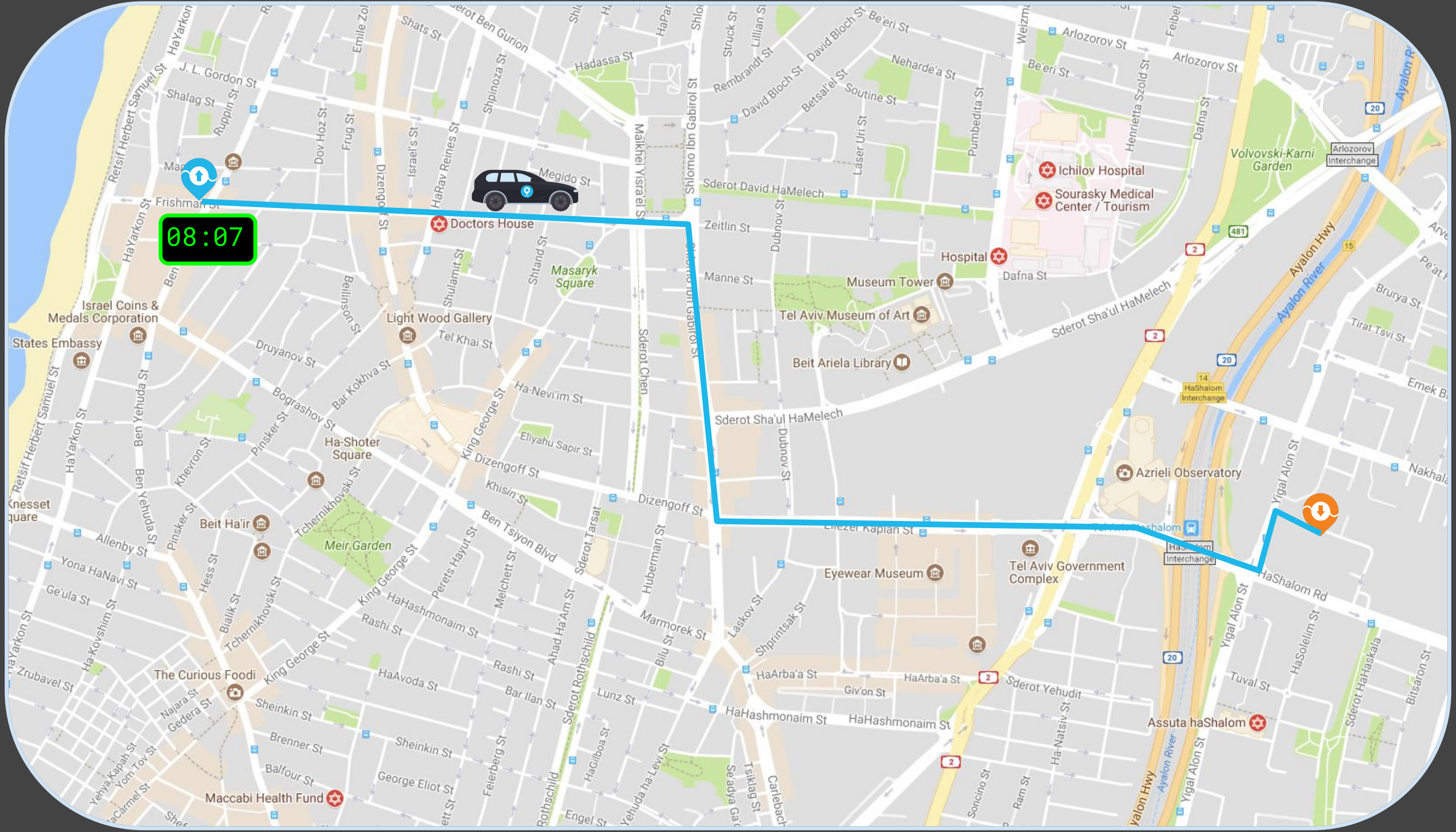
Offline Solutions to Online Problems

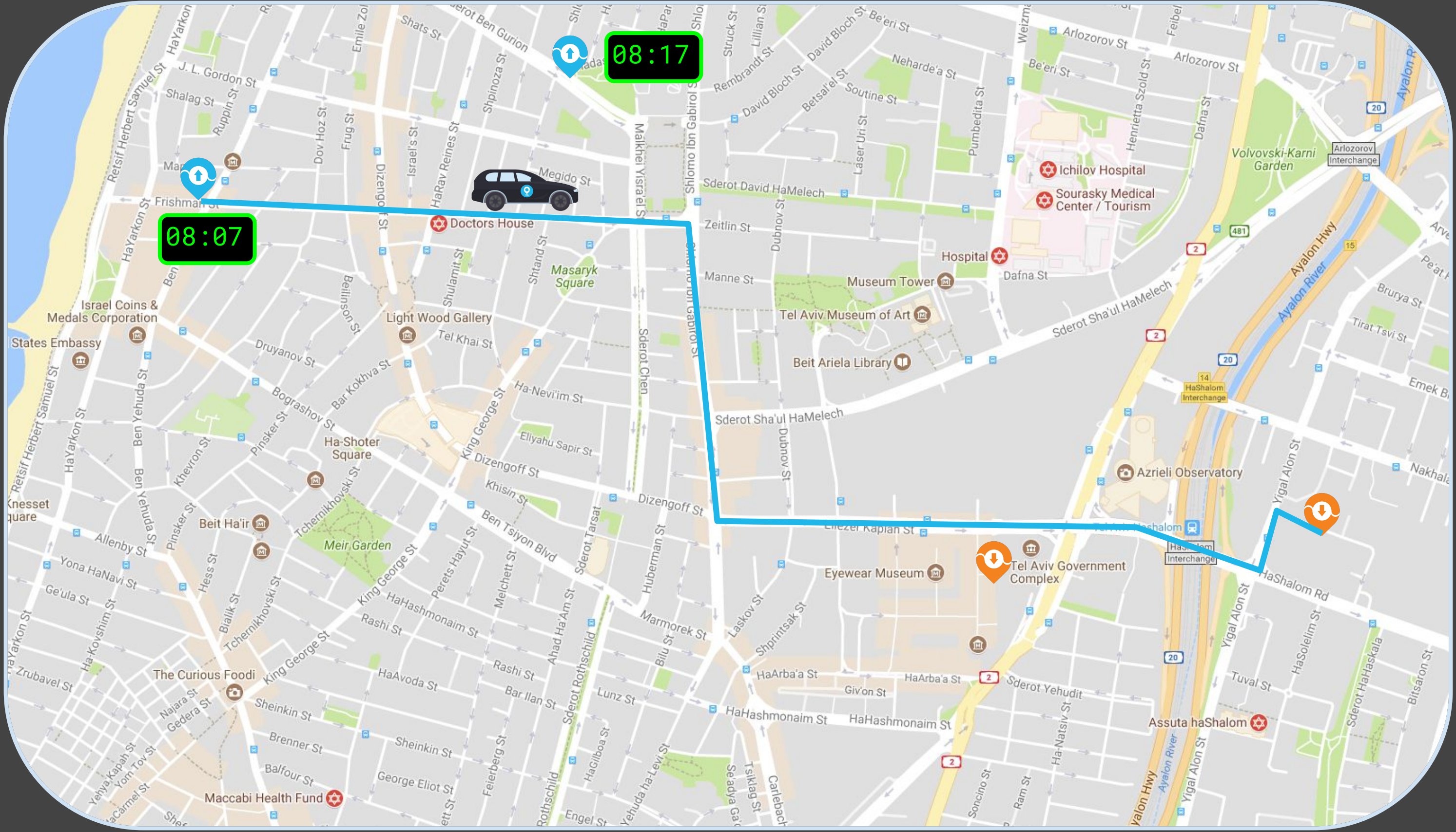
Dalya Gartzman

via

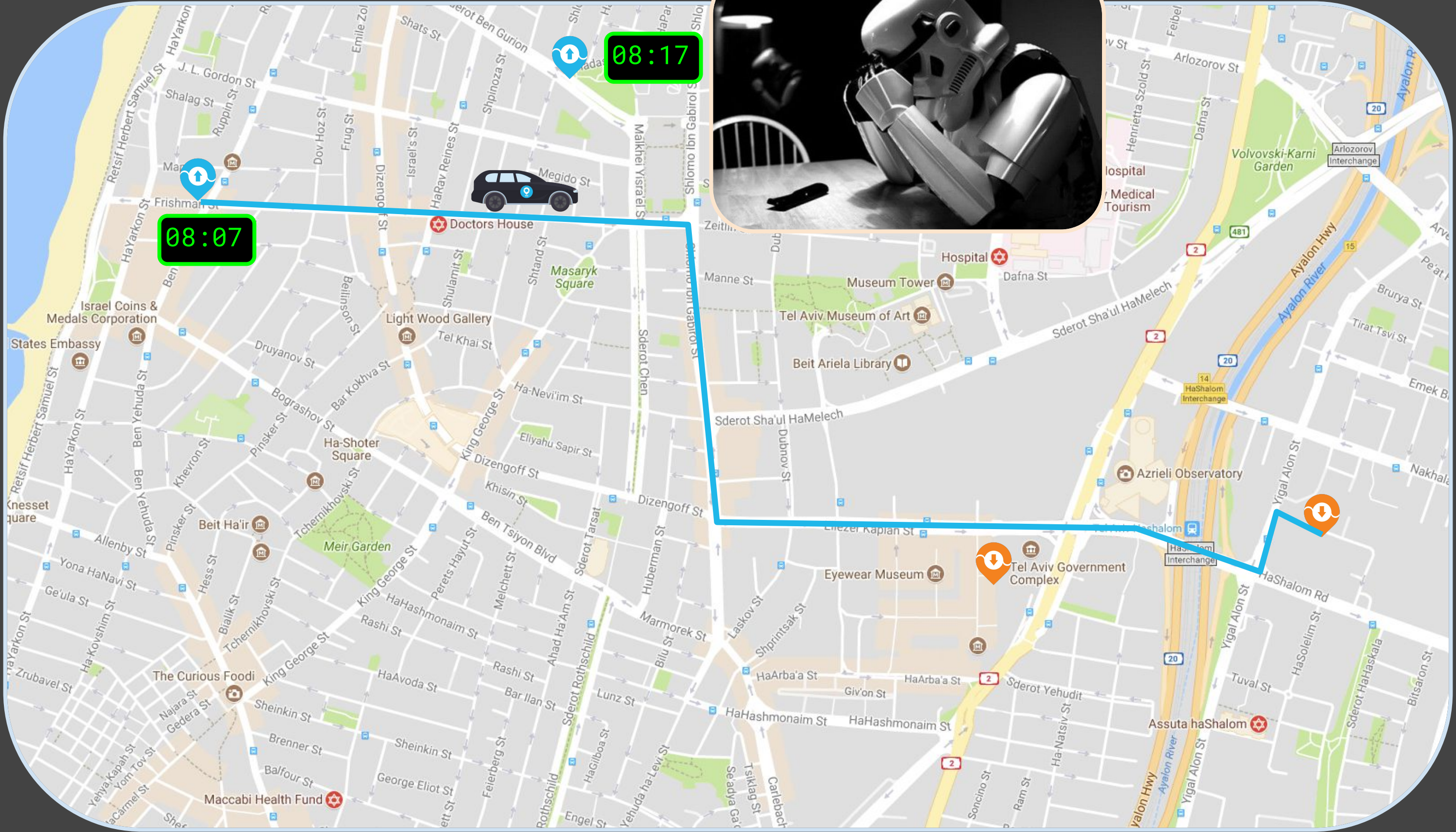
What does it mean?



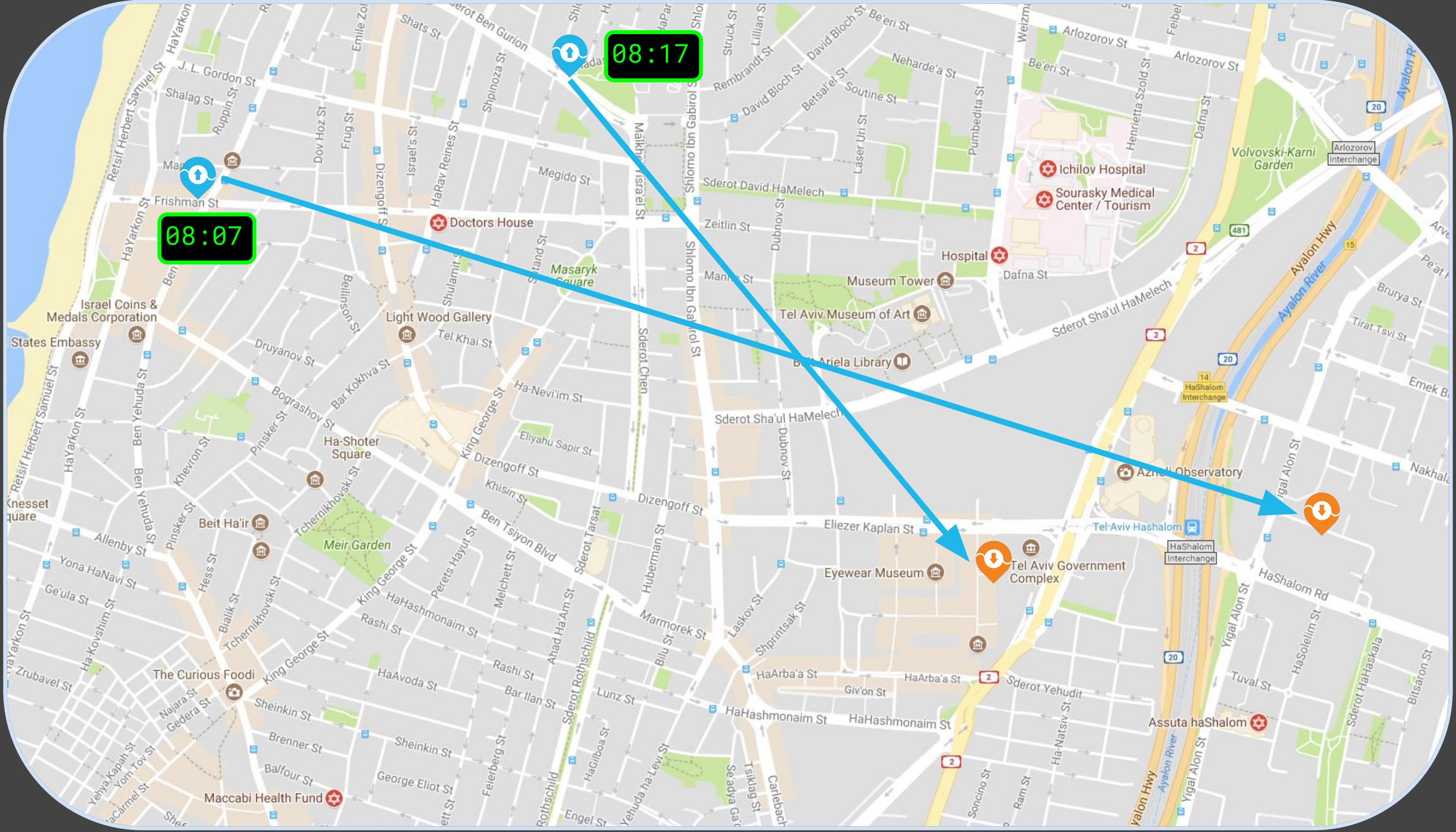
What does it mean?



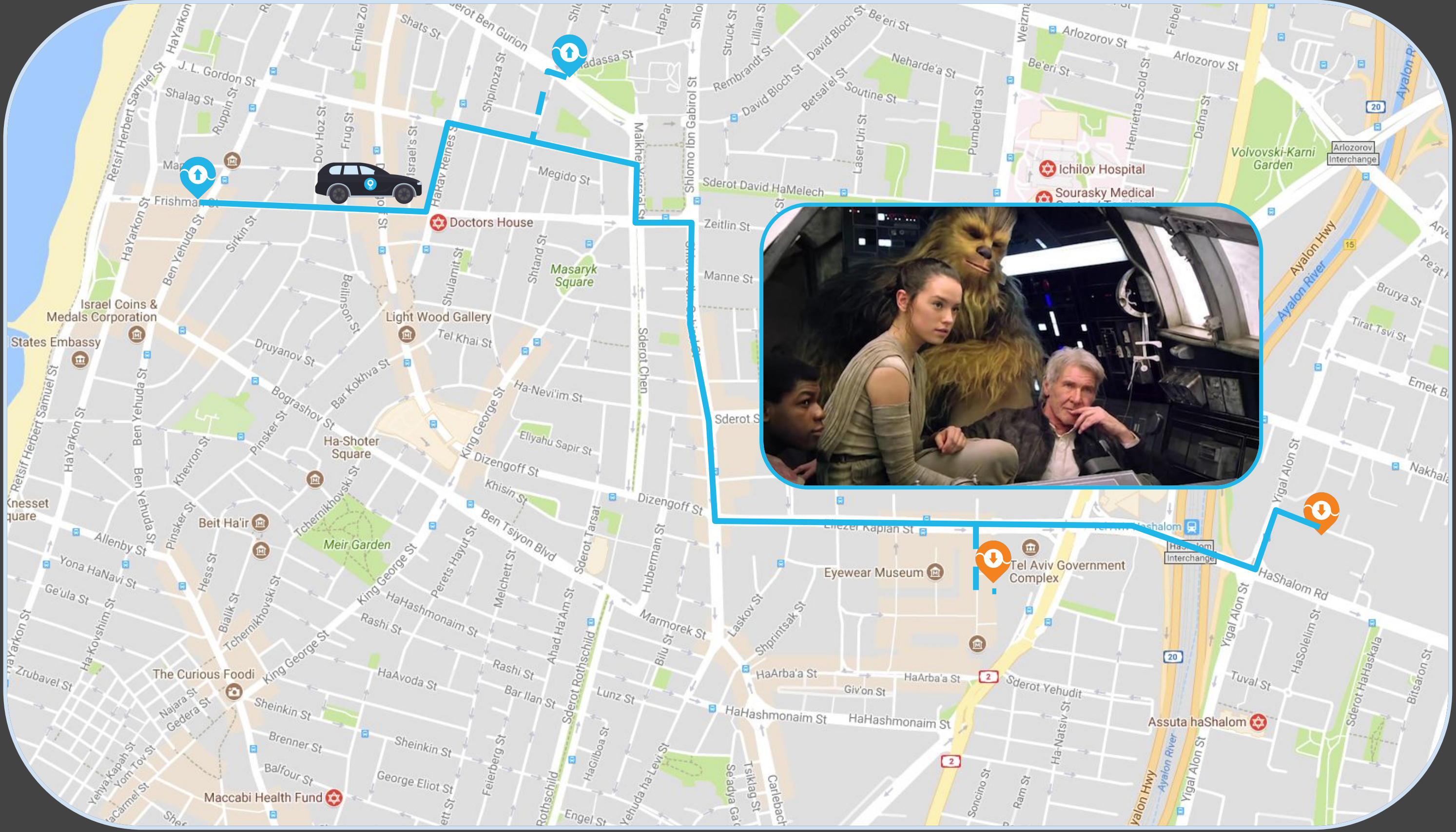
What does it mean?



What does it mean?



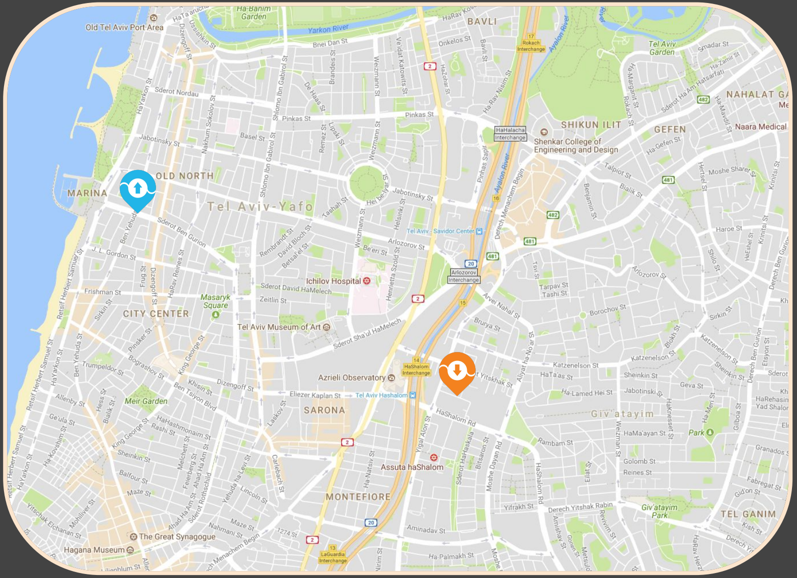
What does it mean?

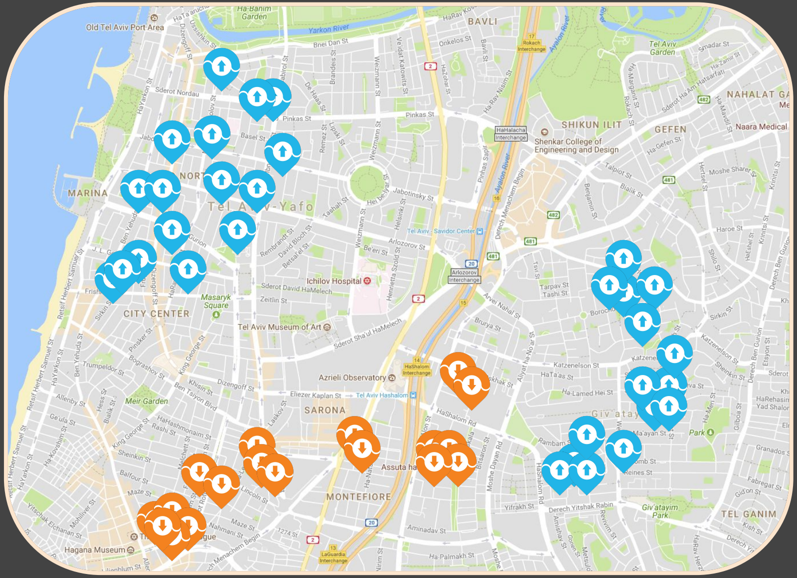


Our Motivation

online problem - individual actions

offline solution - smarter decisions





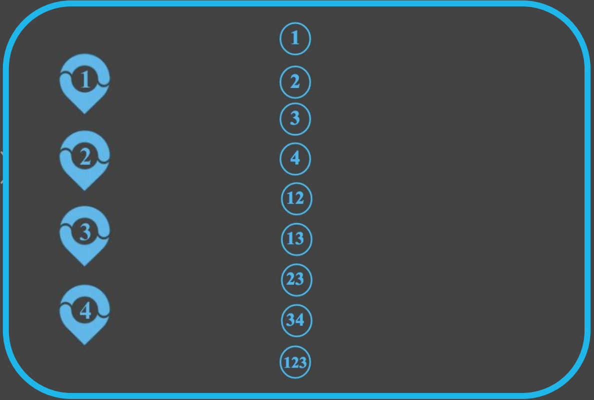
networkx

```
import networkx as nx

def BuildGraph(requests, vans):
   G = nx.Graph()

   # build left side
   G.add_nodes_from([ (r.name, {‘data’:r.data}) for r in requests ])

   # build middle side
   G.add_nodes_from([ (c.name, {‘data’:c.data})
                      for c in GetClusters(requests) ])
   G.add_edges_from([ (r, c) for r,c in requests,clusters])

   # build right side
   G.add_nodes_from([ (v.name, {‘data’:v.data}) for v in vans ])
   G.add_edges_from([(c,v) for c,v in clusters, vans
                           if CanVanTakeCluster(v,c)])

   return G
```

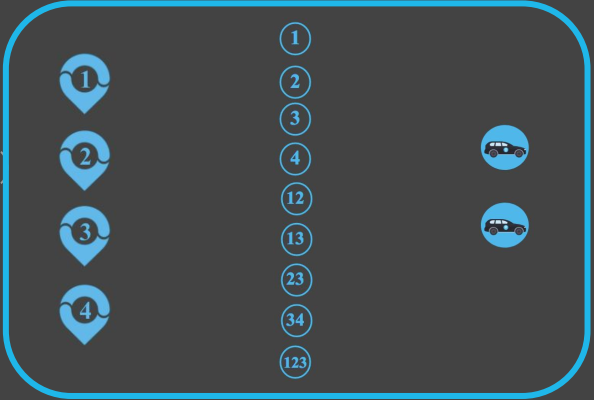
```
import networkx as nx

def BuildGraph(requests, vans):
 G = nx.Graph()

  # build left side
  G.add_nodes_from([ (r.name, {‘data’:r.data}) for r in requests ])

  # build middle side
  G.add_nodes_from([ (c.name, {‘data’:c.data})
                     for c in GetClusters(requests) ])
  G.add_edges_from([ (r, c) for r,c in requests,clusters ])

  # build right side
  G.add_nodes_from([ (v.name, {‘data’:v.data}) for v in vans ])
  G.add_edges_from([(c,v) for c,v in clusters, vans
                          if CanVanTakeCluster(v,c)])

 return G
```

```
import networkx as nx

def BuildGraph(requests, vans):
   G = nx.Graph()

   # build left side
   G.add_nodes_from(
     [ (r.name, {‘data’:r.data})
      for r in requests ])

   # build middle side
   G.add_nodes_from([ (c.name, {‘data’:c.data})
                      for c in GetClusters(requests) ])
   G.add_edges_from([ (r, c) for r,c in requests,clusters ])

   # build right side
   G.add_nodes_from([ (v.name, {‘data’:v.data}) for v in vans ])
   G.add_edges_from([(c,v) for c,v in clusters, vans if CanVanTakeCluster(v,c)])
   return G
```

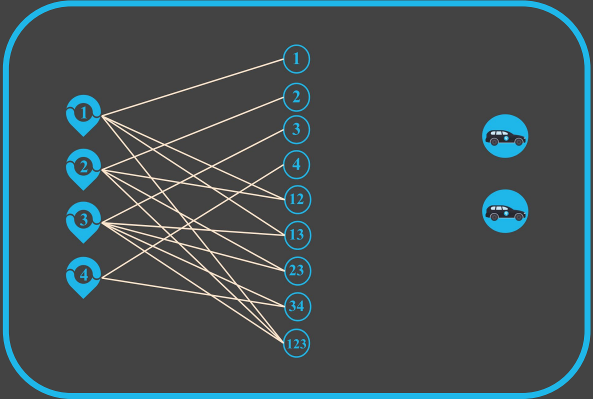
```
import networkx as nx
def BuildGraph(requests, vans):
   G = nx.Graph()
   # build left side
   G.add_nodes_from([(r.name, {‘data’:r.data}) for r in requests])

   # build middle side
   G.add_nodes_from([
     (c.name,{‘data’:c.data})
     for c in GetClusters(requests)])
   G.add_edges_from([ (r, c) for r,c in requests,clusters ])
   # build right side
   G.add_nodes_from([ (v.name, {‘data’:v.data}) for v in vans ])
   G.add_edges_from([(c,v) for c,v in clusters, vans if CanVanTakeCluster(v,c)])

   return G
```



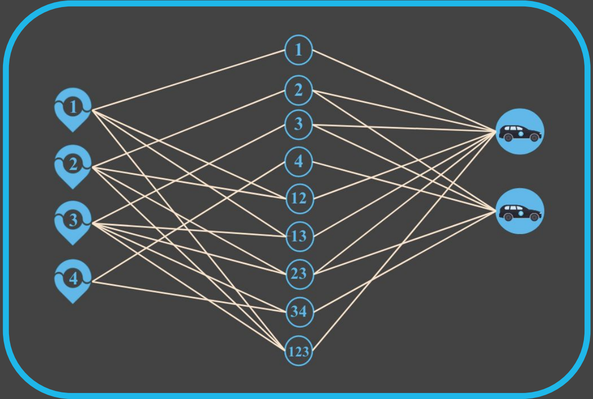
```
import networkx as nx
def BuildGraph(requests, vans):
   G = nx.Graph()
   # build left side
   G.add_nodes_from([(r.name, {'data':r.data}) for r in requests])

   # build middle side
   G.add_nodes_from([ (c.name, {'data':c.data})
                      for c in GetClusters(requests) ])
   G.add_edges_from([ (r, c) for r,c in requests,clusters ])

   # build right side
   G.add_nodes_from(
         [ (v.name, {'data':v.data})
          for v in vans ])
   G.add_edges_from([(c,v) for c,v in (middle_layer), vans
                           if CanVanTakeClique(v,c)])
   return G
```



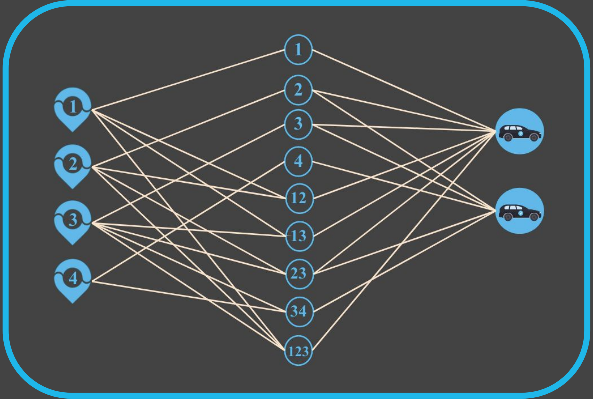
```
import networkx as nx

def BuildGraph(requests, vans):
   G = nx.Graph()

   # build left side
   G.add_nodes_from([ (r.name, {'data':r.data}) for r in requests

   # build middle side
   G.add_nodes_from([ (c.name, {'data':c.data})
                      for c in GetClusters(requests) ])
   G.add_edges_from(
   [(r,c) for r,c in requests,clusters])

   # build right side
   G.add_nodes_from([ (v.name, {'data':v.data}) for v in vans ])
   G.add_edges_from([(c,v) for c,v in clusters, vans
                           if CanVanTakeCluster(v,c)])

   return G
```



```
import networkx as nx

def BuildGraph(requests, vans):
    G = nx.Graph()

    # build left side
    G.add_nodes_from([ (r.name, {'data':r.data}) for r in requests

    # build middle side
    G.add_nodes_from([ (c.name, {'data':c.data})
                       for c in GetClusters(requests) ])
    G.add_edges_from([ (r,c) for r,c in requests,clusters ])

    # build right side
    G.add_nodes_from([ (v.name, {'data':v.data}) for v in vans ])
    G.add_edges_from(
      [(c,v) for c,v in clusters,vans
        if CanVanTakeCluster(v,c)])

    return G
```



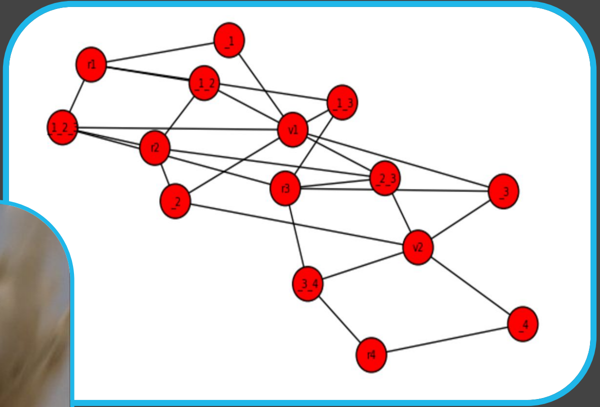
```
import networkx as nx

def BuildGraph(requests, vans):
   G = nx.Graph()

   # build left side
   G.add_nodes_from([ (r.name, {‘data’:r.data}) for r in requests

   # build middle side
   G.add_nodes_from([ (c.name, {‘data’:c.data})
                      for c in GetClusters(requests) ])
   G.add_edges_from([ (r,c) for r,c in requests,clusters ])

   # build right side
   G.add_nodes_from([ (v.name, {‘data’:v.data}) for v in vans ])
   G.add_edges_from([(c,v) for c,v in clusters, vans
                           if CanVanTakeCluster(v,c)])
```



G.draw()

```
   return G
```

```
import networkx as nx

def BuildGraph(requests, vans):
   G = nx.Graph()

   # build left side
   G.add_nodes_from([ (r.name,

   # build middle side
   G.add_nodes_from([ (c.name,
                      for c in
   G.add_edges_from([ (r,c) for

   # build right side
   G.add_nodes_from([ (v.name,
   G.add_edges_from([(c,v) for
                           if C
```



G.draw()

```
   return G
```

```
import networkx as nx
def BuildGraph(requests, vans):
   G = nx.Graph()
   # build left side
   G.add_nodes_from([ (r.name, {'data':r.data}) for r in requests ])
   # build middle side
   G.add_nodes_from([ (c.name, {'data':c.data})
                      for c in GetClusters(requests) ])
   G.add_edges_from([ (r, c) for r,c in requests,clusters ])
   # build right side
   G.add_nodes_from([ (v.name, {'data':v.data}) for v in vans ])
   G.add_edges_from([(c,v) for c,v in clusters, vans
                          if CanVanTakeCluster(v,c)])
   # define position for draw
   pos =
   {r.name: [0,idx]
   for idx,r in enumerate(requests)} + \
   { c.name: [1,idx] for idx,c in enumerate(GetClusters(G))} + \
   { v.name: [2,idx] for idx,v in enumerate(vans)}
   nx.draw(G,pos)
   return G
```

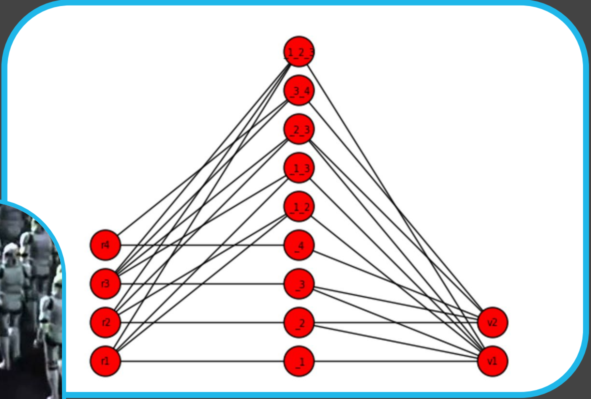
```
import networkx as nx
def BuildGraph(requests, vans):
   G = nx.Graph()
   # build left side
   G.add_nodes_from([ (r.name, {'data':r.data}) for r in requests ])
   # build middle side
   G.add_nodes_from([ (c.name, {'data':c.data})
                      for c in GetClusters(requests) ])
   G.add_edges_from([ (r,c) for r,c in requests,clusters ])
   # build right side
   G.add_nodes_from([ (v.name, {'data':v.data}) for v in vans ])
   G.add_edges_from([(c,v) for c,v in clusters, vans
                          if CanVanTakeCluster(v,c)])
  # define position for draw
  pos =

  {r.name: [0,idx] for idx,r in enumerate(requests)} + \
  { c.name: [1,idx] for idx,c
      in enumerate(GetClusters(G))} + \
  { v.name: [2,idx] for idx,v in enumerate(vans)}
   nx.draw(G,pos)
   return G
```

```
import networkx as nx
def BuildGraph(requests, vans):
   G = nx.Graph()
   # build left side
   G.add_nodes_from([ (r.name, {'data':r.data}) for r in requests ])
   # build middle side
   G.add_nodes_from([ (c.name, {'data':c.data})
                      for c in GetClusters(requests) ])
   G.add_edges_from([ (r,c) for r,c in requests,clusters ])
   # build right side
   G.add_nodes_from([ (v.name, {'data':v.data}) for v in vans ])
   G.add_edges_from([(c,v) for c,v in clusters, vans
                          if CanVanTakeCluster(v,c)])
   # define position for draw
   pos =

   {r.name: [0,idx] for idx,r in enumerate(requests)} + \
   { c.name: [1,idx] for idx,c in enumerate(GetClusters(G))} + \
   { v.name: [2,idx] for idx,v
       in enumerate(vans)}
   nx.draw(G,pos)
   return G
```

```
import networkx as nx
def BuildGraph(requests, vans):
    G = nx.Graph()
    # build left side
    G.add_nodes_from([ (r.name, {'data':r.data}) for r in requests
    # build middle side
    G.add_nodes_from([ (c.name,
                         for c in
    G.add_edges_from([ (r,c) fo
    # build right side
    G.add_nodes_from([ (v.name,
    G.add_edges_from([(c,v) for
                              if (
   # define position for draw
    pos =
   {r.name: [0,idx] for idx,r in
   { c.name: [1,idx] for idx,c i
   { v.name: [2,idx] for idx,v in enumerate(vans)}
```

nx.draw(G,pos)

```
    return G
```



networkx

1. List [comprehension]
2. Helper functions for logical structure
3. Use “pos” for easy drawing