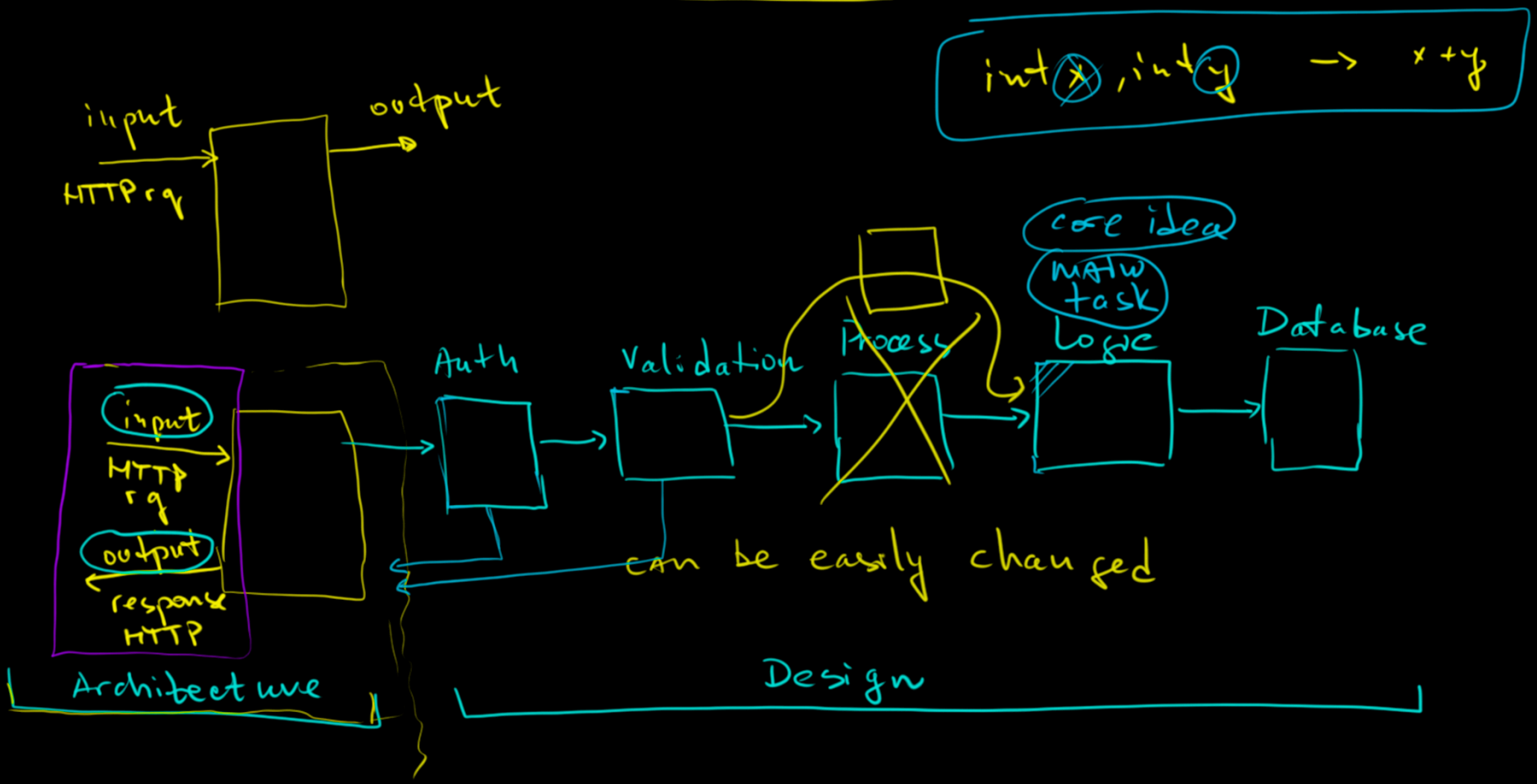
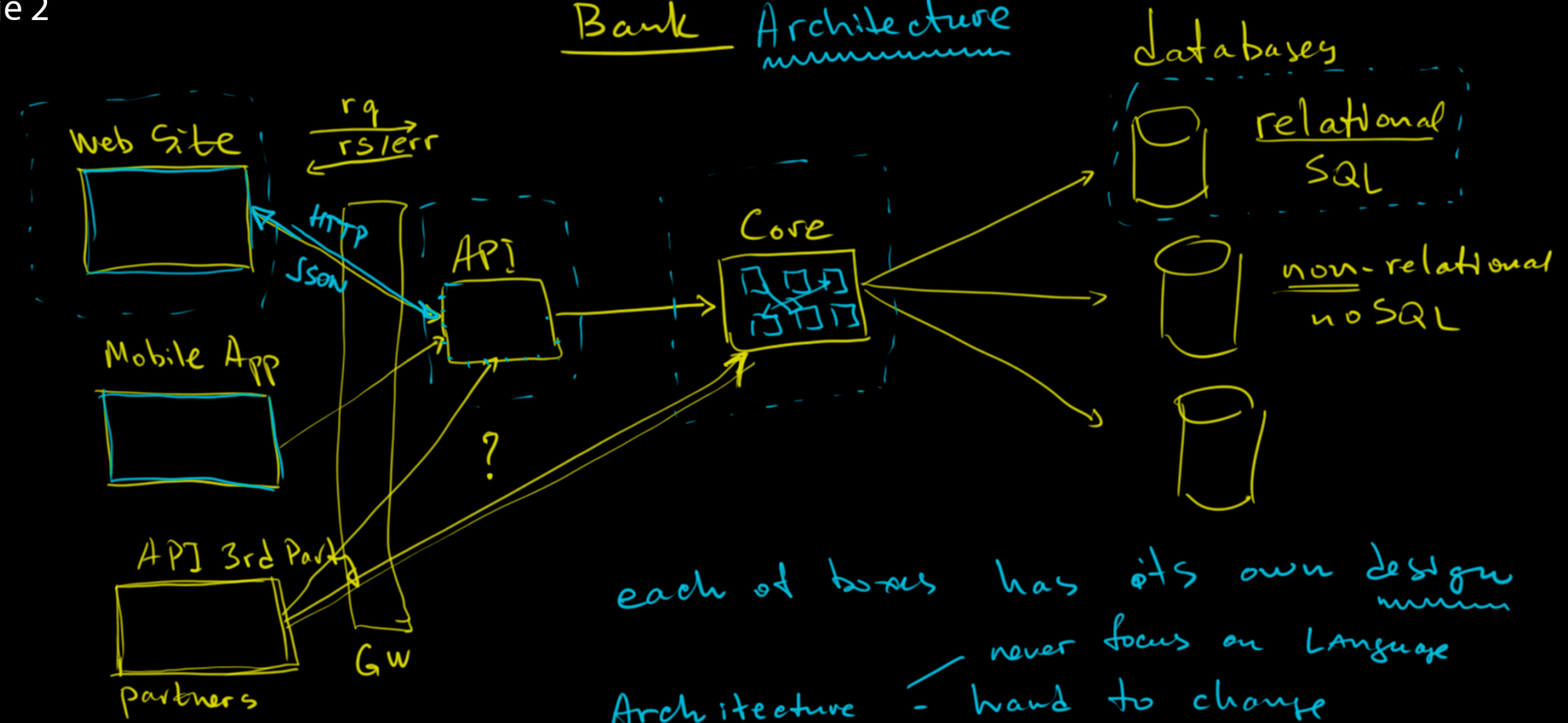
Architecture



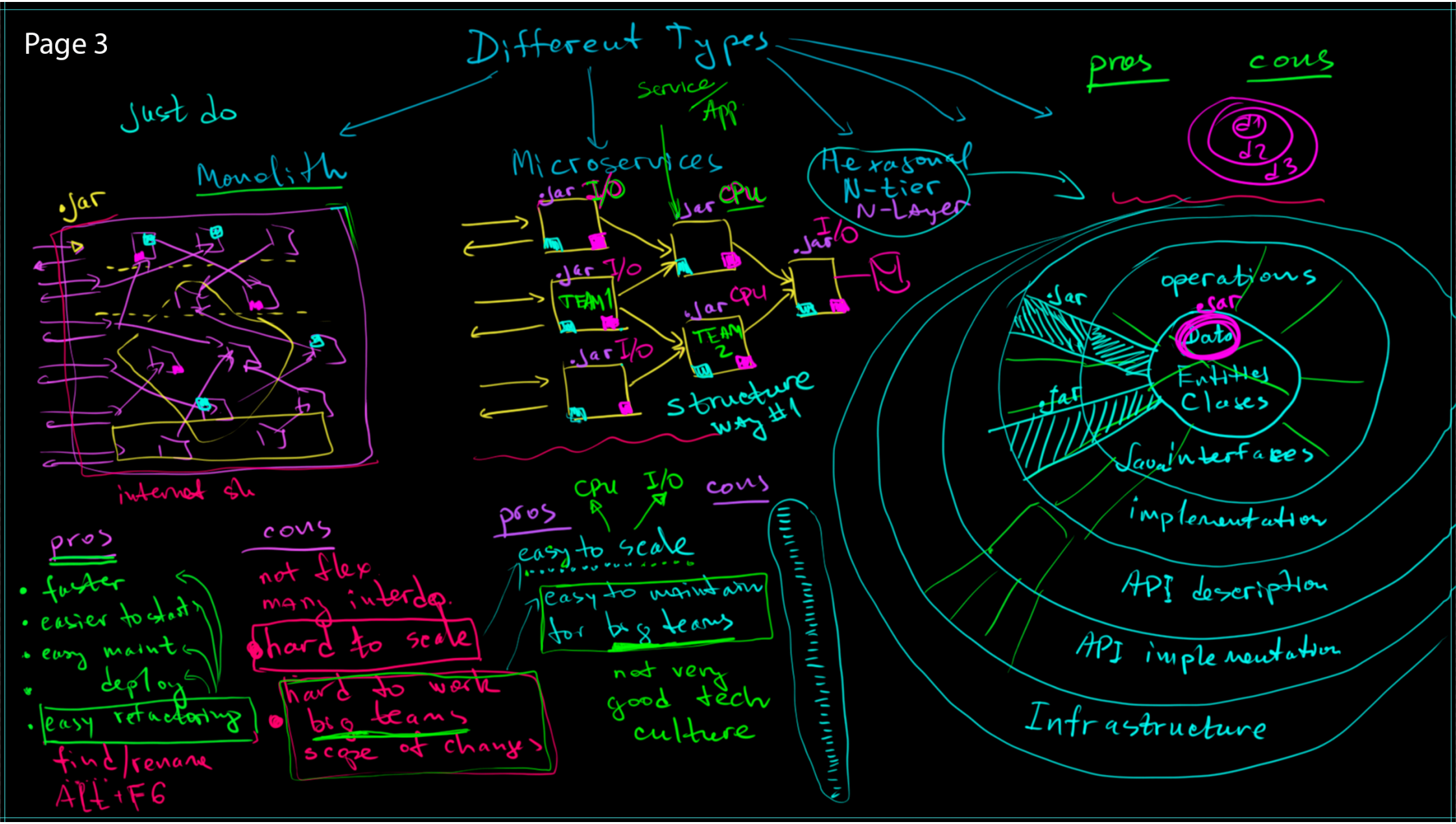
Bank Architecture

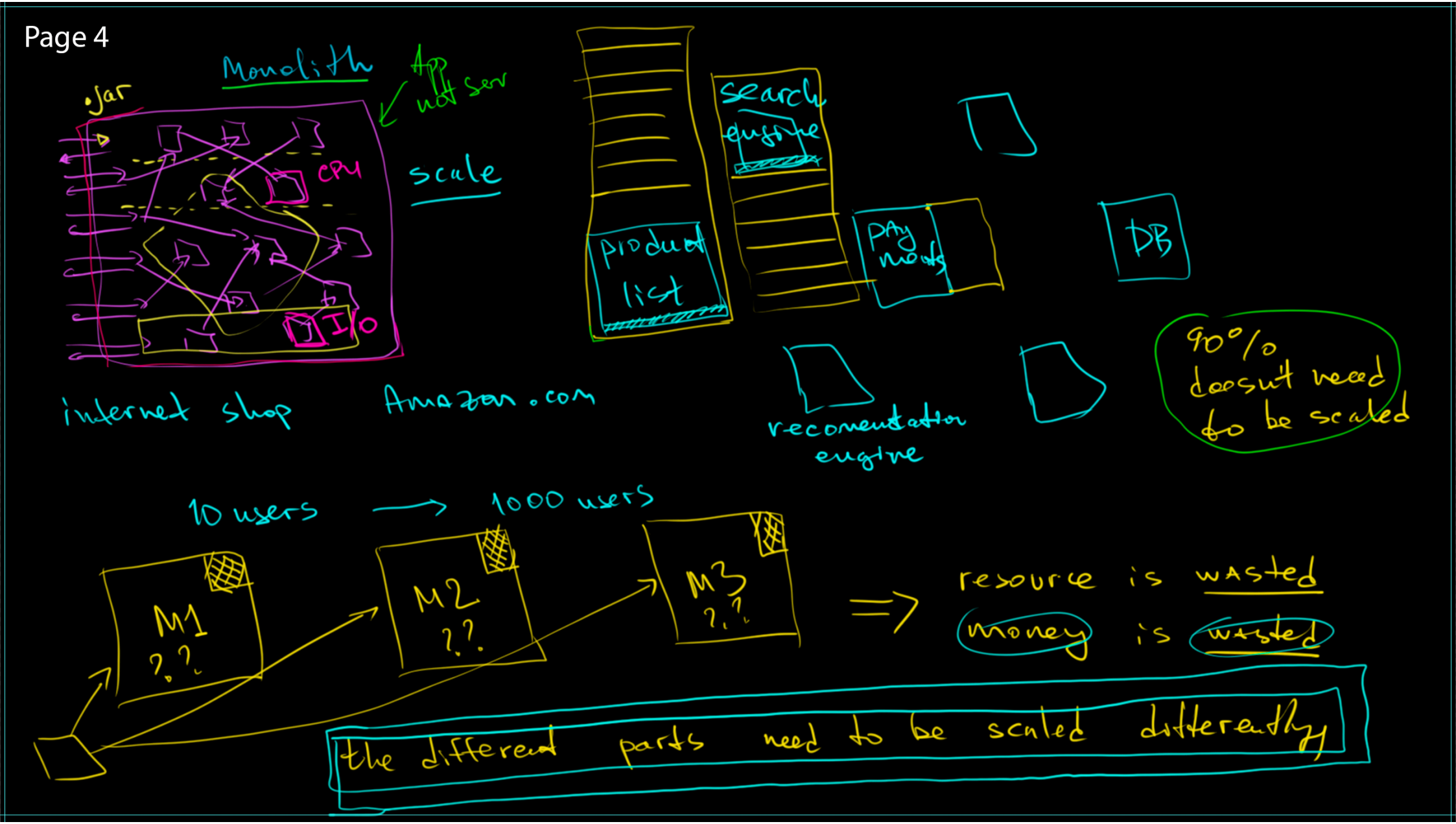


each of boxes has its own design

Architecture - never focus on Language
Design - hard to change
easy

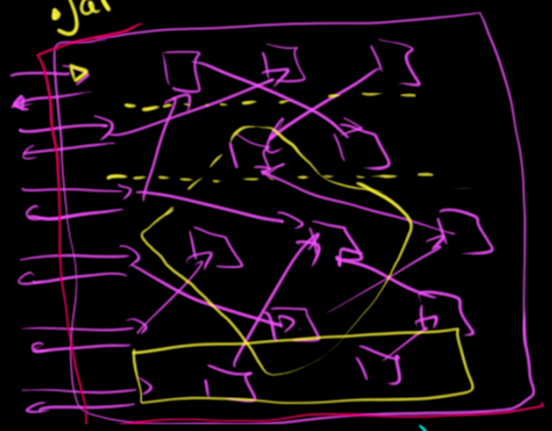
for deep integration


Different Types
Just do
Monolith
.jar
internet sh
pros
faster
easier to start
easy maint
deploy
easy refactoring
find/rename
Alt+F6
cons
not flex
many interdep.
hard to scale
hard to work
big teams
scope of changes
Microservices
Service / App
.jar I/O
.jar CPU
TEAM 1
TEAM 2
structure way #1
CPU I/O
pros
easy to scale
easy to maintain for big teams
not very good tech culture
cons
Hexagonal N-tier N-Layer
pros
cons
d1
d2
d3
operations
.jar
Data
Entities Clases
Java interfaces
implementation
API description
API implementation
Infrastructure


Monolith
.jar
App not serv
CPU
I/O
scale
internet shop
Amazon.com
product list
search engine
Pay
DB
recomendation engine
90% doesn't need to be scaled
10 users
1000 users
M1
??
M2
??
M3
??
resource is wasted
money is wasted
the different parts need to be scaled differently

Monolith

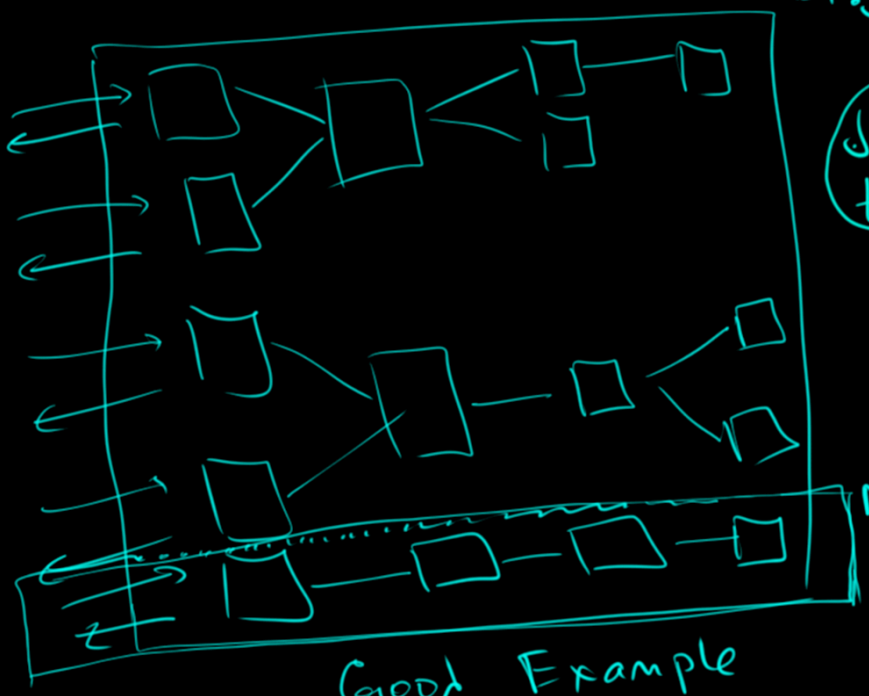
.jar



Bad Example

Structured Moholith

MS1.jar



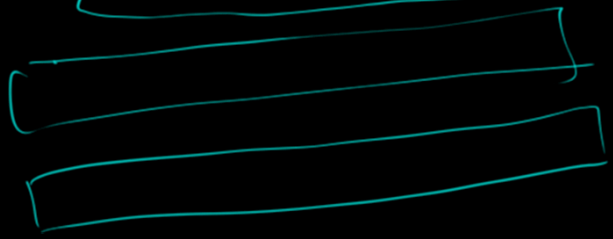
MS2.jar

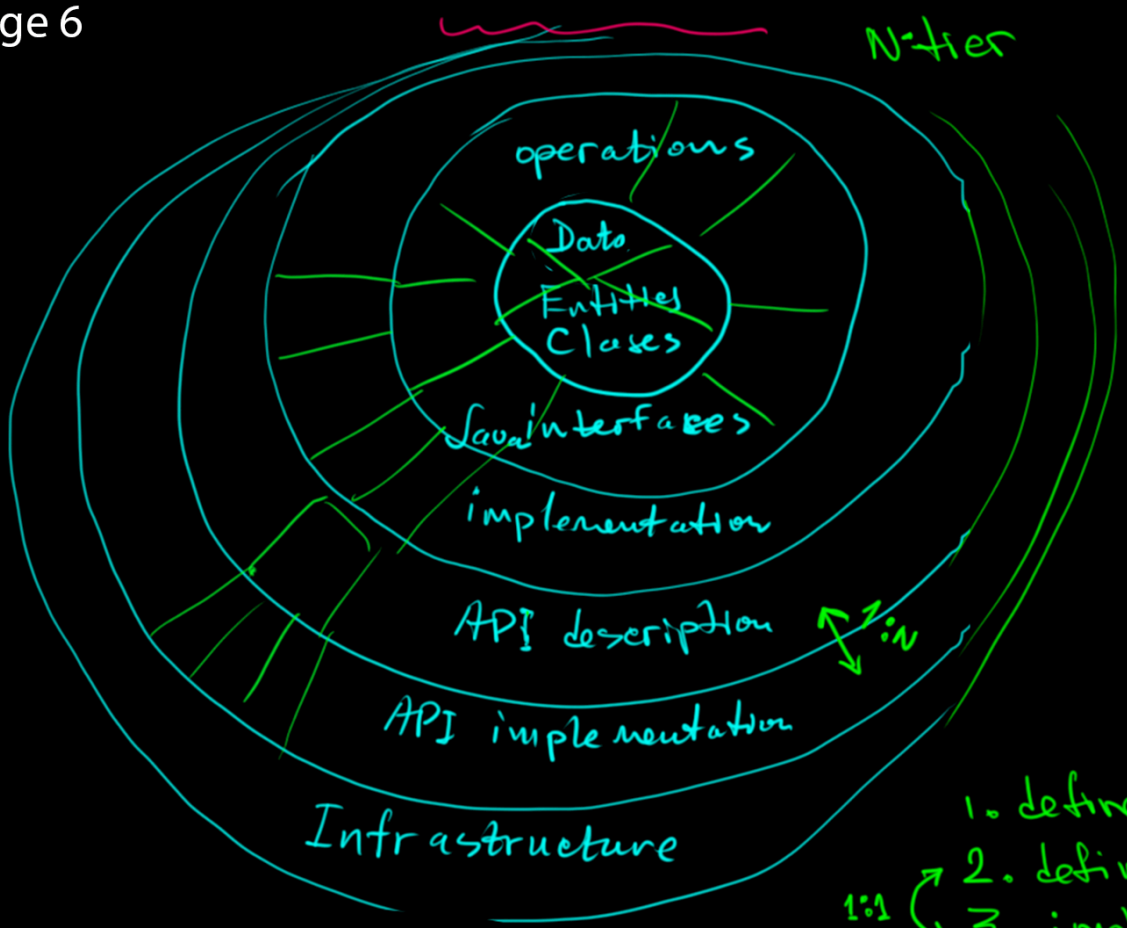
Just do try to structure

Good Example

more than 1 considered to be microservice

How small MS should be.



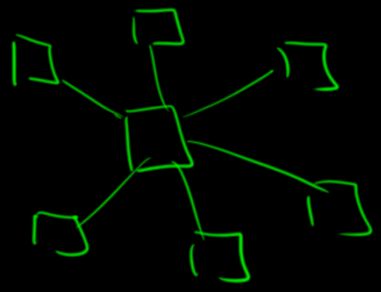


structured in another way #2

1:1

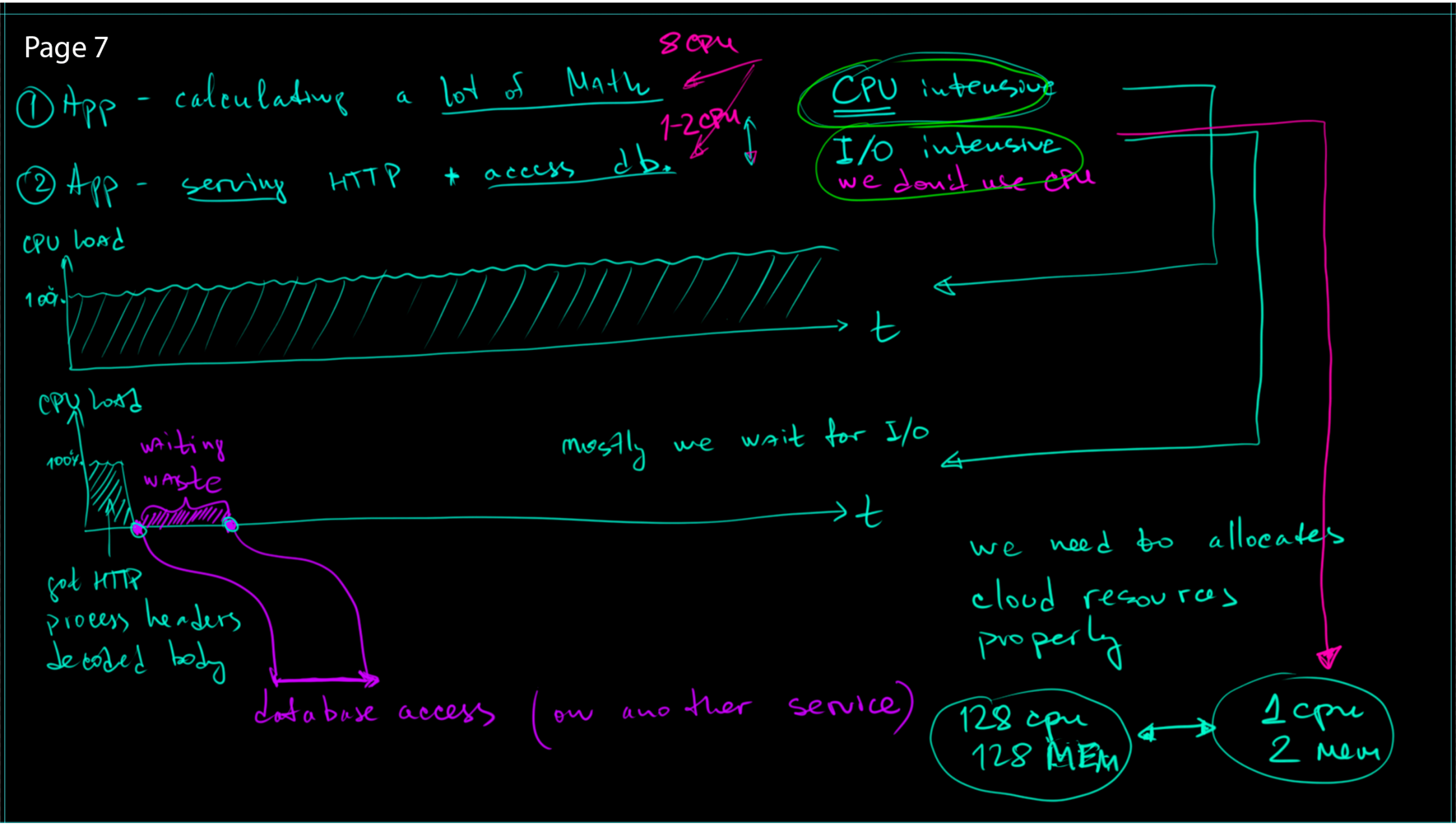
1. define data / domain
2. define operations w/ data
3. implementation
4. API store data (db)
   represent (http)
5. Infrastructure

Hexagonal



mkPayment (A1, A2, Amount,)

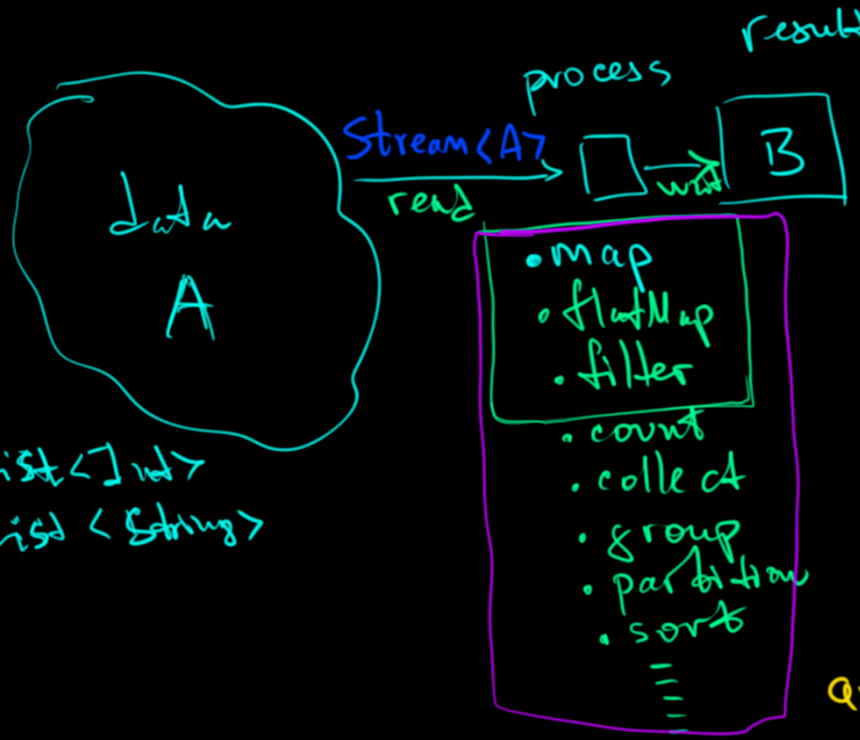
payment can be done in diff ways


8 CPU
1-2 CPU
① App - calculating a lot of Math
② App - serving HTTP + access db.
CPU intensive
I/O intensive
we don't use CPU
CPU load
100%
t
CPU load
100%
waiting
waste
got HTTP
process headers
decoded body
mostly we wait for I/O
t
database access (on another service)
we need to allocate
cloud resources
properly
128 cpu
128 MEM
1 cpu
2 mem

Big Data & ML

Big Data → data doesn't fit in the memory
→ data doesn't fit on the one machine (hard/ssd drive)

problem: we can't load to the memory ALL data

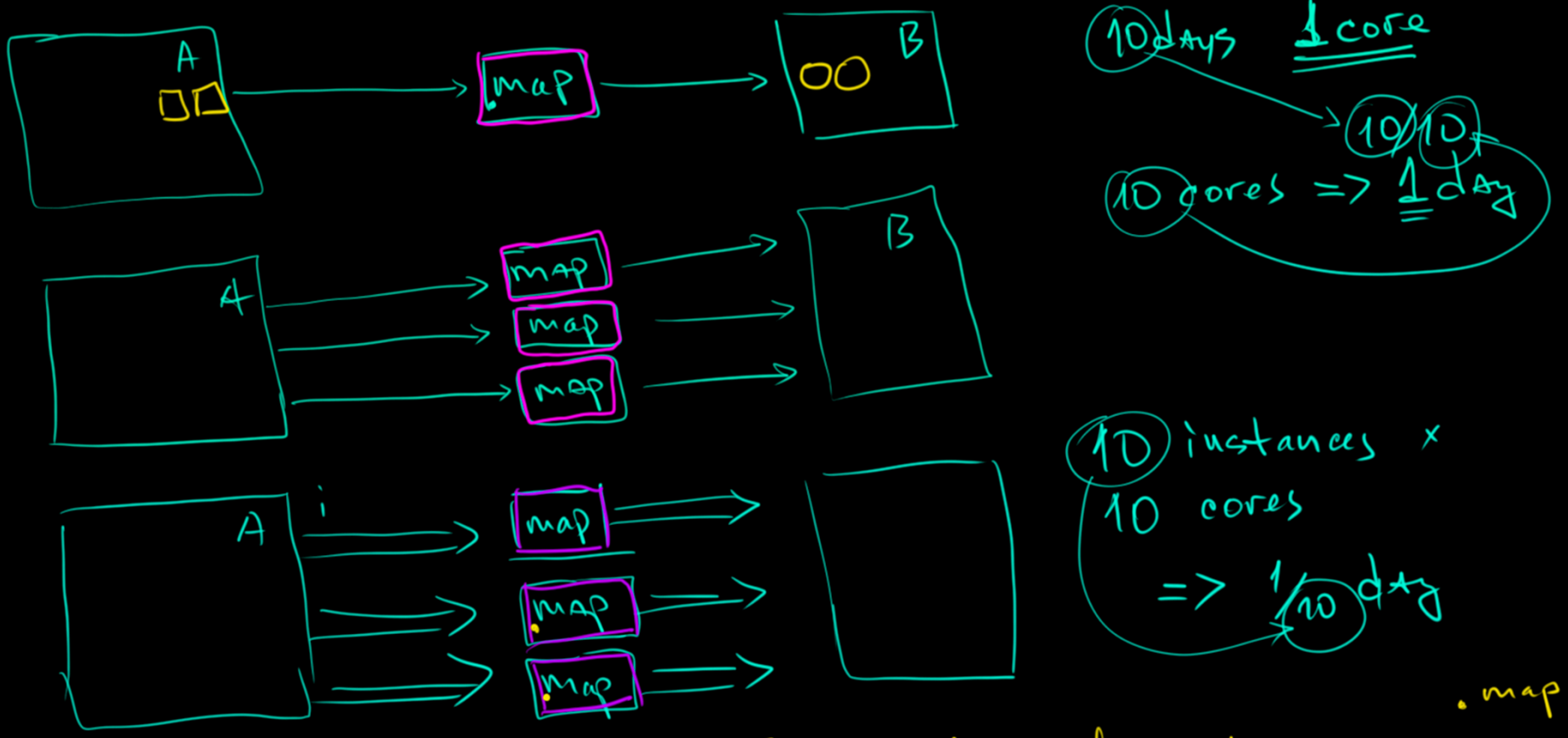


by using frameworks
we don't know
how to read the data
how to write the data


```
var a = read(source)
var b = map(a)
write(target, b)
```

provided by framework

app = read >> map >> write



10 days 1 core

10 10

10 cores => 1 day

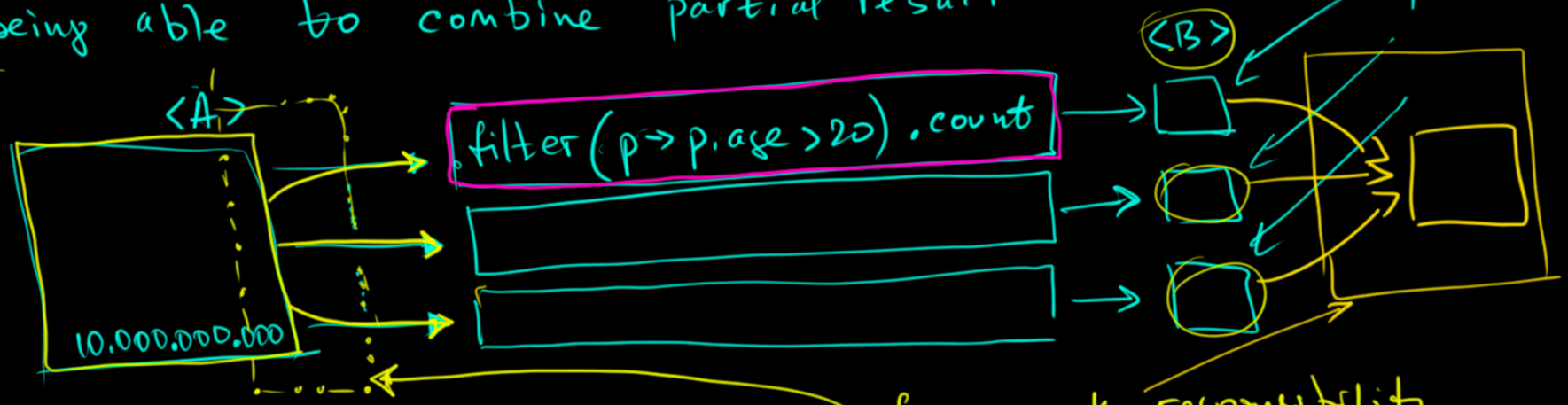
10 instances x
10 cores
=> 1/10 day

.map

Ultimate idea: to represent your transformations and abstract on reading and writing

count people whose age is >20

- being able to split source to process in parallel
- being able to combine partial result



framework responsibility

① map : List<People> => Long

② combine partial results:

- map A => B
- reduce (B,B) => B

.map : A => B

List<B> => B

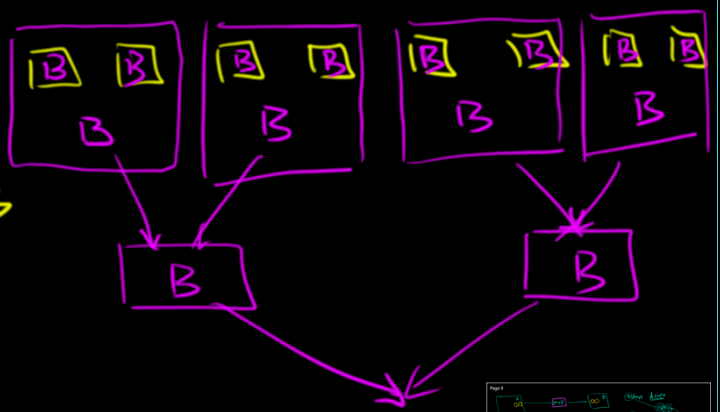
combine (B, B) => B

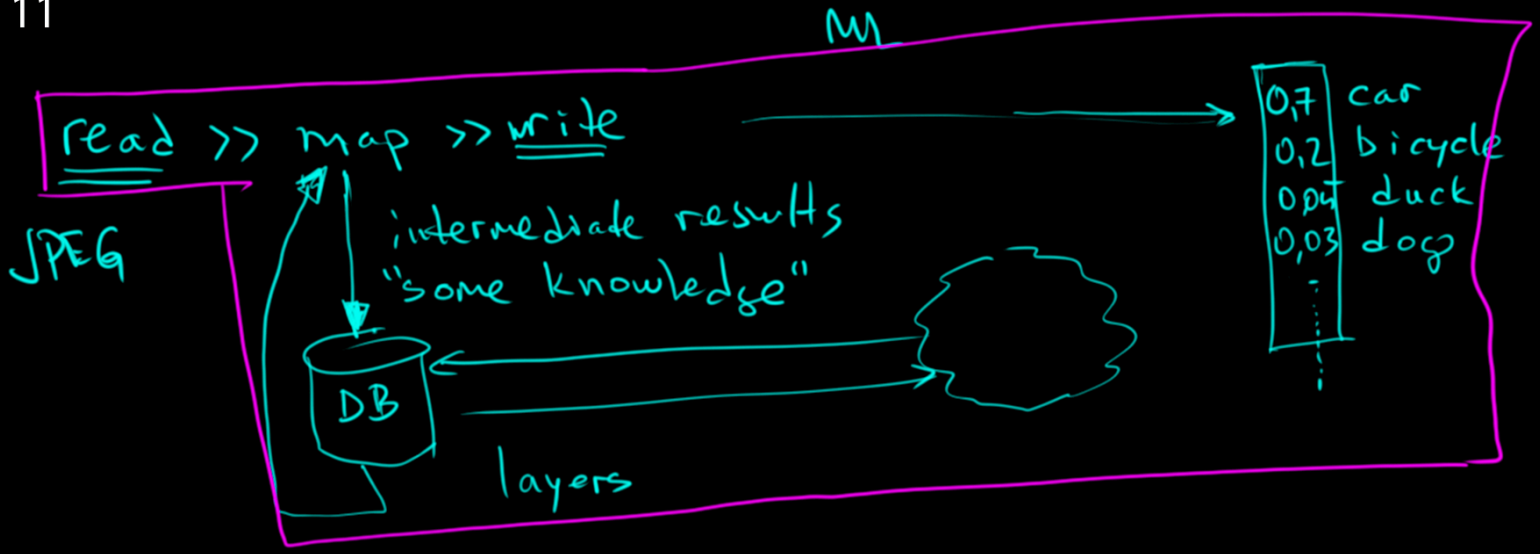
- SQL
- text
- REST JSON
- WS
- ⋮

A ~ (B,B)

f: A => B

(B,B) => B




read >> map >> write
JPEG
intermediate results
"some knowledge"
DB
layers
0,7 car
0,2 bicycle
0,04 duck
0,03 dog

mapReduce