# Algorithms and Data Structures

**COMP261**

**Graph 2: Display and Trie**

Yi Mei

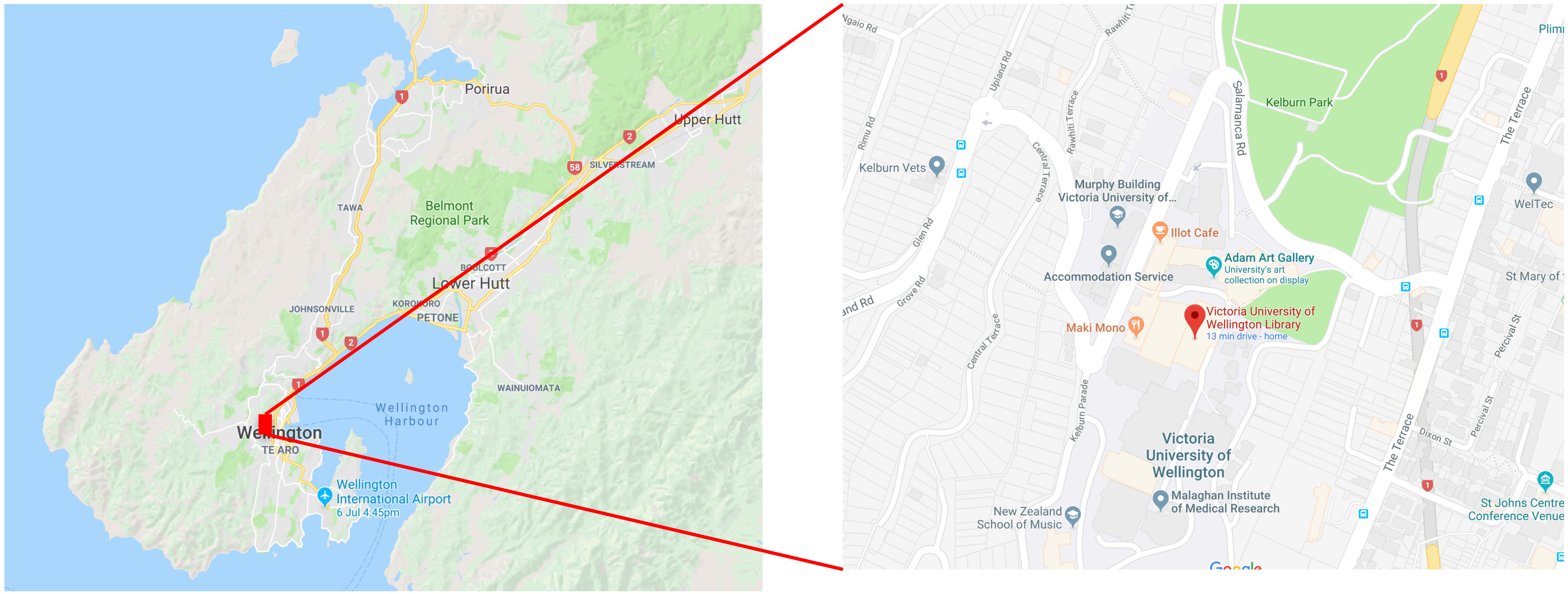
*yi.mei@ecs.vuw.ac.nz*

# Outline

- Display graph
  - Coordinate system (absolute, relative, pixel)
  - Redraw under movements (shift, zoom in/out)
- Trie

## Display Graph

- We store the entire graph in the memory
  - e.g. the map of the whole Wellington
- But we do not always display the entire graph
  - e.g. the local detailed area around my current place



# Display Graph

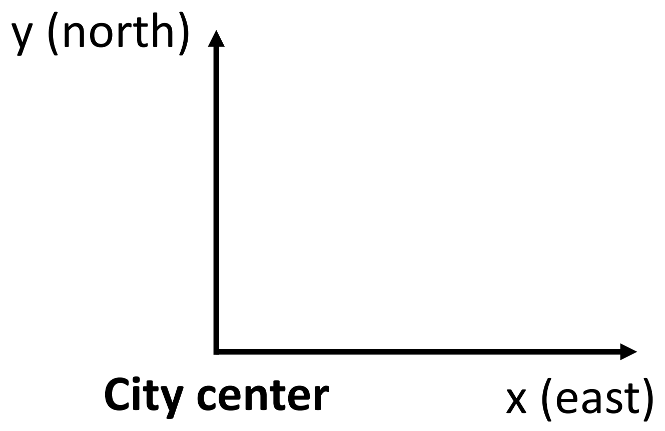
- **Problem**: given the size of display screen (number of pixels) and an area to be displayed, display the area of the graph
  - Display after shifting the area (e.g. from CBD to VUW)
  - Display after zoom-in/out

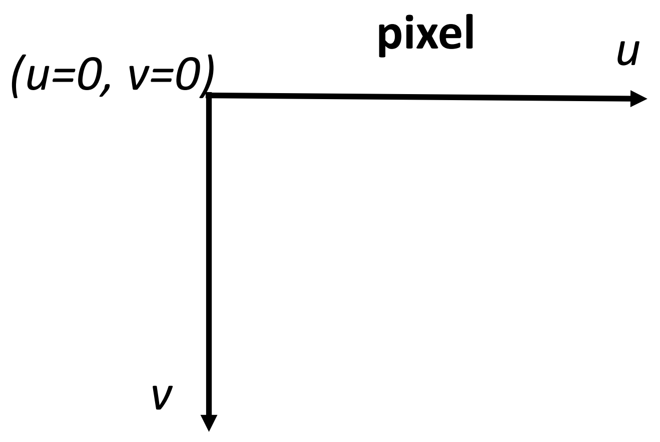
- **A simple approach** (required by assignment 1)
  - Assume the **graph** only contains **nodes** and **edges**
  - Step 1: decide which nodes to draw
  - Step 2: decide where to draw the nodes
  - Step 3: draw the links involving these nodes, and highlight the nodes

- Steps 1 and 2 are key steps
- Require coordinate systems

# Coordinate Systems for Location

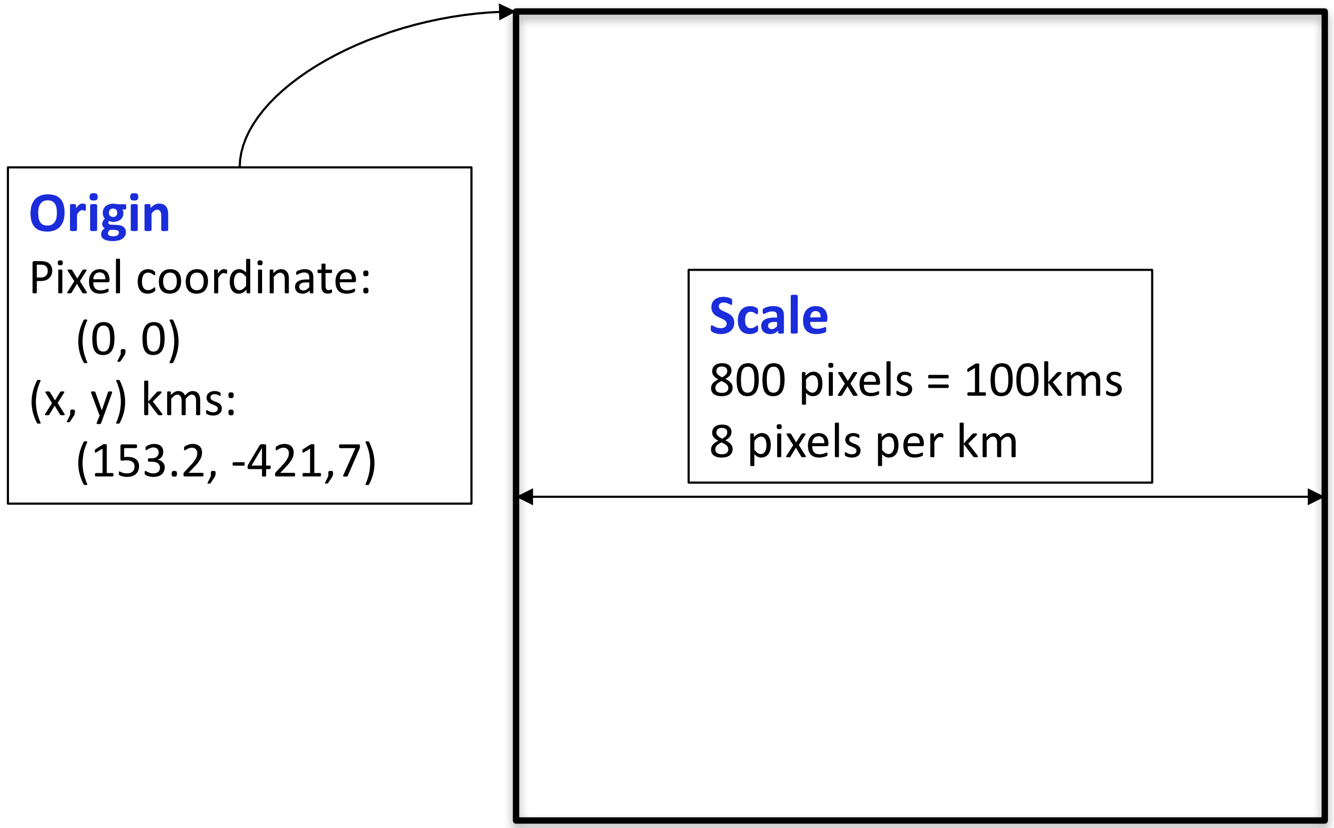
- Coordinate systems to represent locations of nodes
  - Absolute (***fixed***): latitude/longitude
  - Relative (***fixed***): x kms to the east, y kms to the north of the city center
    - Assume a flat map (the earth is a globe actually), but OK
    - Will be useful for shortest path finding
  - Pixel coordinate: for display
    - ***Dynamic***, depends on the area to display





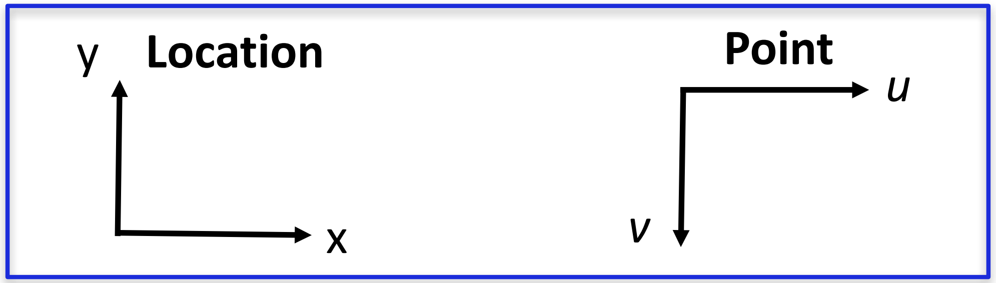
# Display Graph

- Size of display screen: number of pixels, e.g. 800 x 800
- The displayed area
  - Origin: e.g. the top-left location
  - Scale: how large the area is covered by the pixels?
    - Number of pixels per kms



# Display Graph

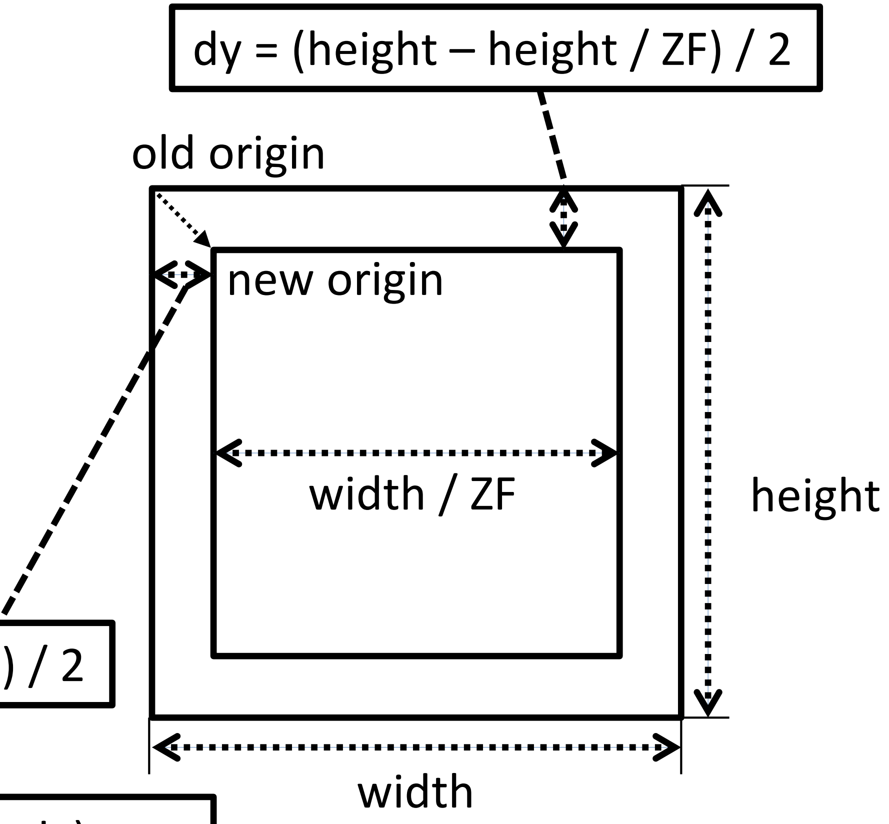
- Transform between (*x, y*) kms and pixel (*u, v*) coordinate



  - *origin.x, origin.y, origin.u = 0, origin.v = 0*
  - *scale*
- What is the pixel coordinate for a location (*loc.x, loc.y*)?
  - Distance to origin: *dx = loc.x-origin.x, dy = origin.y– loc.y*
  - From distance to #pixels: *dpx = dx * scale, dpy = dy * scale*
  - Pixel coordinates: (*origin.u + dpx, origin.v + dpy*)
- What is the (*x, y*) kms of a point (*u, v*)?
  - Distance to the origin from #pixels: *dx = u / scale, dy = v / scale*
  - *x = origin.x + dx, y = origin.y - dy*

# Adjust Display Under Movements

- Shift the displayed area: shift the origin
  - `orig.x = orig.x + dx, orig.y = orig.y + dy;`
- Zoom in/out the displayed area around the current center
  - Change both scale and origin: ZOOM_FACTOR > 1
  - Calculate width and height in kms (using topLeft, topRight, botLeft, botRight)
    - topLeft = …, topRight = …;
    - width = topRight.x – topLeft.x;
    - height = botLeft.y – topLeft.y;
  - Zoom-in: increase scale
    - scale = scale * ZOOM_FACTOR
    - width (height) /= ZOOM_FACTOR
  - Zoom-out: decrease scale
    - scale = scale / ZOOM_FACTOR
    - width (height) *= ZOOM_FACTOR



dy = (height – height / ZF) / 2

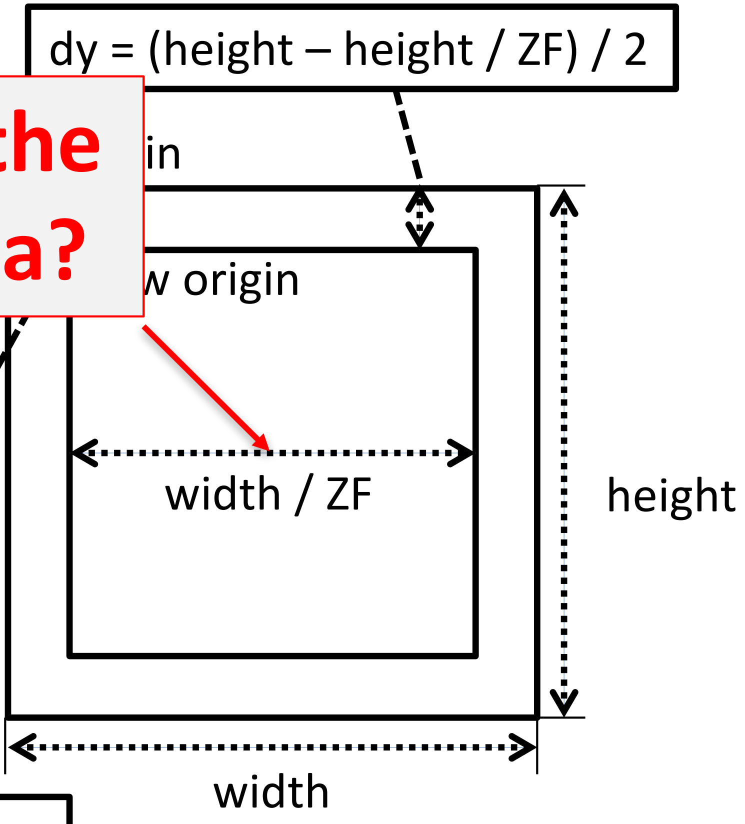
dx = (width – width / ZF) / 2

New origin = (origin.x + dx, origin.y + dy)

# Adjust Display Under Movements

- Shift the displayed area: shift the origin
  - `orig.x = orig.x + dx, orig.y = orig.y + dy;`
- Zoom in/out the displayed area around the current center
  - Change both scale and origin: ZOOM_FACTOR > 1
  - Calculate width and height in kms (using topLeft, topRight, botLeft, botRight)
    - topLeft = …, topRight = …;
    - width = topRight.x – topLeft.x;
    - width (height) /= ZOOM_FACTOR
  - Zoom-out: decrease scale
    - scale = scale / ZOOM_FACTOR
    - width (height) *= ZOOM_FACTOR

**What if the origin is set to the center of the displayed area?**

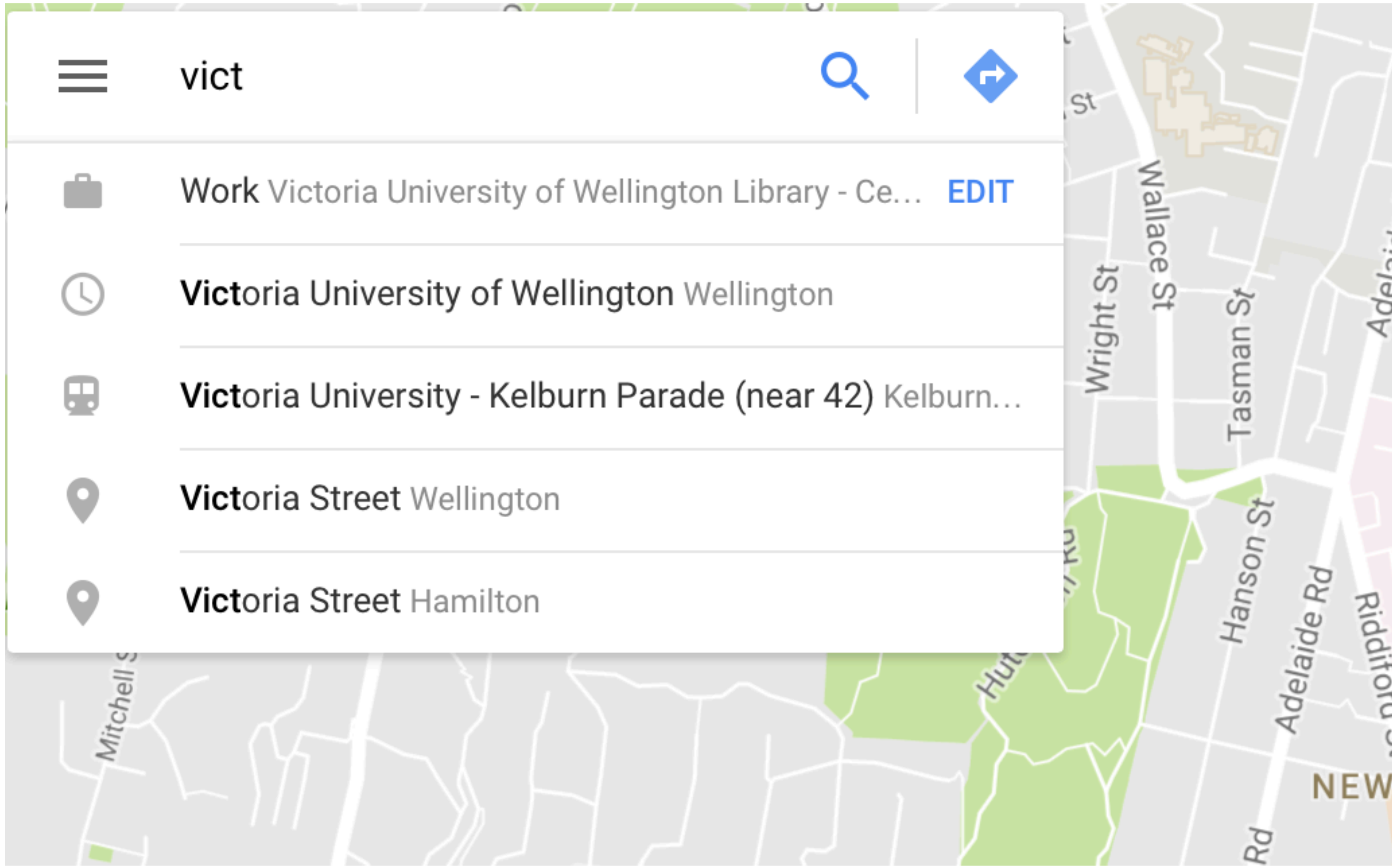


dx = (width – width / ZF) / 2

New origin = (origin.x + dx, origin.y + dy)

# Trie

- Pop-up matched results very quickly


vict

Work Victoria University of Wellington Library - Ce... EDIT

Victoria University of Wellington Wellington

Victoria University - Kelburn Parade (near 42) Kelburn...

Victoria Street Wellington

Victoria Street Hamilton

**Trie is the data structure to achieve this**

## Example: Road Name Search

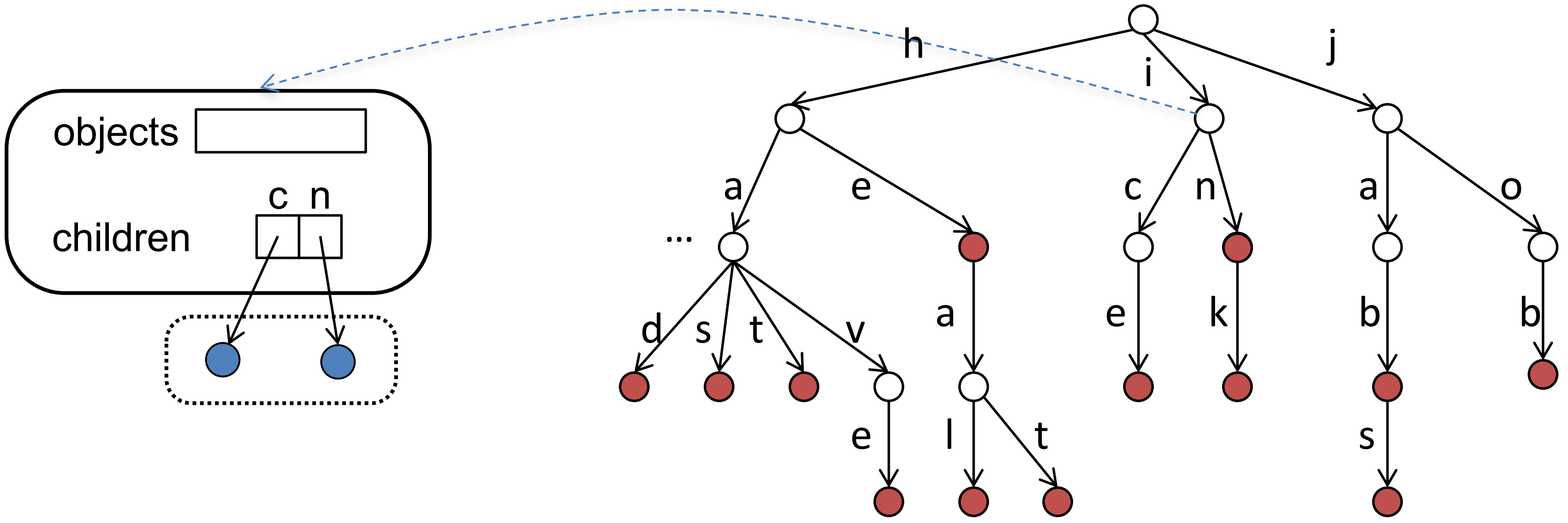
- Problem: given a query string, quickly find out **ALL** the road objects with the prefix of the road name matching the query string
- The efficiency of this operation depends on
  - How to store the road objects in the memory
  - How to scan them in the search
- Possible data structures
  - List of road objects
  - Hash map: name -> road object
  - What are their complexity?

```
  :
acton pl avondale
ada st remuera
adair pl weymouth
adam st greenlane
adam sunde pl glen eden
adams pl kamo
adams rd awarua
adams rd kaukapakapa
adams rd manurewa
adams rd thornton bay
:
```

# Trie

- A trie (prefix tree): an ordered tree data structure
- Each node contains
  - associated objects
  - a set of child nodes (each corresponding to a character)

```
Class TrieNode {
    List<Object> objects;
    HashMap<Character, TrieNode> children;
}
```



# Add and Get in a Trie

```
public void add(char[] word, Object obj) {
    Set node to the root of the trie;

    for (c : word) {
        if (node's children do not contain c)
            create a new child of node, connecting to node via c
        move node to the child corresponding to c;
    }

    add obj into node.objects;
}
```

```
public List<Object> get(char[] word) {
    Set node to the root of the trie;

    for (c : word) {
        if (node's children do not contain c)
            return null;
        move node to the child corresponding to c;
    }

    return node.objects;
}
```

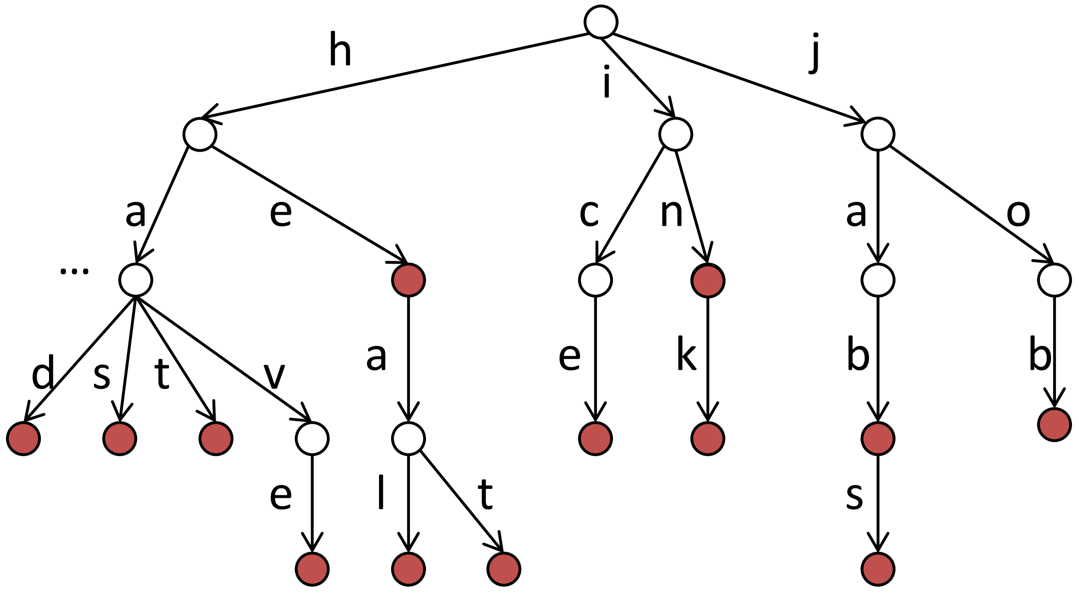
## Get All in a Trie

```
public List<Object> getAll(char[] prefix) {
    List<Object> results = new ArrayList<Object>();
    Set node to the root of the trie;

    for (c : prefix) {
        if (node's children do not contain c)
            return null;
        move node to the child corresponding to c;
    }

    getAllFrom(node, results);
    return results;
}
```

```
public void getAllFrom(TrieNode node, List<Object> results) {
    add node.objects into results;

    for (each child of node)
        getAllFrom(child, results);
}
```

# Example

- `add(“inch”, o1)`
- `get(“ink”)`
- `getAll(“he”)`



## Summary

- Display
  - Coordinate systems: static (kms) and dynamic (pixel)
  - Transform between different coordinate systems
  - Change display upon movement: change scale and origin
- Trie
  - Data structure
  - Add and get values
  - Get all values from a prefix