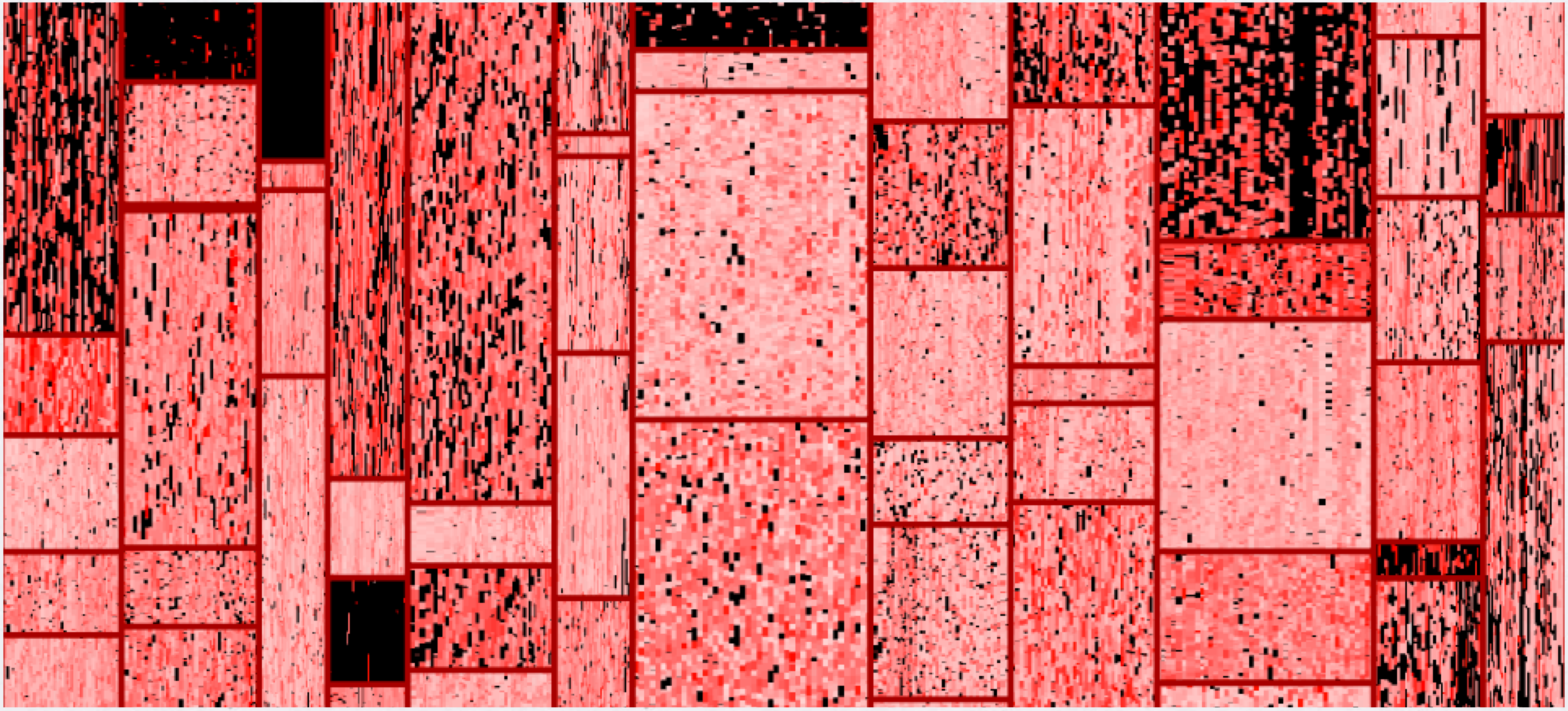
SUPERCONDUCTOR

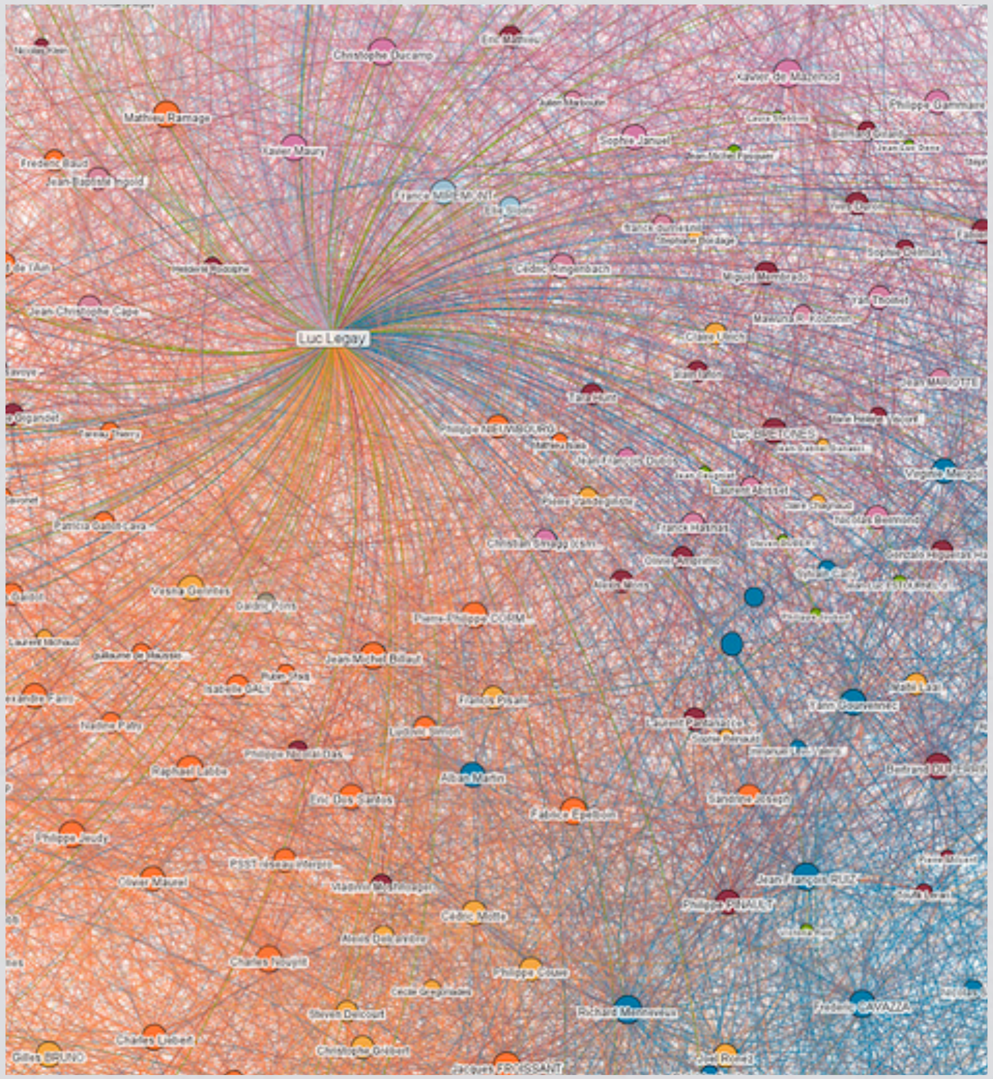
1 Million Nodes at 30 FPS



Matthew E. Torok, UC Berkeley Par Lab
mtorok@eecs.berkeley.edu

New Age of Visualizations

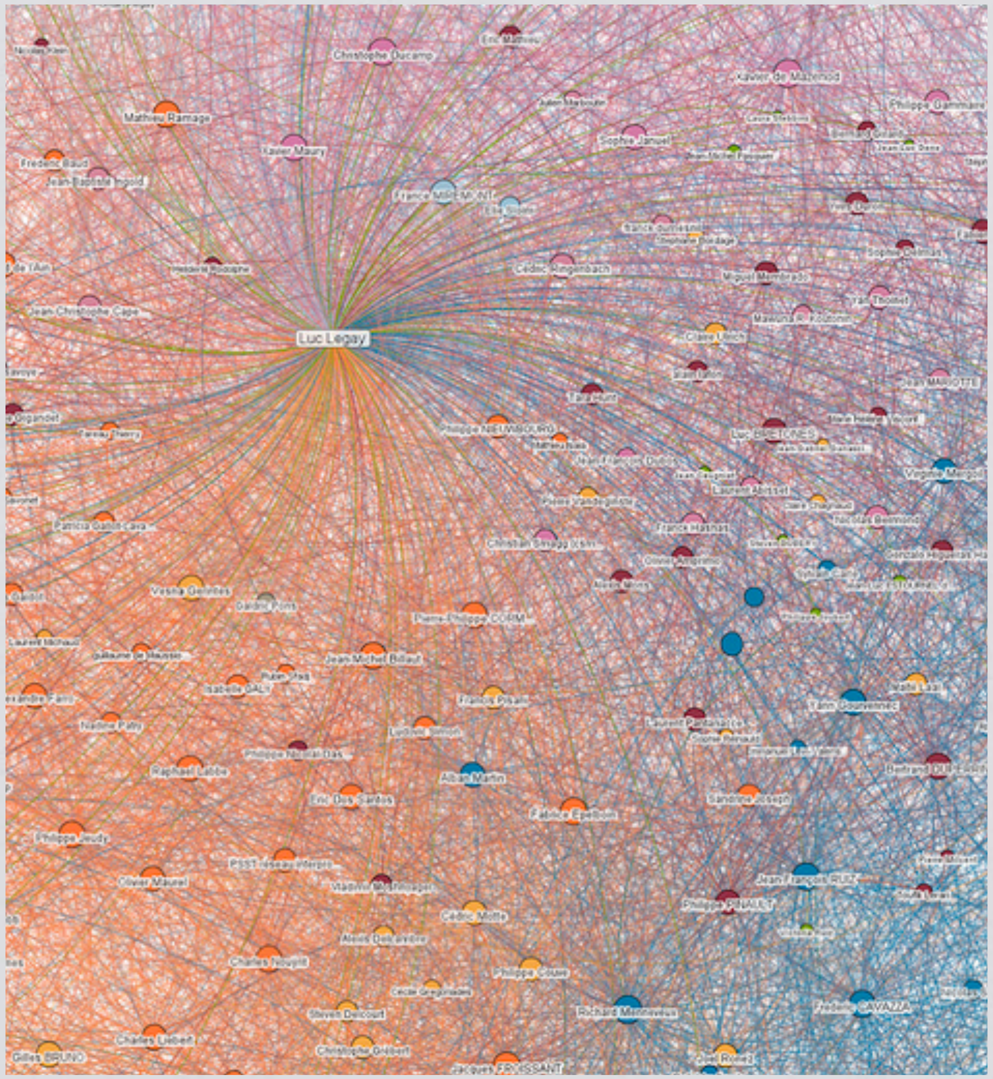
New Age of Visualizations



Big Data

speed

New Age of Visualizations

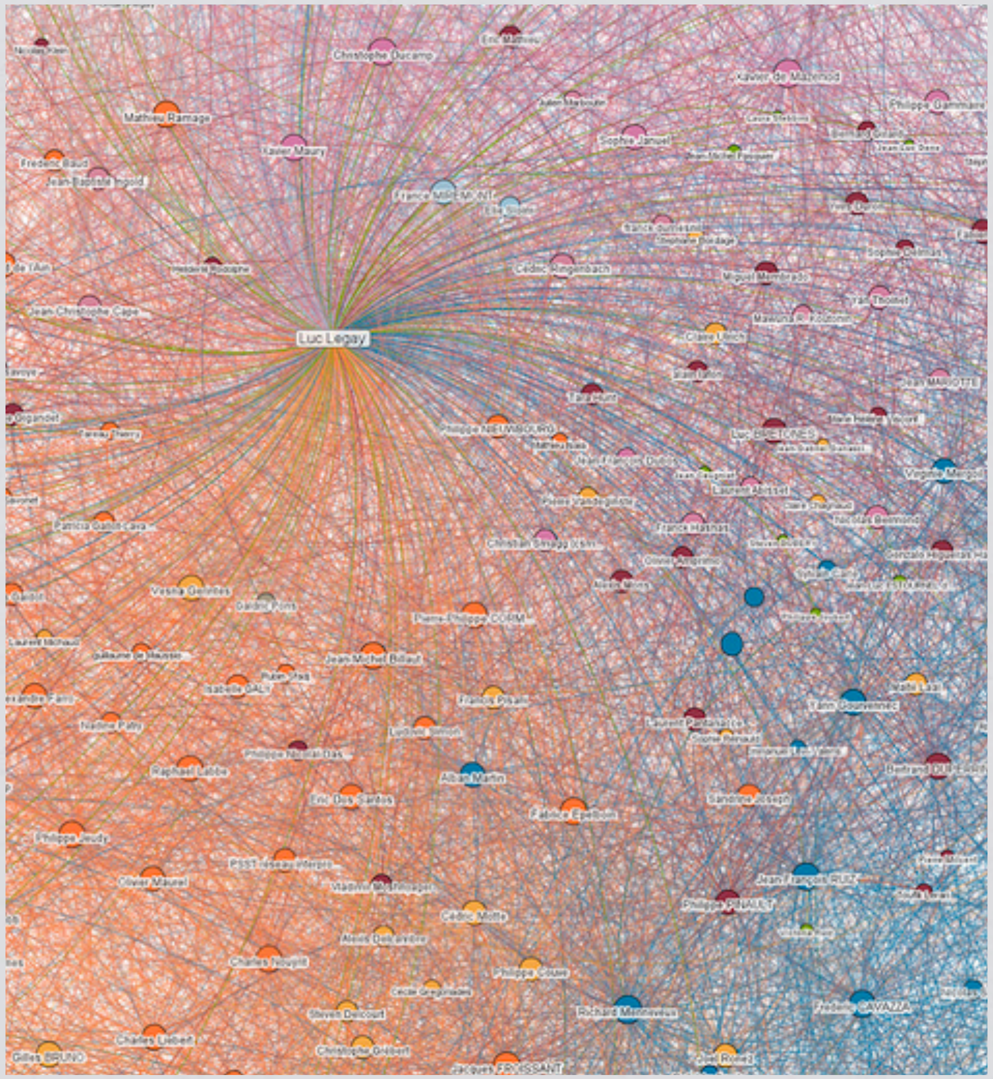


Big Data

speed

- and -

New Age of Visualizations



Big Data

speed

- and -

Small Devices

power

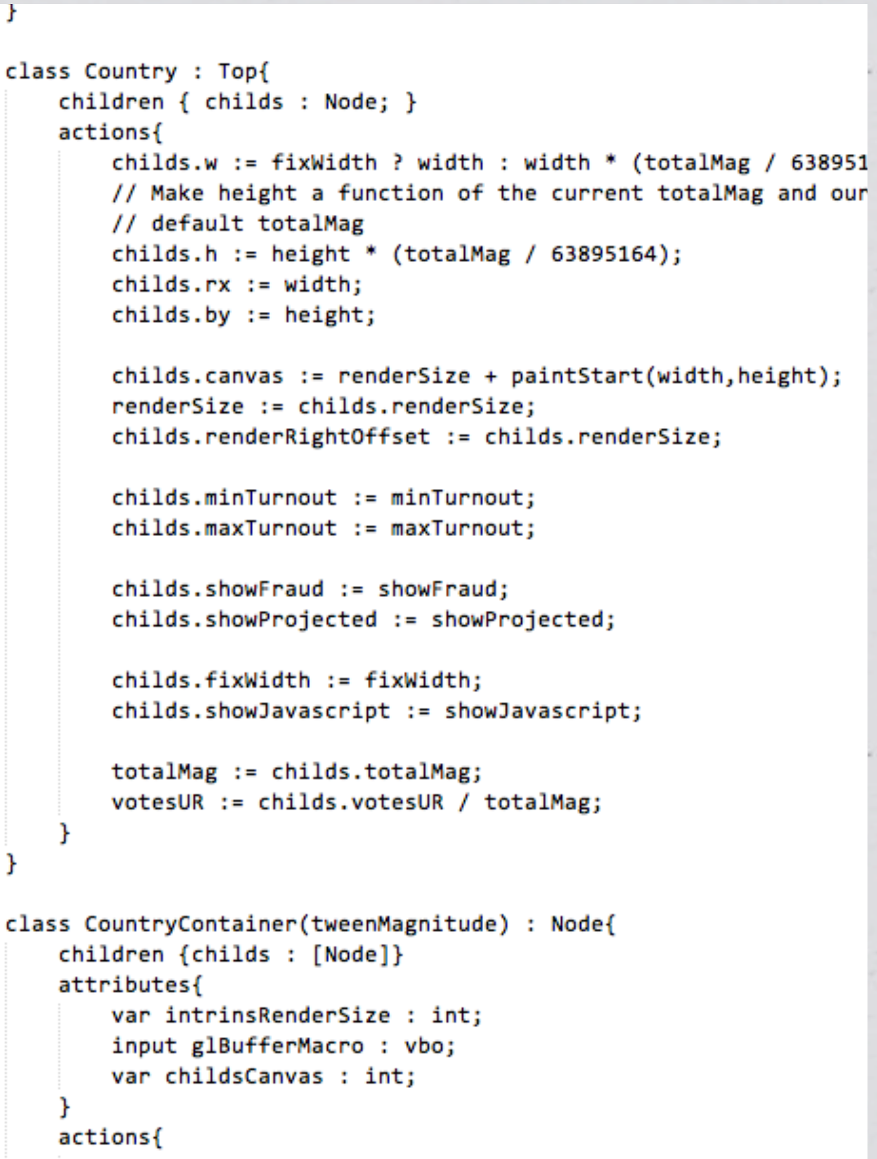
Superconductor is...

- Data visualization system
- Use GPU for layout and rendering
  - Fast: 100,000 nodes at 27fps
  - Power efficient: data parallelism
- Web-Friendly: OpenCL today, WebCL tomorrow

ARCHITECTURE

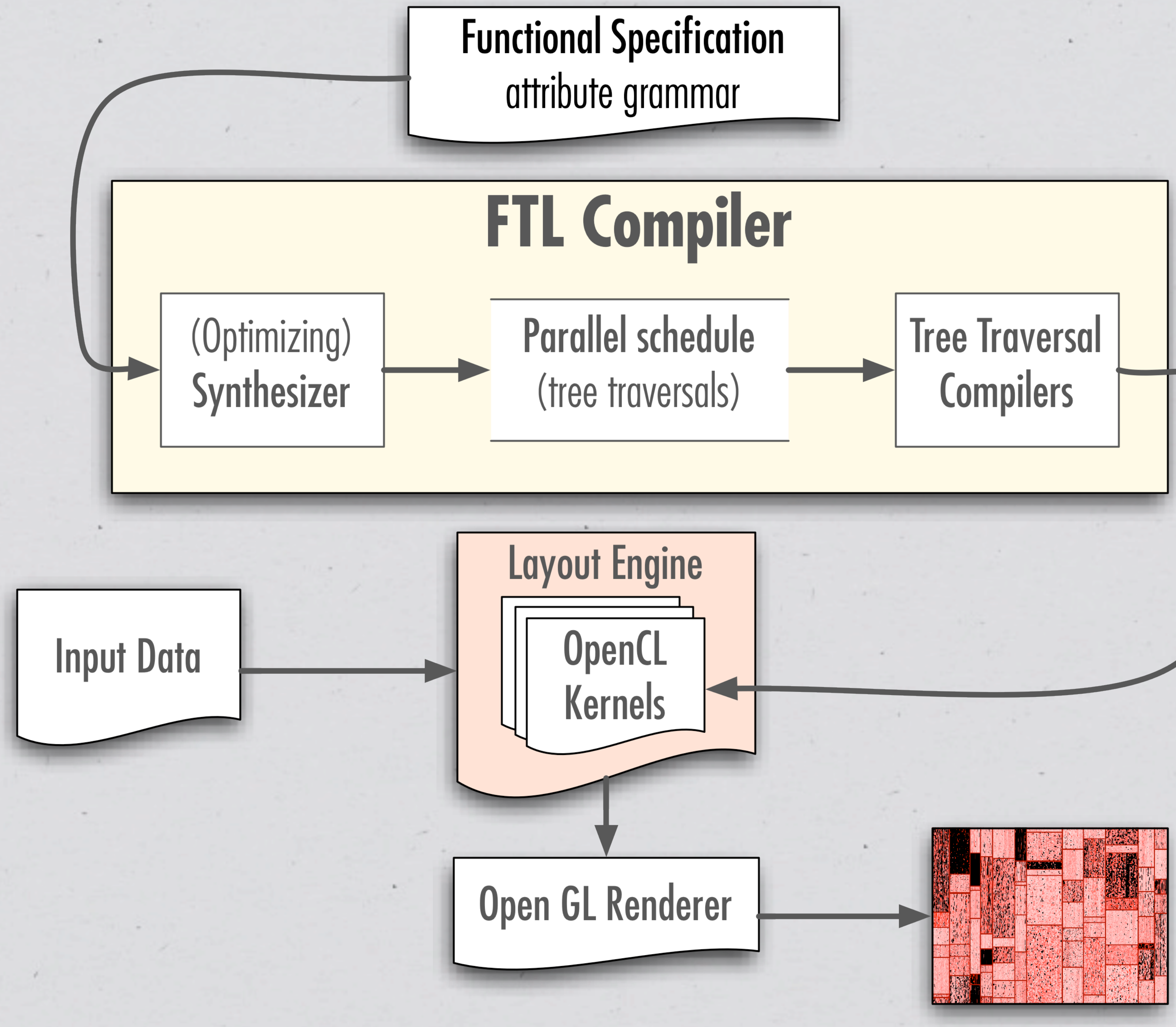
Creating Widgets

- Use FTL language and compiler
- Simple, declarative syntax
- Simplicity of Javascript, speed of GPU
- FTL: Synthesizes traversal schedule from layout spec, uses specializers to create tuned OpenCL kernels



```
}

class Country : Top{
    children { childs : Node; }
    actions{
        childs.w := fixWidth ? width : width * (totalMag / 638951
        // Make height a function of the current totalMag and our
        // default totalMag
        childs.h := height * (totalMag / 63895164);
        childs.rx := width;
        childs.by := height;

        childs.canvas := renderSize + paintStart(width,height);
        renderSize := childs.renderSize;
        childs.renderRightOffset := childs.renderSize;

        childs.minTurnout := minTurnout;
        childs.maxTurnout := maxTurnout;

        childs.showFraud := showFraud;
        childs.showProjected := showProjected;

        childs.fixWidth := fixWidth;
        childs.showJavascript := showJavascript;

        totalMag := childs.totalMag;
        votesUR := childs.votesUR / totalMag;
    }
}

class CountryContainer(tweenMagnitude) : Node{
    children {childs : [Node]}
    attributes{
        var intrinsRenderSize : int;
        input glBufferMacro : vbo;
        var childsCanvas : int;
    }
    actions{
```

FTL Data Flow



Functional Specification
attribute grammar

FTL Compiler

(Optimizing)
Synthesizer

Parallel schedule
(tree traversals)

Tree Traversal
Compilers

Layout Engine

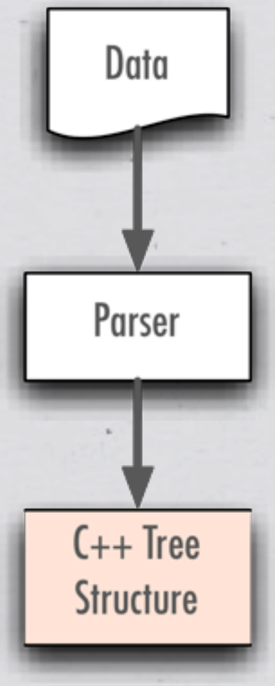
OpenCL
Kernels

Input Data

Open GL Renderer

Runtime

CPU



Data

Parser

C++ Tree Structure

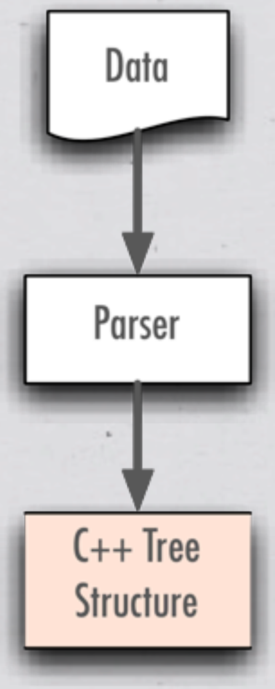
GPU

Runtime

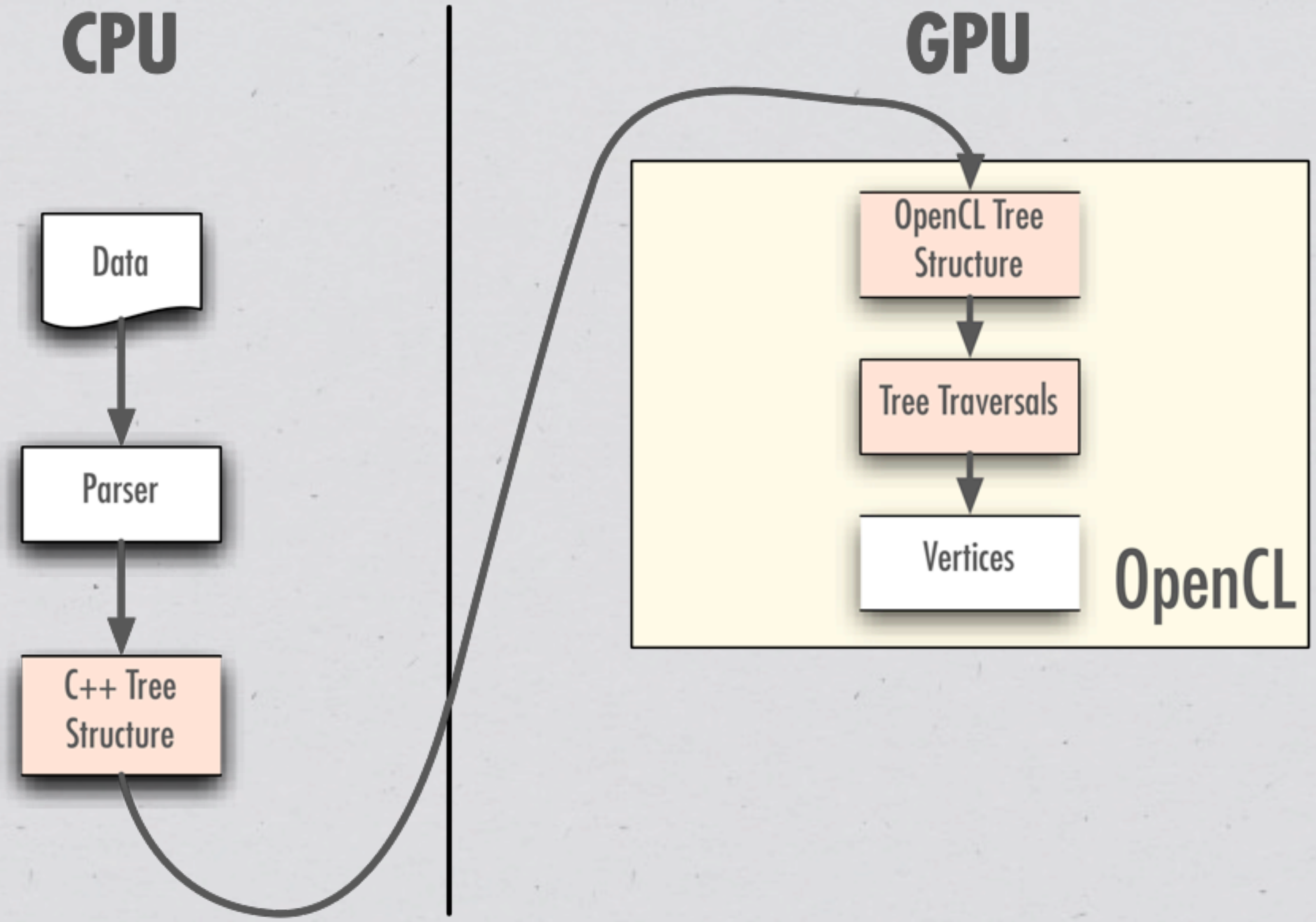
CPU

GPU

Host code reads data into tree structure



Runtime



CPU

GPU

Data

Parser

C++ Tree Structure

OpenCL Tree Structure

Tree Traversals

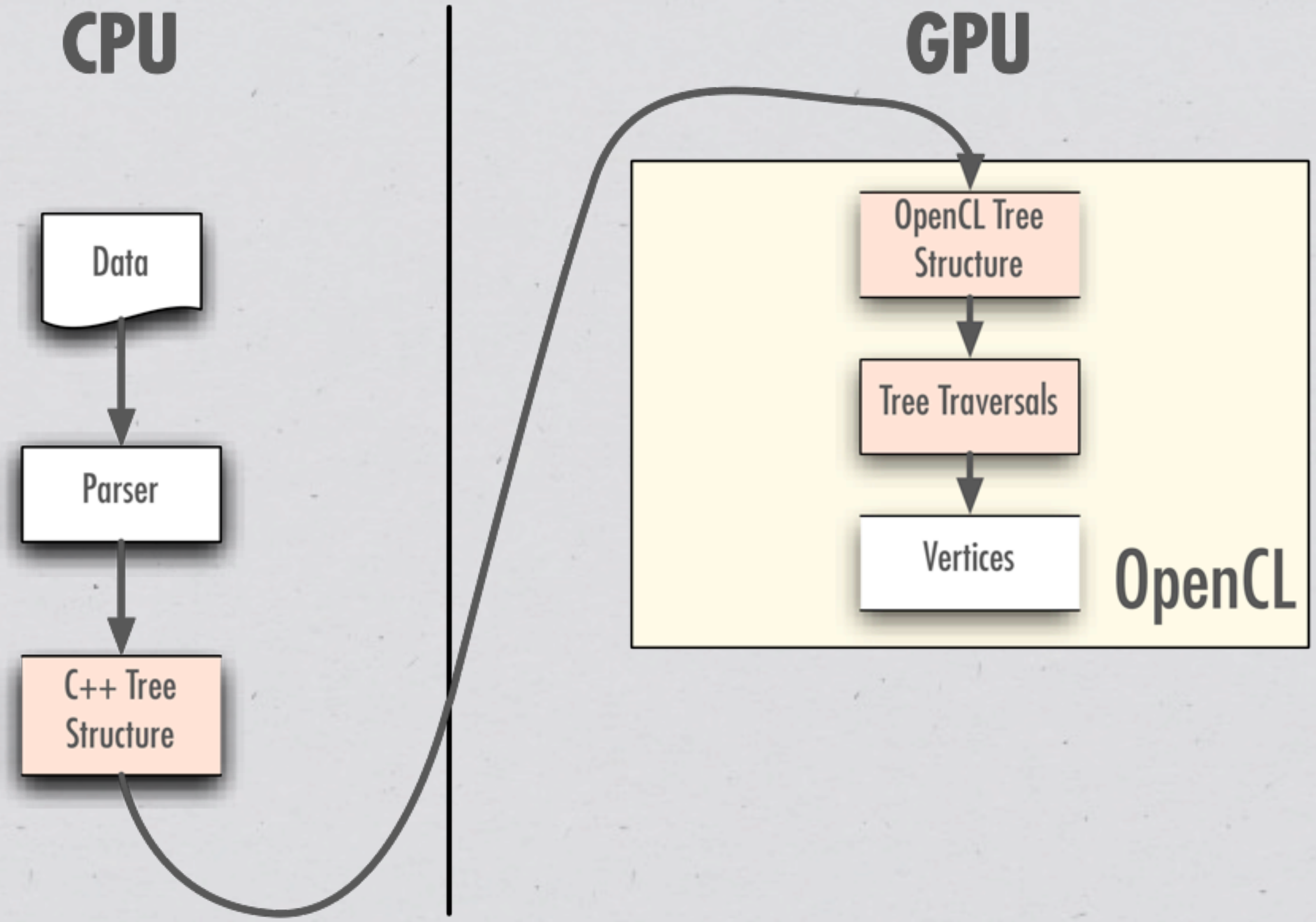
Vertices

OpenCL

Host code reads data into tree structure

Tree transferred into GPU memory

Runtime

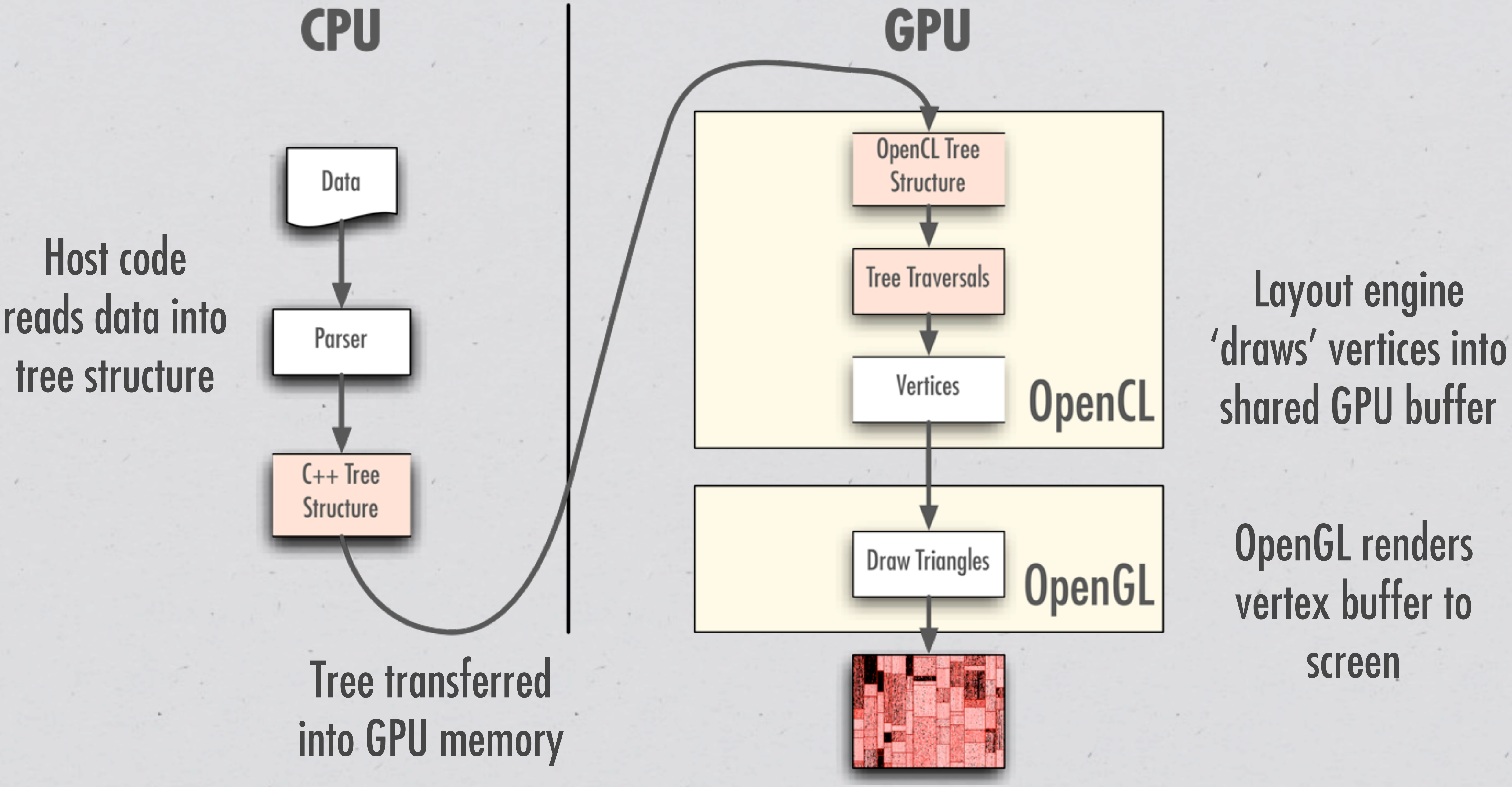


Host code reads data into tree structure

Tree transferred into GPU memory

Layout engine 'draws' vertices into shared GPU buffer

Runtime



Host code reads data into tree structure

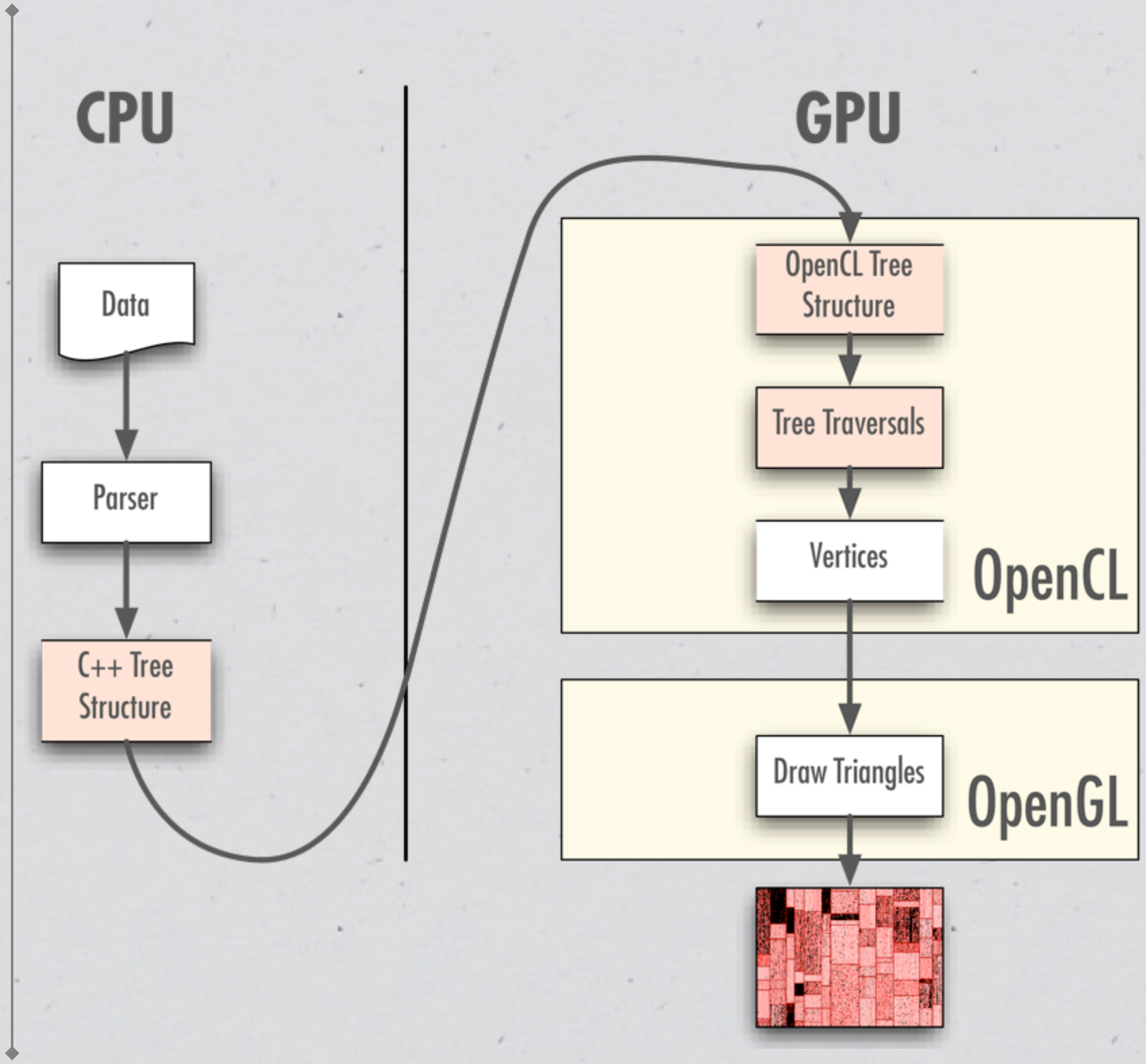
Tree transferred into GPU memory

Layout engine ‘draws’ vertices into shared GPU buffer

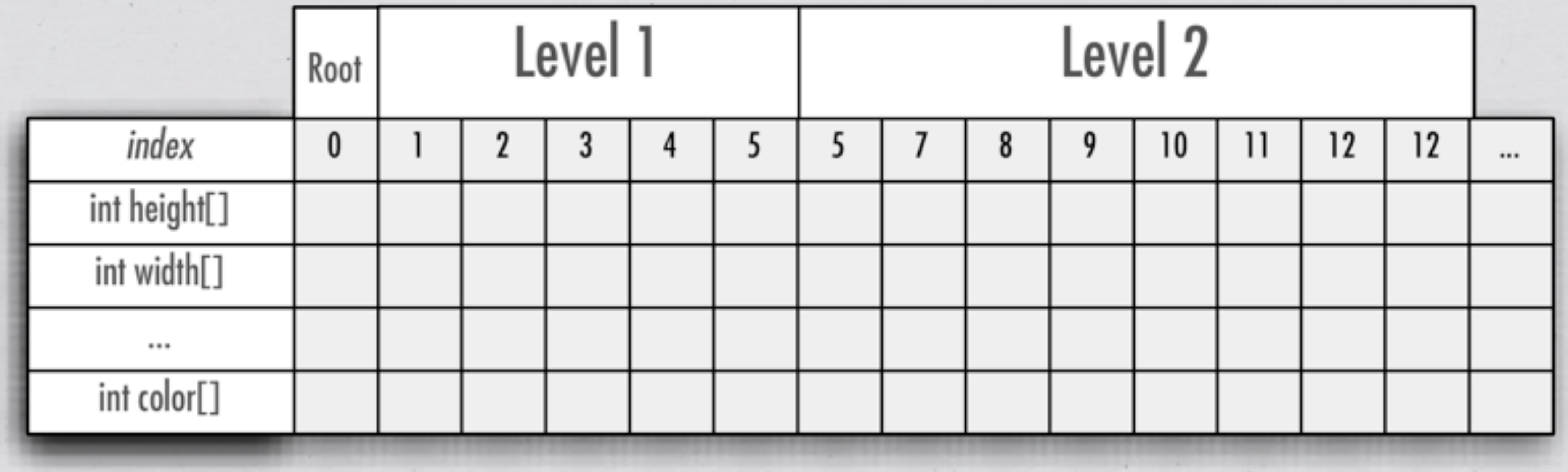
OpenGL renders vertex buffer to screen

Architecting for Speed

- Data only moved once (we have big data)
- Layout engine generated vertices — speaks language of OpenGL



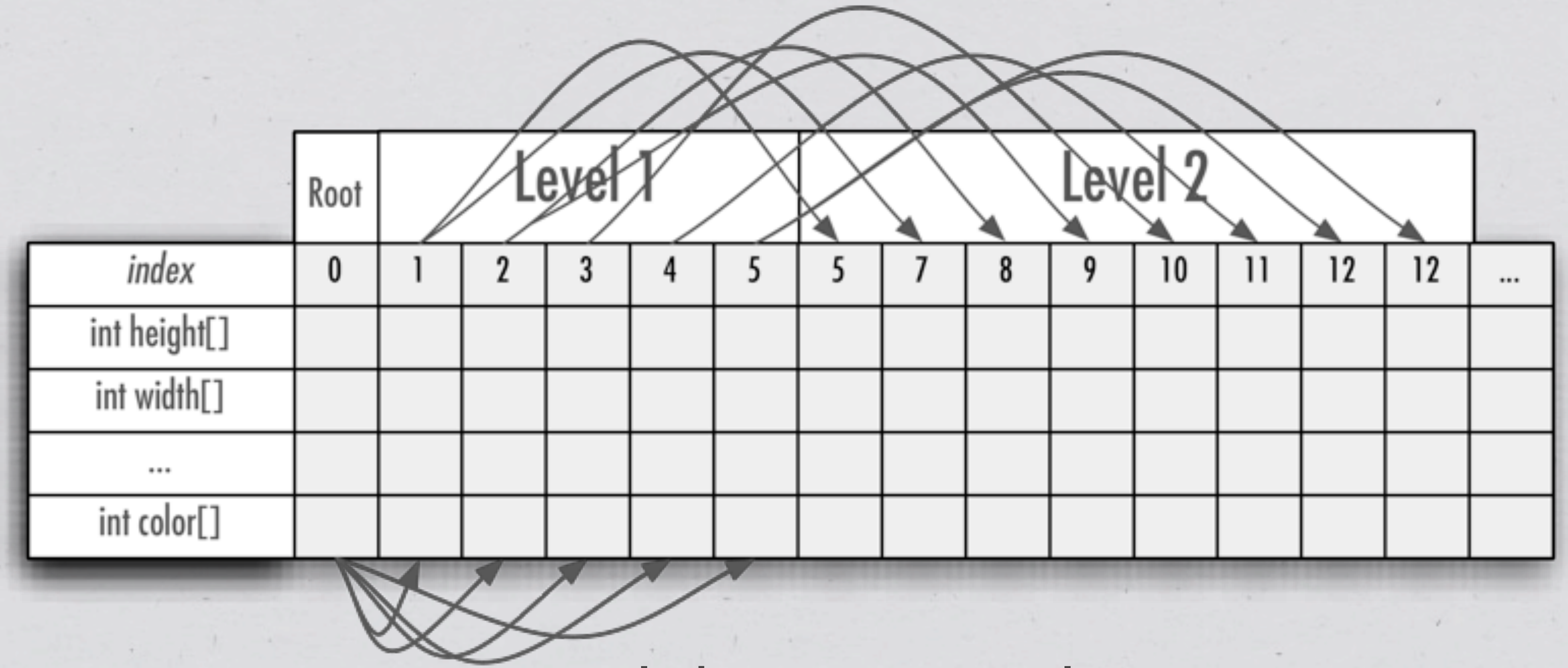
Parallel Tree Traversals



	Root	Level 1					Level 2								
index	0	1	2	3	4	5	5	7	8	9	10	11	12	12	...
int height[]															
int width[]															
...															
int color[]															

Tree laid out as structure-split arrays

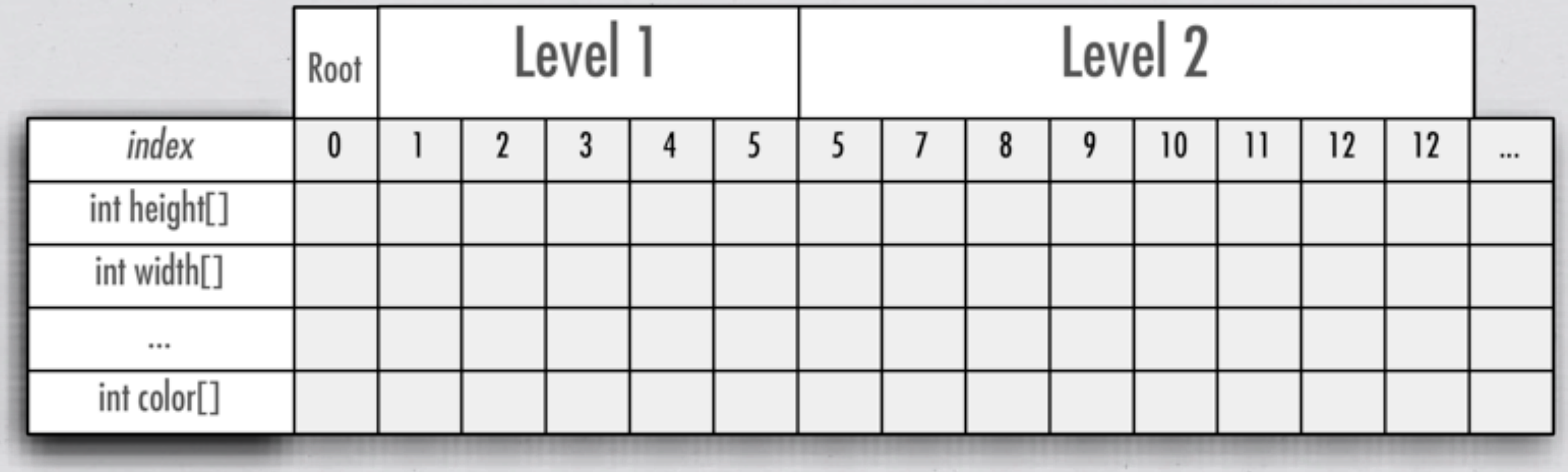
Parallel Tree Traversals



	Root	Level 1					Level 2								
index	0	1	2	3	4	5	5	7	8	9	10	11	12	12	...
int height[]															
int width[]															
...															
int color[]															

Tree laid out as structure-split arrays

Parallel Tree Traversals



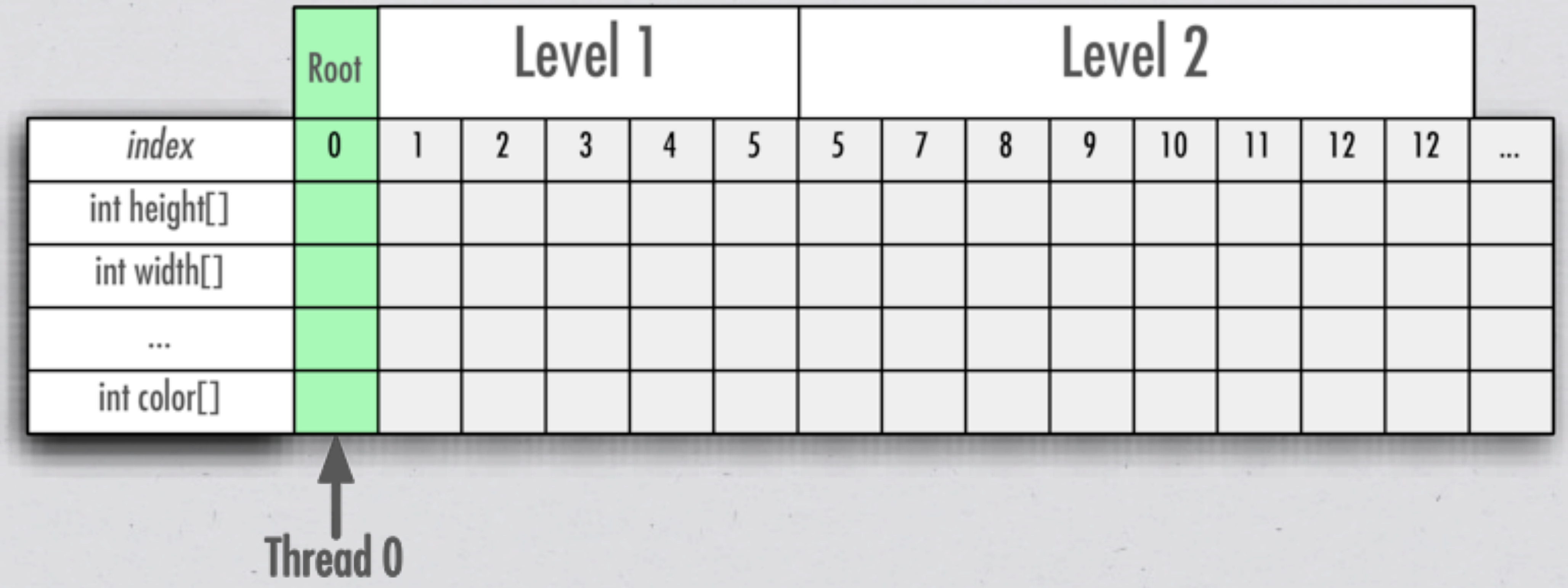
	Root	Level 1					Level 2								
index	0	1	2	3	4	5	5	7	8	9	10	11	12	12	...
int height[]															
int width[]															
...															
int color[]															

Then traversed level-by-level, synchronous

Parallel Tree Traversals

Level 0 (root)

In top-down traversal, start with root

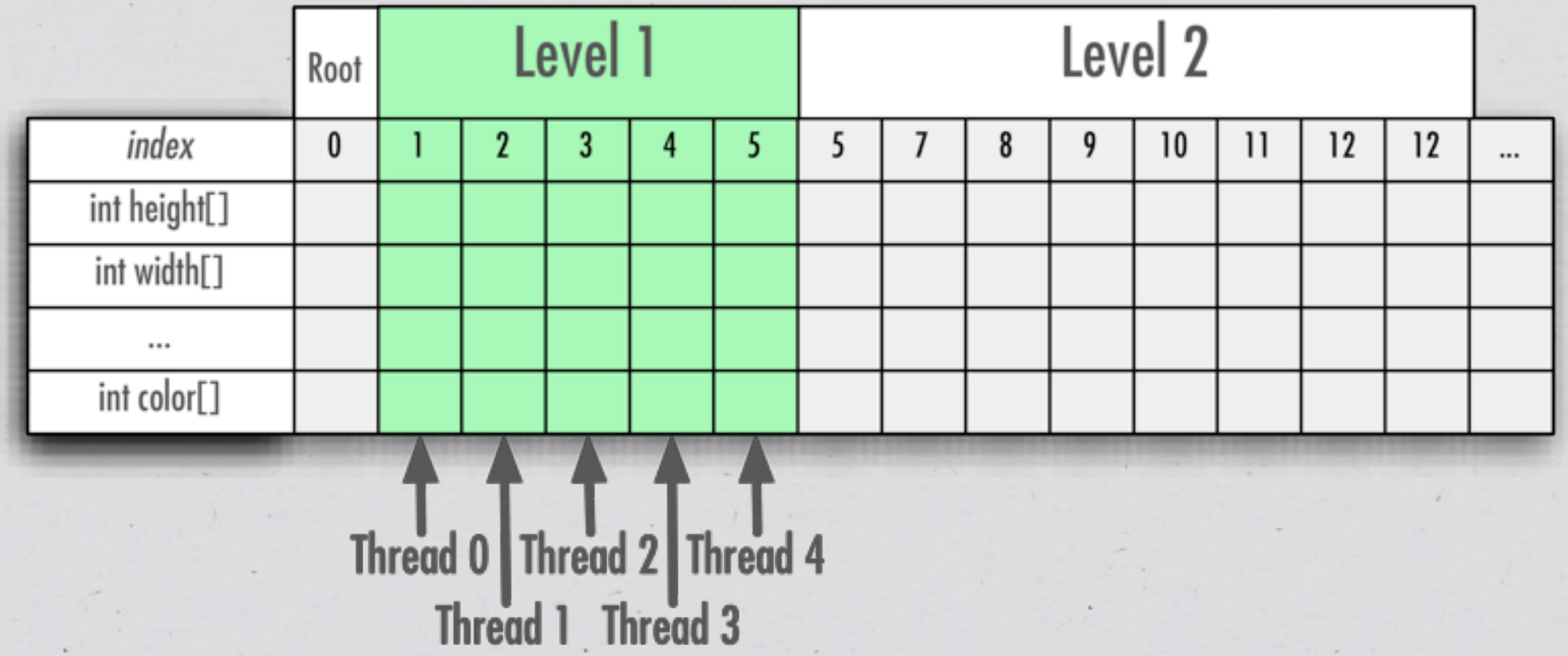


	Root	Level 1					Level 2								
index	0	1	2	3	4	5	5	7	8	9	10	11	12	12	...
int height[]															
int width[]															
...															
int color[]															

Parallel Tree Traversals

Level 1

Then process each subsequent level serially

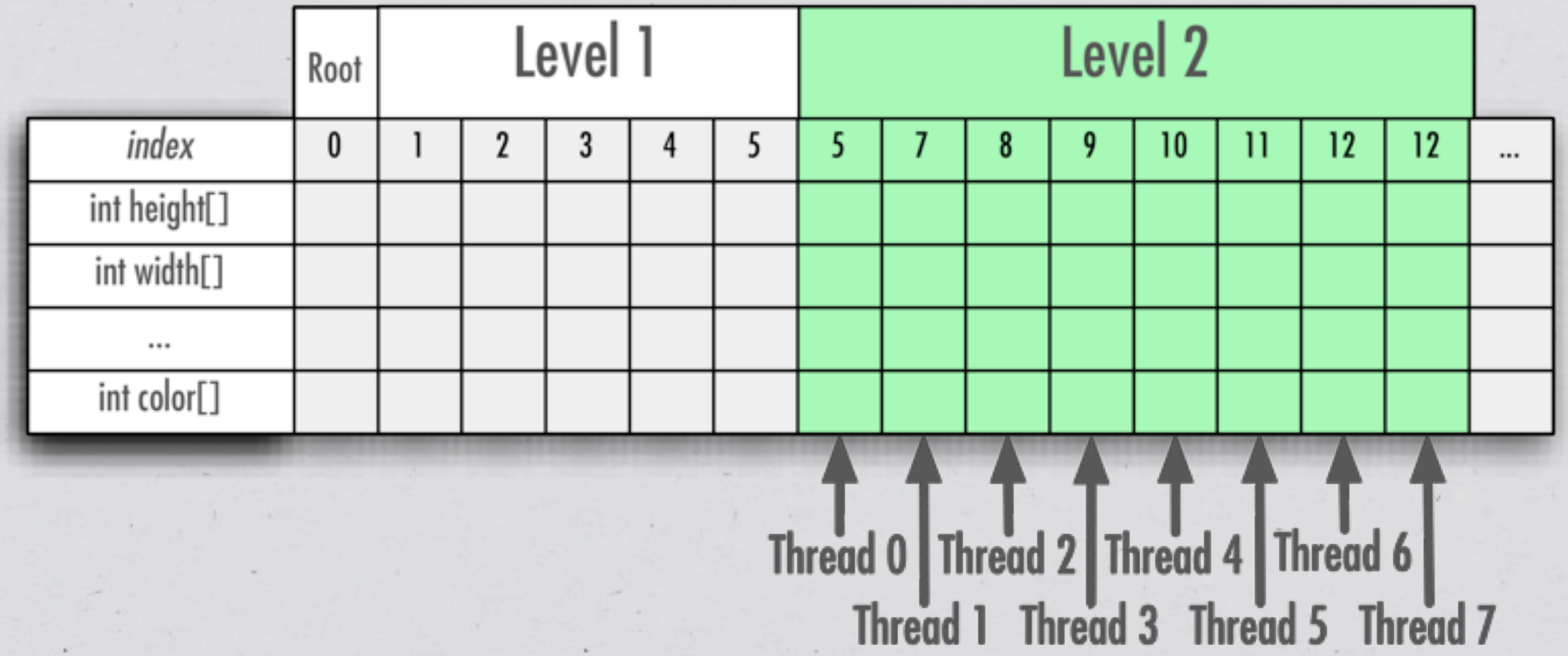


	Root	Level 1					Level 2								
index	0	1	2	3	4	5	5	7	8	9	10	11	12	12	...
int height[]															
int width[]															
...															
int color[]															

Parallel Tree Traversals

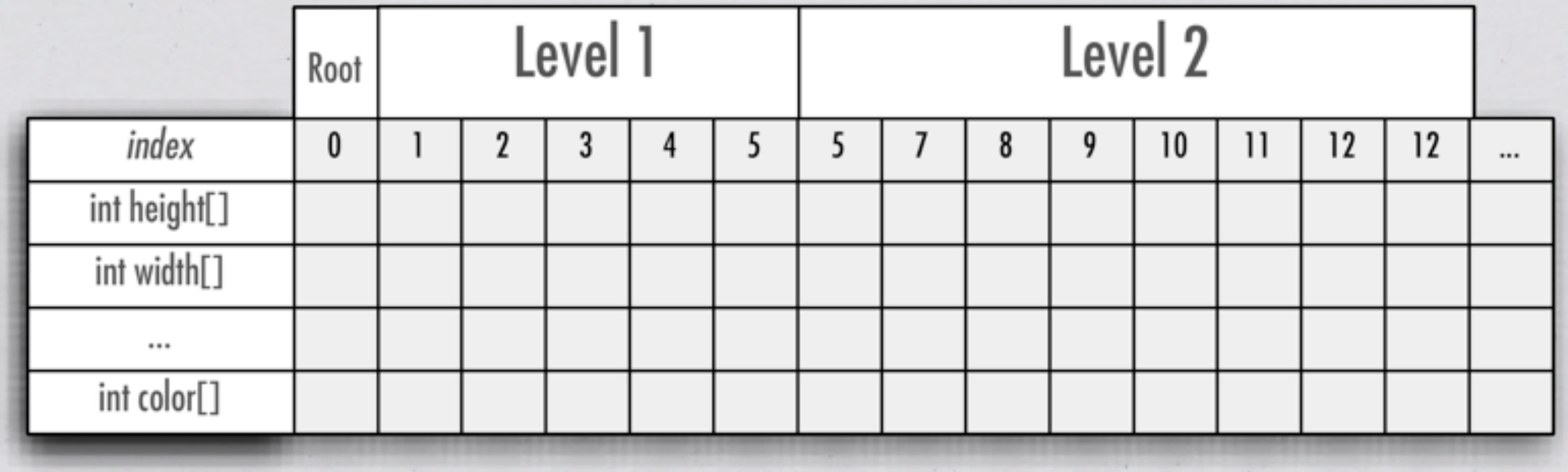
Level 2

The nodes within a level are processed in parallel



	Root	Level 1					Level 2								
index	0	1	2	3	4	5	5	7	8	9	10	11	12	12	...
int height[]															
int width[]															
...															
int color[]															

Parallel Tree Traversals



	Root	Level 1					Level 2								
index	0	1	2	3	4	5	5	7	8	9	10	11	12	12	...
int height[]															
int width[]															
...															
int color[]															

DEMO

2011 Russian Legislative Election

JavaScript Comparison: Treemap

	27 FPS
JavaScript 2.3GHz Core i7	500
Superconductor GeForce GT 650M	100,000

Weak Scaling on GPU

300 line declarative spec synthesized into 5 efficient tree traversals

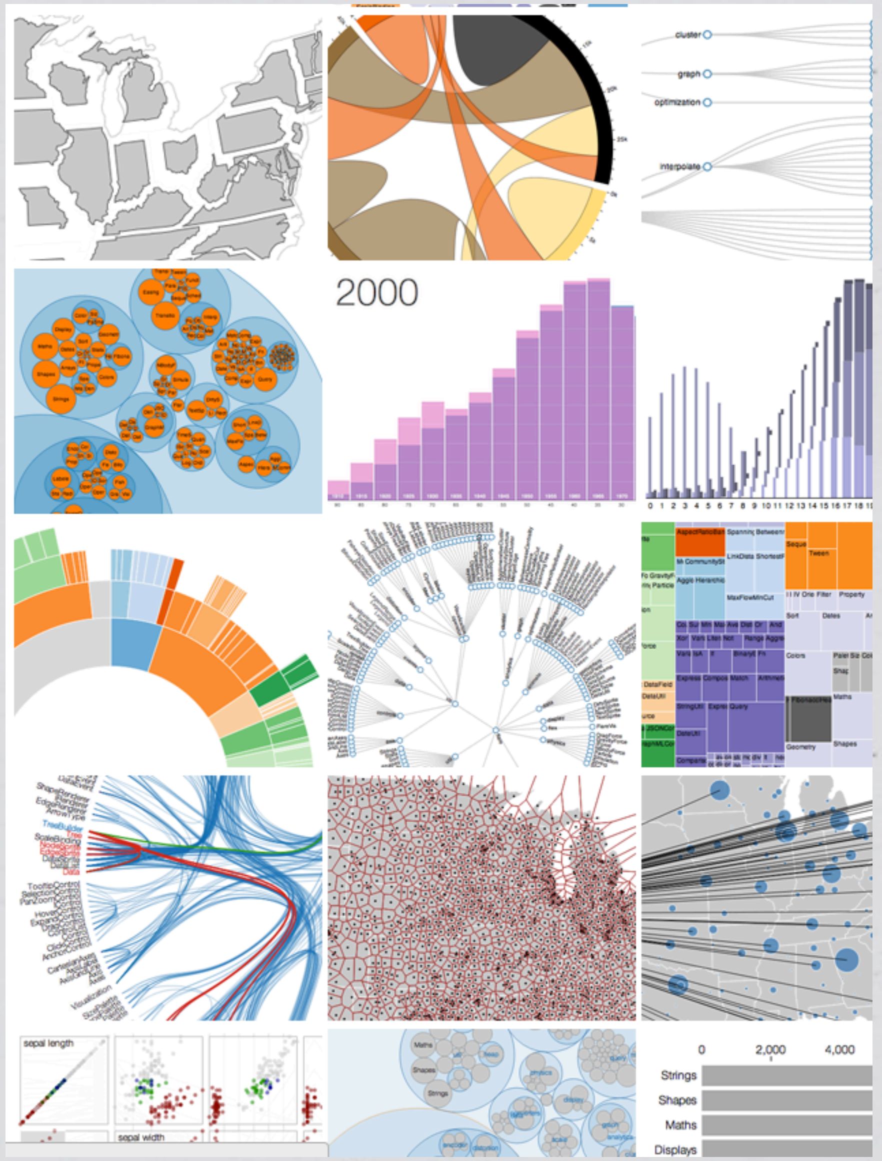
Nodes	Speed
10,000	26 FPS
100,000	26 FPS
1,000,000	4.5 FPS

Early Results

WHAT’S NEXT?

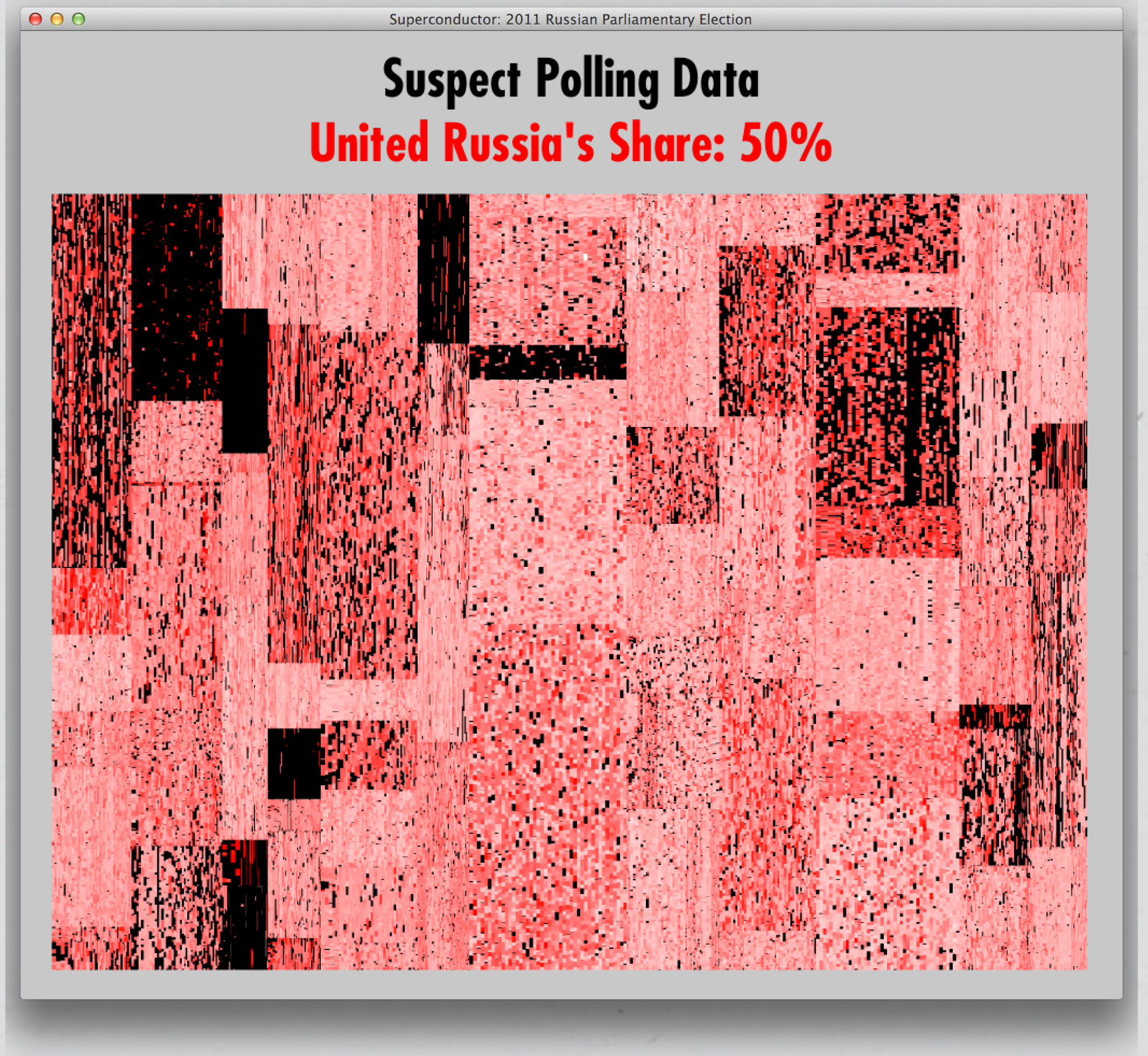
Future Extensions

- Beyond tree traversals: graphs
- Beyond OpenCL: WebCL
- Data binding & mutation
- **Open Source: 2013?**



Summary

- Declarative visualization language
- GPU: big data & small devices
- Result:
    - 100,000 node interactive animation
    - 200x speedup vs. other high-level language (JS)



Superconductor: 2011 Russian Parliamentary Election

Suspect Polling Data

United Russia's Share: 50%

SUPERCONDUCTOR

Coming Soon