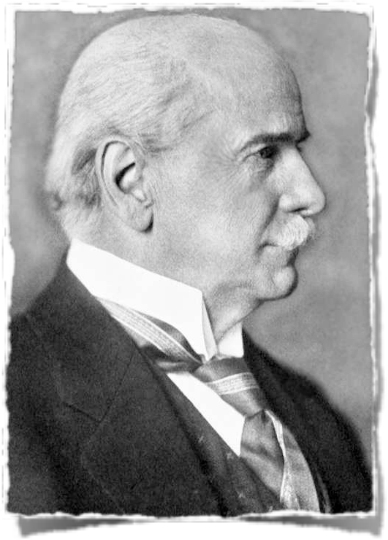
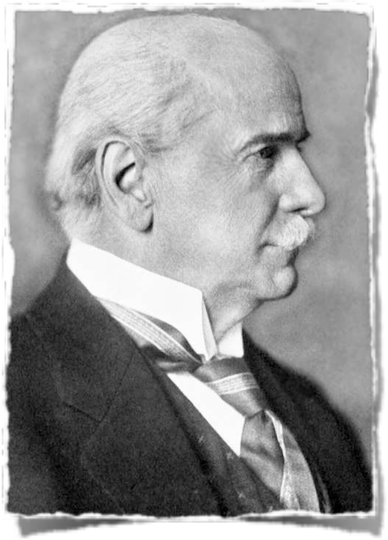
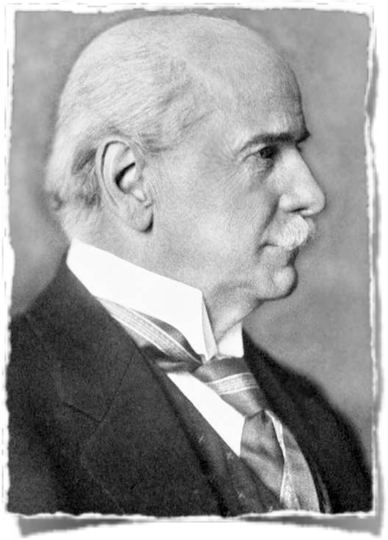


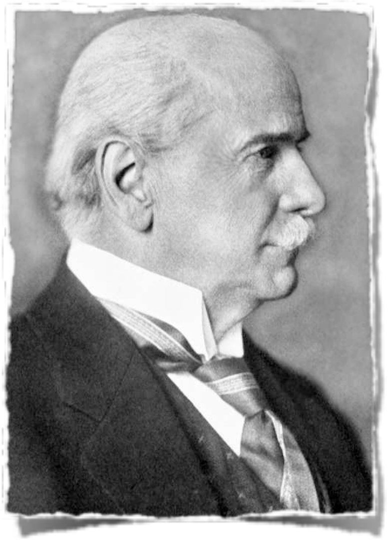


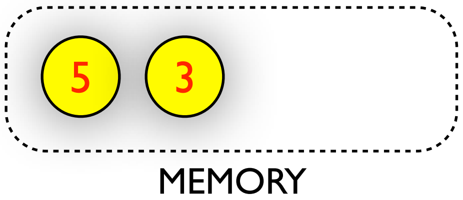
- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:



- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

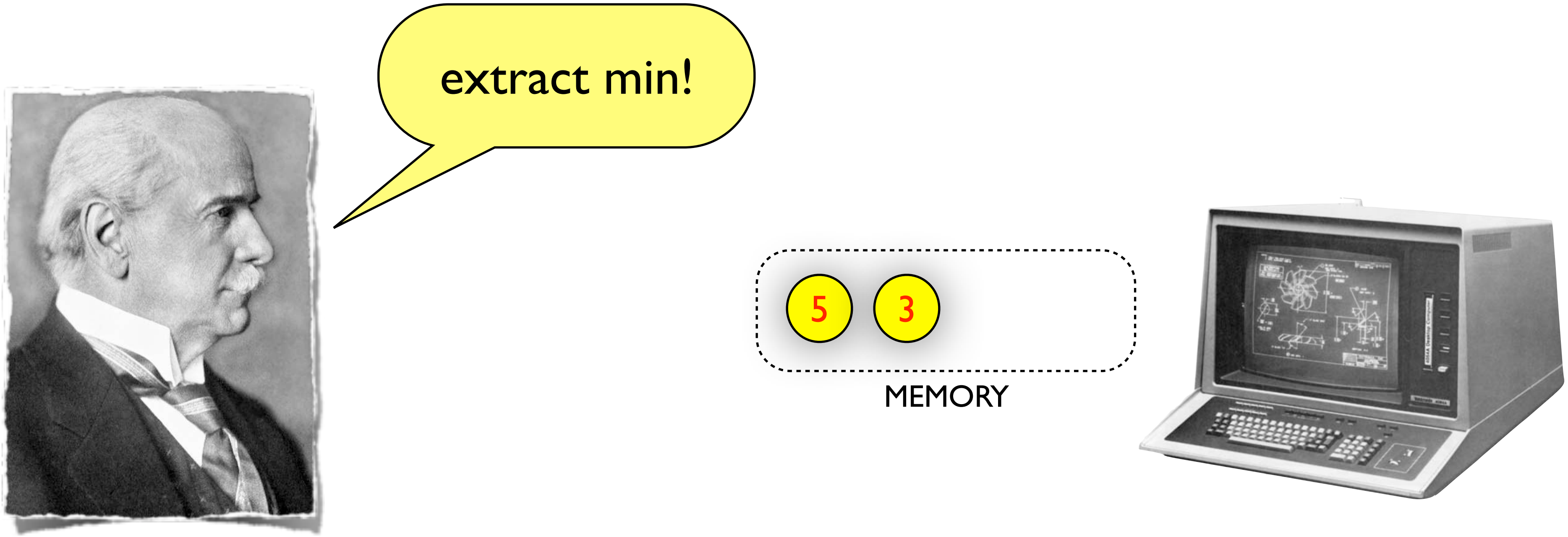
  ins(5)





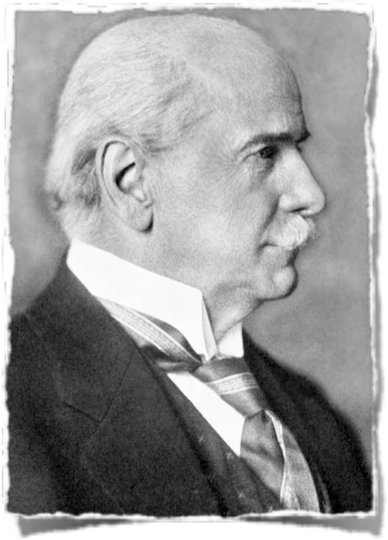
- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

    ins(5) ins(3)



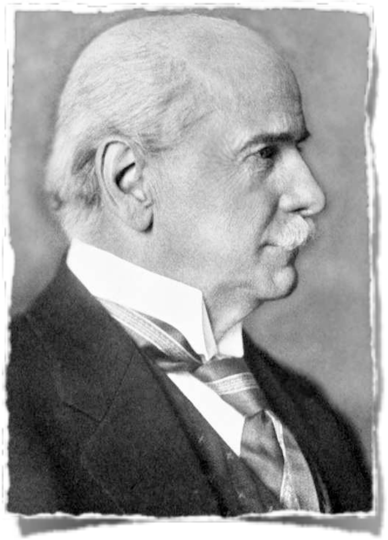
- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

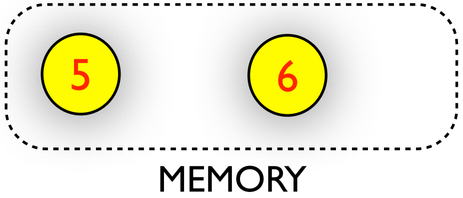
  ins(5) ins(3)



- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

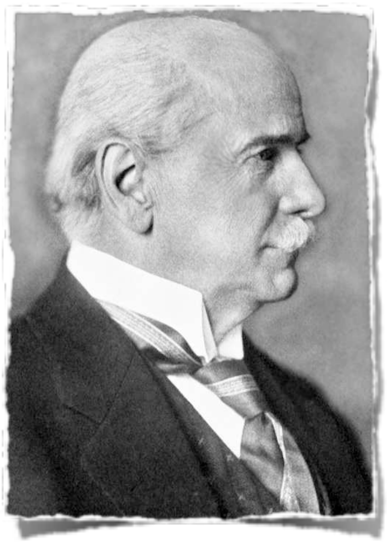
  ins(5) ins(3) ext(3)

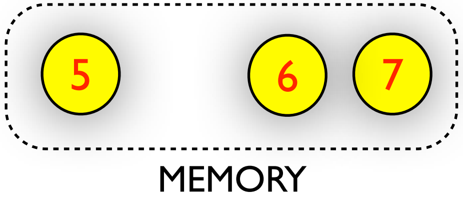




- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

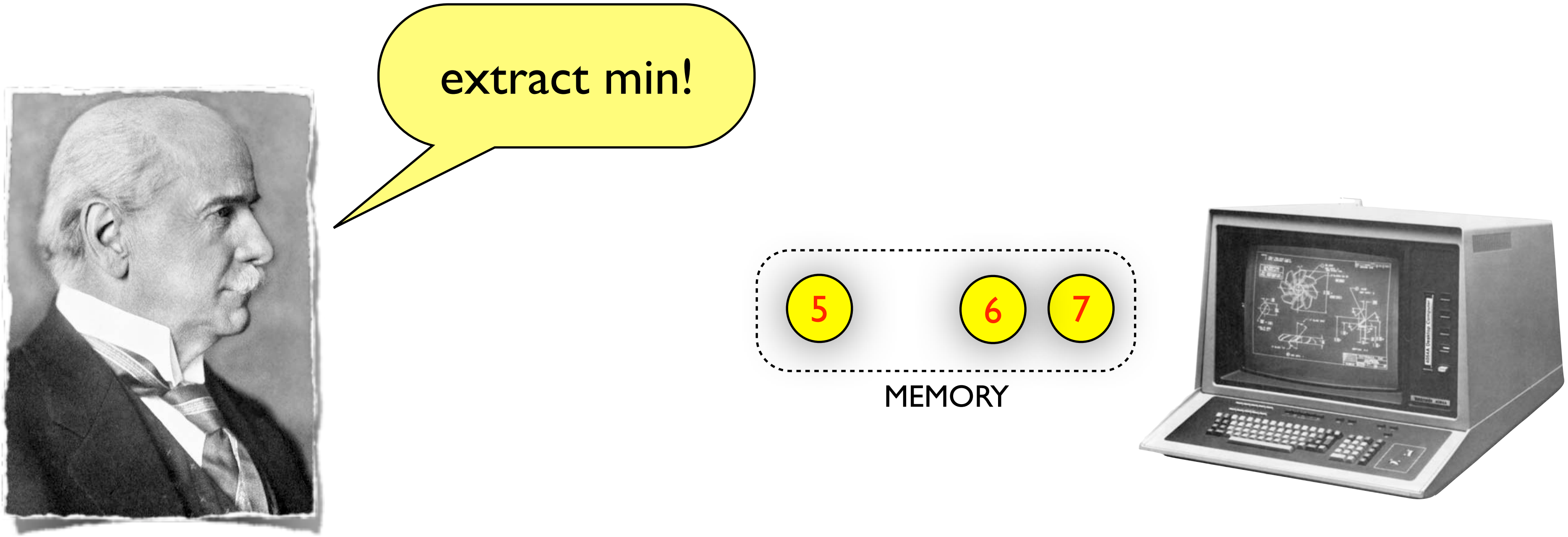
  ins(5) ins(3) ext(3) ins(6)





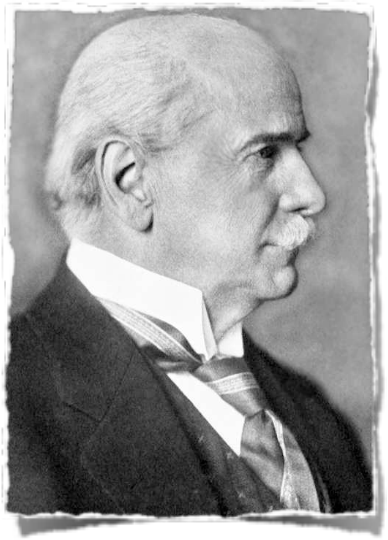
- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

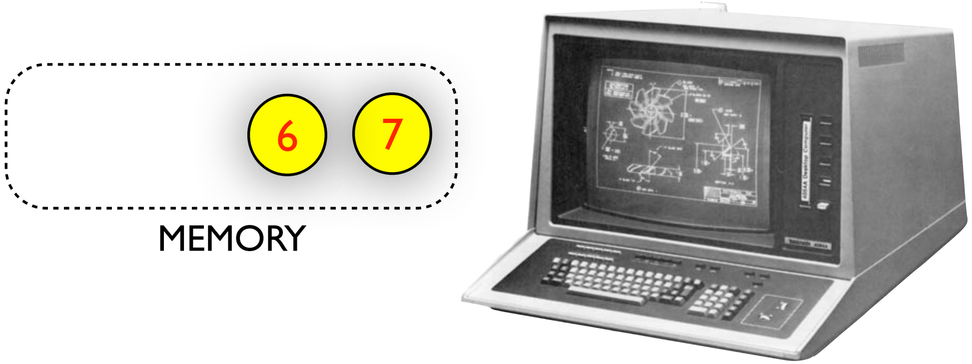
  ins(5) ins(3) ext(3) ins(6) ins(7)



- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

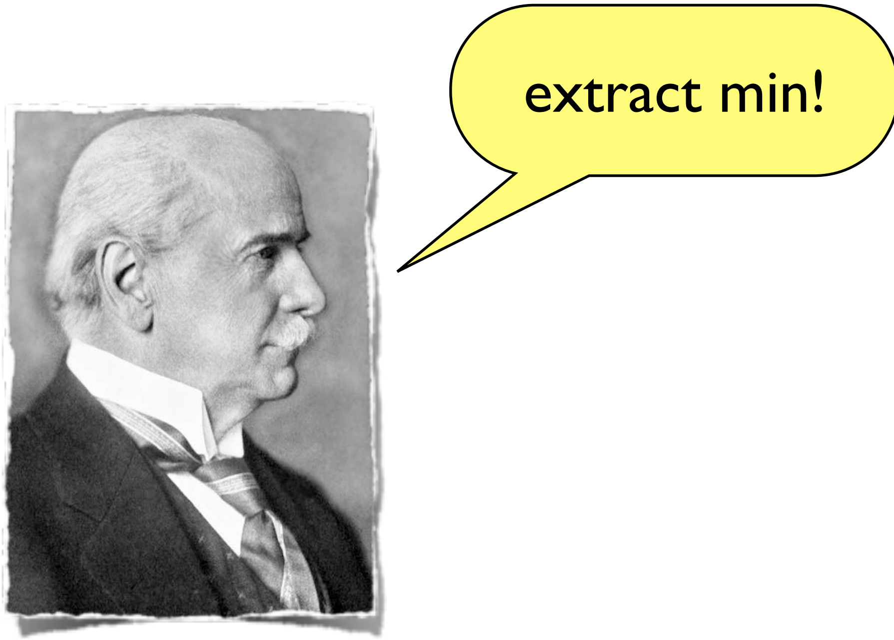
  ins(5) ins(3) ext(3) ins(6) ins(7)

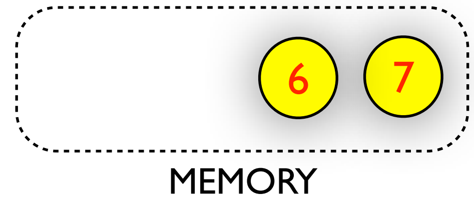




- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

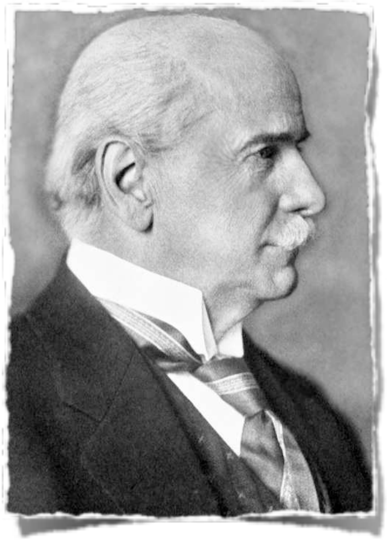
  ins(5) ins(3) ext(3) ins(6) ins(7) ext(5)





- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

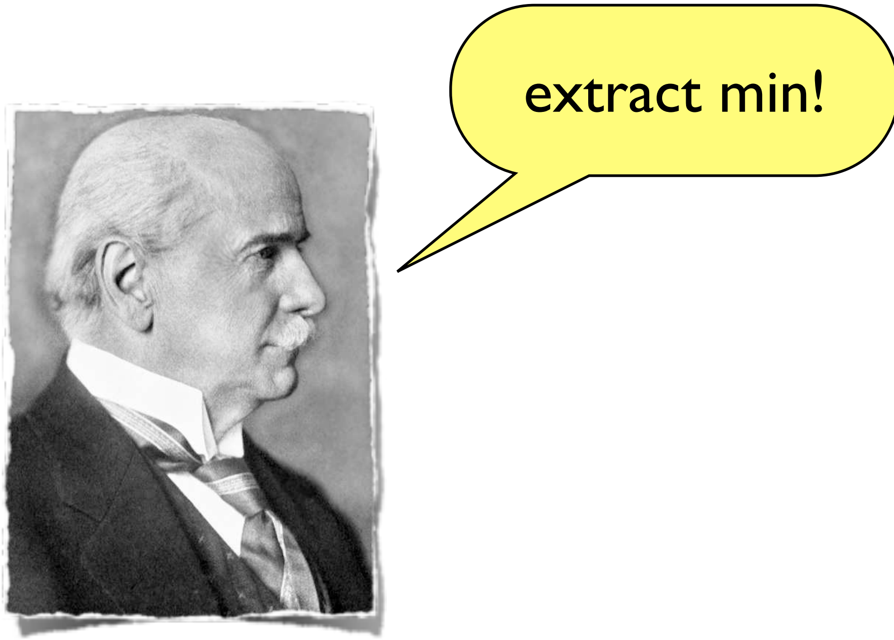
  ins(5) ins(3) ext(3) ins(6) ins(7) ext(5)



MEMORY

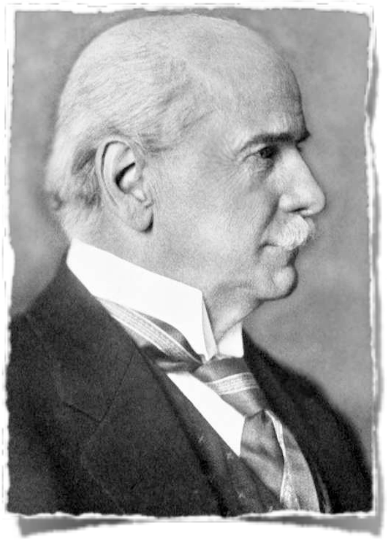
- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

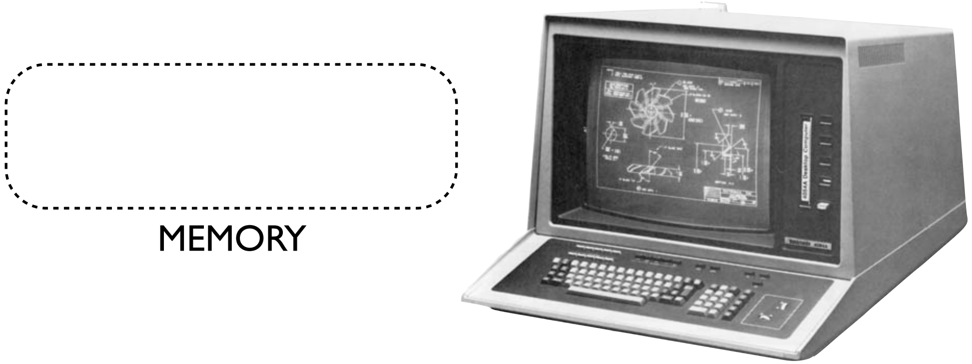
  ins(5) ins(3) ext(3) ins(6) ins(7) ext(5) ext(6)



- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

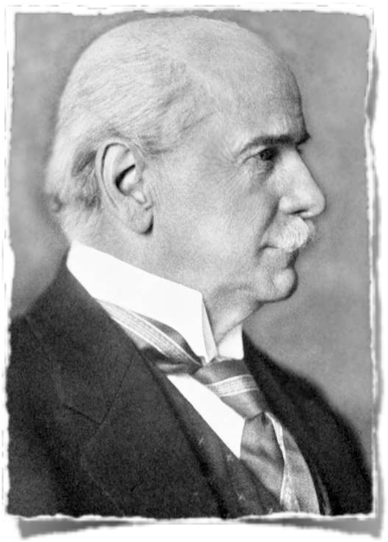
  ins(5) ins(3) ext(3) ins(6) ins(7) ext(5) ext(6)

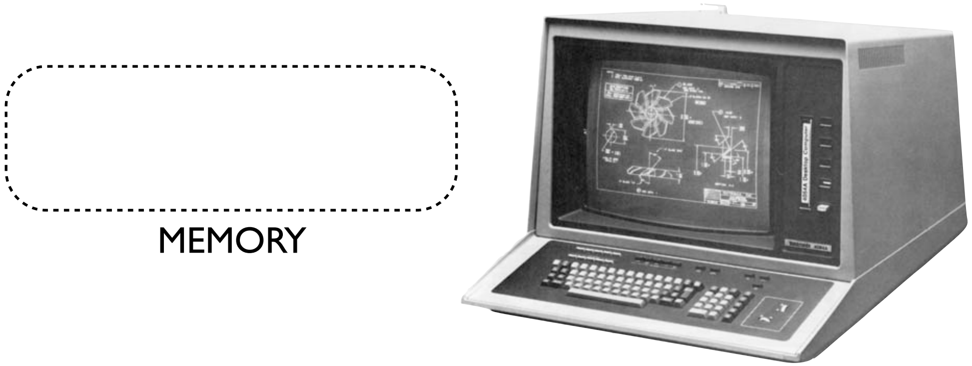




- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

  ins(5) ins(3) ext(3) ins(6) ins(7) ext(5) ext(6) ext(7)

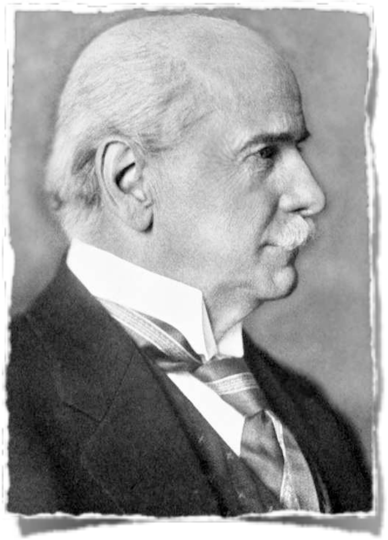




- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

  ins(5) ins(3) ext(3) ins(6) ins(7) ext(5) ext(6) ext(7)

? *Challenge:* Without remembering the entire interaction stream, can you verify priority queue performed correctly?



MEMORY

- At each step, user inserts a value into the memory or asks that the *smallest* value is extracted:

  ins(5) ins(3) ext(3) ins(6) ins(7) ext(5) ext(6) ext(7)

? *Challenge:* Without remembering the entire interaction stream, can you verify priority queue performed correctly?

- *Motivation:* Want to use cheap commodity hardware.

PQ Verification

PQ Verification

- *Thm:* Exists a $O(\sqrt{N} \log N)$ space stream algorithm with $O(\log N)$ amortized update time for verifying transcript.

PQ Verification

- *Thm:* Exists a O($\sqrt{N}$ log N) space stream algorithm with O(log N) amortized update time for verifying transcript.

"Can verify terabytes of transcript with only megabytes!"

PQ Verification

- *Thm:* Exists a $O(\sqrt{N} \log N)$ space stream algorithm with $O(\log N)$ amortized update time for verifying transcript.

  "Can verify terabytes of transcript with only megabytes!"

- *Prelim:* Easy to check that set of values inserted equals set of values extracted using fingerprinting:

$$f_S(x) = \prod_{a \in S} (x - a) \mod p$$

PQ Verification

- *Thm:* Exists a $O(\sqrt{N} \log N)$ space stream algorithm with $O(\log N)$ amortized update time for verifying transcript.

"Can verify terabytes of transcript with only megabytes!"

- *Prelim:* Easy to check that set of values inserted equals set of values extracted using fingerprinting:

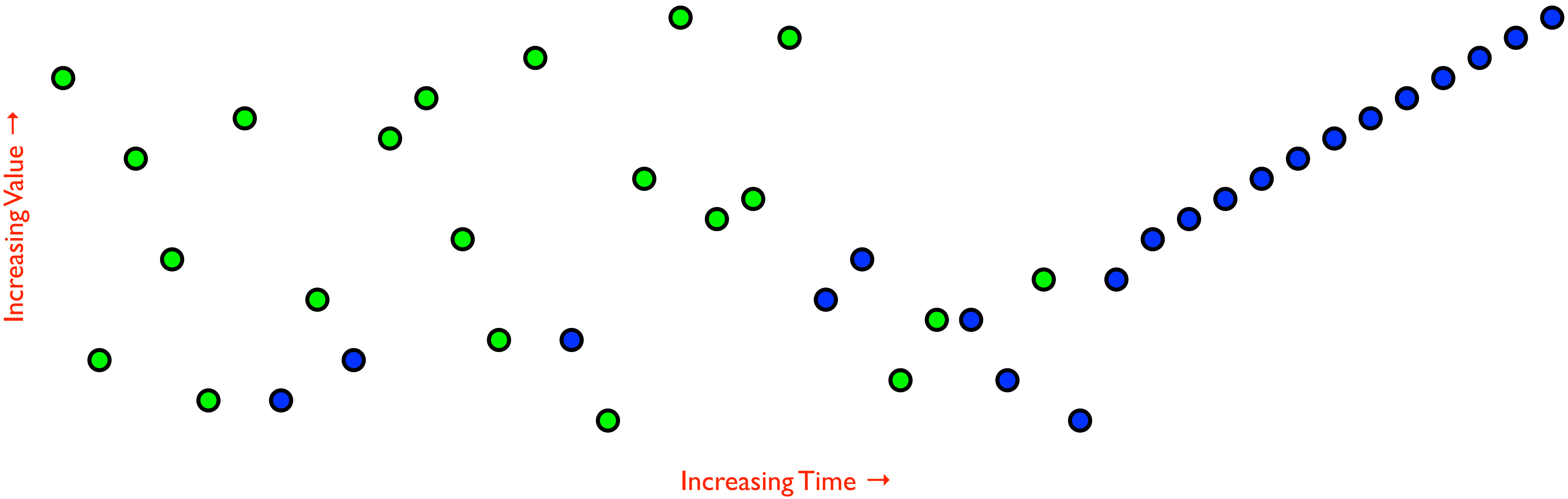
$$f_S(x) = \prod_{a \in S} (x - a) \mod p$$

! *For this talk:* Assume inserted elements are distinct and that inserts come before their corresponding extract. I.e., we're trying to identify the following *bad pattern:*

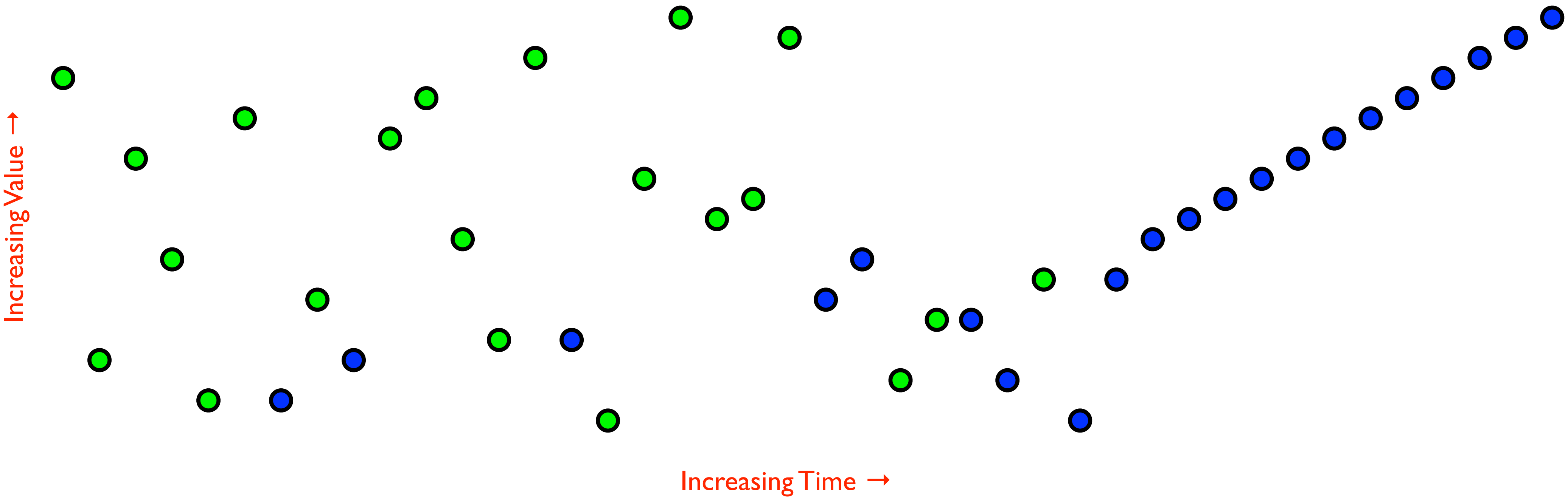
ins(u) .... ext(v) ... ext(u) for some $u < v$

Epochs and Local Bad Patterns…

Epochs and Local Bad Patterns...

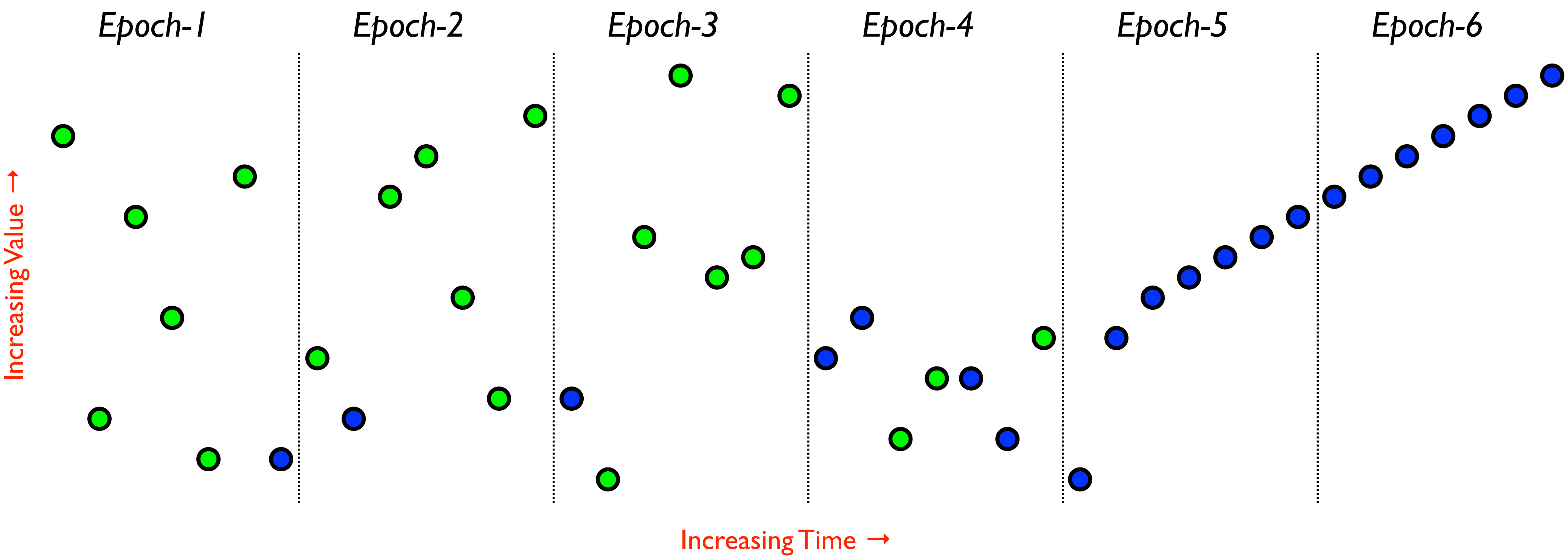


Epochs and Local Bad Patterns...



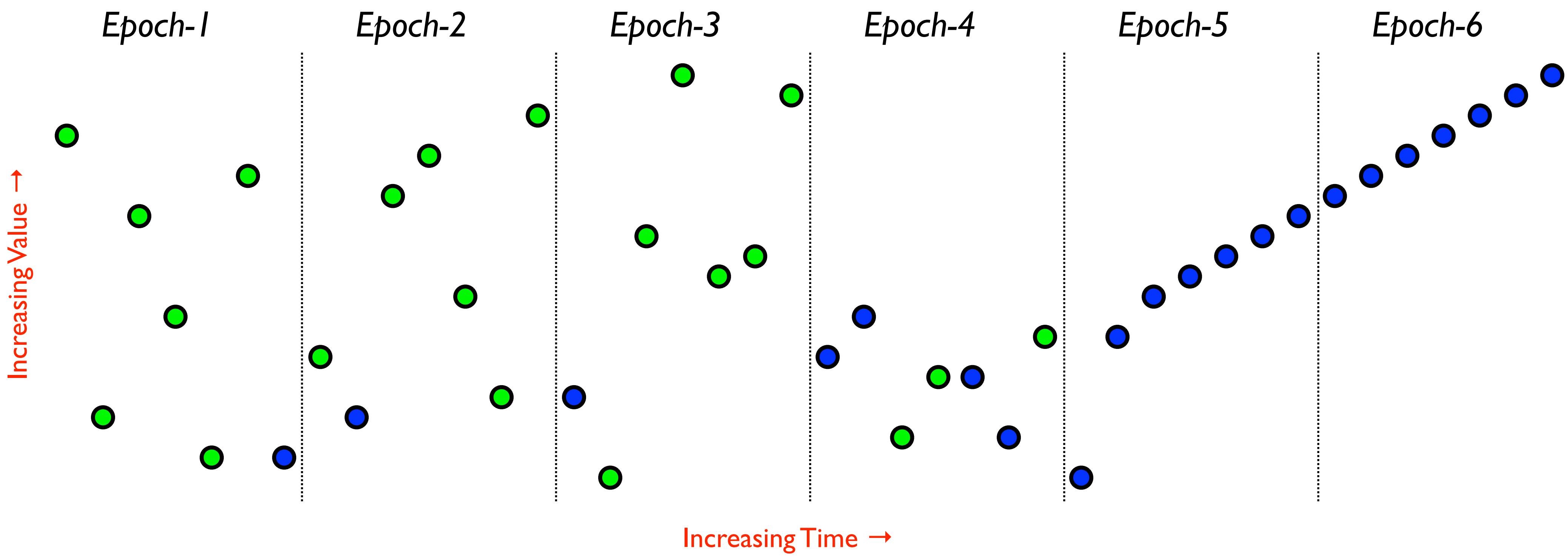
- Split length N sequence into $\sqrt{N}$ epochs of length $\sqrt{N}$

Epochs and Local Bad Patterns...



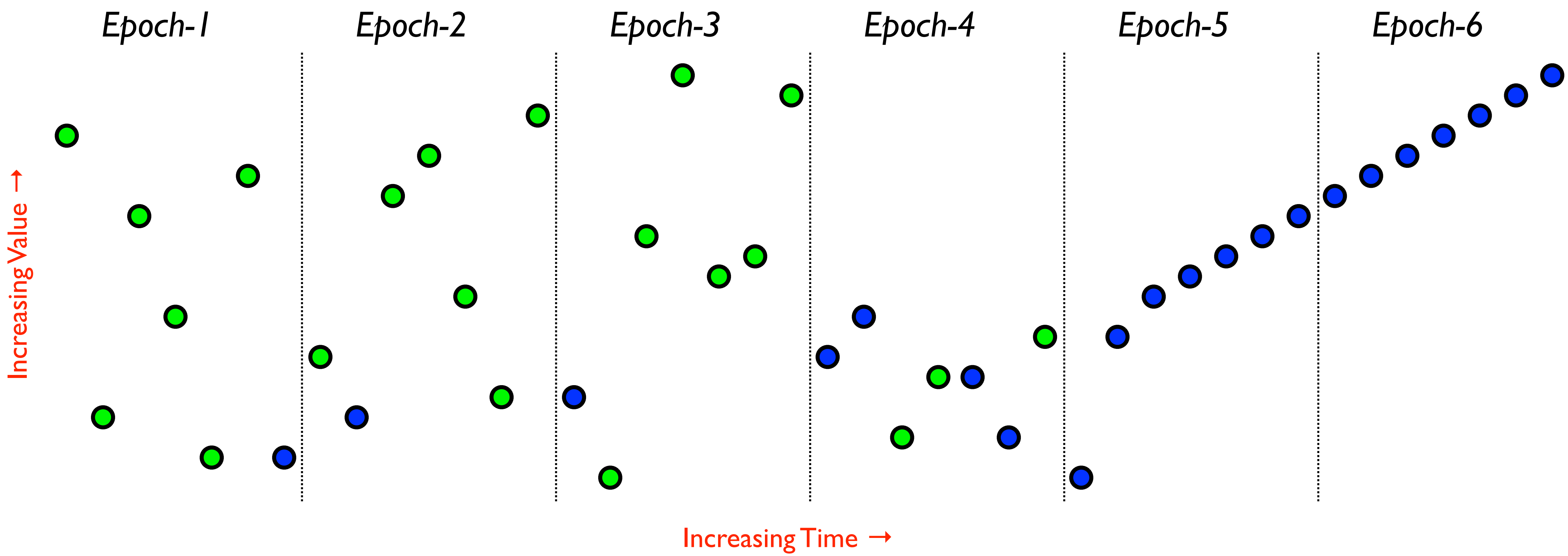
- Split length N sequence into $\sqrt{N}$ epochs of length $\sqrt{N}$

Epochs and Local Bad Patterns...



- Split length N sequence into $\sqrt{N}$ epochs of length $\sqrt{N}$
- *Defn:* Bad pattern ins(u) ... ext(v) ... ext(u) is *local* if ins(u) and ext(v) occur in same epoch and *long-range* otherwise.

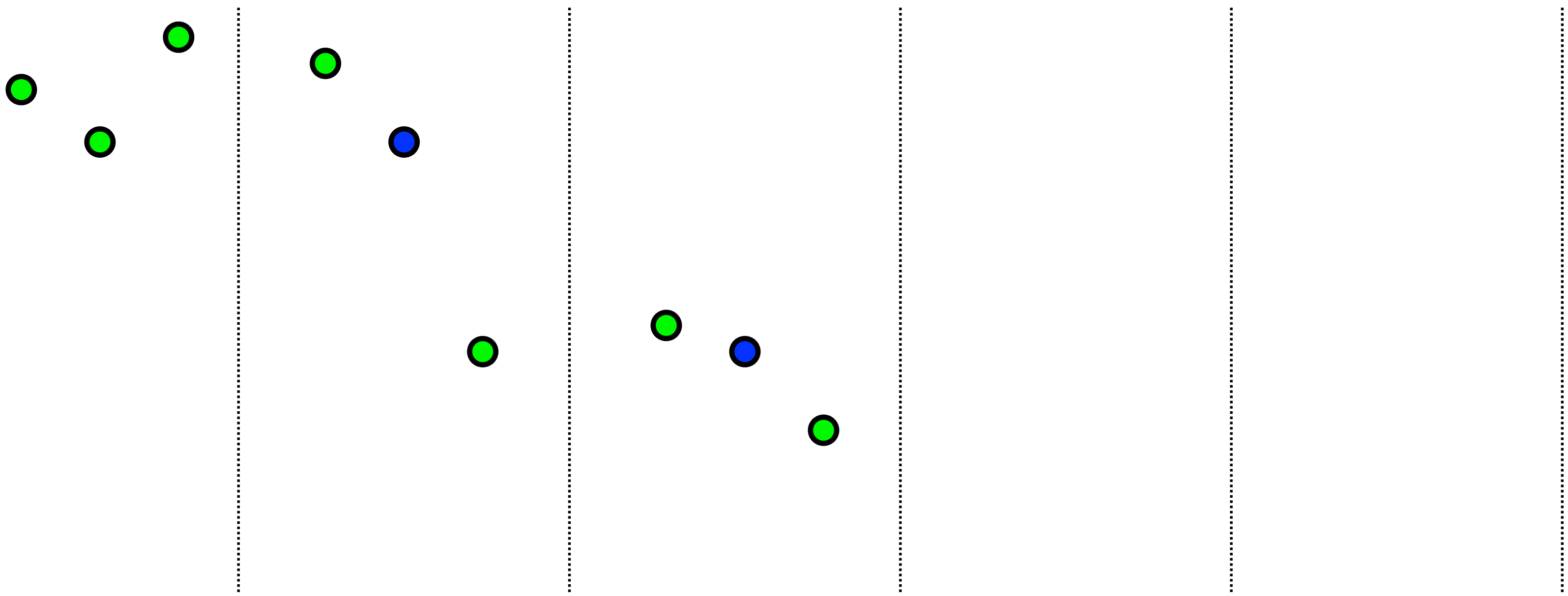
Epochs and Local Bad Patterns...



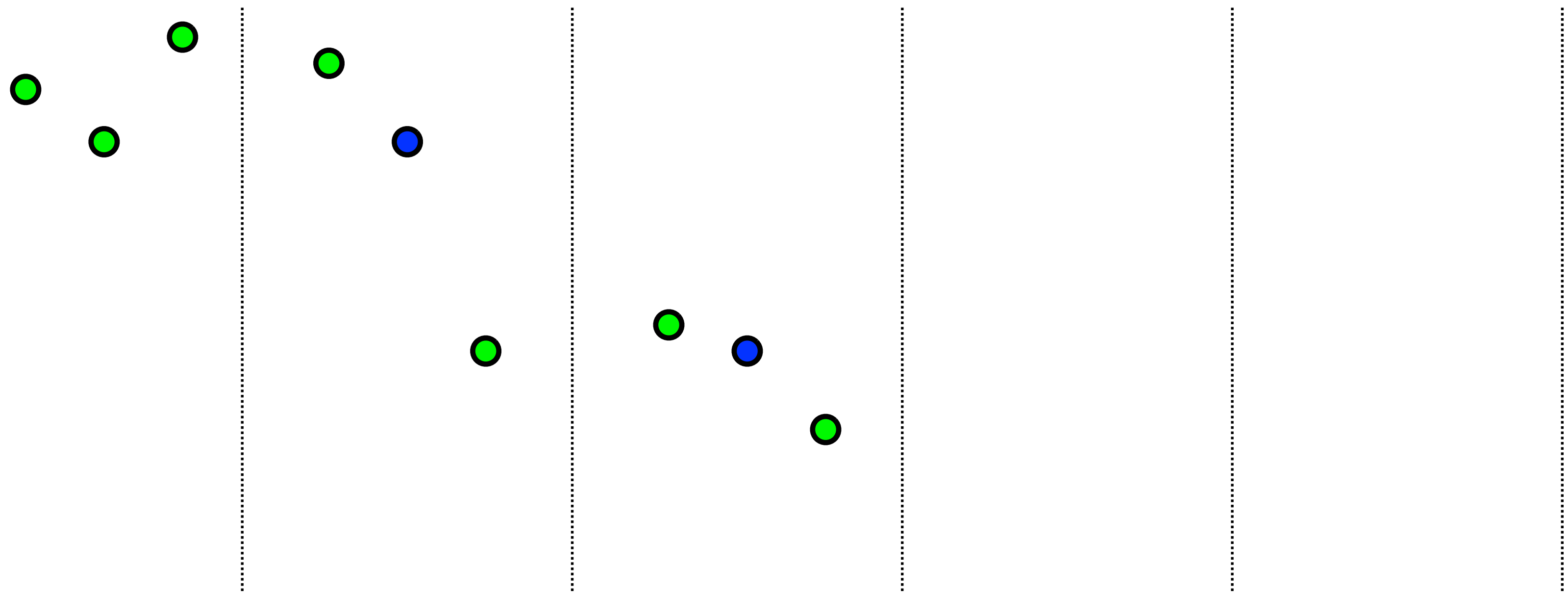
- Split length N sequence into $\sqrt{N}$ epochs of length $\sqrt{N}$
- *Defn:* Bad pattern ins(u) ... ext(v) ... ext(u) is *local* if ins(u) and ext(v) occur in same epoch and *long-range* otherwise.
- Using $O(\sqrt{N})$ space, we can buffer each epoch and check for local bad patterns.

Catching Long-Range Bad Patterns… 1/2

Catching Long-Range Bad Patterns… 1/2

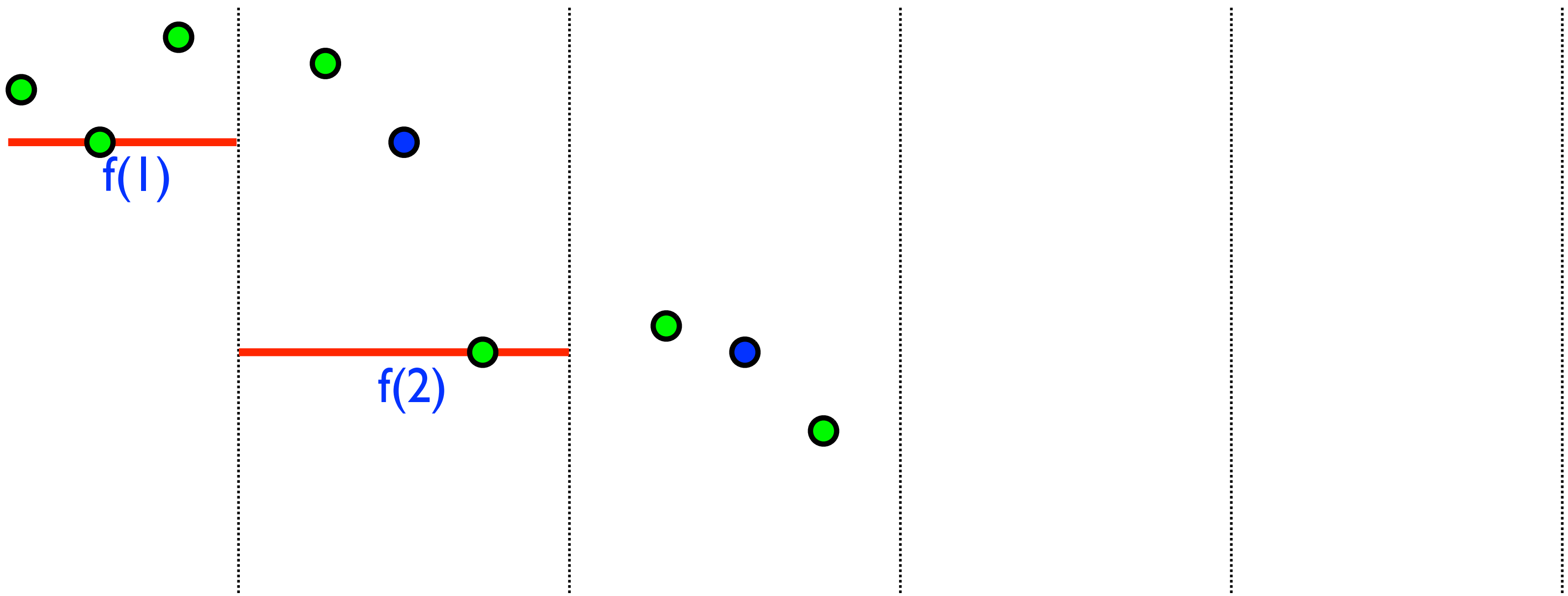


Catching Long-Range Bad Patterns... 1/2



- Maintain the max value extracted between end of i-th epoch and *current time*. Call it f(i).

Catching Long-Range Bad Patterns... 1/2

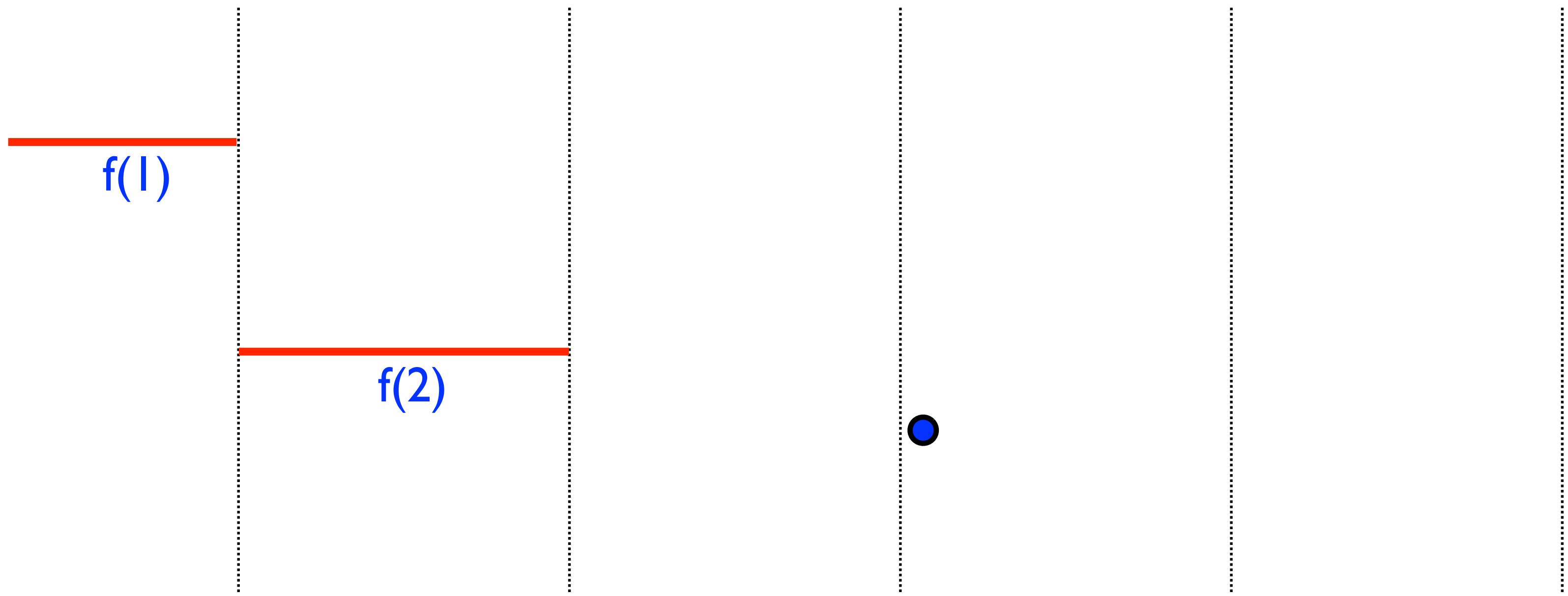


- Maintain the max value extracted between end of i-th epoch and *current time*. Call it f(i).

Catching Long-Range Bad Patterns... 1/2

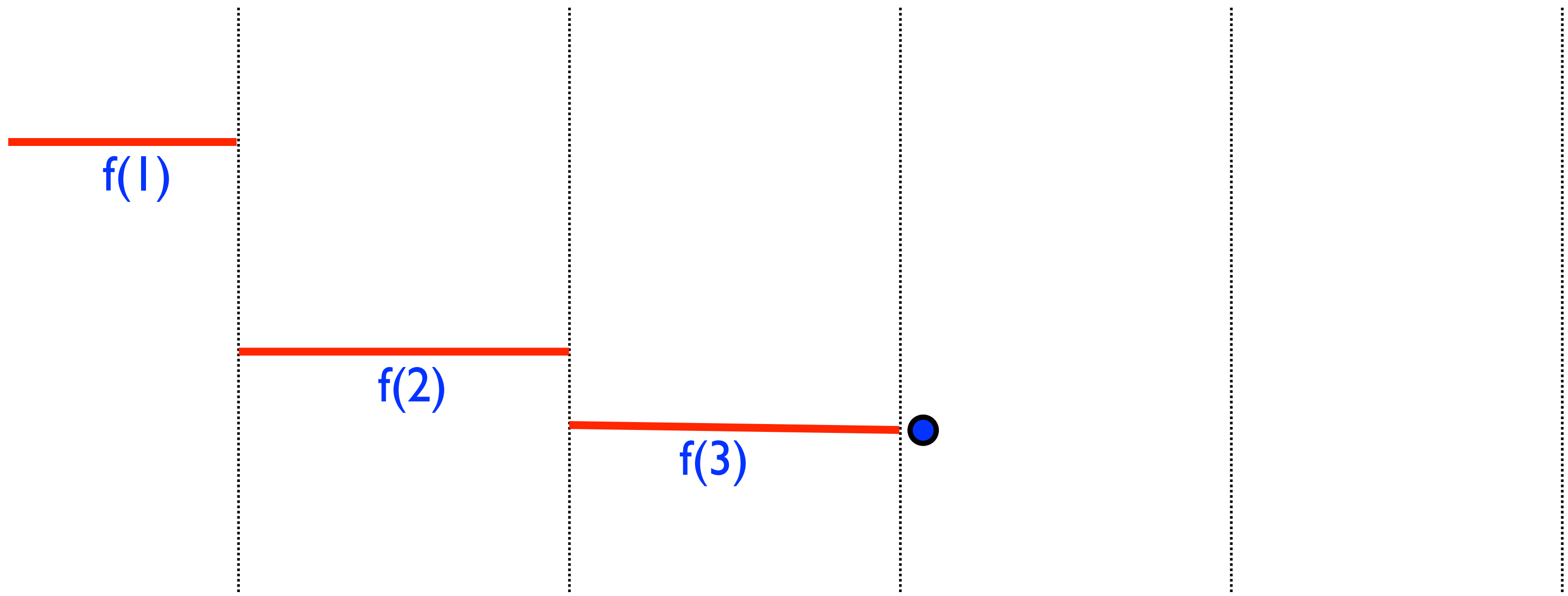
- Maintain the max value extracted between end of i-th epoch and *current time*. Call it f(i).

Catching Long-Range Bad Patterns… 1/2



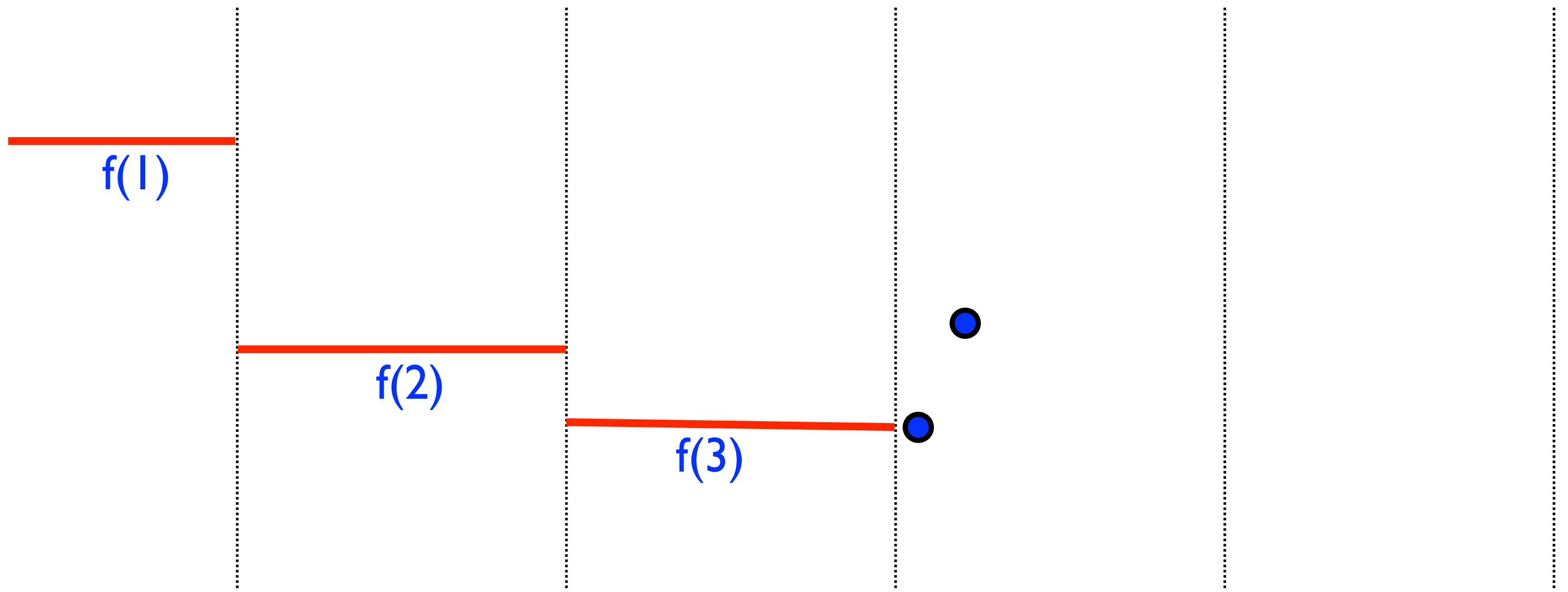
- Maintain the max value extracted between end of i-th epoch and *current time*. Call it f(i).

Catching Long-Range Bad Patterns... 1/2



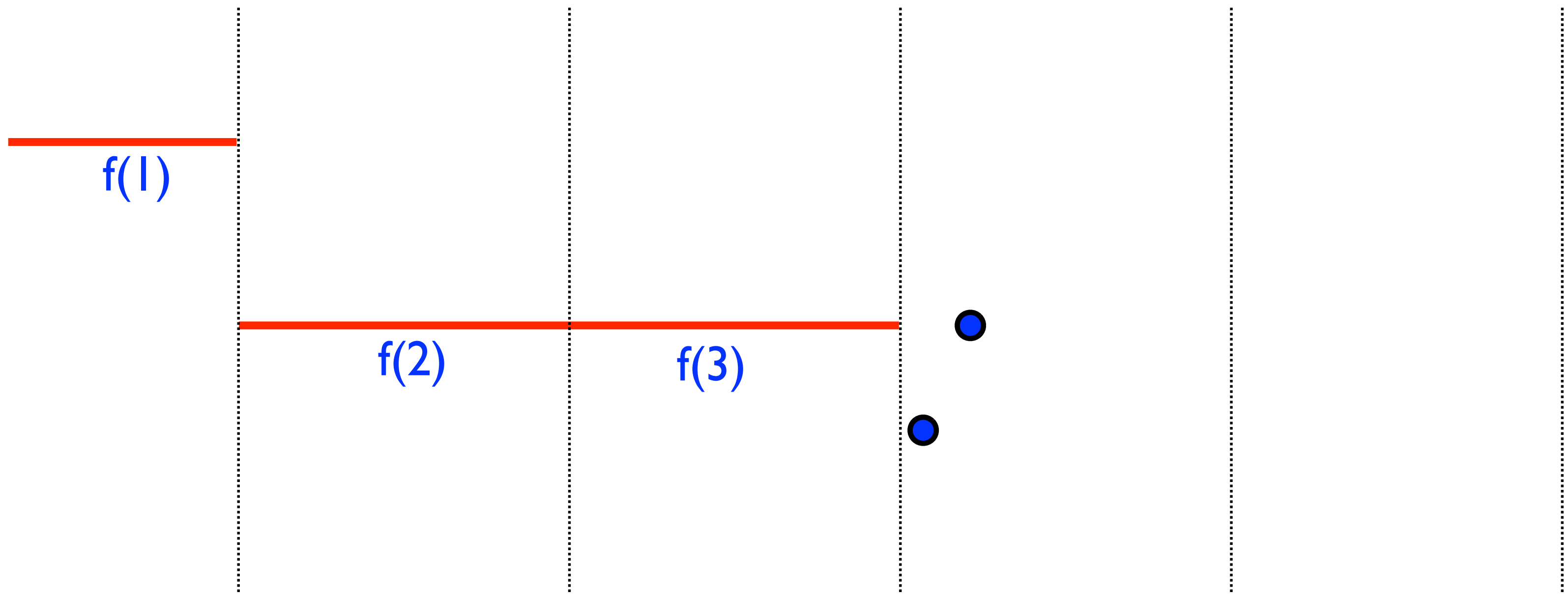
- Maintain the max value extracted between end of i-th epoch and *current time*. Call it f(i).

Catching Long-Range Bad Patterns... 1/2



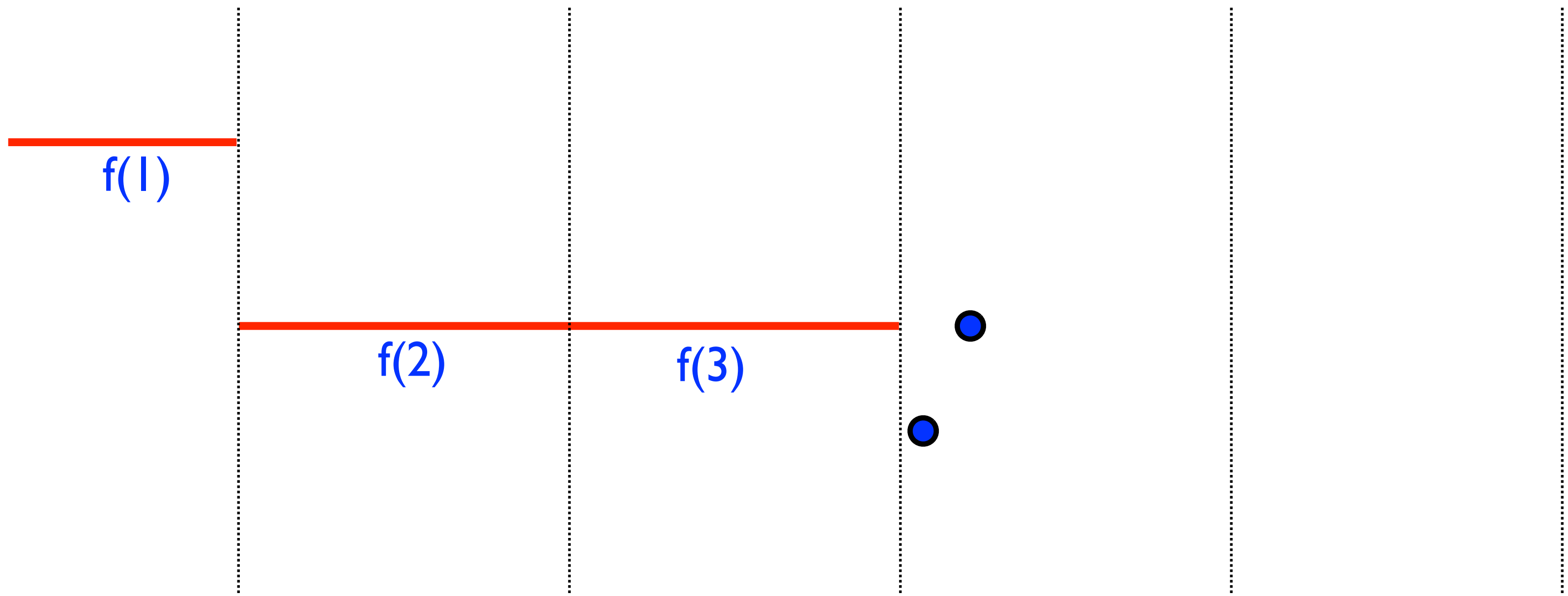
- Maintain the max value extracted between end of i-th epoch and *current time*. Call it f(i).

Catching Long-Range Bad Patterns... 1/2



- Maintain the max value extracted between end of i-th epoch and *current time*. Call it f(i).

Catching Long-Range Bad Patterns... 1/2



- Maintain the max value extracted between end of i-th epoch and *current time*. Call it f(i).
- *Defn:* Each ins(u) or ext(u) is *adopted* by earliest epoch k with f(k)≤u.

Catching Long-Range Bad Patterns... 2/2

Catching Long-Range Bad Patterns... 2/2

- *Lemma:* If ins(u) ... ext(v) ... ext(u) is a long-range bad pattern then ins(u) and ext(u) are adopted by different epochs.

Catching Long-Range Bad Patterns… 2/2

- *Lemma:* If ins(u) … ext(v) … ext(u) is a long-range bad pattern then ins(u) and ext(u) are adopted by different epochs.
- *Proof:*
  1. Let ins(u) be adopted by k-th epoch.

Catching Long-Range Bad Patterns... 2/2

- *Lemma:* If ins(u) ... ext(v) ... ext(u) is a long-range bad pattern then ins(u) and ext(u) are adopted by different epochs.
- *Proof:*
  1. Let ins(u) be adopted by k-th epoch.
  2. After v is extracted $f(k) \geq v > u$.

Catching Long-Range Bad Patterns... 2/2

- *Lemma:* If ins(u) ... ext(v) ... ext(u) is a long-range bad pattern then ins(u) and ext(u) are adopted by different epochs.
- *Proof:*
    i. Let ins(u) be adopted by k-th epoch.
    ii. After v is extracted $f(k) \geq v > u$.
    iii. ext(u) can no longer be adopted by k-th epoch.

Catching Long-Range Bad Patterns... 2/2

- *Lemma:* If ins(u) ... ext(v) ... ext(u) is a long-range bad pattern then ins(u) and ext(u) are adopted by different epochs.
- *Proof:*
  i. Let ins(u) be adopted by k-th epoch.
  ii. After v is extracted $f(k) \geq v > u$.
  iii. ext(u) can no longer be adopted by k-th epoch.
- *Lemma:* If there are no bad patterns, every ins(u) and ext(u) pair get adopted by the same epoch.

Catching Long-Range Bad Patterns... 2/2

- *Lemma:* If ins(u) ... ext(v) ... ext(u) is a long-range bad pattern then ins(u) and ext(u) are adopted by different epochs.
- *Proof:*
  i. Let ins(u) be adopted by k-th epoch.
  ii. After v is extracted $f(k) \geq v > u$.
  iii. ext(u) can no longer be adopted by k-th epoch.
- *Lemma:* If there are no bad patterns, every ins(u) and ext(u) pair get adopted by the same epoch.
- *Algorithm:* Using fingerprints to check: for each epoch k

$$\{u : \text{ins}(u) \text{ adopted by } k\} = \{u : \text{ext}(u) \text{ adopted by } k\}.$$

Conclusions

Conclusions

- *Thm:* There exists a $O(\sqrt{N} \log N)$ space algorithm with $O(\log N)$ amortized update time for recognizing PQ.

  "Can verify terabytes of transcript with only megabytes"

Conclusions

- *Thm:* There exists a $O(\sqrt{N} \log N)$ space algorithm with $O(\log N)$ amortized update time for recognizing PQ.

  "Can verify terabytes of transcript with only megabytes"

- *Extensions:* Sub-linear space streaming recognition of other data structures like stacks, double-ended queues...