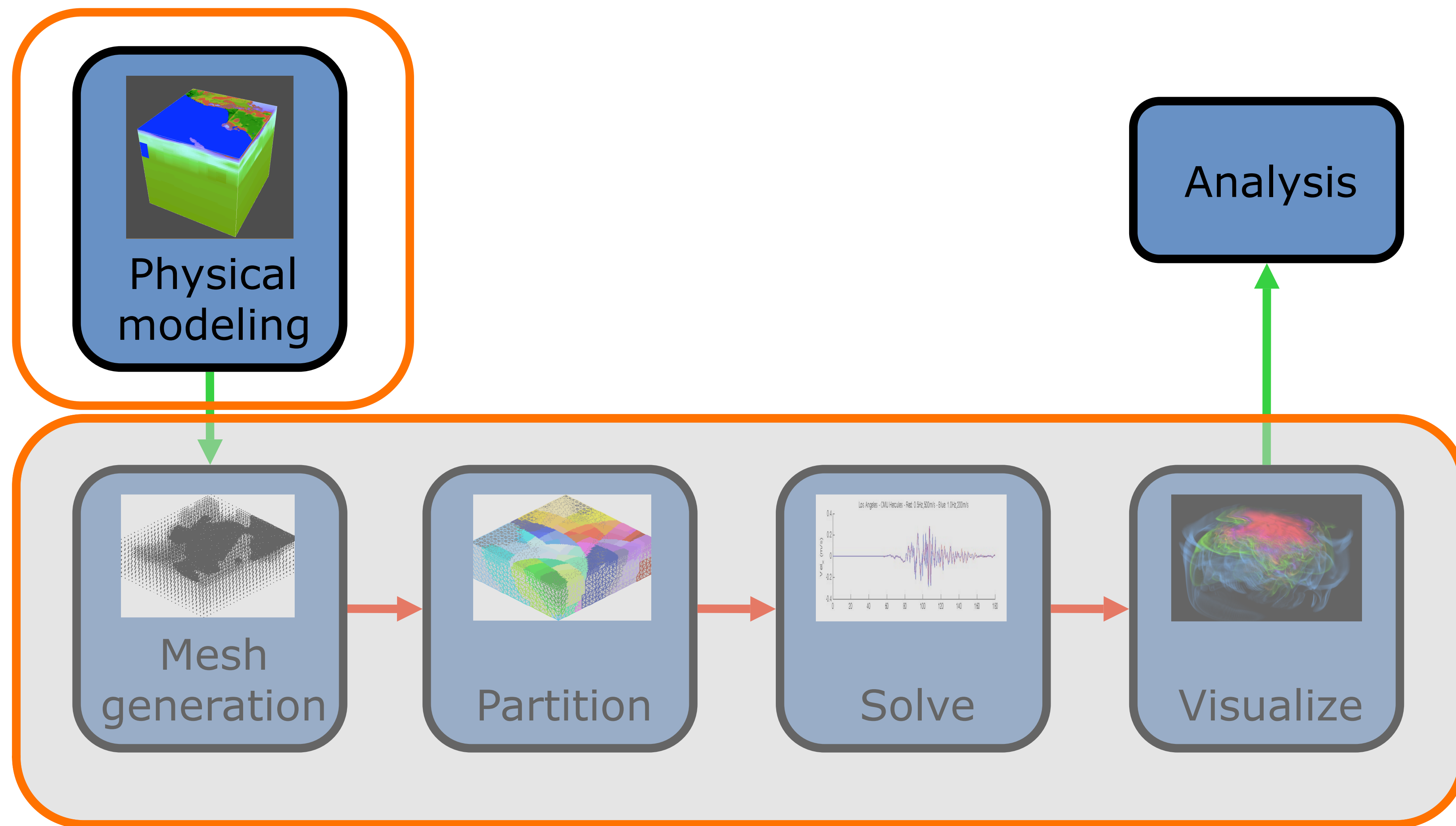
Building ground models of Southern California

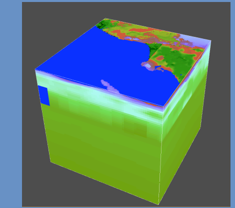
Steve Schlosser
Michael Ryan
Dave O'Hallaron
Intel Research Pittsburgh

Julio López
Ricardo Taborda
Jacobo Bielak
Carnegie Mellon University

Ground motion modeling 101



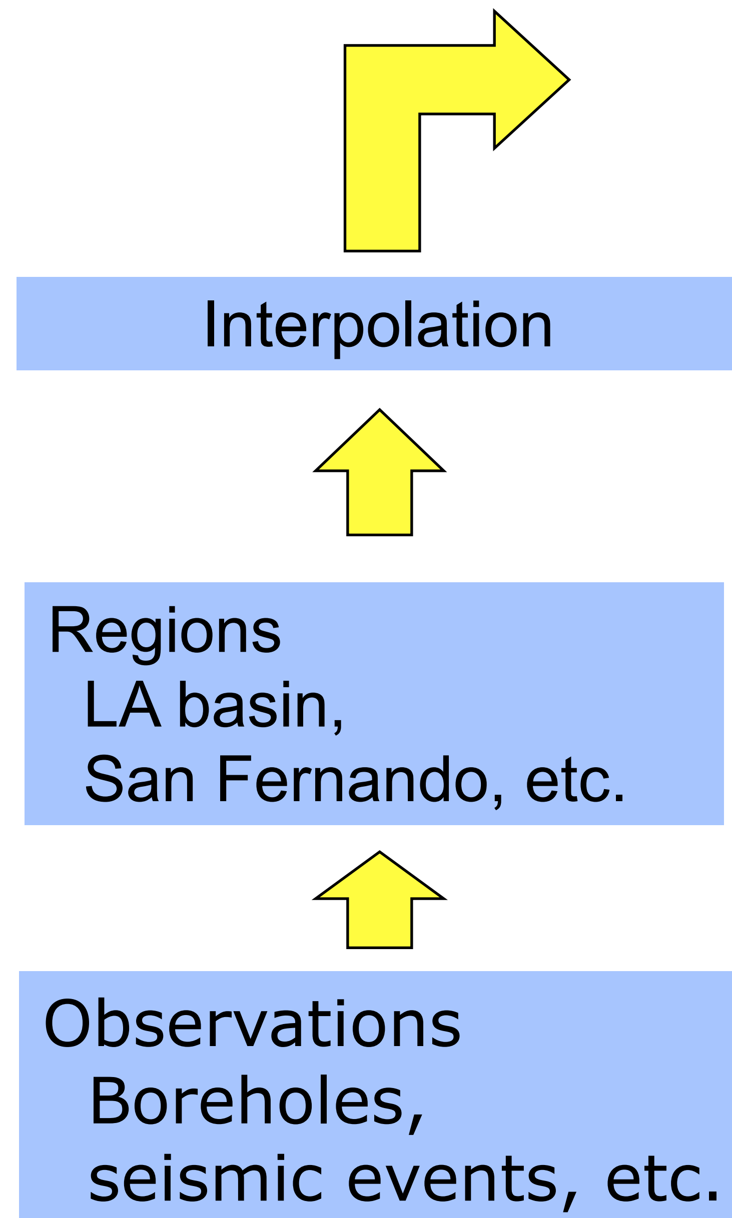
Benefits of materialization



Physical modeling

- Query cost
  - Materialized octrees are cheaper to query by 10-100x
- Ease of use
  - Eliminates days or weeks of dataset build time
  - No need to integrate user-level code into SC apps
  - Avoids quirks of existing model programs
- Standardization
  - Sharing among many scientists improves repeatability, validation

CVM Programs



CVM Program: Community Velocity Model

Input: lat, long, depth tuples

Output: ground characteristics at that location

Two public implementations

- SCEC (Fortran)
- Harvard (C)

Basic building block

SCEC

Image credit: Amit Chourasia, Visualization Services, SDSC

100 km deep

Goal:
  Sample entire region at 10m resolution

$6x10^4$ x $3x10^4$ x $1x10^4$ = $18x10^{12}$ sample points!

~1 PB of data uncompressed

Approach:
  Reduce early and reduce often

Map/Reduce implementation

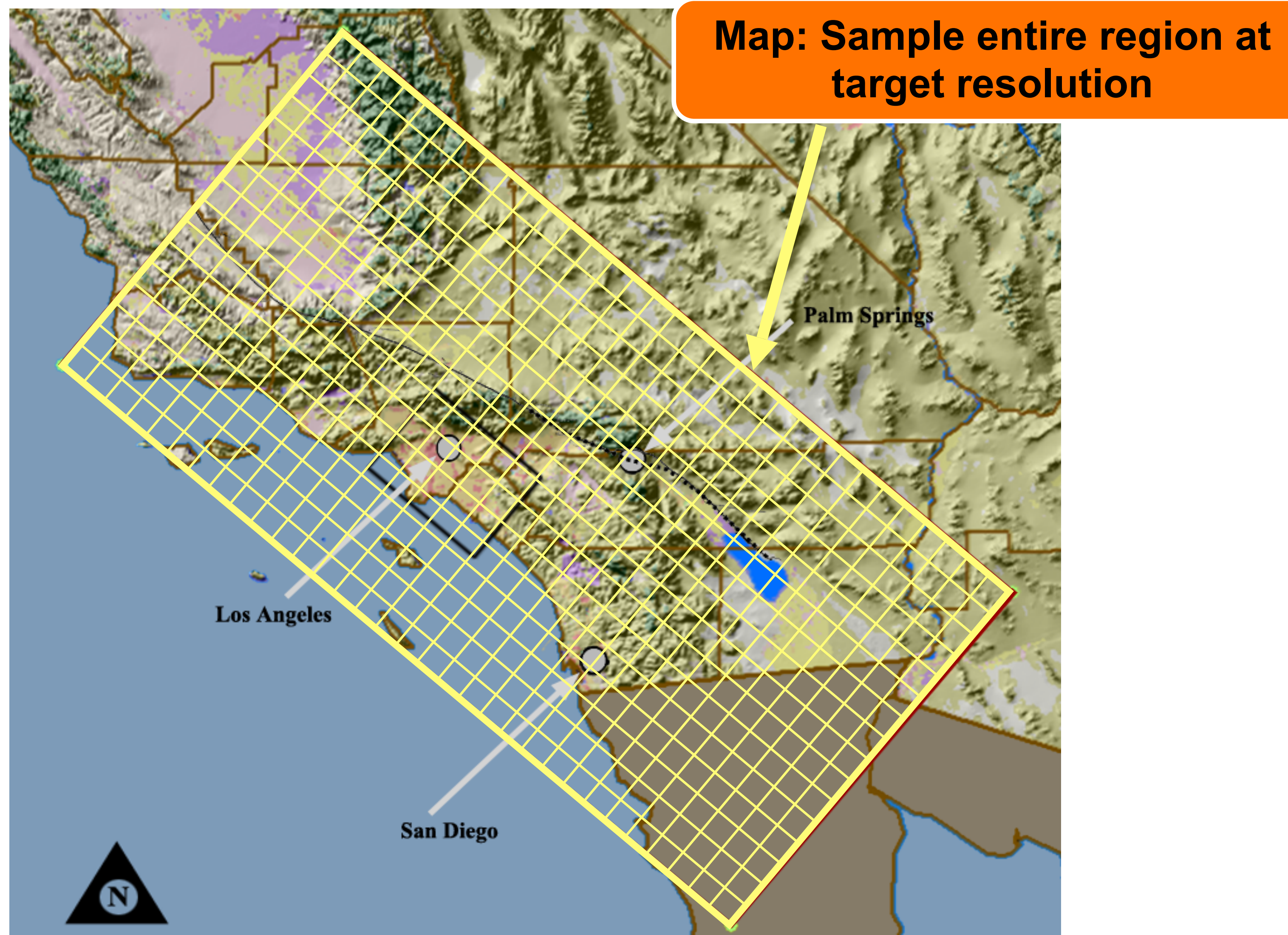


Image credit: Amit Chourasia, Visualization Services, SDSC

Map/Reduce implementation

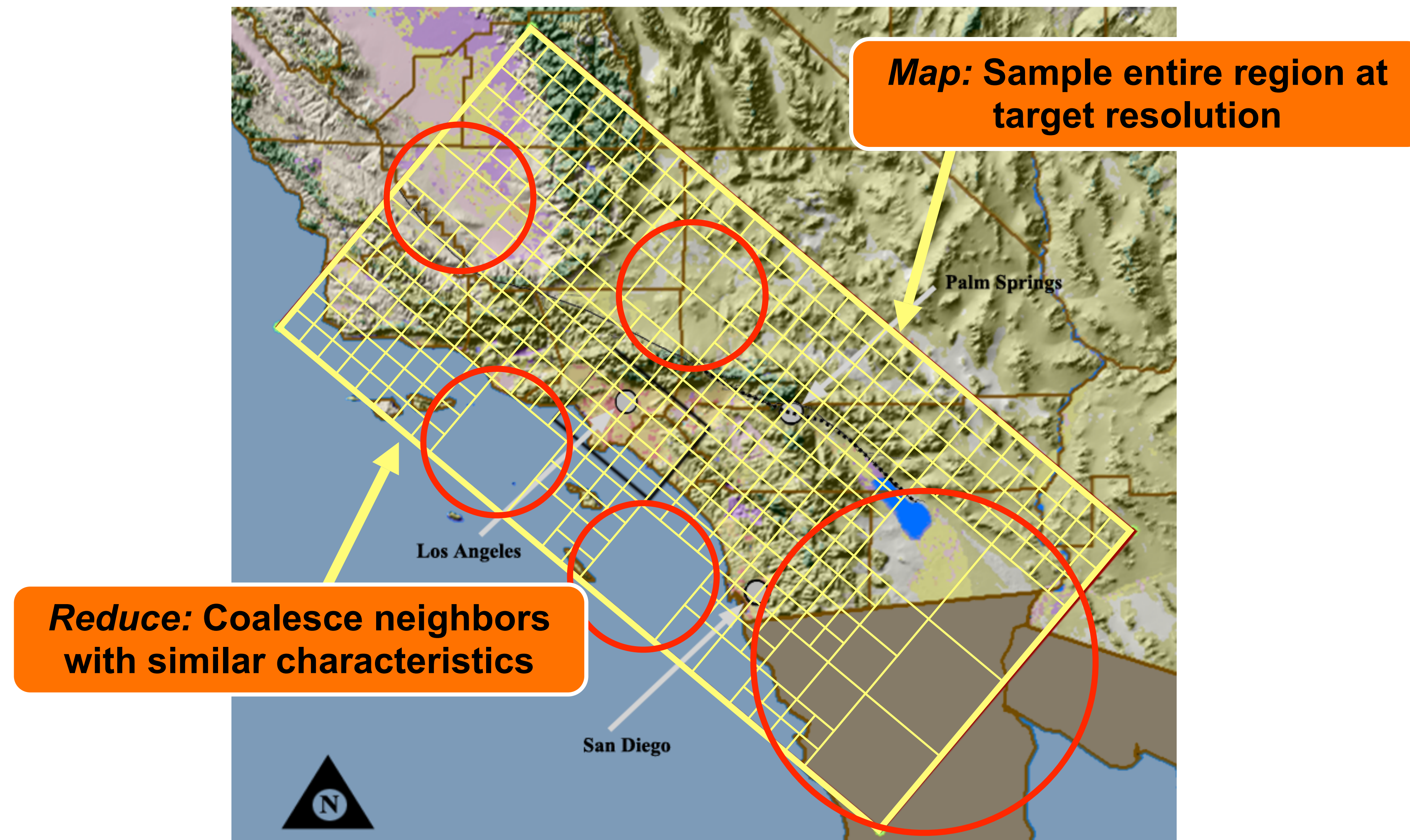
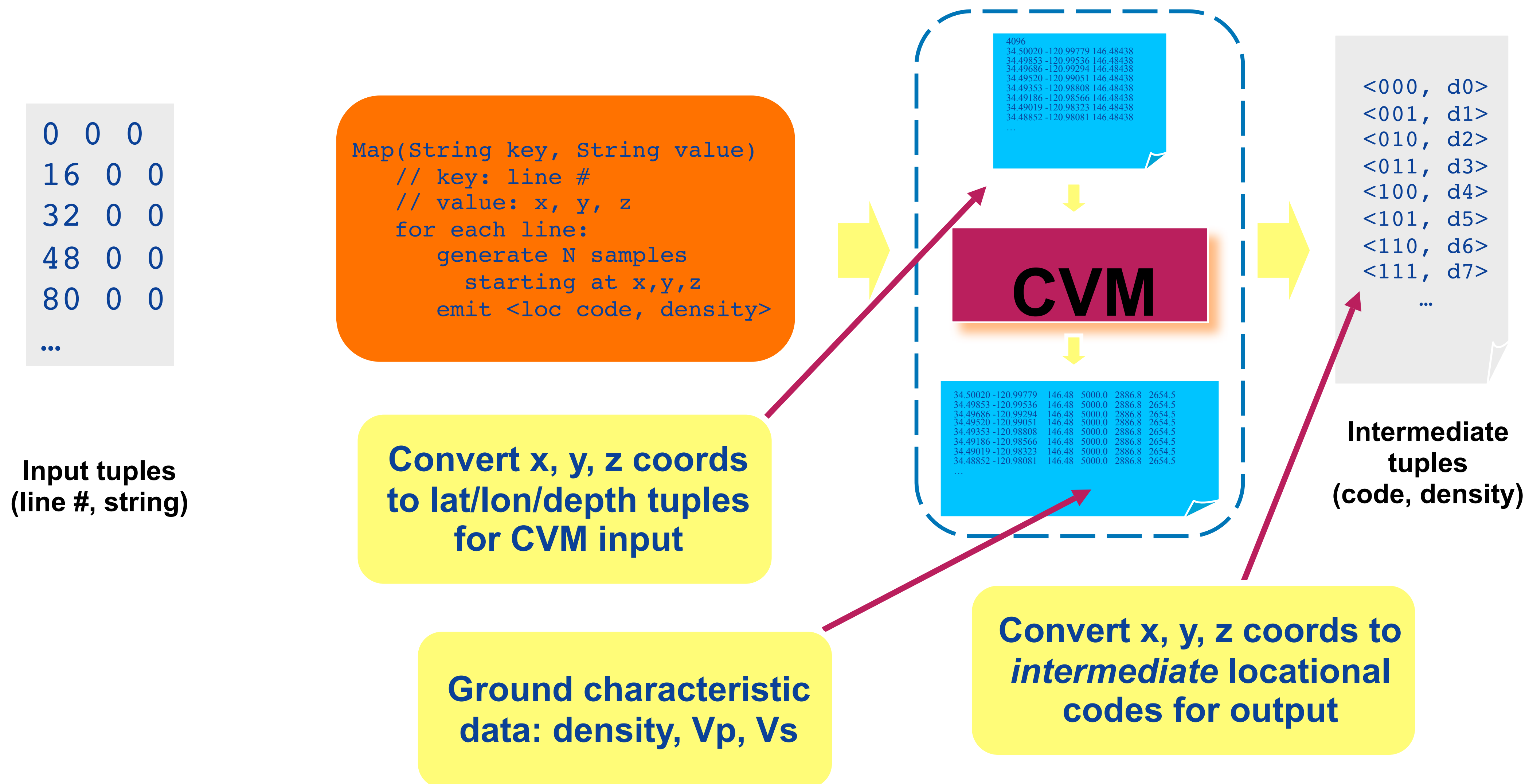


Image credit: Amit Chourasia, Visualization Services, SDSC

Map implementation



Intermediate key manipulation

<000, d0>
<001, d1>
<010, d2>
<011, d3>
<100, d4>
<101, d5>
<110, d6>
<111, d7>
…

Intermediate tuples (code, density)

Clear 3 low-order bits per octree level

<000, 000 d0>
<000, 001 d1>
<000, 010 d2>
<000, 011 d3>
<000, 100 d4>
<000, 101 d5>
<000, 110 d6>
<000, 111 d7>
…

Manipulated tuples (code, density)

Naturally gathers neighboring tuples together for Reduce

Reduce implementation

Key = 000

```
<000, 000 d0>
<000, 001 d1>
<000, 010 d2>
<000, 011 d3>
<000, 100 d4>
<000, 101 d5>
<000, 110 d6>
<000, 111 d7>
…
```

Intermediate tuples (code, density)

```
Reduce(String k, Iterator value)
  // k: locational code
  // value: sample data
  vector samples = ();
  foreach v in value
    samples.push(v);
  if (tryCoalesce(samples))
    emit <coalesce(samples)>
  else
    emit <samples>
```

```
<000 d0>
<001 d1>
<010 d2>
<011 d3>
<100 d4>
<101 d5>
<110 d6>
<111 d7>
…
```

Are they neighbors?
Yes.

Are densities equal?
No.

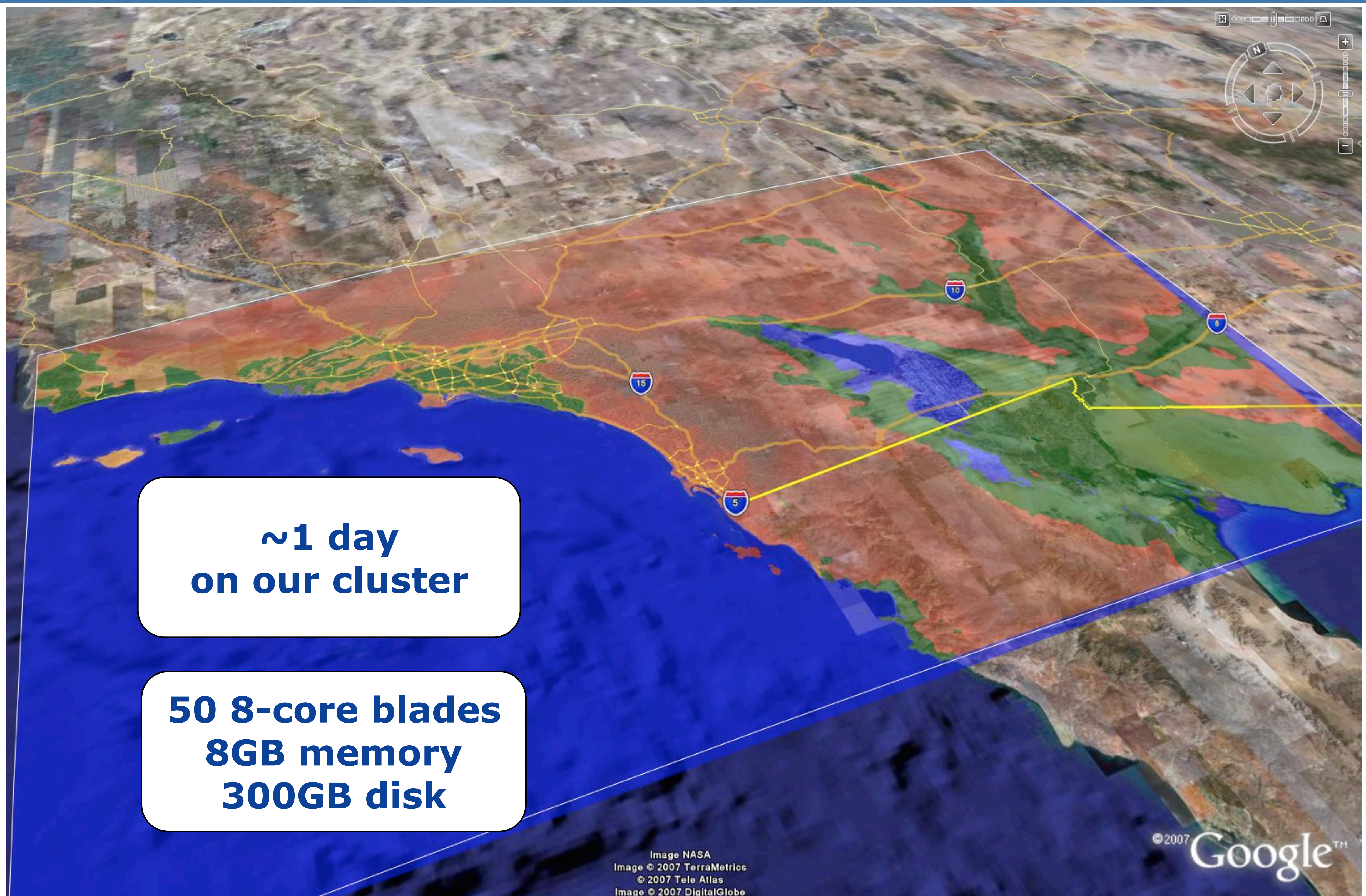
Just emit

```
<000, d0>
<001, d1>
<010, d2>
<011, d3>
<100, d4>
<101, d5>
<110, d6>
<111, d7>
…
```

Output (code, density)

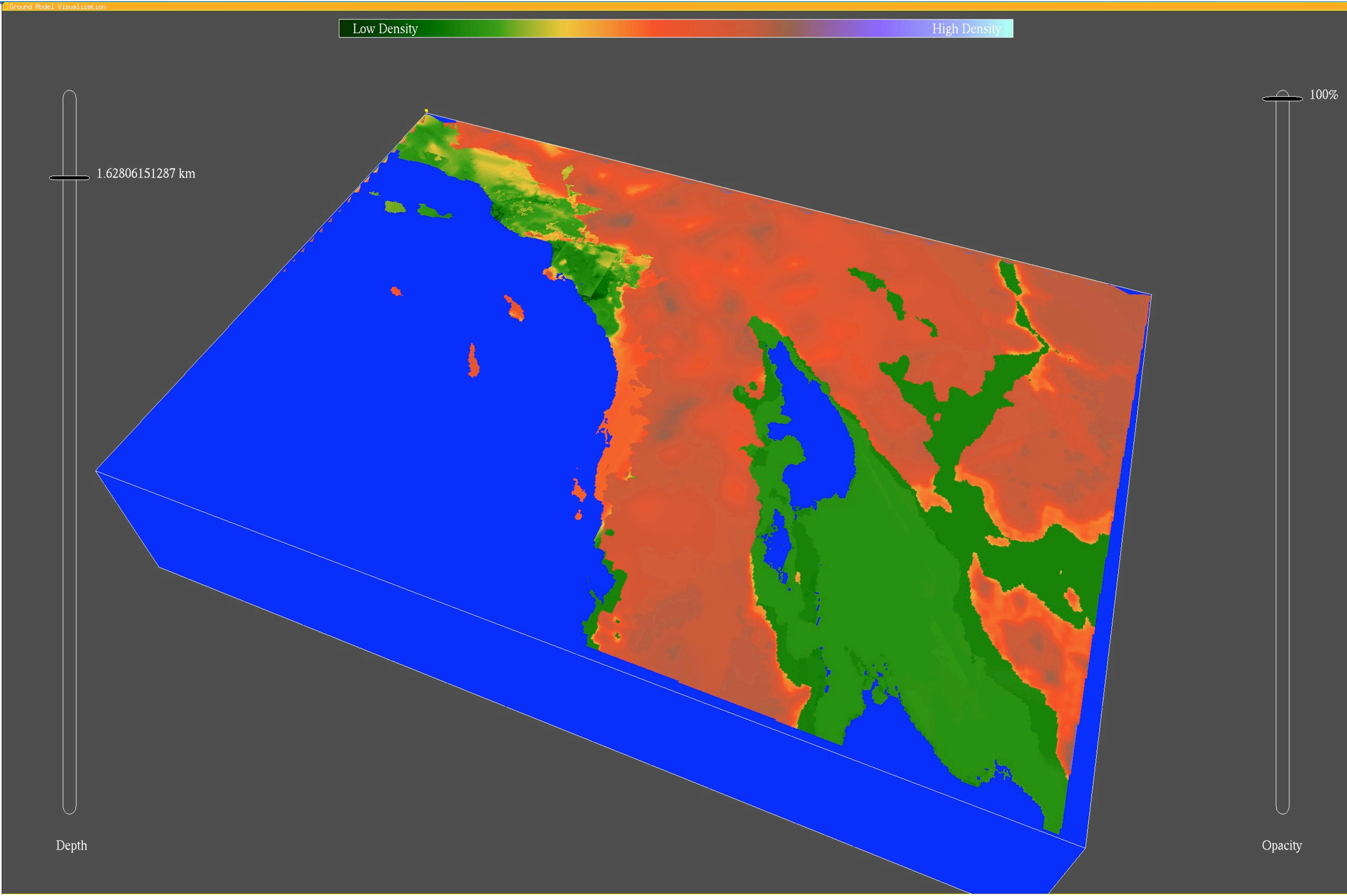
Run successive Reduces until data cannot be further coalesced

Harvard

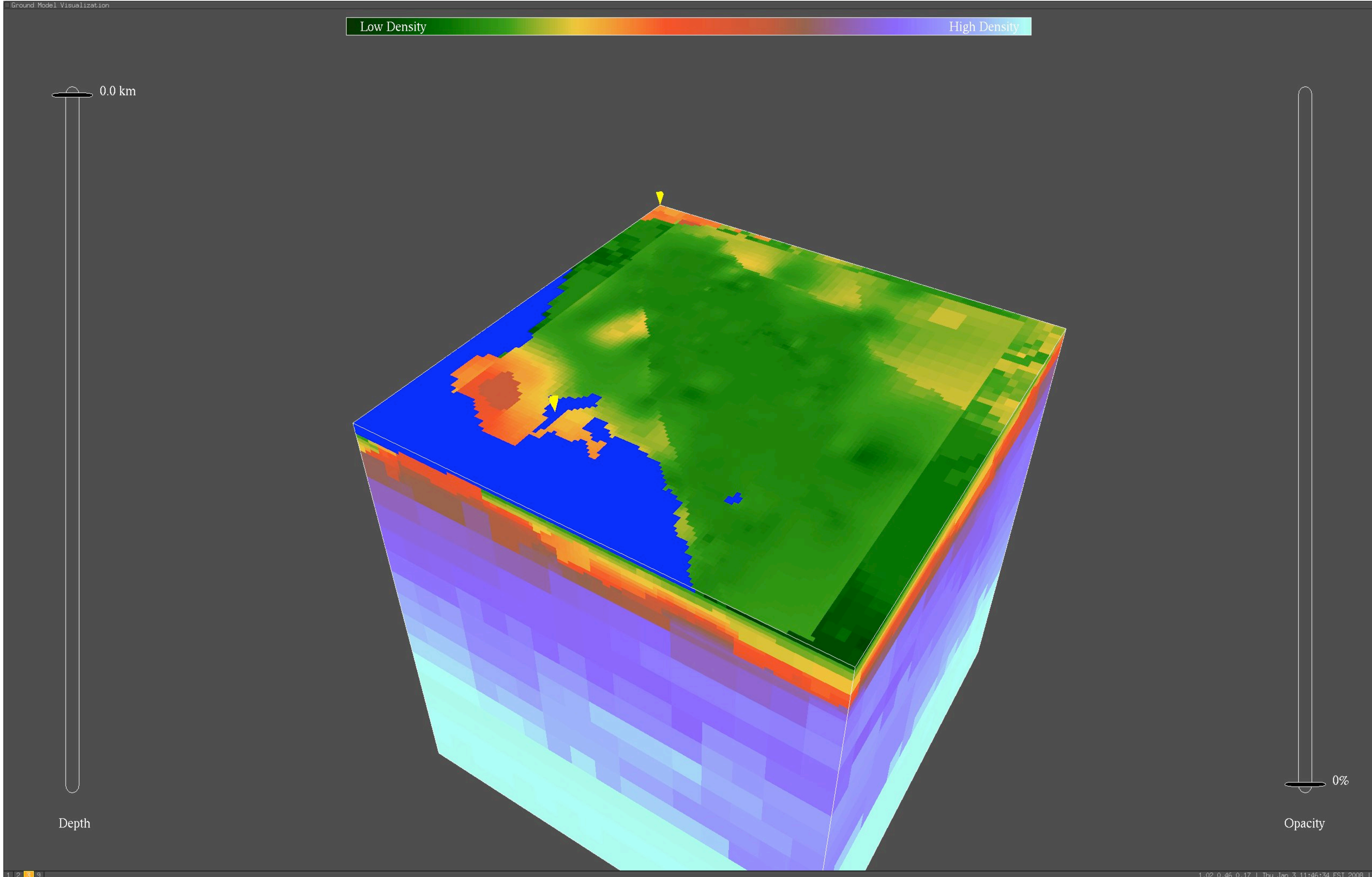


SCEC

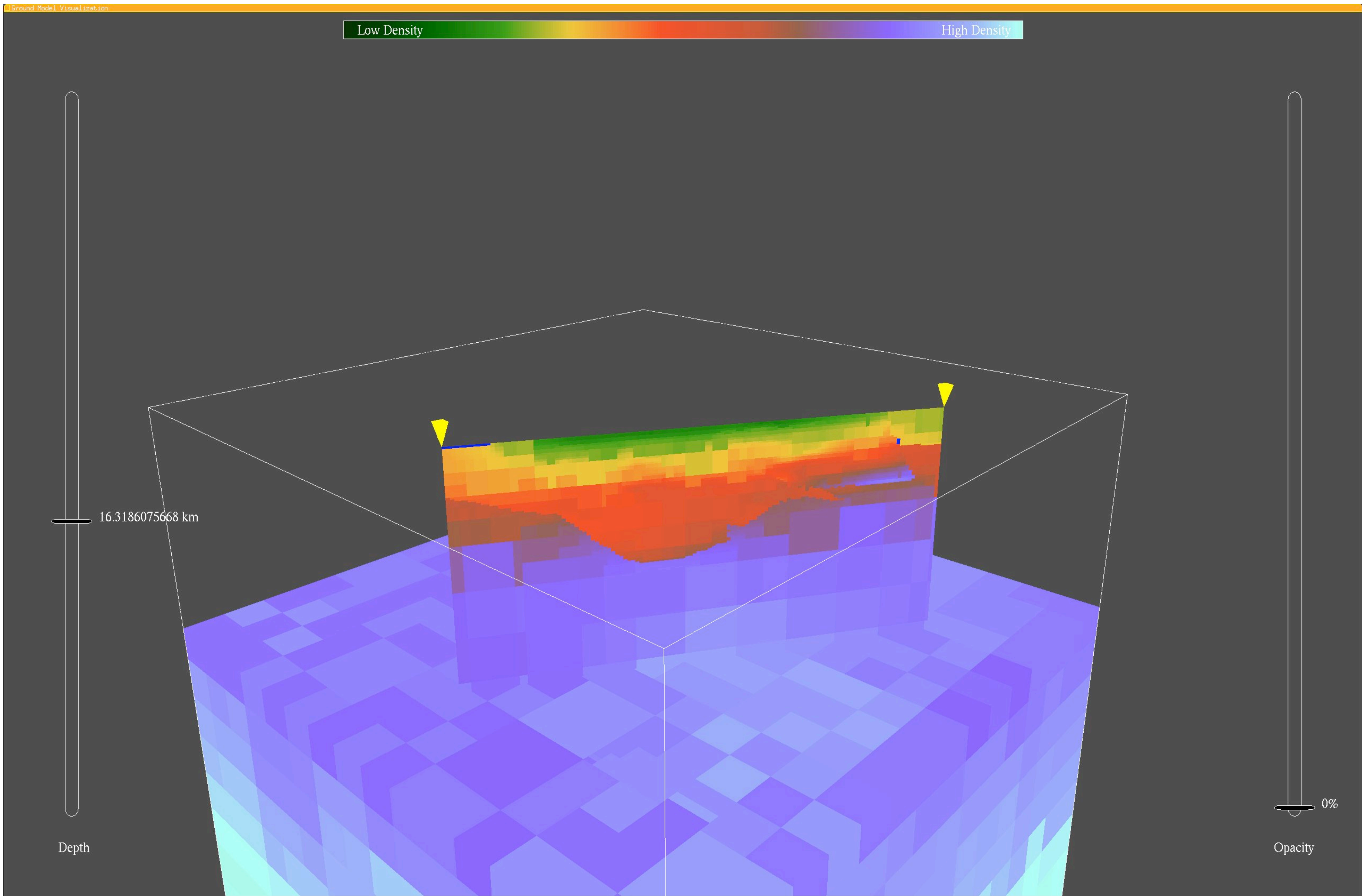
Harvard



Harvard



Harvard

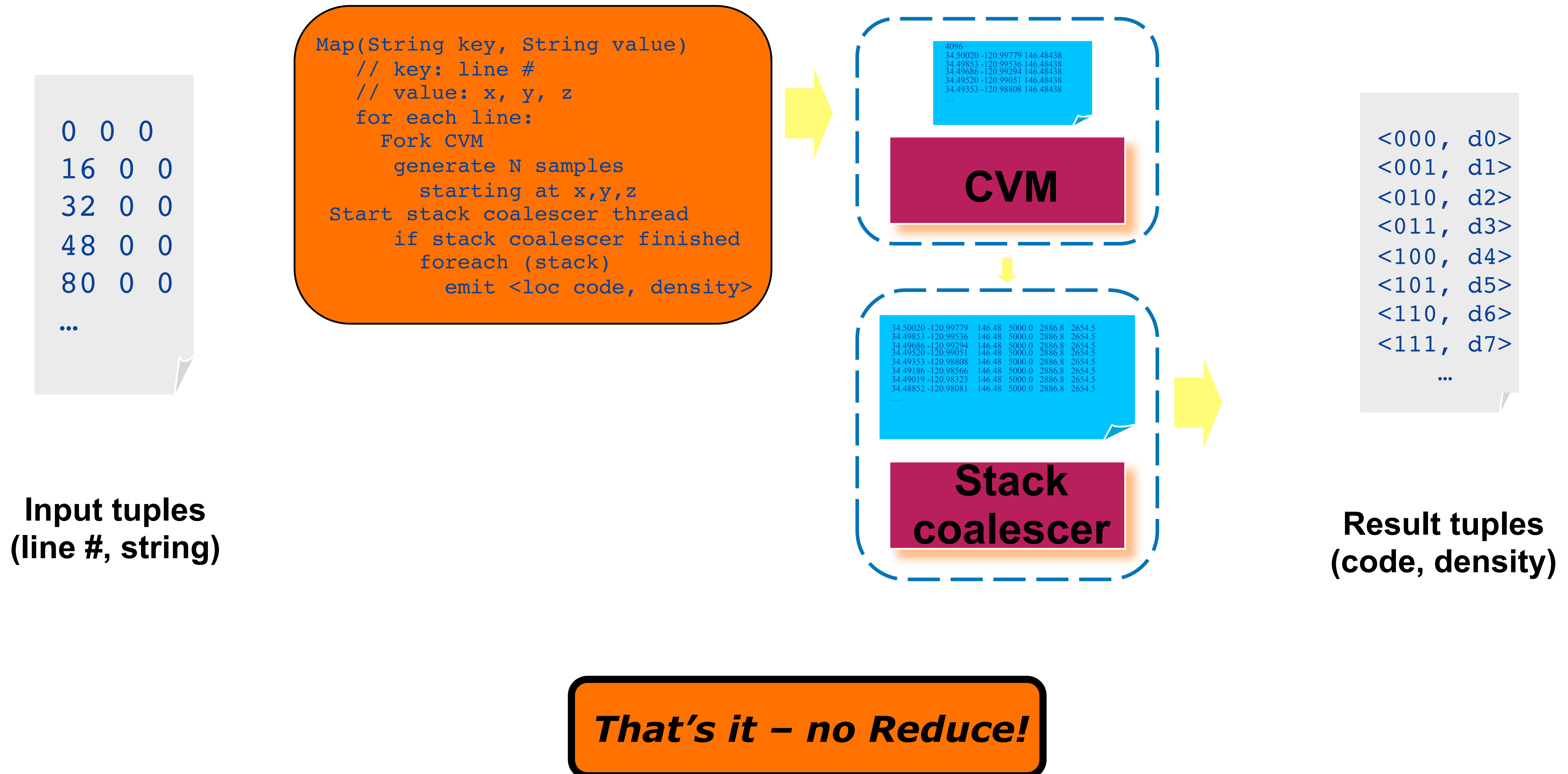


"What if we did this in C?"

- Assign each region to a Maui/Torque task
- Generate samples in ascending loc code order
- Use stack-based coalescing algorithm
- Avoids all task/task communication

Harvard – a few hours
SCEC – ~1 day

Hadoop implementation



Conclusions

- Used Hadoop to build a ground model generator
- Hadoop implementation runs in O(days)
- Stack-based C and Hadoop versions run in several hours
- Cost of distributed group-by are not necessary for this app
- What is the lesson?

[illegible]
Shuffle/Sort/Reduce/OutputFormat

Map/Reduce hides a lot of complexity