Census: Location-Aware Membership Management for Large-Scale Distributed Systems

James Cowling **Dan R. K. Ports** Barbara Liskov
Raluca Ada Popa Abhijeet Gaikwad*

MIT CSAIL *École Centrale Paris

Motivation

Large-scale distributed systems becoming more common
    multiple datacenters, cloud computing, etc.

Reconfigurable distributed services adapt as nodes join, leave, or fail

A *membership service* that tracks changes in system membership can simplify system design

Census

A platform for building *large-scale, distributed applications*

Two main components:
- Membership service
- Multicast communication mechanism

Designed to work in the wide-area
- Locality-aware; fault tolerant

Membership Service

Time divided into sequential, fixed-duration *epochs*

Each epoch has a *membership view*:
List of nodes (ID, IP address, location, etc.)

Consistency property:
every node sees the *same* membership view
for a particular epoch
➡ can simplify protocol design (e.g. partitioning storage)

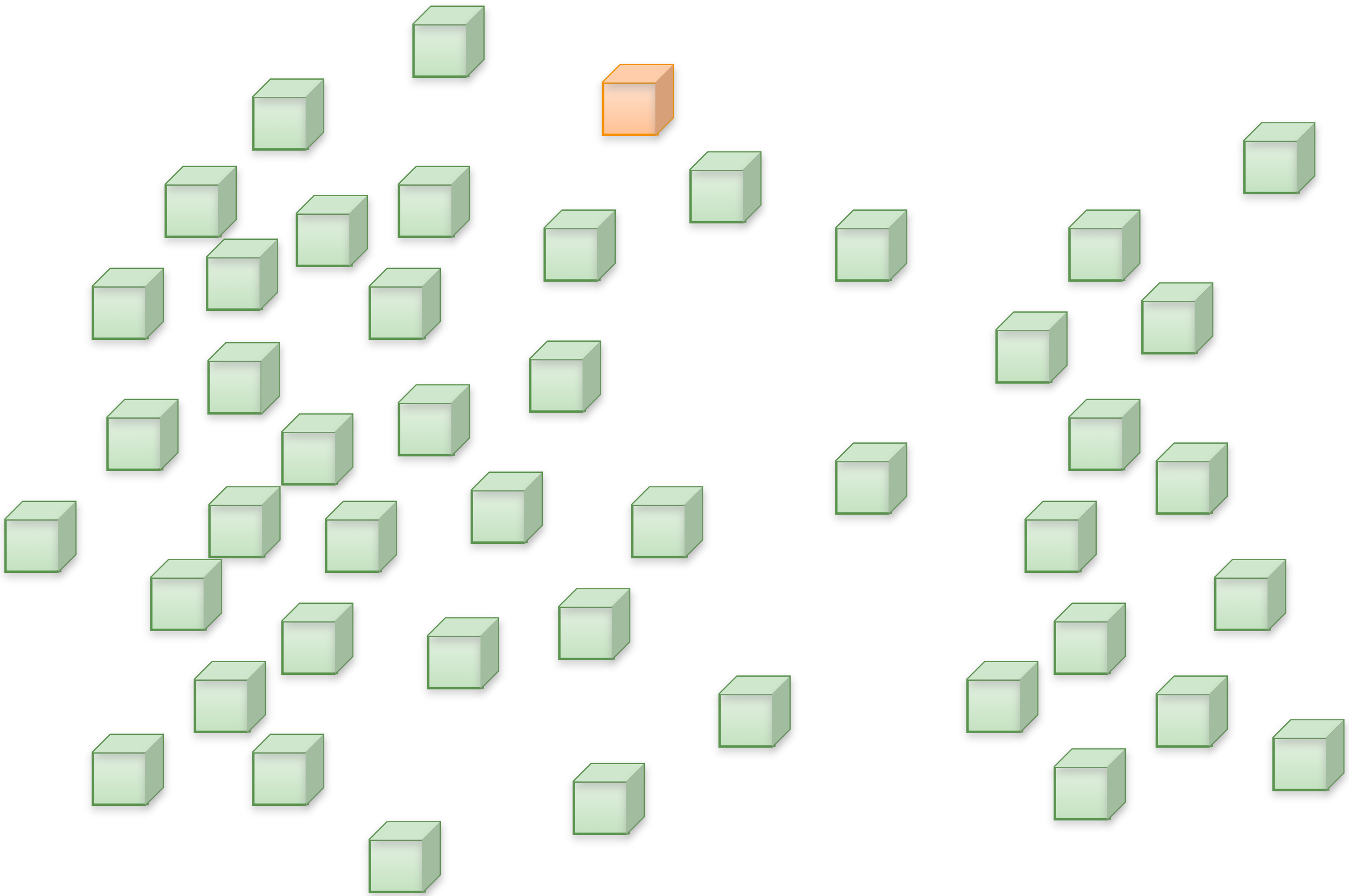
Consistency & Scalability

Existing systems:
tradeoff between consistency and scalability

Examples:
- virtual synchrony (e.g. ISIS, Spread)
- distributed hash tables (e.g. Chord, Pastry)

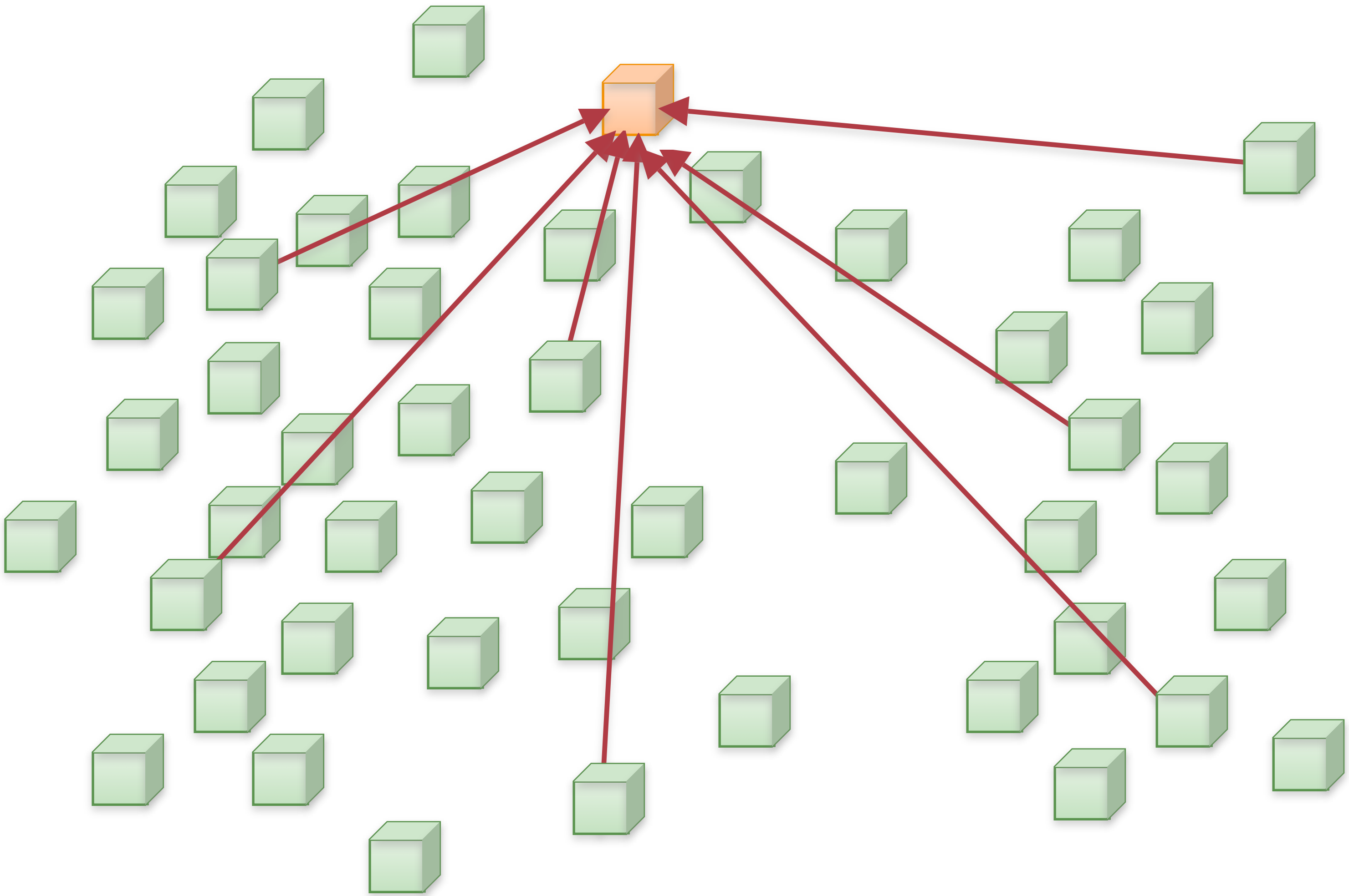
Census provides consistent membership views
and is designed for large-scale, wide-area systems

Membership Service: Basic Approach



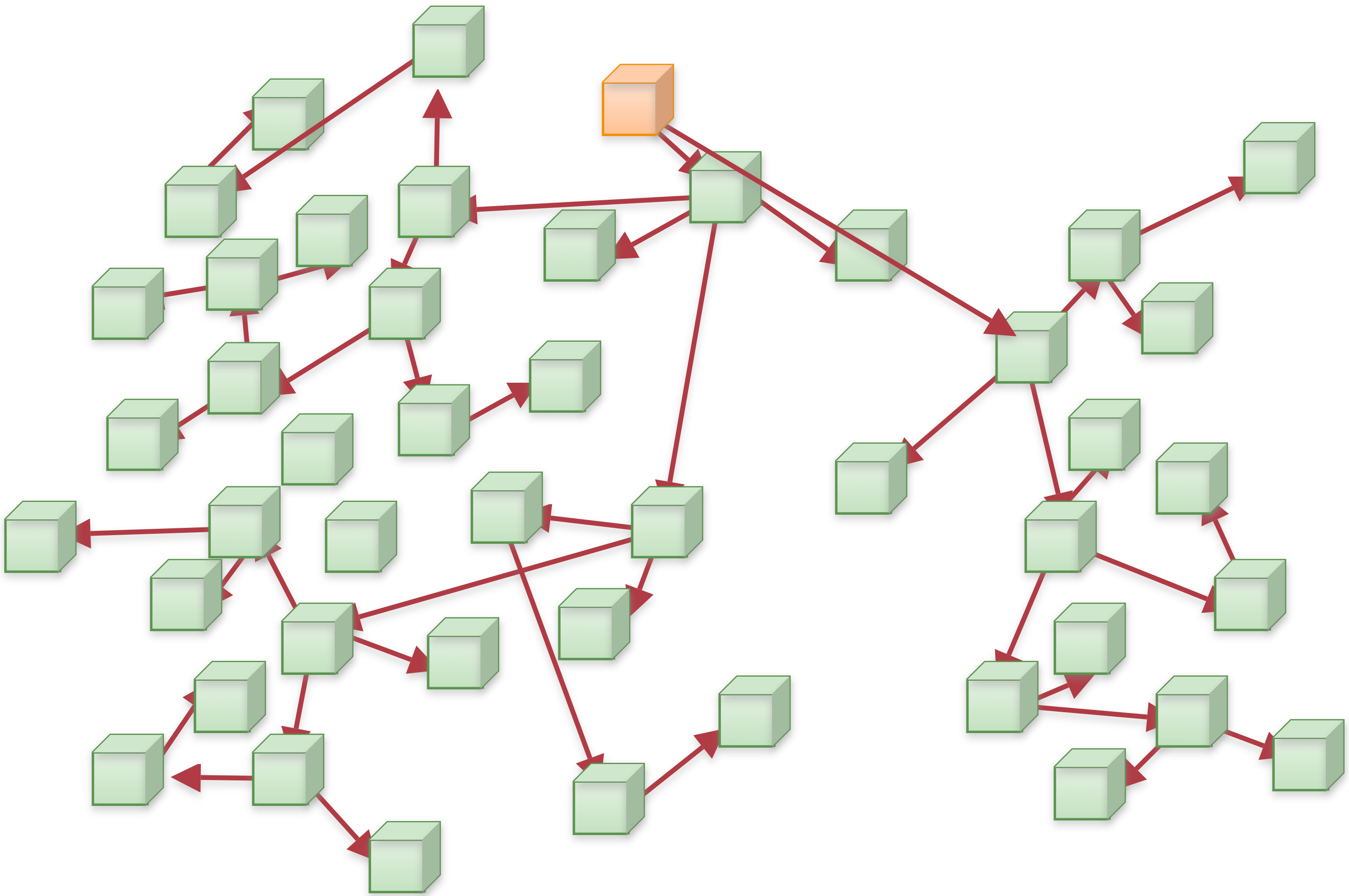
- Designate one node as *leader*

Membership Service: Basic Approach



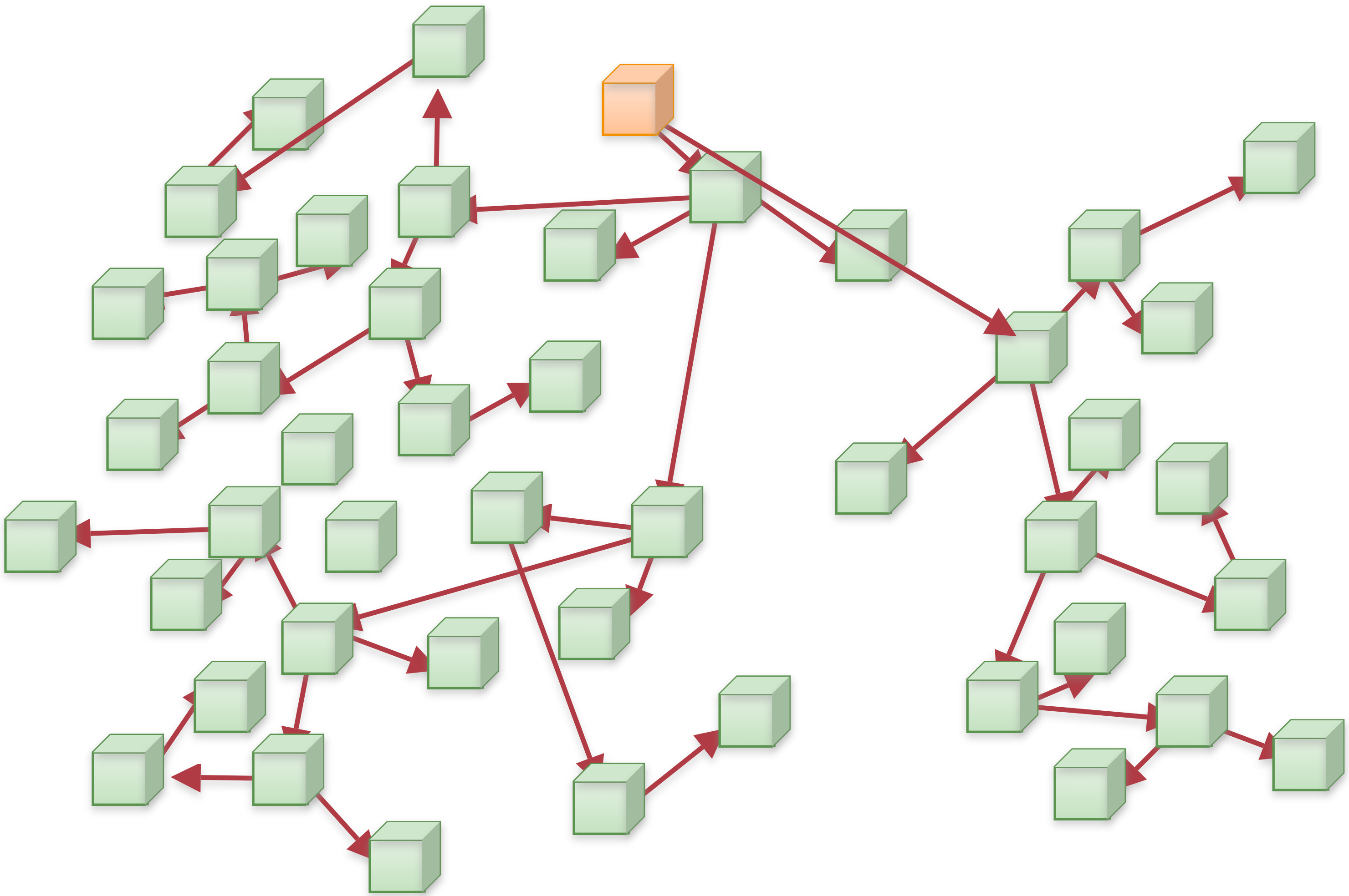
- Designate one node as *leader*
- Nodes report membership changes to leader

Membership Service: Basic Approach



- Designate one node as *leader*
- Nodes report membership changes to leader
- Leader aggregates changes; multicasts *item*

Membership Service: Basic Approach



- Designate one node as *leader*
- Nodes report membership changes to leader
- Leader aggregates changes; multicasts *item*
- Members enter next epoch, update membership

What are the Challenges?

Delivering items efficiently and reliably

➡ Multicast mechanism

Reducing load on the leader

➡ Multi-region structure

Dealing with leader failure

➡ Fault tolerance

Outline

- Overview
- Basic Approach
- **Multicast Mechanism**
- Multi-region Design
- Fault Tolerance
- Evaluation

Multicast Mechanism

Need multicast to distribute membership updates
and application data efficiently

Goals: high reliability, low latency, fair load balancing

Many multicast protocols exist...

Census takes a different approach
    exploits consistent membership information
    for a simpler design and lower overhead

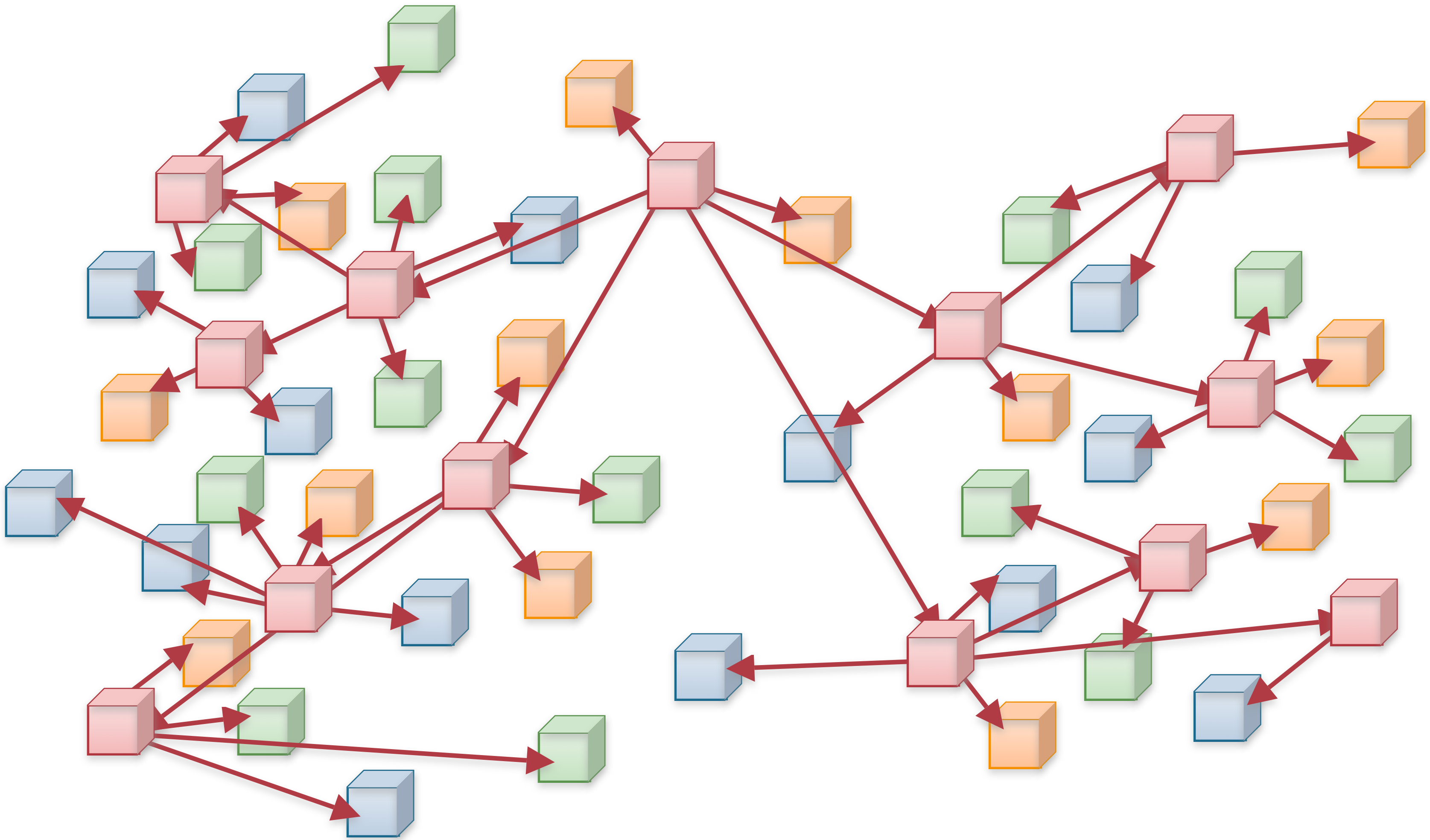
Multicast Topology

Multiple interior-disjoint trees (similar to SplitStream)
- Each node interior in one tree, leaf in others

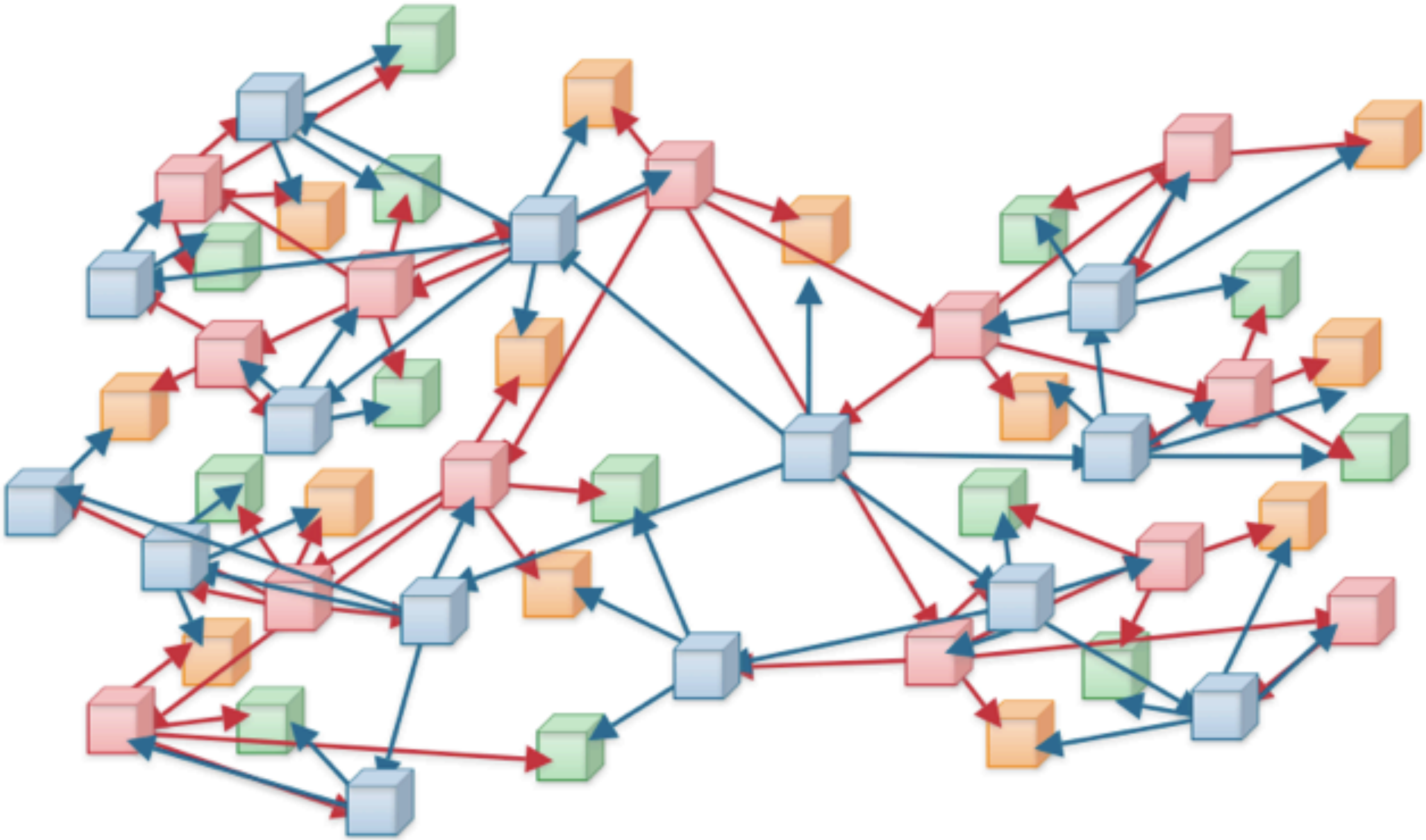
Membership data distributed in full on each tree.
Application's multicast data erasure-coded

Improved reliability and load balancing vs. a single tree

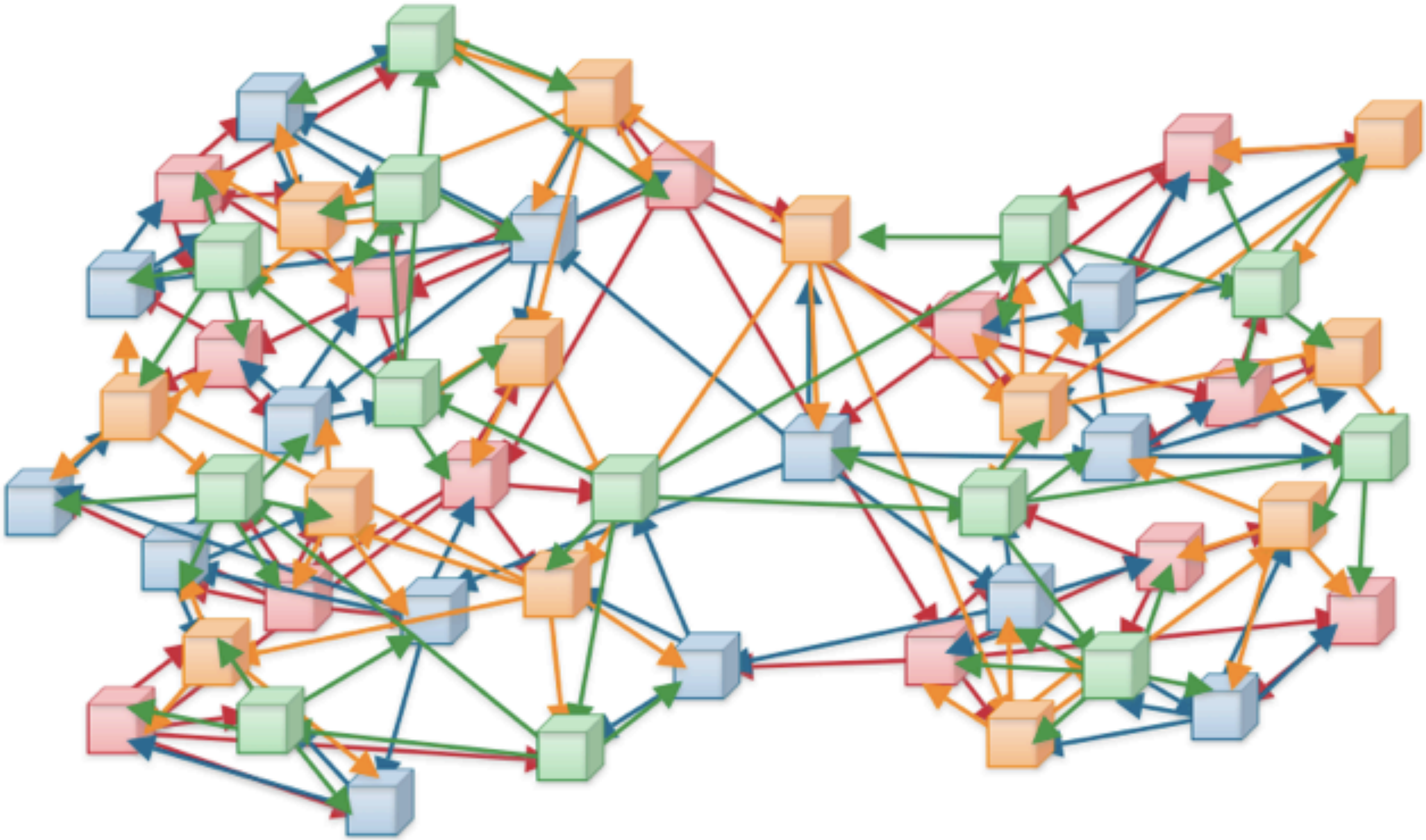
Multicast Topology



Multicast Topology



Multicast Topology



Building Multicast Trees

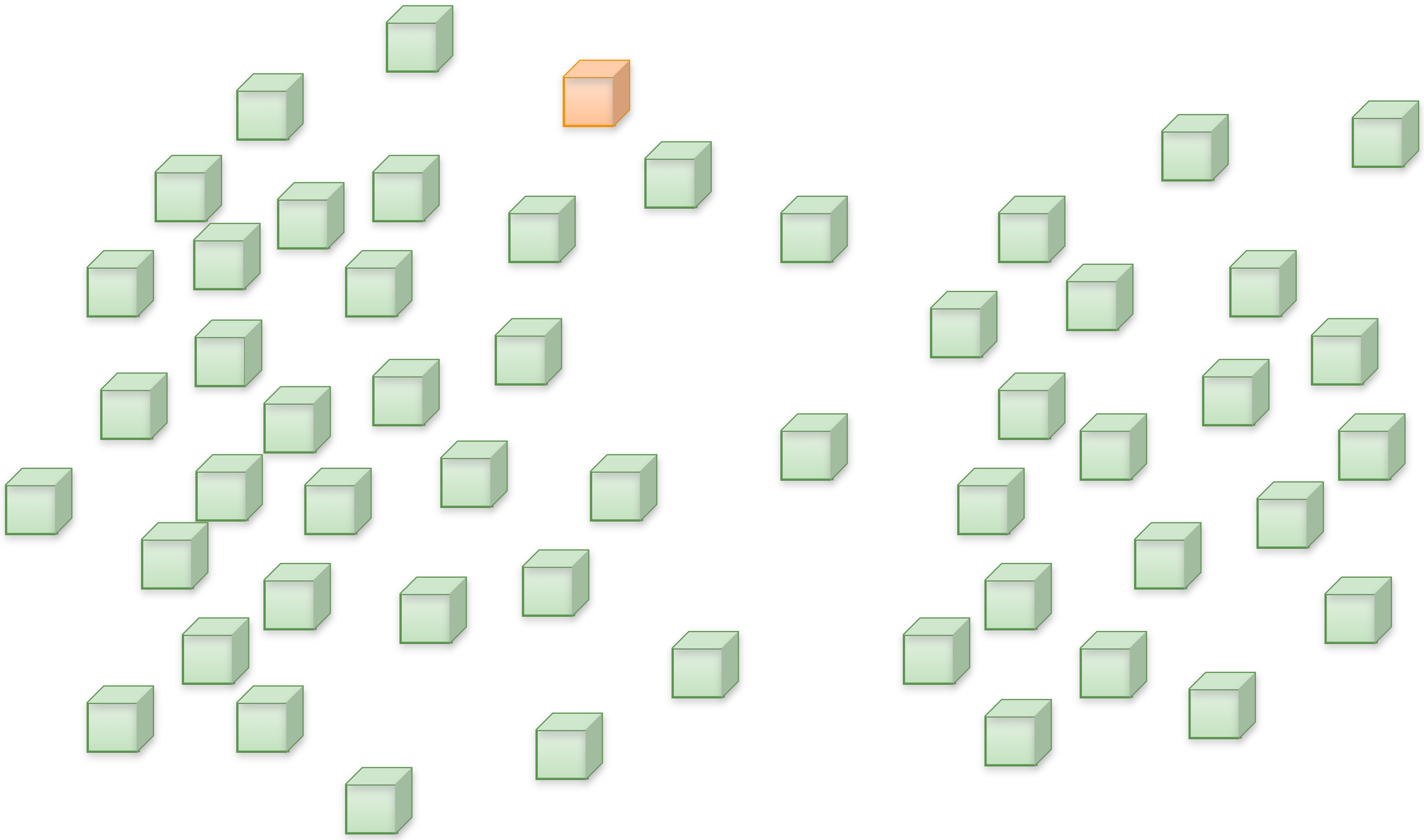
Exploit consistent membership knowledge:
tree structure given by deterministic function of membership

➡ Allows simple “centralized” algorithm in distributed context

Nodes independently recompute trees “on-the-fly”,
upon receiving membership updates

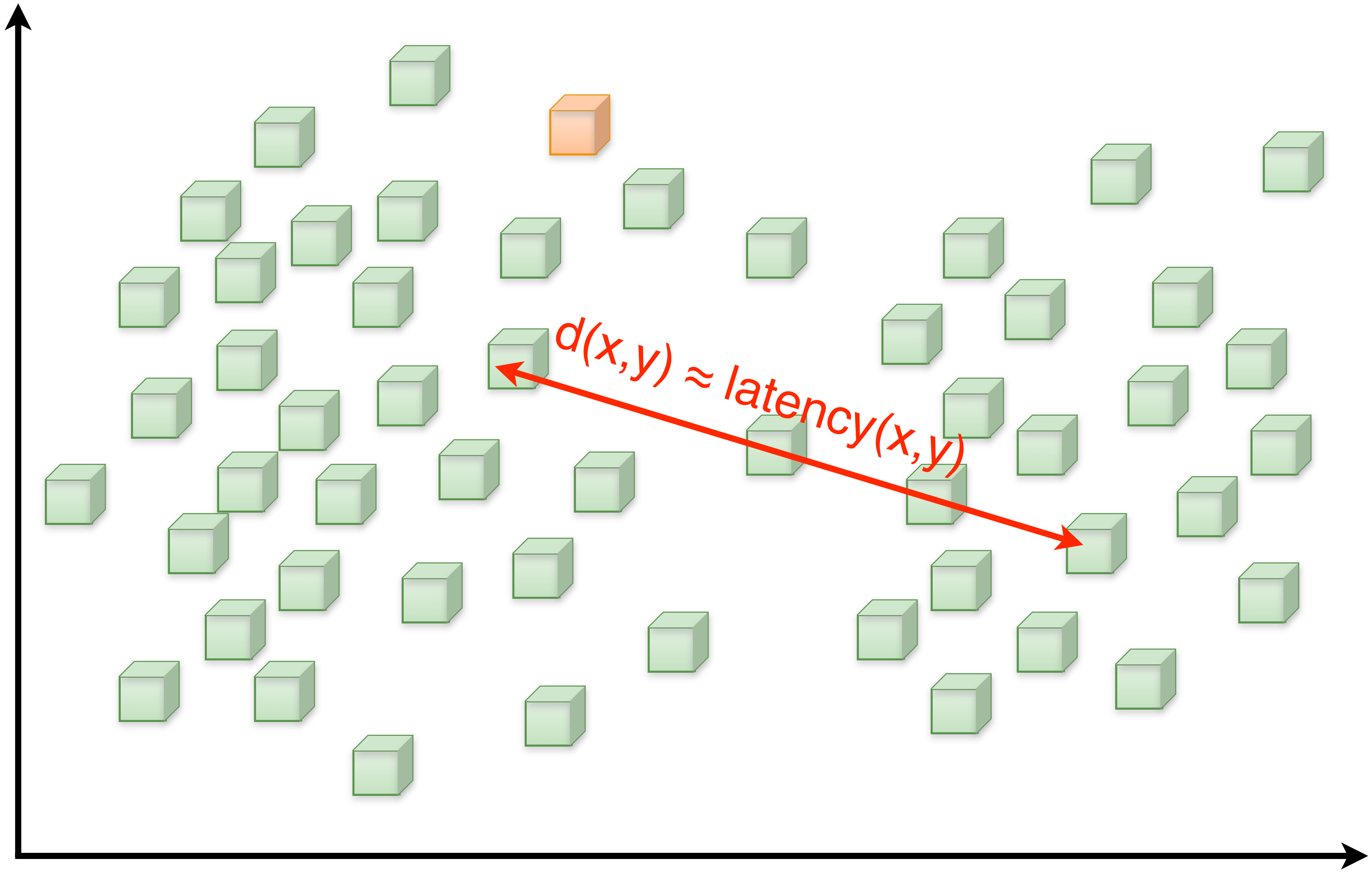
No protocol overhead beyond that of membership service
(even during churn!)

Tree Building Algorithm



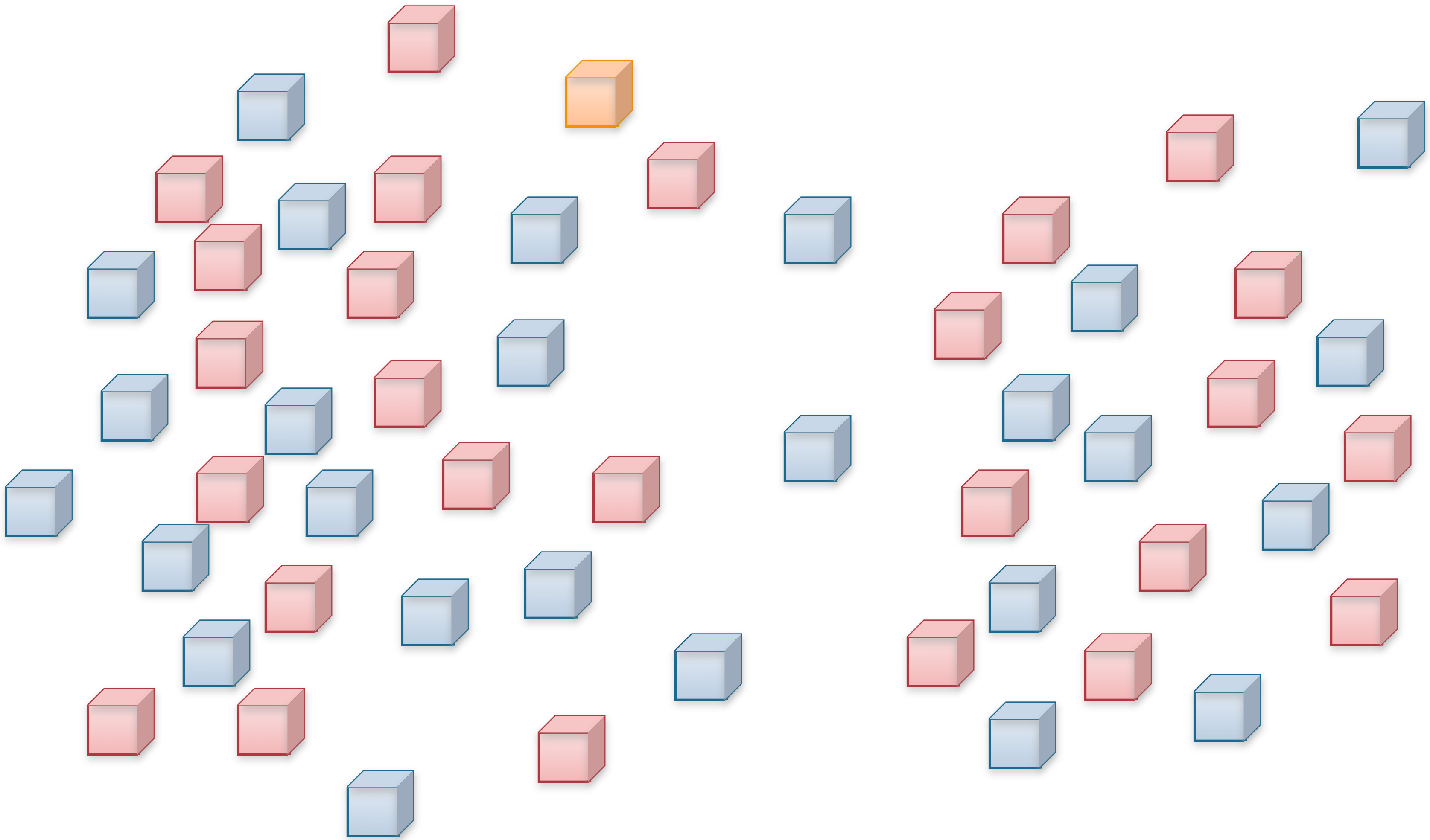
Tree Building Algorithm

Background: network coordinates (e.g. Vivaldi)



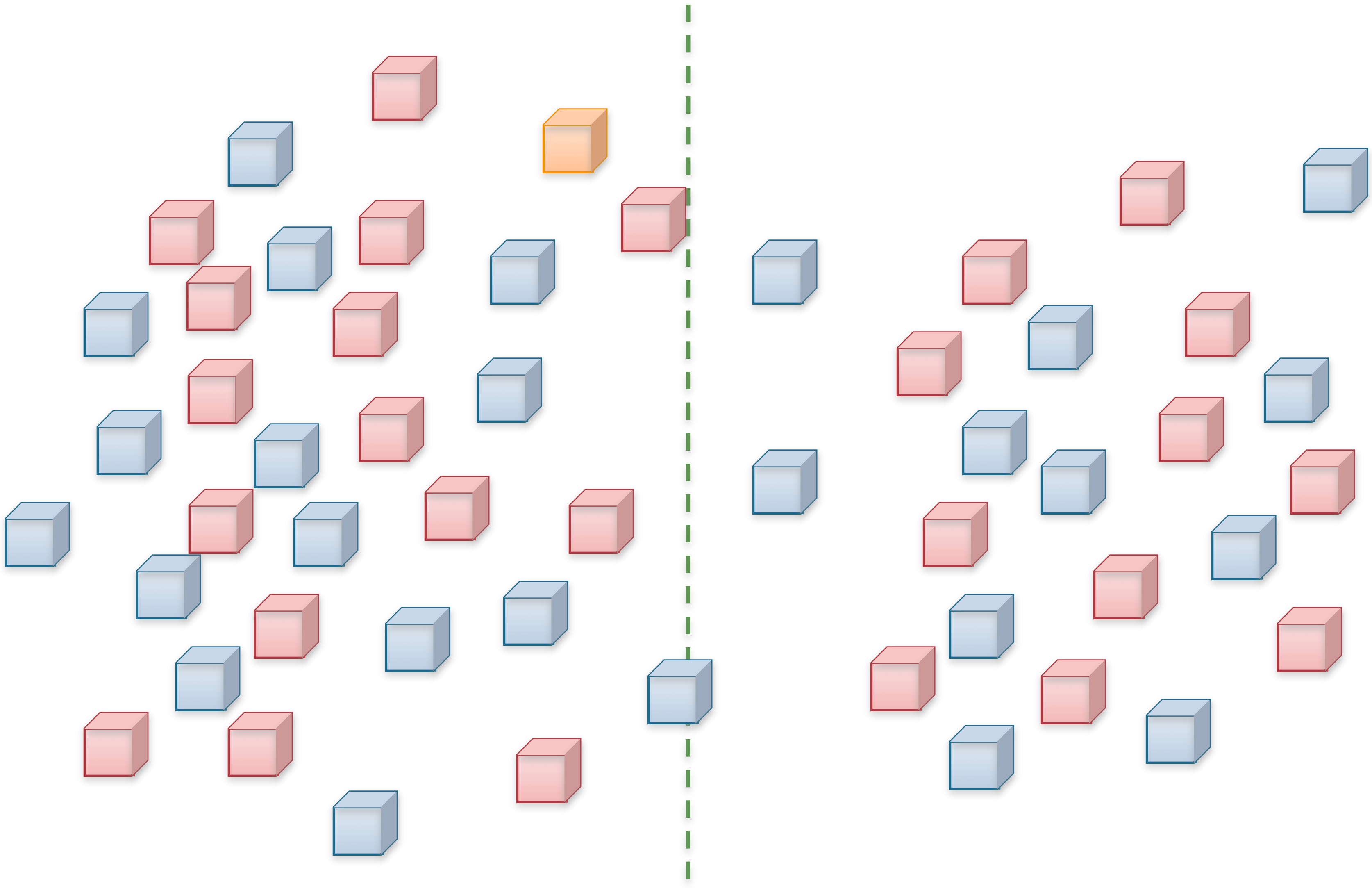
Tree Building Algorithm

Assign nodes to a tree (color) based on ID



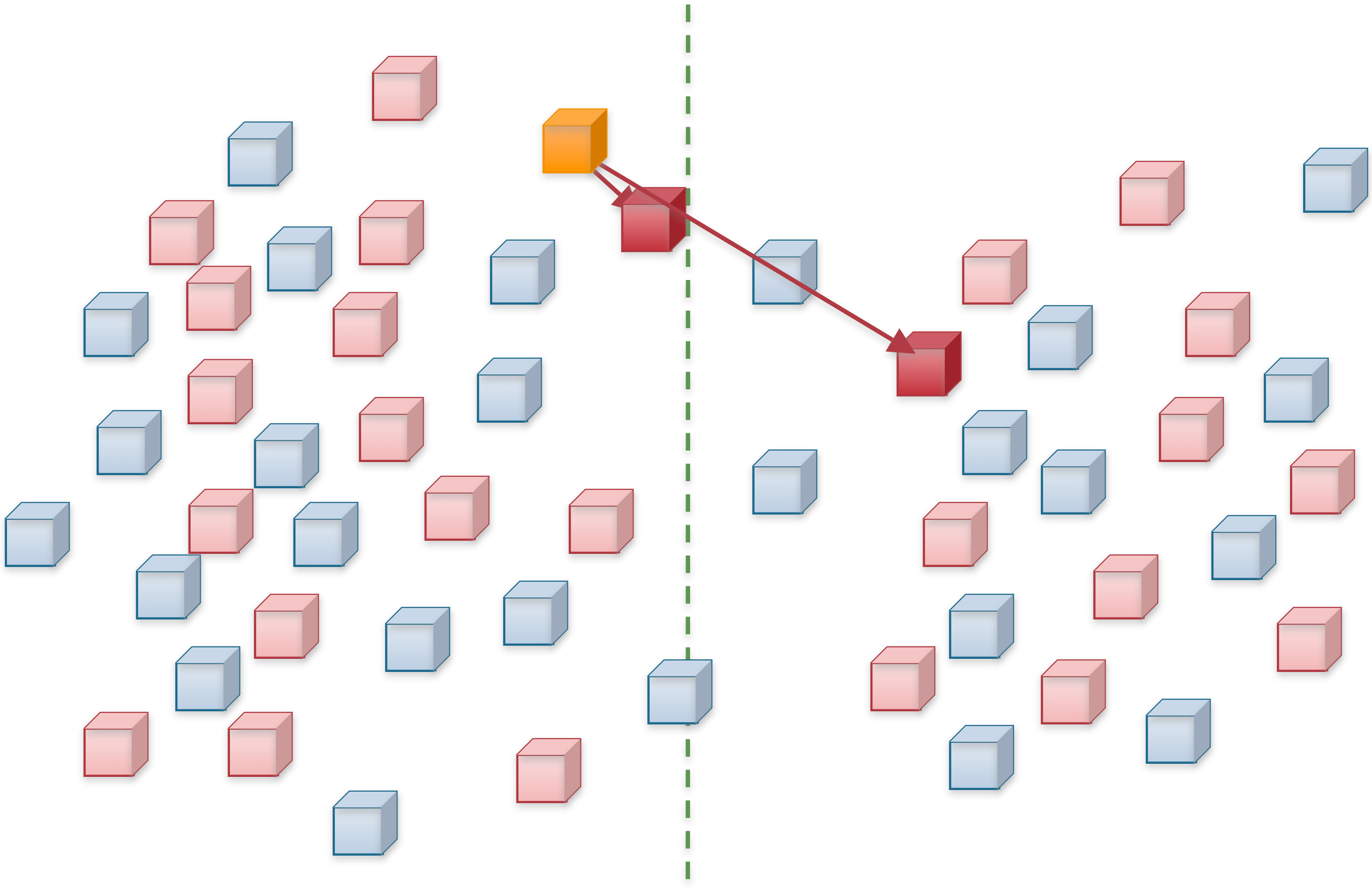
Building the Red Tree

Split region through center of mass, along widest axis



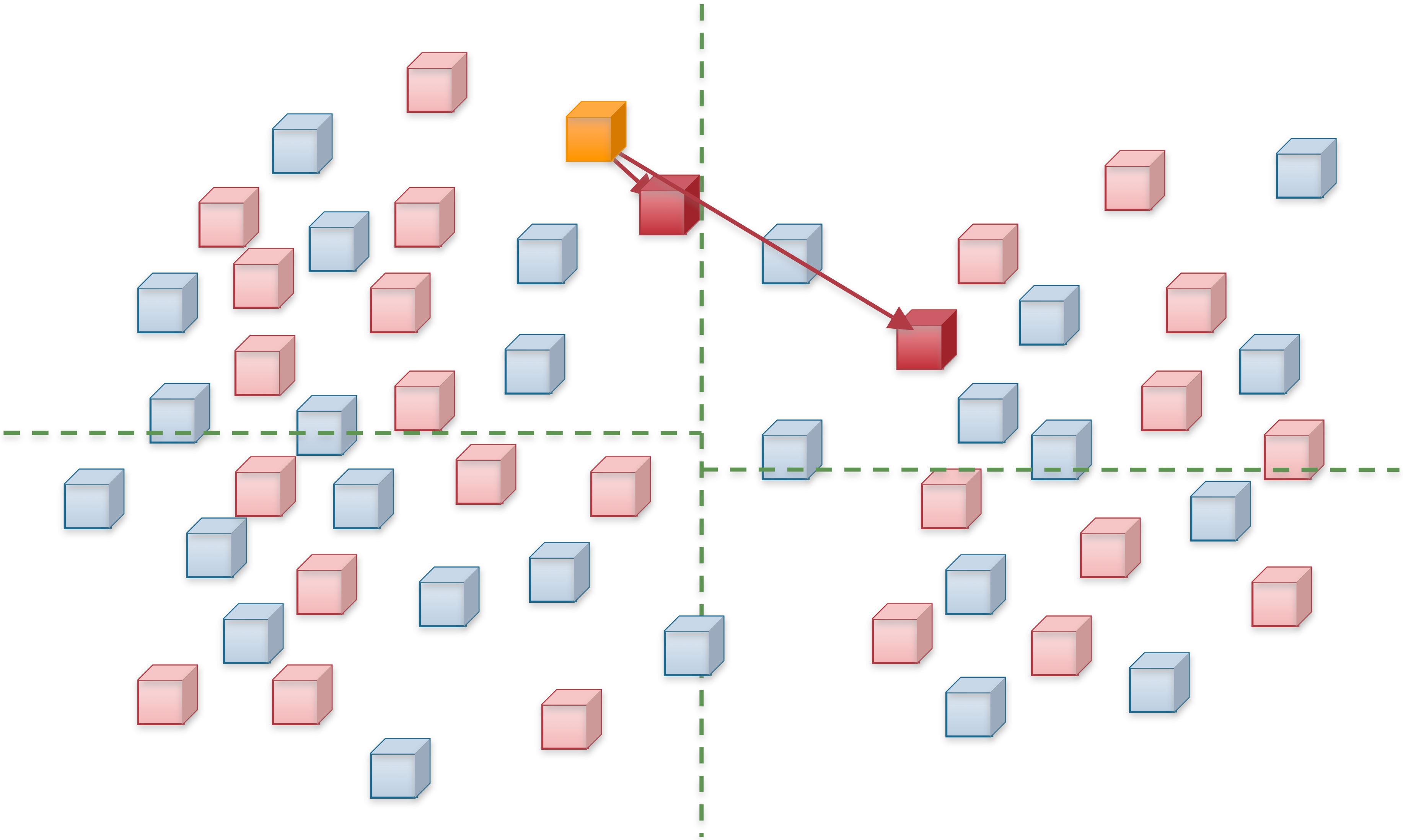
Building the Red Tree

Choose closest red node in each subregion, attach to root



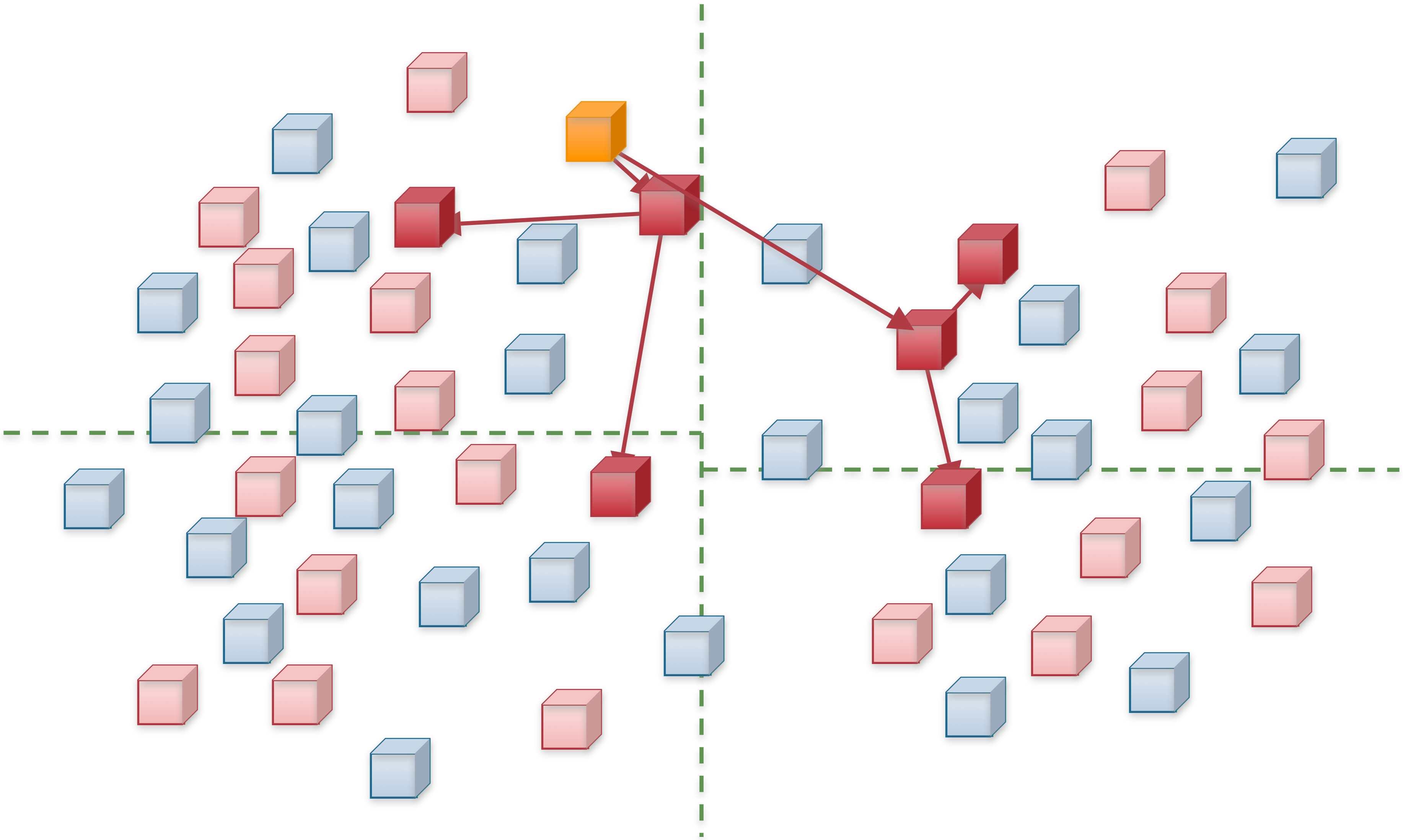
Building the Red Tree

Recursively subdivide each subregion in the same way



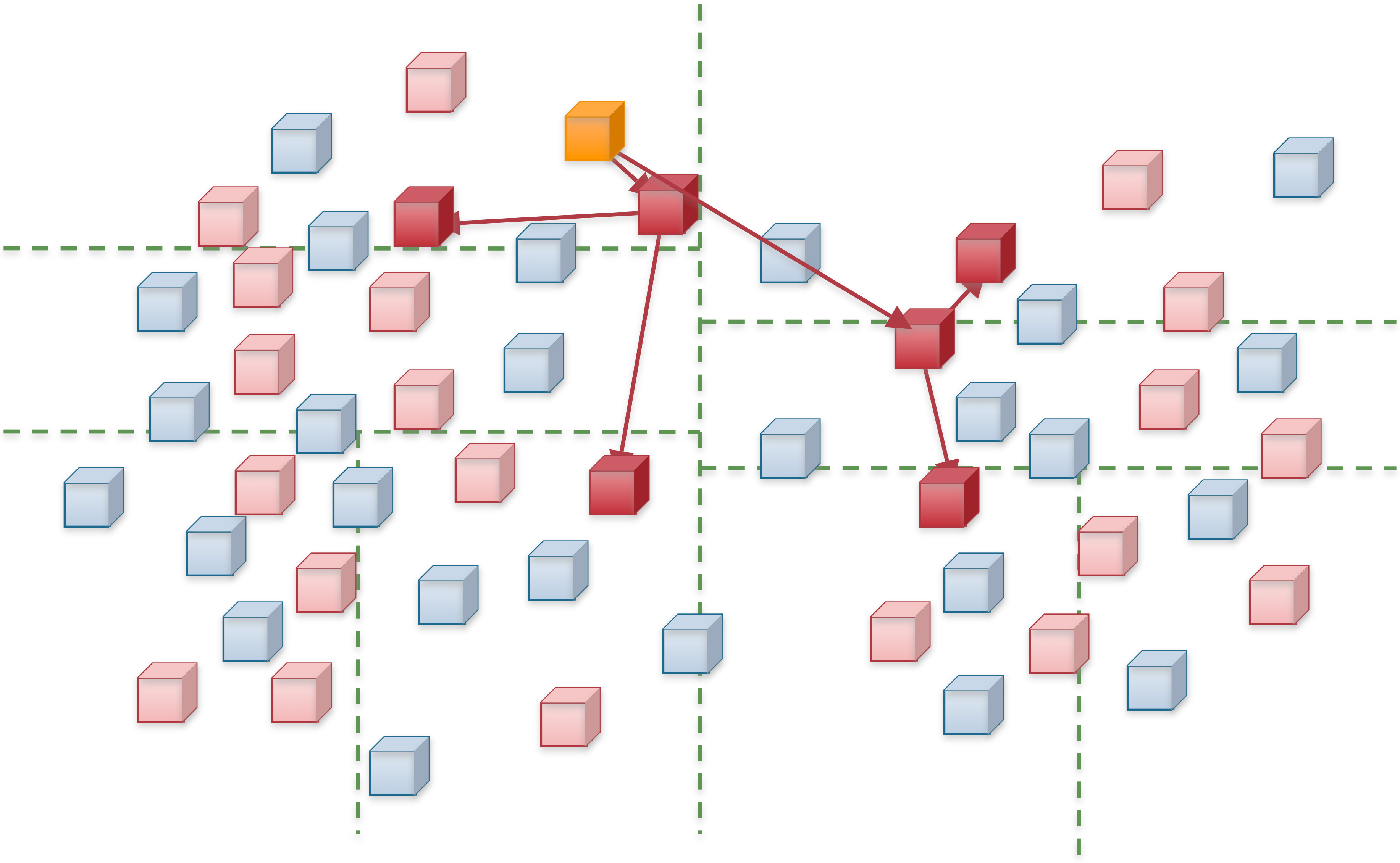
Building the Red Tree

Recursively subdivide each subregion in the same way



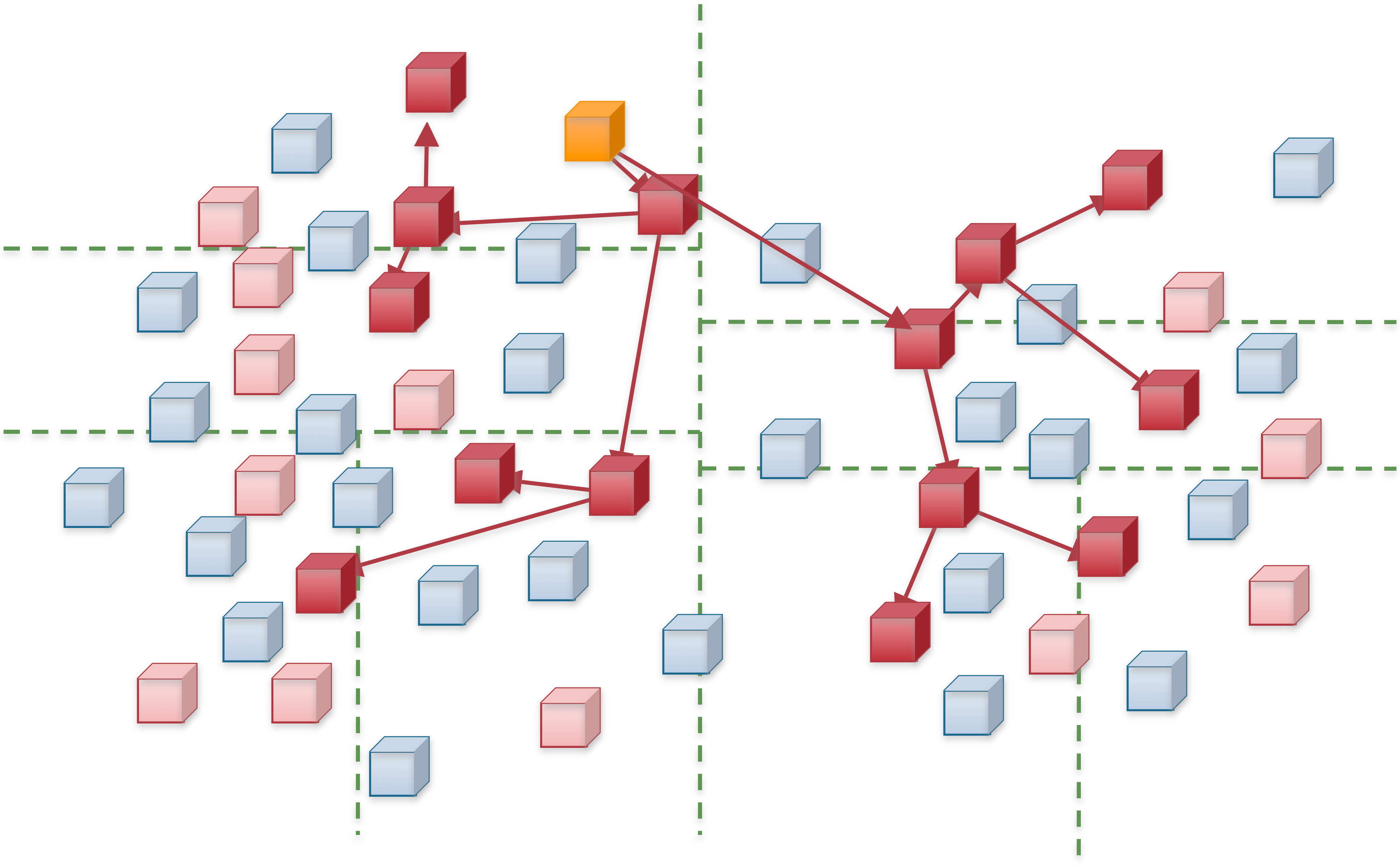
Building the Red Tree

Recursively subdivide each subregion in the same way



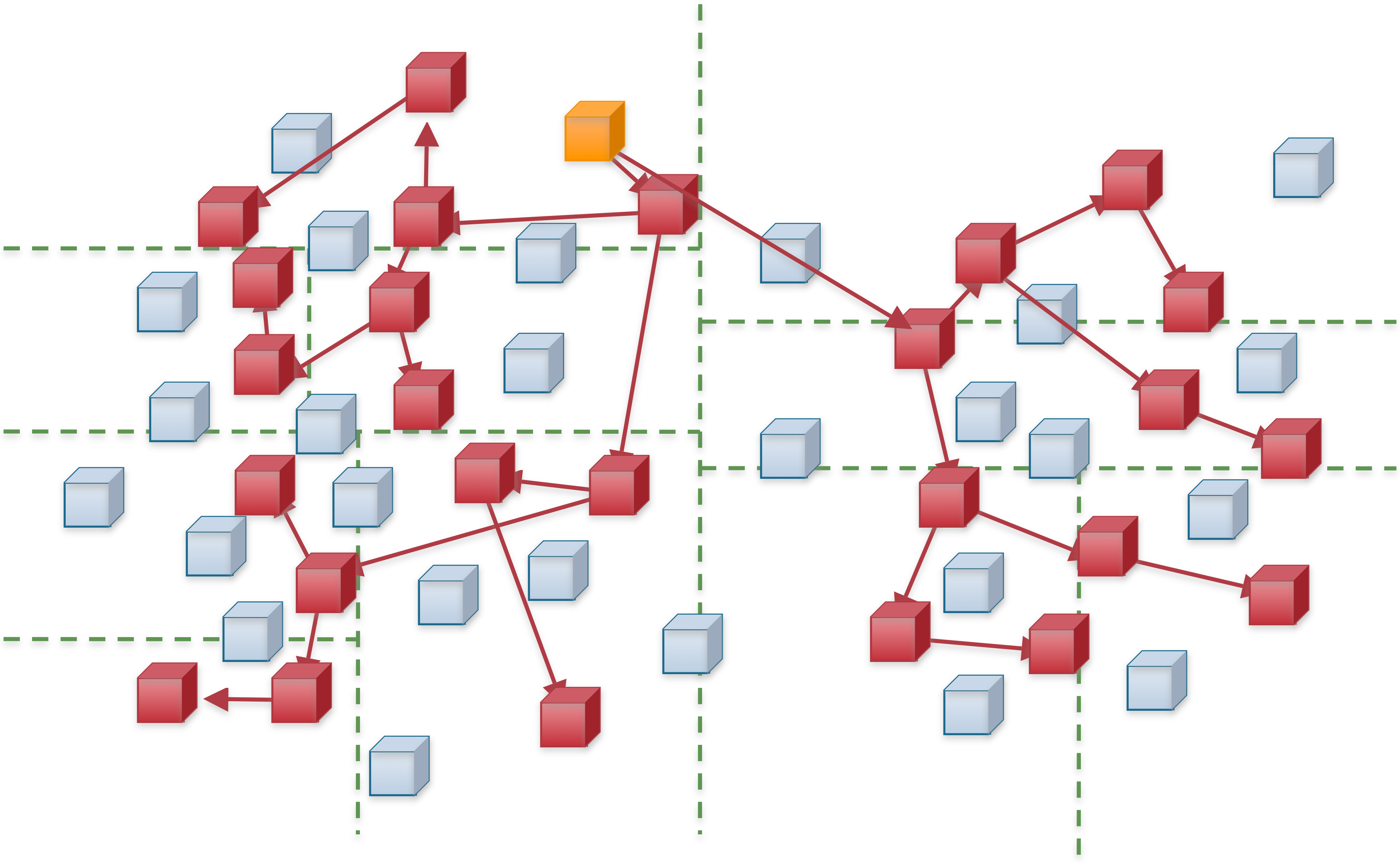
Building the Red Tree

Recursively subdivide each subregion in the same way



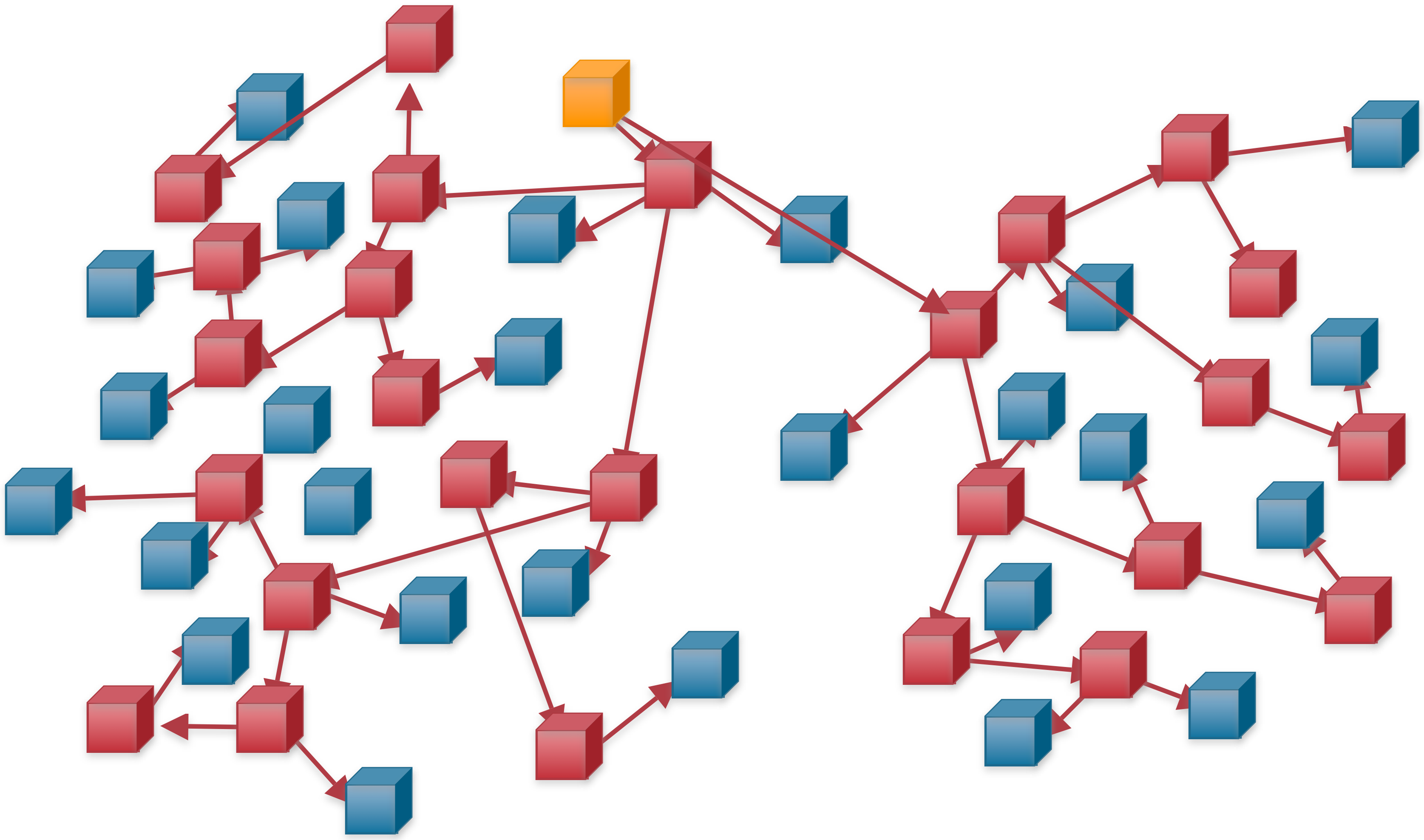
Building the Red Tree

Recursively subdivide each subregion in the same way



Building the Red Tree

Attach other-colored nodes to the nearest available red node



Multicast Improvements

Reduce bandwidth overhead
- avoid sending redundant data

Reduce multicast latency
- choose fragments to send based on expected path length

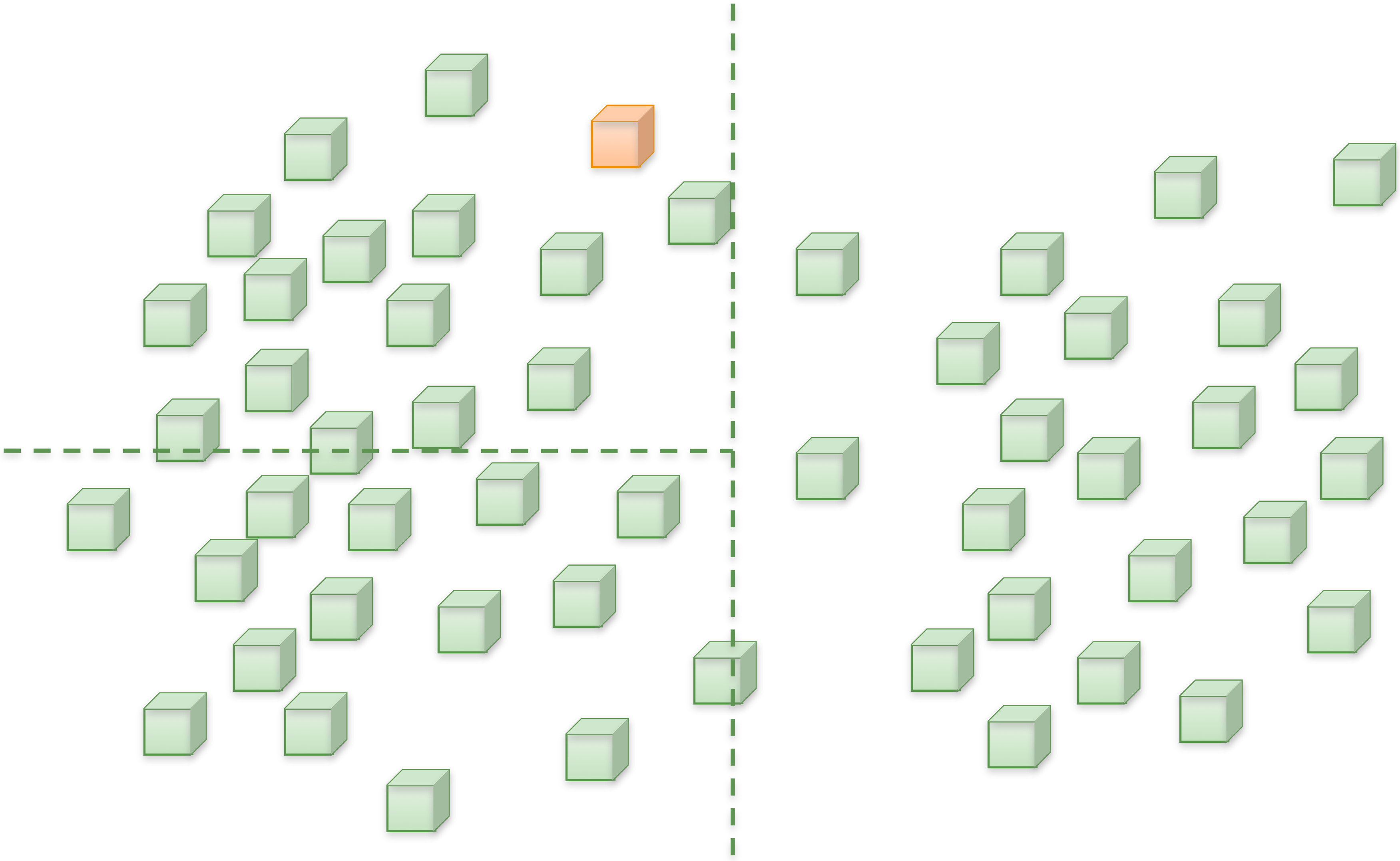
Improve reliability during failures
- reconstruct missing fragments from other trees

Outline

- Overview
- Basic Approach
- Multicast Mechanism
- **Multi-region Design**
- Fault Tolerance
- Evaluation

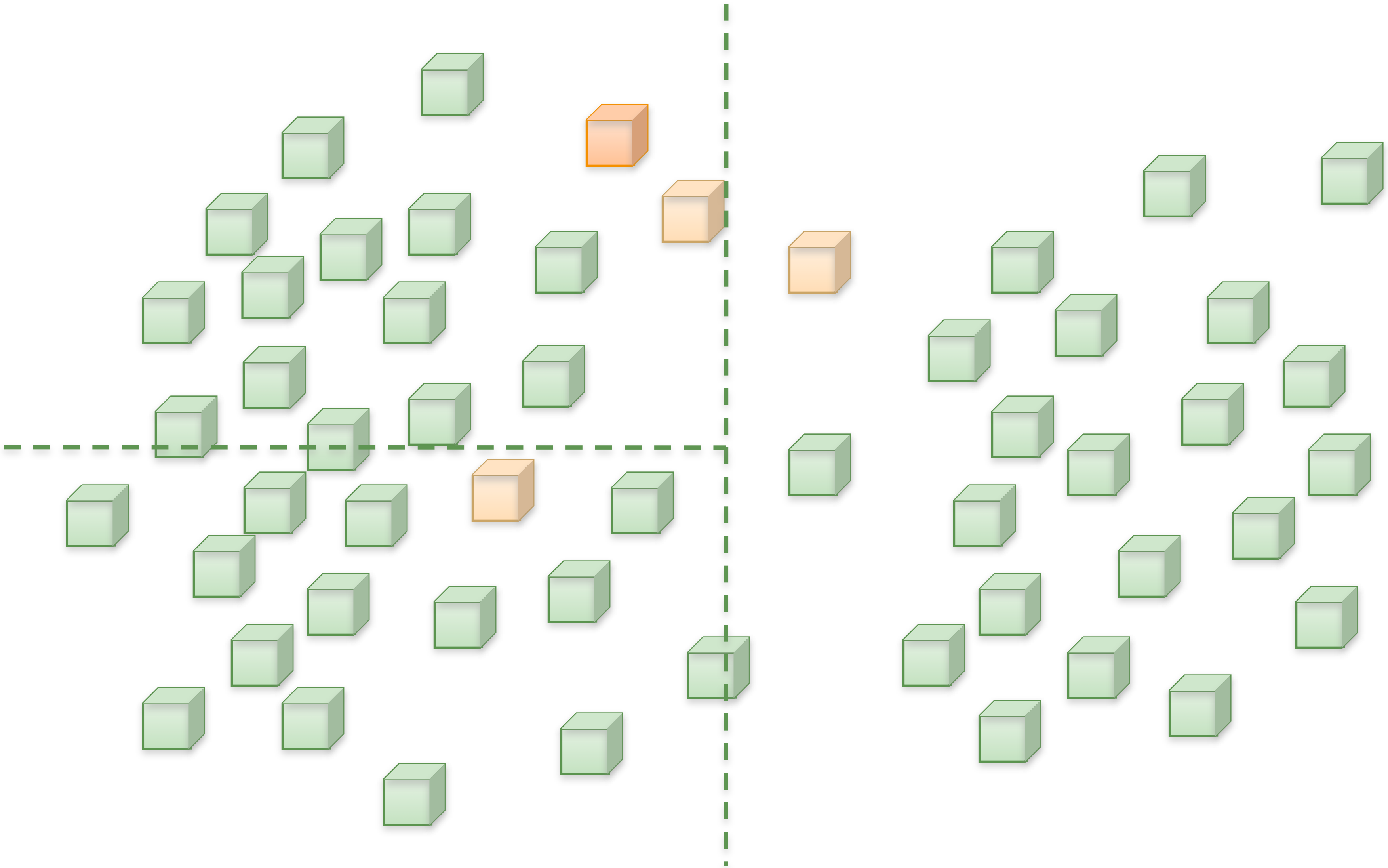
Multi-Region Structure

Divide large deployments into location-based regions



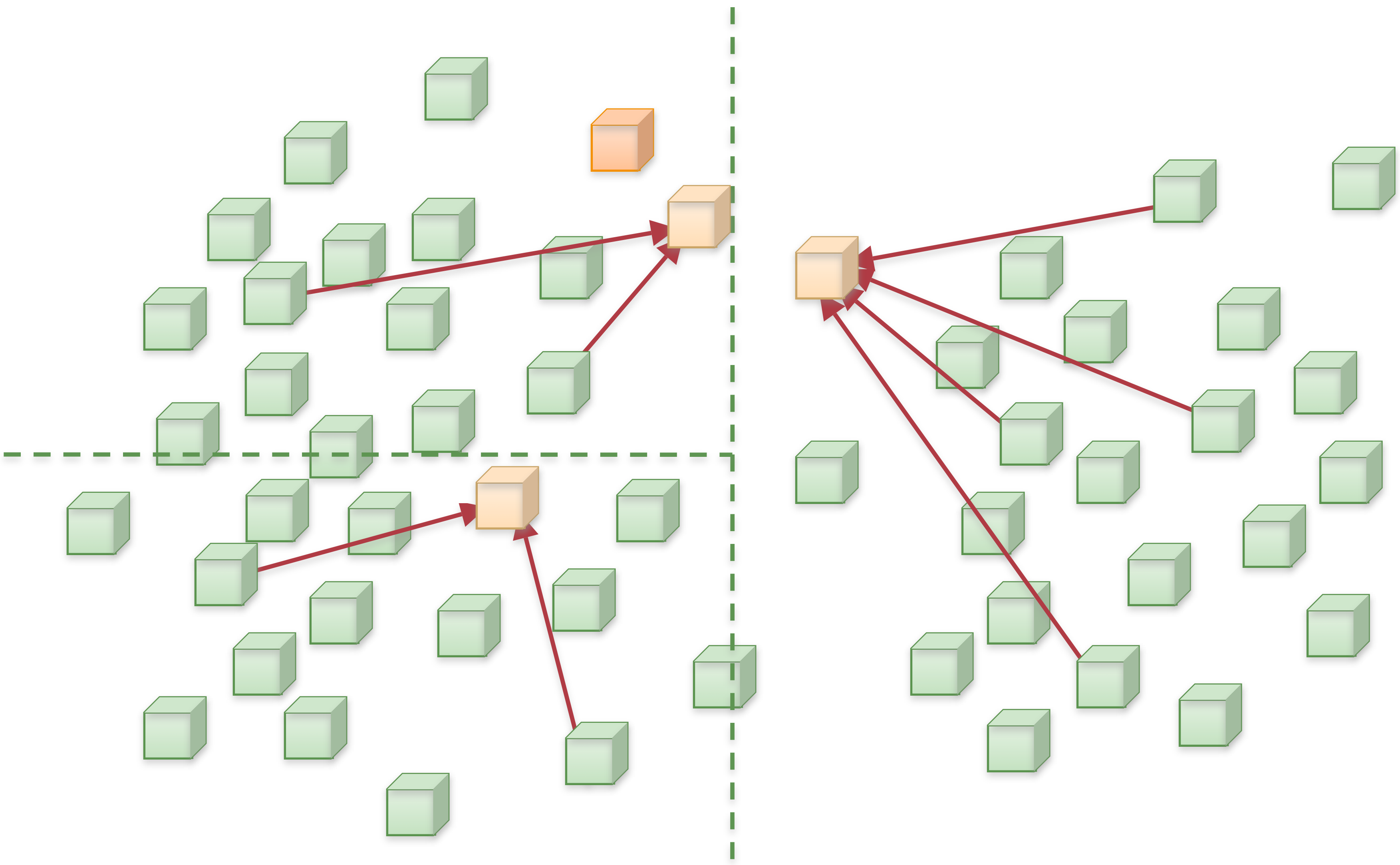
Multi-Region Structure

One *region leader* per region, plus global leader



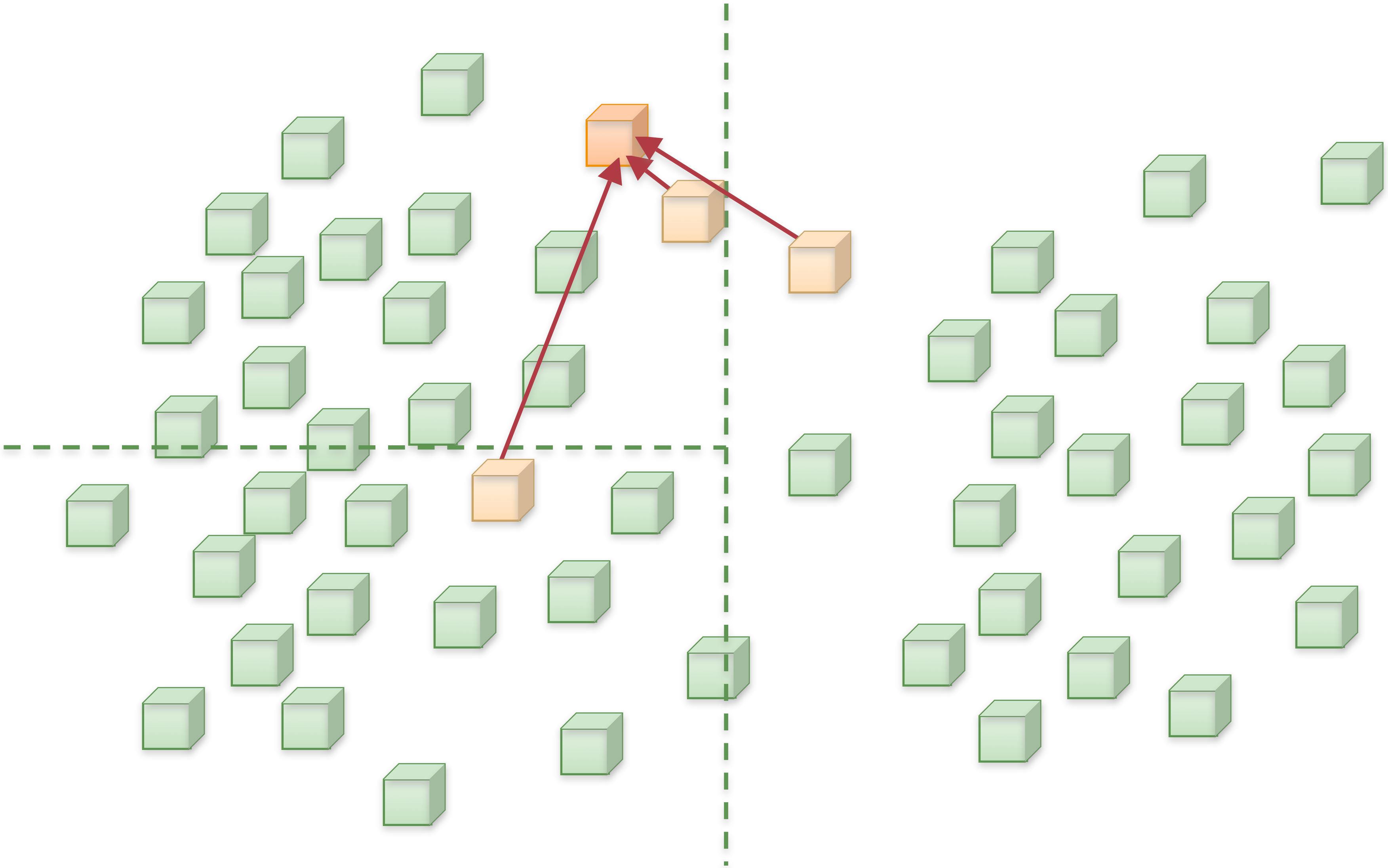
Multi-Region Structure

Region leaders aggregate membership changes from region



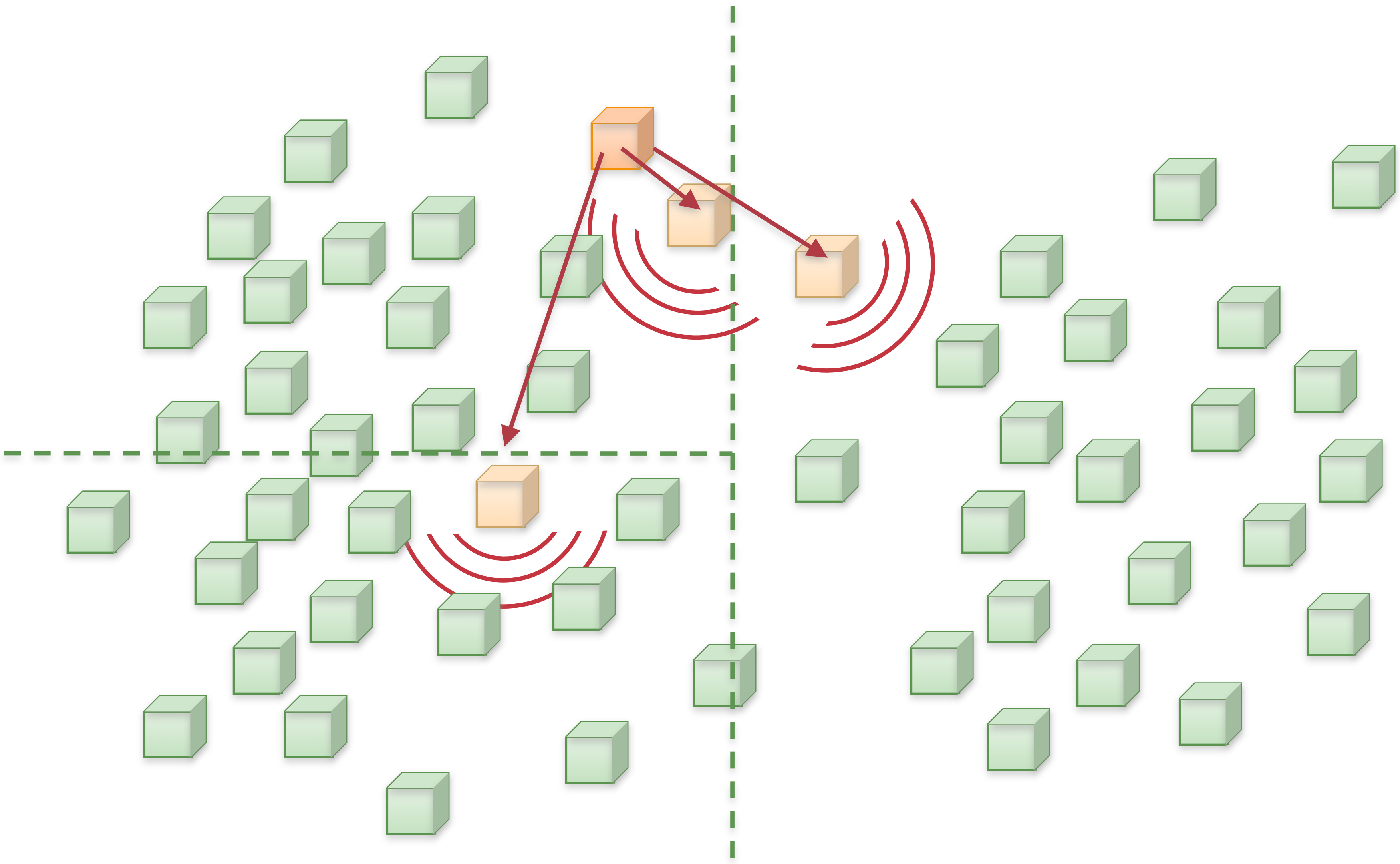
Multi-Region Structure

Region leaders aggregate membership changes from region



Multi-Region Structure

Global leader combines region reports to produce item



Region Dynamics

Regions split when they grow too large
- Global leader signals split in the next item
- Nodes independently split region across widest axis
  - using consistent membership knowledge

Regions merge when one grows too small
- Similar process

Nodes assigned to nearest region on joining

Multi-Region Structure

Benefits

- fewer messages processed by leader
- fewer wide-area communications
- cheaper multicast tree computation
- useful abstraction for applications

Partial Knowledge

Maintaining global membership knowledge is usually feasible

Except: very large, dynamic, and/or bandwidth-constrained systems

Partial knowledge:
  each node knows only the membership of its own region
  and summary information of other regions

Outline

- Overview
- Basic Approach
- Multicast Mechanism
- Multi-region Design
- **Fault Tolerance**
- Evaluation

Fault Tolerance

Global leader and region leaders can fail

Solution: replication

  Use standard state machine replication techniques

  Replication level based on expected *concurrent* failures

Optional: tolerating Byzantine faults

Outline

- Overview
- Basic Approach
- Multicast Mechanism
- Multi-region Design
- Fault Tolerance
- **Evaluation**

Evaluation

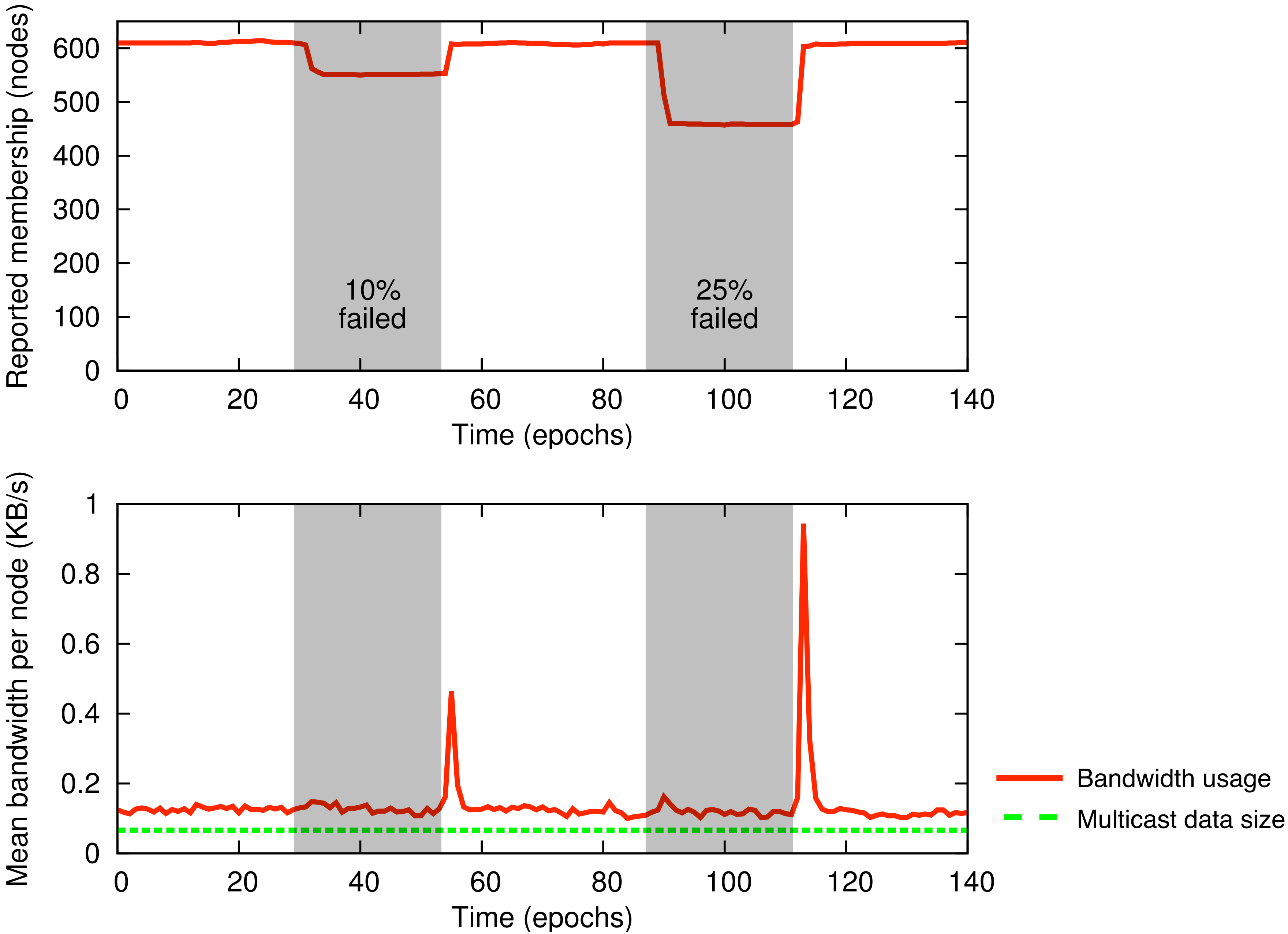
PlanetLab deployment
  614 nodes

Theoretical analysis
  scalability to larger systems

Simulator
  evaluate multicast performance

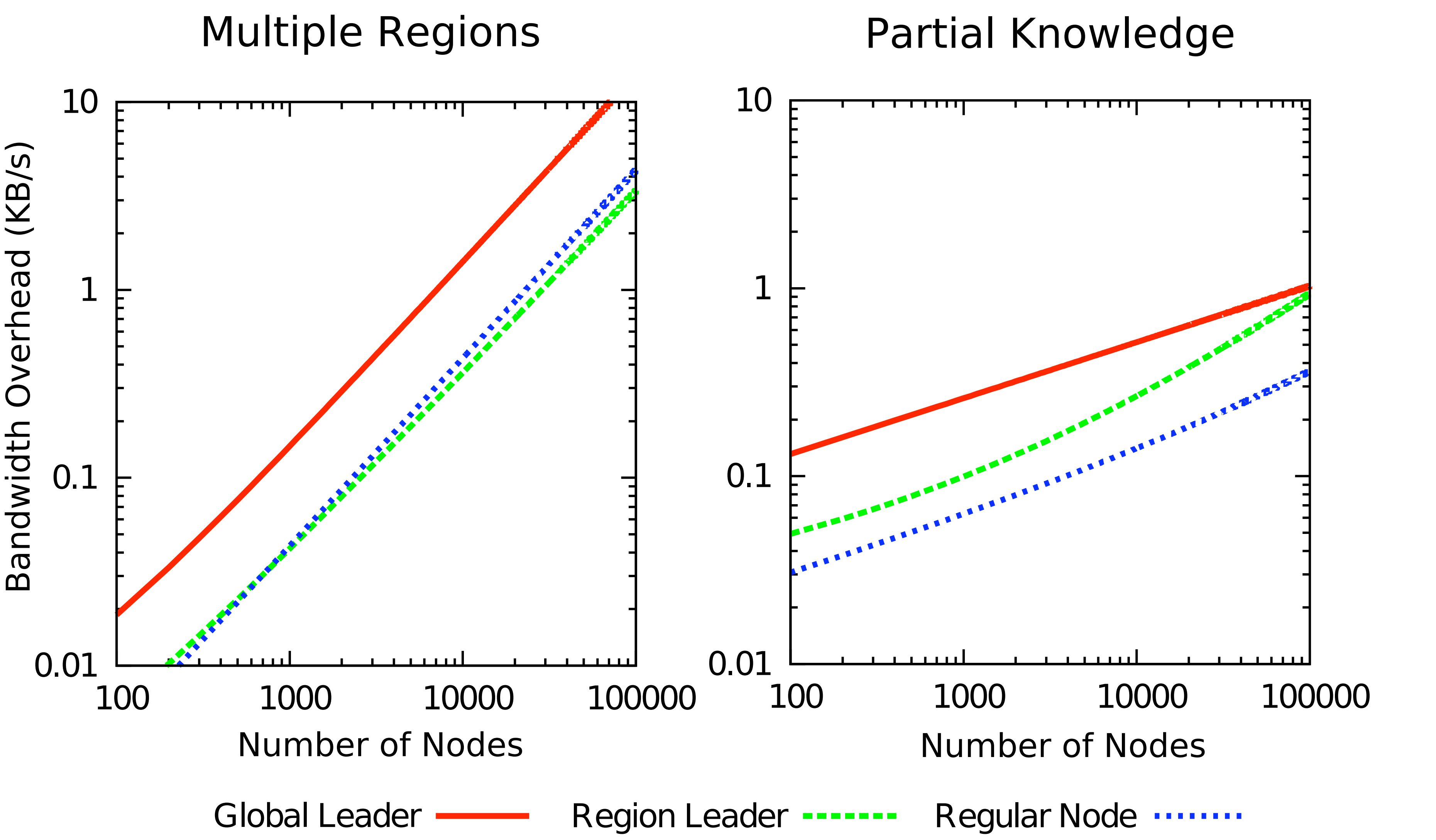
PlanetLab Deployment

614 nodes; 30 second epochs; 1 KB/epoch multicast



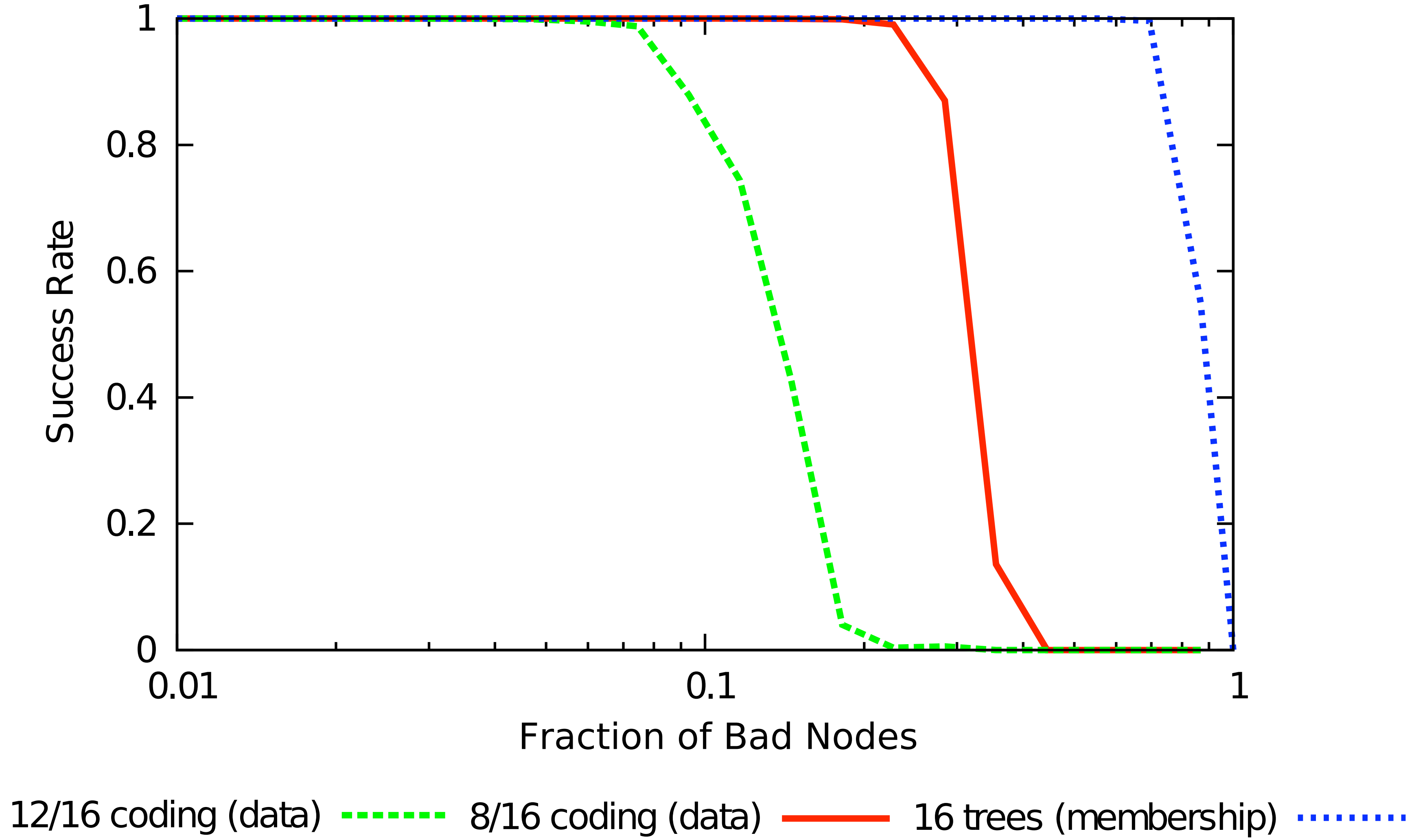
Bandwidth Overhead

Membership management cost analysis
Very high churn rate (avg. node lifetime 30 minutes)



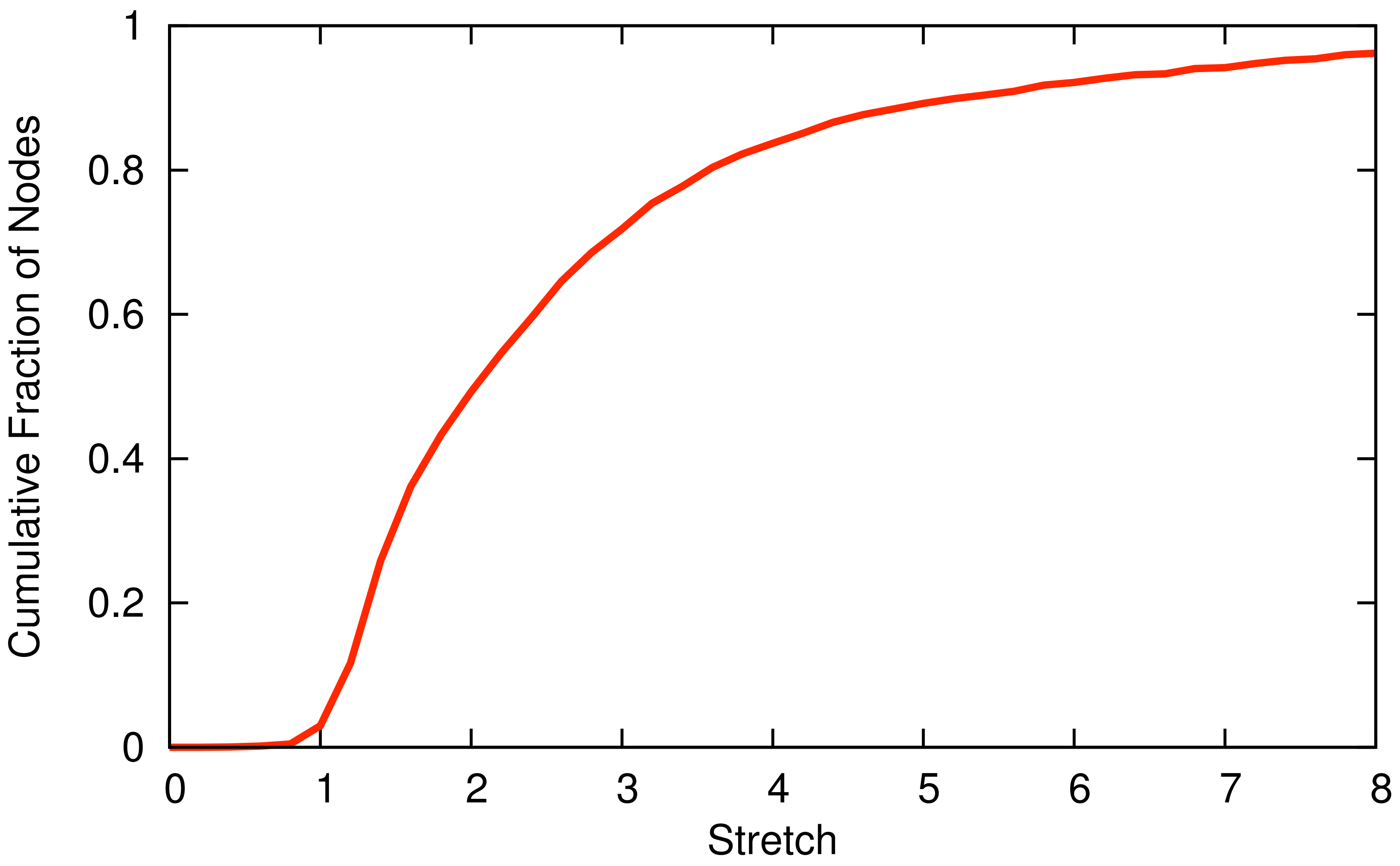
Multicast Reliability

Fraction of nodes successfully receiving multicast
Simulation results (10,000 nodes)



Multicast Performance

Stretch: multicast latency / ideal (unicast) latency
1740-node measurement-derived topology



Conclusion

Census: a platform for membership management and communication in large distributed systems

Provides consistent views while scaling to extreme sizes
- Support future wide-scale distributed applications

Builds on an efficient multicast mechanism
- High reliability, low latency, low bandwidth overhead

Exploit consistent knowledge
- High performance while avoiding complexity

Conclusion

Census: a platform for membership management and communication in large distributed systems

Provides consistent views while scaling to extreme sizes
    Support future wide-scale distributed applications

Builds on an efficient multicast mechanism
    High reliability, low latency, low bandwidth overhead

Exploit consistent knowledge
    High performance while avoiding complexity

Thank you. Questions?