Escalabilidad

- IIC2173 - Arquitectura de sistemas de software
- Departamento de Ciencia de la Computación
- Escuela de Ingeniería – PUC
- Nicolás Acosta – Gerardo Crot



Sobre la escalabilidad

- Capacidad del software de adaptarse para alcanzar nuevos requisitos de tamaño y alcance manteniendo un performance objetivo

- Permite dar servicio a multitud de usuarios con una experiencia suave mientras se eleva la demanda



¿Por qué escalamos las aplicaciones?

- Alta demanda
- Userbase y MAUs
- Features demandantes
- Balance de costo y performance
  - Optimización de costo



Requerimientos para la escalabilidad

- ¿Qué replicar? ¿Qué minimizar? ¿Qué optimizar? ¿Cómo medir?

"Escalar es como reemplazar todos los componentes de un auto mientras se conduce a 100mph"

Scaling Instagram, Mike Krieger

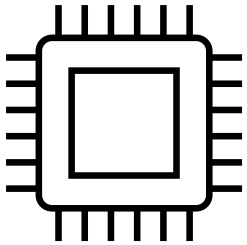
- 25K sign ups día 1
- Peak de carga (404 Django), ¿culpable?



favicon.ico

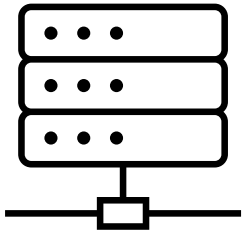
https://techcrunch.com/2012/04/12/how-to-scale-a-1-billion-startup-a-guide-from-instagram-co-founder-mike-krieger/

Requerimientos para la escalabilidad



Heterogeneidad

Capacidad del software de estar formado por múltiples partes o funciones en múltiples y dispares ambientes computacionales



Portabilidad

Capacidad del software para ejecutarse en múltiples plataformas (HW/SW) con modificaciones mínimas y sin degradación de sus características funcionales y no funcionales

Métricas clave en la escalabilidad

- Tiempos de respuesta
  - ms, jitter
  - Inter-redes
- Throughput
  - Req/s
  - Req/instancia/s
- Monitoreo

Estas métricas se deben medir porque no necesariamente escalan linealmente



Proveedores de monitoreo

- Proveedores PaaS y manejados
  - Experiencia sumamente manejada
  - Almacenamiento de datos
- Open Source
  - Prometheus
  - Grafana
  - Nagios
- Ligado a Cloud
  - AWS Cloudwatch













Sobre las métricas

- El “stack” es complejo
- Las herramientas de medición no solo no miden lo mismo, sino que pueden introducir retardos
- La optimización del código, en general, no es significativa



Métodos de escalabilidad

Método	Descripción
Dimensionales	Añadir más recursos, como servidores, discos duros, memoria, etc., para que un sistema pueda manejar cargas de trabajo más grandes y complejas sin reducir su rendimiento y disponibilidad
Modelación de sistema	Diseñar sistemas con una arquitectura modular y flexible que pueda adaptarse fácilmente a los cambios en la demanda y el crecimiento del sistema. Esto implica el uso de patrones de diseño y prácticas de desarrollo de software que permitan una fácil expansión y mantenimiento.
Estructuras y bases de datos	Diseñar bases de datos y estructuras de almacenamiento de datos que manejen grandes cantidades de información y aumentar su capacidad de manera eficiente, mediante el uso de técnicas como la partición de datos, la replicación, la asignación dinámica de recursos, la gestión de índices y la optimización de consultas.

Métodos dimensionales



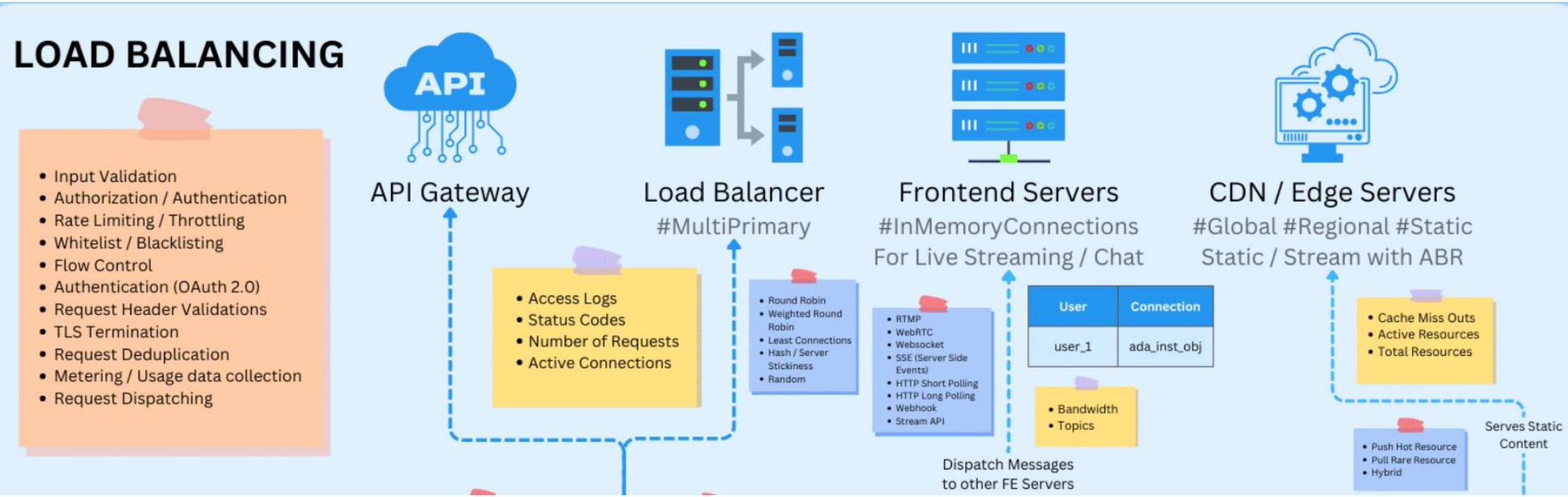
Métodos dimensionales

- Vertical
  - Mejorar características de computadores existentes
- Horizontal
  - Añadir más instancias de computación
    - Máquinas físicas
    - Máquinas virtuales
    - Peers
  - Mejorar capacidades de red
    - Ancho y latencia
    - Ethernet 10GbE , InfiniBand (hasta 600Gb/s), Myrinet

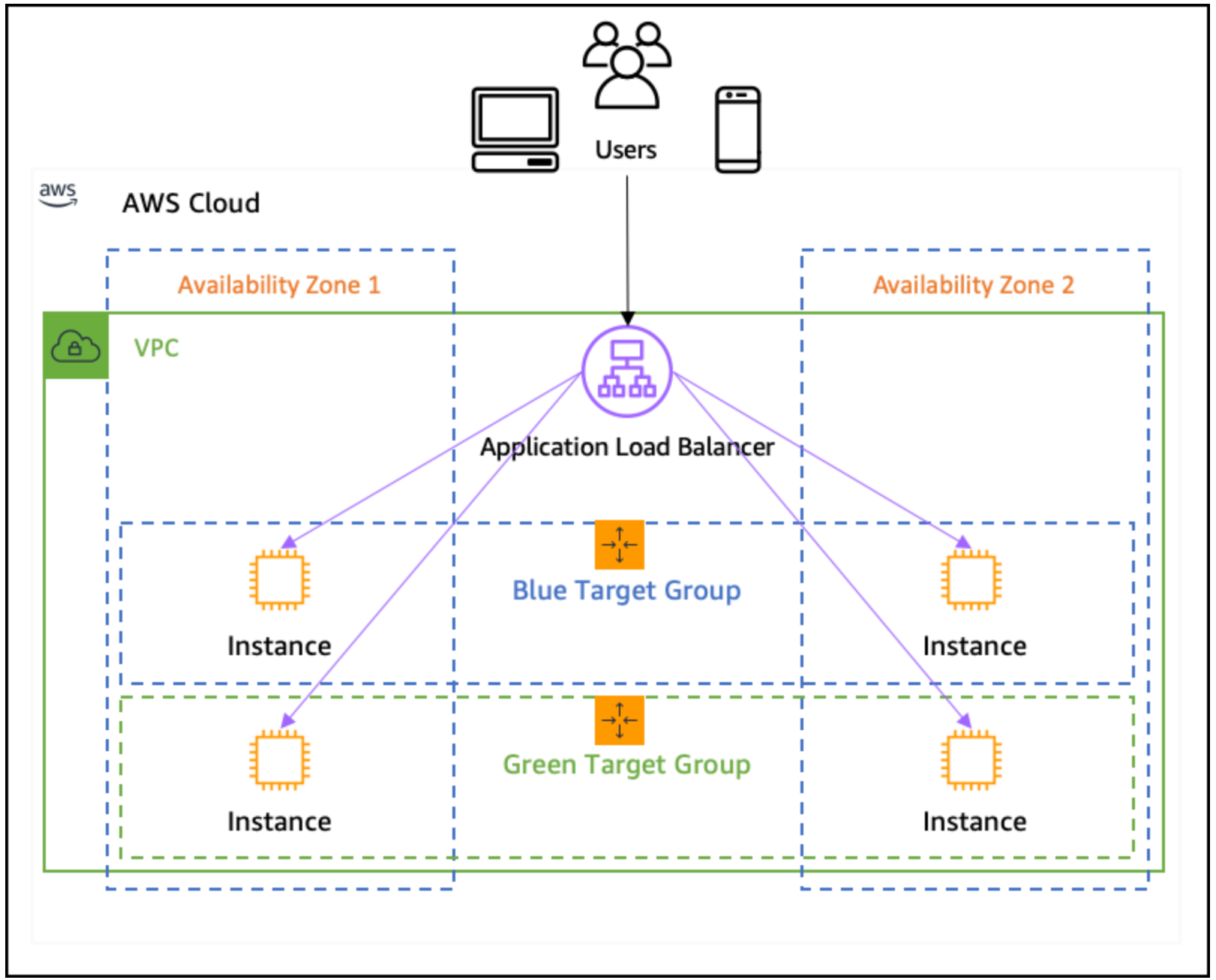


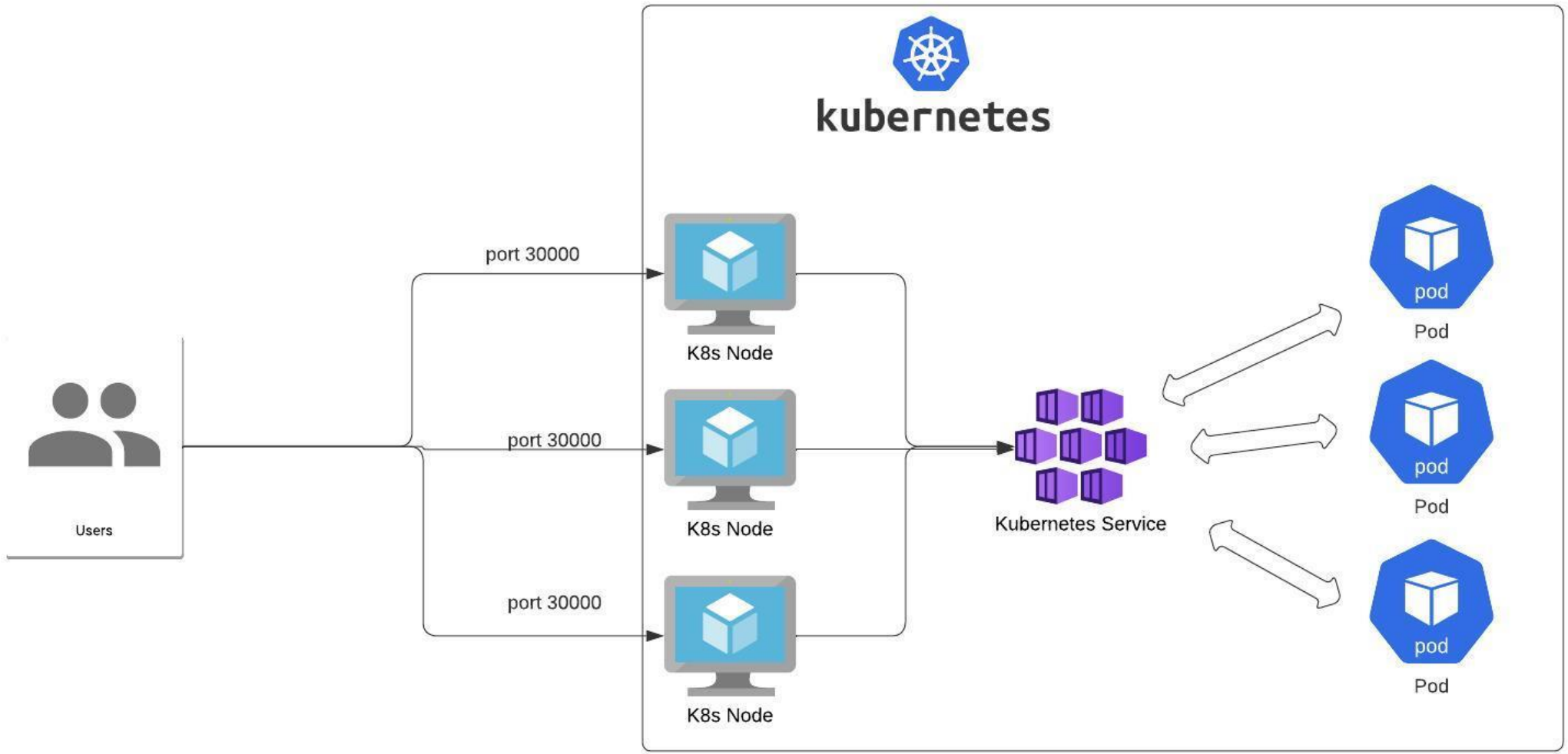
Tamaño de la instancia	CPU virtual	Memoria (GiB)	Almacenamiento de instancias (GB)	Ancho de banda de red (Gbps)	Banda ancha de EBS (Gbps)
m6i.large	2	8	Solo EBS	Hasta 12,5	Hasta 10
m6i.xlarge	4	16	Solo EBS	Hasta 12,5	Hasta 10
m6i.2xlarge	8	32	Solo EBS	Hasta 12,5	Hasta 10
m6i.4xlarge	16	64	Solo EBS	Hasta 12,5	Hasta 10
m6i.8xlarge	32	128	Solo EBS	12,5	10
m6i.12xlarge	48	192	Solo EBS	18.75	15

LOAD BALANCING



https://zonito.gumroad.com/l/systemdesignblueprint


Users
aws
AWS Cloud
Availability Zone 1
Availability Zone 2
VPC
Application Load Balancer
Blue Target Group
Instance
Instance
Green Target Group
Instance
Instance


kubernetes
Users
port 30000
port 30000
port 30000
K8s Node
K8s Node
K8s Node
Kubernetes Service
pod
Pod
pod
Pod
pod
Pod

Métodos dimensionales: consideraciones

- Vertical
  - No siempre añade performance
- Horizontal
  - Requiere intermediarios de coordinación
    - Load Balancers
    - NATs, Redes
  - Añade complejidad en sistemas
    - Asincronía, concurrencia, paralelismo
  - Añade latencia por distribución
    - En SOA y uServicios
  - Ley de retornos decrecientes

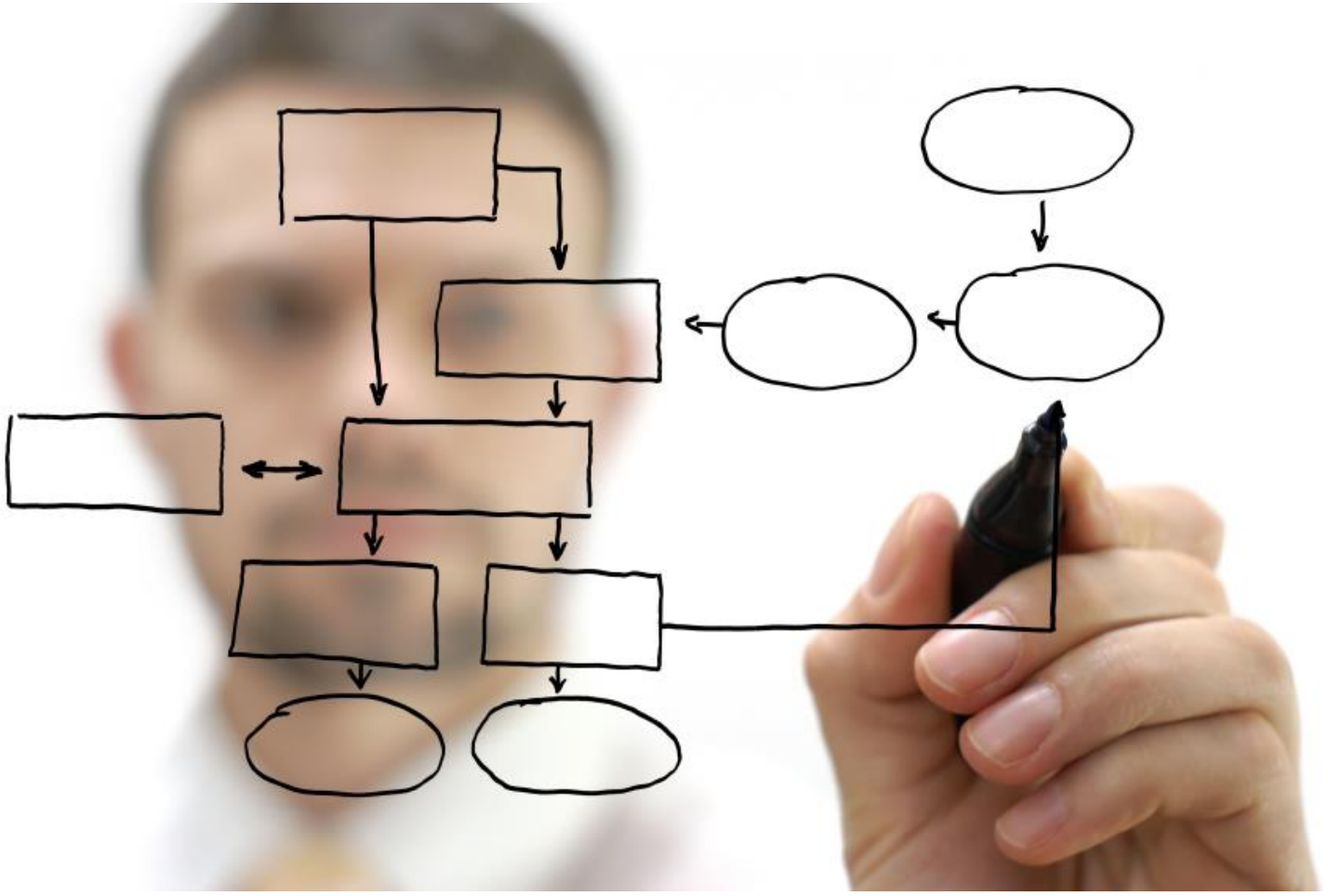


Escalabilidad II

- IIC2173 - Arquitectura de sistemas de software
- Departamento de Ciencia de la Computación
- Escuela de Ingeniería – PUC
- Nicolás Acosta – Gerardo Crot



Métodos en modelación de sistemas



Métodos en modelación de sistemas

- Basados en recomendaciones y buenas prácticas
- Modelación de componentes
  - Dar a cada componente un propósito claro y bien definido
  - Dar a cada componente una interfaz simple y entendible
  - No recargar a componentes con responsabilidades de interacción
  - Evitar heterogeneidad innecesaria
    - Sobrecarga comunicación
  - Fuentes de datos distribuidas
  - Replicar data cuando es necesario



MMS: Conectores

- Modelación de conectores
  - Usar conectores explícitos
  - Dar a cada conector una responsabilidad claramente definida
  - Elegir el conector más simple para las tareas
  - Diferenciar entre dependencias directas e indirectas (y asíncronas)
  - No poner funcionalidad de la aplicación en los conectores
  - Aprovechar conectores explícitos para escalabilidad de datos
    - Caché, hoarding, pre-fetching, buffering



MMS: Configuraciones

- Configuraciones:
  - Evitar cuellos de botella
    - Añadir componentes: Réplica, Balance de carga
  - Usar capacidades de procesamiento paralelo
    - Si el problema es naturalmente paralelizable -> Escala con eficiencia
  - Ubicar las fuentes de datos cerca de los consumidores de datos
    - Caching, prefetching, réplica
    - A nivel físico



MMS: Configuraciones

- Configuraciones:
  - Hacer distribución transparente
    - Réplicas, facilidad de deployment
  - Usar los estilos apropiadamente
    - Pub/Sub, Event based (+)
    - Intérprete, Pipe&Filter (componentes)
  - Comunicación asíncrona

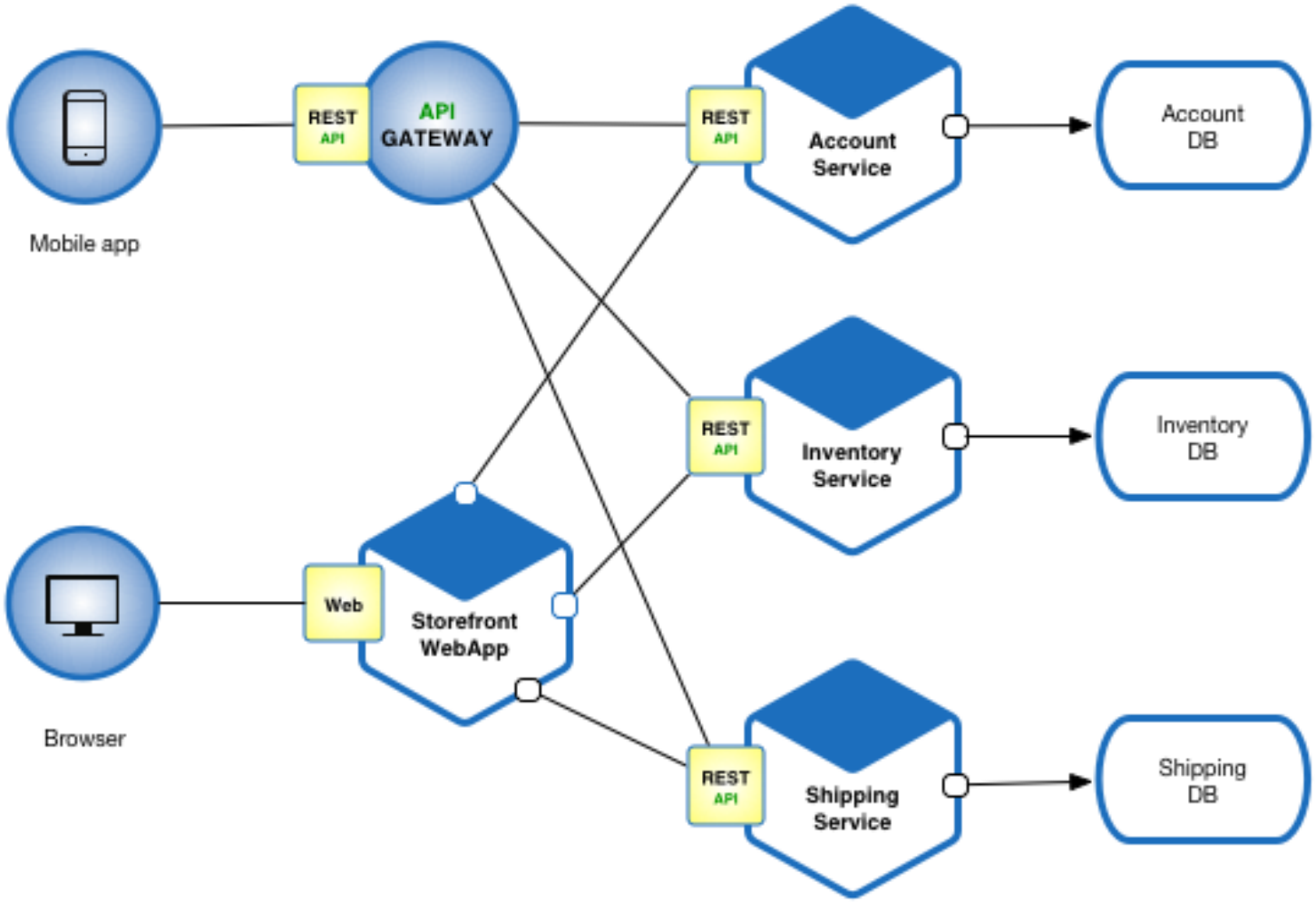


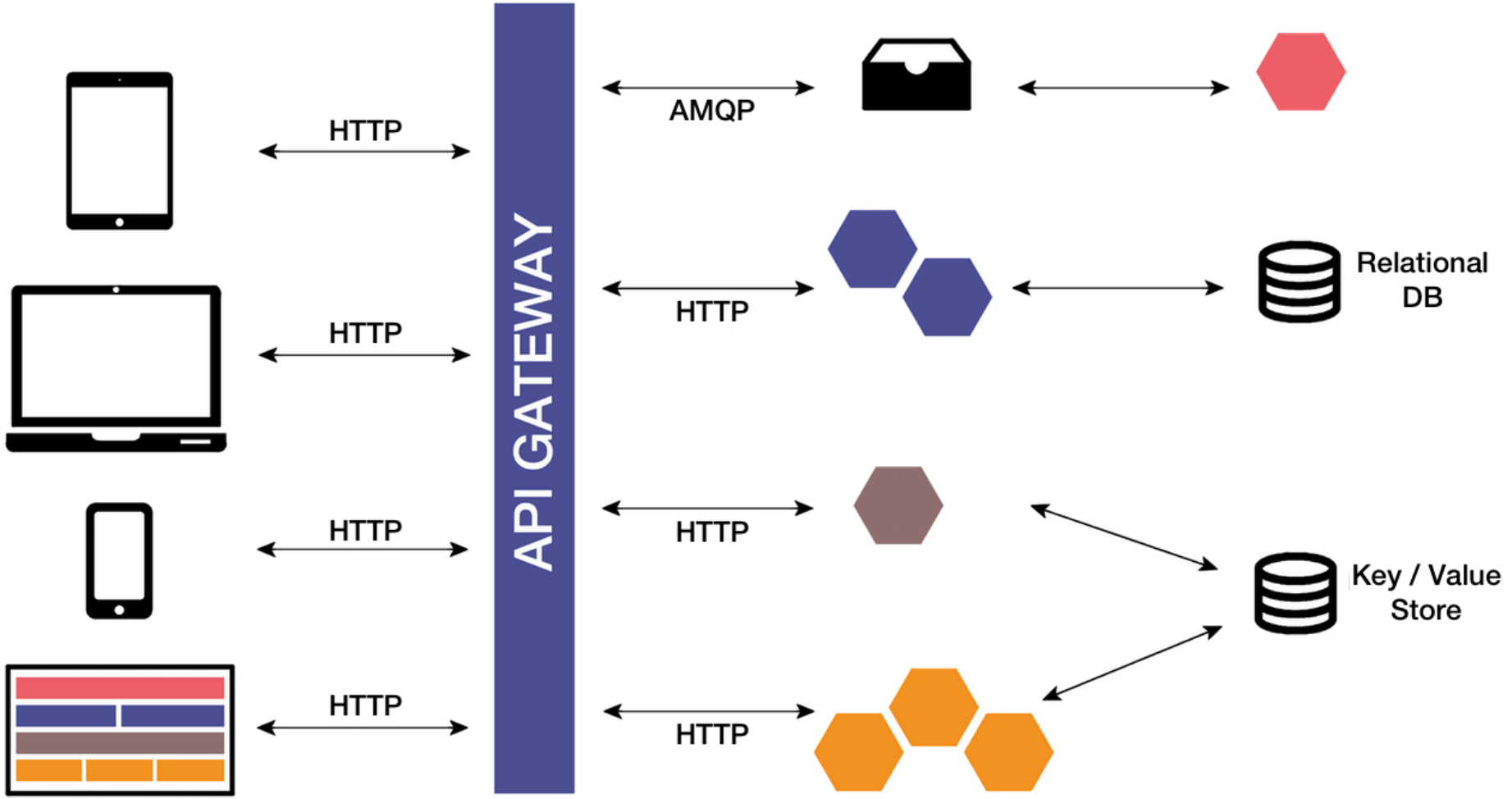
MMS: Caso Instagram

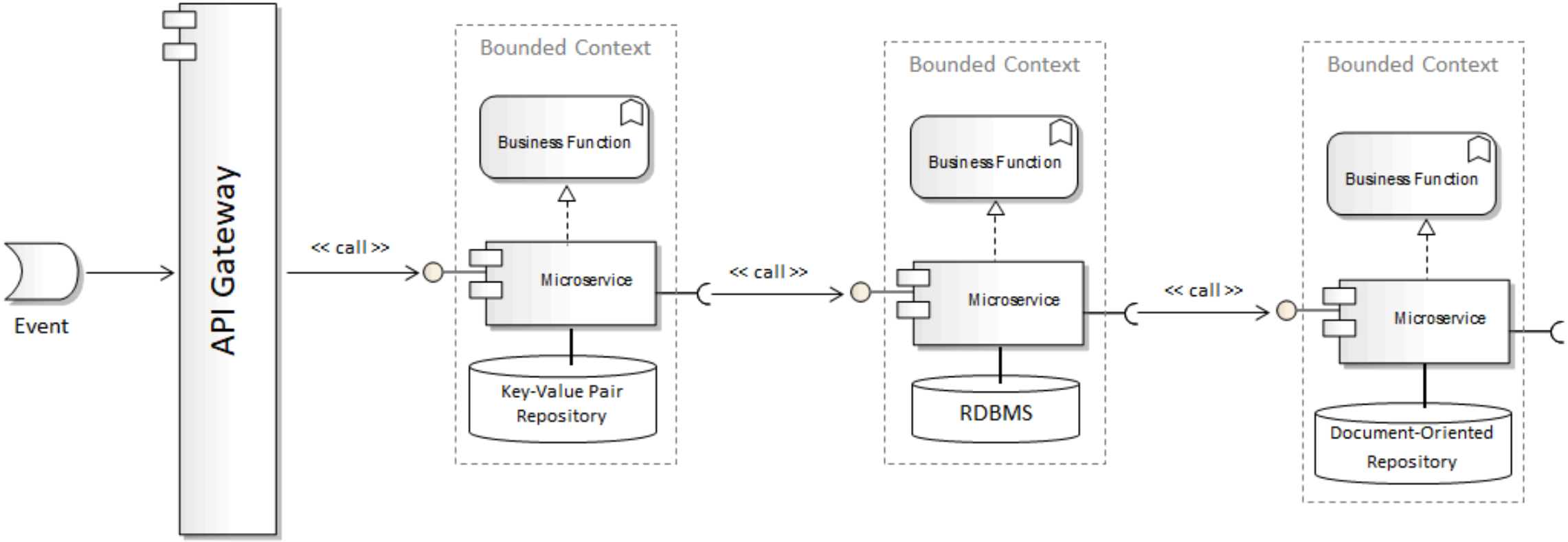
1. Pruebas unitarias y funcionales extensivas
2. Mantén el sistema DRY (Don’t Repeat Yourself)
3. Acoplamiento débil (notificaciones/señales)
4. Gran parte del código en Python (C sólo si hace falta)
   - Promueve especialización y homogeneidad
5. Revisiones de código frecuentes
6. Mucho monitoreo (munin, statsd, pagerduty, …)
7. ¡NO reinventes la rueda!
8. No sobre optimices, ni adivines donde vendrá el problema de escalabilidad

MMS: Distribución

- Sistemas distribuidos
  - División por dominio
  - Ej:
    - SoA, Microservicios
- Consideraciones
  - Complejidad por coordinación
  - Latencia e implementación de intermediarios
  - El glorioso monolito




HTTP
HTTP
HTTP
HTTP
API GATEWAY
AMQP
HTTP
HTTP
HTTP
Relational
DB
Key / Value
Store


Event
API Gateway
<< call >>
Bounded Context
Business Function
Microservice
Key-Value Pair Repository
<< call >>
Bounded Context
Business Function
Microservice
RDBMS
<< call >>
Bounded Context
Business Function
Microservice
Document-Oriented Repository

MMS: Serverless

- Sistemas serverless
  - Útil para PoCs y scripts puntuales
- Servicios (en AWS):
  - Computing: Lambda, Fargate
  - Storage: S3
  - DB: RDS
  - Cord: SNS, SQS
  - Data: Aurora, Dynamo
- Pero hay que tener ojo
  - Recordar caso de Amazon Prime y su distribución Serverless



MMS: Frontend

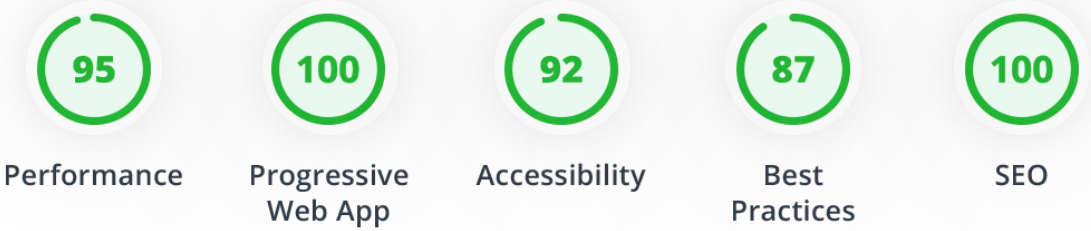
- http://developer.yahoo.com/performance/rules.html
  - Evitar imágenes, usar sprites, combinar archivos (scripts, imágenes), zipear archivos, usar caches de servidor, manejar caches, E-Tags
  - Stylesheets al principio
  - Scripts al final
  - No usar CSS expressions
  - Javascript y CSS deben ser externos
  - Hacer Ajax cacheable
  - Minimizar los Iframes, cookies…
- Uso inteligente de preloads
- Uso de HTTP/2.0-3.0

MMS: Frontend

- http://modernizr.com/
  - Javascript que detecta las capacidades de HTML5 y CSS3 del browser y según eso adapta/degrada la página
- http://html5boilerplate.com/
  - Plantilla de la capa front-end de usuario optimizada y configurable (directorios, archivos por defecto, drafts, etc.)
- Lighthouse
  - Análisis de rendimiento y carga
    - Mejorar la carga y detectar data innecesaria mejora la escalabilidad



Google
Lighthouse



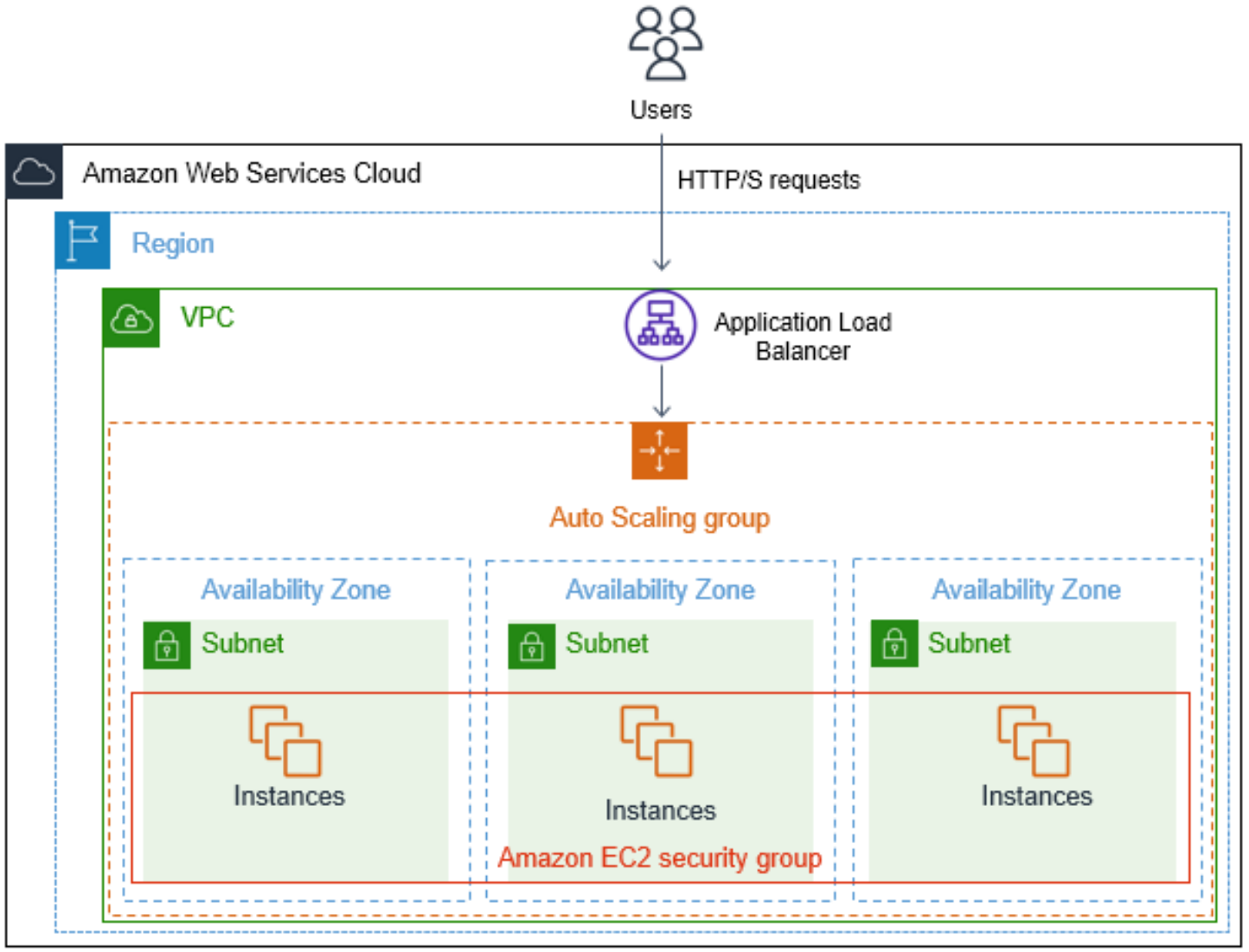
MMS: Frontend

- Web core vitals
- Preload – prefetch – priorities
- Chunking – tree shaking

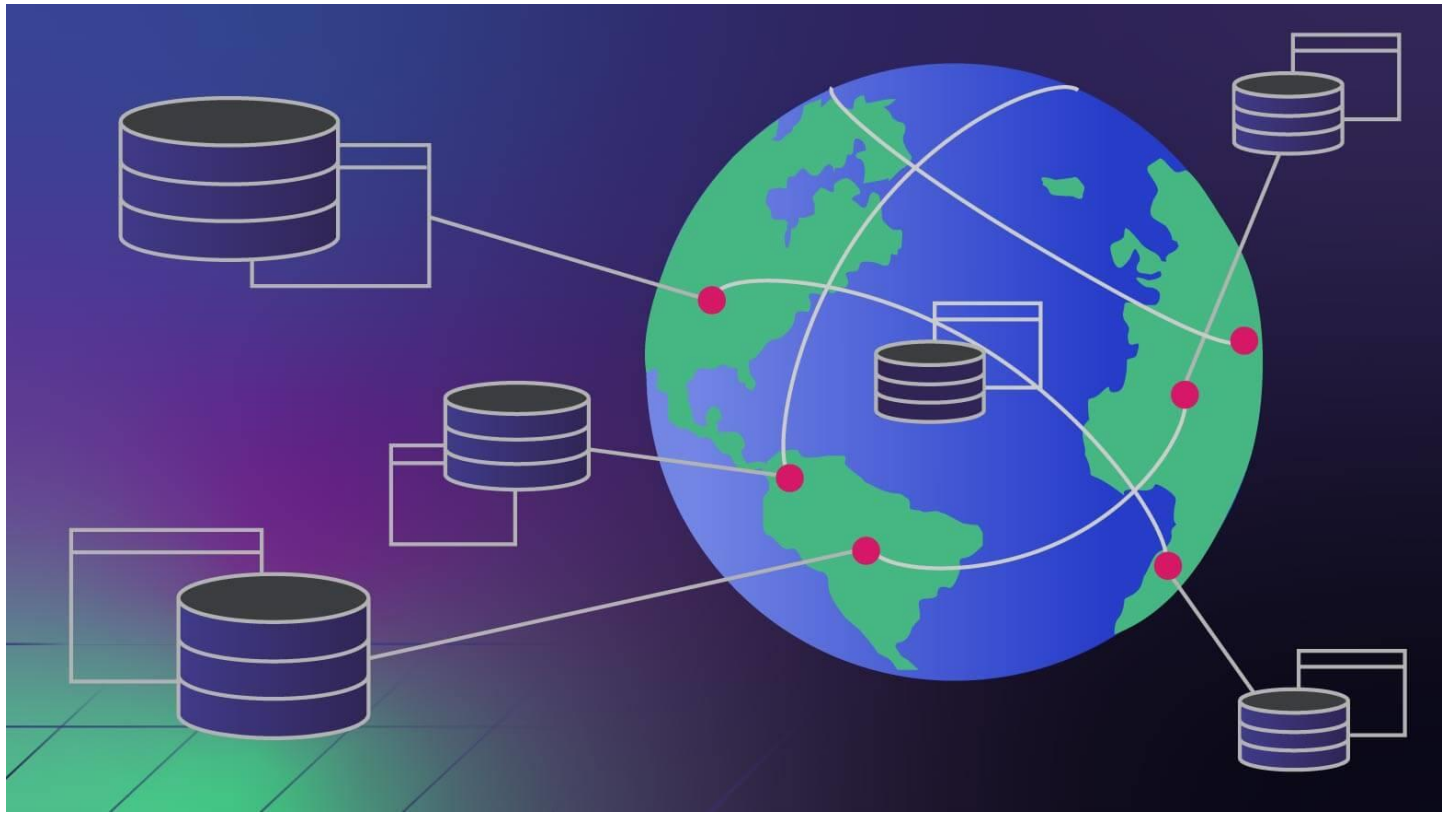


MMS: Cloud

- Elasticidad de alto nivel
- OPEX proporciona escalabilidad financiera
- Sistemas de autoescalabilidad
- Consideraciones
  - Es una bala de oro (¿quieres disparar oro?)
  - Vendor lock-in
  - Descontrol sobre la infra subyacente



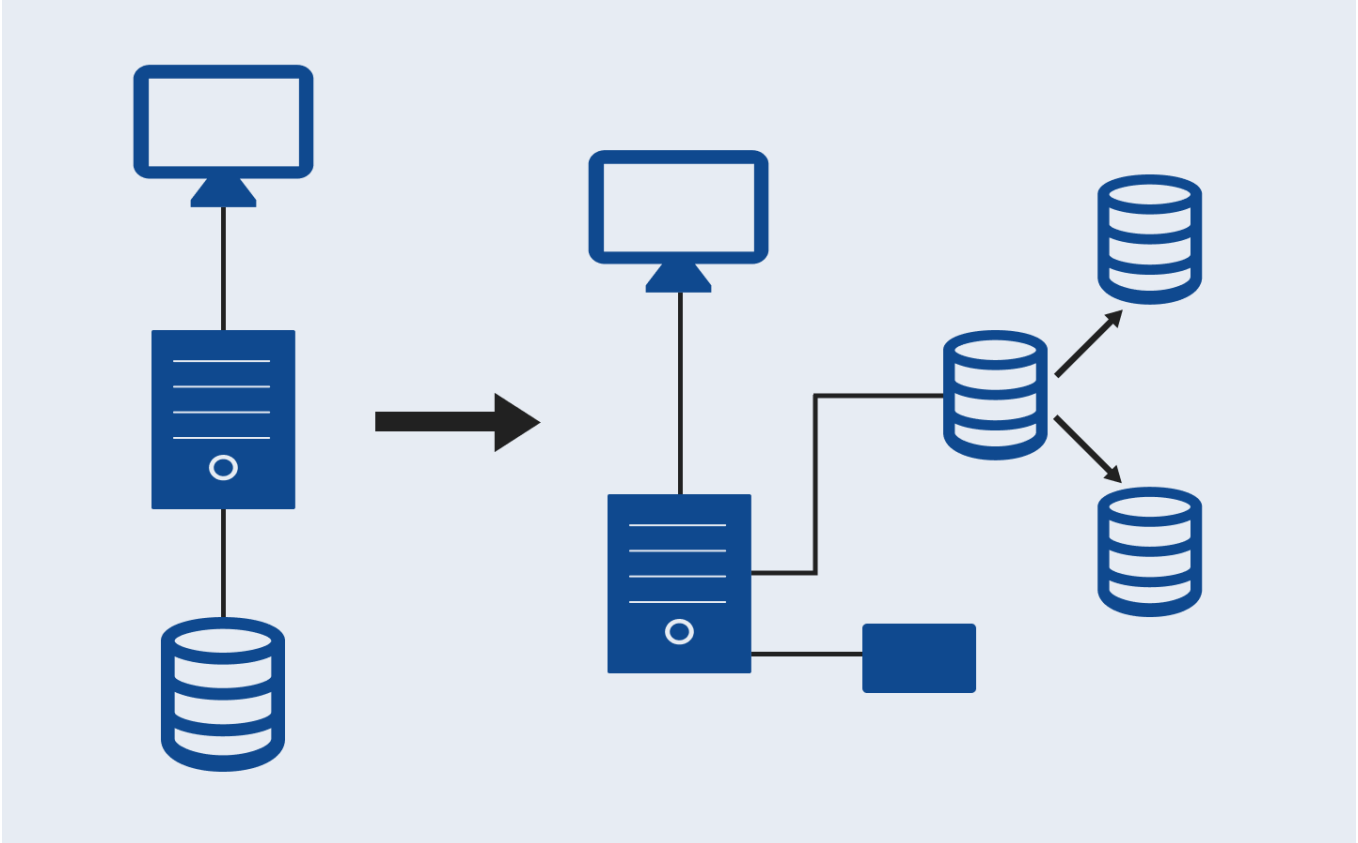
Métodos en data



Métodos en data

- Principalmente enfocados en paralelizar trabajo
- Mantener performance, consistencia y manejar costos

- Herramientas
  - Modelos alternativos de DBs
  - Distribuciones de datos
  - Not only SQL
  - Modelos de datos avanzados



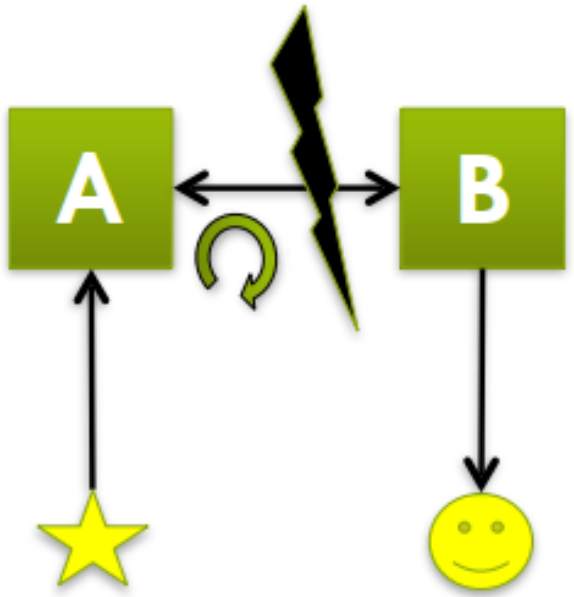
Métodos en data: CAP

Teorema CAP: Consistency, Availability, Partition Tolerant

- Solo podemos tener 2 de los 3

2000, Conjetura de Brewer

2002, Prueba Gilber & Lynch



Availability + Partition tolerant = No Consistency

Consistency + Partition tolerant = No Available

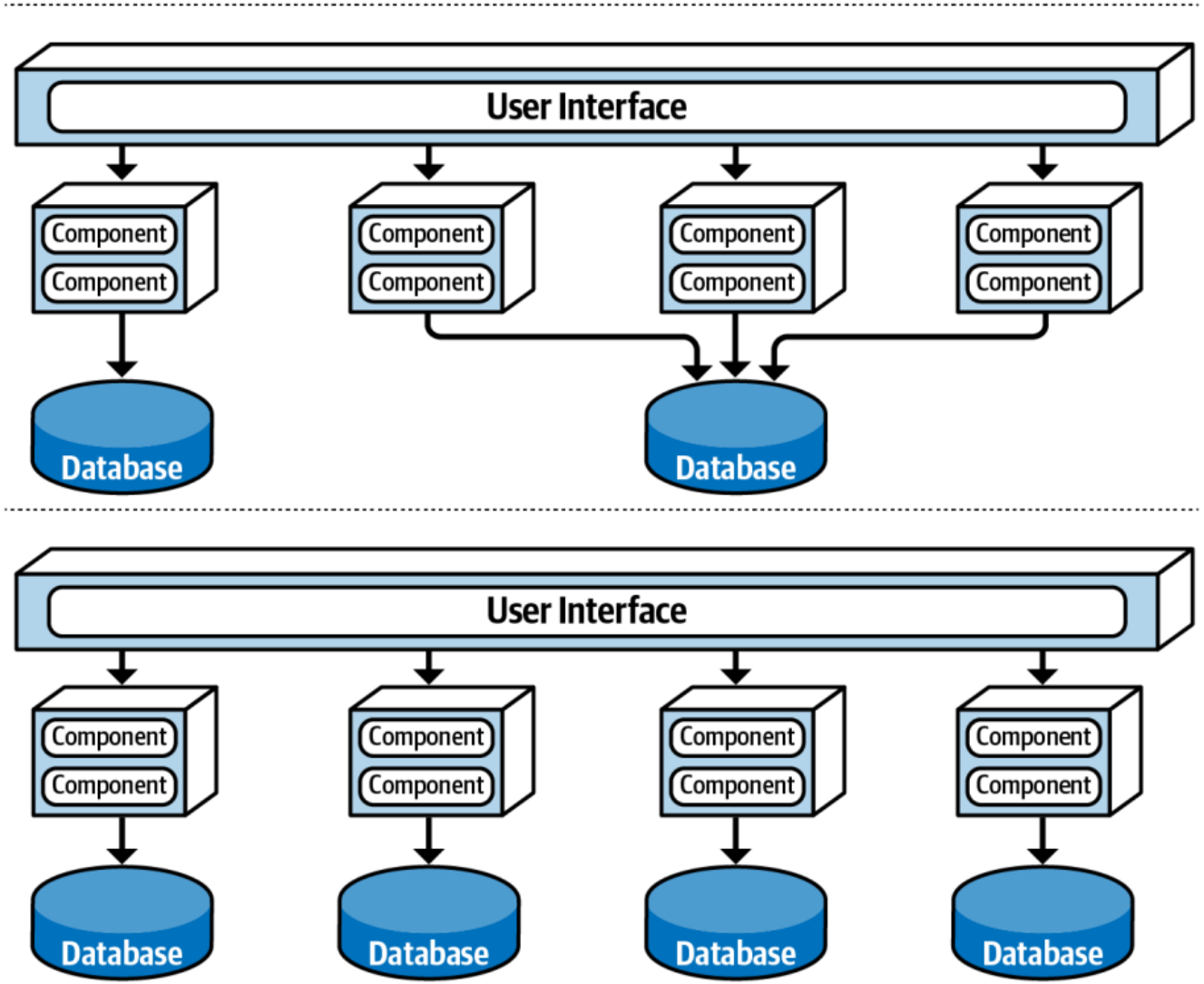
Consistency + Availability = No Partition tolerant

CAP

- Mayo 30, 2012, Articulo de Brewer revisando su teorema
  http://www.infoq.com/articles/cap-twelve-years-later-how-the-rules-have-changed
- No considera latencia
- No considera mecanismos actuales para gestionar
- Particiones
- No considera compensación
- Concepto de BASE (Basically Available, Soft state, Eventually consistent)
  - Soft state: cache

Data en arquitecturas distribuidas

- EDA, SoA y Microservicios intencionan uso de datos por servicio
- Pros
  - Ayuda con separación de responsabilidades
  - Permite escalar cada fuente de datos según necesidad
- Cons
  - Añade complejidad
  - Puede reducir el performance si se cruza mucha data



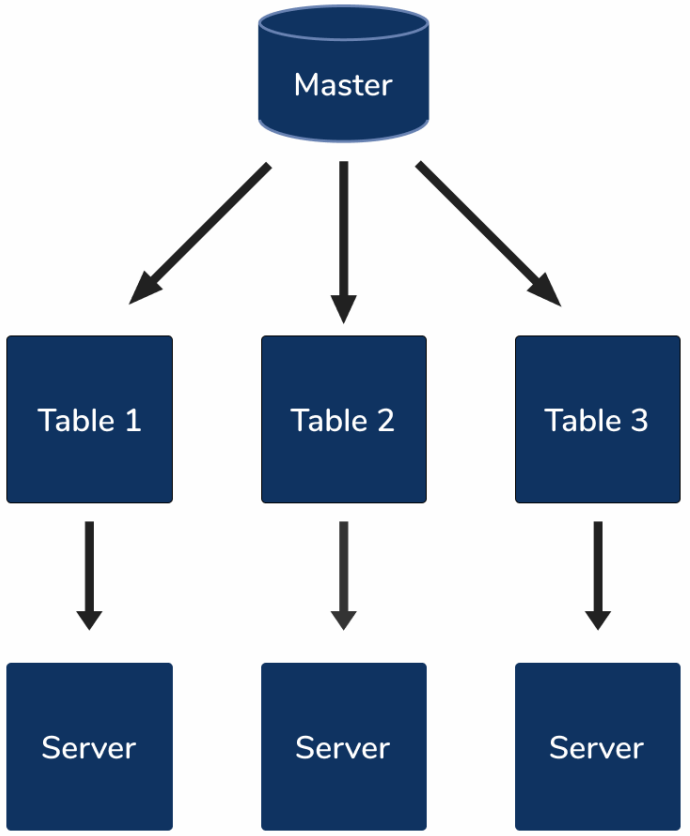


Escalabilidad III

- IIC2173 - Arquitectura de sistemas de software
- Departamento de Ciencia de la Computación
- Escuela de Ingeniería – PUC
- Nicolás Acosta – Gerardo Crot

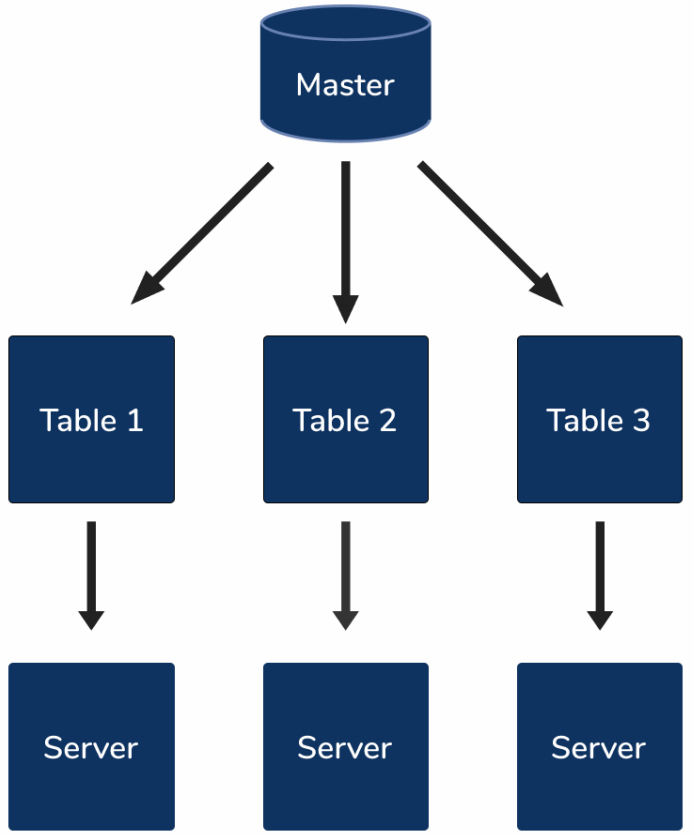
Sharding

- Particionar los datos de manera que tenga sentido para una aplicación
  - Ej. Por índice, ID, por ubicación geográfica, únicamente algunas tablas, etc.
  - Partición horizontal (shard)
    - Las filas de una DB (pueden ser de varias tablas)
    - Un shard puede distribuirse en varias máquinas
  - Partición vertical (normalización)
    - Grandes tablas con pocas columnas
- App debe saber a qué server preguntar



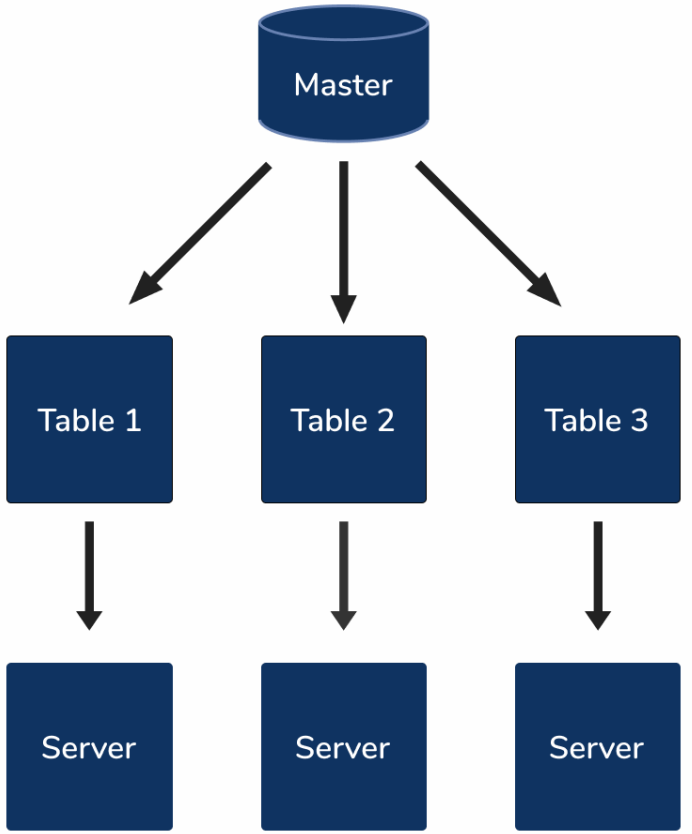
Sharding: Ventajas

- Tablas pequeñas
  - Pocas tuplas
  - Búsquedas más rápidas
- Escalabilidad horizontal: Distribuidas en varias máquinas
  - Tiempo de búsqueda se divide si se coordina
- Es posible aplicar redundancia
- Cuidado: un buen shard no requiere compartir datos



Sharding: Riesgos

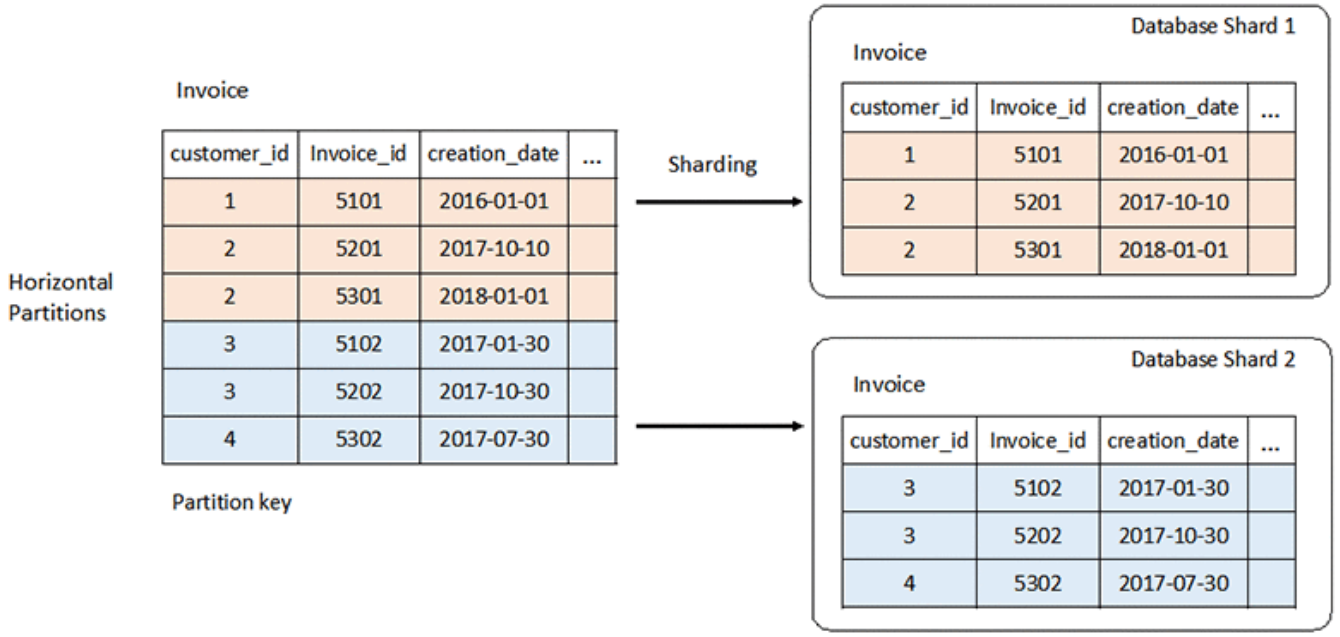
- Complejidad
  - Del SQL
  - Del Software
  - A nivel Operacional
    - Ej: Modificar el schema
- Punto único de falla
  - Un shard corrupto podría destruir todo el sistema
- Failover más complejo
  - Hash de los shards
- Backup complejo
- Rebalanceo de datos en shards
- ¡¡Writing!! (retardo en réplicas lentas)
- "Cross-shard joins"



http://codefutures.com/database-sharding/

Partitioning

- Otra forma de separar la información es por algún otro parámetro
  - Tiempo, valor de parámetros, etc.
- Pros
  - Índices más eficientes
  - Menores tamaños de tablas, mejores desempeños
    - Solo si se diseña correctamente
  - Puede combinarse con sharding
- Cons
  - Consultas más complejas



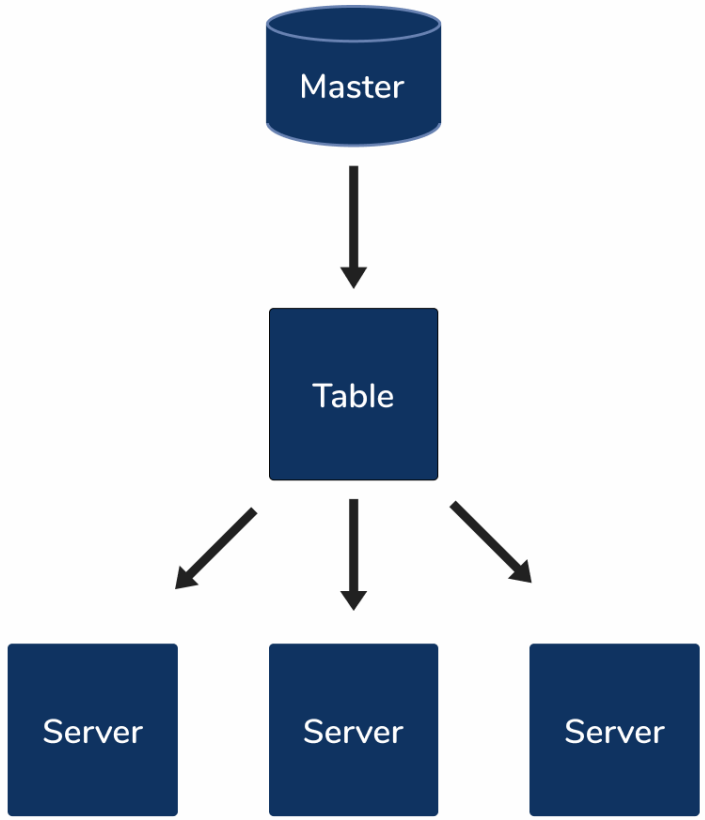
https://aws.amazon.com/es/blogs/database/sharding-with-amazon-relational-database-service/

Sharding vs partitioning

Aspecto	Sharding	Partitioning
Alcance	Distribuye datos entre múltiples **bases de datos/nodos**	Divide datos dentro de una **sola base de datos**, aunque puede usarse para implementar sharding
Caso de uso	Usado para **escalar horizontalmente** (bases de datos distribuidas)	Usado para **mejorar el rendimiento** en una sola base de datos
Gestión	La aplicación a menudo coordina qué shard consultar	La base de datos gestiona las particiones automáticamente
Distribución de datos	Basado en una **clave de shard** que determina el shard de los datos	Basado en **reglas de particionamiento** como rangos o valores hash
Tipos	Horizontal (entre instancias/nodos de bases de datos)	Horizontal o vertical (dentro de una sola base de datos)
Complejidad	Requiere lógica y coordinación más complejas en la aplicación	Más sencillo, ya que es gestionado dentro de una sola base de datos
Tolerancia a fallos	Distribuido entre varias máquinas, añadiendo redundancia	Las particiones suelen estar en una sola base de datos, por lo que la tolerancia a fallos depende del propio sistema de bases de datos

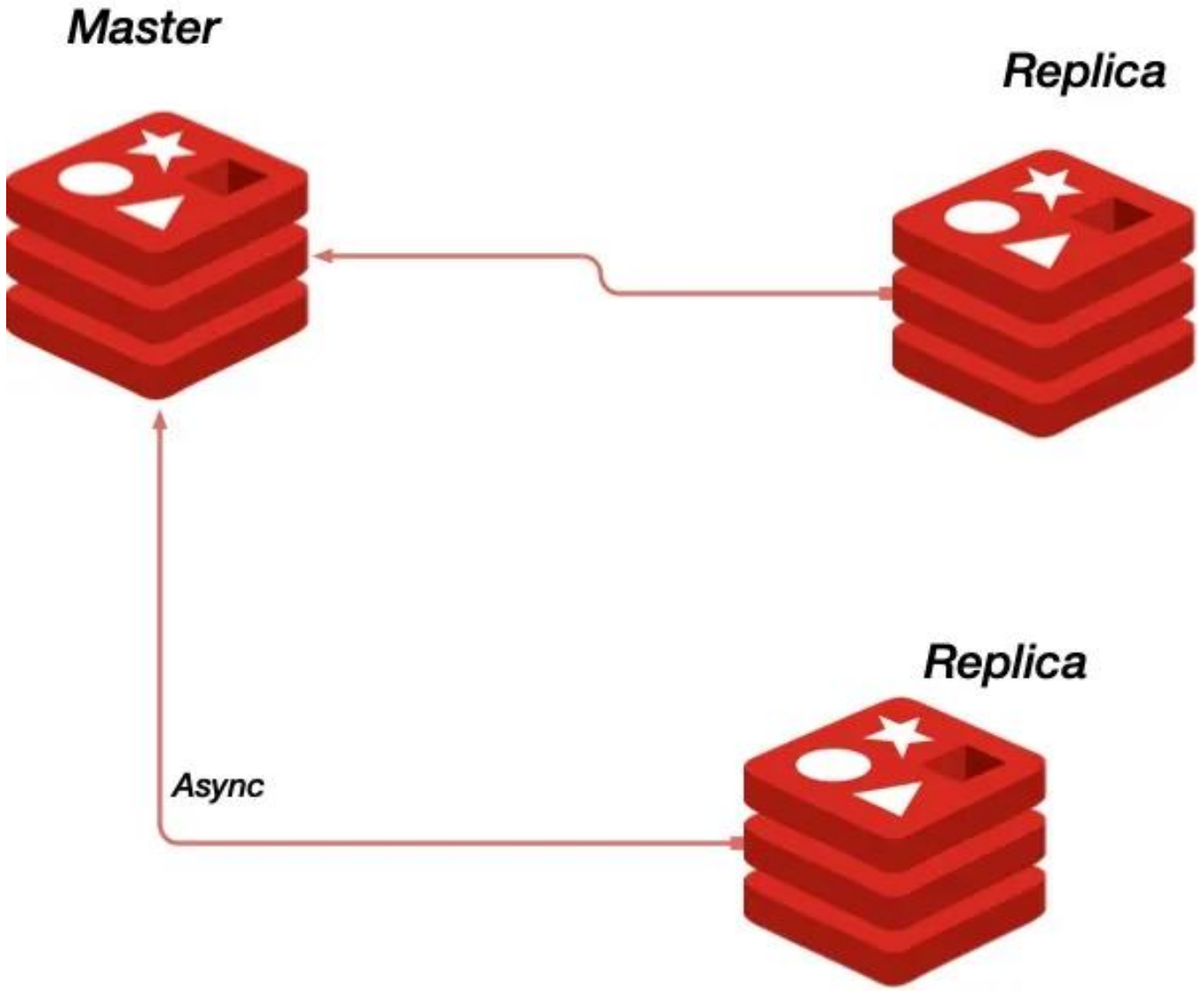
Partitioning

- https://www.postgresql.org/docs/10/static/ddl-partitioning.html
- https://blog.timescale.com/scaling-partitioning-data-postgresql-10-explained-cd48a712a9a1
- https://blog.timescale.com/time-series-data-postgresql-10-vstimescaledb-816ee808bac5



Caché

- Almacenamos una respuesta (que creemos va a ser solicitada con alta frecuencia) para reducir la latencia
- Problemas:
  - *Cache invalidation*
  - Consistencia
  - Cuellos de botella
- "Soluciones":
  - Content Delivery Networks (CDNs cache)
  - Memoization
  - Statelessness
  - buenas prácticas

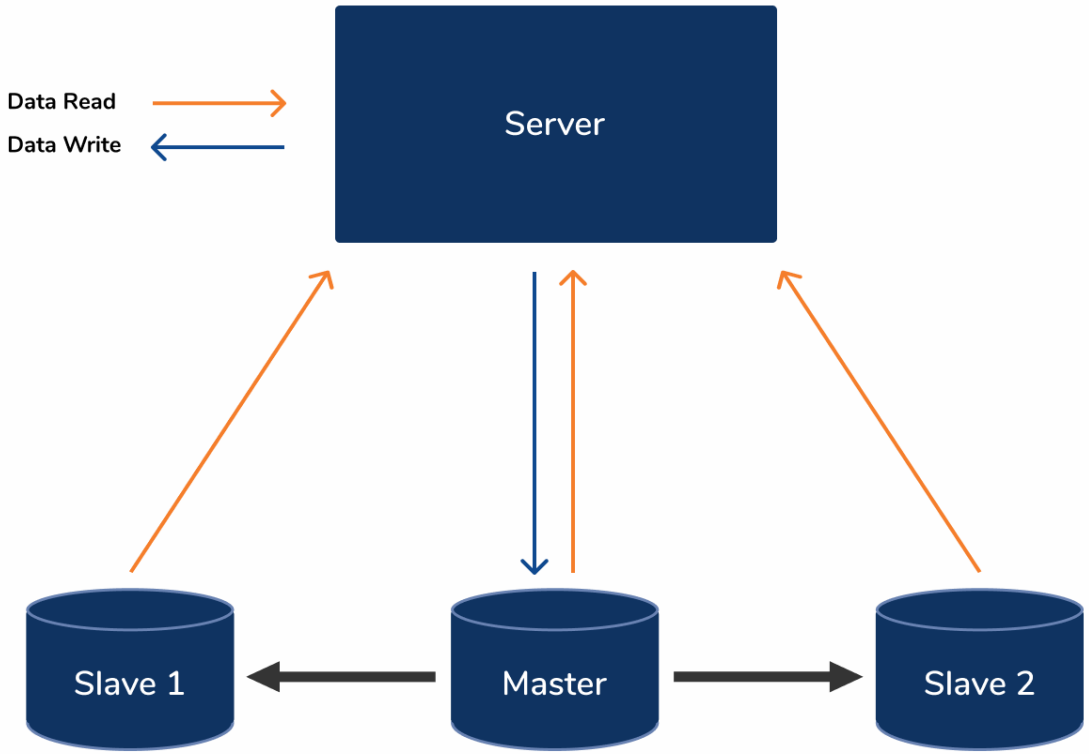


https://redis.io/docs/management/replication/

Master-slave aka. Leader-follower

- Leader que recibe writes y propaga a followers para reads

- Pros
  - Mejora capacidad de lectura al ofrecer nuevos nodos
  - Mejora redundancia mediante replicación
- Cons
  - Complejidad en coordinación
  - Latencia de escritura
  - Escalabilidad del Master



Costos de los métodos

- Añadir nuevas features se vuelve más complejo
- El sistema se vuelve más complejo
- El código se puede volver menos “testeable”
- Identificar y corregir defectos/bugs se vuelve más complejo

Igual que en performance: Medir

- Latencia
- Espacio
- Redundancia
- Chaos













Ejemplo: Arquitectura de datos de Google

- La tarea: ordenar 1 PB (petabyte) en 2010
  - = 1.000 TB (terabytes)
  - = 1,000,000 GBs

- Ordenar 10 trillones (TB) de registros de 100-bytes
  - 6 horas, 2mins
  - 4,000 computadores
  - Replicado 3 veces en 48,000 discos
- Actualmente: 2.5 EBs por día (1 EB = 1000 TB)

Stack de datos

- Linux
- Python, Java, C++
- +200 GFS clusters
  - Un cluster = 1.000 a 5.000 máquinas
    - Dual core, GB ethernet, 4-8GB RAM
    - Tasa de lectura/escritura 40 GB/segundo
- 6.000 aplicaciones MapReduce
- BigTable almacena billones de URLs, cientos de TB de imagen satelital, preferencias de cientos de millones de usuarios

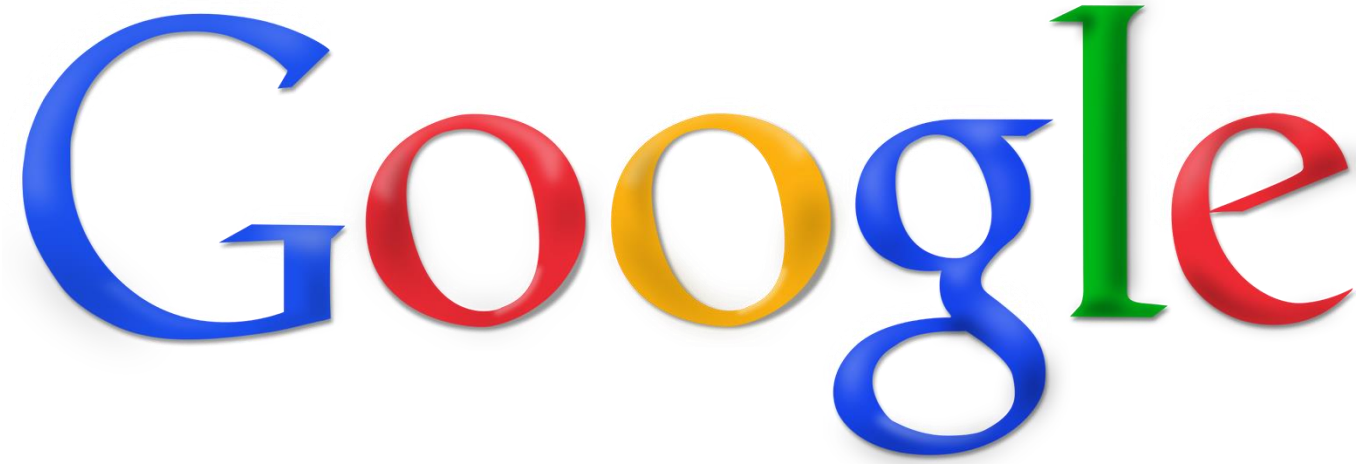
Stack de servicios

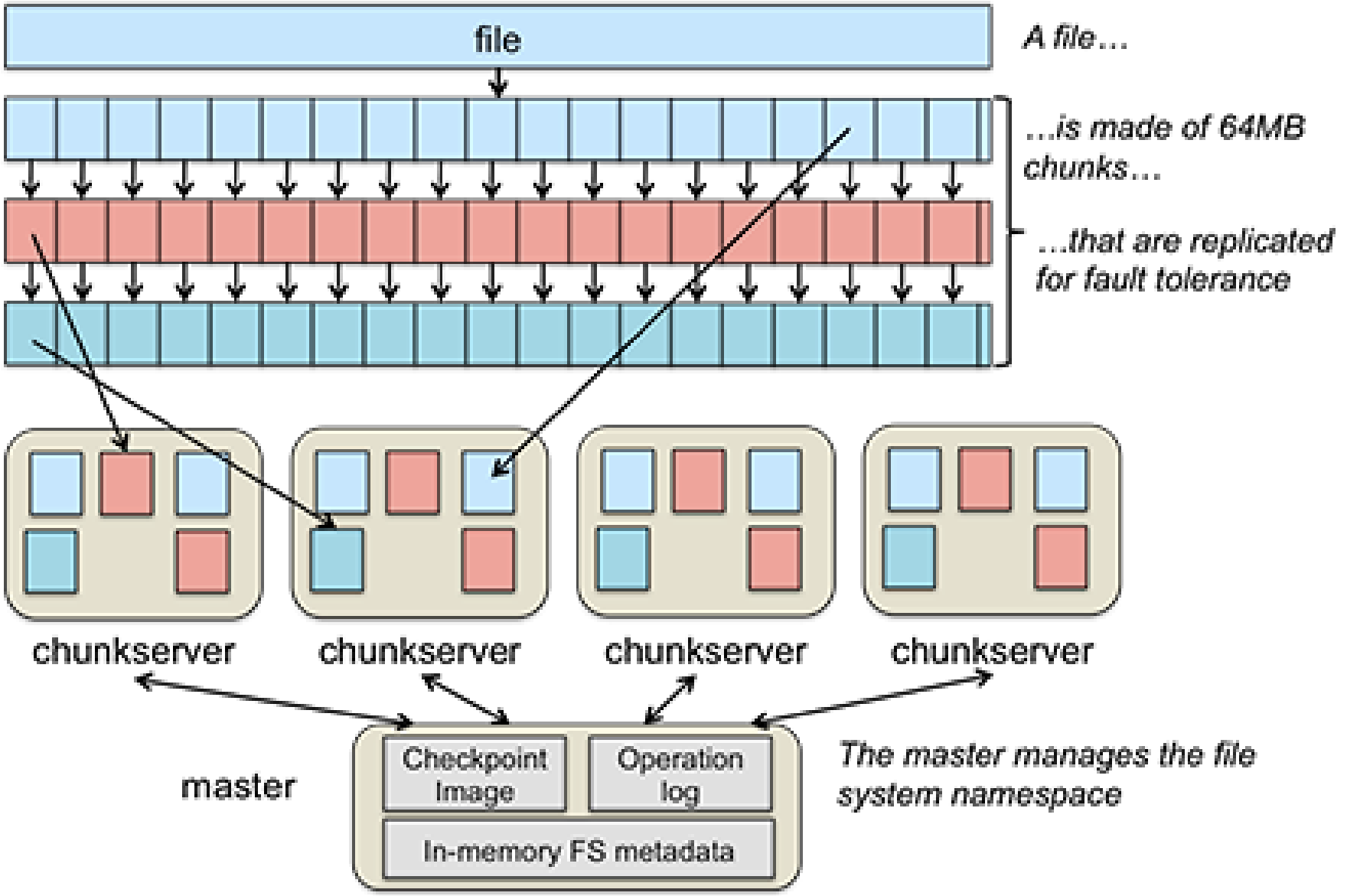
- Productos: search, advertising, email, maps, video, chat, blogger
- Infraestructura de Sistema Distribuido: GFS (Google File System), MapReduce, BigTable
- Plataforma: Máquinas baratas en diferentes data center: commodity hardware



Google File System (GFS)

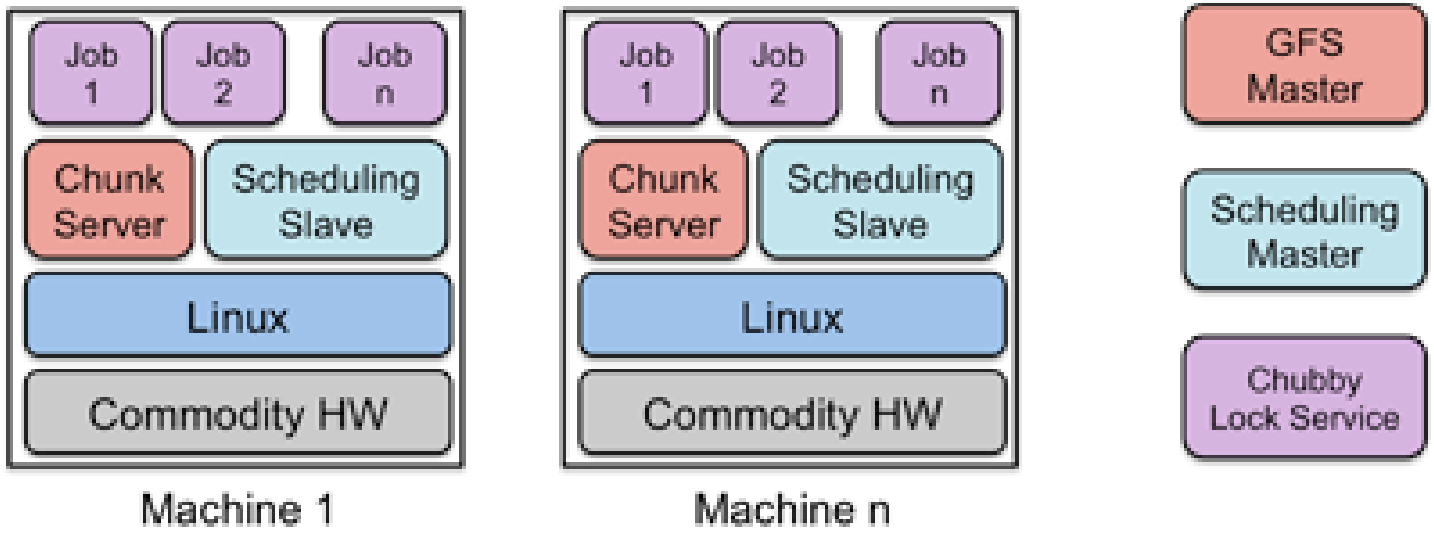
- Sistema de archivos masivo, estructurado, distribuido
  - Alta confiabilidad entre data centers
  - Miles de nodos de red
  - Alto ancho de banda de R/W
  - Bloques de datos en GBs
  - Operaciones:
    - create, delete, open, close, read, write, snapshot, append
- Relación asimétrica:
  - Master(metadata)
  - Chunk Servers(data, replicados-3)




file
A file…
…is made of 64MB chunks…
…that are replicated for fault tolerance
chunkserver
chunkserver
chunkserver
chunkserver
master
Checkpoint Image
Operation log
In-memory FS metadata
The master manages the file system namespace

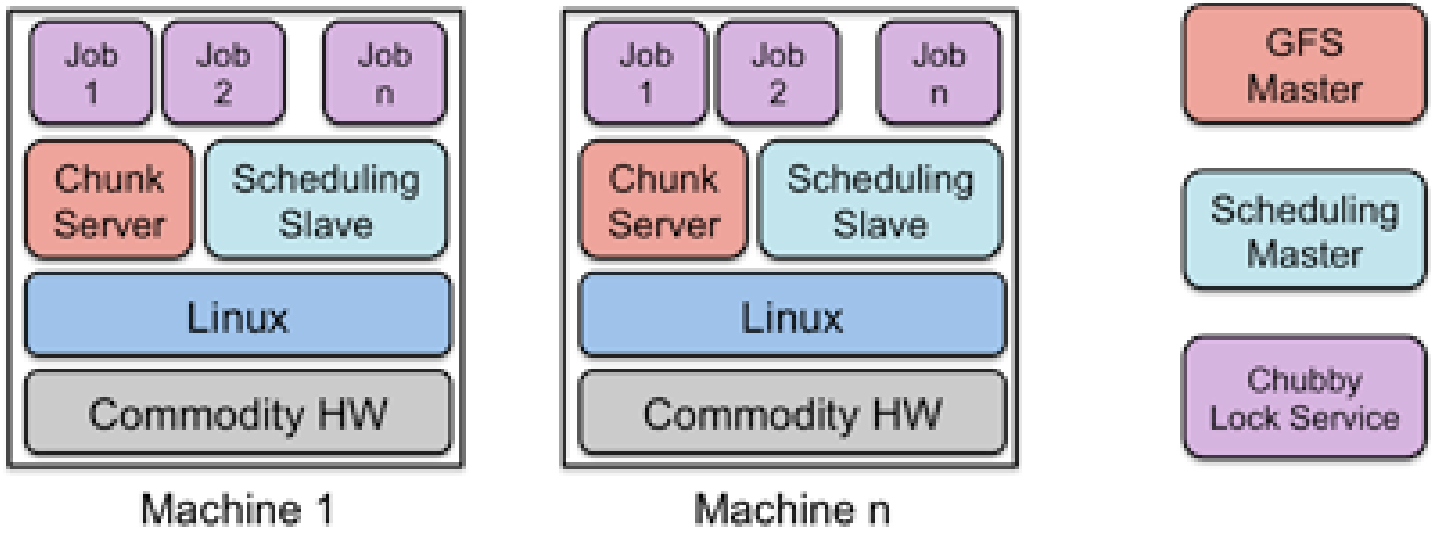
GFS y clusters

- Cientos de miles de Jobs activos
- GFS no tiene interfaz de usuario, es invocado mediante una
- API desde las aplicaciones
- No hard/soft links
- No directorios, solo path



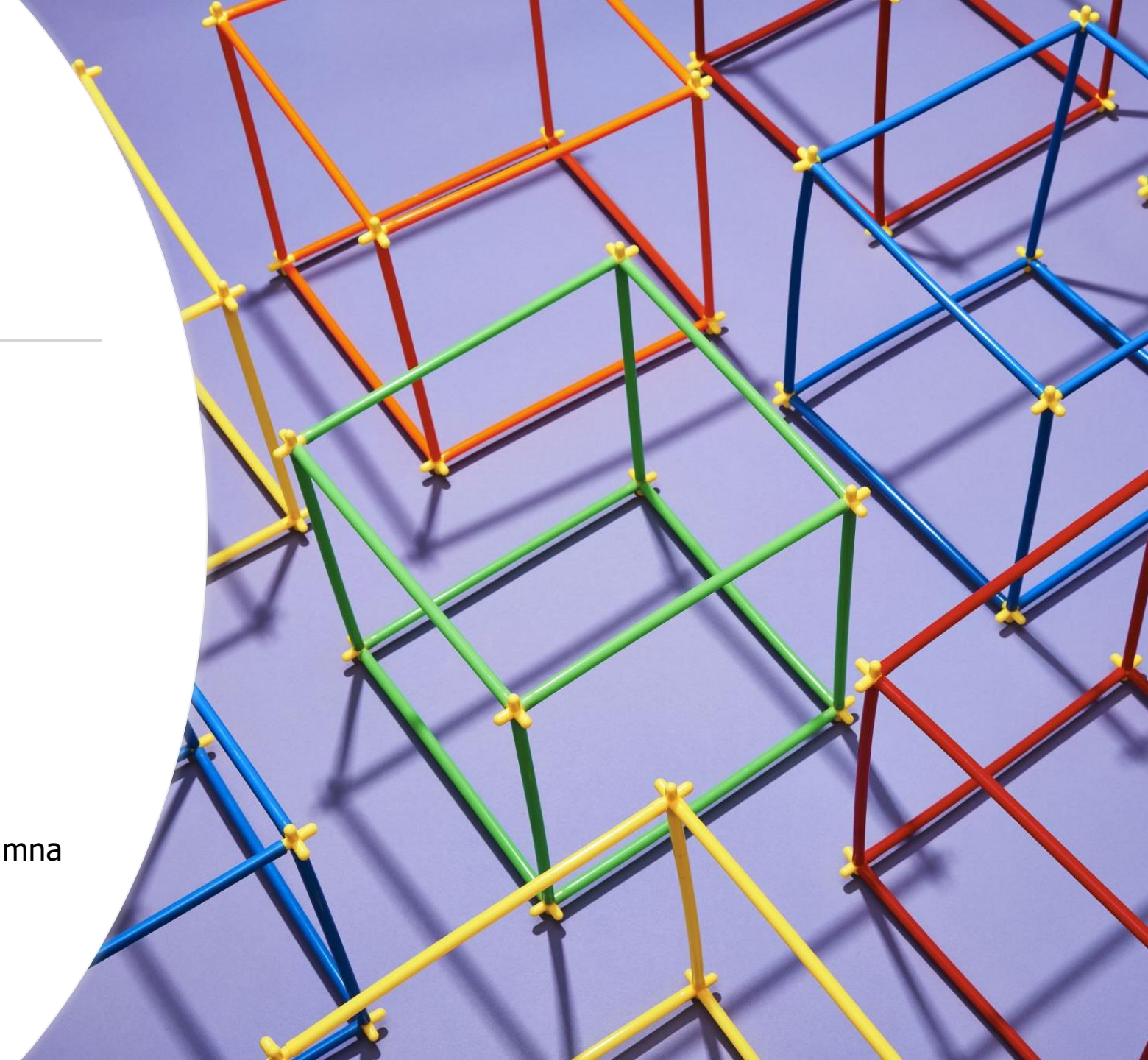
Limitaciones sobre GFS

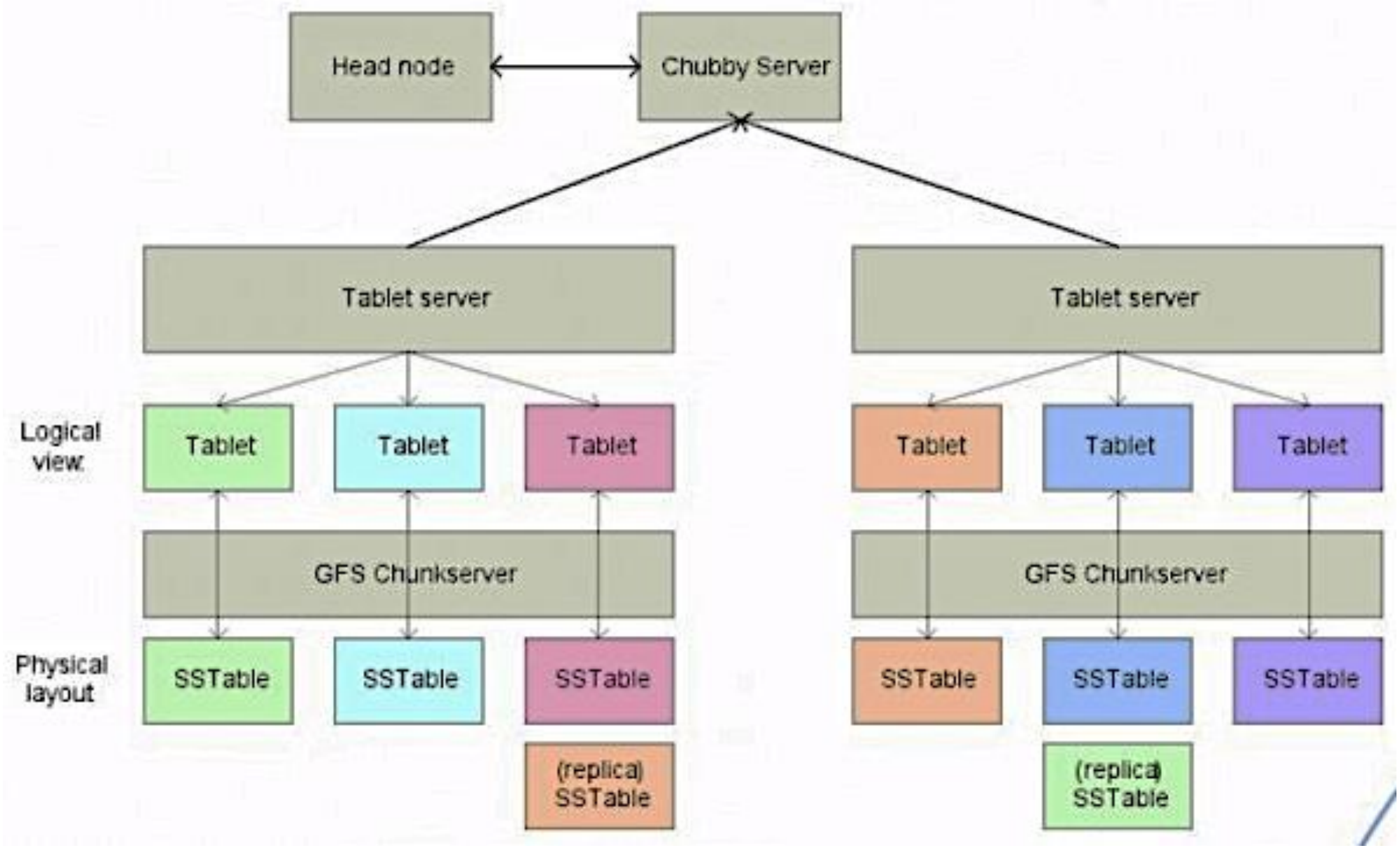
- Performance en archivos pequeños
- Manejo de inconsistencias
- Manejo de metadata en memoria
  - Mucha metadata = mucha memoria



Bigtable

- Más reads que writes
- Fallas en componentes son comunes
- Discos baratos
- Mecanismos de recuperación de datos simplificados
  - (row, col, timestamp)
  - Búsqueda de valores (key, value)
  - No hay operadores relacionales
  - Número de columnas variables
  - Tipos de datos diferentes para cada columna




Head node
Chubby Server
Tablet server
Tablet server
Logical view
Tablet
Tablet
Tablet
Tablet
Tablet
Tablet
GFS Chunkserver
GFS Chunkserver
Physical layout
SSTable
SSTable
SSTable
SSTable
SSTable
SSTable
(replica) SSTable
(replica) SSTable

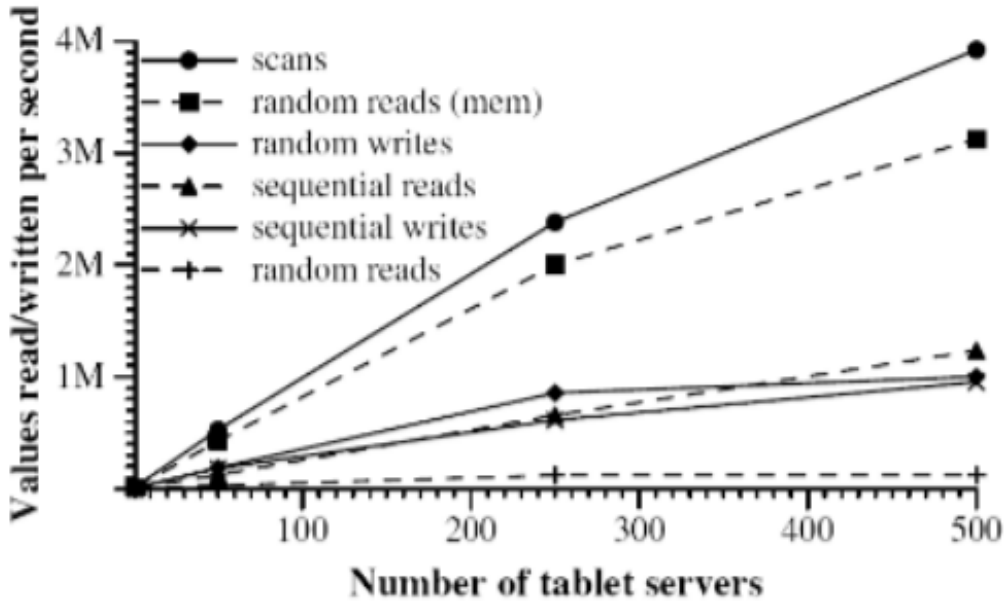
Big table

- BigTable es un mecanismo de hash distribuido construido en un inicio sobre GFS (ahora colossus)
  - No soporta SQL sino búsquedas por key
- Las DBs comerciales no escalan a este nivel y no funcionan sobre miles de máquinas
- Cada dato es guardado en una celda que se accede usando una key de rows, columna, o timestamp
- Cada fila se guarda en una o más tablets
- Una tablet es una secuencia de bloques de 64KB en un formato llamado SSTable

Big Table: Rendimiento

Experiment	# of Tablet Servers			
	1	50	250	500
random reads	1212	593	479	241
random reads (mem)	10811	8511	8000	6250
random writes	8850	3745	3425	2000
sequential reads	4425	2463	2625	2469
sequential writes	8547	3623	2451	1905
scans	15385	10526	9524	7843

Number of 1KB reads/writes per second, per server



Big Table: Tipos de servidores

- Master server:
  - Asigna tablets a servidores Tablet. Siguen la pista de la ubicación de las tablets y las redistribuyen si es necesario
- Tablet server:
  - Procesa requests de lectura/escritura para tablets
  - Divide tablets grandes (100MB - 200MB)
- Si falla, 100 tablet servers toman 1 nueva tablet
- “Chubby” Lock servers:
  - Operaciones tales como abrir una tablet para escritura, árbitro de Master, y ACL requieren exclusión mutua
- Las tablets se cachean en RAM siempre que es posible

Apache Hadoop

- Framework para almacenar y procesar grandes cantidades de datos sobre clusters de computadores baratos
- Ecosistema bajo Apache License 2.0
  - Librerías
  - Hadoop Distributed File System (HDFS)
  - Hadoop YARN (gestión de recursos)
  - Hadoop Map Reduce
- Apache Pig, Apache Hive, Apache Hbase, Apache Spark, etc

Apache Hadoop Ecosystem

Ambari
Provisioning, Managing and Monitoring Hadoop Clusters

Sqoop
Data Exchange

Flume
Log Collector

Zookeeper
Coordination

Oozie
Workflow

Pig
Scripting

Mahout
Machine Learning

R Connectors
Statistics

Hive
SQL Query

Hbase
Columnar Store

YARN Map Reduce v2
Distributed Processing Framework

HDFS
Hadoop Distributed File System

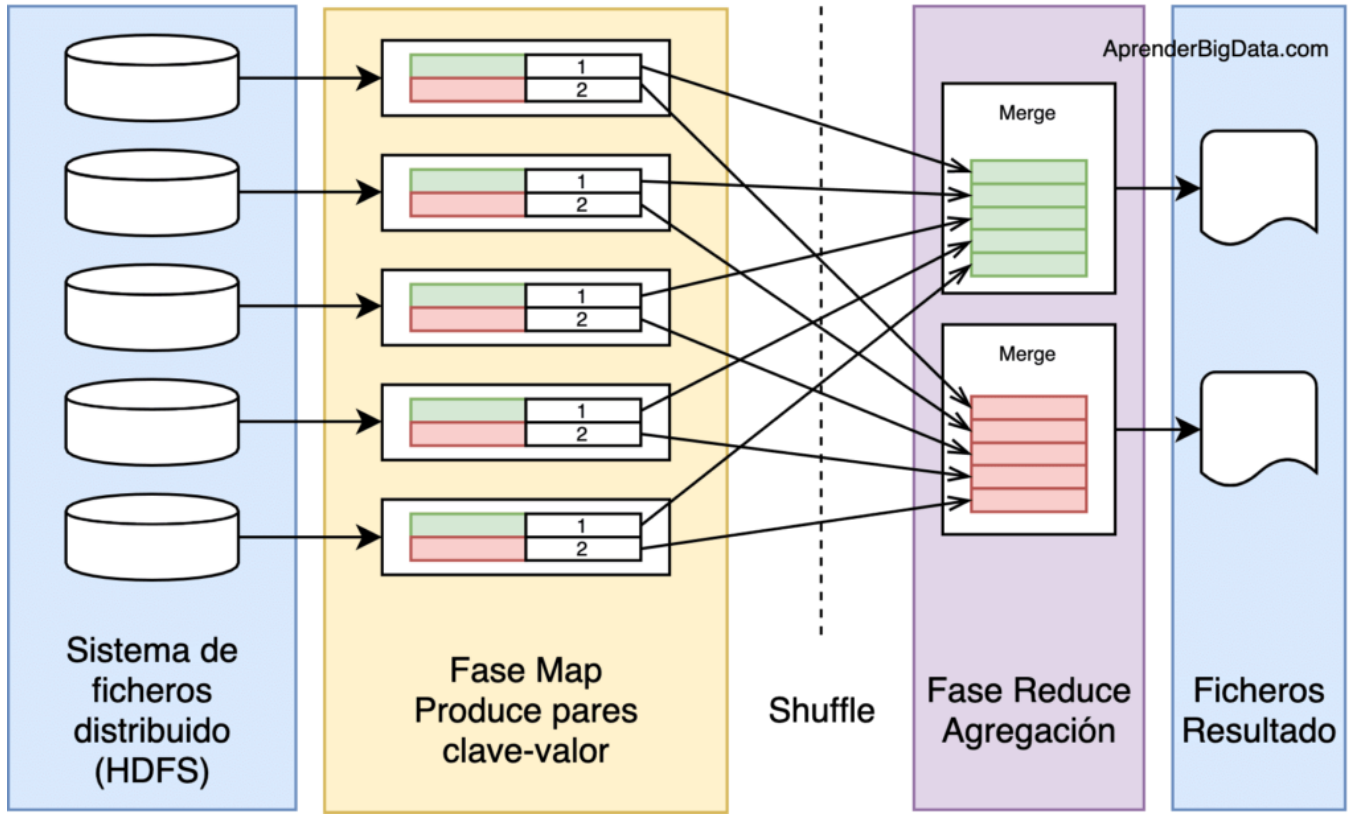
Hadoop: Usos

- Big Data
- Search
  - Creación de índices
- Procesamiento de Logs
  - Text mining
  - Sentiment Analysis
- Sistemas de recomendación
- Modelos predictivos
- Reconocimiento de patrones
- Análisis de video e imagen
- Creación y análisis de grafos



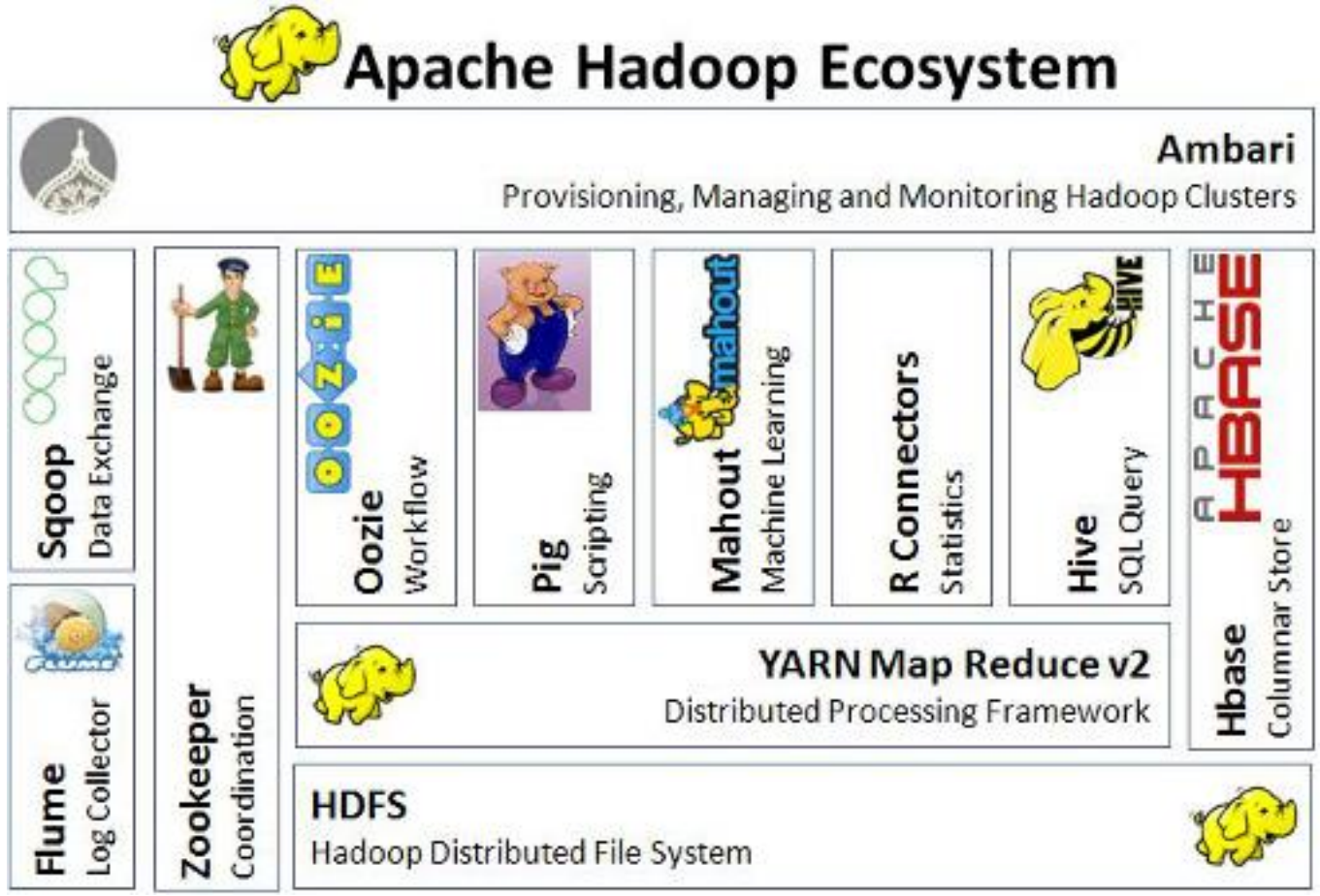
Hadoop: Usos

- Industrias que requieran procesamientos paralelos de grandes cantidades de datos
  - Modelar riesgo
  - Análisis de abandono por clientes (churn)
  - Posicionamiento de publicidad
  - Análisis de transacciones PoS
  - Análisis de datos de red para predecir fallas
  - Análisis de amenazas
  - Supervisión de ventas



Hadoop: Usos

- Datos complejos
- Múltiples fuentes de datos
- Muchos datos
- Procesamiento en batch
- Ejecución en paralelo
- Muchas máquinas



Hadoop: HDFS

Influenciado por GFS

GFS Name	HDFS Name
Master	NameNode
Chunkserver	DataNode
chunk	block
Checkpoint image	FsImage
Operation log	EditLog

Hadoop: Casos

- Tracking de trillones de canciones, likes y más
  - Next Big Sound
    - Spotify, iTunes, Youtube
    - Cluster de 40 nodos Hadoop
    - 10 TB data
- Twitter analiza 100 billones de tweets
  - Hadoop + Pig
- MapReduce

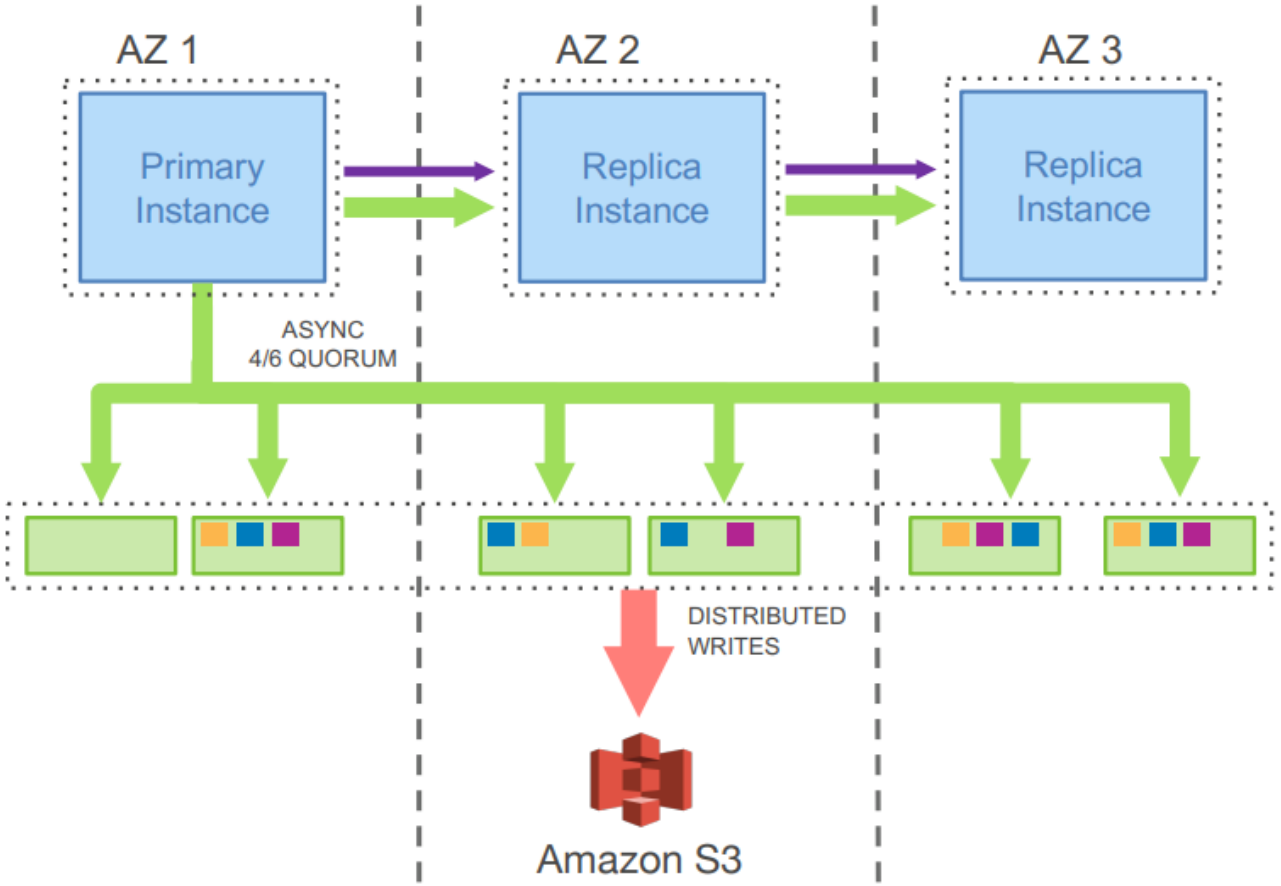
Hadoop: Cons

- NO se puede usar en línea
  - Latencia mínima: 1 min
- Hadoop es muy malo con datos de grafo
- HBase problemas con garbage collection
- Otras soluciones actuales proveen mayor velocidad porque corren en la RAM
  - Optimizan las particiones de Hadoop

AWS Aurora

Base de datos SQL compatible distribuida en la nube

- Pros
  - Escalabilidad bajo demanda
  - Replicas autogestionadas
  - Compatible con DBs Postgres y MySQL
- Cons
  - Costo más elevado que otras soluciones
  - Poco personalizable



Escalabilidad

- IIC2173 - Arquitectura de sistemas de software
- Departamento de Ciencia de la Computación
- Escuela de Ingeniería – PUC
- Nicolás Acosta – Gerardo Crot

