Arquitectura de Sistemas de Software

Departamento de Ciencia de la Computación
Escuela de Ingeniería – PUC
Hans Findel {hjfindel@uc.cl}

Escalabilidad

Diseño para NFRs

- Un requisito no-funcional (NFR) es una restricción sobre la manera en la cual el sistema implementa y entrega su funcionalidad
  - Son multidimensionales
  - Se miden en términos cuantitativos y cualitativos
  - Influyen fuertemente los componentes, conectores y sus configuraciones

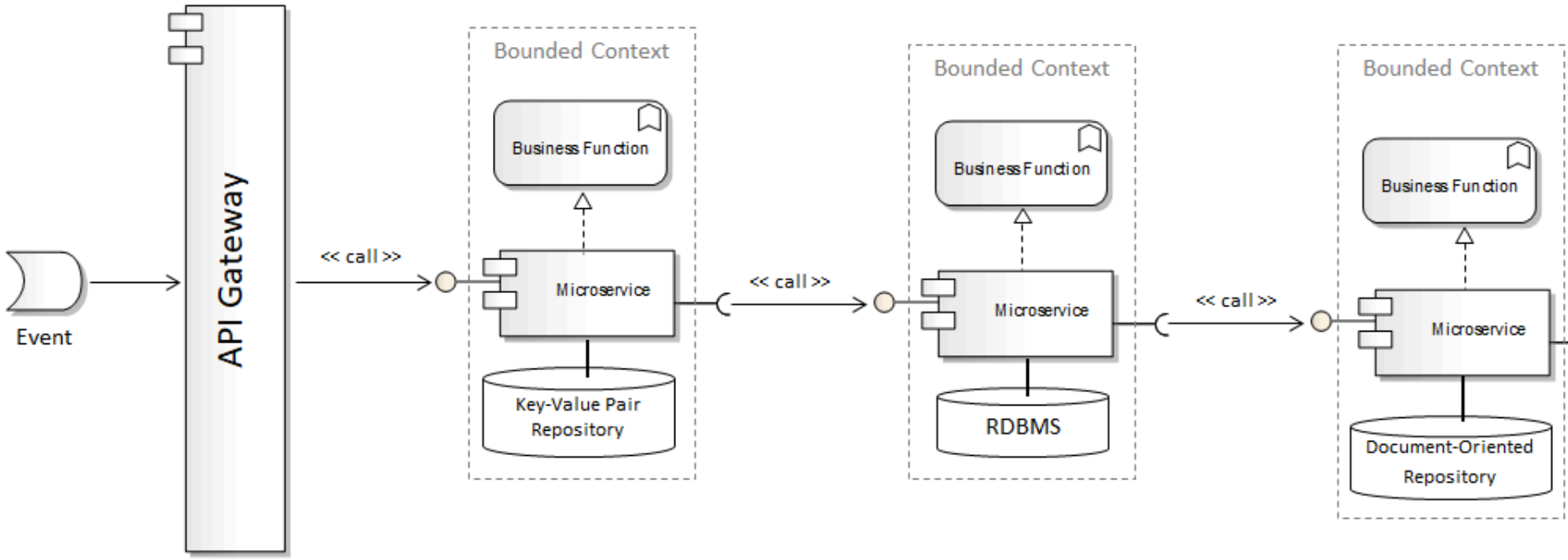
Escalabilidad

- Escalabilidad:
  - Capacidad del software de adaptarse para alcanzar nuevos requisitos de tamaño y alcance
- Heterogeneidad:
  - Capacidad del software de estar formado por múltiples partes o funciones en múltiples y dispares ambientes computacionales
  - Interna o Externa
- Portabilidad:
  - Capacidad del software para ejecutarse en múltiples plataformas (Hw/ Sw) con modificaciones mínimas y sin degradación de sus características funcionales y no funcionales

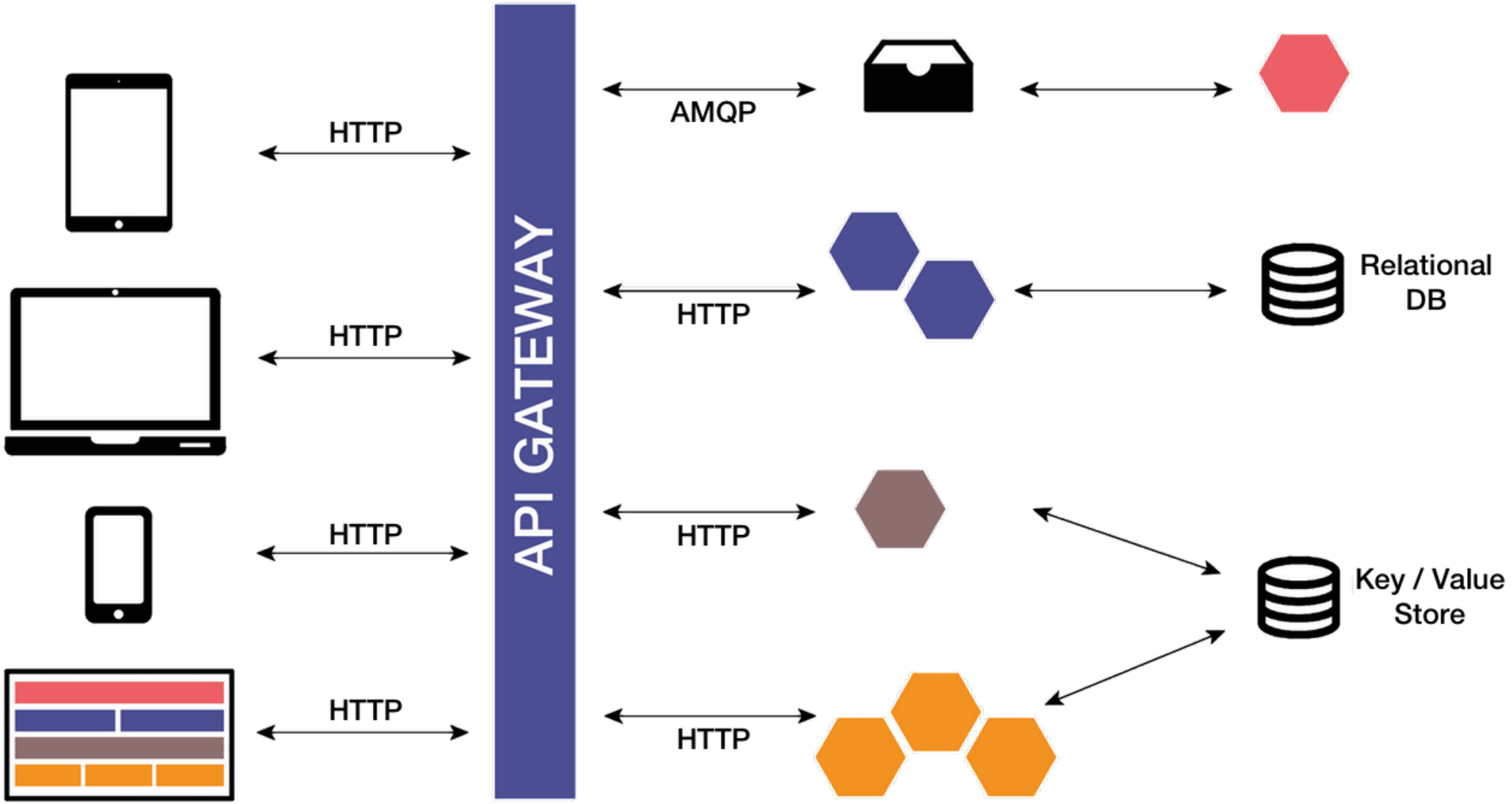
Tipos de escalabilidad

- Horizontal (scale out)
  - Añadir más computadores
  - Mejorar la velocidad de transmisión de datos entre computadores
    - Ethernet Gigabits, InfiniBand, Myrinet
- Vertical (scale up)
  - Mejorar los computadores existentes (CPU, RAM, HDD)
  - Virtualización

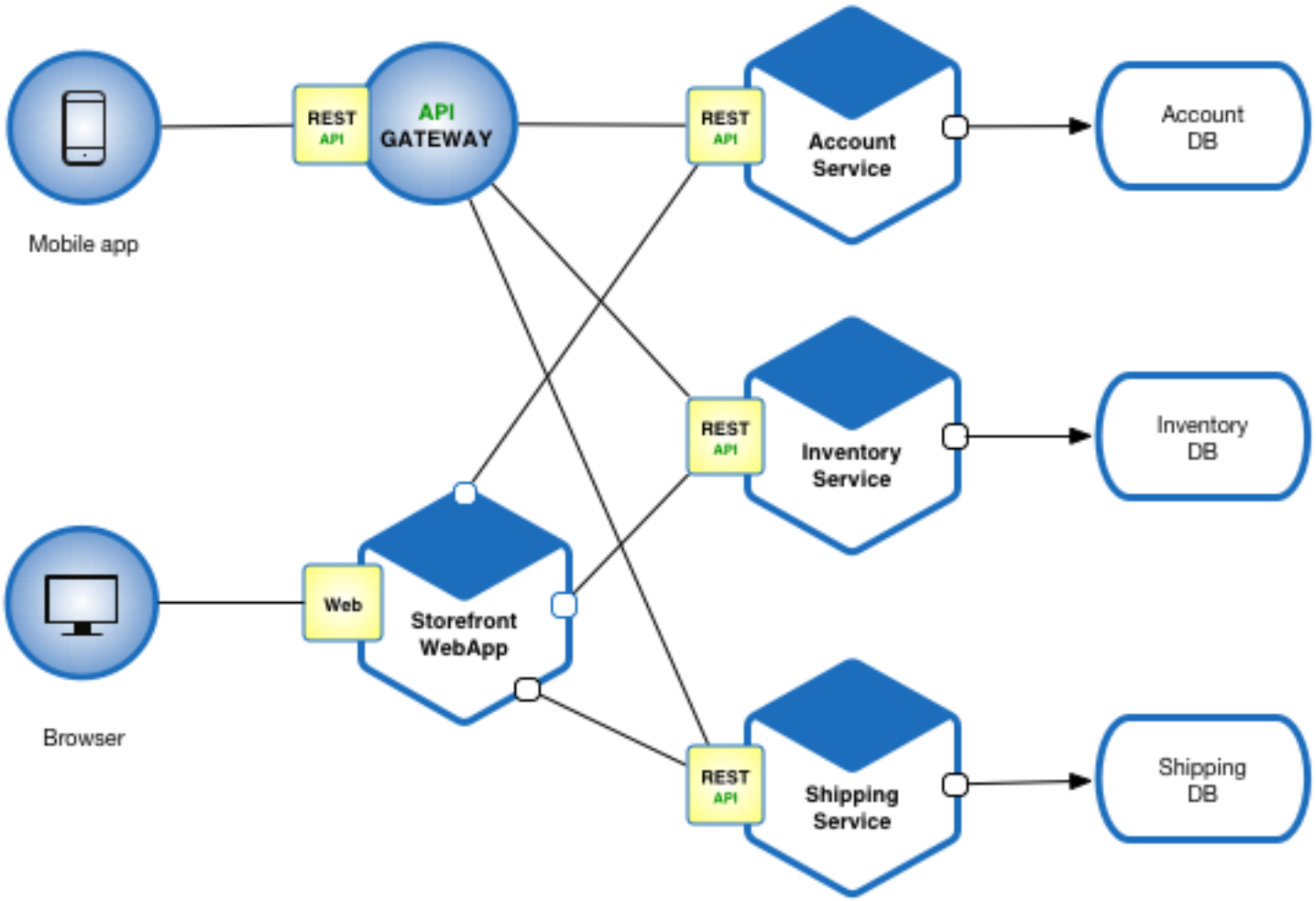
Diagramas de sistemas



Diagramas de sistemas



Diagramas de sistemas



El problema de 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
http://tcrn.ch/I91ZoT

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

favicon.ico

El problema de la escalabilidad

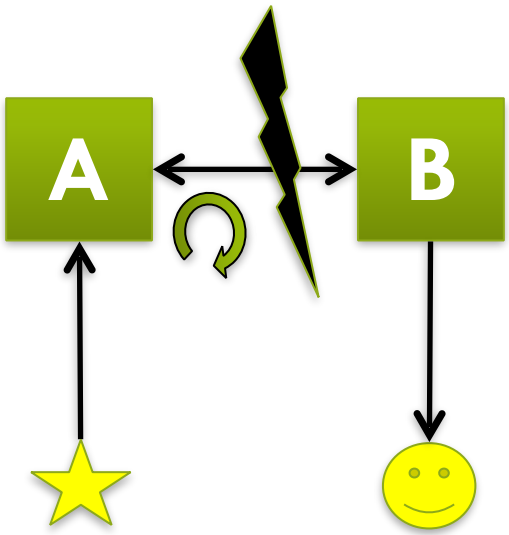
- Qué replicar? Qué minimizar? Qué optimizar? Cómo medir?
  - El “stack” es complejo
  - Las herramientas de medición no solo **no miden lo mismo** sino que pueden introducir retardos
  - La optimización del código, en general, no es significativa *

Al escalar se introducen más problemas … simultáneamente

- Integridad de cachés
  - Cache invalidation
- Concurrencia y paralelismo
  - Transacciones fuera de orden
- Seguridad
  - Se aumentan los puntos vulnerables
- Confiabilidad
  - Se aumentan los puntos de falla a X (fallará X-veces más a menudo!! o en más difícil de encontrar)

CAP Theorem

- Consistency, Availability, Partition tolerance
  - Two-out-of three
- 2000, Conjetura de Brewer, 2002 prueba Gilber & Lynch



Availability + Partition tolerant = No Consistency

Consistency + Partition tolerant = No Available

Consistency + Availability = No Partition tolerant

CAP Theorem

- 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
- BASE (Basically Available, Soft state, Eventually consistent)
  - Soft state: cache

Escalabilidad

- 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
  - Fuentes de datos distribuídas
  - Replicar data cuando es necesario

Escalabilidad

- 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
    - Cache, hoarding, pre-fetching, buffering

Escalabilidad

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

Problemas: Crecimiento data

- Youtube http://highscalability.com/youtube-architecture/
  - Usar MySQL, RAID, 10 disks
    - Comprar hardware : toma tiempo
  - De servidor único a master (una máquina potente, multithread)/slave (máquinas simples, single thread) (redundancia)
  - A réplica con DB particionada
  - A sharding
  - Retardos en escritura por parte de los slaves (pérdida de integridad de datos)

Sharding

- Particionar los datos de manera que tenga sentido para una aplicación (ej. Por índice, 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*
- Ventajas
  - Tablas pequeñas (pocas tuplas -> búsquedas más rápidas)
  - Distribuídas en varias máquinas (tiempo de búsqueda se divide)
  - Cuidado: un buen shard no requiere compartir datos

Sharding

- Riesgos:
  - Complejidad del SQL
  - Complejidad en el software
  - Punto único de falla (un shard corrupto podría destruir todo el sistema)
  - Failover más complejo (hash de los shards)
  - Backup complejo
  - Complejidad operacional (e.g. modificar esquema)
  - 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)
  - Menores tamaños de tablas, mejores desempeños
  - Índeces más eficientes
  - Consultas más complejas

- 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-vs-timescaledb-816ee808bac5

Problemas: Crecimiento data

- Solución
  - Partición de DB
  - Sharding: con usuarios asignados a diferentes shards
  - Distribuir las lecturas y las escrituras
  - Mejora de cache local de disco (menos tiempo de IO)
  - Host providers con administración

  - 5 a 6 data centers propietarios + CDN (cache)
  - 30% de reducción de hardware
  - Retardo se reduce “a 0”
  - La DB puede escalar arbitrariamente
  - NoSQL: BigTable

Cache

- 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

Cambios de Instagram

De :

- Nginx
- Redis
- Postgres
- Django

A :

- Nginx & HAProxy
- Redis & Memcached
- Postgres & Gearman
- Django

Recomendaciones de Instagram

1. Pruebas unitarias y funcionales **extensivas**
2. Manten el sistema DRY (Don´t Repeat Yourself)
3. Acoplamiento débil (notificaciones/señales)
4. Gran parte del código en Python (C sólo si hace falta)
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

Buenas prácticas en el 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…

Buenas prácticas

- 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.)

Links de interés

- Memcached (NoSQL key/value) Facebook, Twitter, Wikipedia
  - http://www.socallinuxexpo.org/scale10x/presentations/modern-memcached
- Lecciones de escalabilidad de Dropbox
  - http://eranki.tumblr.com/post/27076431887/scaling-lessons-learned-at-dropbox-part-1
- Escalando MySQL en Flickr
  - http://code.flickr.com/blog/2010/02/08/using-abusing-and-scaling-mysql-at-flickr/
- 7 lecciones de Reedit
  - http://www.infoq.com/news/2010/05/7-Lessons-Reddit
- 7 (NoSQL) DBs en 7 semanas
  - http://pragprog.com/book/rwdata/seven-databases-in-seven-weeks
- http://highscalability.com/

Servicios y opciones de Amazon

Database

RDS
DynamoDB
ElastiCache
Neptune
Amazon Redshift

Storage

S3
EFS
Glacier
Storage Gateway

Compute

EC2
Lightsail
Elastic Container Service
EKS
Lambda
Batch
Elastic Beanstalk

Ejemplos de escalabilidad

La arquitectura de Google

- La tarea ordenar 1 PB (petabyte)
  - = 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: >20 PBs por día

La arquitectura de Google

- 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

El stack

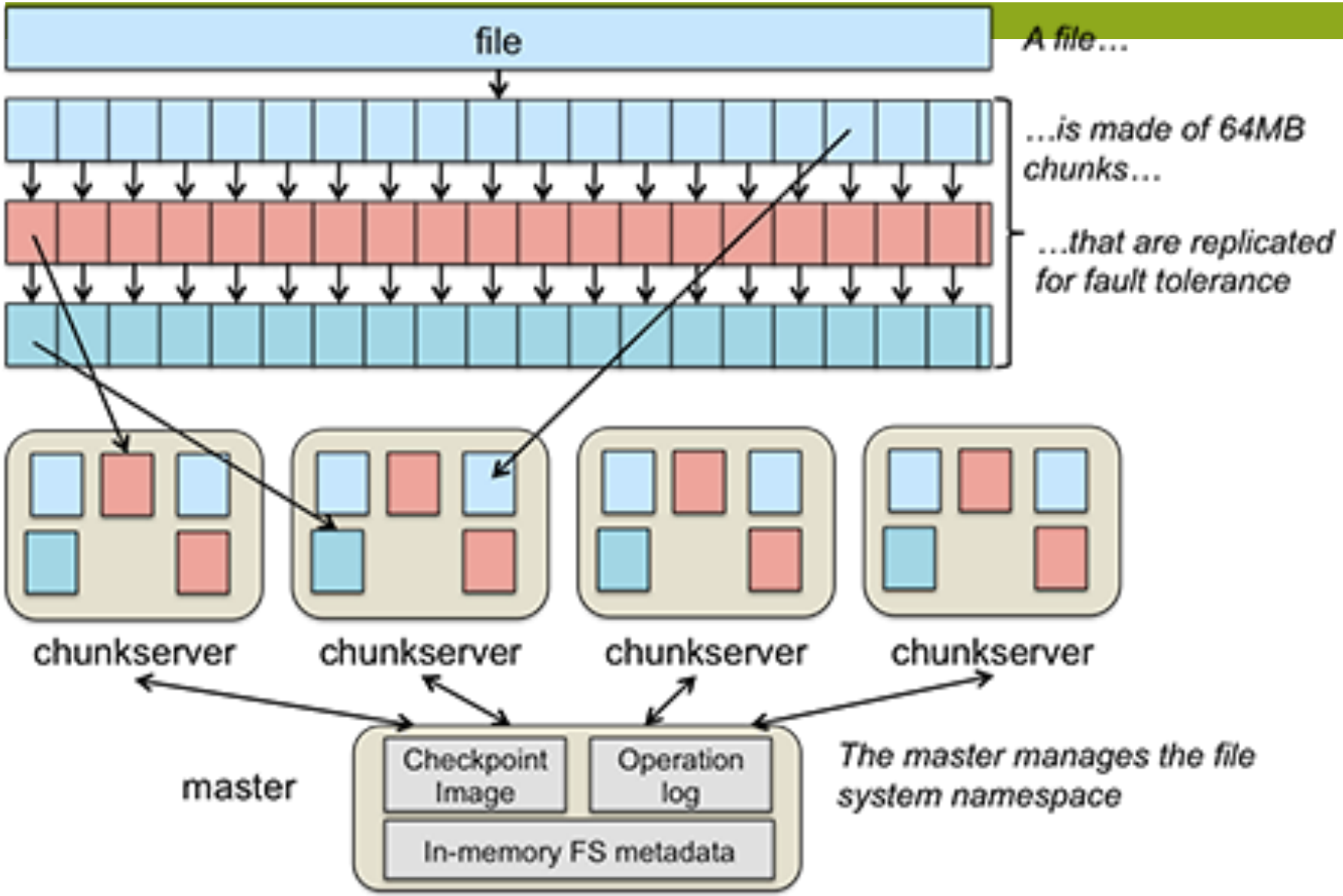
- Productos: search, advertising, email, maps, video, chat, blogger
- Infraestructura de Sistema Distribuído: GFS (Google File System), MapReduce, BigTable
- Plataforma: Máquinas baratas en diferentes data centers

Google File System

- Sistema de archivos masivo, estructurado, distribuído
  - 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

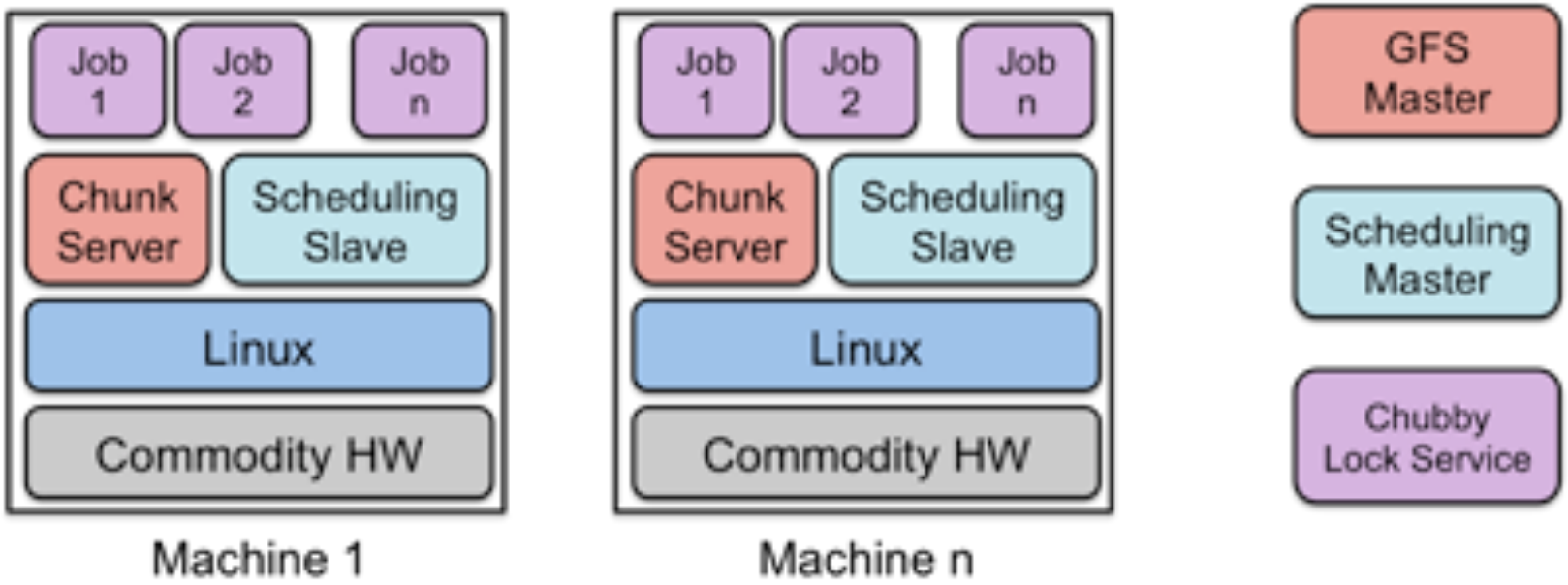
- Master(metadata) - Chunk Servers(data, replicados-3)

Google File System

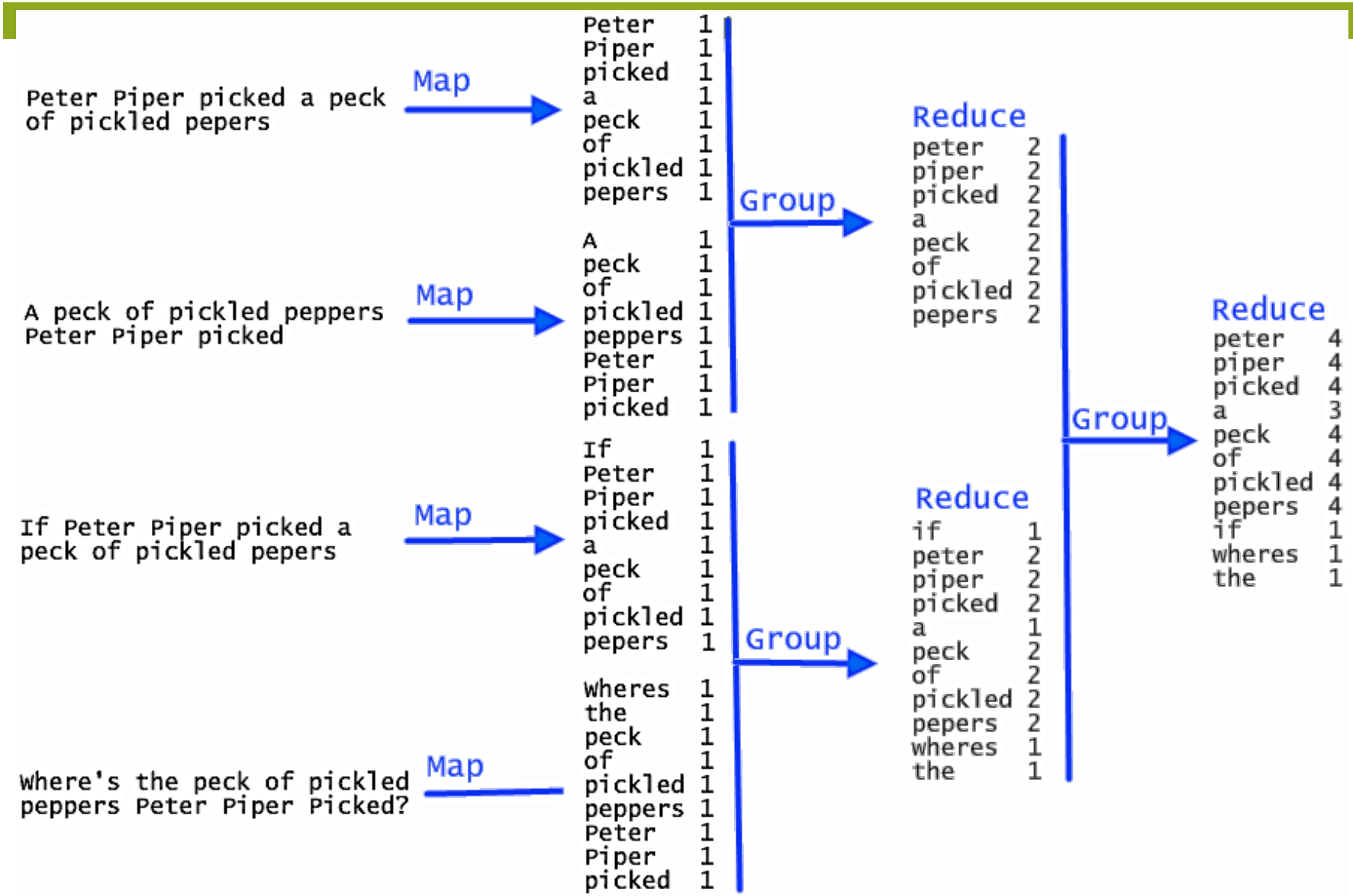


Google File System y los Clusters

- 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 paths



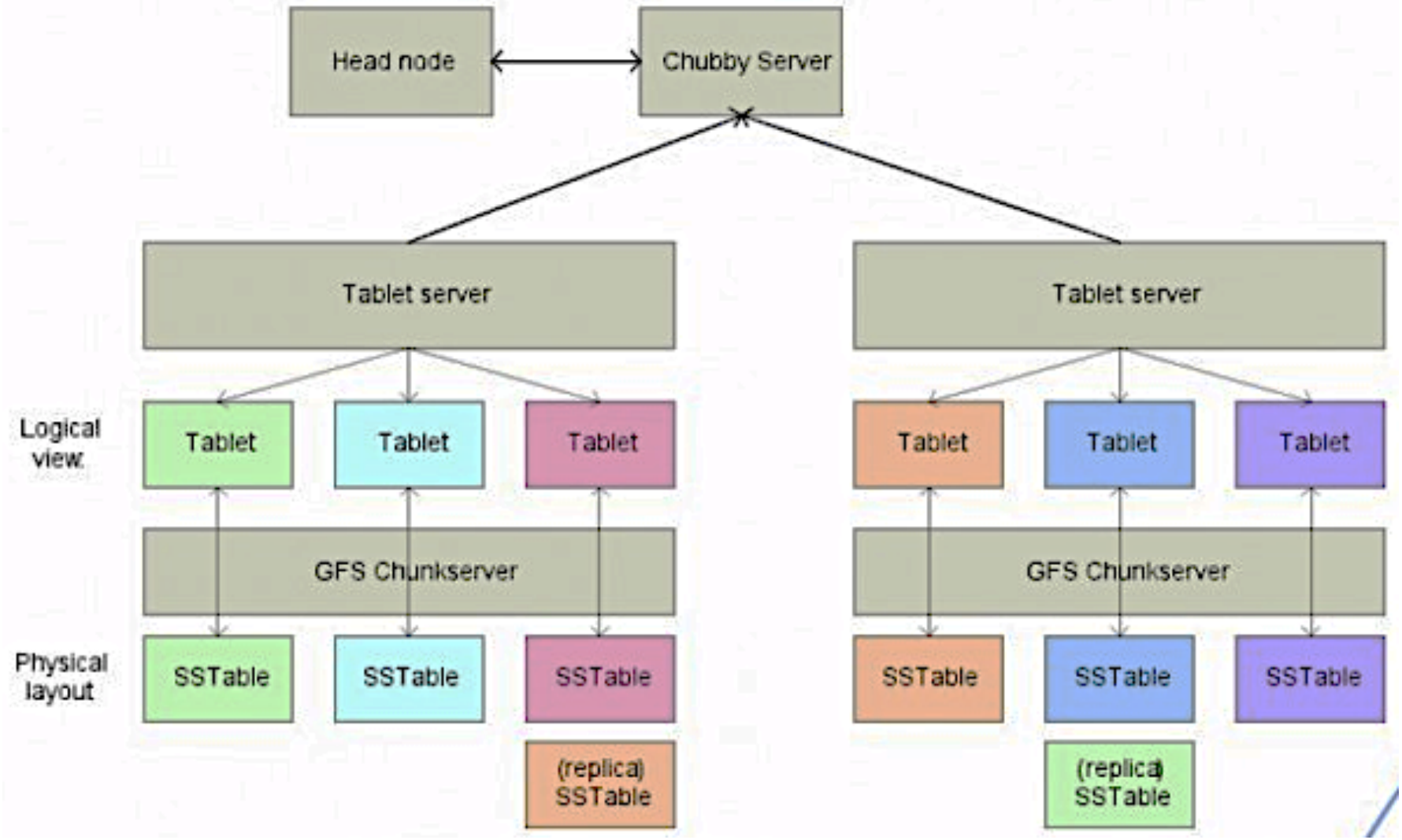
Map Reduce


Peter Piper picked a peck of pickled pepers
Map
Peter 1
Piper 1
picked 1
a 1
peck 1
of 1
pickled 1
pepers 1
A peck of pickled peppers Peter Piper picked
Map
A 1
peck 1
of 1
pickled 1
peppers 1
Peter 1
Piper 1
picked 1
If Peter Piper picked a peck of pickled pepers
Map
If 1
Peter 1
Piper 1
picked 1
a 1
peck 1
of 1
pickled 1
pepers 1
Where's the peck of pickled peppers Peter Piper Picked?
Map
Wheres 1
the 1
peck 1
of 1
pickled 1
peppers 1
Peter 1
Piper 1
picked 1
Group
Group
Reduce
peter 2
piper 2
picked 2
a 2
peck 2
of 2
pickled 2
pepers 2
Reduce
if 1
peter 2
piper 2
picked 2
a 1
peck 2
of 2
pickled 2
pepers 2
wheres 1
the 1
Group
Reduce
peter 4
piper 4
picked 4
a 3
peck 4
of 4
pickled 4
pepers 4
if 1
wheres 1
the 1

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

Bigtable


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

Bigtable

- BigTable es un mecanismo de hash distribuído construído sobre GFS
  - 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

Bigtable: Tipos de servidores

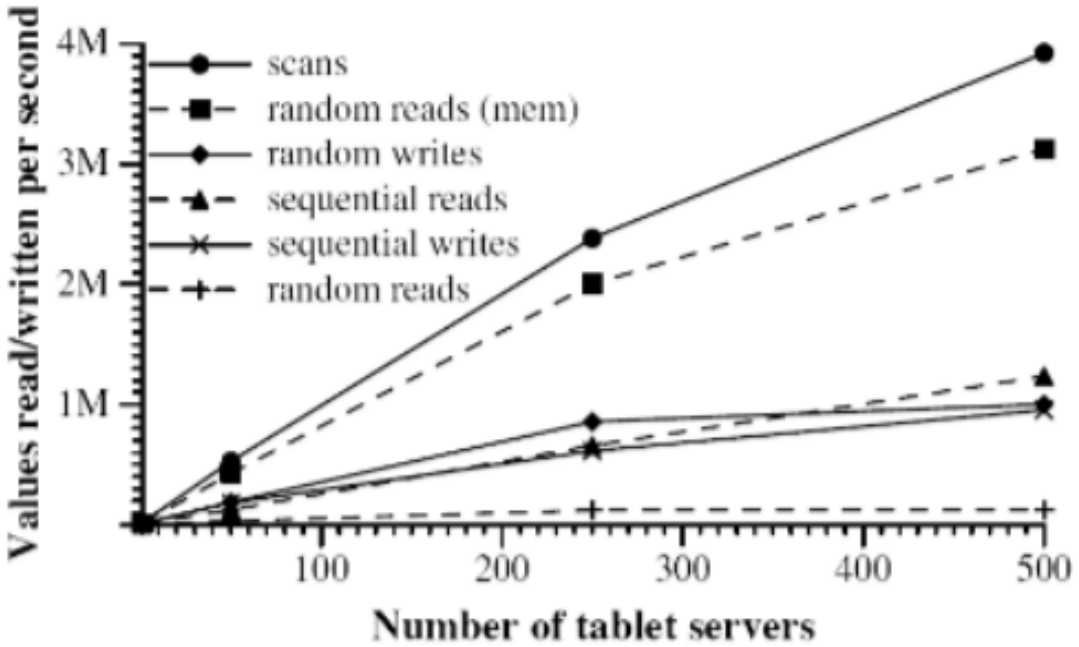
- Master server:
  - Asigna tablets a servidores Tablet. Siguen la pista de la ubicación de las tablets y las redistribuyen si es necesario
- Tablet server:
  - Procesa requests de lectura/escritura para tablets
  - Divide tablets grandes (100MB - 200MB)
  - Si falla, 100 tablet servers toman 1 nueva 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

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



Apache 

- Framework para almacenar y procesar grandes cantidades de datos sobre clusters de computadores baratos
- 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

Para que sirve?

- 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

Para que sirve?

- 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

Para que sirve

- Datos complejos
- Múltiples fuentes de datos
- Muchos datos

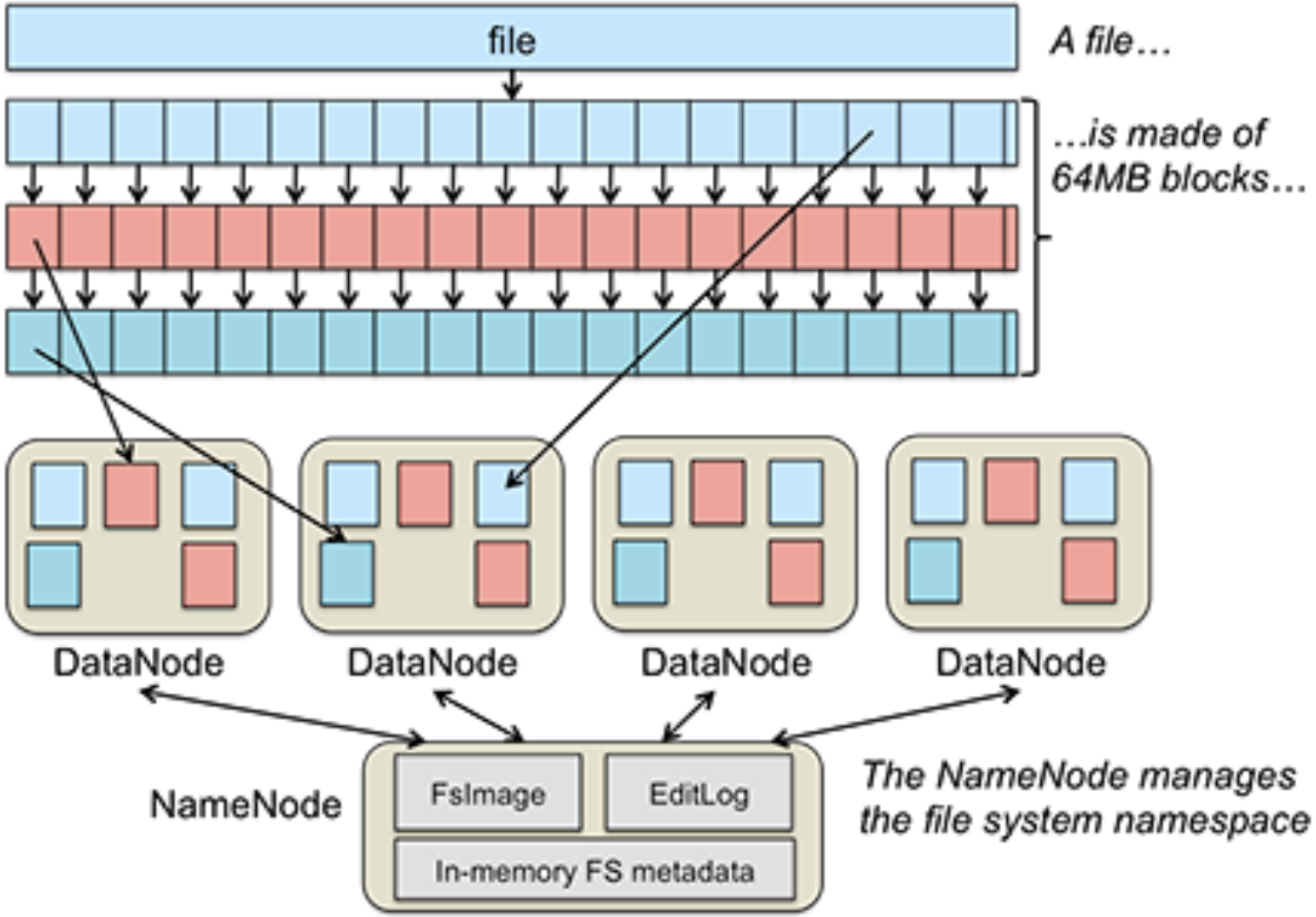
- Procesamiento en batch
- Ejecución en paralelo
- Muchas máquinas

Hadoop HDFS

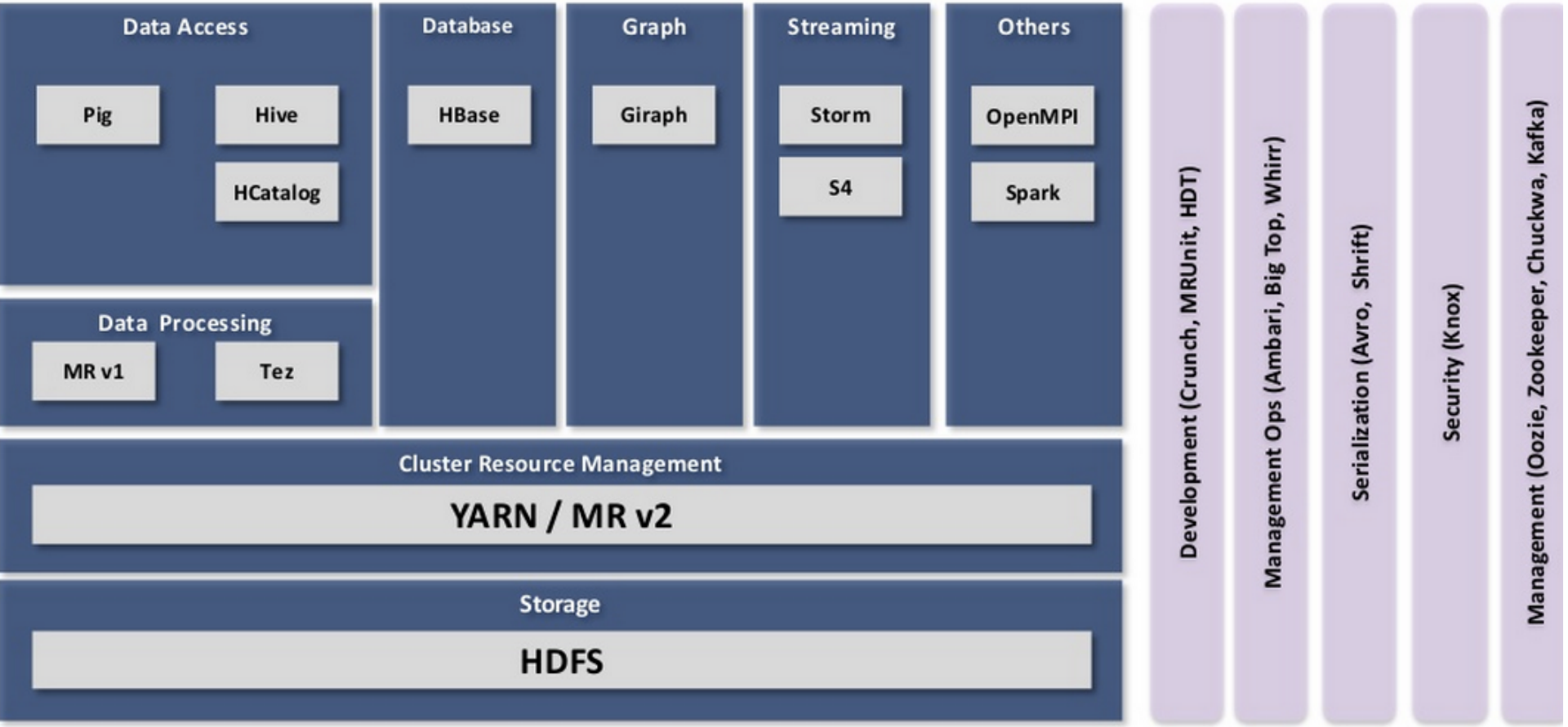
- Fuertemente influenciado por Google GFS

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

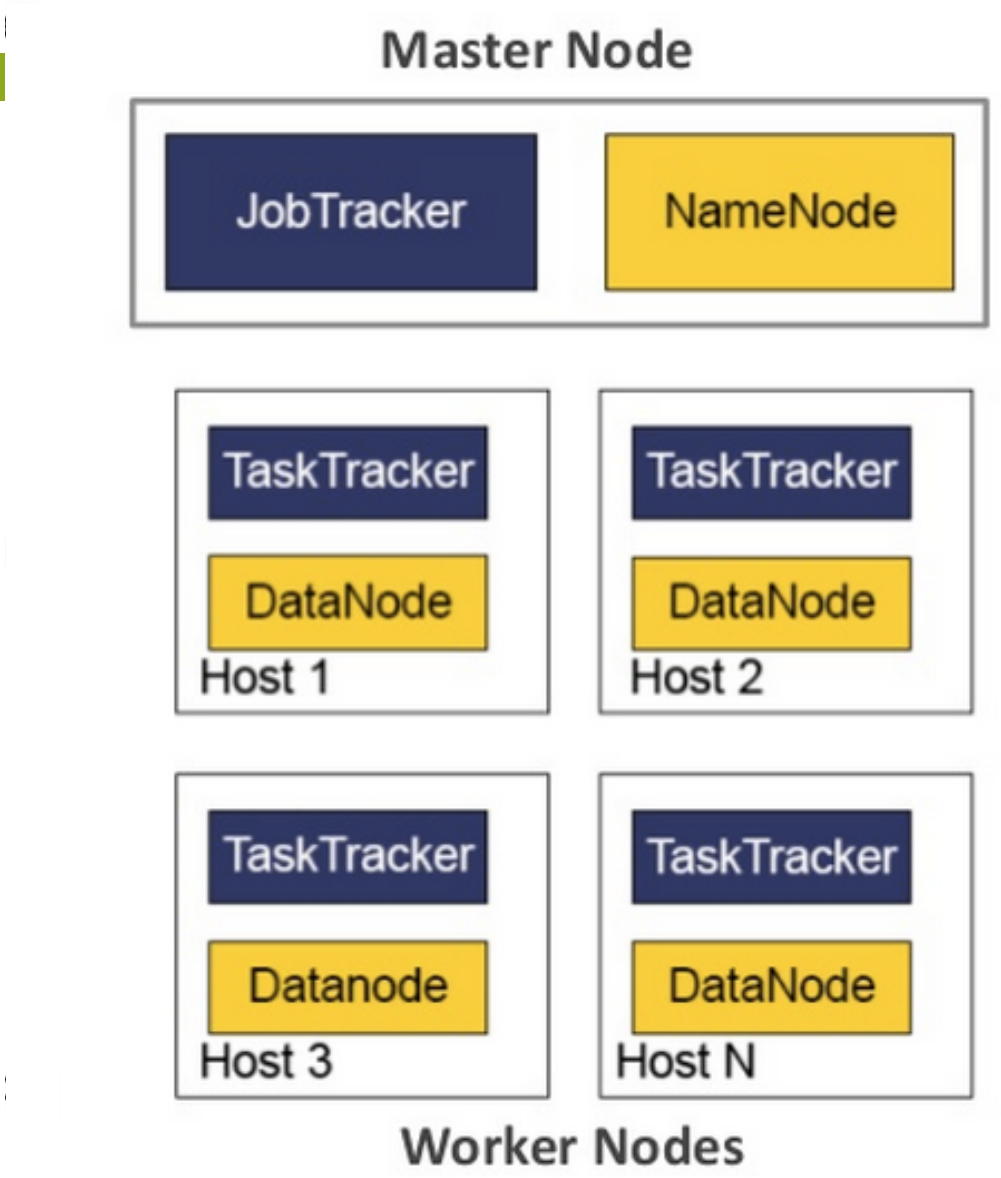
Hadoop HDFS


file
A file…
…is made of
64MB blocks…
DataNode
DataNode
DataNode
DataNode
NameNode
FsImage
EditLog
In-memory FS metadata
The NameNode manages
the file system namespace

Ecosistema de Hadoop



Hadoop MapReduce



Monitoreo

- Nagios
- Ganglia
- Statsd + Graphite
- Zenoss
- Cube
- Lipstick
- Prometheus

Aplicaciones

- 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

Limitaciones

- 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