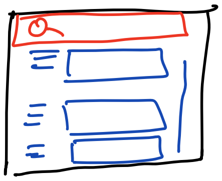
Agenda

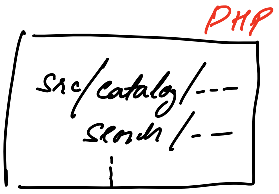
- one
- Monolith
  - Advantages & Disadvantages
- sm
- Microservices
  - Communication
  - Distributed Transactions
  - Advantages & Disadvantages
- When to use what

Amazon — 1994

- online **book** selling — webpage ≡ books
  - HTML
  - Phone number
- Online e-commerce system → Service Oriented Architecture (SOA)
  - catalog service
  - search service
  - cart service
  - order service
  - payment service
  - logistics service
  - user service / auth



└ inventory management



Single Codebase with multiple modules

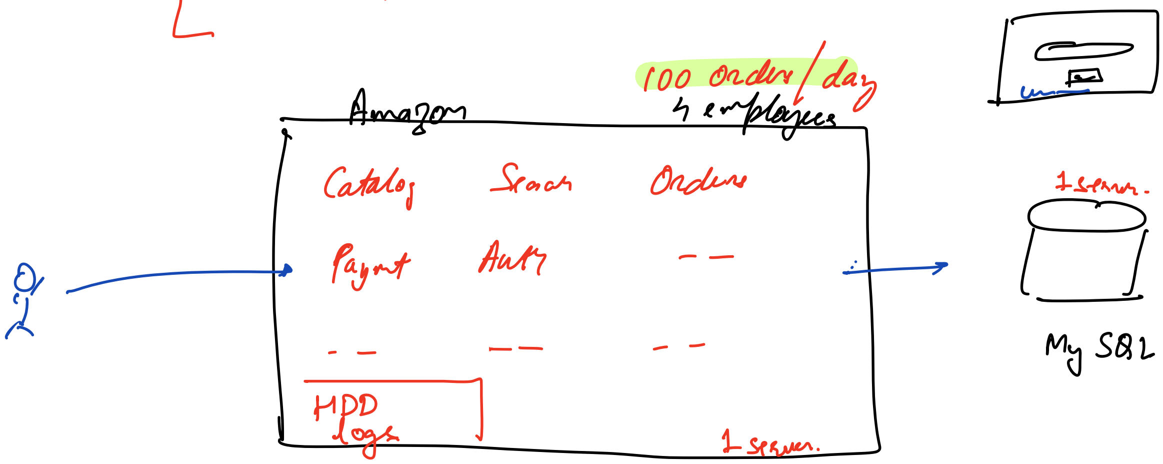
One common database

Communication → a function call

a lot of money ┬ for deploying?
└ to hire experts?
└ time? (dev for 2 years — launch)

- iterate fast / fail fast
- product-market fit
- time to market (v.v. important)
  - └ Swiggy (wordpress)
  - └ excel sheets
  - └

google ┬ Trillion $
└ best talent
└ Compute
└ political
└ brand



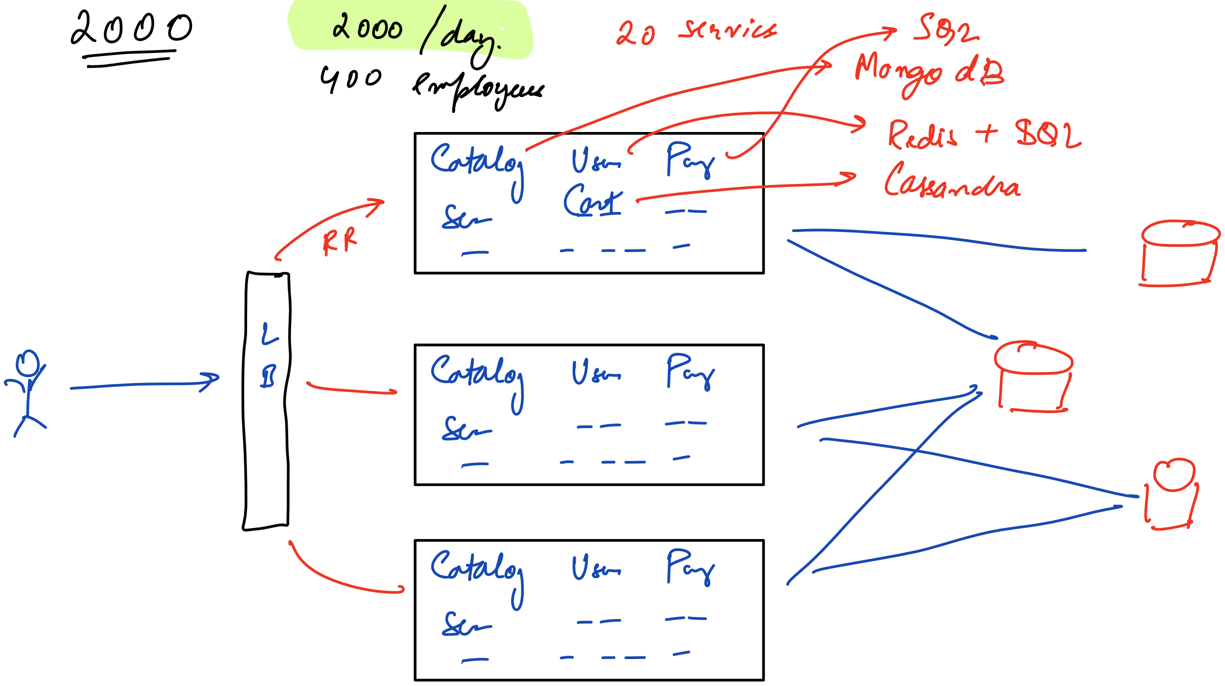
Advantages of Monolith

- simple to code & get started with
- Simple architecture → simple to deploy
- a single person can do this
- easy to test & debug.
- good performance due to direct DB connection
- easy maintainance & updates

2000

2000 /day.

400 employees



2015

2000 orders/min

90,000 employees → 2000 programmers.

200 services

Cons of Monolith

- making changes
  - codebase bloat
  - technical debt — big ball of mud
- testing
  - single function can have hundreds of uses
  - isolating bugs
- requires down time
  - hours to deploy simple changes
  - many deployment per day → impossible
- Services are tightly coupled
  - One db technology → locked into it
  - scale independently
    - each service has to be full fledged
- developer experience
  - Onboarding new devs — nightmare
  - all devs must be experts

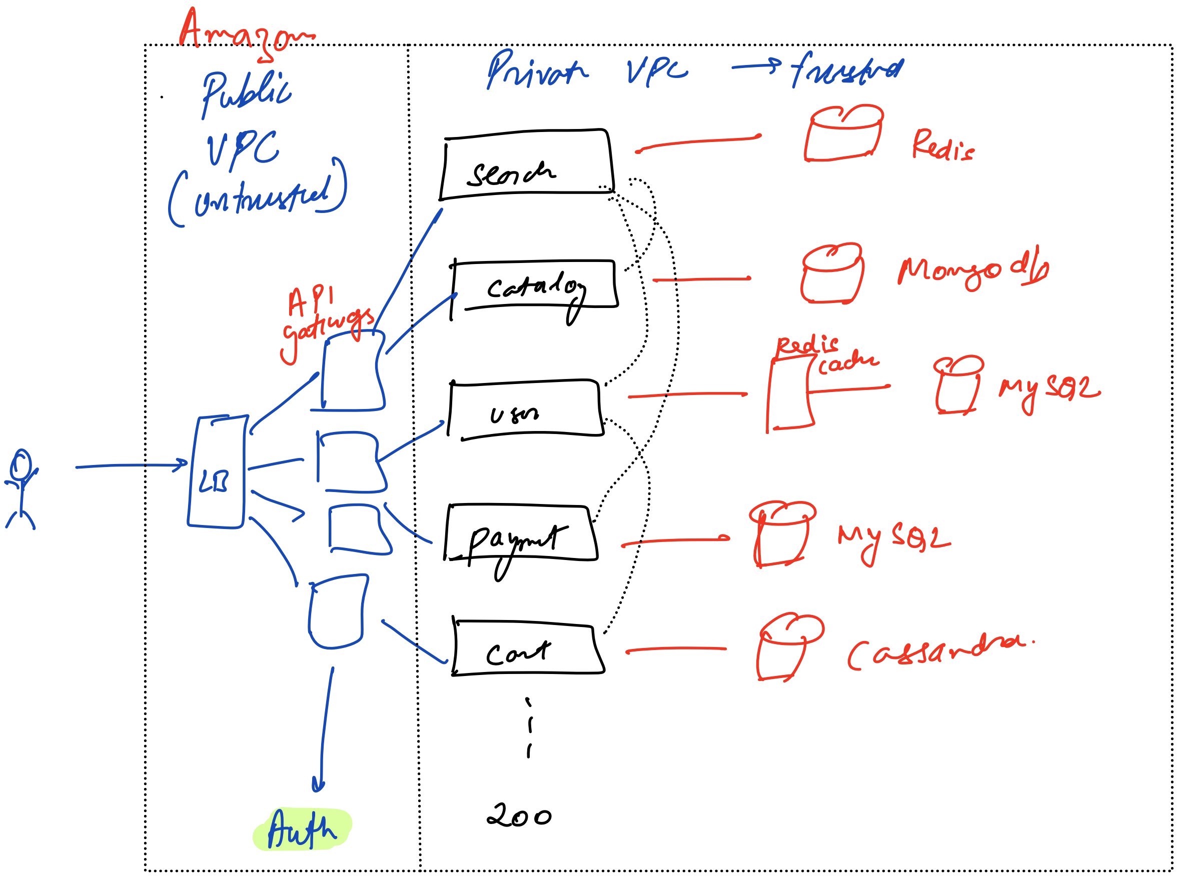
- employees leaving company.

:($\xrightarrow[\text{migrate}]{}$:)

Monolith is an architecture in which all code lives together, sharing a database & is tightly coupled.

How to migrate

- identify the different services
- how much load does each service have?
  - one which have large load
    - easiest ones (lowest hanging fruit)



Internet (global) → dangerous

Communication

└ sync / async

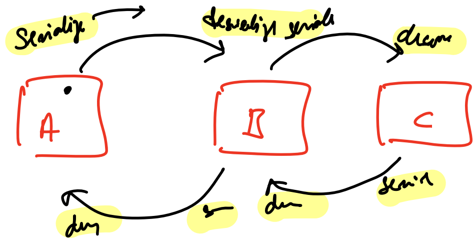
① REST APIs
   └ simple
   └ battle tested
   └ high payload size
   └ slow

application / json
application / octet-stream

JSON

XML

{ "user id": 123,
  ...
}



List < Double > = { 3.1415 - - , 2.7 - - , 299792458 }

8 byte
64 bit

8 8 8

= 24 bytes.

" [3.1415 . . . , 2.7 - - - ; 299792458] "

2 byte

50

100 bytes

→ IPC → Inter process Communication

② RPC → Remote procedure calls

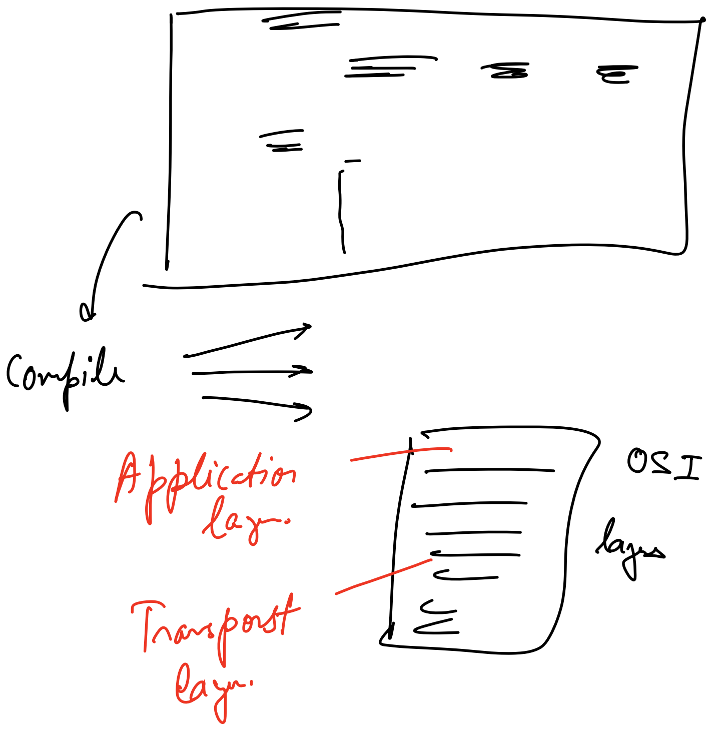
gRPC — protobuf (typed interface)

↳ google

↳ binary encoding

↳ compiled schema

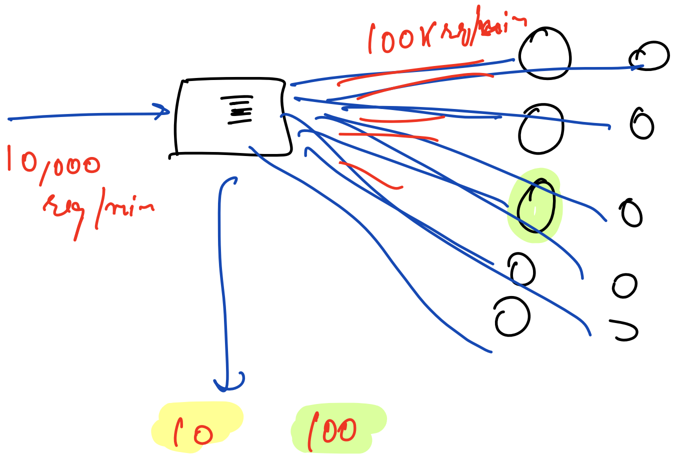
- .proto



Pain points → services need to be aware of each other

↳ API

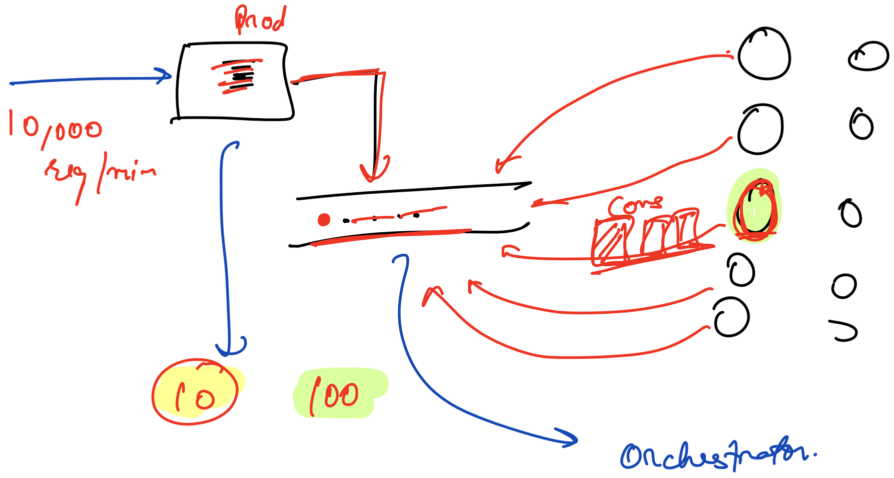
↳ Scaling might become dependent.



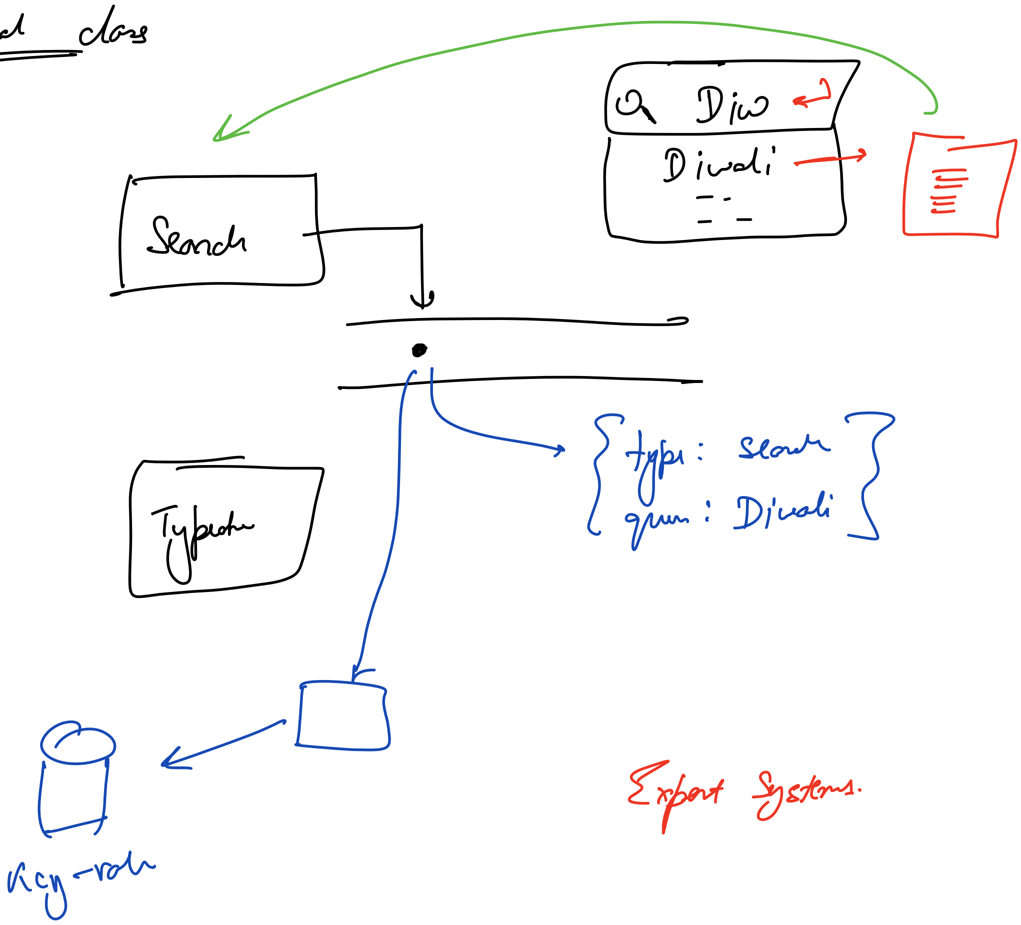
Event Driven Architecture

③ Events

- most flexible
- most scalable
- most popular (for large scale companies)
- async



Typeahead class



Expert Systems.

10:40 → 11:05 pm

└ 10 mins   Mind.wiki

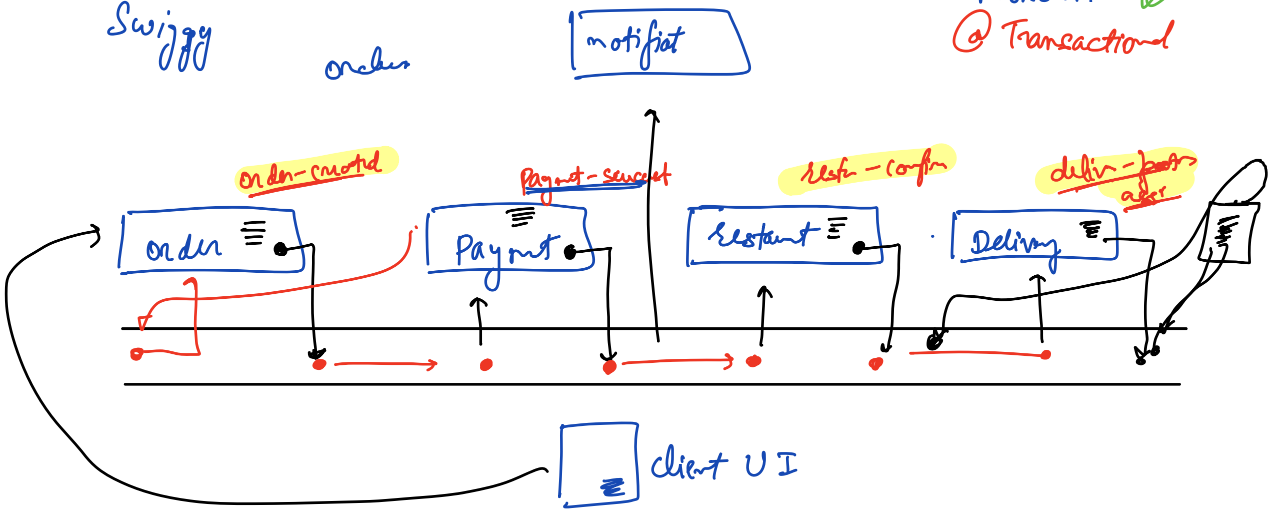
Distributed Transactions

→ 2 phase Commit
(sync)

Swiggy

orders

Monolith ✓
@Transactional



Saga Pattern (~~SAGA~~)

→ Story

Sagas
Epics
Chapters

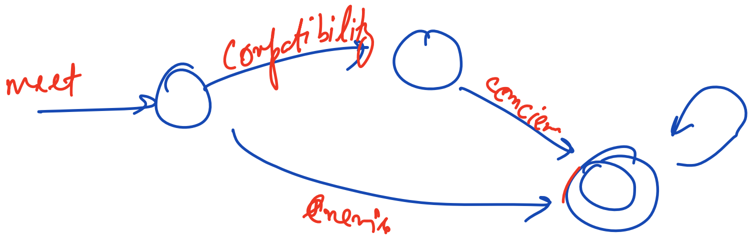
① Orchestration
② Choreography

} v.v.v similar

↳ distributed (no central controller)

Orchestrator which owns the transaction

State machine



sexual reproduction

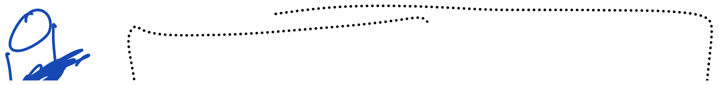
Rollbacks

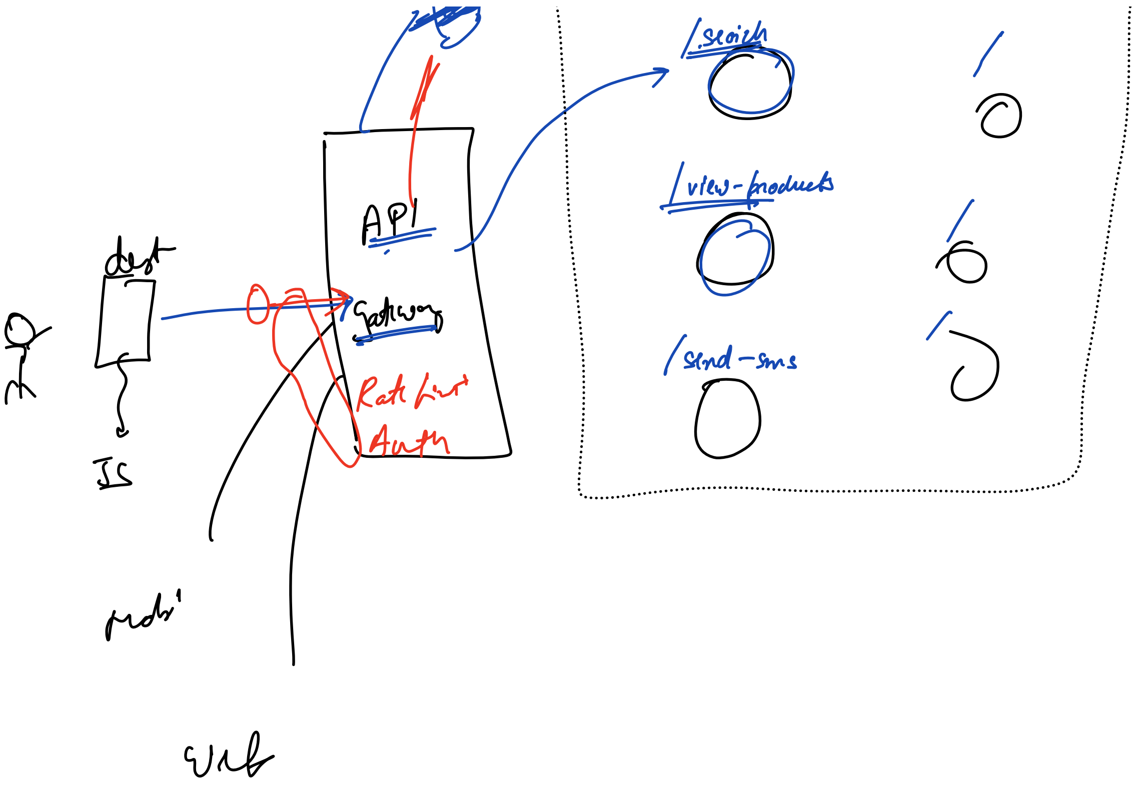
→ emit failure events (compensating transactions)

SaaS → software as service

BaaS → Backend as Service

- → db → REST
  - → Realtime
- Auth
- Notification
- Edge Functions





① allows frontend to create data products without dependence on backend team

② solves n+1 query problem.

/user/id/courses

/users	/user/id
/courses?fi--	/cours/id
/lecture	/lect/id

```
{                      }

{
    users : {
                id : ---,
                course : { id :
                           name :
                           title :      }
            ]
}
```

find all emails of all friends of Hemanth.

① find list of friends of hemanth → [ ]   1

n

② loop over list:

email of friend.

n+1

```
{ user : { id :
           frin : { email :  }
         }
}
```