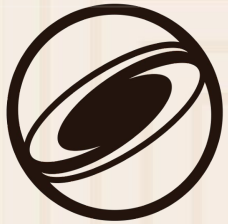


SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

THE PREMIER CONFERENCE & EXHIBITION ON COMPUTER GRAPHICS & INTERACTIVE TECHNIQUES

Multi-Threading in OpenVDB

Dan Bailey
Staff Software Engineer
Industrial Light & Magic

Talk Overview

Sequential Access
- Tree Hierarchy
- Tree Iterators
- Depth-First vs Breadth-First
- Tree Visitor Methods

Multi-Threaded Case Study
- DIsney Cloud Value Clamp
- Copy, Clamp, Prune
- Mask Topology
- Scatter Merge
- Dynamic Build

Parallel Constructs
- Thread Safety
- DynamicNodeManager

Build Performance
- Slow Compile Times
- Explicit Template Instantiation

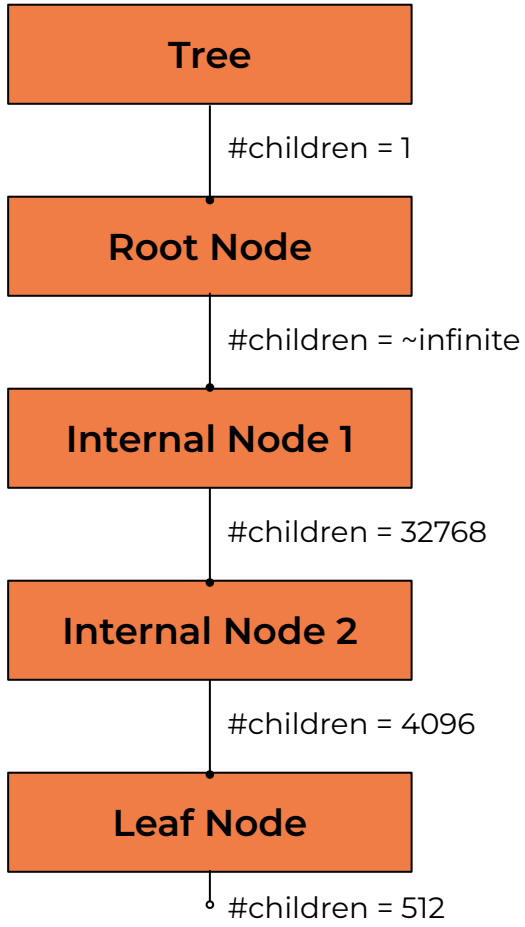

THE PREMIER CONFERENCE & EXHIBITION ON COMPUTER GRAPHICS & INTERACTIVE TECHNIQUES

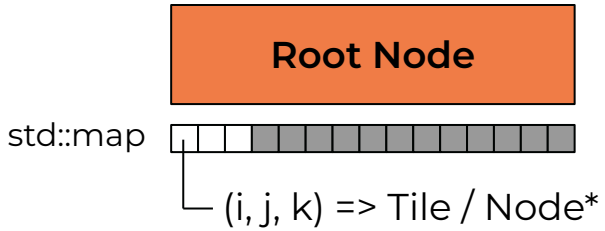
SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

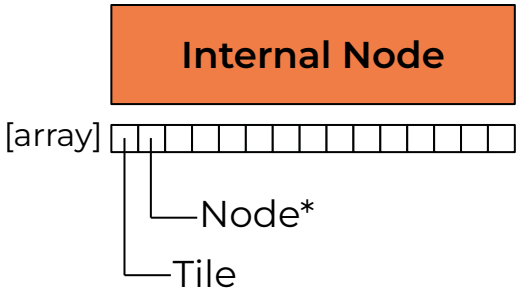
Sequential Access

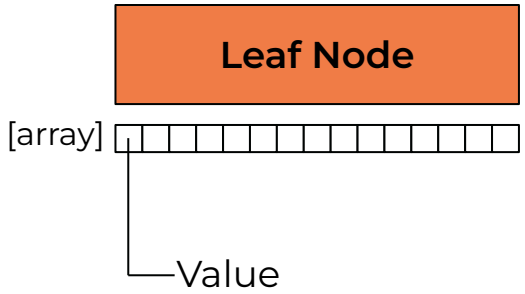


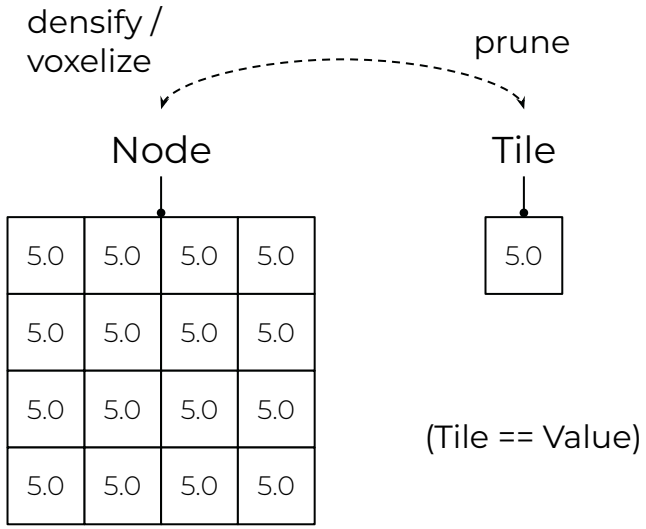

Tree Hierarchy



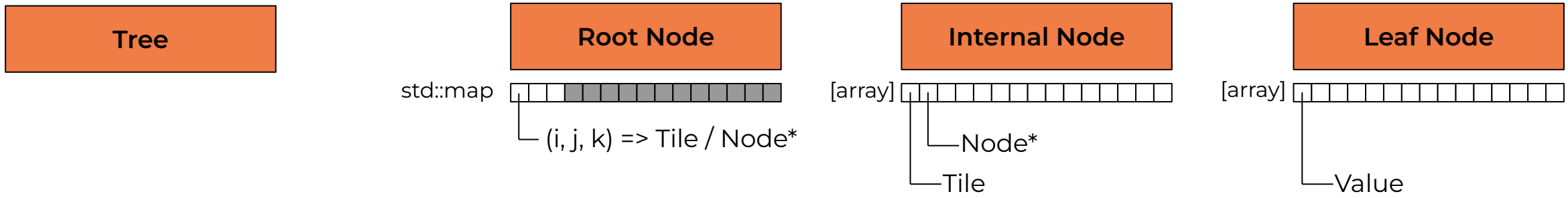








Tree Iterators



- beginRootChildren()
- beginRootTiles()
- beginRootDense()

- **beginNode()**
- **beginLeaf()**
- **beginValueOn()**
- **beginValueOff()**
- **beginValueAll()**

Hierarchical Iterators

- beginChildOn()
- beginChildOff()
- beginChildAll()
- beginValueOn()
- beginValueOff()
- beginValueAll()

All Elements

All Tiles

- beginChildOn()
- beginChildOff()
- beginChildAll()
- beginValueOn()
- beginValueOff()
- beginValueAll()

All Inactive Tiles + Children

All Elements

All Tiles

- *beginChildOn()*
- *beginChildOff()*
- *beginChildAll()*
- beginValueOn()
- beginValueOff()
- beginValueAll()

Tree Iterators

Option 1: Manual Iteration

```
for (auto iter1 = tree.cbeginRootChildren(); iter1; ++iter1) {
  for (auto iter2 = iter1->cbeginChildOn(); iter2; ++iter2) {
    for (auto iter3 = iter2->cbeginChildOn(); iter3; ++iter3) {
      for (auto iter4 = iter3->cbeginValueOn(); iter4; ++iter4) {
        sum += iter4.getValue();
} } } }
```

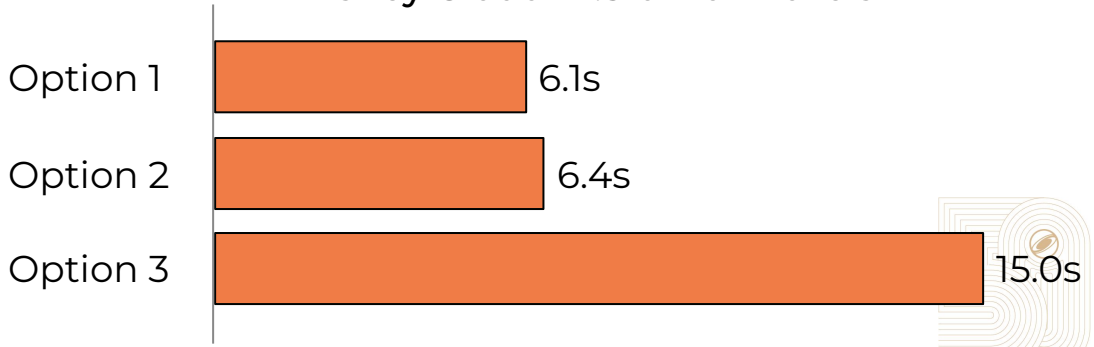
Option 2: Leaf Iteration

```
for (auto leaf = tree.cbeginLeaf(); leaf; ++leaf) {
  for (auto iter = leaf->cbeginValueOn(); iter; ++iter) {
    sum += iter.getValue();
} }
```

Option 3: Value Iteration

```
for (auto iter = tree.cbeginValueOn(); iter; ++iter) {
  sum += iter.getValue();
}
```

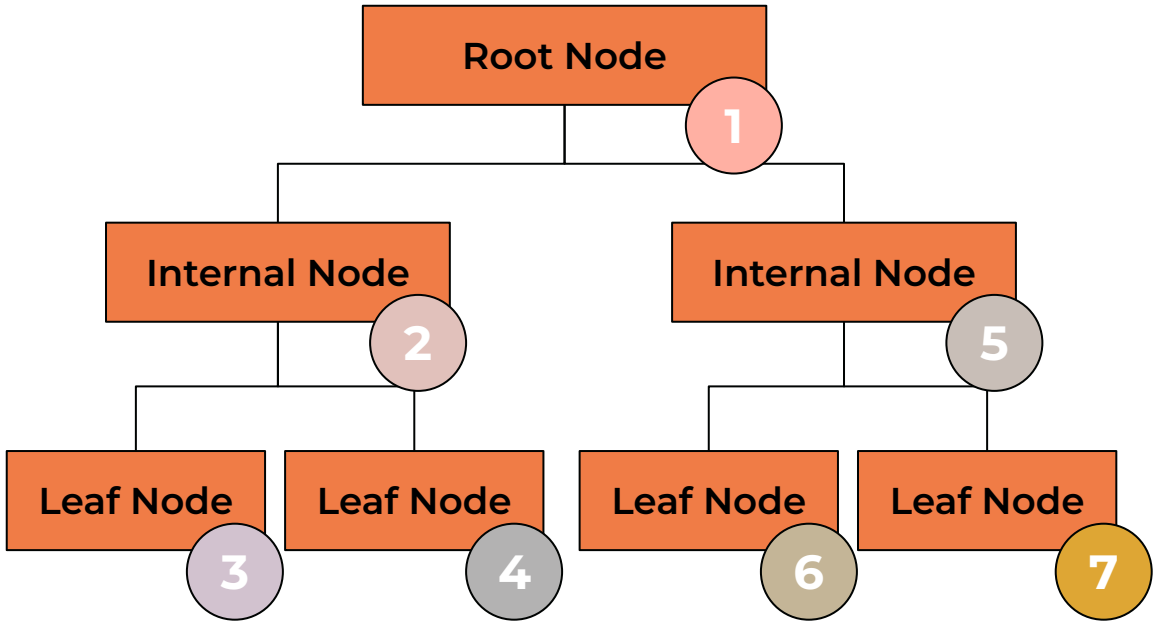
Disney Cloud - 1.5 billion voxels *



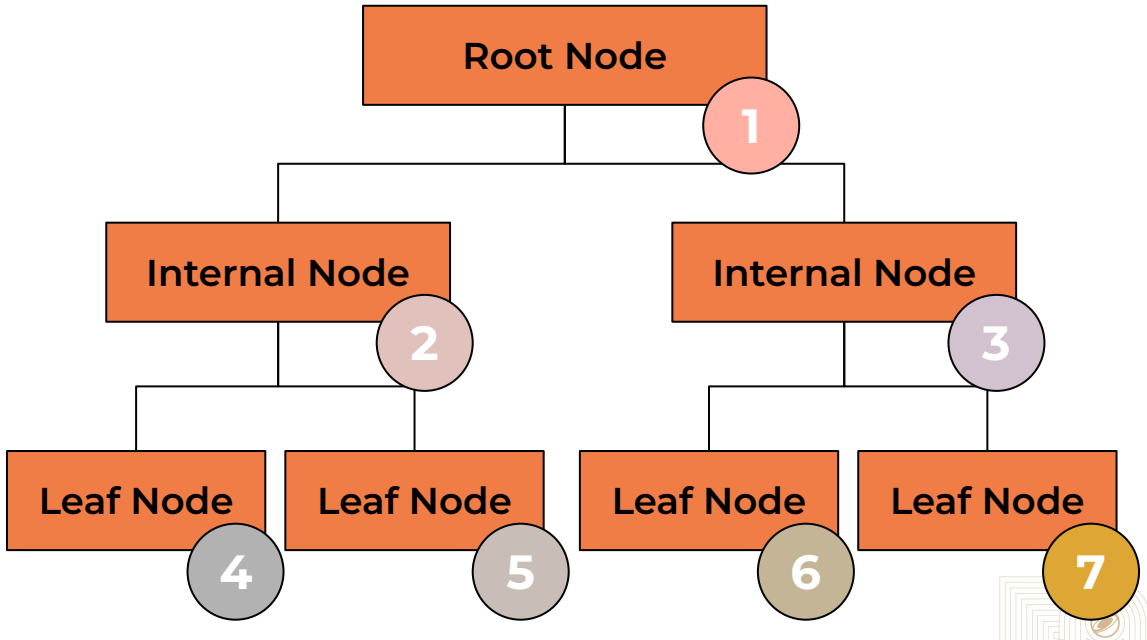
* https://www.disneyanimation.com/data-sets

Depth First vs Breadth First

Depth First



Breadth First



Tree Visitor Methods

Tree

- *visitActiveBBox(...)*
- *visit(...)*
- *visit2(...)*
- *combine(...)*
- *combineExtended(...)*
- *combine2(...)*
- *combine2Extended(...)*

Root Node

- *visitActiveBBox(...)*
- *visit(...)*
- *visit2Node(...)*
- *visit2(...)*
- *combine(...)*
- *combine2(...)*

Internal Node

- *visitActiveBBox(...)*
- *visit(...)*
- *visit2Node(...)*
- *visit2(...)*
- *combine(...)*
- *combine2(...)*

Leaf Node

- *visitActiveBBox(...)*
- *visit(...)*
- *visit2Node(...)*
- *visit2(...)*
- *combine(...)*
- *combine2(...)*

All Tree visitor methods are being *deprecated*, instead:

- Use `tools::visitNodesDepthFirst()` [single-threaded]
- Use `tree::DynamicNodeManager` [multi-threaded]


SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

Parallel Constructs

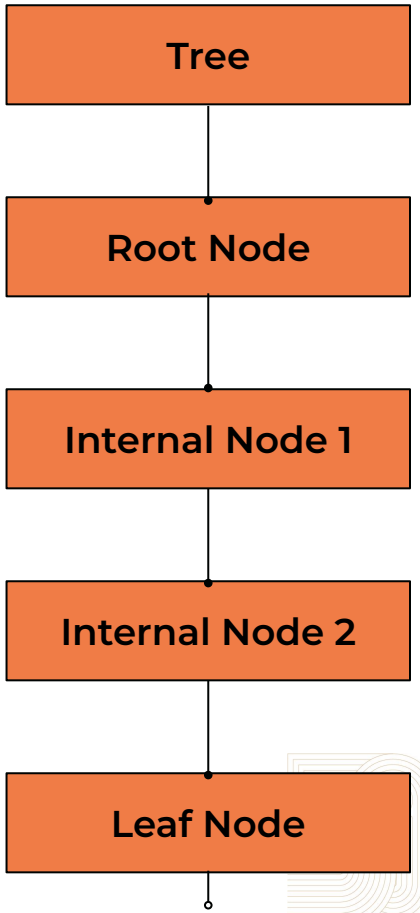



Thread Safety

Modifying Leaf or Tile Values is Thread-Safe

Modifying Active Masks is Thread-Safe

Modifying Topology is *Not* Thread-Safe



Thread Safety

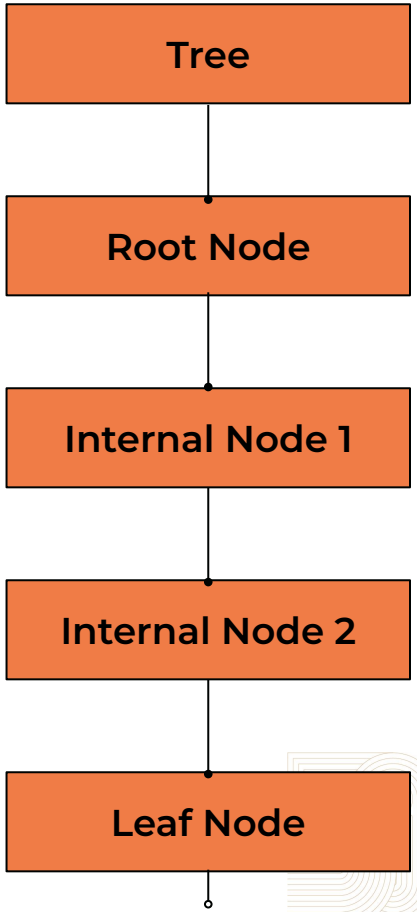
Modifying Leaf or Tile Values is Thread-Safe

Modifying Active Masks is Thread-Safe

~~Modifying Topology is *Not* Thread-Safe~~

Adding/Removing Nodes under Same Parent is *Not* Thread-Safe

Adding/Removing Nodes under Different Parents is Thread-Safe

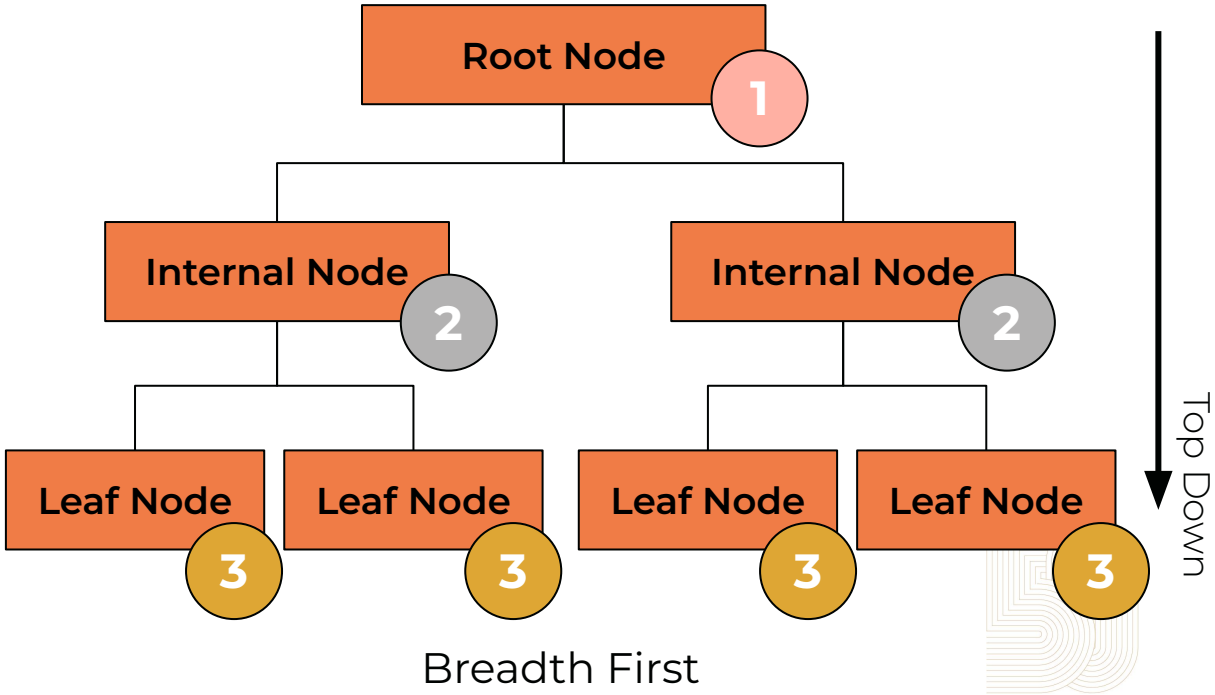


DynamicNodeManager


```
template<typename NodeOp>
void DynamicNodeManager::foreachTopDown(const NodeOp& op, bool threaded = true, size_t grainSize=1);
```

Node functor

- Constructs node arrays lazily instead of up-front
- Primarily designed for topology-changing operations
- Can filter out sub-trees

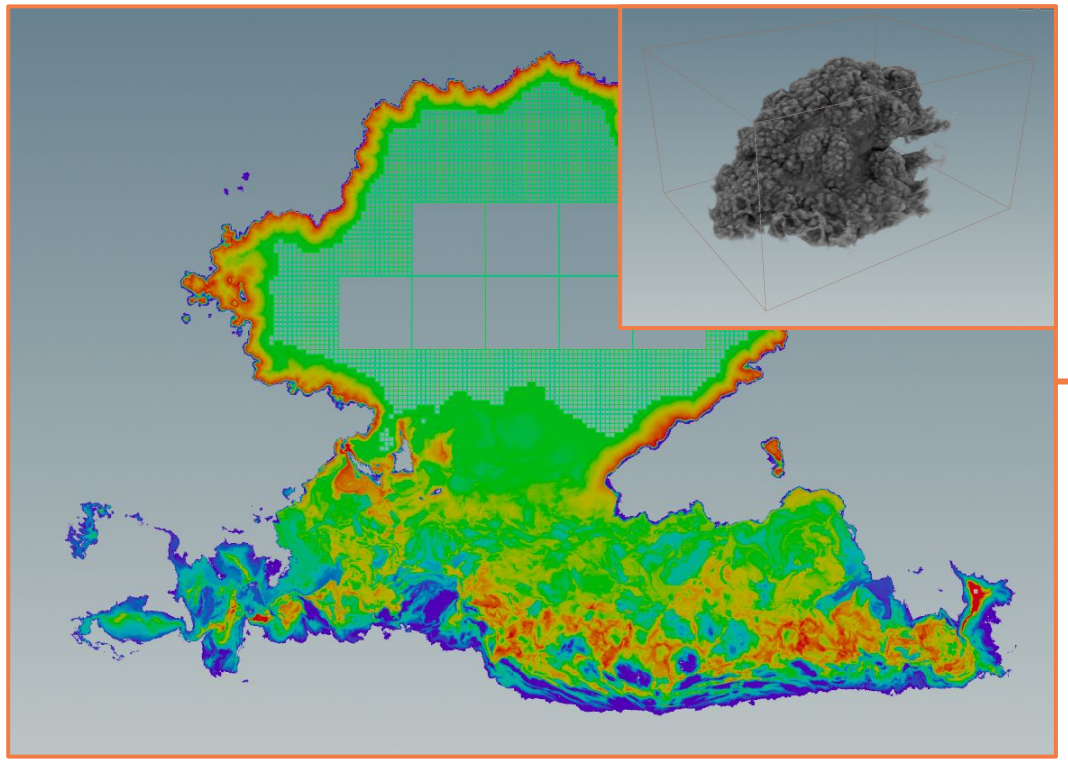



SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

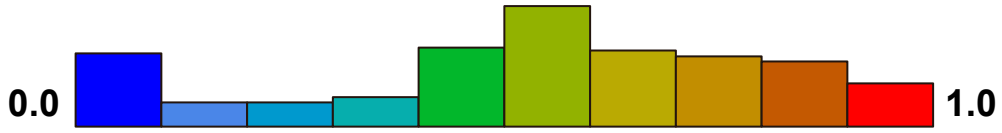
Multi-Threaded Case Study

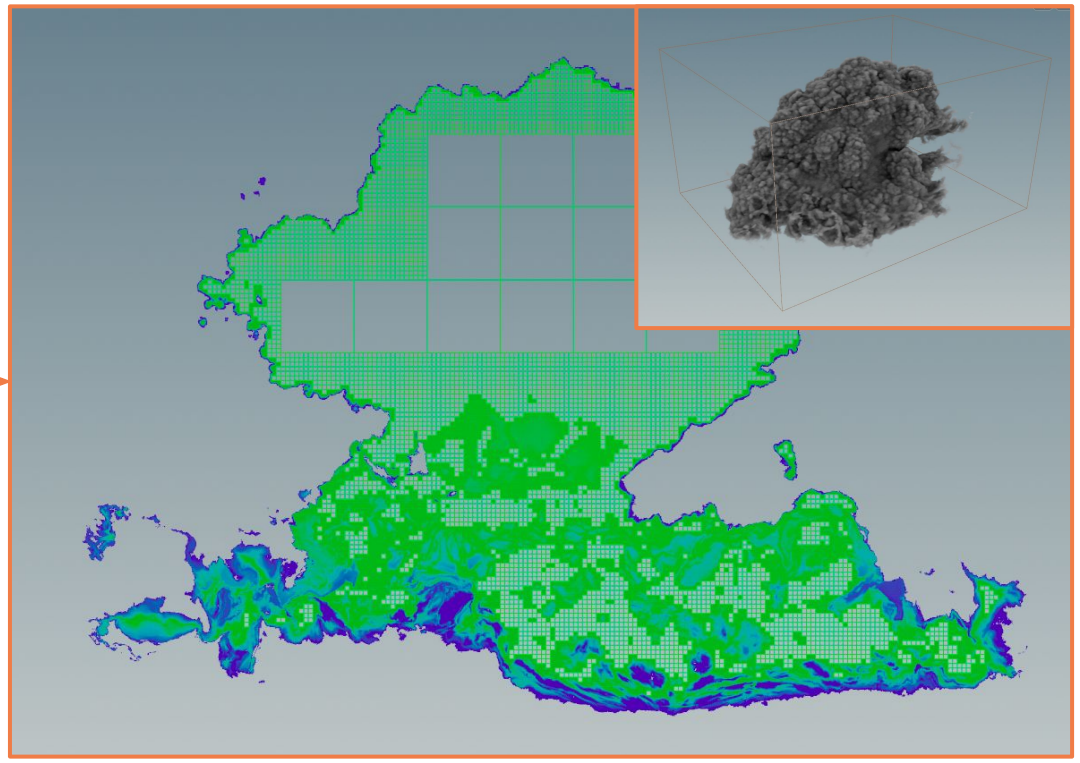



Disney Cloud Value Clamp

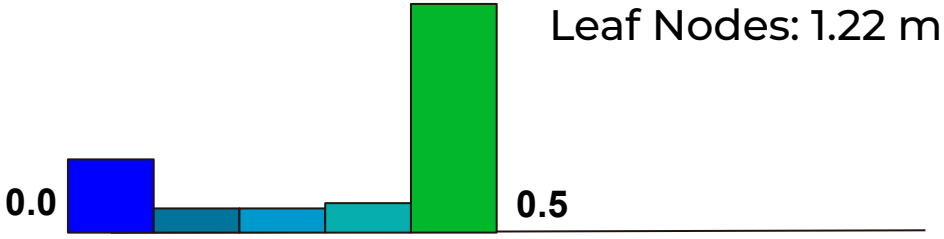


Leaf Nodes: 1.89 million



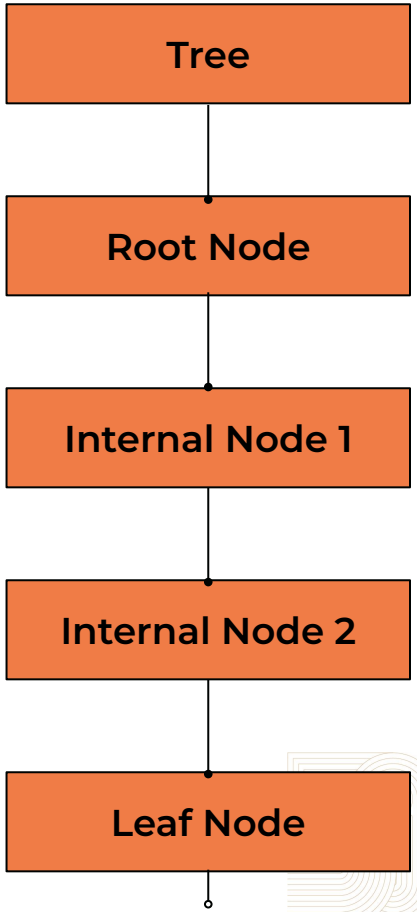


Leaf Nodes: 1.22 million



Copy, Clamp, Prune


```
using namespace openvdb;

void eval(const FloatTree& inputTree)
{
    FloatTree tree(inputTree);

    for (auto iter = tree.beginValueOn(); iter; ++iter) {
        if (*iter > threshold) {
            iter.setValue(threshold);
        }
    }

    tools::prune(tree);
}
```



Copy, Clamp, Prune

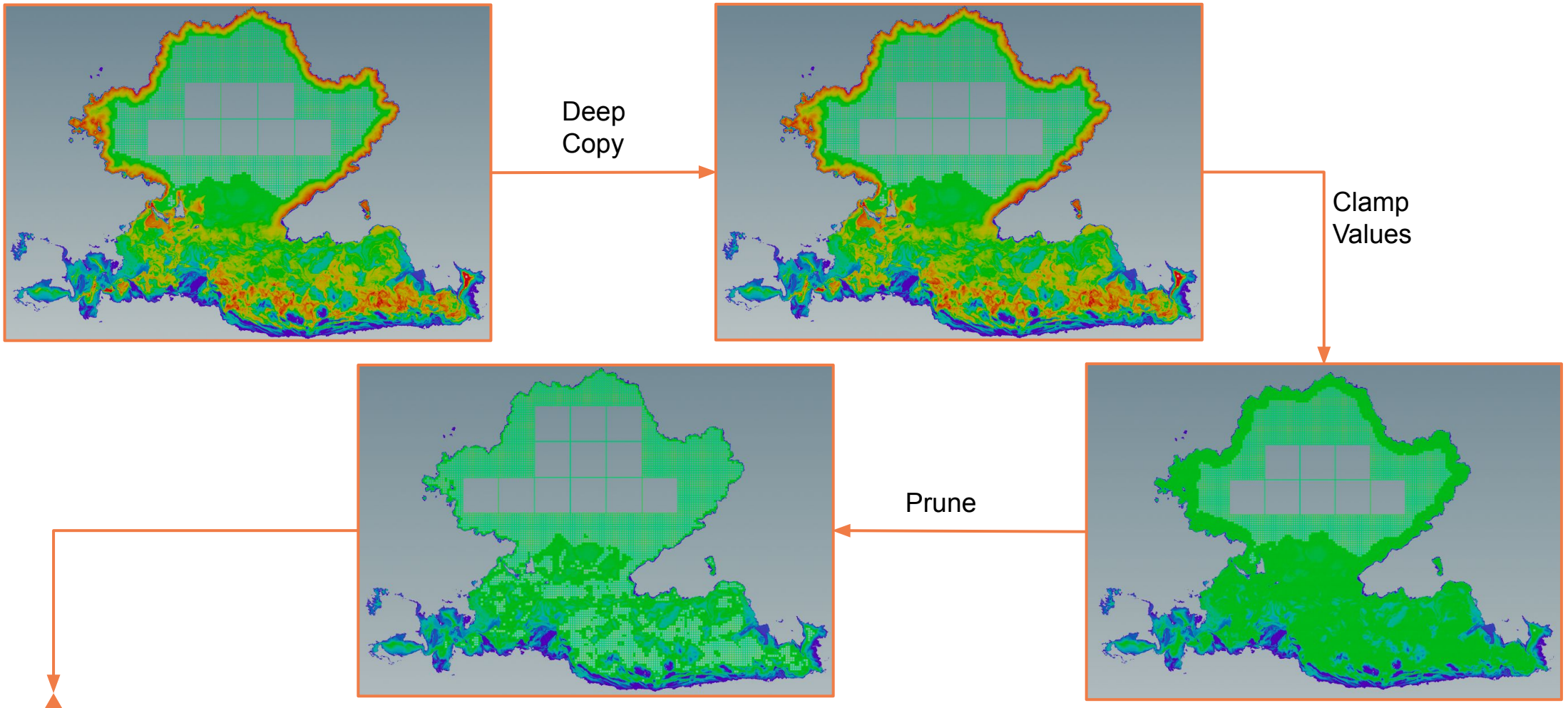

```
for (auto iter = tree.beginValueOn(); iter; ++iter) {
    if (*iter > threshold) {
        iter.setValue(threshold);
    }
}
```

Single-threaded method uses a hierarchical Tree Iterator (8 seconds 130ms)

```
auto clampOp = [&](auto& node)
{
    for (auto iter = node.beginValueOn(); iter; ++iter) {
        if (*iter > threshold) {
            iter.setValue(threshold);
        }
    }
};
tree::NodeManager<FloatTree> nodeManager(tree);
nodeManager.foreachTopDown(clampOp);
```

Parallel method uses a NodeManager and Value Iterators (145ms)

Copy, Clamp, Prune

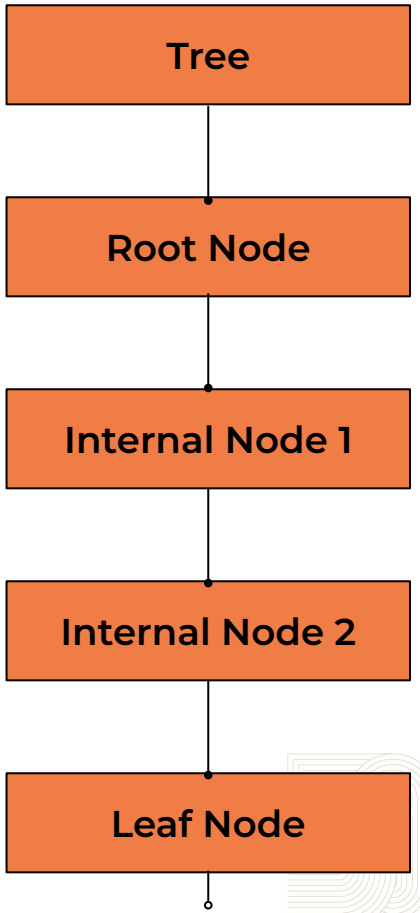


Collapse Leaf?

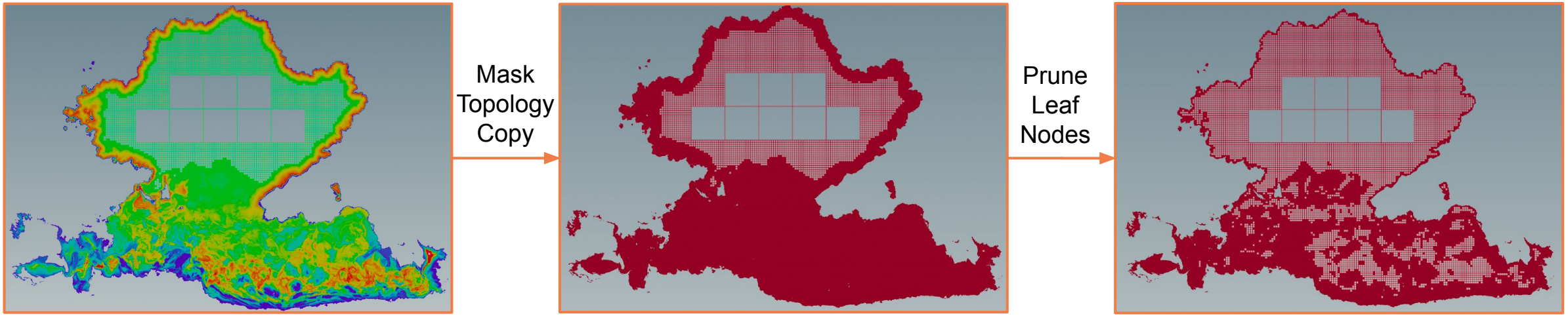

```
bool collapseLeaf(const FloatTree::LeafNodeType& leaf)
{
    if (!leaf.isDense()) {
        // no - at least one of the values is inactive
        return false;
    }

    for (auto iter = leaf.cbeginValueOn(); iter; ++iter) {
        if (*iter < threshold) {
            // no - at least one of the values is less than the threshold
            return false;
        }
    }
    return true;
}
```

Mask Topology - Phase 1


```
MaskTree maskTree(inputTree);

auto deleteMaskLeafsOp = [&](auto& node)
{
    if constexpr(node.LEVEL == 0) { // leaf node
        const auto* sourceLeaf = inputTree.probeLeaf(node.origin());
        if (collapseLeaf(*sourceLeaf)) {
            node.setValuesOff();
        }
    } else if constexpr(node.LEVEL == 1) { // leaf node parent
        for (auto iter = node.beginChildOn(); iter; ++iter) {
            if (iter->isEmpty()) {
                node.addTile(iter.pos(), true, true);
            }
        }
    }
};
tree::NodeManager<MaskTree> maskNodeManager(maskTree);
maskNodeManager.foreachBottomUp(deleteMaskLeafsOp);
```

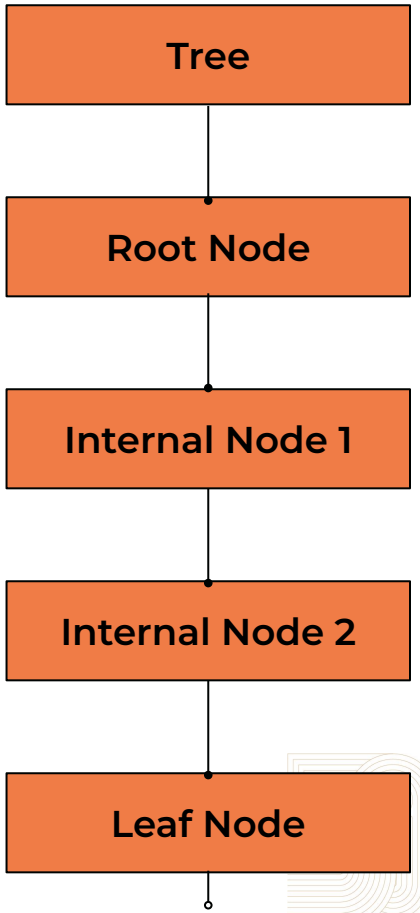


Mask Topology - Phase 1

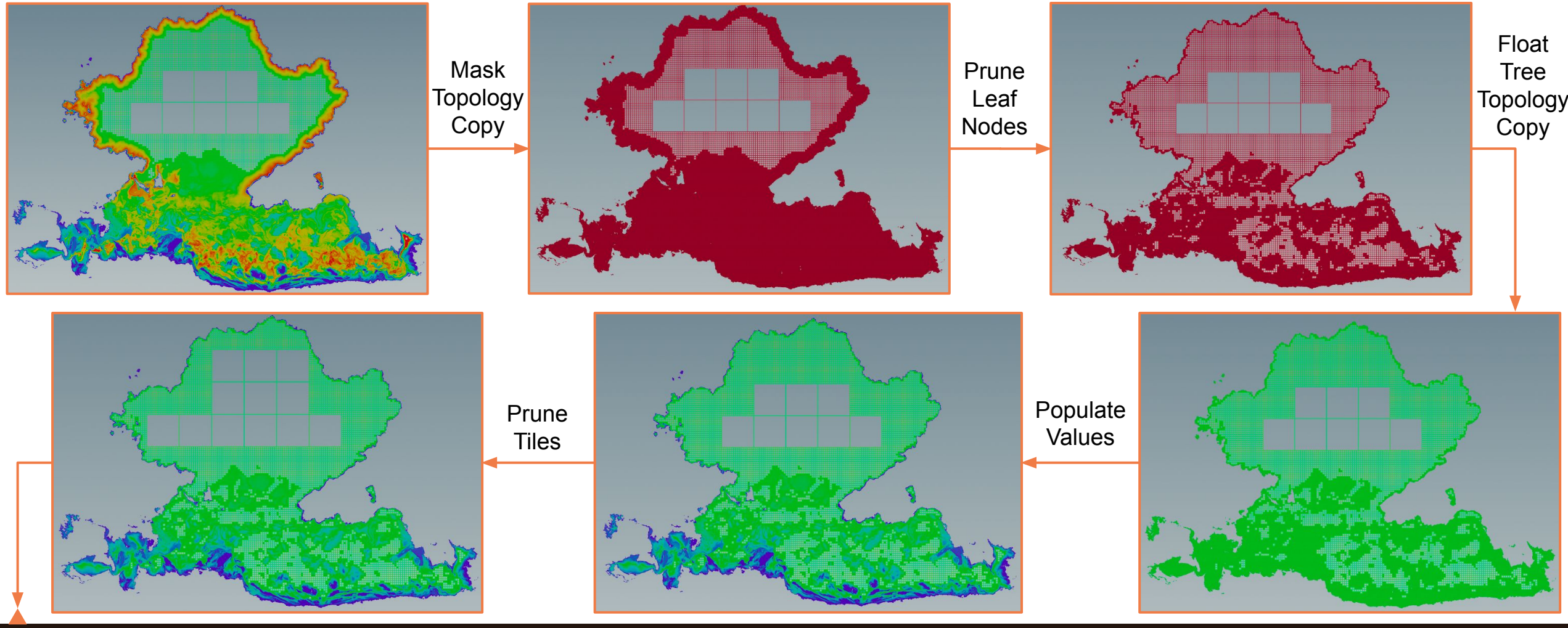


Mask Topology - Phase 2

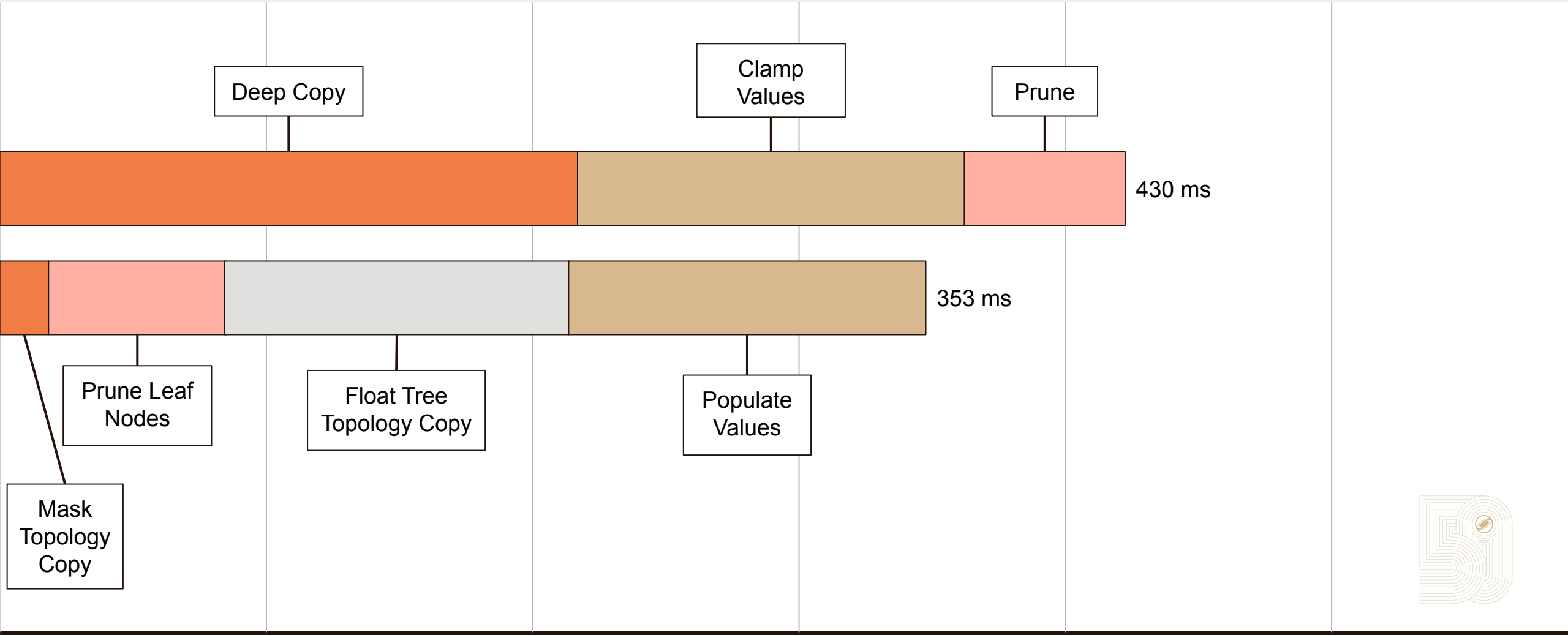

```
FloatTree tree(maskTree, threshold, TopologyCopy());
tree.root().setBackground(0.0f, /*updateChildNodes=*/false);

auto deepCopyFloatOp = ...
tree::NodeManager<FloatTree> nodeManager(tree);
CopyFloatValuesOp deepCopyFloatOp(inputTree);
nodeManager.foreachTopDown(deepCopyFloatOp);

tools::pruneTiles(tree);
```



Mask Topology

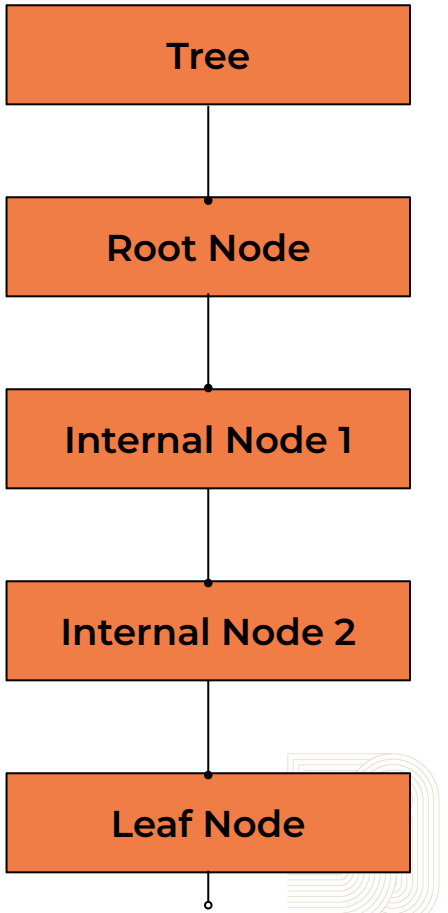


Naive vs Mask Topology

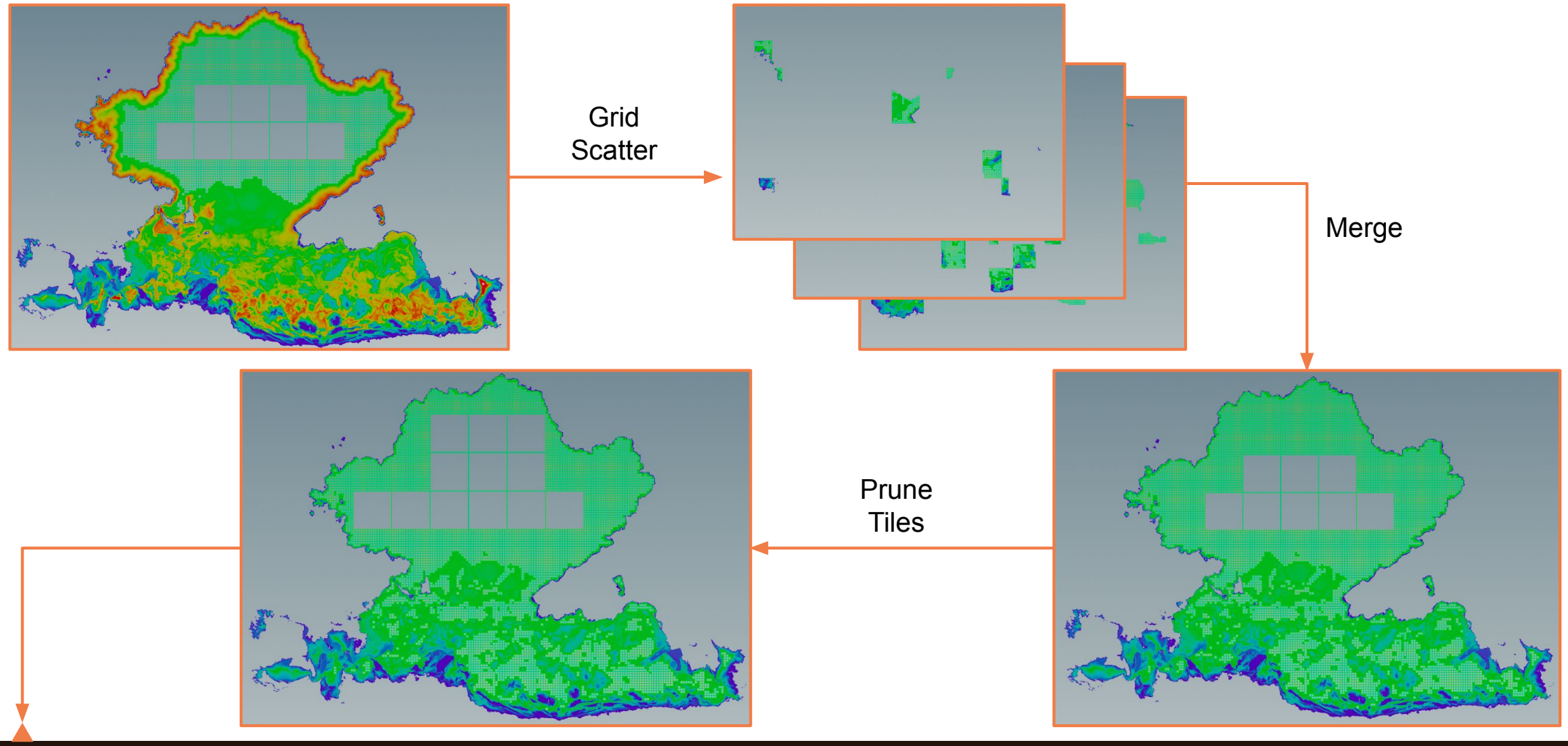


Scatter Merge

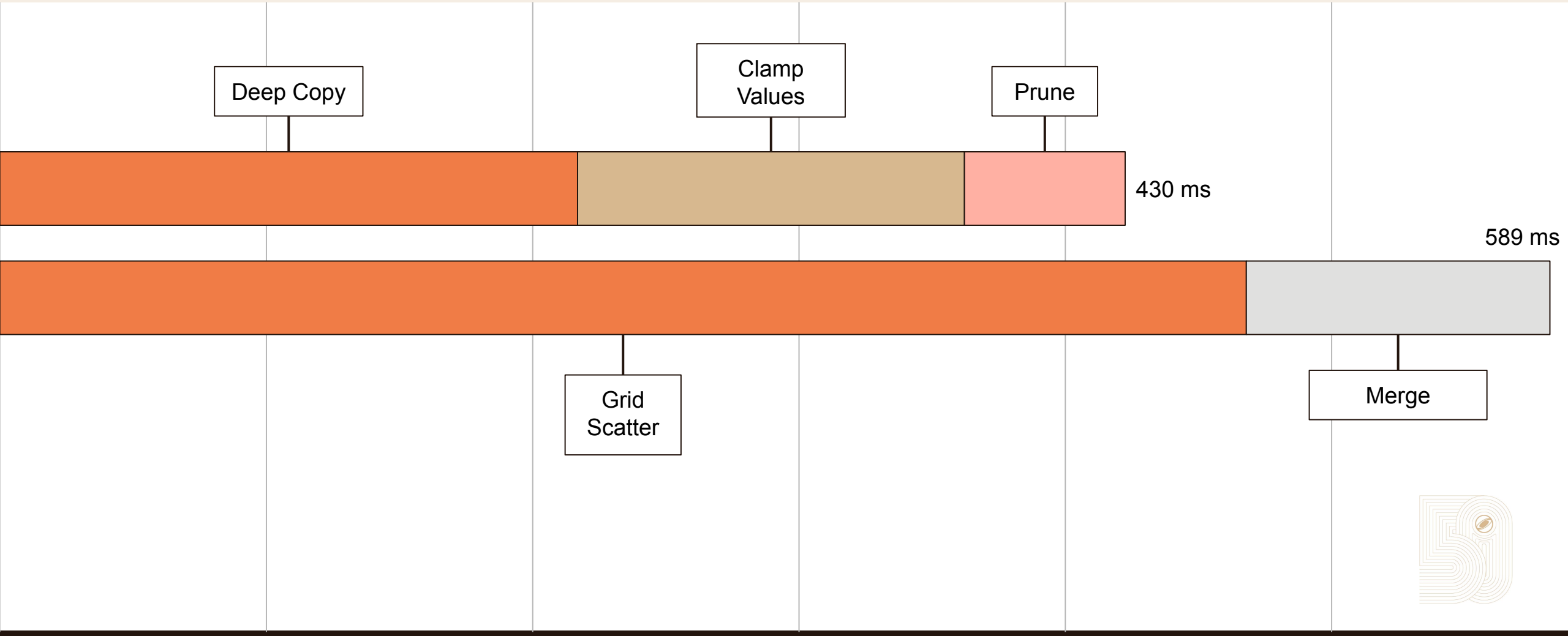

```
FloatTree tree(0.0f);

tbb::enumerable_thread_specific<FloatTree> pool(tree);

struct ScatterOp
{
   ScatterOp(tbb::enumerable_thread_specific<FloatTree>&) { ... }
   void operator()(const FloatTree::LeafNodeType& leaf) const
   {
       FloatTree& tree = pool.local();
       ...
   }
};
ScatterOp scatterOp(pool);
tree::NodeManager<const FloatTree> nodeManager(inputTree);
nodeManager.foreachTopDown(scatterOp);

for (auto it = pool.begin(); it < pool.end(); ++it) {
   tree.merge(*it);
}
```



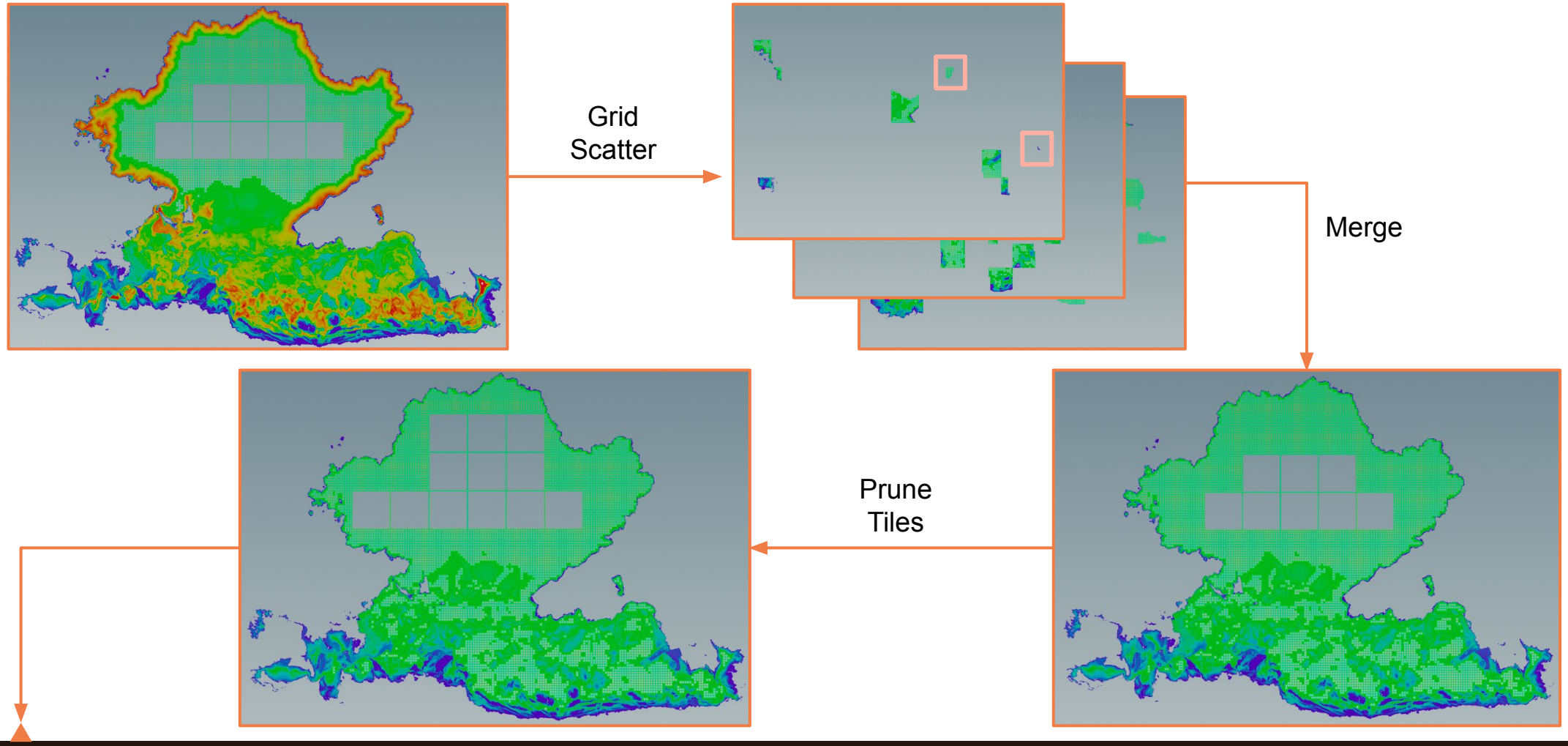
Scatter Merge



Naive vs Scatter Merge (Leaf Node)

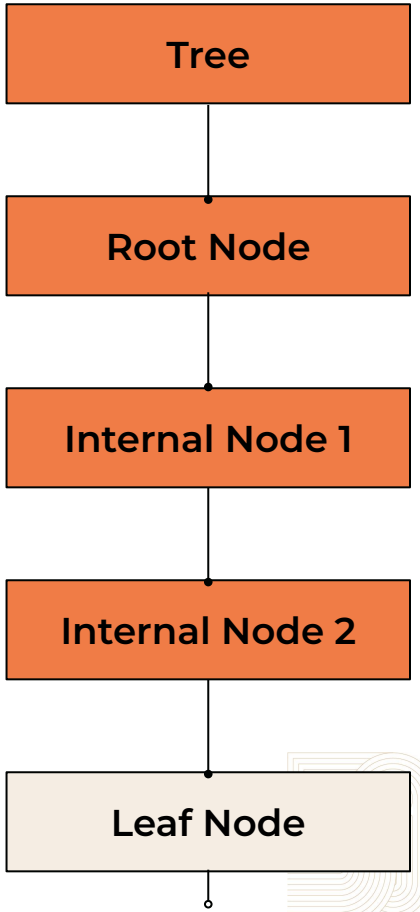


Scatter Merge

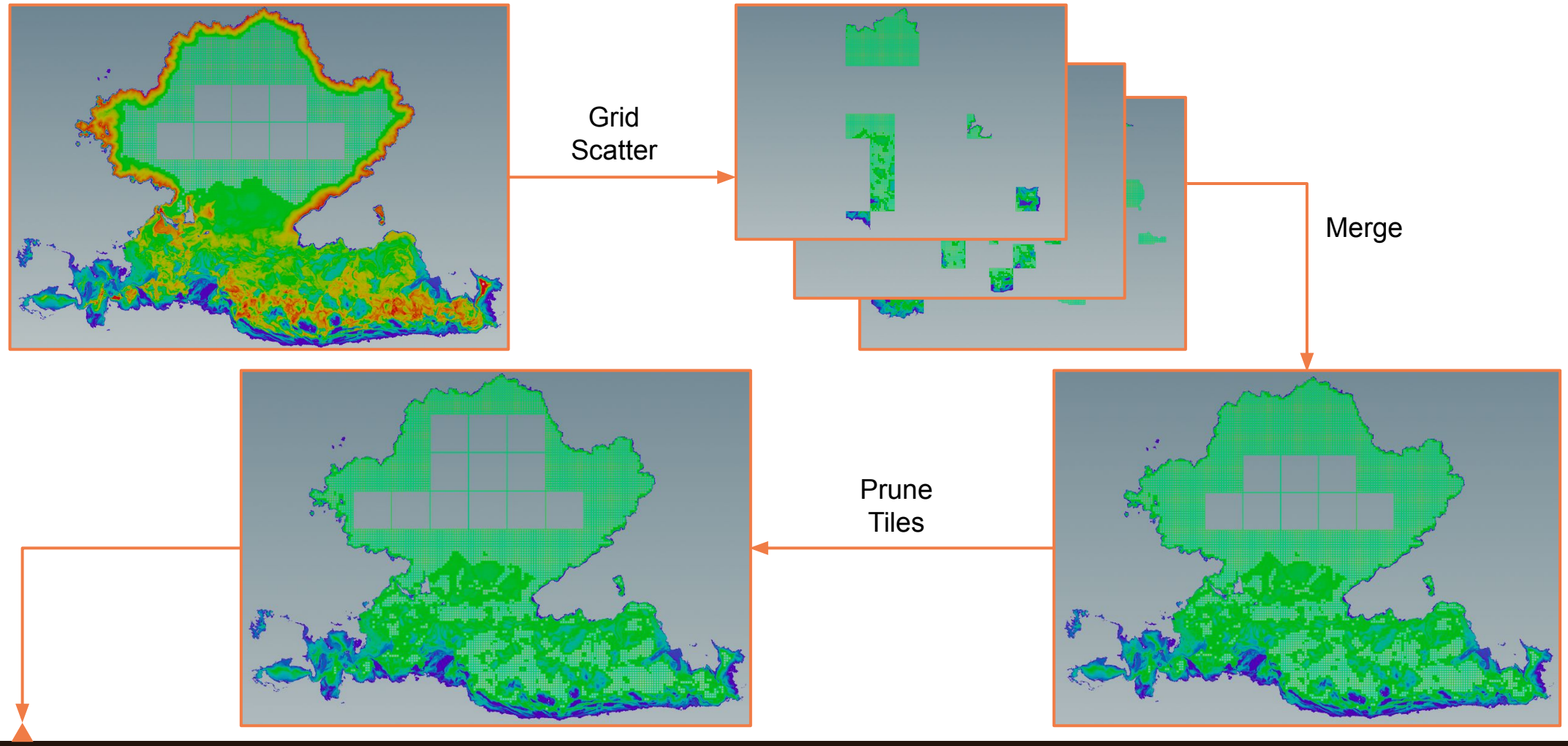


Scatter Merge

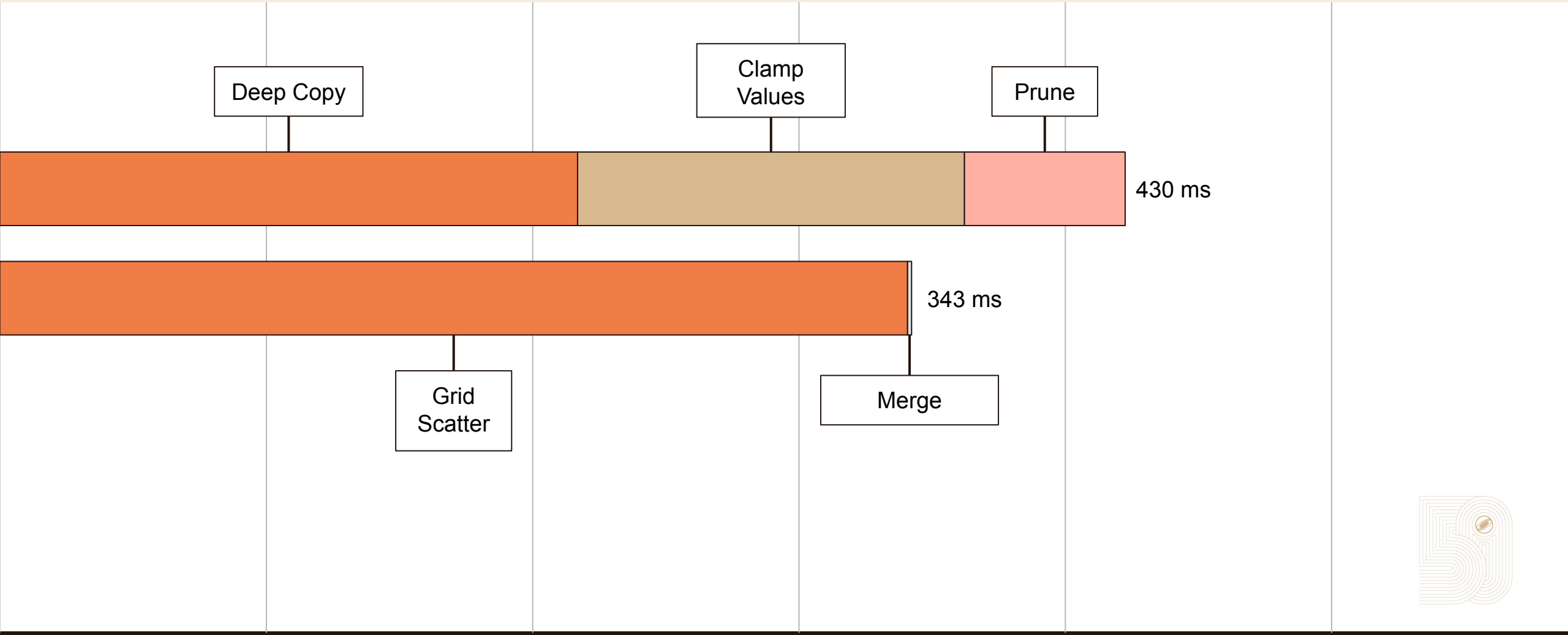

```
template <typename T> void operator()(const T& node) const
{
   FloatTree& tree = mPool.local();
   if constexpr (node.LEVEL == 1) { // parent of leaf node
       for (auto leaf = node.cbeginChildOn(); leaf; ++leaf) {
           ...
       }
   }
}

void operator()(const FloatTree::LeafNodeType& leaf) const
{
   // do nothing
}
```



Scatter Merge

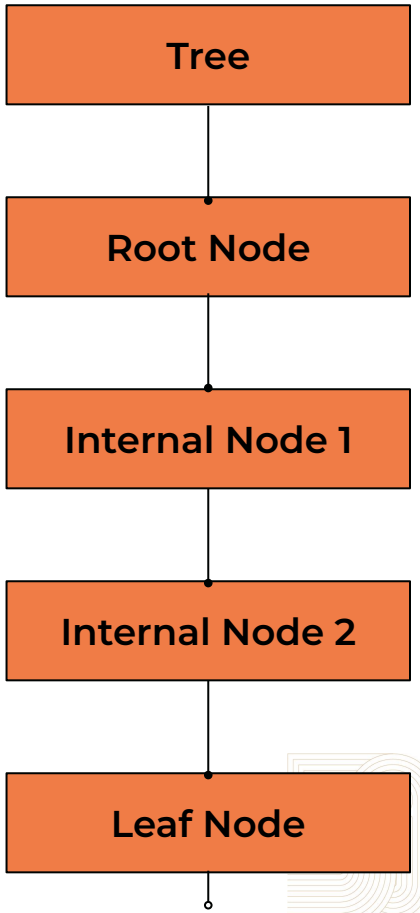


Naive vs Scatter Merge (Internal Node)



Dynamic Build

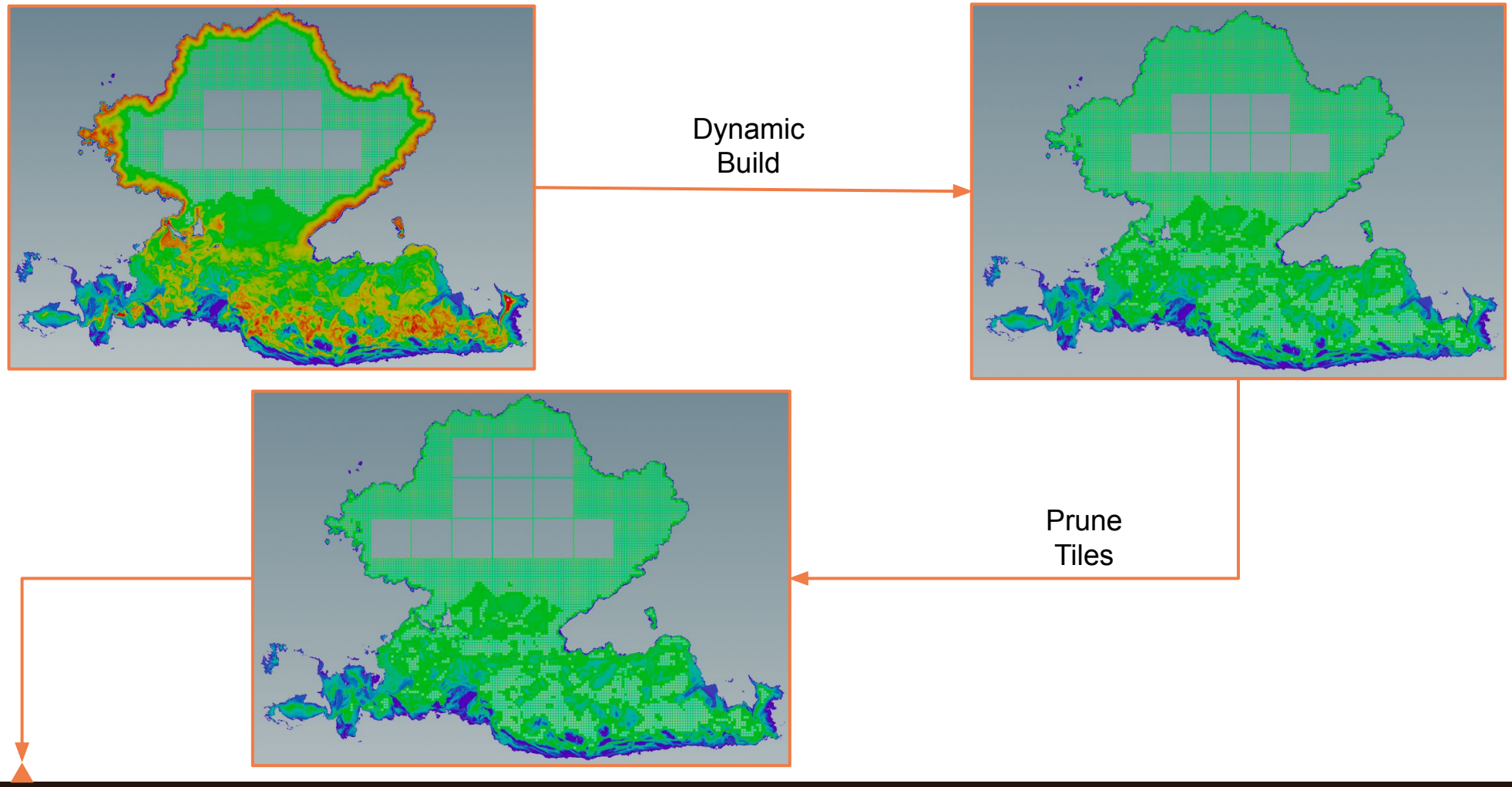

```
FloatTree tree(0.0f);

DynamicBuildOp dynamicBuildOp(inputTree);
tree::DynamicNodeManager<FloatTree> nodeManager(tree);
nodeManager.foreachTopDown(dynamicBuildOp);

tools::pruneTiles(tree);
```



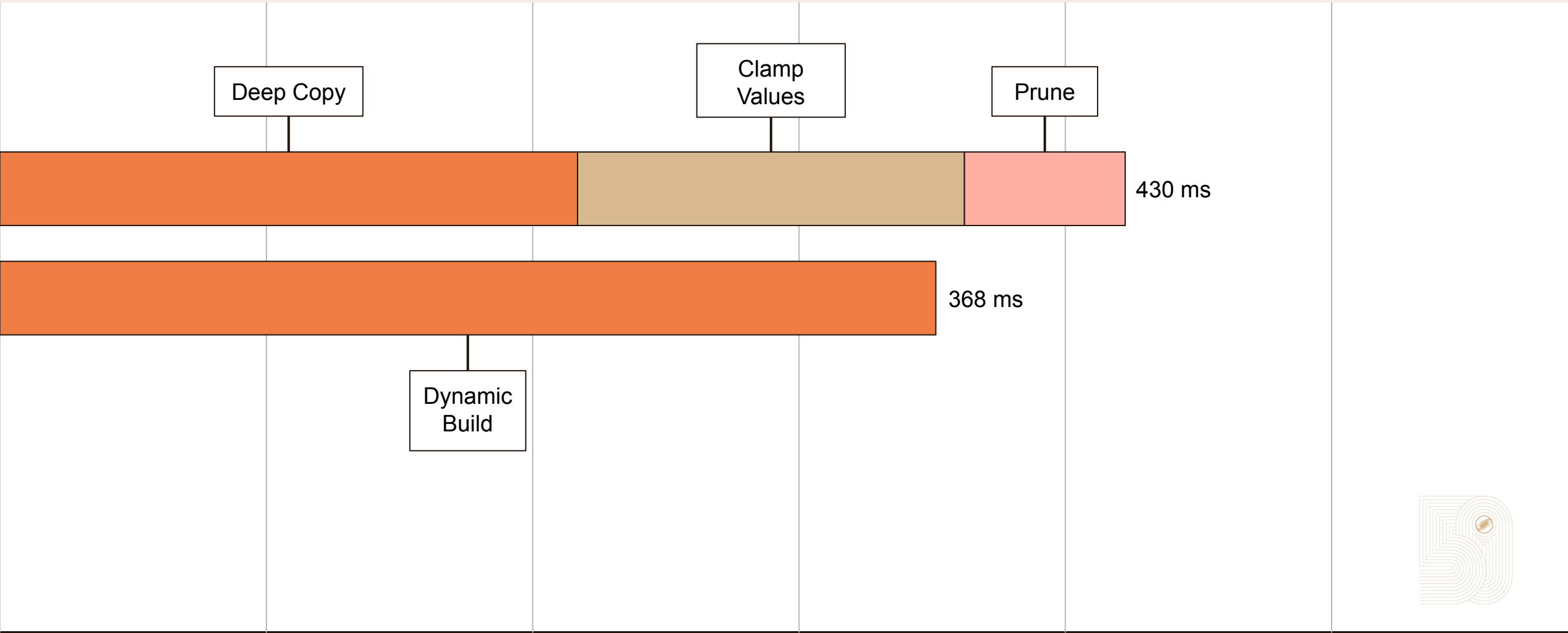
Dynamic Build


```
struct DynamicBuildOp
{
    template <typename T> bool operator()(T& node, size_t) const
    {
        const auto* sourceNode = inputTree.root().template probeConstNode<T>(node.origin());
        for (auto iter = sourceNode->cbeginChildOn(); iter; ++iter) {
            if (collapseLeaf(*iter)) {
                node.addTile(iter.pos(), threshold, true);
            } else {
                auto* child = new typename T::ChildNodeType(iter.getCoord(), 0.0f, false);
                for (auto valueIter = iter->cbeginValueOn(); valueIter; ++valueIter) {
                    float value = *valueIter > threshold ? threshold : *valueIter;
                    child->setValueOnly(valueIter.pos(), value);
                    child->setValueOn();
                }
                node.addChild(child);
            }
        }
        return true;
    }
};
```

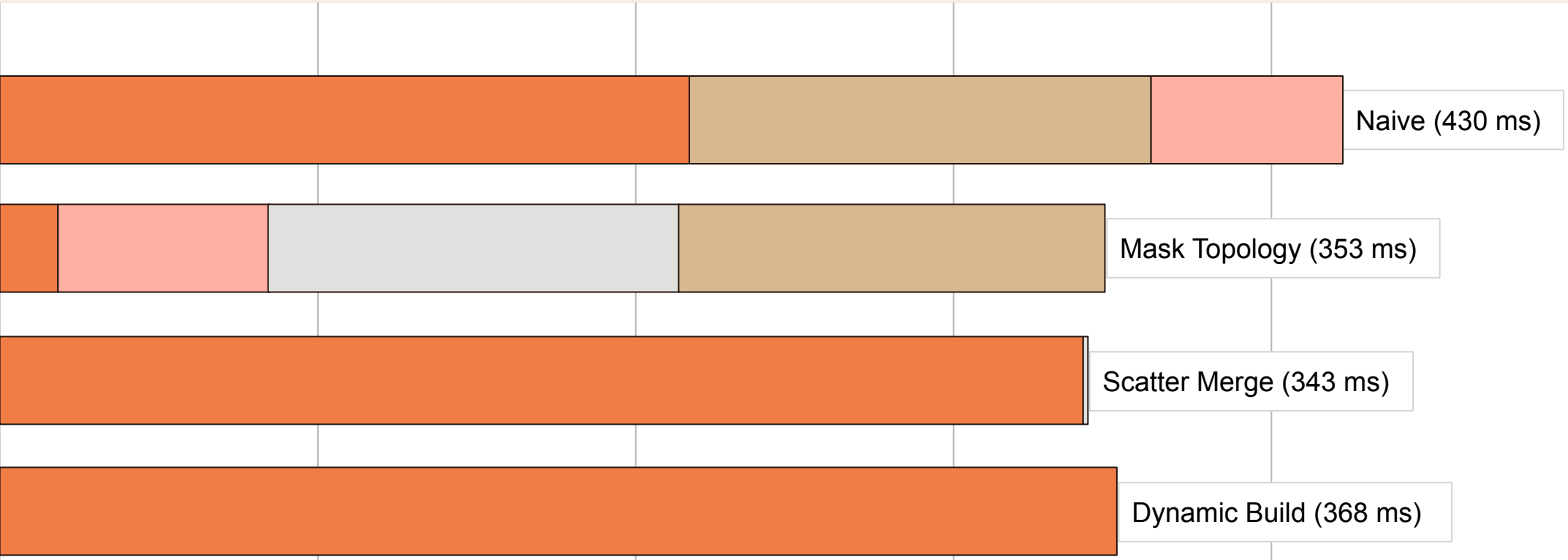
Dynamic Build



Naive vs Dynamic Build



Compare Timings



Dynamic Build Further Optimizations

Optimization 1: Prevent Leaf Node Dispatch

```
tree::DynamicNodeManager<FloatTree> nodeManager(tree);
```

```
tree::DynamicNodeManager<FloatTree, FloatTree::DEPTH-2> nodeManager(tree);
```

Optimization 2: Bulk Write Mask Data

```
for (auto valueIter = iter->cbeginValueOn(); valueIter; ++valueIter) {
   leaf->setValueOn(valueIter.pos());
}
```

```
leaf->setValueMask(iter->getValueMask());
```

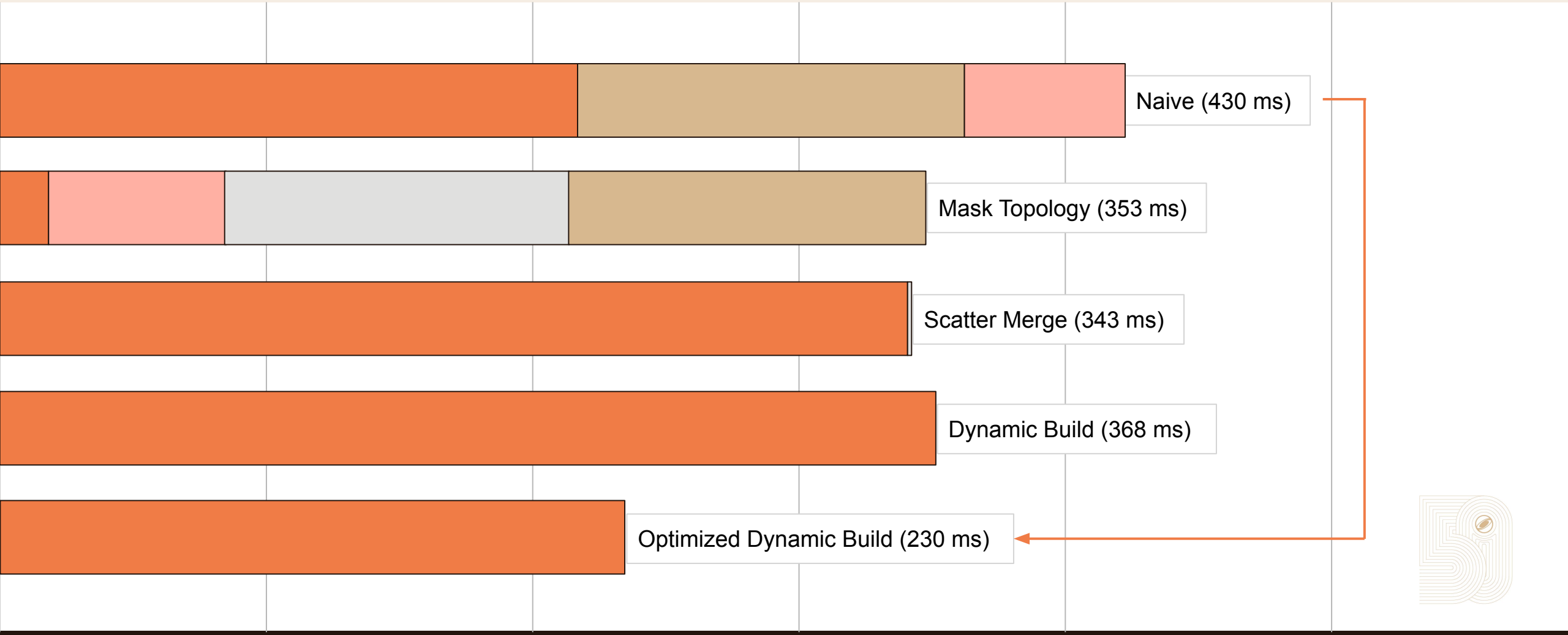
Dynamic Build Further Optimizations

Optimization 3: Read/Write Leaf Node Buffers Directly

```
for (auto valueIter = iter->cbeginValueOn(); valueIter; ++valueIter) {
   Index idx = valueIter.pos();
   float value = (*valueIter > threshold) ? threshold : *valueIter;
   leaf->setValueOnly(idx, value);
}
```

```
const float* sourceData = iter->buffer().data();
float* targetData = child->buffer().data();
for (auto valueIter = iter->cbeginValueOn(); valueIter; ++valueIter) {
   Index idx = valueIter.pos();
   float value = (sourceData[idx] > threshold) ? threshold : sourceData[idx];
   targetData[idx] = value;
}
```

Compare Timings




THE PREMIER CONFERENCE & EXHIBITION ON COMPUTER GRAPHICS & INTERACTIVE TECHNIQUES

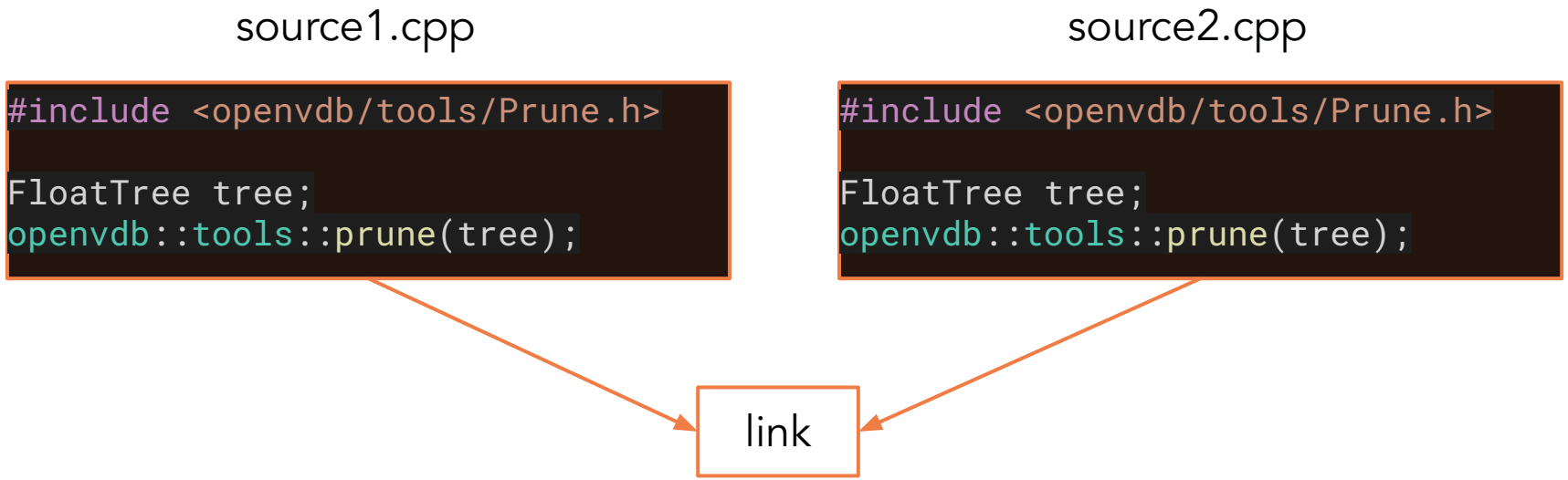
SIGGRAPH 2023
LOS ANGELES+ 6-10 AUG

Build Performance




Slow Compile Times

- OpenVDB frequently cited as an example of slow build performance
- Both the core library and any client code that includes the library headers
- Heavy use of C++ Templates leads to *expensive* template instantiation costs



Explicit Template Instantiation

- Pre-computes and stores lots of common template instantiations in core library
- Suppresses template instantiations in client code
- Enabled by default - switch off when developing core library
- See end of openvdb/tools source files for implementation examples

```
cmake -DUSE_EXPLICIT_INSTANTIATION=ON ...
```

Explicit Instantiation	Build Time (1 Core)	Build Time (32 Cores)	Disk Space (libopenvdb.so)
Disabled	**4m 50s**	**3m 12s**	**9.7MB**
Enabled	**32m 21s**	**5m 55s**	**74.3MB**

Conclusion

Benchmarks: git clone git@github.com:danrbailey/siggraph2023_openvdb.git