MLS

An Efficient Location Service for Mobile Ad Hoc Networks

Roland Flury
Roger Wattenhofer

Distributed Computing Group

ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Geographic Routing

- To each message, piggyback the position of the receiver
- The routing decision is solely based on this information
  - **Greedy forwarding**: A node forwards a message to its neighbor closest to the destination
  - **Face routing**: To surround routing voids, a message may be routed along the border of the hole

Geographic Routing

- **Advantages of georouting**
  – No routing tables, efficient and scalable
  – No setup time, each new node can participate immediately
  – Fully reactive (on demand) routing

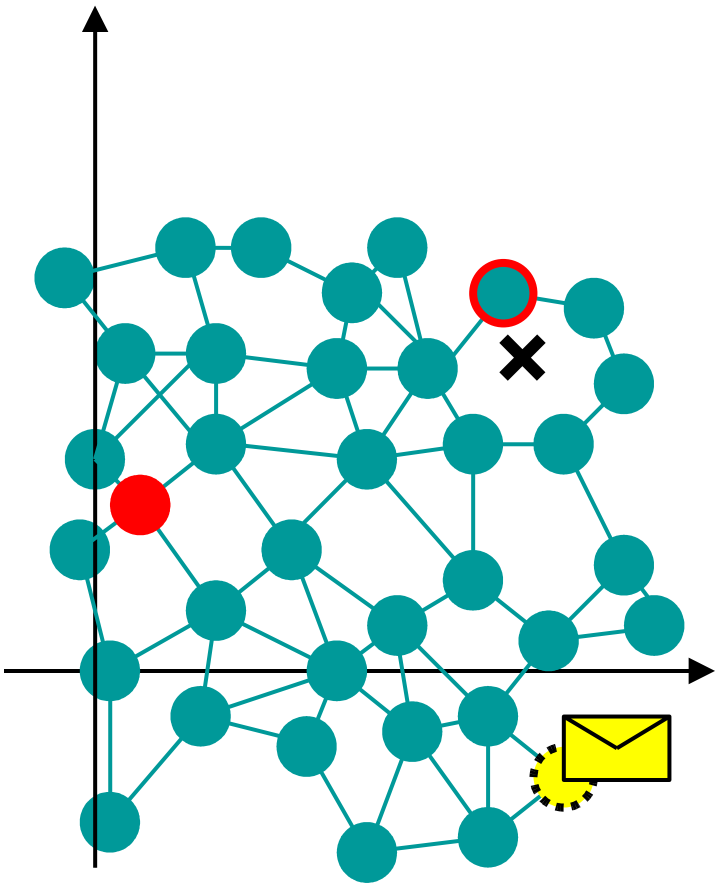
… once the position of the destination is known

Location Service

- **Q: How can a node determine the position of another node?**

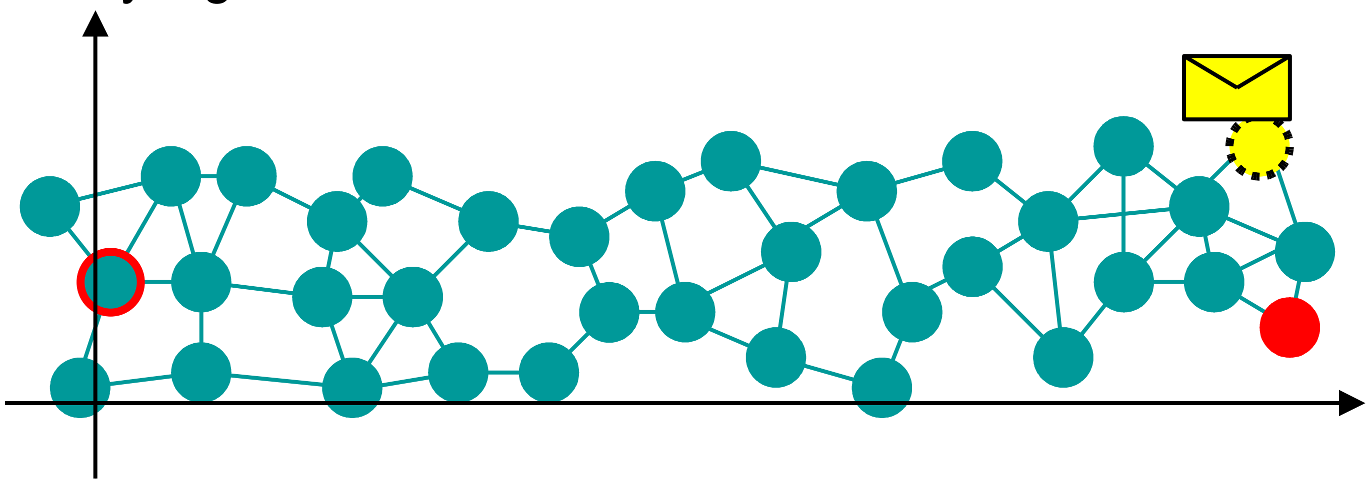
- **Proactive distribution**
  - Each node broadcasts its position
  - VERY expensive

- **Home based** location service
  - Each node has an associated place (its home) where it stores its location
  - The place is determined by the hash value of the node's ID
  - Any node can determine this place
  - This is a **GHT** (Geographic Hash Table)



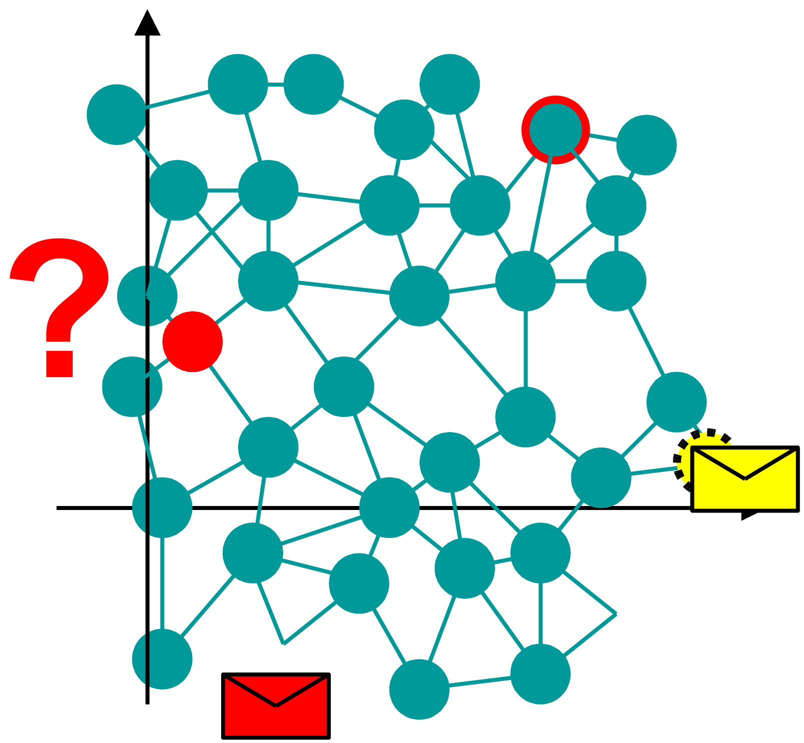
GHT: Home Based Location Service

- Each node has a single location server
  - Chosen at a random position
- Good load balancing
  - Each node has to store location information only for a few nodes
- Arbitrary high stretch:



Mobility

- Mobile nodes complicate location services tremendously
- The **position of a node cannot be known exactly**, as it may change continuously
- Any location information has to be considered **stale**
  - Lookup of position not wise, rather route messages through location server



Some Related Work in the field of Location Services

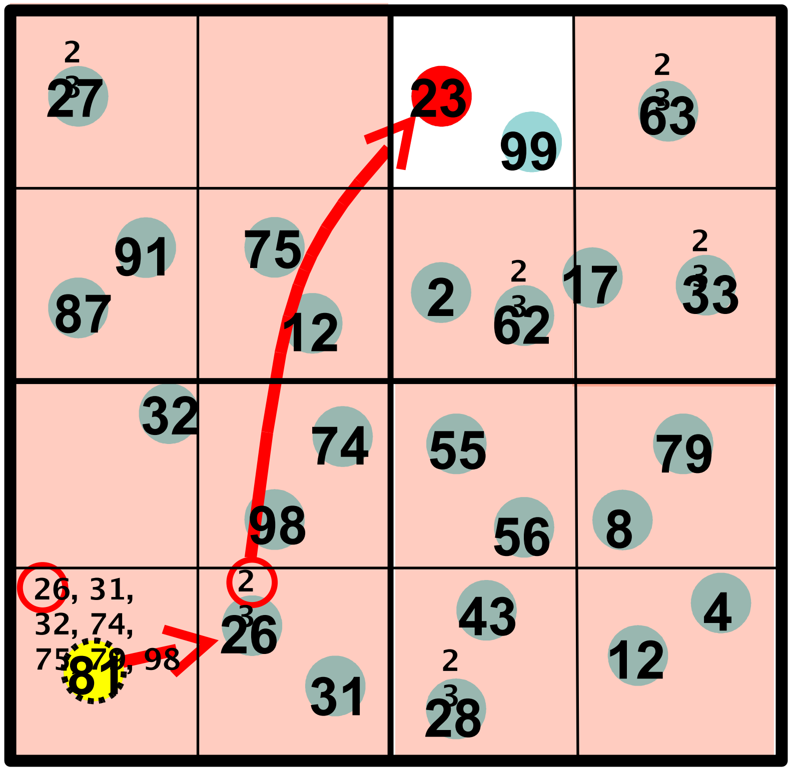
- **LLS**: "*LLS: A Locality Aware Location Service for Mobile Ad Hoc Networks*" by Abraham et al. @ DIALM-POMC 2004
- **GLS**: "*A Scalable Location Service for Geographic Ad Hoc Routing*" by Li et al. @ MobiCom 2000
- **DLM**: "*A Scalable Location Management Scheme in Mobile Ad Hoc Networks*" by Xue et al. in LCN 2001
- **HIGH-GRADE**: "*Enhancing Location Service Scalability with High-Grade*" by Yu et al. @ MASS 2005
- "*Scalable Ad Hoc Routing: The Case for Dynamic Addressing*" by Eriksson et al. @ InfoCom 2004
- "*Geographic Routing without Location Information*" by Rao et al. @ MobiCom 2003
- "*Topology Independent Location Service for Self-Organizing Networks*" by Rezende et al. @ MobiHoc 2005

Related Work (Excerpt) - GLS

- **GLS**: "*A Scalable Location Service for Geographic Ad Hoc Routing*" by Li et al. @ MobiCom 2000

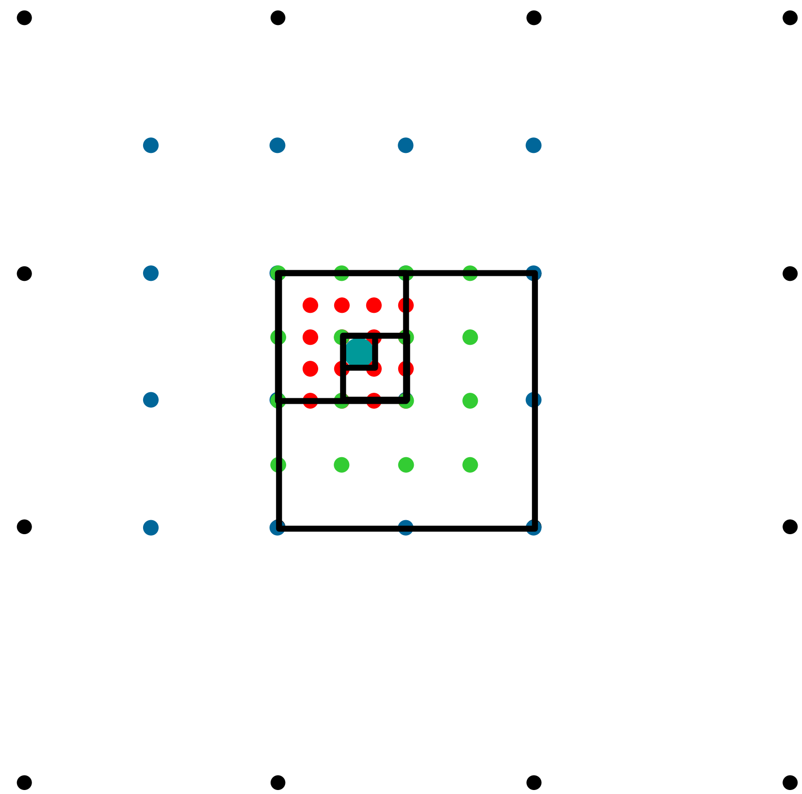
- **Lookup cost:** Size of the biggest surrounding square (level)

- **Publish cost:** not bounded



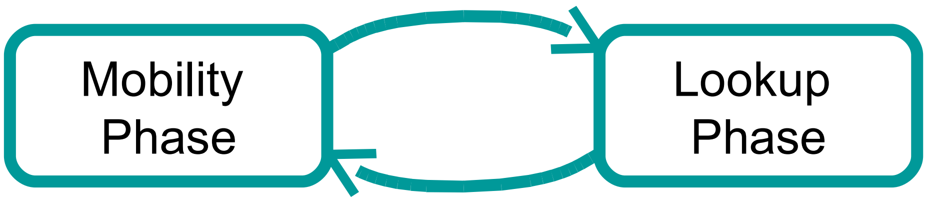
Related Work (Excerpt) - LLS

- **LLS**: "*LLS: A Locality Aware Location Service for Mobile Ad Hoc Networks*" by Abraham et al. @ DIALM-POMC 2004
- **Lookup cost:** O(d^2)
- **Publish cost (amortized):** O(d log d)



Mobility revisited

- GLS and LLS support mobile nodes, but not **concurrent lookup and mobility**
- Existing solutions assume that mobility and lookups do not interfere

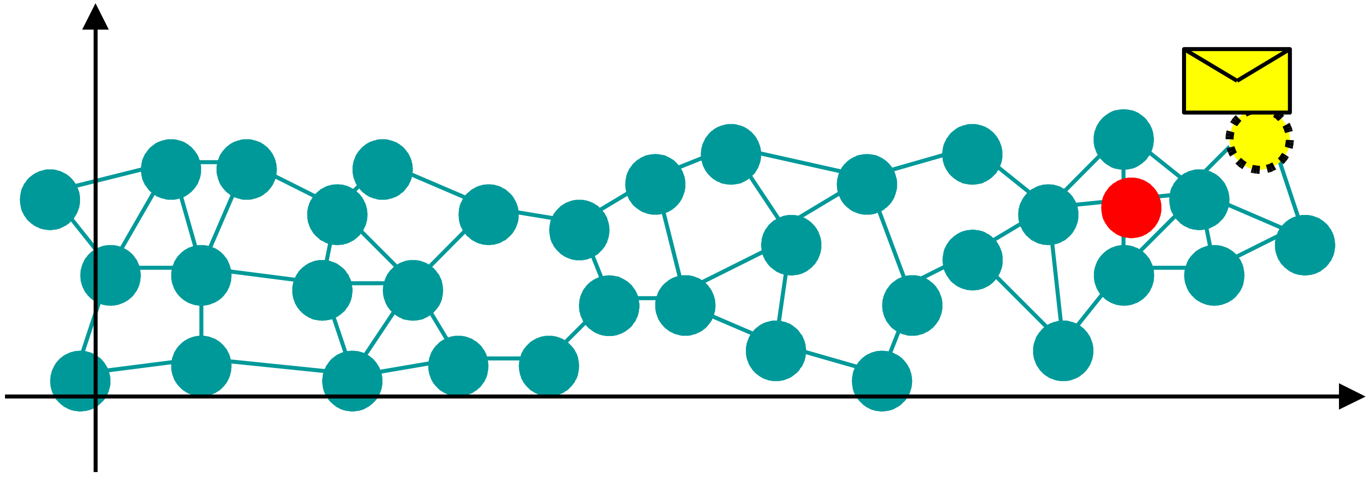


The nodes are mobile and update their location servers **XOR** The nodes perform lookups

- That's clearly not what we expect from a **M**ANET!
  - Nodes cannot move freely
  - Lookups / routing cannot happen concurrently
  - Synchronization would be required to switch between the two phases

Mobility – but limited

- Nodes may not move arbitrarily fast!
  - If a node moves faster than the message propagation speed, no routing algorithm can ensure delivery
- Thus, we must **limit the maximum node speed**
  - Maximum node speed can be expressed as a function of the message propagation speed of the underlying routing algorithm



Outline

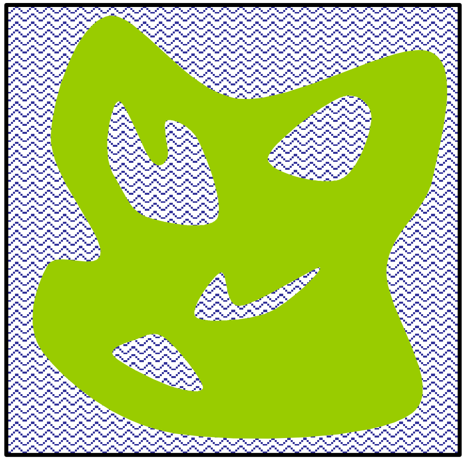
- **Motivation** for building location services
- **Mobility** and its issues
- **Related work**
- **MLS** our contribution
- **Model**
- **Algorithm & Analysis**
- **Conclusion**

MLS – Location Service for Truly Mobile Nodes

- The lookup service works despite of **concurrent mobility**
  - Nodes may move freely at any time and anywhere
  - Lookup and routing requests may execute at any time
- Routing is performed through the lookup mechanism
  - No stale location information
- **Lookup / routing overhead**: close to optimal
  - Routing to a node costs O(d), where d is the distance
- Moderate **publish overhead** due to mobility
  - Amortized bit-meter cost is O($d \log d$) for moving a distance d
- Quite **fast moving nodes**!
  - Nodes may move up to 1/15 of the message propagation speed
  - On arbitrary paths

Model

- **Deployment Area**
  - Nodes populate land areas
  - Lake denote holes
  - Connected graph, no islands
- **Connectivity**
  - n_1, n_2 are connected if $d(n_1, n_2) \leq r_{min}$
- **Density**
  - For any point on land, there exists a node at most r_{min} / 3 away
  - Thus, relatively dense node deployment
- **Node Equipment**
  - Position module (GPS, Galileo, local system, …)
  - Communication module
- **Underlying Routing**
  - Given a destination position p_t, we can route a message in $\eta\, d(p_s, p_t)$ from the sender position p_s to p_t

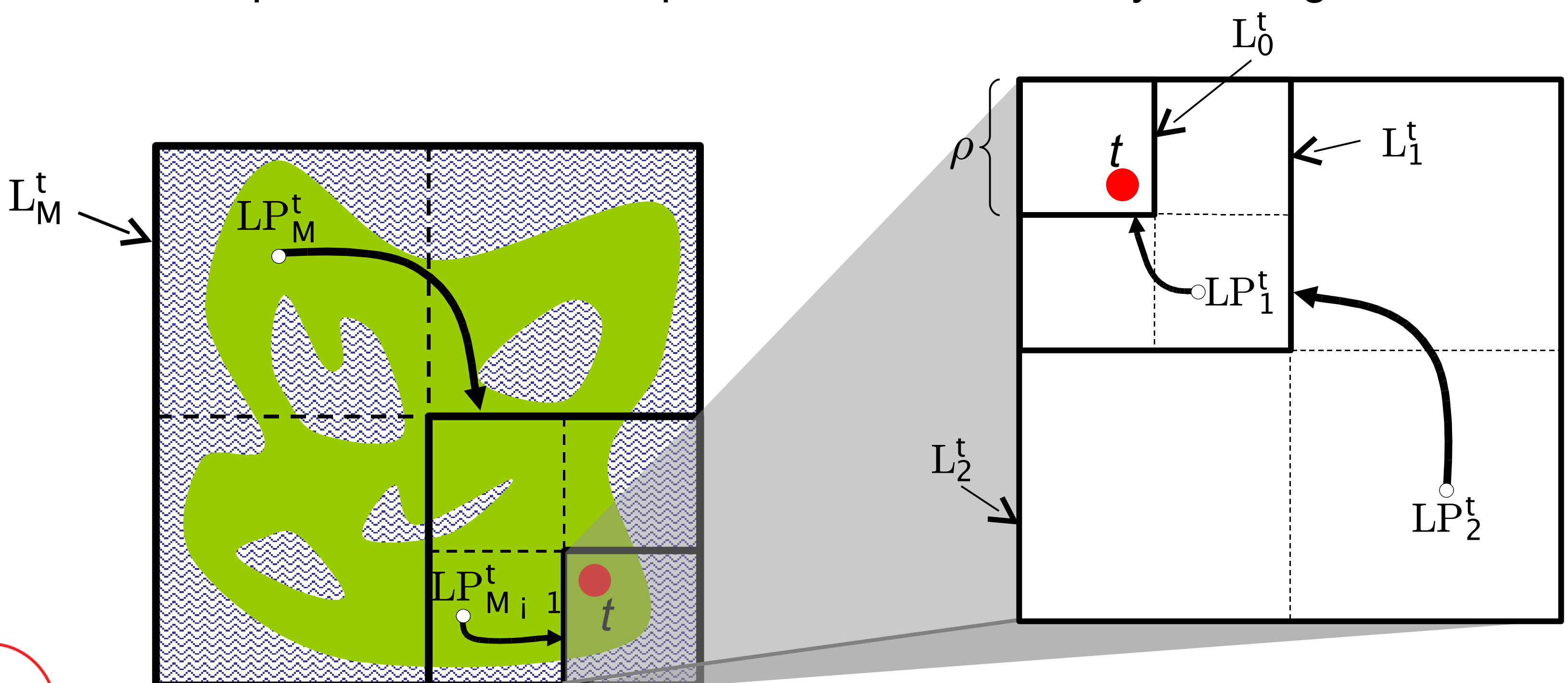


Outline

- **Motivation** for building location services
- **Mobility** and its issues
- **Related work**
- **MLS** our contribution
- **Model**
- **Algorithm & Analysis**
- **Conclusion**

Selection of Location Servers

- Each node builds a hierarchy of location servers that are located in exponentially increasing areas around the node.
  - Top **level** surrounds entire world
  - Each level is divided in 4 sub-squares
  - A **level pointer** points to the next smaller level that surrounds *t*
  - The position of the level pointer is determined by hashing the ID of *t*

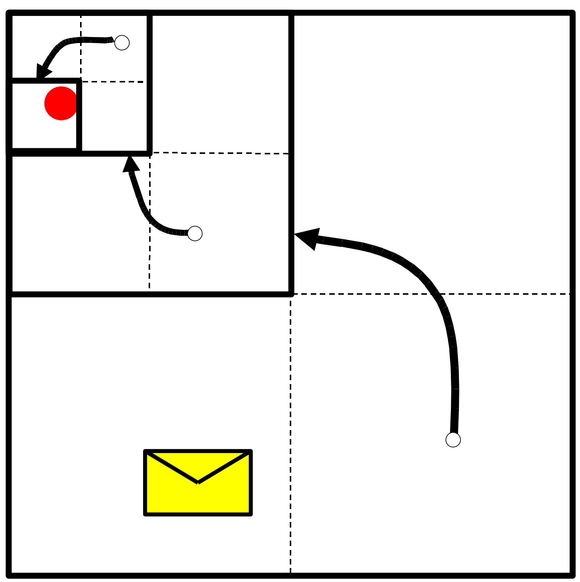


Routing in MLS

- Routing in MLS consists of two phases

1) Find a Location Pointer of the destination
2) Recursively follow the Location Pointers

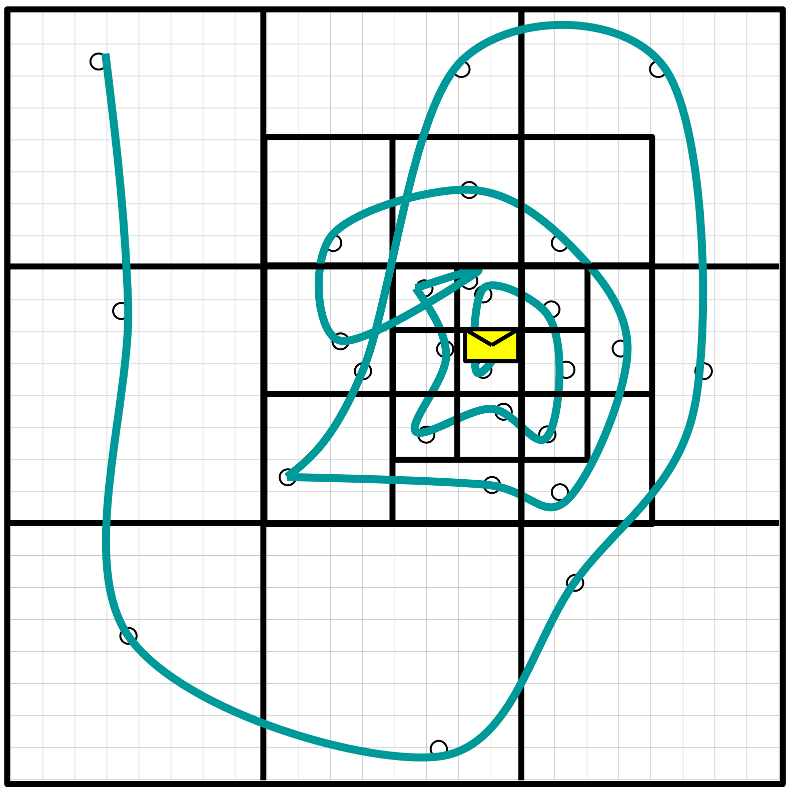
- The second step pretty easy:



Performance
If the destination is *d* away from sender, the lookup path is O(*d*).

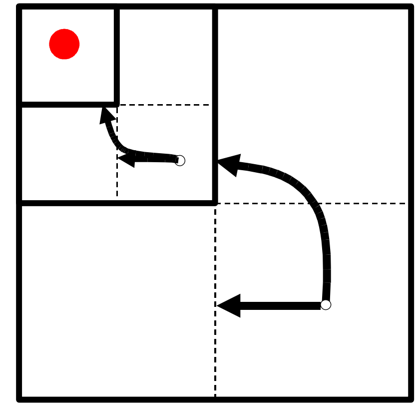
How to find a Location Pointer

- First, the sender assumes that the destination is in its vicinity
- While the lookup request fails to find a location pointer, it increases the search area

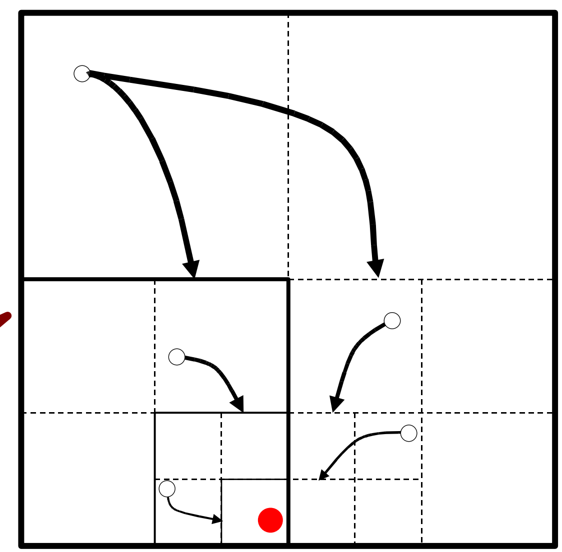


Performance
If the destination is *d* away from sender, the lookup path to *find a first location pointer* is O(*d*).

Supporting Mobility



- A location pointer only needs to be updated when the node leaves the corresponding sub-square.
- *Most of the time*, only the closest few location pointers need to be updated due to mobility
- **Not enough to guarantee low publish overhead!**
  - If node oscillates over grid-boundary of several layers, *many* location pointers need to be updated.

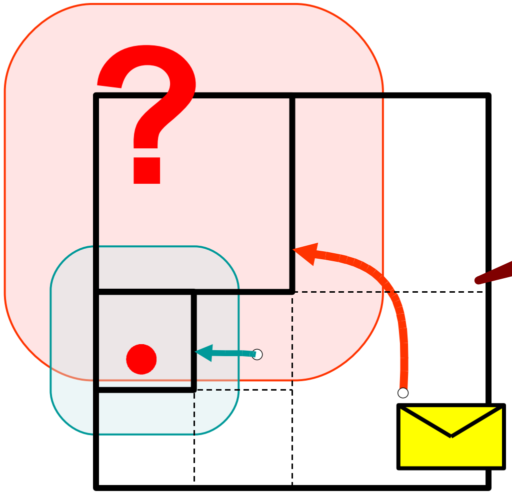


Unbounded publish cost!

Lazy Publishing

- To overcome exorbitant publish cost due to oscillating ε–moves:

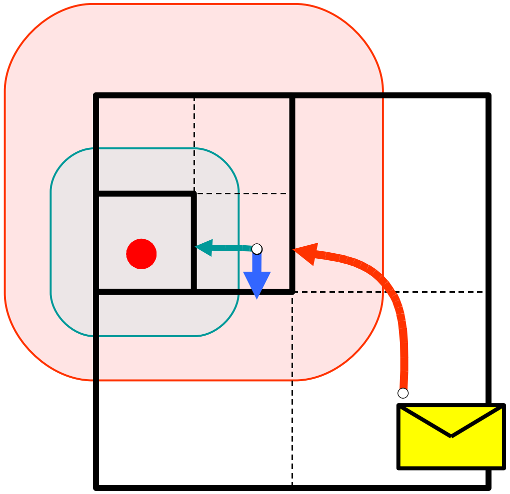
Only update a location pointer if the node has moved away quite a bit



Breaks the location pointer chain!

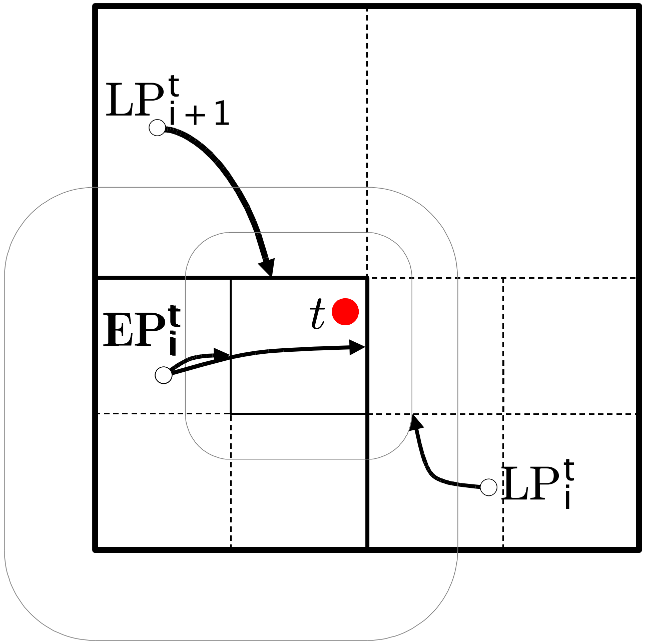
Lazy Publishing with Forwarding Pointers

- To repair the lookup path, add a **forwarding pointer** that points to the neighboring level that contains the location pointer.



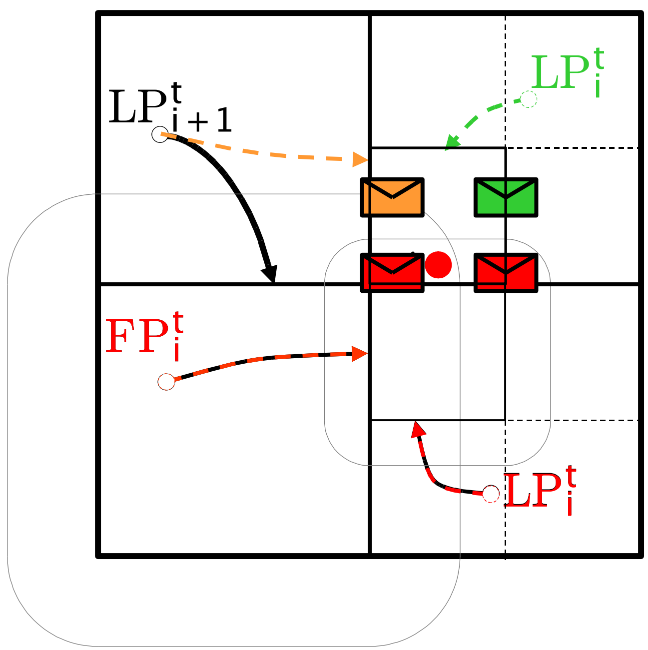
Supporting Concurrency

- Allowing for **concurrent lookup requests and node mobility** is somewhat tricky



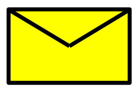
Supporting Concurrency

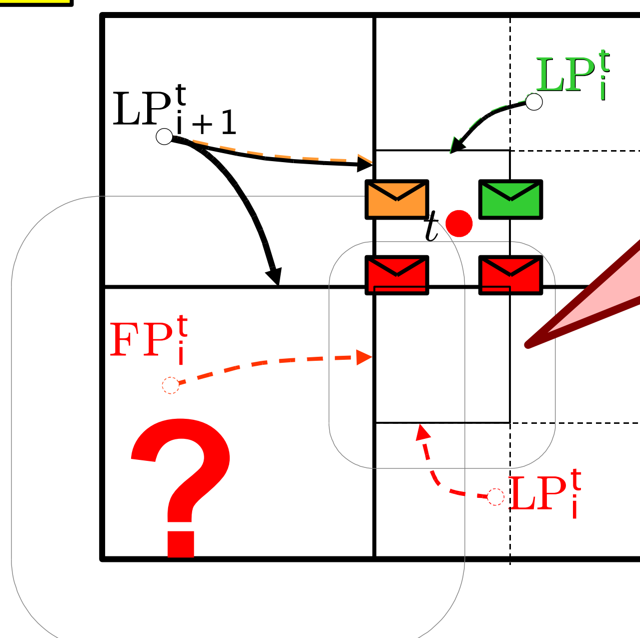
- Allowing for **concurrent lookup requests and node mobility** is somewhat tricky



Supporting Concurrency

- Allowing for **concurrent lookup requests and node mobility** is somewhat tricky
  - Especially the *deletion* of location pointers and forwarding pointers
- **Routing of messages needs time**
  - Sending a message to the next location pointer
  - Sending command messages to update / delete / create a location pointer





Note:
These problems arise independently of the node speed.

Supporting Concurrency – TFP

- Solution to overcome the concurrency issue:

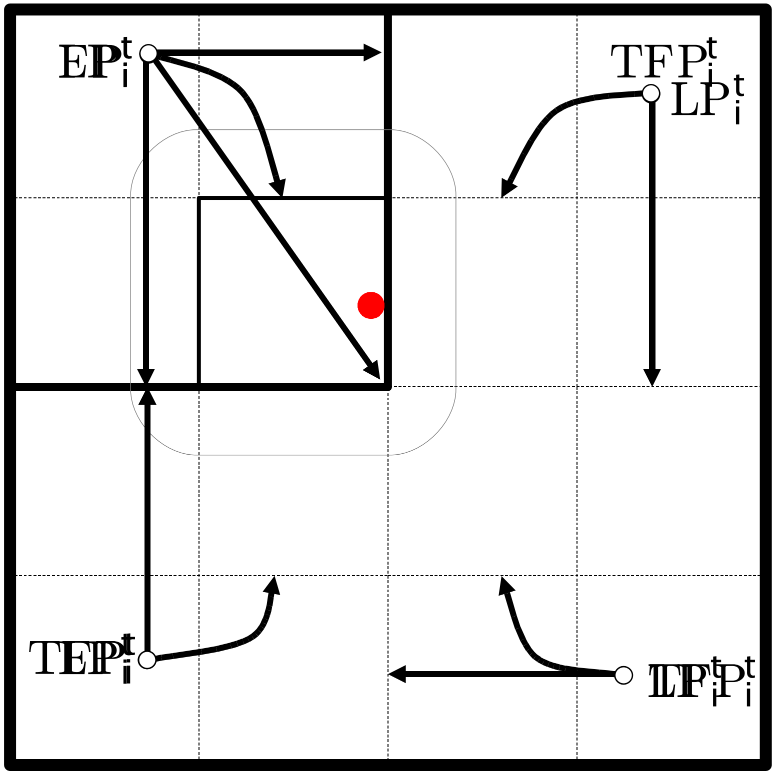
Do not delete a location or forwarding pointer, but replace it with a Temporary Forwarding Pointer (TFP)

- A temporary pointer redirects a lookup to the neighbor level where the node is located.

TFP are temporary and must be removed after a well known time.

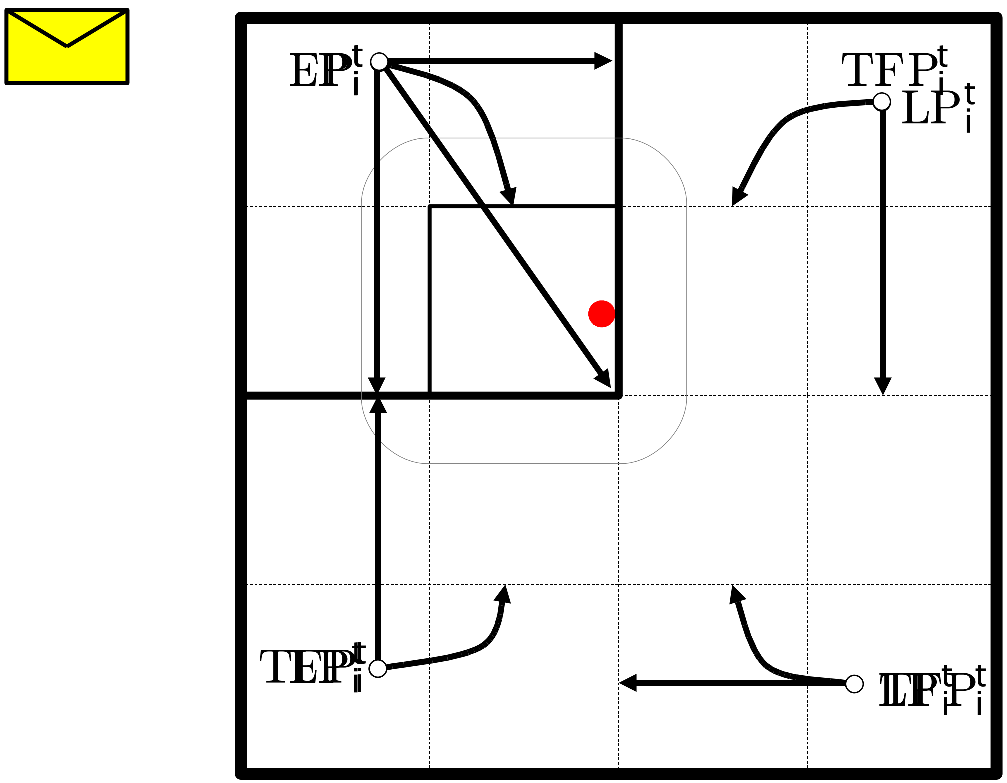
Speeding

- A mobile node may generate many forwarding pointers while a lookup searches for it
  - If the lookup is not fast enough, it permanently follows forwarding pointers



Speeding

- A mobile node may generate many forwarding pointers while a lookup searches for it
  - If the lookup is not fast enough, it permanently follows forwarding pointers



Performance of MLS

- The **maximum node speed** depends on several parameters
  - Min. speed of underlying routing
  - Lazyness in lazy publishing
  - How long we are willingly to follow temporary and forwarding pointers of a moving node

- […] Without lakes, the maximum node speed m… of the minimum message speed of the underlying …

Please see paper for details…

$$v_{node} \leq \frac{v_{msg}}{15}$$

- Despite of this relatively high node speed
  - **Lookup cost is O(*d*)**
  - **Amortized publish cost is O(*d* log *d*)**

Thank you!

Questions / Comments?

Distributed
Computing
Group

Roland Flury
Roger Wattenhofer

BACKUP

- Publish Algorithm
- Lookup Algorithm
- Nomenclature
- GLS (Related work)
- LLS (Related work)
- LLS WC

Publish Algorithm

if($\delta_i^t \geq \alpha \cdot \rho \cdot 2^i$) {
    if(i>0) {change FP_i^t in $^*\mathrm{LP}_{i+1}^t$ to TFP_i^t;}
    if($\mathrm{LP}_{i+1}^t \in \mathrm{L}_{i+1}^t$) {
        change LP_{i+1}^t to point to L_i^t;
    } **else** {
        if($\mathrm{LP}_{i+1}^t \in {}^*\mathrm{LP}_{i+2}^t$) {
            change LP_{i+1}^t to FP_{i+1}^t that points to L_{i+1}^t;
        } **elseif**($\mathrm{L}_{i+1}^t = {}^*\mathrm{LP}_{i+2}^t$) {
            change LP_{i+1}^t to TFP_{i+1}^t that points to L_{i+1}^t;
        } **else** {
            change LP_{i+1}^t to TFP_{i+1}^t that points to L_{i+1}^t;
            change FP_{i+1}^t to point to L_{i+1}^t;
        }
        on L_{i+1}^t, add LP_{i+1}^t that points to L_i^t;
    }
    if(i>0 **and** $\mathrm{LP}_i^t \notin \mathrm{L}_i^t$) {
        add FP_i^t on L_i^t that points to $\mathrm{L}_i \ni \mathrm{LP}_i^t$;
    }
}

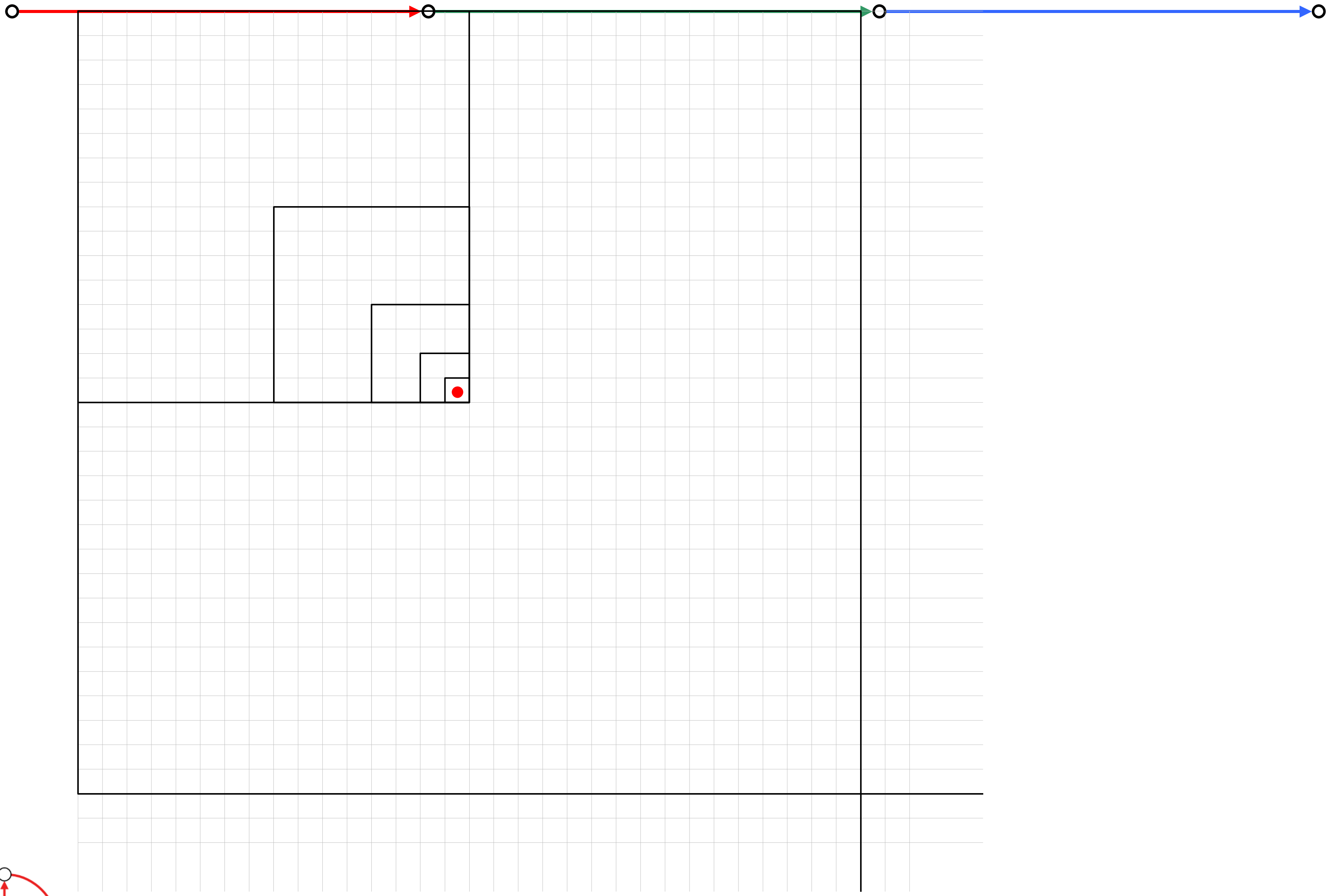
Lookup Algorithm

if($t \in \mathrm{L}_0^s \cup (\mathrm{L}_0^s)^8$) { exit(); }
for(i=1; **true**; i++) {
    if($P_i^t \in \mathrm{L}_i^s \;||\; P_i^t \in (\mathrm{L}_i^s)^8$) {
        $p = P_i^t$;
        break;
    }
}
Follow p until LP_1^t is reached.
Route to a node closest to an arbitrary point on land in $^*\mathrm{LP}_1^t$.
Forward to t.

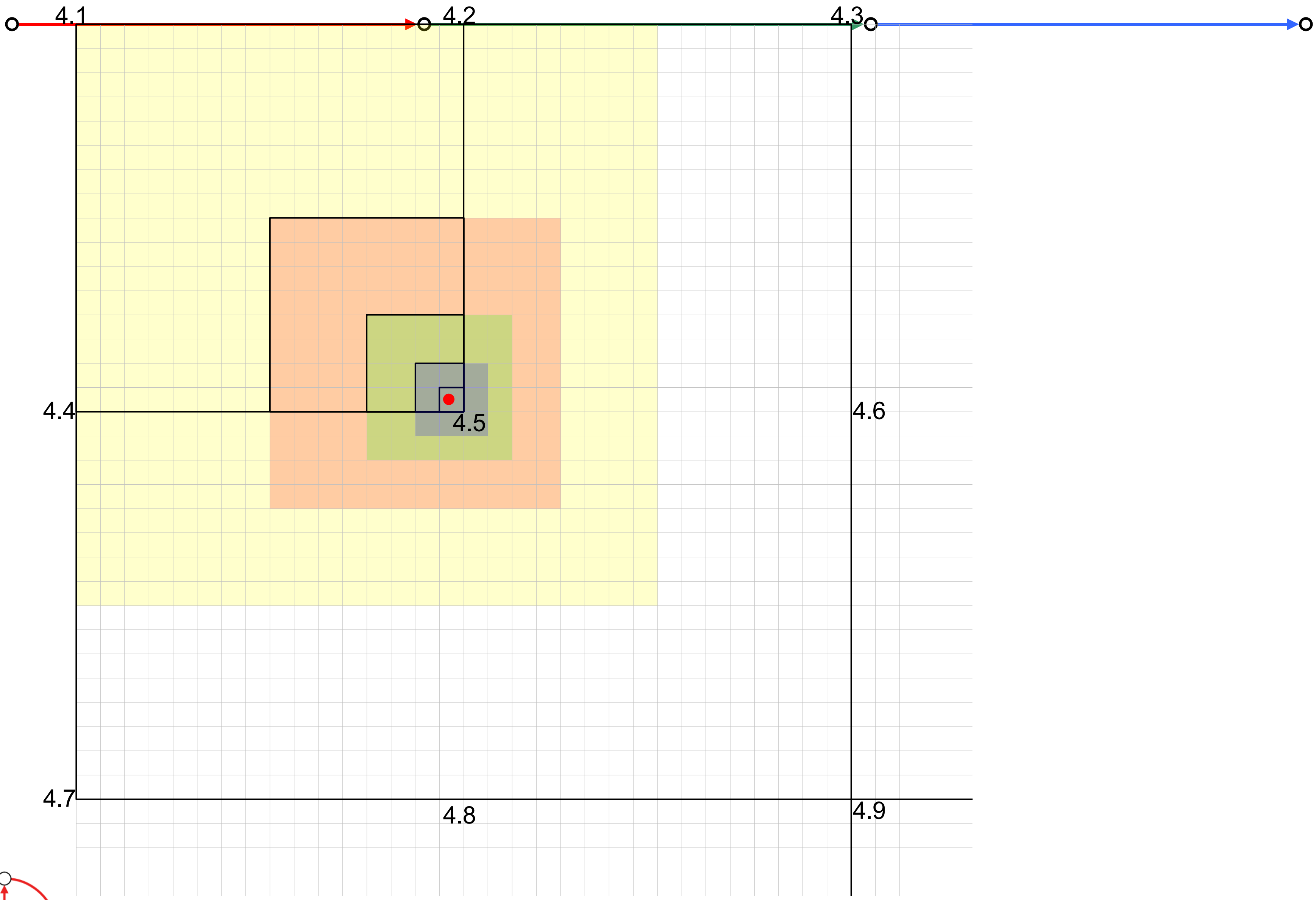
Nomenclature

L_i^t	Level that contains t with side-length $\rho \cdot 2^i$
$(\mathrm{L}_i^s)^8$	The 8 surrounding squares of L_i^s
LP_i^t	Level pointer on L_i for node t; points to L_{i-1}
$^*\mathrm{LP}_i^t$	The L_{i-1} where LP_i^t points to
δ_i^t	distance of a node t to $^*\mathrm{LP}_{i+1}^t$
FP_i^t	Forwarding pointer if $\mathrm{LP}_i^t \notin {}^*\mathrm{LP}_{i+1}^t$
$^*\mathrm{FP}_i^t$	The L_i where $^*\mathrm{FP}_i^t$ points
TFP_i^t	Temporary forwarding pointer, before a pointer to t is removed
$^*\mathrm{TFP}_i^t$	The L_i where TFP_i^t points
TTL_i	Time to live of a TFP_i
v_{max}^{node}	Max. speed of nodes
r_{min}	Min. communication range of a node
λ	Min. distance to a node from any land point
ρ	Side length of L_0; $\rho = \lambda/\sqrt{2}$
M	L_M surrounds the entire world
α	When $\delta_i^t \geq \alpha \cdot \rho \cdot 2^i$, LP_{i+1}^t is updated
$\beta(\beta_T)$	Max. number of forwarding hops to reach LP_i^t from a FP_i^t (TFP_i^t)
γ	See Lemma 8.2
η	Routing overhead to route to a given position

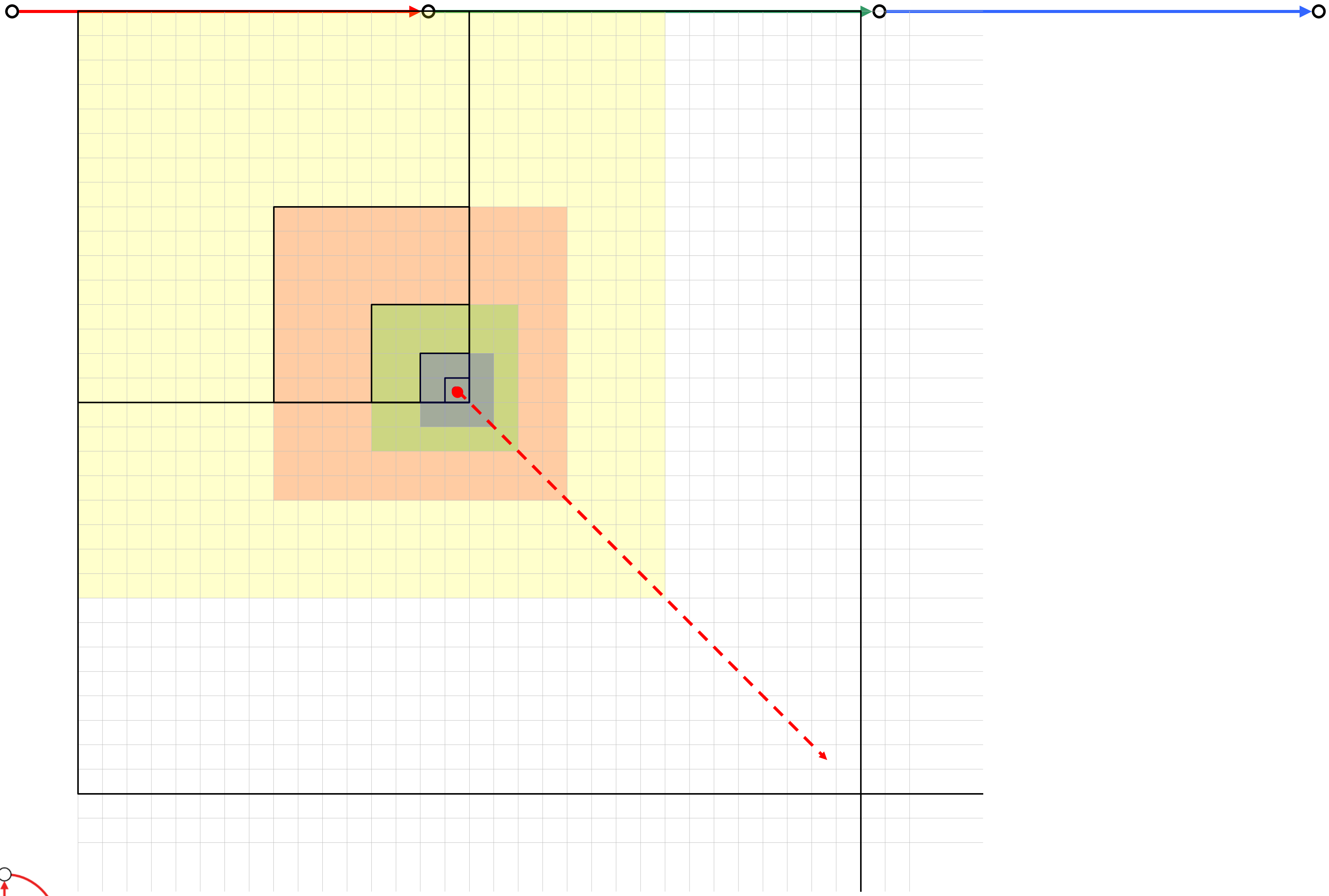
LLS Worst Case



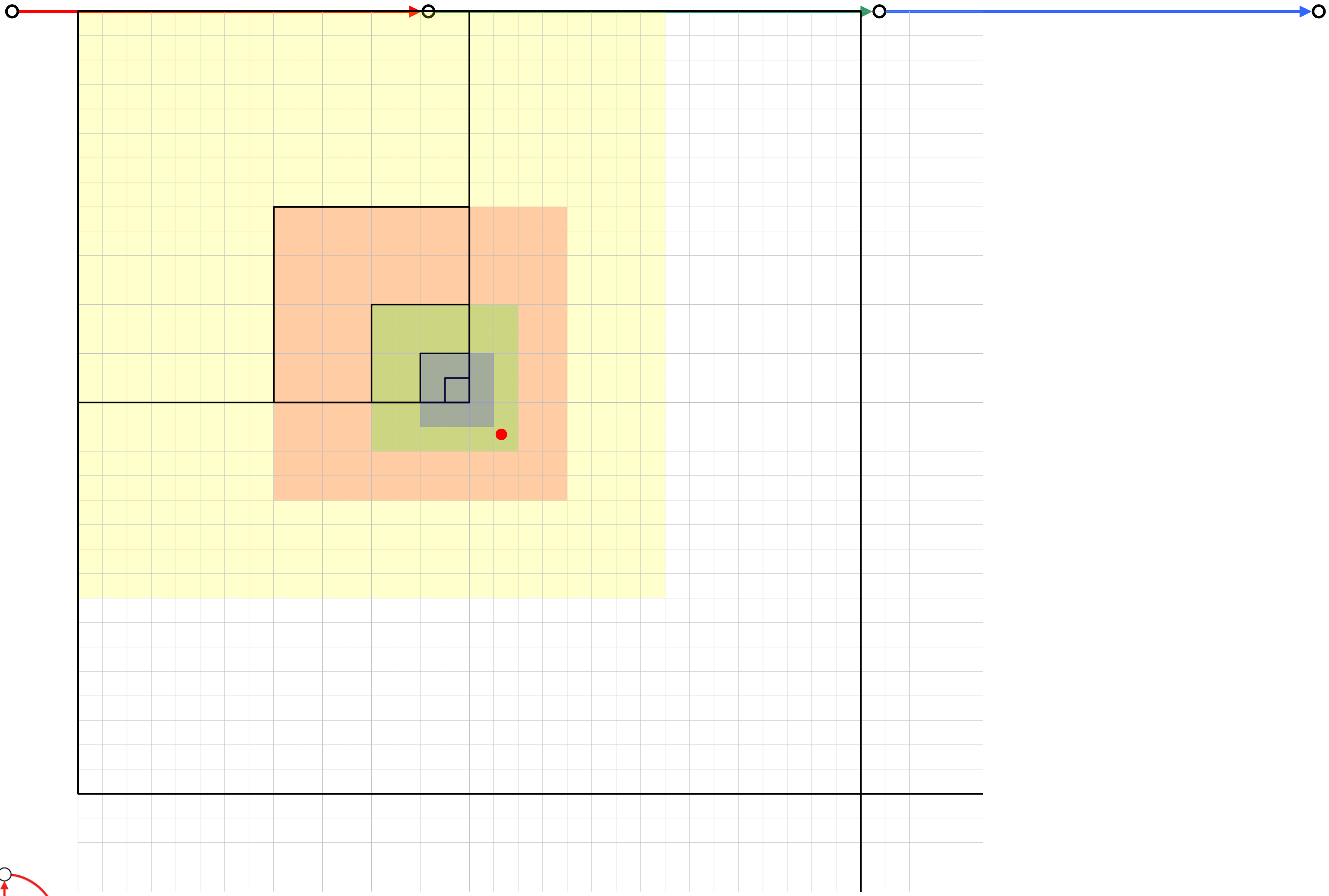
LLS Worst Case



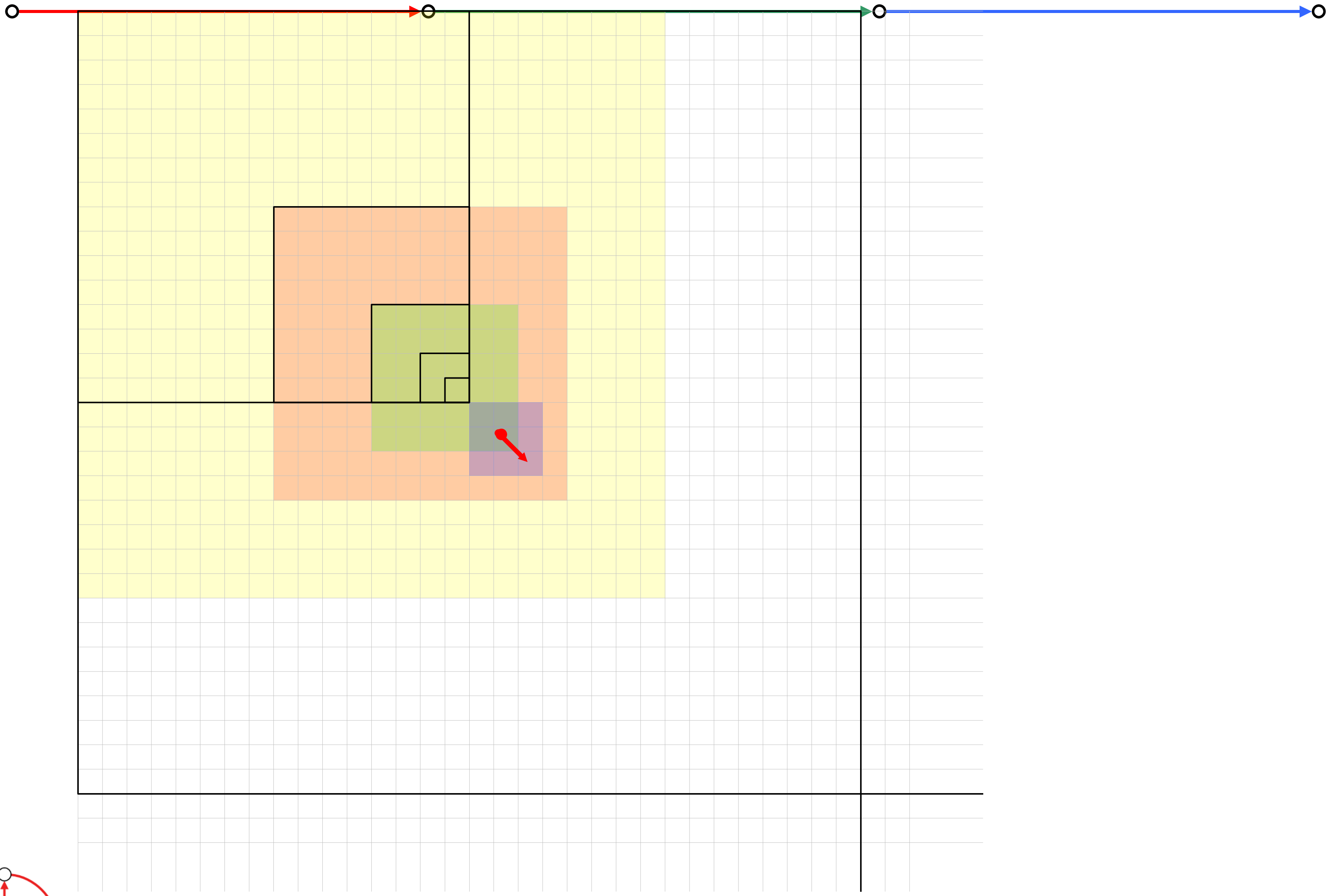
LLS Worst Case



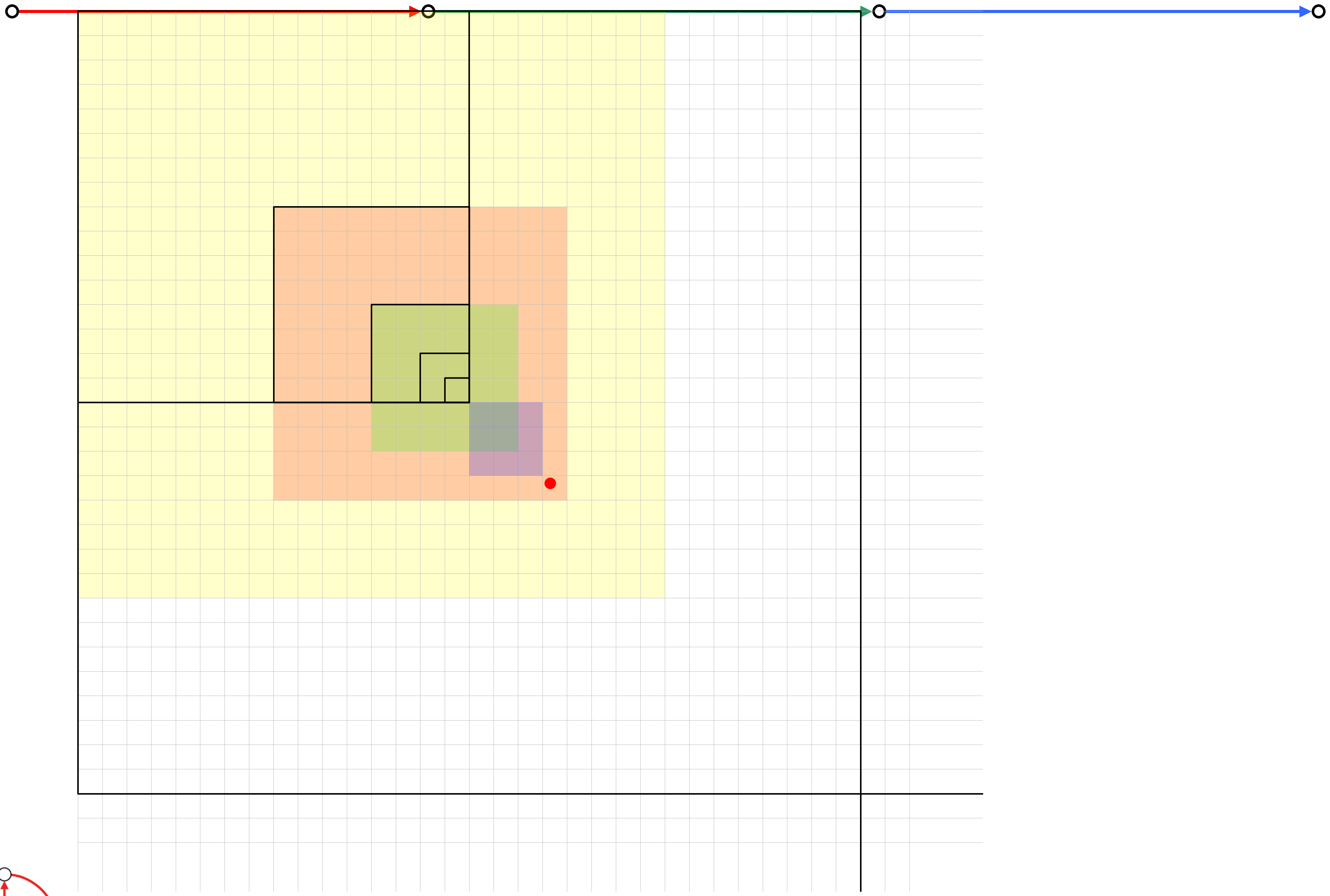
LLS Worst Case



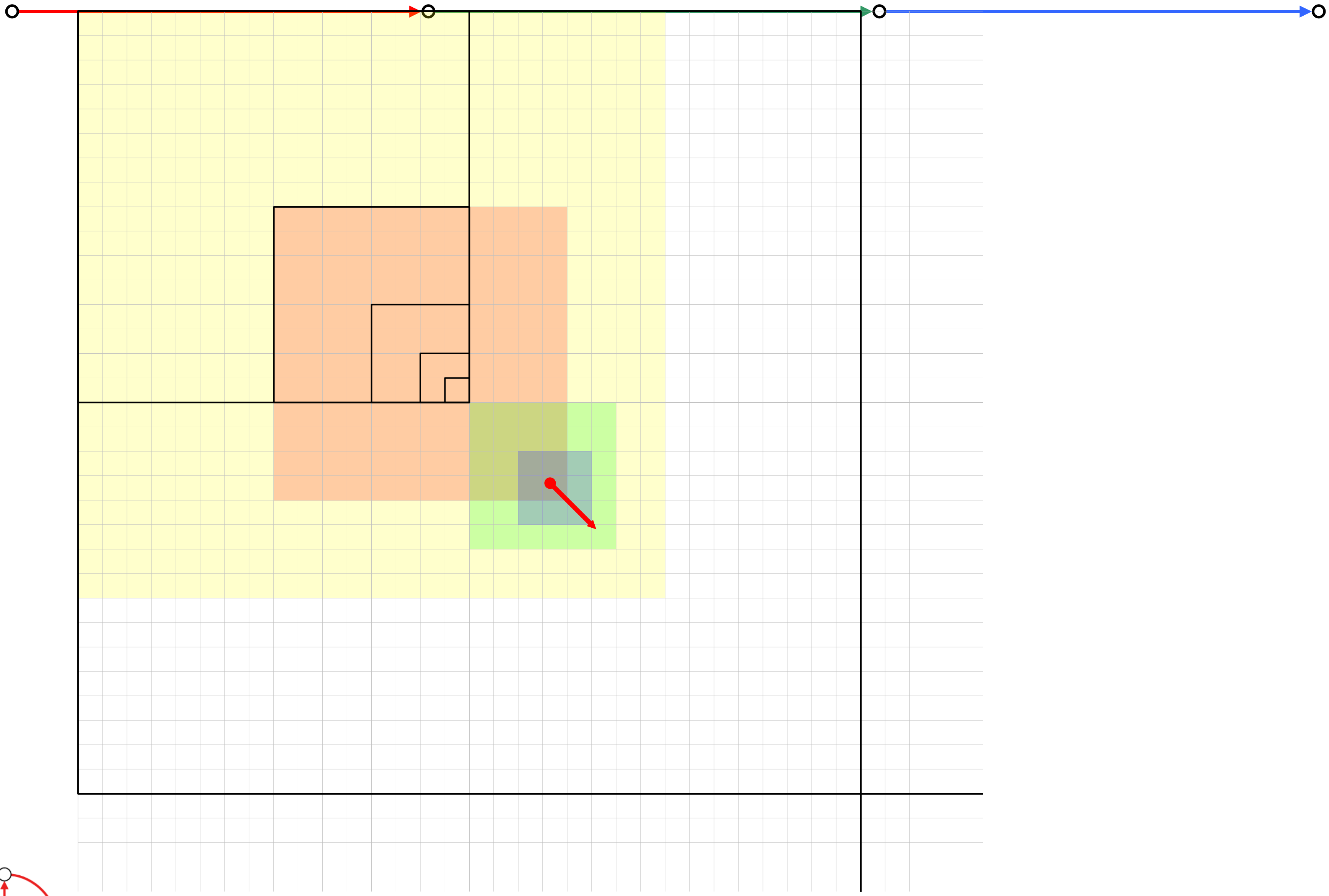
LLS Worst Case



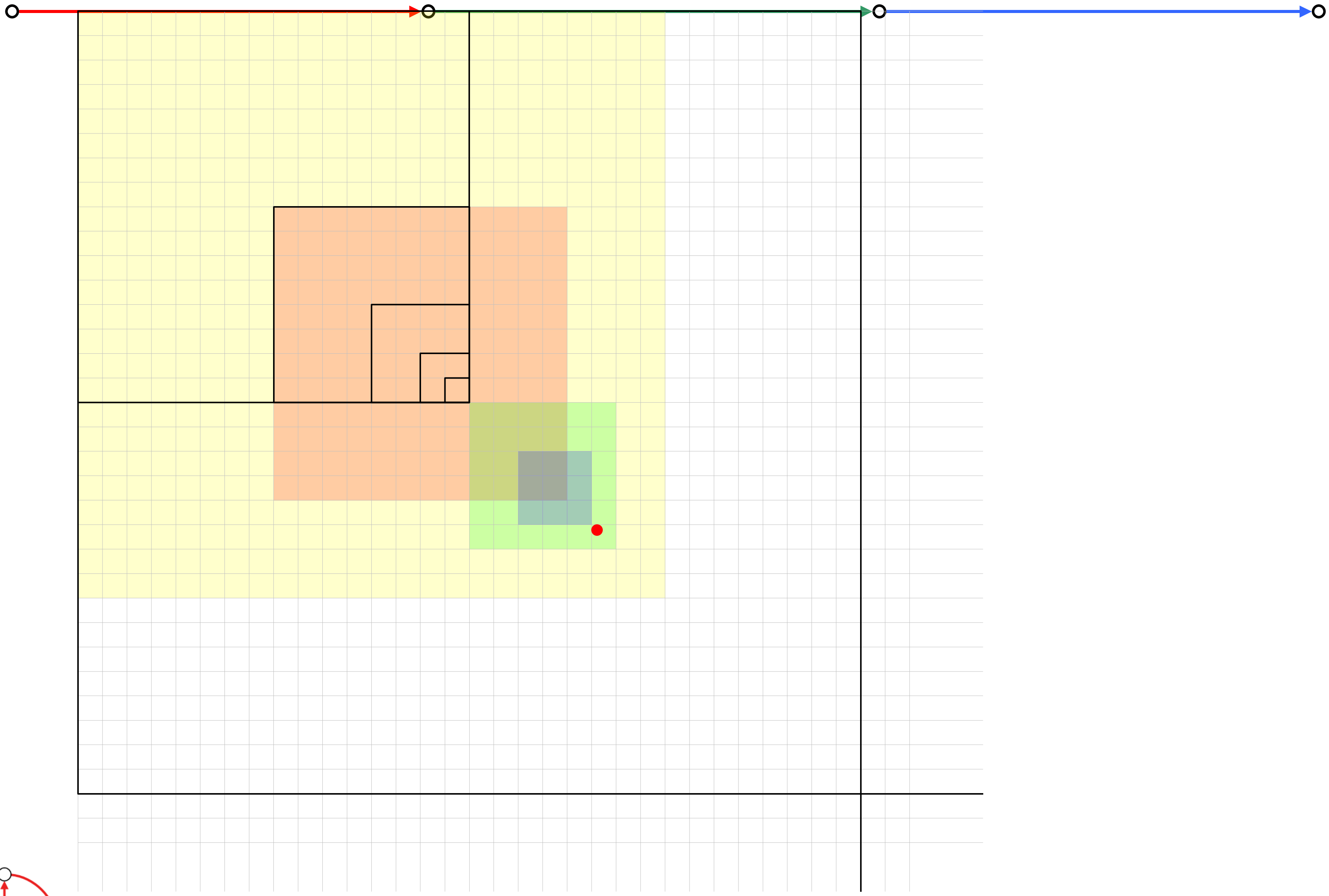
LLS Worst Case



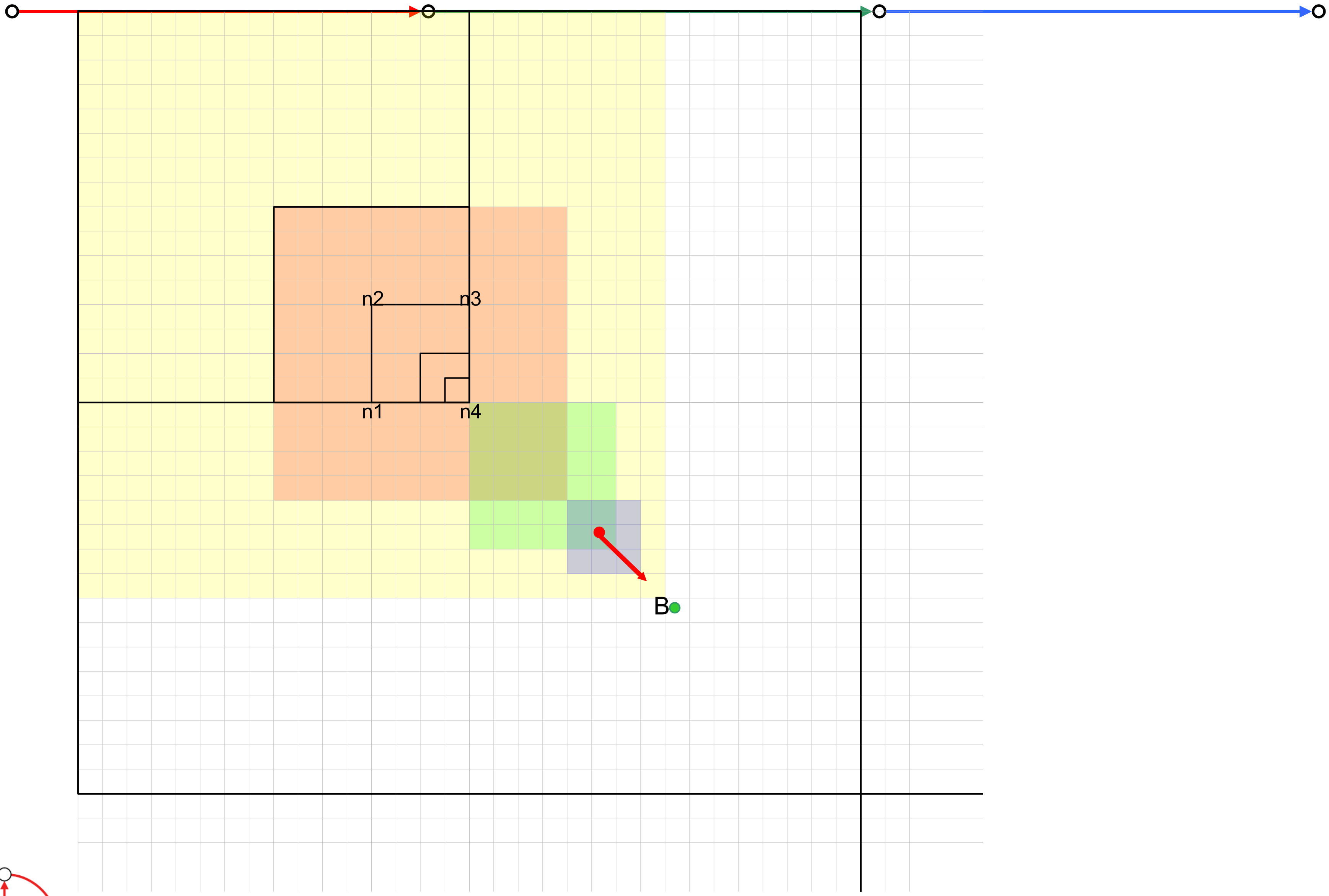
LLS Worst Case



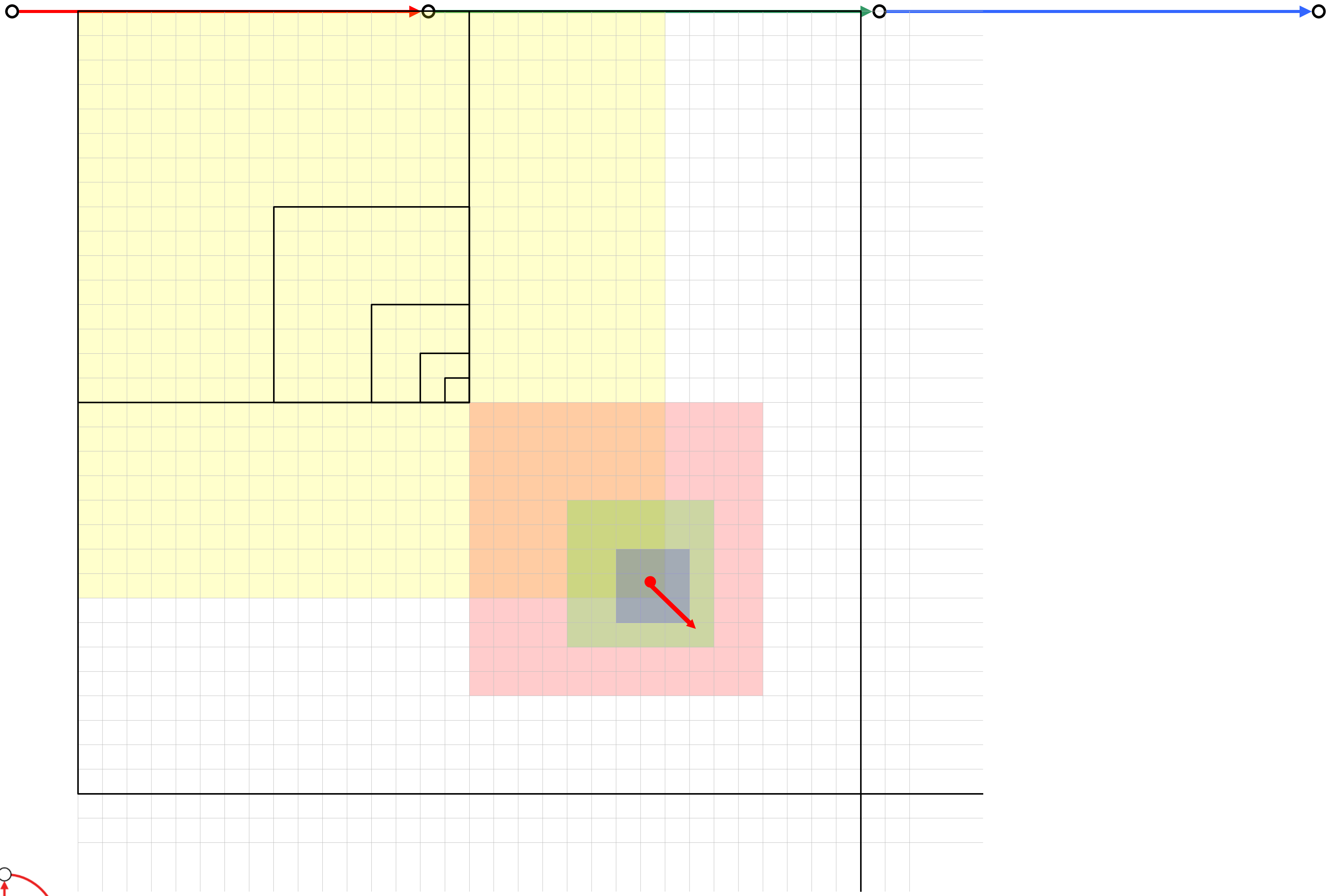
LLS Worst Case



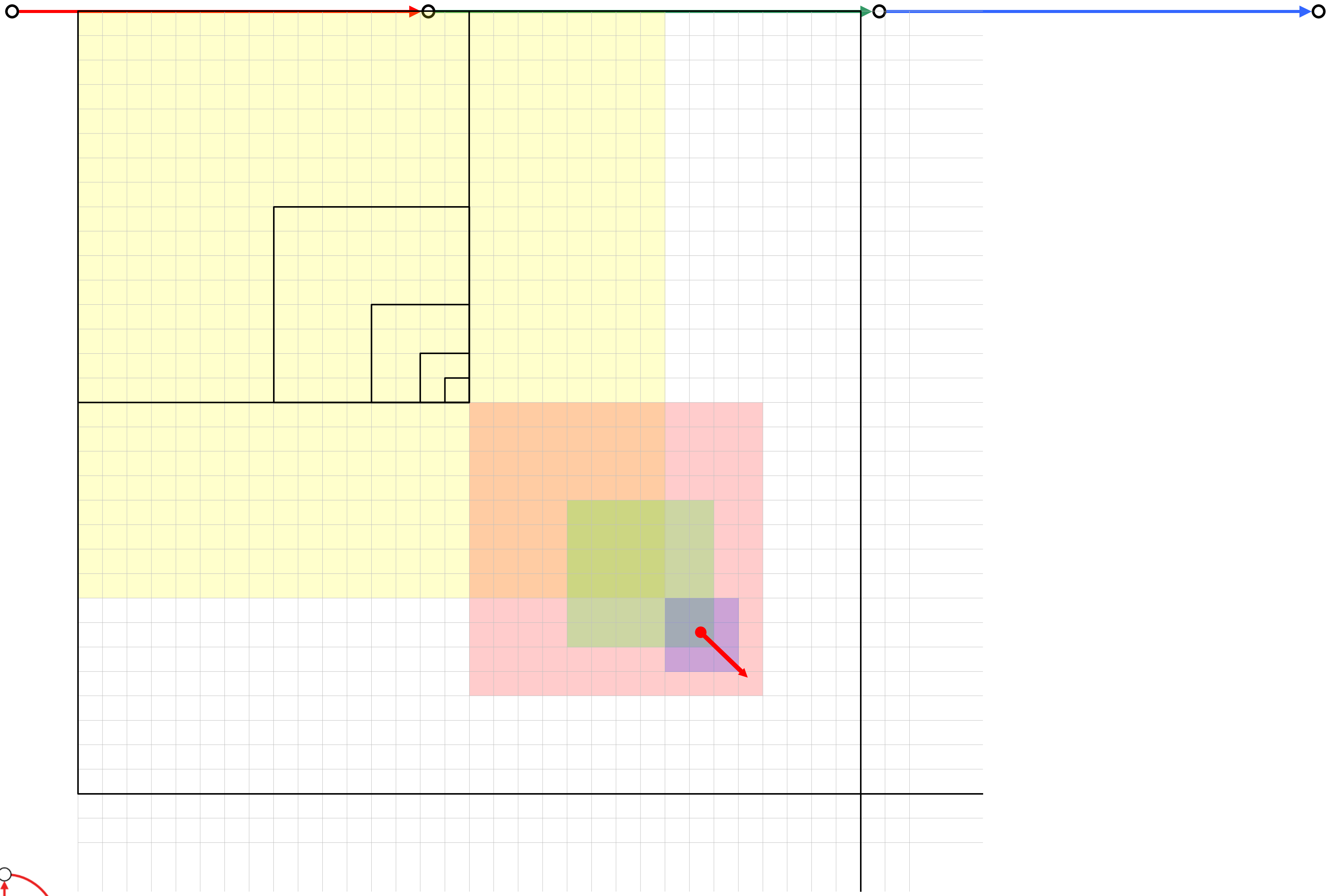
LLS Worst Case



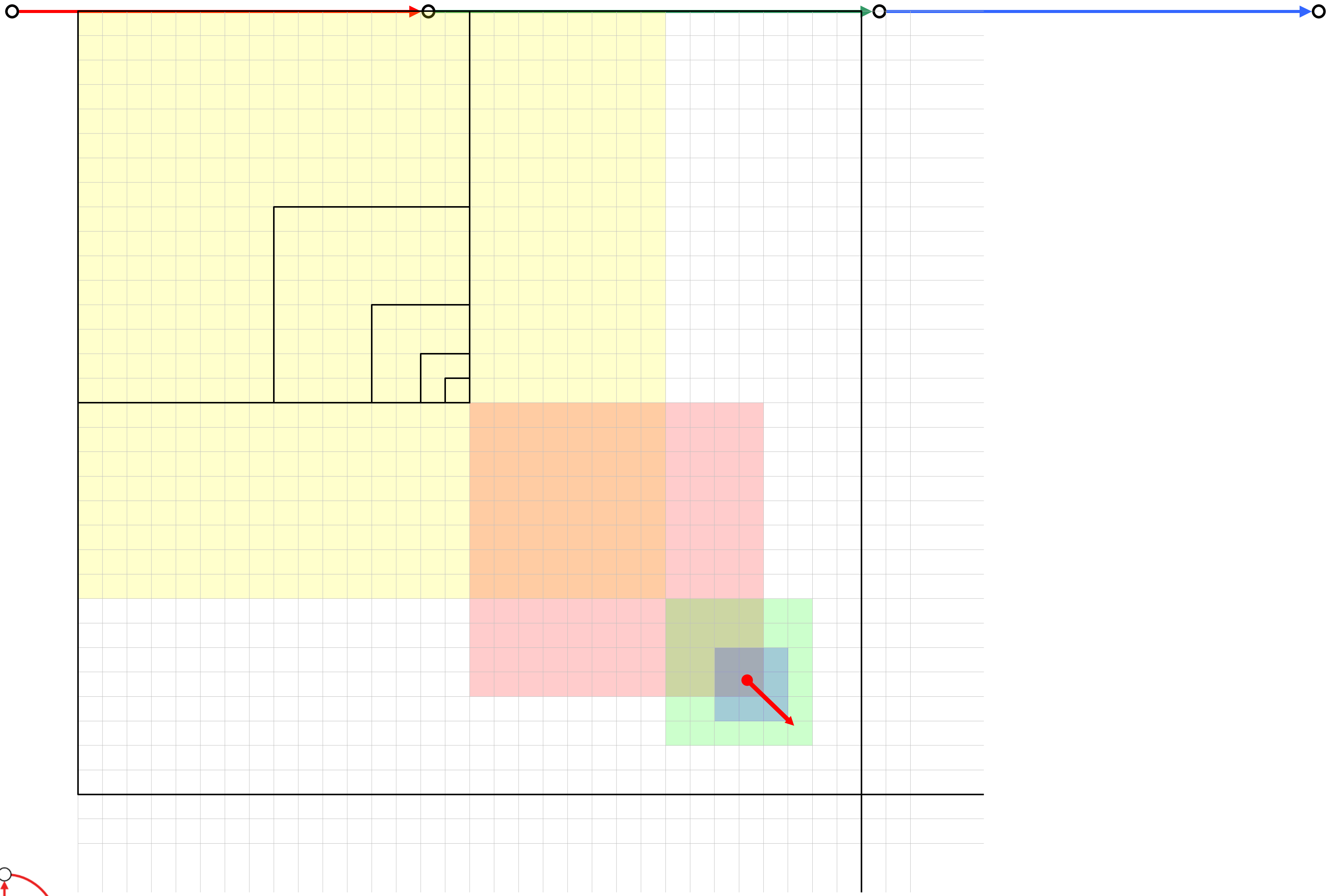
LLS Worst Case



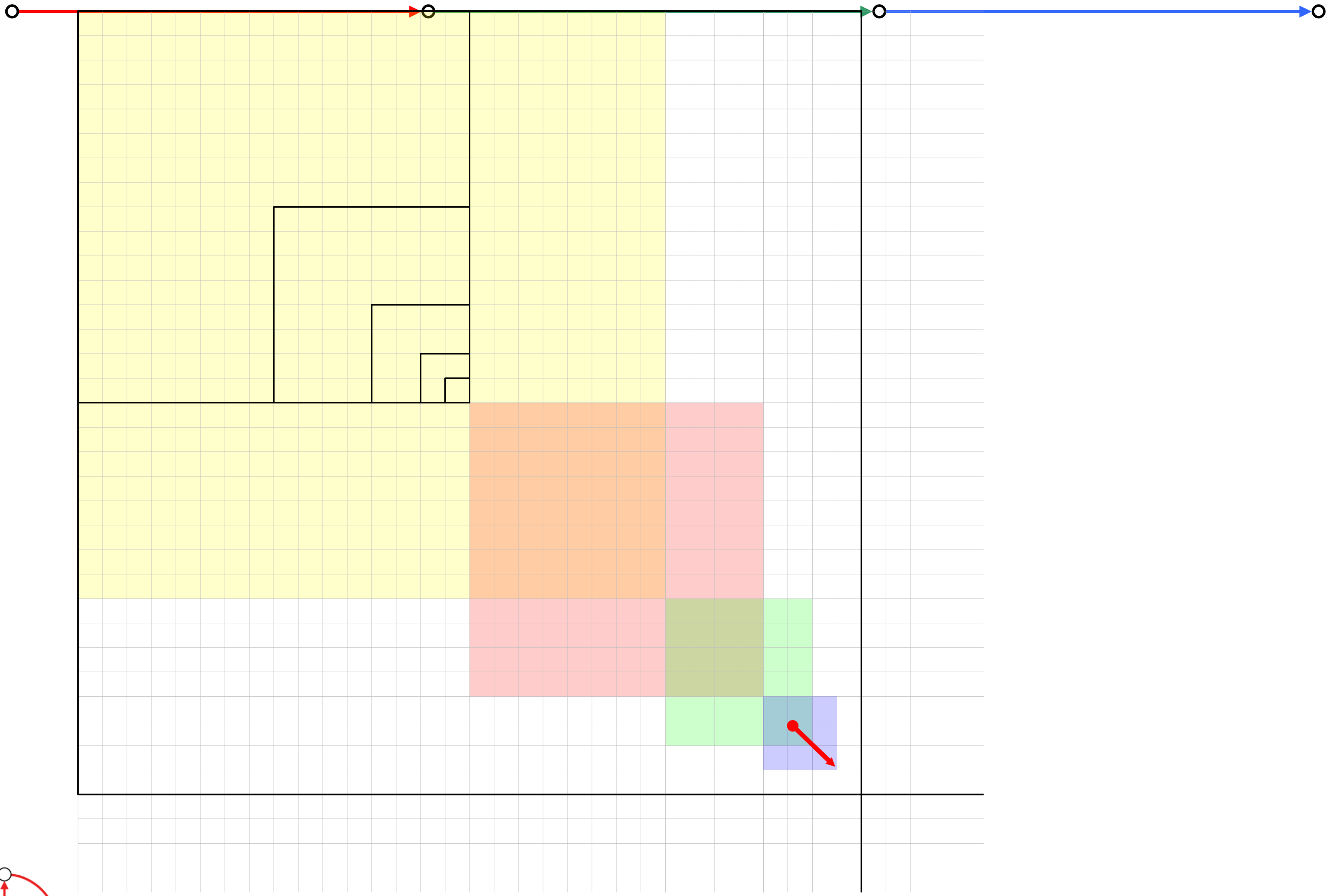
LLS Worst Case



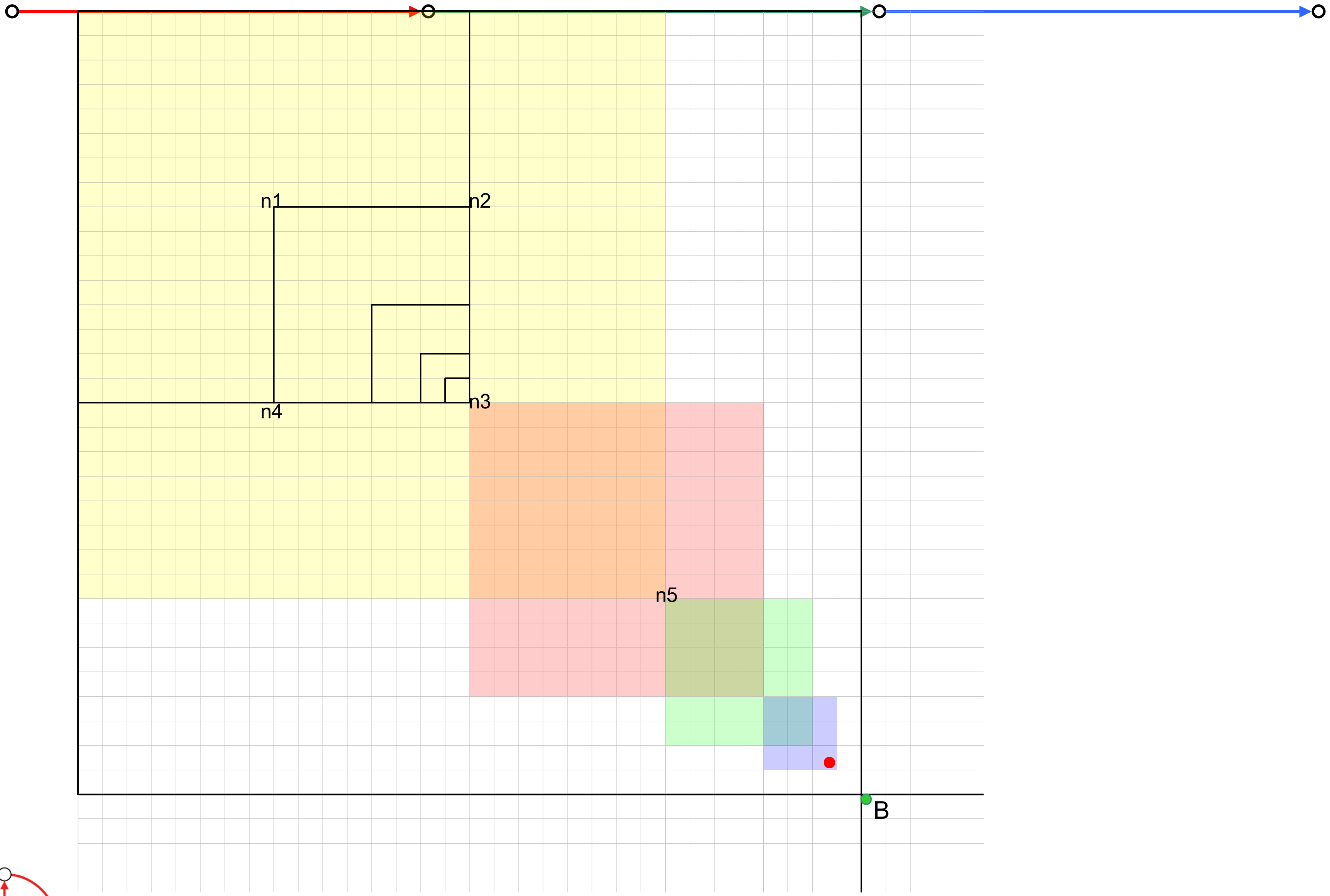
LLS Worst Case

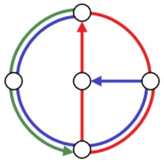


LLS Worst Case

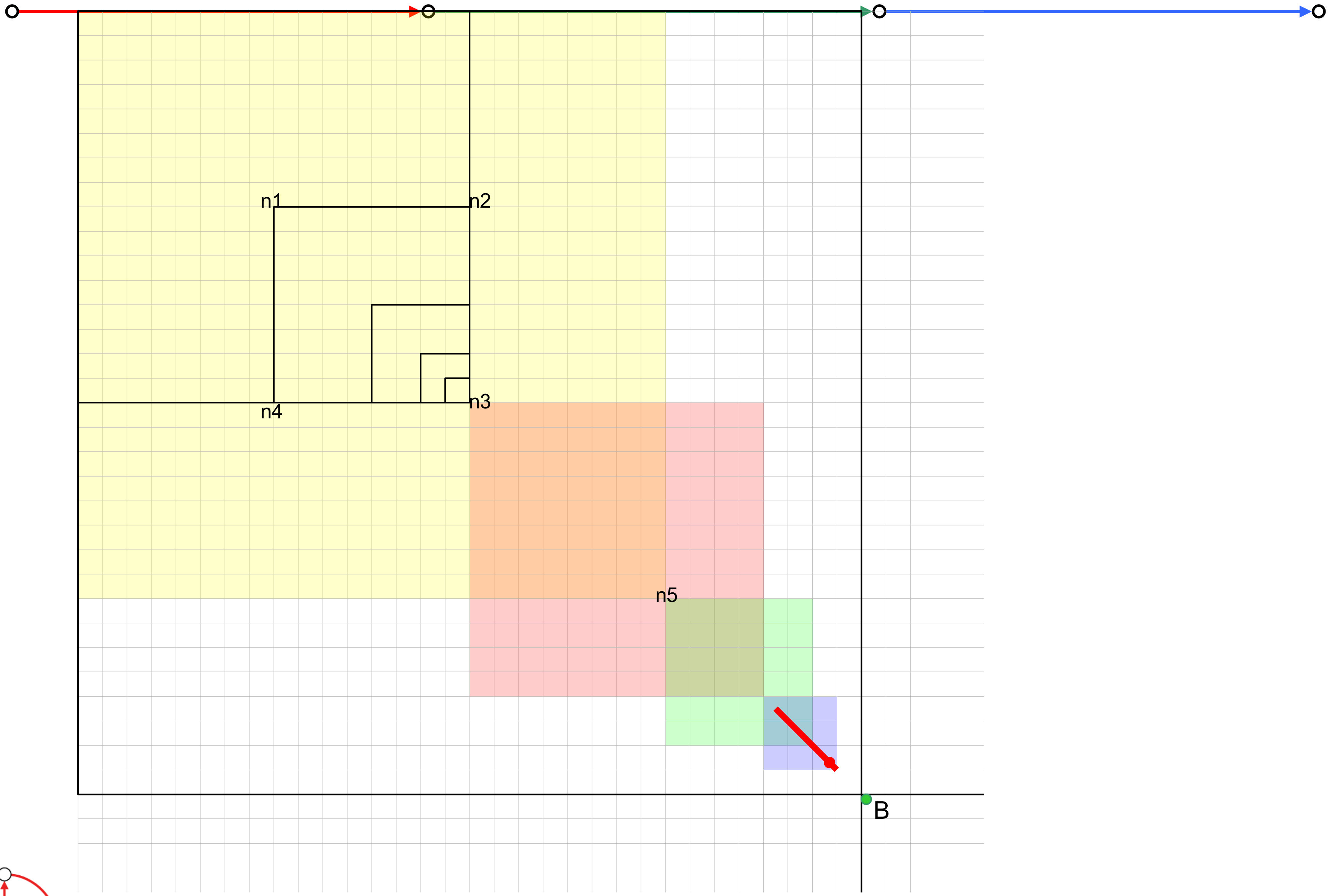


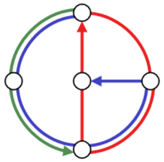
LLS Worst Case

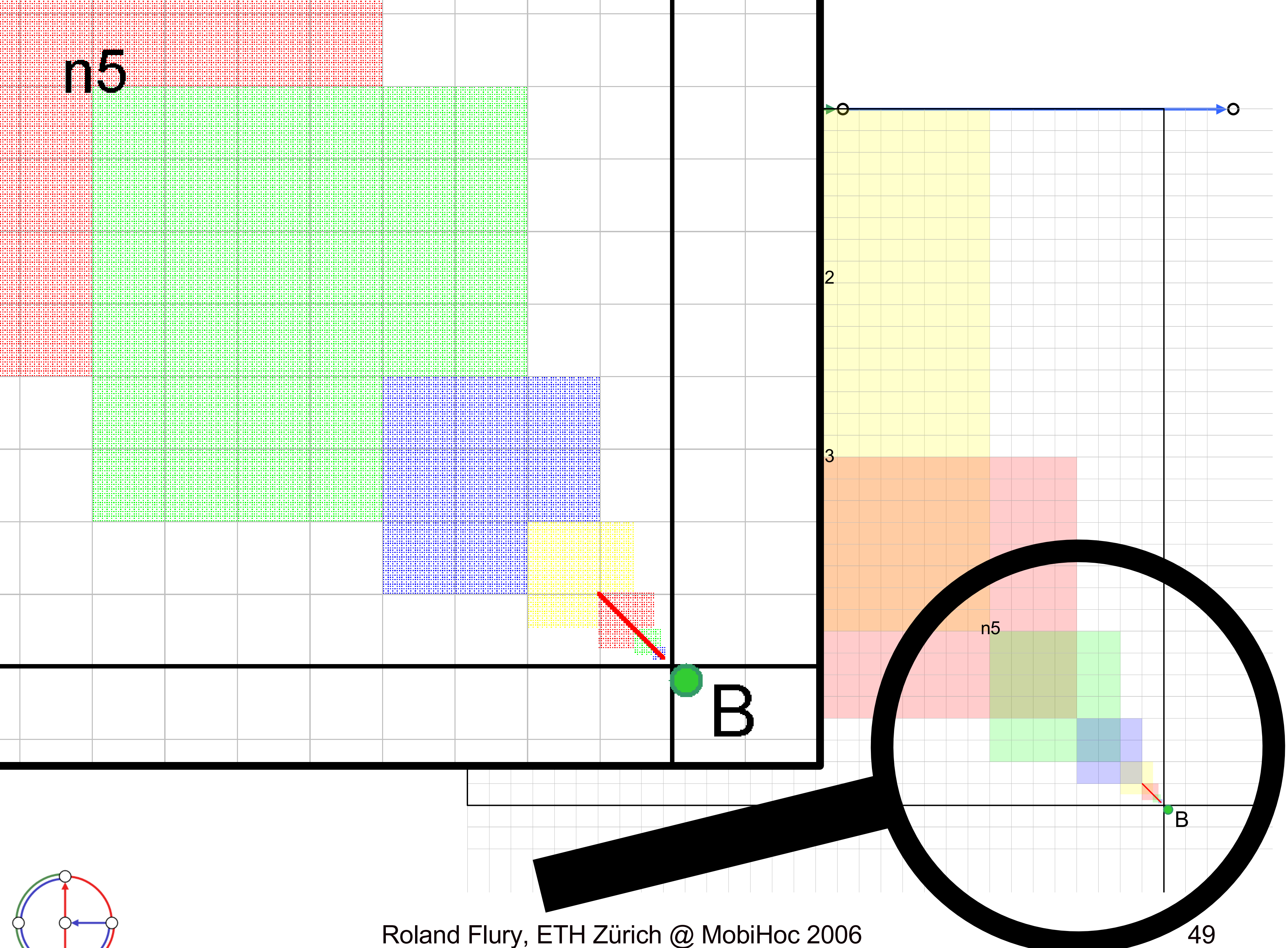




LLS Worst Case






n5
B
2
3
n5
B