ARDEIM

(AR)CHITECTURE, (DE)SIGN & (IM)PLEMENTATION

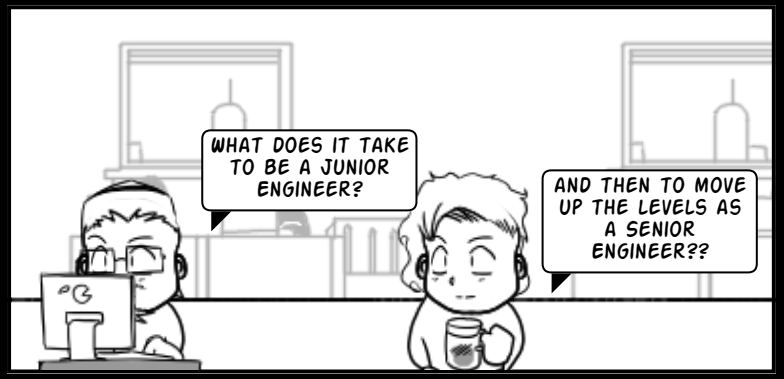
THANKS TO:
RASHMI

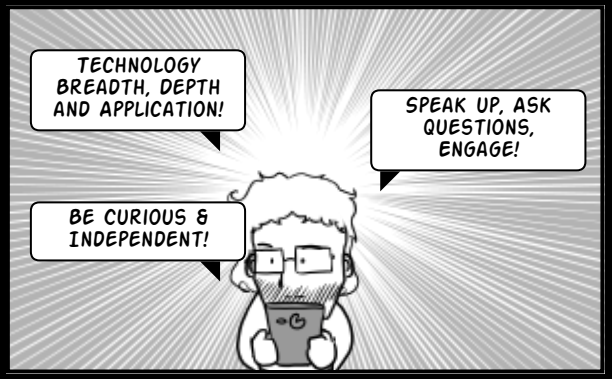
CREDITS:
REVEAL.JS, STRIPTHIS (KESIEV)

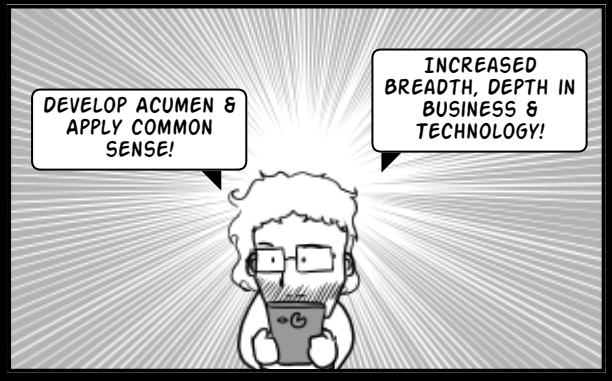
CONTEXT

THE TEAM!

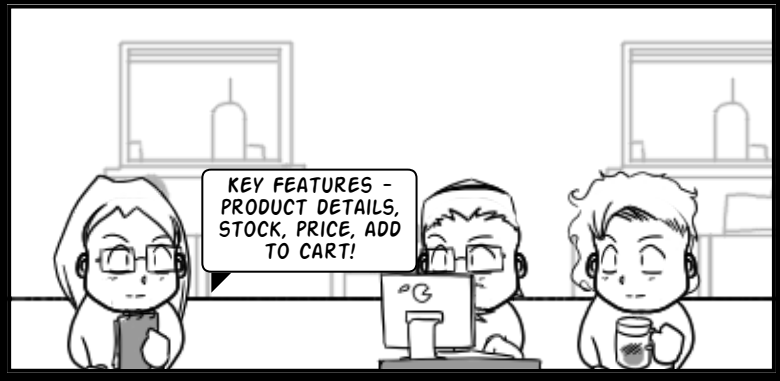
THE ENGINEER'S DILEMMA



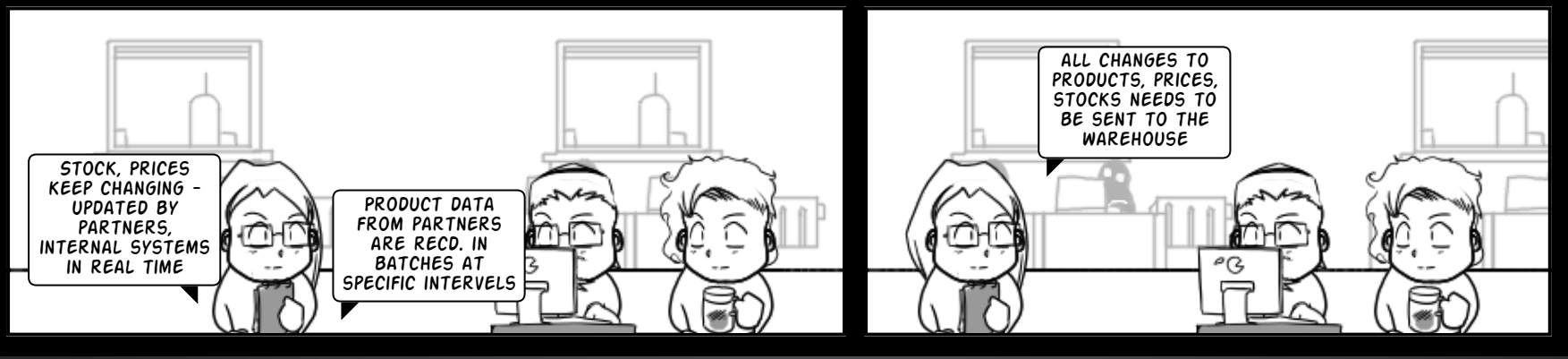




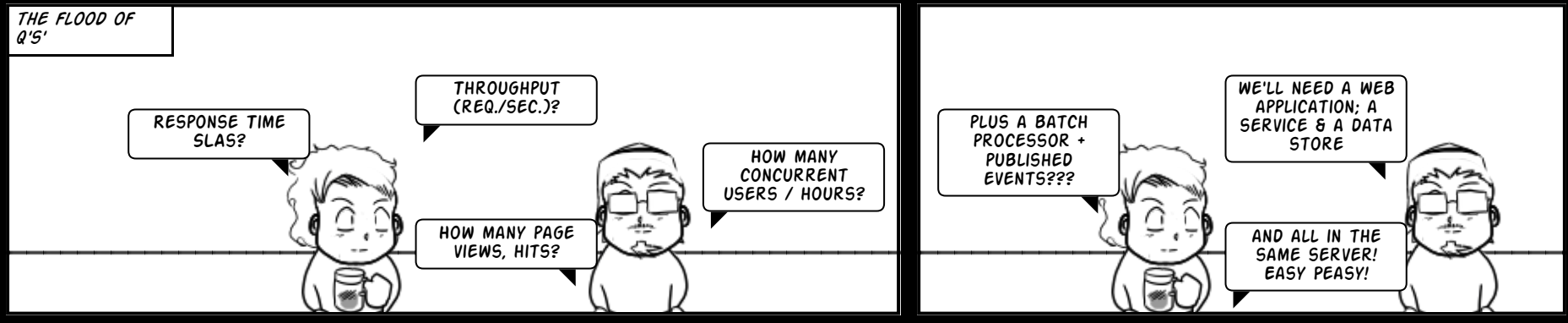
BUILD A PDP WEB UI POWERED BY PUBLIC FACING PRODUCT, PRICE SERVICES



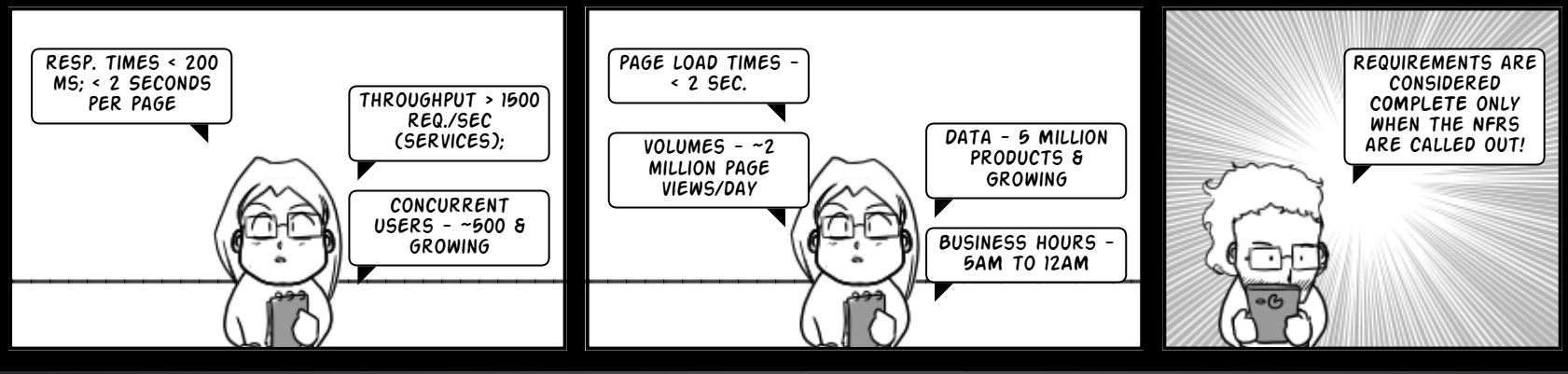
... CONTD : BUILD A PDP WEB UI POWERED BY PUBLIC FACING PRODUCT, PRICE SERVICES



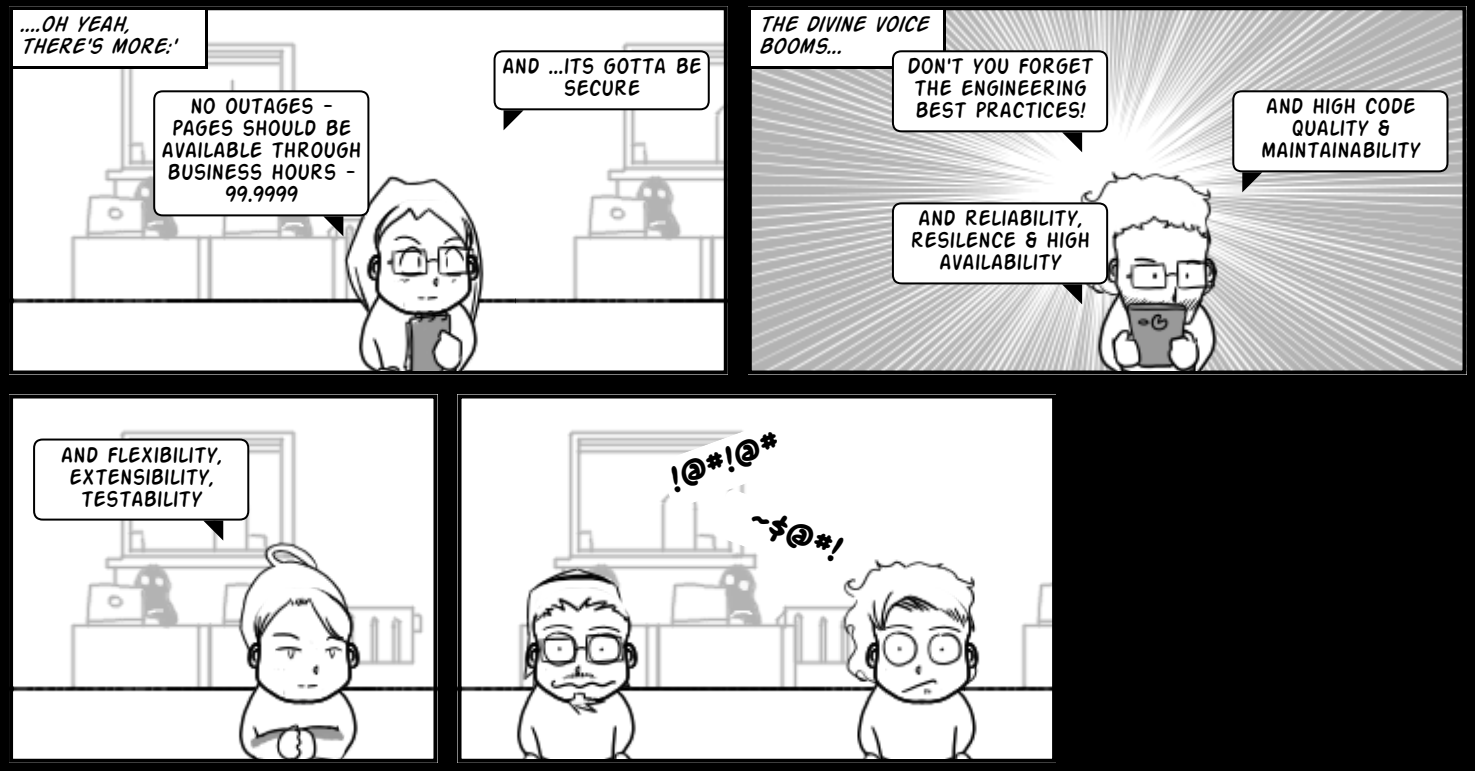
... CONTD : BUILD A PDP WEB UI POWERED BY PUBLIC FACING PRODUCT, PRICE SERVICES



... CONTD : BUILD A PDP WEB UI POWERED BY PUBLIC FACING PRODUCT, PRICE SERVICES

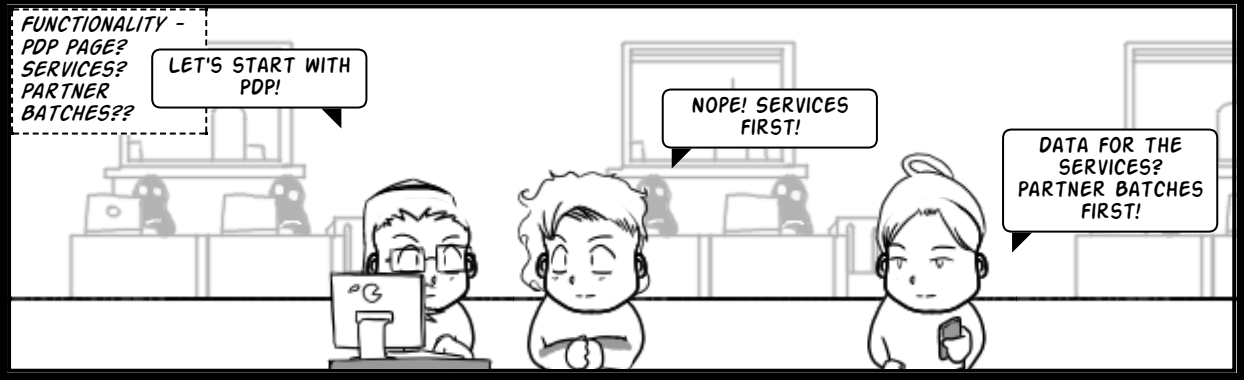


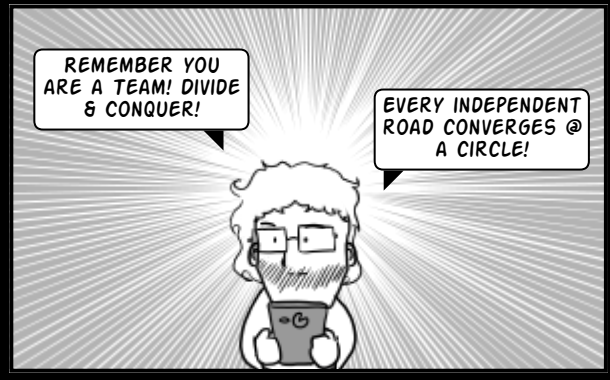
WAIT A MIN ... THERE'S MORE



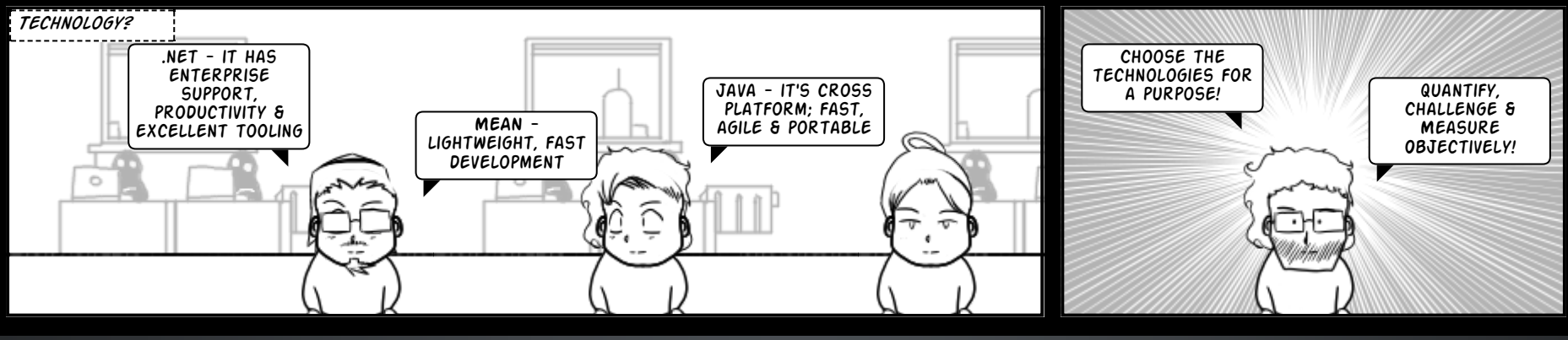
WHAT NEXT!?

TOO MUCH TO CHEW! WHERE DO WE START?!

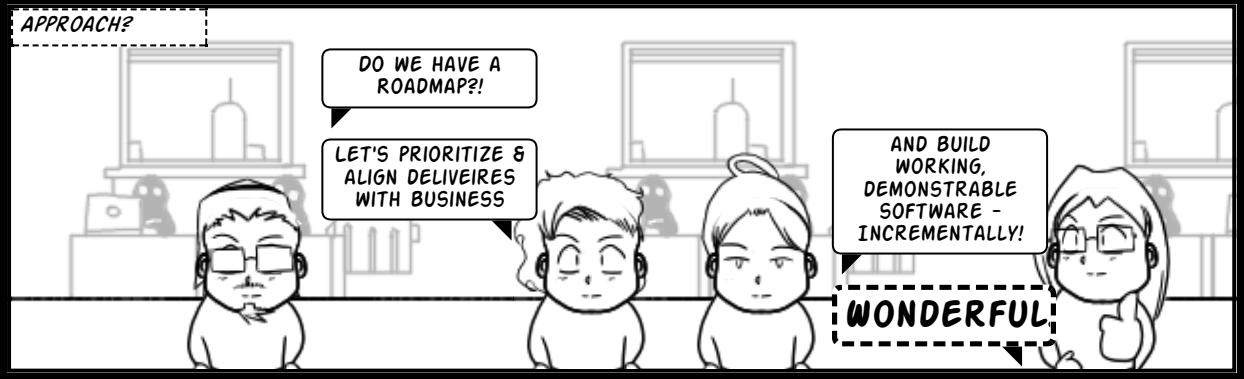


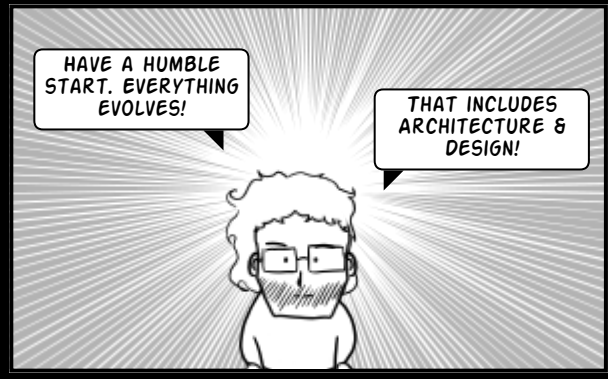


...TOO MUCH TO CHEW! WHERE DO WE START?!



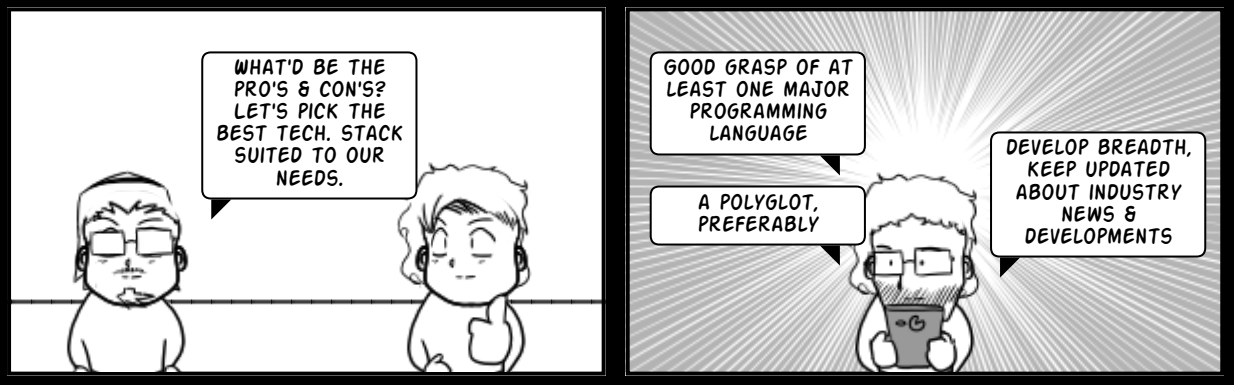
...TOO MUCH TO CHEW! WHERE DO WE START?!





Links: Domain Driven Design - Eric Evans

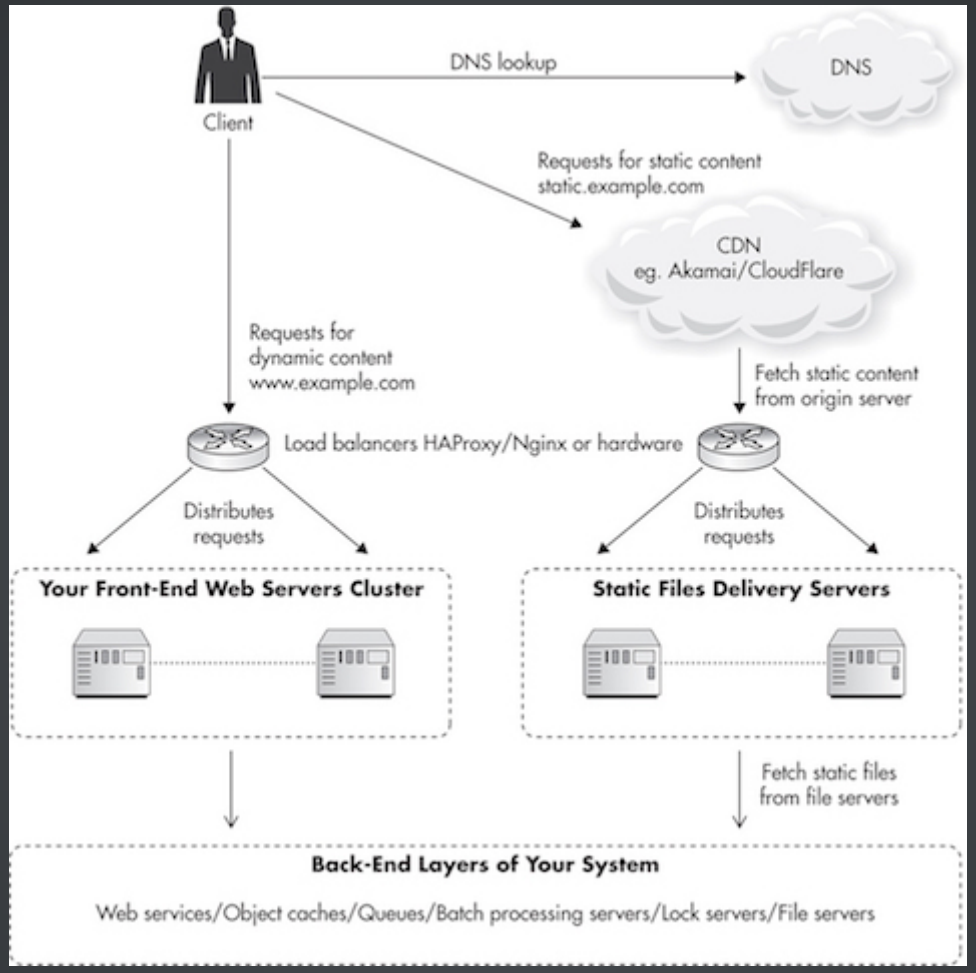
...TOO MUCH TO CHEW! WHERE DO WE START?!



...CONTD!


JAVA VS .NET VS .NET CORE VS MEAN???
SPRING OR WEBAPIS??
VERT.X OR NODE??
PLAY OR NANCY?
RXJAVA OR AKKA??
DDD & DOMAIN MODELLING FOR SERVICES

Which Technology to choose for App Layer?



NGINX beased on event loop scales well also provides load balncing and reverse proxy capabilities

Choose the Framework with right Threading Model Multiteactor Pattern Node.js or Vert.x?

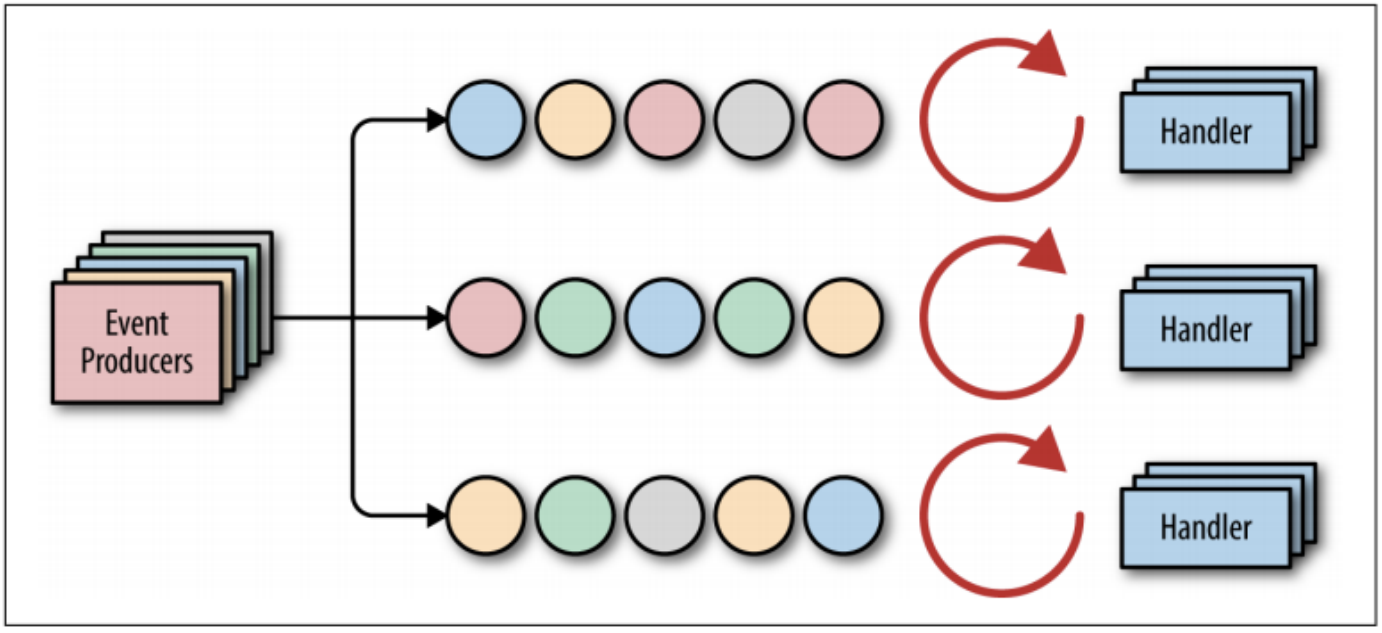
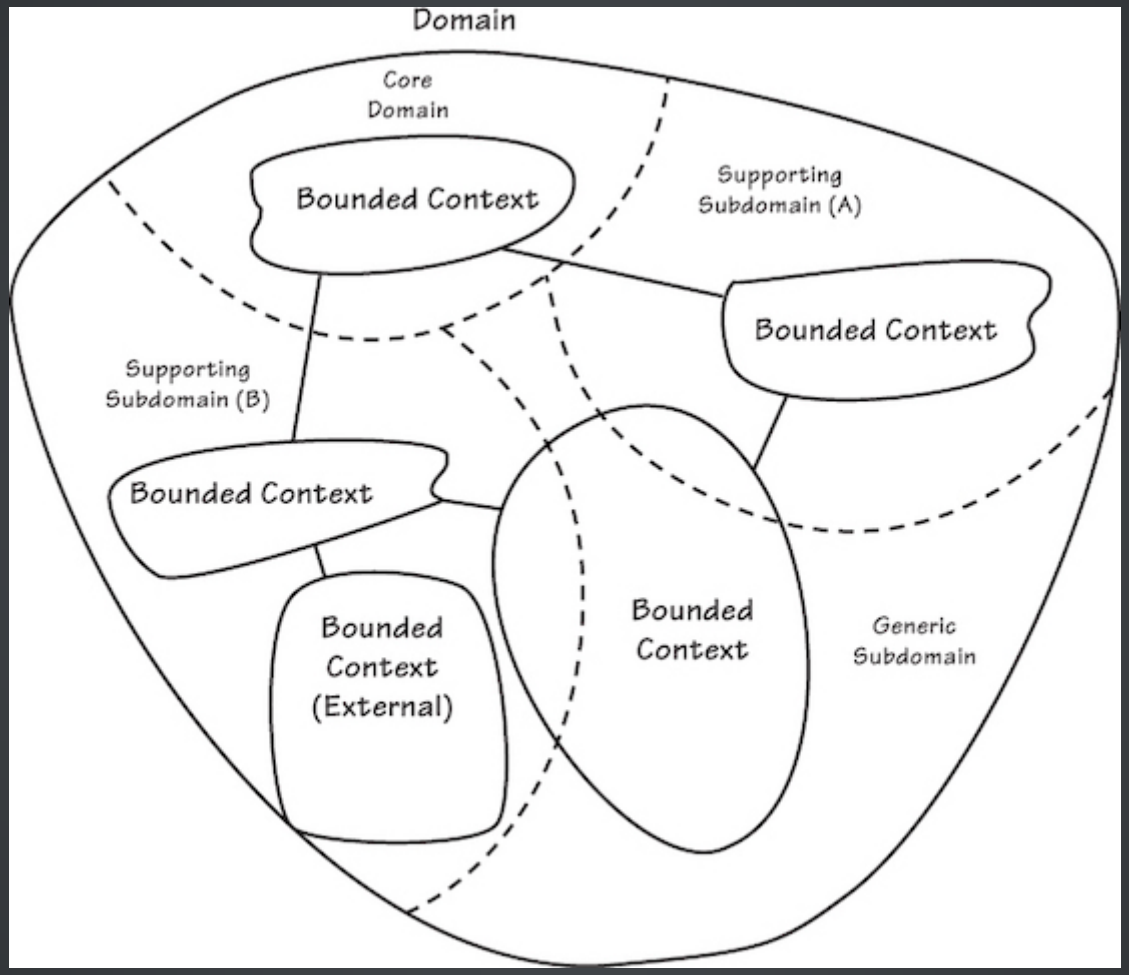
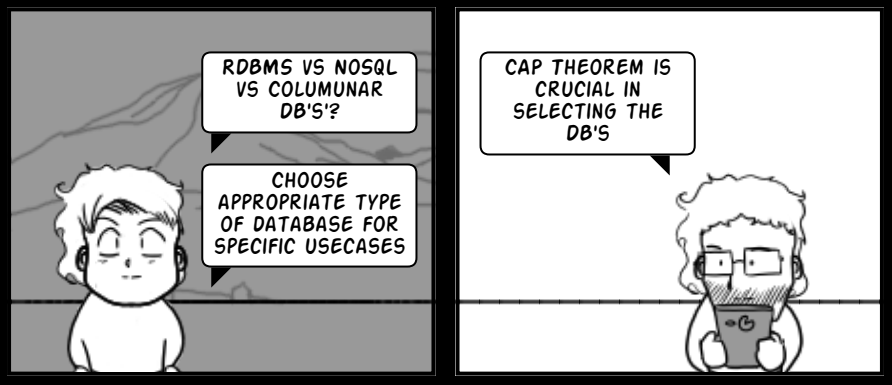


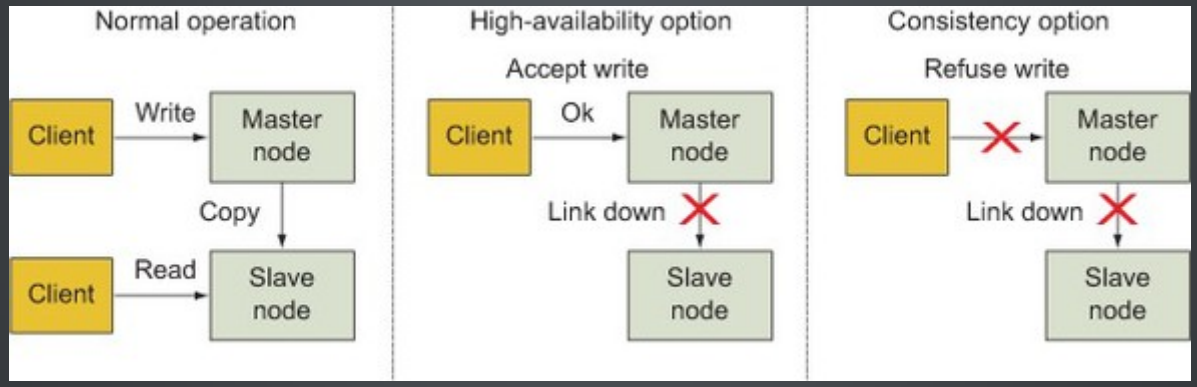
Figure 2-4. The multireactor principle


Domain
Core
Domain
Bounded Context
Supporting
Subdomain (A)
Bounded Context
Supporting
Subdomain (B)
Bounded Context
Bounded
Context
(External)
Bounded
Context
Generic
Subdomain

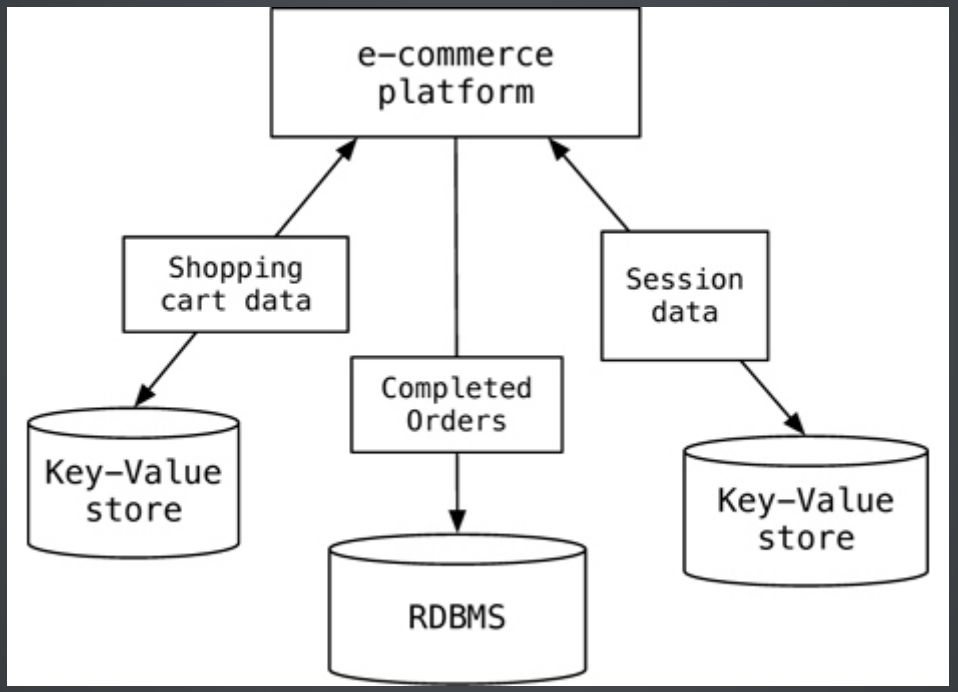
CHOOSE THE TECHNOLOGIES, FRAMEWORKS & TOOLS FOR DB LAYER



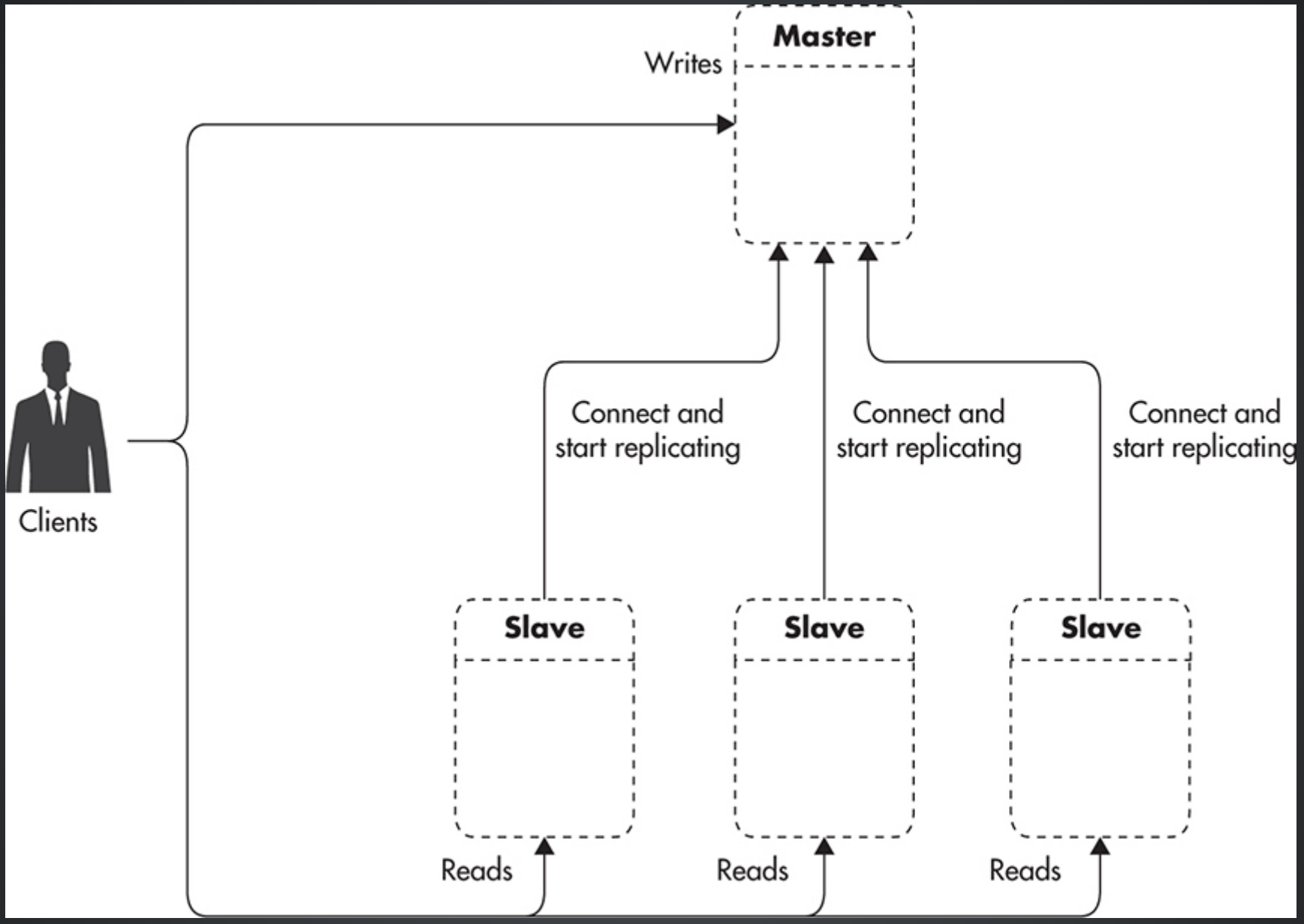
CAP Therom
Availability , Constitancy, Partition Tolerance



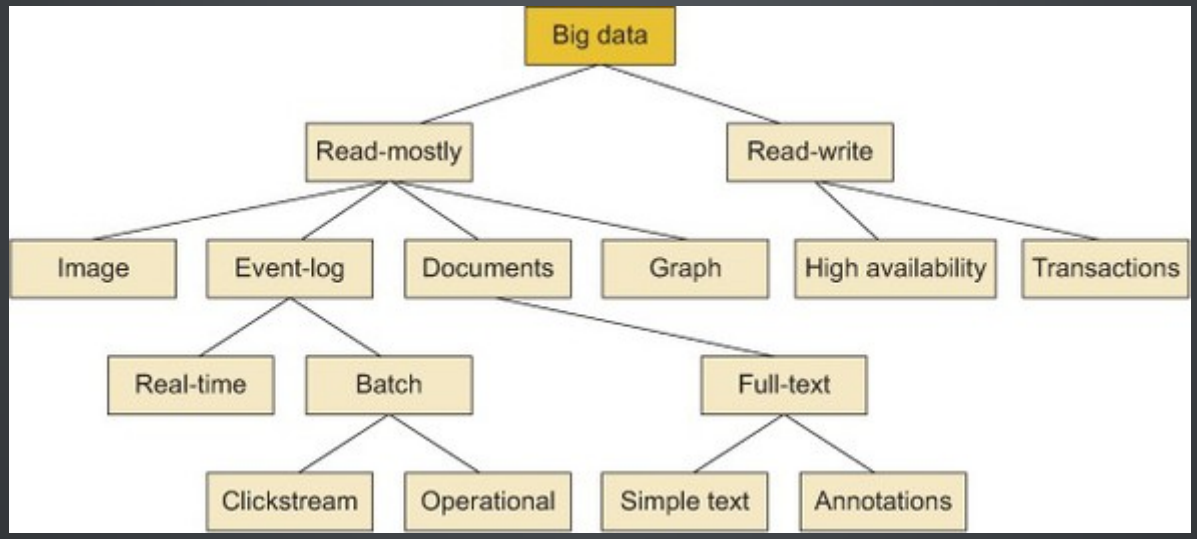
Ploygot DataBase for different purpose



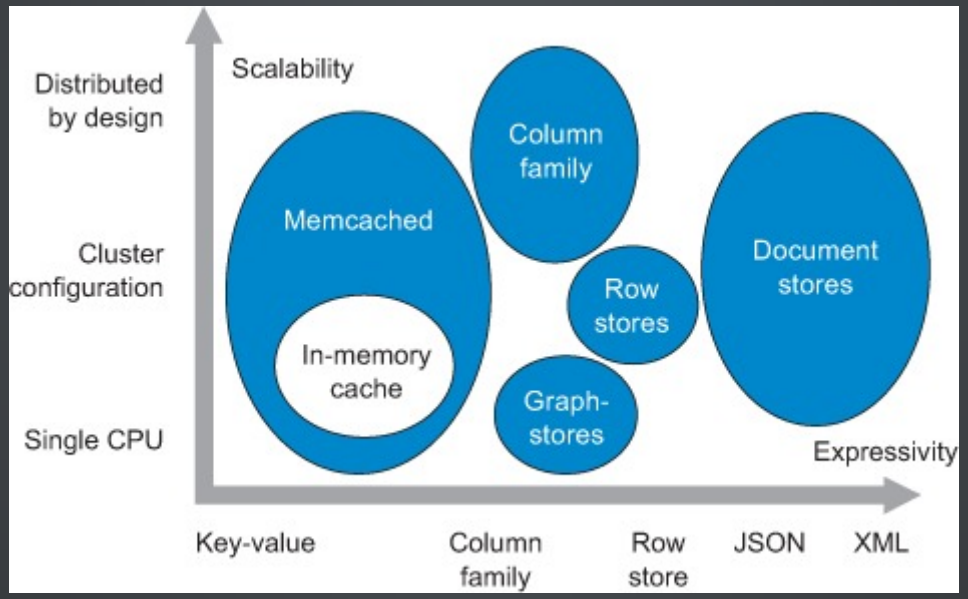
Ploygot DataBase for different purpose


Master
Writes
Clients
Connect and start replicating
Connect and start replicating
Connect and start replicating
Slave
Slave
Slave
Reads
Reads
Reads

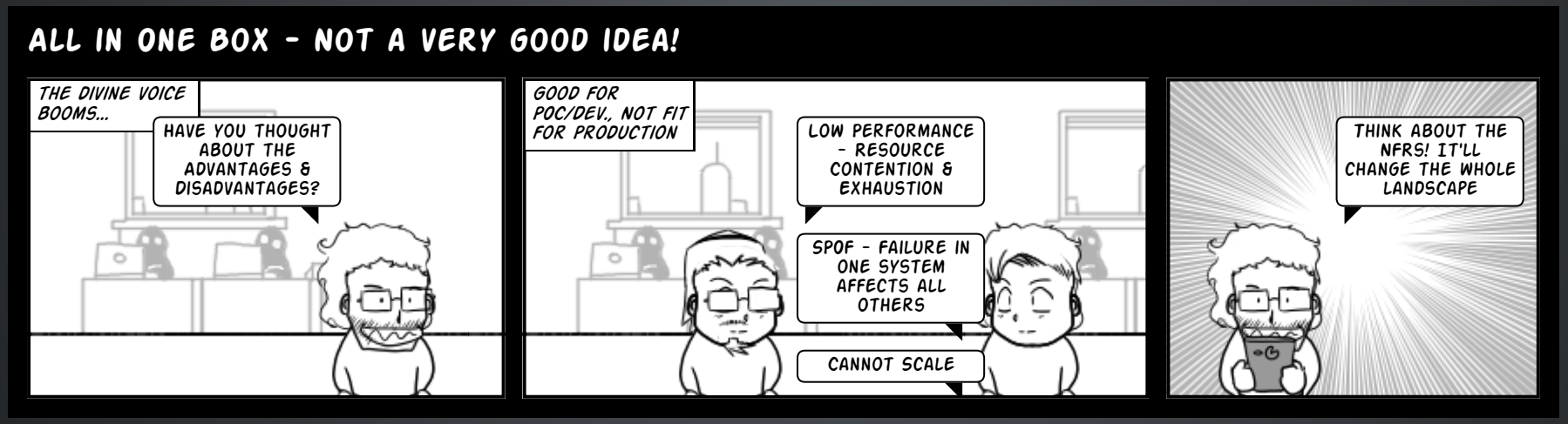
Which Technology to choose in DB Layer? Choose the right data base based on the use case for data generation



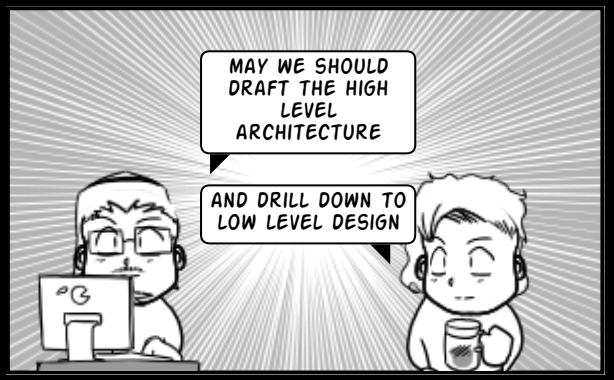
Which Technology to choose for nature of data
Choose the right data base based on the type of data

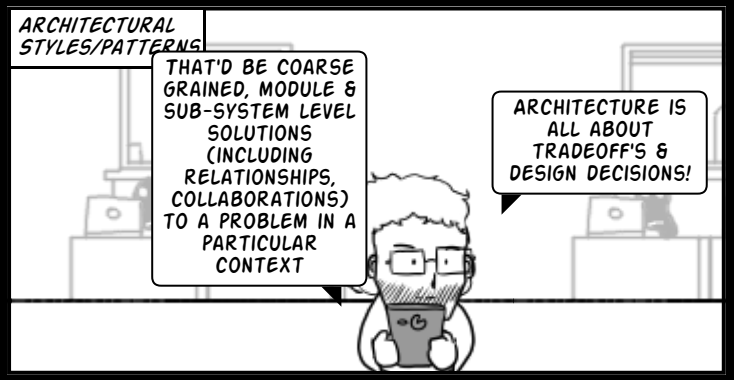


ARCHITECTURE & DESIGN ...

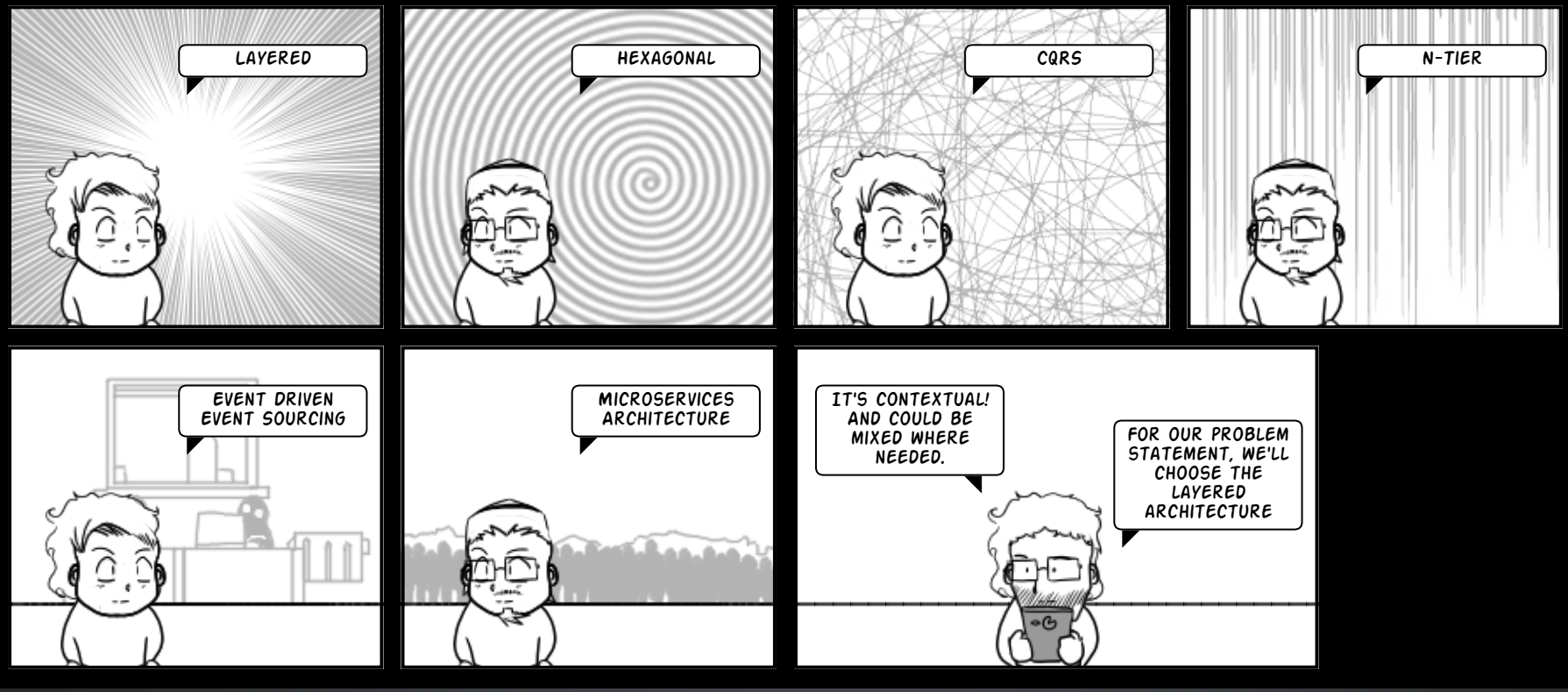


DEVS THINKING...



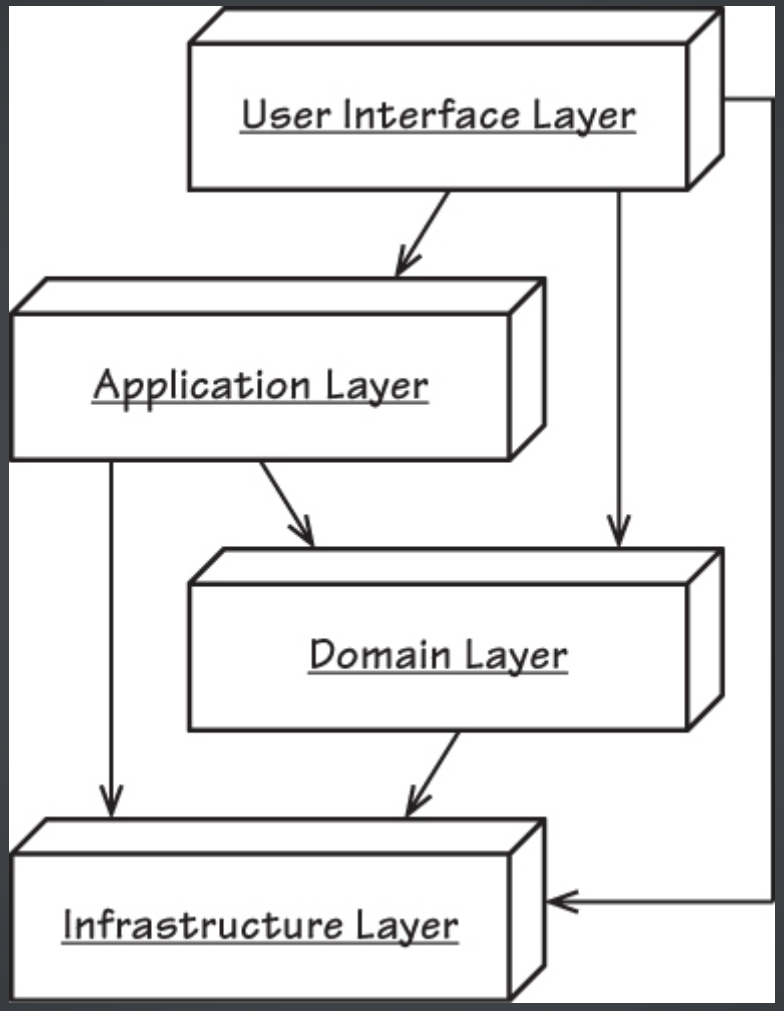


ARCHITECTURAL STYLES/PATTERNS & SELECTION



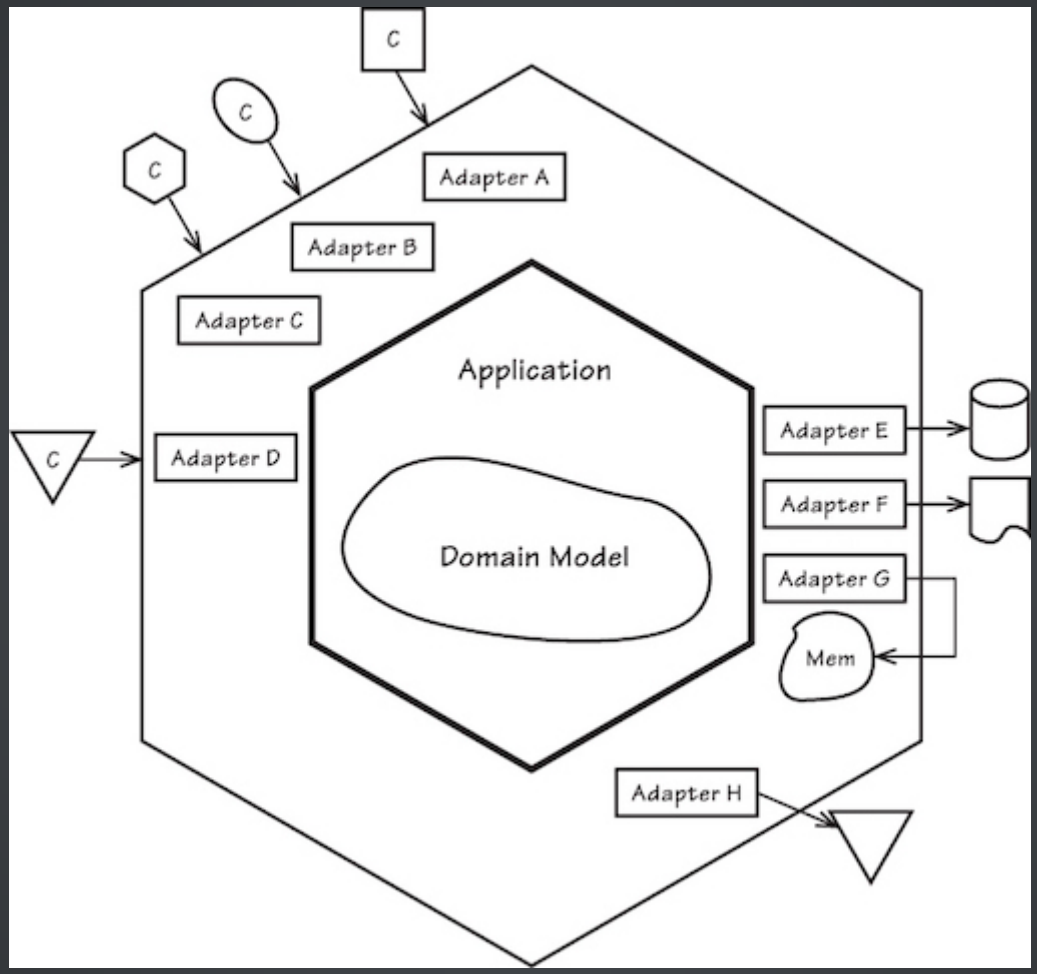
```
* [Layered Architectures](http://www.softwarearchitectures.com/qa.html)
* [Hexagonal Architecture](https://en.wikipedia.org/wiki/Multitier_arch
* [CQRS](https://en.wikipedia.org/wiki/Multitier_architecture)
* [N-Tier](https://en.wikipedia.org/wiki/Multitier_architecture)
* [Event Driven / Event Sourcing](https://en.wikipedia.org/wiki/Multiti
```

Layered Architecture



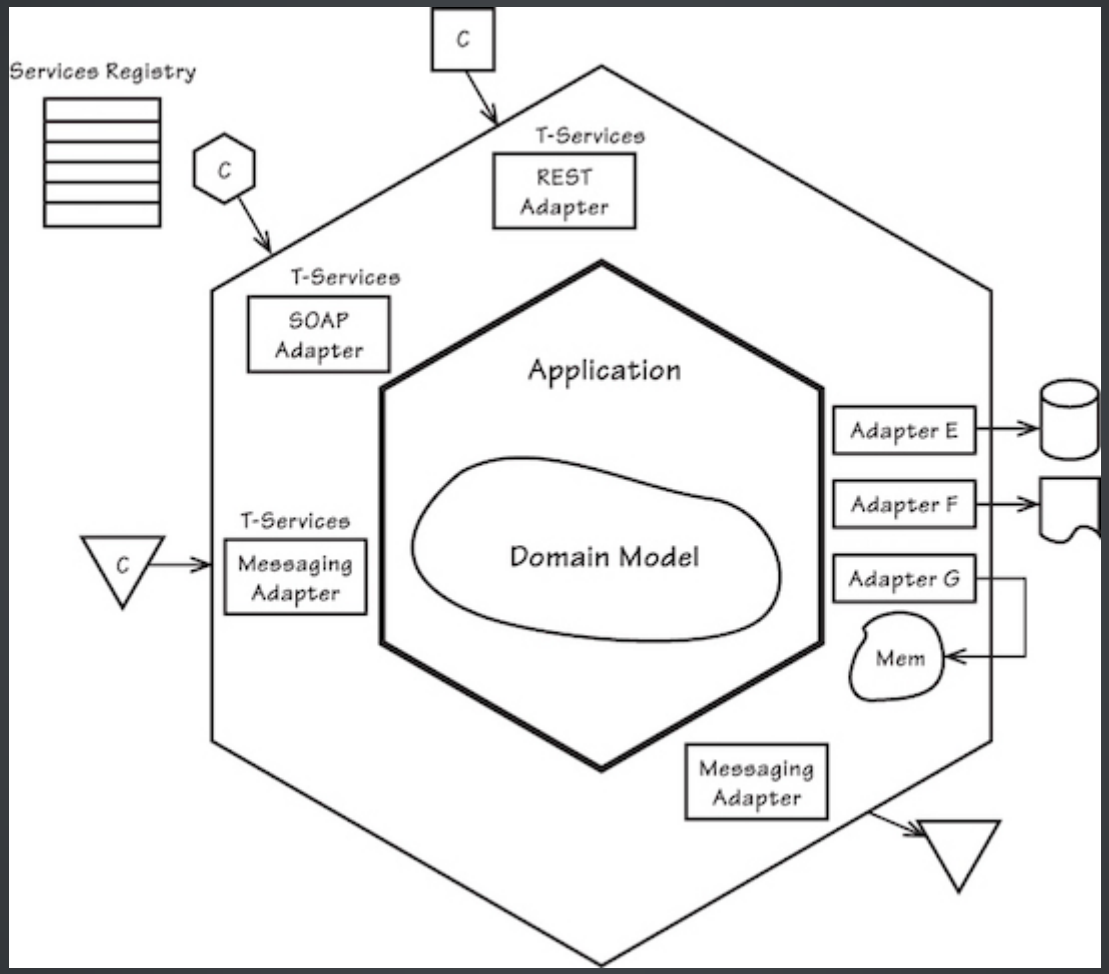
First Basic Architecture

Hexagonal Architecture

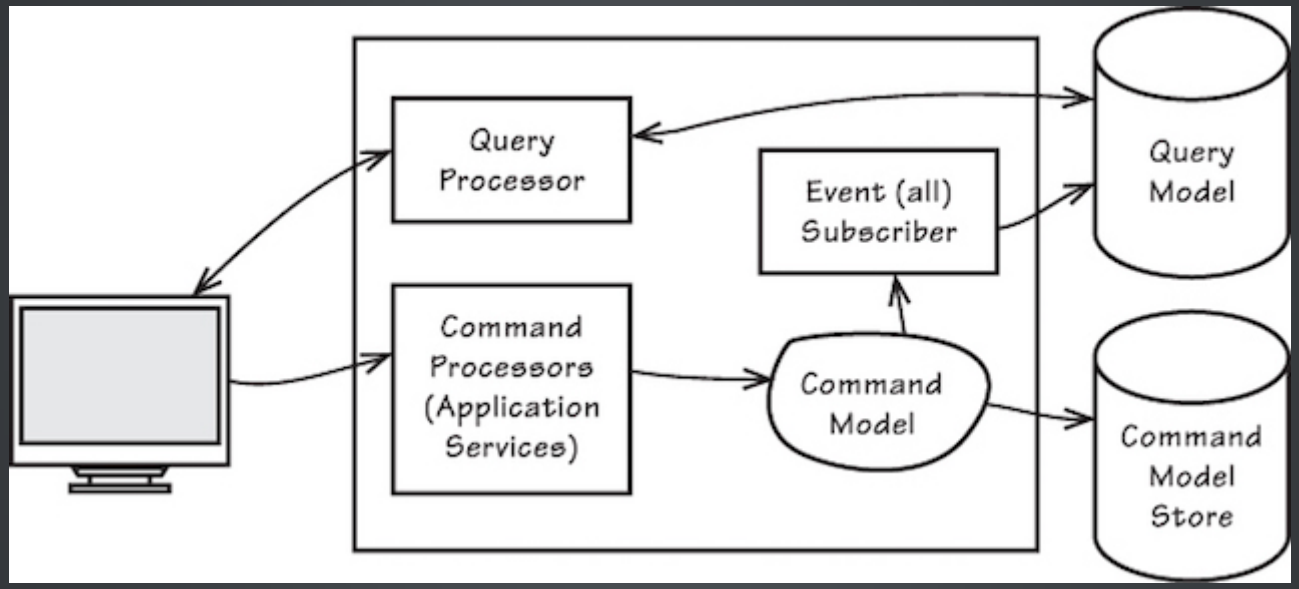


The concepts of Adapters with CoreDomain

Hexogonal Architecture SOA

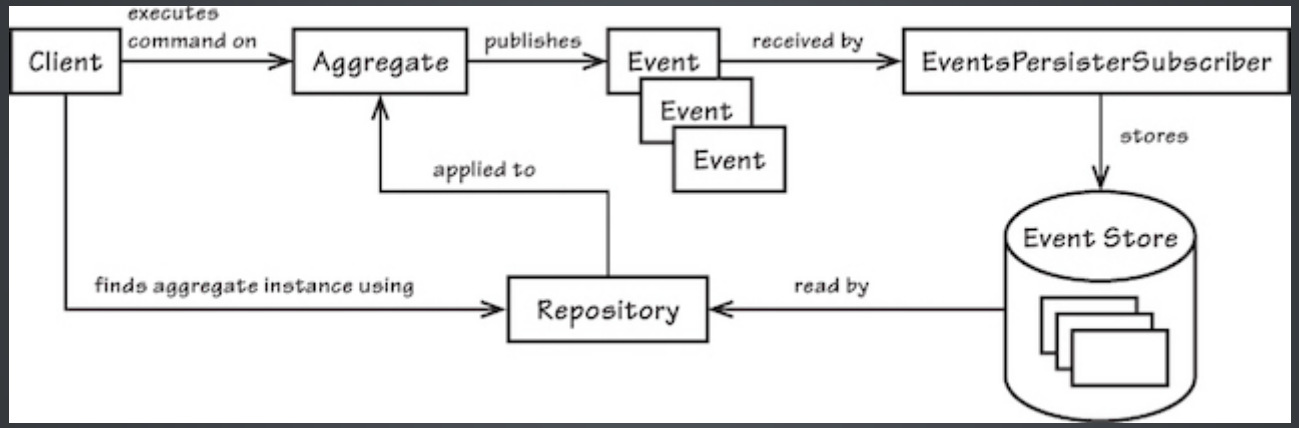


CQRS



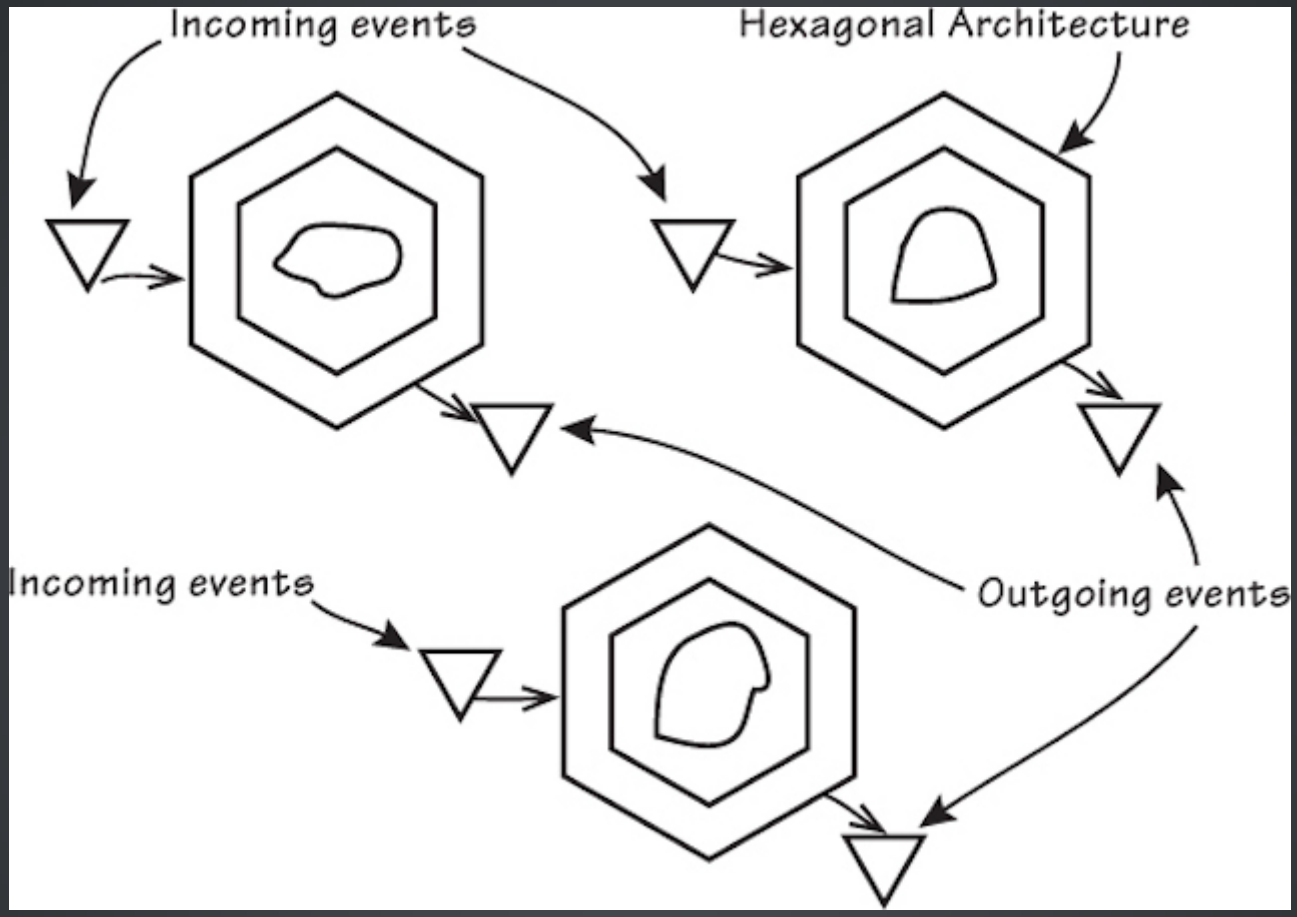
Seperate FLow for Query and Create

Event Sourcing



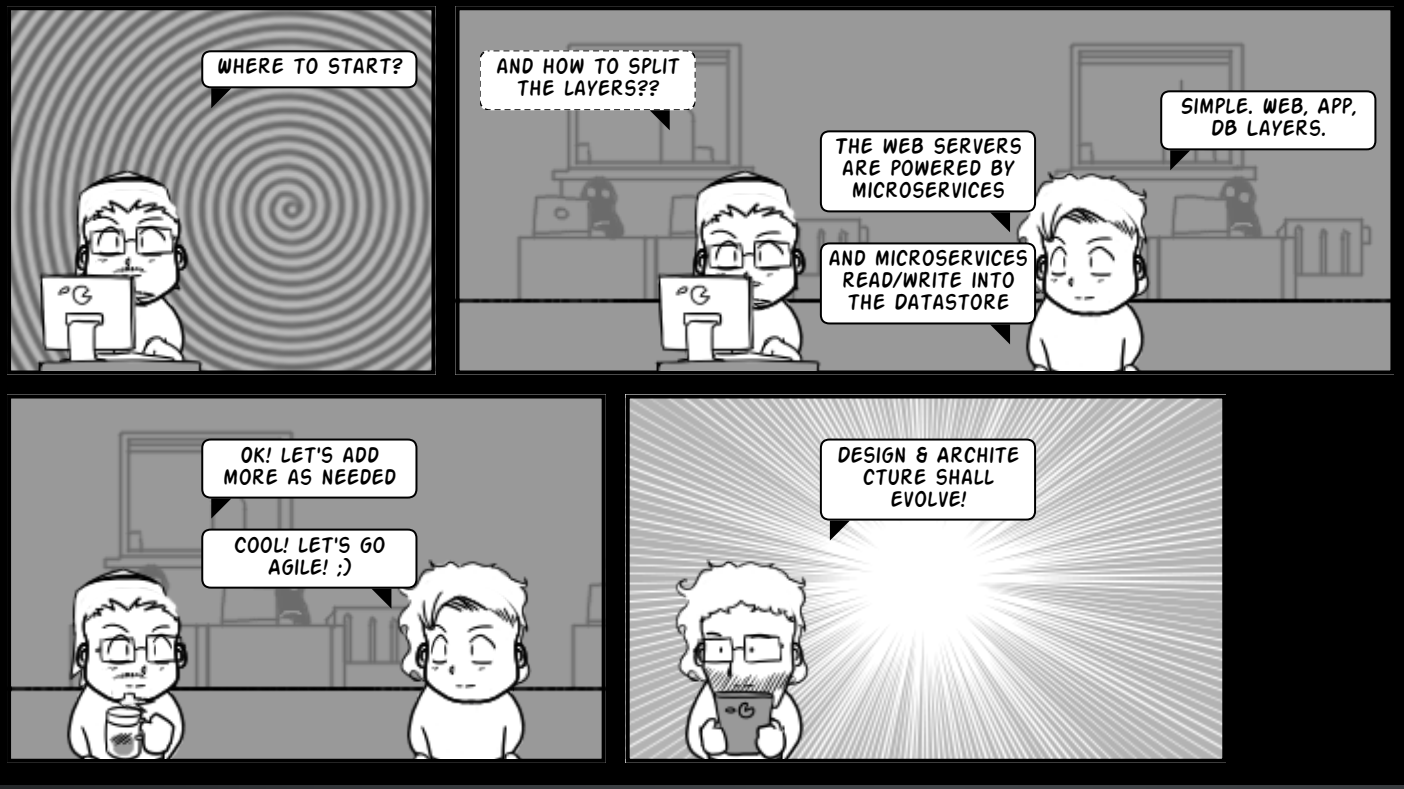
Events are stored no events are updates or deleted

Event Driven

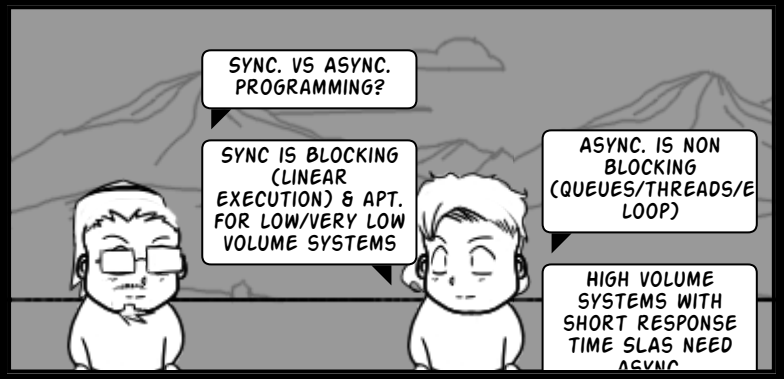


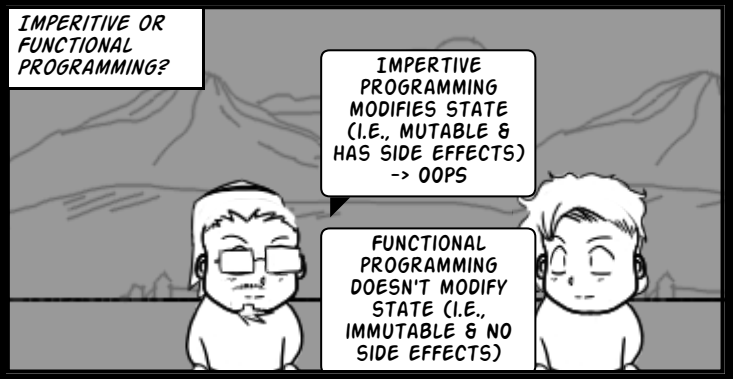
Different System integrtate using Domain Events

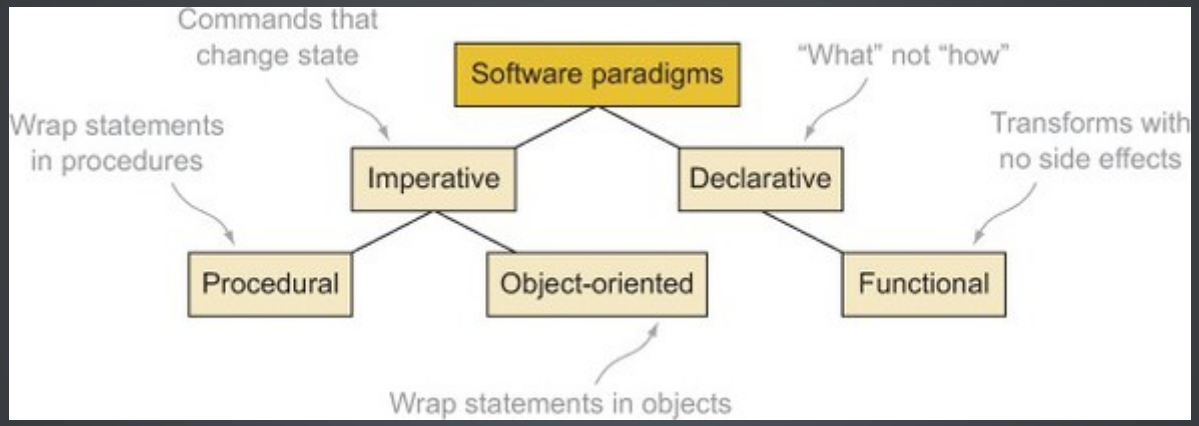
N-TIER ARCHITECTURE & LAYERS

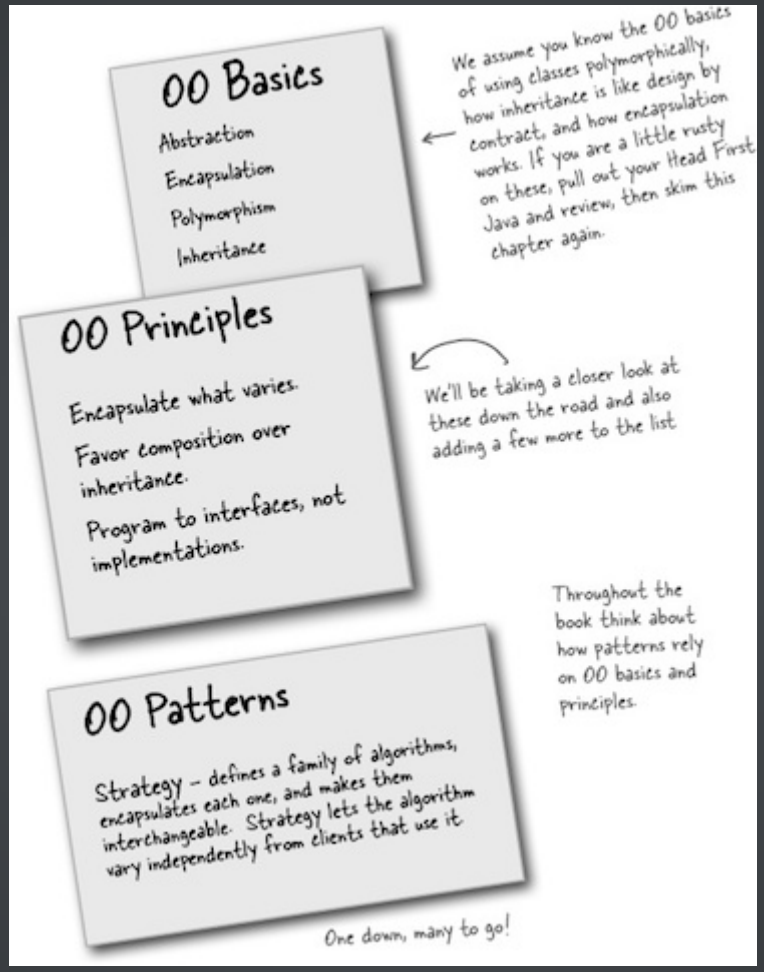


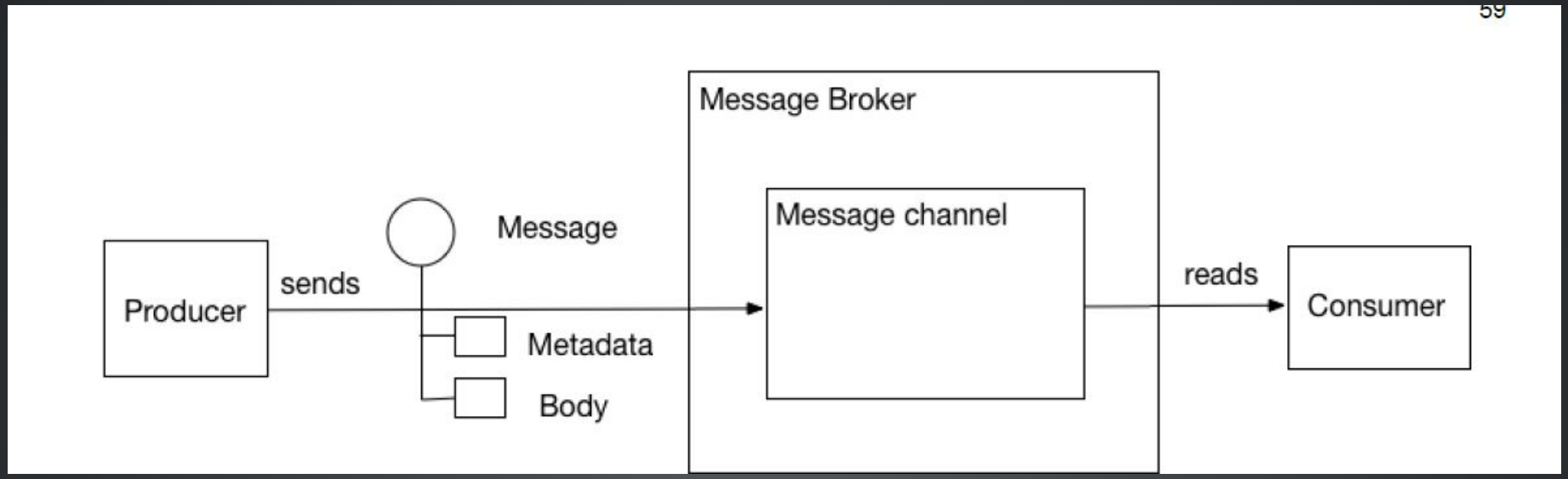
BUILDING THE APP - CHOICE OF THE PROGRAMMING STYLE






Commands that change state
Software paradigms
"What" not "how"
Wrap statements in procedures
Transforms with no side effects
Imperative
Declarative
Procedural
Object-oriented
Functional
Wrap statements in objects


OO Basics
Abstraction
Encapsulation
Polymorphism
Inheritance
We assume you know the OO basics of using classes polymorphically, how inheritance is like design by contract, and how encapsulation works. If you are a little rusty on these, pull out your Head First Java and review, then skim this chapter again.
OO Principles
Encapsulate what varies.
Favor composition over inheritance.
Program to interfaces, not implementations.
We'll be taking a closer look at these down the road and also adding a few more to the list
Throughout the book think about how patterns rely on OO basics and principles.
OO Patterns
Strategy – defines a family of algorithms, encapsulates each one, and makes them interchangeable. Strategy lets the algorithm vary independently from clients that use it.
One down, many to go!

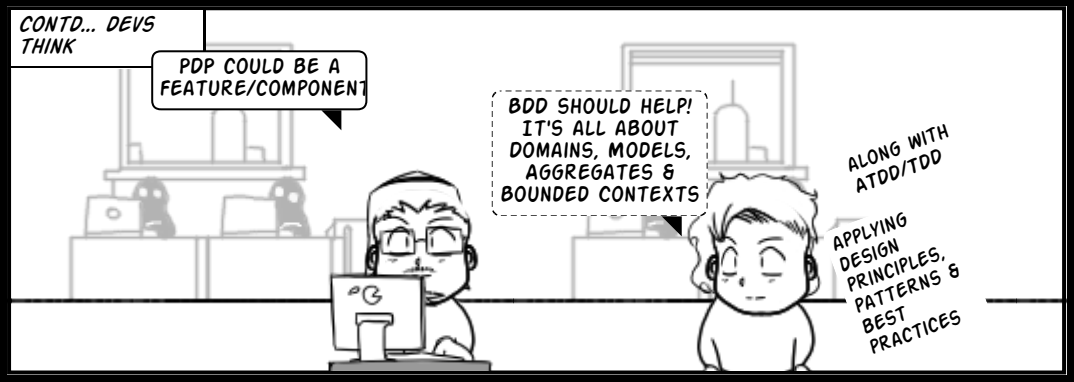


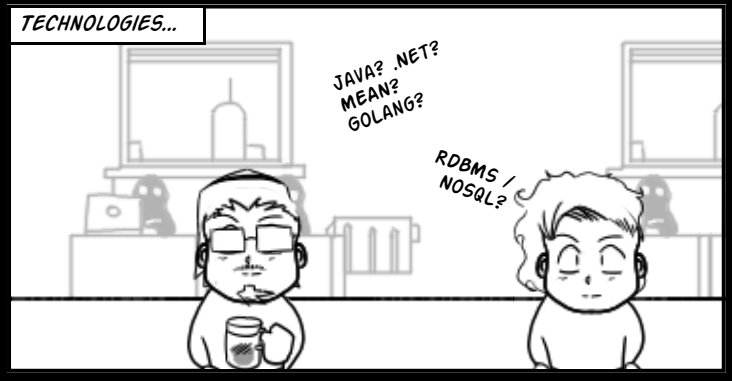
Synchronous does not perform all the time

AMQP based Prtocool Product Rabit MQ

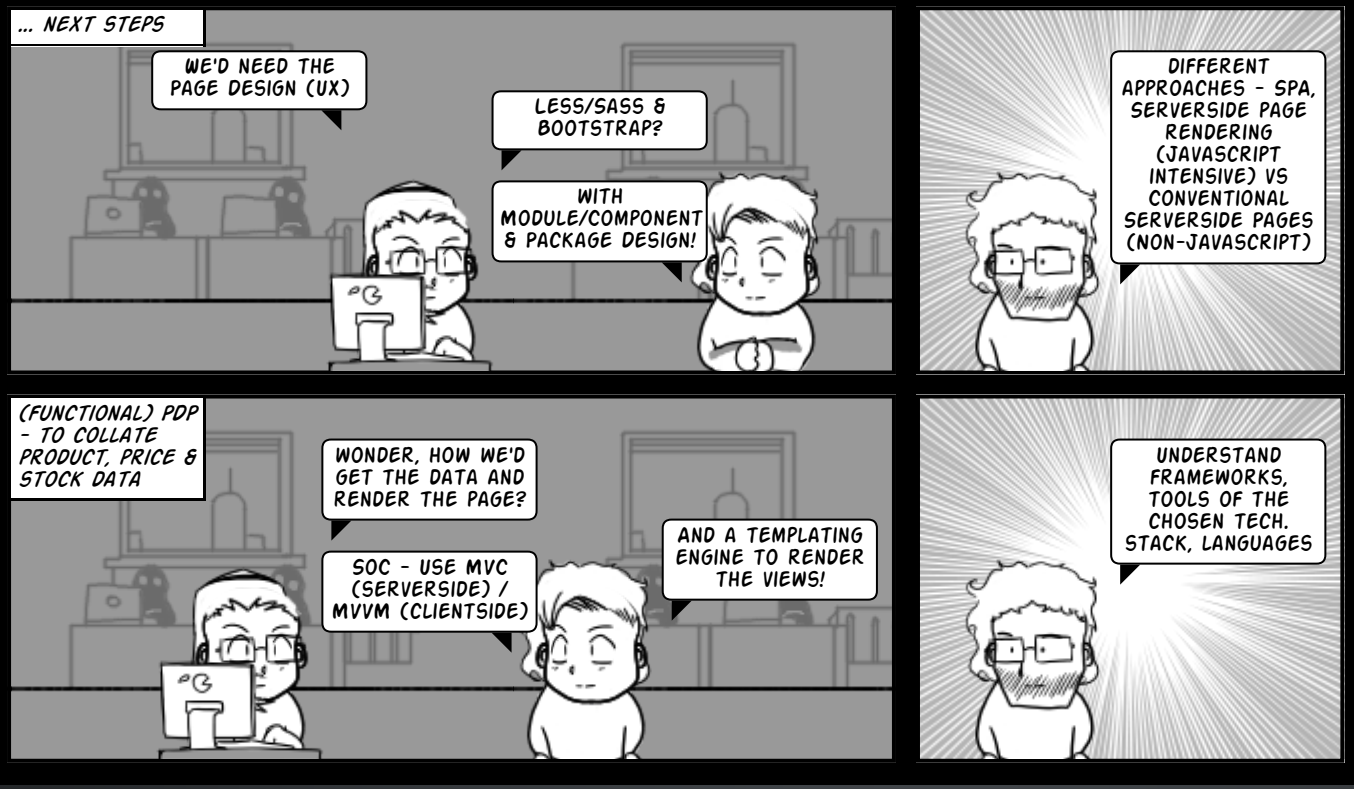
https://maryrosecook.com/blog/post/a-practical-introduction-to-functior
https://blog.heroku.com/concurrency_is_not_parallelism
https://talks.golang.org/2012/waza.slide#1

DESIGN, DESIGN, DESIGN

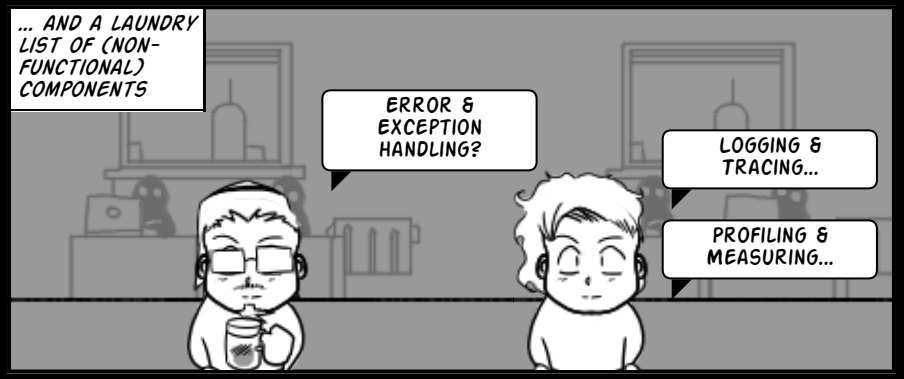


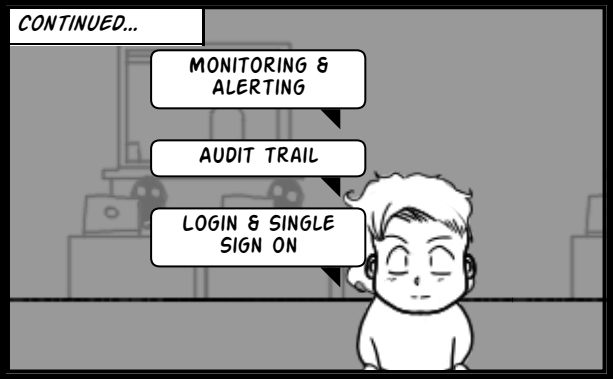


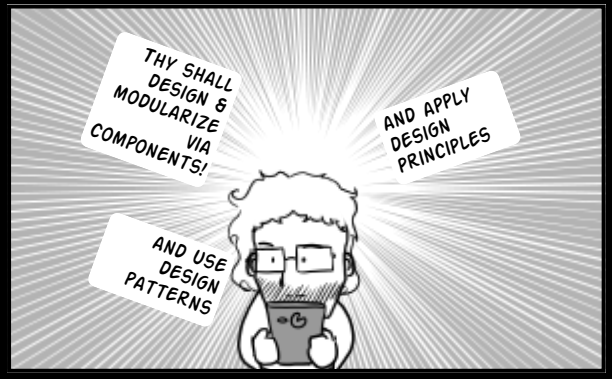
BREAKING DOWN INTO MANAGABLE PIECES - FUNCTIONAL COMPONENTS



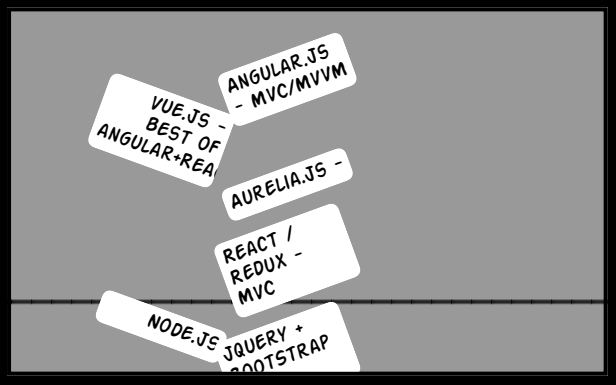
BREAKING DOWN INTO MANAGABLE PIECES - NON-FUNCTIONAL COMPONENTS

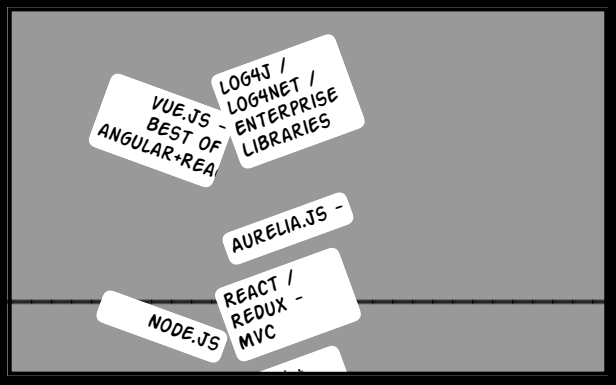






WEB LAYER - LINKS

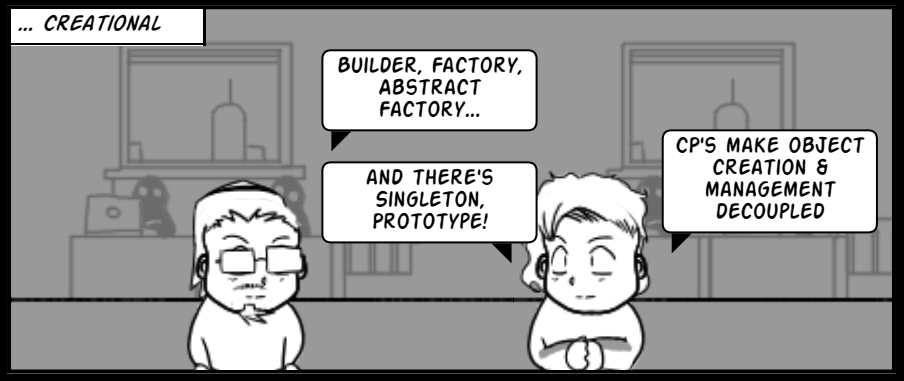


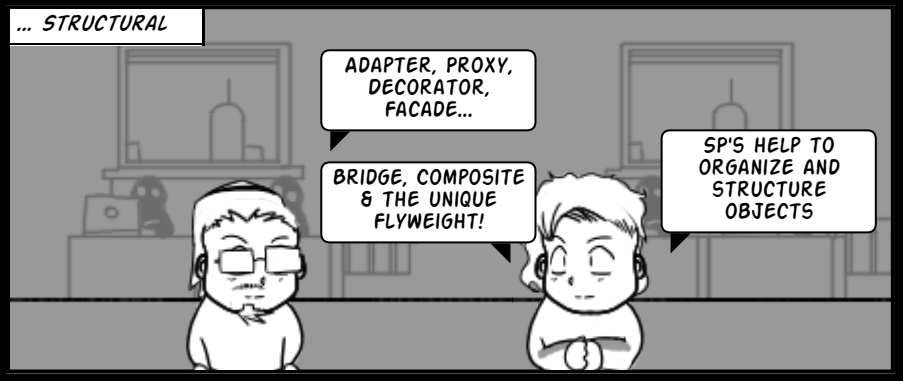


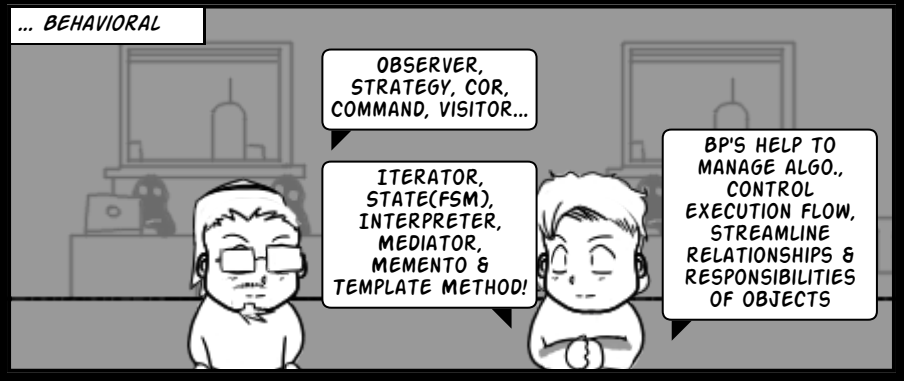
https://msdn.microsoft.com/en-us/library/ms998530.aspx
https://www.microsoft.com/en-us/download/confirmation.aspx?id=11711
https://msdn.microsoft.com/en-us/library/ff921345.aspx
https://msdn.microsoft.com/en-gb/library/ff648138.aspx
http://dataguidance.codeplex.com/releases

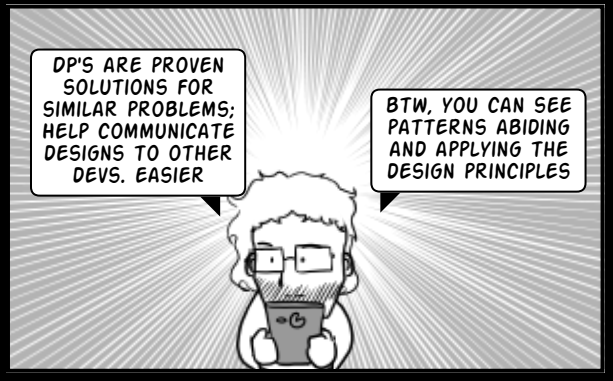
Architecture
http://shapingsoftware.com/
http://sourcesofinsight.com/

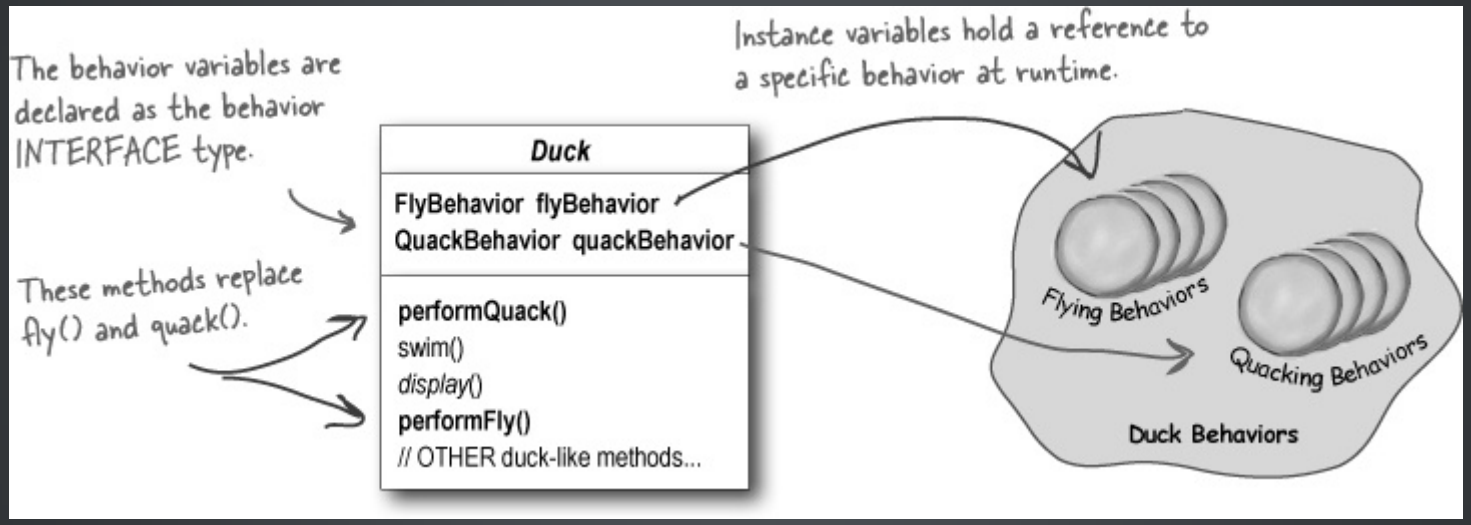
DESIGN PATTERNS - FINE GRAINED, CLASS LEVEL SOLUTION TO A PROBLEM IN A PARTICULAR CONTEXT

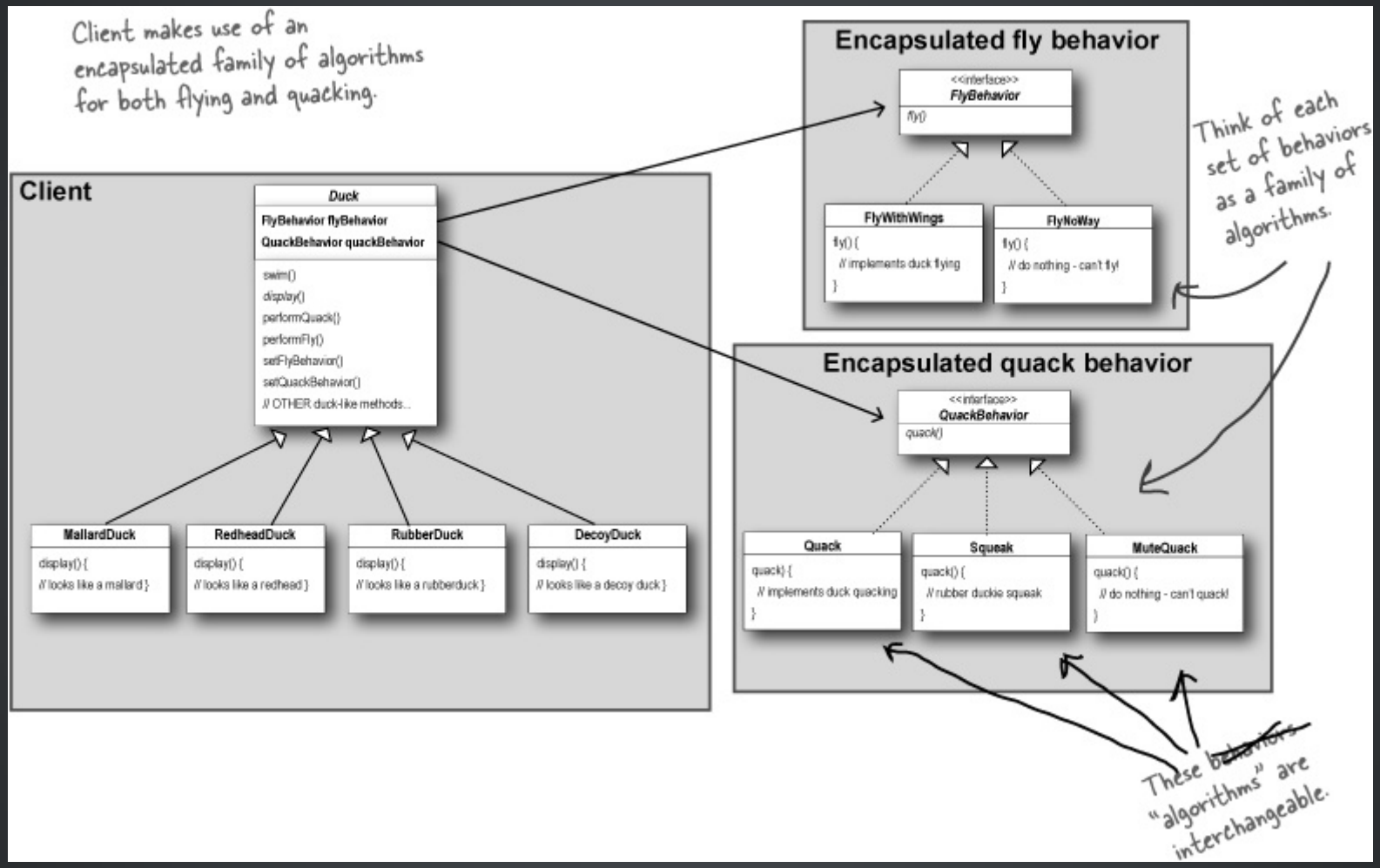


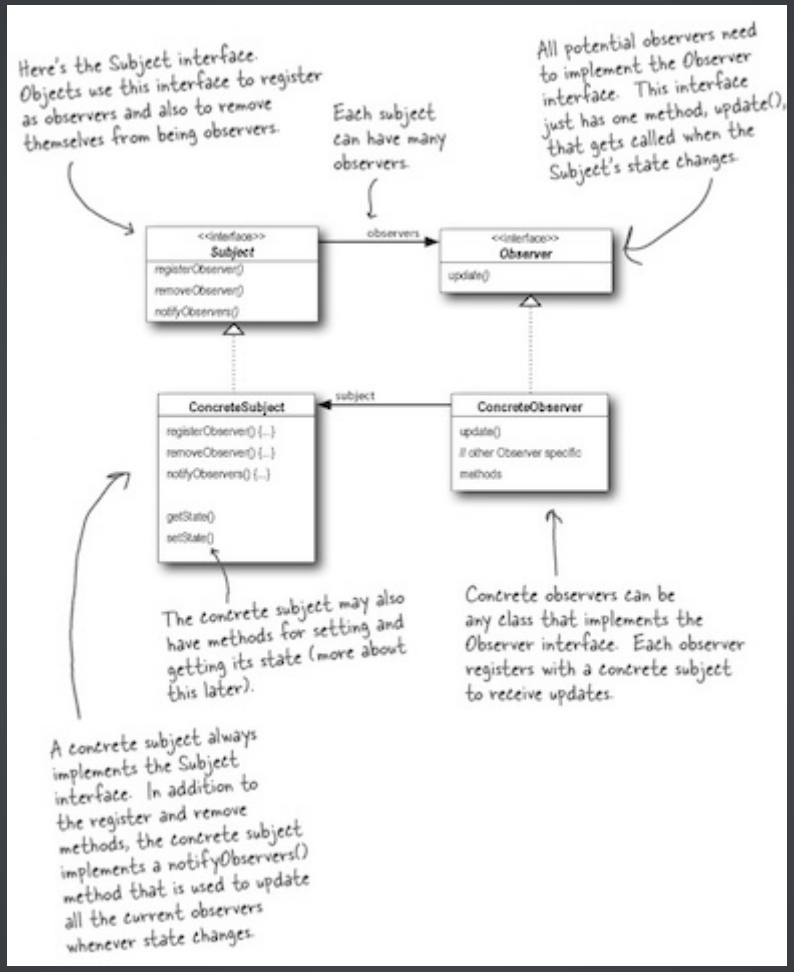


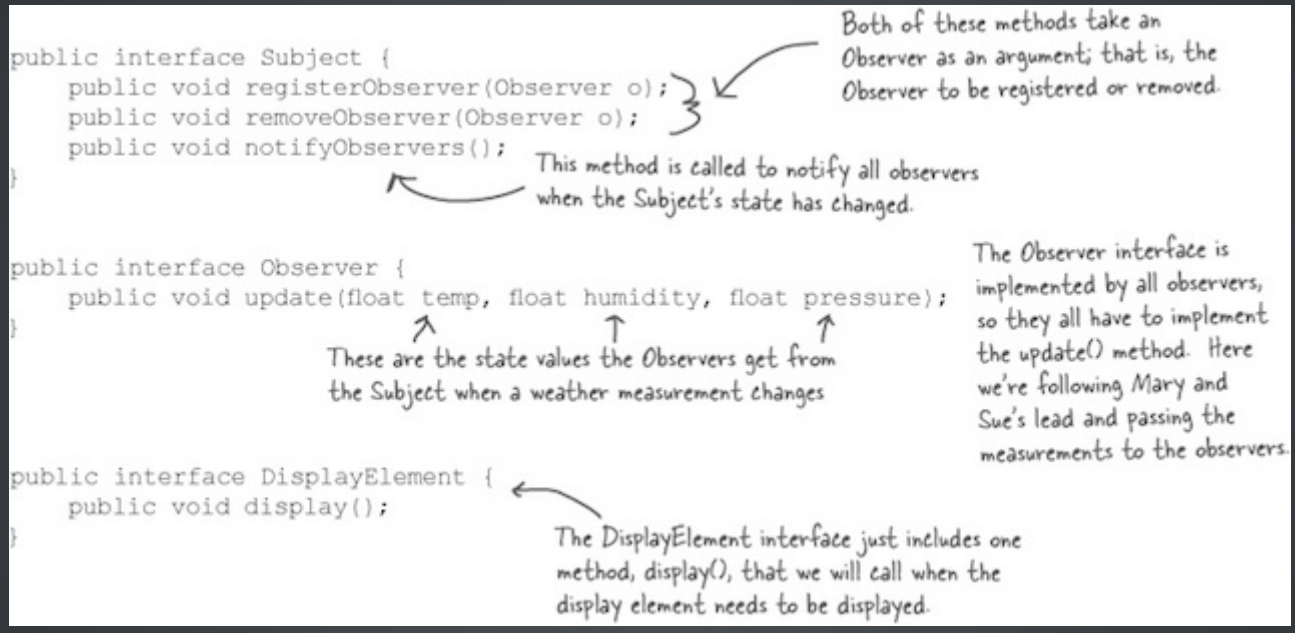





The behavior variables are declared as the behavior INTERFACE type.
These methods replace fly() and quack().
Instance variables hold a reference to a specific behavior at runtime.
Duck
FlyBehavior flyBehavior
QuackBehavior quackBehavior
performQuack()
swim()
display()
performFly()
// OTHER duck-like methods...
Flying Behaviors
Quacking Behaviors
Duck Behaviors

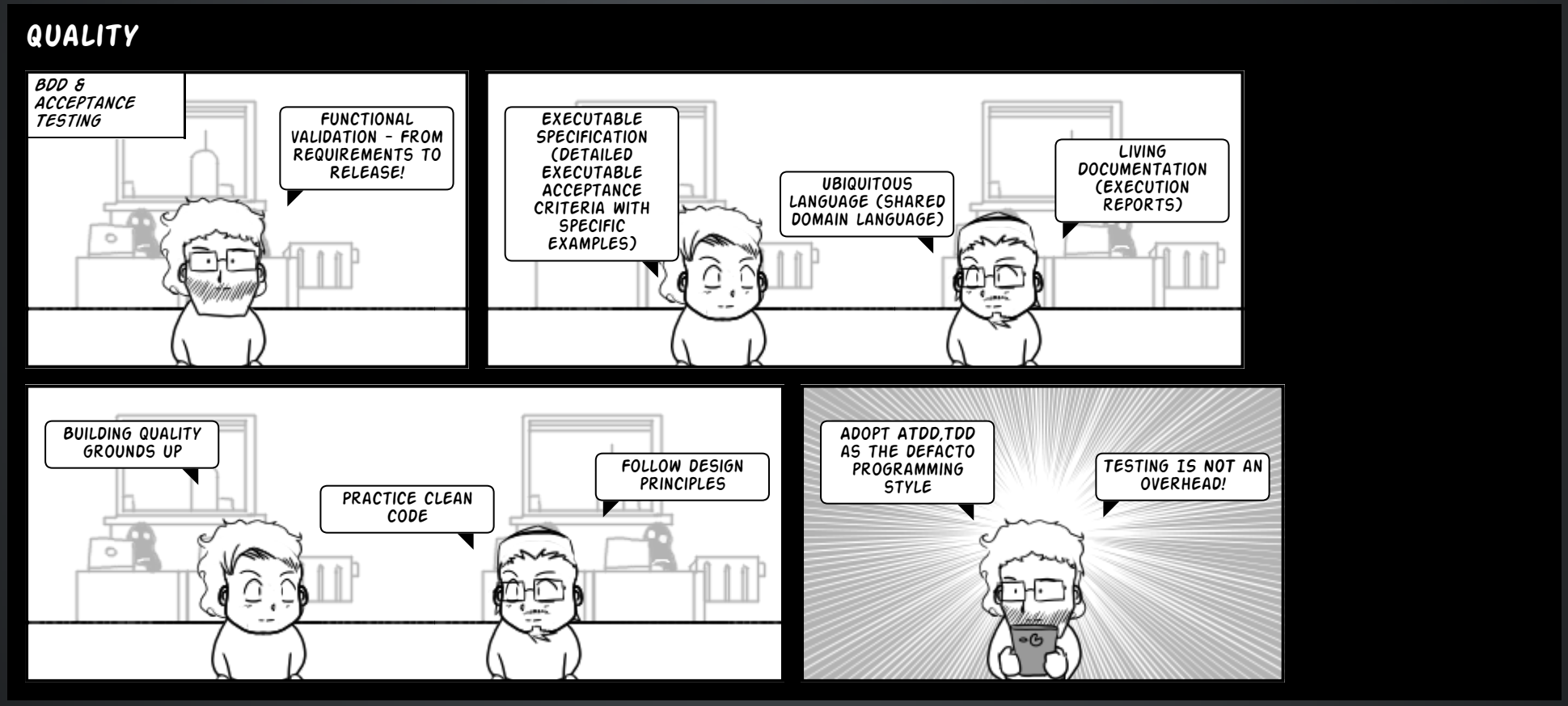

Client makes use of an encapsulated family of algorithms for both flying and quacking.
Client
Duck
FlyBehavior flyBehavior
QuackBehavior quackBehavior
swim()
display()
performQuack()
performFly()
setFlyBehavior()
setQuackBehavior()
// OTHER duck-like methods...
MallardDuck
display() {
// looks like a mallard }
RedheadDuck
display() {
// looks like a redhead }
RubberDuck
display() {
// looks like a rubberduck }
DecoyDuck
display() {
// looks like a decoy duck }
Encapsulated fly behavior
<<interface>>
FlyBehavior
fly()
FlyWithWings
fly() {
// implements duck flying
}
FlyNoWay
fly() {
// do nothing - can't fly!
}
Think of each set of behaviors as a family of algorithms.
Encapsulated quack behavior
<<interface>>
QuackBehavior
quack()
Quack
quack() {
// implements duck quacking
}
Squeak
quack() {
// rubber duckie squeak
}
MuteQuack
quack() {
// do nothing - can't quack!
}
These ~~behaviors~~ "algorithms" are interchangeable.

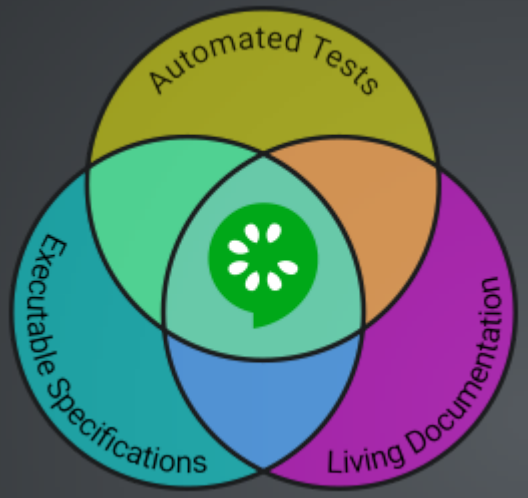

Here's the Subject interface. Objects use this interface to register as observers and also to remove themselves from being observers.
Each subject can have many observers.
All potential observers need to implement the Observer interface. This interface just has one method, update(), that gets called when the Subject's state changes.
<<interface>>
Subject
registerObserver()
removeObserver()
notifyObservers()
observers
<<interface>>
Observer
update()
ConcreteSubject
registerObserver() {...}
removeObserver() {...}
notifyObservers() {...}
getState()
setState()
subject
ConcreteObserver
update()
// other Observer specific methods
The concrete subject may also have methods for setting and getting its state (more about this later).
Concrete observers can be any class that implements the Observer interface. Each observer registers with a concrete subject to receive updates.
A concrete subject always implements the Subject interface. In addition to the register and remove methods, the concrete subject implements a notifyObservers() method that is used to update all the current observers whenever state changes.


public interface Subject {
public void registerObserver(Observer o);
public void removeObserver(Observer o);
public void notifyObservers();
}
Both of these methods take an Observer as an argument; that is, the Observer to be registered or removed.
This method is called to notify all observers when the Subject's state has changed.
public interface Observer {
public void update(float temp, float humidity, float pressure);
}
These are the state values the Observers get from the Subject when a weather measurement changes
The Observer interface is implemented by all observers, so they all have to implement the update() method. Here we're following Mary and Sue's lead and passing the measurements to the observers.
public interface DisplayElement {
public void display();
}
The DisplayElement interface just includes one method, display(), that we will call when the display element needs to be displayed.

https://dzone.com/refcardz/design-patterns

ENSURING QUALITY....





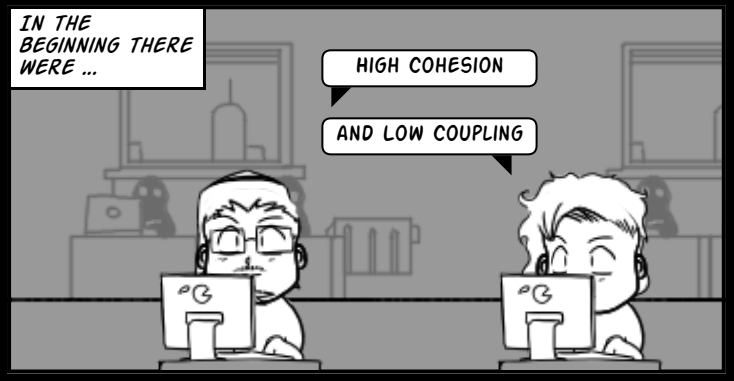
benefits of bdd

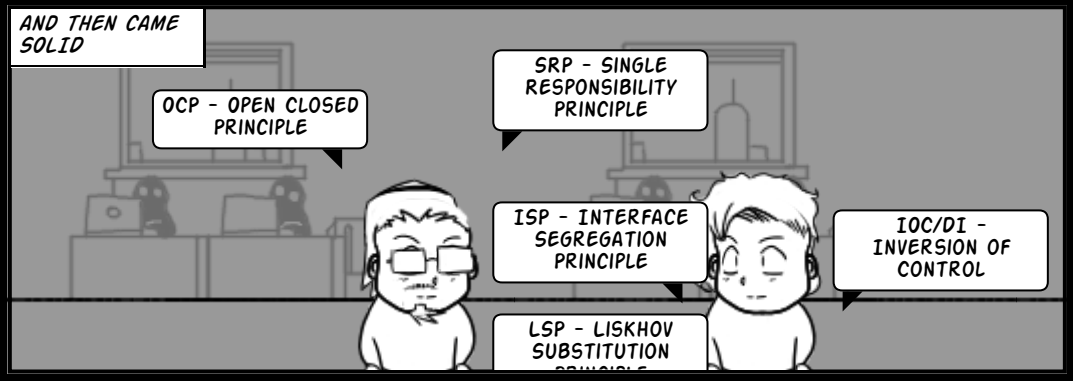
BDD Frameworks

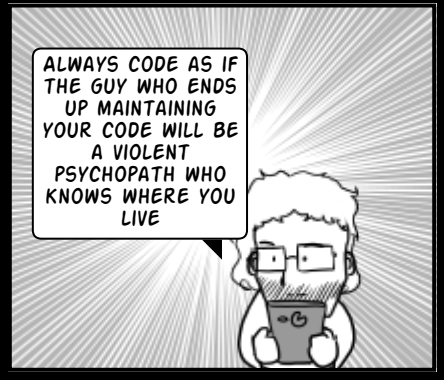
Book: BDD in Action

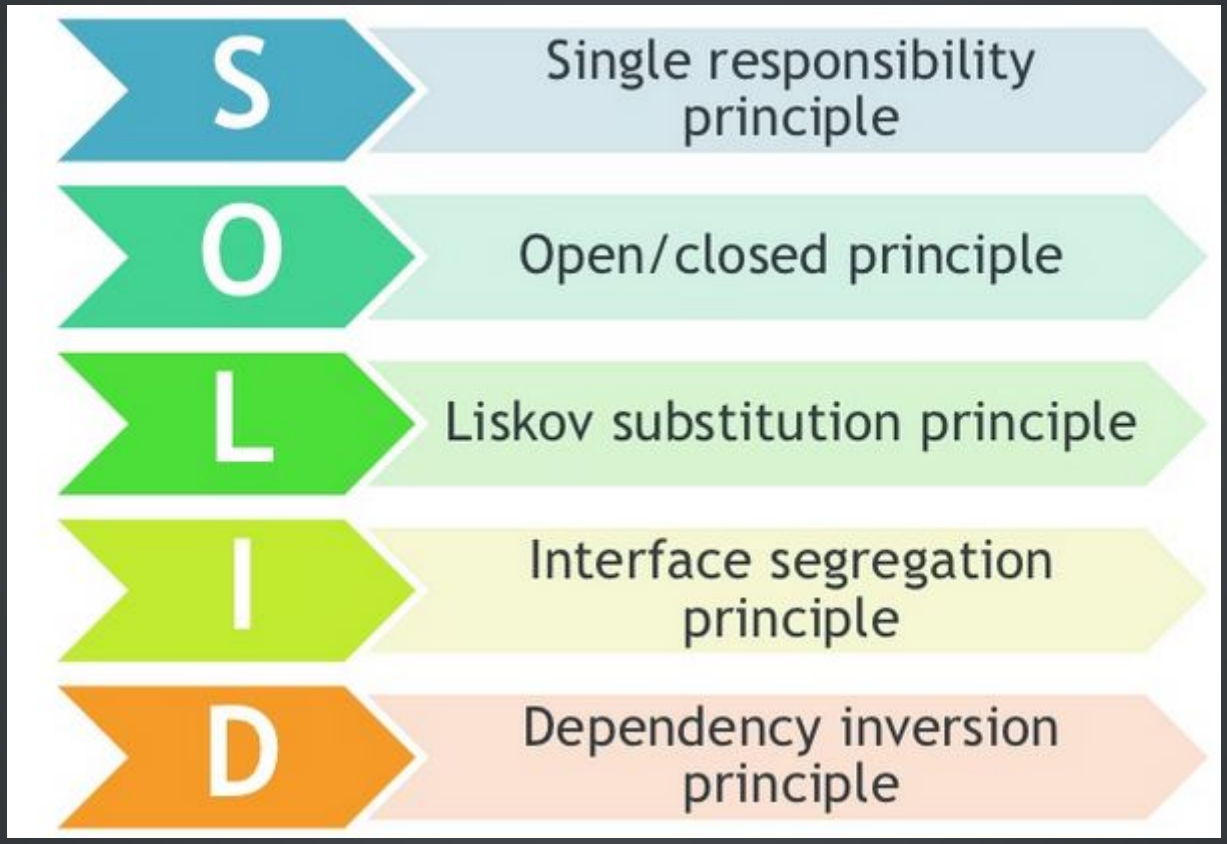
BDD - Introduction from Dan North

DESIGN PRINCIPLES



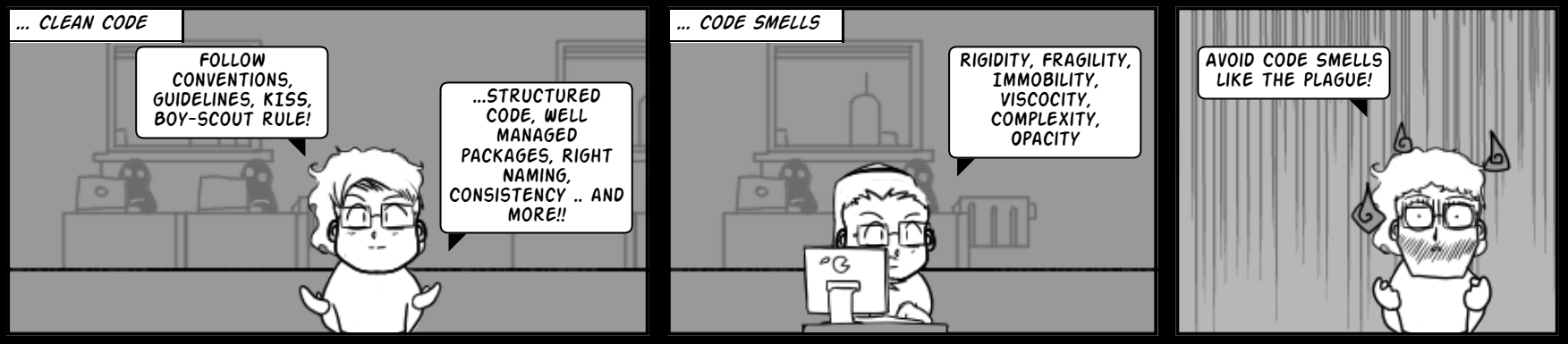





S
Single responsibility principle
O
Open/closed principle
L
Liskov substitution principle
I
Interface segregation principle
D
Dependency inversion principle

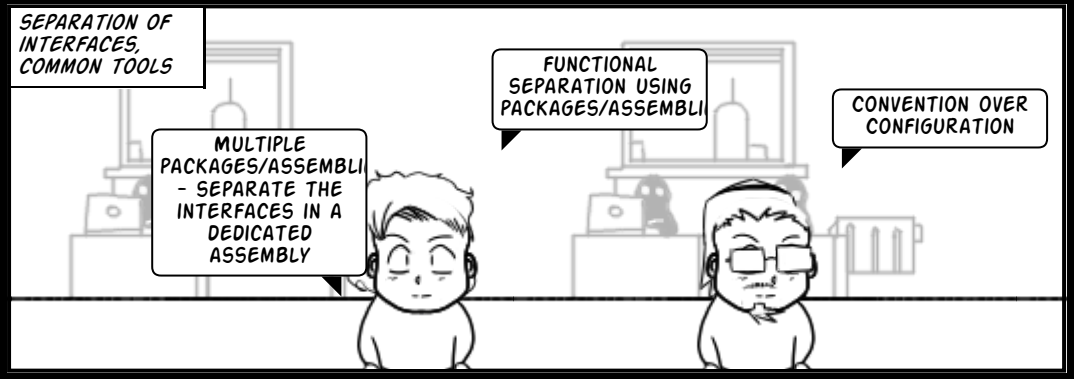
https://leanpub.com/solid/read#leanpub-auto-test-driven-design---tdd
http://principles-wiki.net/principles:start

CODING BEST PRACTICES

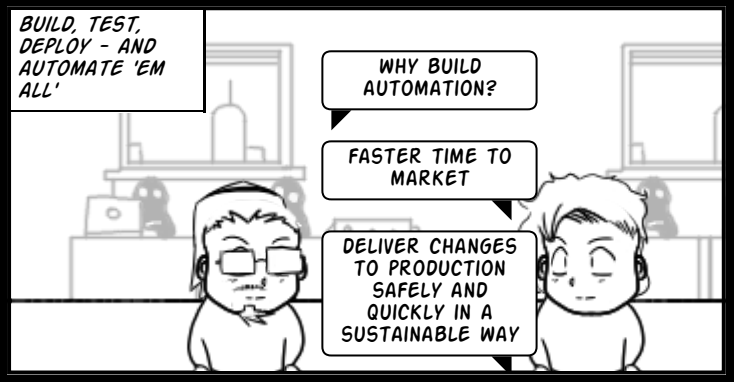


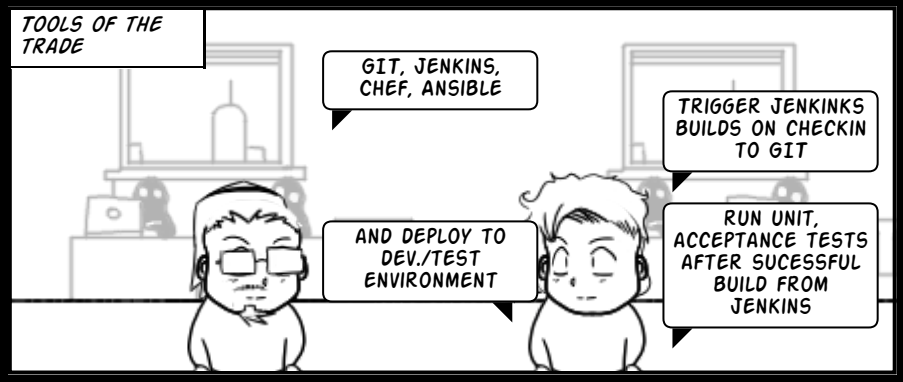
http://www.planetgeek.ch/wp-content/uploads/2013/06/Clean-Code-V2.1.pdf
https://www.amazon.in/Clean-Code-Handbook-Software-Craftsmanship-ebook/
http://www.hanselman.com/blog/SixEssentialLanguageAgnosticProgrammingBo
https://github.com/chhantyal/influential-cs-books

CONTD.. ENSURING QUALITY - PACKAGING FOR DELIVERY

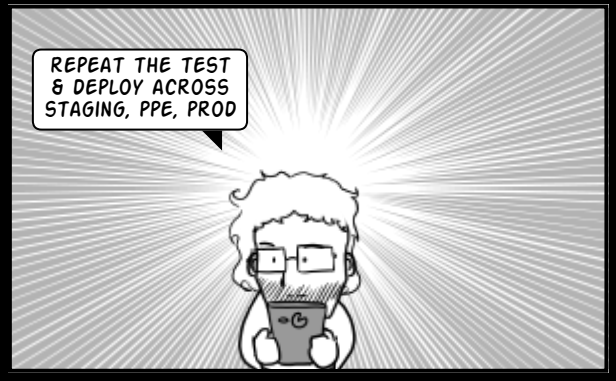


SCM, CI/CD

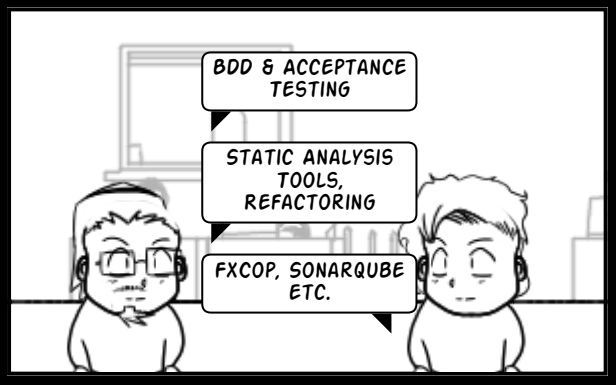

BUILD, TEST, DEPLOY - AND AUTOMATE 'EM ALL'
WHY BUILD AUTOMATION?
FASTER TIME TO MARKET
DELIVER CHANGES TO PRODUCTION SAFELY AND QUICKLY IN A SUSTAINABLE WAY

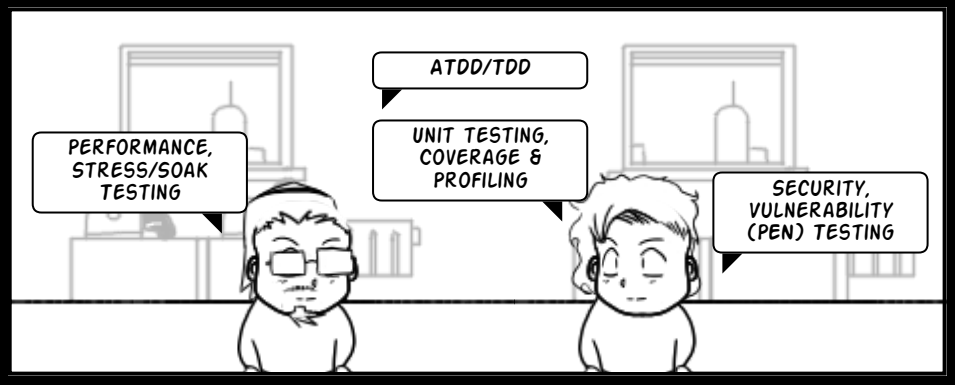

TOOLS OF THE TRADE
GIT, JENKINS, CHEF, ANSIBLE
TRIGGER JENKINKS BUILDS ON CHECKIN TO GIT
AND DEPLOY TO DEV./TEST ENVIRONMENT
RUN UNIT, ACCEPTANCE TESTS AFTER SUCESSFUL BUILD FROM JENKINS


CANARY RELEASES / BLUE GREEN DEPLOYMENT
WITH LEAST DISTRUPTION TO THE ENVIRONMENTS


REPEAT THE TEST & DEPLOY ACROSS STAGING, PPE, PROD

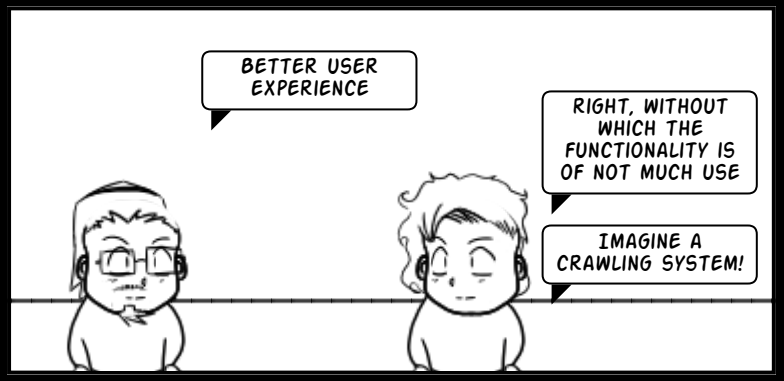
BUILDING THE APP & SERVICE - COMPONENTS, ENSURING QUALITY

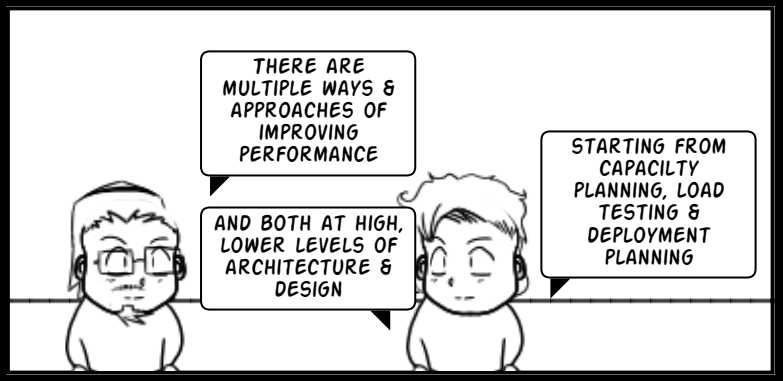




PERFORMANCE

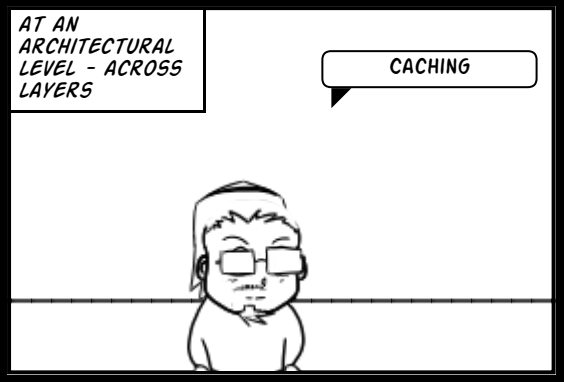
WHY IMPROVE PERFORMANCE?

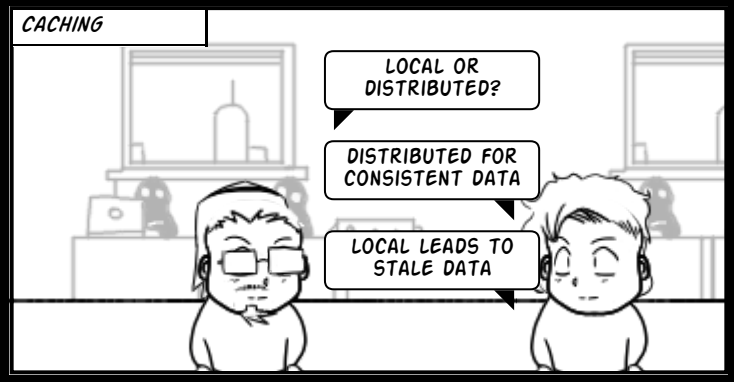


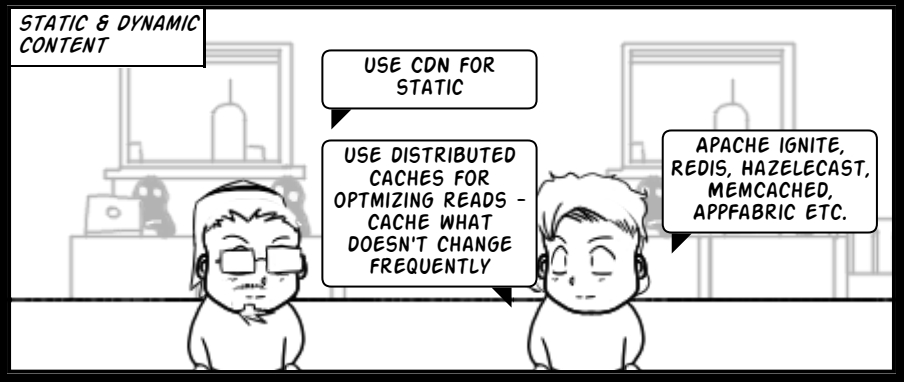


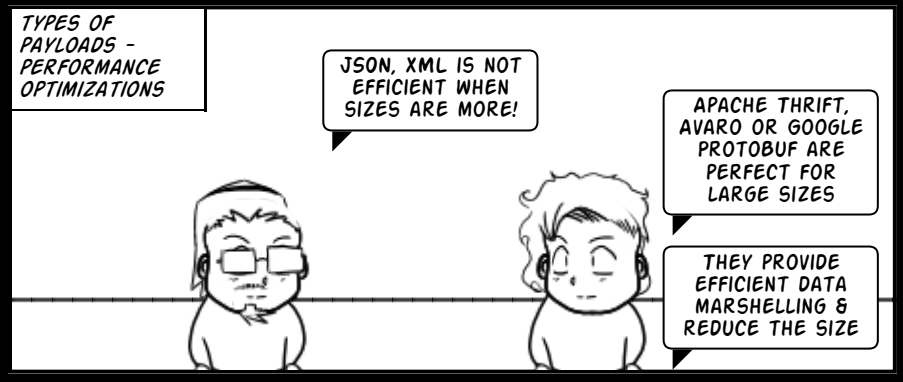
http://perftesting.codeplex.com/
http://guidanceengineering.codeplex.com/
https://msdn.microsoft.com/en-us/library/ms998408.aspx
https://msdn.microsoft.com/practices

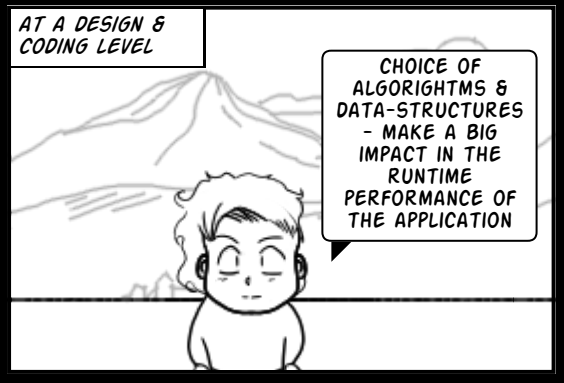
HOW TO IMPROVE PERFORMANCE

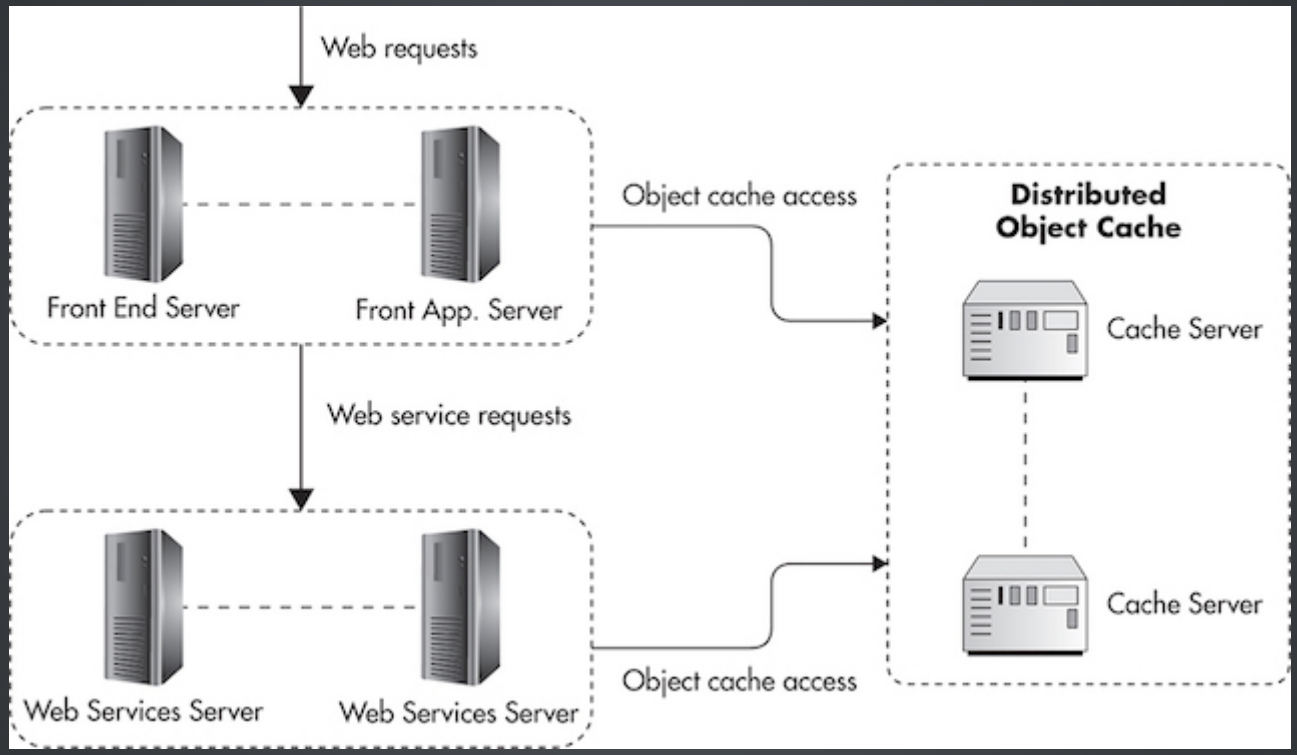




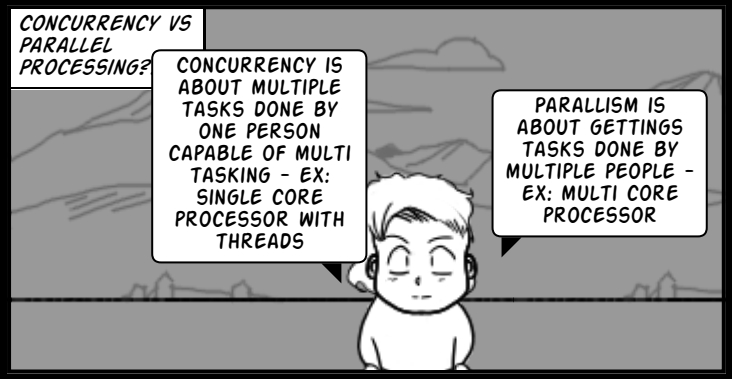


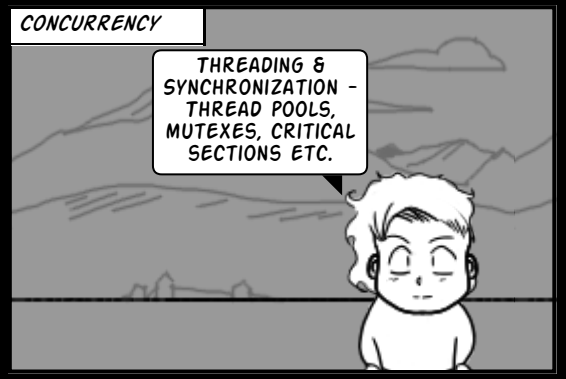


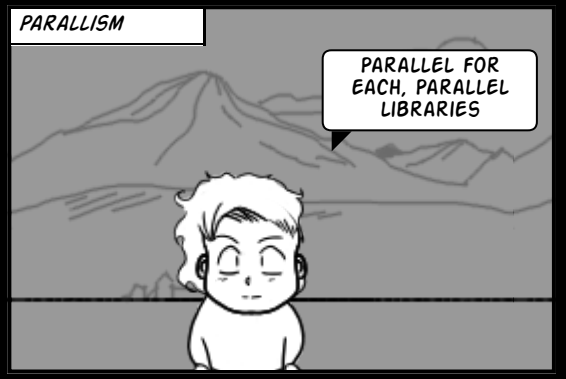


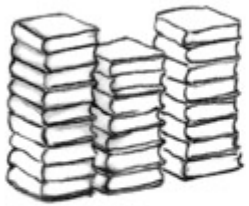

Web requests
Front End Server
Front App. Server
Object cache access
Distributed
Object Cache
Cache Server
Web service requests
Cache Server
Web Services Server
Web Services Server
Object cache access

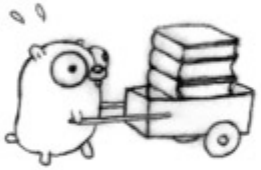
CONCURRENCY & PARALLELISM

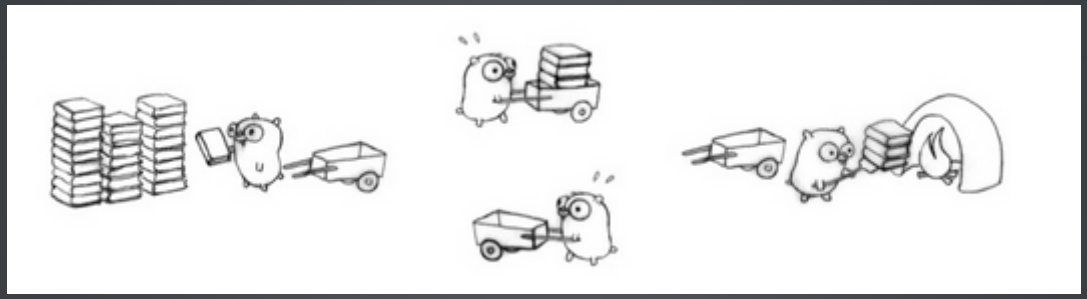


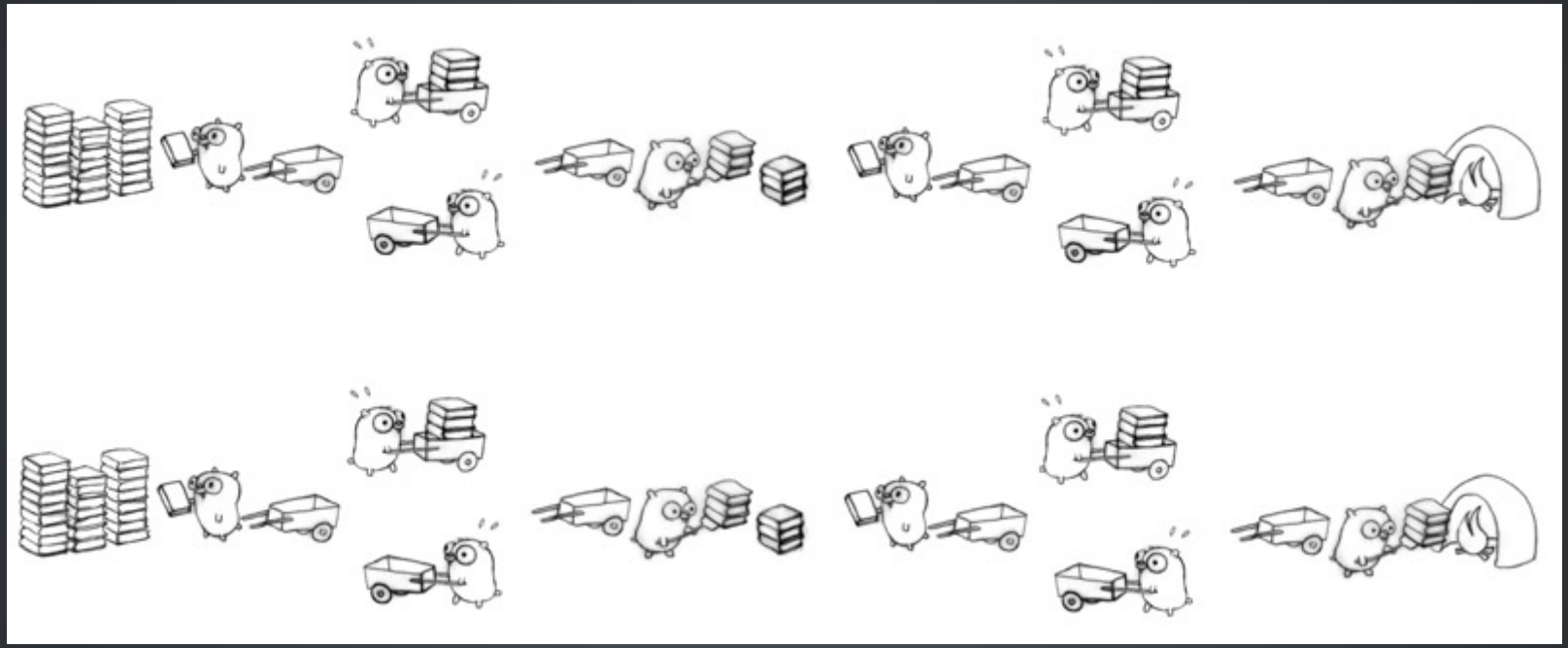




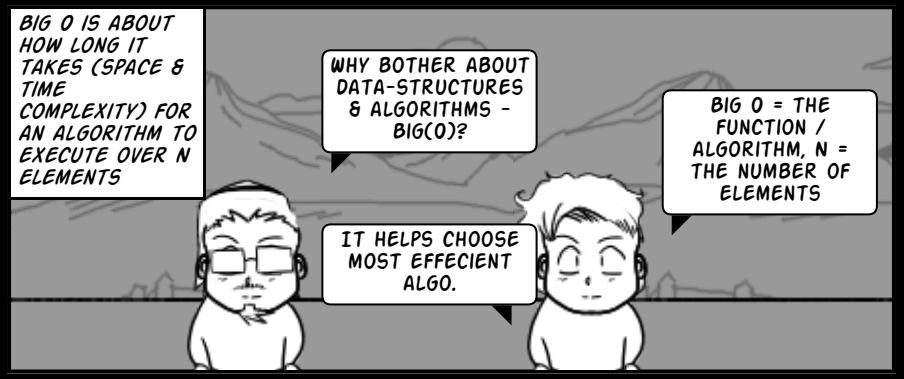


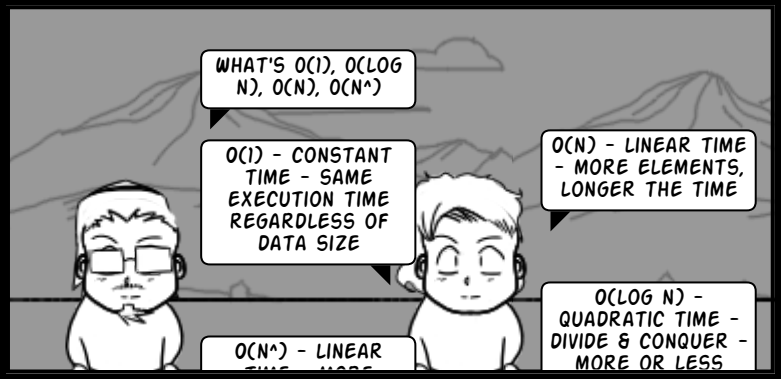






OPTIMIZATIONS - SPACE & TIME COMPLEXITY





Big O

HOW TIME SCALES WITH RESPECT TO SOME INPUT VARIABLES

1. Different steps get added
2. Drop constants
3. Different inputs => different variables

4. Drop non-dominate terms

```
function whyWouldIDoThis(array){
    max = NULL
O(n) { for each a in array {
           max = MAX(a, max)
       }
       print max

O(n²) { for each a in array {
          for each b in array {
            print a, b
          }
        }
}
```

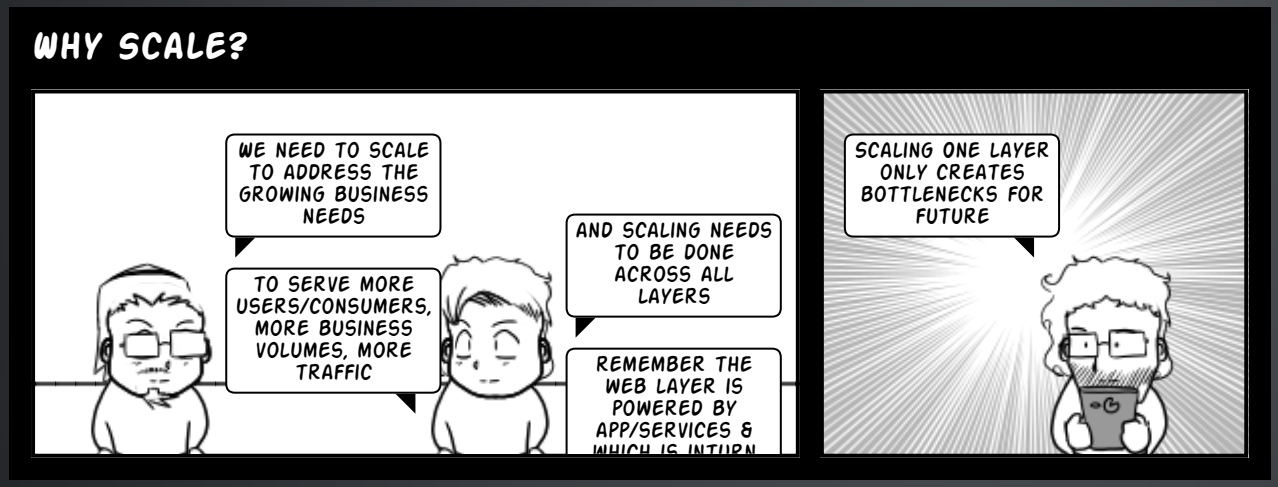
$O(n^2) \leq O(n+n^2) \leq O(n^2+n^2)$

if LEFT and RIGHT are equivalent (see RULE 2), then CENTER is too

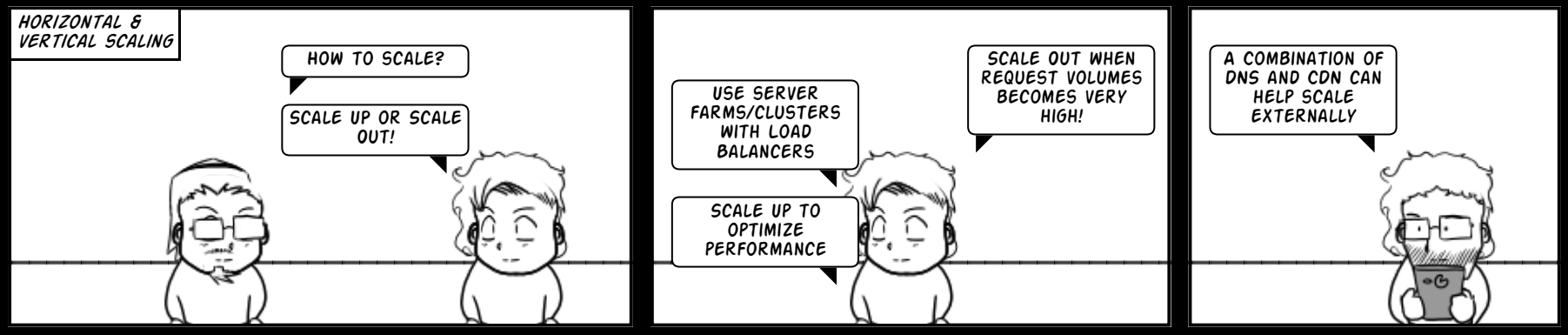
$O(n+n^2) \Rightarrow O(n^2)$

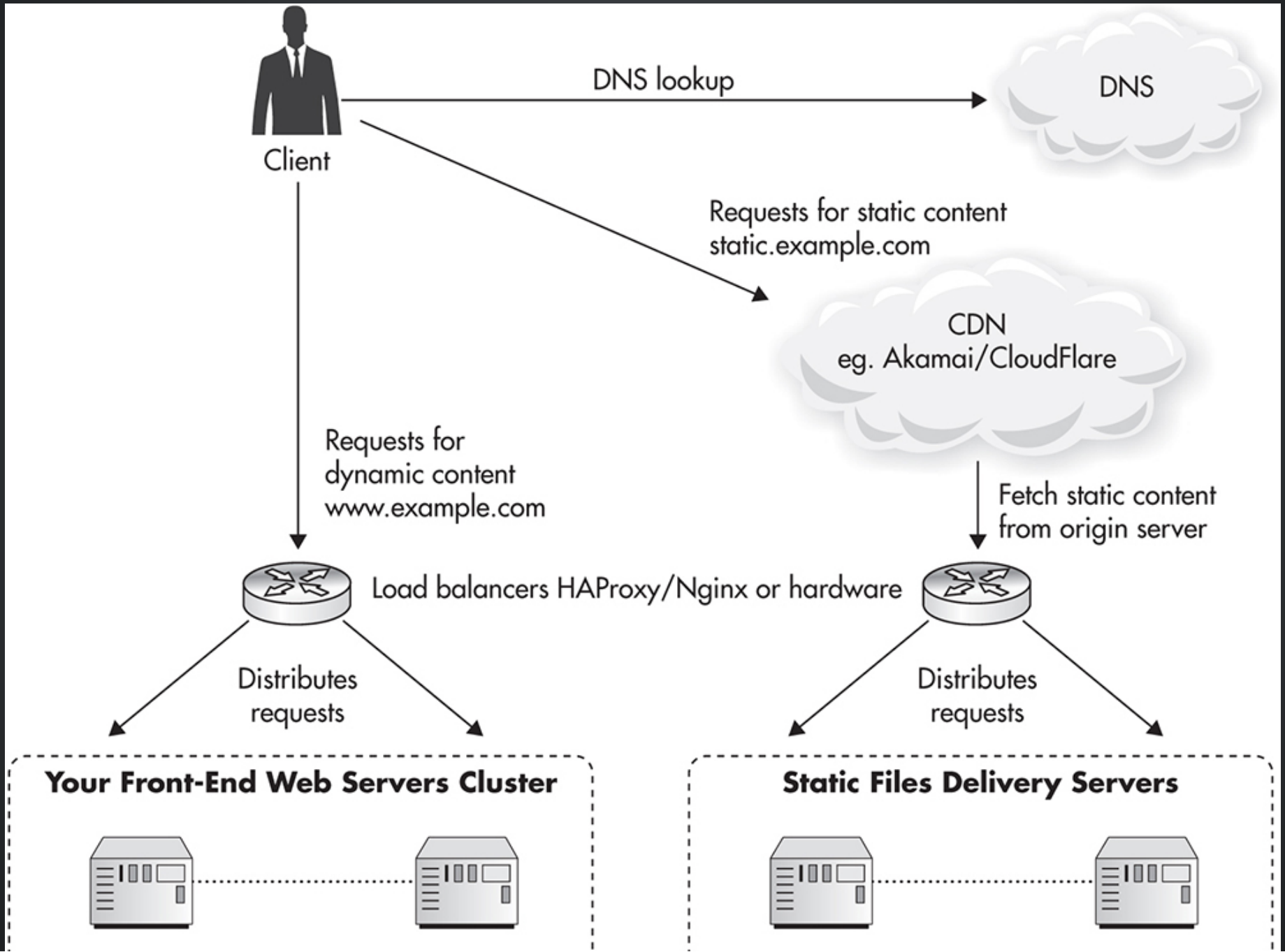
https://rob-bell.net/2009/06/a-beginners-guide-to-big-o-notation/
https://www.youtube.com/watch?v=__vX2sjlpXU
https://www.youtube.com/watch?v=v4cd1O4zkGw
https://www.youtube.com/watch?v=-Eiw_-v__Vo
http://bigocheatsheet.com/
https://en.wikipedia.org/wiki/Big_O_notation
http://adrianmejia.com/blog/2014/02/13/algorithms-for-dummies-part-1-sc
http://algorithmiccomplexity.com/
http://web.mit.edu/16.070/www/lecture/big_o.pdf

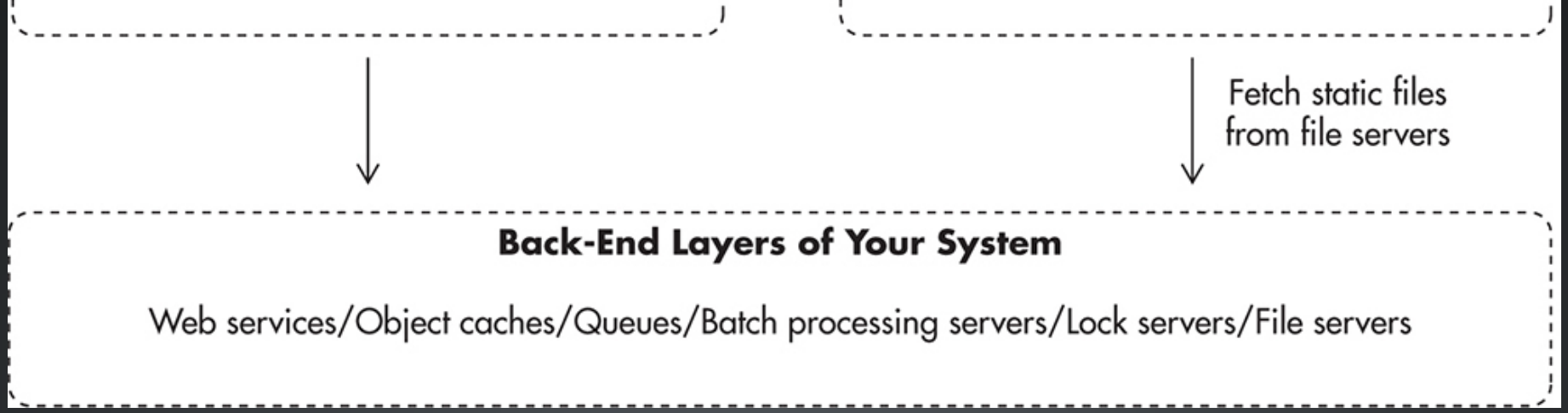
SCALABILITY



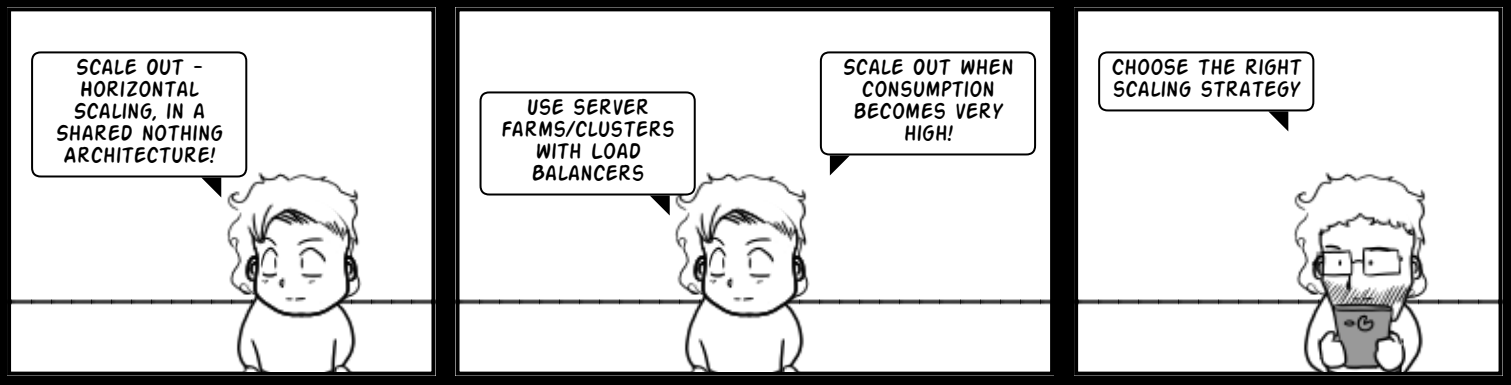
HOW TO SCALE THE WEB LAYER

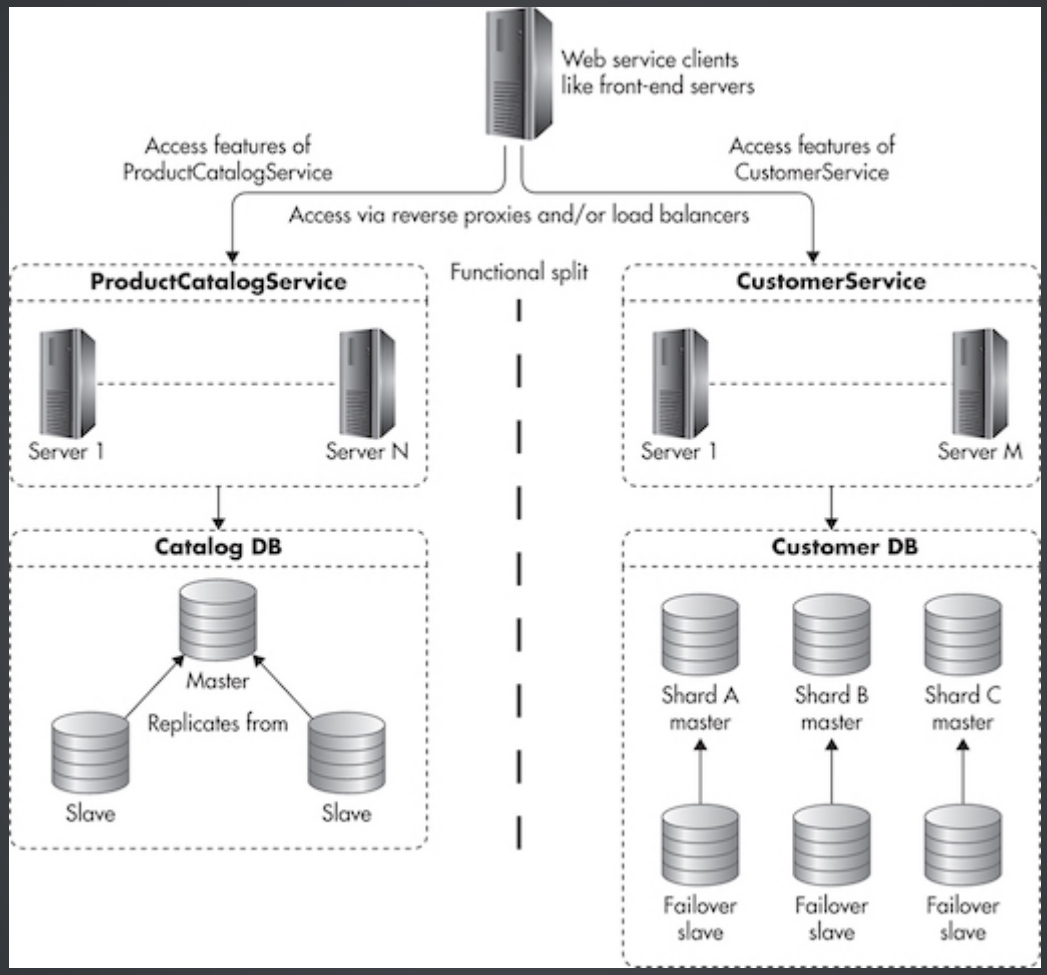



DNS lookup
DNS
Client
Requests for static content
static.example.com
CDN
eg. Akamai/CloudFlare
Requests for
dynamic content
www.example.com
Fetch static content
from origin server
Load balancers HAProxy/Nginx or hardware
Distributes
requests
Distributes
requests
Your Front-End Web Servers Cluster
Static Files Delivery Servers

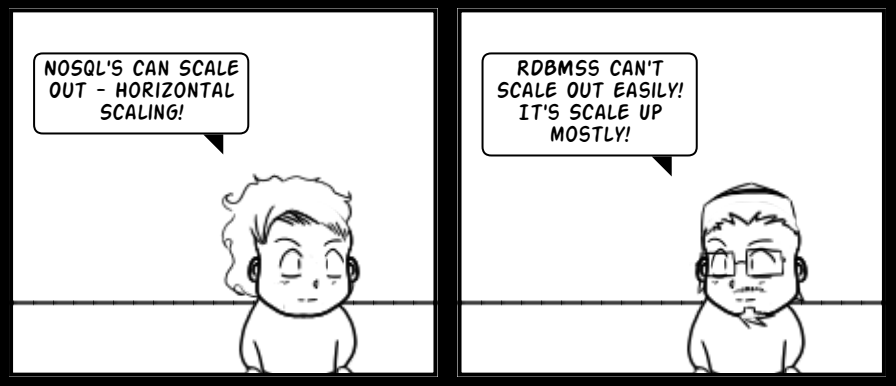

Fetch static files
from file servers
Back-End Layers of Your System
Web services/Object caches/Queues/Batch processing servers/Lock servers/File servers

HOW TO SCALE THE APP LAYER

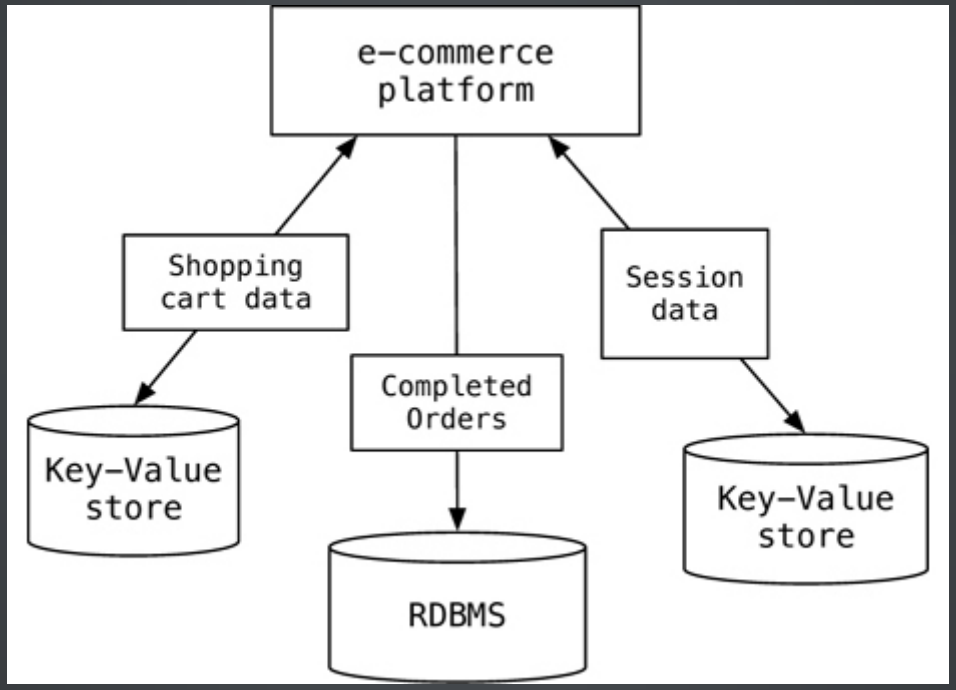



Web service clients
like front-end servers
Access features of
ProductCatalogService
Access features of
CustomerService
Access via reverse proxies and/or load balancers
ProductCatalogService
Functional split
CustomerService
Server 1
Server N
Server 1
Server M
Catalog DB
Customer DB
Master
Replicates from
Slave
Slave
Shard A
master
Shard B
master
Shard C
master
Failover
slave
Failover
slave
Failover
slave

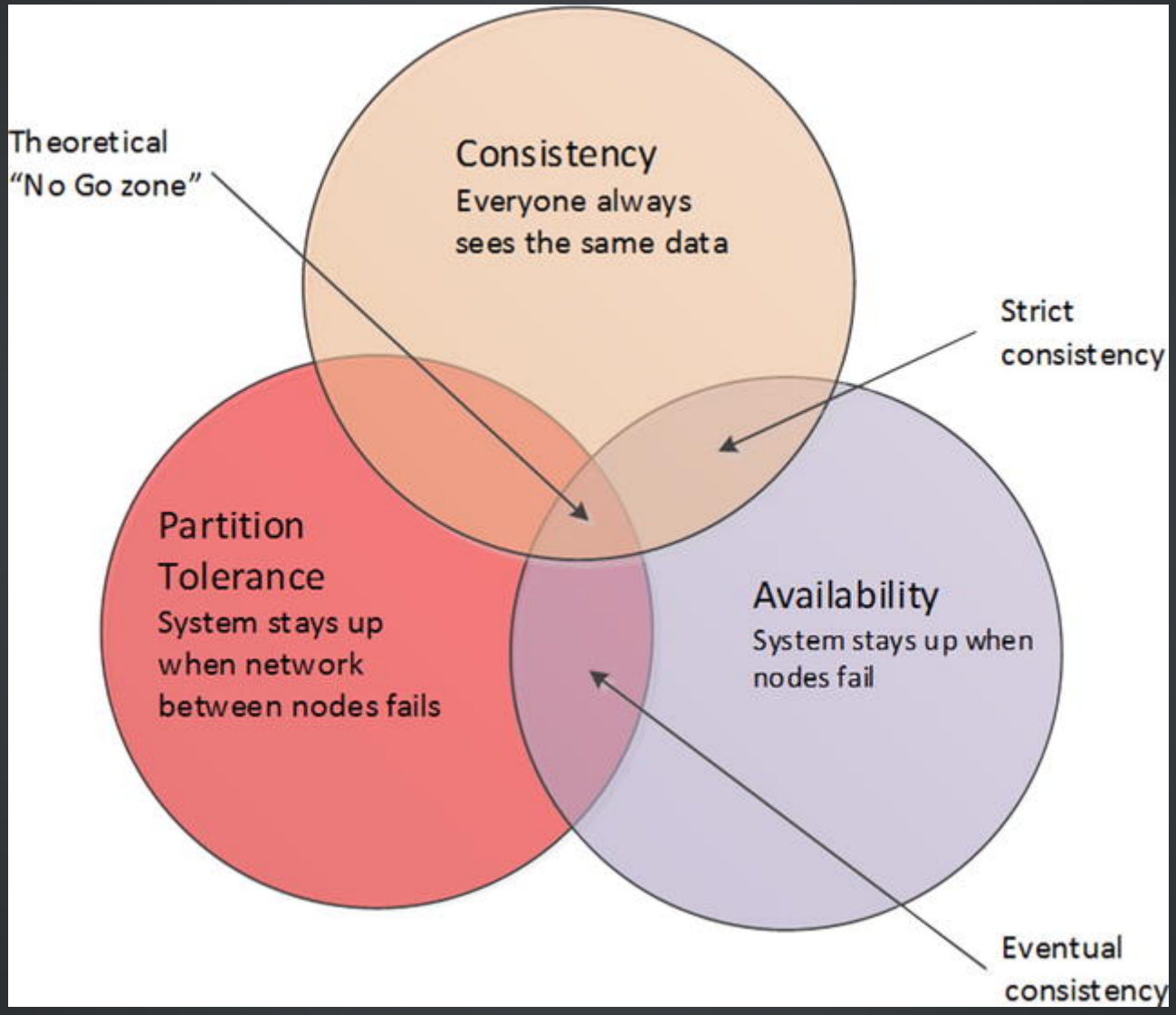
HOW TO SCALE THE DB LAYER



Scale Database Layer

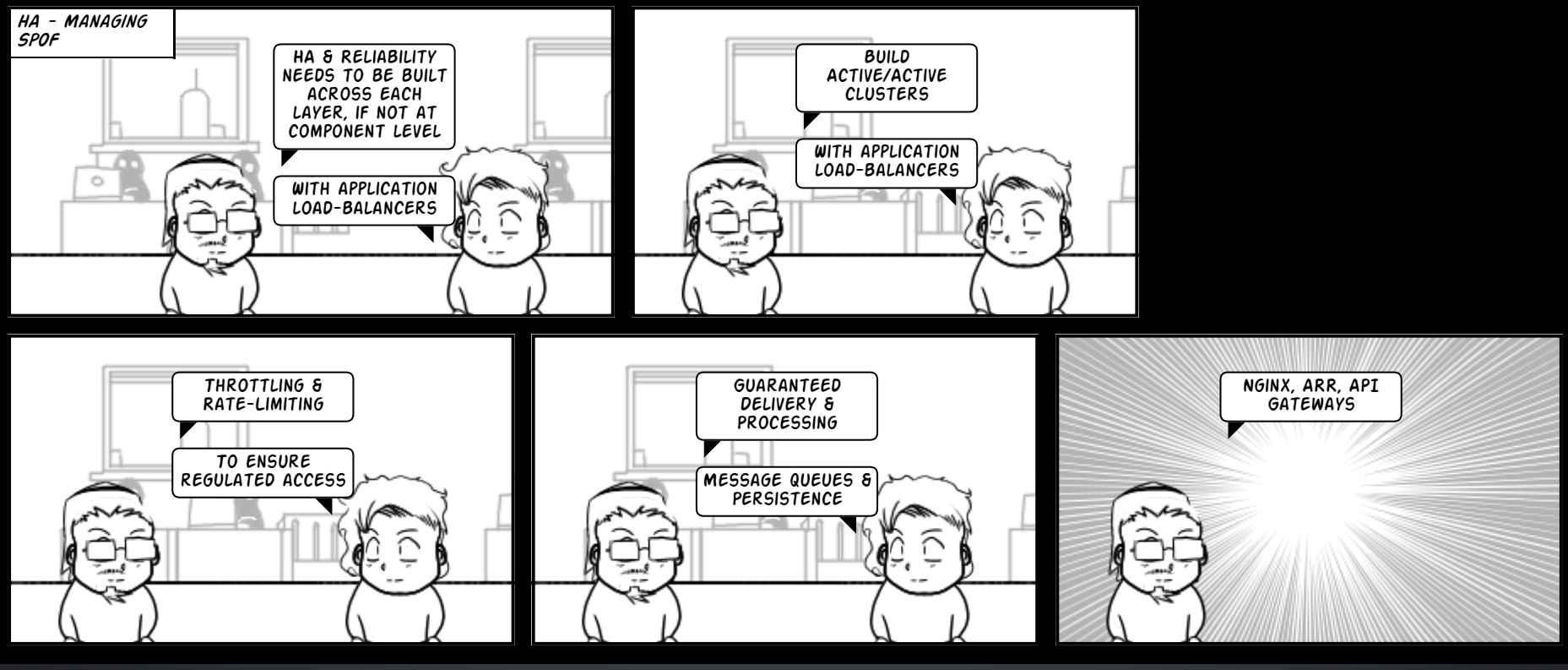


CAP Theorem Revisited

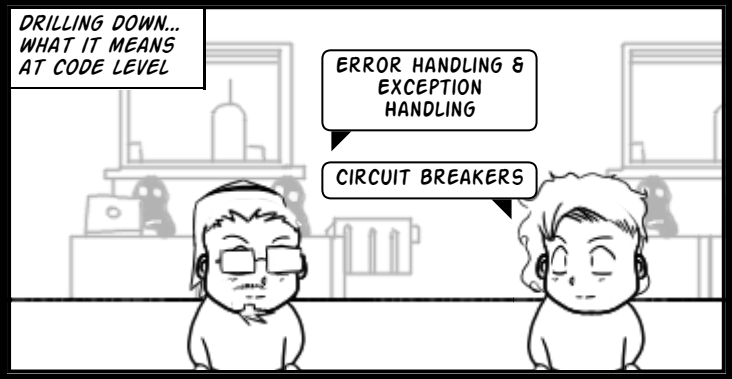


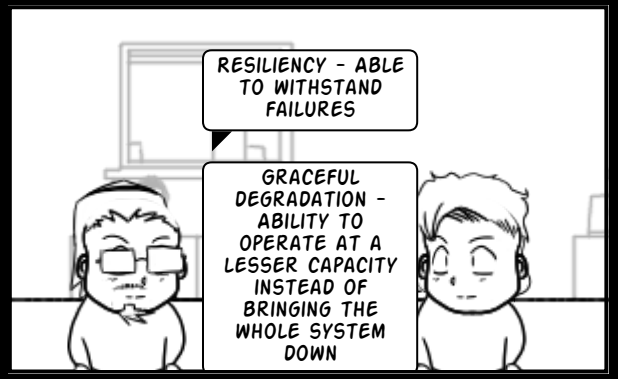
HA & RELIABILITY

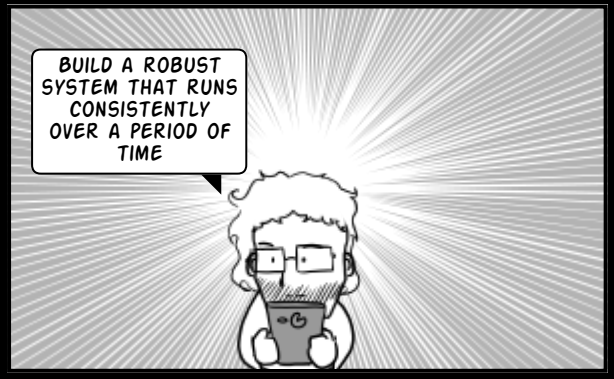
HOW TO ENSURE HIGH AVAILABILITY OF THE WEB LAYER



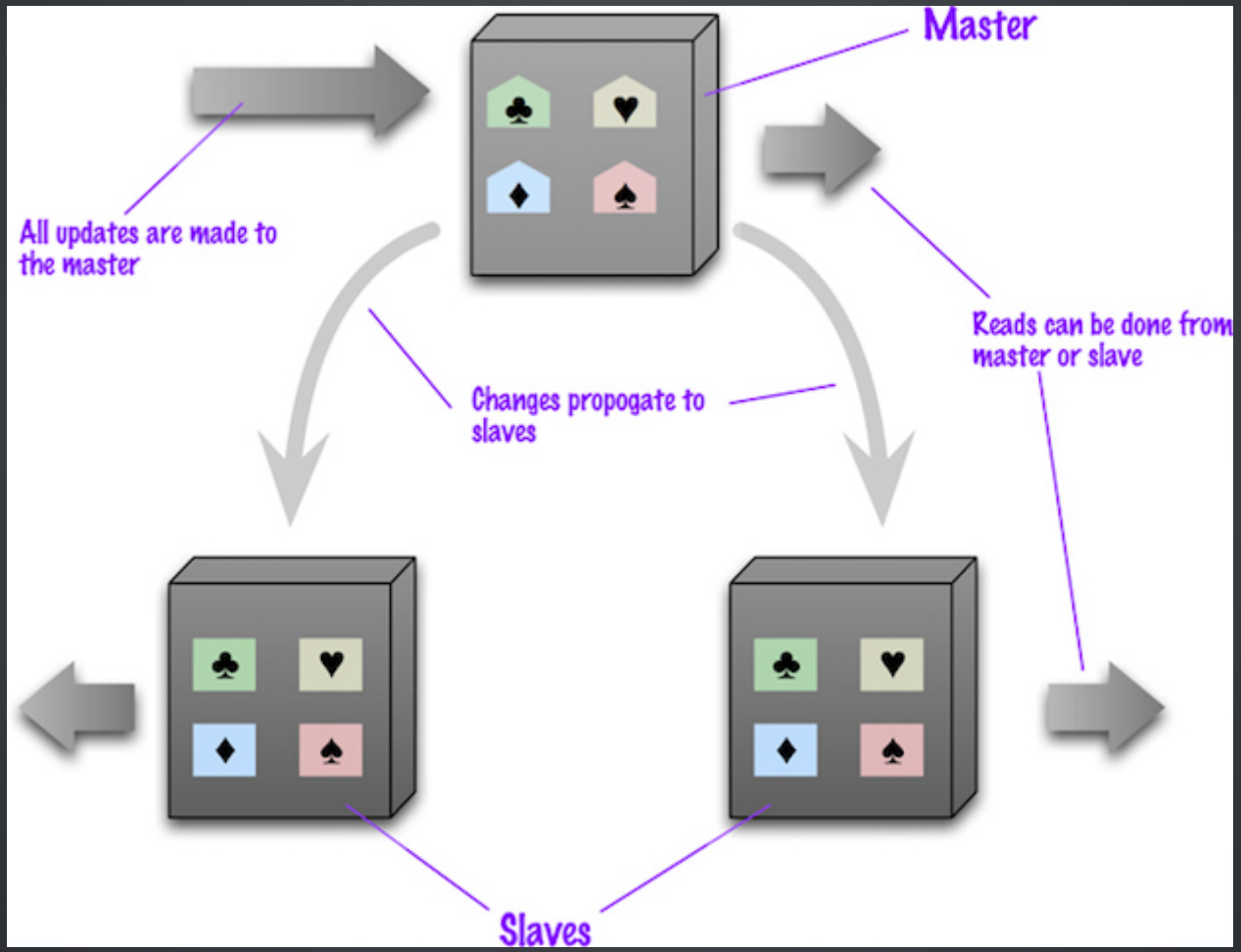
RELIABILITY, RESILIENCY & ROBUSTNESS



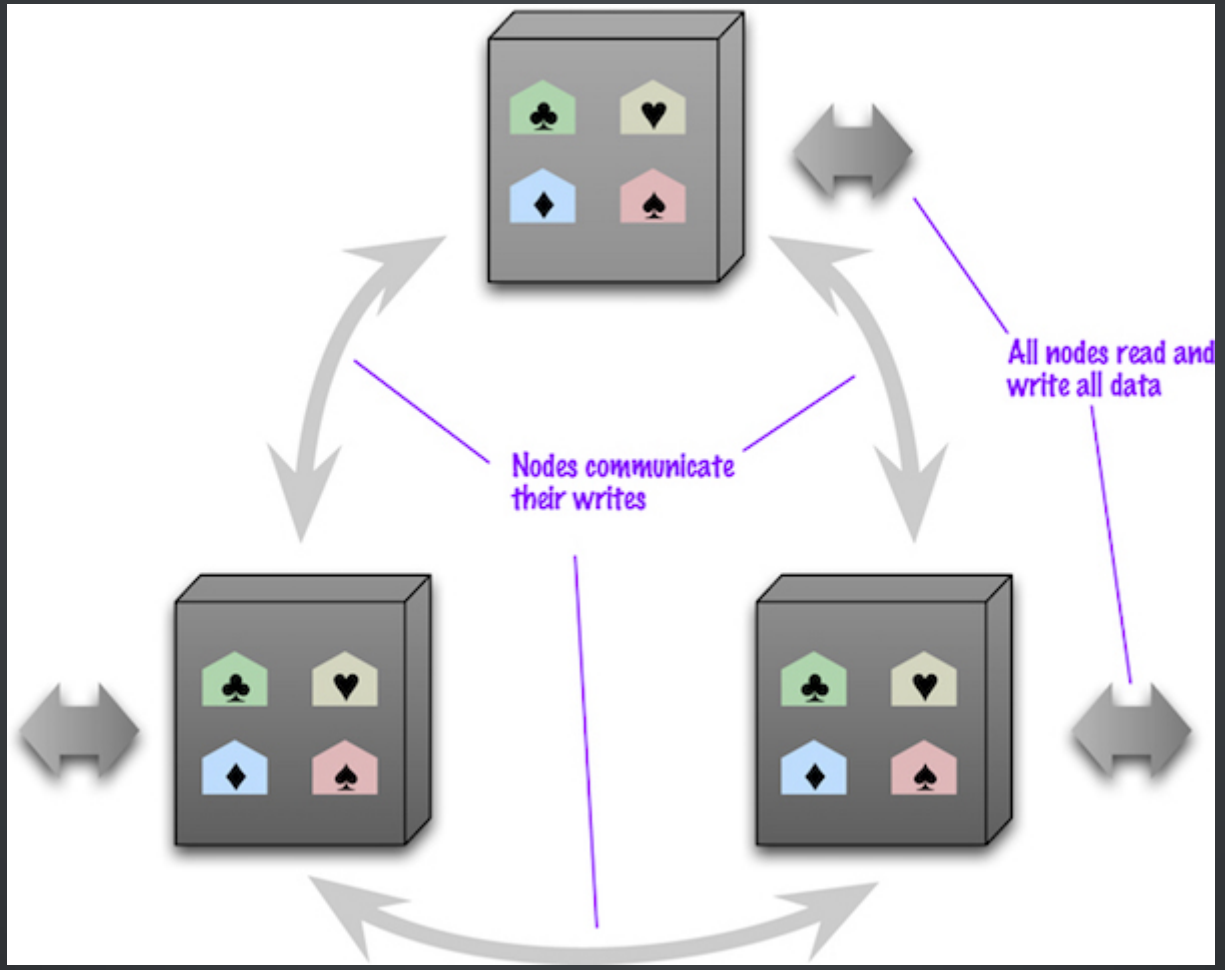




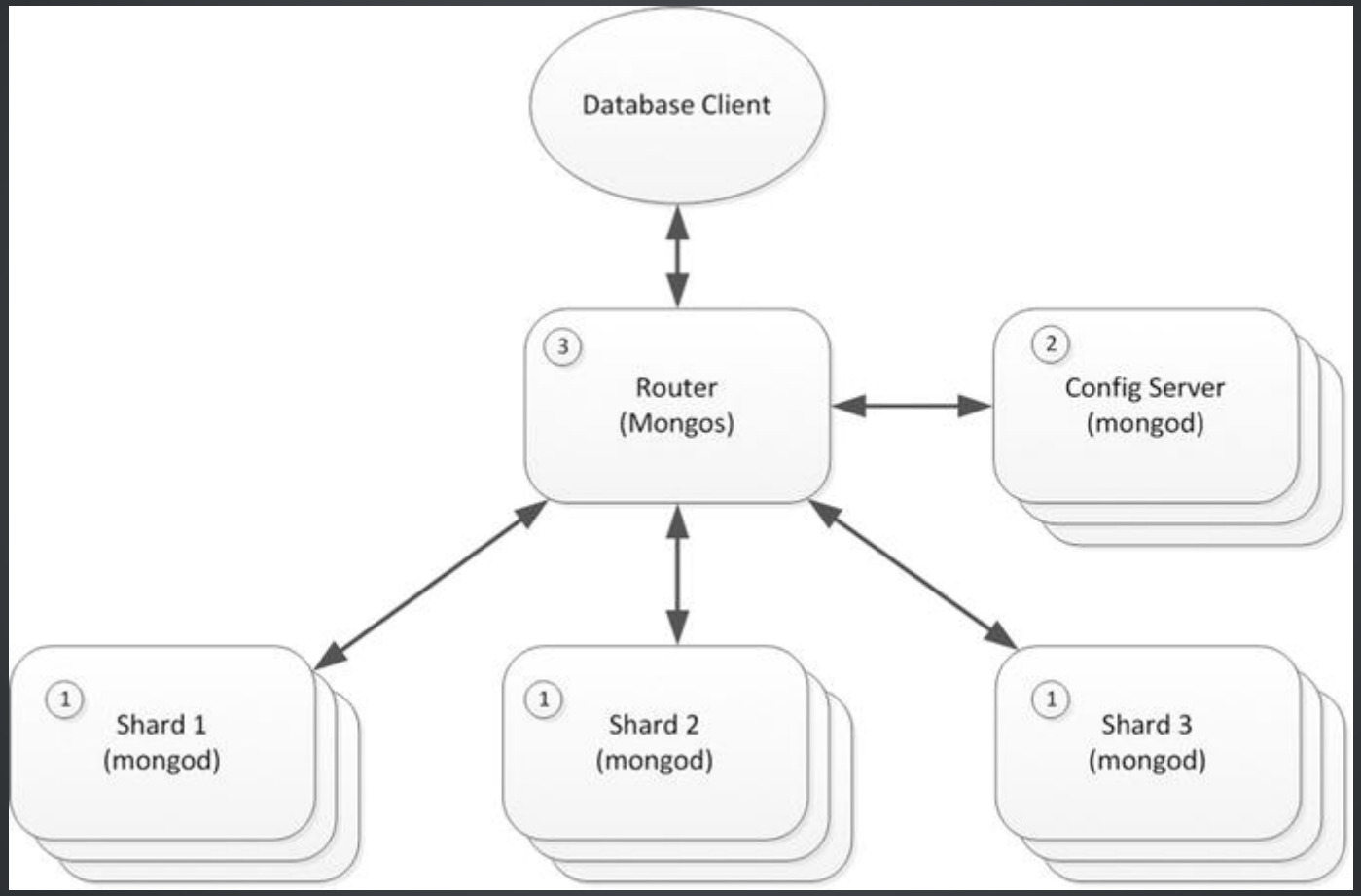
Master Slave
Master Slave replication



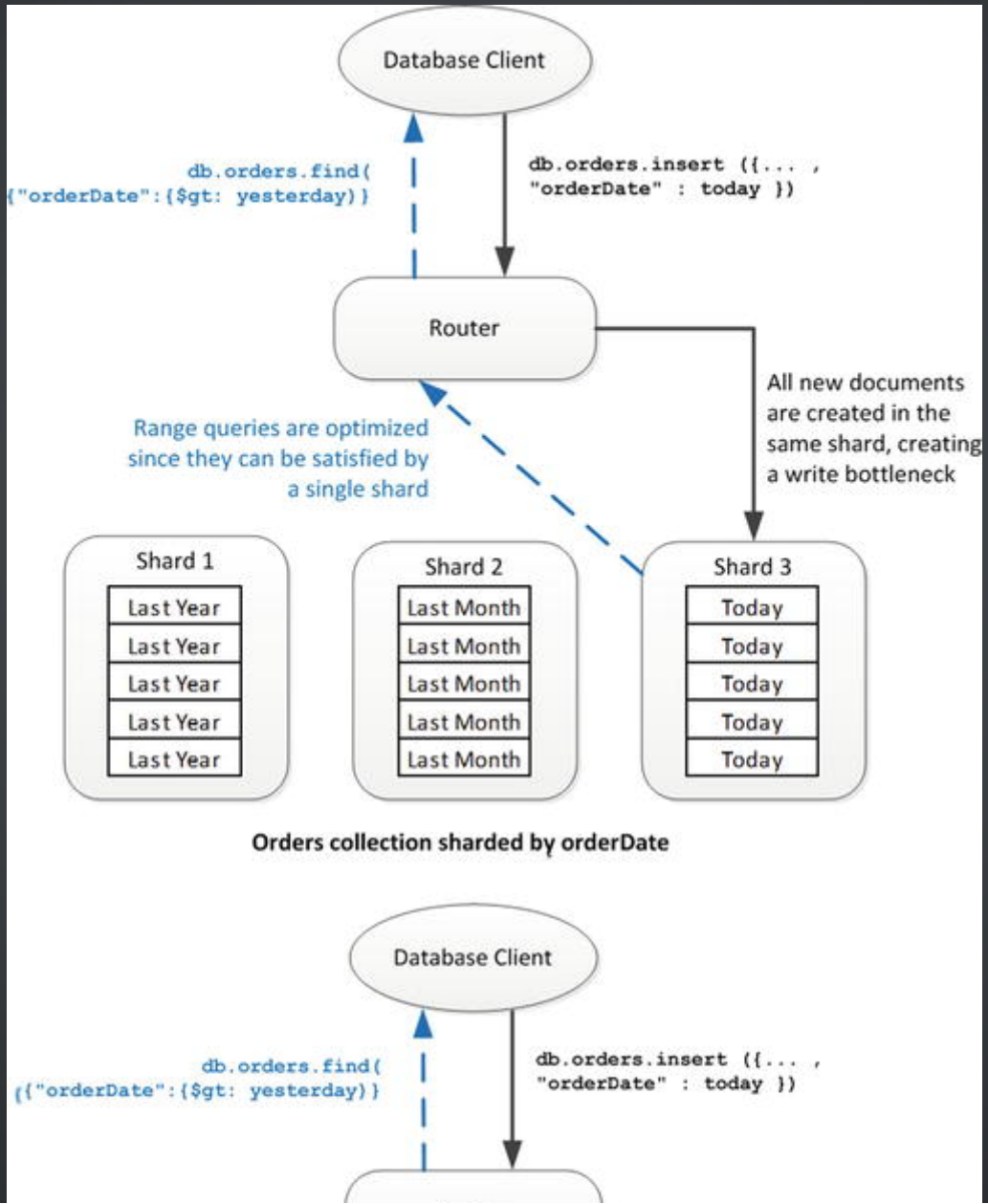
Master Slave
Peer to Peer replication

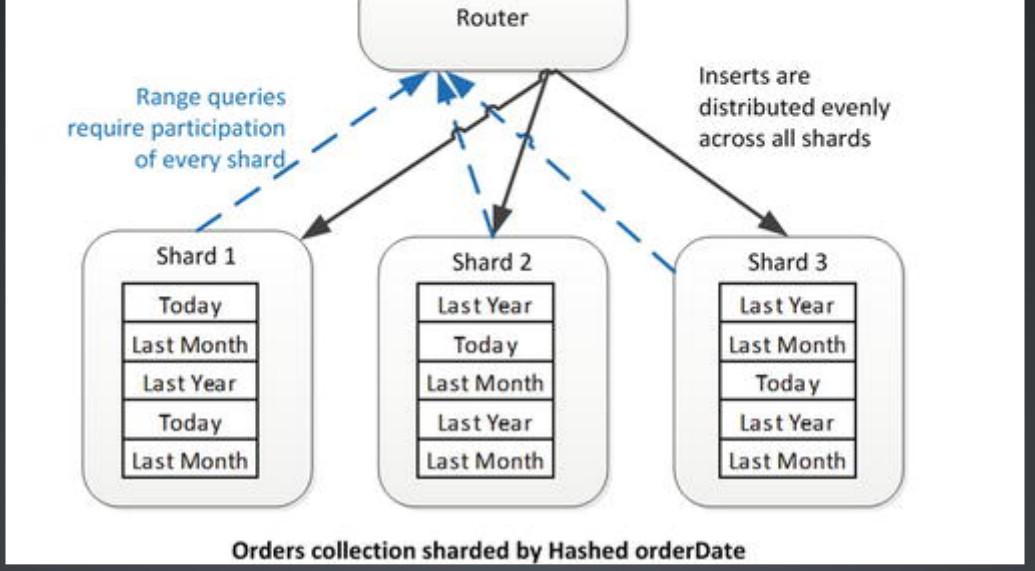


Sharding of NoSQL Database (Mongo DB)



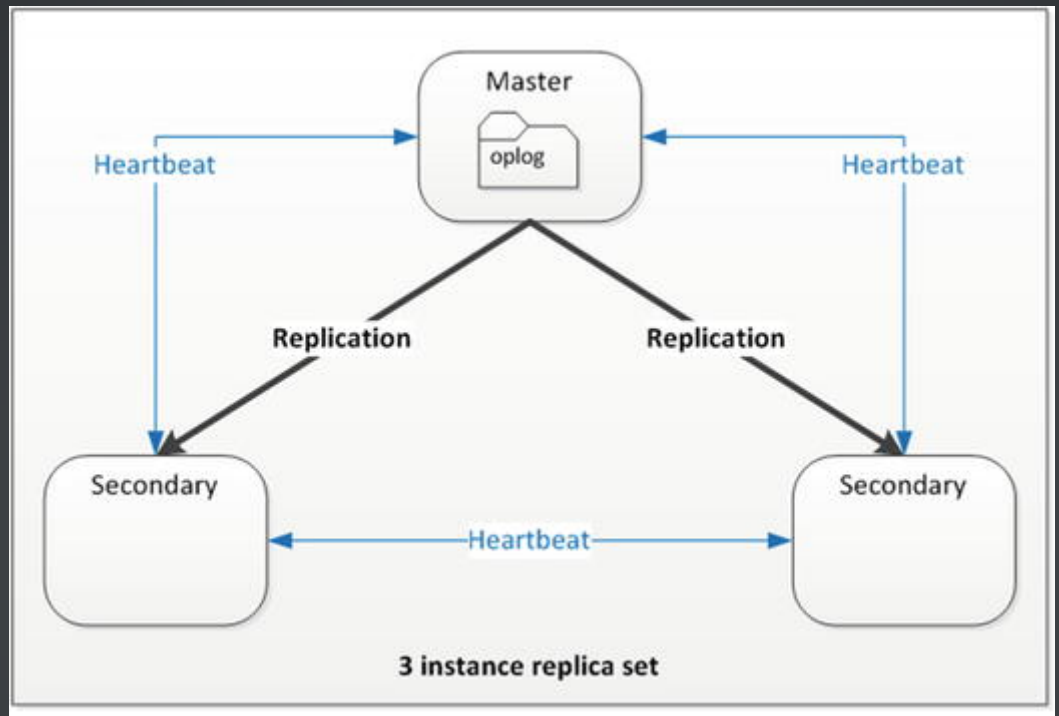
Sharding of NoSQL Database (Mongo DB .. Continued)

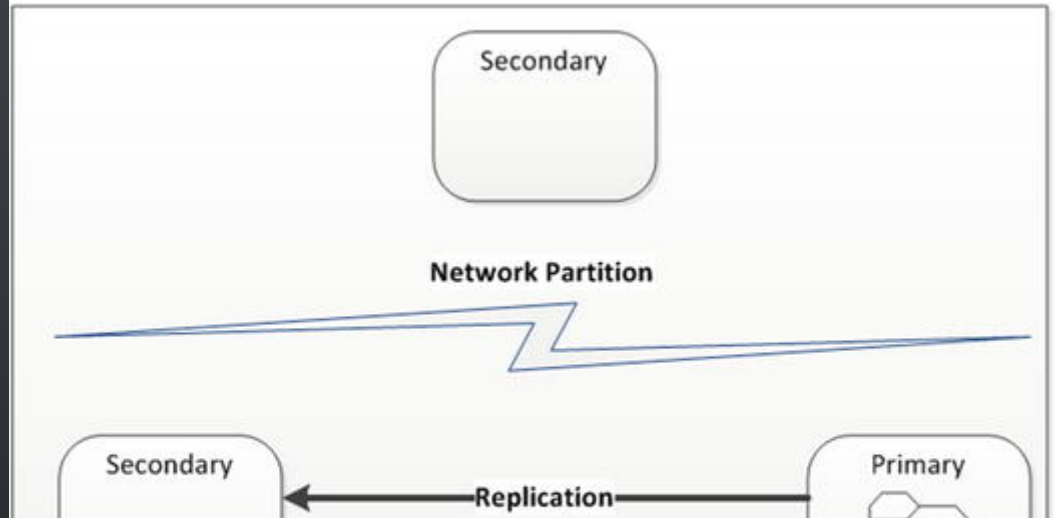




Orders collection sharded by Hashed orderDate

Replication set

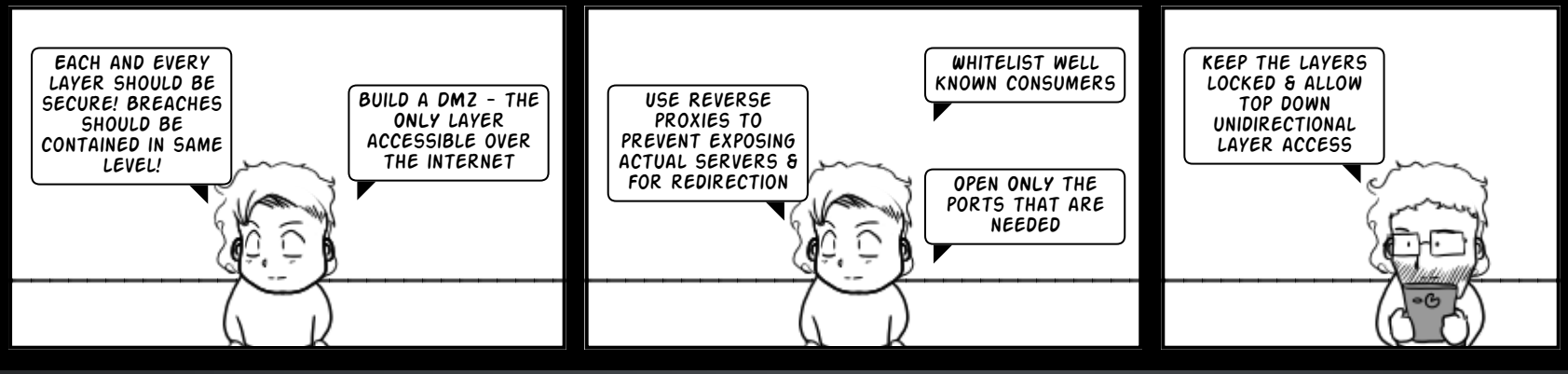




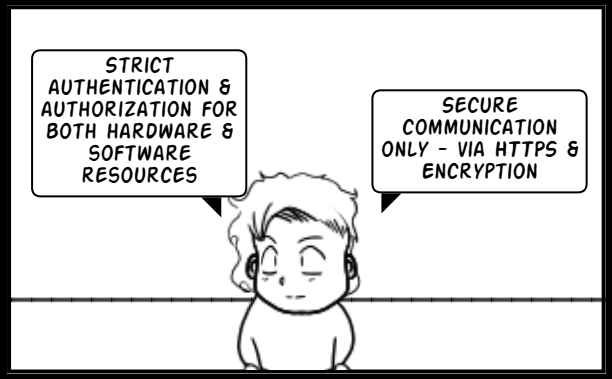
Network partition forces election of new primary

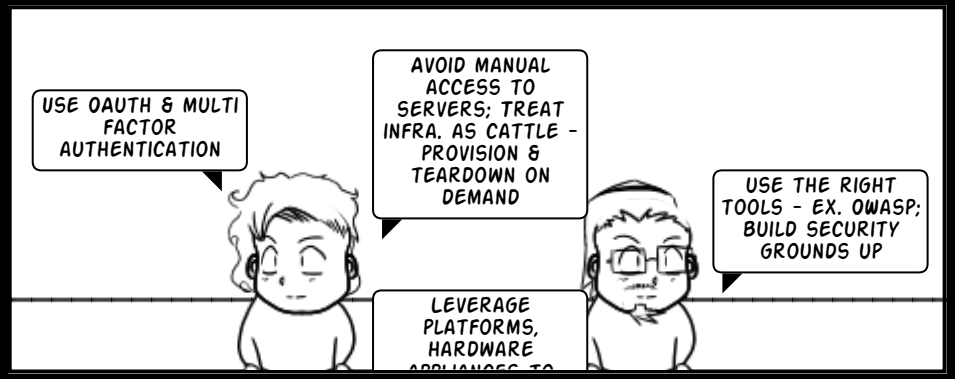
SECURING THE SYSTEM

SECURITY ACROSS LAYERS



INTERNAL & EXTERNAL SECURITY

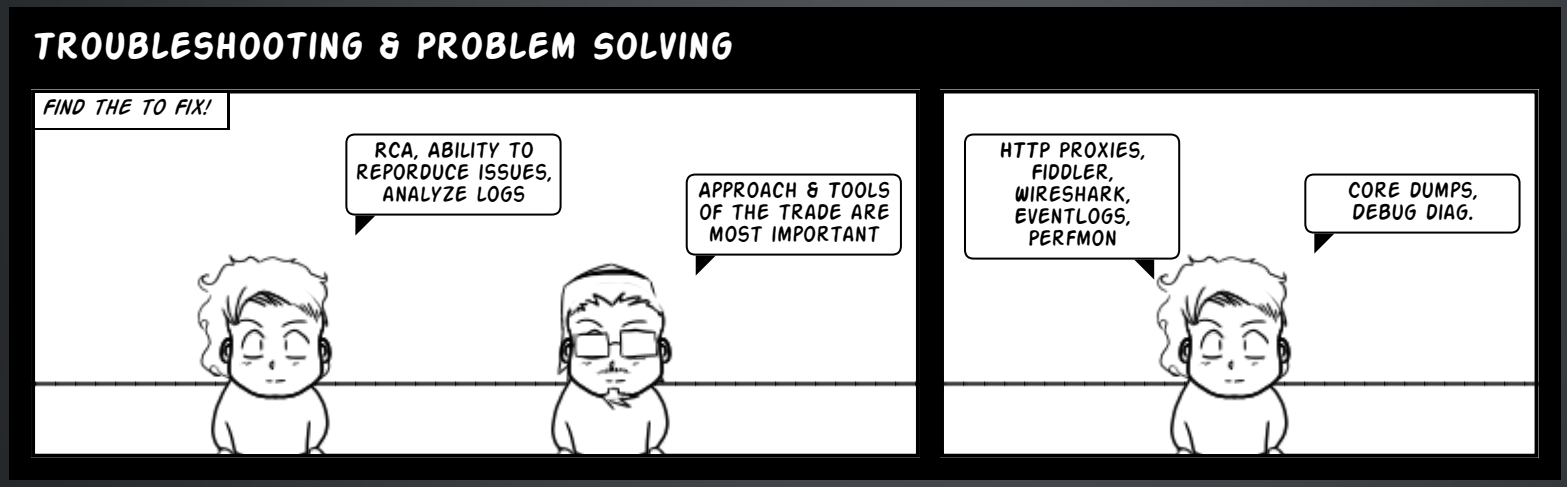




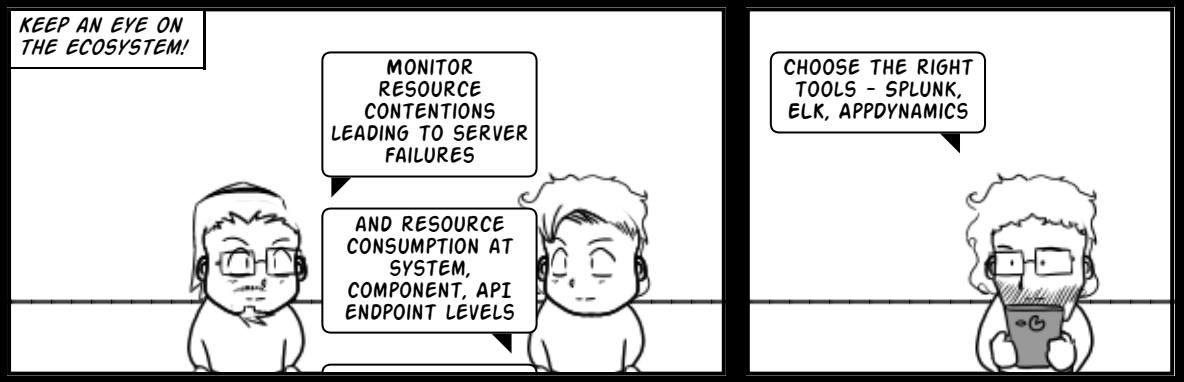
https://www.microsoft.com/en-us/download/confirmation.aspx?id=1330
http://azuresecurity.codeplex.com/

https://msdn.microsoft.com/en-us/library/ms998530.aspx
https://www.microsoft.com/en-us/download/confirmation.aspx?id=11711
https://msdn.microsoft.com/en-us/library/ff921345.aspx
https://msdn.microsoft.com/en-gb/library/ff648138.aspx
http://dataguidance.codeplex.com/releases

OPERATIONAL NEEDS



MONITORING & ALERTING



Links

Donne Martin's System Design Primer

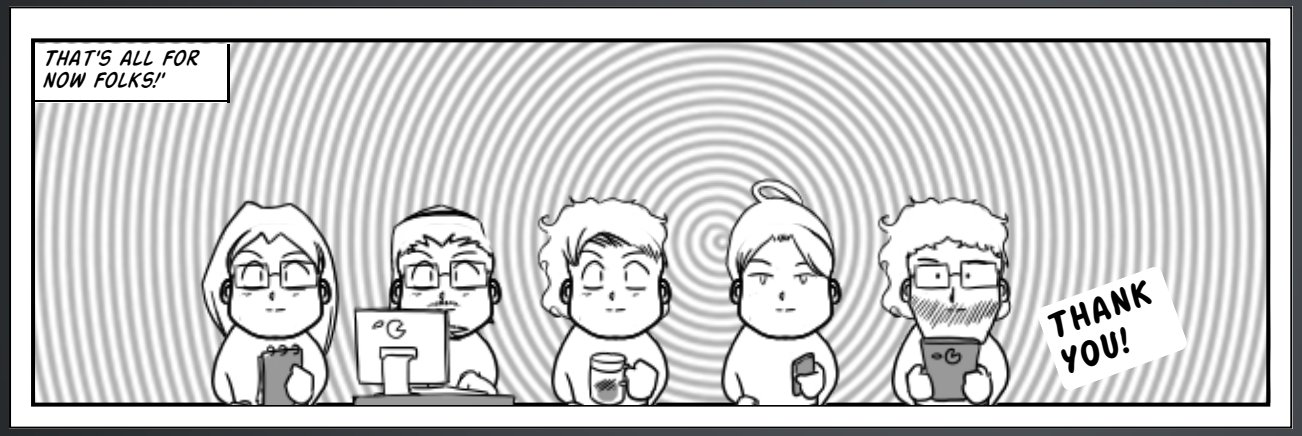
Single absolute reference : an organized collection of resources for system design

There's much much more, but these links give you focus

* http://blog.cleancoder.com/uncle-bob/2014/10/01/CleanMicroserviceArch
* http://microservices.io/
* http://www.codingthearchitecture.com/
* https://awesome-tech.readthedocs.io/

http://alistair.cockburn.us/Hexagonal+architecture
http://alistair.cockburn.us/Foundations+for+Software+Engineering
http://www.idesign.net/
http://www.bredemeyer.com/
https://github.com/checkcheckzz/system-design-interview
https://github.com/benas/awesome-software-craftsmanship
https://github.com/onurakpolat/awesome-bigdata

Books:
https://www.manning.com/books/microservice-patterns
https://github.com/miguellgt/books


THAT'S ALL FOR NOW FOLKS!'
THANK YOU!


ARe We THeRe YeT?
A journey of a thousand miles
begins with a single step
... And lots of coffee!
Visit Jim Hunt at facebook.com/huntcartoons