



Communication-Efficient Algorithms for Tracking Distributed Data Streams

Qin Zhang

Department of Computer Science and Engineering
Hong Kong University of Science & Technology

PhD thesis defense
May 18, 2010



Outline

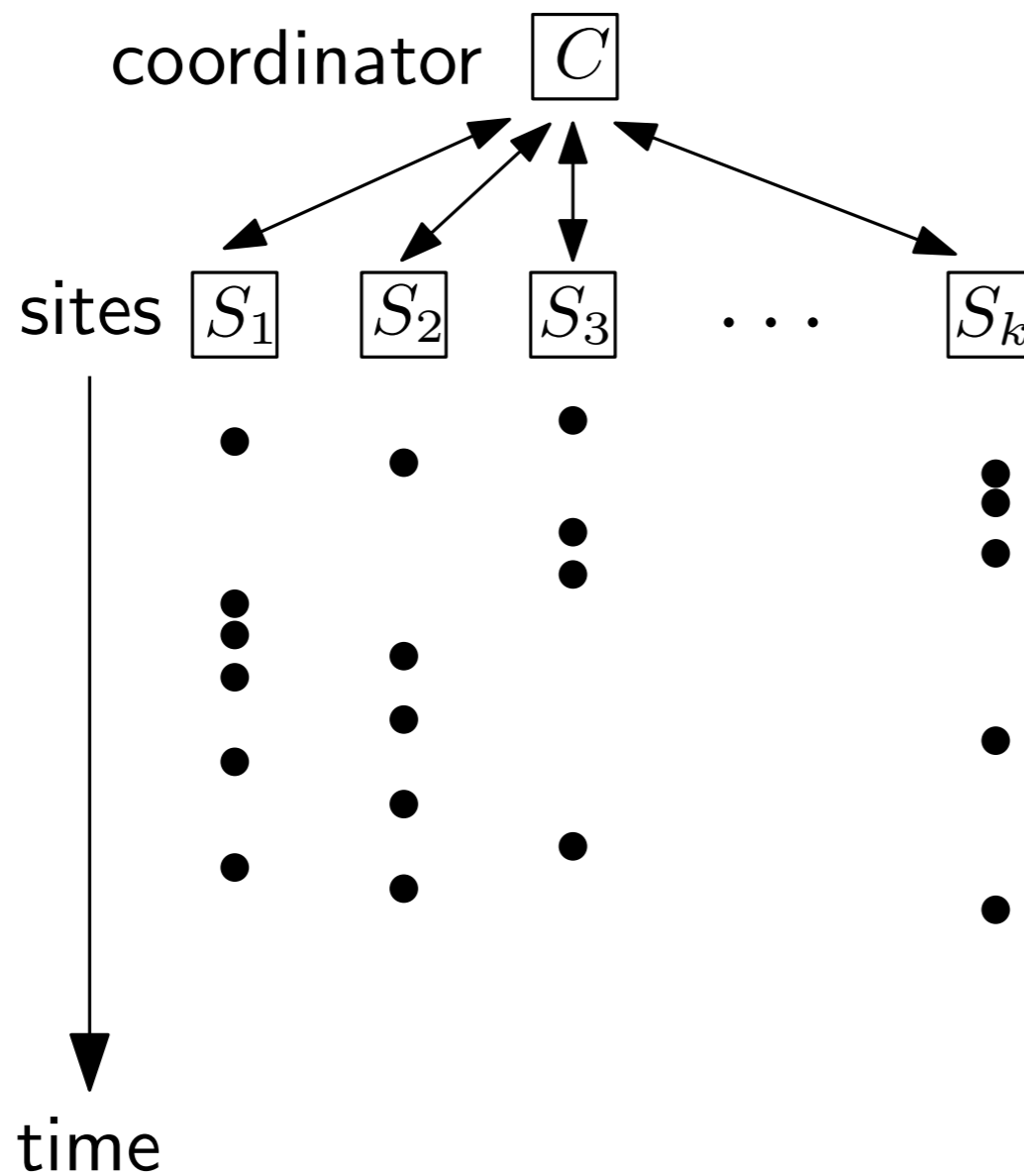
- ▣ Background and Previous Work
- ▣ Random Sampling
- ▣ Heavy hitters and Quantiles
- ▣ Arbitrary Function
- ▣ Future Work



Outline

- ▣ Background and Previous Work
- ▣ Random Sampling
- ▣ Heavy hitters and Quantiles
- ▣ Arbitrary Function
- ▣ Future Work

The Distributed Streaming Model

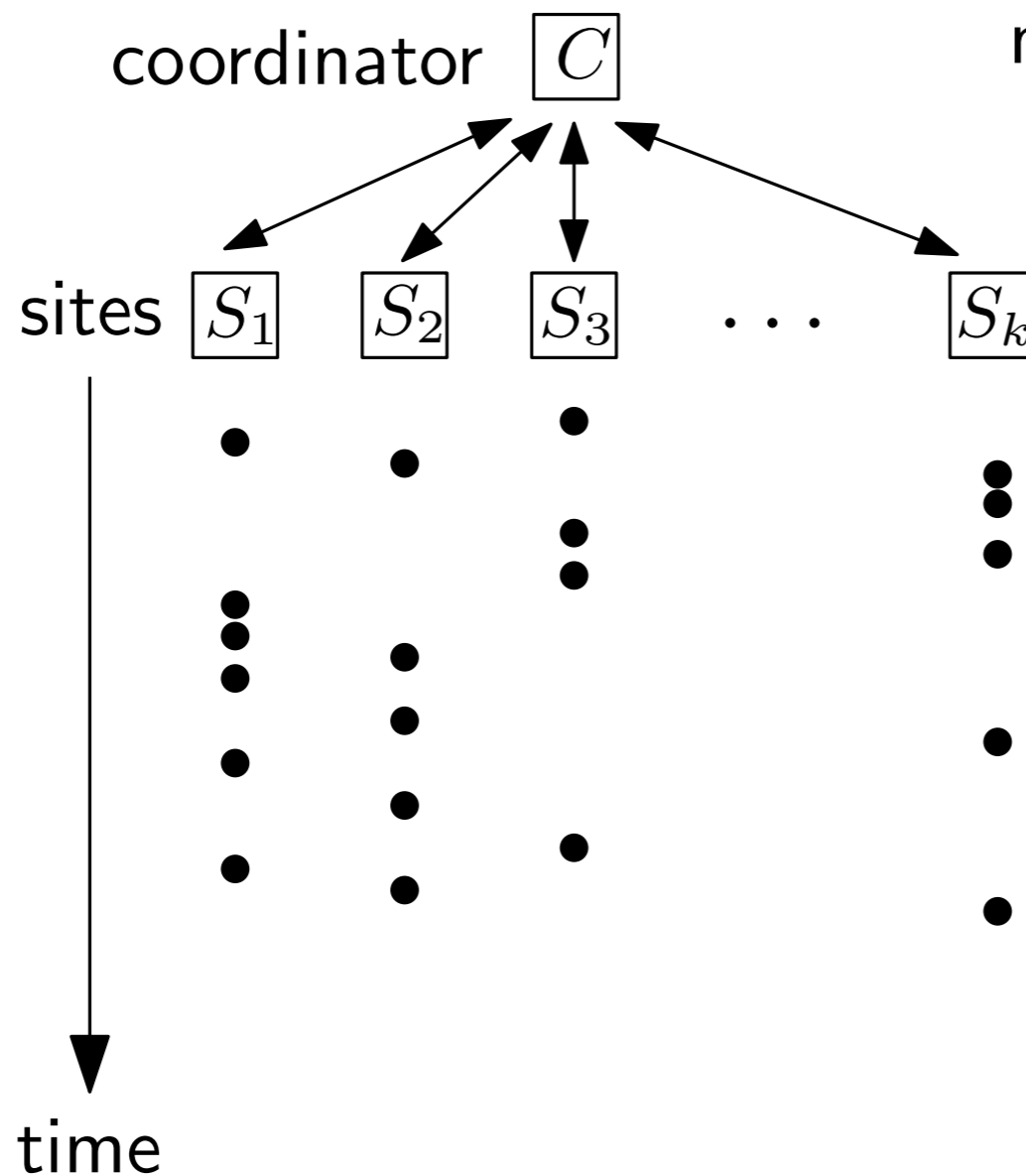


$A(t)$: set of elements received up to time t from all sites.
^a

^aAssume ≤ 1 item comes at each time unit.

The Distributed Streaming Model

The coordinator needs to maintain $f(A(t))$ for **all** t .

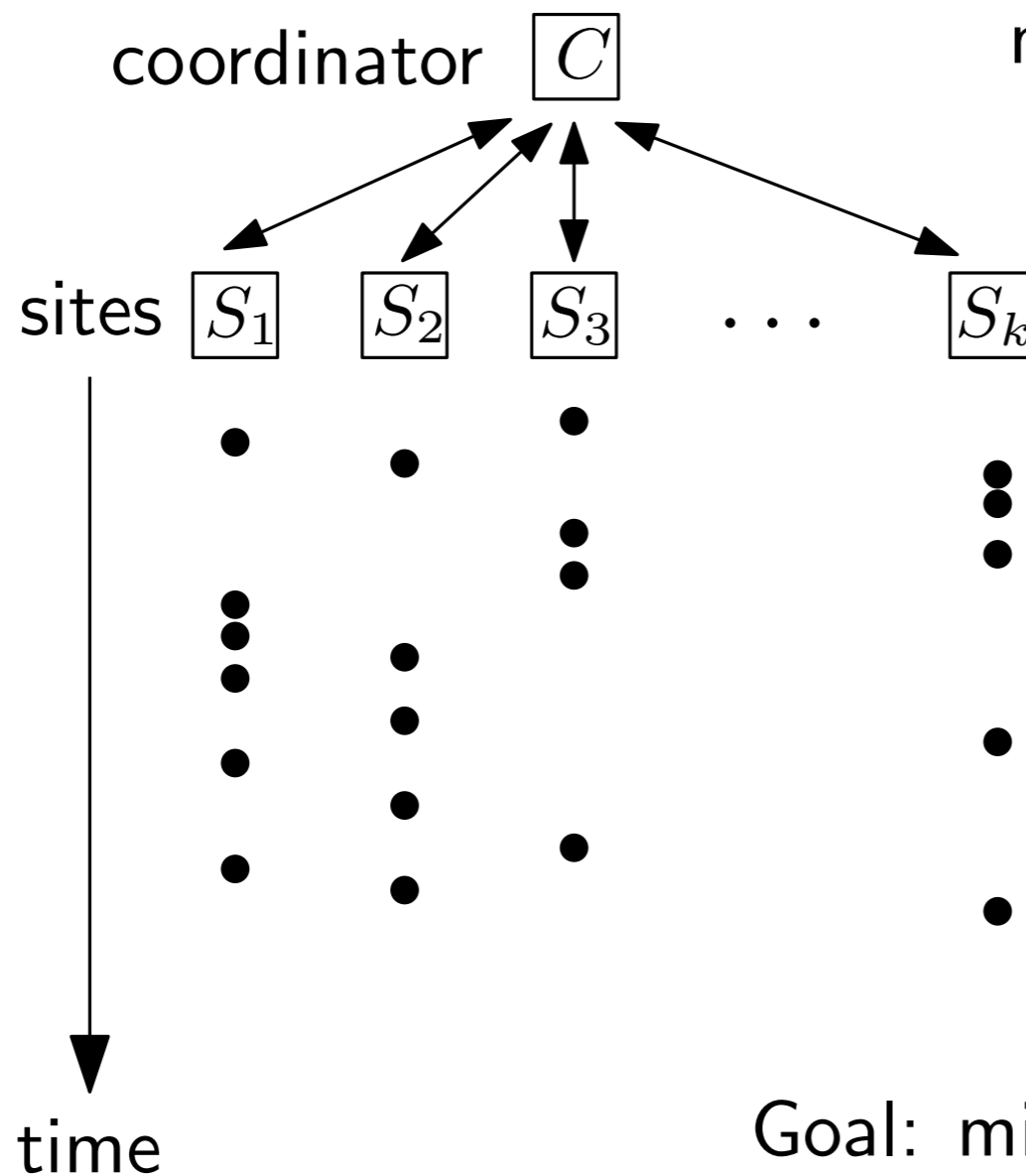


$A(t)$: set of elements received up to time t from all sites.
^a

^aAssume ≤ 1 item comes at each time unit.

The Distributed Streaming Model

The coordinator needs to maintain $f(A(t))$ for **all** t .

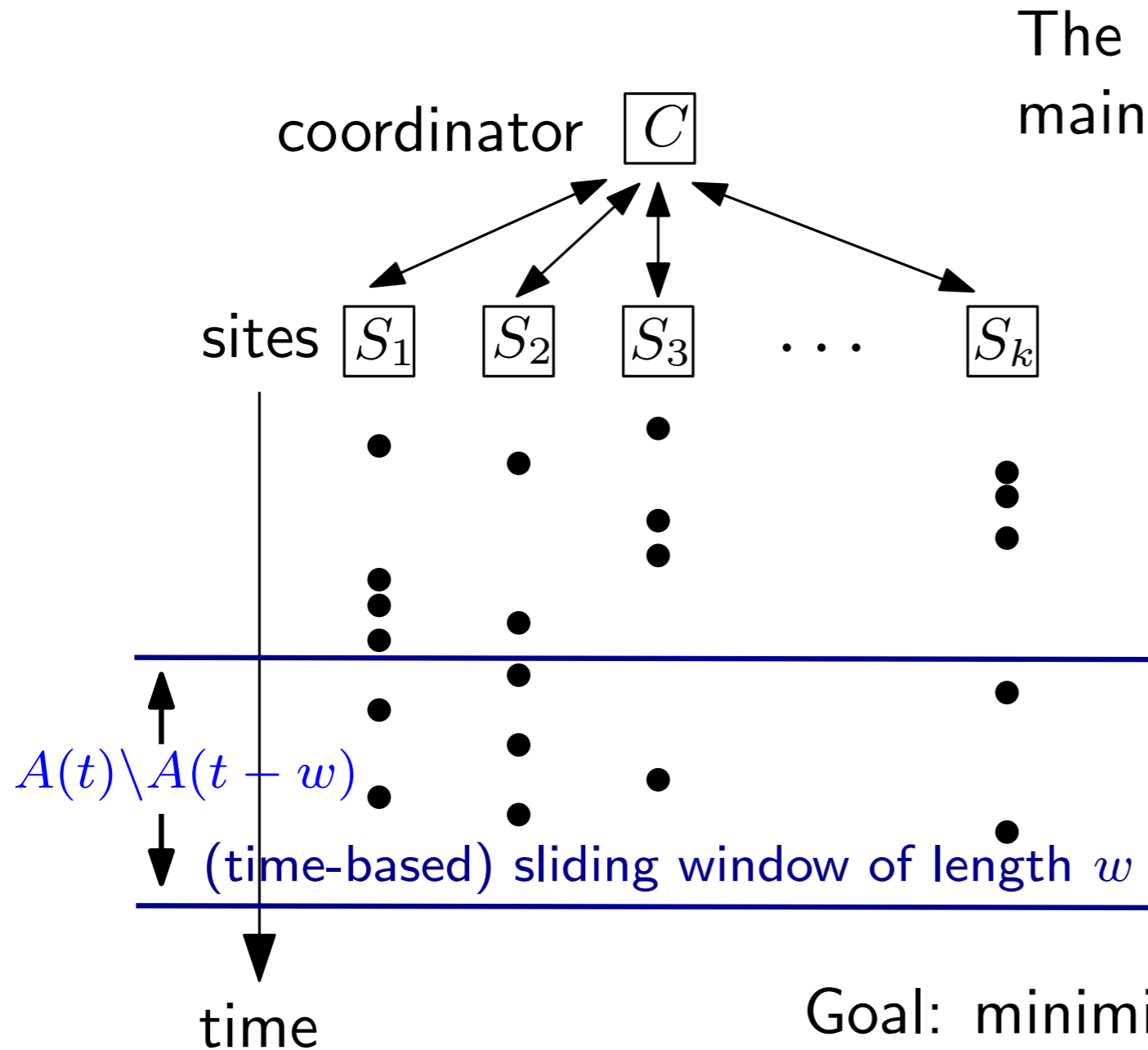


$A(t)$: set of elements received up to time t from all sites.
^a

^aAssume ≤ 1 item comes at each time unit.

Goal: minimize **communication cost**

The Distributed Streaming Model



The coordinator needs to maintain ~~$f(A(t))$~~ for **all** t .

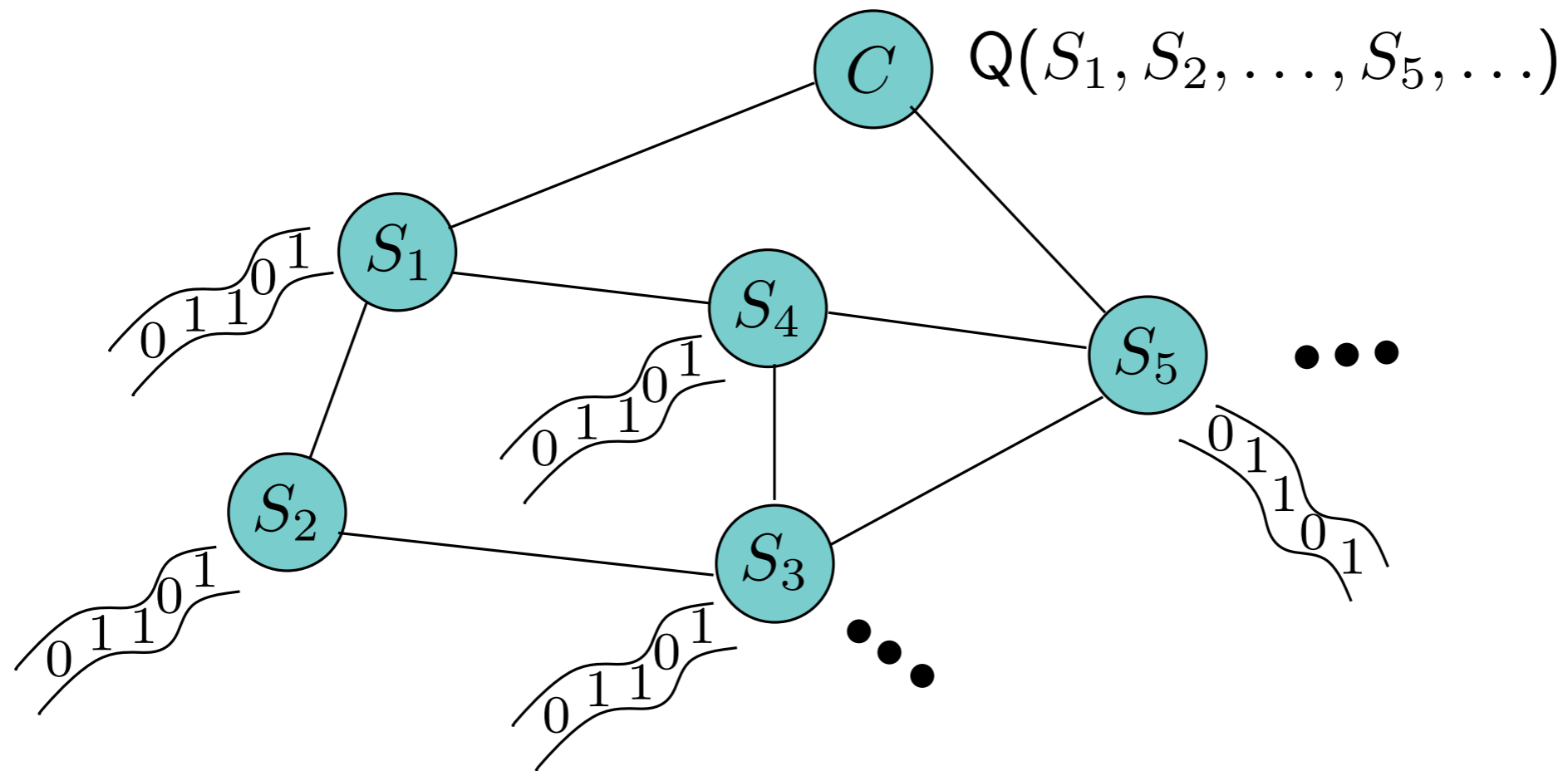
$$f(A(t) \setminus A(t-w))$$

$A(t)$: set of elements received up to time t from all sites.
^a

^aAssume ≤ 1 item comes at each time unit.

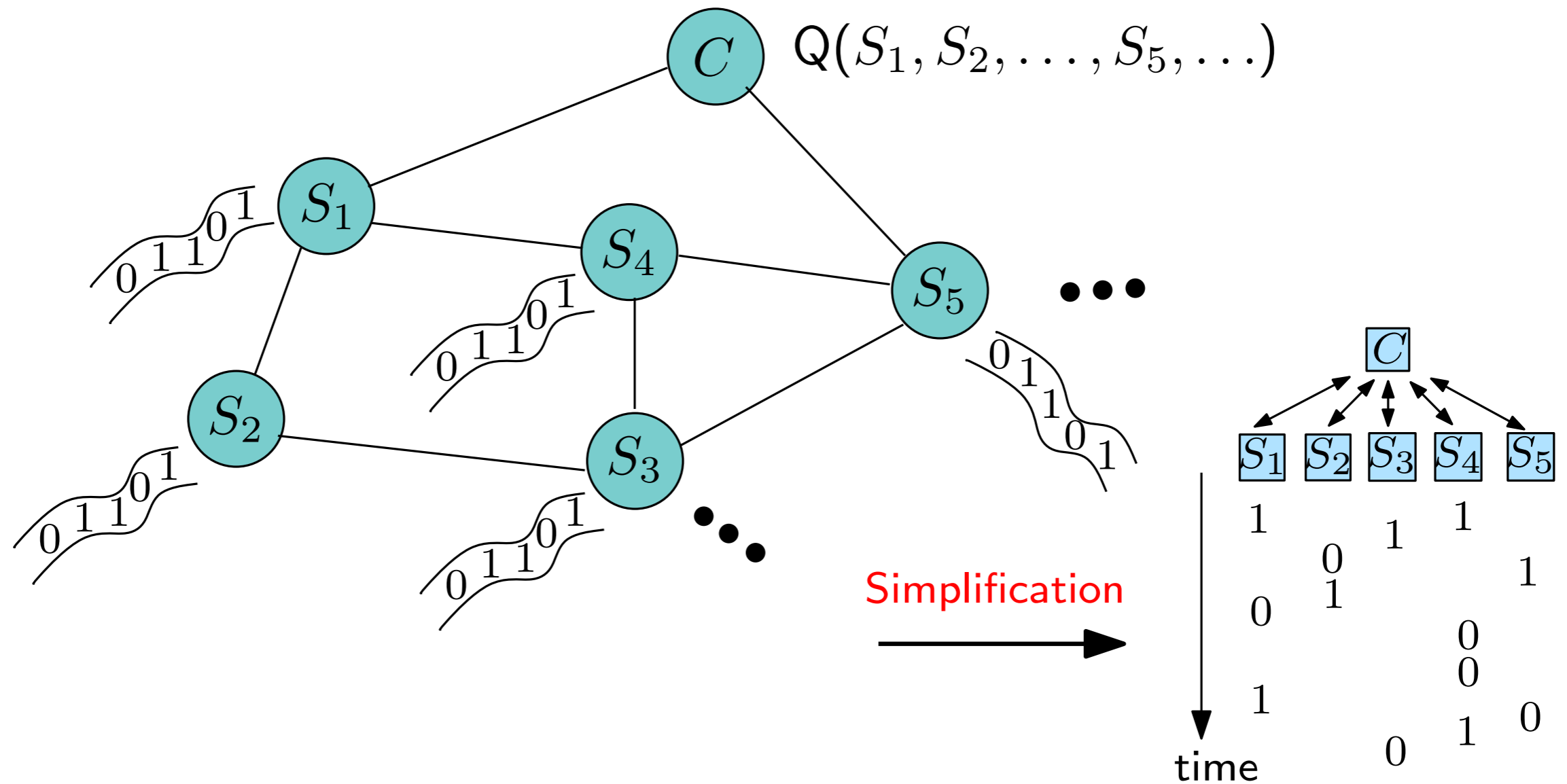
Goal: minimize **communication cost**

Applied Motivation: Distributed Monitoring



Large-scale **distributed** “holistic” querying/monitoring.

Applied Motivation: Distributed Monitoring



Large-scale **distributed** "holistic" querying/monitoring.

Related Models

- Communication complexity

Alice (has x) $\xleftrightarrow{\text{compute } f(x, y)}$ Bob (has y).
(can be extended to k -party)

1. Offline VS Online
2. One-shot VS Continuous

Related Models

- Communication complexity

Alice (has x) $\xleftrightarrow{\text{compute } f(x,y)}$ Bob (has y).
(can be extended to k -party)

- Offline VS Online
- One-shot VS Continuous

- Data streams



- Small space VS Communication cost.
- One stream VS k streams

Related Models

- Communication complexity

Alice (has x) $\xleftrightarrow{\text{compute } f(x,y)}$ Bob (has y).
(can be extended to k -party)

- Offline VS Online
- One-shot VS Continuous

- Data streams



- Small space VS Communication cost.
- One stream VS k streams

- Distributed streaming is a natural combination of communication complexity & data streams



Catalog of Problems

- ▣ Random Sampling: maintain a sample of size s at C
 - ▣ Sampling over an infinite window
 - ▣ Sampling from [sequence/time-based](#) sliding windows



Catalog of Problems

Each subset of size s of the n items is chosen with probability $1/\binom{n}{s}$.

- Random Sampling: maintain a sample of size s at C
 - Sampling over an infinite window
 - Sampling from [sequence/time-based](#) sliding windows



Catalog of Problems

- ▣ Random Sampling: maintain a sample of size s at C
 - ▣ Sampling over an infinite window
 - ▣ Sampling from [sequence/time-based](#) sliding windows

[Why study?](#) Random sampling is an extremely useful tool for many data analysis tasks: mean, standard deviation.

Catalog of Problems

- Random Sampling: maintain a sample of size s at C
 - Sampling over an infinite window
 - Sampling from **sequence/time-based** sliding windows

Why study? Random sampling is an extremely useful tool for many data analysis tasks: mean, standard deviation.

- Heavy Hitters and Quantiles
 - ϕ -heavy hitters: $\mathcal{H}_\phi(A) = \{x \mid \#x \text{ in } A \geq \phi|A|\}$
 - ϕ -quantile x : at most $\phi|A|$ items of A are **smaller** than x and at most $(1 - \phi)|A|$ are **greater** than x

Catalog of Problems

- Random Sampling: maintain a sample of size s at C
 - Sampling over an infinite window
 - Sampling from **sequence/time-based** sliding windows

Why study? Random sampling is an extremely useful tool for many data analysis tasks: mean, standard deviation.

- Heavy Hitters and Quantiles ϵ error: **heavy hitter** if $\#x > \phi|A|$,
non-heavy hitter if $\#x < (\phi - \epsilon)|A|$
 - ϕ -heavy hitters**: $\mathcal{H}_\phi(A) = \{x \mid \#x \text{ in } A \geq \phi|A|\}$
 - ϕ -quantile x** : at most $\phi|A|$ items of A are **smaller** than x and at most $(1 - \phi)|A|$ are **greater** than x
 $(1 - \phi + \epsilon)|A|$

Catalog of Problems

- Random Sampling: maintain a sample of size s at C
 - Sampling over an infinite window
 - Sampling from **sequence/time-based** sliding windows

Why study? Random sampling is an extremely useful tool for many data analysis tasks: mean, standard deviation.

- Heavy Hitters and Quantiles
 - **ϕ -heavy hitters**: $\mathcal{H}_\phi(A) = \{x \mid \#x \text{ in } A \geq \phi|A|\}$
 - **ϕ -quantile x** : at most $\phi|A|$ items of A are **smaller** than x and at most $(1 - \phi)|A|$ are **greater** than x
 - **All quantiles**: A DS that can answer all ϕ -quantile for $0 \leq \phi \leq 1$

Catalog of Problems

- Random Sampling: maintain a sample of size s at C
 - Sampling over an infinite window
 - Sampling from **sequence/time-based** sliding windows

Why study? Random sampling is an extremely useful tool for many data analysis tasks: mean, standard deviation.

- Heavy Hitters and Quantiles
 - **ϕ -heavy hitters**: $\mathcal{H}_\phi(A) = \{x \mid \#x \text{ in } A \geq \phi|A|\}$
 - **ϕ -quantile x** : at most $\phi|A|$ items of A are **smaller** than x and at most $(1 - \phi)|A|$ are **greater** than x
 - **All quantiles**: A DS that can answer all ϕ -quantile for $0 \leq \phi \leq 1$

Why study? Both are good descriptions of distribution of data



Catalog of Problems (cont.)

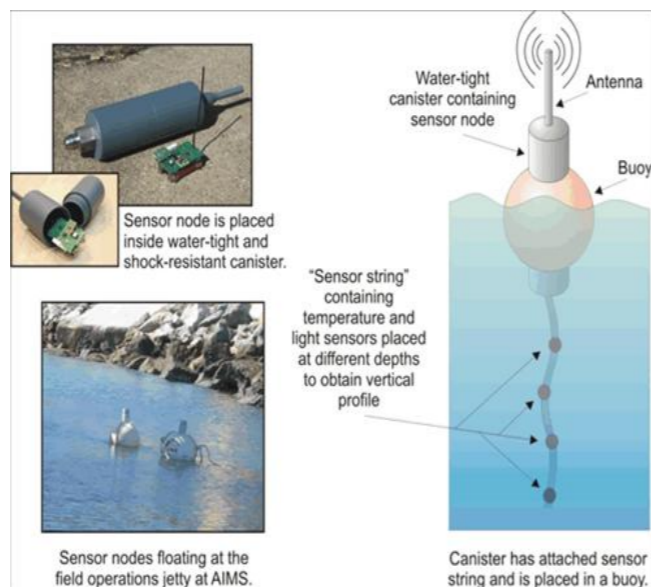
- ▣ General function $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}^d$ (one site). Allow Δ absolute error.

Catalog of Problems (cont.)

- General function $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}^d$ (one site). Allow Δ absolute error.

Why study?

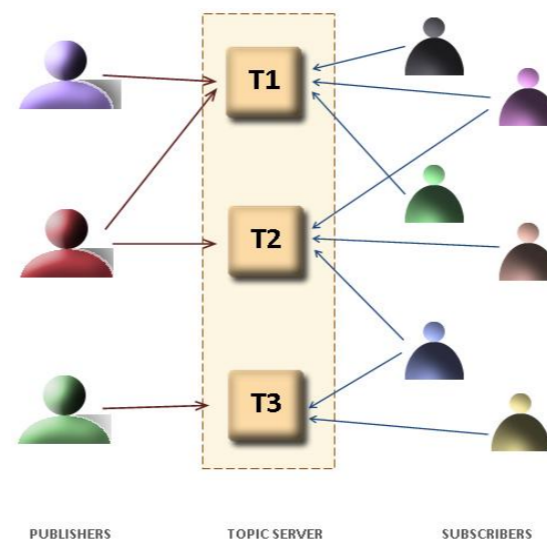
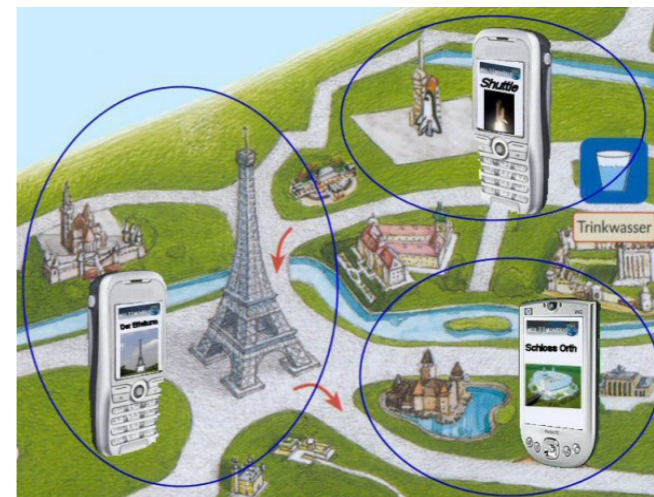
- Wireless sensors



Monitoring the temperature \rightarrow 1D case

- Publish/subscribe system

- Location-based services



Subscriber registers a query at the publisher; results (a set of items) changes over time \rightarrow high-D case



Previous Works

- Other works on Distributed Tracking
 - Frequency moments (Cormode, Muthukrishnan & Yi '08)
 - Entropy functions (Arackaparambil, Brody & Chakrabarti '09)



Previous Works

- ▣ Other works on Distributed Tracking
 - ▣ Frequency moments (Cormode, Muthukrishnan & Yi '08)
 - ▣ Entropy functions (Arackaparambil, Brody & Chakrabarti '09)
- ▣ Random Sampling
 - ▣ Single stream: [reservoir](#) sampling (Vitter, 1985)
 - ▣ Single stream sliding window: Braverman, Ostrovsky & Zaniolo '09

Previous Works

- ▣ Other works on Distributed Tracking
 - ▣ Frequency moments (Cormode, Muthukrishnan & Yi '08)
 - ▣ Entropy functions (Arackaparambil, Brody & Chakrabarti '09)
- ▣ Random Sampling
 - ▣ Single stream: [reservoir](#) sampling (Vitter, 1985)
 - ▣ Single stream sliding window: Braverman, Ostrovsky & Zaniolo '09
- ▣ Heavy Hitters and Quantiles
 - ▣ Single stream: quite a few works, see Muthukrishnan's survey '05
 - ▣ Distributed streams:
 - Heavy Hitters: Babcock & Olston '03, [heuristics](#).
 - All quantiles: $O(k/\epsilon^2 \cdot \log n)$. Cormode, Garofalaski, Muthukrishnan & Rastogi '05
 - Sliding window: Chan, Lam, Lee & Tin. '10

Our Results

- Random Sampling (without replacement; with replacement is similar)

window	upper bounds	lower bounds
infinite	$O((k + s) \log n)$	$\Omega(k + s \log n)$
sequence-based	$O(ks \log(w/s))$	$\Omega(ks \log(w/ks))$
time-based	$O((k + s) \log W)$	$\Omega(k + s \log W)$

w : length of sliding window. W : max # live items in a sliding window.

Our Results

- Random Sampling (without replacement; with replacement is similar)

window	upper bounds	lower bounds
infinite	$O((k + s) \log n)$	$\Omega(k + s \log n)$
sequence-based	$O(k s \log(w/s))$	$\Omega(k s \log(w/k s))$
time-based	$O((k + s) \log W)$	$\Omega(k + s \log W)$

w : length of sliding window. W : max # live items in a sliding window.

- Heavy Hitters and Quantiles

heavy hitters	one quantile	all quantiles
$\Theta(k/\epsilon \cdot \log n)$	$\Theta(k/\epsilon \cdot \log n)$	$O(k/\epsilon \cdot \log^2(1/\epsilon) \log n)$

ϵ : relative error.

Our Results

- Random Sampling (without replacement; with replacement is similar)

window	upper bounds	lower bounds
infinite	$O((k + s) \log n)$	$\Omega(k + s \log n)$
sequence-based	$O(ks \log(w/s))$	$\Omega(ks \log(w/ks))$
time-based	$O((k + s) \log W)$	$\Omega(k + s \log W)$

w : length of sliding window. W : max # live items in a sliding window.

- Heavy Hitters and Quantiles

heavy hitters	one quantile	all quantiles
$\Theta(k/\epsilon \cdot \log n)$	$\Theta(k/\epsilon \cdot \log n)$	$O(k/\epsilon \cdot \log^2(1/\epsilon) \log n)$

ϵ : relative error.

Our Results (cont.)

- General function $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}^d$ (one site).

problem	comp. ratio	running time
1-dim	$\Theta(\log \Delta)$	$O(1)$
d -dim	$O(d^2 \log(d\Delta))$	$\text{poly}(d, \log \Delta)$
1-dim + prediction	$O(\log(\Delta T))$	$\text{poly}(\Delta, T)$

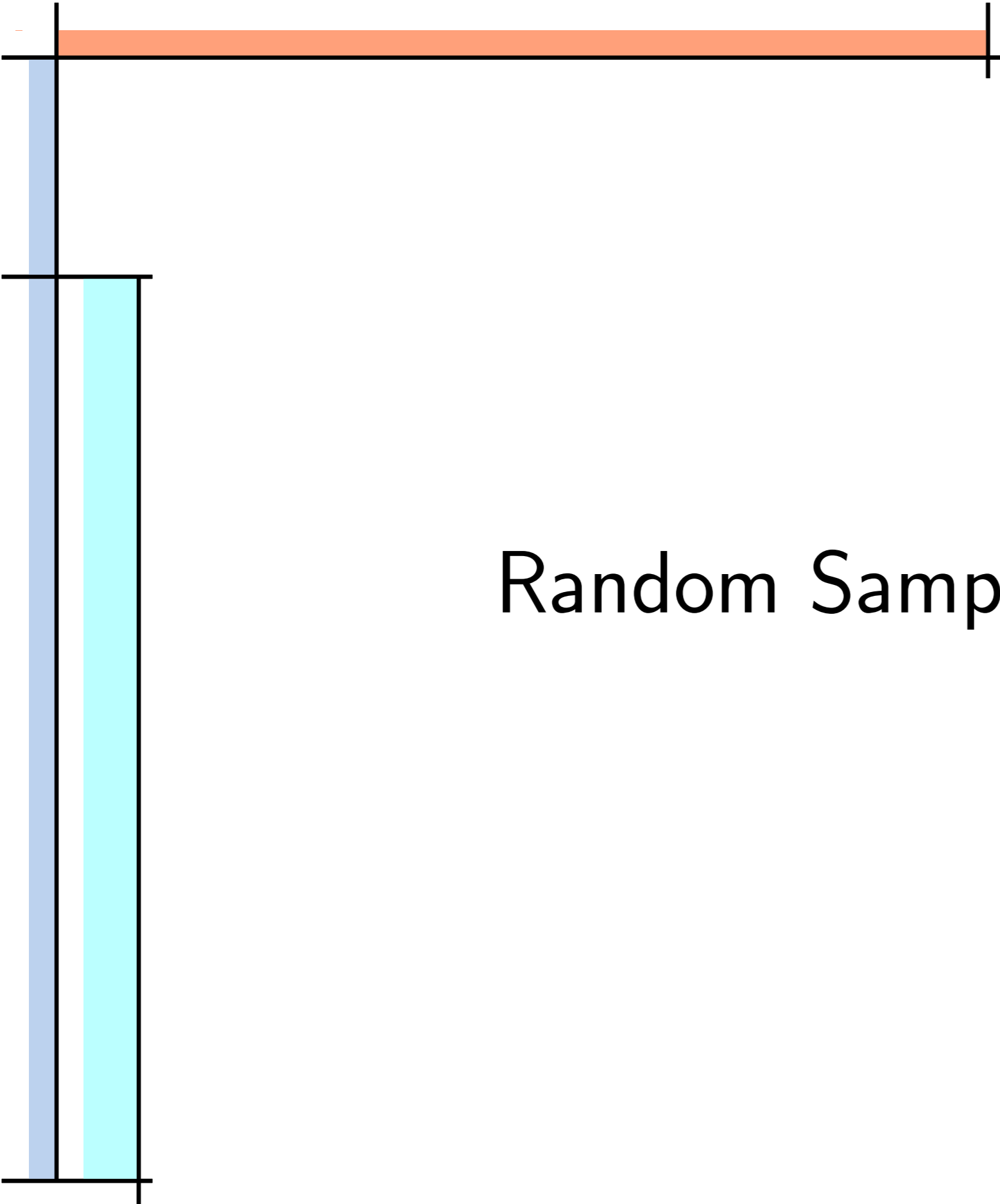
Δ : absolute error. T : length of the tracking period.

Our Results (cont.)

- General function $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}^d$ (one site).

problem	comp. ratio	running time
1-dim	$O(\log \Delta)$	$O(1)$
d -dim	$O(d^2 \log(d\Delta))$	$\text{poly}(d, \log \Delta)$
1-dim + prediction	$O(\log(\Delta T))$	$\text{poly}(\Delta, T)$

Δ : absolute error. T : length of the tracking period.



Random Sampling



Random Sampling

- Goal: maintain a random sampling of size s at C .
- We give two examples:
 1. Infinite window without replacement (**ISWoR**).
 2. Time-based window without replacement (**TSWoR**).
- Some notation:
 - k : the number of sites
 - w : length of the sliding window
 - W : maximum # live items in a sliding window



ISWoR

- The protocol

- **Site:** always maintains an **upper bound** u (initialized to be 1) and **lower bound** l (initialized to be 0), and **only sends those items with rank in the range $[l, u]$.**

Rank: for each item coming, generate a random number in $[0, 1]$ as its rank.

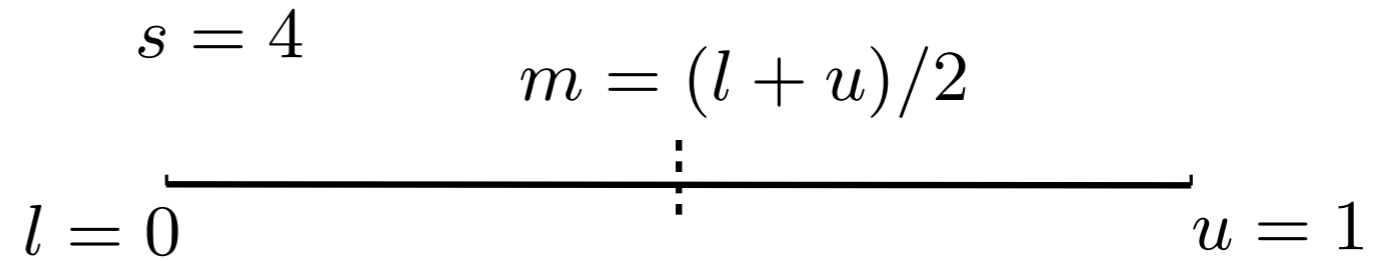
ISWoR

- The protocol
 - **Site:** always maintains an **upper bound** u (initialized to be 1) and **lower bound** l (initialized to be 0), and **only sends those items with rank in the range** $[l, u]$.
 - **Coordinator:** let $m = (l + u)/2$, waits until
 - $\#$ items received in the range $[l, m]$ becomes $\geq s$, **updates** each site with $u = m$.
 - $\#$ items received in the range $[m, u]$ becomes $\geq s$, **updates** each site with $l = m$.

Report:

subsamples s items from all items in $[l, u]$.

ISWoR



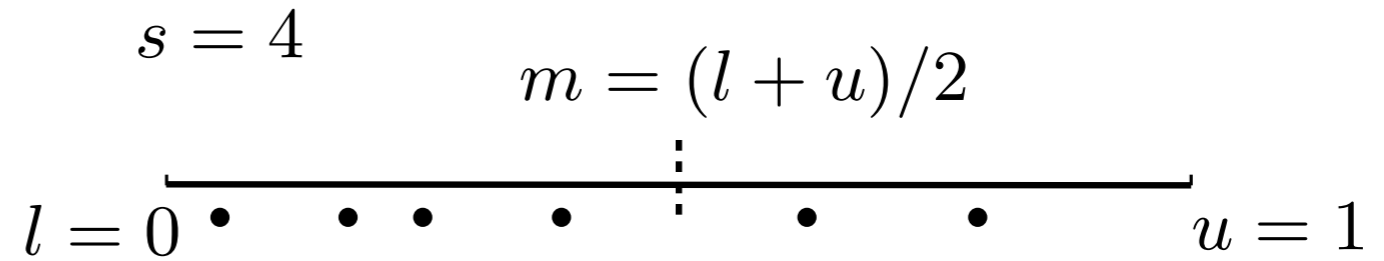
□ The protocol

- **Site:** always maintains an **upper bound** u (initialized to be 1) and **lower bound** l (initialized to be 0), and **only sends those items with rank in the range** $[l, u]$.
- **Coordinator:** let $m = (l + u)/2$, waits until
 - $\#$ items received in the range $[l, m]$ becomes $\geq s$, **updates** each site with $u = m$.
 - $\#$ items received in the range $[m, u]$ becomes $\geq s$, **updates** each site with $l = m$.

Report:

subsamples s items from all items in $[l, u]$.

ISWoR



□ The protocol

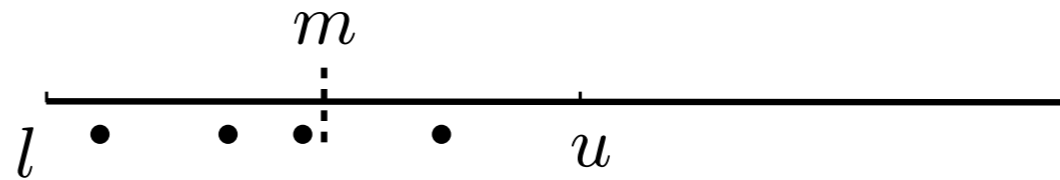
- **Site:** always maintains an **upper bound** u (initialized to be 1) and **lower bound** l (initialized to be 0), and **only sends those items with rank in the range** $[l, u]$.
- **Coordinator:** let $m = (l + u)/2$, waits until
 - $\#$ items received in the range $[l, m]$ becomes $\geq s$, **updates** each site with $u = m$.
 - $\#$ items received in the range $[m, u]$ becomes $\geq s$, **updates** each site with $l = m$.

Report:

subsamples s items from all items in $[l, u]$.

ISWoR

$$s = 4$$



□ The protocol

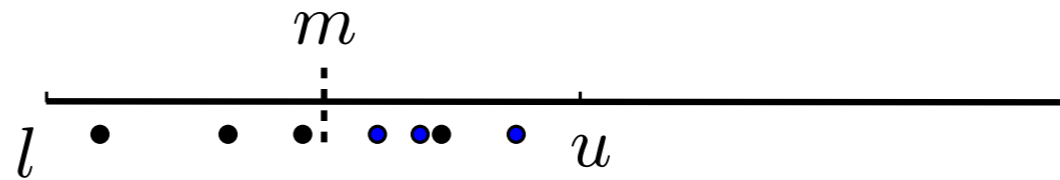
- **Site:** always maintains an **upper bound** u (initialized to be 1) and **lower bound** l (initialized to be 0), and **only sends those items with rank in the range** $[l, u]$.
- **Coordinator:** let $m = (l + u)/2$, waits until
 - $\#$ items received in the range $[l, m]$ becomes $\geq s$, **updates** each site with $u = m$.
 - $\#$ items received in the range $[m, u]$ becomes $\geq s$, **updates** each site with $l = m$.

Report:

subsamples s items from all items in $[l, u]$.

ISWoR

$$s = 4$$



□ The protocol

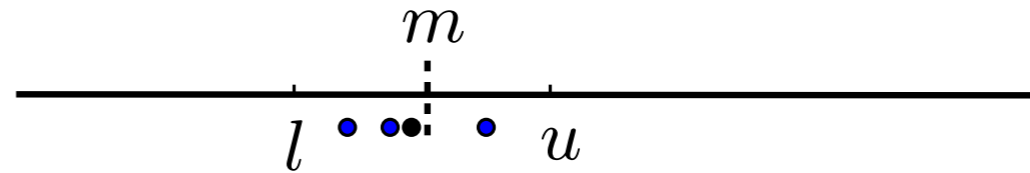
- **Site:** always maintains an **upper bound** u (initialized to be 1) and **lower bound** l (initialized to be 0), and **only sends those items with rank in the range** $[l, u]$.
- **Coordinator:** let $m = (l + u)/2$, waits until
 - $\#$ items received in the range $[l, m]$ becomes $\geq s$, **updates** each site with $u = m$.
 - $\#$ items received in the range $[m, u]$ becomes $\geq s$, **updates** each site with $l = m$.

Report:

subsamples s items from all items in $[l, u]$.

ISWoR

$$s = 4$$



□ The protocol

□ **Site:** always maintains an **upper bound** u (initialized to be 1) and **lower bound** l (initialized to be 0), and **only sends those items with rank in the range** $[l, u]$.

□ **Coordinator:** let $m = (l + u)/2$, waits until

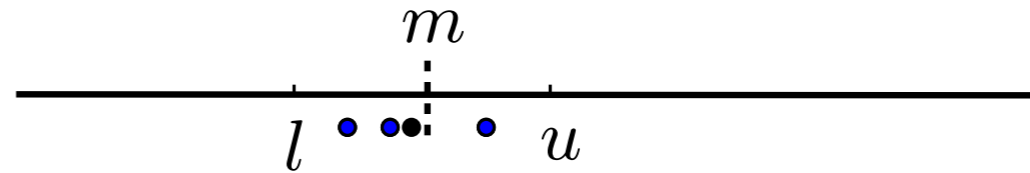
- $\#$ items received in the range $[l, m]$ becomes $\geq s$, **updates** each site with $u = m$.
- $\#$ items received in the range $[m, u]$ becomes $\geq s$, **updates** each site with $l = m$.

Report:

subsamples s items from all items in $[l, u]$.

ISWoR

$$s = 4$$



- The protocol
 - **Site:** always maintains an **upper bound** u (initialized to be 1) and **lower bound** l (initialized to be 0), and **only sends those items with rank in the range** $[l, u]$.
 - **Coordinator:** let $m = (l + u)/2$, waits until
 - $\#$ items received in the range $[l, m]$ becomes $\geq s$, **updates** each site with $u = m$.
 - $\#$ items received in the range $[m, u]$ becomes $\geq s$, **updates** each site with $l = m$.

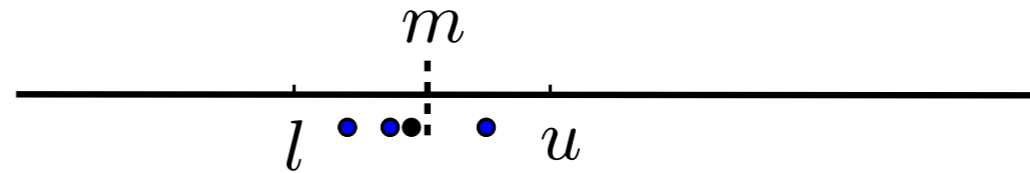
Report:

subsamples s items from all items in $[l, u]$.

Like Binary Search :)

ISWoR

$$s = 4$$



□ The protocol

- **Site:** always maintains an **upper bound** u (initialized to be 1) and **lower bound** l (initialized to be 0), and **only sends those items with rank in the range** $[l, u]$.
- **Coordinator:** let $m = (l + u)/2$, waits until
 - $\#$ items received in the range $[l, m]$ becomes $\geq s$, **updates** each site with $u = m$.
 - $\#$ items received in the range $[m, u]$ becomes $\geq s$, **updates** each site with $l = m$.

Report:

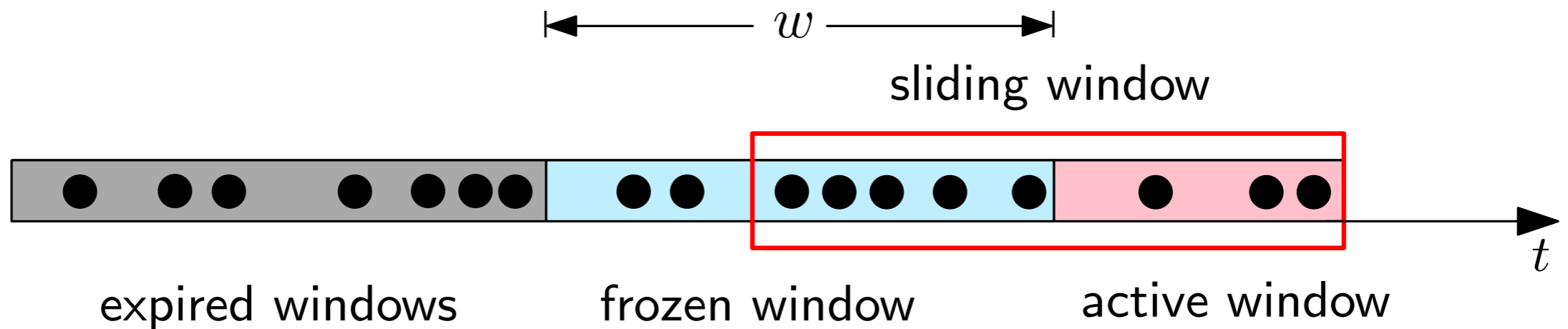
subsamples s items from all items in $[l, u]$.

Like Binary Search :)

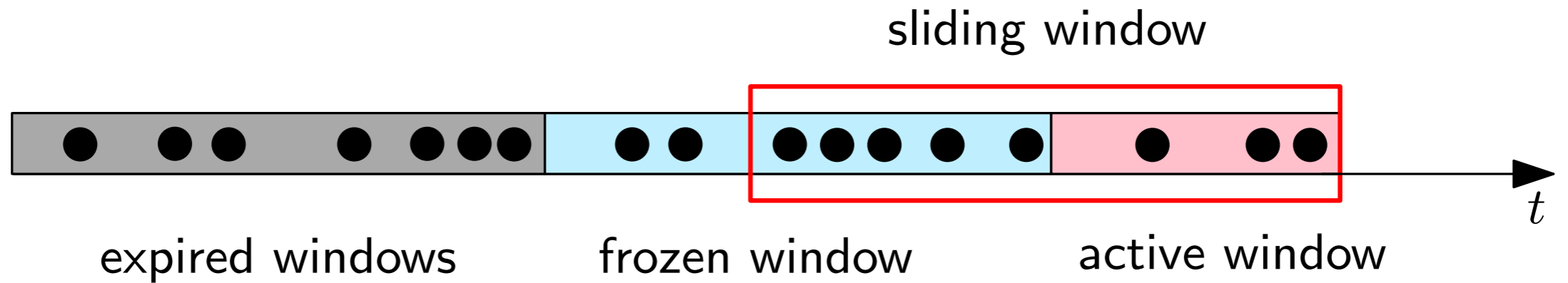
Communication cost: $(k + s) \log n$

TSWoR

- Split the whole stream to windows of size w (take a **global** view).

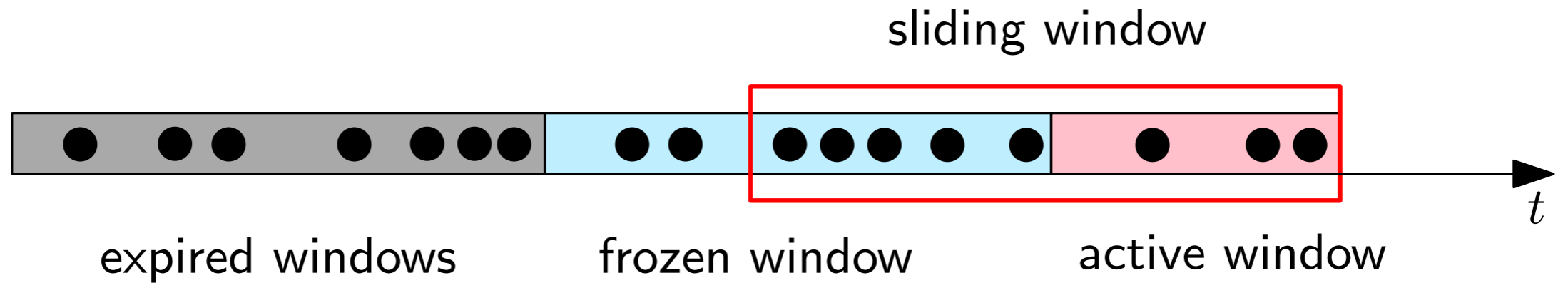


TSWoR



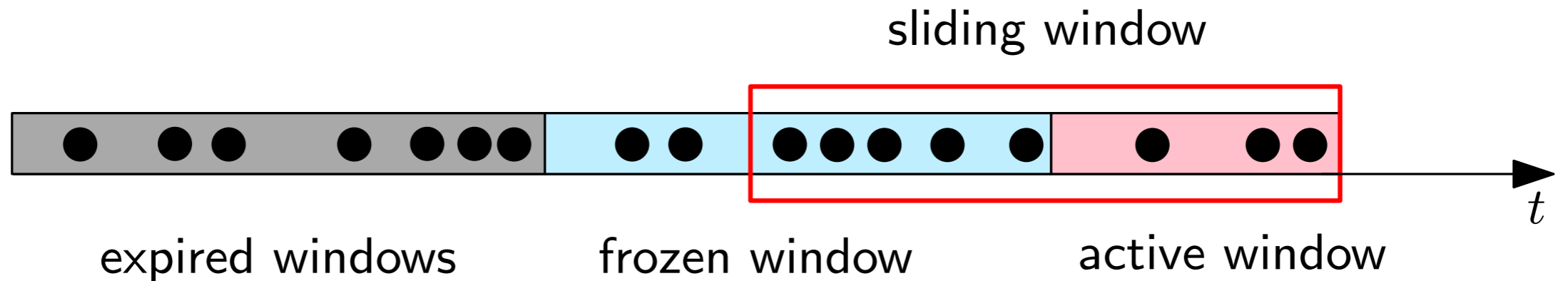
- Sample for sliding window =
 - (1) a subsample of the (unexpired) sample of frozen window +
 - (2) a subsample of the sample of current window

TSWoR



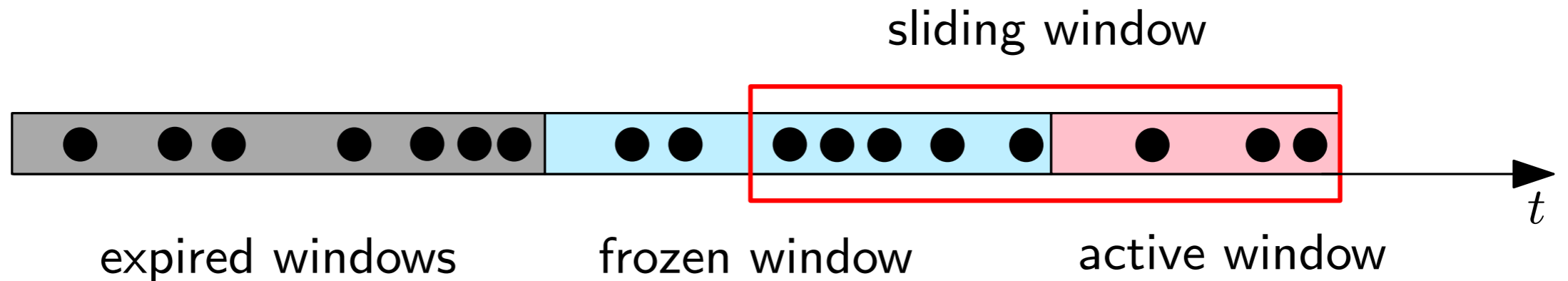
- Sample for sliding window = need new ideas
(1) a subsample of the (unexpired) sample of frozen window +
(2) a subsample of the sample of current window by ISWoR

TSWoR



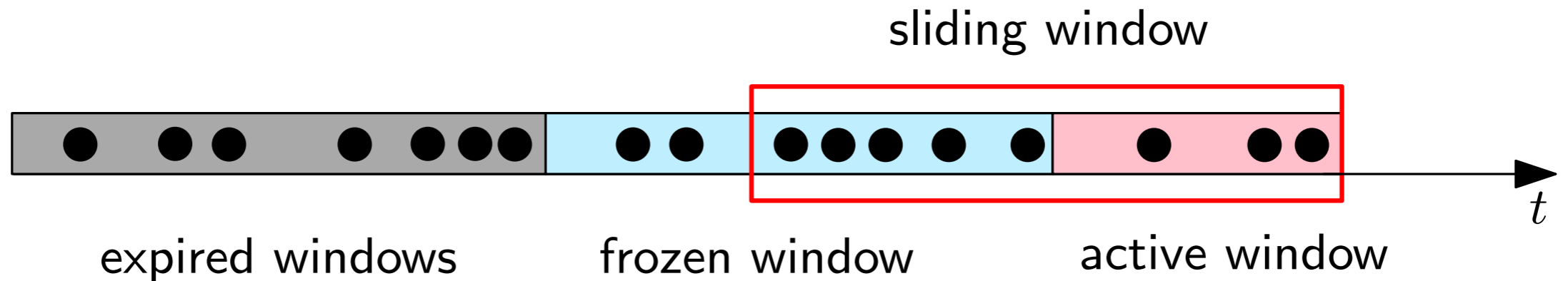
- Sample for sliding window = need new ideas
(1) a subsample of the (unexpired) sample of frozen window +
(2) a subsample of the sample of current window by ISWoR
- (1), (2) may be sampled by different rates.
But as long as both have sizes $\geq \min\{s, \# \text{ live items}\}$, fine.

TSWoR



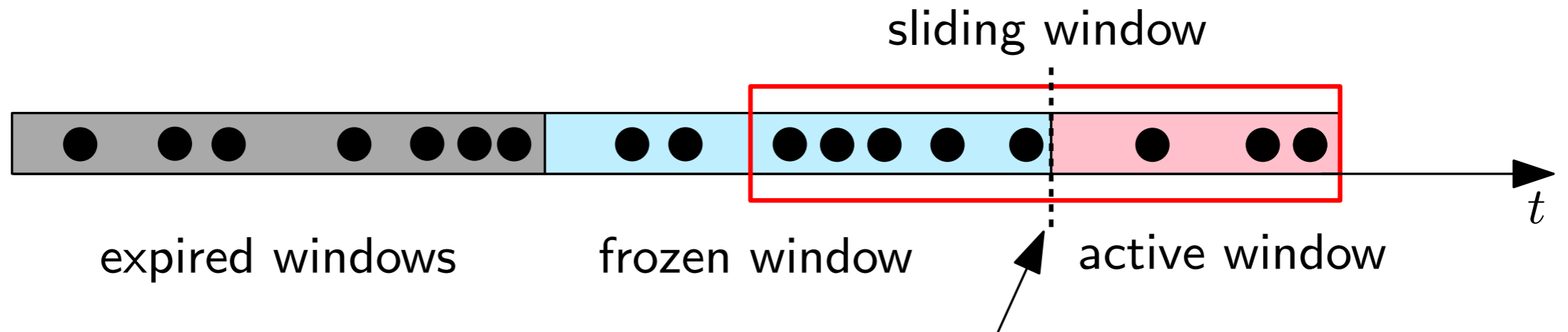
- Sample for sliding window = need new ideas
(1) a subsample of the (unexpired) sample of frozen window +
(2) a subsample of the sample of current window by ISWoR
- (1), (2) may be sampled by different rates.
But as long as both have sizes $\geq \min\{s, \# \text{ live items}\}$, fine.
- The key issue: how to guarantee “both have sizes $\geq s$ ”?
as items in the frozen window are expiring ...

TSWoR



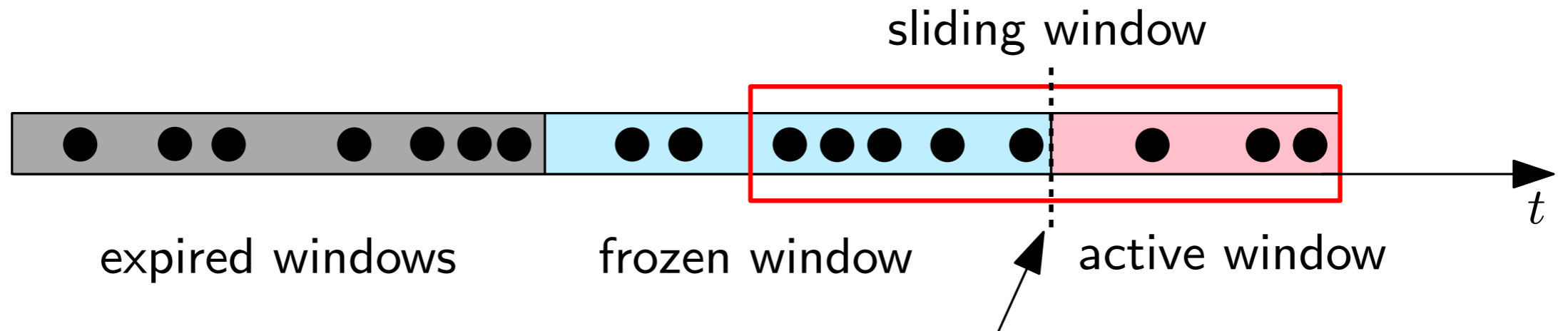
- Sample for sliding window = need new ideas
(1) a subsample of the (unexpired) sample of frozen window +
(2) a subsample of the sample of current window by ISWoR
- (1), (2) may be sampled by different rates.
But as long as both have sizes $\geq \min\{s, \# \text{ live items}\}$, fine.
- The key issue: how to guarantee “both have sizes $\geq s$ ”?
as items in the frozen window are expiring ...
- Solution: In the frozen window, find a good sample rate such that the sample size $\geq s$. Assume we can “restart the protocol”.

TSWoR (cont.)



Avoid “restart”: Build a **level-sampling structure** at C for the frozen window so that a sample of $\geq s$ live items can be extracted at any future w time steps.

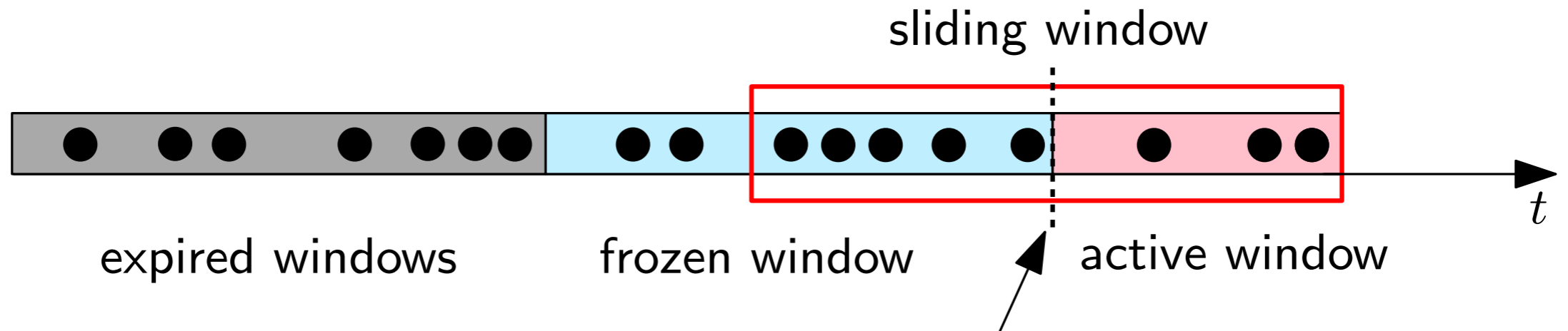
TSWoR (cont.)



Avoid “restart”: Build a **level-sampling structure** at C for the frozen window so that a sample of $\geq s$ live items can be extracted at any future w time steps.

- Each site **builds its own** level-sampling structure for the current window until it freezes.
- Can be done in $O(s \log W)$ space and $O(1)$ time per item.

TSWoR (cont.)



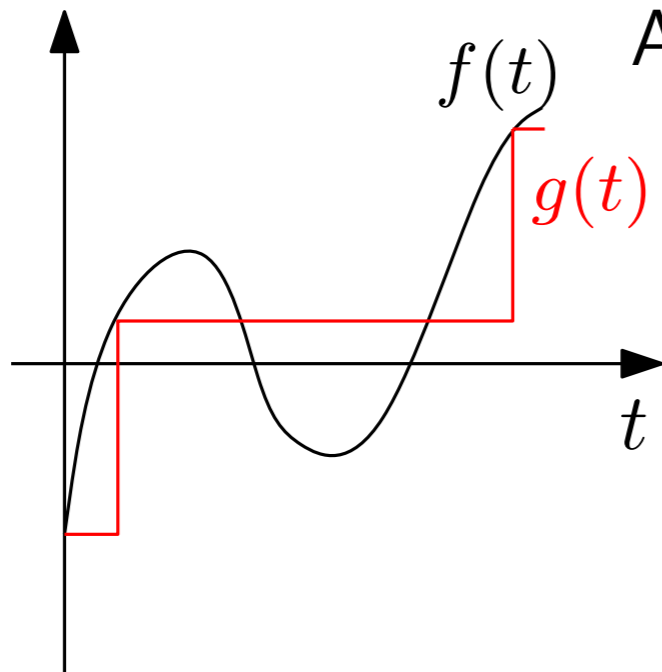
Avoid “restart”: Build a **level-sampling structure** at C for the frozen window so that a sample of $\geq s$ live items can be extracted at any future w time steps.

- Each site **builds its own** level-sampling structure for the current window until it freezes.
 - Can be done in $O(s \log W)$ space and $O(1)$ time per item.
- When the current window freezes, for each level, do a **k -way merge** to build the global level-sampling structure at C .
 - Total communication $O((k + s) \log W)$.



Tracking Arbitrary Functions

Online Tracking



Alice: *observer*



$(t, g(t))$

Bob: *tracker*

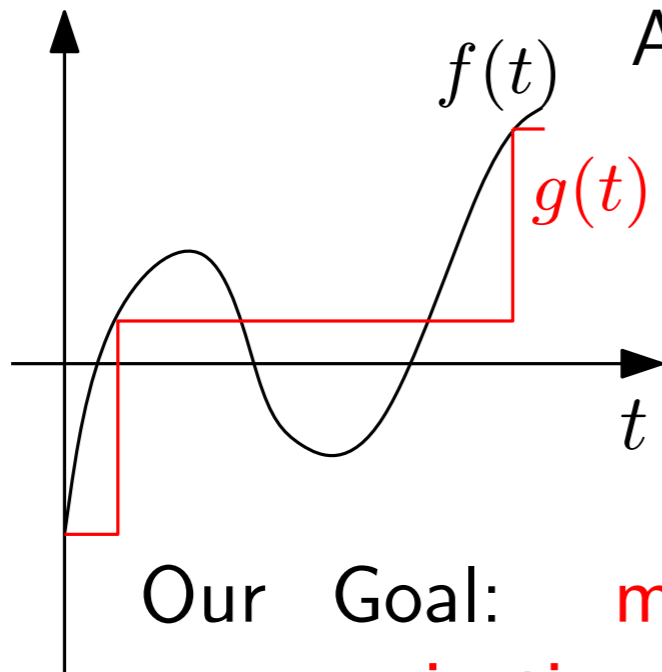


- message format at time t_{now} : $(t_{now}, g(t_{now}))$.
- communicate to guarantee that at $\forall t_{now}$

$$\|f(t_{now}) - g(t_{last})\| \leq \Delta$$

t_{last} : the last time Bob get informed.

Online Tracking



Alice: *observer*



$(t, g(t))$

Bob: *tracker*



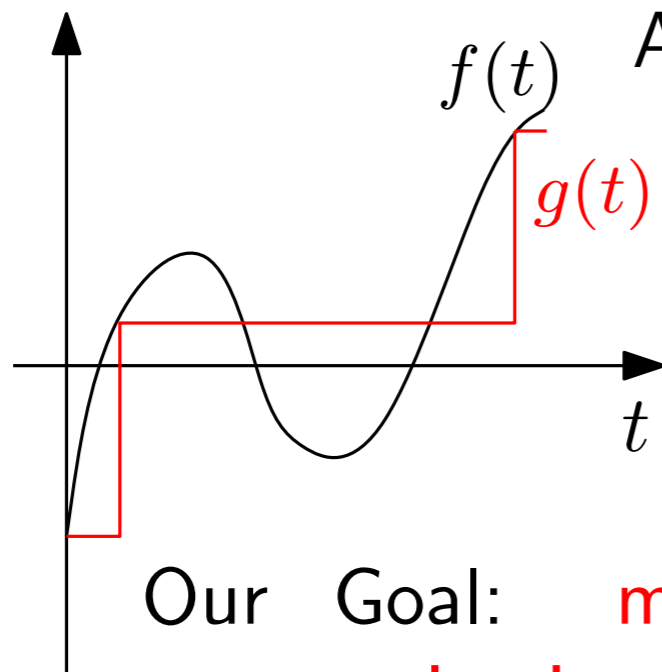
Our Goal: **minimize the communication**, in terms of *competitive ratio*. In

- message format at time t_{now} : $(t_{now}, g(t_{now}))$.
- communicate to guarantee that at $\forall t_{now}$

$$\|f(t_{now}) - g(t_{last})\| \leq \Delta$$

t_{last} : the last time Bob get informed.

Online Tracking



Alice: *observer*



$(t, g(t))$

Bob: *tracker*



Our Goal: **minimize the communication**, in terms of *competitive ratio*. In

$\mathbf{f} : \mathbf{Z}^+ \rightarrow \mathbf{Z}^d$
(today's focus: $d = 1$)

- message format at time t_{now} : $(t_{now}, g(t_{now}))$.
- communicate to guarantee that at $\forall t_{now}$

$$\|f(t_{now}) - g(t_{last})\| \leq \Delta$$

t_{last} : the last time Bob get informed.

Naive Solution Fails

- Consider tracking the function $f : Z^+ \rightarrow Z$, and require an absolute error of at most Δ .
- The natural solution is to
 1. first communicate $f(0)$ to Bob.
 2. every time $f(t)$ has changed by more than Δ since the last communication, Alice updates Bob with the current $f(t)$.

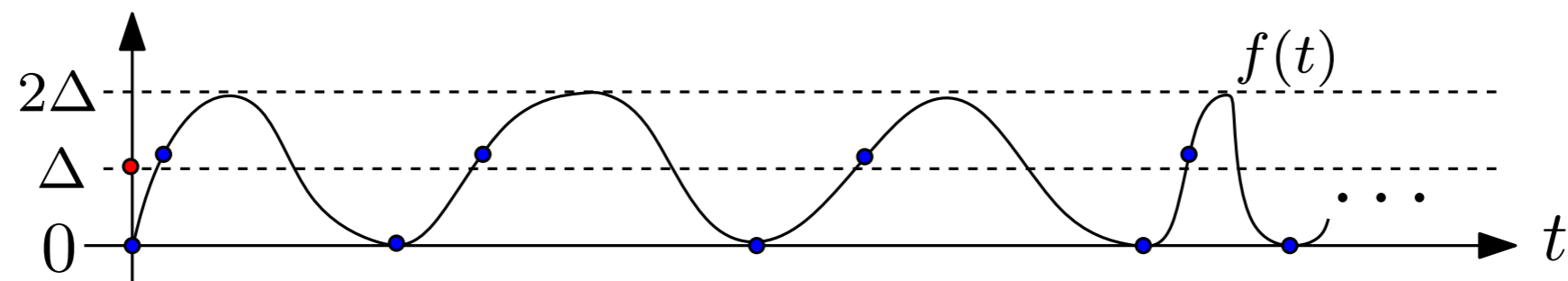
Naive Solution Fails

- Consider tracking the function $f : Z^+ \rightarrow Z$, and require an absolute error of at most Δ .
- The natural solution is to
 1. first communicate $f(0)$ to Bob.
 2. every time $f(t)$ has changed by more than Δ since the last communication, Alice updates Bob with the current $f(t)$.
- **Unbounded competitive ratio!**

Naive Solution Fails

- Consider tracking the function $f : Z^+ \rightarrow Z$, and require an absolute error of at most Δ .
- The natural solution is to
 1. first communicate $f(0)$ to Bob.
 2. every time $f(t)$ has changed by more than Δ since the last communication, Alice updates Bob with the current $f(t)$.

□ **Unbounded competitive ratio!**



SOL = ∞ , OPT = 1!

Our Solution

- General idea to track $f : Z^+ \rightarrow Z$.

Divide the whole tracking period into **rounds**, and show that \mathcal{A}_{OPT} must communicate once **in each round**, while our **algorithm** communicates at most, say, k times

→ **competitive ratio** k .

Our Solution (Cont.)

- The Algorithm to track $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}$

Algorithm 1: One round of 1D tracking

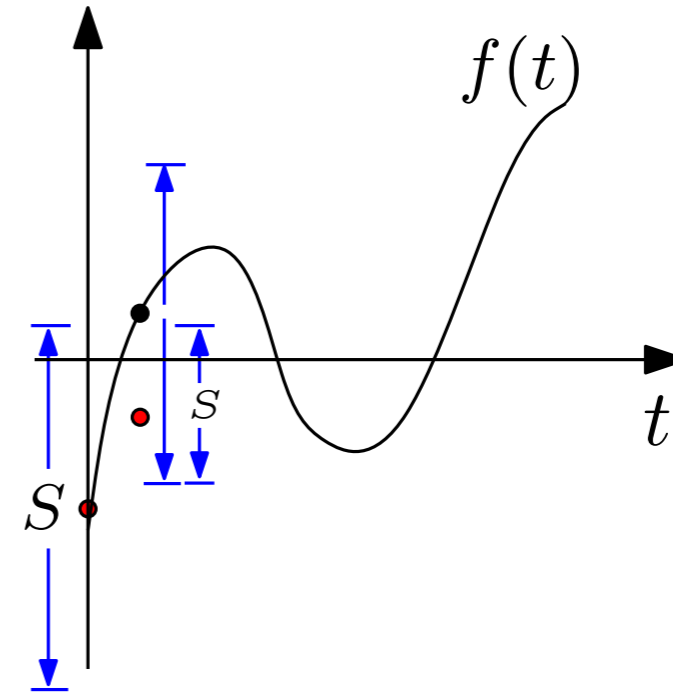
```
1 let  $S = [f(t_{now}) - \Delta, f(t_{now}) + \Delta] \cap \mathbb{Z}$ ;  
2 while  $S \neq \emptyset$  do  
3   | let  $g(t_{now})$  be the median of  $S$ ;  
4   | send  $g(t_{now})$  to Bob;  
5   | wait until  $\|f(t_{now}) - g(t_{last})\| > \Delta$ ;  
6   |  $S \leftarrow S \cap [f(t_{now}) - \Delta, f(t_{now}) + \Delta]$ ;
```

Our Solution (Cont.)

- The Algorithm to track $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}$

Algorithm 1: One round of 1D tracking

- 1 let $S = [f(t_{now}) - \Delta, f(t_{now}) + \Delta] \cap \mathbb{Z}$;
 - 2 **while** $S \neq \emptyset$ **do**
 - 3 let $g(t_{now})$ be the median of S ;
 - 4 send $g(t_{now})$ to Bob;
 - 5 wait until $\|f(t_{now}) - g(t_{last})\| > \Delta$;
 - 6 $S \leftarrow S \cap [f(t_{now}) - \Delta, f(t_{now}) + \Delta]$;
-

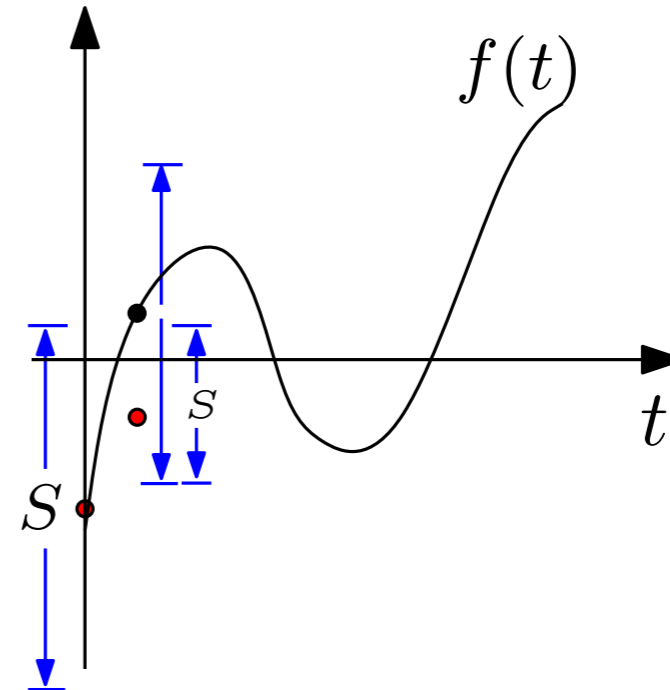


Our Solution (Cont.)

- The Algorithm to track $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}$

Algorithm 1: One round of 1D tracking

```
1 let  $S = [f(t_{now}) - \Delta, f(t_{now}) + \Delta] \cap \mathbb{Z}$ ;  
2 while  $S \neq \emptyset$  do  
3   let  $g(t_{now})$  be the median of  $S$ ;  
4   send  $g(t_{now})$  to Bob;  
5   wait until  $\|f(t_{now}) - g(t_{last})\| > \Delta$ ;  
6    $S \leftarrow S \cap [f(t_{now}) - \Delta, f(t_{now}) + \Delta]$ ;
```



- The Analysis

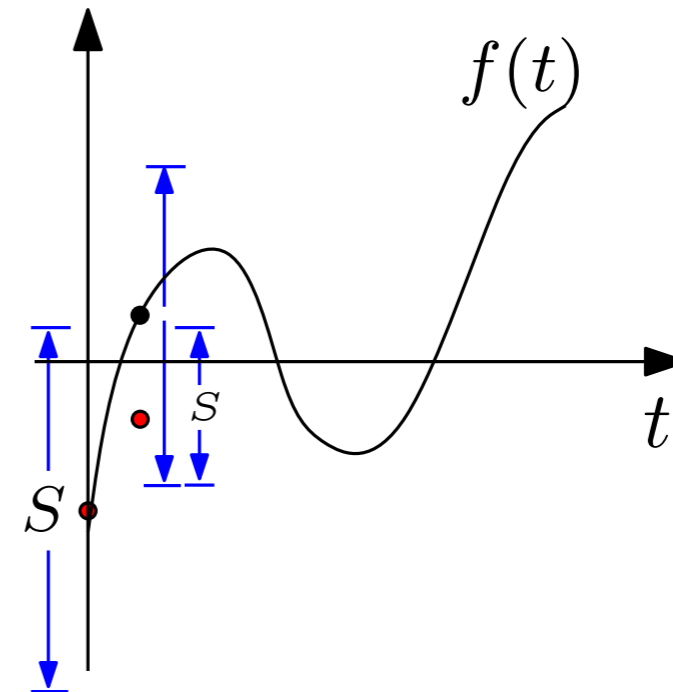
- If \mathcal{A}_{OPT} hasn't send a message in the current round, then the value sent by its last message must be included in S .
- The cardinality of S decreases by half whenever Algorithm 1 sends a message.

Our Solution (Cont.)

- The Algorithm to track $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}$

Algorithm 1: One round of 1D tracking

```
1 let  $S = [f(t_{now}) - \Delta, f(t_{now}) + \Delta] \cap \mathbb{Z}$ ;  
2 while  $S \neq \emptyset$  do  
3   let  $g(t_{now})$  be the median of  $S$ ;  
4   send  $g(t_{now})$  to Bob;  
5   wait until  $\|f(t_{now}) - g(t_{last})\| > \Delta$ ;  
6    $S \leftarrow S \cap [f(t_{now}) - \Delta, f(t_{now}) + \Delta]$ ;
```



- The Analysis

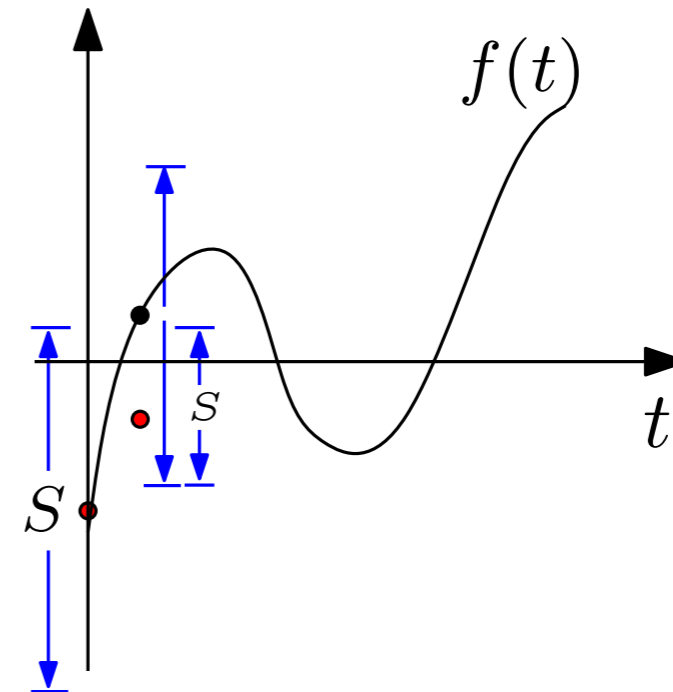
- If \mathcal{A}_{OPT} hasn't send a message in the current round, then the value sent by its last message must be included in S . $\Rightarrow O(\log \Delta)$ -competitive
- The cardinality of S decreases by half whenever Algorithm 1 sends a message.

Our Solution (Cont.)

- The Algorithm to track $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}$

Algorithm 1: One round of 1D tracking

```
1 let  $S = [f(t_{now}) - \Delta, f(t_{now}) + \Delta] \cap \mathbb{Z}$ ;  
2 while  $S \neq \emptyset$  do  
3   let  $g(t_{now})$  be the median of  $S$ ;  
4   send  $g(t_{now})$  to Bob;  
5   wait until  $\|f(t_{now}) - g(t_{last})\| > \Delta$ ;  
6    $S \leftarrow S \cap [f(t_{now}) - \Delta, f(t_{now}) + \Delta]$ ;
```



- The Analysis

- If \mathcal{A}_{OPT} hasn't send a message in the current round, then the value sent by its last message must be included in S . $\Rightarrow O(\log \Delta)$ -competitive
- The cardinality of S decreases by half whenever Algorithm 1 sends a message. Also tight!

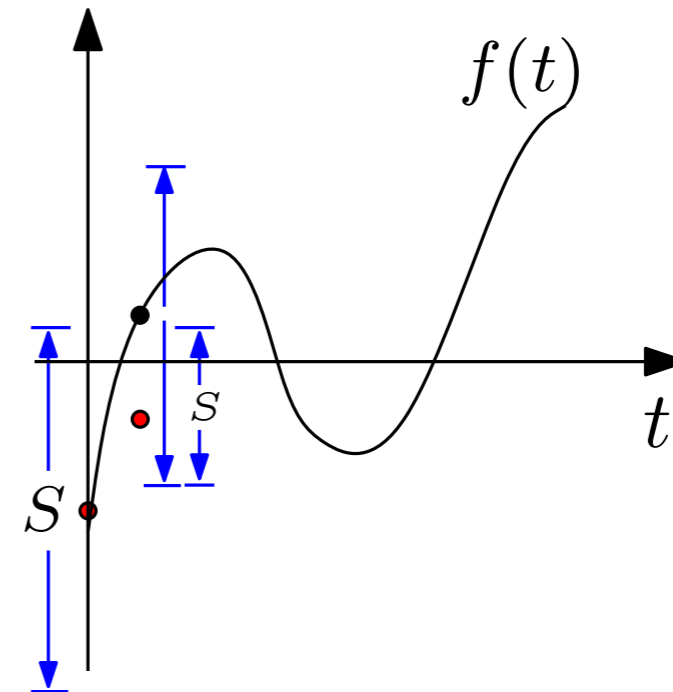
Our Solution (Cont.)

- The Algorithm to track $f : \mathbb{Z}^+ \rightarrow \mathbb{Z}$

Algorithm 1: One round of 1D tracking

```
1 let  $S = [f(t_{now}) - \Delta, f(t_{now}) + \Delta] \cap \mathbb{Z}$ ;  
2 while  $S \neq \emptyset$  do  
3   let  $g(t_{now})$  be the median of  $S$ ;  
4   send  $g(t_{now})$  to Bob;  
5   wait until  $\|f(t_{now}) - g(t_{last})\| > \Delta$ ;  
6    $S \leftarrow S \cap [f(t_{now}) - \Delta, f(t_{now}) + \Delta]$ ;
```

Real domain
unbounded!



- The Analysis

- If \mathcal{A}_{OPT} hasn't send a message in the current round, then the value sent by its last message must be included in S . $\Rightarrow O(\log \Delta)$ -competitive
- The cardinality of S decreases by half whenever Algorithm 1 sends a message. Also tight!



High Dimensions

- The general idea follows from 1D

Divide the whole tracking period into **rounds**, and show that **the competitive ratio in each round is k** .

High Dimensions

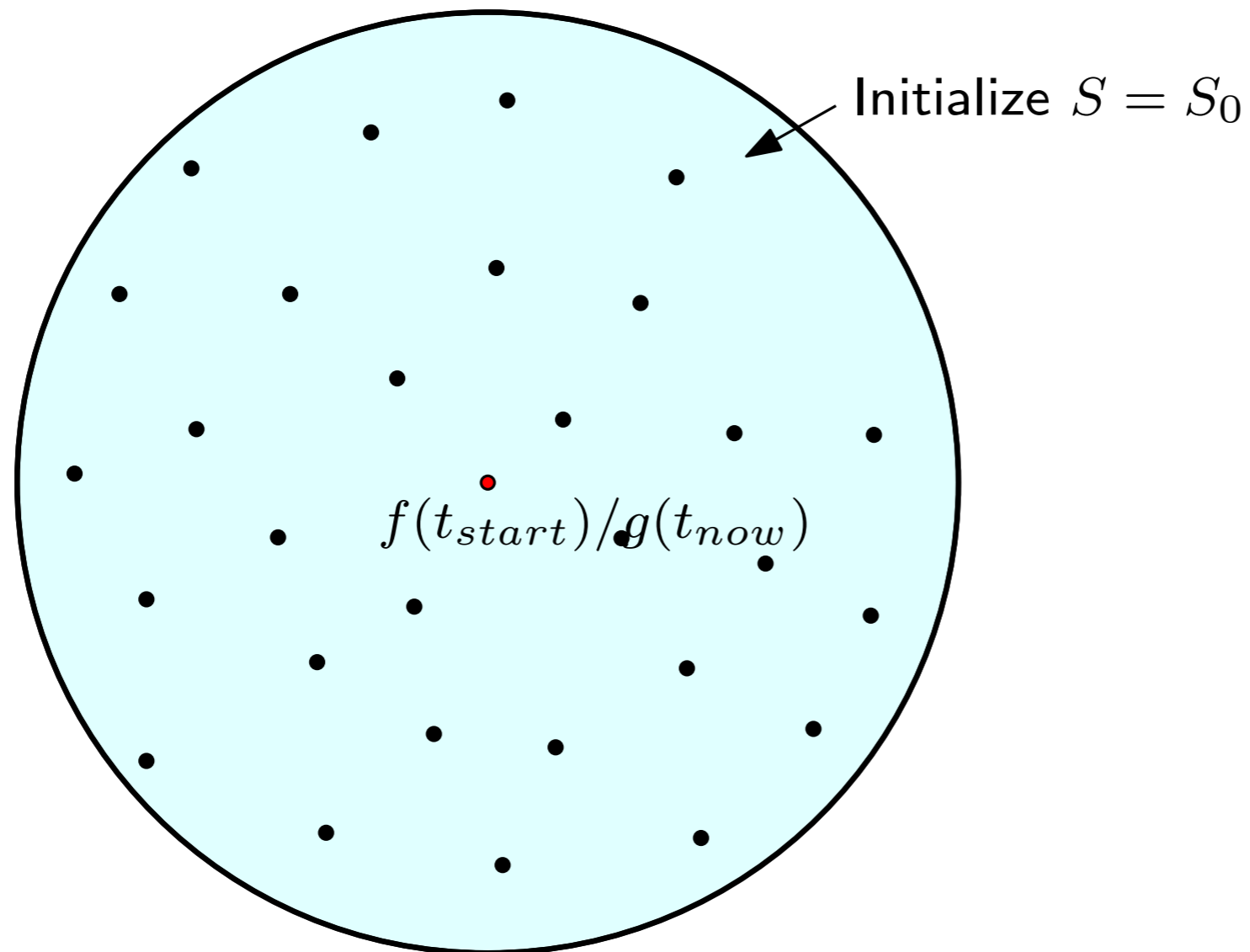
- The general idea follows from 1D

Divide the whole tracking period into **rounds**, and show that **the competitive ratio in each round is k** .

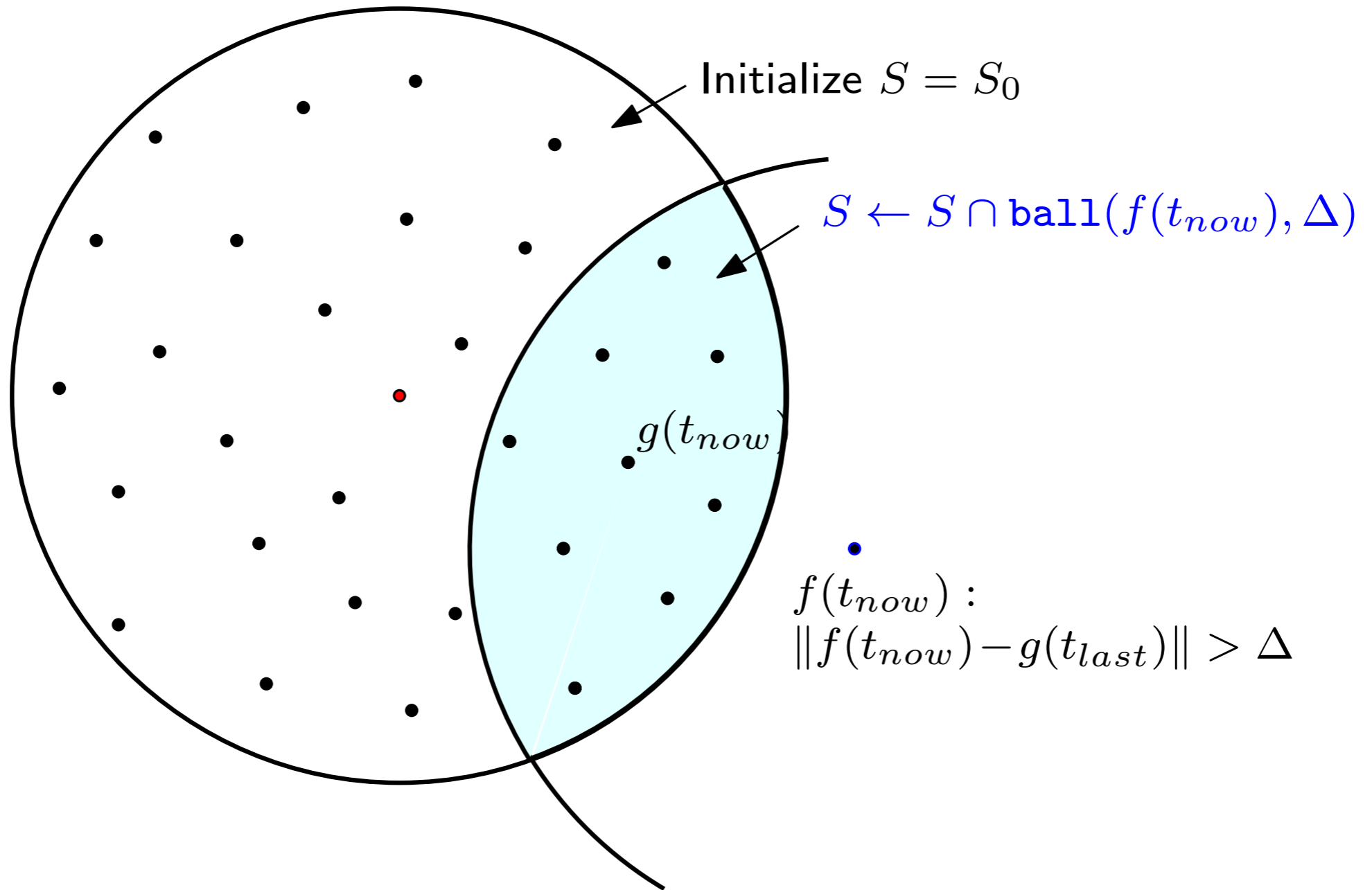
- The framework of one round

1. At time $t = t_{start}$, initialize a set $S = S_0$. (Many choices of S_0)
2. In each iteration in the while loop, we first **pick a “median”** from S as $g(t_{now})$ and send it to Bob.
3. When f deviates from $g(t_{last})$ by more than Δ , we cut S as $S \leftarrow S \cap \text{ball}(f(t_{now}), \Delta)$.
4. When S becomes empty, we can terminate the round.

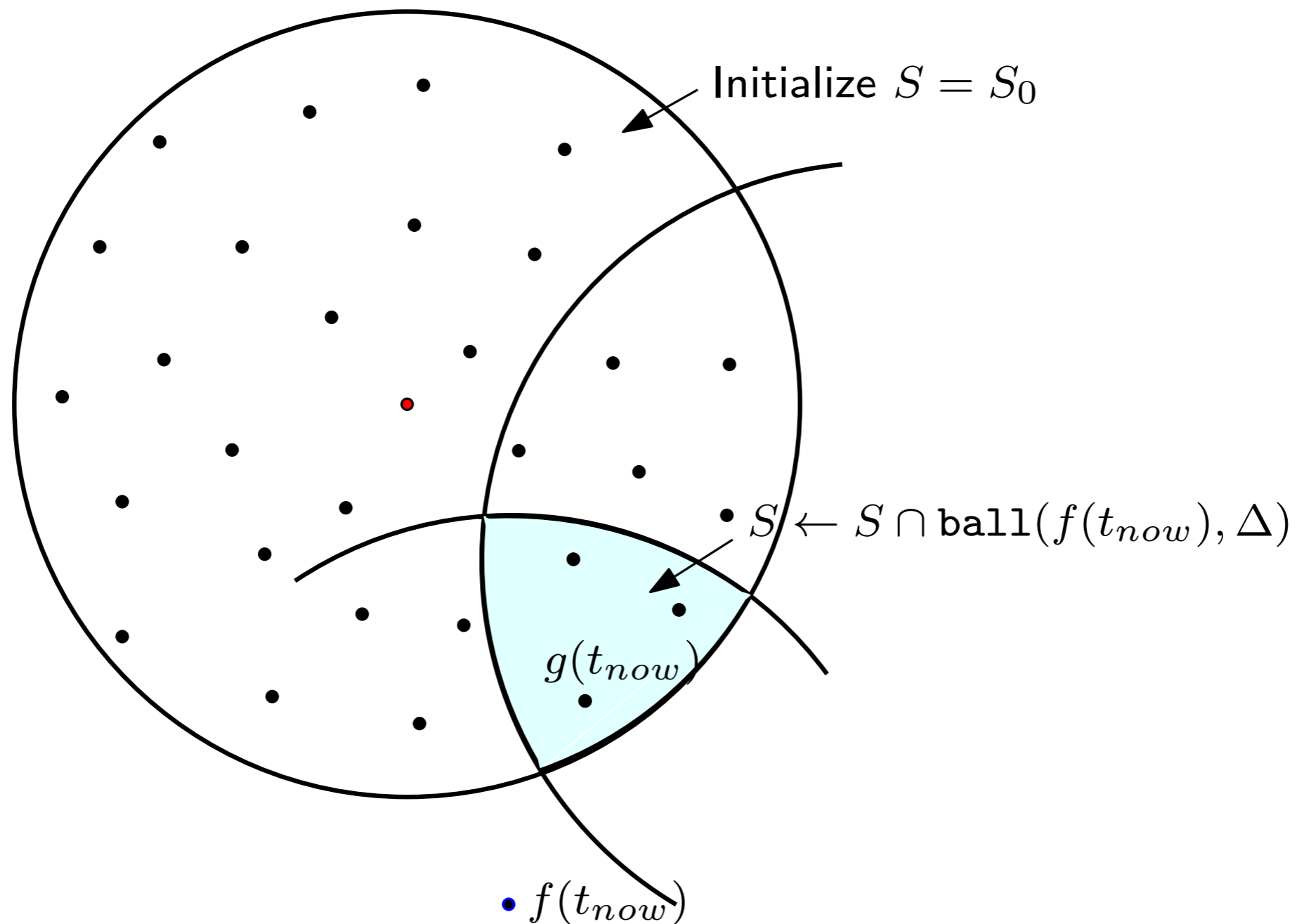
High Dimensions (cont.)



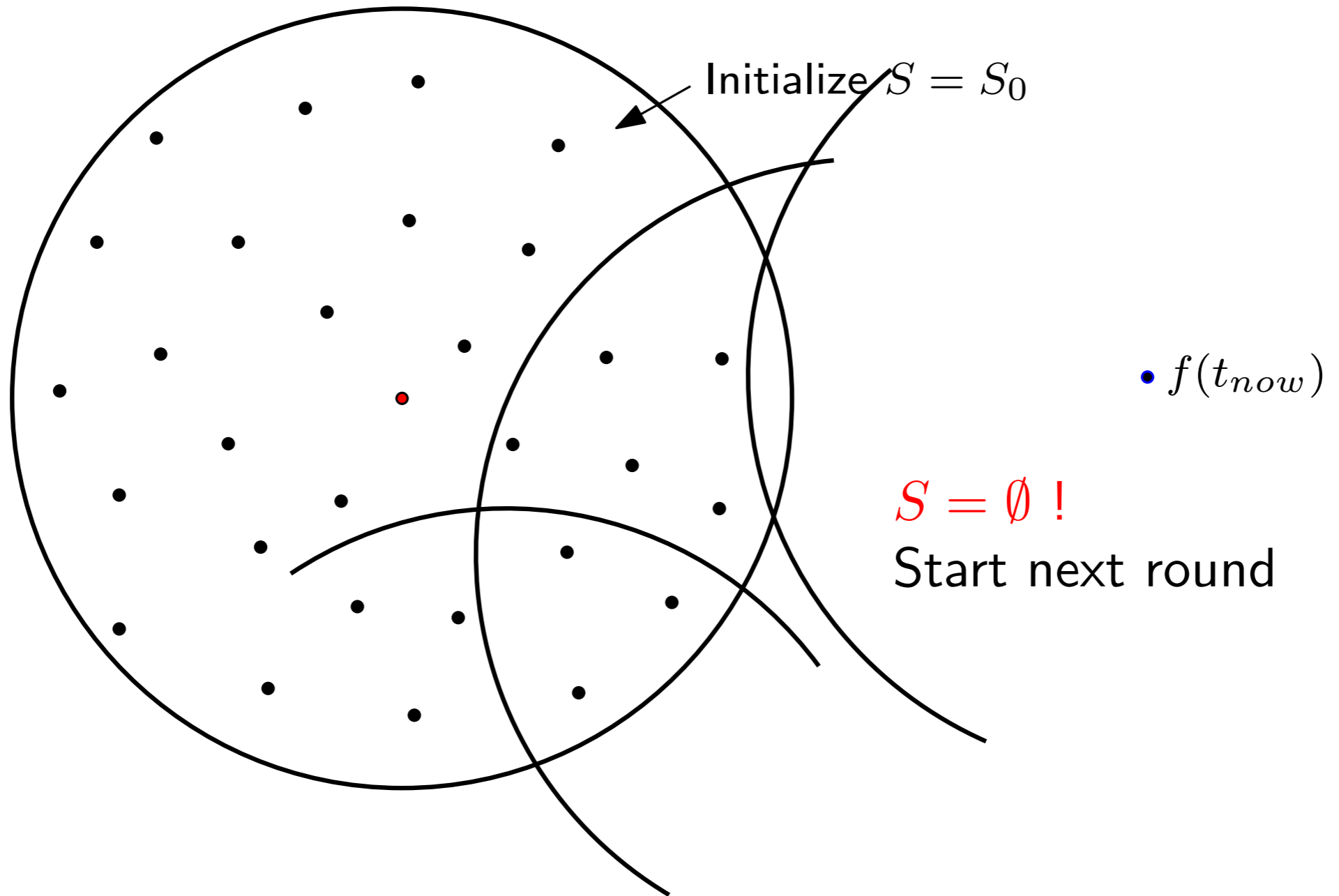
High Dimensions (cont.)



High Dimensions (cont.)



High Dimensions (cont.)





High Dimensions (cont.)

- Choose a **good set** S_0 as the starting point set S .
- Send the (approximate) **centroid** of a **convex set** containing S .

A round ends after $O(d^2 \log(d\Delta))$ messages.

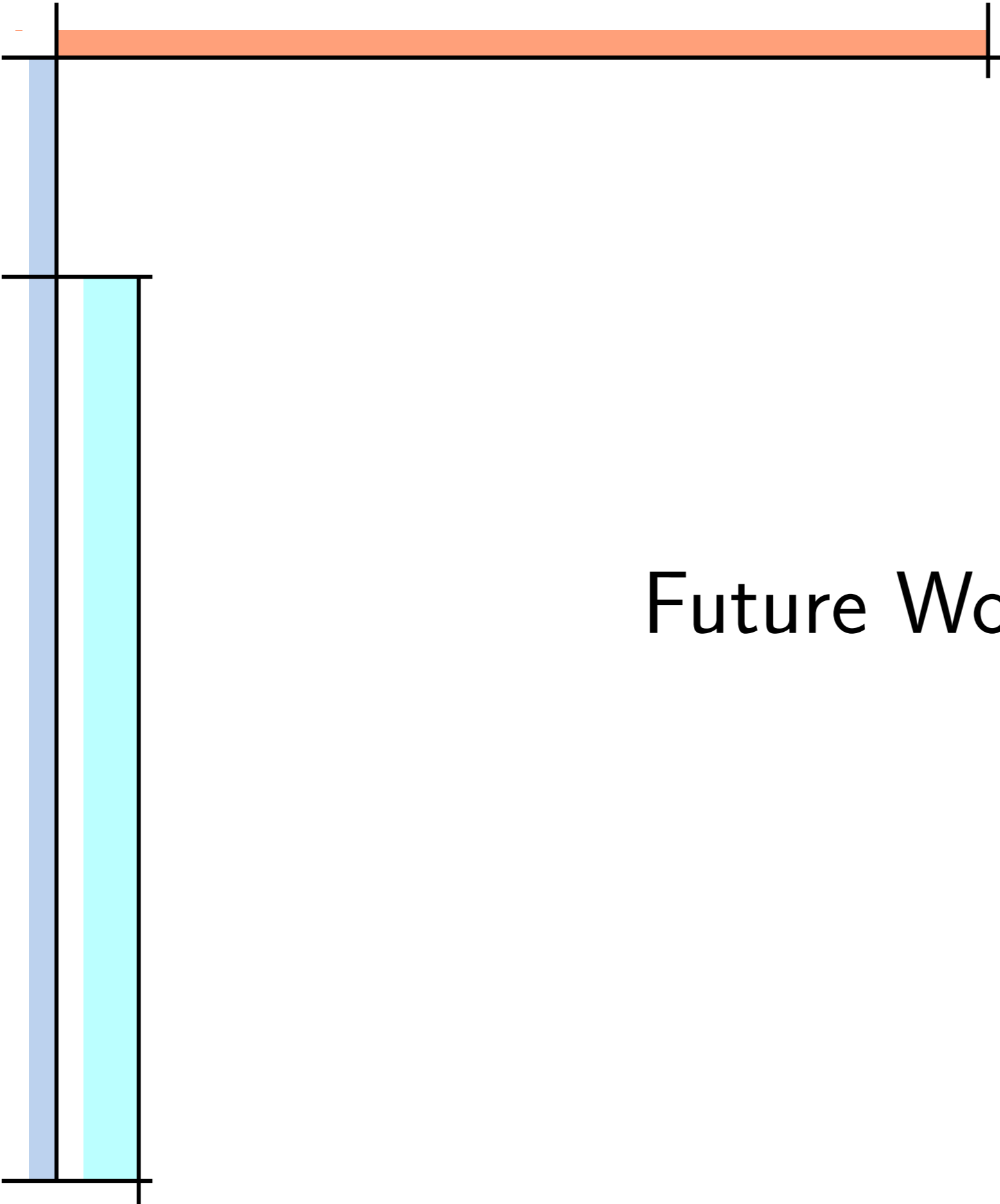


High Dimensions (cont.)

- ▣ Choose a **good set** S_0 as the starting point set S .
- ▣ Send the (approximate) **centroid** of a **convex set** containing S .

A round ends after $O(d^2 \log(d\Delta))$ messages.

Techniques are similar to Khachiyan's **Ellipsoid method**.



Future Work



Future Work

- Randomized algorithms (insertions only)
 - Tracking sum: we have $(k + \sqrt{k}/\epsilon) \log n$, can be better?
 - How about heavy hitters and quantiles?
 - How about lower bounds? Current best: $\Omega(k)$

Future Work

- Randomized algorithms (insertions only)
 - Tracking sum: we have $(k + \sqrt{k}/\epsilon) \log n$, can be better?
 - How about heavy hitters and quantiles?
 - How about lower bounds? Current best: $\Omega(k)$
- Deletions in multiple sites
 - Tracking sum: require new models?
 - Sequence/time-based sliding windows for heavy hitters and quantiles?

Things Done in the PhD Study

□ Data structures

- *Cache-Oblivious Hashing*, with Pagh, Wei & Yi, PODS '10.
- *The Limits of Buffering: A Tight Lower Bound for Dynamic Membership in the External Memory Model*, with Verbin, STOC '10.
- *On the Cell Probe Complexity of Dynamic Membership*, with Yi, SODA '10.
- *Dynamic External Hashing: The Limit of Buffering*, with Wei & Yi, SPAA '09. Journal version (combined with another paper by Yi) submitted to JACM.

□ Distributed data streams

- *Optimal Sampling From Distributed Streams*, with Cormode, Muthukrishnan, & Yi, PODS '10.
- *Optimal Tracking of Distributed Heavy Hitters & Quantiles*, with Yi, PODS '09. Journal version submitted to TALG.
- *Multi-Dimensional Online Tracking*, with Yi, SODA '09. Journal: TALG minor revision.

□ Others

- *Clustering with Diversity*, with Li & Yi, ICALP '10.
- *Revenue Generation for Truthful Spectrum Auction in Dynamic Spectrum Access*, with Jia, Zhang & Liu, MobiHoc '09.
- *Finding Frequent Items in Probabilistic Data*, with Li & Yi, SIGMOD '08.
- *The Art of Metric Embeddings – A technique oriented approach*, PQE 2007

Things Done in the PhD Study

□ Data structures

- *Cache-Oblivious Hashing*, with Pagh, Wei & Yi, PODS '10.
- *The Limits of Buffering: A Tight Lower Bound for Dynamic Membership in the External Memory Model*, with Verbin, STOC '10.
- *On the Cell Probe Complexity of Dynamic Membership*, with Yi, SODA '10.
- *Dynamic External Hashing: The Limit of Buffering*, with Wei & Yi, SPAA '09. Journal version (combined with another paper by Yi) submitted to JACM.

□ Distributed data streams

This Thesis

- *Optimal Sampling From Distributed Streams*, with Cormode, Muthukrishnan, & Yi, PODS '10.
- *Optimal Tracking of Distributed Heavy Hitters & Quantiles*, with Yi, PODS '09. Journal version submitted to TALG.
- *Multi-Dimensional Online Tracking*, with Yi, SODA '09. Journal: TALG minor revision.

□ Others

- *Clustering with Diversity*, with Li & Yi, ICALP '10.
- *Revenue Generation for Truthful Spectrum Auction in Dynamic Spectrum Access*, with Jia, Zhang & Liu, MobiHoc '09.
- *Finding Frequent Items in Probabilistic Data*, with Li & Yi, SIGMOD '08.
- *The Art of Metric Embeddings – A technique oriented approach*, PQE 2007



The End

THANK YOU

Q and A