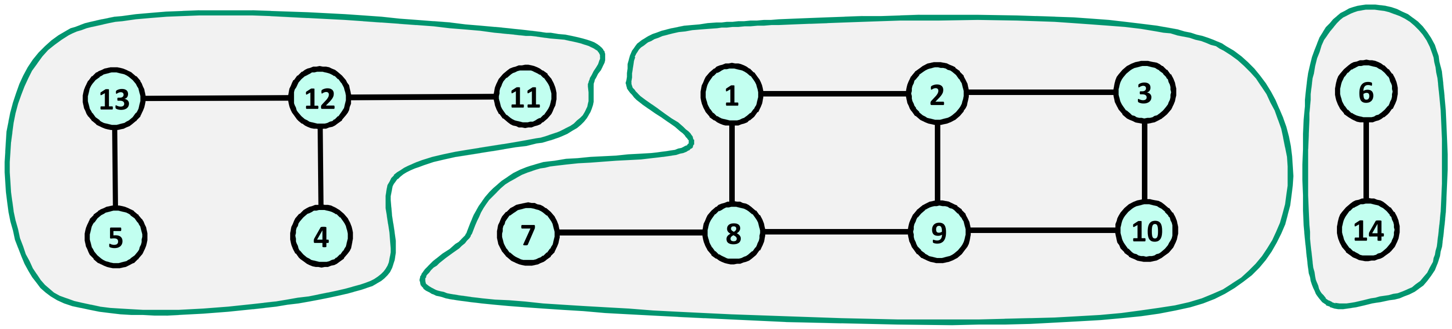
# CSE 613: Parallel Programming

# Lecture 11
# ( Graph Algorithms: Connected Components )

**Rezaul A. Chowdhury**
**Department of Computer Science**
**SUNY Stony Brook**
**Spring 2012**

# Graph Connectivity



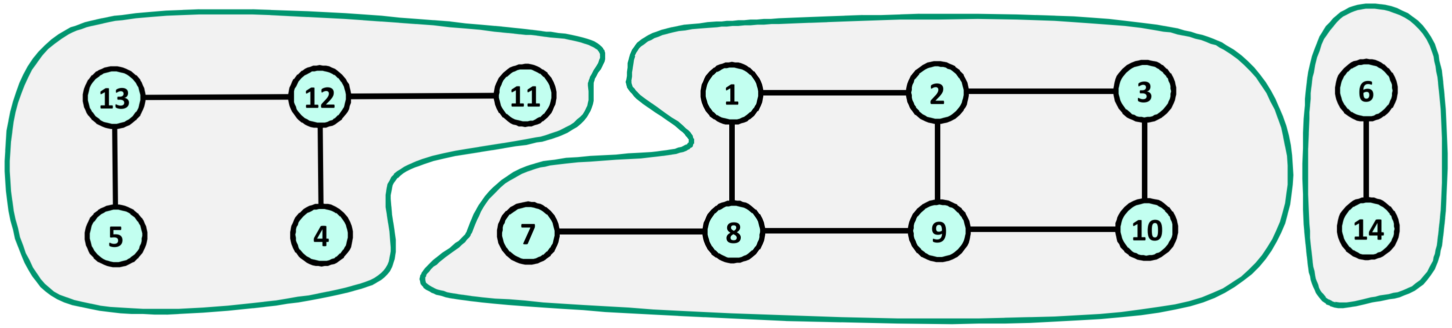

**Connected Components:** A *connected component C* of an undirected graph *G* is a maximal subgraph of *G* such that every vertex in *C* is reachable from every other vertex in *C* following a path in *G.*

**Problem:** Given an undirected graph identify all its connected components.

Suppose *n* is the number of vertices in the graph, and

*m* is the number of edges.

# Graph Connectivity




**Problem:** Identify All connected components of an undirected graph.

Suppose $n$ is the number of vertices in the graph, and

$m$ is the number of edges.

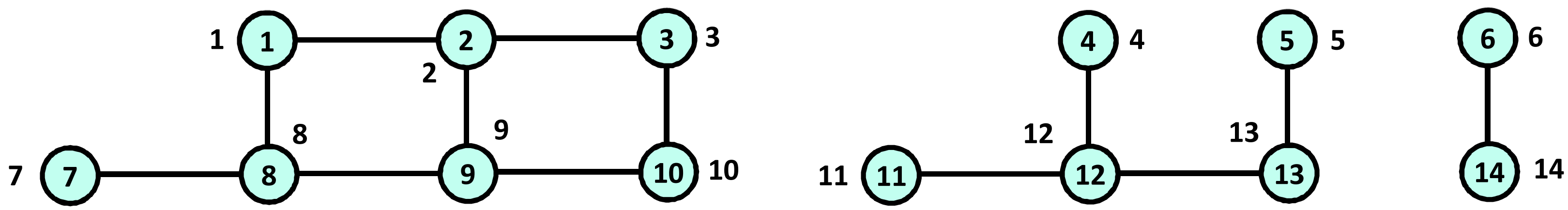
**Serial Algorithms:** Easy to solve in $\Theta(m + n)$ time using

- Depth First Search ( DFS )
- Breadth First Search ( BFS )

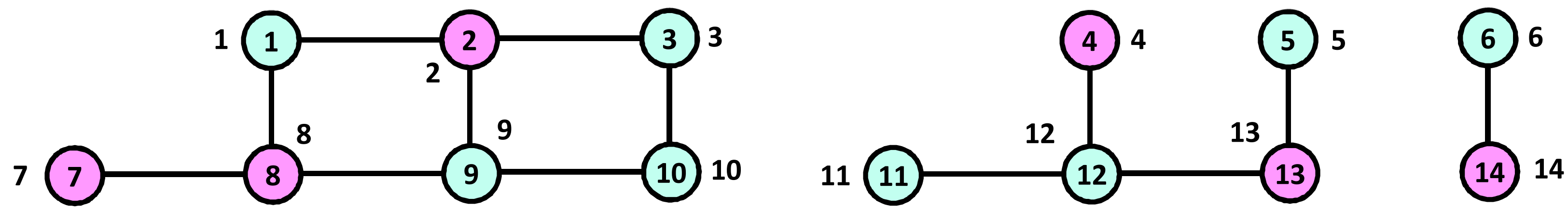
**Parallel Algorithms:**

- **DFS:** Inherently sequential
- **BFS:** Depth equal to the diameter of the graph
- **Graph Contraction:** Can achieve polylogarithmic depth

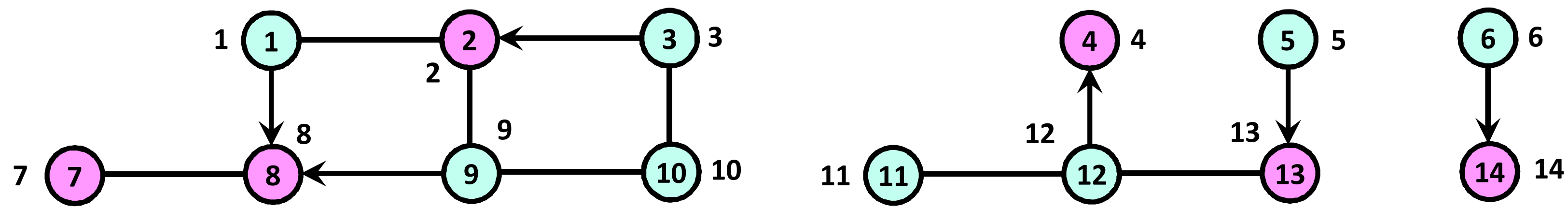
# Randomized Parallel Connected Components



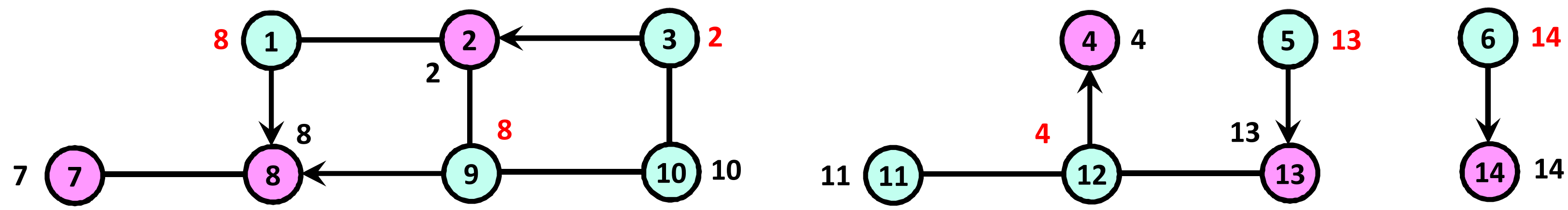
# Randomized Parallel Connected Components



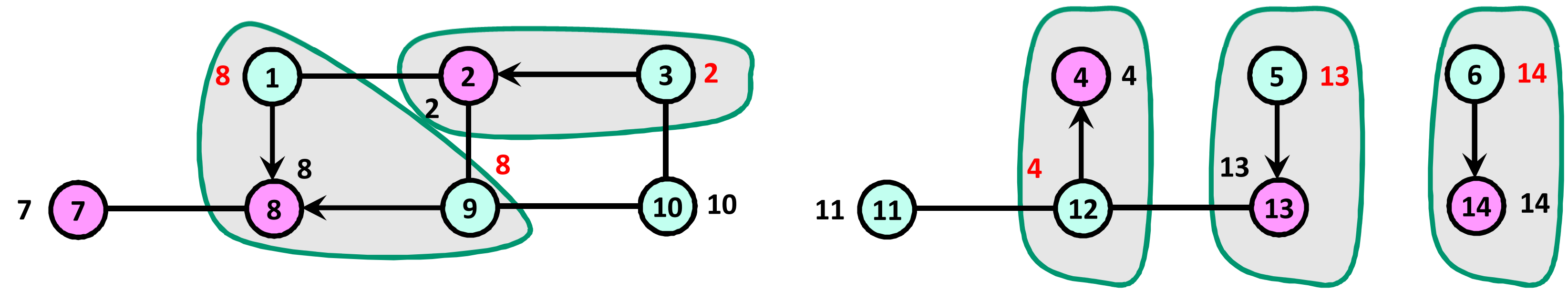
# Randomized Parallel Connected Components



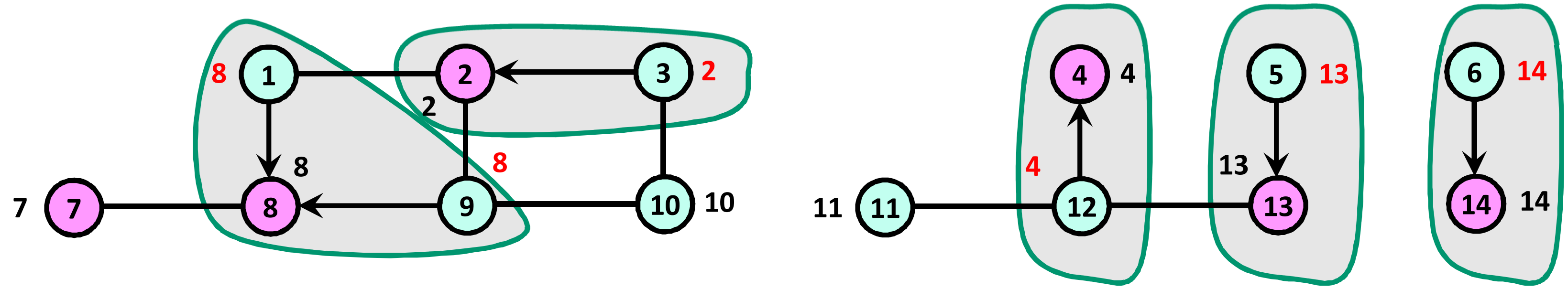
# Randomized Parallel Connected Components



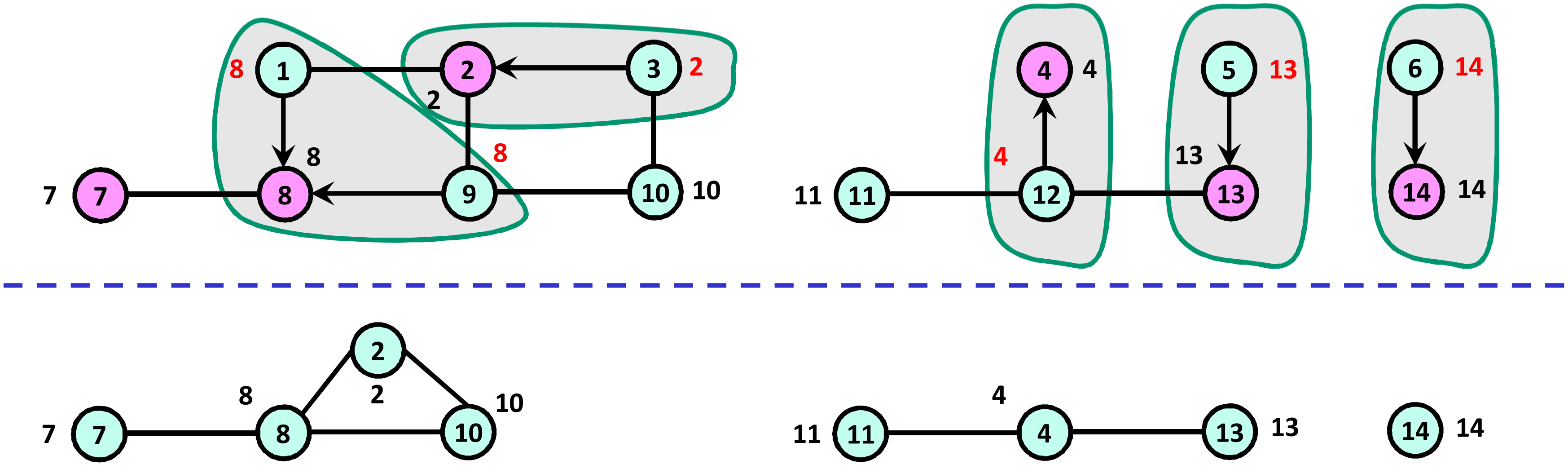
# Randomized Parallel Connected Components



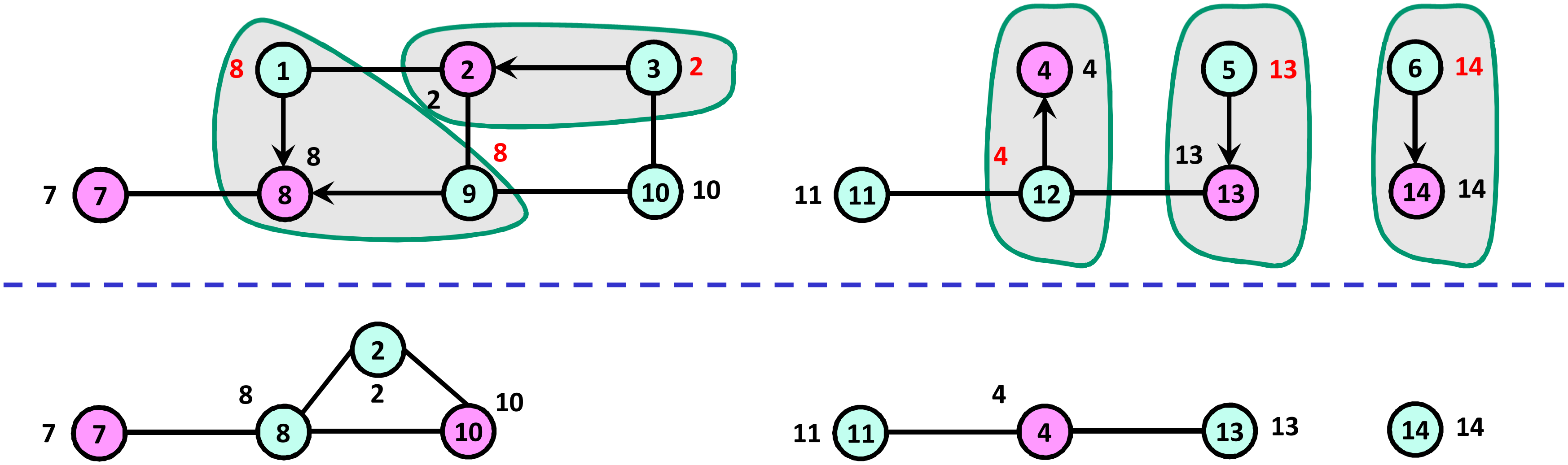
# Randomized Parallel Connected Components



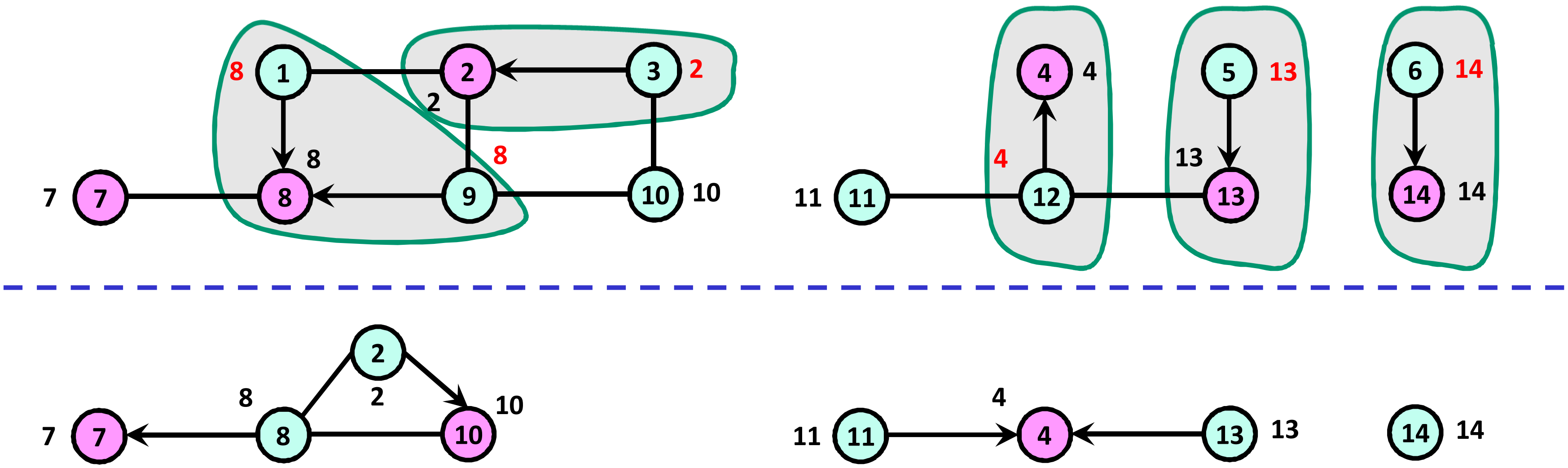
# Randomized Parallel Connected Components



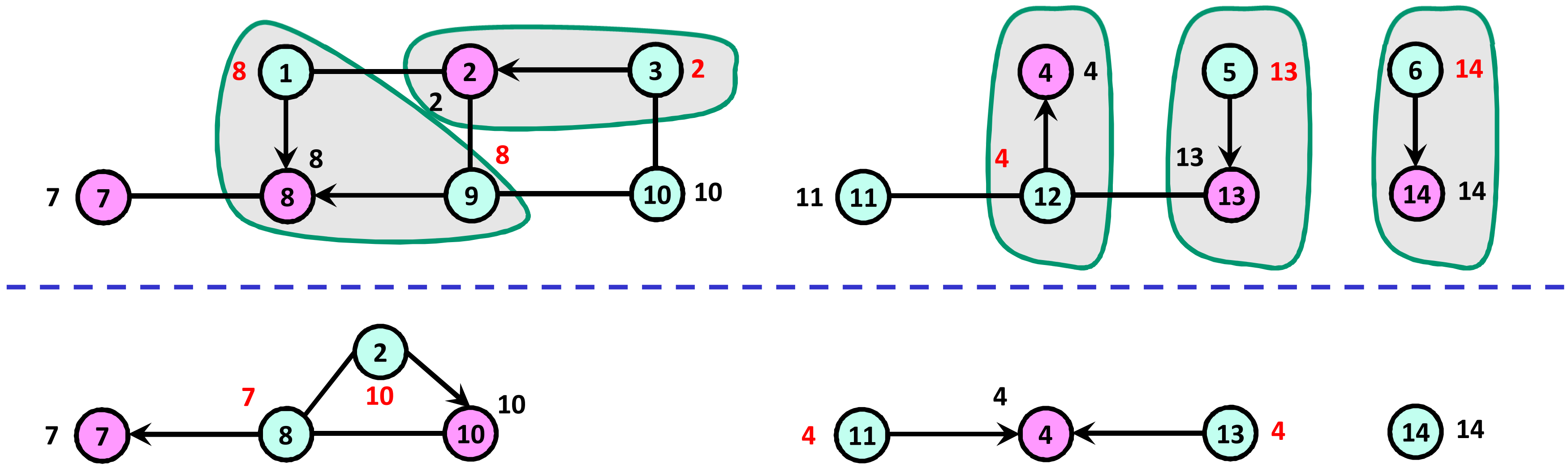
# Randomized Parallel Connected Components



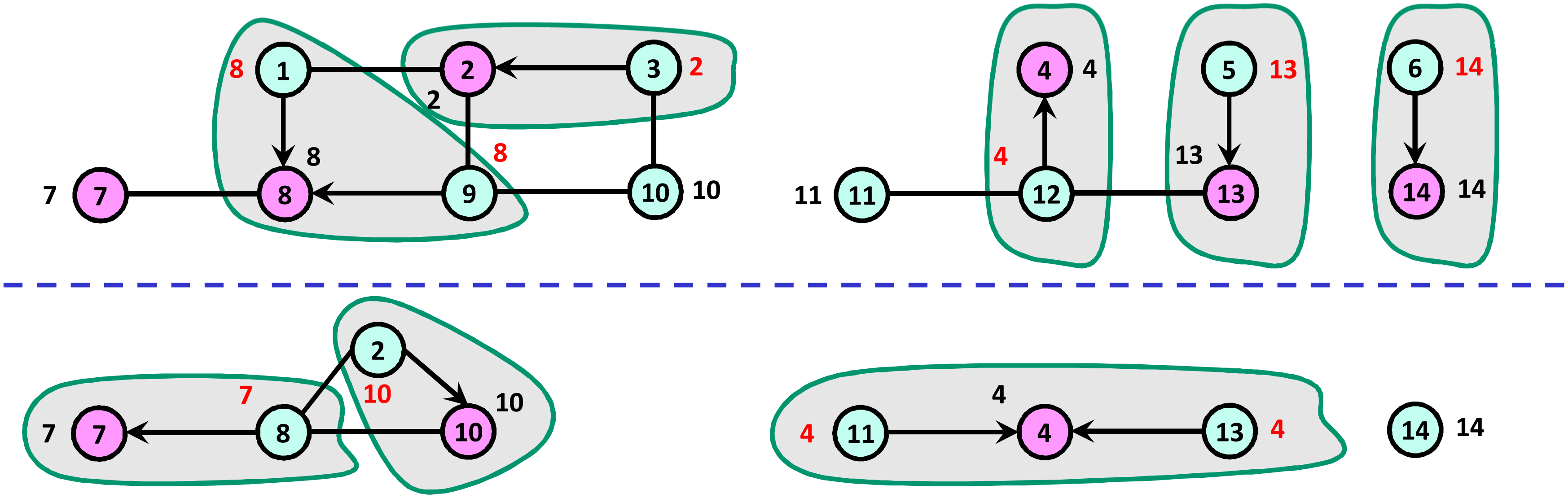
# Randomized Parallel Connected Components



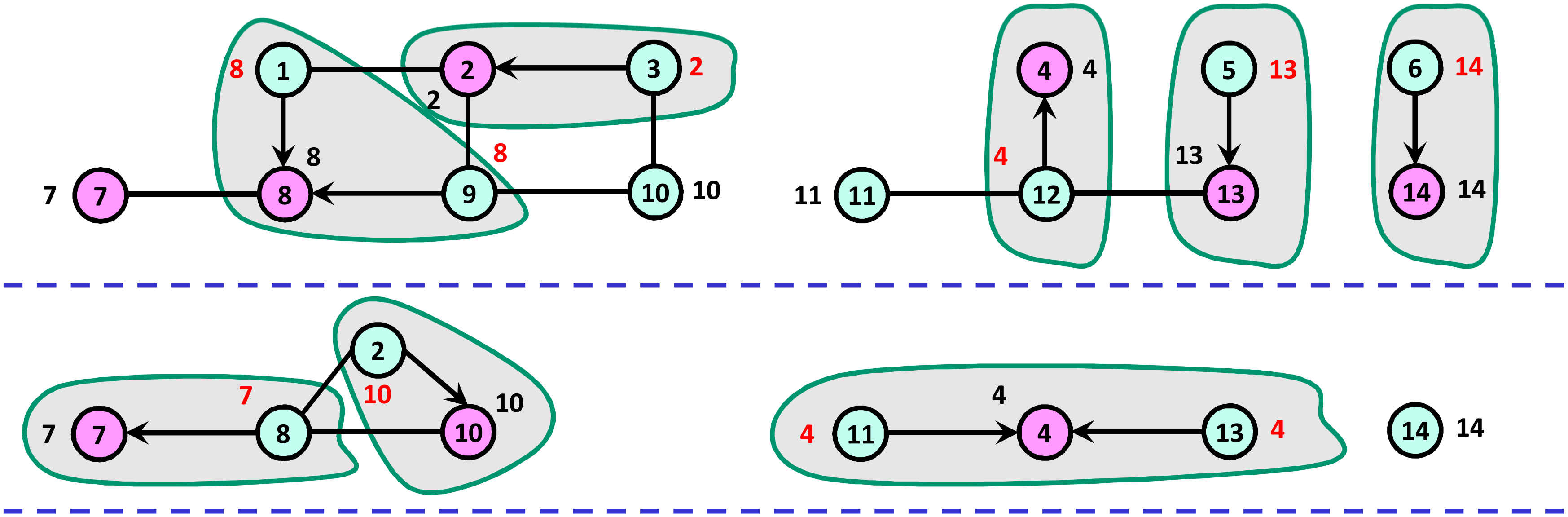
# Randomized Parallel Connected Components



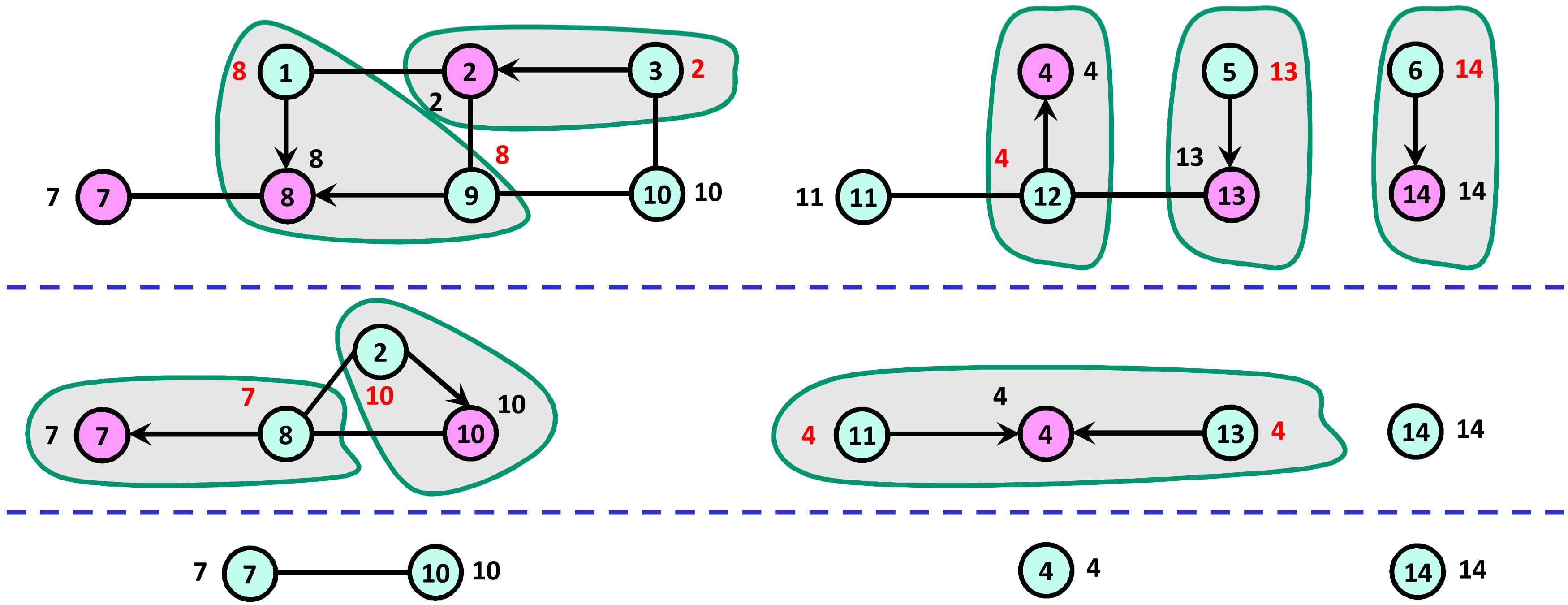
# Randomized Parallel Connected Components



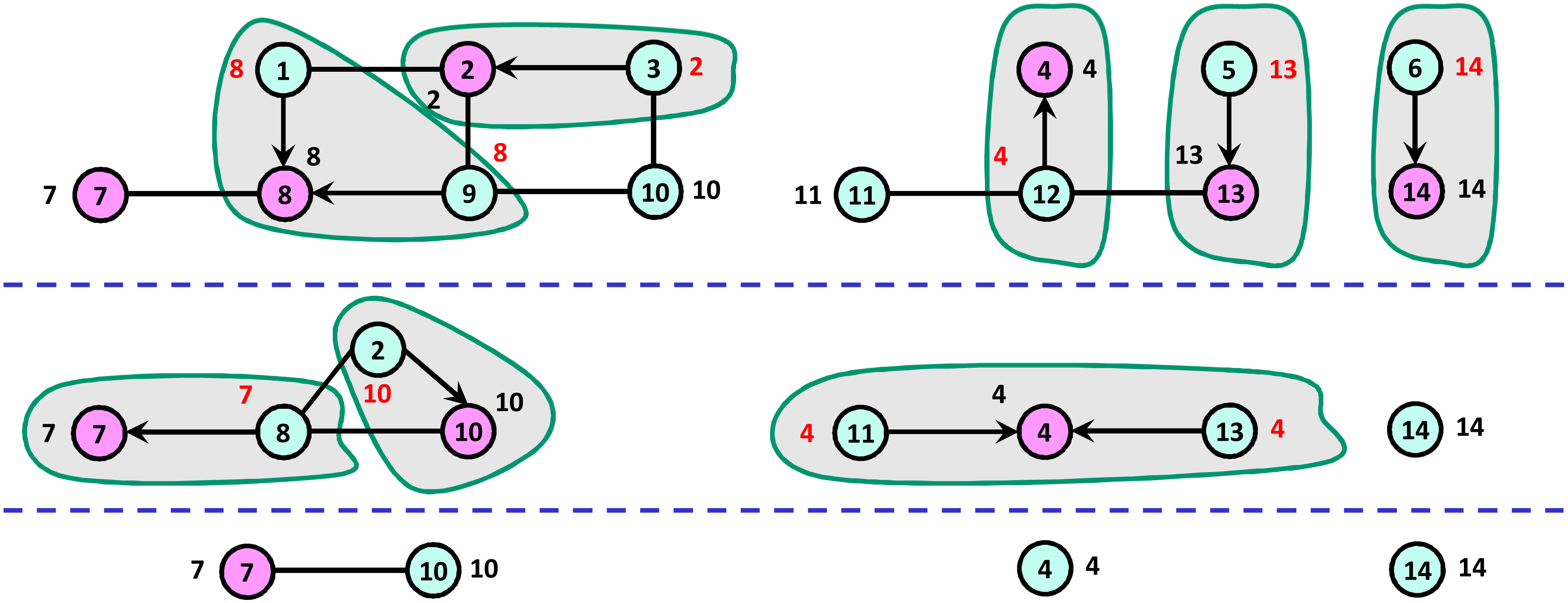
# Randomized Parallel Connected Components



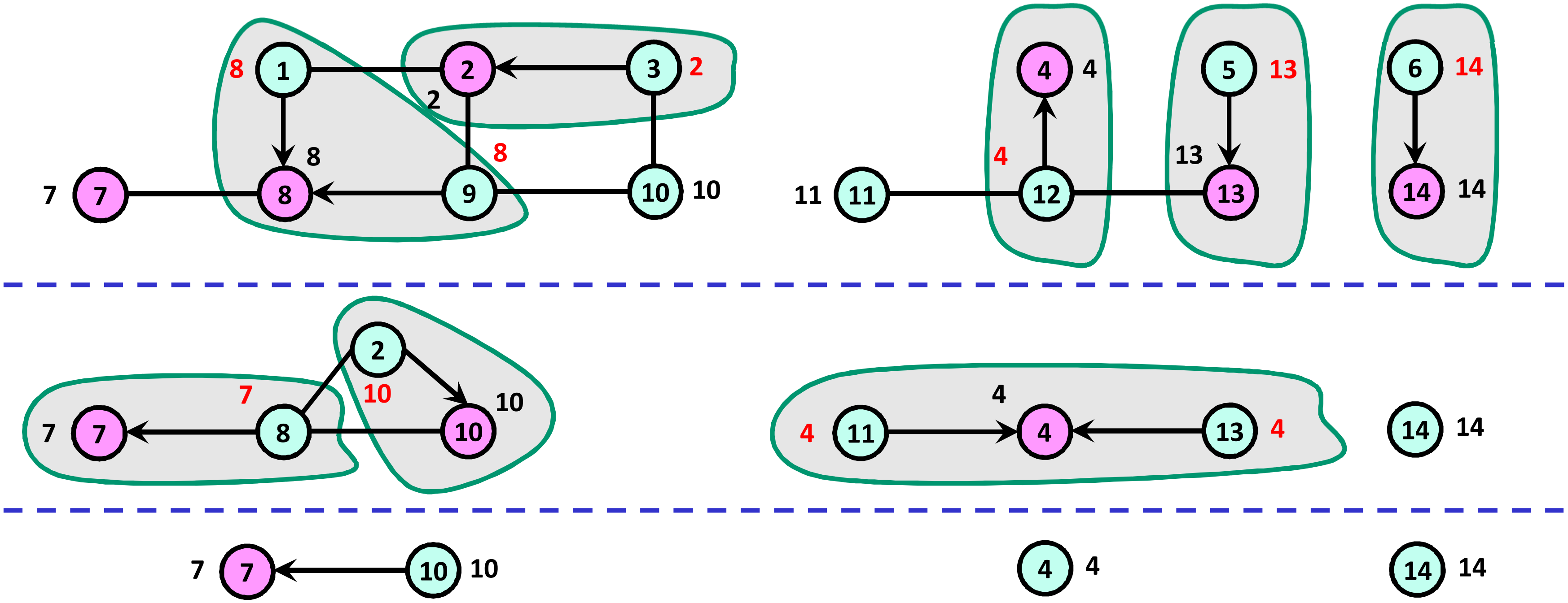
# Randomized Parallel Connected Components



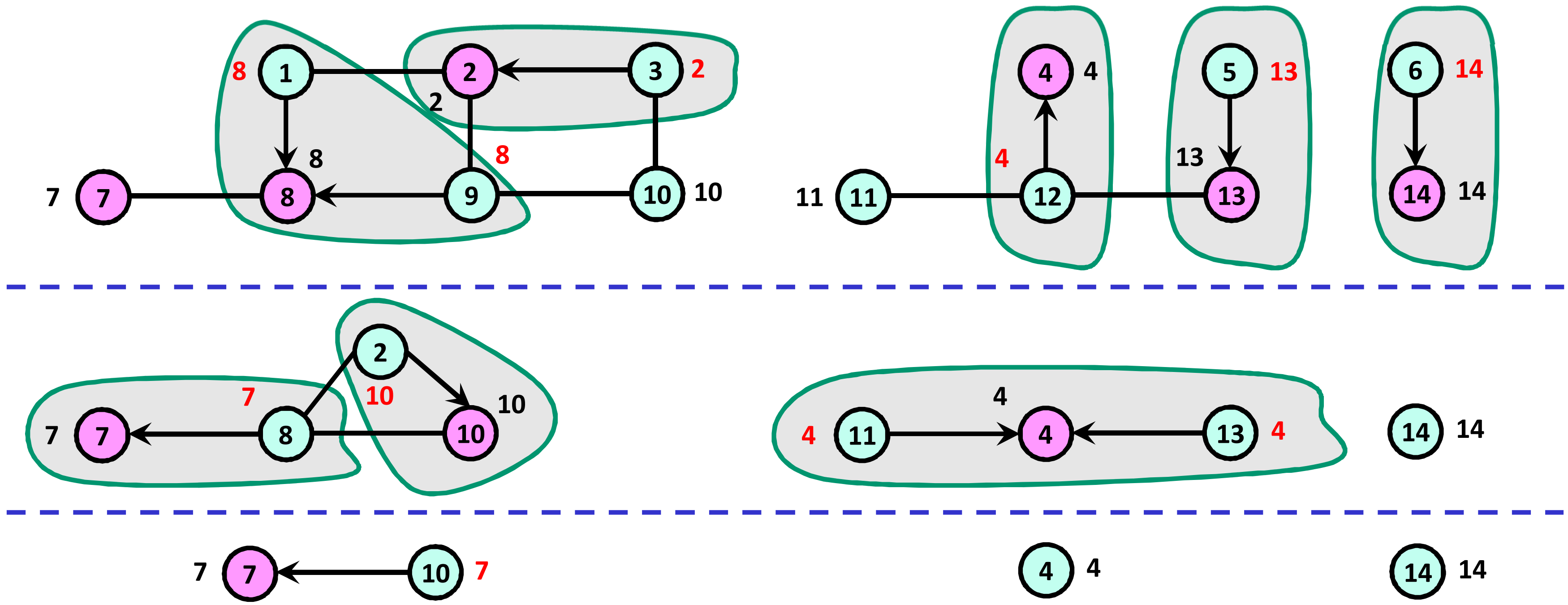
# Randomized Parallel Connected Components



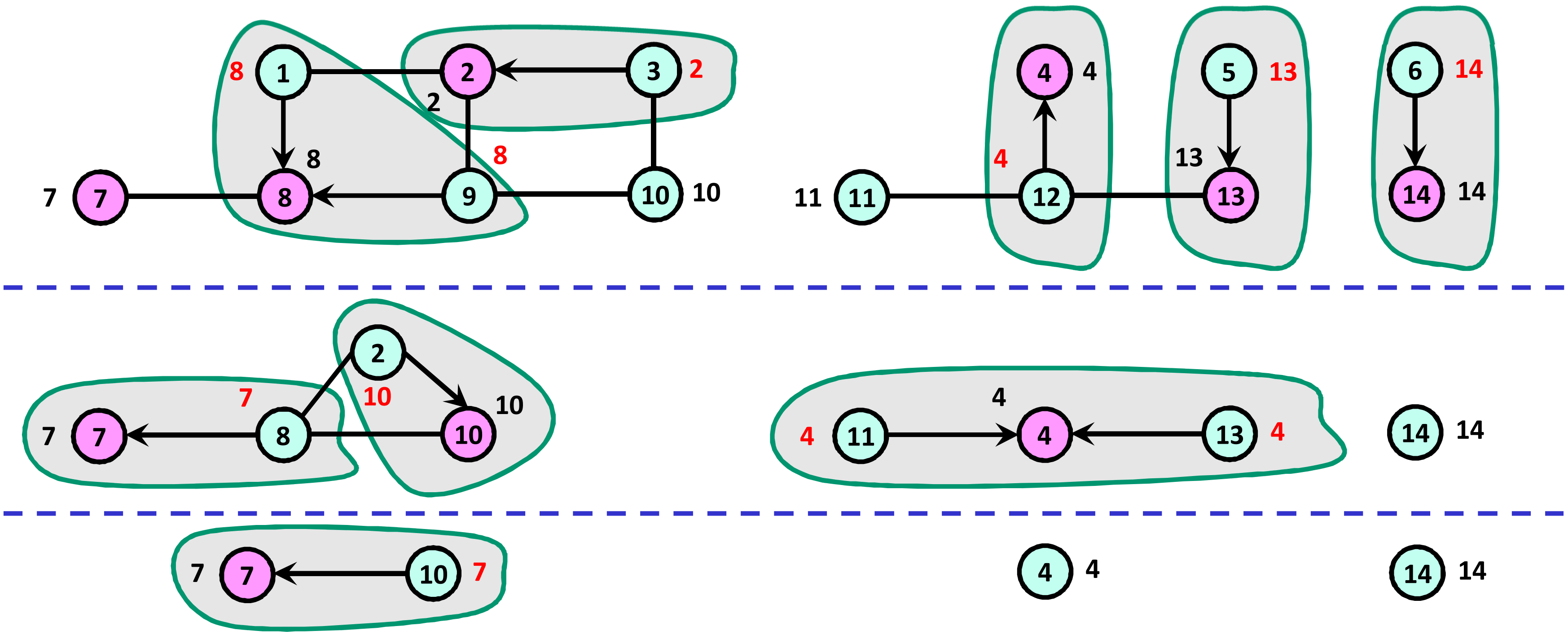
# Randomized Parallel Connected Components



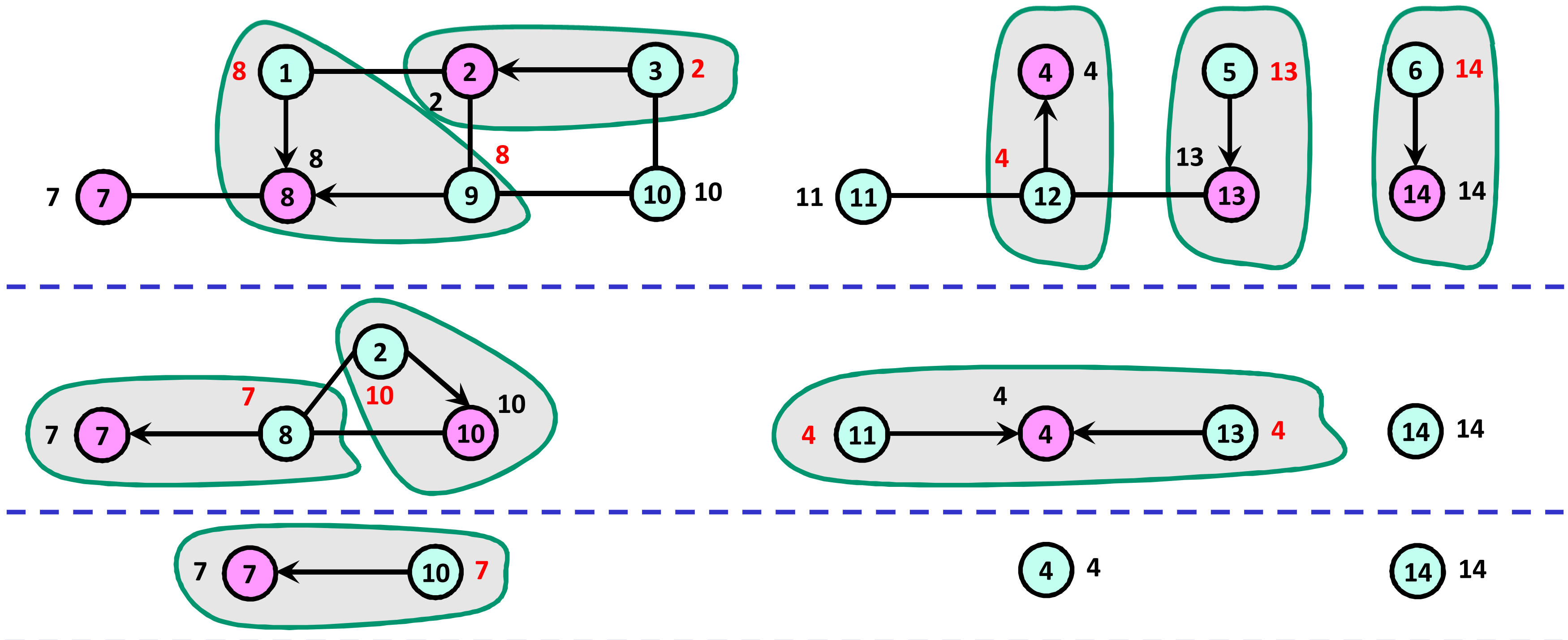
# Randomized Parallel Connected Components



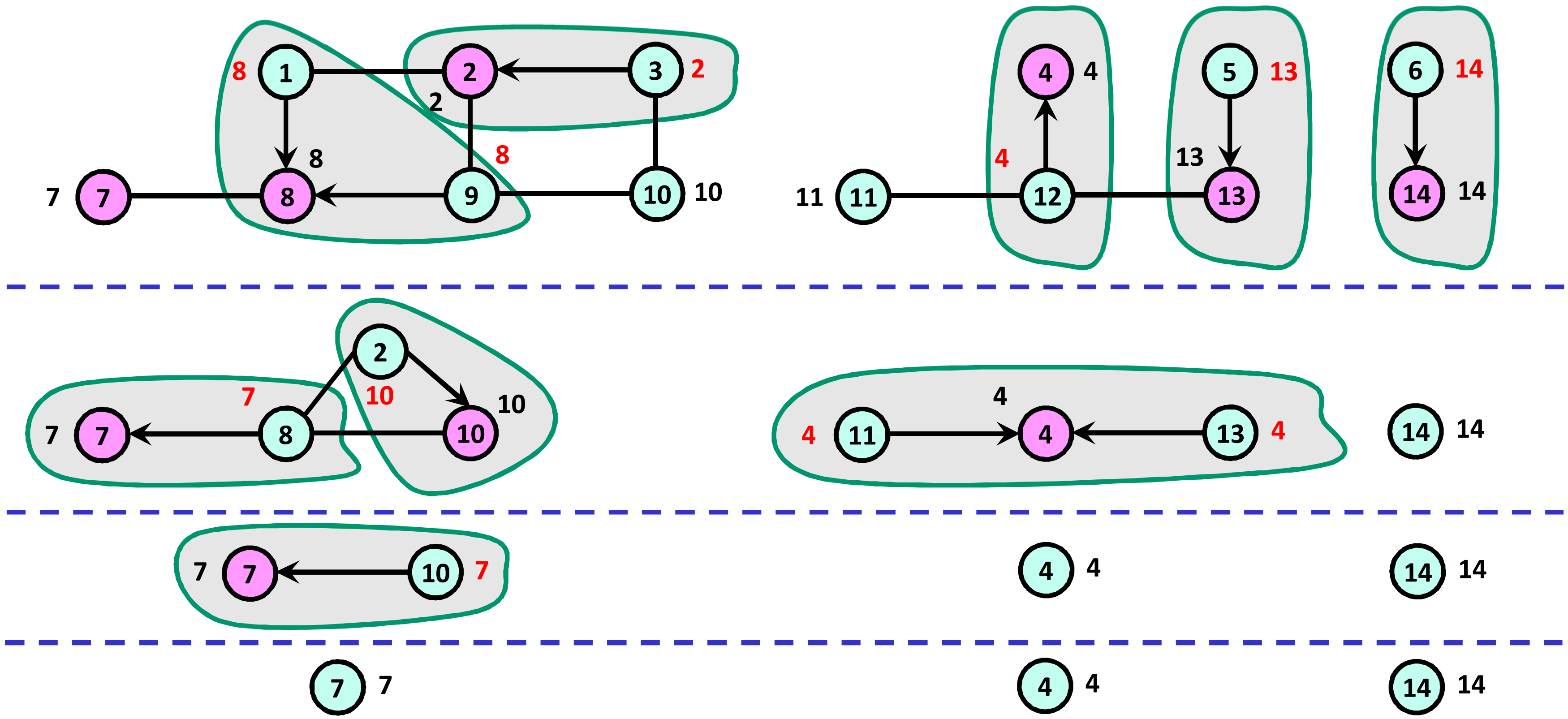
# Randomized Parallel Connected Components



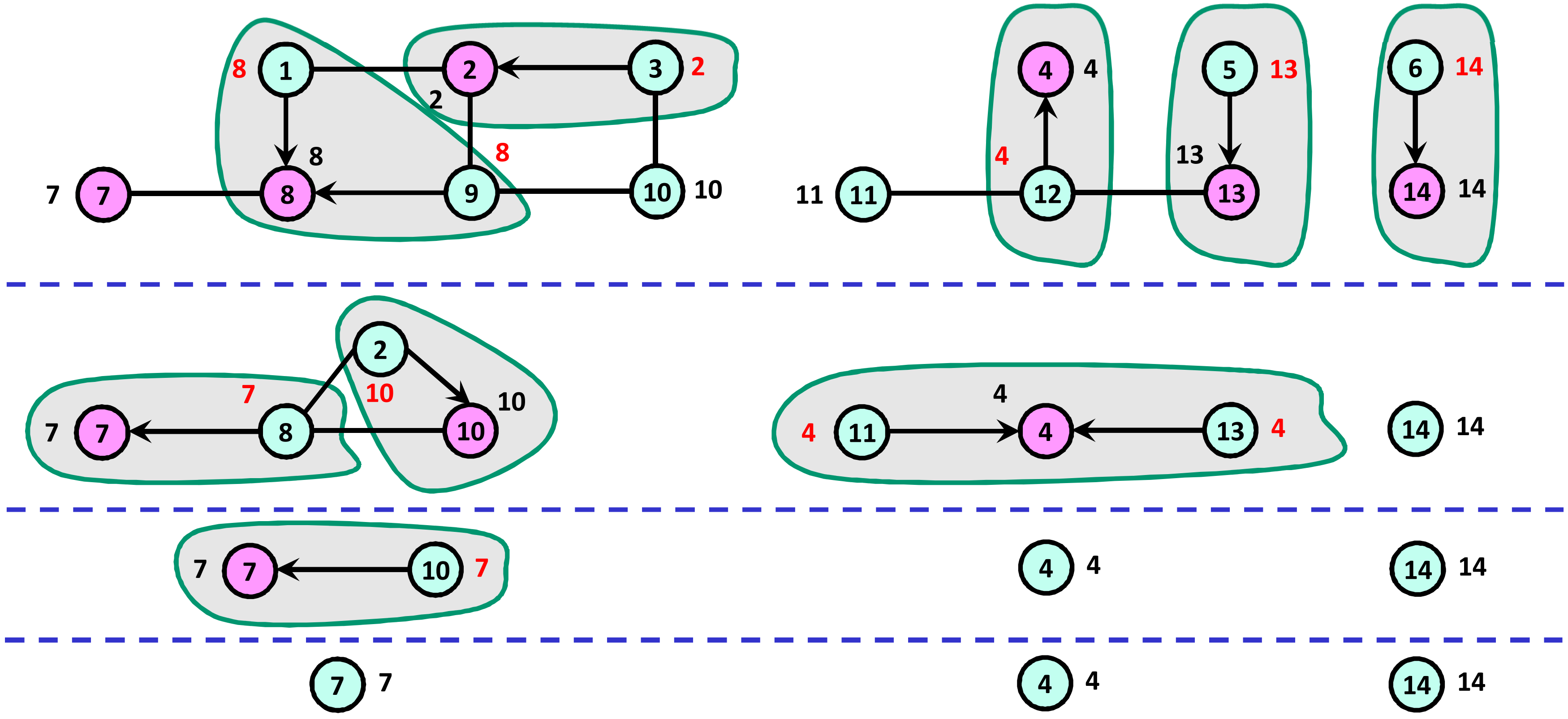
# Randomized Parallel Connected Components



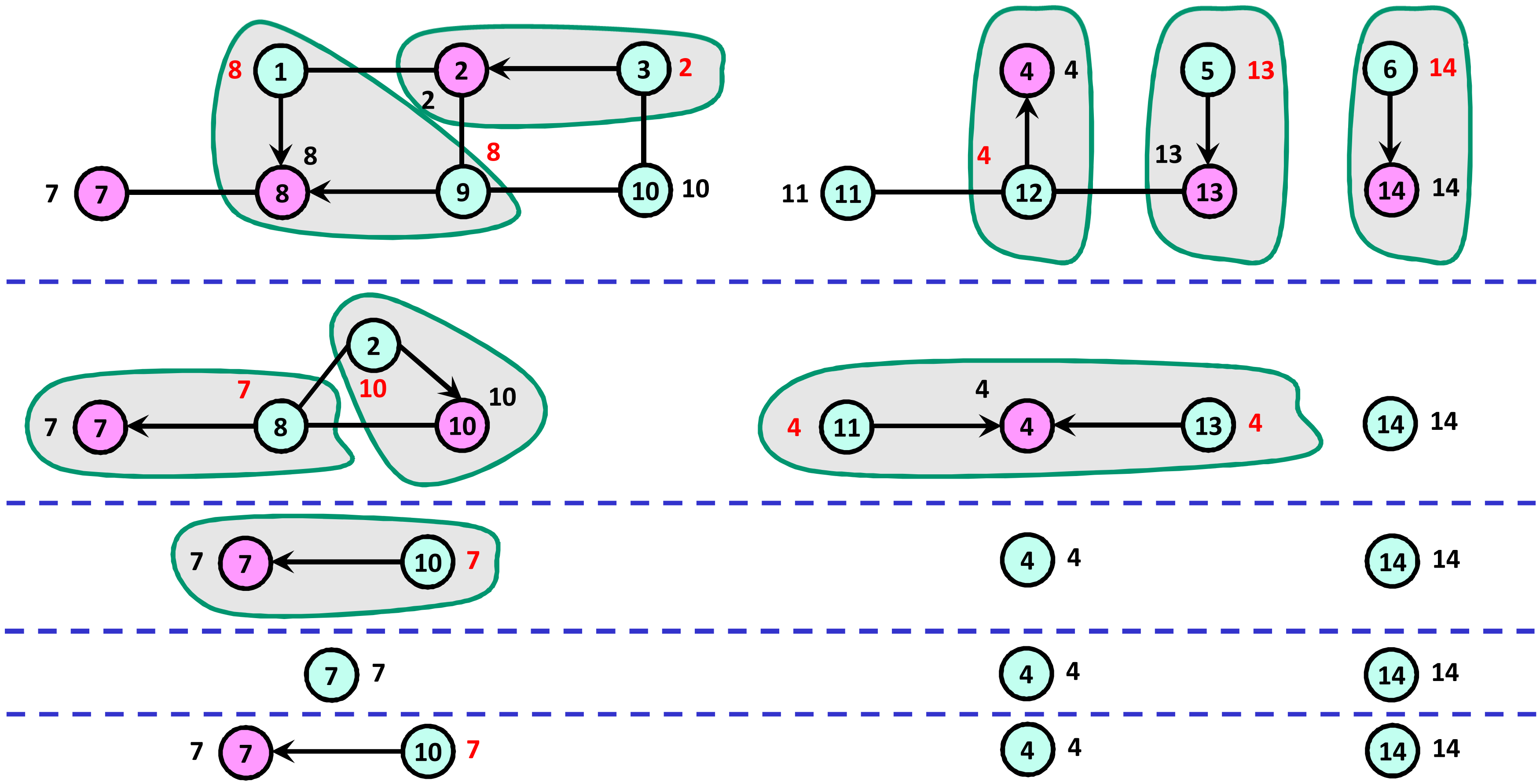
# Randomized Parallel Connected Components



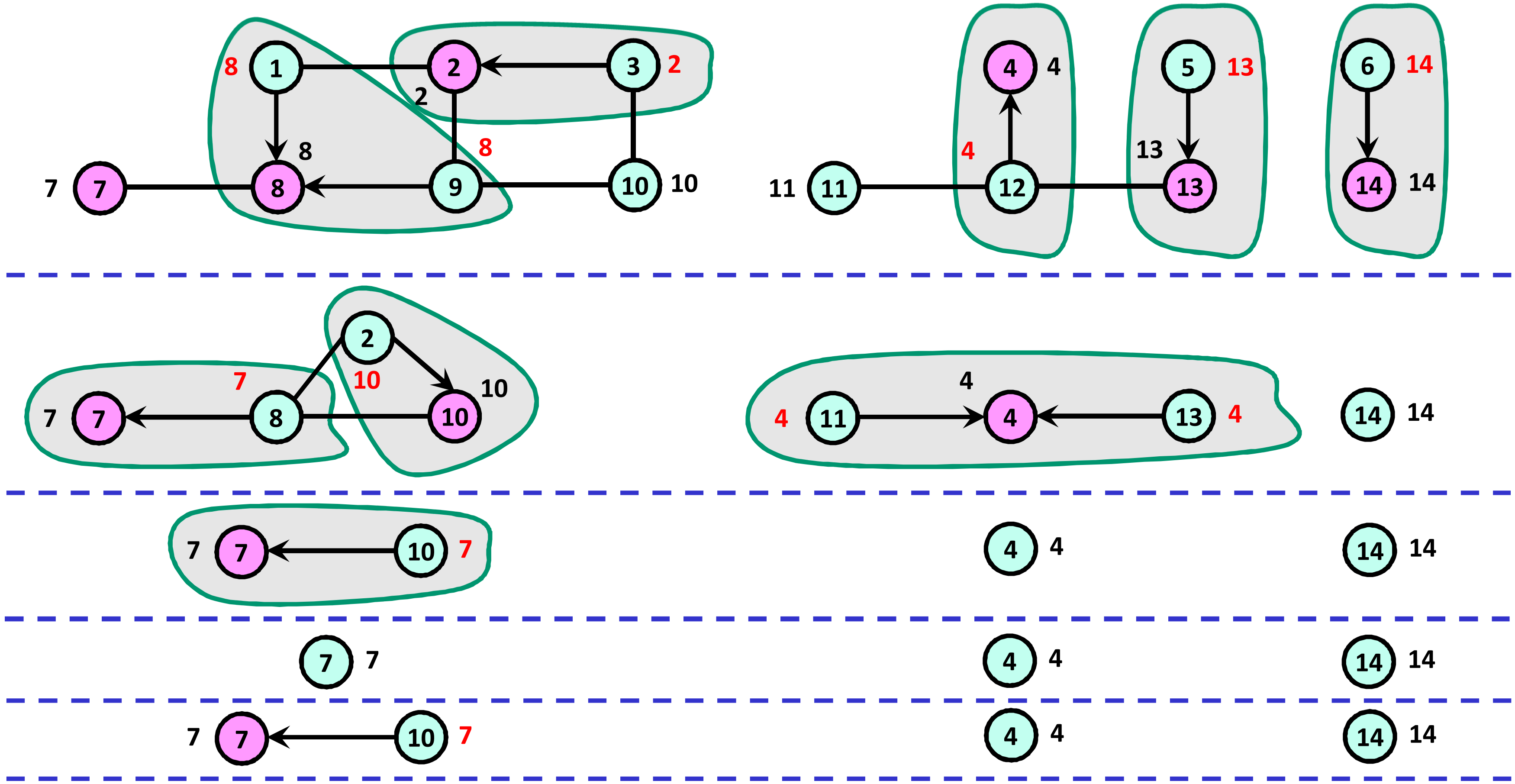
# Randomized Parallel Connected Components



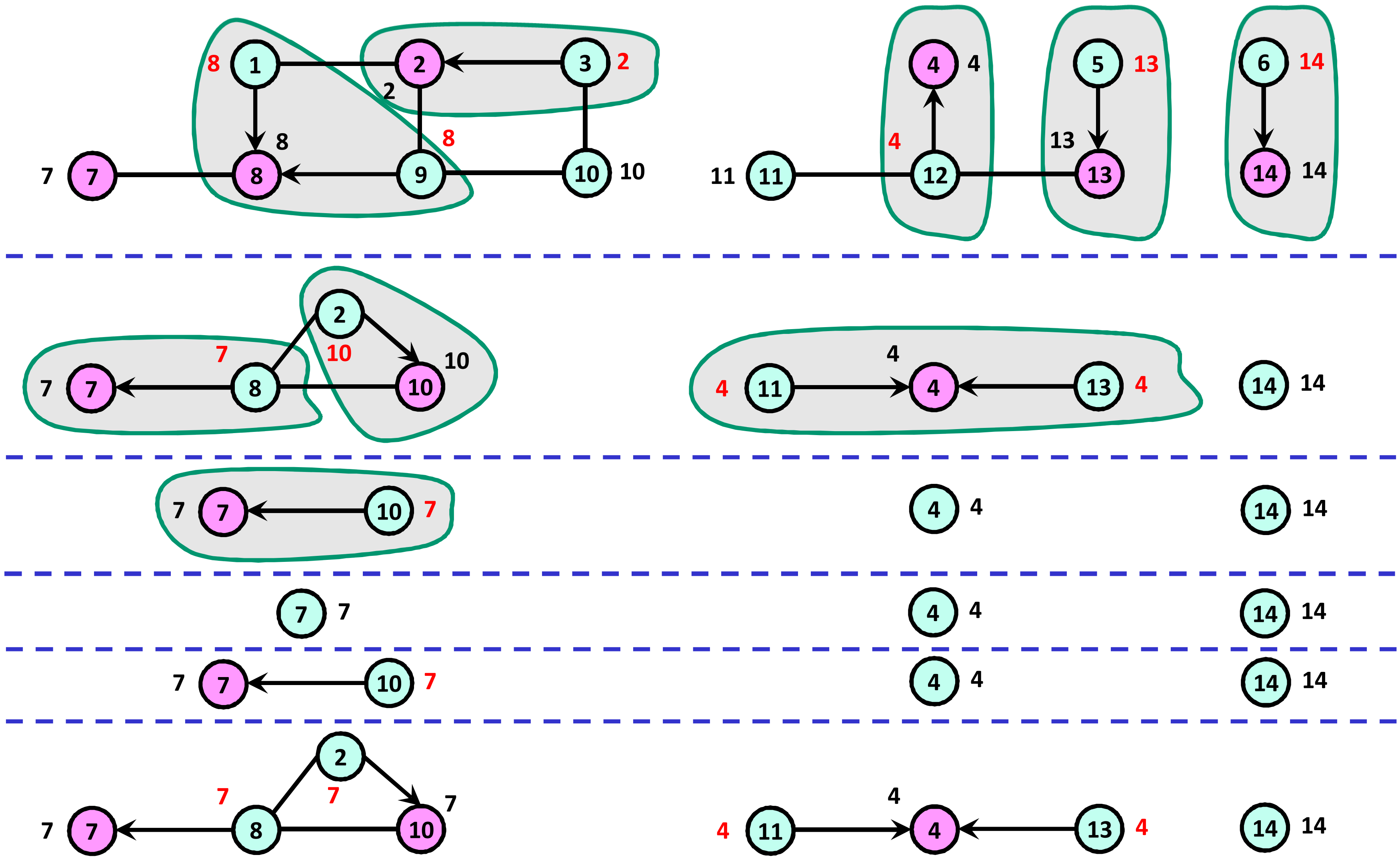
# Randomized Parallel Connected Components



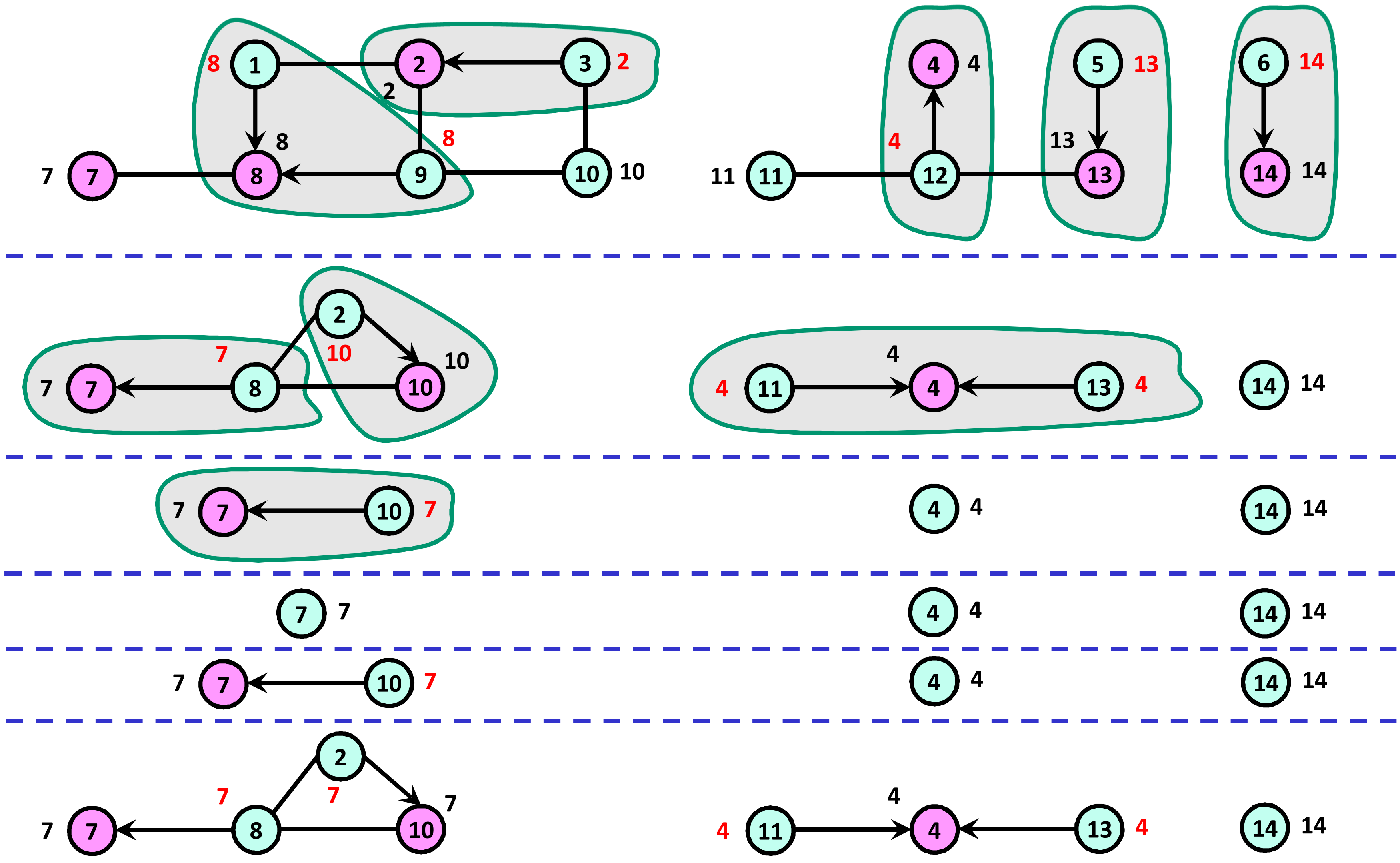
# Randomized Parallel Connected Components



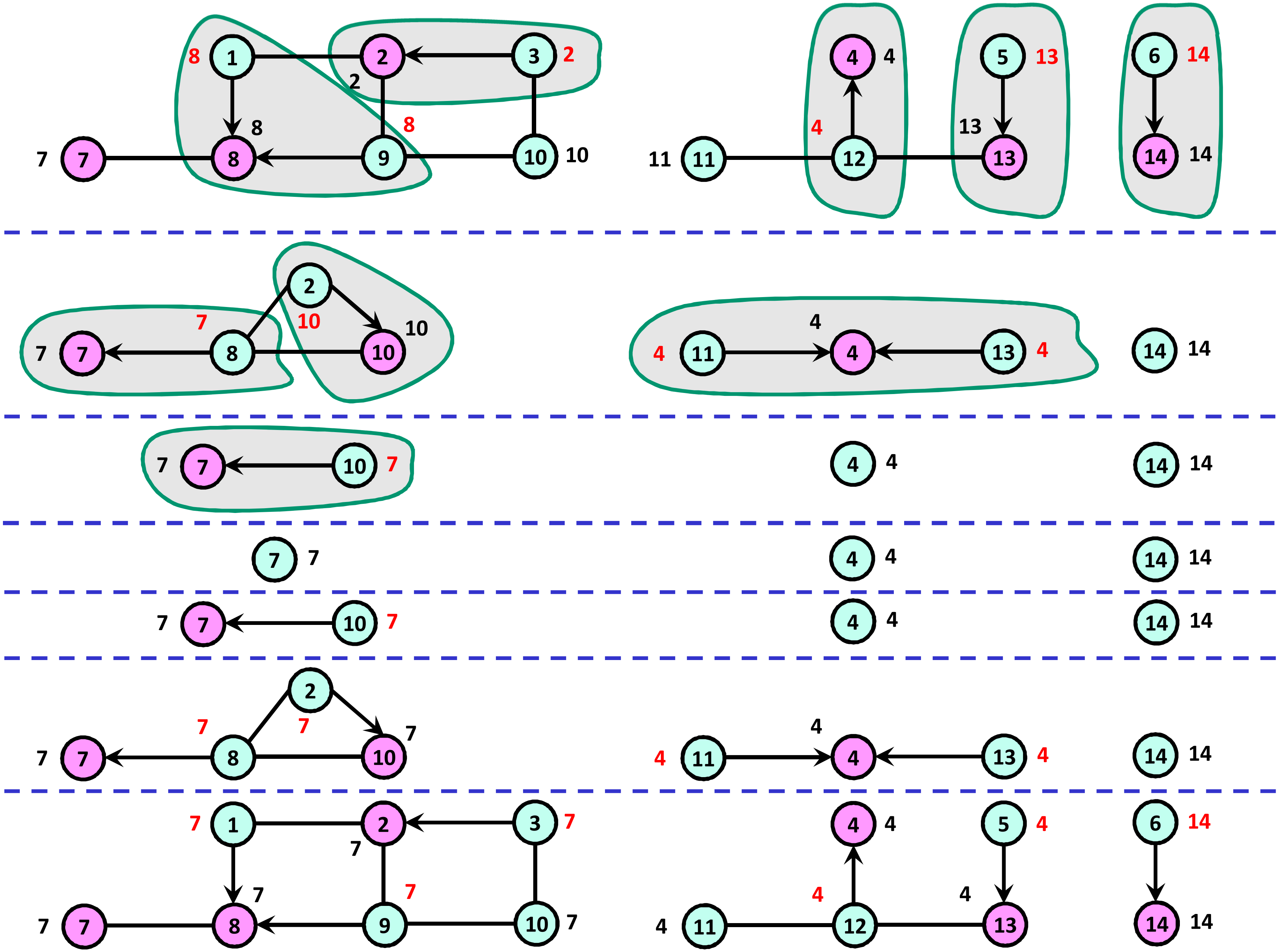
# Randomized Parallel Connected Components



# Randomized Parallel Connected Components



# Randomized Parallel Connected Components



# Randomized Parallel Connected Components (CC)

**Input:** *n* is the number of vertices in the graph numbered from 1 to *n*, *E* is the set of edges, and *L*[ 1 : *n* ] are vertex labels with *L*[ *v* ] = *v* initially for all *v*.

**Output:** An array *M*[ 1: *n* ] where for all *v*, *M*[ *v* ] is the unique id of the connected component containing *v*.

```
Par-Randomized-CC ( n, E, L )
if |E| = 0 then return L
array C[ 1 : n ], M[ 1 : n ], S[ 1 : |E| ]
parallel for v ← 1 to n do C[ v ] ← RANDOM{ Head, Tail }
parallel for each ( u, v ) ∈ E do
    if C[ u ] = Tail and C[ v ] = Head then L[ u ] ← v
parallel for i ← 1 to |E| do
    if L[ E[ i ].u ] ≠ L[ E[ i ].v ] then S[ i ] ← 1 else S[ i ] ← 0
S ← Par-Prefix-Sum ( S, + )
array F[ 1 : S[ |E| ] ]
parallel for i ← 1 to |E| do
    if L[ E[ i ].u ] ≠ L[ E[ i ].v ] then
        F[ S[ i ] ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
M ← Par-Randomized-CC ( n, F, L )
parallel for each ( u, v ) ∈ E do
    if v = L[ u ] then M[ u ] ← M[ v ]
return M
```

unbiased coin toss at each vertex

group: hook child to a parent ( race! )

prepare to remove intra-group edges

find the rank of each inter-group edge among all such edges

copy the inter-group edges to *F*

find CC in the contracted graph

Map results back to the original graph

# Randomized Parallel Connected Components (CC)

```
Par-Randomized-CC ( n, E, L )
    if |E| = 0 then return L
    array C[ 1 : n ], M[ 1 : n ], S[ 1 : |E| ]
    parallel for v ← 1 to n do
        C[ v ] ← RANDOM{ Head, Tail }
    parallel for each ( u, v ) ∈ E do
        if C[ u ] = Tail and C[ v ] = Head then L[ u ] ← v
    parallel for i ← 1 to |E| do
        if L[ E[ i ].u ] ≠ L[ E[ i ].v ] then S[ i ] ← 1
        else S[ i ] ← 0
    S ← Par-Prefix-Sum ( S, + )
    array F[ 1 : S[ |E| ] ]
    parallel for i ← 1 to |E| do
        if L[ E[ i ].u ] ≠ L[ E[ i ].v ] then
            F[ S[ i ] ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
    M ← Par-Randomized-CC ( n, F, L )
    parallel for each ( u, v ) ∈ E do
        if v = L[ u ] then M[ u ] ← M[ v ]
    return M
```

Suppose *n* is the number of vertices and *m* is the number of edges in the original graph.

Each contraction is expected to reduce #vertices by a factor of at least $\frac{1}{4}$. [ why? ]

So, the expected number of contraction steps, $D = O(\log n)$. [ show: the bound holds w.h.p. ]

For each contraction step span is $\Theta(\log^2 n)$, and work is $\Theta(n + m)$. [ why? ]

**Work:** $T_1(n, m) = \Theta\big(D(n + m)\big)$

$= O\big((n + m)\log n\big)$ ( w.h.p. )

**Span:** $T_\infty(n, m) = \Theta(D\log^2 n)$

$= O(\log^3 n)$ ( w.h.p. )

**Parallelism:** $\frac{T_1(n,m)}{T_\infty(n,m)} = \Theta\left(\frac{n+m}{\log^2 n}\right)$

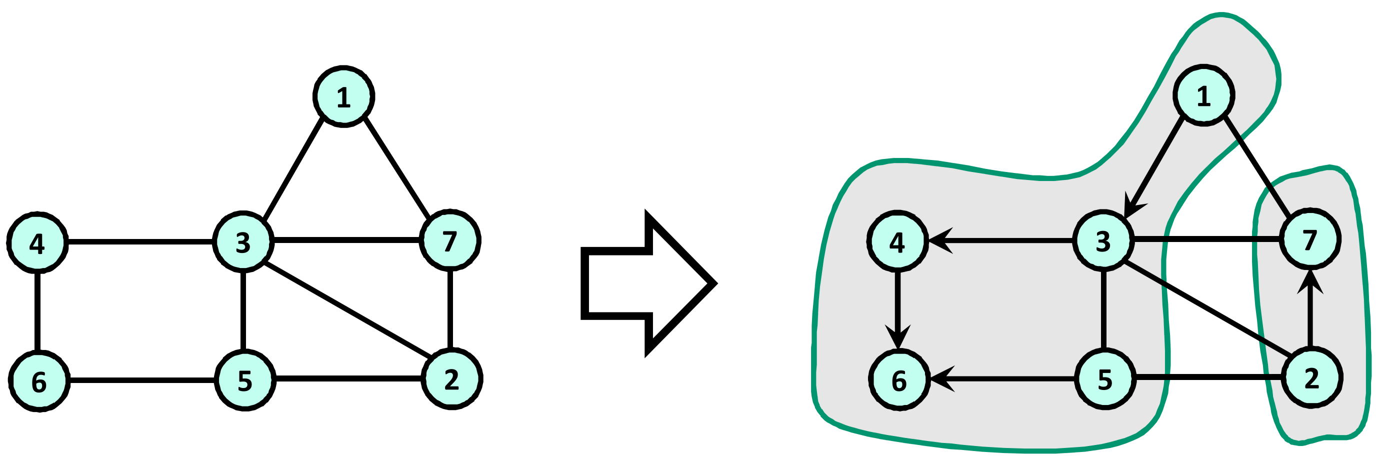
# Deterministic Parallel Connected Components (CC)

**Approach**

- Form a set of disjoint subtrees
- Use pointer-jumping to reduce each subtree to a single vertex
- Recursively apply the same trick on the contracted graph

**Forming Disjoint Subtrees**

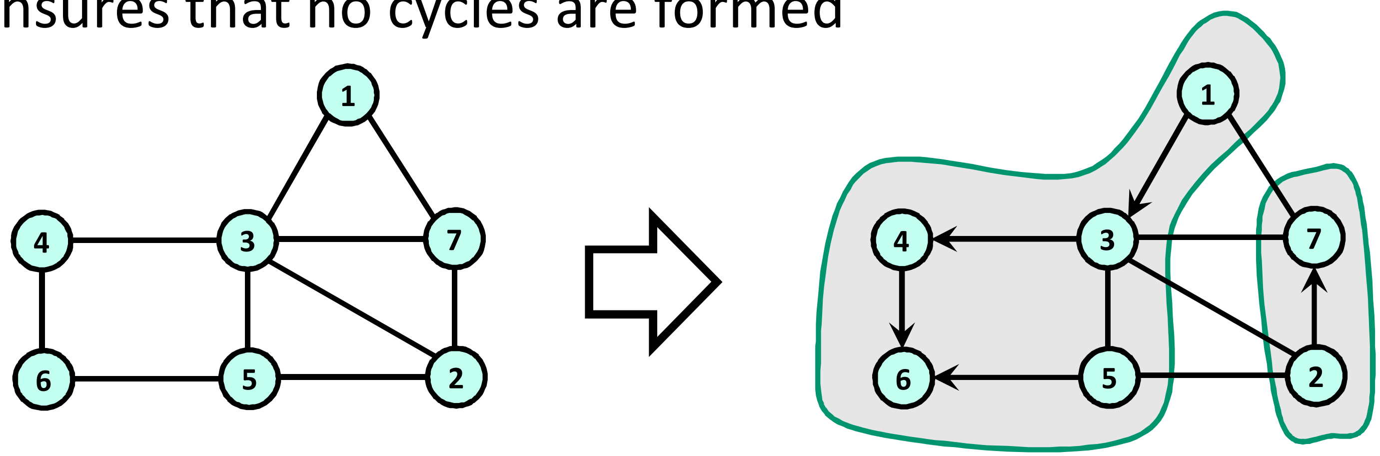
- Hook each vertex to a neighbor with larger label ( if exists )
- Ensures that no cycles are formed



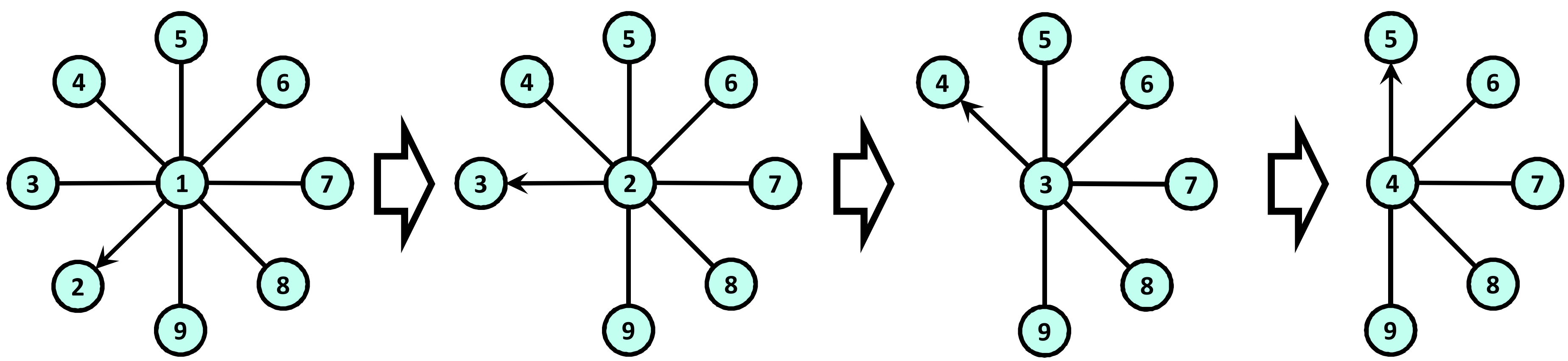
# Deterministic Parallel Connected Components (CC)

**Forming Disjoint Subtrees**

- Hook each vertex to a neighbor with larger label ( if exists )
- Ensures that no cycles are formed




- But the number of contraction steps can be as large as $n - 1$!



# Deterministic Parallel Connected Components (CC)

**Observation:**

Let $G = (V, E)$ be an undirected graph with $n$ vertices in which each vertex has at least one neighbor. Then

$$\text{either } |\{u | (u, v) \in E \wedge (u < v)\}| \geq \frac{n}{2}$$
$$\text{or } |\{u | (u, v) \in E \wedge (u > v)\}| \geq \frac{n}{2}$$

**Implication:**

Between the two directions for hooking (i.e., smaller to larger label, and larger to smaller label) always choose the one that hooks the greater number of vertices.

Then in each contraction step the number of vertices will be reduced by a factor of at least $\frac{1}{2}$.

# Deterministic Parallel Connected Components (CC)

**Input:** *n* is the number of vertices in the graph numbered from 1 to *n*, *E* is the set of edges, and *L*[ 1 : *n* ] are vertex labels with *L*[ *v* ] = *v* initially for all *v*.

**Output:** Updated array *L*[ 1: *n* ] where for all *v*, *L*[ *v* ] is the unique id of the connected component containing *v*.

```
Par-Deterministic-CC ( n, E, L )
if |E| = 0 then return L
array R[ 1 : n ], C[ 1 : n ], S[ 1 : |E| ]
parallel for v ← 1 to n do C[ v ] ← 0, R[ v ] ← ( v = L[ v ] ) ? 1 : 0
parallel for each ( u, v ) ∈ E do
    if u < v then C[ u ] ← 1
k ← Par-Sum ( C, + ), n′ ← Par-Sum ( R, + )
parallel for each ( u, v ) ∈ E do
    if k ≥ n′ / 2 and u < v then L[ u ] ← v
    else if k < n′ / 2 and u > v then L[ v ] ← u
Find-Roots ( n, L, L )
parallel for i ← 1 to |E| do S[ i ] ← ( L[ E[ i ].u ] ≠ L[ E[ i ].v ] ) ? 1 : 0
S ← Par-Prefix-Sum ( S, + )
array F[ 1 : S[ |E| ] ]
parallel for i ← 1 to |E| do
    if L[ E[ i ].u ] ≠ L[ E[ i ].v ] then F[ S[ i ] ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
L ← Par-Deterministic-CC ( n, F, L )
return L
```

count hooks from smaller to larger indices, and the number of roots

use pointer jumping to label each vertex with the id of its root

mark roots

mark hooks from smaller to larger indices

choose hook direction to maximize #hooks

similar to *Par-Randomized-CC*, except that relabeling is not needed after the recursive call

# Deterministic Parallel Connected Components (CC)

```
Par-Deterministic-CC ( n, E, L )
1.  if |E| = 0 then return L
2.  array R[ 1 : n ], C[ 1 : n ], S[ 1 : |E| ]
3.  parallel for v ← 1 to n do
        C[ v ] ← 0, R[ v ] ← ( v = L[ v ] ) ? 1 : 0
4.  parallel for each ( u, v ) ∈ E do
5.      if u < v then C[ u ] ← 1
6.  k ← Par-Sum ( C, + ), n′ ← Par-Sum ( R, + )
7.  parallel for each ( u, v ) ∈ E do
8.      if k ≥ n′ / 2 and u < v then L[ u ] ← v
9.      else if k < n′ / 2 and u > v then L[ v ] ← u
10. Find-Roots ( n, L, L )
11. parallel for i ← 1 to |E| do
        S[ i ] ← ( L[ E[ i ].u ] ≠ L[ E[ i ].v ] ) ? 1 : 0
12. S ← Par-Prefix-Sum ( S, + )
13. array F[ 1 : S[ |E| ] ]
14. parallel for i ← 1 to |E| do
15.     if L[ E[ i ].u ] ≠ L[ E[ i ].v ] then
            F[ S[ i ] ] ← ( L[ E[ i ].u ], L[ E[ i ].v ] )
16. L ← Par-Deterministic-CC ( n, F, L )
17. return L
```

Each contraction step reduces the number of vertices by a factor of at least $\frac{1}{2}$.

So, number of contraction steps, $D = \mathrm{O}(\log n)$.

For contraction step $k \geq 0$ span is $\mathrm{O}(\log^2 n)$, and work is $\mathrm{O}\left(\frac{n}{2^k}\log\frac{n}{2^k} + m\right)$. [ why? ]

**Work:** $T_1(n,m) = \mathrm{O}\left(\sum_{0 \leq i < D}\left(\frac{n}{2^k}\log\frac{n}{2^k} + m\right)\right)$

$= \mathrm{O}(n\log n + Dm)$

$= \mathrm{O}\big((n+m)\log n\big)$

**Span:** $T_\infty(n,m) = \mathrm{O}(D\log^2 n)$

$= \mathrm{O}(\log^3 n)$