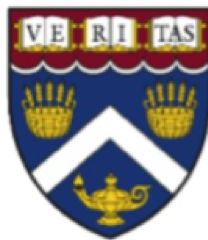


CSCI E-88 Principles Of Big Data Processing

Harvard University Extension, Fall 2019

Marina Popova



Lecture 1 - Introduction

@Marina Popova

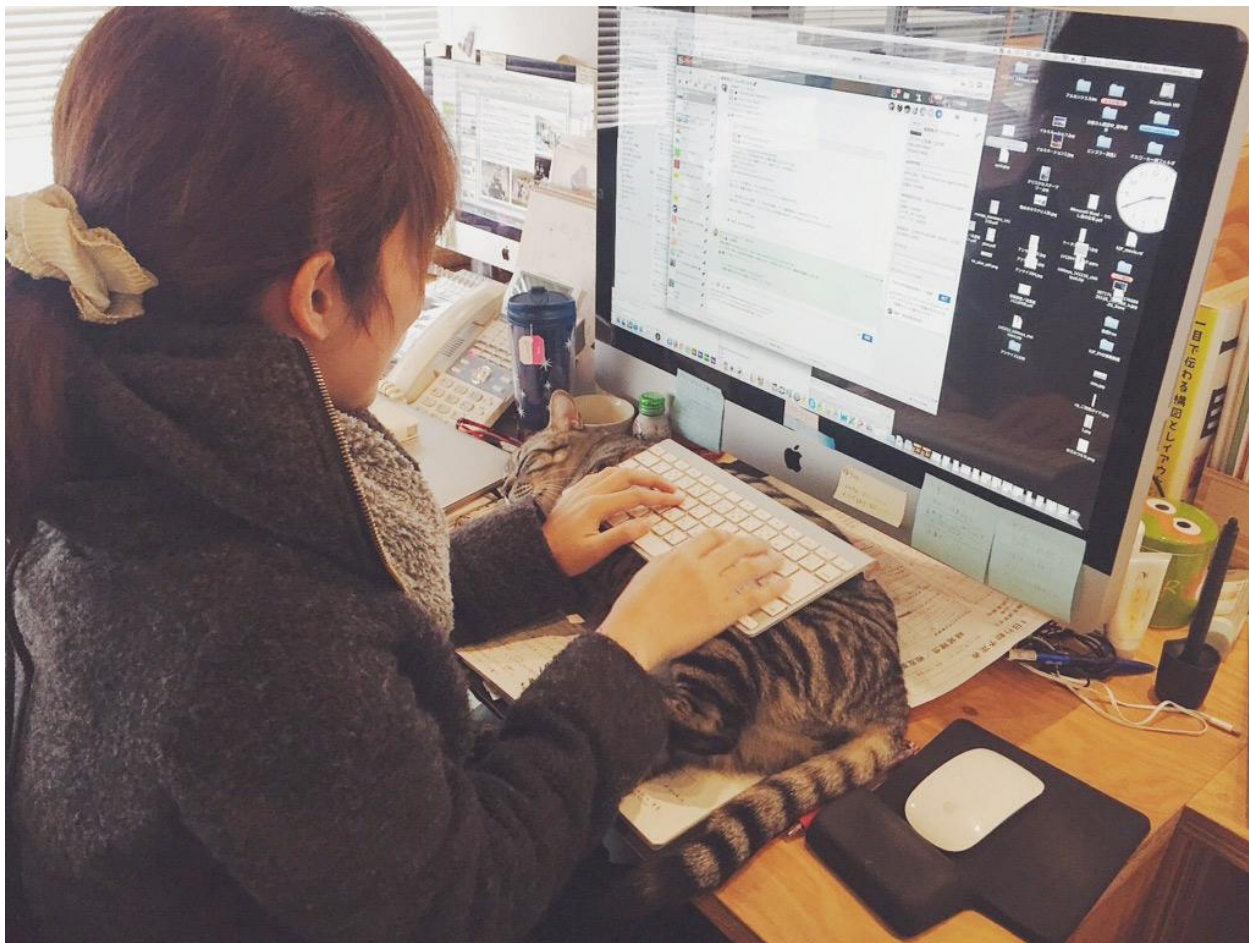
Agenda

- what part of the Big Data World are we focusing on?
- why processing "Big" Data is different from processing "Small" Data?
- types of BDP systems
- class goals and objectives
- administrative details

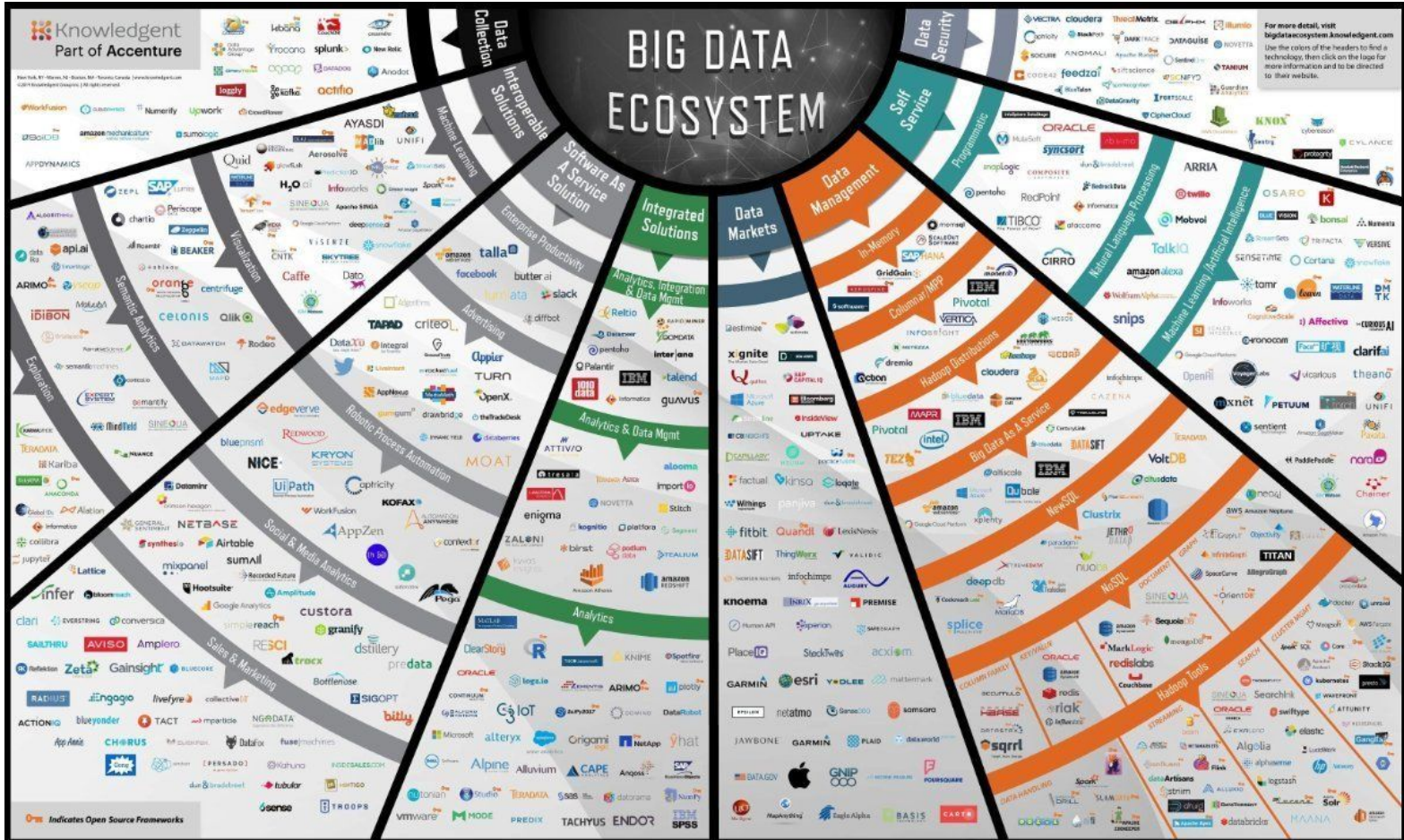
But first - who are the Teaching Staff ?

Who Am I?

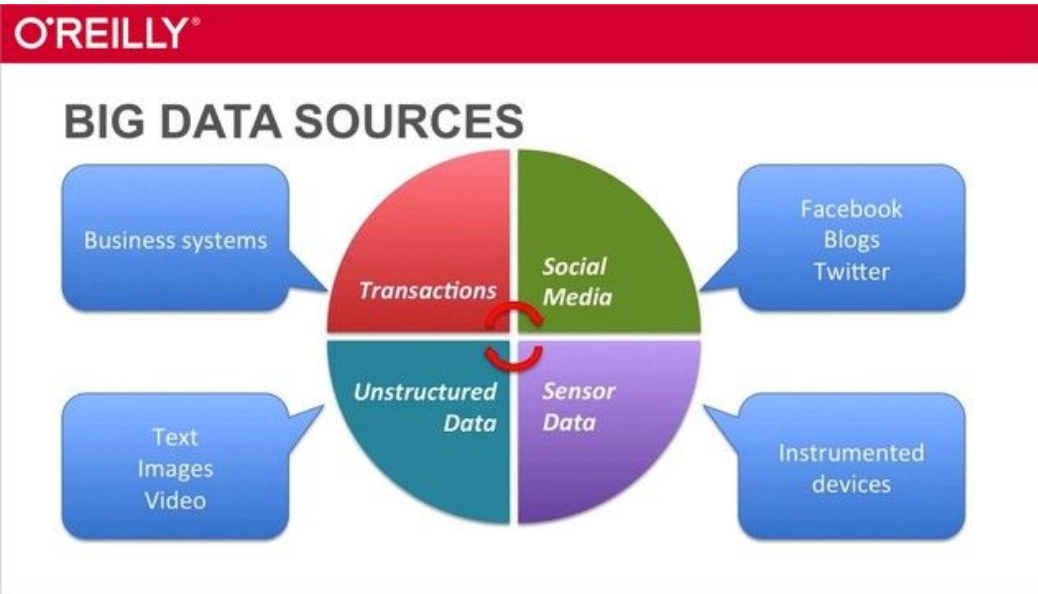
- I am an Engineer !
- love building distributed systems
- instructor at Harvard University Extension
- work at Yottaa



Big Data World



In a nutshell:
You've got data



... what for ??

to make sense out of it!



Ref: <https://cloud.google.com/products/big-data/>

Data Science breakdown:

There is a lot that goes into supporting and implementing a real-world data science system ...

Monica Rogati:

"Think of AI as the top of a pyramid of needs. Yes, self-actualization (AI) is great, but you first need food, water and shelter (data literacy, collection and infrastructure)."

THE DATA SCIENCE HIERARCHY OF NEEDS

LEARN/OPTIMIZE

AGGREGATE/LABEL

EXPLORE/TRANSFORM

MOVE/STORE

COLLECT

AI,
DEEP
LEARNING

A/B TESTING,
EXPERIMENTATION,
SIMPLE ML ALGORITHMS

ANALYTICS, METRICS,
SEGMENTS, AGGREGATES,
FEATURES, TRAINING DATA

CLEANING, ANOMALY DETECTION, PREP

RELIABLE DATA FLOW, INFRASTRUCTURE,
PIPELINES, ETL, STRUCTURED AND
UNSTRUCTURED DATA STORAGE

INSTRUMENTATION, LOGGING, SENSORS,
EXTERNAL DATA, USER GENERATED CONTENT

@mrogati

<https://medium.com/@rchang/a-beginners-guide-to-data-engineering-part-i-4227c5c457d7>

<https://hackernoon.com/the-ai-hierarchy-of-needs-18f111fcc007>

Data Science breakdown: Data Processing vs. Data Analytics

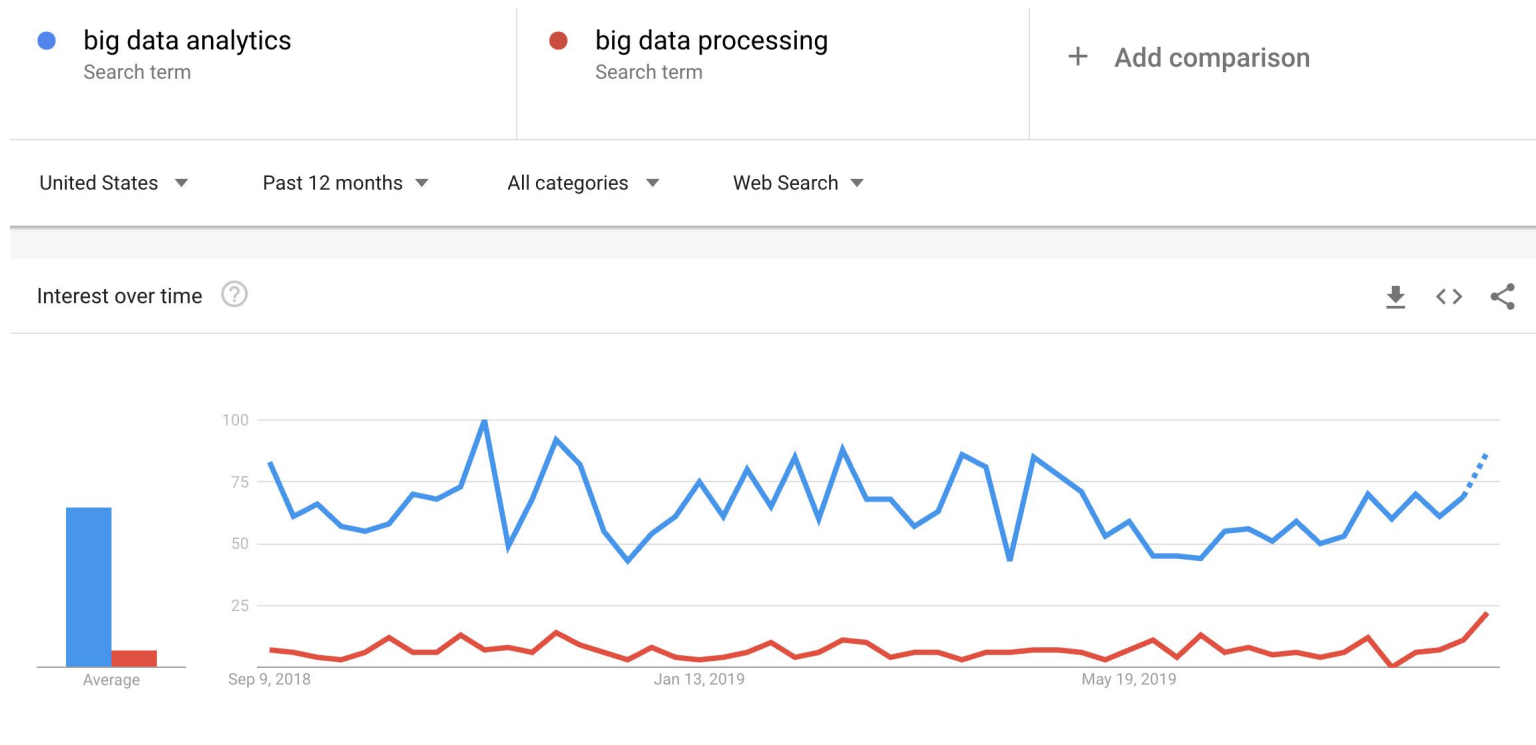
Data Processing: is about sourcing, processing and preserving data in a reliable and fast way

- Main question: **how do I get data and make it available/useful to Data Scientists for further analyses?**
- How to make sure that:
 - No data is lost
 - Data is not duplicated
 - ALL data is available for queries
- How do I store data reliably ?
- How do I process incoming data as fast as I need to and make it available for analyses in real-time?
- How do I ensure that Data Scientists' questions are answered with minimal latency (sub-second response time)?

Data Analytics takes it from where the Data Processing has left it and starts making sense out of the data

- Main question: **what does my data mean ?**
- how do I model data so that I can query it in a meaningful way?
- How similar is this set of data to other sets ?
- How can I predict what my future data will be like based on historical data sets?

Data Processing vs. Data Analytics: popularity context

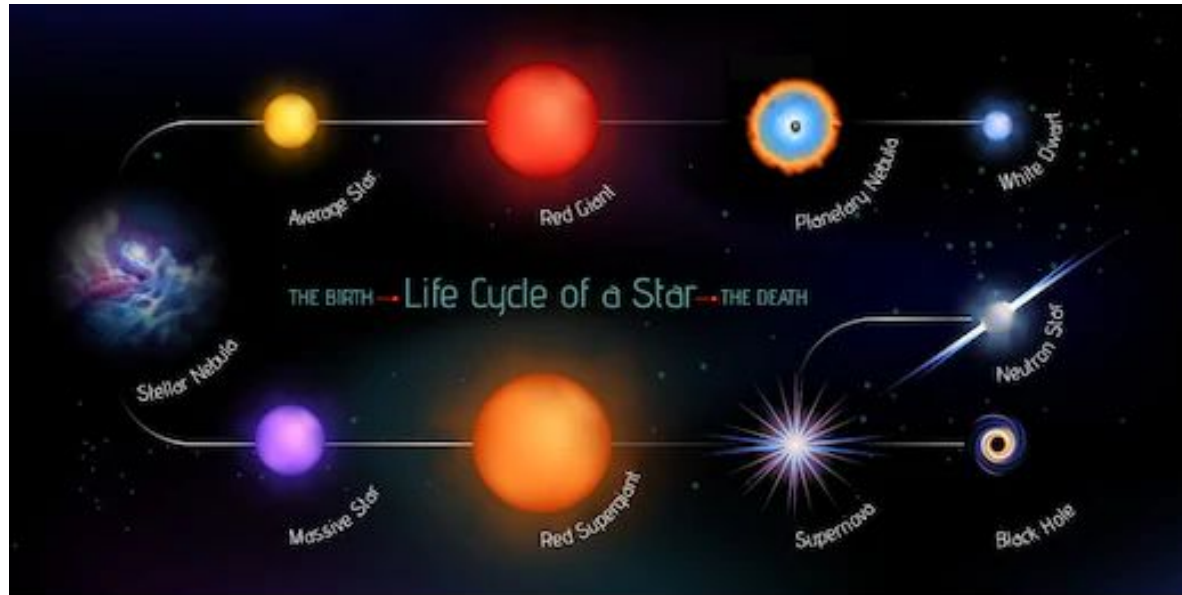


Why do we even need to distinguish Big Data Processing from just Data Processing?

Data Processing Evolution

Lets take a look at how data processing applications evolve as they need to process more and more data

....



shutterstock.com • 655949932

@Marina Popova

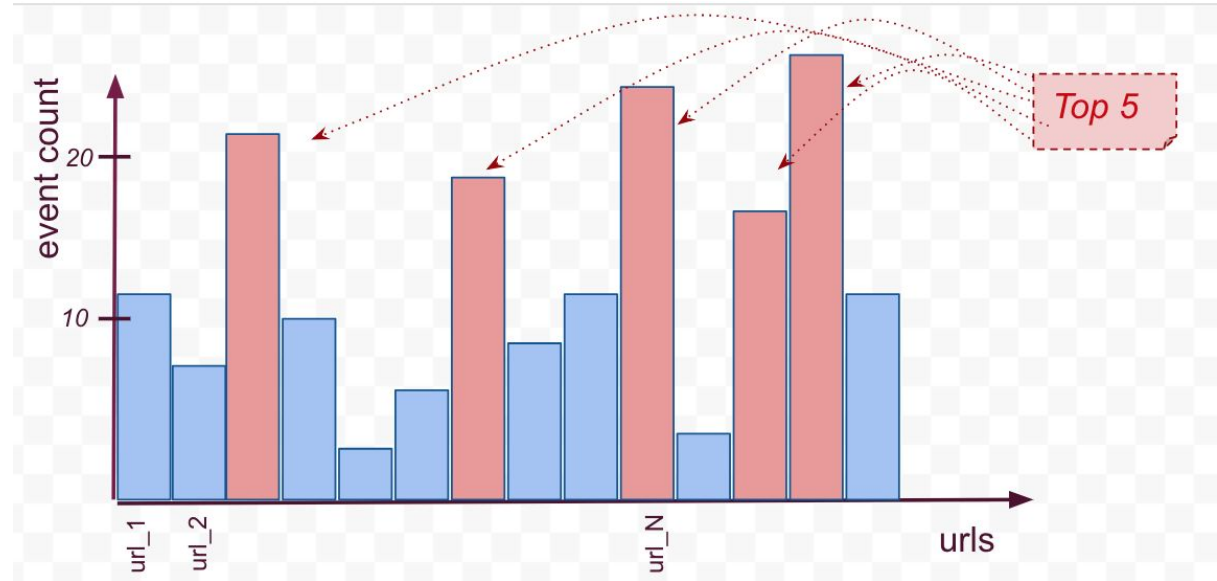
The Journey: Small \rightarrow Big

Example Application:

Online Store Page Analytics

Problem Statement/ Questions:

- how many clicks are done per each page URL ?
- what are Top 5 URLs based on their click numbers ?



The Journey: Small → Big

Traditional 3-tier data applications architecture:

- Web app to process HTTP requests
- SQL DB with relational schema to store the data
- Analytics app to query/ visualize the data (could be the same web app, but is rarely so)

Flow of actions:

- user clicks on pages
- web app gets each request and calls DB to increment the count for the URL
- analytics app queries the DB for the counts by URL or the top 10 URLs

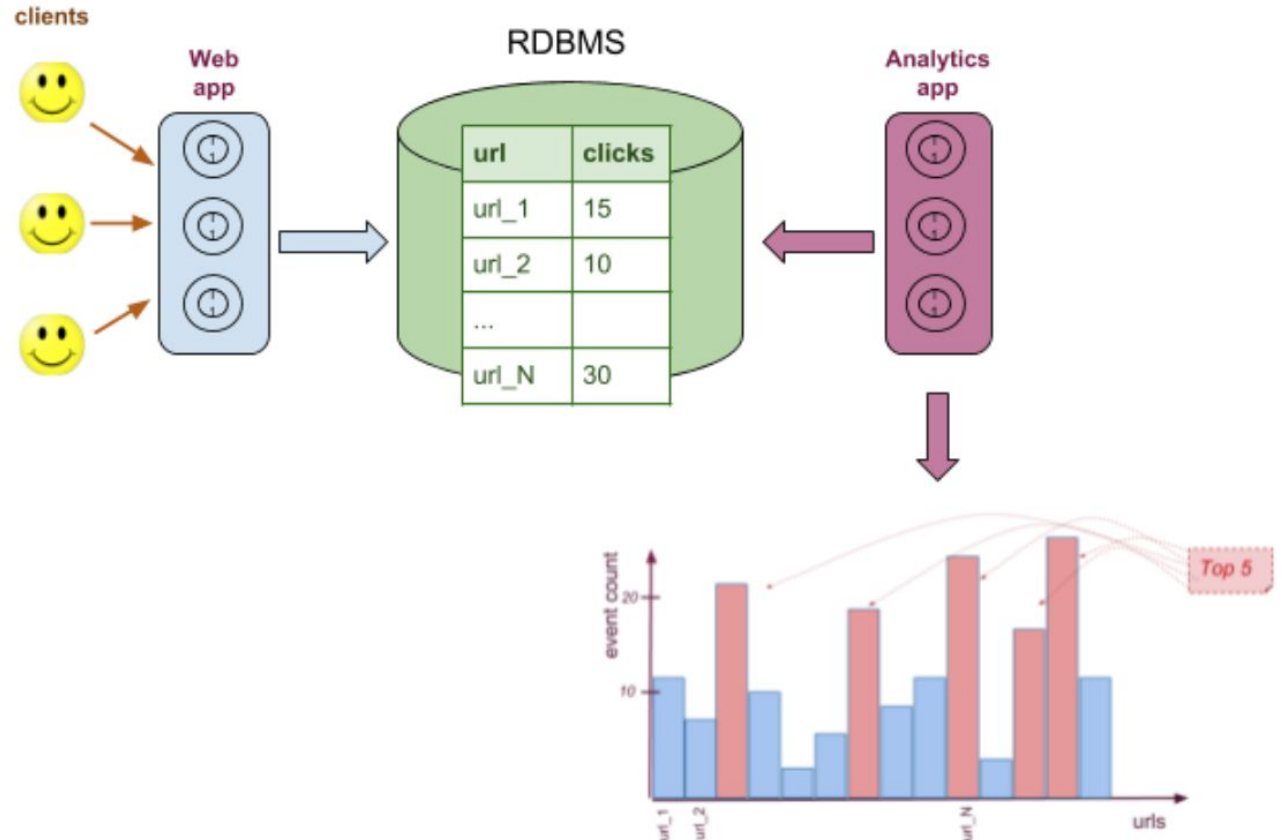
column	type
page_url	String
number_of_clicks	Integer

page_url	number_of_clicks
url1	10
url2	5
...	...

The Journey: Small → Big

"Small" data

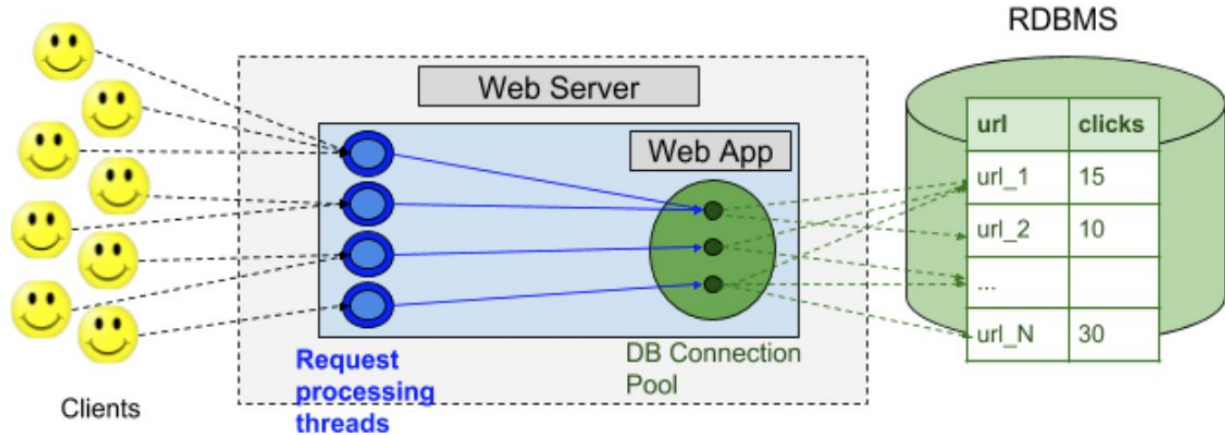
Example architecture:



The Journey: Small → Big

a few more details:
there is a bit more going
on under the covers ...

*we will be using "thread" in a
more generic sense -
combining request processing
and DB connection pool
threads, assuming 1-to-1 ratio*



How can we estimate the MAX
throughput of this system?

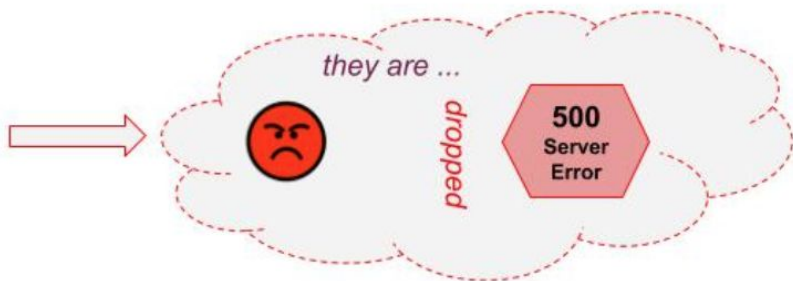
we are relying on the DB to increment
click counts atomically !!

The Journey: Small → Big

Lets do some **conceptual** math - focusing on the ingestion part only for now:

- Lets assume: 1 client (HTTP) request => 1 write to the DB == **100ms**
- the max throughput per second per thread?
 $1000\text{ms}/100\text{ms} == 10$ requests per second

What happens to the 11-th, 12-th, ... MAX+1 request??



Solution:

add more threads and use more connections to the DB:

Threads number	MAX processing rate per second
1	$(1000\text{ms}/100\text{ms}) == 10$
10	100 ??
100	1000 ??
more ??	not really

The Journey: Small → Big

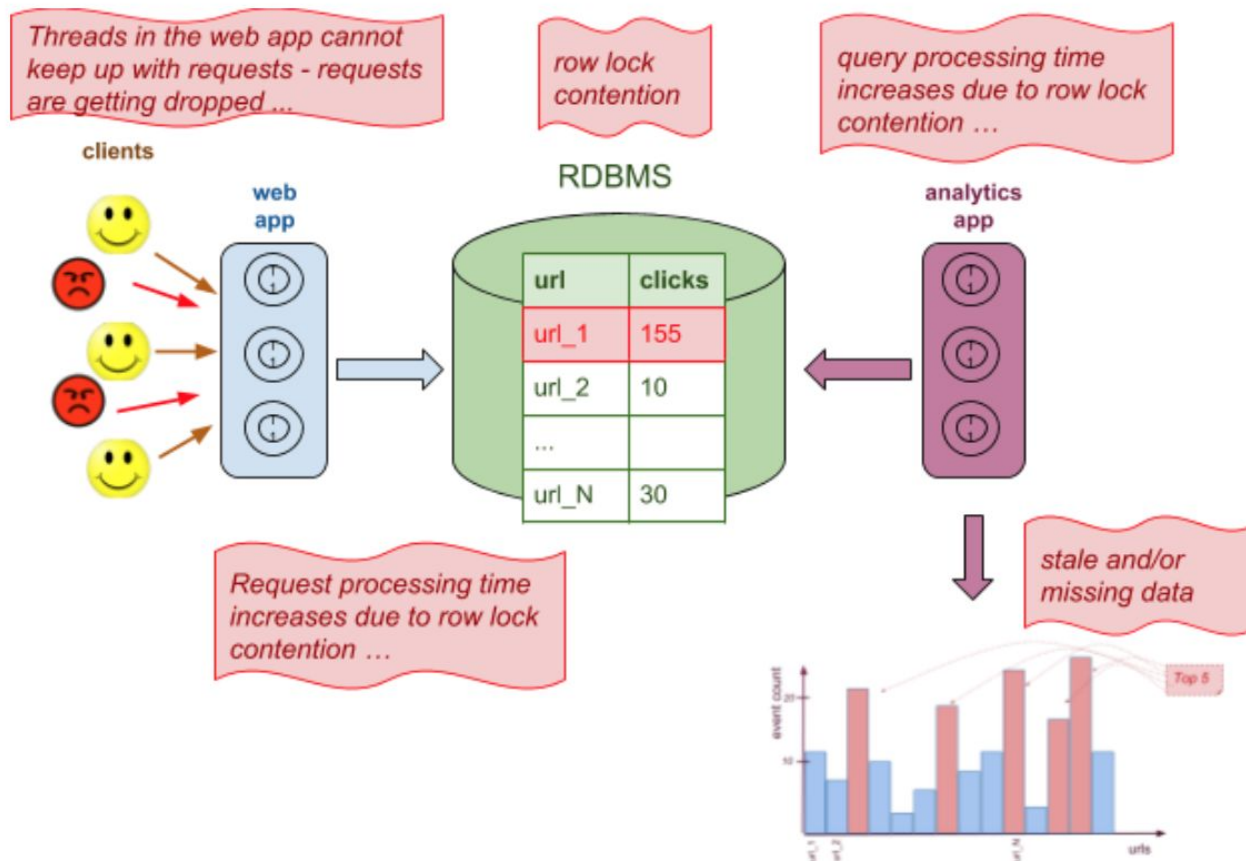
Main issues:

- **client requests** (MAX+1 and above) are **dropped**
- **request processing times and query response times increase** due to:
 - contention on DB connections
 - DB row locking: as multiple threads keep hitting the same rows (URLs) in the table
- **stale data in the DB**

why? math again:

- originally, 1 request took 100ms to complete --> 10 req/sec rate
- lets assume we've added 10 threads - should be 100 req/sec rate , theoretically
- each request processing time might increase, say, to 140ms due to the issues above
- real rate: $(1000\text{ms}/140\text{ms} * 10 \text{ threads}) \approx 71 \text{ requests per sec}$

What problems do we start seeing as traffic increases?



The Journey: Small → Big

Lets address the Problem with dropped connections first - as we cannot afford losing customers/clients

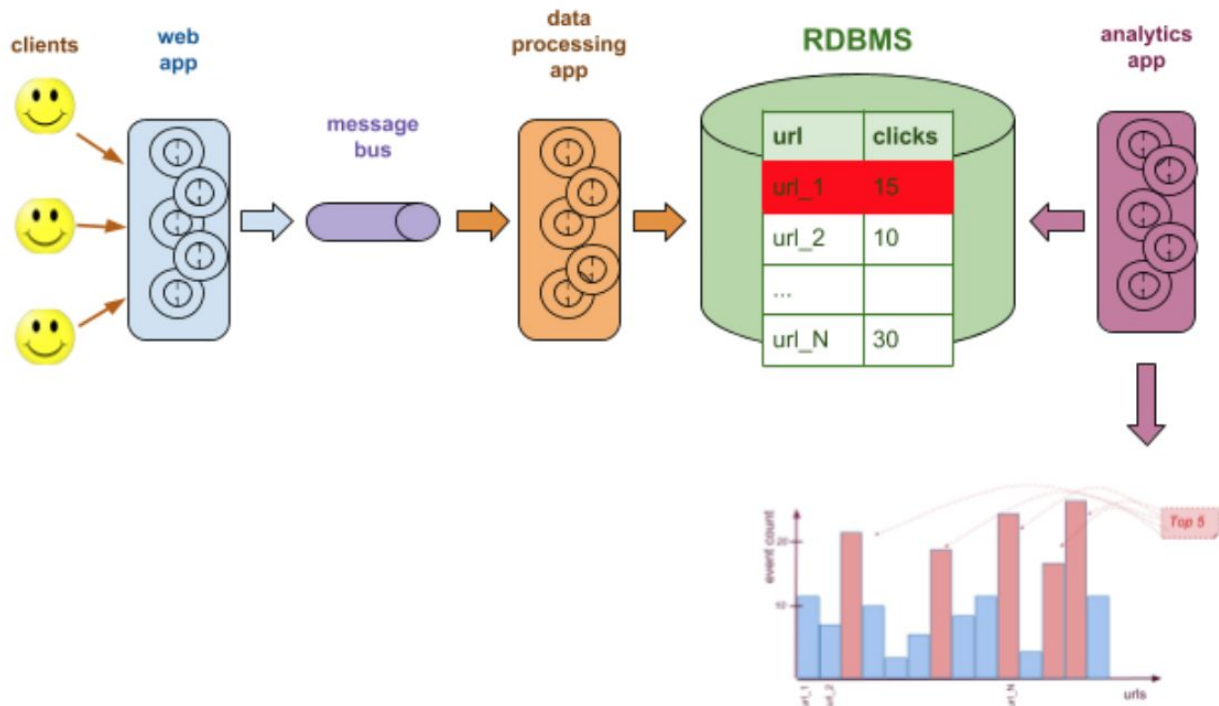
Solution: separate web app request handling from calls to the DB: add a messaging layer and a backend/business app to process messages into the DB

- webapp threads do not call DB synchronously anymore, they send requests as messages to the message bus
- business app picks up each message and updates the corresponding number_of_clicks in the DB
- can start increasing the number of threads (and processing rate) in the web app again ...

The Journey: Small → Big

Fixing dropped client requests :

- add messaging tier
 - async event processing in the web app - unblock clients
- add data processing app between messaging tier and RDBMS
- trying to increase number of threads in each application ...



Traffic increases ...what issues do we start seeing next?

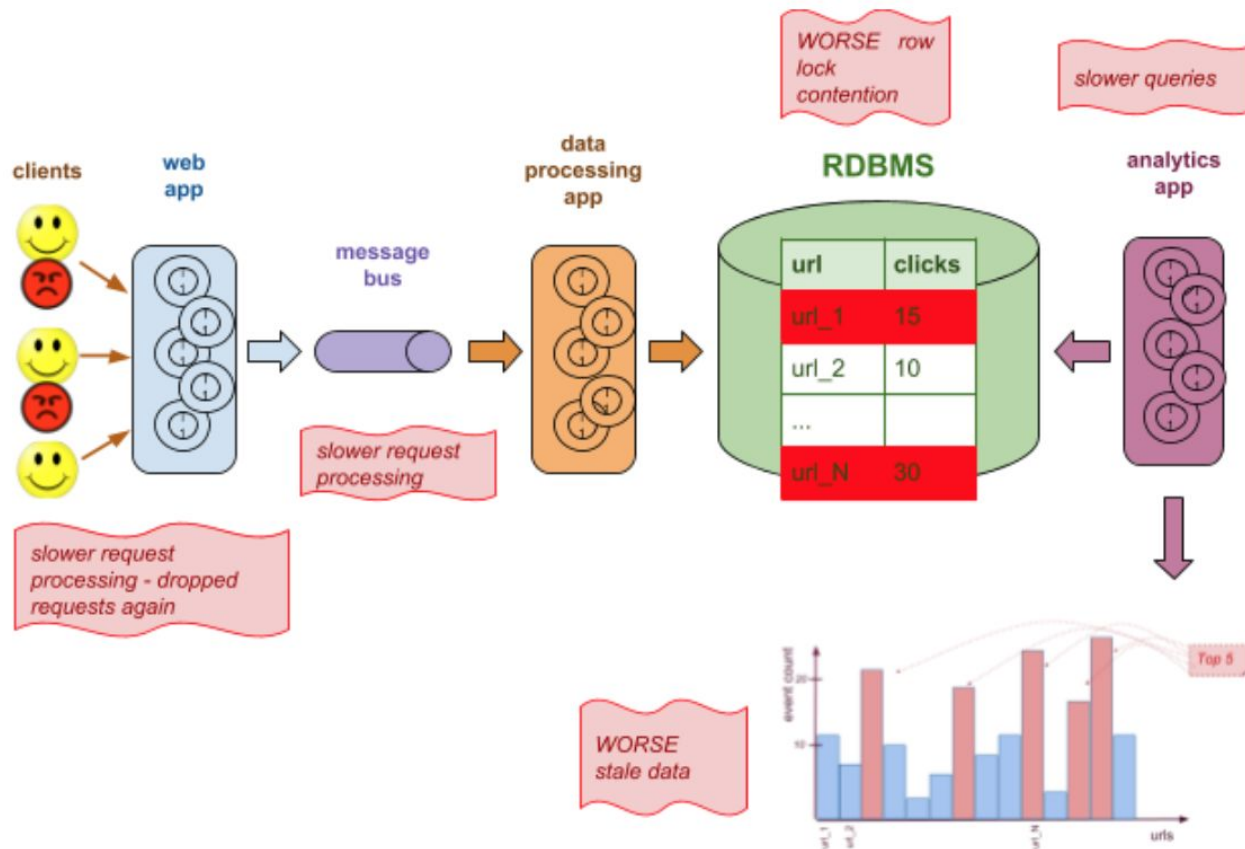
Main issue: single server resource limitations!

Limits are hit on:

- CPUs
- IOPs

Solution?

- multi-node setup
- DB - not easy



The Journey: Small → Big

Lets review main issues:

- adding more threads to apps causes:
 - processing rates degradation - due to thread starvation (CPU, IOPs resource contention)
 - why? we will see in the next Lecture

Solution? add more instances of the apps on separate physical machines

- for web apps, it means we have to add an LB ...
- for all apps: have to distribute processing over multiple physical servers - HOW?
 - this is what we will be learning in the next few lectures ...

The Journey: Small → Big

Main issues - continue:

- message bus: requests processing rate limit (say, 5ms MIN per request)
 - same reasons - resource limitations of the physical server (CPUs, IOPs)

Solution? Use a distributed message bus - across multiple servers

This means, as we will see later:

- less "transactional" message processing
- weaker delivery guarantees (at-least-once, instead of exactly-once)
- no "point-to-point" delivery semantics

Good reference: comparing RabbitMQ with Kafka:

<https://content.pivotal.io/rabbitmq/understanding-when-to-use-rabbitmq-or-apache-kafka>

The Journey: Small → Big

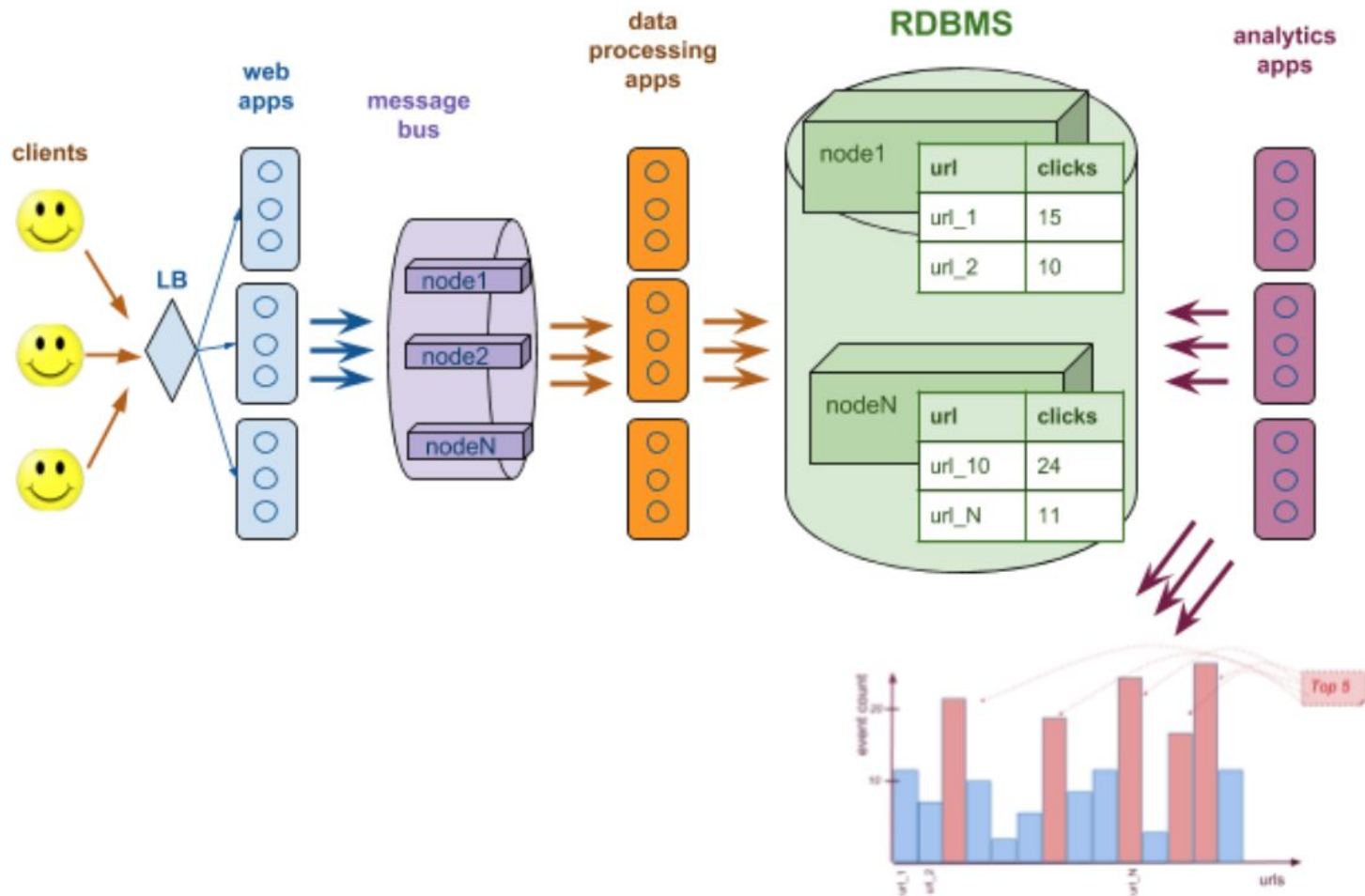
Main issues - continue:

- RDBMS limits: increasing number of threads in the apps causes:
 - sharp increase in the request processing time - say, 1-5 seconds per request - due to row contention
 - why? ACID (as we will see later ...)
 - higher contention on connections to the DB - cannot increase parallelism anymore

Solution? distribute your DB across multiple servers

how? in a moment...

Distributed Architecture



Data Processing Evolution

How do we distribute RDBMS?

- need to make DB writes much faster despite increasing the number of writes - this means:
- use **horizontal sharding** - distribute your DB over multiple servers
- this requires a new approach to storing results - as the same URL and its number_of_clicks has to be stored on the same shard[server].
- have to compute **hash code** of a URL based on the number of shard[servers] available in the cluster.
- either your business app has to do this work - or your DB (if it has this ability).

Next version of the DB schema:

column	type
page_url	String
number_of_clicks	Integer
page_url hashcode [could be maintained internally by the DB]	Integer

Results:

- processing of requests/count increments is very fast again
- Top 10 query becomes slower, because it has to scan multiple shards now

Data Processing Evolution

Biggest concerns here:

A good Relational DB that can be distributed as well will be VERY expensive! There are not many Open Source and free solutions. There are some very good commercial ones (like Teradata, VoltDB, Impala, etc.).

Due to this, you are trying to implement this logic in your Business app. Usually.

Traffic increases - **same problems (slow response from DB) arise again ...**

Solution: **you have to add more shards to the DB (servers)**

If you are managing sharding/hashing in your Business app:

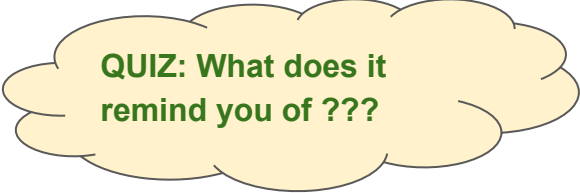
- you have to re-hash all your data for the new number of servers
- you cannot do this in real time - you have to keep your webapps up and running - so you devise a set of background jobs to do this as an async work, which is not an easy thing to do
- you have to make sure you do not lose any of the real-time traffic updates
- it takes a few hours/days - and you are back in business, all is good

Data Processing Evolution

As the traffic grows:

.... You have to do it again..... And again....

Soon all you are doing is rebalancing data in your DB, and adding servers to your webapps, messaging system, business apps and DB



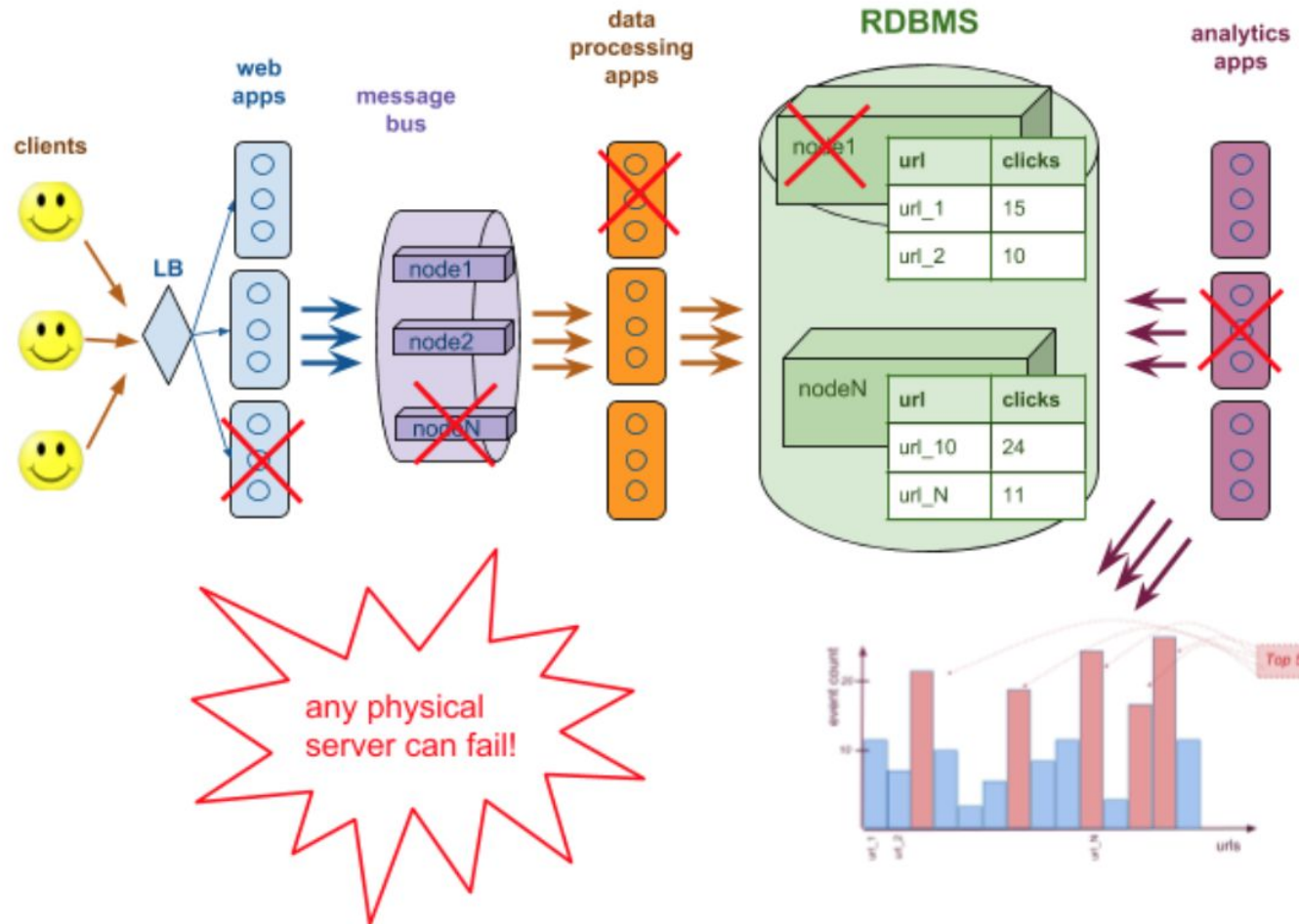
**QUIZ: What does it
remind you of ???**

Wack-a-Mole Game!!!



But WAIT! It's going to be even more fun!!!!

What fails NEXT ??



The Journey: Small → Big

Let's consider possible break points and workarounds

- **LB fails** - very bad, your whole business is down - no easy solution (consider something like AWS Route 53, maybe)
- **one of the web apps fails** - it's Ok, LB will forward the failed request and all others to a different web app instance.
 - Prerequisites: your web app should never ACK request back to LB until it is processed - message is sent to the message bus
- **one of the nodes of your messaging system fails:**
 - If this is a non-replicated setup - data will be lost
 - If this is a system like Kafka with replicas of each event - it is perfectly OK

The Journey: Small → Big

- **one of the business apps fails/dies**
 - Ok as long as it does not ACK message as processed to the bus until the update is finished in the DB
 - However, if you are batching multiple requests - this becomes much more complicated - you may have duplicates or lost batches

and there is much more to this ...

- how do you detect the app instance is "dead" ?
- how do you split input data for processing between multiple instances?
- how do you re-process only the affected part of the data?
- how do you ensure the final data in the DB is consistent (if using distributed DB) ?

all this will be discussed in much more details in next set of lectures - this is what is called "Scaling of Data processing"

The Journey: Small → Big

- **one of your DB nodes goes down** - that's very bad, you will lose data from those shard[s]

Solution: each shard has to have a replica on a different server

But this brings the next set of challenges/ issues:

- who manages this replication ?? - ideally it should be your DB... making it very expensive ...
- it becomes extremely expensive to keep more data - money, processing complexity and resource-wise
- atomic operations are becoming more and more expensive, to the point of: **atomic increments of counts are no longer fast, reliable and / or supported by your DB**
- you may also be getting additional requirements ... -->

Additional Requirements:

- count clicks not just by pageURL, but by userID as well
- query results by userID
- get Top users results - users with highest number of clicks across all (or some subset of) URLs
- and other criteria/ request attributes will be added later as well (such as UserAgent, request origin IP, etc.)

The Journey: Small → Big

Solution:


using incremental architecture is no longer a viable option

- you have to **store EACH individual request**, with all associated information

This means:

- the amount of data stored is much much bigger now
- your queries now have to scan all or most shards of your DB and have the same (< 2s usually) response time
- your writes still have to be very fast (< 30ms usually)
- **your Relational DB cannot keep up with the amount of data and provide fast enough writes and reads**

Solution ?

column	type	Needs index?
page_url	string	yes
<u>requestUUID</u>	string	yes
userId	string	yes
clientIP		yes
userAgent_browser	string	yes 
userAgent_OS	string	yes
userAgent_deviceType	string	yes

The Journey: Small → Big

You need to use a truly scalable distributed DB that supports:

- **automatic partitioning of data**
- **replication**
- **node failures**
- **fast query response times**
- **fast writes**

Time to move to NoSQL solutions!

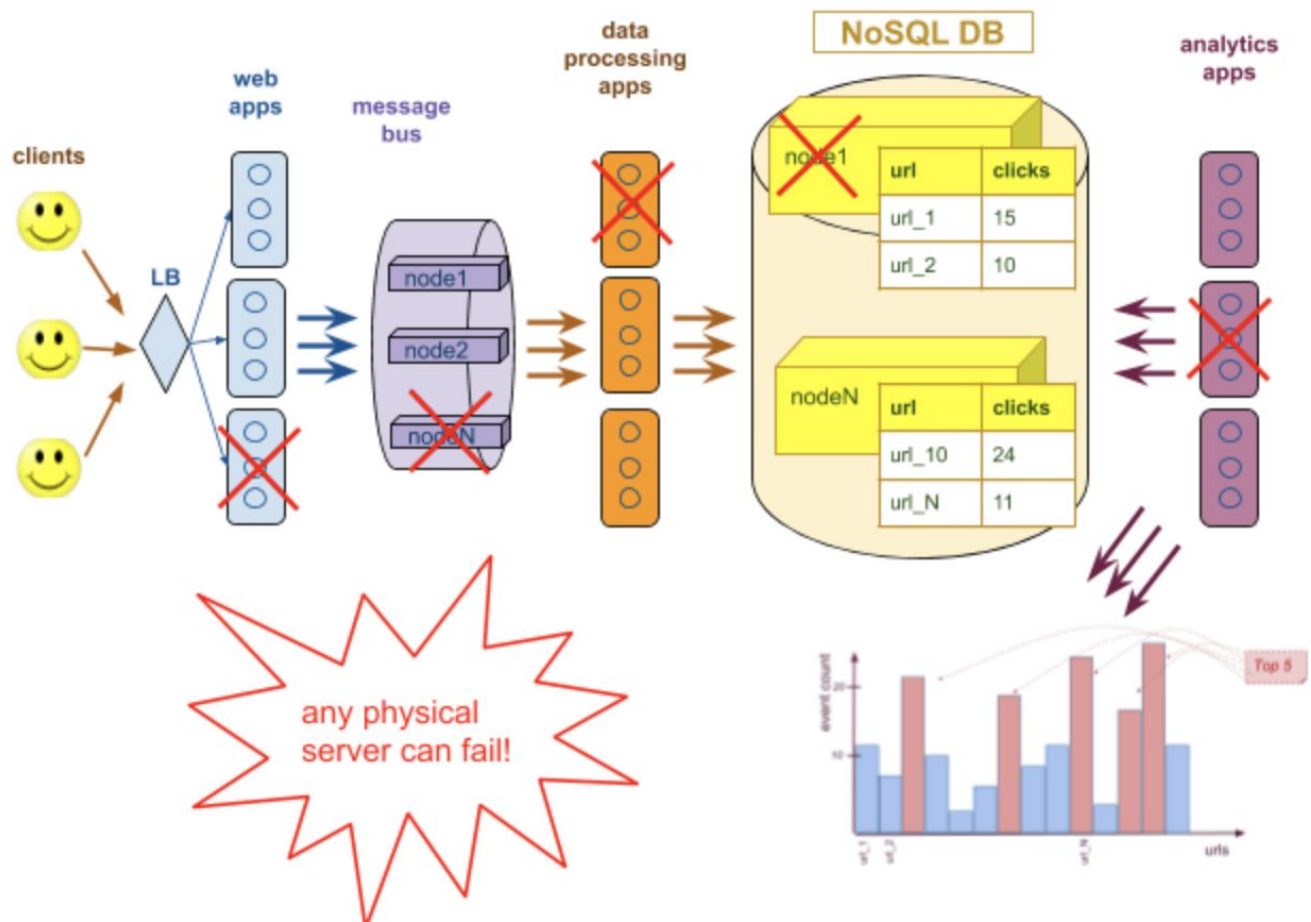
Why? We will spend a lot of time discussing this in the future lectures...

For now - the magic words are ACID (or rather, no ACID)

Interesting Reference: how Cassandra supports updates:

<http://www.datastax.com/dev/blog/lightweight-transactions-in-cassandra-2-0>


The Journey: Small → Big



The Journey: Small → Big

Next set of challenges:

- given the new data schema:
- you have to re-architect your business app to query data differently - to provide required views/reports for the UI:
- you have to calculate the counts now, not just query one column/raw as before
- calculating counts means scanning all relevant rows - which brings the need to have correct indexes created

column	type	Needs index?
page_url	string	yes
<u>requestUUID</u>	string	yes
userId	string	yes
clientIP		yes
userAgent_browser	string	yes 
userAgent_OS	string	yes
userAgent_deviceType	string	yes

The Journey: Small → Big

All works fine for some time, and then

- you start noticing that queries to get **Top 10 results are becoming slower and slower** and no longer fall into < 2sec category
- your business app starts lagging in processing incoming real-time requests - and as a result your queries that calculate counts by some criteria (userId, US, etc) do not return up-to-date counts anymore

The Problem:

you have too much data stored in the Raw Storage DB, and queries over ALL data are taking more and more time to complete

Solution - ????

you don't know what to do
to fix all this anymore! ...



I GET TIRED FROM JUST

**THINKING OF EVERYTHING I
HAVE TO DO.**

You are not alone!

There is a lot of help available !

Final 2018 version, updated 07/15/2018

© Matt Turck (@mattturck), Demi Obayomi (@demi_obayomi), & FirstMark (@firstmarkcap)

mattturck.com/bigdata2018

FIRSTMARK 
EARLY STAGE VENTURE CAPITAL

BIG DATA & AI LANDSCAPE 2018



Main Class Objective:

Do not despair facing the Big Data World!

We understand now that processing Big Data is different from processing "small" data

Next:

- formalize main concepts
- review big data processing pipeline types and architectures

What is Big Data ?

Ref:

<http://searchcloudcomputing.techtarget.com/definition/big-data-Big-Data>

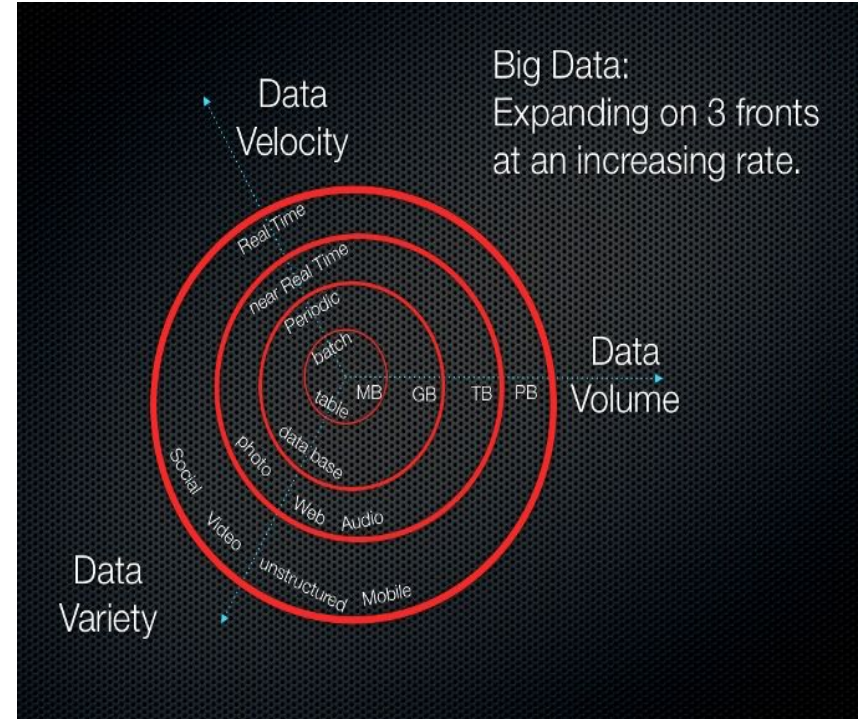
by [Margaret Rouse](#)

Big Data is defined by three Vs:

- **volume** - amount of data
- **variety** - number of types of data
- **velocity** - speed of data processing

According to the 3Vs model, the challenges of big data management result from the expansion of all three properties, rather than just the volume alone -- the sheer amount of data to be managed.

Gartner analyst Doug Laney introduced the 3Vs concept in a 2001 MetaGroup research publication, *3D data management: Controlling data volume, variety and velocity*



Interesting definitions of Big Data from some luminaries

Amy Escobar, Data Scientist, 2U, Inc

[Big data is] an opportunity to gain a more complex understanding of the relationships between different factors and to uncover previously undetected patterns in data by leveraging advances in the technical aspects of collecting, storing, and retrieving data along with innovative ideas and techniques for manipulating and analyzing data.

My personal favorite :-)

Brian Wilt, Senior Data Scientist, Jawbone

The joke is that **big data is data that breaks Excel**, but we try not to be snooty about whether you measure your data in MBs or PBs. Data is more about your team and the results they can get.

Big Data Processing Systems

What is a Big Data System?

Definition: (Ref: "Big Data" by Nathan Marz and James Warren)

A **Data System** is a [set of] application[s] that provides answers to general questions about information that has been collected so far or is being collected in real time.

A **"Big" Data System** is a data system that, on top of that, can work on Terabyte- and Petabyte-scale of data.

What is a Big Data Processing Pipeline?

A system/architecture that addresses all aspects of a required full working solution: including data ingestion, storage, processing, analytics and visualization

The Big Picture

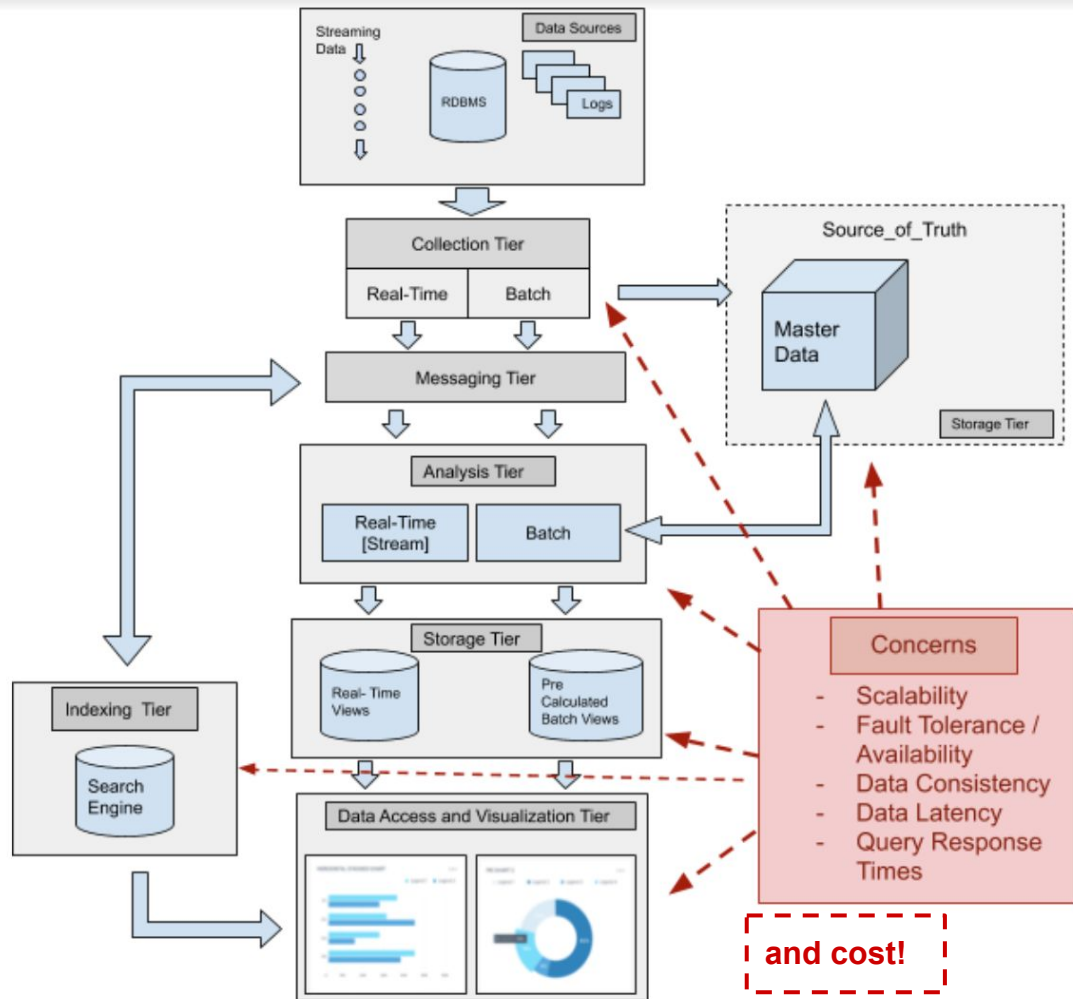
Common building blocks/ tiers of such systems:

1. Collection tier
2. Messaging tier
3. Analysis tier
 - a. Fast real-time
 - b. Batch analysis
4. Data storage tier
 - a. Fast short-term real-time access
 - b. Slower long term storage
5. Data access tier / UI
 - a. Indexing/ Search Engine tiers
 - b. Data access/ visualization tiers

... and **concerns that are common to most of the tiers:**

1. Scalability
2. Fault Tolerance/ Availability
3. Data Consistency
4. Data Latency (staleness)
5. Query response timings

And, of course, **cost** - but this goes without saying :-)



Do you always have to build a system like that every time you wanted to collect/analyze your data? ... ->>

If you did - you'd soon look like this:



Thankfully, not every system requires all that - that's why we categorized them into 4 types

@Marina Popova

BDP System Types:

Type 1: Limited Real-time + Ad-hoc queries systems:

Systems that can answer any [ad-hoc] question based on the **limited raw data** (in-memory only) received during some small recent time window

Type 2: Full Historical + Limited Real-Time + Pre-defined queries systems:

systems that can answer pre-defined questions based on the **historical pre-aggregated data (metrics)** - not on all historical raw data - and on the limited latest real-time raw data

Type 3: Full Historical + No Real-Time + Ad-hoc queries systems:

systems that can answer any [ad-hoc] question on **ALL raw data** collected so far - but no real-time up-to-the-second data

Type 4: Full Historical + Real-Time + Ad-hoc queries systems:

systems that can answer any [ad-hoc] question on **ALL raw data** collected so far, including up-to-the-second data

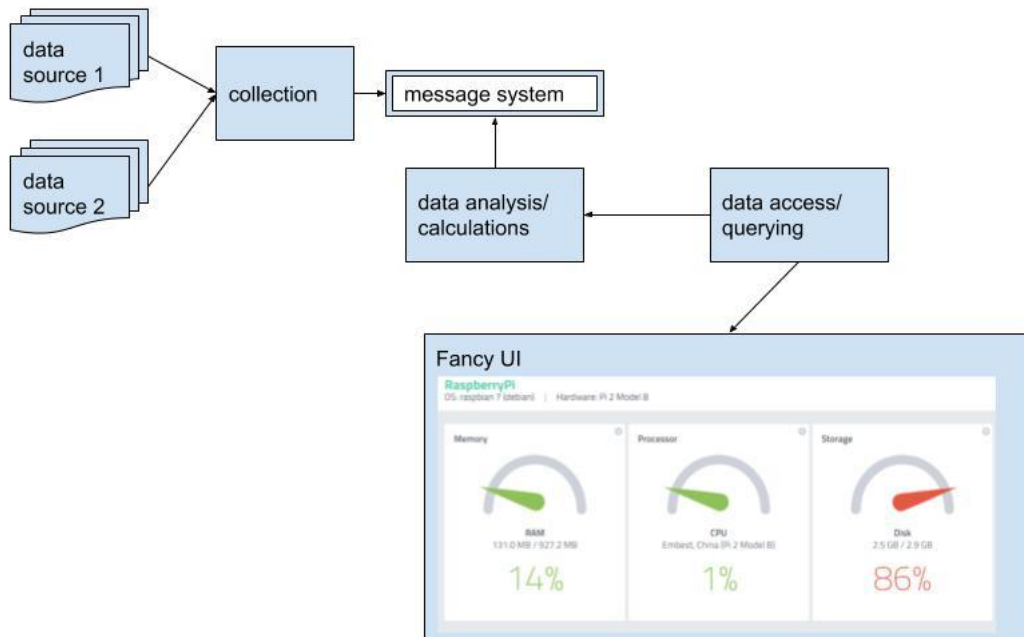
BDP Type 1: Real-time [Limited range] + Ad-hoc queries systems:

These are systems that can answer **any [ad-hoc]** questions based on the **raw** data received during some small recent time window

Examples: Alerting, Fraud detection

Such system can be defined as:

Query = function (recent windowed data)



BDP Type 1: Real-time [Limited range] + Ad-hoc queries systems:

Requirements Rundown:

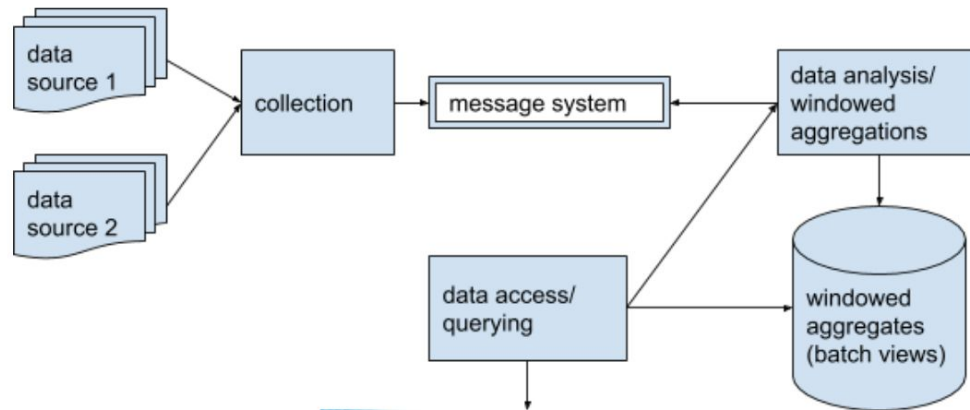
requirement	is required?	can be fulfilled?
Fault tolerance	yes	yes
Low query response times	yes	yes
Low data latency	yes	yes
Data consistency	no	no / limited
Historical data	no	no
Scalability	yes	yes
Ad-hoc queries	yes	yes
Low maintenance/ cost	yes	yes

BDP Type 2: Full Historical + Limited Real-Time + Pre-defined queries systems:

systems that allow you to ask questions based on the **historical pre-calculated data** points (metrics - aggregates) - not on all historical raw data - and on the **limited latest real-time raw data**

Examples: show a trend of pageURL clicks (total over 5-min window) for the past month
Such system can be defined as:

Query =
function(pre-calculated windowed aggregates) +
function [function(recent windowed raw data)]



@Marina Popova

BDP Type 2

Requirements Rundown:

requirement	is required?	can be fulfilled?
Fault tolerance	yes	yes
Low query response times	yes	yes
Low data latency	yes	yes
Data consistency	Yes, moderate	Yes, partially
Historical data	Yes, pre-calculated	yes
Scalability	yes	yes
Ad-hoc queries	no	no
Low maintenance/ cost	yes	yes/ somewhat

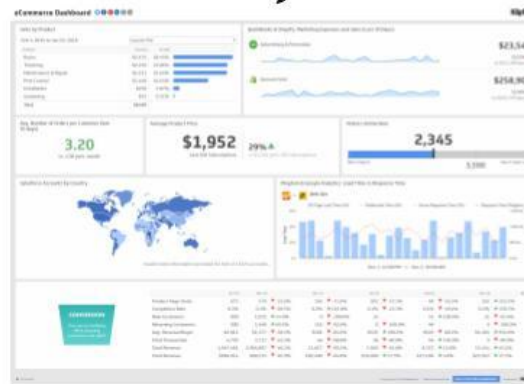
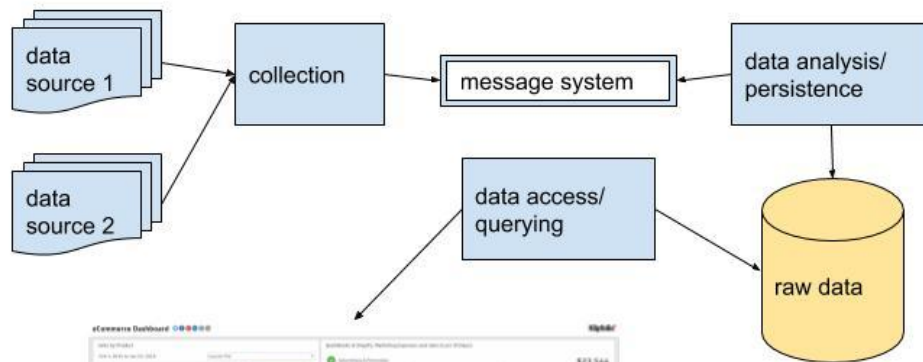
BDP Type 3: Full Historical + No Real-Time + Ad-hoc queries systems

systems that allow you to ask **any question** (ad-hoc query) over **all stored raw data** - but no real-time up-to-the-second data

Example: similar to traditional DB applications - where you can run queries like: "find all tweets on the subject 'cute cats' posted by any of my friends"

Query = function (all raw data)

The data can be stale when the results of a query are finally computed !!



BDP Type 3

Requirements Rundown:

requirement	is required?	can be fulfilled?
Fault tolerance	yes	yes
Low query response times	no	no
Low data latency	no	no
Data consistency	Yes, very strong	yes
Historical data	yes	yes
Scalability	yes	yes
Ad-hoc queries	yes	yes
Low maintenance/cost	yes	yes/somewhat

BDP Type 4: Full Historical + Real-Time + Ad-hoc queries systems

systems that can answer **ANY question based on ALL data** collected so far

Examples: same as above - but the results include up-to-the-second data

Such systems can be defined as following:

Query = function (all data)

Big difference with the Type3/Batch-only systems:

the results of the queries are not stale - they include both historical and real-time data

Have your cake and eat it too ...



BDP Type 4

Requirements Rundown:

requirement	is required?	can be fulfilled?
Fault tolerance	yes	yes
Low query response times	yes	yes
Low data latency	yes	yes
Data consistency	Mix: strong for most data, moderate for latest data	yes
Historical data	yes	yes
Scalability	yes	yes
Ad-hoc queries	yes	yes
Low maintenance/ cost	desirable	not really...

Types of Big Data Processing Systems - Type 4

Why would the previous architecture types not work for these requirements?

Because **running a query over ALL raw data may take a long time** if there is a lot of data (Ts and Ps).

If execution of , say, "Top 10 URLs" query over 10T data store takes 1 hr - by the time you get the results they will be stale by exactly 1 hr, since the new data that came during this 1 hr was not utilized by the Query app.

So, you would need to combine the results of the query on ALL (available at the query start time) data, and the results of the same query over the last 1 hr of the latest data.

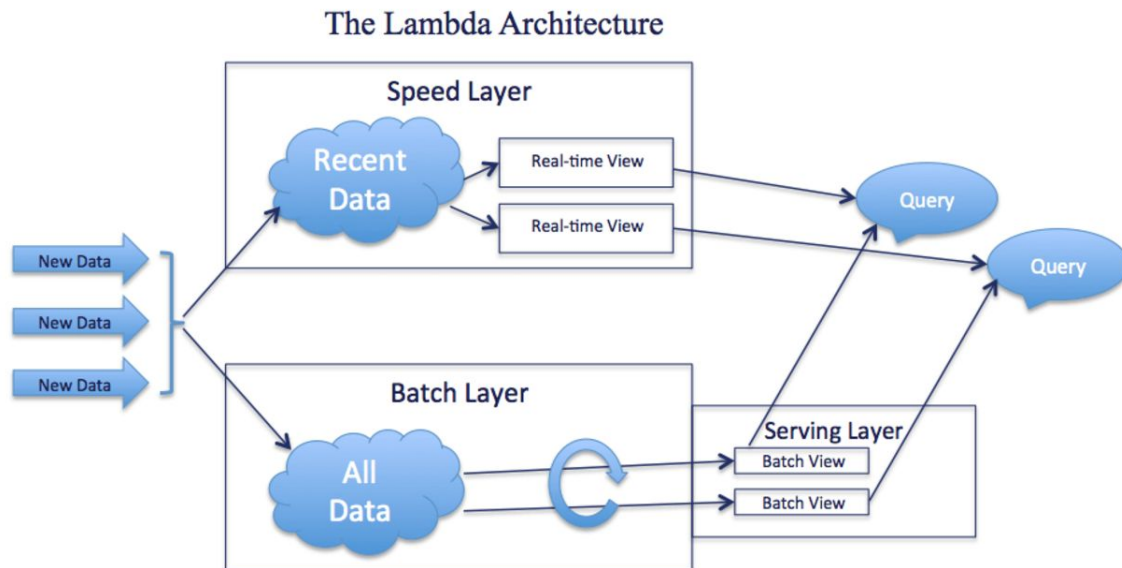
In fact, this type of architecture is a well-known and defined Lambda Architecture

Lambda Architecture - Overview

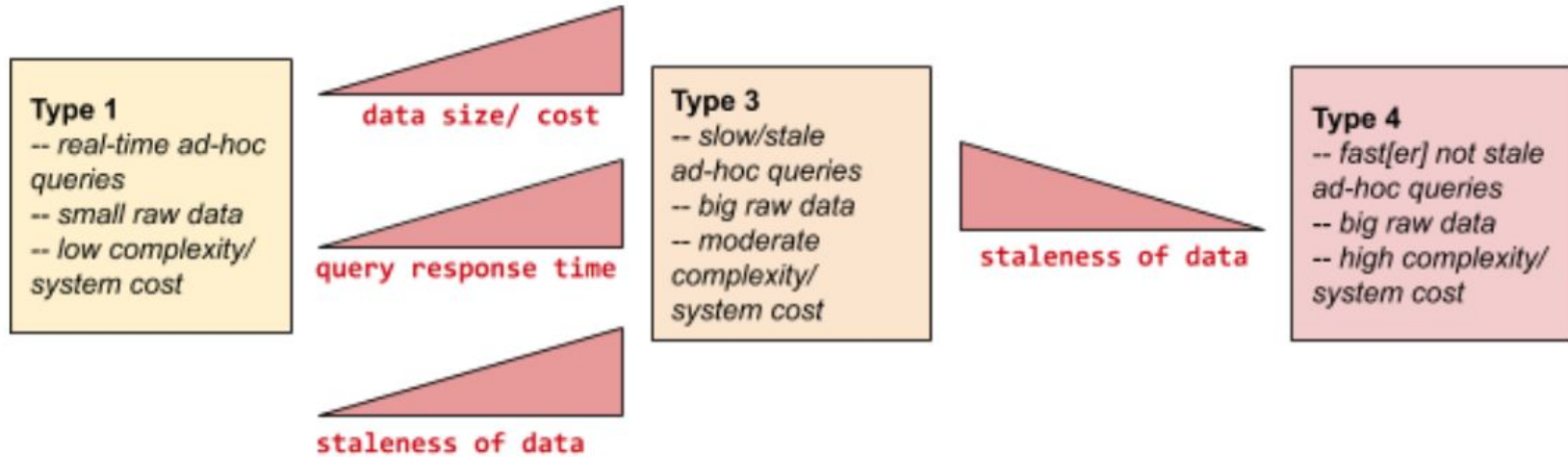
In his book “*Big Data – Principles and best practices of scalable realtime data systems*”, Nathan Marz introduces the Lambda Architecture as:

“... a **general purpose approach to implementing an arbitrary function on an arbitrary dataset and having the function return its results with low latency.** ...

The Lambda Architecture solves the problem of computing arbitrary functions on arbitrary data in realtime by decomposing the problem into three layers: **the batch layer, the serving layer, and the speed layer.** “



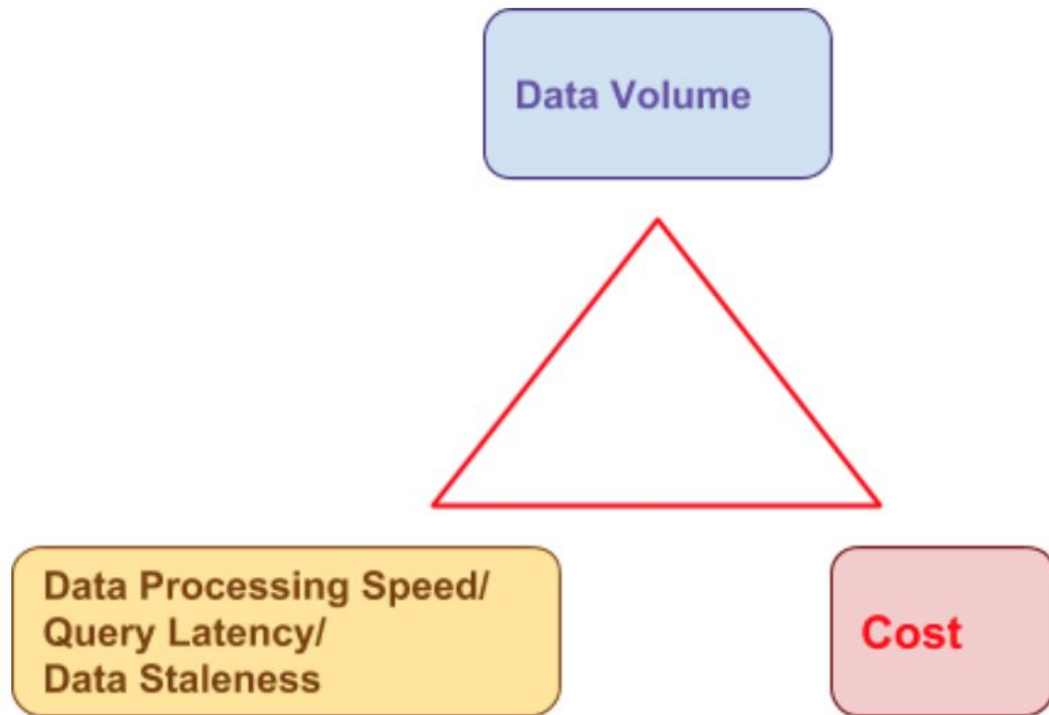
Evolution of systems:



How "small" [raw data] is small?

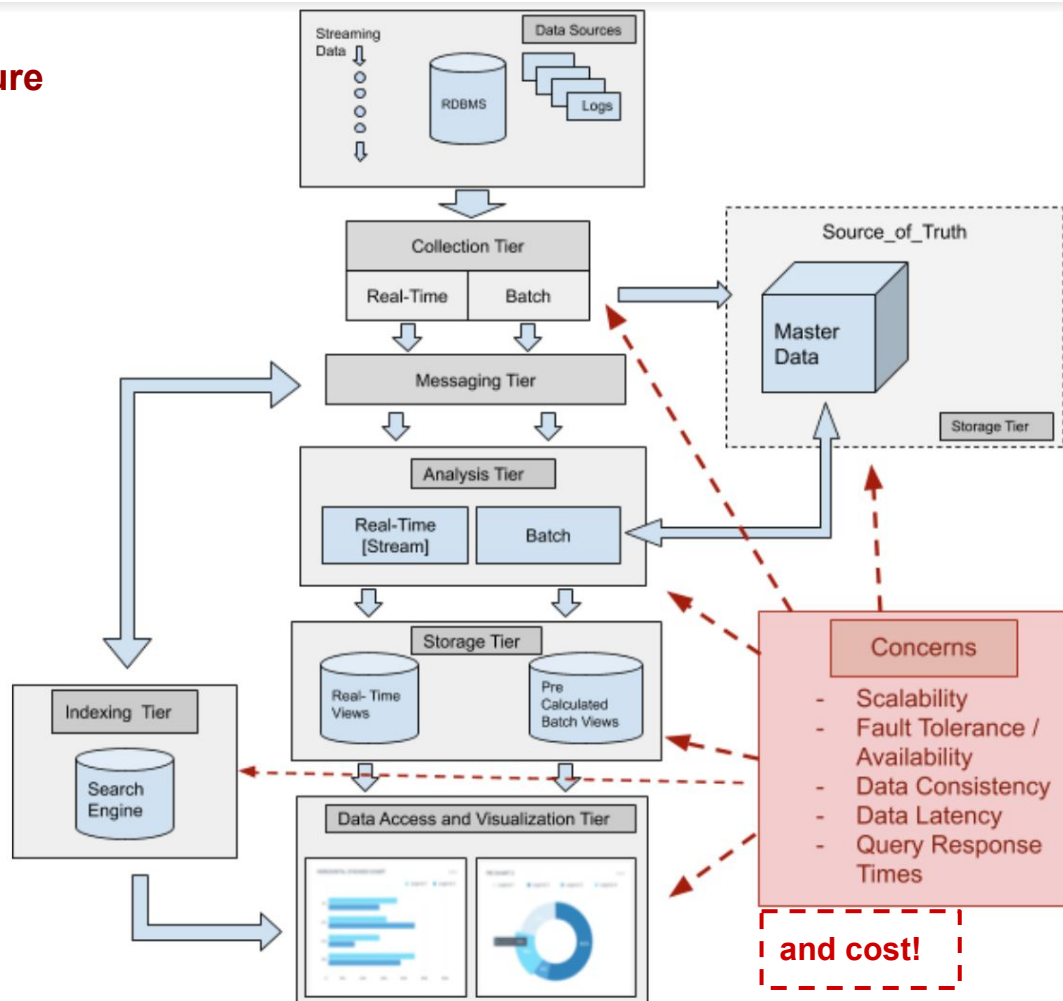
It is small enough to fit your budget and latency/staleness tolerance

So, what are you really trading off ???



**pick 2 !
or 1 ...**

BDP Pipelines Architecture again:



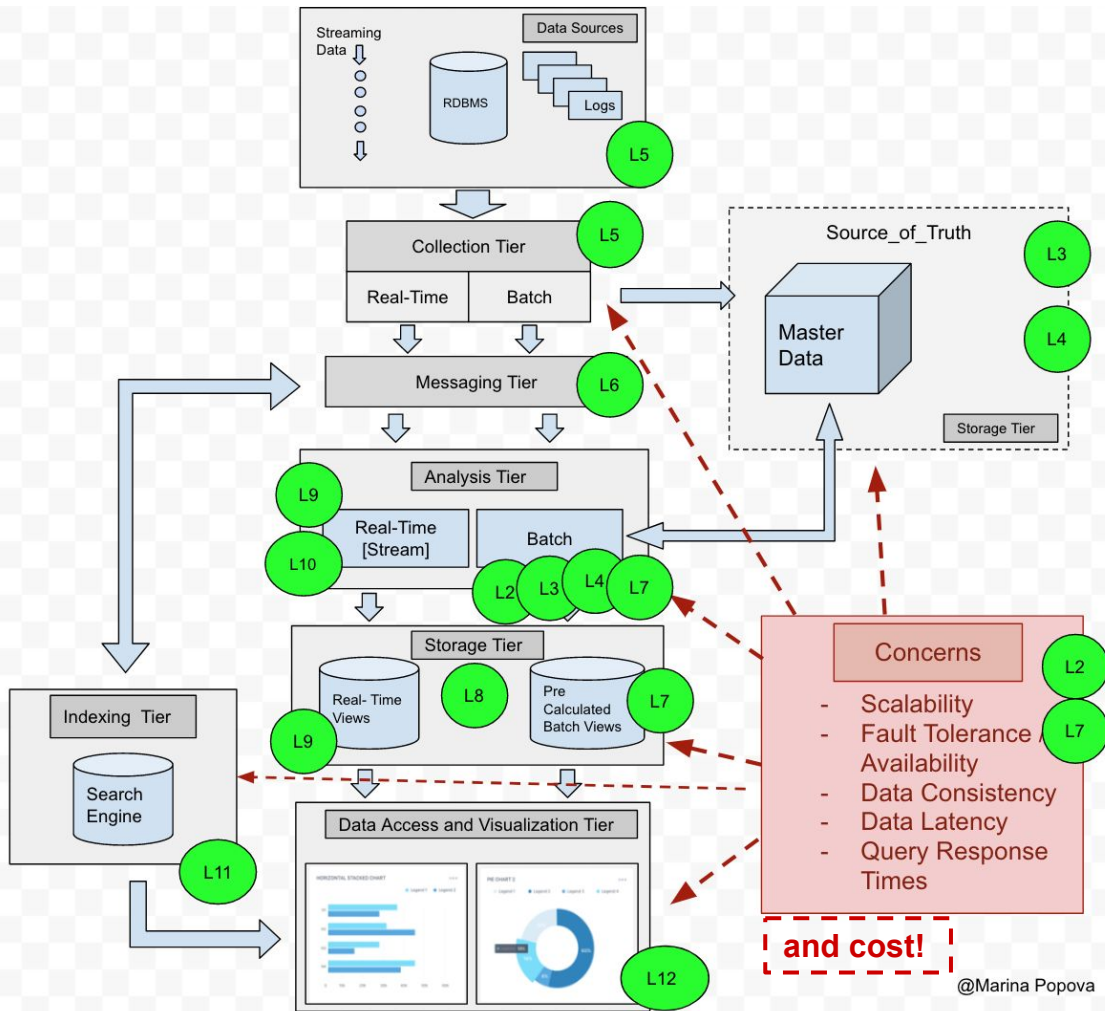
Finale - Class Roadmap

Next, we will see how our class schedule maps onto these topics!

Quiz: :-)

What does it look more like:

1. A Treasure Map
2. "Where is Waldo?" edition



Class Goals

- Understand what are the most important building blocks of full data processing pipelines
- Understand how they are usually implemented
- Understand tradeoffs, common issues and solution in such implementations
- Be able to make informed decisions on the minimal viable architectures for real-life systems based on the specific requirements
- And, most importantly, **have fun doing it!**

What will we learn? And NOT learn?

YES! :

- common types and building blocks of large data processing pipelines
- core principles of each pipeline component
- common challenges and solution approaches
- hands-on experience of building each pipeline component

NO! :

- Tutorial for specific frameworks and languages
- Proprietary architectures and frameworks
- ML/ AI/ Deep Learning

Administrative Details

- Teaching Staff
 - Pavithra Venkatachalam
 - Misha Smirnov
- Class Roadmap - homeworks, Mid-term Quiz, Final Project
- Homework submission policies
- Labs
- Piazza
- Feedback !
- Next Lab: Wed, 09/04 - time TBD