Stackless Multi-BVH Traversal for CPU, MIC and GPU Ray Tracing

Attila T. Áfra[1,2]
László Szirmay-Kalos[1]

Budapest University of Technology and Economics[1]
Babeș-Bolyai University[2]

EGSR 26th EUROGRAPHICS SYMPOSIUM ON RENDERING

BVH traversal

- Binary BVH
- Multi-BVH
  - *N*-way tree
  - *N* boxes or triangles per node
  - SIMD-friendly, smaller footprint

- Most ray traversal algorithms require a stack
  - About 200-1000 bytes per ray!
- Very high memory cost if many rays are in flight
- Solution: stackless traversal algorithms

Stackless traversal algorithms

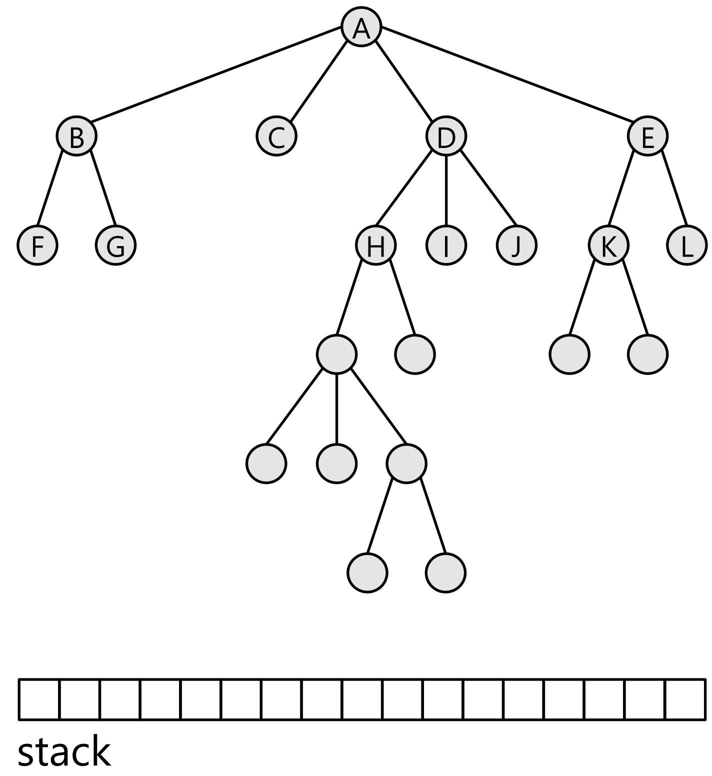
- Binary BVH
  - Skip pointers [Smits 1998]
  - Restart trail [Laine 2010]
  - Parent pointers, state logic [Hapala et al. 2011]
  - Dynamic stackless binary tree traversal [Barringer & Akenine-Möller 2013]

Stackless traversal algorithms

- Binary BVH
  - Skip pointers [Smits 1998]
  - Restart trail [Laine 2010]
  - Parent pointers, state logic [Hapala et al. 2011]
  - Dynamic stackless binary tree traversal [Barringer & Akenine-Möller 2013]
- Multi-BVH
  - None!

Stack-based MBVH traversal

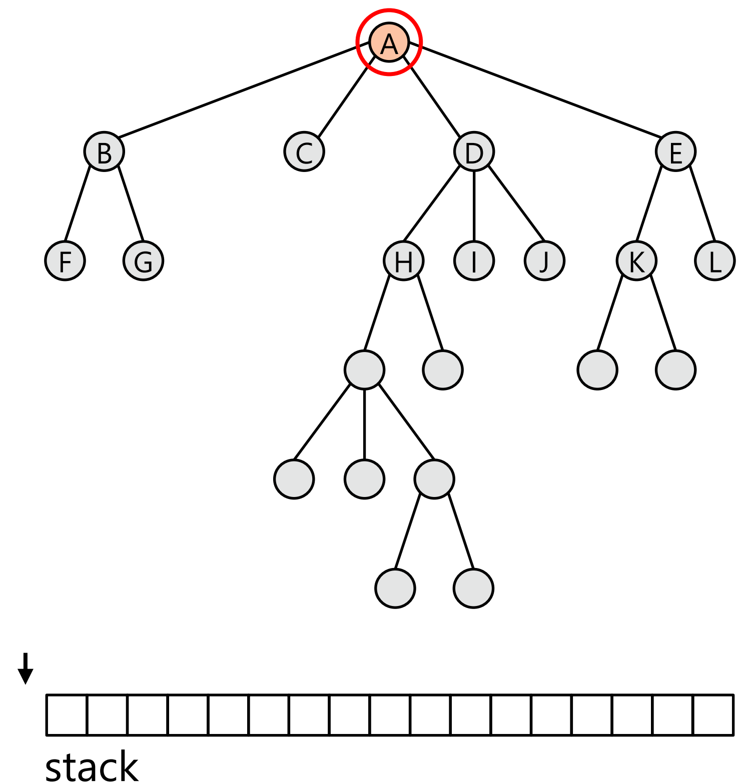
- Embree [Wald et al. 2014]



Stack-based MBVH traversal

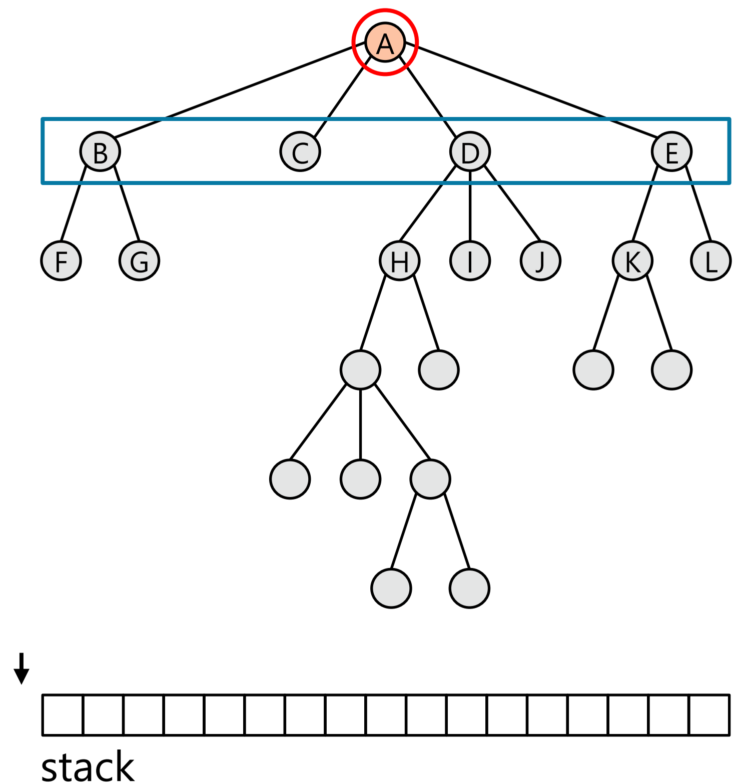
- Embree [Wald et al. 2014]

- Inner nodes:



Stack-based MBVH traversal

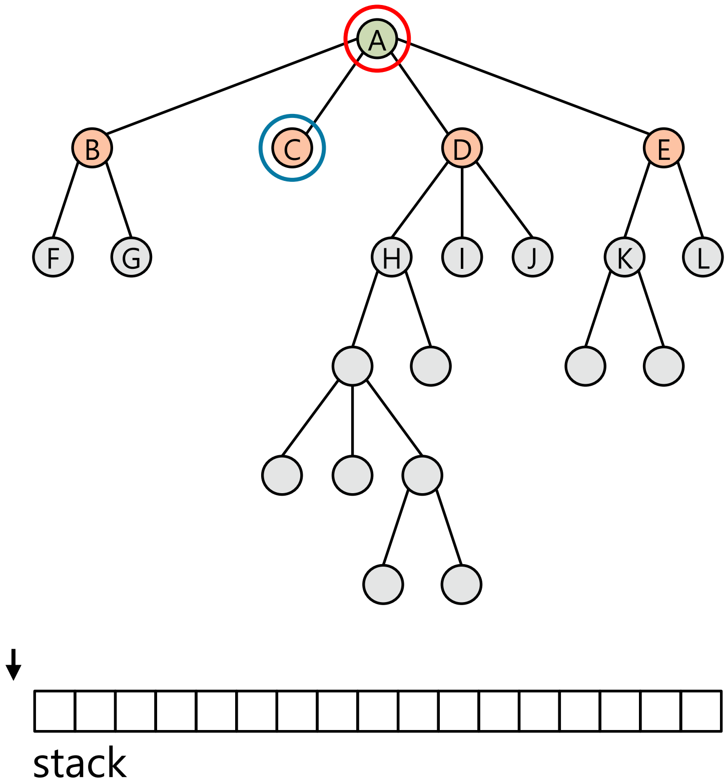
- Embree [Wald et al. 2014]
- Inner nodes:
  - Intersect *N* child boxes



Stack-based MBVH traversal

- Embree [Wald et al. 2014]

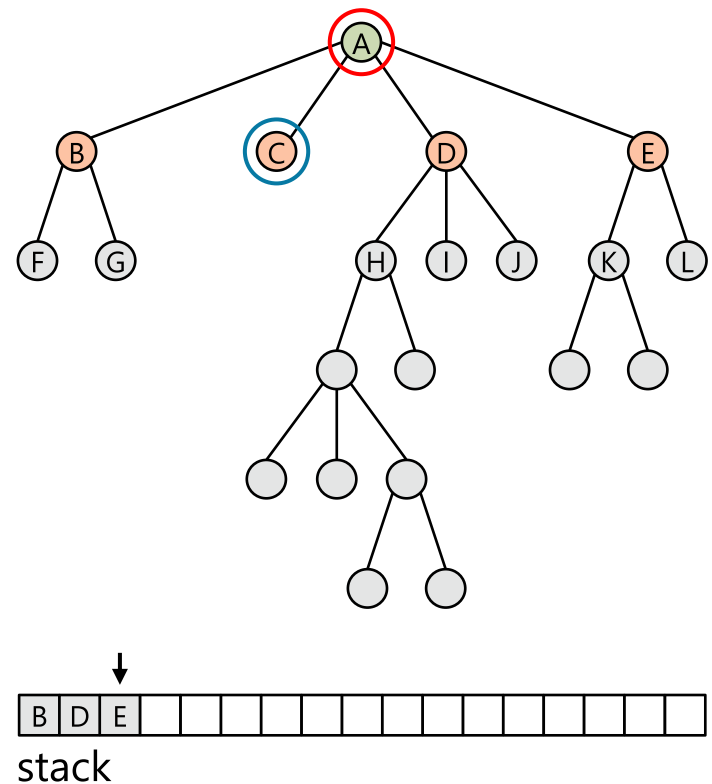
- Inner nodes:
  - Intersect *N* child boxes
  - Select the closest child



Stack-based MBVH traversal

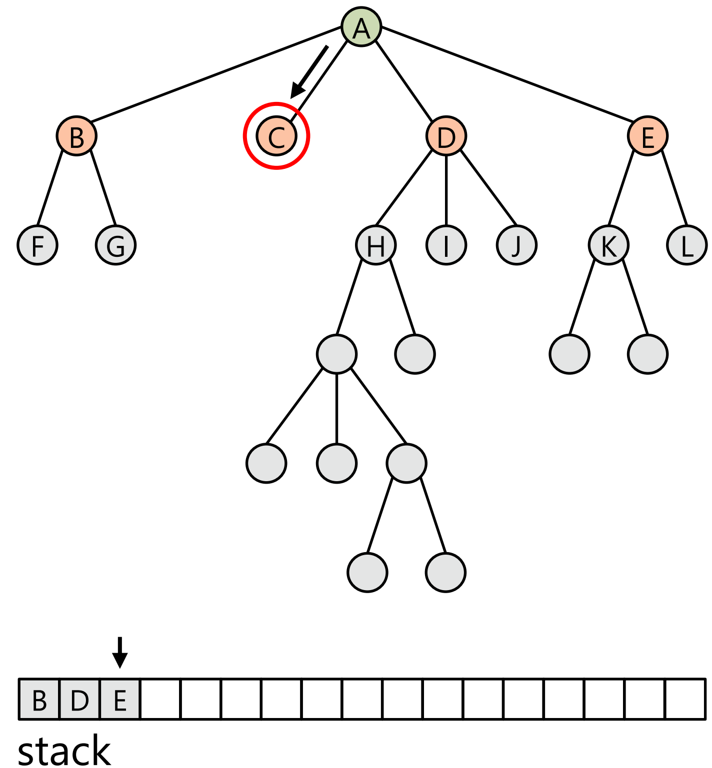
- Embree [Wald et al. 2014]

- Inner nodes:
  - Intersect *N* child boxes
  - Select the closest child
  - Push the other children to the stack



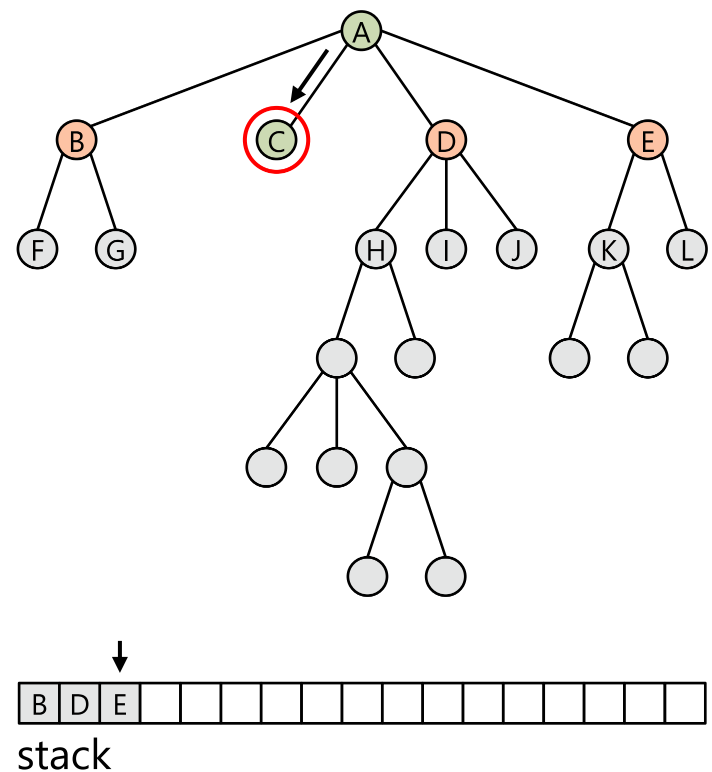
Stack-based MBVH traversal

- Embree [Wald et al. 2014]
- Inner nodes:
  - Intersect *N* child boxes
  - Select the closest child
  - Push the other children to the stack



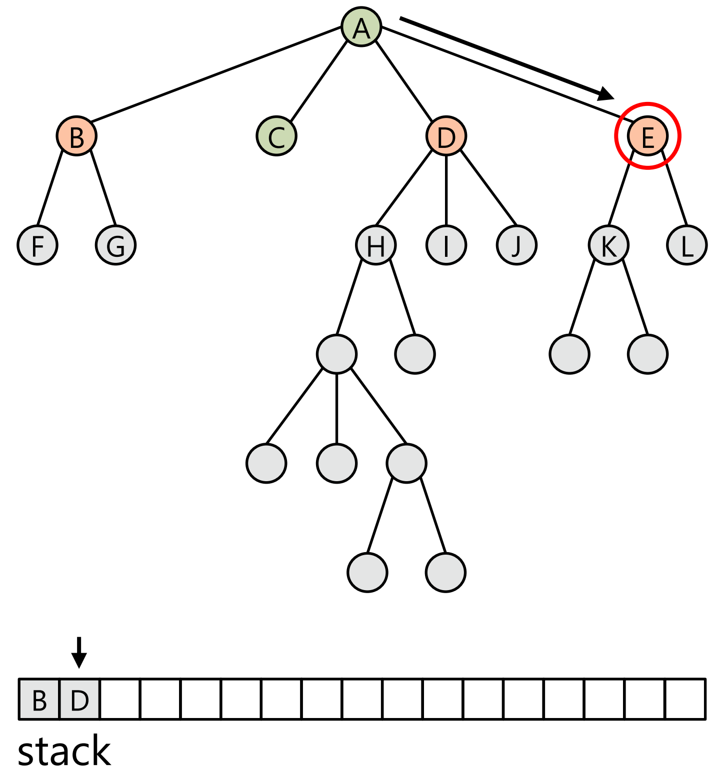
Stack-based MBVH traversal

- Embree [Wald et al. 2014]
- Inner nodes:
  - Intersect *N* child boxes
  - Select the closest child
  - Push the other children to the stack
- Leaf nodes:
  - Intersect *N* triangles



Stack-based MBVH traversal

- Embree [Wald et al. 2014]
- Inner nodes:
  - Intersect *N* child boxes
  - Select the closest child
  - Push the other children to the stack
- Leaf nodes:
  - Intersect *N* triangles
  - Pop the next node from the stack

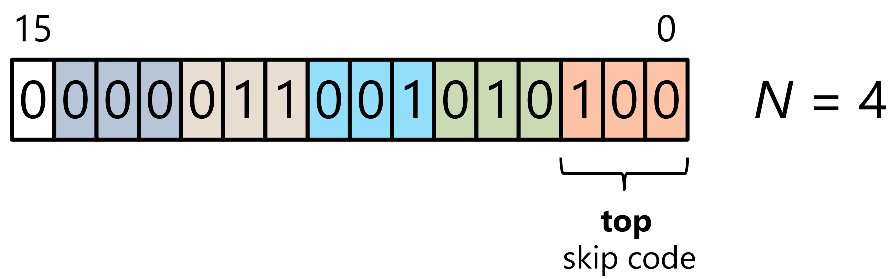


Our approach

- Replaces the stack pop with backtracking in the tree
- The backtracking is guided by a small bitstack
  - Stored in an integer
- Supports dynamic ordered traversal
- Intersects each visited node only once
- Requires parent and sibling pointers

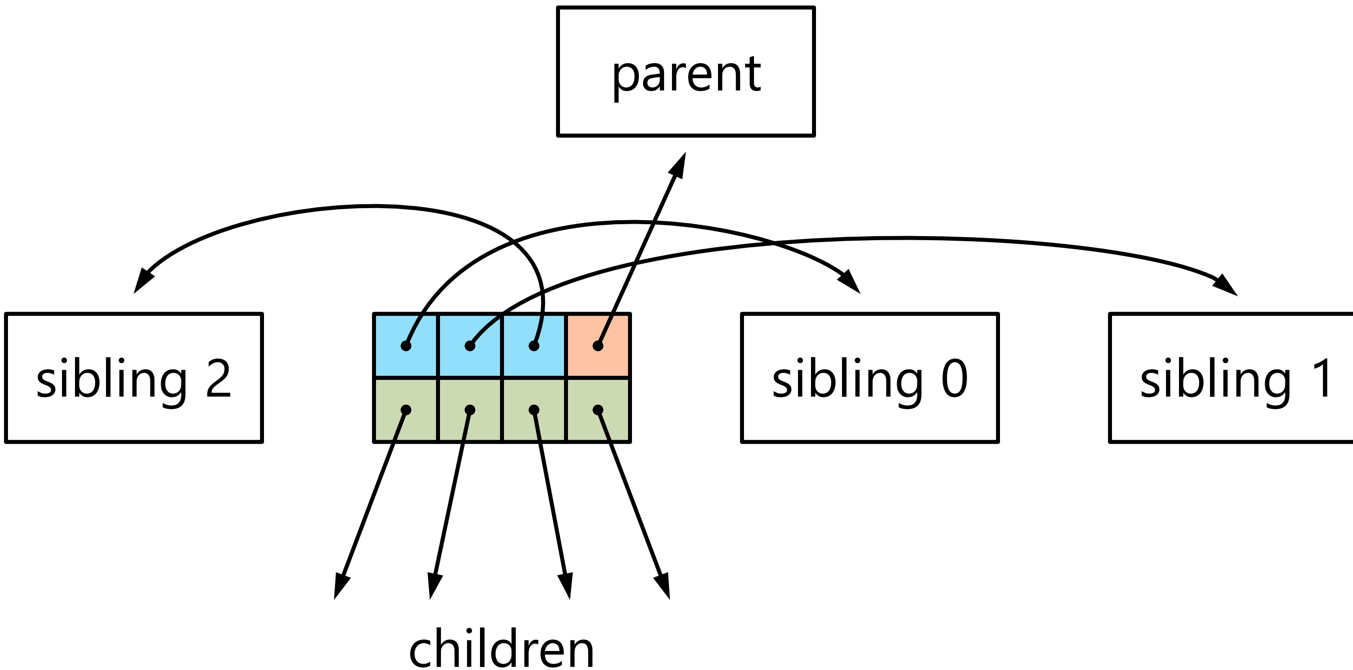
Bitstack

- A skip code is pushed for each visited tree level
  - N–1 bits
  - Encodes which siblings of the current node must be traversed



- The bits refer to siblings in circular order:
  - 0 → skip
  - 1 → traverse

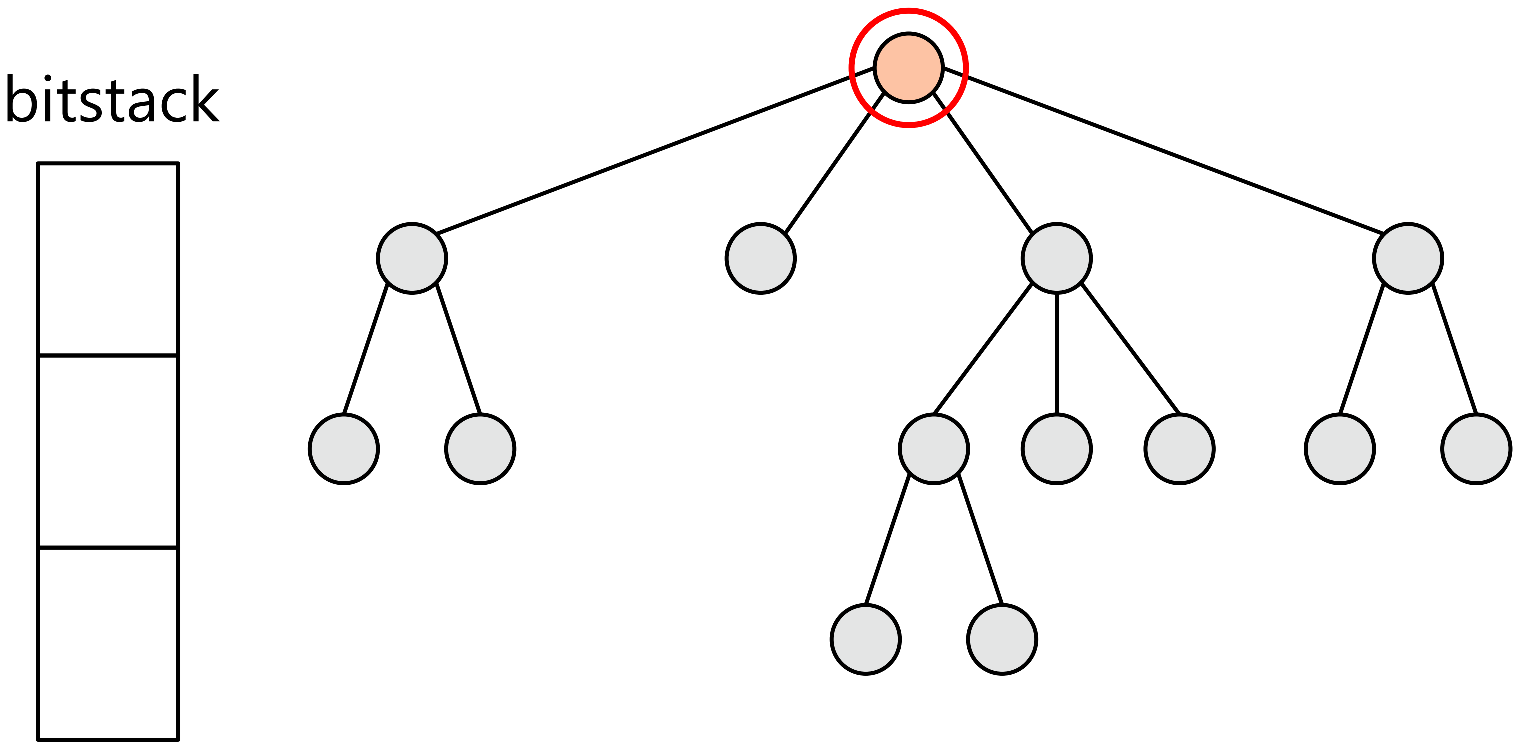
Node pointers



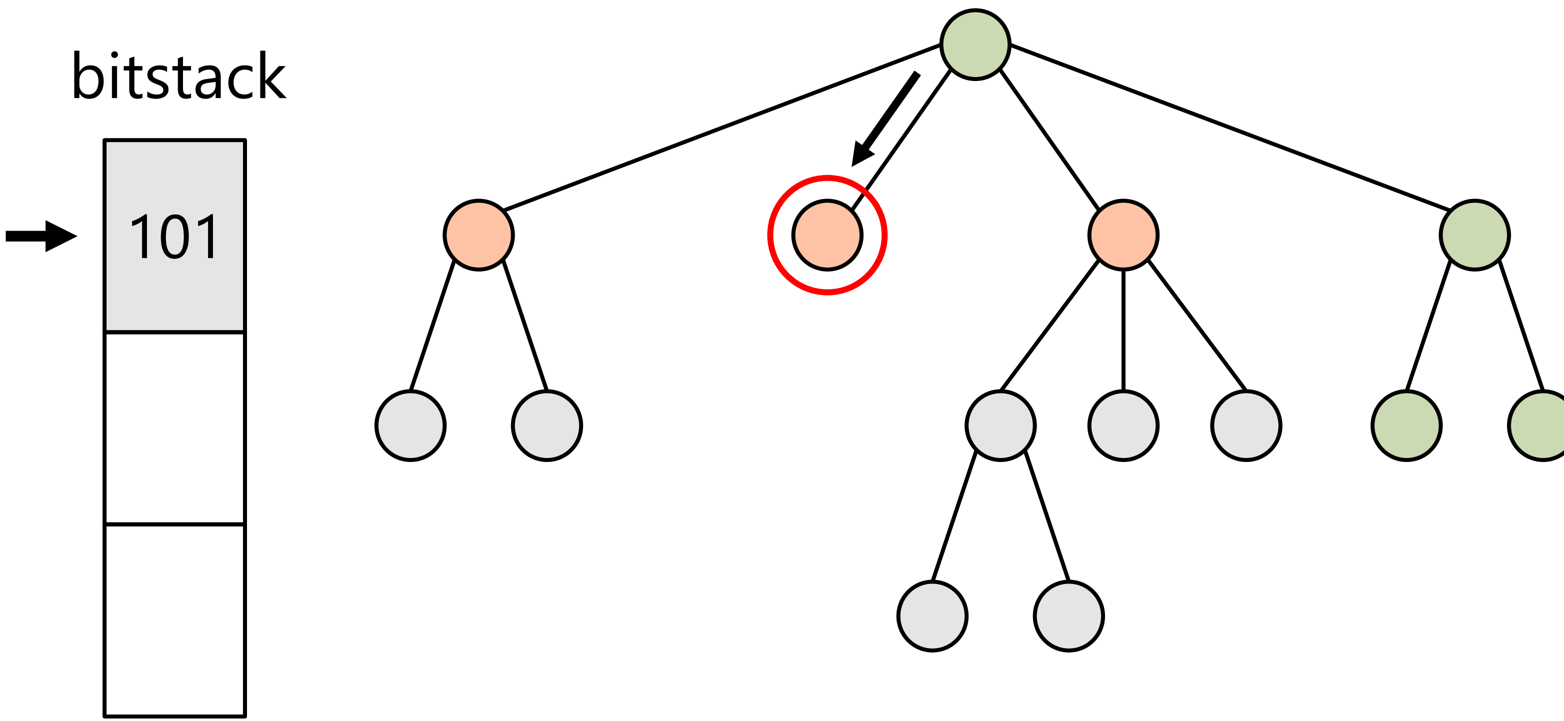
Backtracking

- While the top skip code is zero:
  - Go to the parent
  - Pop the skip code
- Go to the next unprocessed sibling
  - Sibling index = first set bit in the skip code
- Update the top skip code
  - Bit shift

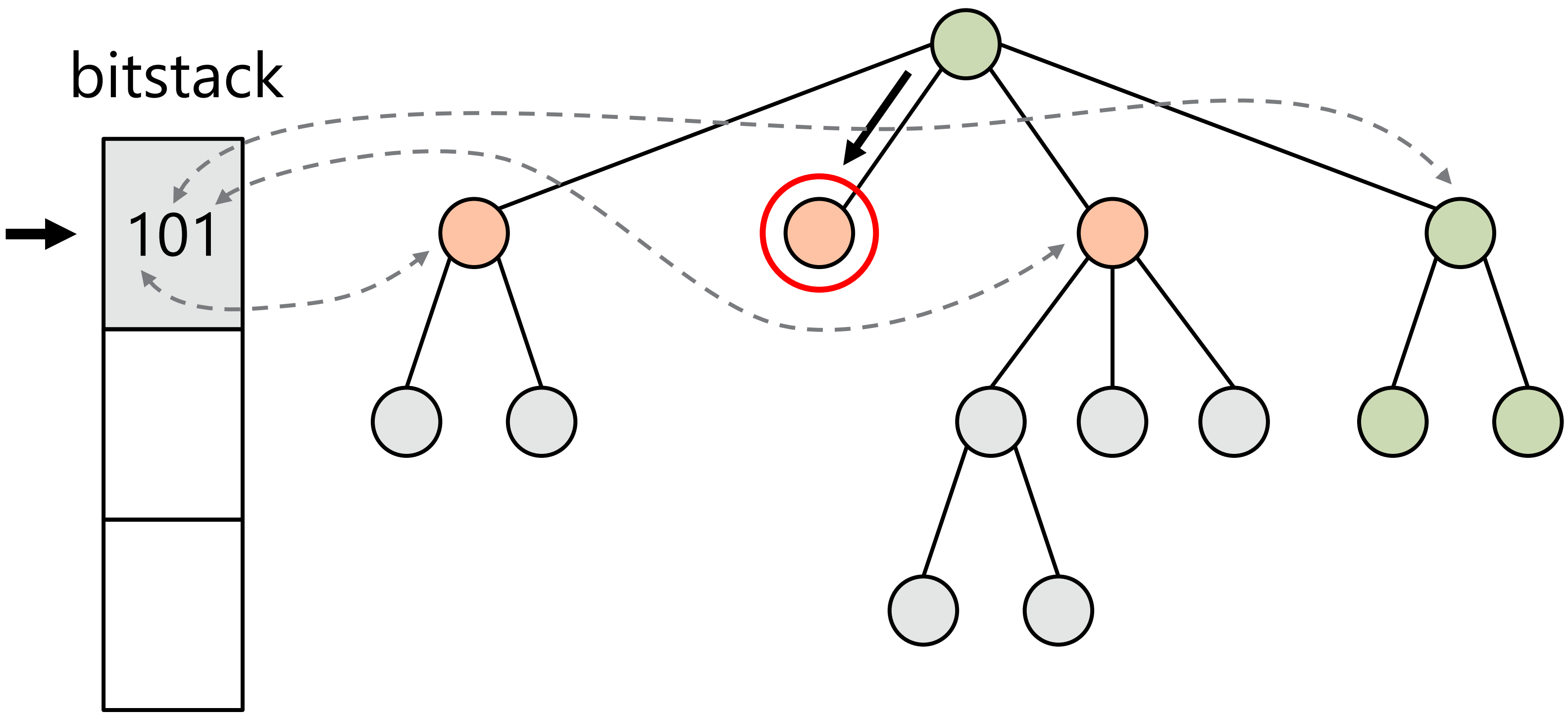
Traversal example



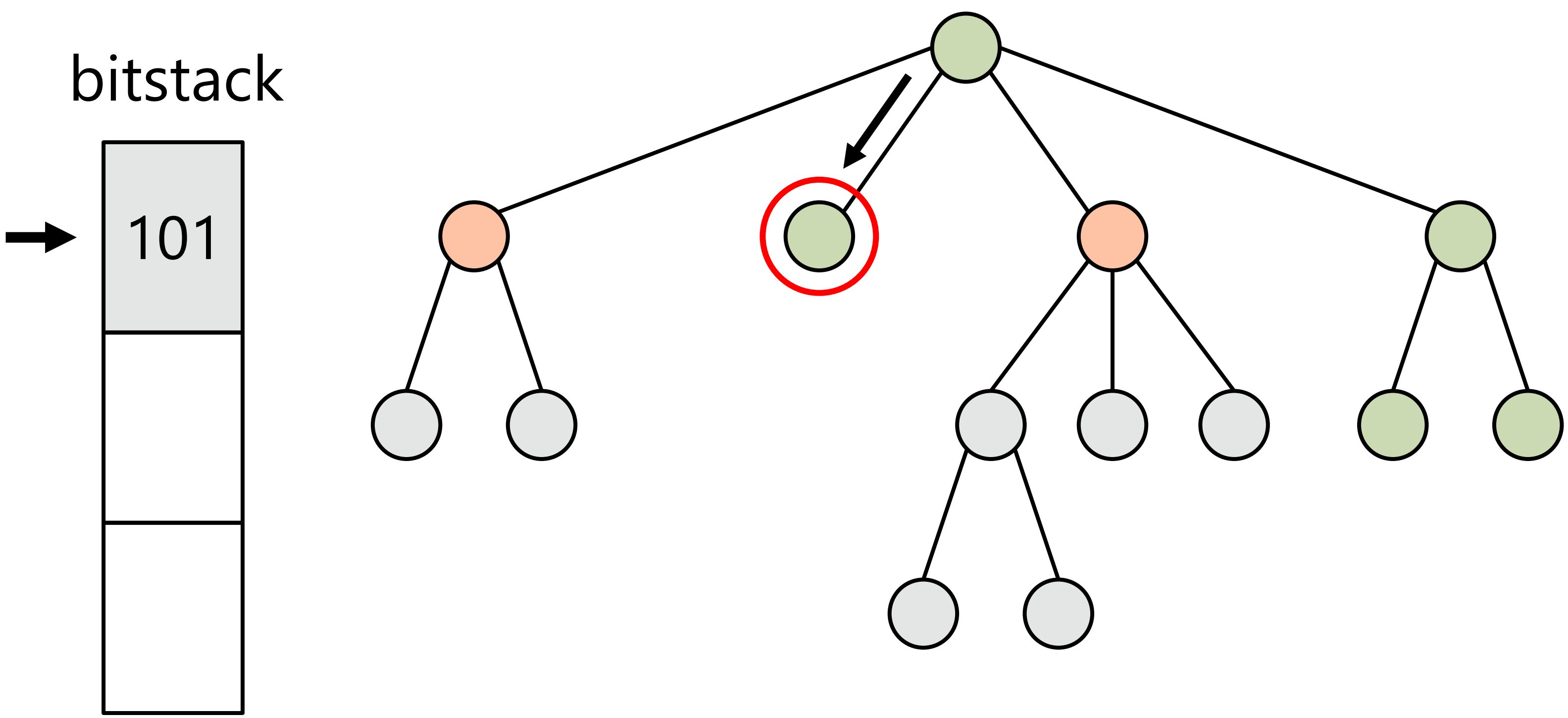
Traversal example



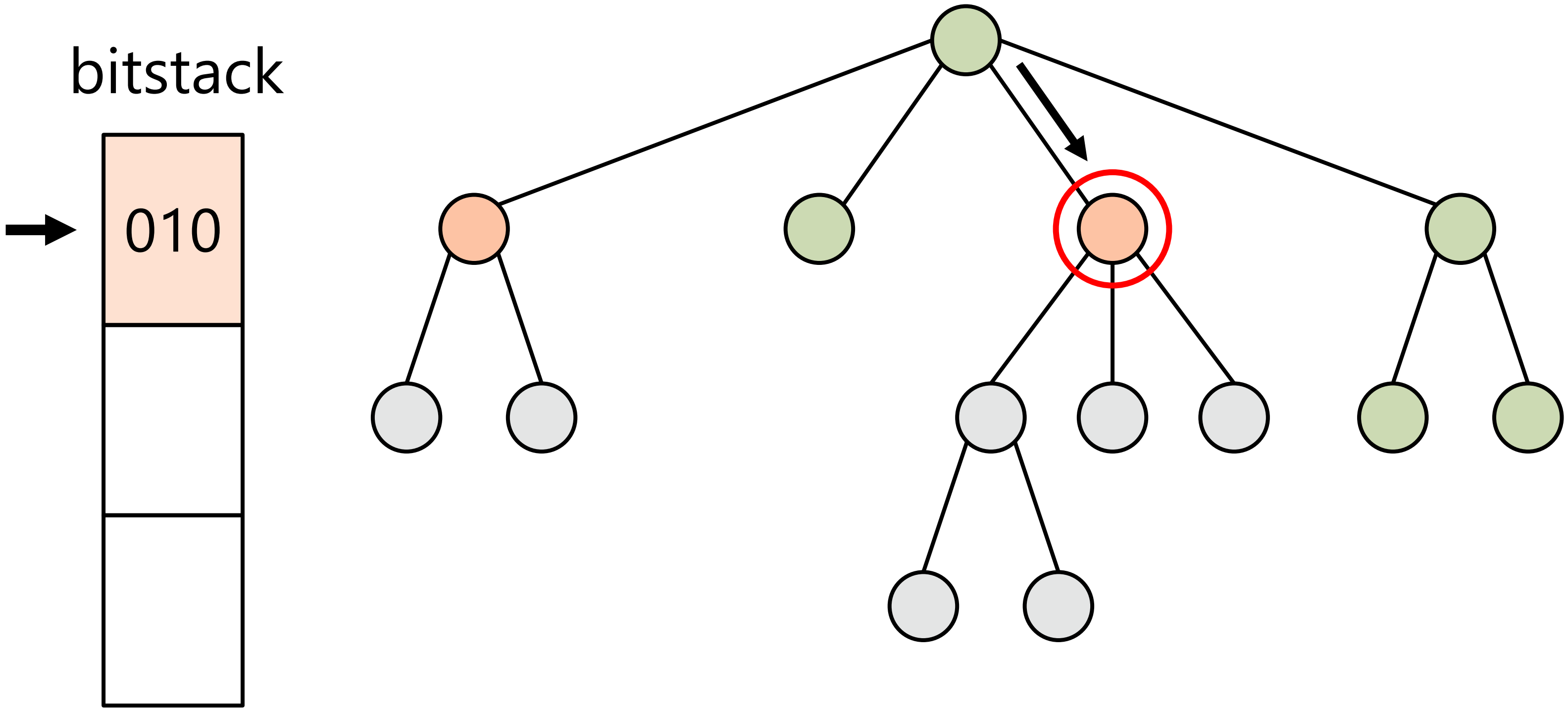
Traversal example



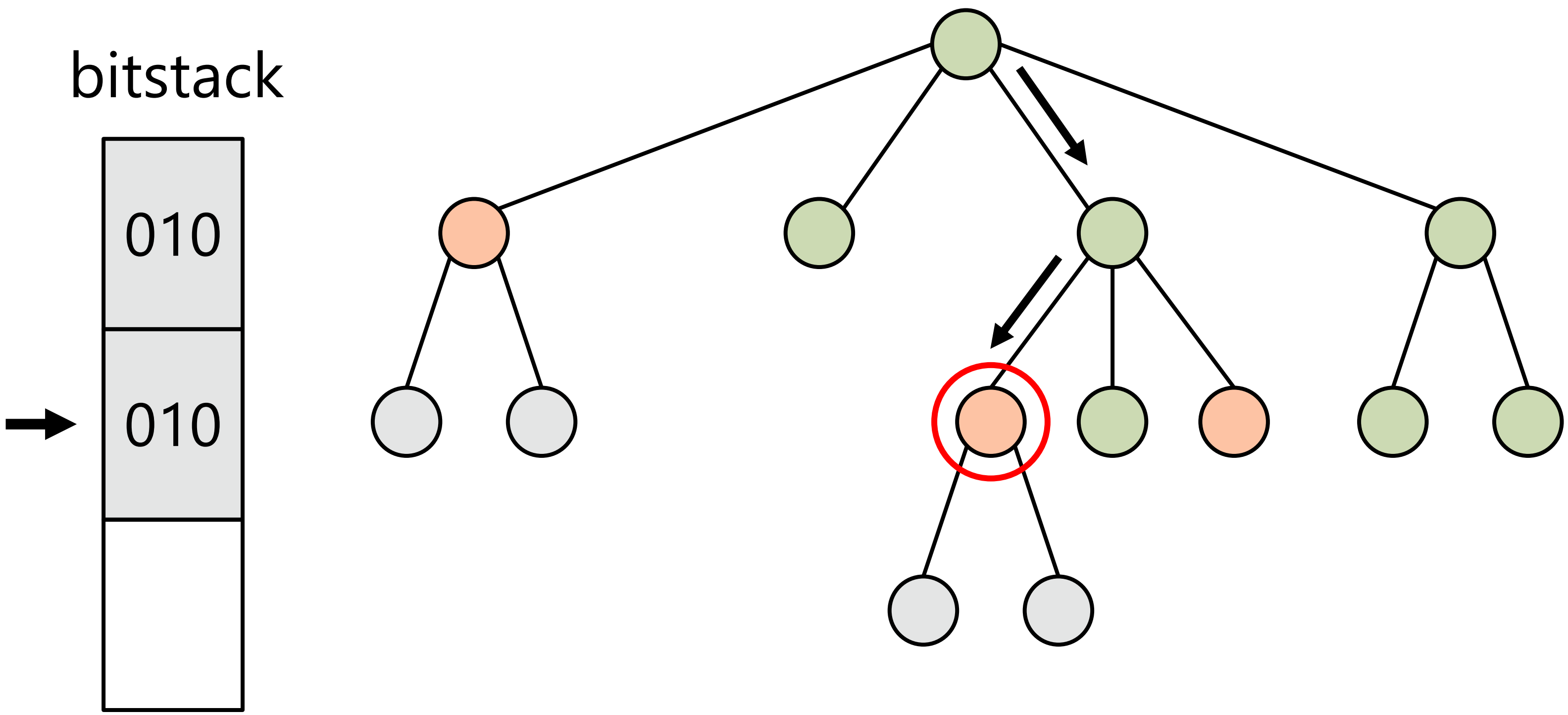
Traversal example



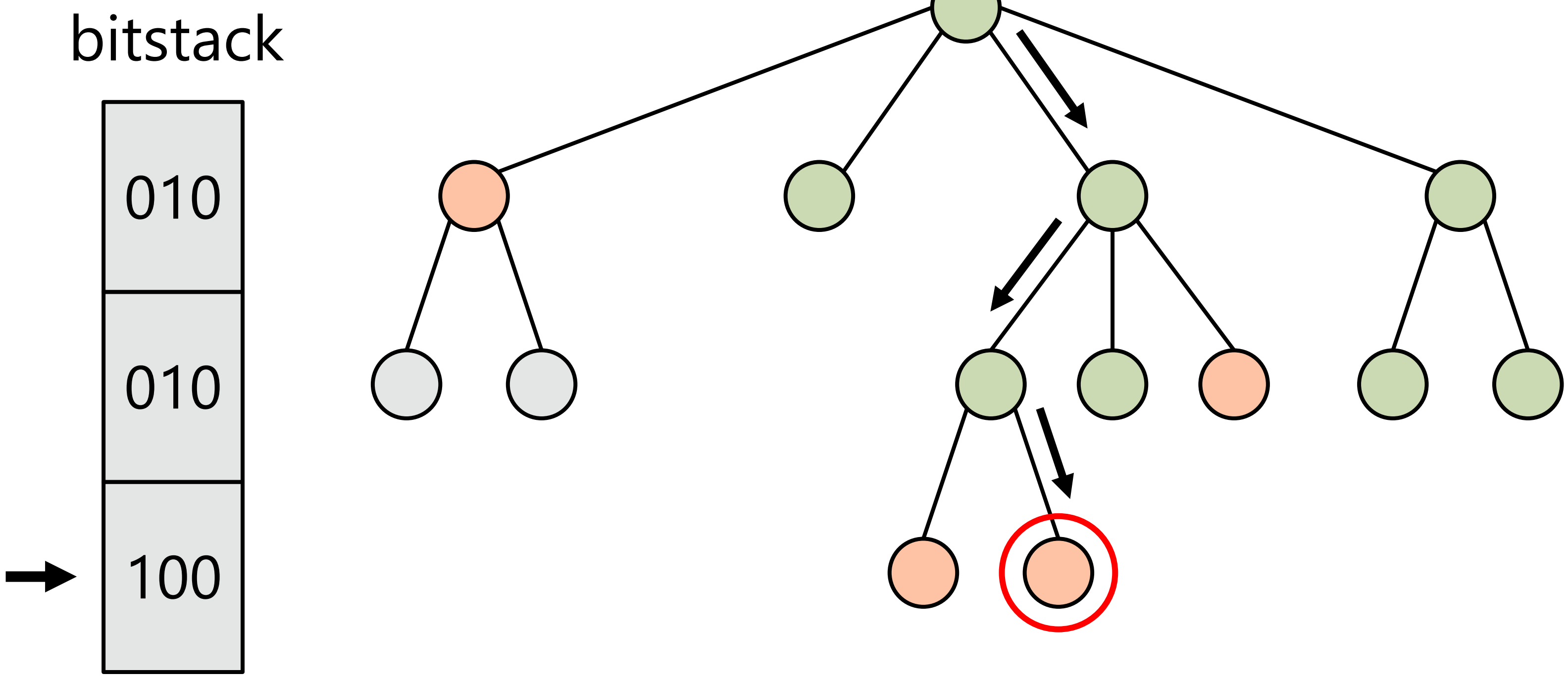
Traversal example



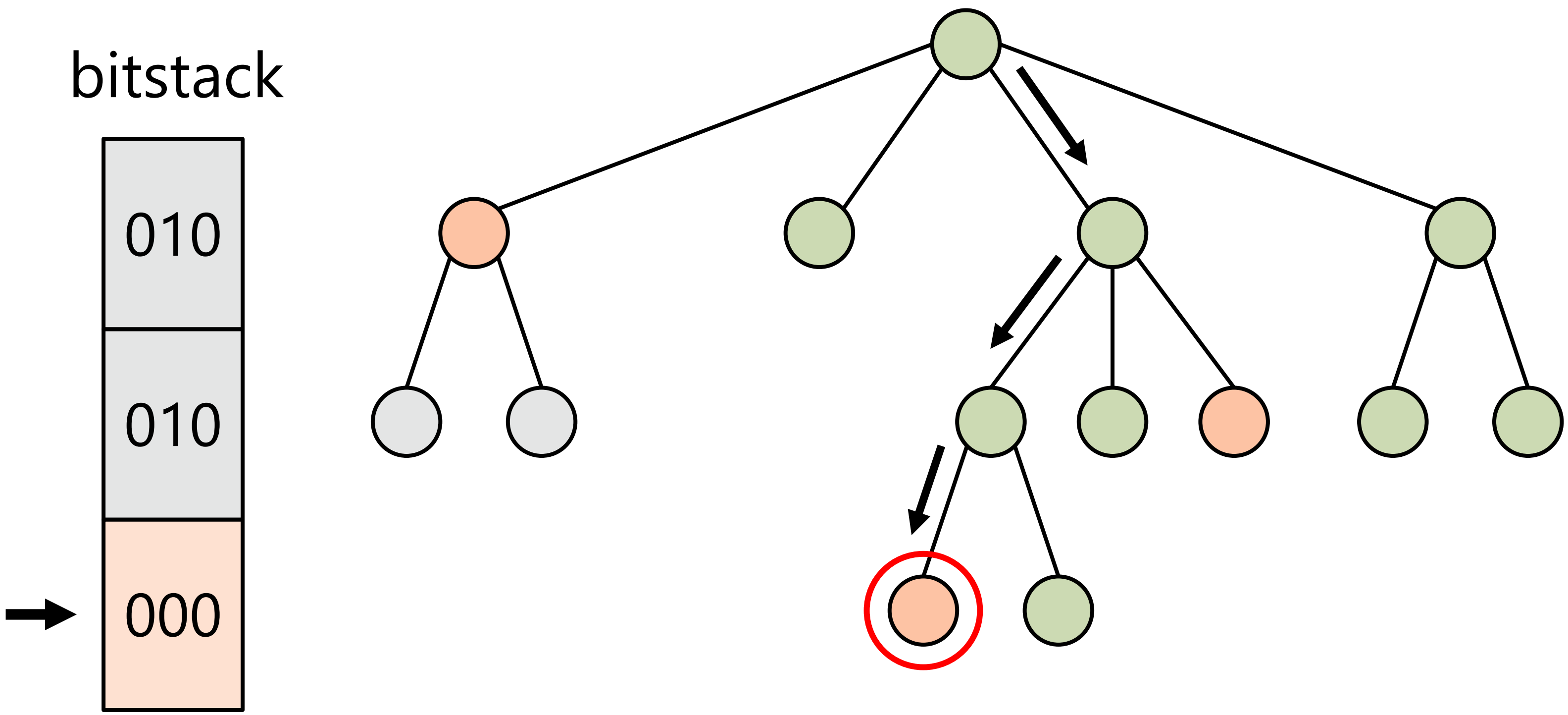
Traversal example



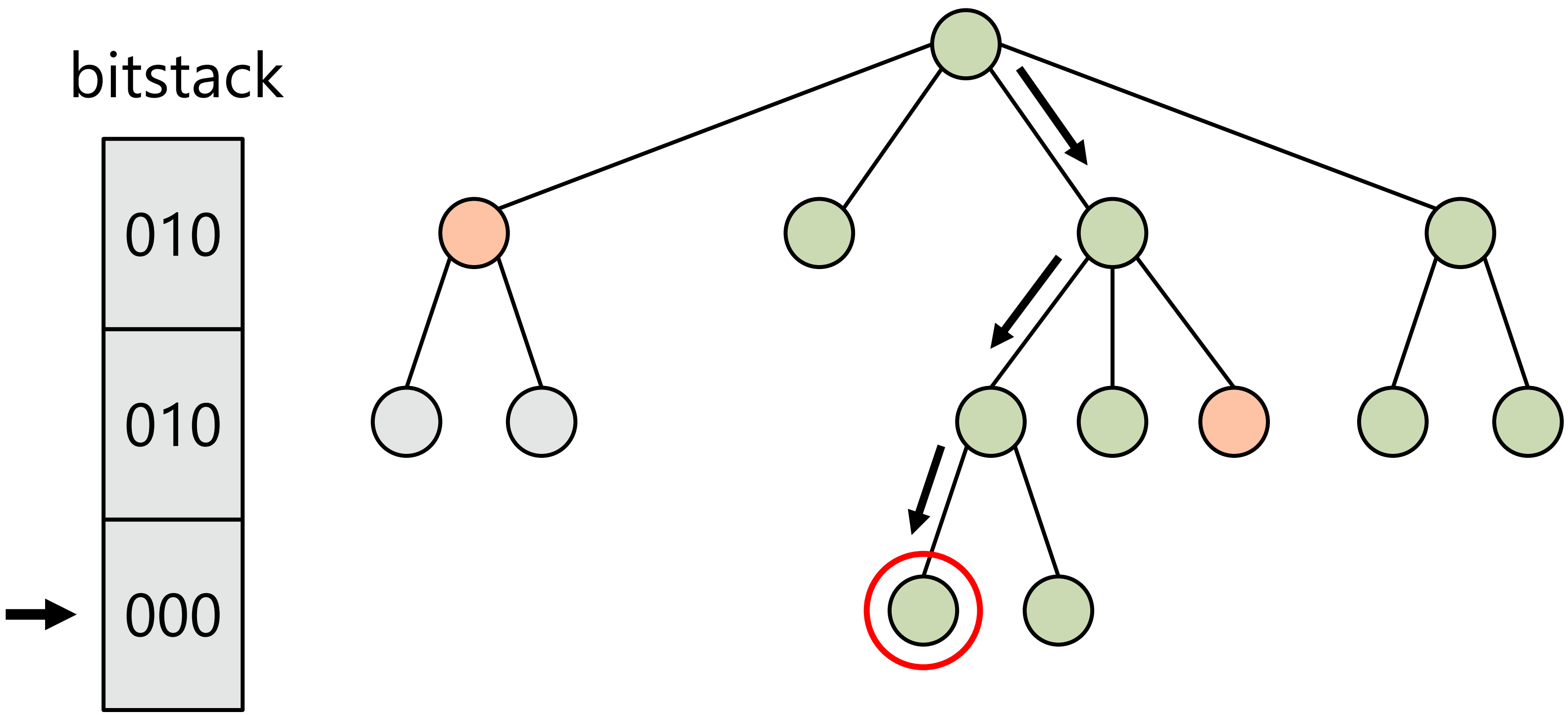
Traversal example



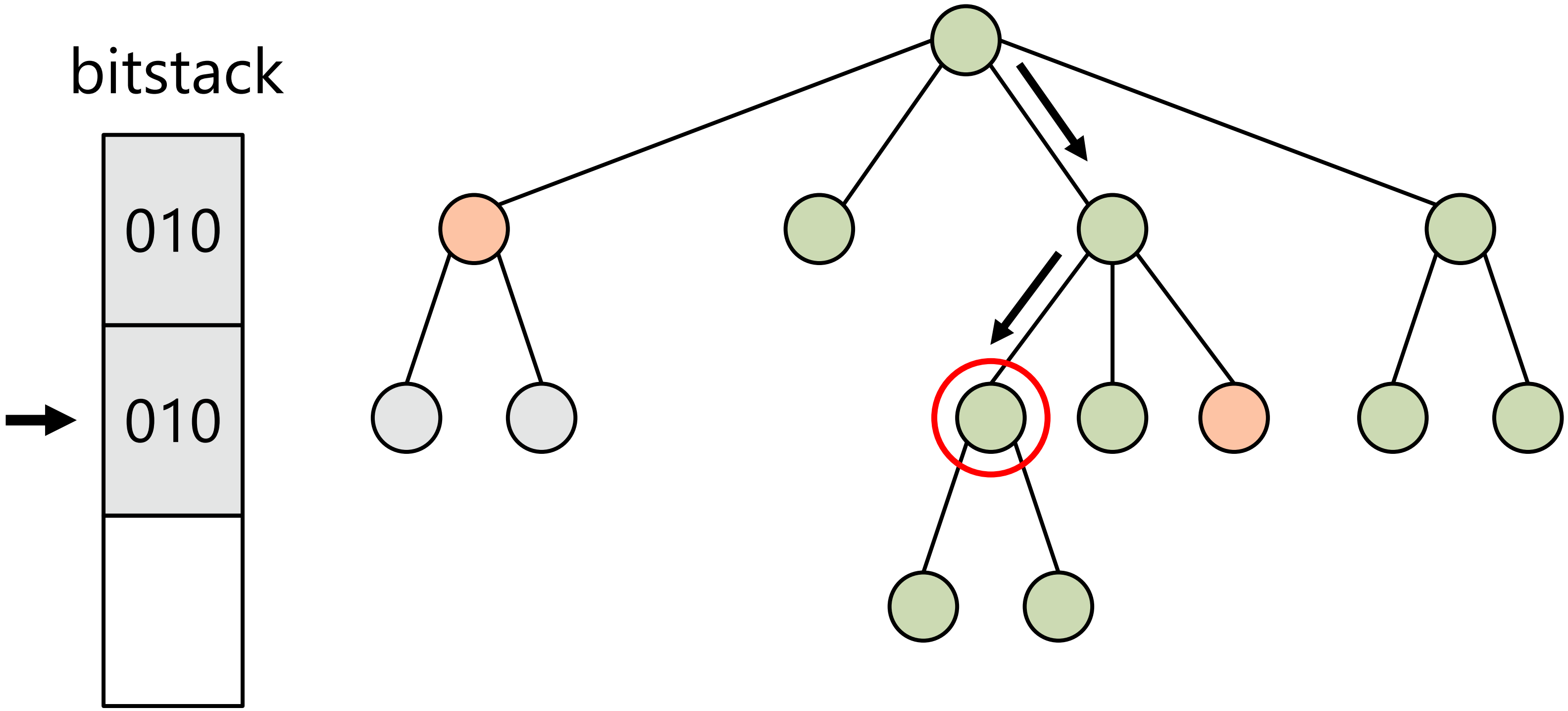
Traversal example



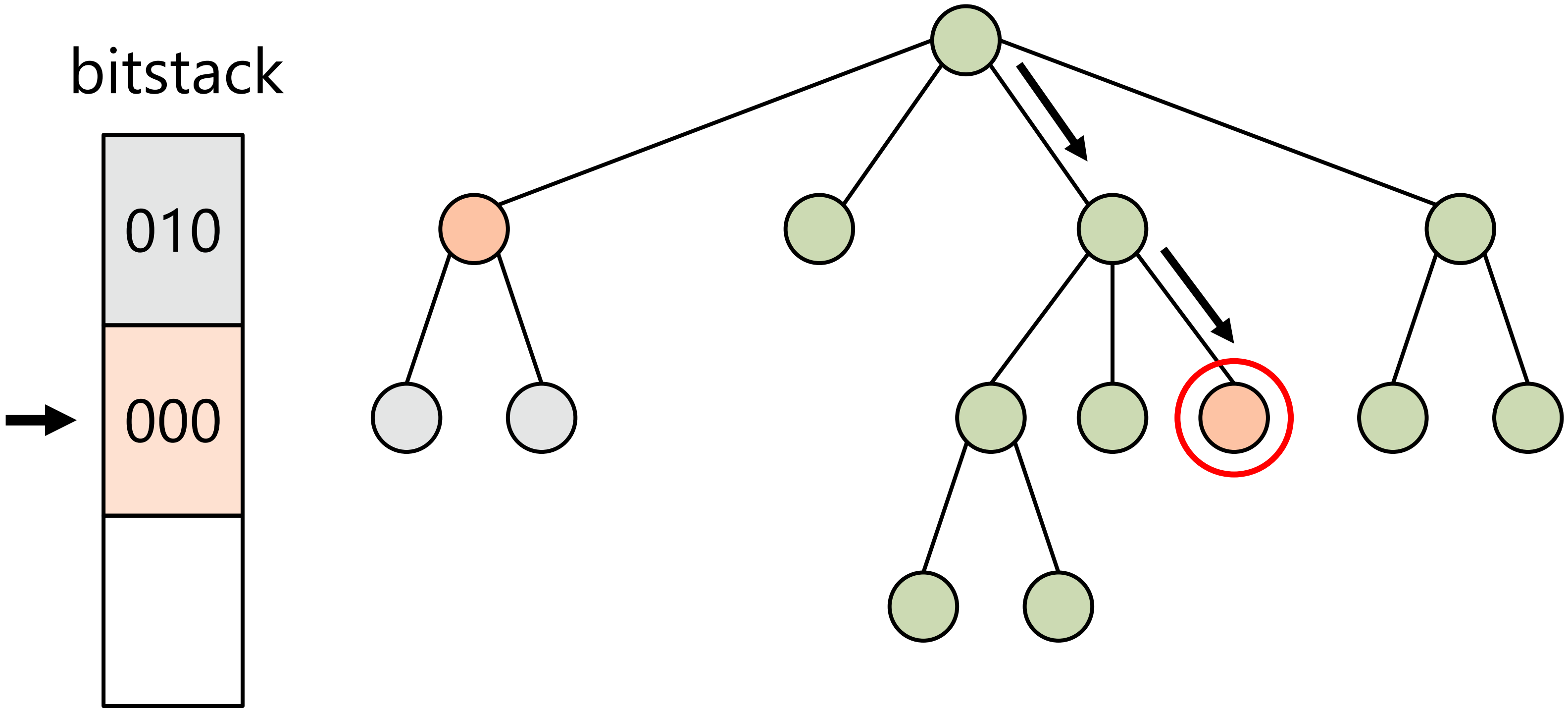
Traversal example



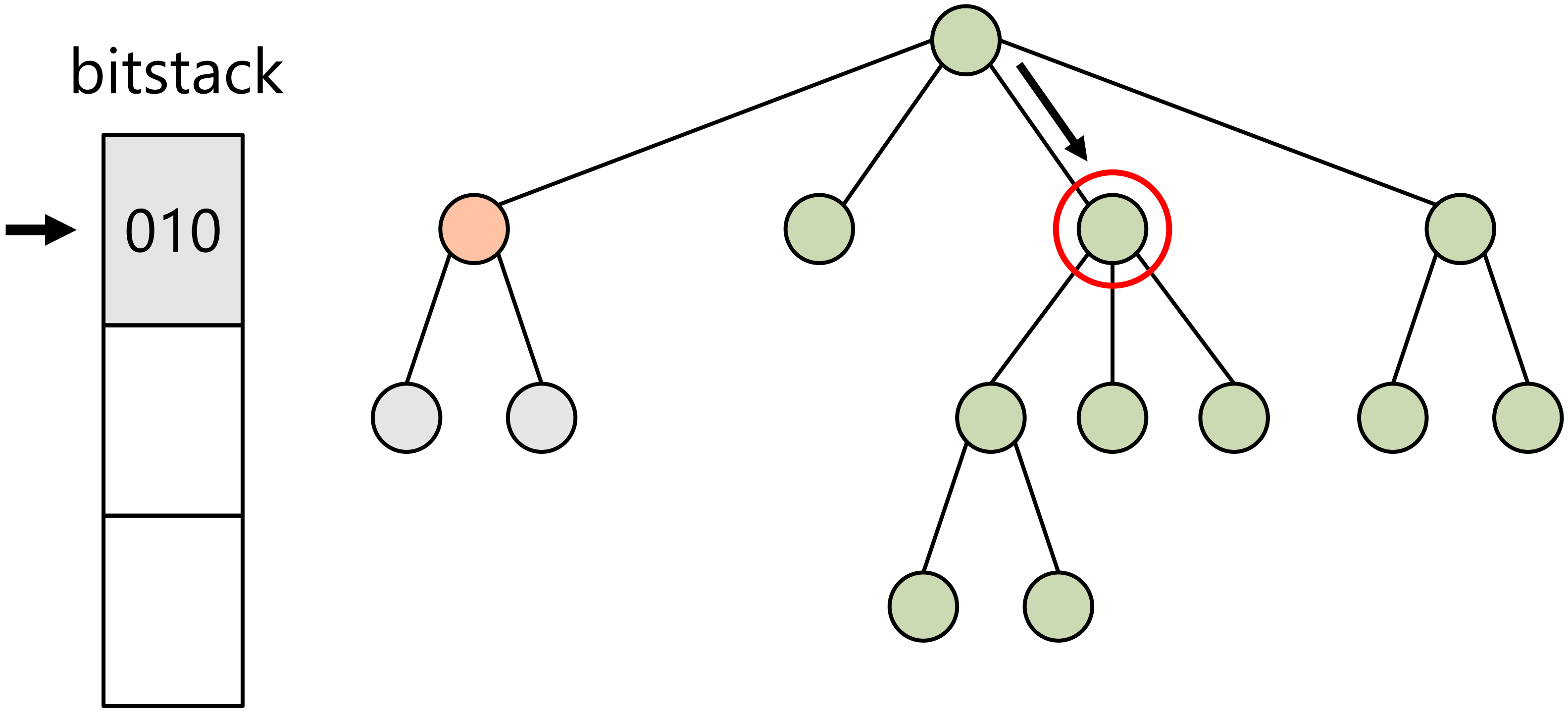
Traversal example



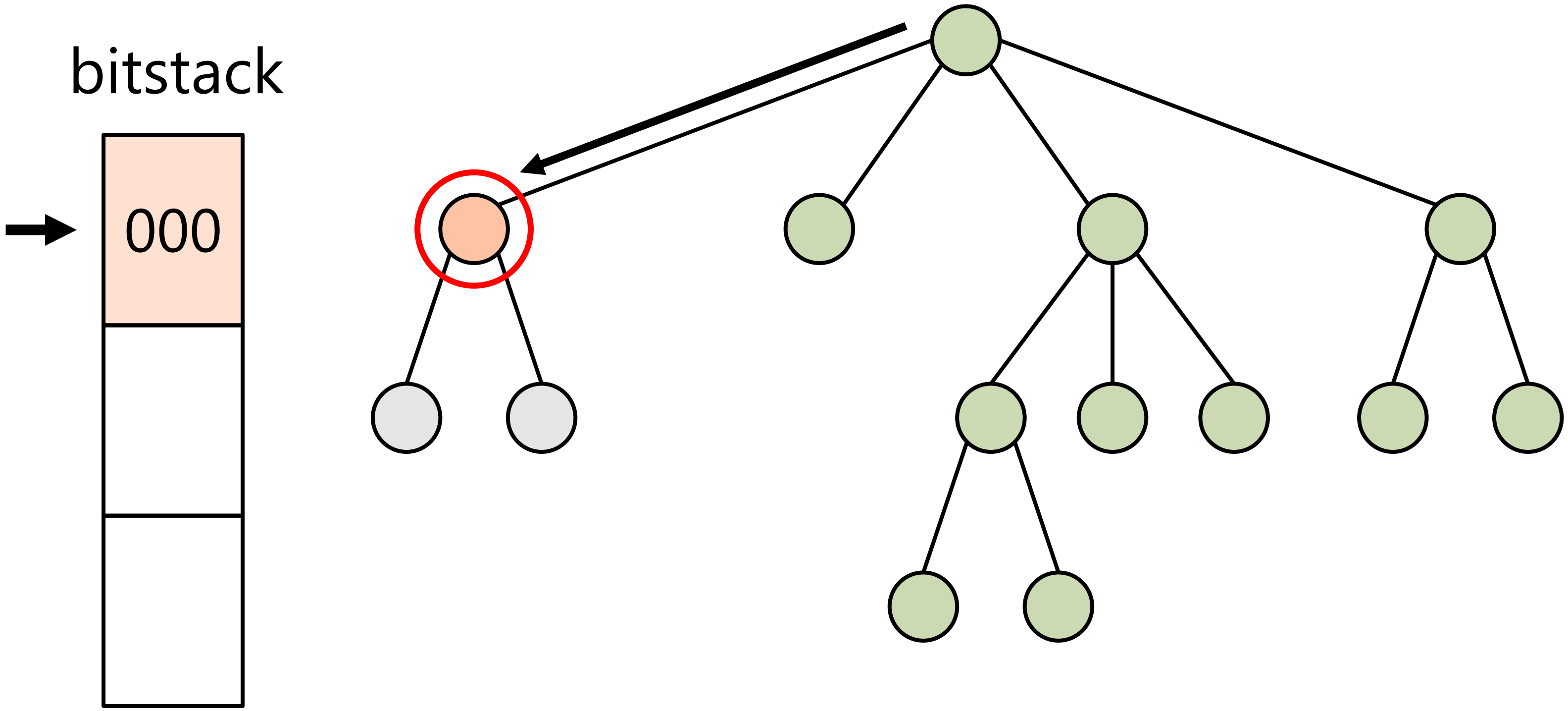
Traversal example



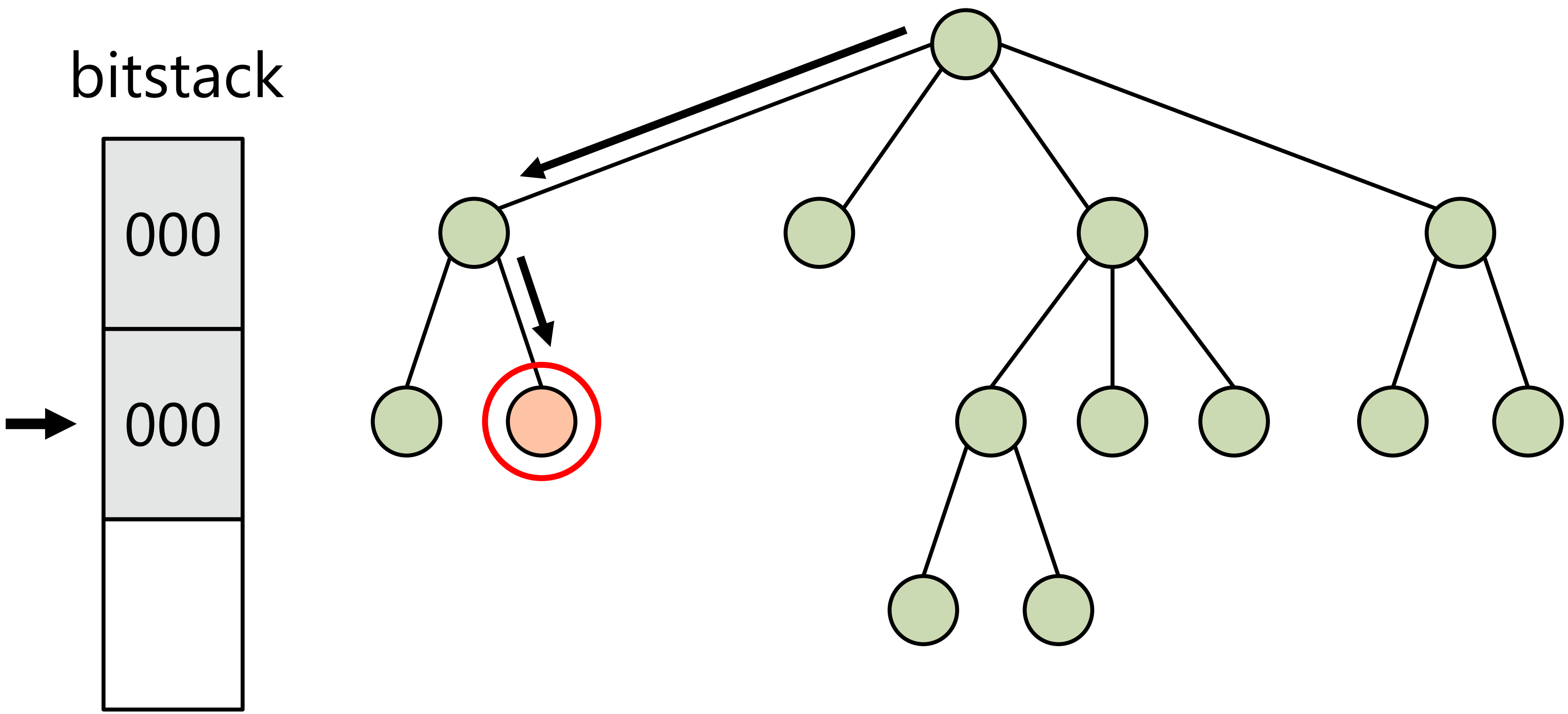
Traversal example



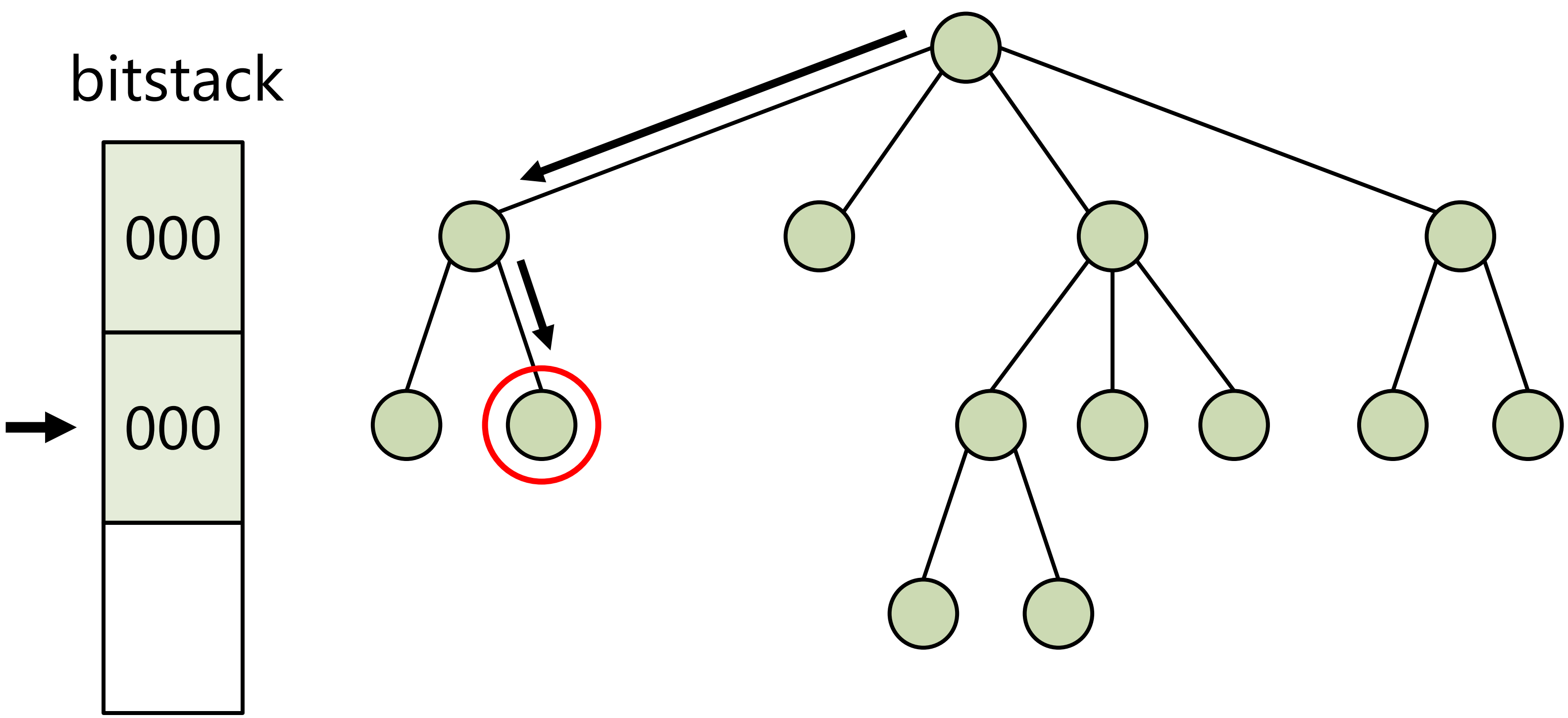
Traversal example



Traversal example



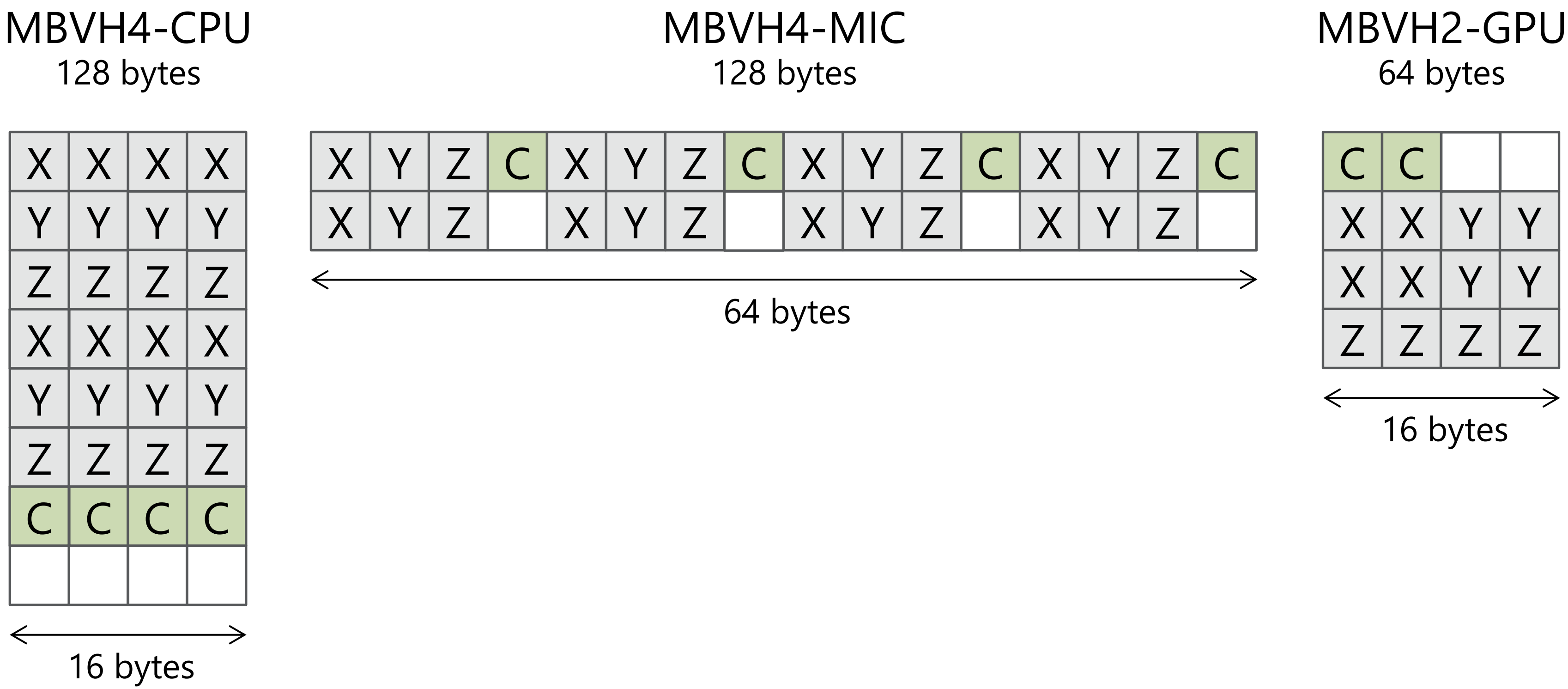
Traversal example



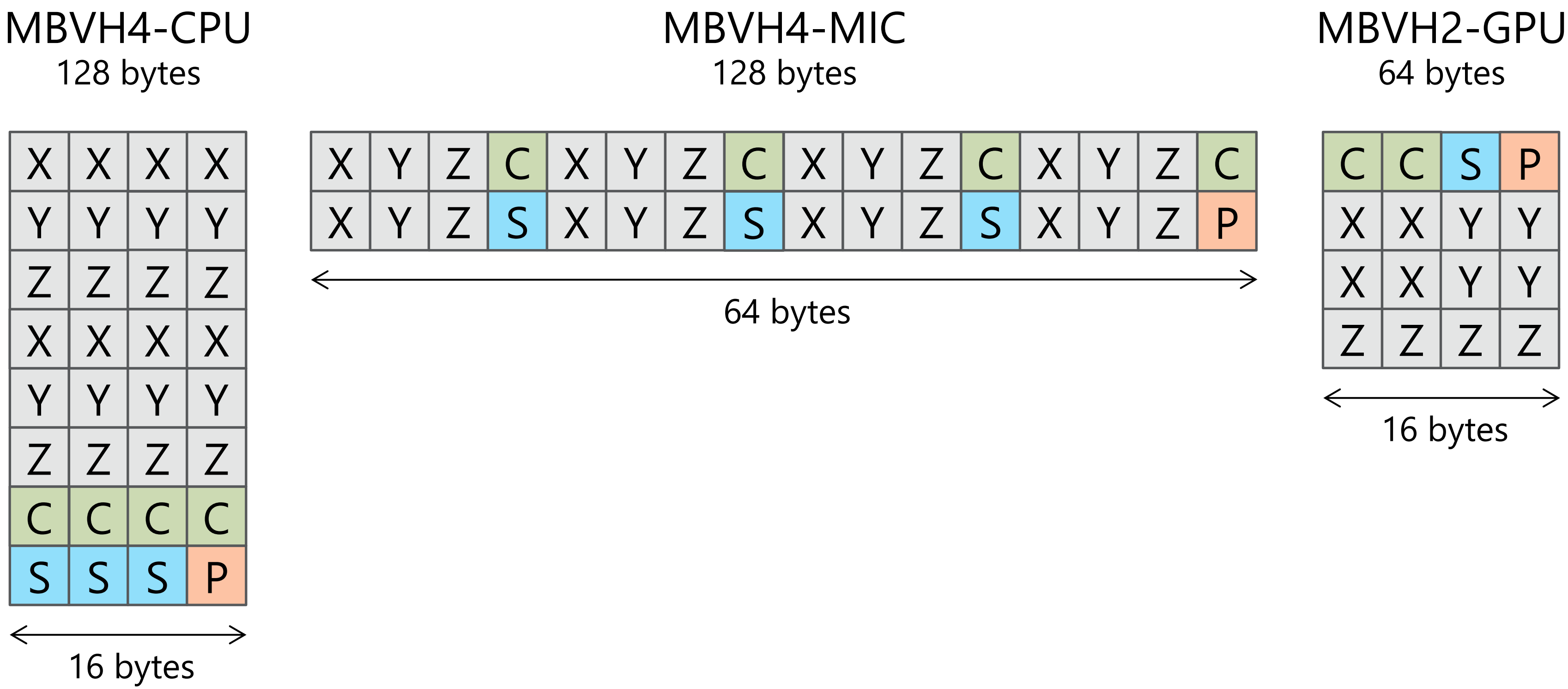
Implementation details

- CPU & MIC:
  - MBVH4 [Benthin et al. 2012]
  - 128-bit bitstack → two 64-bit registers
  - Skip codes computed using small lookup tables
- GPU:
  - MBVH2 [Aila et al. 2012]
  - 64-bit bitstack
- The additional pointers do not increase the node size

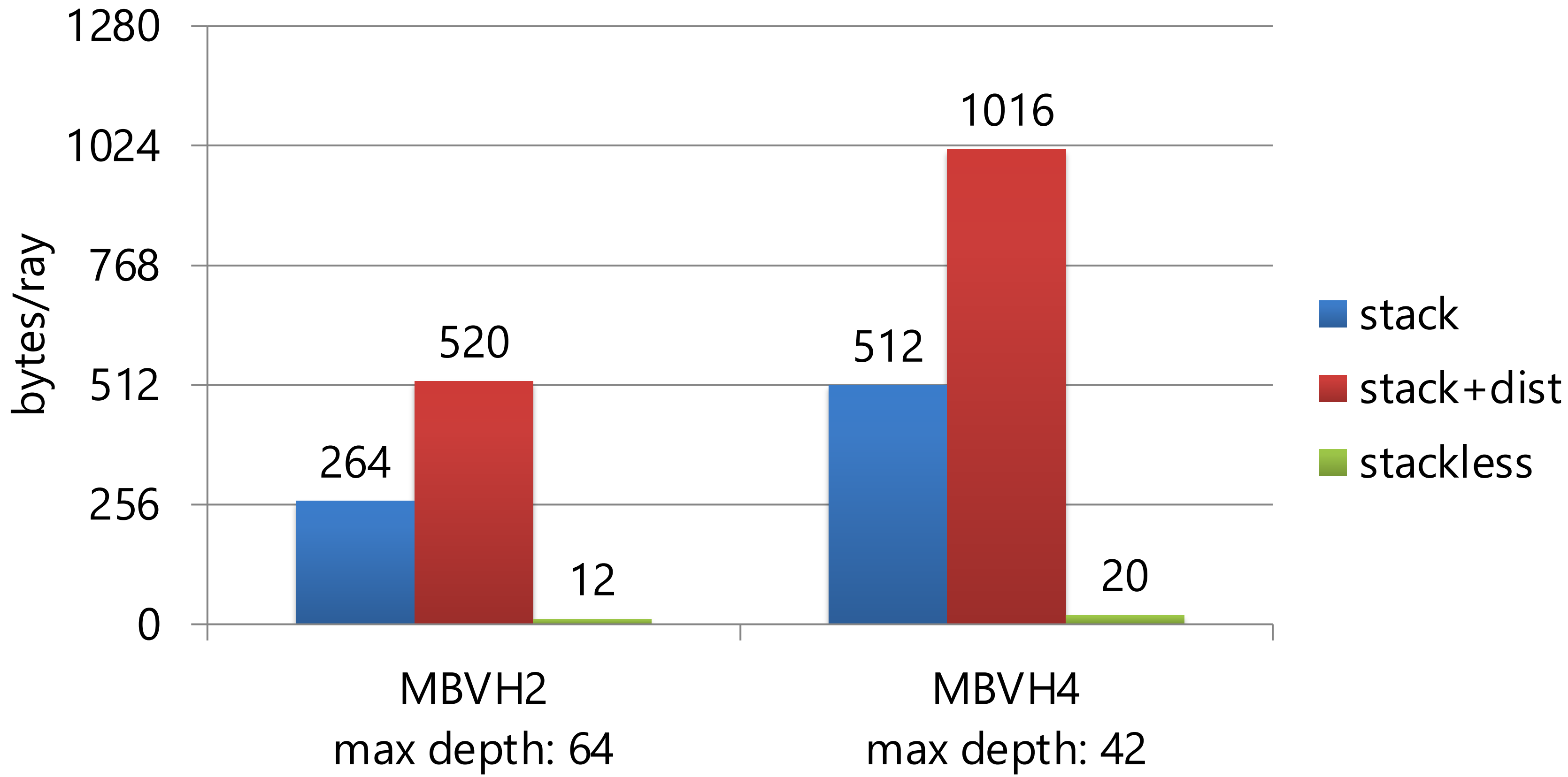
Node layout


MBVH4-CPU
128 bytes
X X X X
Y Y Y Y
Z Z Z Z
X X X X
Y Y Y Y
Z Z Z Z
C C C C
16 bytes
MBVH4-MIC
128 bytes
X Y Z C X Y Z C X Y Z C X Y Z C
X Y Z X Y Z X Y Z X Y Z
64 bytes
MBVH2-GPU
64 bytes
C C
X X Y Y
X X Y Y
Z Z Z Z
16 bytes

Node layout

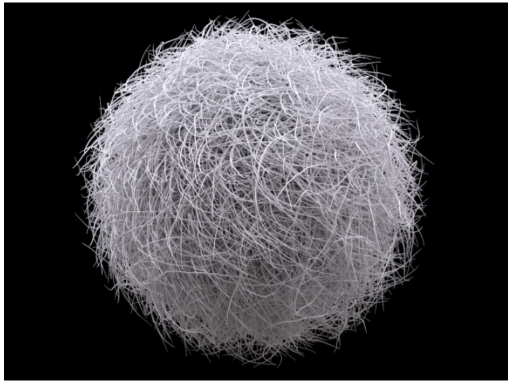
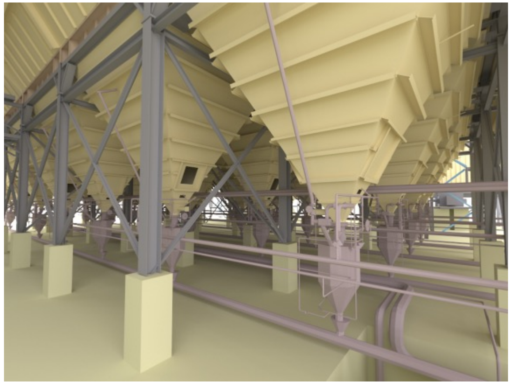

MBVH4-CPU
128 bytes
X X X X
Y Y Y Y
Z Z Z Z
X X X X
Y Y Y Y
Z Z Z Z
C C C C
S S S P
16 bytes
MBVH4-MIC
128 bytes
X Y Z C X Y Z C X Y Z C X Y Z C
X Y Z S X Y Z S X Y Z S X Y Z P
64 bytes
MBVH2-GPU
64 bytes
C C S P
X X Y Y
X X Y Y
Z Z Z Z
16 bytes

Traversal state size



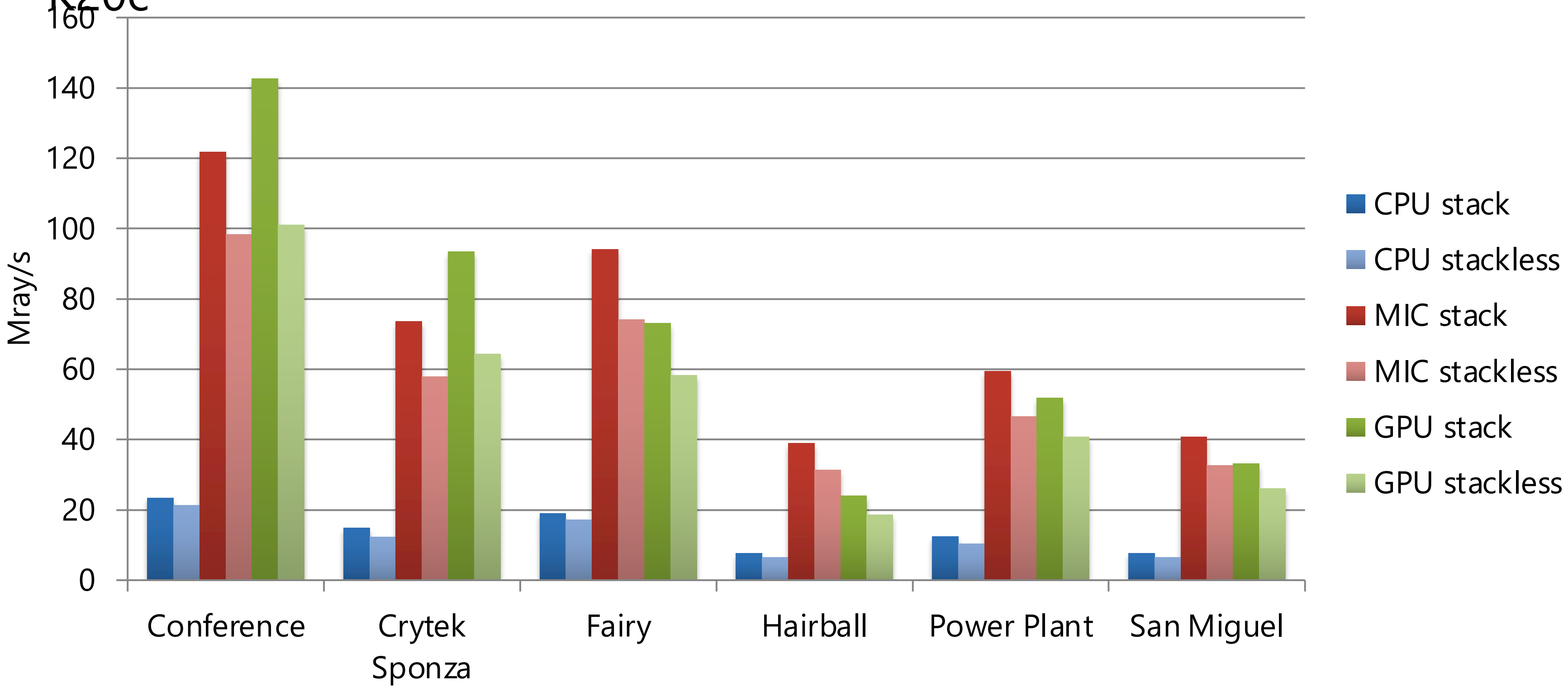
Test scenes




Diffuse path tracing performance

- CPU: Intel Core i7-3770, MIC: Intel Xeon Phi SE10P, GPU: NVIDIA Tesla K20c



Conclusions

- Performance overhead: current hardware, traditional ray tracing
  - More complex traversal logic
  - More irregular memory accesses
  - Slightly higher number of intersections

- Significantly smaller traversal state
  - About 10× more rays in flight within the same memory budget
  - Example: 128 MB (ray data and traversal state)
    - Stack-based: 240K rays
    - Stackless: 2.2M rays

Possible use cases, future work

- Future hardware
  - GPUs
  - Ray tracing hardware
- Coherence extraction [Aila & Karras 2010]
  - Decrease bandwidth usage, increase SIMD efficiency
  - Many rays in flight
  - More than one ray per hardware thread
  - Suspend/resume traversal
- Out-of-core [Pharr et al. 1997]

Thank you!

MBVH8 child hits