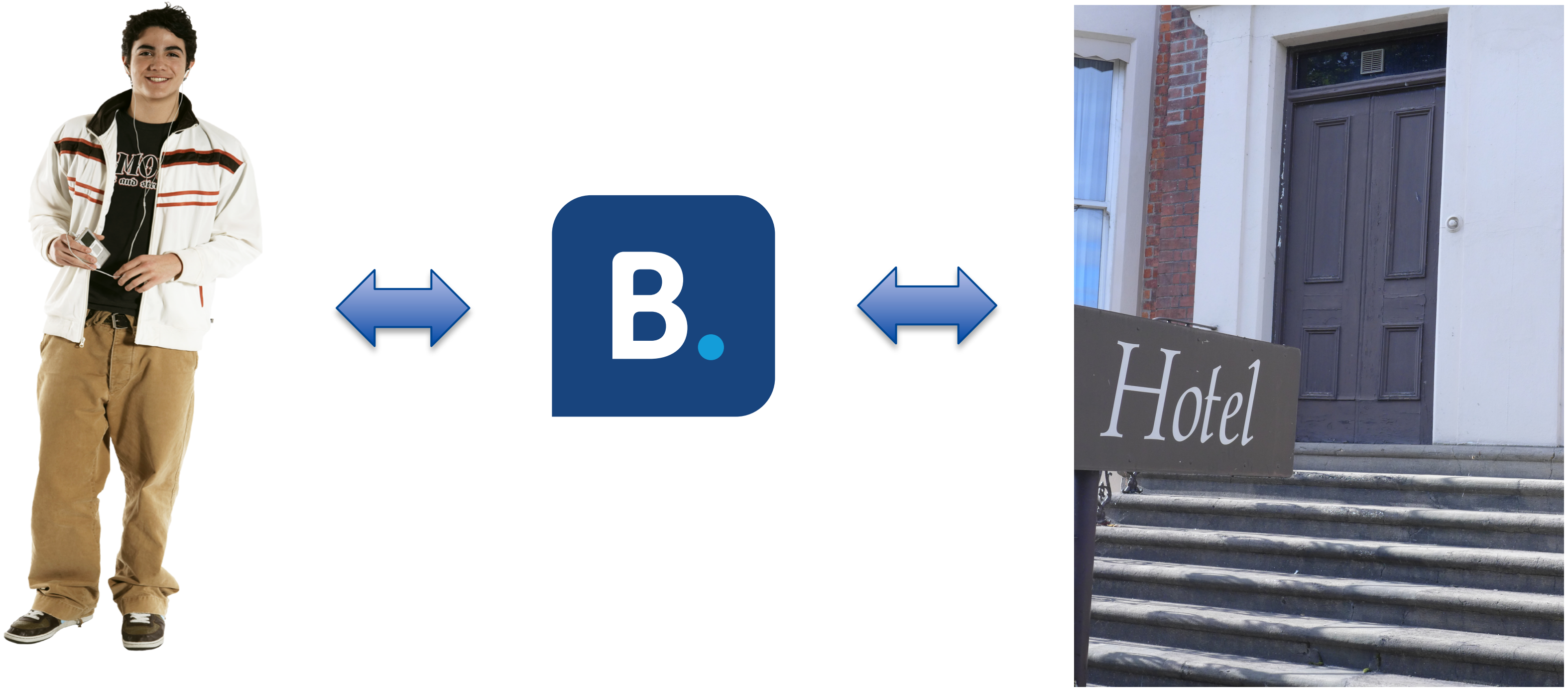
Booking.com

Booking.com:
Evolution of MySQL System Design

Nicolai Plum <nicolai.plum@booking.com>

Booking.com



Early days

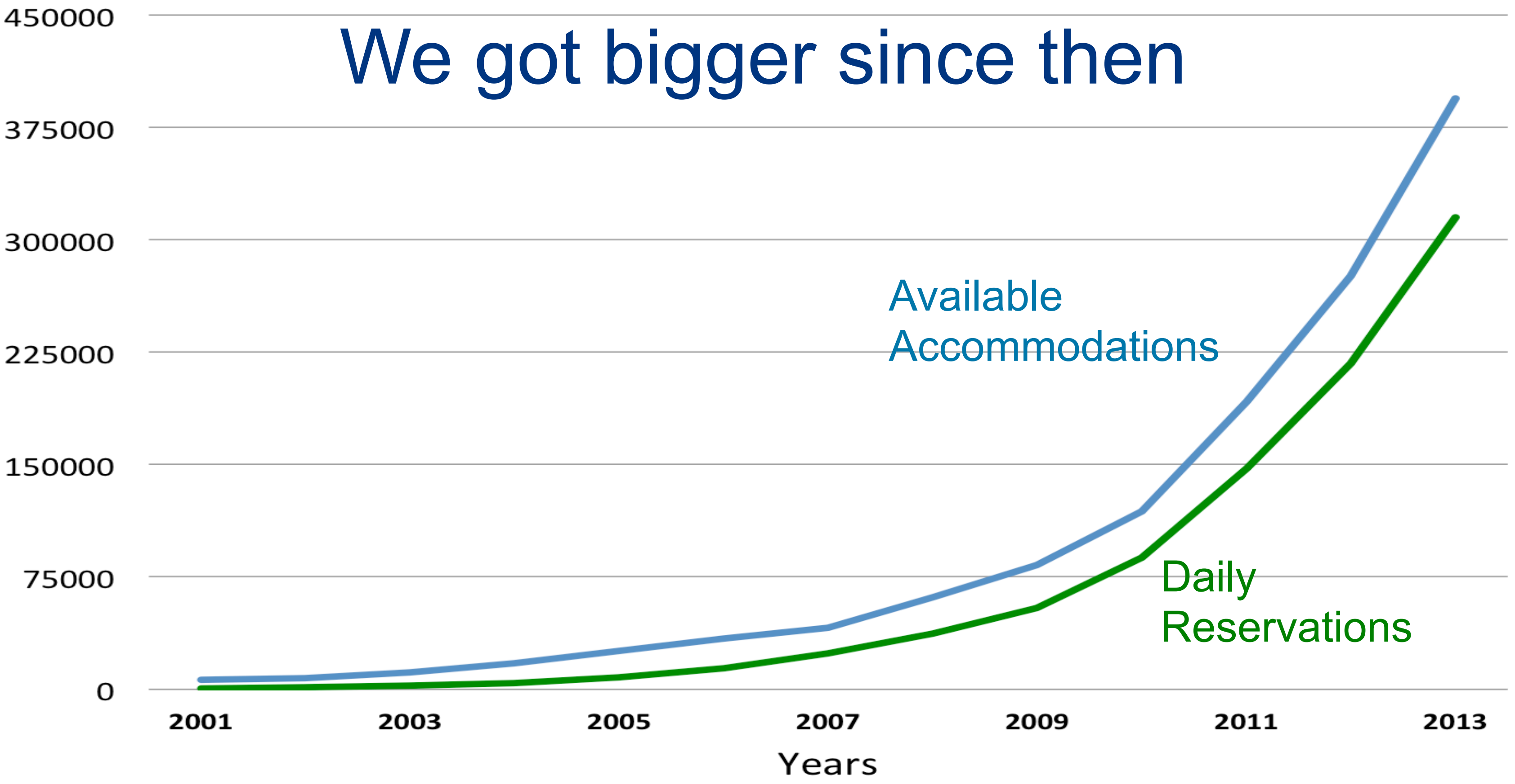
- Founded in 1996
    - as bookingsportal.nl
- Purchased by Priceline.com Inc in 2005
- Became Booking.com in 2006

Still small in 1999…

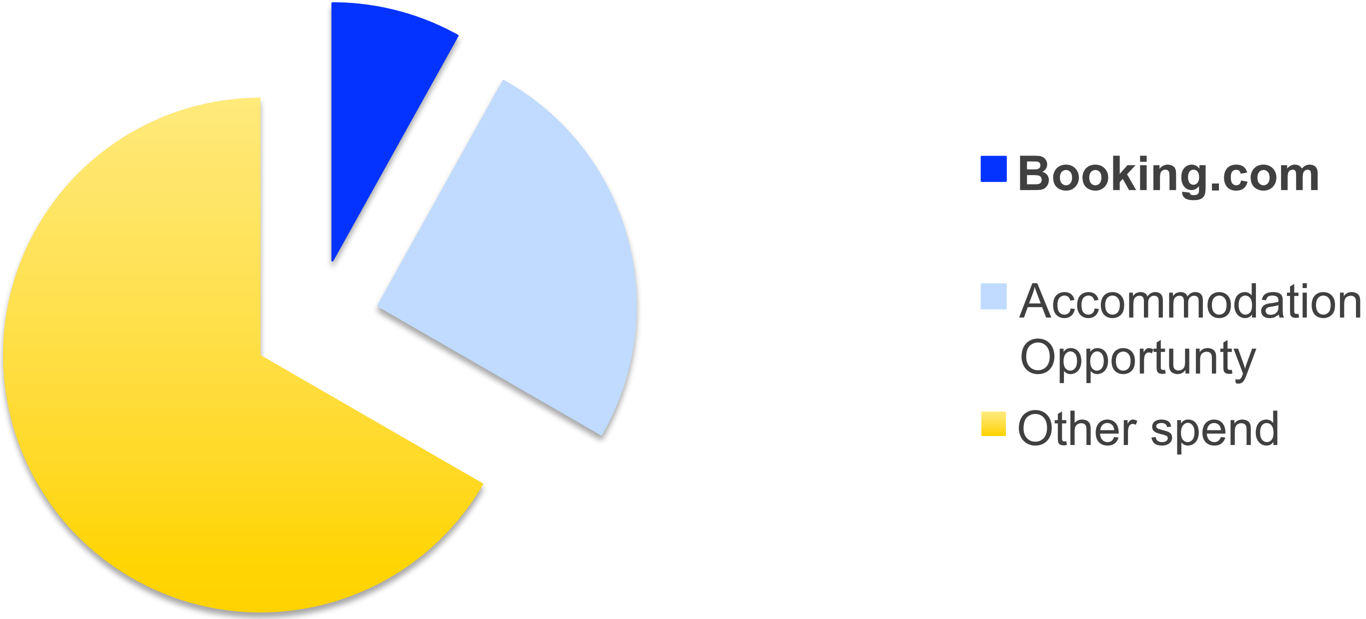
Picture by Geert-Jan Bruinsma

Booking.com

We got bigger since then



Travel business opportunity



Architecture decisions

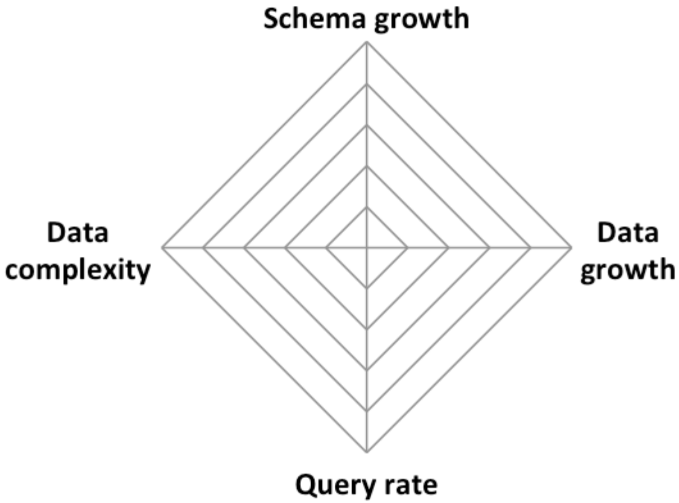
- Around 2003 we decided
  - Keep using Perl
  - MySQL + replication
  - Analytics and dashboards
  - A/B testing

Hotel reservation website design experts

Picture by SantiMB.Photos on flickr; license cc-by-na

Scalability dimensions

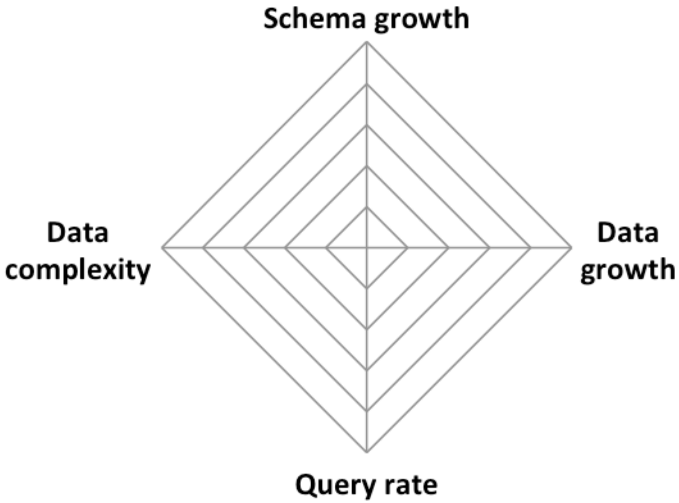
- Schema growth
- Data growth
- Query rate
- Data complexity



Each has different solutions

Scalability dimensions

- Schema growth
- Data growth
- Query rate
- **Data complexity**



Each has different solutions

Data complexity

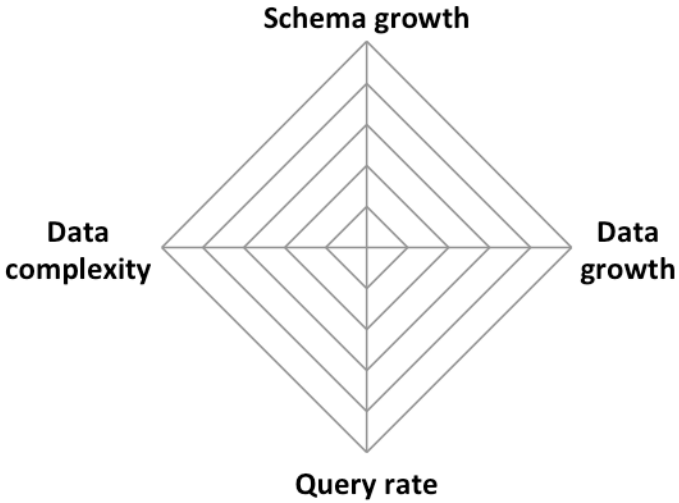
- Complex multi-directional relations and normalisation
- Many-way JOINs
- Foreign Key constraints
- All put stress on…
    - SQL Optimiser query plans
    - Storage engines
    - Schema design
    - **Developers**
    - **DBAs**

Data complexity reduction

- Prefer client-side logic to Foreign Keys and Stored Procedures
  - Client-side scales better in CPU
  - We have control of all our code
- Prefer simpler joins
- Denormalise pragmatically
- Fast schema changes
  - Online schema change, low bureaucracy

Scalability dimensions

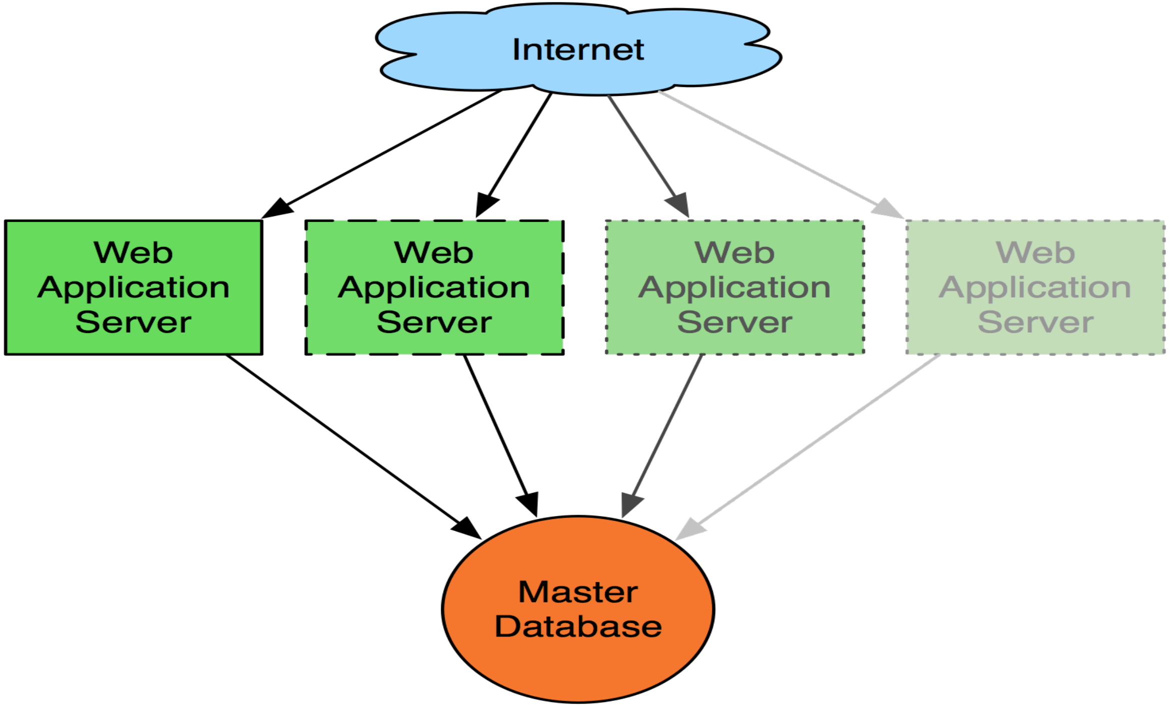
- Schema growth
- Data growth
- **Query rate**
- Data complexity



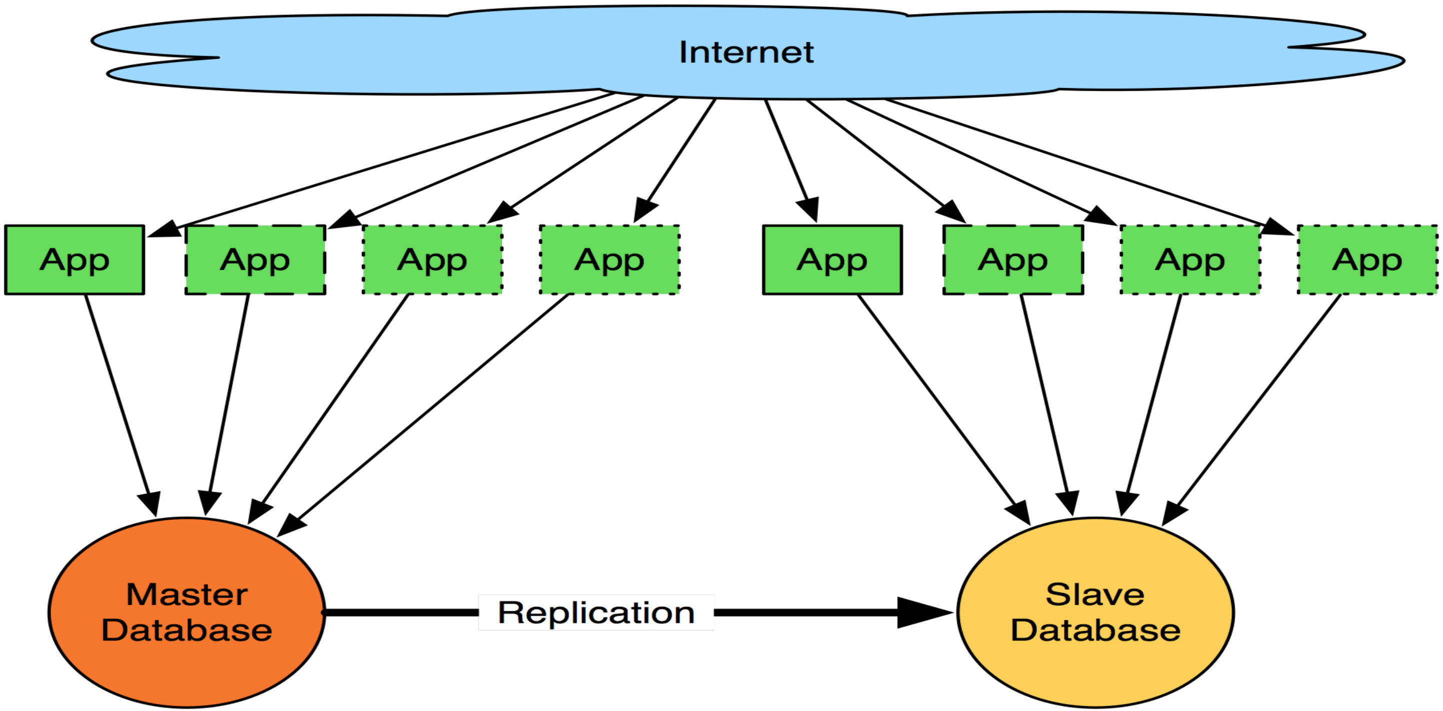
Query rate

- Travel websites are all read-intensive
- Replication for the win!
  - Winning for us since 2003
- How to monitor and manage?
  - Puppet, Graphite, Nagios, etc
- Comprehensive application event and error analysis

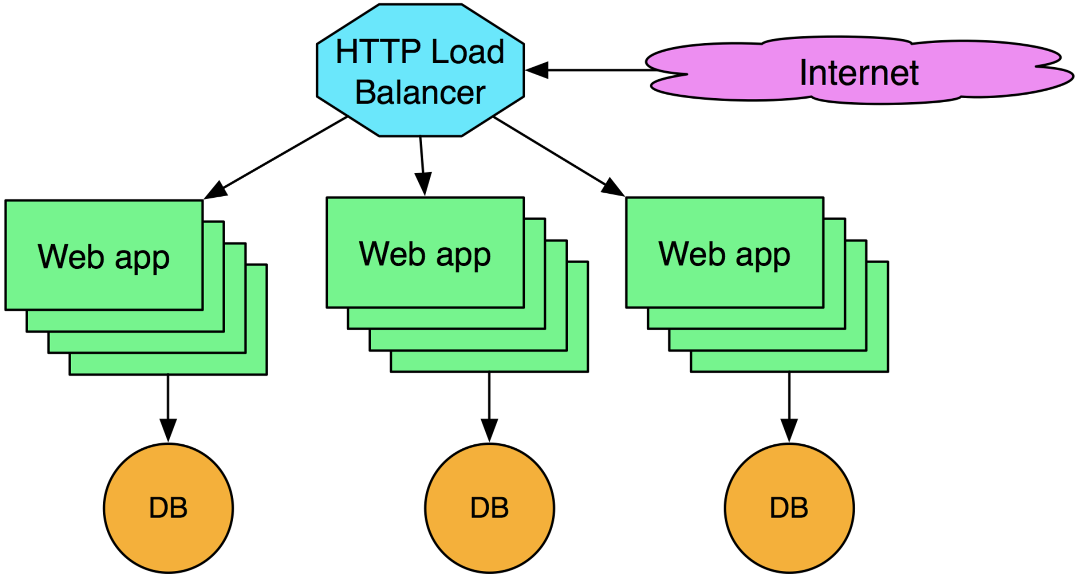
Databases - beginning



Database replication



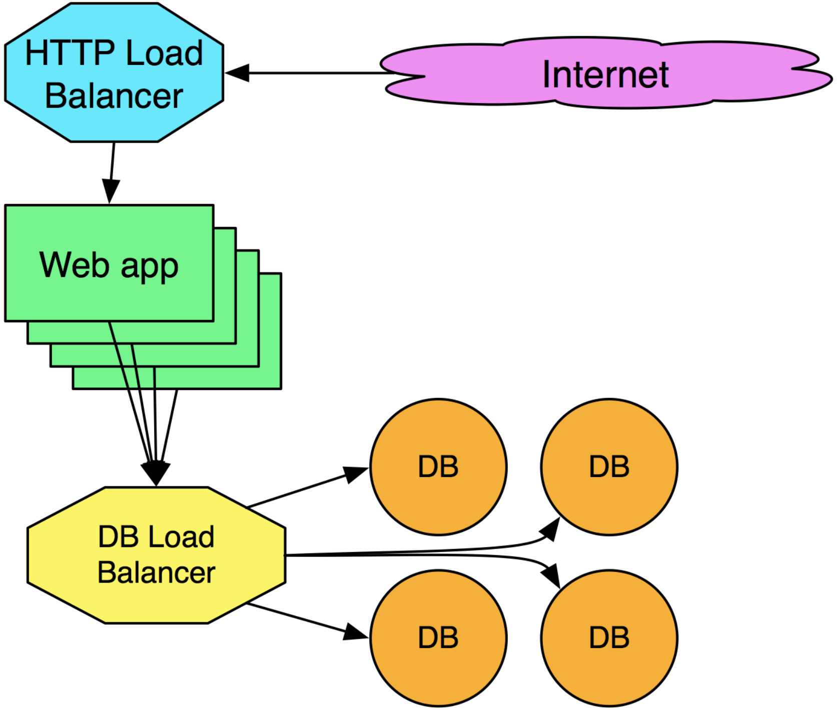
Sharing databases – Cells



Sharing databases – Cells

- Simple to administer
- Good failure isolation
- Poor efficiency
  - Even worse with many schemas

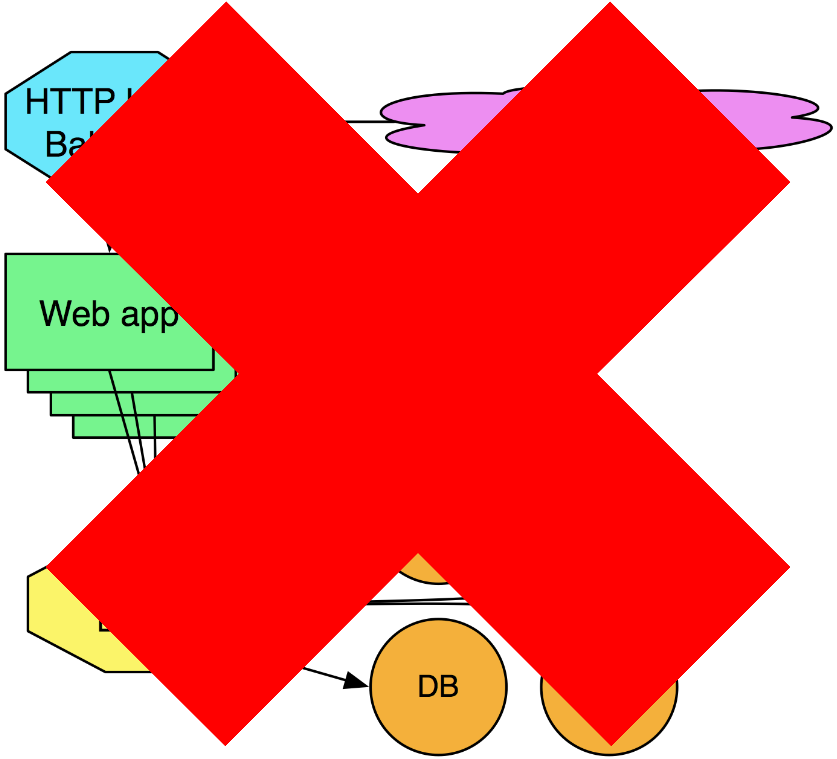
Use a Load Balancer !



Use a Load Balancer!

- Network stress
- Single Point Of Failure
- Scalability nightmare

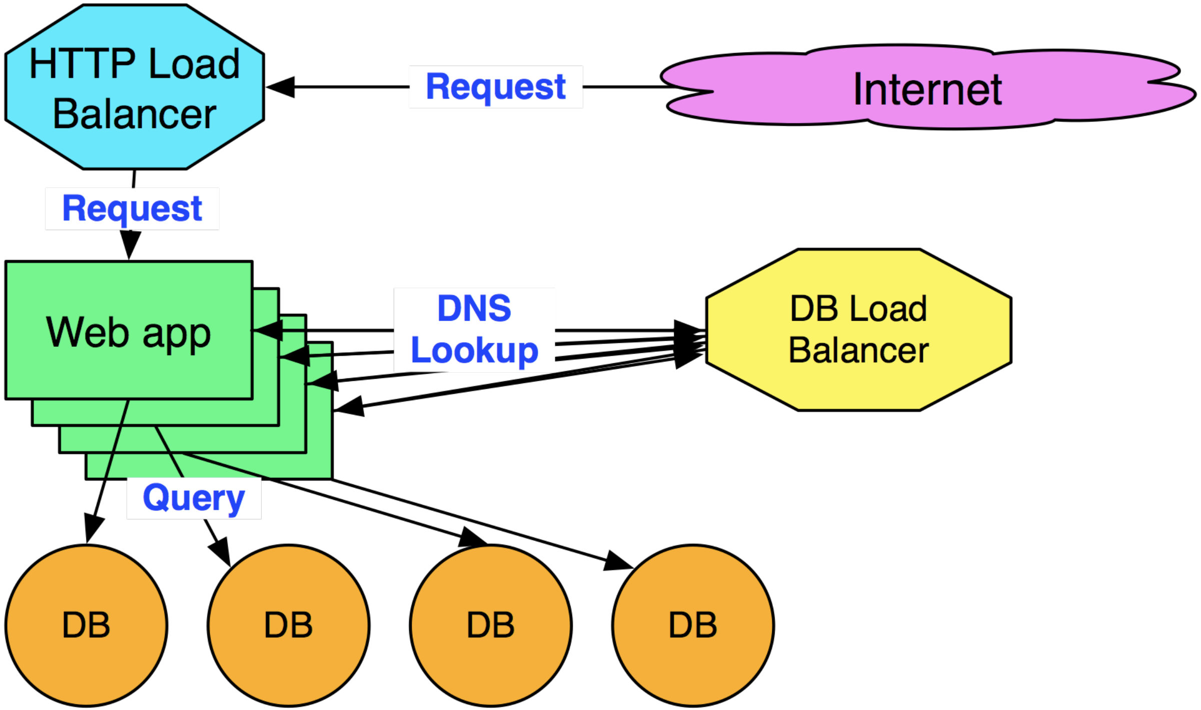
Use a Load Balancer!



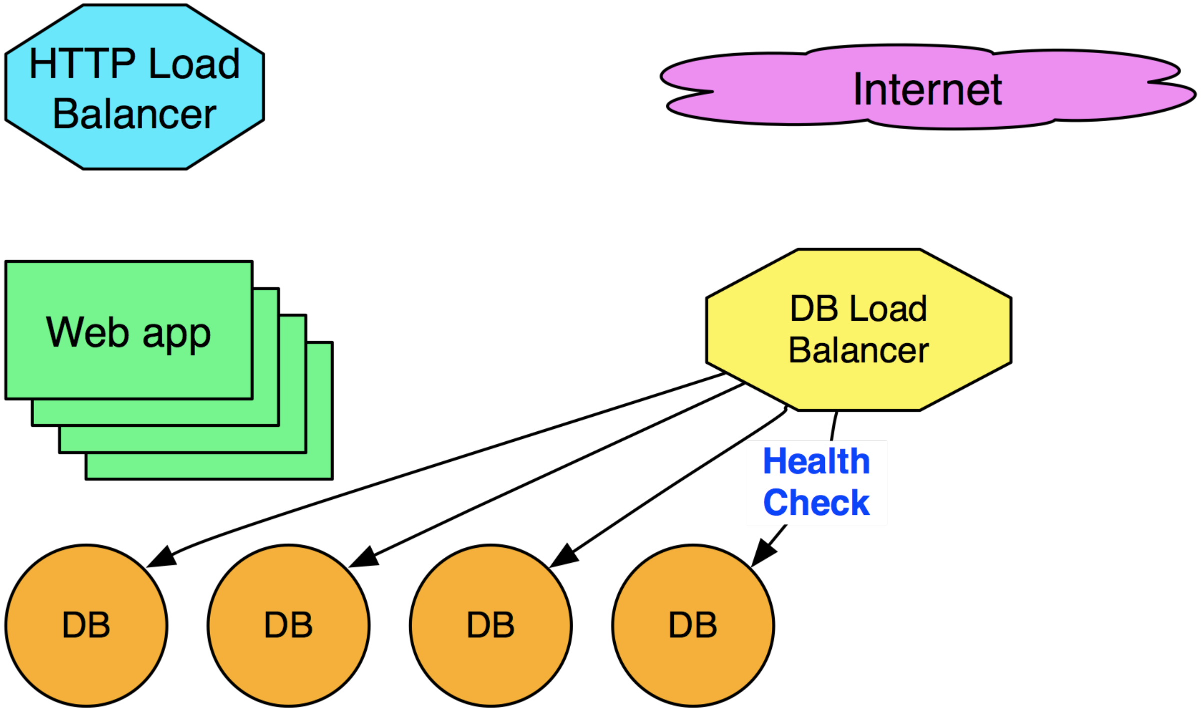
DNS Database Load Balancer

- Separate control and signal path
- Modified HAProxy
  - Standard HAProxy MySQL healthcheck
  - HAProxy tracks server availability
  - Returns list of severs in DNS query

DNS Database Load Balancer



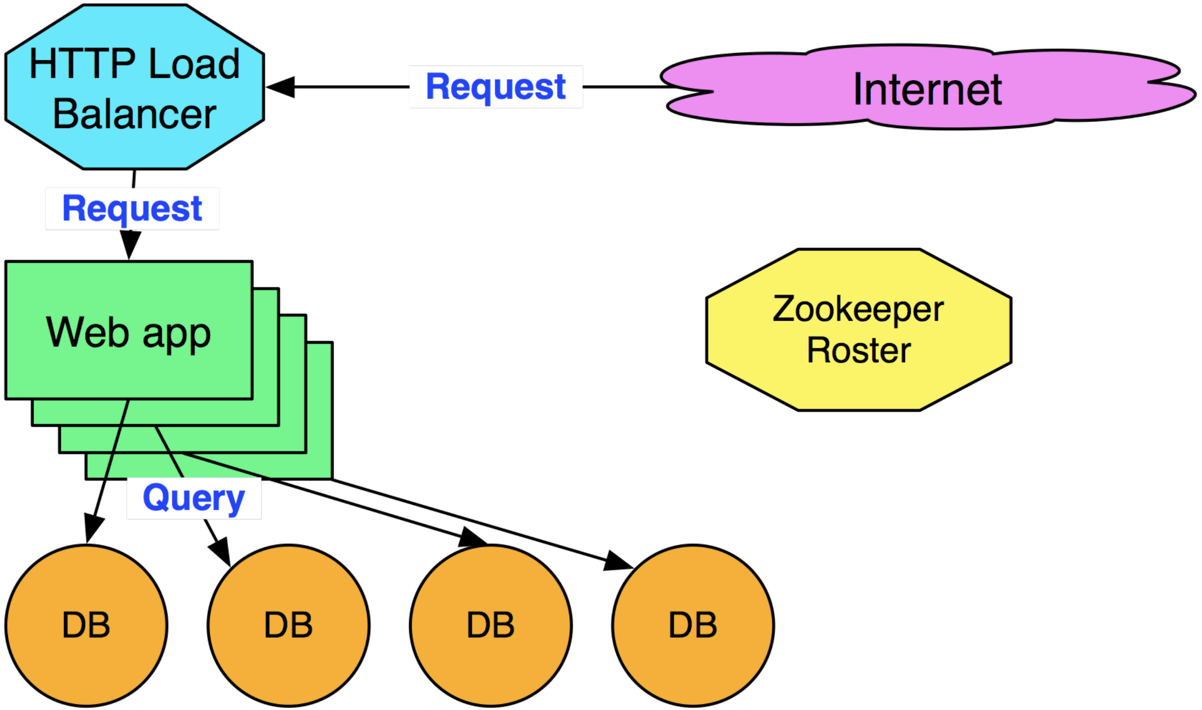
DNS Database Load Balancer



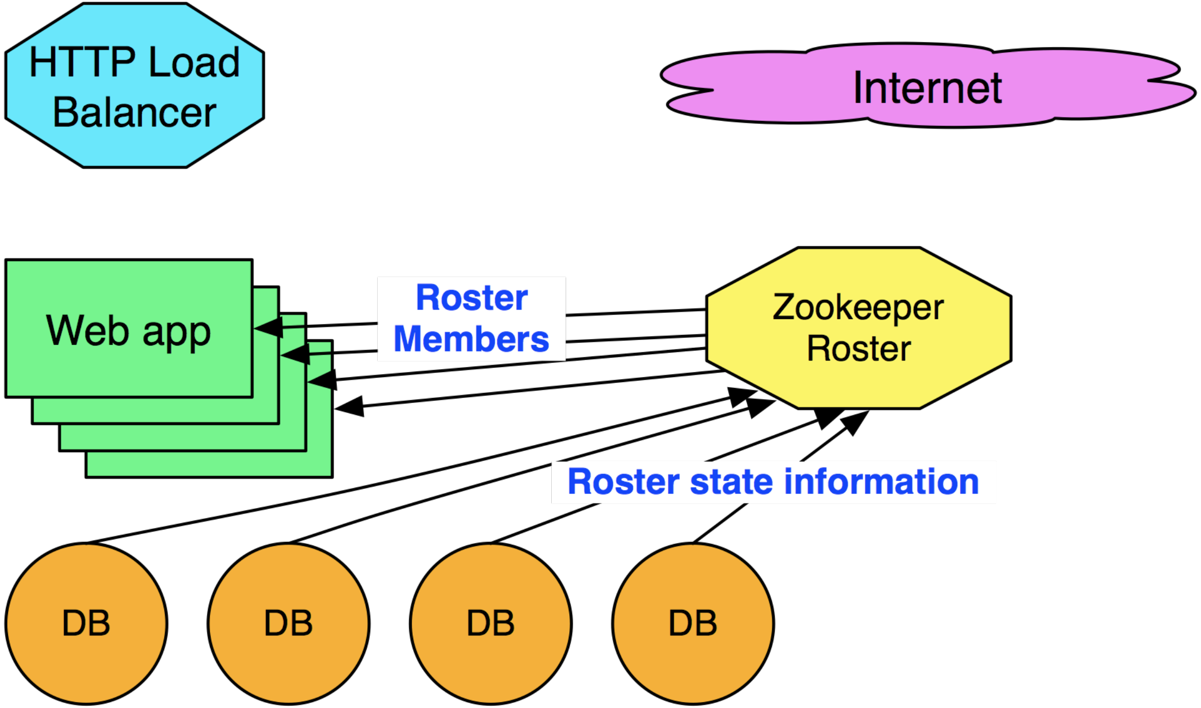
Rosters of eligible DB servers

- Separate control and signal path
- De-centralised service checks
- Apache Zookeeper
  - Pools of available servers
- ZooAnimal deamon registers available database servers
- ZooRoster deamon retrieves servers for clients

Rosters of eligible DB servers



Rosters of eligible DB servers



Reliability

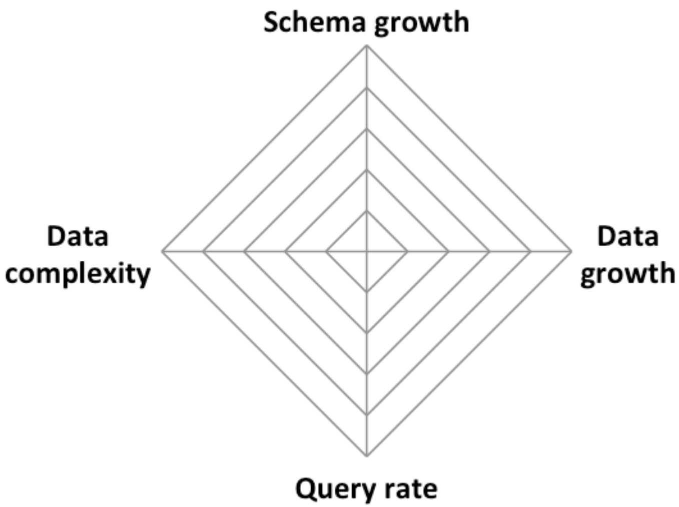
- Cells
  - Strong failure isolation, inflexible
- DNS LB
  - Less failure isolation, more flexible, LB is scaling and reliability problem
- Rosters
  - Less failure isolation, more flexible, very scalable, Zookeper very reliable

Replication

- Speed challenges
  - Single threading hurts us
  - Especially on a SAN
  - Careful optimisation of bulk jobs
- Binlog server, make it all faster
  - Some help with failover
- Bodge: copy tables
  - Works on myisam
  - Needs transportable tables for InnoDB
  - Alter tables from innodb to myisam, copy

Scalability dimensions

- **Schema growth**
- Data growth
- Query rate
- Data complexity



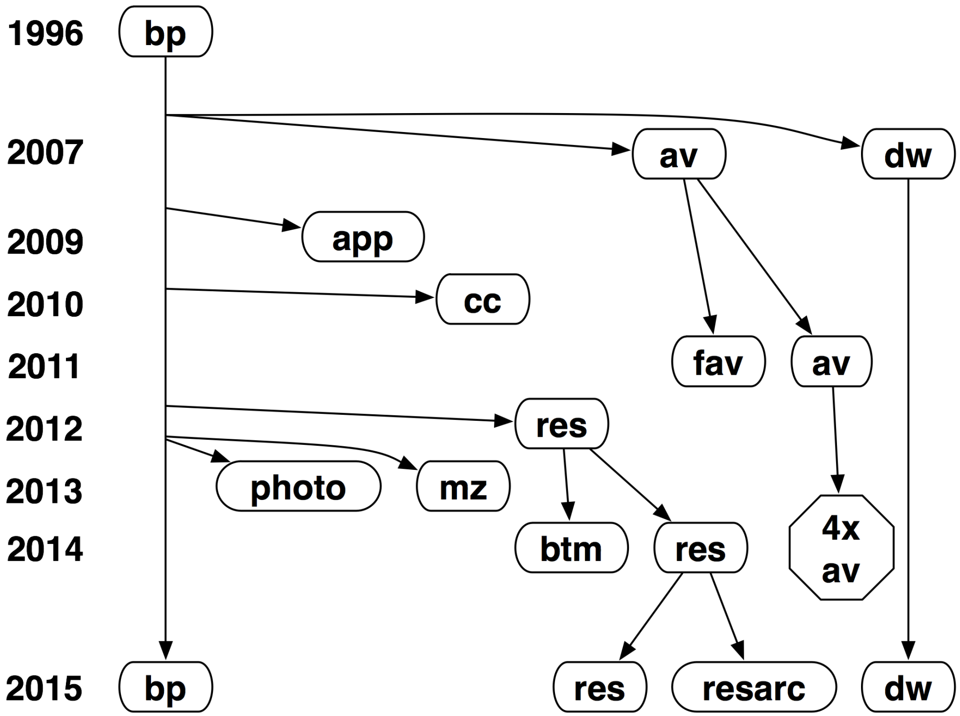
Acommodation reservation data

- Accommodation catalogue
    - Descriptions, amenities, policies
- Inventory
    - Room prices, quantities and restrictions
- Customer details
    - Names, contact info, payment
- Different growth and use patterns

Multiple schemas

- Split data by function
- Keeps it simple for most developers
  - Queries against single schema
- Keeps it simple for DBAs
- Less simple for infrastructure developers
  - ORM changes, data pumps, consistency checks
  - Just feed them more coffee…

Multiple schemas



Multiple schemas

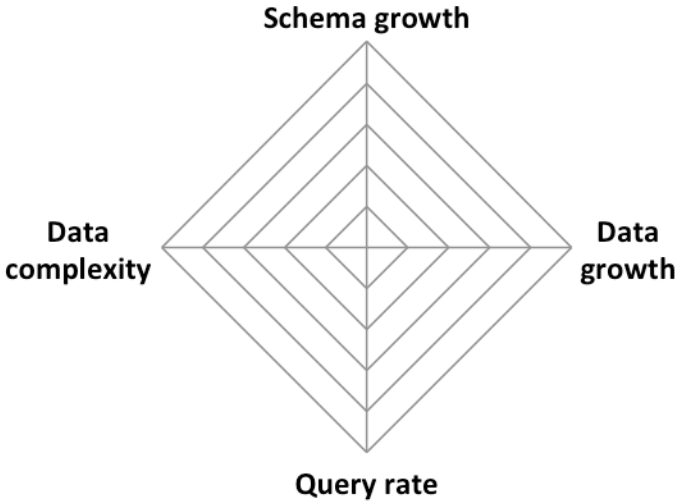
- Consistency…
  - Distributed transactions, XA = pain
  - DB failures, code bugs, app server crashes
  - Careful order of updates so critical things last
  - Consistency check references later
- Requires skilled developers and strong code knowledge
- APIs and ORM layers help

When to split?

- Analysis tools for busiest tables
    - Performance_Schema and SYS Schema
- Business impacts, development time
- Isolate critical functions from complex, less critical functions

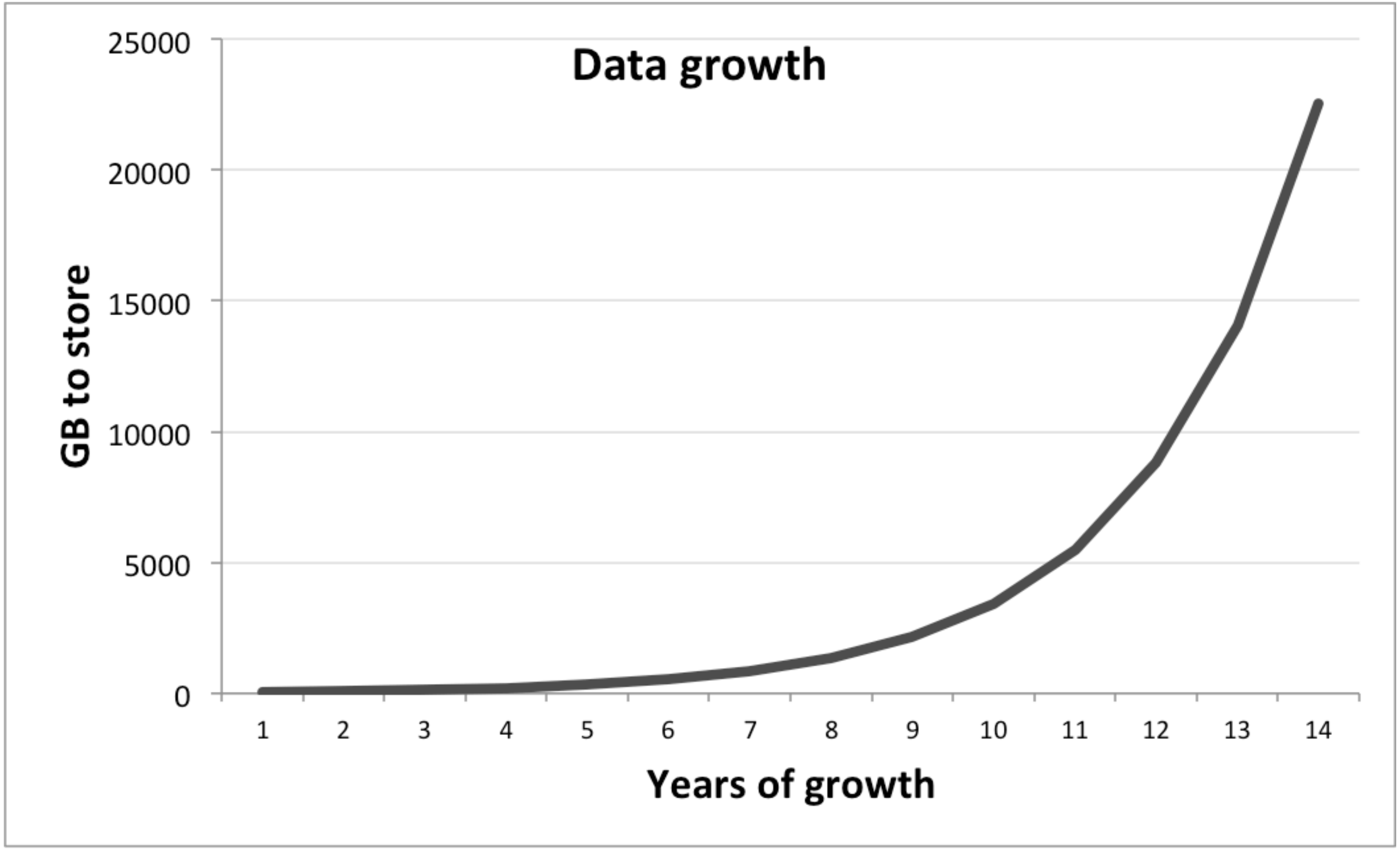
Scalability dimensions

- Schema growth
- **Data growth**
- Query rate
- Data complexity

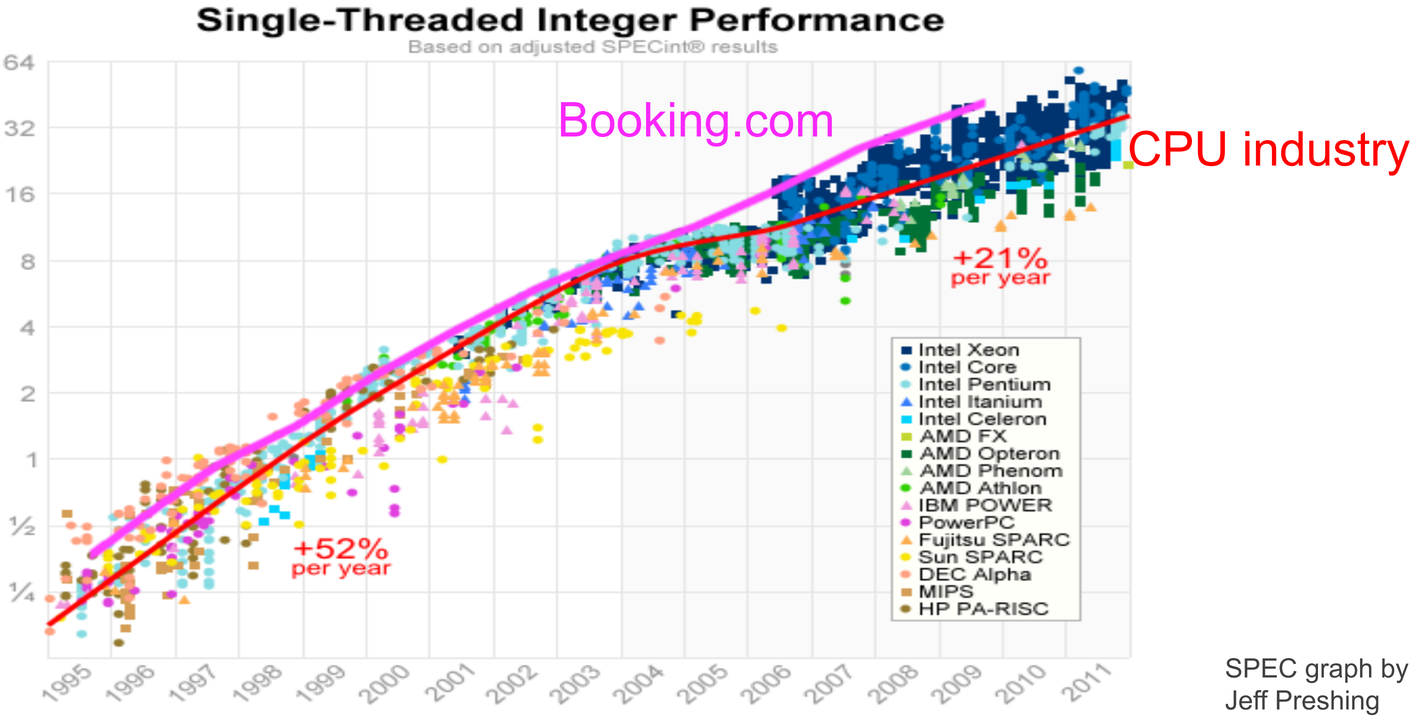


Data growth

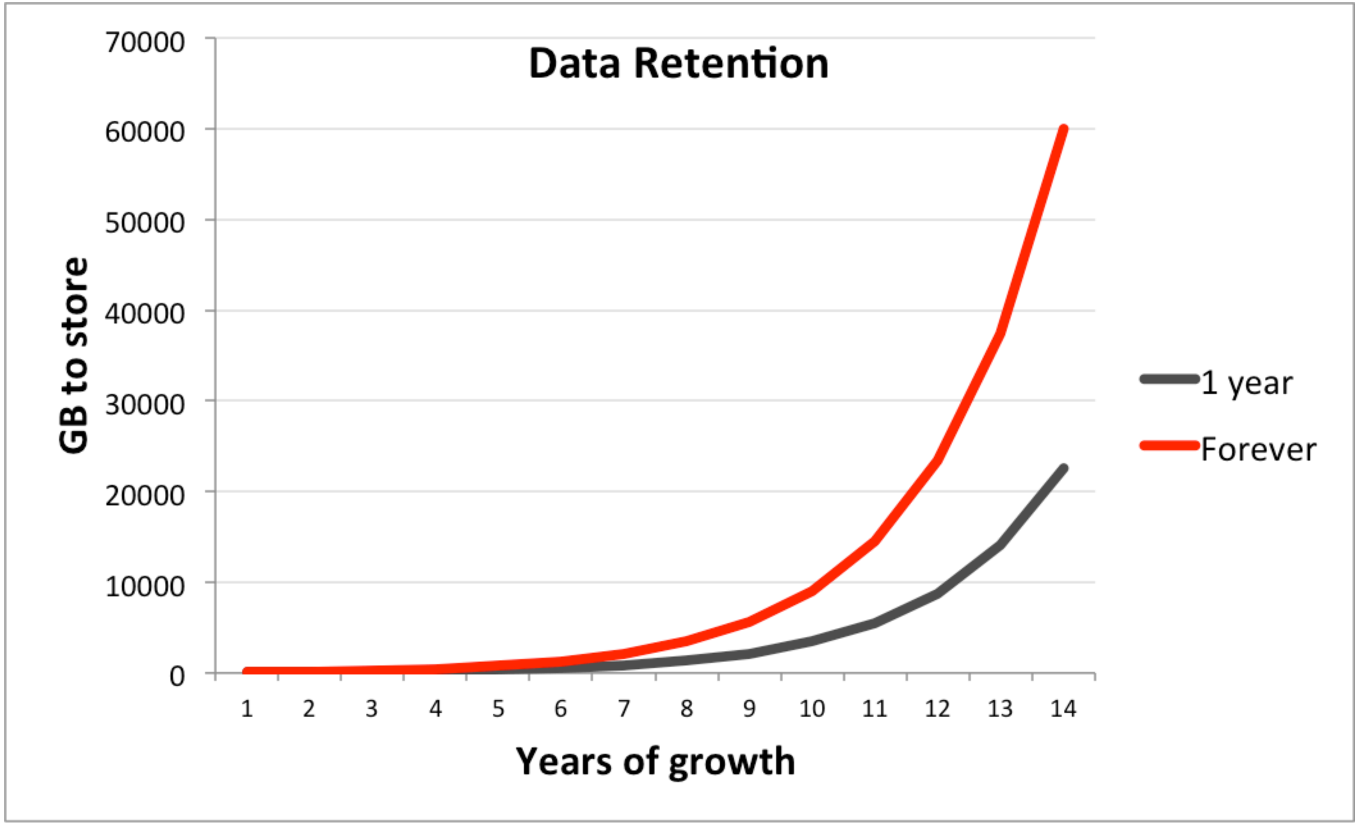
- Business growth 30-50% annually
- Data growth 40-60% annually
- Faster than Moore’s Law
  - And disk IOPS


Data growth
GB to store
25000
20000
15000
10000
5000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Years of growth

We outgrow CPU speed



SPEC graph by
Jeff Preshing


Data Retention
GB to store
70000
60000
50000
40000
30000
20000
10000
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
Years of growth
1 year
Forever

Database growing pains

Dataset size exceeds memory	Read performance decreases (a lot)
Dataset size exceeds local disc size	SAN latency, management, cost Write perf decreases, Read perf decreases more
Dataset exceeds size a CPU can scan in reasonable time	Ad-hoc queries are impossible, analyse table difficult, schema changes difficult, table scans lethal
Dataset exceeds storage volume size, disc array size, backup capacity, filesystem limits	Totally unmanageable. Give up!

Database growing pains

Dataset size exceeds memory	Read performance decreases (a lot)	~200GB
Dataset size exceeds local disc size	SAN latency, cost, complexity Write perf decreases, Read perf decreases more	~5TB
Dataset exceeds size a CPU can scan in reasonable time	Ad-hoc queries are impossible, analyse table difficult, schema changes difficult, table scans lethal	~20TB
Dataset exceeds storage volume size, disc array size, backup capacity, filesystem limits	Totally unmanageable. Give up!	~300TB

Archives

- Separate transcational and analytical
  - Store the past in another schema
- File off payment, PII where possible
  - Also shrinks dataset
  - Win-win ☺
- … but you need more (later)

Materialisation and data models

- Read-optimised is not write-optimised
  - OLTP vs OLAP – the timeless struggle
- Two schemas
- Different read and write data models
- Data pumps, materialisation queues
- Inevitably more complex
- Needs smarter infrastructure to keep feature development easy

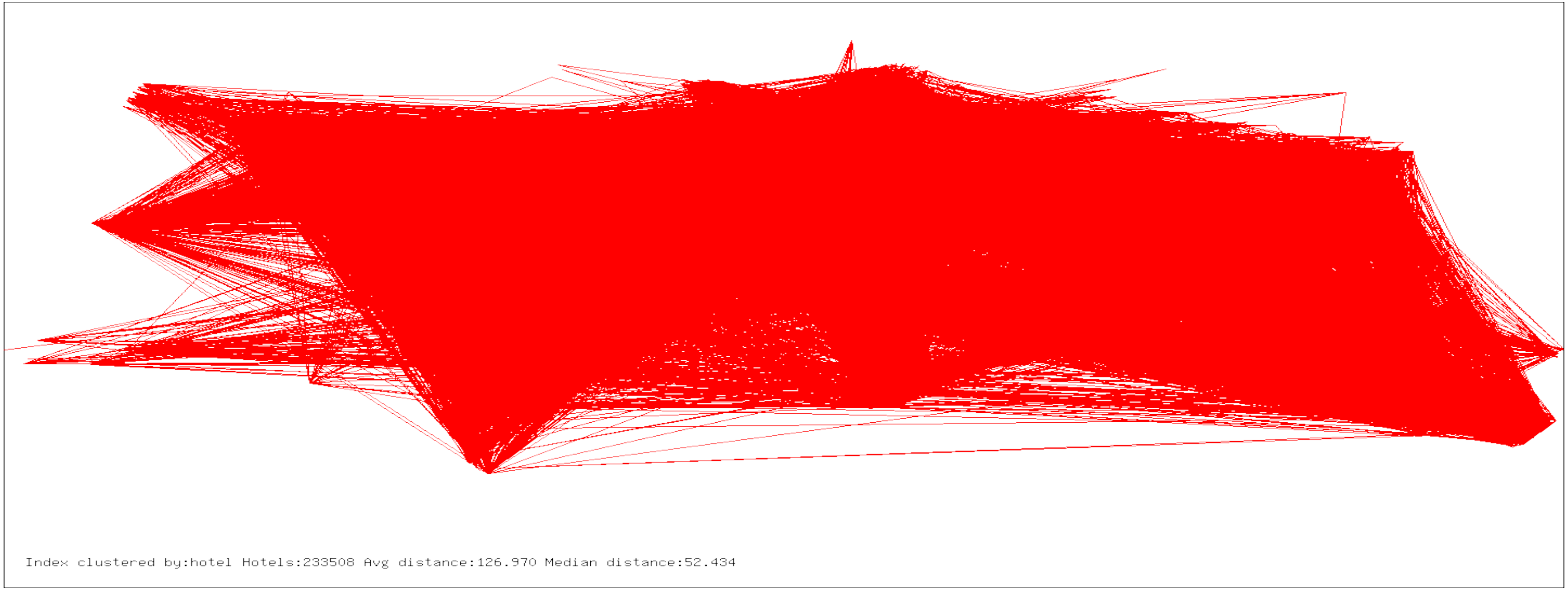
Inventory – first materialisation

- Flat availability
- Write
  - Complex relational structure of rooms, rates, restrictions
- Read
  - Simple point query for inventory for a single stay
- Much more predictable than caching

Row index – Hotel ID order

Hotel_ID	District	City UFI	Country
1	Kensington	London	England
2	Chaoyang	Beijing	China
…	…	…	…
20000	La Défense	Paris	France
20001	Dongchen	Beijing	China
20002	TriBeCa	New York	USA
….			
35678	Xicheng	Beijing	China
35679	Gentofte	København	Danmark
….			
70035	Chaoyang	Beijing	China

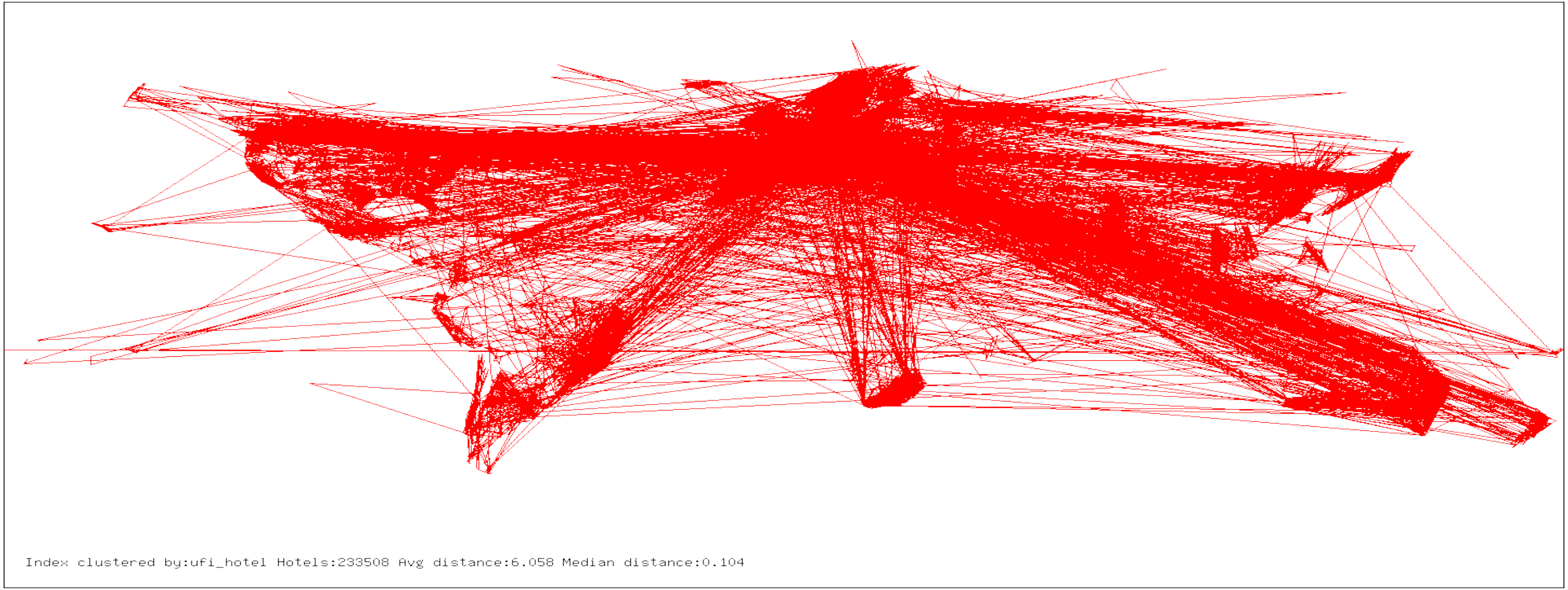
Row indexing – Hotel ID order



Row index – UFI order

Hotel_ID	District	City UFI	Country
1	Kensington	London	England
…			
70035	Chaoyang	Beijing	China
35678	Xicheng	Beijing	China
2	Chaoyang	Beijing	China
20001	Dongchen	Beijing	China
…			
20002	TriBeCa	New York	USA
20000	La Défense	Paris	France
…			
35679	Gentofte	København	Danmark

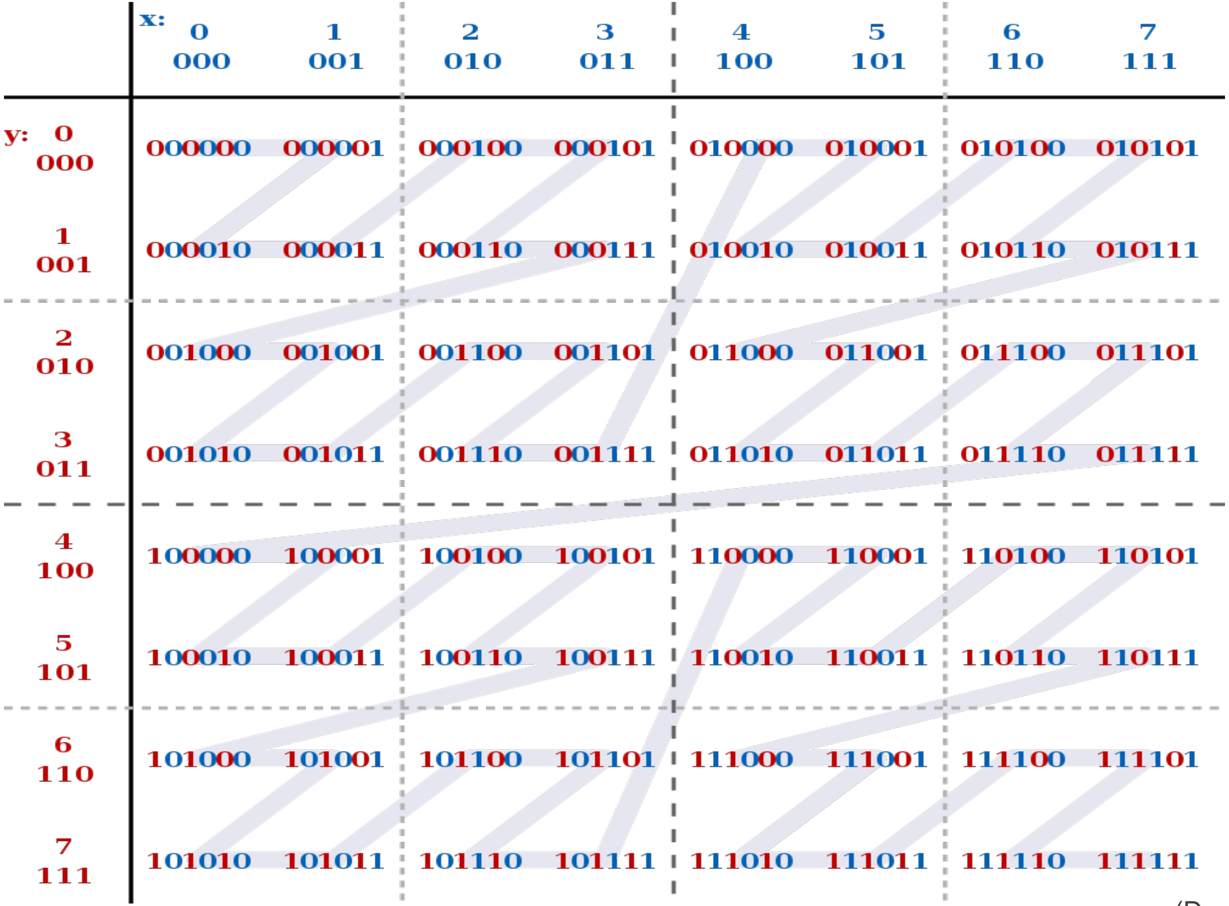
Row indexing – UFI order



```
Index clustered by:ufi_hotel Hotels:233508 Avg distance:6.058 Median distance:0.104
```

Location coding - Z-order curve

- Location latitude and longitude
- 12 bits is
  - 10km longitude
  - 6-10km latitude (for most hotels)
- Index with bitwise interleave of latitude and longitude in a **space-filling curve**



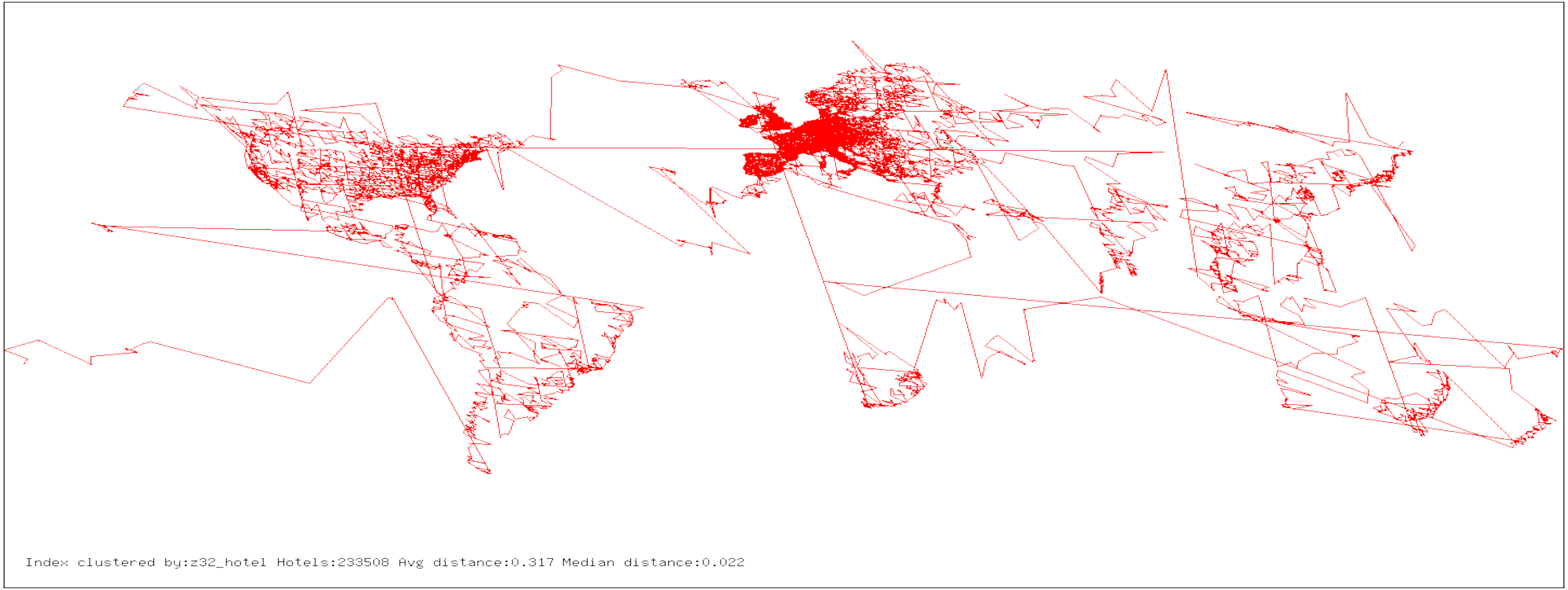
	x: 0 000	1 001	2 010	3 011	4 100	5 101	6 110	7 111
y: 0 000	000000	000001	000100	000101	010000	010001	010100	010101
1 001	000010	000011	000110	000111	010010	010011	010110	010111
2 010	001000	001001	001100	001101	011000	011001	011100	011101
3 011	001010	001011	001110	001111	011010	011011	011110	011111
4 100	100000	100001	100100	100101	110000	110001	110100	110101
5 101	100010	100011	100110	100111	110010	110011	110110	110111
6 110	101000	101001	101100	101101	111000	111001	111100	111101
7 111	101010	101011	101110	101111	111010	111011	111110	111111

(David Eppstein, via Wikipedia)

Row index – Z-order

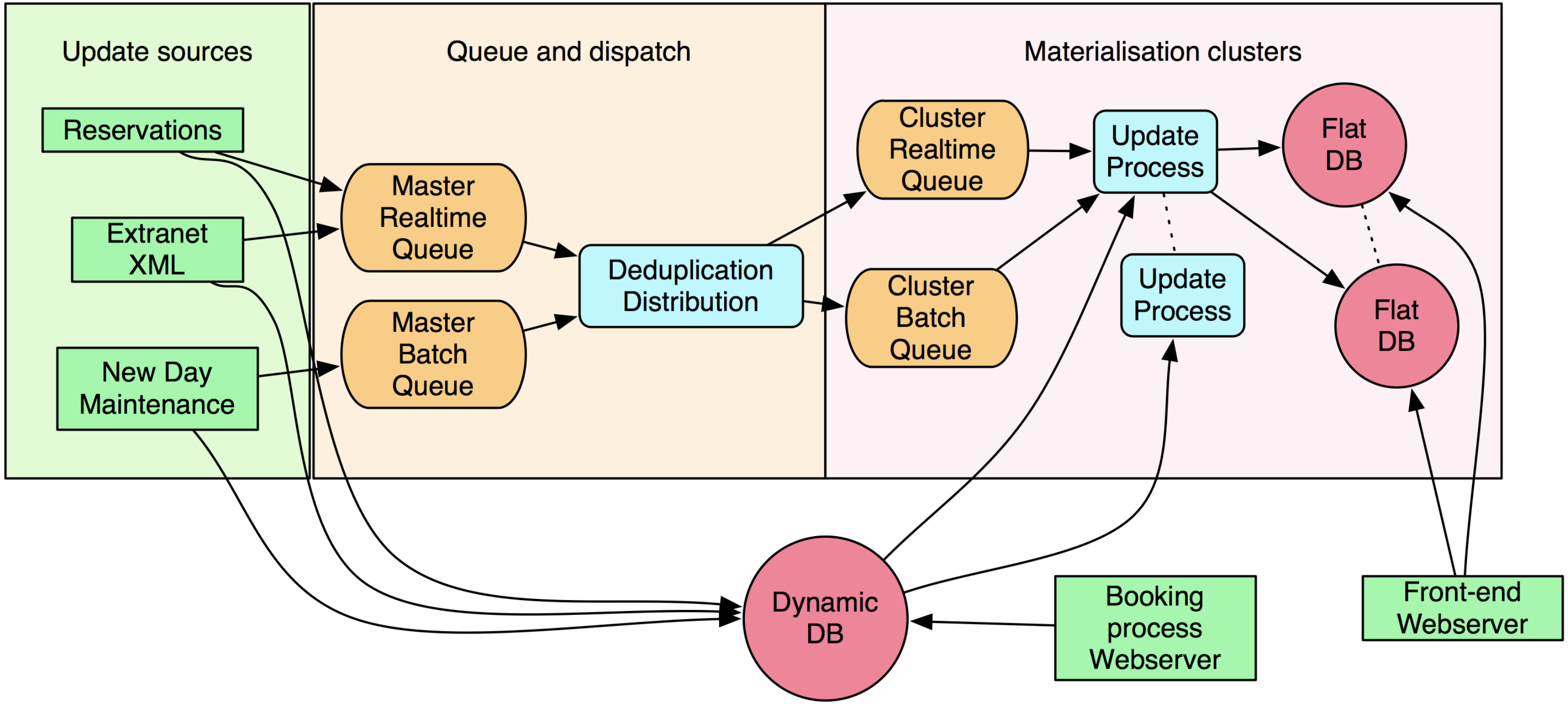
Hotel_ID	District	City UFI	Country	Z-location
1	Kensington	London	England	3456789
…				
35678	Xicheng	Beijing	China	6788567
70035	Chaoyang	Beijing	China	6789456
2	Chaoyang	Beijing	China	6789456
20001	Dongchen	Beijing	China	6790463
…				
20002	TriBeCa	New York	USA	8534535
20000	La Défense	Paris	France	10013346
…				
35679	Gentofte	København	Danmark	13036743

Row indexing Z-order curve



```
Index clustered by:z32_hotel Hotels:233508 Avg distance:0.317 Median distance:0.022
```

Materialised inventory



Booking.com

Sharding

- Prefer schema split to sharding
- Necessary in growing, busy, transactional schemas
  - Inventory
  - Materialised datasets
- Requires good API (or developer awareness)
  - Complexity, overhead

Analytics

- Two types of analytics
- Exploitation
  - Canned reports with some parameters exploited many staff
- Exploration
  - New queries, unknown unknowns

Analytics – exploitation

- Pre-prepared reports - Controlrooms
- Part pre-aggregated data
  - Intermediate infrastructure between raw data and single purpose report data
  - Satisfies need for regular reports, common questions
    - Fixed reports with parameters
  - Less flexible for ad-hoc queries
  - Technical dead-end, useful for medium-term
- Moving to Hadoop

Analytics – exploration

- Surprisingly easy to make a write only dataset in various ways
  - Too big to query
  - Queries hit performance too hard
  - Can’t add indexes so queries hit too hard
  - Users too bad at SQL (Excel/ODBC) so they give up

Analytics - exploration

- Need constant compute power per unit of data during growth
- Data to MySQL and Hadoop
- Hadoop answers in linear time
  - but not quickly
- Business analysts love it
  - Most people need a friendly interface

Business systems

- Web marketing
  - More traditional database
  - Large imports, ETL
  - >20TB MySQL
    - Even with split schemas
  - Analysis is moving to Hadoop

Masters

- First DRBD
- Now SAN Netapp filers
- Future is trying to reduce number of important machines
- Now: rapid automated failovers
- SAN == safety + latency
- For arbitrary topology changes, need a global way to identify of changes (transactions)
  - GTID
- Future: (pseudo) gtid, no special masters

Measuring capacity

- In fixed config cells, you need more DB than app
- In flexible pools, capacity is hard
  - Metrics lie, due to nonlinearity in the database
  - Qps, etc, help a bit
  - Traffic replay at high rate helps more (if you can)
  - Replication capacity is also important
    - Single thread, often a limit
    - Stop slave and measure time to catch up
    - P_S replication stats too complicated in 5.6
    - Contention and non-linearity really hard

Abstraction Layers

- No need for a full microservice intercommunication framework architecture standardisation committee…
- Just a function call will do
- Inventory was easy
  - Few calls to well defined API functions
- Search was not
  - Search: everyone fetched hotels and filtered themselves even for common searches

?

nicolai.plum@booking.com