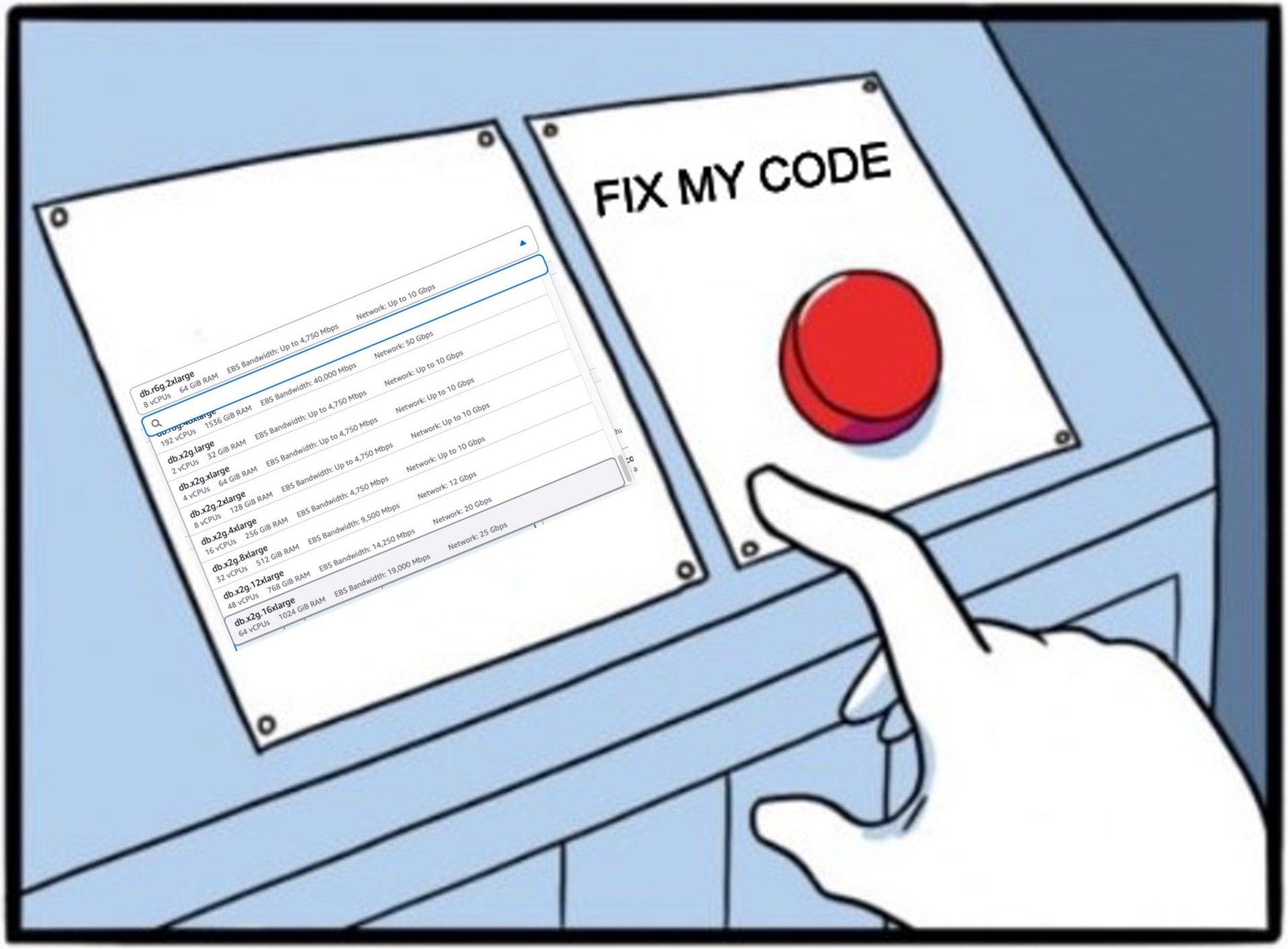
Stop scaling and start tuning

Matt Burke - https://mattburke.dev

“Customer states reports are too slow”

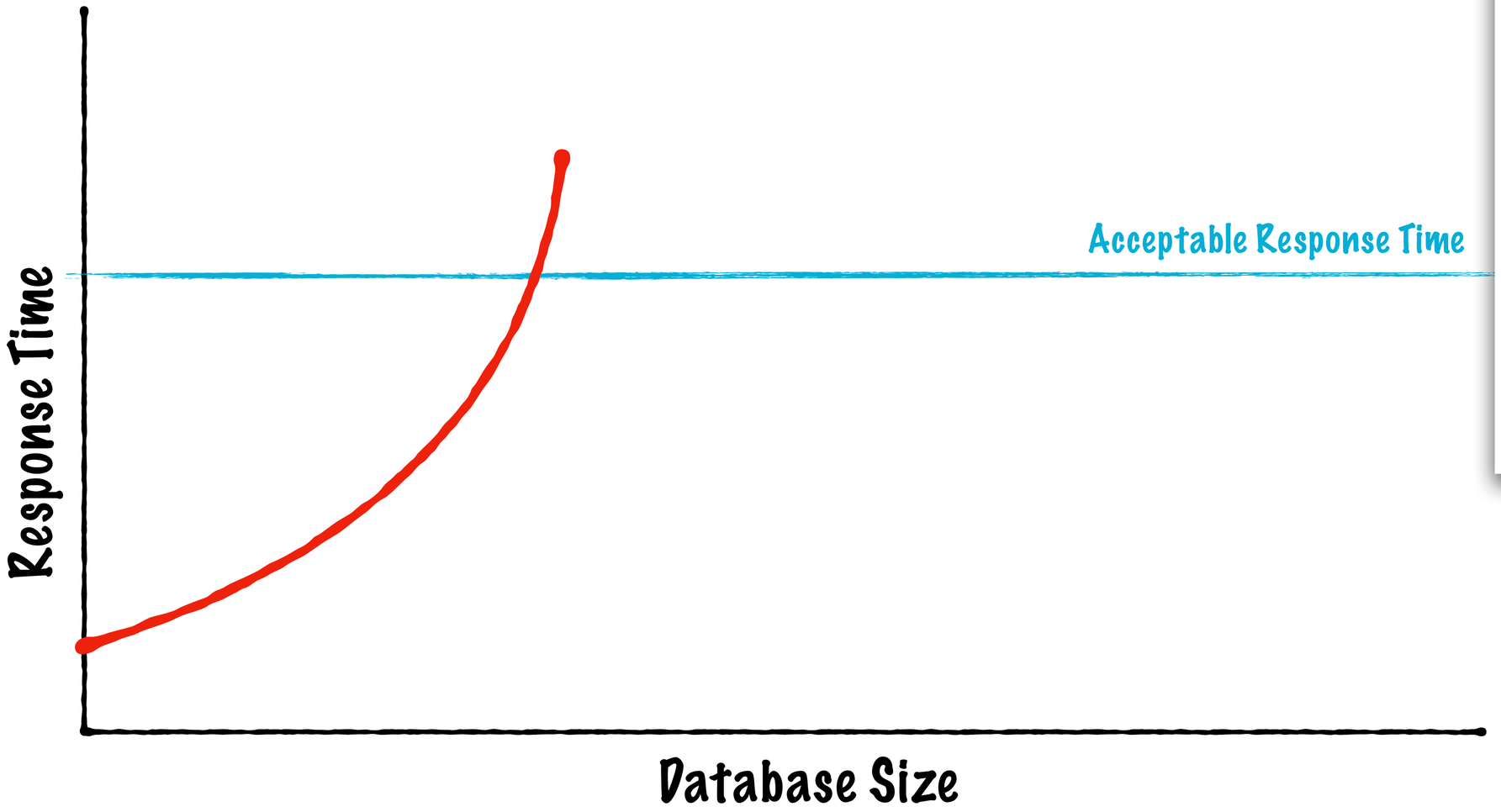
- JIRA API-52301


PROGRAMMERS


FIX MY CODE
db.r6g.2xlarge 8 vCPUs 64 GiB RAM EBS Bandwidth: Up to 4,750 Mbps Network: Up to 10 Gbps
192 vCPUs 1536 GiB RAM EBS Bandwidth: 40,000 Mbps Network: 50 Gbps
db.x2g.large 2 vCPUs 32 GiB RAM EBS Bandwidth: Up to 4,750 Mbps Network: Up to 10 Gbps
db.x2g.xlarge 4 vCPUs 64 GiB RAM EBS Bandwidth: Up to 4,750 Mbps Network: Up to 10 Gbps
db.x2g.2xlarge 8 vCPUs 128 GiB RAM EBS Bandwidth: Up to 4,750 Mbps Network: Up to 10 Gbps
db.x2g.4xlarge 16 vCPUs 256 GiB RAM EBS Bandwidth: 4,750 Mbps Network: Up to 10 Gbps
db.x2g.8xlarge 32 vCPUs 512 GiB RAM EBS Bandwidth: 9,500 Mbps Network: 12 Gbps
db.x2g.12xlarge 48 vCPUs 768 GiB RAM EBS Bandwidth: 14,250 Mbps Network: 20 Gbps
db.x2g.16xlarge 64 vCPUs 1024 GiB RAM EBS Bandwidth: 19,000 Mbps Network: 25 Gbps

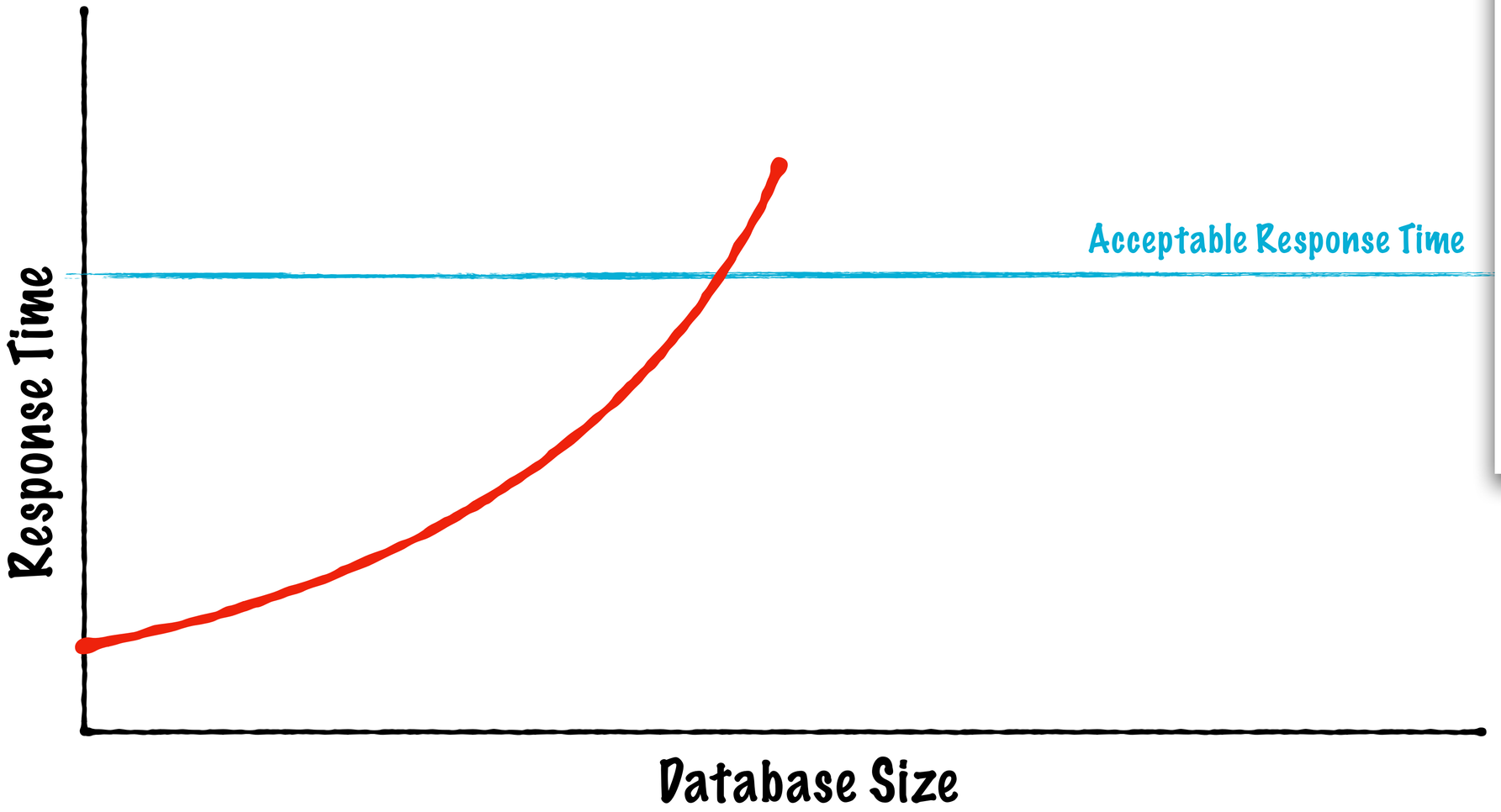

FIX MY CODE
db.r6g.2xlarge 8 vCPUs 64 GiB RAM EBS Bandwidth: Up to 4,750 Mbps Network: Up to 10 Gbps
192 vCPUs 1536 GiB RAM EBS Bandwidth: 40,000 Mbps Network: 50 Gbps
db.x2g.large 2 vCPUs 32 GiB RAM EBS Bandwidth: Up to 4,750 Mbps Network: Up to 10 Gbps
db.x2g.xlarge 4 vCPUs 64 GiB RAM EBS Bandwidth: Up to 4,750 Mbps Network: Up to 10 Gbps
db.x2g.2xlarge 8 vCPUs 128 GiB RAM EBS Bandwidth: Up to 4,750 Mbps Network: Up to 10 Gbps
db.x2g.4xlarge 16 vCPUs 256 GiB RAM EBS Bandwidth: 4,750 Mbps Network: Up to 10 Gbps
db.x2g.8xlarge 32 vCPUs 512 GiB RAM EBS Bandwidth: 9,500 Mbps Network: 12 Gbps
db.x2g.12xlarge 48 vCPUs 768 GiB RAM EBS Bandwidth: 14,250 Mbps Network: 20 Gbps
db.x2g.16xlarge 64 vCPUs 1024 GiB RAM EBS Bandwidth: 19,000 Mbps Network: 25 Gbps

Cloud is a great way to trade $ for performance



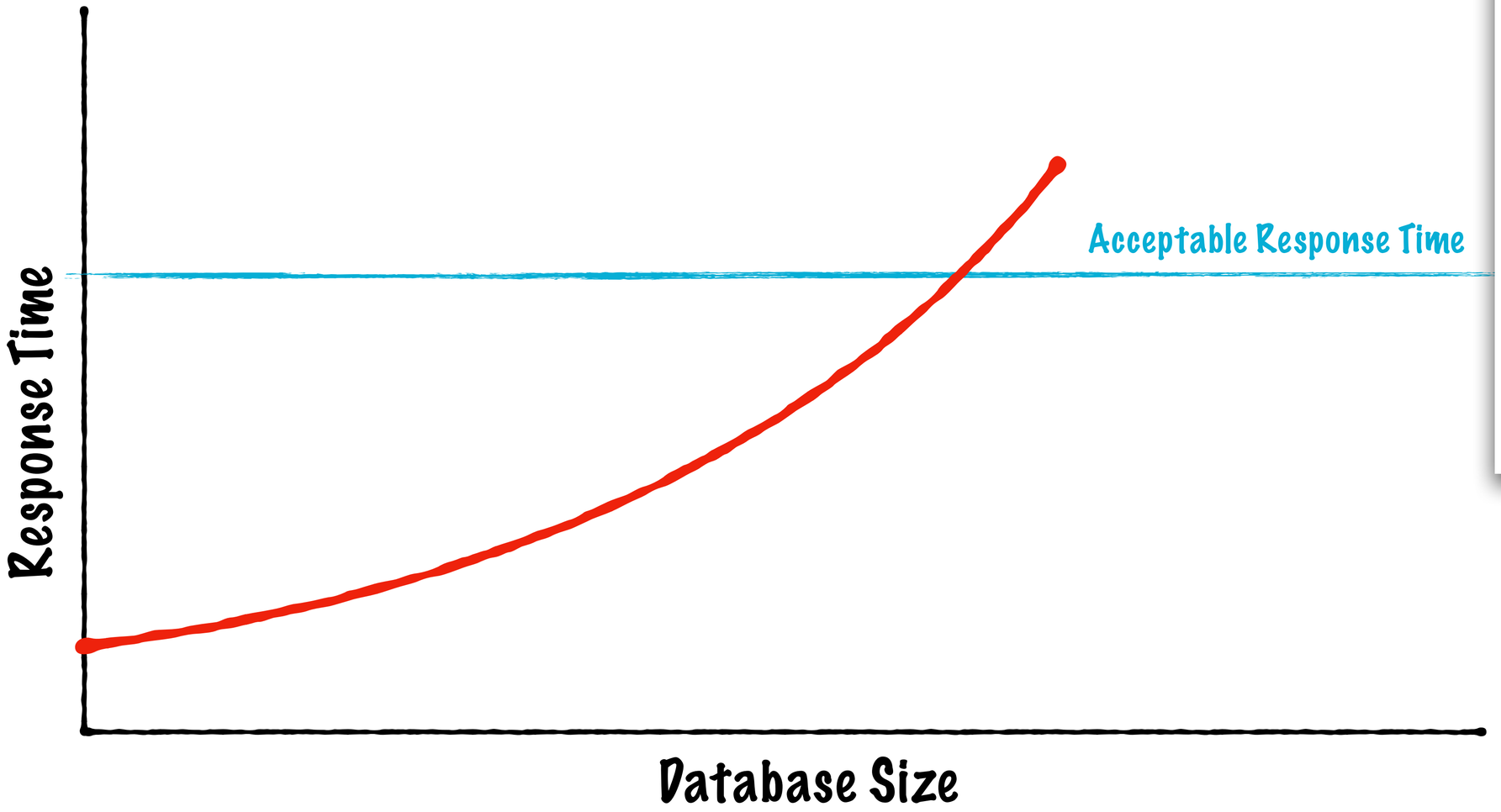
Estimated costs per month

Compute	$825.27
Cost per VCore (in USD) GeneralPurpose - Gen5	$206.32
VCores selected Provisioned	4
Storage	$35.33
Cost per GB (in USD)	$0.14
Max storage selected (in GB) Data max size + log storage	256
Discount	$4.42
32 GB storage included	$4.42
Azure Hybrid Benefit	$0.00
Estimated total	$856.18



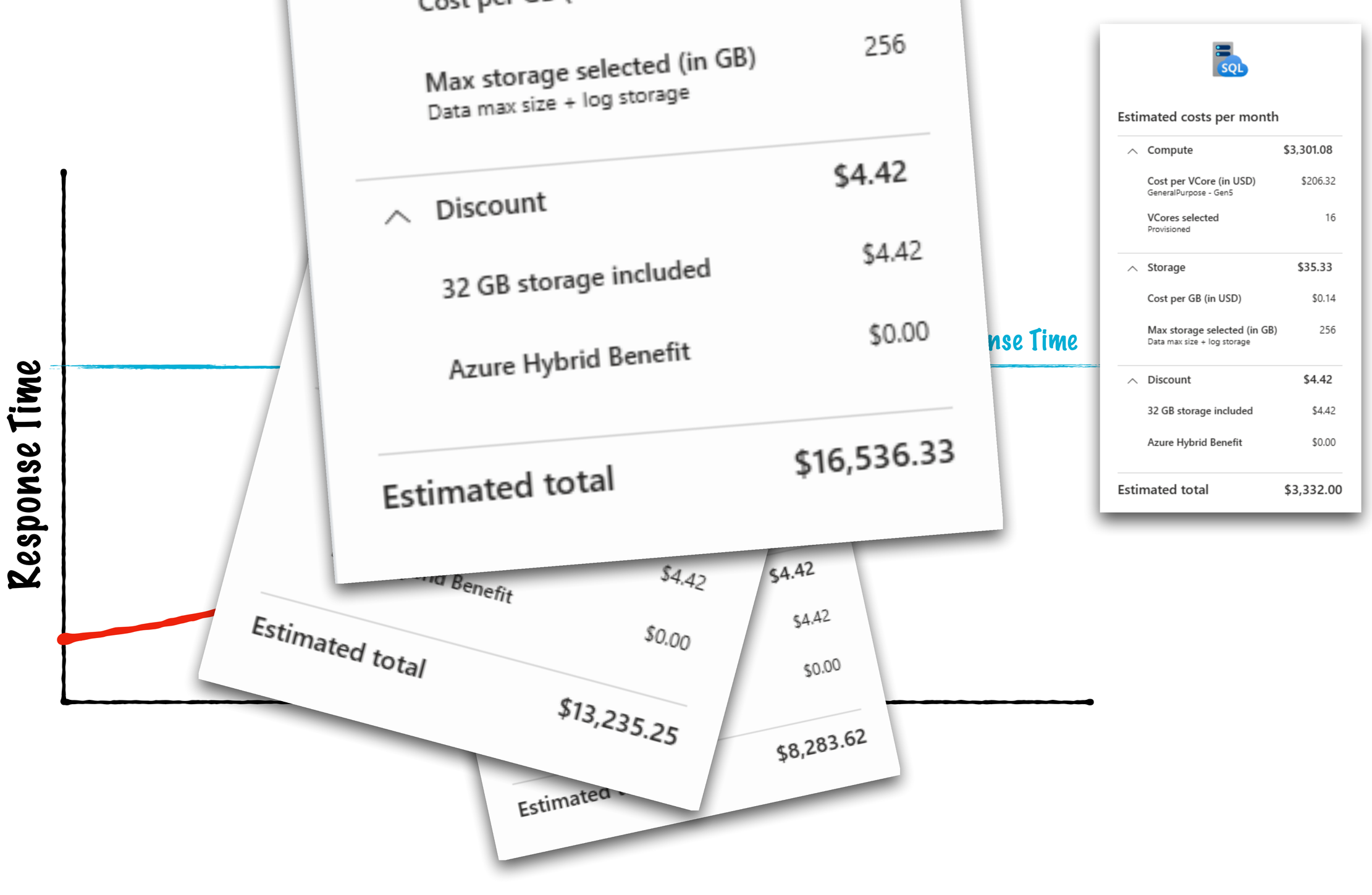
Estimated costs per month

Compute	$1,650.54
Cost per VCore (in USD) GeneralPurpose - Gen5	$206.32
VCores selected Provisioned	8
Storage	$35.33
Cost per GB (in USD)	$0.14
Max storage selected (in GB) Data max size + log storage	256
Discount	$4.42
32 GB storage included	$4.42
Azure Hybrid Benefit	$0.00
Estimated total	$1,681.45



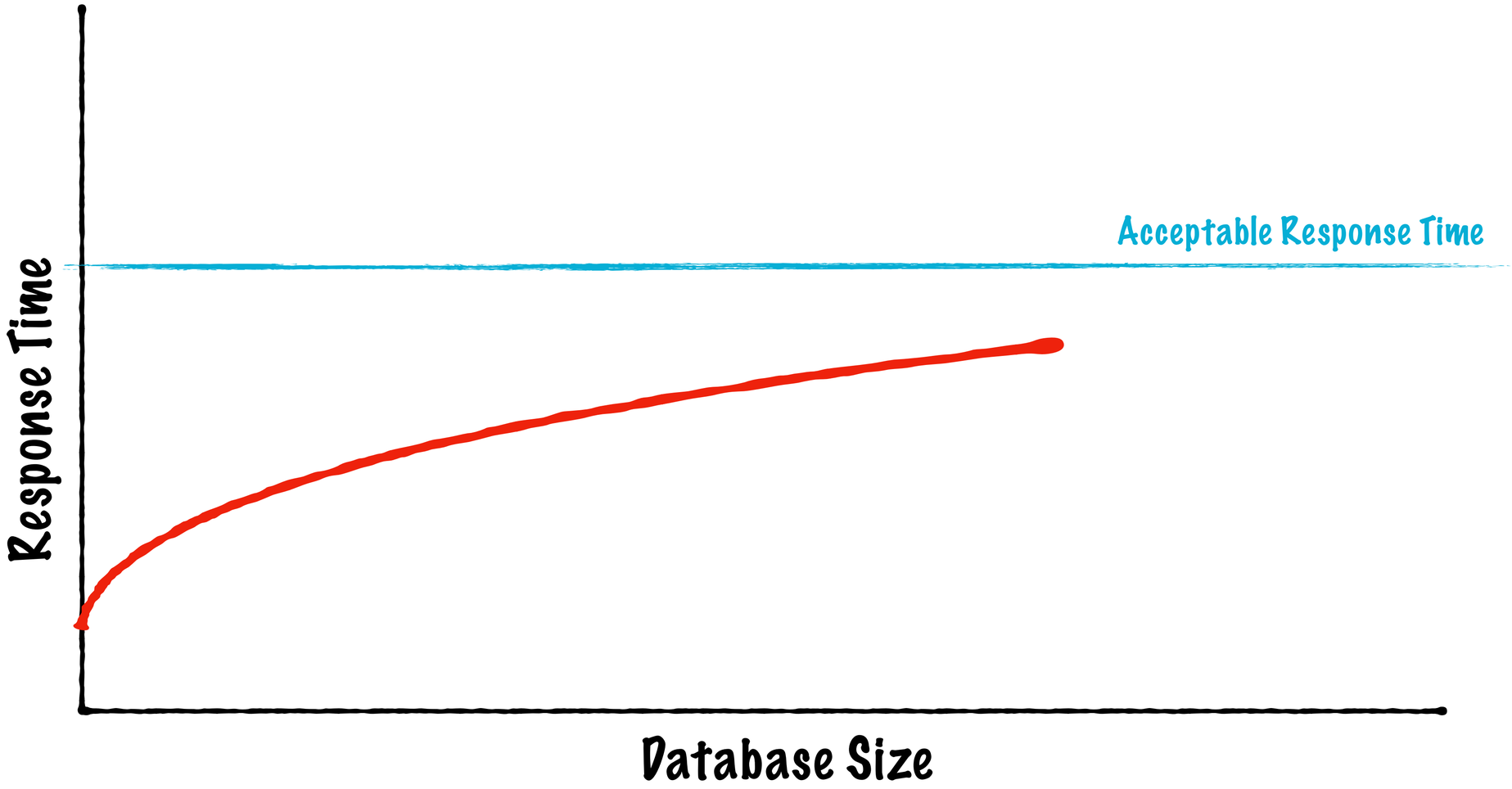
Estimated costs per month

Compute	$3,301.08
Cost per VCore (in USD) GeneralPurpose - Gen5	$206.32
VCores selected Provisioned	16
Storage	$35.33
Cost per GB (in USD)	$0.14
Max storage selected (in GB) Data max size + log storage	256
Discount	$4.42
32 GB storage included	$4.42
Azure Hybrid Benefit	$0.00
Estimated total	$3,332.00

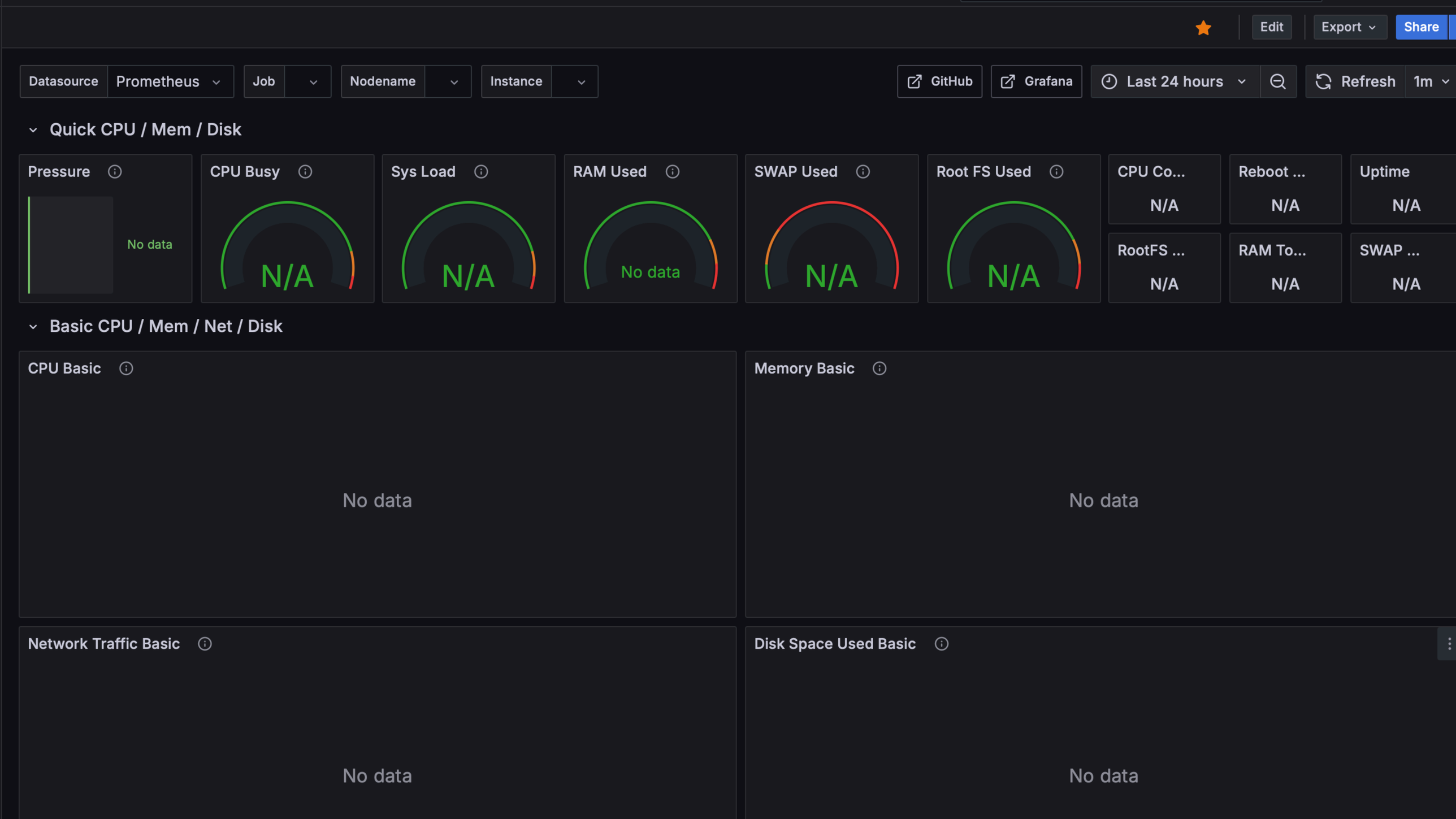

Response Time
nse Time
Max storage selected (in GB)
Data max size + log storage
256
Discount
$4.42
32 GB storage included
$4.42
Azure Hybrid Benefit
$0.00
Estimated total
$16,536.33
Estimated costs per month
Compute
$3,301.08
Cost per VCore (in USD)
GeneralPurpose - Gen5
$206.32
VCores selected
Provisioned
16
Storage
$35.33
Cost per GB (in USD)
$0.14
Max storage selected (in GB)
Data max size + log storage
256
Discount
$4.42
32 GB storage included
$4.42
Azure Hybrid Benefit
$0.00
Estimated total
$3,332.00
$4.42
$0.00
Estimated total
$13,235.25
$4.42
$4.42
$0.00
$8,283.62

Paying more is not necessarily wrong

You can’t outspend $O(N^2)$


Acceptable Response Time
Response Time
Database Size

```
git checkout -b fix/performance
```


Edit
Export
Share
Datasource
Prometheus
Job
Nodename
Instance
GitHub
Grafana
Last 24 hours
Refresh
1m
Quick CPU / Mem / Disk
Pressure
No data
CPU Busy
N/A
Sys Load
N/A
RAM Used
No data
SWAP Used
N/A
Root FS Used
N/A
CPU Co...
N/A
Reboot ...
N/A
Uptime
N/A
RootFS ...
N/A
RAM To...
N/A
SWAP ...
N/A
Basic CPU / Mem / Net / Disk
CPU Basic
No data
Memory Basic
No data
Network Traffic Basic
No data
Disk Space Used Basic
No data

First things first: Monitoring

Diagnostics

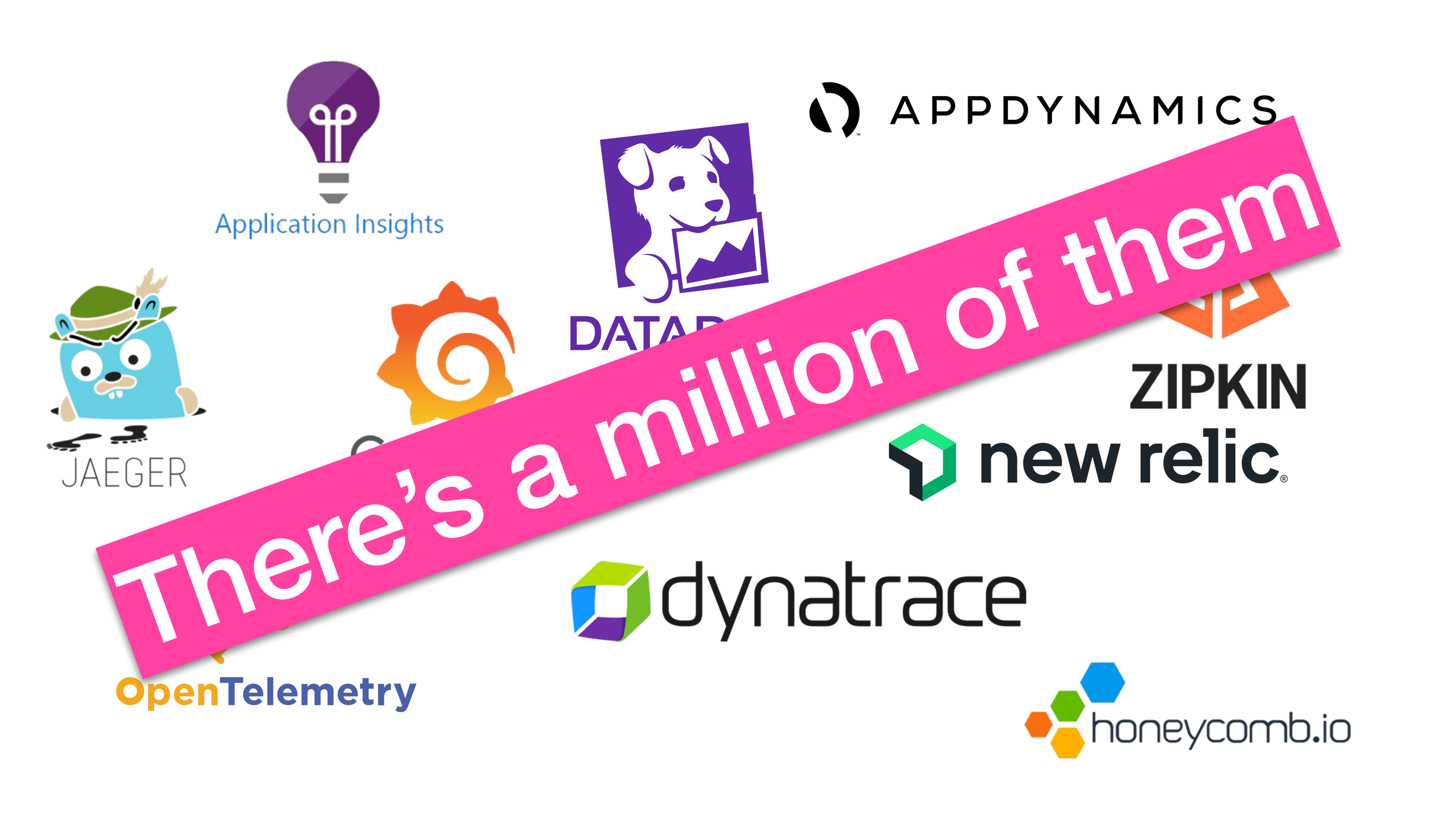
Find your trouble-makers

Establish Baselines

Know you’ve fixed it

Justify the cost

Executives love pretty graphs


Application Insights
APPDYNAMICS
JAEGER
ZIPKIN
new relic
There's a million of them
dynatrace
OpenTelemetry
honeycomb.io

Pick one and learn it

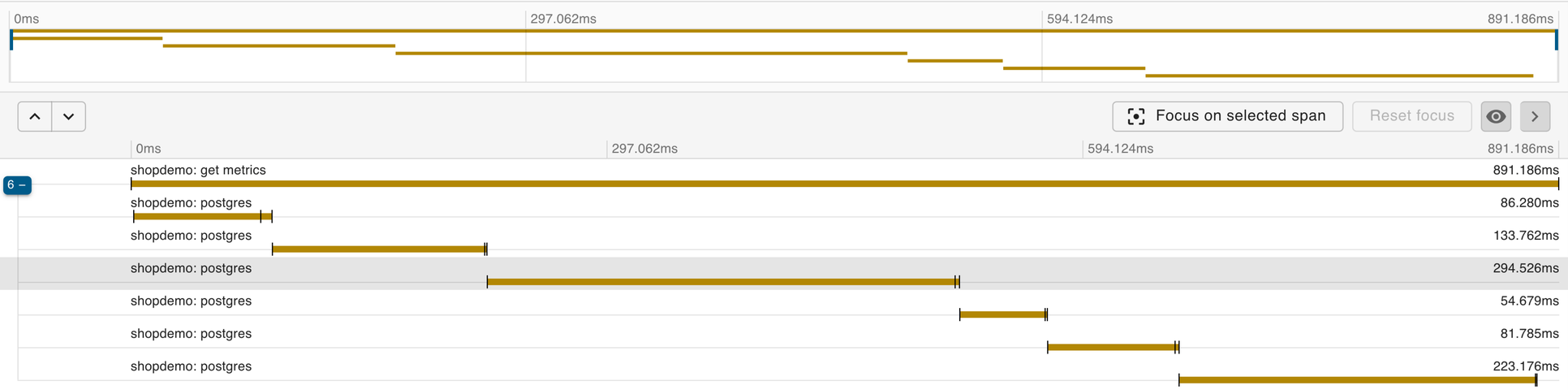
Spans

Zipkin | Find a trace | Dependencies | Search by trace ID | EN

shopdemo: get metrics

SPAN TABLE

Duration 891.186ms | Services 1 | Total Spans 7 | Trace ID 8b5bc0d5350f53bbc91005fa64f7c7b7



Service name
shopdemo

Span name
postgres

Span ID
79bfd3b1ef79a836

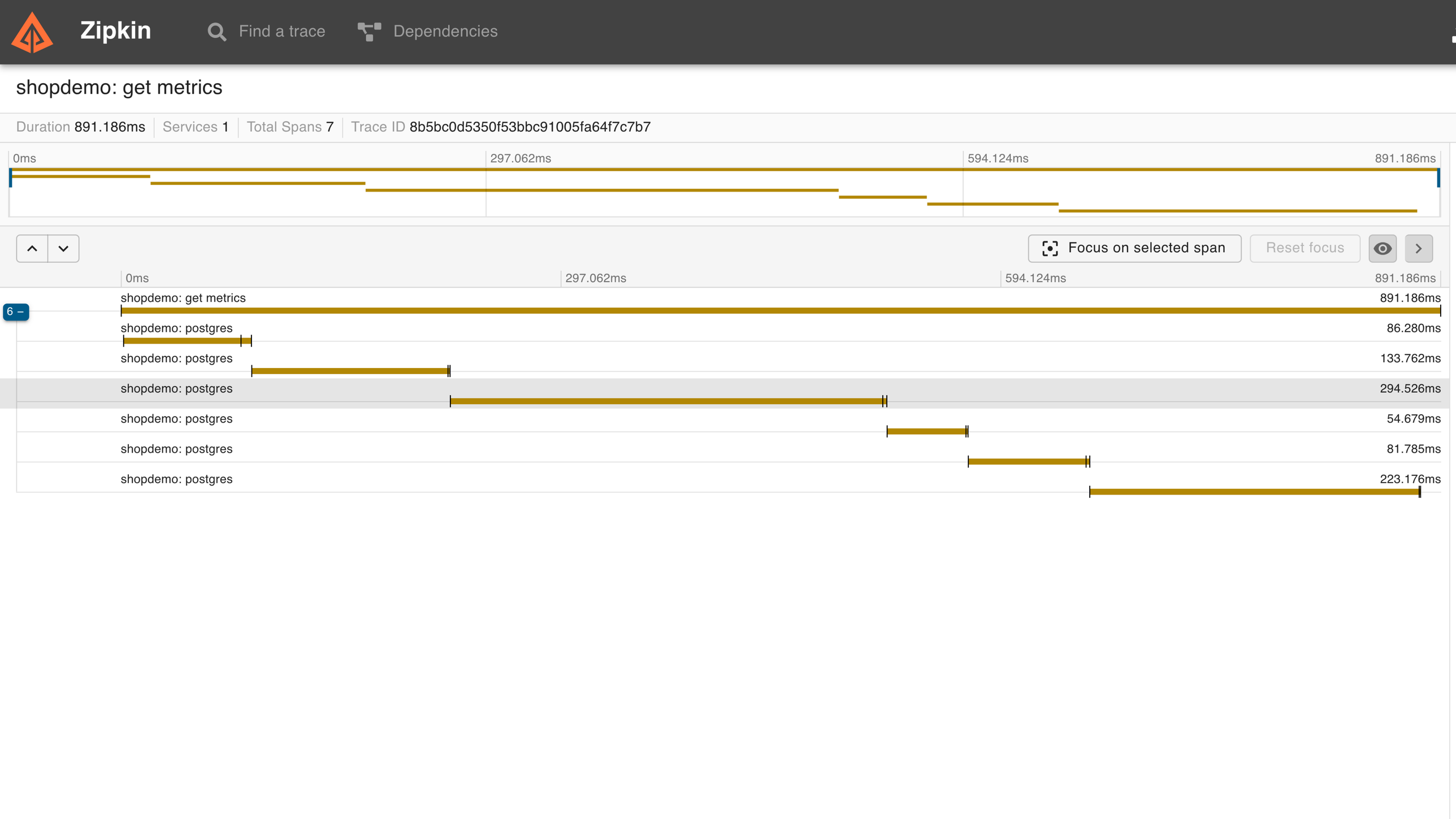
Parent ID
ea92c0a55297513a

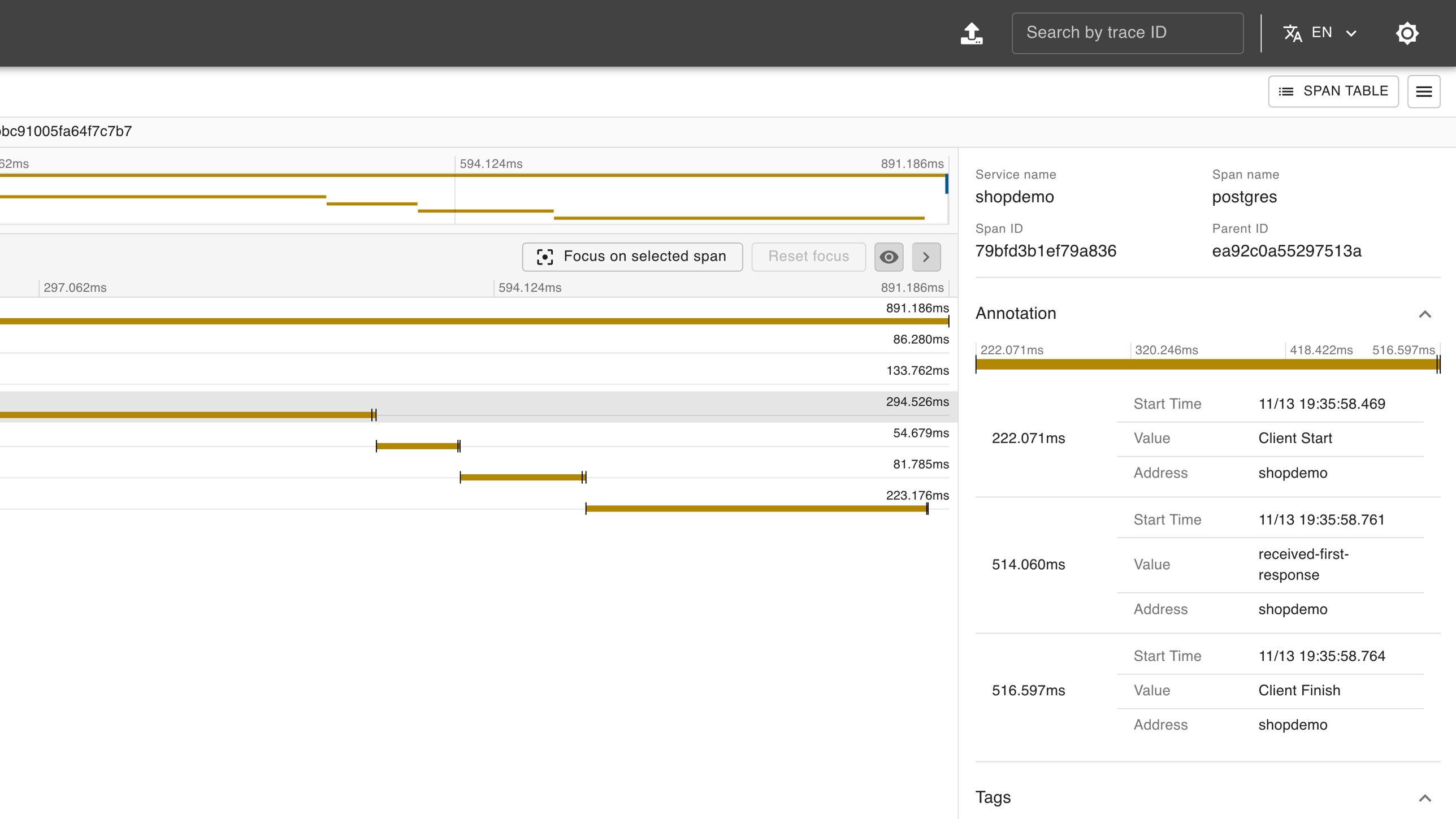
Annotation

222.071ms	Start Time	11/13 19:35:58.469
	Value	Client Start
	Address	shopdemo
514.060ms	Start Time	11/13 19:35:58.761
	Value	received-first-response
	Address	shopdemo
516.597ms	Start Time	11/13 19:35:58.764
	Value	Client Finish
	Address	shopdemo

Tags

db.connection_id	86
db.connection_string	Host=localhost;Port=5432;Database=postgres;Username=postgres;Maximum Pool Size=10
db.name	postgres
	SELECT p.id as product_id, p.name as product_name, p.sku, SUM(ol.quantity) as total_quantity, SUM(ol.quantity * ol.unit_price) as


Zipkin
Find a trace
Dependencies
shopdemo: get metrics
Duration 891.186ms
Services 1
Total Spans 7
Trace ID 8b5bc0d5350f53bbc91005fa64f7c7b7
0ms
297.062ms
594.124ms
891.186ms
Focus on selected span
Reset focus
0ms
297.062ms
594.124ms
891.186ms
6 –
shopdemo: get metrics
891.186ms
shopdemo: postgres
86.280ms
shopdemo: postgres
133.762ms
shopdemo: postgres
294.526ms
shopdemo: postgres
54.679ms
shopdemo: postgres
81.785ms
shopdemo: postgres
223.176ms

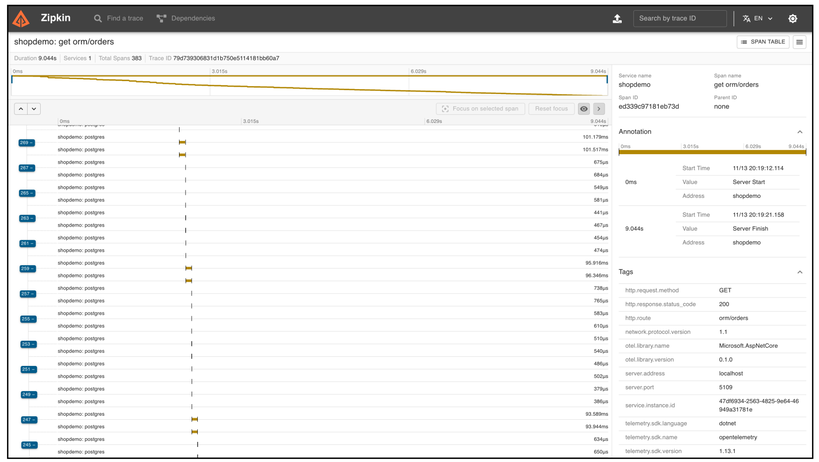

Search by trace ID
EN
SPAN TABLE
bc91005fa64f7c7b7
62ms
594.124ms
891.186ms
Focus on selected span
Reset focus
297.062ms
594.124ms
891.186ms
891.186ms
86.280ms
133.762ms
294.526ms
54.679ms
81.785ms
223.176ms
Service name
shopdemo
Span name
postgres
Span ID
79bfd3b1ef79a836
Parent ID
ea92c0a55297513a
Annotation
222.071ms
320.246ms
418.422ms
516.597ms
222.071ms
Start Time 11/13 19:35:58.469
Value Client Start
Address shopdemo
514.060ms
Start Time 11/13 19:35:58.761
Value received-first-response
Address shopdemo
516.597ms
Start Time 11/13 19:35:58.764
Value Client Finish
Address shopdemo
Tags

	Start Time	11/13 19:35:58.764
516.597ms	Value	Client Finish
	Address	shopdemo

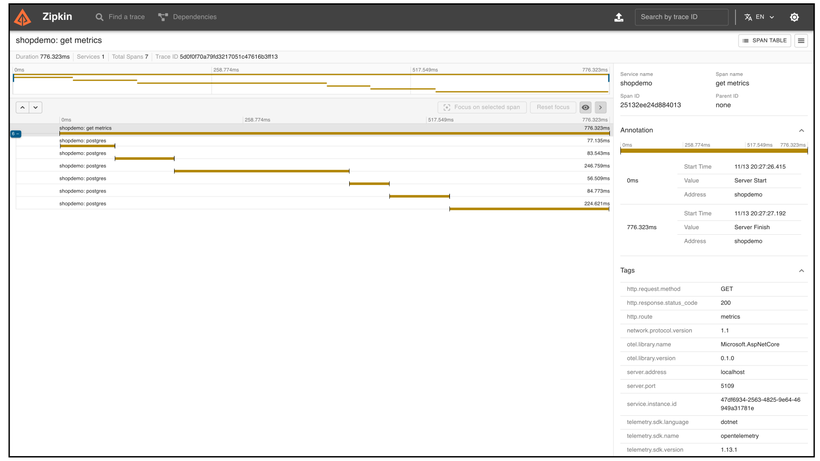
Tags

db.connection_id	86
db.connection_string	Host=localhost;Port=5432;Database=postgres;Username=postgres;Maximum Pool Size=10
db.name	postgres
db.statement	SELECT p.id as product_id, p.name as product_name, p.sku, SUM(ol.quantity) as total_quantity, SUM(ol.quantity * ol.unit_price) as total_revenue, COUNT(DISTINCT ol.order_id) as order_count, AVG(ol.unit_price) as avg_unit_price FROM products p INNER JOIN order_lines ol ON p.id = ol.product_id INNER JOIN orders o ON ol.order_id = o.id WHERE o.created_at >= @StartDate AND o.created_at < @EndDate GROUP BY p.id, p.name, p.sku ORDER BY total_revenue DESC LIMIT 15
db.system	postgresql

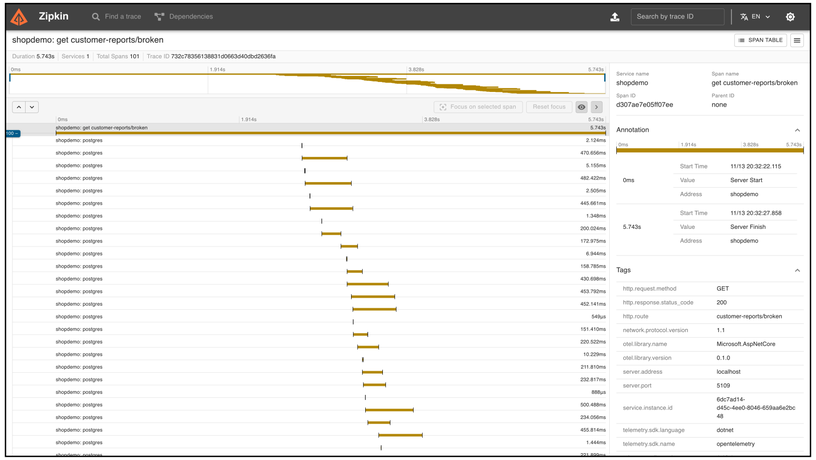
Let it bake



Many Small Spans



Large Spans

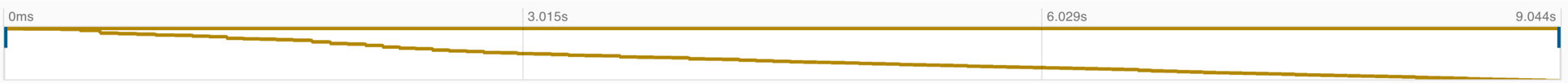


Span Gaps

shopdemo: get orm/orders

SPAN TABLE

Duration 9.044s | Services 1 | Total Spans 383 | Trace ID 79d739306831d1b750e5114181bb60a7



Focus on selected span | Reset focus

	0ms	3.015s	6.029s	9.044s
269 –	shopdemo: postgres			101.179ms
	shopdemo: postgres			101.517ms
267 –	shopdemo: postgres			675µs
	shopdemo: postgres			684µs
265 –	shopdemo: postgres			549µs
	shopdemo: postgres			581µs
263 –	shopdemo: postgres			441µs
	shopdemo: postgres			467µs
261 –	shopdemo: postgres			454µs
	shopdemo: postgres			474µs
259 –	shopdemo: postgres			95.916ms
	shopdemo: postgres			96.346ms
257 –	shopdemo: postgres			738µs
	shopdemo: postgres			765µs
255 –	shopdemo: postgres			583µs
	shopdemo: postgres			610µs
253 –	shopdemo: postgres			510µs
	shopdemo: postgres			540µs
251 –	shopdemo: postgres			486µs
	shopdemo: postgres			502µs
249 –	shopdemo: postgres			379µs
	shopdemo: postgres			386µs
247 –	shopdemo: postgres			93.589ms
	shopdemo: postgres			93.944ms
245 –	shopdemo: postgres			634µs
	shopdemo: postgres			650µs

Service name: shopdemo
Span name: get orm/orders
Span ID: ed339c97181eb73d
Parent ID: none

Annotation

0ms	Start Time	11/13 20:19:12.114
	Value	Server Start
	Address	shopdemo
9.044s	Start Time	11/13 20:19:21.158
	Value	Server Finish
	Address	shopdemo

Tags

http.request.method	GET
http.response.status_code	200
http.route	orm/orders
network.protocol.version	1.1
otel.library.name	Microsoft.AspNetCore
otel.library.version	0.1.0
server.address	localhost
server.port	5109
service.instance.id	47df6934-2563-4825-9e64-46949a31781e
telemetry.sdk.language	dotnet
telemetry.sdk.name	opentelemetry
telemetry.sdk.version	1.13.1

http://localhost:5109/sql/orders

```
SELECT * FROM orders;

SELECT * FROM order_lines WHERE order_id = 1;
SELECT * FROM order_lines WHERE order_id = 2;
SELECT * FROM order_lines WHERE order_id = 3;
SELECT * FROM order_lines WHERE order_id = 4;
SELECT * FROM order_lines WHERE order_id = 5;
SELECT * FROM order_lines WHERE order_id = 6;
SELECT * FROM order_lines WHERE order_id = 7;
SELECT * FROM order_lines WHERE order_id = 8;
SELECT * FROM order_lines WHERE order_id = 9;
…
```

N+1

```
SELECT * FROM orders;

SELECT * FROM order_lines WHERE order_id = 1;
SELECT * FROM order_lines WHERE order_id = 2;
SELECT * FROM order_lines WHERE order_id = 3;
SELECT * FROM order_lines WHERE order_id = 4;
SELECT * FROM order_lines WHERE order_id = 5;
SELECT * FROM order_lines WHERE order_id = 6;
SELECT * FROM order_lines WHERE order_id = 7;
SELECT * FROM order_lines WHERE order_id = 8;
SELECT * FROM order_lines WHERE order_id = 9;
…
```

```
SELECT * FROM orders;

SELECT * FROM order_lines WHERE order_id = 1;
SELECT * FROM order_lines WHERE order_id = 2;
SELECT * FROM order_lines WHERE order_id = 3;
SELECT * FROM order_lines WHERE order_id = 4;
SELECT * FROM order_lines WHERE order_id = 5;
SELECT * FROM order_lines WHERE order_id = 6;
SELECT * FROM order_lines WHERE order_id = 7;
SELECT * FROM order_lines WHERE order_id = 8;
SELECT * FROM order_lines WHERE order_id = 9;
…
```

Cause 1: Architecture

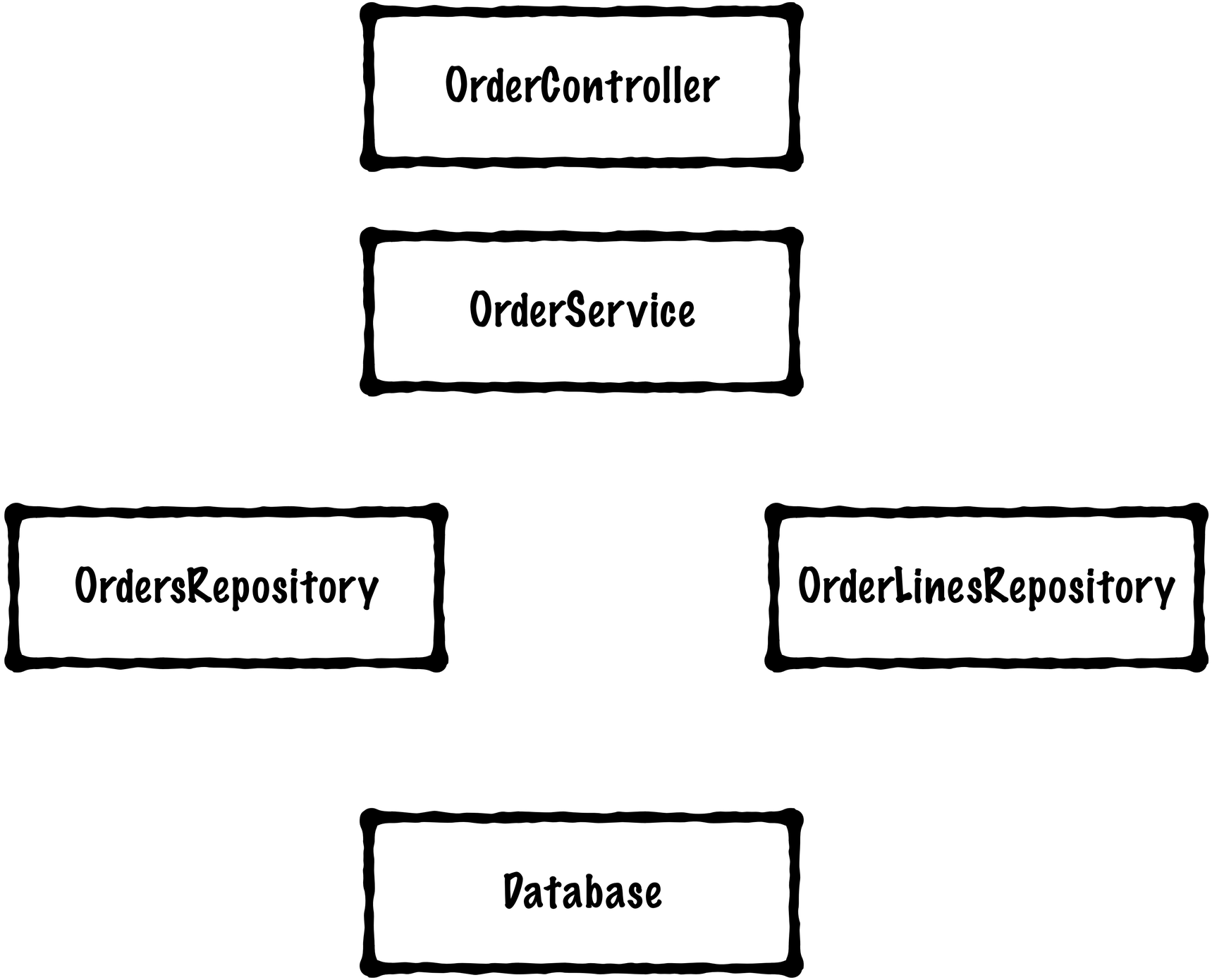
```
Order[] orders = LoadOrdersFromDb();

foreach (var orders in orders)
{
    order.Lines = LoadLinesFromDb(order.Id);
}
```

```
Order[] orders = LoadOrdersFromDb();

foreach (var orders in orders)
{
    order.Lines = LoadLinesFromDb(order.Id);
}
```

```
Order[] orders = LoadOrdersFromDb();

foreach (var orders in orders)
{
    order.Lines = LoadLinesFromDb(order.Id);
}
```

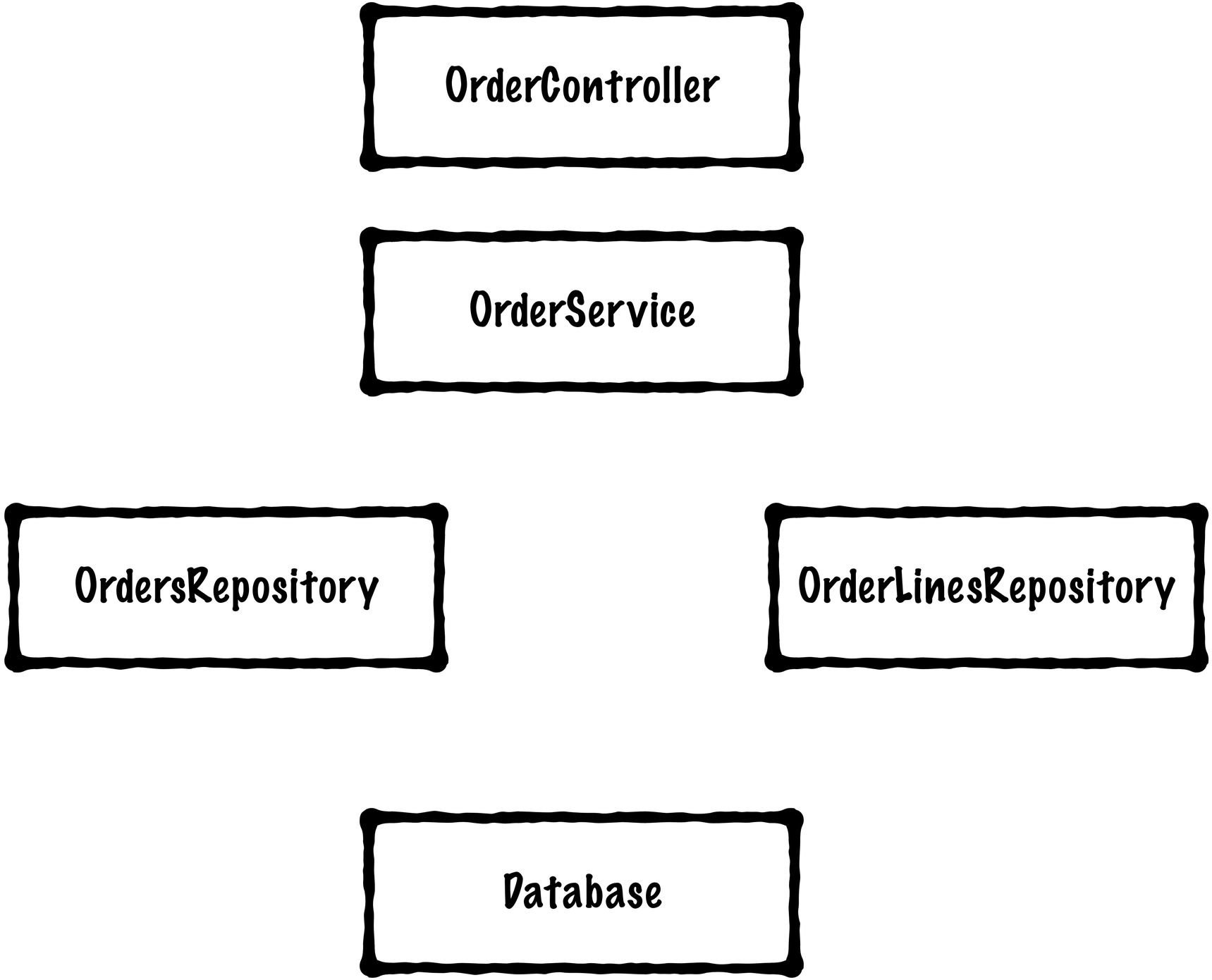
```
Order[] orders = LoadOrdersFromDb();

foreach (var orders in orders)
{
    order.Lines = LoadLinesFromDb(order.Id);
}
```

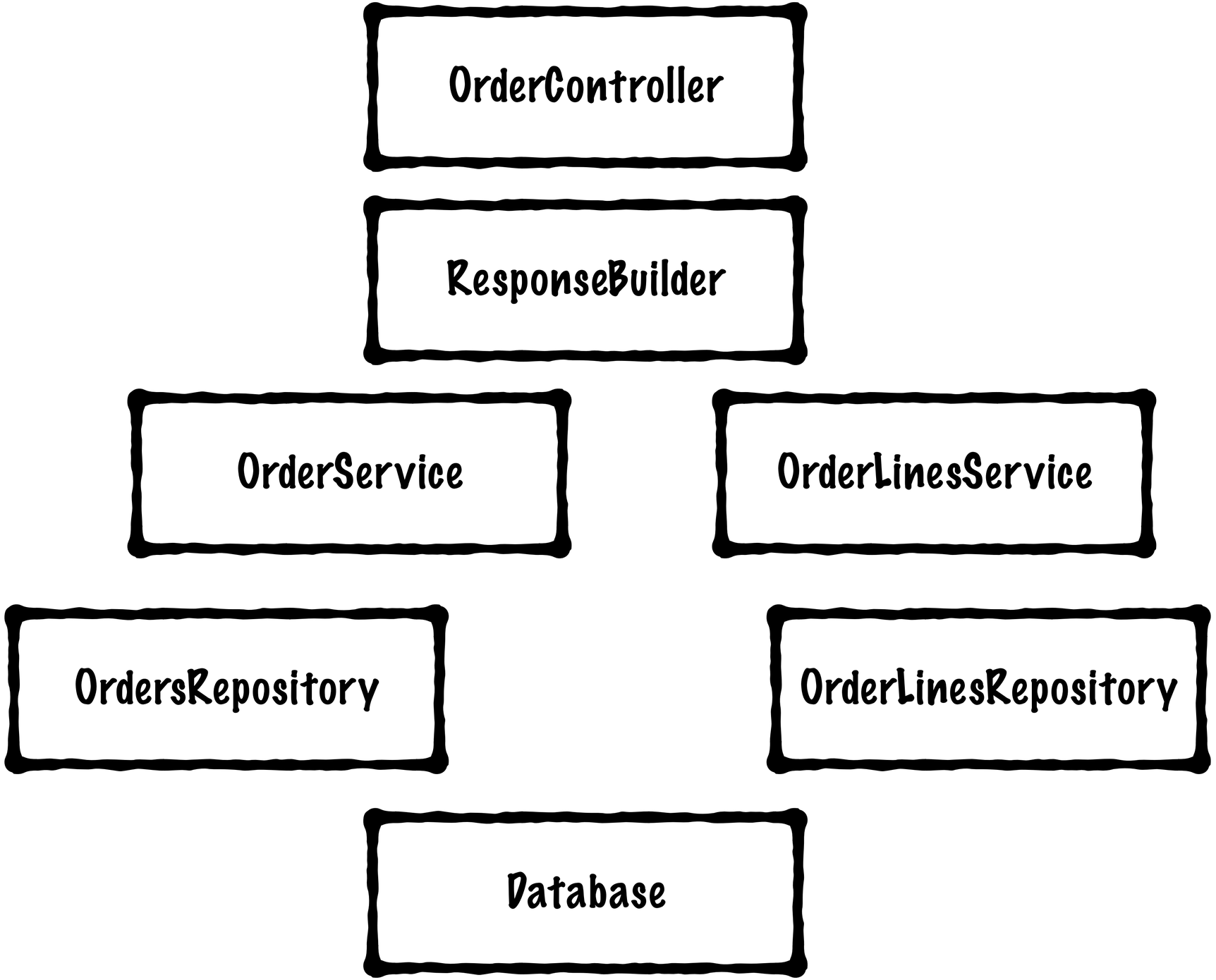
```
Order[] orders = LoadOrdersFromDb();

foreach (var orders in orders)
{
    order.Lines = LoadLinesFromDb(order.Id);
}
```

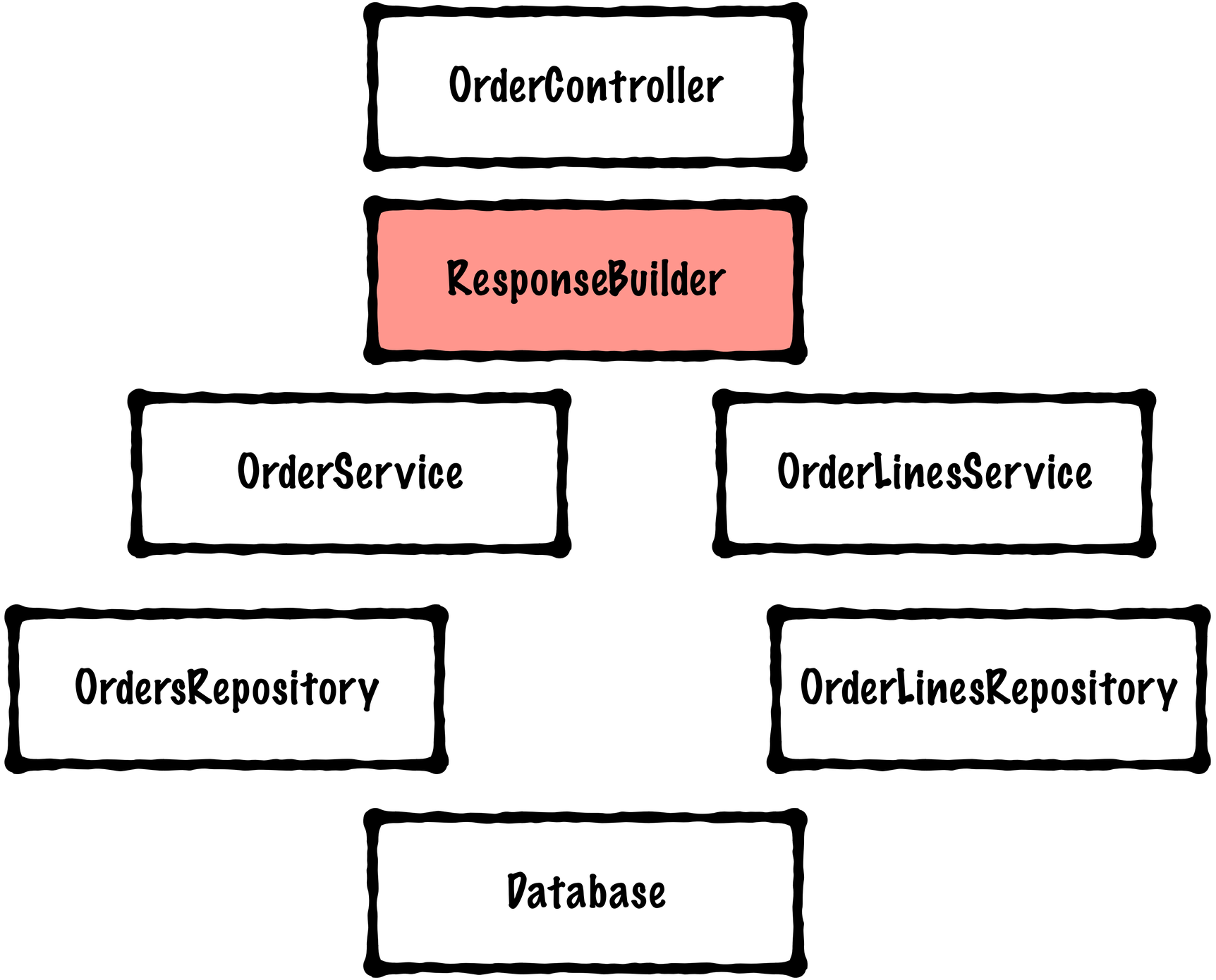

OrderController
OrderService
OrdersRepository
OrderLinesRepository
Database

```
Order[] orders = LoadOrdersFromDb();

foreach (var orders in orders)
{
    order.Lines = LoadLinesFromDb(order.Id);
}
```

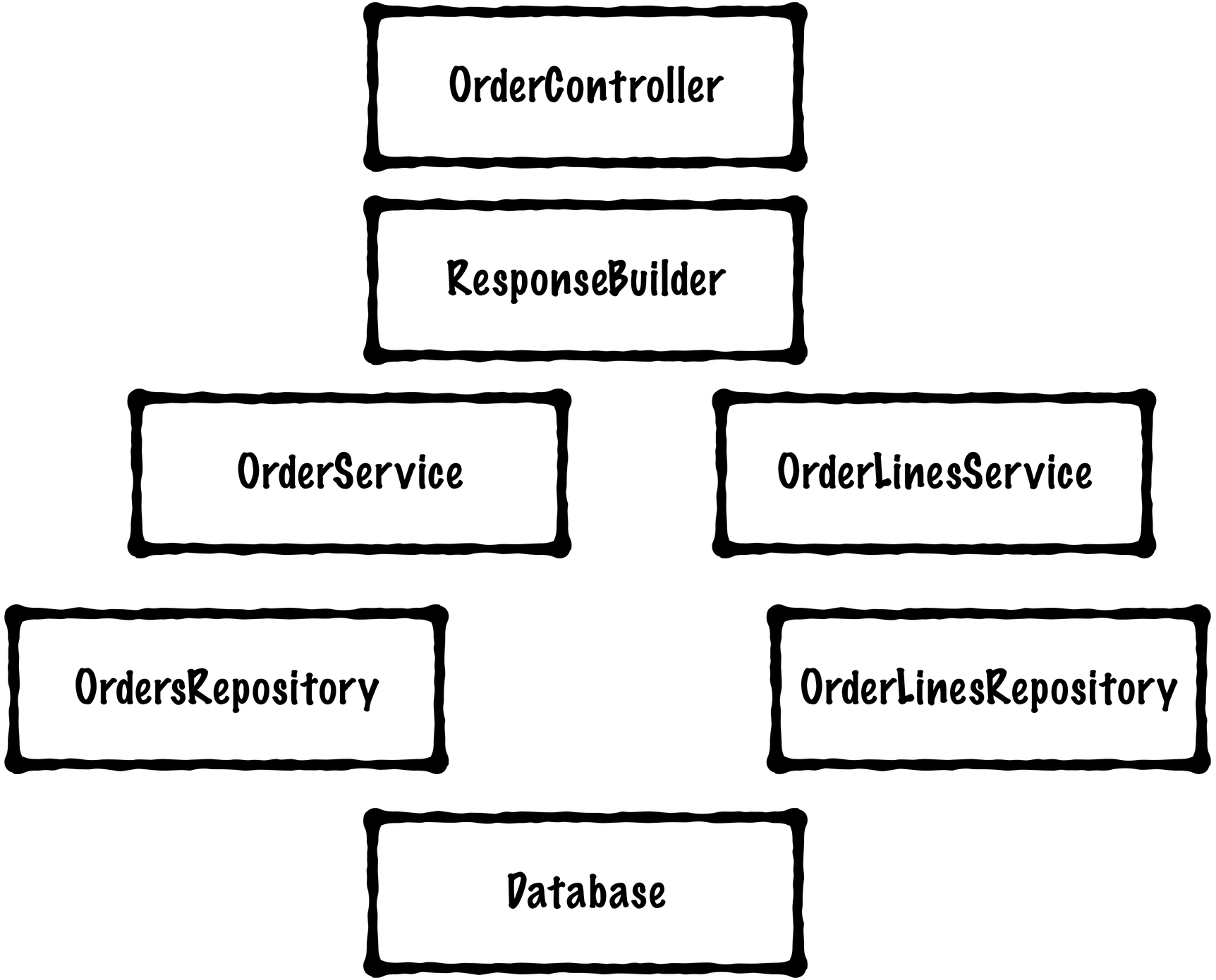
```
Order[] orders = orderRepo.GetAll();

foreach (var orders in orders)
{
    order.Lines = linesRepo.GetAll(order.Id);
}
```


OrderController
OrderService
OrdersRepository
OrderLinesRepository
Database

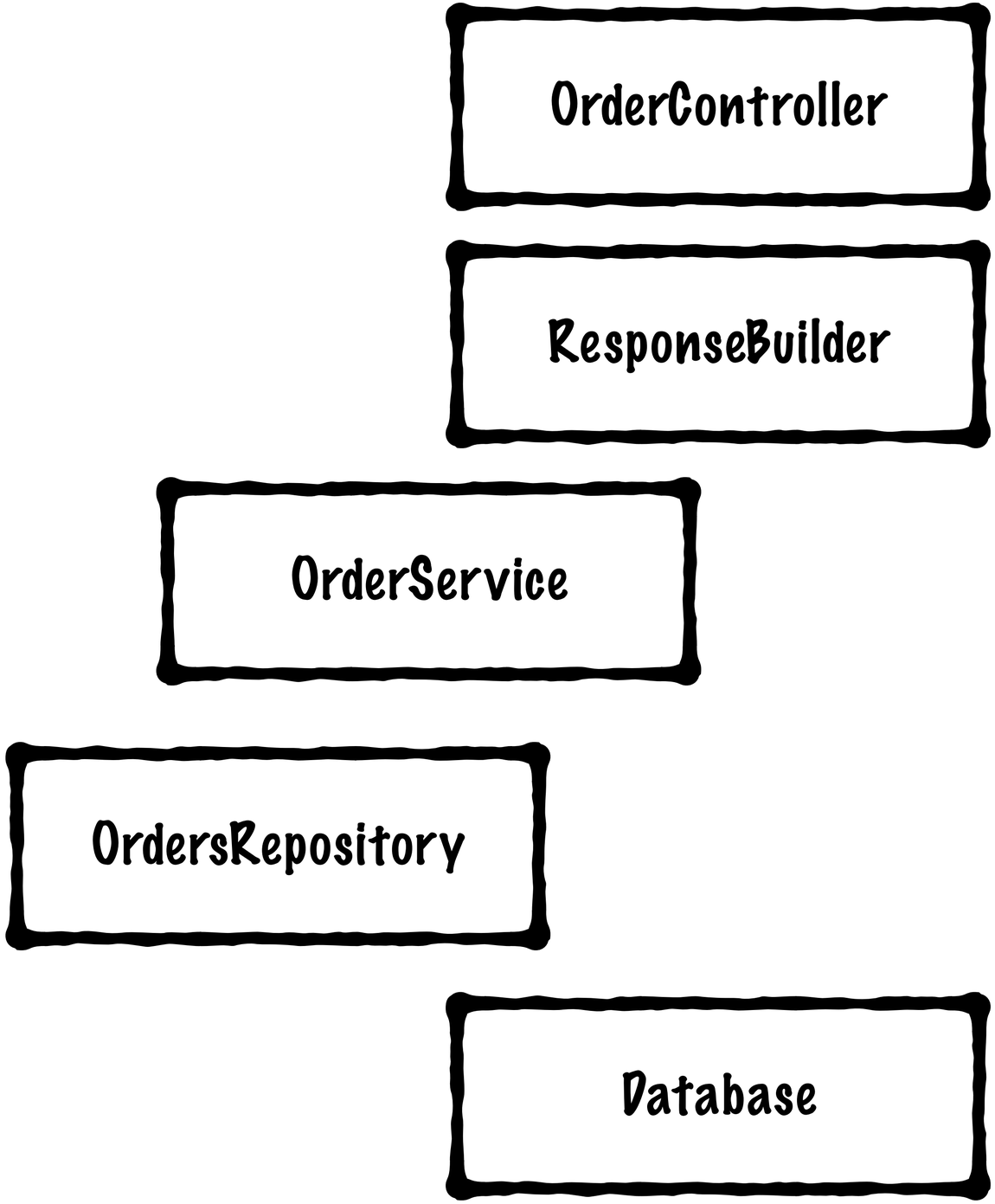

OrderController
ResponseBuilder
OrderService
OrderLinesService
OrdersRepository
OrderLinesRepository
Database


OrderController
ResponseBuilder
OrderService
OrderLinesService
OrdersRepository
OrderLinesRepository
Database

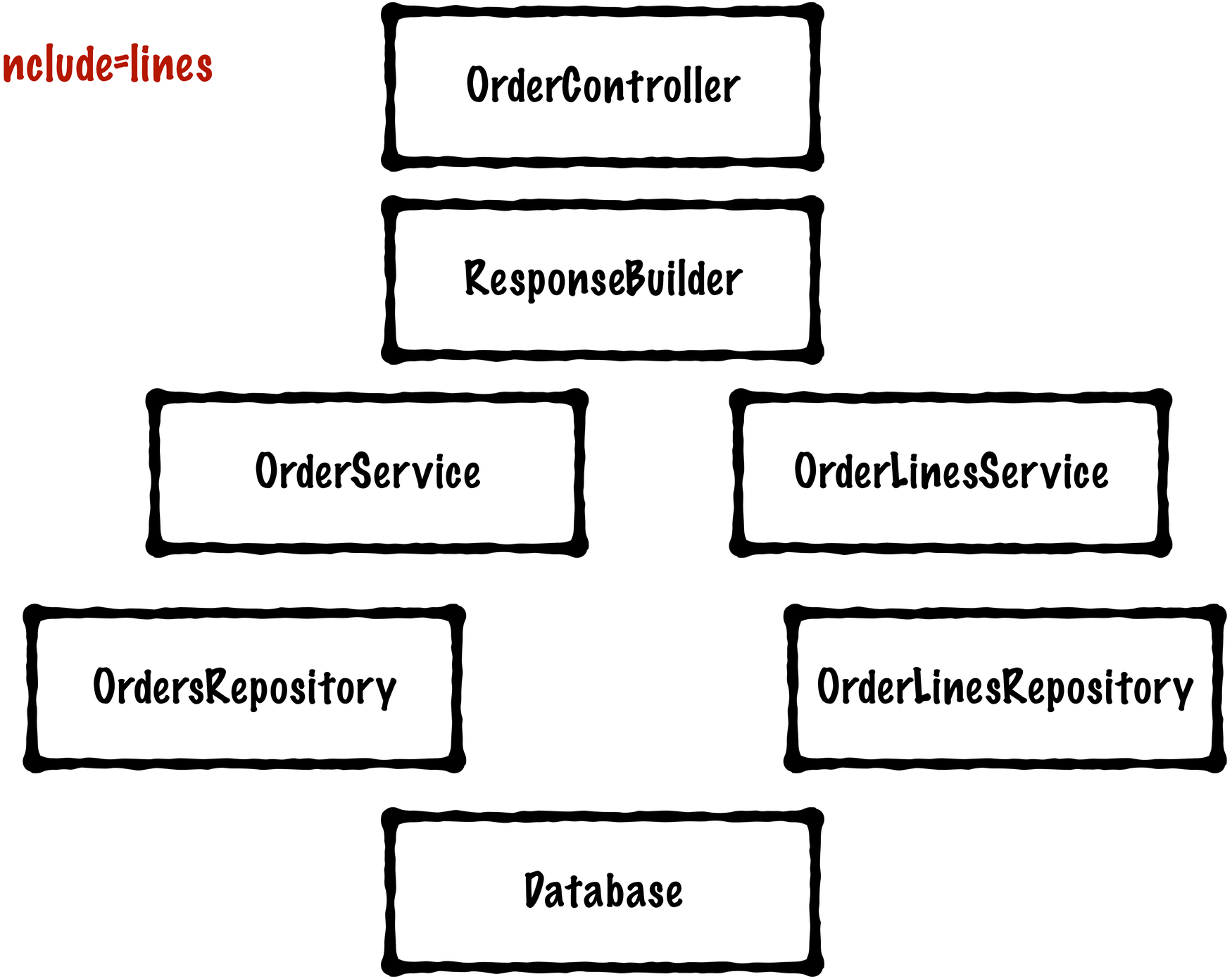
GET /orders


OrderController
ResponseBuilder
OrderService
OrderLinesService
OrdersRepository
OrderLinesRepository
Database

GET /orders


OrderController
ResponseBuilder
OrderService
OrdersRepository
Database

GET /orders?_include=lines



Cause 2: ORM Betrayal

```
Order[] orders = orderRepo.GetAll();
OrderResponse[] results = [];

foreach (var order in orders)
{
 OrderResponse orderResponse = Map(orders);
 orderResponse.Lines = Map(order.Lines);

 results.Add(orderResponse);
}
```

```
Order[] orders = orderRepo.GetAll();
OrderResponse[] results = [];

foreach (var order in orders)
{
 OrderResponse orderResponse = Map(orders);
 orderResponse.Lines = Map(order.Lines);

 results.Add(orderResponse);
}
```

```
Order[] orders = orderRepo.GetAll();
OrderResponse[] results = [];

foreach (var order in orders)
{
 OrderResponse orderResponse = Map(orders);
 orderResponse.Lines = Map(order.Lines);

 results.Add(orderResponse);
}
```

```
Order[] orders = orderRepo.GetAll();
OrderResponse[] results = [];

foreach (var order in orders)
{
 OrderResponse orderResponse = Map(orders);
 orderResponse.Lines = Map(order.Lines);

 results.Add(orderResponse);
}
```

```
Order[] orders = orderRepo.GetAll();
OrderResponse[] results = [];

foreach (var order in orders)
{
 OrderResponse orderResponse = Map(orders);
 orderResponse.Lines = Map(order.Lines);

 results.Add(orderResponse);
}
```

Do you see the problem?

- Your Senior Developer in the 15th GitHub pull request comment

```
Order[] orders = orderRepo.GetAll();
OrderResponse[] results = [];

foreach (var order in orders)
{
 OrderResponse orderResponse = Map(orders);
 orderResponse.Lines = Map(order.Lines);

 results.Add(orderResponse);
}
```

```
Order[] orders = orderRepo.GetAll();
OrderResponse[] results = [];

foreach (var order in orders)
{
 OrderResponse orderResponse = Map(orders);
 orderResponse.Lines = Map(order.Lines);

 results.Add(orderResponse);
}
```

Another DB query!

Solution 1: Eager Loading

```
SELECT * FROM orders;

SELECT * FROM order_lines WHERE order_id = 1;
SELECT * FROM order_lines WHERE order_id = 2;
SELECT * FROM order_lines WHERE order_id = 3;
SELECT * FROM order_lines WHERE order_id = 4;
SELECT * FROM order_lines WHERE order_id = 5;
SELECT * FROM order_lines WHERE order_id = 6;
SELECT * FROM order_lines WHERE order_id = 7;
SELECT * FROM order_lines WHERE order_id = 8;
SELECT * FROM order_lines WHERE order_id = 9;
…
```

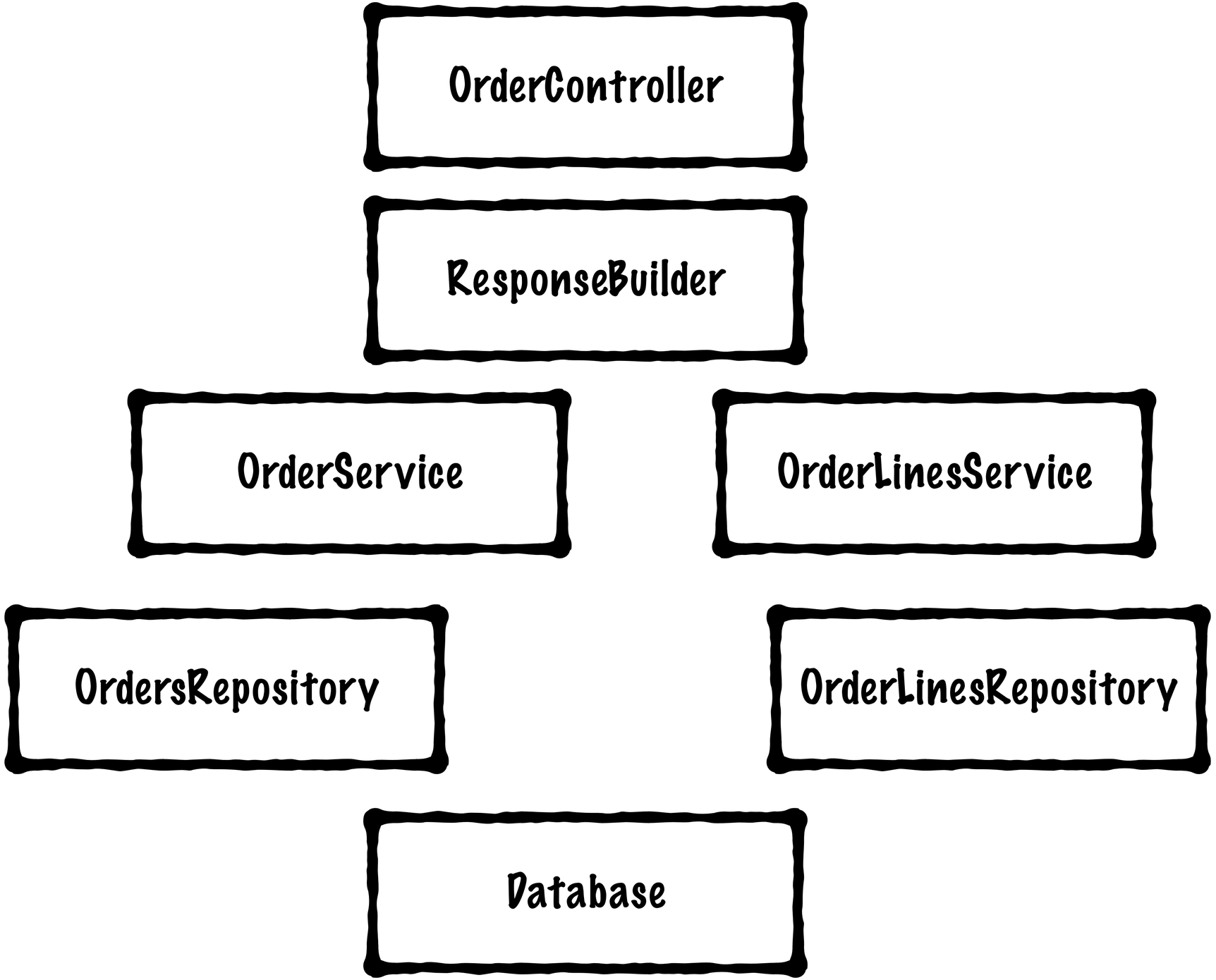
```
SELECT * FROM orders
JOIN order_lines
ON orders.id = order_lines.order_id
```

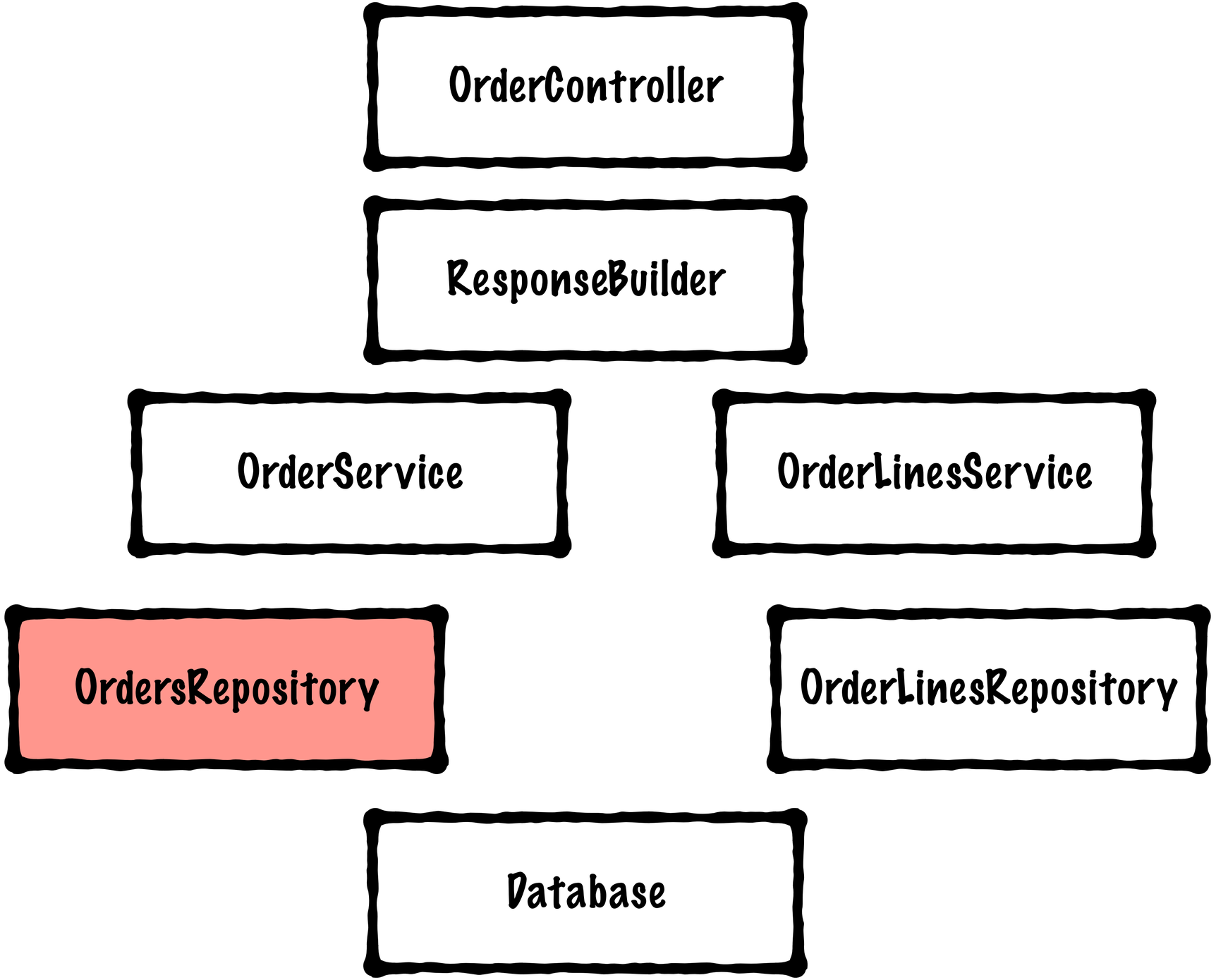
```
SELECT * FROM orders
JOIN order_lines
ON orders.id = order_lines.order_id
```

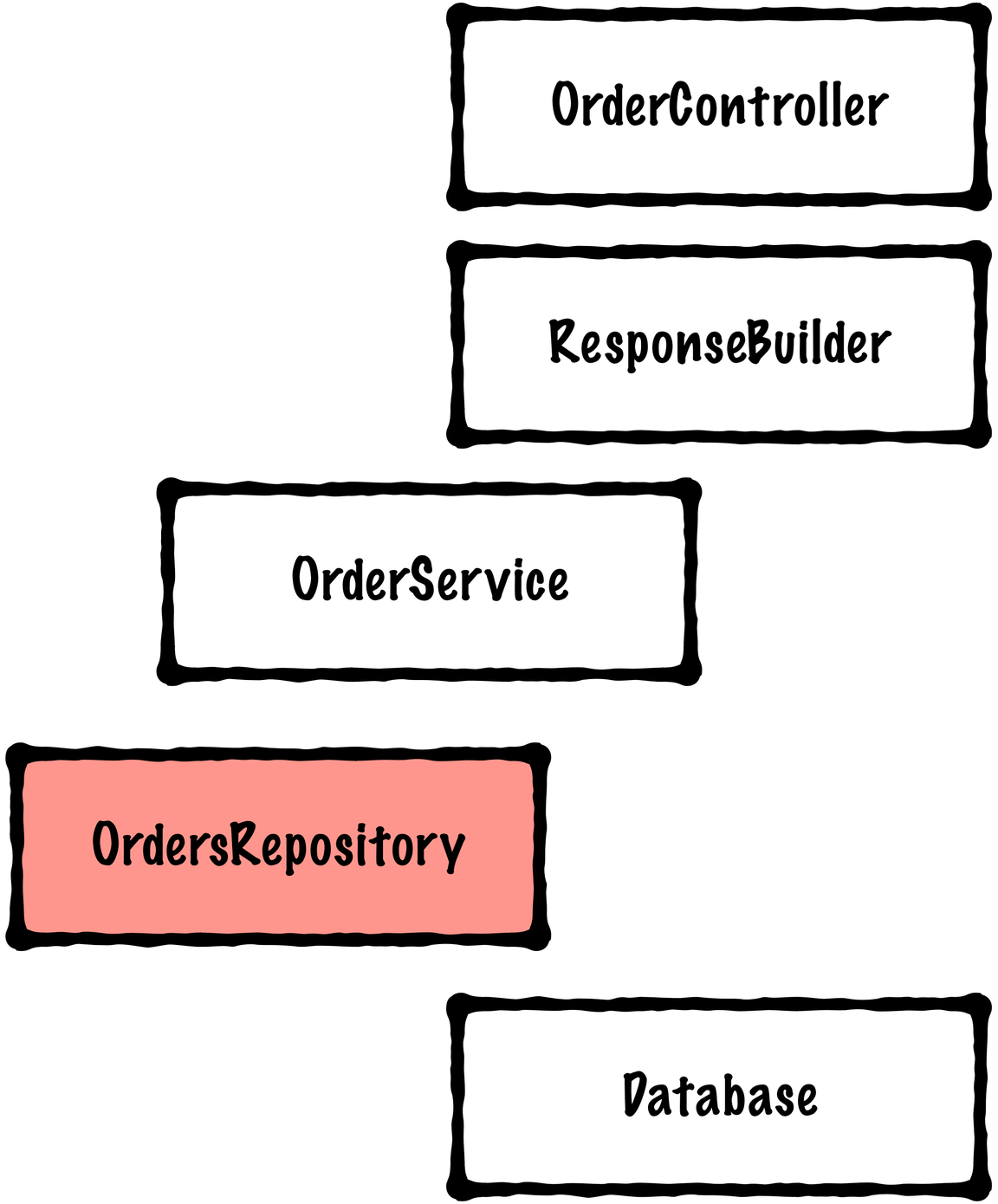
```
dbContext.Orders
        .Include(o => o.Lines)
        .ToList()
```

“Eager Loading”

“Includes”


OrderController
ResponseBuilder
OrderService
OrderLinesService
OrdersRepository
OrderLinesRepository
Database


OrderController
ResponseBuilder
OrderService
OrderLinesService
OrdersRepository
OrderLinesRepository
Database


OrderController
ResponseBuilder
OrderService
OrdersRepository
Database

“Ugh, reworking the whole stack to fix this will get take a million points and make my PM mad at me”

- Me

“There must be an easier way!”

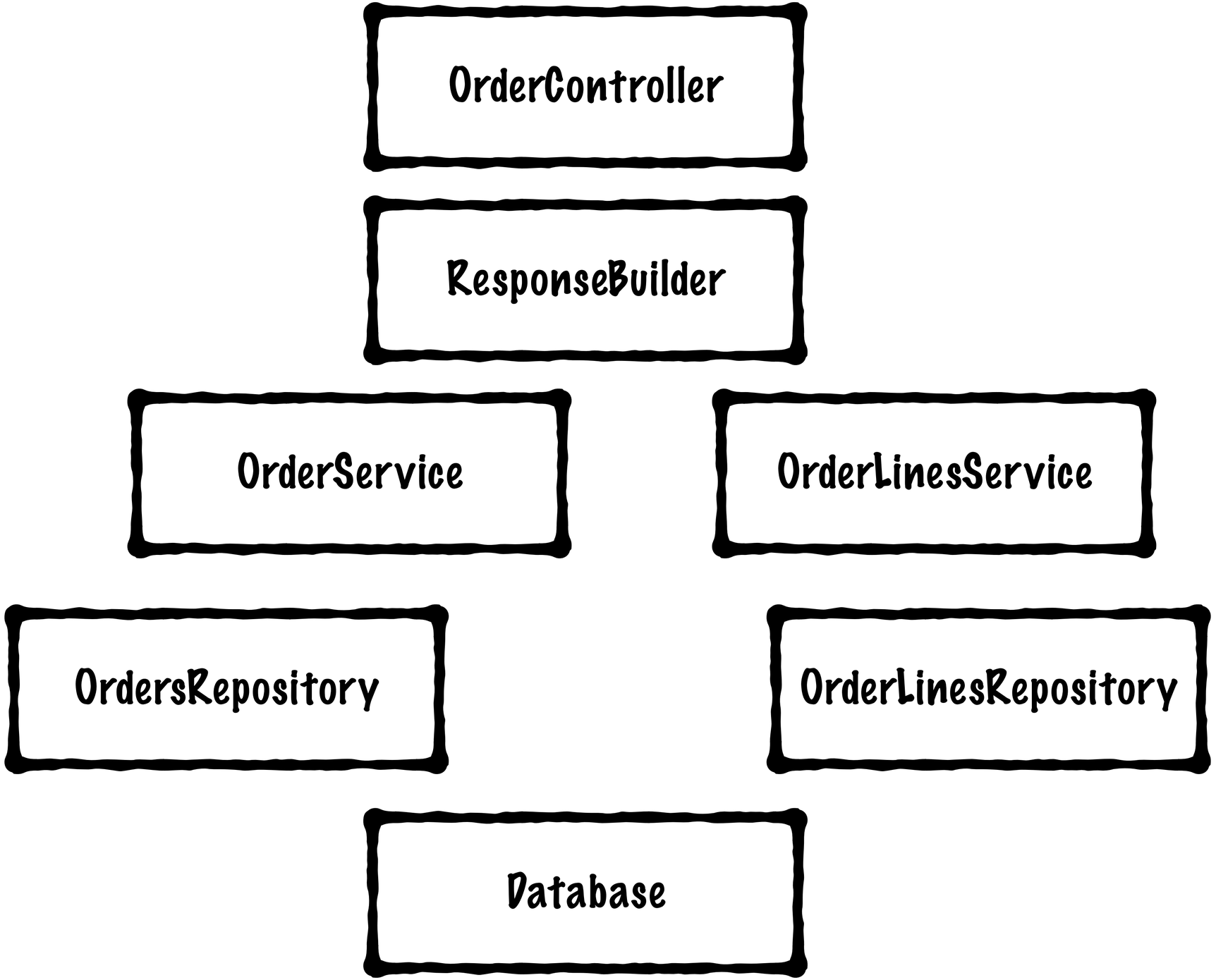
- Me

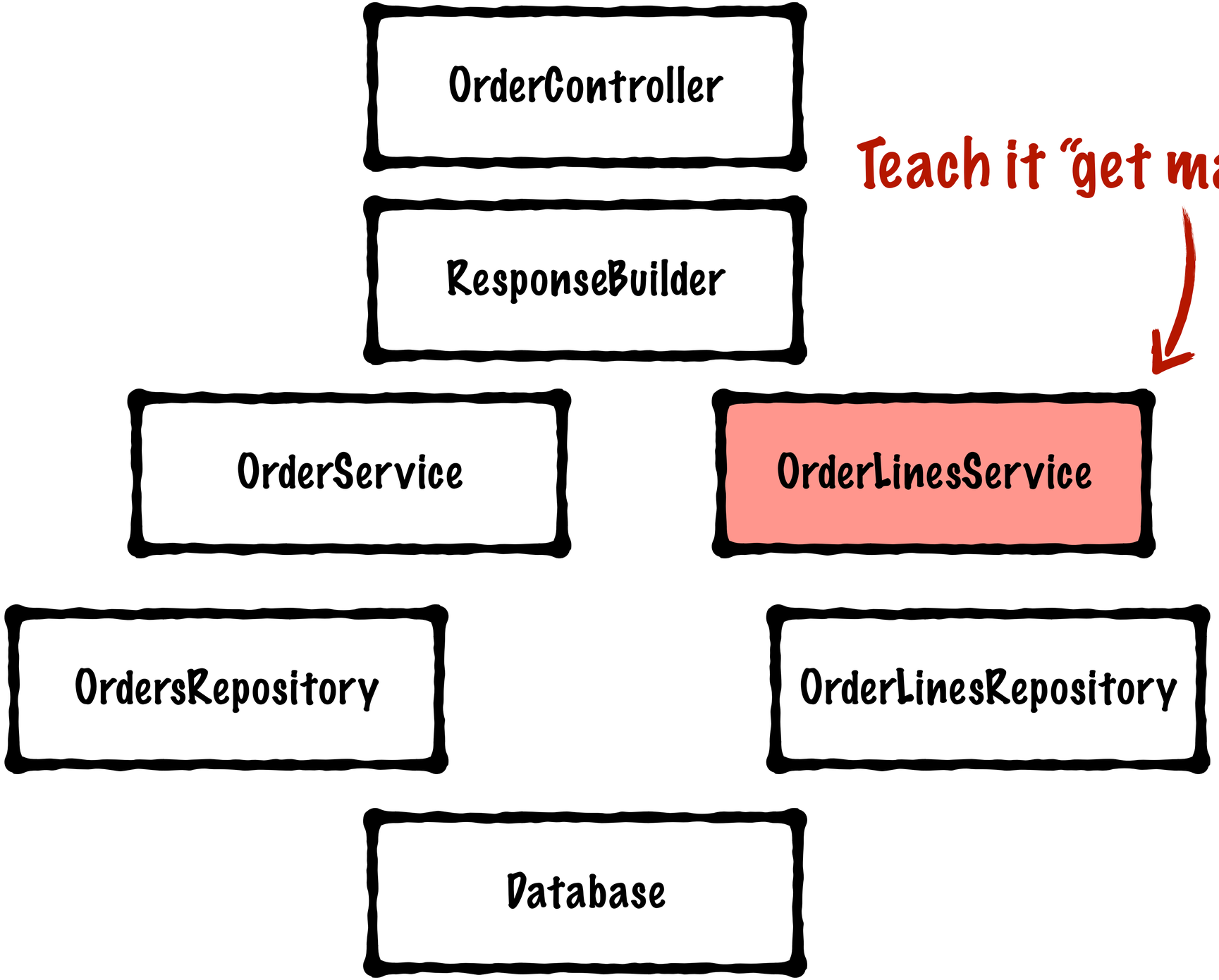
Solution 2: Batching

```
SELECT * FROM orders;

SELECT * FROM order_lines
WHERE order_id in (1,2,3,4,5,6,7,8,9,10);
```

“Split Query”

“Batch Query”


OrderController
ResponseBuilder
OrderService
OrderLinesService
OrdersRepository
OrderLinesRepository
Database


OrderController
ResponseBuilder
OrderService
OrderLinesService
OrdersRepository
OrderLinesRepository
Database
Teach it "get many by ids"

```
const orders = orderService.getAll()

const ids = orders.map(o => o.id);

const lines =
    orderLinesService.getByIds(ids);

// TODO: match lines to orders
```

```
const orders = orderService.getAll()

const ids = orders.map(o => o.id);

const lines =
    orderLinesService.getByIds(ids);

// TODO: match lines to orders
```

```
const orders = orderService.getAll()

const ids = orders.map(o => o.id);

const lines =
    orderLinesService.getByIds(ids);

// TODO: match lines to orders
```

```
const orders = orderService.getAll()

const ids = orders.map(o => o.id);

const lines =
    orderLinesService.getByIds(ids);

// TODO: match lines to orders
```

http://localhost:5109/sql/orders-split-query

Run with SQL Logging Enabled

dotnet run

```
      FROM products AS p
      WHERE p.id = @__p_0
      LIMIT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[@__p_0='?' (DbType = Int64)], CommandType='Text', CommandTimeout='30']
      SELECT p.id, p.brand, p.category, p.cost, p.description, p.is_active, p.name, p.price, p.sku, p.stock_quantity, p.weight
      FROM products AS p
      WHERE p.id = @__p_0
      LIMIT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (92ms) [Parameters=[@__p_0='?' (DbType = Int64)], CommandType='Text', CommandTimeout='30']
      SELECT o.id, o.discount_percent, o.line_number, o.order_id, o.product_id, o.quantity, o.tax_amount, o.unit_price
      FROM order_lines AS o
      WHERE o.order_id = @__p_0
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[@__p_0='?' (DbType = Int64)], CommandType='Text', CommandTimeout='30']
      SELECT p.id, p.brand, p.category, p.cost, p.description, p.is_active, p.name, p.price, p.sku, p.stock_quantity, p.weight
      FROM products AS p
      WHERE p.id = @__p_0
      LIMIT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (0ms) [Parameters=[@__p_0='?' (DbType = Int64)], CommandType='Text', CommandTimeout='30']
      SELECT p.id, p.brand, p.category, p.cost, p.description, p.is_active, p.name, p.price, p.sku, p.stock_quantity, p.weight
      FROM products AS p
      WHERE p.id = @__p_0
      LIMIT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (0ms) [Parameters=[@__p_0='?' (DbType = Int64)], CommandType='Text', CommandTimeout='30']
      SELECT p.id, p.brand, p.category, p.cost, p.description, p.is_active, p.name, p.price, p.sku, p.stock_quantity, p.weight
      FROM products AS p
      WHERE p.id = @__p_0
      LIMIT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (0ms) [Parameters=[@__p_0='?' (DbType = Int64)], CommandType='Text', CommandTimeout='30']
      SELECT p.id, p.brand, p.category, p.cost, p.description, p.is_active, p.name, p.price, p.sku, p.stock_quantity, p.weight
      FROM products AS p
      WHERE p.id = @__p_0
      LIMIT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (1ms) [Parameters=[@__p_0='?' (DbType = Int64)], CommandType='Text', CommandTimeout='30']
      SELECT p.id, p.brand, p.category, p.cost, p.description, p.is_active, p.name, p.price, p.sku, p.stock_quantity, p.weight
      FROM products AS p
      WHERE p.id = @__p_0
      LIMIT 1
info: Microsoft.EntityFrameworkCore.Database.Command[20101]
      Executed DbCommand (0ms) [Parameters=[@__p_0='?' (DbType = Int64)], CommandType='Text', CommandTimeout='30']
      SELECT p.id, p.brand, p.category, p.cost, p.description, p.is_active, p.name, p.price, p.sku, p.stock_quantity, p.weight
      FROM products AS p
      WHERE p.id = @__p_0
      LIMIT 1
```

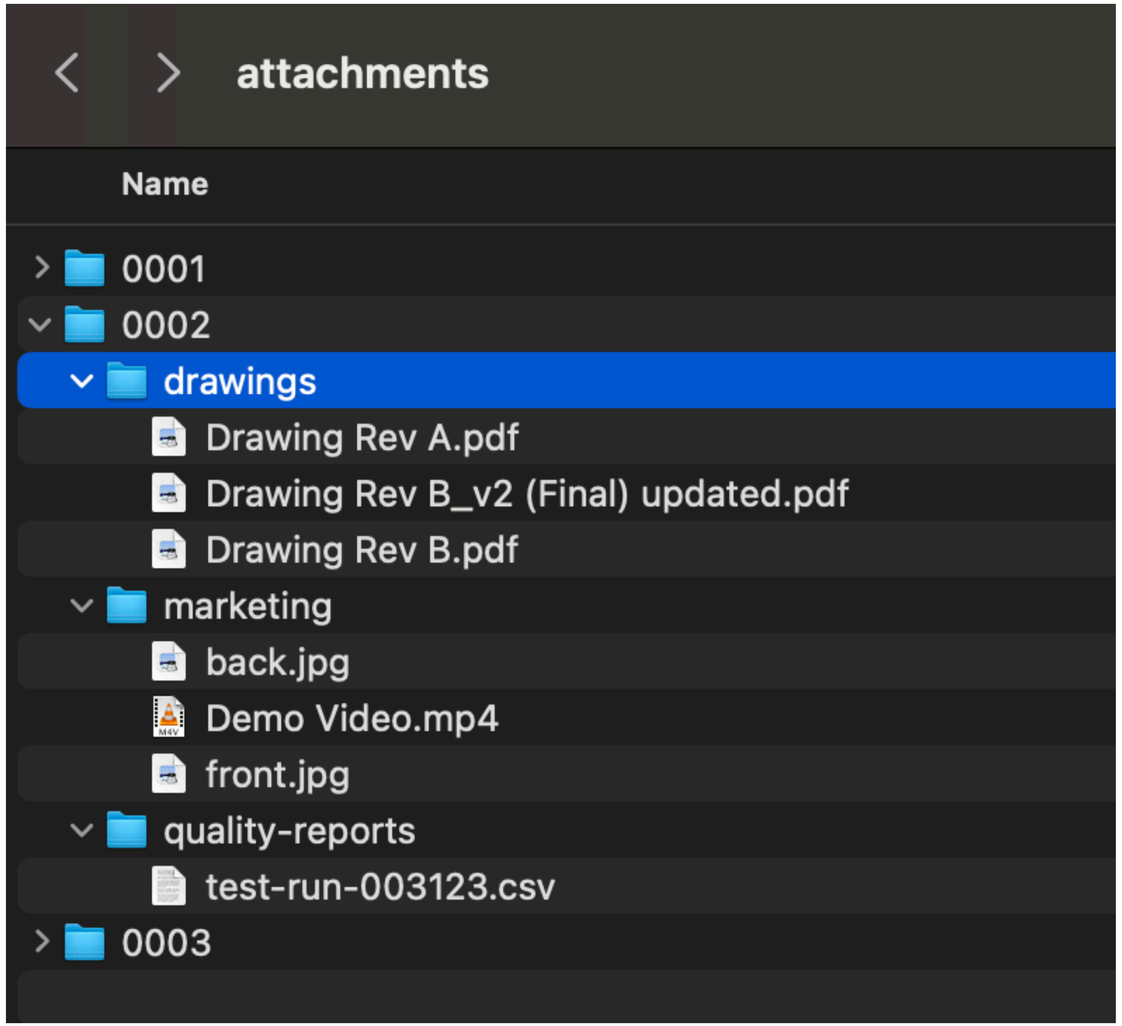
Disable Lazy Loading

```
Order[] orders = orderRepo.GetAll();
OrderResponse[] results = [];

foreach (var order in orders)
{
 OrderResponse orderResponse = Map(orders);
 orderResponse.Lines = Map(order.Lines);

 results.Add(orderResponse);
}
```

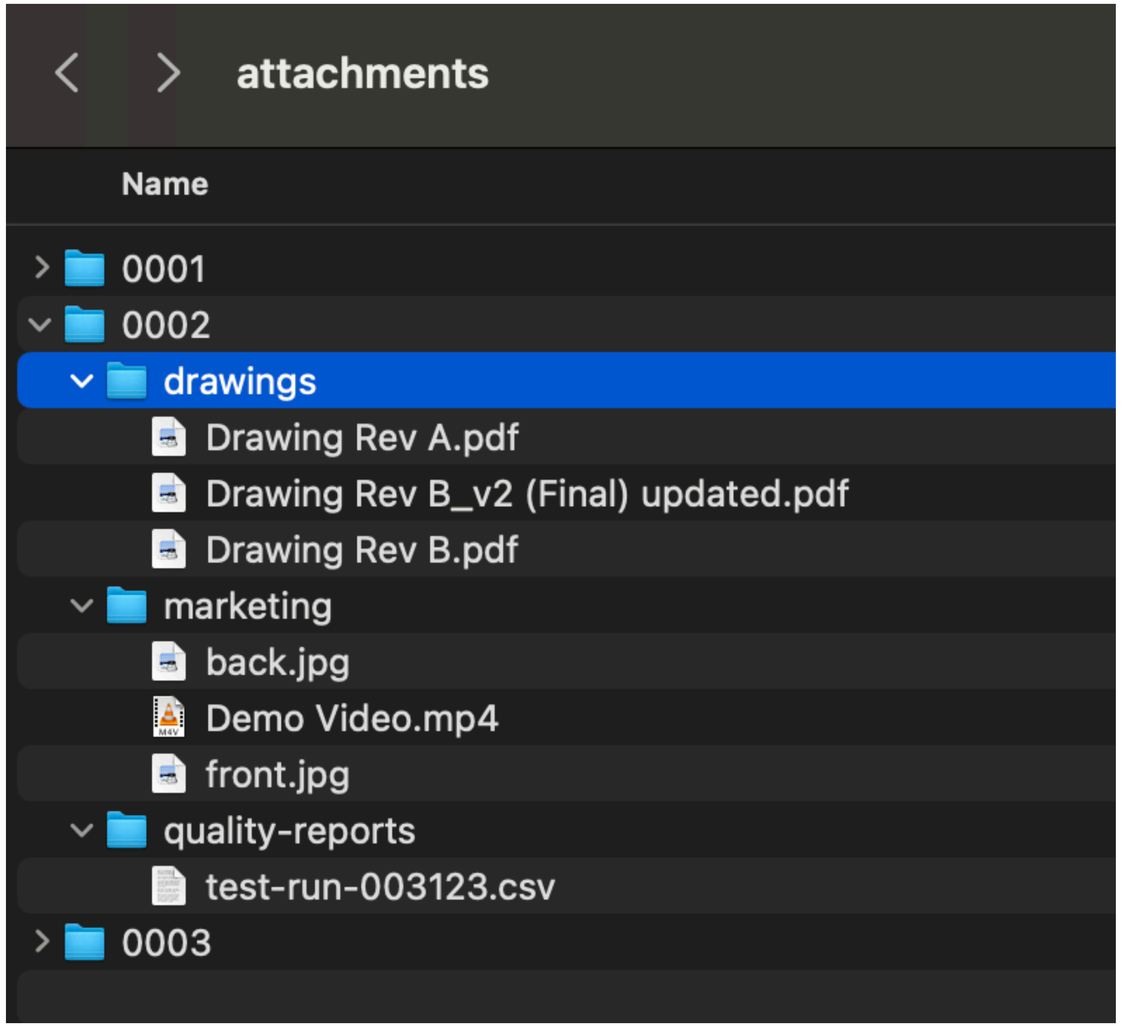
Database query!

```
Order[] orders = orderRepo.GetAll();
OrderResponse[] results = [];

foreach (var order in orders)
{
 OrderResponse orderResponse = Map(orders);
 orderResponse.Lines = Map(order.Lines);

 results.Add(orderResponse);
}
```

Drop your ORM and write some SQL




```
create table folders (
    id bigserial primary key,
    name text not null,
    parent_id bigint,
    foreign key (parent_id) references folders(id)
);

create table files (
    id bigserial primary key,
    name text not null,
    folder_id bigint,
    size bigint not null,
    foreign key (folder_id) references folders(id)
);
```

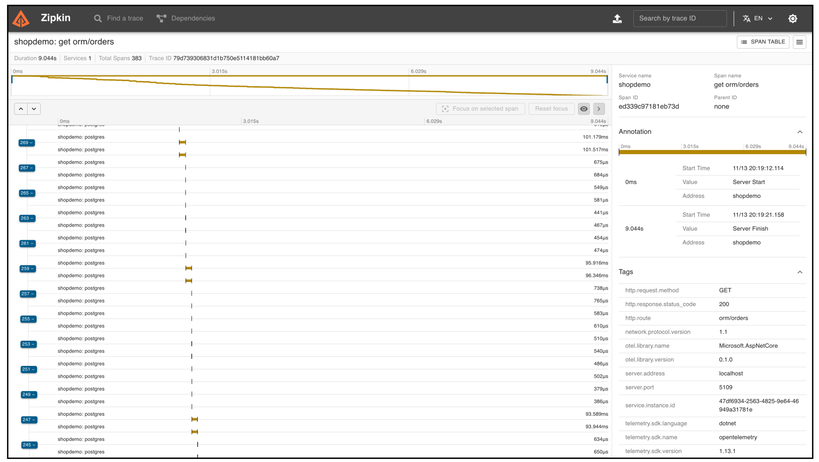



```
create table folders (
    id bigserial primary key,
    name text not null,
    parent_id bigint,
    foreign key (parent_id) references folders(id)
);

create table files (
    id bigserial primary key,
    name text not null,
    folder_id bigint,
    size bigint not null,
    foreign key (folder_id) references folders(id)
);
```

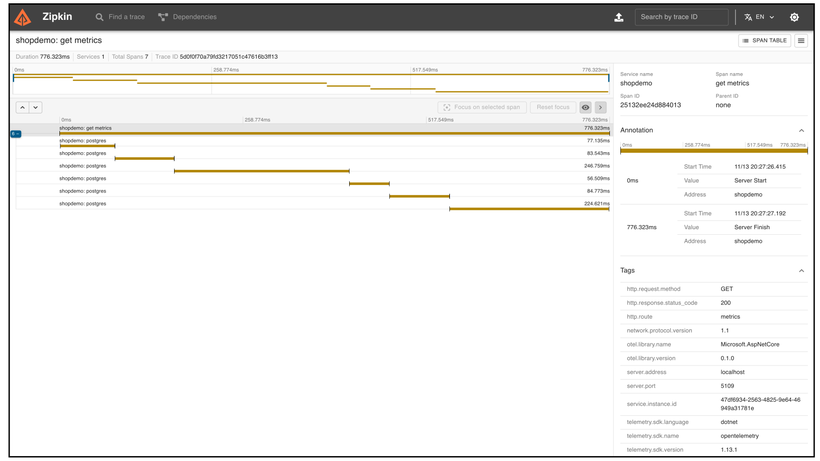
Good luck getting your ORM to search that in one round-trip

```
with recursive cte as (
    select f.*, name as path
    from folders f where f.parent_id is null

    union all

    select
        f.*,
        cte.path || '/' || f.name as path
    from folders f join cte on f.parent_id = cte.id
)
select files.id, cte.path || '/' || files.name as
path
from cte join files on files.folder_id = cte.id
order by path
```

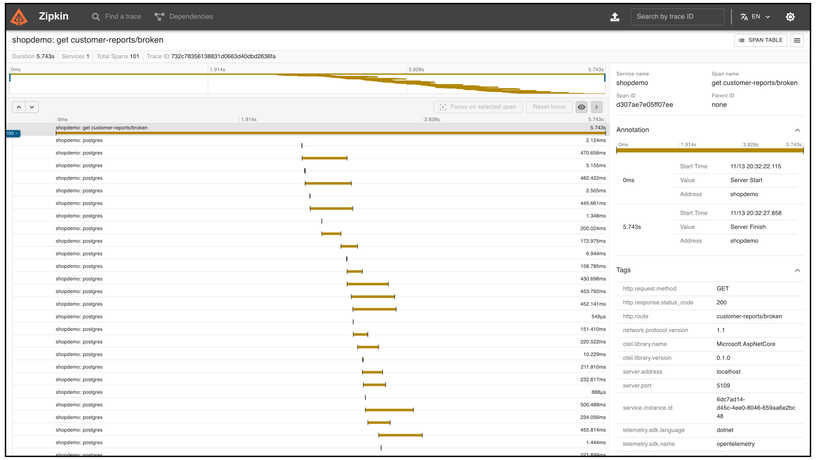
```
with recursive cte as (
    select f.*, name as path
    from folders f where f.parent_id is null

    union all

    select
        f.*,
        cte.path || '/' || f.name as path
    from folders f join cte on f.parent_id = cte.id
)
select files.id, cte.path || '/' || files.name as
path
from cte join files on files.folder_id = cte.id
order by path
```



Many Small Spans



Large Spans

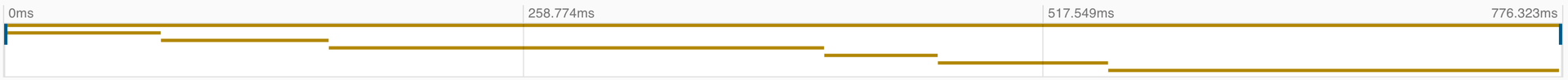


Span Gaps

shopdemo: get metrics

SPAN TABLE

Duration 776.323ms | Services 1 | Total Spans 7 | Trace ID 5d0f0f70a79fd3217051c47616b3ff13



Focus on selected span | Reset focus

	0ms	258.774ms	517.549ms	776.323ms
6 –	shopdemo: get metrics			776.323ms
	shopdemo: postgres			77.135ms
	shopdemo: postgres			83.543ms
	shopdemo: postgres			246.759ms
	shopdemo: postgres			56.509ms
	shopdemo: postgres			84.773ms
	shopdemo: postgres			224.621ms

Service name: shopdemo

Span name: get metrics

Span ID: 25132ee24d884013

Parent ID: none

Annotation

0ms	Start Time	11/13 20:27:26.415
	Value	Server Start
	Address	shopdemo
776.323ms	Start Time	11/13 20:27:27.192
	Value	Server Finish
	Address	shopdemo

Tags

http.request.method	GET
http.response.status_code	200
http.route	metrics
network.protocol.version	1.1
otel.library.name	Microsoft.AspNetCore
otel.library.version	0.1.0
server.address	localhost
server.port	5109
service.instance.id	47df6934-2563-4825-9e64-46949a31781e
telemetry.sdk.language	dotnet
telemetry.sdk.name	opentelemetry
telemetry.sdk.version	1.13.1

http://localhost:5109/metrics?range=today


Pathetic.

```
SELECT
    P.ID AS PRODUCT_ID,
    P.NAME AS PRODUCT_NAME,
    P.SKU,
    SUM(OL.QUANTITY) AS TOTAL_QUANTITY,
    SUM(OL.QUANTITY * OL.UNIT_PRICE) AS TOTAL_REVENUE,
    COUNT(DISTINCT OL.ORDER_ID) AS ORDER_COUNT,
    AVG(OL.UNIT_PRICE) AS AVG_UNIT_PRICE
FROM
    PRODUCTS P
    INNER JOIN ORDER_LINES OL ON P.ID = OL.PRODUCT_ID
    INNER JOIN ORDERS O ON OL.ORDER_ID = O.ID
WHERE
    O.CREATED_AT >= '2025-12-09'
    AND O.CREATED_AT < '2025-12-11'
GROUP BY
    P.ID,
    P.NAME,
    P.SKU
```

```
EXPLAIN SELECT
    P.ID AS PRODUCT_ID,
    P.NAME AS PRODUCT_NAME,
    P.SKU,
    SUM(OL.QUANTITY) AS TOTAL_QUANTITY,
    SUM(OL.QUANTITY * OL.UNIT_PRICE) AS TOTAL_REVENUE,
    COUNT(DISTINCT OL.ORDER_ID) AS ORDER_COUNT,
    AVG(OL.UNIT_PRICE) AS AVG_UNIT_PRICE
FROM
    PRODUCTS P
    INNER JOIN ORDER_LINES OL ON P.ID = OL.PRODUCT_ID
    INNER JOIN ORDERS O ON OL.ORDER_ID = O.ID
WHERE
    O.CREATED_AT >= '2025-12-09'
    AND O.CREATED_AT < '2025-12-11'
GROUP BY
    P.ID,
    P.NAME,
    P.SKU
```

http://localhost:8080/browser/

```
Limit  (cost=147418.55..147418.56 rows=5 width=123)
  ->  Sort  (cost=147418.55..147418.56 rows=5 width=123)
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate  (cost=70450.79..147418.49 rows=5 width=123)
              Group Key: p.id
              ->  Incremental Sort  (cost=70450.79..147418.33 rows=5 width=61)
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop  (cost=51208.96..147418.10 rows=5 width=61)
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
(cost=0.14..15.64 rows=100 width=43)
                          ->  Materialize  (cost=51208.81..147394.97 rows=5 width=26)
                                ->  Gather  (cost=51208.81..147394.95 rows=5 width=26)
                                      Workers Planned: 2
                                      ->  Parallel Hash Join  (cost=50208.81..146394.45
rows=2 width=26)
                                            Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
(cost=0.00..89563.83 rows=2522583 width=26)
                                            ->  Parallel Hash  (cost=50208.80..50208.80
rows=1 width=8)
                                                  ->  Parallel Seq Scan on orders o
(cost=0.00..50208.80 rows=1 width=8)
                                                        Filter: ((created_at >=
'2025-12-09 00:00:00+00'::timestamp with time zone) AND (created_at < '2025-12-11
00:00:00+00'::timestamp with time zone))
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```

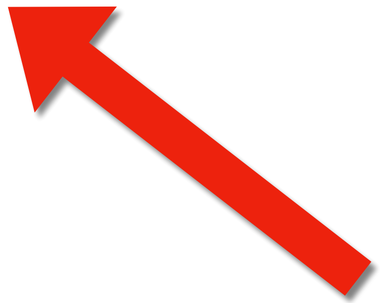
```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```

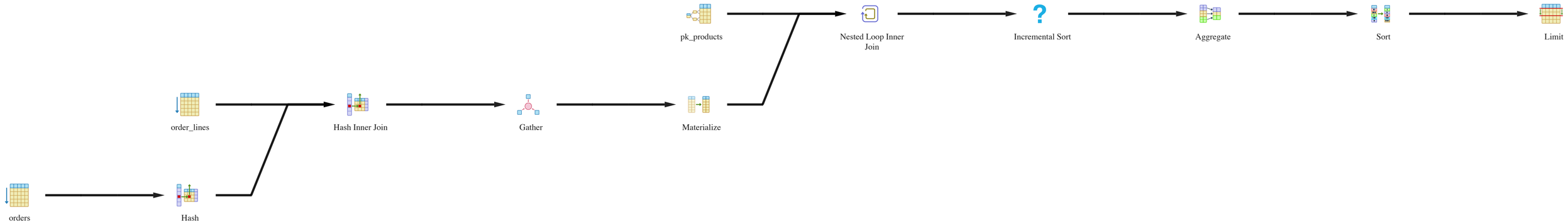
```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```

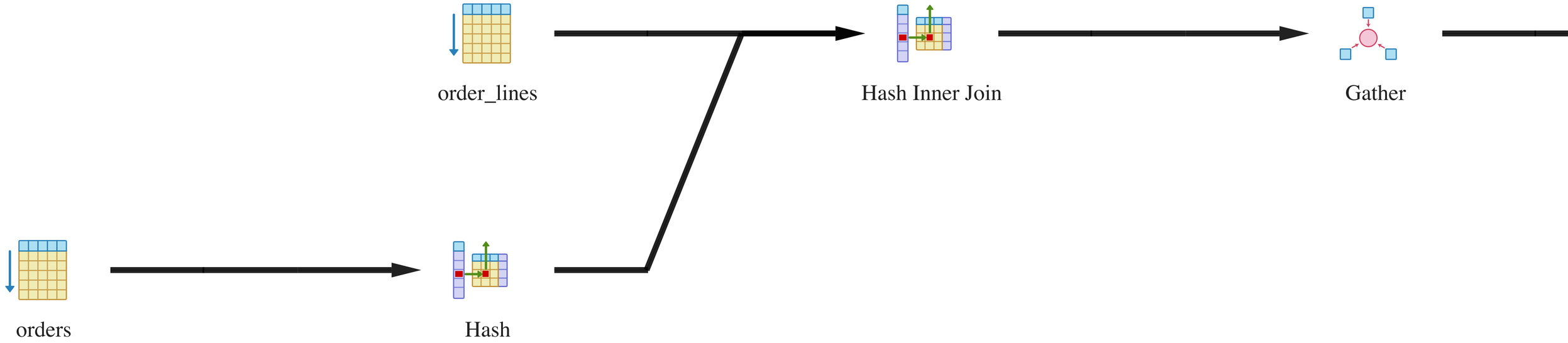
```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```



http://localhost:8080/browser/


orders
Hash
order_lines
Hash Inner Join
Gather
Materialize
pk_products
Nested Loop Inner Join
Incremental Sort
Aggregate
Sort
Limit


order_lines
Hash Inner Join
Gather
orders
Hash

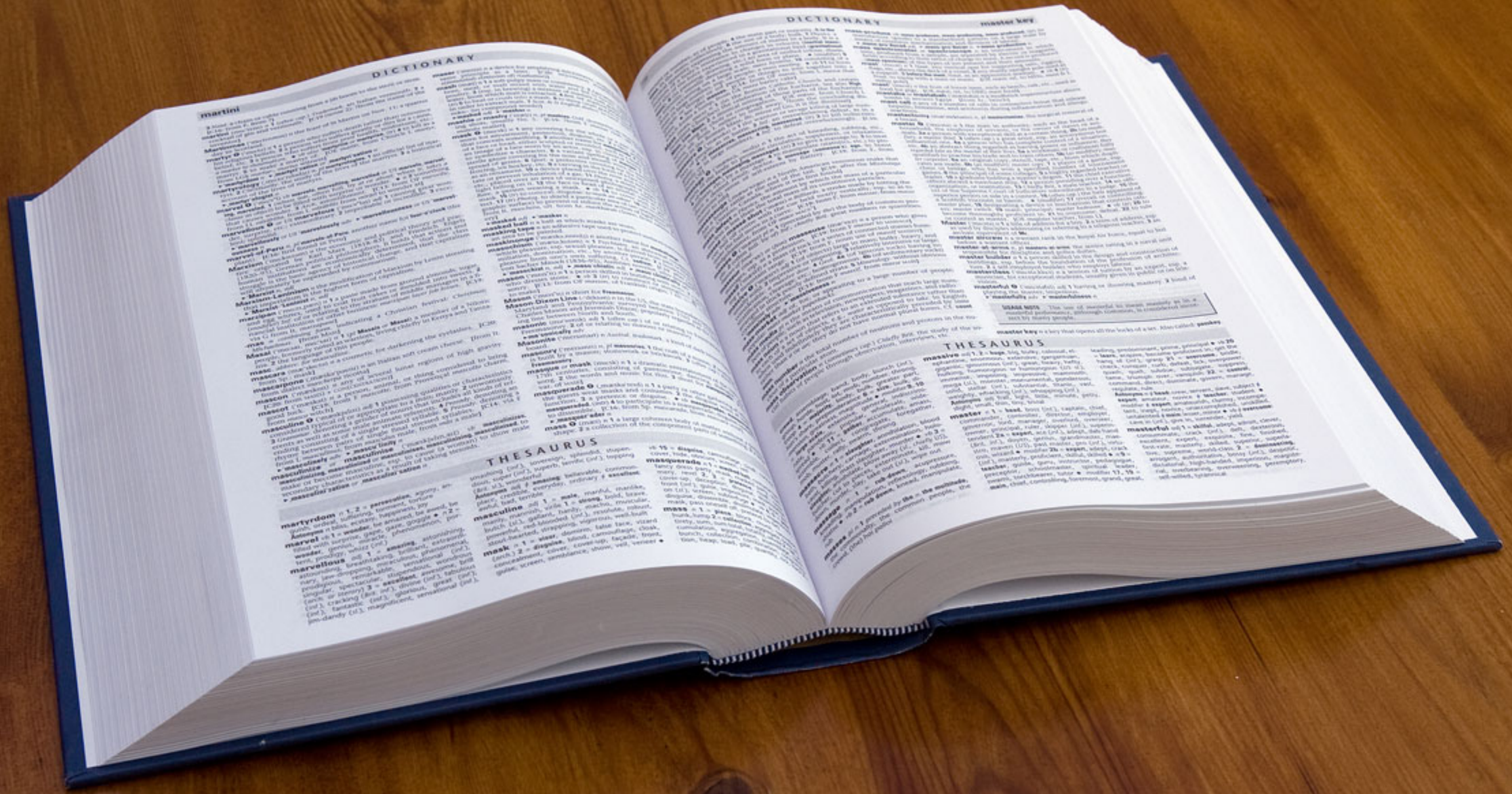

Incremental Sort
Aggregate
Sort
Limit

Sequential Scan

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```


martini
DICTIONARY
martyrdom
THESAURUS
DICTIONARY
master key
THESAURUS


INDEX
Archimedes,
Area, eleme
in pola
in
74, 442
Absolute co
Absolute
Absolu
Ab
verse compone
and normal components of,
of, 11, 243
00
ates, 325, 465
coordinates, 107, 459
of, 99, 325
ved surface, 474
a polygon, 20
of a surface of revolution, 333
of a triangle, 19
of an ellipse, 288
Astroid, 10,.221
Asymptotes, 150, 173
265

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Incremental Sort
                    Sort Key: p.id, ol.order_id
                    Presorted Key: p.id
                    ->  Nested Loop
                          Join Filter: (p.id = ol.product_id)
                          ->  Index Scan using pk_products on products p
                          ->  Materialize
                                ->  Gather
                                      Workers Planned: 2
                                      ->  Parallel Hash Join
                                          Hash Cond: (ol.order_id = o.id)
                                            ->  Parallel Seq Scan on order_lines ol
                                            ->  Parallel Hash
                                                  ->  Parallel Seq Scan on orders o
                                                        Filter: ((created_at >= '2025-12-09'
                                                             AND (created_at < '2025-12-11'))
```

```
CREATE INDEX idx_orders_created_at
    ON orders (created_at);

CREATE INDEX idx_order_lines_order_id
    ON order_lines (order_id)
```

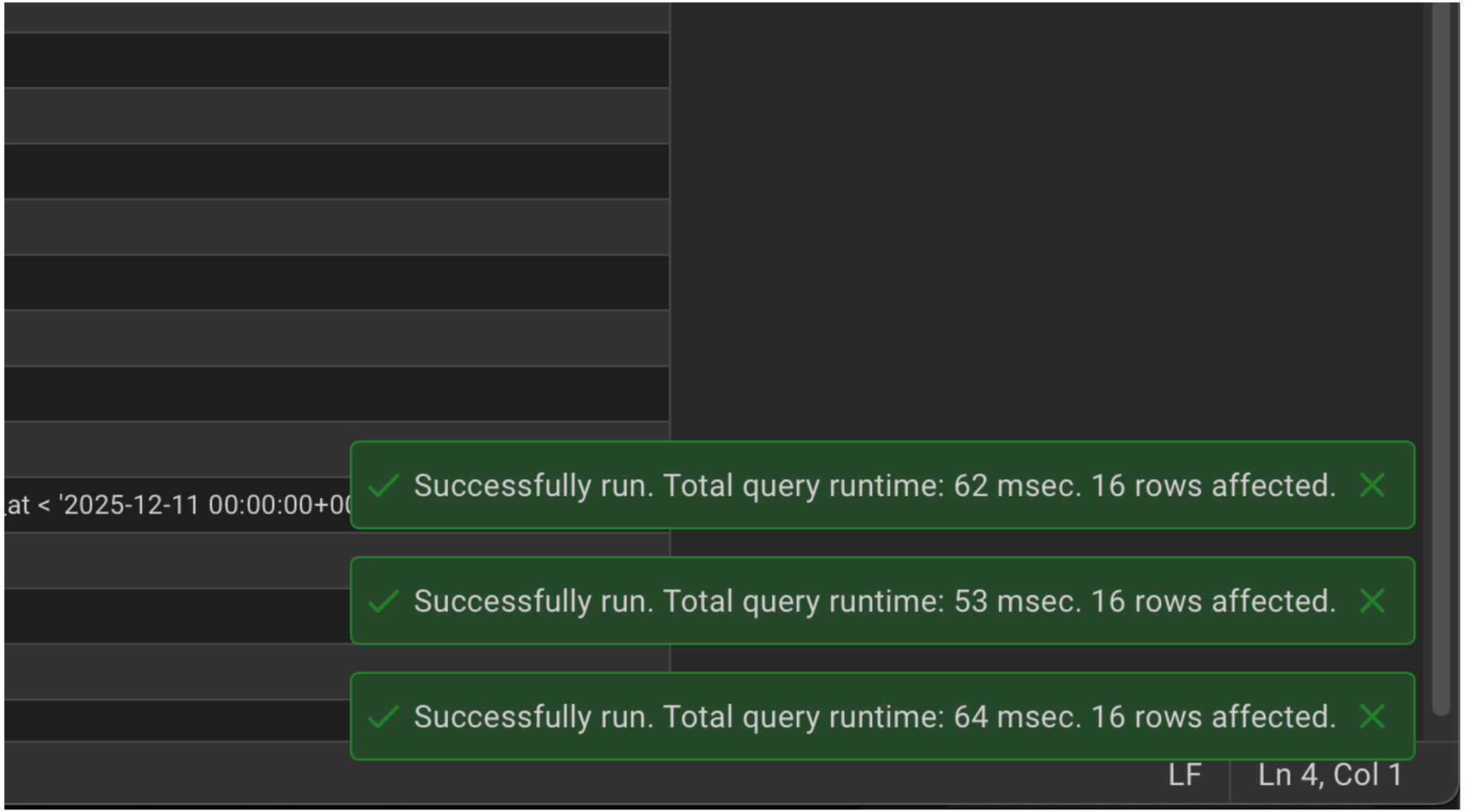
```
CREATE INDEX idx_orders_created_at
    ON orders (created_at);

CREATE INDEX idx_order_lines_order_id
    ON order_lines (order_id)
```

```
CREATE INDEX idx_orders_created_at
    ON orders (created_at);

CREATE INDEX idx_order_lines_order_id
    ON order_lines (order_id)
```

```
CREATE INDEX idx_orders_created_at
    ON orders (created_at);

CREATE INDEX idx_order_lines_order_id
    ON order_lines (order_id)
```

```
CREATE INDEX idx_orders_created_at
    ON orders (created_at);

CREATE INDEX idx_order_lines_order_id
    ON order_lines (order_id)
```

http://localhost:5109/dev-tools


at < '2025-12-11 00:00:00+0
Successfully run. Total query runtime: 62 msec. 16 rows affected.
Successfully run. Total query runtime: 53 msec. 16 rows affected.
Successfully run. Total query runtime: 64 msec. 16 rows affected.
LF
Ln 4, Col 1

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Sort
                    Sort Key: p.id, ol.order_id
                    ->  Hash Join
                          Hash Cond: (ol.product_id = p.id)
                          ->  Nested Loop
                                ->  Index Scan using idx_orders_created_at on orders o
                                      Index Cond: ((created_at >= '2025-12-09')
                                                AND (created_at < '2025-12-11'))
                                ->  Index Scan using idx_order_lines_order_id on order_lines ol
                                      Index Cond: (order_id = o.id)
                          ->  Hash
                                ->  Seq Scan on products p
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Sort
                    Sort Key: p.id, ol.order_id
                    ->  Hash Join
                          Hash Cond: (ol.product_id = p.id)
                          ->  Nested Loop
                                ->  Index Scan using idx_orders_created_at on orders o
                                      Index Cond: ((created_at >= '2025-12-09')
                                                AND (created_at < '2025-12-11'))
                                ->  Index Scan using idx_order_lines_order_id on order_lines ol
                                      Index Cond: (order_id = o.id)
                          ->  Hash
                                ->  Seq Scan on products p
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Sort
                    Sort Key: p.id, ol.order_id
                    ->  Hash Join
                          Hash Cond: (ol.product_id = p.id)
                          ->  Nested Loop
                                ->  Index Scan using idx_orders_created_at on orders o
                                      Index Cond: ((created_at >= '2025-12-09')
                                                AND (created_at < '2025-12-11'))
                                ->  Index Scan using idx_order_lines_order_id on order_lines ol
                                      Index Cond: (order_id = o.id)
                          ->  Hash
                                ->  Seq Scan on products p
```

```
SELECT
    P.ID AS PRODUCT_ID,
    P.NAME AS PRODUCT_NAME,
    P.SKU,
    SUM(OL.QUANTITY) AS TOTAL_QUANTITY,
    SUM(OL.QUANTITY * OL.UNIT_PRICE) AS TOTAL_REVENUE,
    COUNT(DISTINCT OL.ORDER_ID) AS ORDER_COUNT,
    AVG(OL.UNIT_PRICE) AS AVG_UNIT_PRICE
FROM
    PRODUCTS P
    INNER JOIN ORDER_LINES OL ON P.ID = OL.PRODUCT_ID
    INNER JOIN ORDERS O ON OL.ORDER_ID = O.ID
WHERE
    O.CREATED_AT >= '2025-12-09'
    AND O.CREATED_AT < '2025-12-11'
GROUP BY
    P.ID,
    P.NAME,
    P.SKU
```

```
SELECT
    P.ID AS PRODUCT_ID,
    P.NAME AS PRODUCT_NAME,
    P.SKU,
    SUM(OL.QUANTITY) AS TOTAL_QUANTITY,
    SUM(OL.QUANTITY * OL.UNIT_PRICE) AS TOTAL_REVENUE,
    COUNT(DISTINCT OL.ORDER_ID) AS ORDER_COUNT,
    AVG(OL.UNIT_PRICE) AS AVG_UNIT_PRICE
FROM
    PRODUCTS P
    INNER JOIN ORDER_LINES OL ON P.ID = OL.PRODUCT_ID
    INNER JOIN ORDERS O ON OL.ORDER_ID = O.ID
WHERE
    O.CREATED_AT >= '2025-12-09'
    AND O.CREATED_AT < '2025-12-11'
GROUP BY
    P.ID,
    P.NAME,
    P.SKU
```

```sql
SELECT
    P.ID AS PRODUCT_ID,
    P.NAME AS PRODUCT_NAME,
    P.SKU,
    SUM(OL.QUANTITY) AS TOTAL_QUANTITY,
    SUM(OL.QUANTITY * OL.UNIT_PRICE) AS TOTAL_REVENUE,
    COUNT(DISTINCT OL.ORDER_ID) AS ORDER_COUNT,
    AVG(OL.UNIT_PRICE) AS AVG_UNIT_PRICE
FROM
    PRODUCTS P
    INNER JOIN ORDER_LINES OL ON P.ID = OL.PRODUCT_ID
    INNER JOIN ORDERS O ON OL.ORDER_ID = O.ID
WHERE
    O.CREATED_AT >= '2025-12-09'
    AND O.CREATED_AT < '2025-12-11'
GROUP BY
    P.ID,
    P.NAME,
    P.SKU
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Sort
                    Sort Key: p.id, ol.order_id
                    ->  Hash Join
                          Hash Cond: (ol.product_id = p.id)
                          ->  Nested Loop
                                ->  Index Scan using idx_orders_created_at on orders o
                                      Index Cond: ((created_at >= '2025-12-09')
                                                AND (created_at < '2025-12-11'))
                                ->  Index Scan using idx_order_lines_order_id on order_lines ol
                                      Index Cond: (order_id = o.id)
                          ->  Hash
                                ->  Seq Scan on products p
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Sort
                    Sort Key: p.id, ol.order_id
                    ->  Hash Join
                          Hash Cond: (ol.product_id = p.id)
                          ->  Nested Loop
                                ->  Index Scan using idx_orders_created_at on orders o
                                      Index Cond: ((created_at >= '2025-12-09')
                                                AND (created_at < '2025-12-11'))
                                ->  Index Scan using idx_order_lines_order_id on order_lines ol
                                      Index Cond: (order_id = o.id)
                          ->  Hash
                                ->  Seq Scan on products p
```

```
Limit
  ->  Sort
        Sort Key: (sum(((ol.quantity)::numeric * ol.unit_price))) DESC
        ->  GroupAggregate
              Group Key: p.id
              ->  Sort
                    Sort Key: p.id, ol.order_id
                    ->  Hash Join
                          Hash Cond: (ol.product_id = p.id)
                          ->  Nested Loop
                                ->  Index Scan using idx_orders_created_at on orders o
                                      Index Cond: ((created_at >= '2025-12-09')
                                                AND (created_at < '2025-12-11'))
                                ->  Index Scan using idx_order_lines_order_id on order_lines ol
                                      Index Cond: (order_id = o.id)
                          ->  Hash
                                ->  Seq Scan on products p
```

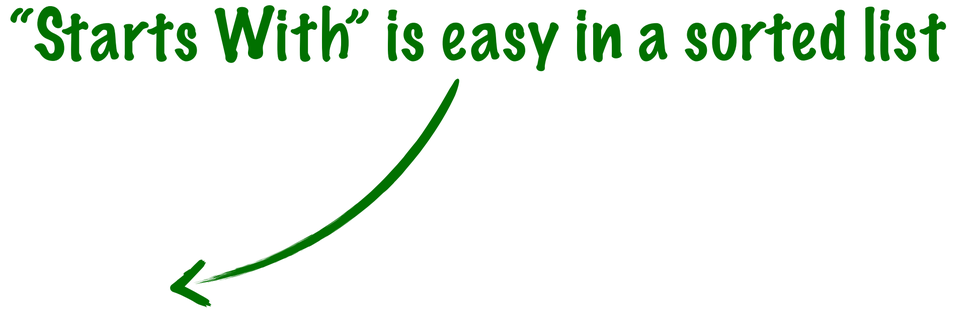
Index Might Not Be Chosen

Some operations are not indexable

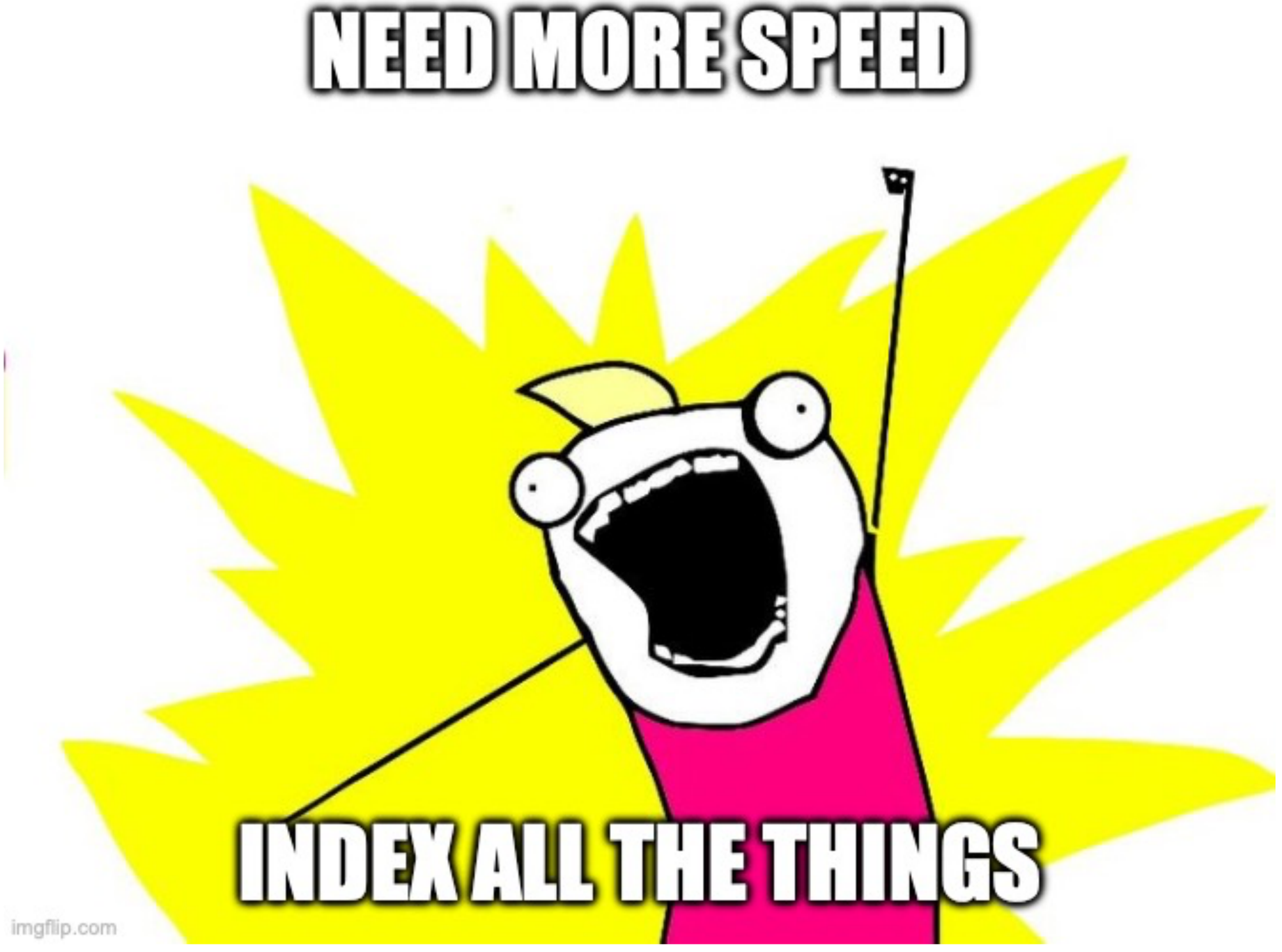
```
CREATE INDEX idx_customers_name
  ON customers(“name”);

SELECT
  *
FROM customers
WHERE name ILIKE ‘Matt%’
```

```
CREATE INDEX idx_customers_name
  ON customers(“name”);

SELECT
  *
FROM customers
WHERE name ILIKE ‘Matt%’
```

```
CREATE INDEX idx_customers_name
  ON customers("name");

SELECT
  *
FROM customers
WHERE name ILIKE 'Matt%'
```



```
CREATE INDEX idx_customers_name
  ON customers("name");

SELECT
  *
FROM customers
WHERE name ILIKE '%Burke'
```


NEED MORE SPEED
INDEX ALL THE THINGS
imgflip.com

There *is* such a thing as over-indexing

QUERY TEXT

\+ EXPLAIN OUTPUT

\+ $10B+ LLM TRAINING

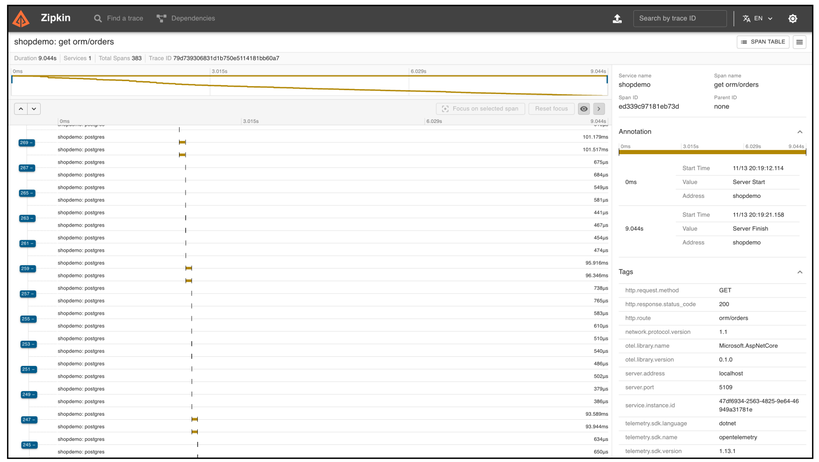
PRETTY DECENT SUGGESTIONS

Use The Index, Luke!

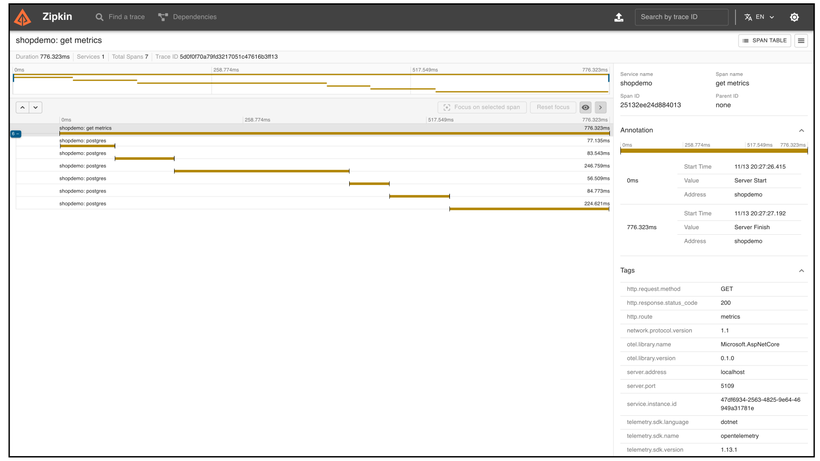
Free e-book on database index tuning by Markus Winand

Caffeinate Your Queries: Brew Up Faster SQL with Tuning

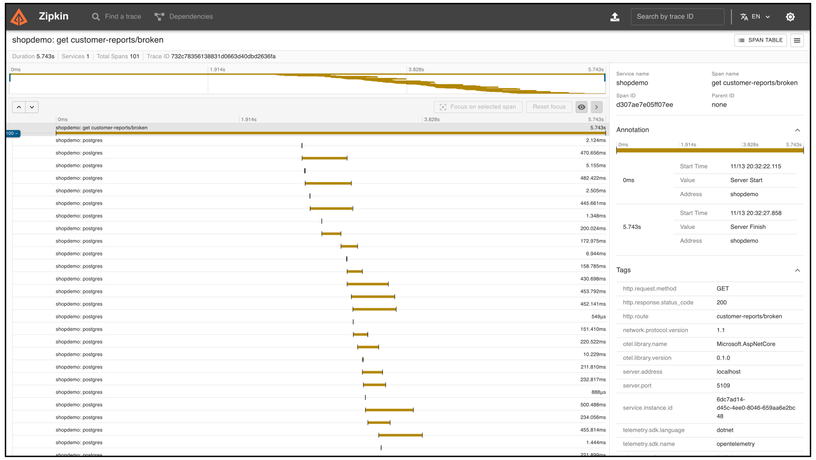
Tristan Chiappisi, Indigo Bay @ 8:30



Many Small Spans



Large Spans



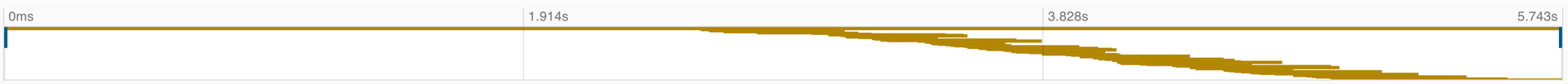
Span Gaps

Zipkin | Find a trace | Dependencies | Search by trace ID | EN

shopdemo: get customer-reports/broken

SPAN TABLE

Duration 5.743s | Services 1 | Total Spans 101 | Trace ID 732c78356138831d0663d40dbd2636fa



Focus on selected span | Reset focus

shopdemo: get customer-reports/broken	0ms – 5.743s	5.743s
shopdemo: postgres	0.000ms – 2.124ms	2.124ms
shopdemo: postgres	0.000ms – 470.656ms	470.656ms
shopdemo: postgres	0.000ms – 5.155ms	5.155ms
shopdemo: postgres	0.000ms – 482.422ms	482.422ms
shopdemo: postgres	0.000ms – 2.505ms	2.505ms
shopdemo: postgres	0.000ms – 445.661ms	445.661ms
shopdemo: postgres	0.000ms – 1.348ms	1.348ms
shopdemo: postgres	0.000ms – 200.024ms	200.024ms
shopdemo: postgres	0.000ms – 172.975ms	172.975ms
shopdemo: postgres	0.000ms – 6.944ms	6.944ms
shopdemo: postgres	0.000ms – 158.785ms	158.785ms
shopdemo: postgres	0.000ms – 430.698ms	430.698ms
shopdemo: postgres	0.000ms – 453.792ms	453.792ms
shopdemo: postgres	0.000ms – 452.141ms	452.141ms
shopdemo: postgres	0.000ms – 549µs	549µs
shopdemo: postgres	0.000ms – 151.410ms	151.410ms
shopdemo: postgres	0.000ms – 220.522ms	220.522ms
shopdemo: postgres	0.000ms – 10.229ms	10.229ms
shopdemo: postgres	0.000ms – 211.810ms	211.810ms
shopdemo: postgres	0.000ms – 232.817ms	232.817ms
shopdemo: postgres	0.000ms – 888µs	888µs
shopdemo: postgres	0.000ms – 500.488ms	500.488ms
shopdemo: postgres	0.000ms – 234.056ms	234.056ms
shopdemo: postgres	0.000ms – 455.814ms	455.814ms
shopdemo: postgres	0.000ms – 1.444ms	1.444ms
shopdemo: postgres	0.000ms – 221.899ms	221.899ms

Service name: shopdemo
Span name: get customer-reports/broken
Span ID: d307ae7e05ff07ee
Parent ID: none

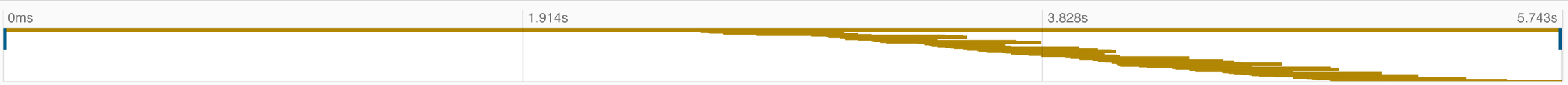
Annotation

0ms	Start Time	11/13 20:32:22.115
	Value	Server Start
	Address	shopdemo
5.743s	Start Time	11/13 20:32:27.858
	Value	Server Finish
	Address	shopdemo

Tags

http.request.method	GET
http.response.status_code	200
http.route	customer-reports/broken
network.protocol.version	1.1
otel.library.name	Microsoft.AspNetCore
otel.library.version	0.1.0
server.address	localhost
server.port	5109
service.instance.id	6dc7ad14-d45c-4ee0-8046-659aa6e2bc48
telemetry.sdk.language	dotnet
telemetry.sdk.name	opentelemetry

Duration 5.743s | Services 1 | Total Spans 101 | Trace ID 732c78356138831d0663d40dbd2636fa

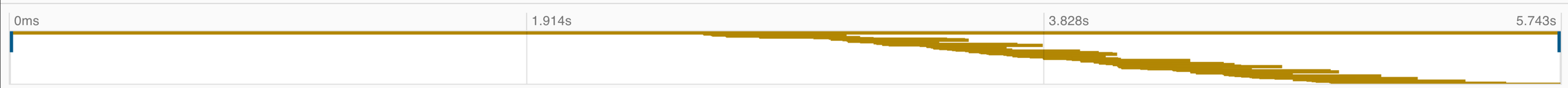


Focus on selected span | Reset focus

shopdemo: get customer-reports/broken	5.743s
shopdemo: postgres	2.124ms
shopdemo: postgres	470.656ms
shopdemo: postgres	5.155ms
shopdemo: postgres	482.422ms
shopdemo: postgres	2.505ms
shopdemo: postgres	445.661ms
shopdemo: postgres	1.348ms
shopdemo: postgres	200.024ms
shopdemo: postgres	172.975ms
shopdemo: postgres	6.944ms
shopdemo: postgres	158.785ms
shopdemo: postgres	430.698ms
shopdemo: postgres	453.792ms
shopdemo: postgres	452.141ms
shopdemo: postgres	549µs
shopdemo: postgres	151.410ms
shopdemo: postgres	220.522ms
shopdemo: postgres	10.229ms
shopdemo: postgres	211.810ms
shopdemo: postgres	232.817ms
shopdemo: postgres	888µs
shopdemo: postgres	500.488ms

Service
shopde
Span ID
d307ae
Annota
0ms
5.743
Tags
http.
http.
http.
netwo
otel.li
otel.li
serve
serve

Duration 5.743s | Services 1 | Total Spans 101 | Trace ID 732c78356138831d0663d40dbd2636fa

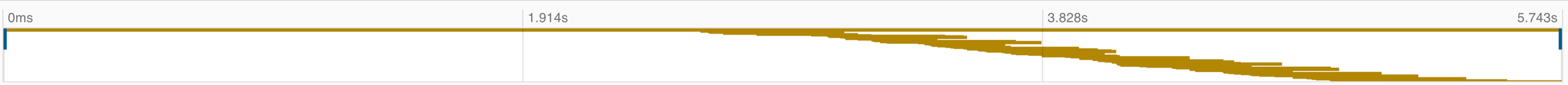


Focus on selected span | Reset focus

shopdemo: get customer-reports/broken	5.743s
shopdemo: postgres	2.124ms
shopdemo: postgres	470.656ms
shopdemo: postgres	5.155ms
shopdemo: postgres	482.422ms
shopdemo: postgres	2.505ms
shopdemo: postgres	445.661ms
shopdemo: postgres	1.348ms
shopdemo: postgres	200.024ms
shopdemo: postgres	172.975ms
shopdemo: postgres	6.944ms
shopdemo: postgres	158.785ms
shopdemo: postgres	430.698ms
shopdemo: postgres	453.792ms
shopdemo: postgres	452.141ms
shopdemo: postgres	549µs
shopdemo: postgres	151.410ms
shopdemo: postgres	220.522ms
shopdemo: postgres	10.229ms
shopdemo: postgres	211.810ms
shopdemo: postgres	232.817ms
shopdemo: postgres	888µs
shopdemo: postgres	500.488ms

Service
shopde
Span ID
d307ae
Annota
0ms
5.743
Tags
http.
http.
http.
netwo
otel.li
otel.li
serve
serve

Duration 5.743s | Services 1 | Total Spans 101 | Trace ID 732c78356138831d0663d40dbd2636fa



Focus on selected span | Reset focus

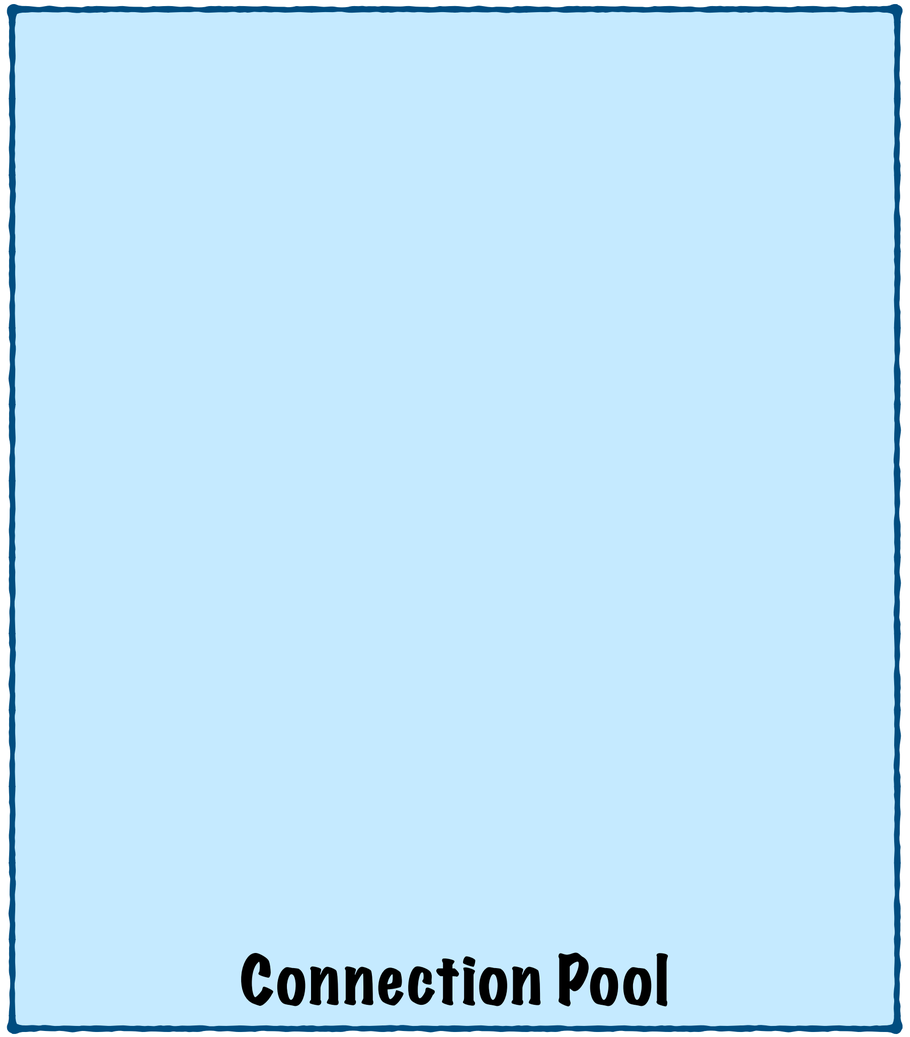
shopdemo: get customer-reports/broken	5.743s
shopdemo: postgres	2.124ms
shopdemo: postgres	470.656ms
shopdemo: postgres	5.155ms
shopdemo: postgres	482.422ms
shopdemo: postgres	2.505ms
shopdemo: postgres	445.661ms
shopdemo: postgres	1.348ms
shopdemo: postgres	200.024ms
shopdemo: postgres	172.975ms
shopdemo: postgres	6.944ms
shopdemo: postgres	158.785ms
shopdemo: postgres	430.698ms
shopdemo: postgres	453.792ms
shopdemo: postgres	452.141ms
shopdemo: postgres	549µs
shopdemo: postgres	151.410ms
shopdemo: postgres	220.522ms
shopdemo: postgres	10.229ms
shopdemo: postgres	211.810ms
shopdemo: postgres	232.817ms
shopdemo: postgres	888µs
shopdemo: postgres	500.488ms

Nothing happens for 2.4 seconds

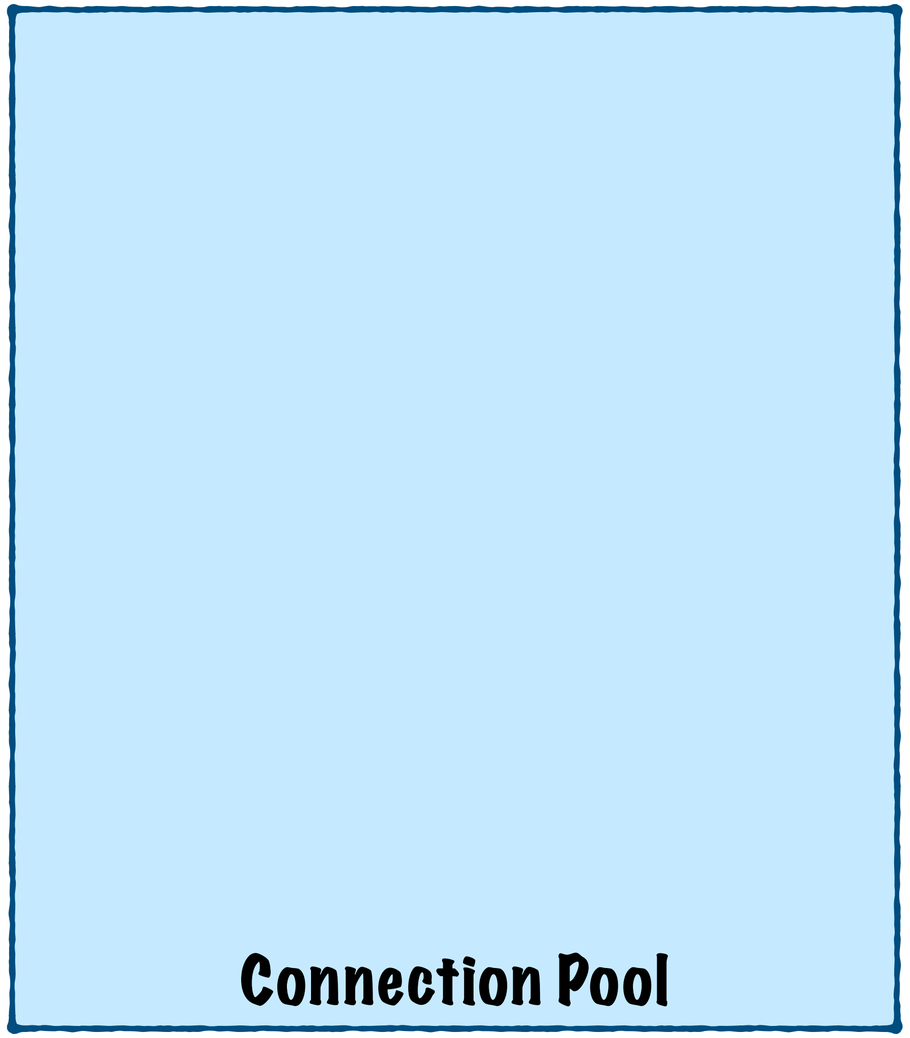
http://localhost:5109/customer-reports/broken?count=5

The Connection Pool

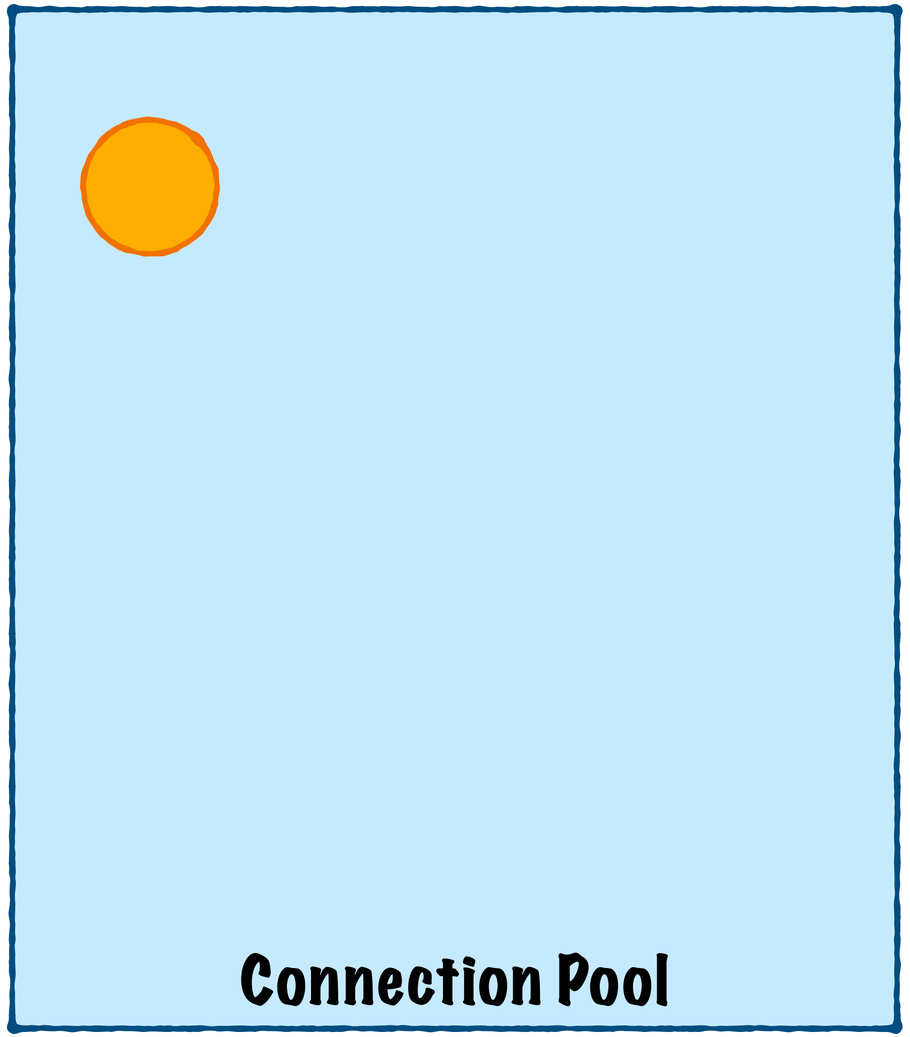
Connections are expensive

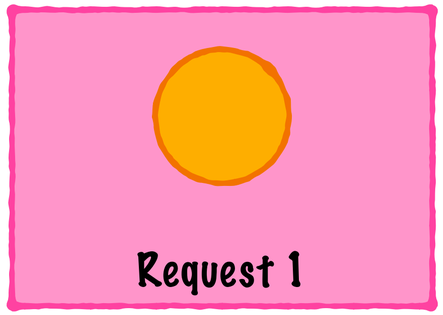

Connection Pool

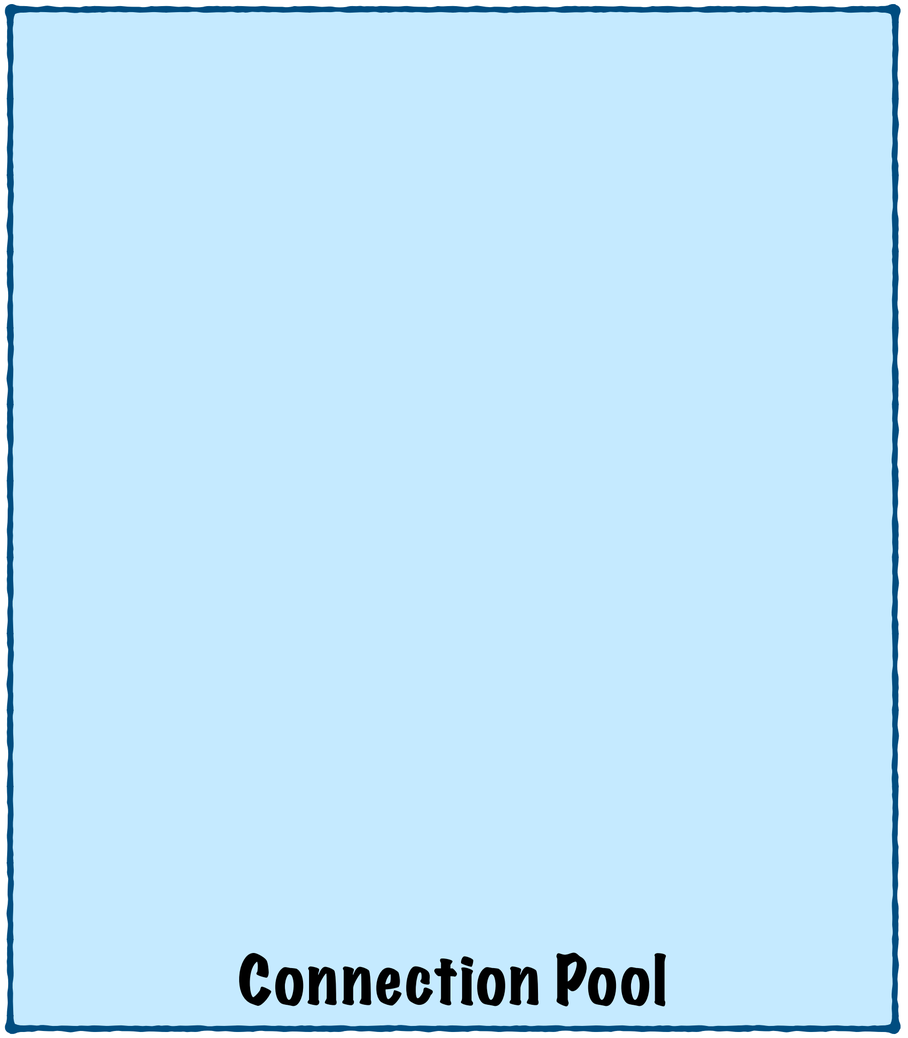

Request 1


Connection Pool

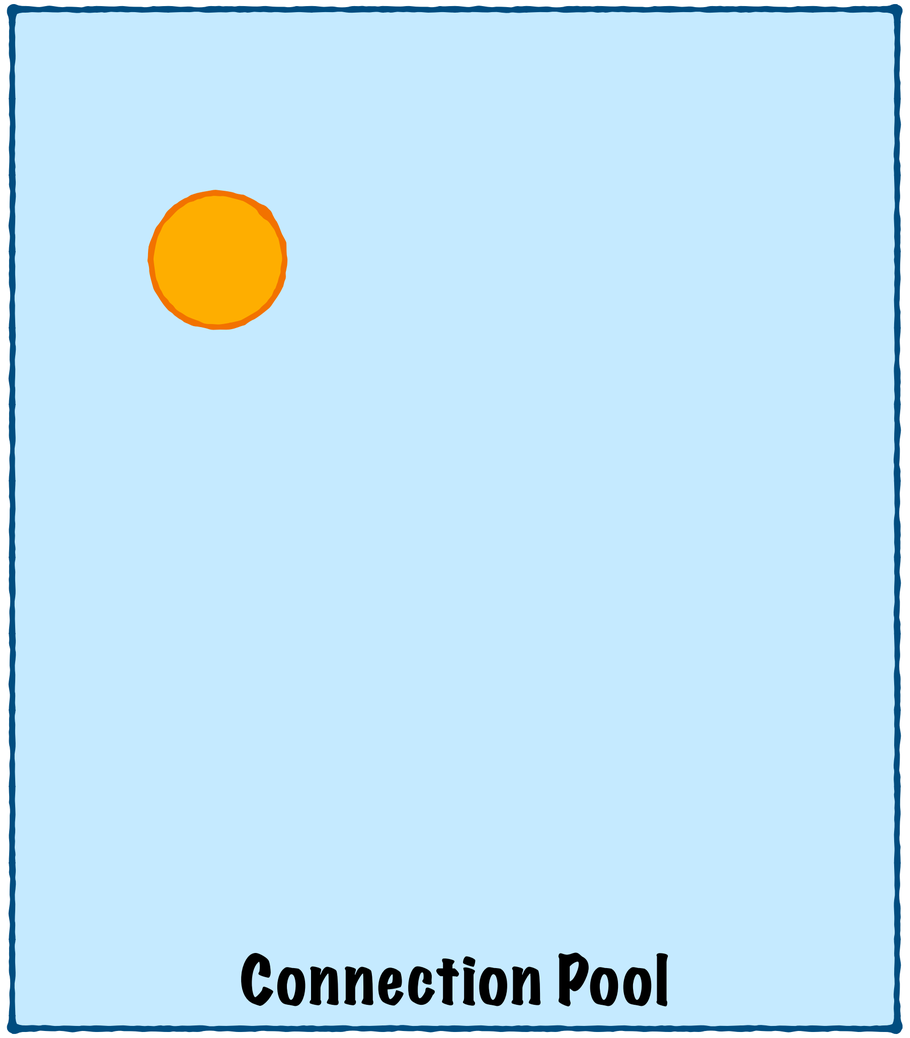
Request 1

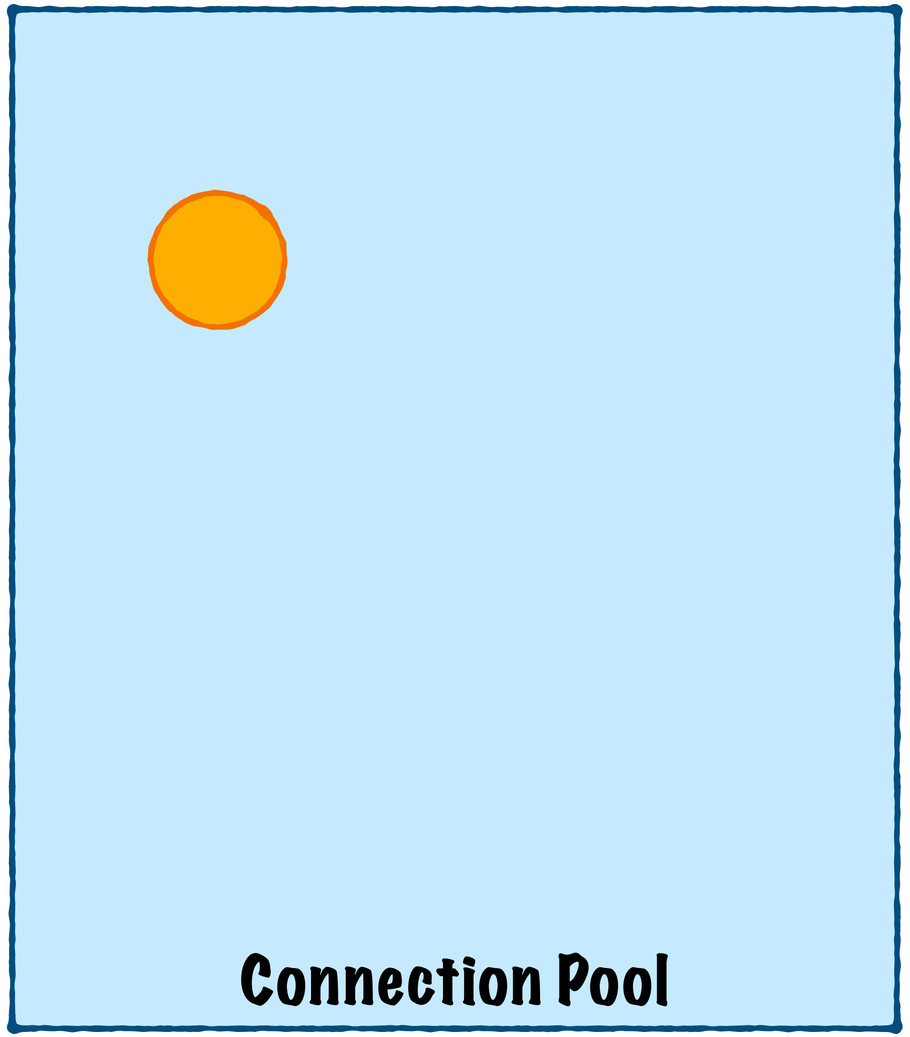



Request 1

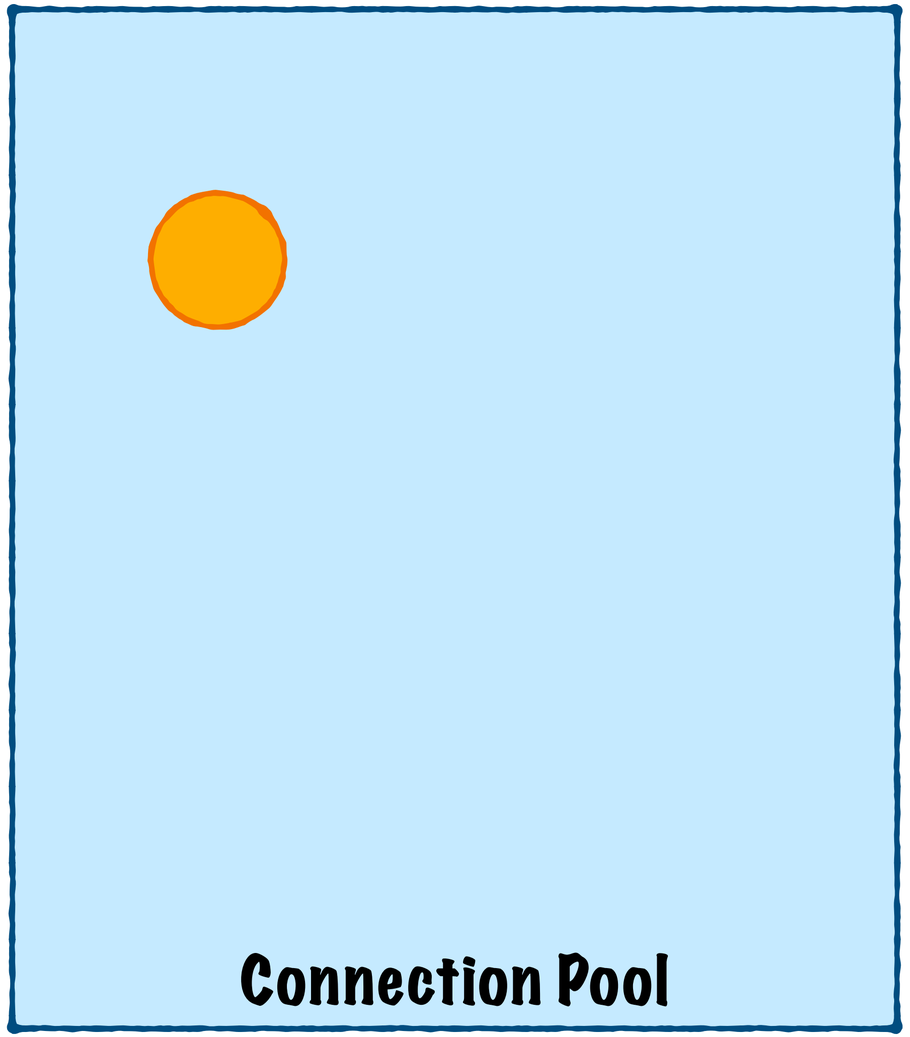

Connection Pool

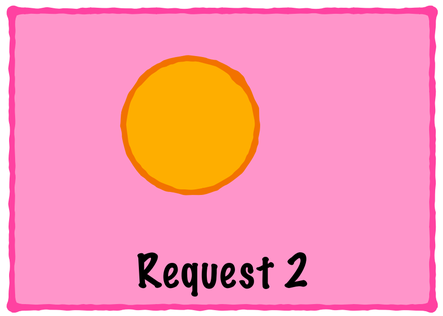

Request 1

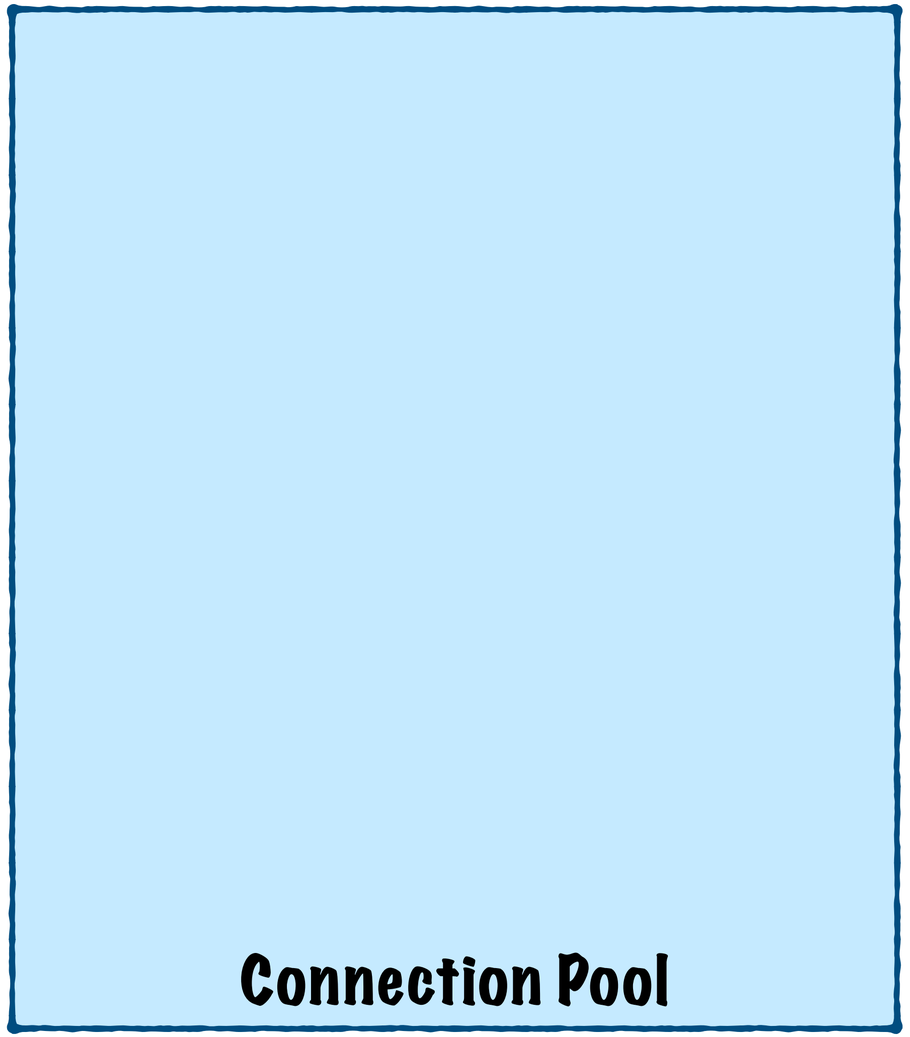

Connection Pool


Connection Pool

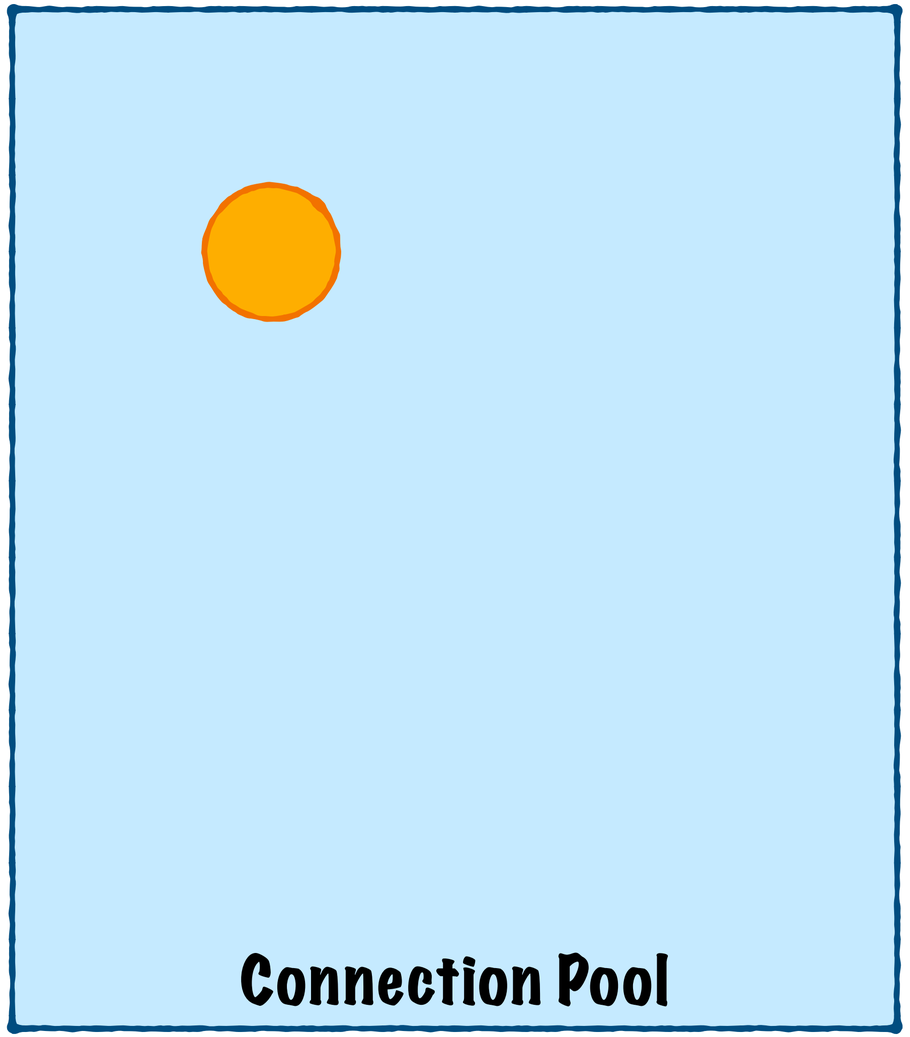

Request 2

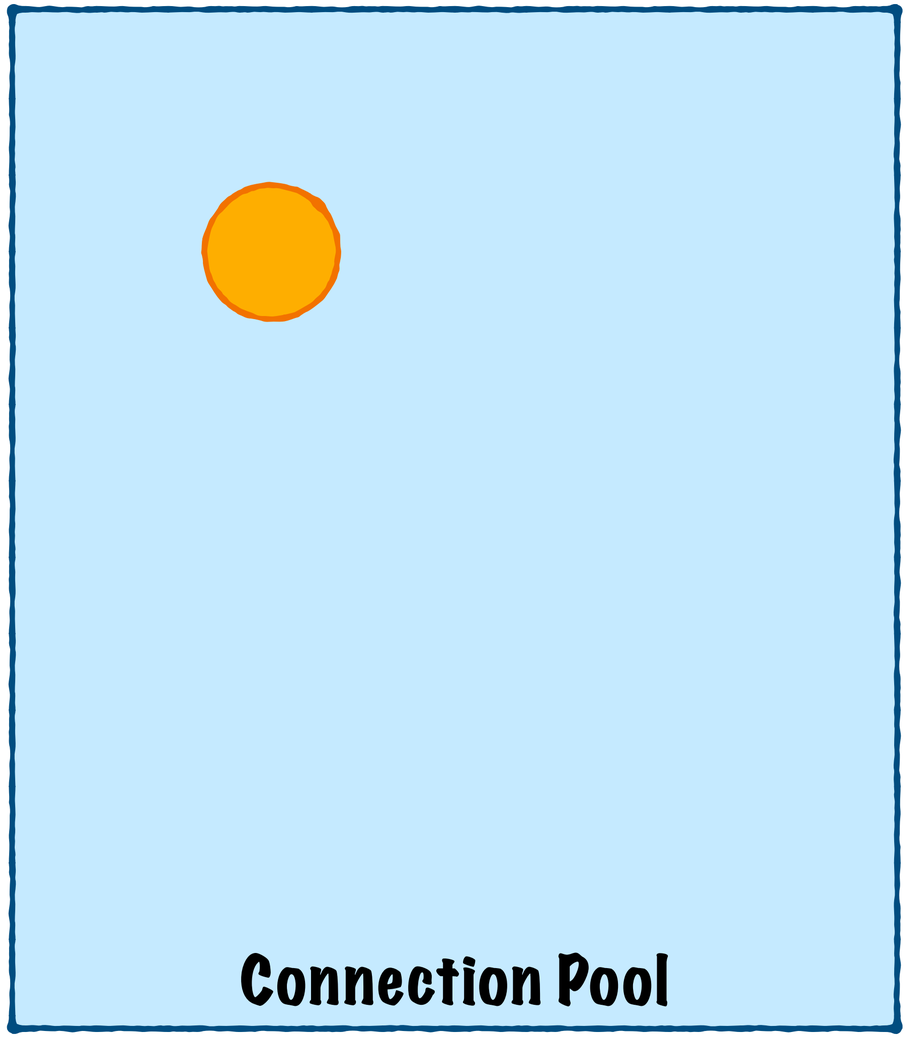

Connection Pool

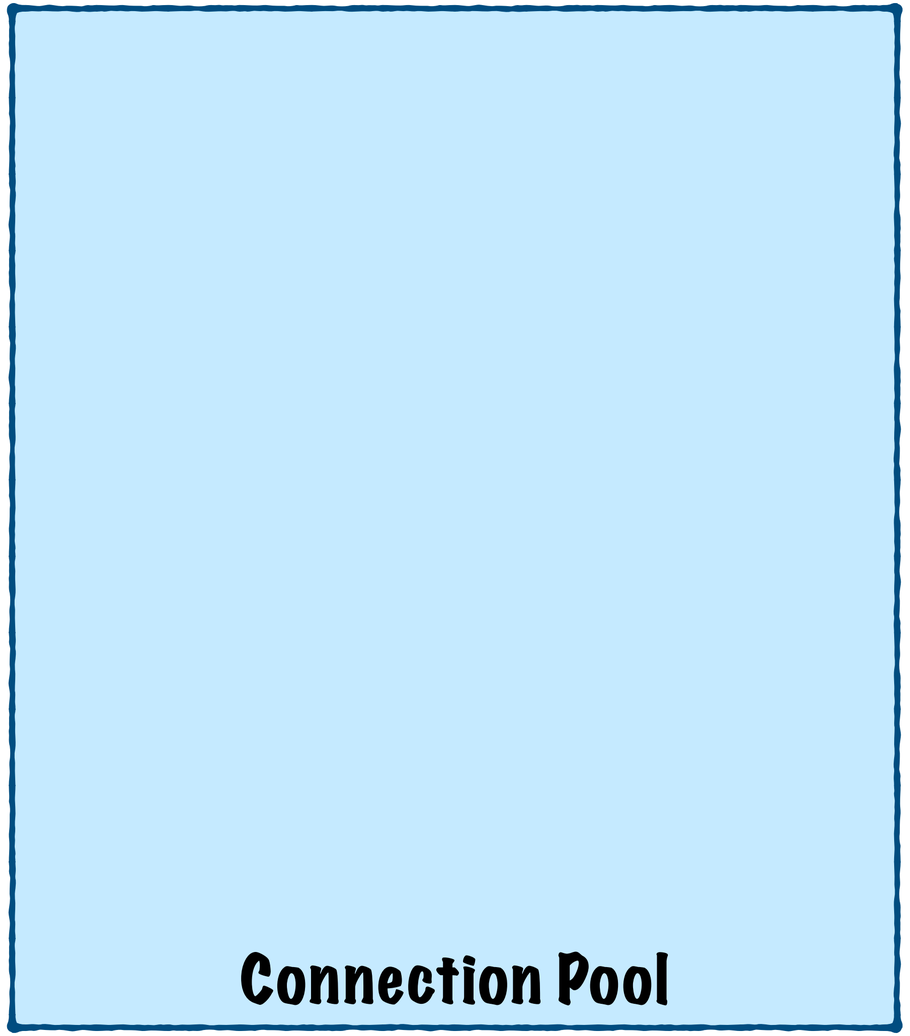

Request 2


Connection Pool

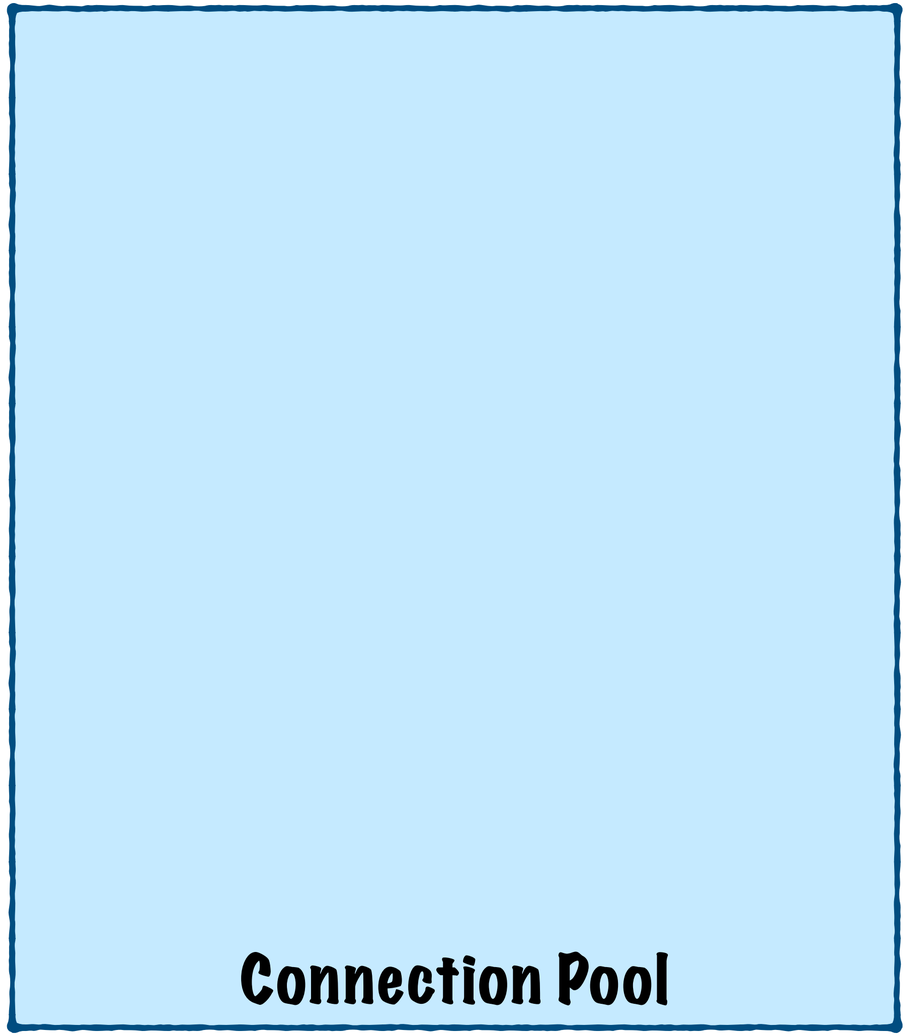

Request 2

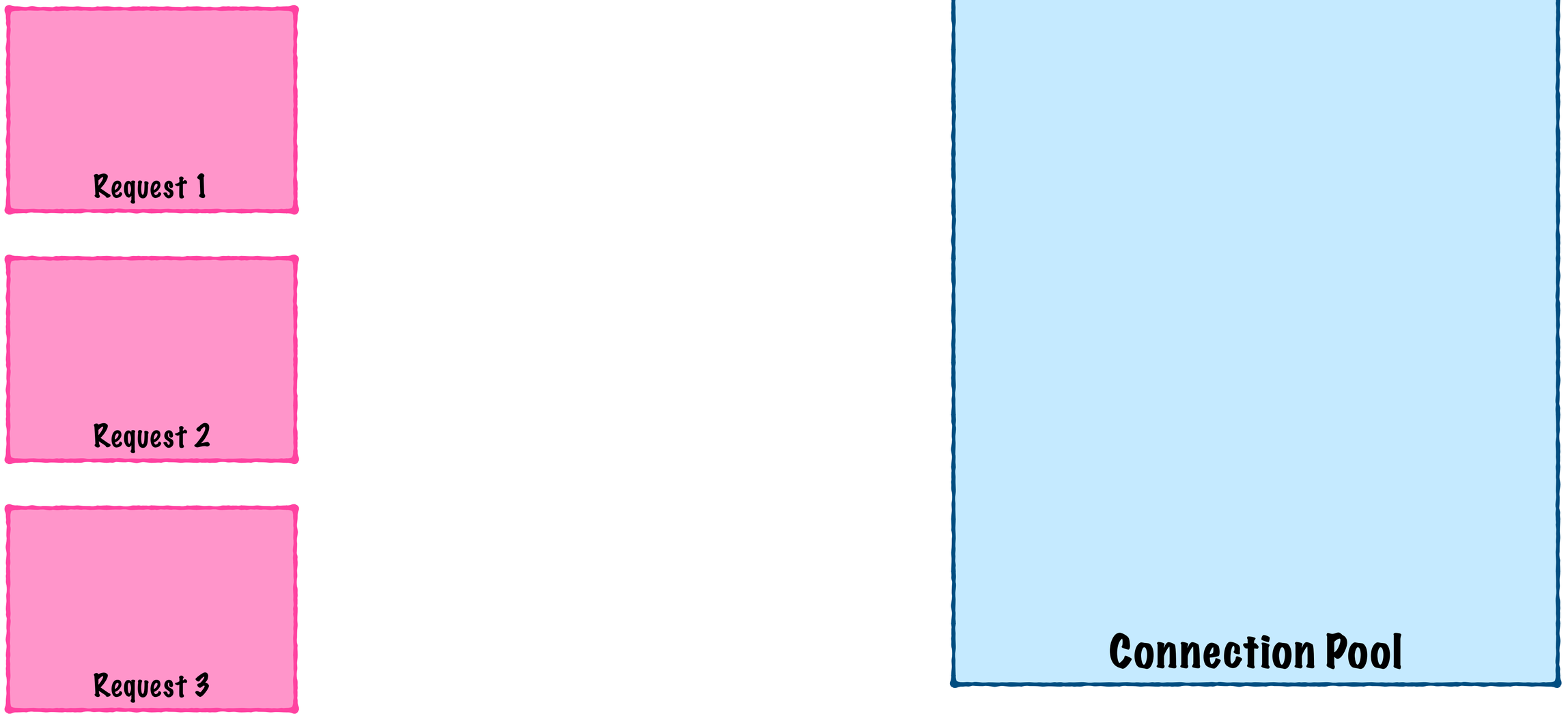

Connection Pool

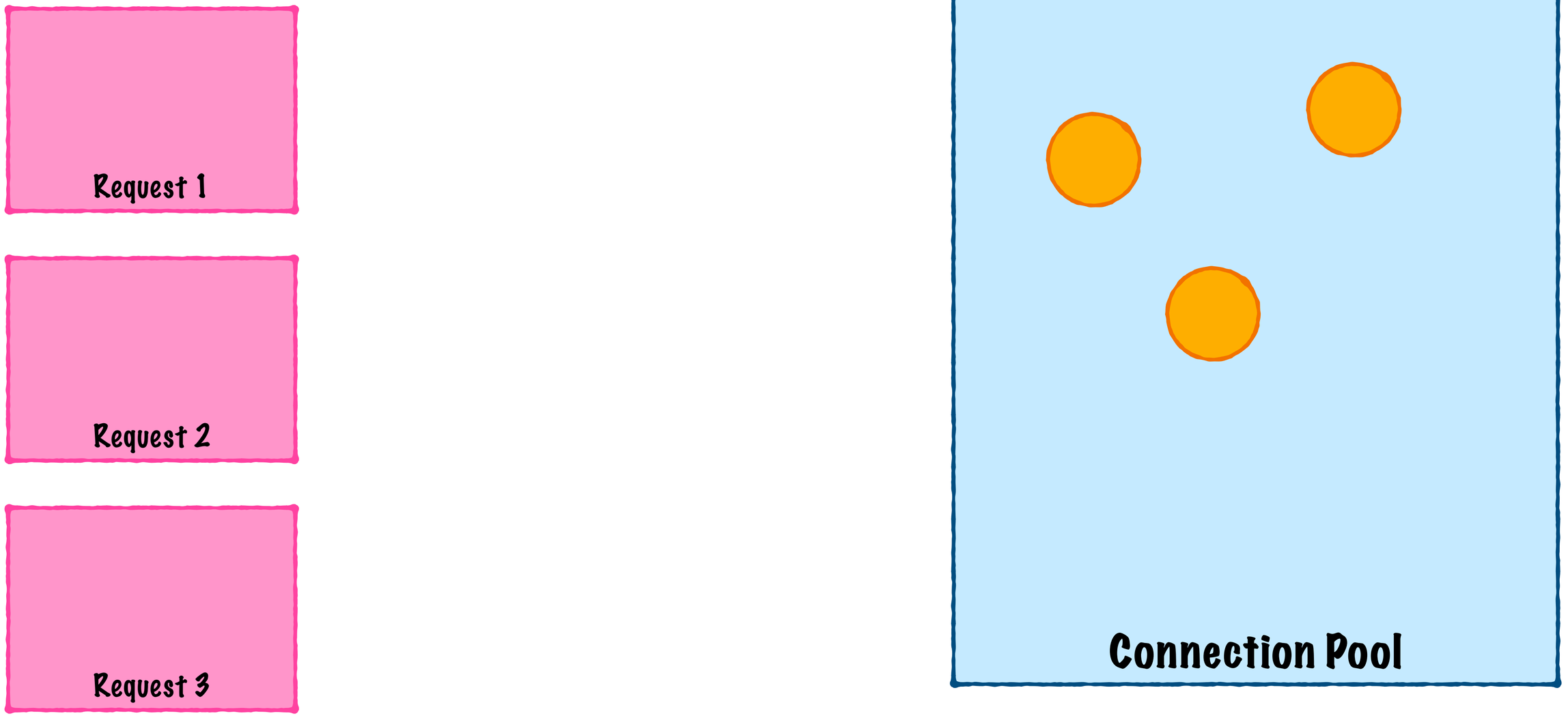

Connection Pool

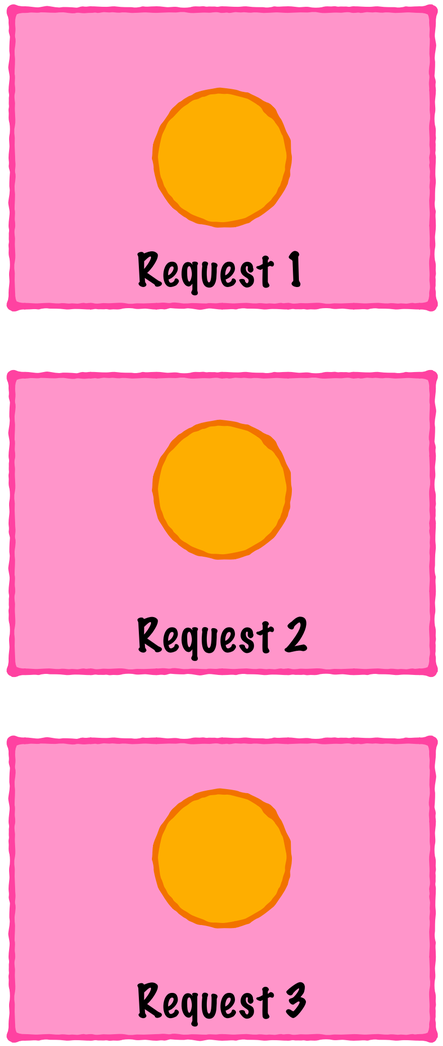

Connection Pool

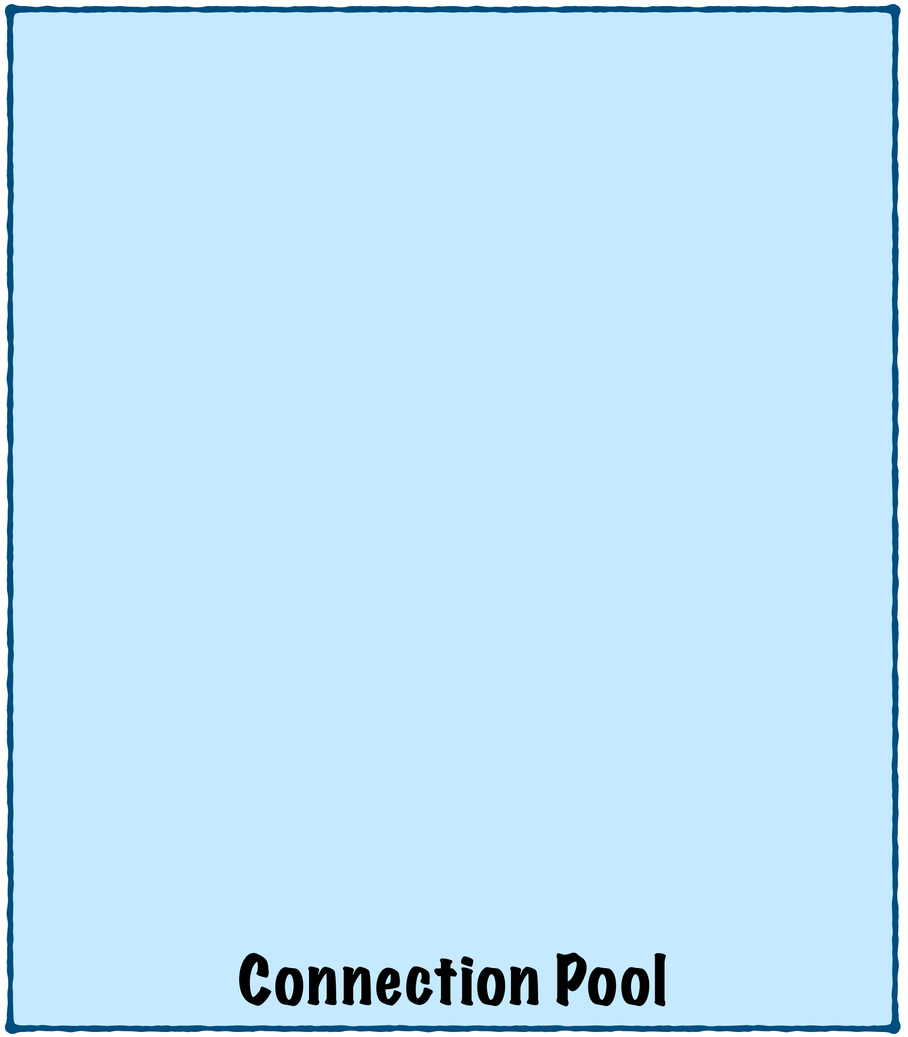
Scenario: Many Concurrent Requests

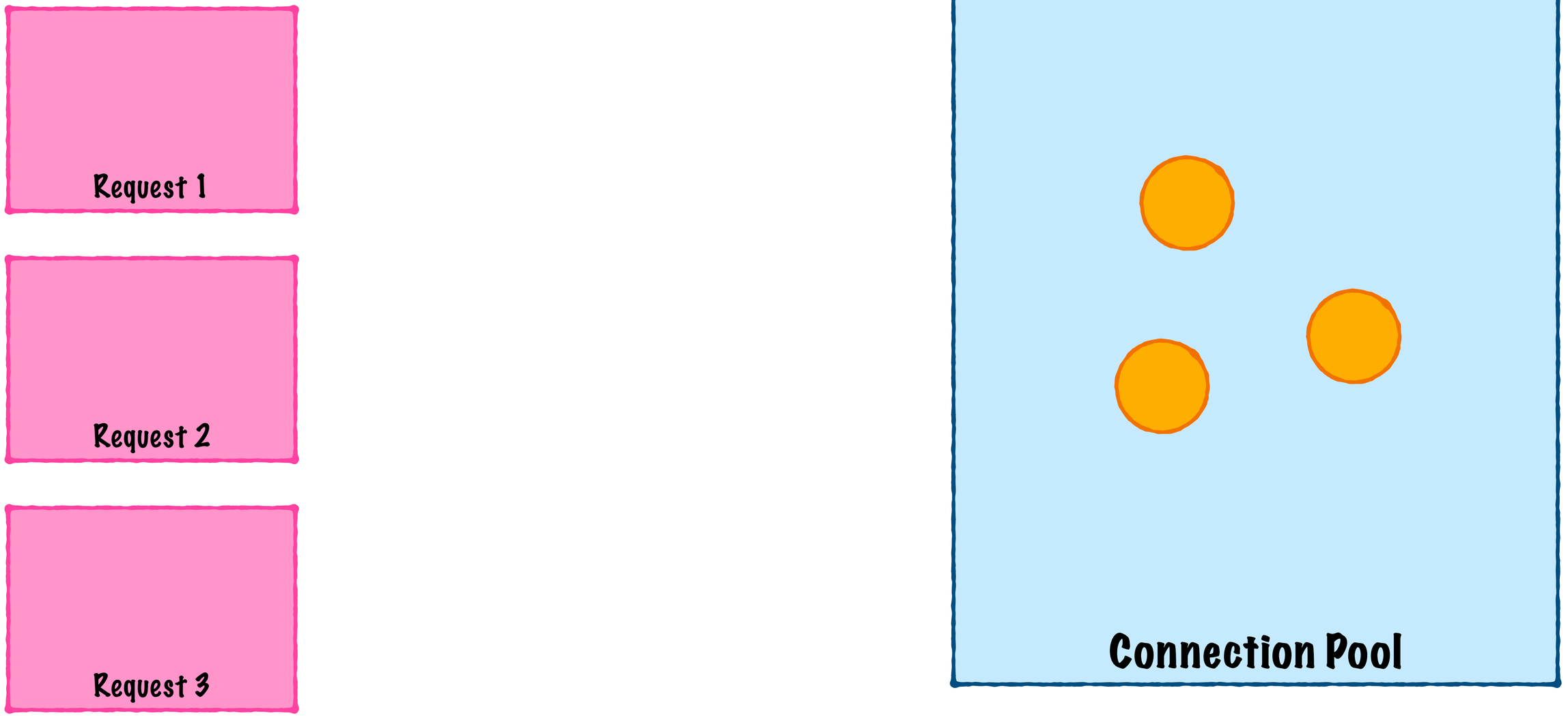

Connection Pool

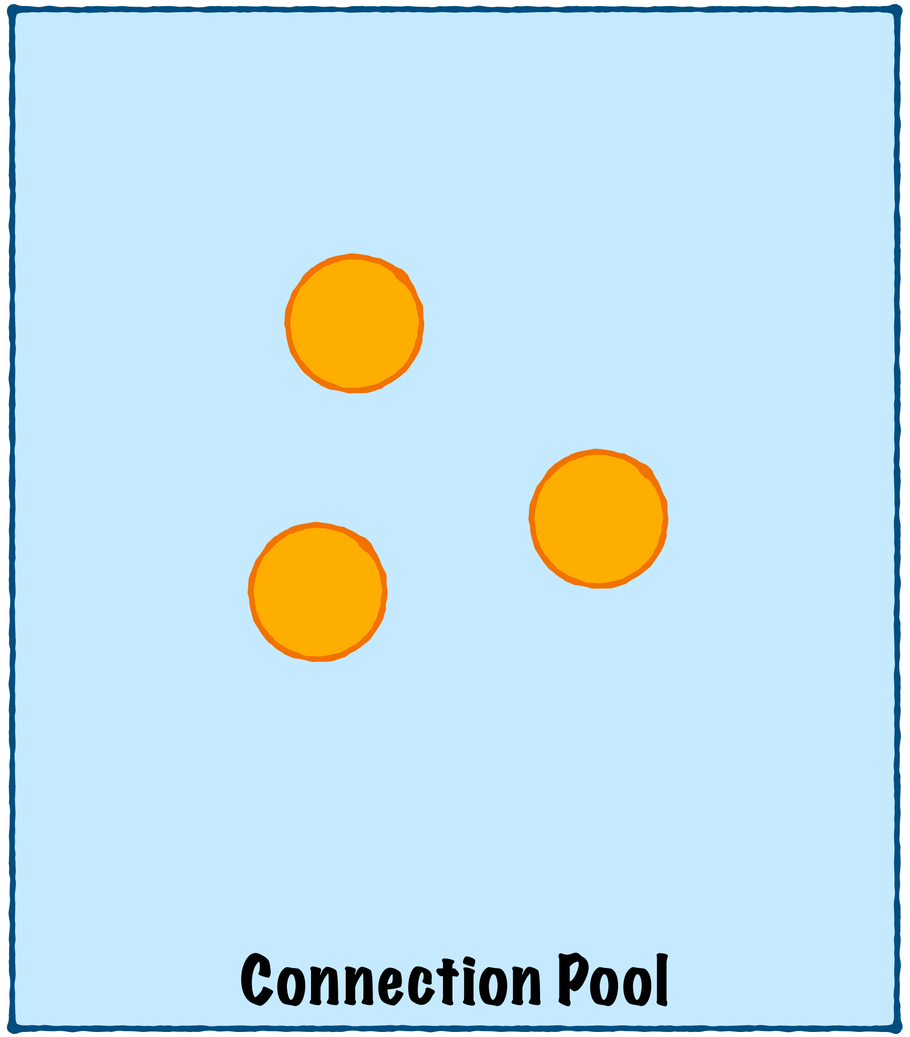

Request 1
Request 2
Request 3
Connection Pool

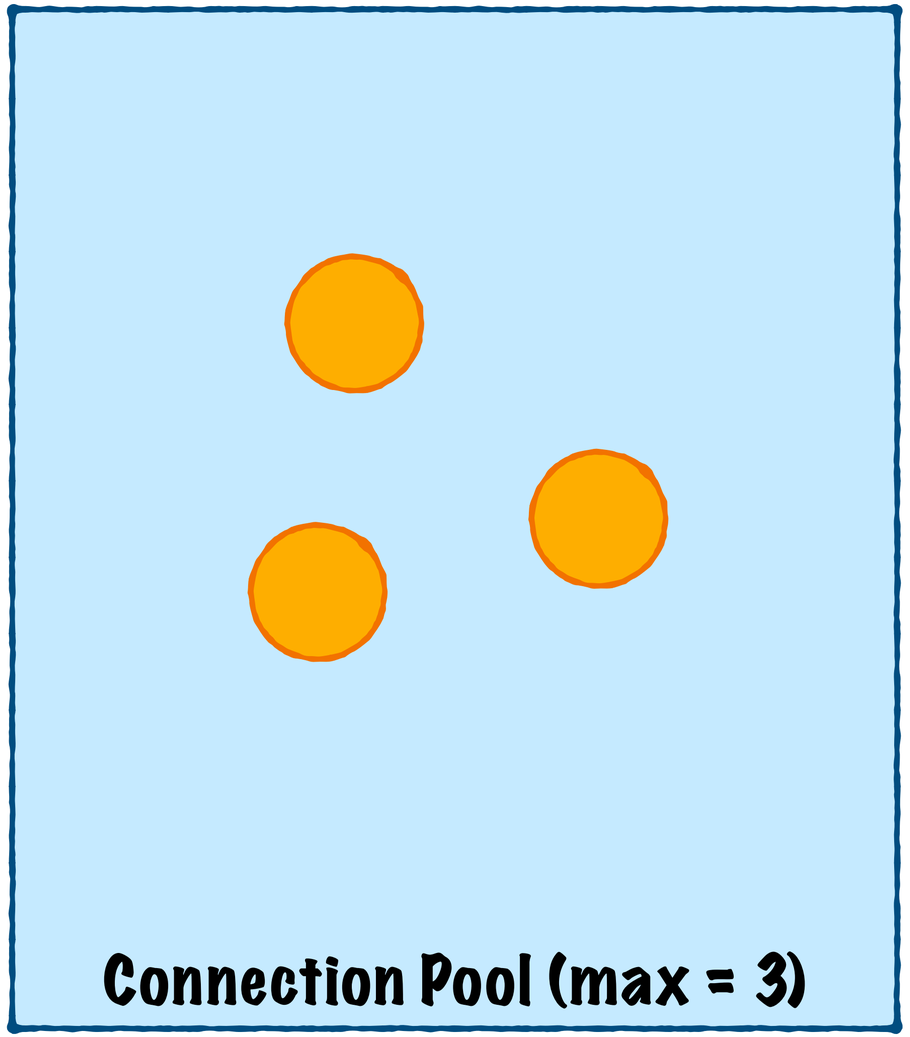

Request 1
Request 2
Request 3
Connection Pool


Request 1
Request 2
Request 3

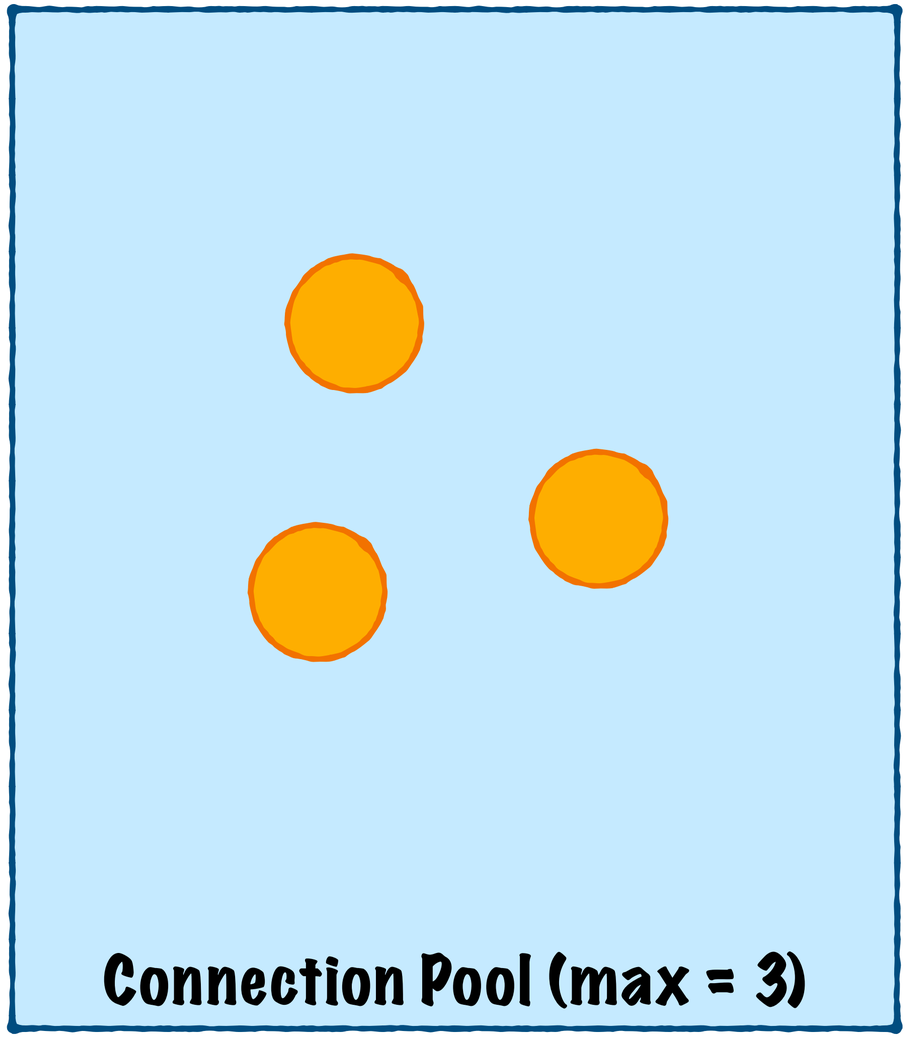

Connection Pool

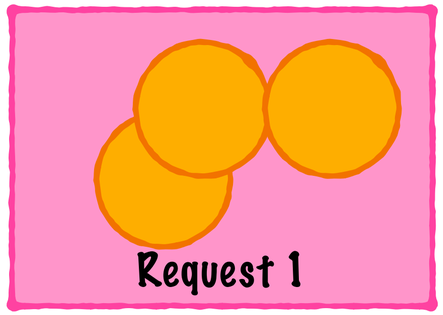

Request 1
Request 2
Request 3
Connection Pool


Connection Pool

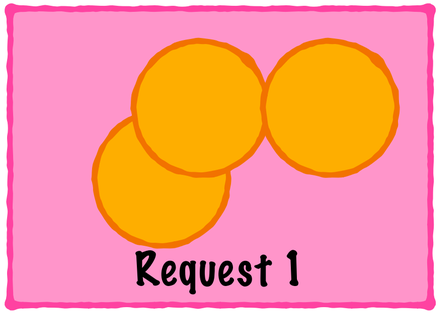

Connection Pool (max = 3)

Request 1

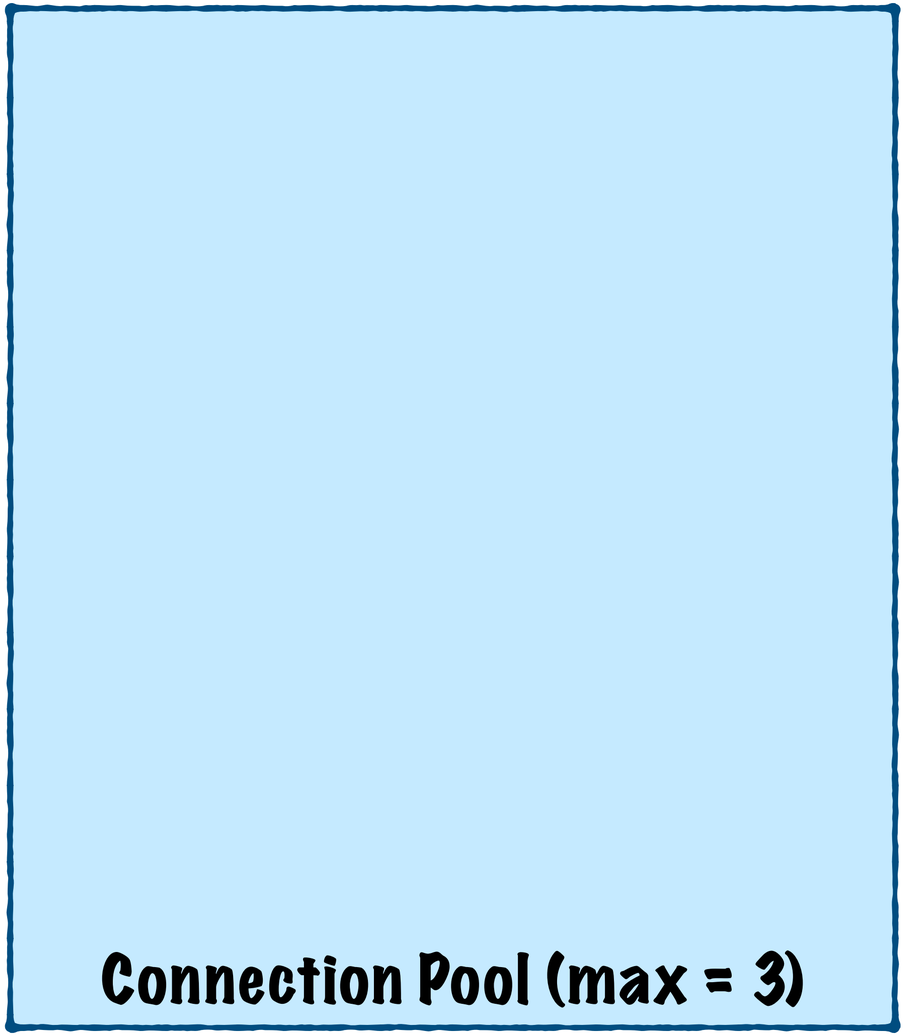


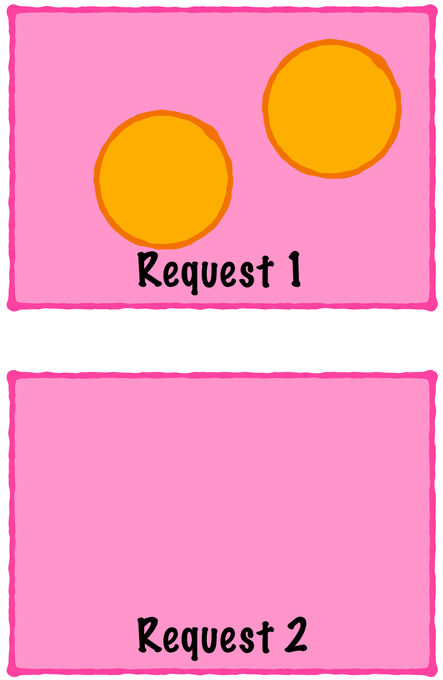


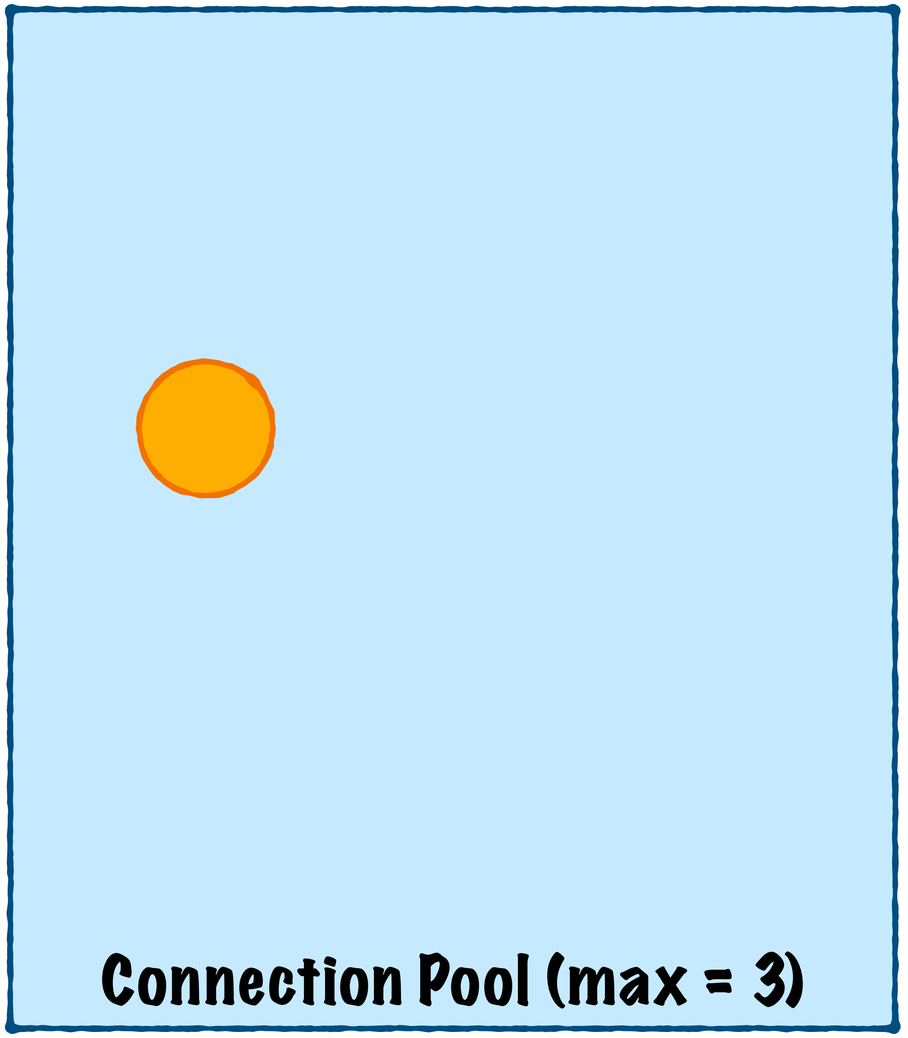
Connection Pool (max = 3)

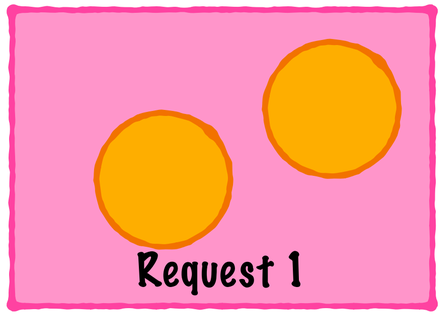

Request 1

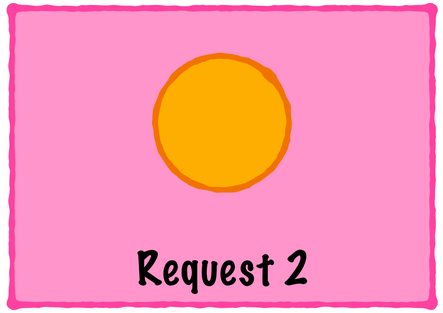

Request 2

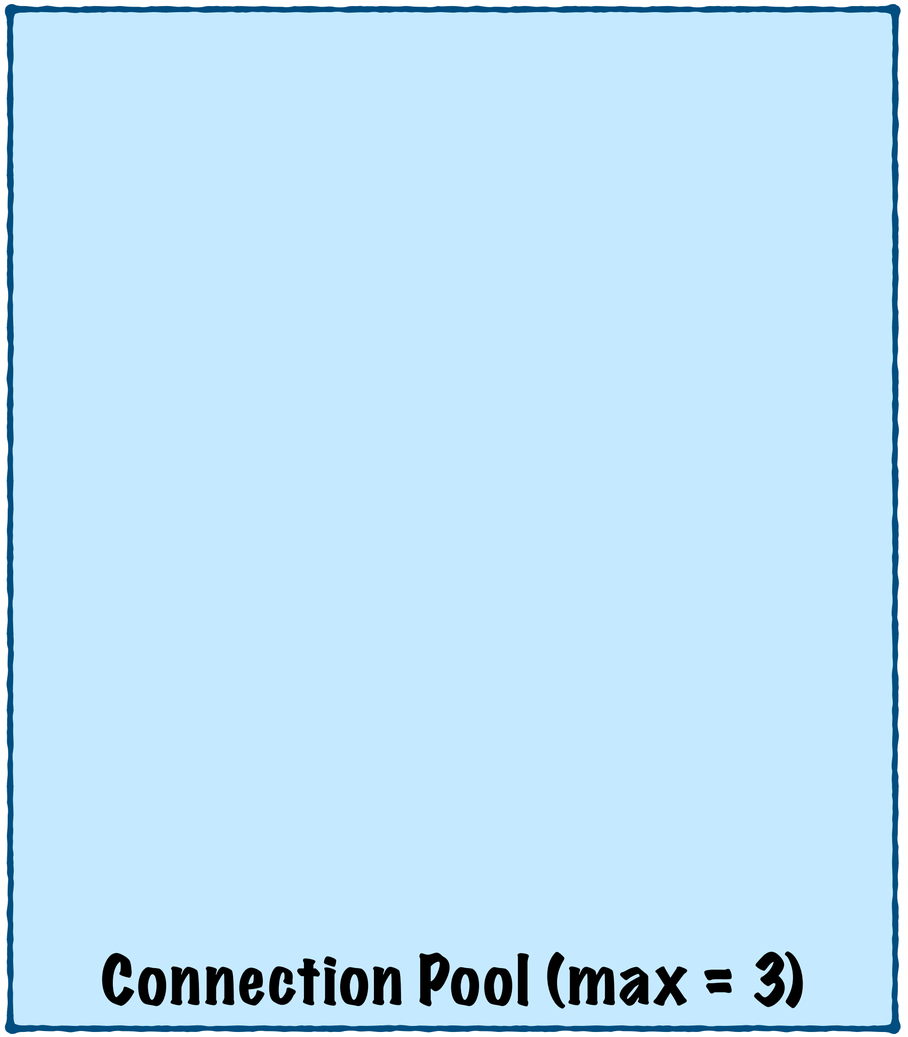

Connection Pool (max = 3)


Request 1
Request 2


Connection Pool (max = 3)


Request 1


Request 2

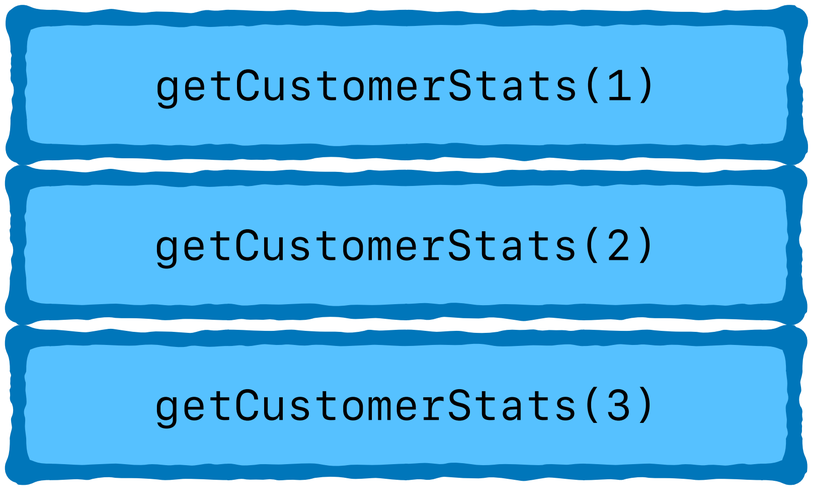

Connection Pool (max = 3)

```
const customers = await getTopCustomers();

for (const c of customers) {
  c.stats = await getCustomerStats(customer.id)
}
```

```
const customers = await getTopCustomers();

for (const c of customers) {
  c.stats = await getCustomerStats(customer.id)
}
```

```
const customers = await getTopCustomers();

for (const c of customers) {
  c.stats = await getCustomerStats(customer.id)
}
```


getCustomerStats(1)
getCustomerStats(2)
getCustomerStats(3)


getCustomerStats(1)
getCustomerStats(2)
getCustomerStats(3)

```
const customers = await getTopCustomers();

for (const c of customers) {
  c.stats = await getCustomerStats(customer.id)
}
```

```
const customers = await getTopCustomers();

await Promise.all(customers.map(async (c) => {
  c.stats = await getCustomerStats(customer.id)
}));
```

```
const customers = await getTopCustomers();

await Promise.all(customers.map(async (c) => {
  c.stats = await getCustomerStats(customer.id)
}));
```

```
const customers = await getTopCustomers();

await Promise.all(customers.map(async (c) => {
  c.stats = await getCustomerStats(customer.id)
}));
```

C#: Task.WhenAll

```
const customers = await getTopCustomers();

await Promise.all(customers.map(async (c) => {
  c.stats = await getCustomerStats(customer.id)
});
```


20 SEQUENTIAL QUERIES


20 PARALLEL QUERIES
imgflip.com

```
const customers = await getTopCustomers();

await Promise.all(customers.map(async (c) => {
  c.stats = await getCustomerStats(customer.id)
});
```

Needs N = customers.length connections from the pool

```
const customers = await getTopCustomers();

await Promise.all(customers.map(async (c) => {
  c.stats = await getCustomerStats(customer.id)
}));
```

```
const customers = await getTopCustomers();

await pMap(customers, async (c) => {
  c.stats = await getCustomerStats(customer.id)
}, { concurrency: 2 });
```

```
const customers = await getTopCustomers();

await pMap(customers, async (c) => {
  c.stats = await getCustomerStats(customer.id)
}, { concurrency: 2 });
```

```
const customers = await getTopCustomers();

await pMap(customers, async (c) => {
  c.stats = await getCustomerStats(customer.id)
}, { concurrency: 2 });
```

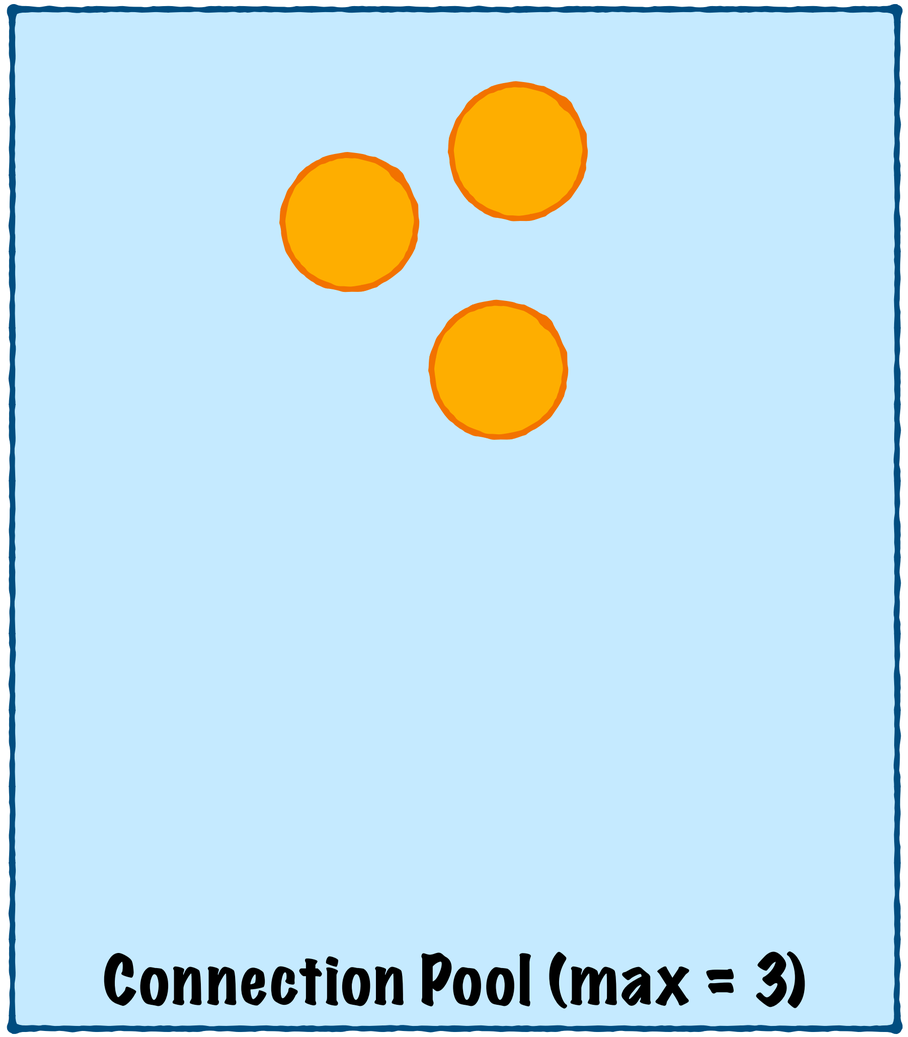
C#: Parallel.ForEachAsync

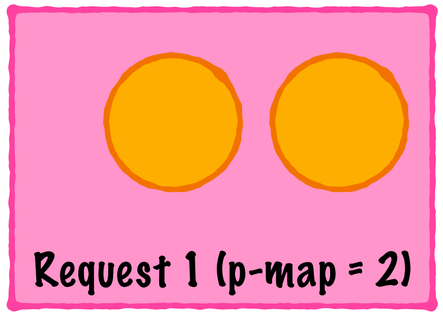
```
const customers = await getTopCustomers();

await pMap(customers, async (c) => {
  c.stats = await getCustomerStats(customer.id)
}, { concurrency: 2 });
```

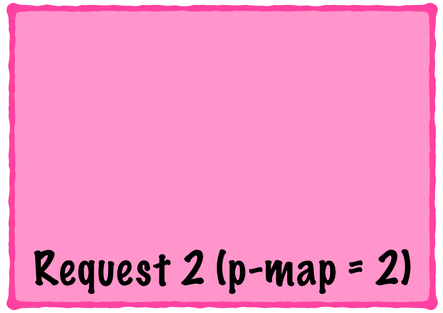
MaxDegreesOfParallelism

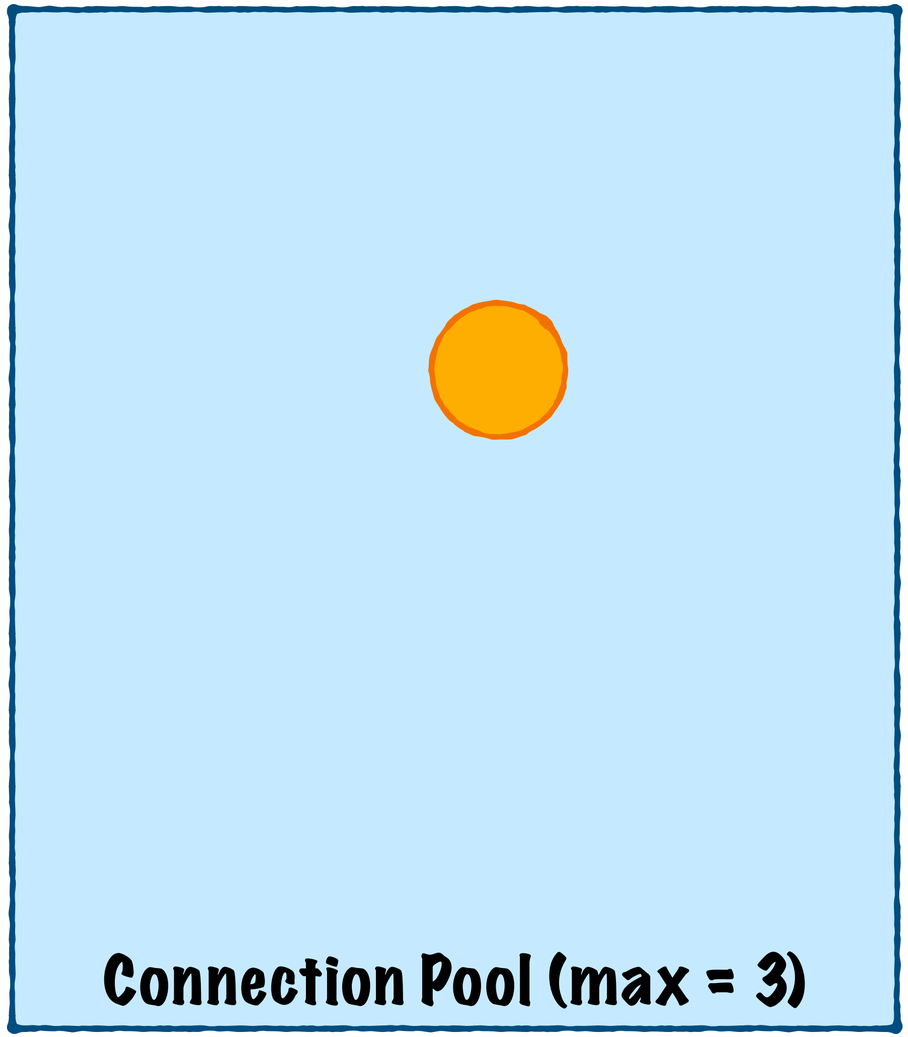
Request 1 (p-map = 2)

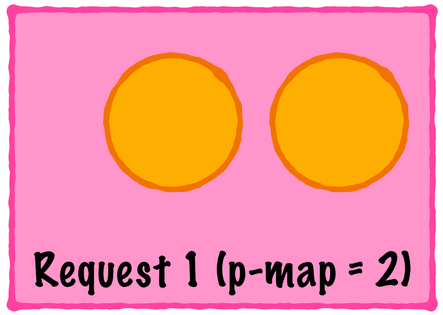
Request 2 (p-map = 2)

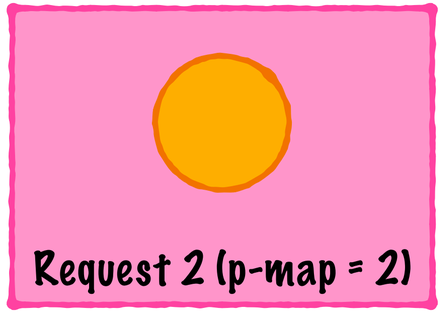


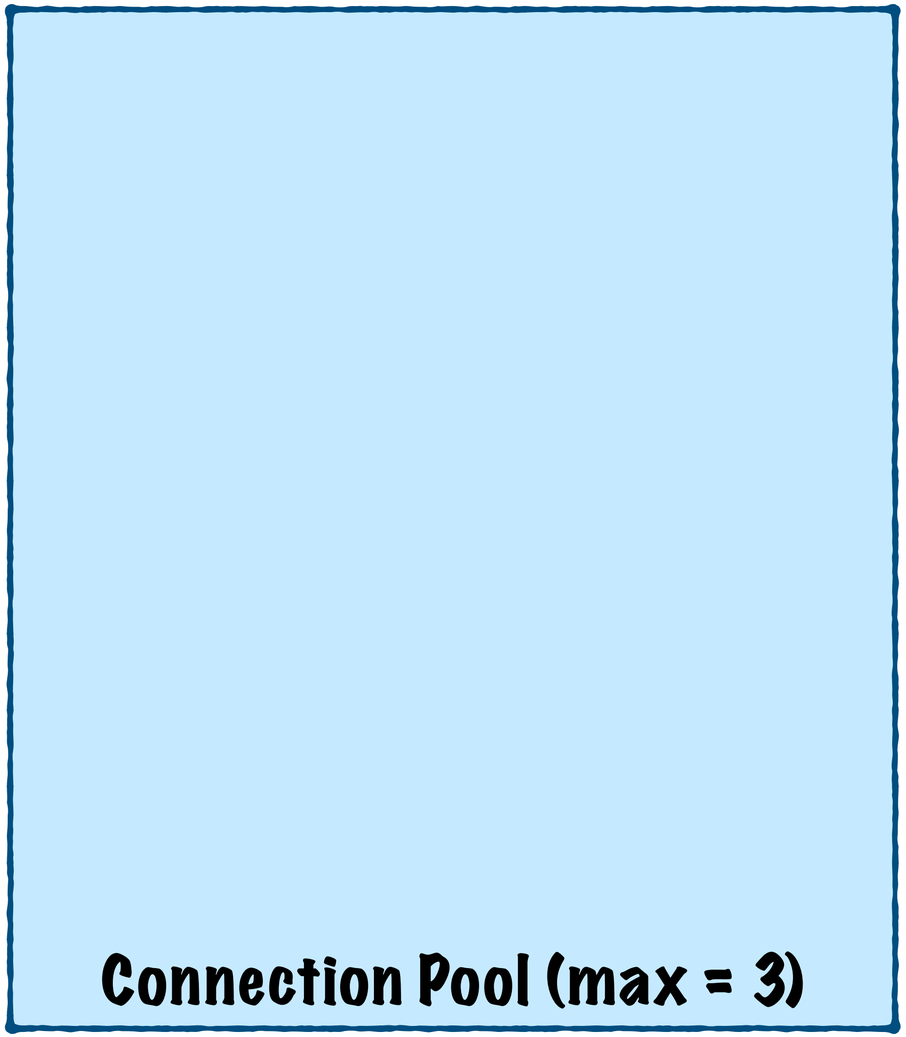

Request 1 (p-map = 2)

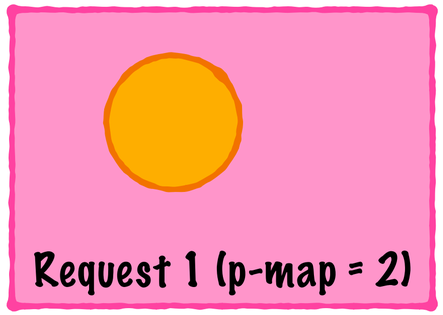

Request 2 (p-map = 2)

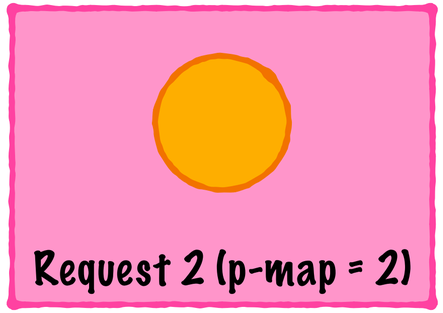

Connection Pool (max = 3)

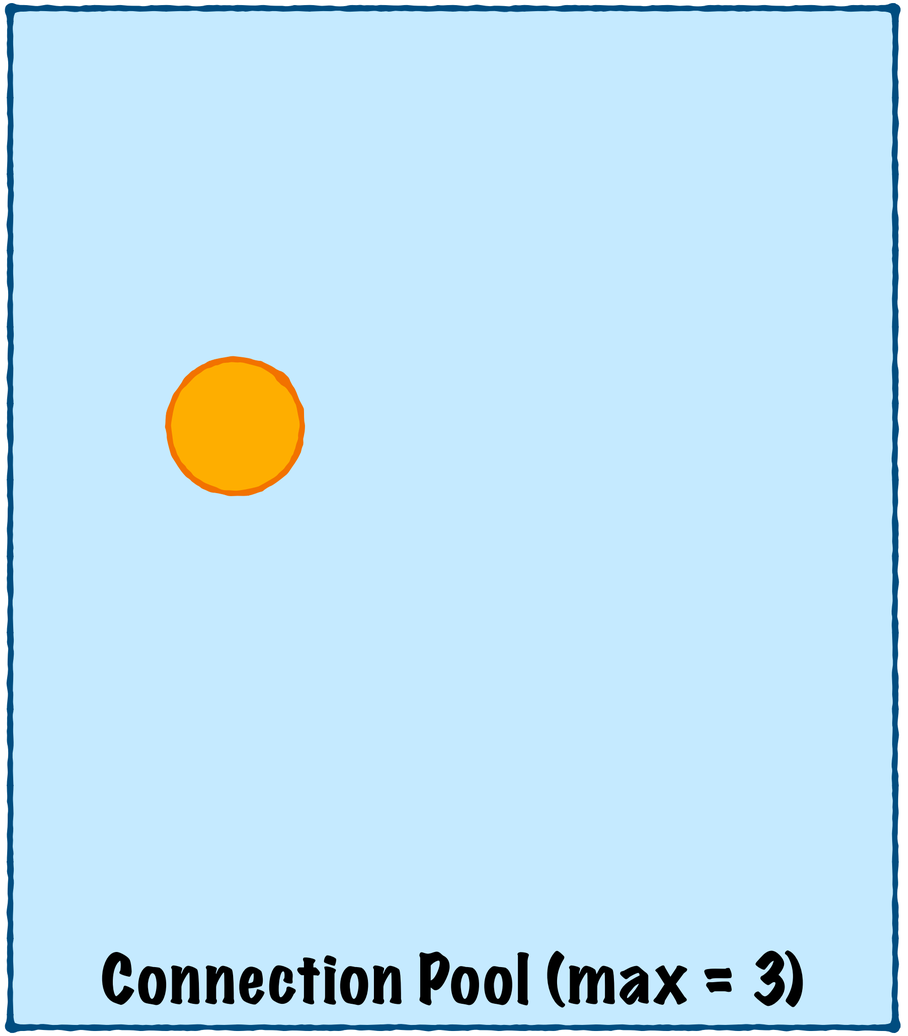

Request 1 (p-map = 2)

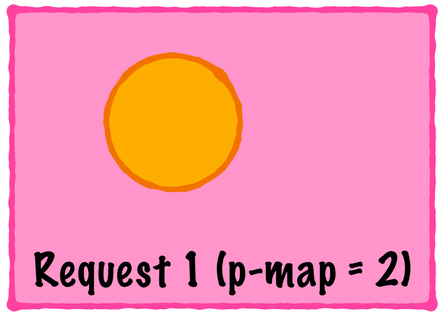

Request 2 (p-map = 2)

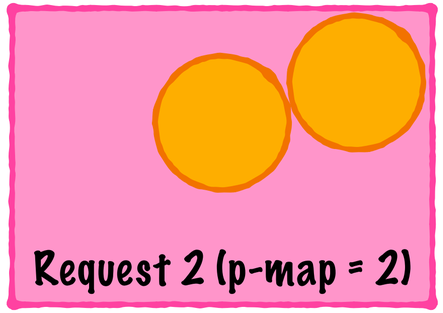

Connection Pool (max = 3)

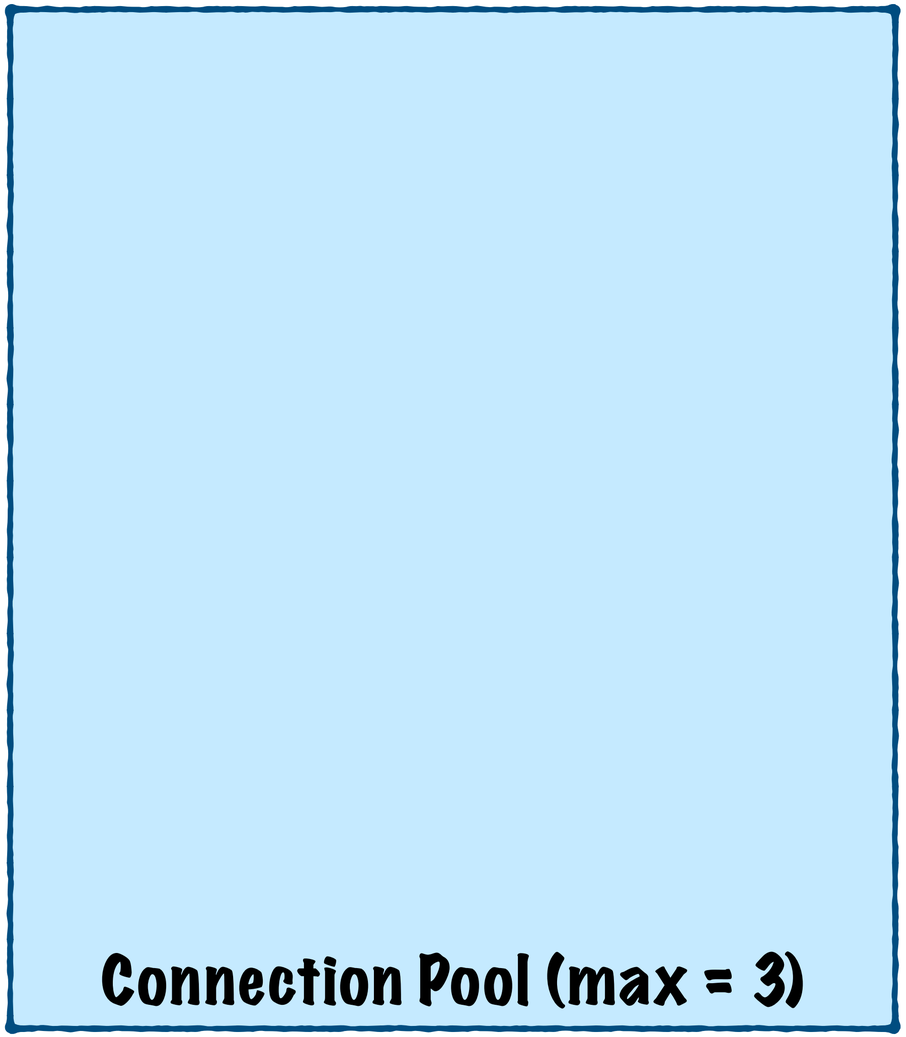

Request 1 (p-map = 2)

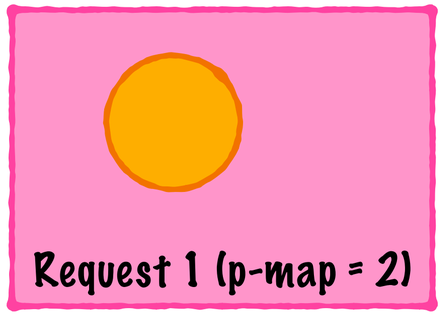

Request 2 (p-map = 2)

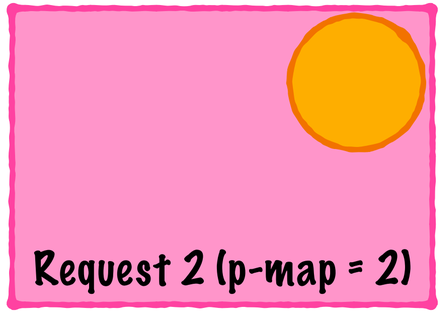

Connection Pool (max = 3)

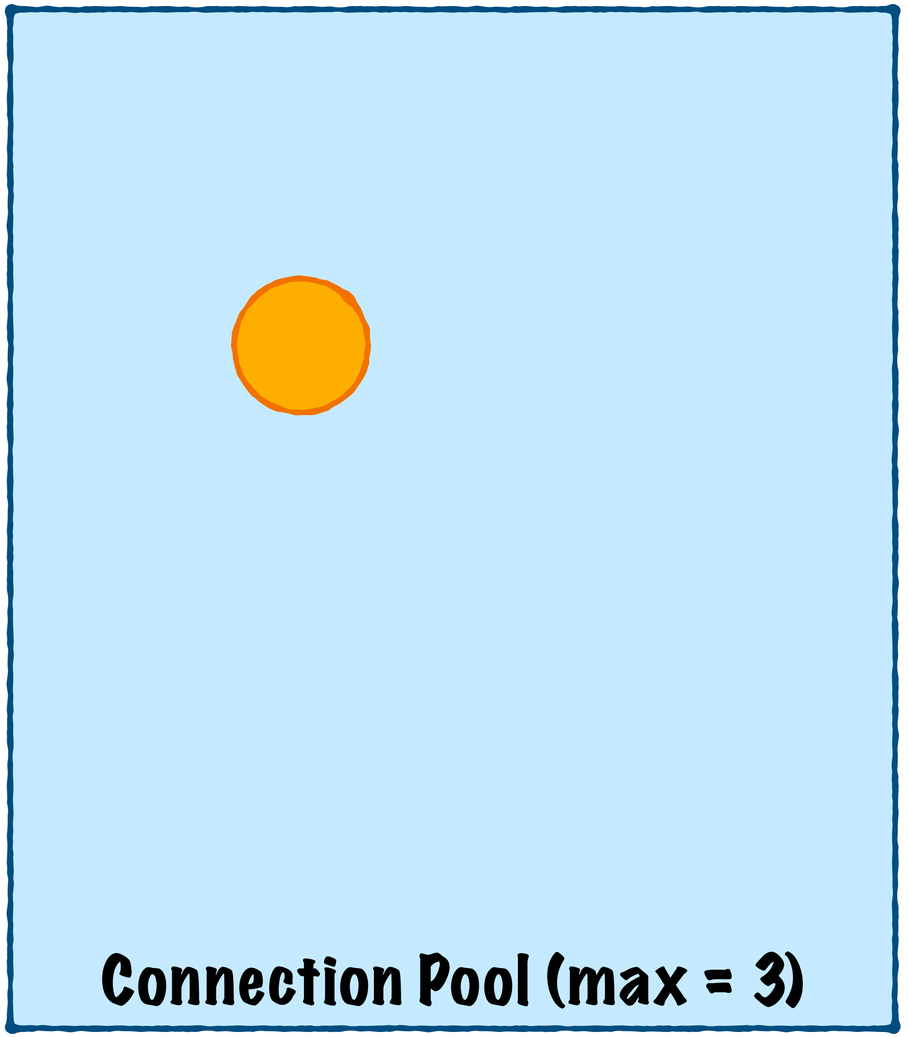

Request 1 (p-map = 2)

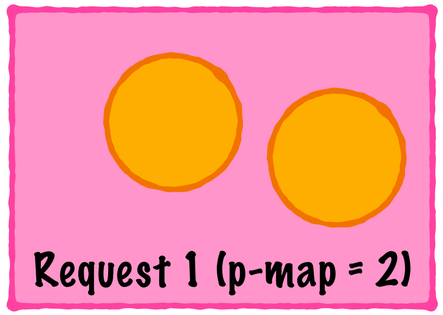

Request 2 (p-map = 2)

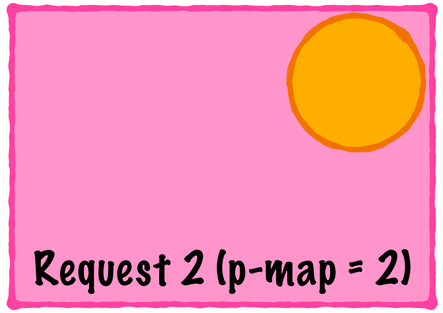

Connection Pool (max = 3)

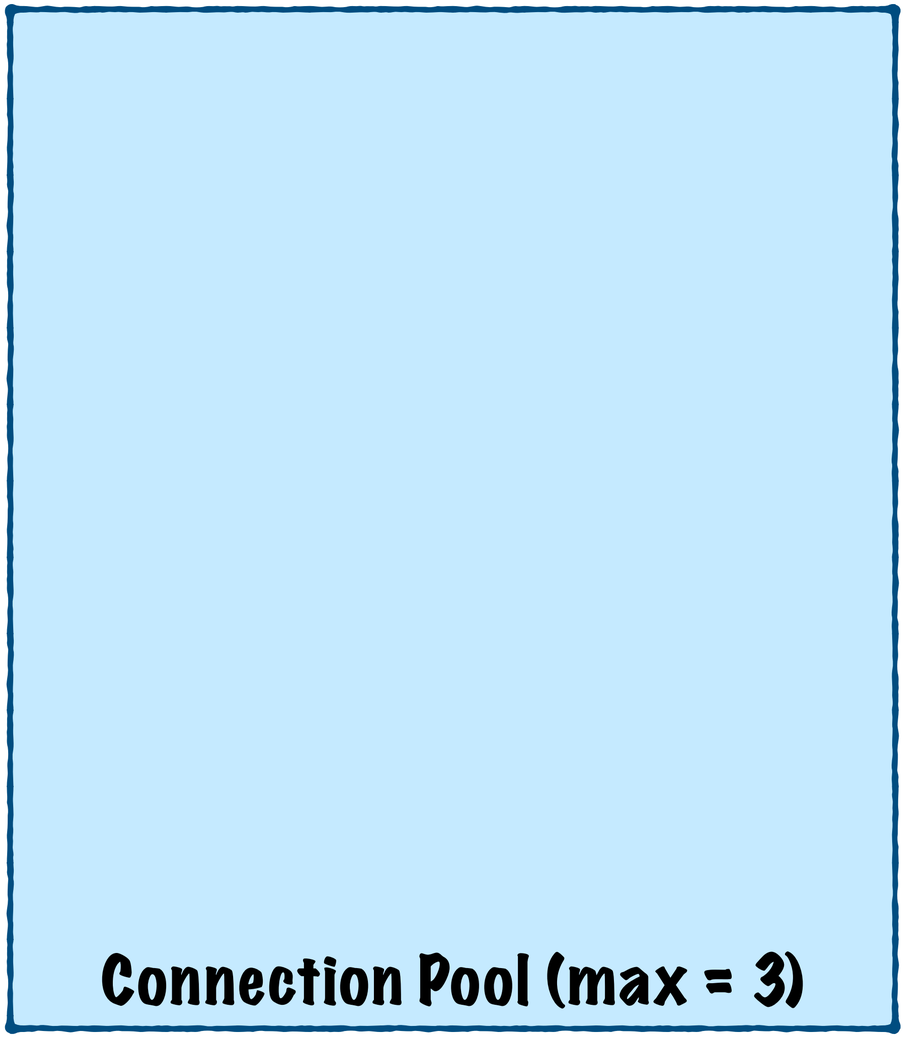

Request 1 (p-map = 2)


Request 2 (p-map = 2)


Connection Pool (max = 3)


Request 1 (p-map = 2)


Request 2 (p-map = 2)

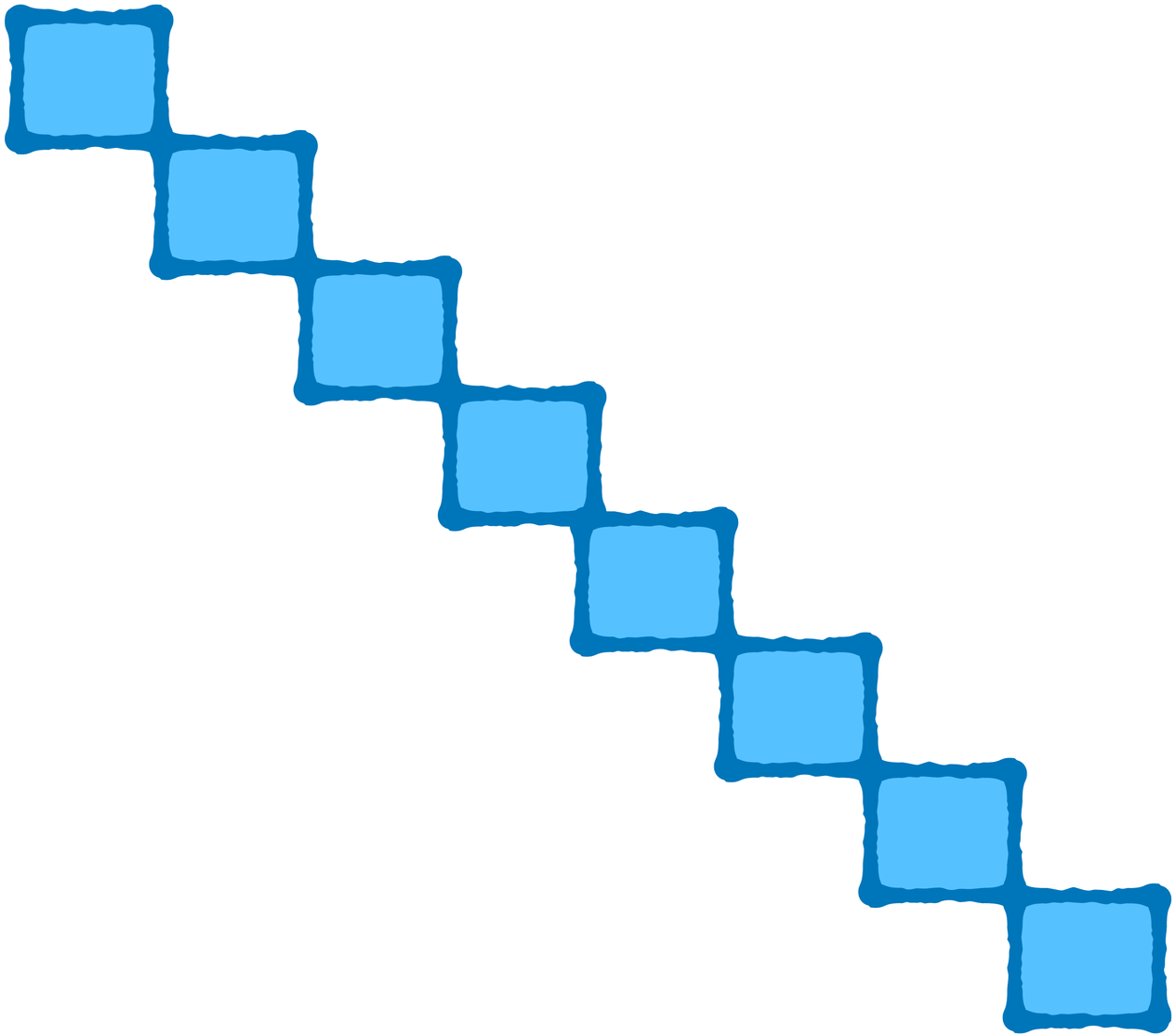

Connection Pool (max = 3)

It doesn’t necessarily complete faster

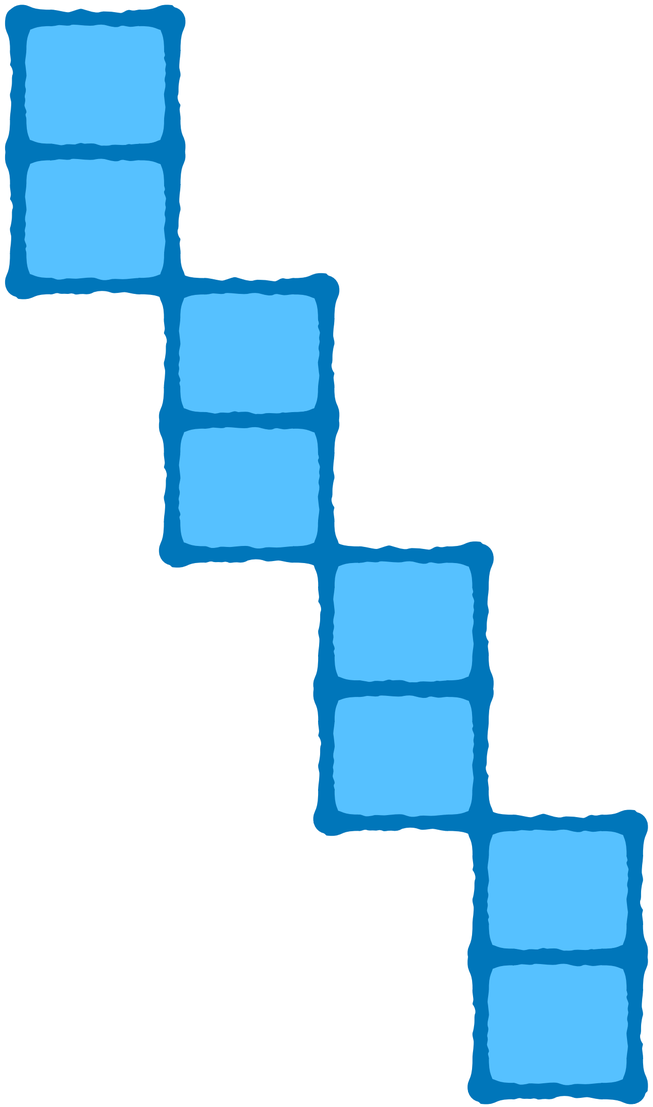
“What concurrency limit should I use?”

-You, probably

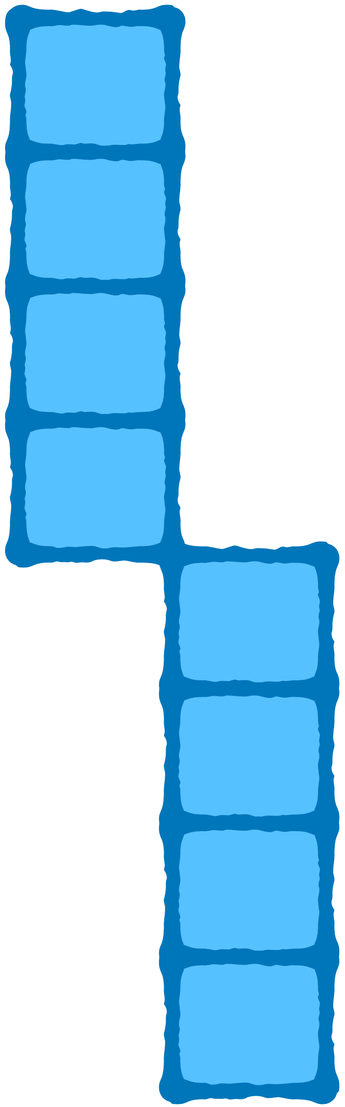
- Concurrency: 1 => 1,000ms



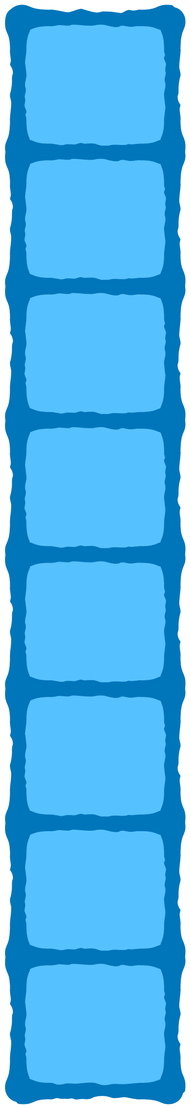
- Concurrency: 1 => 1,000ms
- Concurrency: 2 => 500ms



- Concurrency: 1 => 1,000ms
- Concurrency: 2 => 500ms
- Concurrency: 4 => 250ms



- Concurrency: 1 => 1,000ms
- Concurrency: 2 => 500ms,
- Concurrency: 4 => 250ms
- Concurrency: 8 => 125ms



Connection Pool Parameters

Fiddle with these a bit

- Maximum Connection Count
- Minimum Connection Count
- Idle Timeout
- Maximum Lifetime

Tricky: Issue is systemic

The princess is in another castle

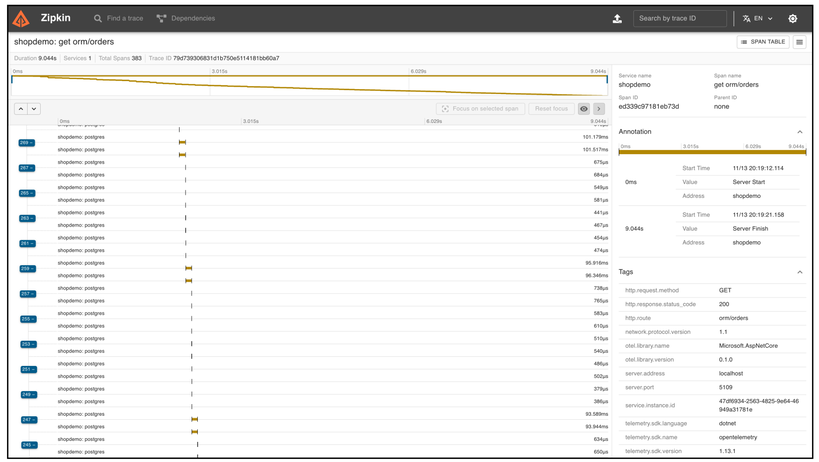
Tricky: Doesn’t reproduce locally

We needed a load test

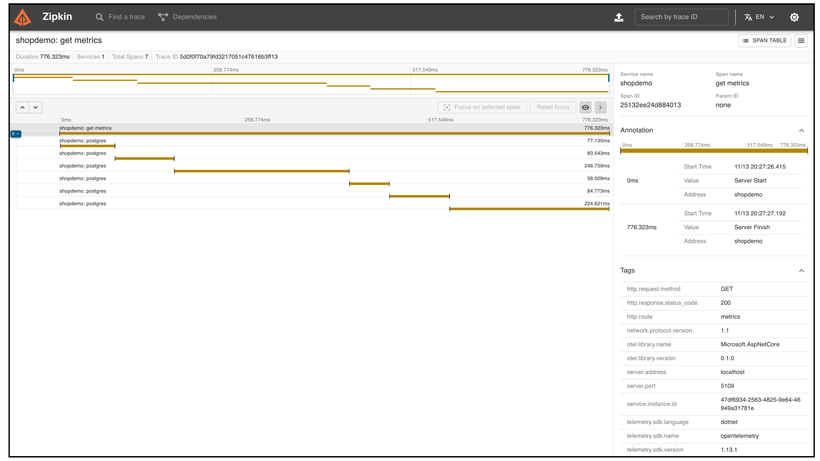
Other Resources

Watch out for these

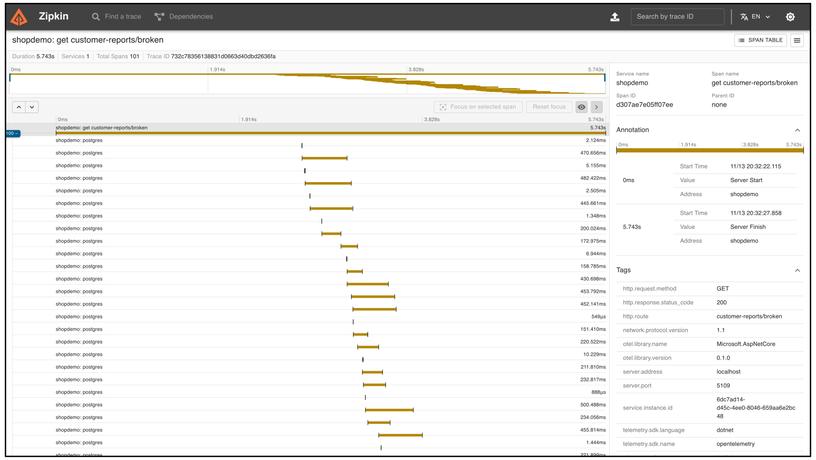
- Threads
- Disk IO (File Handles)
- TCP Connections / Ports (HTTP Clients)



Many Small Spans



Large Spans



Span Gaps

Often a lot of low hanging fruit

Database indexing is easiest to fix architecturally

N+1 will often have biggest impact

Stop scaling and start tuning

Feedback

Resources