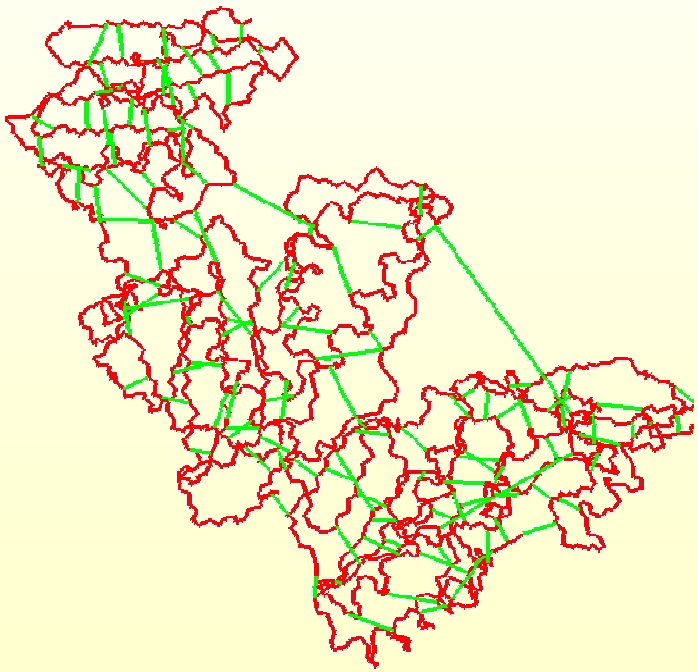
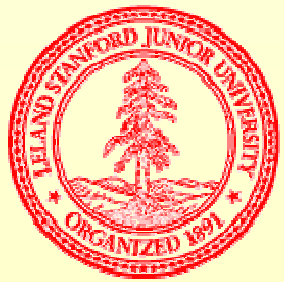
Multiresolution Proximity Maintenance for Moving Objects



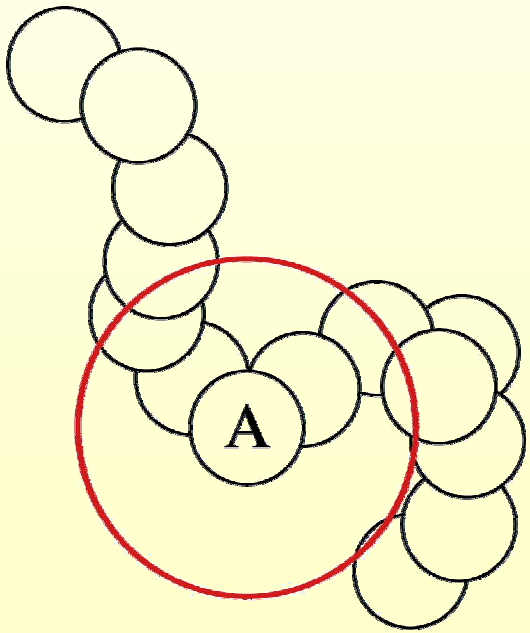
Including a Quick Introduction to Kinetic Data Structures

Jie Gao, Leonidas J. Guibas, An Nguyen
Computer Science
Stanford University

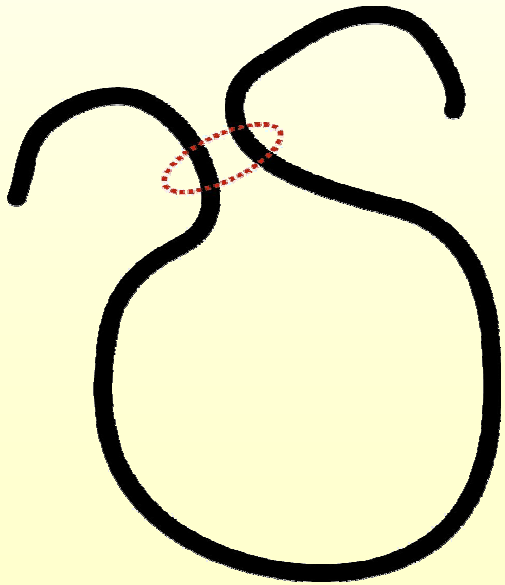


Proximity Information in Physical Simulations

- Most forces in nature are *short range*: neighbor lists in MD



- Collision, or self-collision detection



objects interact when they are near …

Challenges in Proximity Maintenance

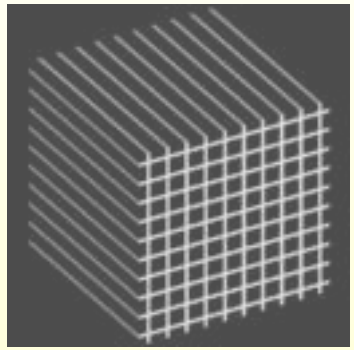
- Proximity information can change rapidly
- Each object can potentially come near every other object ($O(n^2)$ interactions)
- We seek a proximity data structure that at the same time
  - is relatively stable under motion
  - is output-sensitive
  - quickly delivers the required (distance, collision, etc.) information

Some Questions …

- *I'm a statistician and only care about data analysis; why do I have to hear about simulations?*
  - Proximity maintenance is also important for observed motions, using sensors --- particularly in a distributed setting
  - Even more so when you have to take actions quickly, based on proximity information
- *I only care about multiresolution analysis and proximity has to do with just small distances; where are the other scales?*
  - Sometimes, to solve a problem, one has to enlarge it first …
- *3-D is trivial; why should I pay attention when I only care about data in high dimensional spaces?*
  - Because you make care about data in spaces that don't even have coordinates, like certain metric spaces

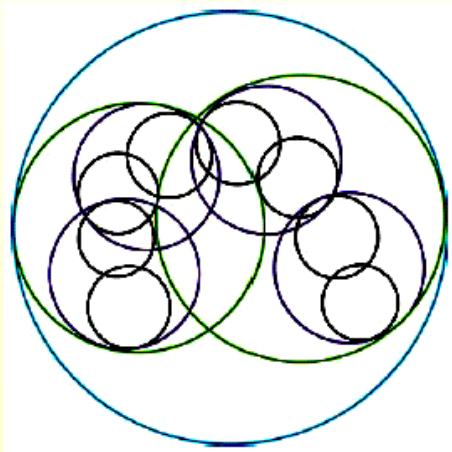
Traditional: Hashed Voxel Grids

- Partition space into a (possibly hierarchical) voxel grid and apportion the objects into cells of that grid
  - Use a hash table, since many cells will be empty
  - To find the neighbors of a given object, search nearby cells
- Requires re-apportioning all the objects into cells after each simulation time step
  - Bad when system is moving but not deforming much
- Efficiency highly dependent on voxel grid size chosen



Traditional: Bounding Volume Hierarchies

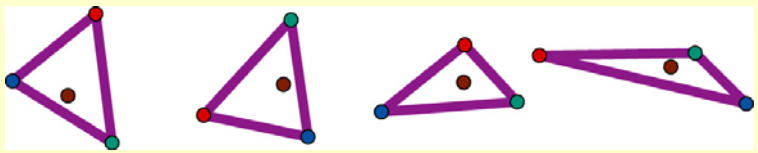
- Bounding volume hierarchies (BVH), using spheres, bounding boxes, etc., have been successfully used for collision checking of rigid objects
- Rigid bounding volume hierarchies are good for rigid objects, but
  - Use a fair amount of space
  - Collision checking requires a hierarchy traversal
  - Hierarchy must be updated when deformation occurs



A Small Detour: Kinetic Data Structures

Kinetic Data Structures (KDS) are used to track geometric attributes in a system of moving or deforming geometric objects in space. A KDS maintains an assertion cache that facilitates or trivializes the computation of the attribute.

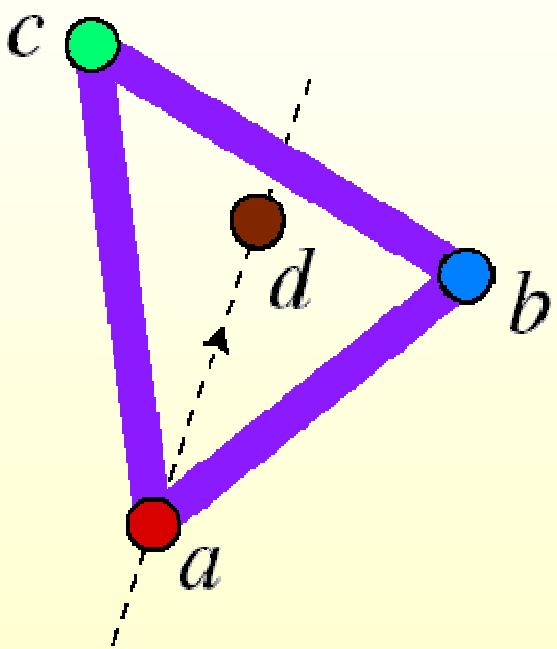
- A KDS for an attribute of interest is an easily repairable set of elementary relations (the certificates of the assertion cache) that allows an easy (re-)computation of the attribute of interest
- At each certificate failure, the KDS procedure repairs the assertion set and updates the attribute value
- At all times, the certificates mathematically prove the validity of the attribute computation



Motion Models

- The certificate failures must be either detected or predicted. Prediction is possible if the objects follow known motion laws.
- Between certificate failures the fundamental structure of the attribute of interest cannot change.
- An event queue of future certificate failures can be used when short term motion prediction is possible.
- However, at any moment, the motion law of an object may change. Then the failure times of all certificates involving that object must be updated.

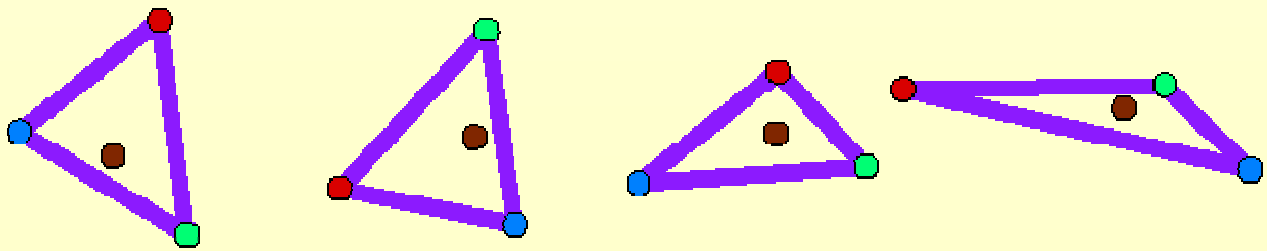
Convex Hull of Four Points



Proof of correctness:

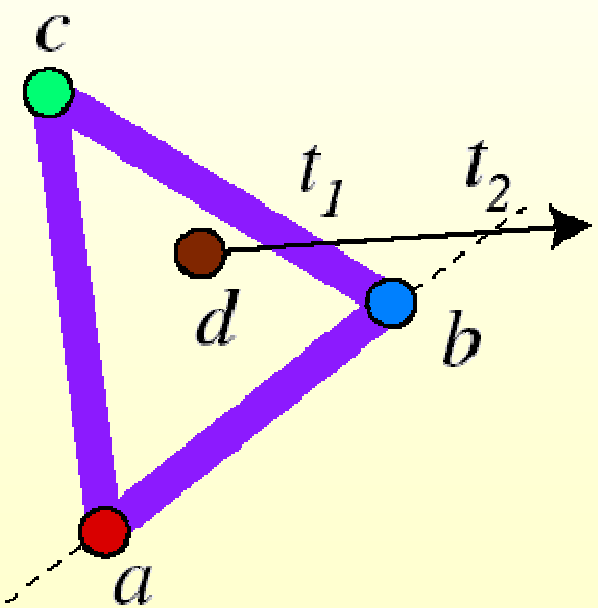
- a is to the left of bc
- d is to the left of bc
- b is to the right of ad
- c is to the left of ad

Four sidedness certificates.



Failure Times and the Event Queue

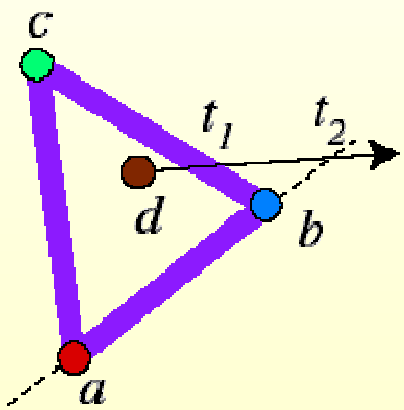
- Failure time of each certificate



Certificate	Failure time
a left of bc	never
d left of bc	t_1
b right of ad	t_2
c left of ad	never

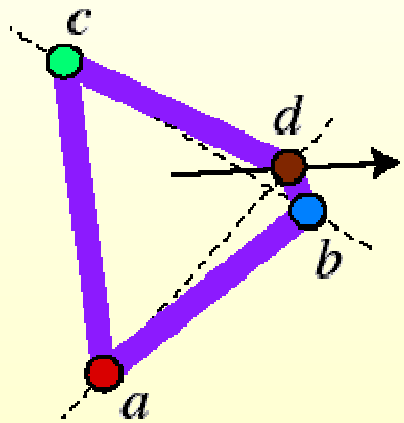
- Put the certificates in an event queue

Processing an Event



Old proof	New proof
a left of bc	a left of bc
d left of bc	d right of bc
b right of ad	b right of ad
c left of ad	c left of ad

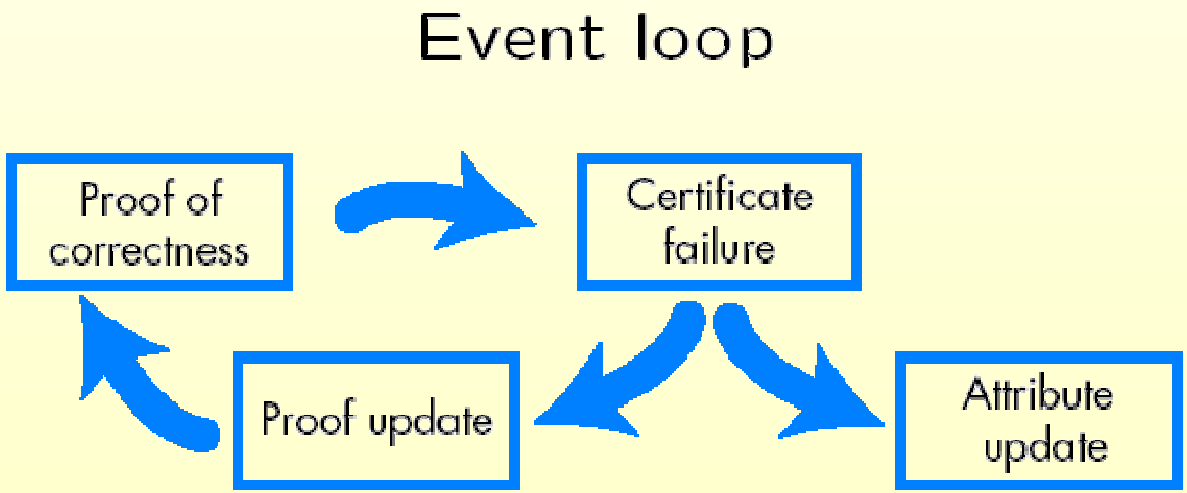
Proof update



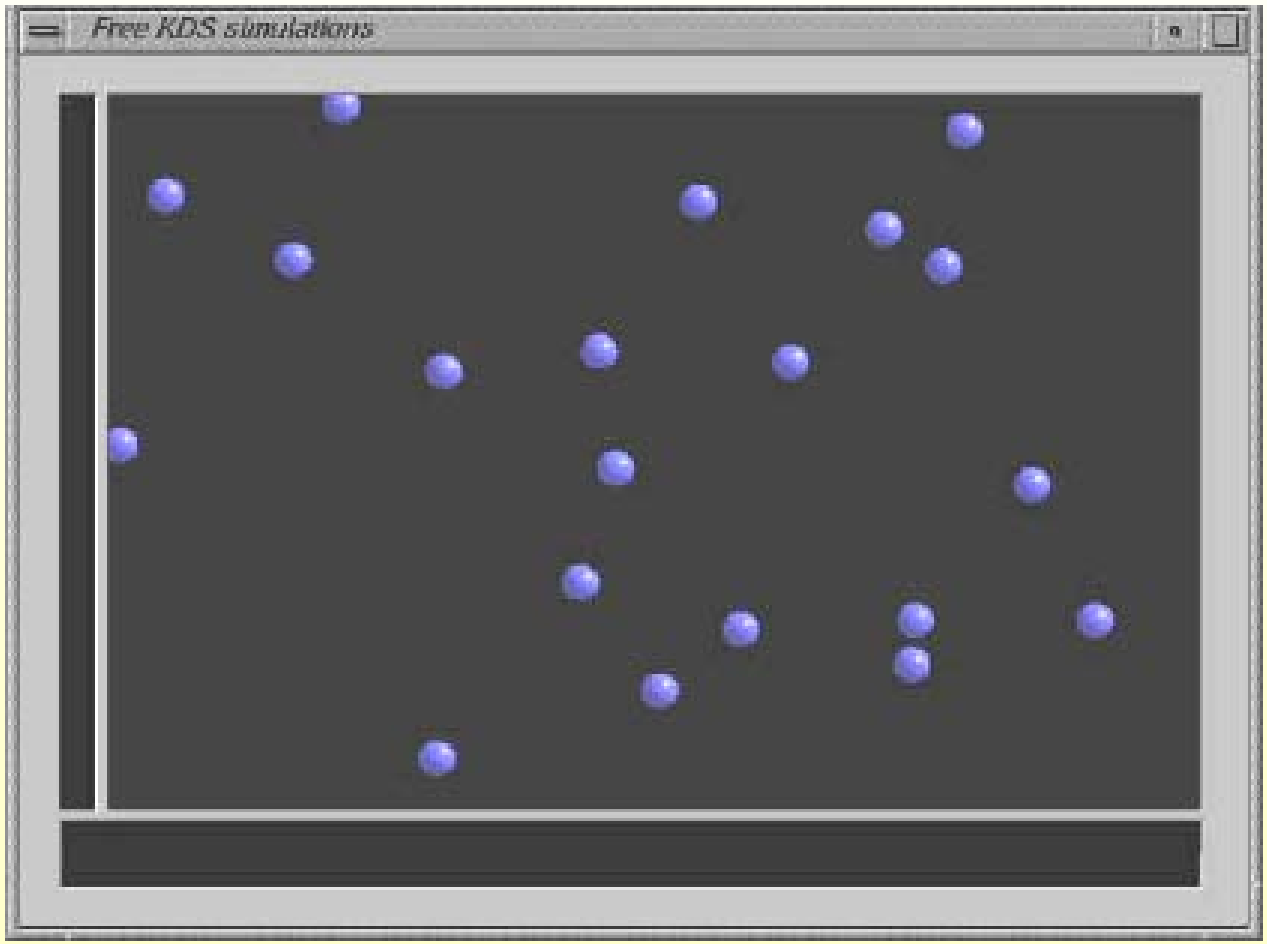
The Eternal KDS Loop

Two structures:

- Proof of correctness (based on certificates),
- Priority queue (sorted by failure time).



A Kinetic CH Animation

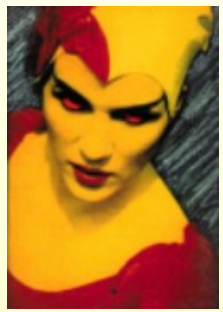


Choosing Which Relations to Track

We need to choose sets of elementary relations to track that:

- vary smoothly and stably as the objects move
- always make the computation of the attribute of interest fast

A Faustian trade-off: *The more we know about the world, the easier it will be to repair the attribute certification. But the more assertions about the world we maintain, the more certificate failures we will have to process.*



Drum hab ich mich der Magie ergeben,
Ob mir durch Geistes Kraft und Mund
Nicht manch Geheimnis würde kund;
Daß ich nicht mehr mit saurem Schweiß
Zu sagen brauche, was ich nicht weiß;
Daß ich erkenne, was die Welt
Im Innersten zusammenhält,
Schau alle Wirkenskraft und Samen,
Und tu nicht mehr in Worten kramen.

KDS Quality Measures

KDSs have a number of associated quality measures:

- Responsiveness: at each certificate failure, the repair cost for the certificate set and the attribute computation is small
- Efficiency: the worst-case number of events to be processed, over a reasonable class of motions, is comparable to the number of combinatorial changes in the attribute of interest
- Compactness: the size of the certificate set maintained is small
- Locality: motion plan updates are inexpensive

The Interface Between KDS and Motion

In the classical KDS setting, objects move according to posted flight plans with "closed-form" motion descriptions (e.g., polynomial trajectories).

Certificates are typically elementary algebraic relations; thus the KDS itself can calculate the certificate failure times and insert those into an event queue.

This limits the applicability of the KDS framework, because

- In physical simulations elements are moved by an ODE or PDE integrator – there are no explicit motion plans
- In real-world settings certificate failures have to be sensed by sensors – there are no motion plans at all ...

KDSs in Physical Simulation

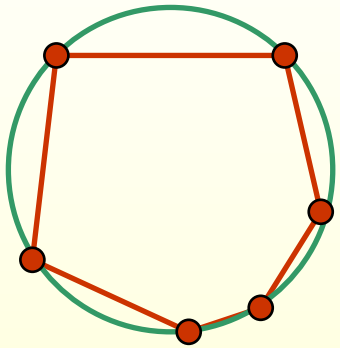
- Solving differential-algebraic inequalities is hard
- But, we can control the integrator time-step size (and therefore element displacements)
- Key geometric problem:

> **Find ways to efficiently repair geometric structures after small motions of their defining elements**

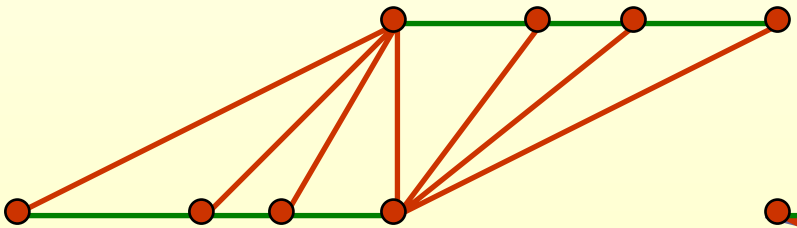
- We need to deal with
  - multiple certificate failures
  - no knowledge of what transpired between system snapshots
- Example: **a deformable spanner**

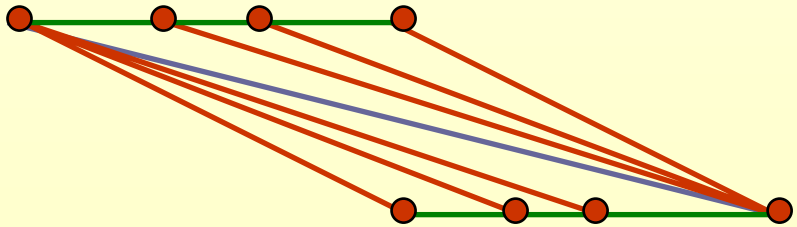
Some Sombering Delaunay Thoughts

...



- Small motions can cause large combinatorial changes
- A small number of certificate failures may require a high repair cost
- Canonical structures not desirable ...

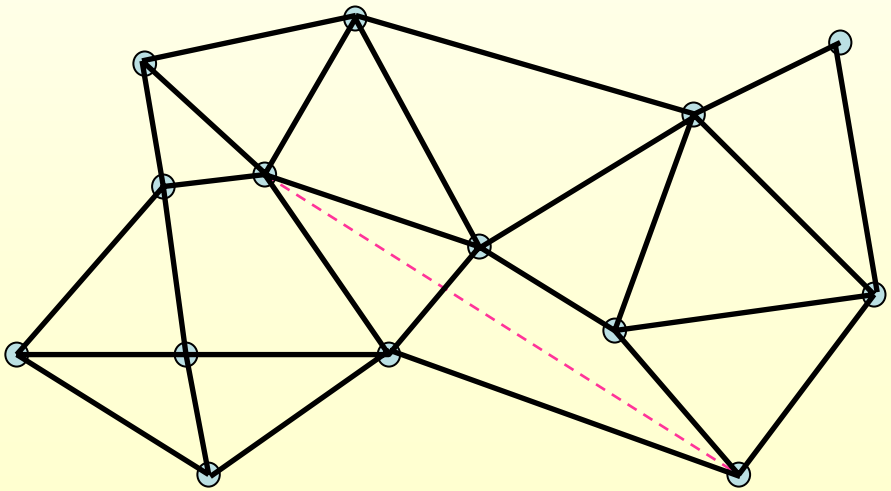




Geometric Spanners

An -spanner is a sparse subgraph G′ of a graph G such that the shortest path distance in G′ is at most times that of G.

- Geometry setting: Approximate all distances between points.



aspect ratio (G) =

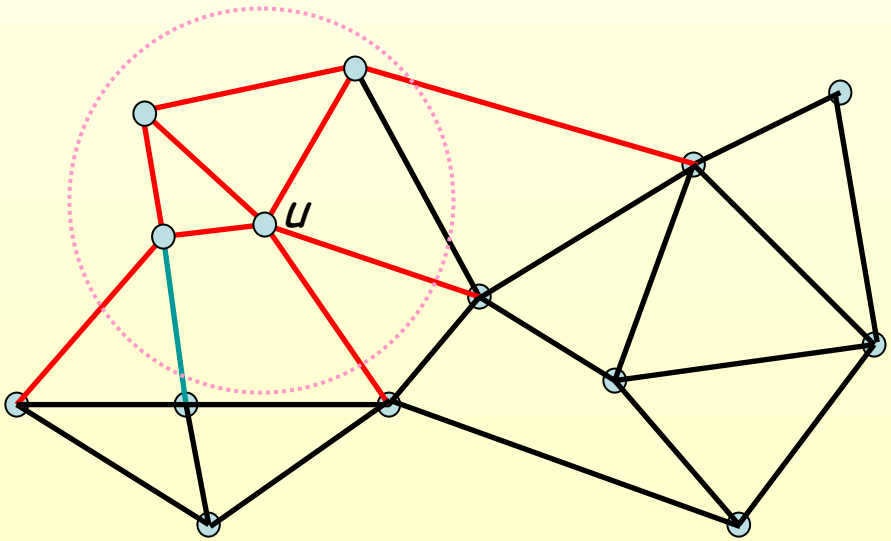
$$\frac{\text{longest pairwise distance}}{\text{shortest pairwise distance}} = \text{poly}(n).$$

spanning ratio (G,G')

- To come: a kinetic (1+ ε)-spanner with O(n) edges total, O(log n) edges per node.

Proximity Queries Using a Spanner

- To find all neighbors of a node *u* within Euclidean distance *d*, just run a breadth-first search from *u* up to distance *d*.



Replace a continuous geometric search by a more efficient graph search.

Spanner Construction

Discrete centers with radius r: A sample of the nodes s.t.

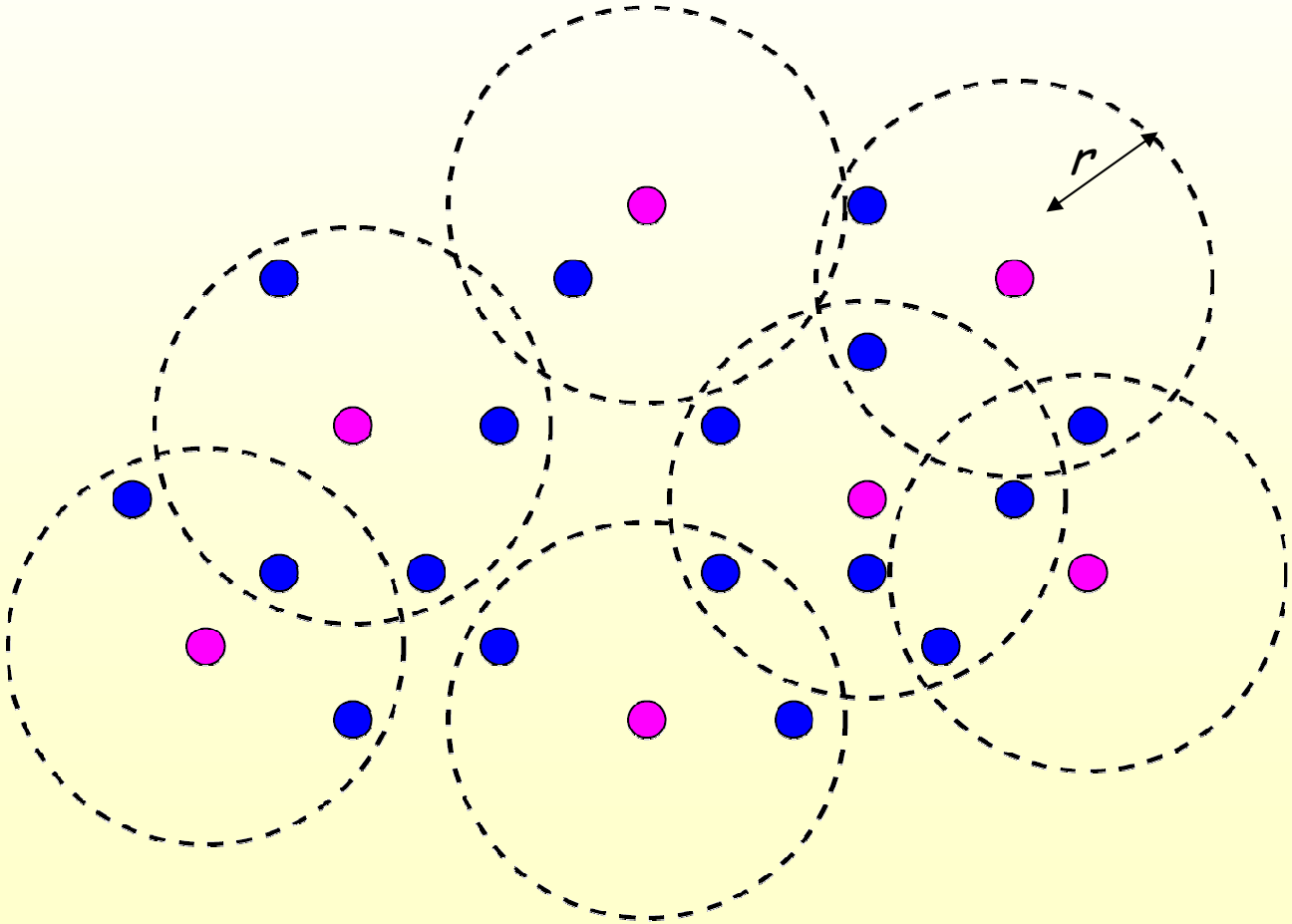
- Every node is covered by at least one center;
- No two centers are covered by each other.

The spanner: a graph G built by

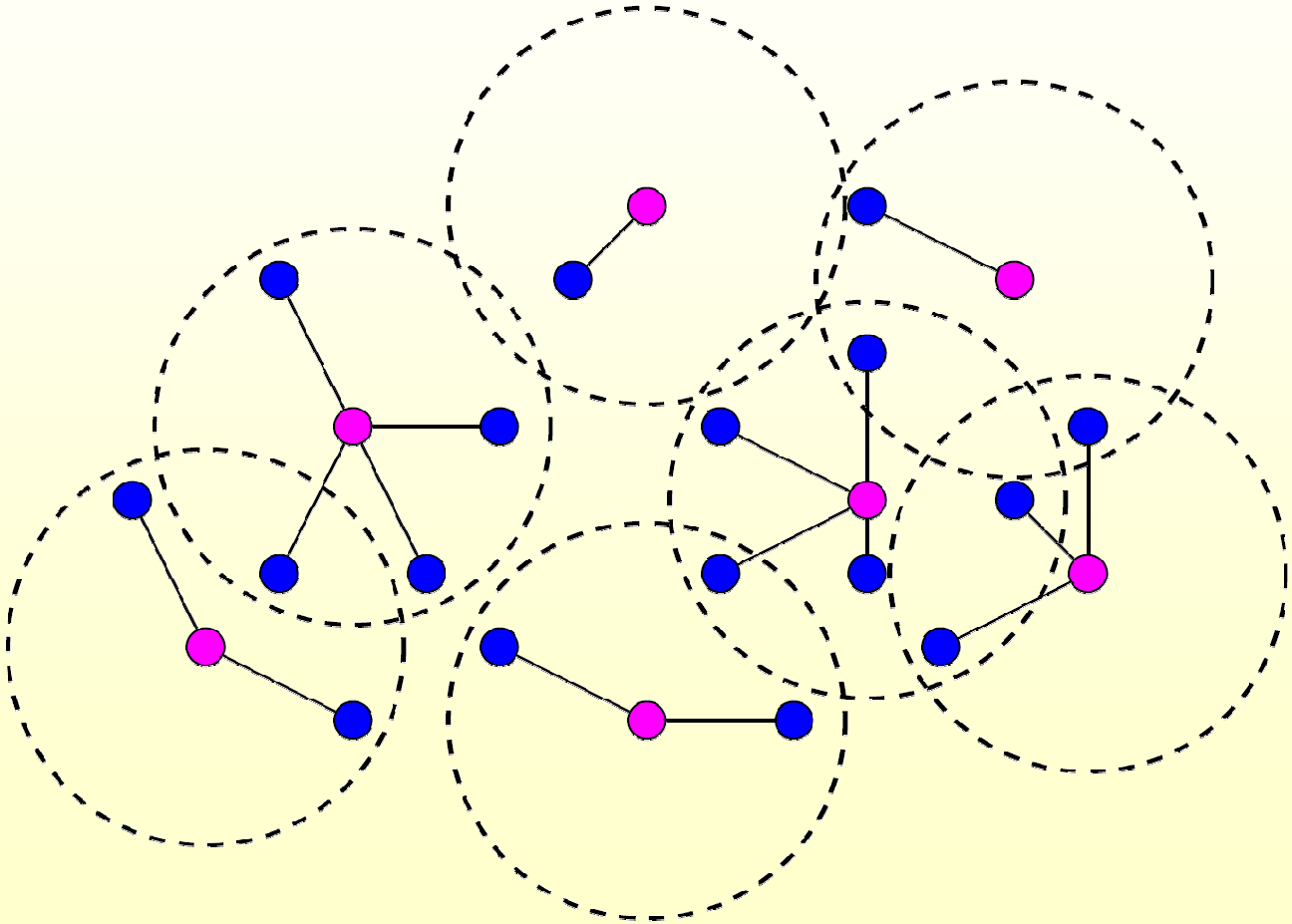
1. Constructing a hierarchy of discrete centers and taking parent-child edges.
   - Level i has radius 2^i.
2. Adding all edges with length $c \cdot 2^i$ within each level.
   - $c = 4+16/\varepsilon$.

Theorem: Graph G is a $(1+\varepsilon)$-spanner.

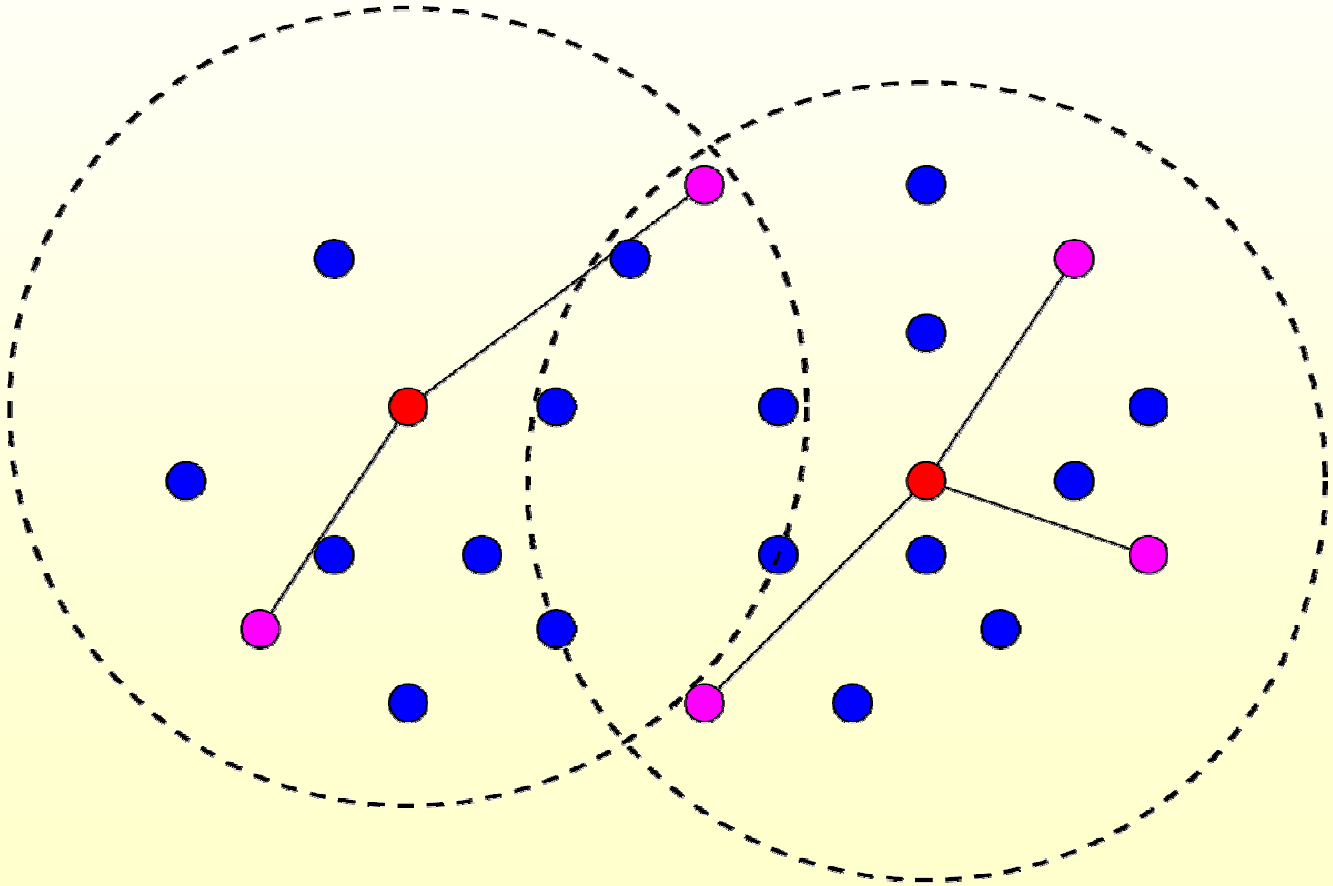
1st Level Centers



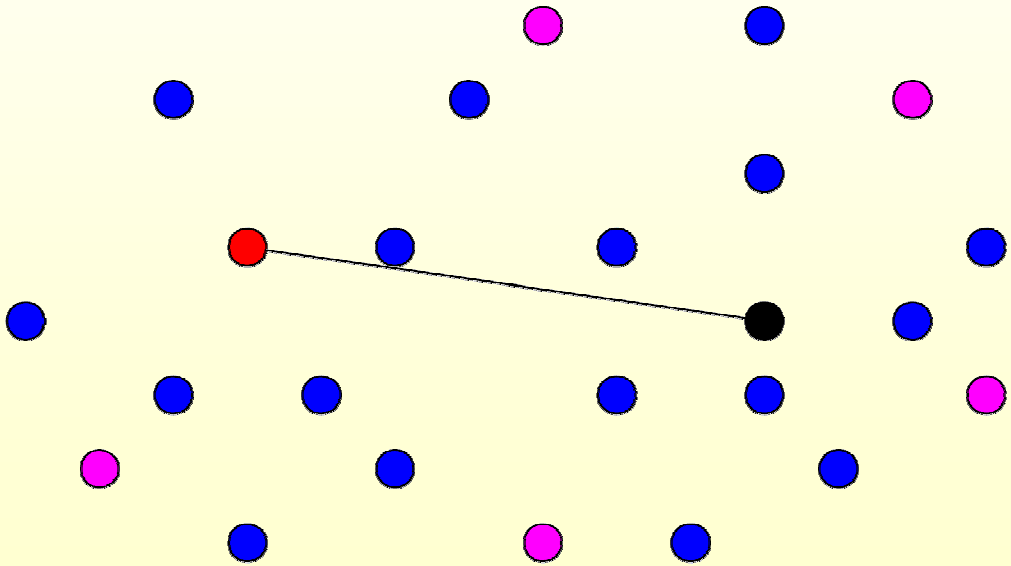
Parent-Child Relationships



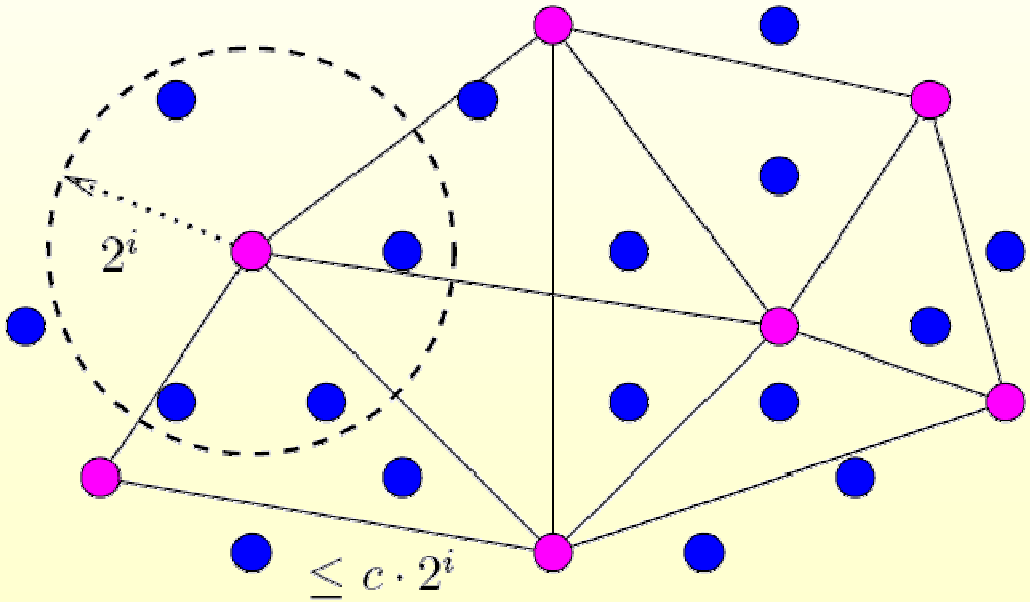
2nd Level Centers



3rd Level Centers

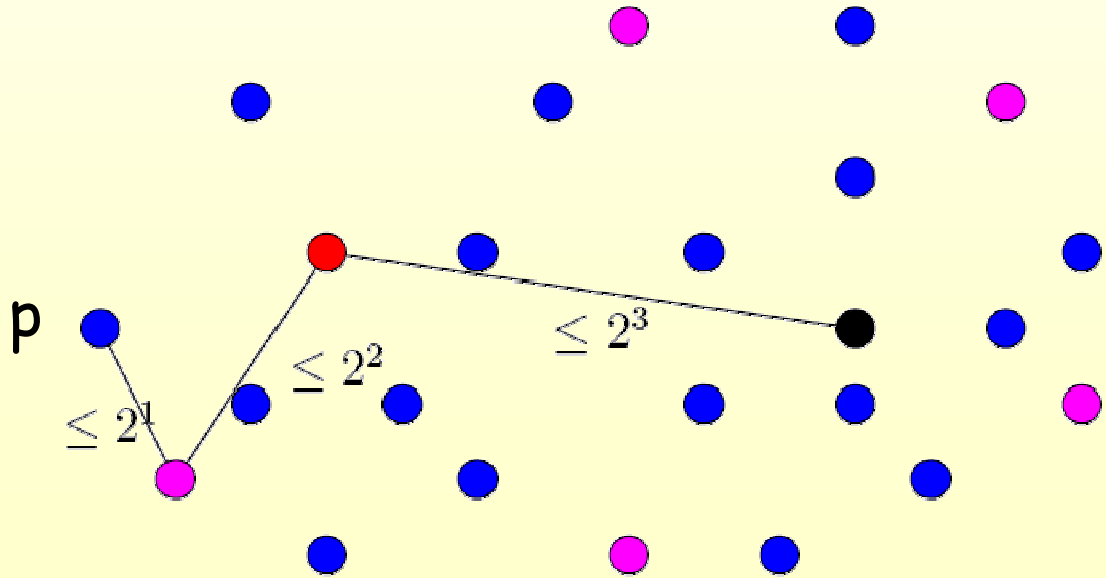


2nd Level Spanner Edges



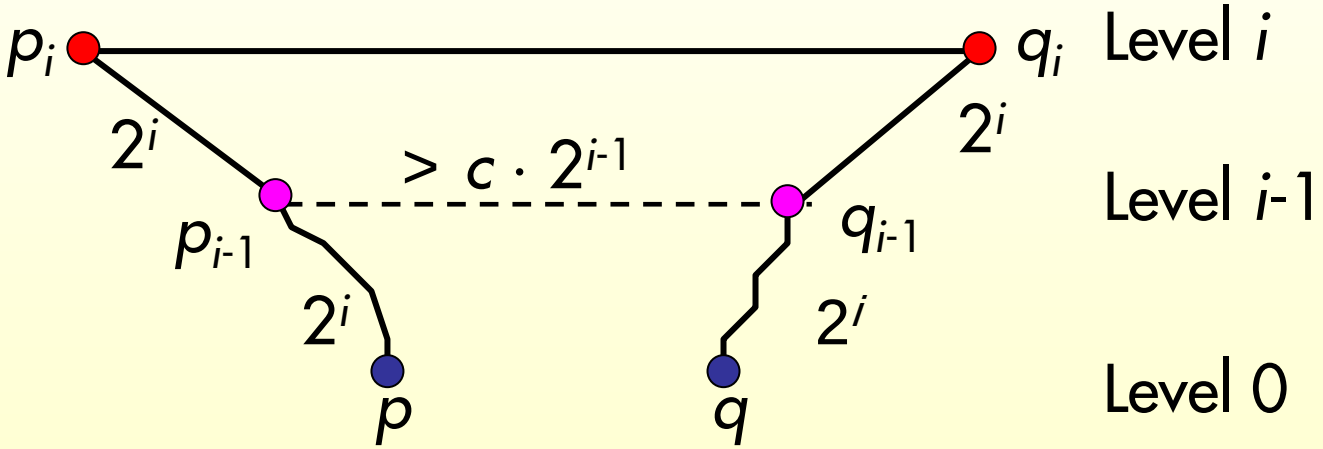
Parent Chain

- All parent-child edges are in *G*.
- A node *p* is connected to its ancestor at level *i* by the parent chain whose total length is 2^{i+1}.



Spanner Theorem

- G is a $(1+\varepsilon)$-spanner, where $\varepsilon = 16/(c-4)$.

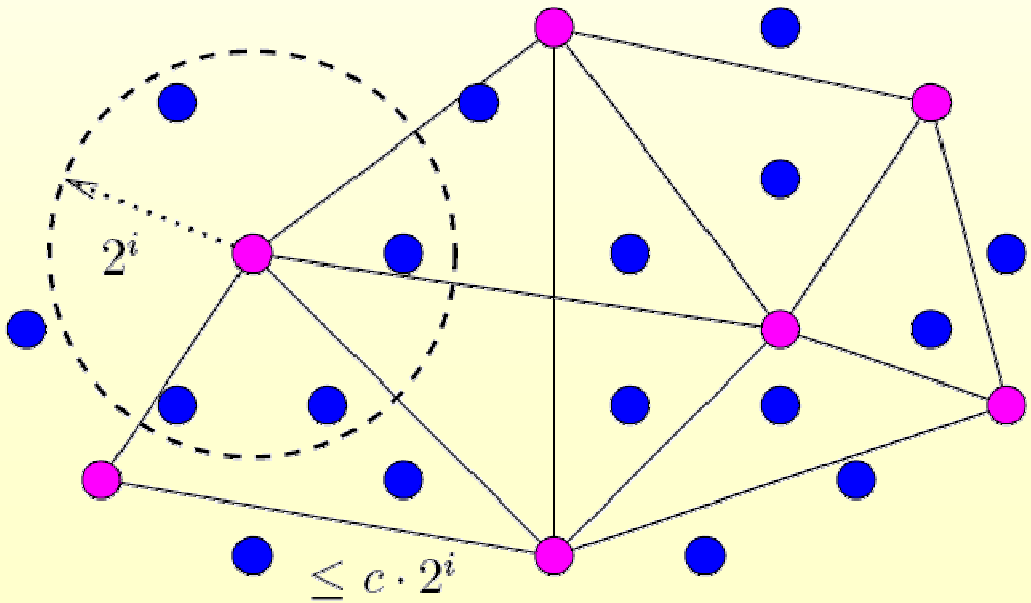


$$|pq| > (c-4)\,2^{i-1}$$

$$\begin{aligned} \text{path}(p,q) \quad & |p_i q_i| + 4 \cdot 2^i \\ & |pq| + 8 \cdot 2^i \\ & < (1+\varepsilon)\,|pq| \end{aligned}$$

Spanner Quality

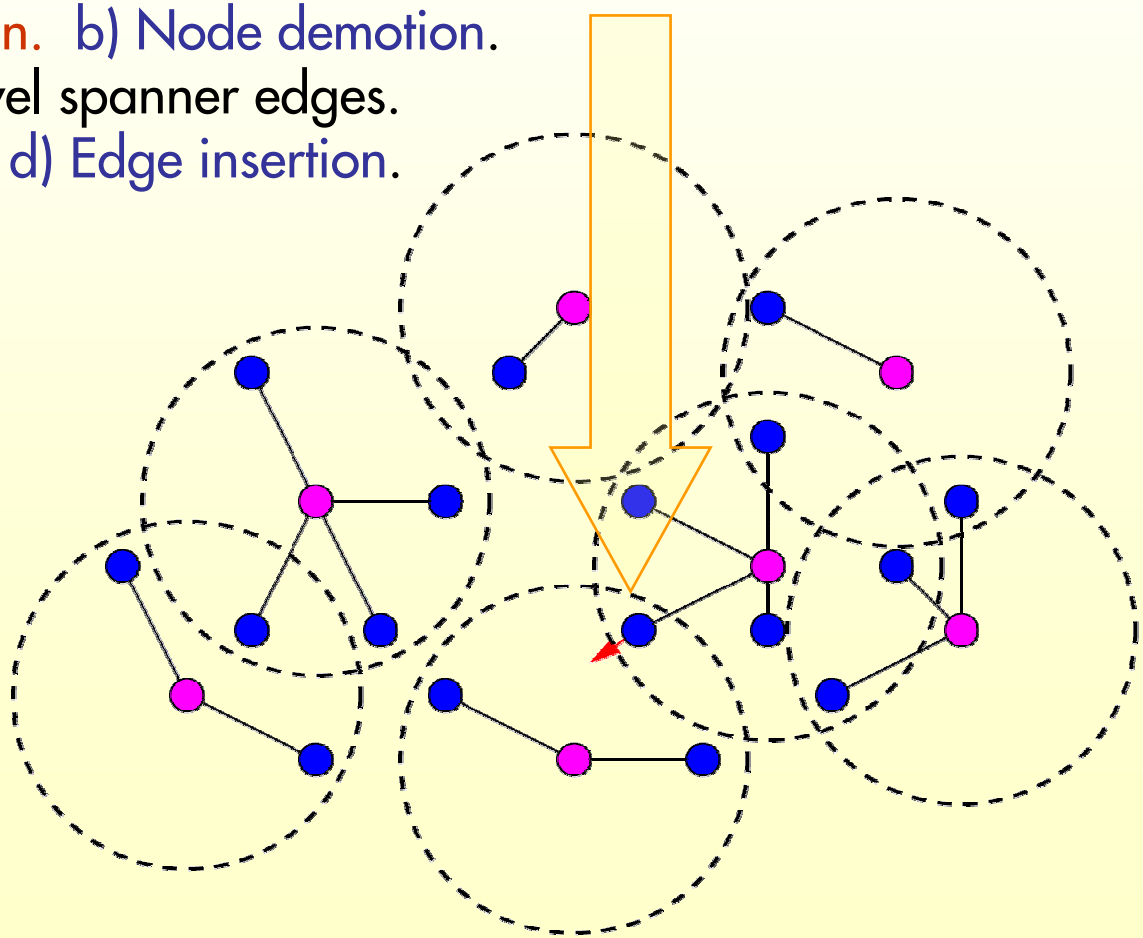
- The hierarchy has log = O(log *n*) levels.
  - Linear size: O(*n*) edges;
  - Small degree: max degree O(log *n*) per node.



Maintenance Under Node Motion

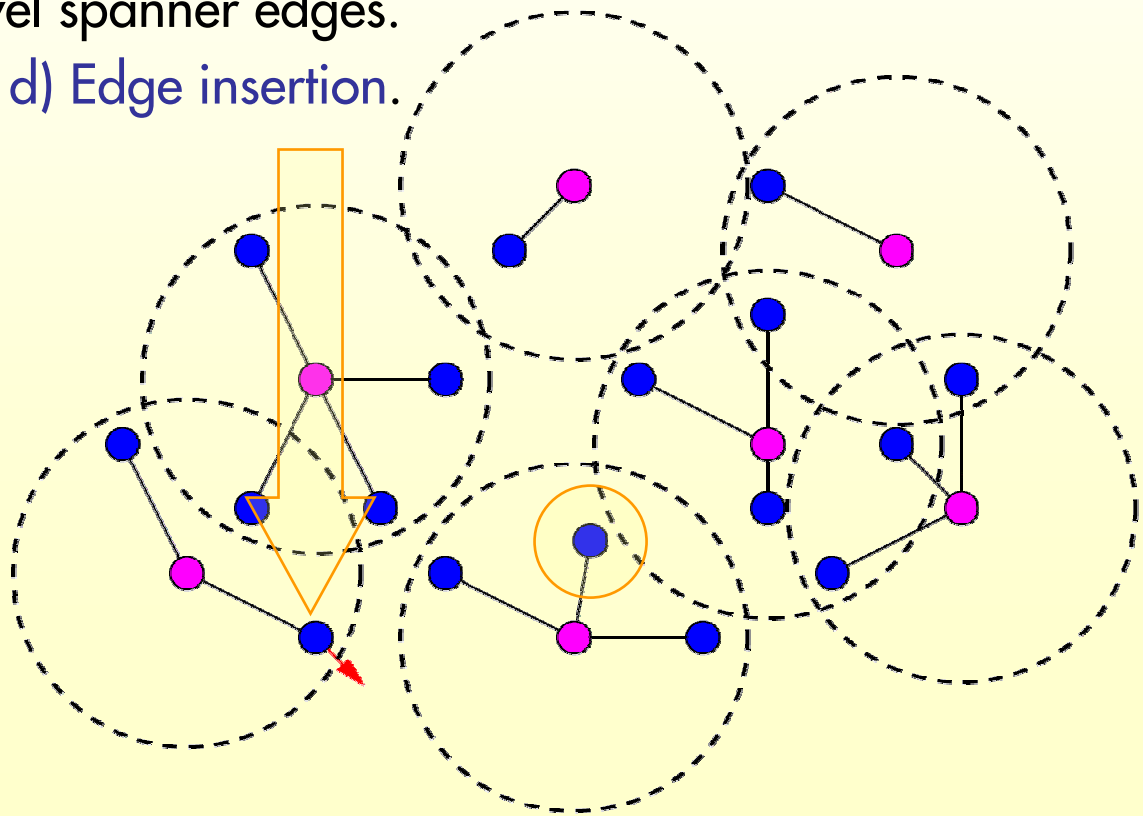
1. Maintain parent-child edges (the discrete centers hierarchy).
   a) Node promotion. b) Node demotion.
2. Maintain within level spanner edges.
   c) Edge deletion. d) Edge insertion.

Only distance certificates get used



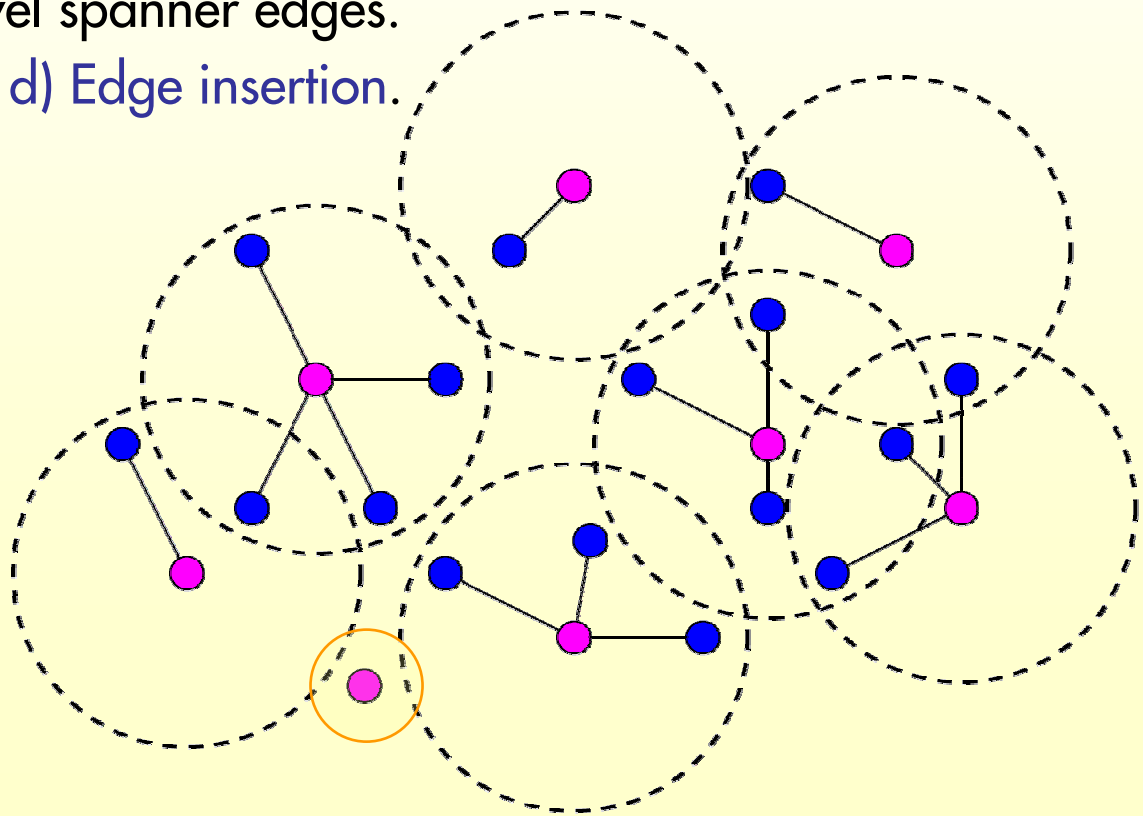
Maintenance Under Node Motion

1. Maintain parent-child edges.
   a) Node promotion. b) Node demotion.
2. Maintain within level spanner edges.
   c) Edge deletion. d) Edge insertion.



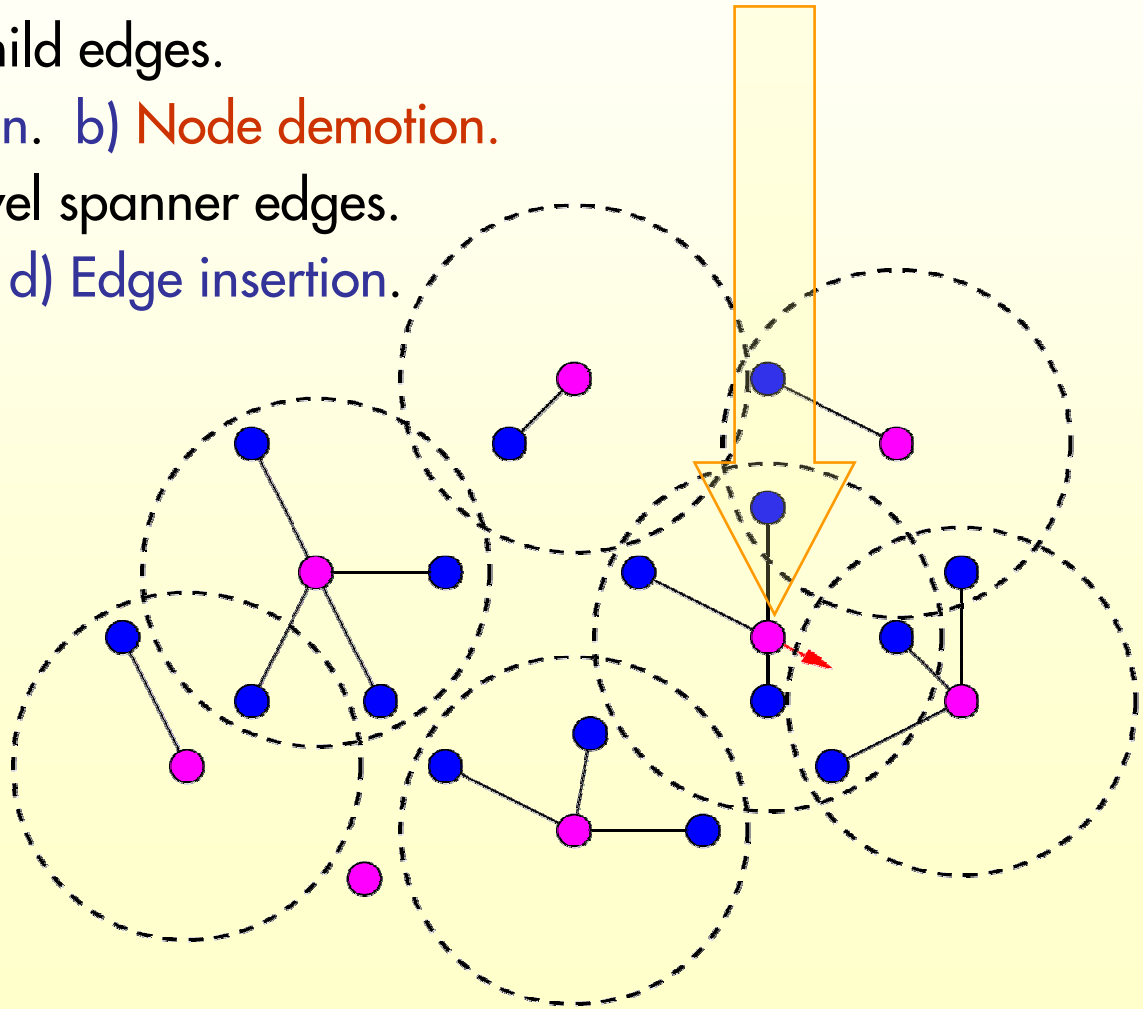
Maintenance Under Node Motion

1. Maintain parent-child edges.
   a) Node promotion. b) Node demotion.
2. Maintain within level spanner edges.
   c) Edge deletion. d) Edge insertion.



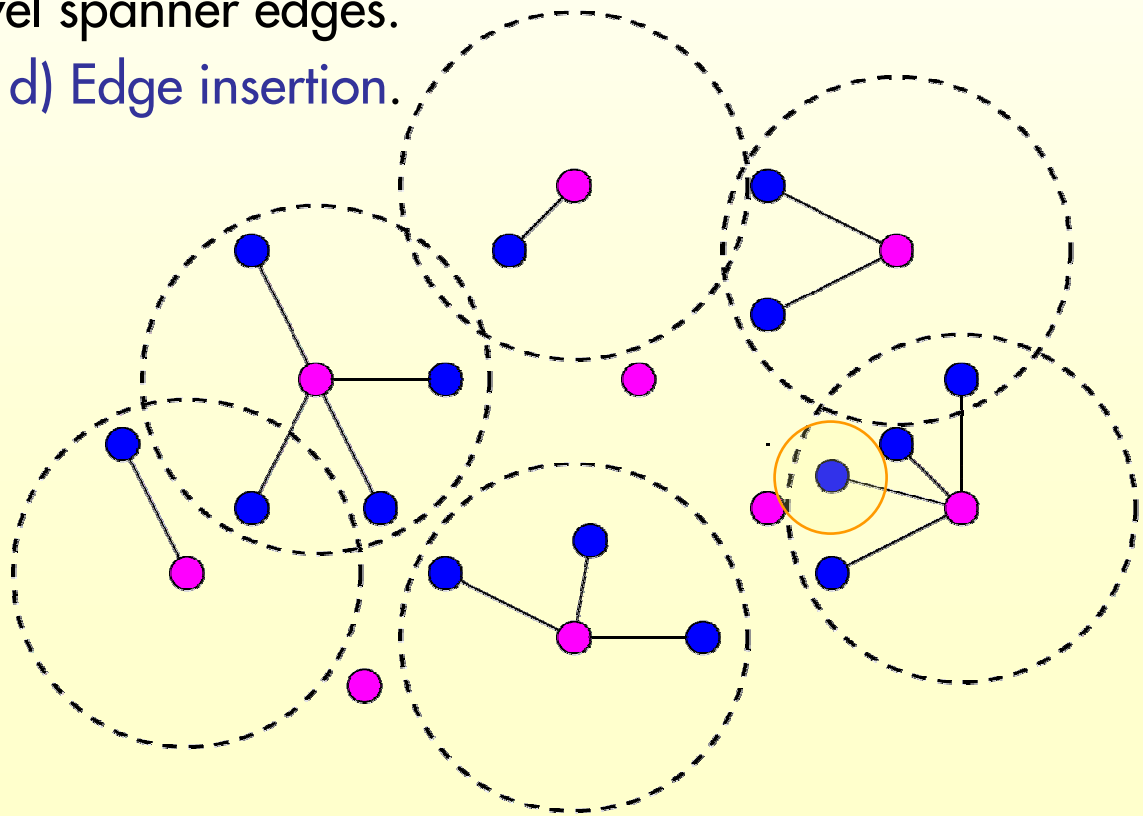
Maintenance Under Node Motion

1. Maintain parent-child edges.
   a) Node promotion. b) Node demotion.
2. Maintain within level spanner edges.
   c) Edge deletion. d) Edge insertion.



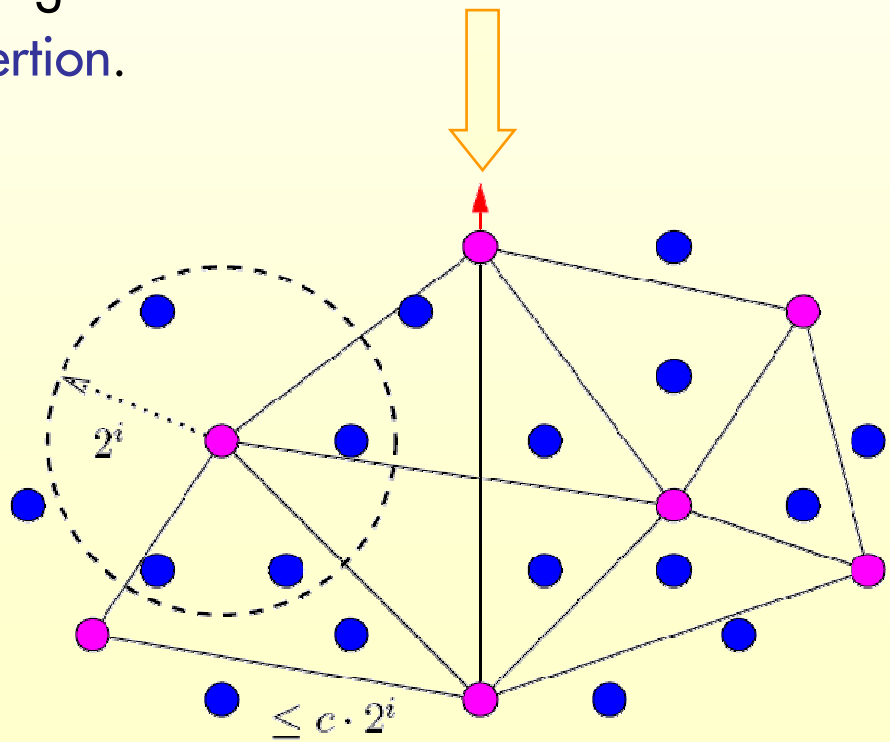
Maintenance Under Node Motion

1. Maintain parent-child edges.
   a) Node promotion. b) Node demotion.
2. Maintain within level spanner edges.
   c) Edge deletion. d) Edge insertion.



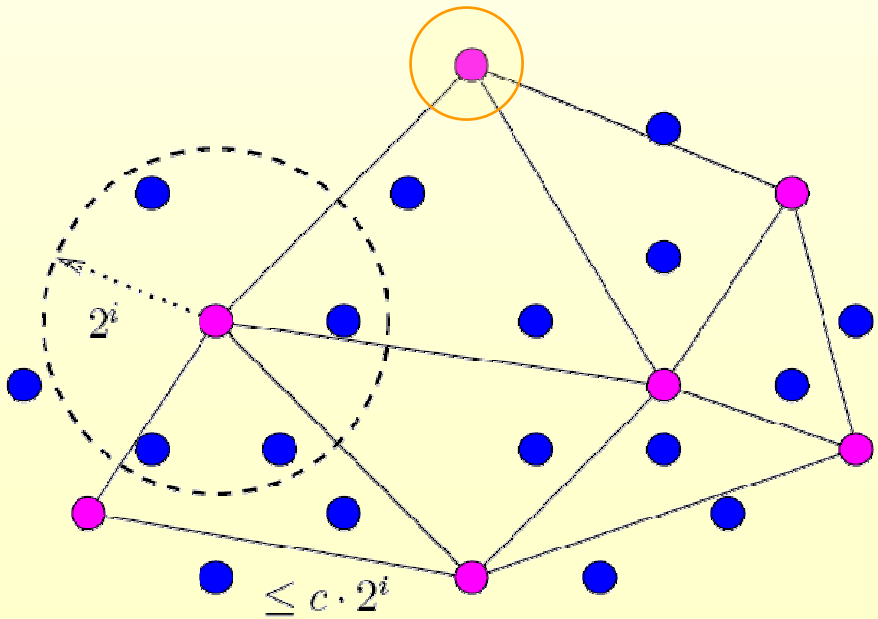
Maintenance Under Node Motion

1. Maintain parent-child edges.
   a) Node promotion. b) Node demotion.
2. Maintain within level spanner edges.
   c) Edge deletion. d) Edge insertion.



Maintenance Under Node Motion

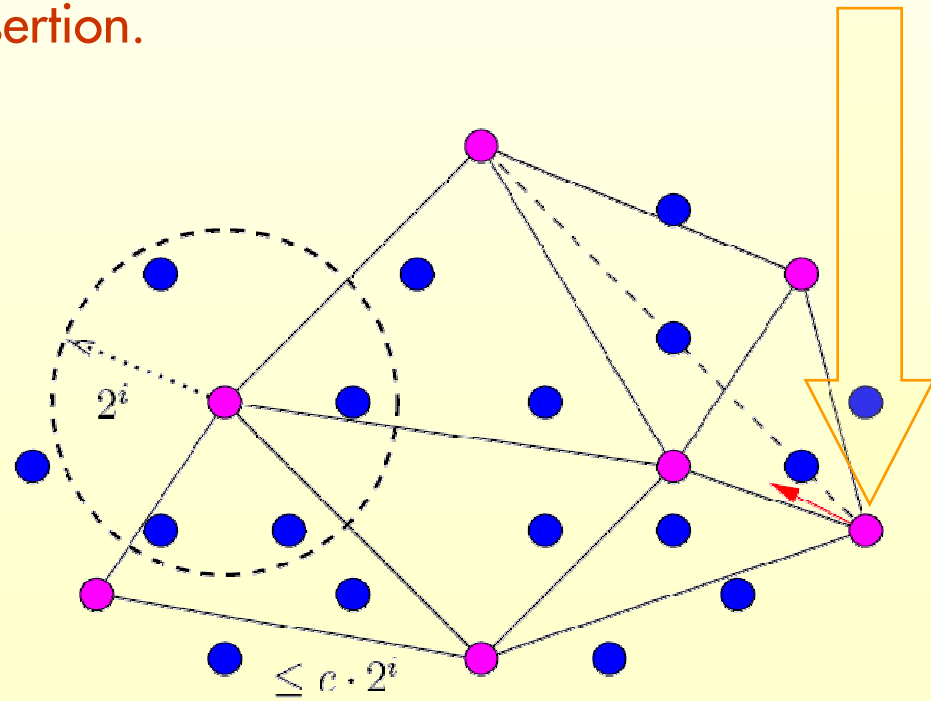
1. Maintain parent-child edges.
   a) Node promotion. b) Node demotion.
2. Maintain within level spanner edges.
   c) Edge deletion. d) Edge insertion.



Maintenance Under Node Motion

1. Maintain parent-child edges.
   a) Node promotion. b) Node demotion.
2. Maintain within level spanner edges.
   c) Edge deletion. d) Edge insertion.

This is the difficult case – each node can have only a few neighbors; losing is easy to know about, but gaining neighbors is hard.

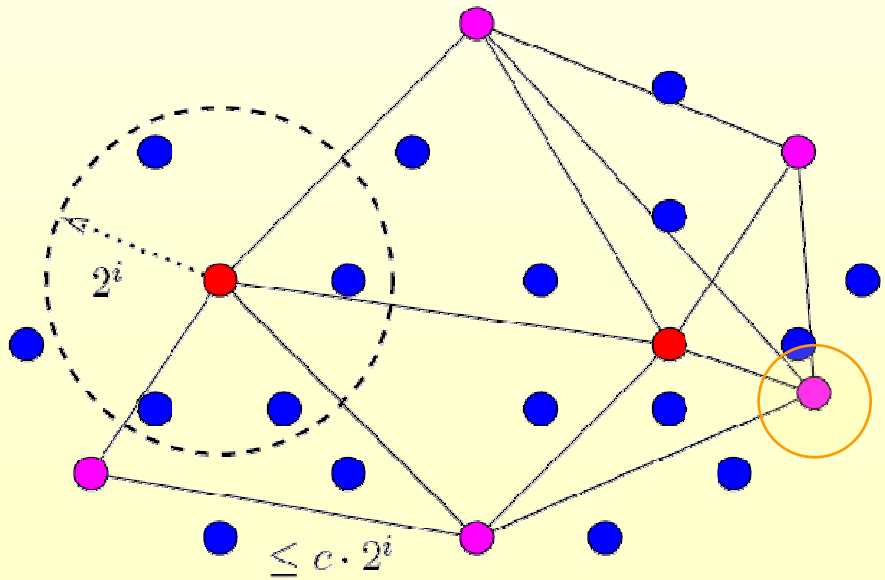


Maintenance Under Node Motion

1. Maintain parent-child edges.
   a) Node promotion. b) Node demotion.
2. Maintain within level spanner edges.
   c) Edge deletion. d) Edge insertion.

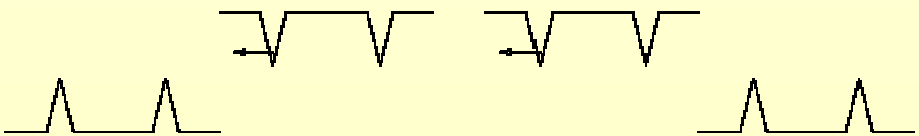
Sufficient to consider only "cousin" pairs.

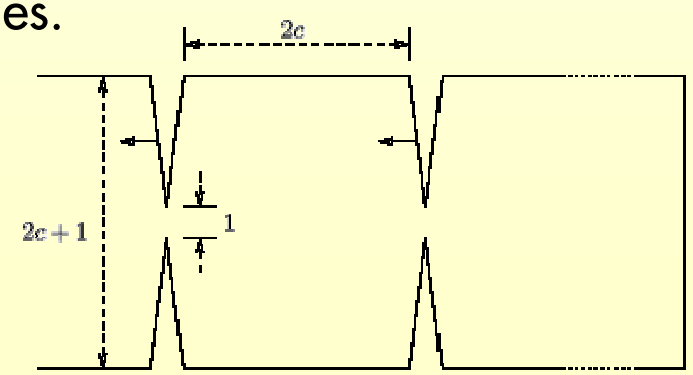
Multiresolution to the rescue!



Flexibility and Stability

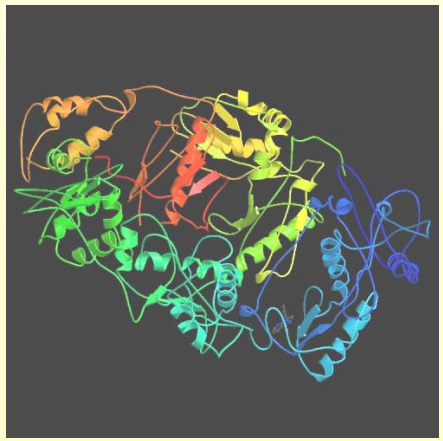
- Dynamic environment
  - Node insertion and deletion: O(log n) time (Do you know about *skip lists*?)
  - Initial construction: O(n log n) time.
- Stability under motion
  - The spanner can be provably repaired, as long as no node at level i moves more than $(c - 4)\ 2^i$-1 in each time step.
  - In the worst-case, any α-spanner changes (n^2/α^2) times under certain smooth algebraic motions.
  - This spanner changes $O(n^2 \log n)$ times.



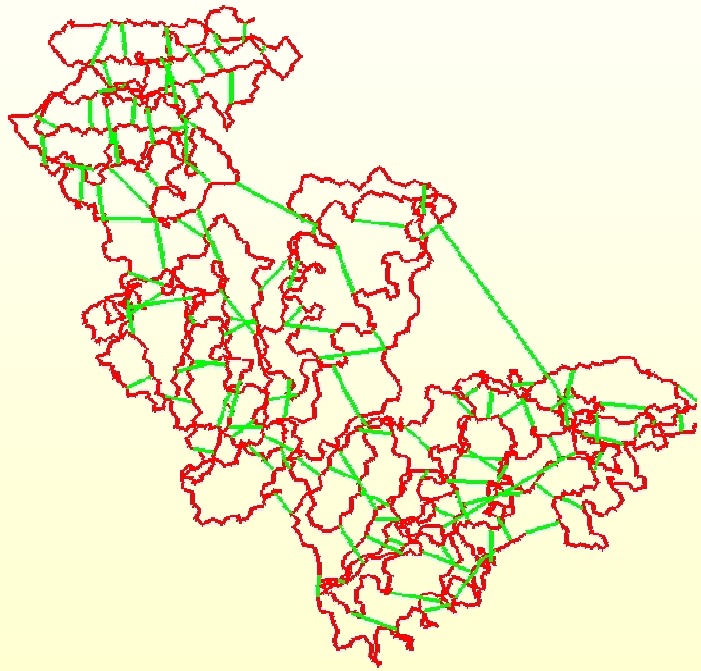


Spanners for Physical Objects

Molecular dynamics: Add a sparse set of shortcuts, sufficient to guarantee the spanning property for a protein backbone.



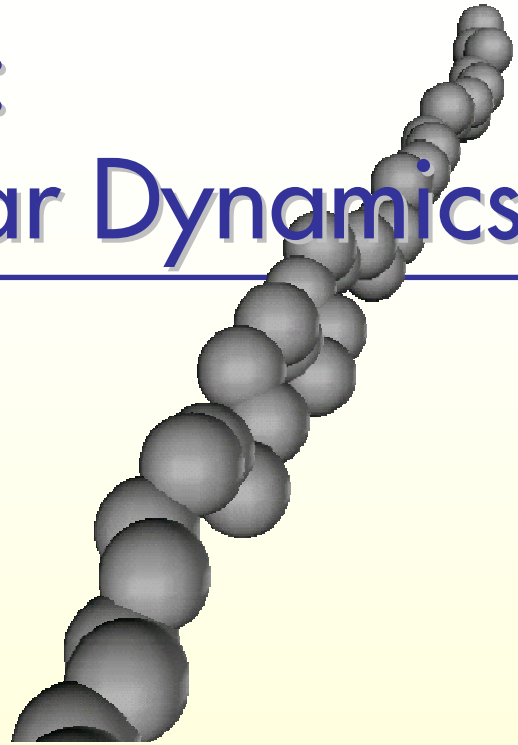
3HVT



A protein example with α = 3

Spanner Performance under Motion: Practice – Molecular Dynamics

- Each frame corresponds to a hundred actual MD steps (Tinker data)
- Only 2-4% of spanner edges change between frames
- Spanner is quite stable, except for bottom-level edges (high frequency atomic vibrations)

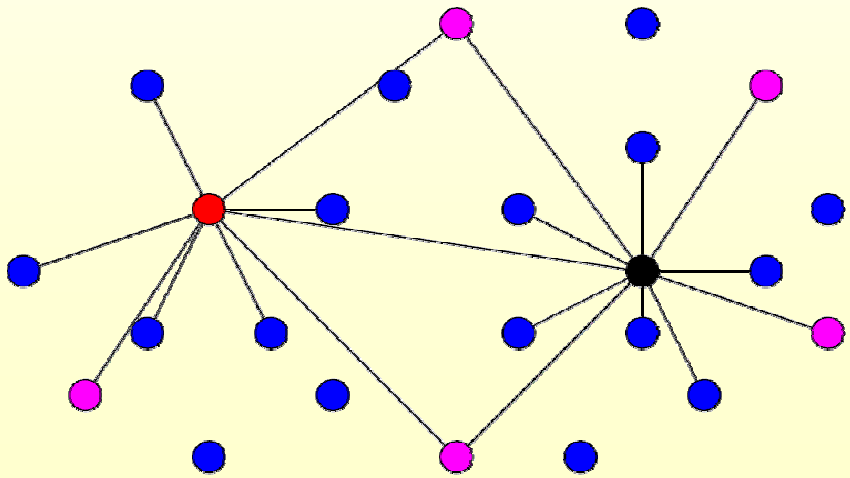


green = edge birth, red = edge death, gray = steady state

	Nodes	Edges	Time frames	Ave. promotions	Ave. demotions	Ave. add edge/	Ave. drop edge
Clone0	50	159-214	249	.22	.23	5.42	5.24
R7C27	110	376-506	33	.38	.59	17.7	13.7

Distributed Spanner Implementation

- Each node maintain its O(log *n*) spanner edges by communicating only with its spanner neighbors.
  - Load balancing.
  - Low communication cost.



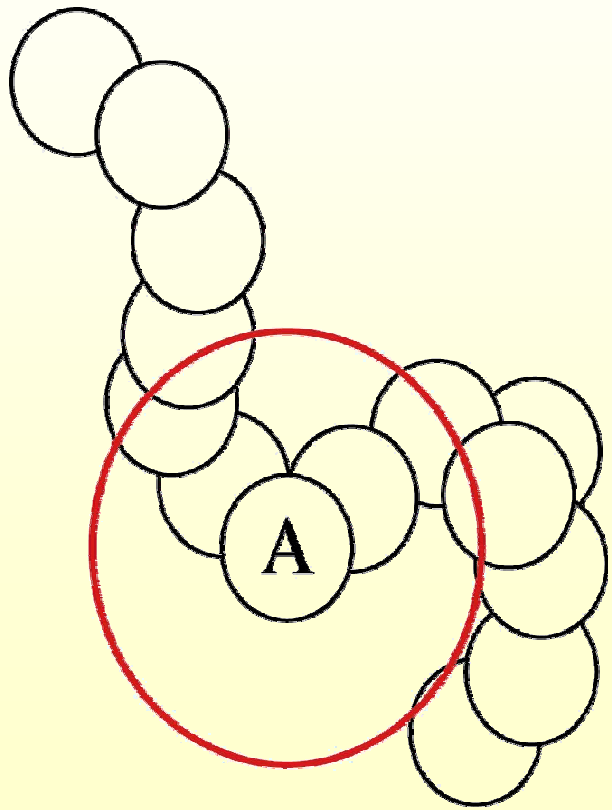
- The first truly distributed Kinetic Data Structure (dKDS)

Additional Spanner Goodies

1. Neighbor lists within distance *r*.
2. (1+ε)-approximate nearest neighbor query.
3. Well-separated pair decomposition.
4. Geometric *k*-center.

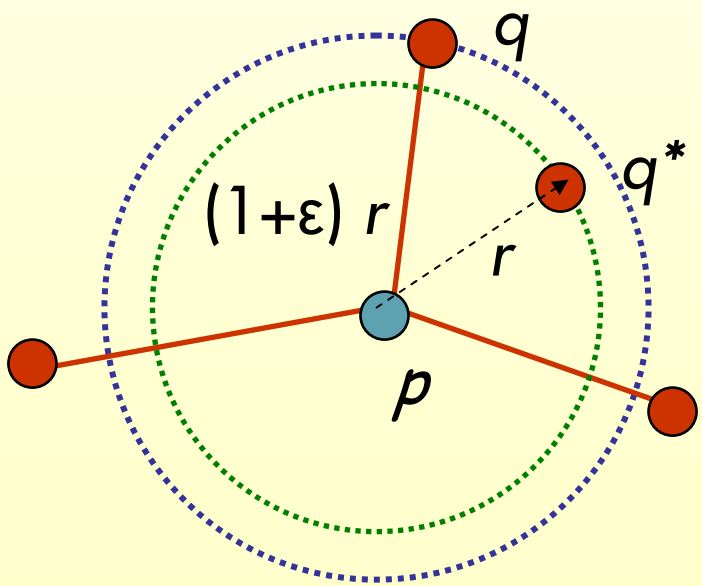
Neighbor Lists

- Output all pairs within distance *r*.
  - Walk on the spanner until the distance to *A* is (1+ ε)*r*.
  - Running time: $O(n + k)$, *k* is the size of the output.



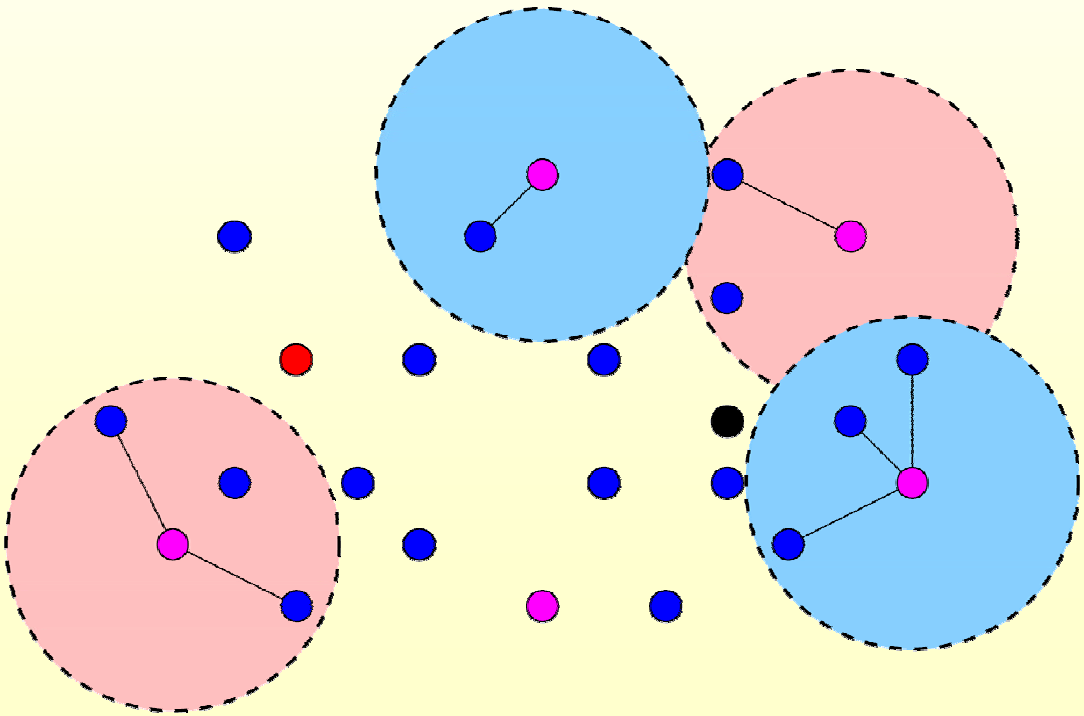
Approximate Nearest Neighbors

- Given any point p, return a node q s.t. $|pq| \leq (1+\varepsilon)|pq^*|$, where q^* is the nearest neighbor of p.
  - Insert p into the spanner, take its shortest edge.
  - Running time: $O(\log n)$.



Well-Separated Pair Decomposition

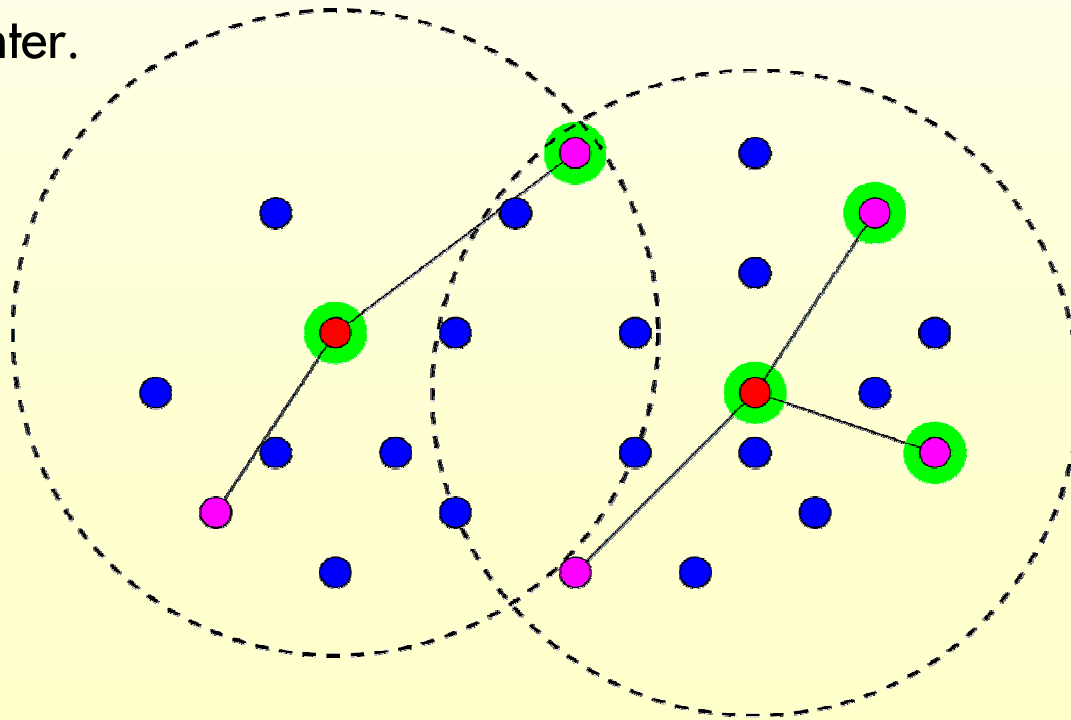
- Take every non-connected cousin pair
- Provides an N-body style approximation



but affixed to the points, not to the ambient space.

Geometric *k*-Centers

- Select a set $K \subseteq S$, assign each node of S to its nearest neighbor in K, so that the maximum radius is minimized.
- Take the lowest level i such that the number of discrete centers is less than k.
- R_i is 8-approximate k-center.



5-center

Higher Dimensions

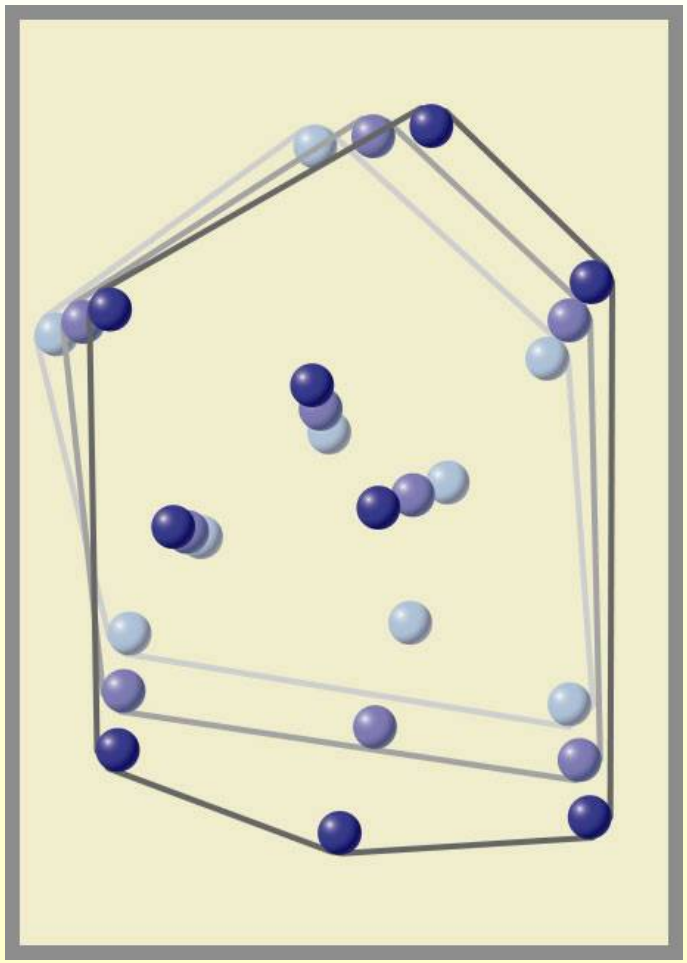
- In E^d, all the foregoing still works, but the storage and the query/update times now have a factor of $1/\ ^d$
- It still works in general metric spaces of bounded doubling dimension (so we can build spanners on collections of shapes, images, etc. – a poor man's manifold learning for sparse data …)

Spanner Summary

- "One-stop shopping" data structure for all proximity information
  - Lightweight and combinatorial
  - Stable under motion
  - Load-balanced
  - Distributed
  - Scalable
  - Provides hierarchical clustering

"Tutto cambia perchè nulla cambi"
T. di Lampedusa, *Il Gattopardo (1860+)*

Kinetic Data Structure Problems Are Fun



- On-line problems – deal with unknown future data
- Learn only what you need to answer the questions that you have to
- Proofs as active objects

KDS Application Taxonomy

Efficient KDSs have been developed for a variety of geometric problems:

- Extent Problems: convex hull, diameter, width for moving points
- Proximity Problems: closest pair, Voronoi/Delaunay diagrams
- Communication Problems: MST for geometric and general graphs, connectivity of moving rectangles and unit disks
- Visibility: BSPs and visibility orders
- Tilings and Triangulations: pseudo-triangulations and triangulations for moving points and polygons
- Collision Detection: for rigid and deformable objects