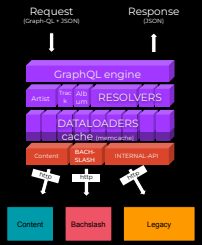

Client & Devices
exposition
internal
WEB (slash)
IOS
ANDROID
DESKTOP
...
API
IOS
ANDROID
WEB
OTHERS
...
Auth
pipe
Bachslash
Catalog
Search
Reco
Payment
User

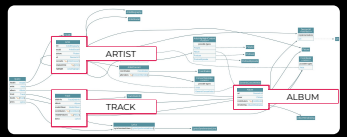

Client & Devices
exposition
internal
WEB (slash)
IOS
ANDROID
DESKTOP
...
API
IOS
ANDROID
WEB
OTHERS
...
Auth
pipe
Bachslash
Catalog
Search
Reco
Payment
User

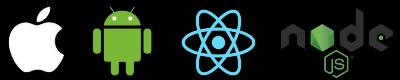
Building

Multiple phases

Bootstrap

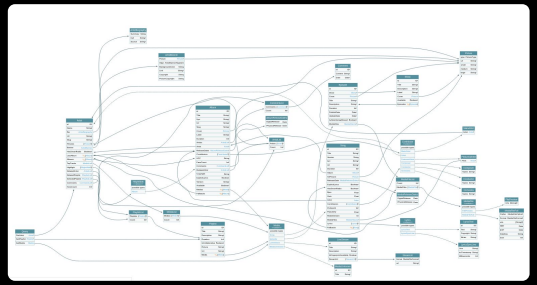






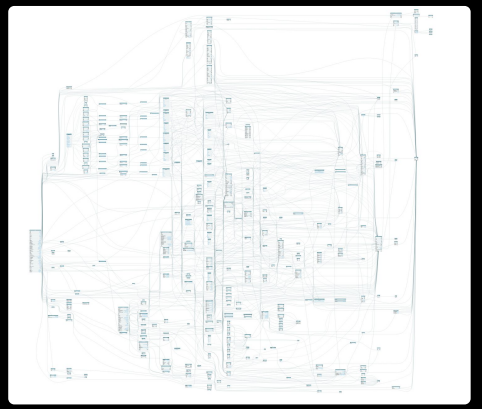
Multiple phases

Bootstrap **Spread**



Multiple phases

Bootstrap **Spread** **Cruise**



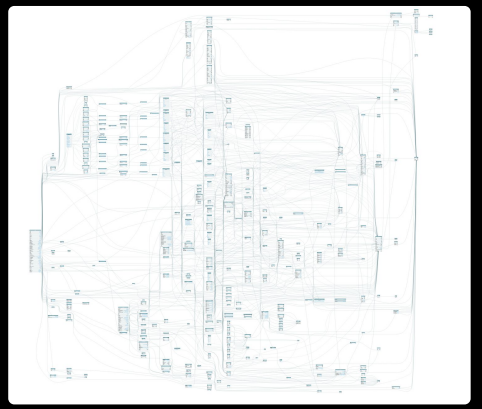
Multiple phases

NOW

Bootstrap

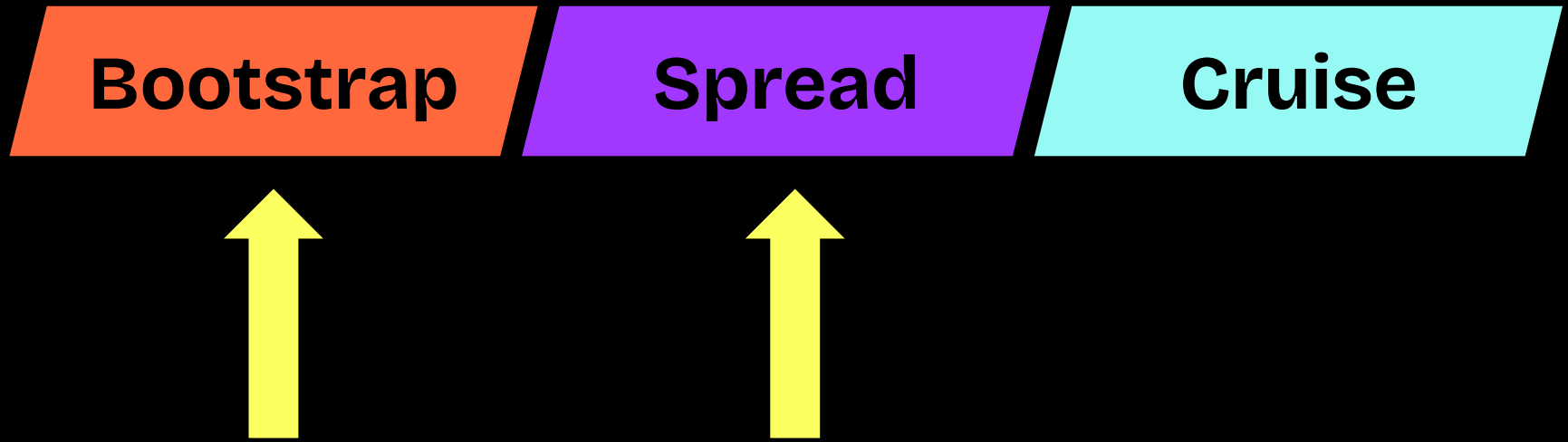
Spread

Cruise



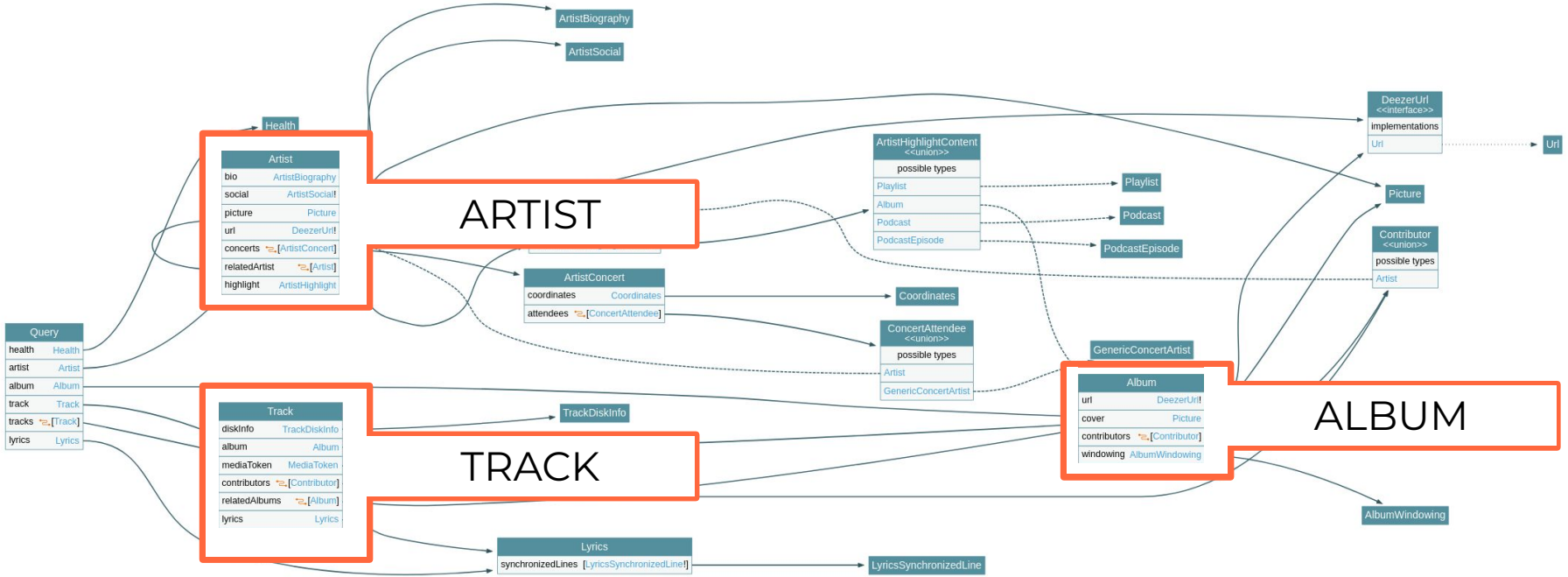
DEEZER

Multiple phases



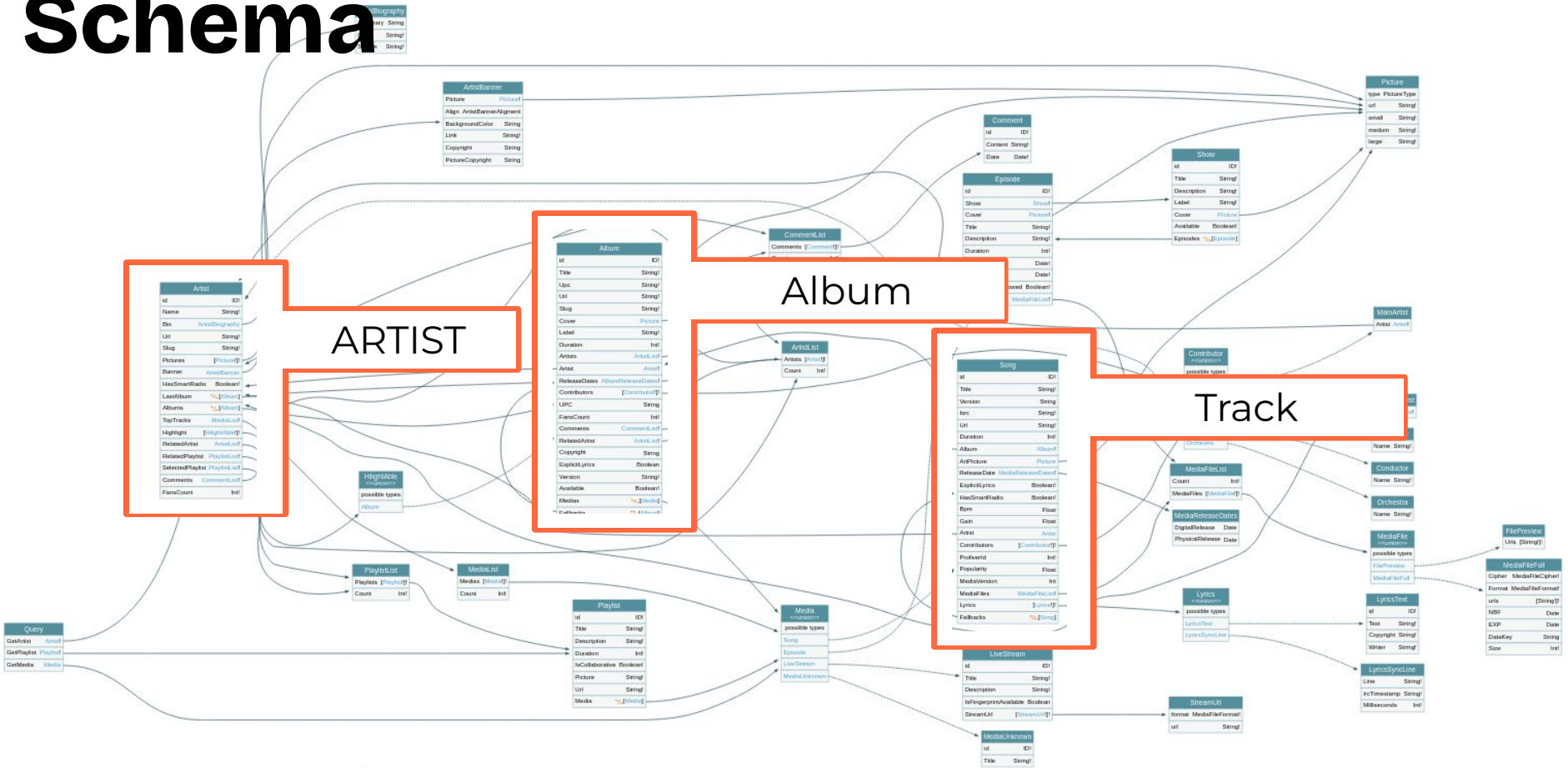
Schema

Schema



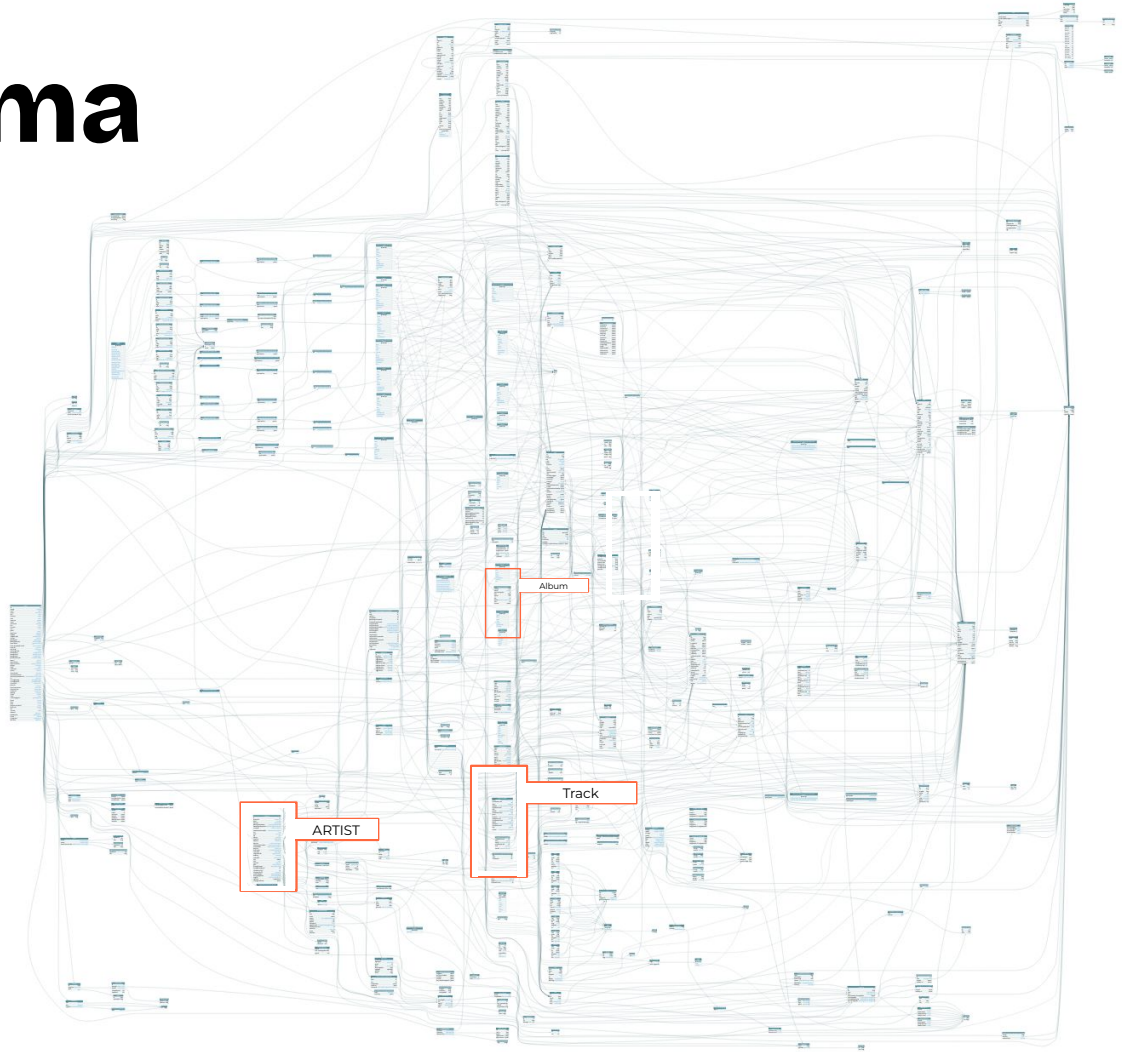
DEEZER

Schema



DEEZER

Schema



DEEZER

Backend architecture

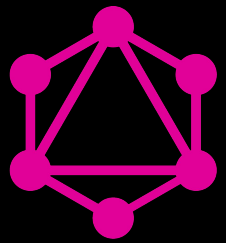
Backend architecture

- NodeJS 10 then 12 (we've upgraded since 😉)
- Typescript
- Express

- Standard GraphQL engine from Facebook

- Classic Resolver/Dataloader

- Gateway making HTTP calls internally
- (almost)No business logic



GraphQL

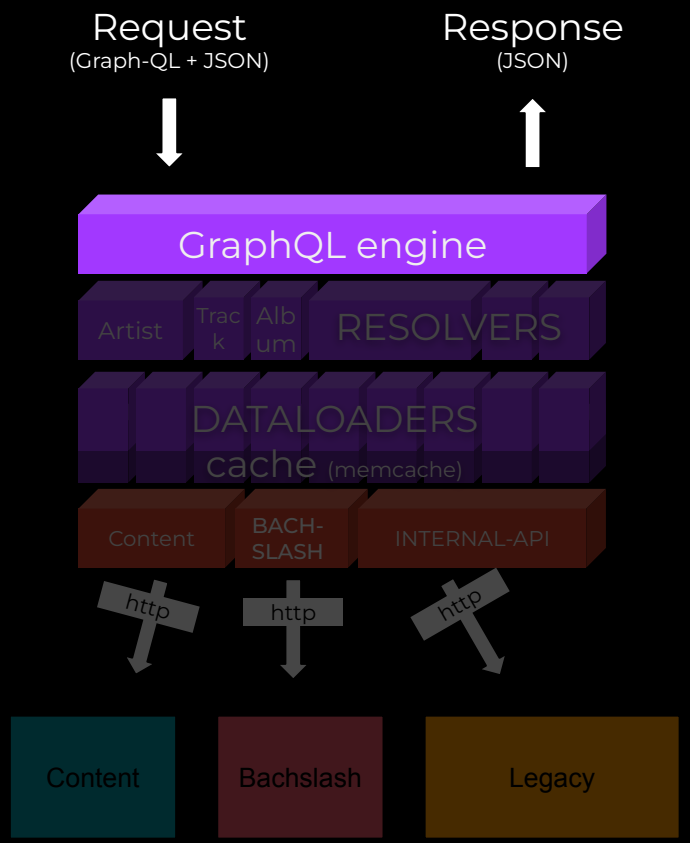
Backend architecture

- NodeJS 10 then 12 (we've upgraded since 😉)
- Typescript
- Express

- Standard GraphQL engine from Facebook

- Classic Resolver/Dataloader

- Gateway making HTTP calls internally
- (almost)No business logic



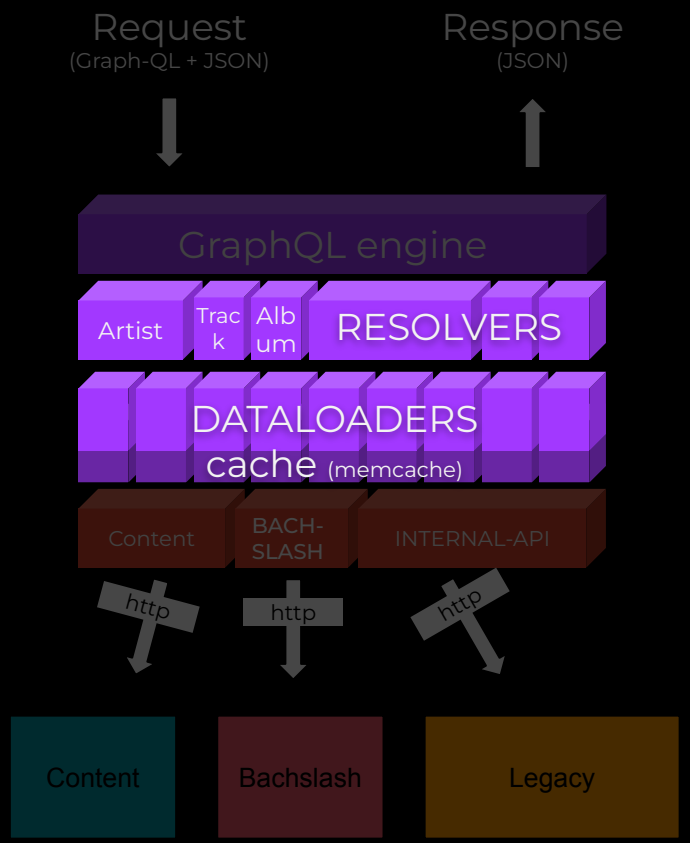
Backend architecture

- NodeJS 10 then 12 (we've upgraded since 😉)
- Typescript
- Express

- Standard GraphQL engine from Facebook

- Classic Resolver/Dataloader

- Gateway making HTTP calls internally
- (almost)No business logic



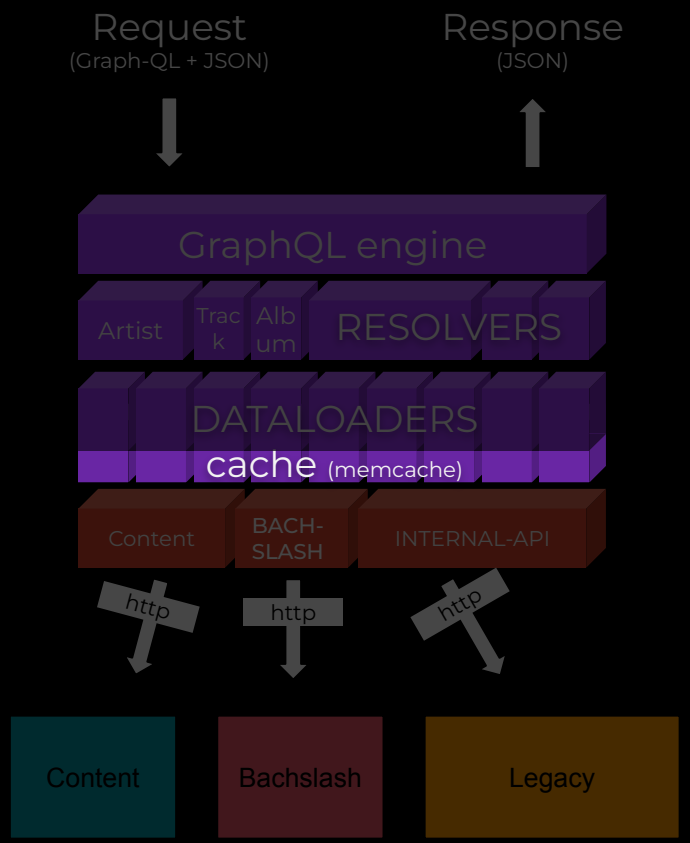
Backend architecture

- NodeJS 10 then 12 (we've upgraded since 😉)
- Typescript
- Express

- Standard GraphQL engine from Facebook

- Classic Resolver/Dataloader

- Gateway making HTTP calls internally
- (almost)No business logic



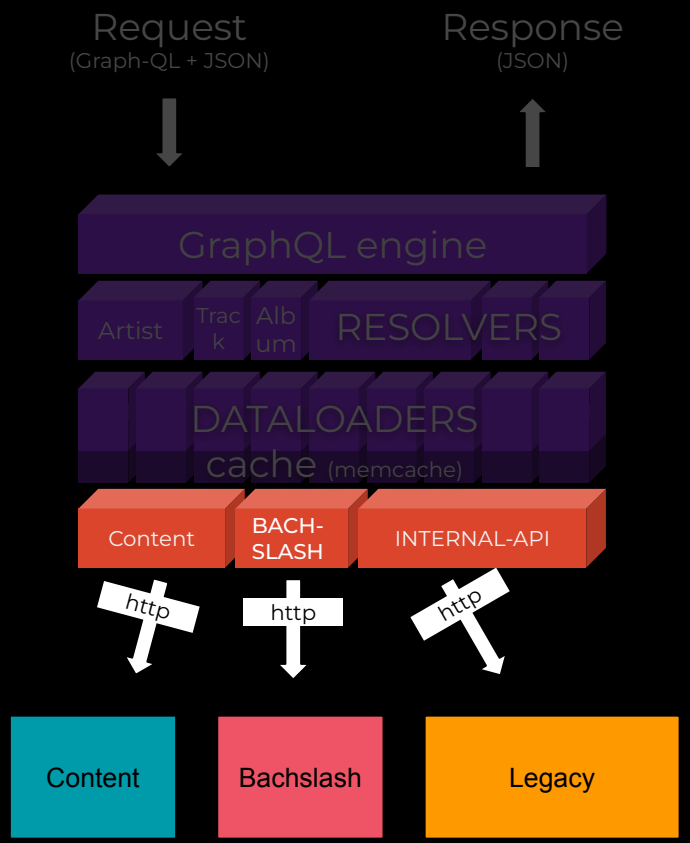
Backend architecture

- NodeJS 10 then 12 (we've upgraded since 😉)
- Typescript
- Express

- Standard GraphQL engine from Facebook

- Classic Resolver/Dataloader

- Gateway making HTTP calls internally
- (almost)No business logic



Performances?

How fast was it?

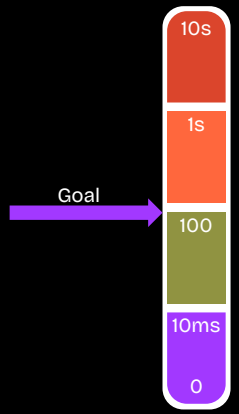
To load an artist

To load an album

To load 50 tracks

How fast was it?

To load an artist

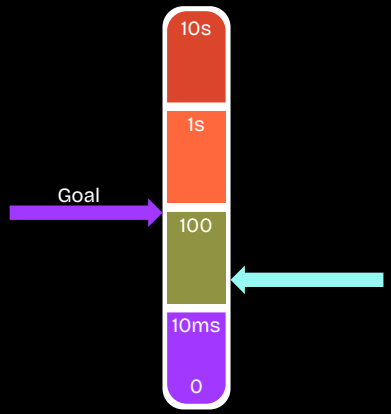


To load an album

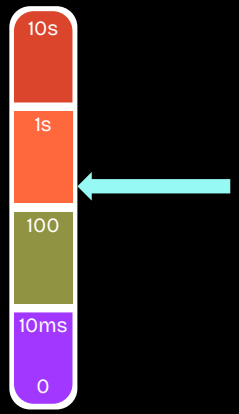
To load 50 tracks

How fast was it?

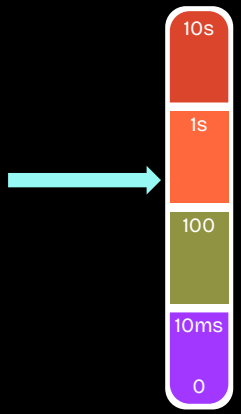
To load an artist



To load an album

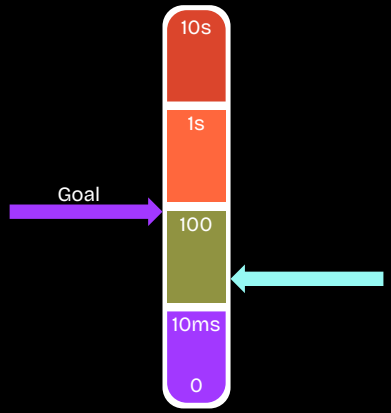


To load 50 tracks

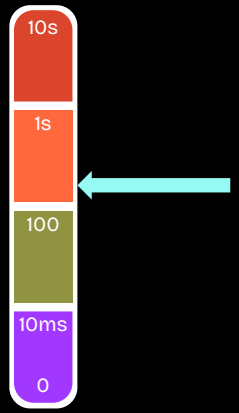


How fast was it?

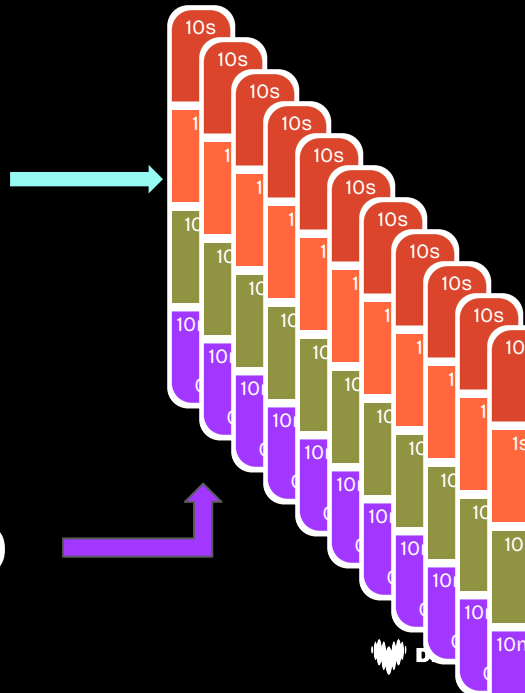
To load an artist



To load an album



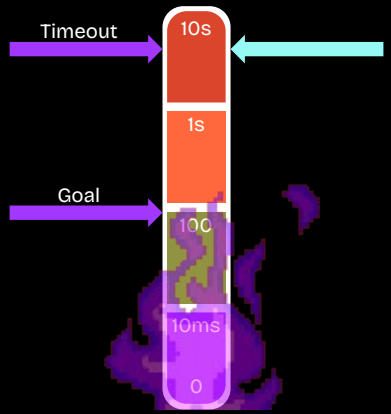
To load 50 tracks



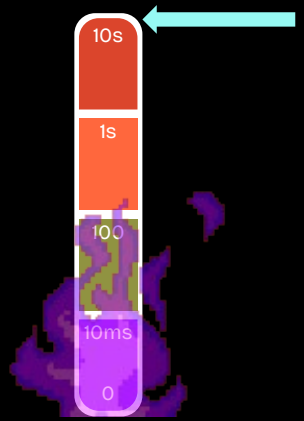
Loading a playlist of 1,000 tracks → **x20**

How fast was it?

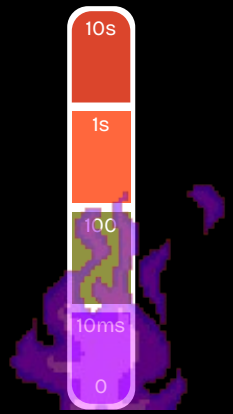
To load an artist



To load an album



To load 50 tracks

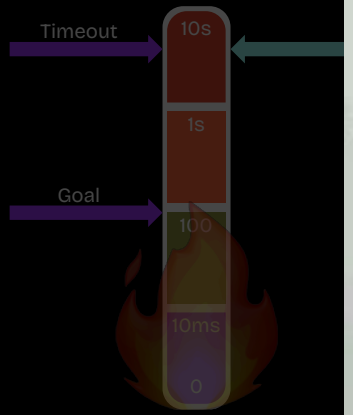


Under (not so)**serious load**

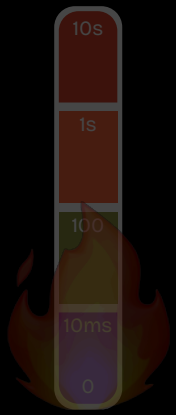
DEEZER

How fast was it ?

To load an Arti



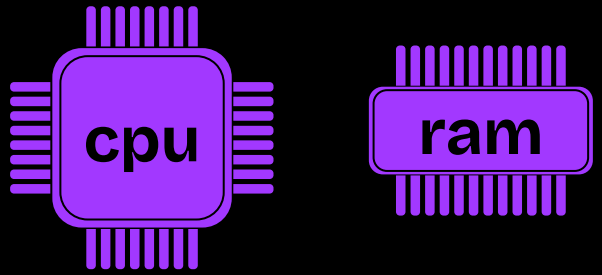
oad 50 tracks



Under (not so)serious load

How hungry is it?

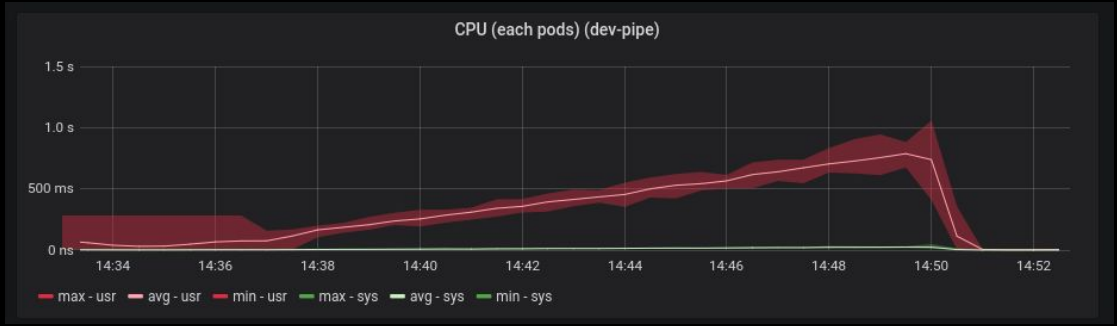
How many servers do I need to
ask the Infra team to buy or re-use?



How hungry is it?

CPU

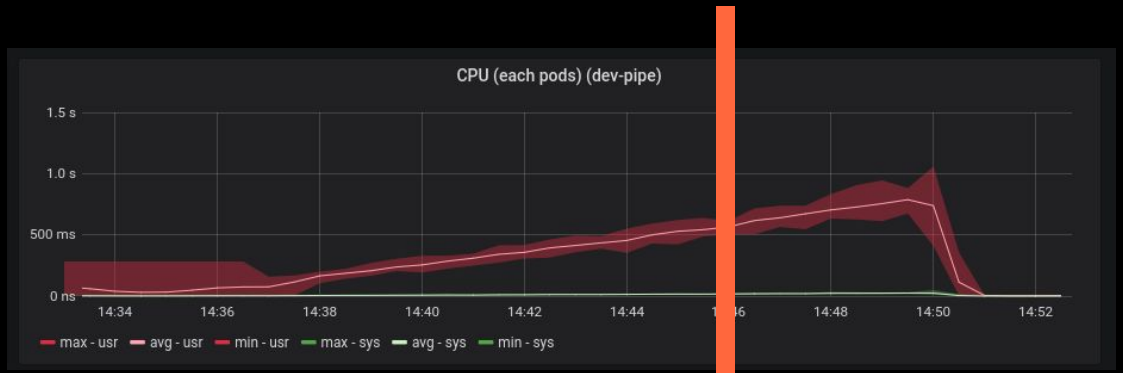
What resources to monitor?



How hungry is it?

CPU

What resources to monitor?

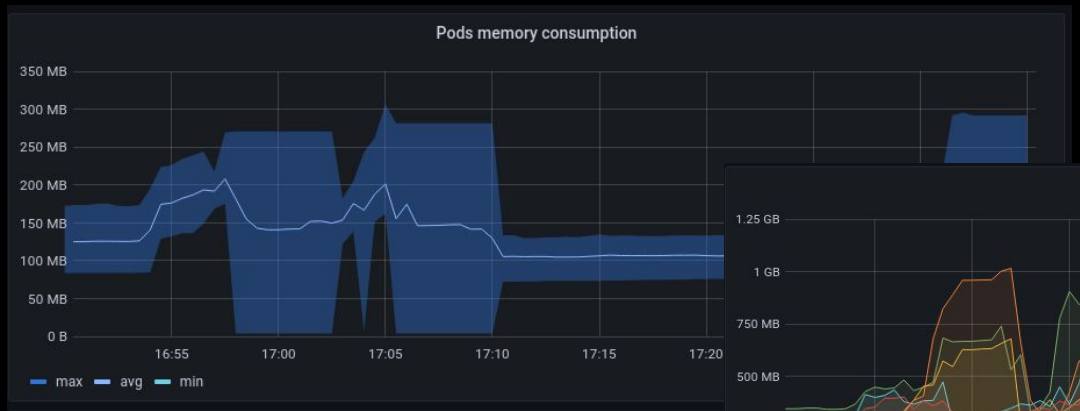


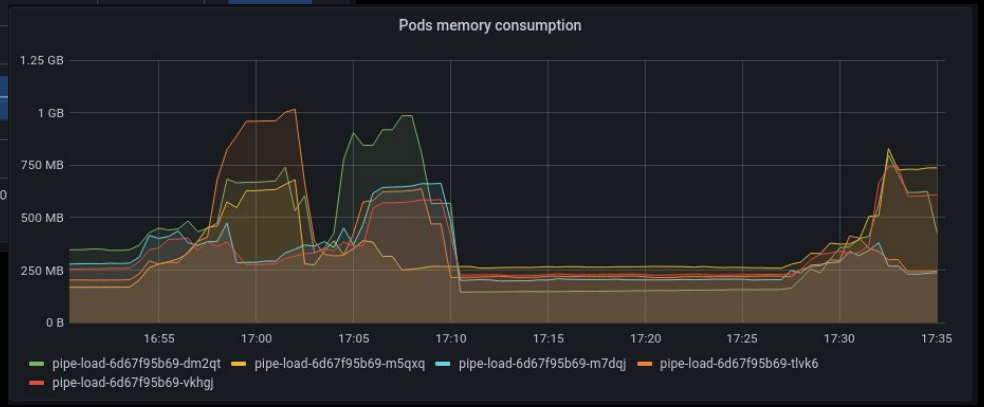
response time becomes bad

How hungry is it?

What resources to monitor?

CPU
RAM



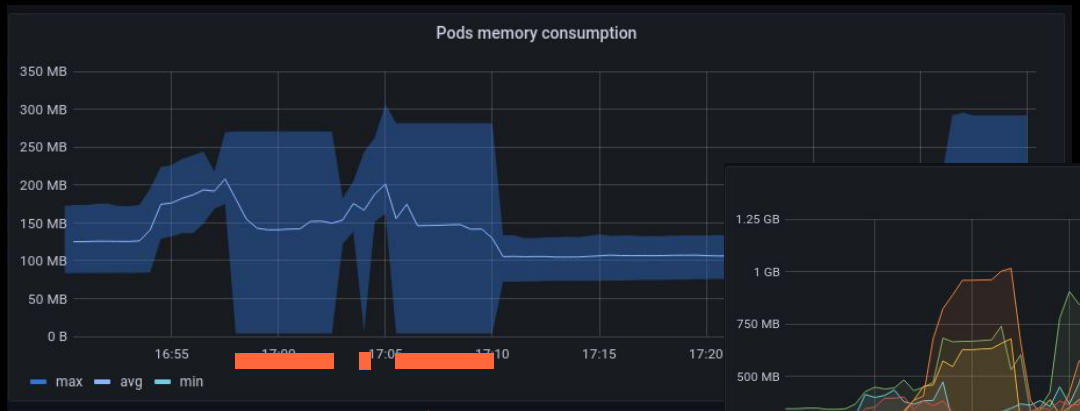


DEEZER

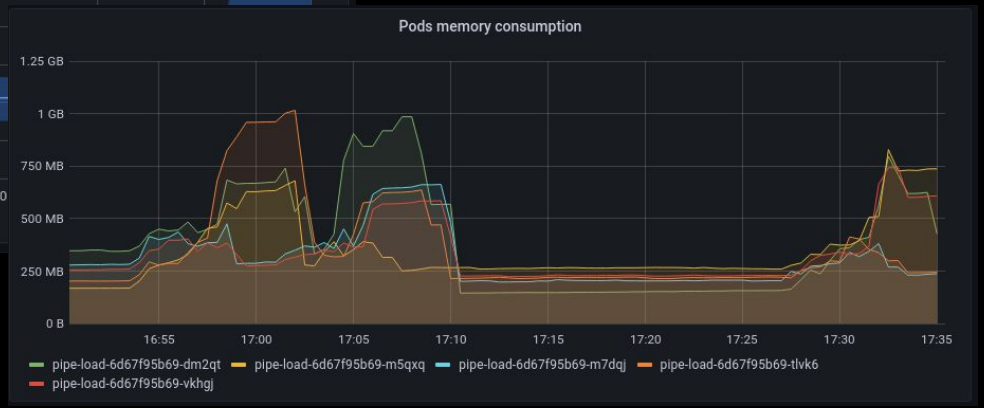
How hungry is it?

What resources to monitor?

CPU
RAM



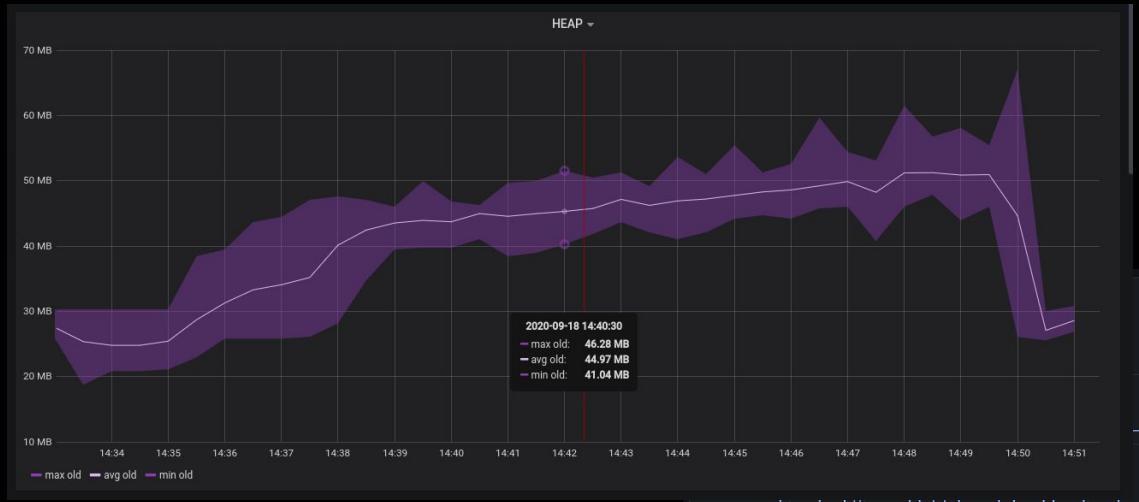
Crash



DEEZER

How hungry is it?

What resources to monitor?



CPU

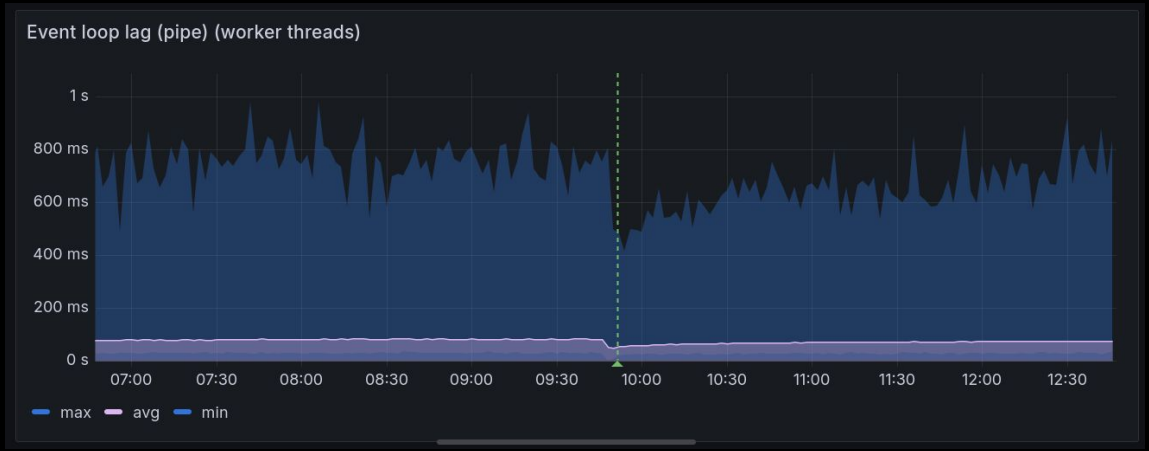
RAM

HEAP

DEEZER

How hungry is it?

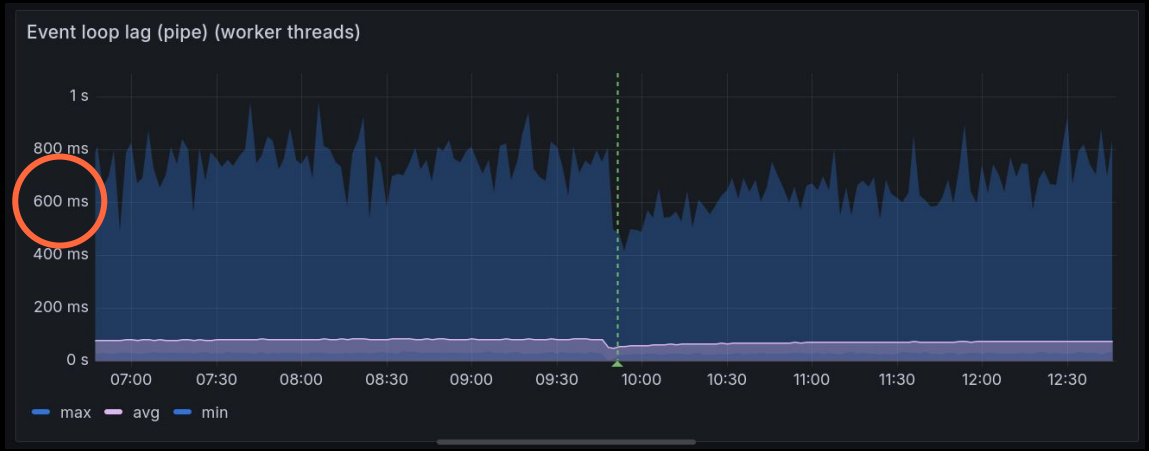
What resources to monitor?



CPU	
RAM	
HEAP	
Event Loop lag	

How hungry is it?

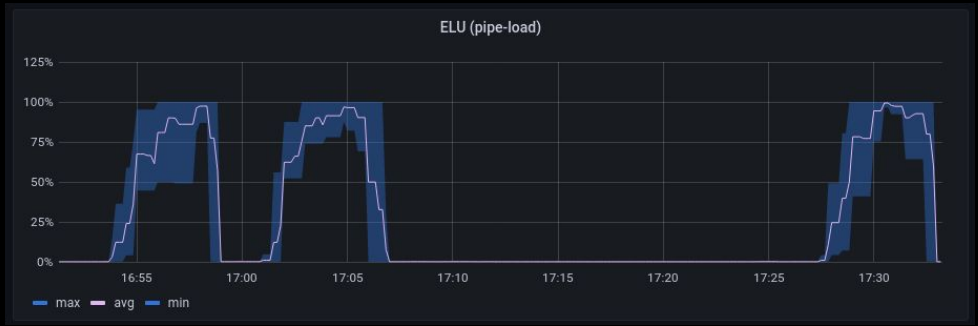
What resources to monitor?

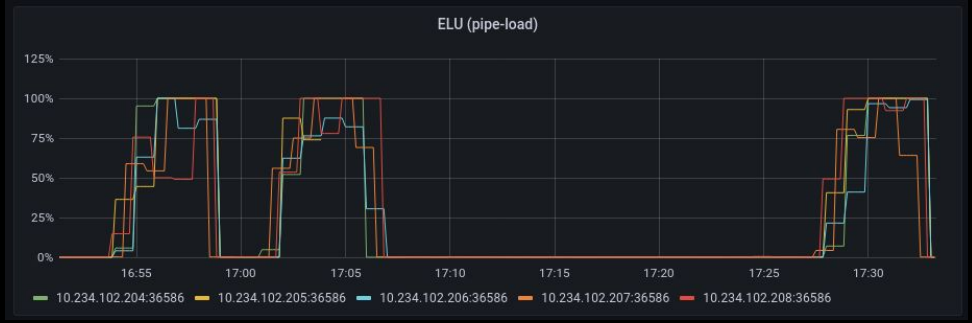


CPU	
RAM	
HEAP	
Event Loop lag	

How hungry is it?

What resources to monitor?

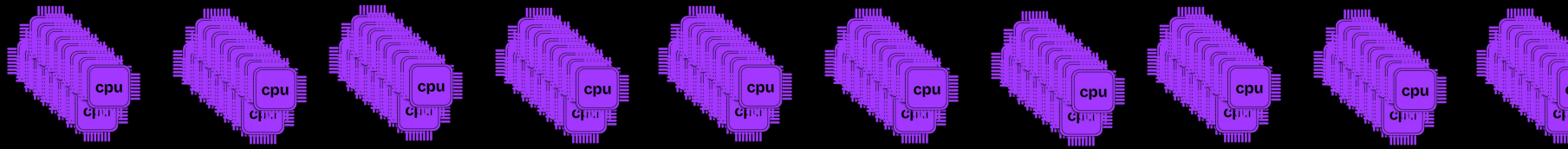




CPU	
RAM	
HEAP	
Event Loop lag	
Event Loop Utilization	

DEEZER

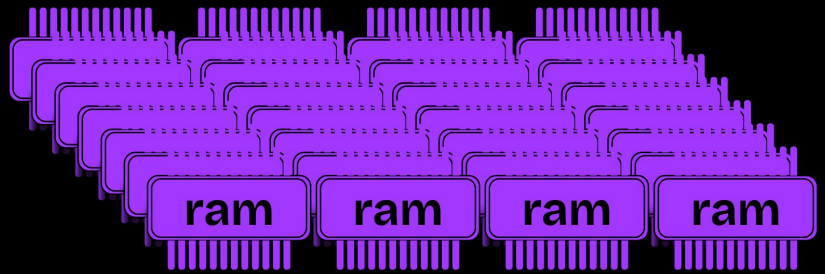
Initial performances insight



More CPUs than the entire fleet of existing servers

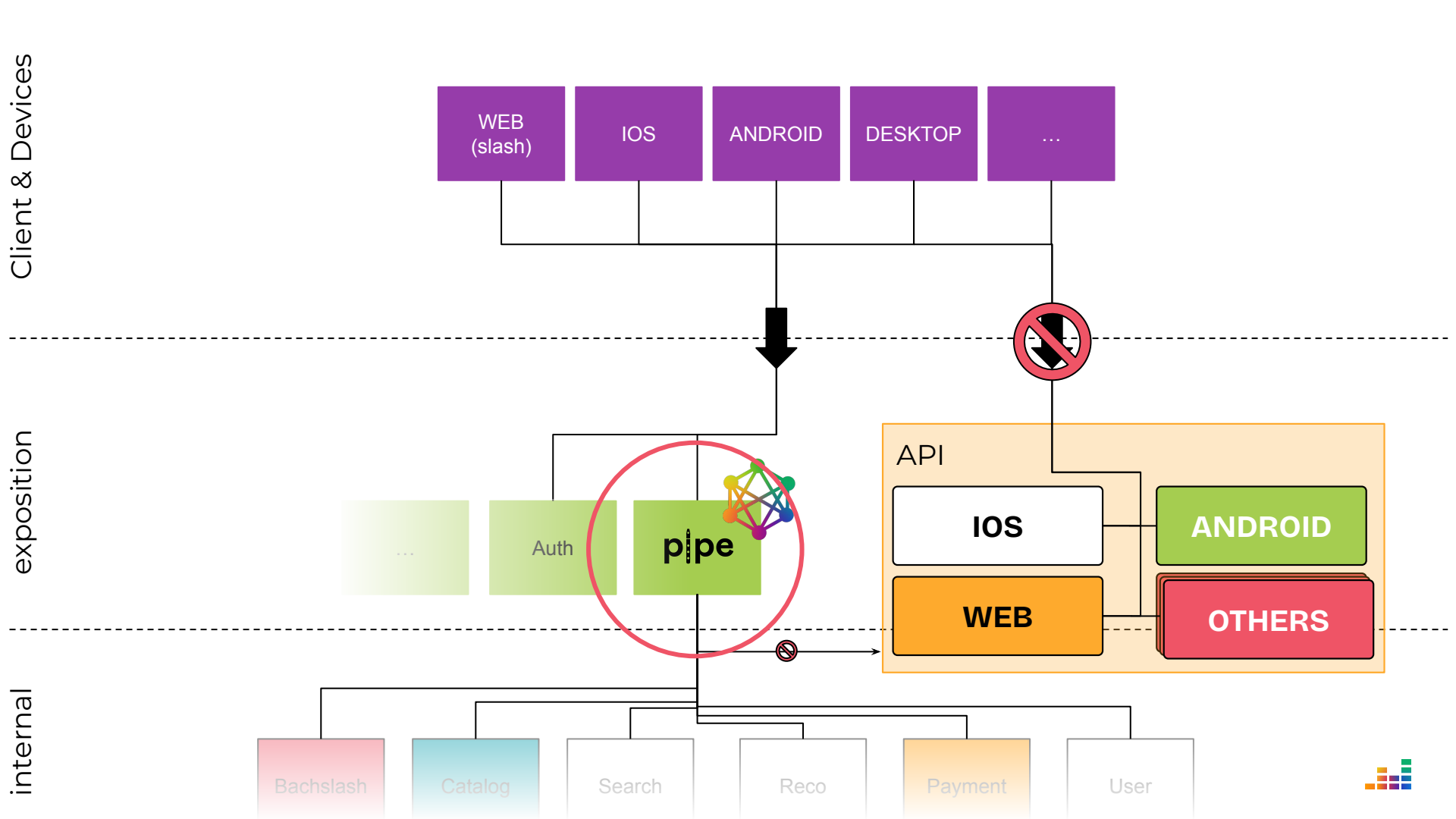
&

Half the RAM of existing servers



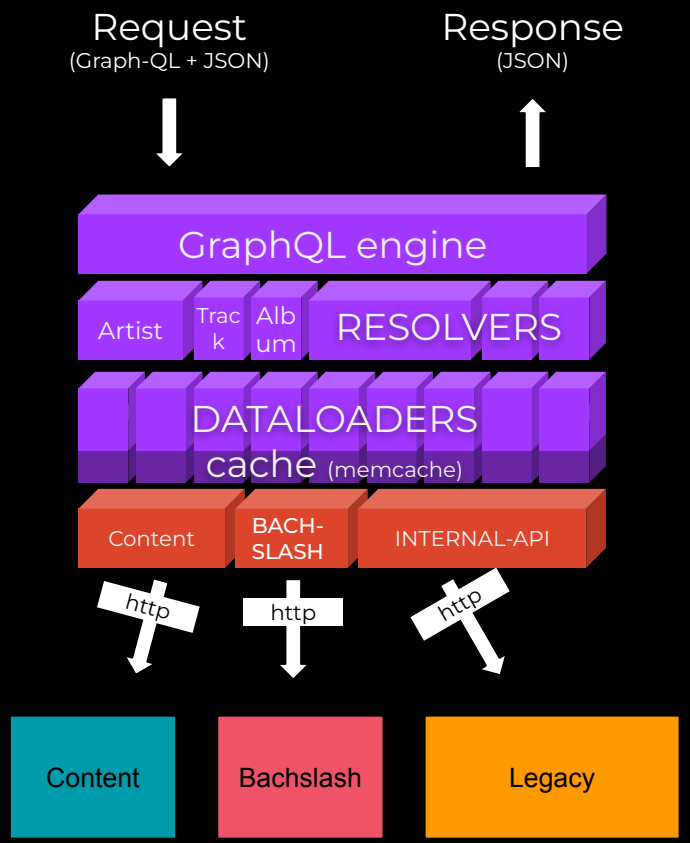
DEEZER

Initial performances insight

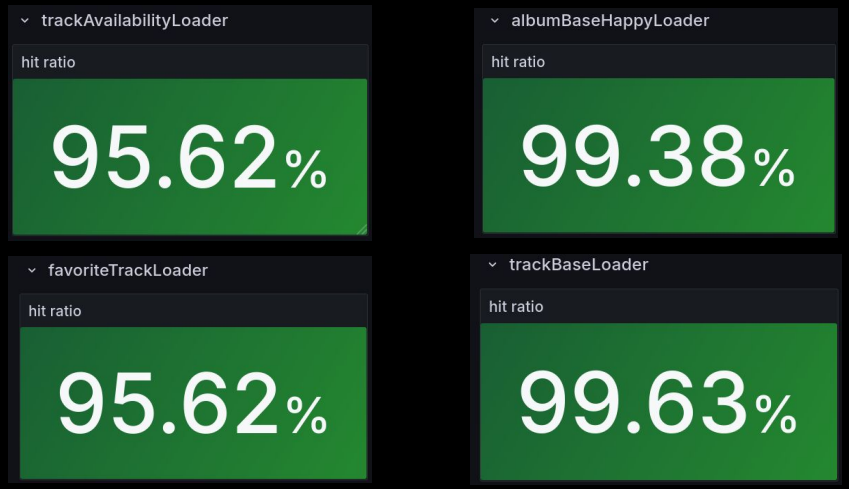

Client & Devices
exposition
internal
WEB (slash)
IOS
ANDROID
DESKTOP
...
API
IOS
ANDROID
WEB
OTHERS
...
Auth
pipe
Bachslash
Catalog
Search
Reco
Payment
User

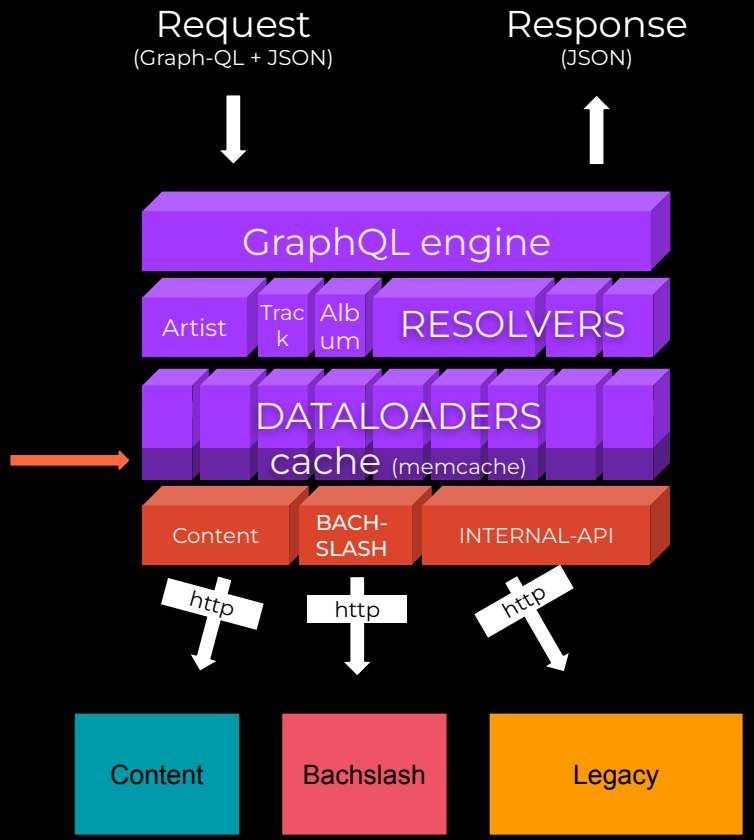
Optimizing

Where to optimize?



Where to optimize?

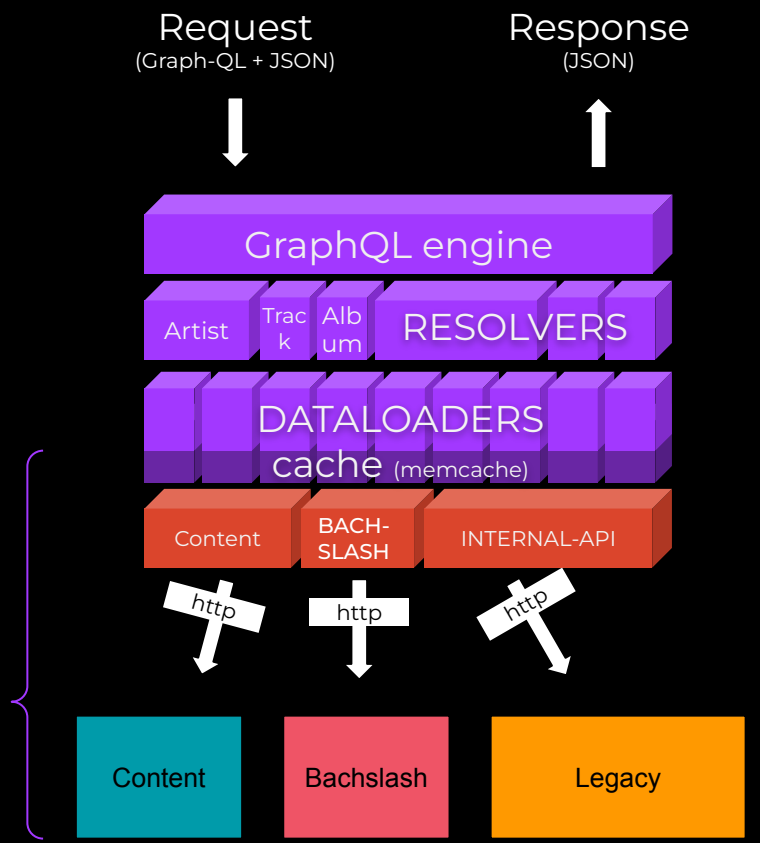




DEEZER

Where to optimize?

Cache allow us to skip the bottom part most of the time



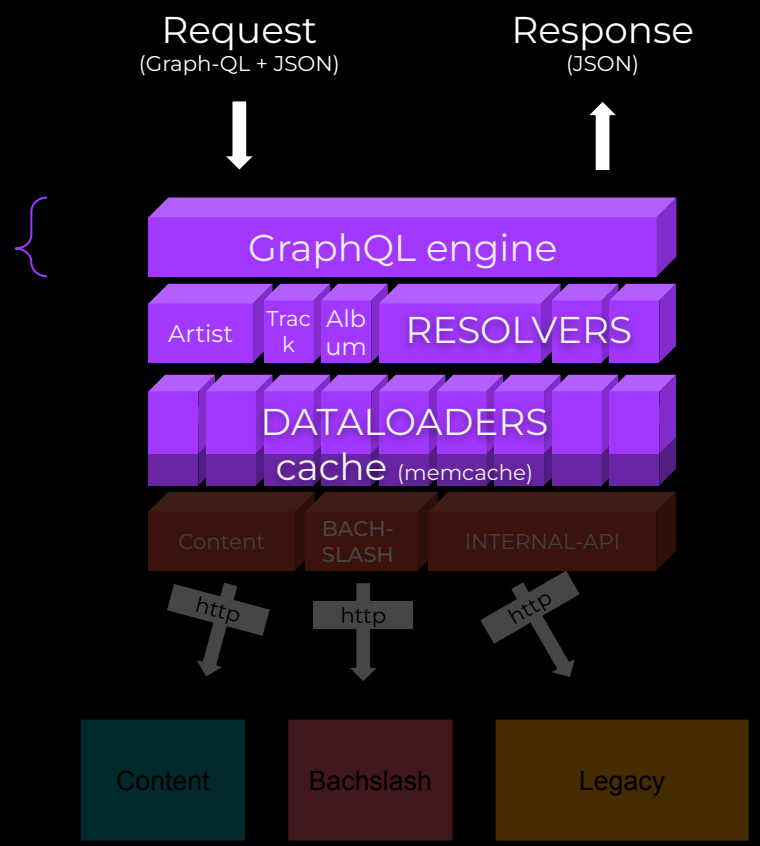
Where to optimize?

GraphQL engine is not ours

- GraphQL JIT was promising and we tested it.

  Turns out it is not worth it for us.

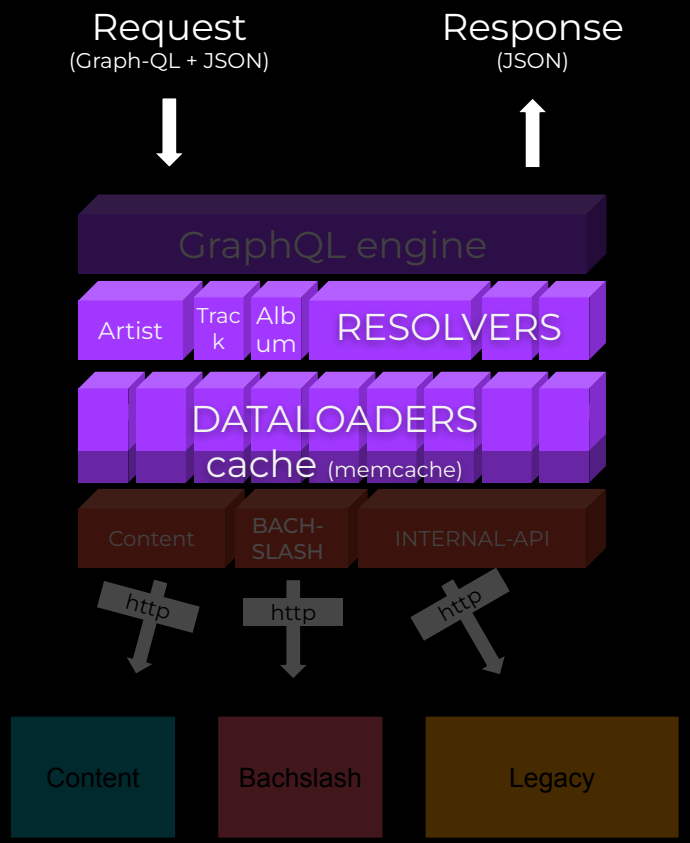
  More info here: [Deezer.io - GraphQL JIT, is it worth it?]



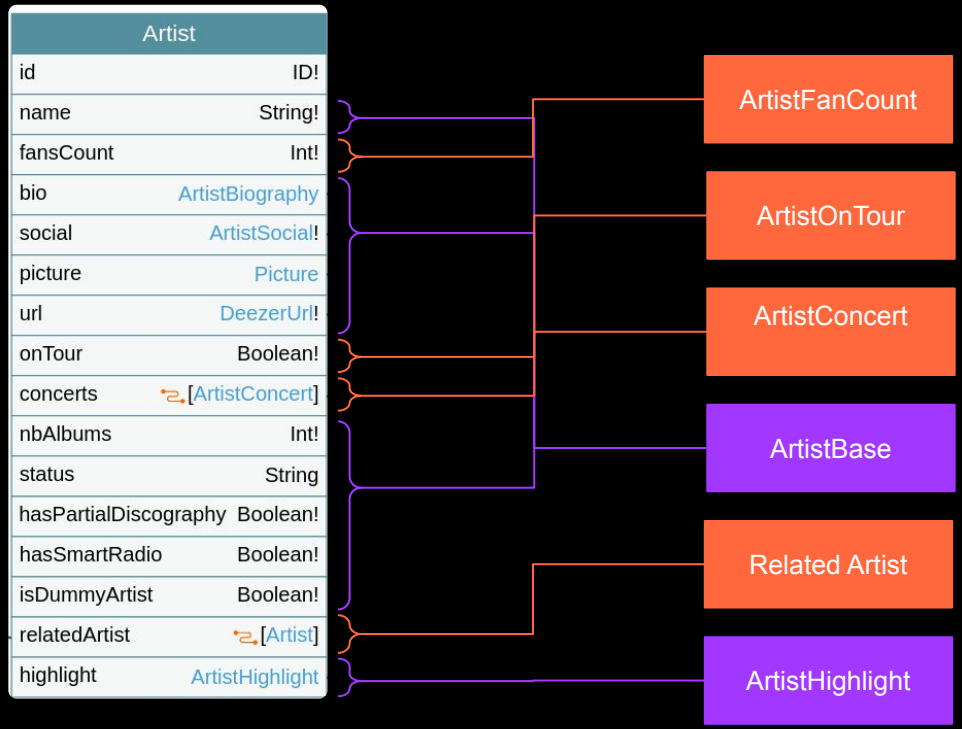
Where to optimize?

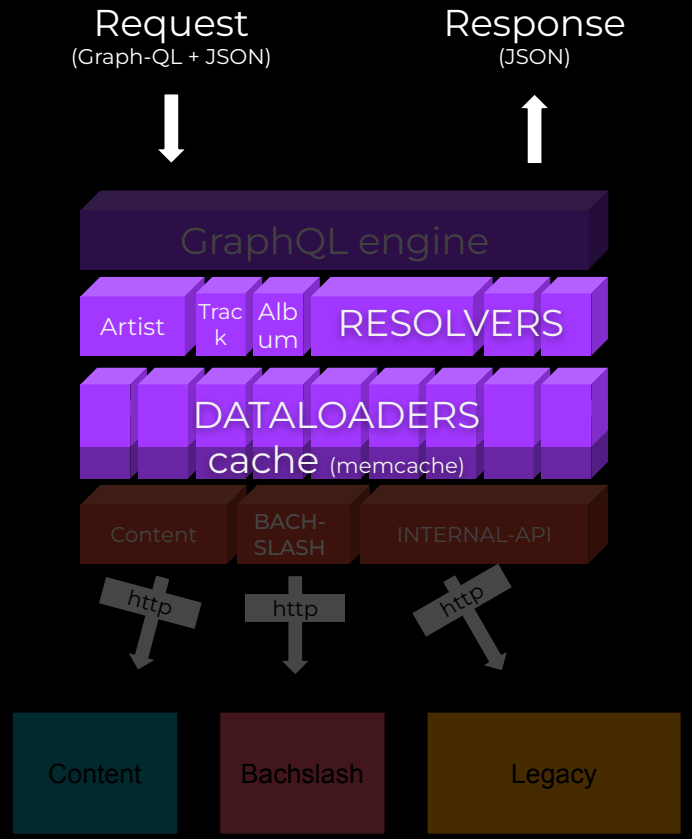
Mainly

- Resolvers
- Dataloaders



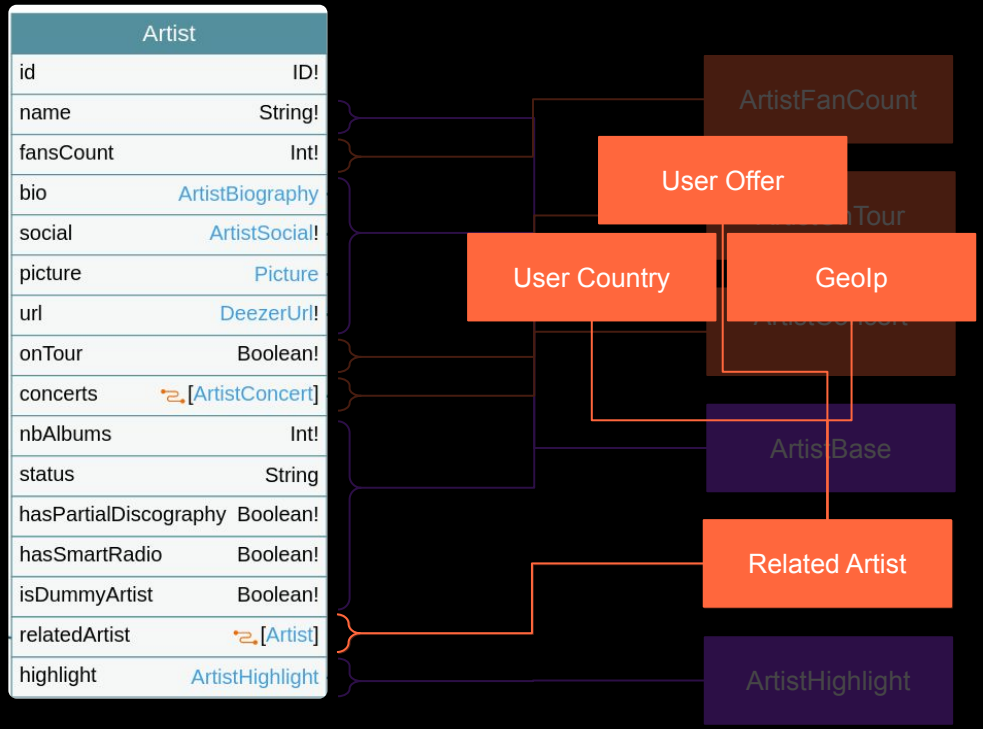
Where to optimize?

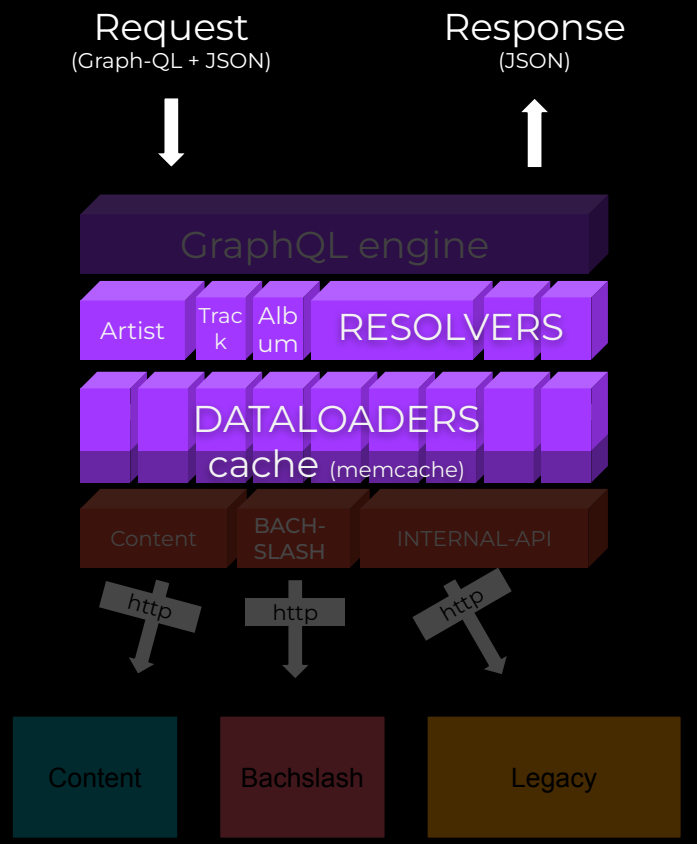




DEEZER

Where to optimize?



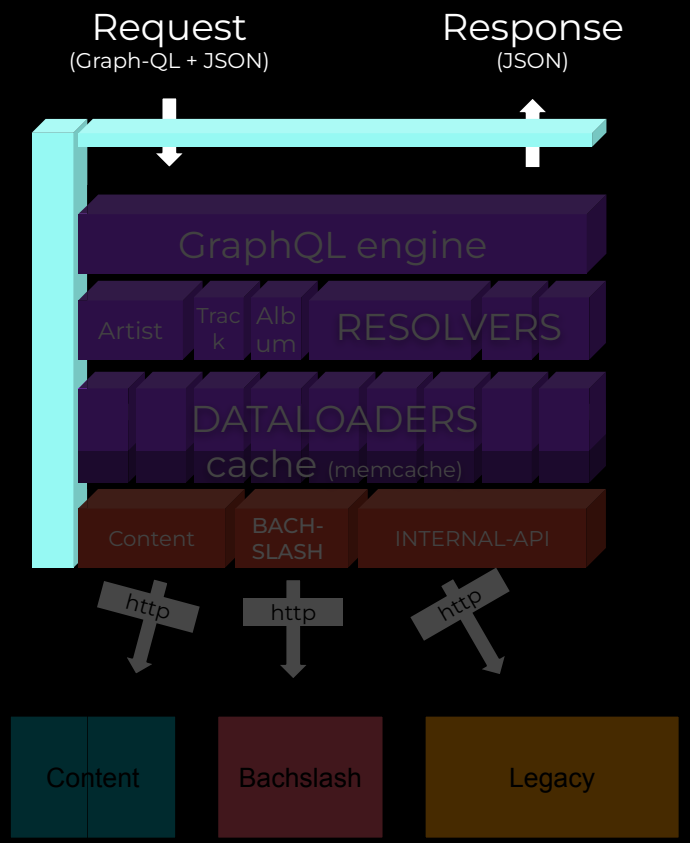


DEEZER

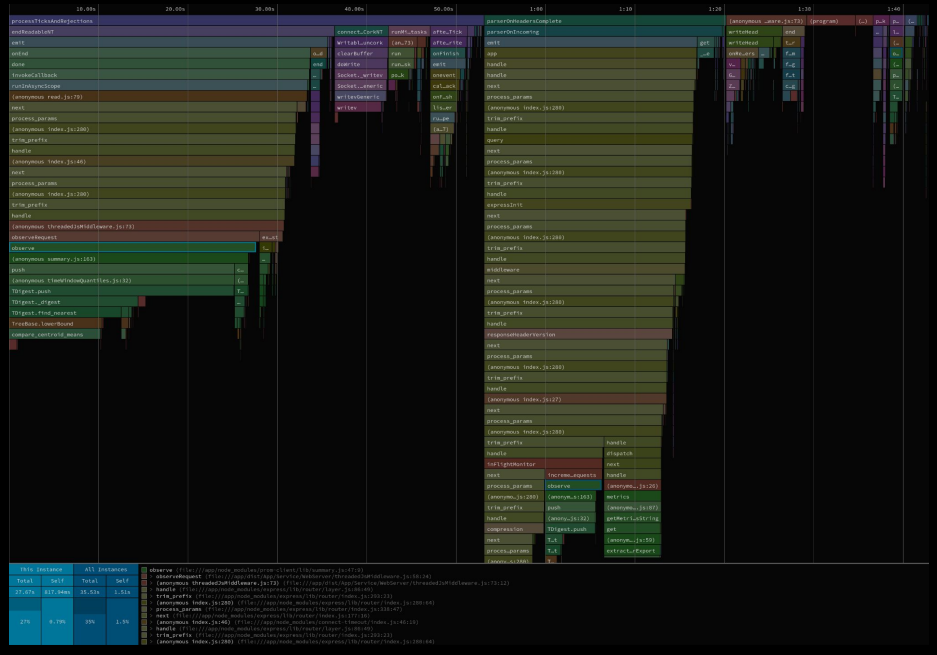
Where to optimize?

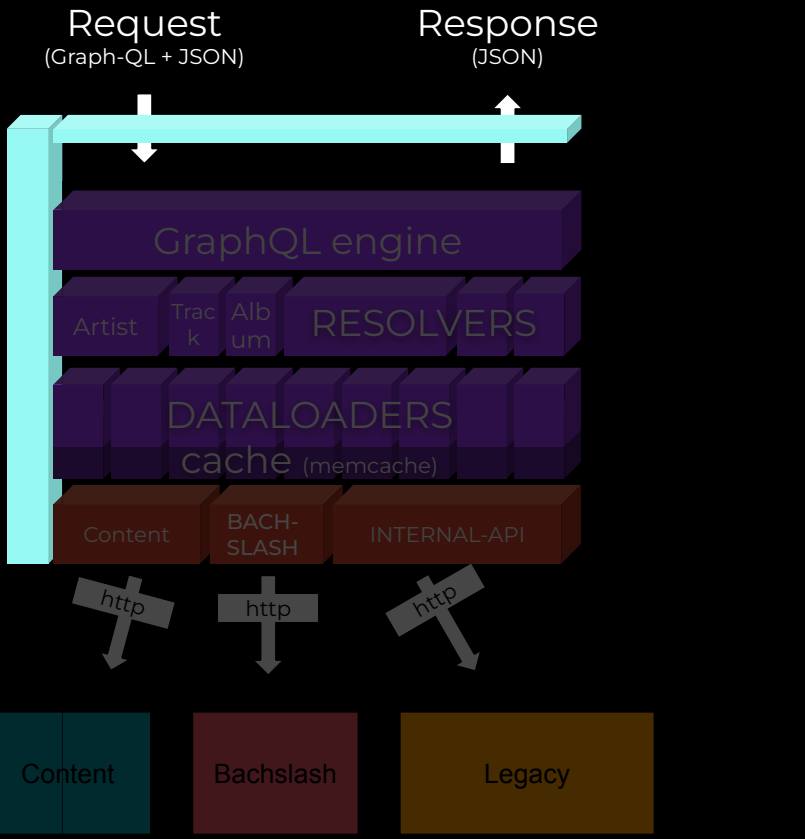
All the utility code

- Monitoring
- Express middleware
- GraphQL middleware
- GraphQL Shield



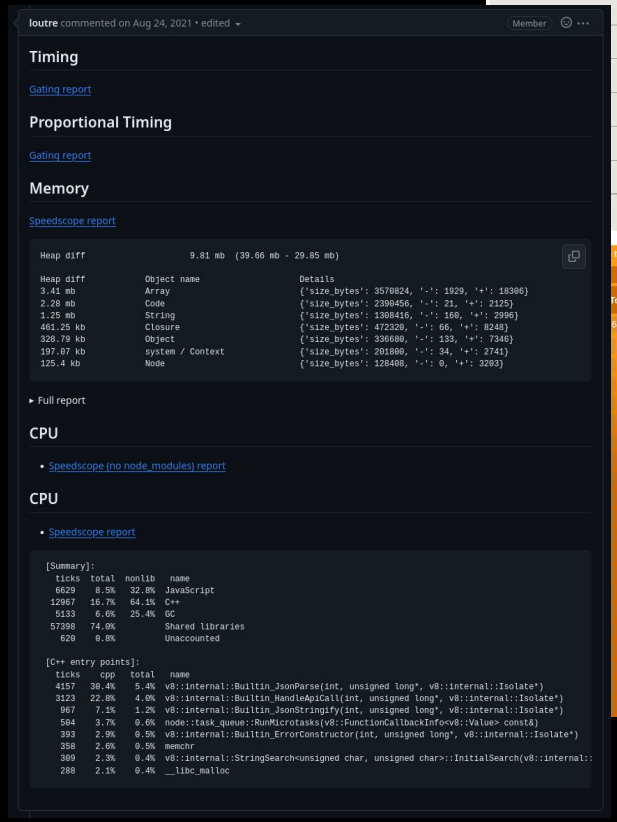
Where to optimize?





Where to optimize?

- Design realistic tests scenarios
- Automate them on each Pull Request
- Observe improvement PRs after PRs



loutre commented on Aug 24, 2021 • edited — Member

Timing

Gating report

Proportional Timing

Gating report

Memory

Speedscope report

```
Heap diff                  9.81 mb  (39.66 mb - 29.85 mb)

Heap diff       Object name                 Details
3.41 mb         Array                       {'size_bytes': 3570824, '-': 1929, '+': 18306}
2.28 mb         Code                        {'size_bytes': 2390456, '-': 21, '+': 2125}
1.25 mb         String                      {'size_bytes': 1308416, '-': 160, '+': 2996}
461.25 kb       Closure                     {'size_bytes': 472320, '-': 66, '+': 8248}
328.79 kb       Object                      {'size_bytes': 336680, '-': 133, '+': 7346}
197.07 kb       system / Context            {'size_bytes': 201800, '-': 34, '+': 2741}
125.4 kb        Node                        {'size_bytes': 128408, '-': 0, '+': 3203}
```

▸ Full report

CPU

- Speedscope (no node_modules) report

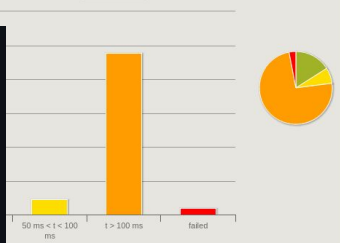
CPU

- Speedscope report

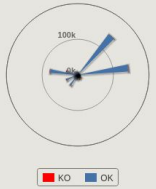
```
[Summary]:
  ticks  total  nonlib   name
   6629   8.5%   32.8%  JavaScript
  12967  16.7%   64.1%  C++
   5133   6.6%   25.4%  GC
  57398  74.0%          Shared libraries
    620   0.8%          Unaccounted

[C++ entry points]:
  ticks    cpp   total   name
   4157  30.4%   5.4%  v8::internal::Builtin_JsonParse(int, unsigned long*, v8::internal::Isolate*)
   3123  22.8%   4.0%  v8::internal::Builtin_HandleApiCall(int, unsigned long*, v8::internal::Isolate*)
    967   7.1%   1.2%  v8::internal::Builtin_JsonStringify(int, unsigned long*, v8::internal::Isolate*)
    504   3.7%   0.6%  node::task_queue::RunMicrotasks(v8::FunctionCallbackInfo<v8::Value> const&)
    393   2.9%   0.5%  v8::internal::Builtin_ErrorConstructor(int, unsigned long*, v8::internal::Isolate*)
    358   2.6%   0.5%  memchr
    309   2.3%   0.4%  v8::internal::StringSearch<unsigned char, unsigned char>::InitialSearch(v8::internal::
    288   2.1%   0.4%  __libc_malloc
```

> Global Information

Indicators



Number of requests



to show more) — Expand all groups | Collapse all groups

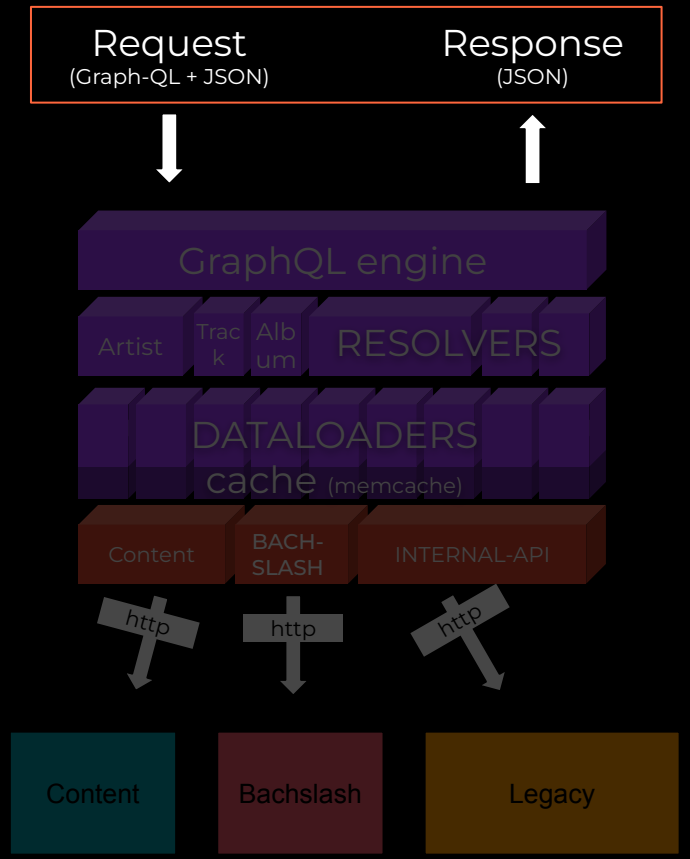
	Executions				Response Time (ms)							
Total	OK	KO	% KO	Cnt/s	Min	50th pct	75th pct	95th pct	99th pct	Max	Mean	Std Dev
640499	621453	19046	3%	1774.235	2	661	2508	7413	12334	36300	1851	2708
8630	8591	39	0%	23.906	115	135	152	312	634	1176	163	89
2201	2191	10	0%	6.097	9	74	505	3271	7737	18464	645	1534
900	898	2	0%	2.493	10	67	508	3290	6707	11339	599	1313
1479	1395	84	6%	4.097	10	139	864	4250	8347	17189	887	1779
143725	143245	480	0%	398.13	2	322	1198	4582	9086	23975	1043	1839
2197	2128	69	3%	6.086	19	353	2167	7686	13185	24060	1719	2857
900	849	51	6%	2.493	6	118	809	4382	9131	26304	914	2022
1478	1460	18	1%	4.094	17	262	1641	6266	11892	18439	1385	2394
142695	142149	546	0%	395.277	3	505	1784	5579	10295	28064	1371	2145
2190	2163	27	1%	6.066	14	304	1870	6643	11134	22054	1465	2449
898	251	647	72%	2.488	23	281	1876	7071	12764	24281	1553	2712
1476	1450	26	2%	4.089	17	277	2056	7040	11770	29339	1540	2610
2186	2139	47	2%	6.055	15	350	2008	7318	11766	23111	1614	2658
12840	8899	3941	31%	35.568	6	860	3181	7710	12767	25868	2113	2856
7155	7050	105	1%	19.82	7	879	3137	7829	13326	26323	2132	2879
2180	2158	22	1%	6.039	16	370	2235	7039	10577	18257	1637	2512
12641	12126	515	4%	35.017	8	1057	3916	8800	14118	27053	2493	3223
7050	6953	97	1%	19.529	3	814	3037	8028	12890	26556	2117	2928
2177	2161	16	1%	6.03	10	234	1304	5583	11031	24630	1194	2257
12404	12116	288	2%	34.36	18	1165	3905	8688	13719	33368	2560	3147

DEEZER

Where to optimize?

Batching

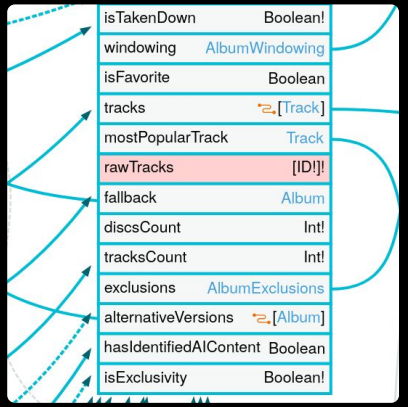
- Smaller sequential batches (500 → 100 or 50)
- Better caching on client side
- Single entity calls for big ones (Artist/Album)

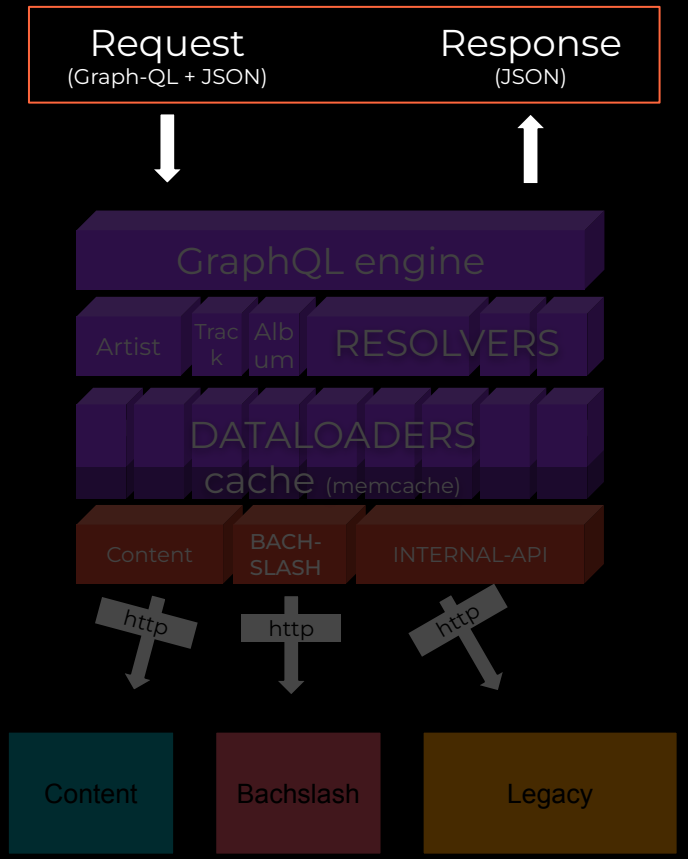


Where to optimize?

Pagination

- Raw Tracks → Fast but no filtering
- Parallel query of smaller batches (50)



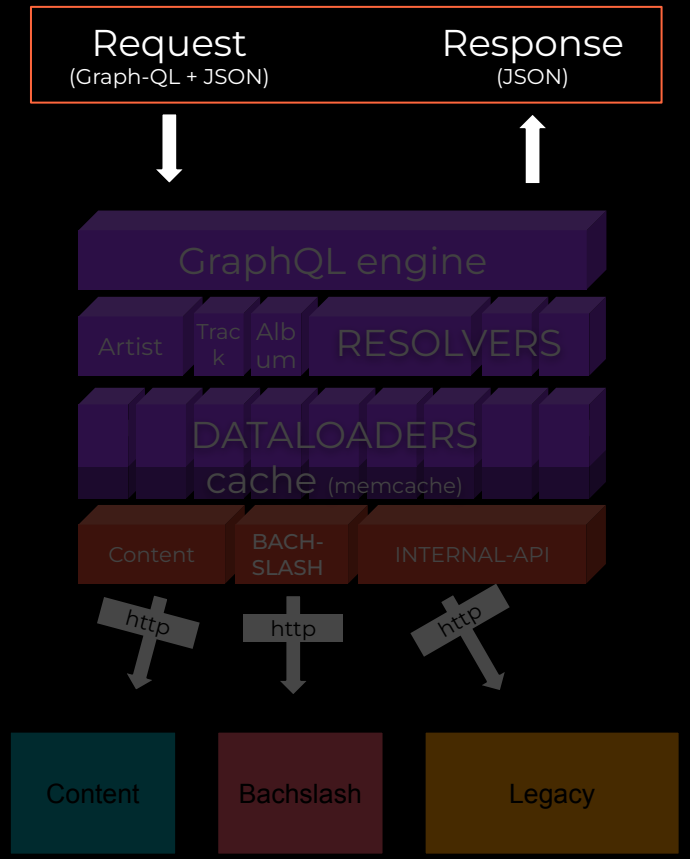


DEEZER

Where to optimize?

Smaller queries

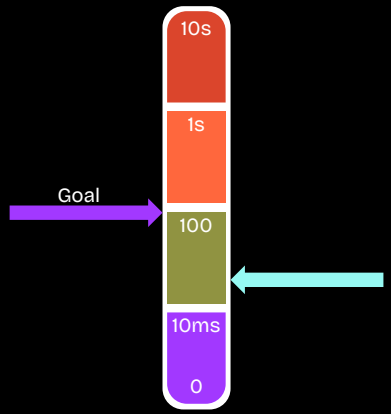
- Artist / Album / Tracks are fetch separately
- Client cache help reduce number of request
- More small requests spread load more easily



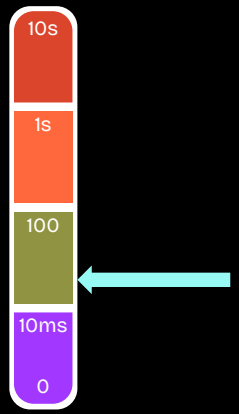
Results

How fast did it become?

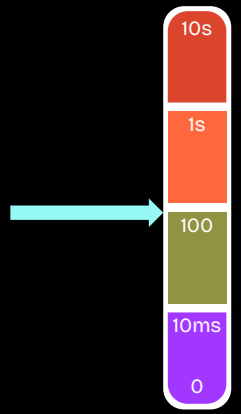
To load an artist



To load an album

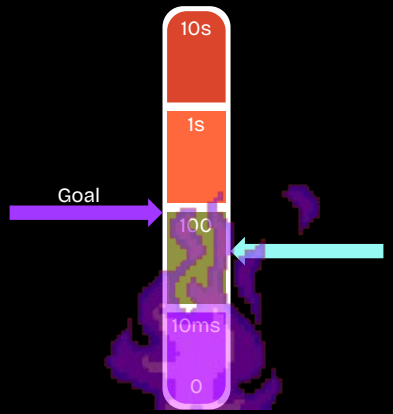


To load ~~50~~ 100 tracks

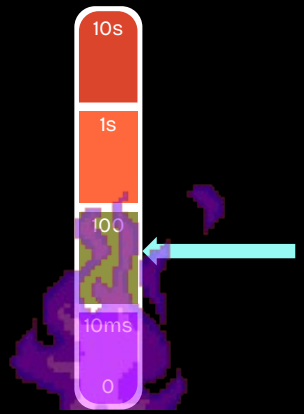


How fast did it become?

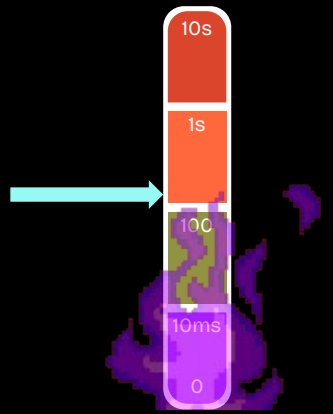
To load an artist



To load an album



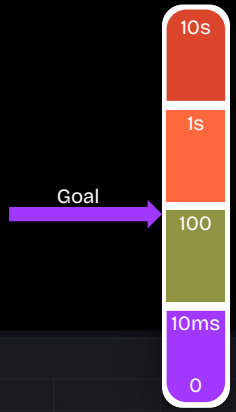
To load ~~50~~ 100 tracks



Under (real)**serious load**

DEEZER

Everything together

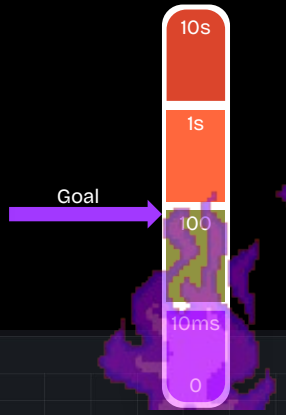


93% of
all requests
under 100ms

DEEZER

Everything together

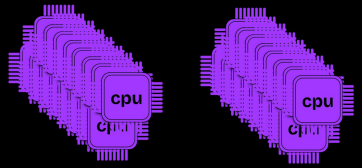
Under serious load



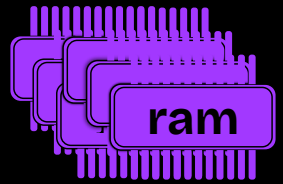
~70 to 80% of all requests under 100ms

DEEZER

Everything together



only ~15% of the estimated resources



Takeaways

Building a GraphQL API

Collaboration

All stacks and clients need to collaborate during all the designs and test phases in order to avoid pitfalls

How the API is used matters

Building a GraphQL API

Collaboration

All stacks and clients need to collaborate during all the designs and test phases in order to avoid pitfalls

How the API is used matters

Cache is paramount

You won't survive without Caching on both sides: Client & Backend

Building a GraphQL API

Collaboration

All stacks and clients need to collaborate during all the designs and test phases in order to avoid pitfalls

How the API is used matters

Cache is paramount

You won't survive without Caching on both sides: Client & Backend

NodeJS is fast but Graph can be big

The bigger the graph, the bigger the data in memory and JSON to parse and build.

Parsing and producing JSON will be the last unavoidable bottleneck

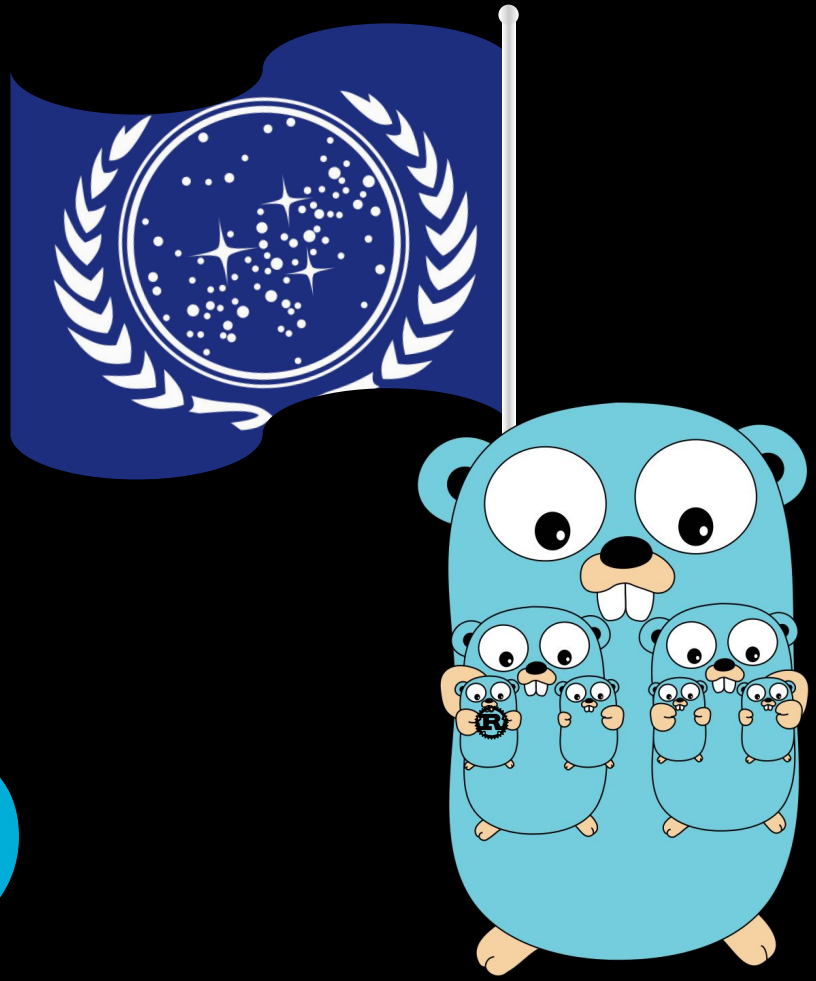
What's next?

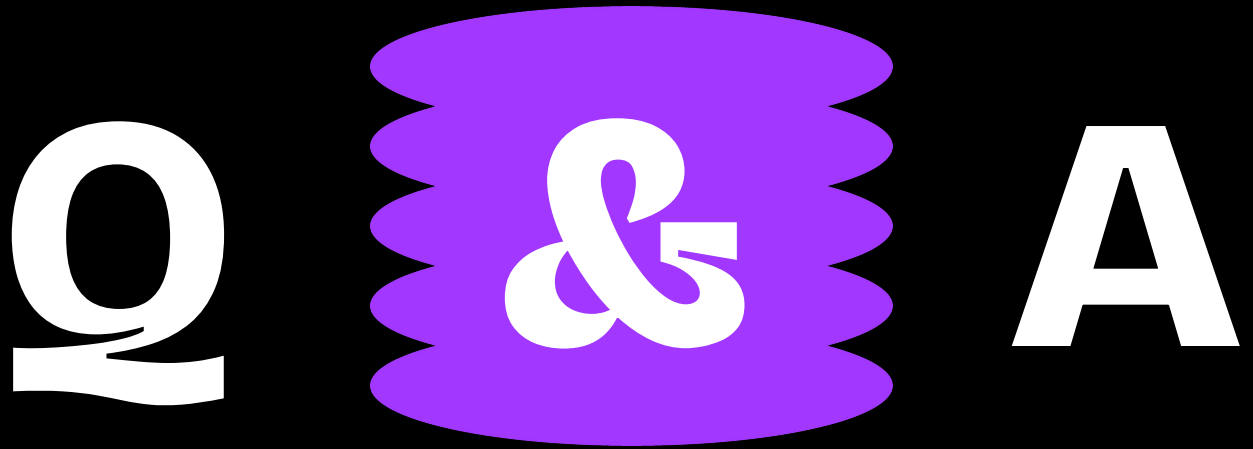
Future plans

- GraphQL Federation?

Future plans

- GraphQL Federation?
- Rewrite everything in GO? 😜




Q
&
A

Thank you

DEEZER