Making Sense of your Data

BUILDING A CUSTOM MONGODB DATASOURCE FOR GRAFANA WITH VERTX

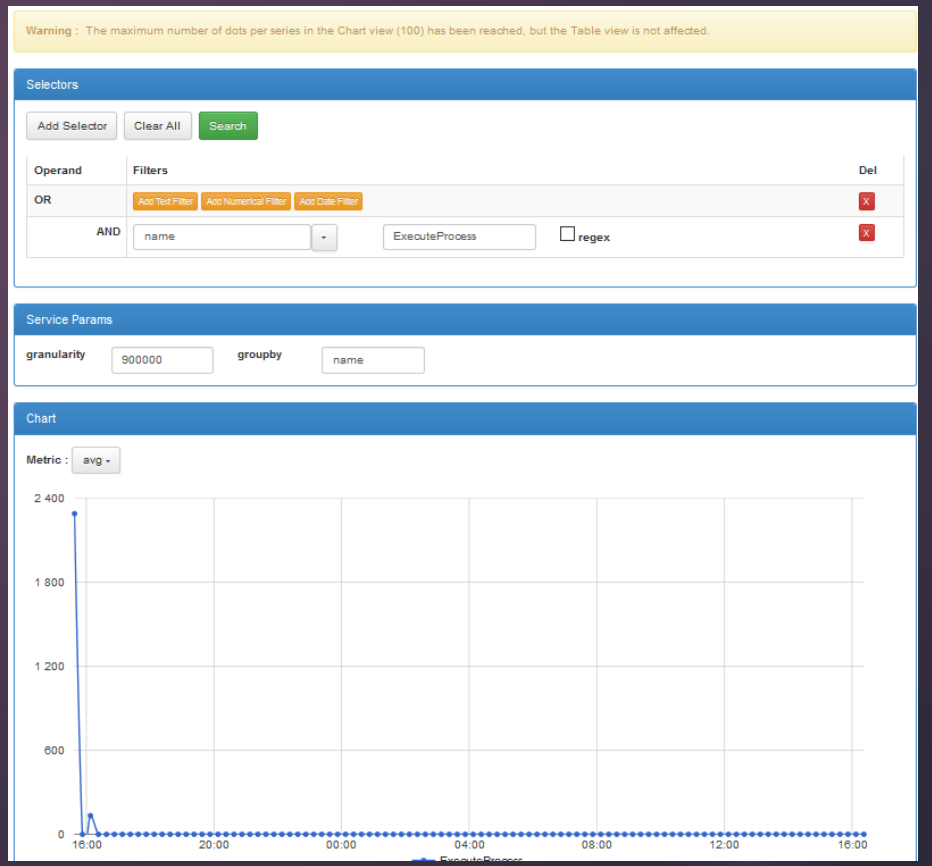
About me

- IT Consultant & Java Specialist at DevCon5 (CH)
- Focal Areas
  - Tool-assisted quality assurance
  - Performance (-testing, -analysis, -tooling)
  - Operational Topics (APM, Monitoring)

- Twitter: @gmuecke

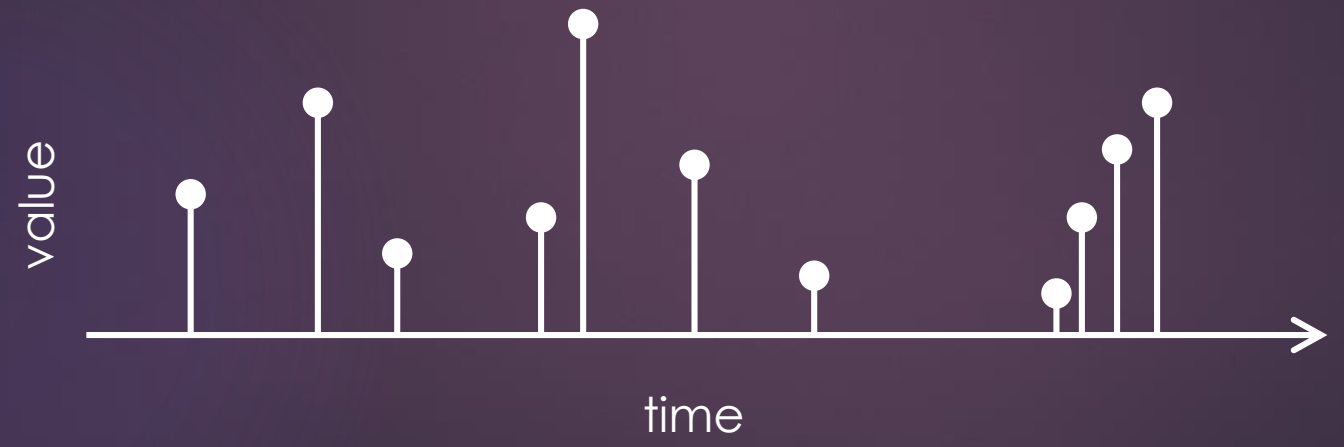
The Starting Point

- Customer stored and keep response time measurement of test runs in a MongoDB
  - Lots of Data
  - Timestamp & Value
  - No Proper Visualization



What are timeseries data?

- a set of datapoints with a timestamp and a value

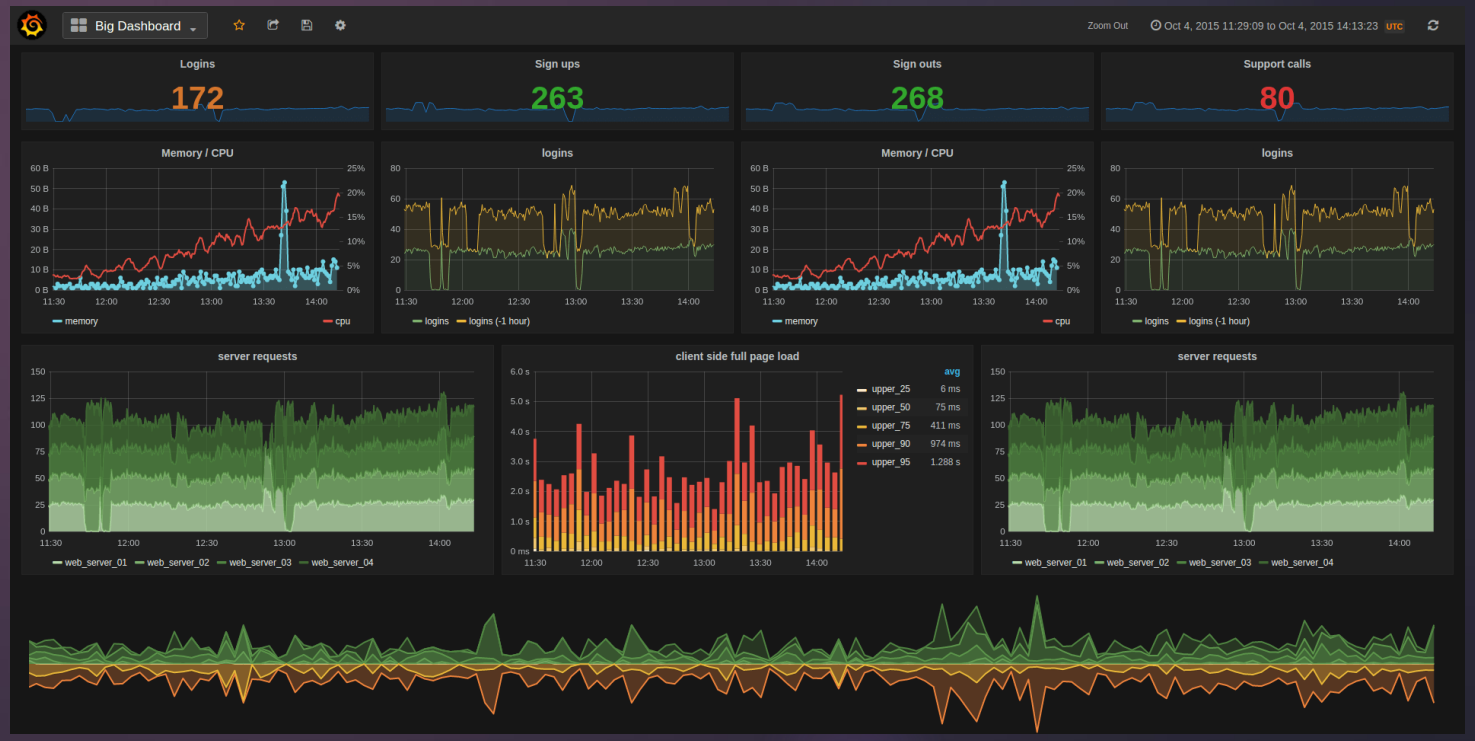


What is MongoDB?

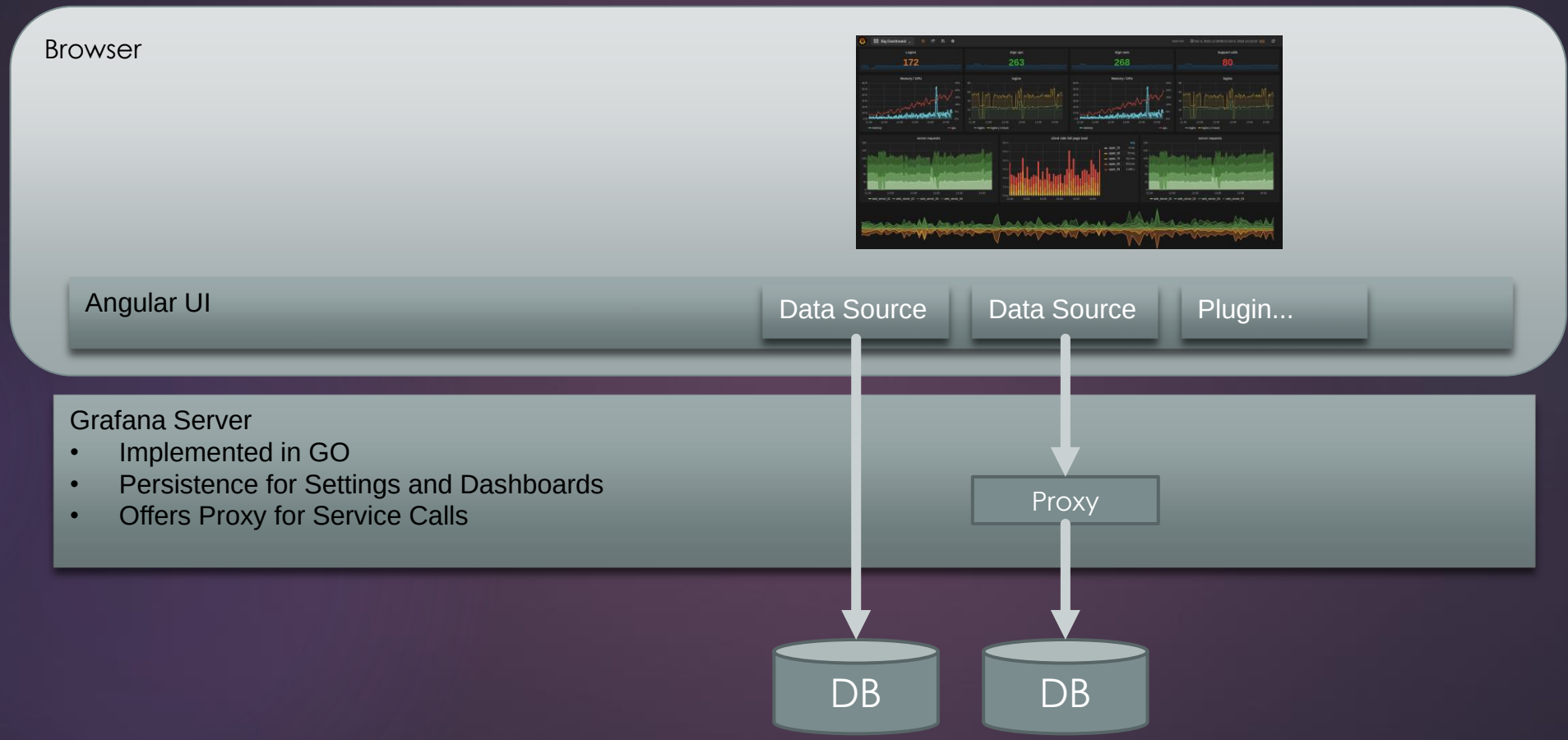
- MongoDB
  - NoSQL database with focus on scale
  - JSON as data representation
  - No HTTP endpoint (TCP based Wire Protocol)
  - Aggregation framework for complex queries
  - Provides an Async Driver

What is Grafana?

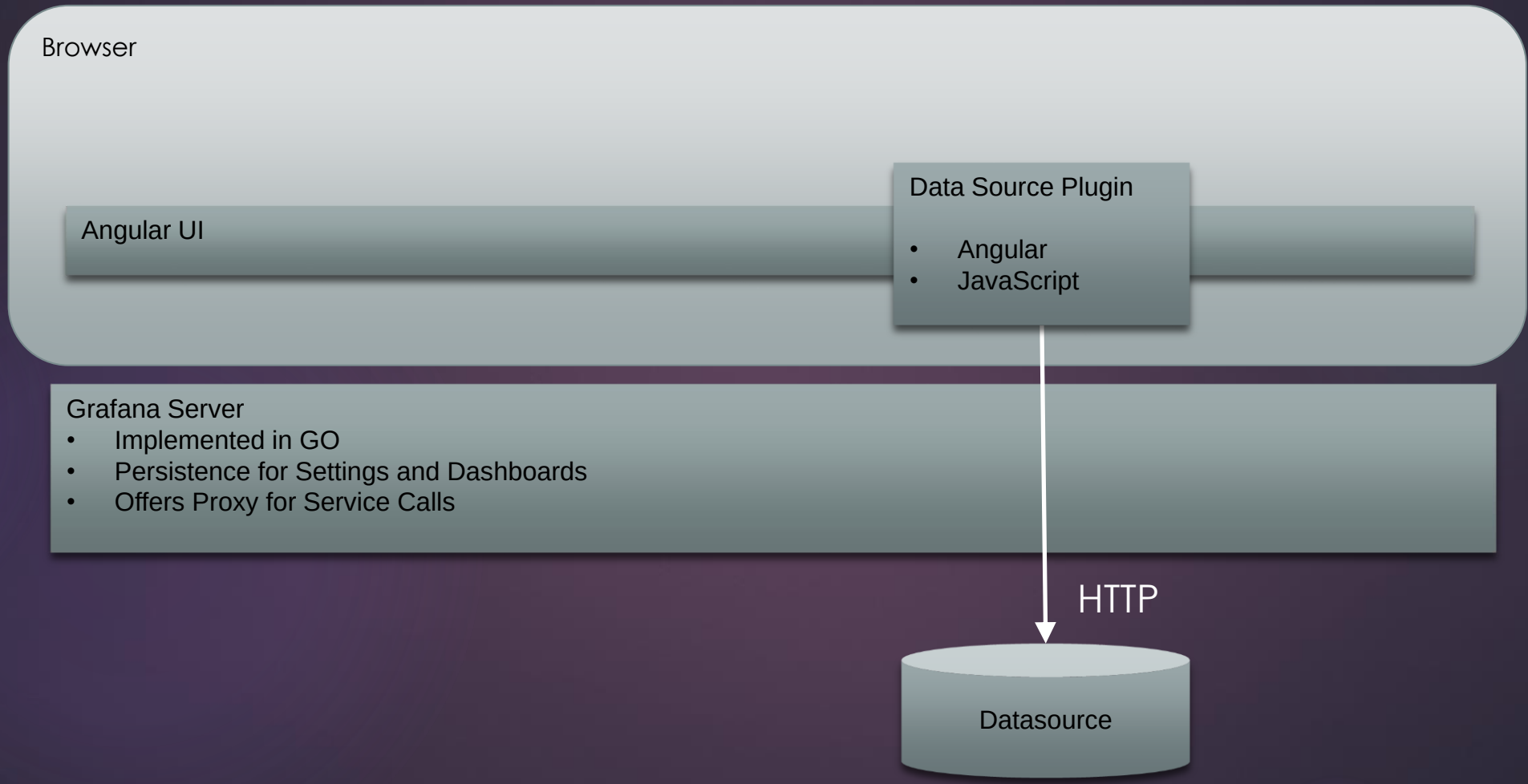
- A Service for Visualizing Time Series Data
- Open Source
- Backend written in Go
- Frontend based on Angular
- Dashboards & Alerts



Grafana Architecture



Datasources for Grafana



Connect Angular Directly to Mongo?

Can angularjs connect directly to mongodb?

2

I am new to angularjs and as the title said, I am wondering if there is a way to connect angularjs directly to mongodb without coding additional server side using express.js. i tried to search on the Internet but i cannot find any resources.

angularjs mongodb

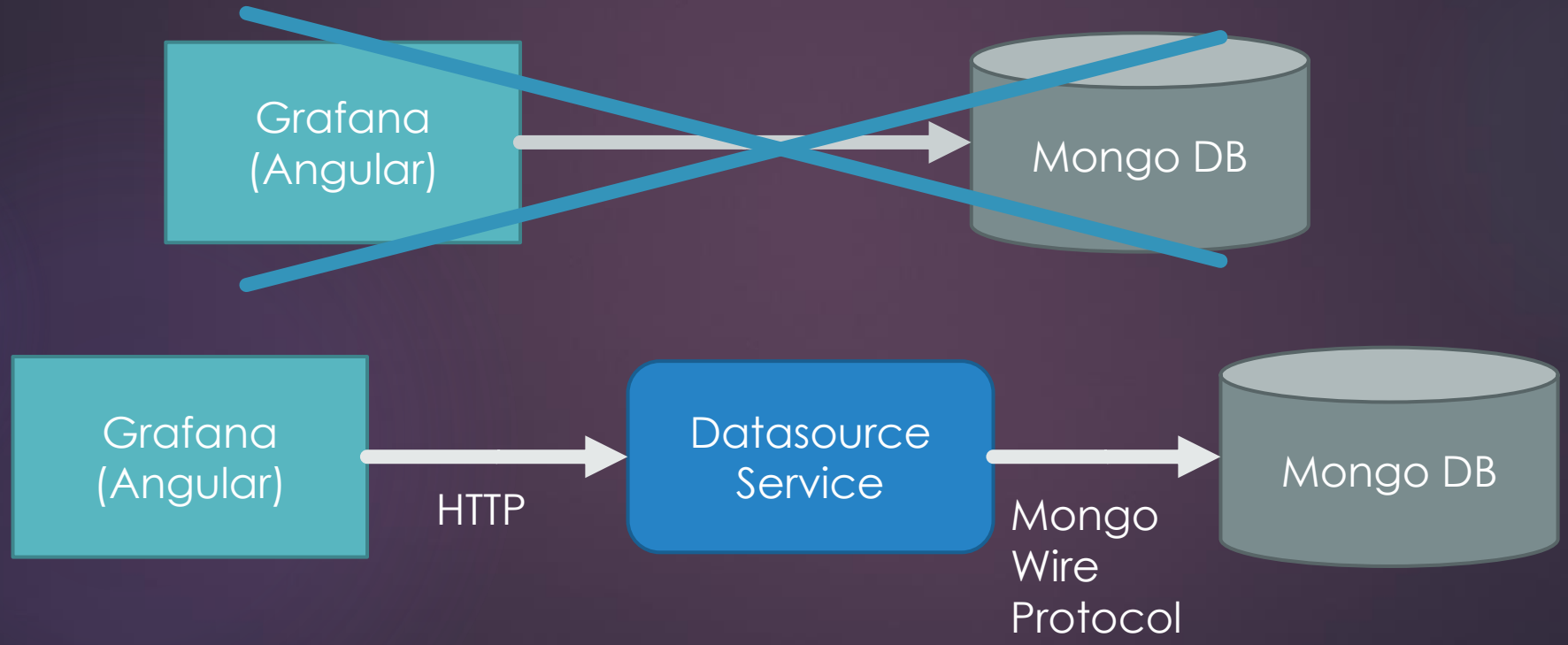
1 No. Try using CouchDB instead – Wayne Ellery Jun 18 '15 at 0:36

3

Sorry what you are trying to do in not possible. You will need to introduce some serverside technologies so that you can talk to the database and form some sort of api that will return JSON data depending on certain business rules coded into the backend of your application. AngularJS has templating built into it which receives JSON data and places it as you direct it to throughout the DOM.

What your asking is logical, I used to wonder this as well coming from the frontend world. If this no-server side code where ever to happen, the database queries would be exposed to the user on the client side. The client could then modify the AngularJS "query syntax" in the code inspector.

From 2 Tier to 3 Tier



Start Simple

- SimpleJsonDatasource (Plugin)
- 3 ServiceEndpoints
  - /search → Labels – names of available timeseries
  - /annotations → Annotations – textual markers
  - /query → Query – actual time series data

https://github.com/grafana/simple-json-datasource

/search Format

Request

```
{
    "target" : "select metric",
    "refId" : "E"
  }
```

Response

```
[
      "Metric Name 1",
      "Metric Name2",
  ]
```

An array of strings

/annotations Format

Request


```
{     "annotation" : {
       "name" : "Test",
       "iconColor" : "rgba(255, 96, 96, 1)",
       "datasource" : "Simple Example DS",
       "enable" : true,
       "query" : "{\"name\":\"Timeseries A\"}"      },
      "range" : {
       "from" : "2016-06-13T12:23:47.387Z",
       "to" : "2016-06-13T12:24:19.217Z"      },
      "rangeRaw" : {
       "from" : "2016-06-13T12:23:47.387Z",
       "to" : "2016-06-13T12:24:19.217Z"
      }    }
```


Response


```
[     {       "annotation": {
         "name": "Test",
         "iconColor": "rgba(255, 96, 96, 1)",
         "datasource": "Simple Example DS",
         "enable": true,
         "query": "{\"name\":\"Timeseries A\"}"       },
        "time": 1465820629774,
        "title": "Marker",
        "tags": [
          "Tag 1",
          "Tag 2"       ]     }    ]
```

/query Format

Request

```
{     "panelId" : 1,
      "maxDataPoints" : 1904,
      "format" : "json",
      "range" : {
        "from" : "2016-06-13T12:23:47.387Z",
        "to" : "2016-06-13T12:24:19.217Z"      },
      "rangeRaw" : {
        "from" : "2016-06-13T12:23:47.387Z",
        "to" : "2016-06-13T12:24:19.217Z"      },
      "interval" : "20ms",
      "targets" : [ {
        "target" : "Time series A",
        "refId" : "A"      },]  }
```

Response

```
[  {  "target":"Timeseries A",
      "datapoints":[
                [1936,1465820629774],
                [2105,1465820632673],
                [4187,1465820635570],
                [30001,1465820645243] },
    {  "target":"Timeseries B",
       "datapoints":[ ] }
    ]
```

Structure of the Source Data


```
{
        "_id" : ObjectId("56375bc54f3c4caedfe68aca"),
        "t" : {
                "eDesc" : "some description",
                "eId" : "56375ae24f3c4caedfe68a07",
                "name" : "some name",
                "profile" : "l01",
                "rnId" : "56375b694f3c4caedfe68aa0",
                "rnStatus" : "PASSED",
                "uId" : "anonymous"
        },
        "n" : {
                "begin" : NumberLong("1446468494689"),
                "value" : NumberLong(283)
        }
}
```

Custom Datasource

- Should be
  - Lightweight
  - Fast / Performant
  - Simple

Microservice?

- Options for implementation
  - Java EE Microservice (i.e. Wildfly Swarm)
  - Springboot Microservice
  - Vert.x Microservice
  - Node.js
  - ...

The Alternative Options

Node.js

- Single Threaded
- Child Worker Processes
- Javascript Only
- Not best-choice for heavy computation

Spring / Java EE

- Multithreaded
- Clusterable
- Java Only
- Solid Workhorses, cumbersome at times

Why Vert.x?

- High Performance, Low Footprint
  - Asynchronous, Non-Blocking
- Actor-like Concurrency
  - Event & Worker Verticles
  - Message Driven
- Polyglott
  - Java, Groovy, Javascript, Scala …
- Scalable
  - Distributed Eventbus
  - Multi-threaded Event Loops

But first, some basics

```
70 B$="WORLD!"
80 A$=B$
90 X=RND(0)
100  X=SIN(X)
110 PRINT I;" ";X;CHR$(65)
120 X=RND(0)
130 X=COS(X)
140 X=EXP(1)
150 X=LOG(X)
160 B=VAL("100")
170 C=LEN(A$)
180 X=ABS(X)
190 IF X<1000 THEN GOSUB 300
195 FOR J=1 TO 5
196 Y=1000*SQR(17*7)/3*17/123456
197 NEXT J
200 NEXT I
210 GOTO 500
300 X=0
320 RETURN
500 PRINT "KONEC ";T

READY
```

Asynchronous non-blocking vs Synchronous blocking

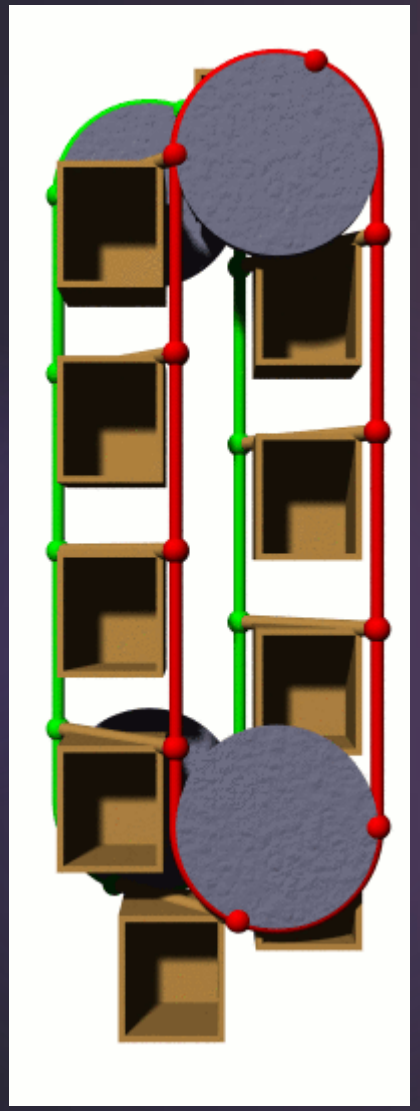
© Fritz Geller-Grimm

© Dontworry

Event Loop

Photo: Andreas Praefcke

Event Loop and Verticles



3rd Floor, Verticle A

2nd Floor, Verticle B

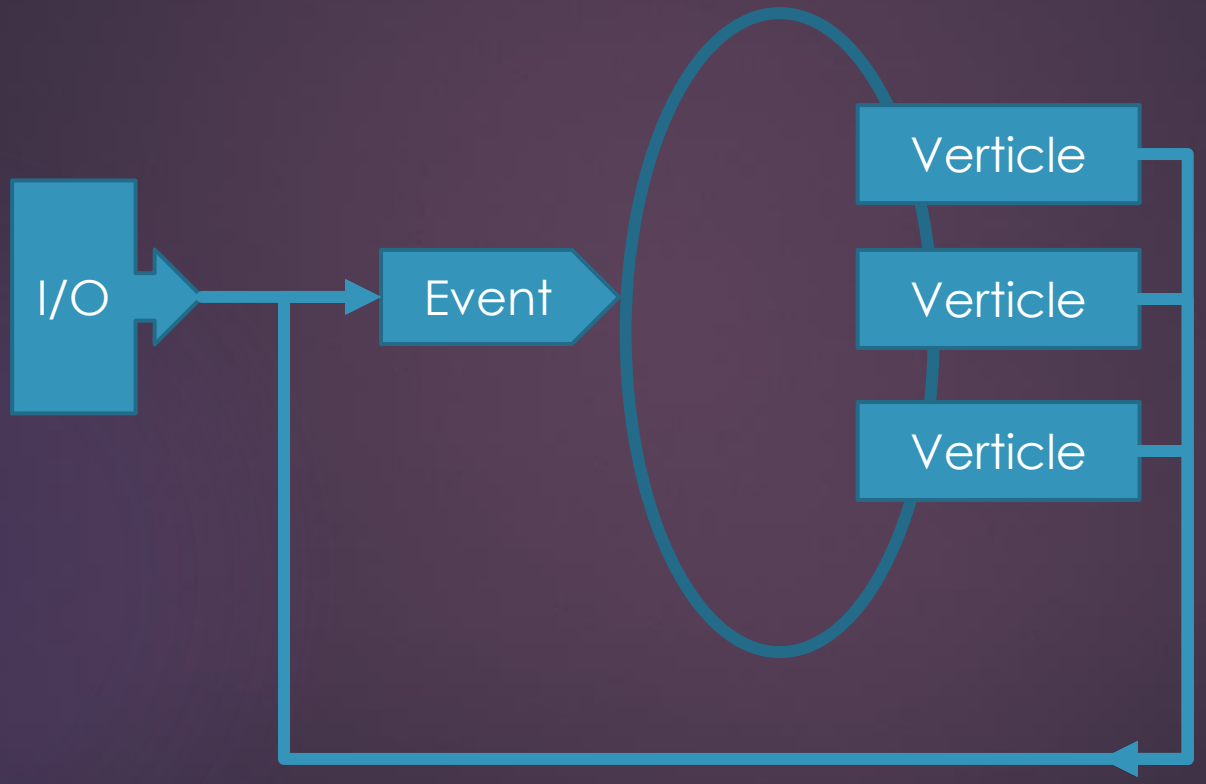
1st Floor, Verticle C

Photo: RokerHRO

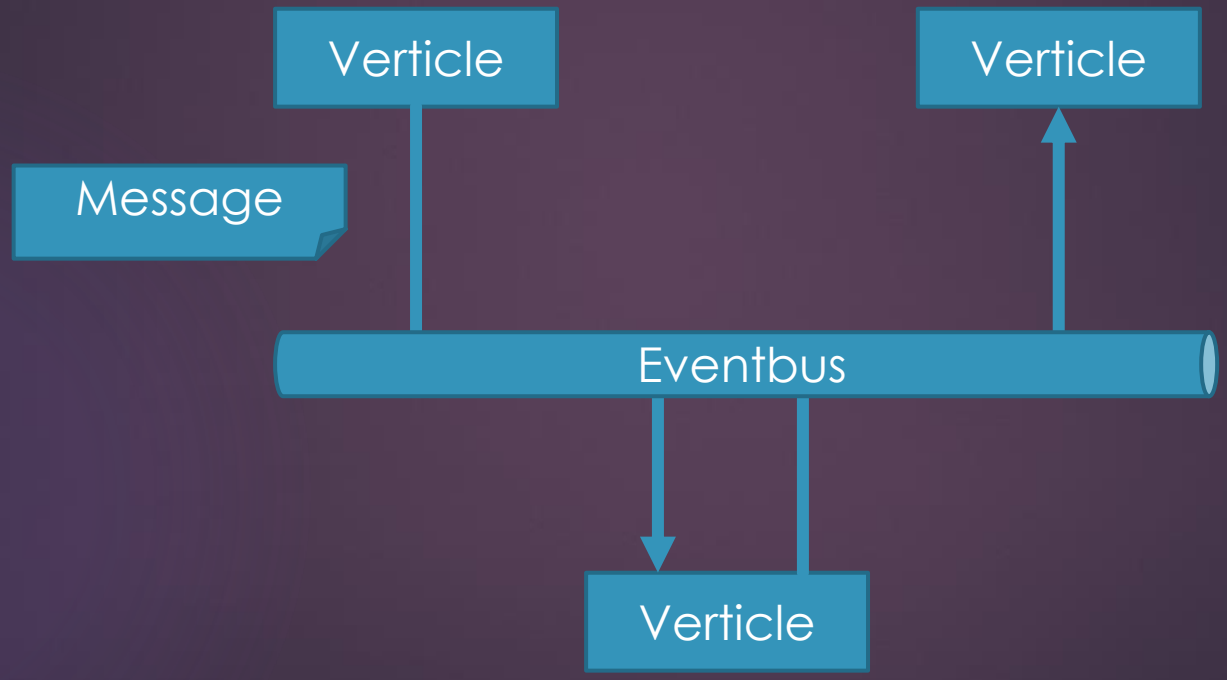

THIS IS NOT NPM
THERE ARE RULES!
memes.com


DON'T BLOCK THE
EVENT LOOP!

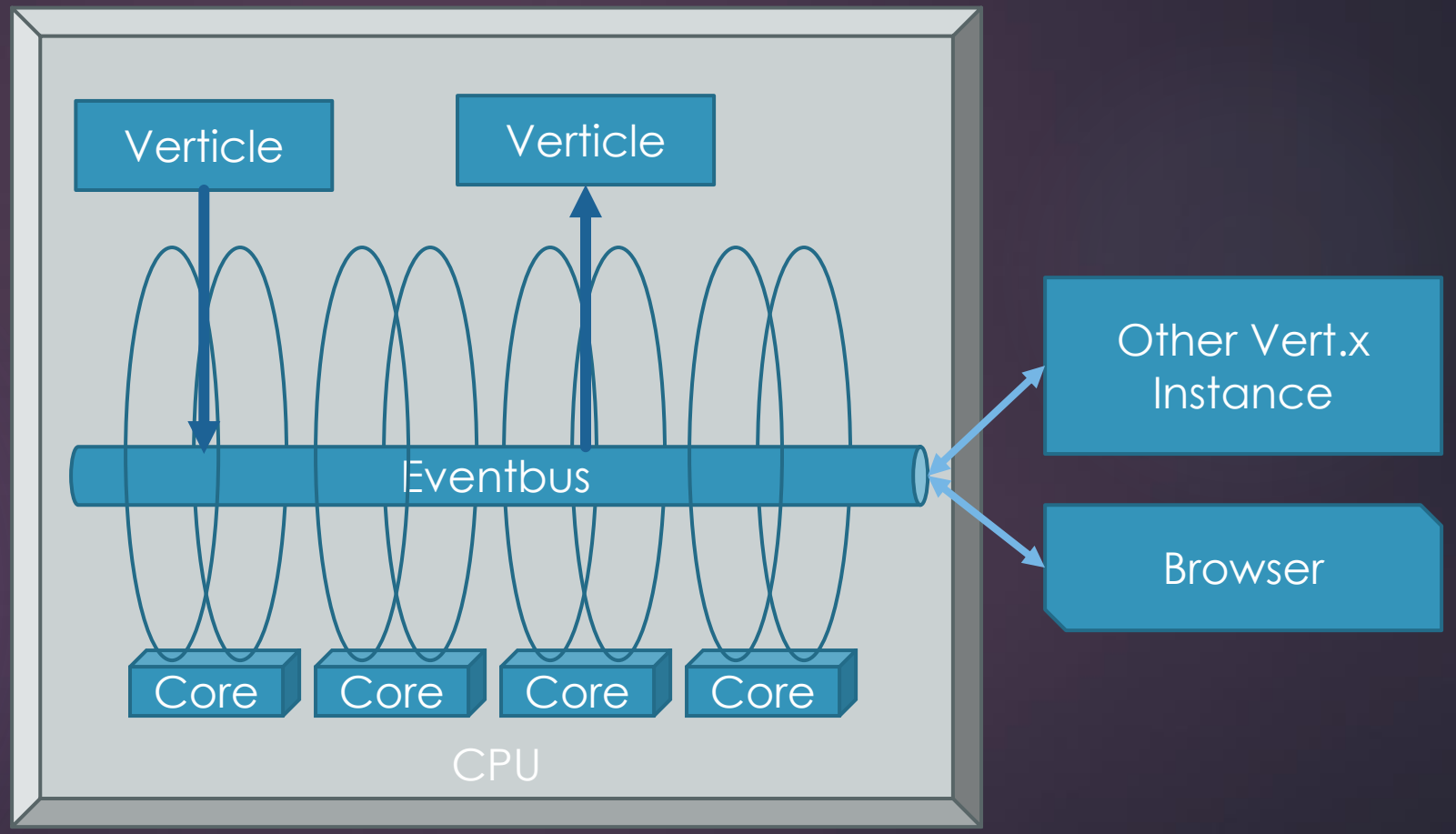
Event Loop



Event Bus

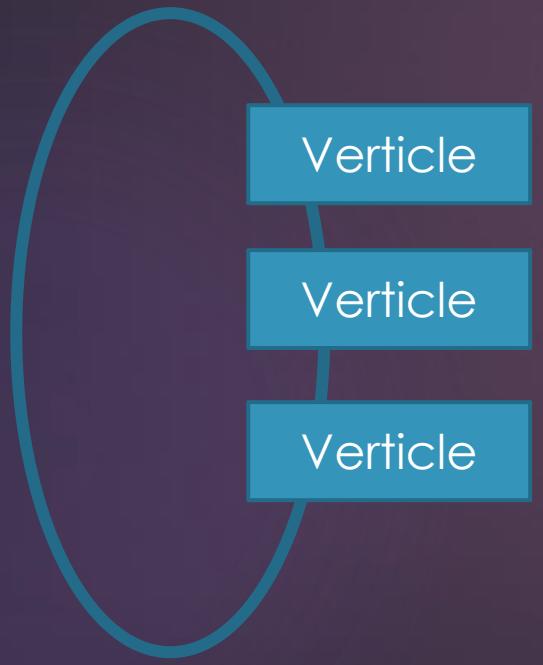


Multi-Reactor

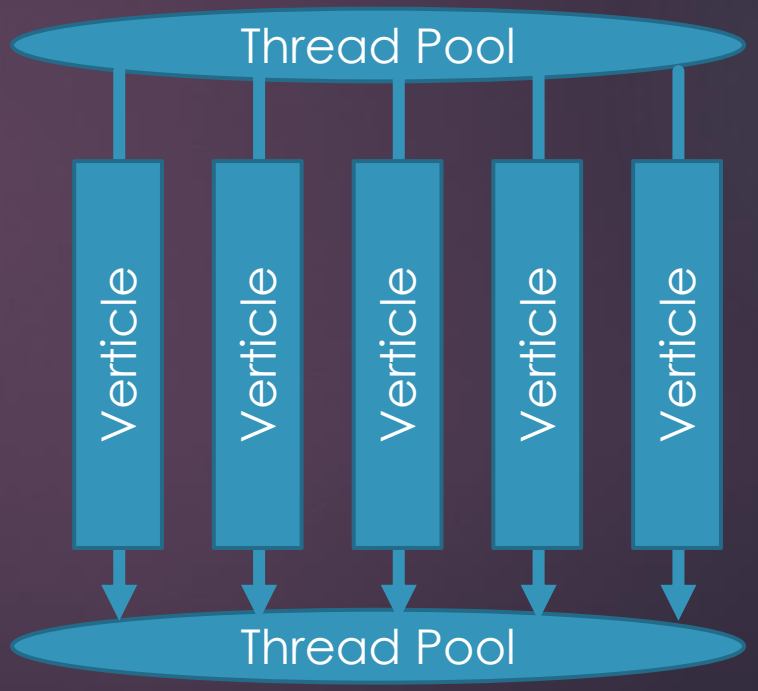


Event & Worker Verticles

Event Driven Verticles



Worker Verticles



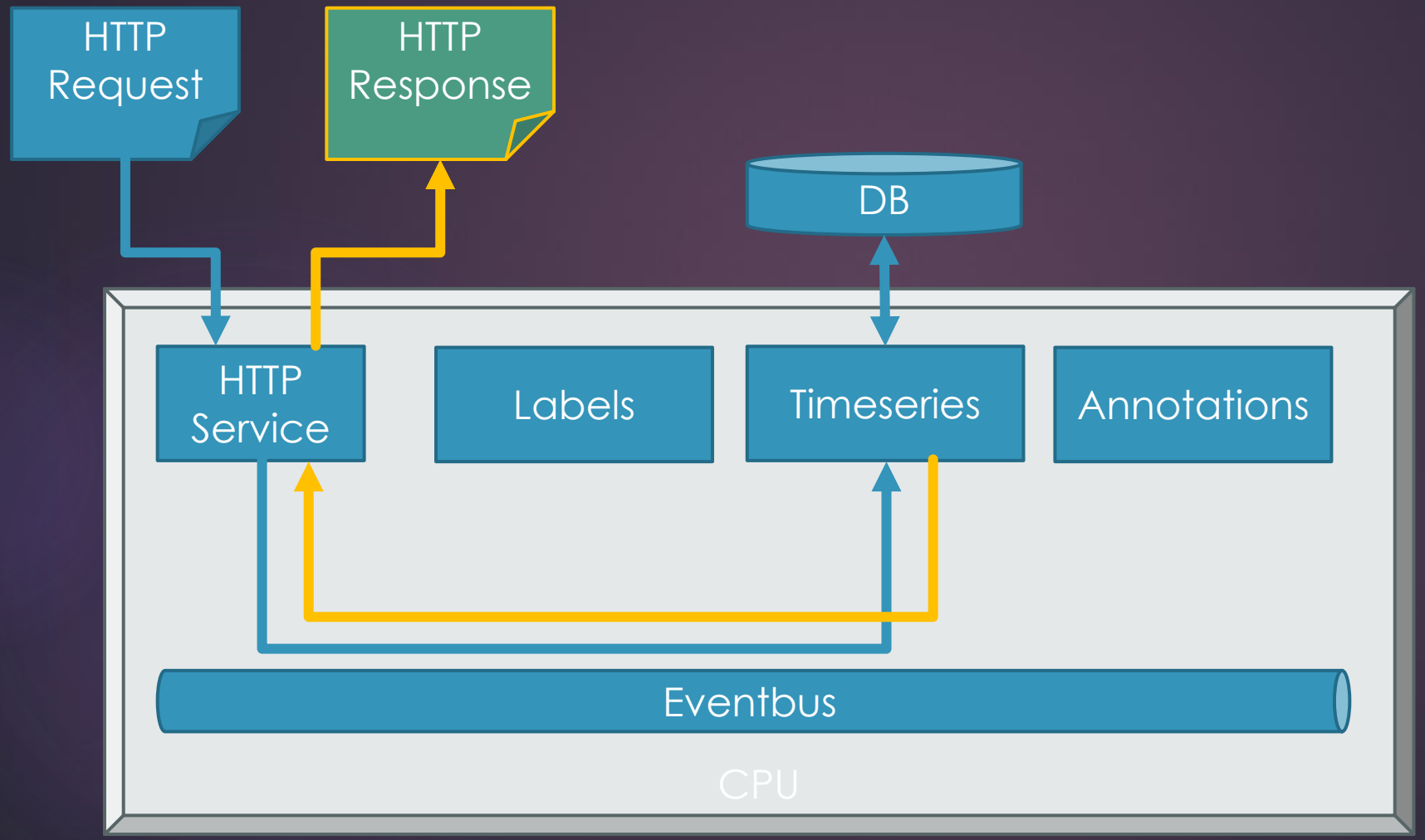
Implementing the datasource

- Http Verticle
  - Routing requests & sending responses
- Verticles querying the DB
  - Searching timeseries labels
  - Annotation
  - Timeseries data points
- Optional Verticles for Post Processing

What is the challenge?

- Optimization
  - Queries can be optimized
  - Large datasets have to be searched, read and transported
  - Source data can not be modified VS data redundancy
- Sizing
  - How to size the analysis system without knowing the query-times?
  - How to size thread pools or database pools if most of the queries will take 100ms – 30s ?

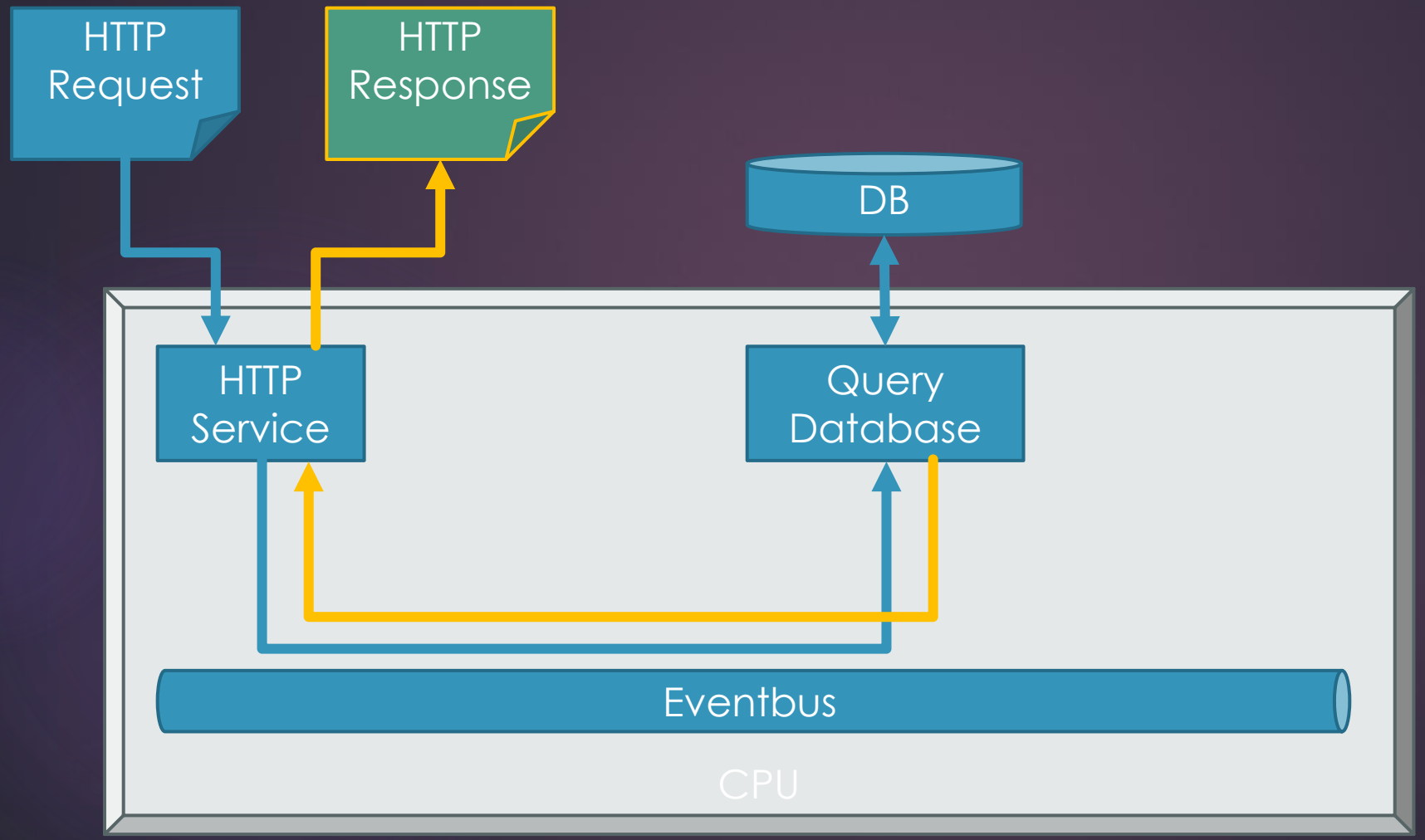
Datasource Architecture



Step 1 – The naive approach

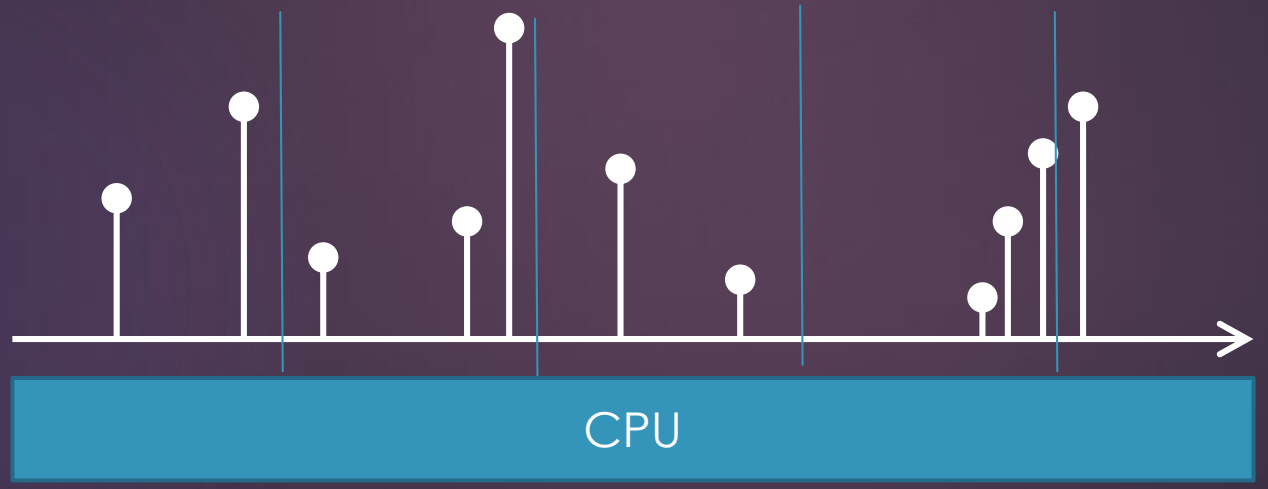
- Find all datapoints within range

Datasource Architecture

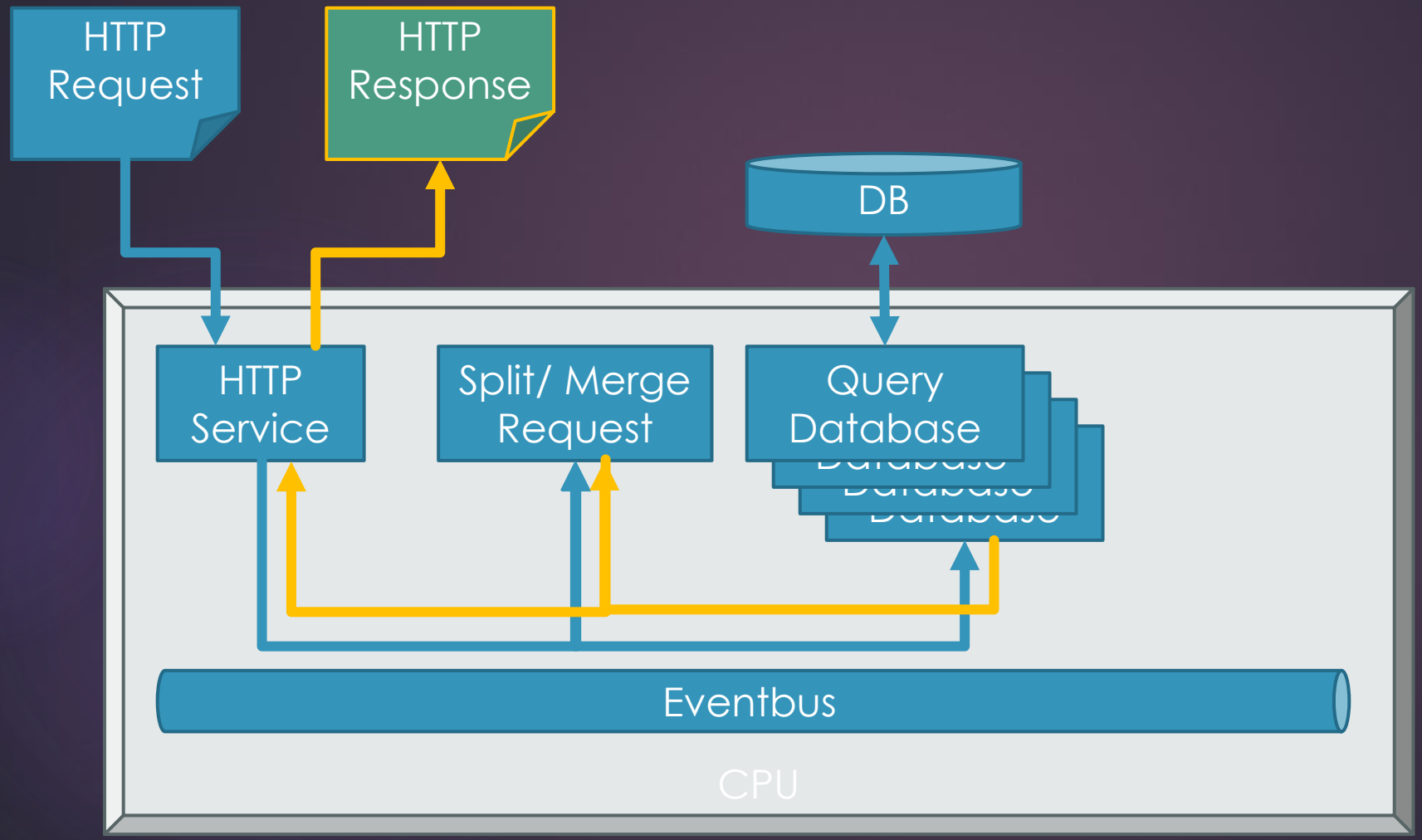


Step 2 – Split Request

- Split request into chunks (#chunks = #cores)
- Use multiple Verticle Instance in parallel (#instances = #cores) ?

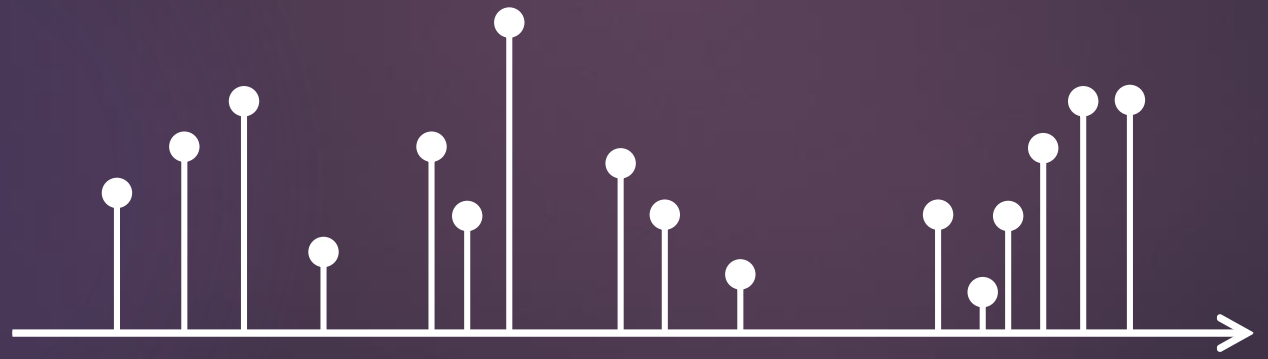


Datasource Architecture

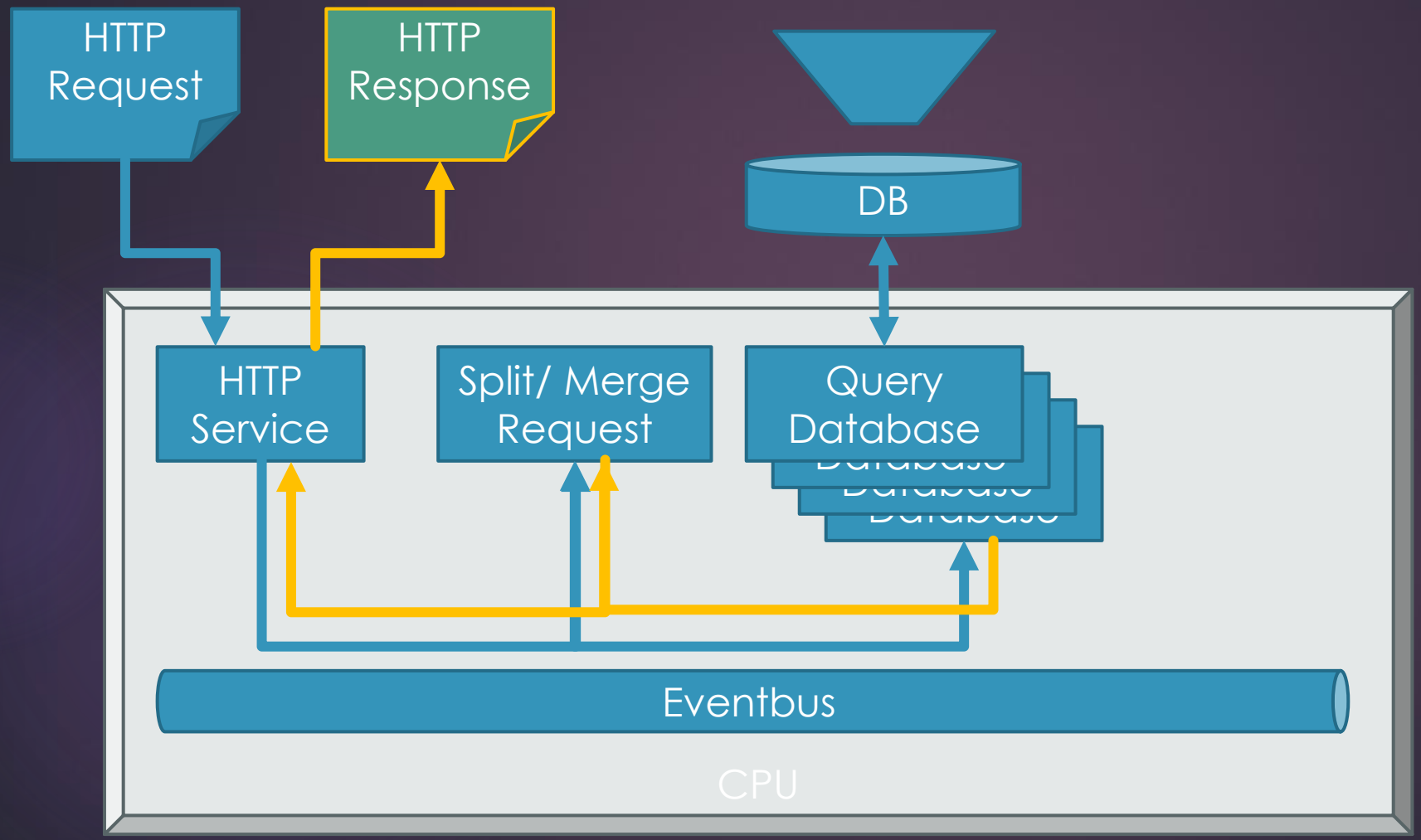


Step 3 – Aggregate Datapoints

- Use Mongo Aggregation Pipeline
- Reduce Datapoints returned to service

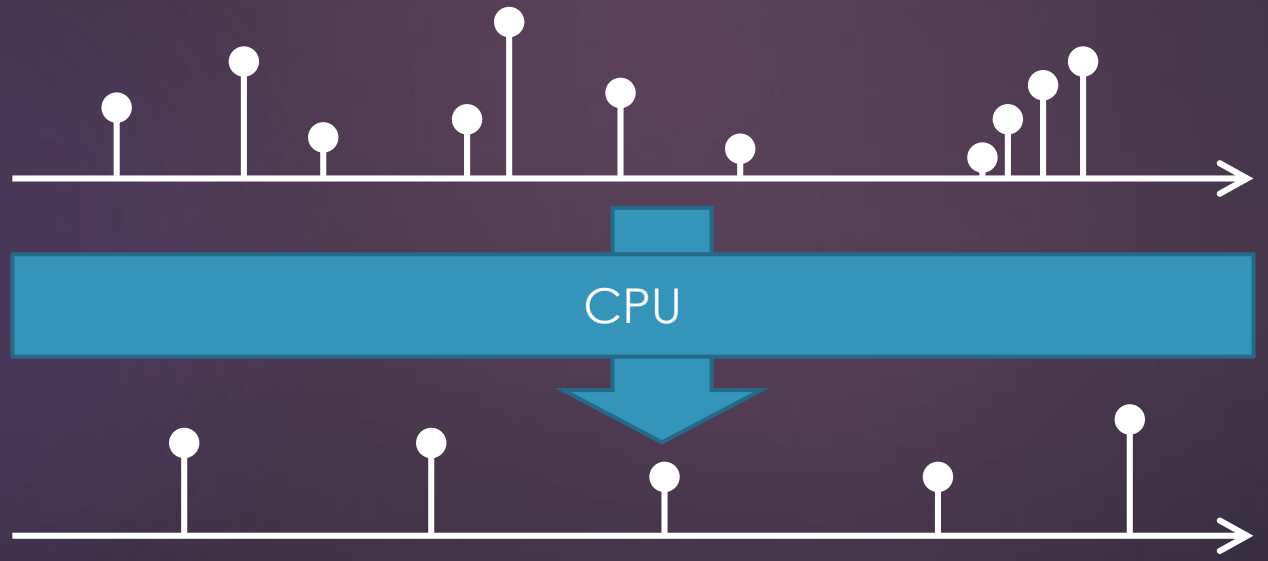


Datasource Architecture



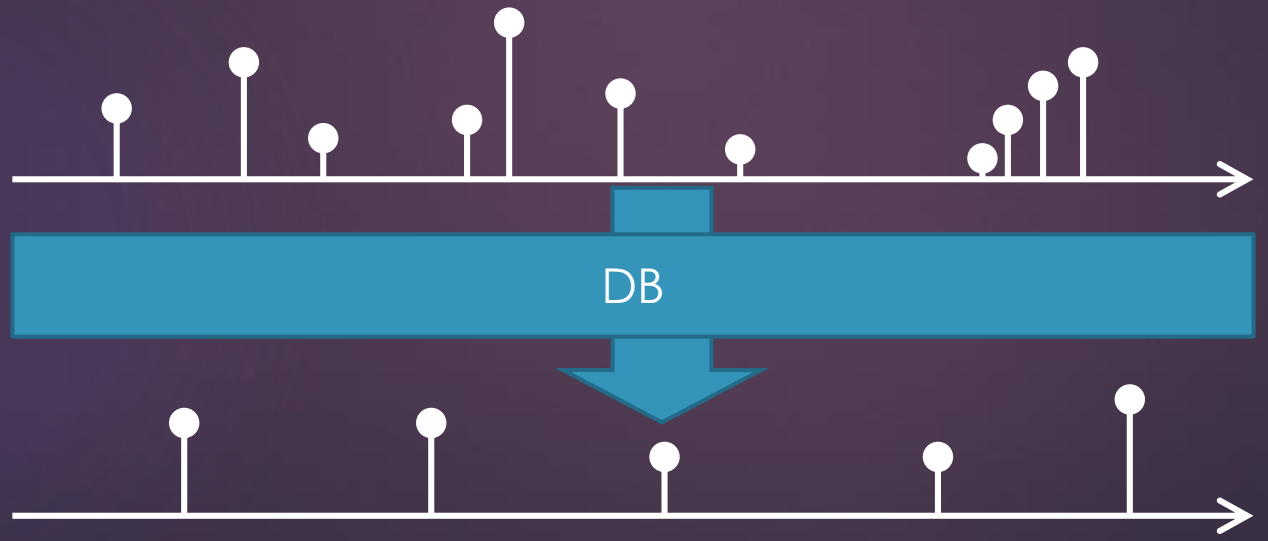
Step 4 – Percentiles (CPU)

- Fetch all data
- Calculate percentiles in service

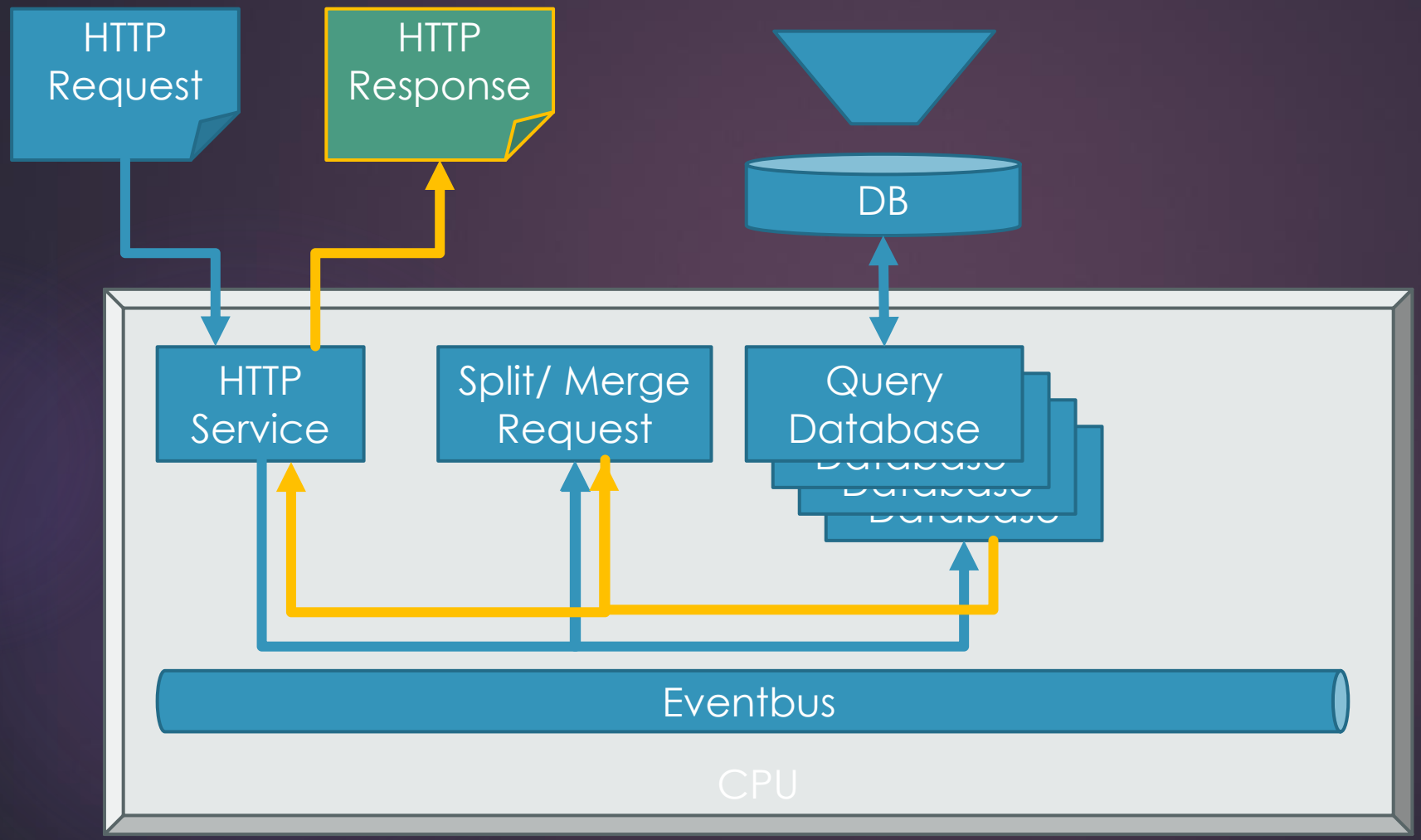


Step 4 – Percentiles (DB)

- Build aggregation pipeline to calculate percentiles
- Algorithm, see http://www.dummies.com/education/math/statistics/how-to-calculate-percentiles-in-statistics/

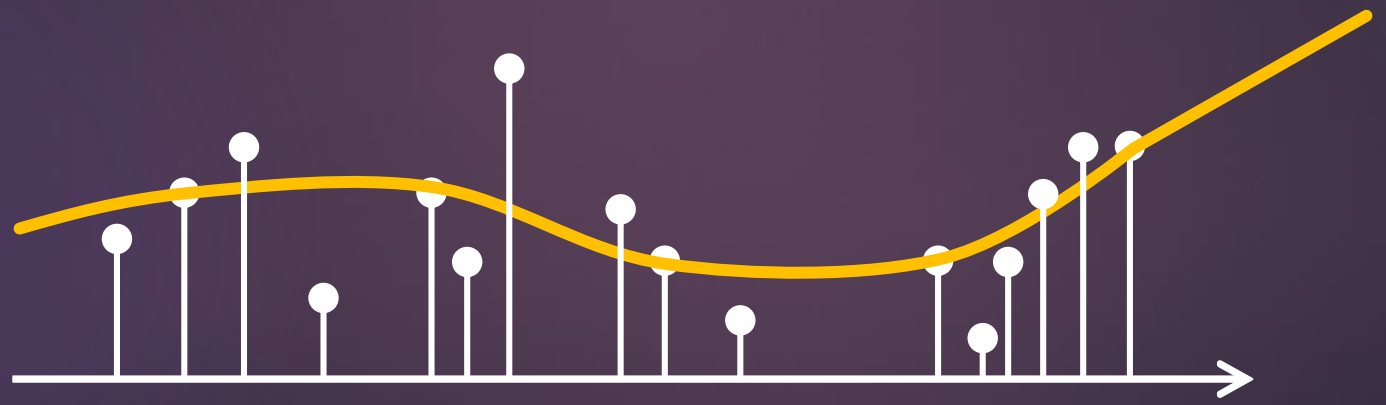


Datasource Architecture

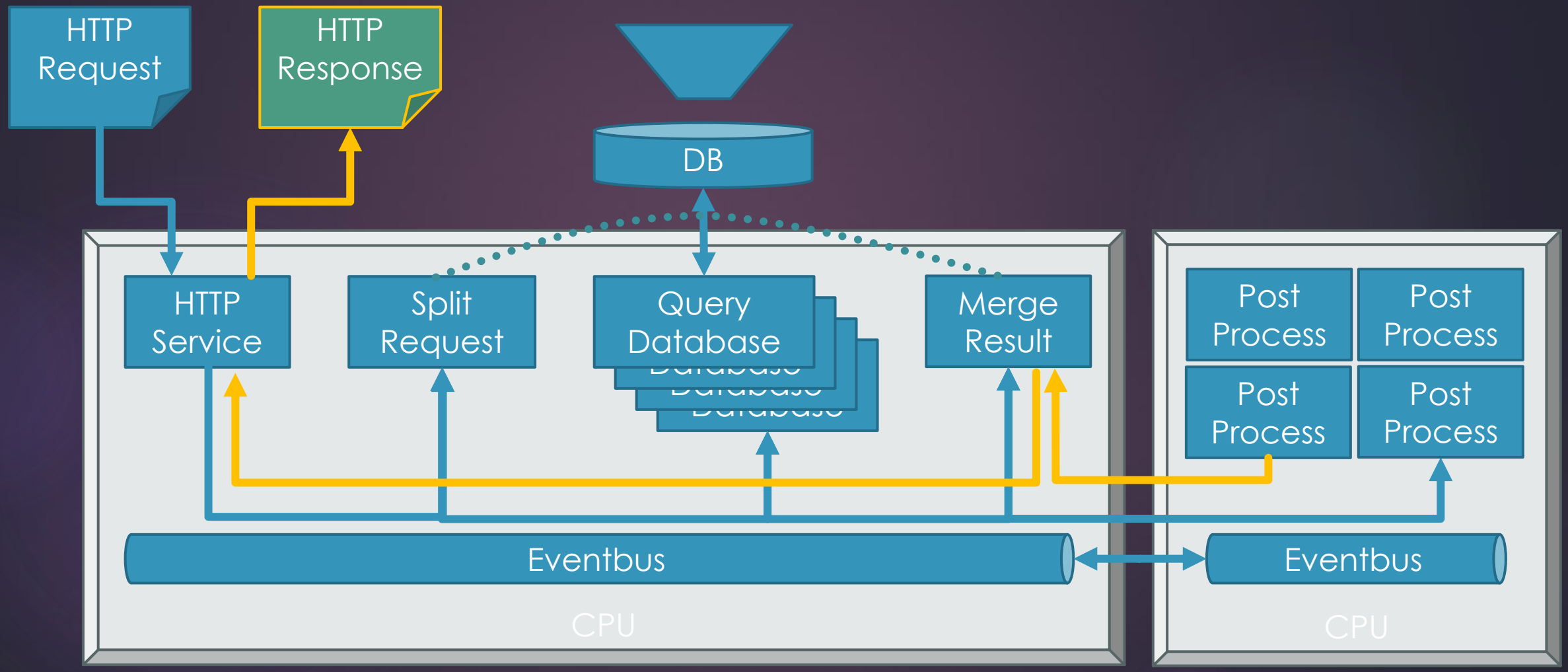


Step 5 - Postprocessing

- Apply additional computation on the result from the database

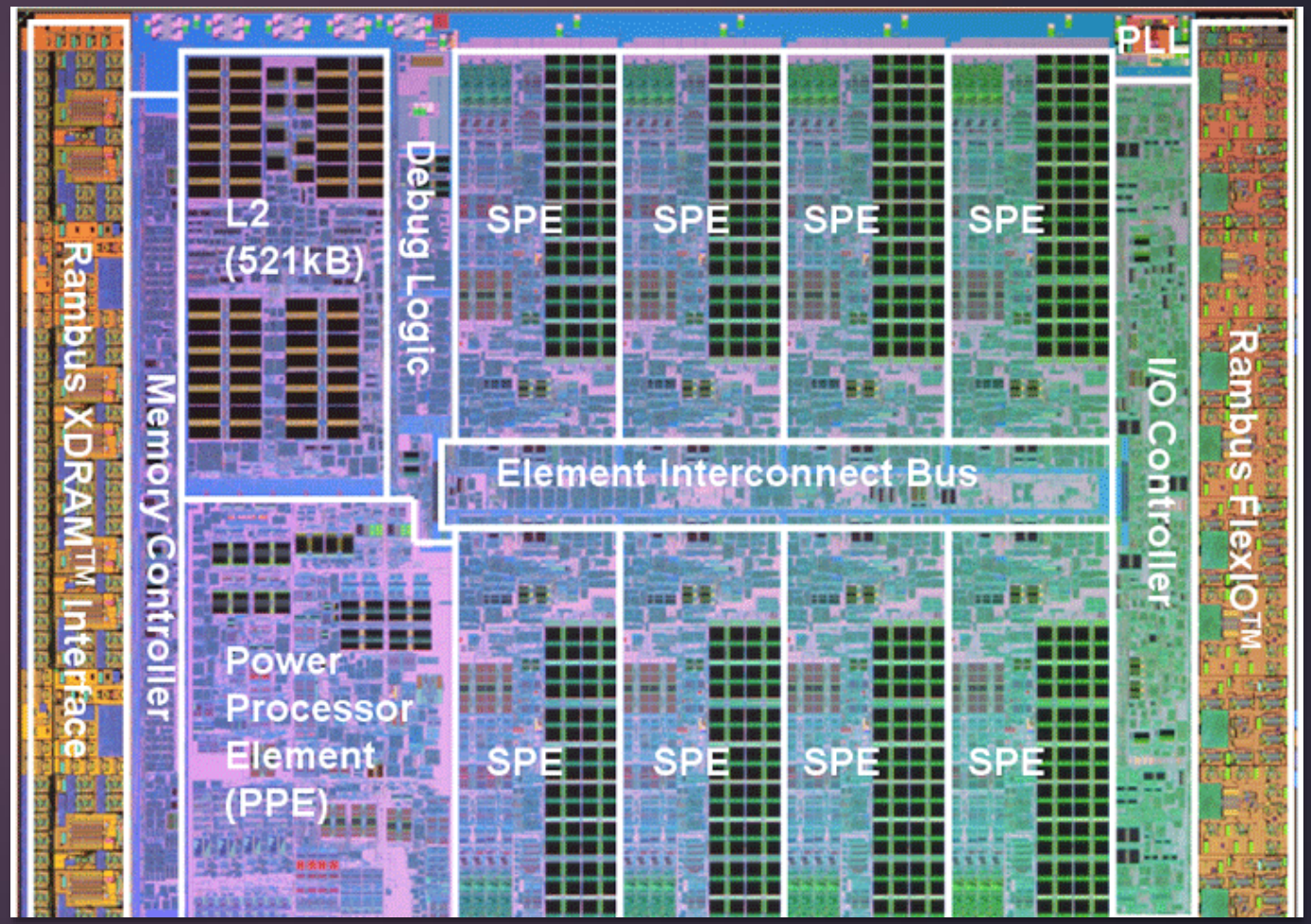


Datasource Architecture (final)



Anyone recognize this chip?

- CPU of the PS3 (Cell BE)
- Central Bus (EIB)
- 1 General Purpose CPU
  - Runs the Game (Event) Loop
- 8 SPUs
  - Special Processing
  - Sound
  - Physics
  - AI
  - ...
- 230 GFLOPS (vs 103 GFLOPS PS4)



Adding more stats & calculation

- Push Calculation to DB if possible
- Add more workers / node for complex (post-) processing
- Aggregate results before post-processing
- DB performance is king

Custom vs. timeseries DB

Custom:

- No migration of existing data
  - No redundant data storage
- More flexibility
  - Better extensibility
  - Custom views
  - Custom aggregation
- More options
  - scalability
  - retention

Timeseries DB:

- Better out-of-the-box
  - experience / performance
  - integration
  - functionality
- Built-in retention policies
- Built for scalability

Takeaways

- Grafana is powerful tool for visualizing data
  - Information becomes consumable through visualization
  - Information is essential for decision making
- Vert.x is
  - Reactive, Non-Blocking, Asynchronous, Scalable
  - Running on JVM
  - Polyglott
  - Fun

Source code on:
https://github.com/gmuecke/grafana-vertx-datasource

Still hungry?

FOR GEEK STUFF

Let's read a large data file

- Datafile is large (> 1GB)
- Every line of the file is a datapoint
- The first 10 characters are a timestamp
- The dataset is sorted
- The datapoints are not equally distributed

- Grafana requires reads ~1900 datapoints per chart request

The Challenges (pick one)

- How to randomly access 1900 datapoints without reading the entire file into memory?

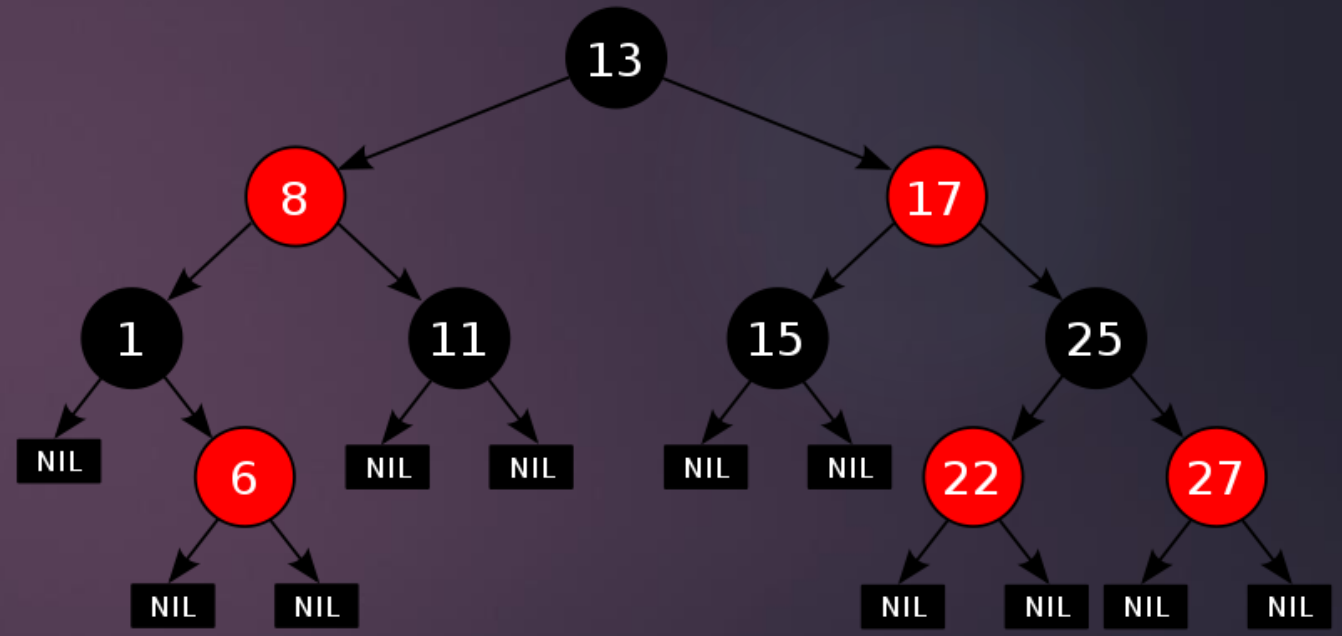
Index
+ Lazy refinement

- How to read a huge file efficiently into memory?

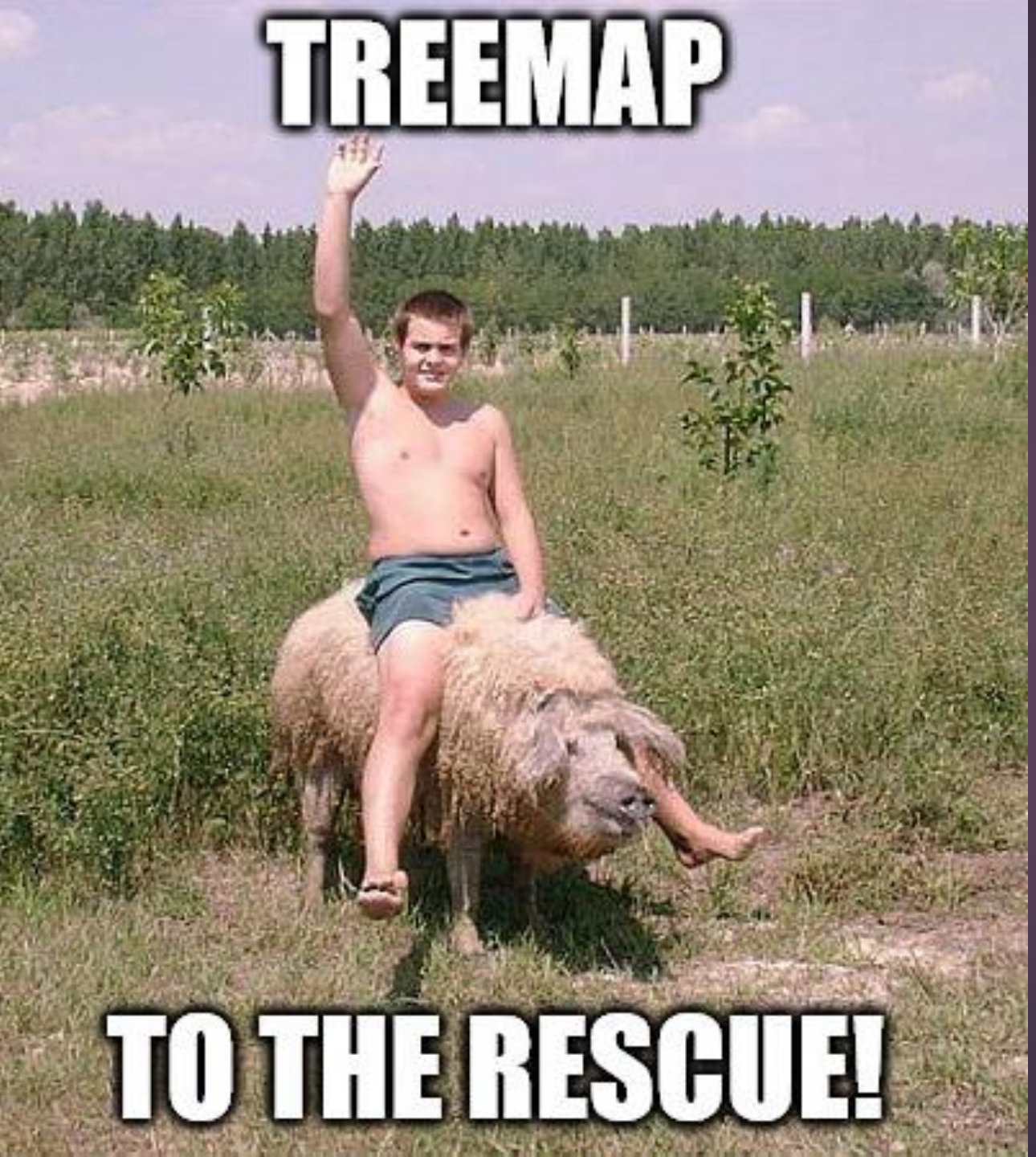
Index
+ Lazy load

Let's build an index

- Indexes can be build using a tree-datastructure
  - Node: Timestamp
  - Leaf: offset position in file
    or the datapoint
- Red-Black Trees provide fast access
  - Read/Insert O(log n)
  - Space n



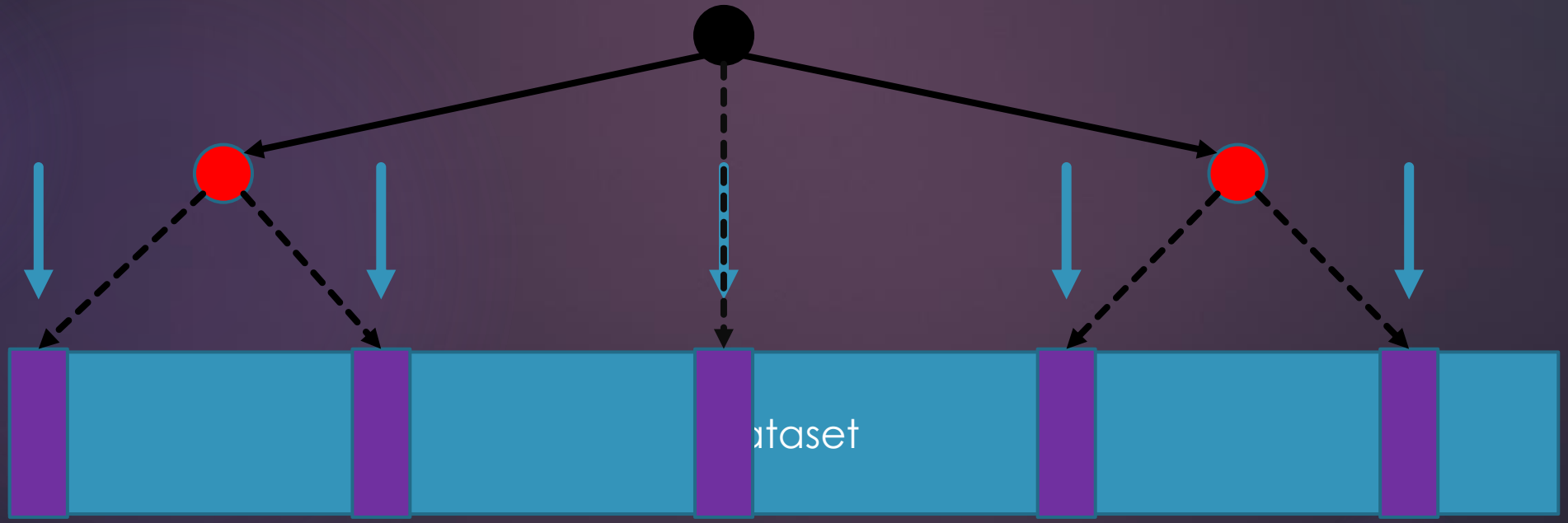

YOU WANT ME TO IMPLEMENT
A RED-BLACK TREE?
imgflip.com



- java.util.TreeMap is a red-black tree based implementation*
- `TreeMap<Long,Long> index = new TreeMap<>();`
- *actually all HashMaps are

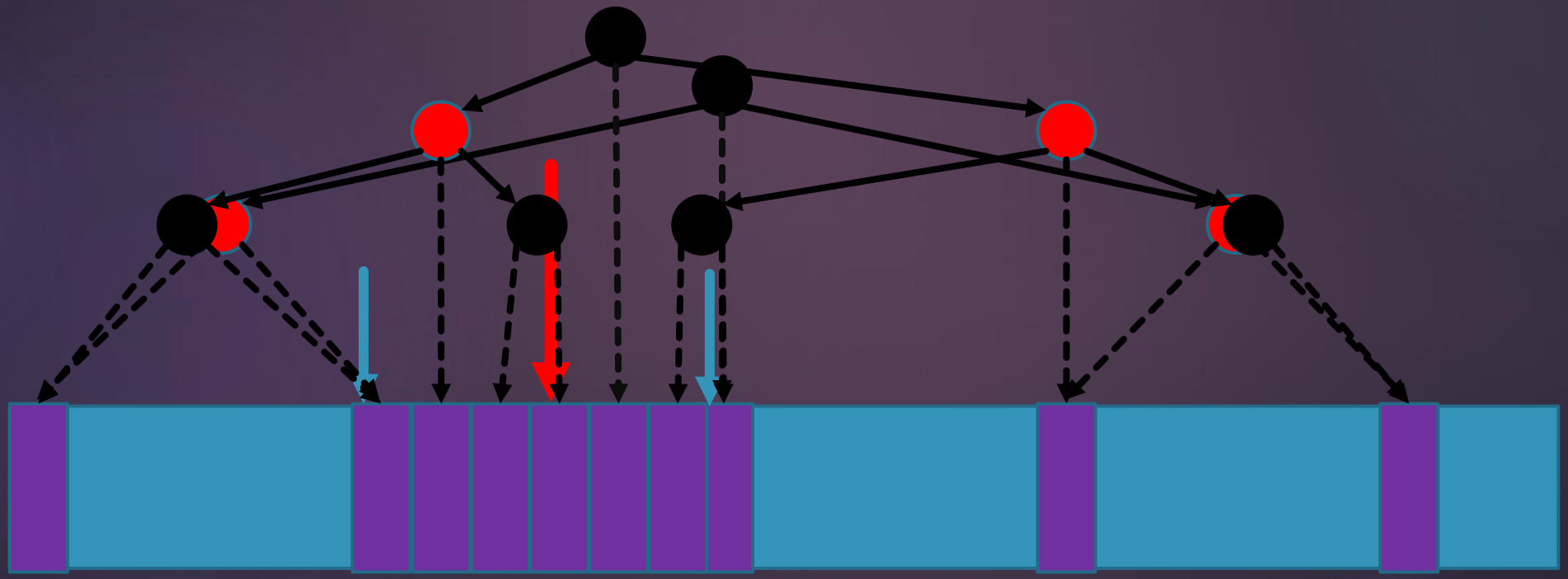
How to build an index (fast)?

- Read datapoint from offset positions
- Build a partial index

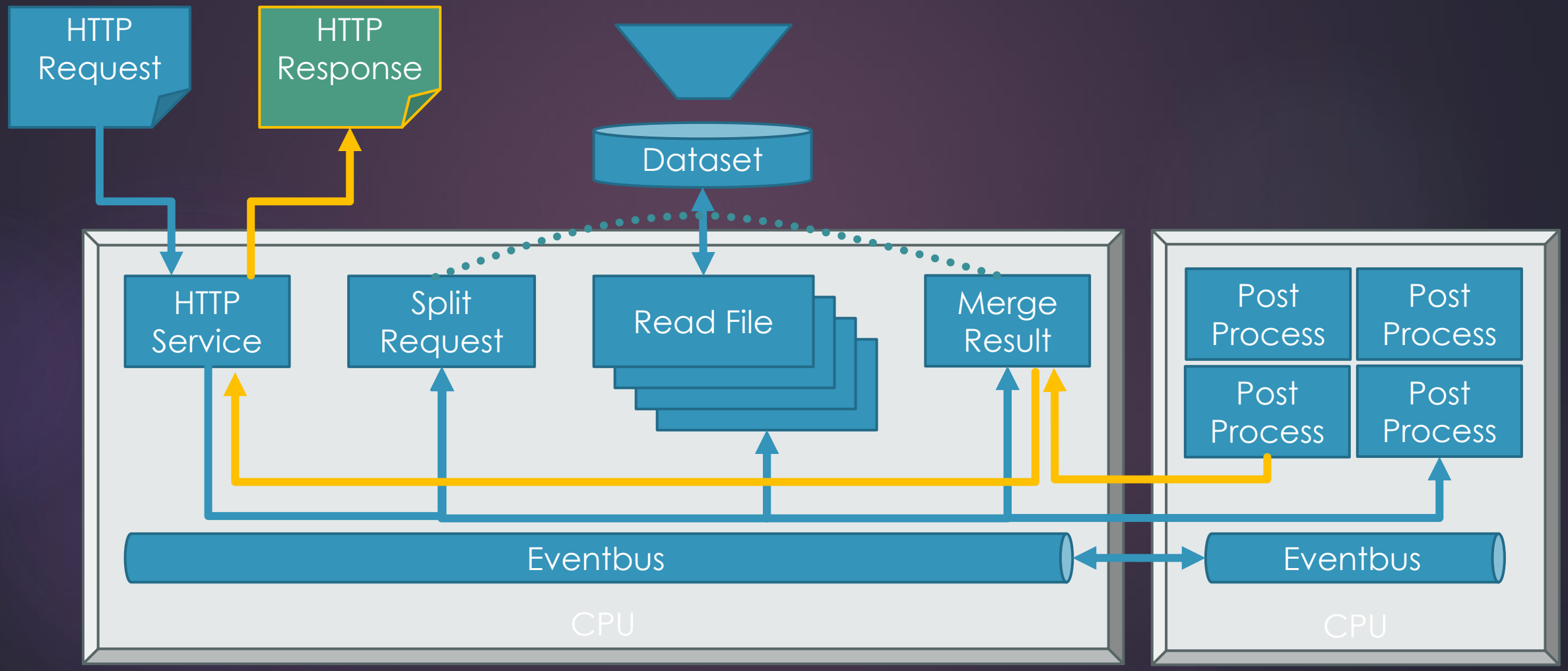


On next query

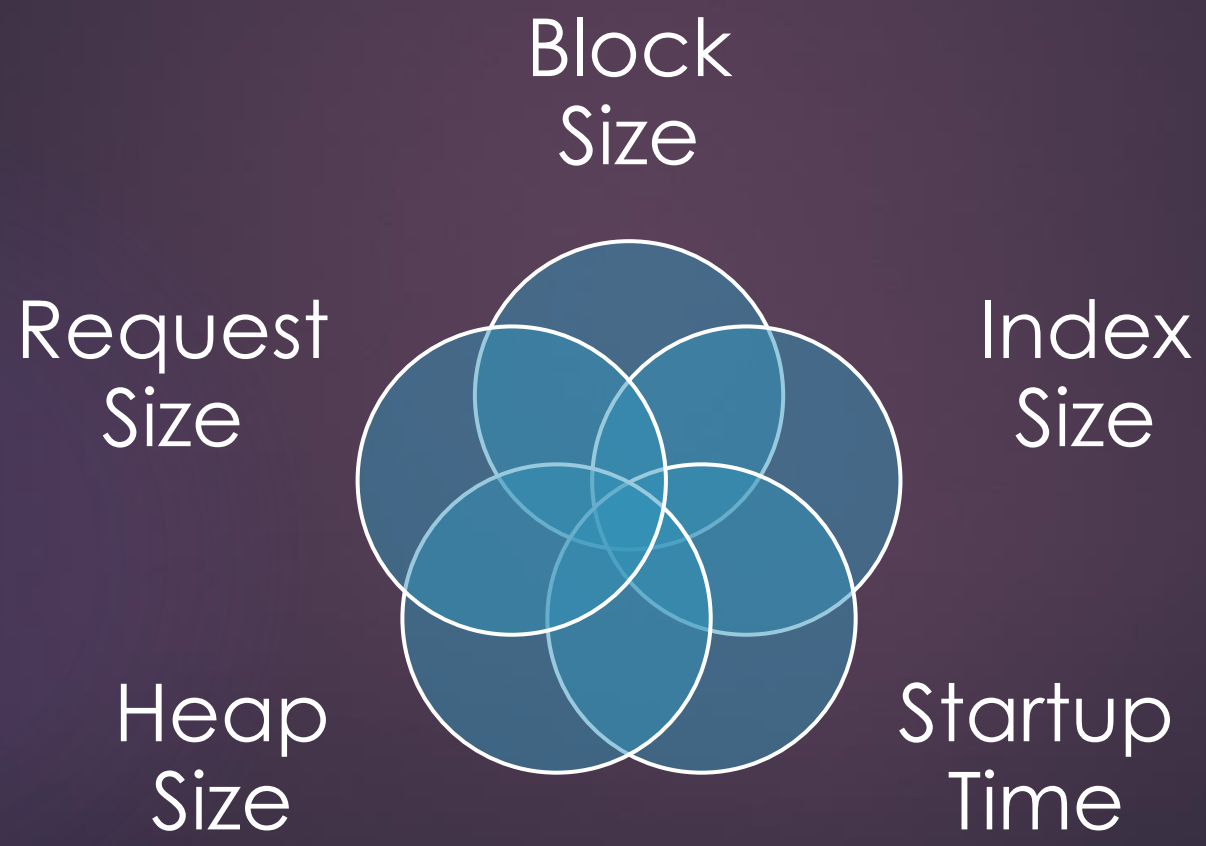
- Locate Block
- Refine Block
- Update Index



Datasource Architecture (again)



Tradeoffs



Thank you!

FEEDBACK APRECIATED!