Dependency Graph Scheduling in a Ray Tracing Architecture

Susan Frank and Arie Kaufman

Center for Visual Computing

Department of Computer Science

State University of New York at Stony Brook, USA

Why use ray tracing?

- + Global illumination
- + Unifying technique for volumes
  - + Processing and rendering
  - + Triangles, points, implicit surfaces etc.
- + Early ray termination
- + Scene complexity independence
- + Inherently parallel

Why not use ray tracing?

- Non-uniform memory access
- Need spatial coherence

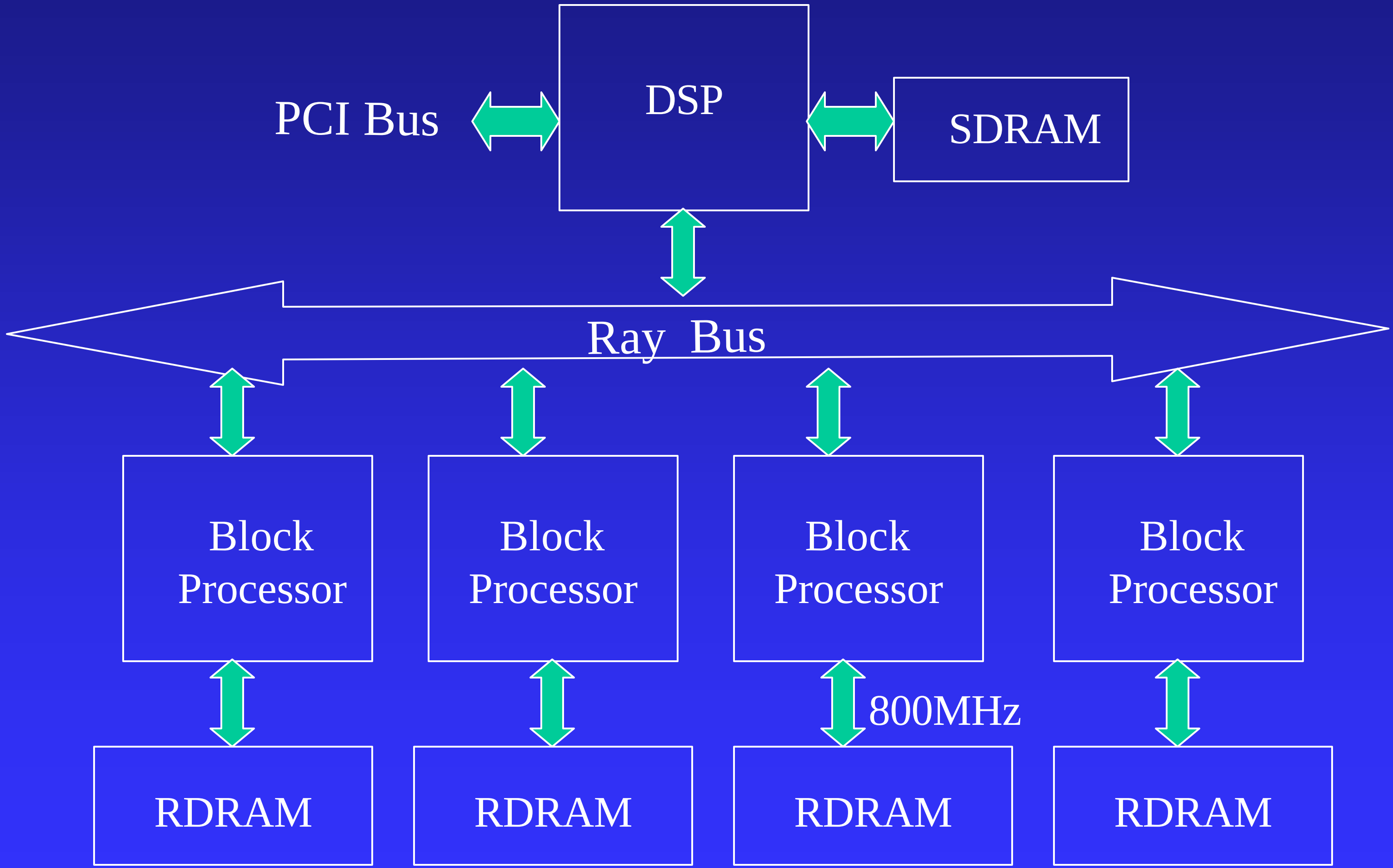
Ray Tracing Systems

- Ray Queues [Pharr et al. ‘97]
- GI-Cube [Dachille, Kaufman ’00]
- Pyramid clipping and octree subdivision [Reinhard et al. ’99]
- Kilauea system [Nishimura et al. ’01]
- AR250 [ART ’99]
- Coherent Ray Tracing [Wald, et al. 2001]

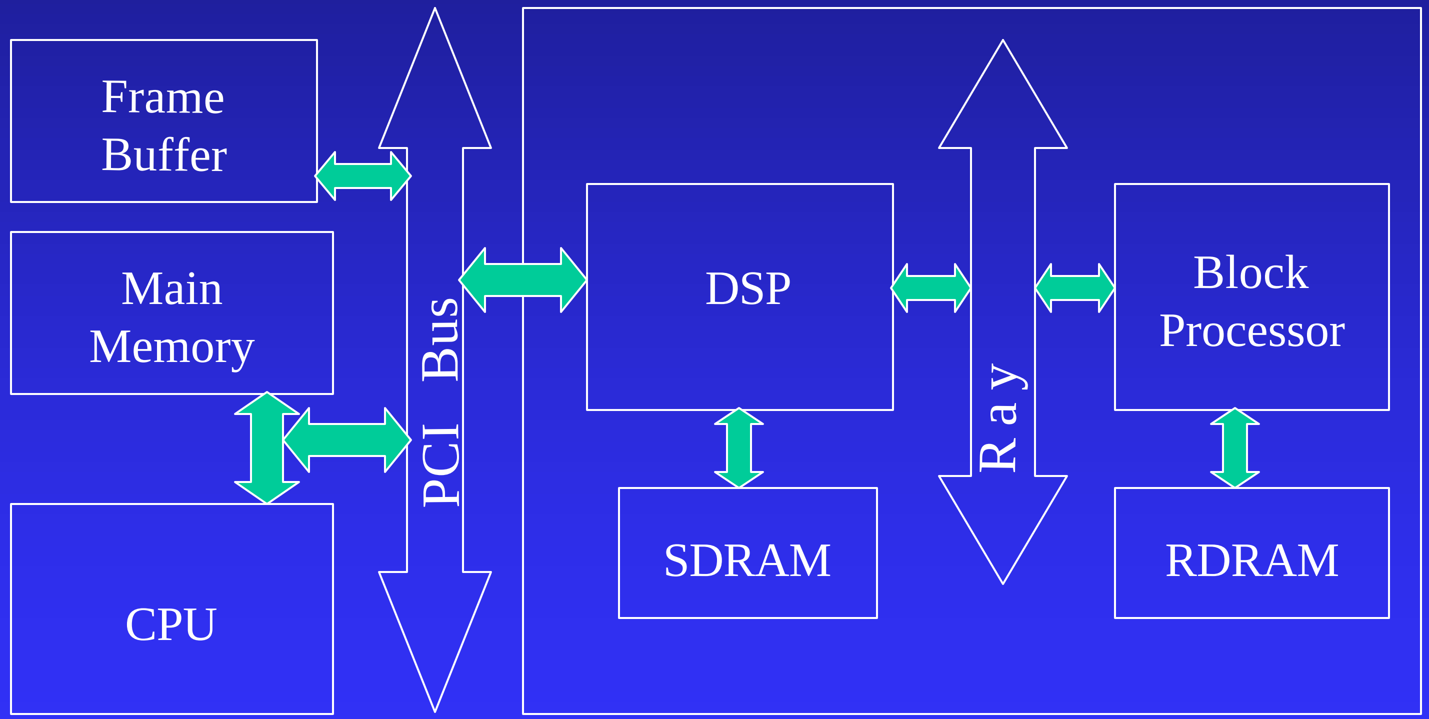
Outline

- **Our System**
- Cell Tree
- Dependency Graph Scheduling
- Peel Algorithm
- Results

GI-Cube Architecture

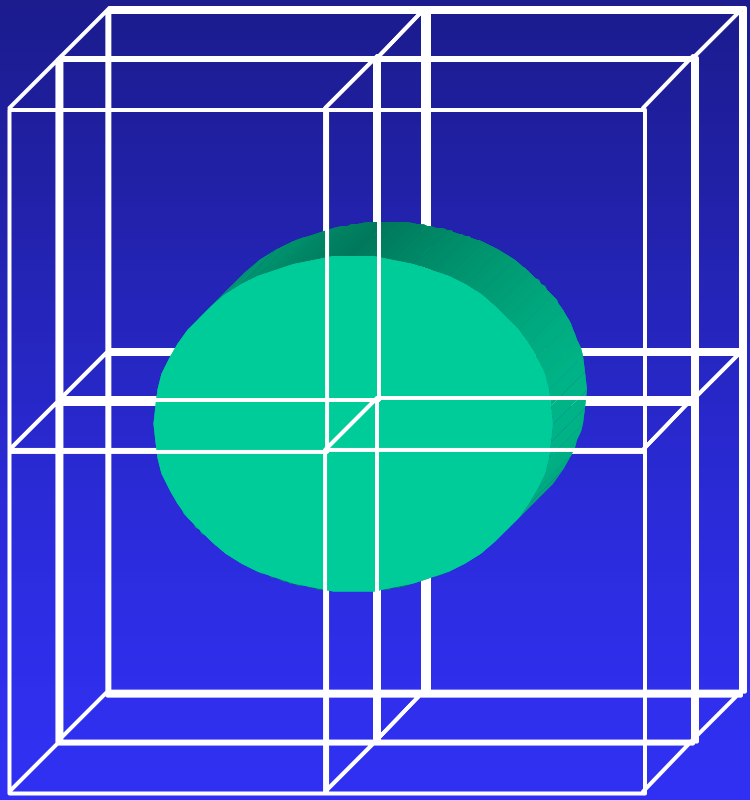


Single Processor



Ray Queues

- Maintain ray queue for each cell
- Process all rays while a cell is in cache
- Spawned rays added to queue of next intersected cell



Subdivide Volume Into Cells

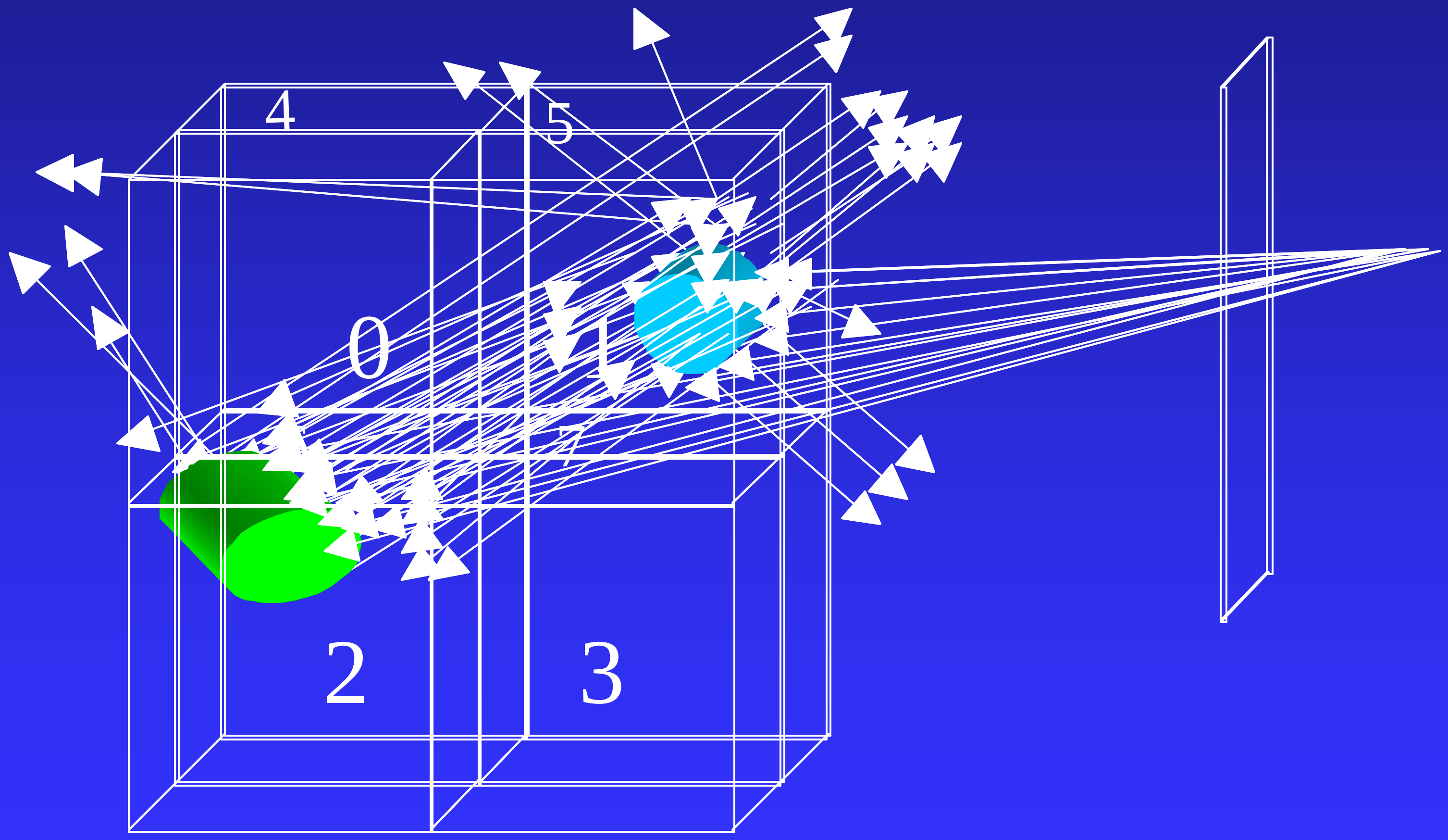
Our Scheduling Schema

- Cell Tree
  - Ray-cell dependencies from frame i used to create schedule for frame i+1
- Max Work
  - First frame (ray dependencies unknown) and if rays remain after Cell Tree schedule
- Any level of the memory hierarchy
- Cell size set to memory size

Outline

- Our System
- **Cell Tree**
- Dependency Graph Scheduling
- Peel Algorithm
- Results

Psuedo-Random Ray Traversal



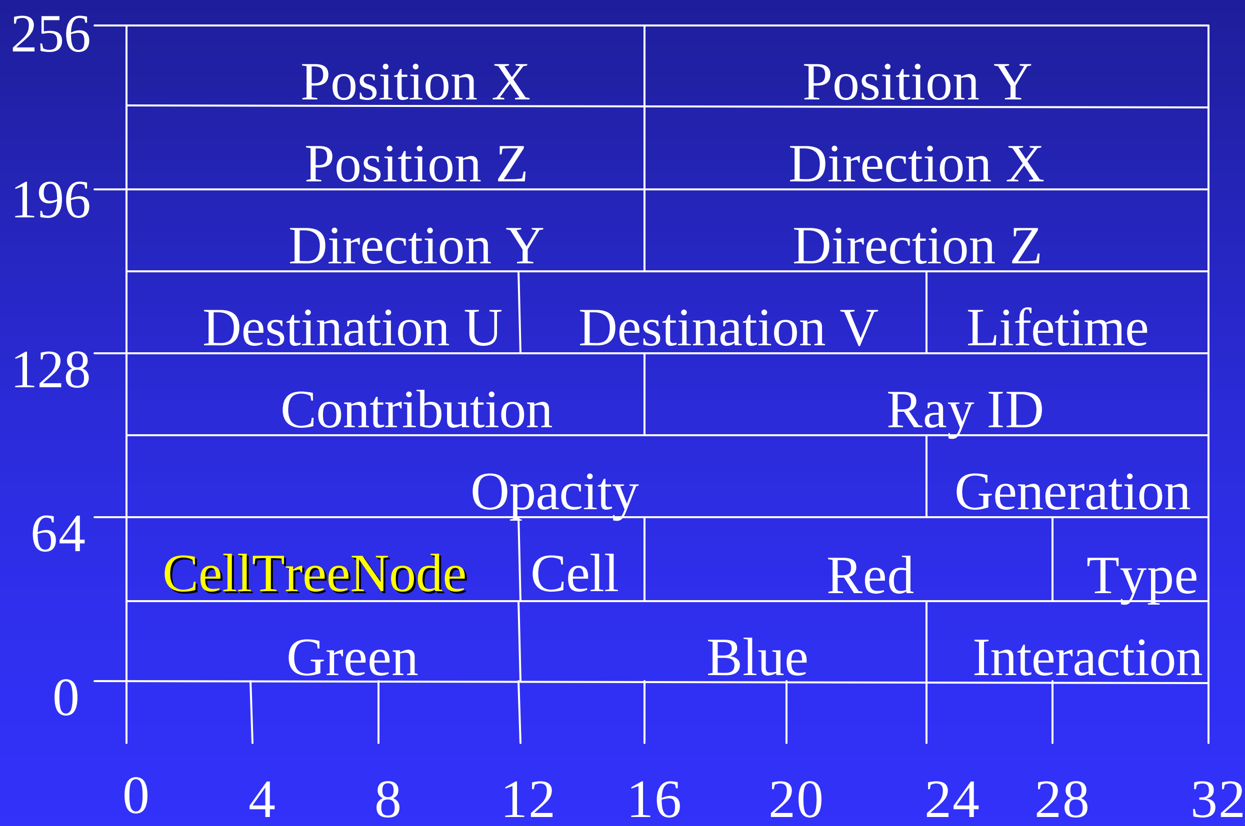
Cell Tree

- Gathers clusters of rays as they're generated
- Concisely describes all ray-cell dependencies of completed frame
  - 100 times fewer nodes than rays represented
- Predict better schedule for next frame

Cell Tree Creation

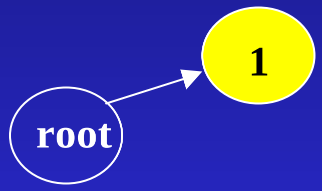
1. Initialize
2. Maintain *CellTreeNode* in Ray Packet
3. Add nodes to Cell Tree as needed to represent ray-cell dependencies

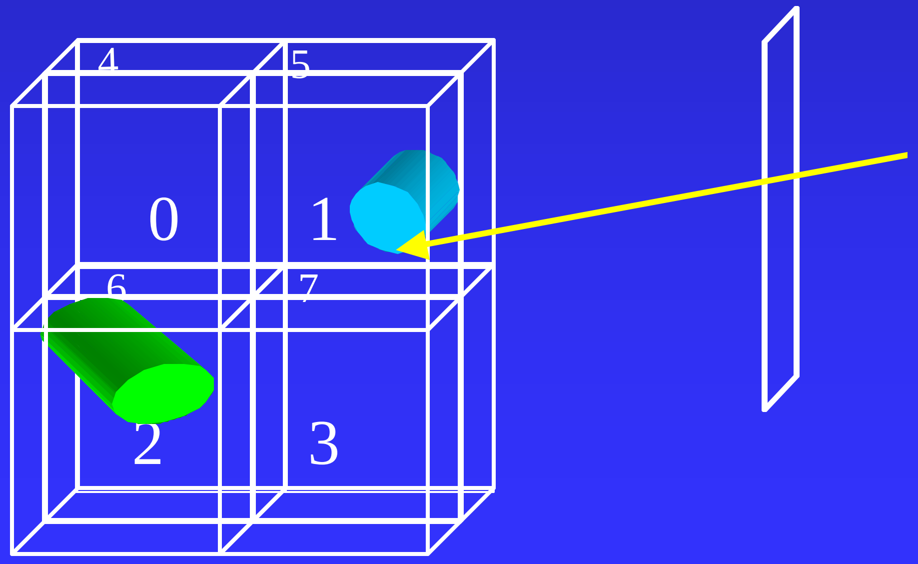
Ray Packet



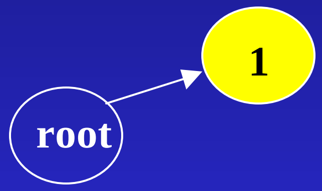
256
Position X
Position Y
Position Z
Direction X
196
Direction Y
Direction Z
Destination U
Destination V
Lifetime
128
Contribution
Ray ID
Opacity
Generation
64
CellTreeNode
Cell
Red
Type
Green
Blue
Interaction
0
0
4
8
12
16
20
24
28
32

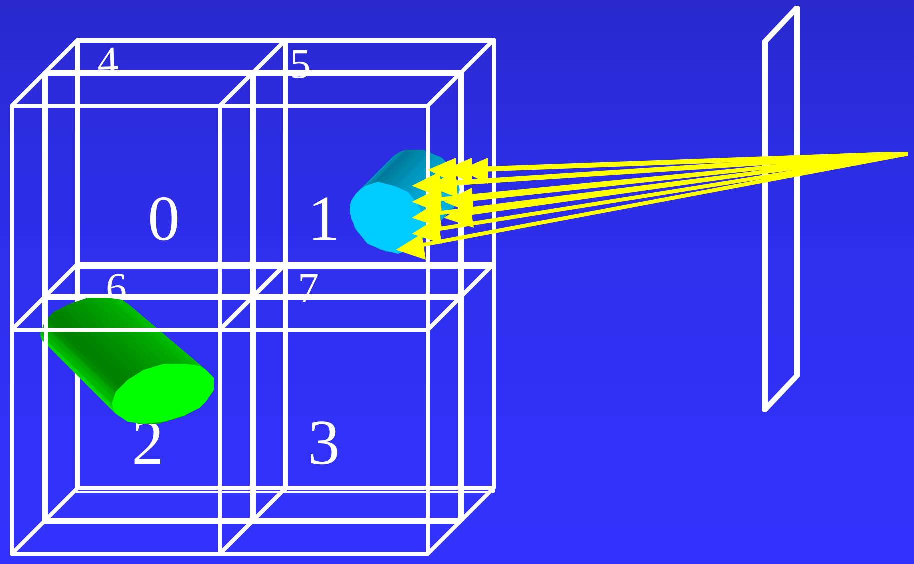
Initialization



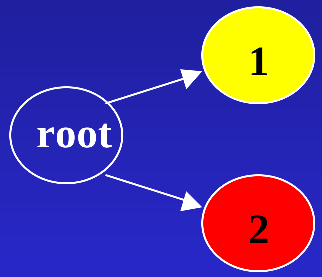


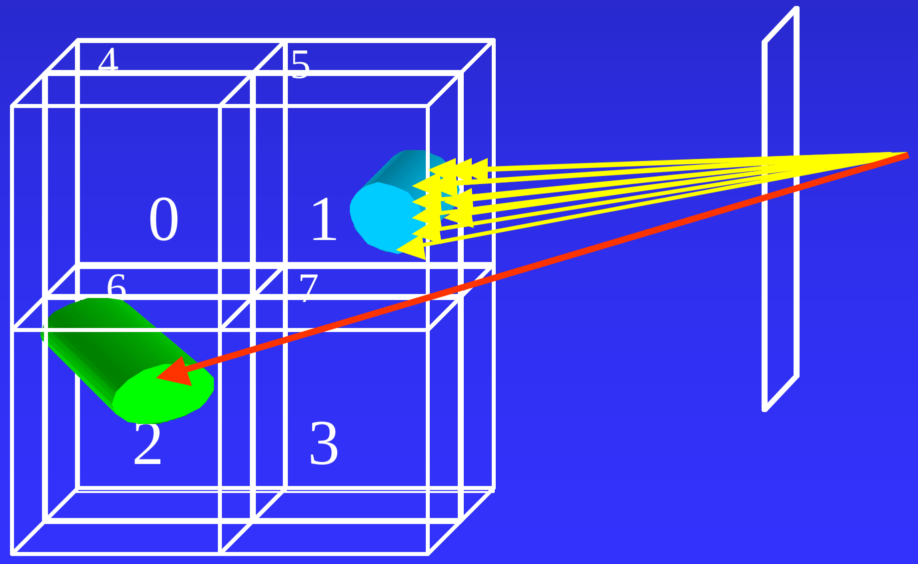
Initialization



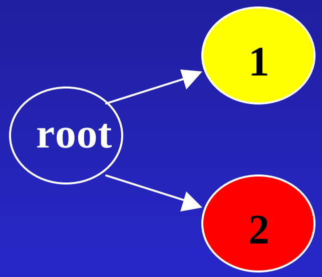


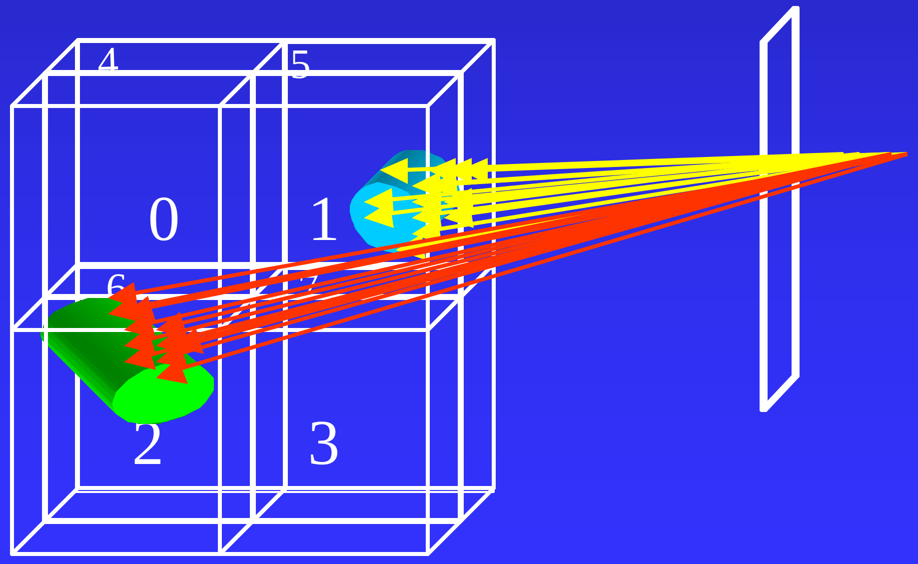
Initialization



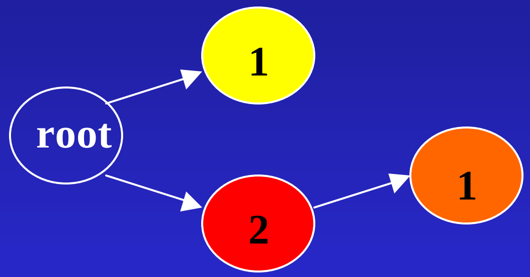


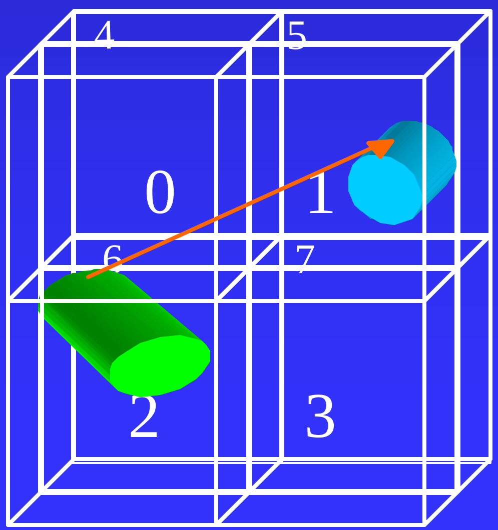
Initialization



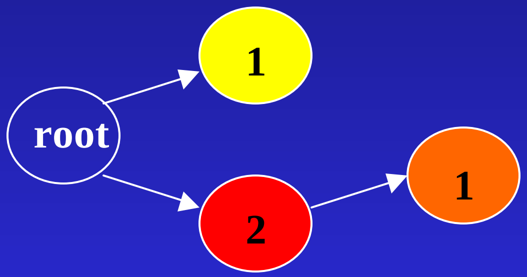


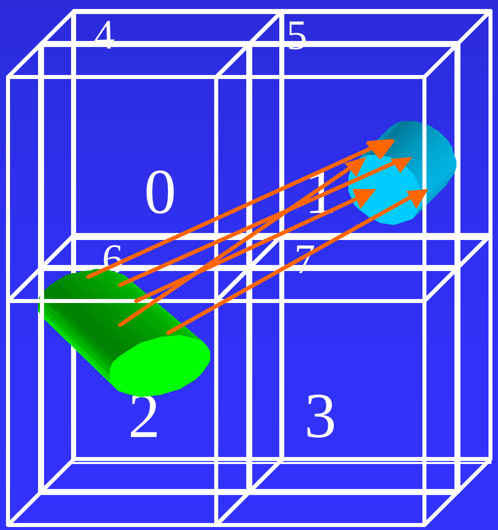
Reflections Refractions and Shadows



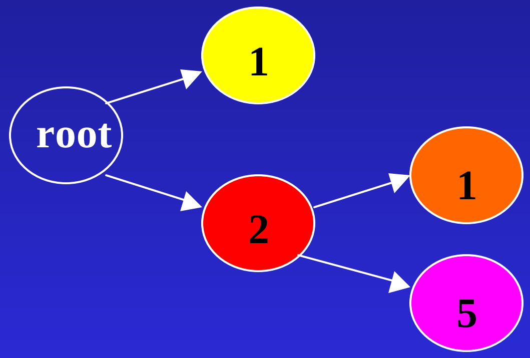


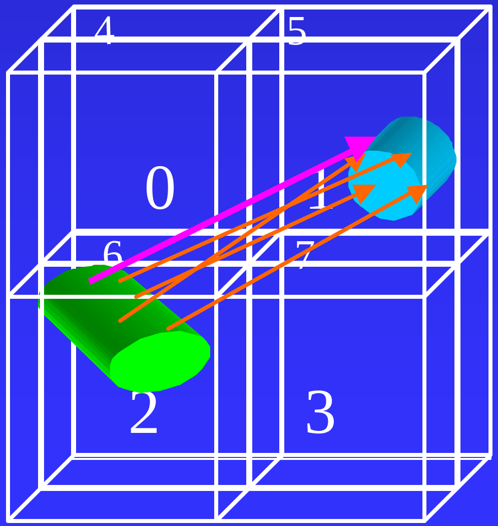
Reflections Refractions and Shadows



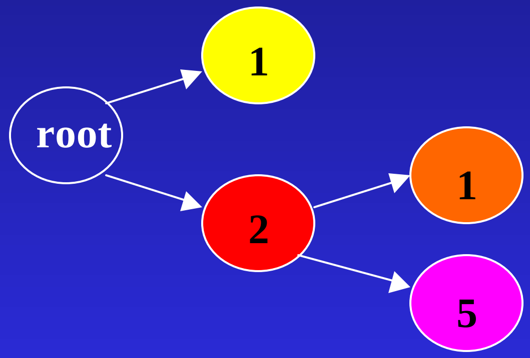


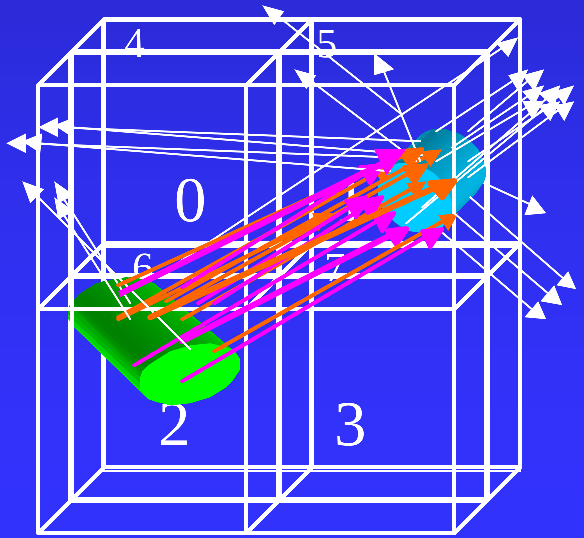
Reflections Refractions and Shadows



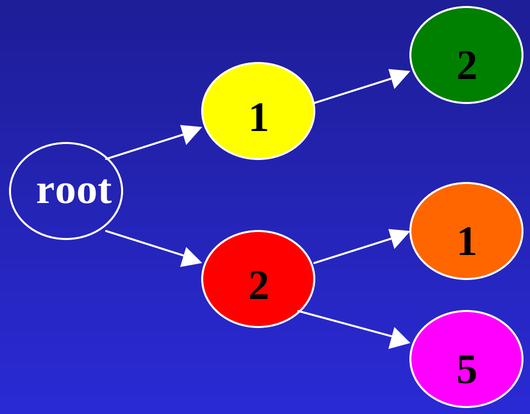


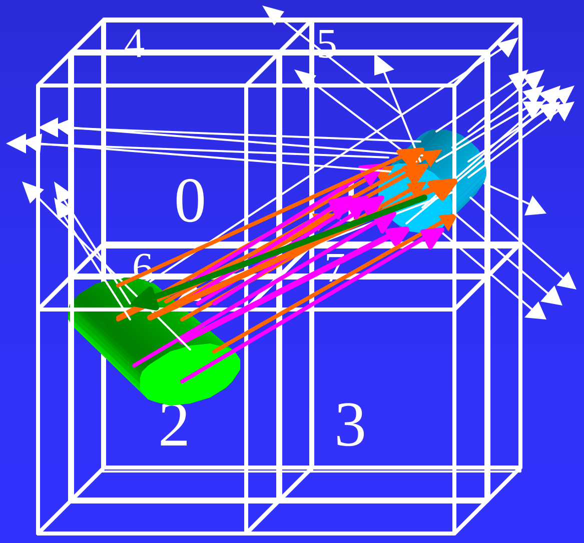
Reflections Refractions and Shadows



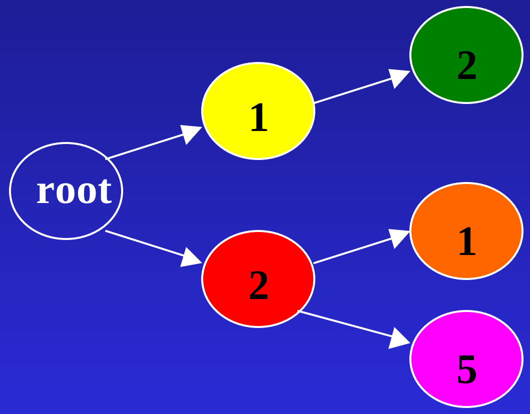


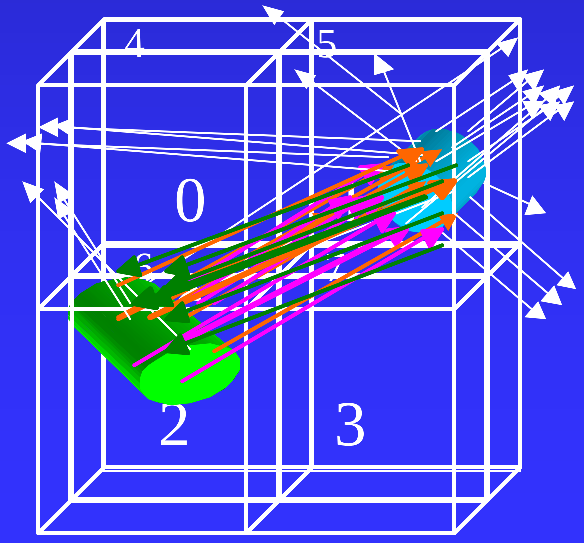
Reflections Refractions and Shadows



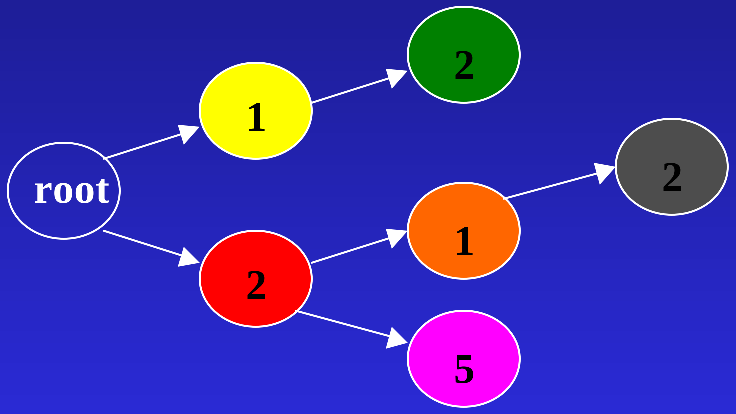


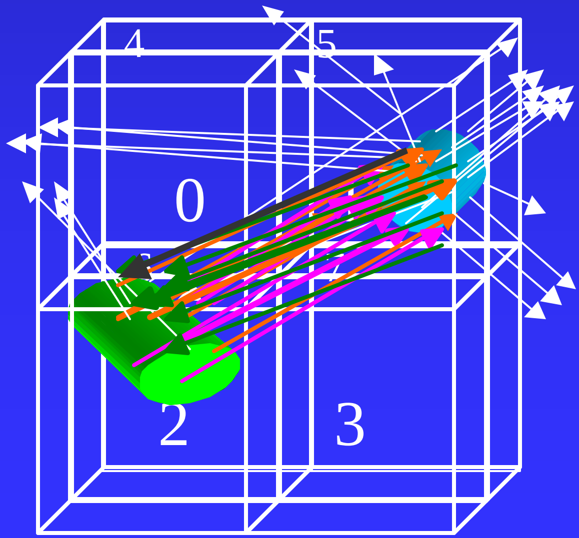
Reflections Refractions and Shadows



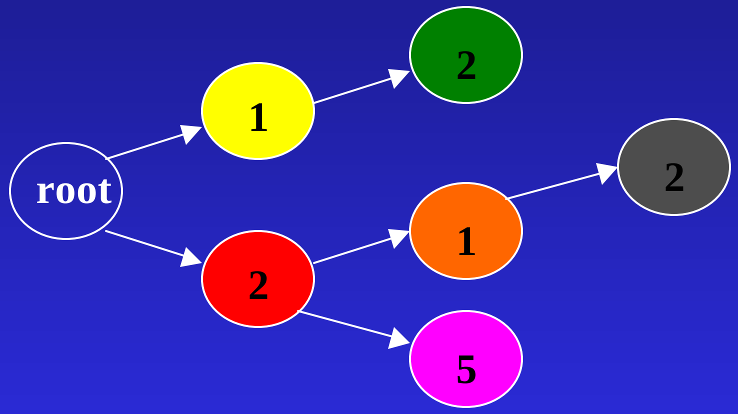


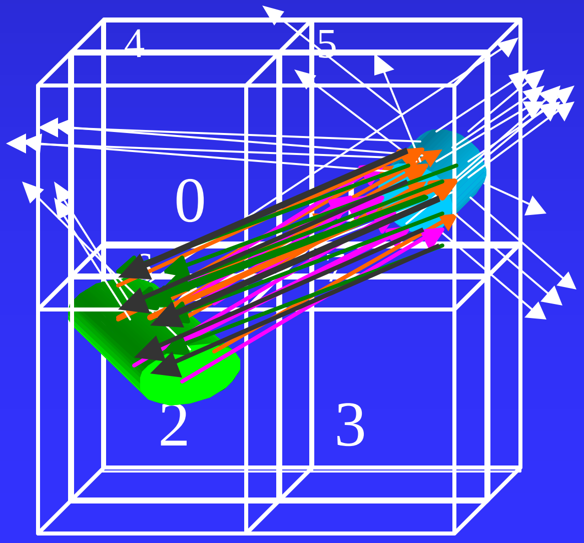
Secondary Reflections



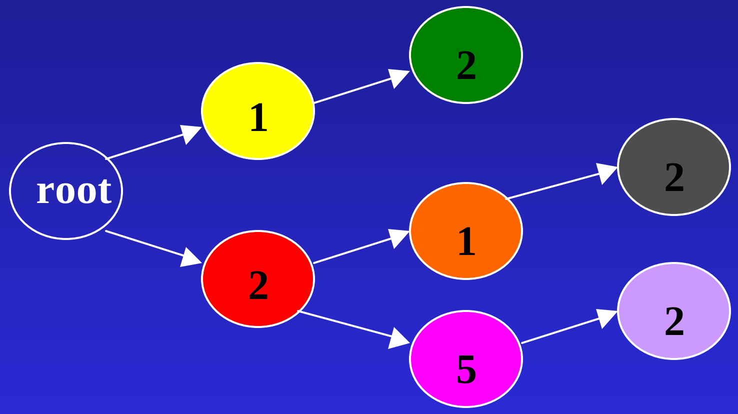


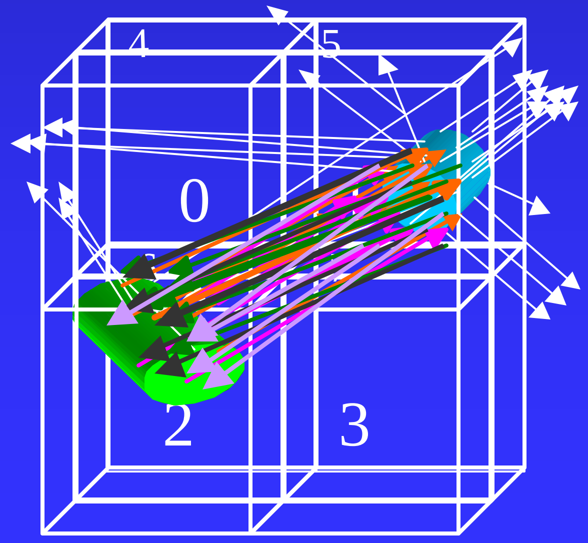
Secondary Reflections



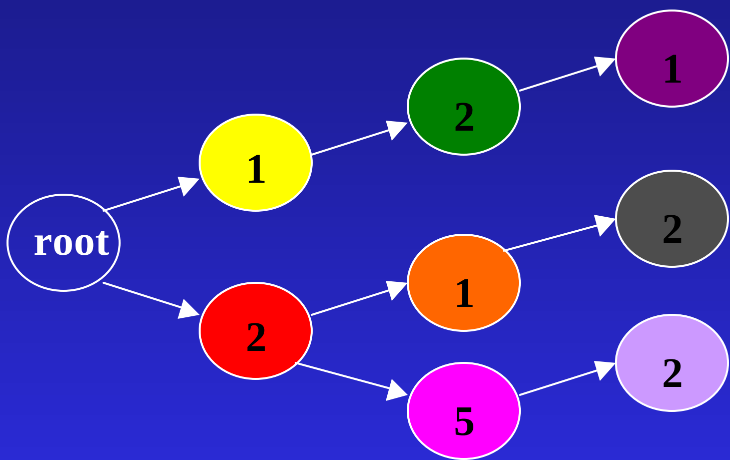


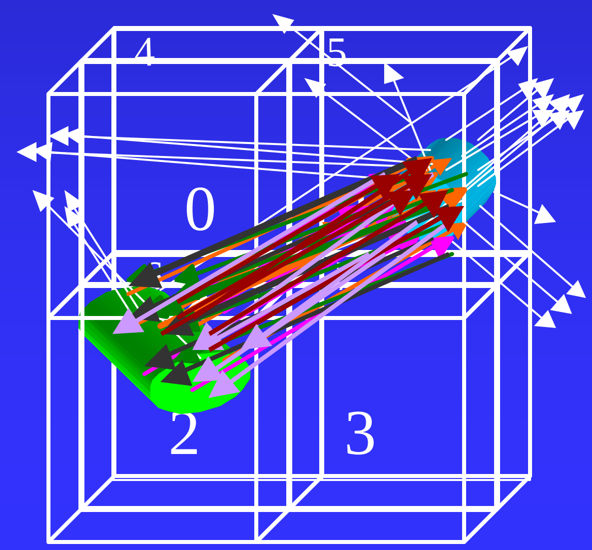
Secondary Reflections





Secondary Reflections



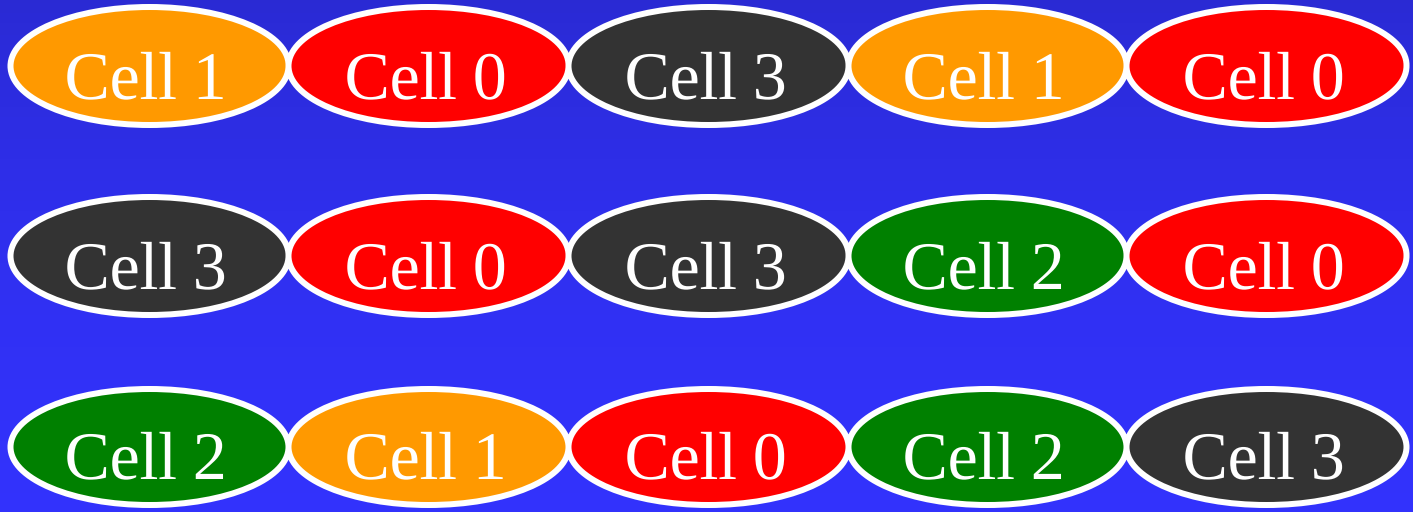


Outline

- Our System
- Cell Tree
- **Dependency Graph Scheduling**
- Peel Algorithm
- Results

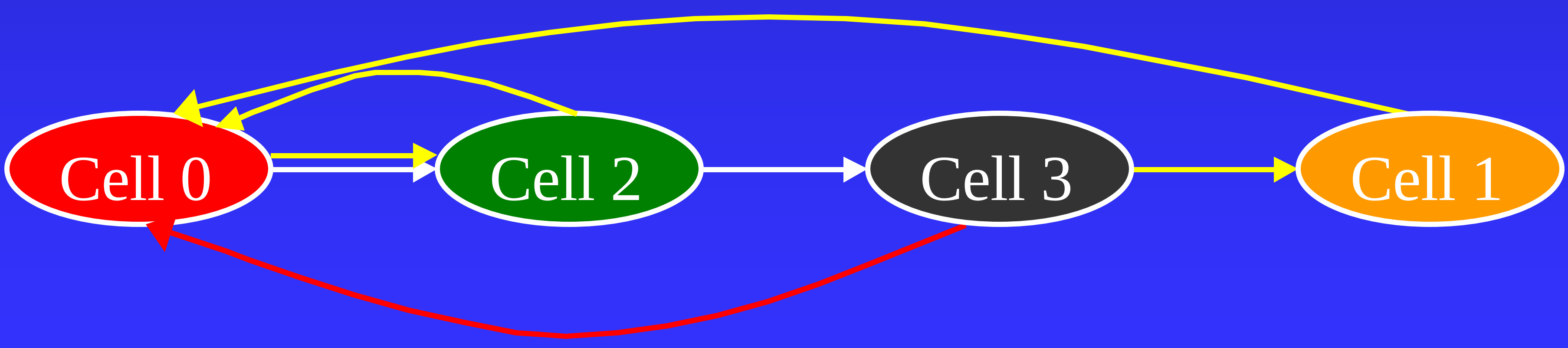
Task Scheduling Problem

- Goal - minimize memory fetches
- Equivalently - minimize color changes in super sequence which contains all sequences

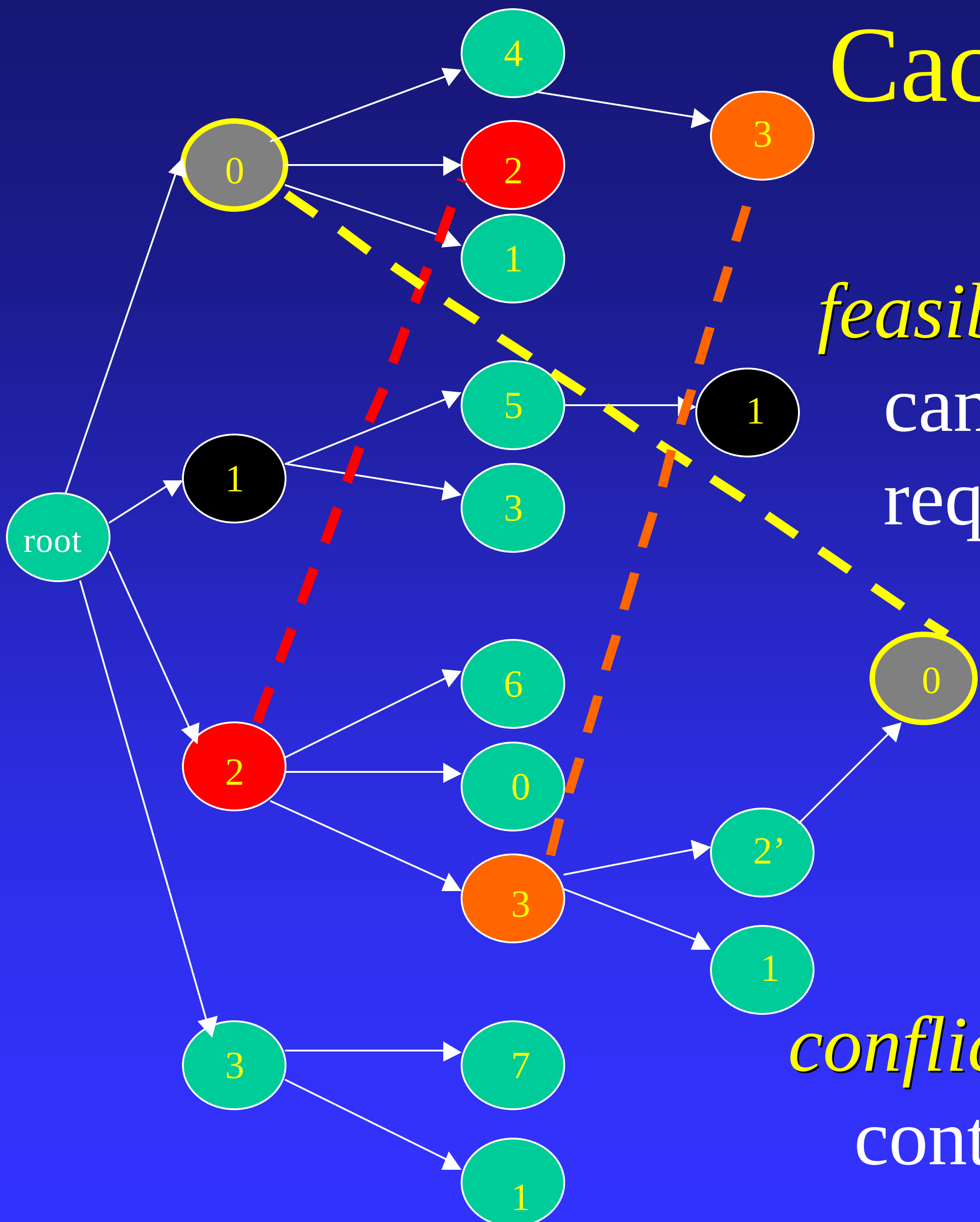


Cyclic Dependency Graphs

- Rays must visit cells in a particular order
- A ray may revisit a cell several times
- Sub-volume must be cached each time



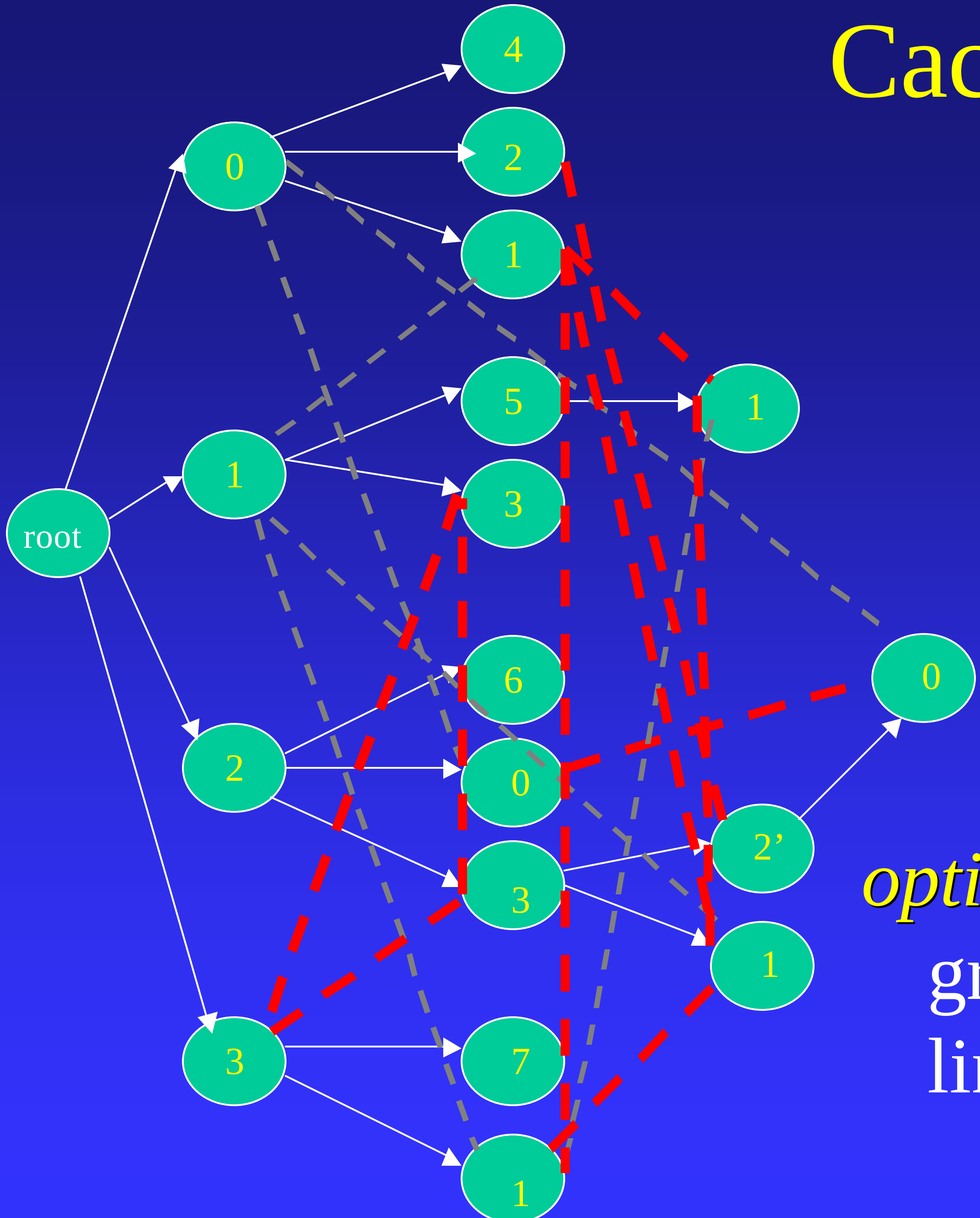
Cache Saving Links



feasible schedule - all rays can be processed in required order

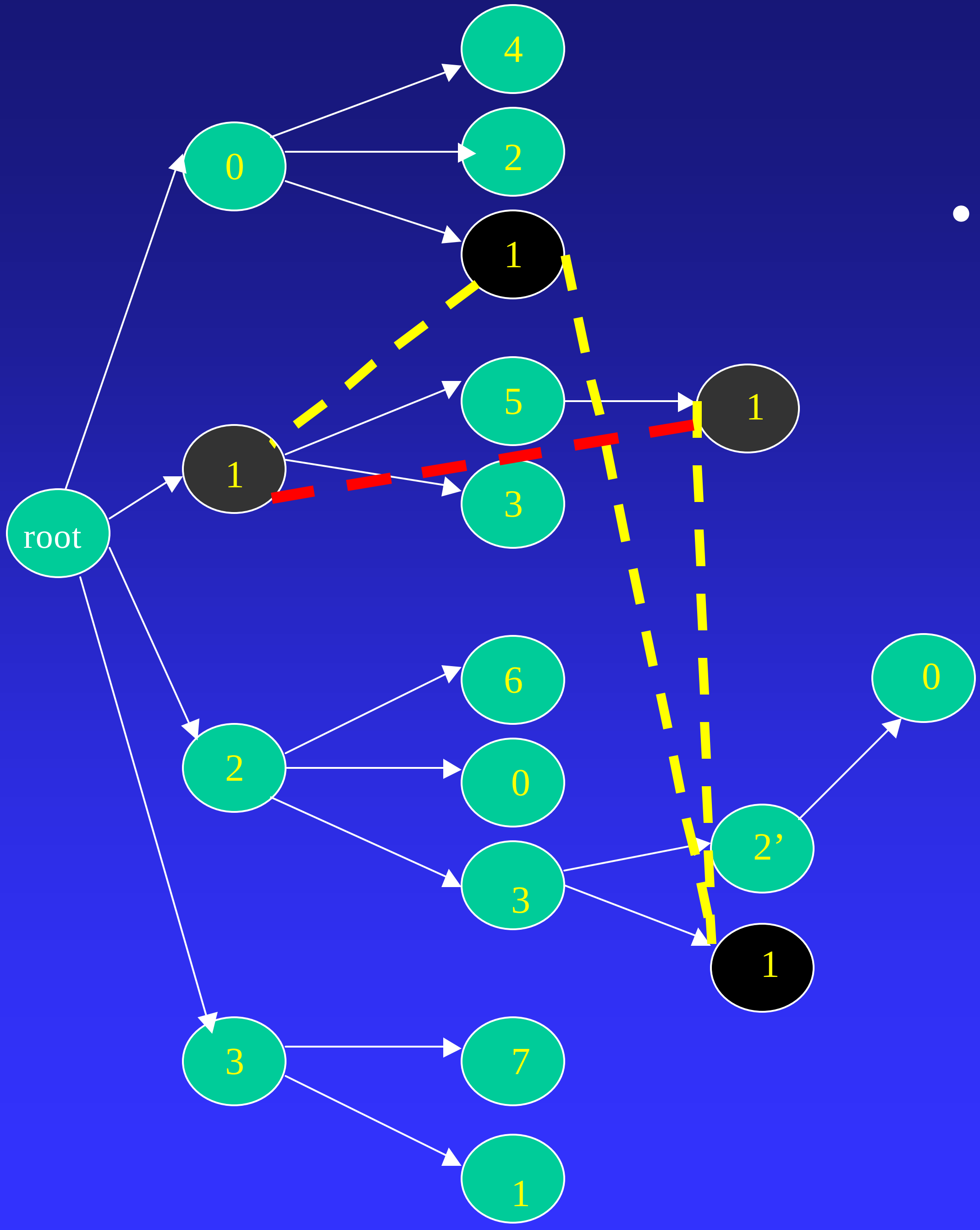
conflict - no feasible schedule contains both links

Cache Saving Links



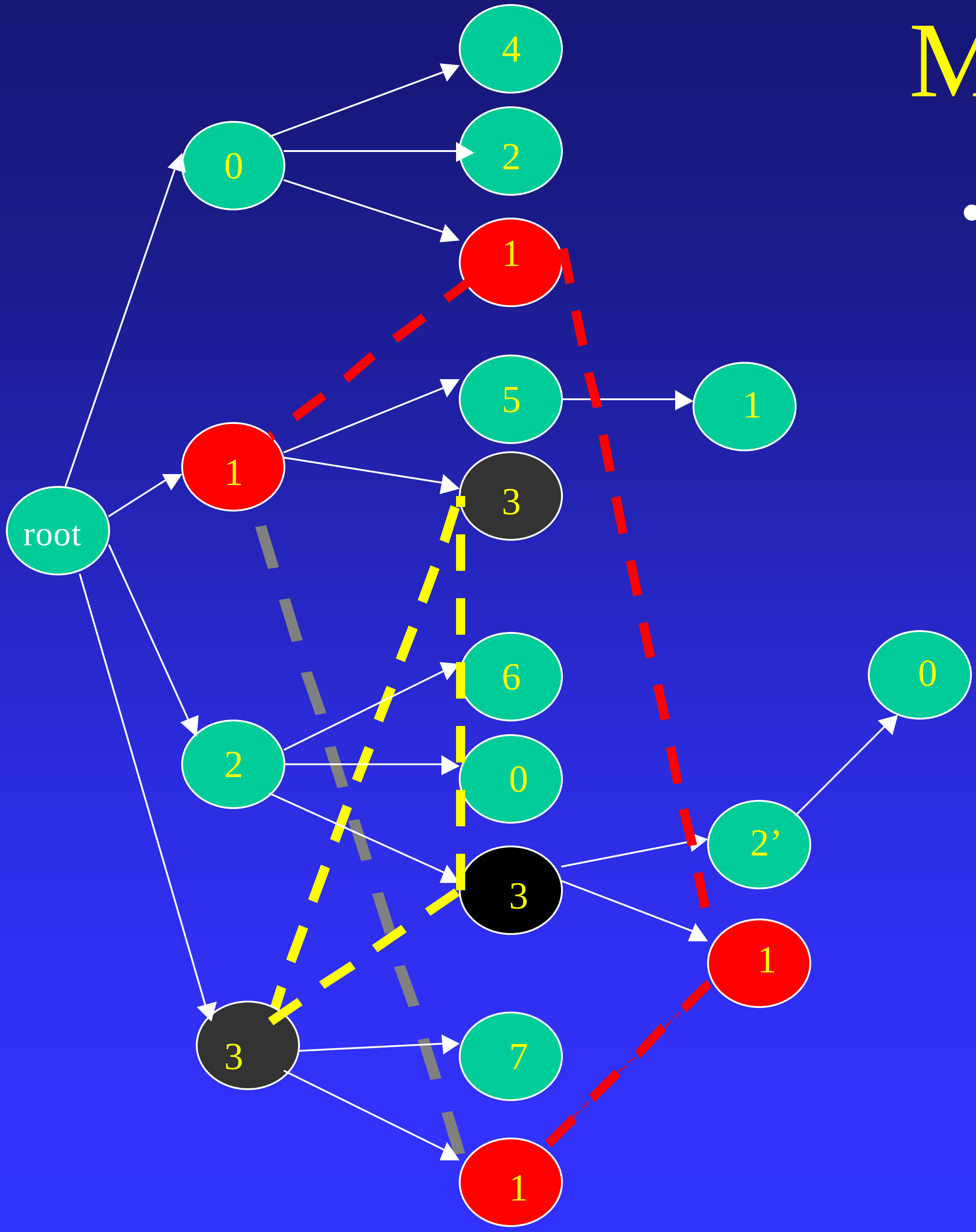
optimal schedule - maximal group of non-conflicting links

Chains



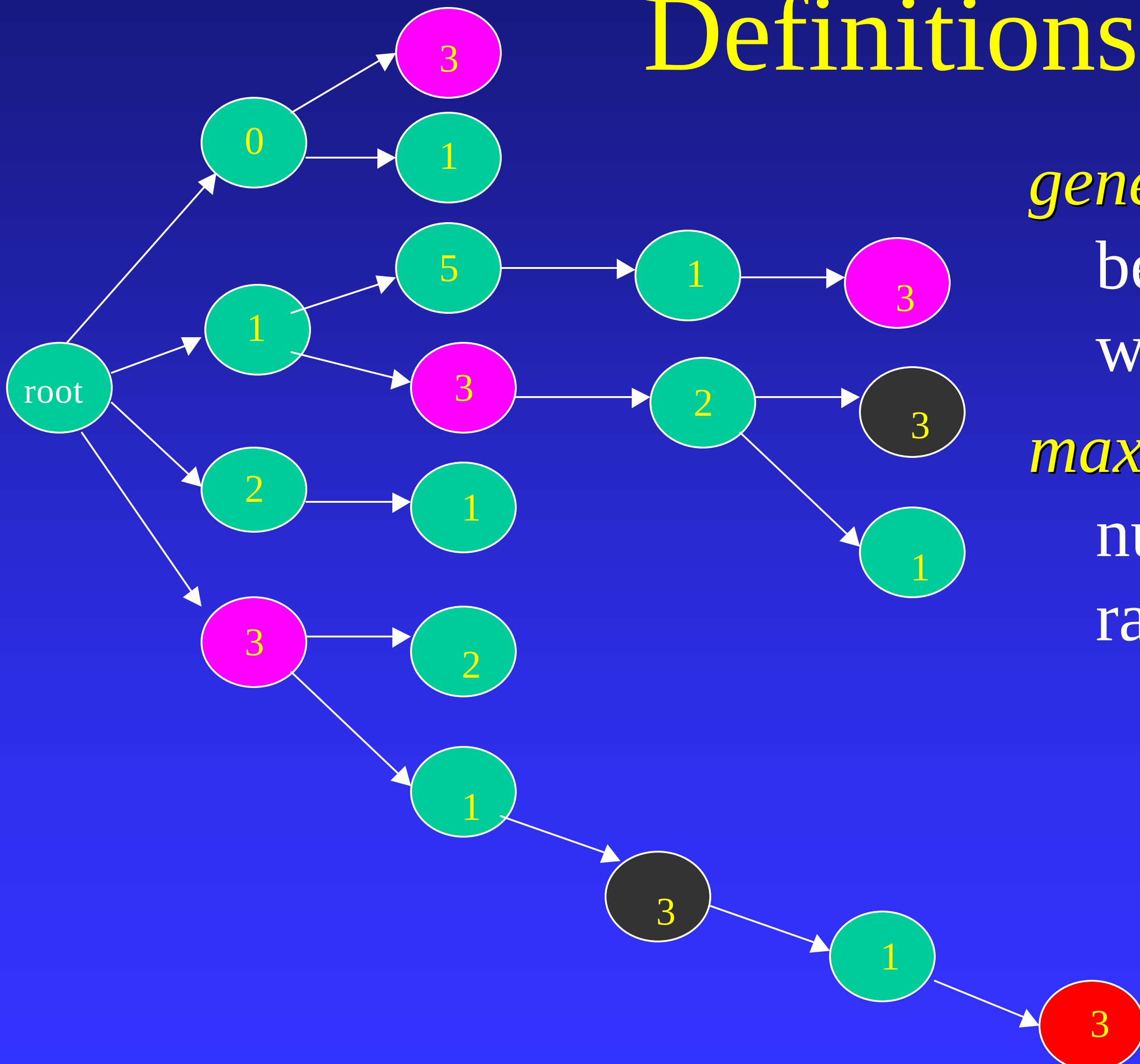
- Chain of non-conflicting links may produce a non-feasible schedule

Multiple Chains



- A combination of chains may also produce a non-feasible schedule

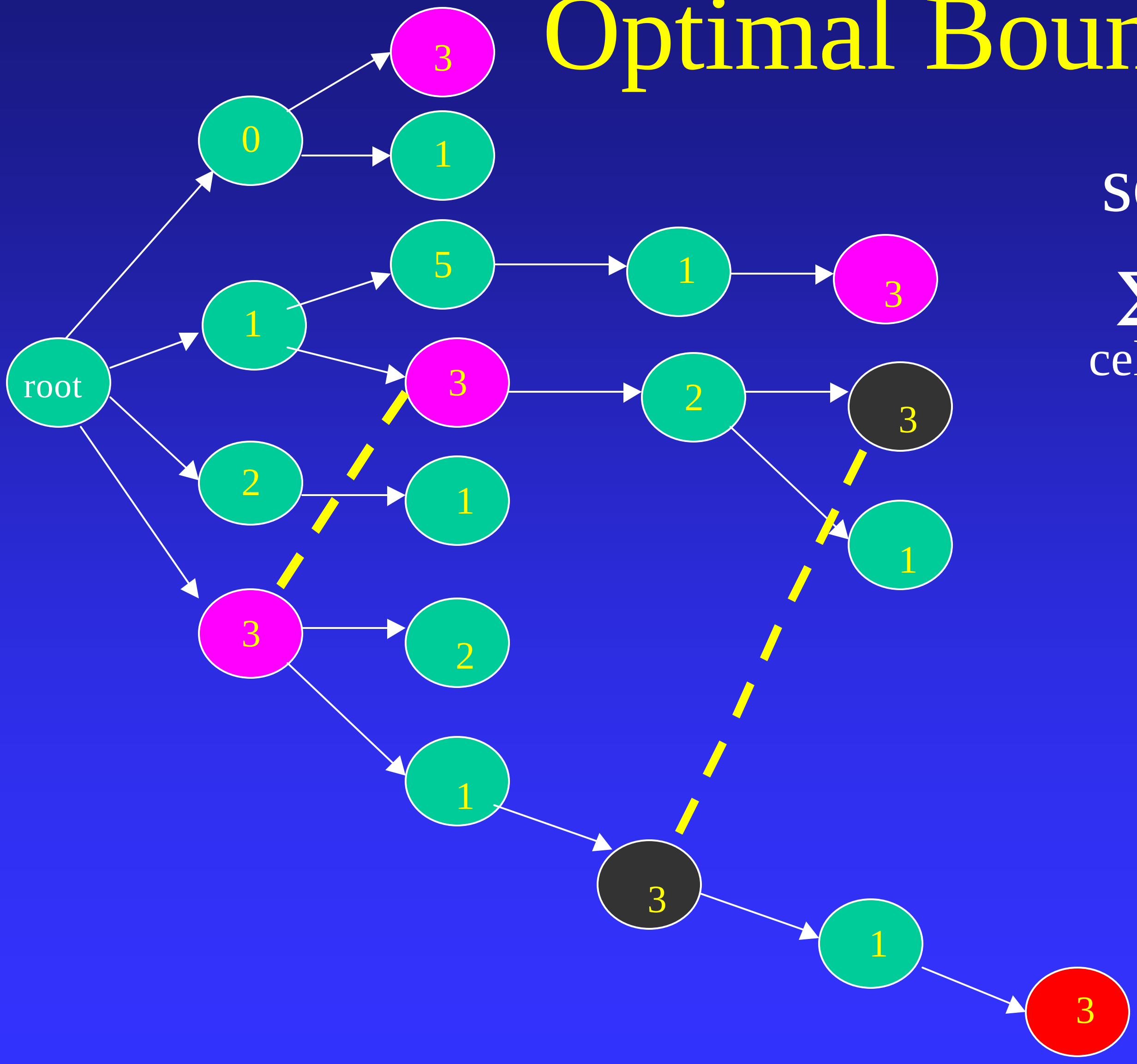
Definitions



generation(node) - nodes between root and *node* with same cell as *node*

maxGen(cell) - max number of times *any* ray enters *cell*

Optimal Bound



schedule size >=

$$\sum_{cells} \text{maxGen(cell)}$$

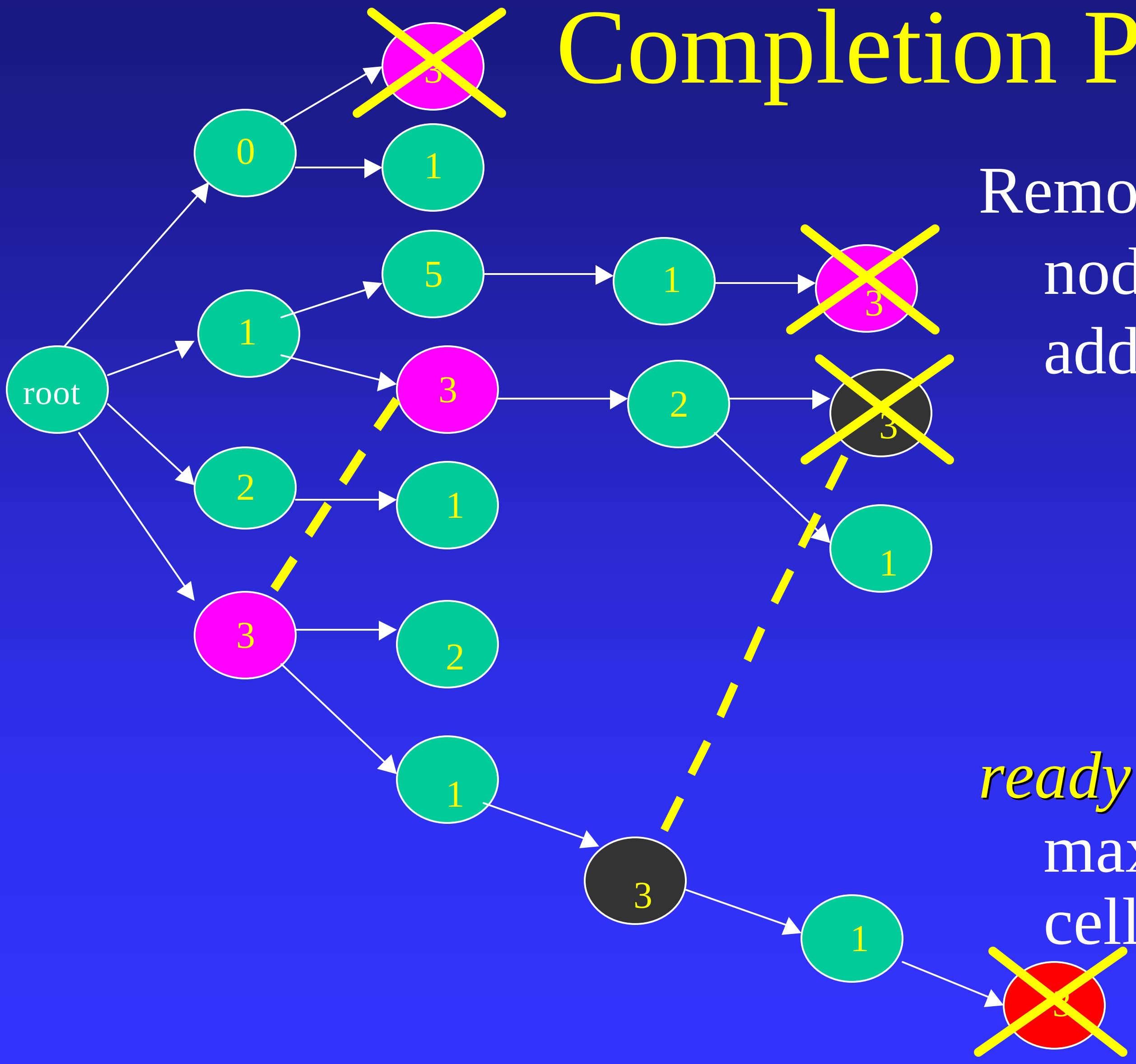
Outline

- Our System
- Cell Tree
- Dependency Graph Scheduling
- **Peel Algorithm**
- Results

Peel Algorithm

- Peel tree leaves to create reverse schedule
- Gather cache savings links

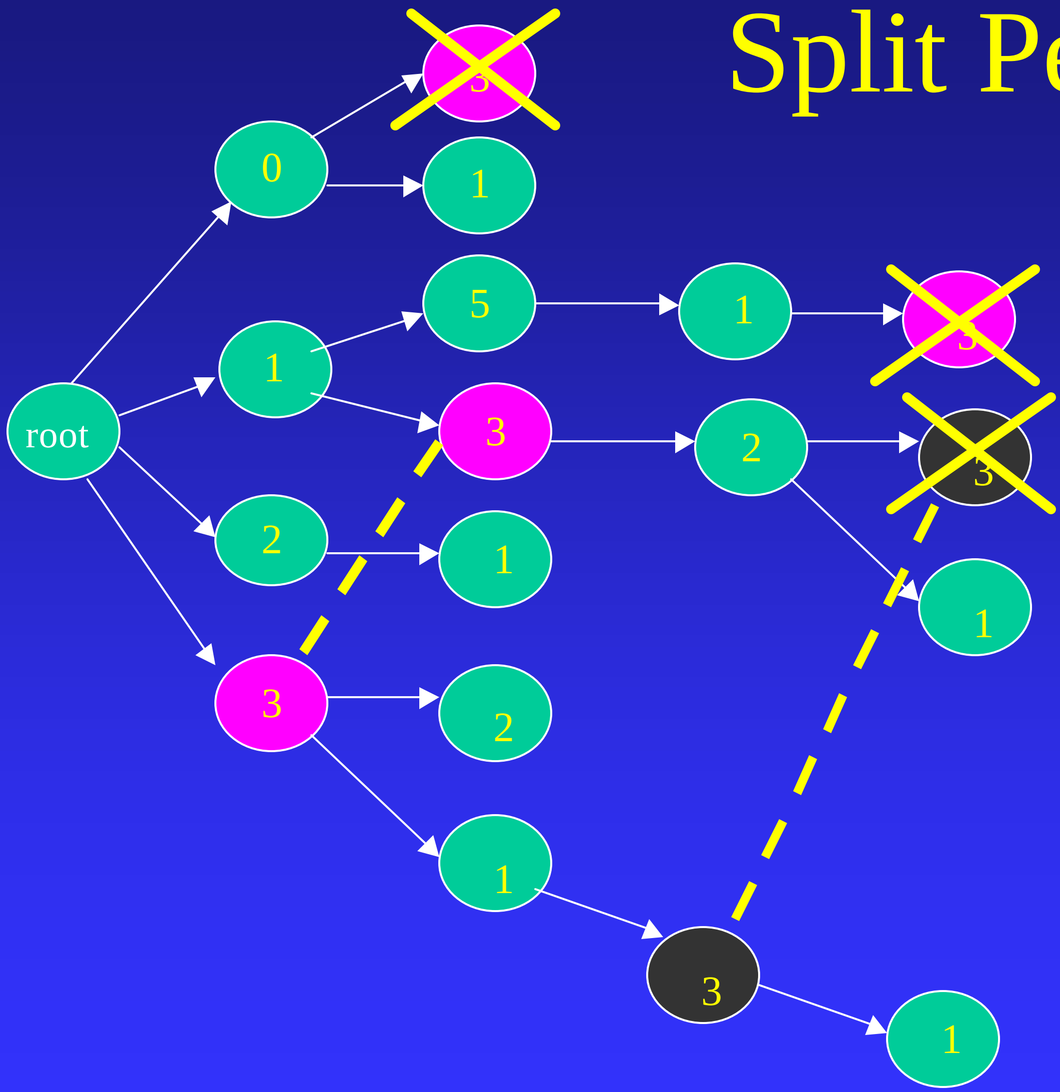
Completion Peel



Remove *ready cell* leaf nodes from tree and add it to schedule

ready cell - all the maxGen nodes of a cell are leaf nodes

Split Peel



Remove non-ready cell leaf nodes from tree and add it to schedule

Peel (***tree***)

While *tree* has any nodes…

Does *tree* have a ready cell *c*?

yes - add *c* to *schedule* and peel *c* leaf nodes

no - Find $cell_{max}$ with most leaf nodes

Peel $cell_{max}$ and add it to *schedule*

Return *schedule*

Peel (***tree***)

While *tree* has any nodes…

Does *tree* have a ready cell *c*?

yes - add *c* to *schedule* and peel *c* leaf nodes

no - Find $cell_{max}$ with most leaf nodes

Peel $cell_{max}$ and add it to *schedule*

Return *schedule*

Peel (***tree***)

While *tree* has any nodes…

Does *tree* have a ready cell *c*?

yes - add *c* to *schedule* and peel *c* leaf nodes

no - Find $cell_{max}$ with most leaf nodes

Peel $cell_{max}$ and add it to *schedule*

Return *schedule*

Peel (*tree*)

While *tree* has any nodes…

Does *tree* have a ready cell *c*?

yes - add *c* to *schedule* and peel *c* leaf nodes

no - Find $cell_{max}$ with most leaf nodes

Peel $cell_{max}$ and add it to *schedule*

Return *schedule*

Peel (***tree***)

While *tree* has any nodes…

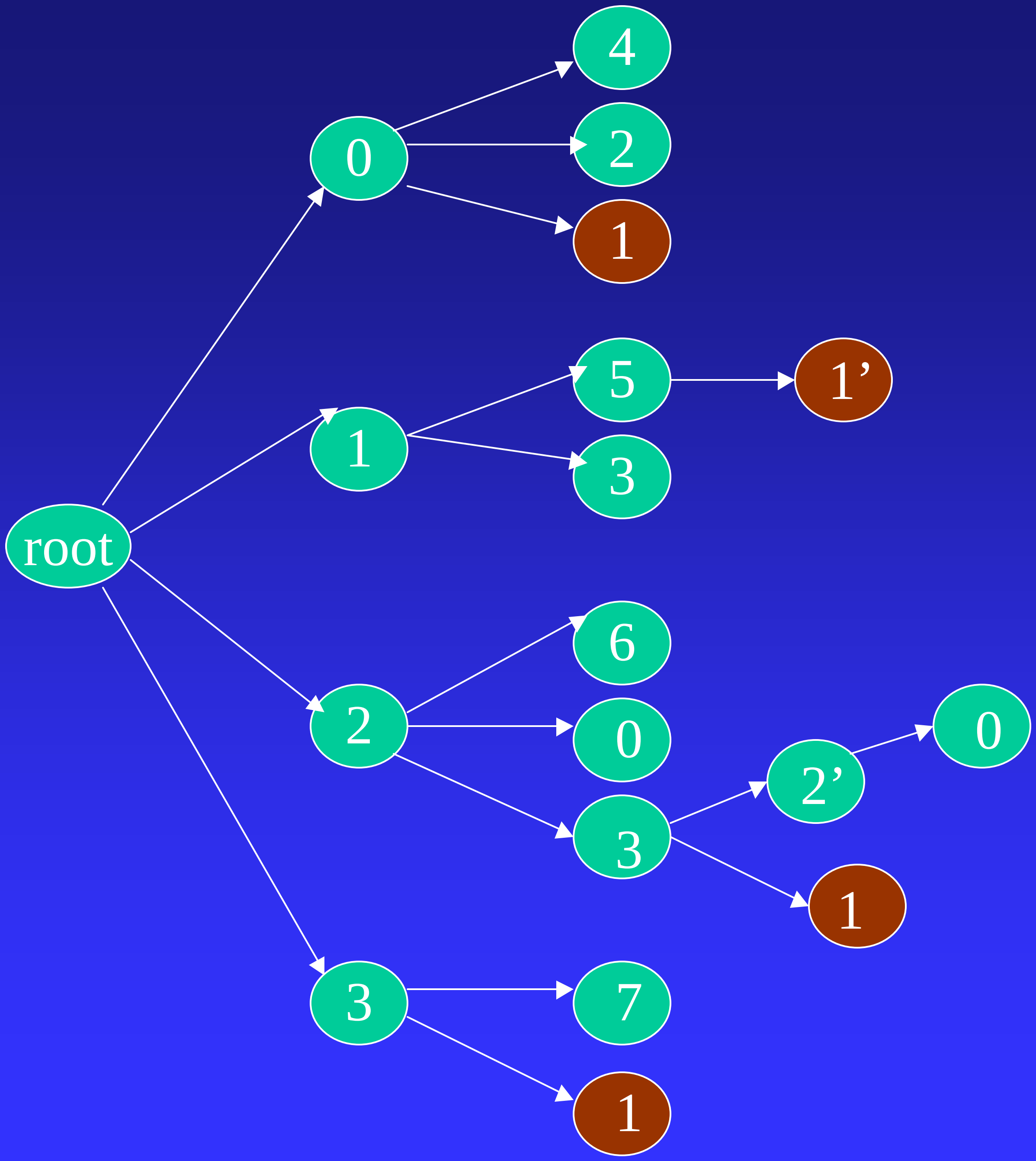
Does *tree* have a ready cell *c*?

yes - add *c* to *schedule* and peel *c* leaf nodes

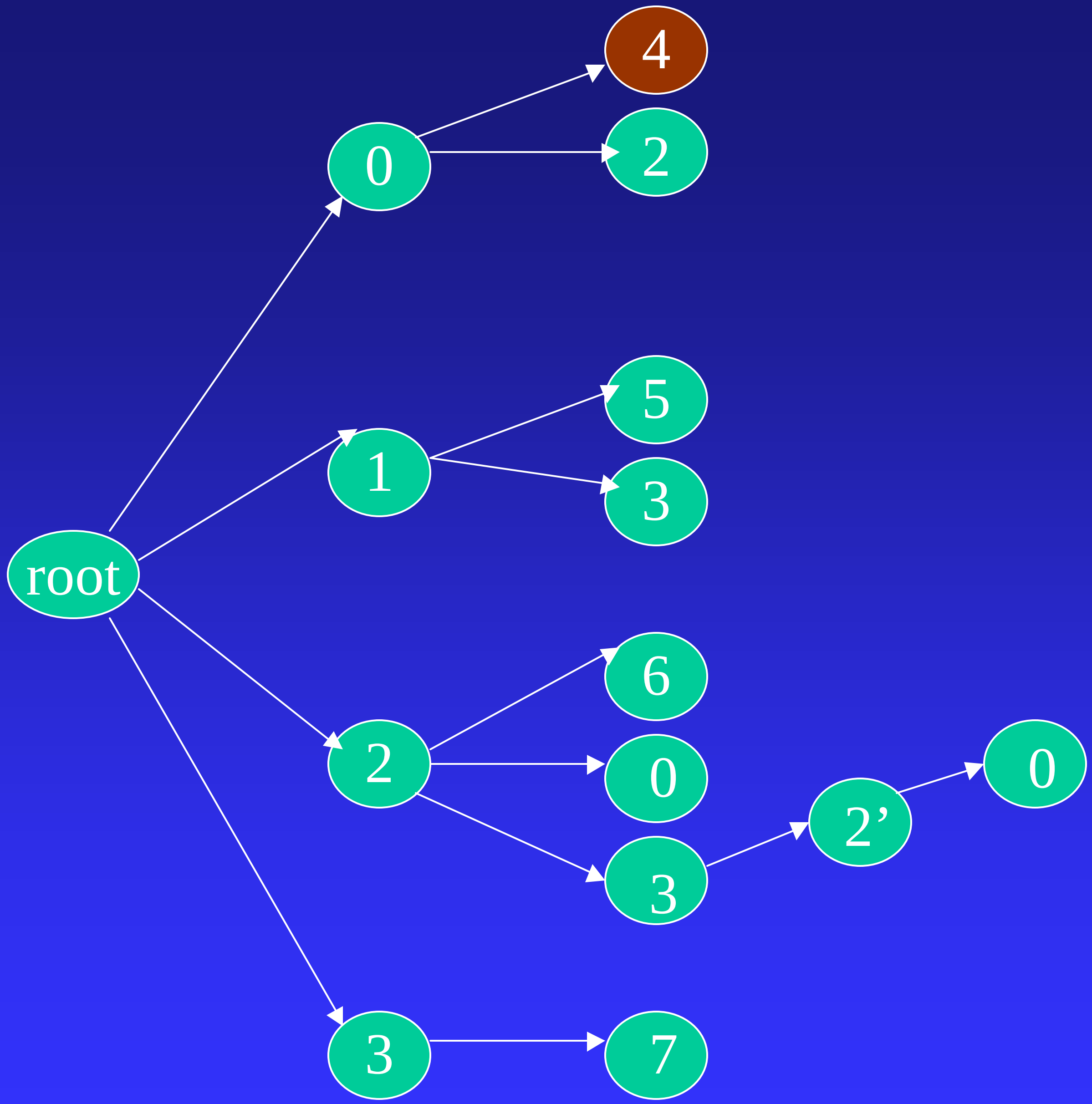
no - Find $cell_{max}$ with most leaf nodes

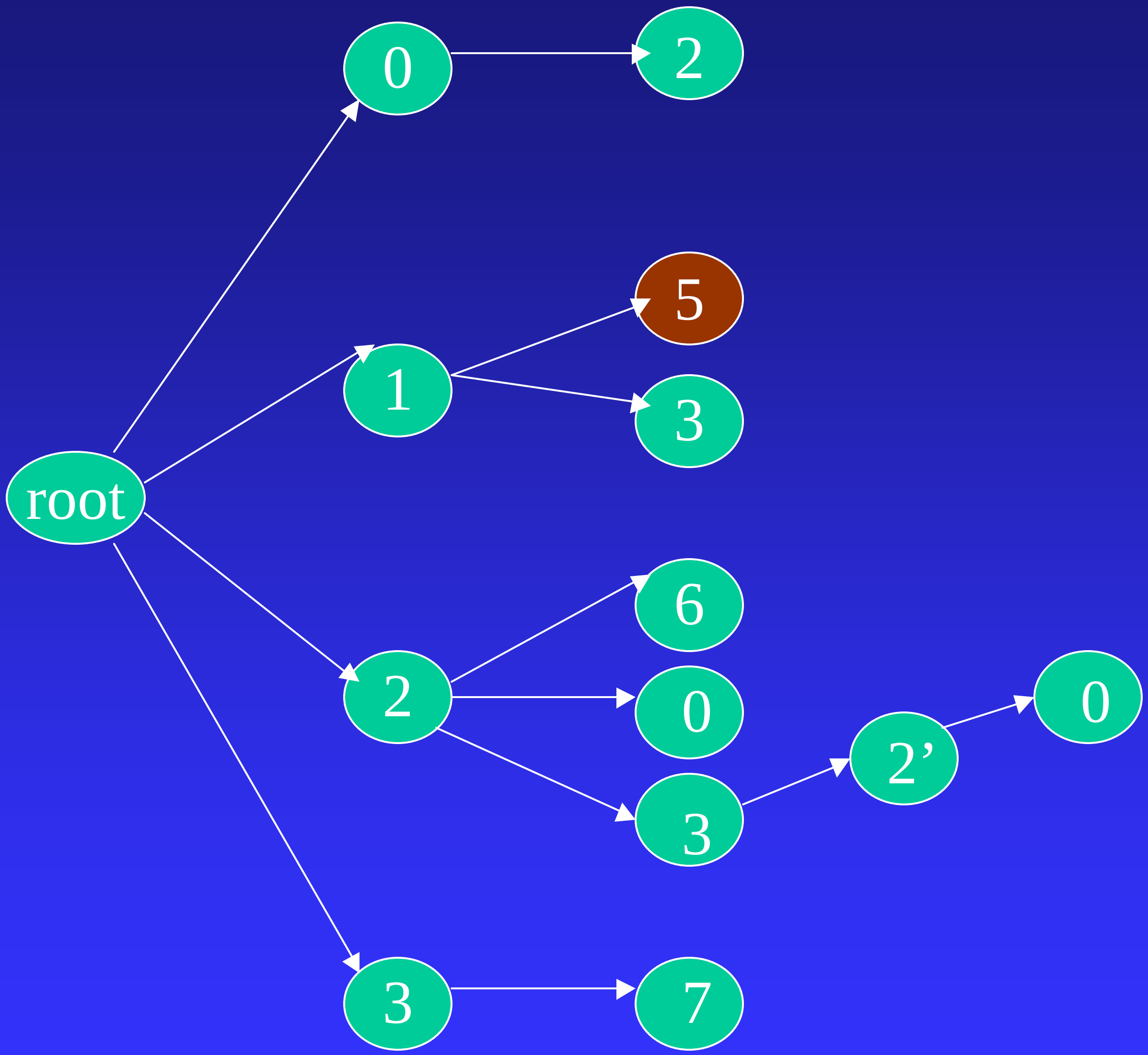
Peel $cell_{max}$ and add it to *schedule*

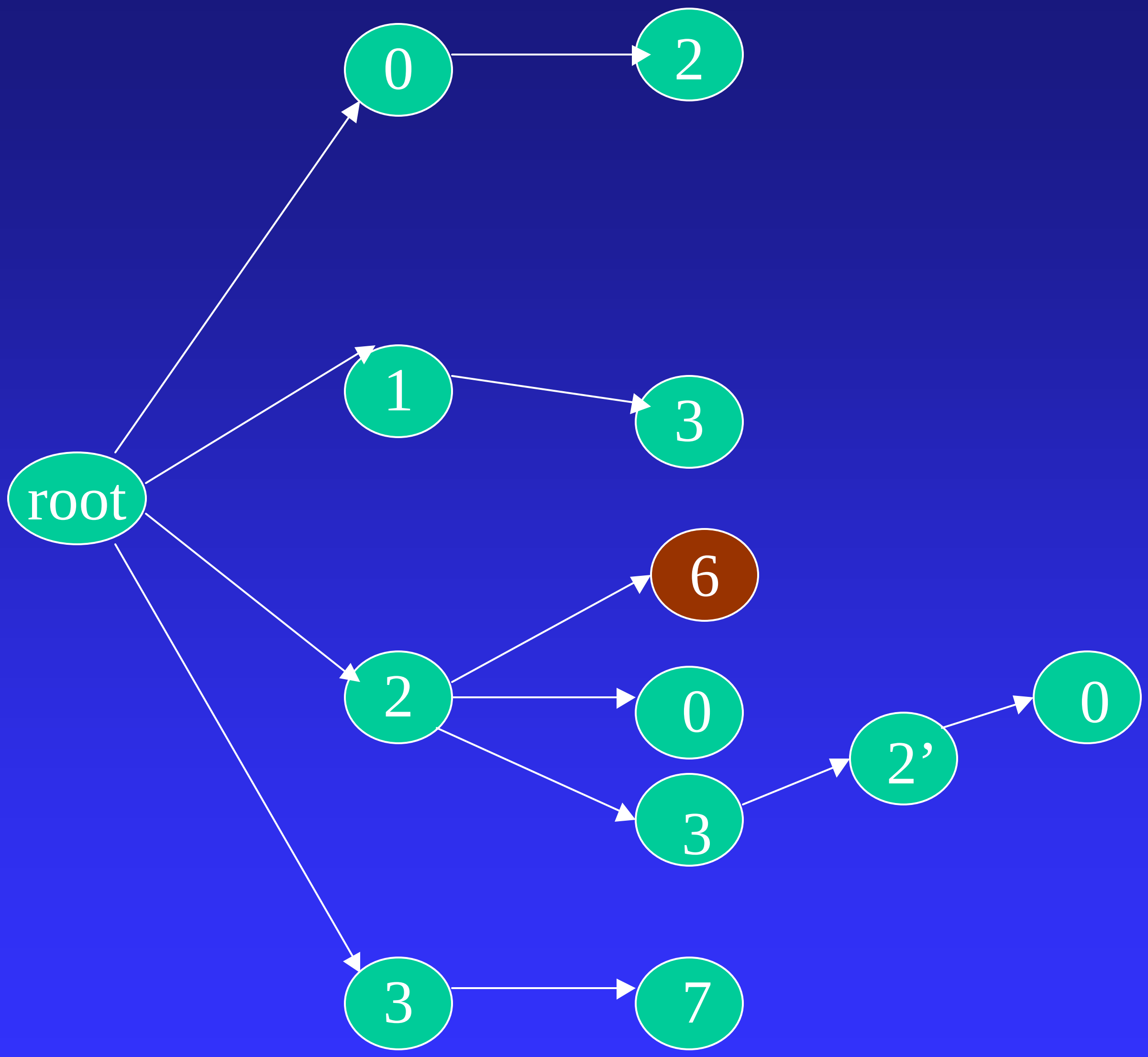
Return *schedule*

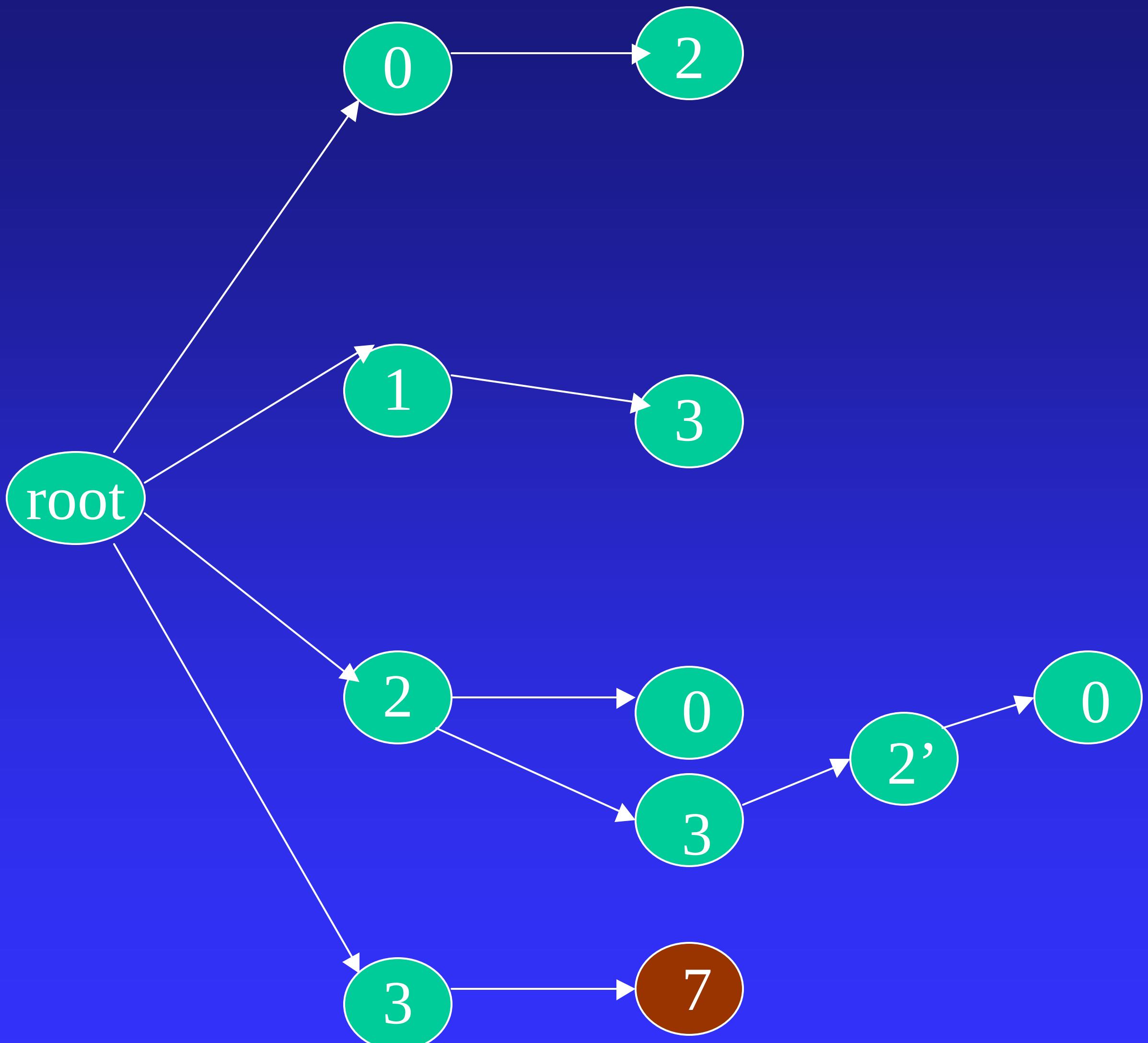


root
0
1
2
3
4
2
1
5
3
1’
6
0
3
2’
0
1
7
1

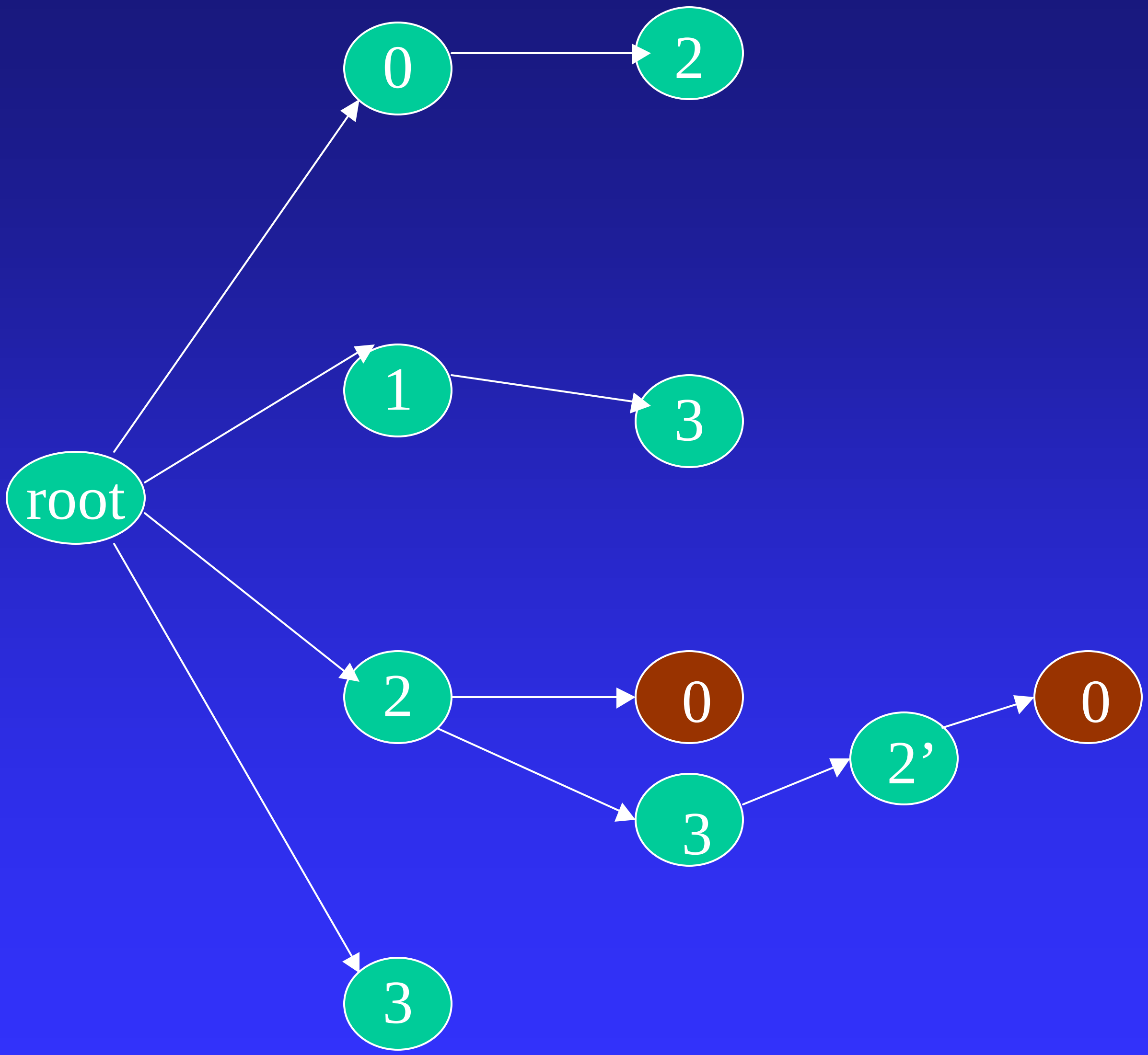

4
0
2
5
1
3
root
6
2
0
0
2’
3
3
7

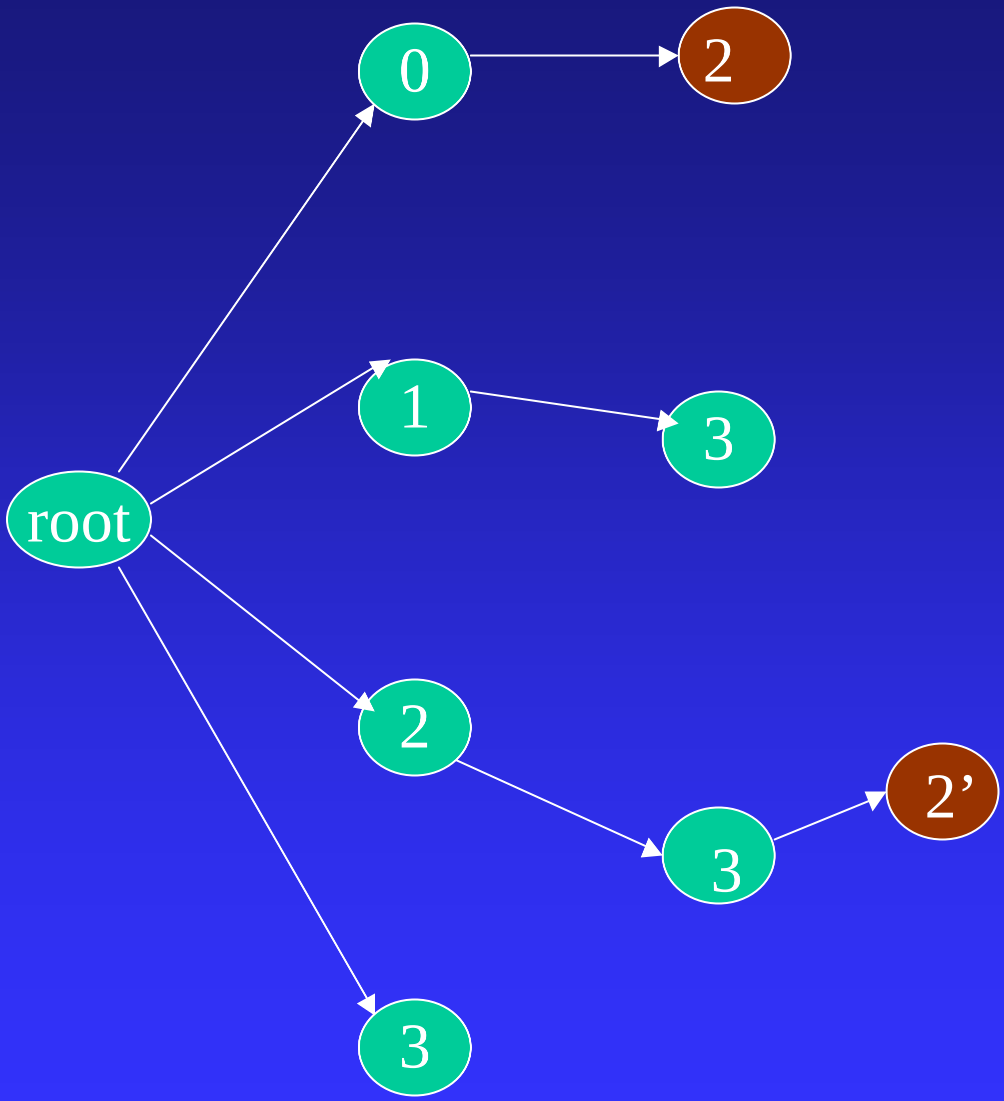

0
2
5
1
3
root
6
2
0
0
2'
3
3
7


root
0
2
1
3
6
2
0
3
2’
0
3
7

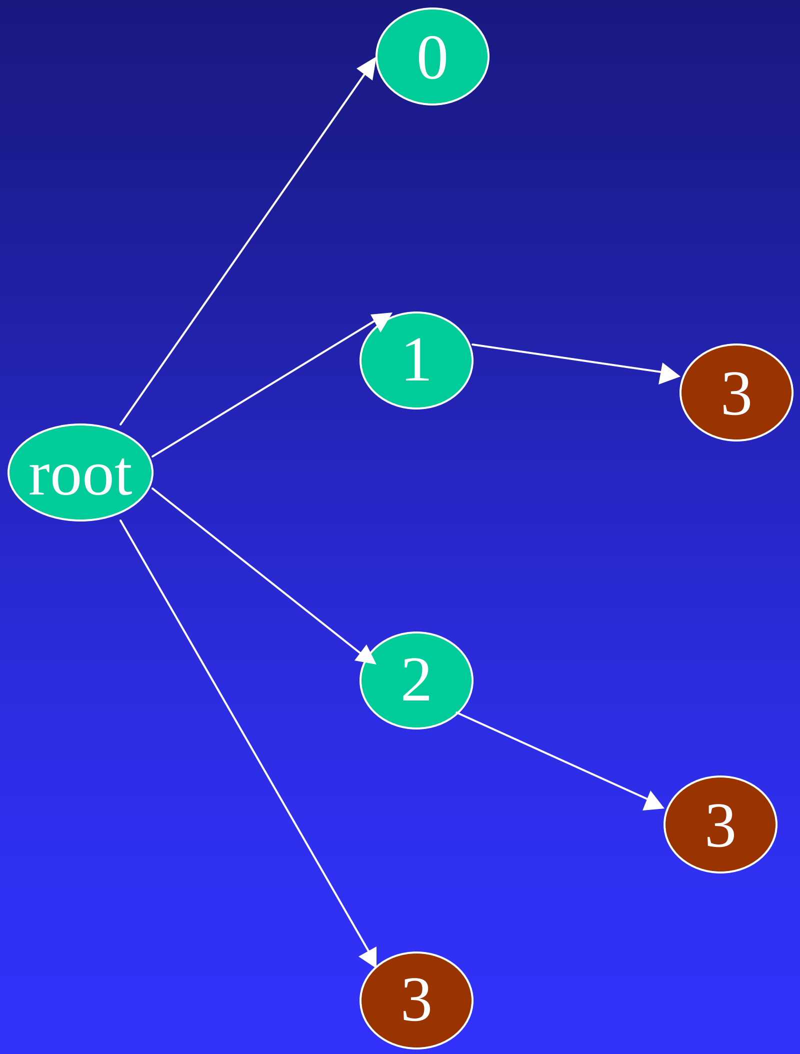


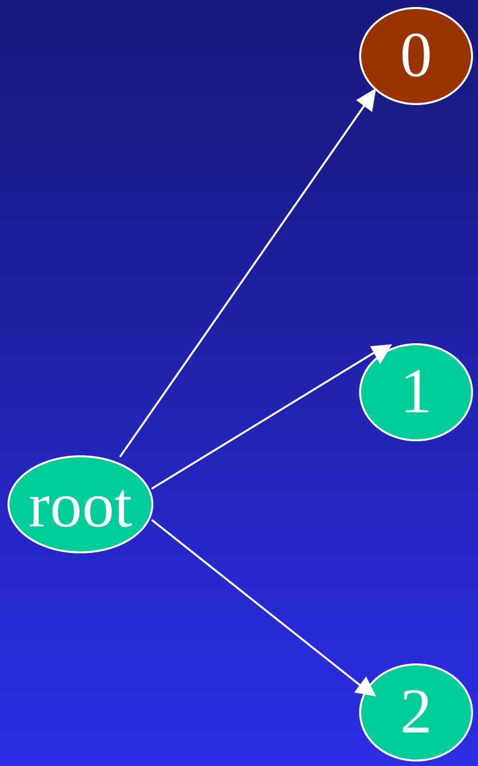
0
2
1
3
root
2
0
2'
0
3
3
7

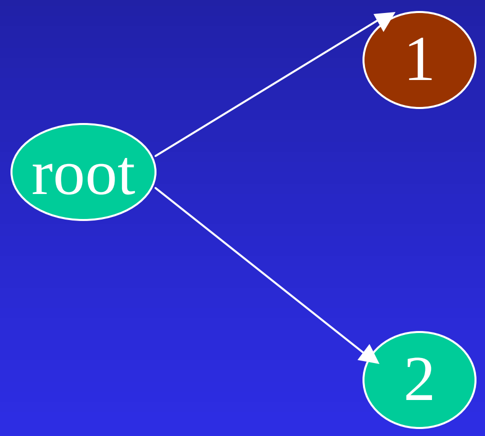

root
0
1
2
3
2
3
0
3
2’
0

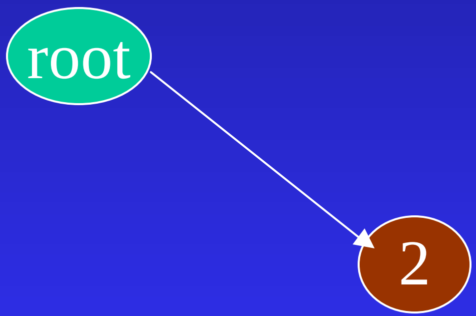


root
0
2
1
3
2
3
2'
3


0
1
3
root
2
3
3


0
1
root
2


root
1
2


root
2

Algorithm Performance

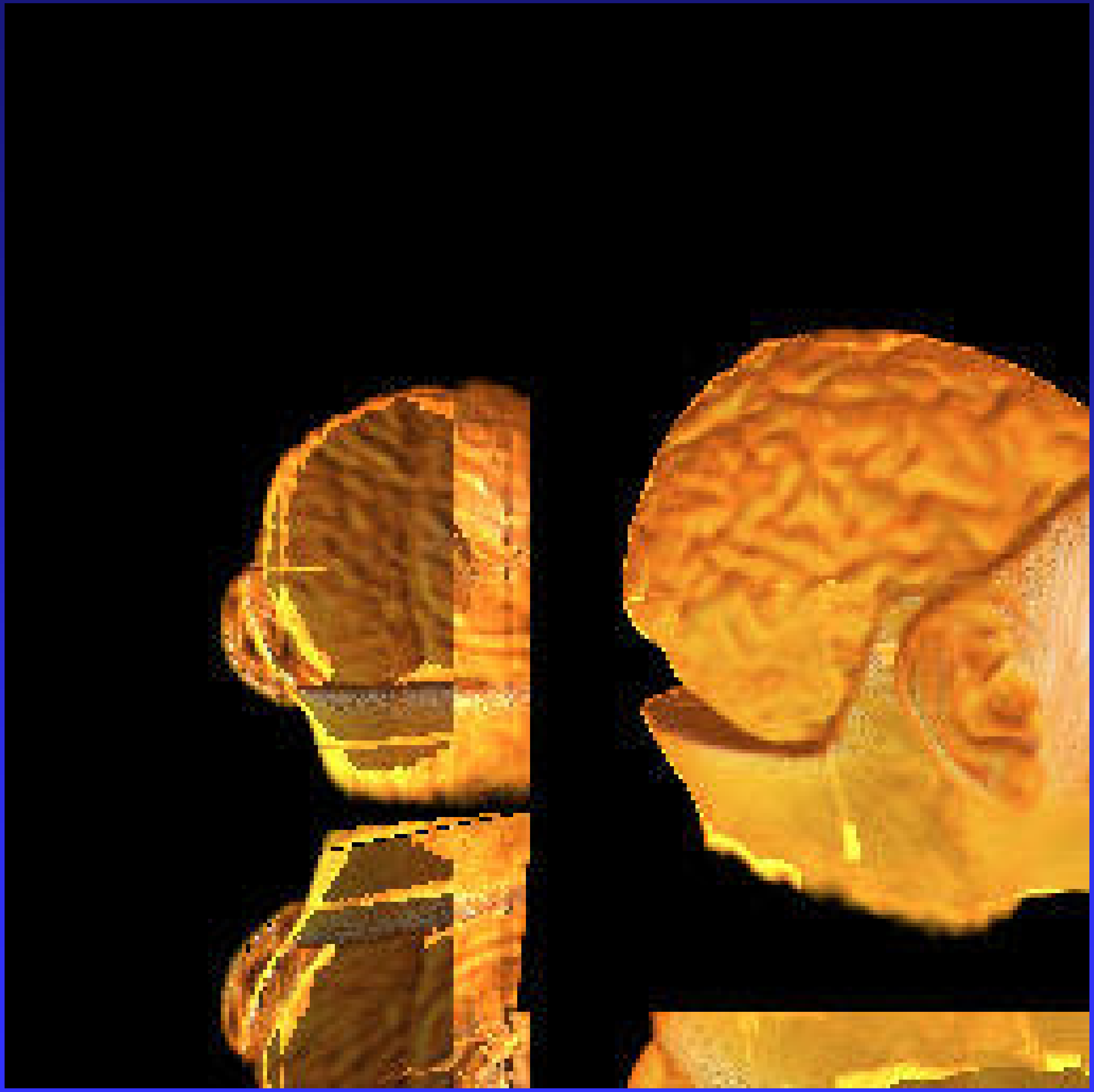
- Guaranteed feasible
- Not guaranteed optimal
- Worst time O(n)
- Improvement over Max Work
- Hardware implementation reasonable

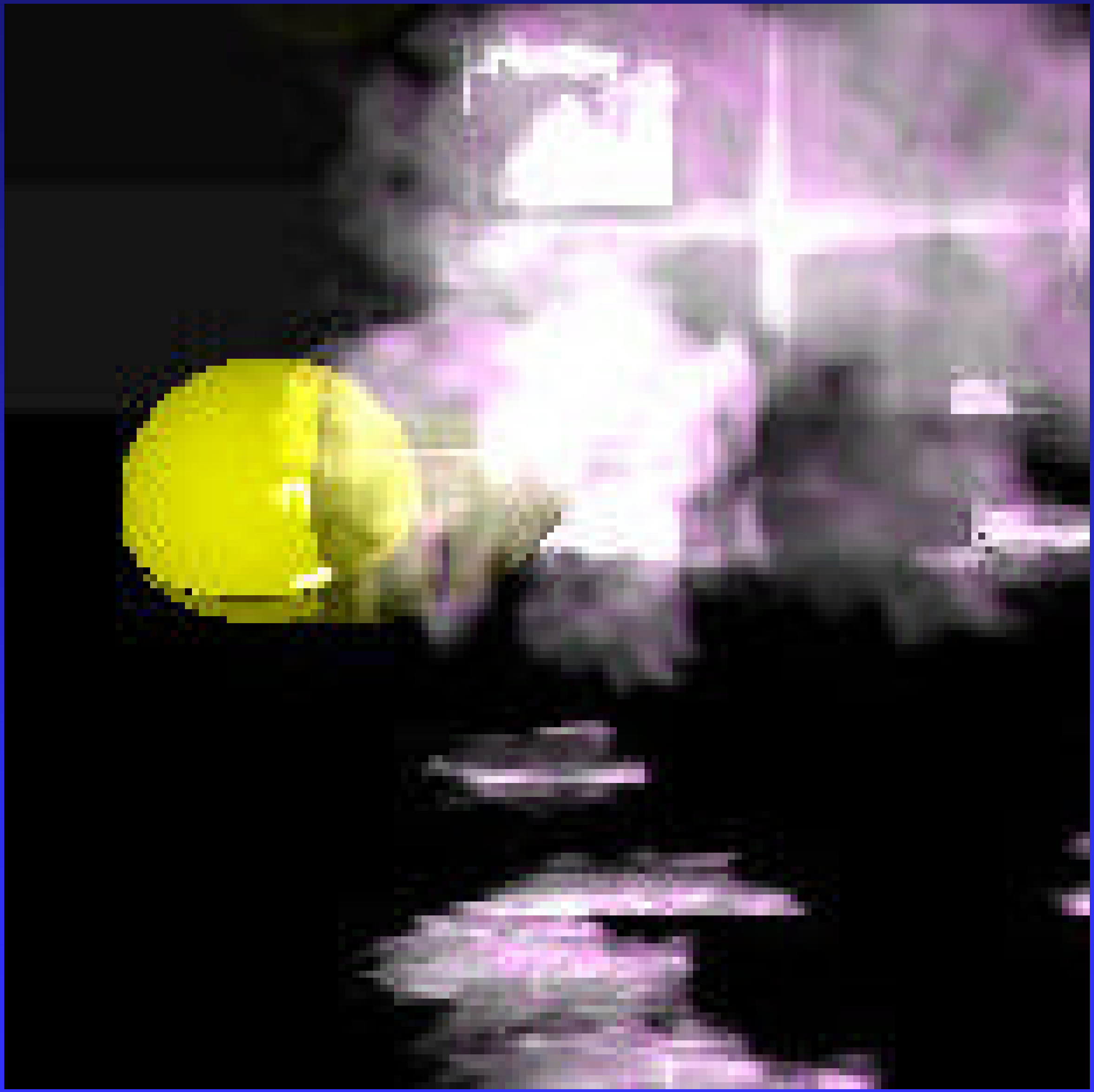
Outline

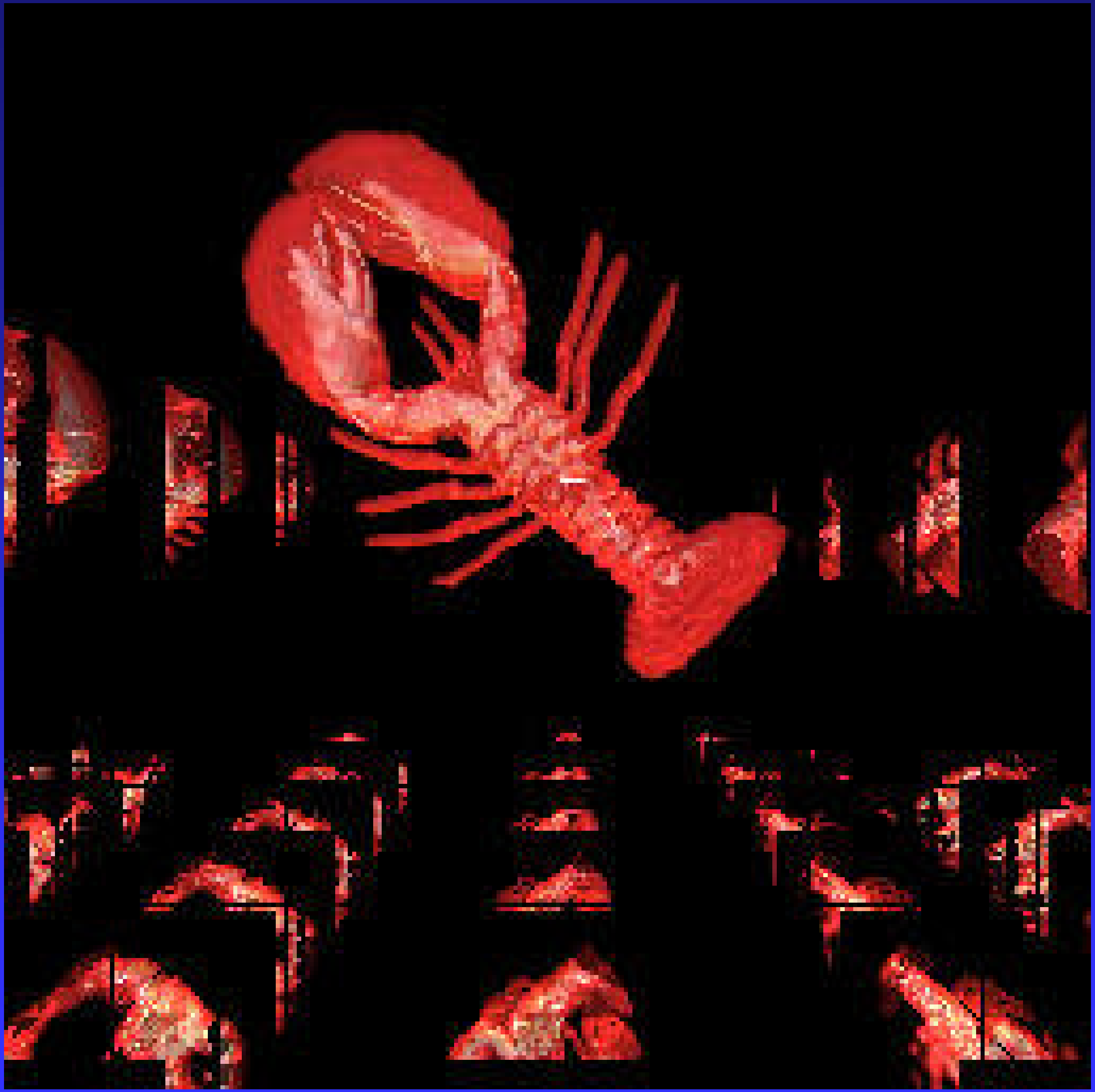
- Our System
- Cell Tree
- Dependency Graph Scheduling
- Peel Algorithm
- **Results**

Tests

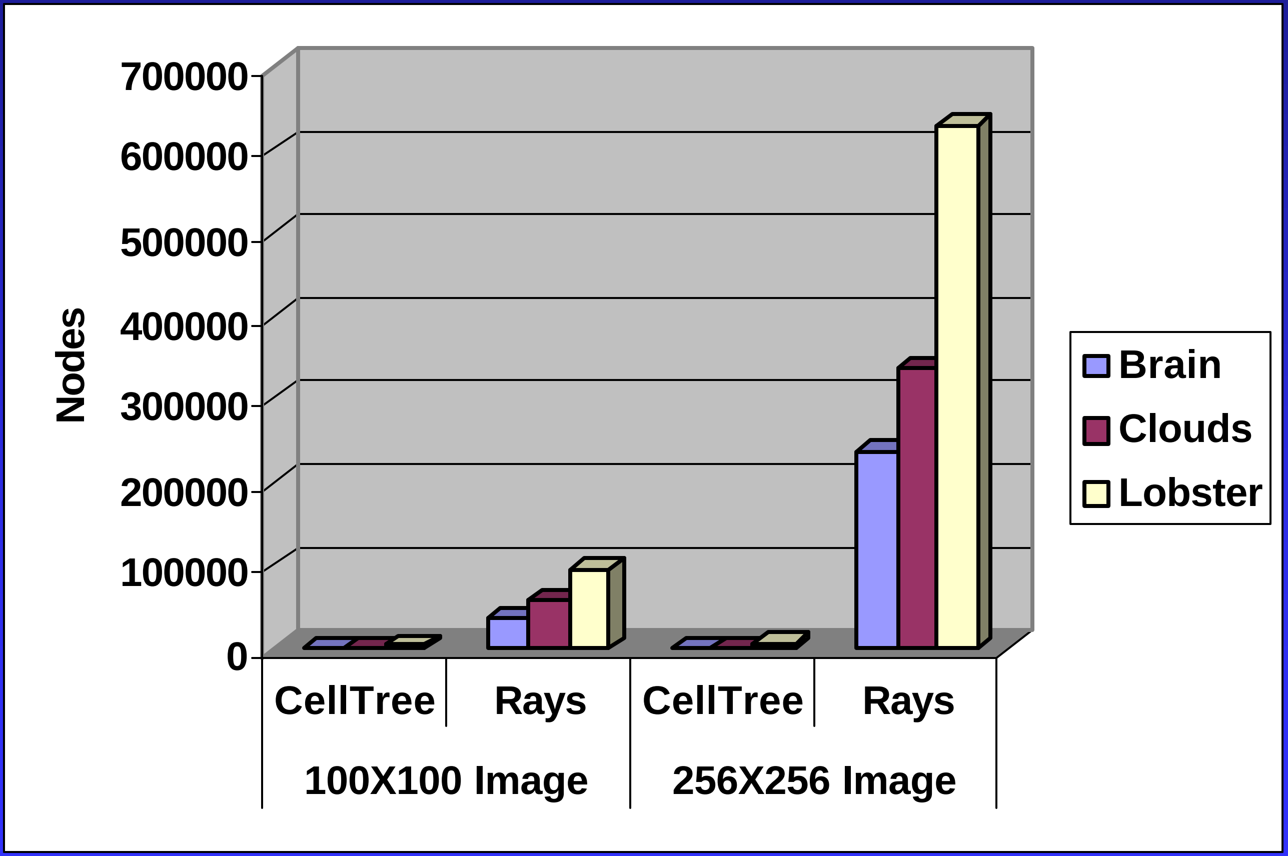
- C++ simulation
- SGI 02/RISC 10000 128MB
- Volumes split into 8 cells and 27 cells
- Image resolution 256^2



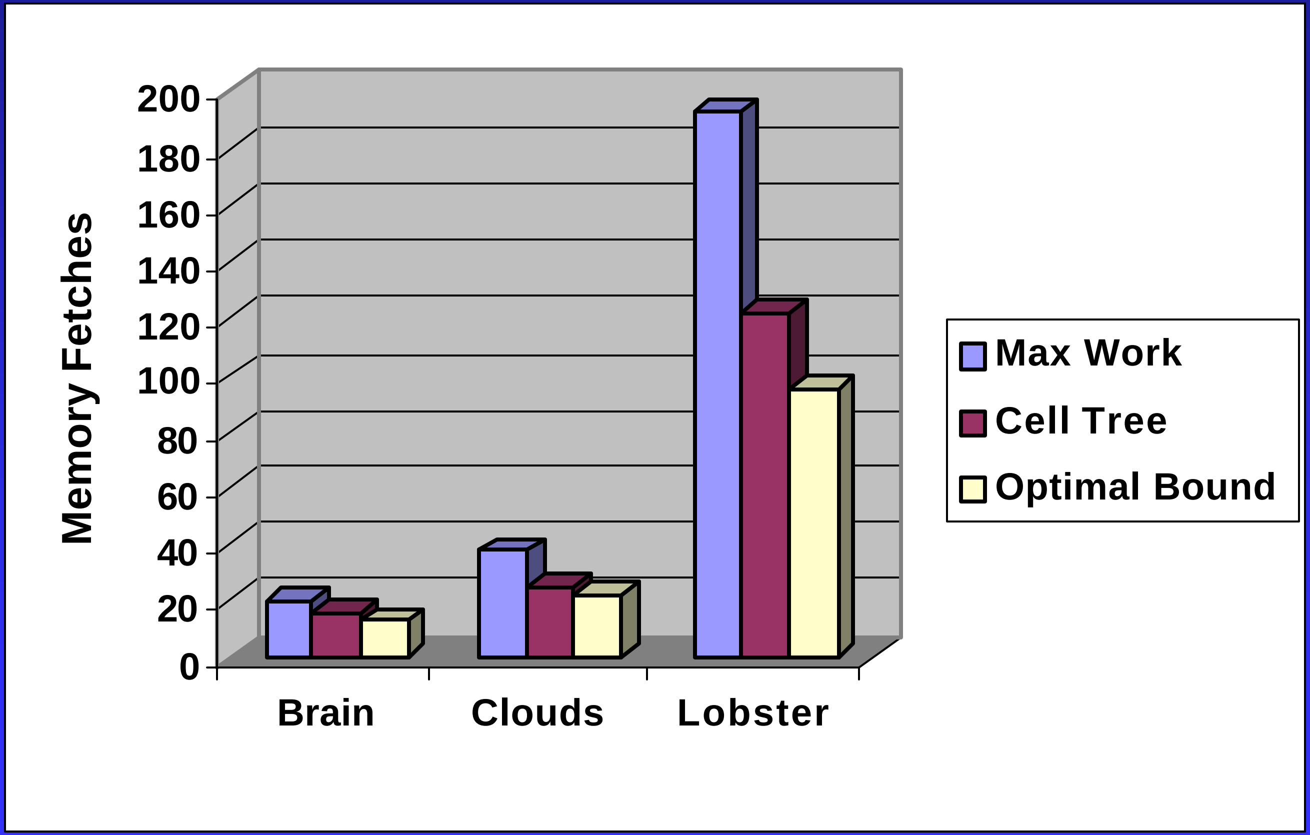




Cell Tree Sizes



30% Fewer Fetches



Conclusion

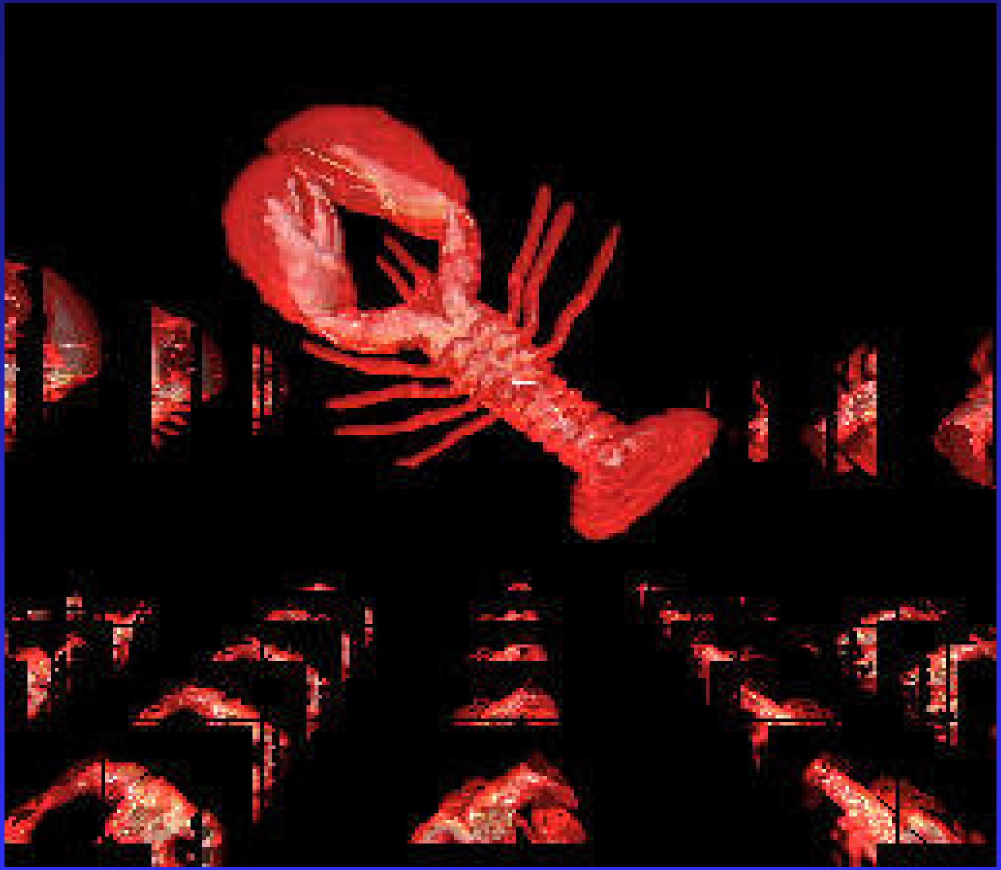
- Cell Tree captures all ray-cell dependencies
- Dependency graph based algorithm significantly improves cache performance

Future Work

- Dynamic load balance
- Dynamic volume subdivision
- Multi-level memory hierarchy
- Limited depth recursion

Acknowledgments

- ONR Grant N00140110034
- NYSTAR Grant COD0057
- CES Computer Solutions Inc.
- Kevin Kreeger, Frank Dachille, Michael Bender, Nan Zhang



Thank you!