Carleton UNIVERSITY

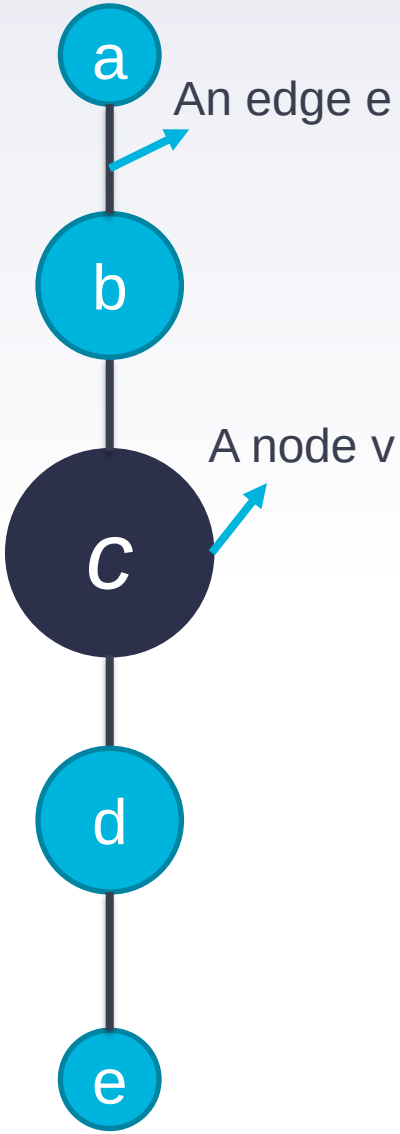
Betweenness Centrality in Dynamic Graphs

Presenter: Nathan Bowness

Betweenness Centrality (BC)

BC for a node *v*

Is the fraction of the shortest paths between all pairs of nodes that pass-through *v* [1]



BC of **c** > BC of **b, d** > BC of **a, e**

Why use Betweenness Centrality?

- Social Networks
- Transportation Networks
- Road Networks

[2]

Calculating Betweenness Centrality in Static Graphs

Formula:

$$BC_G[v] = \sum_{\substack{s,t \in V, \\ s \neq t \neq v}} \frac{\sigma_{st}(v)}{\sigma_{st}} = \sum_{\substack{s,t \in V, \\ s \neq t \neq v}} \frac{\#\ shortest\ paths\ from\ s\ to\ t\ that\ include\ v}{\#\ shortest\ paths\ from\ s\ to\ t} \qquad (1)$$

Brandes Algorithm:

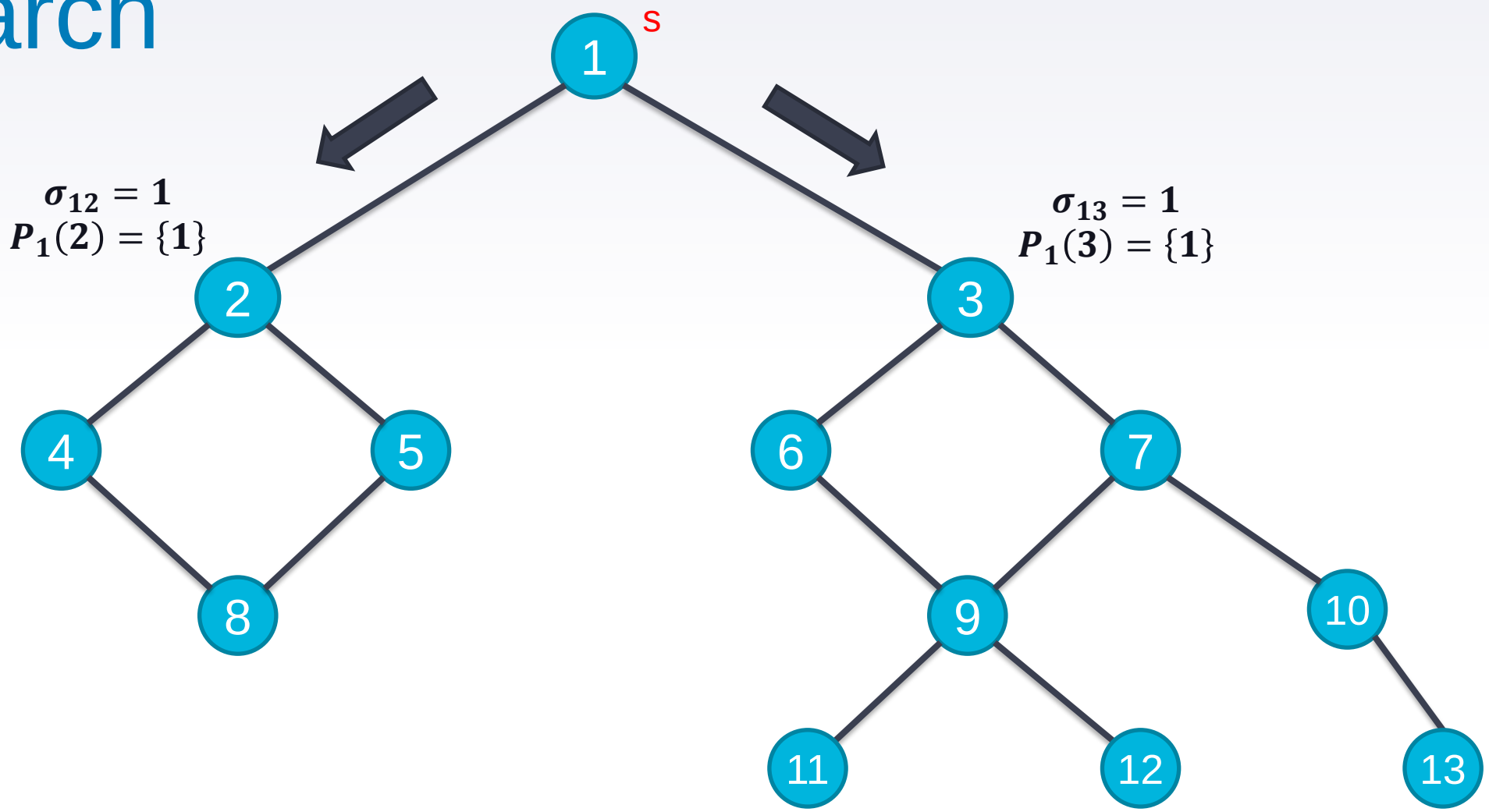
- Use source dependencies to perform calculation
  - Breadth-first search (BFS)
  - Reverse breadth-first search (R-BFS)

$$BC_G[v] = \sum_{s \in V, s \neq v} \delta_{s\bullet}(v) \qquad (2)$$

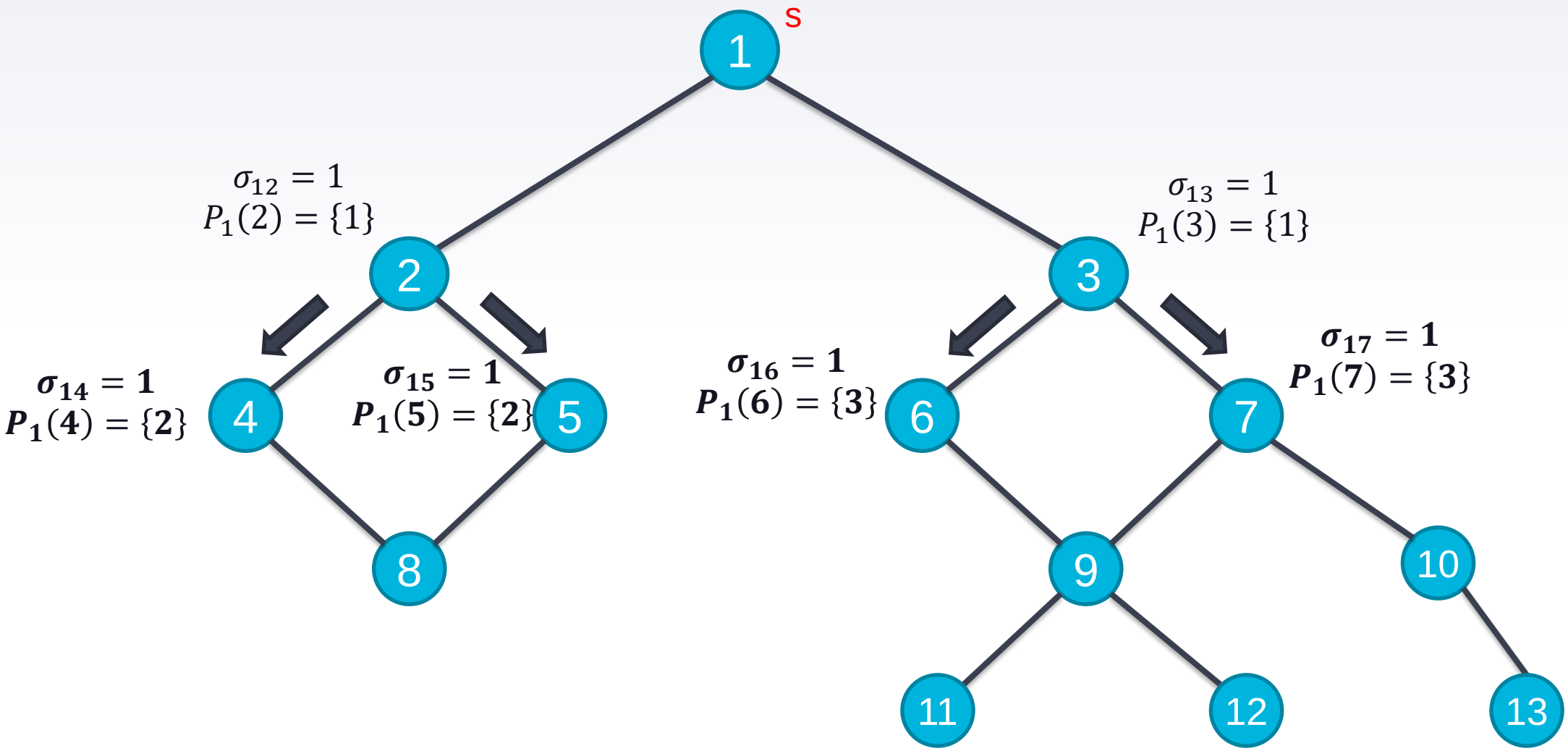
$$where, \qquad \delta_{s\bullet}(v) = \sum_{v \in P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot \left(1 + \delta_{s\bullet}(w)\right) \qquad (3)\ [3]$$

Brandes' Breadth-First Search

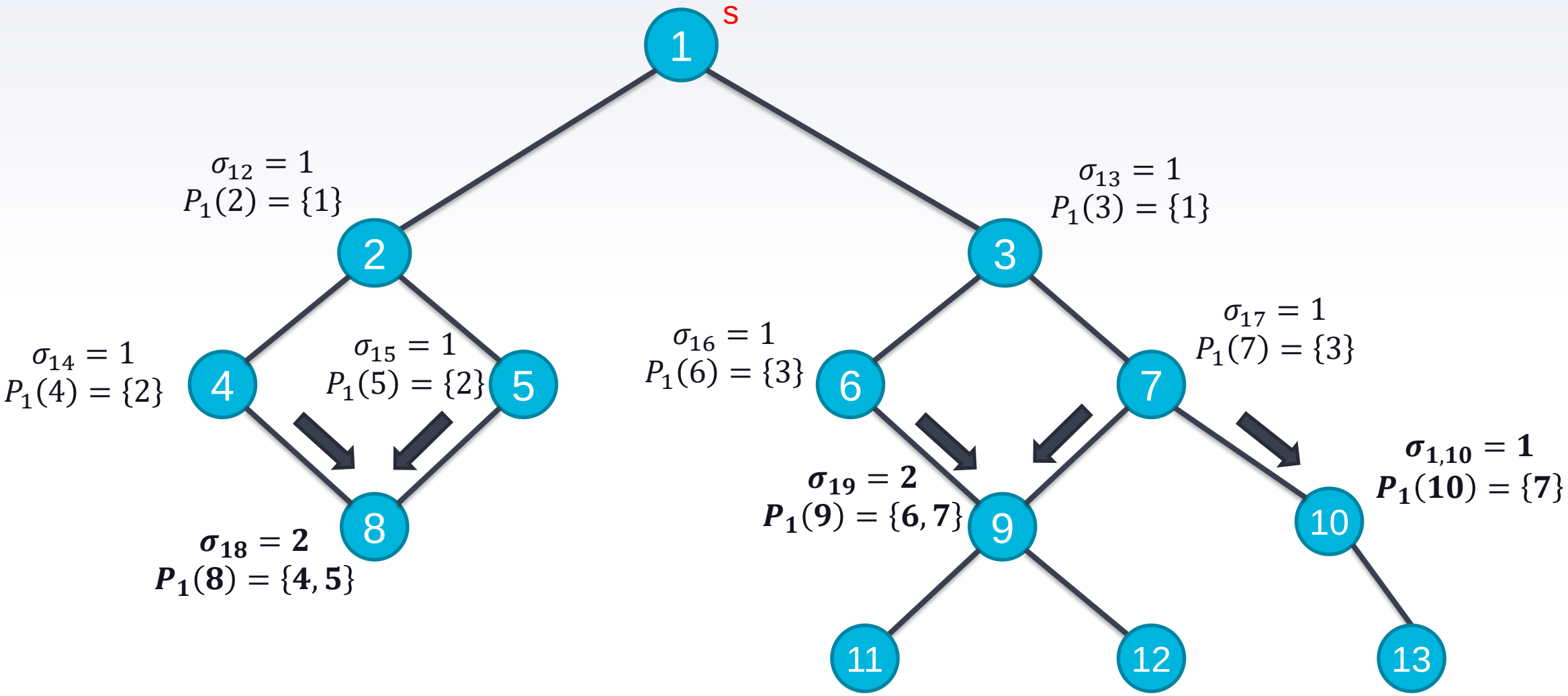
σ_{st} → # of shortest path from s to t
$P_s(t)$ → parents of a node



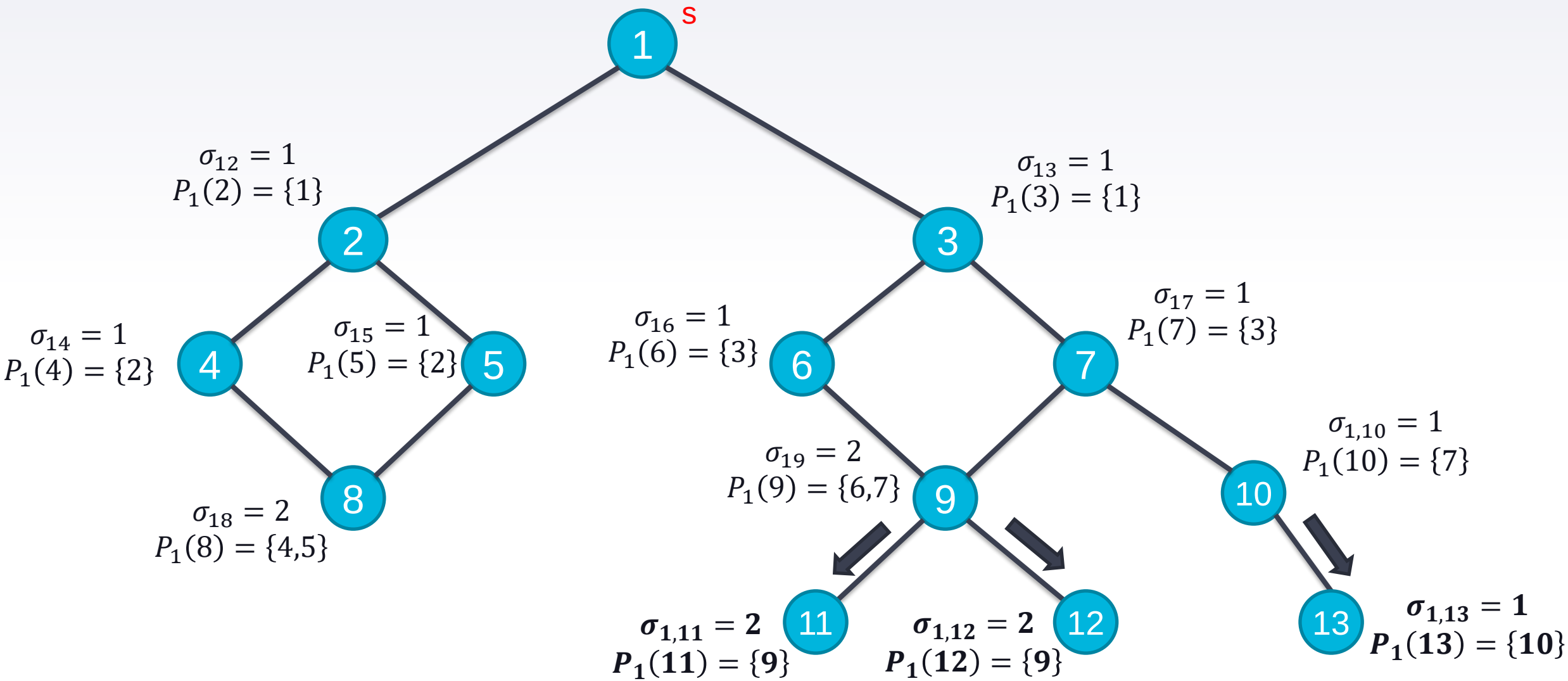
Brandes' BFS (Cont'd)



Brandes' BFS (Cont'd)

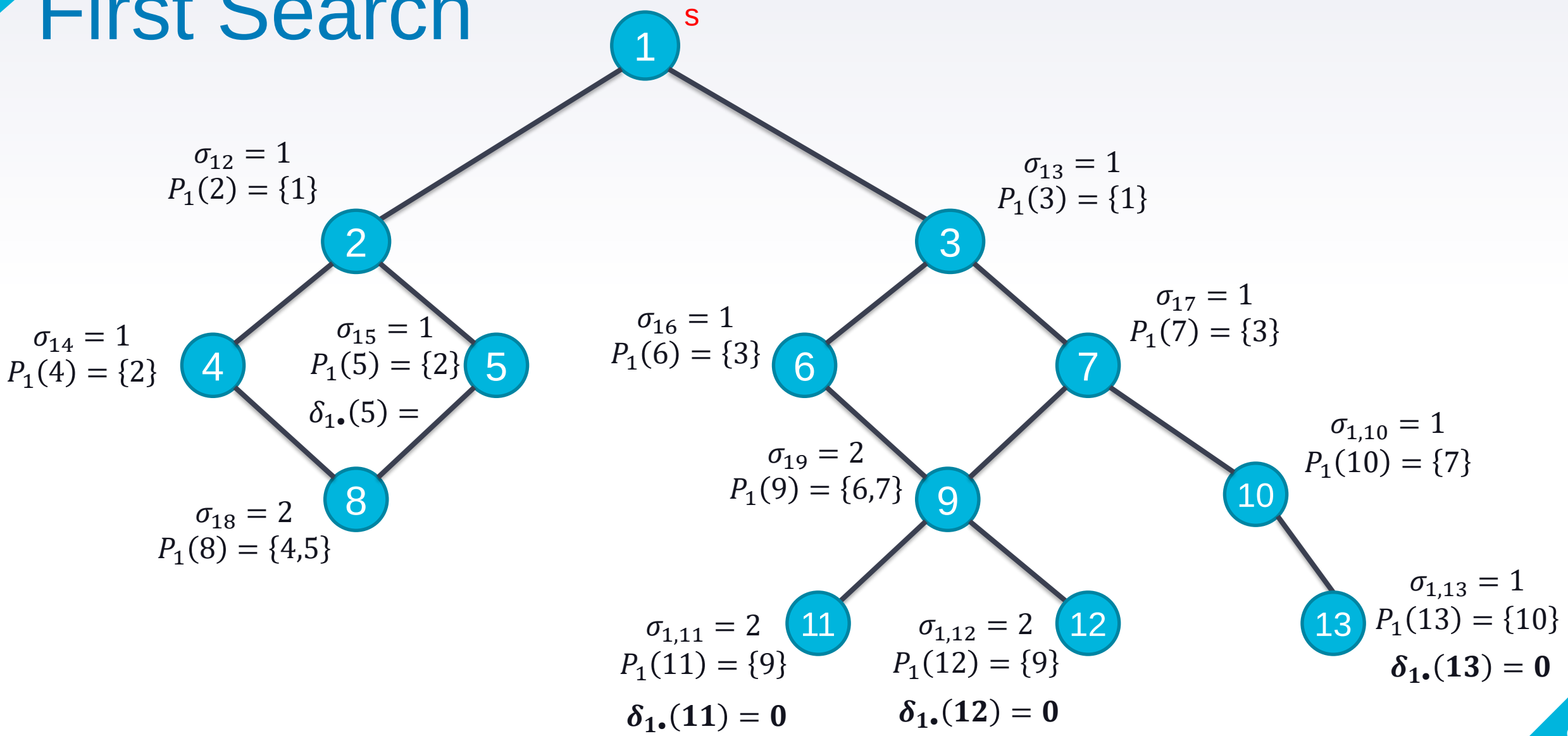


Brandes' BFS (Cont'd)

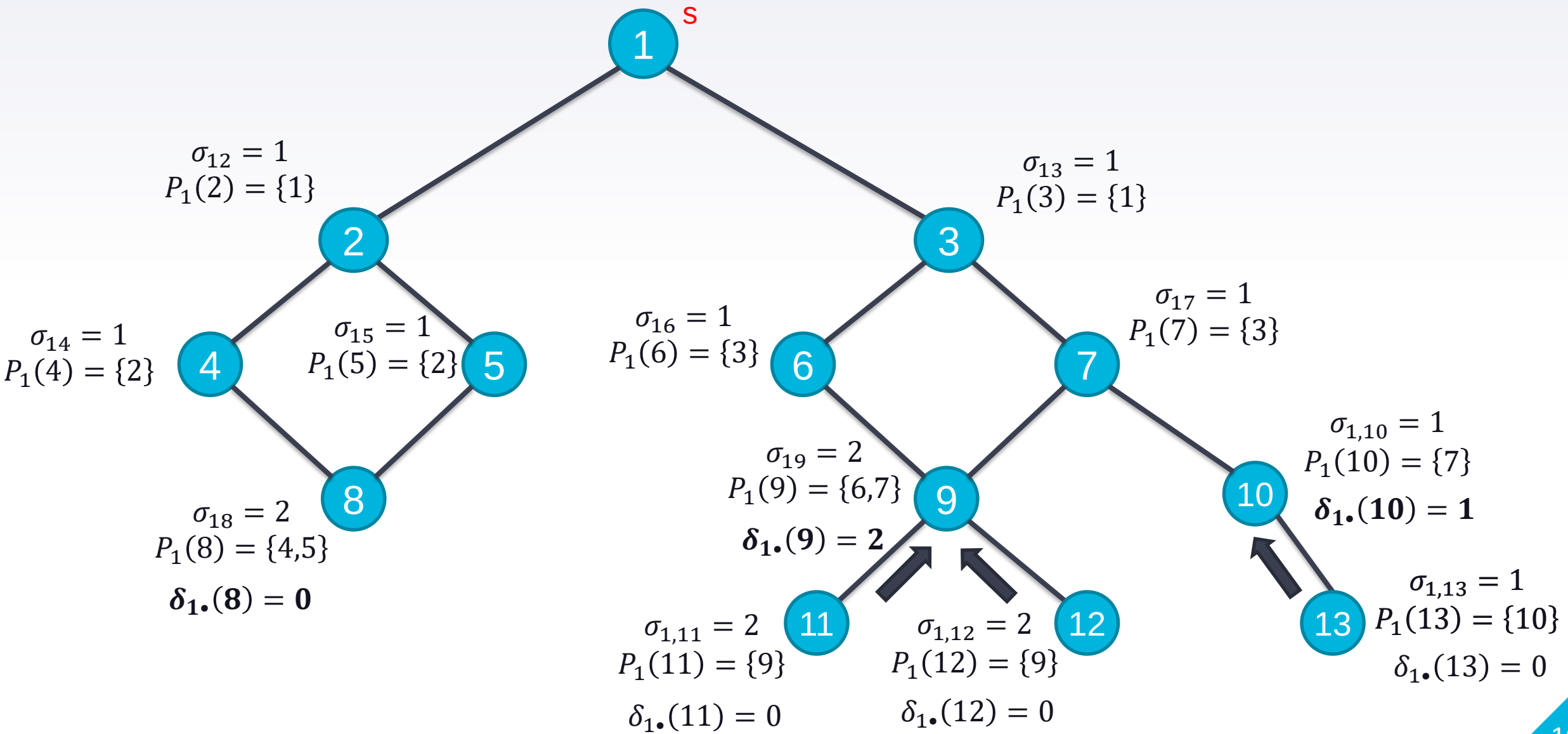


Brandes' Reverse Breadth-First Search

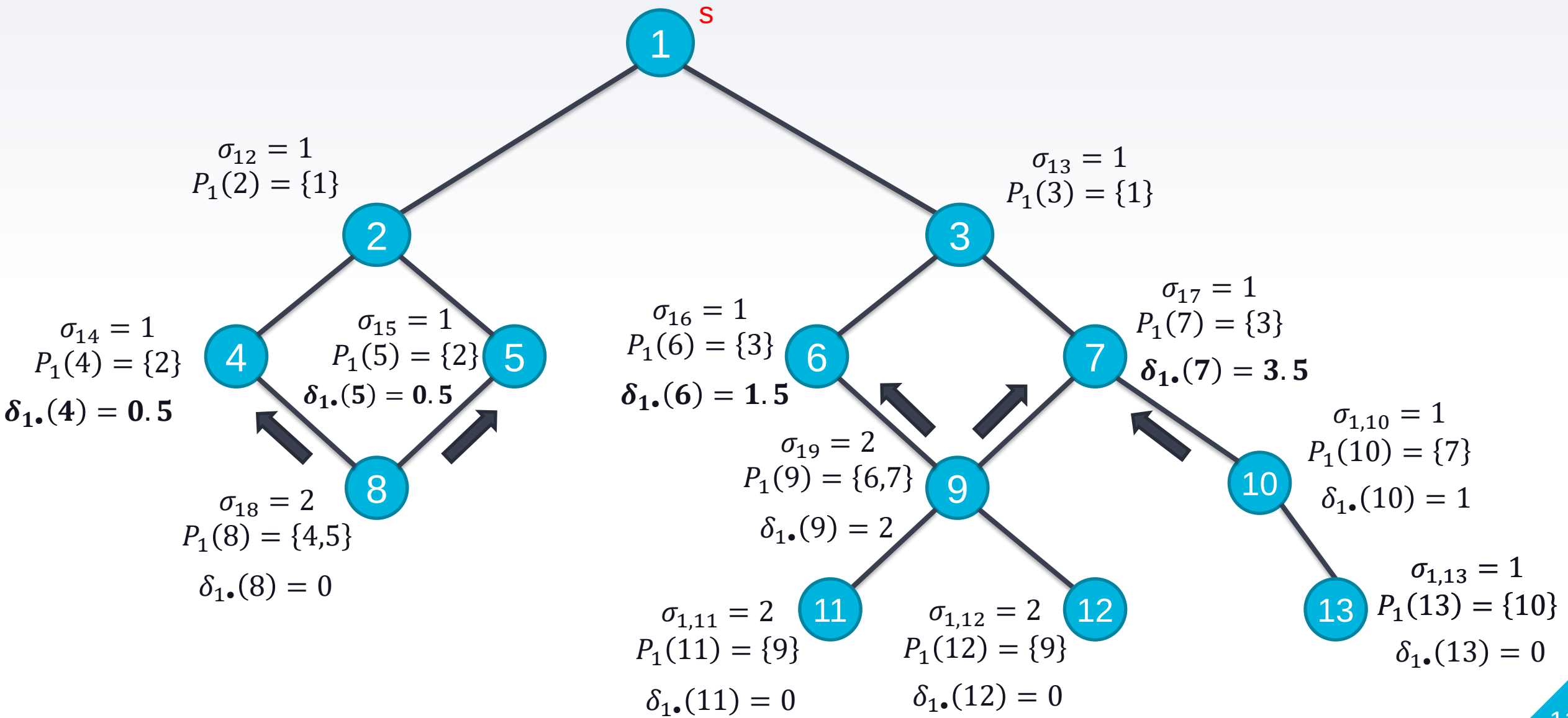
$$\delta_s\bullet(v) = \sum_{v \epsilon P_s(w)} \frac{\sigma_{sv}}{\sigma_{sw}} \cdot (1 + \delta_s\bullet(w))$$



Brandes' R-BFS (Cont'd)

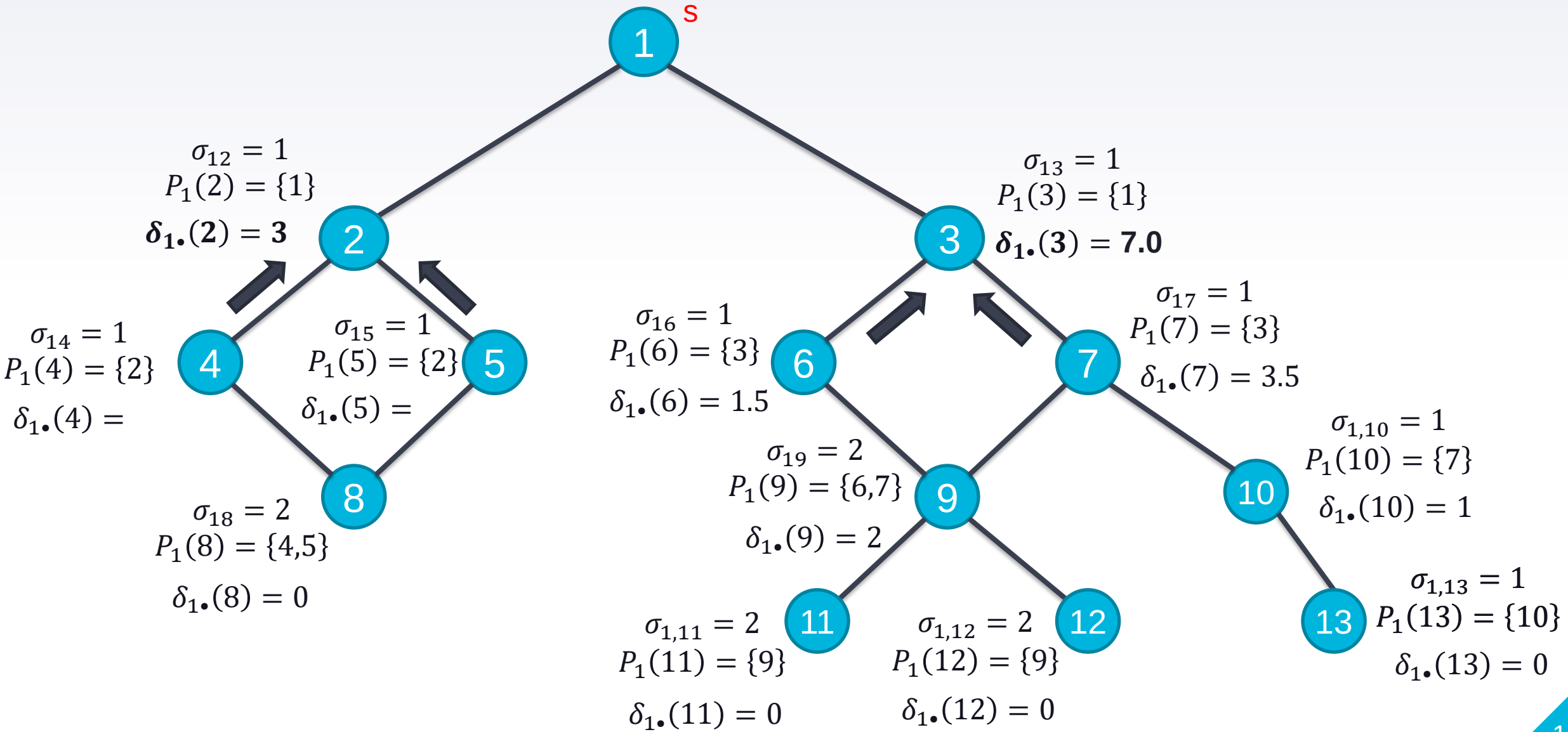


Brandes' R-BFS (Cont'd)



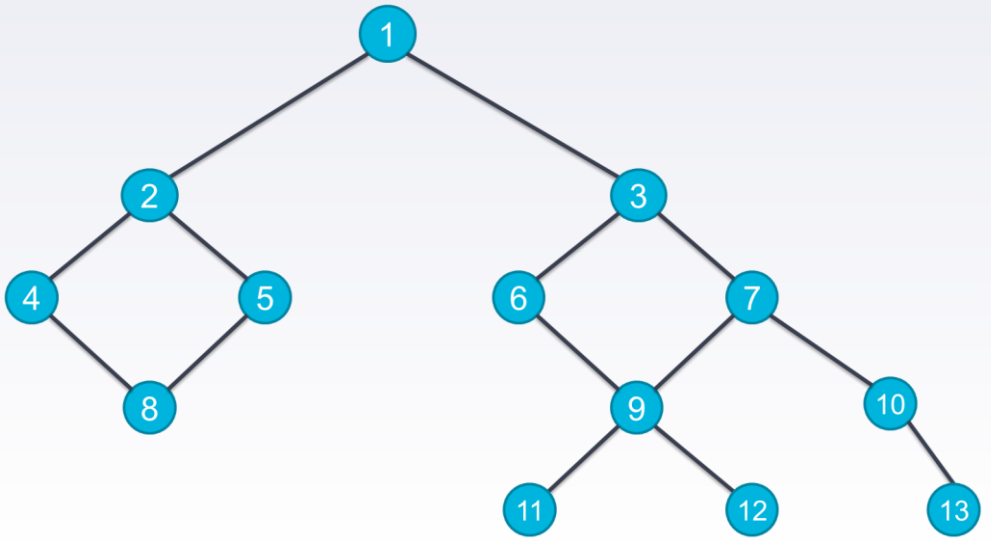
Brandes' R-BFS (Cont'd)

$$BC_G[v] = \sum_{s \in V, s \neq v} \delta_{s\bullet}(v)$$



Calculating Betweenness Centrality

$$Betweeenness\ Centrality \rightarrow BC_G[v] = \sum_{s \epsilon V, s \neq v} \delta_{s\bullet}(v)$$



- **Completed** → s = 1
- **Remaining** → s = 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13

- Maintains data for
  - σ_{st}
  - P_S
  - S_S
  - D_S

Betweenness Centrality in Dynamic Graphs

- Social Networks
- Transportation Networks
- Road Networks

Networks are always changing

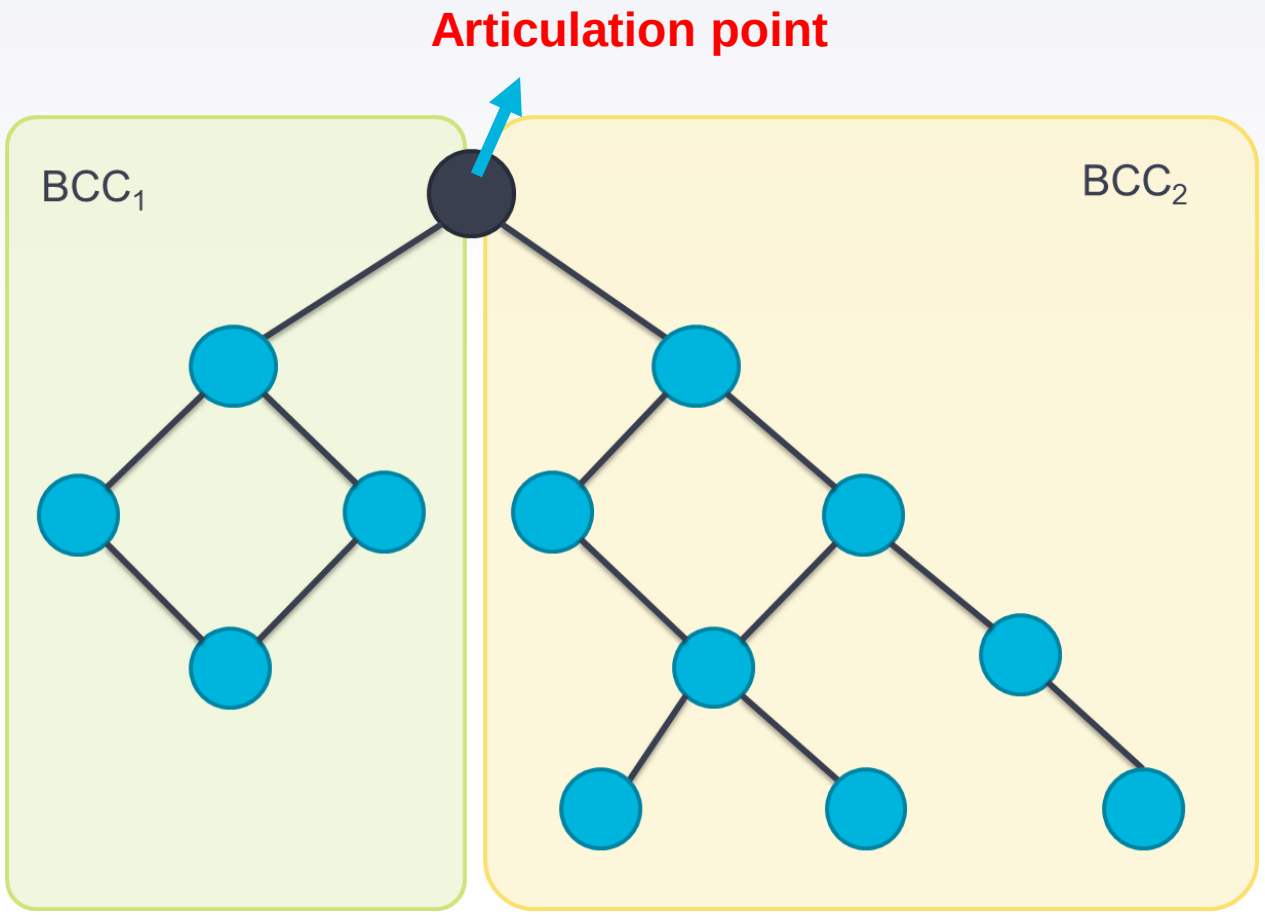
Calculating BC in Dynamic Graphs - 2020

- Recent algorithm in 2020 → **Batch iCentral** by Shukla et Al. [4],[5]
- Key Concepts:
  - Avoid BFS
  - Recompute betweenness centrality after a batch of updates
  - Leverage previous stored data from Brandes'

Biconnected Components (BCC)

- are a “maximal biconnected subgraph” [5]
- Connected only by **articulation points**

- Allow graph to be split into subsections

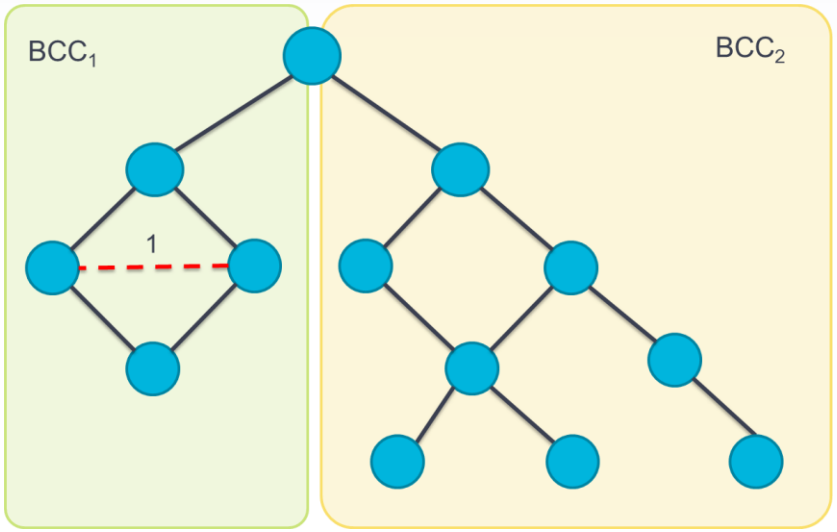


Updating Edges in Sequence

- Previous algorithms applied edges 1-by-1 for Dynamic Graphs

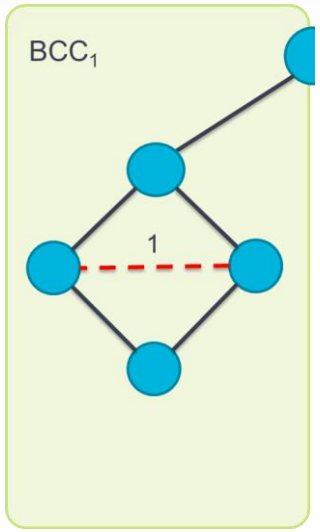
Step 1.

Add Edge



Step 2.

Find affected BCC

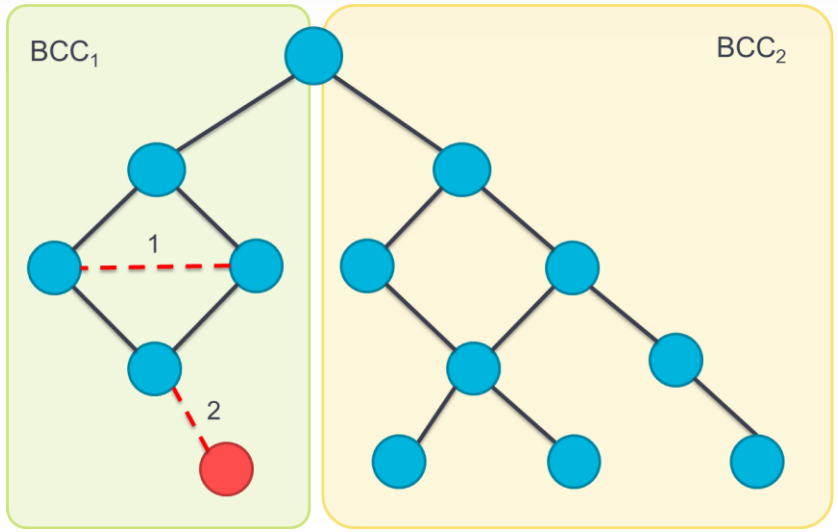


Step 3.

Recompute Betweenness Centrality

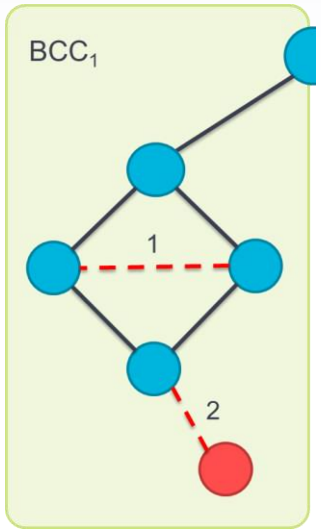
Step 4.

Add Edge



Step 5.

Find affected BCC



Step 6…. And so on

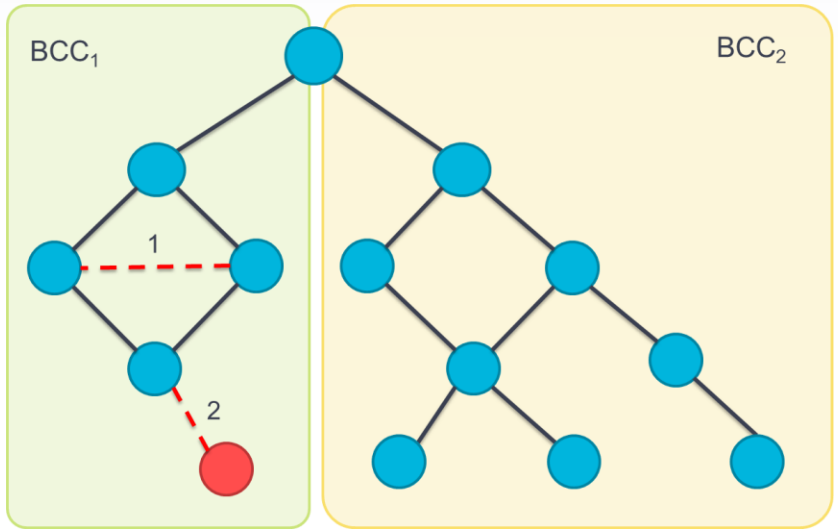
Recompute Betweenness Centrality

Updating Edges in Batches

- Newer algorithms apply all edges to start, then recompute BC

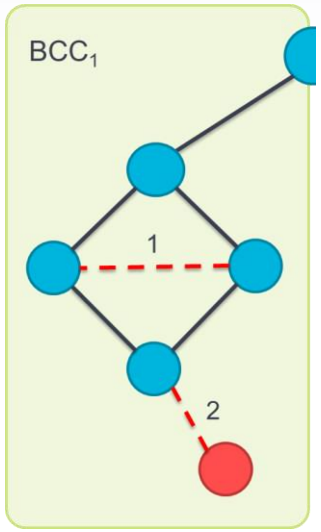
Step 1.

Add All Edges



Step 2.

Find any affected BCCs



Step 3. Done.

Recompute BC on affected nodes

Parallelizing BC Calculation

[6]

- Few ways to parallelize
  - On affected Biconnected Components
  - On affected nodes
  - (Rare) On Graphs

Parallelizing BC Calculation

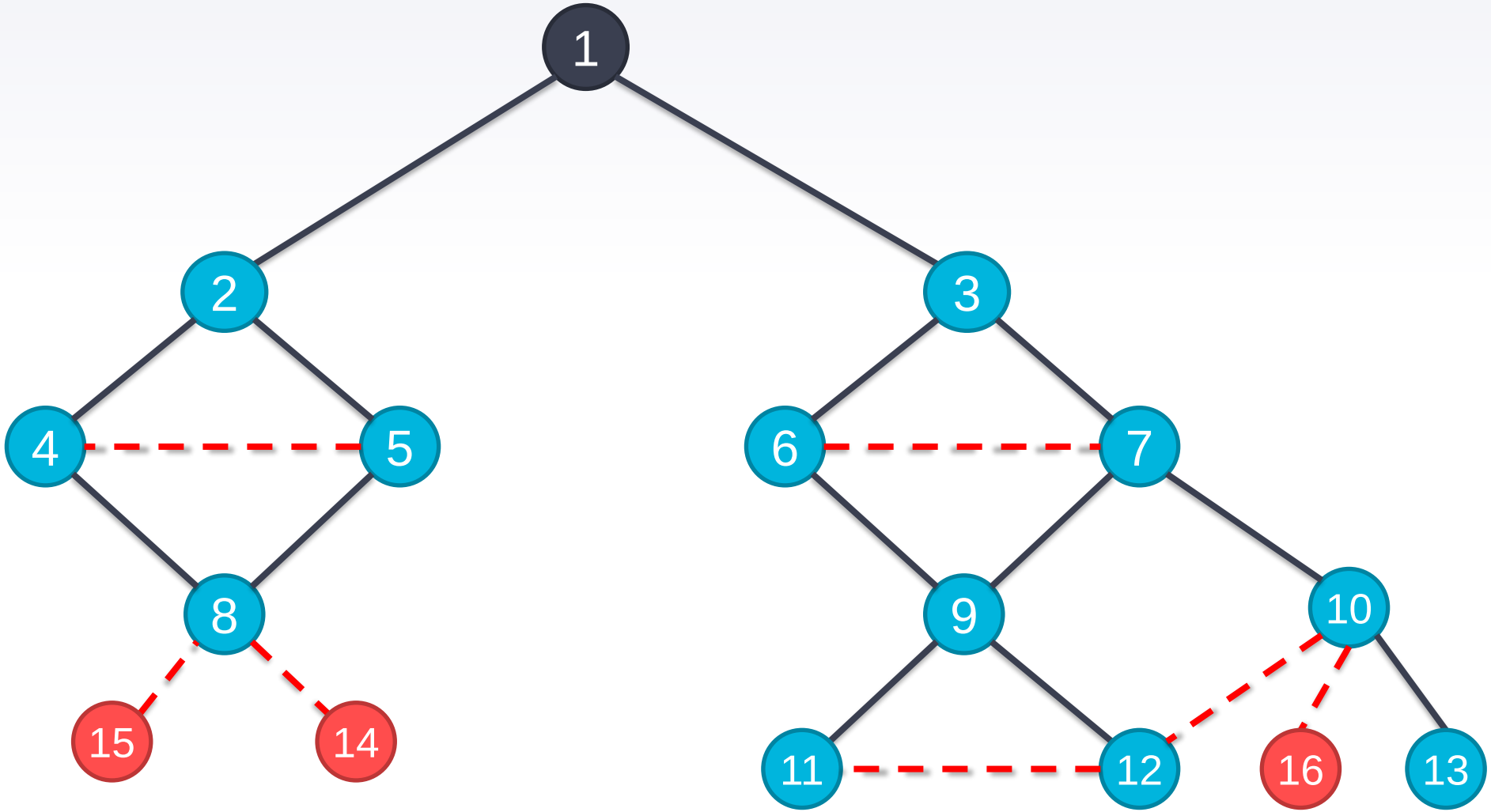
- Few ways to parallelize
  - On affected Biconnected Components
  - **On affected nodes ←**
  - (Rare) On Graphs

Node *v* is affected if:

$$distance(v, s) \neq distance(v, t)$$

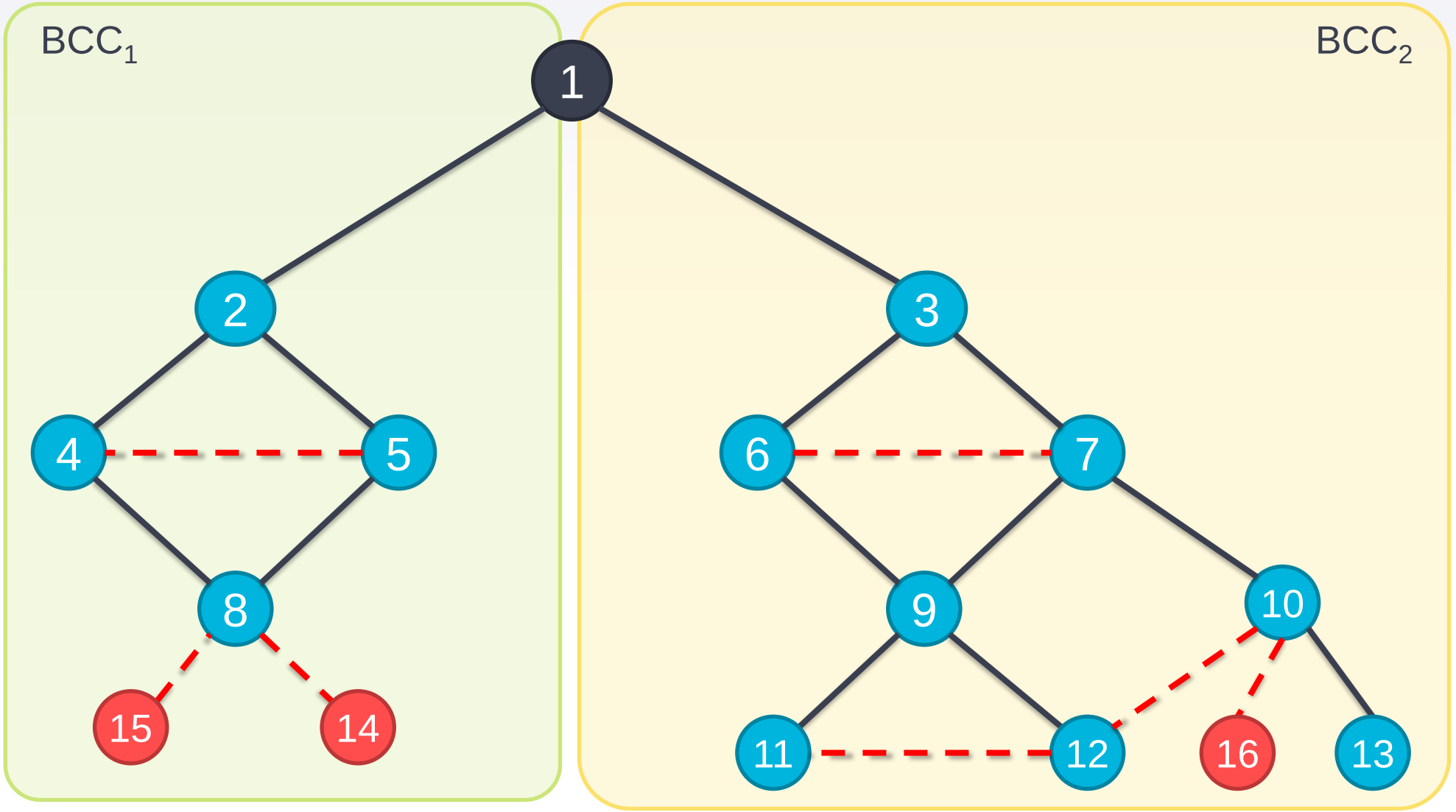
Parallelizing BC: Example

- Add 7 edges to the graph



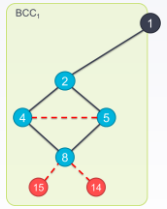
Parallelizing BC: Example (Cont'd)

- Identify affected Biconnected Components

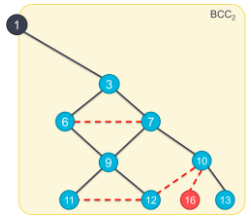


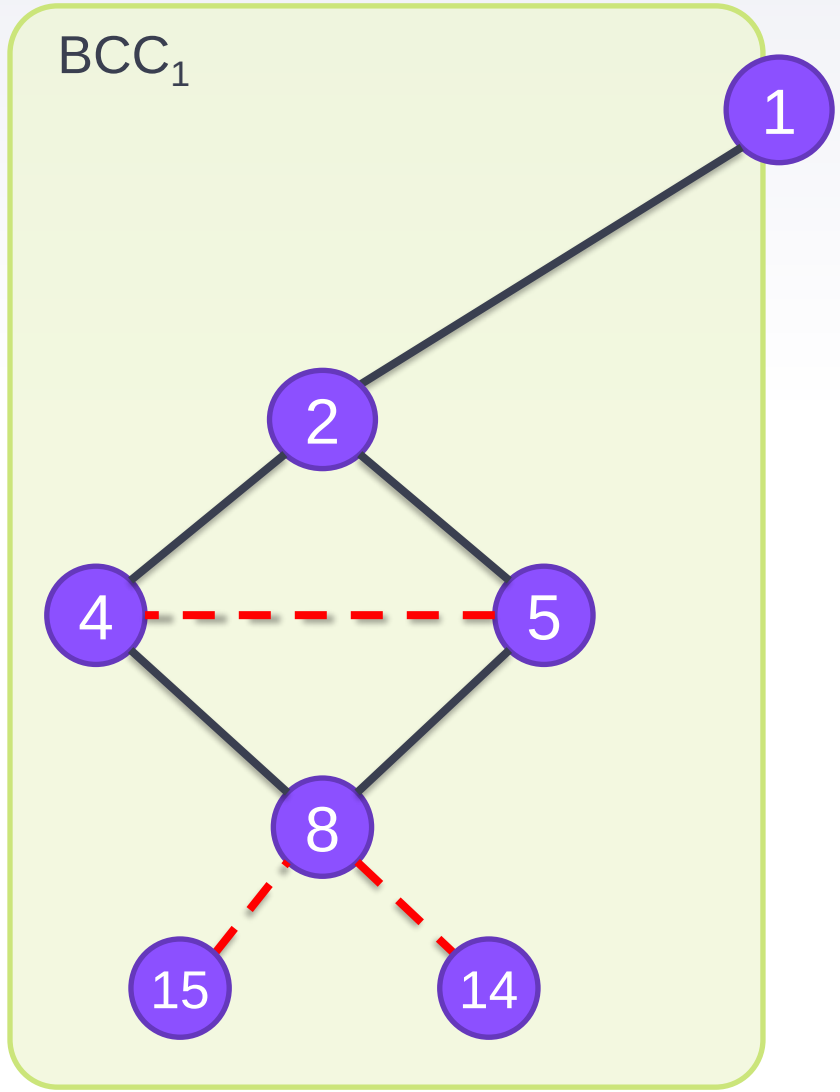
Parallelizing BC: Example (Cont'd)

- Identify affected nodes for each BCC

1st 

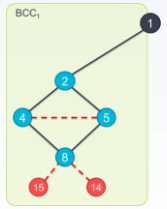
All affected!

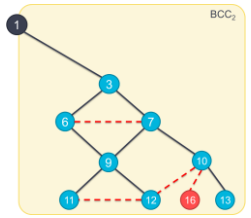
2nd 



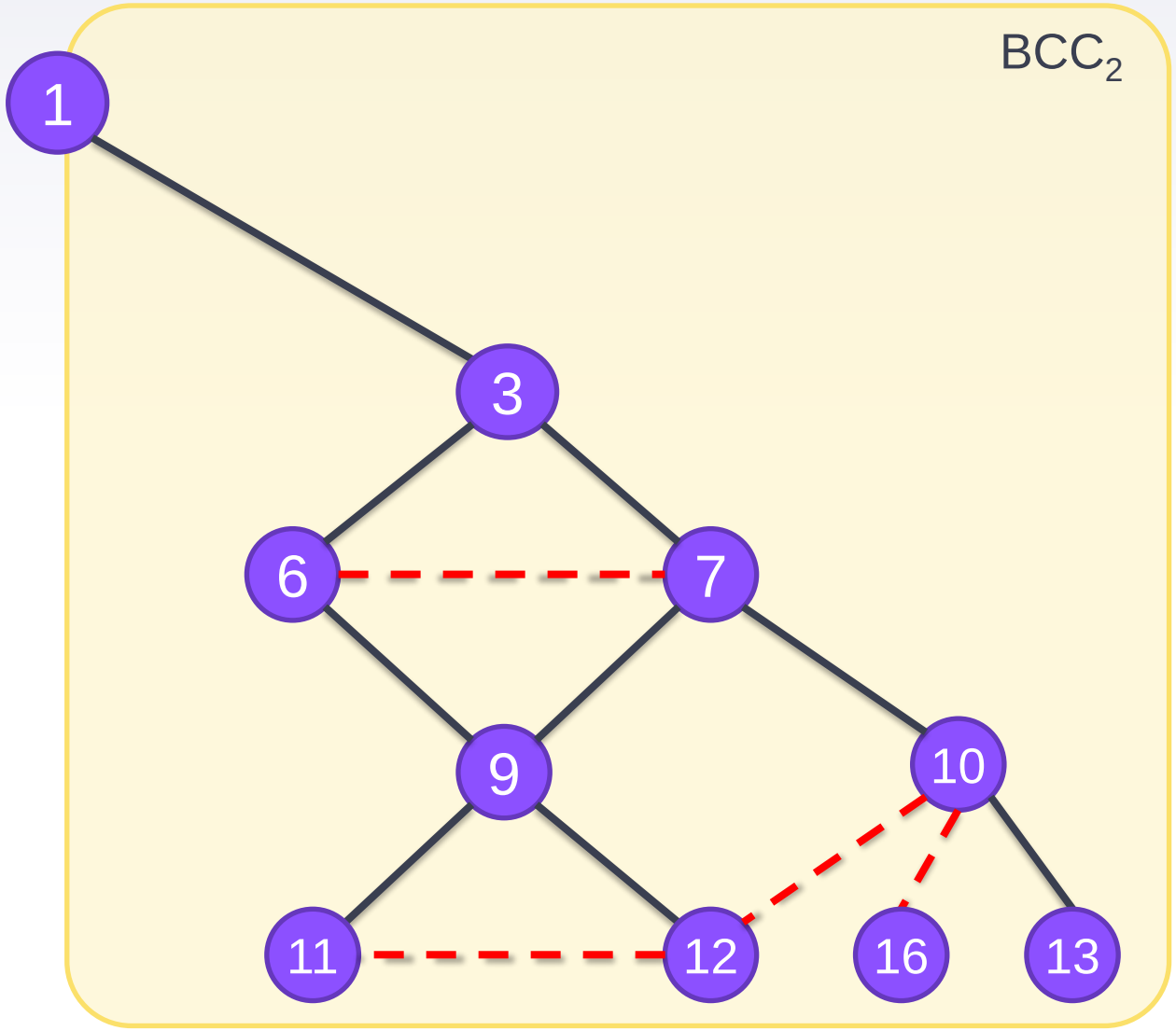
Parallelizing BC: Example (Cont'd)

- Identify affected nodes for each BCC

1st 

2nd 

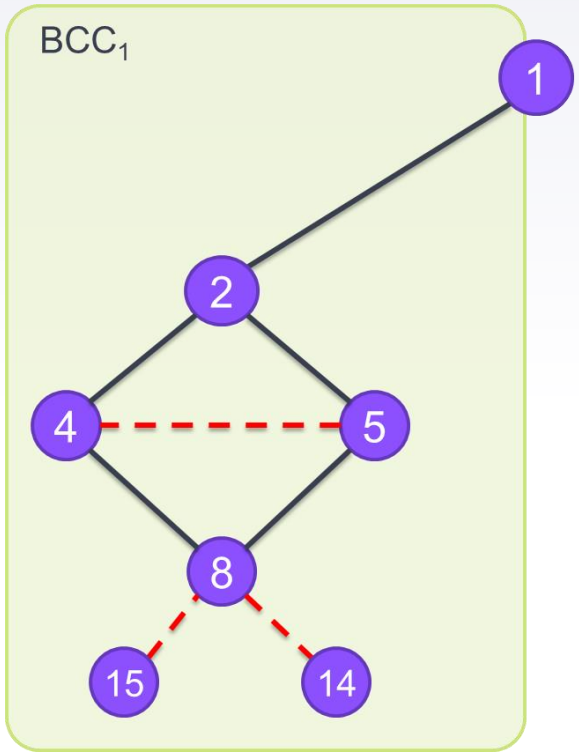
All affected!



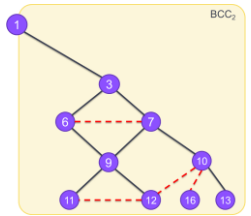
Parallelizing BC: Example (Cont'd)

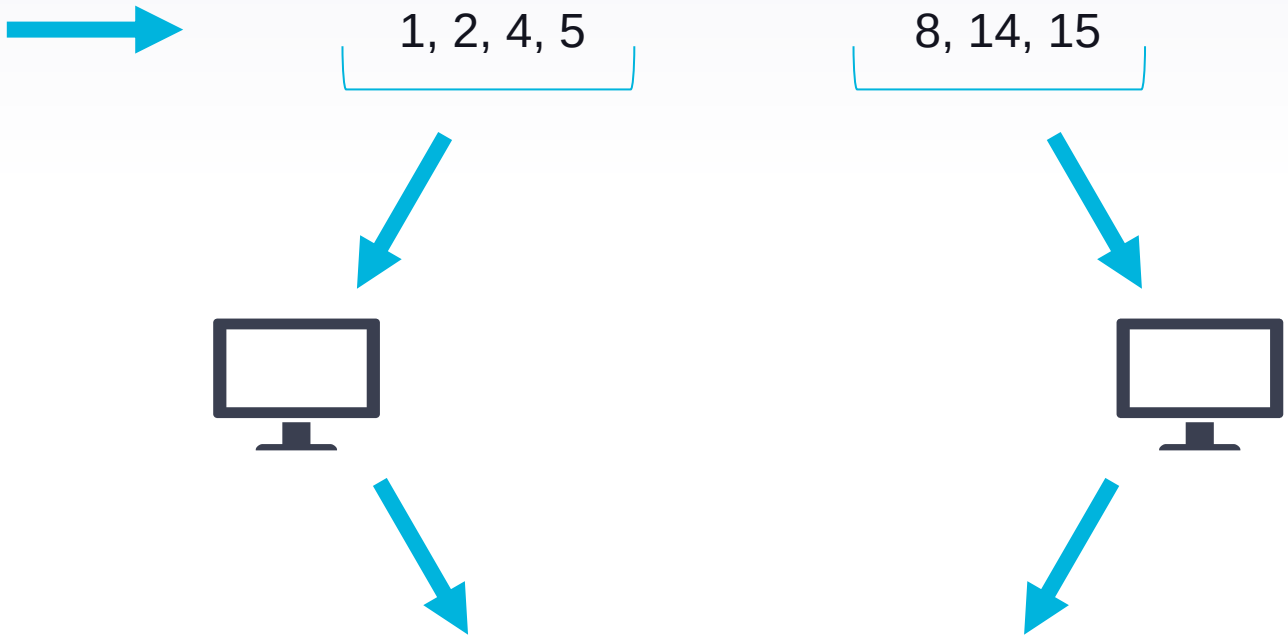
- Send groups of affected nodes to separate threads/machines

1st



2nd



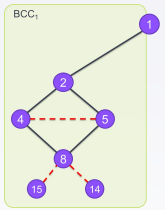


Arrays of source dependencies ($\delta_{s\bullet}$) values

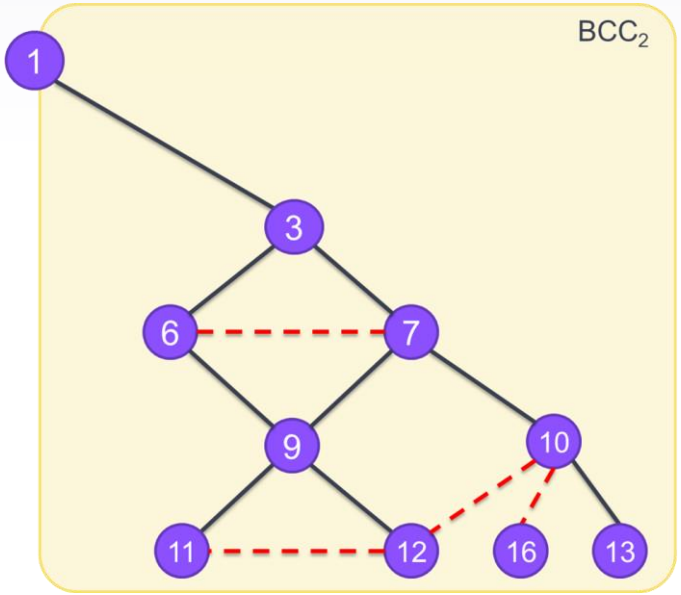
Parallelizing BC: Example (Cont'd)

- Send groups of affected nodes to separate threads/machines

1st



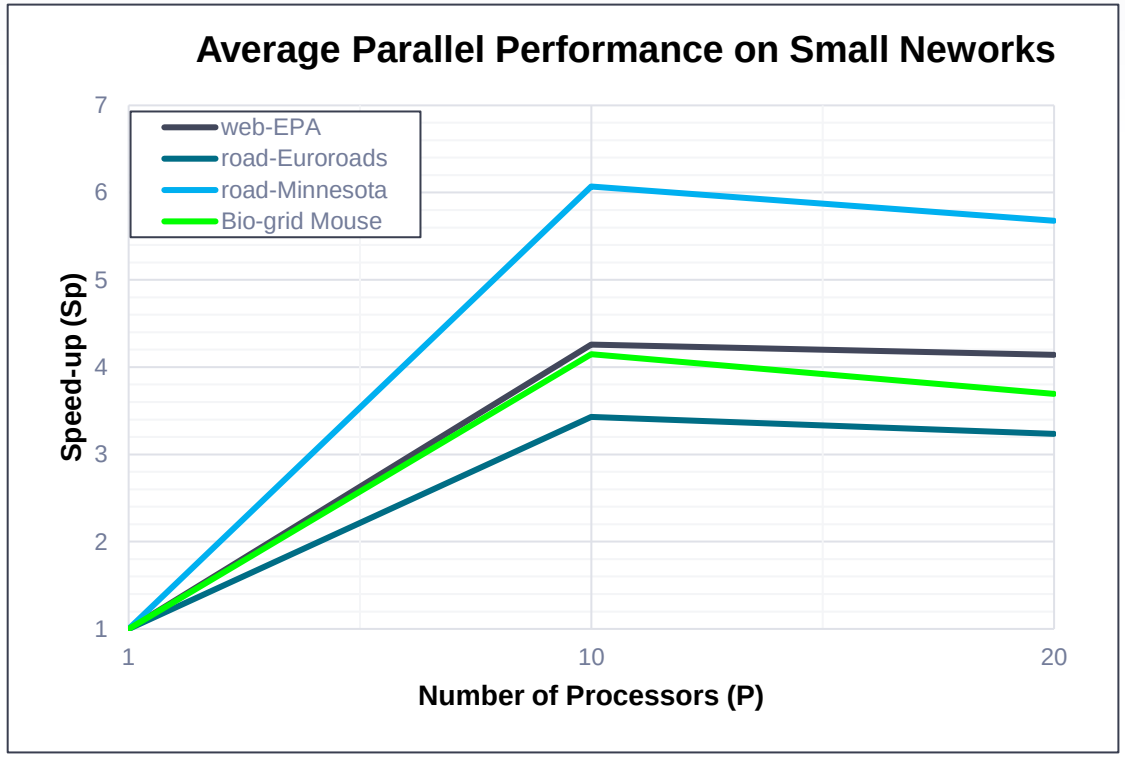
2nd

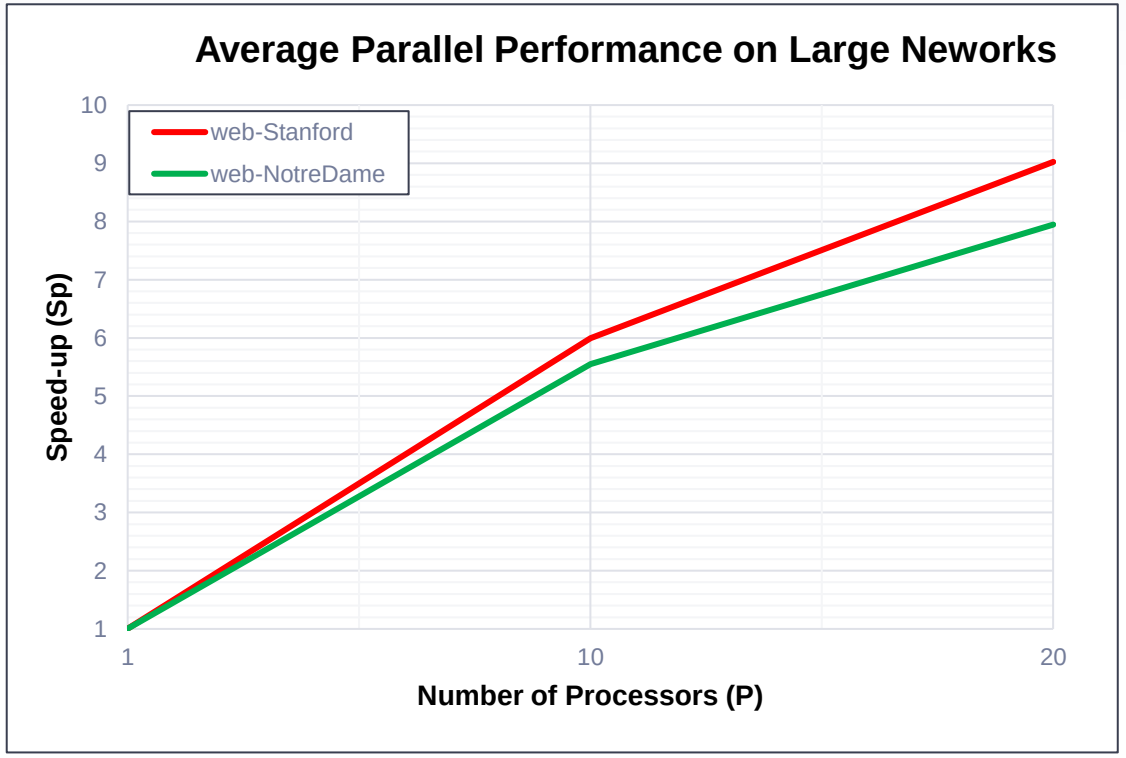


Arrays of source dependencies ($\delta_{s\bullet}$) values

Parallel Performance

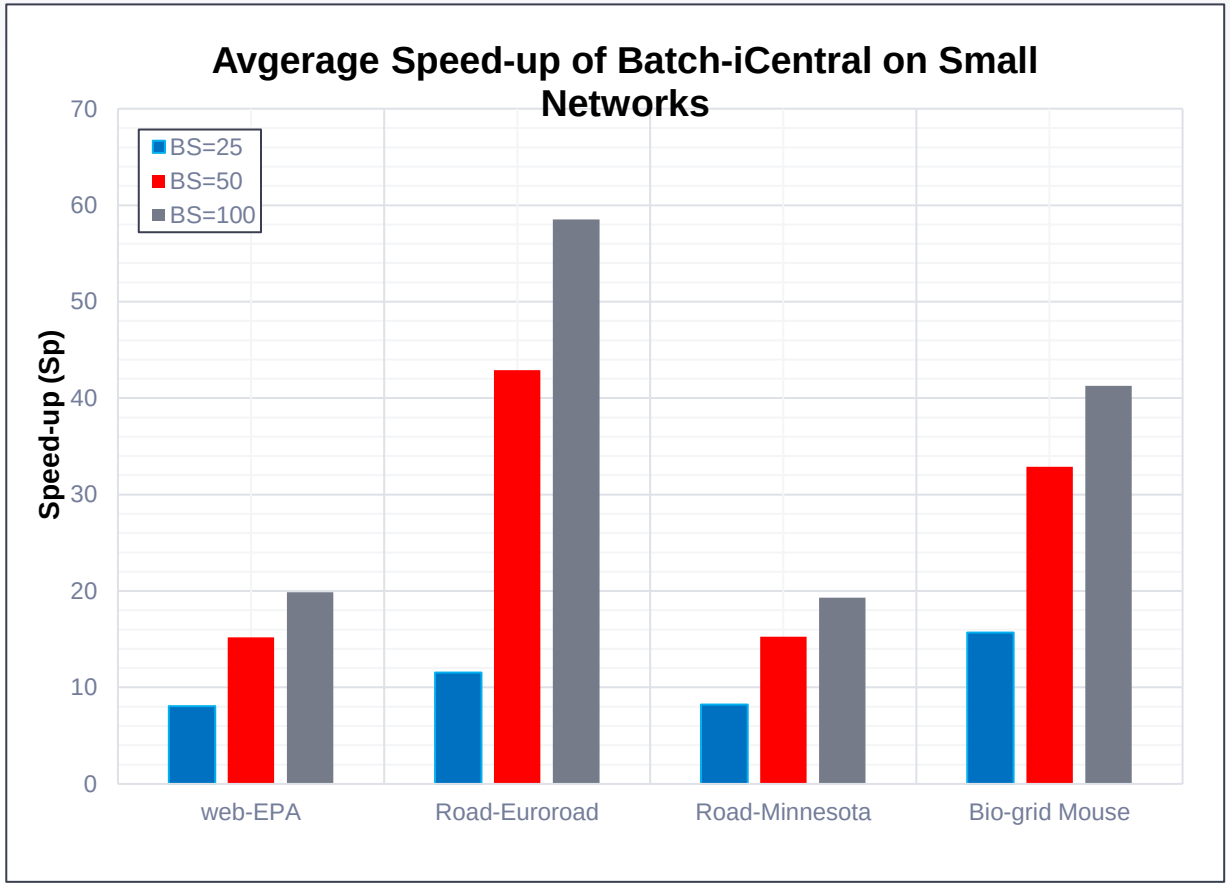
- Non-linear speed-up
- Better parallel performance on larger graphs – to be expected

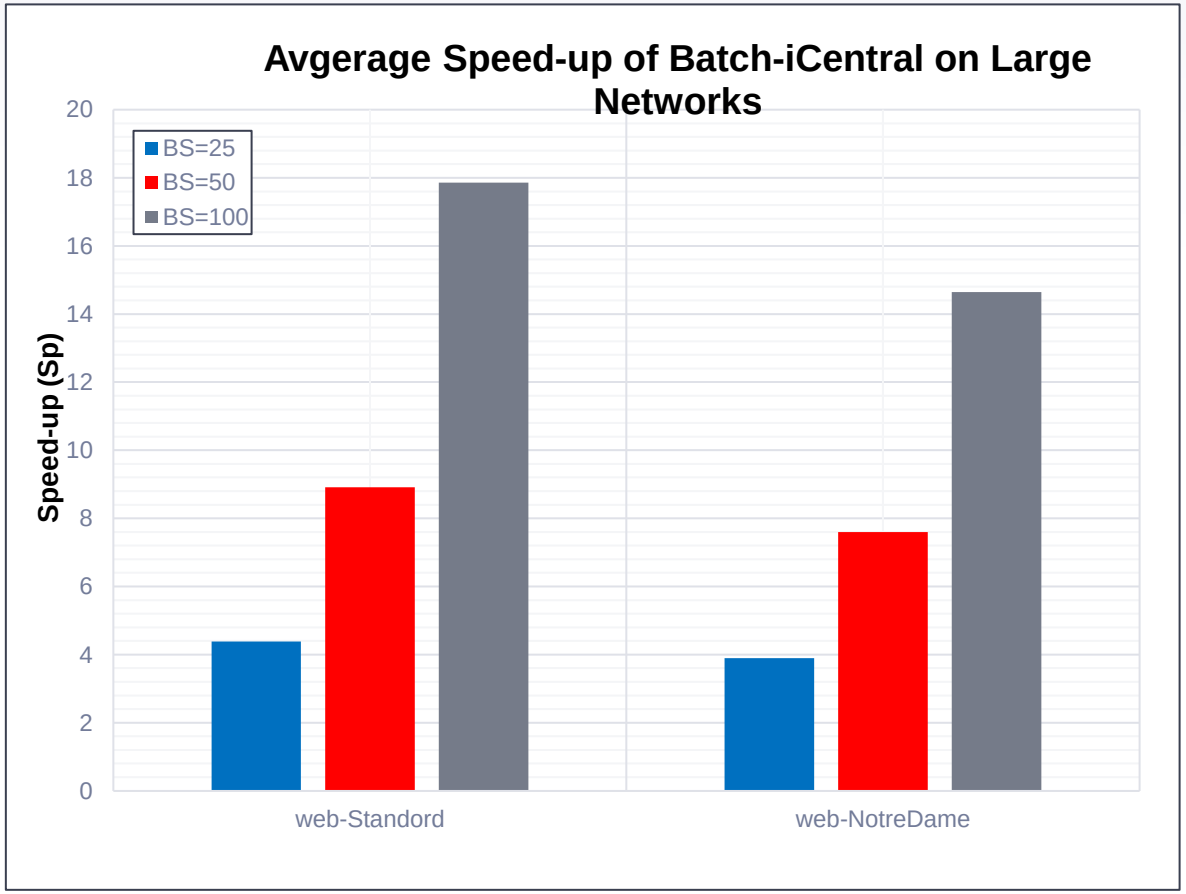




Comparison of Results

- Batch iCentral vastly out-performs regular iCentral





Thanks For Listening!

Any questions?

Question 1:

- **What type of tree traversal is used when calculating betweenness centrality?**

Question 2:

- **When recalculating BC in a dynamic graph, is it more effective to process edges one-by-one or in a batch?**

Question 3:

- **What is one section of the graph that betweenness centrality can be parallelized on?**

References:

- [1] L. C. Freeman, “A Set of Measures of Centrality Based on Betweenness,” *Sociometry*, vol. 40, no. 1, pp. 35–41, 1977, doi: 10.2307/3033543.
- [2] M. Grandjean, *English: Graph representing the metadata of thousands of archive documents, documenting the social network of hundreds of League of Nations personals.* 2013.
- [3] U. Brandes, “A faster algorithm for betweenness centrality,” *The Journal of Mathematical Sociology*, vol. 25, no. 2, pp. 163–177, Jun. 2001, doi: 10.1080/0022250X.2001.9990249.
- [4] K. Shukla, S. C. Regunta, S. H. Tondomker, and K. Kothapalli, “Efficient parallel algorithms for betweenness- and closeness-centrality in dynamic graphs,” in *Proceedings of the 34th ACM International Conference on Supercomputing*, New York, NY, USA, Jun. 2020, pp. 1–12, doi: 10.1145/3392717.3392743.
- [5] F. Jamour, S. Skiadopoulos, and P. Kalnis, “Parallel Algorithm for Incremental Betweenness Centrality on Large Graphs,” *IEEE Transactions on Parallel and Distributed Systems*, vol. 29, no. 3, pp. 659–672, Mar. 2018, doi: 10.1109/TPDS.2017.2763951.
- [6] “open-mpi/ompi: Open MPI main development repository.” https://github.com/open-mpi/ompi (accessed Dec. 07, 2020).