



# How to Get There and What's Along the Way: Driving Directions and Geospatial Search

CPOSC, Oct 17 2009

Eric Beyeler  
Senior Software Engineer  
[ebeyeler@mapquest.com](mailto:ebeyeler@mapquest.com)

The background of the slide features a faded map of San Francisco, showing streets like W 29th Ave, W 27th Ave, and W 25th Ave, as well as landmarks like Hirston Park and Comstock Park. A semi-transparent globe is overlaid on the map, centered over the city. The title "Introduction" is positioned in the upper right area.

# Introduction

Geospatial information is becoming  
integral to our daily lives

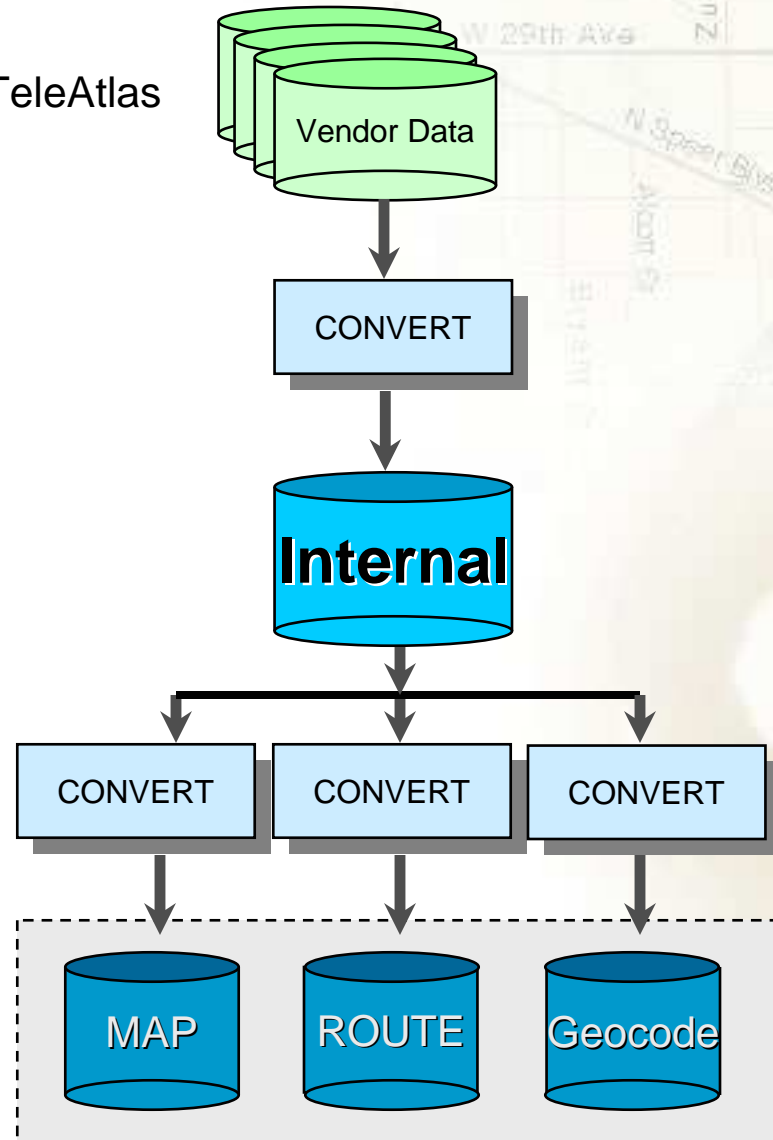


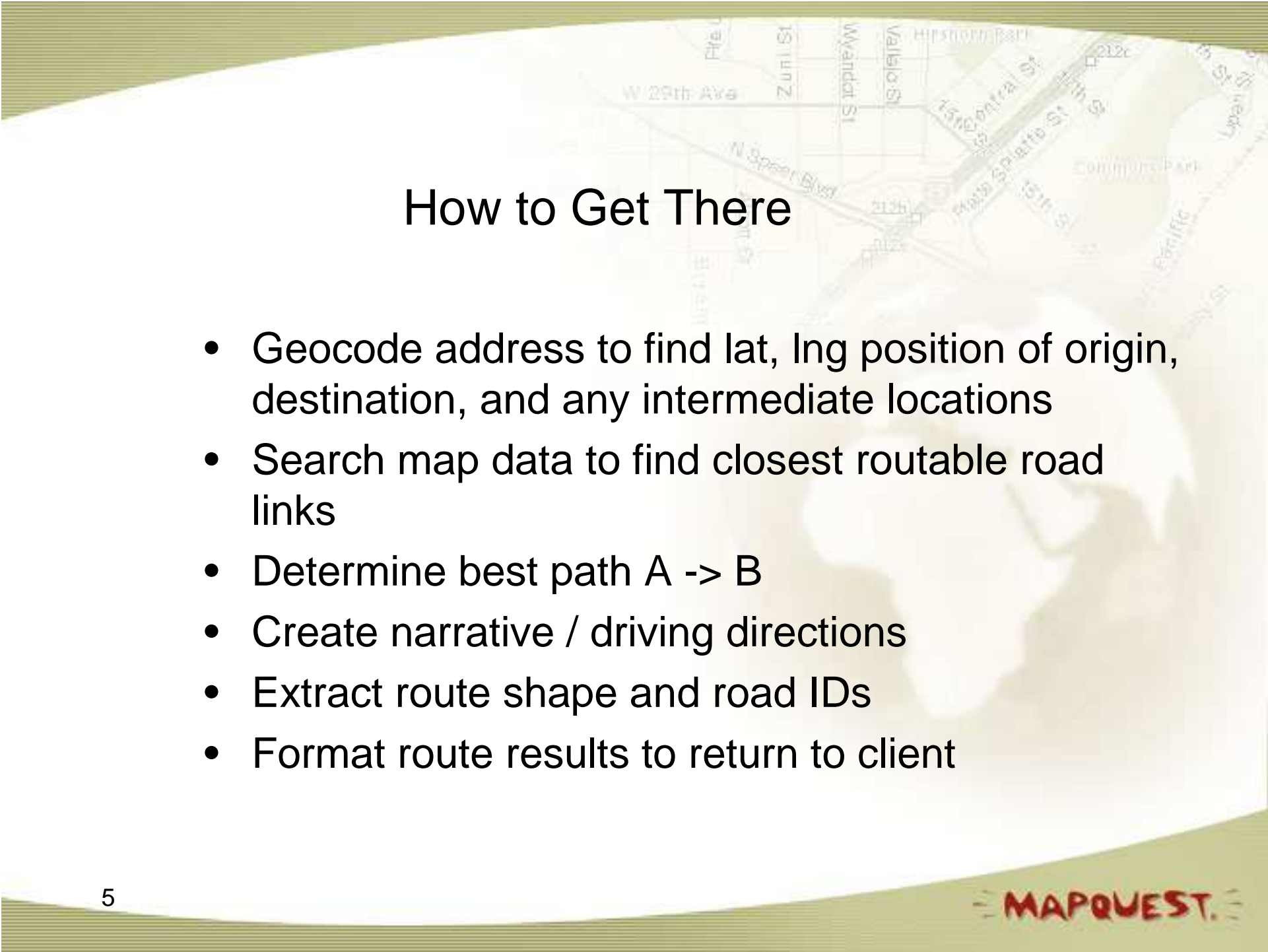
# Introduction

- How to get there: Road network routing
- What's along the way: Geospatial Search
- Testing & Performance
- At MapQuest we use a combination of proprietary and open source technologies

NavTeq, TeleAtlas

# Data Conversion



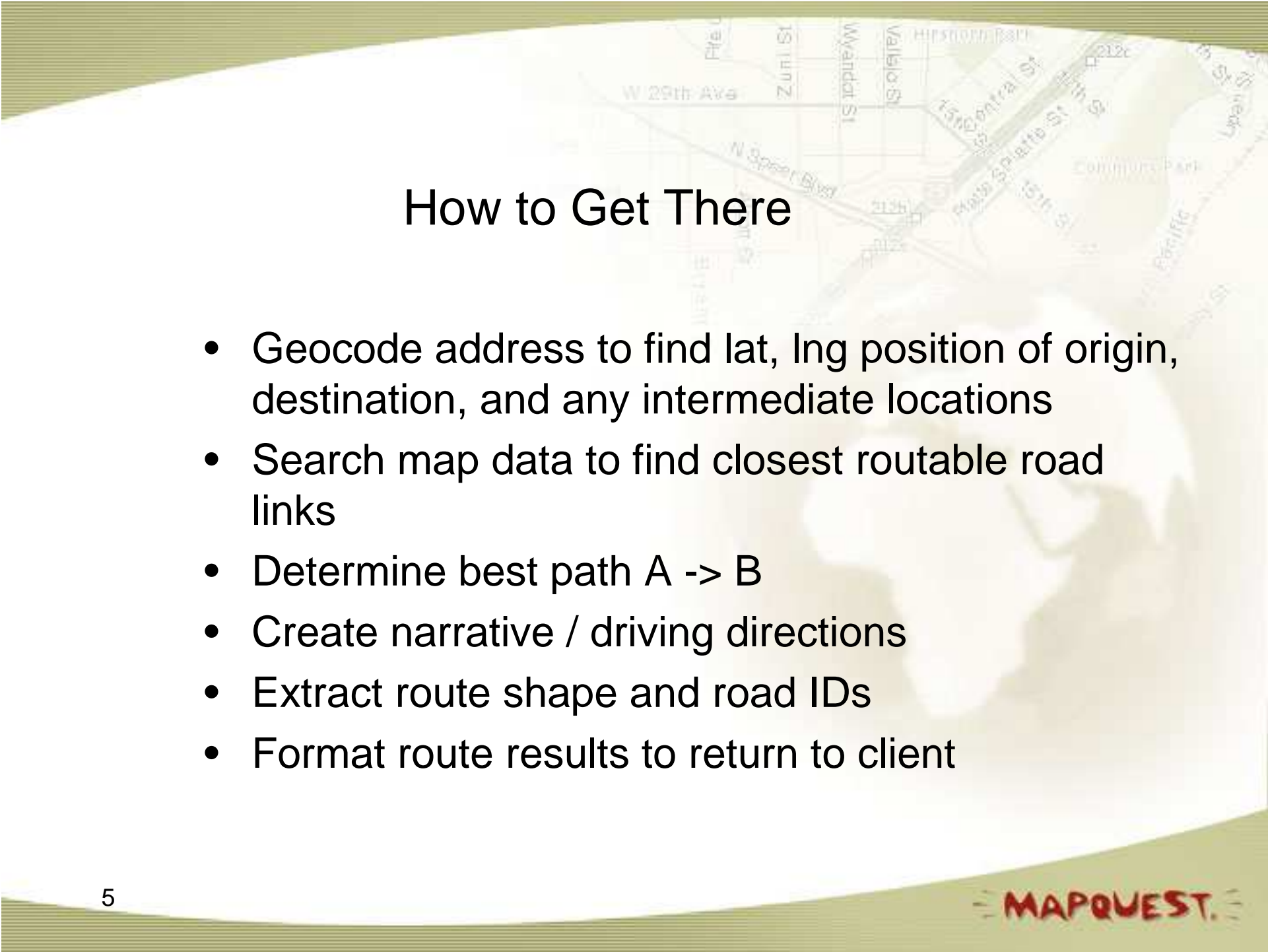


## How to Get There

- Geocode address to find lat, lng position of origin, destination, and any intermediate locations
- Search map data to find closest routable road links
- Determine best path A -> B
- Create narrative / driving directions
- Extract route shape and road IDs
- Format route results to return to client

5

MAPQUEST.

- 
- ## How to Get There
- Geocode address to find lat, lng position of origin, destination, and any intermediate locations
  - Search map data to find closest routable road links
  - Determine best path A -> B
  - Create narrative / driving directions
  - Extract route shape and road IDs
  - Format route results to return to client
- 5
- MAPQUEST.



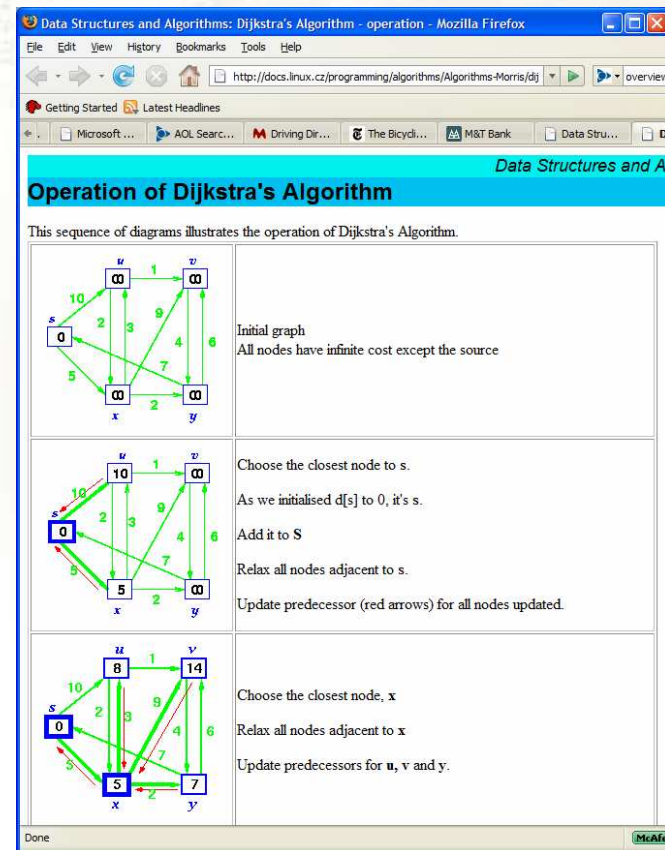
## Overview of Shortest Path Methods

Online illustration of Dijkstra's Algorithm

<http://docs.linux.cz/programming/algorithms/Algorithms-Morris/dijkstra.html>

Some interesting sources:

<http://theory.stanford.edu/~amitp/GameProgramming/>





# Shortest Path Algorithms

- **Network** is defined as a **directed graph**  $G = (N, A)$ 
  - Set of **nodes** (N)
    - Nodes occur at intersections or ends of roads
  - Set of arcs (A) or **links**
    - Link represents a road section from one node to another node
  - Weight (distance, time, other cost metric) of a link connecting nodes Directed from one node to another
- **Shortest path** problem
  - Given a network, find the shortest distance or lowest cost from a set of nodes to another set of nodes

# Shortest Path Algorithms

- **Network** is defined as a **directed graph**  $G = (N, A)$ 
  - Set of **nodes** (N)
    - $n$  = Number of nodes
    - Nodes occur at intersections or ends of roads
  - Set of arcs (A) or **links**
    - $m$  = Number of links
    - Link represents a road section from one node to another node
  - Weight (distance, time, other cost metric) of a link connecting nodes  $u$  and  $v$  is  $w(u,v)$ 
    - Directed from one node to another
- **Shortest path** problem
  - Given a network, find the shortest distance or lowest cost from a set of nodes to another set of nodes
    - **One-to-one**
    - **One-to-some**
    - One-to-all
    - All-to-all (all-paths problem)



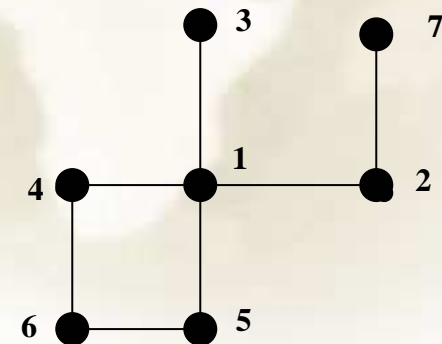
# Evaluation of Shortest Path Algorithms

- In 2000 we reviewed academic approaches and recommendations to the shortest path problem
  - “Shortest Path Algorithms: An Evaluation Using Real Road Networks”
    - Benjamin Zahn, Southwest Texas State University
    - Charles Noon, University of Tennessee
    - One-to-one best solved using **Dijkstra's** algorithm
      - Can be terminated once shortest path to destination node is found
- Variations of Dijkstra algorithm
  - Structure of the priority queue (adjacency list)
    - Implementation recommendation
      - Approximate Double Buckets
  - Storage of set and predecessor information
  - A\*, weighted A\*
  - Dual-ended

# Network Representation

- Network representation impacts shortest path performance
- **Forward star representation**
  - Most efficient data structure
    - Research by Gallo and Pallotino (1988), Ahuja (1993)
  - Two sets of arrays
    - Node data
    - Directed link data
  - Node array maintains index of the first directed link emanating from each node
    - Count of links from that node
  - Directed links maintained in an ordered list
    - Links from nodes are ordered sequentially
      - Grouped together
    - Links contain index of the end node

Nodes			DirectedLinks	
Node #	#Links	Index	End Node	
1	4	1	→	2
2	2	5	→	3
3	1	7	→	4
4	2	8	→	5
5	2	10	→	1
6	2	12	→	7
7	1	14	→	1
			→	1
			→	6
			→	1
			→	6
			→	4
			→	5
			→	2



Portion of Road Network  
Described Above

# Dijkstra Shortest Path Algorithm

- Divides nodes into two sets
  - Done set (or closed set) R
    - Set of nodes whose **final** shortest path distance/cost has been found
      - Permanently labeled
  - Open set  $Q = N - R$
- Open set Q divided into 2 subsets: U and PQ
  - U contains nodes not yet visited
    - Initially contains all nodes except the start node s
  - PQ is the priority queue or adjacency list (temporarily labeled)
    - Contains nodes that are **adjacent** to nodes in the done set
    - Nodes are added to PQ when they are first visited
- Node selection rule
  - Selects the node in the adjacency list with **lowest cost** (distance, time)
    - Permanently “labels” it - adds to the done set
      - Its optimal shortest path from the source node is then known

# Dijkstra Algorithm Pseudo Code

```
Dijkstra(Network N, LinkWeights w, Node s, Node d)
{
    Node u,v;
    Initialize(R)                // Done set-set to empty
    Initialize(PQ)               // Set adjacency list to empty
    for all neighbors v of s
        PQ.Add(v, w(s,v))
    while (!PQ.Empty())
    {
        u = PQ.ExtractMin()      // Get the lowest cost node from adjacency list
        R.Add(u)                 // Add u to the done set
        if (u == d)              // Found the destination
            return;
        for all neighbors v of u
        {
            if v is in R:        // Shortest path to this node already found
                continue
            d = g[u] + w(u, v)
            if v is not in PQ:
                PQ.Add(v, d)      // Add to adjacency list
            else if (g[v] > d)    // New path yields a lower cost
                PQ.Decrease(v, d) // Update predecessor and cost for this node in adj. list
        }
    }
}
```

# Adjacency List Structures

- Dijkstra performance highly depends on adjacency list structure
  - Data structure and method to sort nodes and find lowest cost node
- Naïve implementation - unsorted list  $O(n^2)$ 
  - Examines all nodes in adjacency list at each iteration

- Heap sort methods

Standard	$O(m \log(n))$	m is number of links in network
Fibonacci	$O(m + n \log(n))$	
R	$O(m + n \log(C))$	C is the maximum link length

- Bucket structures

- Each bucket stores nodes with distance within a range b
  - Maintained in a FIFO queue within each bucket

Dial's Algorithm	$O(m + nC)$	ith bucket contains nodes with cost i
		Prohibitive memory requirement ( $nC$ buckets)
Dial's algorithm (overflow)	$O(m + n(C/b + b))$	Overflow bag to limit memory
		Nodes moved from overflow to low level buckets
Approximate buckets	$O(mb + n(b + C/b))$	Bucket contains nodes with cost $[ib, (i+1)b - 1]$
		b is the size of the bucket
Double buckets	$O(m + n(b + C/b))$	2 levels of node containers: upper level and lower level
		Nodes moved from upper to lower level



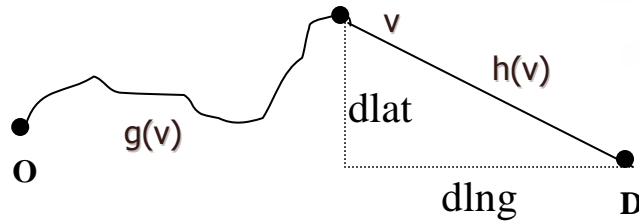
# A\* and Weighted A\* Algorithms

- **A\*** originated as an artificial intelligence search algorithm
  - Hart, Nilsson, and Raphael 1982
  - Narrows the search by guiding the search towards the destination
- Adds heuristic to the Dijkstra shortest path distance
  - $f(v) = g(v) + h(v)$ 
    - $g(v)$  = actual cost from origin to node
      - Cost to previous node + link cost
    - $h(v)$  = heuristic: estimate of cost from node to the destination
- **Weighted A\*** applies a weight ( $W$ ) to the heuristic
  - $f(v) = (1-W) * g(v) + W * h(v)$
  - $W$  in range  $[0,1]$ 
    - $W = 0$ : standard Dijkstra breadth first search
    - $W = 1$ : best first search
      - At each node will get you closer to the destination
- Validity of A\*
  - If  $h(v)$  underestimates actual cost to destination then A\* will produce an optimal shortest path just like Dijkstra
  - Heuristic must also be consistent / monotonic

[http://en.wikipedia.org/wiki/A-star\\_algorithm](http://en.wikipedia.org/wiki/A-star_algorithm)



# A\* Heuristic



Heuristic  $f(v)$  for sorting nodes in adjacency list

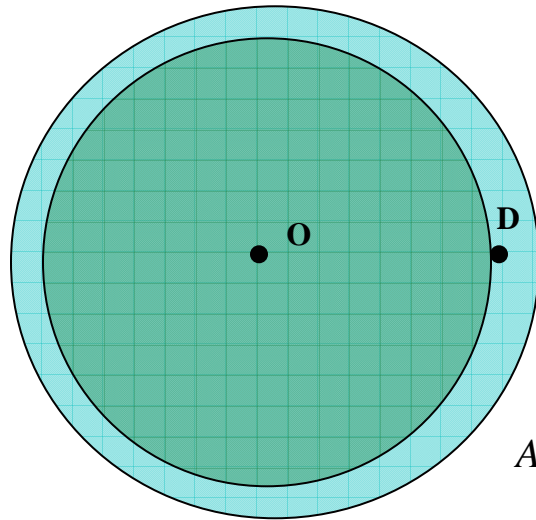
$$f(v) = g(v) + h(v)$$

$g(v)$  = Actual cost from origin to node  $v$

$h(v)$  = **Estimate** of cost from node  $v$  to destination

- A\* heuristic
  - Shortest distance routes: use distance
  - Direct (shortest time) routes: use time estimate
    - Distance / maximum speed
      - Avoid overestimating the time to destination
- Performance depends on cost of the heuristic
  - Distance between 2 lat/lng positions requires 6 trig functions
    - Spherical trigonometry
    - 2 trig expressions are constant since destination is fixed
  - Use an approximate distance between 2 lat/lng
    - $\text{sqrt}(\text{dlatMiles} * \text{dlatMiles} + \text{dlngMiles} * \text{dlngMiles})$ 
      - Uses a table of miles per longitude degree (indexed by latitude)

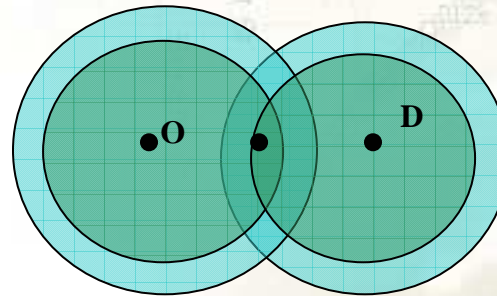
# Shortest Path Algorithm Comparison



$$Area \cong \pi r^2$$

## Single Tree Dijkstra

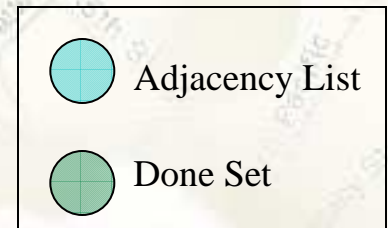
Spreads out uniformly from origin. At each iteration it adds the node in the adjacency list with least cost to the done set and evaluates all links from that node. Nodes at the end of these links are added to the adjacency if not already in the list. If the node is in the adjacency list and the new path yields a lower cost, the adjacency list is updated with new cost and predecessor information. Shortest path is found when the destination node is added to the done set.



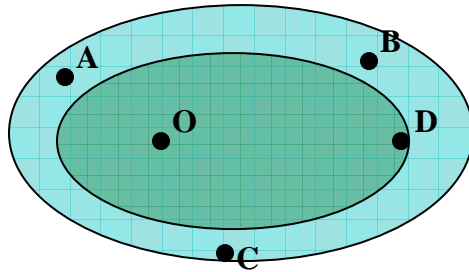
$$Area \cong 2\pi \left(\frac{r}{2}\right)^2 = \frac{\pi r^2}{2}$$

## Two Tree Dijkstra

Dual ended search where one search emanates from the origin and the other from the destination. Each search spreads out uniformly and acts similarly to the single tree Dijkstra search. At each iteration the node with least cost (from either the origin or destination tree) is evaluated. The search ends when a node is in the done set of both the origin and destination tree. Since the search is ordered by least cost we know that the shortest path is through the common node.



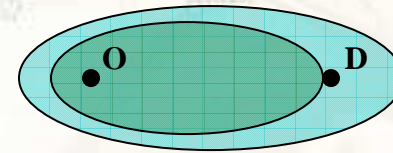
# Shortest Path Algorithm Comparison



A\*

Spreads out from origin but uses a heuristic of predicted cost to the destination. Nodes in the adjacency list are sorted by the addition of cost from the origin to the node plus a prediction of cost to the destination. In the diagram above, assume nodes A,B,and C all have equal cost from the origin. Node B will be selected first since it is closest to the destination. The heuristic tends to guide the search towards the destination.

$$f(v) = g(v) + h(v)$$



Weighted A\*

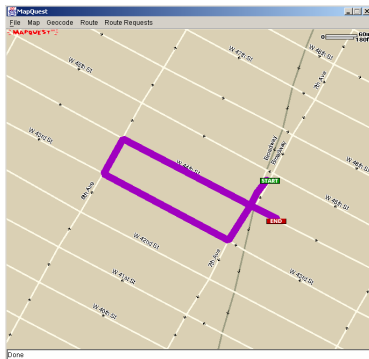
Same idea as A\*, except weights are applied to the cost to the node and to the predicted cost to the destination. Higher weights to the heuristic narrow the search but may exclude a less direct path over higher speed roads.  $W = 0$  yields Dijkstra and  $W=1$  yields a Best First Search.

$$f(v) = (1-W) * g(v) + W * h(v)$$

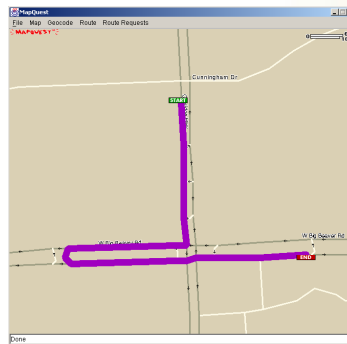
# Link-Based Dijkstra Algorithm

- *Standard* Dijkstra method marks nodes
  - Cannot traverse same node twice in the path
    - Once shortest path to a node is found it is placed in the done set
  - Turn restrictions on at-grade intersections cause problems
- To solve turn restriction problems
  - Store **directed links** in adjacency and done set
    - Expand at the end node of the directed link
  - Allows traversal though an intersection more than once
    - With different directed links

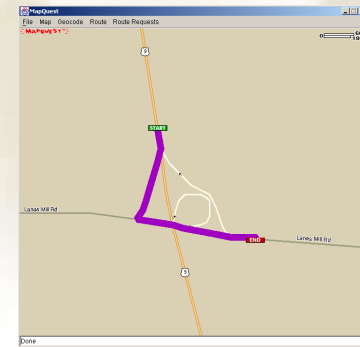
Paths Below Fail (or take a roundabout path) with Node-Based Method



Turn Restriction-NY City



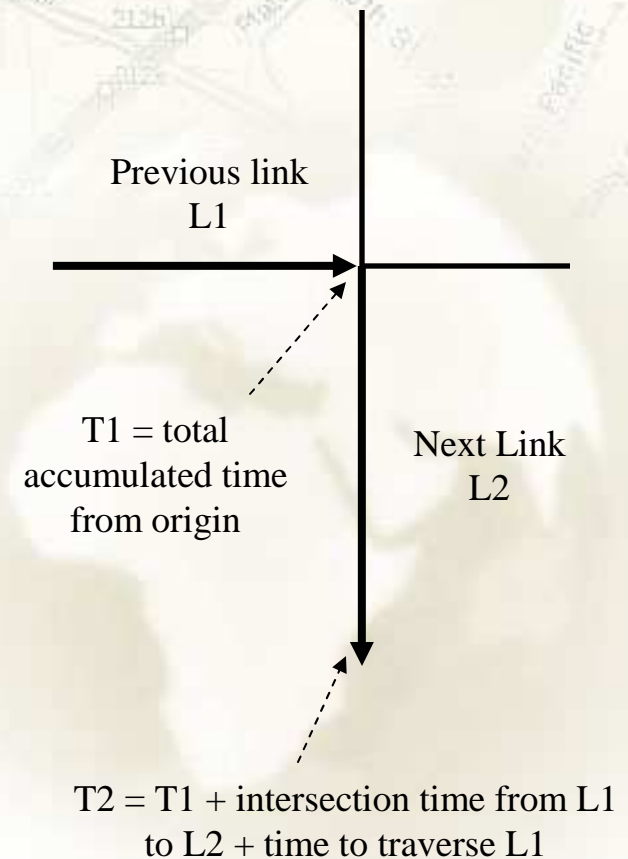
Michigan Left



New Jersey Cup Handle

# Intersection / Turn Costs

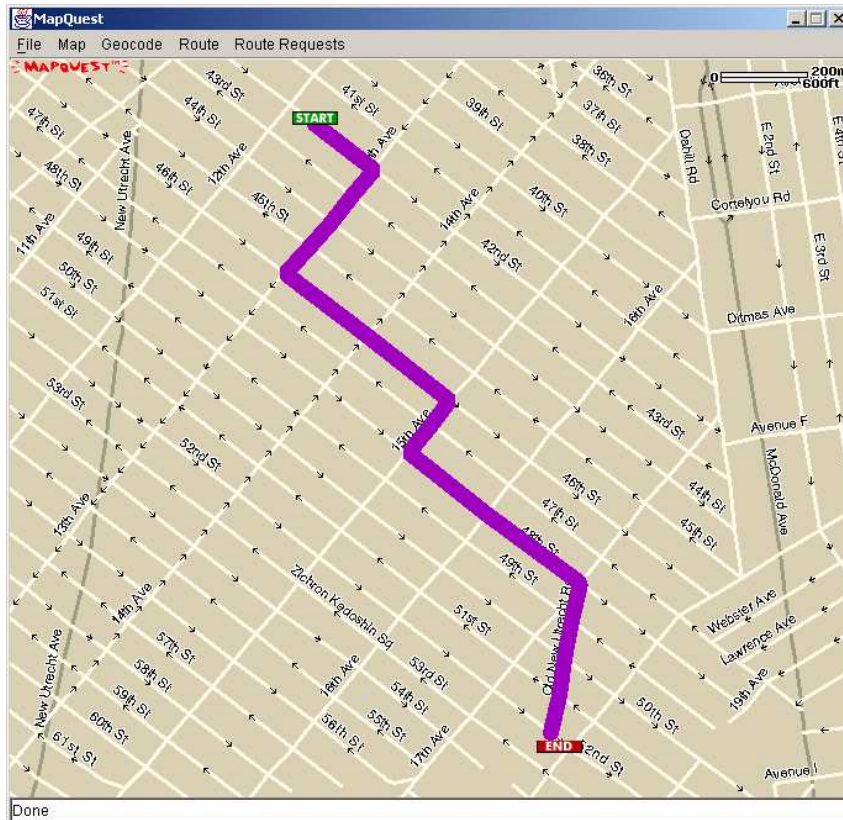
- Improves route quality
  - Reduce number of left turns
  - Reduce maneuvers
    - Simplified directions
  - Solve some problem routes
    - Local lanes on I-270
  - Identify and avoid gates, private roads, unpaved roads, etc.
    - Apply time penalty to enter
  - Allow more realistic use of road speed for shortest path determination
- General method
  - Apply time at intersections
  - Shortest path algorithm considers time at intersection plus time to traverse link (distance/speed)



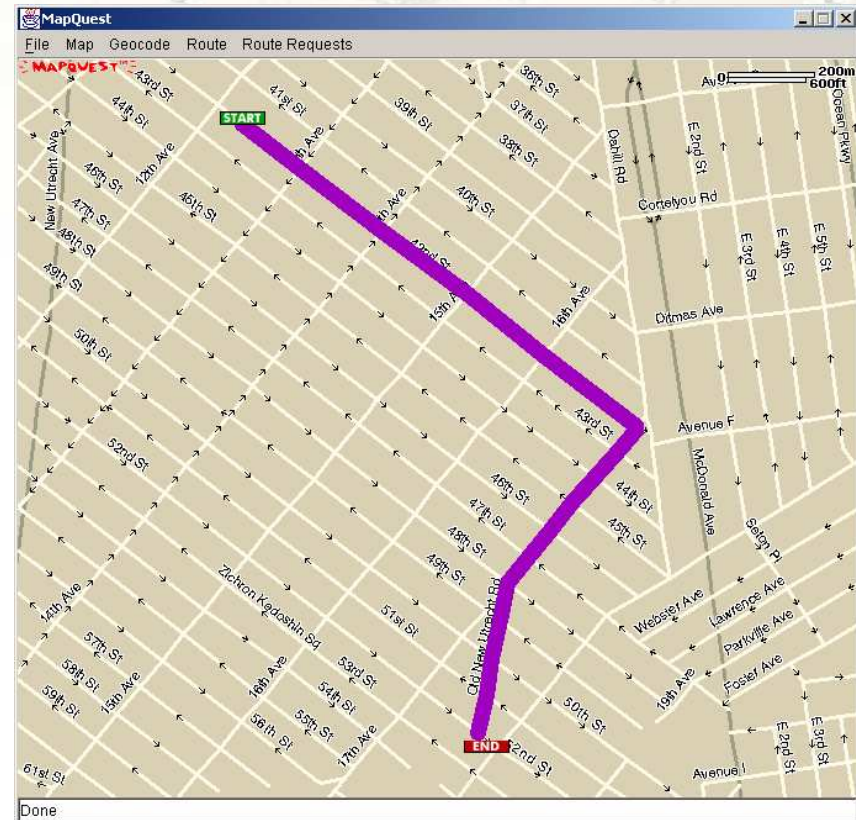


# Route Improvements in Downtown Areas

New York City: 42<sup>nd</sup> St near 12<sup>th</sup> Ave. to 52<sup>nd</sup> St near 18<sup>th</sup> Ave.



Without Turn Costing: 5 turns (2 left)



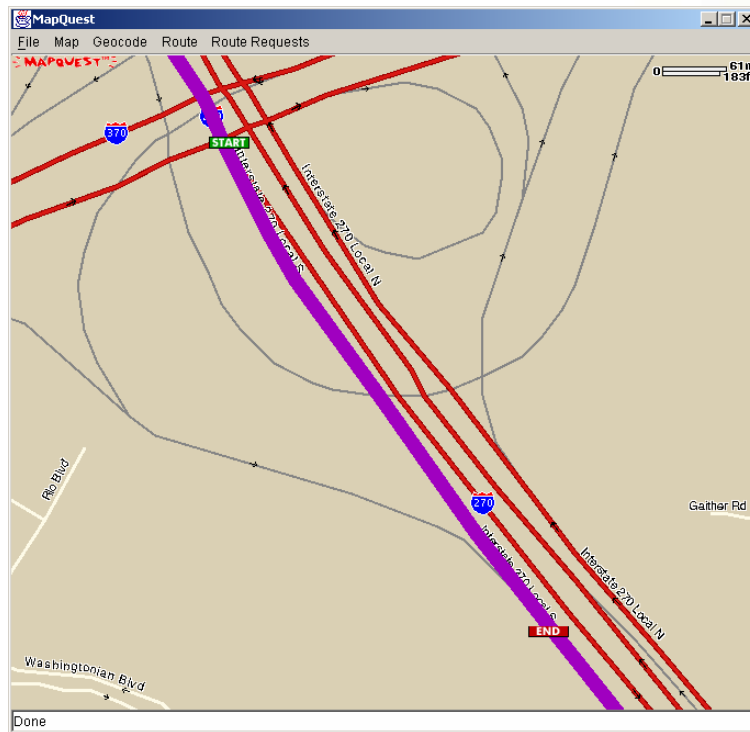
With Turn Costing: 2 turns (1 left)

Without turn costing: shortest path method tends to follow shorter distance (diagonal) path  
More turns, more complex route narrative.



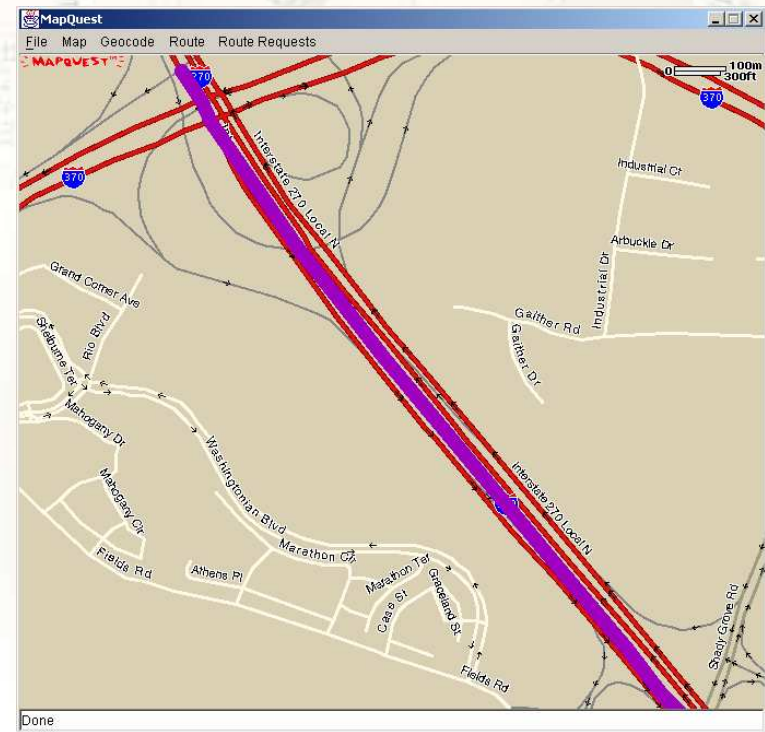
# Favor Roads with Fewer Intersections

Frederick, MD to Washington DC



## Without Turn Costing

Uses I-270 Local Lanes. Slightly shorter distance, but not preferred. Adds complexity.



## With Turn Costing

Route uses regular I-270 Lanes. Less complex narrative.

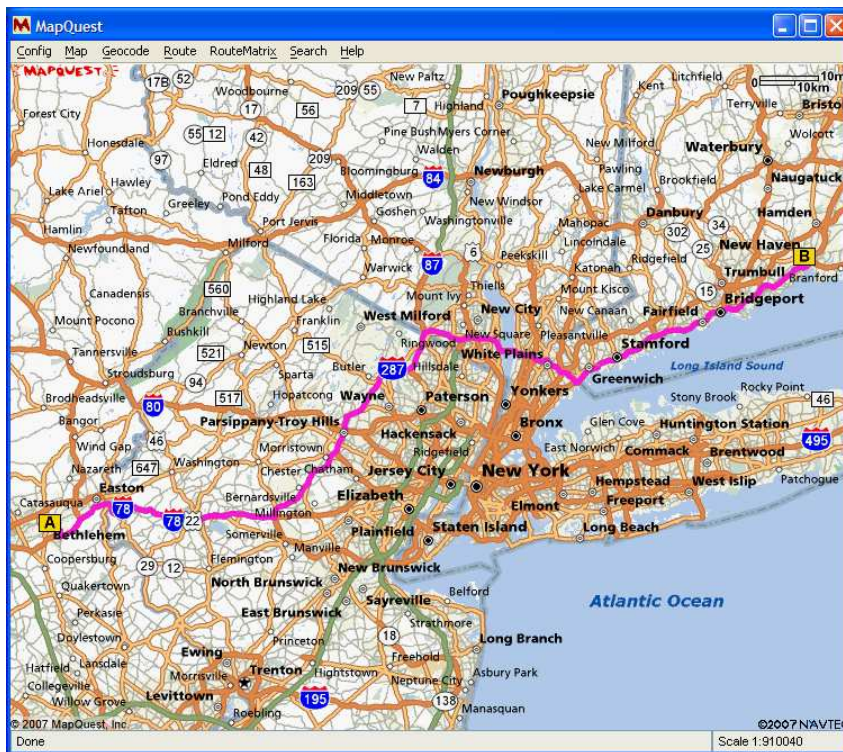
**Addition of intersection costs – routes favor paths with fewer intersections.**

# Assignment of Intersection Costs

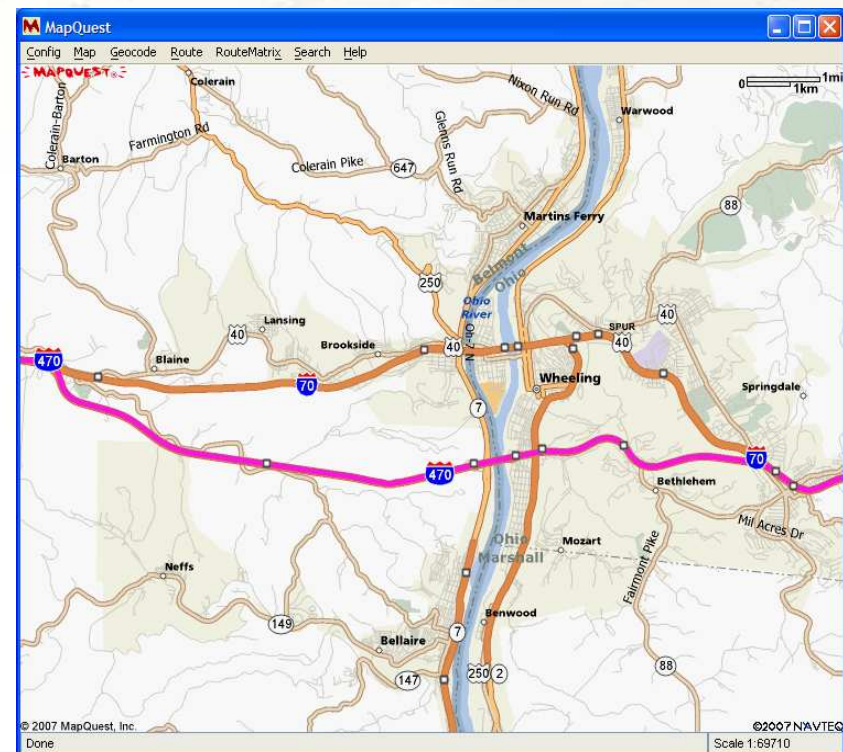
- Intersection costs are assigned during data conversion
- Cost factor depends on:
  - Link geometry
    - Turn direction (left, right, straight) and degree (sharp vs. slight)
    - Number and orientation of links at the intersection
  - Likelihood of stop
    - High cost to cross or make lefts onto higher class roads
    - NOTE: NavTeq plans to add stop sign and traffic light data in the future
  - Ability to combine links into a single maneuver
    - Increase costs where maneuver changes occur
    - Routes may tend to stay on same roads longer
      - Support for “simple” routes with less turns/maneuvers
    - Apply maneuver change cost to exits and not merges
      - So we do not doubly penalize use of ramps
  - Special values for restricted turns, gates, unpaved, others

# Urban Avoidance / Density Weighting

- Slightly favor rural areas vs. urban areas
  - Apply weight factor to speed based on density of links



Avoid I-95 near NYC



Help avoid highways through city centers in favor of bypasses





# DDF Optimizations

- Attempt to keep data needed by algorithm in close proximity on disk
  - Node and link data is tiled
    - Segmented by lat,lng of begin node
    - Data for short routes stored in smaller number of pages on disk
  - Nodes within a tile are sorted by highest artery level
- Directed links are sorted by artery level
  - Early exit when expanding a node
    - Once maximum artery level is reached no need to access other directed links from that node
- Aggregation
  - Combine links at nodes where only 2 links connect
    - When important routing attributes are the same
      - “False” nodes
    - About 8% of NavTeq NA links can be aggregated

# Narrative Generation

The goal is to accurately identify what information is needed for a driver while detecting and eliminating extraneous information

25

MAPQUEST.

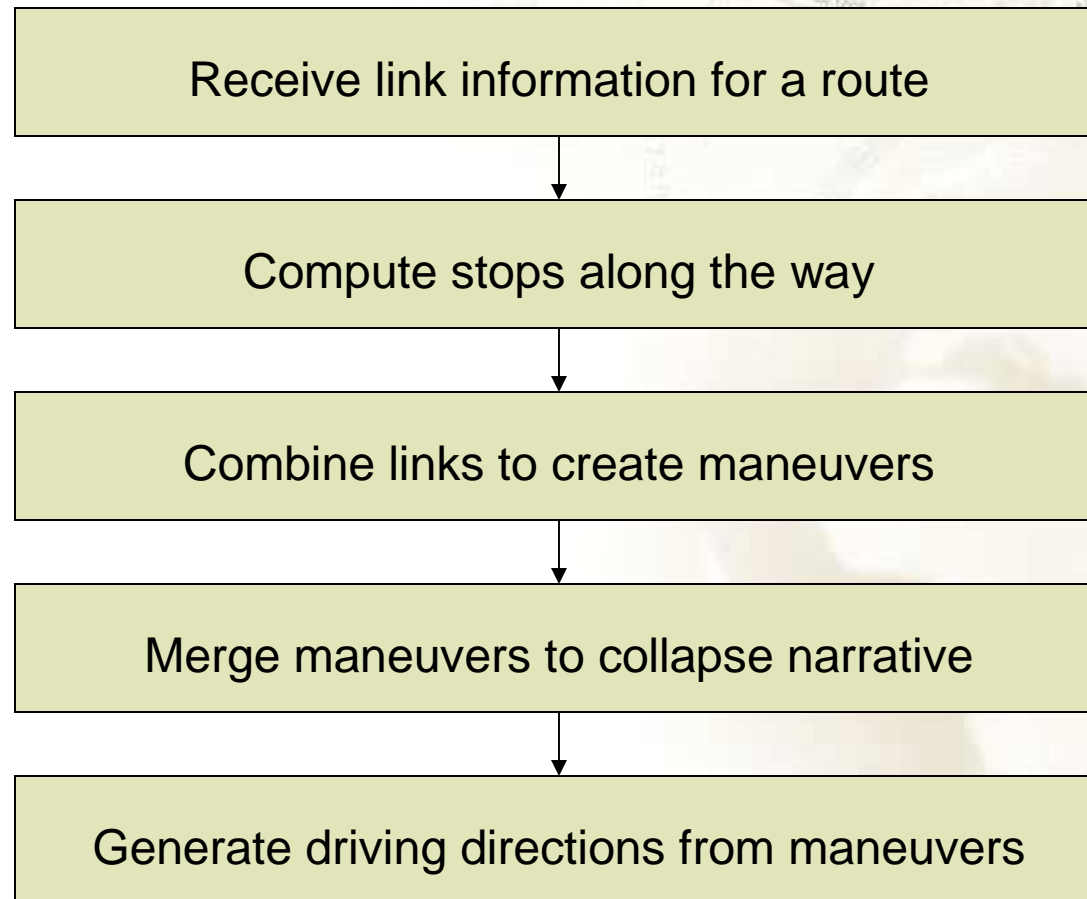
# Narrative Generation

The goal is to accurately identify what information is needed for a driver while detecting and eliminating extraneous information

25

MAPQUEST.

# Narrative Generation Flow





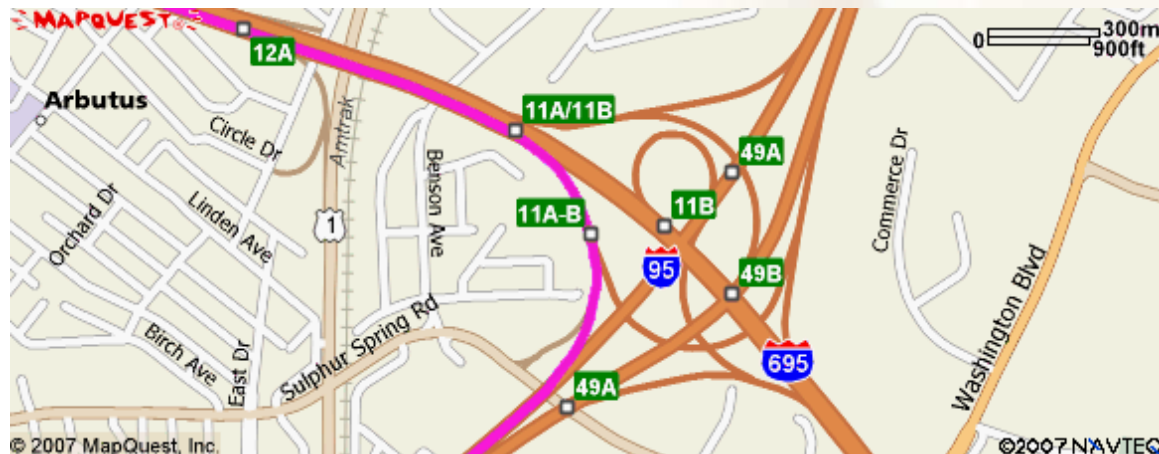
# Succinct Driving Direction Example

## BEFORE













- 5: Take the I-95 N/I-95 S exit, exit number 11A/11B, towards BALTIMORE/WASHINGTON.
- 6: Keep RIGHT at the fork in the ramp.
- 7: Keep LEFT at the fork in the ramp.
- 8: Merge onto I-95 S.

## AFTER

- 5: Merge onto I-95 S via EXIT 11B toward WASHINGTON.



# Final Result

	1: Start out going NORTH on <b>BRASS MILL RD</b> toward PHILADELPHIA RD/MD-7.	0.2
 	2: Turn LEFT onto <b>PHILADELPHIA RD/MD-7 S.</b>	1.0
 	3: Turn RIGHT onto <b>MD-543 N/CRESWELL RD.</b>	0.3
 	4: Merge onto <b>I-95 S</b> via the ramp on the LEFT (Portions toll).	52.6
 	5: Merge onto <b>I-495 W/CAPITAL BELTWAY</b> via <b>EXIT 27</b> toward SILVER SPRING (Crossing into VIRGINIA).	30.1
 	6: Merge onto <b>I-95 S</b> via <b>EXIT 57A</b> toward RICHMOND.	85.8
 	7: Merge onto <b>I-295 S</b> via <b>EXIT 84A</b> on the LEFT toward ROCKY MT NC.	14.7
 	8: Merge onto <b>I-64 E</b> via <b>EXIT 28A</b> toward NORFOLK/VA BEACH.	83.6
 	9: Merge onto <b>I-264 E</b> via <b>EXIT 284A</b> toward VA BEACH.	12.7



## Tuning Routes – The Secret Sauce

- Route types
  - Direct / shortest time
  - Shortest distance
- Speed
- Route options / controls



# Route Types

- Direct / shortest time
  - Uses link speeds to compute time along the road link
    - Some route options modulate speeds
  - Uses turn costs and artery filtering
- Shortest distance
  - Distance and turn costs are considered
  - May not be absolute shortest path
  - Often leads to undesirable routes
    - Can take short cut paths through neighborhoods and mountain passes
    - Often leads to large number of maneuvers
- Artery filtering is used for both types
  - Less aggressive artery filtering is used for shortest distance
- Pedestrian
  - No artery filtering, turn restrictions, or one-way restrictions
  - Avoids limited access and allows use of walkways



## Speed

- Compute time using link length and speed
  - Rather than storing a time cost in the data
  - Allows more flexibility
    - e.g. using time dependent speeds
- Route quality is highly dependent on vendor speed data
  - Speed limit
    - Exact but only available on FC 1 and 2 roads in NavTeq data
  - Speed category – range of speeds
    - e.g., 55-70 MPH
      - Assign single speed
  - Limit speeds based on road class
    - Prevent “bad” speed data
    - Have widened the limits as vendor data quality has improved
- Investigating road curvature to reduce speeds
  - Analyze shape to identify switchbacks/highly curved roads



## Testing and Quality Analysis

What determines a good route?

- Time
- Distance
- Number of maneuvers
- Types of turns
- Road class
- Urban Area
- Detailed data



# Automated Testing

- Route quality tests executed in the Fitnesse framework
  - Web interface on top of Fit
    - <http://fitnesse.org/>
- Fitnesse tests scheduled and archived through CruiseControl
  - <http://cruisecontrol.sourceforge.net/>
- Selenium is used for front-end web page testing
  - <http://seleniumhq.org/>

# Fitness

## PinpointTests.AvoidMountainPassPinpoints

Import

com.mapquest.route.fit

► *Avoid Mountain Pass: Description*

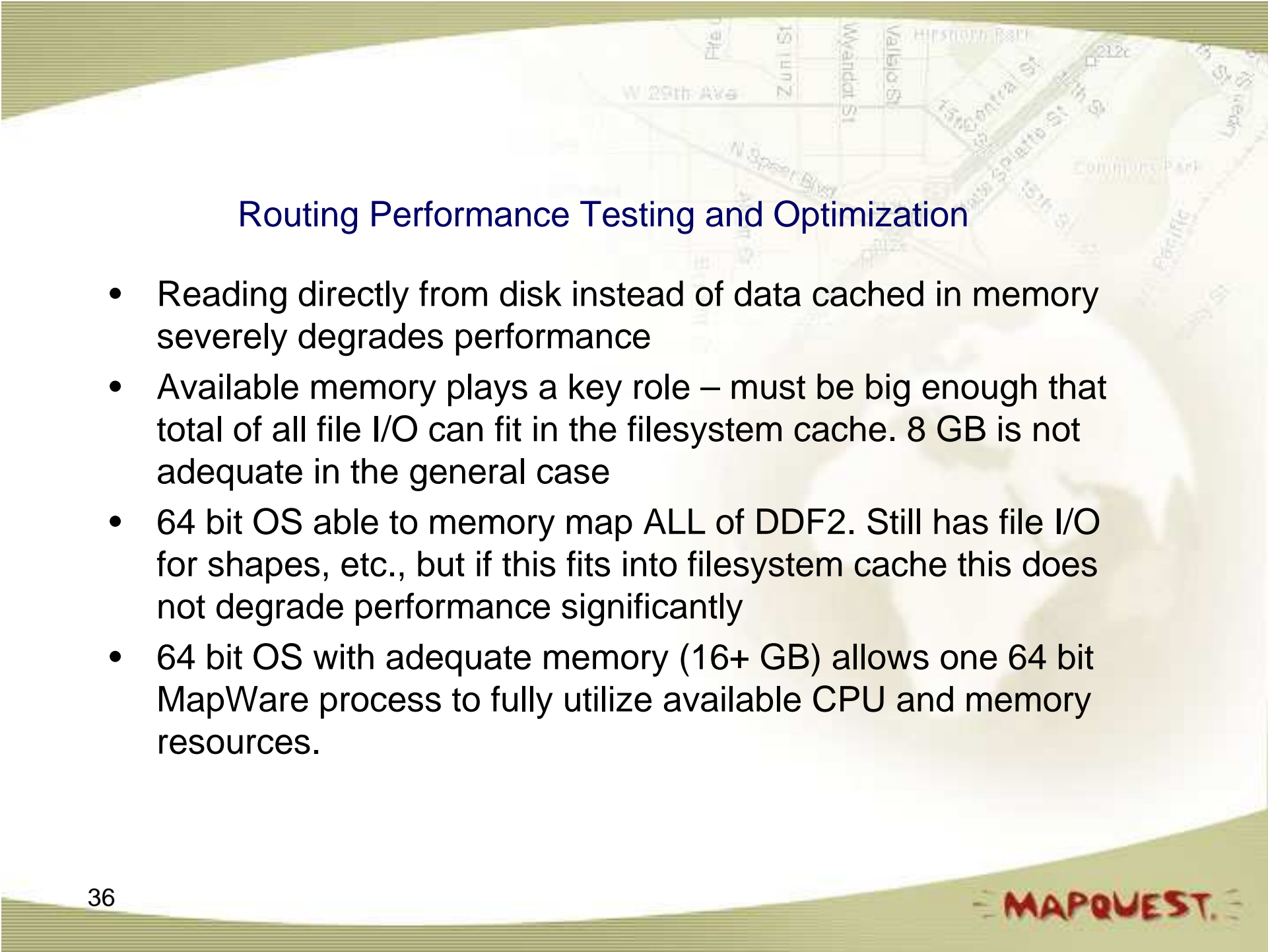
[Expand All](#) | [Collapse All](#)

AvoidMountainPassPinpointTest routeTestData/properties/run.properties											
id	oLat	oLng	oStreet	oCity	oState	dLat	dLng	dStreet	dCity	dState	getDistance?
0	40.03365	-77.298588	Pine Grove Rd	Pine Grove Furnace	PA	40.068592	-77.384472	High Mountain Rd	Walnut Bottom	PA	<b>13.035997</b> > 12.0
1	37.882891	-79.015369	Love Rd	Love	VA	37.842658	-79.026997	Crabtree Falls	blank	VA	<b>22.084</b> > 20.0
2	37.981029	-78.954925	blank	Sherando	VA	37.981869	-78.907376	Blue Ridge Pkwy	Humpback Rocks	VA	<b>14.601997</b> > 14.0
3	37.9375	-107.811699	blank	Telluride	CO	37.811901	-107.663902	blank	Silverton	CO	<b>73.171005</b> > 60.0
4	37.508833	-119.965927	140; State Route 140; State Highway 140	blank	CA	37.537735	-119.656082	Forest Dr	blank	CA	<b>49.261997</b> > 45.0

# Performance

## Motivation for 64 Bit MapWare

- We wanted to lessen the amount of time spent trying to fit data structures into 4GB (32bit address space). This allows us to load more data sets into 1 server and not spend time working around the 4GB single process address space limit.
- Better developer throughput and process management are the first tangible benefits of this project. The real benefits come in the future since now that we are on a 64bit platform we are poised for growth.
- Fully take advantage of new hardware advances

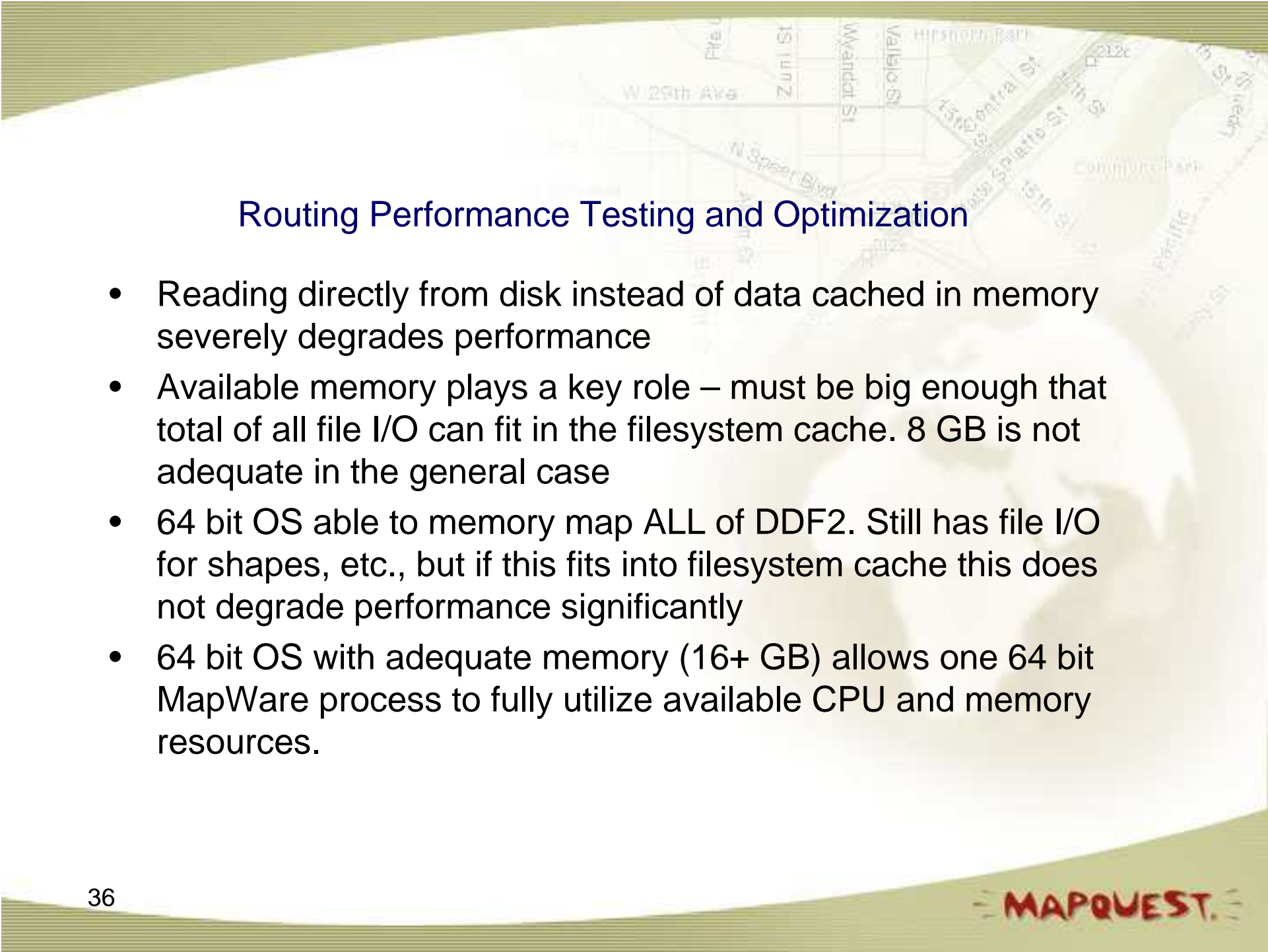


## Routing Performance Testing and Optimization

- Reading directly from disk instead of data cached in memory severely degrades performance
- Available memory plays a key role – must be big enough that total of all file I/O can fit in the filesystem cache. 8 GB is not adequate in the general case
- 64 bit OS able to memory map ALL of DDF2. Still has file I/O for shapes, etc., but if this fits into filesystem cache this does not degrade performance significantly
- 64 bit OS with adequate memory (16+ GB) allows one 64 bit MapWare process to fully utilize available CPU and memory resources.

36

MAPQUEST

- 
- ## Routing Performance Testing and Optimization
- Reading directly from disk instead of data cached in memory severely degrades performance
  - Available memory plays a key role – must be big enough that total of all file I/O can fit in the filesystem cache. 8 GB is not adequate in the general case
  - 64 bit OS able to memory map ALL of DDF2. Still has file I/O for shapes, etc., but if this fits into filesystem cache this does not degrade performance significantly
  - 64 bit OS with adequate memory (16+ GB) allows one 64 bit MapWare process to fully utilize available CPU and memory resources.
- 36
- MAPQUEST

## Routing Performance Testing and Optimization

IntelXeon5150@2.66GHz, 64bit RHEL 5, 8 GB, 4 processors  
US 8K route set; numbers in routes calculated per second

MW threads	32 bit MapWare	64 bit MapWare
4	127/sec; 32/s/proc	178/sec; 44/s/proc

- These tests were a best-case scenario run, 16 GB machines would be needed for real-world duplication of results
- Processors were running at approx 75-80% utilization during these tests
- Also ran on a 16 GB machine with 64 bit MapWare; throughput for 4 threads: 195/sec
- Apples to apples comparison:  
64 bit MapWare has ~40% more throughput for routing



# Artery Level Comparisons

## Directed Link Counts by Artery Level (NavTeq Functional Class)

Artery level	North America		
1		0.50%	
2		1.68%	
3		3.56%	
4		10.25%	
5		84.02%	
Total			

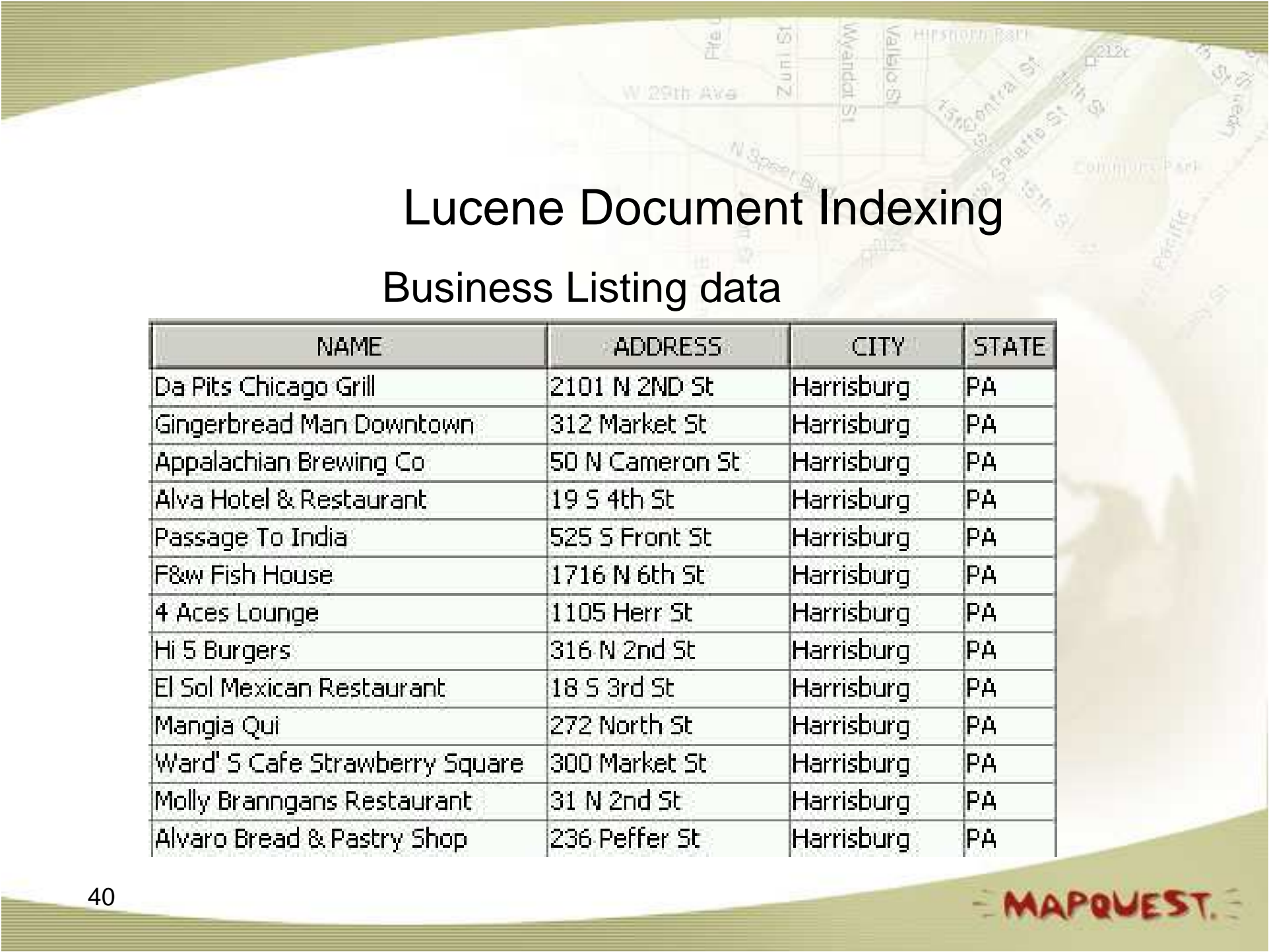
## Sizes and Counts

North America		
Node Count	20 million	
nodeInfo.rte	317 MB	
directedLink.rte	1.2 GB	
GEFToRoute.db	308 MB	
linkInfo.rte	925 MB	
Shape.db	1.3 GB	



## Geospatial Searching

- Lucene text search engine
  - <http://lucene.apache.org/>
- SOLR management layer
  - <http://lucene.apache.org/solr/>
  - Web API / caching / replication features
  - Runs under Tomcat <http://tomcat.apache.org/>
- Spatial grid / indexing within Lucene
  - Geohash - <http://en.wikipedia.org/wiki/Geohash>
  - Lucene implementation  
<http://sourceforge.net/projects/locallucene/>

A faint background map of Harrisburg, Pennsylvania, showing streets like W 29th Ave, Zuni St, Wyandot St, Vallejo St, Hirston Rd, Central St, Spratle St, 15th St, and 17th St. It also shows landmarks like Hirston Park and Conant Park.

# Lucene Document Indexing

## Business Listing data

NAME	ADDRESS	CITY	STATE
Da Pits Chicago Grill	2101 N 2ND St	Harrisburg	PA
Gingerbread Man Downtown	312 Market St	Harrisburg	PA
Appalachian Brewing Co	50 N Cameron St	Harrisburg	PA
Alva Hotel & Restaurant	19 S 4th St	Harrisburg	PA
Passage To India	525 S Front St	Harrisburg	PA
F&w Fish House	1716 N 6th St	Harrisburg	PA
4 Aces Lounge	1105 Herr St	Harrisburg	PA
Hi 5 Burgers	316 N 2nd St	Harrisburg	PA
El Sol Mexican Restaurant	18 S 3rd St	Harrisburg	PA
Mangia Qui	272 North St	Harrisburg	PA
Ward' S Cafe Strawberry Square	300 Market St	Harrisburg	PA
Molly Brannigans Restaurant	31 N 2nd St	Harrisburg	PA
Alvaro Bread & Pastry Shop	236 Pepper St	Harrisburg	PA

# Lucene Document Indexing

Based on an “inverted index”

For every term, list all documents that contain that term.

<u>Pizza</u>	<u>Hut</u>
10	139
234	234
568	987
987	

# Lucene Document Searching

Create a “term vector” based on documents that match all criteria

Pizza

10

234

568

987

Hut

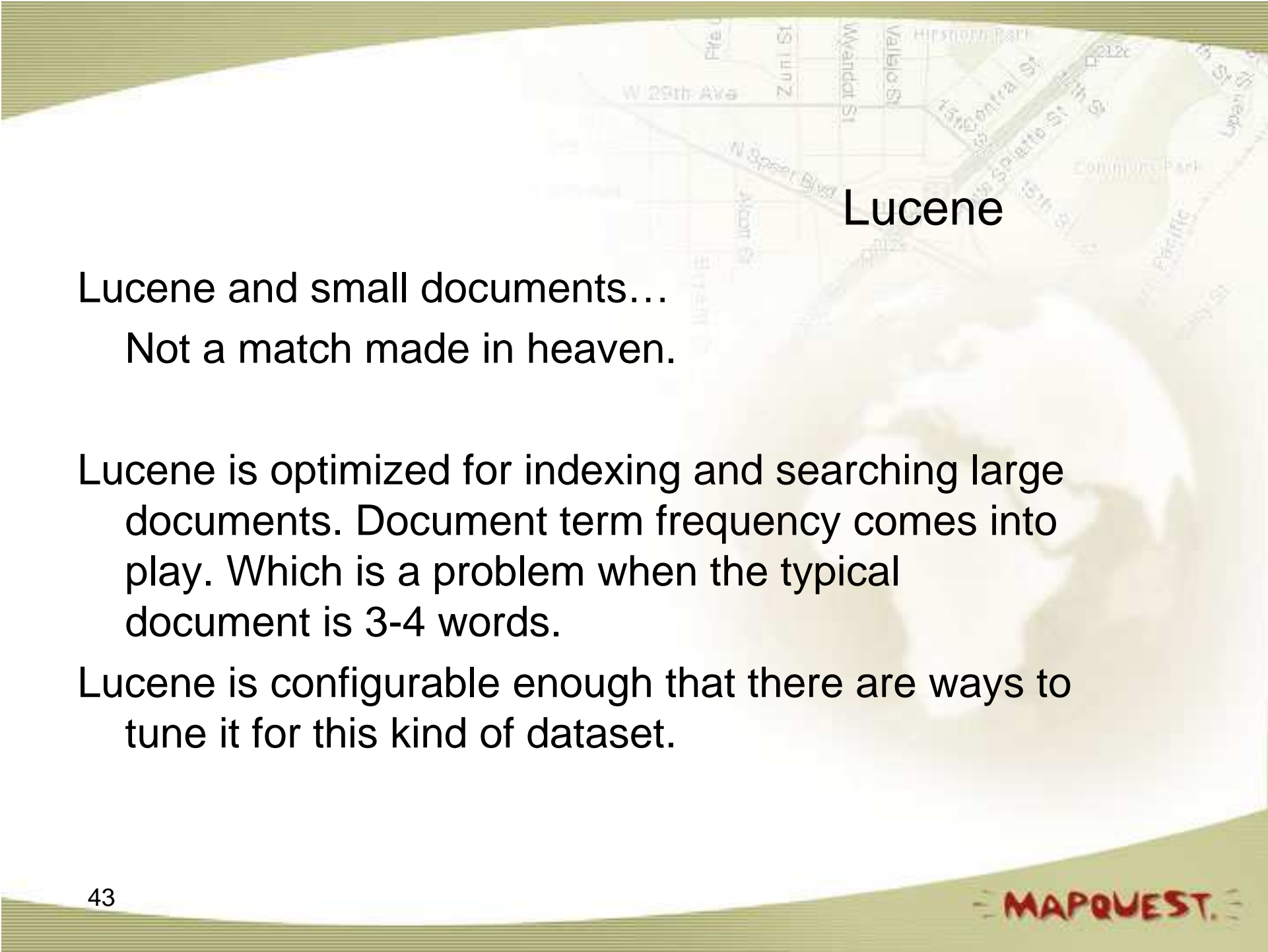
139

234

987

“Pizza Hut”





# Lucene

Lucene and small documents...

Not a match made in heaven.

Lucene is optimized for indexing and searching large documents. Document term frequency comes into play. Which is a problem when the typical document is 3-4 words.

Lucene is configurable enough that there are ways to tune it for this kind of dataset.

43

MAPQUEST

Not a match made in heaven.

Lucene is optimized for indexing and searching large documents. Document term frequency comes into play. Which is a problem when the typical document is 3-4 words.

Lucene is configurable enough that there are ways to tune it for this kind of dataset.

# Spatial Search



# Searching on the Spatial Grid

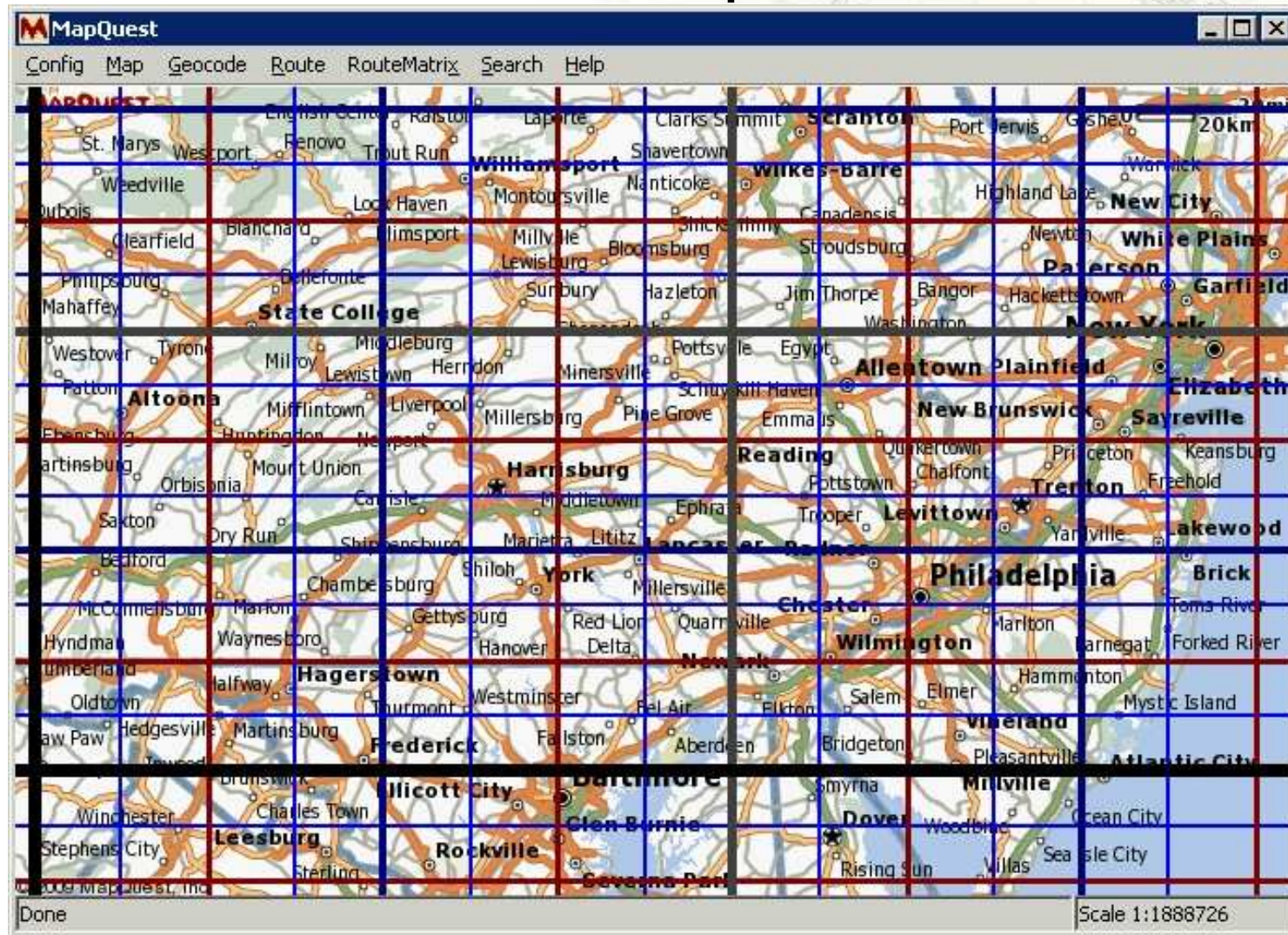
Based on “Geohash” concept

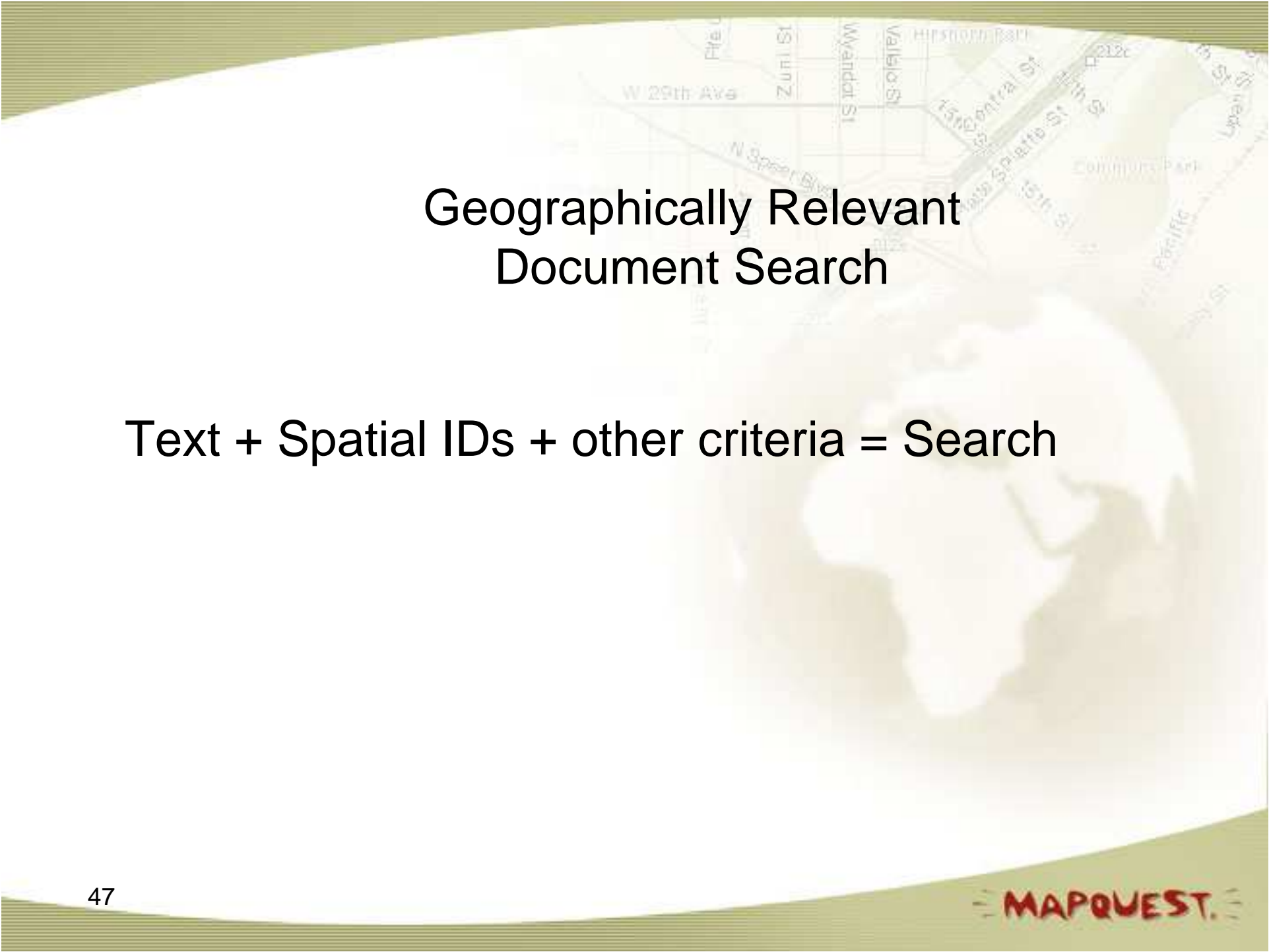
The world is divided into  
quadrants based on Lat/Lng  
Each quadrant is subdivided into  
quadrants, etc.  
Every quadrant on every level  
gets an ID. A spatial ID is  
defined by all the level IDs  
needed to identify a “box” at a  
particular granularity.

11			12		2
141	1421	1422	13		
	1424	1425			
144	143				
4					3



# Spatial Grid

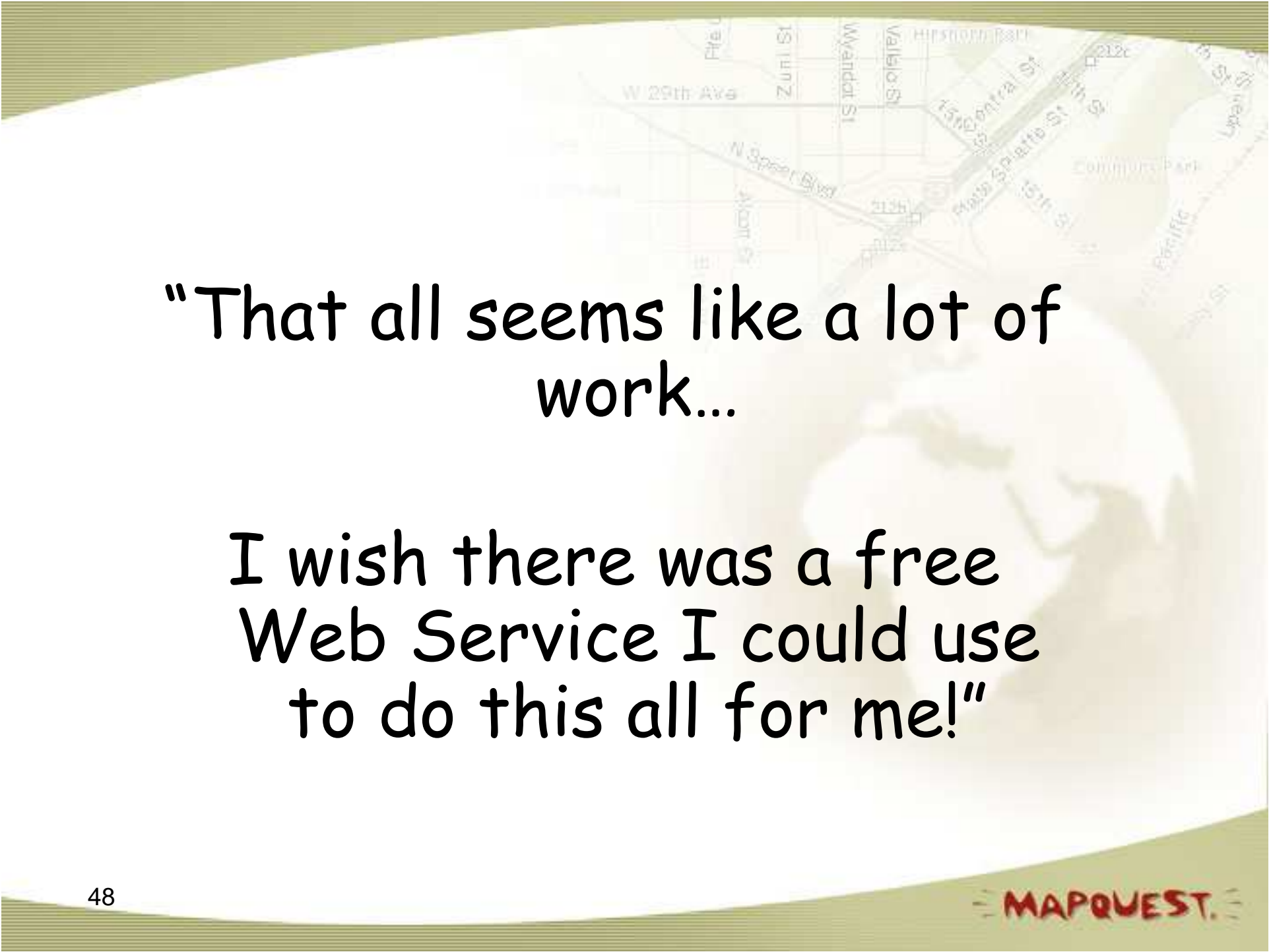




## Geographically Relevant Document Search

Text + Spatial IDs + other criteria = Search





"That all seems like a lot of  
work..."

I wish there was a free  
Web Service I could use  
to do this all for me!"



## Platform Web Services

- Free Web Services for Routing, Mapping, Geocoding
- Search and traffic on the way!
- [www.mapquestapi.com](http://www.mapquestapi.com)
- <http://platform.beta.mapquest.com/>
- Easily embed maps and routes in your web page using only HTML and JavaScript



Questions?