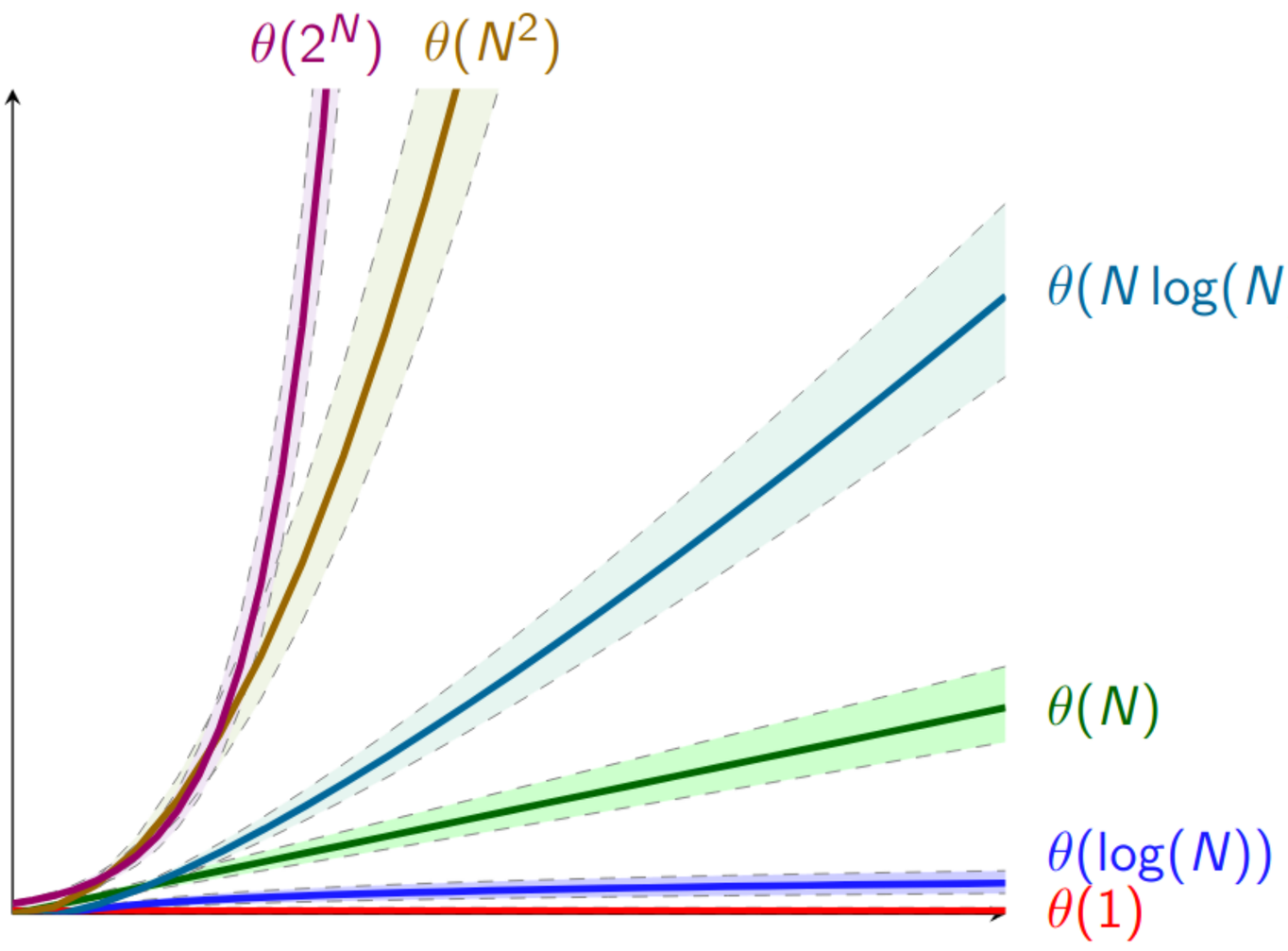
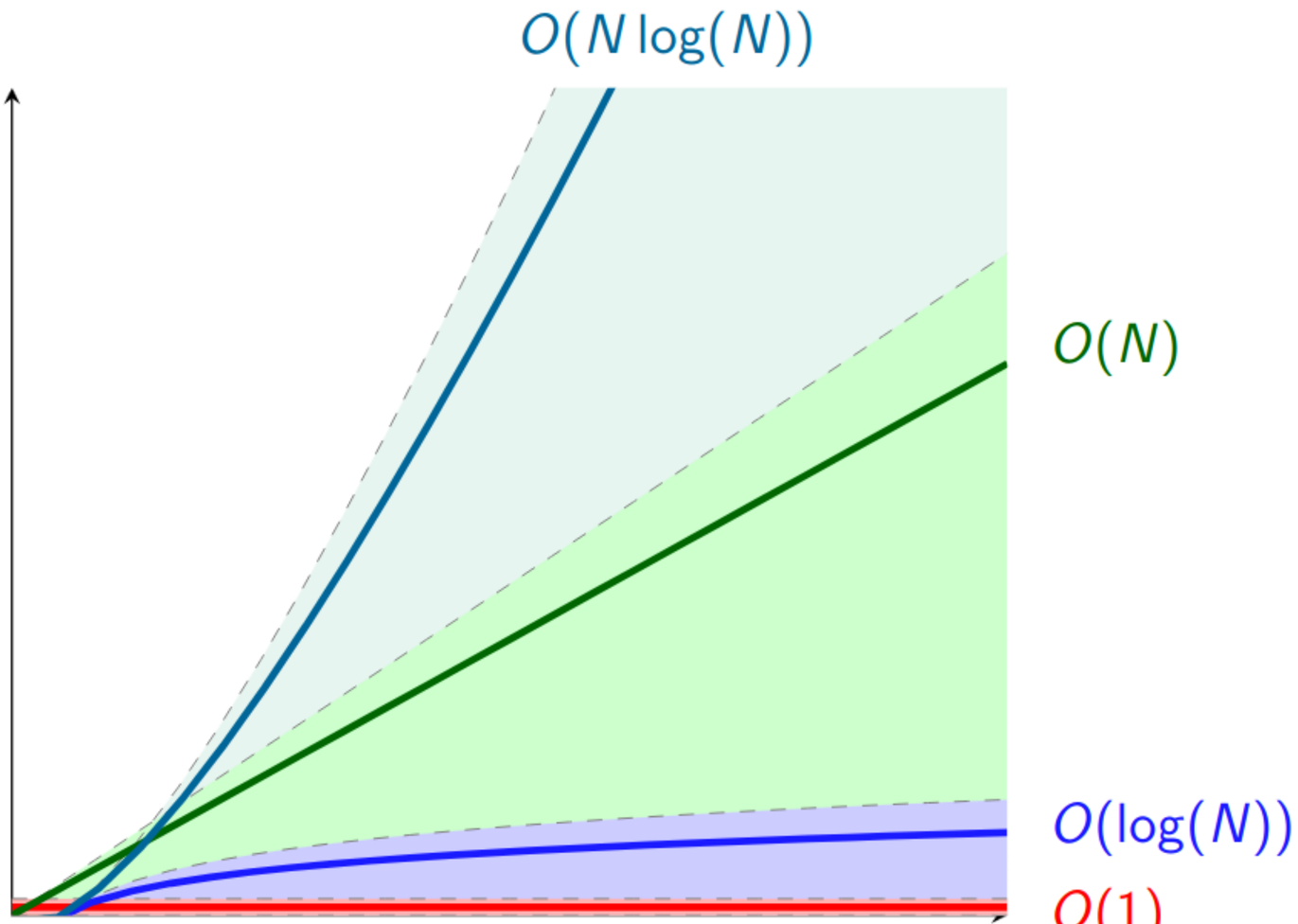
CSE 410 – Advanced Data Structures
Topic 03: Background

Oliver Kennedy

Complexity Classes



Big–O Bounds



Analyzing Code

File: **DFS.java**

```
1   public Map<Vertex,Vertex> traverse(Graph graph, Vertex start, TodoList todos)
2   {
3     Map<Vertex,Vertex> backEdges = new HashMap<>()
4     todos.add( new Edge(null, start) )
5     while(!todos.isEmpty())
6     {
7       Edge nextEdge = todos.remove()
8       if(!backEdges.contains(nextEdge.to))
9       {
10        backEdges.put(nextEdge.to, nextEdge.from)
11        for(edge : graph.getOutEdges(nextVertex))
12        {
13          todos.add(edge)
14        }
15      }
16    }
17  }
```


add, isEmpty, remove

A, E, R are parameters of TodoList

#E, #V are parameters of the graph

Amortized Bounds

File: **AmortizedQueue.java**

```
1  class AmortizedQueue<T>
2  {
3    SinglyLinkedList<T> head = new SinglyLinkedList();
4    SinglyLinkedList<T> tail = new SinglyLinkedList();
5
6    public void enqueue(T value)
7    {
8      tail.prepend(value)   — O(1)
9    }
10   public T dequeue()
11   {
12     if(head.isEmpty){                — O(N)
13       head = tail.reverse()
14       tail = new SinglyLinkedList()  O(1)
15     }
16     return head.removeFirst()  — O(1)
17   }
18 }
```

Dequeue is O(N) in general but...

usually O(1)

Each O(N) run "credits" us for N calls to dequeue

The total runtime for N calls to dequeue is Always O(N)

Amortized O(1)

Expectations

Unqualified Runtime:
Always guaranteed

Amortized Runtime
Guaranteed average over N calls
↗ individual calls may be slower

Expected Runtime
Statistically Likely

Collection ADTs

Collection, Elements Identified by Position

List (Sequence)

Collection, Elements Identified by Position in Sort Order

Sorted List

Collection, Elements Identified by Their Value

Set

Collection, Elements Identified by Key

Value

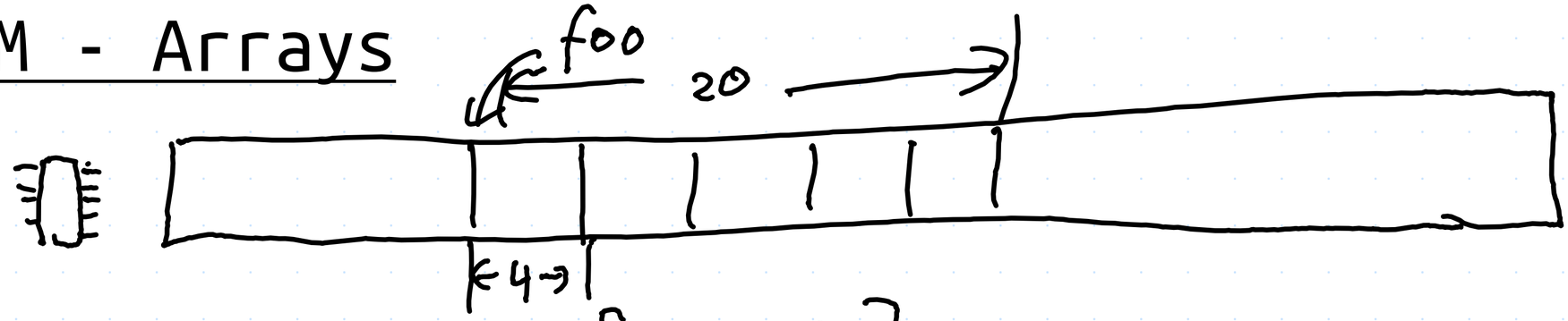
Data Structure Building Blocks

List
↳ Array, Linked list, Vector (aka Array List, Array Buffer)

Set
↳ Tree, Hash Table

Map
↳ Tree, Hash Table

RAM - Arrays

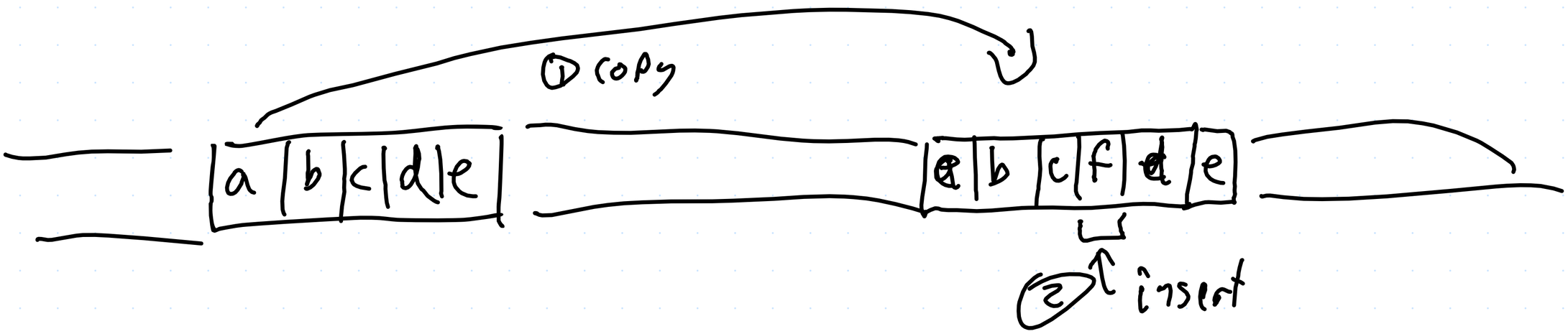


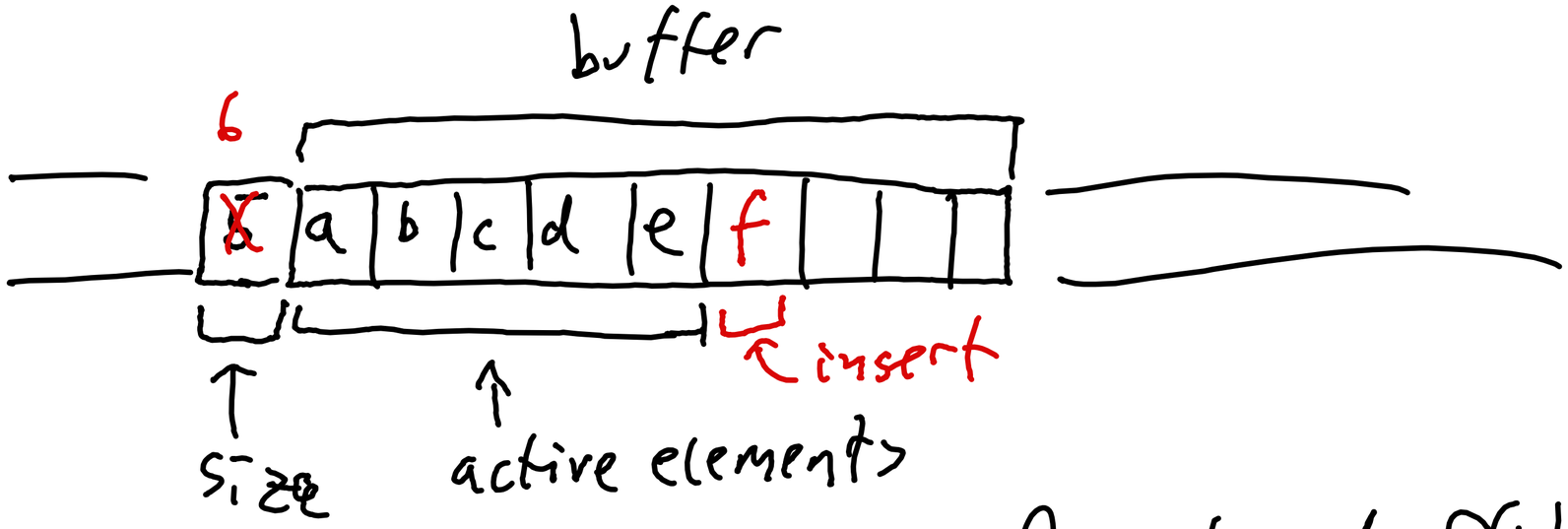
let foo : [u32; 5] = _____

4b = 32/8

foo[3] ← foo + 3 · 4

sizeof one element

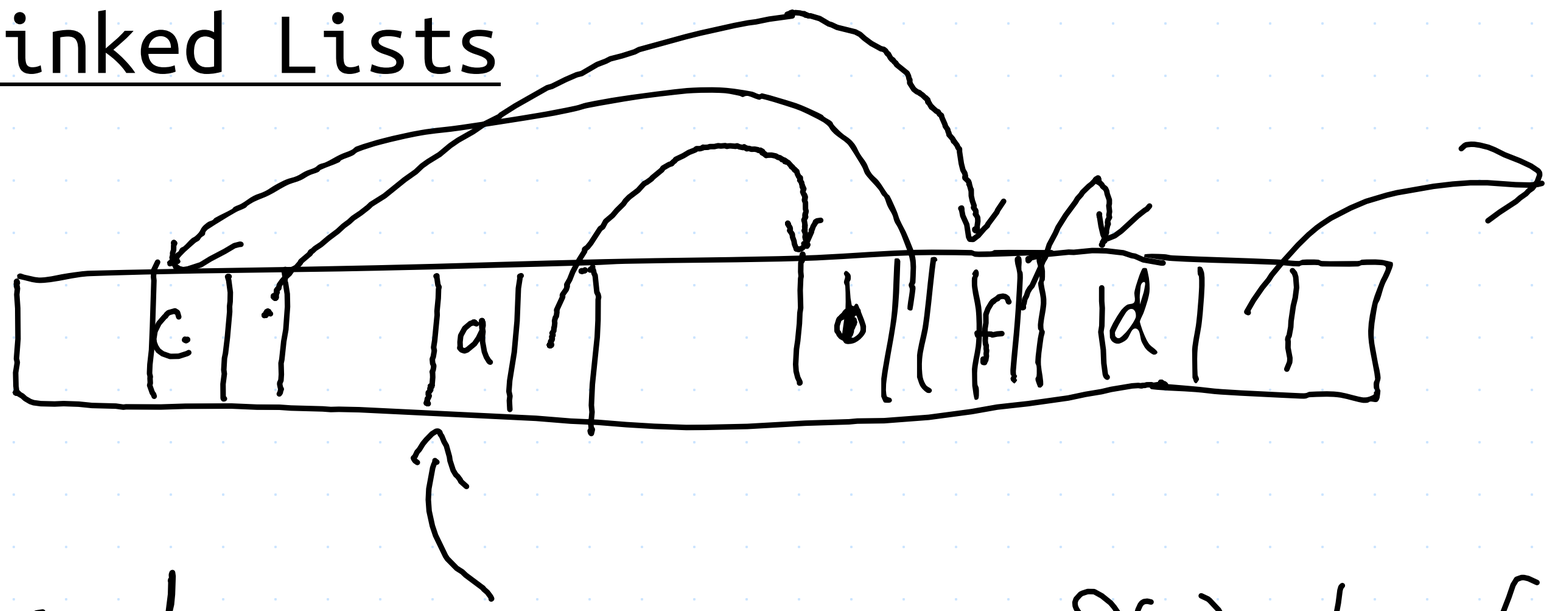




Amortized O(1) insert

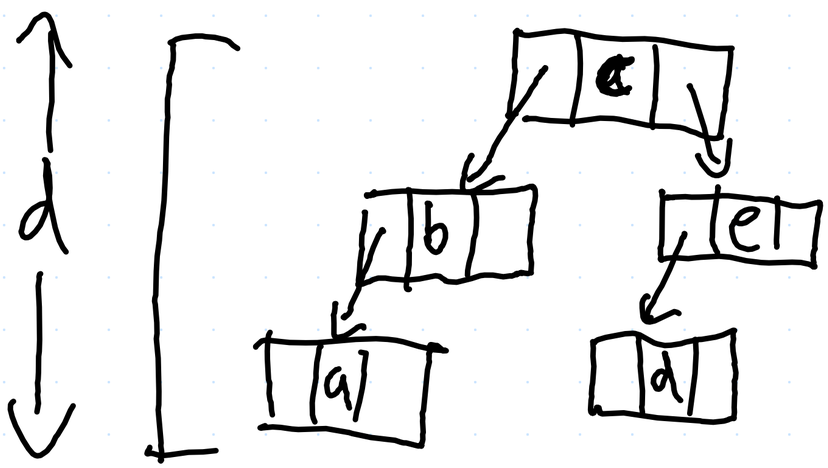
- ArrayBuffer (Scala)
- ArrayList (Java)
- Vector (Vec) (Rust)

RAM - Linked Lists



a, b, c, d, e

O(i) to find ith element

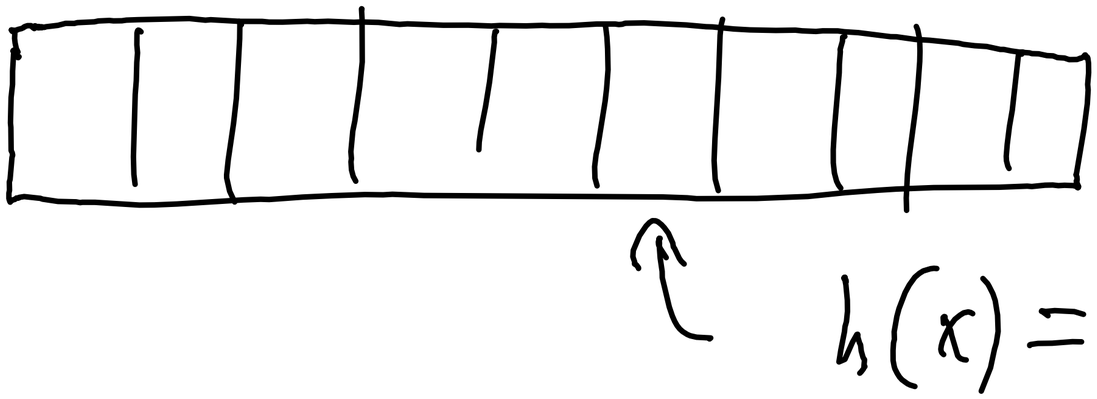


Binary search tree

$O(d) \leq O(N)$

Balanced Binary Search tree

↳ $d \simeq O(\log N)$



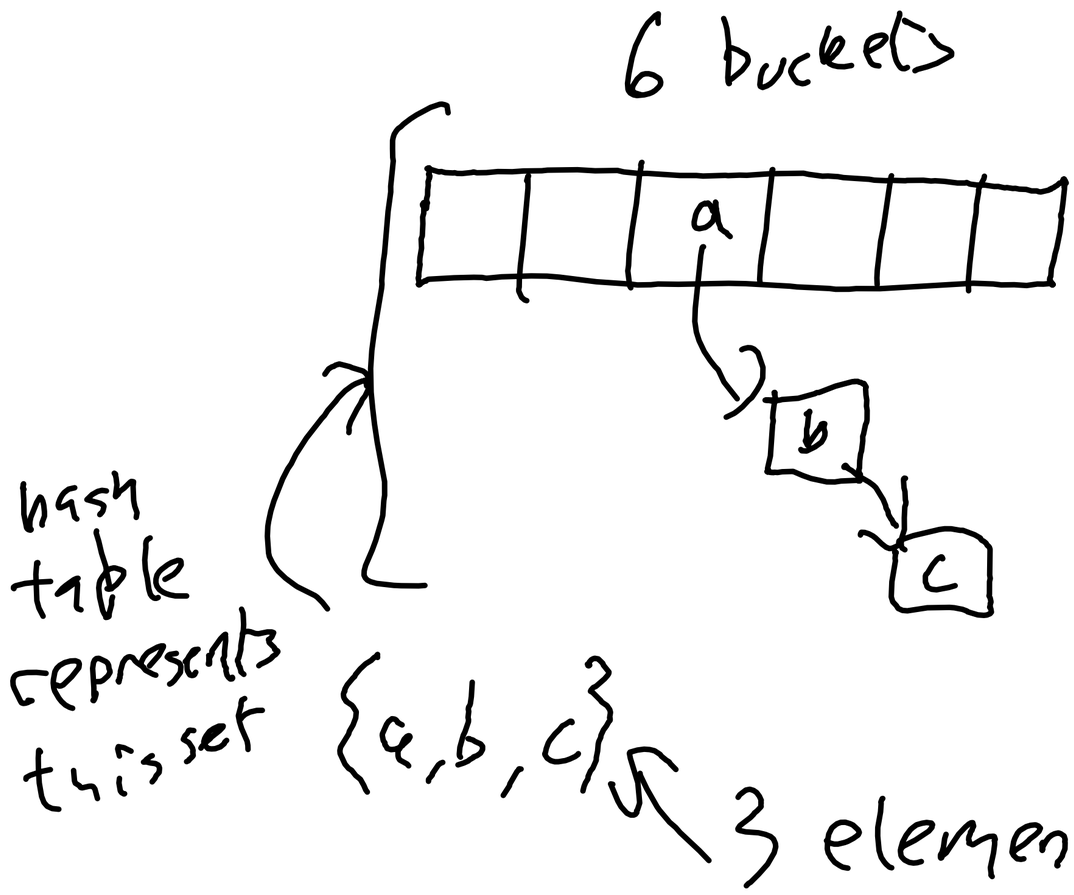
Hash function

↳ Produces a pseudorandom output

↳ Output is deterministic

$$h(42) = h(42)$$

$h(42) \% N \rightarrow$ position in an N element array where 42 is stored



$h(a) = h(b) = h(c)$

Hash Table w/ Chaining

↳ Expected $O(1)$

More Structure in the Data

Primitive: Int, Float, Char, (String) ← usually

Combine primitives as

↳ Struct / Tuple } - fixed set of fields
- Types are independent
- fields identified by name

↳ Collection
↳ List/Vector
↳ Set/Map

- variable number of elements
- All of the same type
- Identified by position (list)
  - by key (map)
- Properties
  ↳ Uniqueness → set

Collection of

Struct of

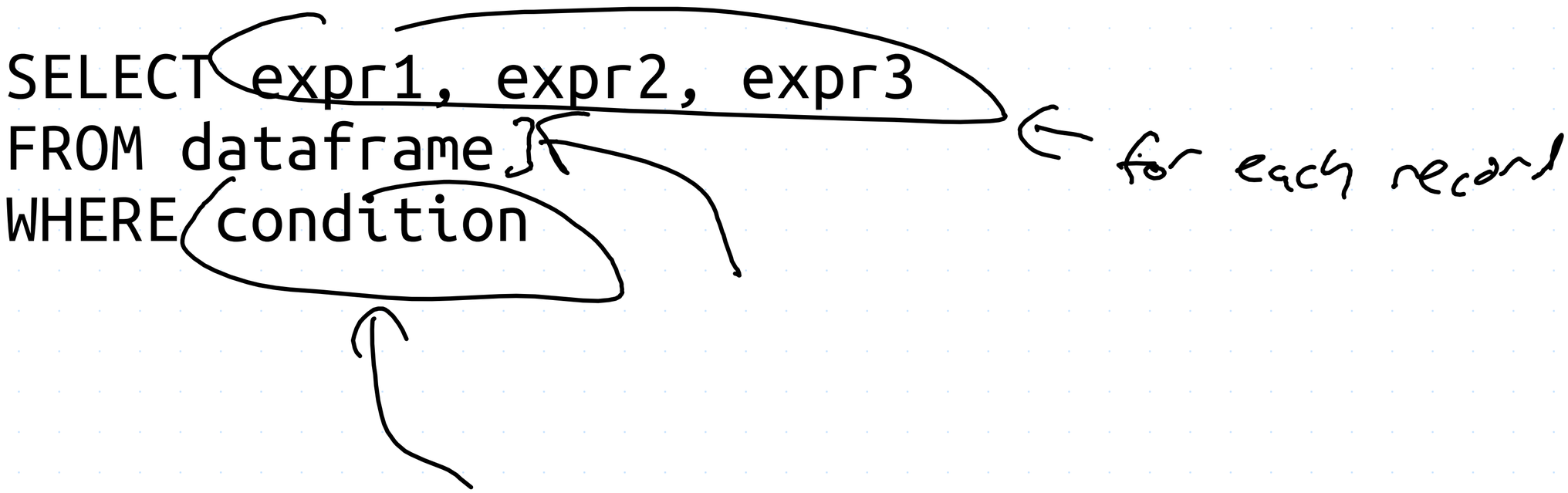
Primitive1, Prim2, ...

Table, Dataframe, Relation

CartId	Account
1	alice
2	bob
3	carol

CartId	Item
1	apple
1	banana

SQL


SELECT expr1, expr2, expr3
FROM dataframe
WHERE condition
for each record

$$O(1) + A + \sum_{V} \left[E + R + O(1) + \deg(v) + \sum_{\text{adjacent to } v} A \right]$$

$$= O(1) + A + \sum_{V} \left[E + R + O(1) + \deg(v) + A \cdot \deg(v) \right]$$

$$= O(1) + A + \sum_{V} \left[E + R + A \cdot \deg(v) \right]$$

$$= O(1) + A + \sum_{V} \left[E + R \right] + \sum_{V} \left[A \cdot \deg(v) \right]$$

$$= O(1) + A + \#V \left[E + R \right] + A \cdot \#E$$

$$= \#V \cdot (E + R) + A \cdot \#E$$

```
SELECT DISTINCT expr1, expr2, expr3
FROM dataframe
WHERE condition
```

Traversal Algo	Todo List	E	R	A	Traversal Runtime
BFS	Queue	$O(1)$	$O(1)$	$O(1)$	$\#V+\#E$
DFS	Stack	$O(1)$	$O(1)$	$O(1)$	$\#V+\#E$
Dijkstra's	Priority Queue	$O(1)$	$O(\log(\#E))$	$O(\log(\#E))$	$(\#V+\#E)\log(\#E)$

```
SELECT expr1, SUM(expr2)
FROM dataframe
WHERE condition
GROUP BY expr1
```