Concurrent disjoint Set Union

Jayanti, Tarjan '21 Distributed Computing

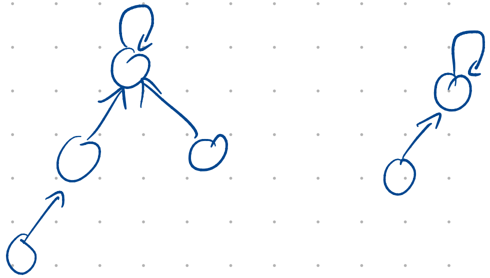
union-find: initially single-element sets

find(x): returns x' representative

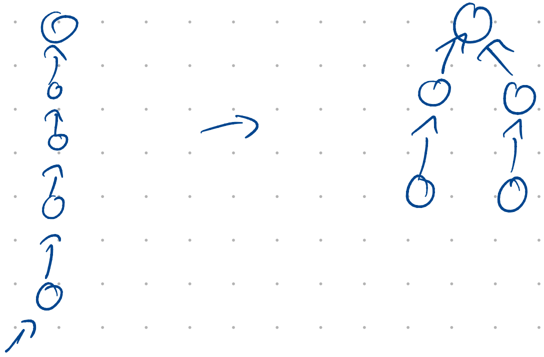
union(x,y): merges sets x,y and assigns a new representative to the set, if find(x) ≠ find(y)

sequential solutions: n elements, m operations $O(m \cdot \alpha(n, \frac{m}{n}))$ time
$O(\log n)$ per operation

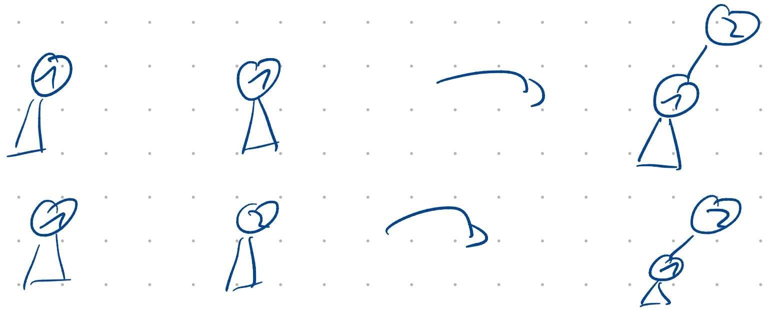
forest, one rooted tree per set



- find includes compaction
  - e.g. splitting:
    u.p = u.p.p.



- union(x,y) link(find(x), find(y)) with "clever" link.
  - eg. link / union by rank.



Goal: Implement concurrent solution for union-find on an asynchronous parallel random-access machine APRAM with p processes

APRAM: • processes are completely asynchronous
• each process has private memory
• shared memory with concurrent reads

• compare-and-swap:

```
CAS(x, y, z): < if *x = y
                    *x = z
                    return true
                else:
                    return false >
  address:
```

```
• DCAS(x, y, z, a, b, c): < if *x = y && *a = b:
                                return true
                          >
```

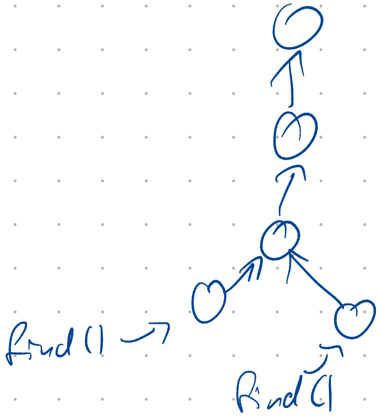
$\underbrace{x, y, z}$, $\underbrace{a, b, c}$

Idea: Use forest with splitting and union by rank

$O(m \cdot (\alpha(n, \frac{m}{np}) + \log(\frac{np}{m} + 1))$ time altogether

$O(\log n)$ time per operation

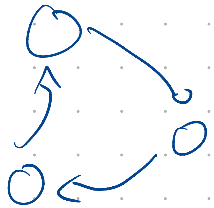
issue: interference!



p_1 union(a, b)

p_2 union (b, c)

p_3 union (c, a)

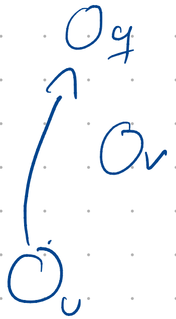


```
union(x, y):  u = find(x)
              v = find(y)
              while u ≠ v
                  link(u, v)
                  u = find(u)
                  v = find(v)
```

interference: · link() fails
· find() doesn't return a root

find with one-try-splitting:

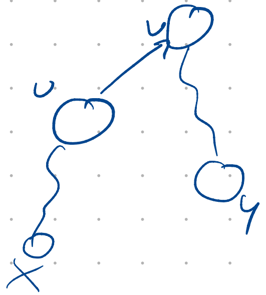
```
find(x):  u = x
          v = u.p
          w = v.p

          while ≠ w:
              CAS(u.p, v, w)
              u = v                 }
              v = u.p               }  distribute charge to
              w = v.p               }       other processes
        return v
```



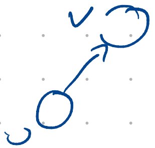
find with two-try-splitting :

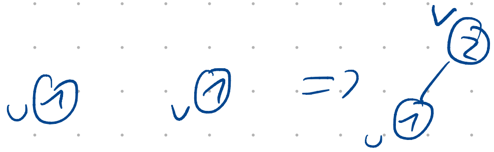
```
find(x) :   u = x
            v . u.p
            w = v.p

            while v ≠ w :
                CAS(u.p, v, w)
                v = u.p          }  u.p
                w = v.p
                CAS(u.p, v, w)
                u = v
                v = u.p
                w = v.p
          return v
```

```
same-set(x, y):

    u = find(x)
    v = find(y)

    while u ≠ v :
        w = u.p
        if u = w :
            return false
        u = find(u)
        v = find(v)
    return true
```



```
link(u,v) :
    r = u.r
    s = v.r

    if r < s :
        CAS( (u.p, u.r) , (u, r) , (v, r))
    else s > r
        CAS( ....)
    else :
        elink(u,v,r)
```

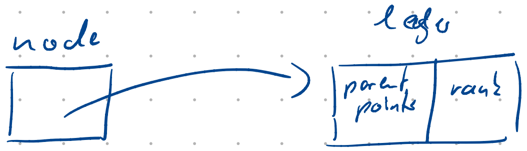




```
elink(u,v,r) :
    DCAS( (u.p, u.r) , (u, r)  (v, r)
          (v.p, v.r) , (v, r) , (v, r+1))
```

$\rightsquigarrow$ $O(\log n)$ height

fixing the cheating:



allocation of ledgers can be cumbersome