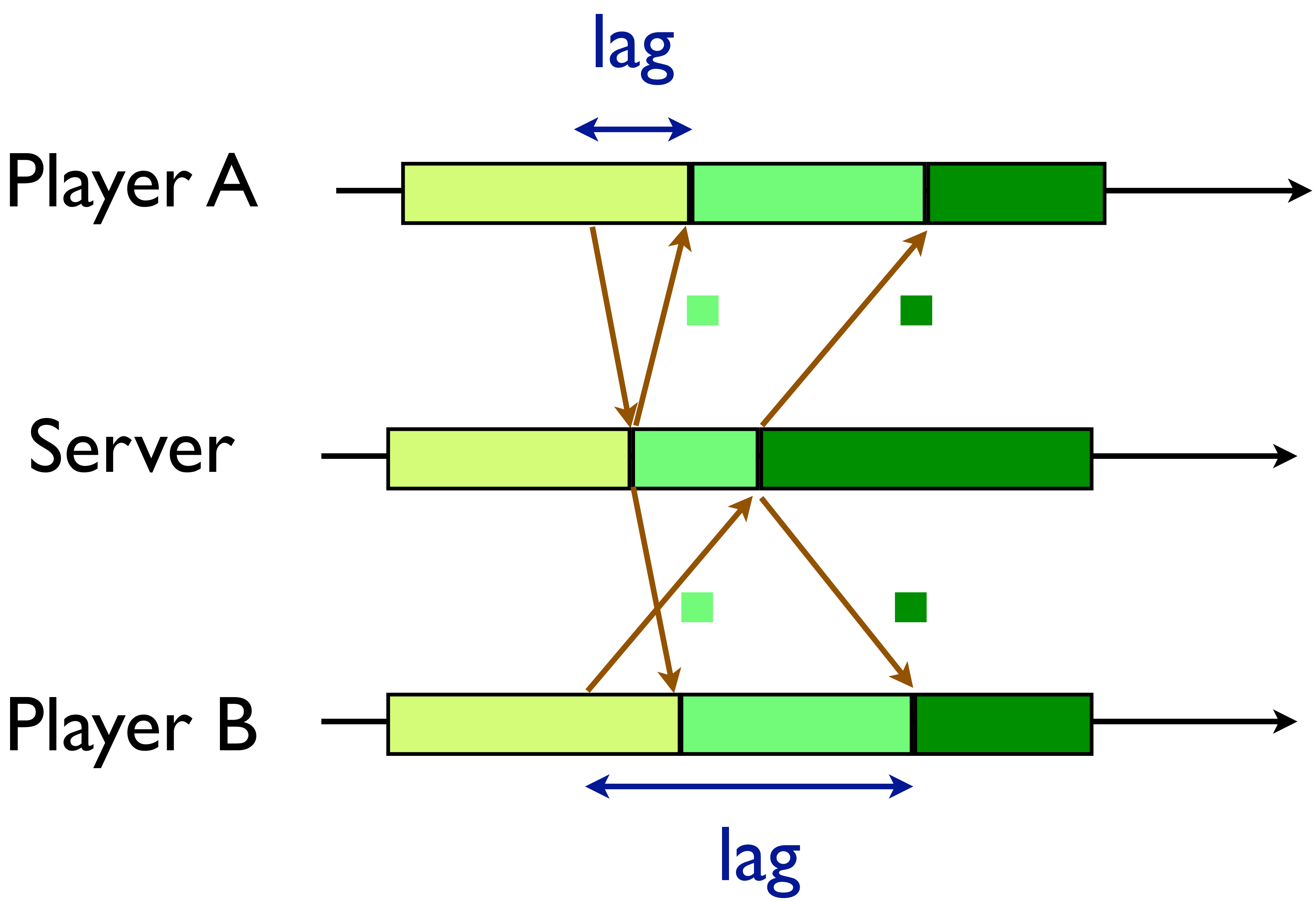
Quiz 1

Effects of delay jitter on four schemes

Some describe effects of latency instead

Permissible Client/Server Architecture

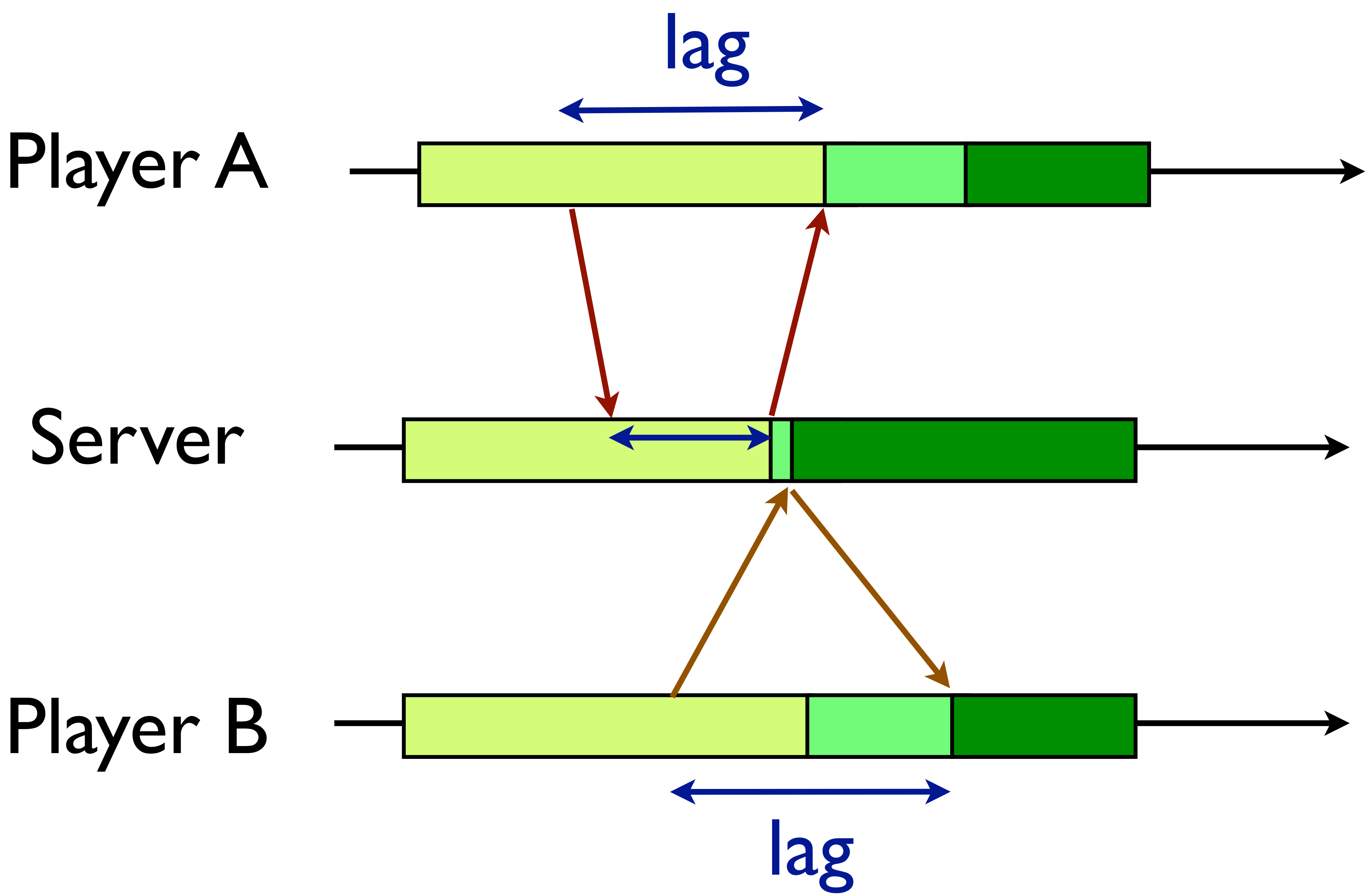


Fluctuating latency: Variable response time annoys users. Hard to compensate.

Clock Sync: Will not help

6

Improve fairness by artificial delay at the server. (longer delay for "closer" player)



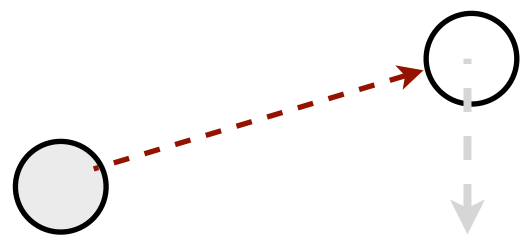
Need to know the RTT between server and client to insert artificial lag.

Fluctuating latency:
Hard to predict RTT.

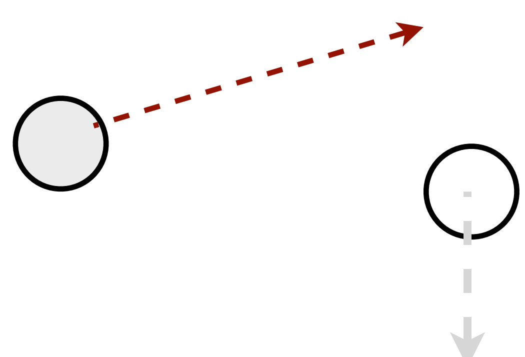
Clock Sync: Insert timestamp to measure latency.

10

Server estimates latency of message and go back to the time the message is generated.



Server
(now - t)



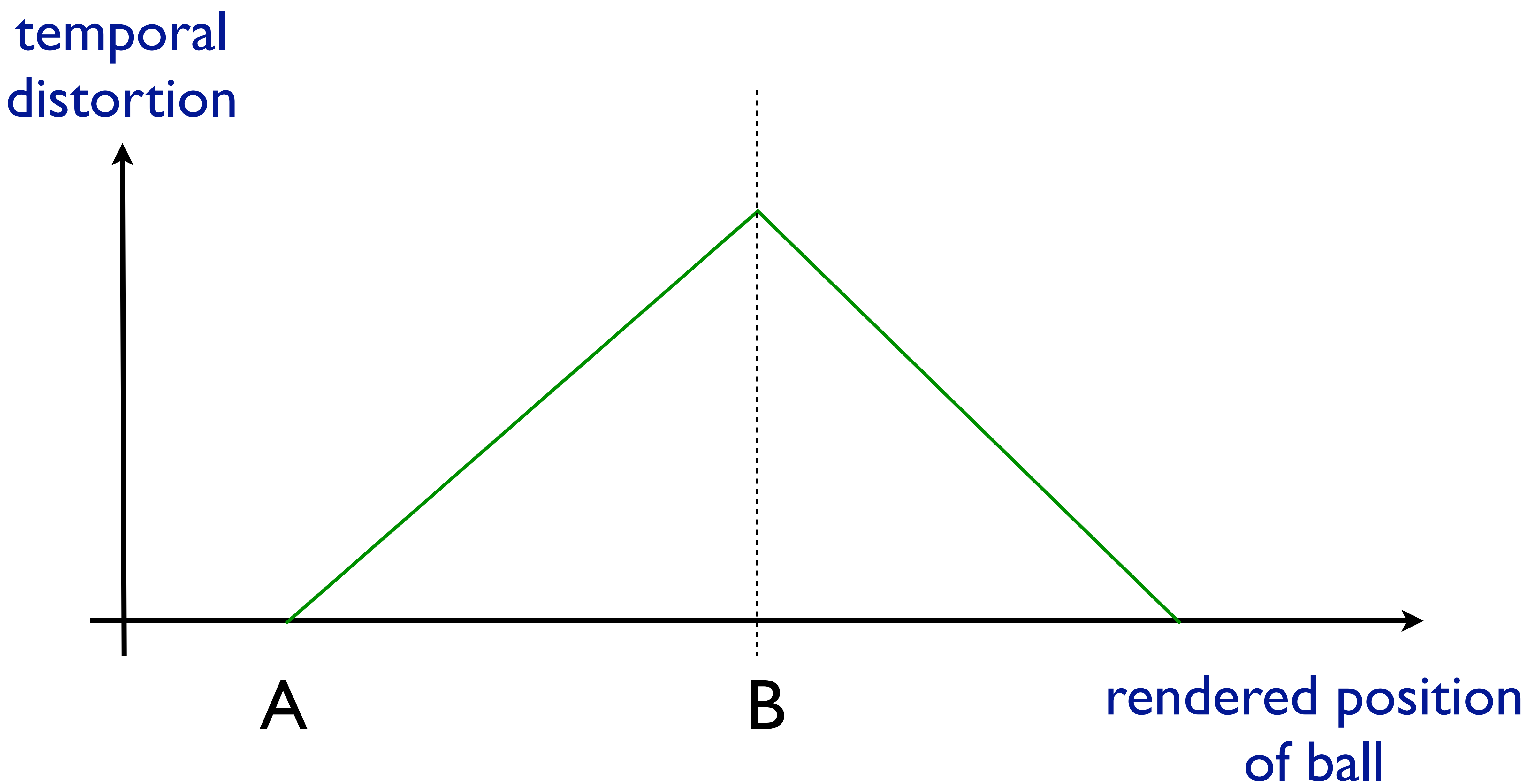
Server
(now)

11

Fluctuating latency:
Hard to estimate RTT

Clock Sync: Insert timestamp to measure latency.

Slow down/speed up movement of passive objects to improve consistency among players.



14

Fluctuating latency:
Hard to estimate RTT.
Speed fluctuates.

Clock Sync: Accurate estimation of latency won't help.

Peer-to-Peer Architecture

Problem:

Communication between Every Pair of Peers

Idea (old): A peer p only needs to communicate with another peer q if p is relevant to q

Recall: In C/S Architecture, the server has global information and decide who is relevant to who.

Problem: No global information in P2P architecture.

Naive Solution: Every peer keeps global information about all other peers and make individual decision.

Maintaining global information is expensive (and that's what we want to avoid in the first place!)

23

Smarter solution: exchange position, then decide when should the next position exchange be.

24

Idea: Assume B is static. If A knows B's position, A can compute the region which is irrelevant to B. Need not update B if A moves within that region.

what if B moves?

It still works if B also knows A position and computes the region that is irrelevant to A.

Position exchanges occur once initially, and when a player moves outside of its irrelevant region wrt another player.

28

Frontier Sets

cell-based, visibility-based IM

Previously, we learnt how to compute cell-to-cell visibility.

30

Frontier for cells X and Y consists of two sets F_{XY} and F_{YX}

31

No cell in F_{XY} is visible from a cell in F_{YX}, and vice versa.

32

F_{XY} and F_{YX} are disjoint

if X and Y are not mutually visible.

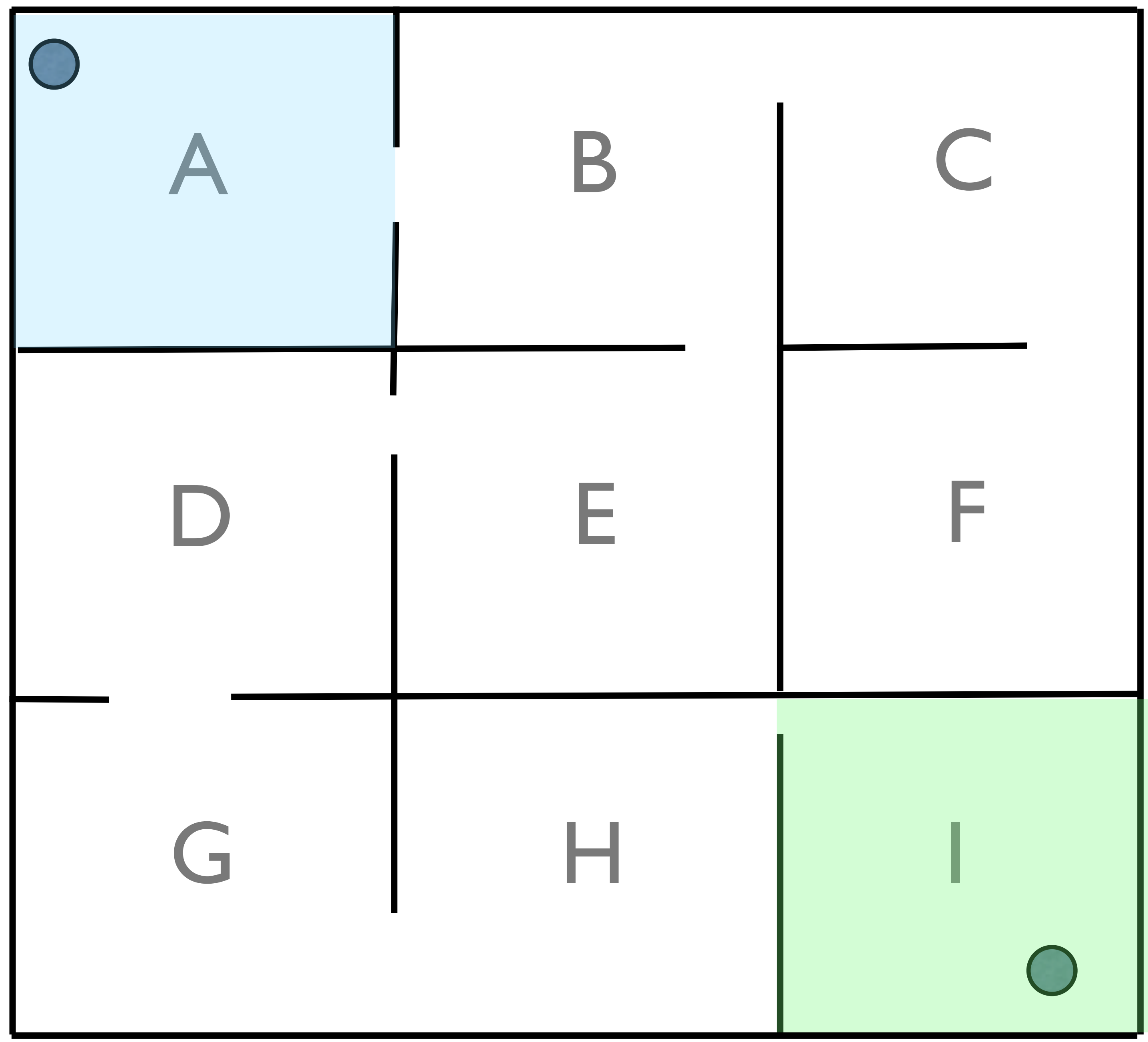
F_{XY} and F_{YX} are empty

if X and Y are mutually visible.

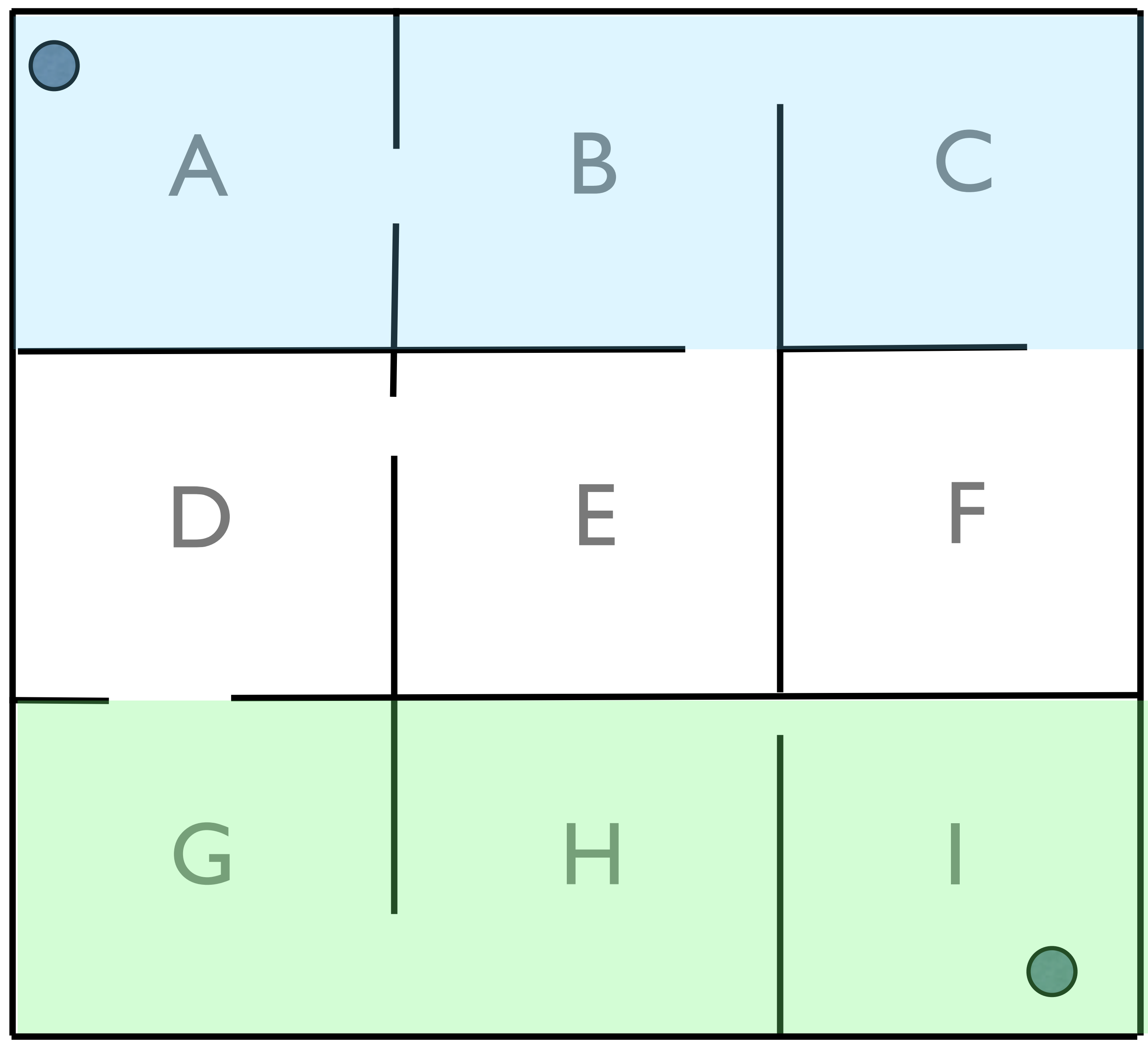
Suppose X and Y are not mutually visible, then a simple frontier is

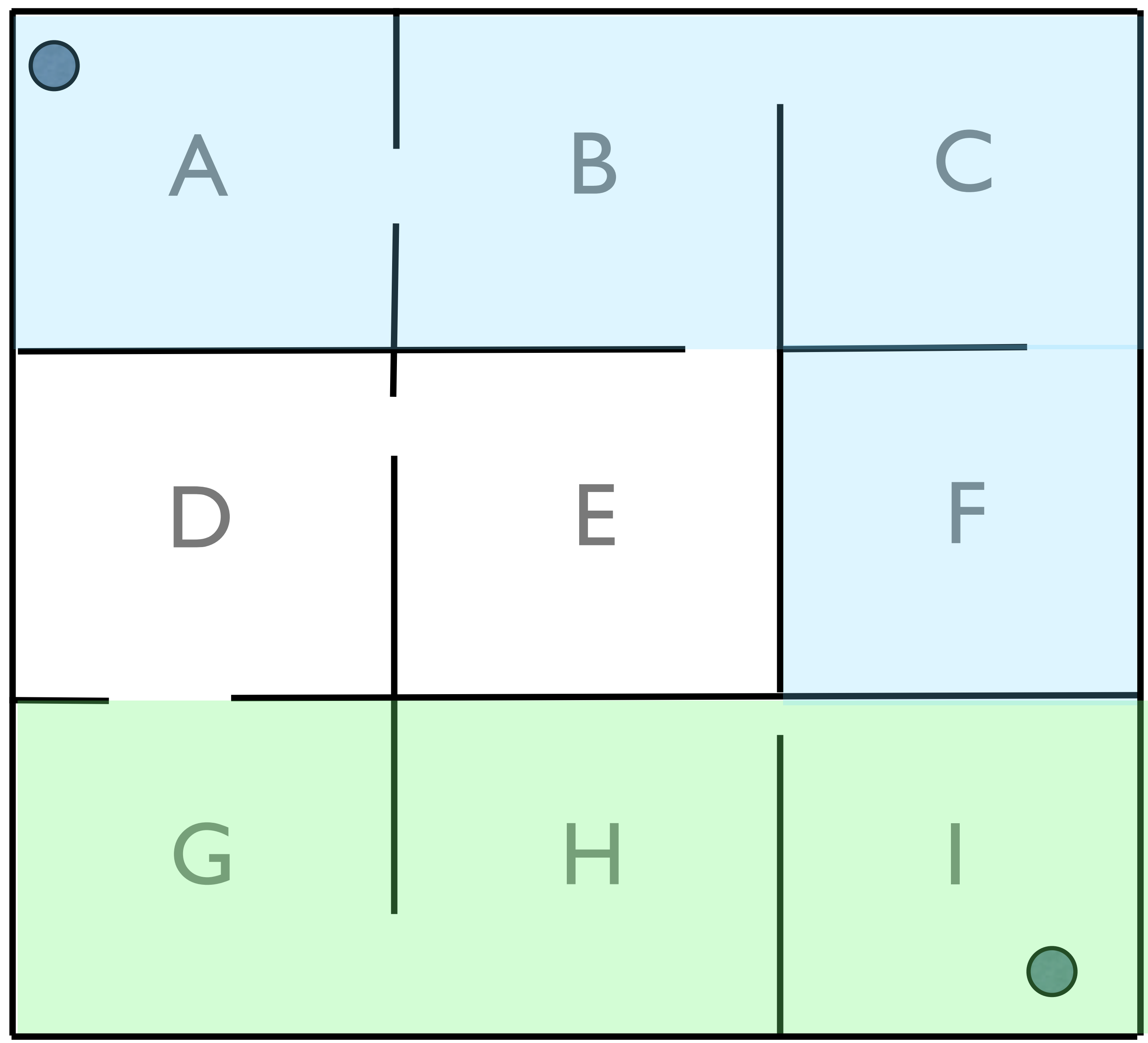
$$F_{XY} = \{X\} \quad F_{YX} = \{Y\}$$

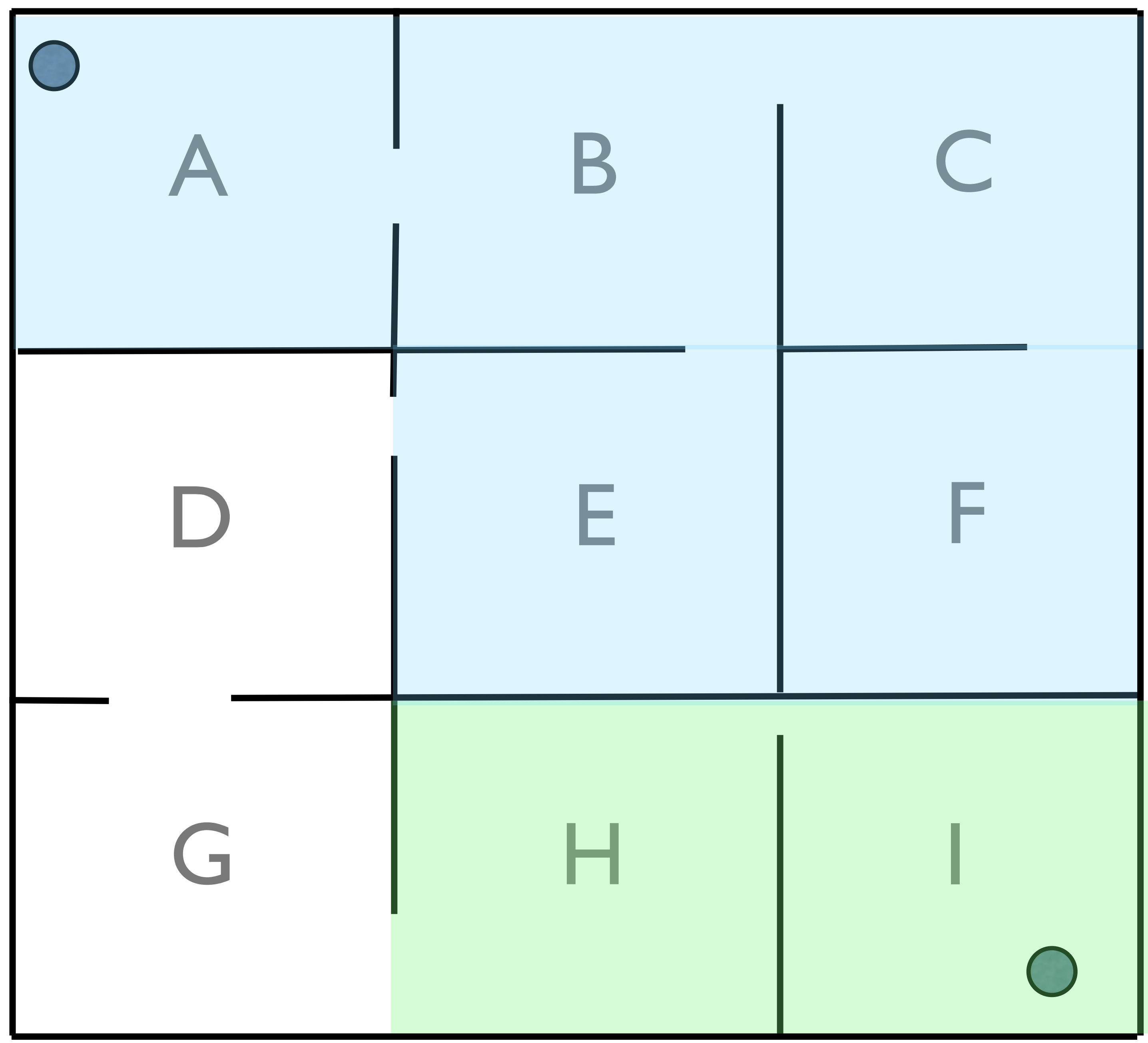
(many others are possible)



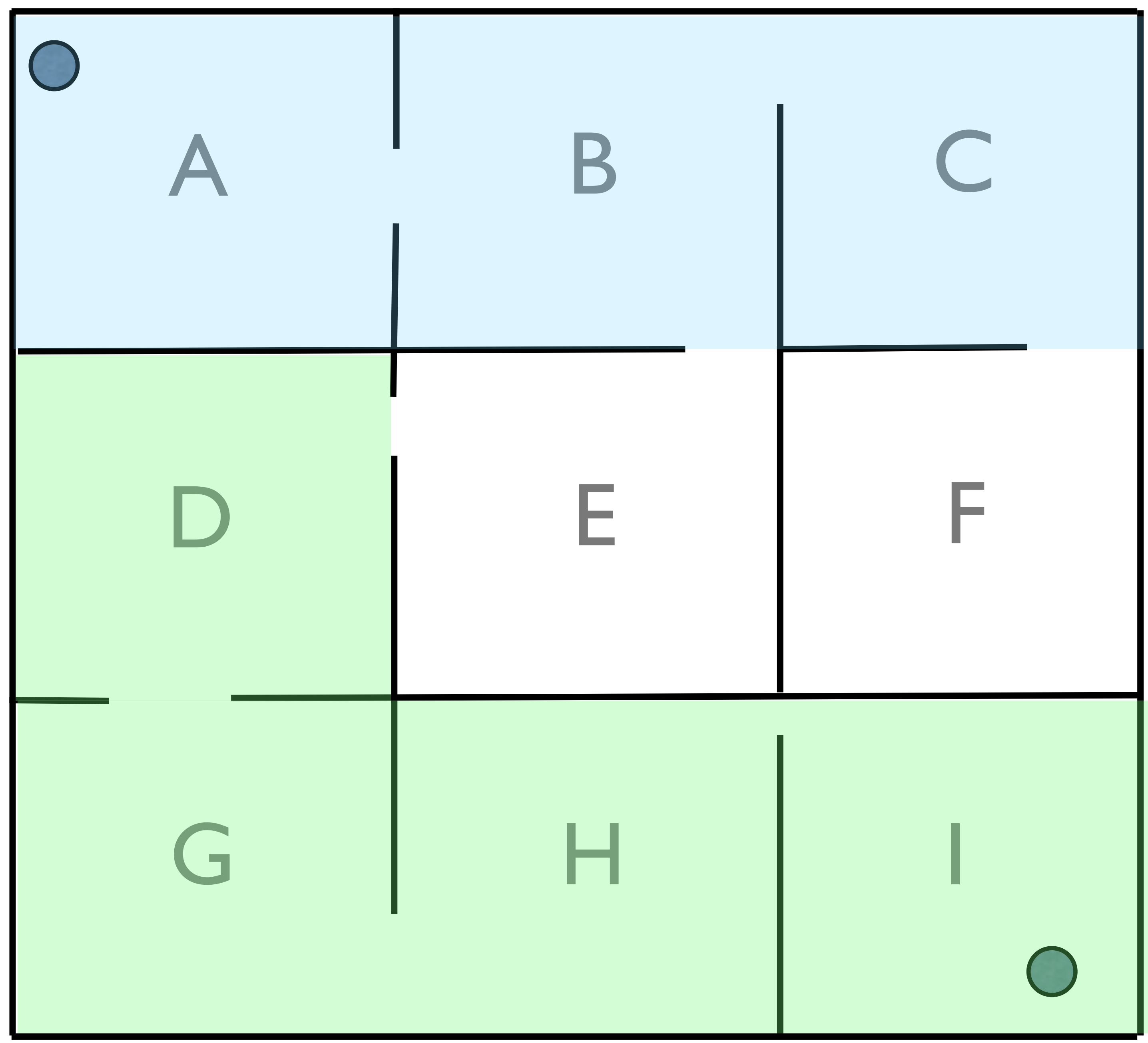
36


A
B
C
D
E
F
G
H
I


A
B
C
D
E
F
G
H
I


A
B
C
D
E
F
G
H
I

NOT a frontier for A and I (D is visible from B).



Position exchanges occur once initially, and when a player moves outside of its irrelevant region wrt another player.

41

Initialize:

Let player P be in cell X

For each player Q

  Let cell of Q be Y

  Compute F_{XY} (or simply F_Q)

Move to new cell:

Let X be new cell

For each player Q

  If X not in F_Q

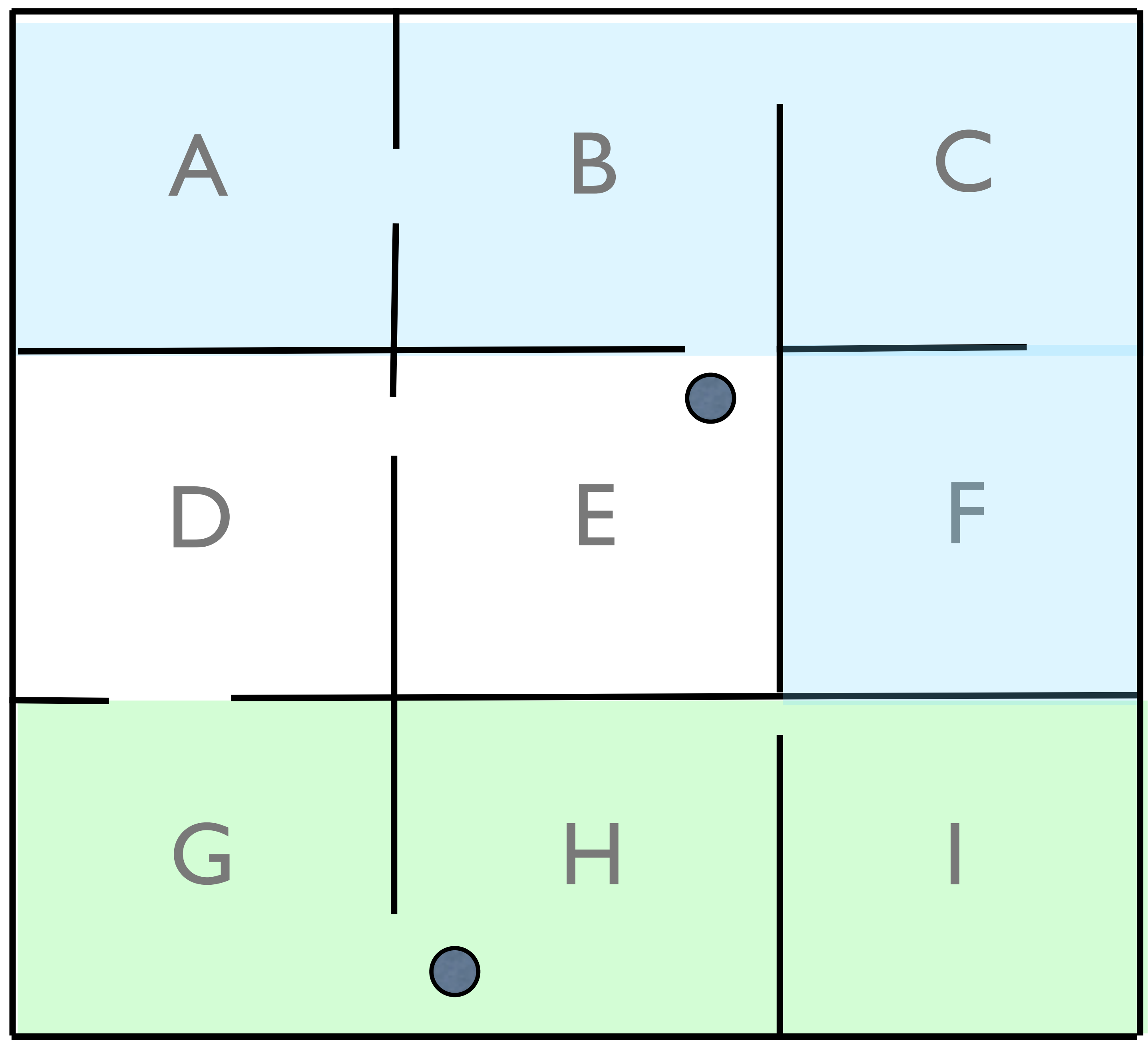
  Send location to Q

43

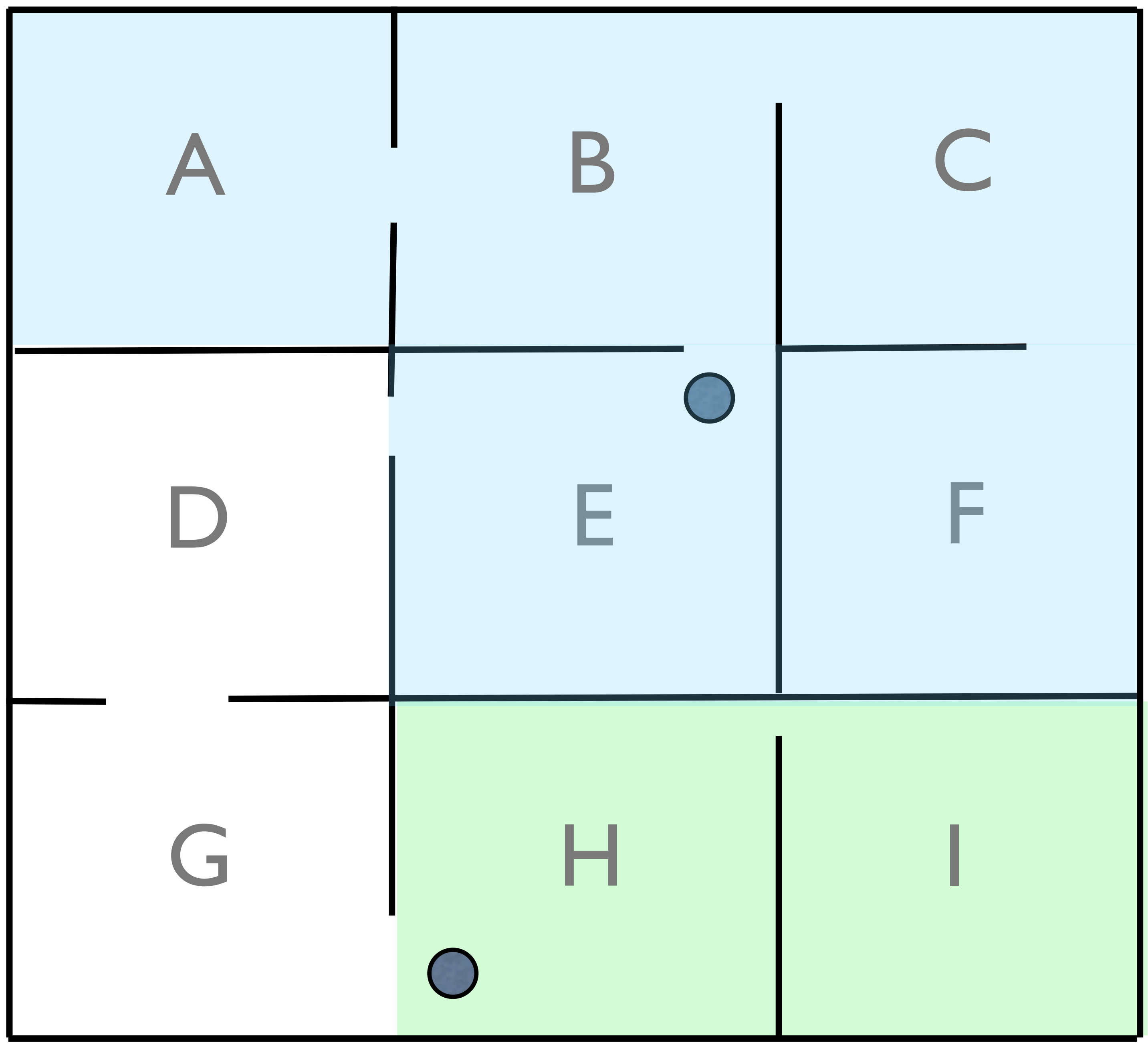
Receive Update:
(location from Q)
Send location to Q
Recompute F_Q

44

Update is triggered.

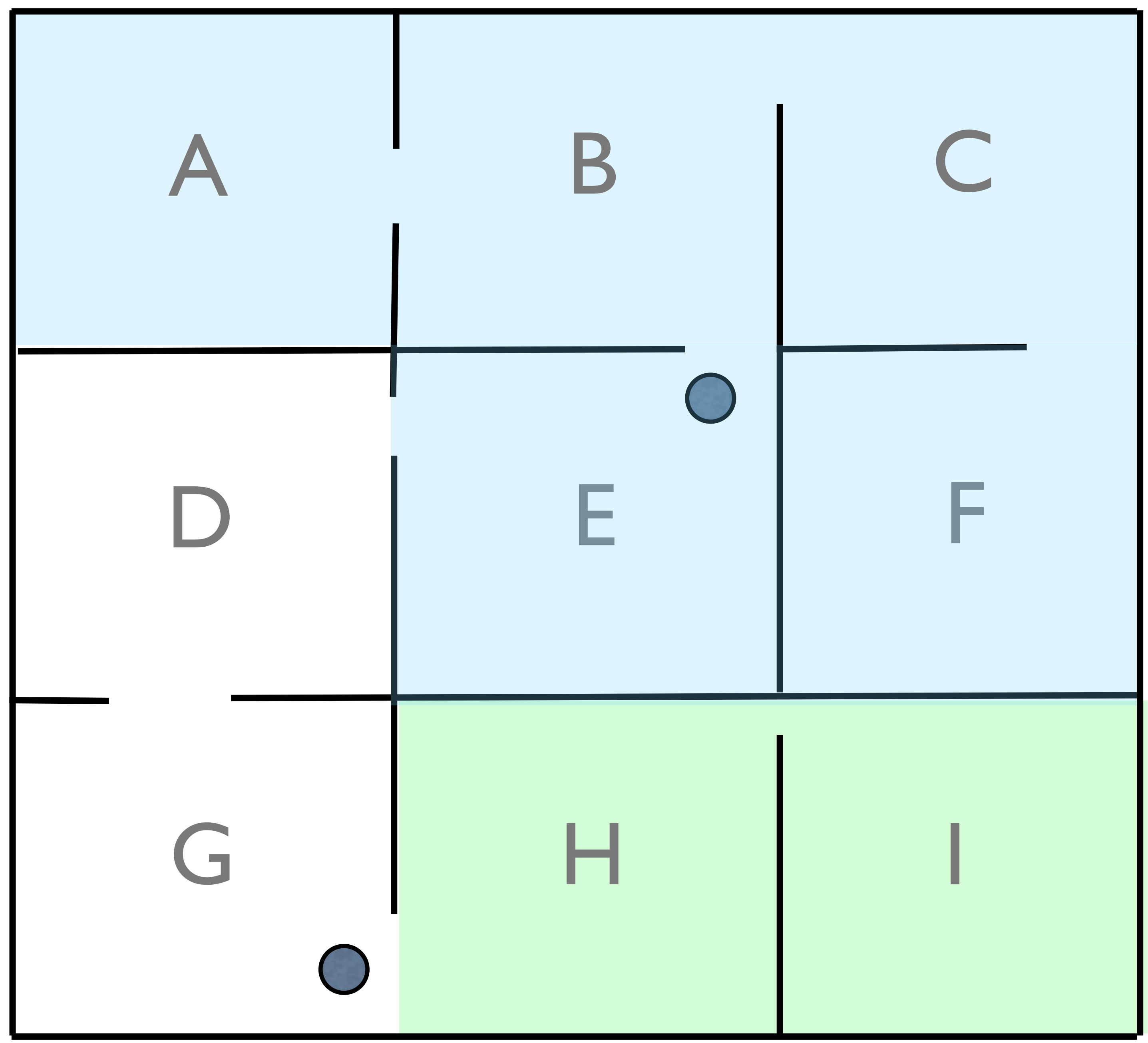


New Frontier.



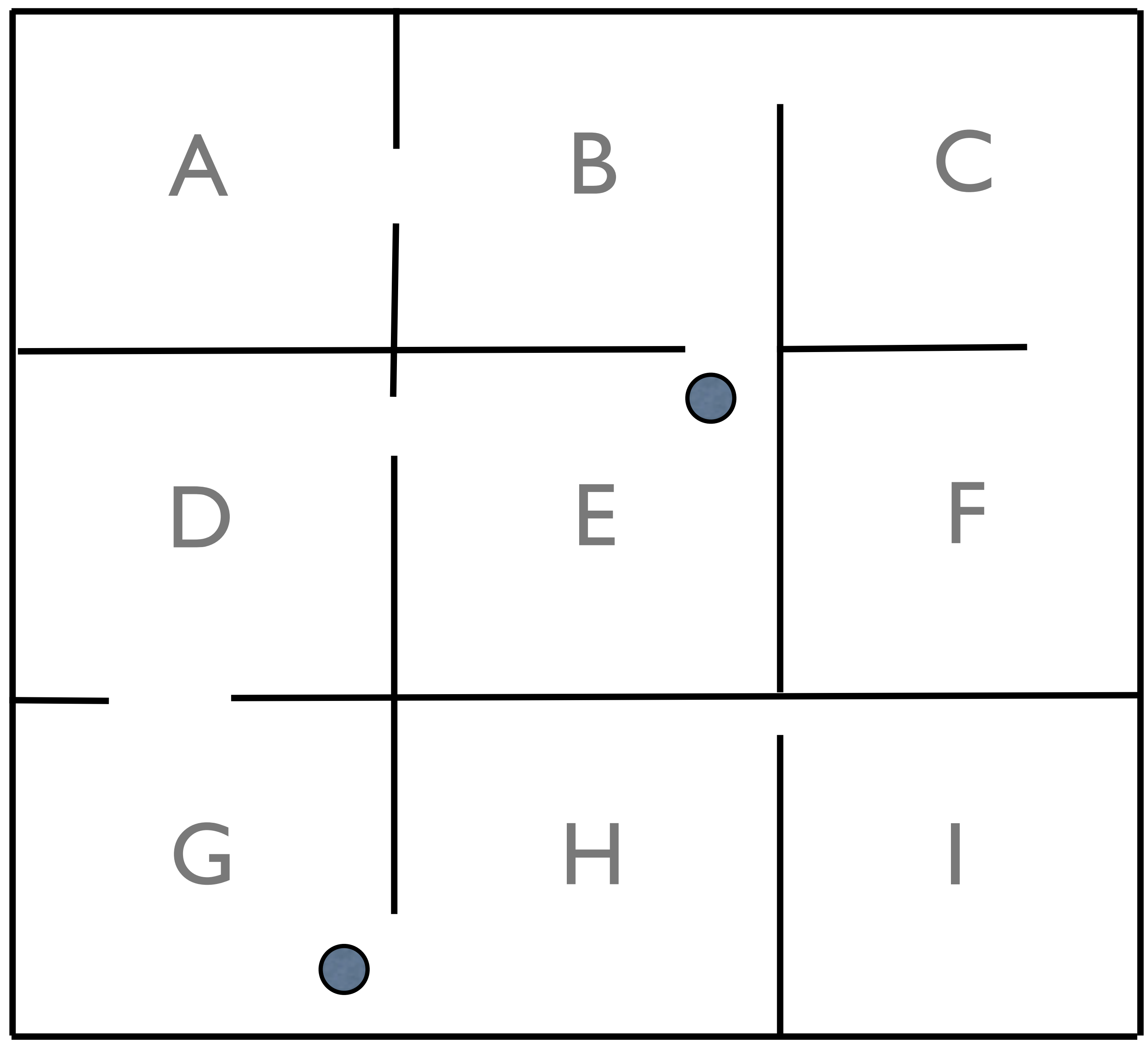
46

Update triggered.



47

New frontier (empty since E can see G)



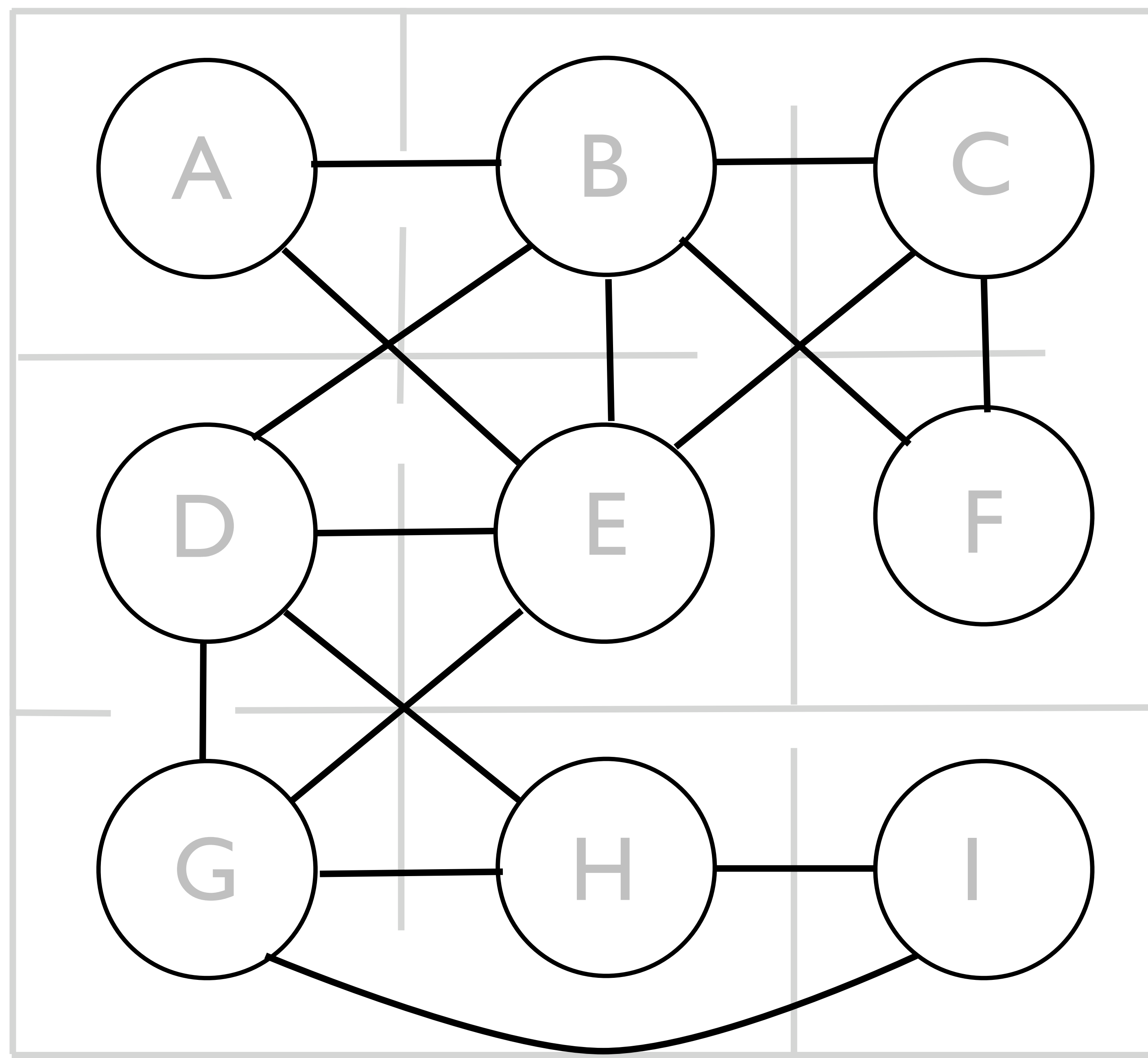
48

How to compute frontier?

49

A good frontier is as large as possible, with two almost equal-size sets.

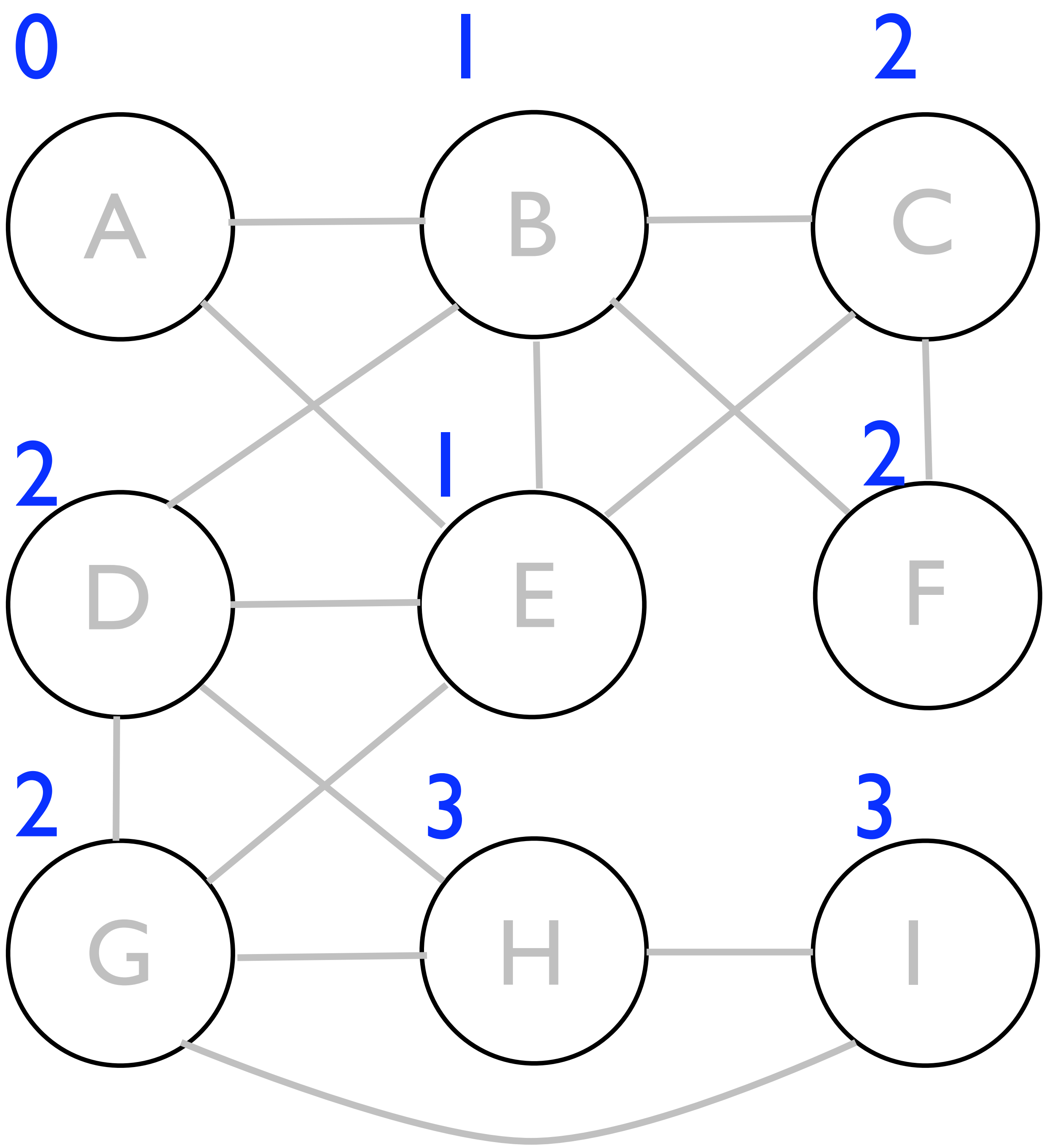
Build a visibility graph. Cells are vertices. Two cells are connected by an edge if they are visible to each other (EVEN if they don't share a boundary)



51

Let dist(X, Y) be the shortest distance between two cells X and Y on the visibility graph.

52



53

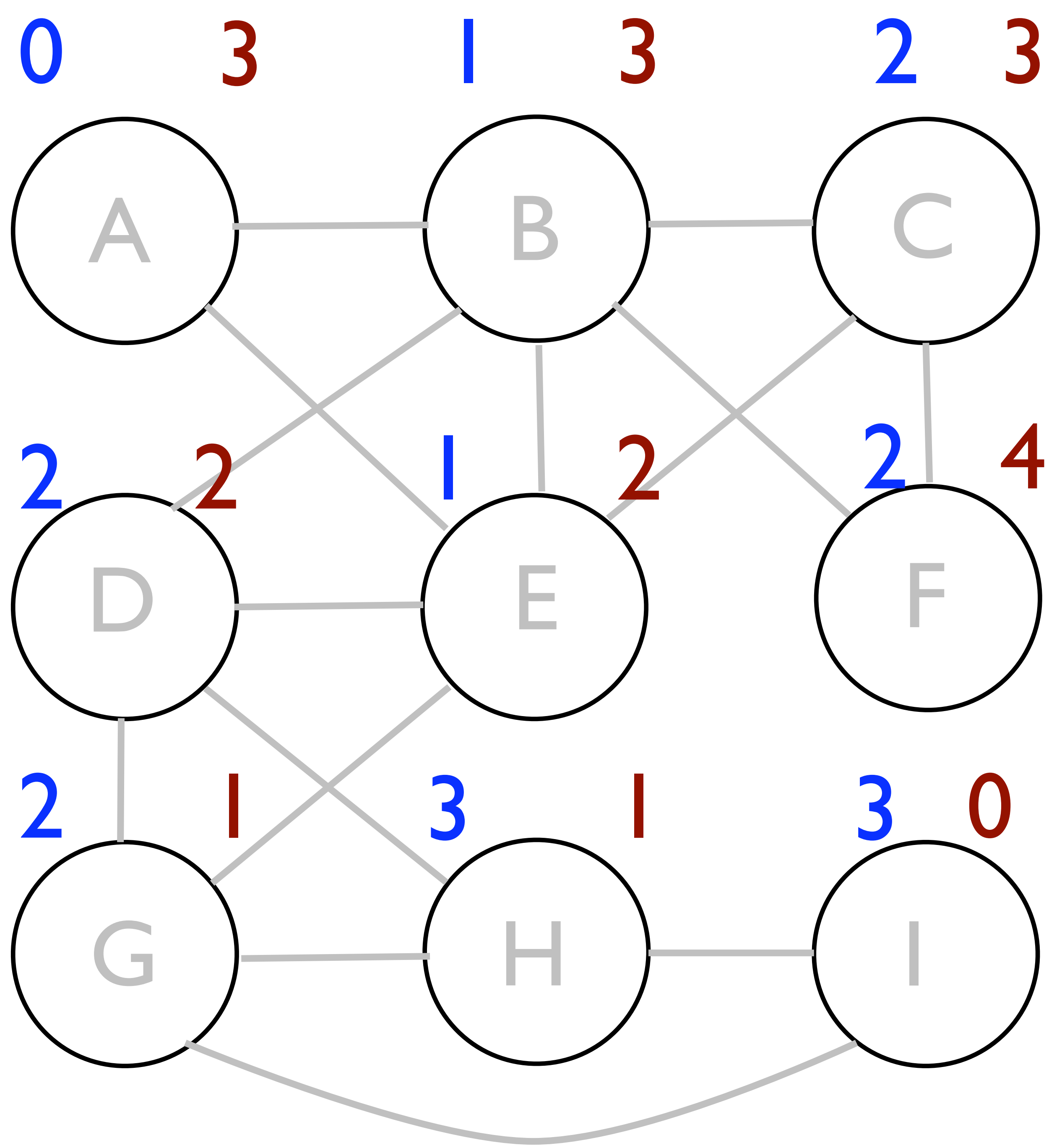
Theorem

$F_{XY} = \{ i \mid dist(X,i) <= dist(Y,i) - 1 \}$

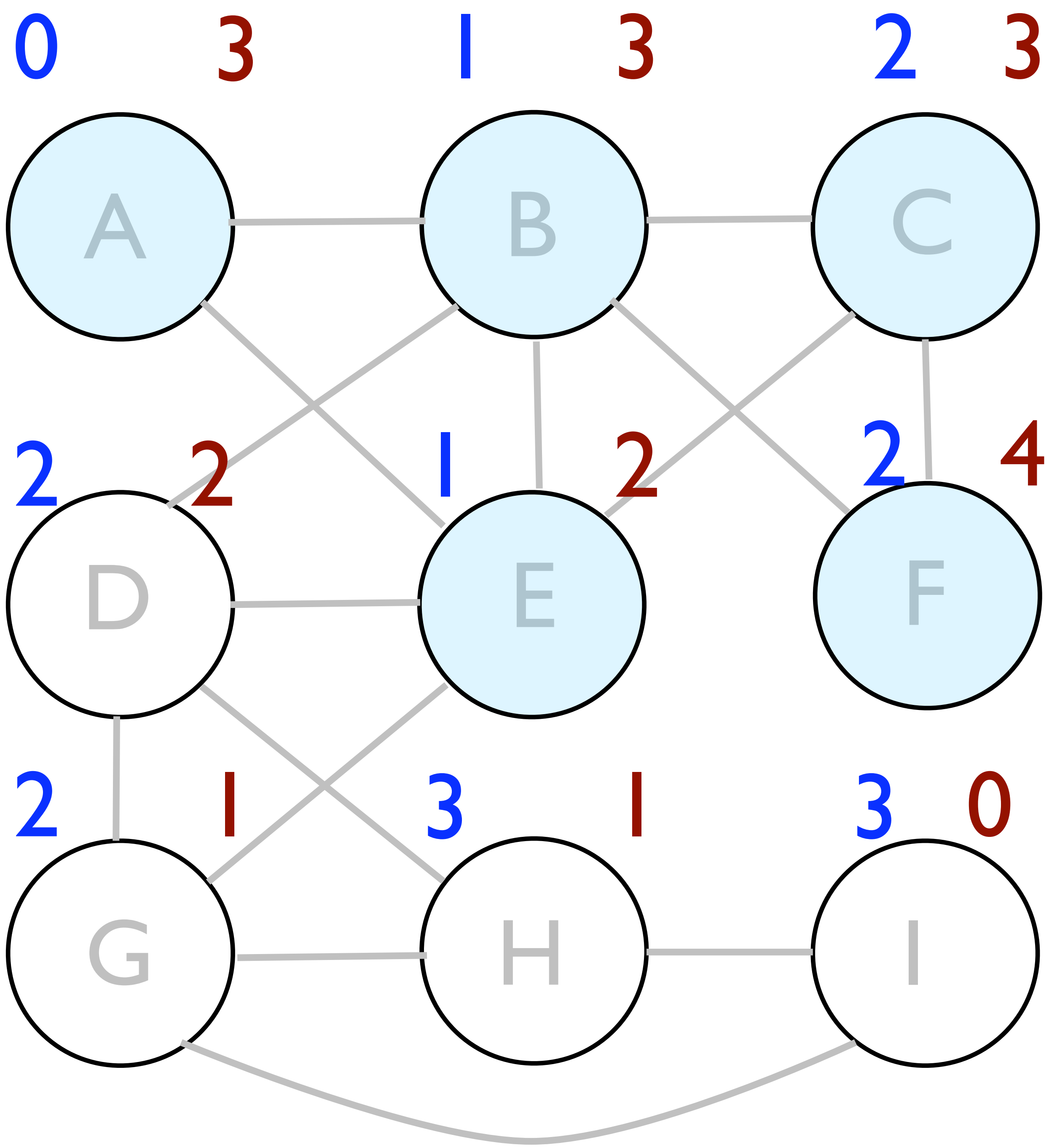
$F_{YX} = \{ j \mid dist(Y,j) < dist(X,j) - 1 \}$

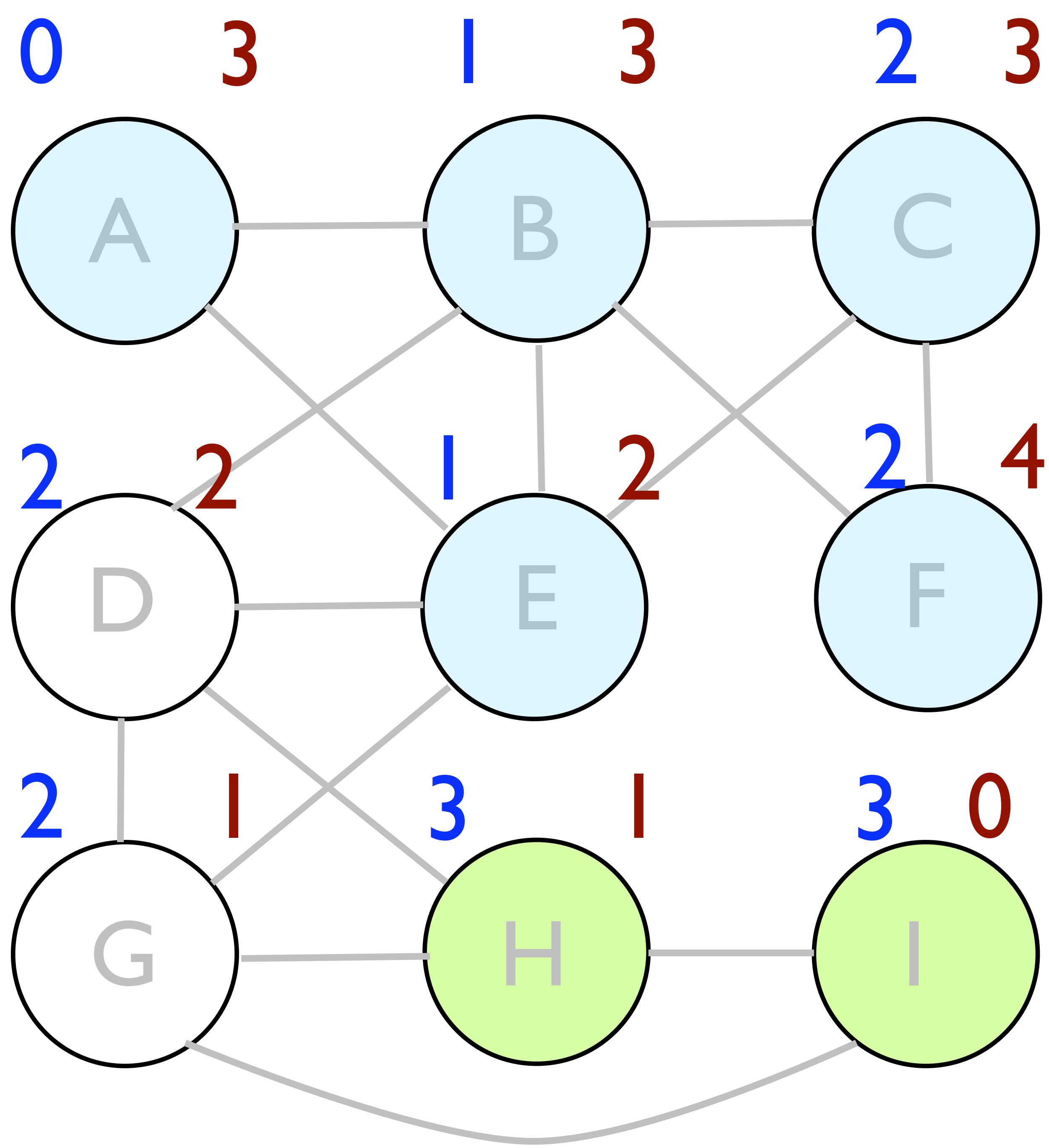
are valid frontiers.

54

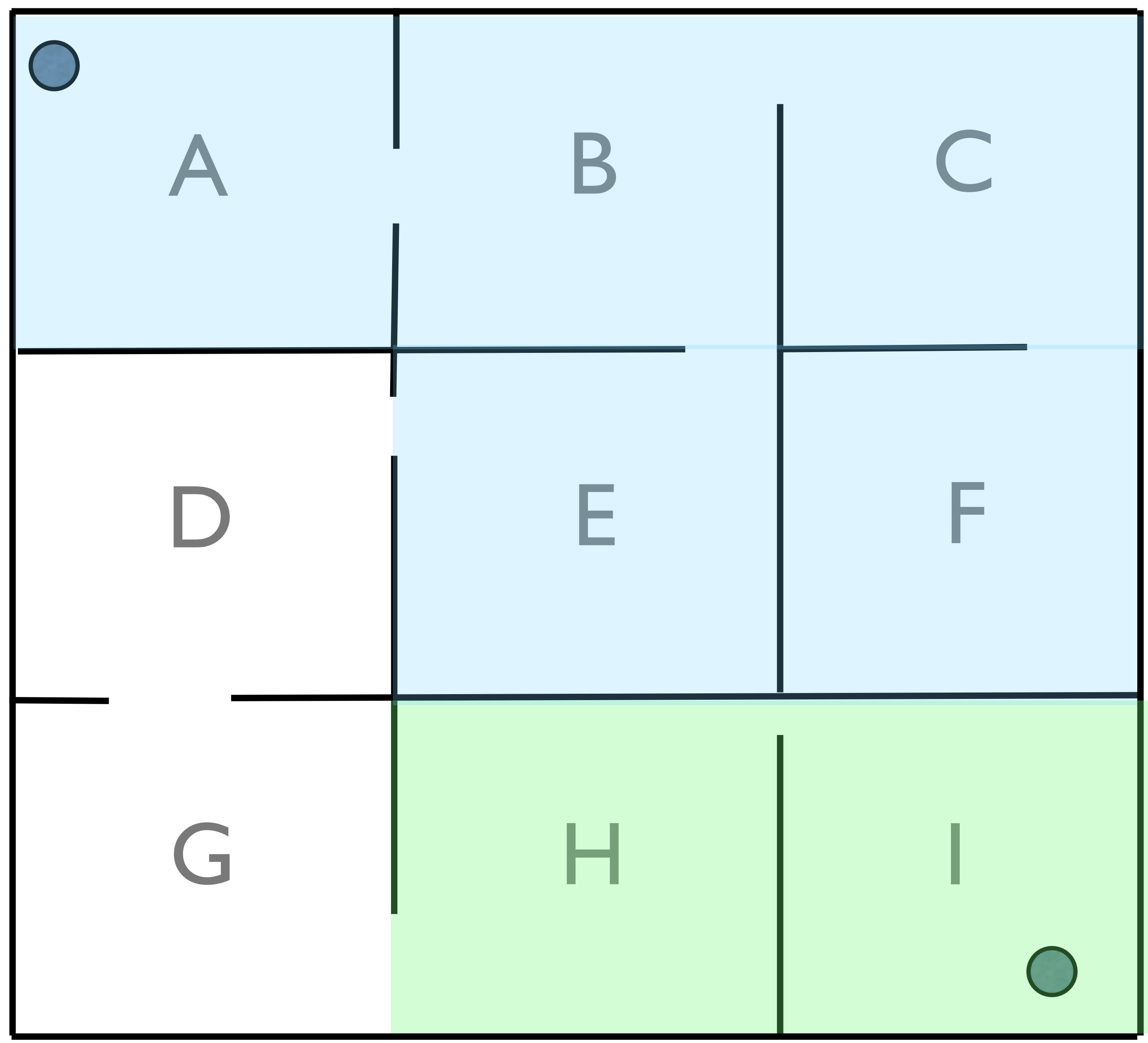


55


0
3
1
3
2
3
A
B
C
2
2
1
2
2
4
D
E
F
2
1
3
1
3
0
G
H
I



57



58

Theorem

$F_{XY} = \{ i \mid dist(X,i) <= dist(Y,i) - 1 \}$

$F_{YX} = \{ j \mid dist(Y,j) < dist(X,j) - 1 \}$

are valid frontiers.

59

$F_{XY} = \{ i \mid dist(X,i) <= dist(Y,i) - 1 \}$
$F_{YX} = \{ j \mid dist(Y,j) < dist(X,j) - 1 \}$

Proof (by contradiction)
Suppose there are two cells, C in F_{XY} and D in F_{YX}, that can see each other.

$$F_{XY} = \{ i \mid dist(X,i) <= dist(Y,i) - 1 \}$$

$$F_{YX} = \{ j \mid dist(Y,j) < dist(X,j) - 1 \}$$

$$dist(X,C) <= dist(Y,C) - 1$$

$$dist(Y,D) < dist(X,D) - 1$$

$$dist(C,D) = dist(D,C) = 1$$

61

dist(X,C) <= dist(Y,C) - 1
dist(Y,D) < dist(X,D) - 1
dist(C,D) = dist(D,C) = 1

We also know that
dist(X,D) <= dist(X,C) + dist(C,D)
dist(Y,C) <= dist(Y,D) + dist(D,C)

62

1. dist(X,C) <= dist(Y,C) - 1
2. dist(Y,D) < dist(X,D) - 1
3. dist(C,D) = 1
4. dist(X,D) <= dist(X,C) + dist(C,D)
5. dist(Y,C) <= dist(Y,D) + dist(D,C)

From 4, 1, and 3:

dist(X,D) <= dist(Y,C) - 1 + 1

From 5:

dist(X,D) <= dist(Y,D) + 1

63

1. dist(X,C) <= dist(Y,C) - 1
2. dist(Y,D) < dist(X,D) - 1
3. dist(C,D) = 1
4. dist(X,D) <= dist(X,C) + dist(C,D)
5. dist(Y,C) <= dist(Y,D) + dist(D,C)

We have

dist(X,D) <= dist(Y,D) + 1

Which contradict 2

dist(X,D) > dist(Y,D) + 1

64

How good is the idea?

(How many messages can we save
by using Frontier Sets?)

65

	q2dm3	q2dm4	q2dm8
Max dist()	4	5	8
Num of cells	666	1902	966

Frontier Density: % of player-pairs with non-empty frontiers.

67

	q2dm3	q2dm4	q2dm8
Frontier Density	83.9	93.0	84.2

68

Frontier Size:

% of cells in the frontier on average

69

	q2dm3	q2dm4	q2dm8
Frontier Size	38.3%	67.3%	68.2%

70

Compare with

1. Naive P2P
2. Perfect P2P

71

Naive P2P
Always send update to 15 other players.

72

Perfect P2P

Hypothetical protocol that sends messages only to visible players.

73

Number of messages per frame per player.

Number of messages per frame per player.

	q2dm3	q2dm4	q2dm8

Number of messages per frame per player.

	q2dm3	q2dm4	q2dm8
NPP	15	15.7	14.4

Number of messages per frame per player.

	q2dm3	q2dm4	q2dm8
NPP	15	15.7	14.4
PPP	3.7	1.9	4.2

Number of messages per frame per player.

	q2dm3	q2dm4	q2dm8
NPP	15	15.7	14.4
PPP	3.7	1.9	4.2
Frontier	5.4	2.6	5.9

Space Complexity

Let N be the number of cells. If we precompute Frontier for every pair of cells, we need

$$O(N^3)$$

space.

If we store visibility graph and compute frontier as needed, we only need

$$O(N^2)$$

space.

Frontier Sets

cell-based, visibility-based IM

77

Limitations

78

Works badly if there's little occlusion in the virtual world.

Still need to exchange locations with every other players occasionally.

80

Frontier Sets

cell-based, visibility-based IM

Voronoi Overlay Network:
Aura-based Interest Management

82

Diagrams and plots in the sections are taken from presentation slides by Shun-yun Hu, available on http://vast.sf.net

x

Keep a list of neighbors within AOI and exchange messages with neighbors.

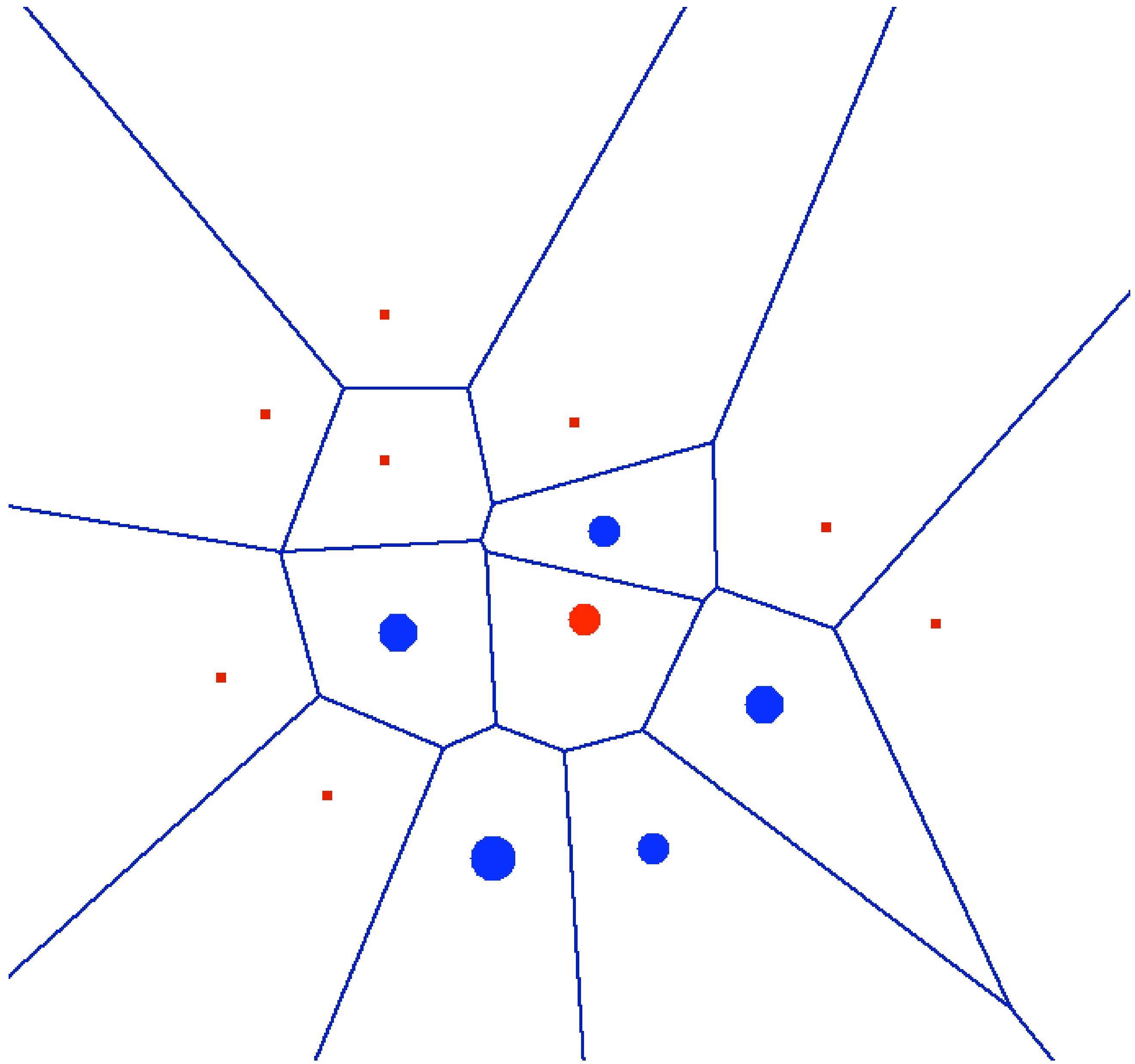
How to initialize list of neighbors?

How to keep list of neighbors up-to-date?

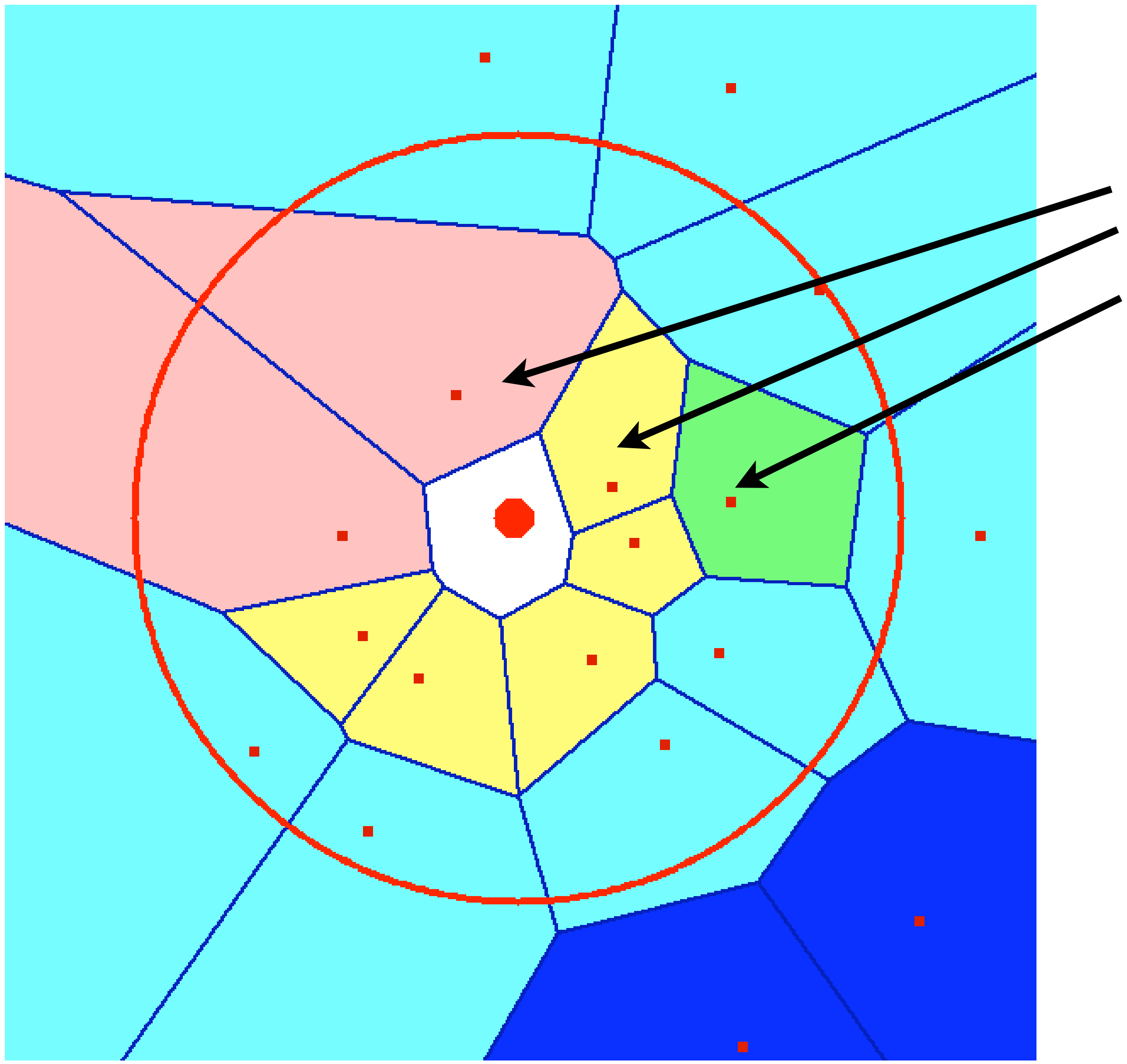
Every node is in charge of a region in the virtual world.

The region contains points closest to the node.

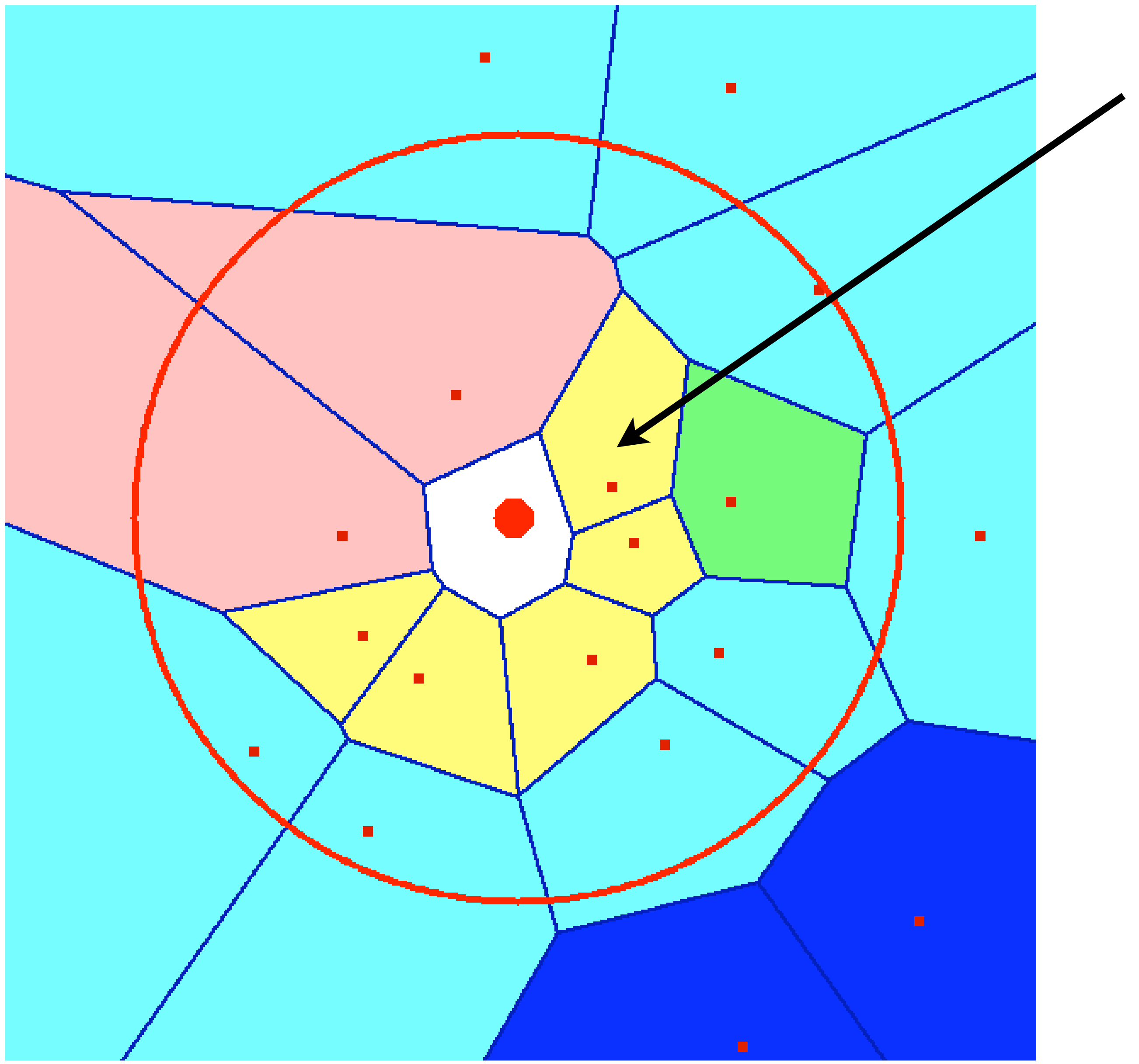
Voronoi Diagram



86

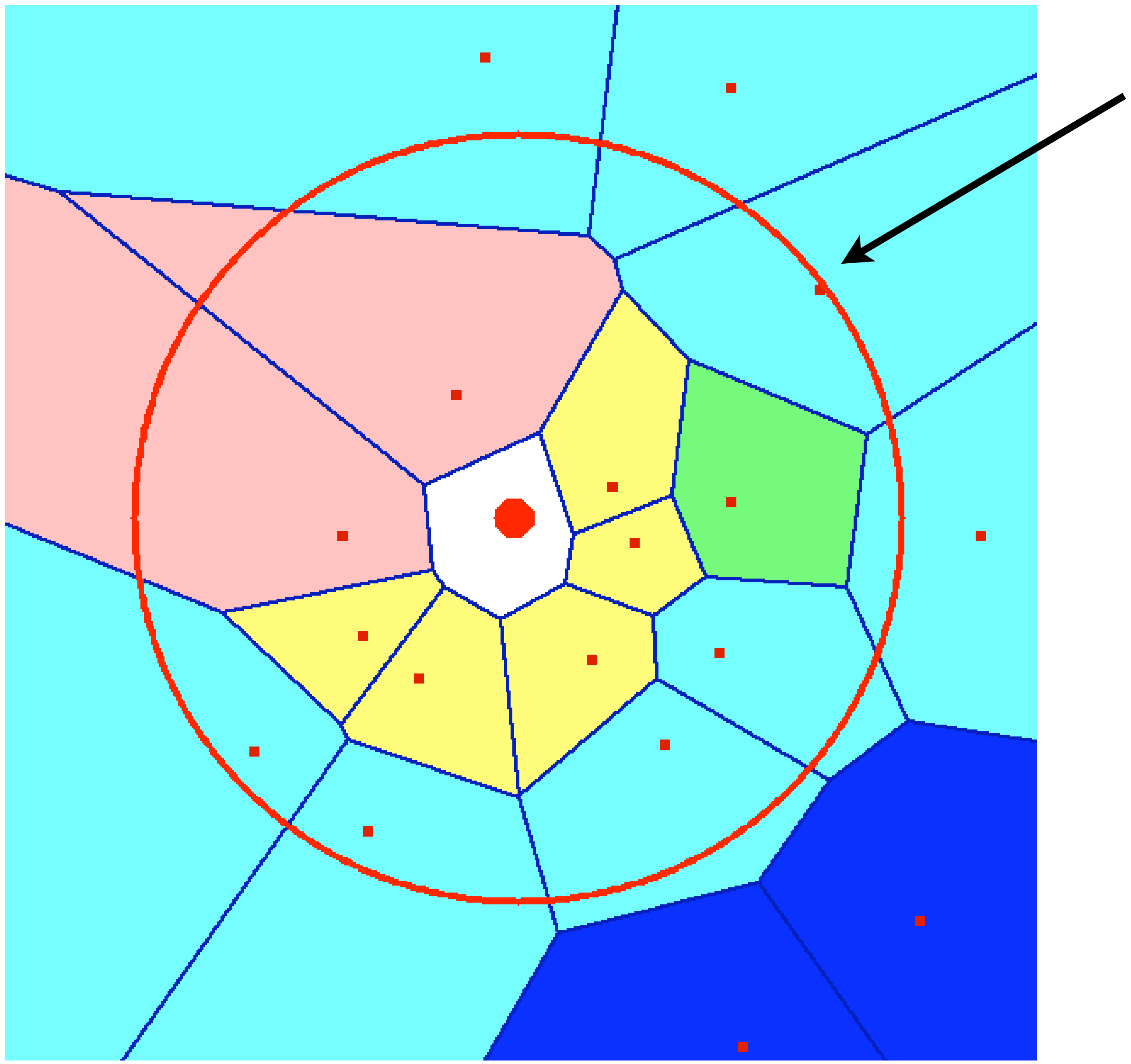


AOI Neighbors:
Neighbors in AOI



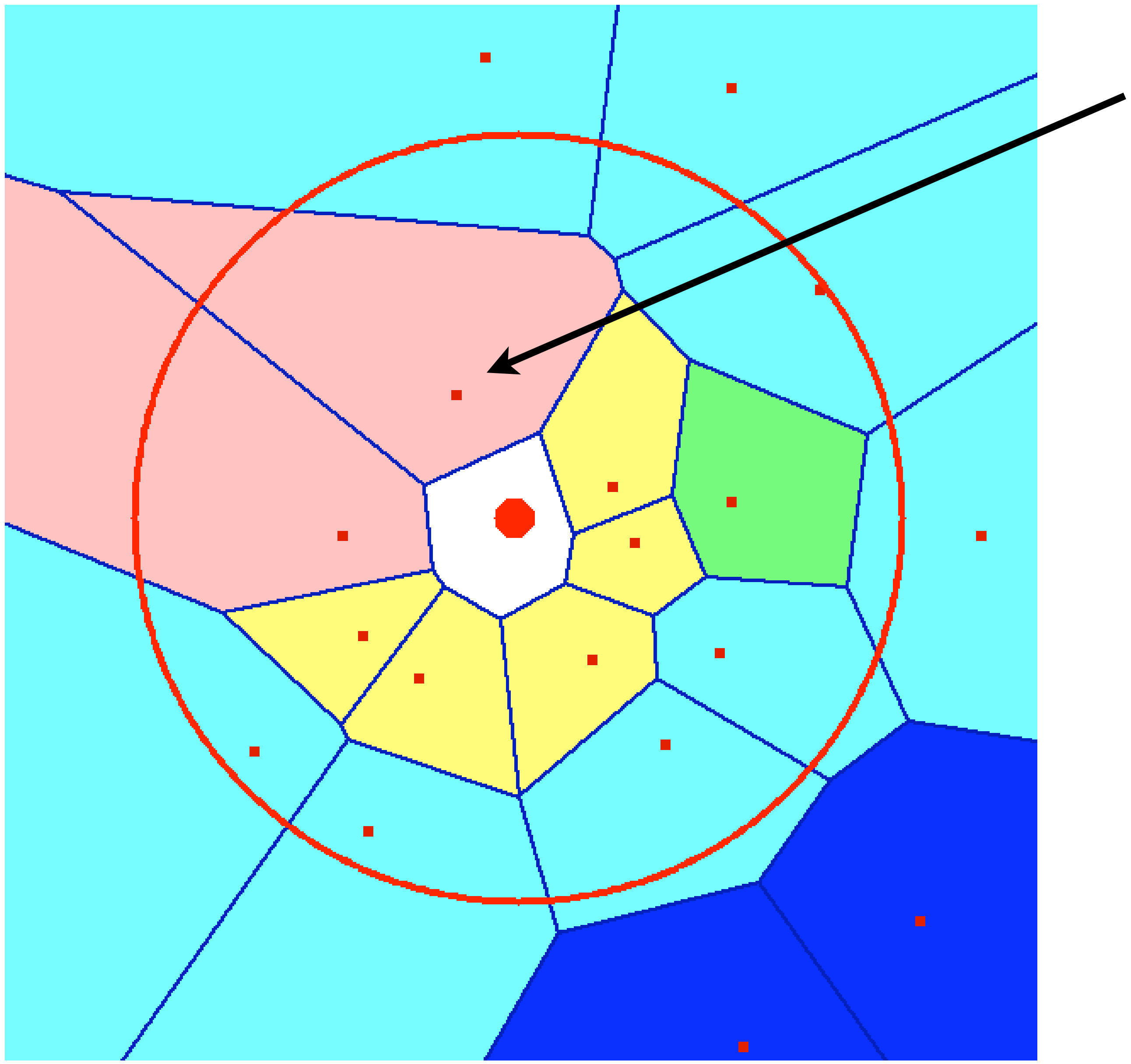
Enclosing Neighbors: Neighbors in adjacent region.

(may or may not be in AOI)

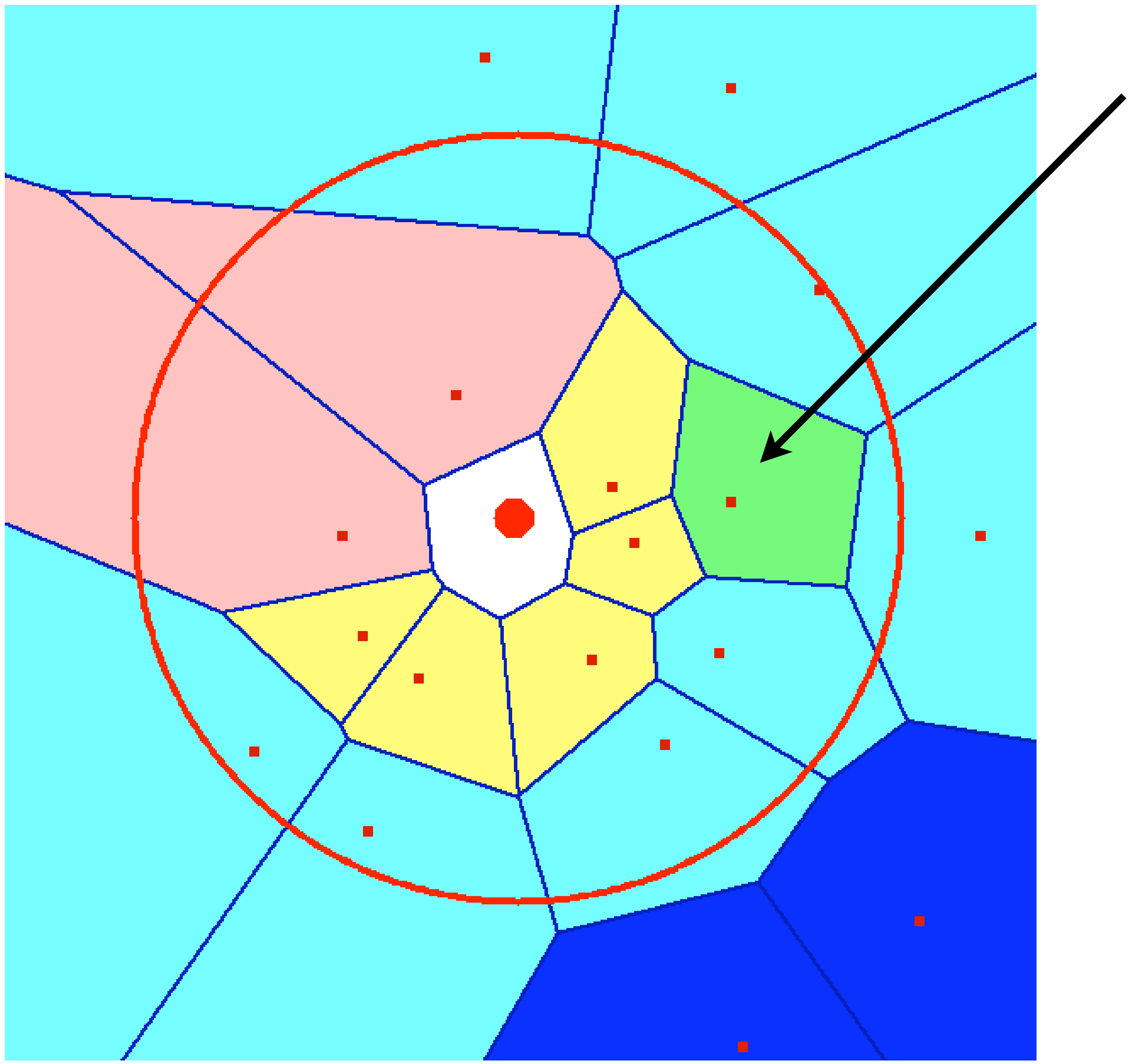


Boundary Neighbors: Neighbors whose region intersect with AOI.

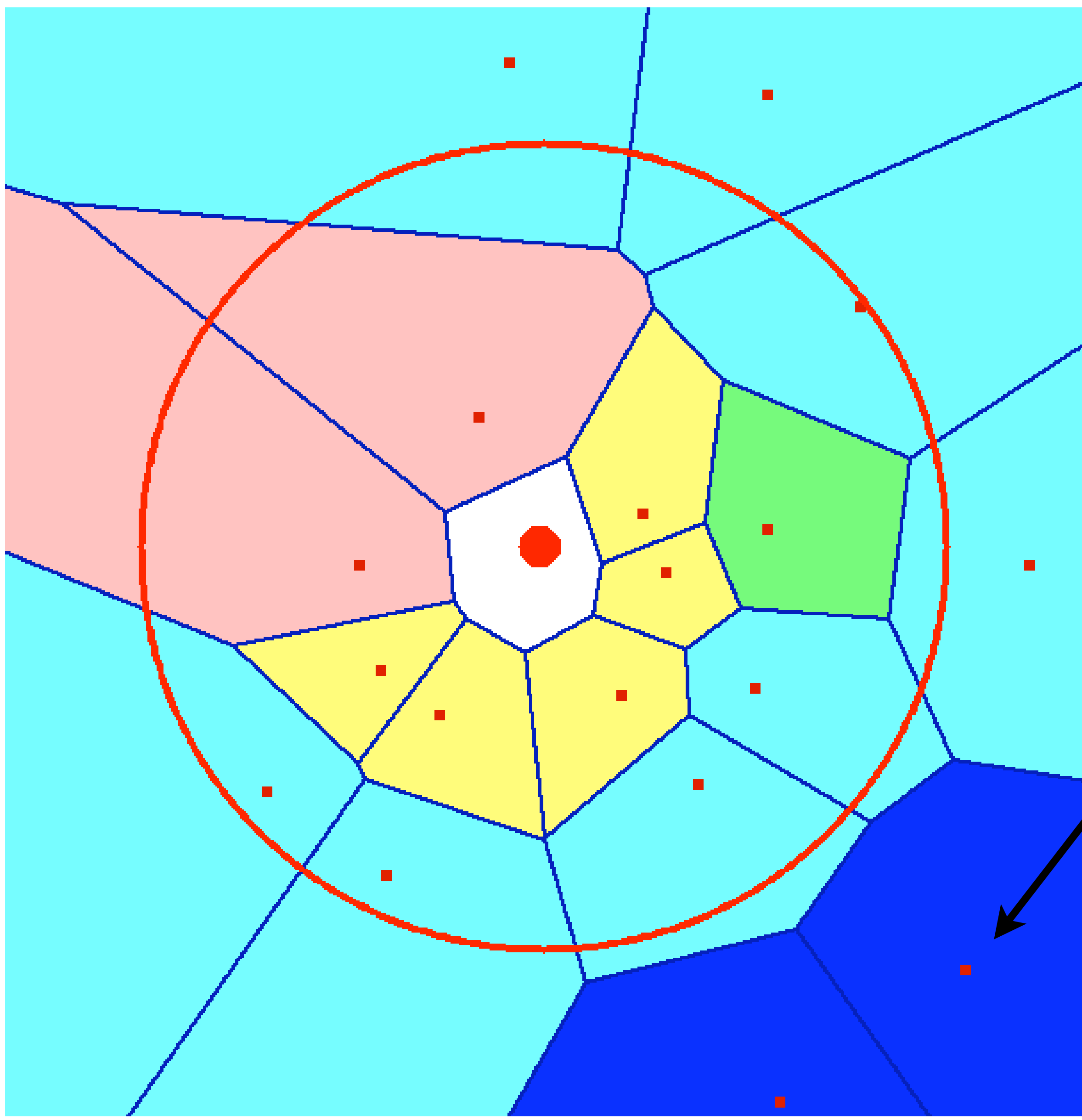
(may or may not be in AOI)



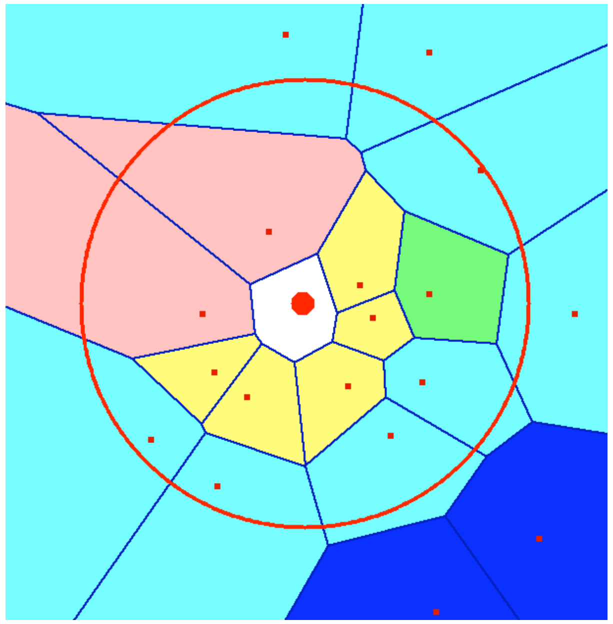
Boundary and Enclosing Neighbor



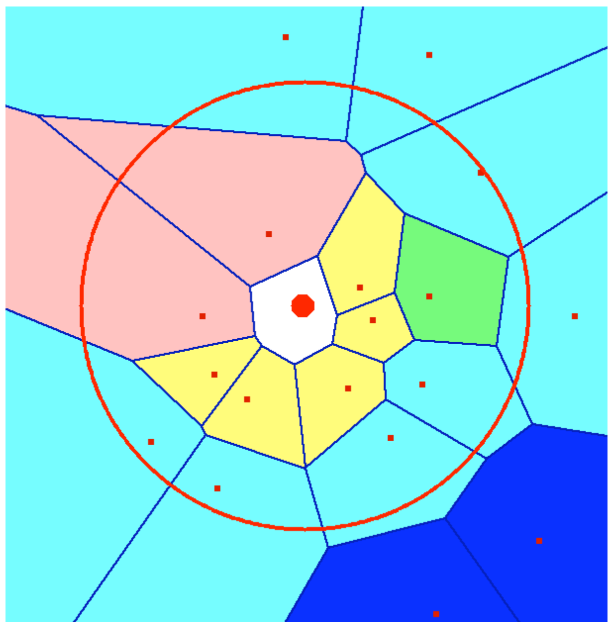
Regular AOI Neighbor: Non-boundary and non-enclosing neighbor in AOI



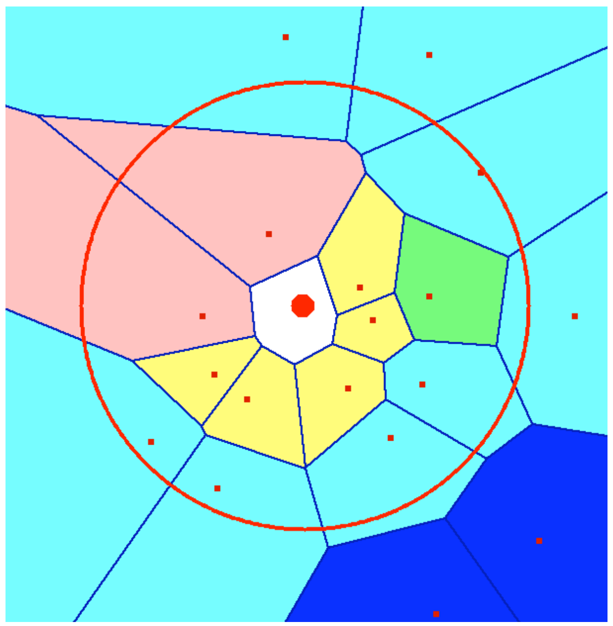
Unknown nodes (not neighbors!)



A node always connect to its enclosing neigbours, regardless of whether they are in the AOI.



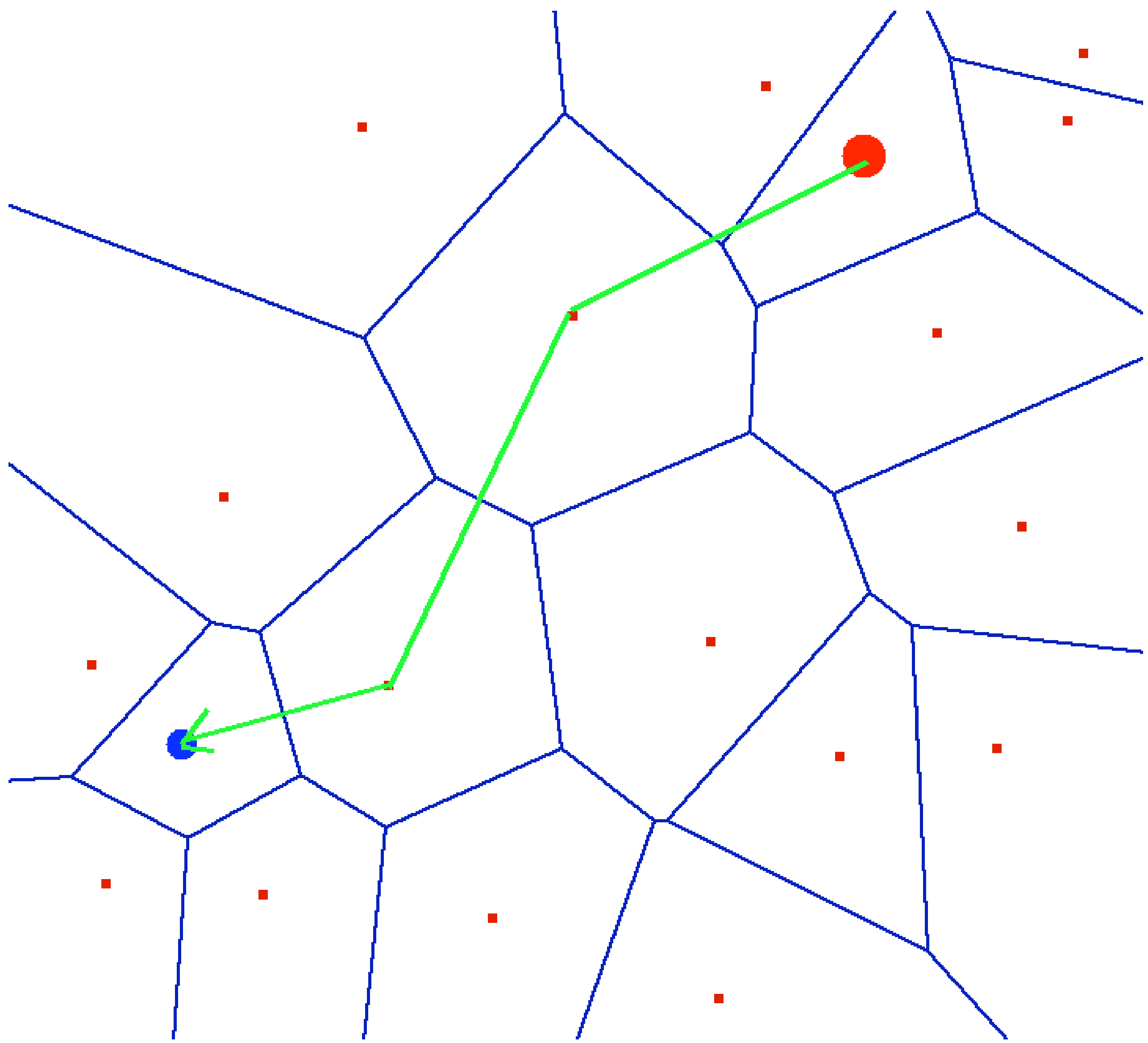
A node exchanges updates with all neighbors.



A node maintain Voronoi of all neighbors (regardless of inside AOI or not)

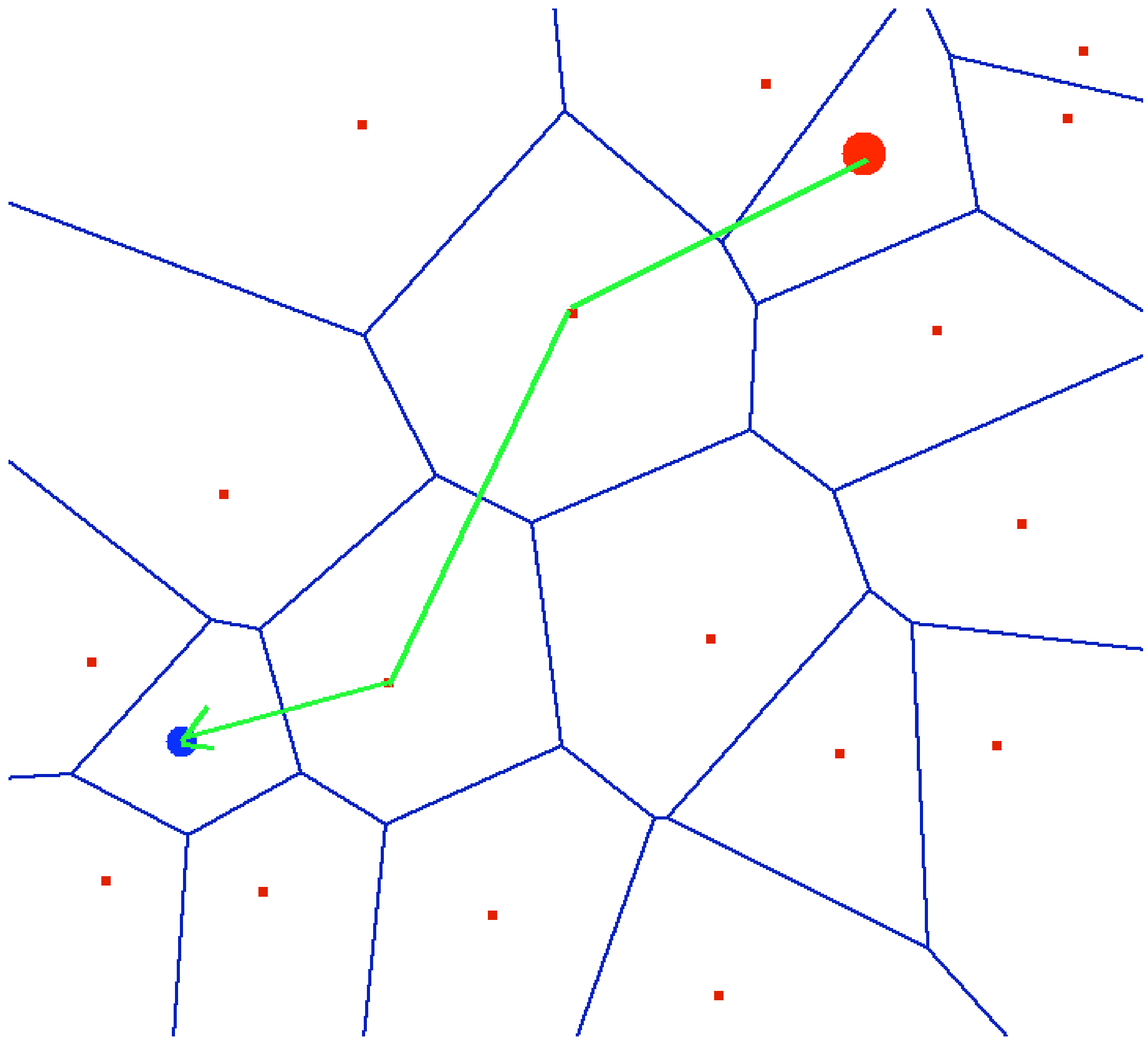
Suppose a player X wants to join. X sends its location to any node in the system.

X join request is forwarded to the node in charge of the region (i.e., closest node to X), called acceptor.

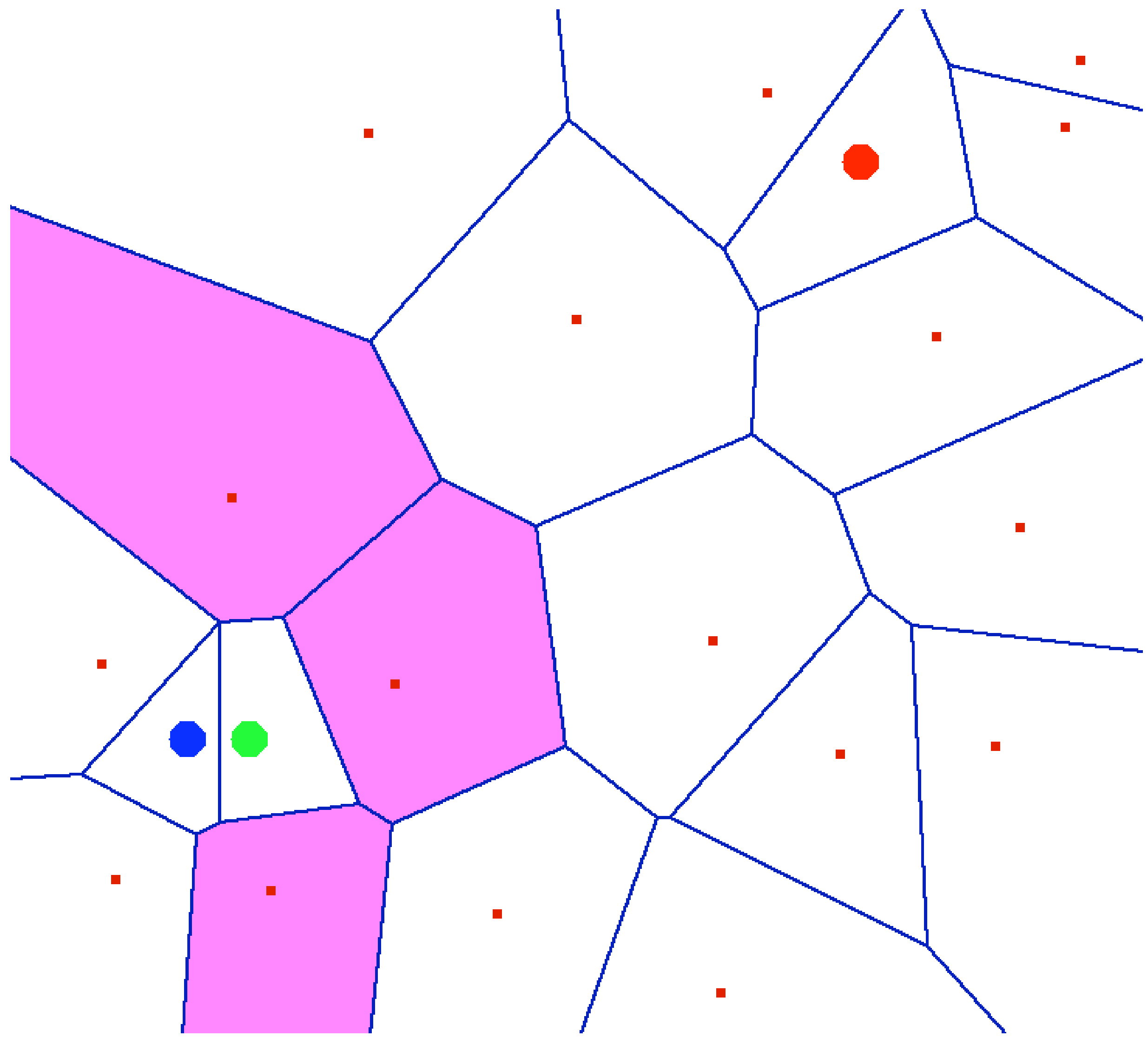


97

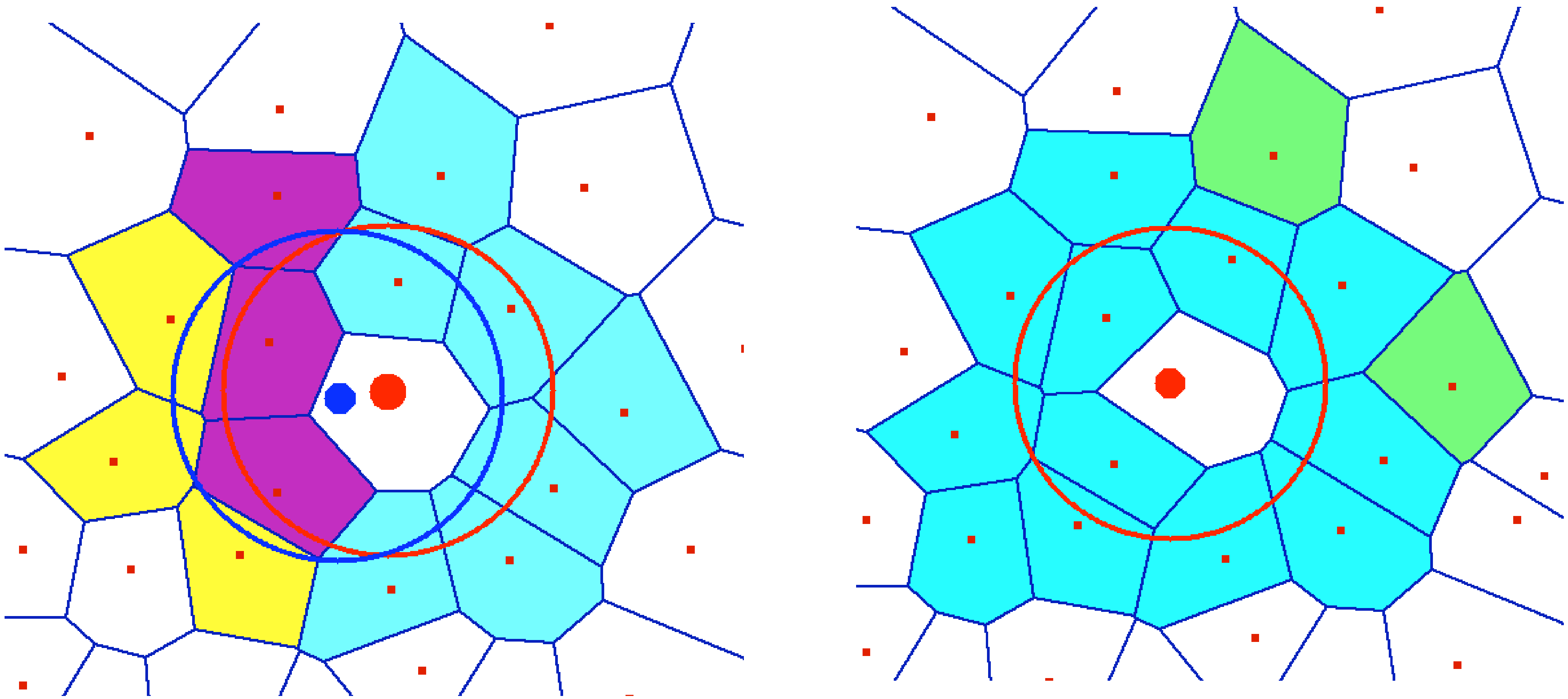
Forwarding is done greedily (every step forward to neighbor closest to X)



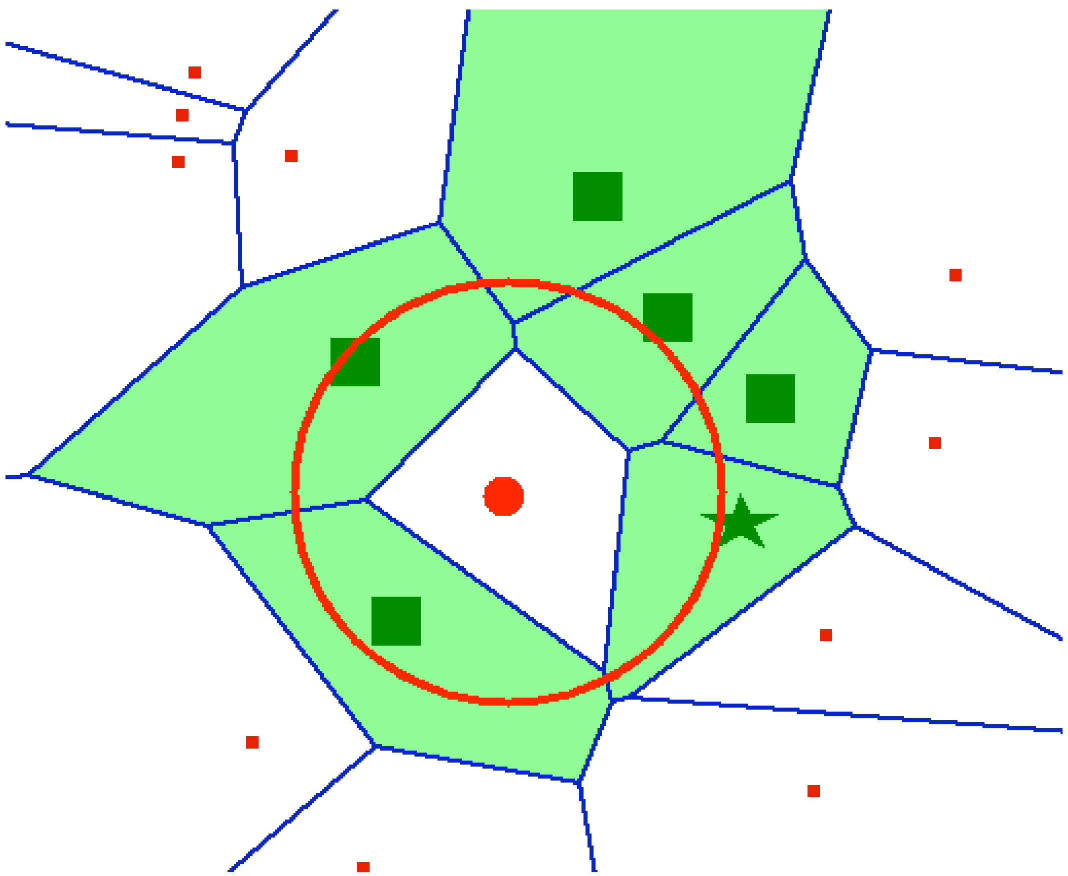
Acceptor inform the joining node X of its neighbors. Acceptor, X, and the neighbors update their Voronoi diagram to include the new node.

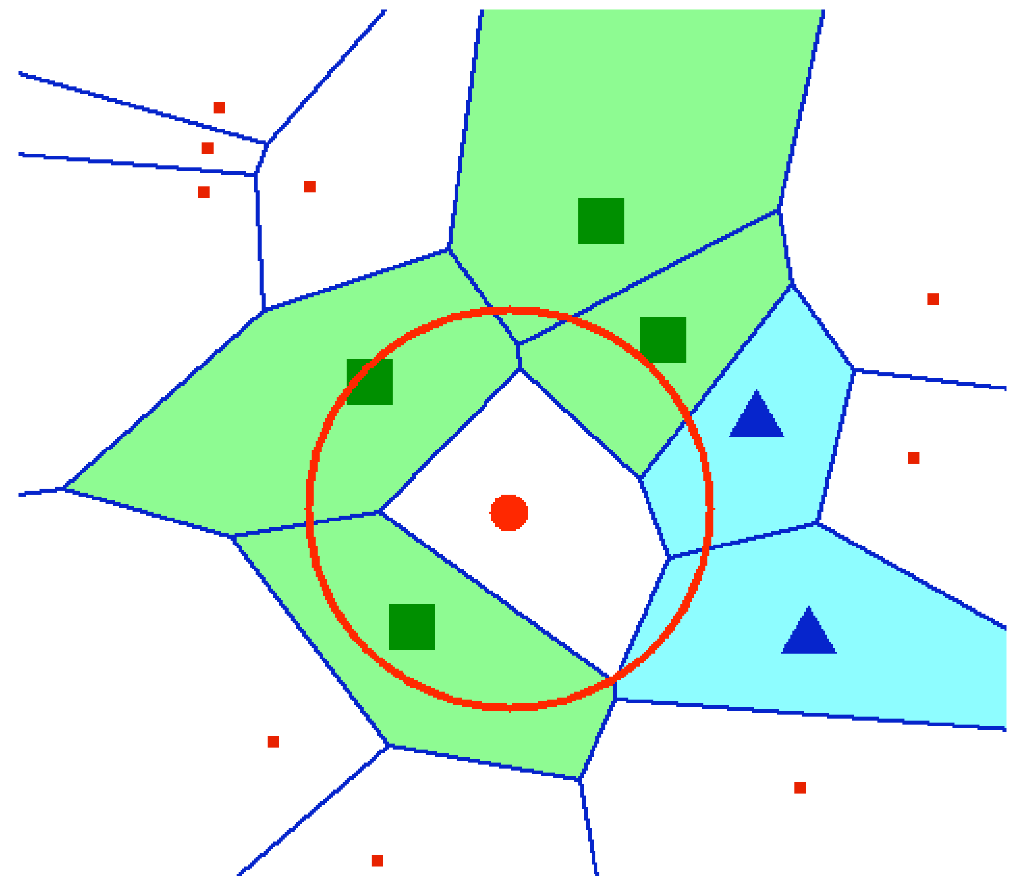


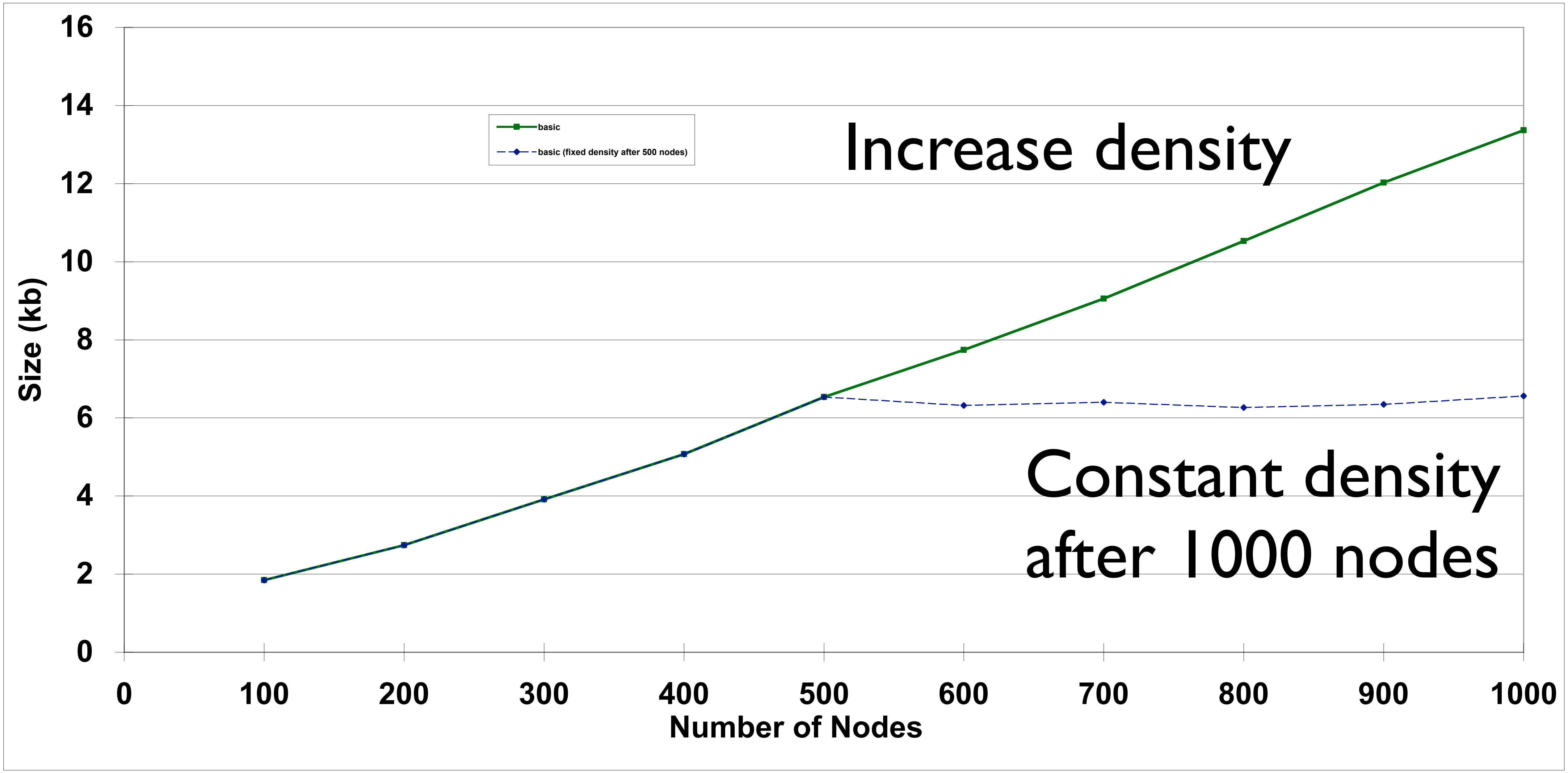
Suppose X moves. Boundary neighbors of X check if their enclosing neighbor is now in X's AOI or has become X's enclosing neighbor. X updates its new neighbor with information about its neighbor. Neighbors outside region is disconnected. Voronoi diagrams are updated.



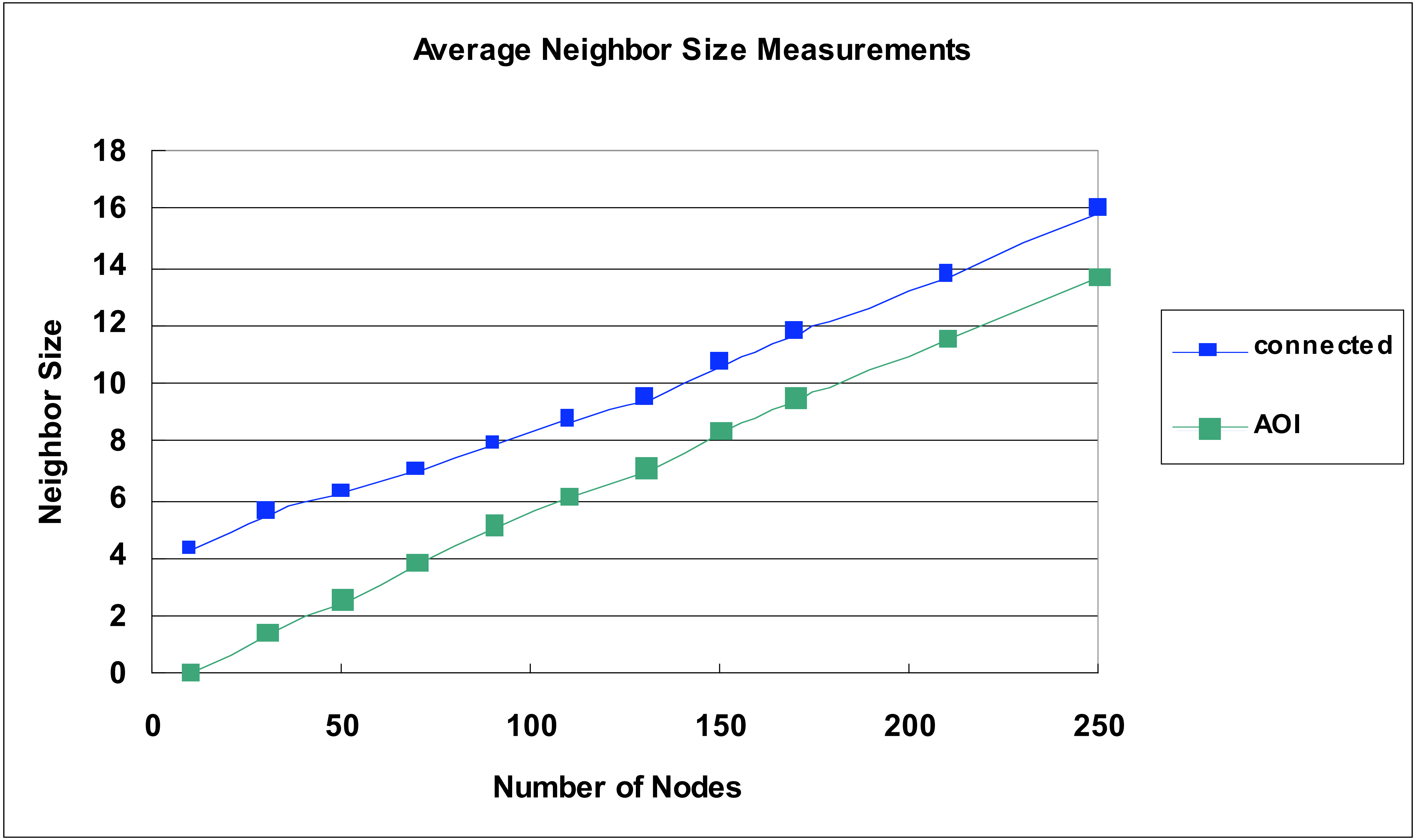
When a node disconnect, Voronoi diagrams are updated by the affected nodes. New boundary neighbors may be discovered.







basic
basic (fixed density after 500 nodes)
Increase density
Constant density
after 1000 nodes
Size (kb)
Number of Nodes
16
14
12
10
8
6
4
2
0
0
100
200
300
400
500
600
700
800
900
1000


Average Neighbor Size Measurements
Neighbor Size
18
16
14
12
10
8
6
4
2
0
0
50
100
150
200
250
Number of Nodes
connected
AOI

Responsive

Consistent

Cheat-Free

Fair

Scalable

Efficient

Robust

Simple