# MapReduce Design Patterns


## Barry Brumitt

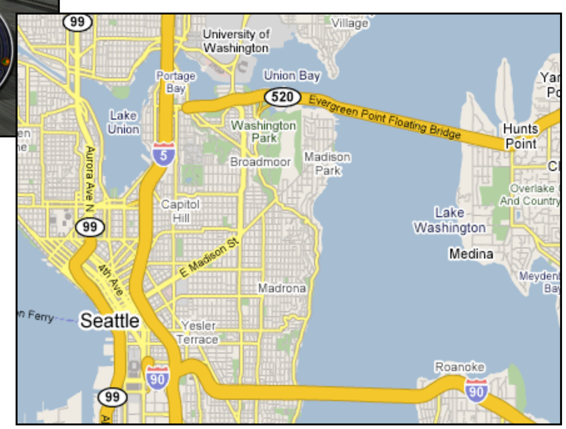
barryb@google.com
Software Engineer

# About your speaker…

**Ph.D. Robotics, Carnegie Mellon, '91 - '97**
Path Planning for Multiple Mobile Robots

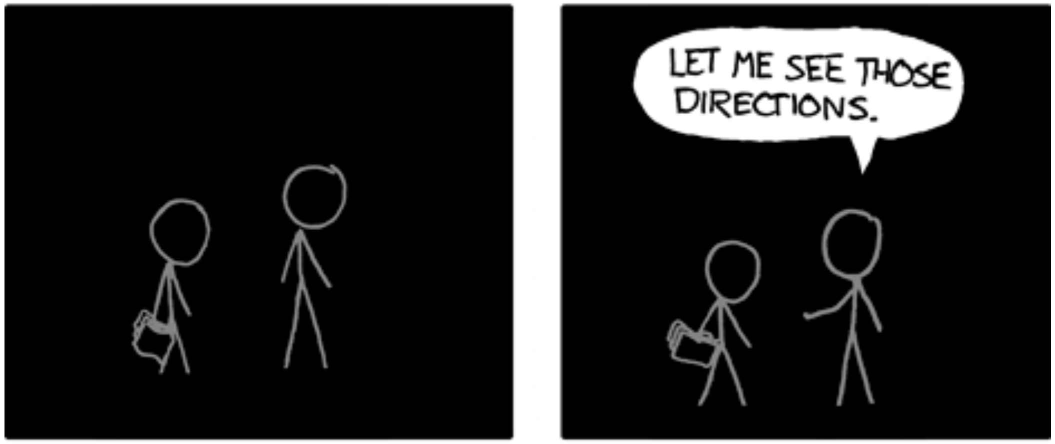
**Researcher, Microsoft Research, '98 - '02**
Ubiquitous Computing


**Software Eng., Microsoft Games, '03 - '05**
AI for Forza Motorsport



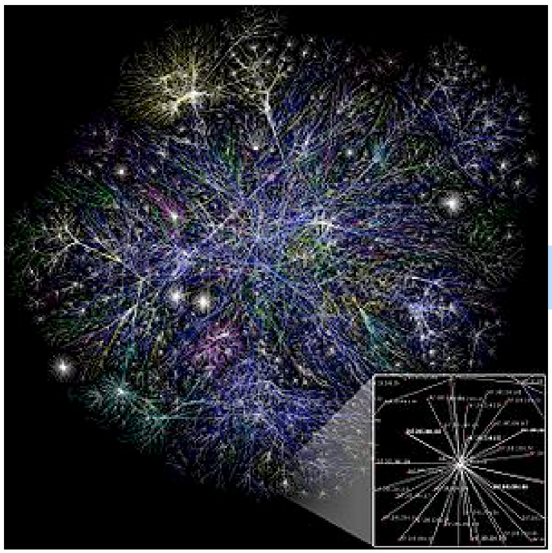

**Software Engineer, Google, '05 - now**
Maps: Pathfinder
Systems: Infrastructure


MY ROAD TRIP WITH MY BROTHER RAN INTO TROUBLE AROUND PAGE THREE OF THE GOOGLE MAPS PRINTOUT.
70. SLIGHT LEFT AT RT-22. GO 6.8 MI
71. TURN RIGHT TO STAY ON RT-22. GO 2.6 MI
72. TURN LEFT AT LAKE SHORE RD. GO 312 FT
73. TURN RIGHT AT DOCK ST. GO 427 FT
74. TAKE THE FERRY ACROSS THE LAKE. GO 2.8 MI
OKAY, NOW TAKE DOCK ST TOWARD THE FERRY
WE'RE SUPPOSED TO TAKE A FERRY? IT'S PAST MIDNIGHT, AND THESE WOODS ARE CREEPY.
GOOGLE MAPS WOULDN'T STEER US WRONG.
CLOSED



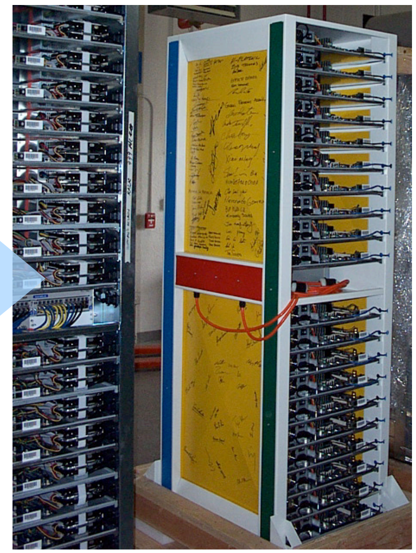

| | | |
|---|---|---|
| 74. | TAKE THE **FERRY** ACROSS THE **LAKE.** | GO 2.8 MI |
| 75. | CLIMB THE **HILL** TOWARD **HANGMAN'S RIDGE,** AVOIDING ANY **MOUNTAIN LIONS.** | UP 1,172 FT |
| 76. | WHEN YOU REACH AN **OLD BARN,** GO AROUND BACK, KNOCK ON THE **SECOND DOOR,** AND ASK FOR **CHARLIE.** | GO 52 FT |
| 77. | TELL **CHARLIE** THE **DANCING STONES** ARE **RESTLESS.** HE WILL GIVE YOU HIS **VAN.** | CAREFUL |
| 78. | TAKE **CHARLIE'S VAN** DOWN **OLD MINE ROAD.** DO NOT WAKE THE **STRAW MAN.** | GO π MI |
| 79. | TURN LEFT ON **COMSTOCK.** WHEN YOU FEEL THE **BLOOD** CHILL IN YOUR **VEINS,** STOP THE VAN AND **GET OUT.** | GO 3.2 MI |
| 80. | STAND VERY STILL. EXITS ARE **NORTH, SOUTH,** AND **EAST,** BUT ARE BLOCKED BY A **SPECTRAL WOLF.** | GO 0 FT |
| 81. | THE **SPECTRAL WOLF** FEARS ONLY **FIRE.** THE **GOOGLE MAPS TEAM** CAN NO LONGER HELP YOU, BUT IF YOU MASTER THE **WOLF,** HE WILL GUIDE YOU. **GODSPEED.** | GO ?? MI |

# Indexing Large Datasets



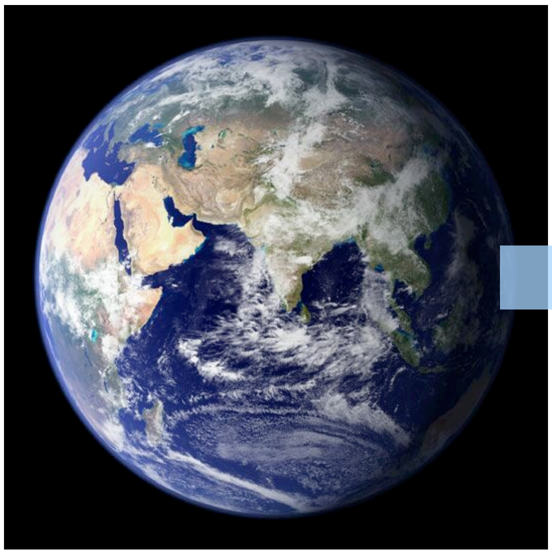
All web pages

Index Files



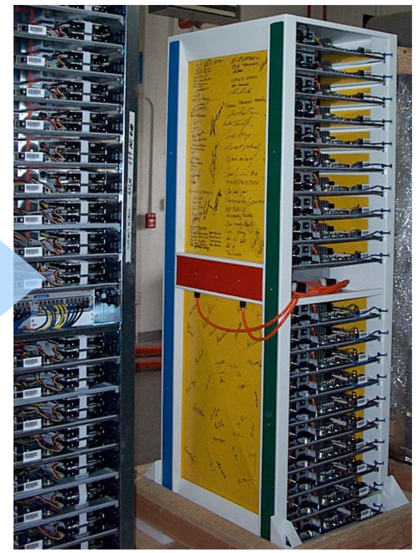
Data Center

# Indexing Large Datasets



Geographic Data

Index Files



Data Center

...not so useful for user-facing applications…

# Pointer Following (or) Joining

**Input**

**Feature List**

1: <type=Road>, <intersections=(3)>, <geom>, …

2: <type=Road>, <intersections=(3)>, <geom>, …

3: <type=Intersection>, stop_type, POI? …

4: <type=Road>, <intersections=(6)>, <geom>,

5: <type=Road>, <intersections=(3,6)>, <geom>, …

6: <type=Intersection>, stop_type, POI?, …

7: <type=Road>, <intersections=(6)>, <geom>, …

8: <type=Town>, <name>, <geom>, …

.
.
.

**Output**

**Intersection List**

3: <type=Intersection>, stop_type, <roads=(

1: <type=Road>, <geom>, <name>, …

2: <type=Road>, <geom>, <name>, …

5: <type=Road>, <geom>, <name>, … )>, …

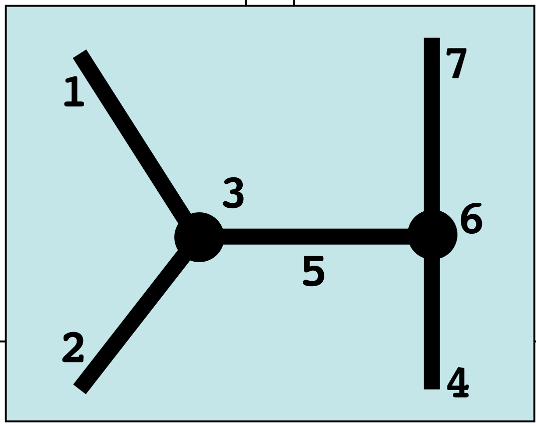
6: <type=Intersection>, stop_type, <roads=(

4: <type=Road>, <geom>, <name>, … ,

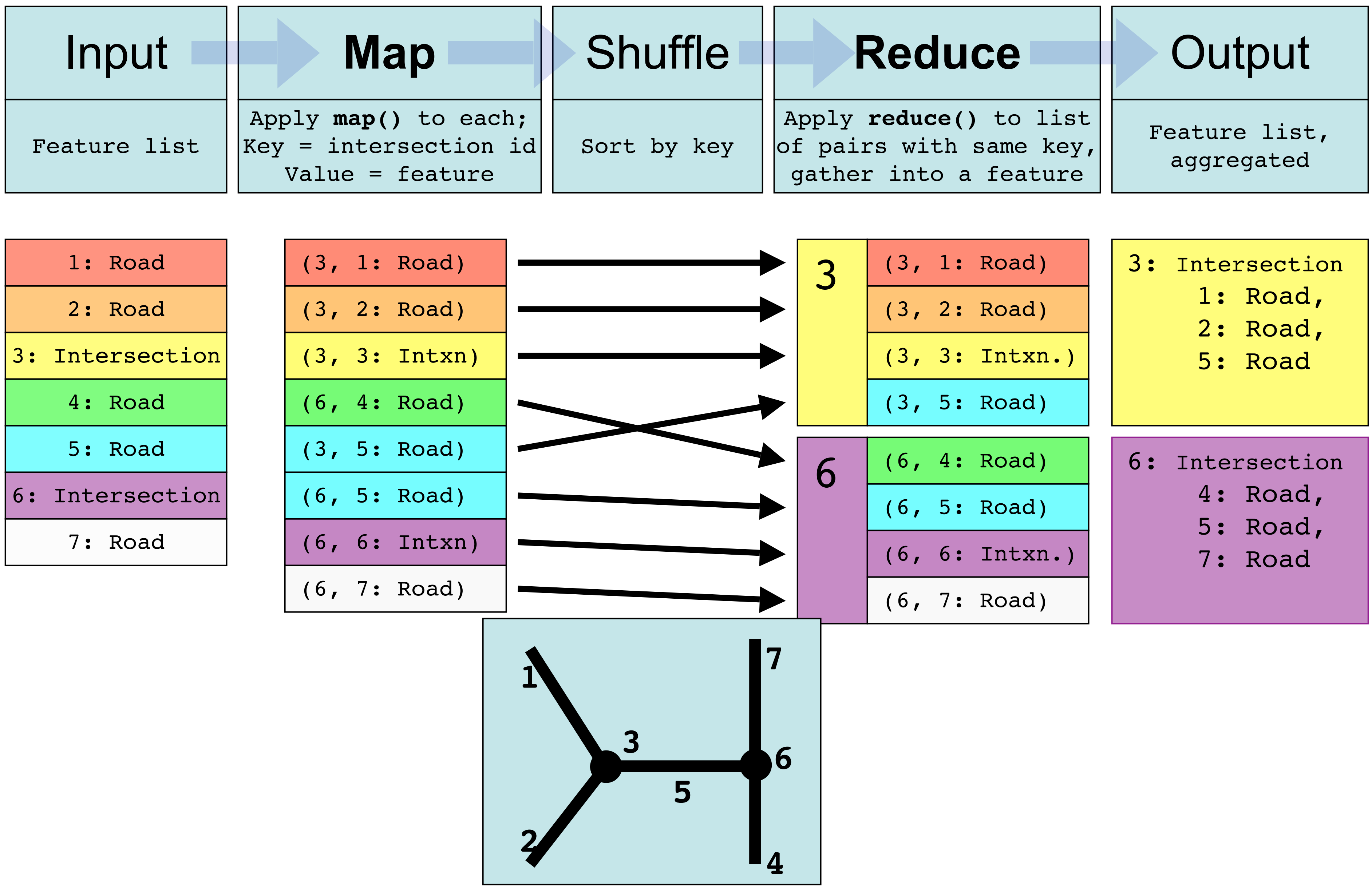
5: <type=Road>, <geom>, <name>, … ,

7: <type=Road>, <geom>, <name>, … )>, …

.
.
.



# Inner Join Pattern



# Inner Join Pattern in SQL

roads

| R | D | I |
|---|---|---|
| 1 | a | 3 |
| 2 | b | 3 |
| 4 | c | 6 |
| 5 | d | 6 |
| 7 | e | 6 |

ints

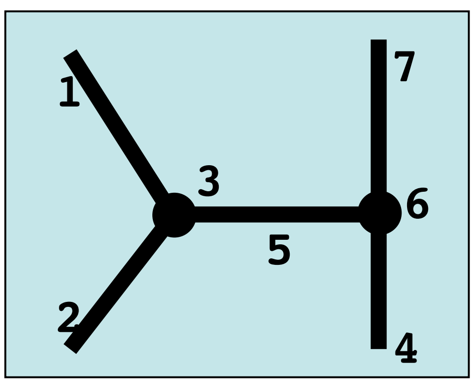
| I | D |
|---|---|
| 3 | x |
| 6 | y |

SELECT roads.R, roads.D, ints.D
FROM roads INNER JOIN ints
ON roads.I = ints.I

| r.R | r.D | r.I | i.I | i.D |
|---|---|---|---|---|
| 1 | a | 3 | 3 | x |
| 2 | b | 3 | 3 | x |
| 4 | c | 6 | 3 | x |
| 5 | d | 6 | 3 | x |
| 7 | e | 6 | 3 | x |
| 1 | a | 3 | 6 | y |
| 2 | b | 3 | 6 | y |
| 4 | c | 6 | 6 | y |
| 5 | d | 6 | 6 | y |
| 7 | e | 6 | 6 | y |

“Cross Join”

| r.R | r.D | i.D |
|---|---|---|
| 1 | a | x |
| 2 | b | x |
| 4 | c | y |
| 5 | d | y |
| 7 | e | x |



# Inner Join Pattern in SQL

roads

| R | D | I |
|---|---|---|
| 1 | a | 3 |
| 2 | b | 3 |
| 4 | c | 6 |
| 5 | d | 6 |
| 7 | e | 5 |

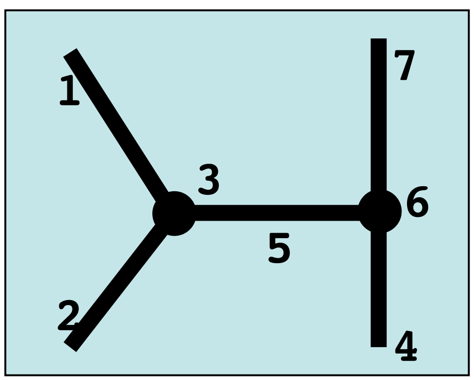
ints

| I | D |
|---|---|
| 3 | x |
| 6 | y |

```
SELECT roads.R, roads.D, ints.D
FROM roads INNER JOIN ints
ON roads.I = ints.I
```

```
SELECT roads.R, roads.D, ints.D
FROM roads, ints
WHERE roads.I = ints.I
```

| r.R | r.D | i.D |
|---|---|---|
| 1 | a | x |
| 2 | b | x |
| 4 | c | y |
| 5 | d | y |
| 7 | e | x |

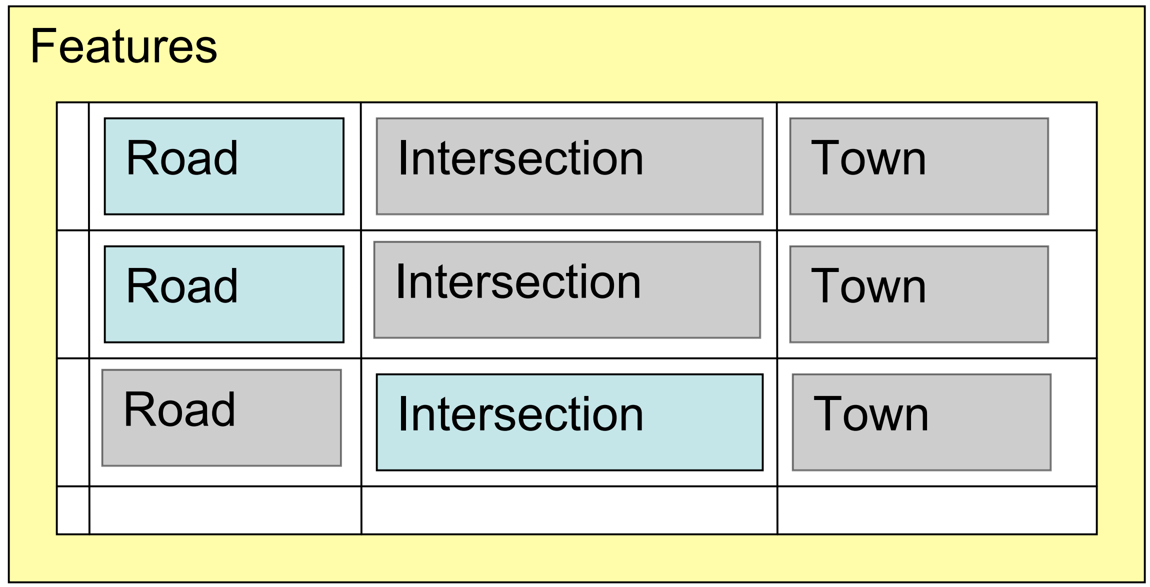



(aka “an Equi Join”)

# Tables vs. Flat File?

## Tables


## Flat File




```
Message GeoFeature {
  enum Type {
    ROAD = 1;
    INTERSECTION = 2;
    TOWN = 3;
  }
  required Type type = 0;
  optional Road road = 1;
  optional Intersection intersection = 2;
  optional Town town = 3 ;
}
```

“ Protocol Buffer ”

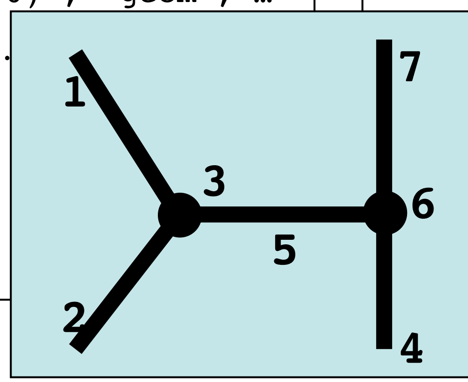
# References vs. Duplication?

**References**

```
1: <type=Road>, <intersections=(3)>, <geom>, …
2: <type=Road>, <intersections=(3)>, <geom>, …
3: <type=Intersection>, <roads=(1,2,5)>, …
4: <type=Road>, <intersections=(6)>, <geom>,
5: <type=Road>, <intersections=(3,6)>, <geom>, …
6: <type=Intersection>, <roads=(5,6,7)>, …
7: <type=Road>, <intersections=(6)>, <geom>, …
8: <type=Town>, <name>, <geom>, .
                    .
                    .
                    .
```

**Duplication**

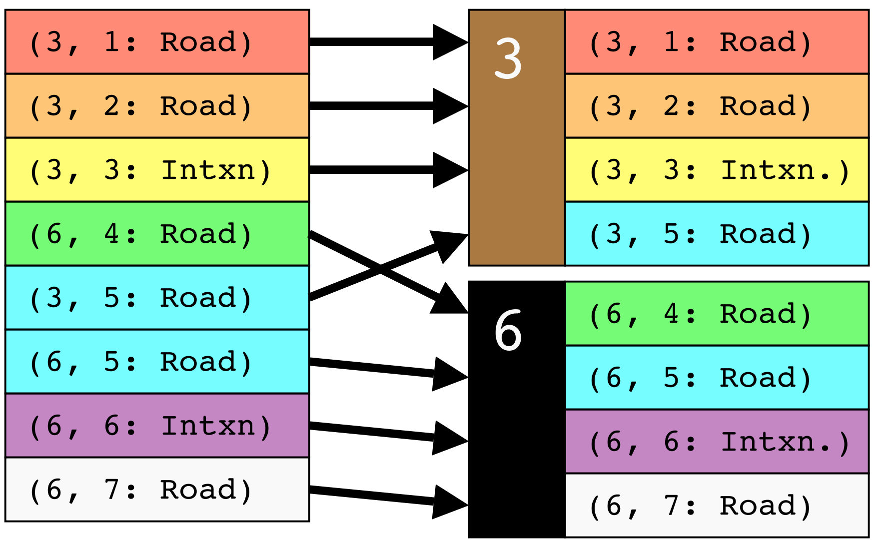
```
3: <type=Intersection>, <roads=(
     1: <type=Road>, <geom>, <name>, …
     2: <type=Road>, <geom>, <name>, …
     5: <type=Road>, <geom>, <name>, …)>, …
6: <type=Intersection>, <roads=(
     4: <type=Road>, <geom>, <name>, … >
     5: <type=Road>, <geom>, <name>, … >
     7: <type=Road>, <geom>, <name>, …)>, …
                    .
                    .
                    .
```




- References: Common primary key; easy restructuring
- Duplication: Avoids additional MR passes; denormalizes data
- …an engineering space / time / complexity tradeoff

# Code Example

```
class IntersectionAssemblerMapper : public
    Mapper {
  virtual void Map(MapInput* input) {
    GeoFeature feature;
    feature.FromMapInput(input);
    if (feature.type()==INTERSECTION) {
      Emit(feature.id(), input);
    } else if (feature.type() == ROAD) {
      Emit(feature.intersection_id(0), input);
      Emit(feature.intersection_id(1), input);
    }
  }
};
REGISTER_MAPPER(IntersectionAssemblerMapper);
```




```
class IntersectionAssemblerReducer : public
    Reducer {
  virtual void Reduce(ReduceInput* input) {
    GeoFeature feature;
    GraphIntersection intersection;
    intersection.id = input->key();
    while(!input->done()) {
      feature.FromMapInput(input->value());
      if (feature.type()==INTERSECTION)
        intersection.SetIntersection(feature);
      else
        intersection.AddRoadFeature(feature);
      input->next();
    }
    Emit(intersection);
  }
};
REGISTER_REDUCER(IntersectionAssemblerReducer);
```

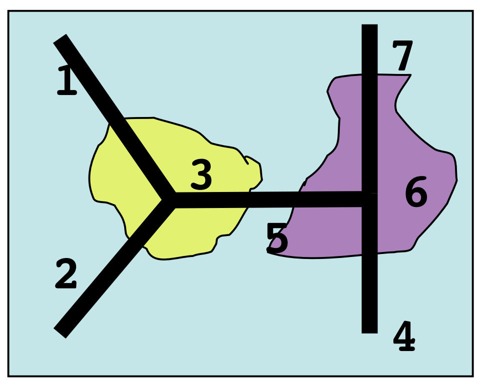
# Join, but no pointers or keys?

| Input | **Map** | Shuffle | **Reduce** | Output |
|---|---|---|---|---|
| List of items | Apply **map()** to each; emit (key,val) pairs | Sort by key | Apply **reduce()** to list of pairs with same key | New list of items |

| |
|---|
| 1: Road |
| 2: Road |
| 3: Town |
| 4: Road |
| 5: Road |
| 6: Town |
| 7: Road |

?

| |
|---|
| 3: 1,2,5 |
| 6: 4,5,7 |

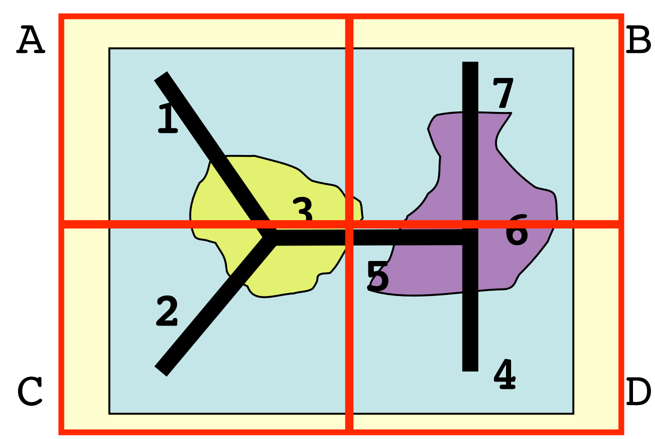


# Bucketing (or) Grace Hash Join

| Input | **Map** | Shuffle | **Reduce** | Output |
|---|---|---|---|---|
| Feature List | Emit (key, item) pair<br>Key = geometric hash<br>Secondary key = Type | Sort by keys | Intersect all towns with all roads; emit intersecting pairs | (town, road) pair |

| |
|---|
| 1: Road |
| 2: Road |
| 3: Town |
| 4: Road |
| 5: Road |
| 6: Town |
| 7: Road |

| |
|---|
| (A-Road, 1) |
| (C-Road, 1) |
| (C-Road, 2) |
| (A-Town, 3) |
| (B-Town, 3) |
| (C-Town, 3) |
| (D-Road, 4) |
| (C-Road, 5) |
| (D-Road, 5) |
| (B-Town, 6) |
| (D-Town, 6) |
| (B-Road, 7) |
| (D-Road, 7) |



# Reduce on Key A

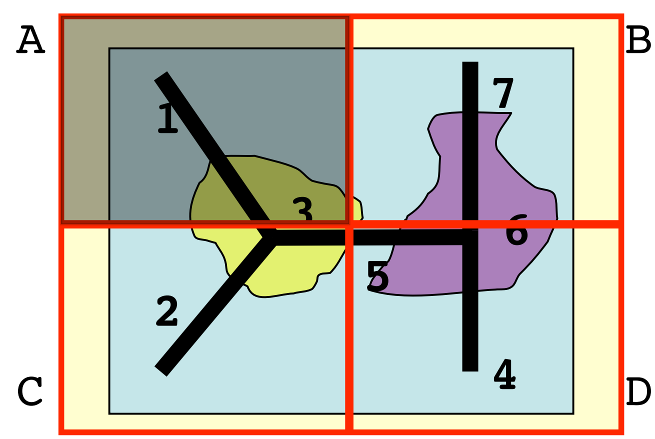
| Input | **Map** | Shuffle | **Reduce** | Output |
|---|---|---|---|---|
| Feature List | Emit (key, item) pair<br>Key = geometric hash<br>Secondary key = Type | Sort by keys | Intersect all towns with all roads; emit intersecting pairs | (town, road) pair |

| |
|---|
| 1: Road |
| 2: Road |
| 3: Town |
| 4: Road |
| 5: Road |
| 6: Town |
| 7: Road |

| |
|---|
| (A-Road, 1) |
| (C-Road, 1) |
| (C-Road, 2) |
| (A-Town, 3) |
| (B-Town, 3) |
| (C-Town, 3) |
| (D-Road, 4) |
| (C-Road, 5) |
| (D-Road, 5) |
| (B-Town, 6) |
| (D-Town, 6) |
| (B-Road, 7) |
| (D-Road, 7) |

| |
|---|
| (A-Town, 3) |
| (A-Road, 1) |

| |
|---|
| (3, 1) |



# Reduce on Key B

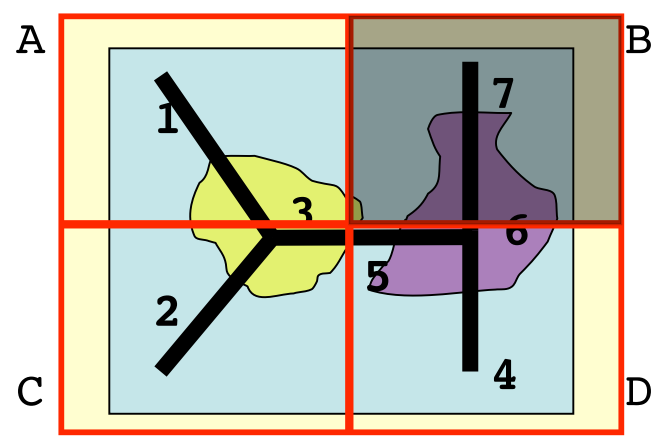
| Input | Map | Shuffle | Reduce | Output |
|---|---|---|---|---|
| Feature List | Emit (key, item) pair<br>Key = geometric hash<br>Secondary key = Type | Sort by keys | Intersect all towns with all roads; emit intersecting pairs | (town, road) pair |

| |
|---|
| 1: Road |
| 2: Road |
| 3: Town |
| 4: Road |
| 5: Road |
| 6: Town |
| 7: Road |

| |
|---|
| (A-Road, 1) |
| (C-Road, 1) |
| (C-Road, 2) |
| (A-Town, 3) |
| (B-Town, 3) |
| (C-Town, 3) |
| (D-Road, 4) |
| (C-Road, 5) |
| (D-Road, 5) |
| (B-Town, 6) |
| (D-Town, 6) |
| (B-Road, 7) |
| (D-Road, 7) |

| |
|---|
| (B-Town, 3) |
| (B-Town, 6) |
| (B-Road, 7) |

| |
|---|
| (6, 7) |



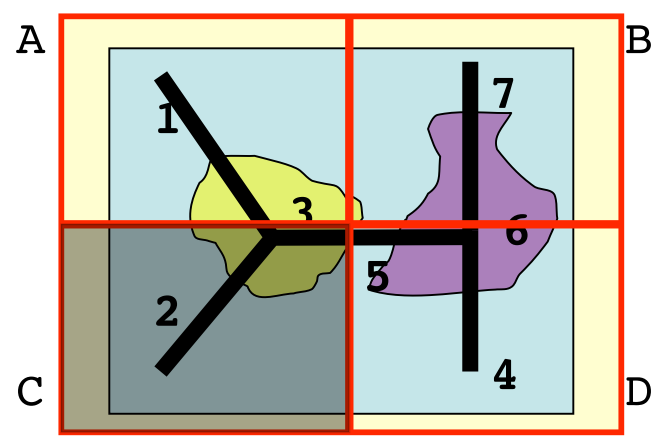
# Reduce on Key C

| Input | Map | Shuffle | Reduce | Output |
|---|---|---|---|---|
| Feature List | Emit (key, item) pair<br>Key = geometric hash<br>Secondary key = Type | Sort by keys | Intersect all towns with all roads; emit intersecting pairs | (town, road) pair |

| | |
|---|---|
| 1: Road | (A-Road, 1) |
| 2: Road | (C-Road, 1) |
| 3: Town | (C-Road, 2) |
| 4: Road | (A-Town, 3) |
| 5: Road | (B-Town, 3) |
| 6: Town | (C-Town, 3) |
| 7: Road | (D-Road, 4) |
| | (C-Road, 5) |
| | (D-Road, 5) |
| | (B-Town, 6) |
| | (D-Town, 6) |
| | (B-Road, 7) |
| | (D-Road, 7) |

| |
|---|
| (C-Town, 3) |
| (C-Road, 1) |
| (C-Road, 5) |
| (C-Road, 2) |

| |
|---|
| (3, 1) |
| (3, 2) |
| (3, 5) |



# Reduce on Key D

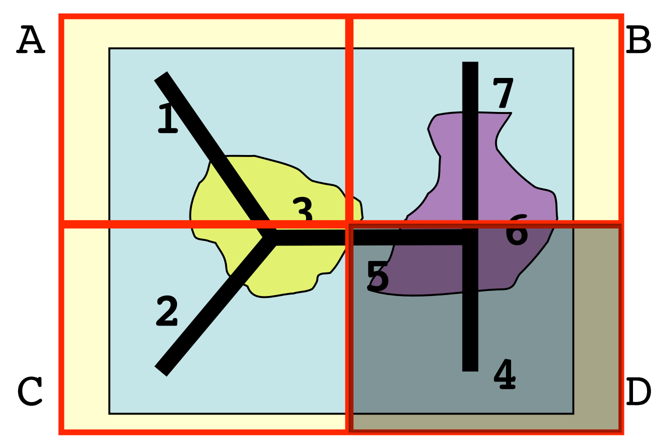
| Input | **Map** | Shuffle | **Reduce** | Output |
|---|---|---|---|---|
| Feature List | Emit (key, item) pair<br>Key = geometric hash<br>Secondary key = Type | Sort by keys | Intersect all towns with all roads; emit intersecting pairs | (town, road) pair |

| |
|---|
| 1: Road |
| 2: Road |
| 3: Town |
| 4: Road |
| 5: Road |
| 6: Town |
| 7: Road |

| |
|---|
| (A-Road, 1) |
| (C-Road, 1) |
| (C-Road, 2) |
| (A-Town, 3) |
| (B-Town, 3) |
| (C-Town, 3) |
| (D-Road, 4) |
| (C-Road, 5) |
| (D-Road, 5) |
| (B-Town, 6) |
| (D-Town, 6) |
| (B-Road, 7) |
| (D-Road, 7) |

| |
|---|
| (D-Town, 6) |
| (D-Road, 4) |
| (D-Road, 5) |
| (D-Road, 7) |

| |
|---|
| (6, 4) |
| (6, 5) |
| (6, 7) |



# Output… not quite...

| Input | Map | Shuffle | Reduce | Output |
|---|---|---|---|---|
| Feature List | Apply **map()** to each; emit (key,val) pairs | Sort by key | Apply **reduce()** to list of pairs with same key | List of items |

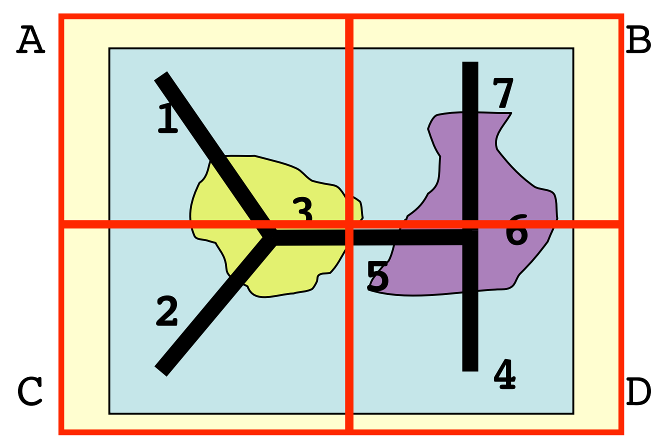
| | |
|---|---|
| 1: | Road |
| 2: | Road |
| 3: | Town |
| 4: | Road |
| 5: | Road |
| 6: | Town |
| 7: | Road |

(3, 1)

(6, 7)

(3, 1)
(3, 2)
(3, 5)

(6, 7)
(6, 4)
(6, 5)

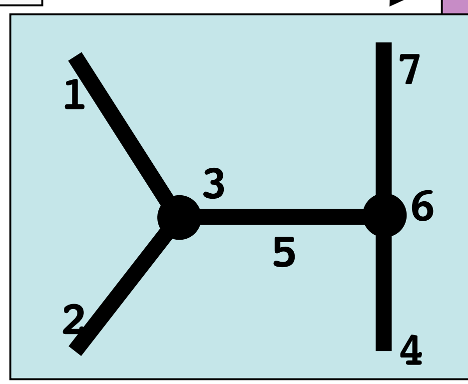


# …recall earlierJoin Pattern

| Input | Map | Shuffle | Reduce | Output |
|---|---|---|---|---|
| List of items | Apply **map()** to each; emit (key,val) pairs | Sort by key | Apply **reduce()** to list of pairs with same key | New list of items |

| Input | Map |
|---|---|
| 1: Road | (3, 1: Road) |
| 2: Road | (3, 2: Road) |
| 3: Intersection | (3, 3: Intxn) |
| 4: Road | (6, 4: Road) |
| 5: Road | (3, 5: Road) |
| 6: Intersection | (6, 5: Road) |
| 7: Road | (6, 6: Intxn) |
| | (6, 7: Road) |

| Key | Reduce input | Output |
|---|---|---|
| 3 | (3, 1: Road)<br>(3, 2: Road)<br>(3, 3: Intxn.)<br>(3, 5: Road) | 3: Intersection<br>1: Road,<br>2: Road,<br>5: Road |
| 6 | (6, 4: Road)<br>(6, 5: Road)<br>(6, 6: Intxn.)<br>(6, 7: Road) | 6: Intersection<br>4: Road,<br>5: Road,<br>7: Road |



# Recursive Key Join Pattern

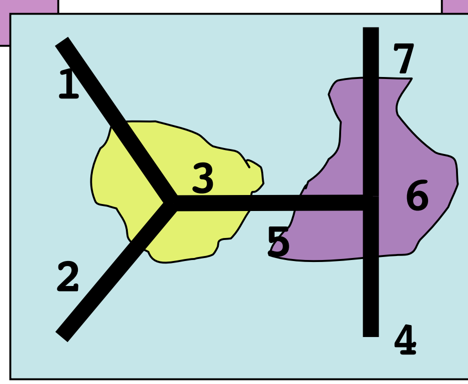
| Input | Map | Shuffle | Reduce | Output |
|---|---|---|---|---|
| Output from previous phase | Identity Mapper, key = town | Sort by key | Reducer sorts, gathers, remove duplicates; similar to join | Index of roads in each town |

| Input |
|---|
| (3, 1) |
| (6, 7) |
| (3, 1) |
| (3, 2) |
| (3, 5) |
| (6, 7) |
| (6, 4) |
| (6, 5) |

| Map |
|---|
| (3, 1) |
| (6, 7) |
| (3, 1) |
| (3, 2) |
| (3, 5) |
| (6, 7) |
| (6, 4) |
| (6, 5) |

| Key | Reduce input |
|---|---|
| 3 | (3, 1) |
| | (3, 5) |
| | (3, 2) |
| | (3, 1) |
| 6 | (6, 7) |
| | (6, 5) |
| | (6, 7) |
| | (6, 4) |

| Output |
|---|
| (3: 1, 2, 5) |
| (6: 4, 5, 7) |



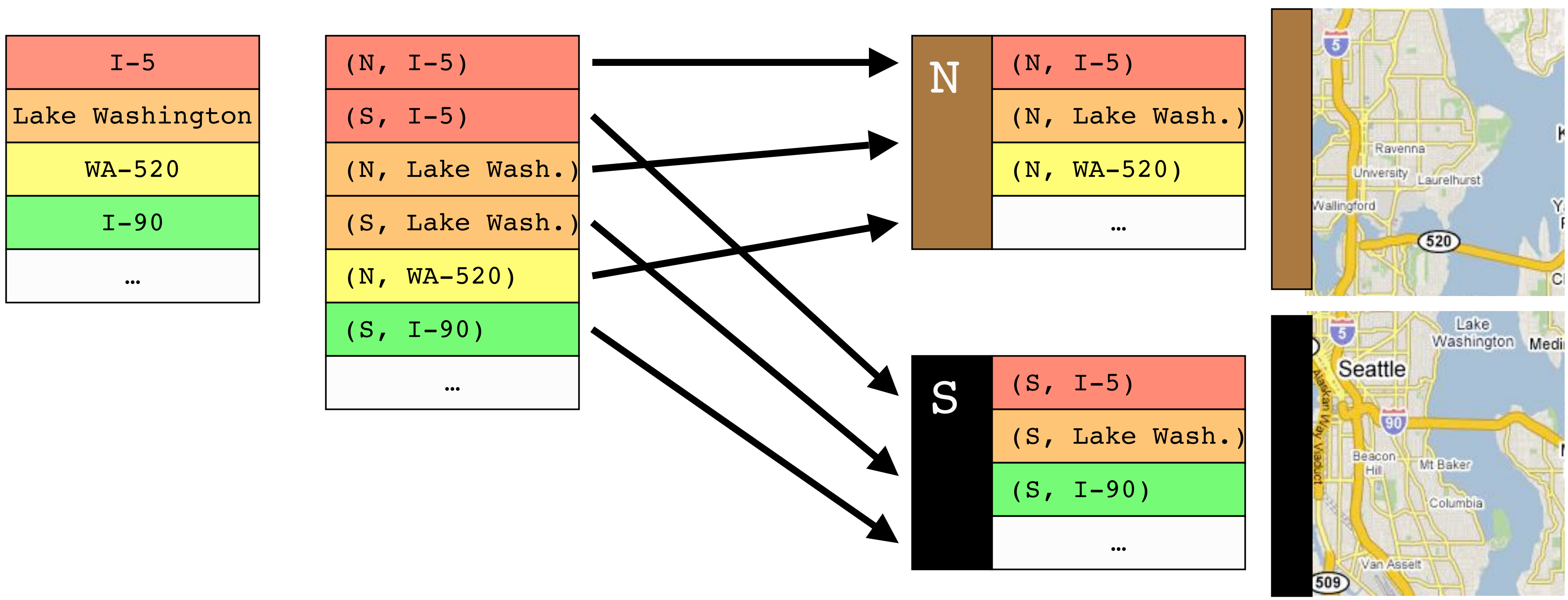

Could use 2ndry keys to avoid reduce sort(), eg: (6-7, 7)

# Chained MapReduce's Pattern

| Input | **Map** | Shuffle | **Reduce** | Output |
|---|---|---|---|---|
| Feature List | Emit (key, item) pair<br>Key = geometric hash<br>Secondary key = Type | Sort by keys | Intersect all towns with all roads; emit intersecting pairs | (town, road) pair |
| (town, road) pair | Identity Mapper, key = town | Sort by key | Reducer sorts, gathers, remove duplicates; similar to join | Index of roads in each town |

# Distributing Costly Computation: e.g. Rendering Map Tiles

| Input | Map | Shuffle | Reduce | Output |
|---|---|---|---|---|
| Geographic feature list | Emit each to all overlapping latitude-longitude rectangles | Sort by key (key= Rect. Id) | Render tile using data for all enclosed features | Rendered tiles |




(Bucket pattern)

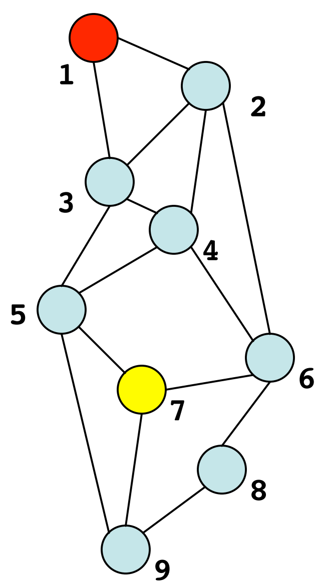
(Parallel rendering)

# Finding Nearest Points Of Interest (POIs)

**Feature List**

```
1, Type, Road, Intersection, …
2, Type, Road, Intersection, …
3, Type, Road, Intersection, …
 . . .
```

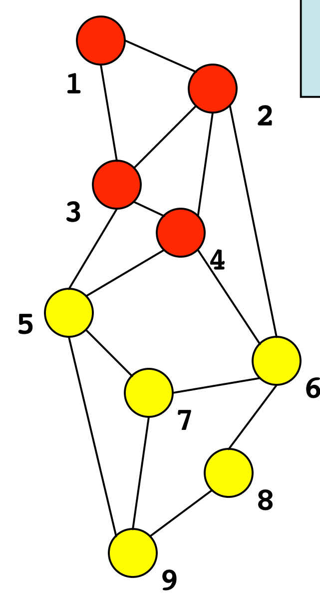
Input




**Nearest POI within 5mi of Intersection**

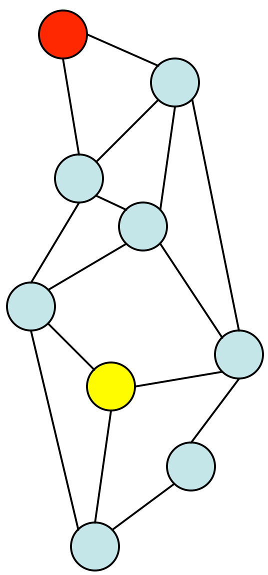
```
(1, 1)
(2, 1)
(3, 1)
(4, 1)
(5, 7)
(6, 7)
(7, 7)
(8, 7)
(9, 7)
```

Output



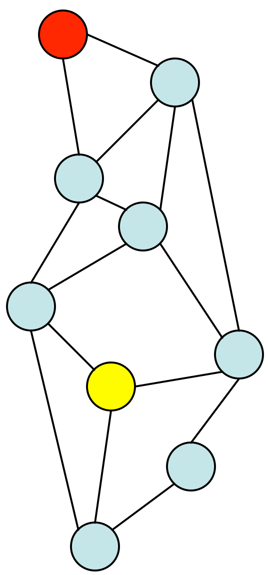
# Finding Nearest POI on a Graph

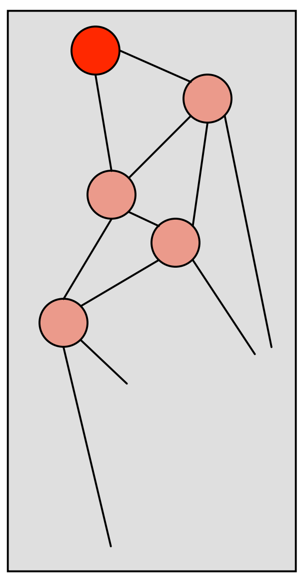
| Input | Map | Shuffle | Reduce | Output |
|---|---|---|---|---|
| Graph | Perform Dijkstra from each POI node. Emit POI & dist. At each node in search. | | For each node, emit closest POI | Nodes with nearest POIs |



# Finding Nearest POI on a Graph

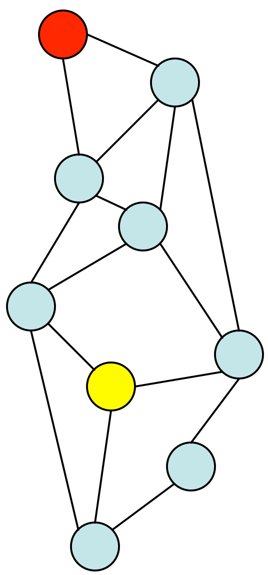
| Input | Map | Shuffle | Reduce | Output |
|---|---|---|---|---|
| Graph | Perform Dijkstra from each POI node. Emit POI & dist. At each node in search. | | For each node, emit closest POI | Nodes with nearest POIs |

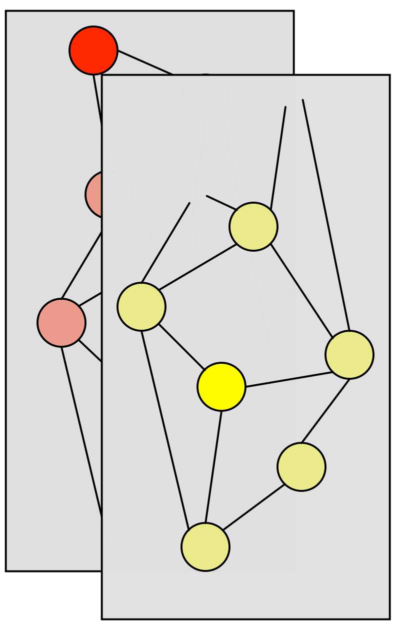


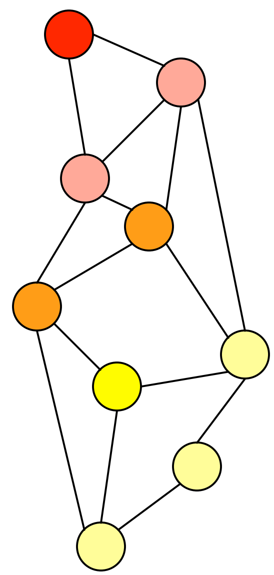


# Finding Nearest POI on a Graph

| Input | **Map** | Shuffle | **Reduce** | Output |
|---|---|---|---|---|
| Graph | Perform Dijkstra from each POI node. Emit POI & dist. At each node in search. | | For each node, emit closest POI | Nodes with nearest POIs |

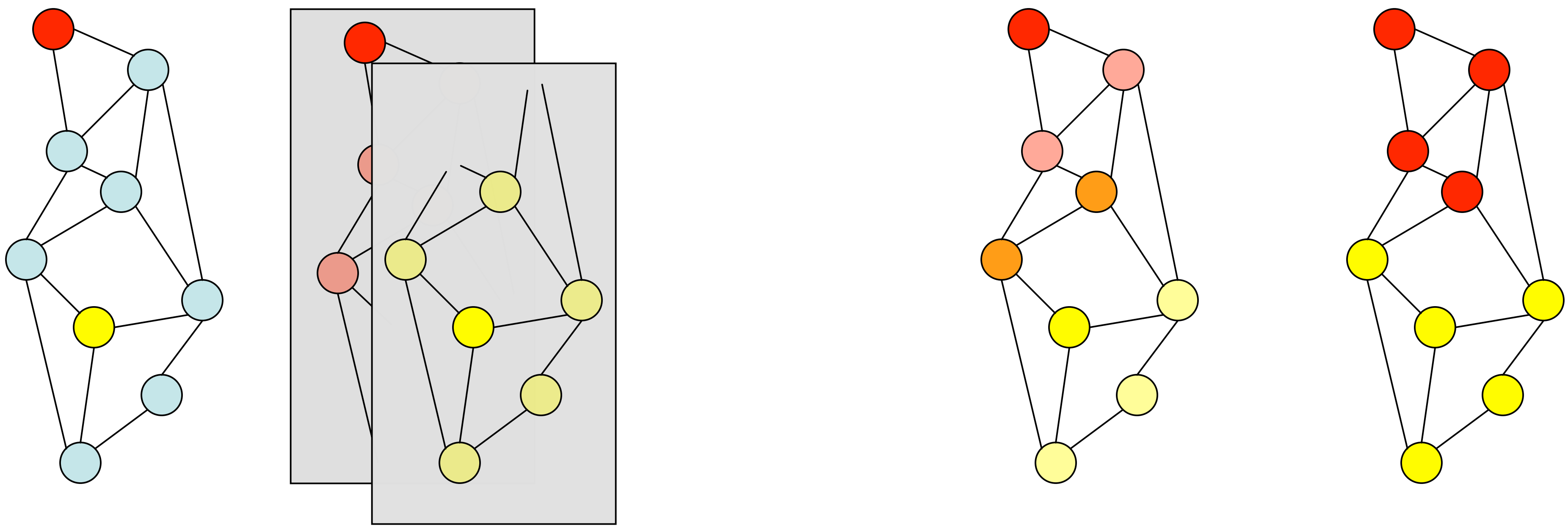






# Finding Nearest POI on a Graph

| Input | **Map** | Shuffle | **Reduce** | Output |
|---|---|---|---|---|
| Graph | Perform Dijkstra from each POI node. Emit POI & dist. At each node in search. | | For each node, emit closest POI | Nodes with nearest POIs |



# Putting it all together: Nearest POI

| Input | **Map** | Shuffle | **Reduce** | Output |
|---|---|---|---|---|
| Feature List | Use key-join pattern to create nodes,edges out of intersections,roads | | | Nodes with edges |
| Nodes with edges | Use bucketing pattern to create “appropriate”(overlapping, large-enough) subgraphs | | | **Subgraphs** |
| **Subgraphs** | Perform Dijkstra from each POI node. Emit POI & dist. At each node in search. | → | For each node, emit closest POI | Nodes with nearest POI & dist |
| Nodes with nearest POI & dist | Use identity mapper & gather pattern to sort and clean-up node, POI pairs | | | Sorted nodes with nearest POI |

# Hard Problems for MapReduce

- Following multiple pointer hops
- Iterative algorithms
- Algorithms with global state
- Operations on graphs without good embeddings

- [insert your favorite challenge here]

# Summary

MapReduce eases:

- Machine coordination
- Network communication
- Fault tolerance
- Scaling
- Productivity

MapReduce patterns:

- “Flat” data structures
- Foreign / Recursive Key Joins (aka pointer following)
- Hash Joins (aka bucketing)
- Distribute $$ computation
- Chain MapReduce phases
- Simplify Reduce() by using secondary keys
- [ insert your pattern here ]

# Questions?

- *MapReduce: Simplified Data Processing on Large Clusters, Jeffrey Dean and Sanjay Ghemawat OSDI'04: Sixth Symposium on Operating System Design and Implementation*

- Contact: barryb@google.com