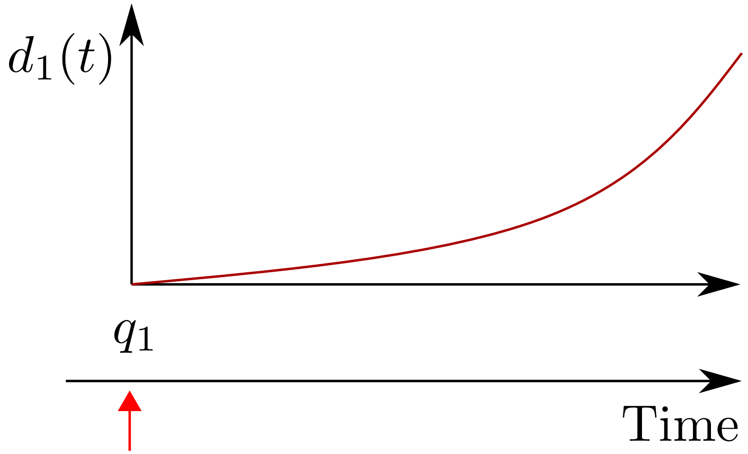
Nearly-Optimal Algorithm for Non-Clairvoyant Service with Delay

Noam Touitou

Online Service with Delay (OSD)

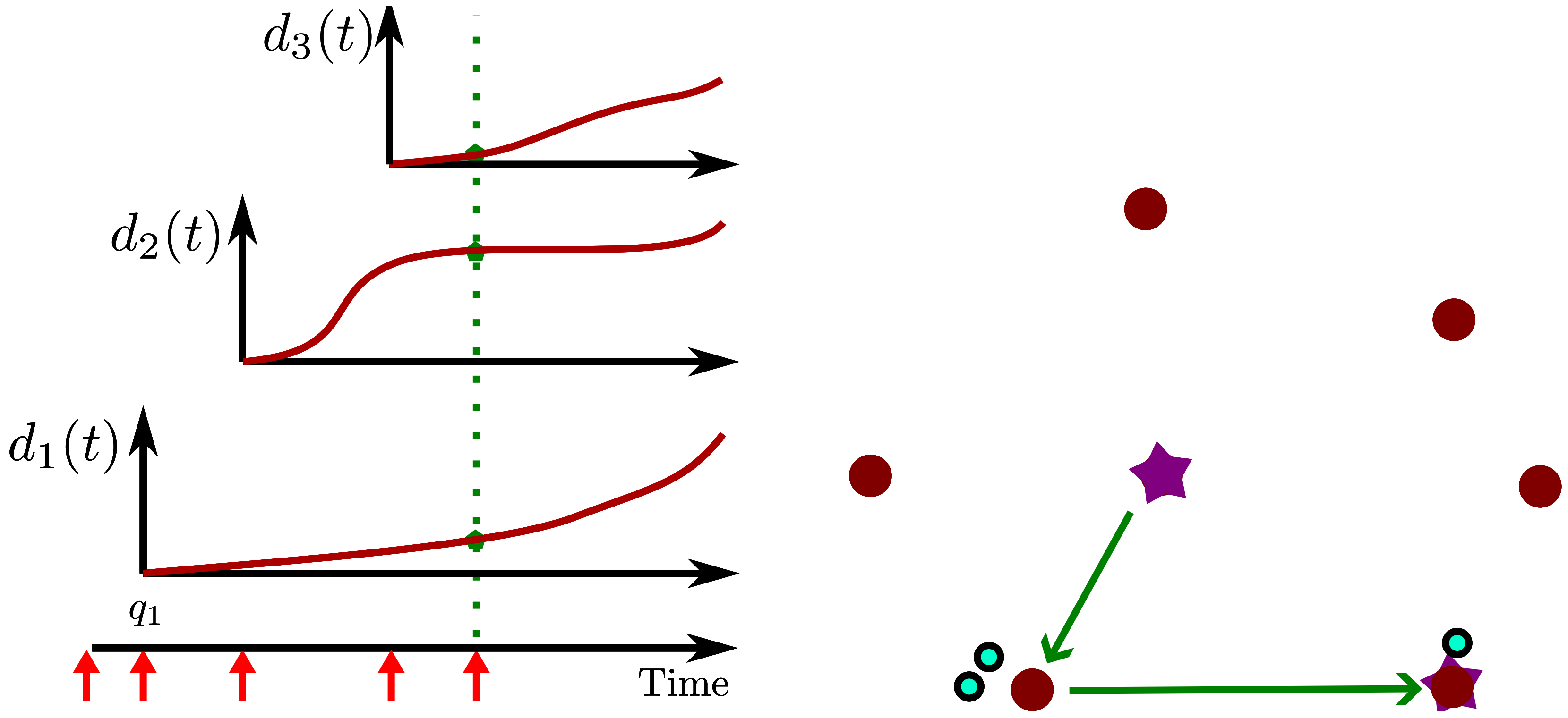
- Requests arrive on a metric space, over time.

- Requests accumulate **delay cost** while pending
  - Delay described as nondecreasing function.



- The algorithm has a single server, which serves requests.
  - Serving request = traversing it with the server.
  - Server movements are instantaneous.

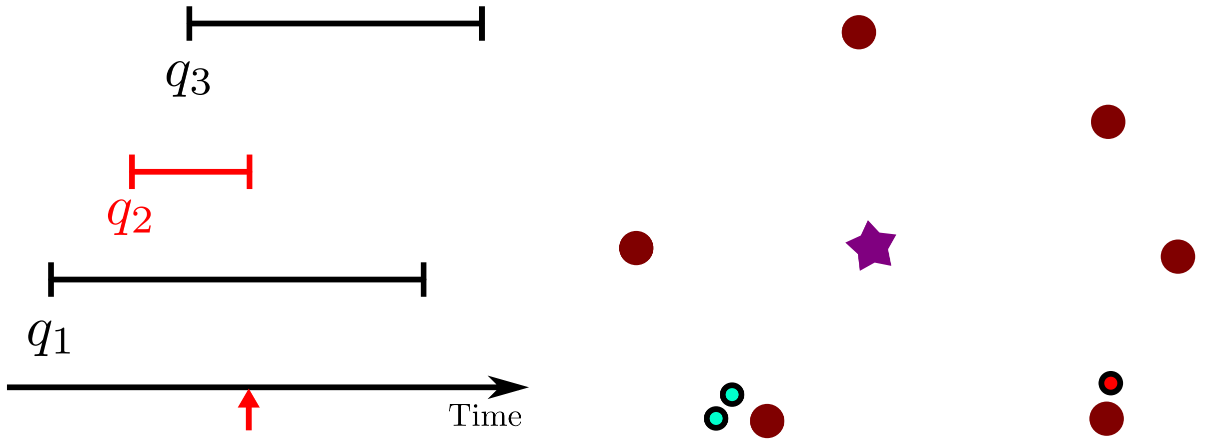
- Goal: Minimize movement cost + delay cost.



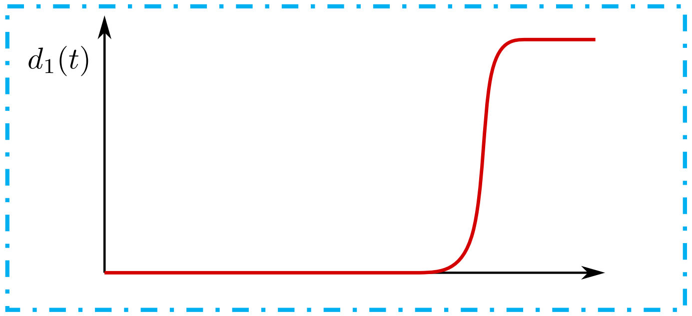
- Total cost: movement cost (green edges) plus delay cost (green plot points).

Online Service with Deadlines

- In service with **deadlines**, the delay functions are replaced with deadline times.



- This is a special case of delay!

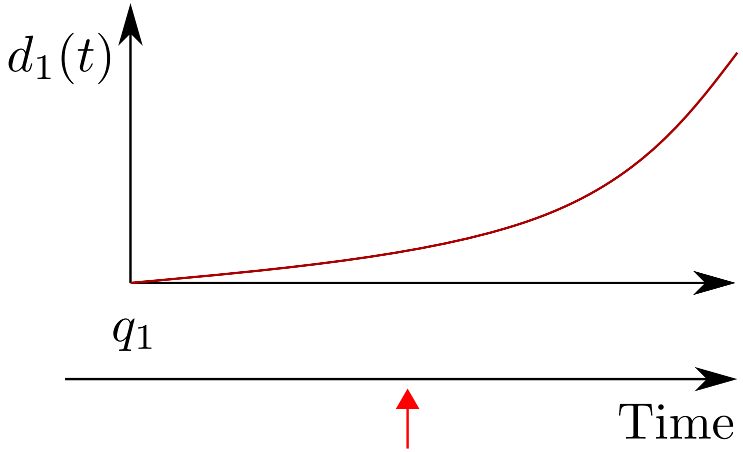


Clairvoyance

- A crucial property of the model is whether the algorithm knows **future delay of current requests.**

- If the answer is yes, we're in the **clairvoyant model.**

- Otherwise, the model is **non-clairvoyant**.
  - We'll focus on the non-clairvoyant model in this talk.



Previous Work

- Let:
  - n be the number of nodes in the metric space.
  - m be the number of requests in the input.
- In the clairvoyant model, there exists a line of work yielding polylogarithmic competitiveness:
  - An $O(\log^4 n)$-competitive randomized algorithm. [Azar-Ganesh-Ge-Panigrahi, STOC'17]
    - Where n is the number of points in the metric space.
  - An $O(\log^2 n)$-competitive randomized algorithm. [Azar-T, FOCS'19]
  - An $O(\log \min(n, m))$-competitive deterministic algorithm. [T, STOC'23]
- In the non-clairvoyant model, there exist $\Omega(\sqrt{n})$ and $\Omega(\sqrt{m})$ lower bounds. [AGGP, STOC '17]
  - In this talk, we introduce an upper bound that nearly matches both lower bounds.

Our Results

- We present the first algorithm for non-clairvoyant OSD.

- The algorithm is $O(\min(\sqrt{n}\log n, \sqrt{m}\log m))$ competitive.
  - This upper bound applies to the deadline special case as well.
  - We'll focus on $\mathrm{O}(\sqrt{n}\log n)$ in this talk.

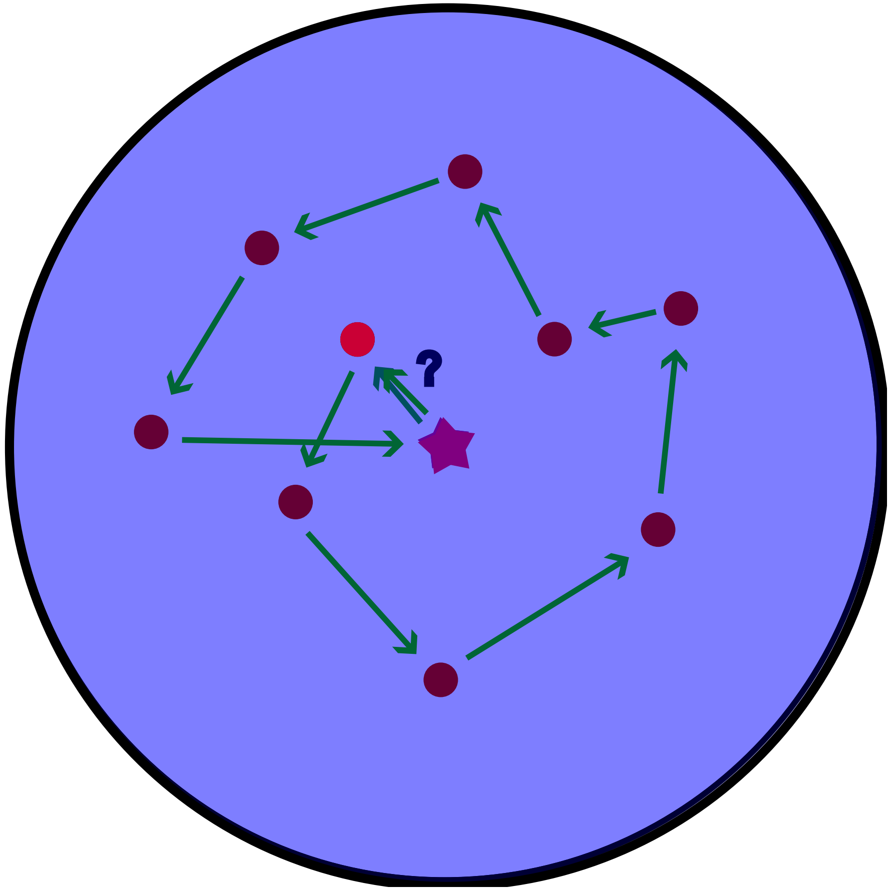
- This upper bound matches the $\Omega(\sqrt{n})$ and $\Omega(\sqrt{m})$ lower bounds up to a logarithmic factor.

The Algorithm

Service Structure

- The algorithm consists of *services*, which are instantaneous movements of the server.
- The algorithm waits until some set of requests accumulates sufficient delay cost, triggering a service:

1. The algorithm chooses a radius R in which to move the server.
2. The algorithm serves some pending requests in the ball, then returns to its initial position.
3. Sometimes, the server rests at some location within the ball.



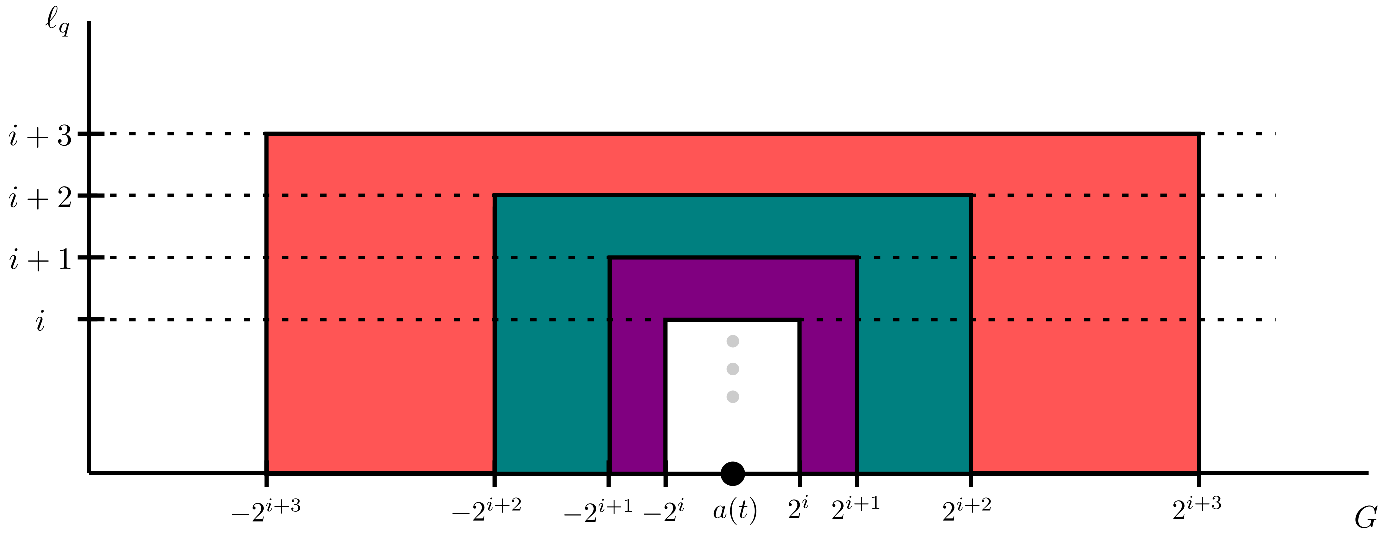
Delay Counters and Residual Delays

- For every location v, and for every integer ℓ, we maintain residual delay counters $g_{v,\ell}$.
- Every pending request q has level ℓ_q that increases over time (initially $-\infty$).
- The residual delay counter $g_{v,\ell}$ grows with the delay of level-ℓ requests on v.
- The counters can also be decreased by the algorithm (can be seen as “paying off” delay).

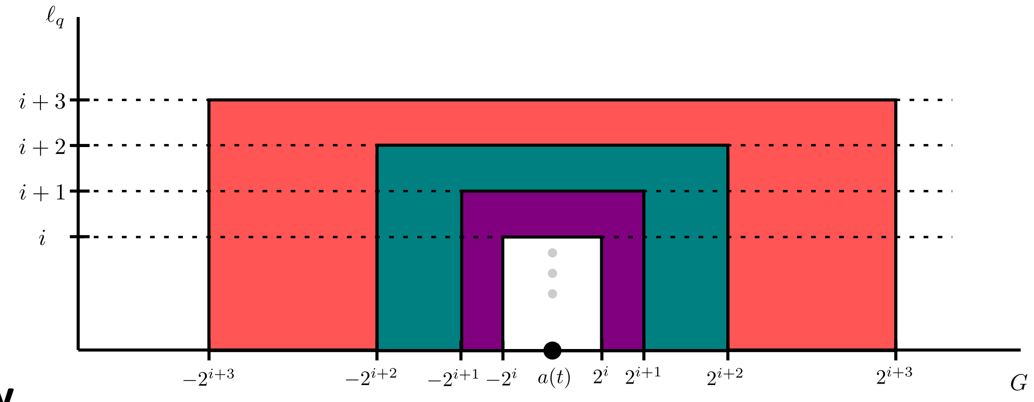
Domes

- For every ℓ, we consider the total residual delay of a **dome** around the server. That is,
  - The sum of positive residual delay counters $g_{v,\ell}$ for v at distance at most $2^{\ell-1}$ from the server, **plus**
  - The sum of positive $g_{v,\ell'}$ for $\ell' \leq \ell$ and v at distance between $2^{\ell-1}$ and 2^{ℓ} from the server.

- When the total residual delay of some dome ℓ exceeds 2^{ℓ}, a service of level $\ell + 4$ is started.



Services



- A service λ of level ℓ_λ serves requests in a $R \coloneqq 2^{\ell_\lambda}$-radius ball.
- There are three types of services: **primary, secondary** and **tertiary.**
- All services start by paying off any positive $g_{v,\ell}$ to zero, for v in the R-radius ball and $\ell \leq \ell_\lambda$.
- Primary/secondary services serve many requests, while tertiary services greedily serve a single request.
  - Primary/Secondary cost = $O(\sqrt{n} \cdot R)$
  - Tertiary cost = $O(R)$
- We first describe primary and secondary services.

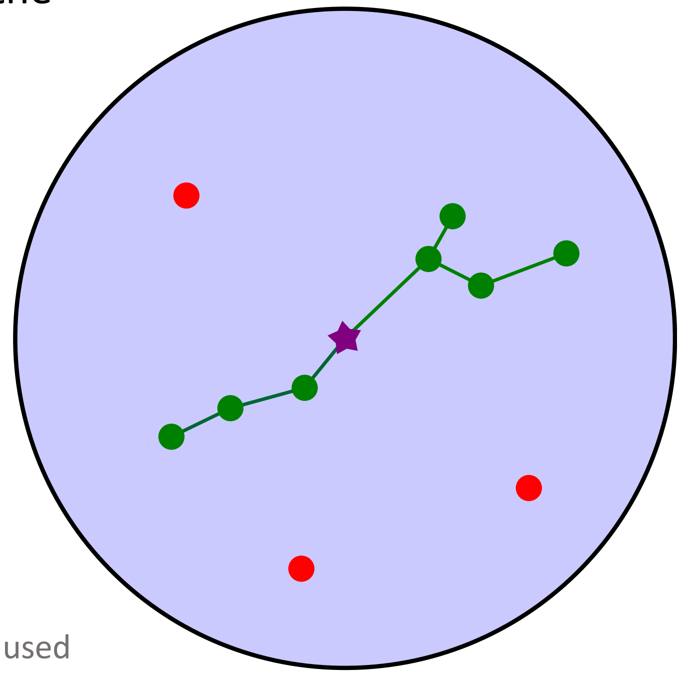
Primary+Secondary Services

- A primary/secondary service λ considers all locations V_λ of pending requests inside the $R = 2^{\ell_\lambda}$ radius ball.

- Intuitively:
  - Serve cost-effective, **dense** locations.
  - Pay off future delay in **sparse** locations.

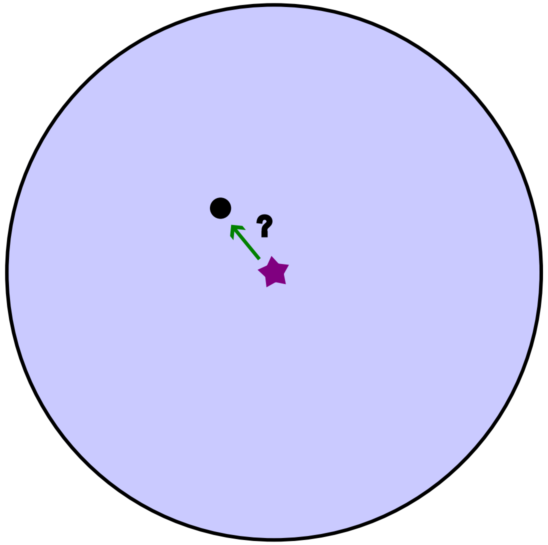
- It traverses a subset $V'_\lambda \subseteq V_\lambda$, where:
  - The average traversal cost is $\mathrm{O}\left(\frac{R}{\sqrt{n}}\right)$ per location in V'_λ.
  - Every subset of $V_\lambda \setminus V'_\lambda$ would cost an average of $> \frac{R}{\sqrt{n}}$ per location to traverse.
  - (The procedure for obtaining V'_λ uses a prize-collecting algorithm for Steiner tree, and is similar to that used for network design problems in [T, ICALP'23].)

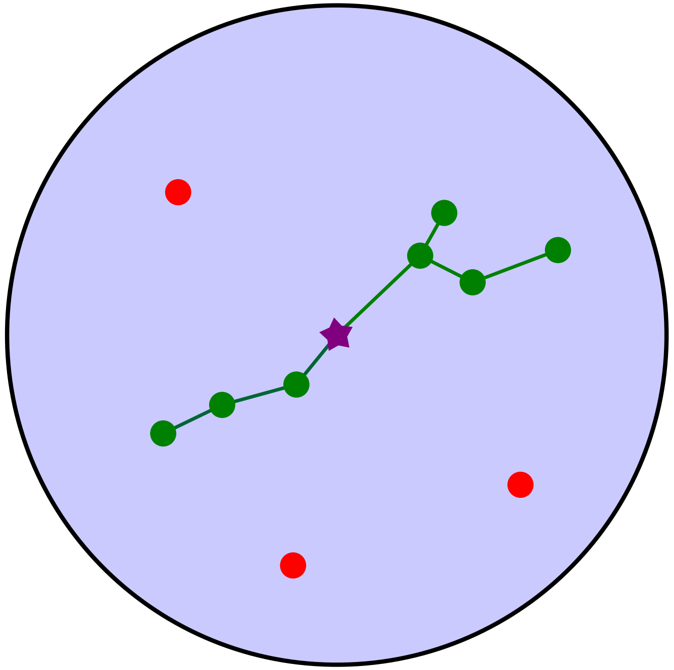
- For each $\mathrm{v} \in V_\lambda \setminus V'_\lambda$ the service then decrements g_{v,ℓ_λ} by $\frac{R}{\sqrt{n}}$.



Primary+Secondary Services

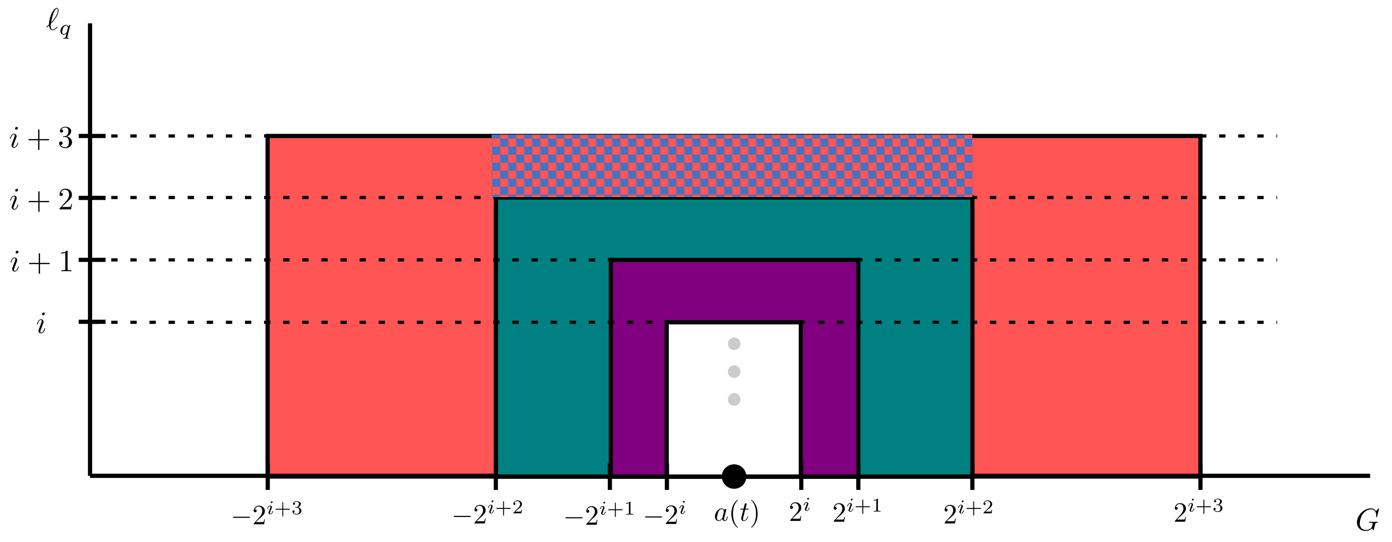
- Additionally, Unserved pending requests of level $\leq \ell_\lambda$ in the ball are **upgraded** to ℓ_λ.
  - This also marks the requests as “witnesses” to λ.
- If delay is concentrated in a small-radius ball, the server might move to the center of that ball. (This only happens in primary services.)





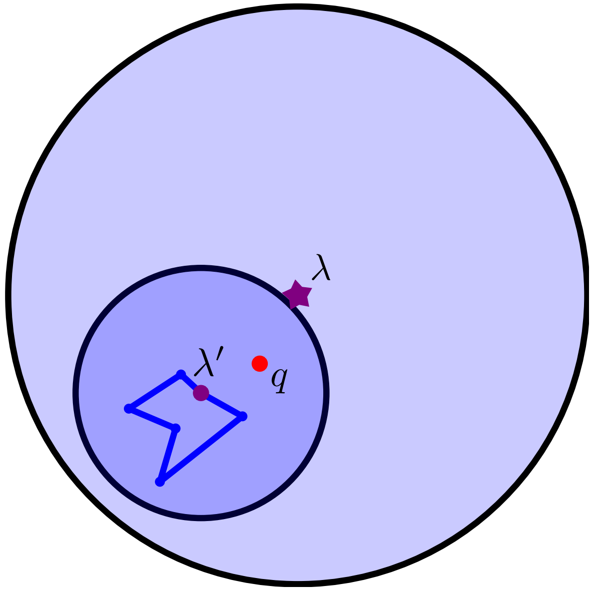
Primary+Secondary Services

- When the service λ is triggered by dome $\ell_\lambda - 4$, are there requests of distance class strictly smaller than $\ell_\lambda - 4$ with positive delay? (That is, a request in the "upper dome".)
  - **"No"** → λ is primary.
  - "**Yes"** → λ is secondary (or tertiary).



Charging intuition

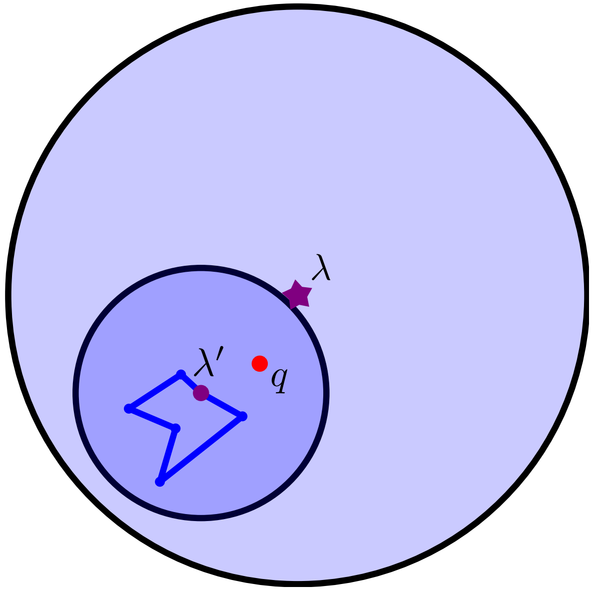
- Before describing tertiary services, we describe the intuition for such services.
- Intuitively, primary services are triggered by delay cost for which the optimal solution cannot prepare; their cost can thus charged to the optimal solution.
- Secondary services are trickier. Note that λ is triggered by a request q in the "upper dome", whose level was last raised by a prior service λ' where $\ell_{\lambda'} = \ell_\lambda - 4$.
- We *want* to claim that if q gathered delay to trigger λ, then the costs of λ' are justified, i.e., were also incurred by the optimal solution.
  - I.e., we want a *doubling argument.*



From charging intuition to tertiary services

- If we were in the **clairvoyant** case, the argument would be easier:
  - λ' could prioritize according to future delay.
  - If q wasn't served but incurred delay to trigger λ, this means that more urgent requests that were served by λ':
    1. Gathered large delay in the optimal solution, **or**
    2. Were served by the optimal solution, incurring significant cost.
  - This justifies the cost of λ', allowing us to charge the cost of λ to λ'.
- Instead, we are in the **non-clairvoyant** case, and thus serve "blindly".

- This is solved by **tertiary services.**



Tertiary Services

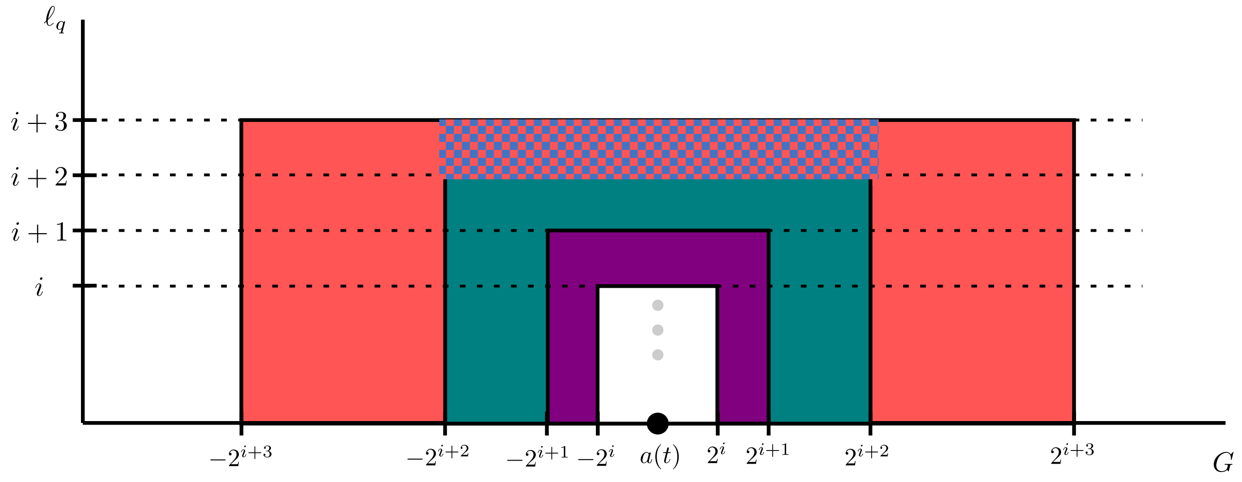
- When the service λ is triggered by dome $\ell_\lambda - 4$, are there requests of distance class strictly smaller than $\ell_\lambda - 4$ with positive residual delay? (That is, a request in the "upper dome".)

  1. **"No"** → λ is primary.

  2. "**Yes"** → there exists such q which is a witness for λ'.

  2.1. If λ' already triggered $\Theta(\sqrt{n})$ tertiary services, then λ will be secondary.

  2.2. Otherwise, λ will be tertiary.



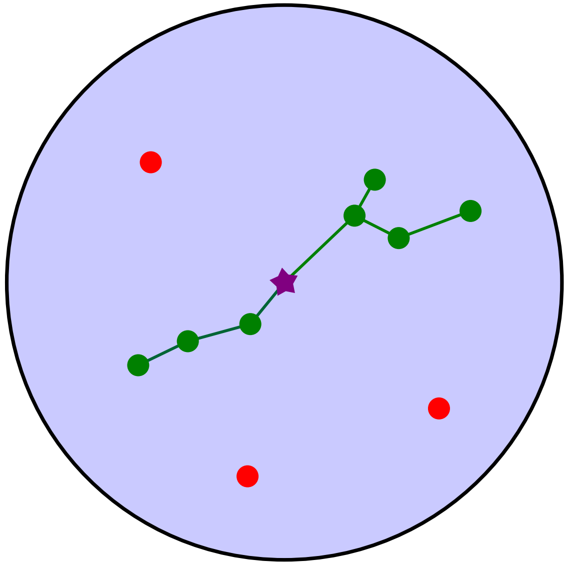
- A tertiary service simply serves the witness request and returns the server to the original location.

From charging intuition to tertiary services

- The doubling now looks like this:

  1. A primary/secondary service λ of level ℓ takes place, leaving "witness" requests.
  2. Then, $\Theta(\sqrt{n})$ tertiary services involve the witnesses of λ.
  3. A secondary service of level $\ell + 4$ takes place, charged to the $\Theta(\sqrt{n})$ tertiary services.

- Note that we are able to charge the tertiary services to OPT because of their sparse structure induced by the previous service λ.



Conclusion

- We present the first algorithm for non-clairvoyant online service with delay.

- The algorithm is deterministic, and has a competitive ratio of $O(\min(\sqrt{n}\log n, \sqrt{m}\log m))$.
    - Where n is the number of points, and m is the number of requests.

- This is nearly tight, matching the previously-known lower bounds of $\Omega(\sqrt{n})$ and $\Omega(\sqrt{m})$.

Thank You!