Why every gopher should be a data scientist.

Ivan Danyliuk, Golang BCN June Meetup
27 June 2017, Barcelona

The recent study from MIT has found…

...there's an 87% chance
Linus Torvalds hates your code.

"Bad programmers **worry about the code**. Good programmers **worry about data structures** and their relationships."

"Show me your [code] and conceal your [data structures], and **I shall continue to be mystified**.

Show me your [data structures], and I won't usually need your [code]; **it'll be obvious**."

Fred Brooks

LEGEND

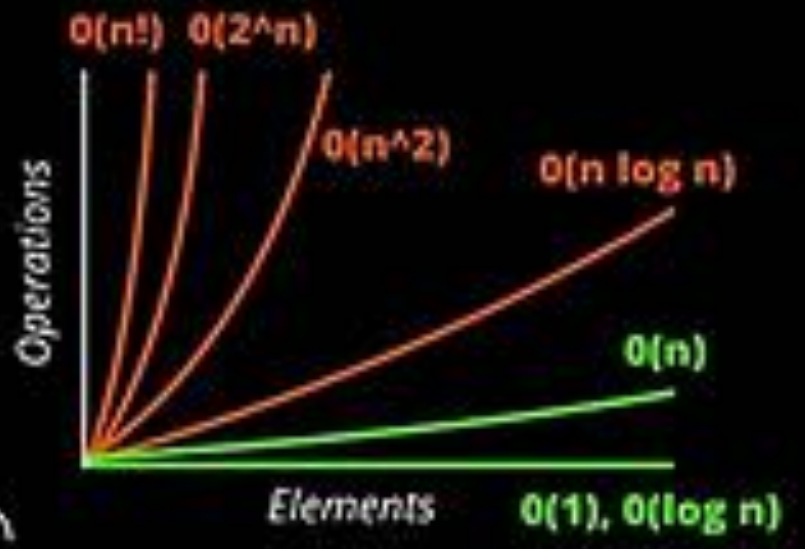
TIME Complexity vs. **SPACE** Complexity

Good Fair Bad

Good Fair Bad

<BIG-O-CHEATSHEET>

</>



www.bigocheatsheet.com

DATA STRUCTURE
Operations

DATA Structure	TIME Complexity								SPACE Complexity
	Average				Worst				Worst
	Access	Search	Insertion	Deletion	Access	Search	Insertion	Deletion	
Array	Θ(1)	Θ(n)	Θ(n)	Θ(n)	O(1)	O(n)	O(n)	O(n)	O(n)
Stack	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Queue	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Singly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Doubly-Linked List	Θ(n)	Θ(n)	Θ(1)	Θ(1)	O(n)	O(n)	O(1)	O(1)	O(n)
Skip List	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n log(n))
Hash Table	N/A	Θ(1)	Θ(1)	Θ(1)	N/A	O(n)	O(n)	O(n)	O(n)
Binary Search Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)
Cartesian Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(n)	O(n)	O(n)	O(n)
B-Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Red-Black Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
Splay Tree	N/A	Θ(log(n))	Θ(log(n))	Θ(log(n))	N/A	O(log(n))	O(log(n))	O(log(n))	O(n)
AVL Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(log(n))	O(log(n))	O(log(n))	O(log(n))	O(n)
KD Tree	Θ(log(n))	Θ(log(n))	Θ(log(n))	Θ(log(n))	O(n)	O(n)	O(n)	O(n)	O(n)

ARRAY SORTING
Algorithms

ARRAY Algorithms	TIME Complexity			SPACE Complexity
	Best	Average	Worst	Worst
Quicksort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(log(n))
Mergesort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(n)
Timsort	Ω(n)	Θ(n log(n))	O(n log(n))	O(1)
Heapsort	Ω(n log(n))	Θ(n log(n))	O(n log(n))	O(1)
Bubble Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Insertion Sort	Ω(n)	Θ(n^2)	O(n^2)	O(1)
Selection Sort	Ω(n^2)	Θ(n^2)	O(n^2)	O(1)
Tree Sort	Ω(n log(n))	Θ(n log(n))	O(n^2)	O(n)
Shell Sort	Ω(n log(n))	Θ(n(log(n))^2)	O(n(log(n))^2)	O(1)
Bucket Sort	Ω(n+k)	Θ(n+k)	O(n^2)	O(n)
Radix Sort	Ω(nk)	Θ(nk)	O(nk)	O(n+k)
Counting Sort	Ω(n+k)	Θ(n+k)	O(n+k)	O(k)
Cubesort	Ω(n)	Θ(n log(n))	O(n log(n))	O(n)

Question

- Raise your hand if you have designed your own non-standard data-structure recently?
- Raise your hand if you have implemented your own custom algorithm recently?

Question

- Now, raise your hand if you have written a new microservice recently?

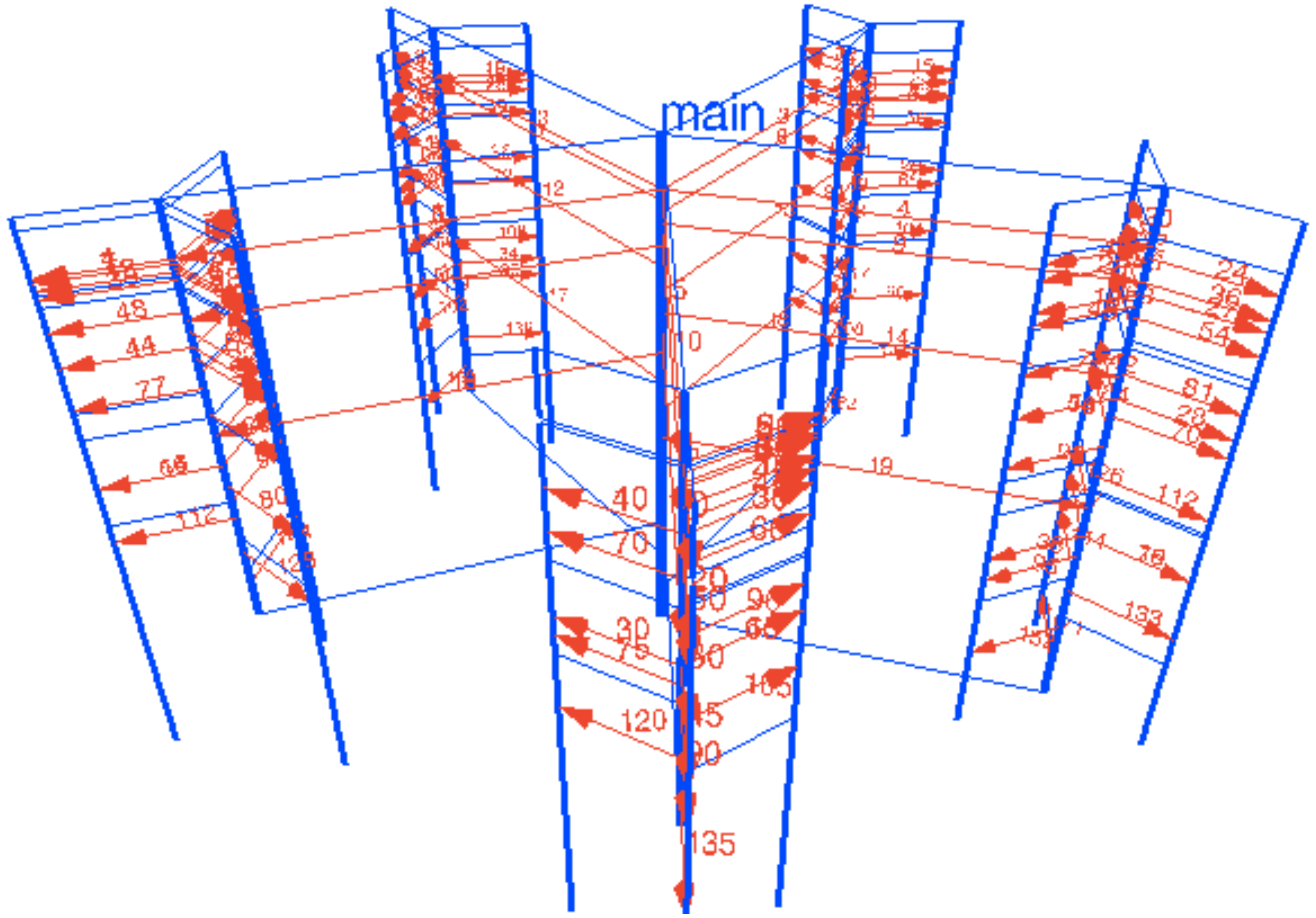
Our "programs" now are distributed systems.

Our "hardware" is a cloud.

Algorithms -> Programs -> Distributed systems

CSP

- Hardware chips CSP implementations
- Software CSP frameworks/languages
- Distributed systems



Calls

- Inlining function vs external function call
- Local code vs external RPC call to service
- Imagine, network call is as cheap as local function call - what's the difference then?

Algorithmia is actually already doing that

BROWSE ALL ALGORITHMS:

Colorful Image Colorization deeplearning

Colorizes given black & white images.

Summarizer nlp

Summarize english text

Sentiment Analysis nlp

Determine positive or negative sentiment from text

Auto-Tag URL tags

Automatically generate keyword tags for a URL

AutoTag nlp

Automatically extract tags from text

Nudity Detection sfw

Detect nudity in pictures

```go
package main

import (
  "fmt"
  algorithmia "github.com/algorithmiaio/algorithmia-go"
)

func main() {
  input := 1429593869

  var client = algorithmia.NewClient("YOUR_API_KEY", "")
  algo, _ := client.Algo("algo://ovi_mihai/TimestampToDate/0.1.0")
  resp, _ := algo.Pipe(input)
  response := resp.(*algorithmia.AlgoResponse)
  fmt.Println(response.Result)
}
```

1. Think about the whole system as one program

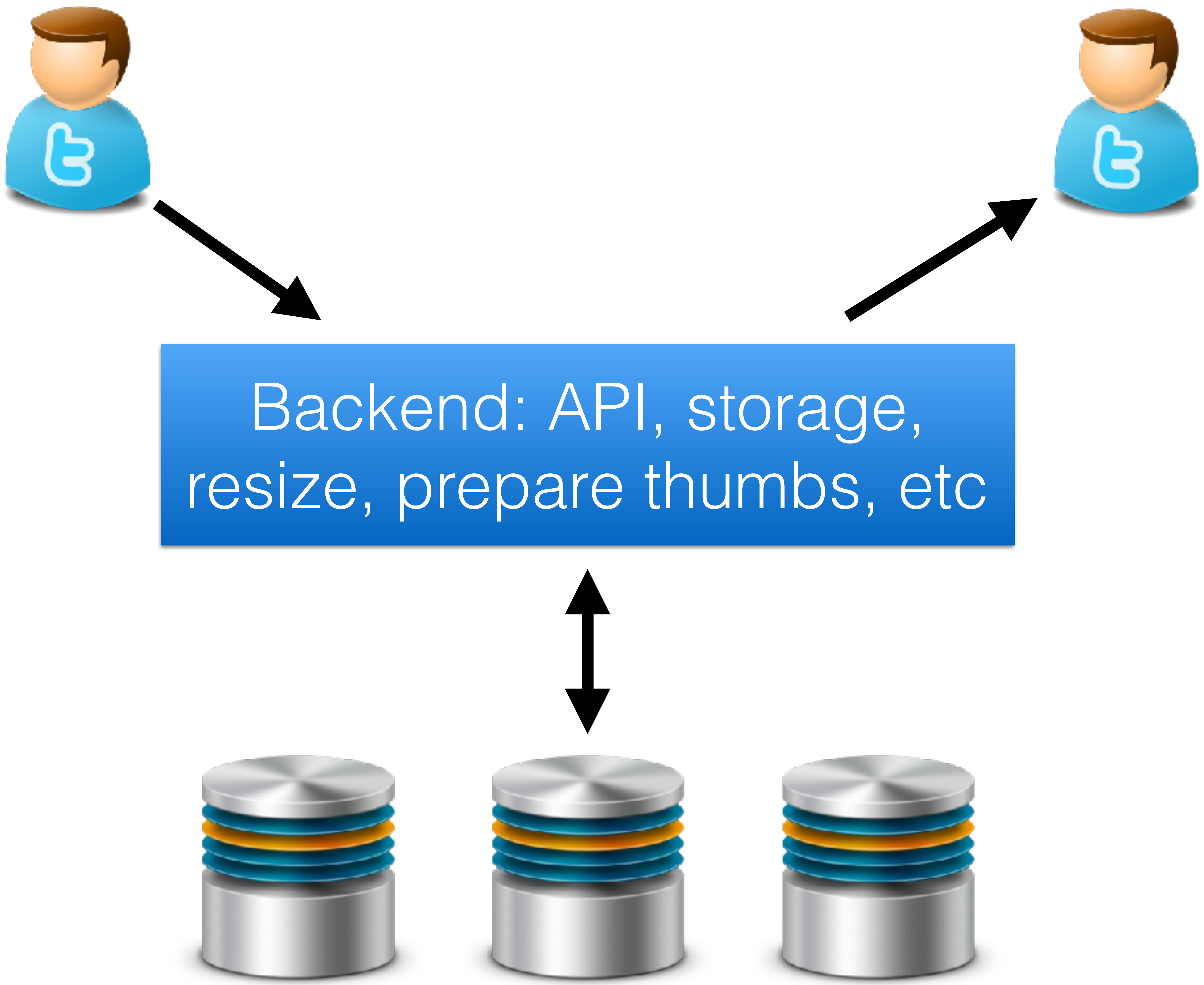
algorithms
design programs around the data
systems

Examples

Twitter story

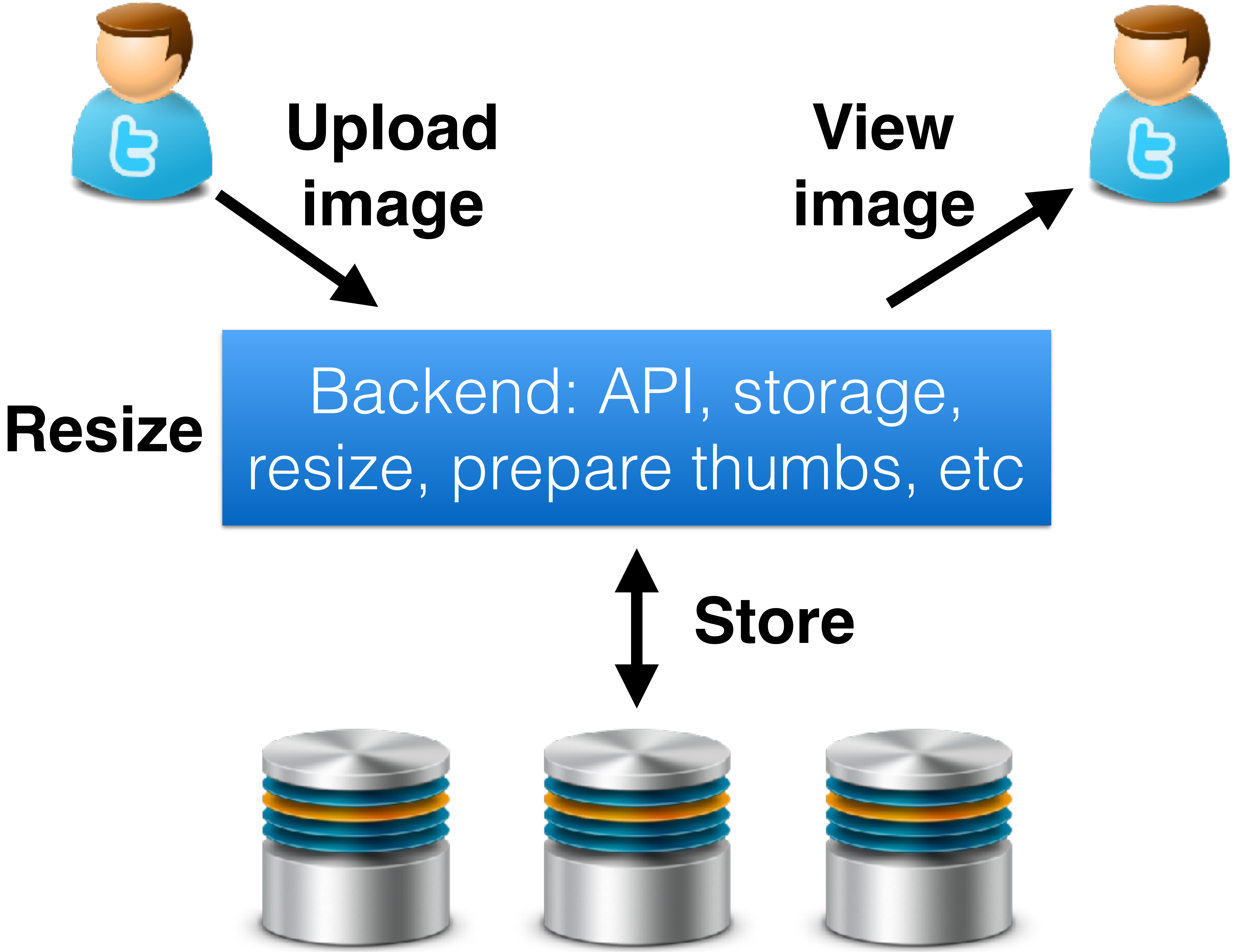
How Twitter **refactored** it's media storage system

Twitter in 2012



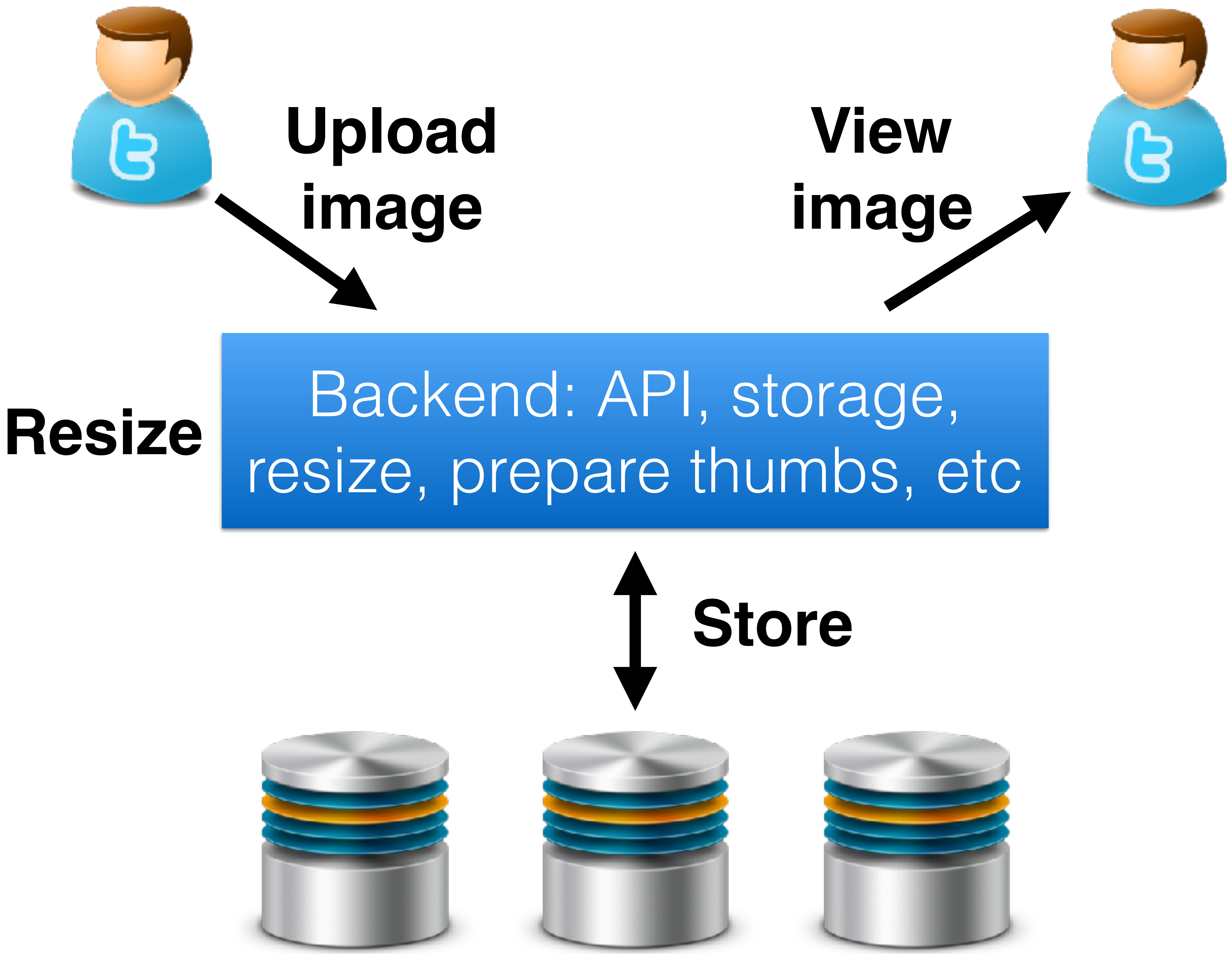
Backend: API, storage, resize, prepare thumbs, etc

Twitter in 2012



- Handle user upload
- Create thumbnails and different size versions
- Store images (write)
- Return on user request (read)

Twitter in 2012



Upload image

View image

Resize

Backend: API, storage, resize, prepare thumbs, etc

Store

- Handle user upload
- Create thumbnails and different size versions
- Store images
- Return on user request (view)

The Problem:

- a lot of storage space
- + 6TB per day

Twitter did a research on the data and found patterns of access

After the data research

- 50% of requested images are at most 15 days old
- After 20 days, probability of image being accessed is really low

Twitter in 2016

- Introduced a CDN Origin Server called MinaBird, which can do resizes on-the-fly
- Slow, but it's a good space-time tradeoff.
- Image variants kept only 20 days.
- Images older than 20 days were resized by MinaBird on the fly.

Twitter in 2016

- Storage usage **dropped by 4TB per day**
- **Twice as less** of computing power
- Saved **$6 million** in 2015
- Just by looking at data and usage patterns

One of my former employers story

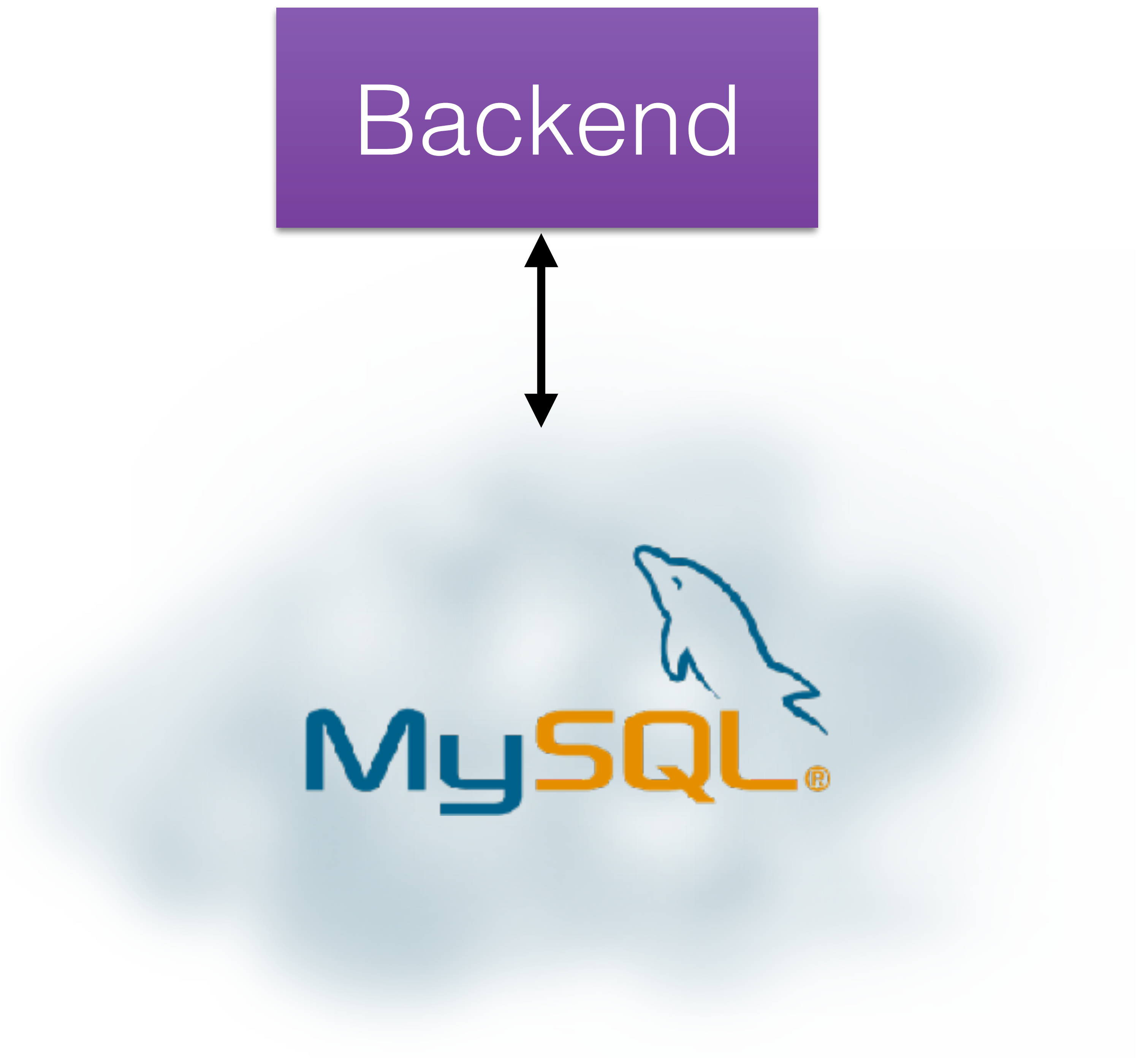
How to ignore data

The problem

- Similar to twitter, they had users writing and reading stuff
- Stuff had to be filtered and searched
- It became slow as the company size grew

Design

- All data was stored in single MySQL instance
- Hundreds of millions of records
- Go backend - simply a proxy to DB
- DB became a bottleneck

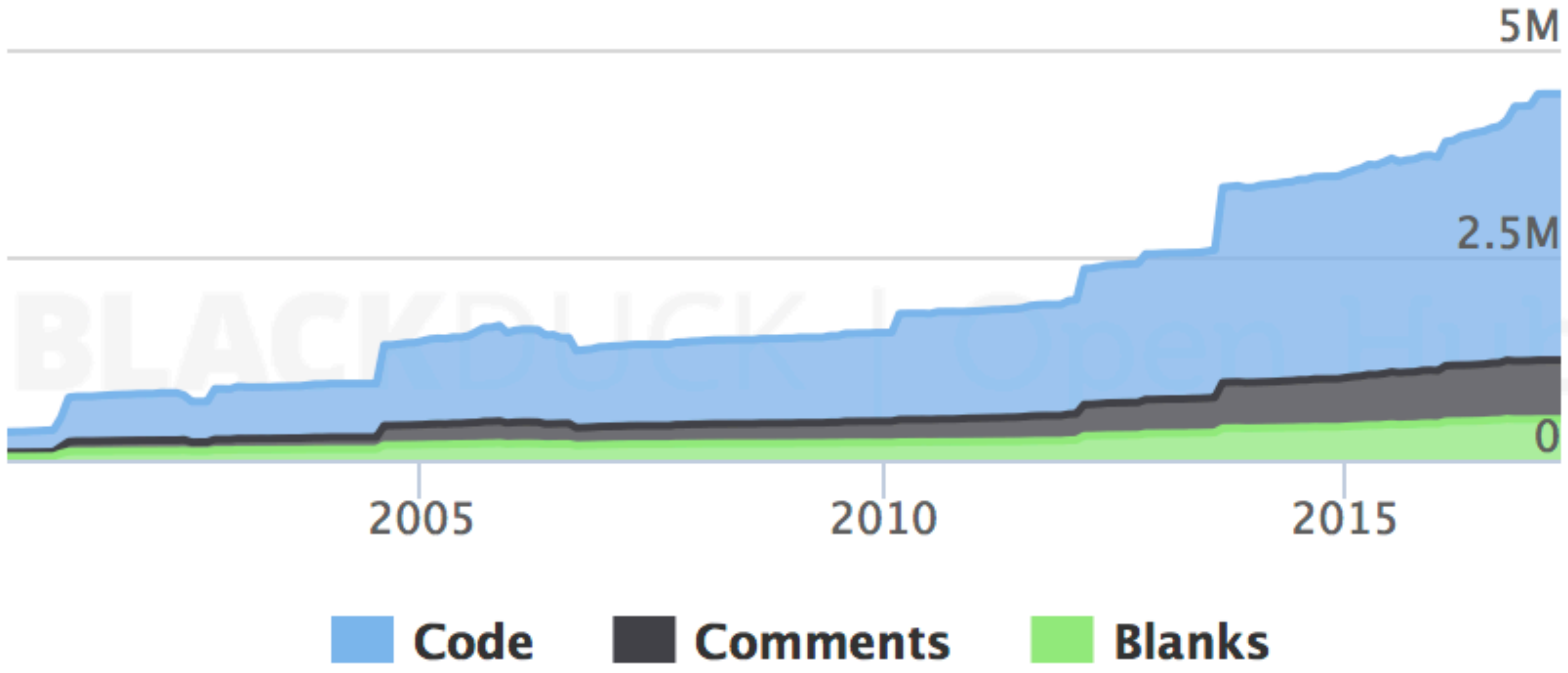


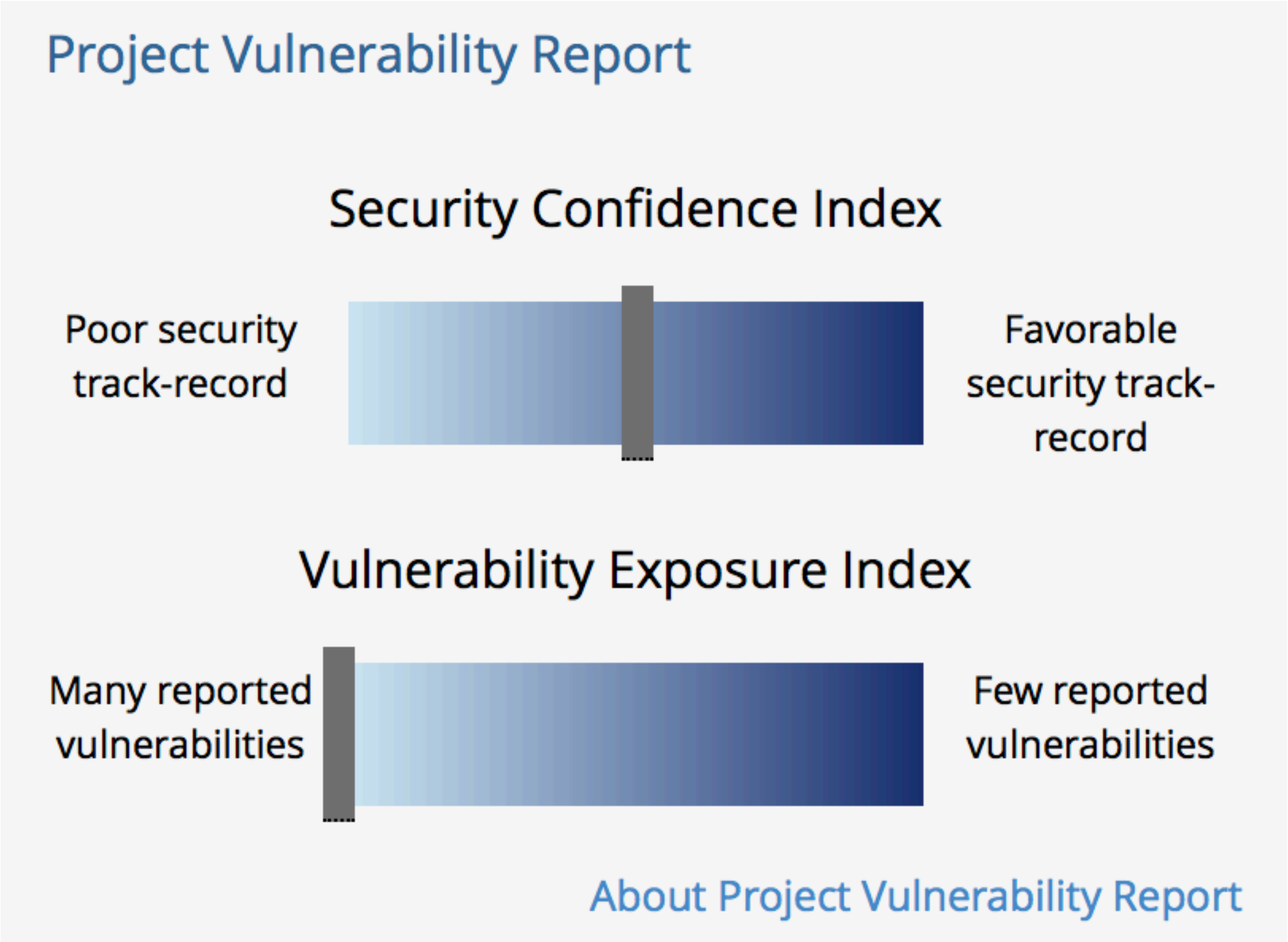
MySQL

MySQL is more than 4.5 million Lines of Code

Code

Lines of Code





Project Vulnerability Report

Security Confidence Index

Poor security track-record

Favorable security track-record

Vulnerability Exposure Index

Many reported vulnerabilities

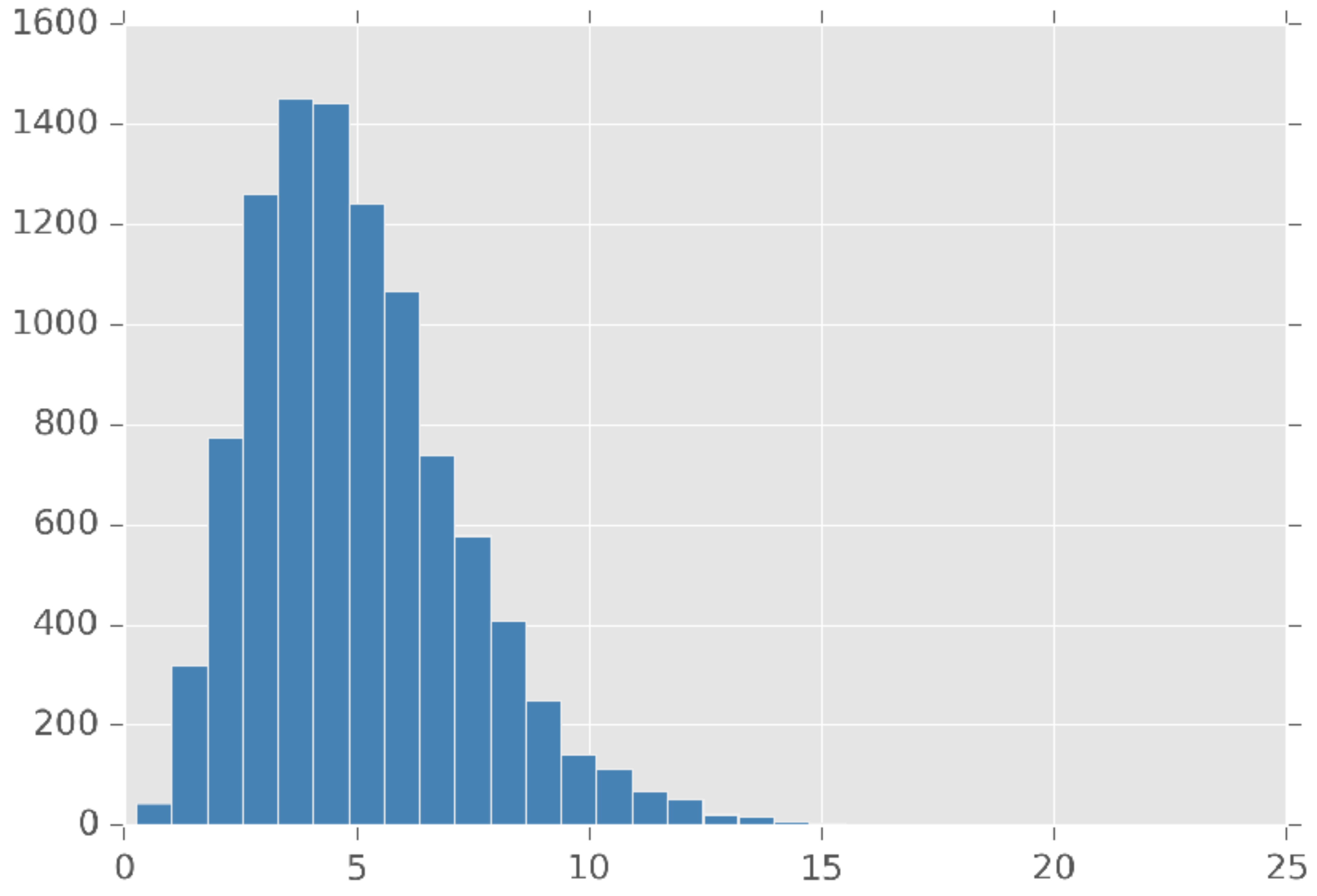
Few reported vulnerabilities

About Project Vulnerability Report

Then we made a thorough data research...

and found two things:

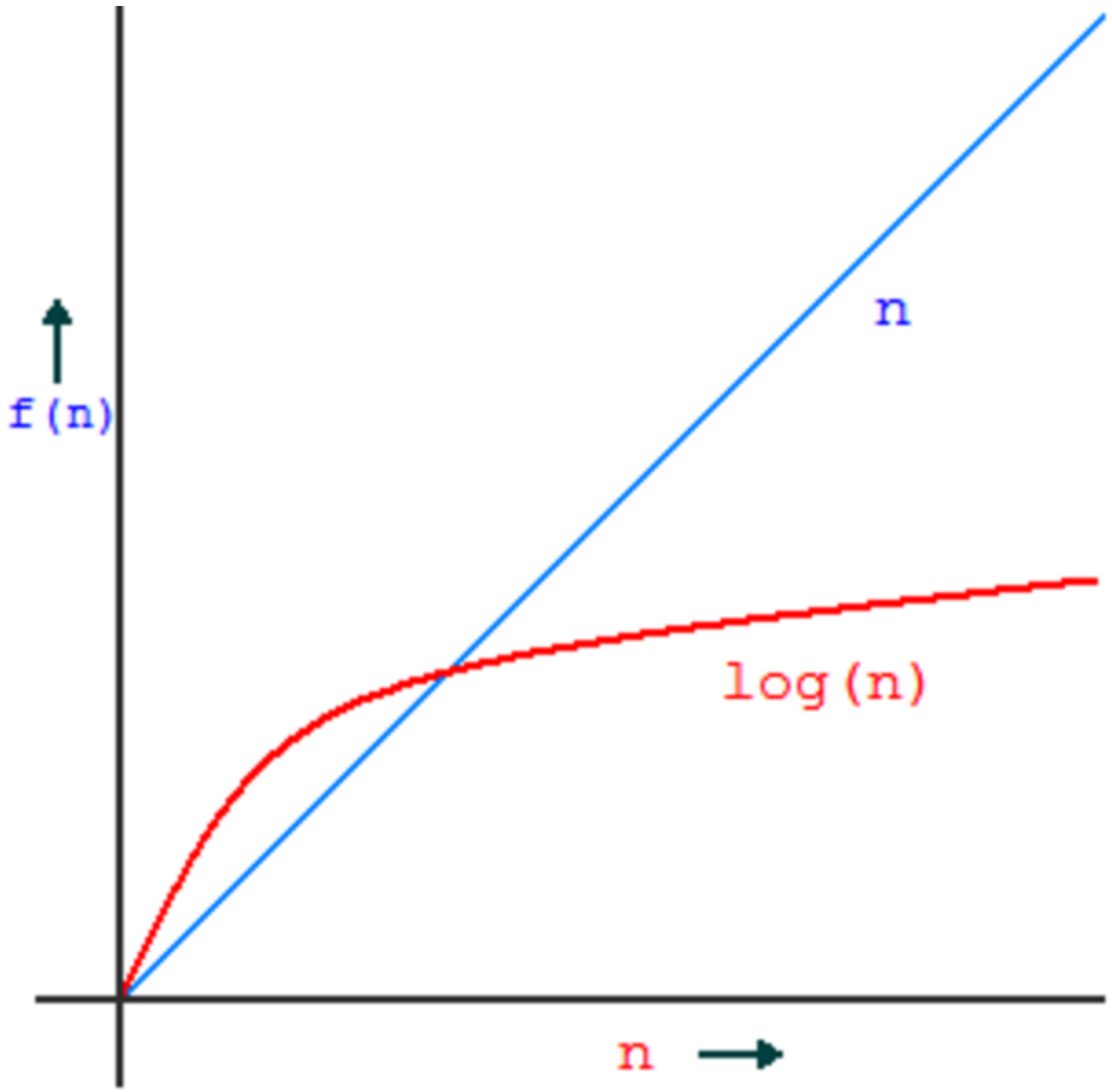
- most (90%) of the data was really small
- basically, 95p is 10x10 table of strings
- searching that >1s didn't make any sense

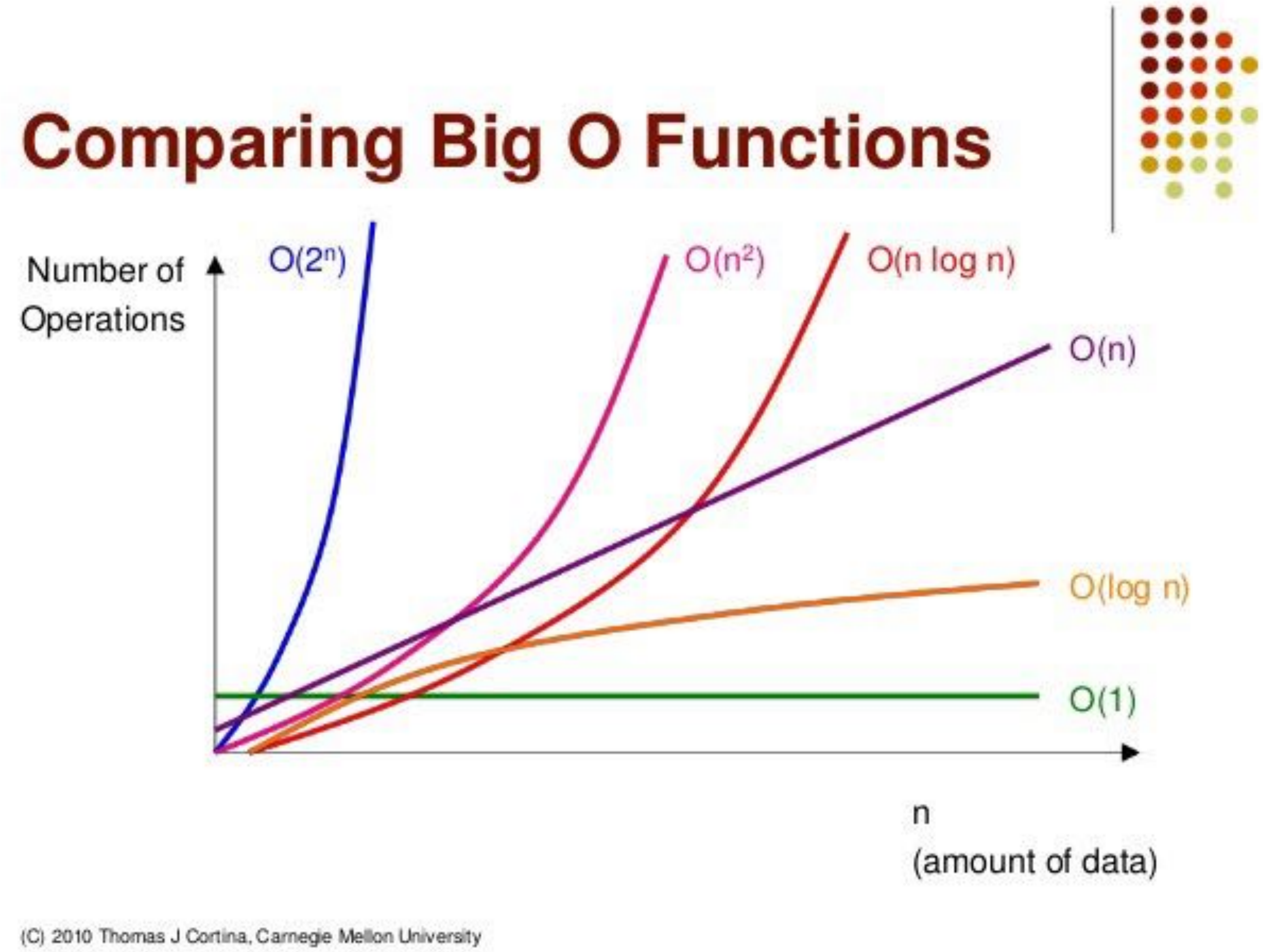


Property of the small numbers

- Linear search can outperform binary search if N is small
- Cumulative costs of indexing, storing, making network request, doing complex search, returning the answers are higher than naive for loop in memory if $N < 10$

Property of the small numbers





(C) 2010 Thomas J Cortina, Carnegie Mellon University

Data retention

- On top of that, data research found very strong usage access pattern similar to twitter's one
- Most of the data was a "dead weight" after two weeks
- And there was a good evidence that it's not going to change

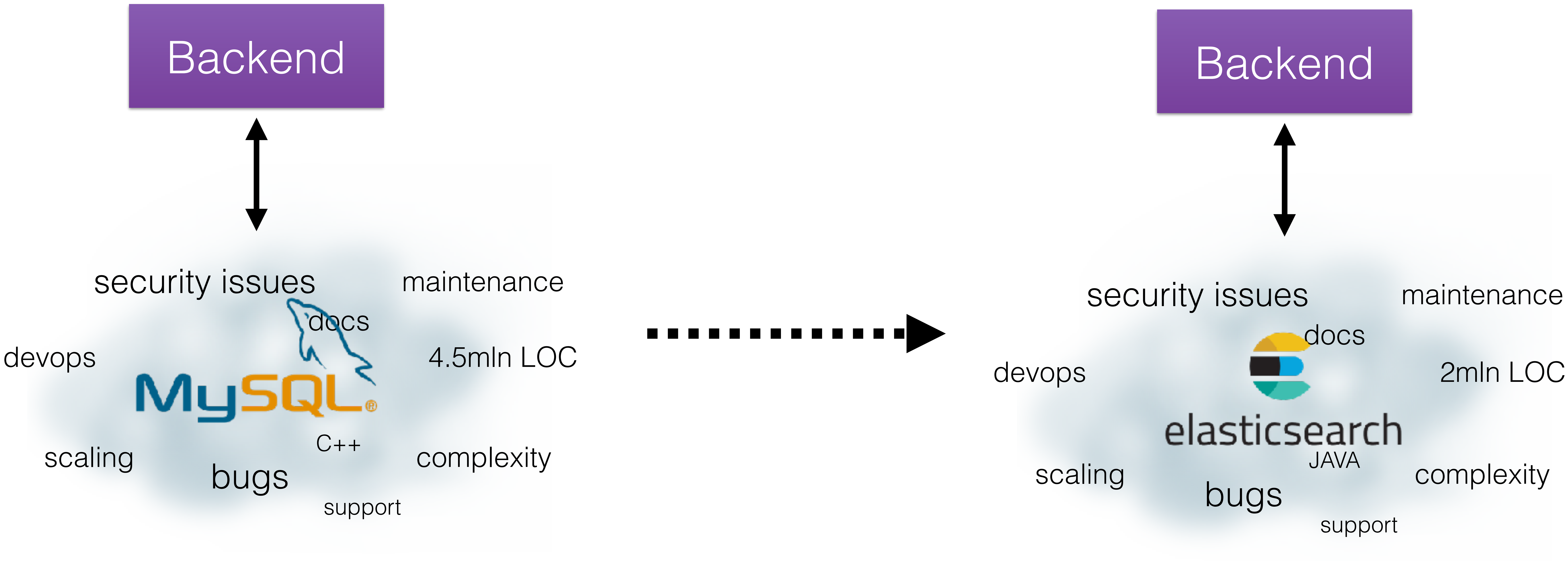
Solution

- Than opens up possibility to a lot of unique trade-offs
    - Use slower & cheaper storage for the old data
    - Pre-load new data and perform search/filtering in the memory
- Scales nicely - just add more servers + consistent hash load-balancing (each user had unique ID)

But, company decided to solve DB problem...

...by switching to another DB.

Outcome



Backend

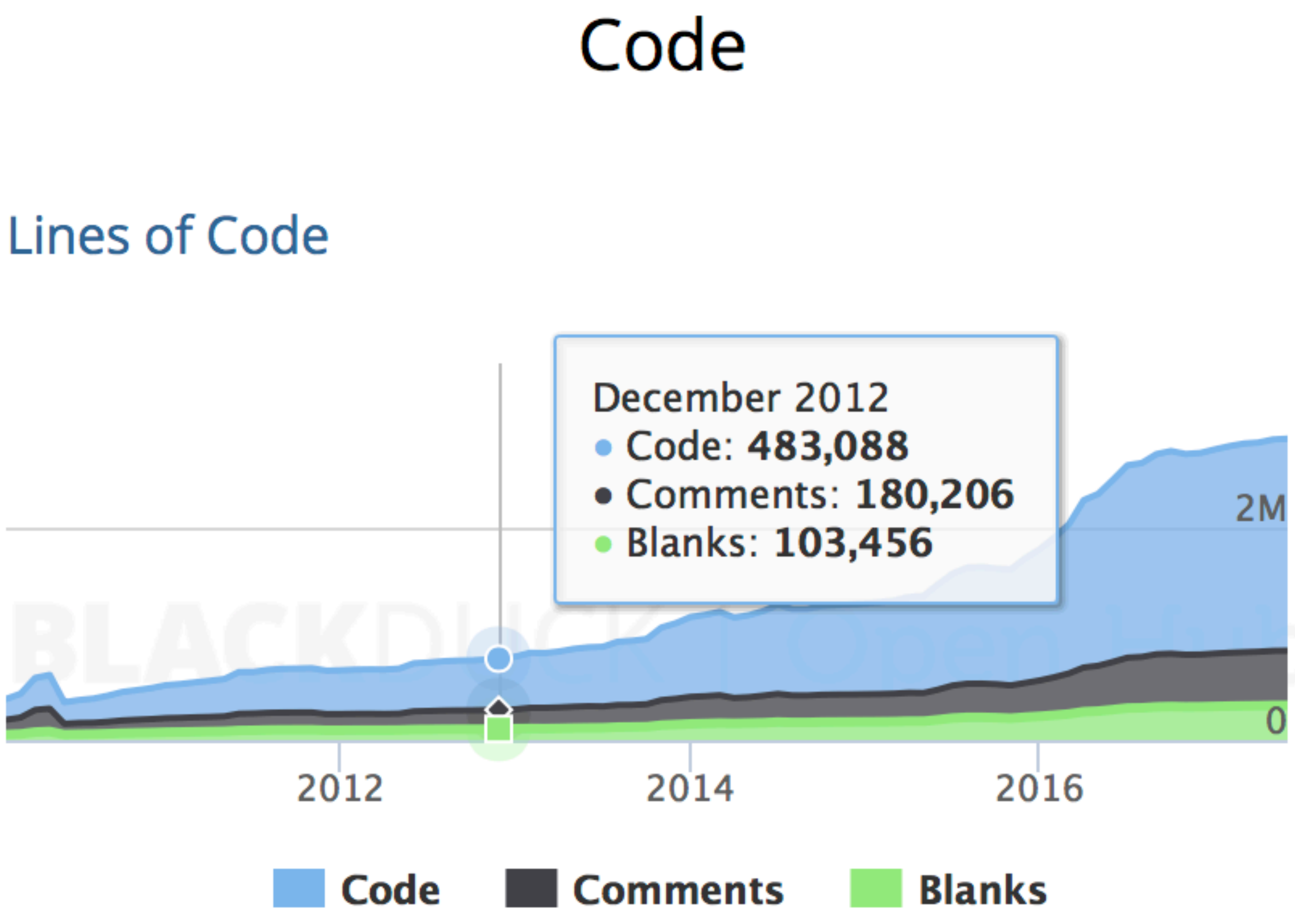
security issues
maintenance
docs
devops
4.5mln LOC
MySQL
C++
scaling
bugs
complexity
support

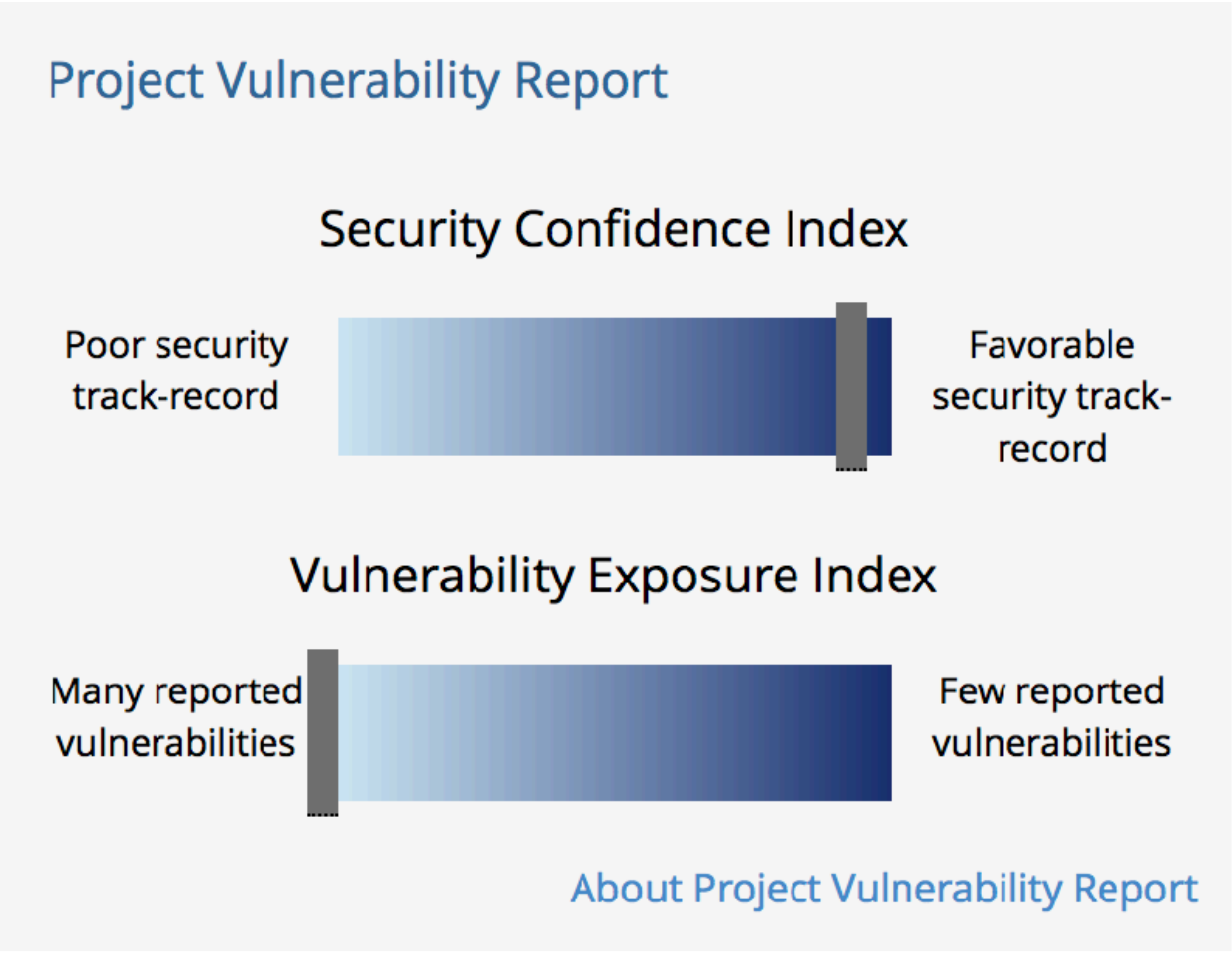
Backend

security issues
maintenance
docs
devops
2mln LOC
elasticsearch
JAVA
scaling
bugs
complexity
support

ElasticSearch

ElasticSearch is around 2 million Lines of Code





When you're ignoring data it's not a software engineering anymore.

It's DevOps (with all respect).

Ravelin story

How Ravelin designed fraud detection system

Graphing in Go

Must watch talk:

https://skillsmatter.com/skillscasts/8355-london-go-usergroup

skills matter

Graphing in Go by Jono MacDougall @ Ravelin

Jono MacDougall

GO London User Group

19.10.2016

skillsmatter.com @skillsmatter #codenode

Ravelin

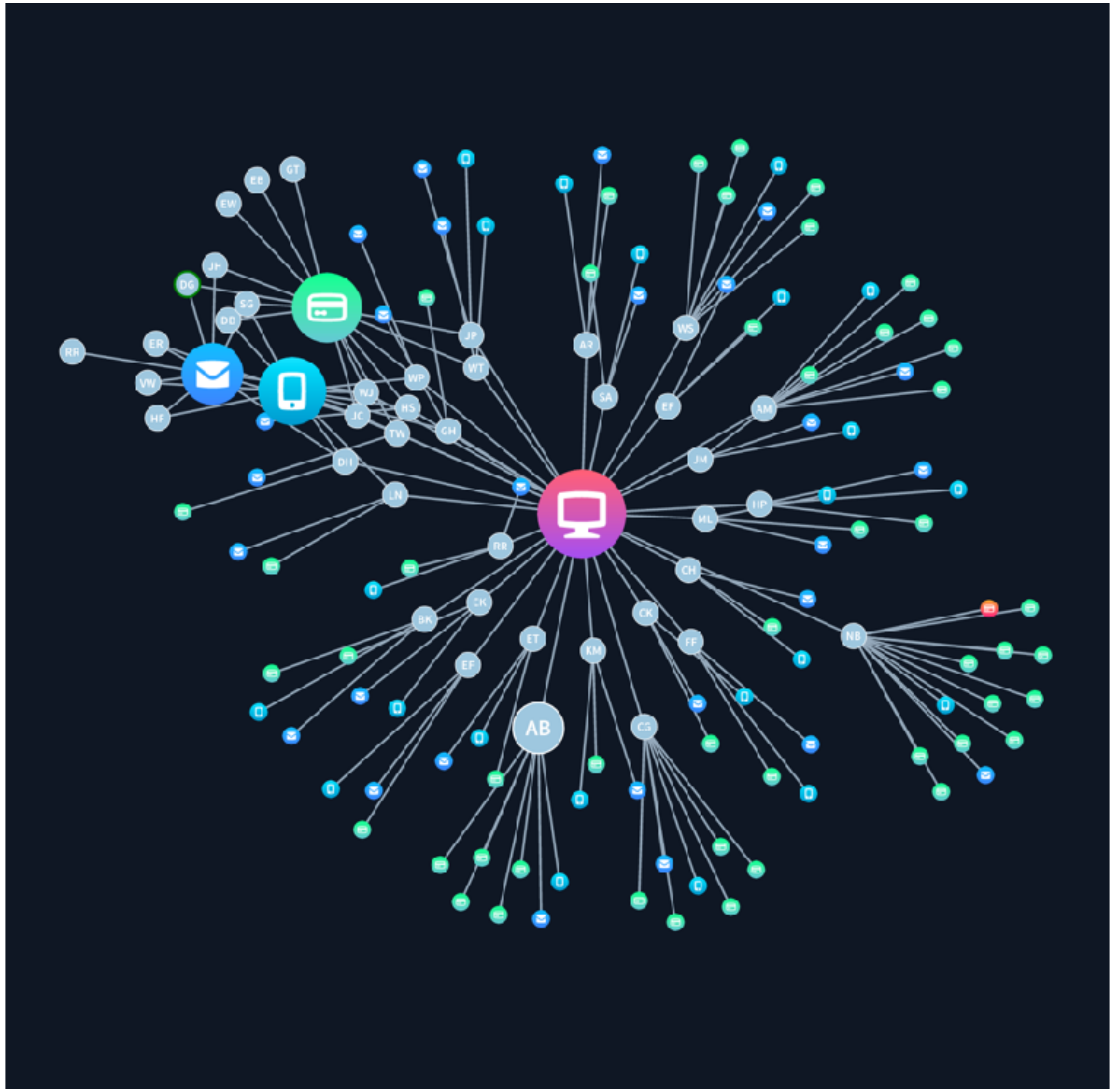
- Ravelin does fraud detection for financial sector
- For the machine learning they need data
- Data is a different features

Ravelin

- Clients make an API call to check if they allow order to proceed
- So the latency is critical here.

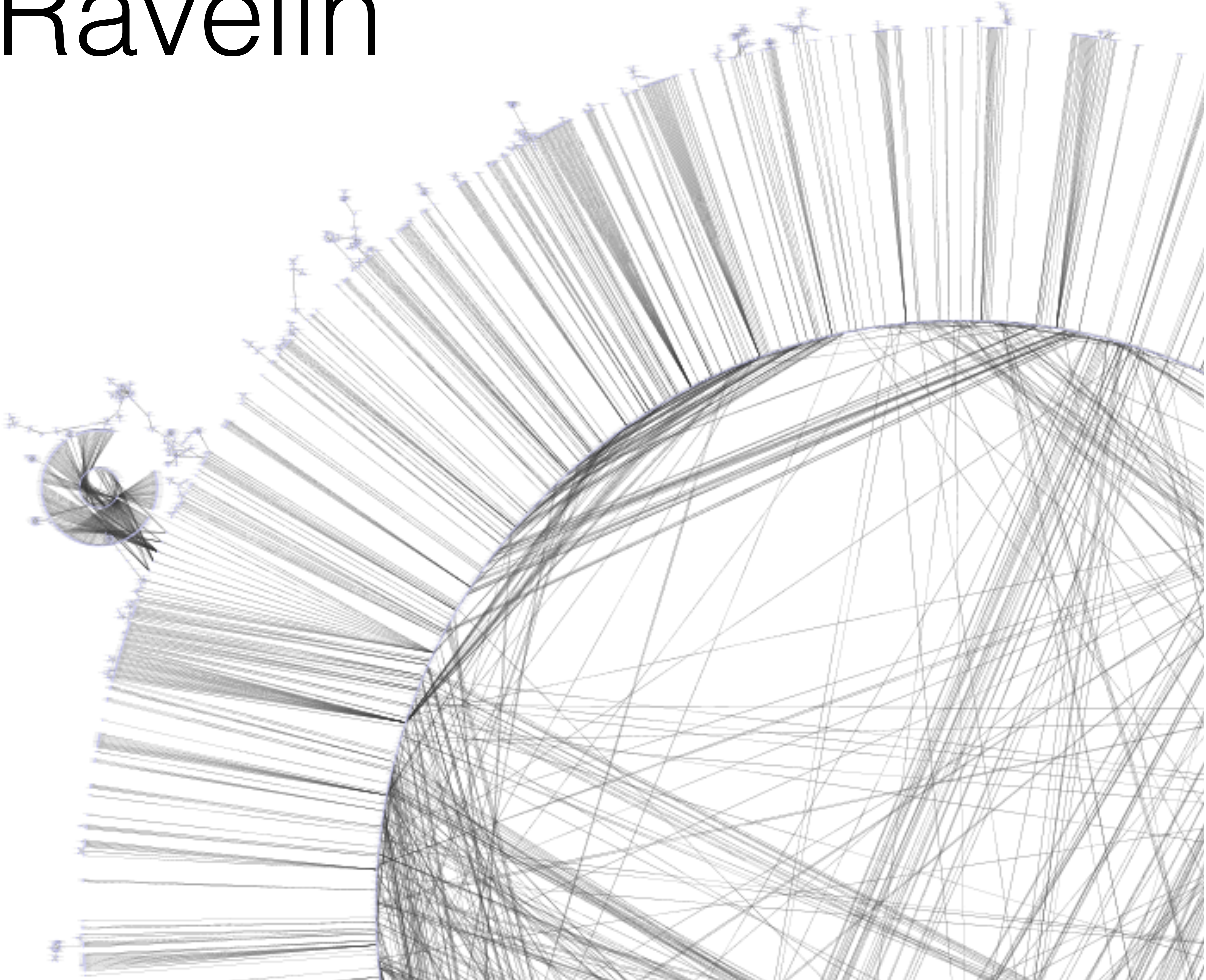
Ravelin

- But there are complex features
- They needed to connect things like phone numbers, credit cards, emails, devices and vouchers
- So new people could be easily connected to known "fraudsters" with very little data.



Ravelin

- They studied the problem offline
- It was clear they need a graph database



Ravelin

So, they looked at major players in Graph databases world...

Cayley

TITAN

neo4j

So they returned to the whiteboard and asked the question:

"What data do we actually need?"

Ravelin

- "What we care about is the **number of people that are connected** to you."
- "And if any of those people are known fraudsters."

Ravelin

- So they come up with the solution by using **Union Find** (*disjoint-set*) data structure
- This data structure basically allows you to:
  - find things (and subsets they are in)
  - join subsets

Union Find

We have 5 people: 1 2 3 4 5

Union Find

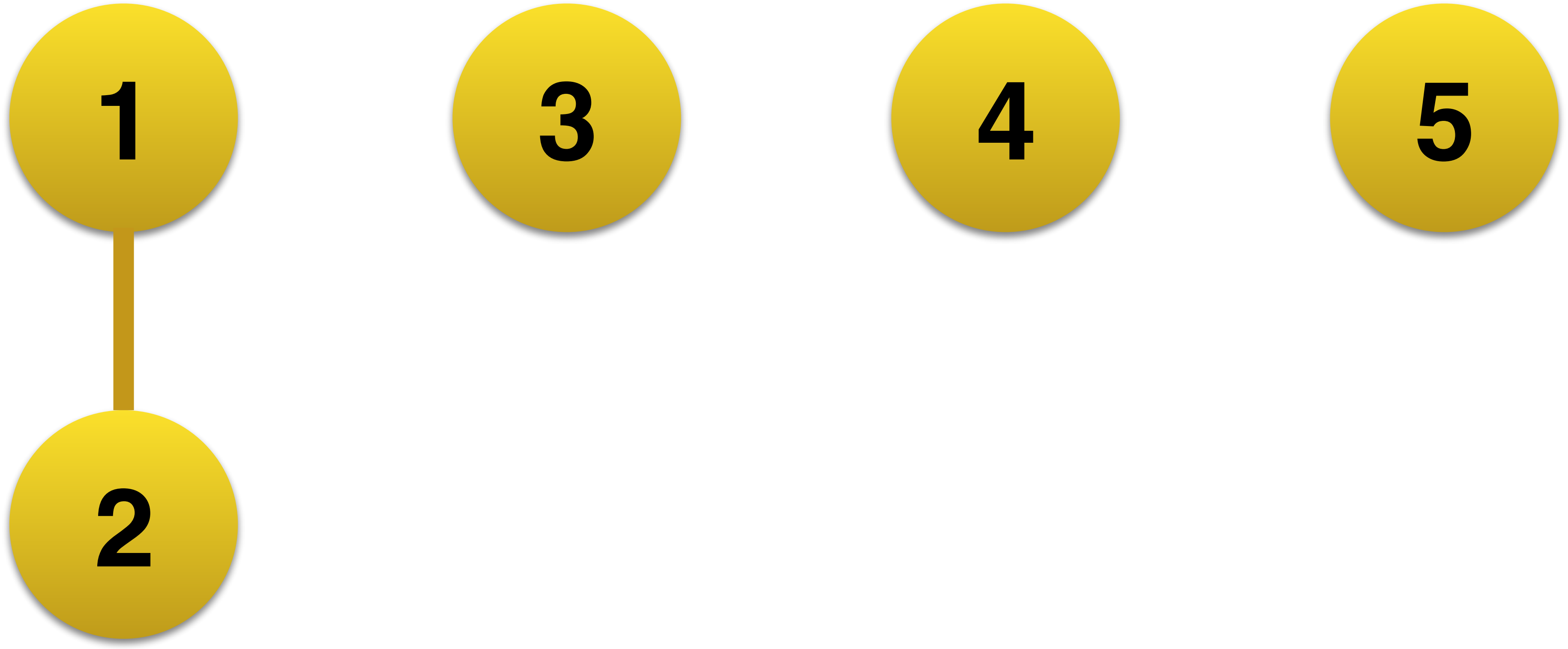
We have 5 people:

1 and 2 are friends

Union Find

We have 5 people:

1 and 2 are friends

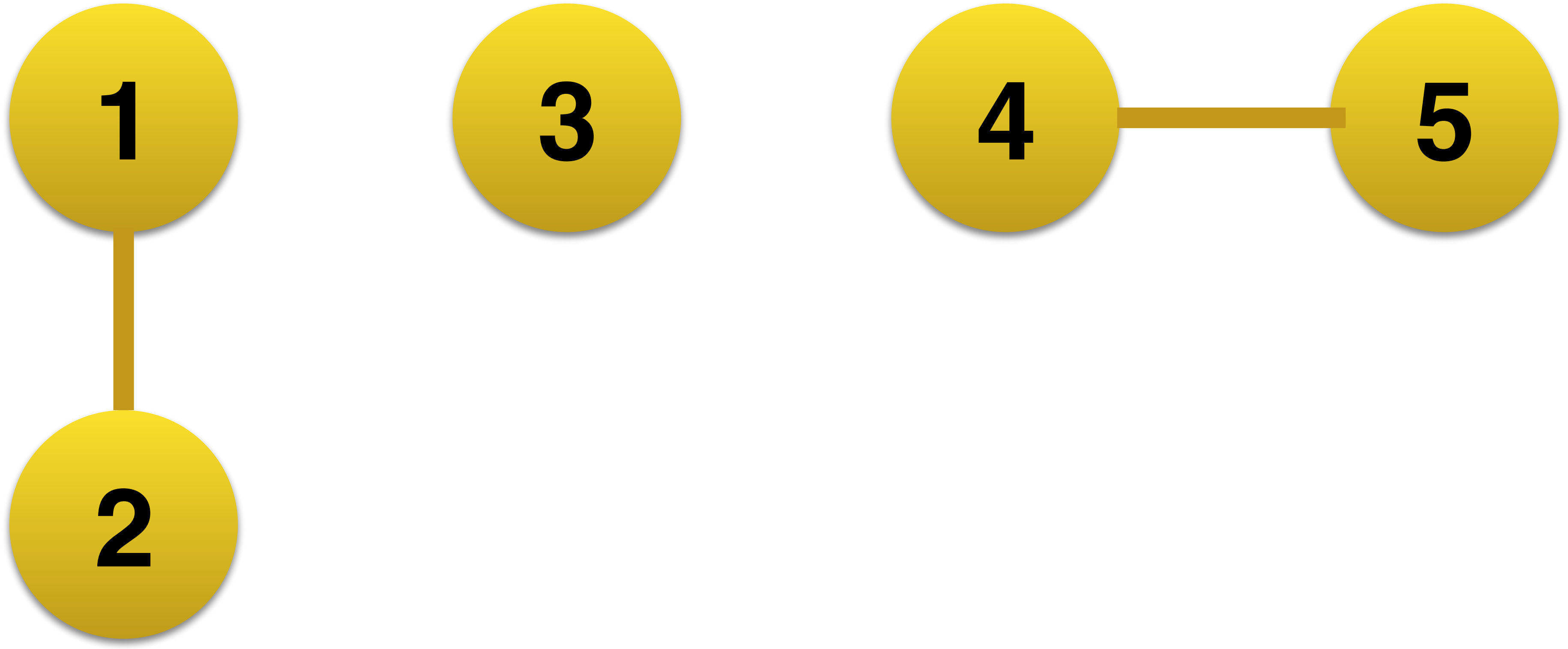


Union Find

We have 5 people:

1 and 2 are friends

4 and 5 are friends

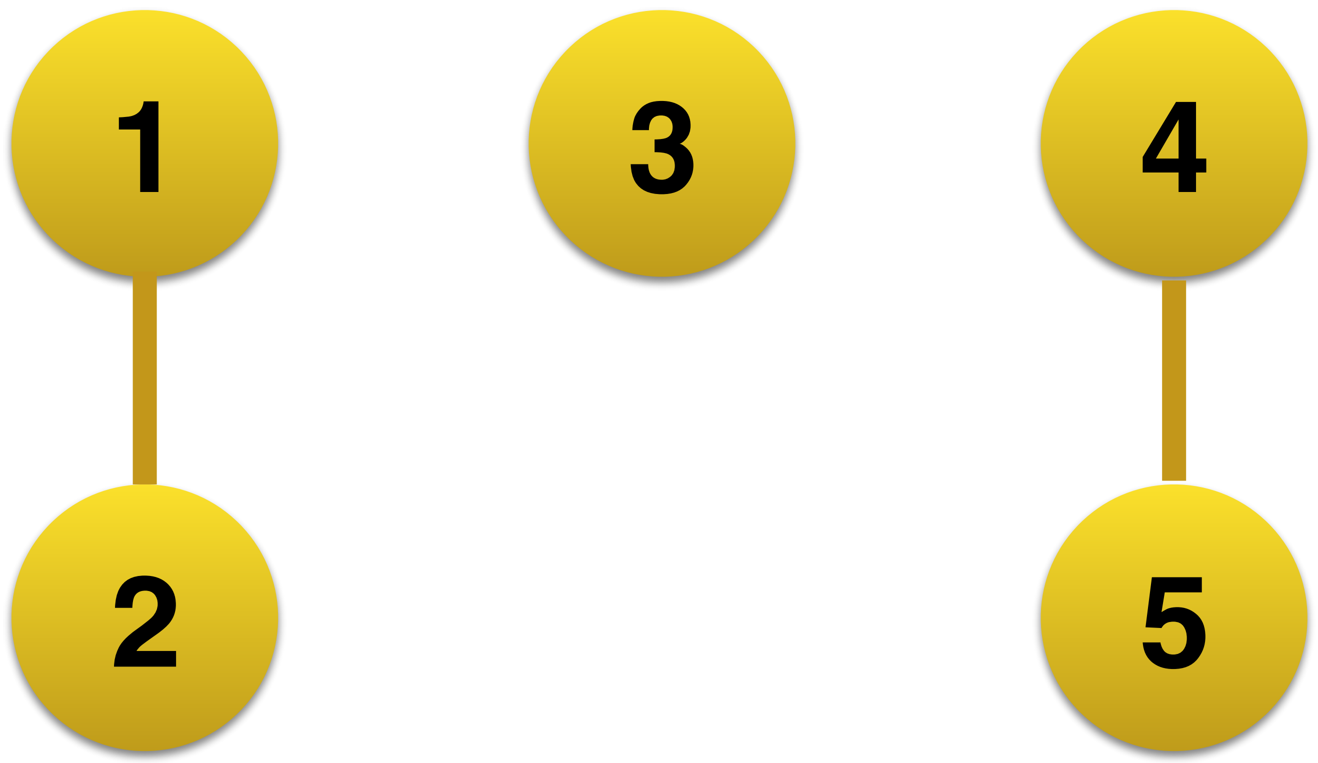


Union Find

We have 5 people:

1 and 2 are friends

4 and 5 are friends



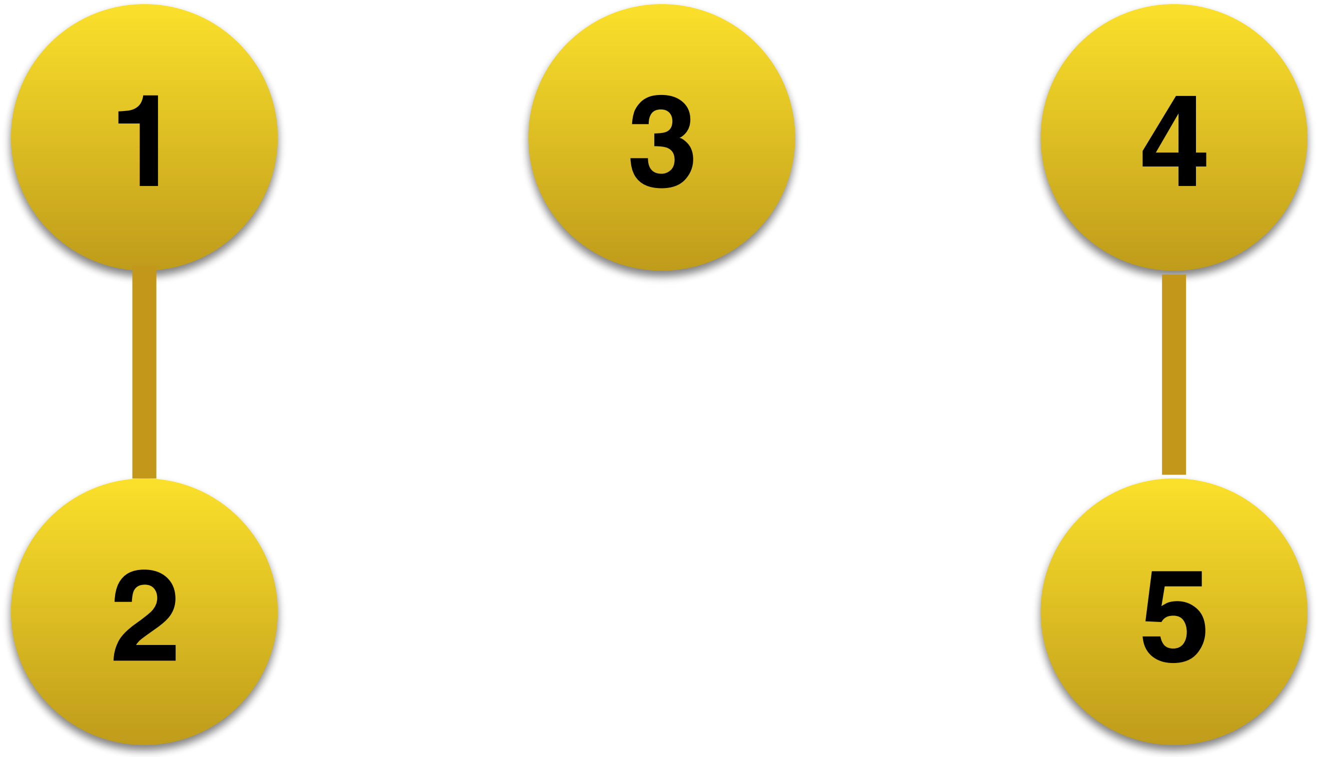
Union Find

We have 5 people:

1 and 2 are friends

4 and 5 are friends

3 and 5 are friends



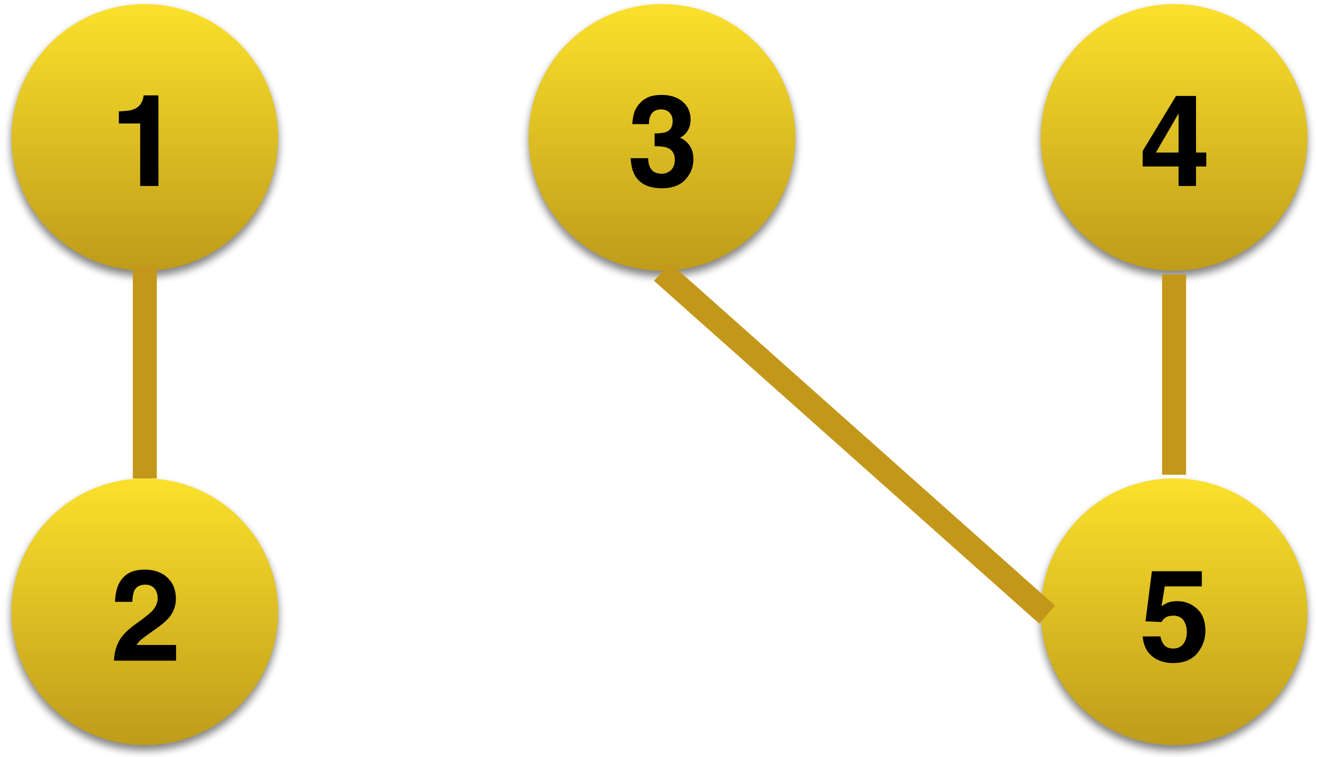
Union Find

We have 5 people:

1 and 2 are friends

4 and 5 are friends

3 and 5 are friends



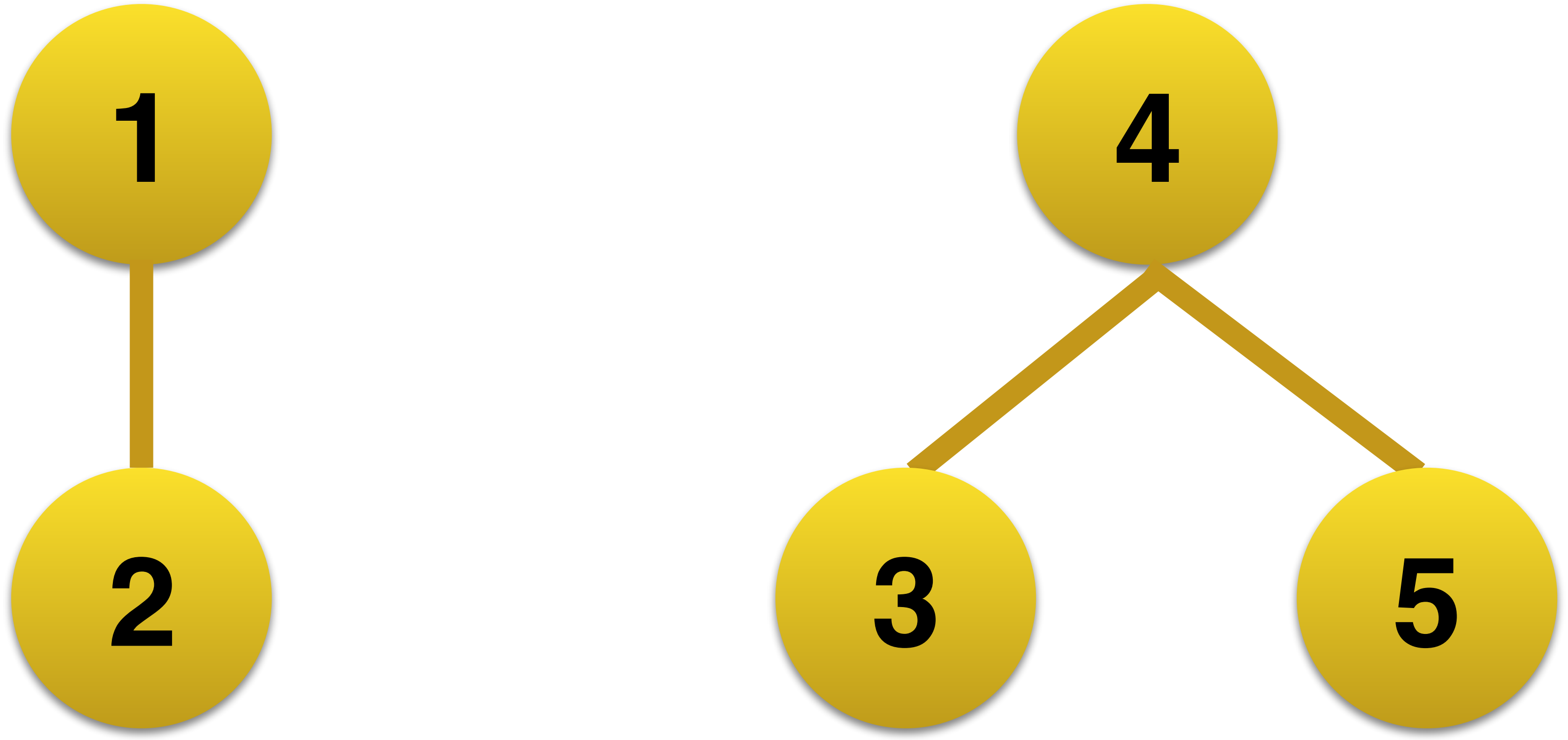
Union Find

We have 5 people:

1 and 2 are friends

4 and 5 are friends

3 and 5 are friends



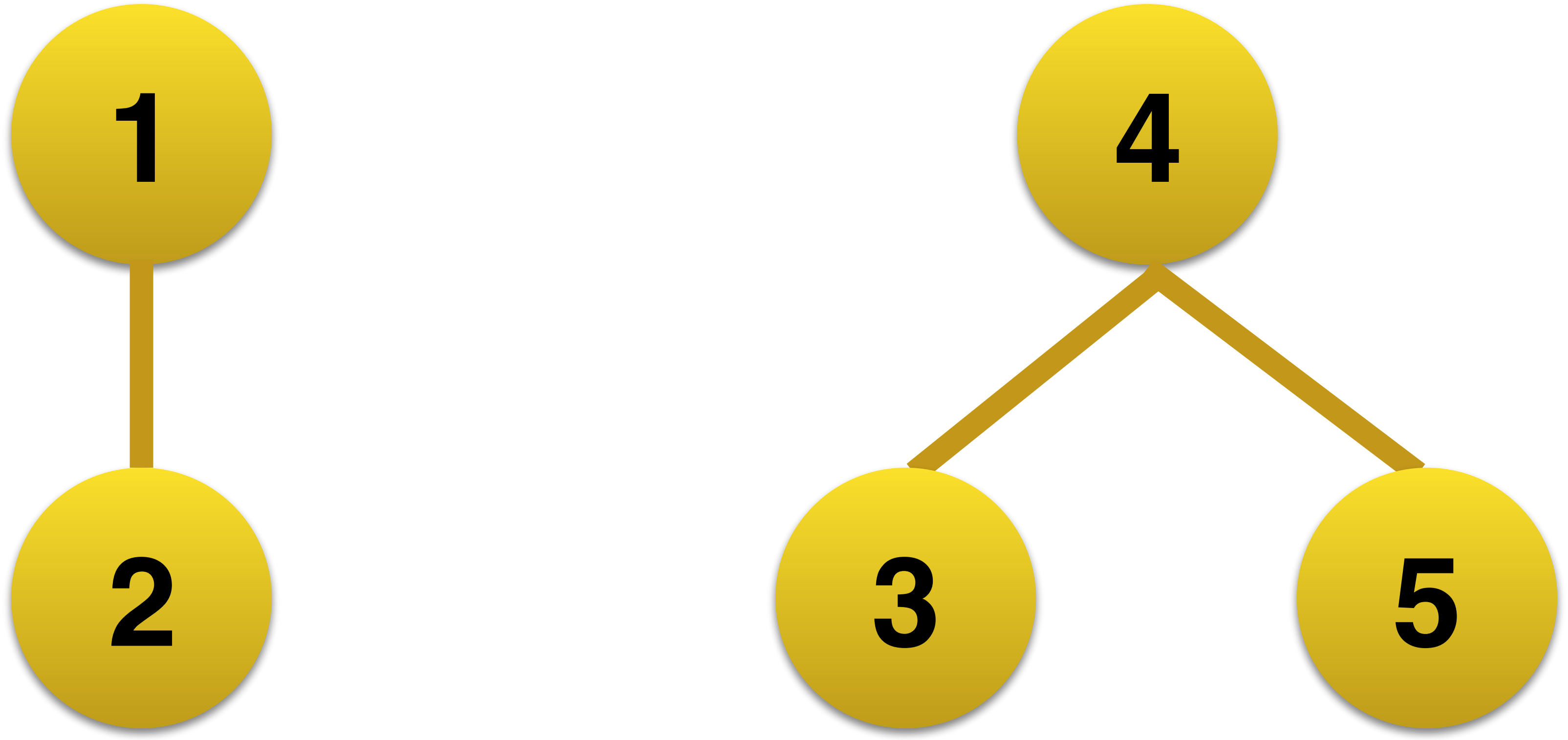
Union Find

We have 5 people:

1 and 2 are friends

4 and 5 are friends

3 and 5 are friends



FindSet - does 3 belongs to the same set as 2?
MergeSet - 3 and 2 are friends, so let's join set

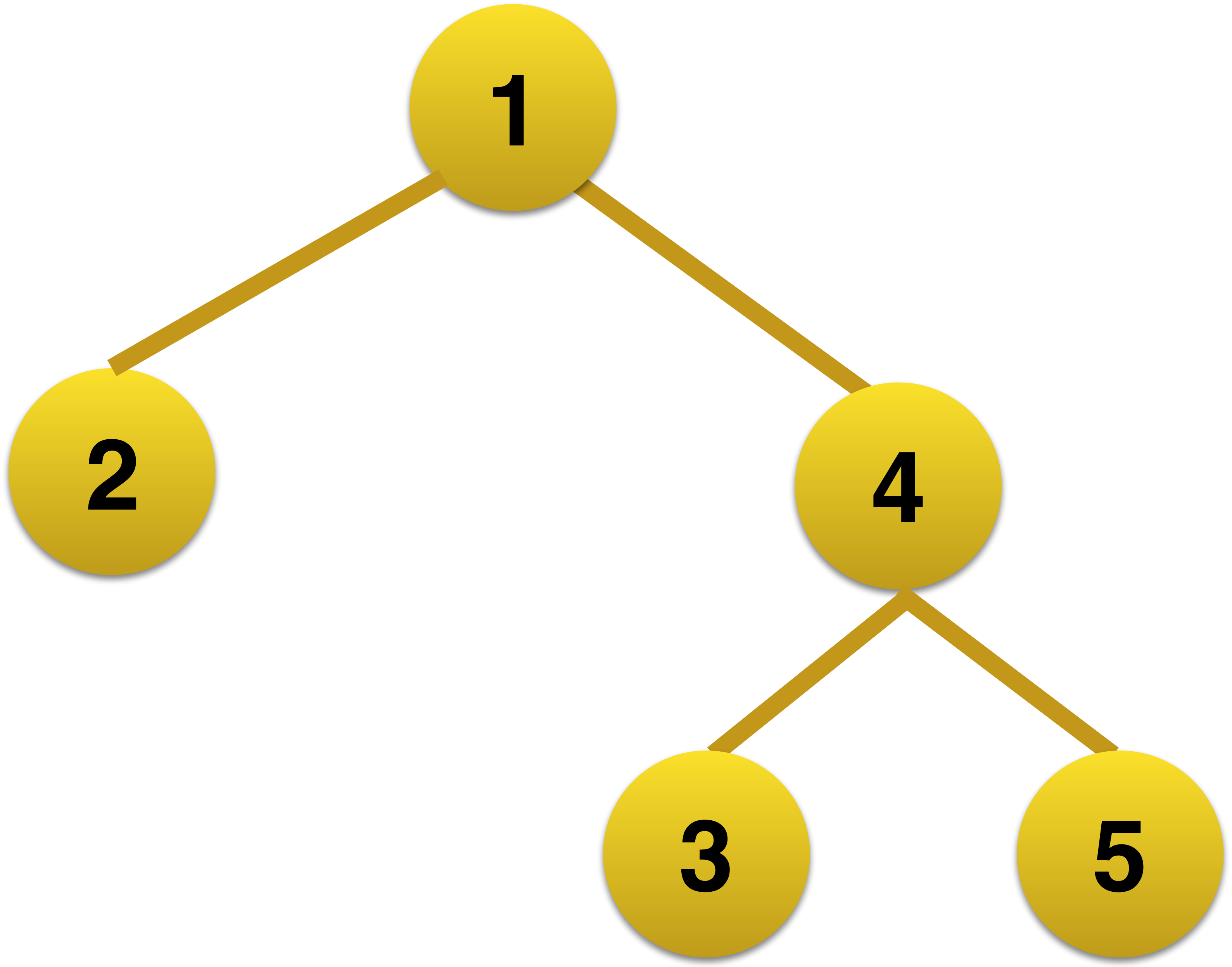
Union Find

We have 5 people:

1 and 2 are friends

4 and 5 are friends

3 and 5 are friends



FindSet - does 3 belongs to the same set as 2?
MergeSet - 3 and 2 are friends, so let's join set

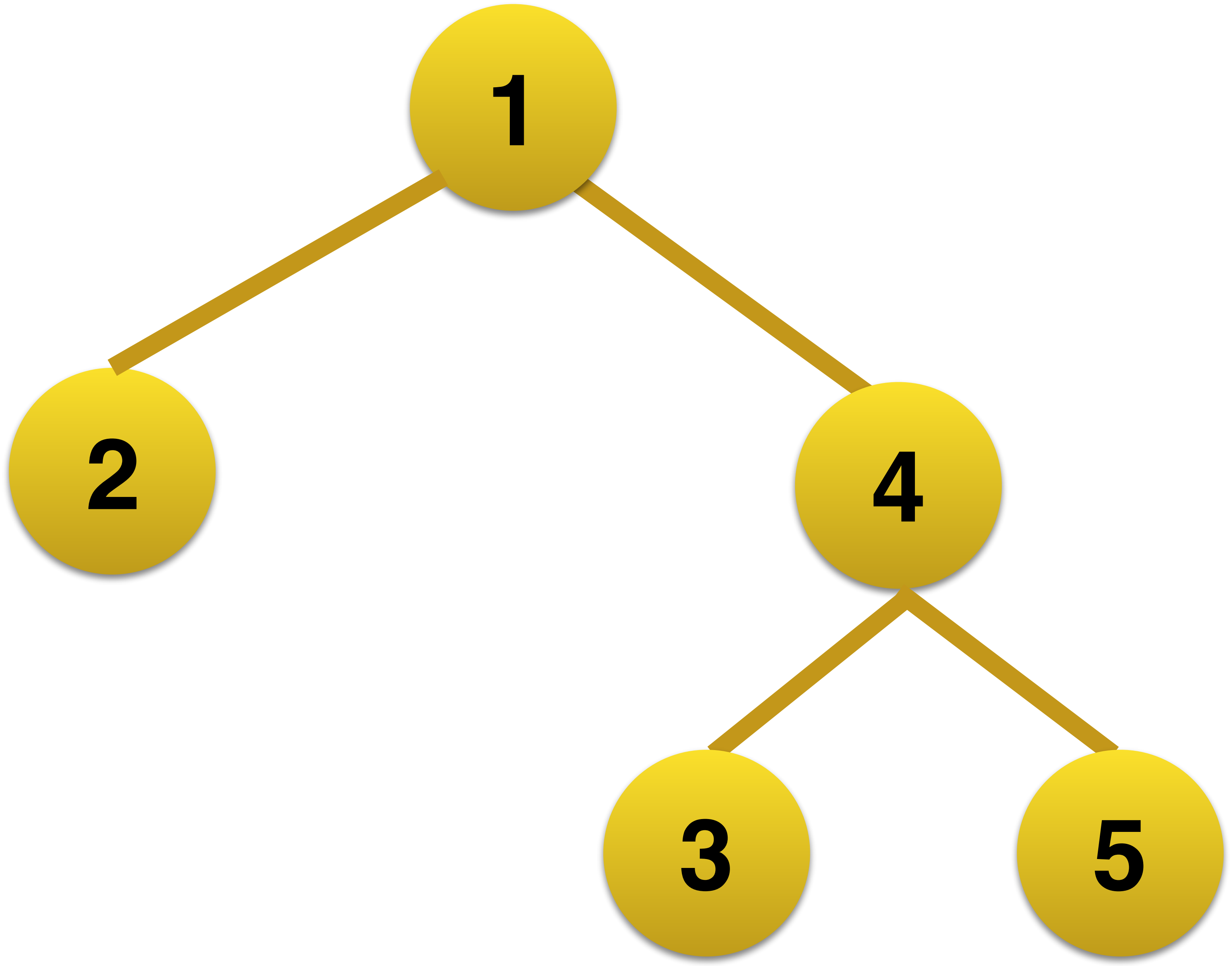
Union Find

We have 5 people:

1 and 2 are friends

4 and 5 are friends

3 and 5 are friends



FindSet - does 3 belongs to the same set as 2?
MergeSet - 3 and 2 are friends, so let's join set
FindSet(3)

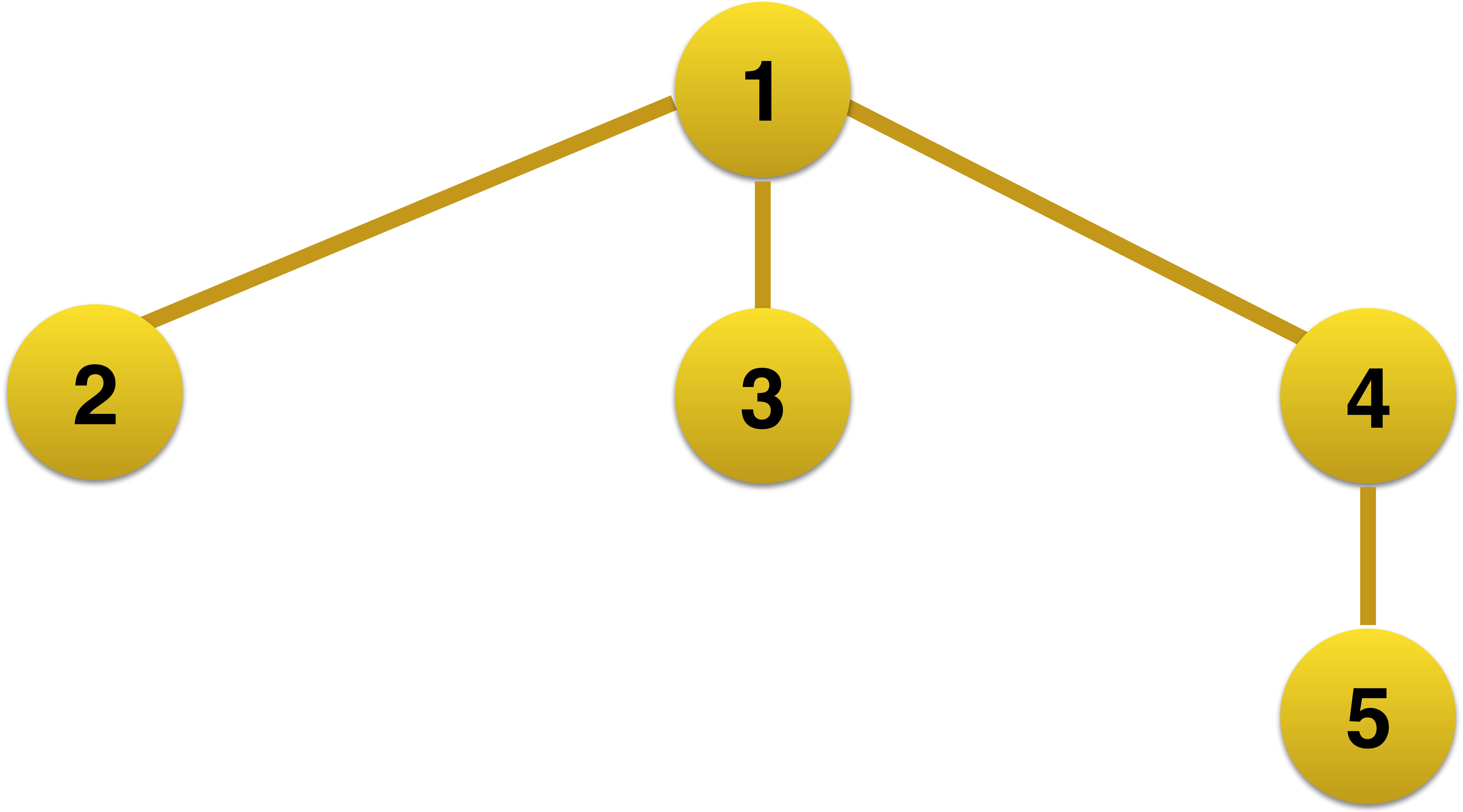
Union Find

We have 5 people:

1 and 2 are friends

4 and 5 are friends

3 and 5 are friends



FindSet - does 3 belongs to the same set as 2?
MergeSet - 3 and 2 are friends, so let's join set
FindSet(3)

Union Find

- It's really fast and has small memory footprint:
  - CreateSet - O(1)
  - FindSet - O($\alpha(n)$)* *(worst case)*
  - MergeSet - O($\alpha(n)$)* *(worst case)*
- *Visualization:* https://visualgo.net/en/ufds

** $\alpha(n)$ - is an inverse Ackerman function, grows slower than log(n)*

Ravelin

Simplified version of code used:

```
type Node struct {
    Count  int32
    Parent string
}

type UnionFind struct {
    Nodes map[string]*Node
}
```

Ravelin

Simplified version of code used:

```
// ...

    for node.Parent != id {

        // `node` is not the parent as parents always point to themselves
        // Set our ID to that of the parent of `node` (move up the graph)
        id = node.Parent

        // Get the new parent after moving up the graph
        newParent := uf.indexNode(id)

        // Push `node` up the graph by pointing it at the parents parent (skip the middle man)
        node.Parent = newParent.Parent

        // Set that parent as `node` and loop.
        // We keep doing this until we are at the top of the graph
        node = newParent
    }

    return node
}
```

Ravelin

Simplified version of code used:

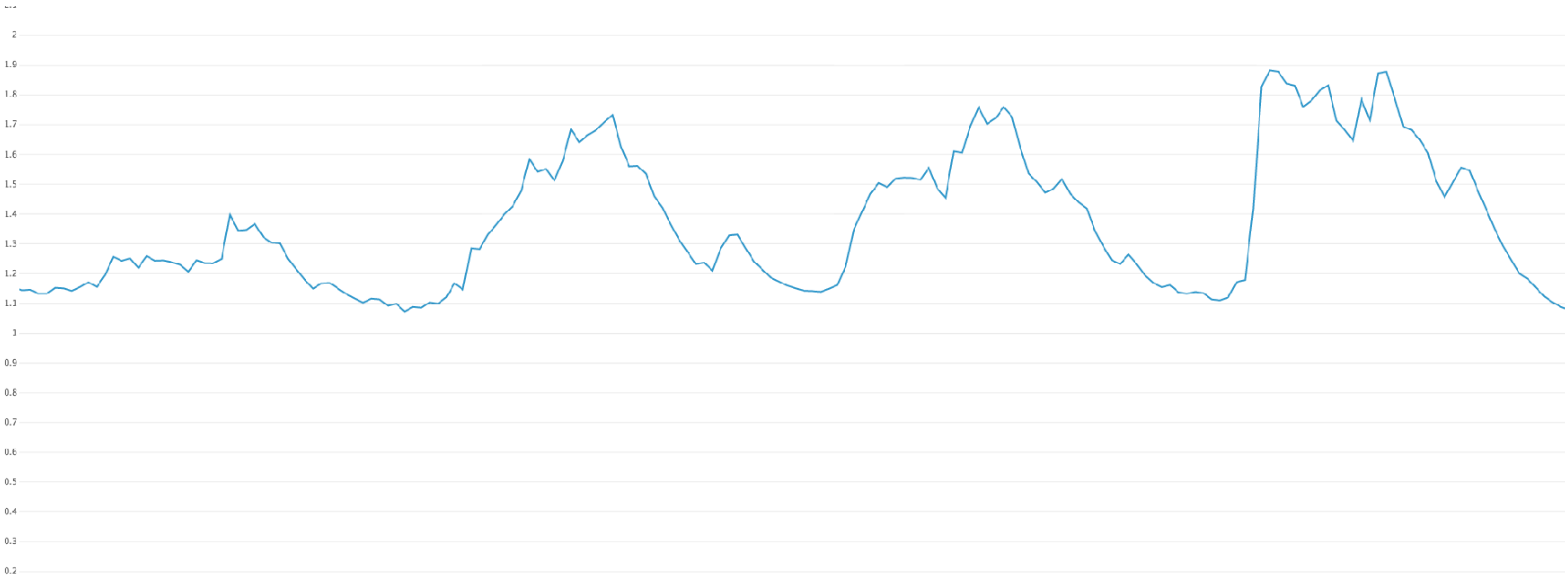
```
// Add adds a connection between two items to the map
func (uf *UnionFind) Add(a, b string) int32 {
    // Find the parent nodes of these two items. If the node is new, this mints a new node.
    firstNode, secondNode := uf.getParentNodeOrNew(a), uf.getParentNodeOrNew(b)

    // Join the two nodes together
    var parent *Node
    if firstNode.Parent == secondNode.Parent {
        // The parents are the same so we are done
        return firstNode.Count
    } else if firstNode.Count > secondNode.Count {
        // We pick a parent by chosing the node with the highest count
        parent = uf.setParent(firstNode, secondNode)
    } else {
        // secondNode is the parent
        parent = uf.setParent(secondNode, firstNode)
    }

    return parent.Count
}
```

Ravelin

95th is under 2ms, average is closer to 1ms

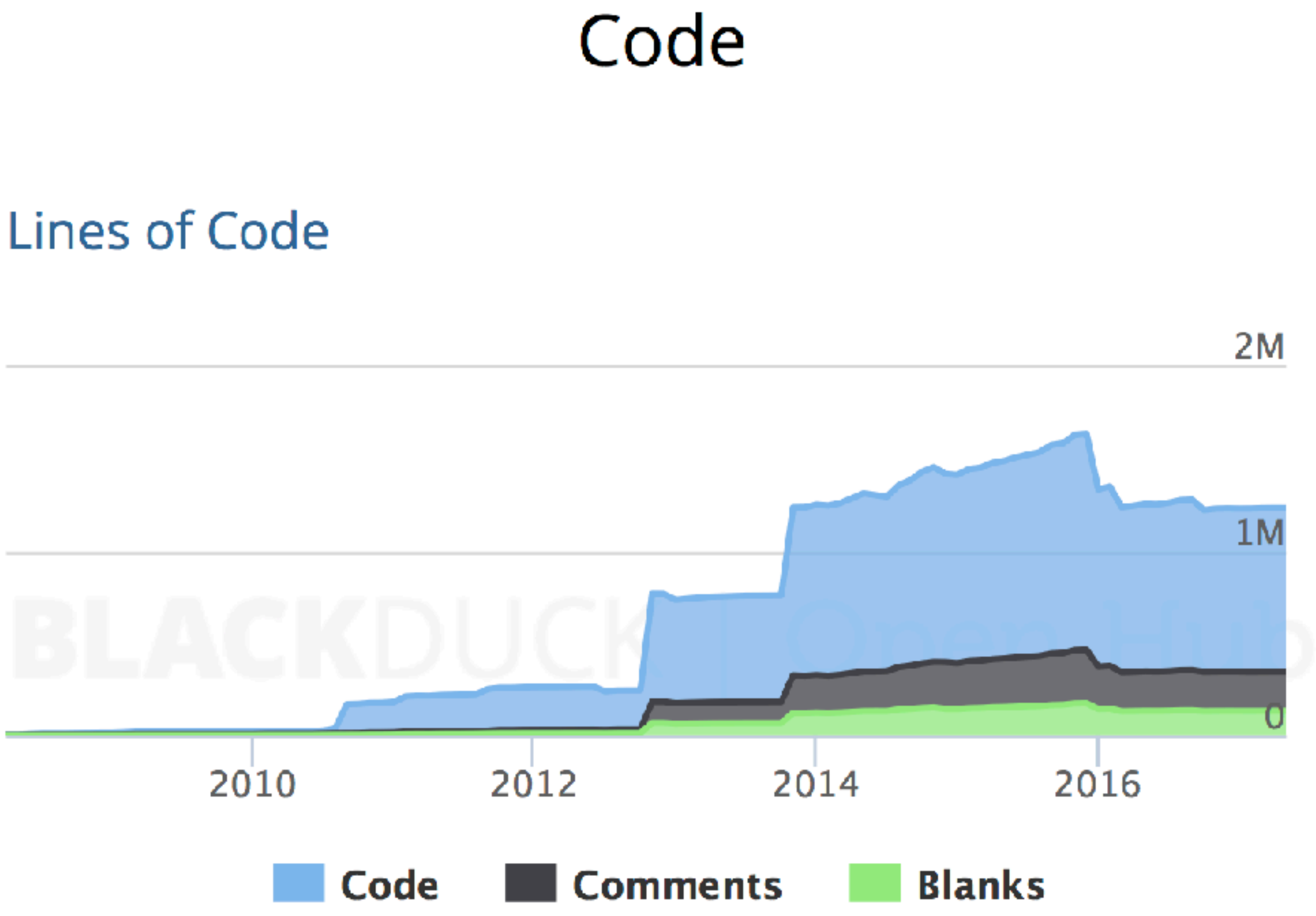


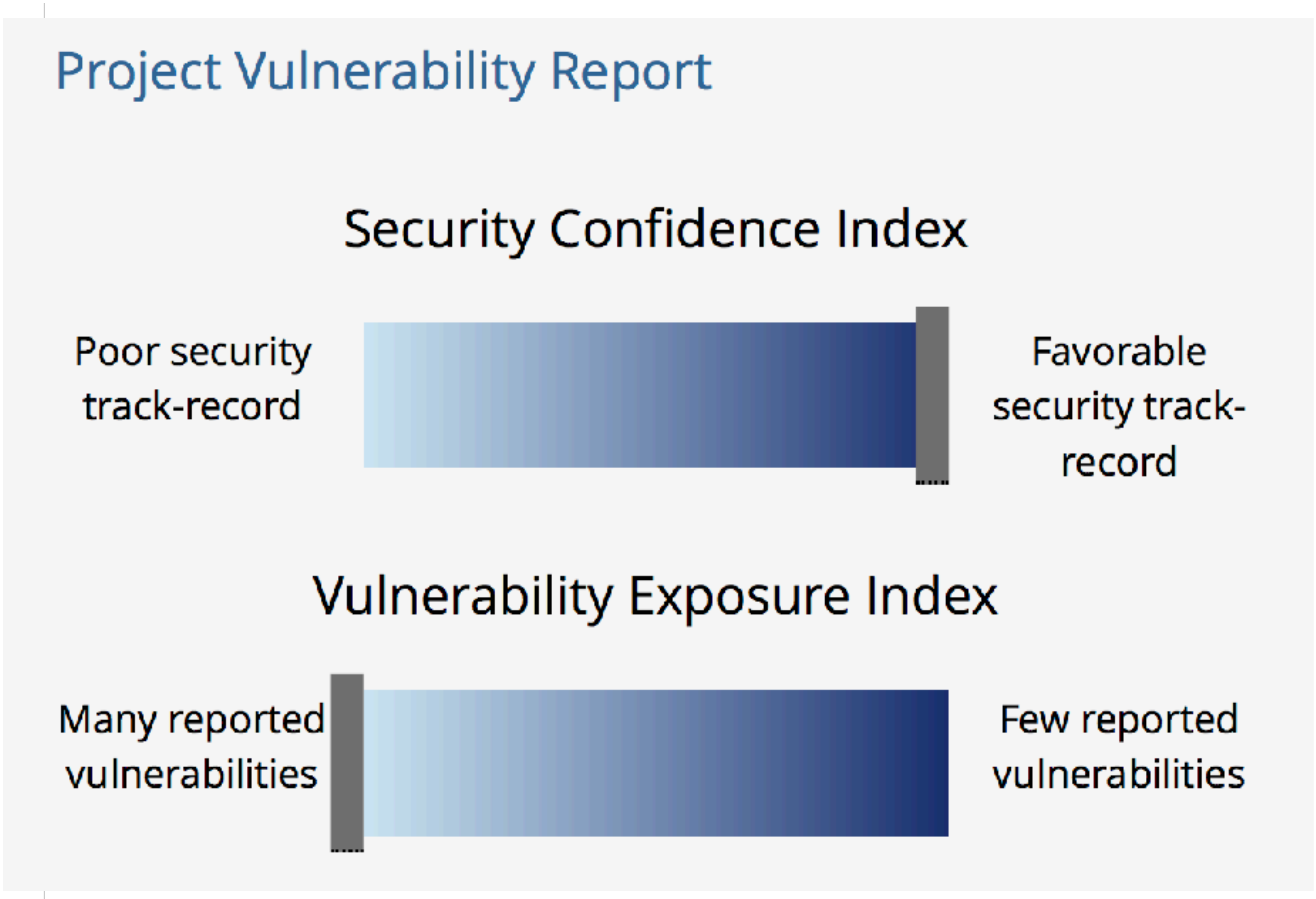
Ravelin

- Store the data in memory
- Persist in BoltDB
- Shard per client for scaling.

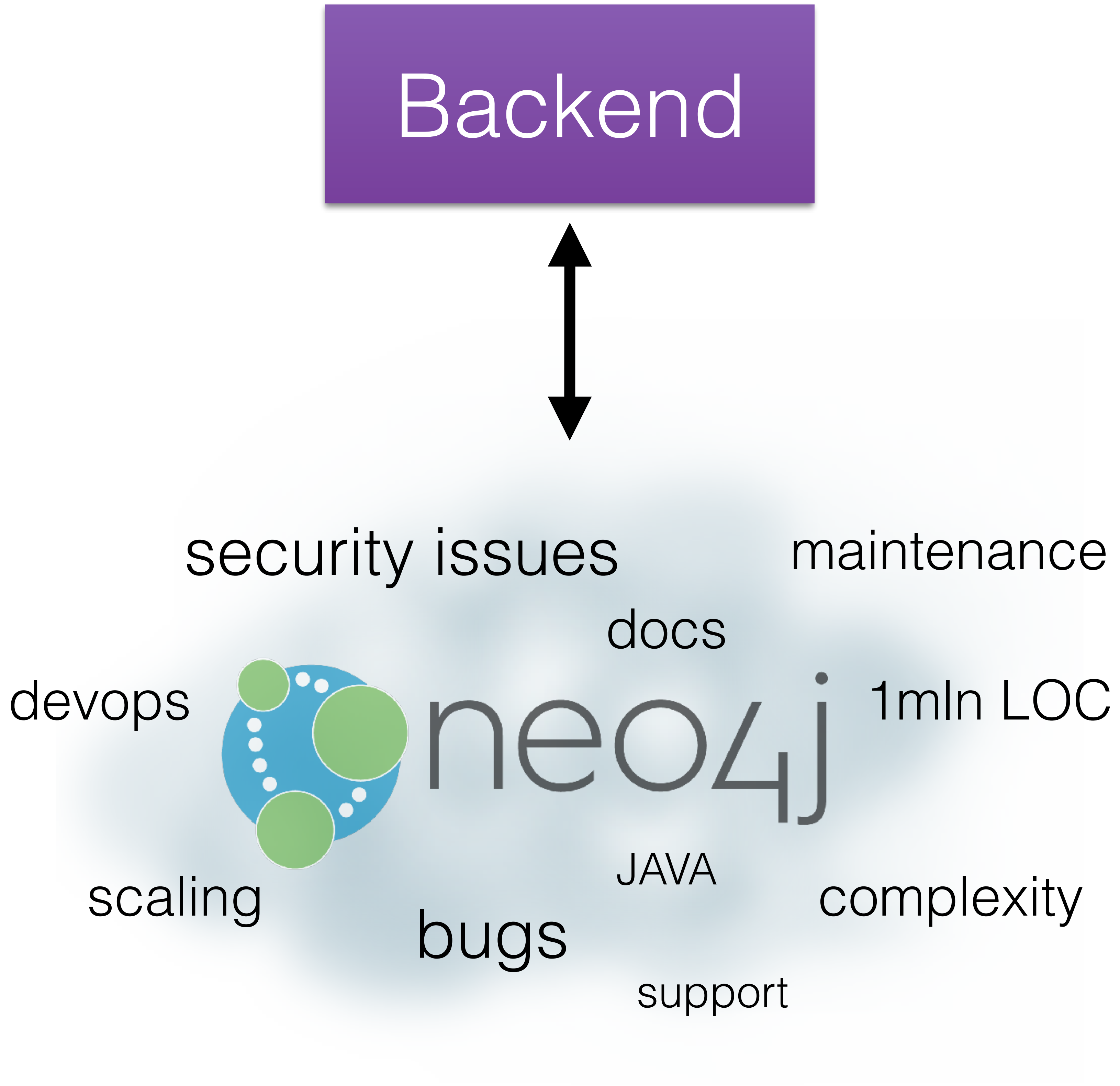
Ravelin

Neo4J is more than 1 million Lines of Code





Ravelin



vs

Backend UnionFind/ BoltDB

Backend UnionFind/ BoltDB

Backend UnionFind/ BoltDB

- Less code by order of magnitude
- Less bugs
- Less maintenance
- More simple
- Code fully owned by team

Systems designed around the data
are much simpler, than you think.

2. Always ask questions about data you work with

US AirForce and averages


485142
FT-142
U.S. AIR FORCE
285-799
TA-799

- In late 1940s, the USAF had a serious problem: the pilots could not keep control of their planes
- Planes were crashing even in the non-war period
- Sometimes up to 17 crashes per day

- Blaming pilots and training program didn't help
- Investigations confirmed that planes were ok as well
- But people were keep dying

- Finally, they turned the attention to the design of the cabin
- It was designed in late 20s for the average pilot
- Data was taken from the massive study of soldiers during the Civil War


124.500
125.000
A HIDR. RENDSZER NYOMÁSMÉRŐJE
FUTÓMŰ NYOMÁSMÉRŐ
KIS SZÖG
LÉGCS. SZÖG
NAGY SZÖG
HIDEG
LEVEGŐ FORL.
MELEG
RÁDIÓ
SZPU
SZPU
RÁDIÓ

- USAF conducted new study of 4000+ pilots
- Measured 140 different body parameters
- And checked how many pilots fit to average by 10 parameters, relevant to the cabin design

Zero

- Only by 3 dimensions, only 3.5% of the pilots were "average sized"
- There was no such thing as an "average pilot"
- So, USAF ordered to make cabins adjustable, to fit wide range of different pilots.
- Unexplained plane mishaps had reduced drastically

But why?

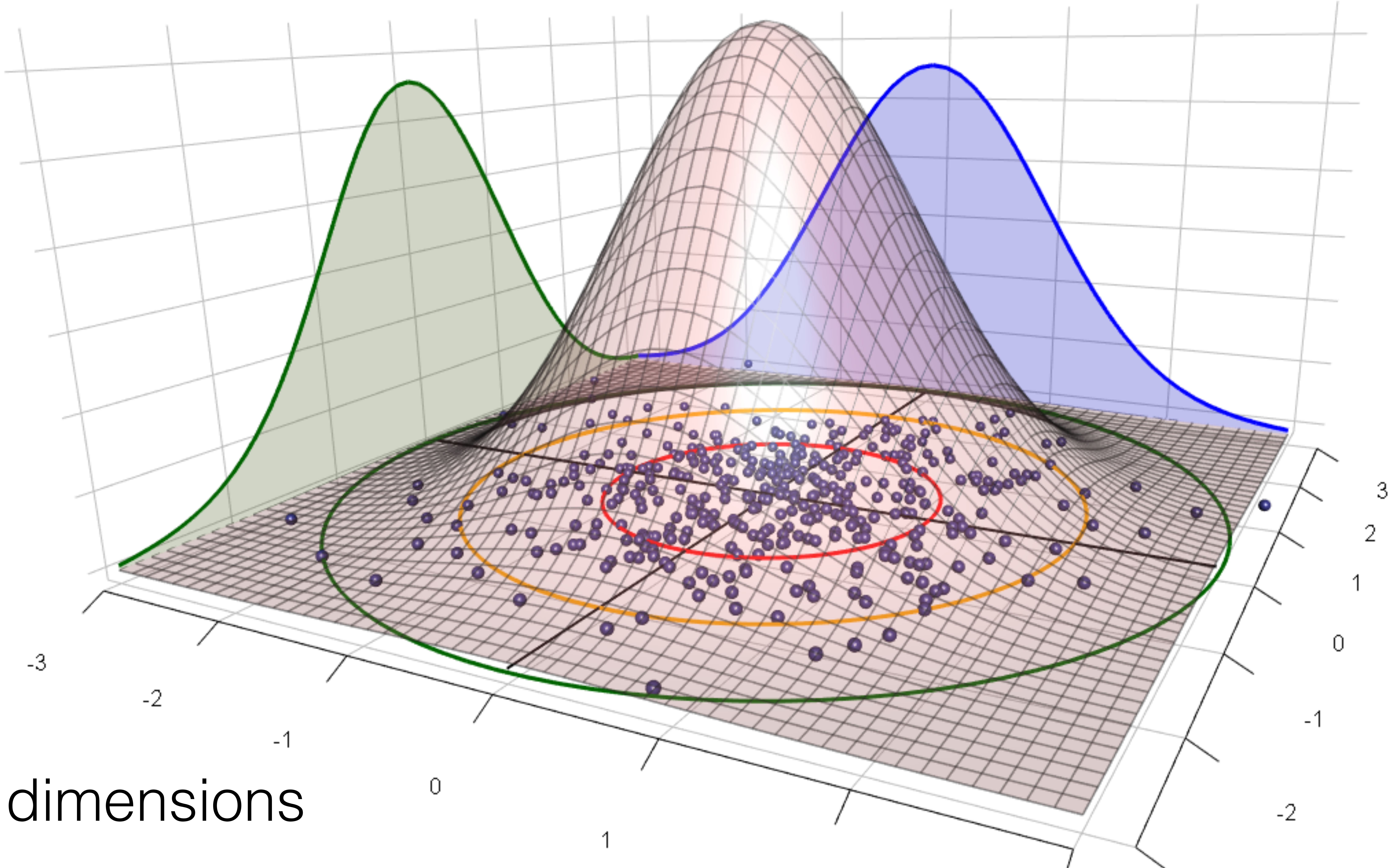
Average in high-dimension spaces

- Our intuition is built mostly on 1 dimension
- We tend to think that average is "where the most of values are"
- But it's only the particular case of:
    - 1 dimensional data
    - Normal or similar distribution

Average in high-dimension spaces

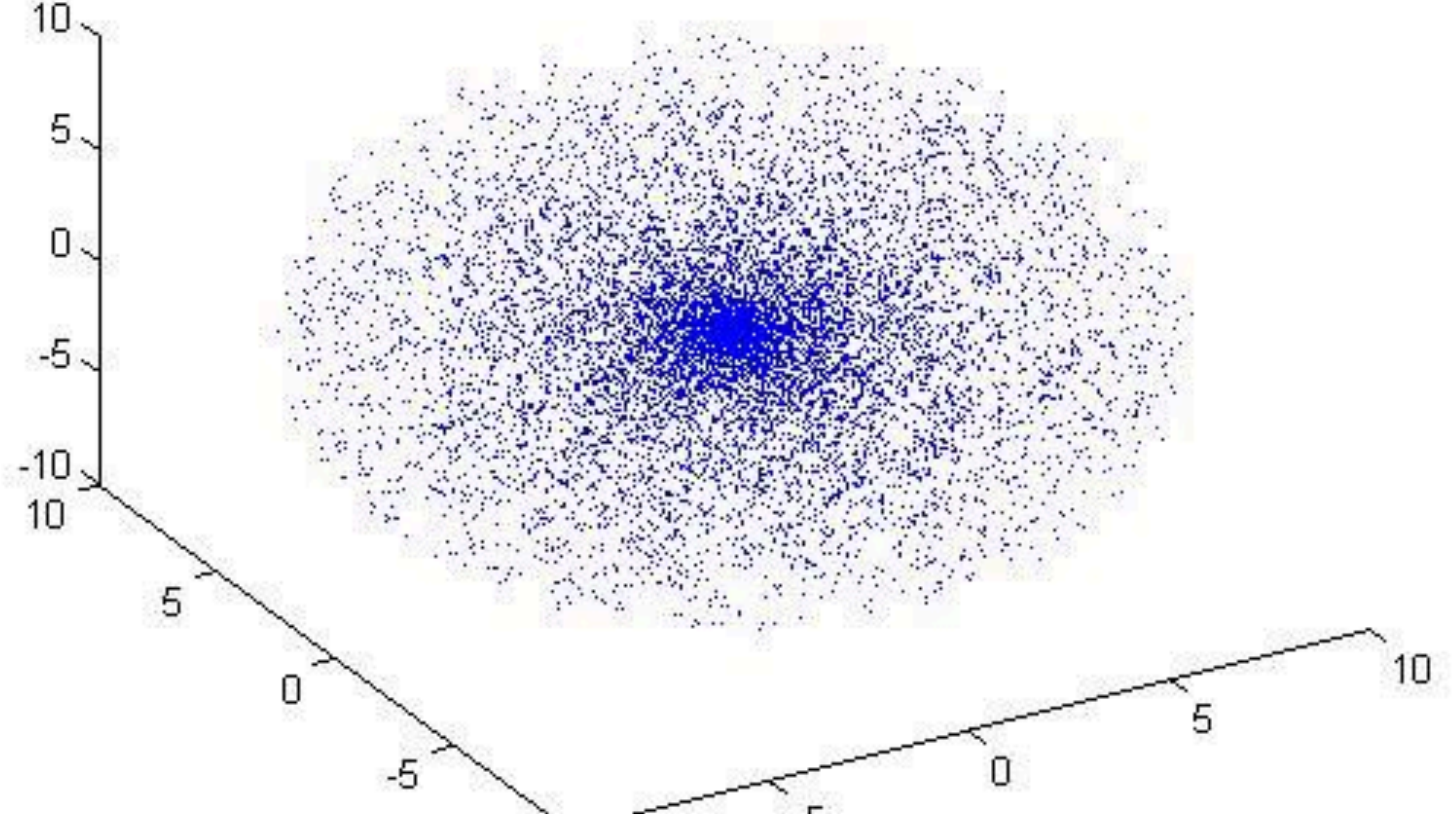
- Average is actually more like "center of the mass"
- Average value of the donut is inside the hole
- But for high dimensions everything is really messed up

1D Normal Distribution



2 dimensions

3 dimensions



- As number of dimensions grows, mass moves from center to the perifery
- In 10 dimensions, almost all values are on the edges*
- As some professors say **"The N-dimensonal orange is all skin"**

- But our intuition is built upon 1-2-3 dimensions
- For many types of data, intuition is not enough, we need math

Requests rate probability function

Requests distribution

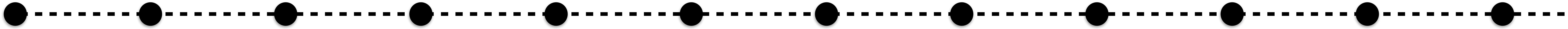
- We do a lot of servers in Go
- And we test them using load testing tools ('ab', 'vegeta', 'boom', own tools, etc)

Requests distribution

- Most tools are sending requests in parallel with constant intervals
- But that's not how requests arrive in reality
- Typical distribution of independent events is a **Poisson distribution**

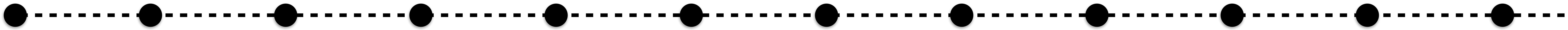
Requests distribution

Constant interval

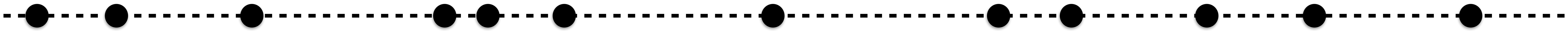


Requests distribution

Constant interval

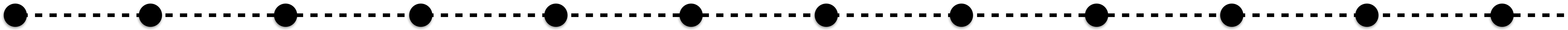


Uniform distribution

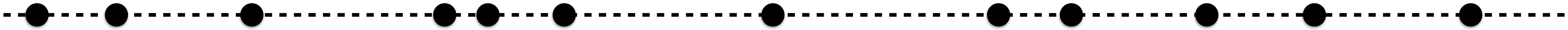


Requests distribution

Constant interval



Uniform distribution



Poisson distribution

Let's see how it may make a difference.

server.go

```go
package main

import "net/http"
import "time"

var locked bool
func handler(w http.ResponseWriter, r *http.Request) {
   if locked {
      w.WriteHeader(http.StatusInternalServerError)
      return
   }

   locked = true
   go time.AfterFunc(time.Millisecond, func() { locked = false })
}

func main() {
   http.HandleFunc("/", handler)
   http.ListenAndServe(":5000", nil)
}
```

client.go

```go
package main

import (
    "fmt"
    "io"
    "io/ioutil"
    "log"
    "net/http"
)

var delay = 100 * time.Millisecond
var N = 100
var r *rng.PoissonGenerator

func main() {
    Loop("Constant", constant)
    Loop("Uniform", uniform)
    Loop("Poisson", poisson)
}
```

```go
type delayFunc func() float32

func constant() float32 {
    return 1
}

func uniform() float32 {
    return rand.Float32()
}

func poisson() float32 {
    return float32(r.Poisson(1))
}
```

client.go

```go
func init() {
    seed := time.Now().UnixNano()
    rand.Seed(seed)
    r = rng.NewPoissonGenerator(seed)
    go func() {
        for {
            fmt.Printf(" ")
            time.Sleep(delay)
        }
    }()
}
func Loop(name string, fn delayFunc) {
   fmt.Println(name)
   for i := 0; i < N; i++ {
      v := fn() * float32(delay)
      time.Sleep(time.Duration(v))
      makeRequest()
   }
   fmt.Println()
}
```

```go
func makeRequest() {
   url := "http://localhost:5000/"
   resp, err := http.Get(url)
   if err != nil {
      log.Fatal(err)
   }
   io.Copy(ioutil.Discard, resp.Body)
   resp.Body.Close()
   if resp.StatusCode == http.StatusOK {
      fmt.Printf(".")
   } else {
      fmt.Printf("🔥")
   }
}
```

```
[mac@client ]$ go run client.go
```

- "Essential complexity" vs "accidental complexity" (F. Brooks, "No Silver Bullet")
- In Go we're lucky to have accidental complexity level pretty low

- We have more time to solve actual problems (essential complexity)
- And you need understand the data first
- Before you start writing the code

3. To make sense of the data, learn data science

Data Science

- It's a interdisciplinary field
- Math, statistics, computer science, visualization, machine learning, etc

If you want to be a good software engineer, you should be a passionate about data science.

1. Think about the whole system as one program
2. Always ask questions about data you work with
3. To make sense of this data, learn data science

Links

- http://highscalability.com/blog/2016/4/20/how-twitter-handles-3000-images-per-second.html
- https://skillsmatter.com/skillscasts/8355-london-go-usergroup
- https://www.thestar.com/news/insight/2016/01/16/when-us-air-force-discovered-the-flaw-of-averages.html
- https://medium.com/@charlie.b.ohara/breaking-down-big-o-notation-40963a0f4e2a
- https://www.youtube.com/watch?v=gas2v1emubU
- https://algorithmia.com/algorithms/ovi_mihai/TimestampToDate
- https://en.wikipedia.org/wiki/Disjoint-set_data_structure

Thank you.