Proxies for Networked Games

Proxy: trusted host providing specialized services for games

Proxy: typically close to the players

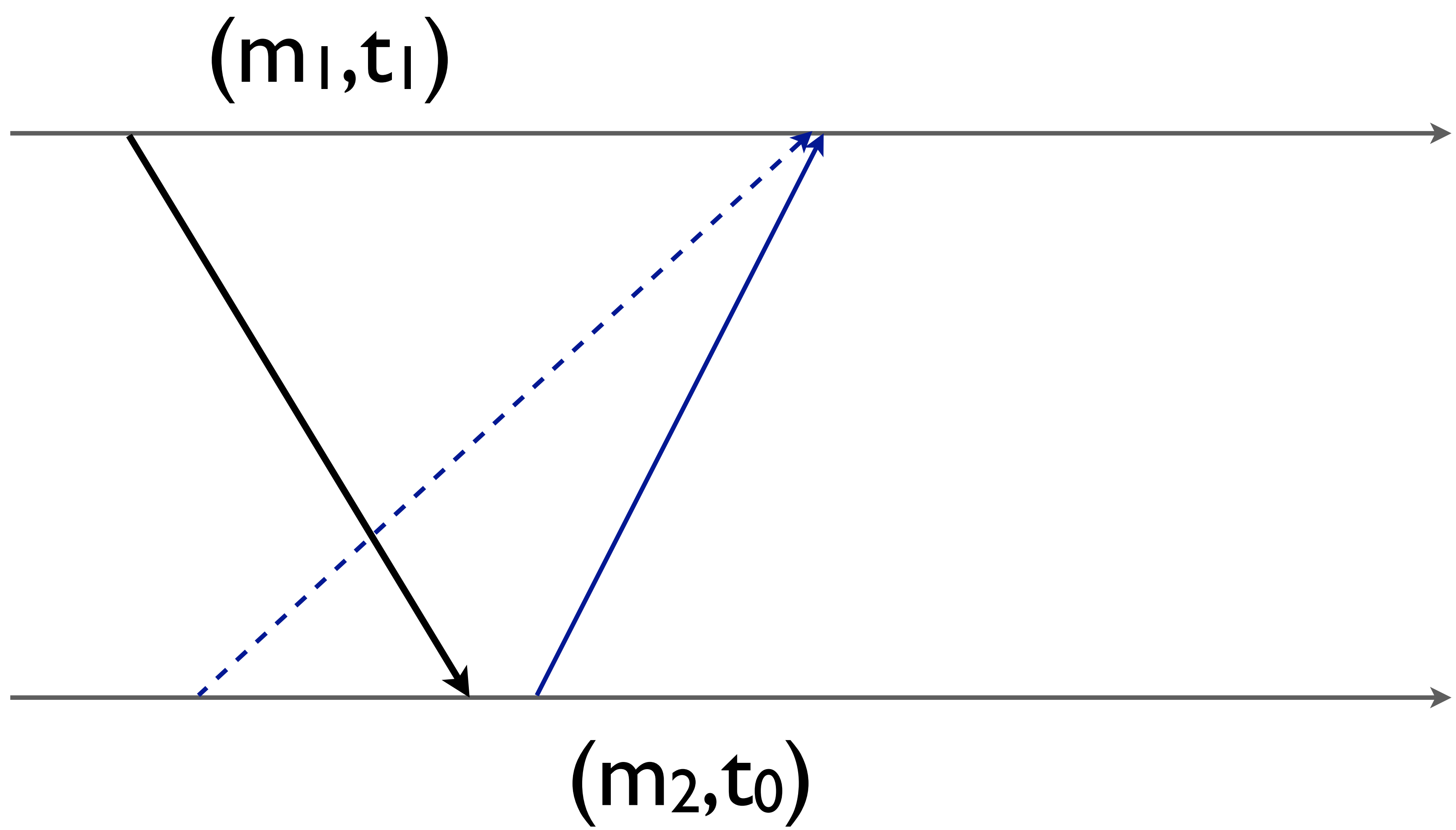
Examples of Proxy Services:

Time-stamping
Message Ordering
Interest Management
App-Level Multicast

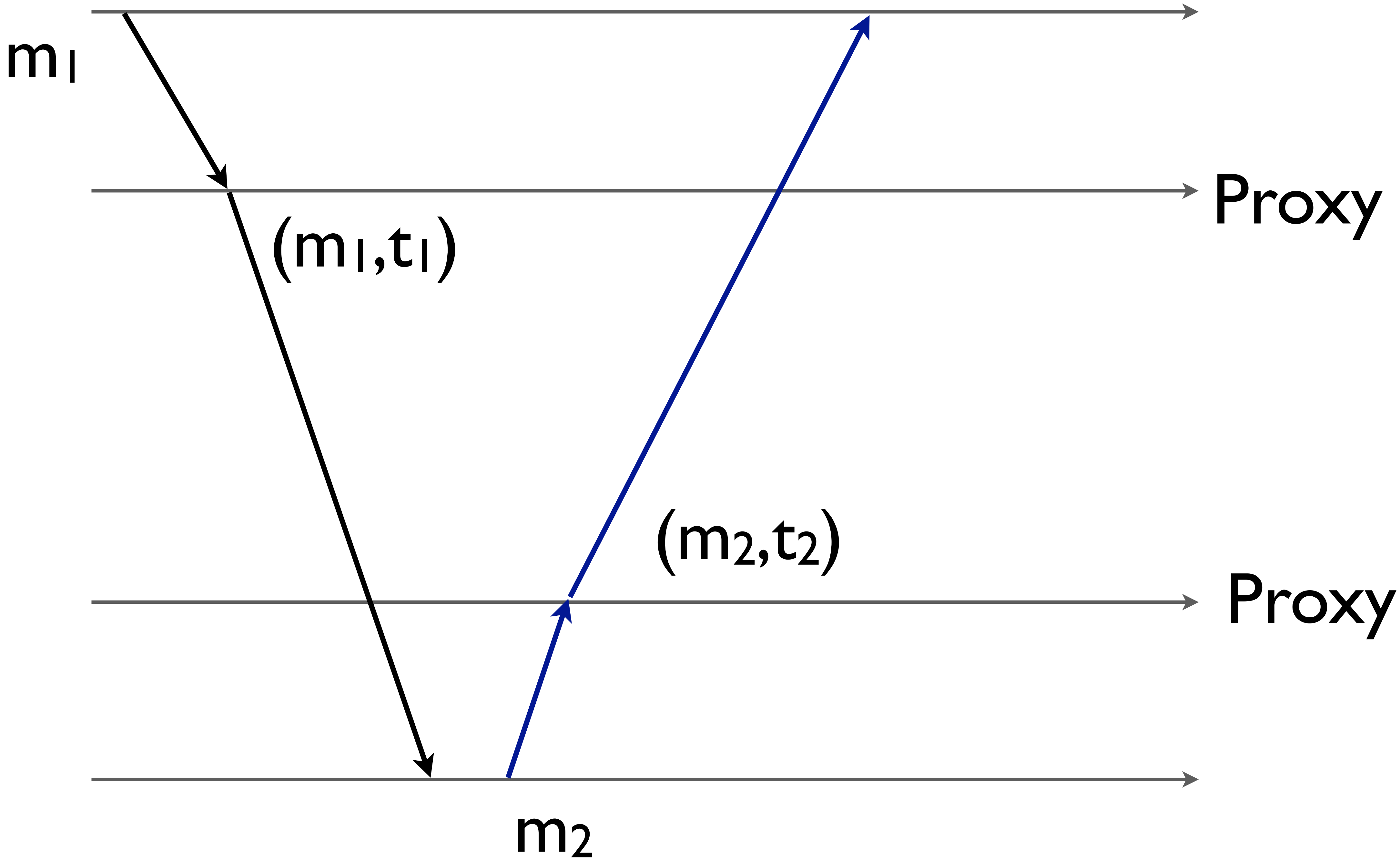
Example 1:
Time-stamping Services

Timestamp Cheat in P2P Games

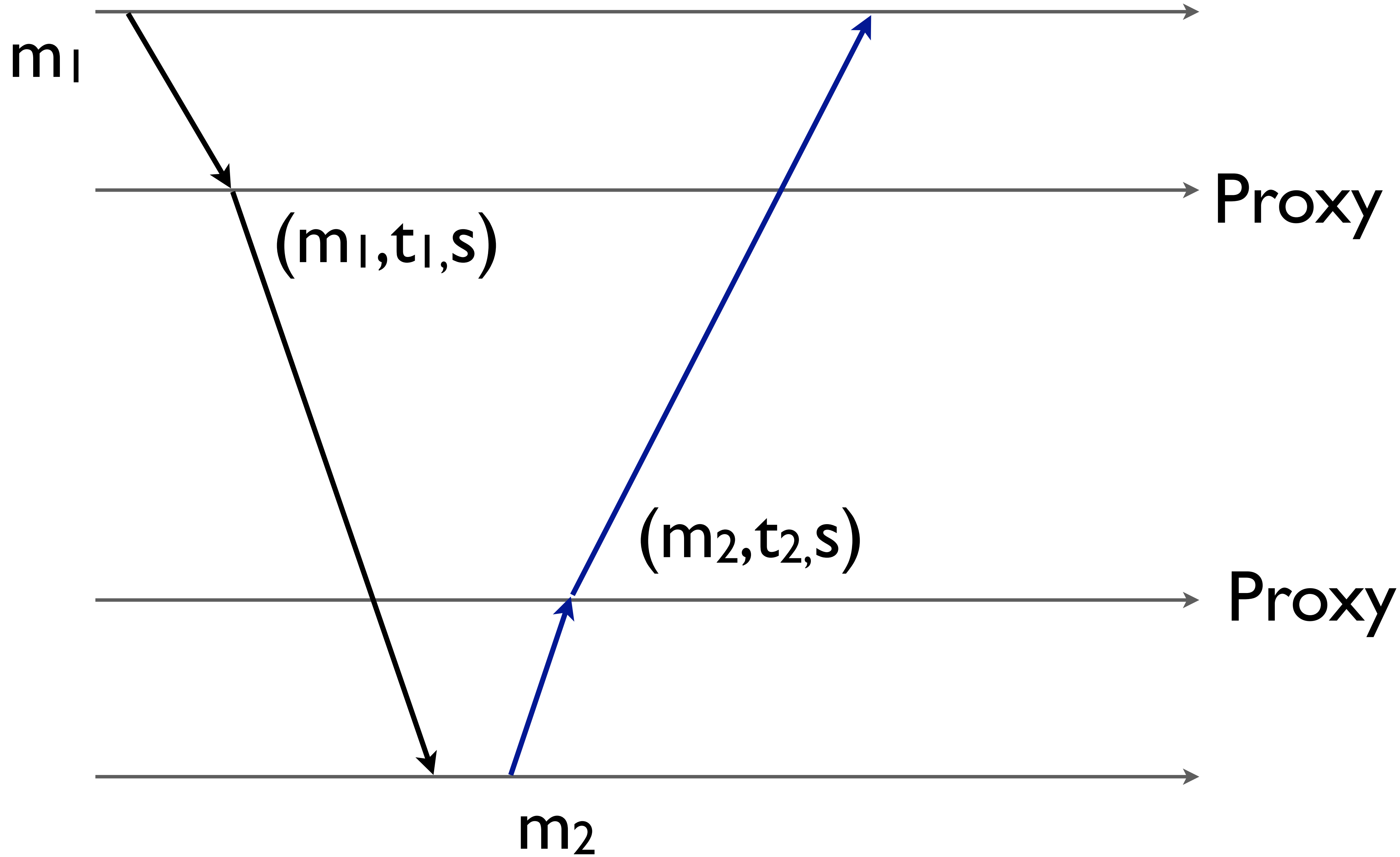
Player generates fake, earlier timestamp to gain advantage after knowing the opponent's move.



But such cheat is impossible if a trusted proxy timestamp the messages instead of the players.



The proxy can digitally signed the message to proof that he is the one that added the timestamp.

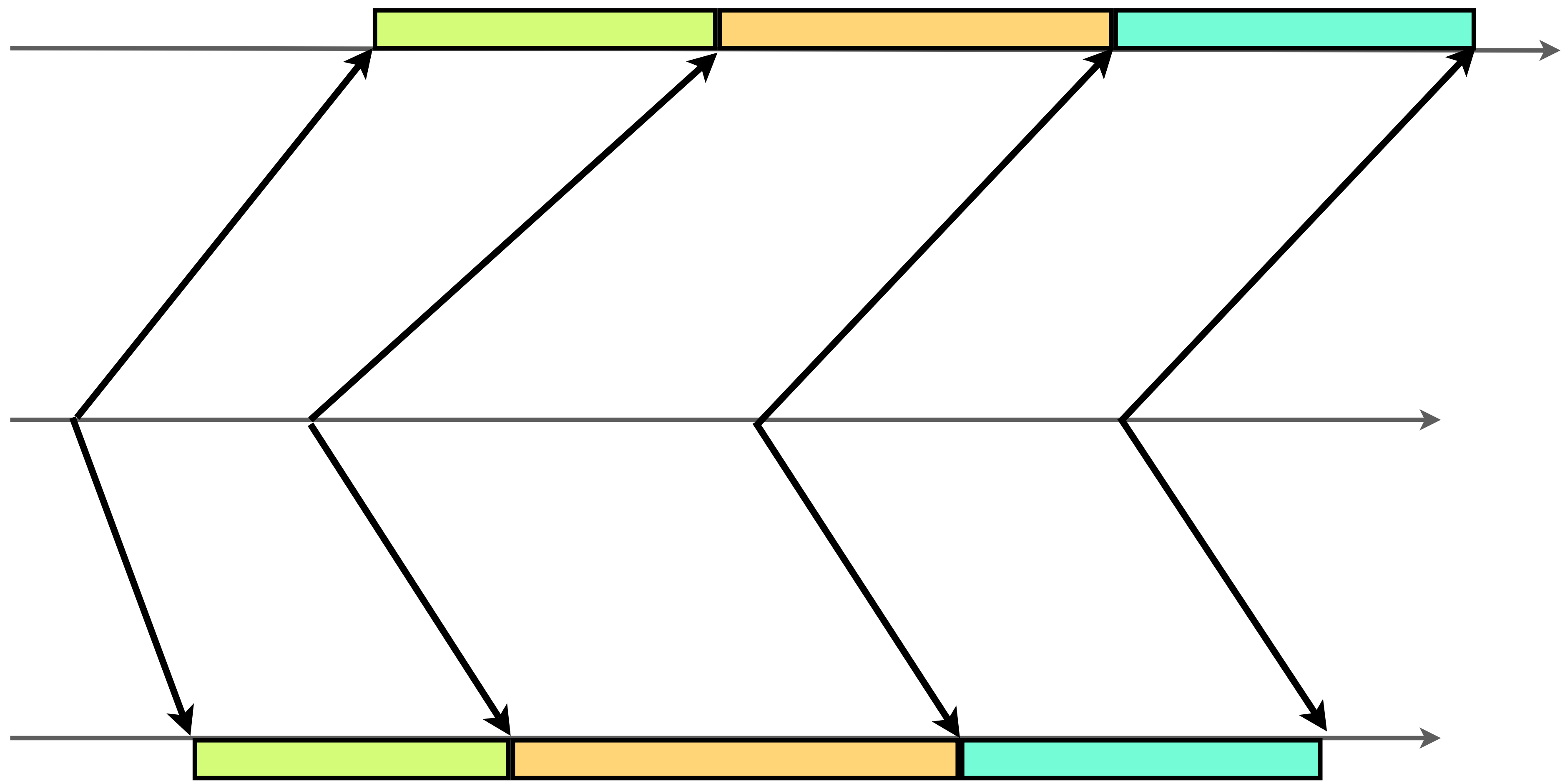


Timestamp Servers is generic and is game independent

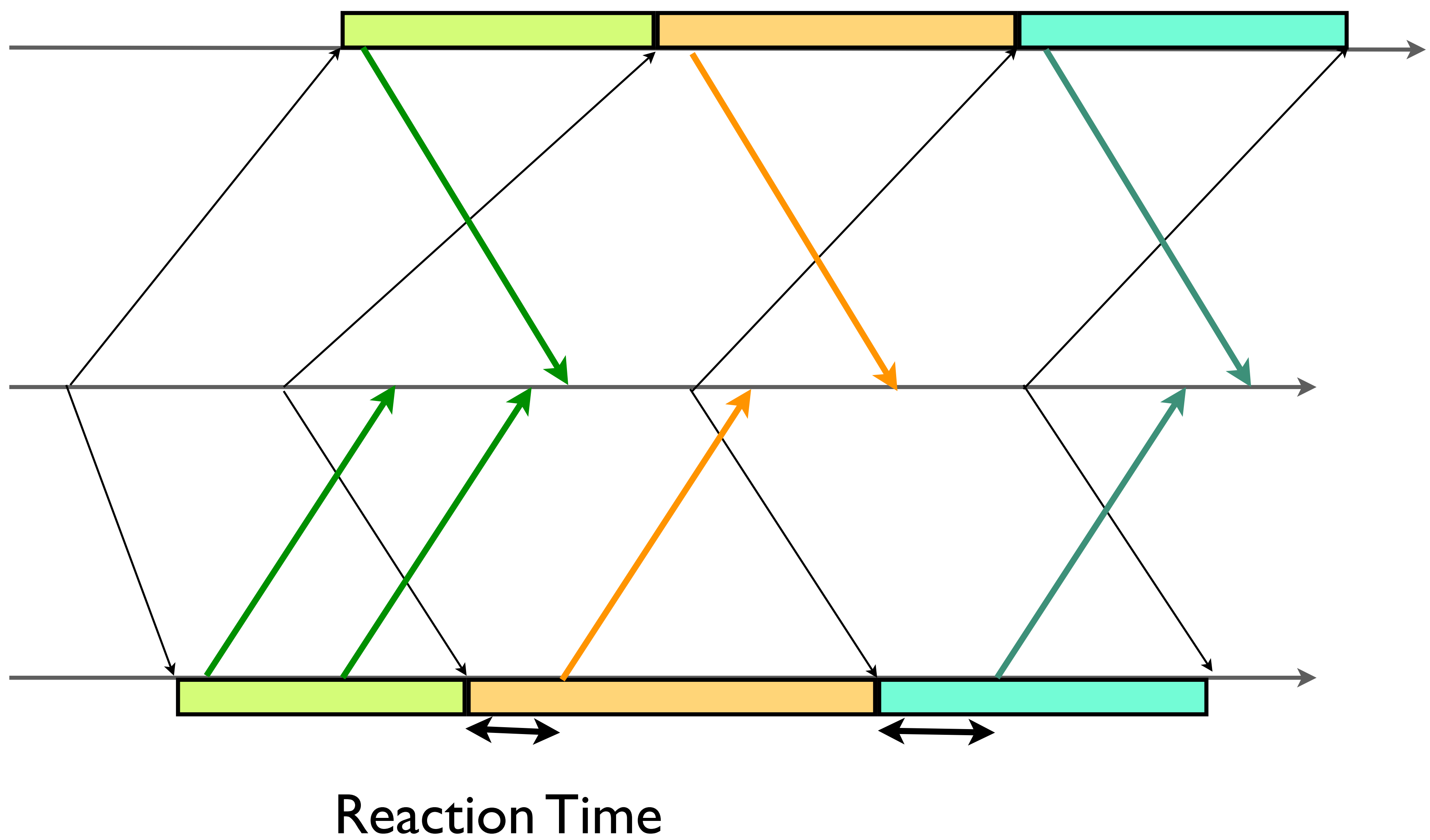
Example 2:
Message Ordering

Client/Server Architecture

Game is divided into rounds based on updates from the server.

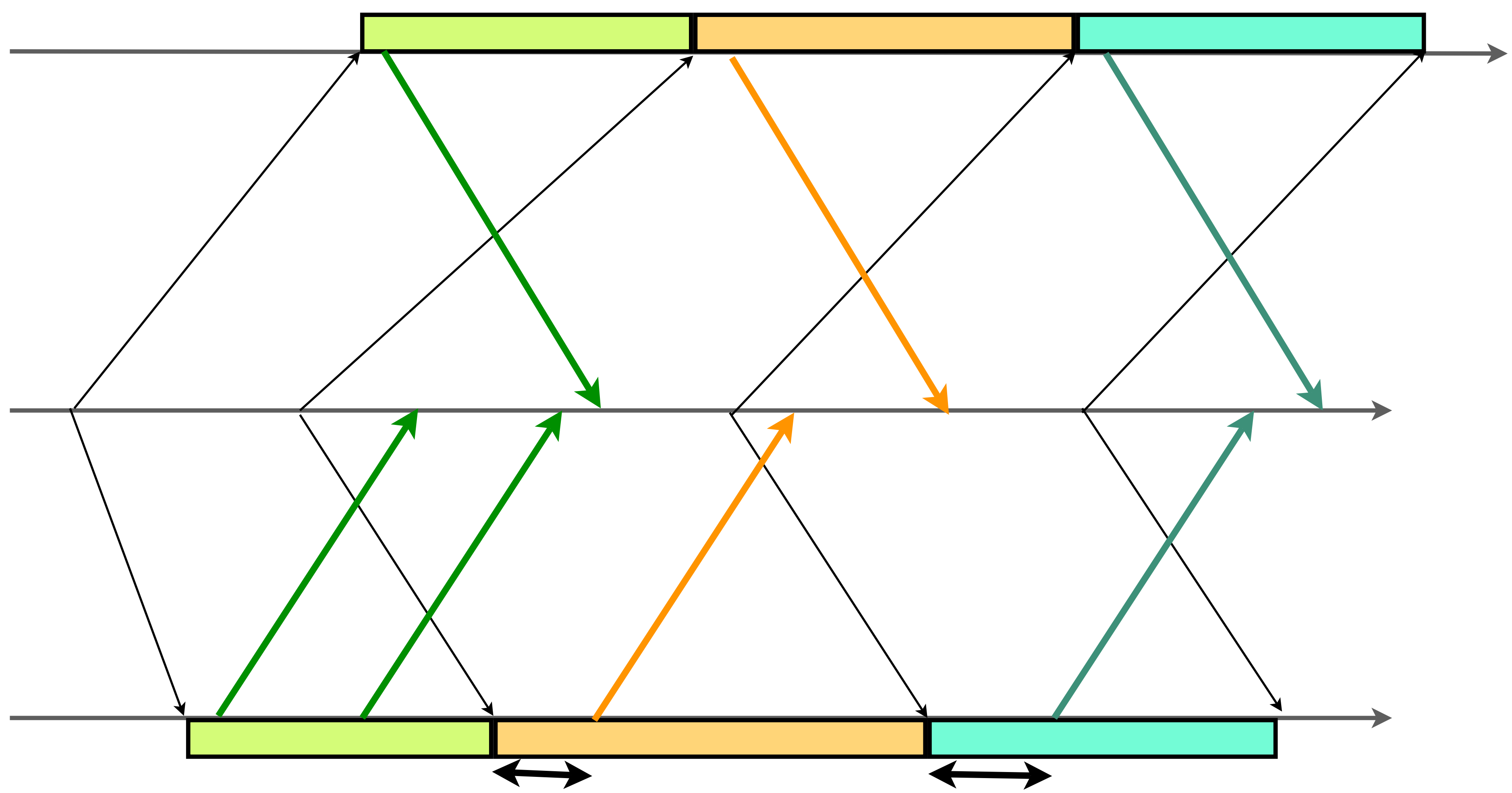


Messages (action) from players are grouped into rounds. Reaction time is the time between receiving message from server and sending an action.



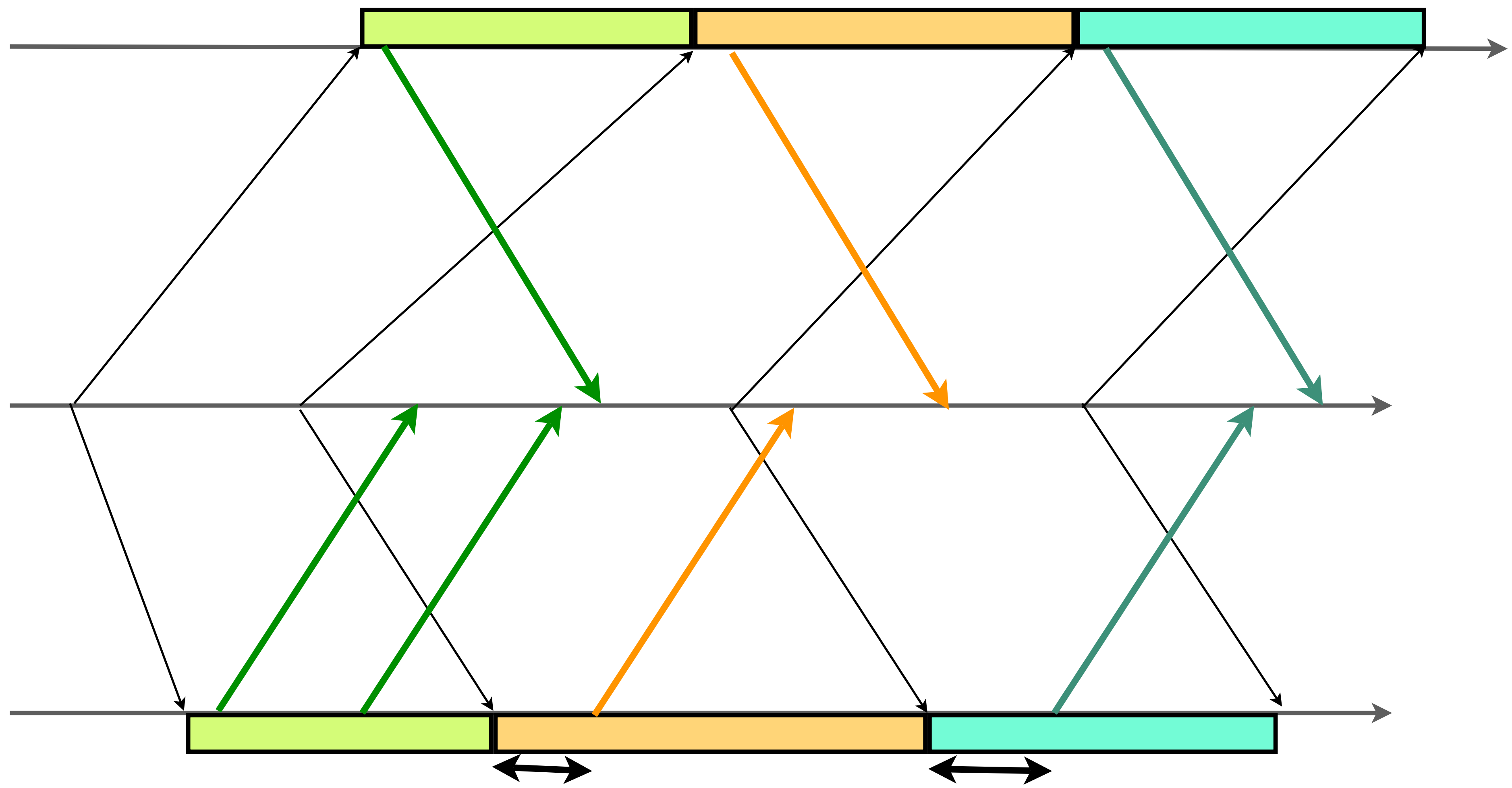
We want to deliver messages to the server in the following order:

1. for a player, messages are delivered in the order they are generated.



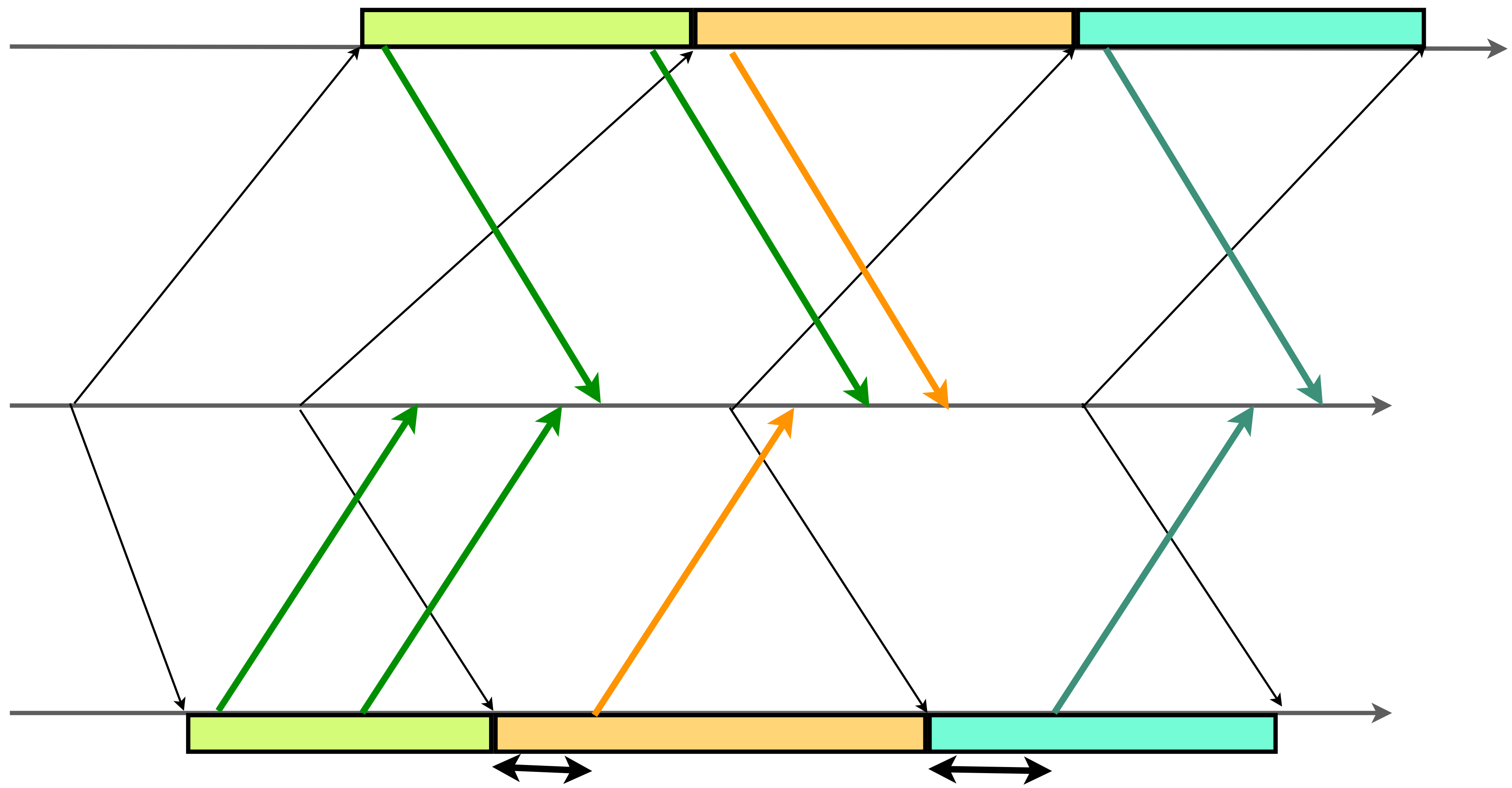
2. messages from the same round from different players are delivered in increasing order of "reaction time".

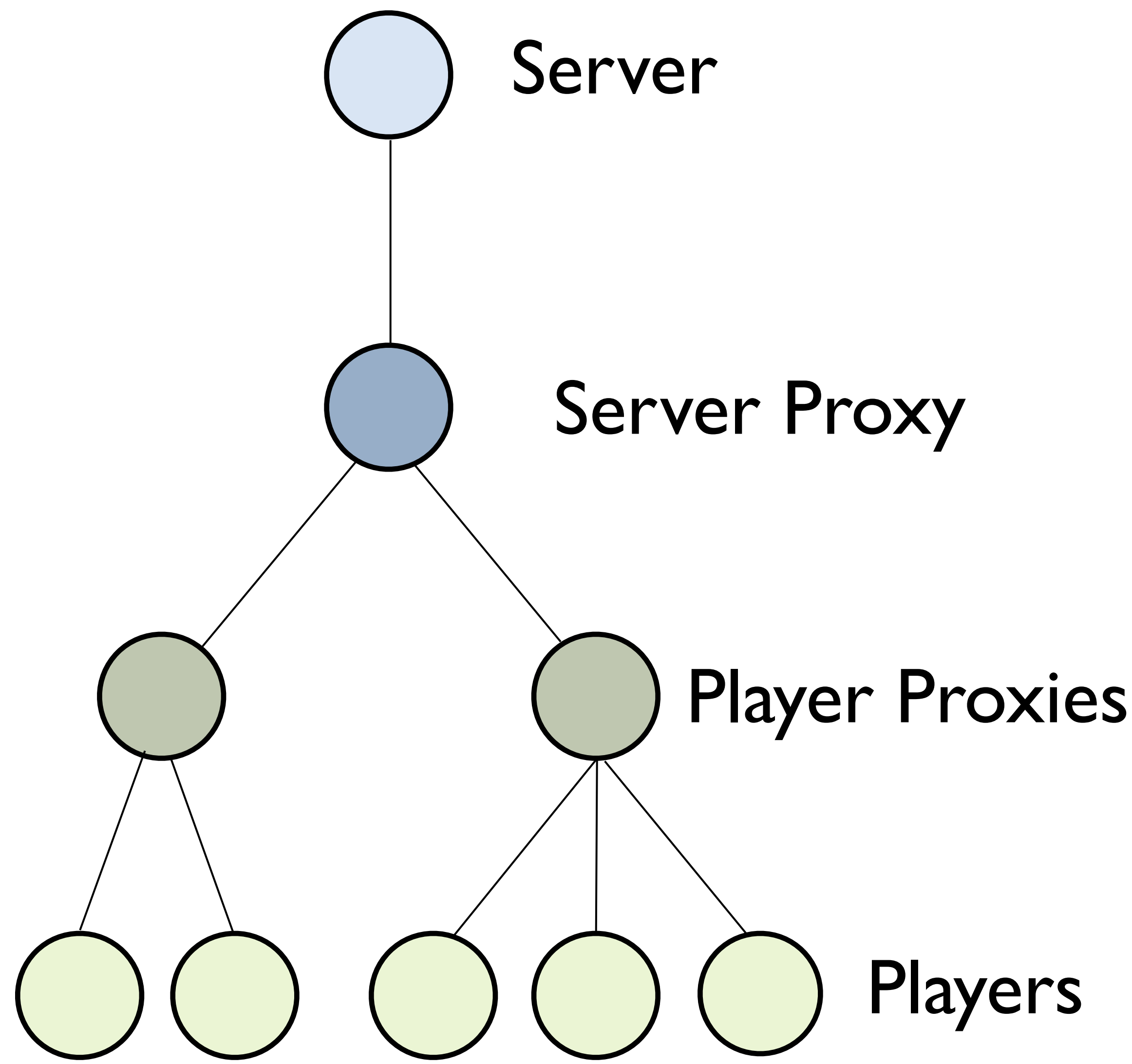
Messages are not delivered according to reaction time.



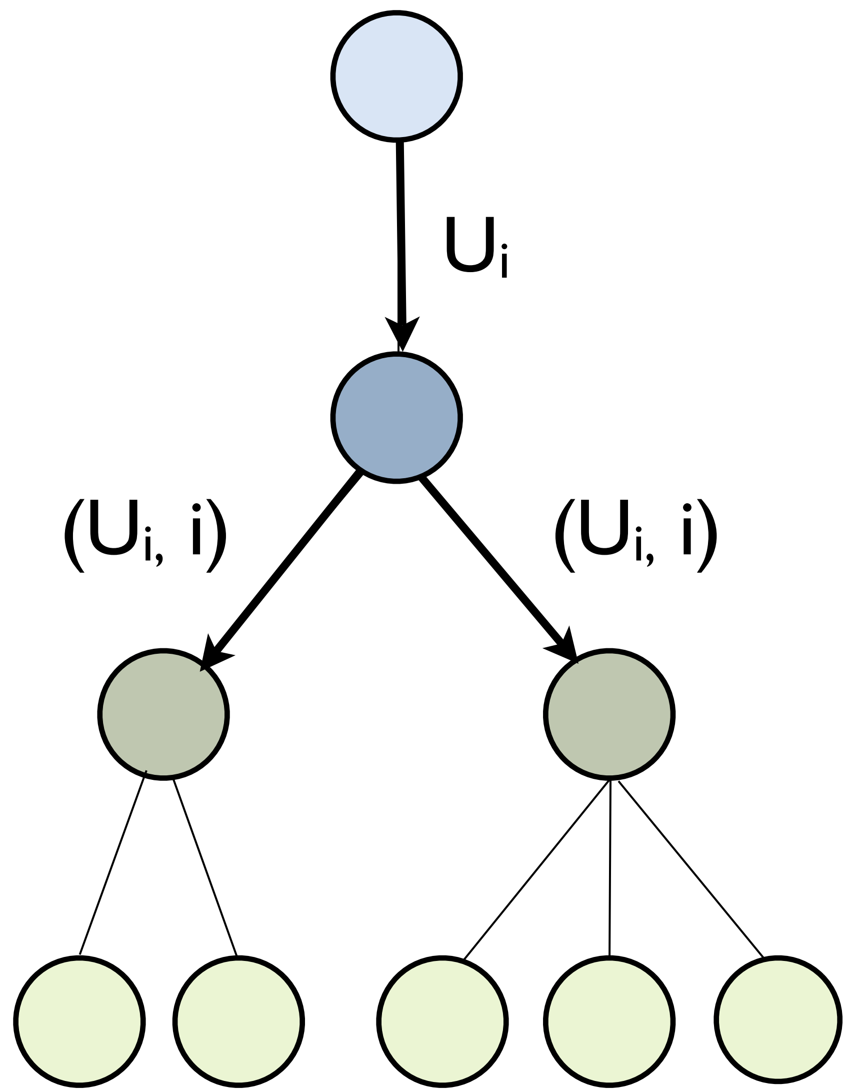
3. messages in a round is delivered to the server before messages from the subsequence rounds.

The second green message from the top player violates this condition.

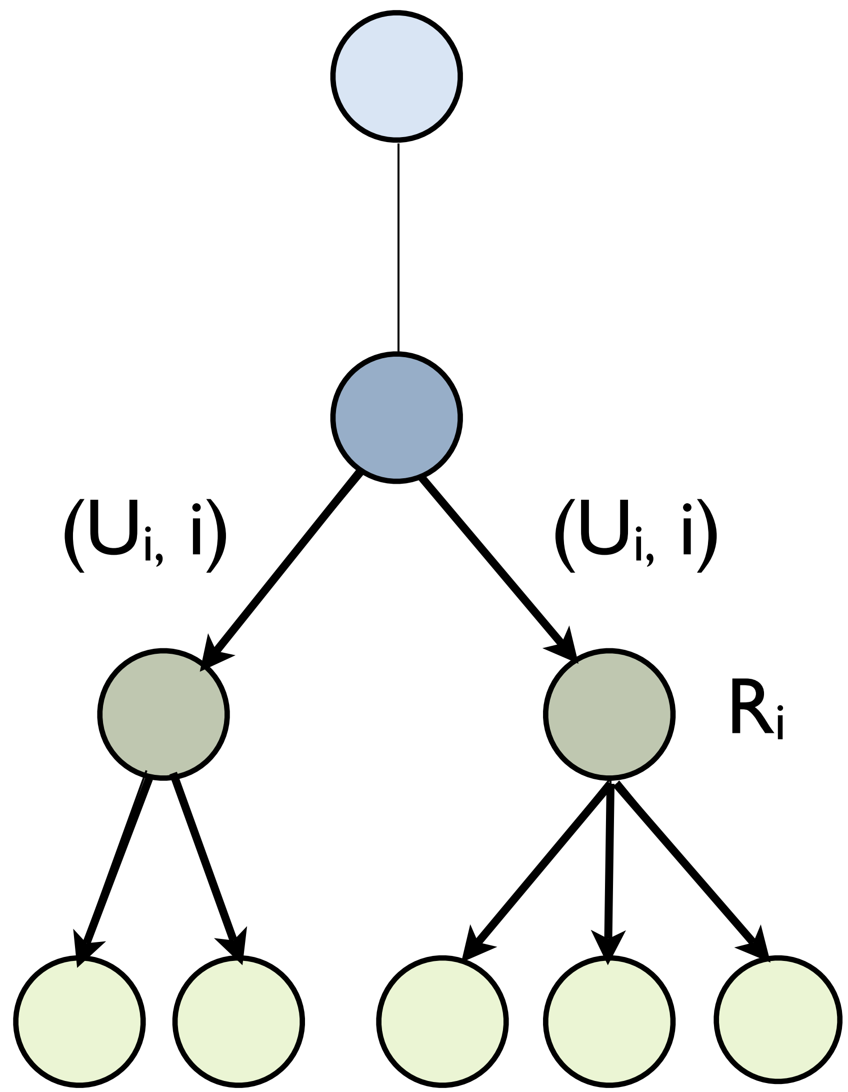



Server
Server Proxy
Player Proxies
Players

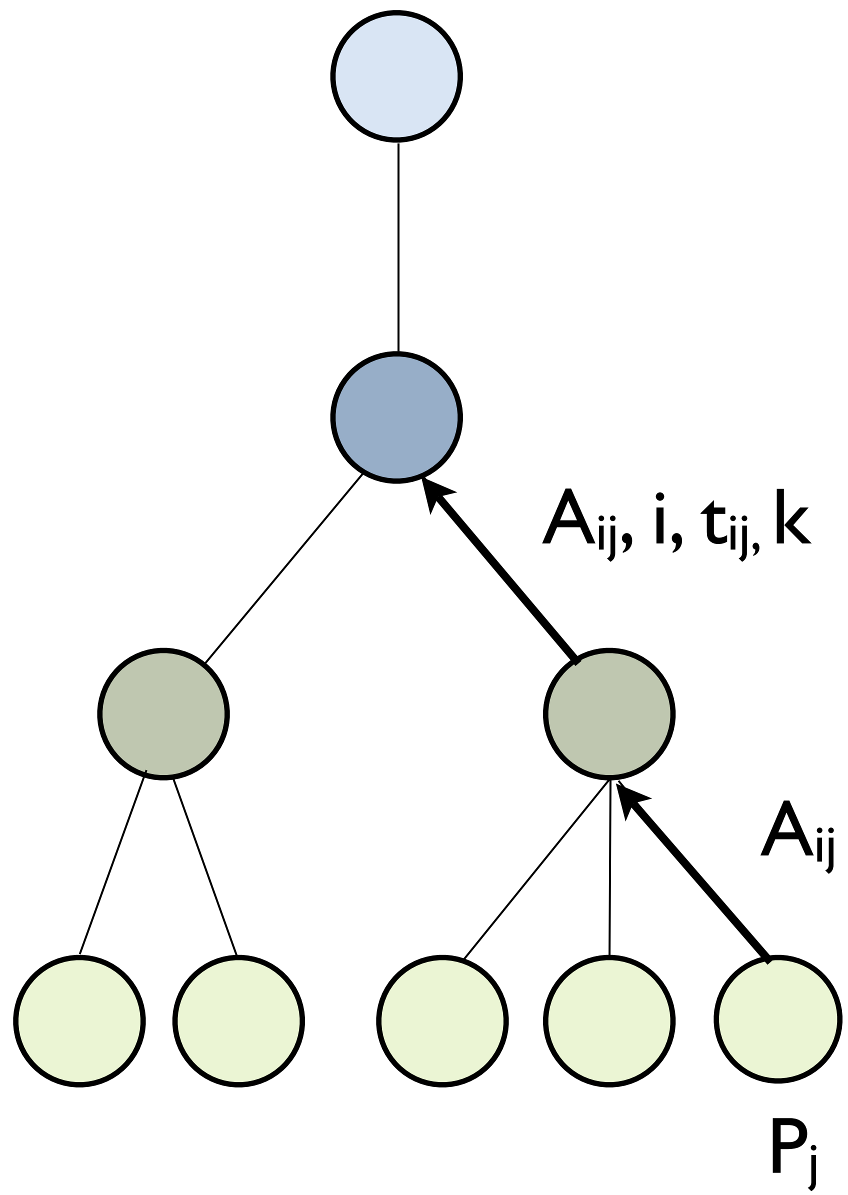
The server proxy tags each message ("updates") from the server with the round number.



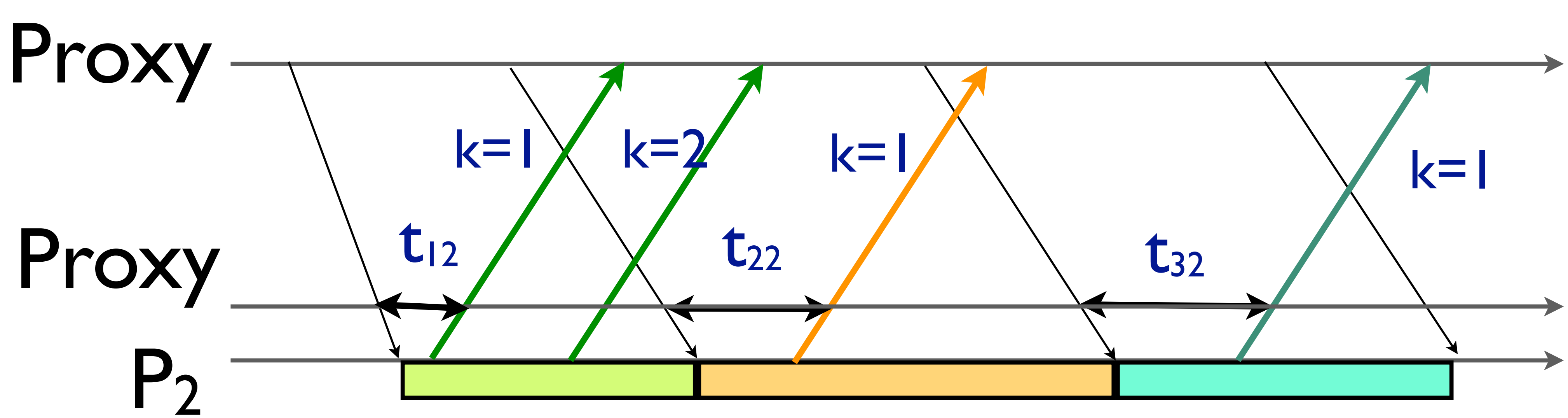
The player proxy remembers R_i, the time message U_i is received, and forwards it to players.



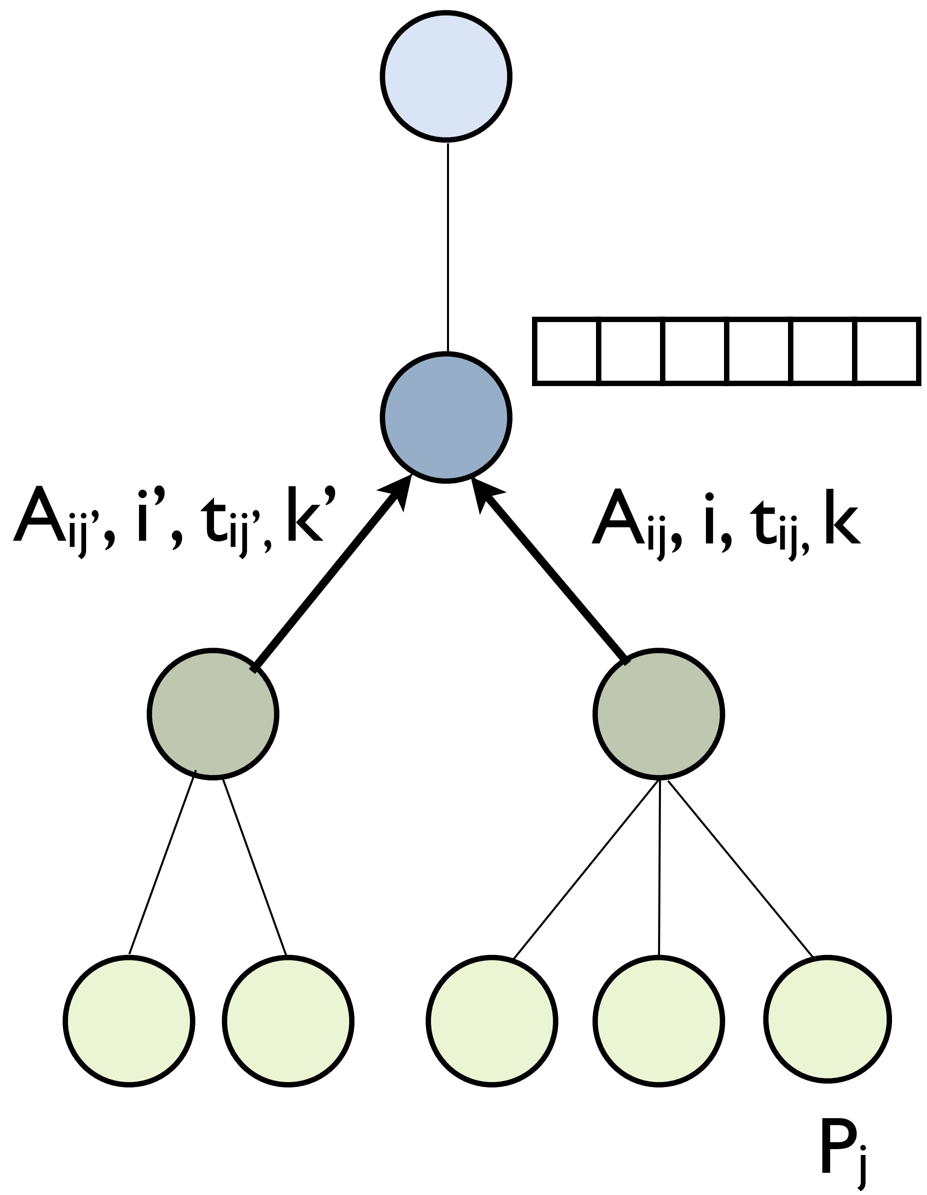
Action messages from the players are tagged by the player proxies with (i) round number *i* (ii) reaction time t_{ij} and (iii) order of action messages from that player.



Example of tags inserted by the proxy.



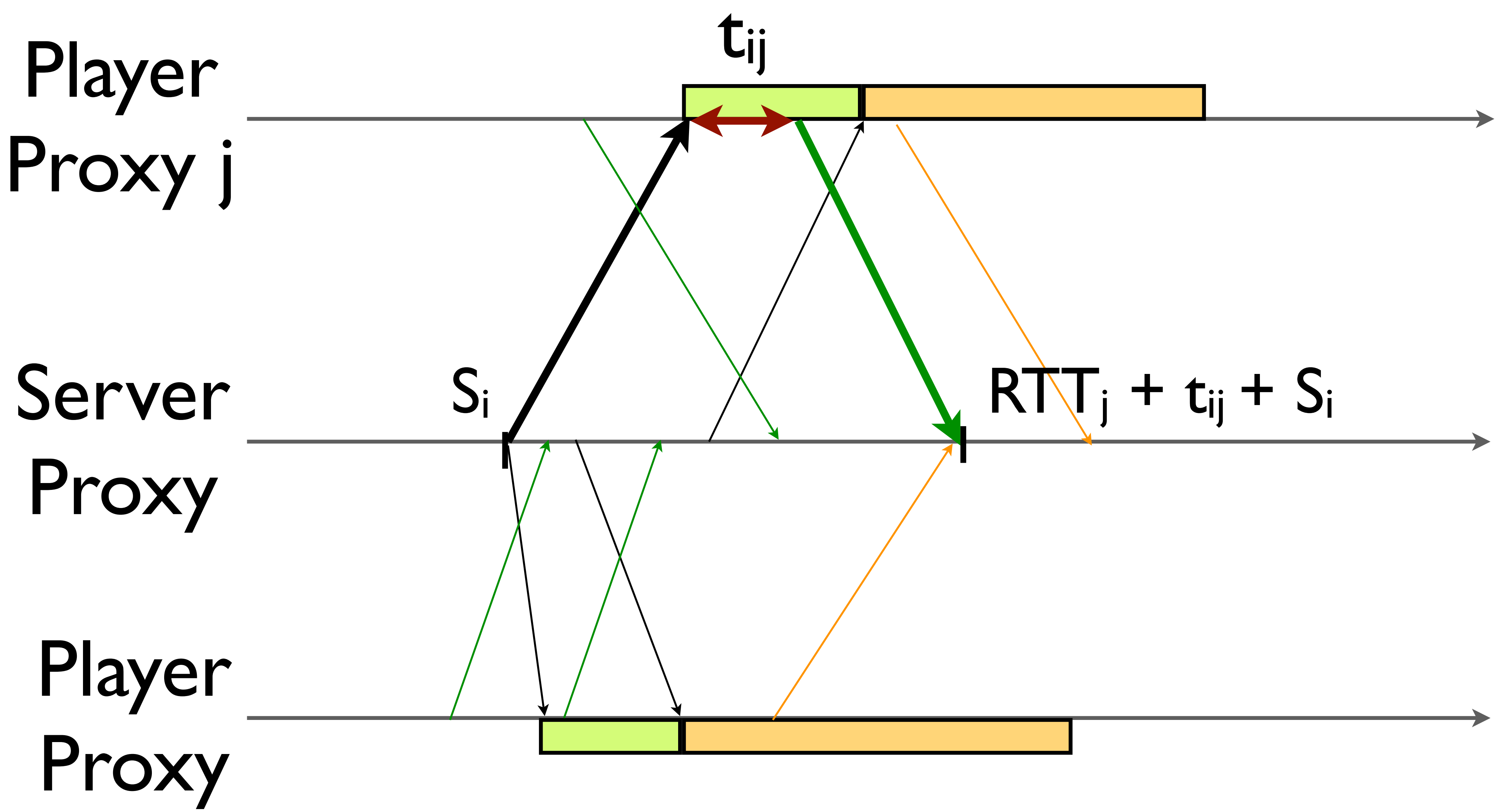
The server proxy now determines two things: (i) in what order to deliver the messages, and (ii) when to delivery the messages



Delivery order: sort the queue according to round number, then by reaction time. Deliver messages in the order inside the queue.

Delivery time: can deliver the head of the queue as long as no other messages from the same round is still in transit.

Receive Time = $RTT_j + t_{ij} + S_i$



Message from k within the same round *i* should be sent first if $t_{ik} < t_{ij}$

$$RTT_j + t_{ij} + S_i$$

If message from k is still in transit when message from j is received, then

$$RTT_j + t_{ij} + S_i < RTT_k + t_{ik} + S_i$$

$$< RTT_k + t_{ij} + S_i$$

If message from k is still in transit when message from j is received, then

$$RTT_j + t_{ij} + S_i < RTT_k + t_{ik} + S_i$$

$$< RTT_k + t_{ij} + S_i$$

If message from j is delivered at this time, then message from k should have arrived

It is safe to set the delivery time of a message from *j* to be

$$\max \{RTT_k\} + t_{ij} + S_i$$

k in the set of hosts whose messages have not arrived

Example: Consider a given round *i*. $S_i = 0$
$t_{i1} = 1 \quad t_{i2} = 2 \quad t_{i3} = 3$
$RTT_1 = 10 \quad RTT_2 = 20 \quad RTT_3 = 10$

At time 11, message from Player 1 is received.
Delivery time for message 1 = 21

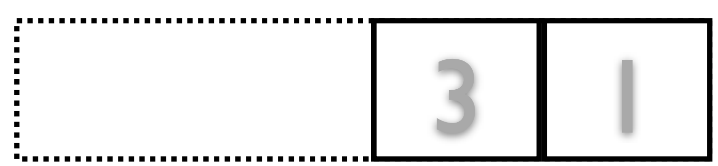
Example: Consider a given round *i*. $S_i = 0$

$t_{i1} = 1$ $t_{i2} = 2$ $t_{i3} = 3$

$RTT_1 = 10$ $RTT_2 = 20$ $RTT_3 = 10$

At time 13, message from Player 3 is received.

Delivery time for message 3 = 23



Example: Consider a given round *i*. $S_i = 0$

$t_{i1} = 1$ $t_{i2} = 2$ $t_{i3} = 3$

$RTT_1 = 10$ $RTT_2 = 20$ $RTT_3 = 10$

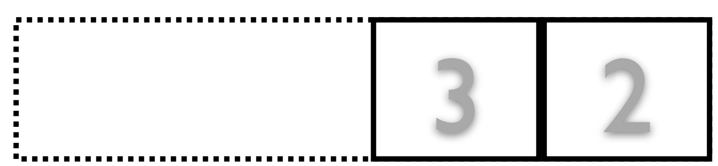
At time 21, message from Player 1 is delivered

Example: Consider a given round *i*. $S_i = 0$

$t_{i1} = 1$ $t_{i2} = 2$ $t_{i3} = 3$

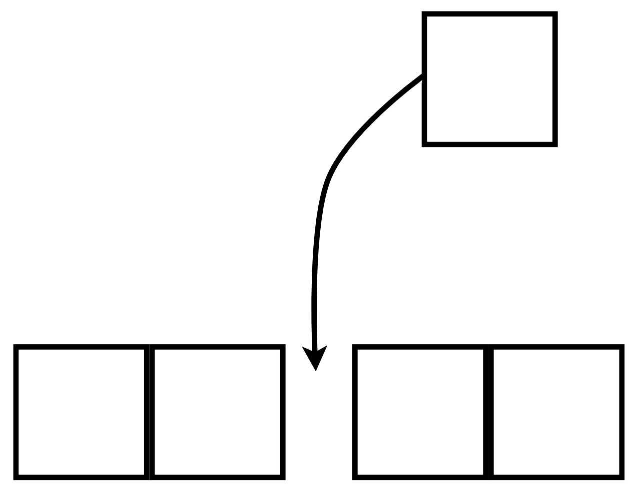
$RTT_1 = 10$ $RTT_2 = 20$ $RTT_3 = 10$

At time 22, message from Player 2 is received. Delivery time for message 2 = 12. The message is sent right away. At time 23, message from Player 3 is delivered.



(We can actually delivery message 3 earlier as an optimization)

When a message is inserted into the queue, the delivery time of messages ahead in the queue may shorten, while the delivery time messages behind remains unchanged.



$$\max \{RTT_k\} + t_{ij} + S_i \;>\; \max \{RTT_k\} + t_{ij} + S_i$$

k in set of hosts whose messages have not arrived

k in set of hosts whose messages have not arrived now

Consider the case where messages from previous round is still in transit, the delivery time should be set as

$$\max \{T_{i-1}, \max \{RTT_k\} + t_{ij} + S_i\}$$

k = set of hosts whose messages have not arrived

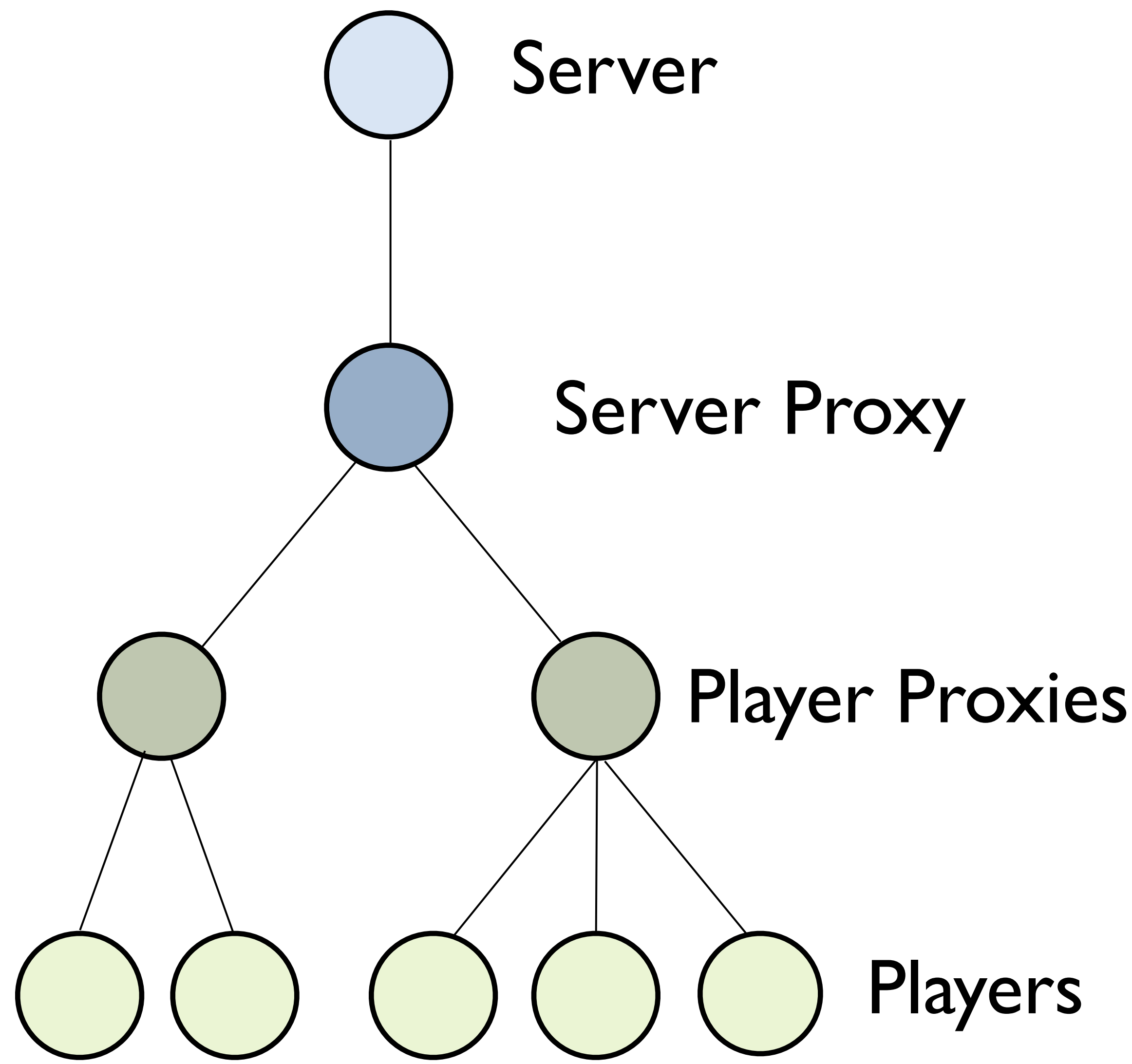
where T_{i-1} is the time where maximum delivery time from previous round

What if a message is late?

Then it is delivered immediately to the server

Advantages: No need to synchronize clock or estimate one way delay.

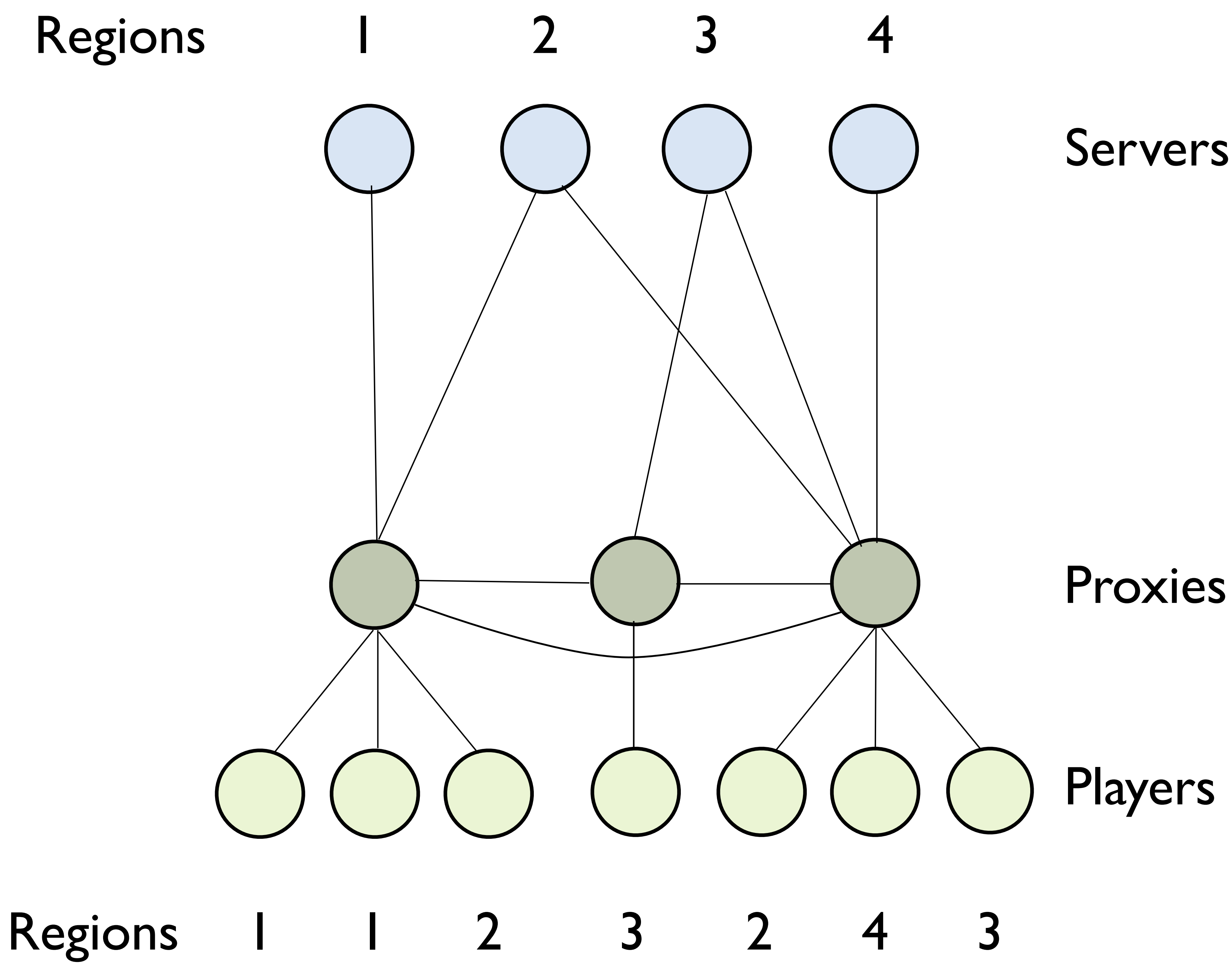
Estimating RTT is easier.

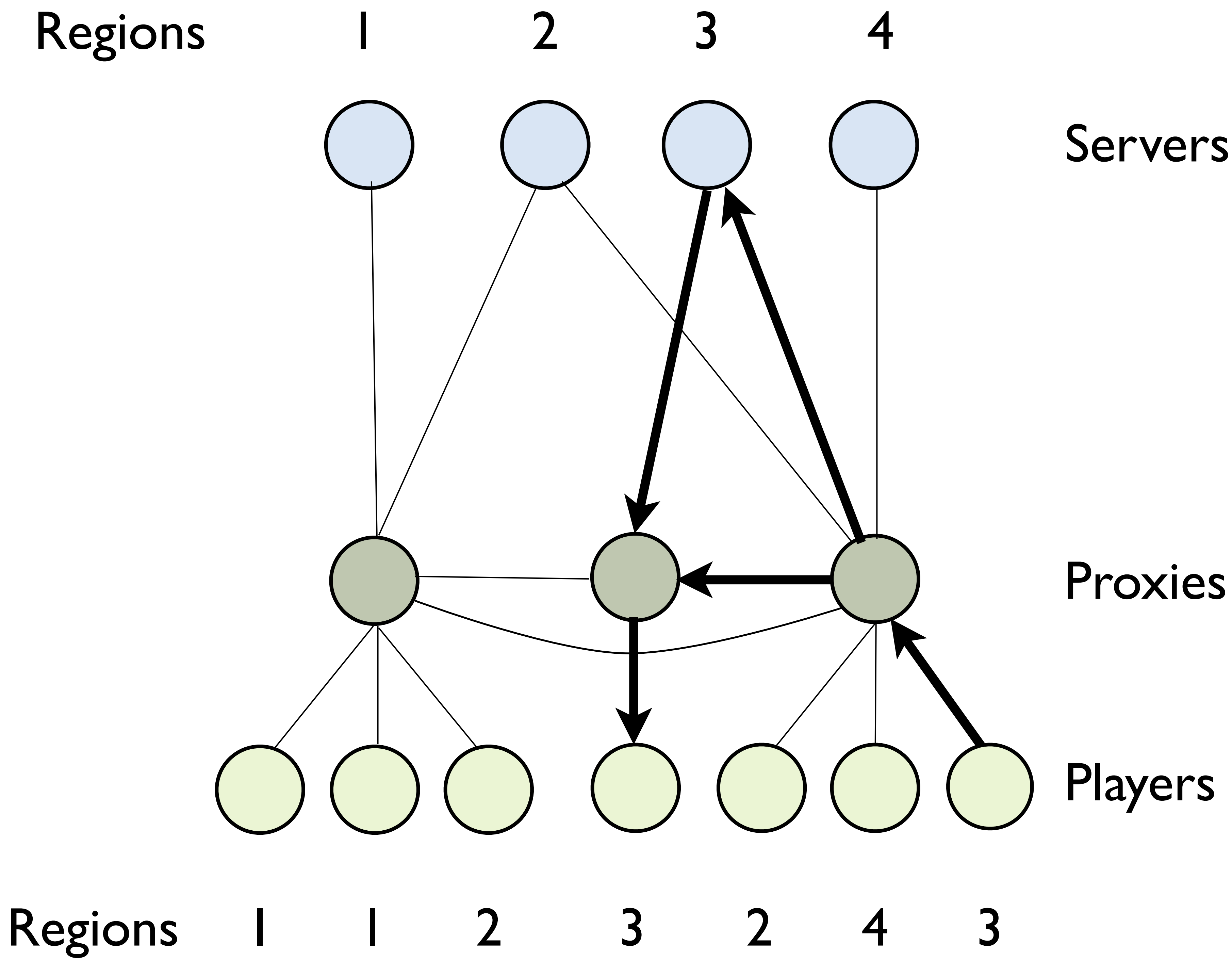

Server
Server Proxy
Player Proxies
Players

Examples of Proxy Services:

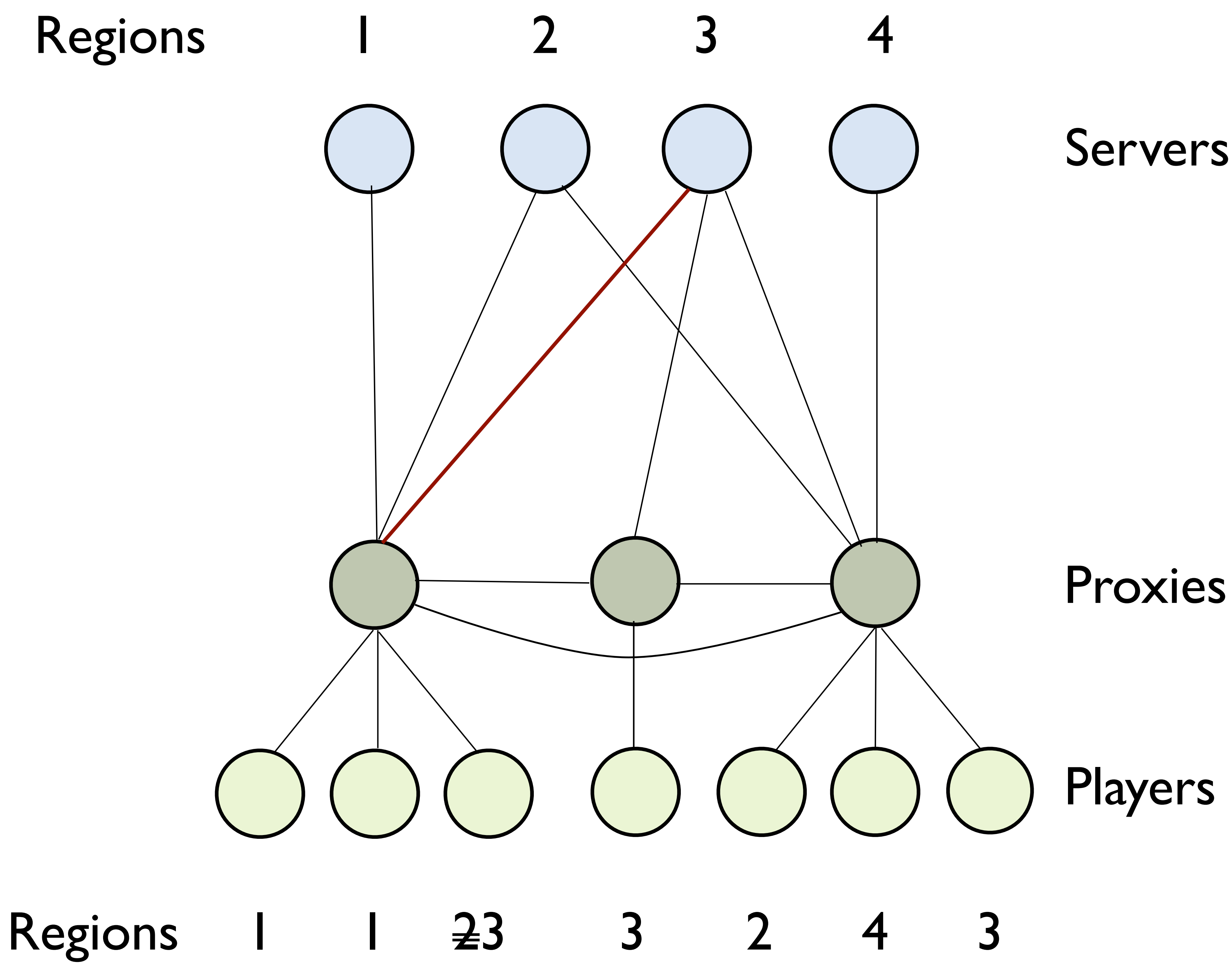
Time-stamping
Message Ordering
Interest Management
App-Level Multicast

Example 3:
Interest Management


Regions 1 2 3 4
Servers
Proxies
Players
Regions 1 1 2 3 2 4 3



Regions
1
2
3
4
Servers
Proxies
Players
Regions
1
1
2
3
2
4
3

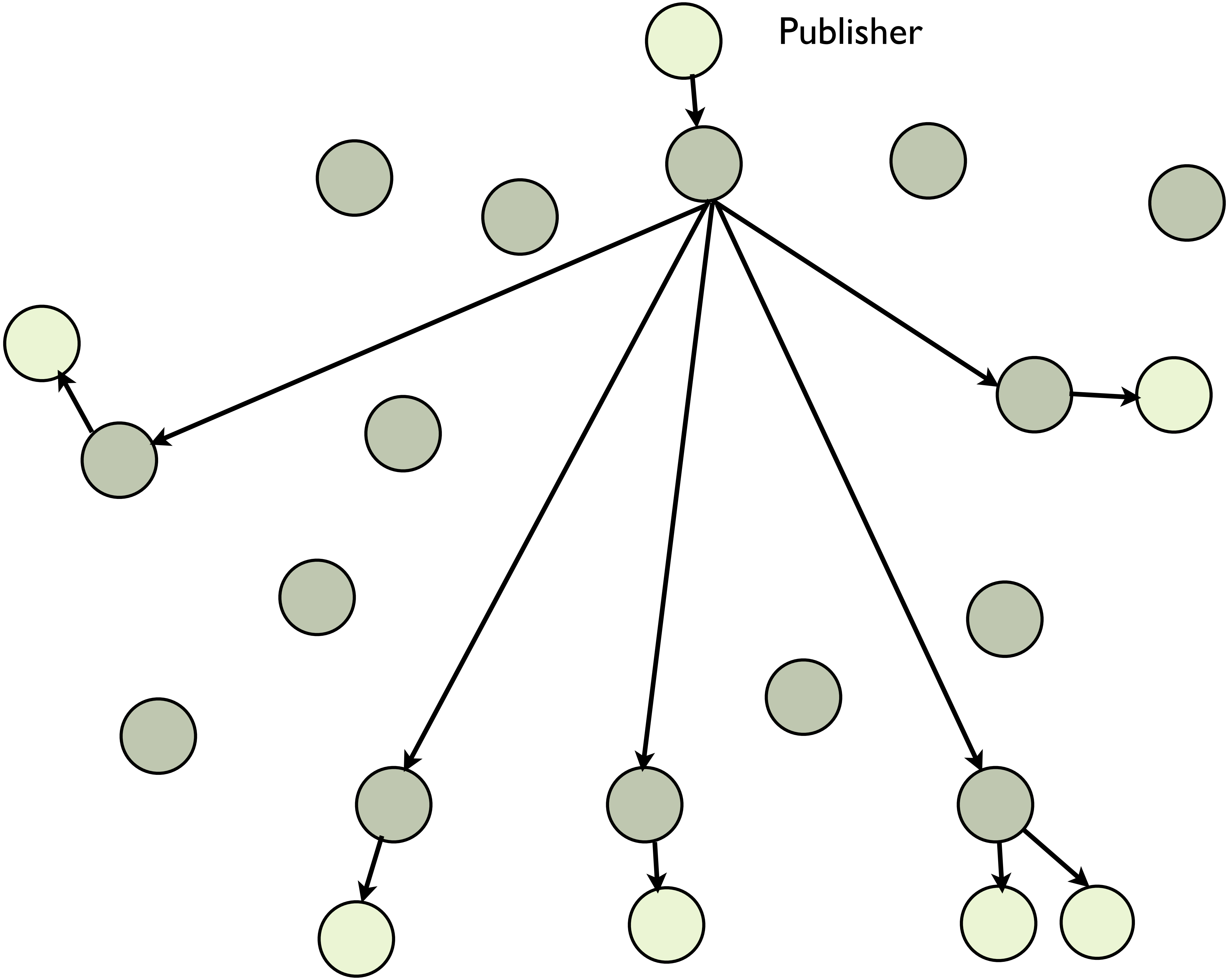

Regions 1 2 3 4
Servers
Proxies
Players
Regions 1 1 ~~2~~3 3 2 4 3

Advantages:

Reduced network cost at server

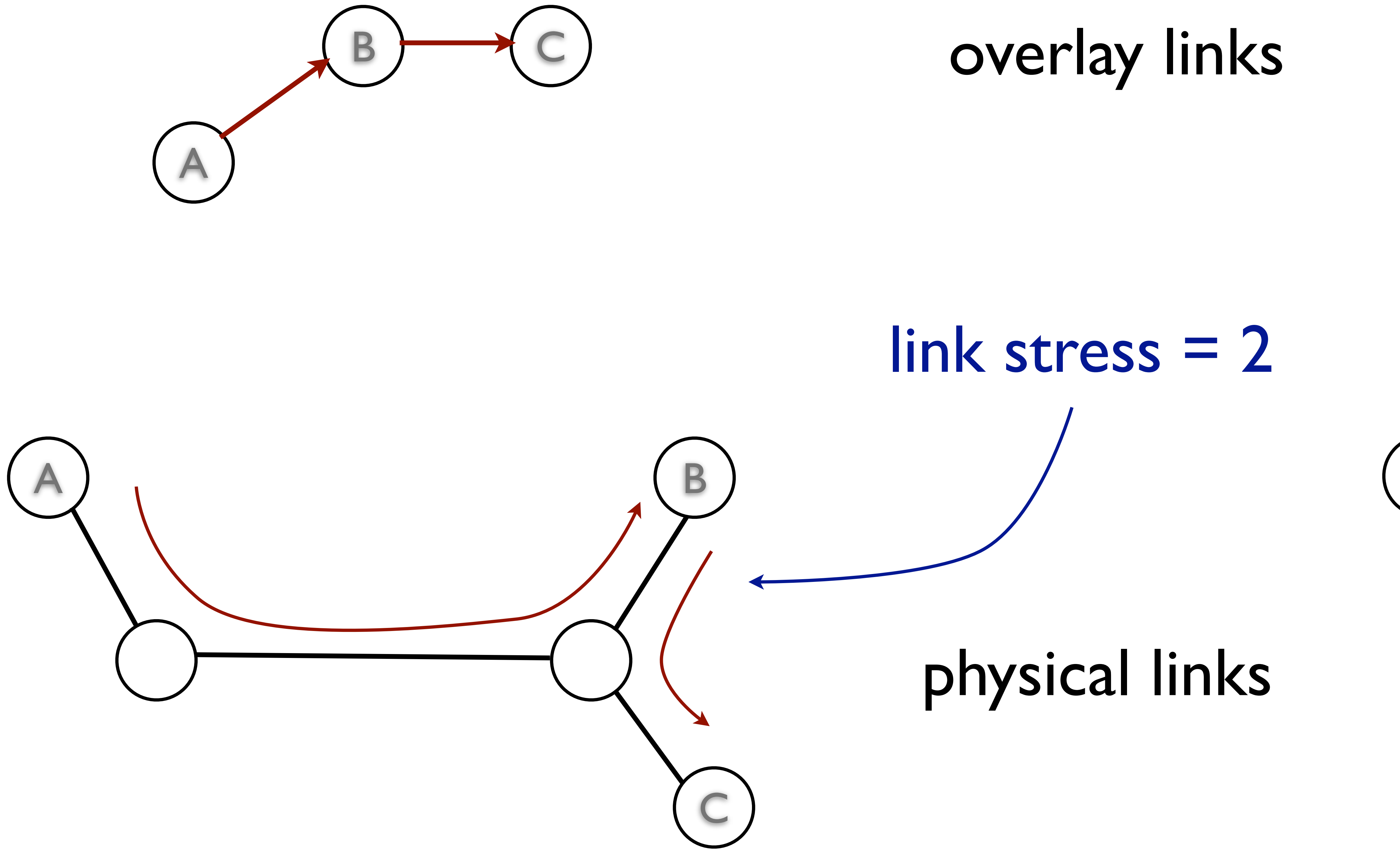
No connect/disconnect at clients when player moves/region migrates

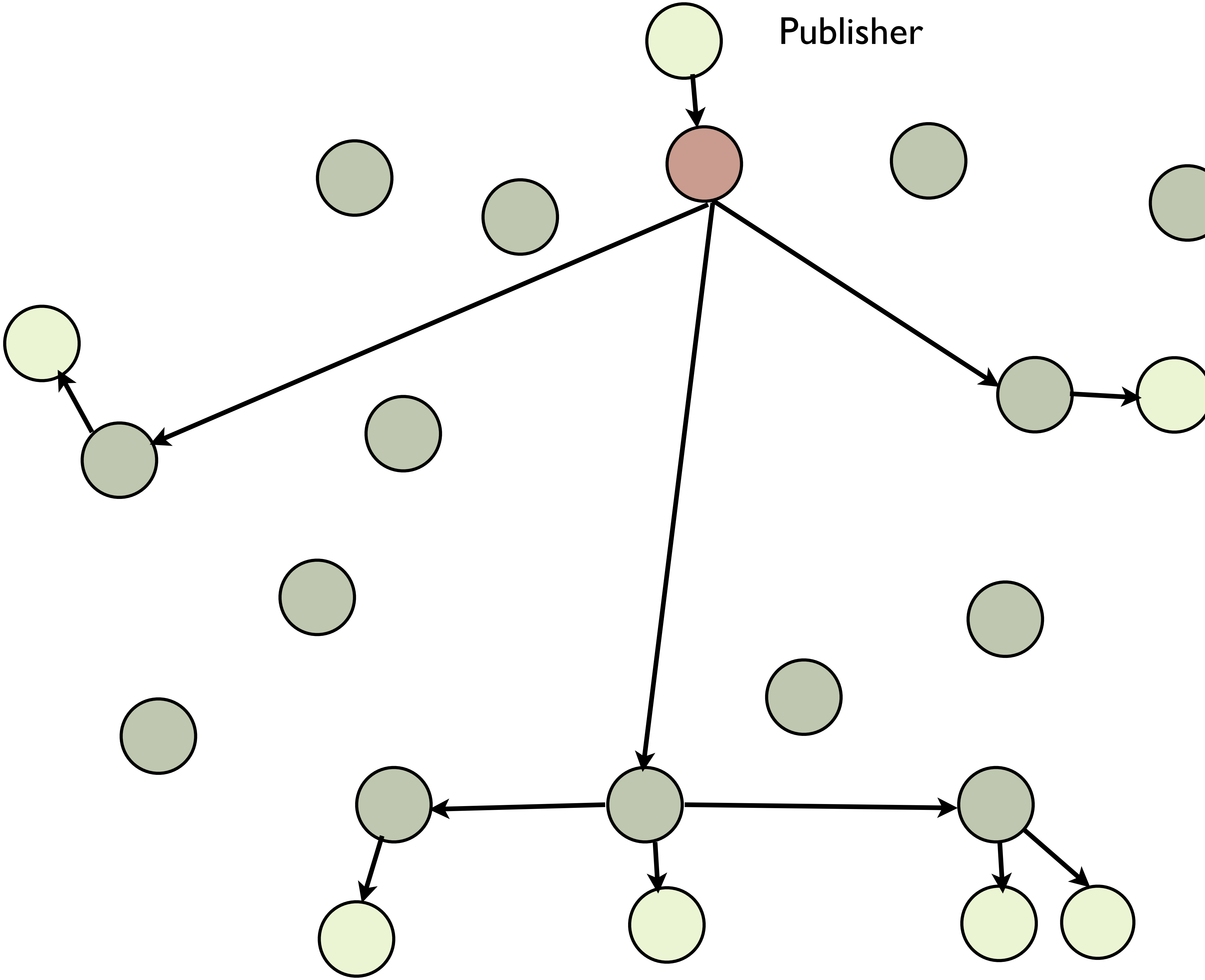
Example 4: Application-Level Multicast


Publisher

Direct connections between the proxies reduces latency, but incur additional cost at the proxy (there may be multiple publishers, multiple games)

Additional bandwidth cost as the same copy might traverse through the same physical link multiple times.


A
B
C
overlay links
link stress = 2
A
B
C
physical links

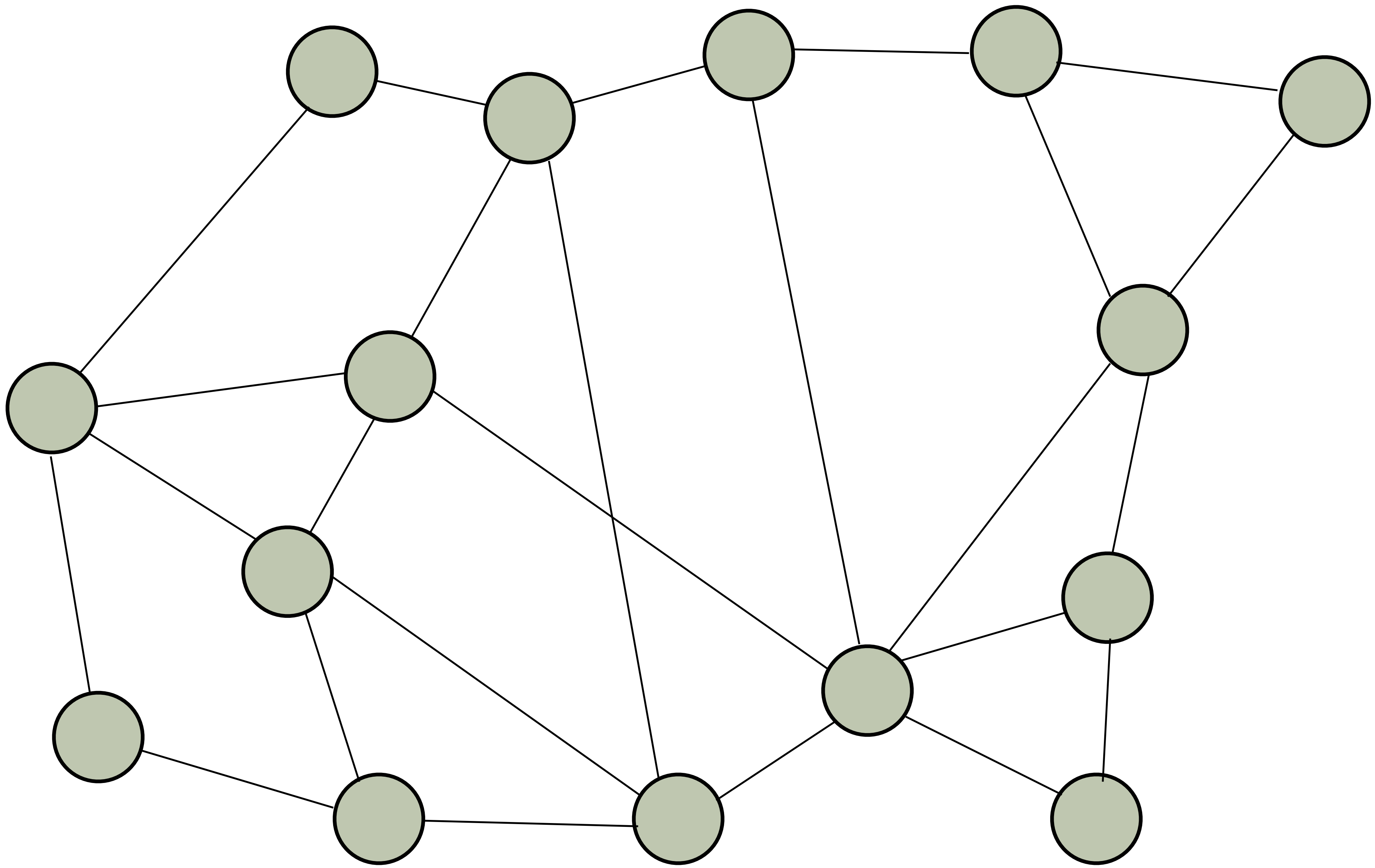

Publisher

How to construct a good application-level multicast tree?

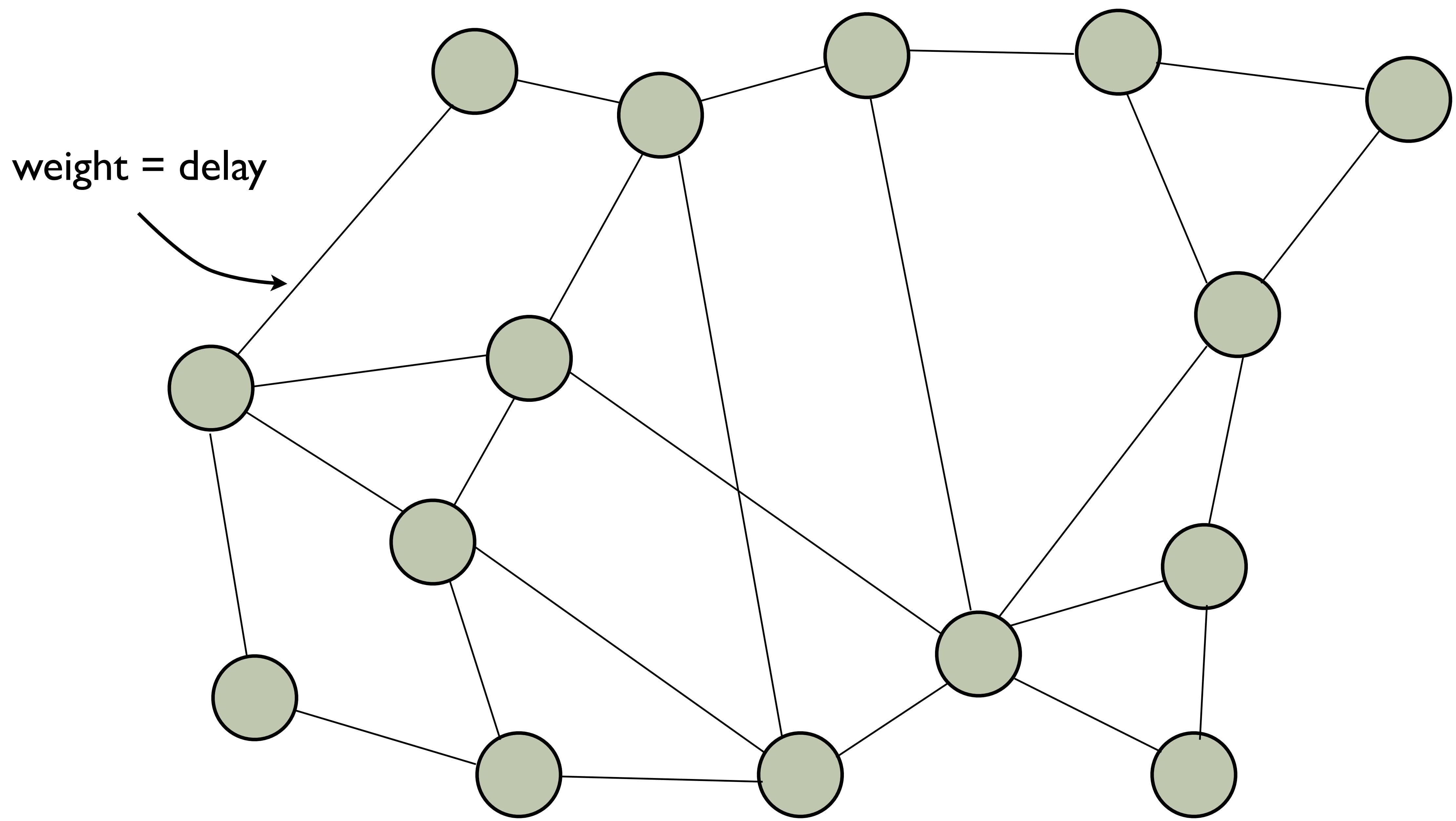
Evaluating all possible trees could be expensive.

Idea: build a mesh among the proxies first, then construct trees on top of the mesh.

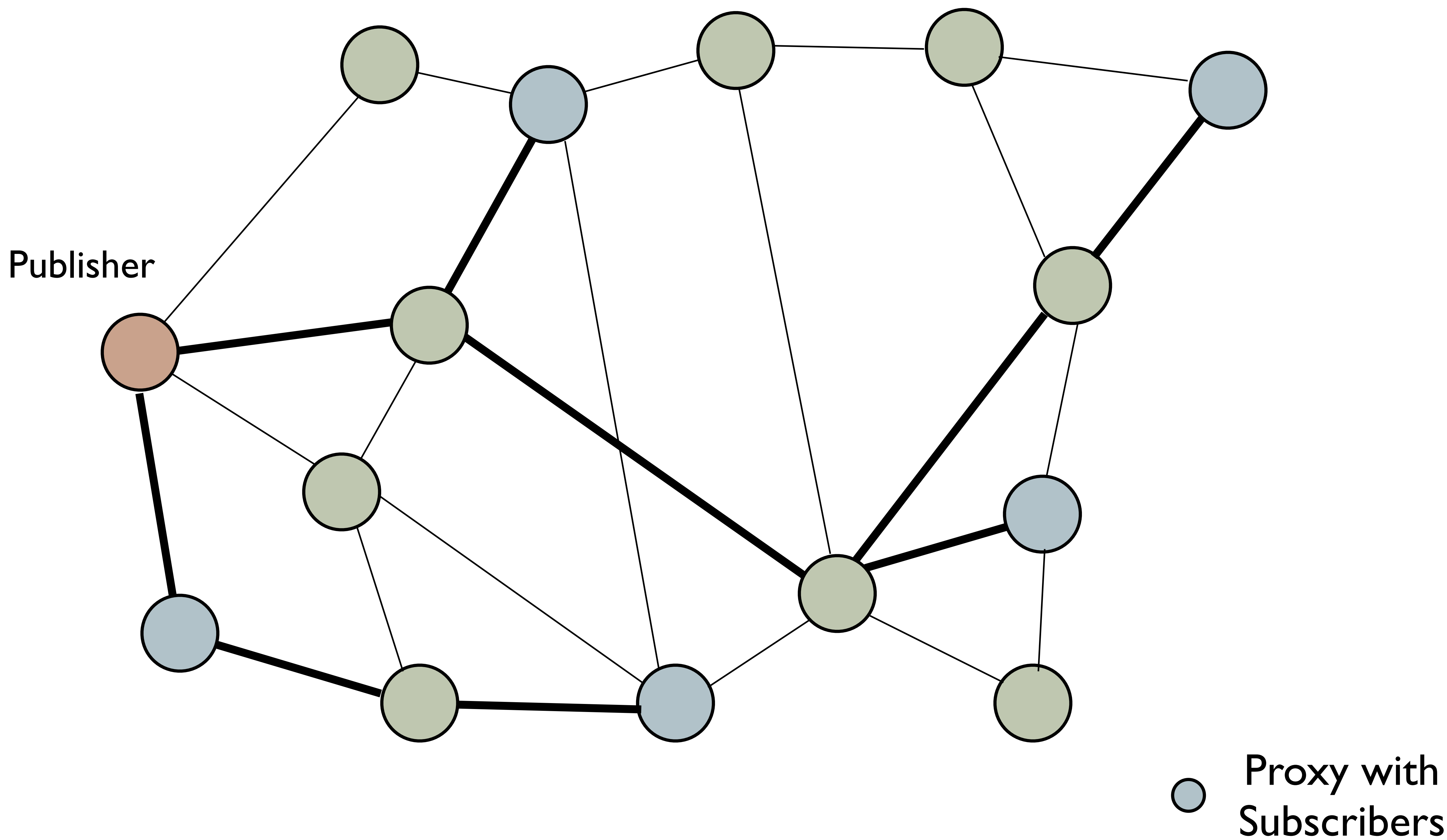
Proxies periodically send probes to other proxies, and “hook up” with proxies with good connection (latency, bandwidth, loss).



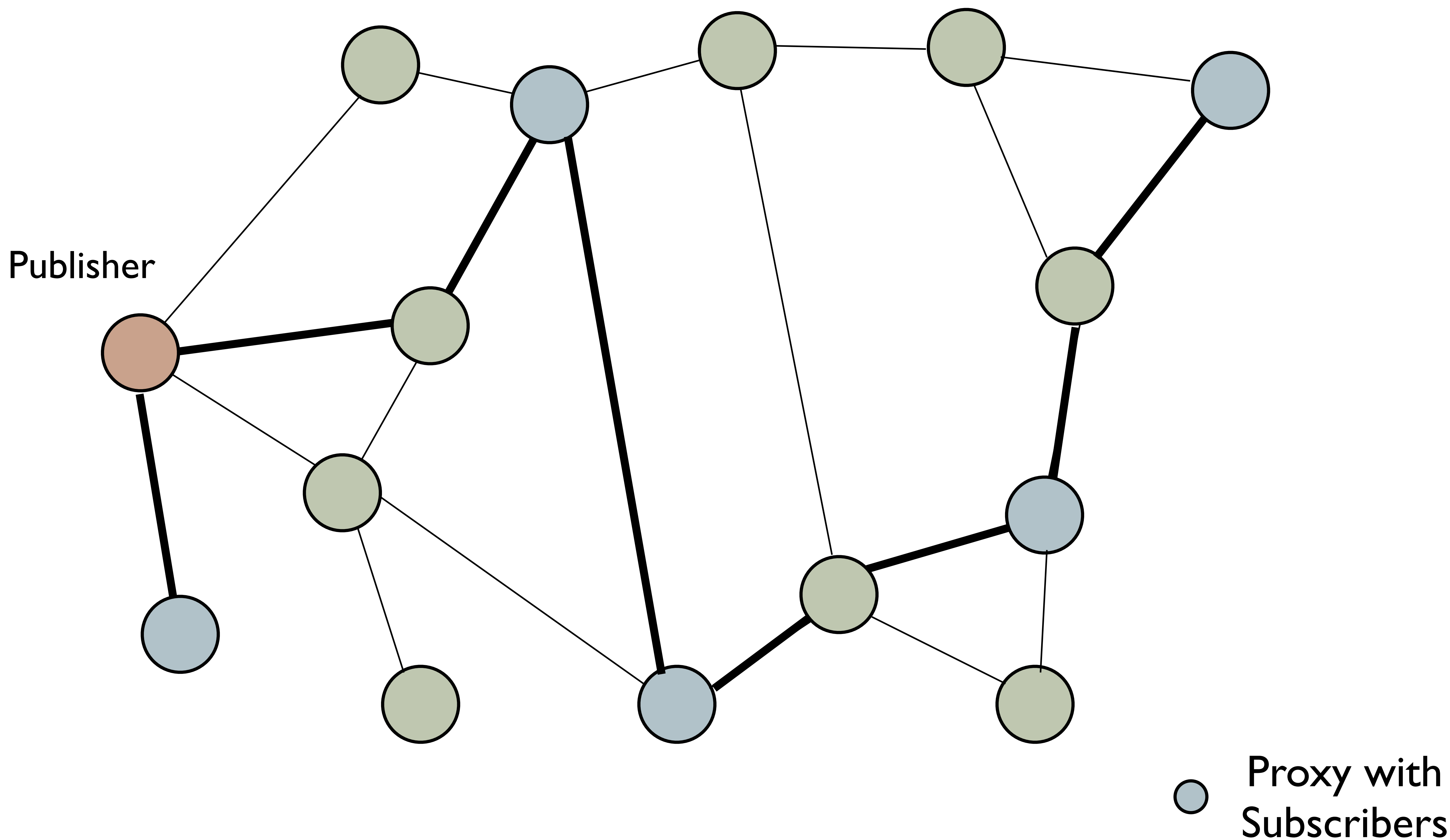
We can now model the proxy network as a weighted undirected graph.



Dijkstra algorithm allows us to find the shortest path tree from a proxy to all other proxies.



Shortest path tree does not minimize the cost. Minimum Steiner Tree does.



Minimum Spanning Tree:
Find a subset of edges such that,

all vertices in the graph are connected,

total cost is minimized

Minimum Steiner Tree: Find a subset of edges such that,

A given subset of vertices in the graph are connected

total cost is minimized

Why are we interested in minimizing total cost, which is equivalent to total delay?

$$\sum_{e \in \mathrm{Overlay}} \mathrm{delay}(e) \quad =$$

$$\sum_{e \in \mathrm{Overlay}} \sum_{i \in \mathrm{Physical}(e)} \mathrm{delay}(i) \quad =$$

$$\sum_{i \in \mathrm{Physical}} \mathrm{delay}(i) * \mathrm{linkstress}(i)$$

Minimum Steiner Tree is NP-complete

End-to-End delay is not bounded on Minimum Steiner Tree

How to balance the two objectives (trade-off between end-to-end delay and total cost) remains a challenging problem.

Why use a proxy network to build a multicast tree (rather than end-hosts?)

Advantages:

Proxies are relatively stable -- no expensive maintenance and reorganization of trees

No issue of cheating (peeking into/ modifying content of packets)

Examples of Proxy Services:

Time-stamping
Message Ordering
Interest Management
App-Level Multicast

Proxy servers:

trusted

game-independent

geographically close

highly available

useful