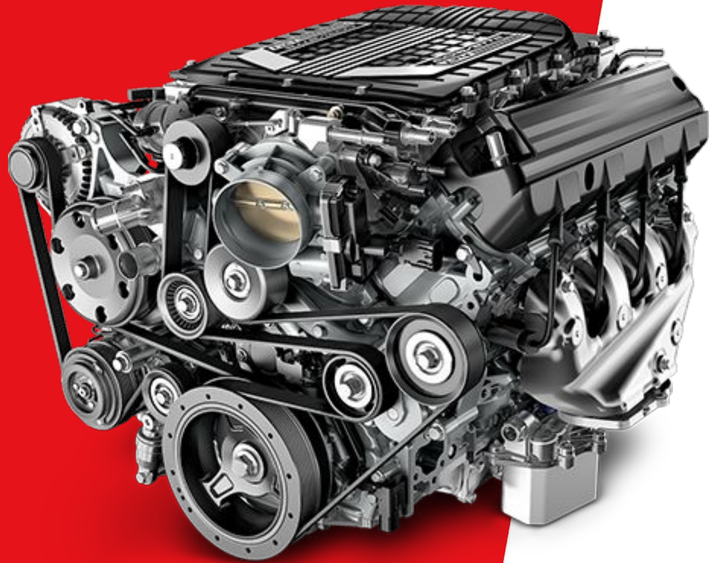
Microservices and go @ vroom



Igor Nodelman

vroom

About me

- Drivers development for Digital Video Recording systems
- Distributed Video Streaming System
- Enterprise Video Surveillance System
- And now: Changing how people buy cars!

<u>Education</u>

BSC - Computer Science - Bar-Ilan University

MBA - Tel Aviv University

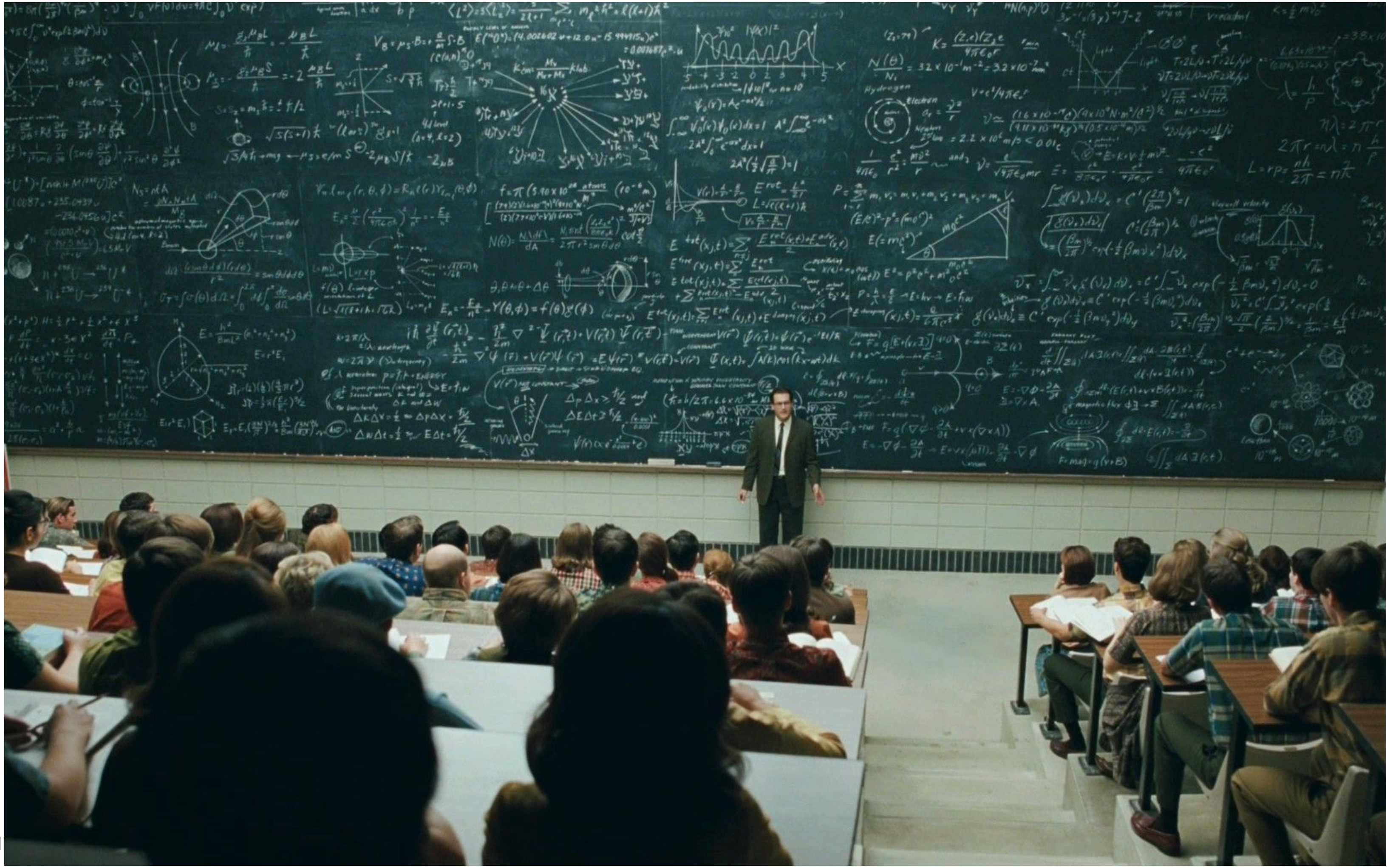
https://www.linkedin.com/in/nodelman

Ciencia
Biografía
Religión
Historia universal
Filosofía
Estética
Letras universales

amazon

TAXI Grand Central Partnership
PERSHING SQUARE
Park Ave
PRIVATE EYES GENTLEMEN'S CLUB
2C72
2L18
NEW YORK 2C72A TAXI
CHASE

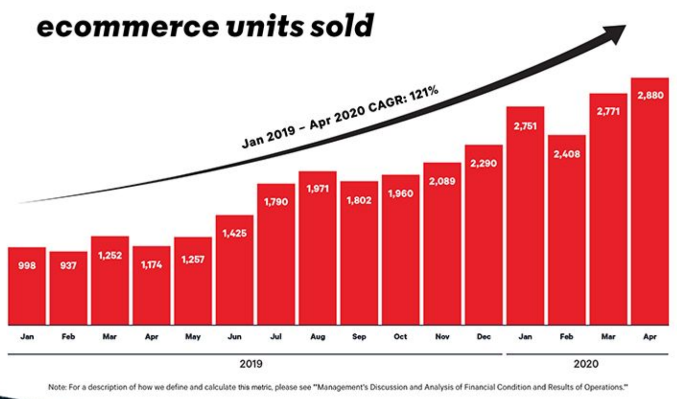
Uber



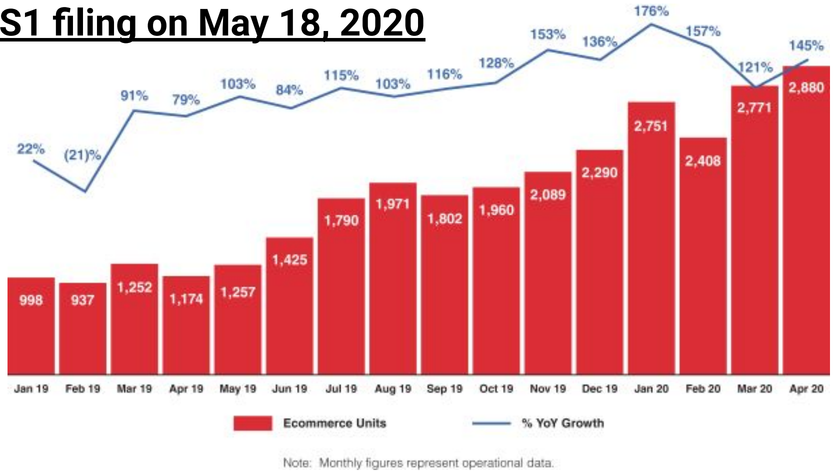
USED CARS • TRADE
RESIDENT PROPRIETOR TERMS
DOLLAR CAR
LOOK!
50

Click.
Buy.
Get In.

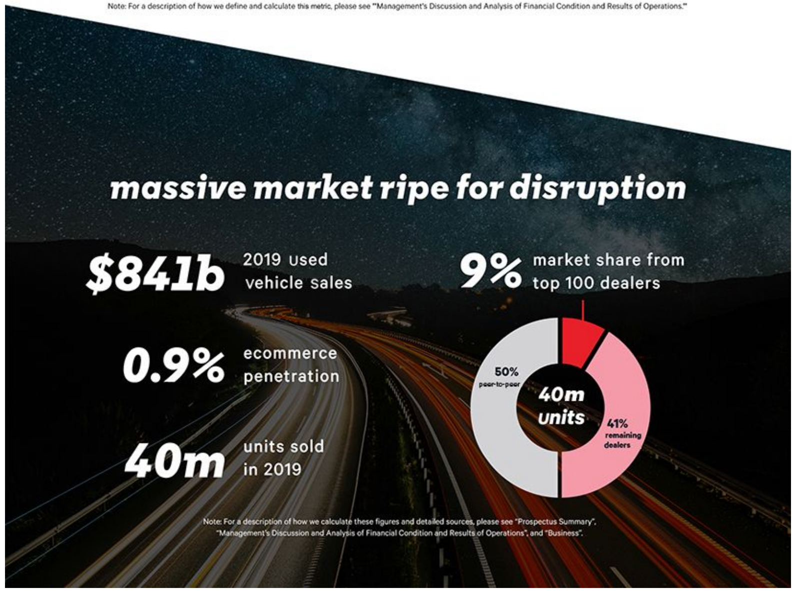
Data shown from IPO S1 filing on May 18, 2020



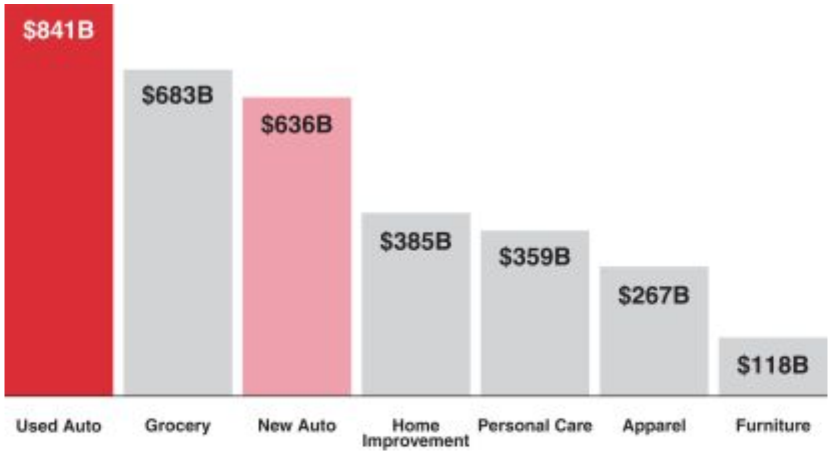
Note: For a description of how we define and calculate this metric, please see "Management's Discussion and Analysis of Financial Condition and Results of Operations."

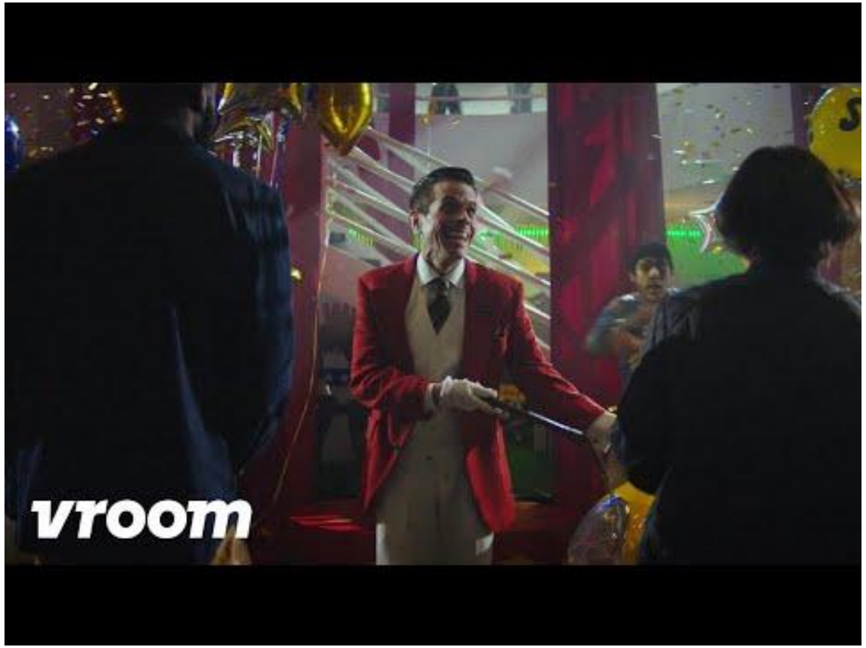


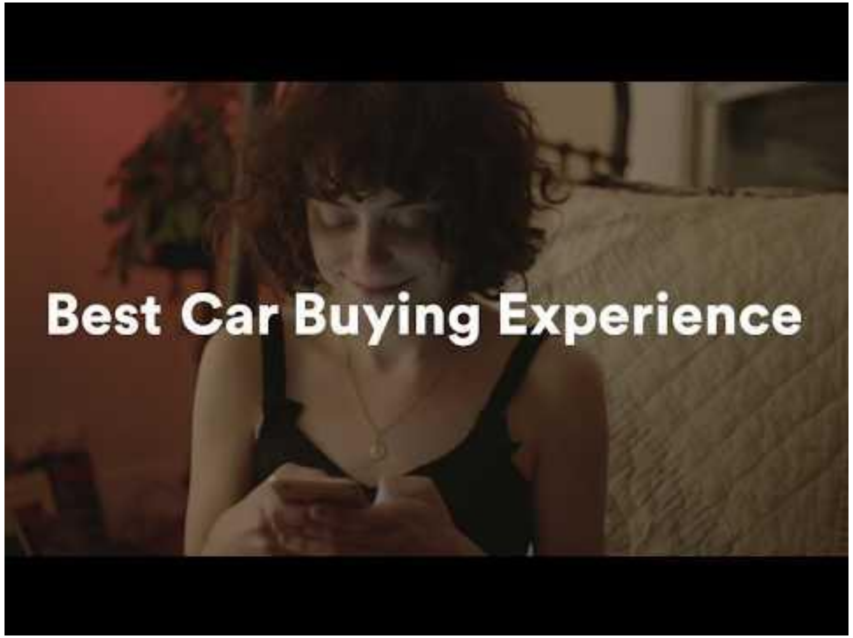
Note: Monthly figures represent operational data.



Note: For a description of how we calculate these figures and detailed sources, please see "Prospectus Summary", "Management's Discussion and Analysis of Financial Condition and Results of Operations", and "Business".




vroom


Best Car Buying Experience

Microservices

vroom

Some History

THE EVOLUTION OF

SOFTWARE ARCHITECTURE

1990's

SPAGHETTI-ORIENTED ARCHITECTURE (aka Copy & Paste)

2000's

LASAGNA-ORIENTED ARCHITECTURE (aka Layered Monolith)

2010's

RAVIOLI-ORIENTED ARCHITECTURE (aka Microservices)

WHAT'S NEXT?

PROBABLY PIZZA-ORIENTED ARCHITECTURE

By @benorama

What are Microservices

The microservice architectural style is an approach to developing a single application as a **suite of small services**, each **running in its own process** and communicating with lightweight mechanisms, often an HTTP resource API. These services are **built around business capabilities** and **independently deployable** by fully automated deployment machinery. There is a **bare minimum of centralized management** of these services, which may be written in different programming languages and use different data storage technologies.

-- James Lewis and Martin Fowler (2014)

Why microservices

- **Strong Module Boundaries:** Microservices reinforce modular structure, which is particularly important for larger teams.
    - Ownership
    - Large tech teams can operate as a small startup
    -
- **Independent Deployment**: Simple services are easier to deploy, and since they are autonomous, are less likely to cause system failures when they go wrong.
    - Improved scalability
    - Increased resilience
    - Faster time to market
    - Testability
    - CI/CD

Why microservices

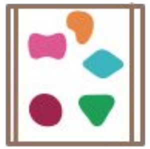
- **Technology Diversity**: With microservices you can mix multiple languages, development frameworks and data-storage technologies.
    - Experiments
    - use the right tool for the right task
    - Stay up to date
    - Evolution instead of revolution

Why not microservices

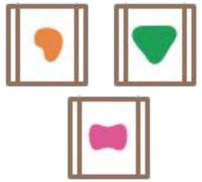
- **Distribution**: Distributed systems are harder to program, since remote calls are slow and are always at risk of failure.
  - Complexity is high
- **Eventual Consistency**: Maintaining strong consistency is extremely difficult for a distributed system, which means everyone has to manage eventual consistency.
  - laudable insistence on decentralized data management
- **Operational Complexity**: You need a mature operations team to manage lots of services, which are being redeployed regularly.
  - Robust monitoring is a must

Scaling

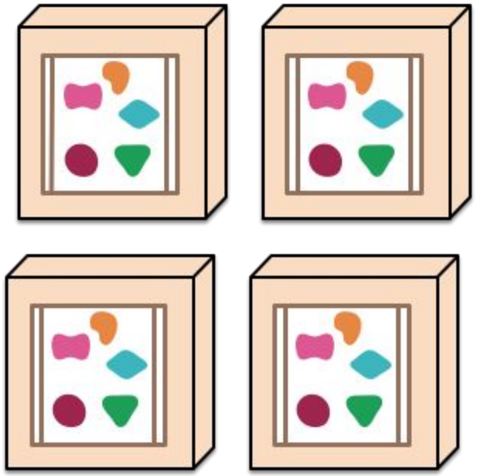
A monolithic application puts all its functionality into a single process...



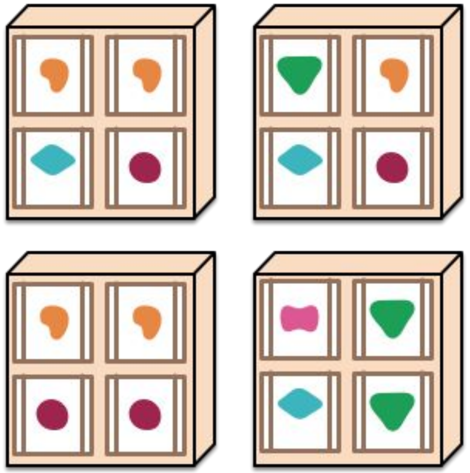
A microservices architecture puts each element of functionality into a separate service...



... and scales by replicating the monolith on multiple servers



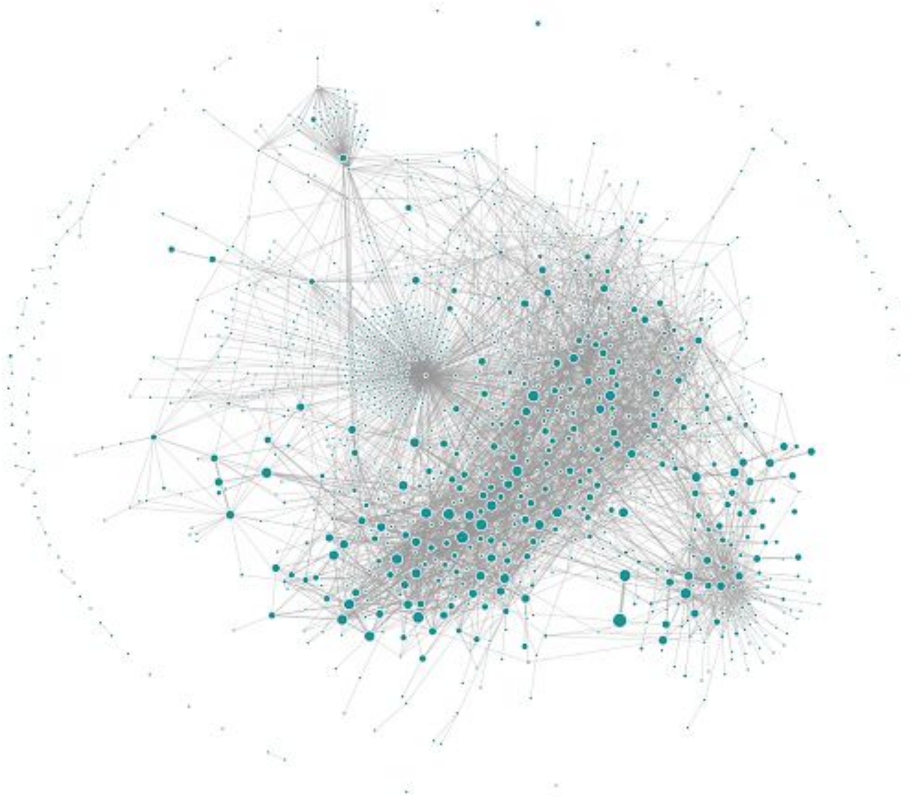
... and scales by distributing these services across servers, replicating as needed.

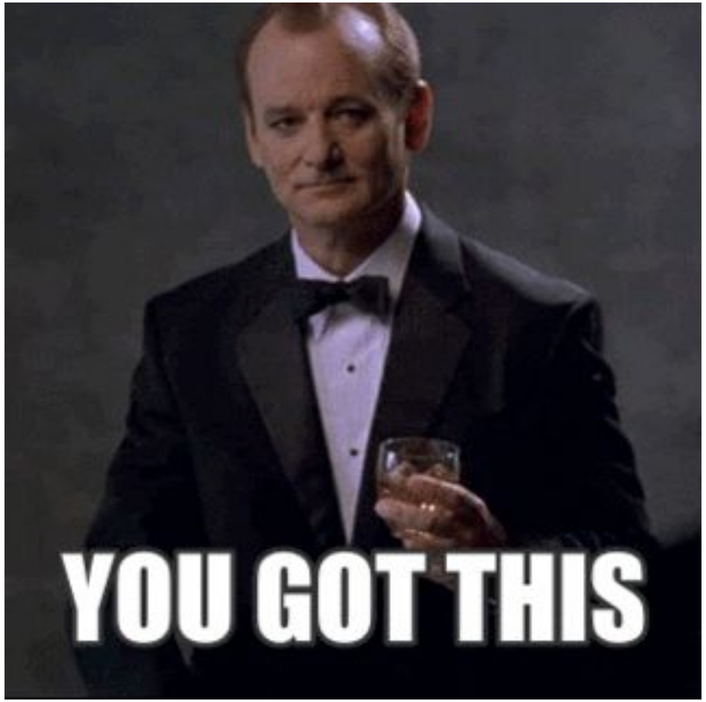


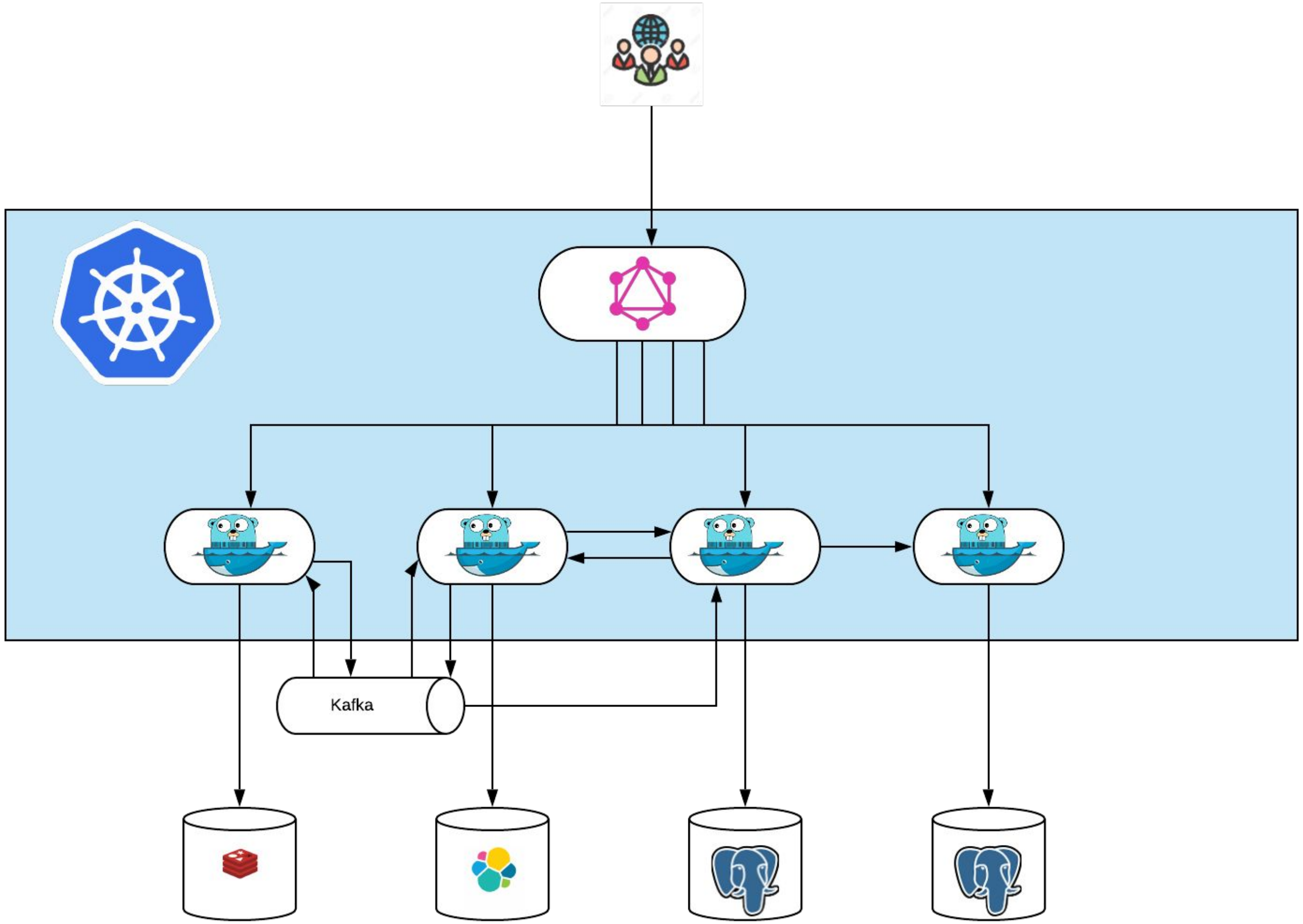
Is microservices always the right choice?

No!

Microservices in vroom

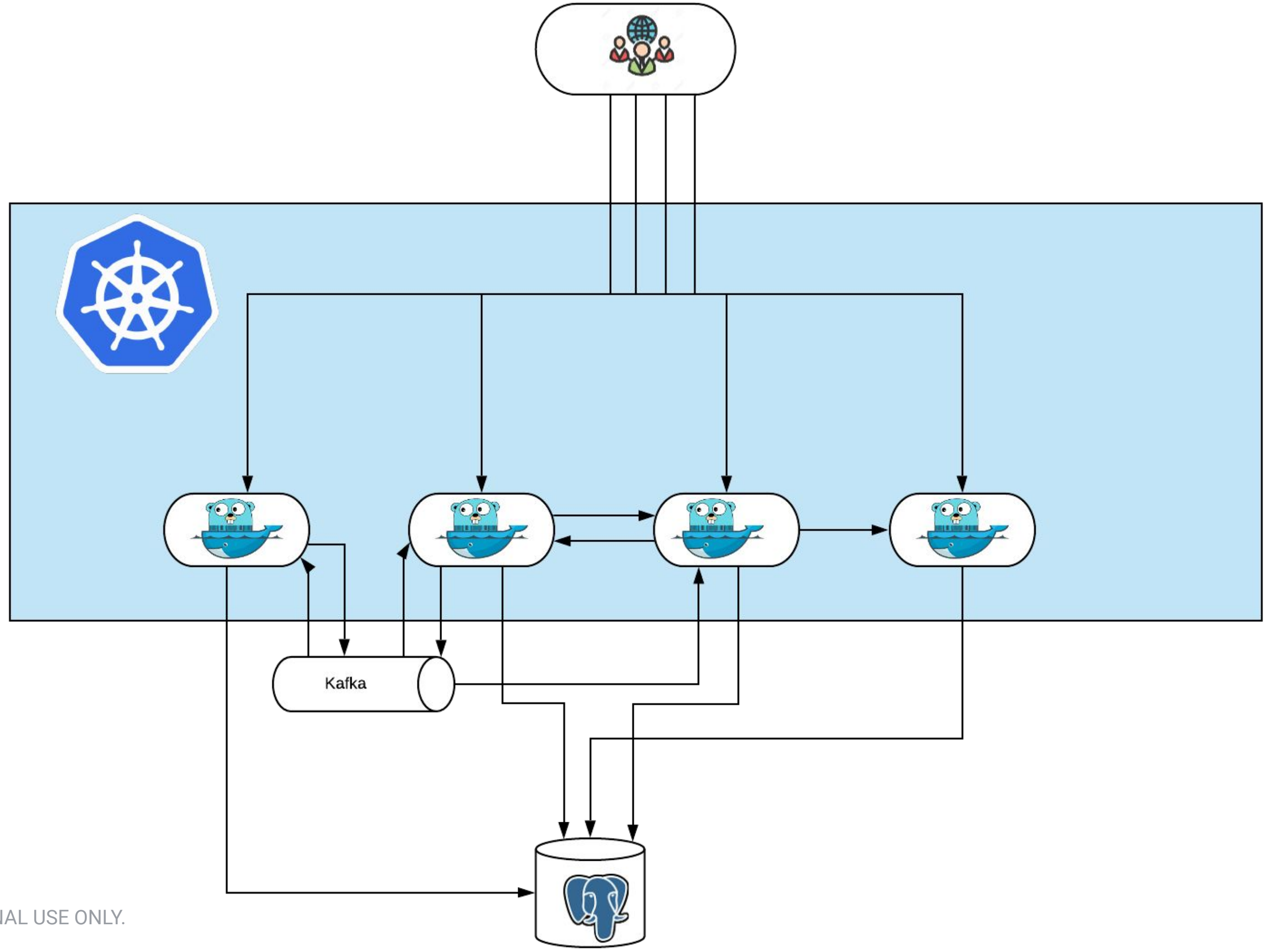




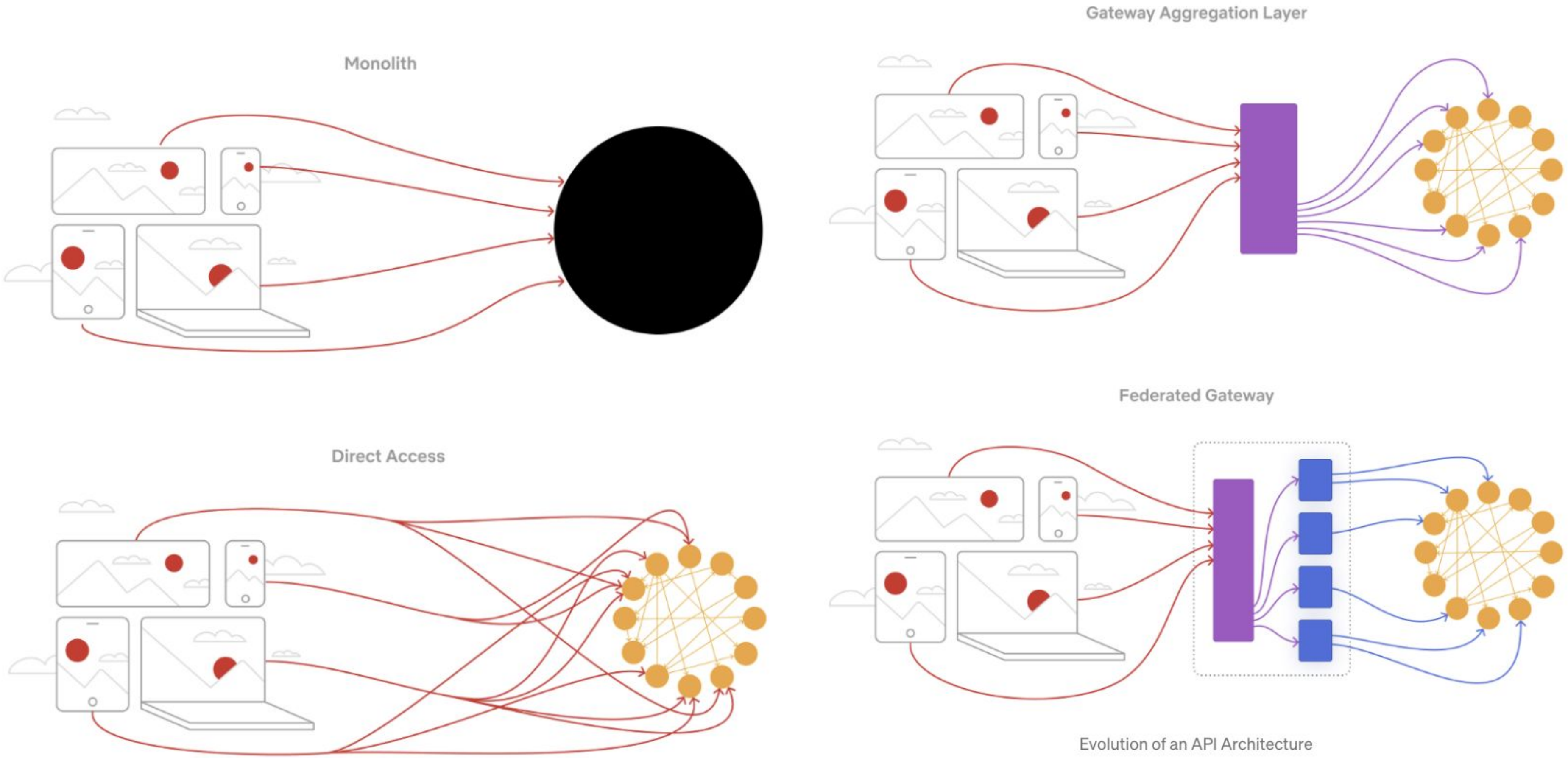

Kafka

Challenges we had

API architecture


Kafka

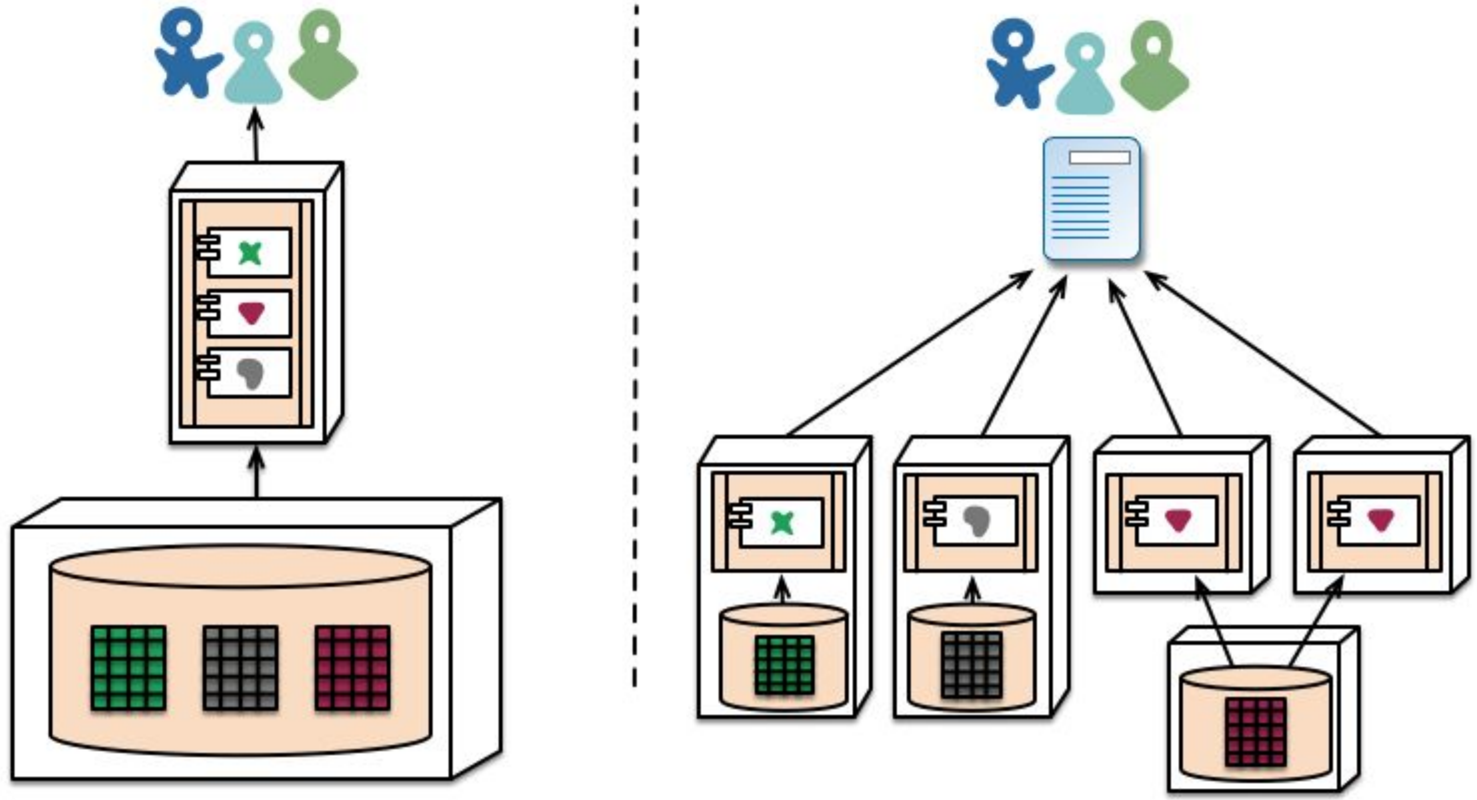
Evolution of an API Architecture



Source: https://netflixtechblog.com/how-netflix-scales-its-api-with-graphql-federation-part-1-ae3557c187e2

Challenges we had

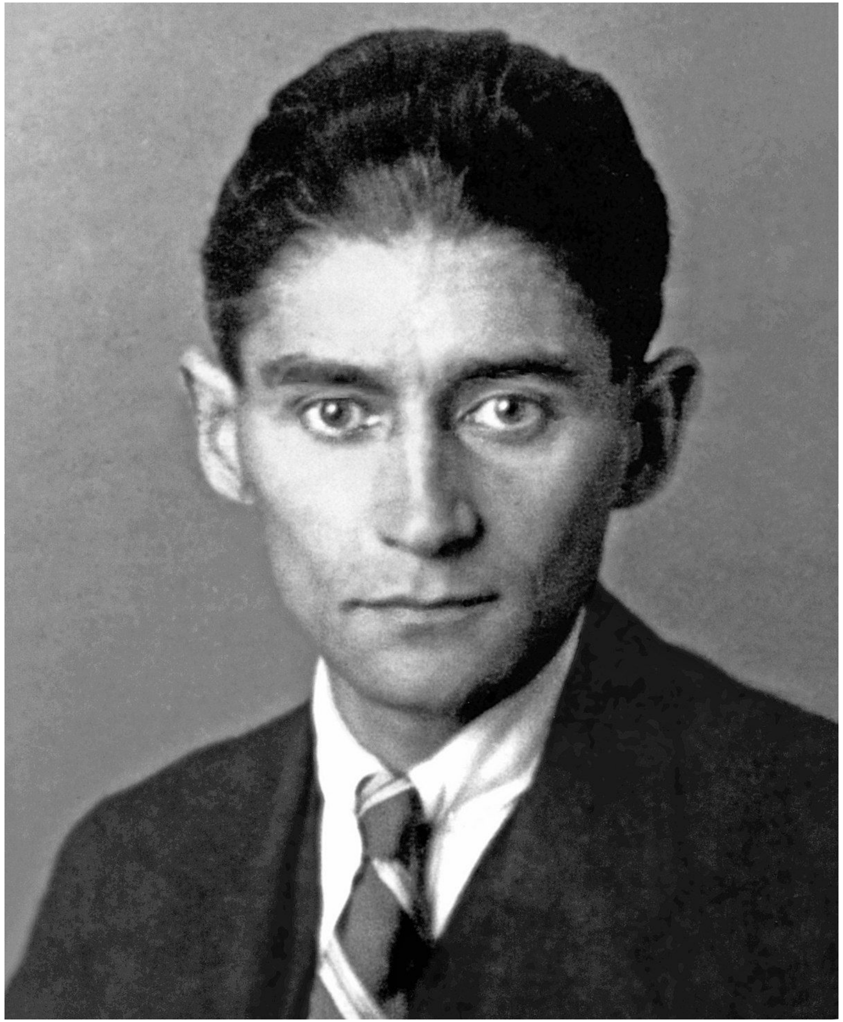
Shared Databases


monolith - single database
microservices - application databases

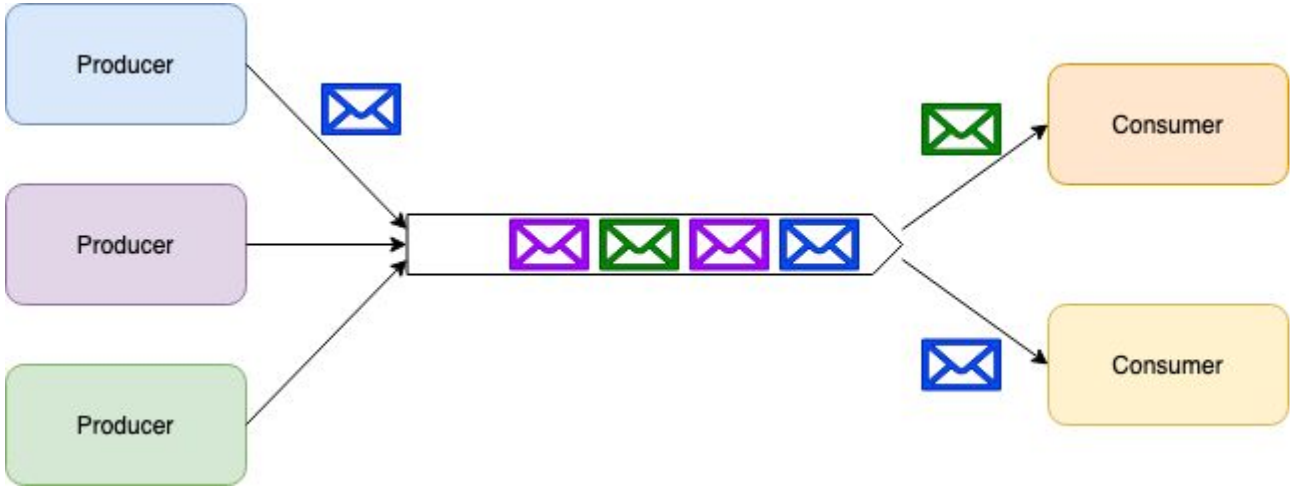
Challenges we had

Slow and unreliable 3rd party

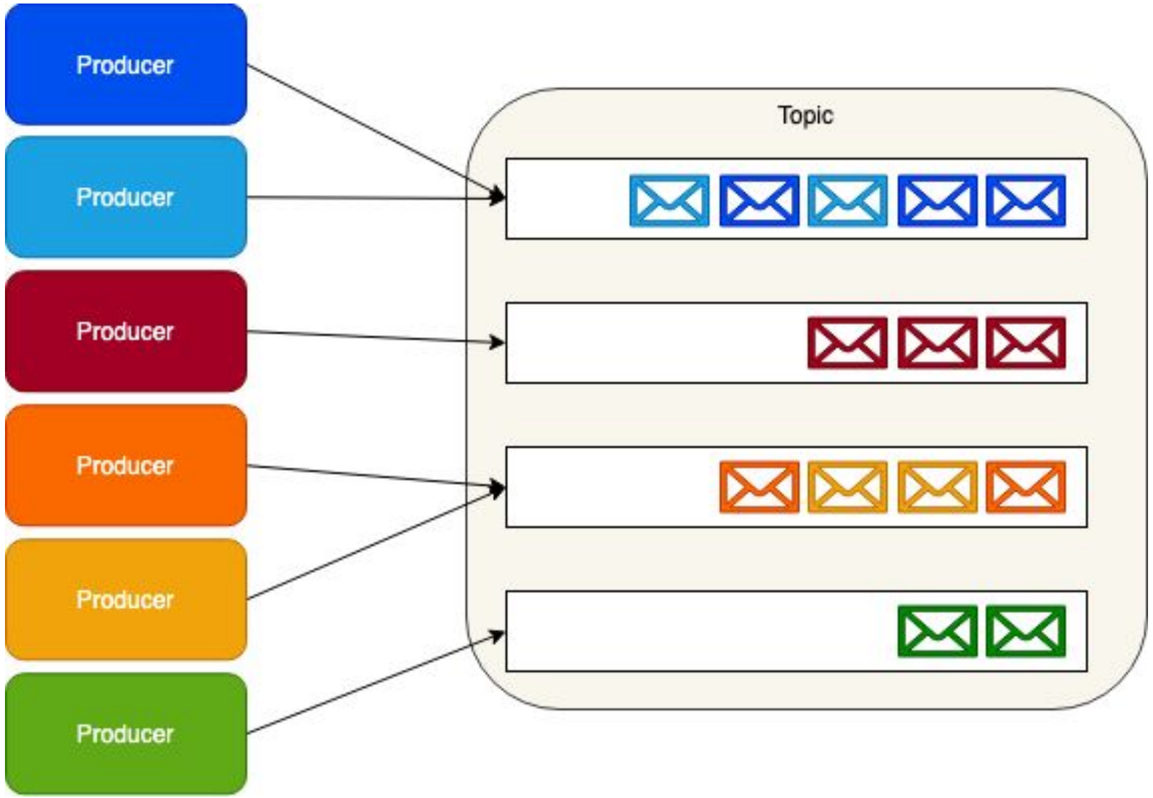
- Retry
- Throttling
- Eventual consistency



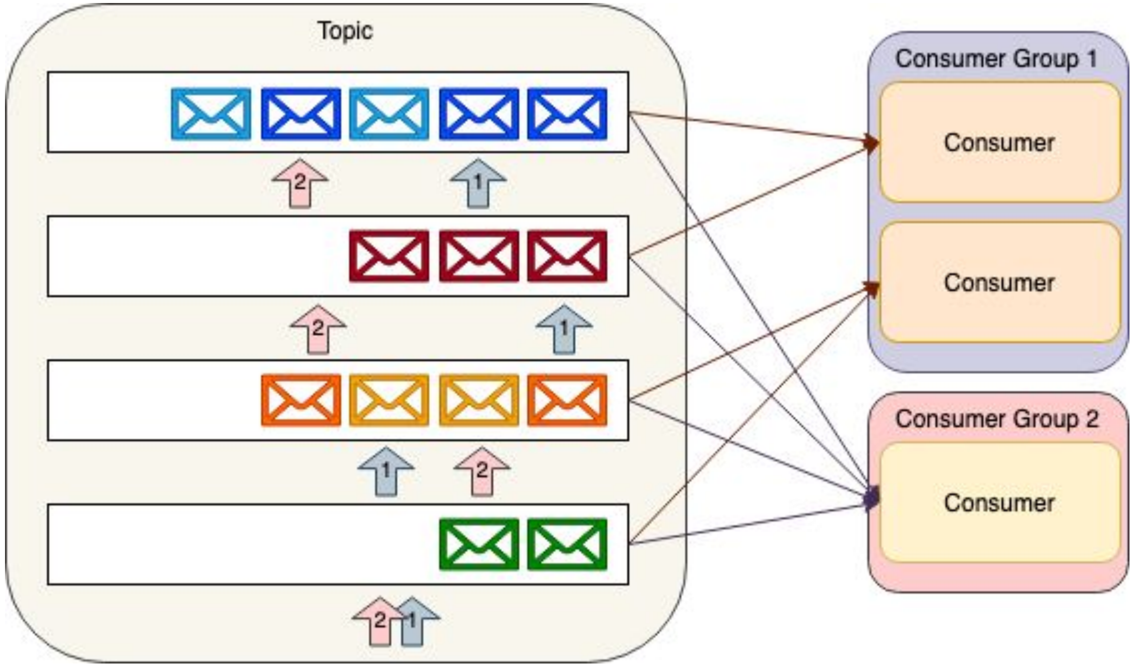
Async Message Queueing - Pub/Sub



Async Message Queueing - Pub/Sub



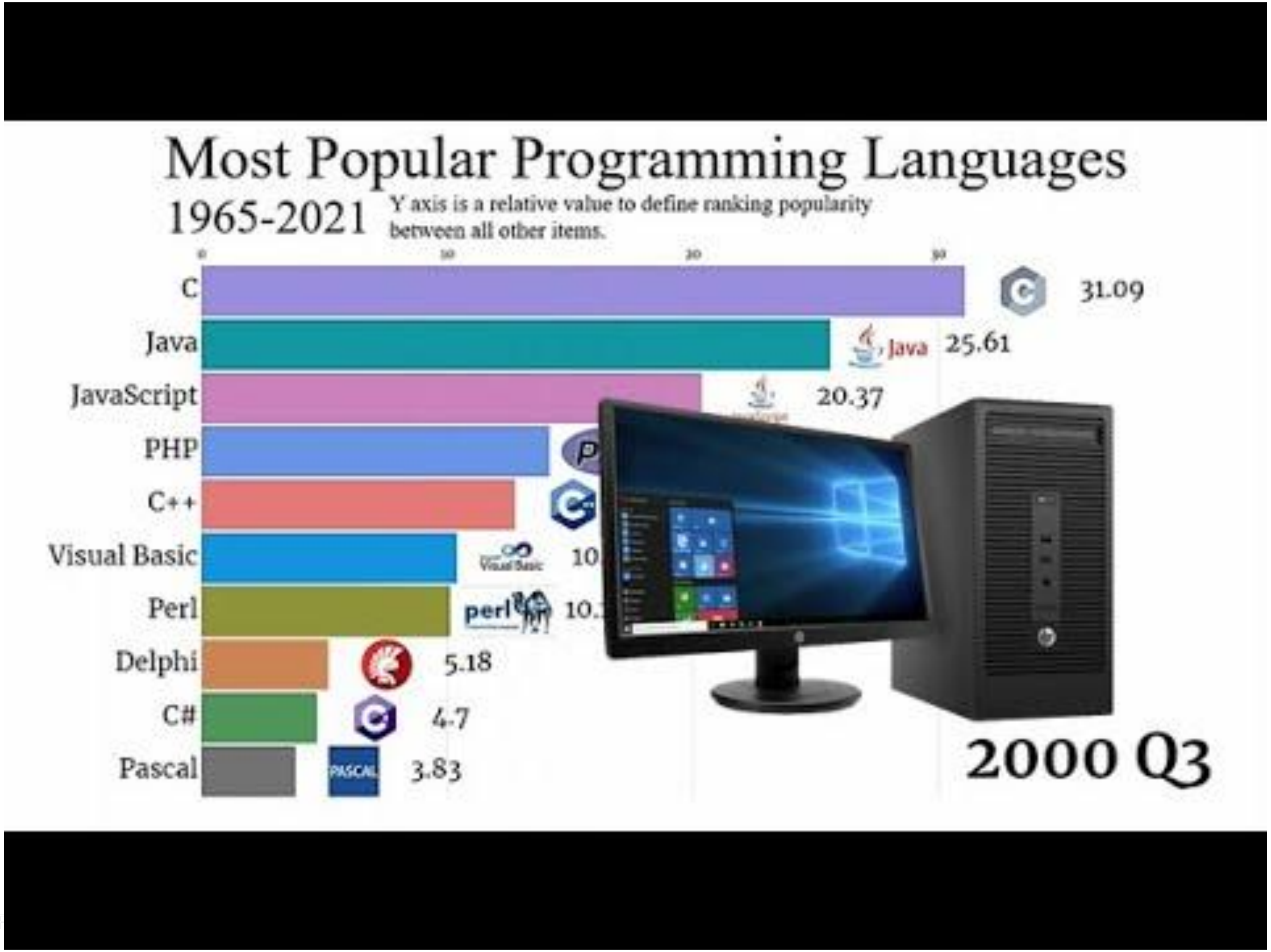
Async Message Queueing - Pub/Sub



Source:
https://stiller.blog/2020/02/rabbitmq-vs-kafka-an-architects-dilemma-part-1/

go go go!

vroom


Most Popular Programming Languages
1965-2021
Y axis is a relative value to define ranking popularity between all other items.
0
10
20
30
C
31.09
Java
Java
25.61
JavaScript
20.37
PHP
C++
Visual Basic
10
Perl
perl
10.
Delphi
5.18
C#
4.7
Pascal
PASCAL
3.83
2000 Q3

Best programming language?

Use the right tool for the right job!

About go (from Wikipedia)

Go is a statically typed, compiled programming language designed at Google. Go is syntactically similar to C, but with memory safety, garbage collection, structural typing, and CSP-style concurrency.

Go was designed at Google in 2007 to improve programming productivity in an era of multicore, networked machines and large codebases. The designers wanted to address criticism of other languages in use at Google, but keep their useful characteristics:

static typing and run-time efficiency (like C),

readability and usability (like Python or JavaScript)

high-performance networking and multiprocessing.

The designers were primarily motivated by their shared dislike of C++.

Go was publicly announced in November 2009, and version 1.0 was released in March 2012. Go is widely used in production at Google and in many other organizations and open-source projects.

https://blog.jetbrains.com/go/2021/02/03/the-state-of-go/

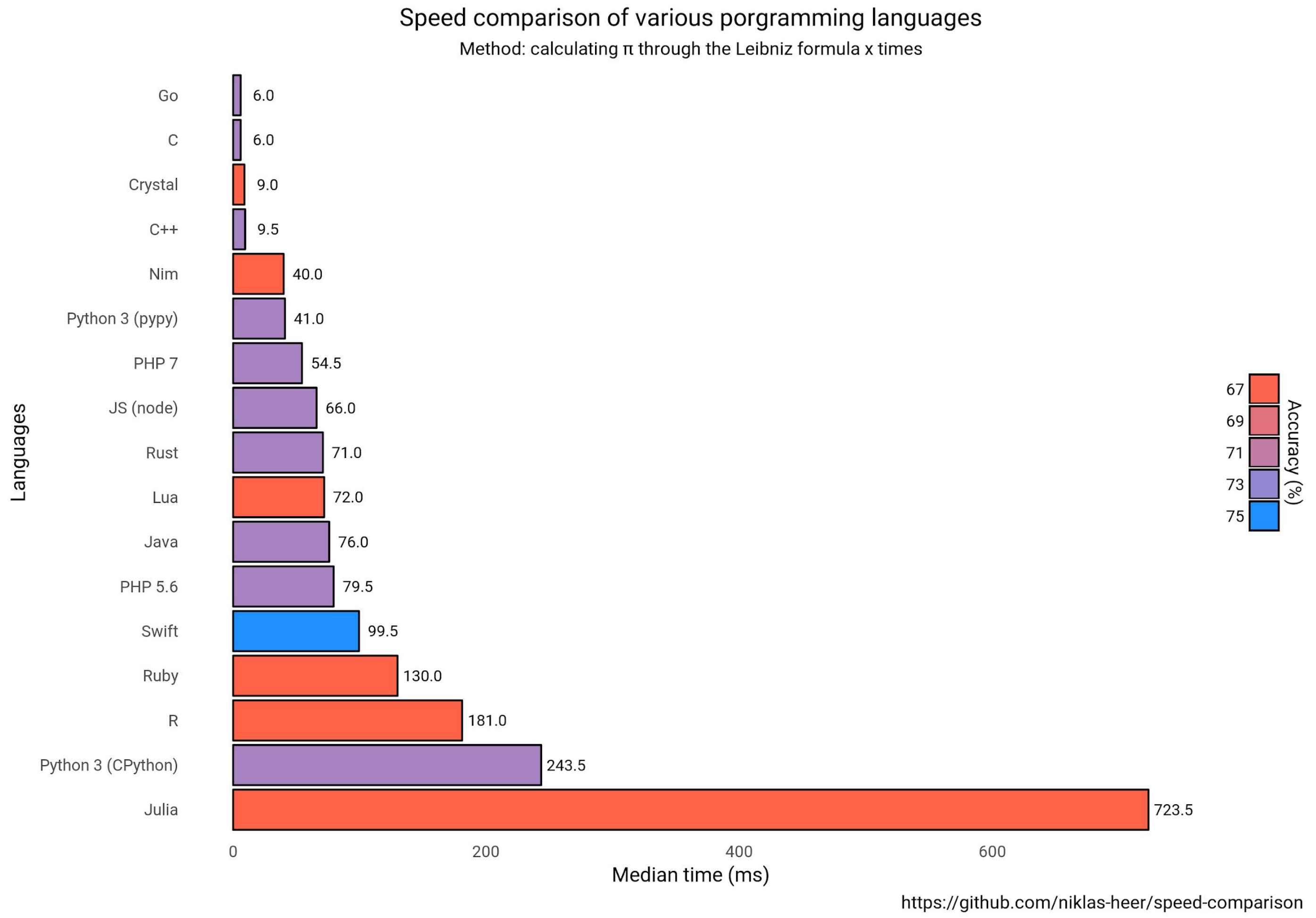
Why go is great for vroom?

Go language is small and simple

Go is meant to be simple to learn, straightforward to work with, and easy to read by other developers. Go does not have a large feature set, especially when compared to languages like C++

Speed

Go binaries run more slowly than their C counterparts, but the difference in speed is negligible for most applications. Go performance is as good as C for the vast majority of work, and generally much faster than other languages known for speed of development (e.g., JavaScript, Python, and Ruby)



Source: https://github.com/niklas-heer/speed-comparison

Why go is great for vroom?

concurrency

Distributed networked services. Network applications live and die by concurrency, and Go’s native concurrency features — goroutines and channels, mainly—are well suited for such work.

Portability

Cloud-native development. Go’s concurrency and networking features, and its high degree of portability, make it well-suited for building cloud-native apps.

Why go is great for vroom?

Low cost for cloud applications

Small runtime, small memory footprint, great utilization of the resources

Interoperability

Go delivers without sacrificing access to the underlying system. Go programs can talk to external C libraries or make native system calls.

What else is written in go

- google's backend
- Uber's backend
- Dropbox's backend
- Docker
- Kubernetes
- Fedora CoreOS
- InfluxDB
- CockroachDB
- Terraform
- Istio

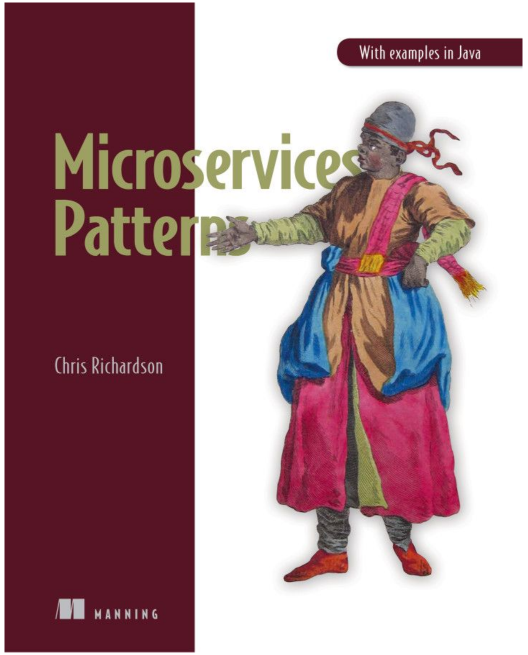
This was just an intro...

https://martinfowler.com

https://microservices.io

https://netflixtechblog.com/

https://eng.uber.com/



We are hiring

https://www.vast.com/vroom-careers-positions

For questions and comments ping:

https://www.linkedin.com/in/nodelman/

https://www.linkedin.com/in/ilijaduni/