Stochastic Streams

Andrew McGregor
UC San Diego

IITK Data Streams Workshop 12/19/06

What are **Stochastic Streams?**

Andrew McGregor
UC San Diego

IITK Data Streams Workshop 12/19/06

What are

Stochastic Streams?

a) Estimating properties of distributions.

***What are* Stochastic Streams?**

a) Estimating properties of distributions.

***What are* Stochastic Streams?**

b) Processing non-deterministic data in streams.

a) Estimating properties of distributions.

What are
Stochastic Streams?

b) Processing non-deterministic data in streams.

c) Where do streams come from?

a) Estimating Properties of Distributions

a) Estimating Properties of Distributions

- Stream: $a_1, a_2, \ldots, a_m$ where $a_j \in [n]$

a) Estimating Properties of Distributions

- **Stream:** $a_1, a_2, \ldots, a_m$ where $a_j \in [n]$
- **Empirical Distribution:** p_i = (freq. of i)/m

a) Estimating Properties of Distributions

- Stream: $a_1, a_2, \ldots, a_m$ where $a_j \in [n]$
- Empirical Distribution: p_i = (freq. of i)/m
- Entropy: $\sum -p_i \log p_i$

$O(\epsilon^{-3} \log^5 m \log \delta^{-1})$
[Bhuvanagiri, Ganguly '06]

$O(\epsilon^{-2} \log m \log \delta^{-1})$
[Chakribarti, Cormode, McGregor '07]

a) Estimating Properties of Distributions

- **Stream:** $a_1, a_2, \ldots, a_m$ where $a_j \in [n]$
- **Empirical Distribution:** p_i = (freq. of i)/m
- **Entropy:** $\sum -p_i \log p_i$

  $O(\epsilon^{-3} \log^5 m \log \delta^{-1})$ [Bhuvanagiri, Ganguly '06]

  $O(\epsilon^{-2} \log m \log \delta^{-1})$ [Chakribarti, Cormode, McGregor '07]

- **Information Distances:** e.g. $\sum (\sqrt{p_i} - \sqrt{q_i})^2$

  Multiplicative Approx: All *f*-Divergences (except L_1) and Bregman-Divergences (except L_2) require $\Omega(n)$ space.

  Additive Approx: Bound *f*-Divergences, Jensen-Shannon...

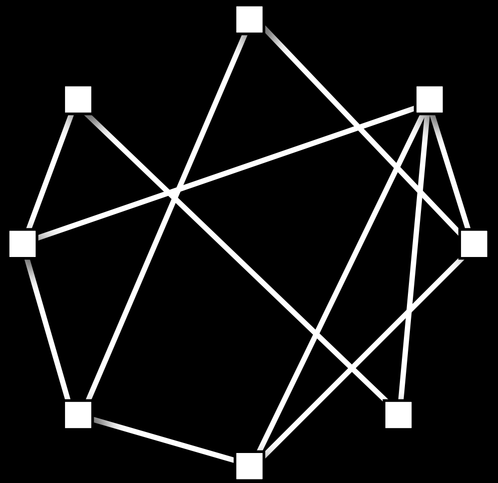
  Embedding: Can embed Hellinger but not approximate
  [Guha, McGregor, Venkatasubramarian '06], [Guha, Indyk, McGregor '07]

a) Estimating Properties of Distributions

- Stream: $a_1, a_2, \ldots, a_m$ where $a_j \in [n]$
- Empirical Markov-Chain: p_{ij} = (freq. of ij)/(freq. of i)

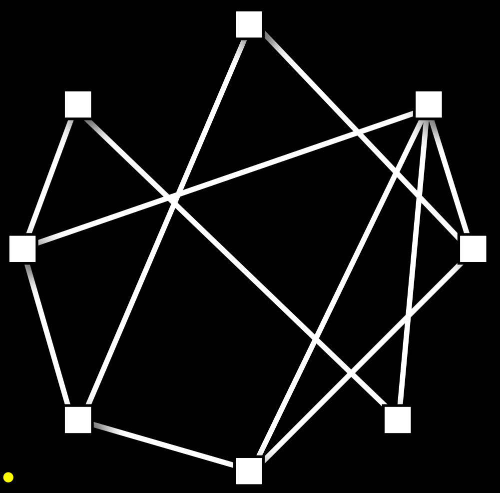
a) Estimating Properties of Distributions

- Stream: $a_1, a_2, \ldots, a_m$ where $a_j \in [n]$
- Empirical Markov-Chain: p_{ij} = (freq. of ij)/(freq. of i)



a) Estimating Properties of Distributions

- Stream: $a_1, a_2, \ldots, a_m$ where $a_j \in [n]$
- Empirical Markov-Chain: p_{ij} = (freq. of ij)/(freq. of i)



- Markov-Entropy:

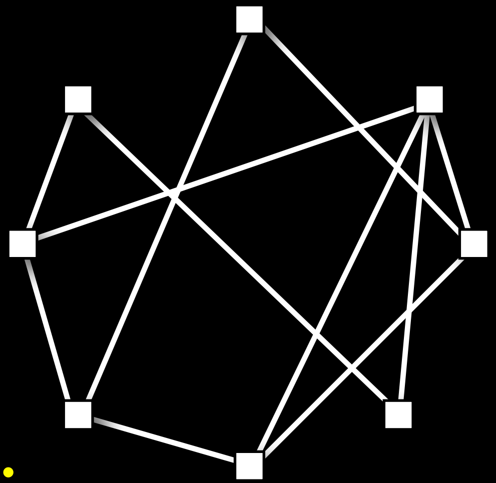
  Undirected/Unweighted: $O(\epsilon^{-2} \log^2 n \log^2 \delta^{-1})$

  General Case: Multiplicative requires $\Omega(n/\log n)$ but additive...

[Chakribarti, Cormode, McGregor '07]

a) Estimating Properties of Distributions

- Stream: $a_1, a_2, \ldots, a_m$ where $a_j \in [n]$
- Empirical Markov-Chain: $p_{ij} = 1/d(i)$ if j is a neighbour



- Markov-Entropy:

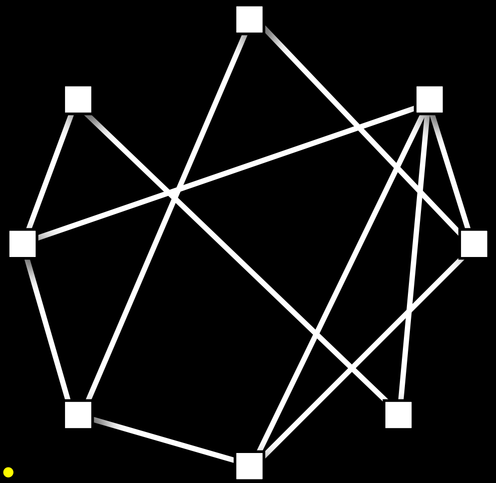
Undirected/Unweighted: $O(\epsilon^{-2} \log^2 n \log^2 \delta^{-1})$

General Case: Multiplicative requires $\Omega(n/\log n)$ but additive...

[Chakribarti, Cormode, McGregor '07]

a) Estimating Properties of Distributions

- Stream: $a_1, a_2, \ldots, a_m$ where $a_j \in [n]$
- Empirical Markov-Chain: $p_{ij} = 1/d(i)$ if j is a neighbour



- Markov-Entropy:

  Undirected/Unweighted: $O(\epsilon^{-2} \log^2 n \log^2 \delta^{-1})$

  General Case: Multiplicative requires $\Omega(n/\log n)$ but additive...

  [Chakribarti, Cormode, McGregor '07]

- What about mixing-time, cover-time etc.?

b) Processing non-deterministic data in streams.

b) Processing non-deterministic data in streams.

- Probabilistic Stream: $A_1, A_2, \ldots, A_m$ where A_j is a density on $[n] \cup \{\perp\}$. [Jayram, Kale, Vee '07]

b) Processing non-deterministic data in streams.

- Probabilistic Stream: $A_1, A_2, \ldots, A_m$ where A_j is a density on $[n] \cup \{\perp\}$. [Jayram, Kale, Vee '07]
- Defines distribution over "Deterministic" Streams:

$$\Pr(\langle a_1, \ldots, a_n \rangle) = \prod \Pr(A_i = a_i)$$

b) Processing non-deterministic data in streams.

- Probabilistic Stream: $A_1, A_2, \ldots, A_m$ where A_j is a density on $[n] \cup \{\perp\}$. [Jayram, Kale, Vee '07]
- Defines distribution over "Deterministic" Streams:

$$\Pr(\langle a_1, \ldots, a_n \rangle) = \prod \Pr(A_i = a_i)$$

- Goal: Compute expected values of aggregates, e.g.

  Mean, Sum, F_1, Max [Jayram, Kale, Vee '07]

  Mean, Median, F_0, F_2 [McGregor, Muthukrishnan '07]

b) Processing non-deterministic data in streams.

- Thm: $O(\log n)$-pass $(1+\epsilon)$-approx for $E[Mean]$ in $O(\epsilon^{-1} \log n)$ space. [Jayram, Kale, Vee '07]

b) Processing non-deterministic data in streams.

- Thm: Single-pass $(1+\epsilon)$-approx for E[*Mean*] in $O(\epsilon^{-1} \log n)$ space. [McGregor, Muthukrishnan '07]

b) Processing non-deterministic data in streams.

- Thm: Single-pass $(1+\epsilon)$-approx for E[*Mean*] in $O(\epsilon^{-1} \log n)$ space. [McGregor, Muthukrishnan '07]
- Proof (Sketch):

Note E[*Mean*] = E[*Sum*/*Count*] ≠ E[*Sum*]/E[*Count*]

b) Processing non-deterministic data in streams.

- **Thm:** Single-pass $(1+\epsilon)$-approx for E[*Mean*] in $O(\epsilon^{-1} \log n)$ space. [McGregor, Muthukrishnan '07]
- **Proof (Sketch):**

  Note E[*Mean*] = E[*Sum*/*Count*] ≠ E[*Sum*]/E[*Count*]

  1) If E[*Count*]=$\Omega(\epsilon^{-2} \log n)$ gives error $\pm\, \epsilon$E[*Sum*/*Count*]

b) Processing non-deterministic data in streams.

- Thm: Single-pass $(1+\epsilon)$-approx for E[*Mean*] in $O(\epsilon^{-1} \log n)$ space. [McGregor, Muthukrishnan '07]
- Proof (Sketch):

Note E[*Mean*] = E[*Sum*/*Count*] ≠ E[*Sum*]/E[*Count*]

1) If E[*Count*]$=\Omega(\epsilon^{-2} \log n)$ gives error $\pm\ \epsilon$E[*Sum*/*Count*]

2) Assume E[*Count*]$=O(\epsilon^{-2} \log n)$:

b) Processing non-deterministic data in streams.

- **Thm:** Single-pass $(1+\epsilon)$-approx for E[*Mean*] in $O(\epsilon^{-1} \log n)$ space. [McGregor, Muthukrishnan '07]
- **Proof (Sketch):**

  Note $E[Mean] = E[Sum/Count] \neq E[Sum]/E[Count]$

  1) If $E[Count]=\Omega(\epsilon^{-2} \log n)$ gives error $\pm\ \epsilon E[Sum/Count]$

  2) Assume $E[Count]=O(\epsilon^{-2} \log n)$:

     Let $Count_j = F_1(a_1, \ldots, a_j)$ and $Mean_j = (a_1 + \ldots + a_j)/Count_j$

b) Processing non-deterministic data in streams.

- Thm: Single-pass $(1+\epsilon)$-approx for E[*Mean*] in $O(\epsilon^{-1} \log n)$ space. [McGregor, Muthukrishnan '07]
- Proof (Sketch):

Note $E[Mean] = E[Sum/Count] \neq E[Sum]/E[Count]$

1) If $E[Count]=\Omega(\epsilon^{-2} \log n)$ gives error $\pm\, \epsilon E[Sum/Count]$
2) Assume $E[Count]=O(\epsilon^{-2} \log n)$:

   Let $Count_j = F_1(a_1, \ldots, a_j)$ and $Mean_j = (a_1 + \ldots + a_j)/Count_j$

   Then whp. $A=\{Count_j=E[Count_j] \pm O(\epsilon^{-1} \log n)\text{: for all } j\}$

b) Processing non-deterministic data in streams.

- Thm: Single-pass $(1+\epsilon)$-approx for E[*Mean*] in $O(\epsilon^{-1} \log n)$ space. [McGregor, Muthukrishnan '07]
- Proof (Sketch):

Note $E[\mathit{Mean}] = E[\mathit{Sum}/\mathit{Count}] \neq E[\mathit{Sum}]/E[\mathit{Count}]$

1) If $E[\mathit{Count}]=\Omega(\epsilon^{-2} \log n)$ gives error $\pm\ \epsilon E[\mathit{Sum}/\mathit{Count}]$
2) Assume $E[\mathit{Count}]=O(\epsilon^{-2} \log n)$:

   Let $\mathit{Count}_j = F_1(a_1, \ldots, a_j)$ and $\mathit{Mean}_j = (a_1 + \ldots + a_j)/\mathit{Count}_j$

   Then whp. $A=\{\mathit{Count}_j=E[\mathit{Count}_j] \pm O(\epsilon^{-1} \log n)\text{: for all } j\}$

   Maintain $\Pr[\mathit{Count}_j=z|A]$ & $E[\mathit{Mean}_j|A]$ in $O(\epsilon^{-1} \log n)$ space.

c) Where do streams come from?

c) Where do streams come from?

? Who cares about the empirical distribution?!

c) Where do streams come from?

- Who cares about the empirical distribution?!
- What if we don't know the probabilistic stream?!

1. Models
2. Quantiles
3. Learning Distributions

“Upstream” Algorithms

“Upstream” Algorithms

- Can we infer the probabilistic stream from one or more deterministic stream(s)?

“Upstream” Algorithms

- Can we infer the probabilistic stream from one or more deterministic stream(s)?

“*source*” $A_1, A_2, \ldots, A_m$

“Upstream” Algorithms

- Can we infer the probabilistic stream from one or more deterministic stream(s)?

$a_1, a_2, a_3, \ldots$ ←

$b_1, b_2, b_3, \ldots$ ←

$c_1, c_2, c_3, \ldots$ ←

“*observed*” stream(s)

“*source*” $A_1, A_2, \ldots, A_m$

"Upstream" Algorithms

- Can we infer the probabilistic stream from one or more deterministic stream(s)?

$a_1, a_2, a_3, \ldots$ ←

$b_1, b_2, b_3, \ldots$ ←

$c_1, c_2, c_3, \ldots$ ←

"*observed*" stream(s)

"*source*" $A_1, A_2, \ldots, A_m$

- E.g. each A_i is $\perp$ w/p p and j w/p 1-p

[Batu, Kannan, Khanna, McGregor '04], [Kannan, McGregor, '05]

“Upstream” Algorithms

- Can we infer the probabilistic stream from one or more deterministic stream(s)?

$a_1, a_2, a_3, \ldots$ ←

$b_1, b_2, b_3, \ldots$ ←

$c_1, c_2, c_3, \ldots$ ←

“*observed*” stream(s)

“*source*” $A_1, A_2, \ldots, A_m$

- E.g. each A_i is $\perp$ w/p p and j w/p 1-p

[Batu, Kannan, Khanna, McGregor ’04], [Kannan, McGregor, ’05]

- E.g. each A_i is identically distributed...

“Upstream” Algorithms

“Upstream” Algorithms

- Each stream element is an independent sample from an unknown distribution μ.
- Goal: Estimate function f(μ)

“Upstream” Algorithms

- Each stream element is an independent sample from an unknown distribution μ.
- Goal: Estimate function f(μ)

  Space-complexity

“Upstream” Algorithms

- Each stream element is an independent sample from an unknown distribution μ.
- Goal: Estimate function f(μ)

  Space-complexity

  Time-complexity

“Upstream” Algorithms

- Each stream element is an independent sample from an unknown distribution μ.
- Goal: Estimate function f(μ)

  Space-complexity

  Time-complexity

  Sample-complexity

1. Models
2. Quantiles
3. Learning Distributions

Quantiles

Quantiles

- Stream: m samples a distribution with density μ

Quantiles

- Stream: m samples a distribution with density μ
- ϵ-approx median: x with $\int_{-\infty}^{x} \mu(y)dy = 1/2 \pm \epsilon$

Quantiles

- Stream: m samples a distribution with density μ
- ϵ-approx median: x with $\int_{-\infty}^{x} \mu(y)dy = 1/2 \pm \epsilon$
- With m samples can find $O(m^{-1/2})$-approx median

Quantiles

- Stream: m samples a distribution with density μ
- ϵ-approx median: x with $\int_{-\infty}^{x} \mu(y)dy = 1/2 \pm \epsilon$
- With m samples can find $O(m^{-1/2})$-approx median
- First attempt:

  Can find element of rank $(1/2\pm\epsilon)m$ in $O(\epsilon^{-1} \log m)$ space

  O(s) space can find $O(\max(m^{-1/2}, s^{-1}))$-approx median

  [Greenwald, Khanna '01], [Cormode, Korn, Muthukrishnan, Srivastava '06]

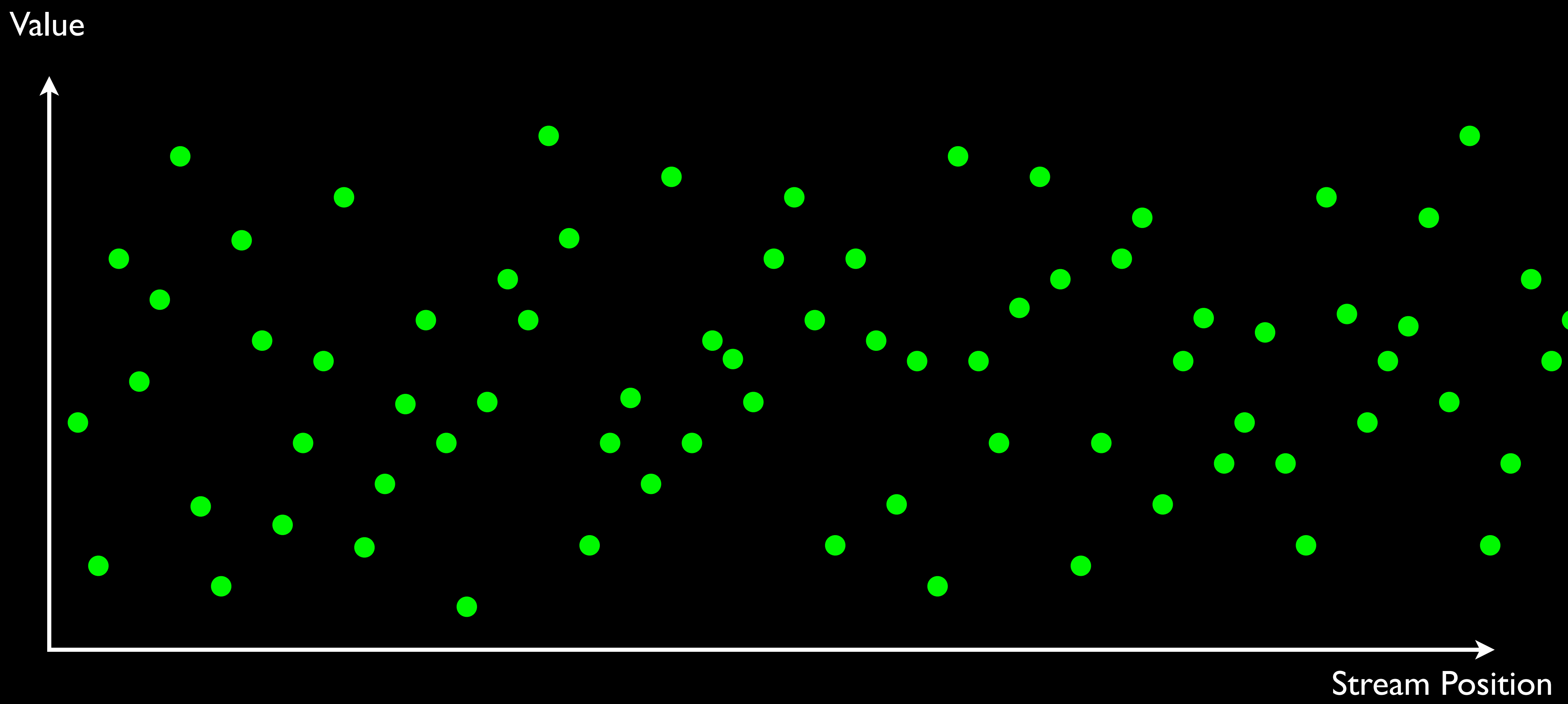
Quantiles

- Stream: m samples a distribution with density μ
- ϵ-approx median: x with $\int_{-\infty}^{x} \mu(y)dy = 1/2 \pm \epsilon$
- With m samples can find $O(m^{-1/2})$-approx median
- First attempt:

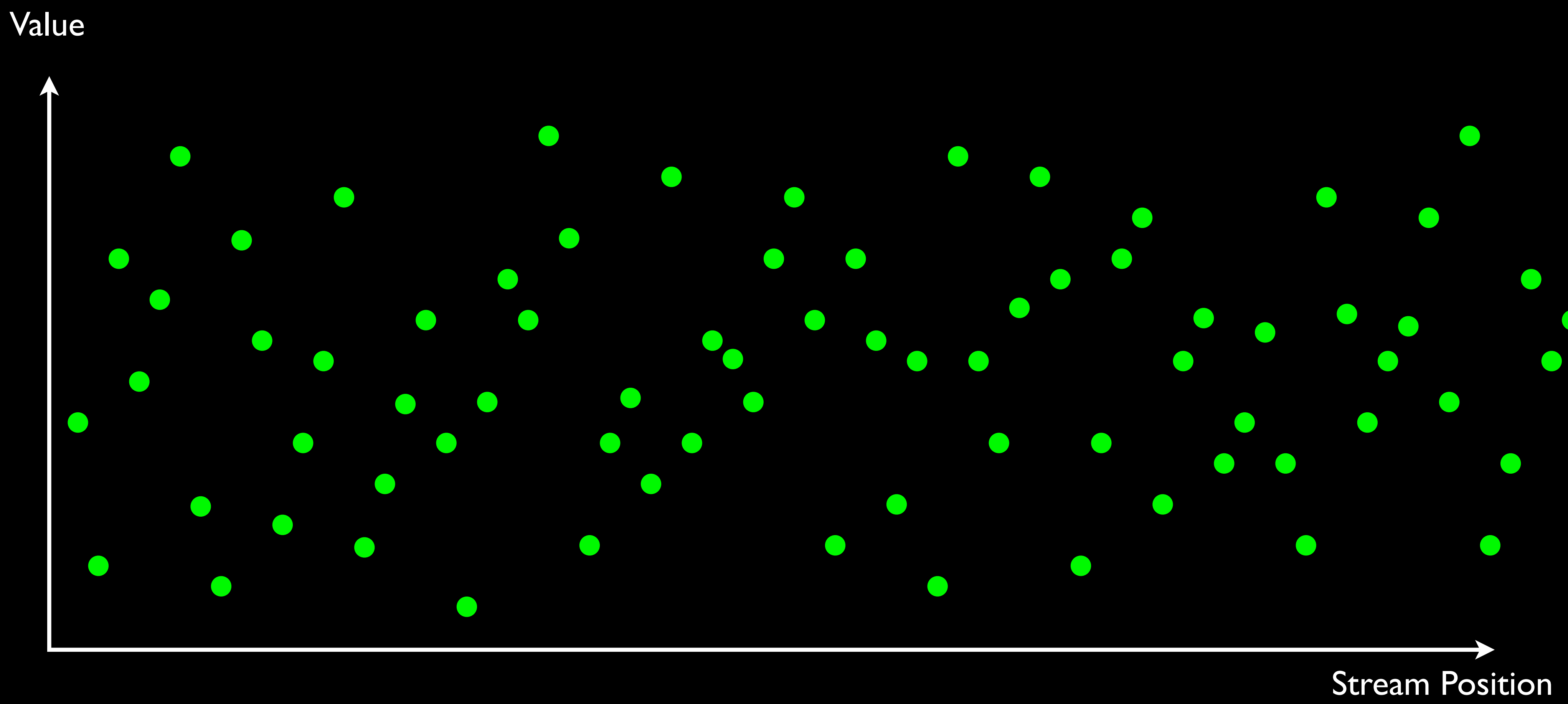
  Can find element of rank $(1/2\pm\epsilon)m$ in $O(\epsilon^{-1} \log m)$ space

  O(s) space can find $O(\max(m^{-1/2}, s^{-1}))$-approx median

  [Greenwald, Khanna '01], [Cormode, Korn, Muthukrishnan, Srivastava '06]
- Thm: Can find $O(m^{-1/2}\log m)$-approx median in O(1) words of space

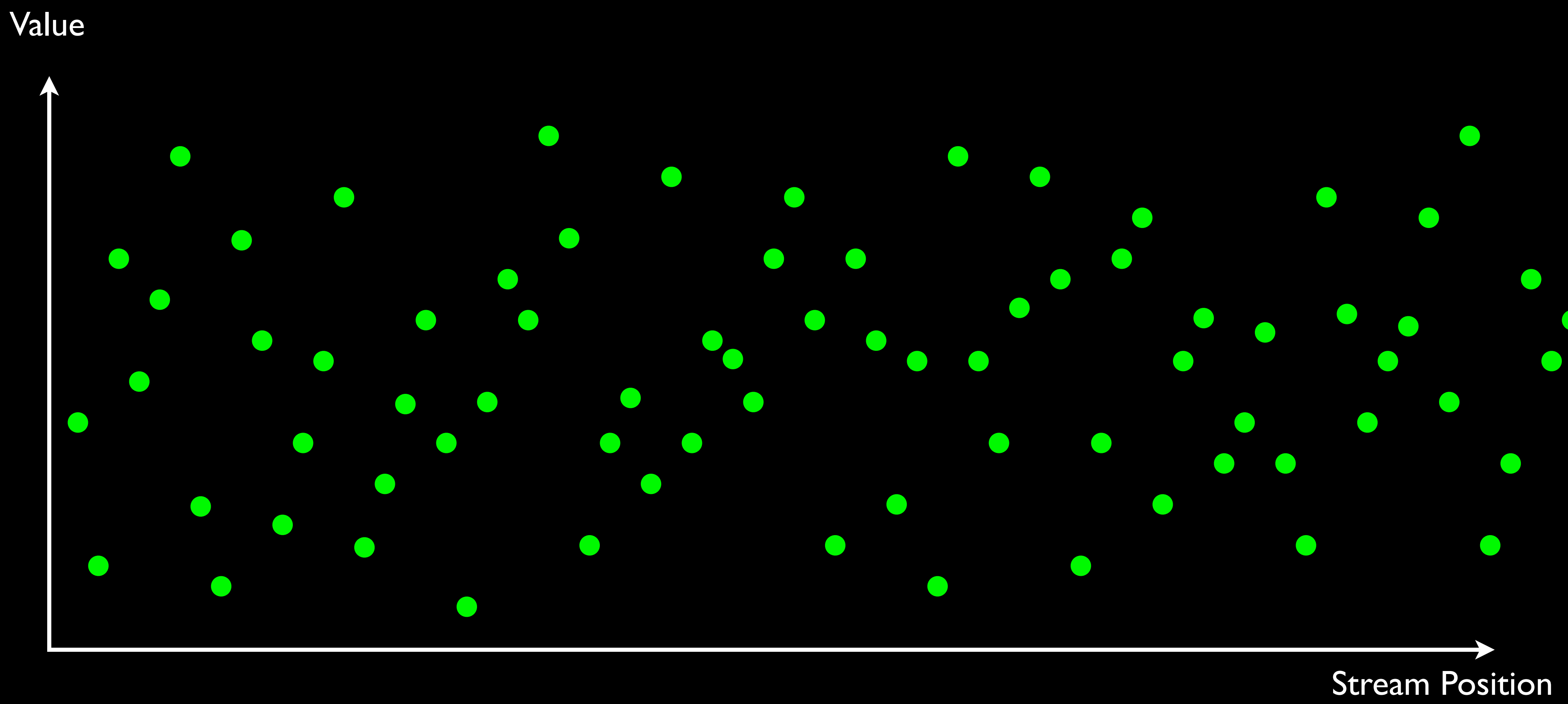

Value
Stream Position

Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]



Algorithm: Maintain lower/upper bound $[a, b]$ for median and c in $[a,b]$

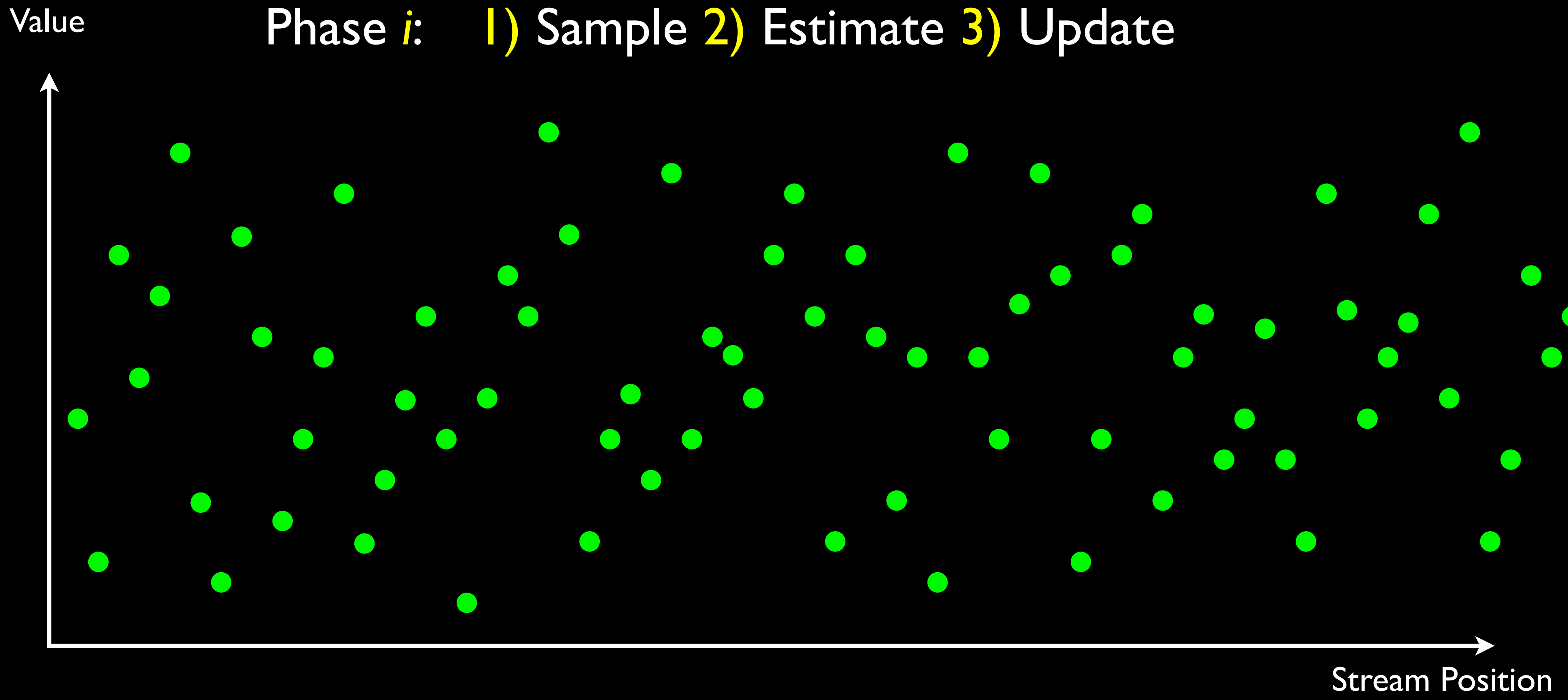
Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$



Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

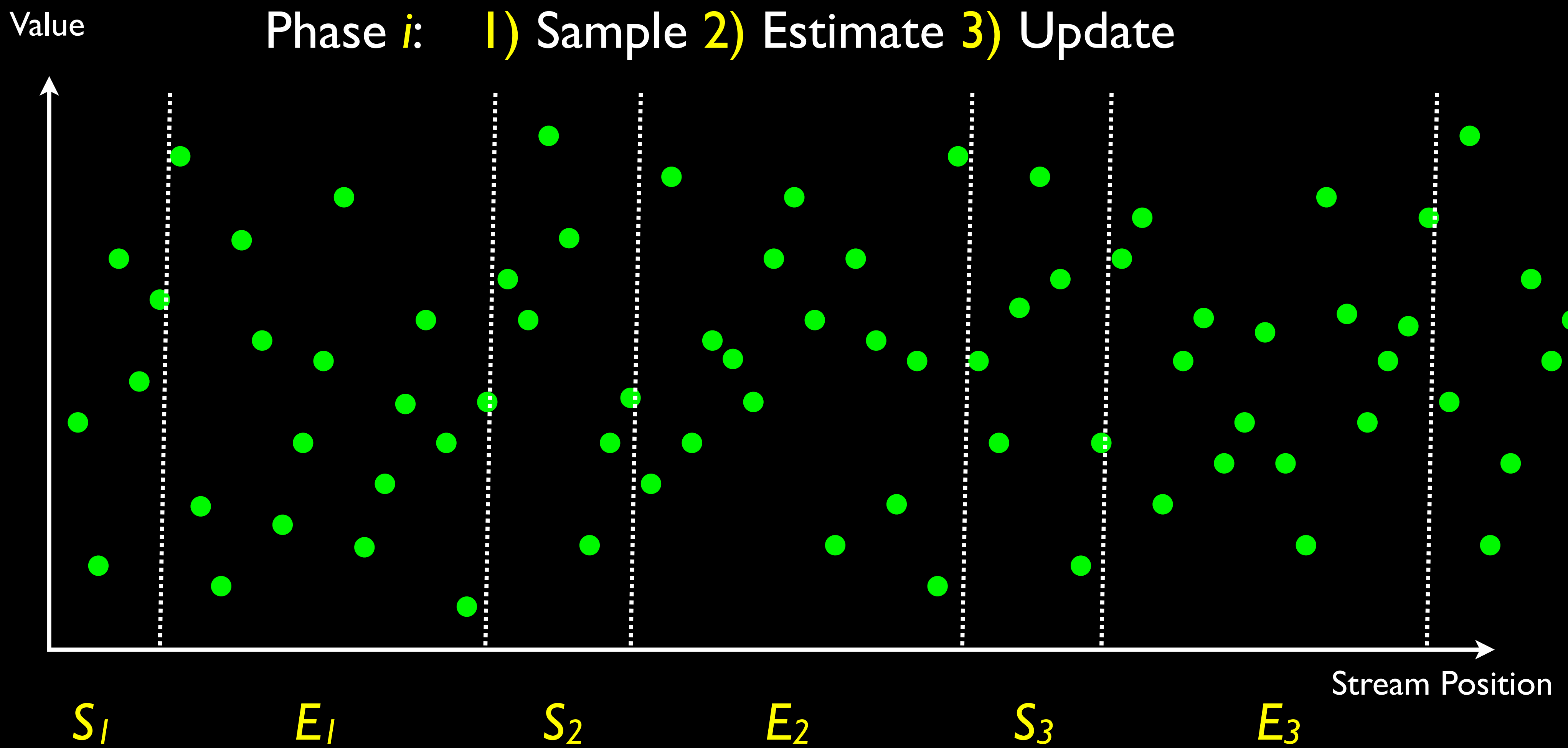
Phase *i*: 1) Sample 2) Estimate 3) Update



Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, ..., S_p, E_p$

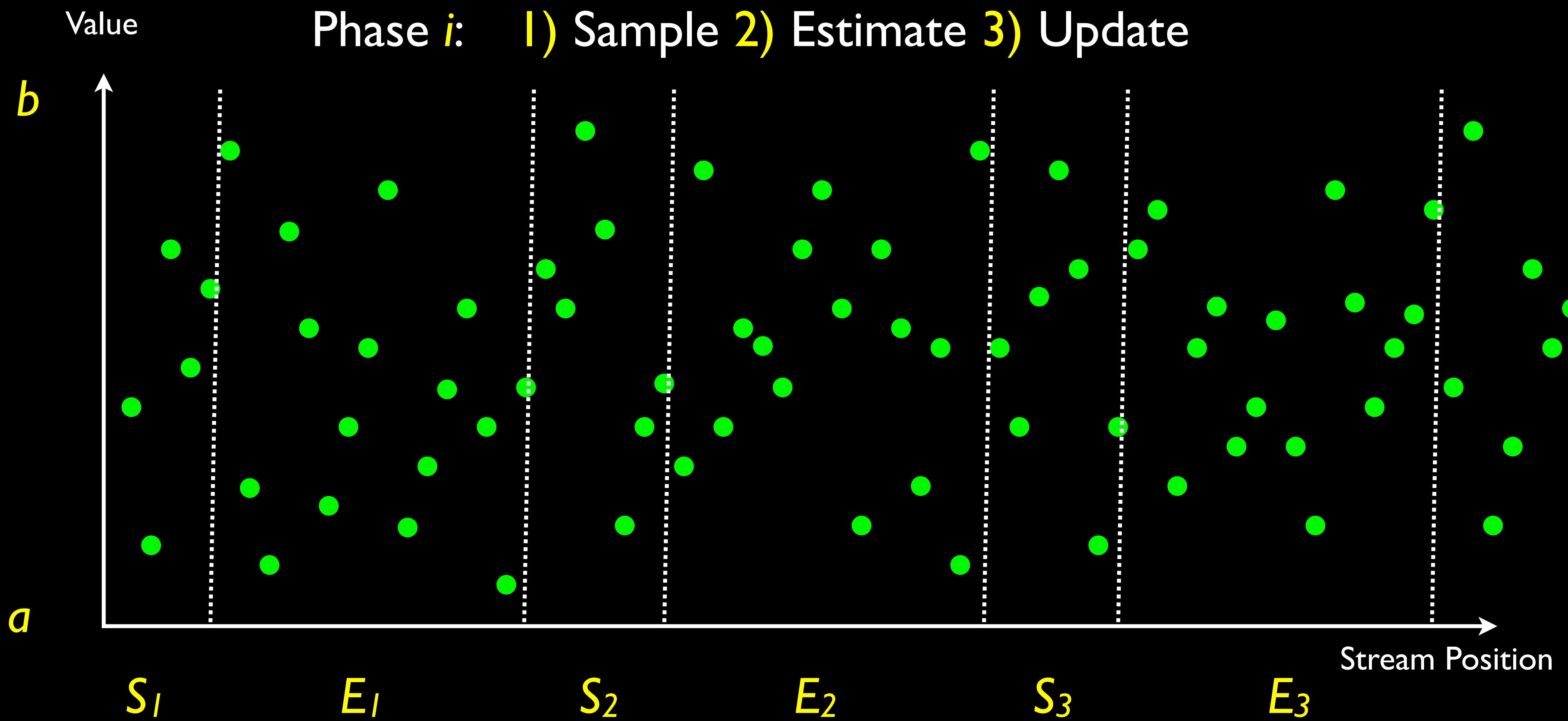
Phase *i*: 1) Sample 2) Estimate 3) Update



Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

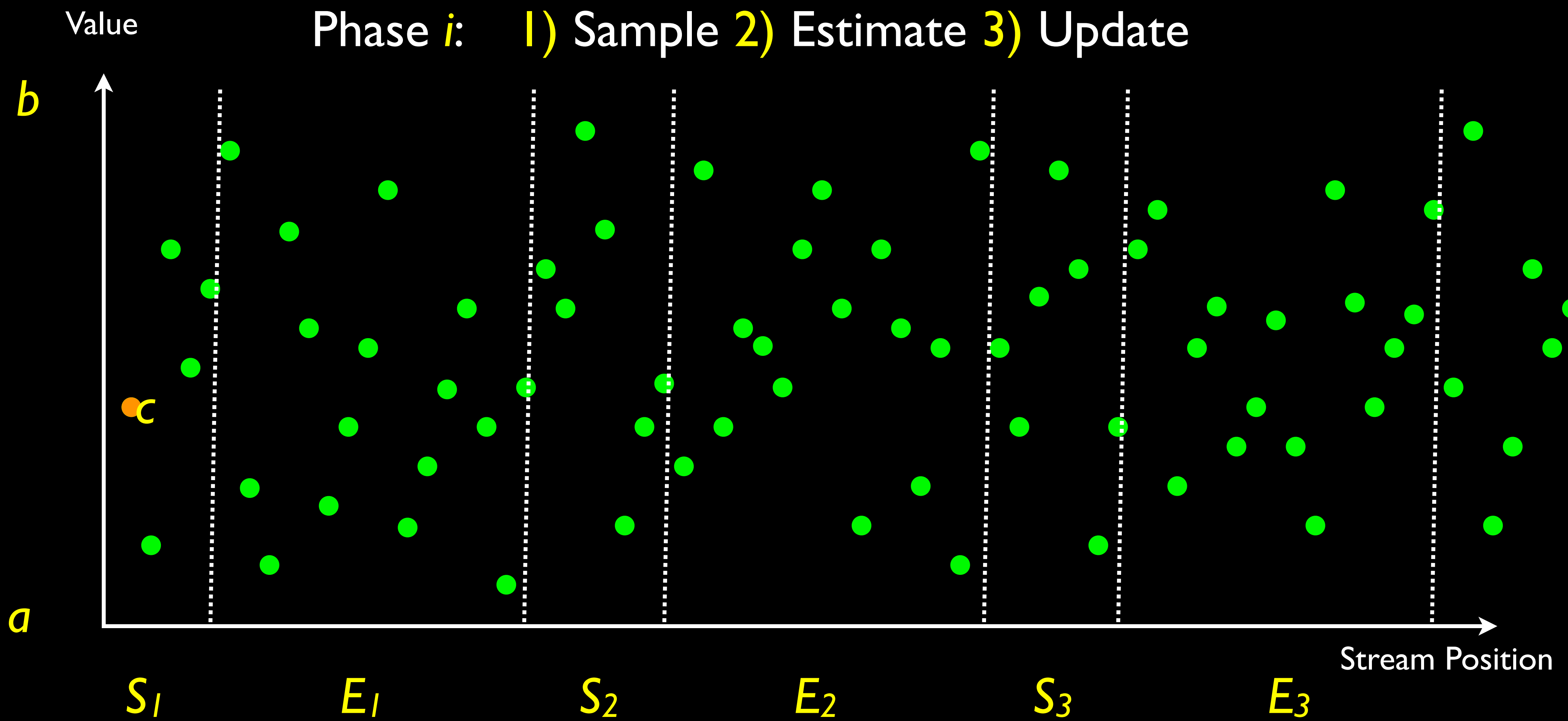
Phase i: 1) Sample 2) Estimate 3) Update



Algorithm: Maintain lower/upper bound $[a, b]$ for median and c in $[a,b]$

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

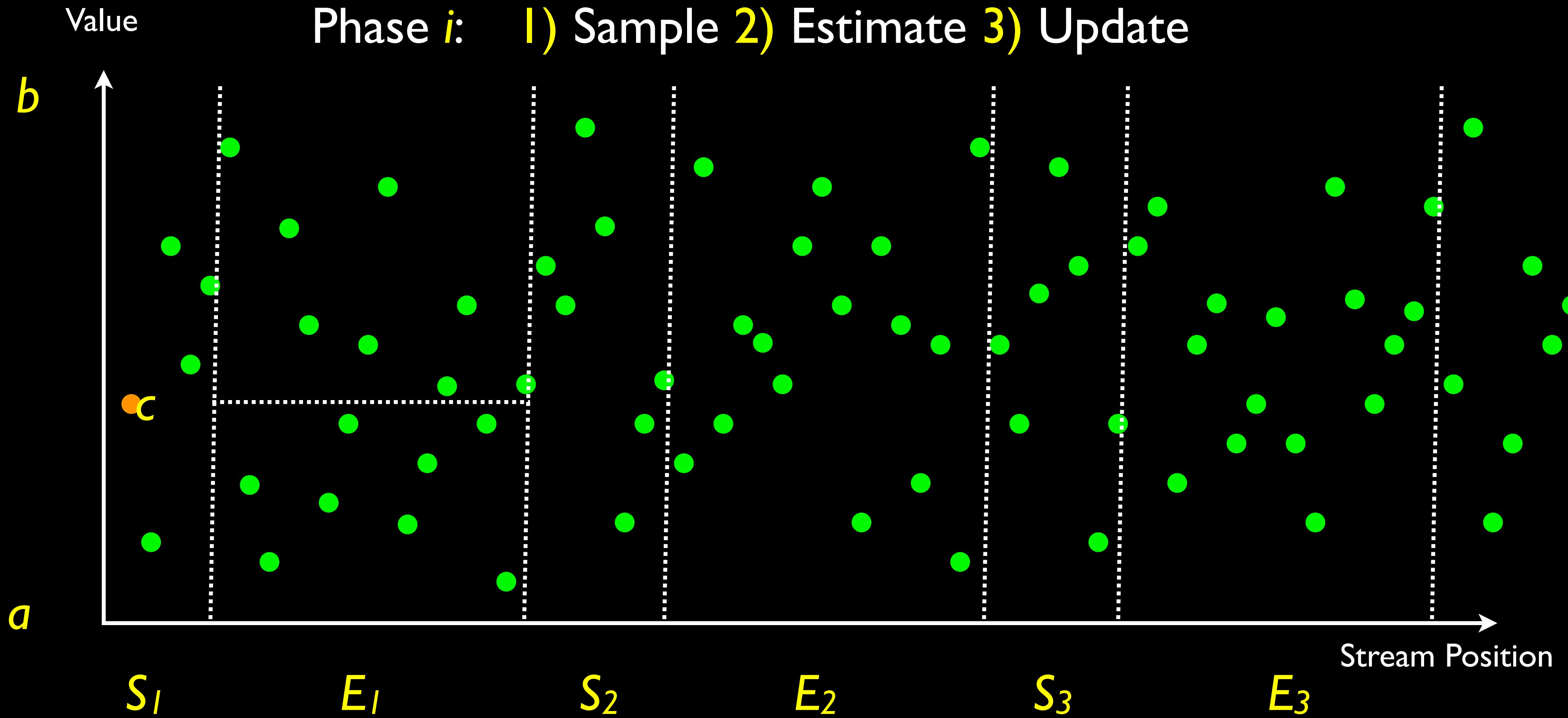
Phase i: 1) Sample 2) Estimate 3) Update



Algorithm: Maintain lower/upper bound $[a, b]$ for median and c in $[a,b]$

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

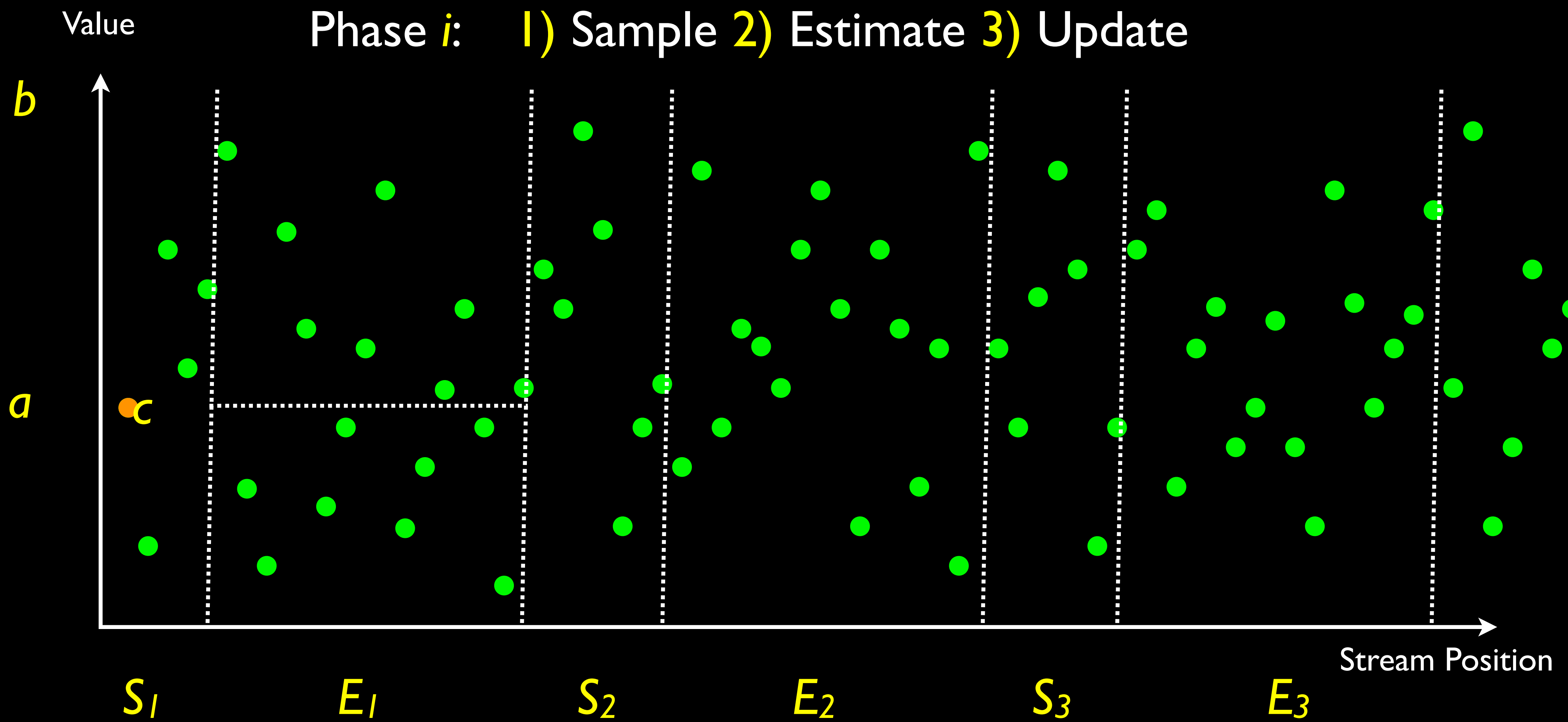
Phase i: 1) Sample 2) Estimate 3) Update



Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

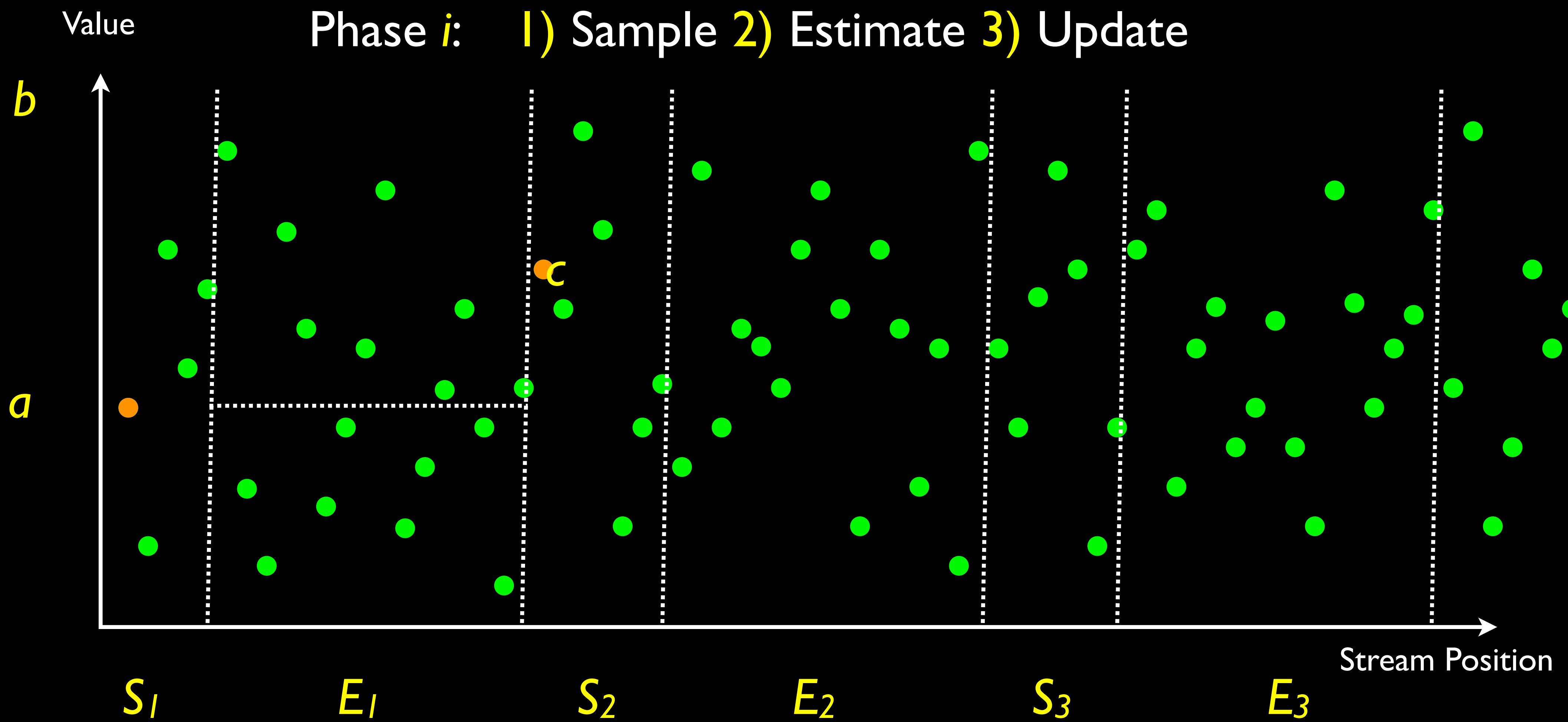
Phase *i*: 1) Sample 2) Estimate 3) Update



Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

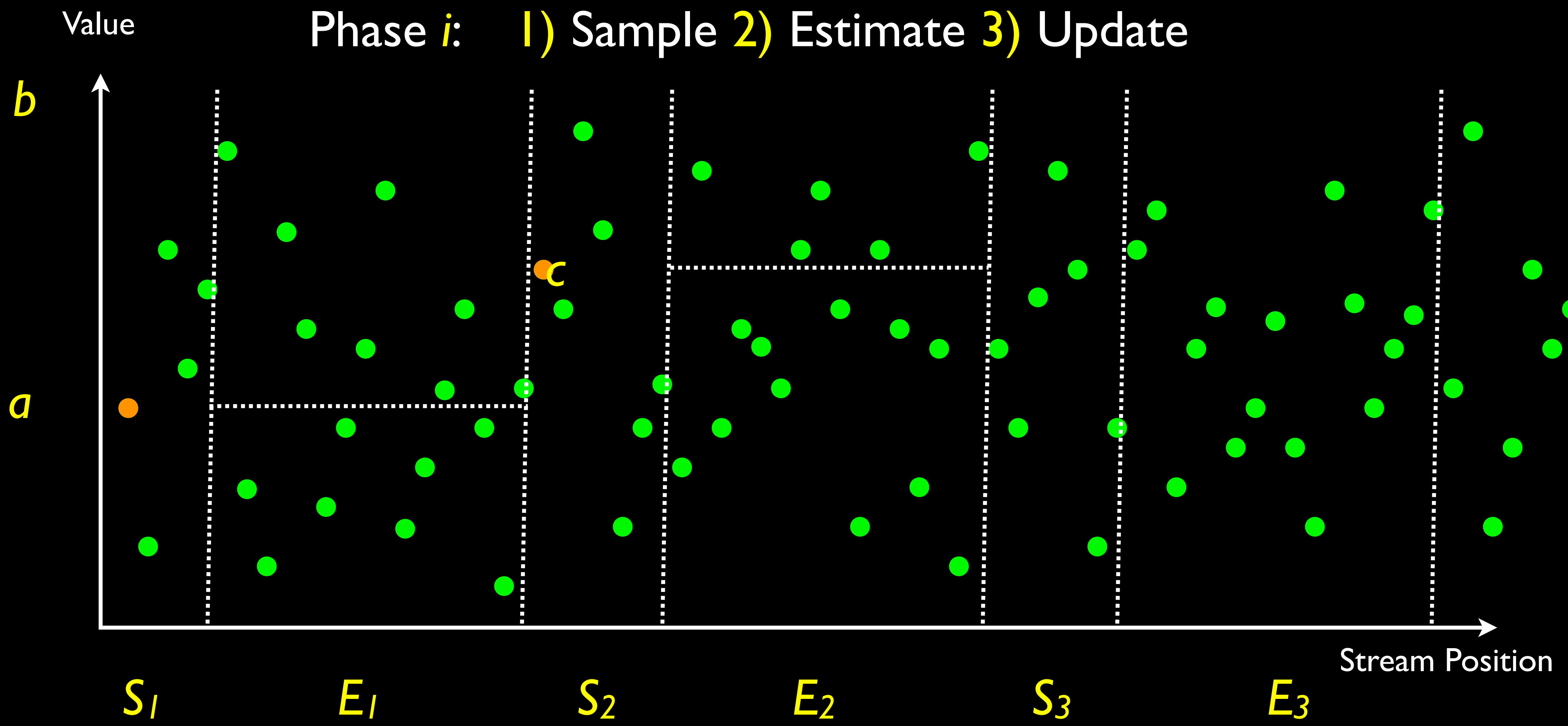
Phase *i*: 1) Sample 2) Estimate 3) Update



Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

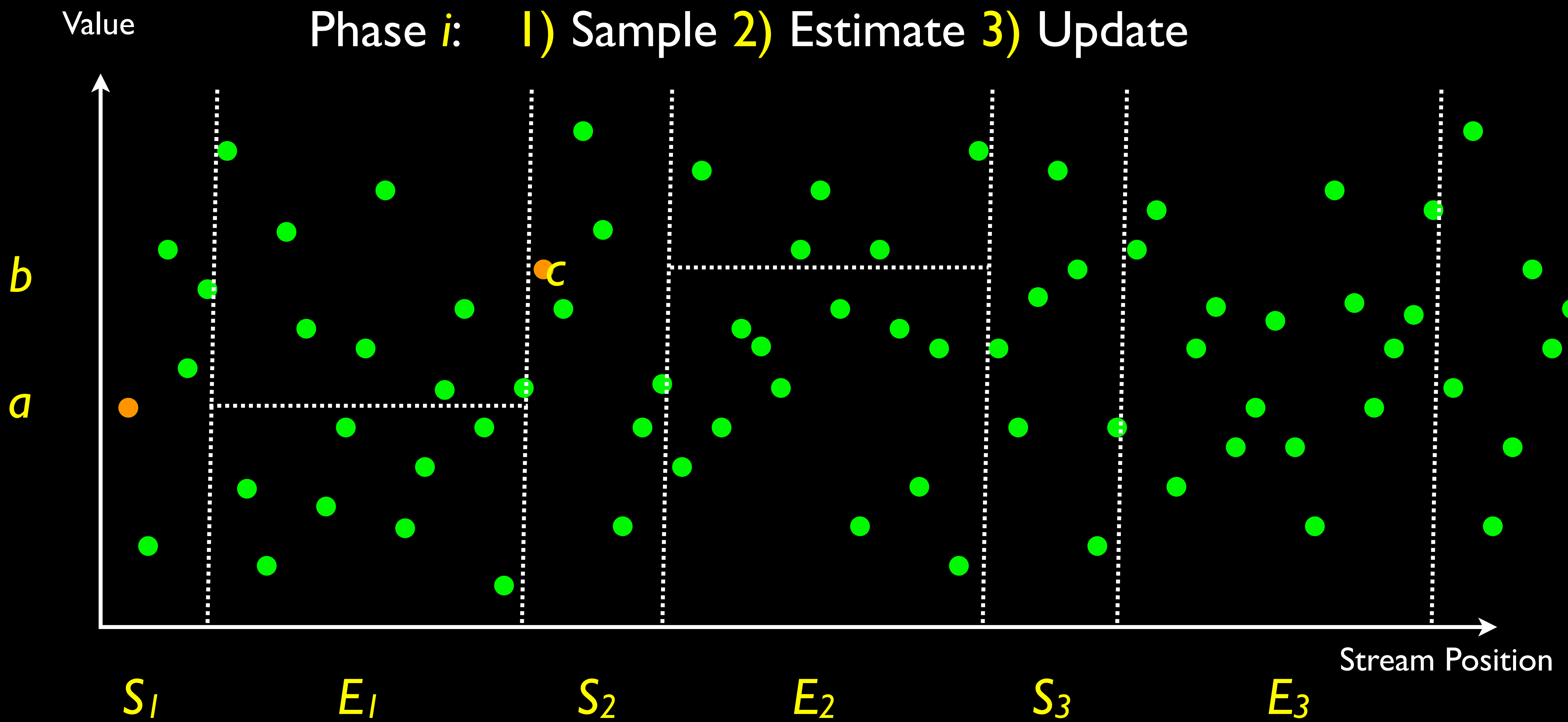
Phase i: 1) Sample 2) Estimate 3) Update



Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, ..., S_p, E_p$

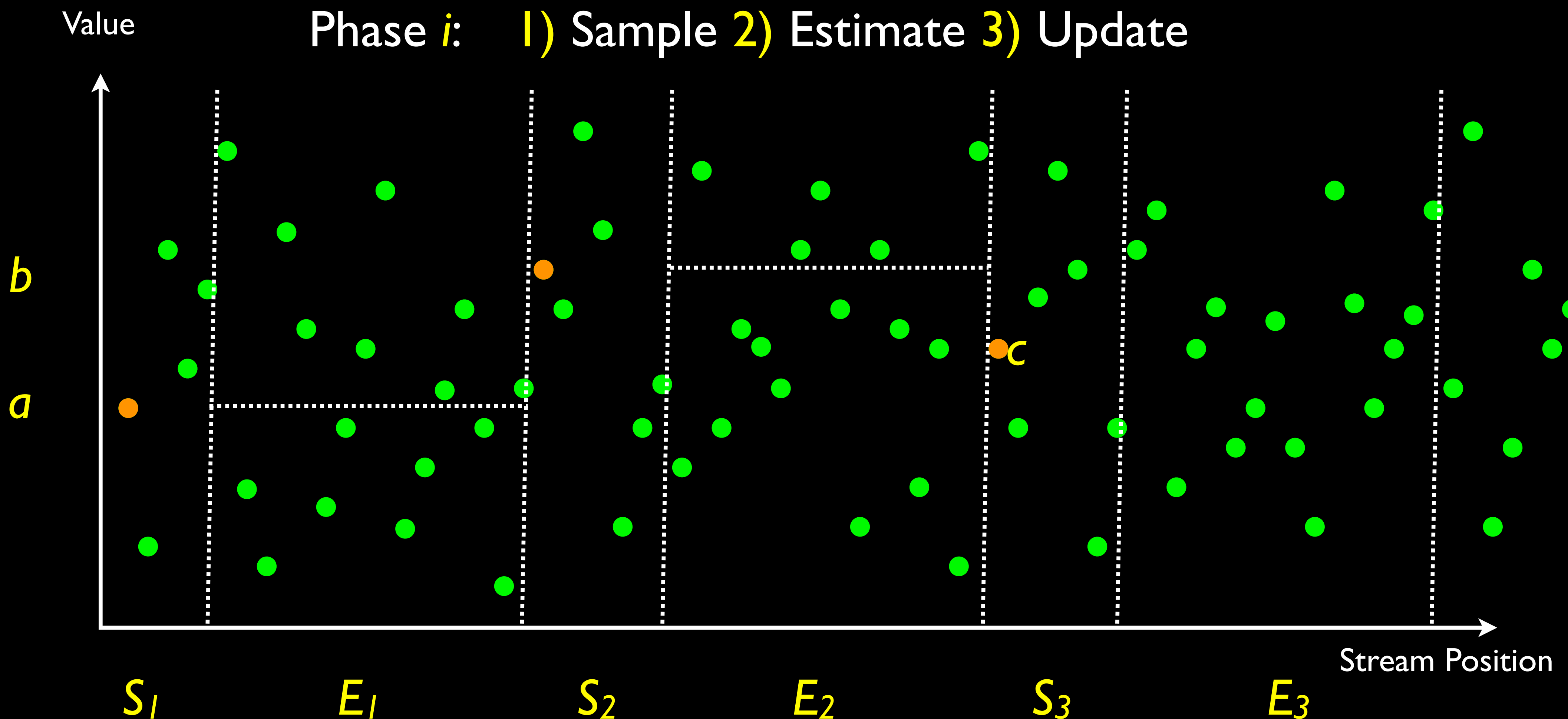
Phase *i*: 1) Sample 2) Estimate 3) Update



Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

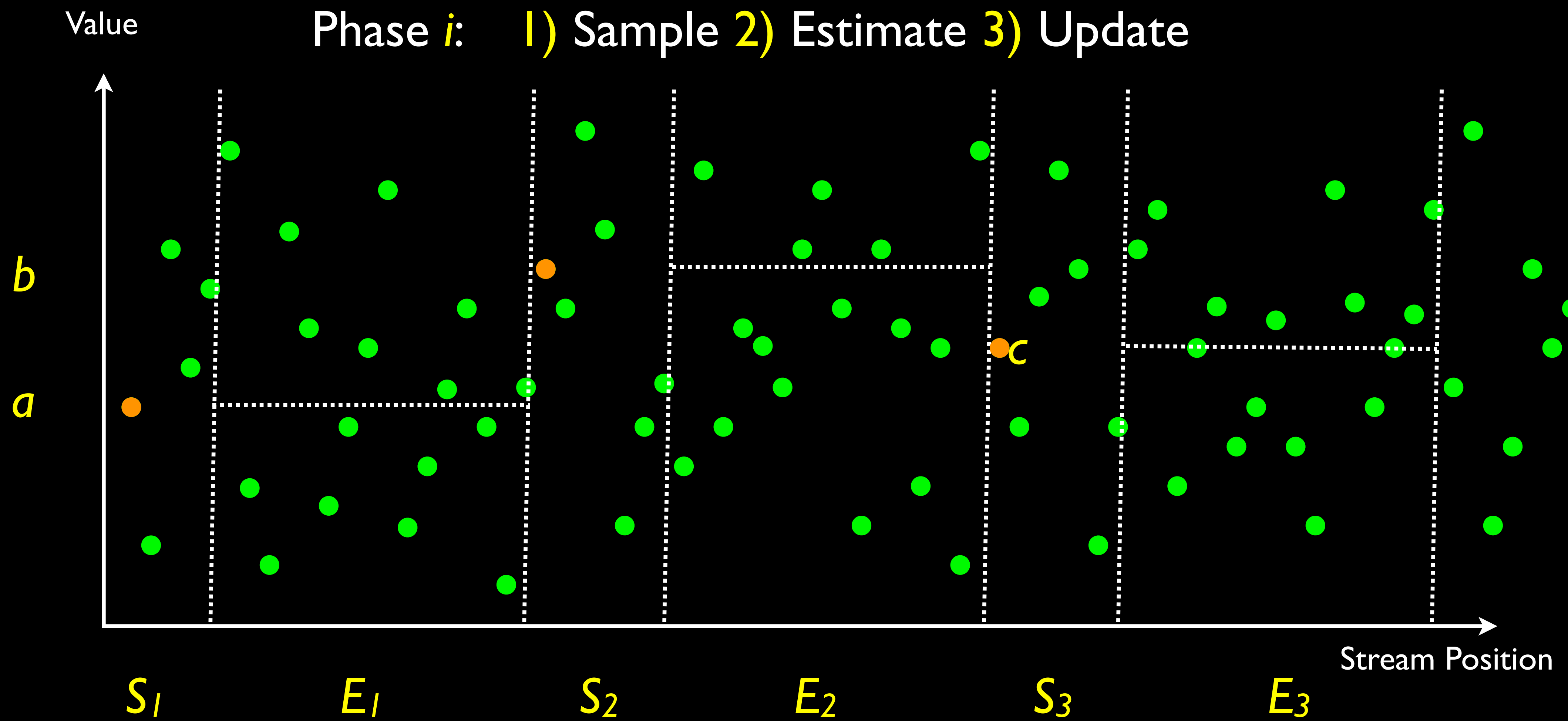
Phase *i*: 1) Sample 2) Estimate 3) Update



Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

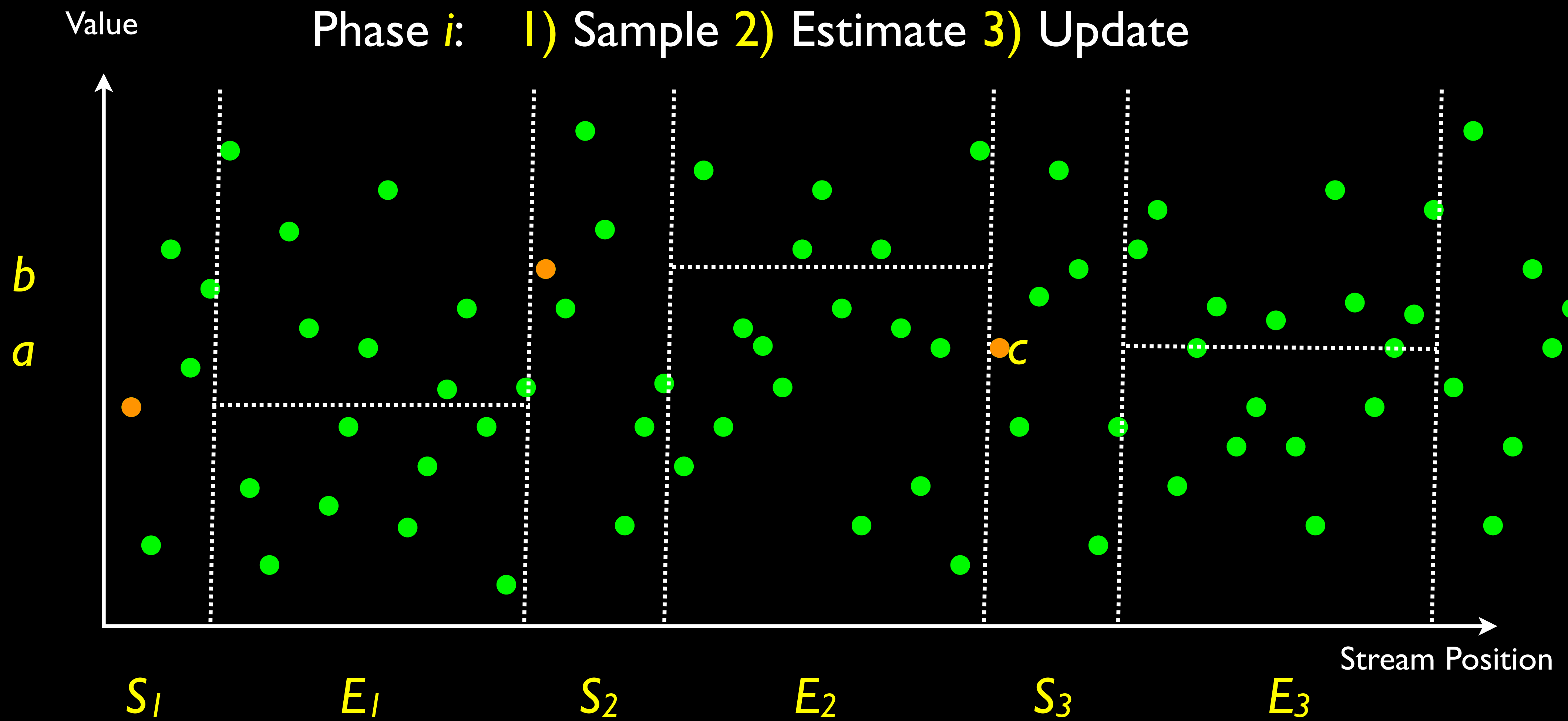
Phase *i*: 1) Sample 2) Estimate 3) Update



Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

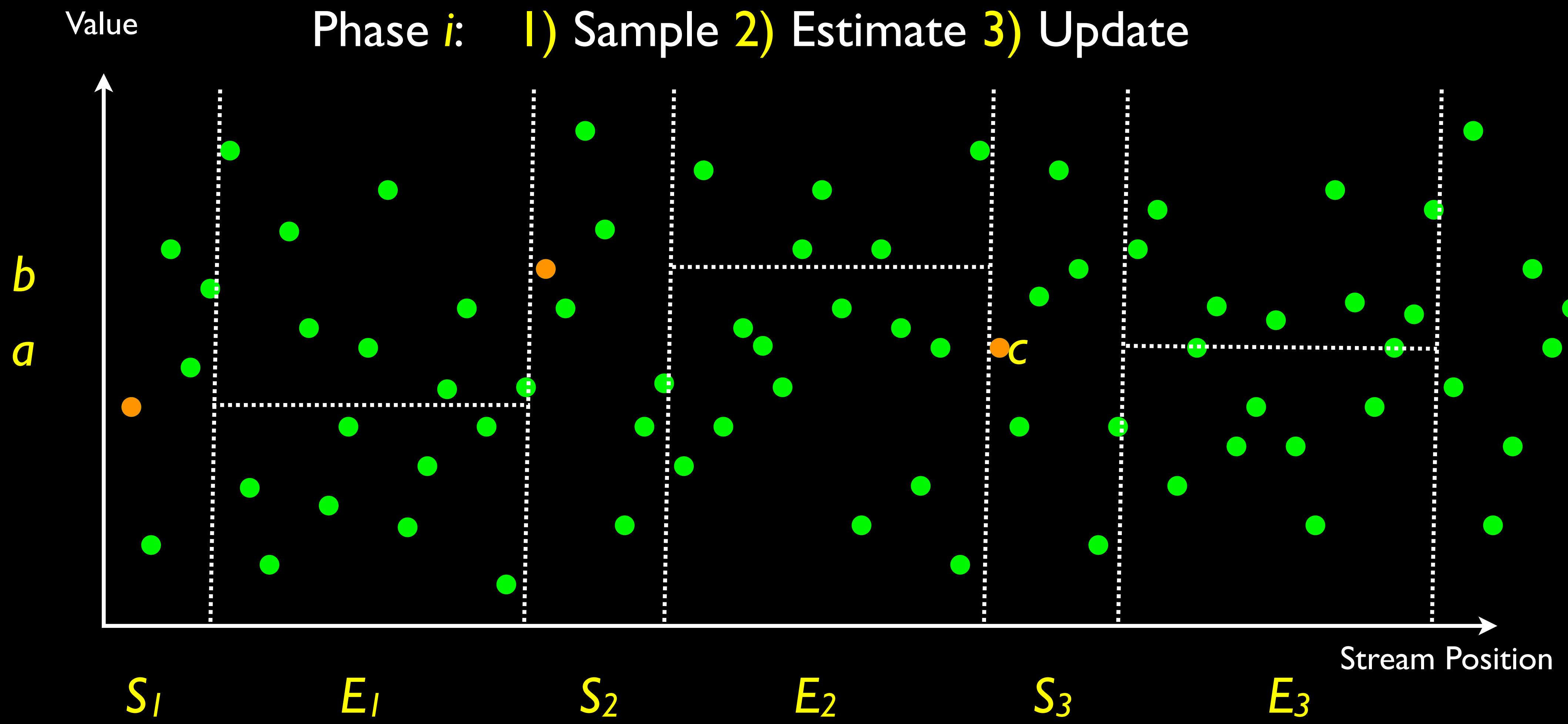
Phase *i*: 1) Sample 2) Estimate 3) Update



Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, ..., S_p, E_p$

Phase i: 1) Sample 2) Estimate 3) Update

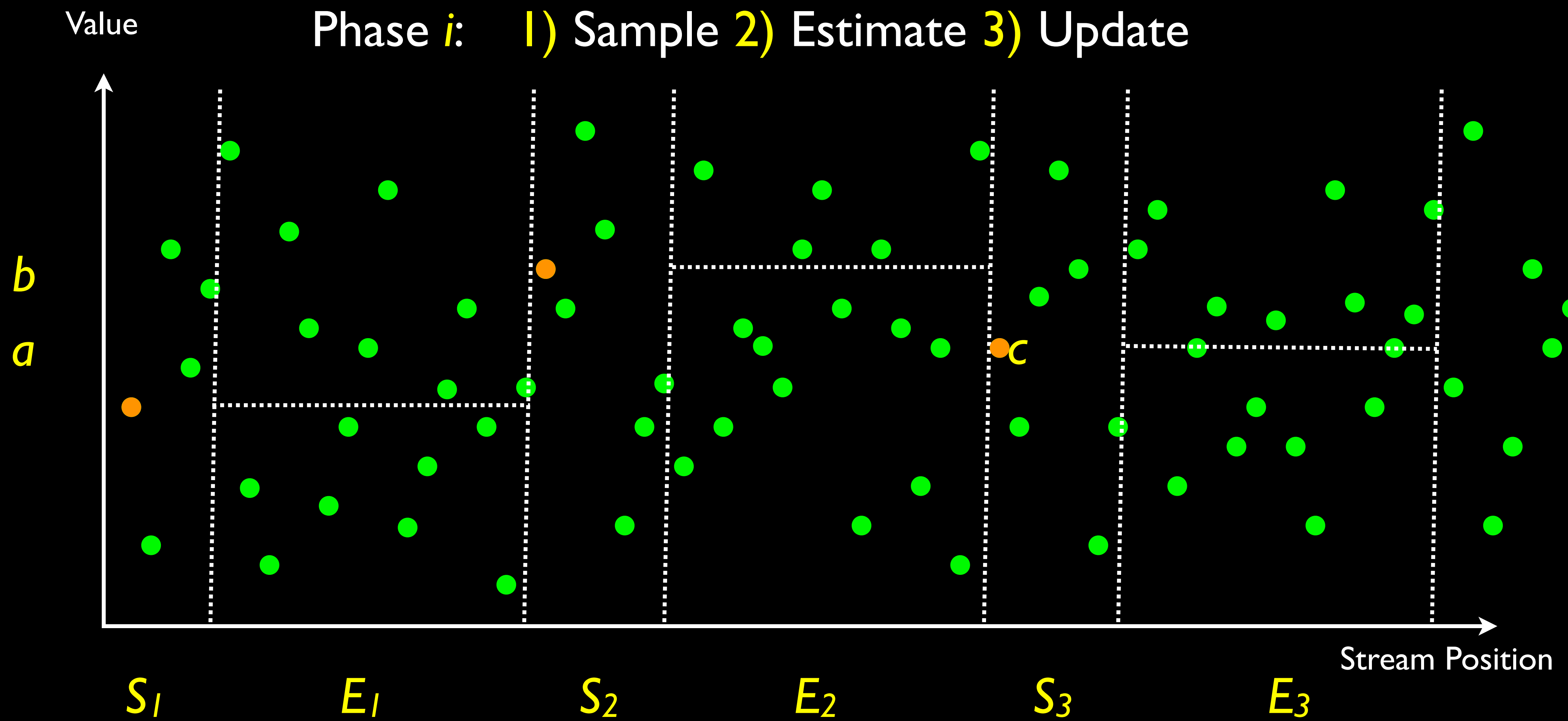


Analysis: If $|E_i|=O(\epsilon^{-2})$, we estimate $\mu(-\infty,c)$ up to $\pm\ \epsilon$.

Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, ..., S_p, E_p$

Phase *i*: 1) Sample 2) Estimate 3) Update



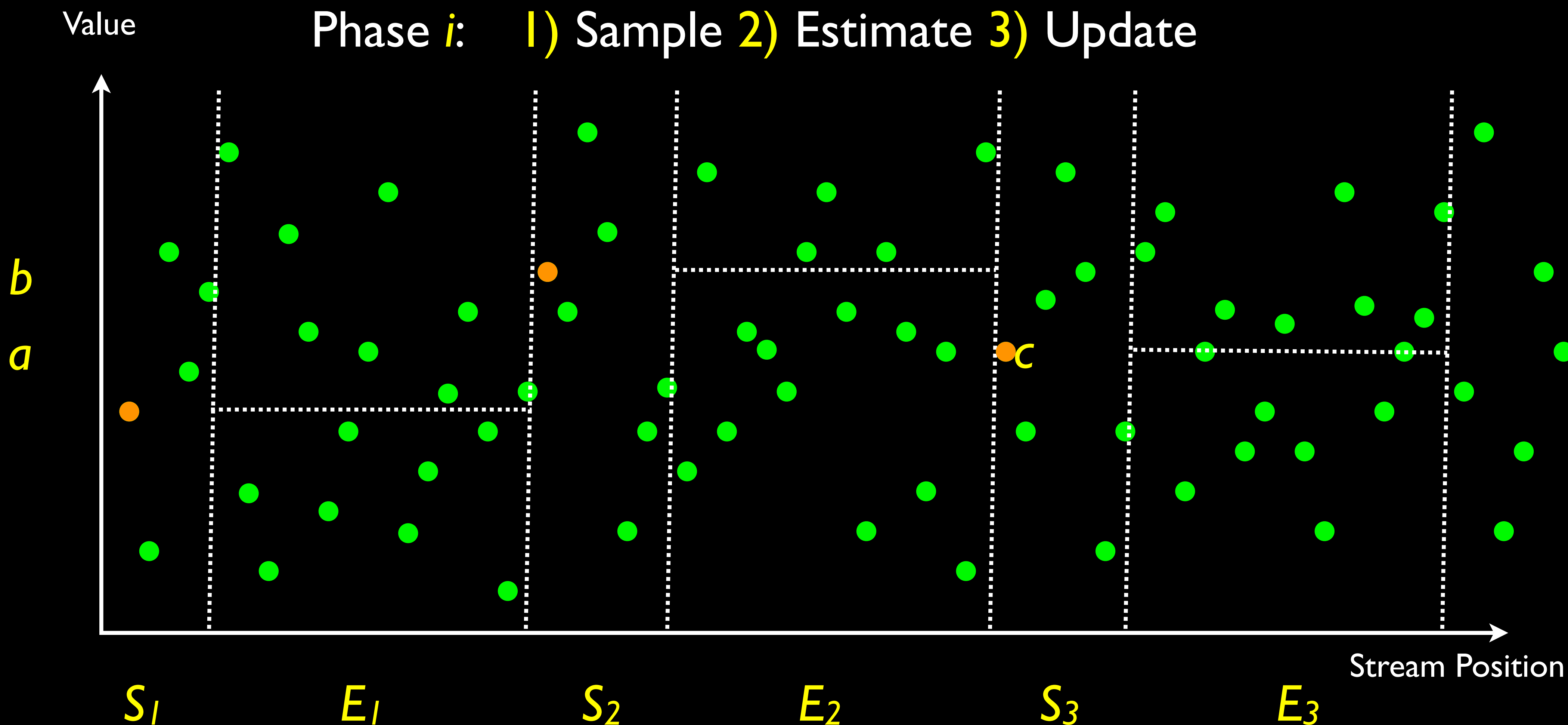
Analysis: If $|E_i|=O(\epsilon^{-2})$, we estimate $\mu(-\infty,c)$ up to $\pm\ \epsilon$.

If $\mu(-\infty,c) = 1/2 \pm \epsilon$, then return c.

Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

Phase *i*: 1) Sample 2) Estimate 3) Update



Analysis: If $|E_i|=O(\epsilon^{-2})$, we estimate $\mu(-\infty,c)$ up to $\pm\ \epsilon$.

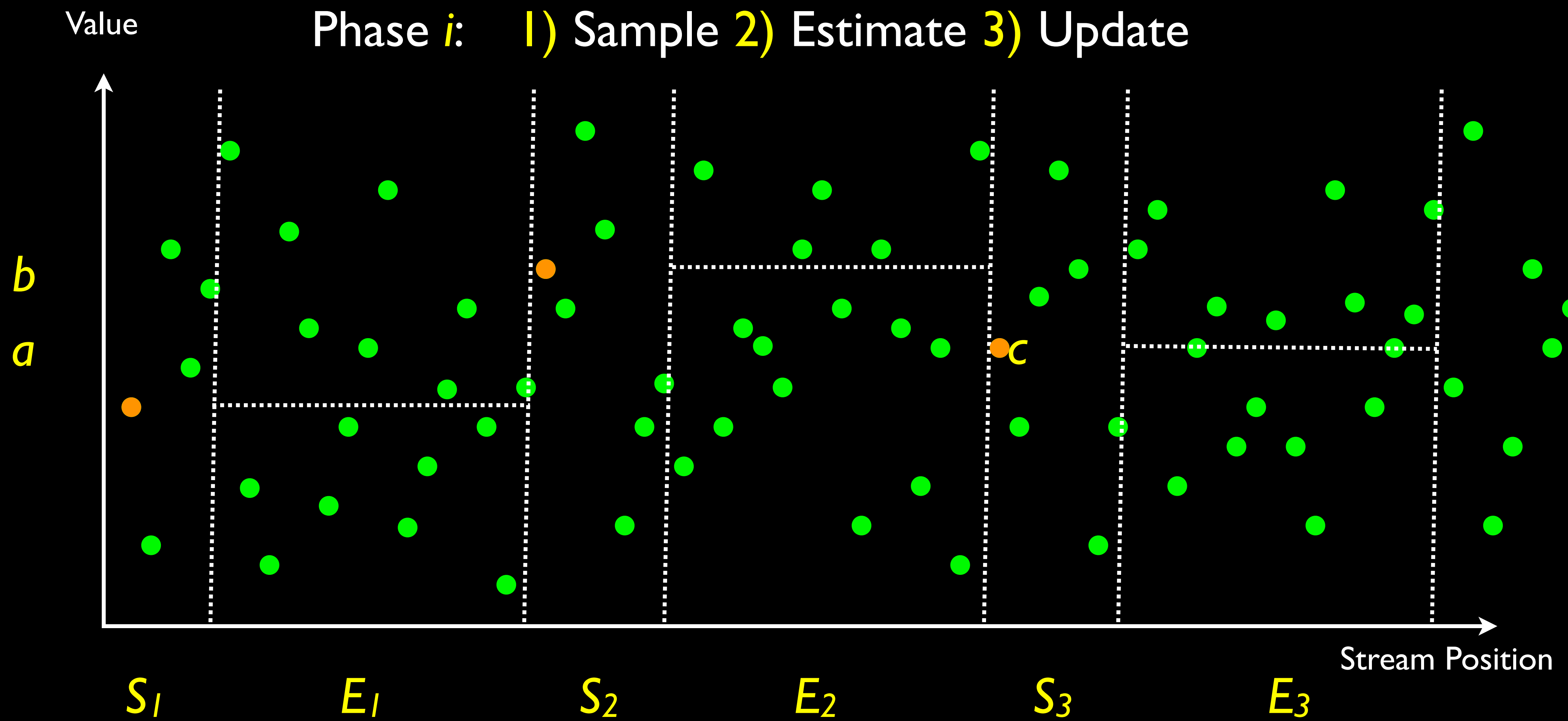
If $\mu(-\infty,c) = 1/2 \pm \epsilon$, then return c.

If $|S_i|=O(\epsilon^{-1})$ and $\mu(a,b)=\Omega(\epsilon^{-1})$, then can find c in [a,b].

Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

Phase i: 1) Sample 2) Estimate 3) Update



Analysis: If $|E_i|=O(\epsilon^{-2})$, we estimate $\mu(-\infty,c)$ up to $\pm\ \epsilon$.

If $\mu(-\infty,c) = 1/2 \pm \epsilon$, then return c.

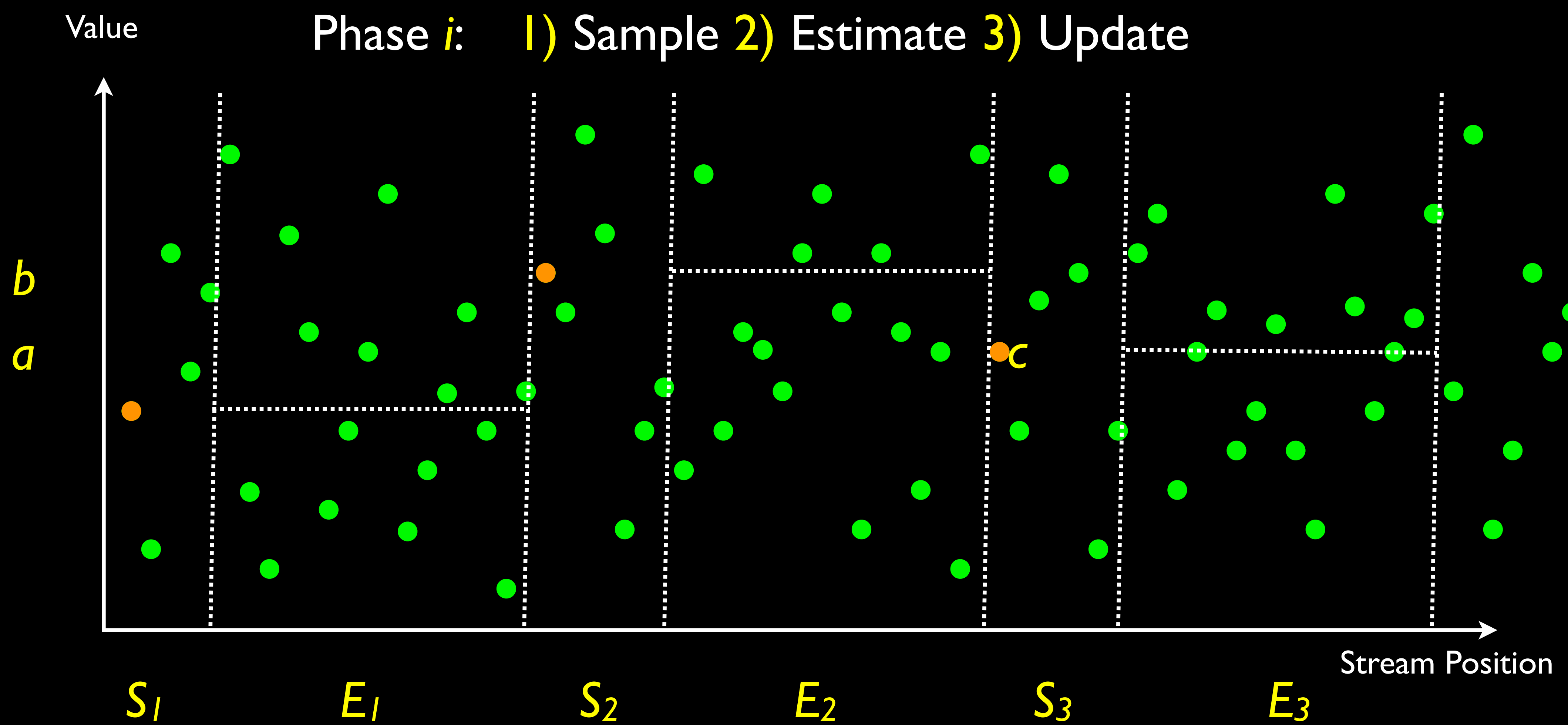
If $|S_i|=O(\epsilon^{-1})$ and $\mu(a,b)=\Omega(\epsilon^{-1})$, then can find c in [a,b].

Expect $\mu(a,b)$ to half in each phase, hence $p = O(\log \epsilon^{-1})$

Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

Phase i: 1) Sample 2) Estimate 3) Update

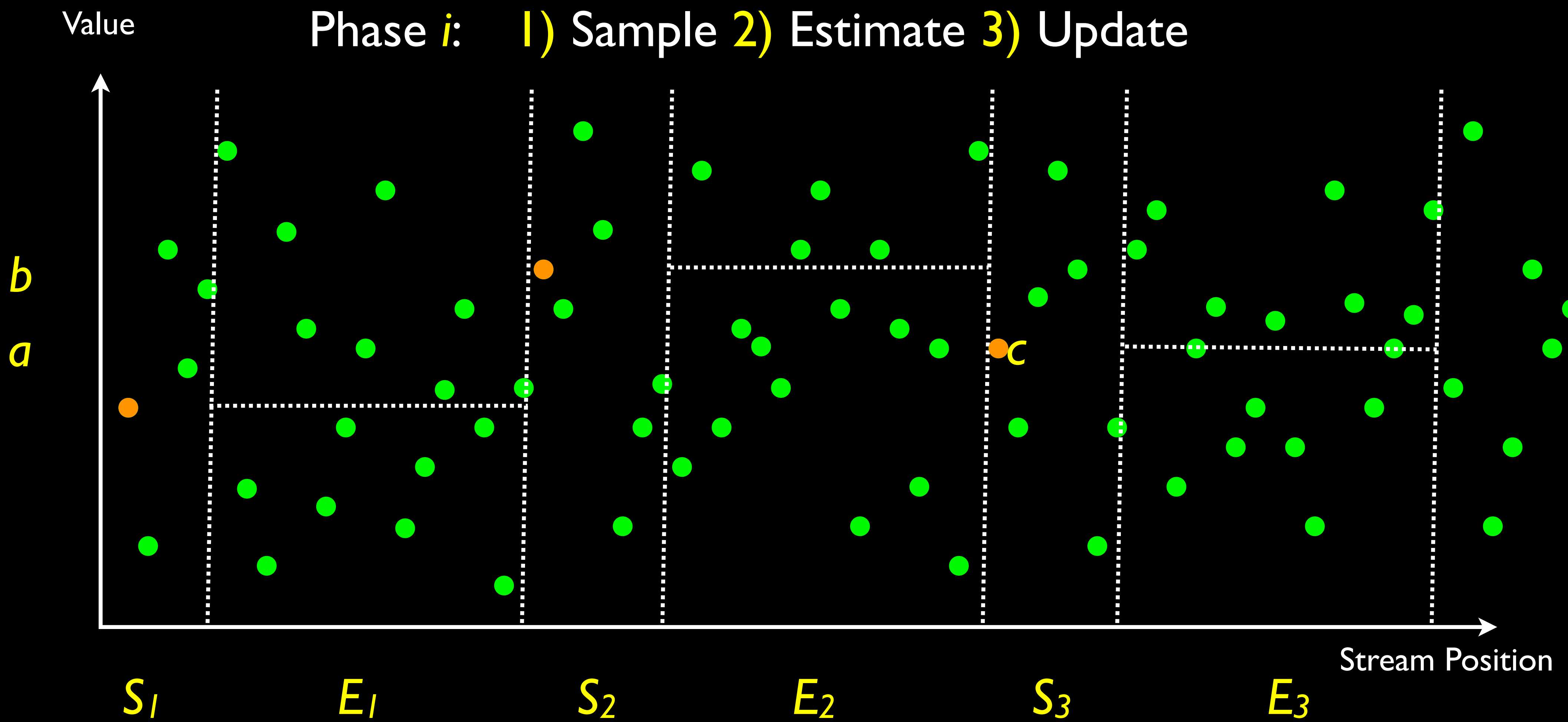


Thm: ϵ-approx median with $O(\epsilon^{-2}\log \epsilon^{-2})$ samples and $O(1)$ space

Algorithm: Maintain lower/upper bound [a, b] for median and c in [a,b]

Split stream in segments $S_1, E_1, S_2, E_2, \ldots, S_p, E_p$

Phase i: 1) Sample 2) Estimate 3) Update



Thm: ϵ-approx median with $O(\epsilon^{-2}\log \epsilon^{-2})$ samples and $O(1)$ space

Thm: Given a length m stream *in random* order, can return an element with rank $m/2 \pm O(m^{1/2} \log^2 m)$ using $O(1)$ space.

Alice

length n
binary string x

Bob

index i in
range [n]

Alice

length n
binary string x

INDEX: "What's the value of x_i?"

Requires $\Omega(n)$ bits transmitted.

Bob

index i in
range [n]

Alice

length n
binary string x

INDEX: “What’s the value of x_i?”

Requires $\Omega(n)$ bits transmitted.

Bob

index i in
range [n]

- Assume there exists single-pass algorithm returning the median with prob. at least 3/4 using S space.

$2+x_1$ ... $2i+x_i$ ... $2n+x_n$

Alice

length n

binary string x

INDEX: "What's the value of x_i?"

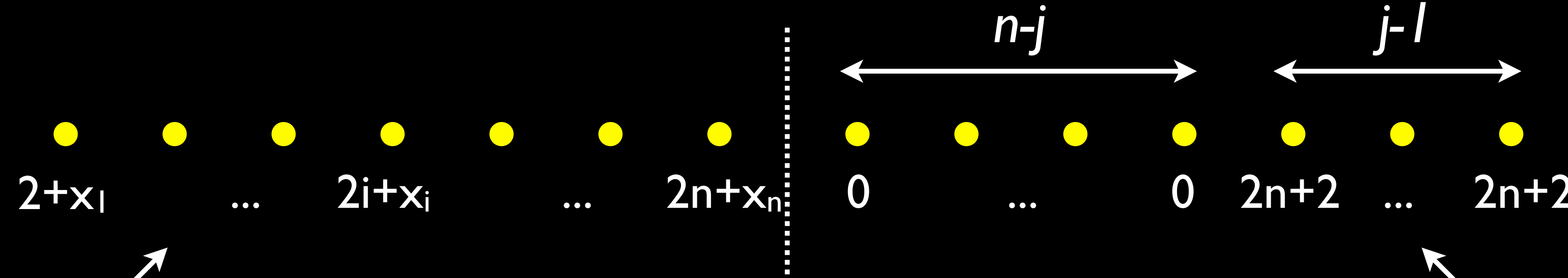
Requires $\Omega(n)$ bits transmitted.

Bob

index i in

range [n]

- Assume there exists single-pass algorithm returning the median with prob. at least 3/4 using S space.



Alice
length n
binary string x

INDEX: "What's the value of x_i?"

Requires $\Omega(n)$ bits transmitted.

Bob
index i in
range [n]

- Assume there exists single-pass algorithm returning the median with prob. at least 3/4 using S space.

n-j ⟷ *j-1*

2+x_1 … 2i+x_i … 2n+x_n | 0 … 0 2n+2 … 2n+2

Alice

length n

binary string x

INDEX: "What's the value of x_i?"

Requires Ω(n) bits transmitted.

MEMORY STATE OF ALGORITHM

Bob

index i in

range [n]

- Assume there exists single-pass algorithm returning the median with prob. at least 3/4 using S space.

n-j ←→ j-1 ←→

2+x_1 … 2i+x_i … 2n+x_n | 0 … 0 | 2n+2 … 2n+2

Alice
length n
binary string x

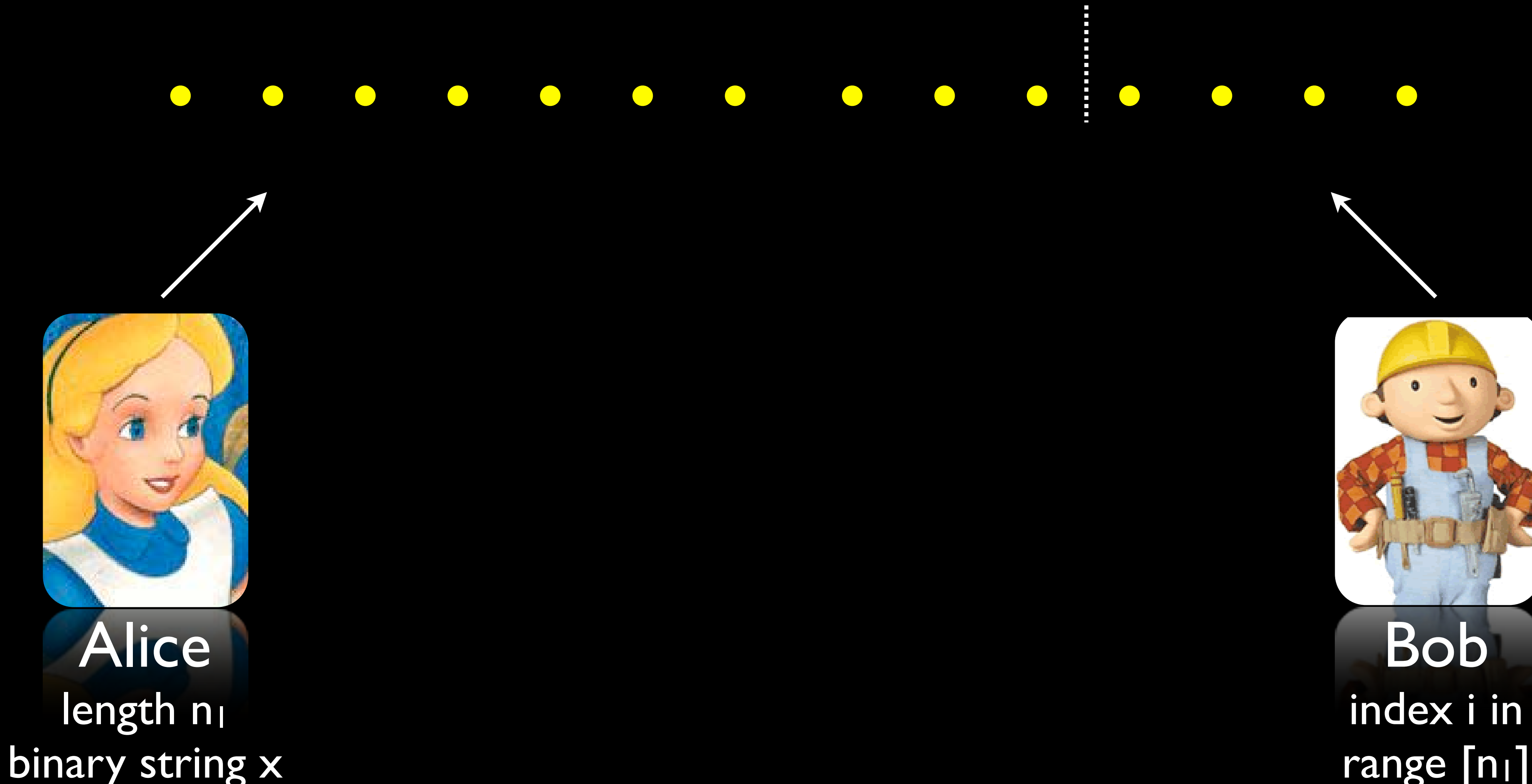
INDEX: "What's the value of x_i?"

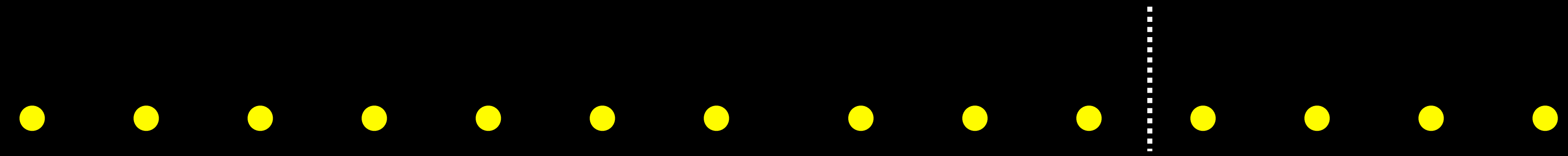
Requires Ω(n) bits transmitted.

MEMORY STATE OF ALGORITHM

Bob
index i in
range [n]

- Assume there exists single-pass algorithm returning the median with prob. at least 3/4 using S space.
- Thm: S=Ω(n) [Henzinger, Raghavan, and Rajagopalan '99]


Alice
length n_1
binary string x
Bob
index i in
range [n_1]



Alice: picks b randomly from $[n_1]$ and inserts a random permutation of,

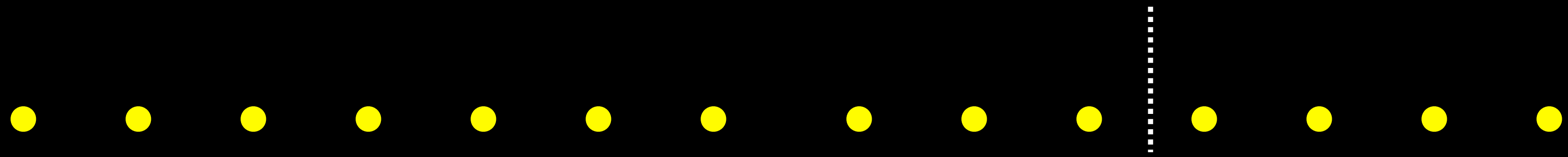
$$\{ \underbrace{0, \quad \ldots \quad, 0}_{(m-n_1-n_2-b)/2}, 2+x_1, \ldots, 2n_1+x_{n_1}, \underbrace{2n+2, \ldots, 2n+2}_{(m-n_1-n_2+b)/2} \}$$

Alice

length n_1 binary string x

Bob

index i in range $[n_1]$



Alice: picks b randomly from $[n_1]$ and inserts a random permutation of,

$$\{\underbrace{0, \quad \ldots \quad, 0}_{(m-n_1-n_2-b)/2}, 2+x_1, \ldots, 2n_1+x_{n_1}, \underbrace{2n+2, \ldots, 2n+2}_{(m-n_1-n_2+b)/2}\}$$

MEMORY STATE OF ALGORITHM and "b"

Alice

length n_1 binary string x

Bob

index i in range $[n_1]$

Alice: picks b randomly from $[n_1]$ and inserts a random permutation of,

$$\{ \underbrace{0, \quad \ldots \quad, 0}_{(m-n_1-n_2-b)/2}, 2+x_1, \ldots, 2n_1+x_{n_1}, \underbrace{2n+2, \ldots, 2n+2}_{(m-n_1-n_2+b)/2} \}$$

Alice

length n_1

binary string x

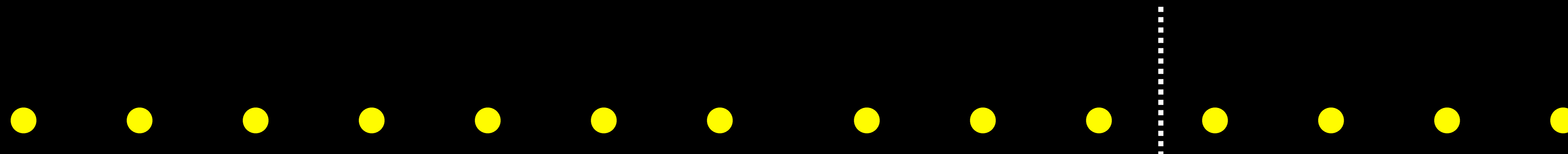
MEMORY STATE OF ALGORITHM and "b"

Bob: inserts a random permutation of,

$$\{ \underbrace{0, \quad \ldots \quad, 0}_{(n_2+b-j-1)/2}, \underbrace{2n+2, \ldots, 2n+2}_{(n_2-b+j)/2} \}$$

Bob

index i in range $[n_1]$



Alice

length n_1

binary string x

Alice: picks b randomly from $[n_1]$ and inserts a random permutation of,

$$\{\underbrace{0, \quad \ldots \quad ,0}_{(m-n_1-n_2-b)/2}, 2+x_1, \ldots, 2n_1+x_{n_1}, \underbrace{2n+2, \ldots, 2n+2}_{(m-n_1-n_2+b)/2}\}$$

MEMORY STATE OF ALGORITHM and "b"

Bob: inserts a random permutation of,

$$\{\underbrace{0, \quad \ldots \quad ,0}_{(n_2+b-j-1)/2}, \underbrace{2n+2, \ldots, 2n+2}_{(n_2-b+j)/2}\}$$

Bob

index i in range $[n_1]$

- If $n_1=o(\sqrt{n})$ and $n_1 n_2/n=o(1)$ then ordering is "close" to random.

Alice: picks b randomly from $[n_1]$ and inserts a random permutation of,

$$\{\underbrace{0, \ldots, 0}_{(m-n_1-n_2-b)/2}, 2+x_1, \ldots, 2n_1+x_{n_1}, \underbrace{2n+2, \ldots, 2n+2}_{(m-n_1-n_2+b)/2}\}$$

MEMORY STATE OF ALGORITHM and “b”

Bob: inserts a random permutation of,

$$\{\underbrace{0, \ldots, 0}_{(n_2+b-j-1)/2}, \underbrace{2n+2, \ldots, 2n+2}_{(n_2-b+j)/2}\}$$

Alice

length n_1 binary string x

Bob

index i in range $[n_1]$

- If $n_1 = o(\sqrt{n})$ and $n_1 n_2 / n = o(1)$ then ordering is “close” to random.
- An algorithm succeeding w/p 3/4 for random-ordering, succeeds w/p 2/3.

Alice: picks b randomly from $[n_1]$ and inserts a random permutation of,

$$\{\underbrace{0, \quad \ldots \quad, 0}_{(m-n_1-n_2-b)/2}, 2+x_1, \ldots, 2n_1+x_{n_1}, \underbrace{2n+2, \ldots, 2n+2}_{(m-n_1-n_2+b)/2}\}$$

MEMORY STATE OF ALGORITHM and "b"

Bob: inserts a random permutation of,

$$\{\underbrace{0, \quad \ldots \quad, 0}_{(n_2+b-j-1)/2}, \underbrace{2n+2, \ldots, 2n+2}_{(n_2-b+j)/2}\}$$

Alice

length n_1 binary string x

Bob

index i in range $[n_1]$

- If $n_1=o(\sqrt{n})$ and $n_1 n_2/n=o(1)$ then ordering is "close" to random.
- An algorithm succeeding w/p 3/4 for random-ordering, succeeds w/p 2/3.
- **Thm:** $S=\Omega(n^{1/3})$

Alice: picks b randomly from $[n_1]$ and inserts a random permutation of,

$$\{\underbrace{0, \quad \ldots \quad, 0}_{(m-n_1-n_2-b)/2}, 2+x_1, \ldots, 2n_1+x_{n_1}, \underbrace{2n+2, \ldots, 2n+2}_{(m-n_1-n_2+b)/2}\}$$

MEMORY STATE OF ALGORITHM and "b"

Alice

length n_1

binary string x

Bob

index i in range $[n_1]$

Bob: inserts a random permutation of,

$$\{\underbrace{0, \quad \ldots \quad, 0}_{(n_2+b-j-1)/2}, \underbrace{2n+2, \ldots, 2n+2}_{(n_2-b+j)/2}\}$$

- If $n_1=o(\sqrt{n})$ and $n_1 n_2/n=o(1)$ then ordering is "close" to random.
- An algorithm succeeding w/p 3/4 for random-ordering, succeeds w/p 2/3.
- **Thm:** $S=\Omega(n^{1/2})$ [Guha, McGregor '06]

1. Models
2. Quantiles
3. Learning Distributions

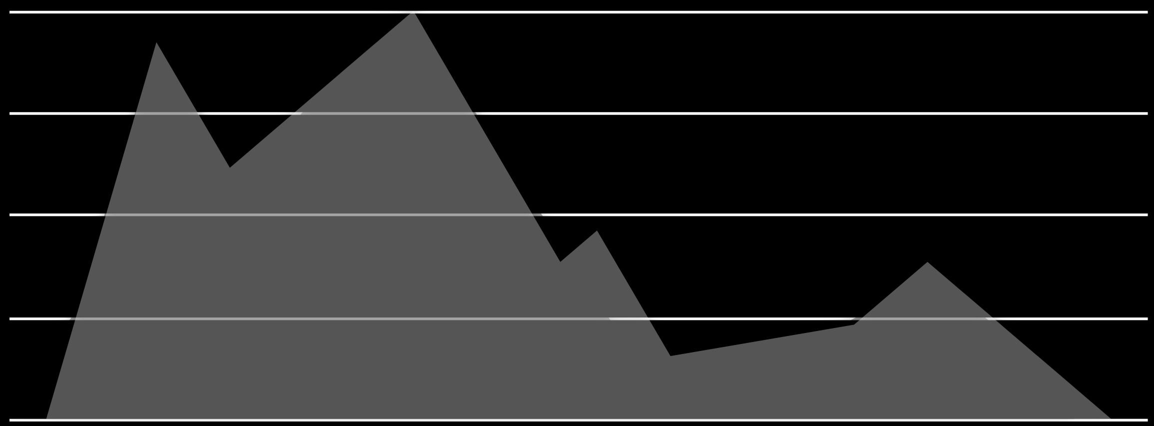
Learning Distributions

Learning Distributions

- Stream: m samples a distribution with k piece-wise linear density function μ

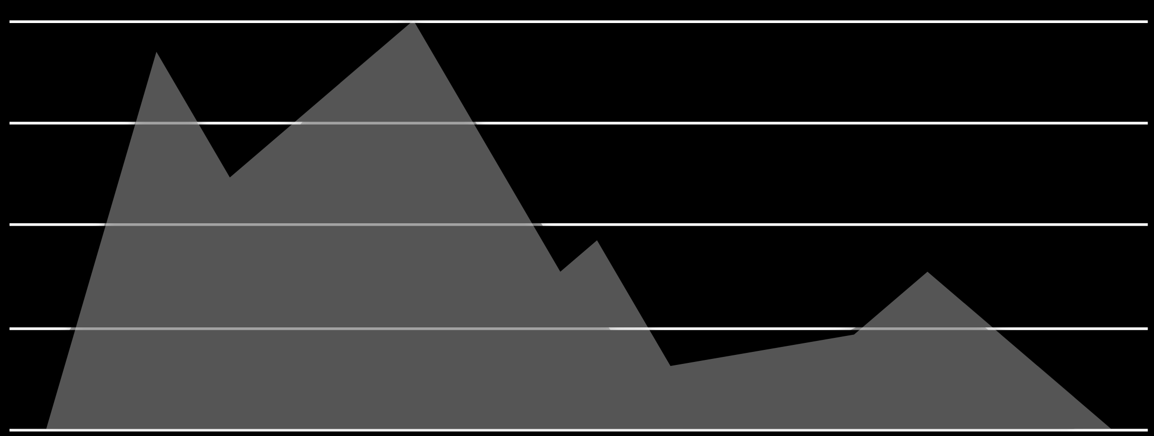
Learning Distributions

- Stream: m samples a distribution with k piece-wise linear density function μ



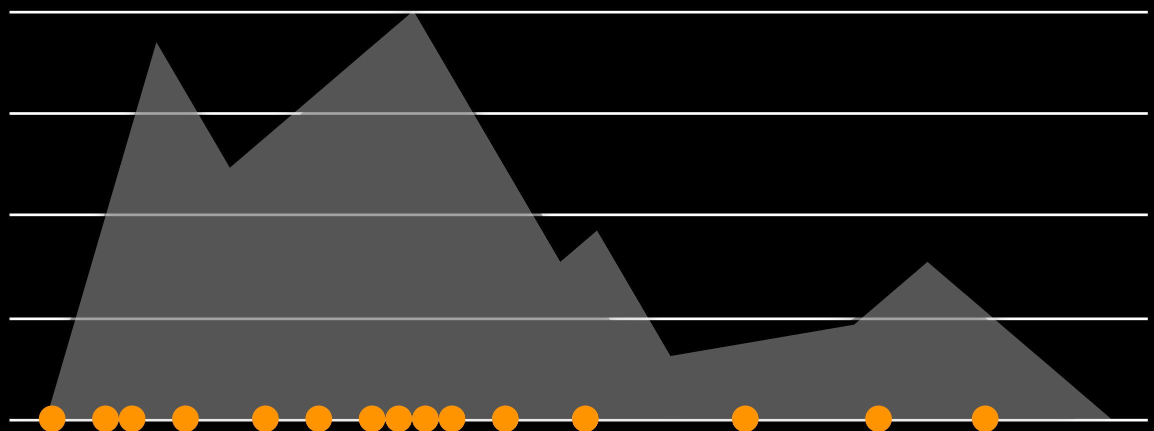
Learning Distributions

- Stream: m samples a distribution with k piece-wise linear density function μ
- Goal: Find k piece-wise linear density function μ' such that $|\mu - \mu'| < \epsilon$



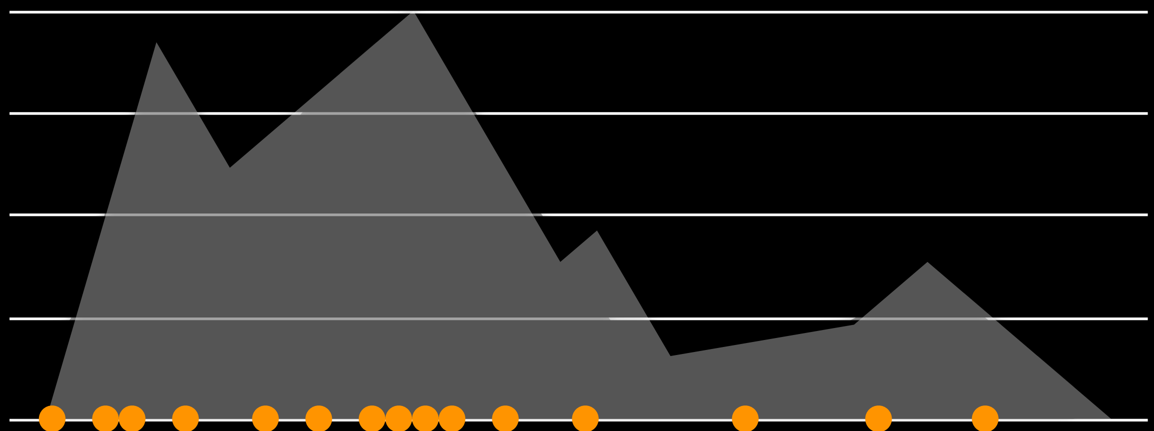
Learning Distributions

- Stream: m samples a distribution with k piece-wise linear density function μ
- Goal: Find k piece-wise linear density function μ' such that |μ-μ'| < ϵ



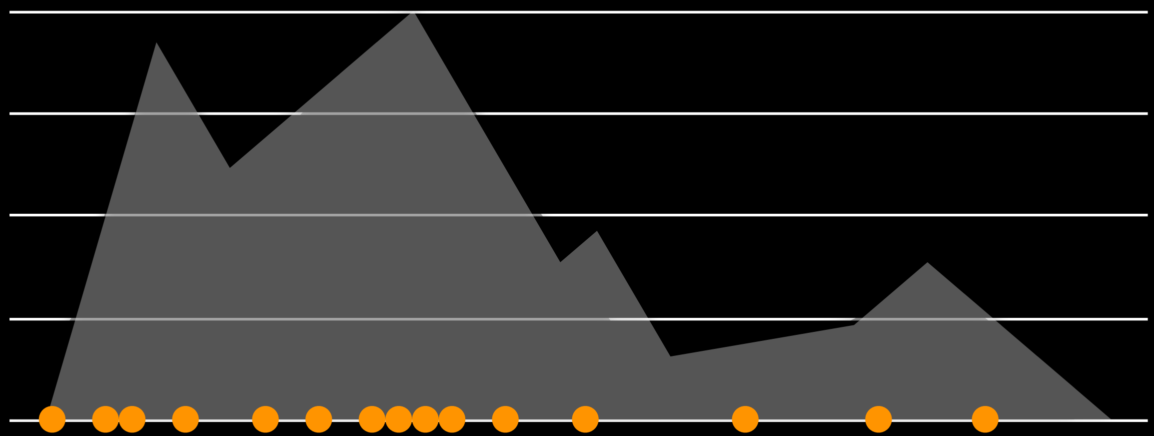
Learning Distributions

- Stream: m samples a distribution with k piece-wise linear density function μ
- Goal: Find k piece-wise linear density function μ' such that |μ-μ'| < ϵ
- Thm: $O(k^6 \epsilon^{-6})$ samples and $O(k^3 \epsilon^{-2/p})$ space with p passes. [Chang, Kannan '06]

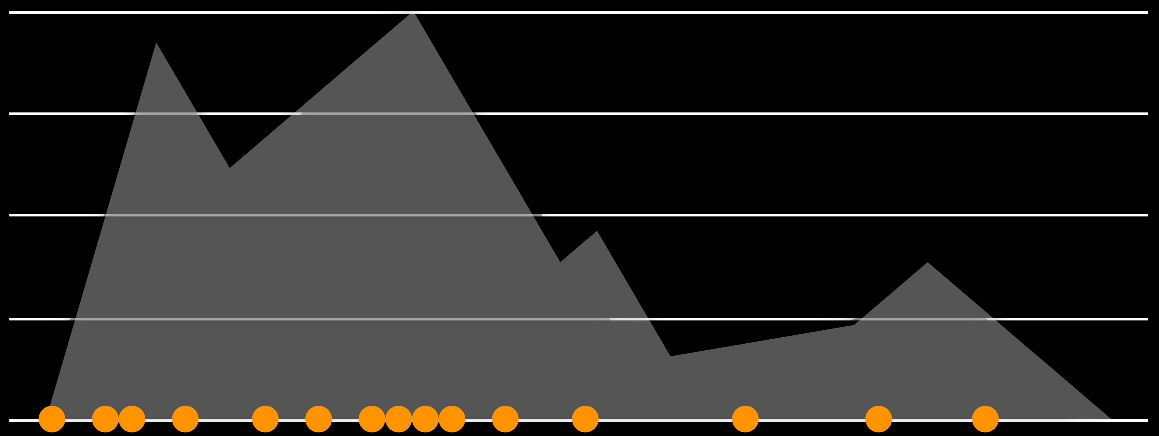


Learning Distributions

- Stream: m samples a distribution with k piece-wise linear density function μ
- Goal: Find k piece-wise linear density function μ' such that $|\mu-\mu'| < \epsilon$
- Thm: $O(k^6 \epsilon^{-6})$ samples and $O(k^3 \epsilon^{-2/p})$ space with p passes. [Chang, Kannan '06]
- Thm: $O(k^2 \epsilon^{-4})$ samples and $O(k)$ space with one pass. [Guha, McGregor '06]

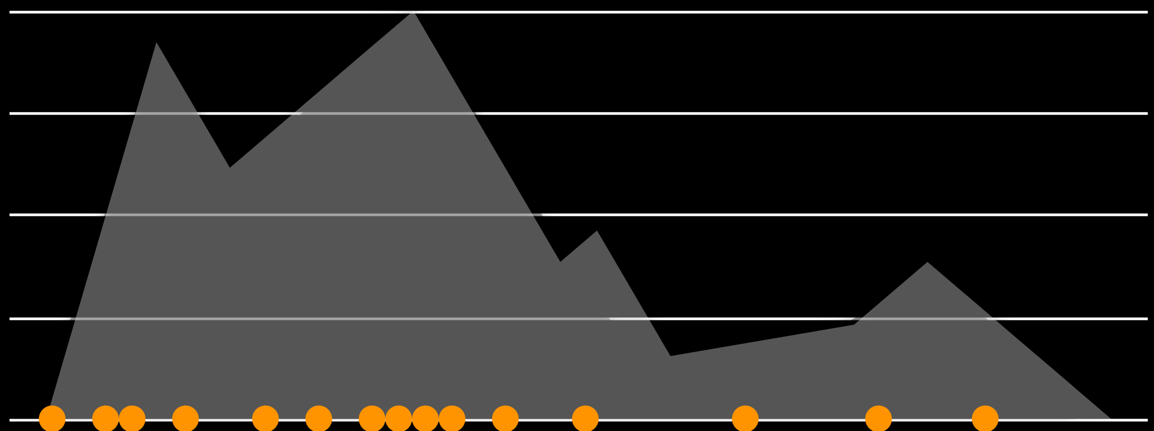


Learning Distributions



Learning Distributions

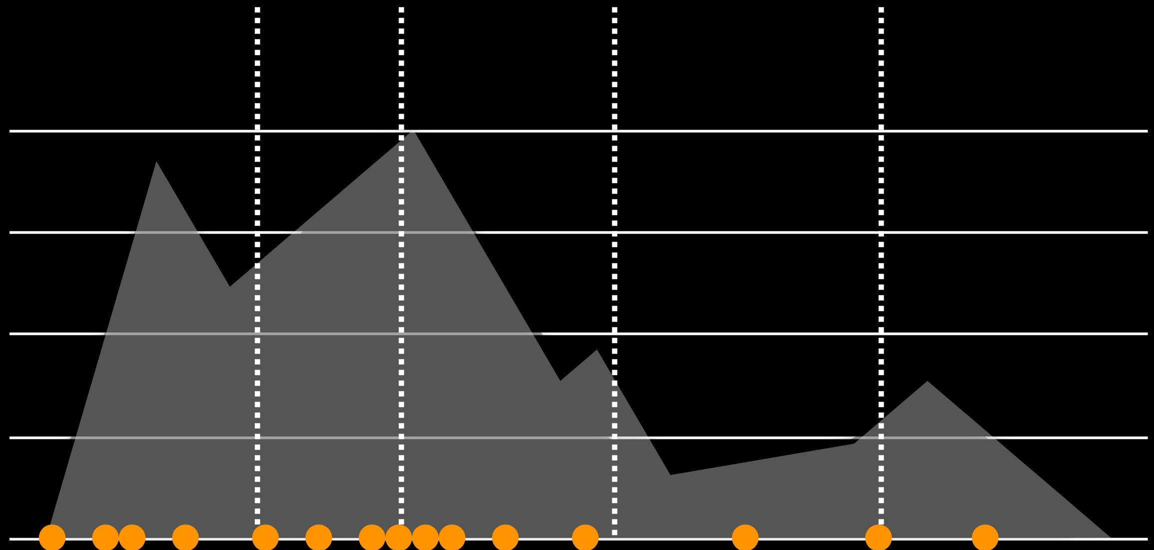
- Split into t_l intervals of approx equal mass



Learning Distributions

- Split into t_l intervals of approx equal mass

  $O(1/t_l^2)$ samples and quantile algorithm



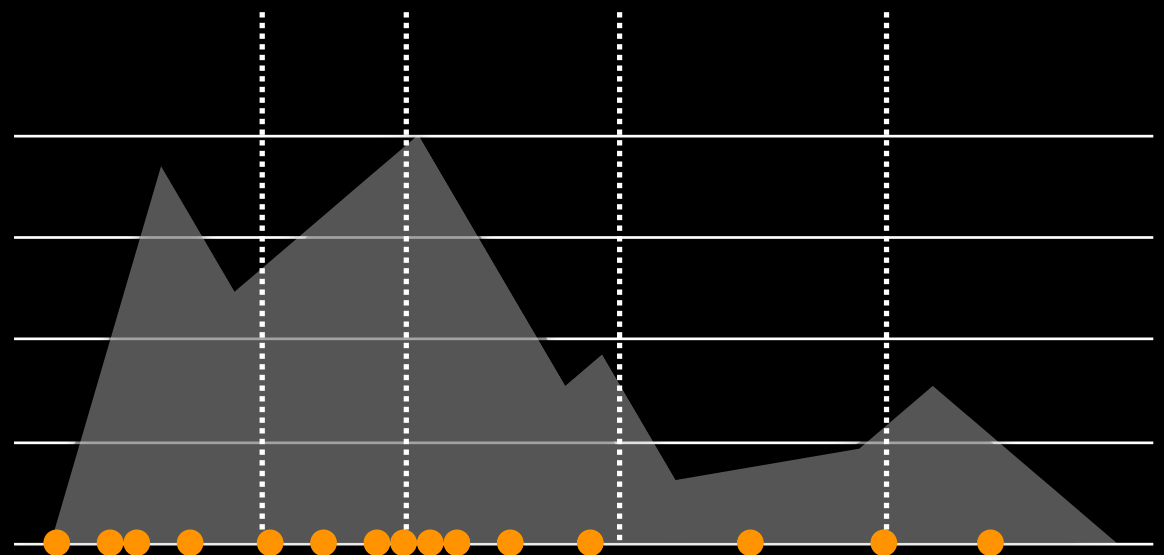
Learning Distributions

- Split into t_l intervals of approx equal mass

  $O(1/t_l^2)$ samples and quantile algorithm

- Test if μ conditioned on each interval [a,b] is β-far from linear

  $O(k/\beta^3)/\mu(a,b)$ samples, quantize, use L_1-sketch



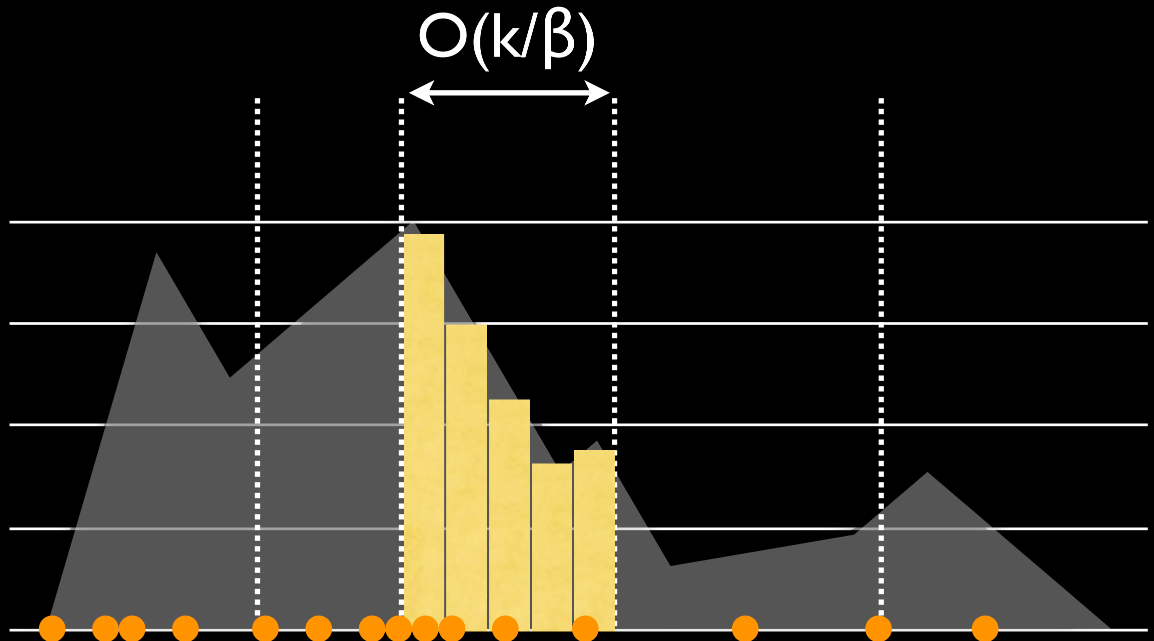
Learning Distributions

- Split into t_1 intervals of approx equal mass

  $O(1/t_1^2)$ samples and quantile algorithm

- Test if μ conditioned on each interval [a,b] is β-far from linear

  $O(k/\beta^3)/\mu(a,b)$ samples, quantize, use L_1-sketch



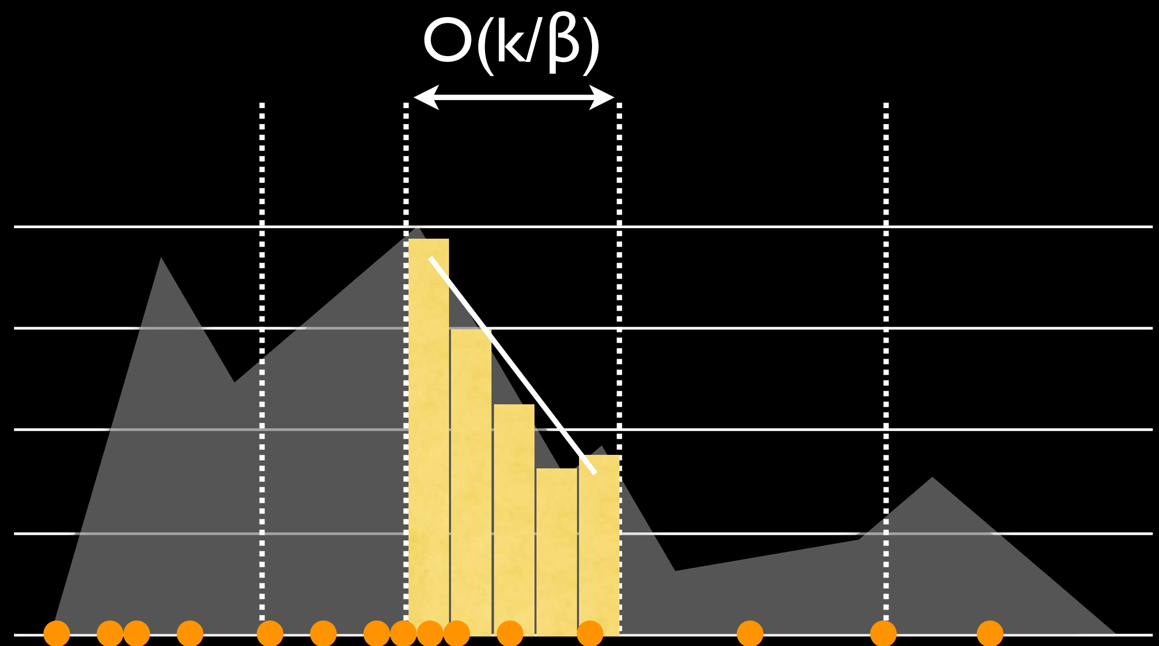
Learning Distributions

- Split into t_1 intervals of approx equal mass

  $O(1/t_1^2)$ samples and quantile algorithm

- Test if μ conditioned on each interval [a,b] is β-far from linear

  $O(k/\beta^3)/\mu(a,b)$ samples, quantize, use L_1-sketch



Learning Distributions

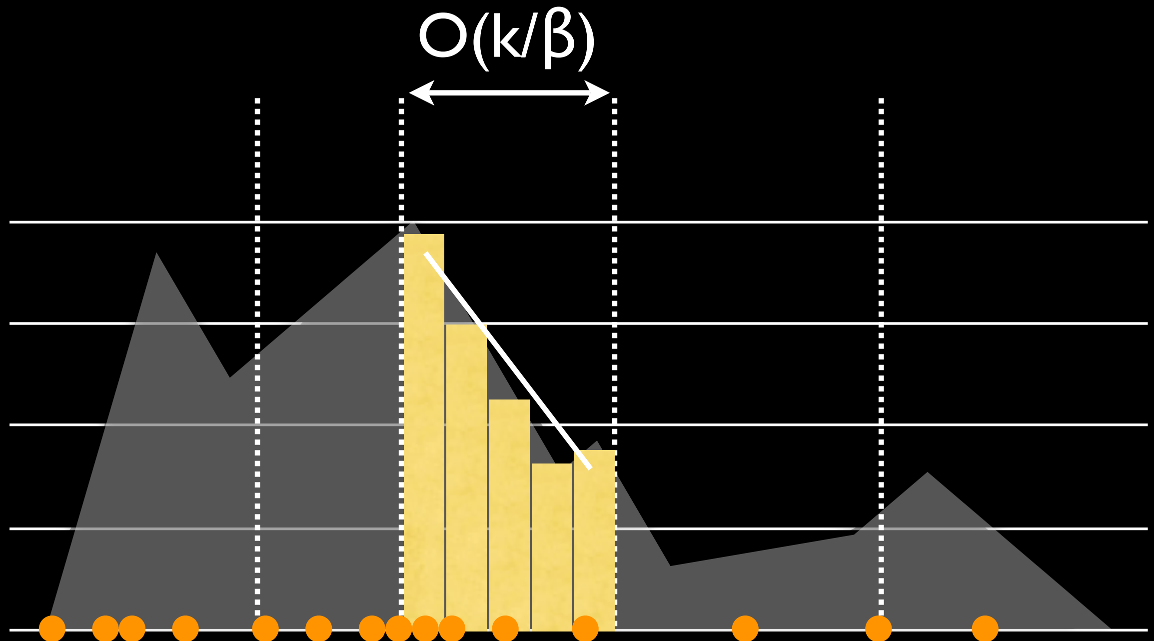
- Split into t_1 intervals of approx equal mass

  $O(1/t_1^2)$ samples and quantile algorithm

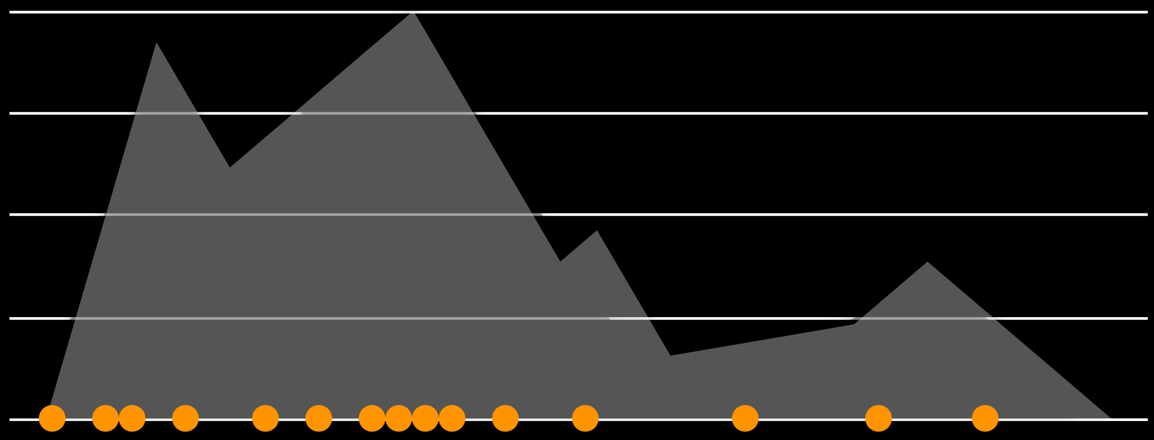
- Test if μ conditioned on each interval [a,b] is β-far from linear

  $O(k/\beta^3)/\mu(a,b)$ samples, quantize, use L_1-sketch

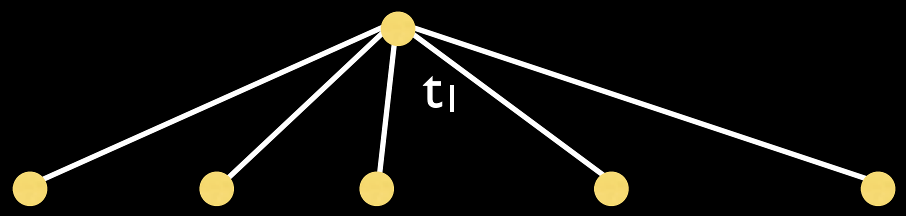
- Recurse on each non-linear interval

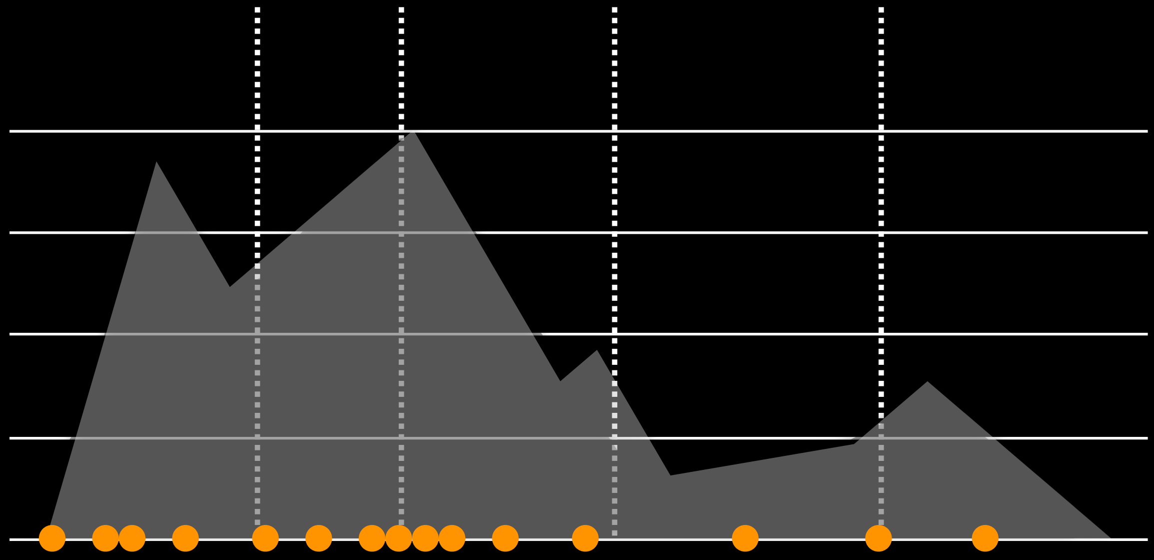


Learning Distributions

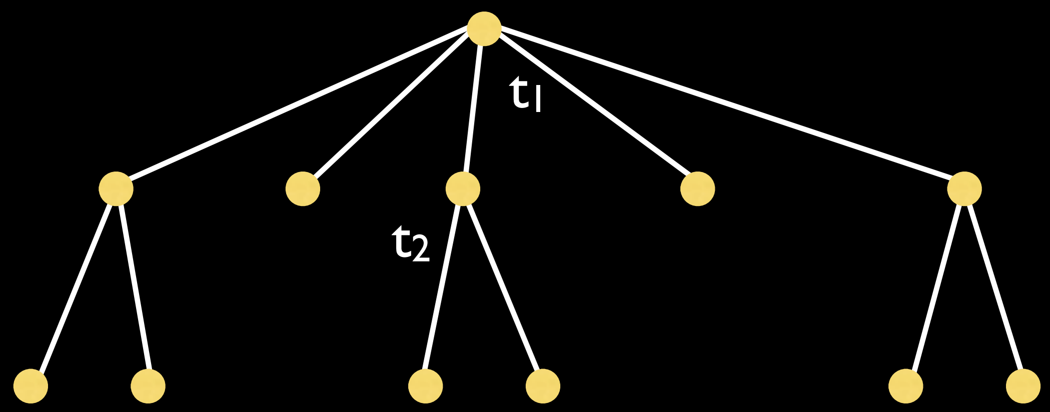


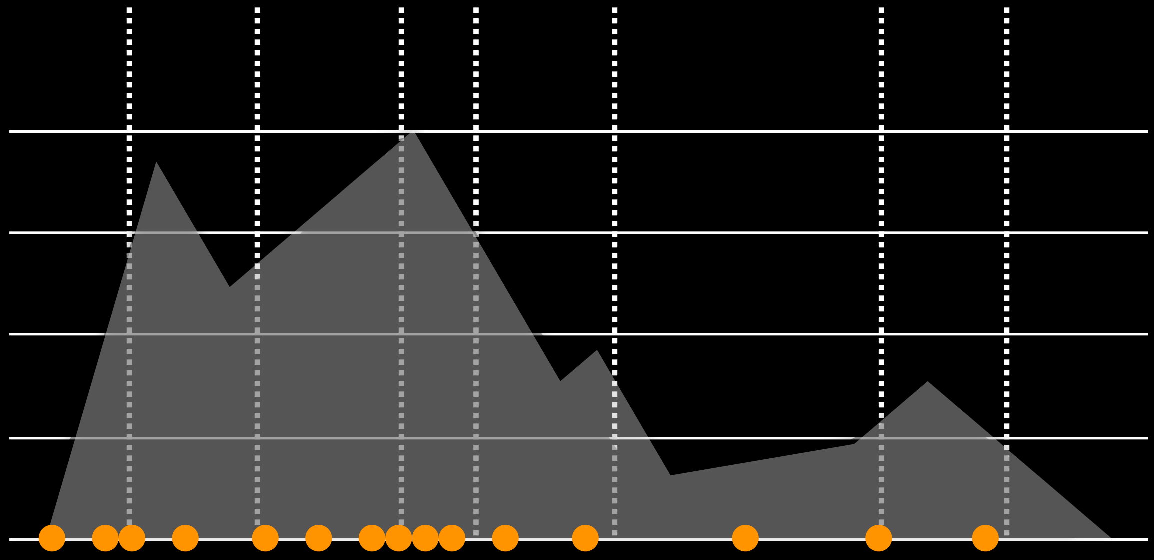
Learning Distributions



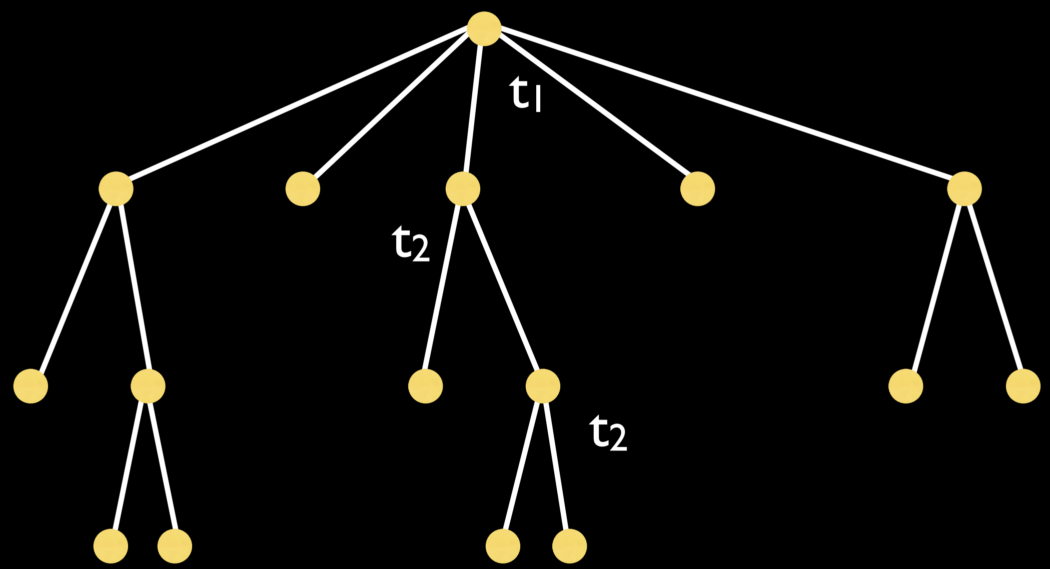


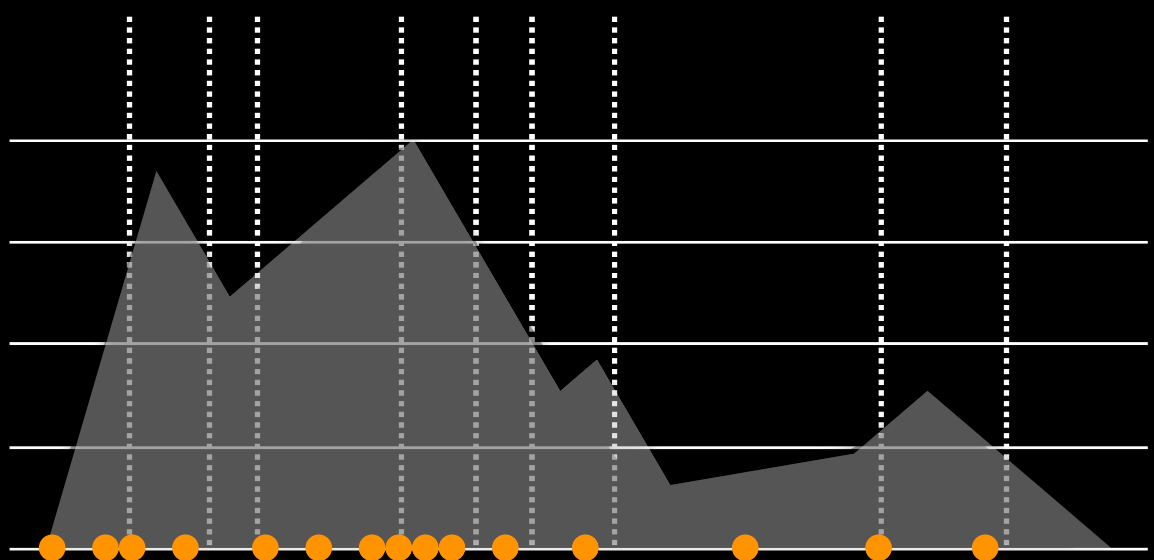
Learning Distributions



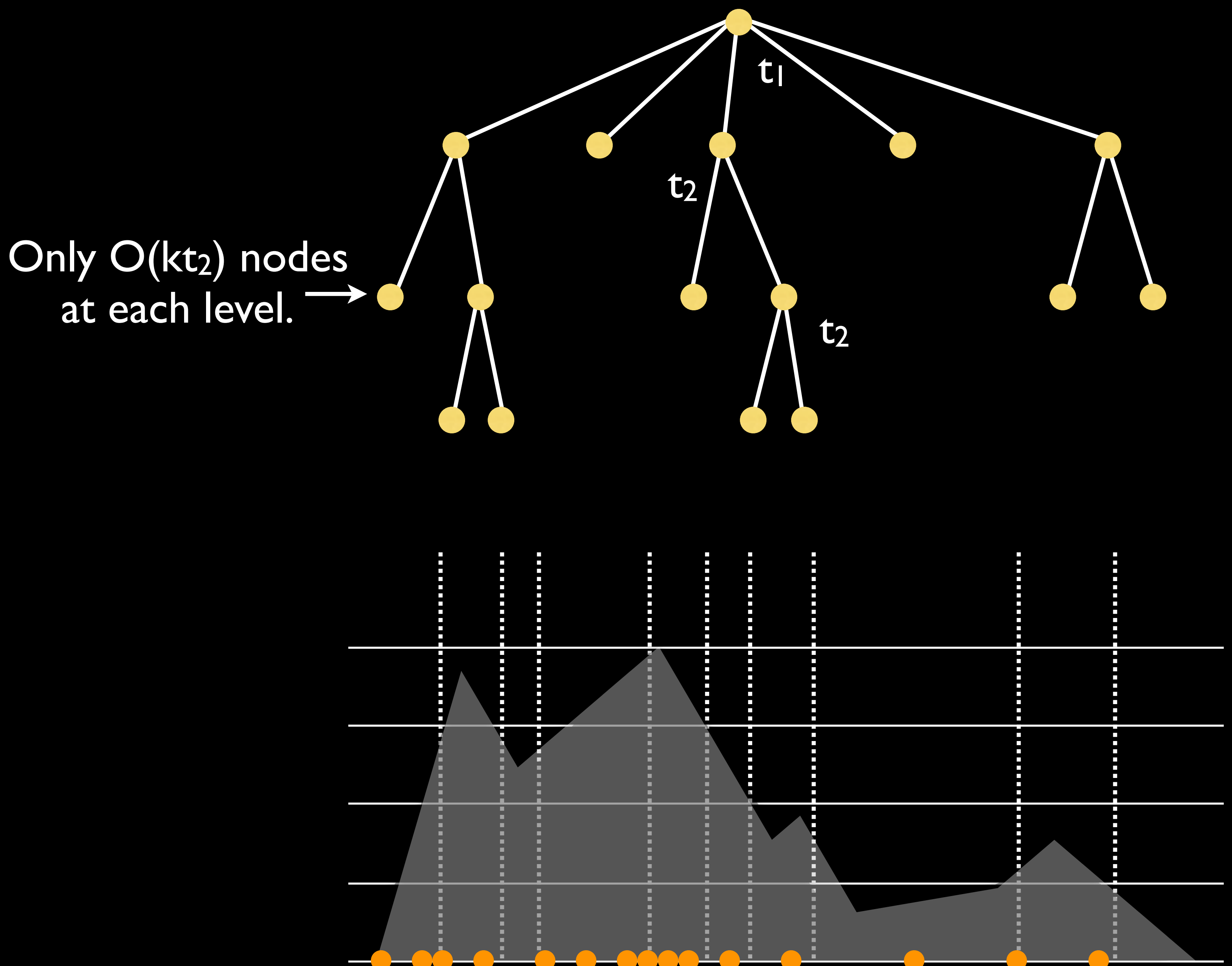


Learning Distributions

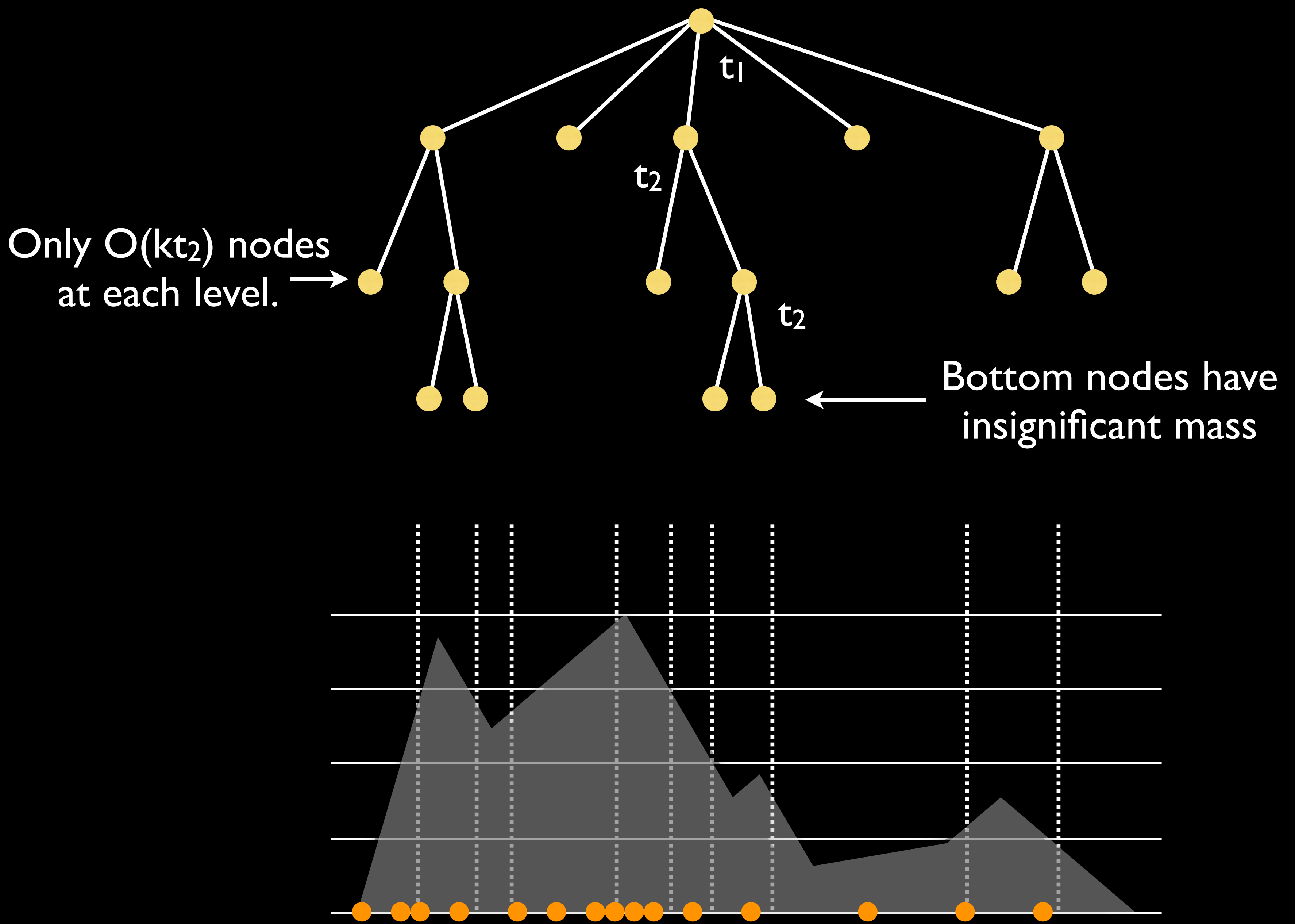




Learning Distributions



Learning Distributions



Summary

	Chang, Kannan '06	
Samples	$O(k^2 \epsilon^{-2})$	$O(k^6 \epsilon^{-6})$
Space	$O(k^2 \epsilon^{-2})$	$O(k^3 \epsilon^{-2/p})$
Passes	1	p
Re-order?	✔	✔

Summary

	Chang, Kannan '06		Guha, McGregor '06	
Samples	$O(k^2\ \epsilon^{-2})$	$O(k^6\ \epsilon^{-6})$	$O(k^2\ \epsilon^{-4})$	$O(k^2\ \epsilon^{-4})$
Space	$O(k^2\ \epsilon^{-2})$	$O(k^3\ \epsilon^{-2/p})$	$O(k)$	$O(k\ \epsilon^{-2/p})$
Passes	1	p	1	p
Re-order?	✔	✔	✖	✔

a) Estimating "*stochastic properties*" such as entropy and *f*-Divergences.

a) Estimating "*stochastic properties*" such as entropy and *f*-Divergences.

b) Estimating expected values of aggregate properties given a "*probabilistic stream*."

a) Estimating "*stochastic properties*" such as entropy and *f*-Divergences.

b) Estimating expected values of aggregate properties given a "*probabilistic stream*."

c) Learning about the source of a stream, i.e. "*upstream algorithms*."

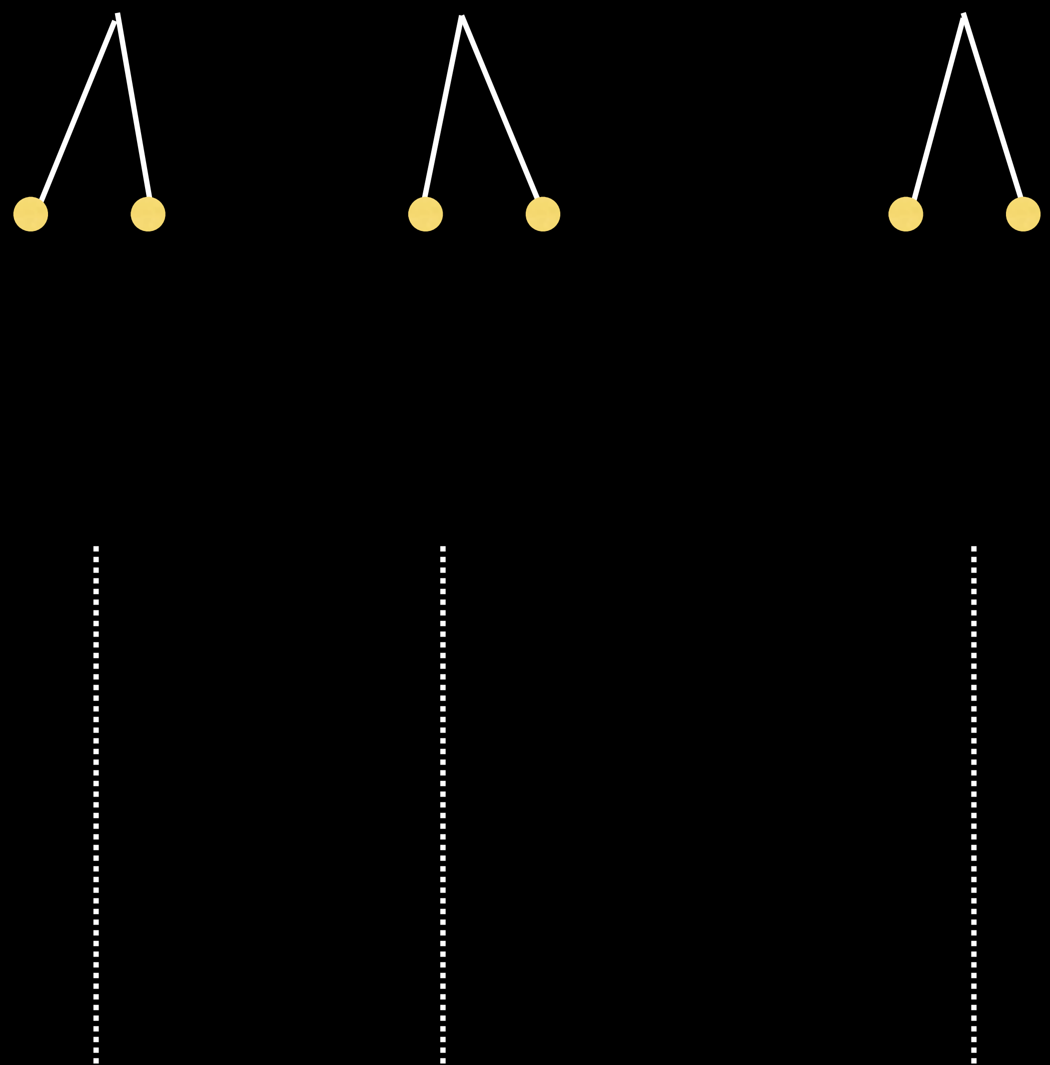
Examples:

sequence reconstruction, stream of iid samples, etc?

Algorithms:

quantiles/sufficient-statistics, piecewise-linear distributions, etc?

Questions?



1. Models
2. Quantiles
3. Learning Distributions
4. Forgettron