Δ-Stepping: A Parallelizable Shortest Path Algorithm

U. Meyer, P. Sanders

Presented by Helen He

Parallel Single Source Shortest Path (SSSP)

- Large graphs need good parallel algorithms
- Parallel SSSP are a bottleneck
- Lots of sequential SSSP with poor worst-case bounds perform well practically

SSSP Basics

- “Relaxing” – update distance label if route through another vertex is shorter
- Label-setting algorithms (e.g. Dijkstra)
- Label-correcting algorithms (e.g. Bellman-Ford)
- Label setting has better worst-case bounds, but label-correcting is often better in practice

Dijkstra’s Overview

- Set source distance to 0, all others at infinity
- Consider all the current node’s neighbors, relax outgoing edges
- Mark the current node as visited, never visit it again
- If the destination node hasn’t been found, continue with the unvisited node with the smallest tentative distance
- Bucket implementation visits multiple nodes at once based off their tentative distances

Δ-Stepping

- Buckets of vertices grouped by their temporary distance labels
- B[i] contains vertices with labels in [i*Δ, (i+1)*Δ]
- Can reuse empty buckets to save space
- Outer loop proceeds through the buckets
- Inner loop processes the bucket until it’s empty

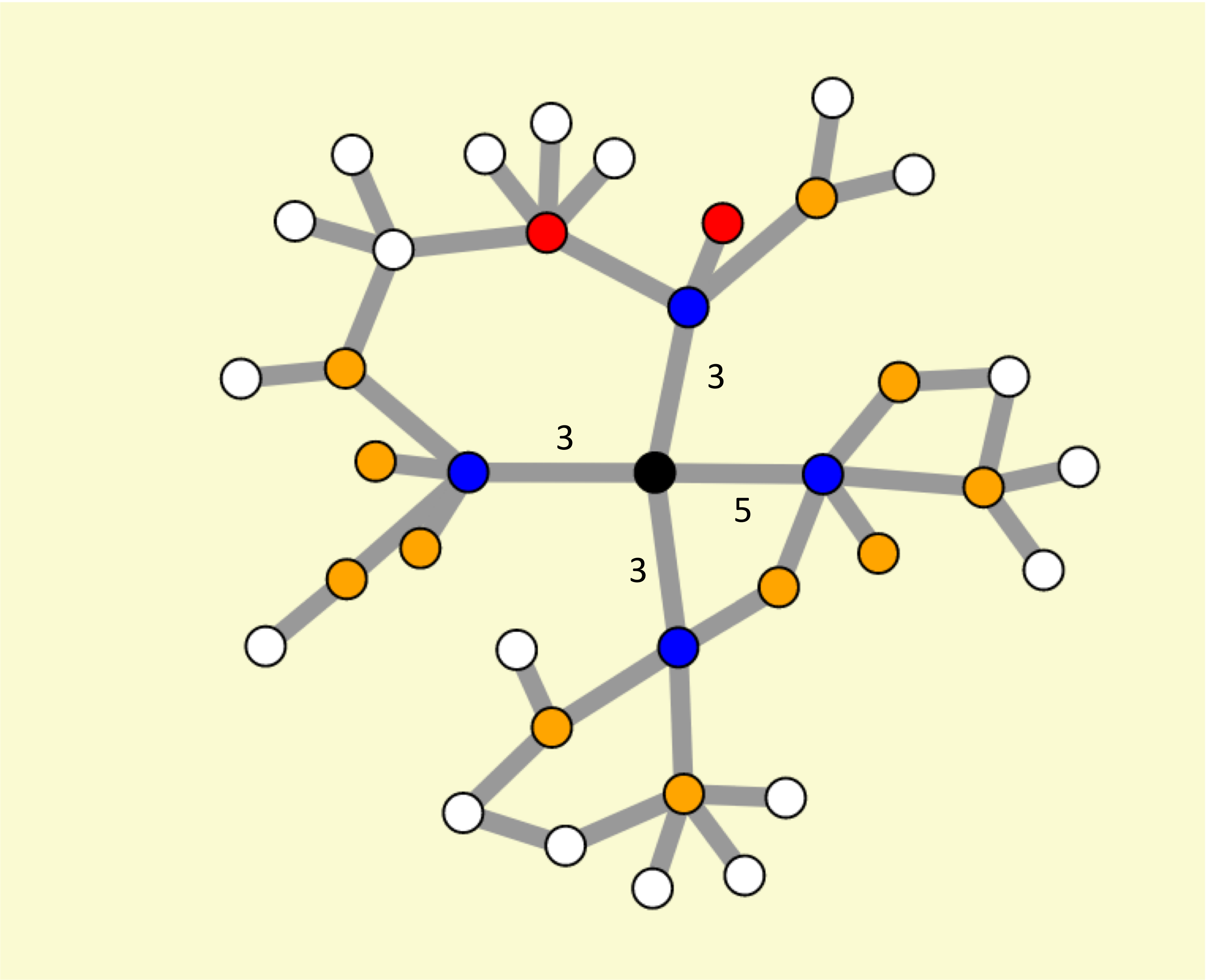
Bucket Processing

- Each vertex in the bucket has outgoing edges which are either “light” (weight $\leq \Delta$) or “heavy” (weight $> \Delta$)
- When a bucket is processed, it is first emptied
- All light edges are relaxed
- Relaxing an edge can cause the destination vertex to be inserted into the current bucket
- Process bucket until it is empty, then relax its heavy edges

Δ=3, B[1] = [3, 6)

Phase 1

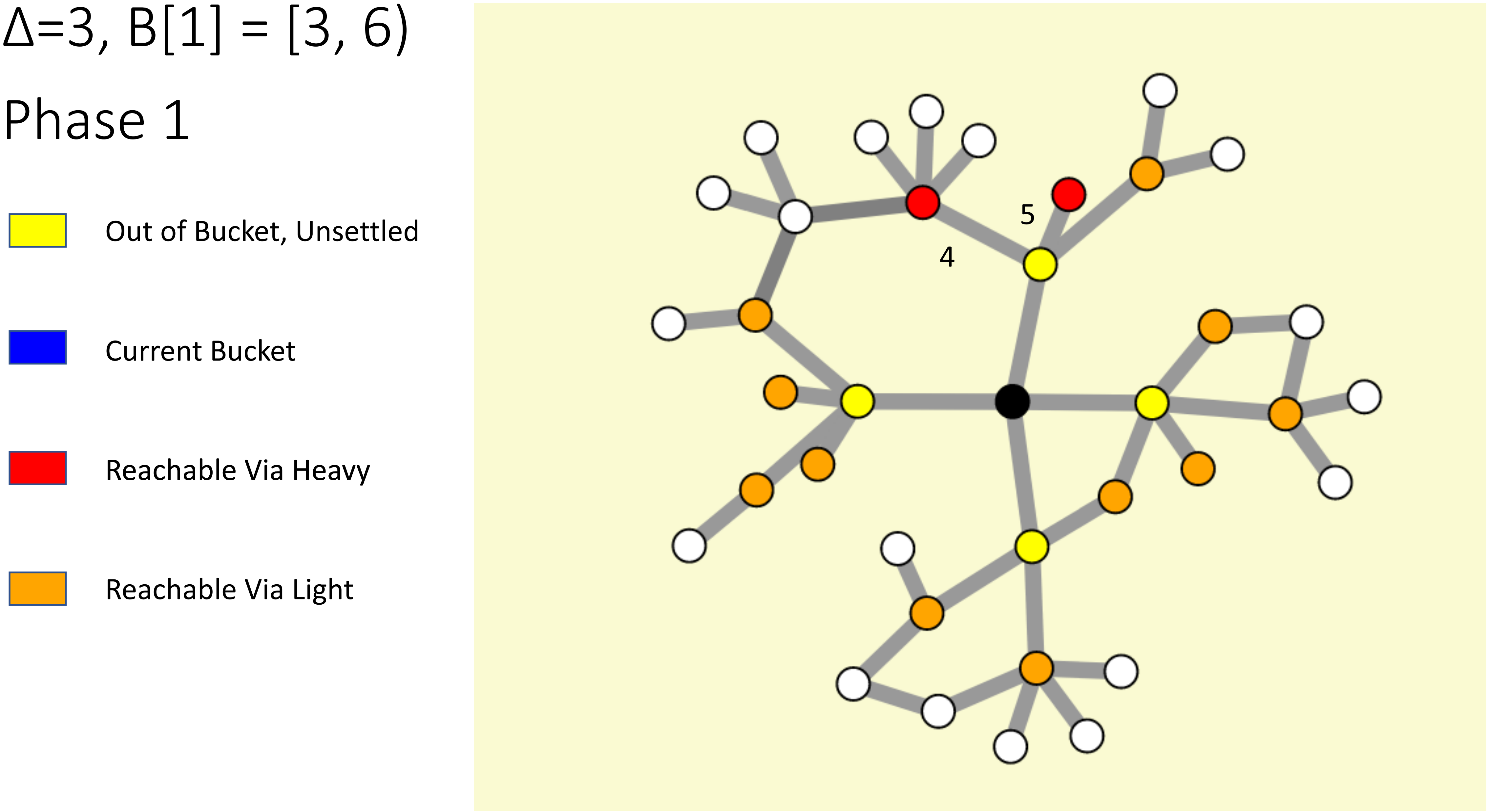
- Source
- Current Bucket
- Reachable Via Heavy
- Reachable Via Light



https://cs.iupui.edu/~fgsong/LearnHPC/sssp/deltaStep.html

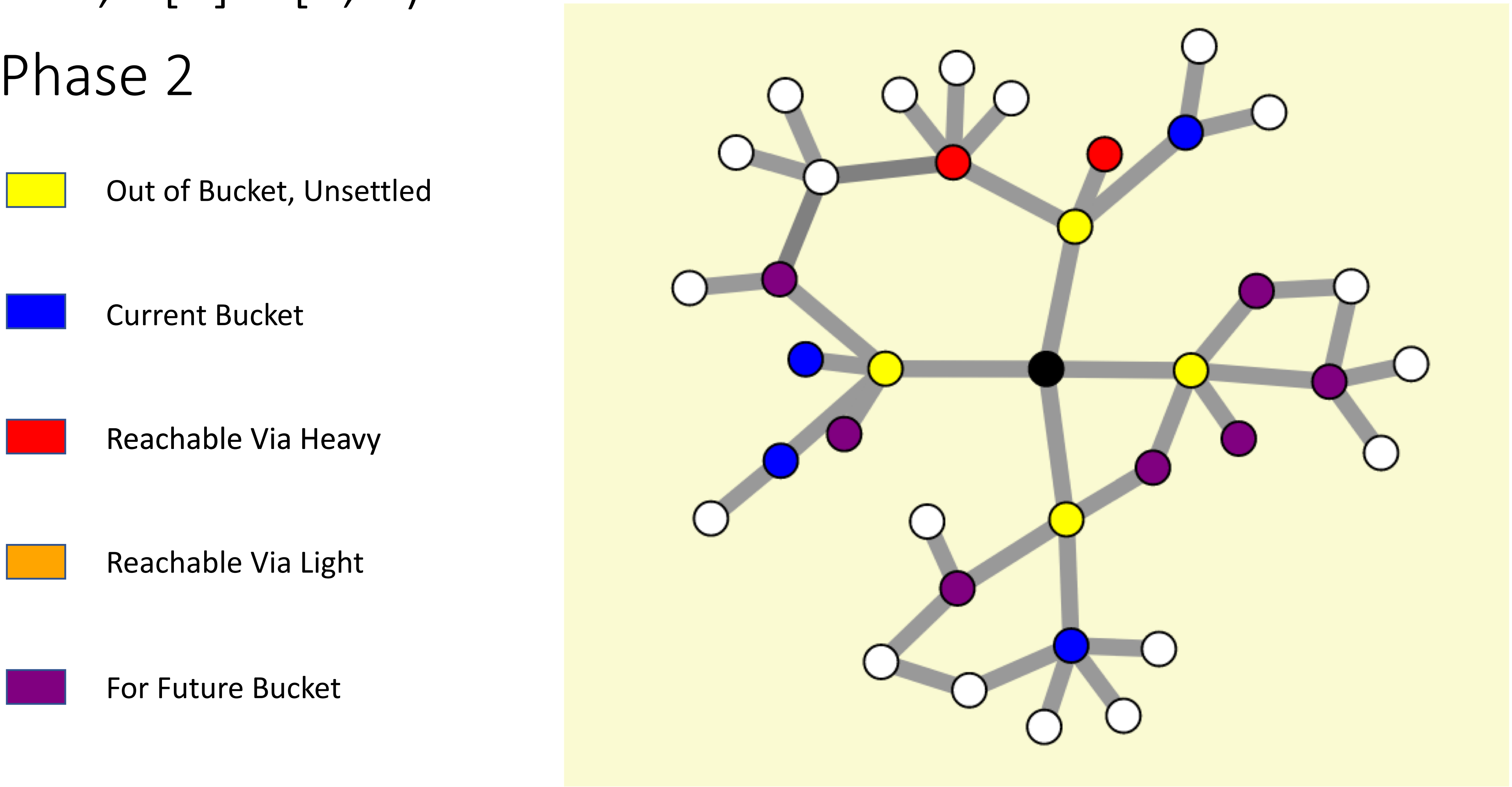
Δ=3, B[1] = [3, 6)

Phase 1



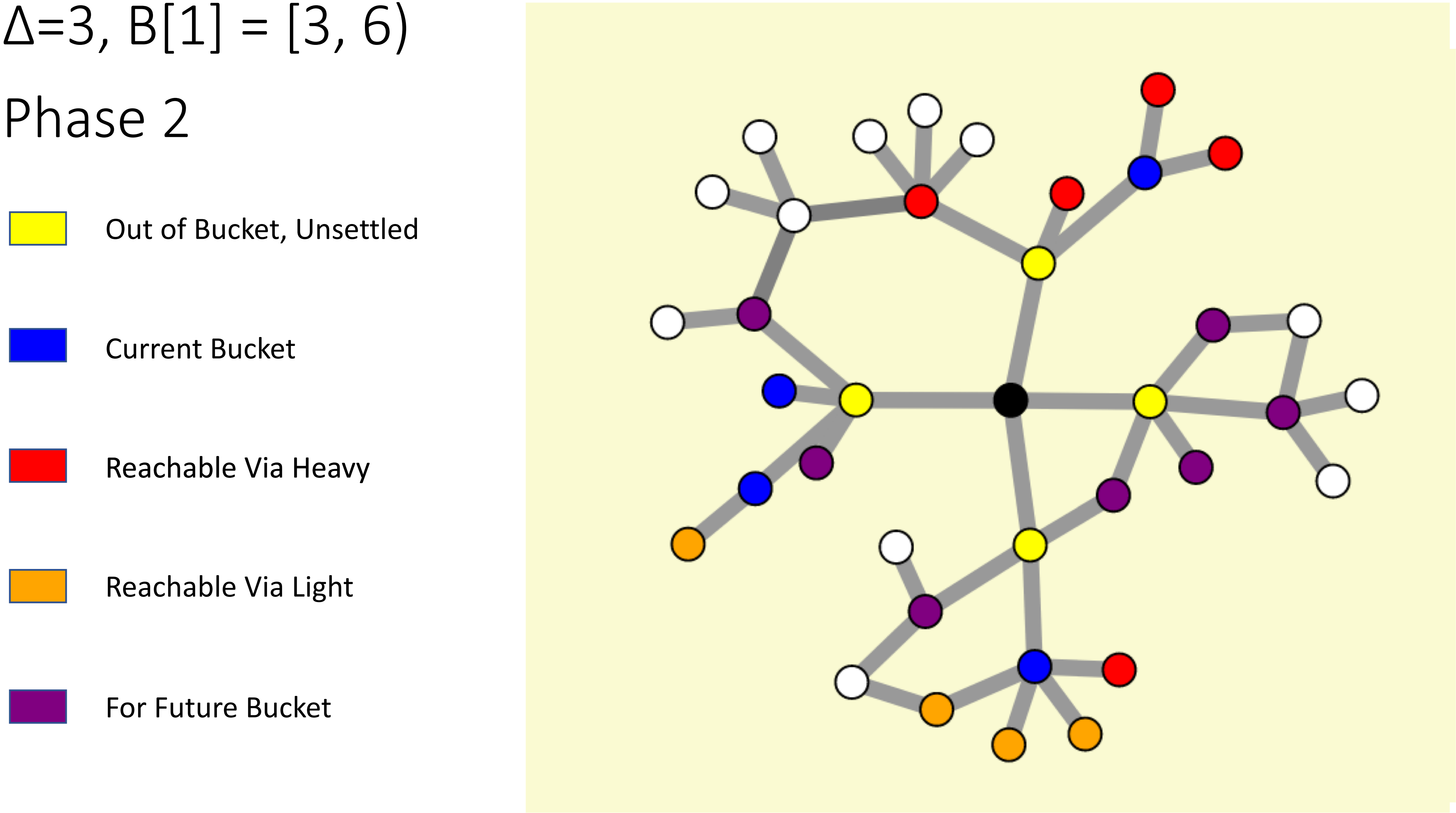
Δ=3, B[1] = [3, 6)

Phase 2



Δ=3, B[1] = [3, 6)

Phase 2



Δ=3, B[1] = [3, 6)

Phase 2

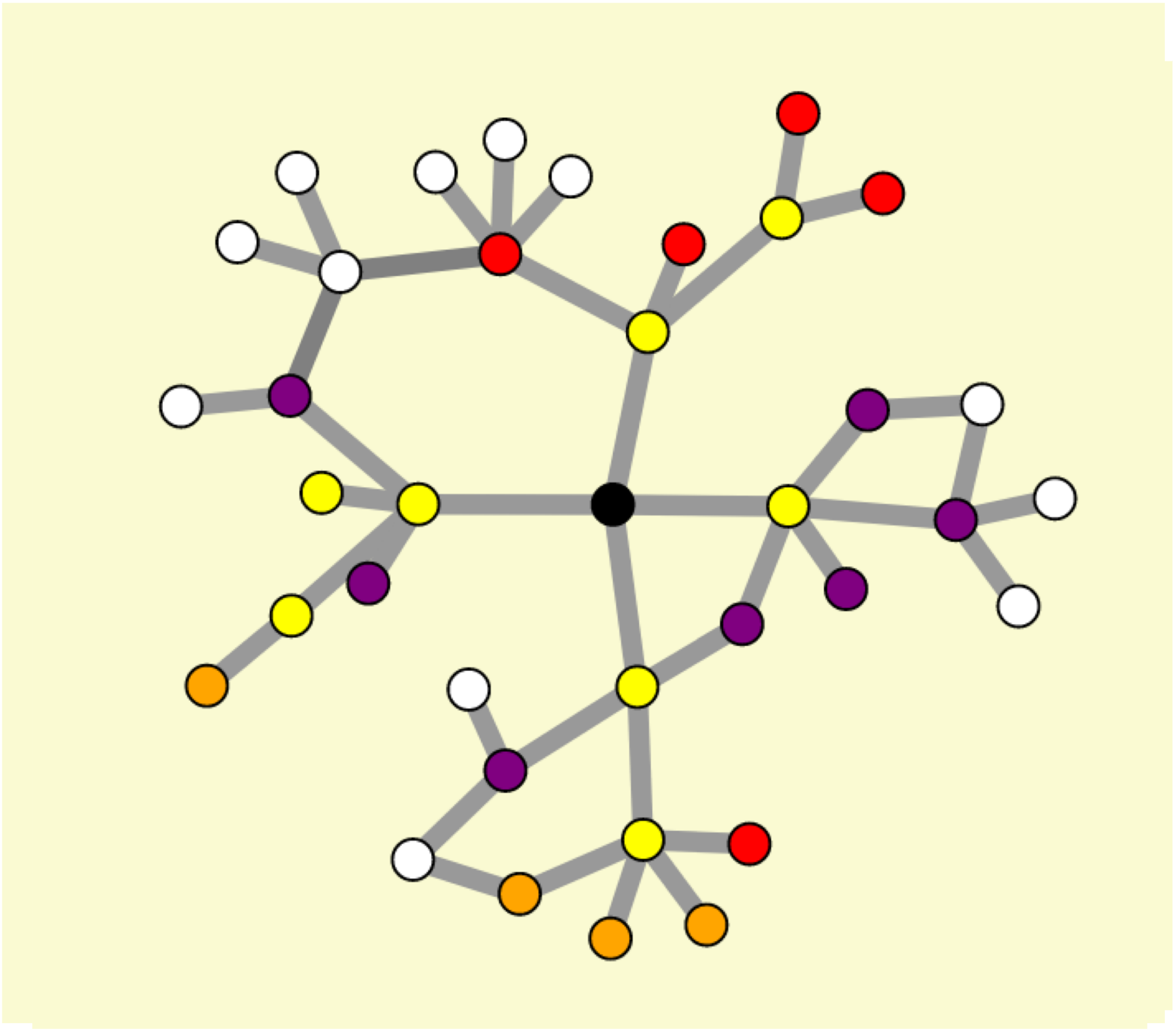
Out of Bucket, Unsettled

Current Bucket

Reachable Via Heavy

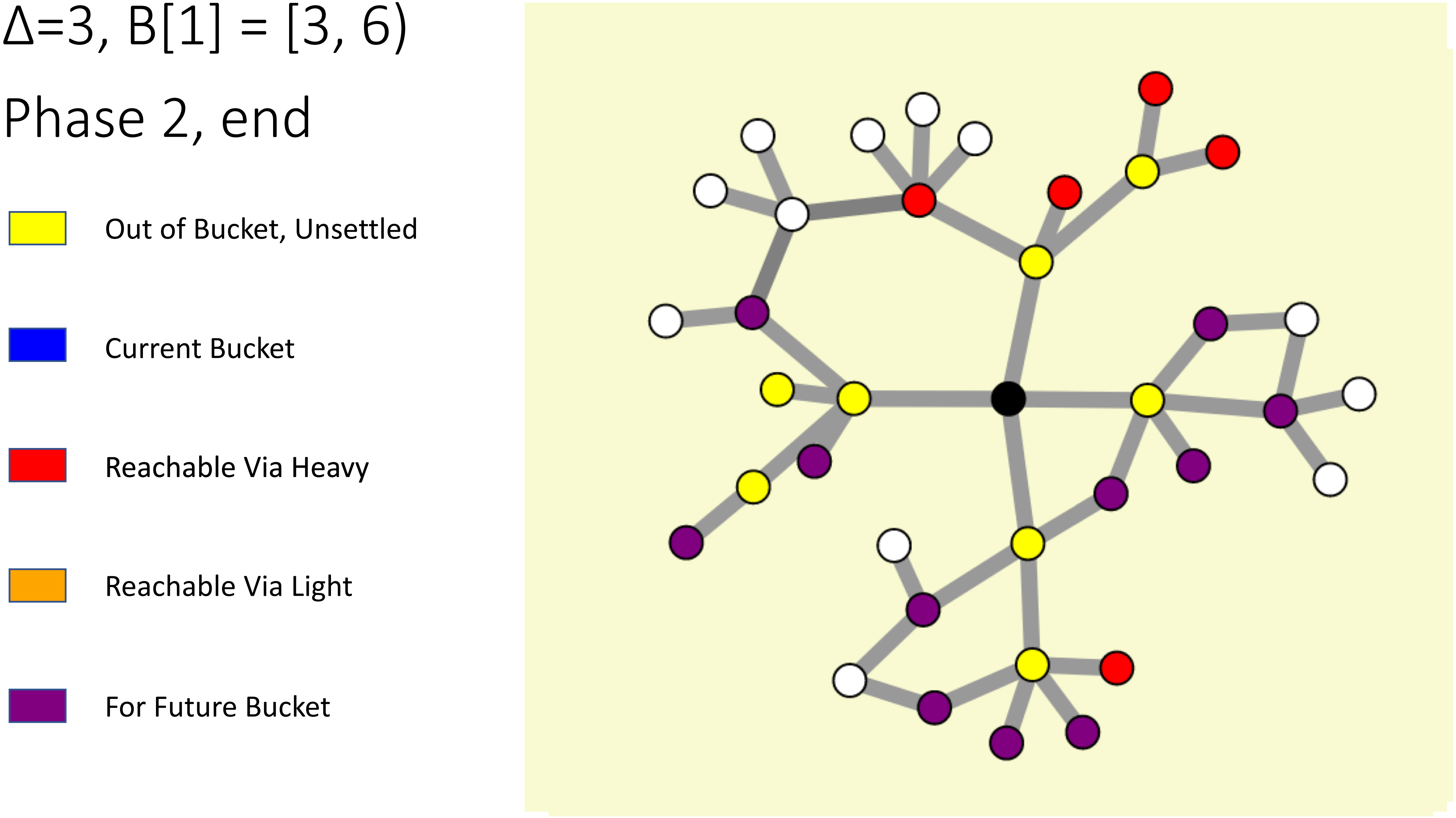
Reachable Via Light

For Future Bucket



Δ=3, B[1] = [3, 6)

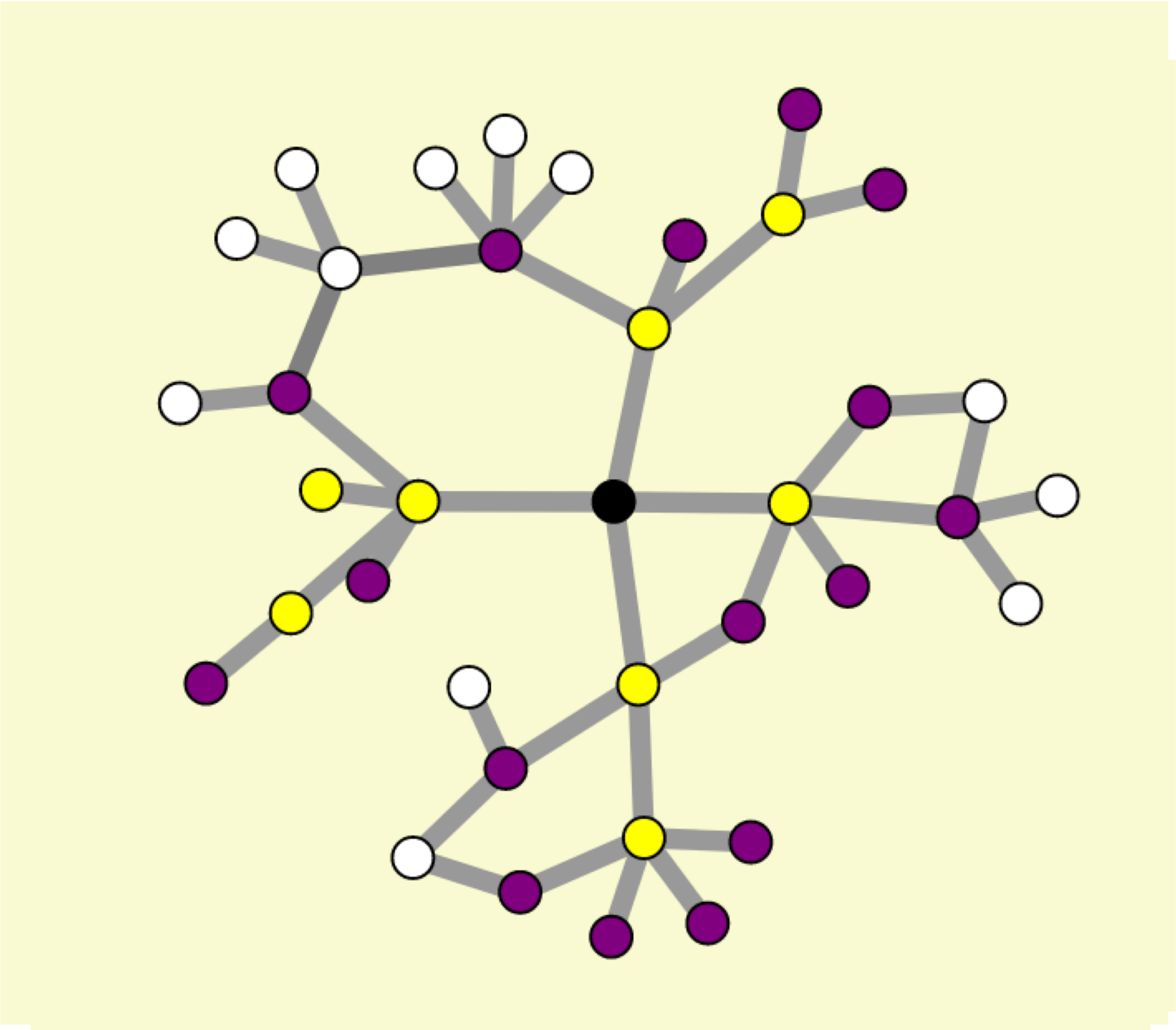
Phase 2, end



Δ=3, B[1] = [3, 6)

Relax Heavy

- Out of Bucket, Settled
- Current Bucket
- Reachable Via Heavy
- Reachable Via Light
- For Future Bucket



Δ=3, B[2] = [6, 9)

Phase 3

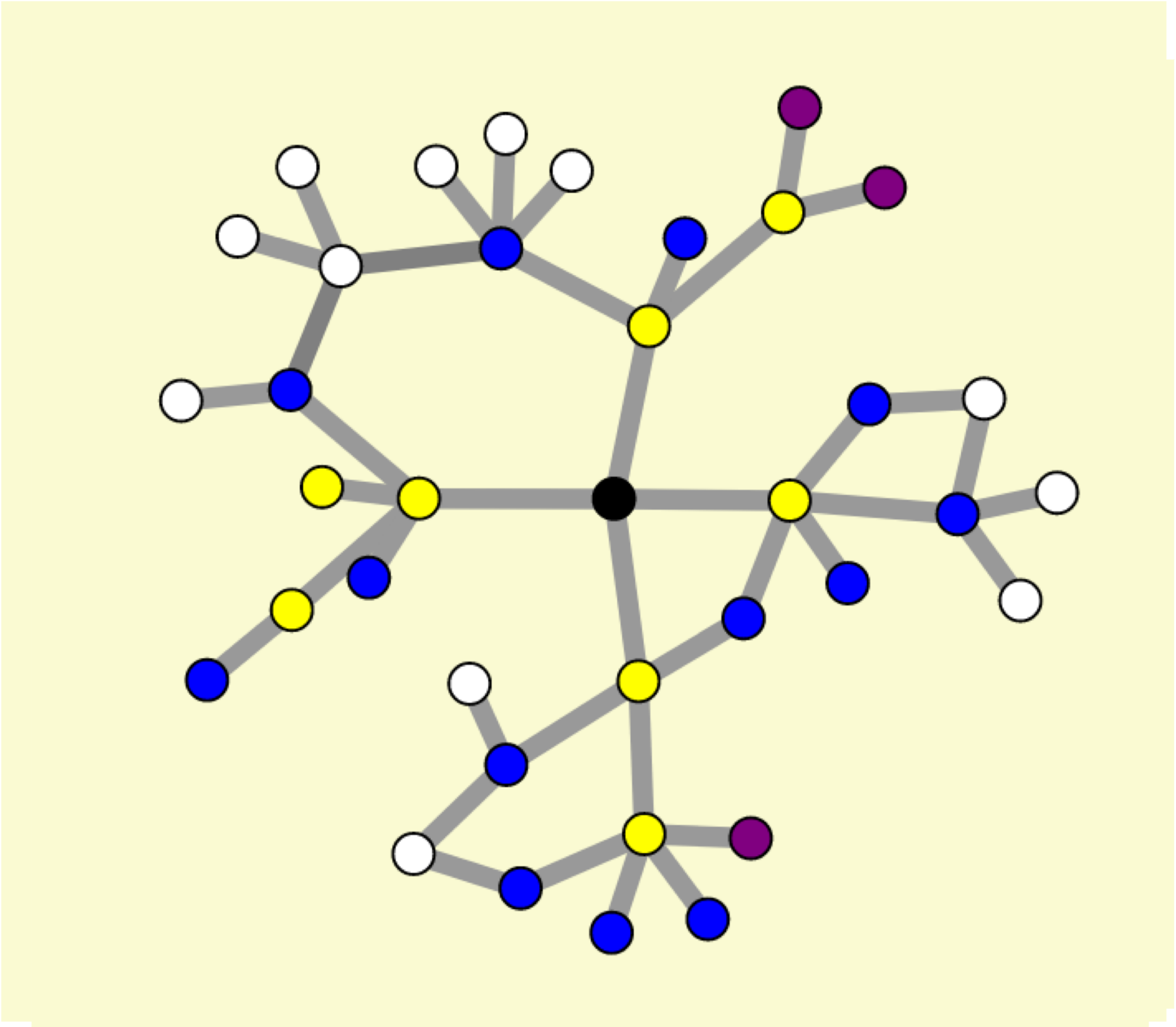
Out of Bucket, Settled

Current Bucket

Reachable Via Heavy

Reachable Via Light

For Future Bucket



Δ=3, B[2] = [6, 9)

Phase 3

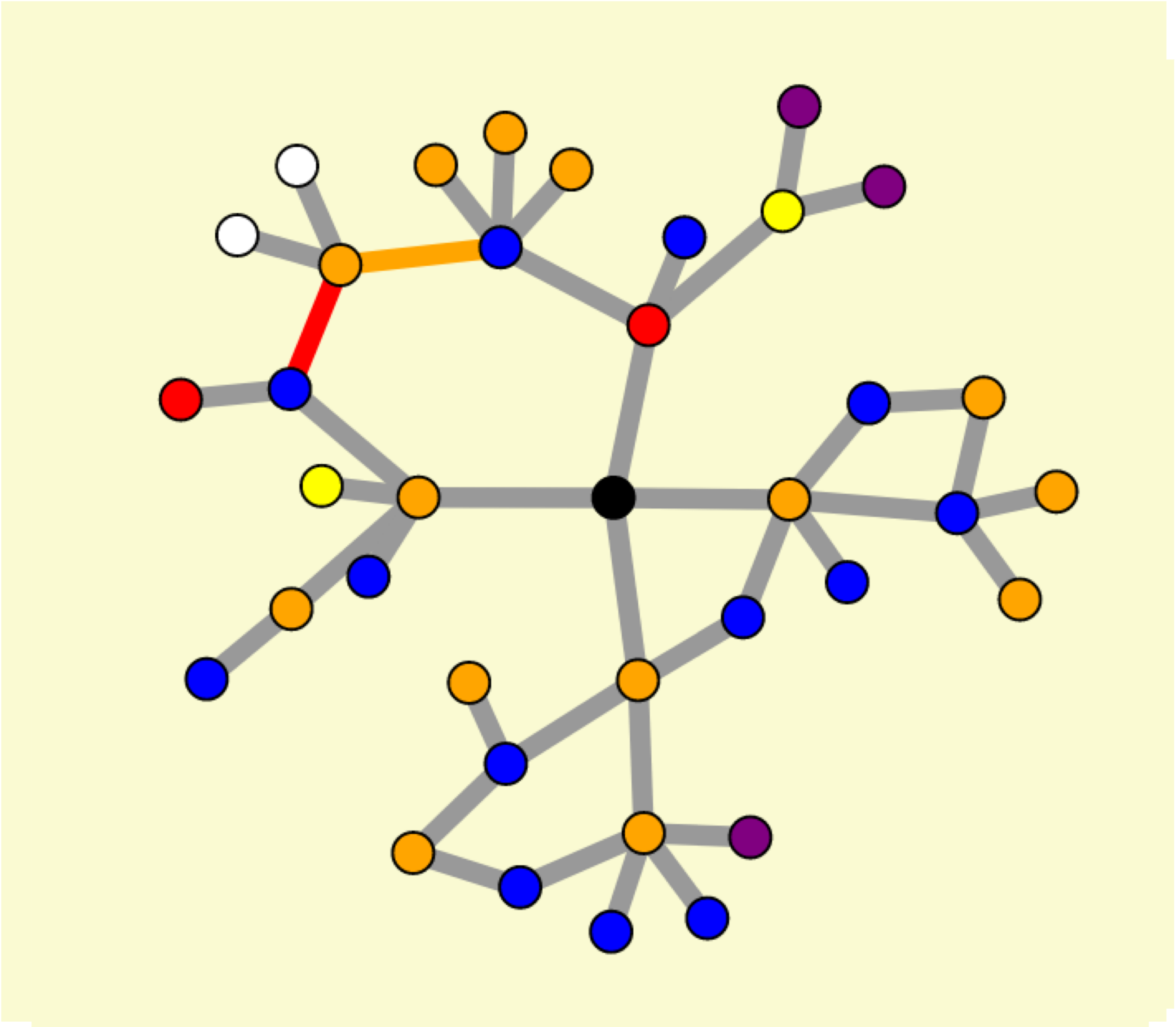
Out of Bucket, Unsettled

Current Bucket

Reachable Via Heavy

Reachable Via Light

For Future Bucket



Choice of Δ

- Δ = 1 reduces to Dijkstra’s
- Δ >= n * *max edge weight* reduces to Bellman-Ford (exclusively use first bucket)
- Δ-stepping wants to find easily computable fixed Δ that yields a good compromise between these two extremes

Analysis

- Sequential Δ-stepping can be implemented to run in time $O(n + m + L/\Delta + n_\Delta + m_\Delta)$
- If the edge weights are random, $n_\Delta + m_\Delta = O(n + m)$ whp for $\Delta = \Theta(1/d)$
- Therefore runs in $O(n + m + d \cdot L)$ on random edge weights
- *d* can be removed from the execution time using more sophisticated load balancing algorithms

Parallel Analysis

- Simple parallelization runs in $O(L/\Delta \cdot d \cdot l_{\Delta} \cdot \log n)$
- Can accelerate by preprocessing the graph with shortcut edges for each shortest path <= Δ
  - Shortcuts ensure constant number of phases per nonempty bucket
  - Shortcuts found by exploring from all nodes in parallel.
- For random edge weights, it can then take $O(d \cdot L \cdot \log n + \log^2 n)$ time and $O(n + m + d \cdot L \cdot \log n)$ work on average

Performance Evaluation

- Implemented algorithm for distributed memory using MPI
- 9.2x speedup against sequential with 16 processors
- Sequential is 3.1x faster than optimized Dijkstra
- Worse on dense graphs

Drawbacks

- No average-case analysis done on non-integer weights
- Tuning Δ on graph without independent random edge weights