

MapReduce Design Patterns

Barry Brumitt

barryb@google.com
Software Engineer



About your speaker...



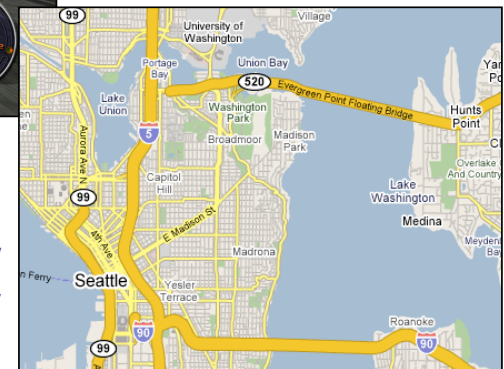
Ph.D. Robotics, Carnegie Mellon, '91 - '97
Path Planning for Multiple Mobile Robots



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



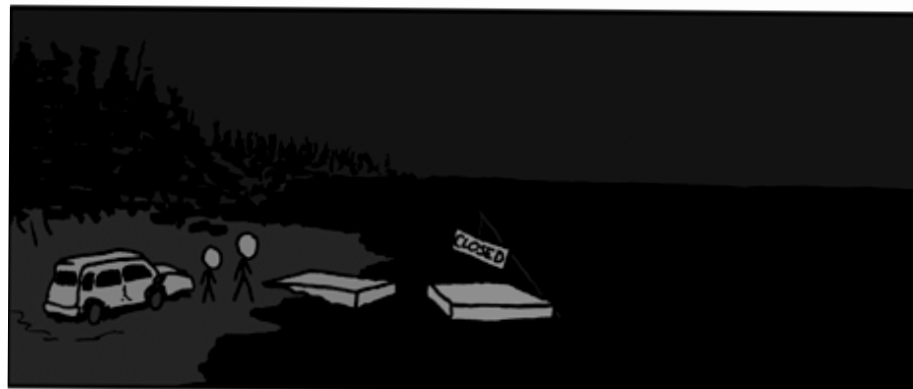
Software Eng., Microsoft Games, '03 - '05
AI for Forza Motorsport

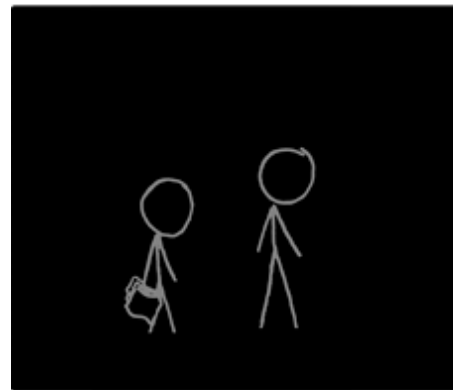


Software Engineer, Google, '05 - now
Maps: Pathfinder
Systems: Infrastructure

MY ROAD TRIP WITH MY BROTHER RAN INTO TROUBLE
AROUND PAGE THREE OF THE GOOGLE MAPS PRINTOUT.

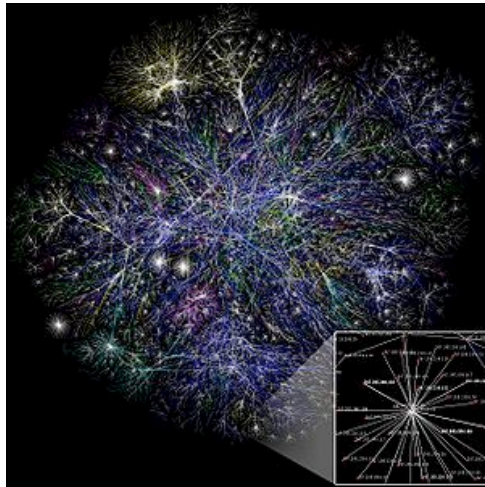
← 70. SIGHT LEFT AT RT-22.	GO 6.8 MI
→ 71. TURN RIGHT TO STAY ON RT-22.	GO 2.6 MI
← 72. TURN LEFT AT LAKE SHORE RD.	GO 312 FT
→ 73. TURN RIGHT AT DOCK ST.	GO 427 FT
~~~~~ 74. TAKE THE FERRY ACROSS THE LAKE.	GO 2.8 MI





74.	TAKE THE <b>FERRY</b> ACROSS THE <b>LAKE</b> .	GO 2.8 MI
75.	CLIMB THE <b>HILL</b> TOWARD <b>HANGMAN'S RIDGE</b> , AVOIDING ANY <b>MOUNTAIN LIONS</b> .	UP 1,172 FT
76.	WHEN YOU REACH AN <b>OLD BARN</b> , GO AROUND BACK, KNOCK ON THE <b>SECOND DOOR</b> , AND ASK FOR <b>CHARLIE</b> .	GO 52 FT
77.	TELL <b>CHARLIE</b> THE <b>DANCING STONES</b> ARE <b>RESTLESS</b> . HE WILL GIVE YOU HIS <b>VAN</b> .	CAREFUL
78.	TAKE <b>CHARLIE'S VAN</b> DOWN <b>OLD MINE ROAD</b> . DO NOT WAKE THE <b>STRAW MAN</b> .	GO 11 MI
79.	TURN LEFT ON <b>COMSTOCK</b> . WHEN YOU FEEL THE <b>BLOOD</b> CHILL IN YOUR <b>VEINS</b> , STOP THE VAN AND <b>GET OUT</b> .	GO 3.2 MI
80.	STAND VERY STILL. EXITS ARE <b>NORTH, SOUTH, AND EAST</b> , BUT ARE BLOCKED BY A <b>SPECTRAL WOLF</b> .	GO 0 FT
81.	THE <b>SPECTRAL WOLF</b> FEARS ONLY <b>FIRE</b> . THE <b>GOOGLE MAPS TEAM</b> CAN NO LONGER HELP YOU, BUT IF YOU MASTER THE <b>WOLF</b> , HE WILL GUIDE YOU. <b>GODSPEED</b> .	GO ?? MI

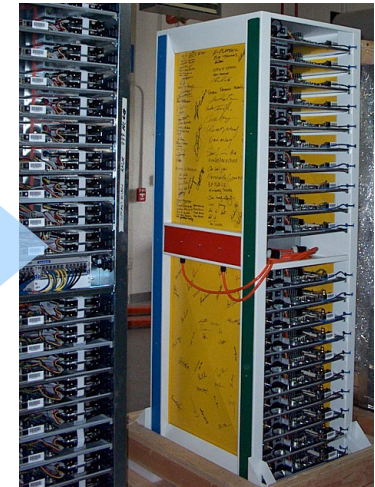
# Indexing Large Datasets



All web pages

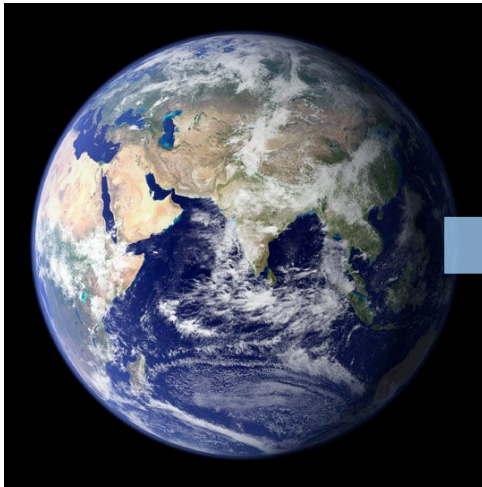


Index Files



Data Center

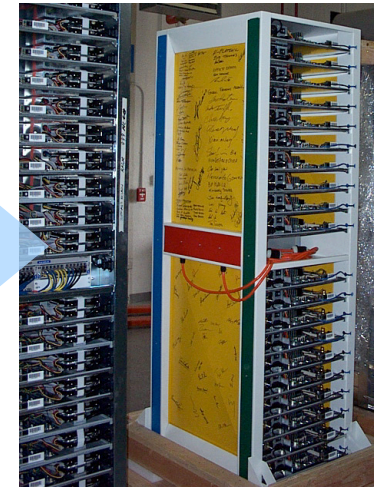
# Indexing Large Datasets



Geographic Data



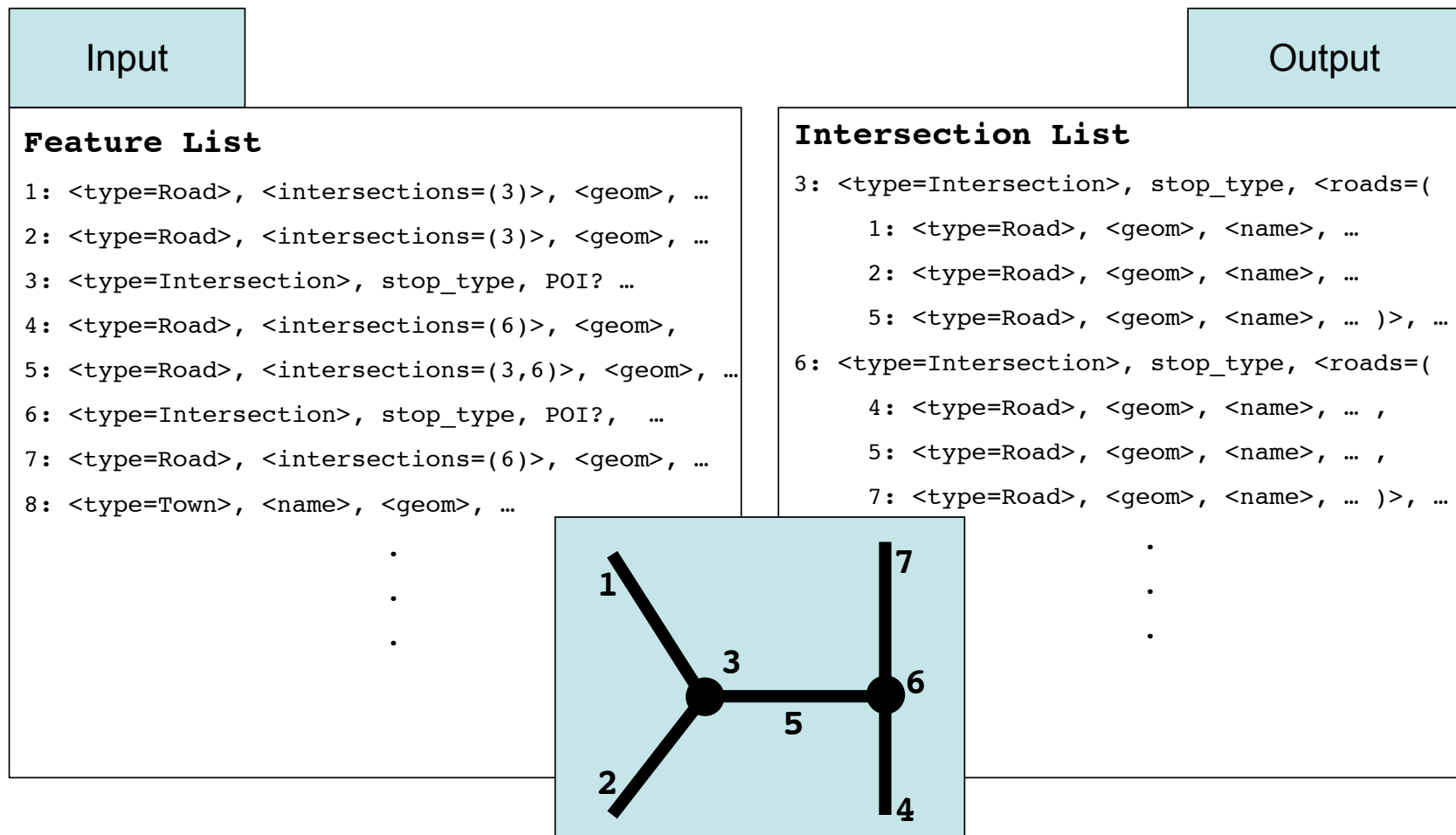
Index Files



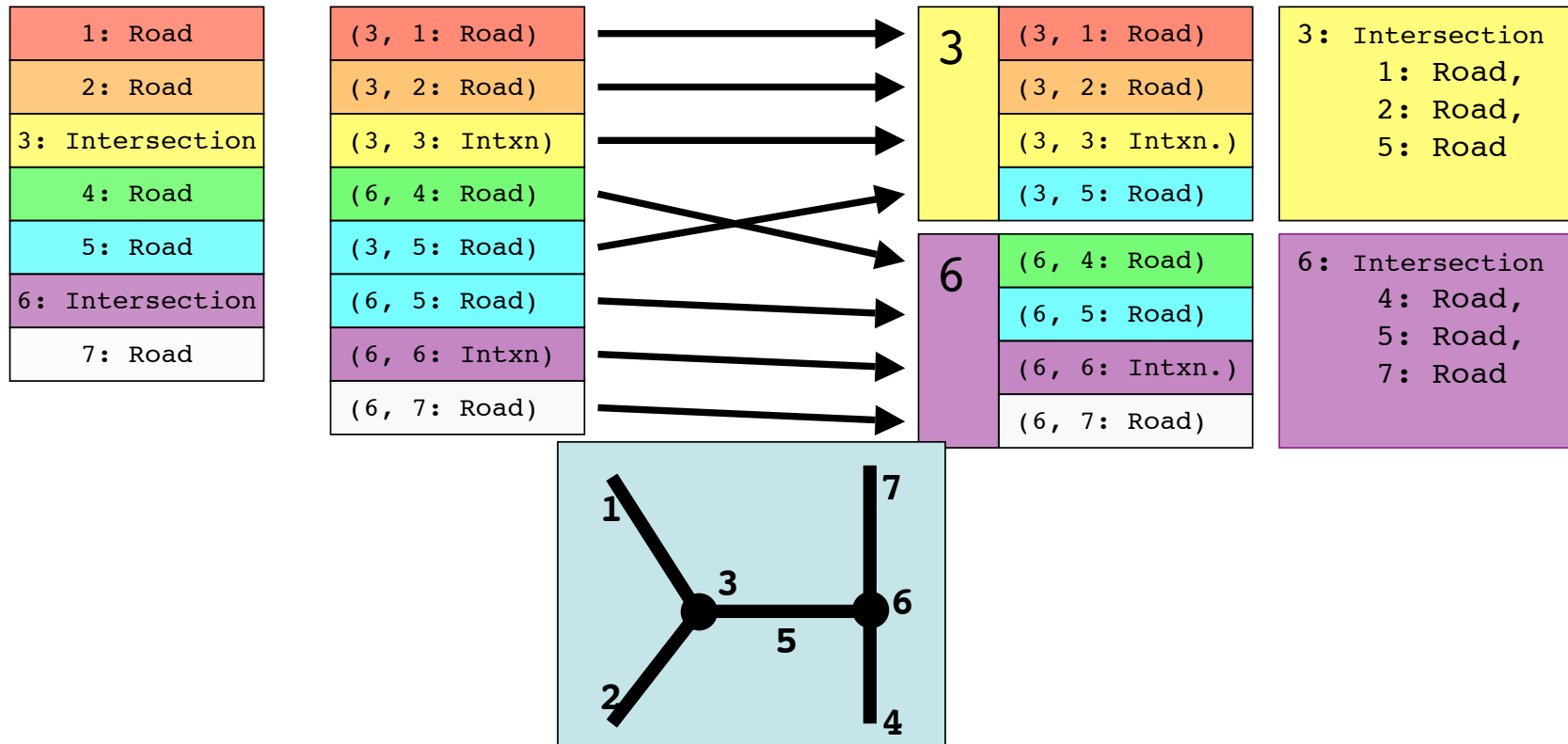
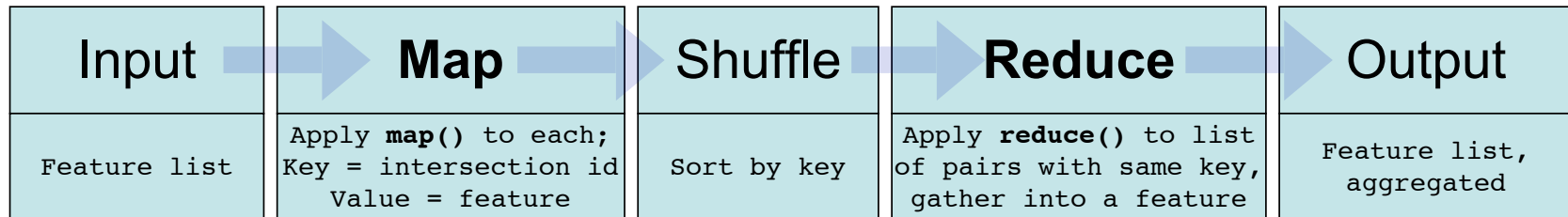
Data Center

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

# Pointer Following (or) Joining



# Inner Join Pattern



# Inner Join Pattern in SQL

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

roads

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

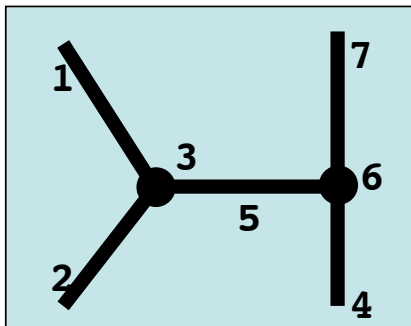
ints

I	D
3	x
6	y



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

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



“Cross Join”

# Inner Join Pattern in SQL

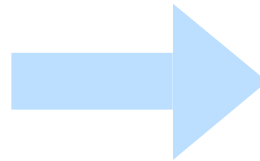
roads

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

ints

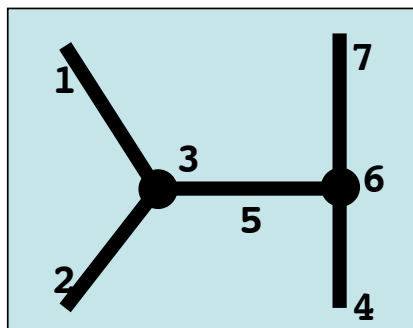
I	D
3	x
6	y

```
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

Roads			

Intersections			

Towns			

## Flat File

Features			
	Road	Intersection	Town
	Road	Intersection	Town
	Road	Intersection	Town

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

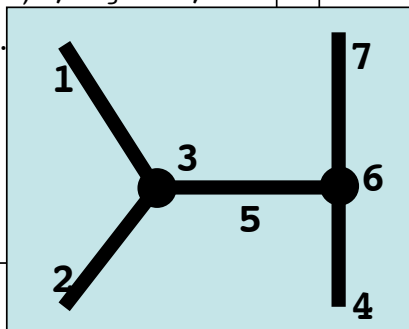
# References vs. Duplication?

## 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

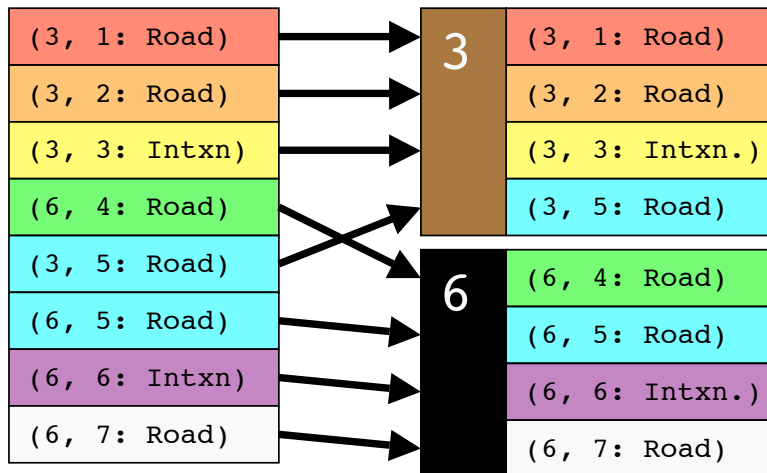
```
3: <type=Intersection>, <roads=(  
    1: <type=Road>, <geom>, <name>, ...  
    2: <type=Road>, <geom>, <name>, ...  
    5: <type=Road>, <geom>, <name>, ...)>, ...  
6: <type=Intersection>, <roads=(  
    4: <type=Road>, <geom>, <name>, ... >  
    5: <type=Road>, <geom>, <name>, ... >  
    7: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



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

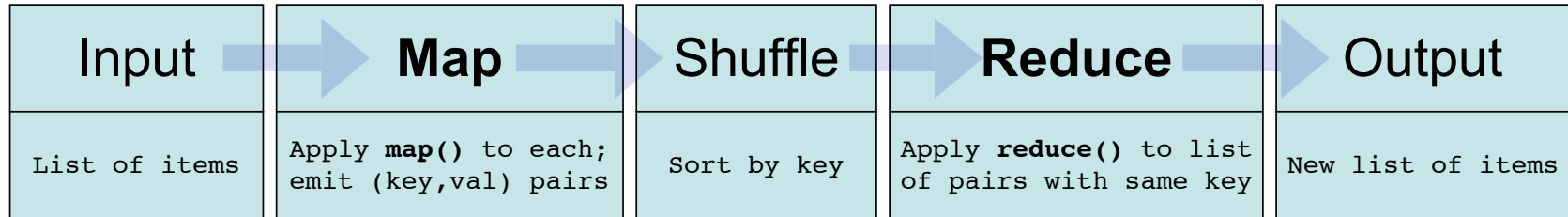
# Code Example

```
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);
```

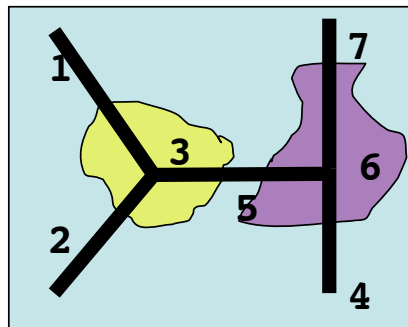
# Join, but no pointers or keys?



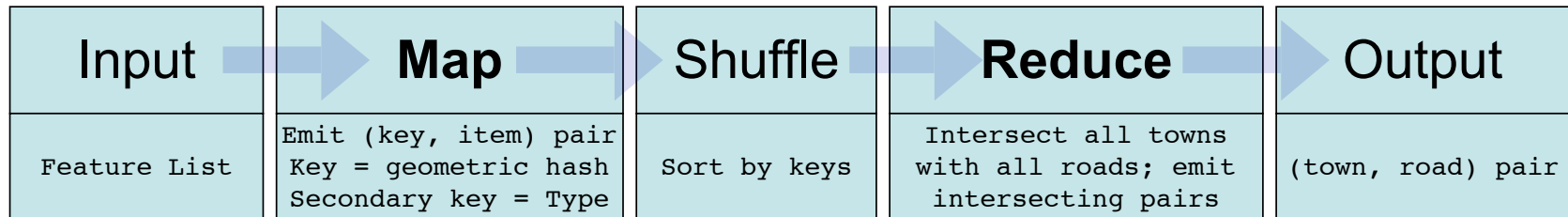
1: Road
2: Road
3: Town
4: Road
5: Road
6: Town
7: Road



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

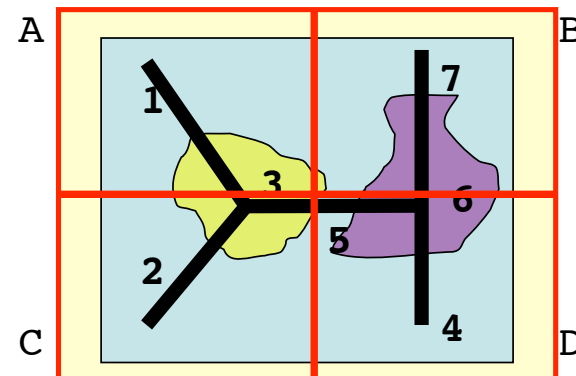


# Bucketing (or) Grace Hash Join

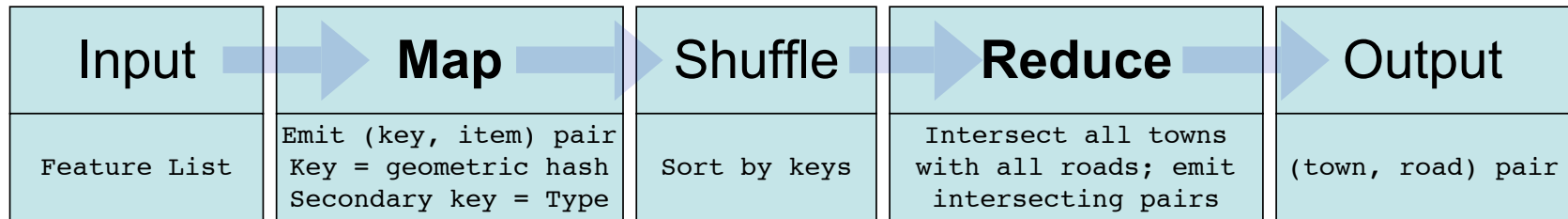


1: Road
2: Road
3: Town
4: Road
5: Road
6: Town
7: Road

(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



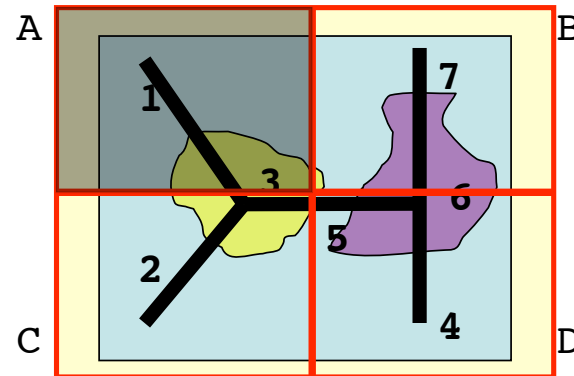
1: Road
2: Road
3: Town
4: Road
5: Road
6: Town
7: Road

(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)

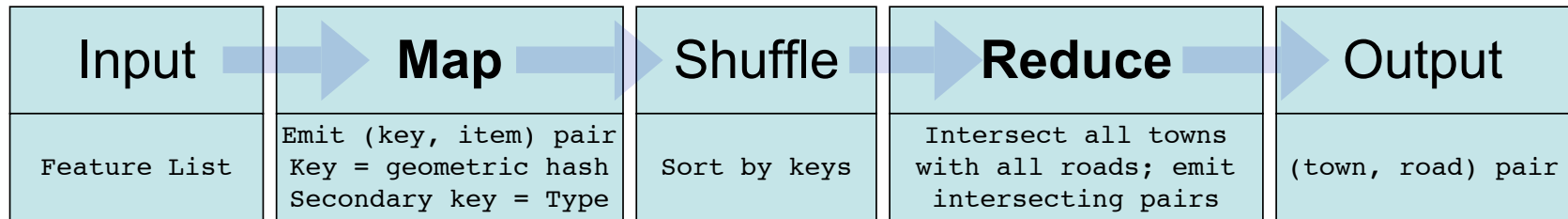


(A-Town, 3)
(A-Road, 1)

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



# Reduce on Key B



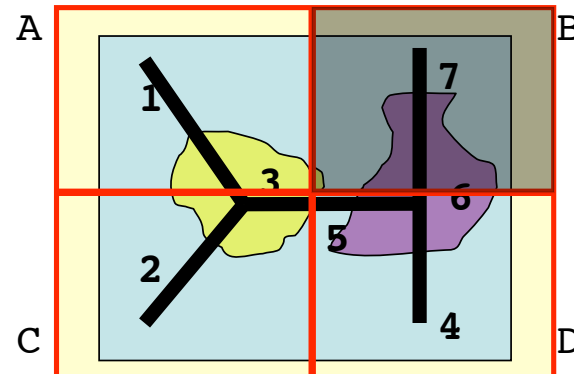
1: Road
2: Road
3: Town
4: Road
5: Road
6: Town
7: Road

(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)

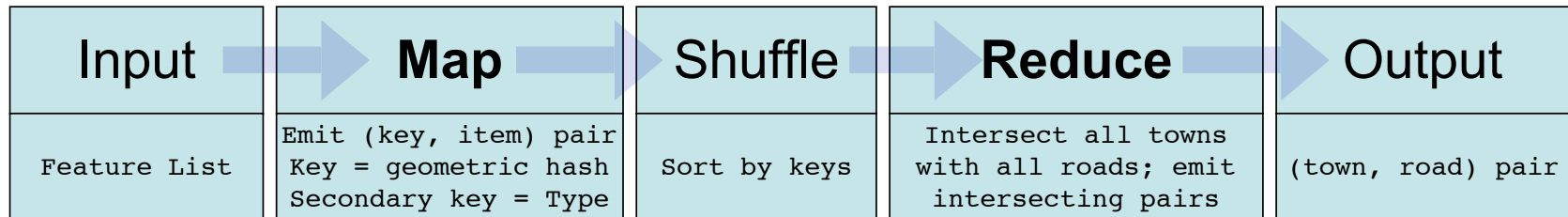


(B-Town, 3)
(B-Town, 6)
(B-Road, 7)

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



# Reduce on Key C



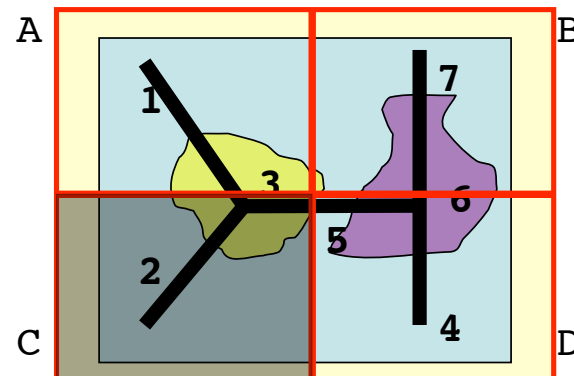
1: Road
2: Road
3: Town
4: Road
5: Road
6: Town
7: Road

(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)

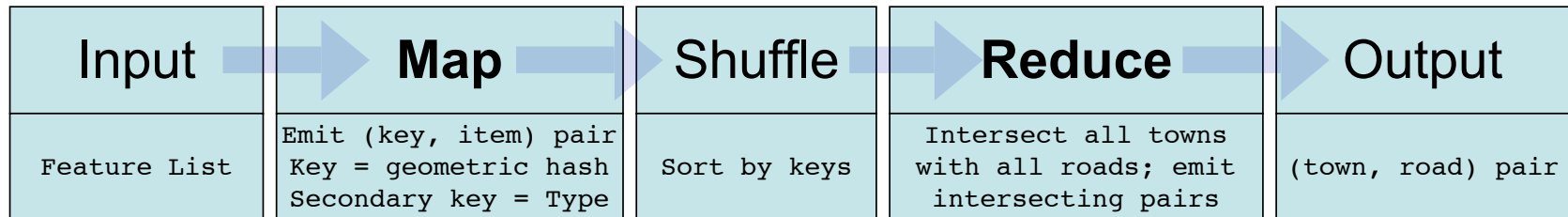


(C-Town, 3)
(C-Road, 1)
(C-Road, 5)
(C-Road, 2)

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



# Reduce on Key D



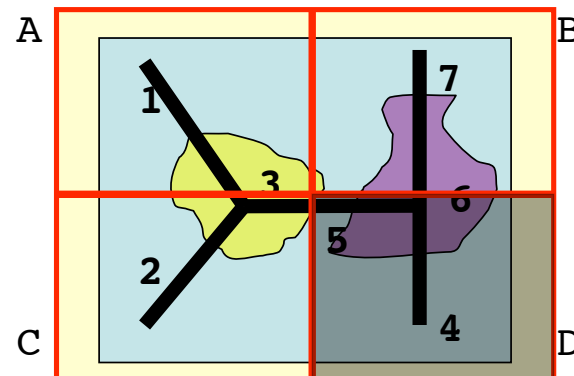
1: Road
2: Road
3: Town
4: Road
5: Road
6: Town
7: Road

(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)

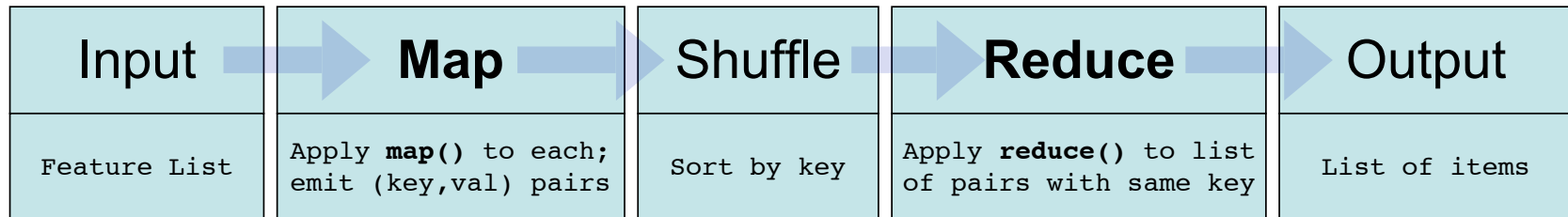


(D-Town, 6)
(D-Road, 4)
(D-Road, 5)
(D-Road, 7)

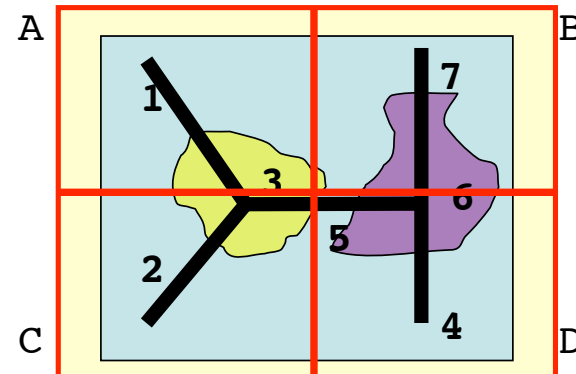
(6, 4)
(6, 5)
(6, 7)



# Output... not quite...

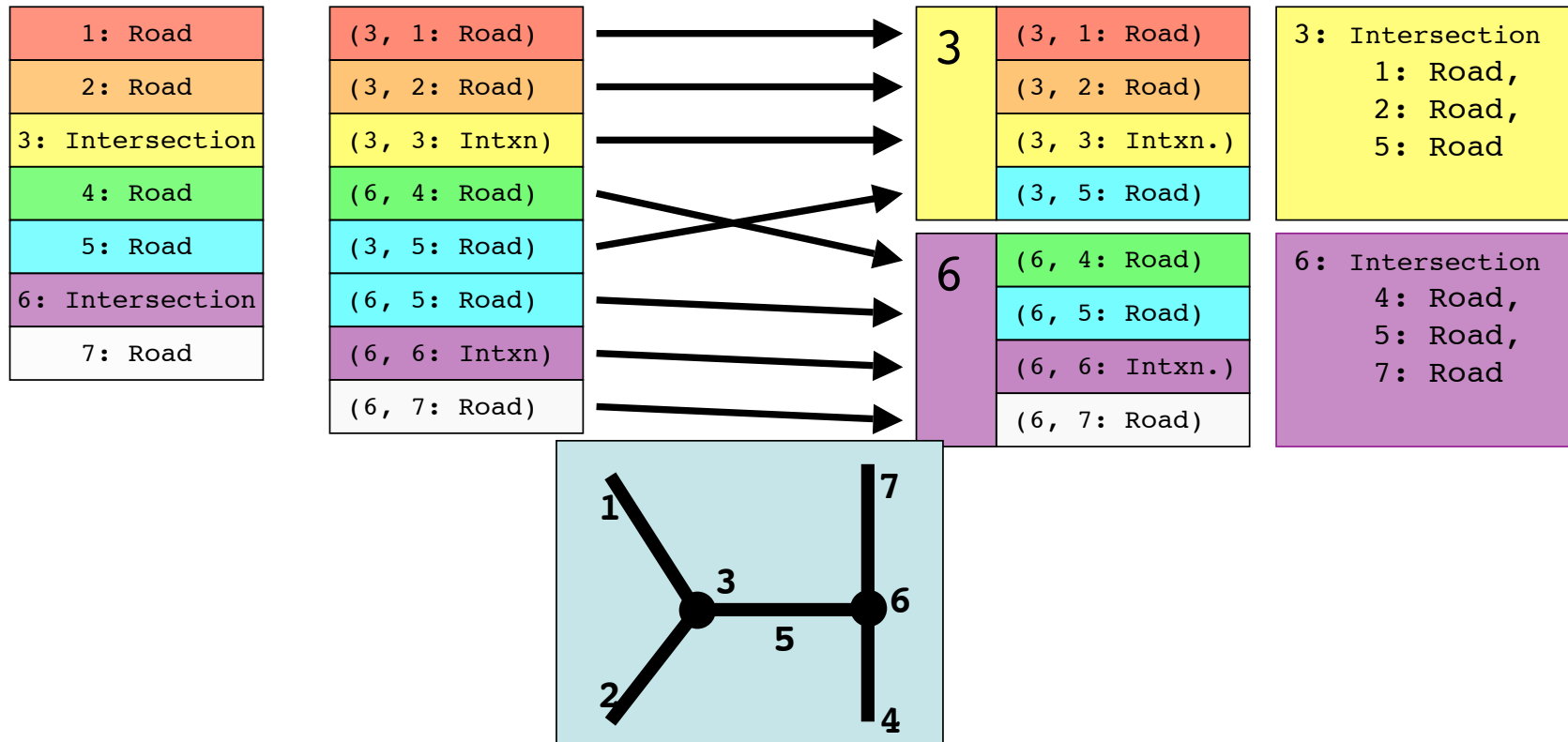
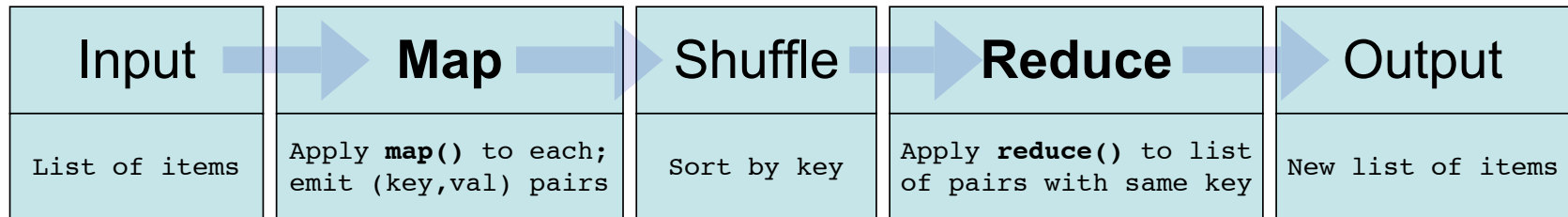


1: Road
2: Road
3: Town
4: Road
5: Road
6: Town
7: Road

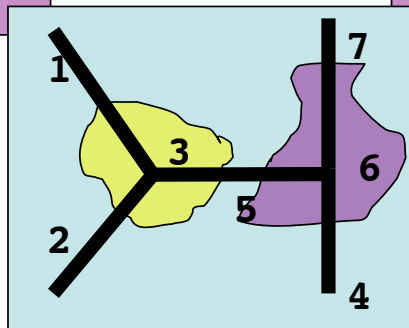
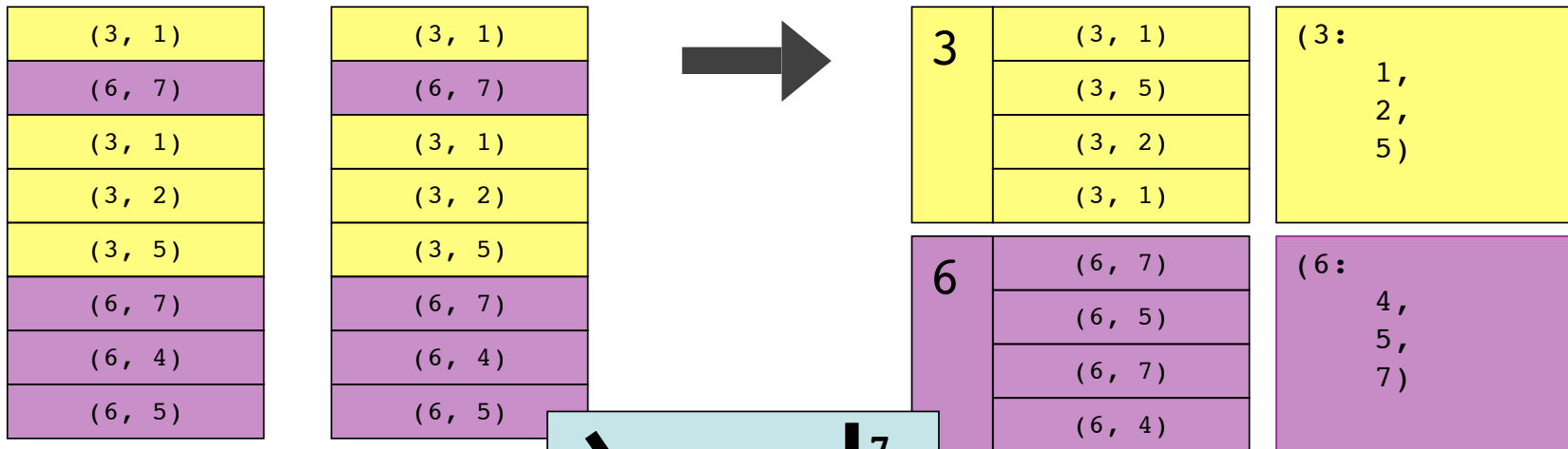
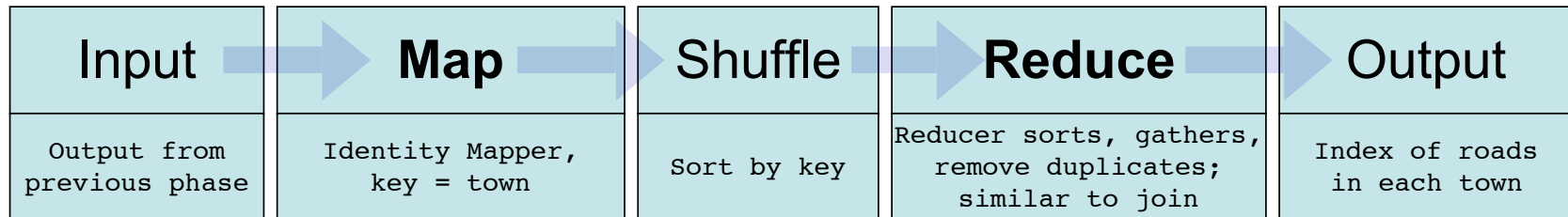


(3, 1)
(6, 7)
(3, 1)
(3, 2)
(3, 5)
(6, 7)
(6, 4)
(6, 5)

# ...recall earlierJoin Pattern

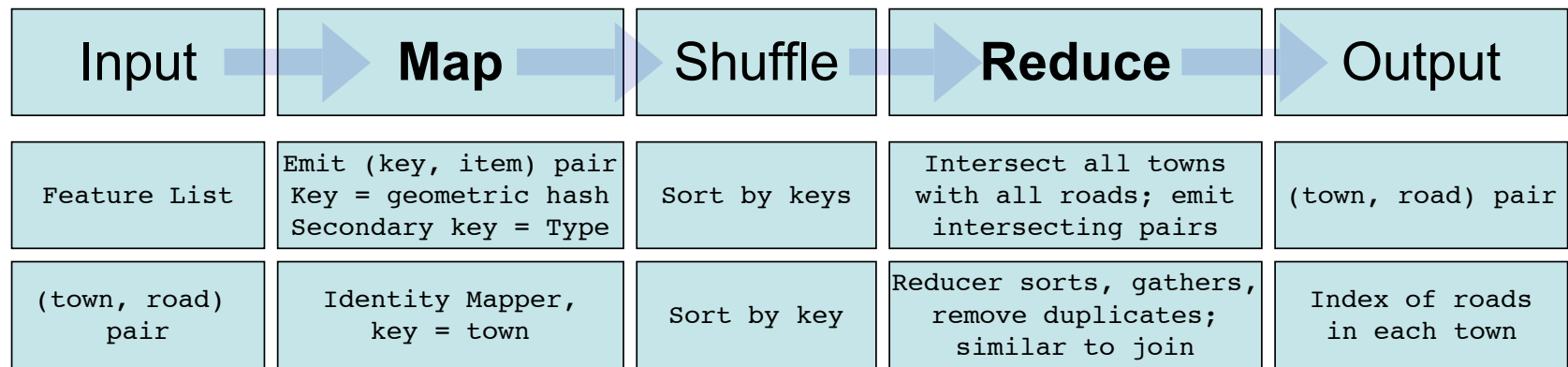


# Recursive Key Join Pattern

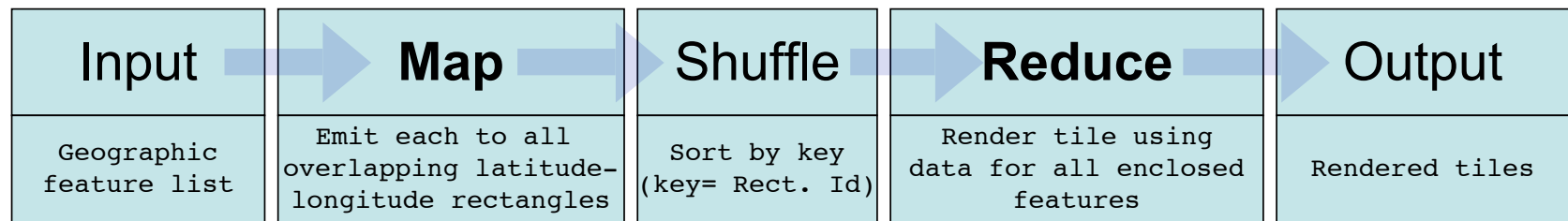


Could use 2ndry keys to avoid reduce sort(),  
eg: (6-7, 7)

# Chained MapReduce's Pattern



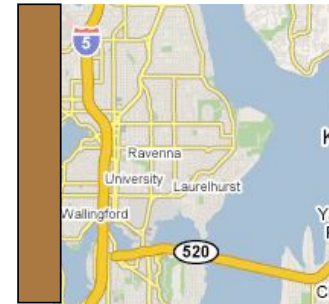
# Distributing Costly Computation: e.g. Rendering Map Tiles



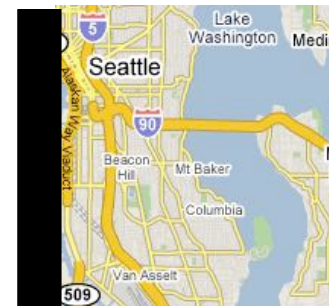
I-5
Lake Washington
WA-520
I-90
...

(N, I-5)
(S, I-5)
(N, Lake Wash.)
(S, Lake Wash.)
(N, WA-520)
(S, I-90)
...

N	(N, I-5)
	(N, Lake Wash.)
	(N, WA-520)
	...



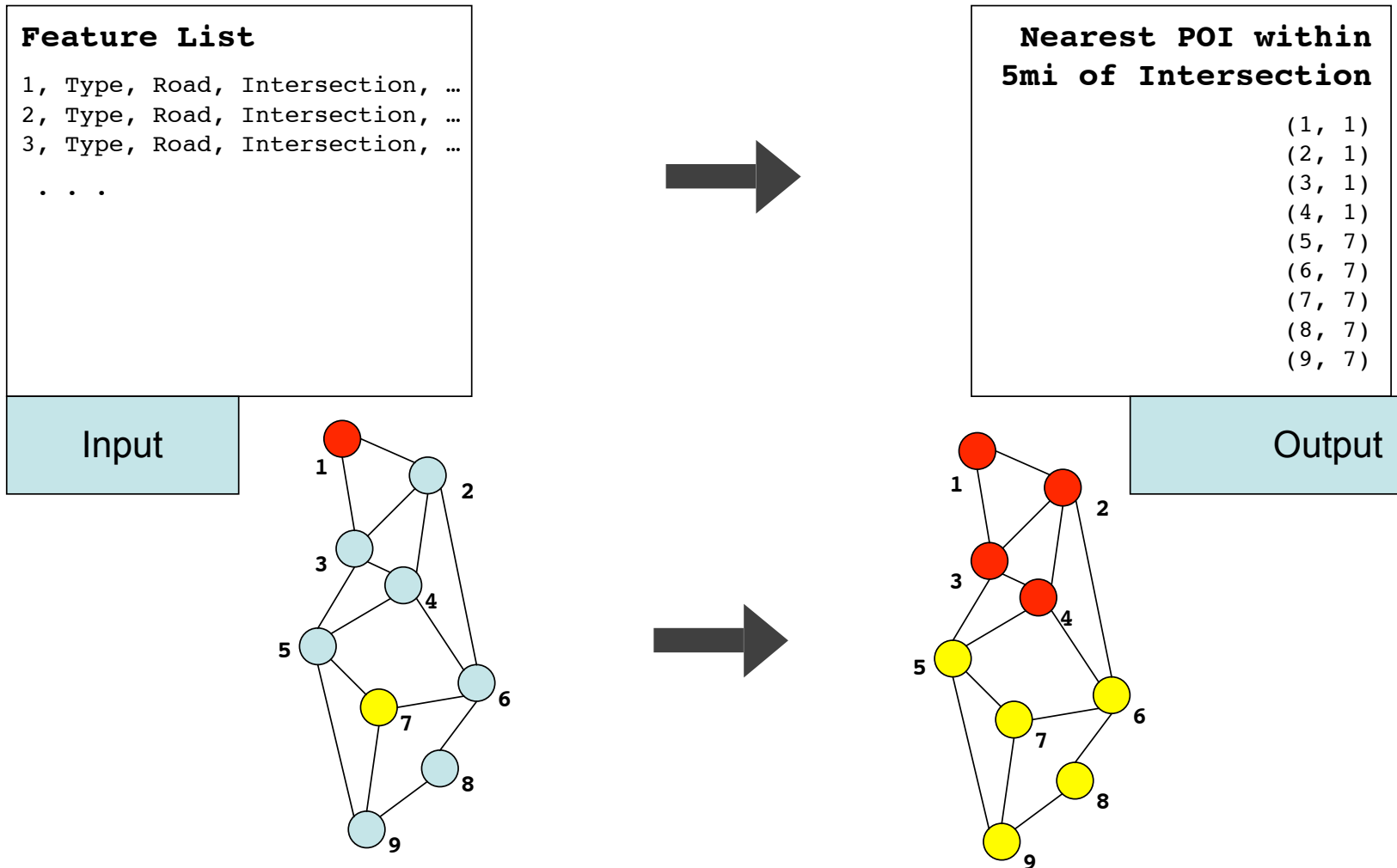
S	(S, I-5)
	(S, Lake Wash.)
	(S, I-90)
	...



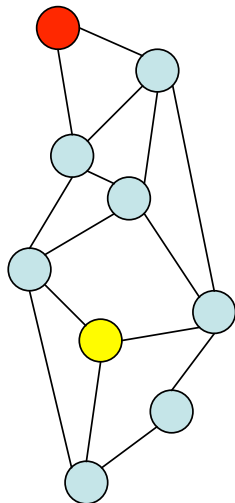
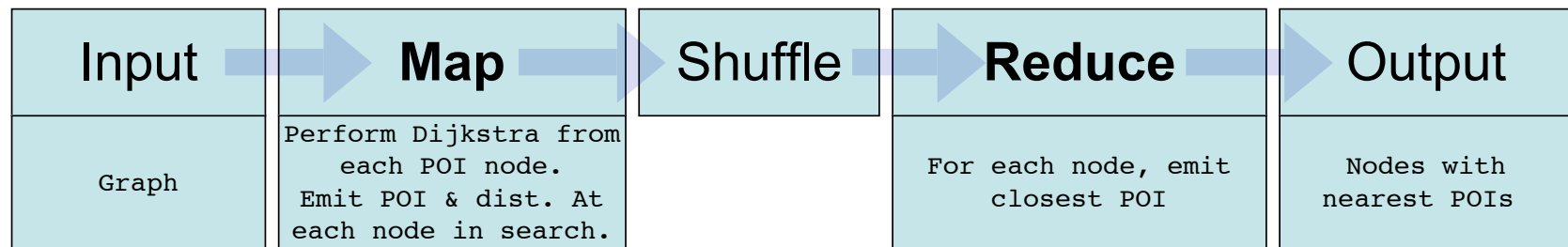
(Bucket pattern)

(Parallel rendering)

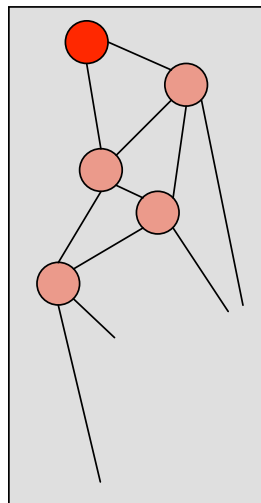
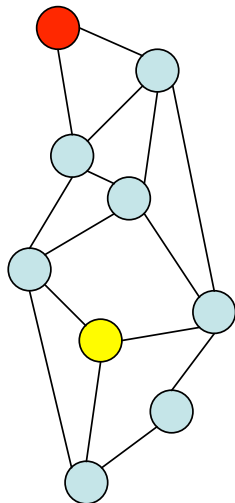
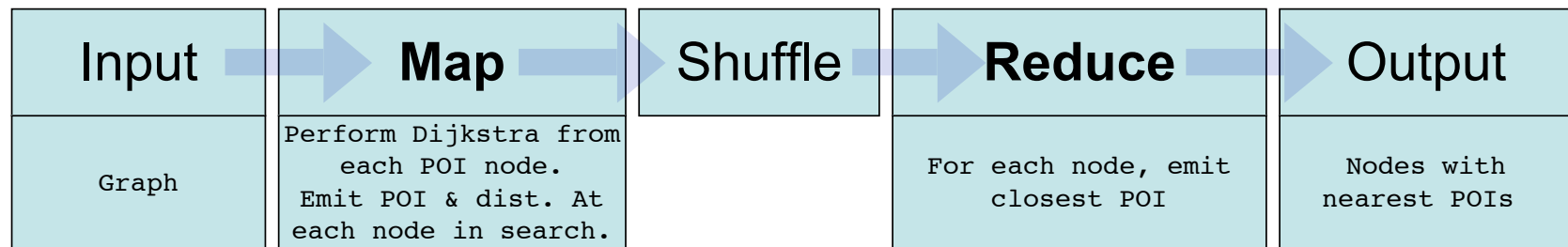
# Finding Nearest Points Of Interest (POIs)



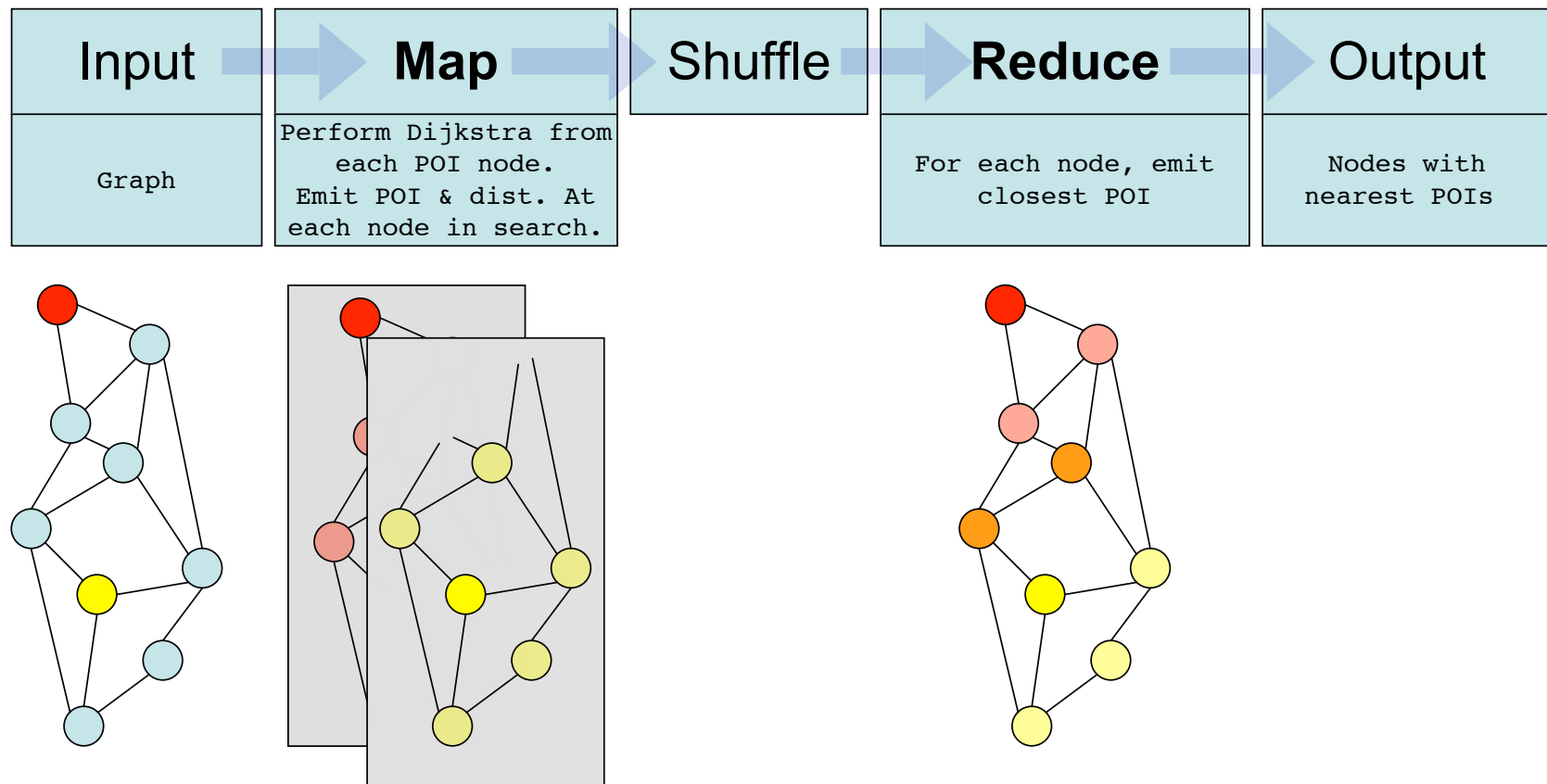
# Finding Nearest POI on a Graph



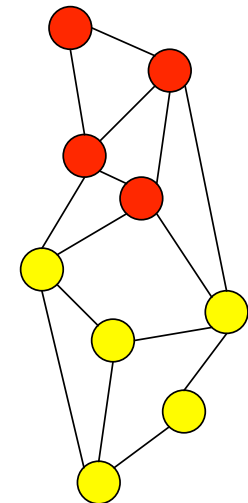
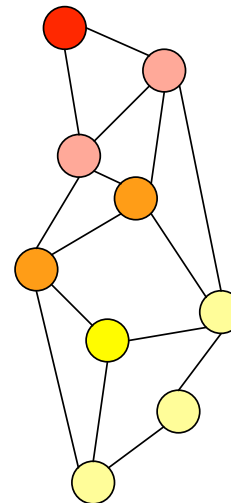
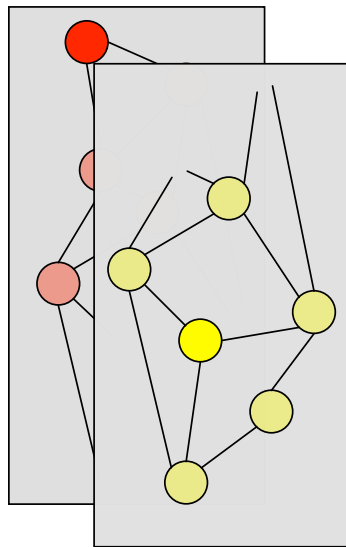
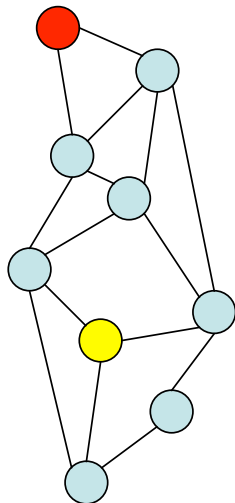
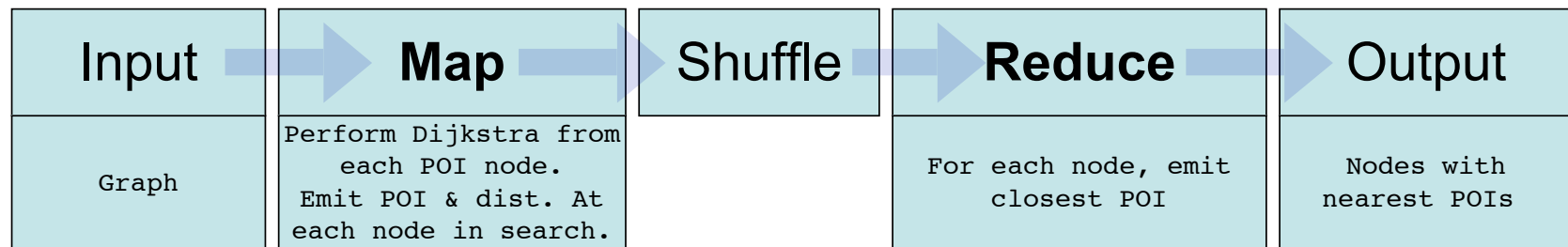
# Finding Nearest POI on a Graph



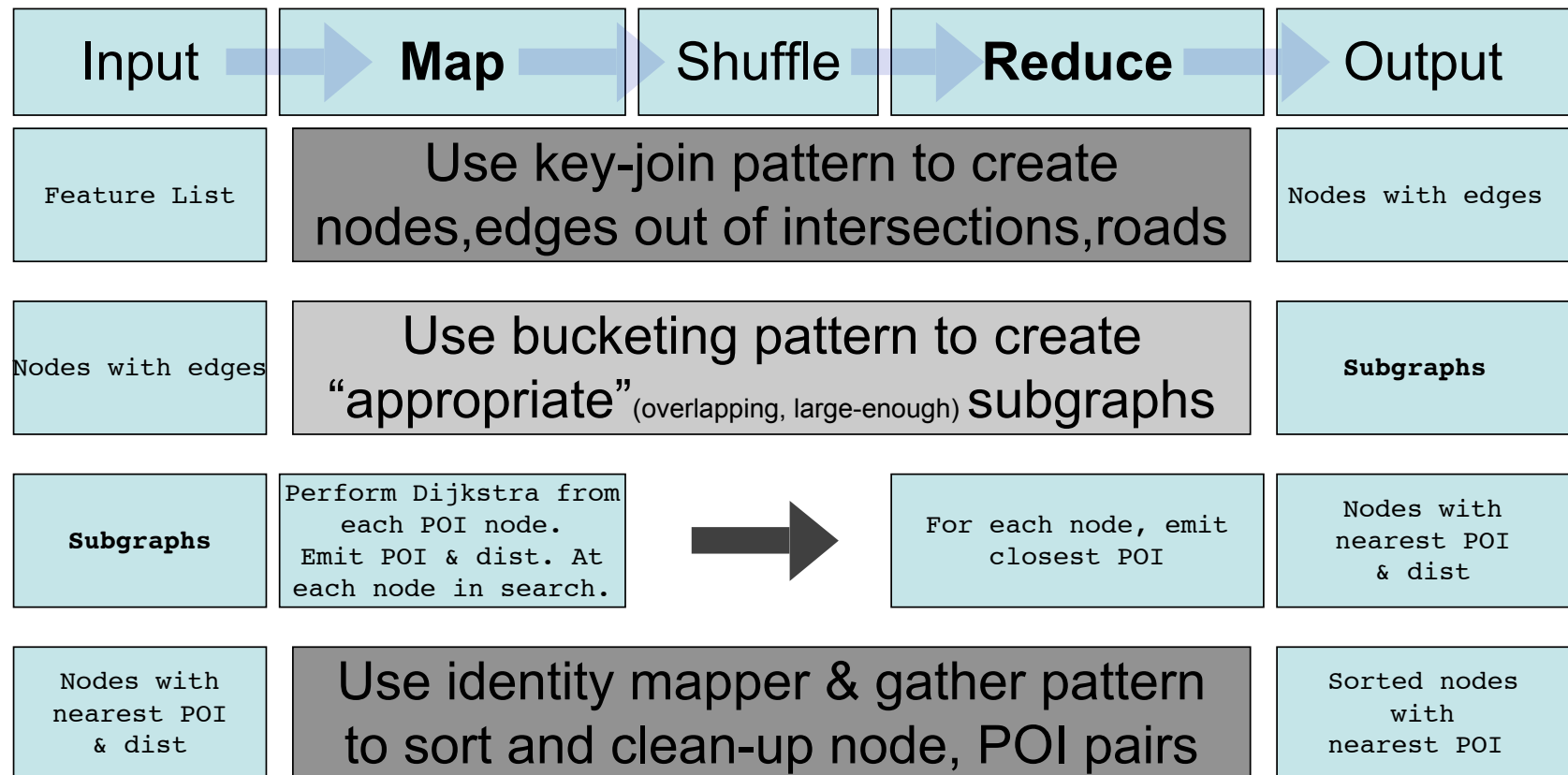
# Finding Nearest POI on a Graph



# Finding Nearest POI on a Graph



# Putting it all together: 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](mailto:barryb@google.com)

