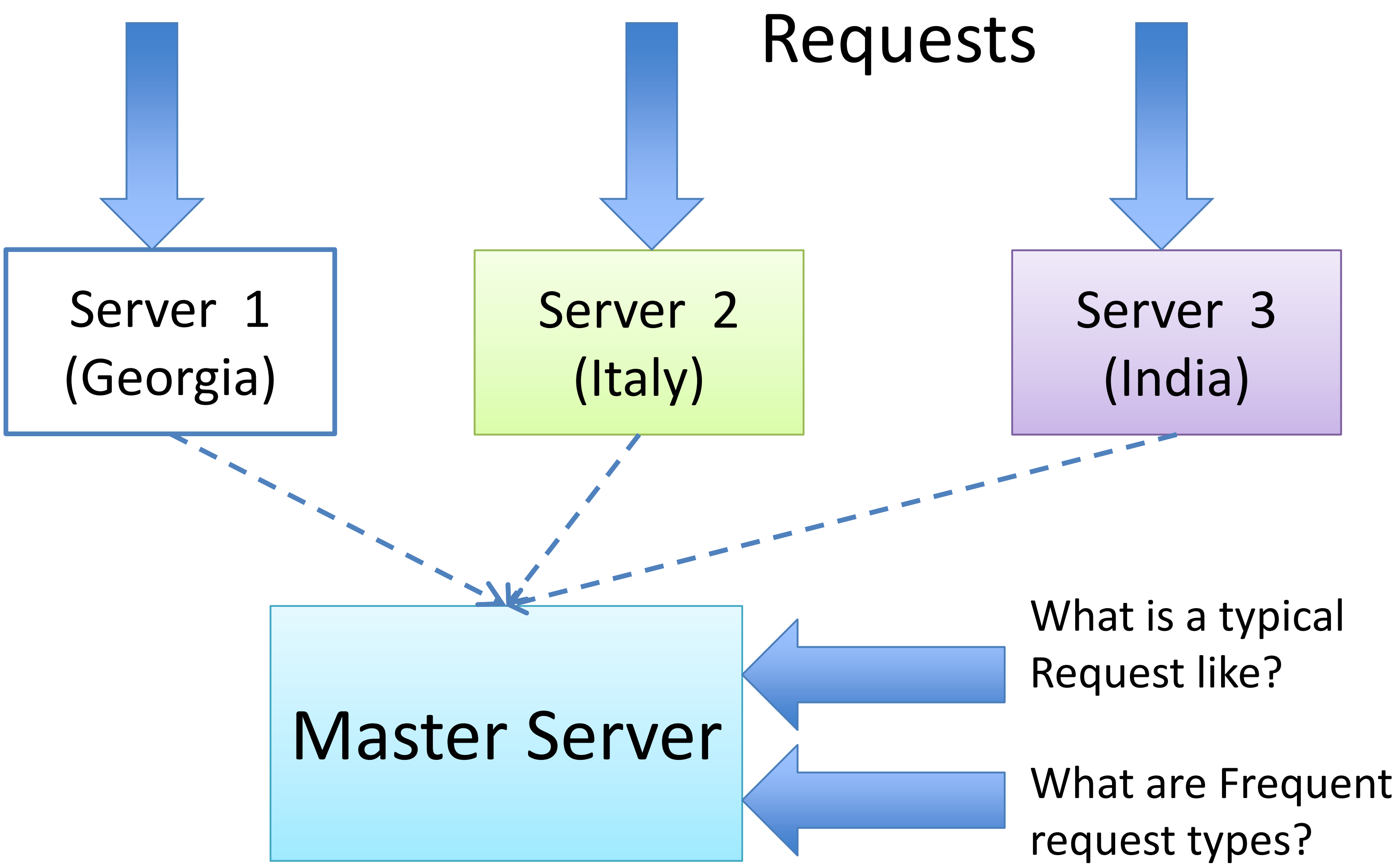
Optimal Sampling from Distributed Streams Revisited

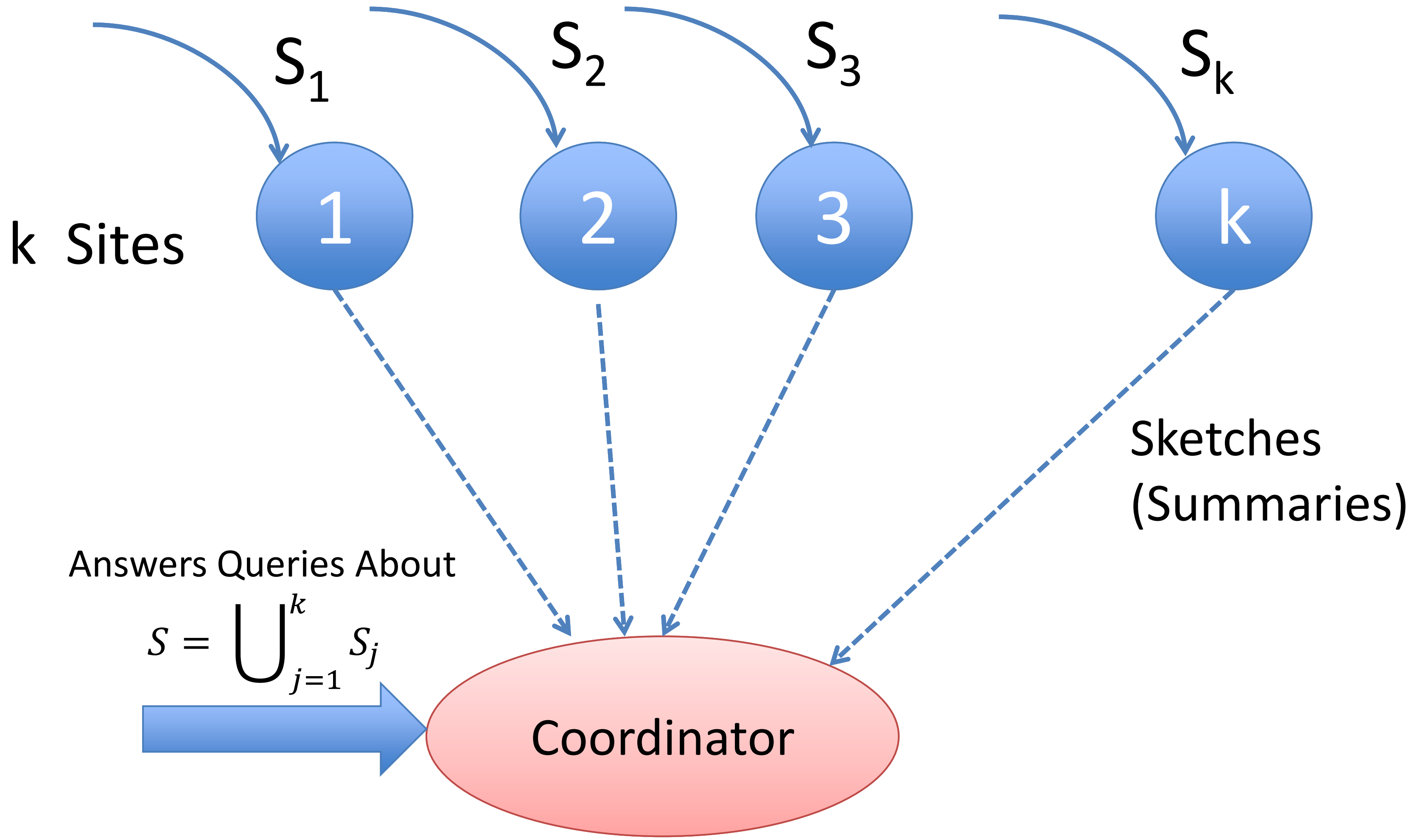
Srikanta Tirthapura (Iowa State University)
David Woodruff (IBM Almaden)

Presentation at DISC 2011

Distributed Streams



Distributed Streams



Continuous Distributed Streaming Model

- Multiple geographically distributed streams
  - Data is a sequence of updates

- Task: A central coordinator continuously maintains a global property over the union of all streams

- Cost Metric: Number of messages transmitted

Problem Definition (1)

- k sites numbered 1,2,3,…,k
- At any point in time, site i has observed stream S_i

$$S = \bigcup_{1}^{k} S_i$$

- Task: At all times, the central coordinator must maintain a random sample of size s from S

Problem Definition (2)

- Synchronous Model
  - Execution proceeds in rounds
  - In each round, each site observes one or more items, and can send a message, receive a response

- Only Site <---> Coordinator communication
  - does not lose generality

- Cost Metric: Total number of messages sent by the protocol over the entire execution of observing n elements

Random Sampling

Given a data set *P* of size *n*, a random sample *S* is defined as the result of a process.

1. Sample Without Replacement of Size *s* ($1 \leq s \leq n$)

   Repeat *s* times

   1. $e \leftarrow$ {a randomly chosen element from *P*}
   2. $P \leftarrow P - \{e\}$
   3. $S \leftarrow S \cup \{e\}$

2. Sample With Replacement of size *s* ($1 \leq s$)

   Repeat *s* times

   1. $e \leftarrow$ {a randomly chosen element from *P*}
   2. $S \leftarrow S \cup \{e\}$

Our Results: Upper Bound

- An algorithm for continuously maintaining a random sample of S with message complexity.

$$O\left(\frac{k \log \frac{n}{s}}{\log\left(1+\frac{k}{s}\right)}\right)$$

- k = number of sites
  n = Total size of stream
  s = desired sample size

Our Results: Matching Lower Bound

- Any algorithm for continuously maintaining a random sample of S must have message complexity:

$$\Omega\left(\frac{k \log \frac{n}{s}}{\log\left(1+\frac{k}{s}\right)}\right)$$

- k = number of sites
  n = Total size of stream
  s = desired sample size

Prior Work

- Single Stream: Reservoir Sampling Algorithm
  – Waterman (1960s)
  – Vitter: *Random sampling with a reservoir*. ACM Transactions on Mathematical Software, 11(1):37–57, 1985.

- Random Sampling on Distributed Streams
  – Cormode, Muthukrishnan, Yi, and Zhang: *Optimal sampling from distributed streams*. ACM PODS, pages 77–86, 2010

Related Work

- "Reactive" Distributed Streams:
  - Gibbons and Tirthapura, *Distributed streams algorithms for sliding windows*, SPAA 2002, pages 63-72
  - Coordinator can contact the sites during query processing

- Frequency Moments, Distinct Elements in Distributed Streams
  - Cormode, Muthukrishnan, and Yi. Algorithms for distributed functional monitoring. SODA, pages 1076–1085, 2008
  - Introduced the continuous distributed streaming model

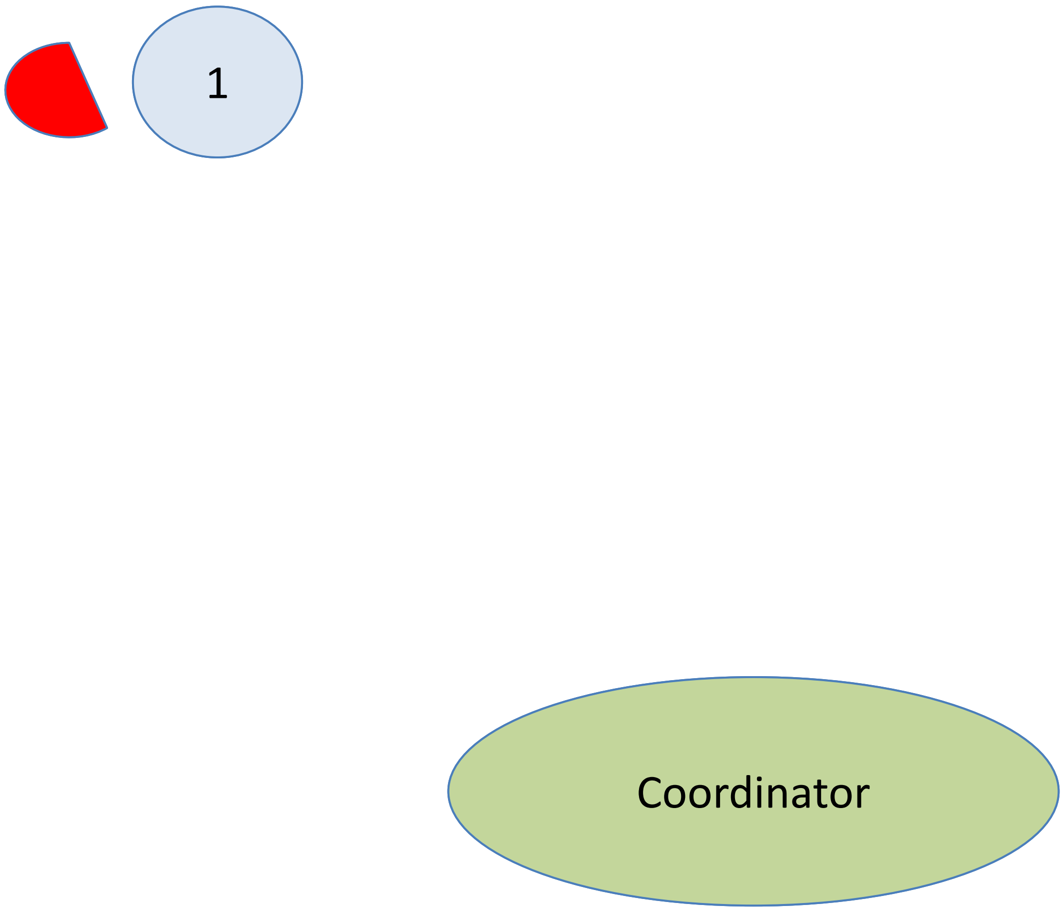
- Entropy on Distributed Streams
  - Arackaparambil, Brody, and Chakrabarti. Functional monitoring without monotonicity. ICALP (1), pages 95–106, 2009
  - Study non-monotonic functions, unlike [Cormode et al. 2008]

Prior Work

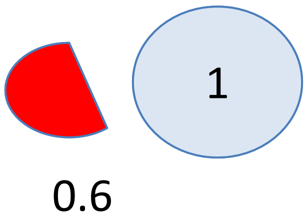
k = number of sites
n = Total size of streams
s = desired sample size

	Upper Bound		Lower Bound	
	Our Result	Cormode et al.	Our Result	Cormode et al.
$s < k/8$	$O\left(\frac{k \log(n/s)}{\log(k/s)}\right)$	$O(k \log n)$	$O\left(\frac{k \log(n/s)}{\log(k/s)}\right)$	$\Omega(k + s \log n)$
$s \geq k/8$	$O(s \log (n/s))$	$O(s \log n)$	$\Omega(s \log (n/s))$	$\Omega(s \log (n/s))$

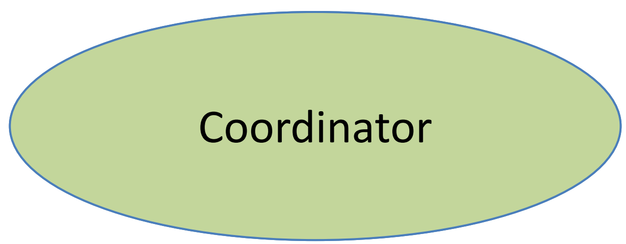
Algorithm: Element arrives at 1



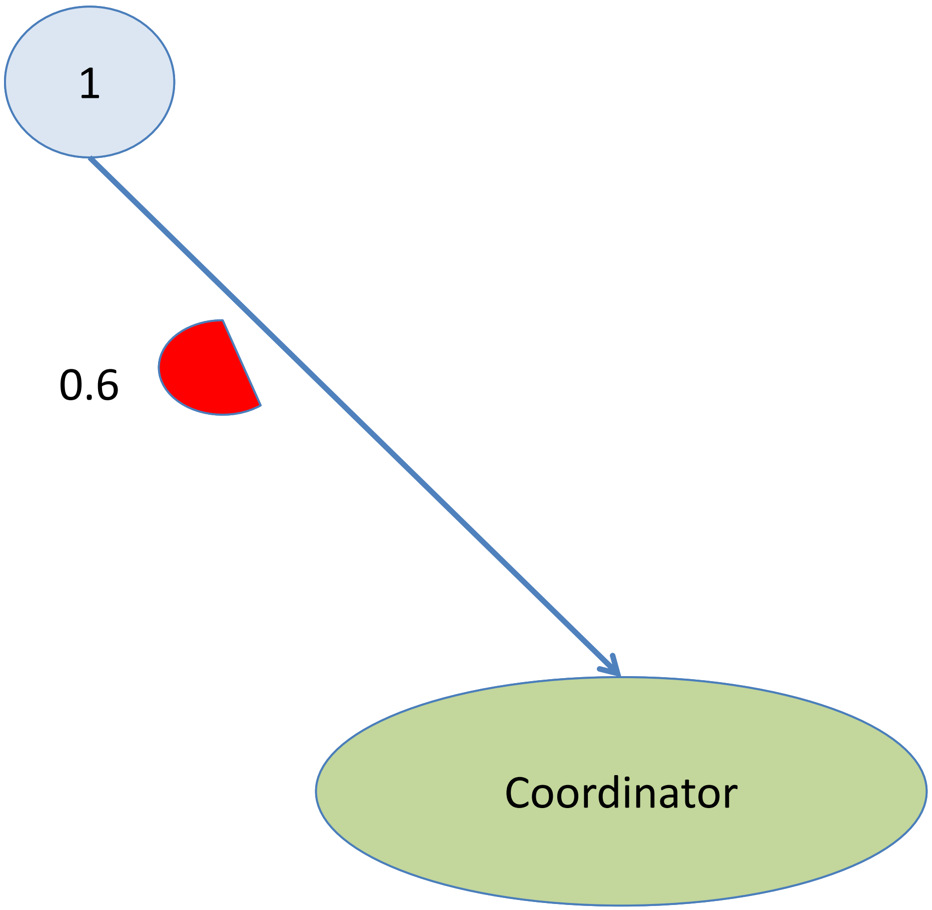
Weight for each element



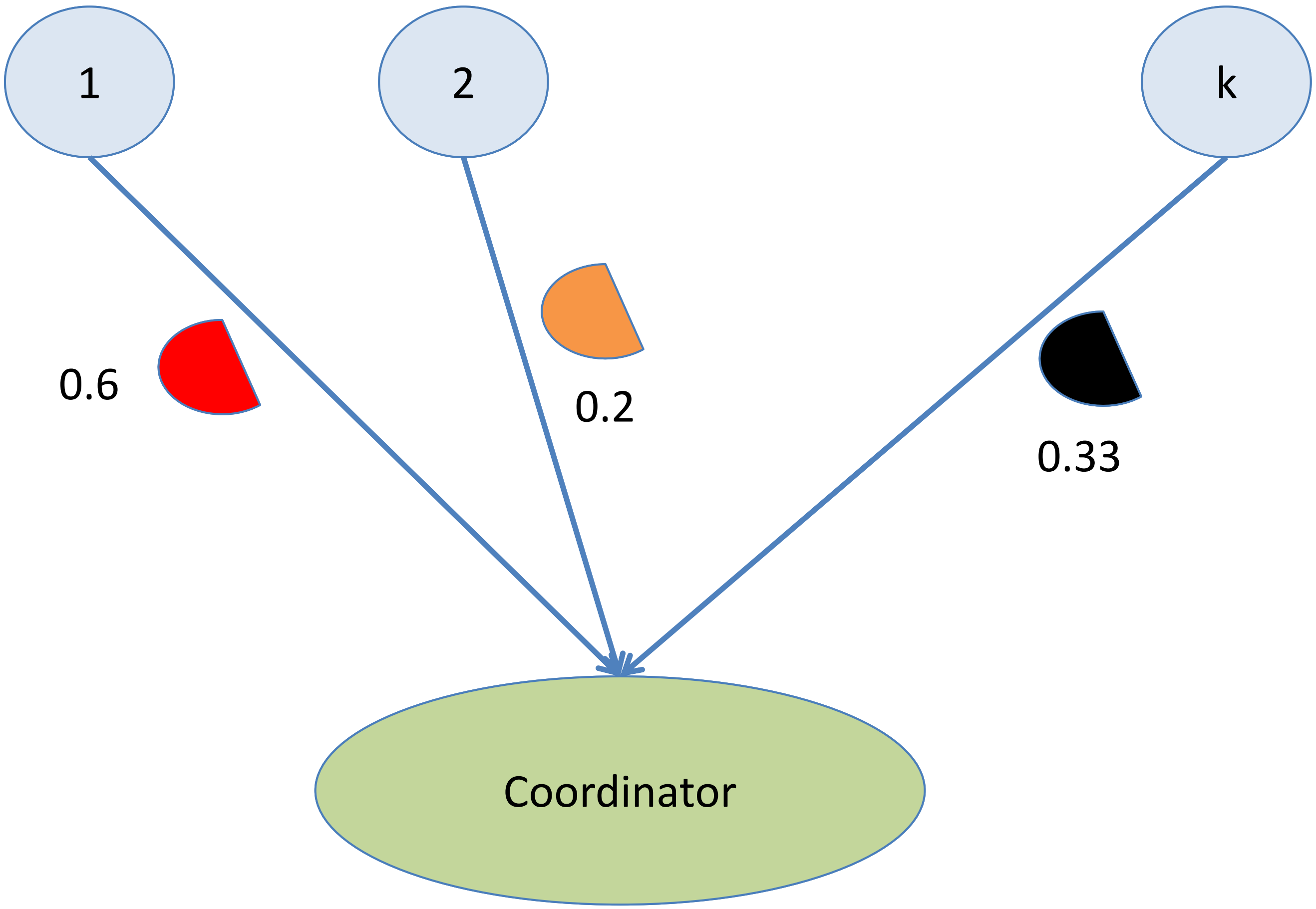
Weight of each element
= random number in [0,1]



Weight for each element



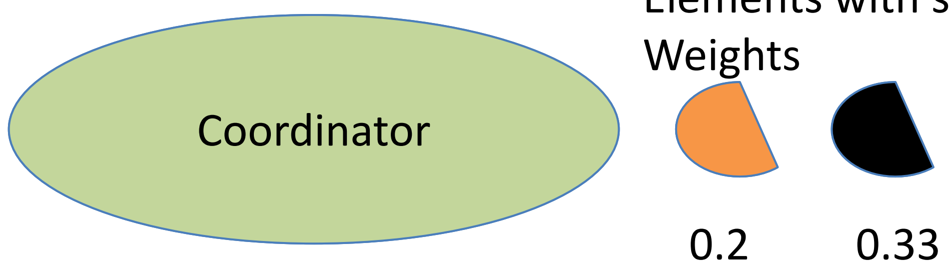
Algorithm



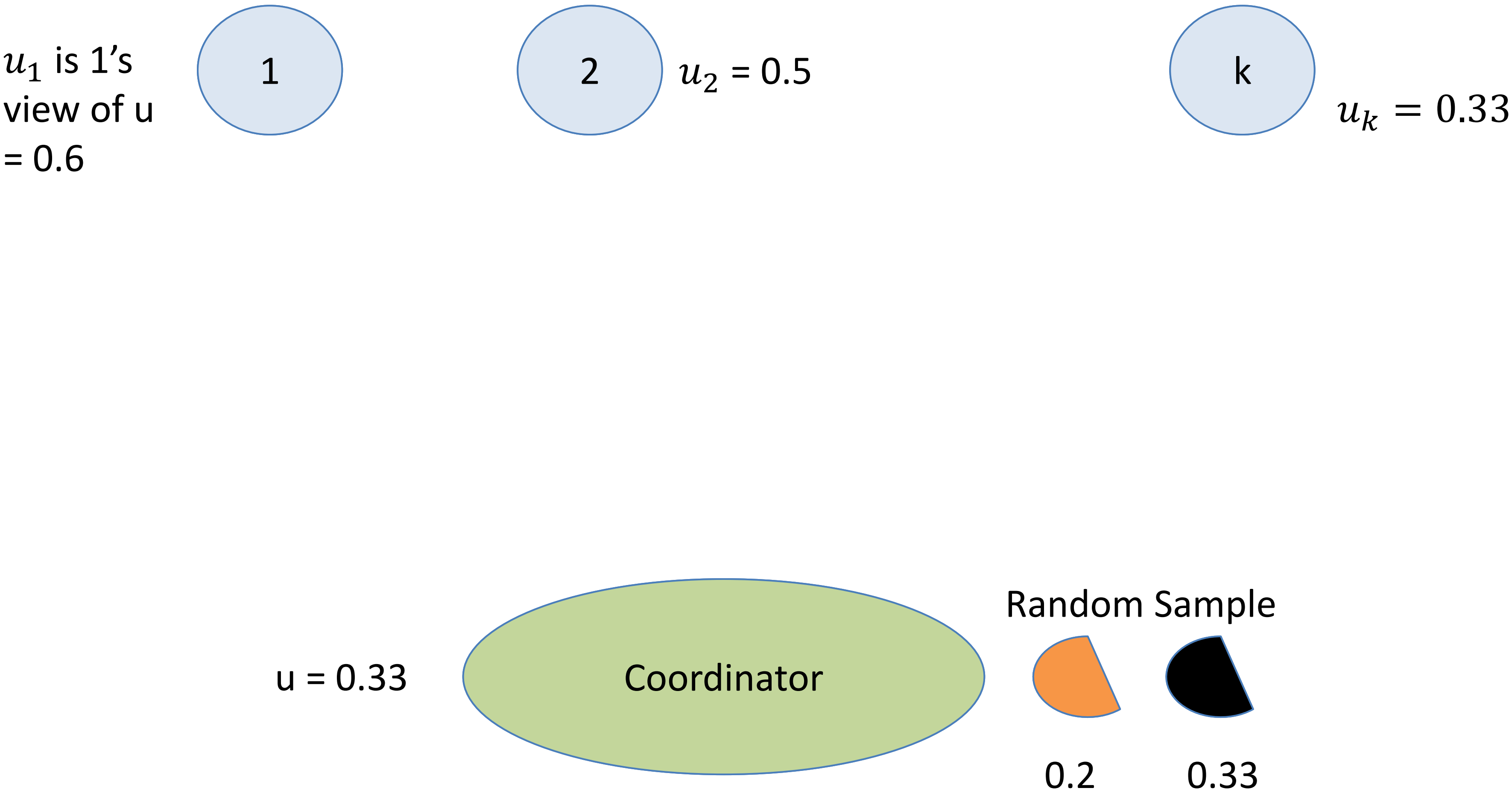
Algorithm: Random Sample

Random Sample = set of Elements with s smallest Weights

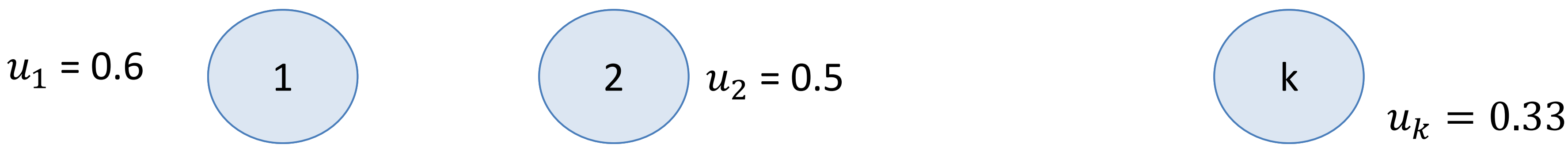
u = 0.33
s-th smallest weight seen so far



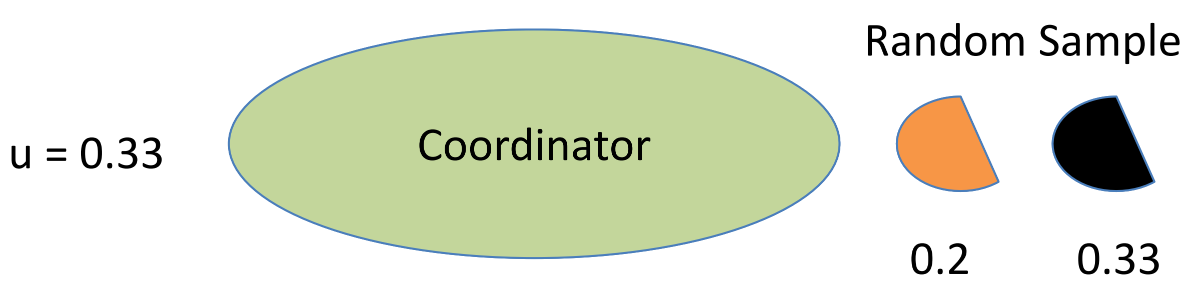
Algorithm: Sites "Cache" value of u



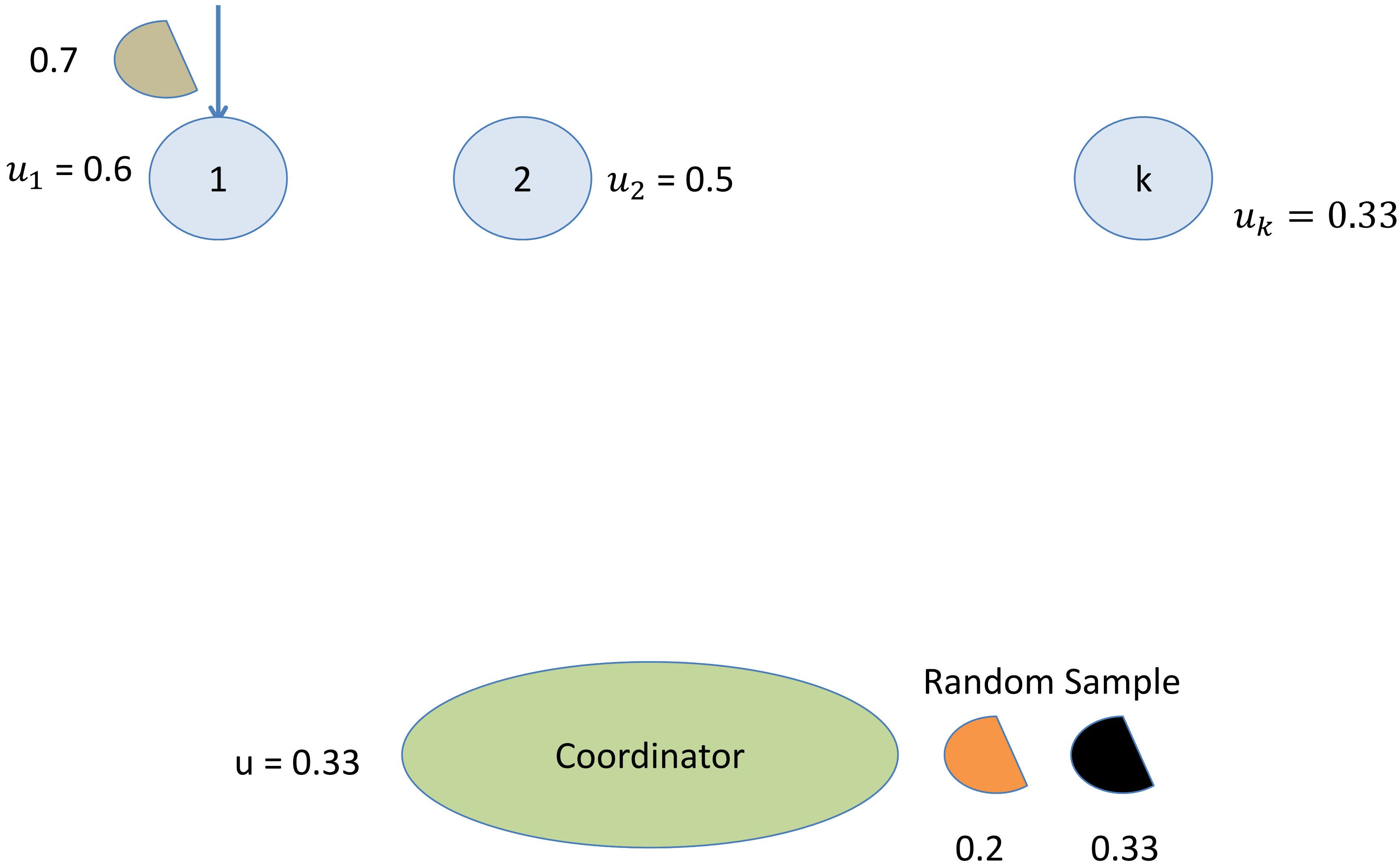
Algorithm: Effect of Caching



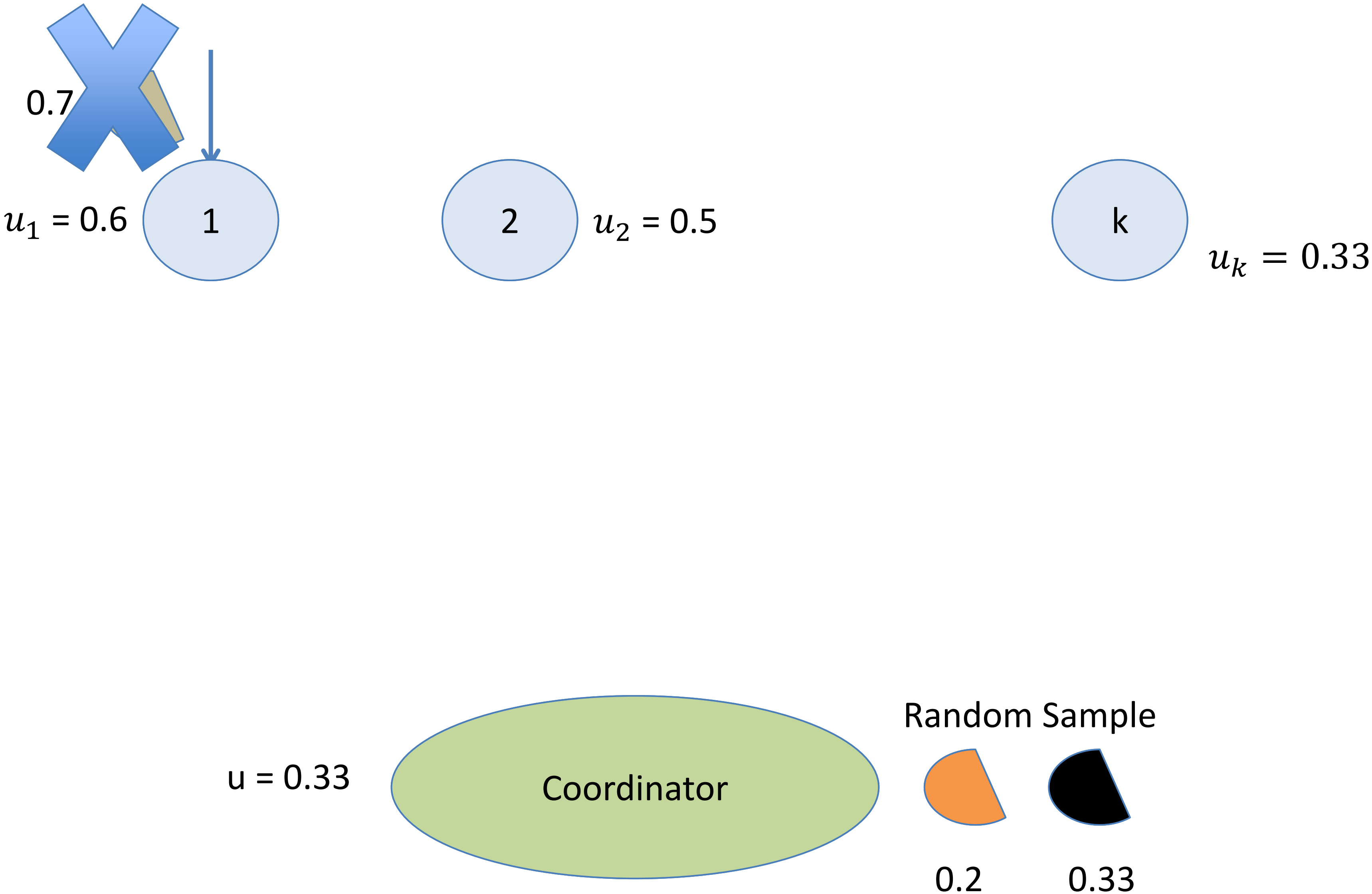
$u_1, u_2, \ldots, .$ are all
at least *u*
So, elements that belong to
The sample are definitely sent



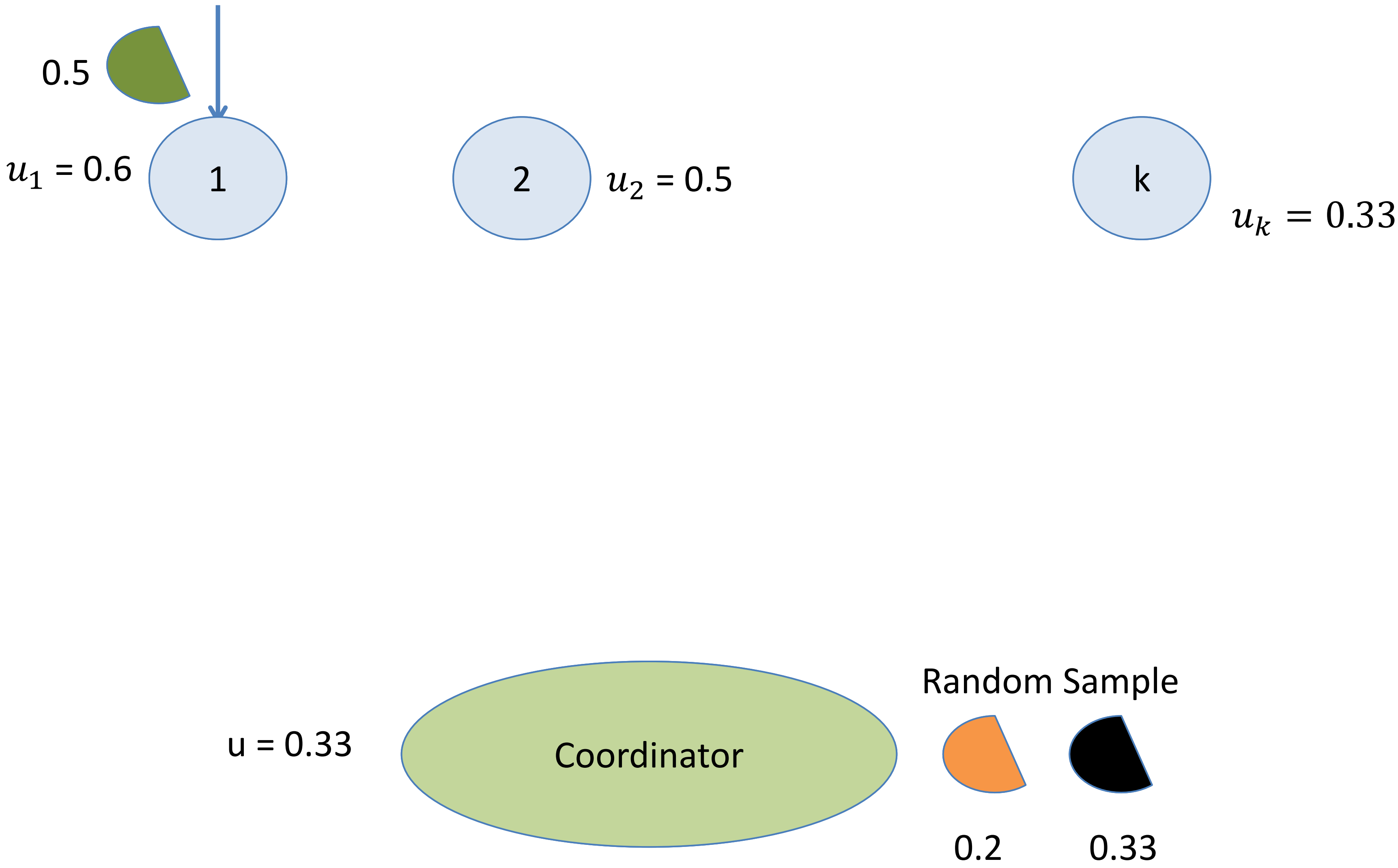
Element at 1


0.7
u_1 = 0.6
1
2
u_2 = 0.5
k
$u_k = 0.33$
Random Sample
u = 0.33
Coordinator
0.2
0.33

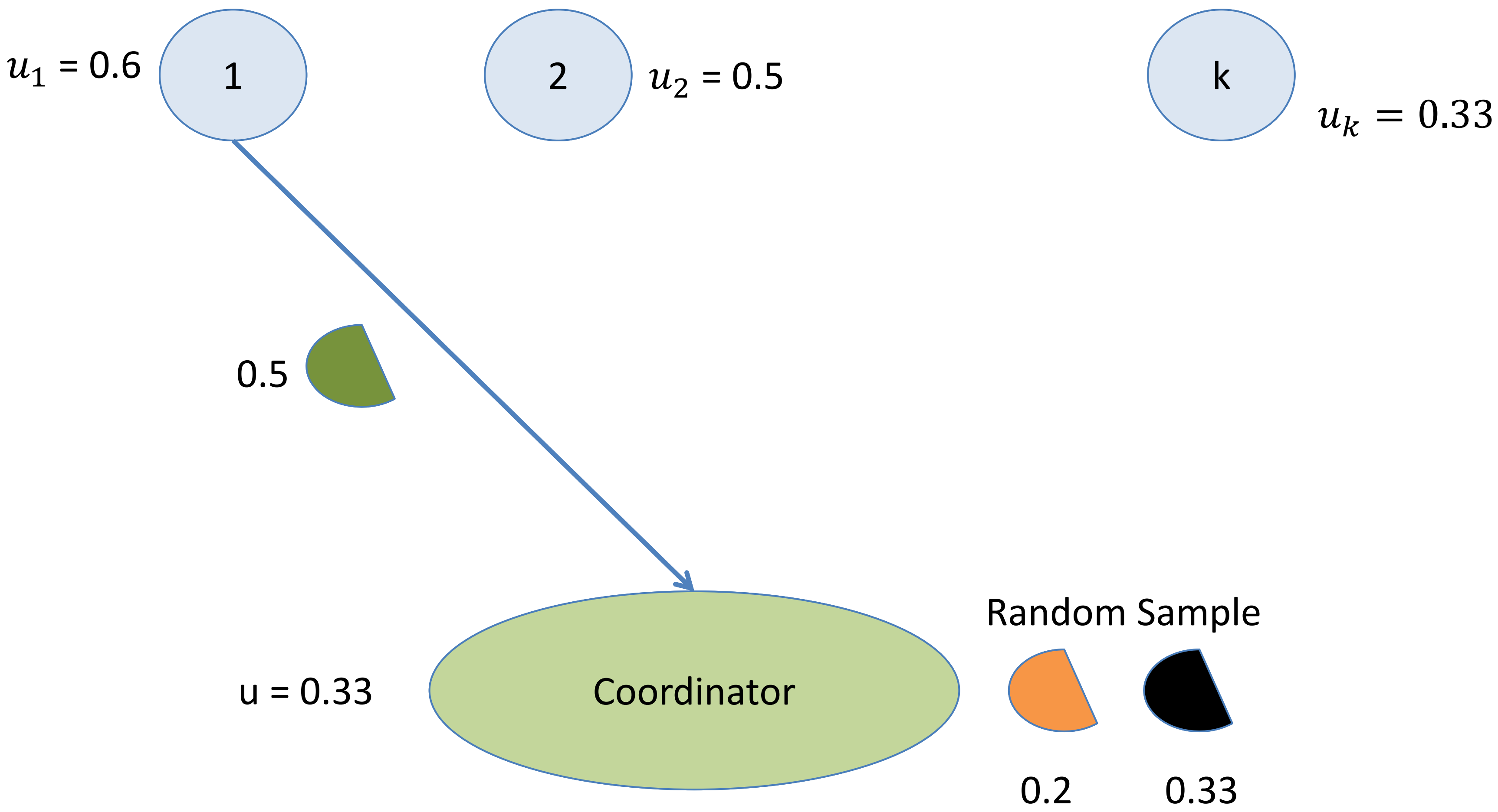
Discarded Locally



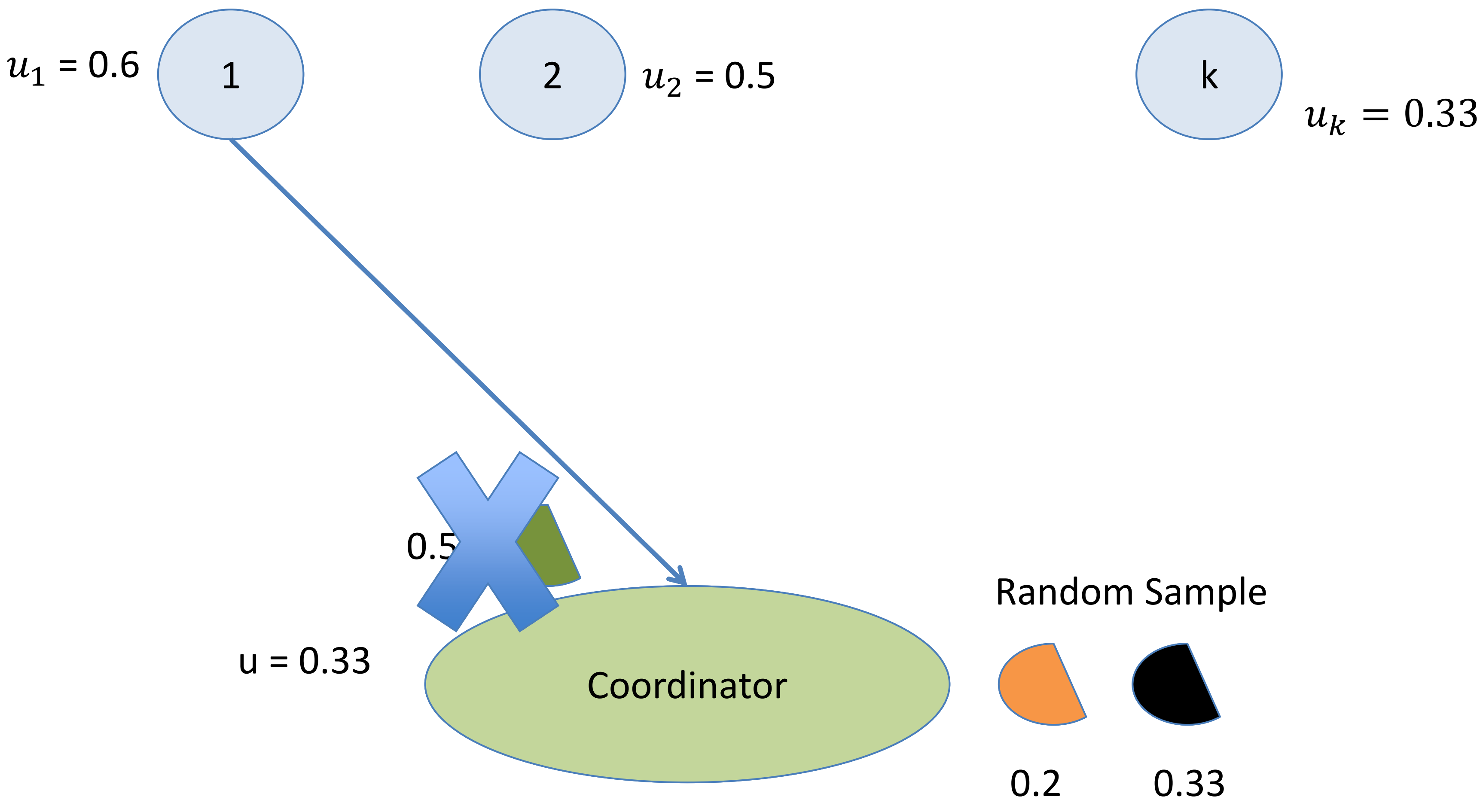
Element at 1


0.5
u_1 = 0.6
1
2
u_2 = 0.5
k
$u_k = 0.33$
u = 0.33
Coordinator
Random Sample
0.2
0.33

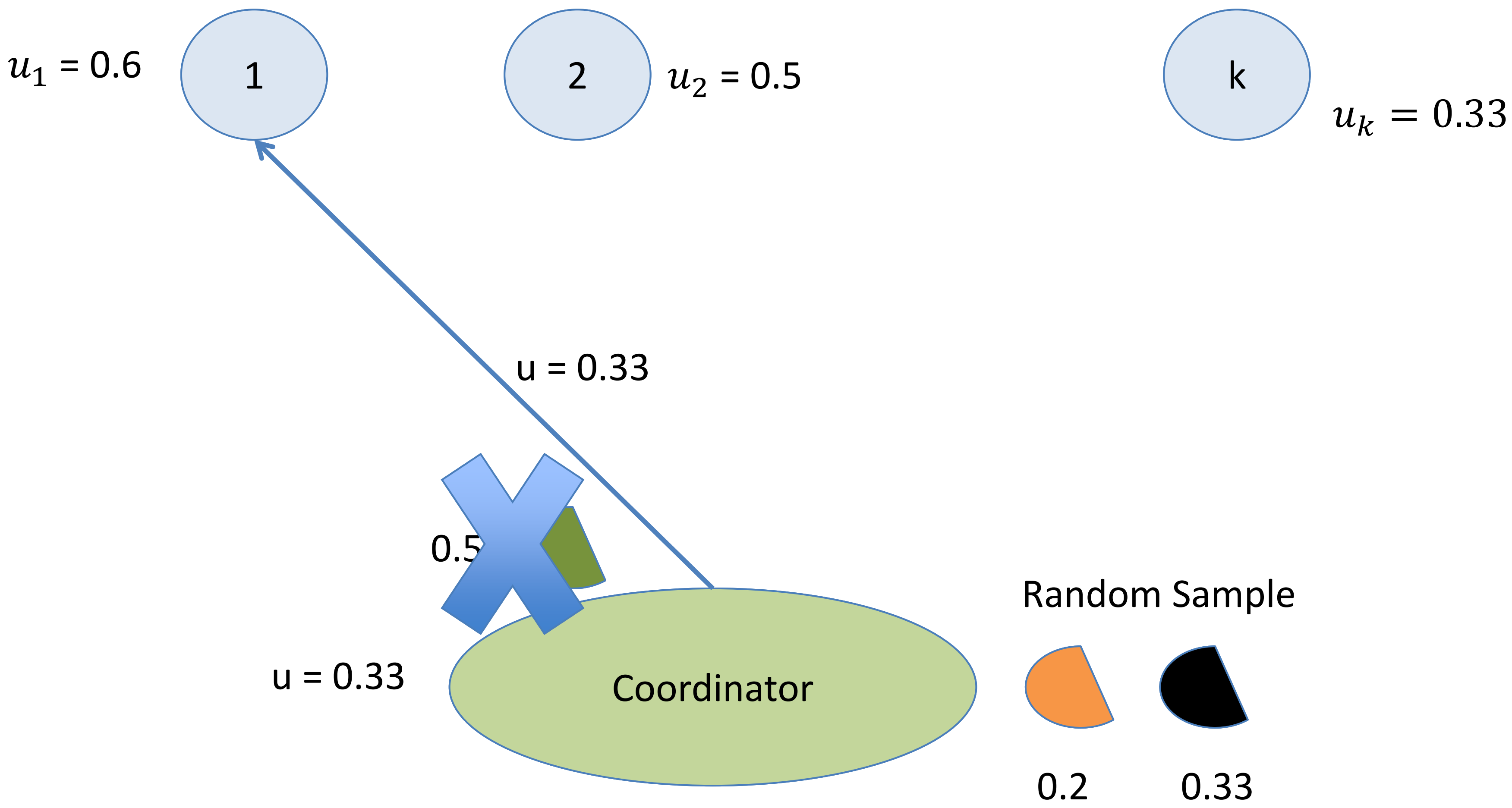
“Wasteful” Send



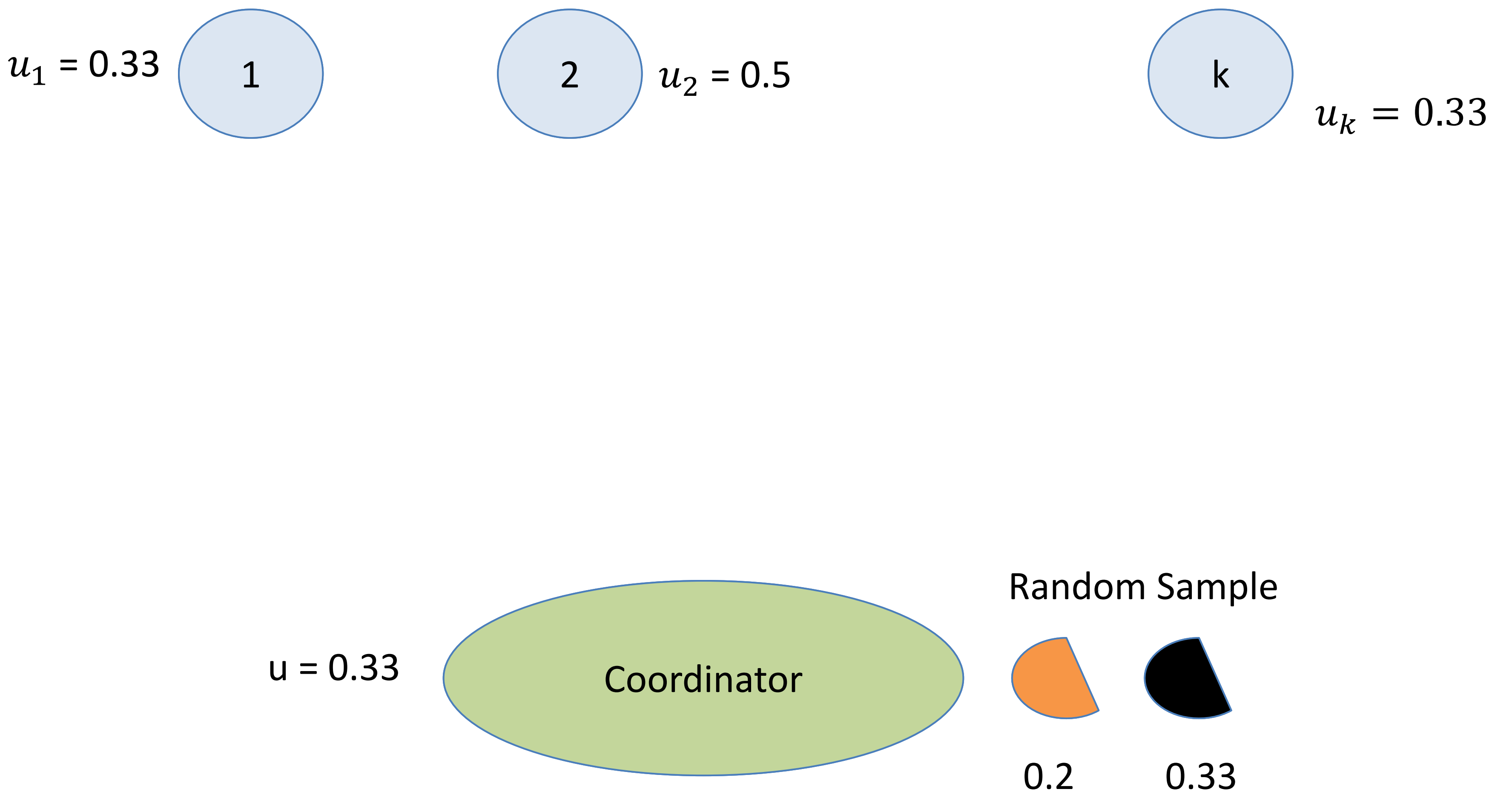
Discarded by Coordinator



But: Coordinator Refreshes Site's View



Site's View is Refreshed



Algorithm Notes

- A message from site to coordinator either
  - Changes the coordinator’s state
  - Or Refreshes the client’s view

Algorithm at Site *i* when it receives element *e*

// u_i is i's view of the minimum weight so far in the system

// u_i is initialized to ∞

1. Let *w(e)* be a random number between 0 and 1

2. If $(w(e) < u_i)$ then
    1. Send *(e,w(e))* to the coordinator, and receive u' in return
    2. $u_i \leftarrow u'$

Algorithm at Coordinator

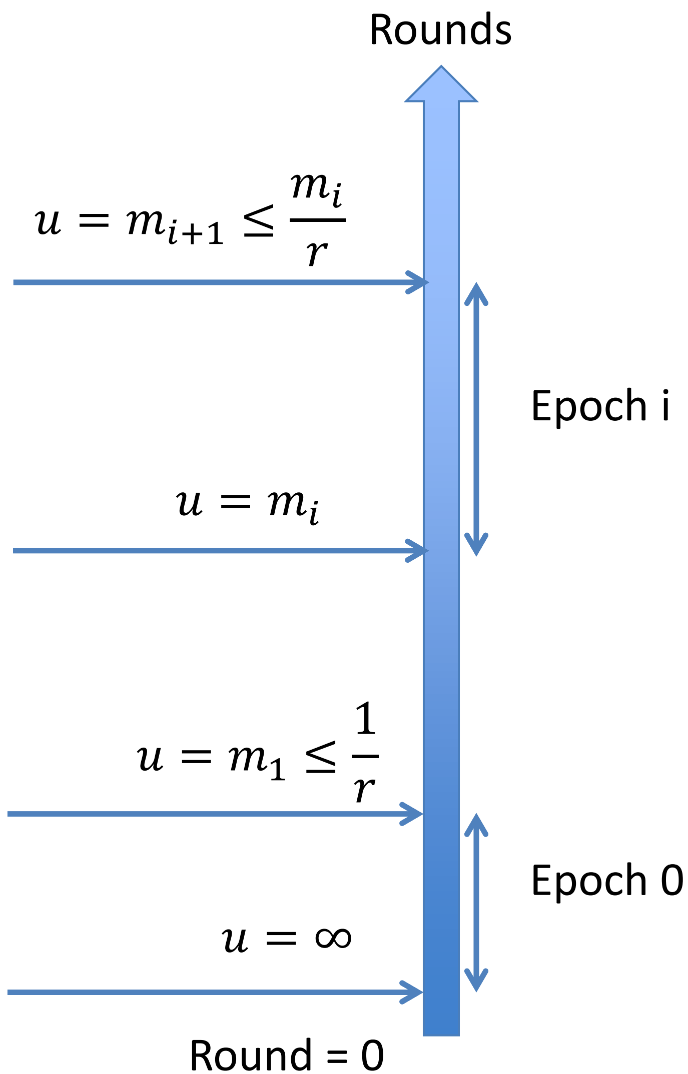
1. Coordinator maintains *u*, the *s-th* smallest weight seen in the system so far

2. If it receives a message (e,w(e)) from site i,
   1. If (u > w(e)), then update u and add e to the sample
   2. Send u back to i

Analysis: High Level View

- An execution divided into a few “Epochs”
- Bound the number of epochs
- Bound the number of messages per epoch

Analysis: Epochs



u is the s-th smallest weight seen in the system, so far.

- Epoch 0: all rounds until u is 1/r or smaller
- Epoch i: all rounds after epoch (i-1) till u has further reduced by a factor r
- Epochs are not known by the algorithm, only used for analysis

Bound on Number of Epochs

Let ξ denote the number of epochs in an execution

Lemma: $\mathrm{E}[\xi] \leq \left(\frac{\log\left(\frac{n}{s}\right)}{\log r}\right) + 2$

n = stream size
s = desired sample size
r = a parameter

Proof: $E[\xi] = \sum_{i \geq 0} \Pr[\xi \geq i]$

At the end of *i* epochs, $u \leq \frac{1}{r^i}$

At the end of $\left(\frac{\log\left(\frac{n}{s}\right)}{\log r}\right) + \mathrm{j}$ epochs, $u \leq \left(\frac{s}{n}\right)\frac{1}{r^j}$

We can show using Markov rule, $\Pr\left[\xi \geq \left(\frac{\log\left(\frac{n}{s}\right)}{\log r}\right) + \mathrm{j}\right] \leq \frac{1}{r^j}$

Algorithm B versus A

- Suppose our algorithm is "A". We define an algorithm "B" that is the same as A, except:
  - At the beginning of each epoch, coordinator broadcasts u (the current s-th minimum) to all sites
  - B easier to analyze since the states of all sites are synchronized at the beginning of each epoch

- Random sample maintained by "B" is the same as that maintained by A

- Lemma: The number of messages sent by A is no more than twice the number sent by B
  - Henceforth, we will analyze B

Analysis of B: Bound on Messages Per Epoch

- μ = total number of messages
- μ_j: number of messages in epoch j
- X_j: number messages sent to coordinator in epoch j
- ξ: number of epochs

- $\mu = \sum_{j=0}^{\xi-1} \mu_j$
- $\mu_j = k + 2X_j$
- $\mu = \xi k + 2\sum_{j=0}^{\xi-1} X_j$

Now, only need to bound X_j, the number of messages to coordinator in epoch j

Bound on X_j

- Lemma: For each epoch j, $E[X_j] \leq 1 + 2rs$

- Proof:
  - First compute $E[X_j]$ conditioned on n_j and m_j
  - Remove the conditioning on n_j (the number of elements in epoch j)
  - Remove the conditioning on m_j (the value of u at the beginning of epoch j)

Upper Bound

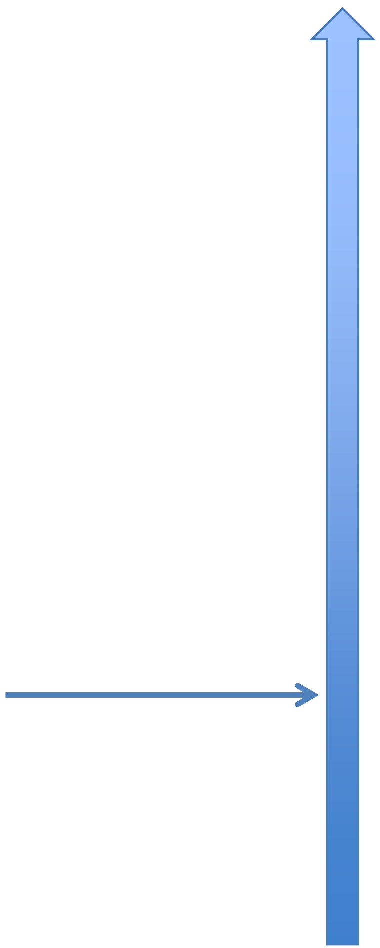
Theorem: The expected message complexity is as follows

- If $s \geq \frac{k}{8}$ then $E[\mu] = O\left(s \log\left(\frac{n}{s}\right)\right)$
- If $s < \frac{k}{8}$ then $E[\mu] = O\left(\frac{k \log\left(\frac{n}{s}\right)}{\log \frac{k}{s}}\right)$

k = number of sites
n = Total size of stream
s = desired sample size
μ = message complexity

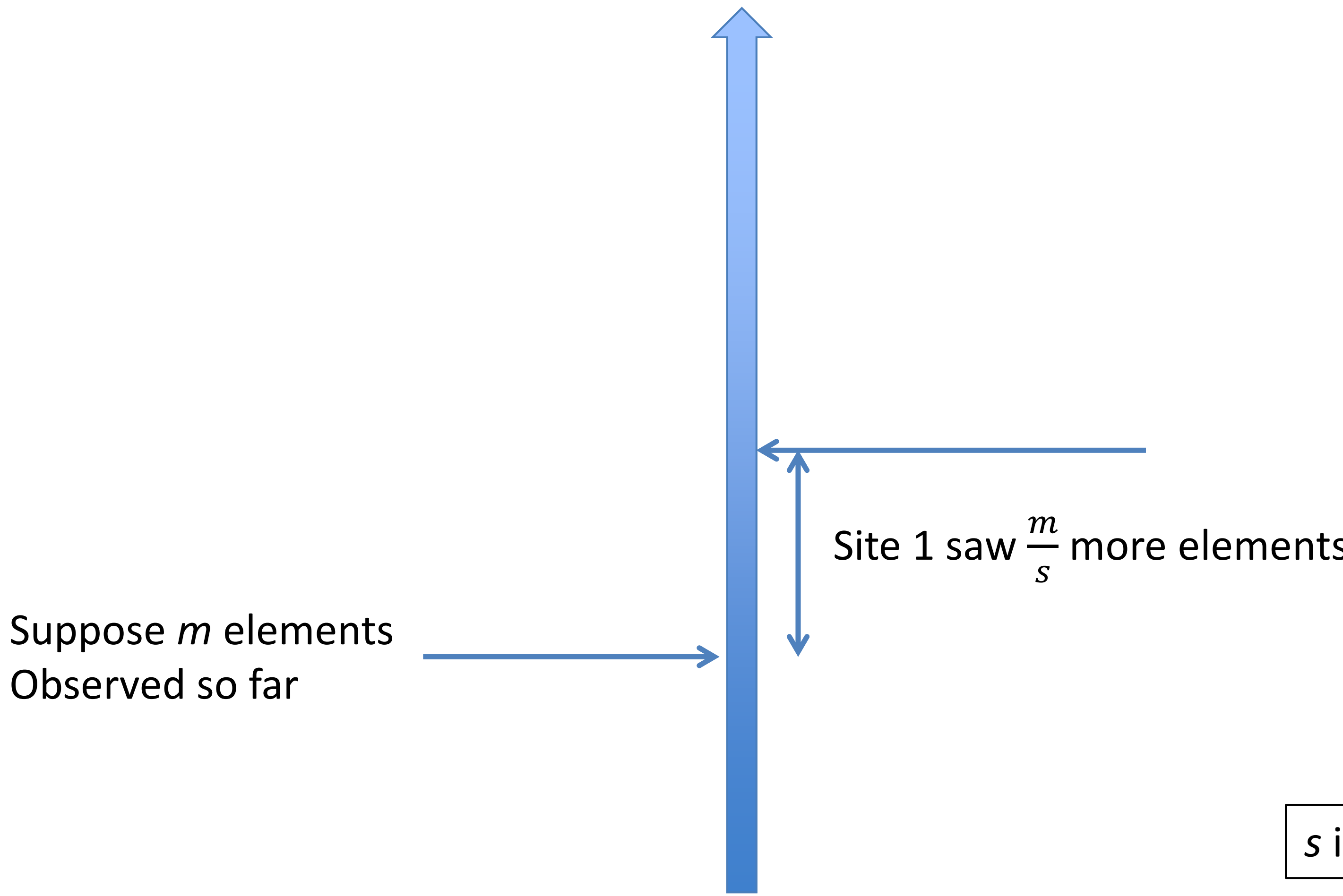
Proof: $E[\mu]$ is a function of r. Minimize with respect to r, to get the desired result.

Lower Bound



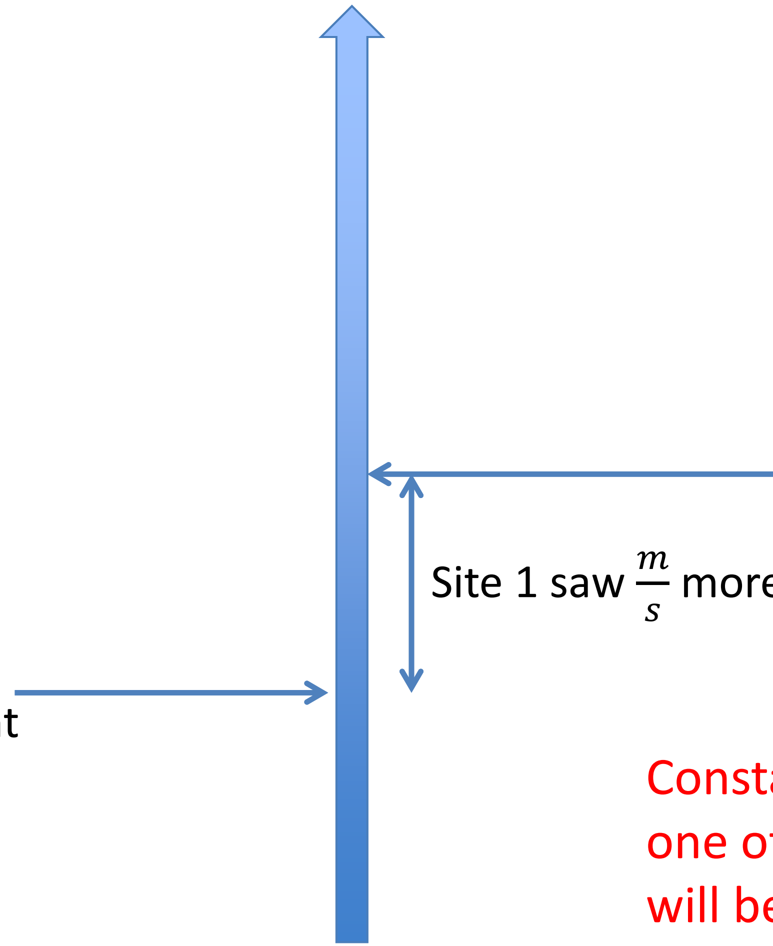
Suppose m elements
Observed so far

Lower Bound: Execution 1



s is the sample size

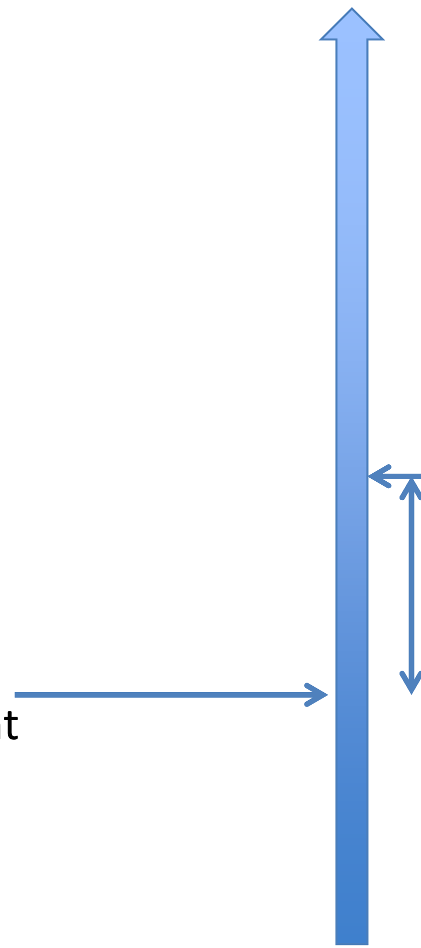
Lower Bound: Execution 1



Constant probability that
one of site 1's elements
will be included in the sample

s is the sample size

Lower Bound: Execution 1



Suppose m elements
Observed till this point

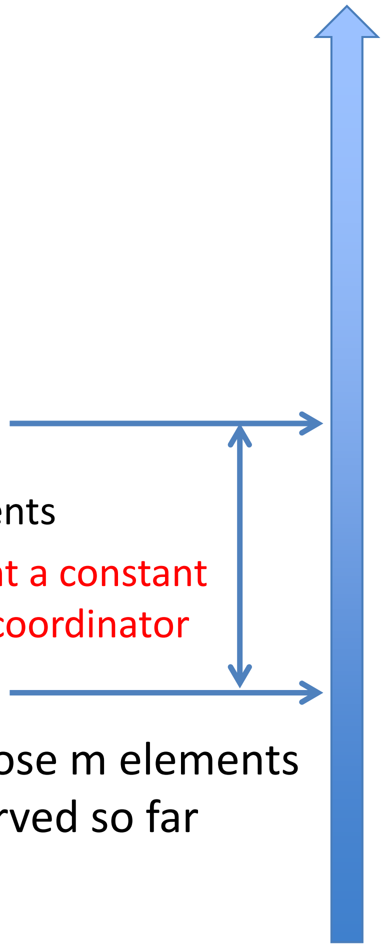
Site 1 saw $\frac{m}{s}$ more elements

And (on expectation) sent a constant number of messages to coordinator

There is a constant probability that one of site 1's elements will be included in the sample

s is the sample size

Lower Bound: Execution 2



Site 2 saw $\frac{m}{s}$ more elements

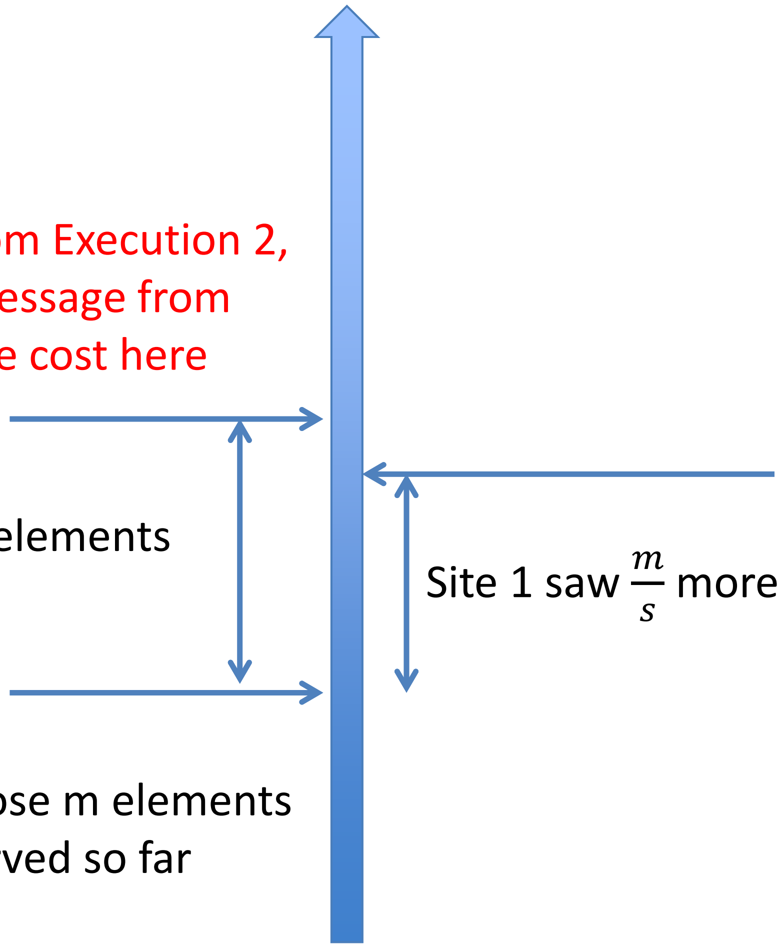
And (on expectation) sent a constant number of messages to coordinator

Suppose m elements
Observed so far

s is the sample size

Lower Bound: Execution 3

Cannot distinguish from Execution 2, unless it received a message from coordinator – message cost here



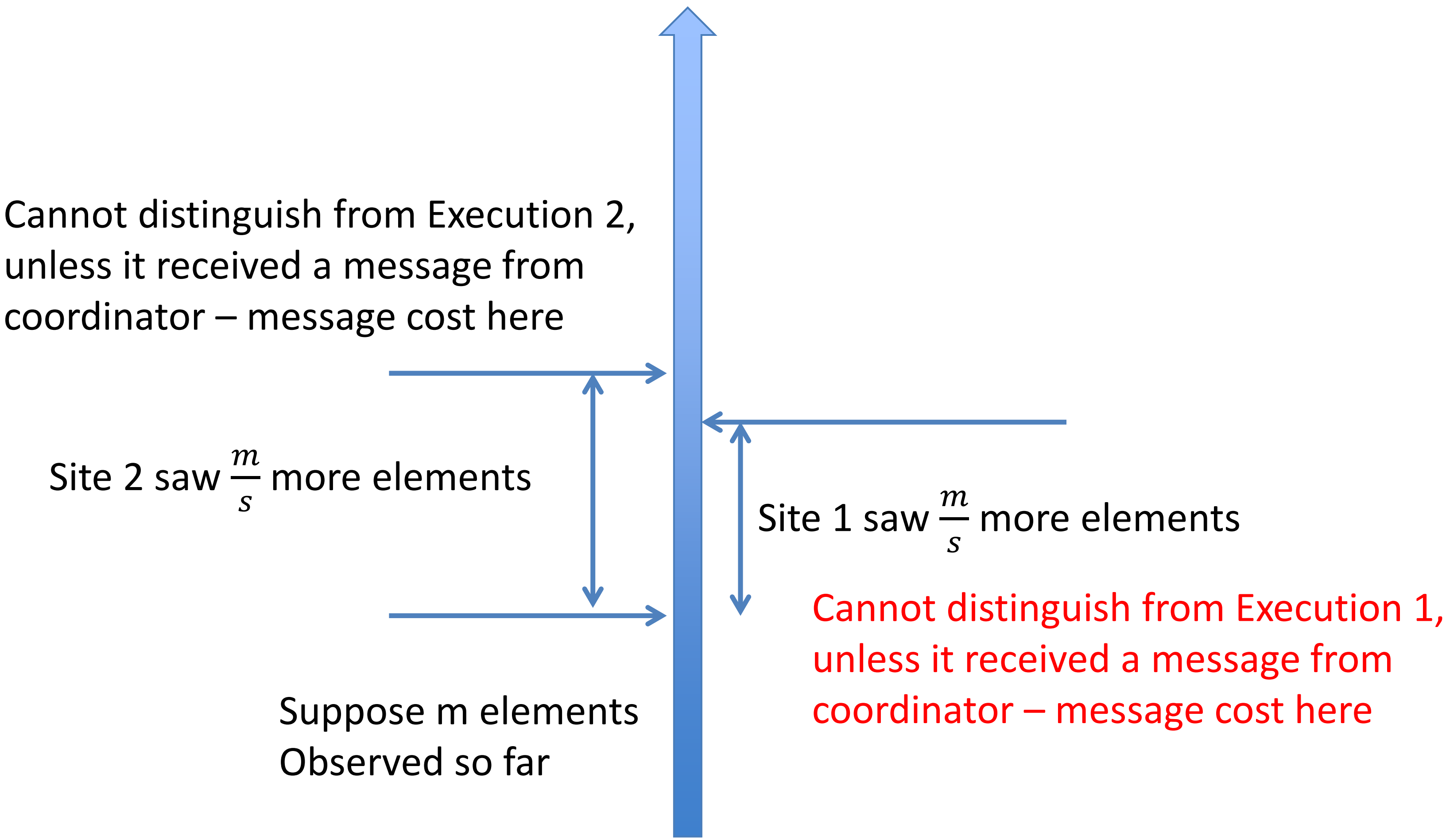
Site 2 saw $\frac{m}{s}$ more elements

Site 1 saw $\frac{m}{s}$ more elements

Suppose m elements
Observed so far

s is the sample size

Lower Bound: Execution 3



Lower Bound

Theorem: For any constant q, 0 < q < 1, any correct protocol must send $\Omega\left(\frac{k \log\left(\frac{n}{s}\right)}{\log\left(1+\frac{k}{s}\right)}\right)$ messages with probability at least 1–q, where the probability is taken over the protocol's internal randomness.

k = number of sites
n = Total size of stream
s = desired sample size

Conclusion

- Random Sampling without replacement on distributed streams
- Optimal message complexity, within constant factors
- Through a reduction, also leads to the best known message complexity for heavy-hitters over continuous distributed streams
- Algorithm for Random Sampling with Replacement

Open Problems

- Tight Lower Bounds for other Problems
  - Estimating Number of Distinct Elements
  - Heavy-Hitters (Frequent Elements)
  - Random Sampling With Replacement
- Fault Tolerance
  - Need definition of fault models