Expanding ACME with BART

W205 Project 3
Spring 2023 - Section 8

Helen Hu
Erik Sambrailo
Ula Zhu

Erik:
Hello, we are Erik, Helen & Ula.
and we are the data engineering team at ACME.

For our project
explored the use of NoSQL databases to
analyze expanding ACME with BART.

Background

ACME Gourmet

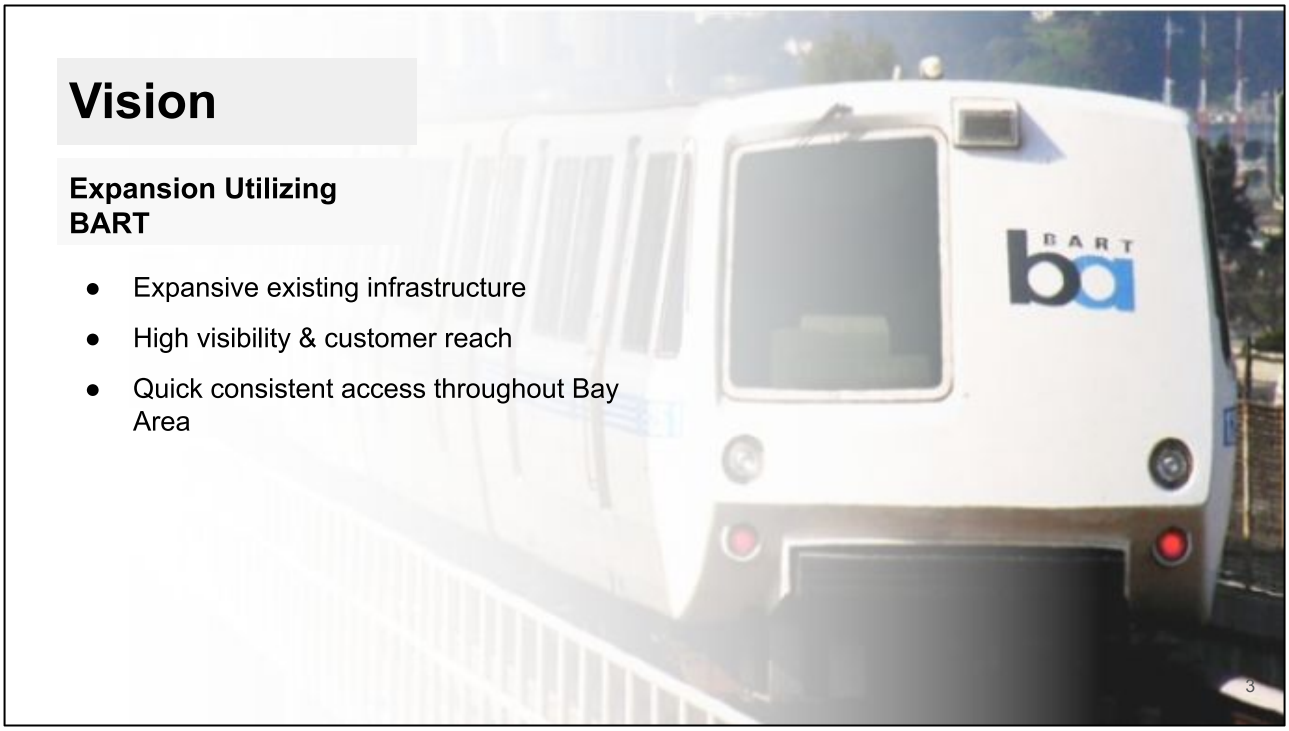
- national gourmet meal producer
- historically brick & mortar
- looking to increase customer outreach
- focused on Berkeley location

Erik:
ACME gourmet meal producer,
providing delicious meals to thousands of people across America.

historically ACME was only accessible to customers through their brick-and-mortar locations.

ACME is now pushing to expand their customer reach by providing additional pickup and delivery services.

We are specifically focusing on their oldest location, Berkeley for this analysis.



Vision

Expansion Utilizing BART

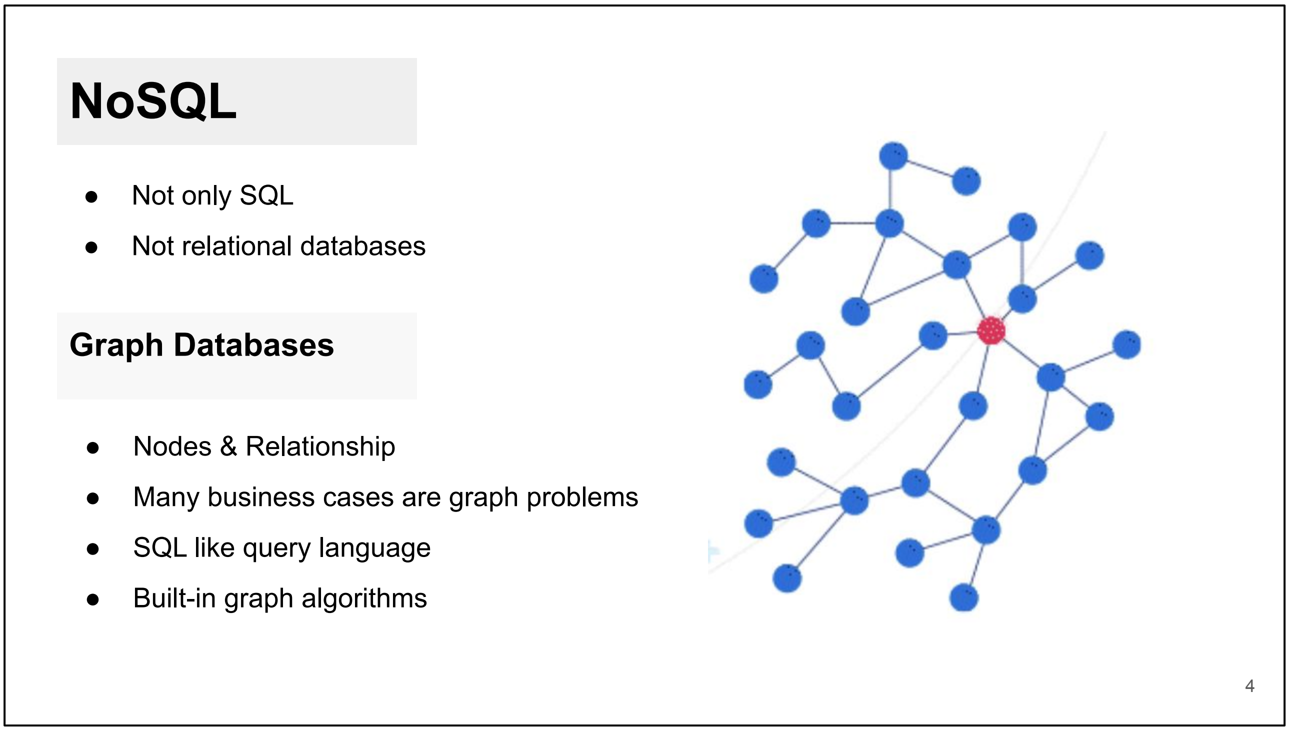
- Expansive existing infrastructure
- High visibility & customer reach
- Quick consistent access throughout Bay Area

Erik:
BART is the expansive bay area rail transit system,
reaching all corners of the bay.

Millions of people live within a couple of miles of BART stations,
and BART is used by thousands daily
providing high visibility and potential customer reach

Our Vision is to utilize BART,
not only as a component of our distribution system,
but also to
pinpoint the best stations to act as hubs and customer pick-up locations.

photo reference:
https://www.wrm.org/about/blog/item/178-help-save-a-historic-bart-train



NoSQL

- Not only SQL
- Not relational databases

Graph Databases

- Nodes & Relationship
- Many business cases are graph problems
- SQL like query language
- Built-in graph algorithms

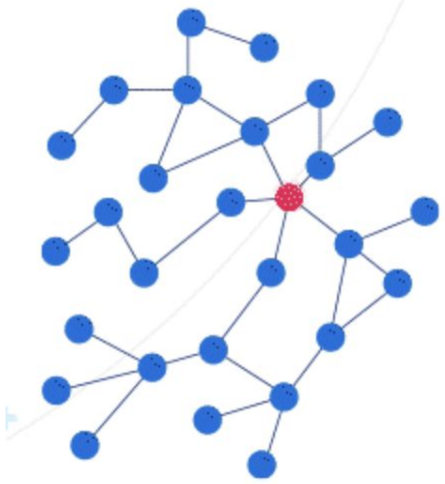


photo reference: https://neo4j.com/blog/graph-embeddings-ai-learns-solve-problems/

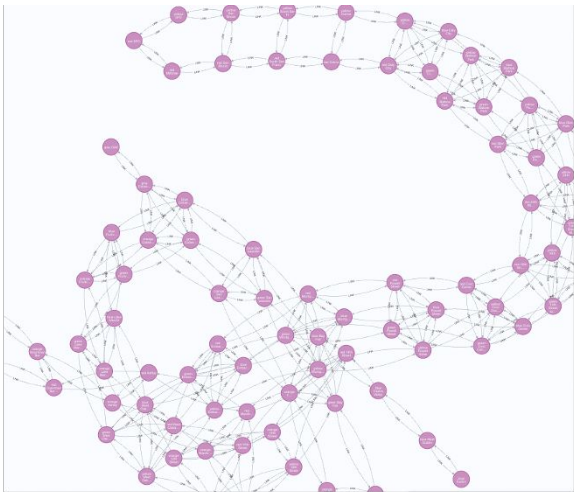
Ula: (1:24)
Our goal of today's presentation is to go over the main types of NoSQL databases so we are aware of the other technologies out there and be able to use the best way to store and retrieve data. NoSQL means not only SQL and it stores data in a format other than relational databases. There's a lot of different types of NoSQL databases. They are all very specific and are able to make a relational database fit when in fact it doesn't.

The first type we'd like to cover is graph databases. A graph is made up of nodes and relationships. Many problems are naturally represented as graphs, like the mapping of the BART station in our case. It learns the structure of the connected stations and adds visibility to our blind spots. It has a SQL like query language, which means the SQL coding skill is transferable. There can also be direction, attributes, and key value pairs setup. One of the most useful functions which we'll show you soon is the built-in graph algorithms. There are different types of algorithms that could be applicable to various business cases, which makes it very easy for graph exploration.

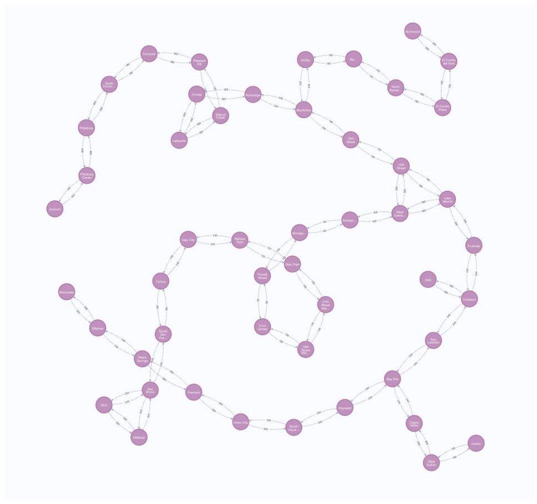
Relational database might not be a good fit in this case because it just simply can't handle graphs. Not to say it won't have built-in graph algorithms so we'll need to create more custom algorithms, which takes way more time and resources.

neo4j

Full Graph: 214 nodes



Simplified Graph: 50 nodes



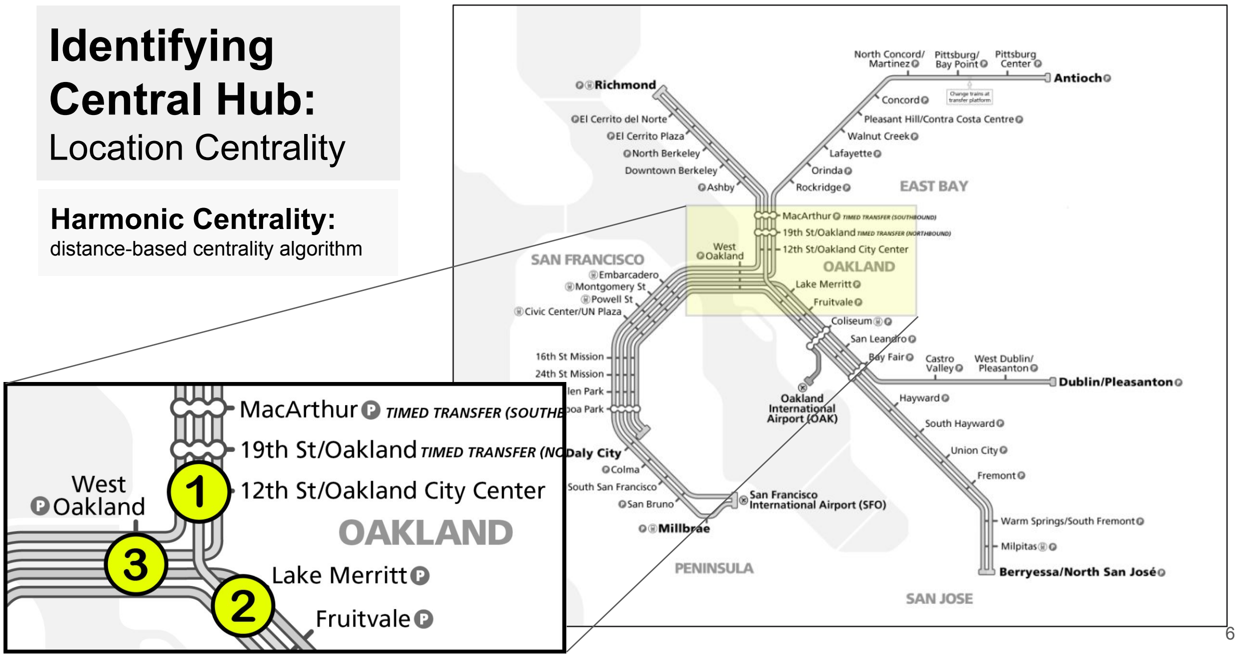
Ula: (1:08)
Neo4j is a graph database that can store and query connected data. We created two graphs using Neo4j with the tables given by our IT department. The full graph on the left side contains at least two nodes per station: one depart and one arrival. It'll have more nodes if there's more than one line bypassing the station. With only 50 stations, there's a total of 214 nodes. Weights between stations are setup as transit time. This full graph is useful to run graph path algorithm because it's designed to take transfer points and transfer times into consideration.

On the right hand side is the simplified graph, it's based on the full graph but we only kept one node per station and removed the weights. It's easier to run centrality and community detection algorithms using this graph because we'll get different values for the same station and it'll need more consolidation work after running the algorithms.

In the next couple slides, we'll go over how each algorithm works and could help us make decisions.

Identifying Central Hub:
Location Centrality

Harmonic Centrality:
distance-based centrality algorithm

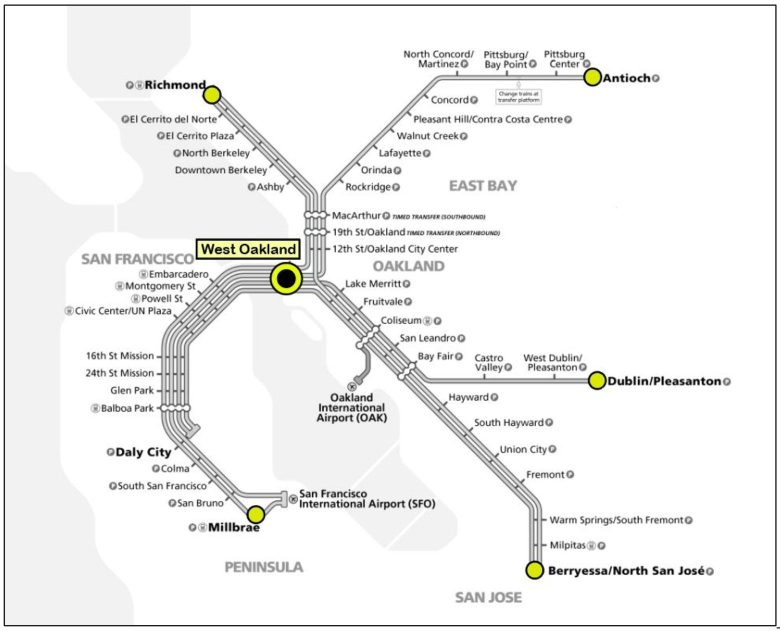


Ula:
First we would like to locate the central hub near the center of the graph for efficiency's sake. Harmonic centrality allows us to rank the nodes that are closest to the center of the graph. The top three stations we found are: 12st street, following by Lake Merrit, and West Oakland, which visually are all located in the middle of the graph.

Identifying Central Hub:
Population Centrality

Dijkstra's algorithm:
shortest path algorithm

Location	Type
West Oakland	Central Hub



Erik:
In our neo4j graph we also quantified the populations within a mile of each Bart station.

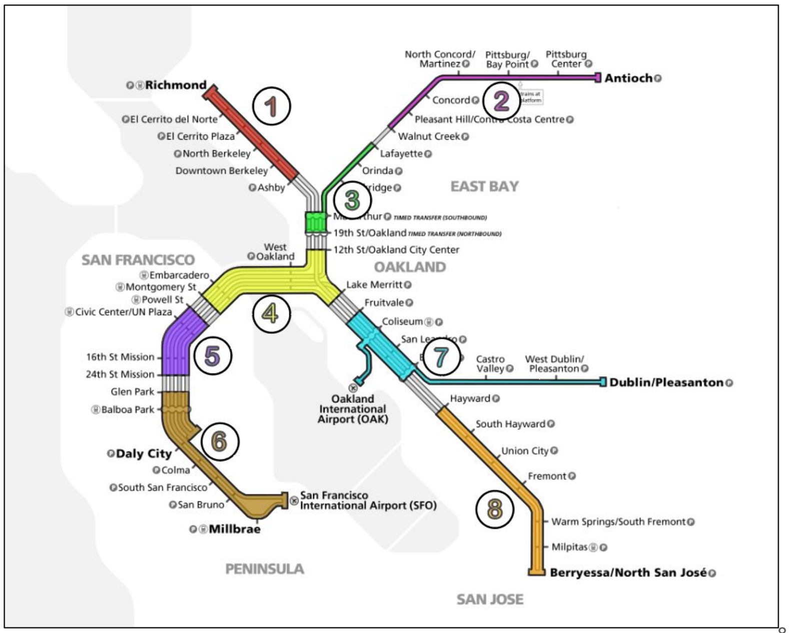
With the potential hub locations ranked,
we used shortest path to loosely quantify populations
between potential hub locations and edge stations (i.e. Richmond & Millbrae)

because populations are denser on peninsula branch,
we chose to have West Oakland as distribution hub

more centrally located
from both a location and population standpoint

Identifying Communities

Louvain Modularity:
Community detection algorithm



Erik:
To better service the entire BART network,
We wanted to cluster the Bart system into communities

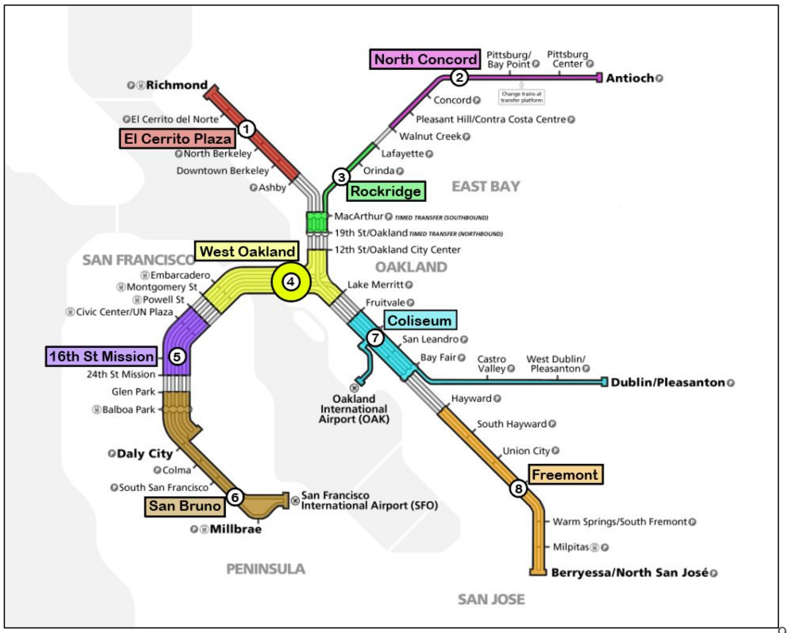
To do this we used the community detection algorithm,
Louvain modularity,
and applied it to our graph with travel times.

Depicted here are the 8 communities that were the result of that algorithm.

Identifying Satellite Hubs

Harmonic Centrality:
distance-based centrality algorithm

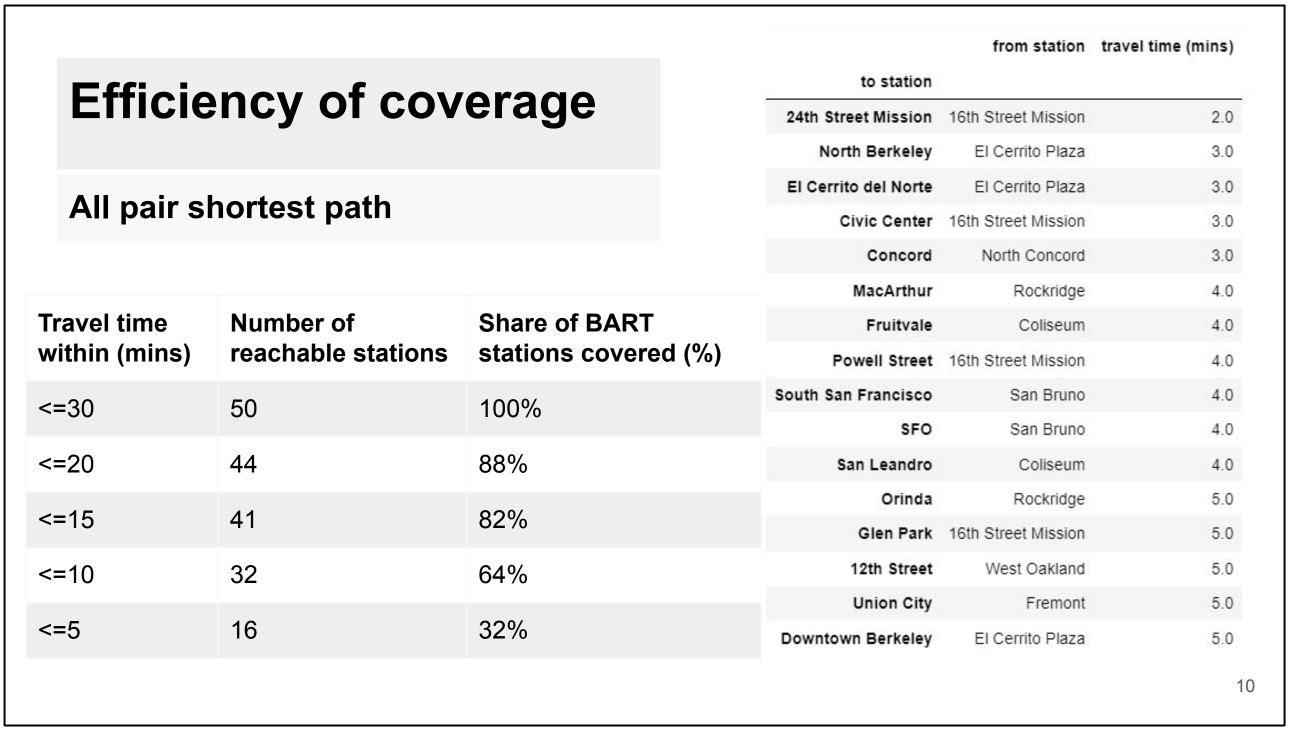
Locations	Type
West Oakland	Central Hub
El Cerrito Plaza	Satellite
North Concord	Satellite
Rockridge	Satellite
16th St Mission	Satellite
San Bruno	Satellite
Coliseum	Satellite
Freemont	Satellite



Erik:
With the communities identified,
we wanted to select the best Bart stations, for each community, to act as satellite hubs and pick up locations.

To do this we iterated through creating graphs for each community
and then running the Harmonic Centrality against them.

Depicted are the stations chosen as our satellite and pick-up locations.



Efficiency of coverage

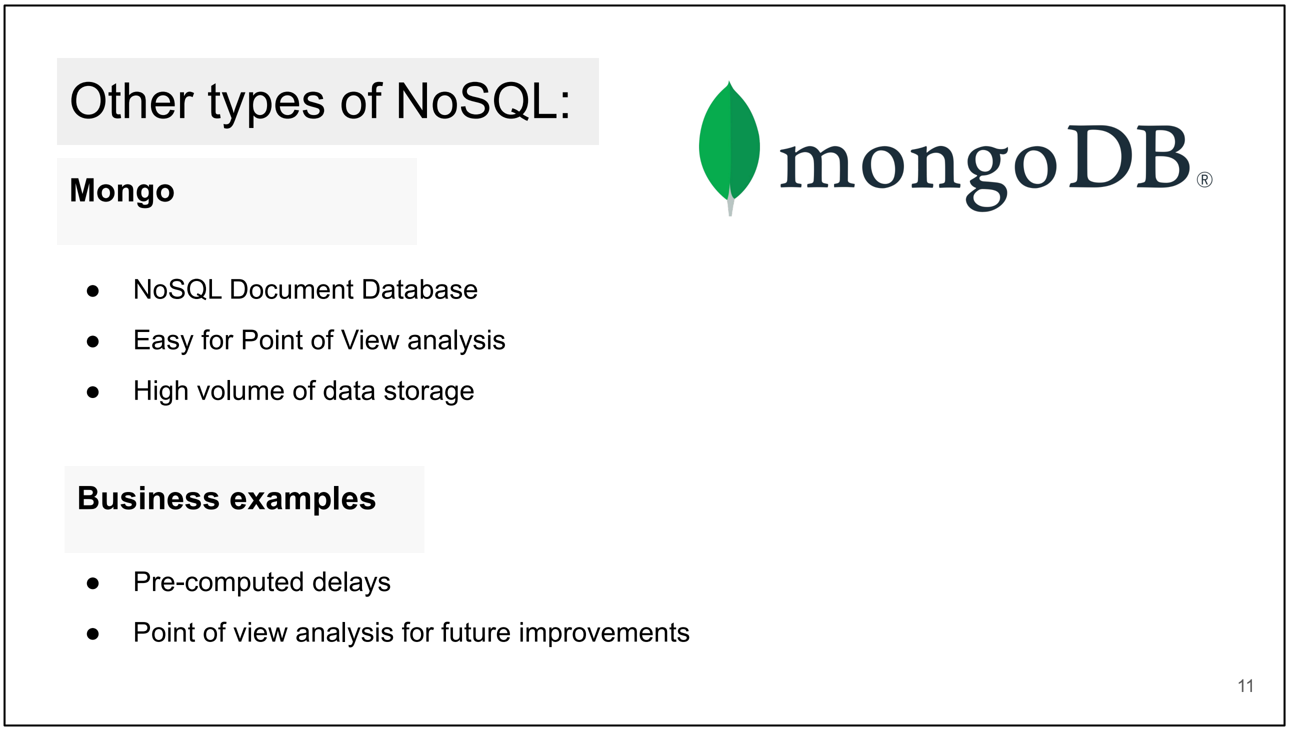
All pair shortest path

Travel time within (mins)	Number of reachable stations	Share of BART stations covered (%)
<=30	50	100%
<=20	44	88%
<=15	41	82%
<=10	32	64%
<=5	16	32%

to station	from station	travel time (mins)
24th Street Mission	16th Street Mission	2.0
North Berkeley	El Cerrito Plaza	3.0
El Cerrito del Norte	El Cerrito Plaza	3.0
Civic Center	16th Street Mission	3.0
Concord	North Concord	3.0
MacArthur	Rockridge	4.0
Fruitvale	Coliseum	4.0
Powell Street	16th Street Mission	4.0
South San Francisco	San Bruno	4.0
SFO	San Bruno	4.0
San Leandro	Coliseum	4.0
Orinda	Rockridge	5.0
Glen Park	16th Street Mission	5.0
12th Street	West Oakland	5.0
Union City	Fremont	5.0
Downtown Berkeley	El Cerrito Plaza	5.0

After selecting the 8 best pickup locations based on louvain modularity and harmonic centrality, we try to check how efficient our distribution system will be.
So, we ran all pair shortest path within BART system. We chose the departure station as one of the 8 pickup locations mentioned in the slides earlier, and filter out the reachable stations with a limit of travel time.

The good news is, we are able to reach all 50 stations on BART network within 30 mins starting from one of our pickup locations. That means any passengers on BART would be able to reach one pick up location within 30mins to get their food. 88% of stations are reachable within 20mins, 82% of them within 15mins. And within 10mins, over 60% are still accessible.
The table on the right hand side is an incomplete screenshot of the most convenient pickup locations from each single station.
Given the results above, we are satisfied with the efficiency of our pickup locations. And our system will make use of the pre-calculated shortest path for distribution.



Other types of NoSQL:

Mongo

- NoSQL Document Database
- Easy for Point of View analysis
- High volume of data storage

Business examples

- Pre-computed delays
- Point of view analysis for future improvements

Apart from Neo4j, other noSQL solutions could also be helpful for the business. MongoDB is very powerful in high volume of data storage and is very easy for Point of View analysis.
For example, it can help store precomputed delays due to pre-announced changes in arrival schedule, due to holidays or special occasions. We can push alerts of potential delays to our clients to manage their expectations. This can also help us improve satisfaction rate despite delays.
Mongo can store multiple points of view (POVs) for analytical purposes. In our business example, using Mongo, we can keep records of each successful or delayed delivery to improve the punctuality of future deliveries. We could also narrow down to see which routes tend to have constant delays, what time during the day delays often happen. This is an advantage of NoSQL document database compared to a relational database, which could be tough for complicated analysis with multiple dimensions.

Other types of NoSQL:

Redis

- NoSQL key value databases
- In memory
- Faster

Business examples

- Last mile drone delivery

Ula: (1:18)
Next we'll talk about Redis. Redis is a no SQL key value database. It uses in memory which means data can be modified or read from the main computer memory as opposed to the much slower disk, therefore it's extremely fast.

If our company ever wants to expand delivery service by adding last mile drone deliveries, we would need to utilize redis. There will be a countable number of drones to deliver meals and the number of drones can easily fit in a scale up in-memory database. We would want to have access to the location of each drone as quickly as possible to be able to display the shipment status to customers. In addition, battery would be limited for drones so it would require careful recharging plan and it'll be crucial for this info to be as accurate as possible therefore speed is key in this scenario.

A relational database will simply not work in this case as it'll take a long time for SQL to provide updates on an individual drone location and we could risk having drones running out of power and landing in unplanned locations, and it would cost a lot more to spend on labour to retrieve these drones. Or we could upset customers who need up to date information and waiting for meals to arrive.

Thank you!