L'esprit de l'escalier

NoSQL:br v2 2011

Who ?

John D. Rowell

- @jdrowell
- github.com/jdrowell
- jdrowell.posterous.com

Gleicon Moraes

- @gleicon
- github.com/gleicon
- 7co.cc
- restmq.com

Você já deveria estar usando NoSQL

NoSQL:br v1

- Tudo era novidade
- Maioria dos projetos em beta
- Documentação escassa
- "Ninguém nunca foi demitido por usar Oracle"

NoSQL:br v2

- Projetos maduros
- Documentação, wikis, Chef recipes, the works
- Use cases bem definidos e battle tested
- MongoDB é o novo MySQL

Você já deveria conhecer NoSQL

github repo count

mysql: 2197
mongodb: 1685
redis: 1322
couchdb: 1279
postgres: 759
memcached: 665
cassandra: 402
membase: 40
voldemort: 32
voltdb: 7

Você já deveria estar usando NoSQL

- Toolbox de modelos de armazenamento de dados, além de cache.
- Pensar na estrutura dos dados e na melhor maneira de armazená-los

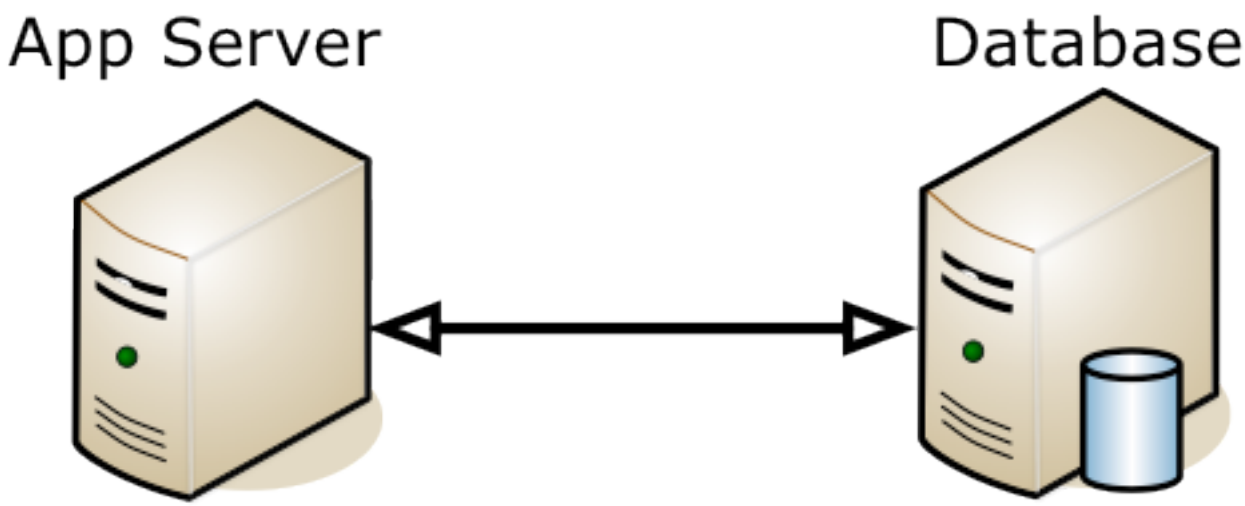
Você já deveria estar usando NoSQL ?

Sistemas legados:

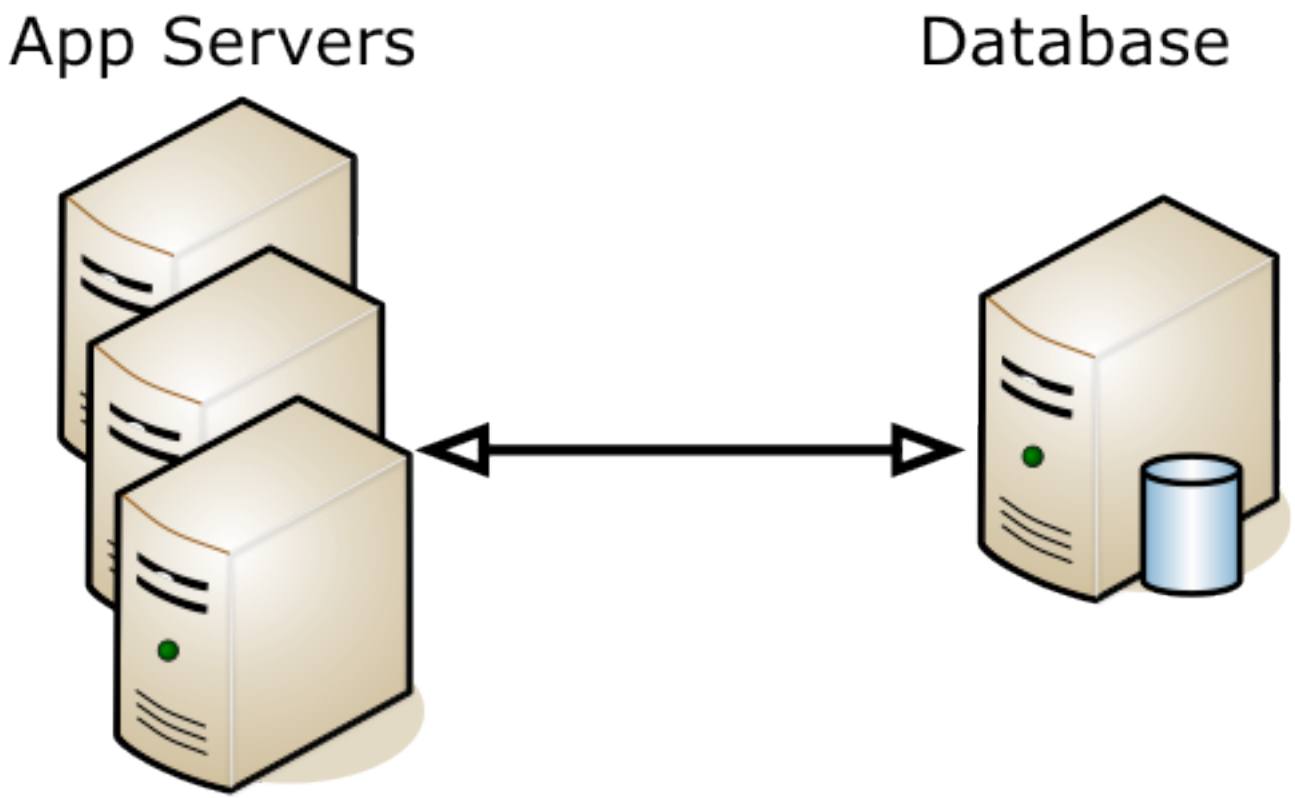
Dependente de cache ?
Queries complexas ?
Tentar reproduzir o que já existe com um SGBD não relacional, sem todos os recursos utilizados pode causar mais problemas.

Dependente de ORM (ou adaptadores de Objetos -> NoSQL)

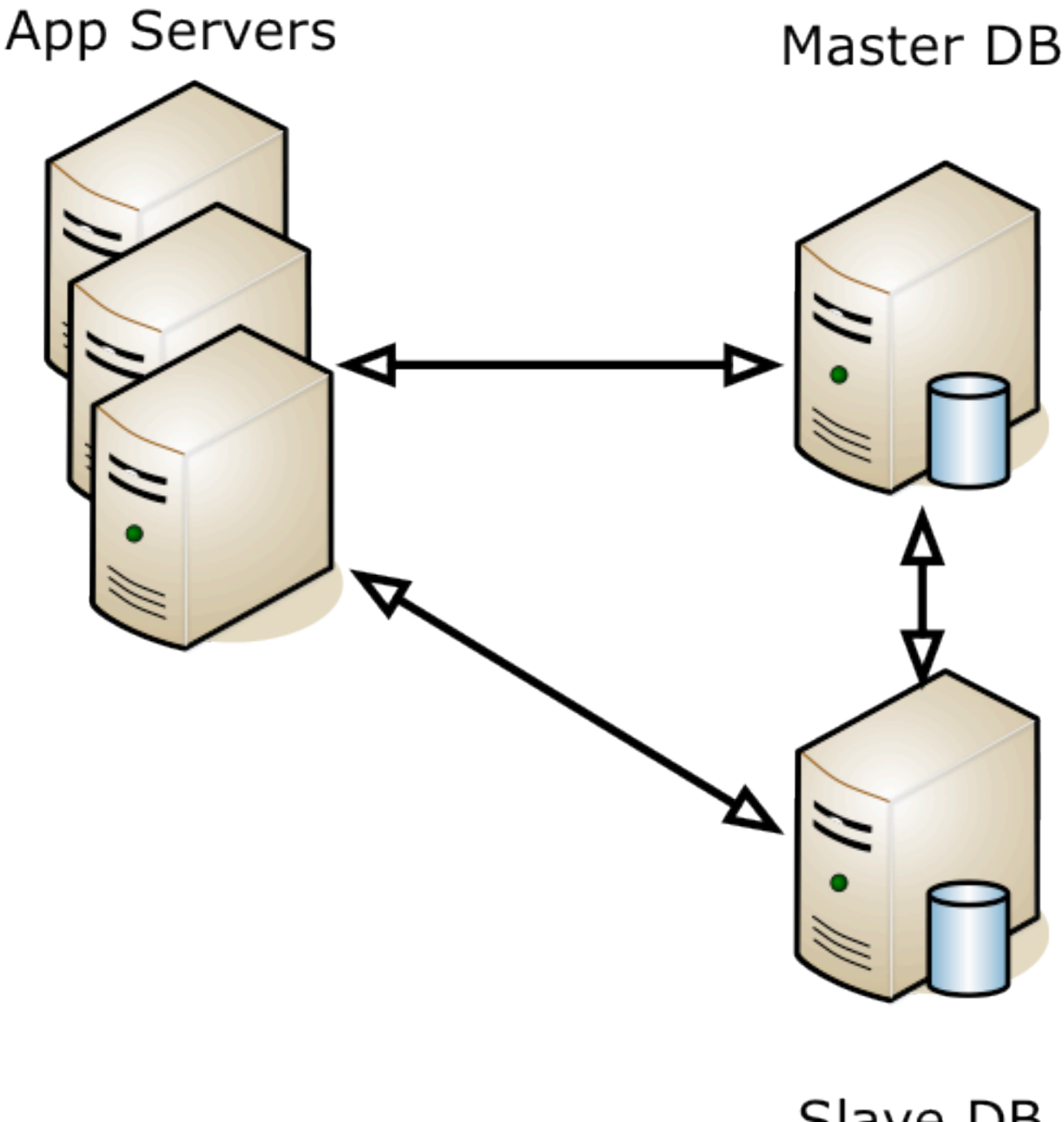
Era uma vez ...



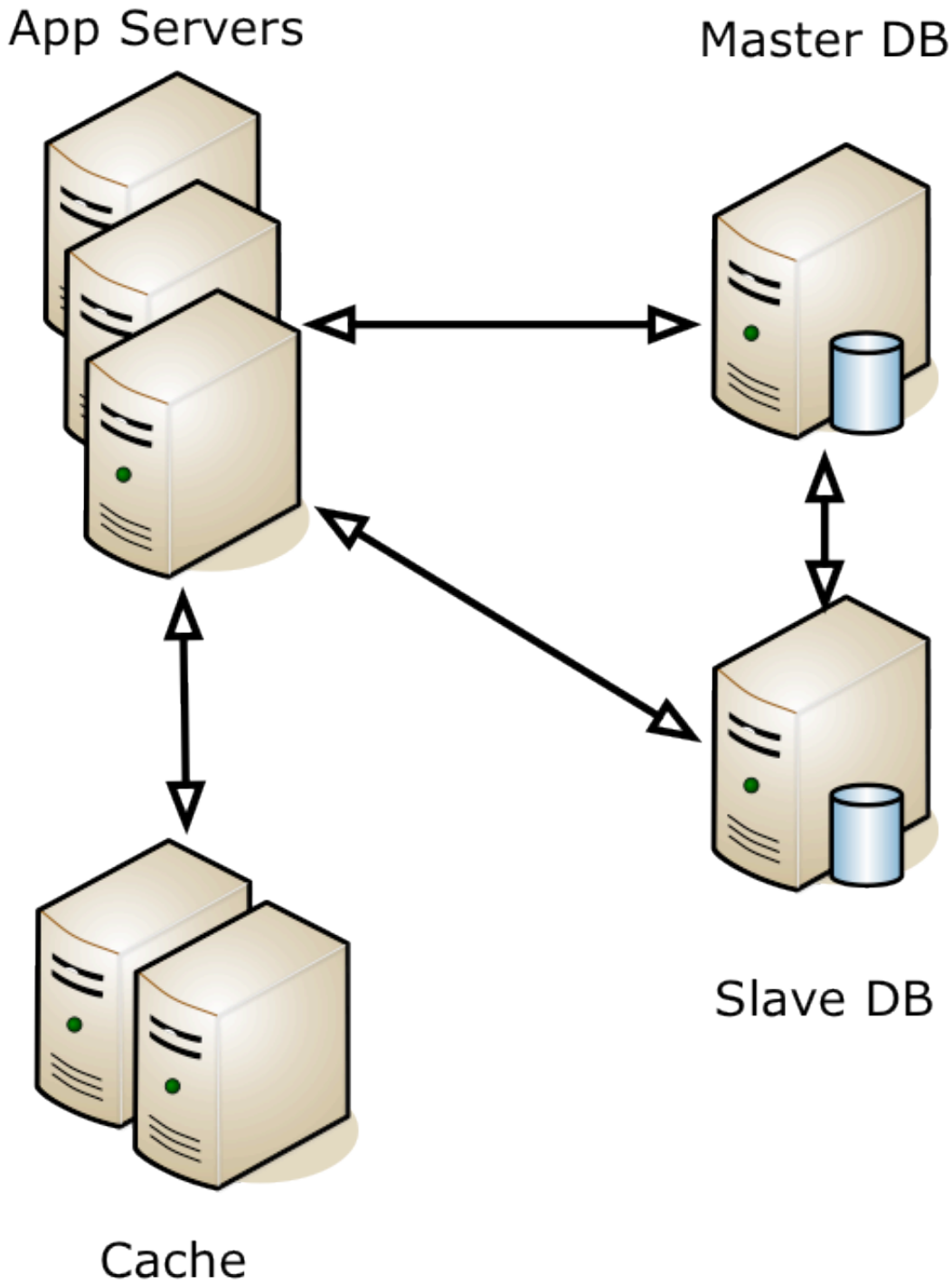
Era uma vez ...



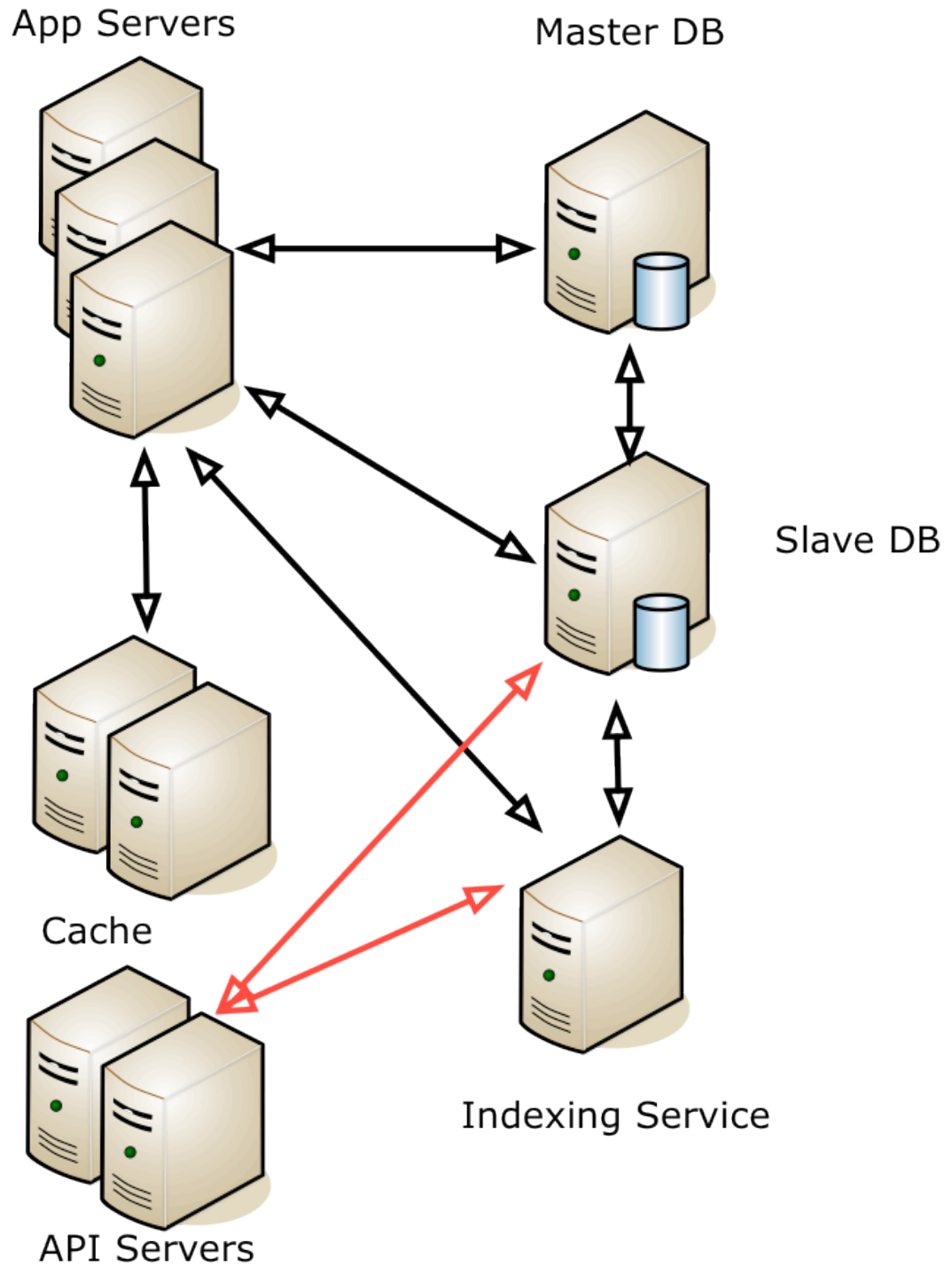
Era uma vez ...



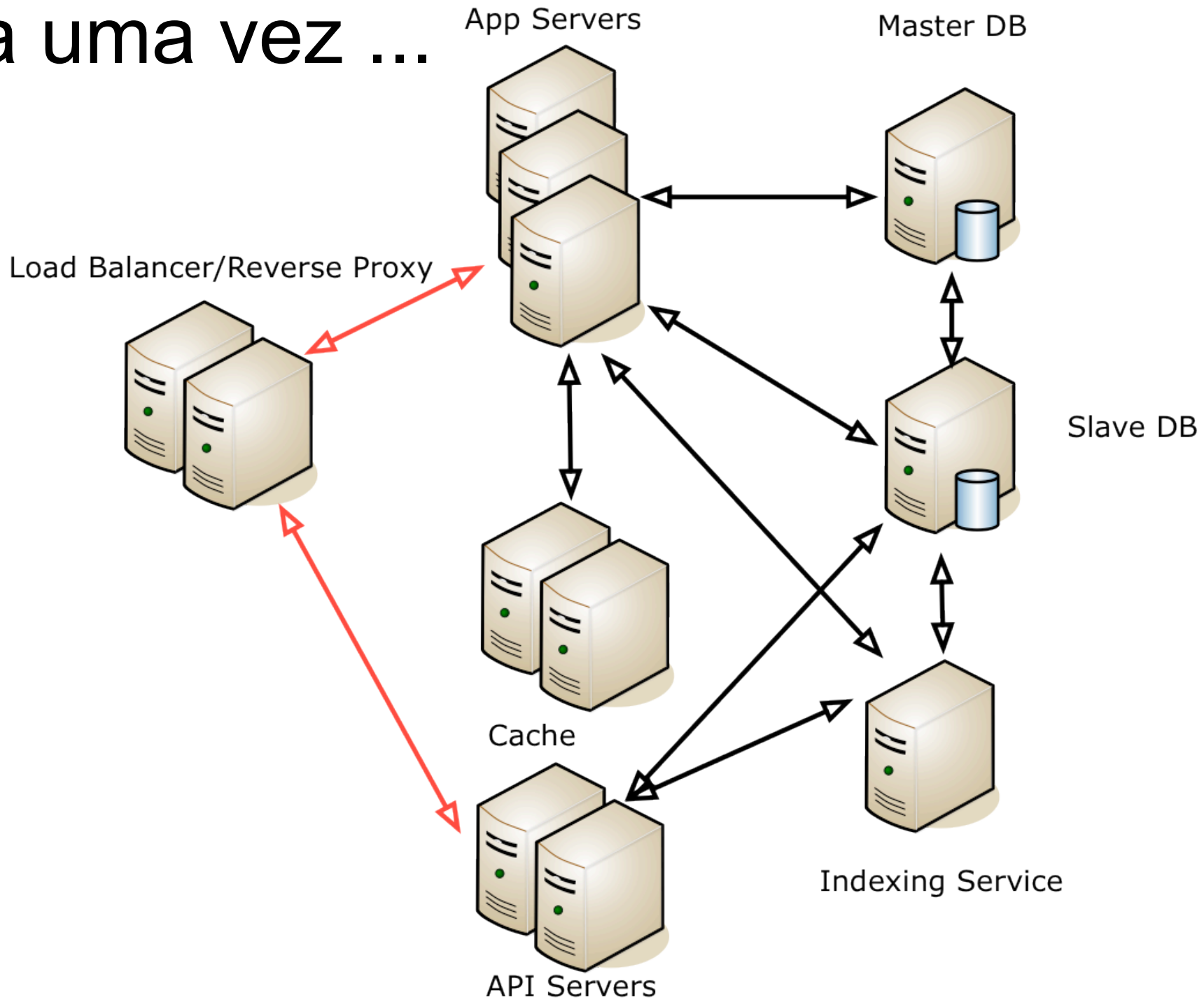
Era uma vez ...



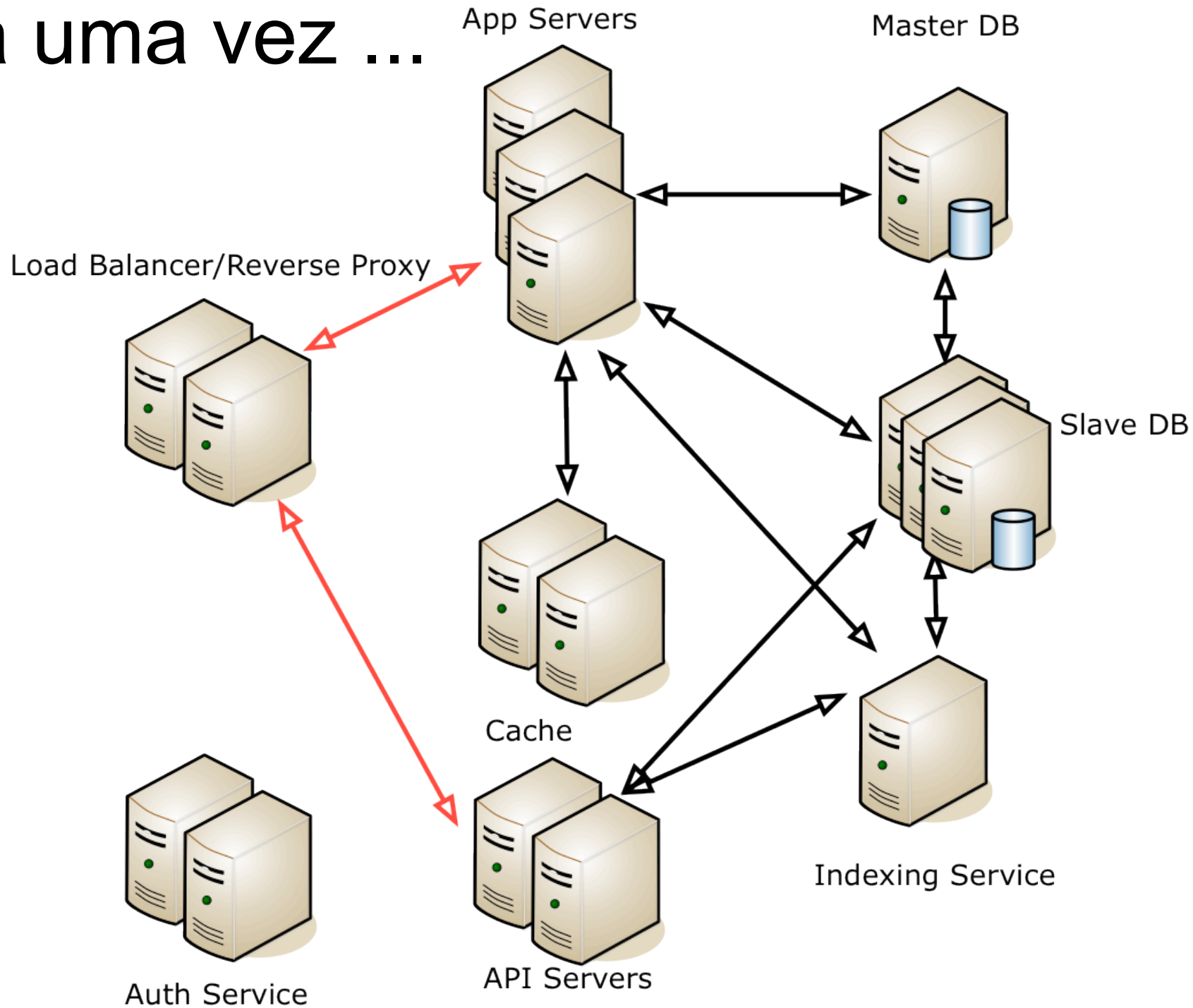
Era uma vez ...


App Servers
Master DB
Slave DB
Cache
API Servers
Indexing Service

Era uma vez ...


App Servers
Master DB
Load Balancer/Reverse Proxy
Slave DB
Cache
Indexing Service
API Servers

Era uma vez ...


App Servers
Master DB
Load Balancer/Reverse Proxy
Slave DB
Cache
Indexing Service
Auth Service
API Servers

Arquitetura

Newsflash: Banco de Dados relacionais não é silver tape

- Message Queues
- Job Schedulers
- Distributed Counters
- Distributed Lock
- Document Storage
- Filesystem
- Cache
- Logger
- Workflow Orchestrator

Estudo de caso - Guia de Cidades

- Apontador, Terra, Maplink, GuiaMais, etc.
- Em relacional, facinho, cada informação na sua tabela
- Tabela de empresas, tabela de filiais, tabela de endereços, tabela de cidades, tabela de usuários, ad nauseum
- Cada update toca diversas tabelas e índices
- Updates concorrentes geram muitos locks
- Cada melhoria no sistema gera novas tabelas, que necessitam de migrações de schema
- Release do código tem que casar com a versão do schema do db
- As operações mais complexas (ex: gerar nuvem de tags) são O(N) e o sistema fica lento exponencialmente com o aumento do volume de dados
- VAMOS MIGRAR PARA NOSQL!

Estudo de caso - Guia de Cidades Usando Redis

- [+] Ganho automático de performance
- [-] Dados opacos e inexistência de índices batem de frente com a arquitetura do sistema
- [-] Necessidade de denormalizar excessivamente os dados em múltiplas estruturas chave/valor para manter a mesma funcionalidade
- FAIL

Estudo de caso - Guia de Cidades Usando Riak

- [-] Sem melhoria de performance
- [-] Não utiliza a maior parte dos diferenciais do produto (não é um sistema "Big Data")
- [-] Mesmas limitações do Redis por ser chave/valor. Mesmo possuindo tratamento interno para JSON, alterações em documentos são "tudo ou nada"
- FAIL

Estudo de caso - Guia de Cidades Usando CouchDB

- [--] Performance pior
- [-] Não utiliza a maior parte dos diferenciais do produto (sistema nunca trabalha offline)
- [+] Integração fácil com ElasticSearch (via river) possibilita buscas avançadas (FTS)
- [-] Sistema de views por map/reduce necessita de uma mudança de paradigma dentro da equipe de devs
- FAIL

Estudo de caso - Guia de Cidades Usando VoltDB

- [+] Ganho significativo de performance
- [+] Mantém estrutura de dados (relacional)
- [-] Limita o volume de dados à RAM do server
- [-] Limitações similares à implementação original (relacional), mas sem problemas com locks e melhor escalabilidade
- FAIL (mas pode ser "good enough")

Estudo de caso - Guia de Cidades Usando MongoDB

- [+] Melhoria significativa de performance
- [+] Arquitetura de dados centrada em documentos (um por empresa)
- [+] Não há necessidade de locks ou transações pois os updates são atômicos (dentro do documento)
- [+] Capacidade de indexar por múltiplos campos por documento, índices geoespaciais
- [+] Buscas ad-hoc performáticas
- [+] Escalabilidade simplificada com replica sets e sharding
- WIN

Estudo de caso: Email em larga escala

- Começou pequeno, cada conta com listas de bloqueio e filtros, armazenadas em MySQL
- Grupos de email com filtros e listas
- Em alguns casos + de 700 queries por mensagem recebida
- Redis para contadores e filtros.
- Cache para regras de recebimento
- Estagios de recebimento: cada fase conhece apenas seus dados
- Queries com condições complexas e resultset breves (max 2 linhas) -> K/V

Estudo de caso - BOVESPA real time

- Em db relacional tradicional explode rapidamente devido à taxa de updates
- Se tirar os índices você grava mais rápido, mas perde a visibilidade nas buscas
- Volume de dados armazenados não é alto, mas as consultas, inserções e updates devem ter latência baixíssima (hint: cabe em memória)

Estudo de caso - BOVESPA real time Usando Riak

- [-] Explosão de vector clocks, alto consumo de espaço em disco (compactação é eventual), headers inflam
- [-] Menor throughput e menor performance no decorrer do dia (após merge e compactação recupera)
- [-] Buscas limitadas e caras
- [+] Melhorias recentes (secondary indexes, eleveldb) aliviam esses problemas
- FAIL

Estudo de caso - BOVESPA real time Usando CouchDB

- [-] Performance de writes e atualização de views inviabiliza o uso
- FAIL

Estudo de caso - BOVESPA real time Usando MongoDB

- [-] Para conseguir a performance de escrita necessária, tem que ser em modo "fire and forget", abrindo mão de consistência e arriscando perda de dados (i.e. qq falha de write passa desapercebida)
- [-] Mesmo admitindo essas limitações, pode não ser performático o suficiente e os writes são centralizados no master (i.e. para escalar writes tem que implementar sharding mesmo que o volume de dados persistentes não justifique)
- FAIL

Estudo de caso - BOVESPA real time Usando VoltDB ou Redis

- [+] Gravação em memória - updates não custam quase nada
- [+] Latência insignificante permite serialização de operações (single threaded), que eliminam contenção por locks
- [+] Persistência garantida por replicação e AOF assíncrono
- Bonus para o Redis por ser chave/valor, pois a busca padrão é por chave (ticker)
- Bonus para o Redis pelos seus data type complexos (lists para timelines, sorted sets para rankeamento)
- WIN (Redis++)

Calcanhares de Aquiles

- Redis: administração em cloud (ainda não tem failover automático nem clustering)
- MongoDB: desaconselhável para Big Data pois a arquitetura do cluster é heterogênea
- Riak: não se comporta bem em ambiente OLTP com muitos updates/deletes, devido à ênfase em sistemas distribuídos (tombstones duram 3s, vector clock explosion)
- CouchDB: não atinge os mesmos níveis de performance de outros NoSQL (foco é em funcionamento offline e conflict resolution)
- VoltDB: necessita de baixa latência intra-cluster para ser performático

FUQ - Frequent Unasked Questions Choose 2

a) como importo dados existentes ?
b) como exporto meus dados ?
c) se acabar a energia, como me recupero ?
d) se é tão rapido, deve gravar na memória. se grava na memória, como coloco um banco maior do que minha memória ?
e) quem me ajuda se tudo der errado ?
f) se posso usar mais de um nó, o que eu ganho ? escalabilidade ou disponibilidade ?
g) e o backup ? e o restore ?
h) migrando dados e backing stores
i) deploy no cloud - com relacional?
j) facilidades através de vendors (hosted solutions) vs. lock-in
m) dá pra processar big data sem usar NoSQL?
n) existe processamento assíncrono com locks? (transações assíncronas)(lamport clock?vector clock ? document db (JSON))

L'Esprit de L'Escalier

"The feeling you get after leaving a conversation, when you think of all the things you should have said"

ou

"Me zoaram porque meu projeto está cheio de anti-patterns, a performance não está dentro do esperado, e ainda falei que ia escalar e não escala" + assista essa palestra = FUCK YEAH!