# Disentanglement in Nested-Parallel Programs

**Sam Westrick**
Carnegie Mellon University

Joint work with
Rohan Yadav, Umut Acar, and Matthew Fluet

## Parallel Programming

**imperative**

mutability
manual memory management
non-determinism

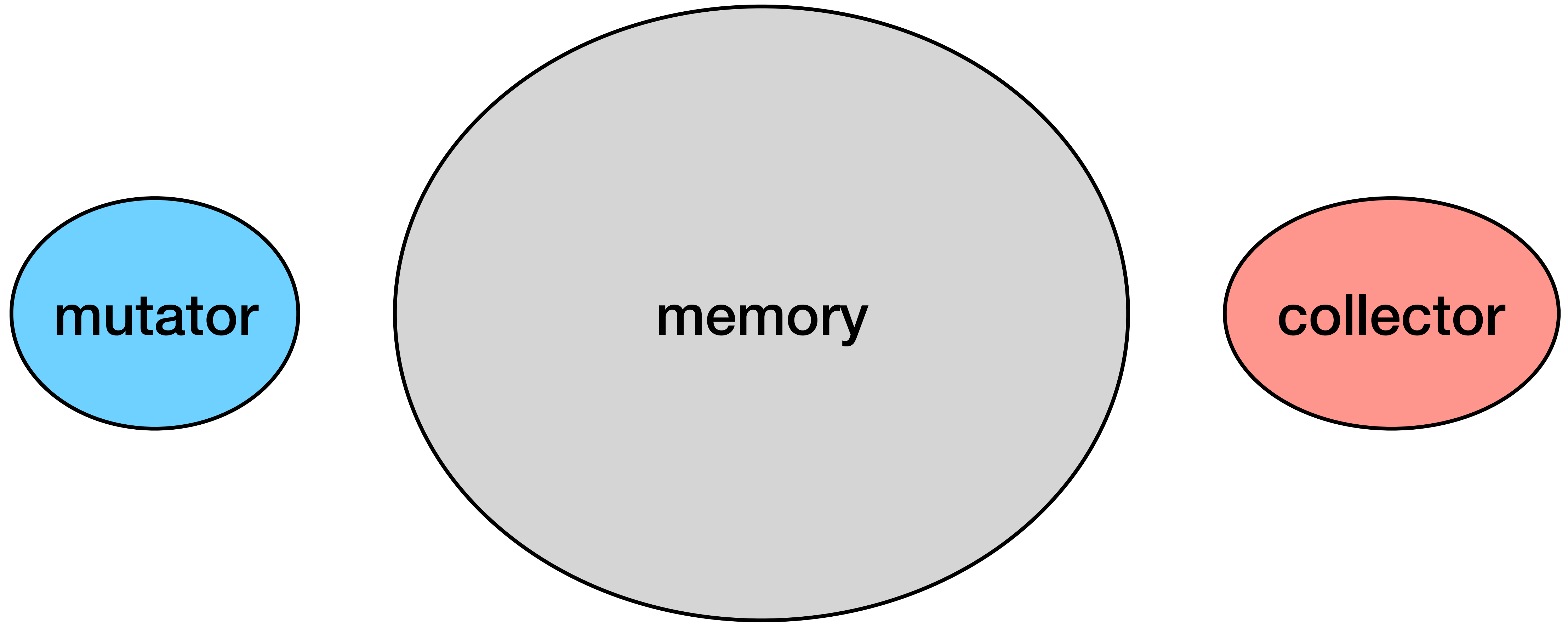
immutability
automatic memory management
determinism

**functional**

**fast**

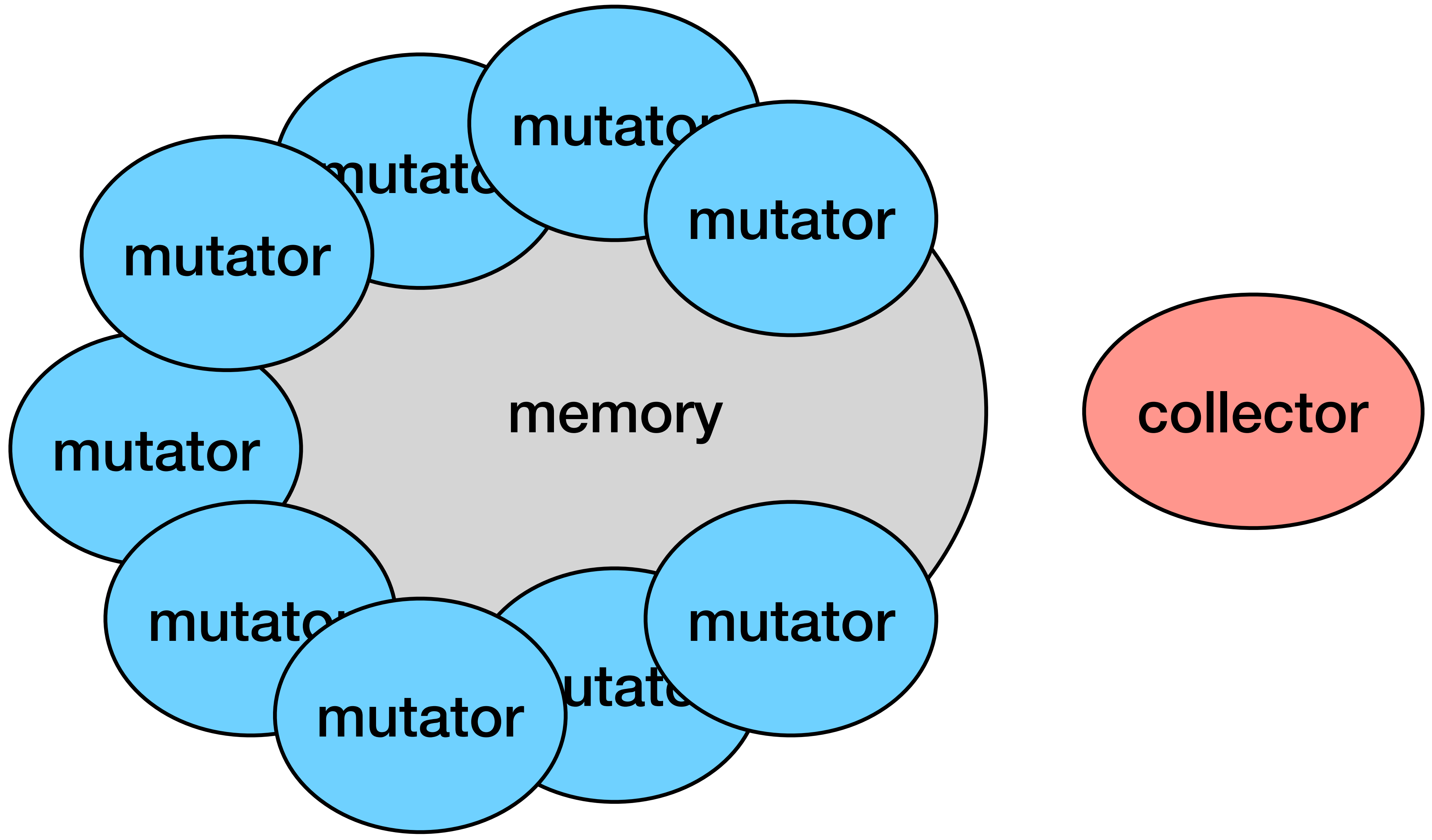
**can parallel functional programming be fast and scalable ?**

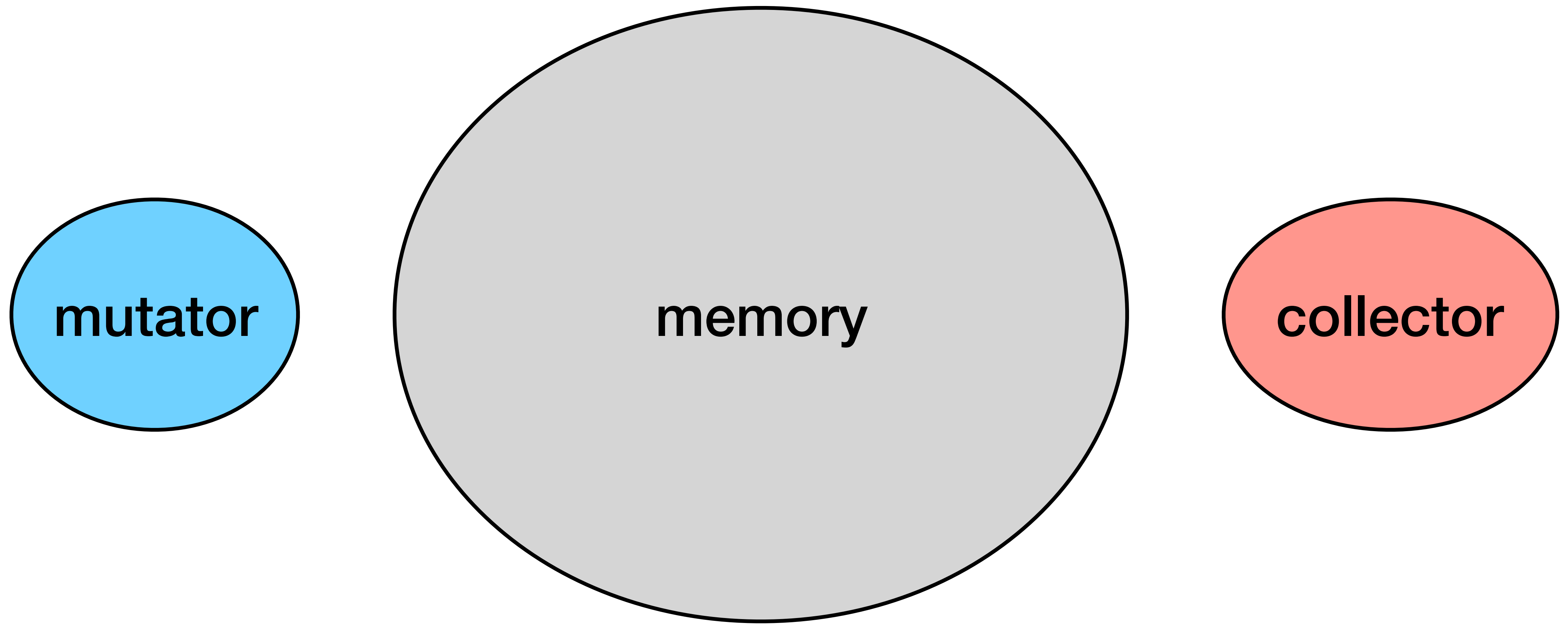
**slow?**



**Sequential**

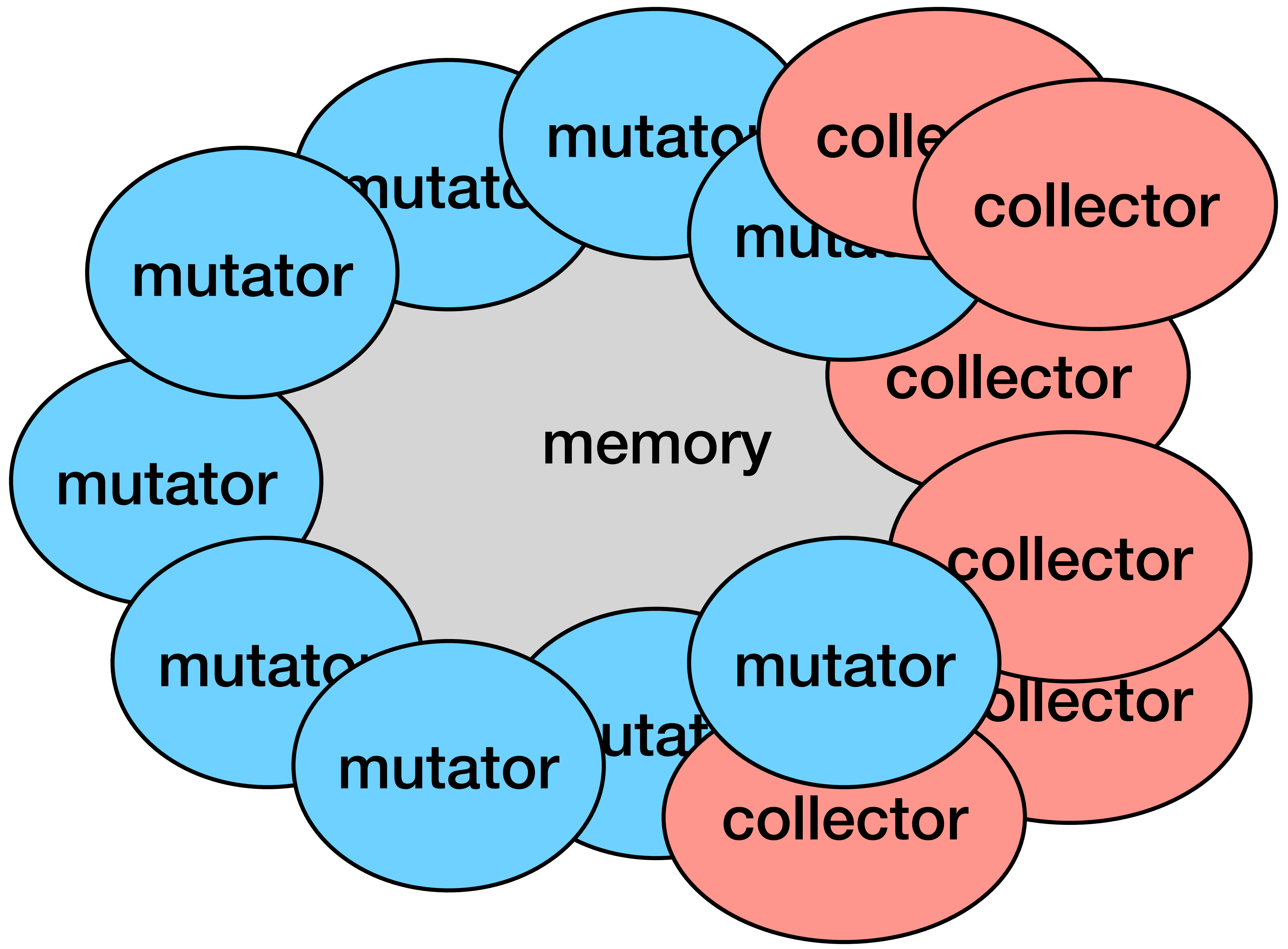
**Parallel**





Sequential

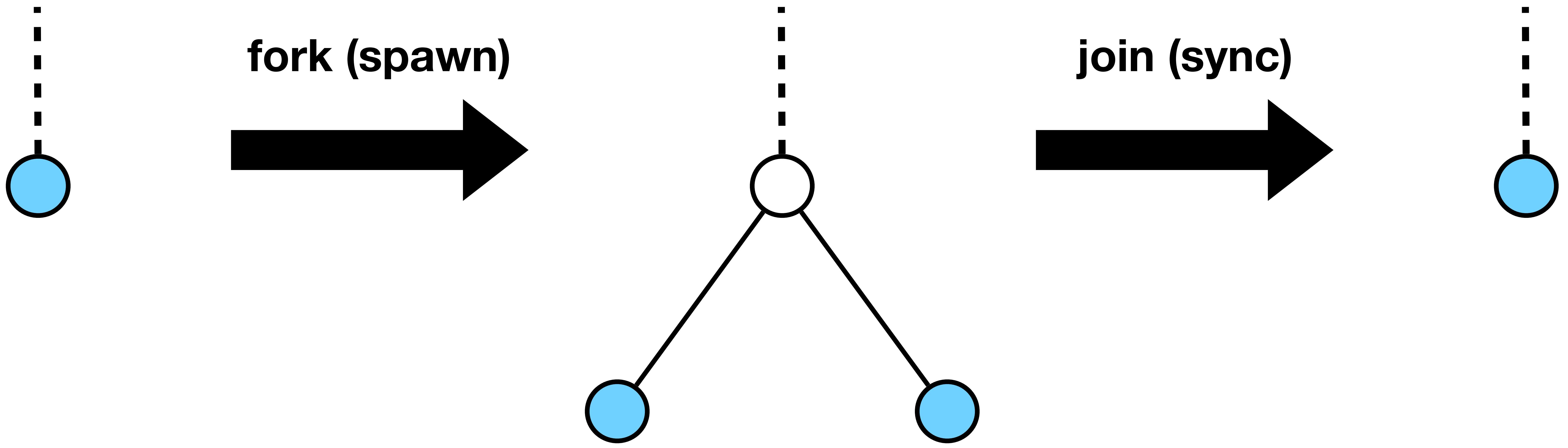
Parallel



**Is there a better way?**

## Nested Parallelism (Fork-Join)

- classic and popular
- MultiLisp, OpenMP, Cilk, Intel TBB, TPL (.NET), Rayon (Rust), Java Fork/Join, Habanero Java, X10, NESL, parallel Haskell, Futhark, Manticore, parallel ML, …

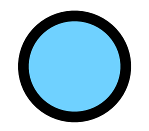


# Nested Parallelism (Fork-Join)

```
map f A =
  let
    B = newArray (length A)

    map' i j =
      case j-i of
      | 0 => ()
      | 1 => B[i] := f (A[i])
      | n =>
          let m = i + n/2
          in (map' i m || map' m j);
             ()
          end
  in
    map' 0 (length A);
    B
  end
```

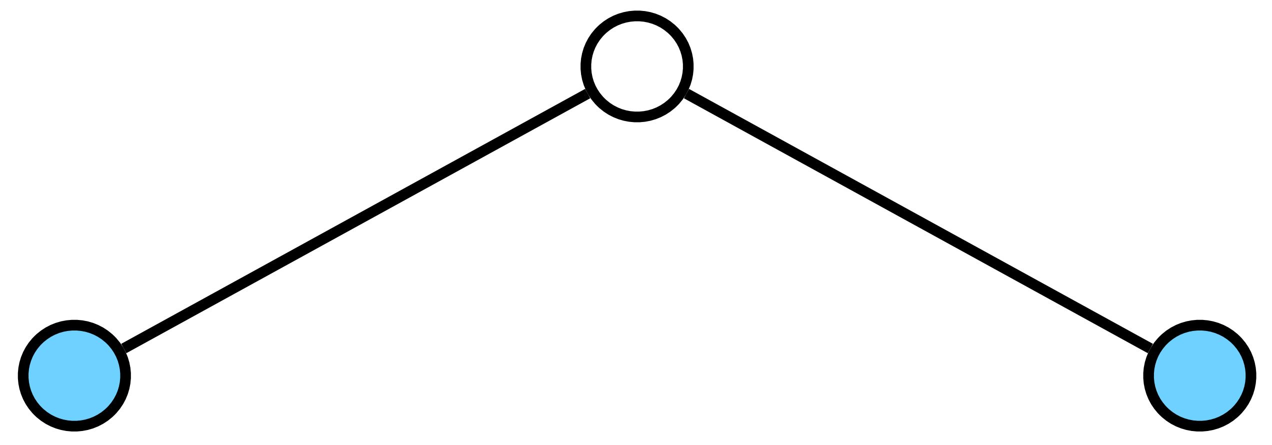
# Nested Parallelism (Fork-Join)



```
map f A =
  let
    B = newArray (length A)

    map' i j =
      case j-i of
      | 0 => ()
      | 1 => B[i] := f (A[i])
      | n =>
          let m = i + n/2
          in (map' i m || map' m j);
             ()
          end
  in
    map' 0 (length A);
    B
  end
```

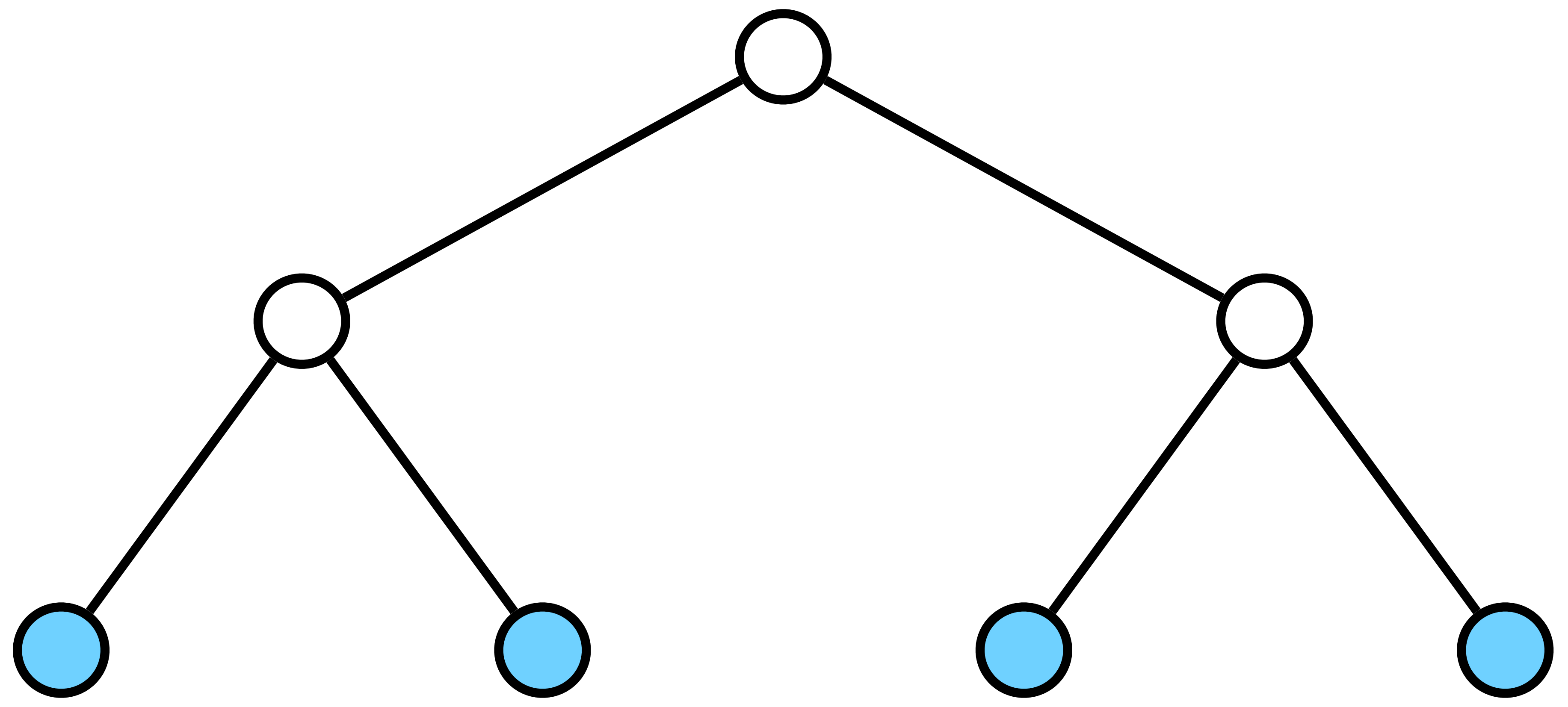
# Nested Parallelism (Fork-Join)



```
map f A =
  let
    B = newArray (length A)

    map' i j =
      case j-i of
      | 0 => ()
      | 1 => B[i] := f (A[i])
      | n =>
          let m = i + n/2
          in (map' i m || map' m j);
             ()
          end
  in
    map' 0 (length A);
    B
  end
```

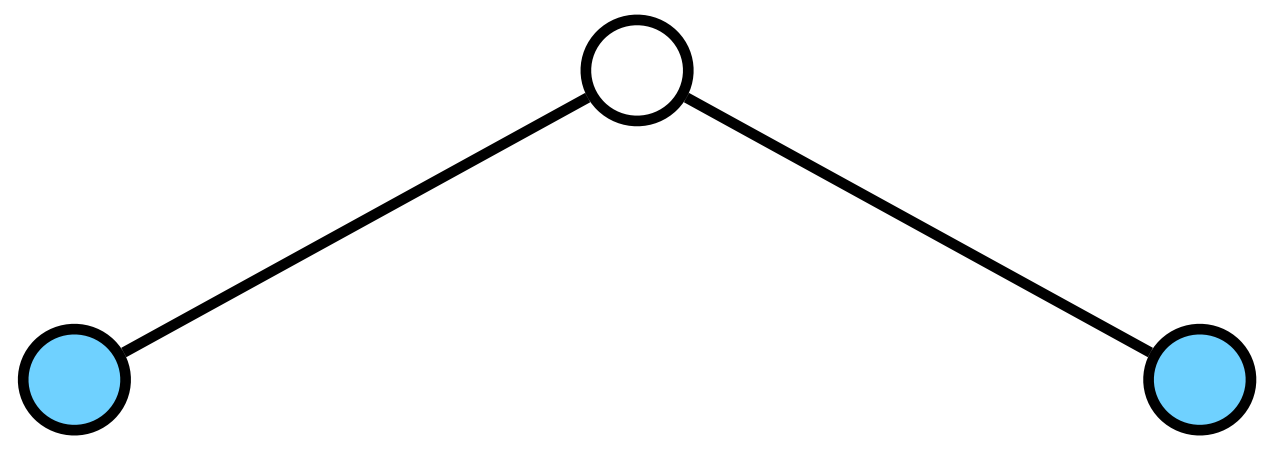
## Nested Parallelism (Fork-Join)



```
map f A =
  let
    B = newArray (length A)

    map' i j =
      case j-i of
      | 0 => ()
      | 1 => B[i] := f (A[i])
      | n =>
          let m = i + n/2
          in (map' i m || map' m j);
             ()
          end
  in
    map' 0 (length A);
    B
  end
```

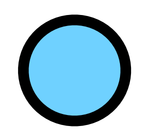
# Nested Parallelism (Fork-Join)



```
map f A =
  let
    B = newArray (length A)

    map' i j =
      case j-i of
      | 0 => ()
      | 1 => B[i] := f (A[i])
      | n =>
          let m = i + n/2
          in (map' i m || map' m j);
             ()
          end
  in
    map' 0 (length A);
    B
  end
```

## Nested Parallelism (Fork-Join)

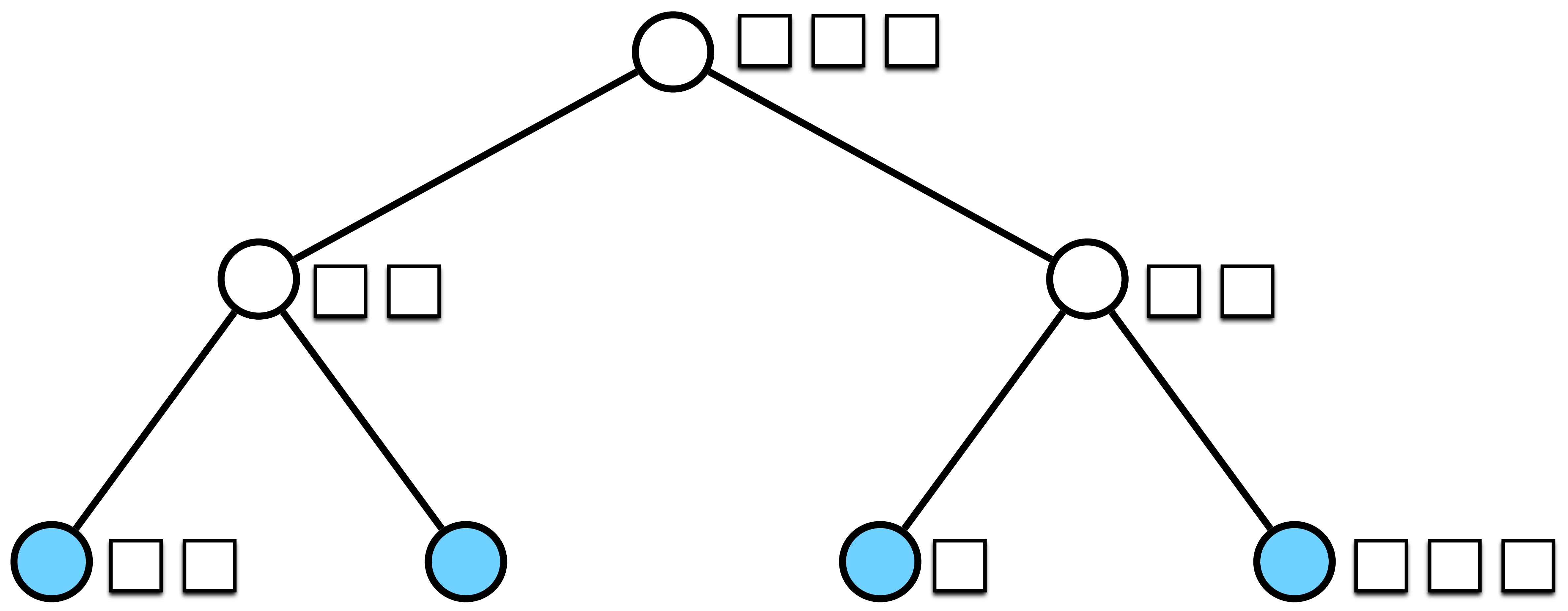


```
map f A =
  let
    B = newArray (length A)

    map’ i j =
      case j-i of
      | 0 => ()
      | 1 => B[i] := f (A[i])
      | n =>
          let m = i + n/2
          in (map’ i m || map’ m j);
             ()
          end
  in
    map’ 0 (length A);
    B
  end
```

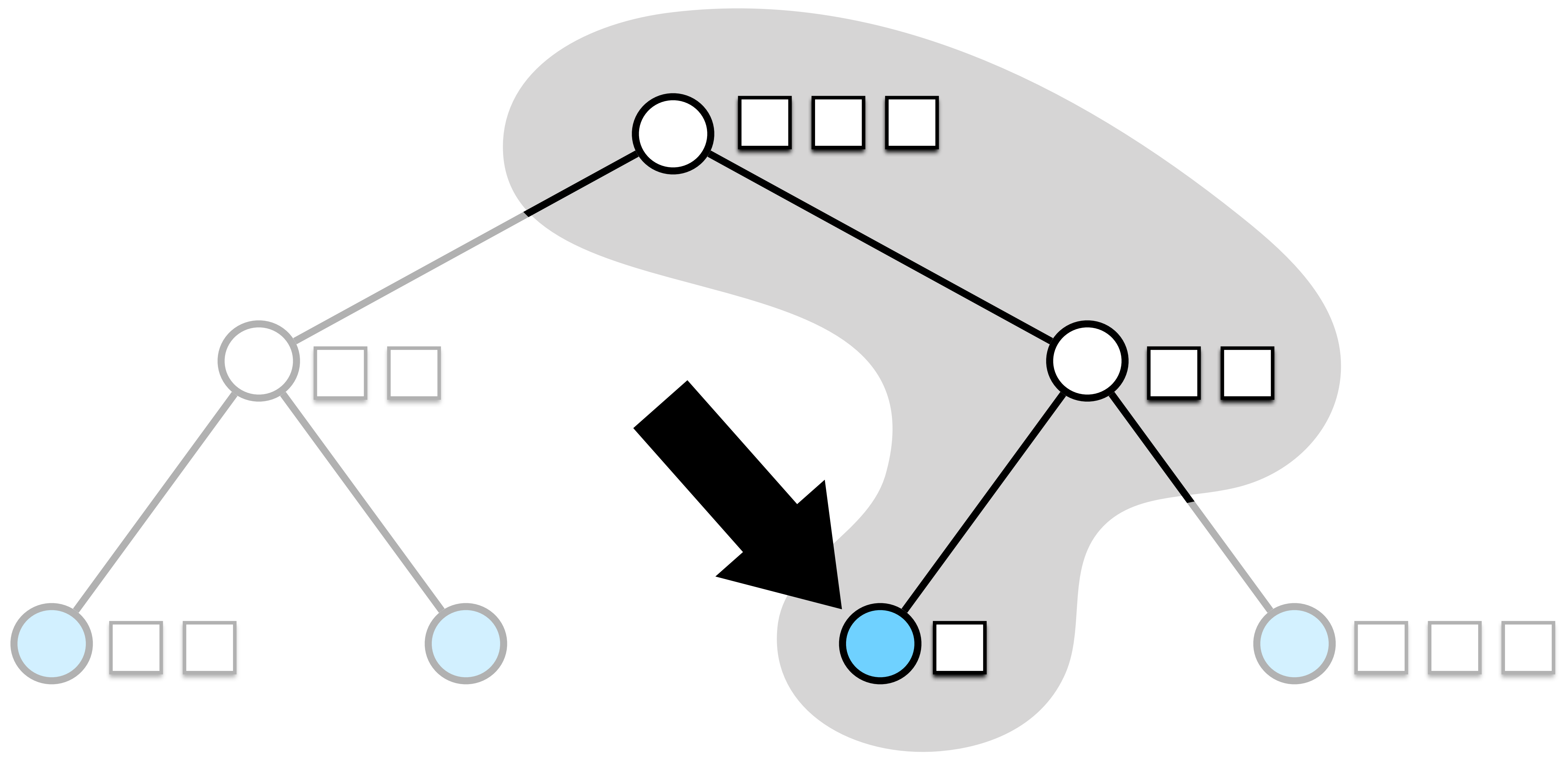
## Disentanglement

**definition**
throughout execution, each thread may only use data allocated by itself or **ancestors**



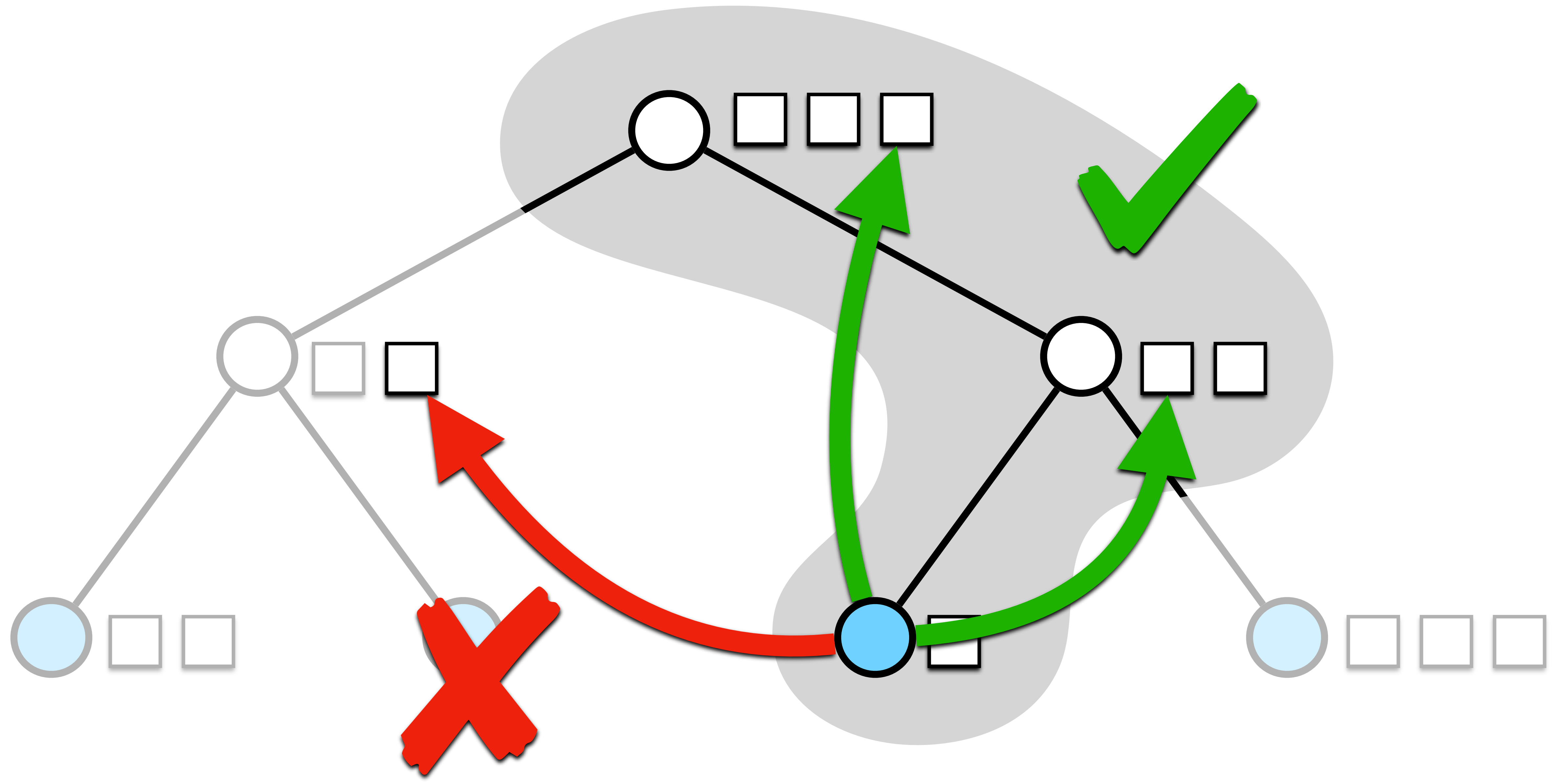
## Disentanglement

**definition**
throughout execution, each thread may only use data allocated by itself or **ancestors**



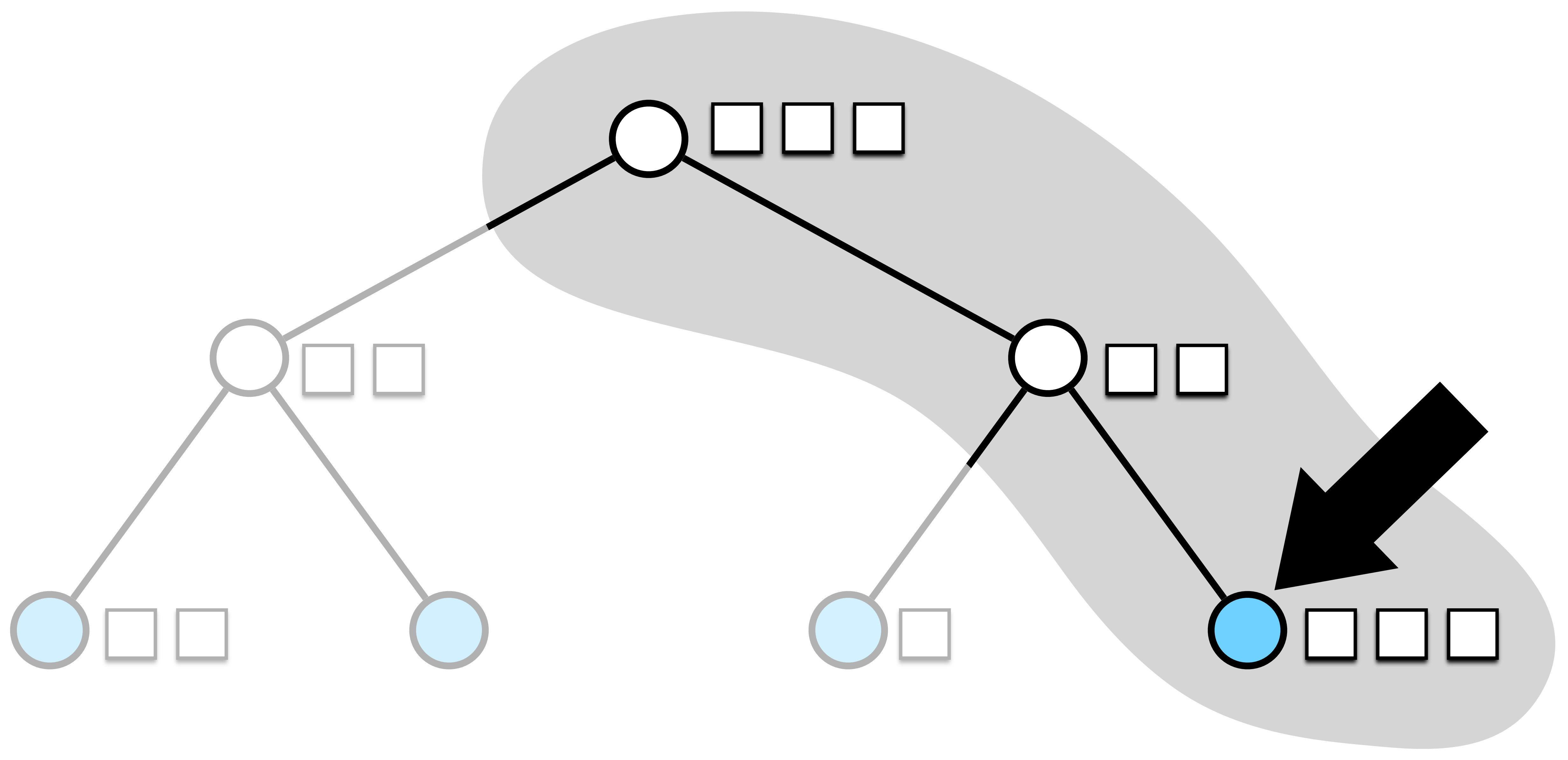
# Disentanglement

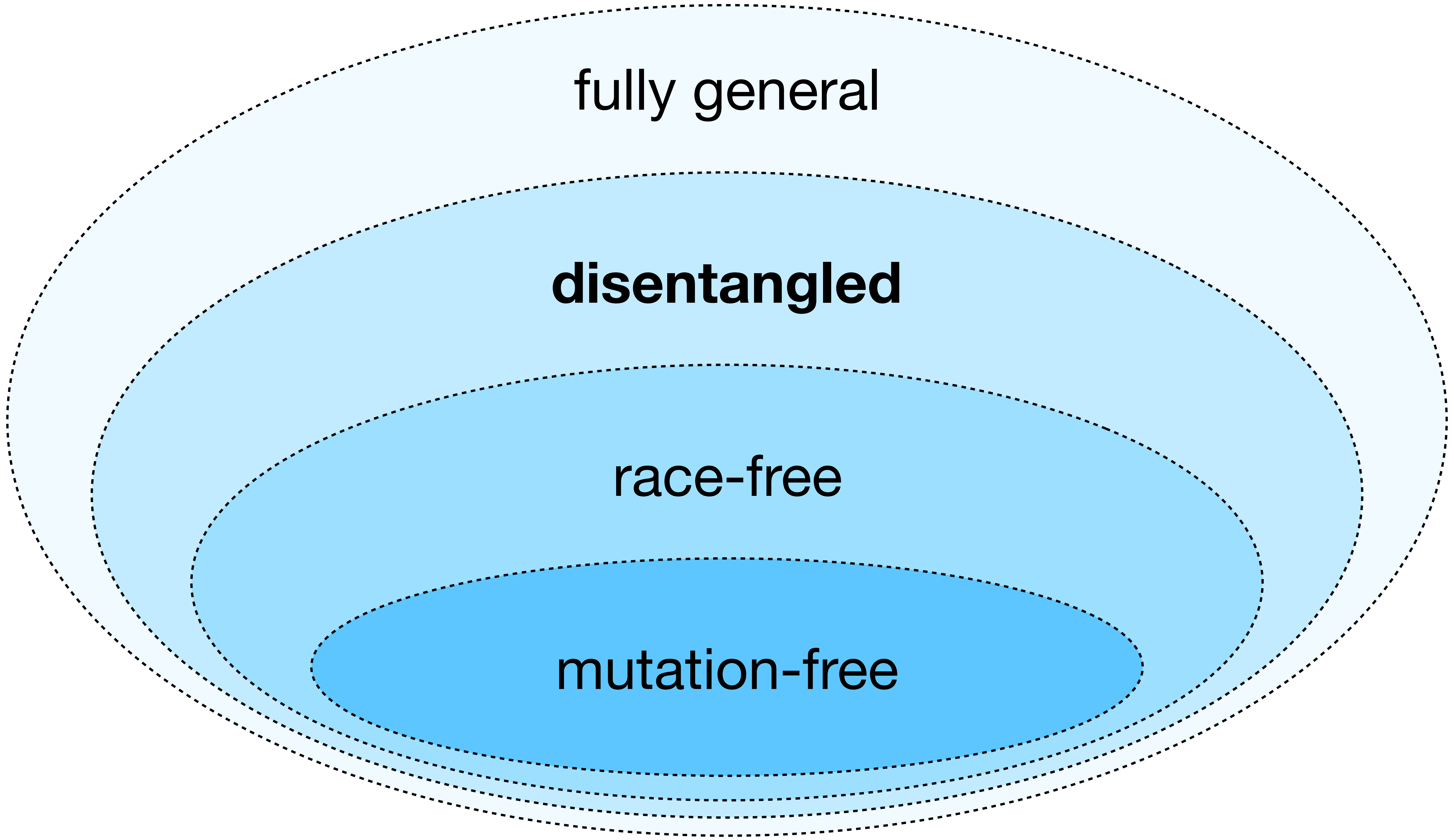
**definition**
throughout execution, each thread may only use data allocated by itself or **ancestors**



## Disentanglement

**definition**
throughout execution, each thread may only use data allocated by itself or **ancestors**




fully general
**disentangled**
race-free
mutation-free

**theorem**
all race-free programs are disentangled

Proof technique:

- use computation graphs for definitions
- identify single-step invariant:
    - if location *X* accessible without a race, then *neighbors(X)* are in root-to-leaf path
- carry invariant through race-free execution

# Disentanglement in Practice

Ligra

- BFS
- betweenness centrality
- Bellman-Ford
- *k*-Core
- Page Rank
- maximal independent set
- eccentricity estimation

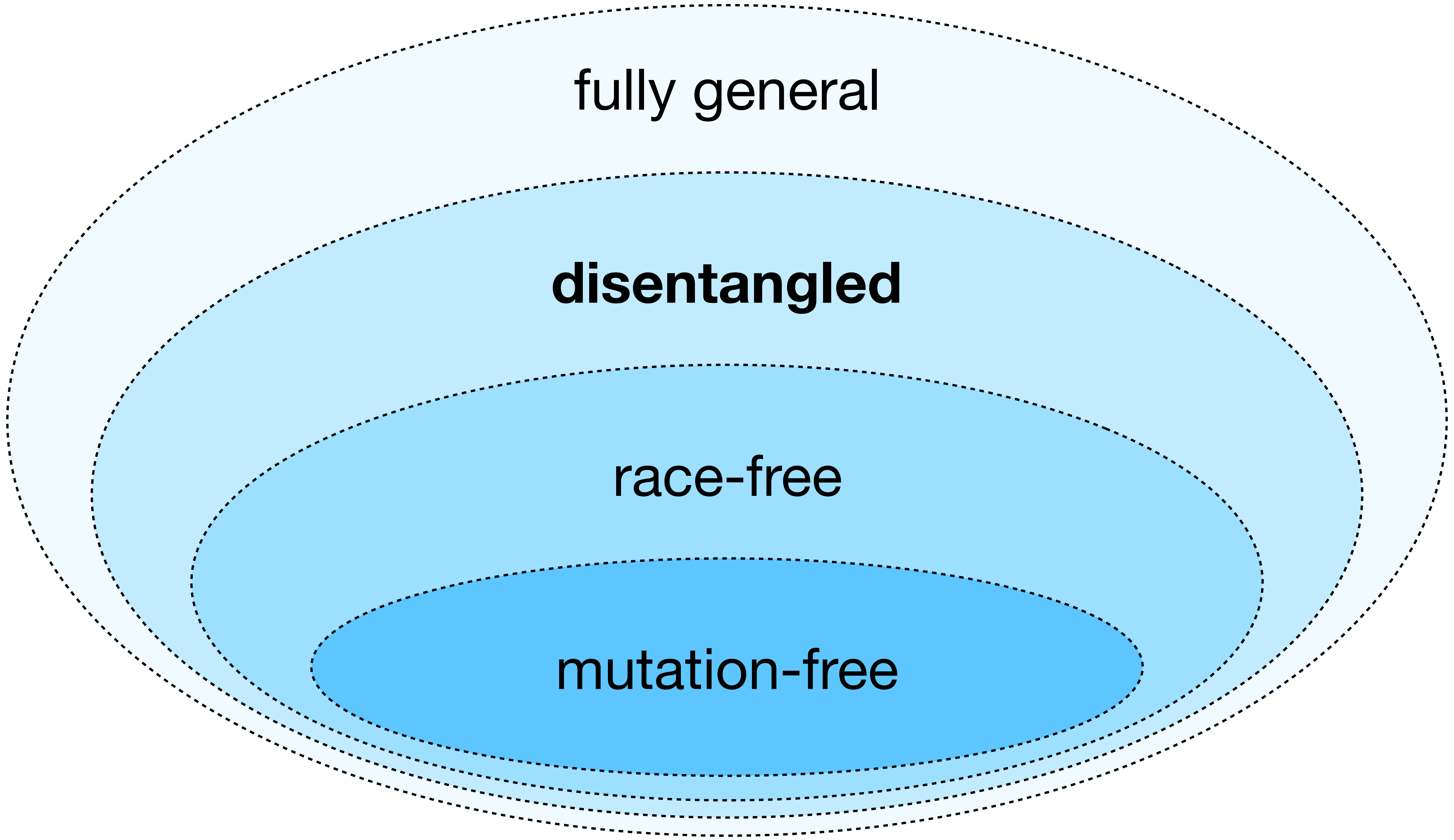
PBBS

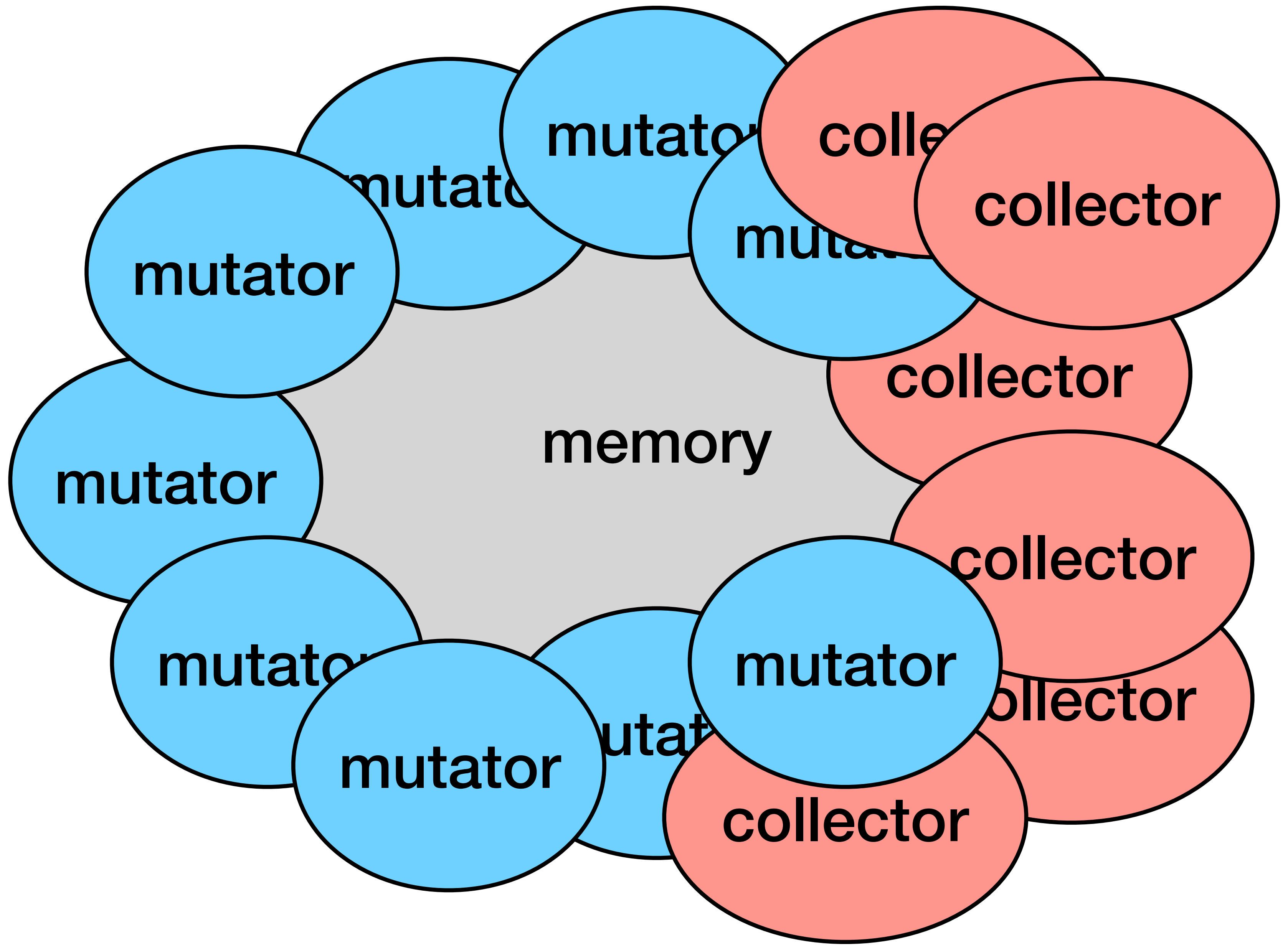
- quickhull
- deduplication
- sorting
- minimum spanning forest
- suffix array
- Barnes-Hut
- nearest neighbors
- ray casting

**all disentangled**

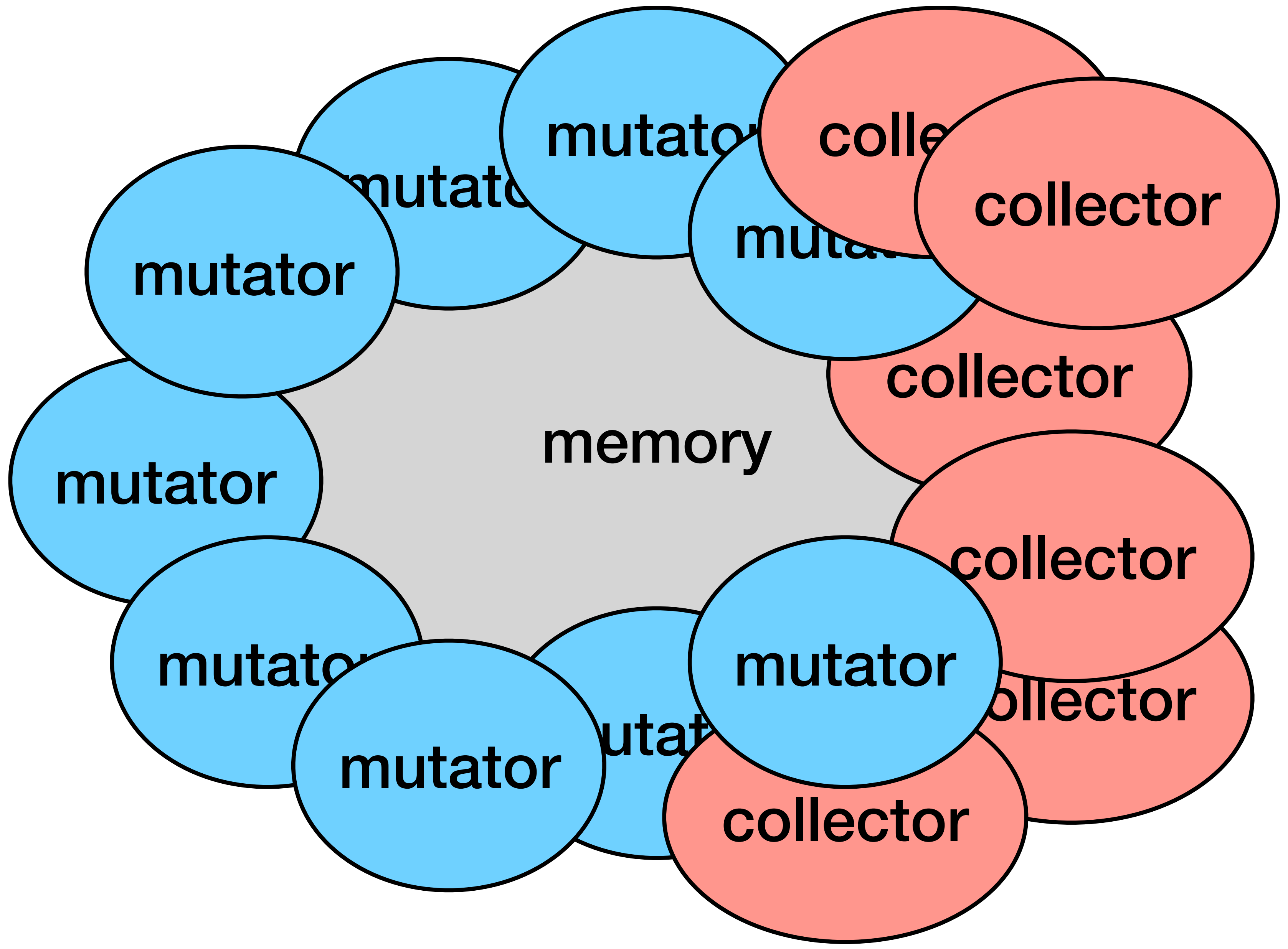
(and likely others too)

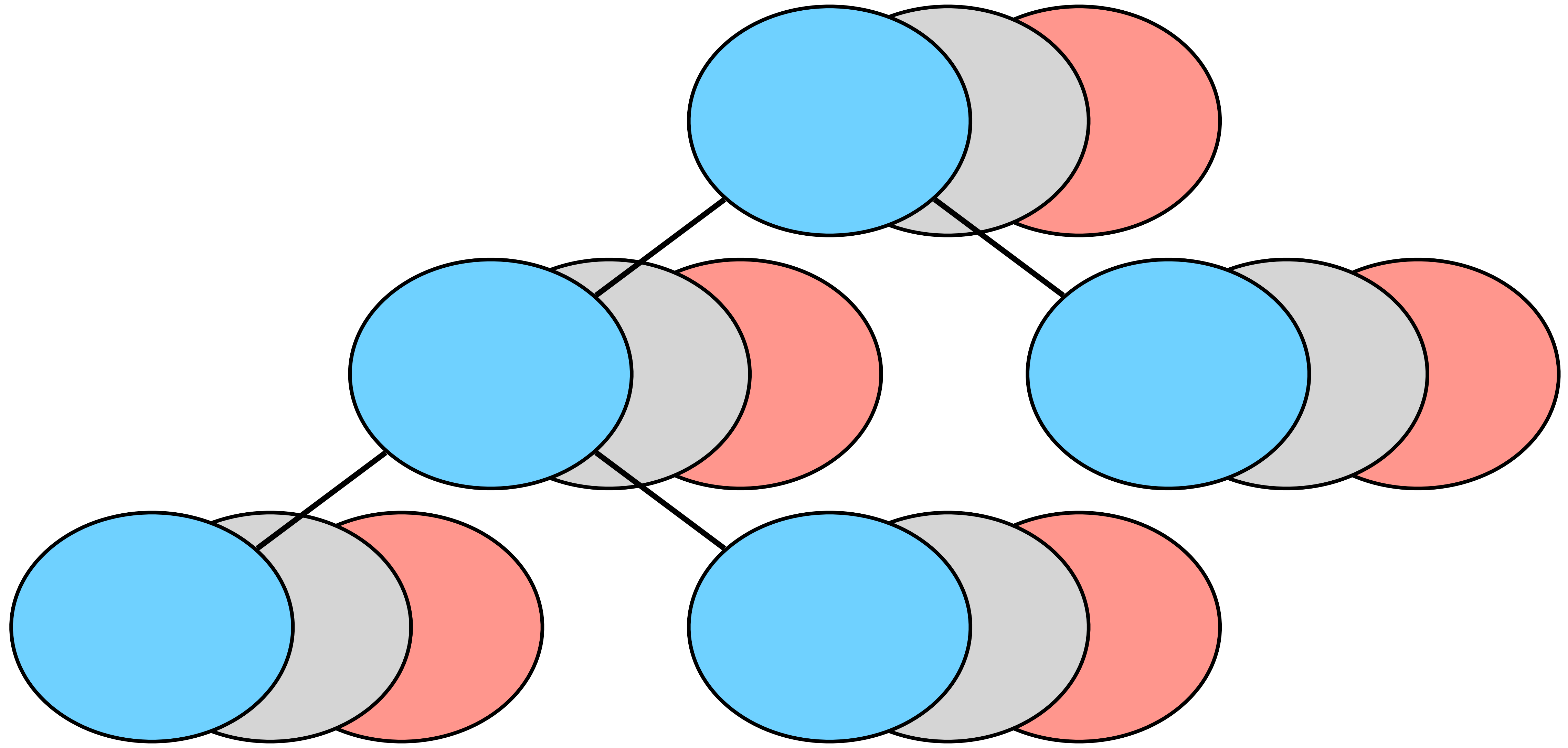
**many benign data races**


fully general
disentangled
race-free
mutation-free

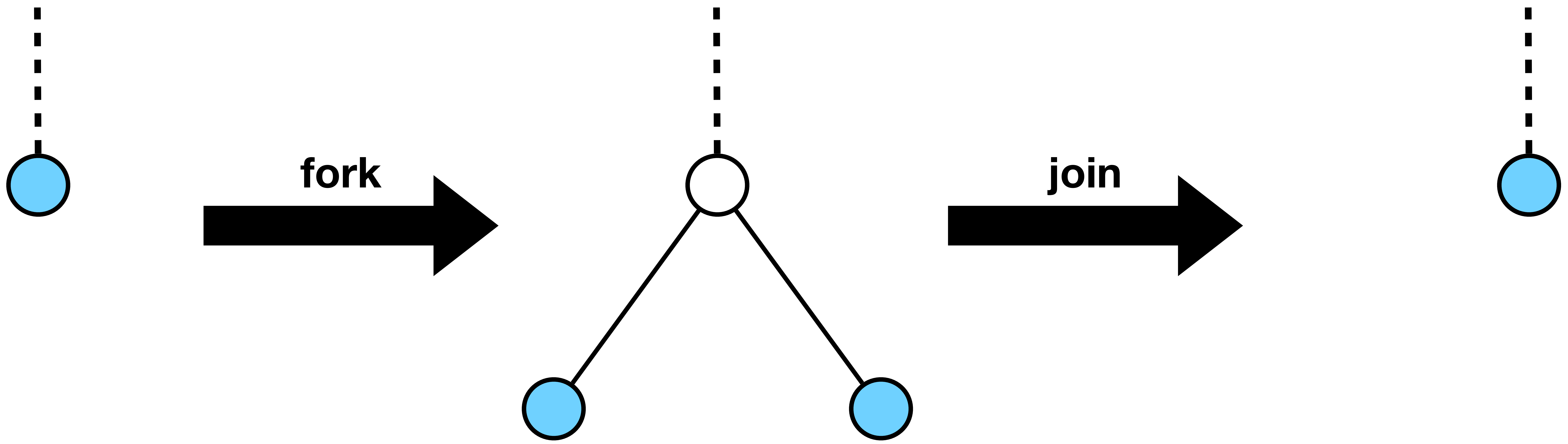


## Is there a better way?

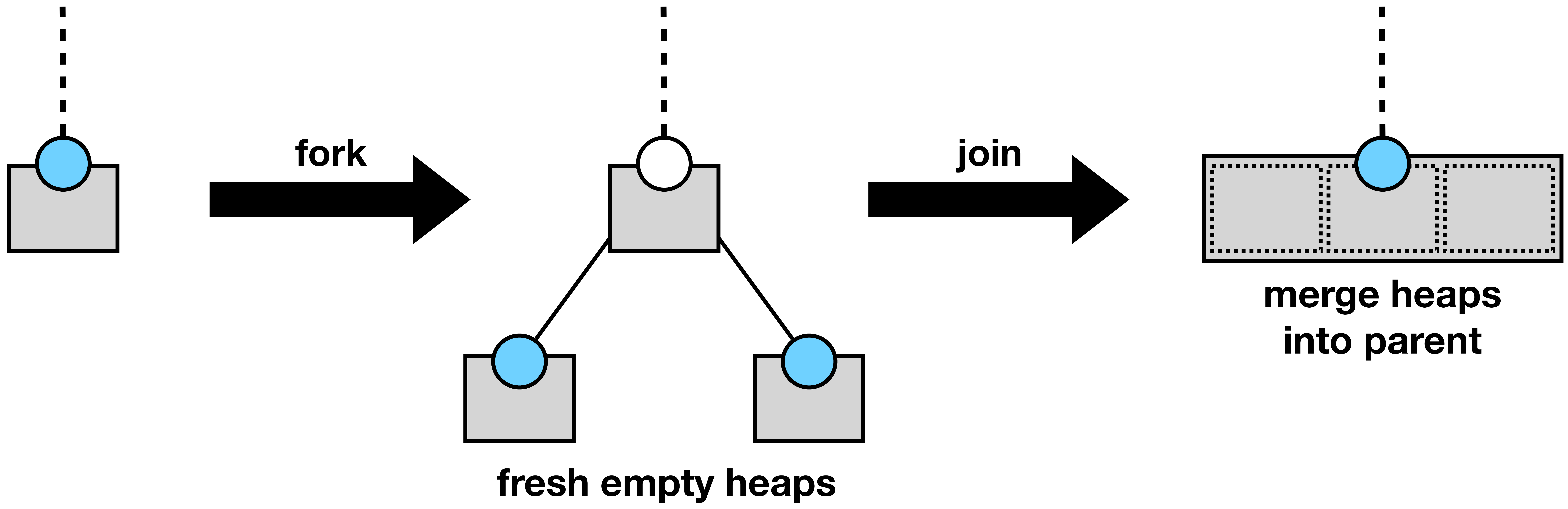

mutator
mutator
mutator
collector
collector
mutator
mutator
collector
memory
mutator
collector
mutator
collector
mutator
mutator
mutator
collector



## Hierarchical Memory Management

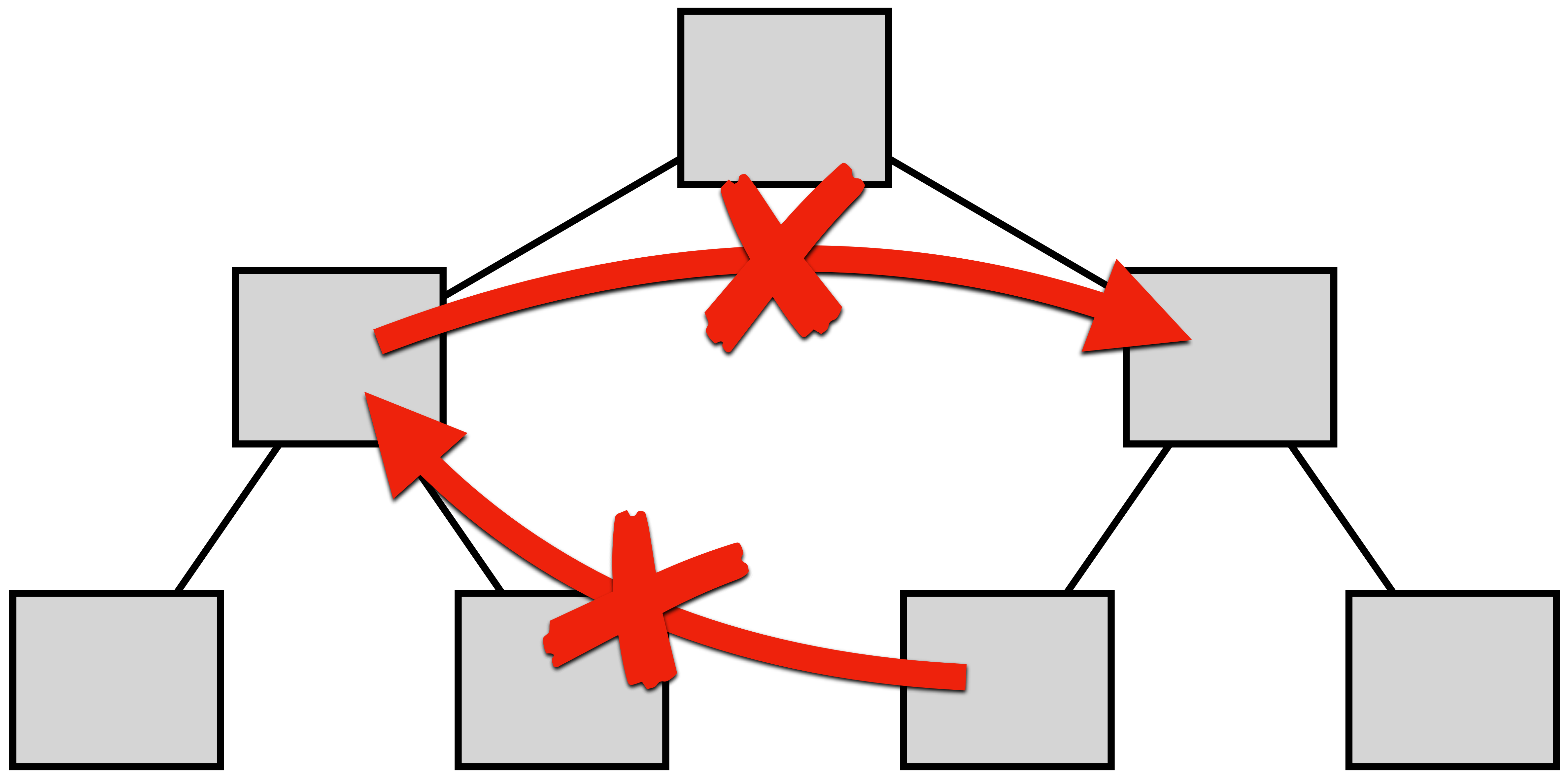


# Hierarchical Memory Management



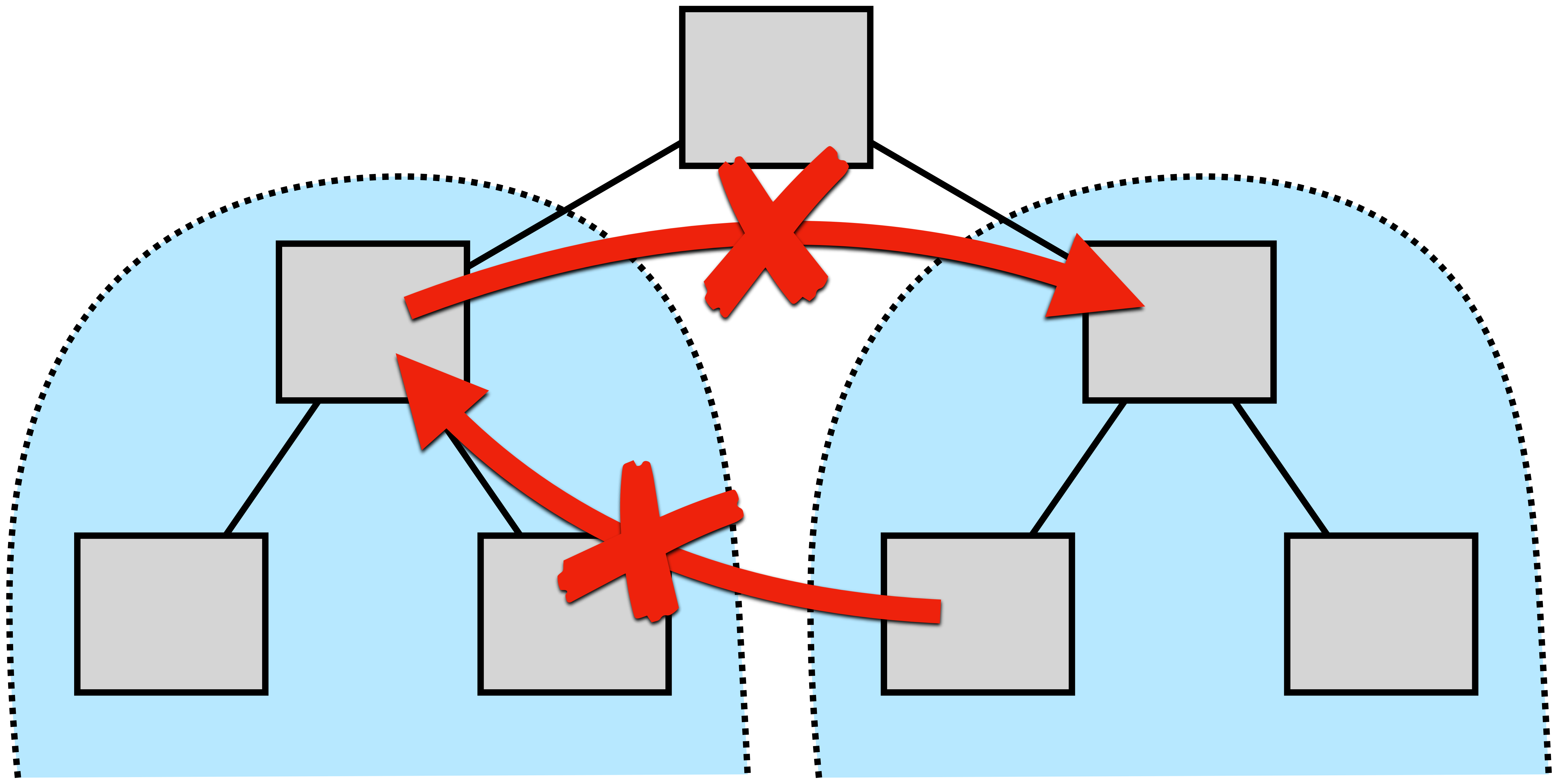
# Hierarchical Memory Management

- disentanglement: ***no cross pointers***



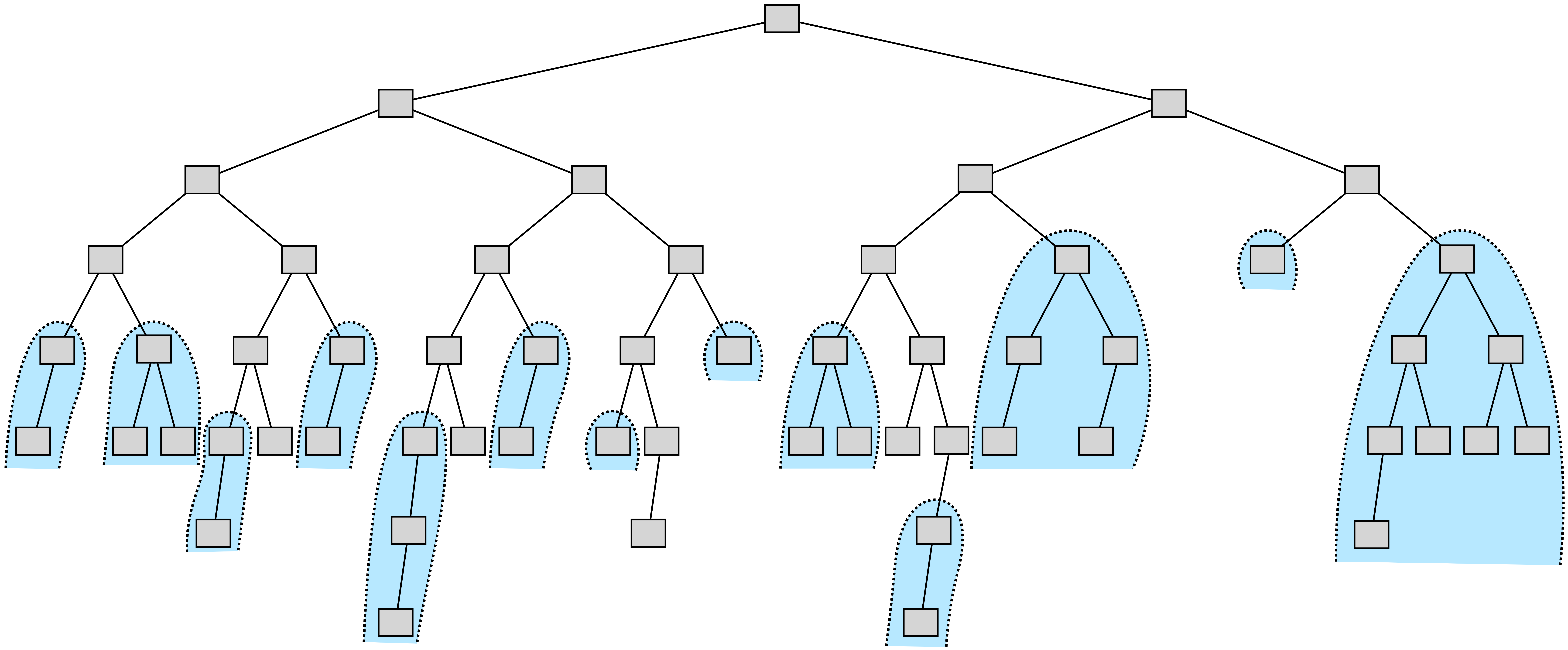
# Hierarchical Memory Management

- disentanglement: ***no cross pointers***
- ***subtree collection***



**reorganize, compact, etc. inside subtree**

**naturally parallel**



## MaPLe

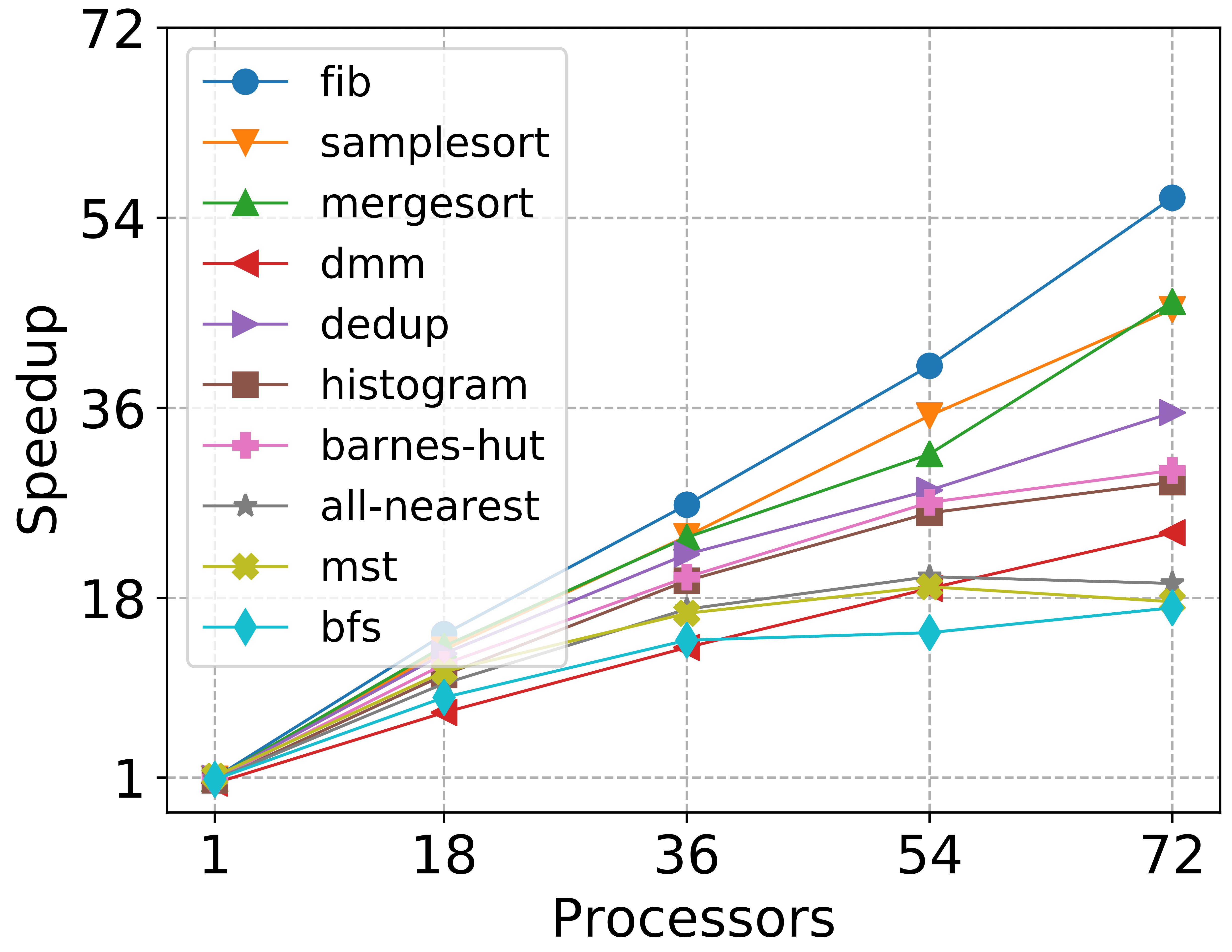
- full ML language, extended with fork-join library

```
val par: (unit -> ‘a) * (unit -> ‘b) -> ‘a * ‘b
```

- used by 500+ students at Carnegie Mellon University each year
- implementation details:
  - extends MLton
  - completely new runtime system
  - subtree collection integrated with scheduling
    - Cheney-style copying/compacting

github.com/mpllang/mpl

# Experiments: Scalability



benchmarks ported to Parallel ML

Speedups relative to MLton

## Experiments: Sorting Shootout

| | $T_1$ | $T_{72}$ |
|---|---|---|
| C++ std::sort | 8.8 | – |
| Cilk samplesort | 7.9 | 0.16 |
| Cilk mergesort | 12.7 | 0.24 |
| MPL (Ours) mergesort | 18.8 | 0.37 |
| Go samplesort | 27.2 | 0.52 |
| Java mergesort | 11.0 | 0.63 |
| Haskell/C mergesort | 10.6 | 1.3 |

2nd fastest, only behind Cilk

## Summary

- disentanglement
    - natural and widespread
    - **question**
      can disentanglement be treated as a correctness condition?
    - **future work**
      static and dynamic checking
- hierarchical memory management
  parallel collection
- MaPLe (MPL)
  real, practical implementation

**github.com/mpllang/mpl**