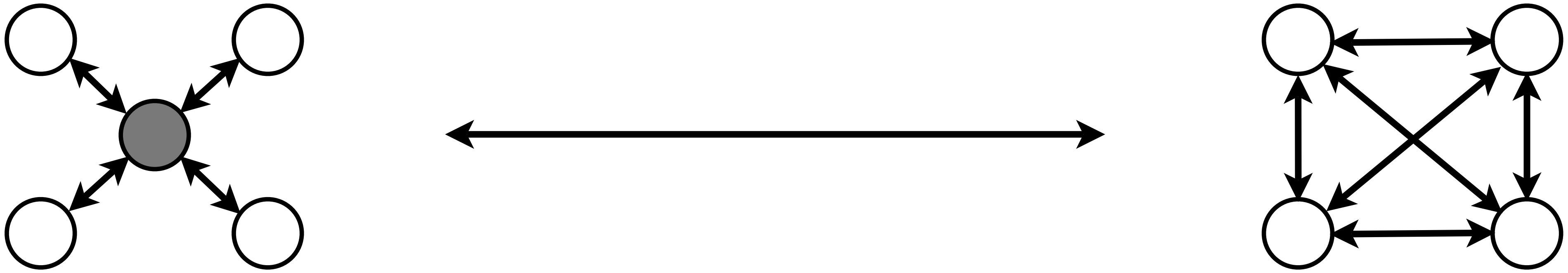
Hybrid Architecture for Networked Games

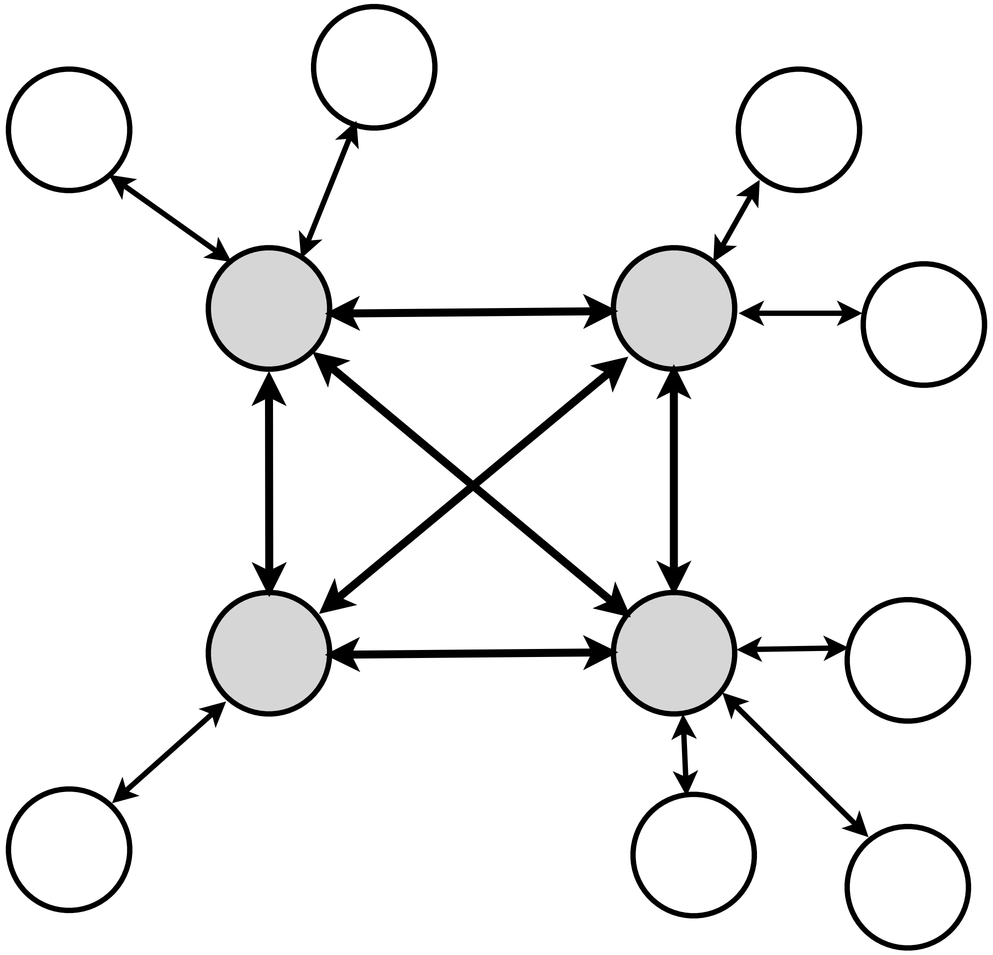


single centralized server

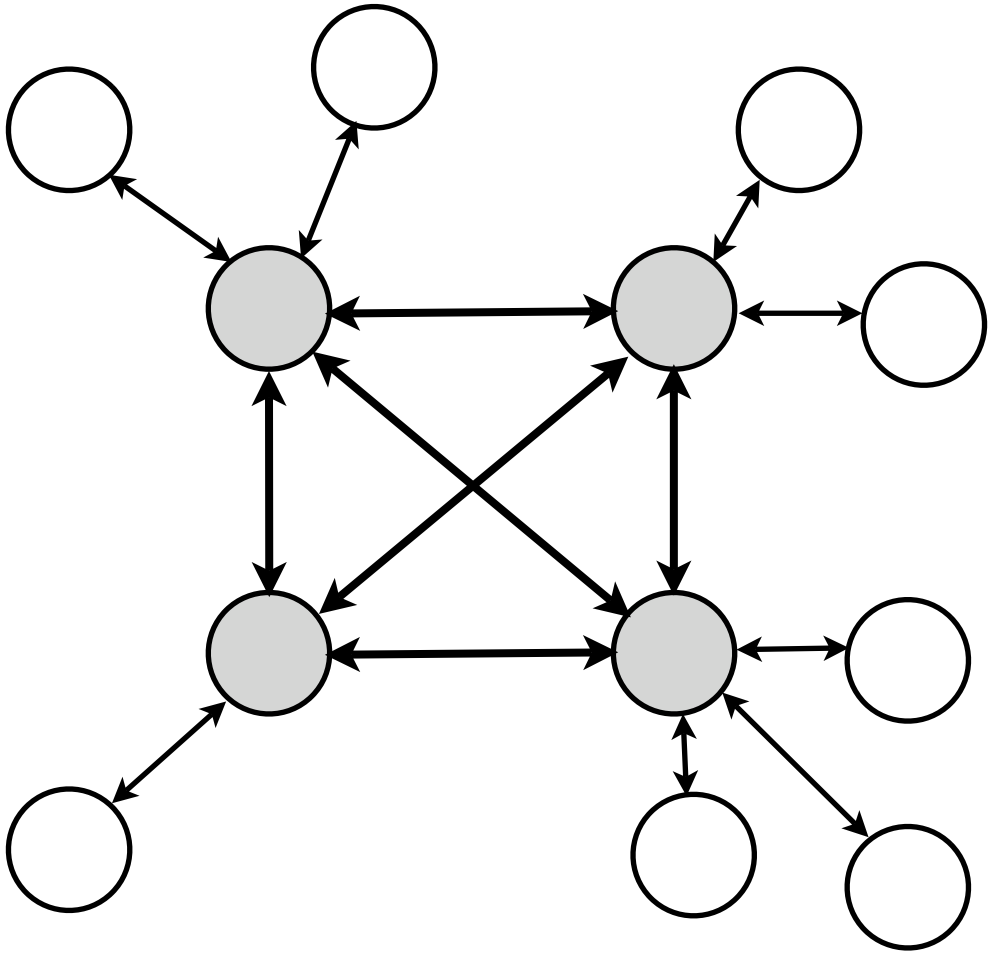
completely decentralized

Mirrored Servers Architecture

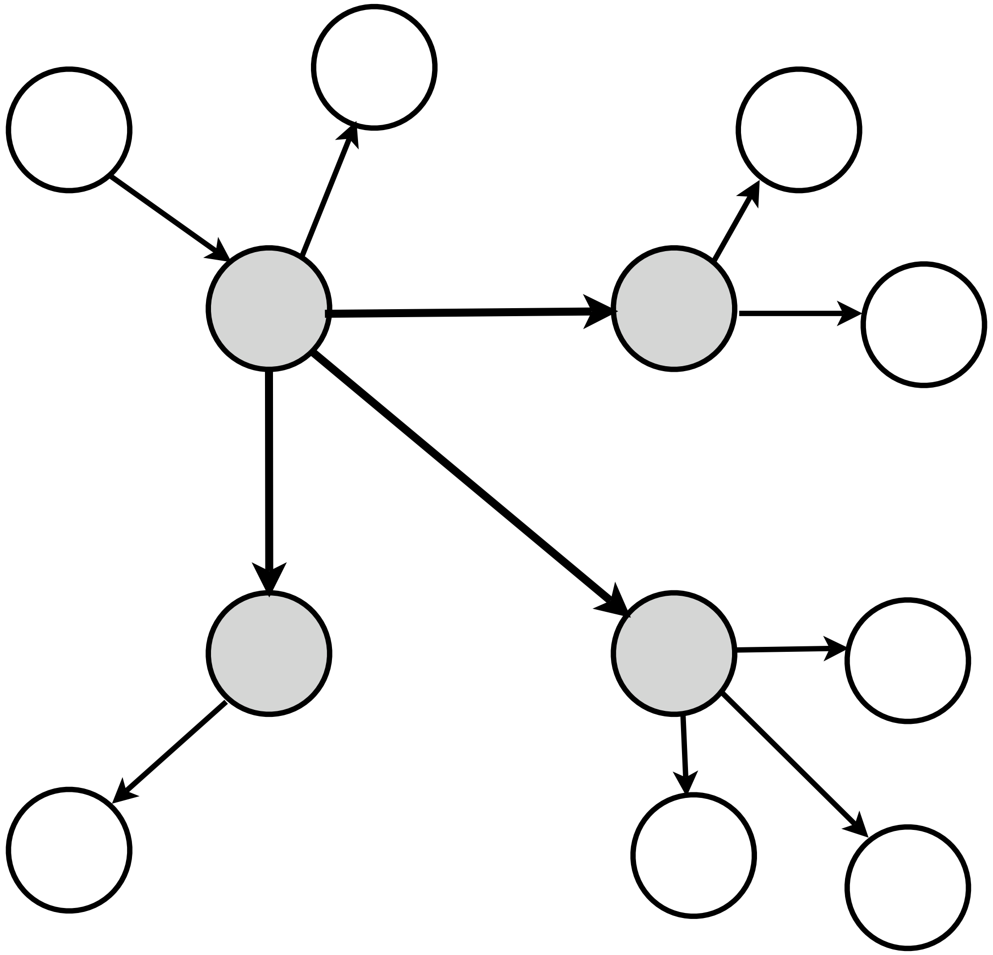
Multiple servers, each replicating the complete game states, serve clients from different geographical regions. Each client connects to one server.



Servers may be connected with high speed, provisioned networked, reducing synchronization latency among the servers.



Updates from a client are sent to its server, which then forwards it to other servers. The servers then forward the updates to all relevant clients.



P2P-style state synchronization among servers is easier, since

1. servers can be trusted
2. clocks can be synchronized
3. IP multicast may be used
4. higher availability

Pros and Cons of Mirrored Servers?

Advantage: No more single point of failures -- client can use another replicated servers if their nearest server fails.

Advantage: Each server can handle more clients (if networking bandwidth is the bottleneck)

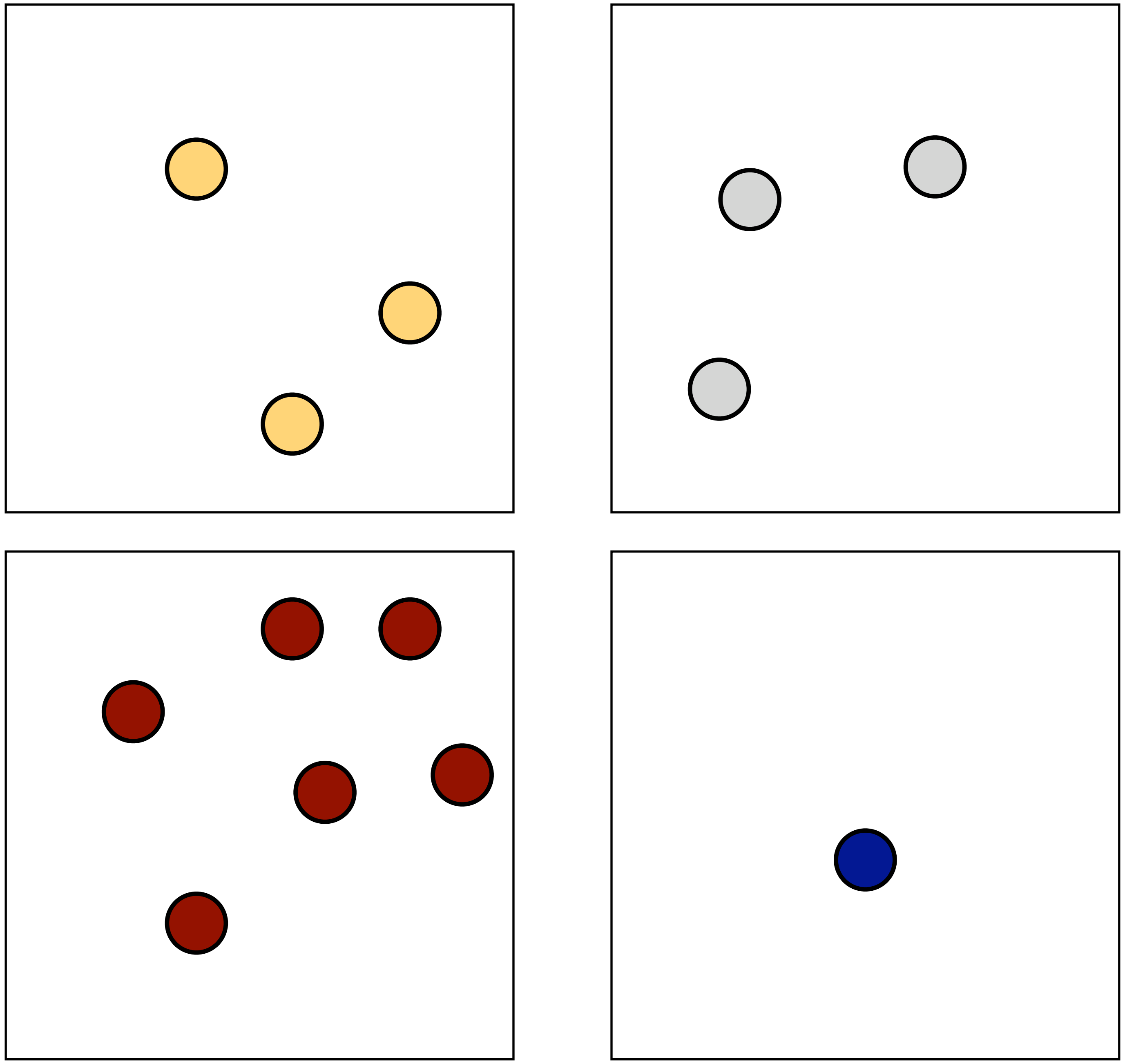
Advantage: Lower latency from client to its server

Cons: Latency to update other players increases due to one additional forwarding.

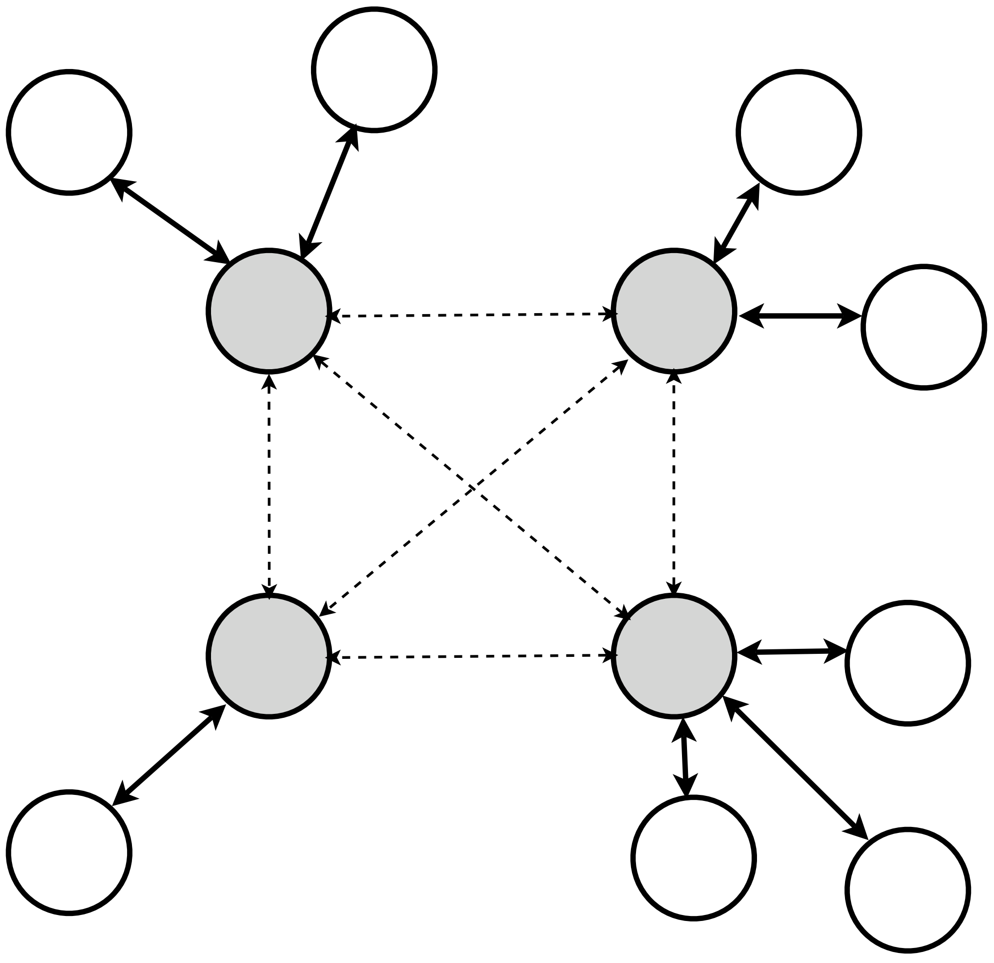
Cons: Scalability is still limited since every server maintains states of the whole game world (if computation is the bottleneck).

Zoned Servers Architecture

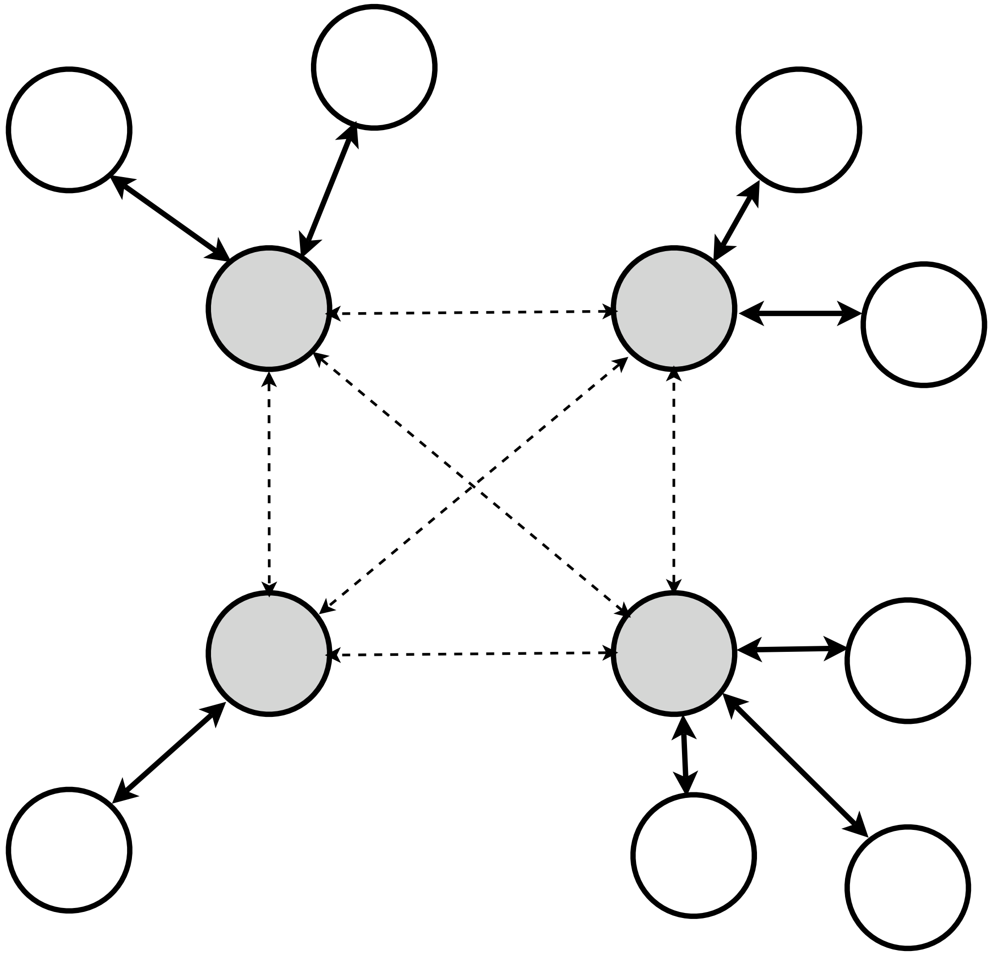
Divide the game world into independent regions. One server is in-charged of one region.



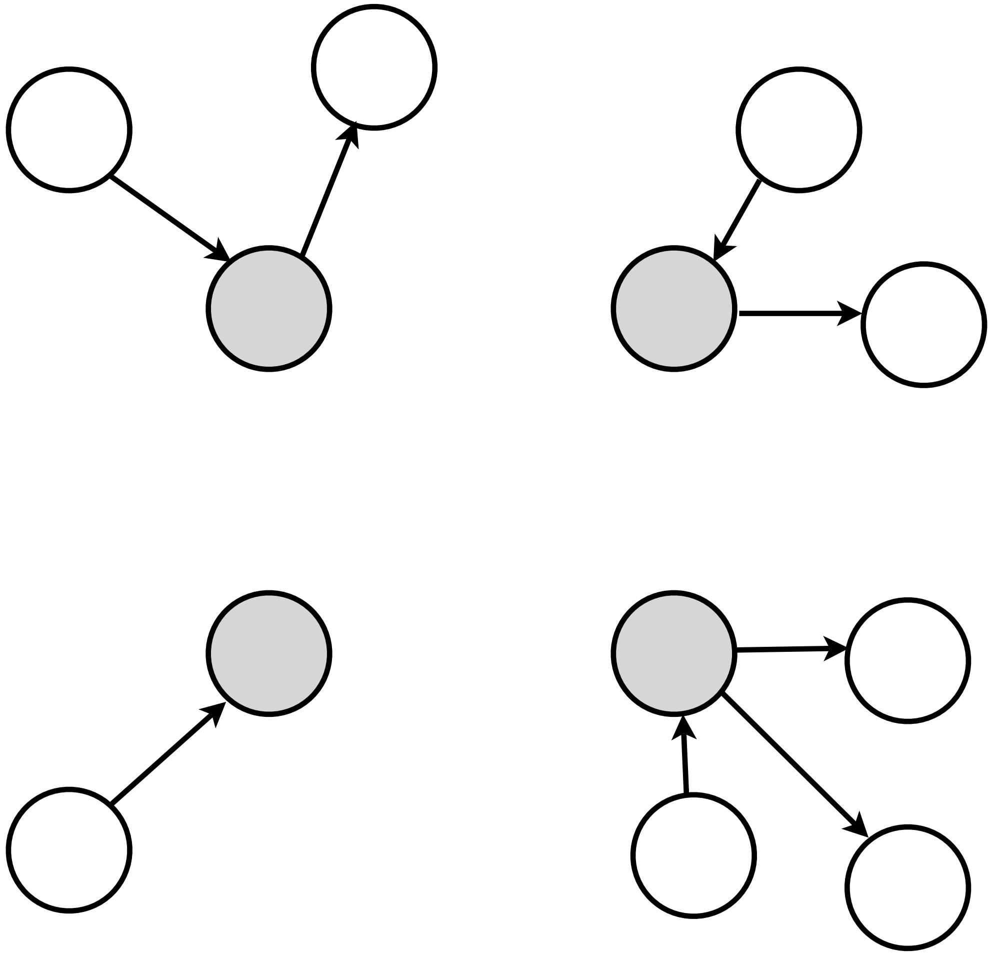
A client connects to the server serving its current region, and is handed-off to another server when it moves to another region.

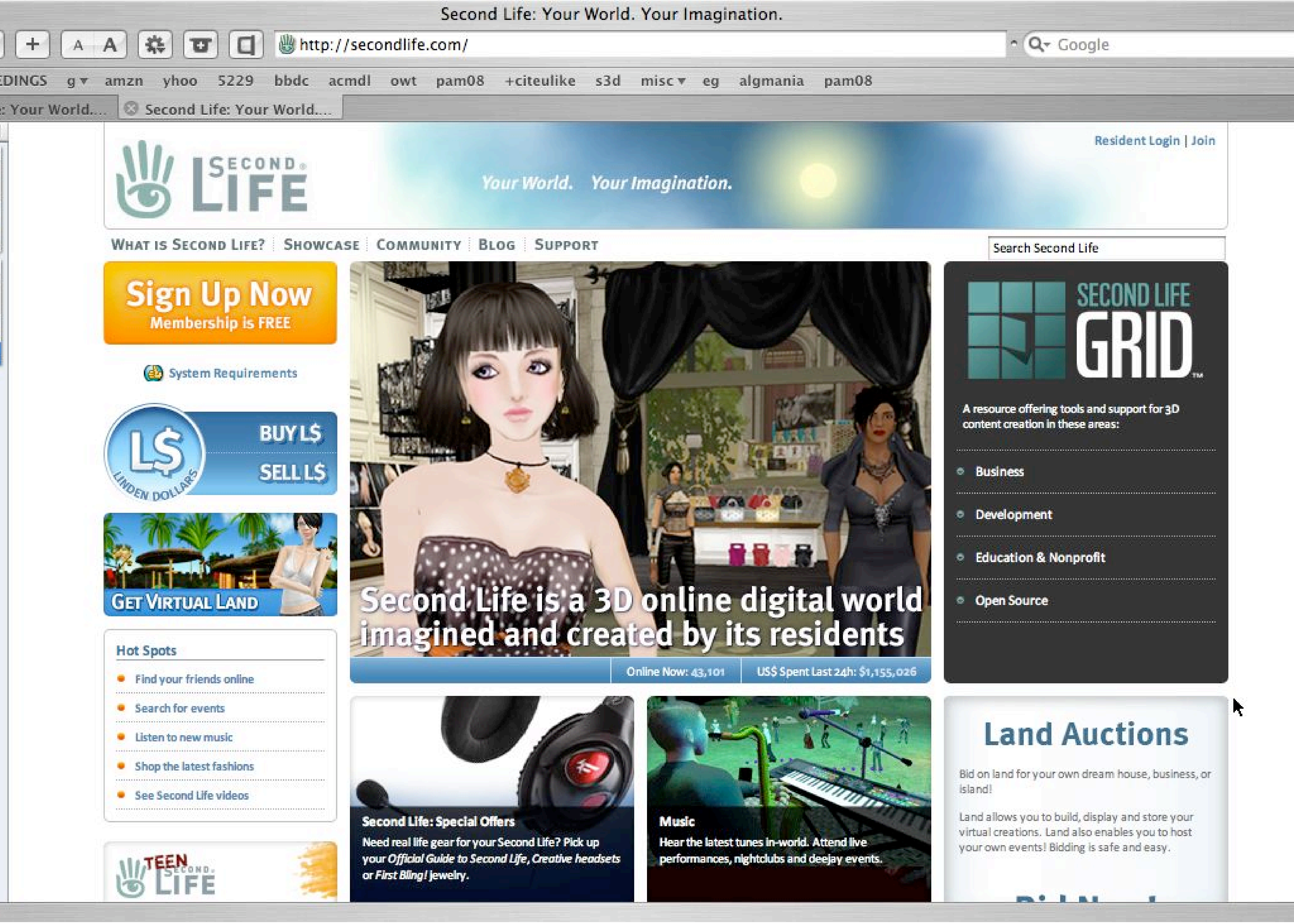


Servers seldom communicate with each other.



Updates from a client are sent to its server, which then forwards it to all relevant clients in the same region.





Second Life: Your World. Your Imagination.

http://secondlife.com/

Resident Login | Join

Your World. Your Imagination.

What is Second Life? | Showcase | Community | Blog | Support

Search Second Life

Sign Up Now
Membership is FREE

System Requirements

BUY L$
SELL L$
LINDEN DOLLARS

GET VIRTUAL LAND

Hot Spots

- Find your friends online
- Search for events
- Listen to new music
- Shop the latest fashions
- See Second Life videos

TEEN SECOND LIFE

Second Life is a 3D online digital world imagined and created by its residents

Online Now: 43,101 | US$ Spent Last 24h: $1,155,026

SECOND LIFE GRID

A resource offering tools and support for 3D content creation in these areas:

- Business
- Development
- Education & Nonprofit
- Open Source

Second Life: Special Offers
Need real life gear for your Second Life? Pick up your *Official Guide to Second Life, Creative headsets* or *First Bling!* jewelry.

Music
Hear the latest tunes in-world. Attend live performances, nightclubs and deejay events.

Land Auctions

Bid on land for your own dream house, business, or island!

Land allows you to build, display and store your virtual creations. Land also enables you to host your own events! Bidding is safe and easy.

A A
Google
DINGS g ▾ amzn yhoo 5229 bbdc acmdl owt pam08 +citeulike s3d misc ▾ eg algmania pam08
: Your World....
Second Life: Your World....

Pros and Cons of Zoned Servers?

Advantage: Each server can handle more clients (if computation is a bottleneck)

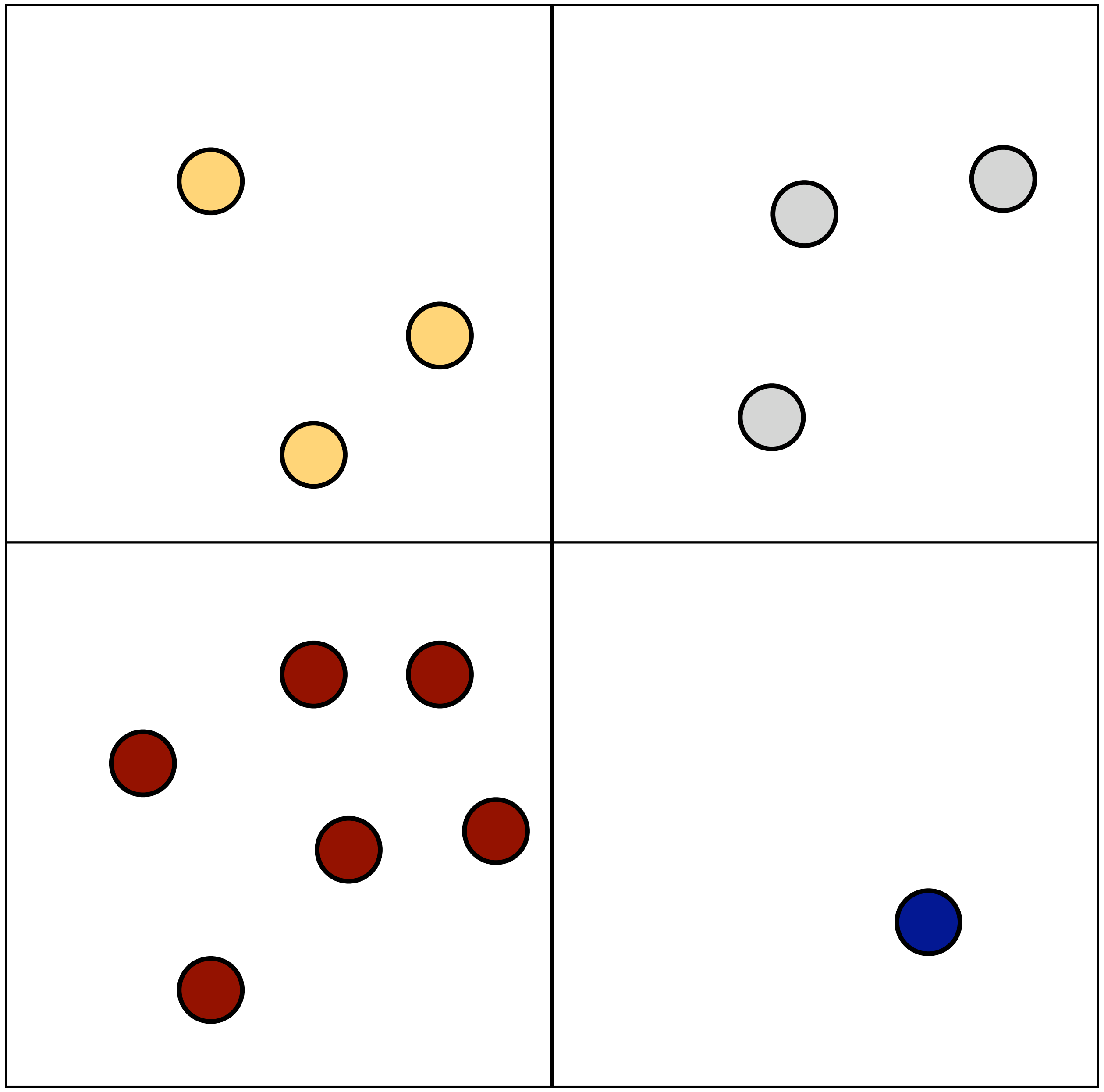
Cons: Single point of failure for each region.

(But we can have replicated servers for each region)

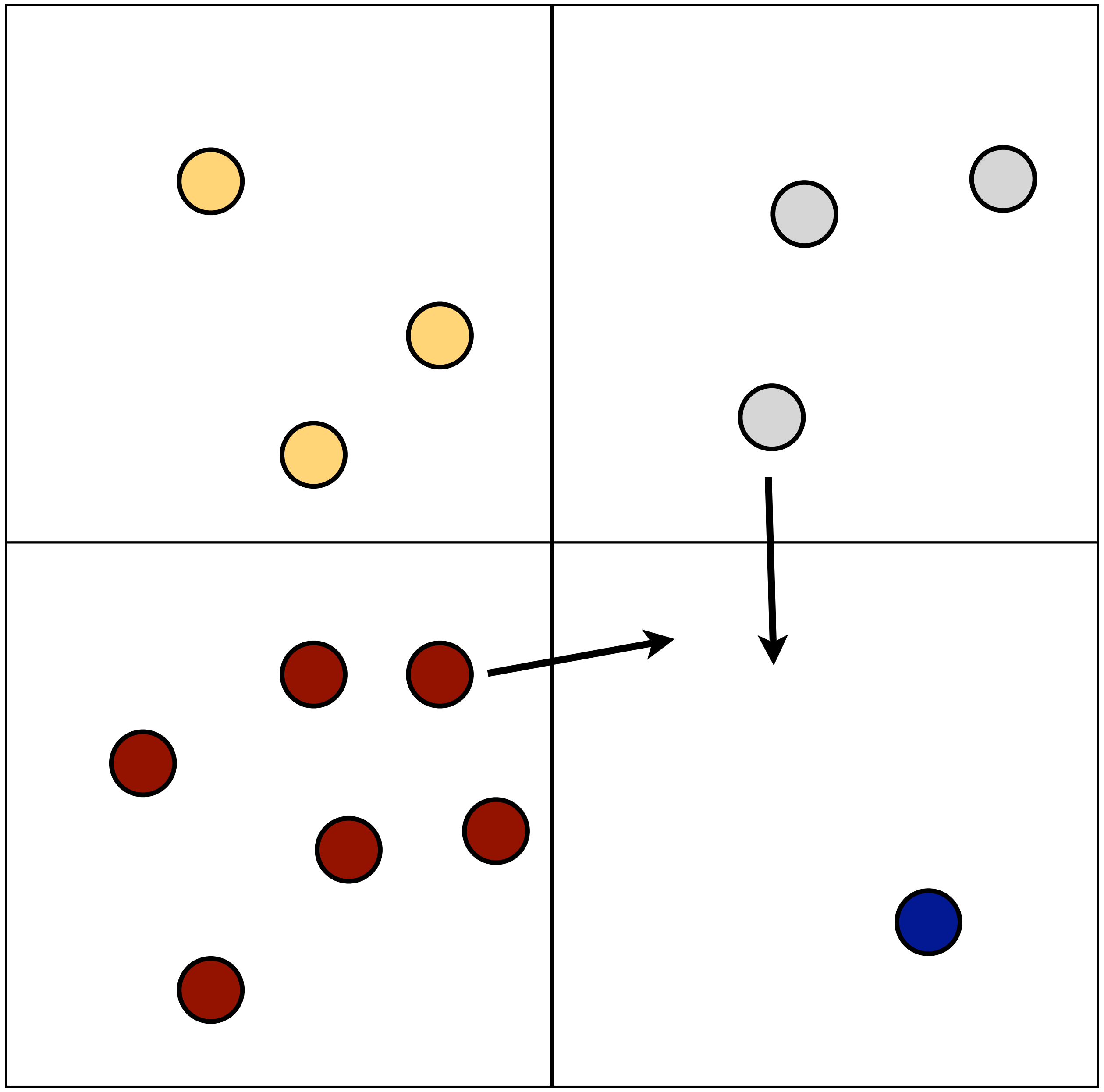
Cons: Severely constraints the design of games.

Supporting Seamless Game Worlds

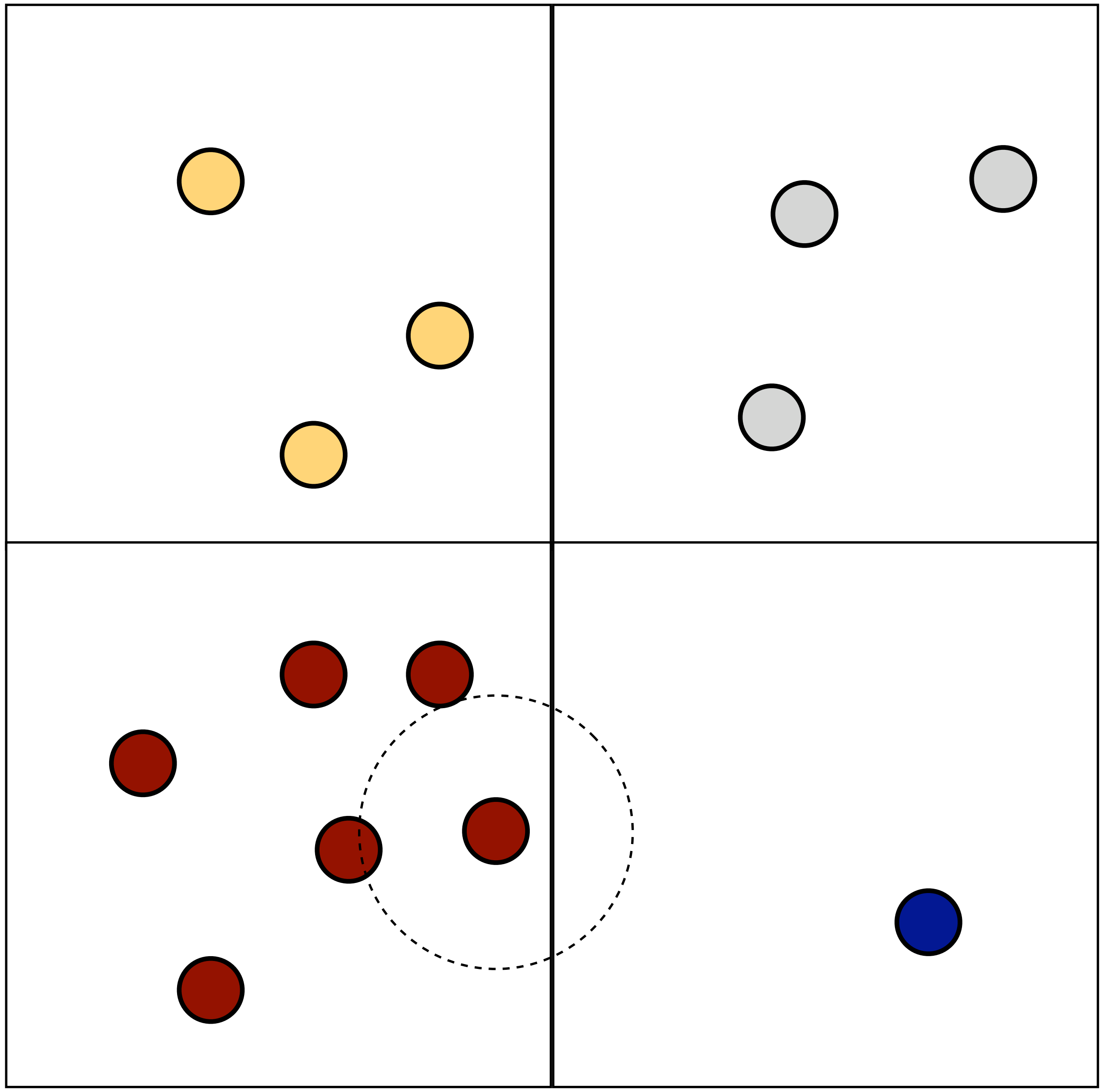
Divide the game world into regions. One server is in-charged of one region.



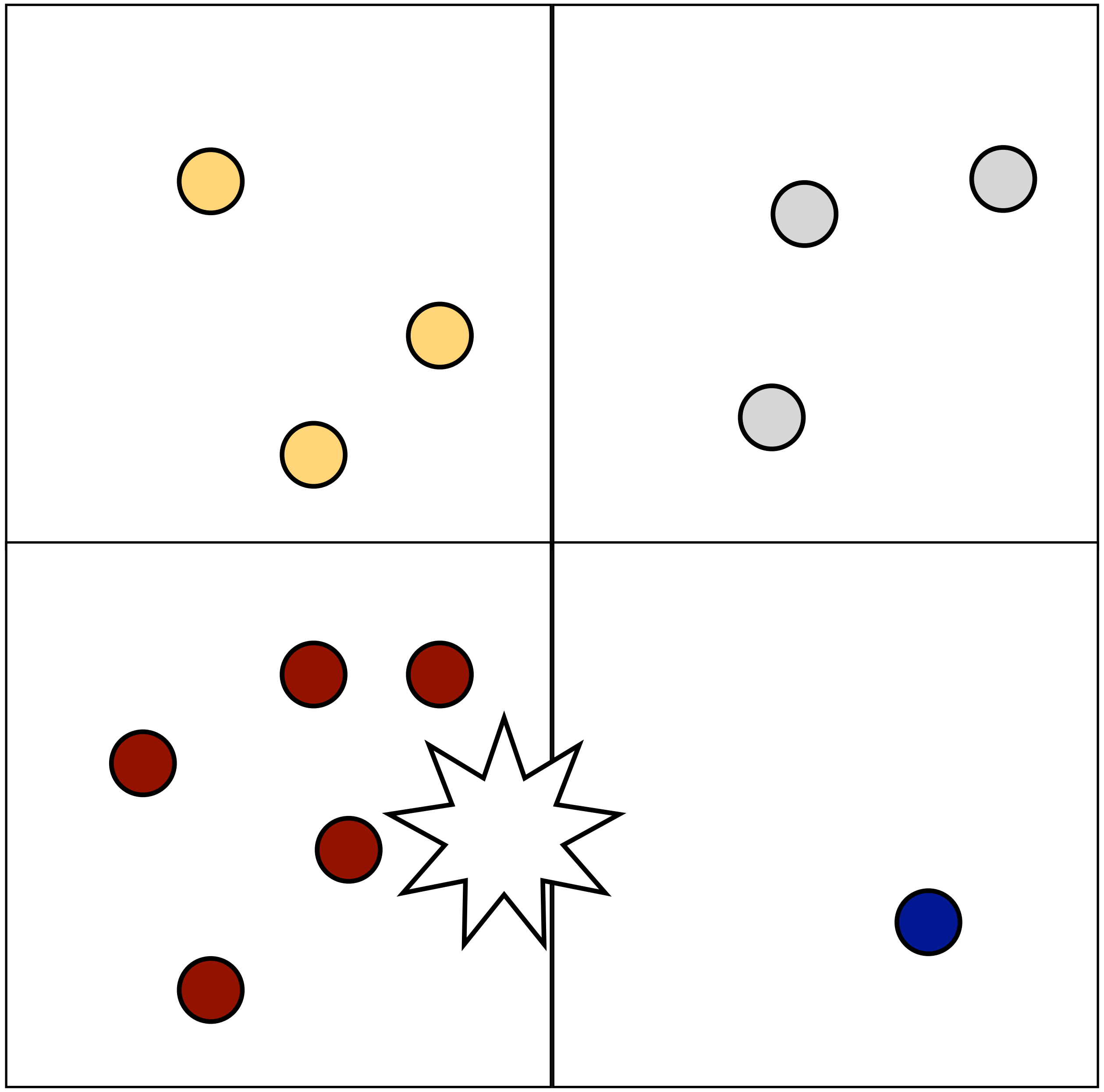
Player may move around the world seamlessly



A client may see events and objects from neighboring regions.

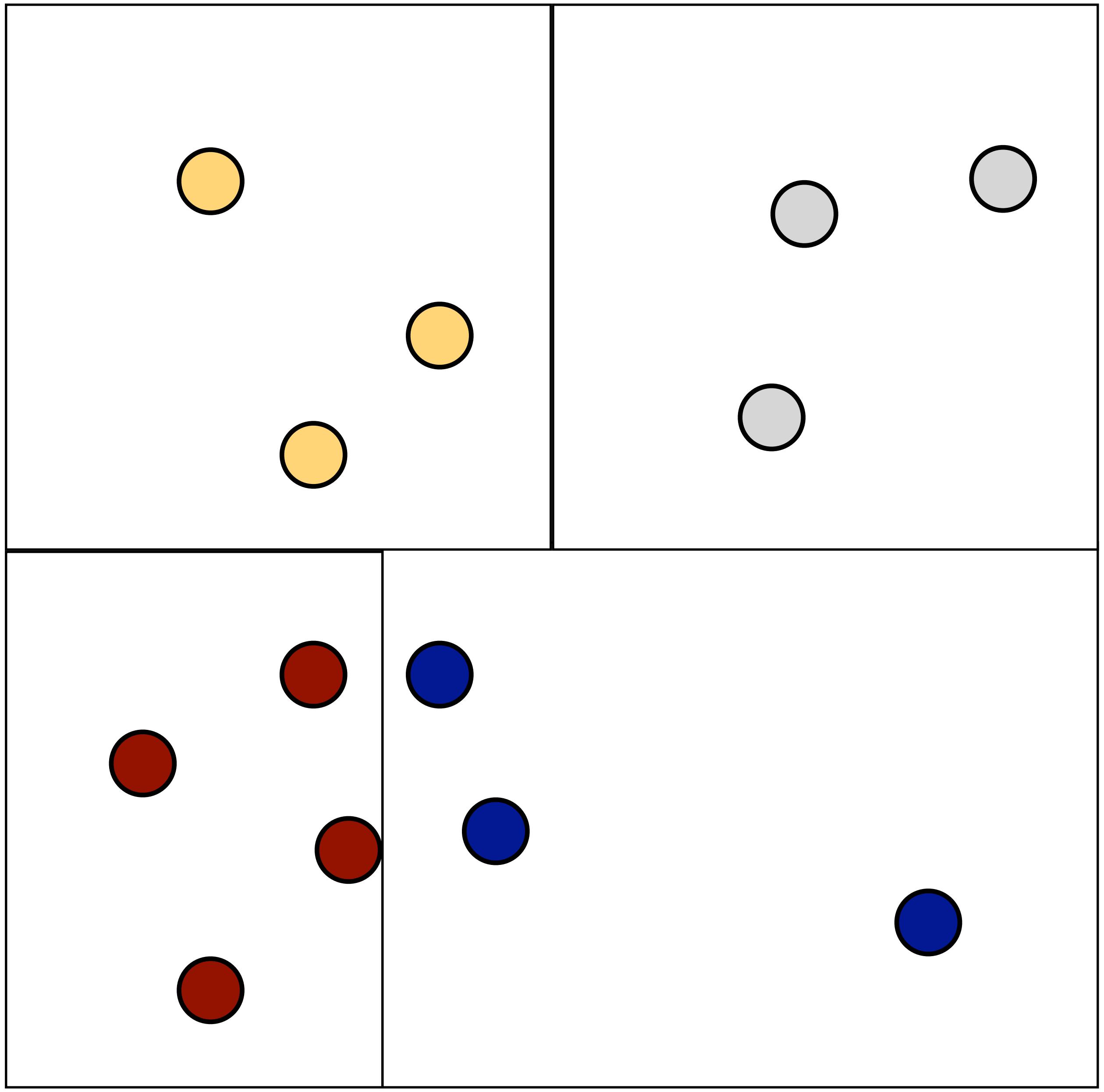


Events occurring in one region may affect objects in another region.

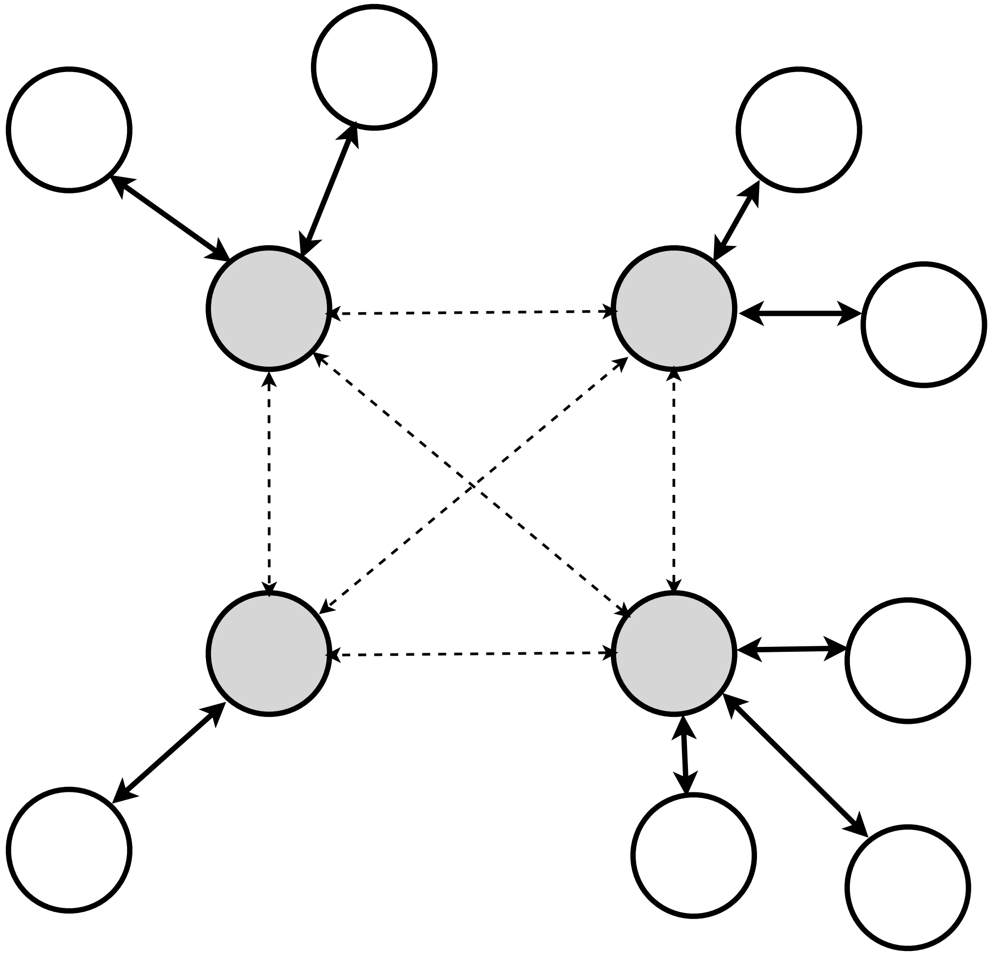


Advantage: Partitioning is transparent to the players. No artificial constraints in game design.

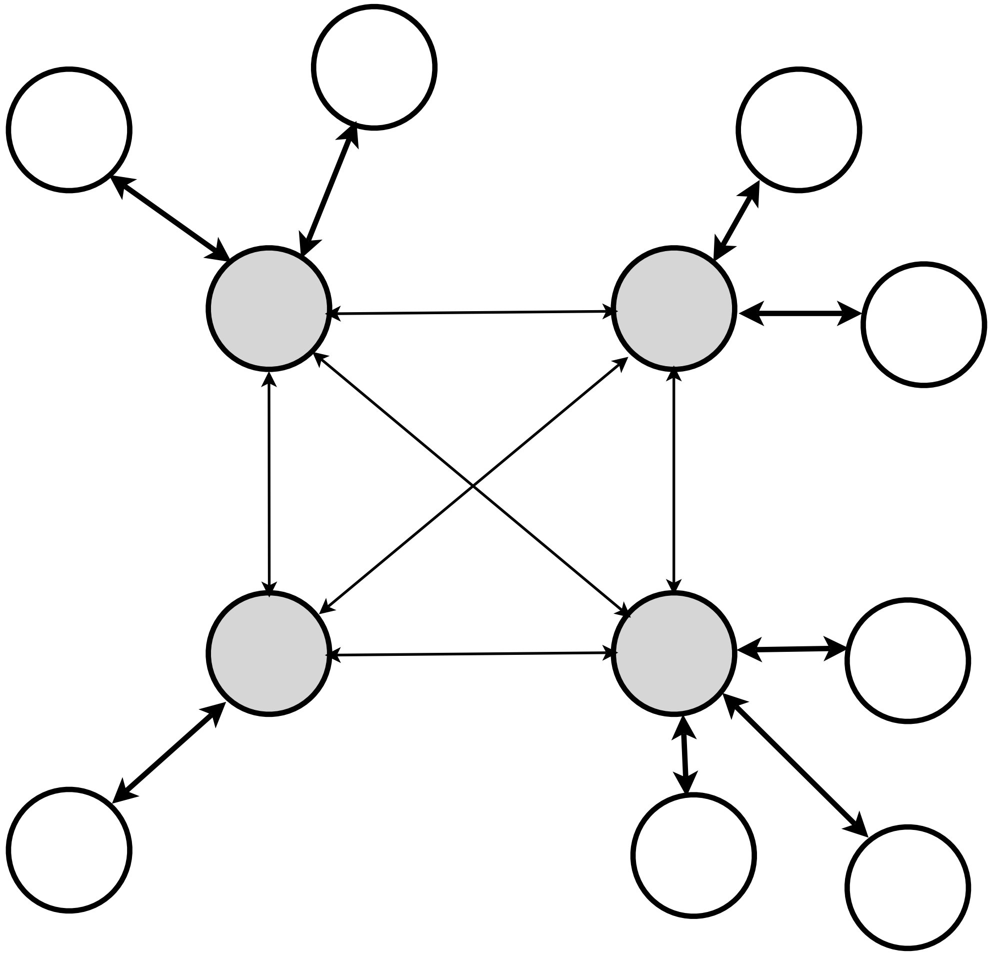
Advantage: We can now resize the regions to load-balance among the servers, improving server utilization.



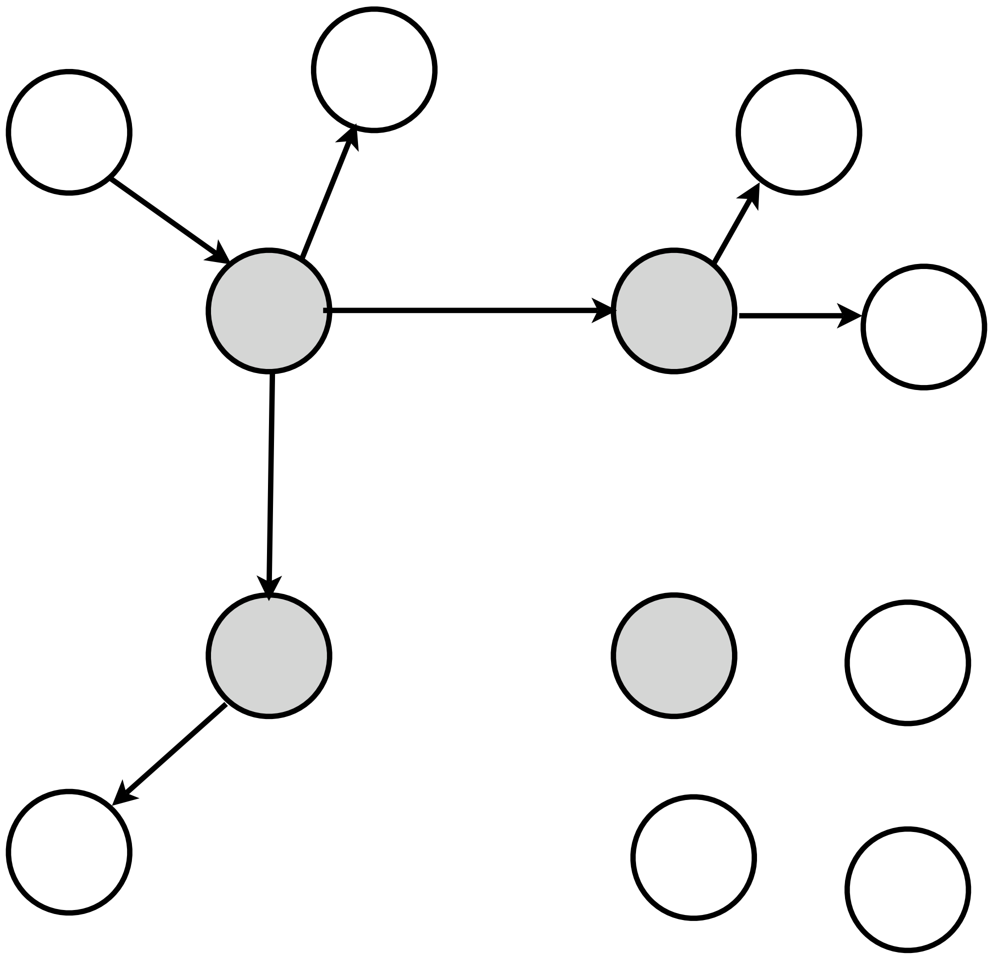
A client connects to the server serving its current region, and is handed-off to another server when it moves to another region.



Server communicates with each other to transfer states, update states etc.

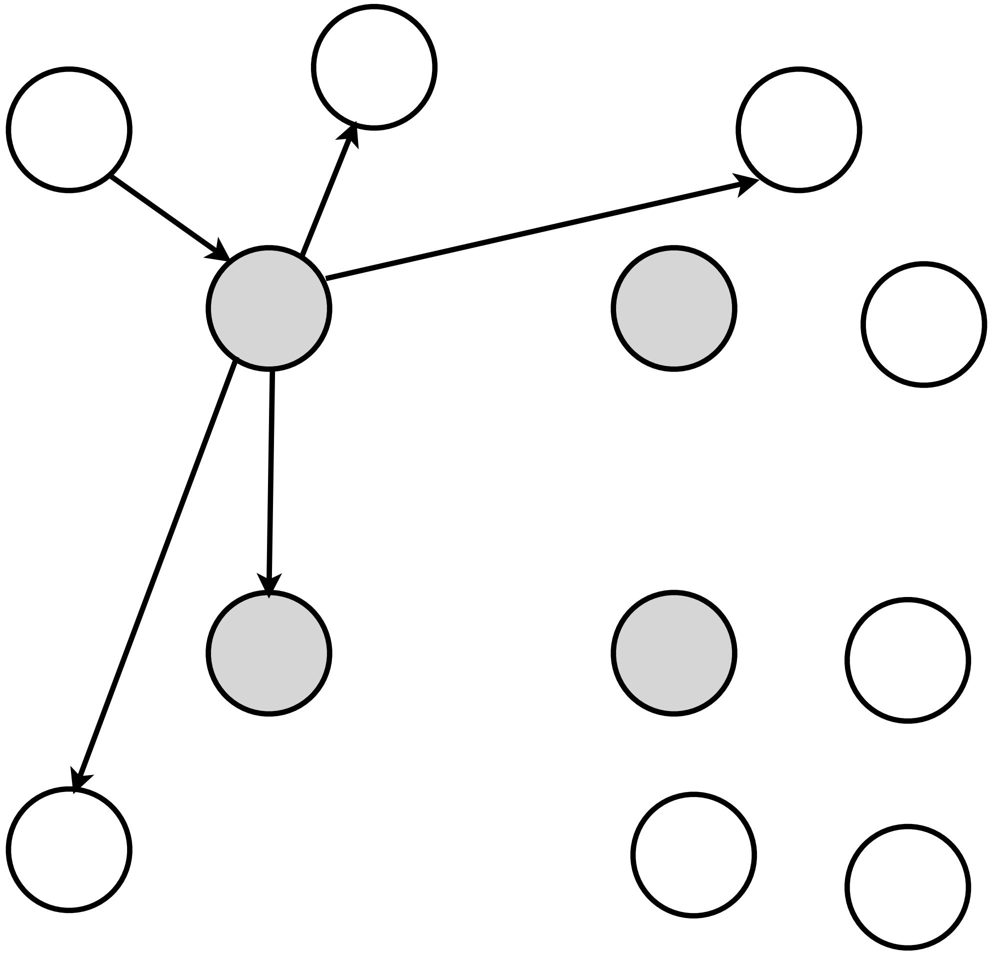


Updates from a client are sent to its server, which then forwards it to all relevant clients in the same region or neighboring regions.



or

A client connects to the servers serving its current region and regions it subscribes to. Updates from neighboring server are sent directly.

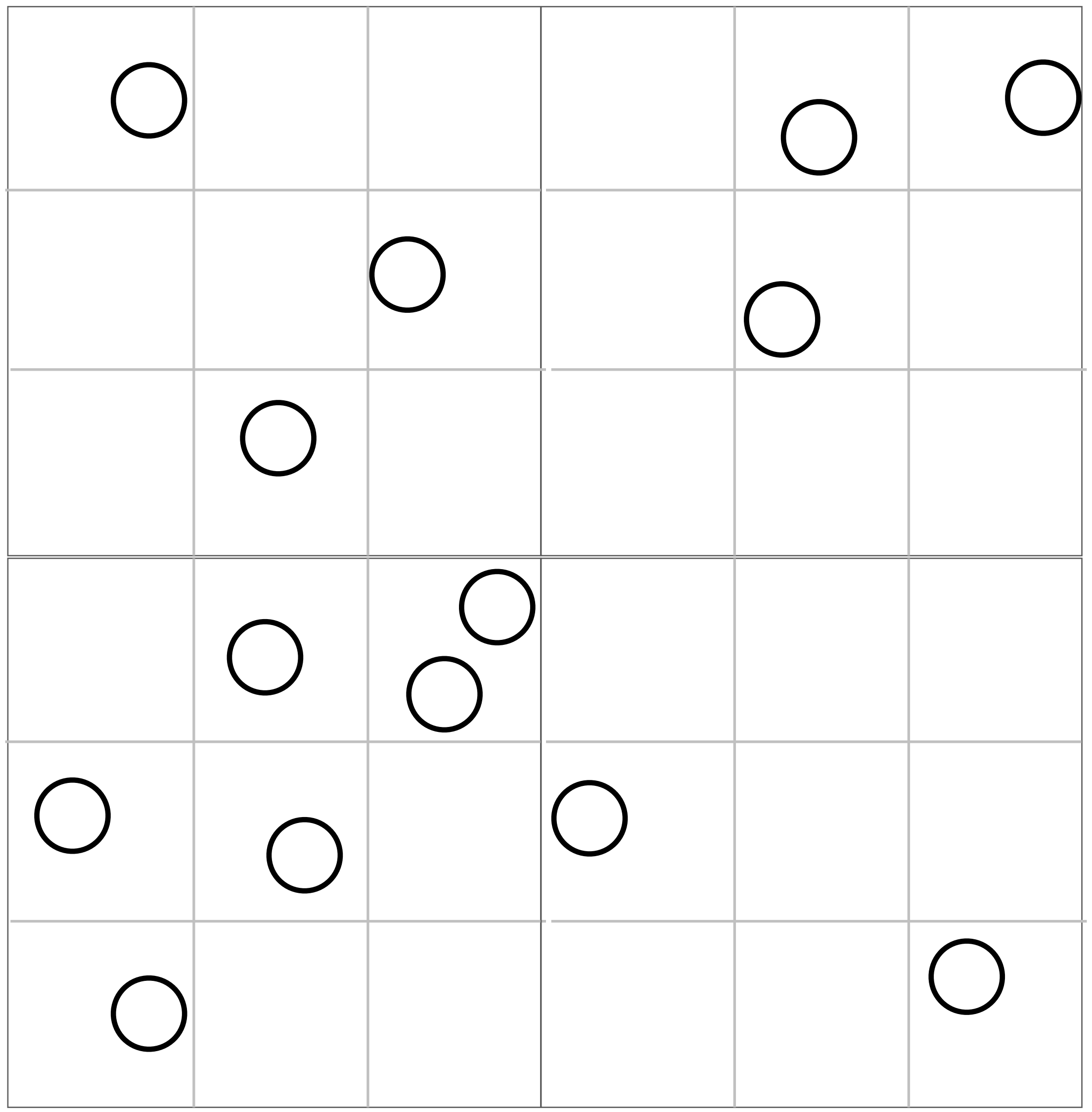


Q: How to partition the game world?

Ideally: each server should handle the same number of players

Ideally: minimize communications between servers

Idea: divide the game world into small cells and assign group of cells to servers.



I will simplify the representation to a grid of cells with number indicating the number of players (or load incurred on server) in this cell.

2	0	3
3	1	4
1	1	0

Centralized approach to partitioning

Balanced Deployment Algorithm

by Bart De Vleeschauwer et al

Idea: sort the cells and assign them one-by-one, highest load first, to the servers to minimize the maximum server load

Suppose we have only two servers (blue and red).

2	0	3
3	1	4
1	1	0

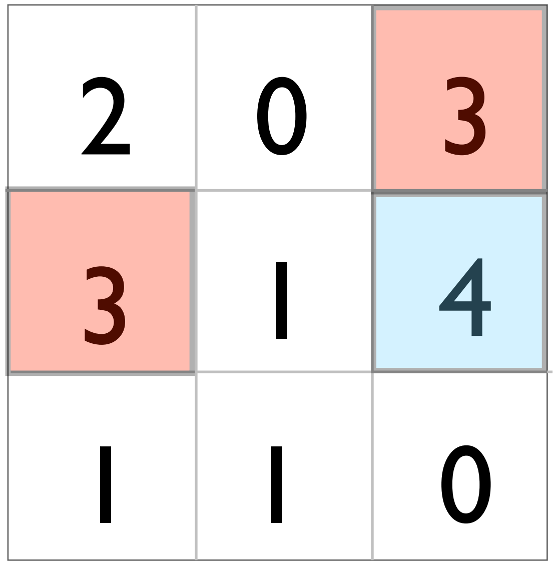
Blue’s load = 4   Red’s load = 0

2	0	3
3	1	4
1	1	0

Blue’s load = 4   Red’s load = 3

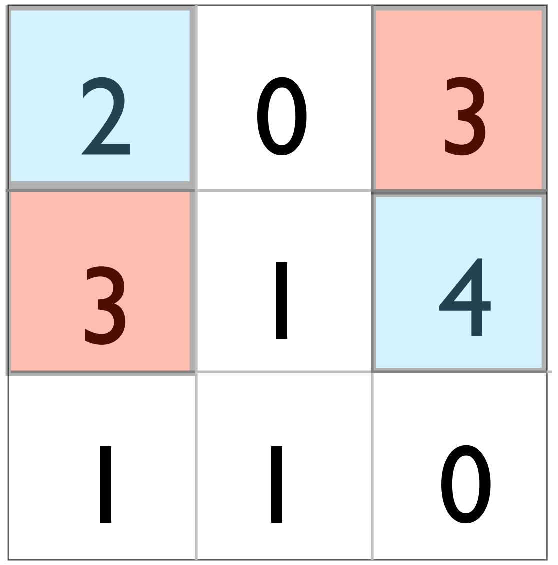
2	0	3
3	1	4
1	1	0

Blue’s load = 4   Red’s load = 6



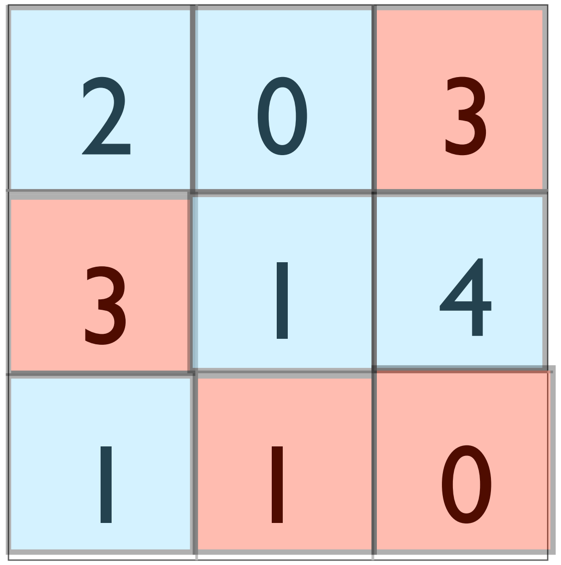
2	0	3
3	1	4
1	1	0

Blue's load = 6   Red's load = 6



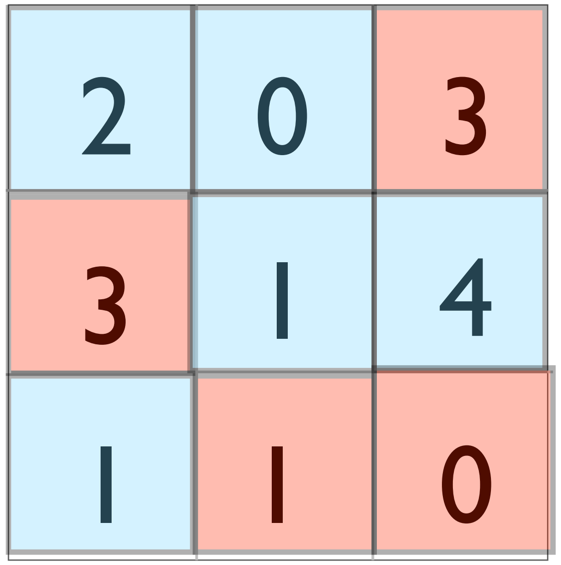
2	0	3
3	1	4
1	1	0

Blue’s load = 8   Red’s load = 7



2	0	3
3	1	4
1	1	0

Problem: neighboring cells are not adjacent. There are 8 “borders” in this solution. Communication overhead is not considered.



2	0	3
3	1	4
1	1	0

Let's define a cost function for assigning a cell c to a server S:

cost(c,S) =

computation load increase in S + communication load increase in S

For simplicity,

CPU load = players,

newtork load = borders

Clustered Deployment Algorithm

by Bart De Vleeschauwer et al

Start with each cell as one cluster

Repeatedly merge two adjacent clusters with least overhead until number of clusters equals to number of servers.

2	0	3
3	1	4
1	1	0

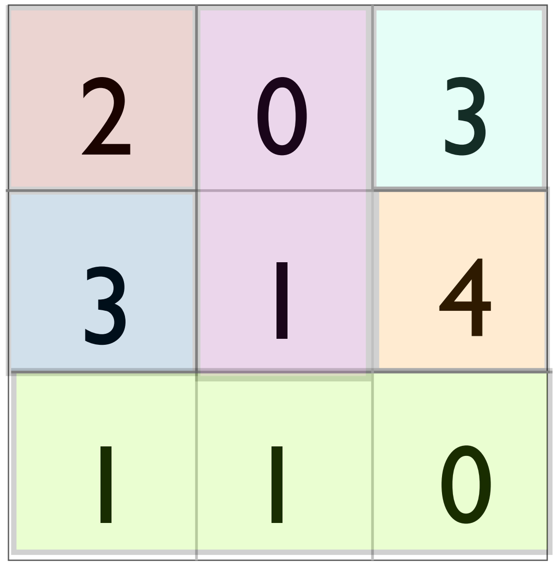
Least cost = 1 + 3 = 4

2	0	3
3	1	4
1	1	0

Least cost = 2 + 3 = 5

2	0	3
3	1	4
1	1	0

Least cost = 1 + 5 = 6



2	0	3
3	1	4
1	1	0

Least cost = 3 + 5 = 8

2	0	3
3	1	4
1	1	0

2	0	3
3	1	4
1	1	0

Problem: Just a heuristic -- not optimal (the optimization problem is NP-complete)

Problem: Last few steps involve heavily loaded cells and can create uneven deployment.

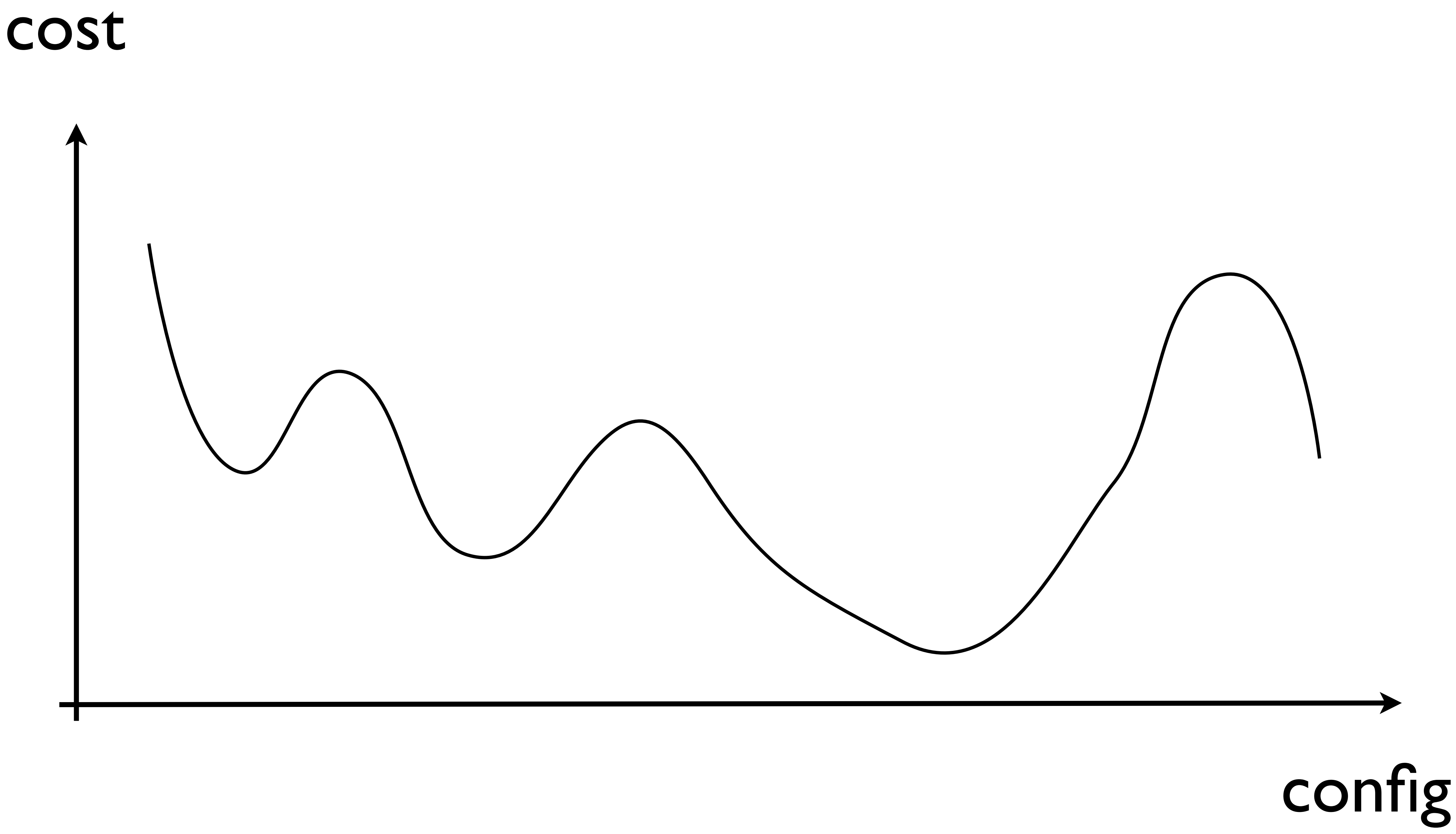
We can tweak this heuristic in many ways to improve it, but it’s still a heuristic.

Simulated Annealing

by Bart De Vleeschauwer et al

A general method for finding good solution in an optimization problem (meta-heuristic)

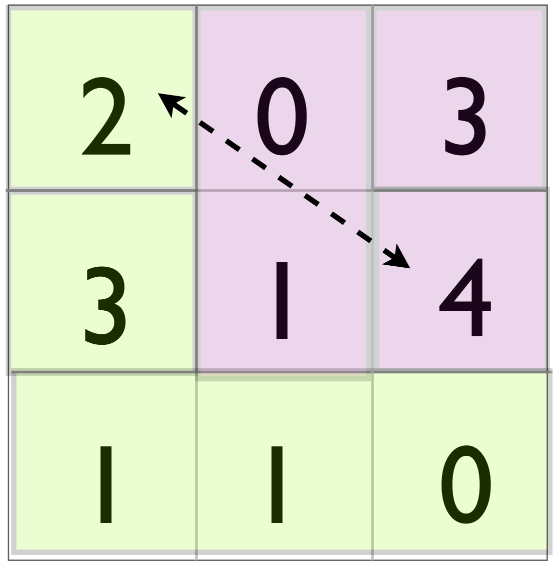
At every step, move towards a nearby, better configuration probabilistically (allowing a move towards worse solution, prevent stuck at local minimum)


cost
config

Start with a solution from a heuristic.

Move to a nearby config: swap cells with an adjacent server, or move cell from one server to another

Is it a better solution?
If so, keep it as current solution
with some probability.



2	0	3
3	1	4
1	1	0

Experimental results show that Simulated Annealing works best among the methods.

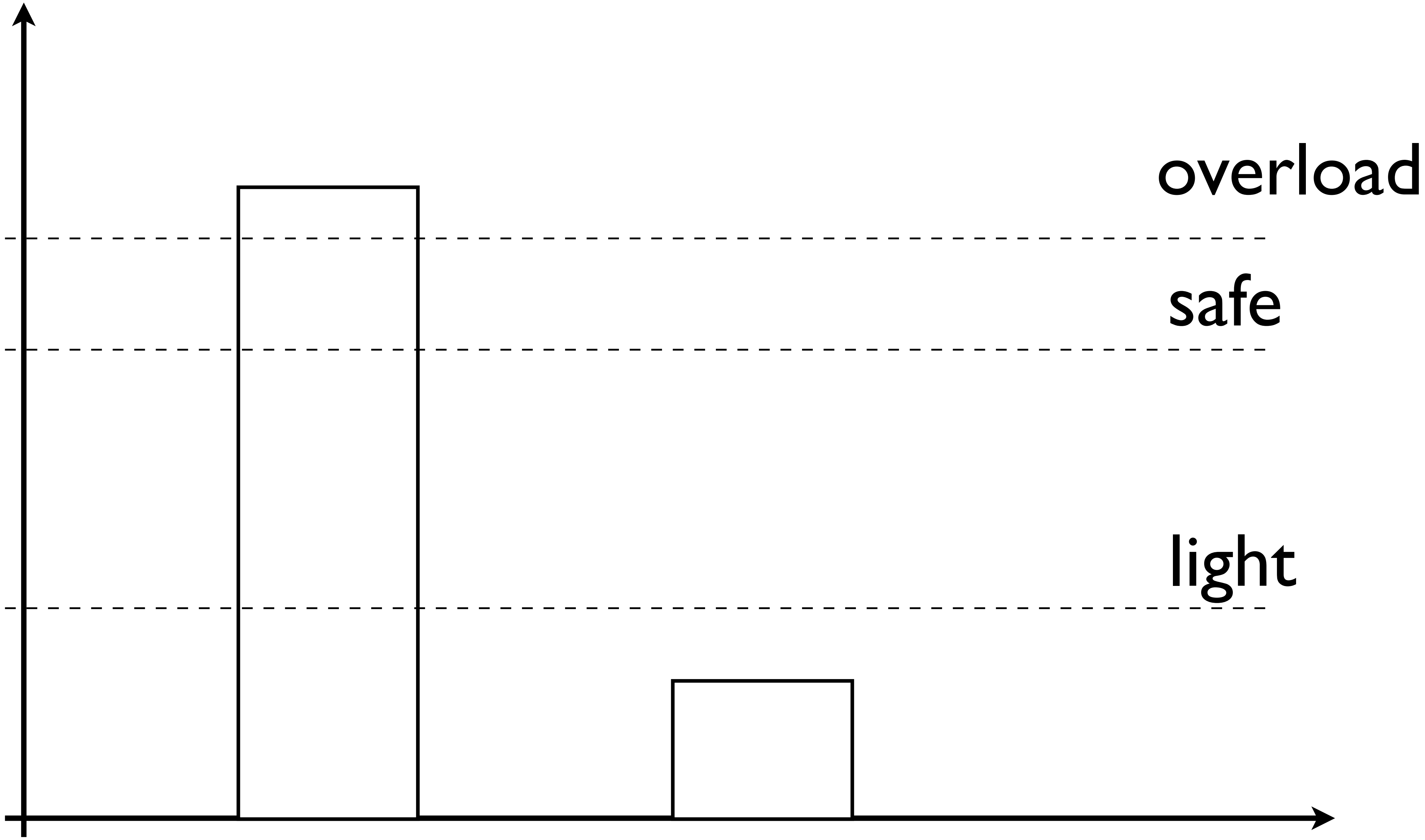
Disadvantages of centralized approach

Expensive to compute a good solution

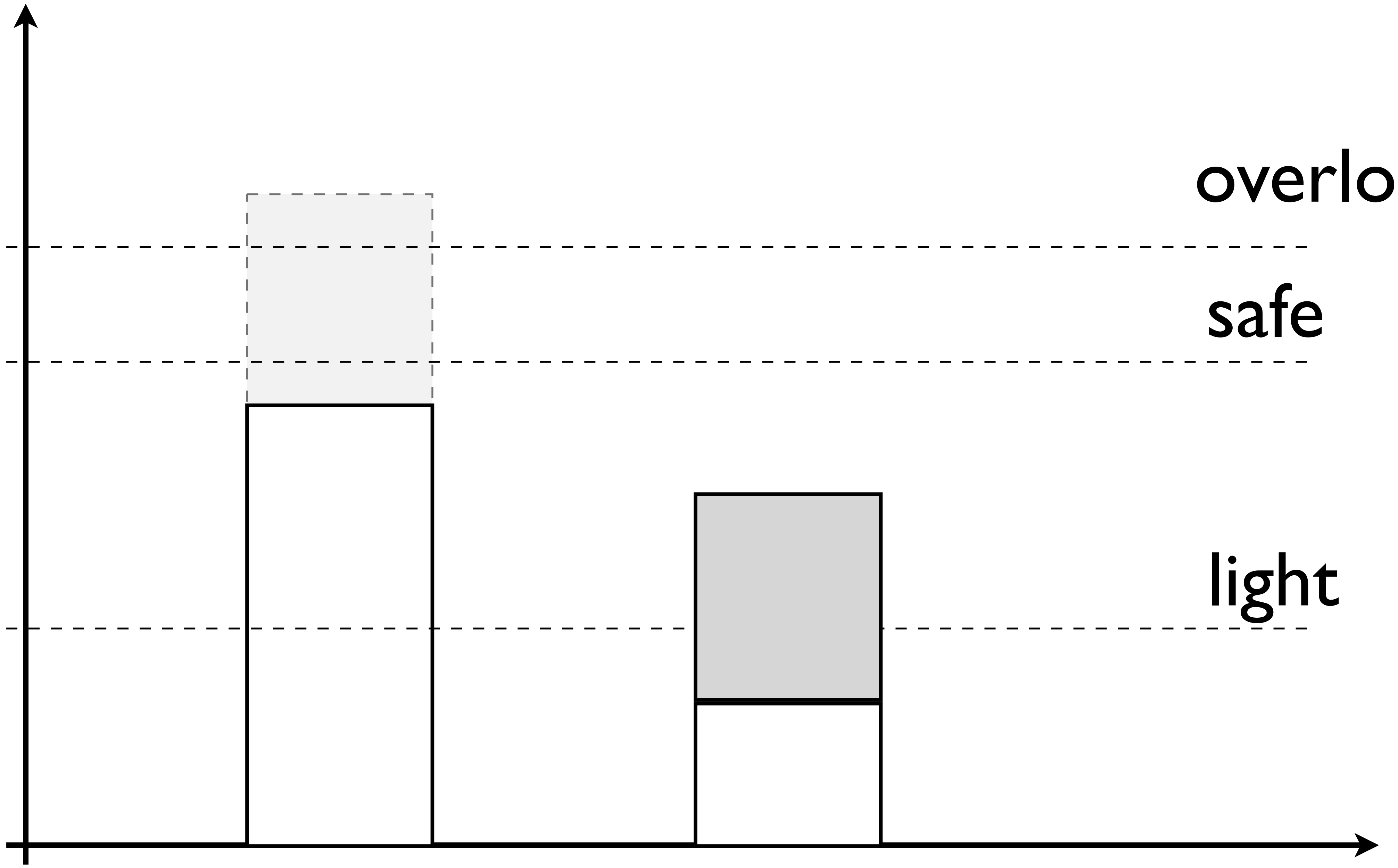
How frequent should the world be repartitioned?

Localized World Partitioning

Idea: Only repartition locally if a server is overloaded. Look for a lightly loaded server to shed some load to.


overload
safe
light

Overloaded server sheds load until it's load is below a safe level. The new load of a previously lightly loaded should remain below safe level.



$$l + o - s < s$$

$$l < 2s - o$$

Who to shed to?

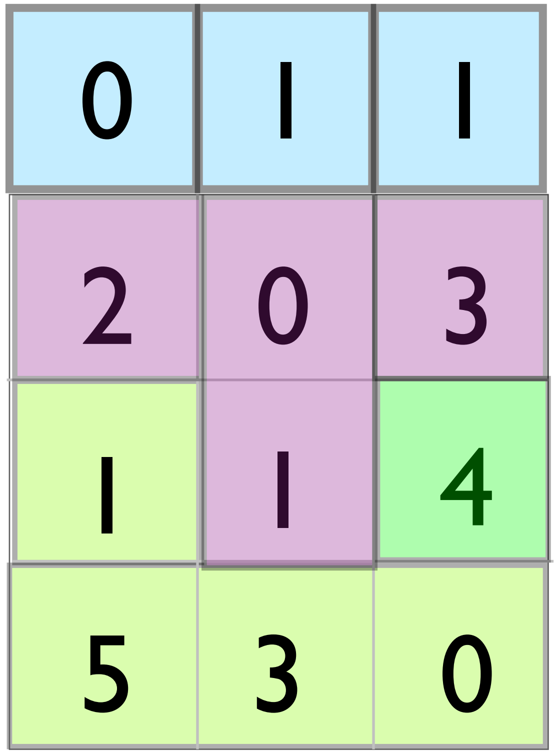
Minimize overhead: try to shed to as few neighboring server as possible

Migrate cells to neighboring servers that are not overloaded

If still not enough, look at the list of lightly loaded servers maintained. Pick the least load and shed to it.

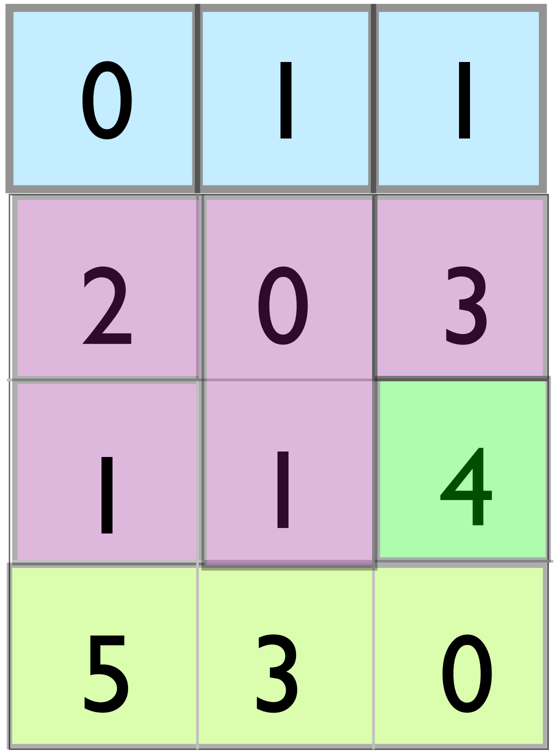
Shed remaining load to remote servers

Overload = 10, Safe = 7, Light = 4



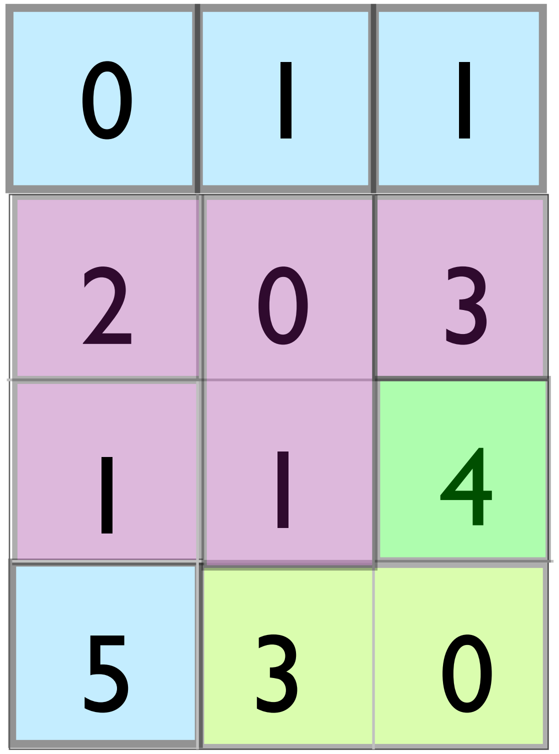
0	1	1
2	0	3
1	1	4
5	3	0

Shed load to neighbor first



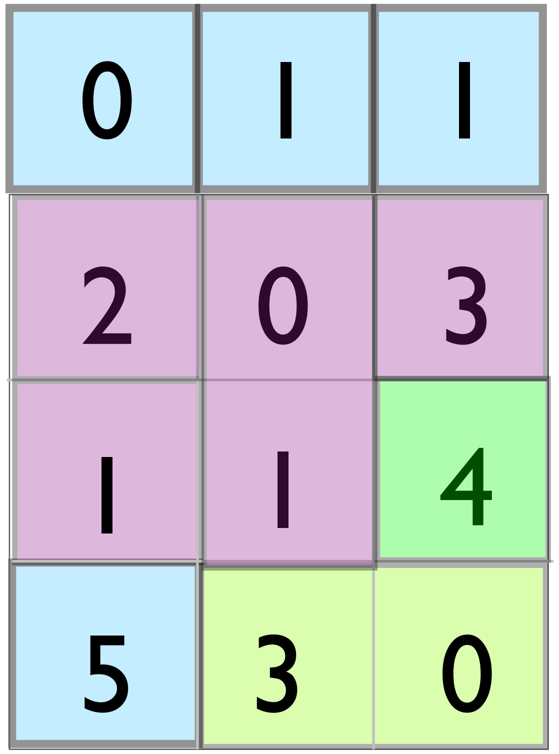
0	1	1
2	0	3
1	1	4
5	3	0

Shed load to non-neighboring server



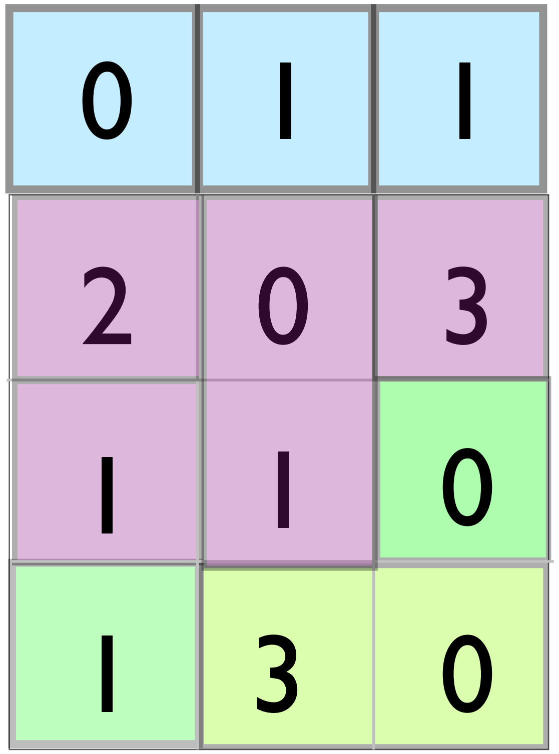
0	1	1
2	0	3
1	1	4
5	3	0

But this could lead to high communication overhead



0	1	1
2	0	3
1	1	4
5	3	0

When communication overhead is too high, isolated cells can be migrated to a nearby server, “merging” with neighboring regions to form contiguous regions, as long as the load remains below the safe threshold.

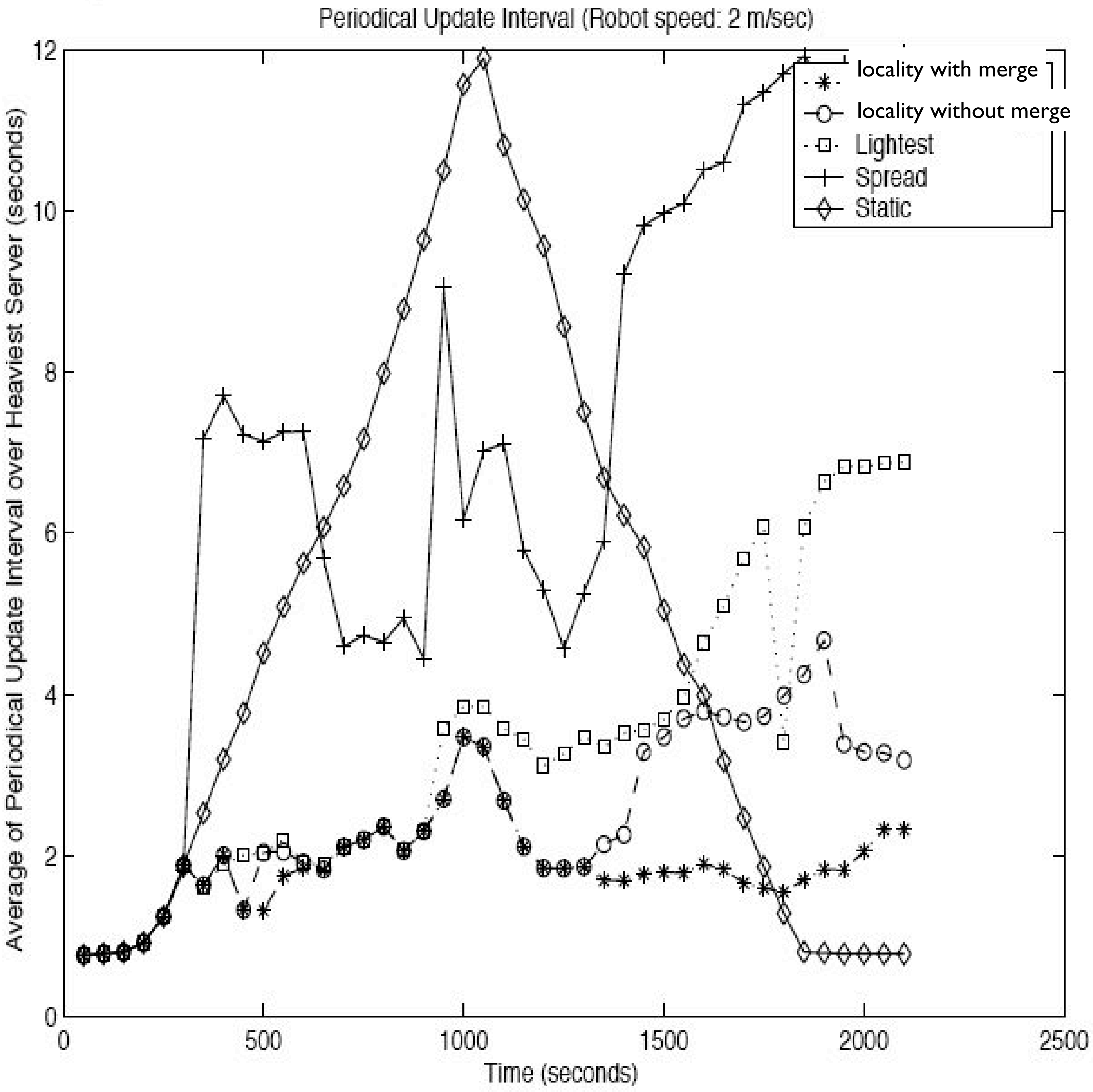


0	1	1
2	0	3
1	1	0
1	3	0

Evaluation

Compare 5 schemes using a simulated games

1. spread - “balanced deployment”
2. always shed to lightest server
3. static
4. locality with merging
5. locality without merging


Periodical Update Interval (Robot speed: 2 m/sec)
Average of Periodical Update Interval over Heaviest Server (seconds)
Time (seconds)
0
500
1000
1500
2000
2500
0
2
4
6
8
10
12
locality with merge
locality without merge
Lightest
Spread
Static

You Are Here

- CS4344
  - Client/Server Architecture
  - Peer-to-Peer Architecture
  - Hybrid
    - Mirrored servers
    - Zoned servers