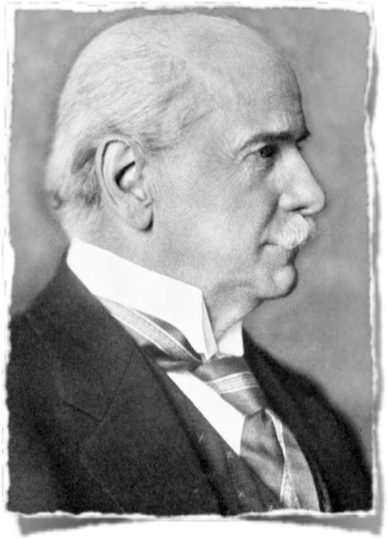
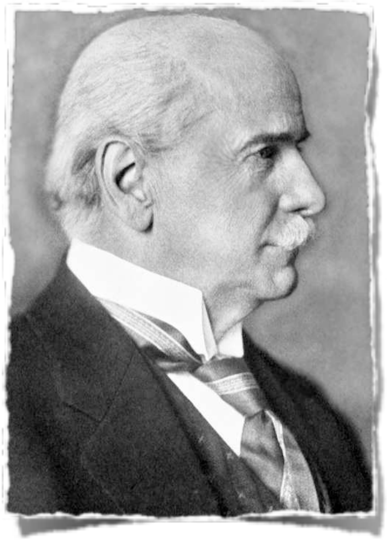
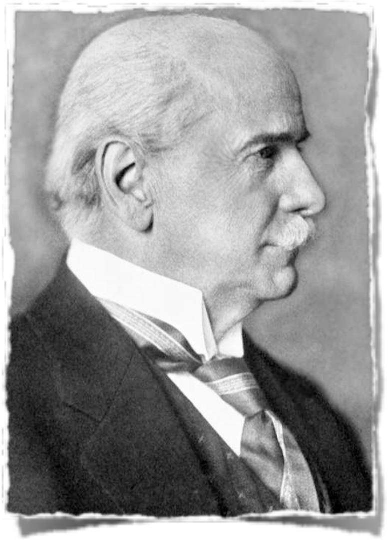
Data Streams, Dyck Languages, and *Detecting Dubious Data Structures*

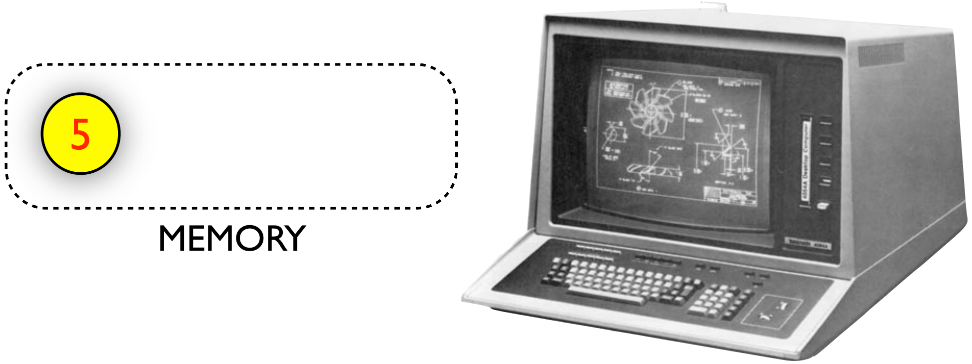
Amit Chakrabarti *Dartmouth College*
Graham Cormode *AT&T Research Labs*
Ranganath Kondapally *Dartmouth College*
Andrew McGregor *University of Massachusetts, Amherst*





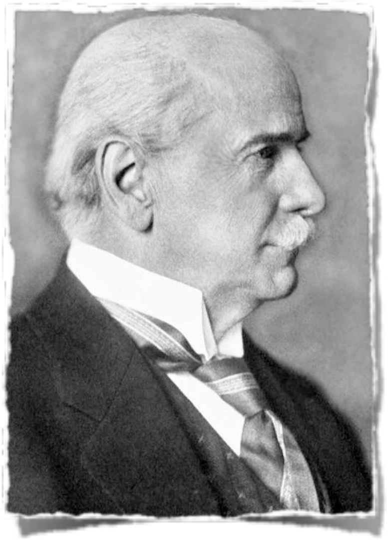
- At each step, user inserts a value into the memory or asks that the smallest currently stored value is extracted

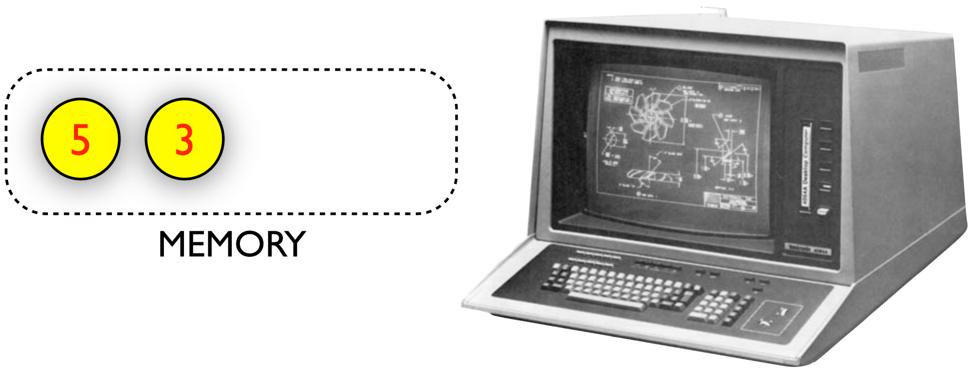




- At each step, user inserts a value into the memory or asks that the smallest currently stored value is extracted

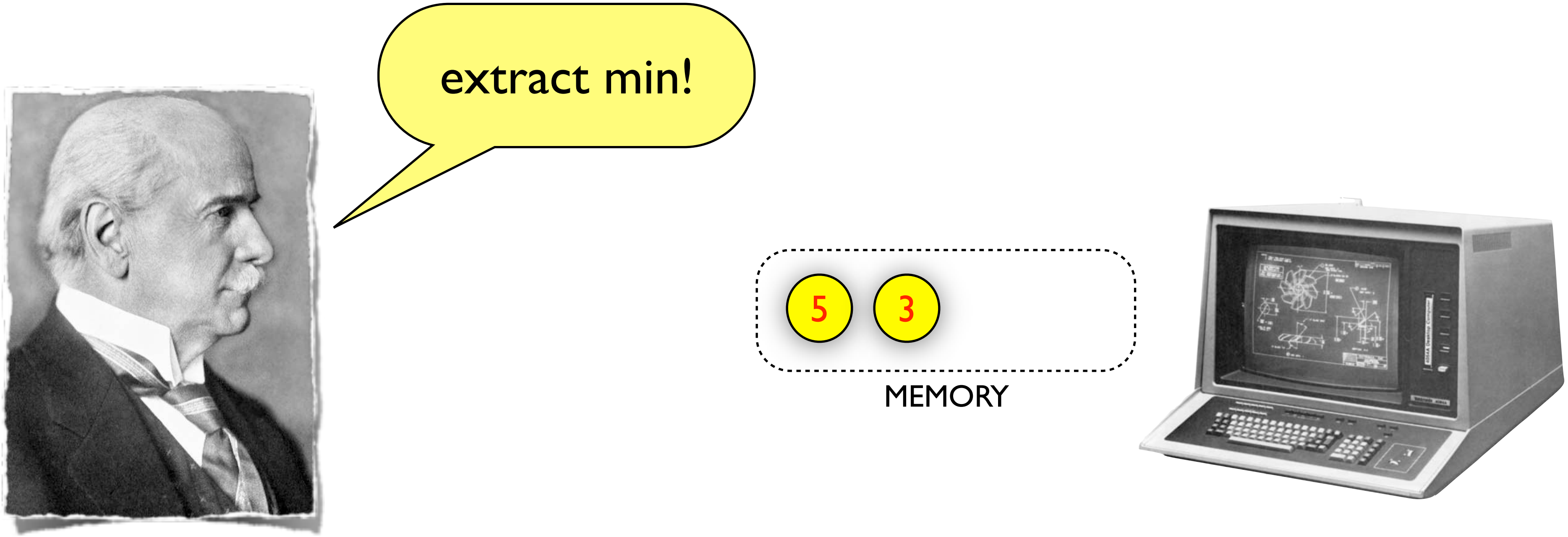
  ins(5)





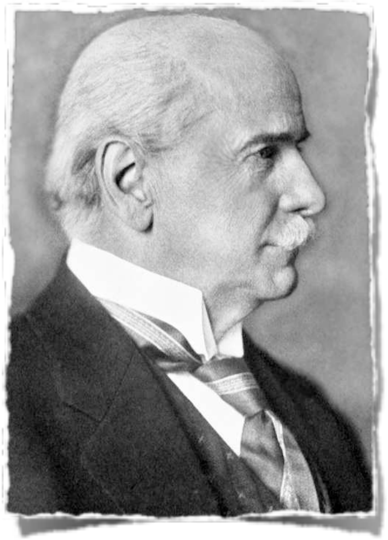
- At each step, user inserts a value into the memory or asks that the smallest currently stored value is extracted

  ins(5) ins(3)



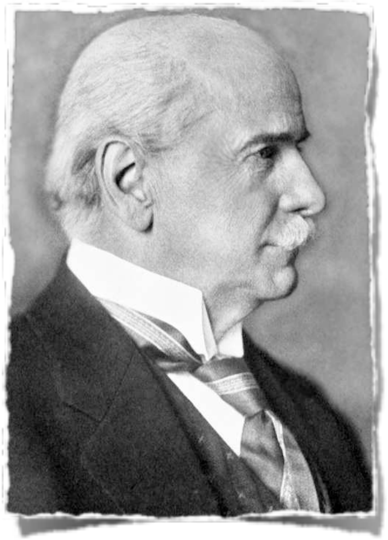
- At each step, user inserts a value into the memory or asks that the smallest currently stored value is extracted

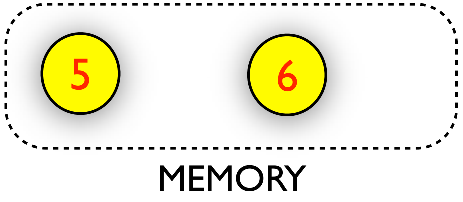
    ins(5) ins(3)



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

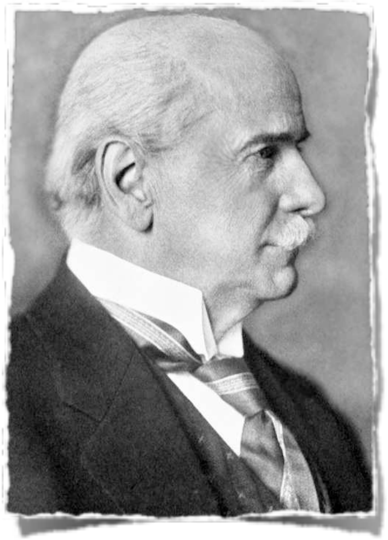
  ins(5) ins(3) ext(3)

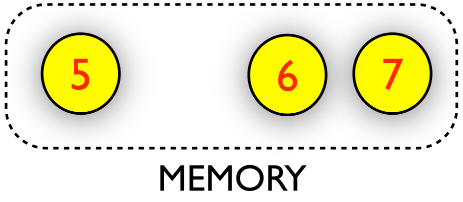




- At each step, user inserts a value into the memory or asks that the smallest currently stored value is extracted

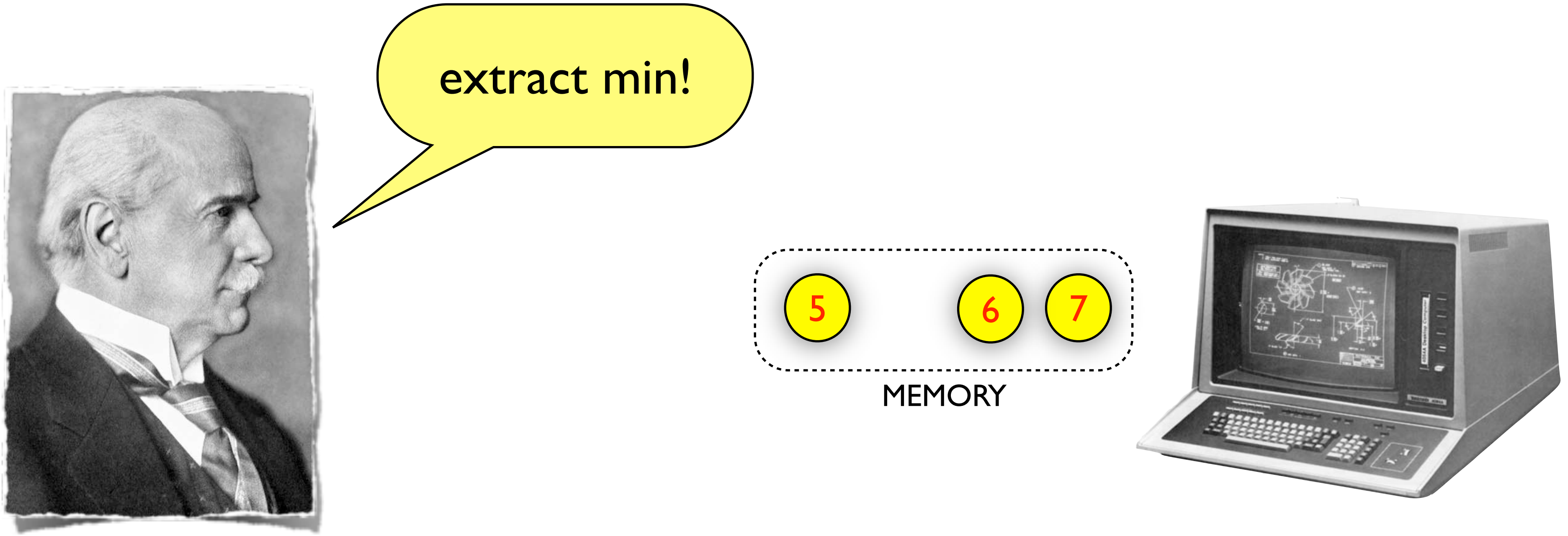
  ins(5) ins(3) ext(3) ins(6)





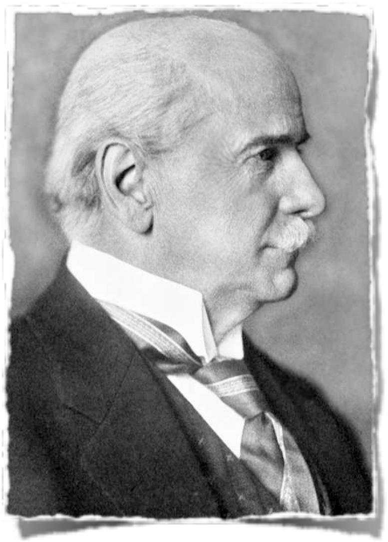
- At each step, user inserts a value into the memory or asks that the smallest currently stored value is extracted

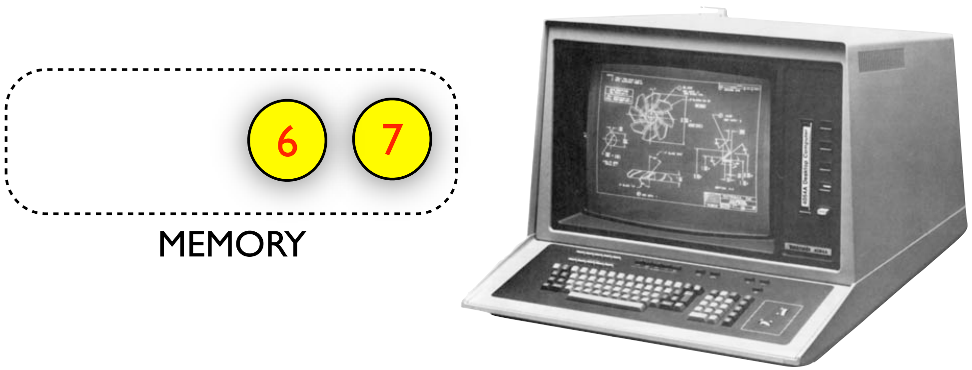
ins(5) ins(3) ext(3) ins(6) ins(7)



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

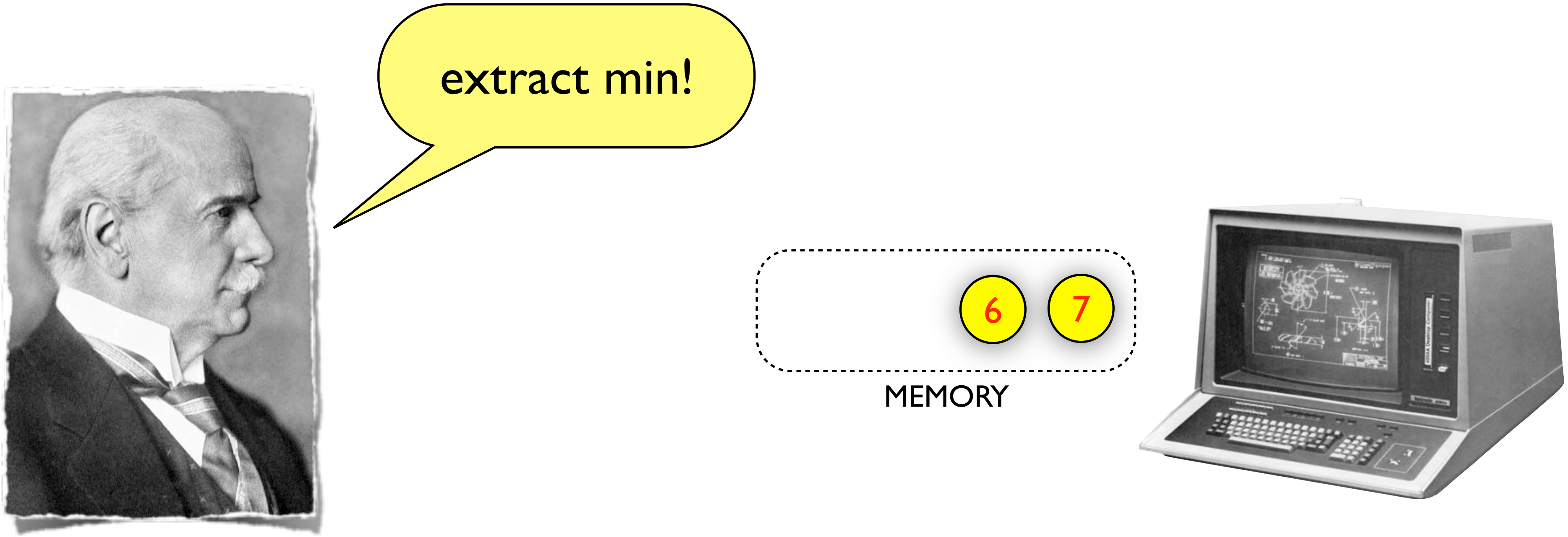
  ins(5) ins(3) ext(3) ins(6) ins(7)





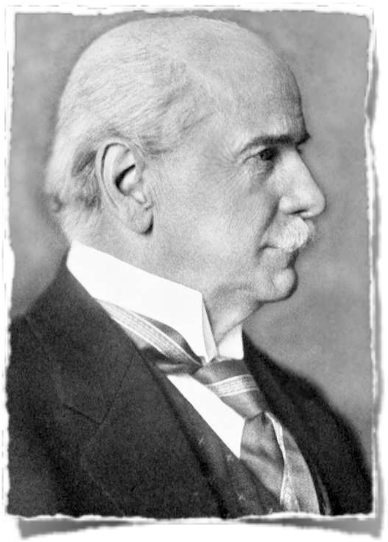
- At each step, user inserts a value into the memory or asks that the smallest currently stored value is extracted

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



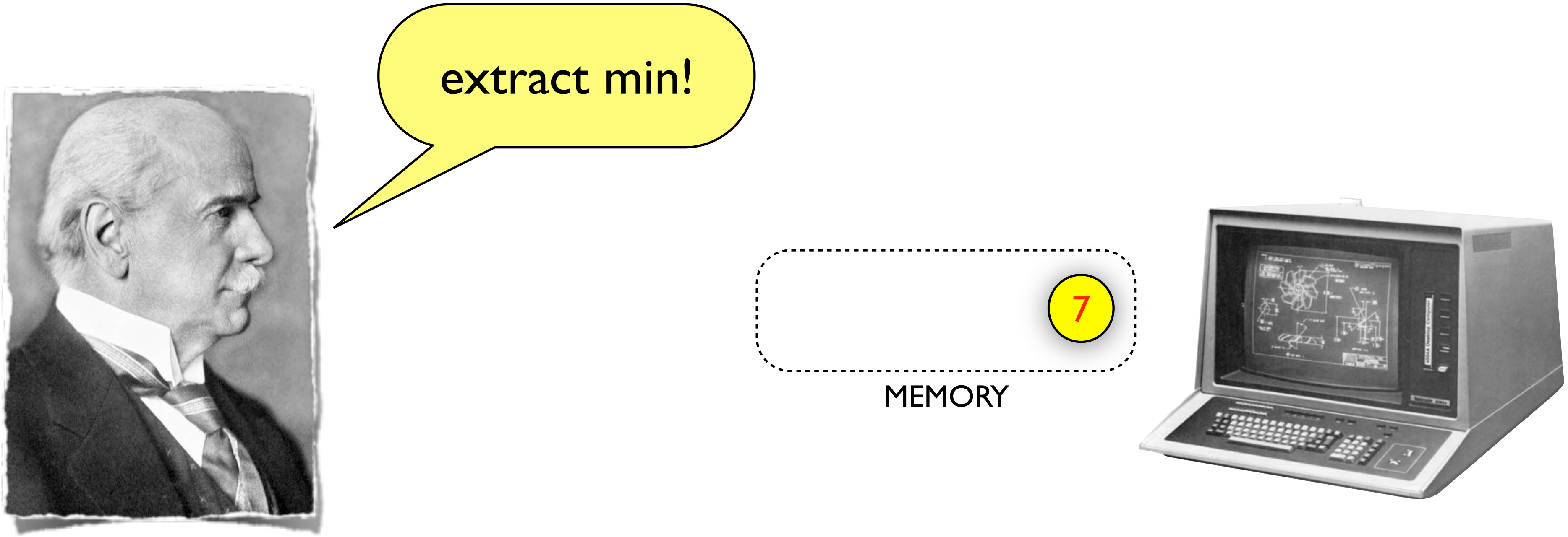
- At each step, user inserts a value into the memory or asks that the smallest currently stored value is extracted

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



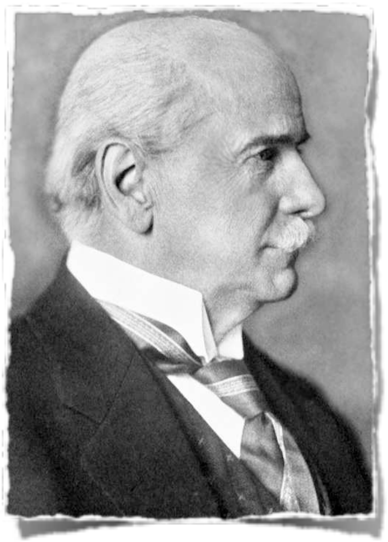
- At each step, user inserts a value into the memory or asks that the smallest currently stored value is extracted

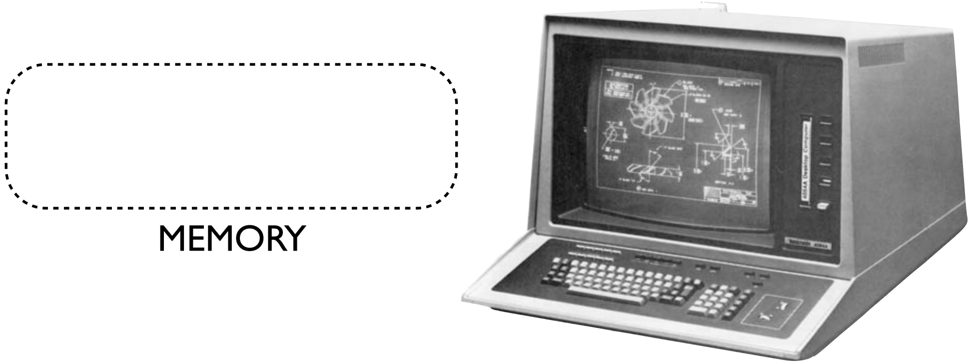
  ins(5) ins(3) ext(3) ins(6) ins(7) ext(5) ext(6)



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

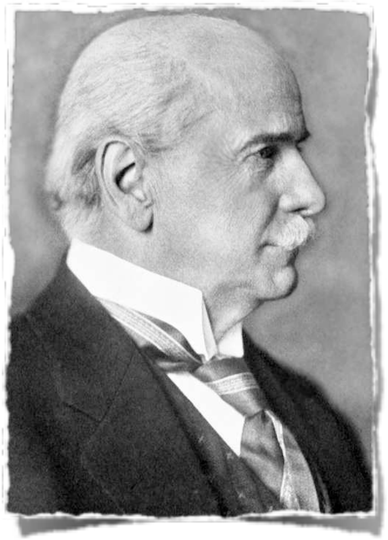
  ins(5) ins(3) ext(3) ins(6) ins(7) ext(5) ext(6)





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

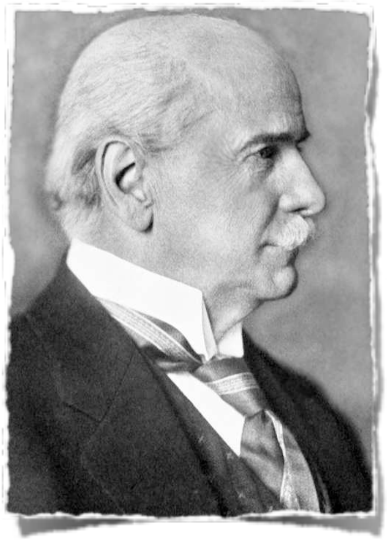
  ins(5) ins(3) ext(3) ins(6) ins(7) ext(5) ext(6) ext(7)



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

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

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



MEMORY

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

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

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

- *Motivation:* Memory checking useful when using cheap commodity hardware. [Blum, Evans, Gemmell, Kannan, Naor '94]

PQ Language Problem

PQ Language Problem

- Let PQ be set of legitimate transcripts of a priority queue that starts and ends empty.

PQ Language Problem

- Let PQ be set of legitimate transcripts of a priority queue that starts and ends empty.

  ins(5), ins(3), ext(3), ins(7), ext(5), ext(7) $\in$ PQ

PQ Language Problem

- Let PQ be set of legitimate transcripts of a priority queue that starts and ends empty.

  ins(5), ins(3), ext(3), ins(7), ext(5), ext(7) $\in$ PQ

  ins(5), ext(3), ins(5), ins(7), ext(7), ext(5) $\notin$ PQ

PQ Language Problem

- Let PQ be set of legitimate transcripts of a priority queue that starts and ends empty.

  ins(5), ins(3), ext(3), ins(7), ext(5), ext(7) $\in$ PQ

  ins(5), ext(3), ins(5), ins(7), ext(7), ext(5) $\notin$ PQ

- *PQ Problem:* Given *streaming* access to length N transcript, determine if it's in PQ using o(N) space.

PQ Language Problem

- Let PQ be set of legitimate transcripts of a priority queue that starts and ends empty.

  ins(5), ins(3), ext(3), ins(7), ext(5), ext(7) $\in$ PQ

  ins(5), ext(3), ins(5), ins(7), ext(7), ext(5) $\notin$ PQ

- *PQ Problem:* Given *streaming* access to length N transcript, determine if it's in PQ using o(N) space.

- *Previous results:* If each extract is *annotated* with insert time, $\tilde{O}(\sqrt{N})$ space suffices. [Chu, Kannan, McGregor '07]

  ins(5), ins(3), ext(3,2), ins(7), ext(5,1), ext(7,4) $\in$ PQ^+

PQ Language Problem

- Let PQ be set of legitimate transcripts of a priority queue that starts and ends empty.

  ins(5), ins(3), ext(3), ins(7), ext(5), ext(7) $\in$ PQ

  ins(5), ext(3), ins(5), ins(7), ext(7), ext(5) $\notin$ PQ

- PQ *Problem:* Given *streaming* access to length N transcript, determine if it's in PQ using o(N) space.

- *Previous results:* If each extract is *annotated* with insert time, $\tilde{O}(\sqrt{N})$ space suffices. [Chu, Kannan, McGregor '07]

  ins(5), ins(3), ext(3,2), ins(7), ext(5,1), ext(7,4) $\in$ PQ^+

? *Big Question: Is annotation necessary for sub-linear space?*

Dyck Language Problem

[Magniez, Mathieu, Nayak '10]

Dyck Language Problem

[Magniez, Mathieu, Nayak '10]

- DYCK_2 is the set of strings of balanced brackets when there are two different types of brackets:

$(((\,[\,]\,)(\,)[\,]\,)) \in \text{DYCK}_2$ $\qquad$ $([(\,[\,]\,])[\,])) \notin \text{DYCK}_2$

Dyck Language Problem

[Magniez, Mathieu, Nayak '10]

- $DYCK_2$ is the set of strings of balanced brackets when there are two different types of brackets:

  $((([])()[])) \in DYCK_2$ $([([]])[])) \notin DYCK_2$

- DYCK *Problem:* Given *streaming* access to length N string, determine if it's in $DYCK_2$ using $o(N)$ space.

Dyck Language Problem

[Magniez, Mathieu, Nayak '10]

- $DYCK_2$ is the set of strings of balanced brackets when there are two different types of brackets:

  ((([])()[])) $\in$ $DYCK_2$ ([(([]])[])) $\notin$ $DYCK_2$

- DYCK *Problem:* Given *streaming* access to length N string, determine if it's in $DYCK_2$ using o(N) space.

- *Previous results:* Given one pass, $\tilde{O}(\sqrt{N})$ space suffices. If you're allowed one forward pass followed by a reverse pass, $\tilde{O}(\log N)$ space suffices.

Dyck Language Problem

[Magniez, Mathieu, Nayak '10]

- DYCK_2 is the set of strings of balanced brackets when there are two different types of brackets:

  $((([])()[])) \in \mathrm{DYCK}_2$ $([([]])[])) \notin \mathrm{DYCK}_2$

- DYCK *Problem:* Given *streaming* access to length N string, determine if it's in DYCK_2 using o(N) space.

- *Previous results:* Given one pass, $\tilde{O}(\sqrt{N})$ space suffices. If you're allowed one forward pass followed by a reverse pass, $\tilde{O}(\log N)$ space suffices.

? Big Question: *Does* $\tilde{O}(\log N)$ *space suffice if we're only allowed multiple forward passes?*

Outline

Outline

I. *Priority Queues and Memory Checking Algorithms*

- $\tilde{O}(\sqrt{N})$ space algorithm for PQ with no annotations!
- Extensions to stacks, double-ended queues, etc.

Outline

I. *Priority Queues and Memory Checking Algorithms*

- $\tilde{O}(\sqrt{N})$ space algorithm for PQ with no annotations!
- Extensions to stacks, double-ended queues, etc.

II. *Multipass Stream Lower Bounds*

- Any constant pass algorithm for $DYCK_2$ or PQ that only uses forward passes requires $\Omega(\sqrt{N})$ space

Outline

I. *Priority Queues and Memory Checking Algorithms*

- $\tilde{O}(\sqrt{N})$ space algorithm for PQ with no annotations!
- Extensions to stacks, double-ended queues, etc.

II. *Multipass Stream Lower Bounds*

- Any constant pass algorithm for $DYCK_2$ or PQ that only uses forward passes requires $\Omega(\sqrt{N})$ space

III. *Information Complexity Trade-offs for Augmented Index*

- Even multi-round protocol leak information.

I. Memory Checking

II. Lower Bounds

III. Augmented Indexing

1. Memory Checking

PQ Algorithm

PQ Algorithm

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

PQ Algorithm

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

- *Thm:* There exists a O($\sqrt{N}$ log N) space algorithm with O(log N) amortized update time for recognizing PQ.
- *Prelim:* Easy to check that set of values inserted equals set of values extracted using fingerprinting:

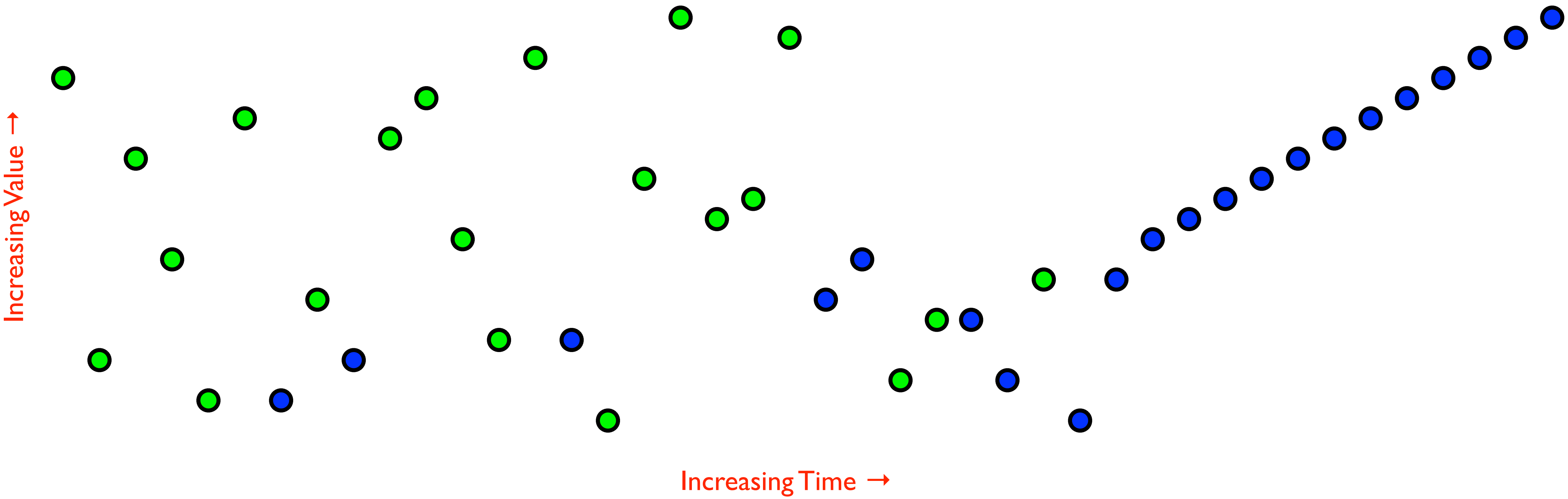
$$f_S(x) = \prod_{a \in S} (x - a) \mod p$$

! *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:*

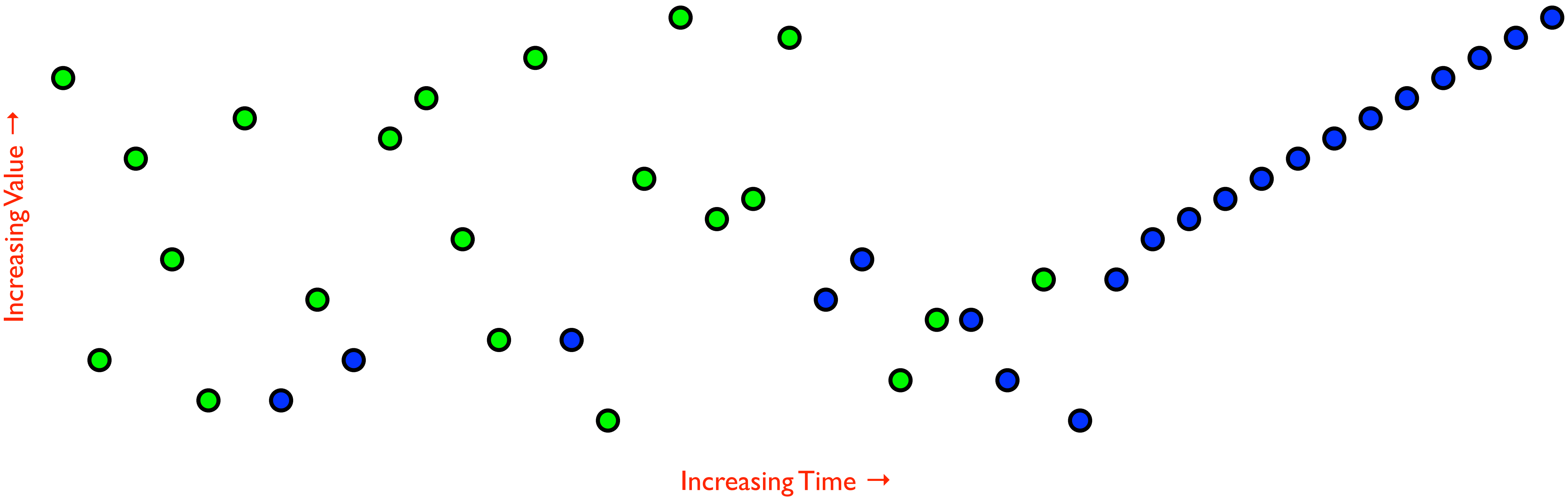
ins(u) .... ext(v) ... ext(u) for some u < v

Epochs and Local Bad Patterns…

Epochs and Local Bad Patterns...

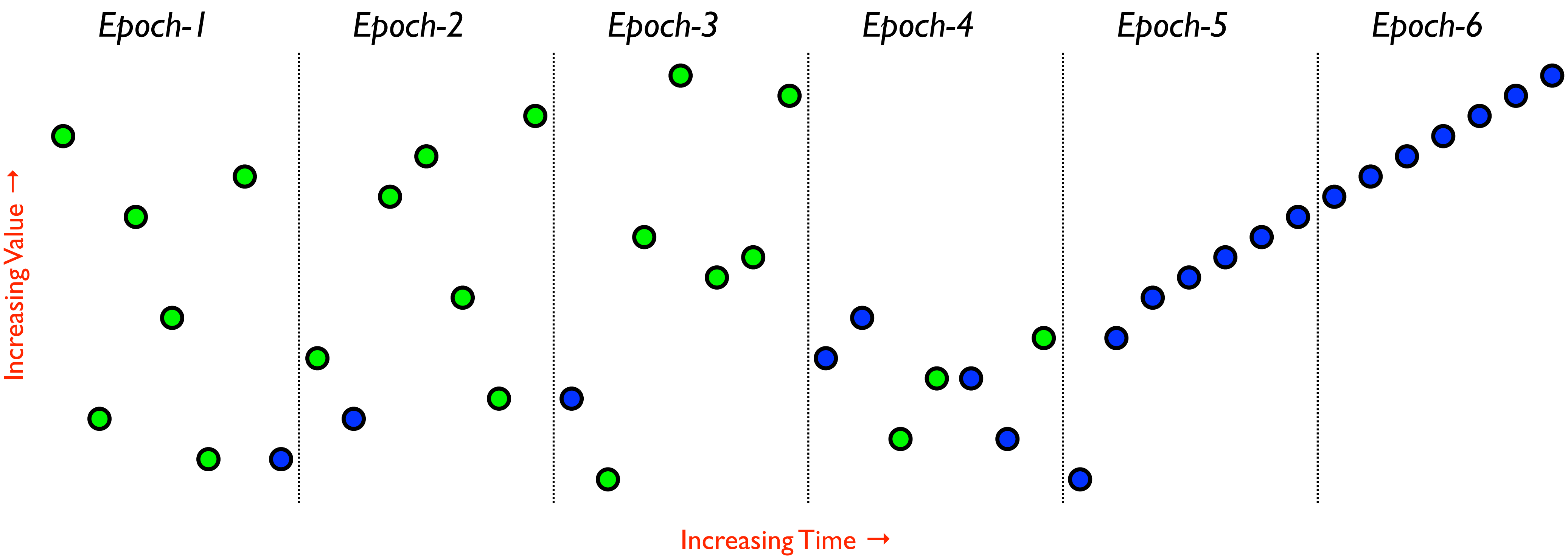


Epochs and Local Bad Patterns...



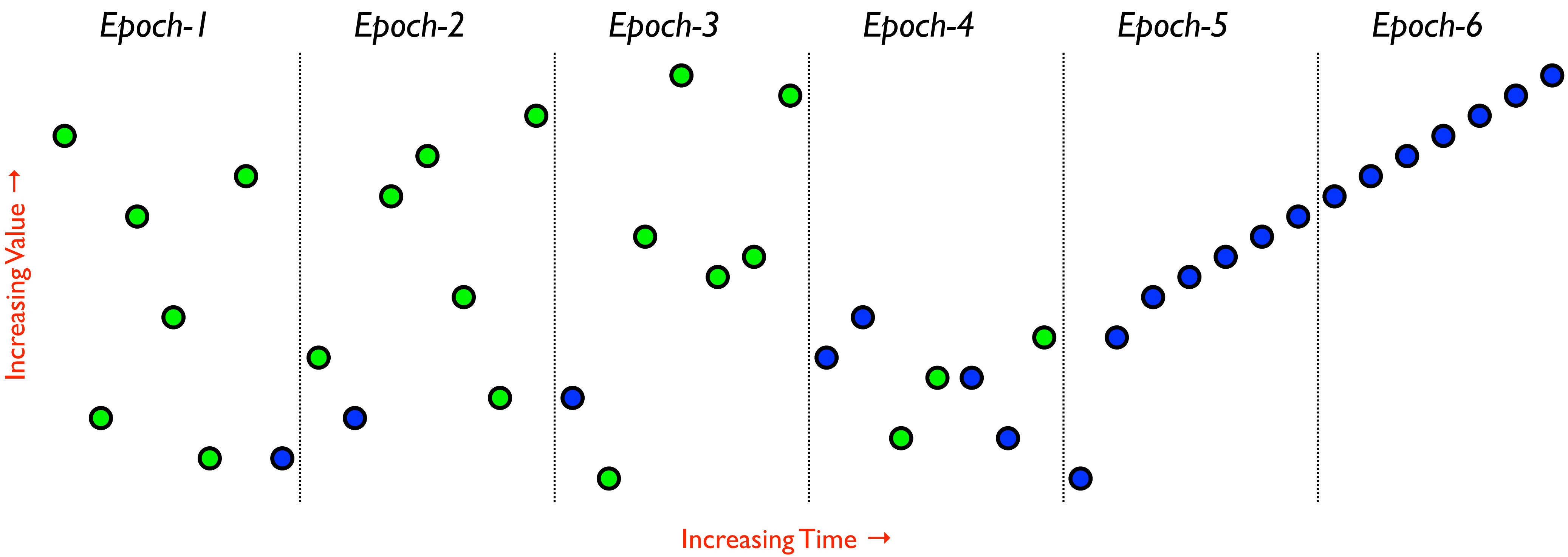
- Split length N sequence into $\sqrt{N}$ epochs of length $\sqrt{N}$

Epochs and Local Bad Patterns...



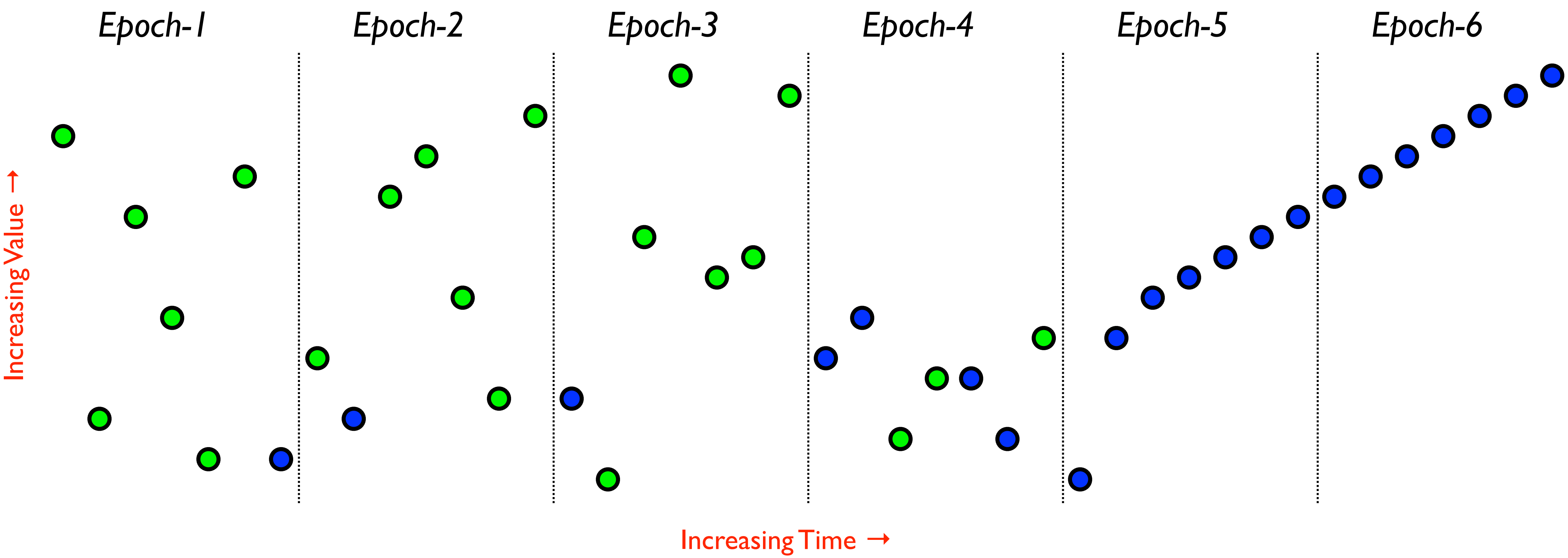
- Split length N sequence into $\sqrt{N}$ epochs of length $\sqrt{N}$

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.

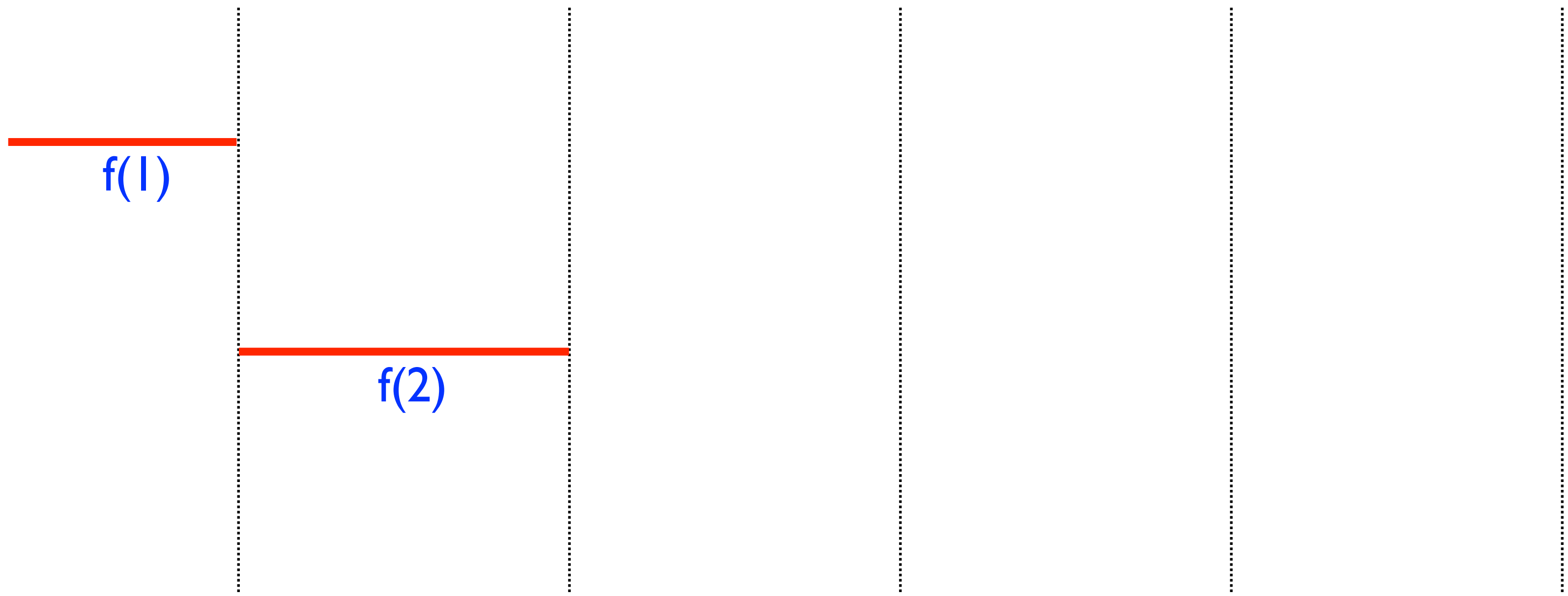
Epochs and Local Bad Patterns...



- 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.

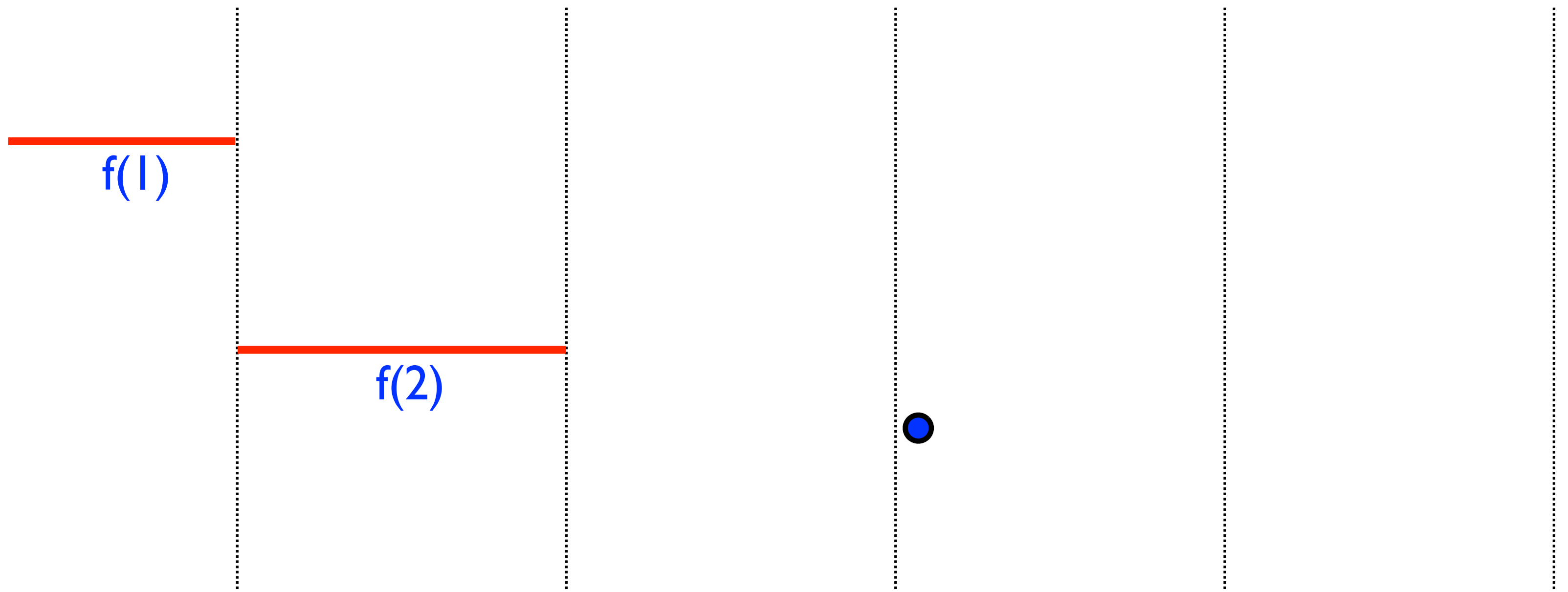
Catching Long-Range Bad Patterns… 1/2

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



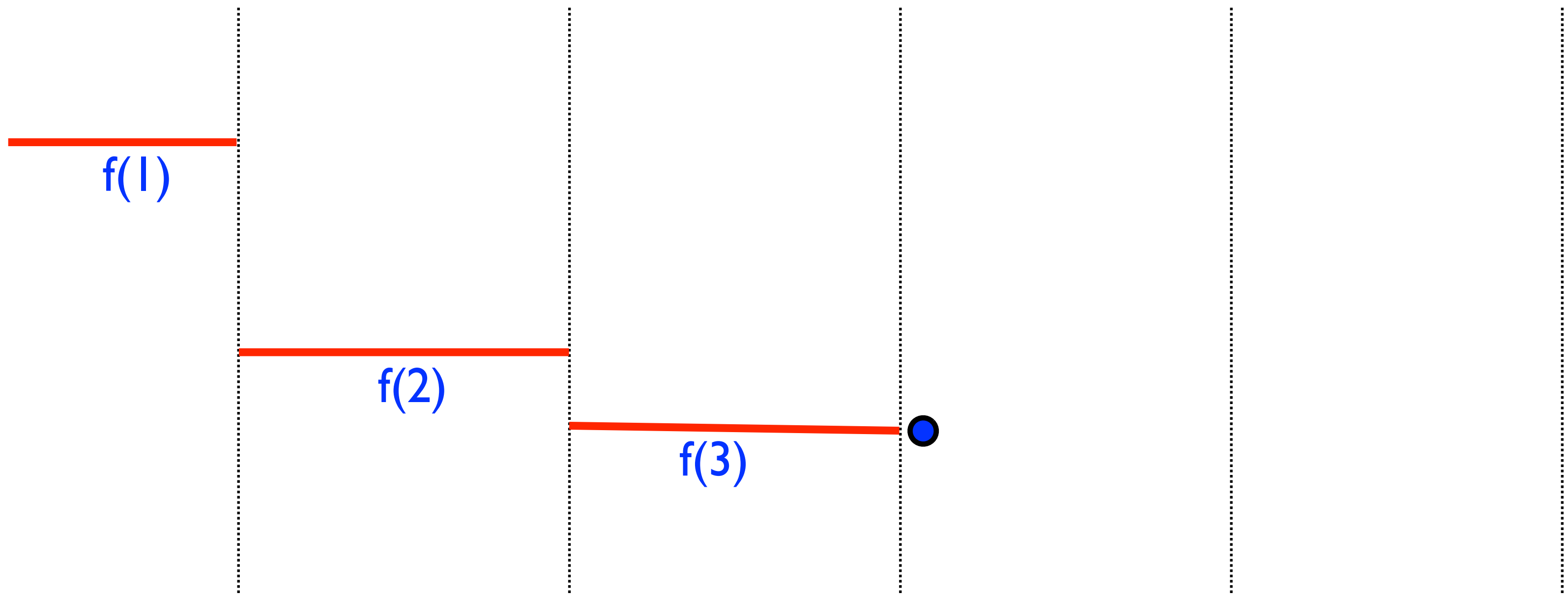
- *Defn:* Let $f_t(i)$ be maximum value extracted between end of i-th epoch and *present time* t.

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



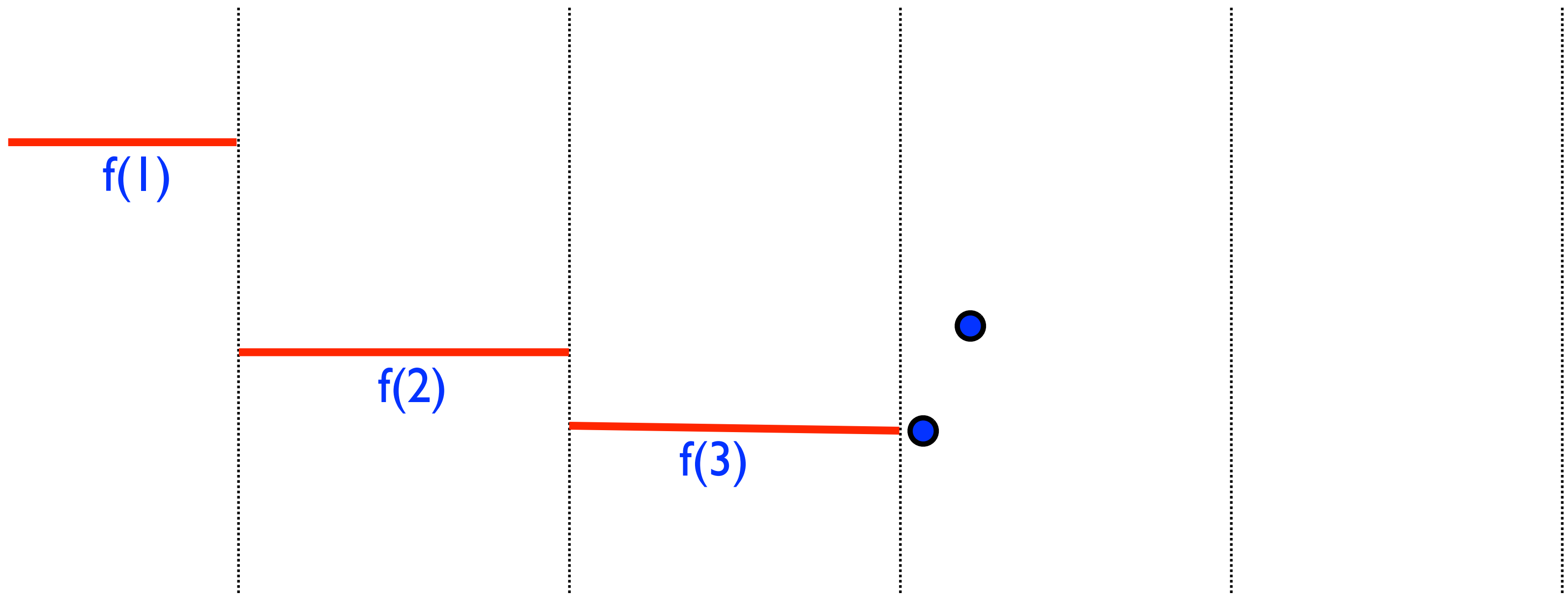
- *Defn:* Let $f_t(i)$ be maximum value extracted between end of i-th epoch and *present time* t.

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



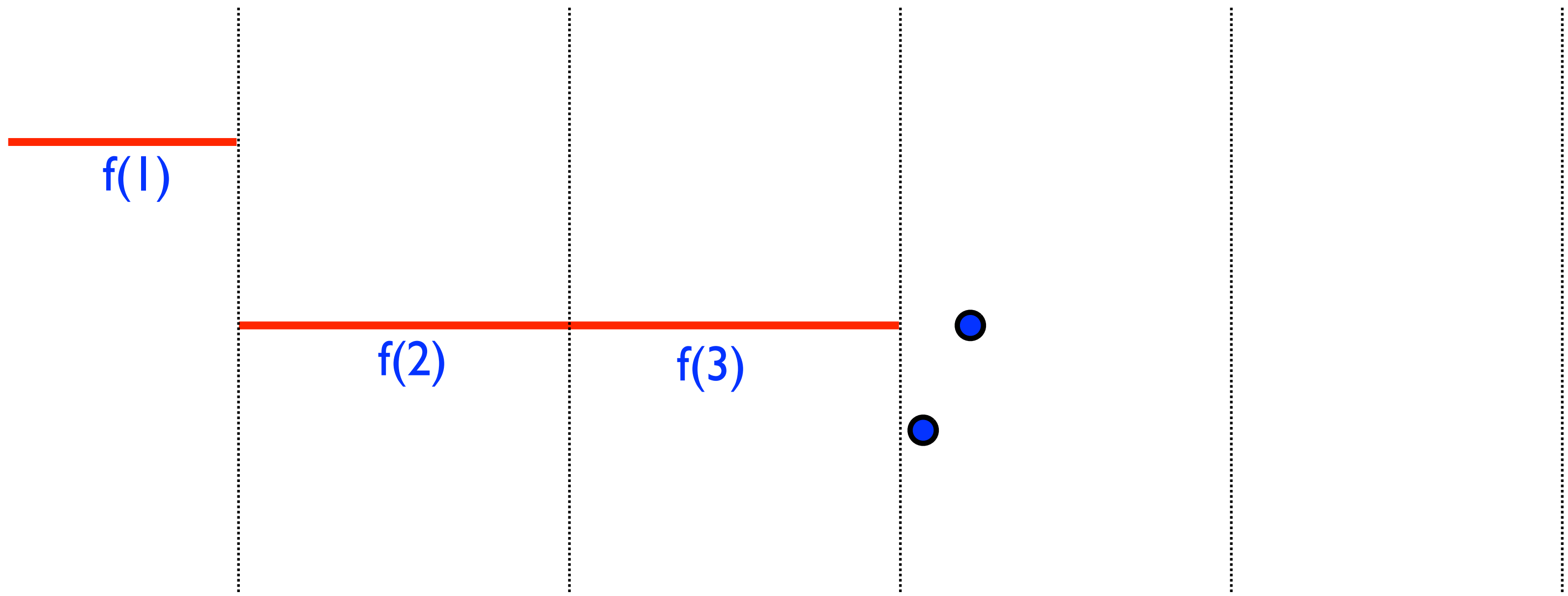
- *Defn:* Let $f_t(i)$ be maximum value extracted between end of i-th epoch and *present time* t.

Catching Long-Range Bad Patterns… 1/2



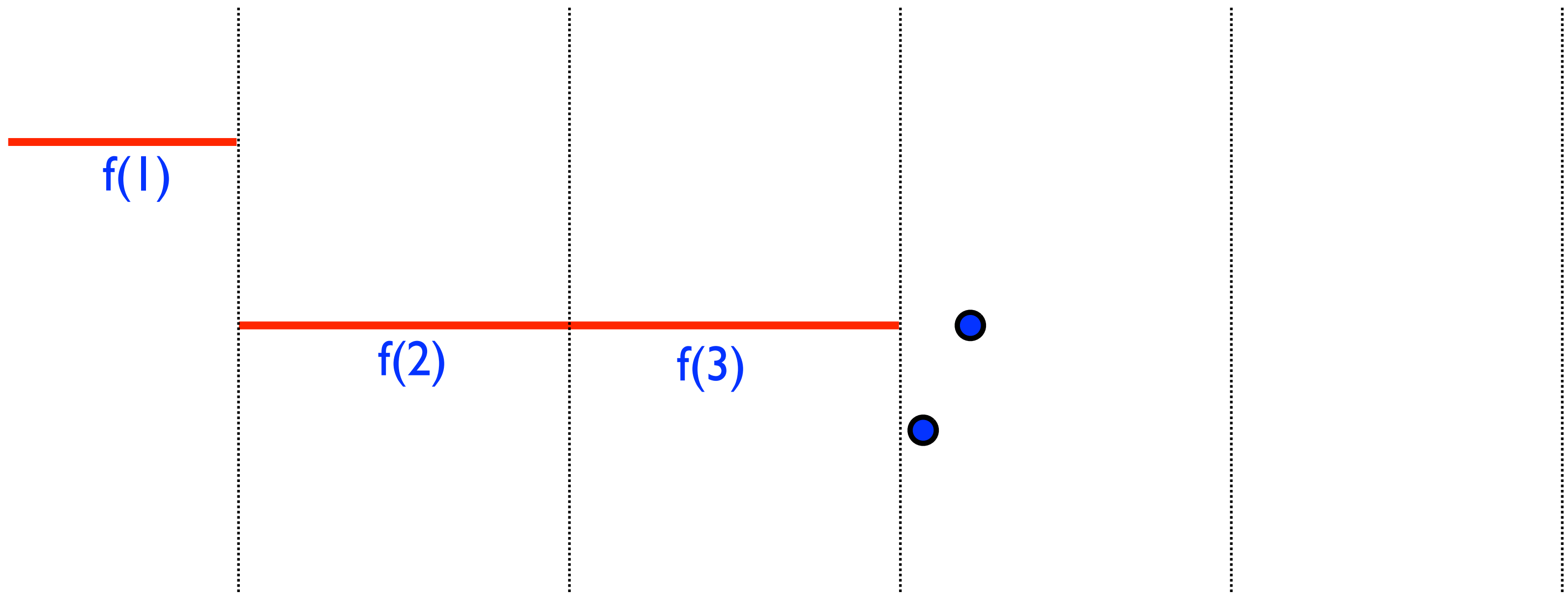
- *Defn:* Let $f_t(i)$ be maximum value extracted between end of i-th epoch and *present time* t.

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



- *Defn:* Let $f_t(i)$ be maximum value extracted between end of i-th epoch and *present time* t.

Catching Long-Range Bad Patterns… 1/2



- ***Defn:*** Let $f_t(i)$ be maximum value extracted between end of i-th epoch and *present time* t.
- ***Defn:*** Each insert or extract is *adopted* by k-th epoch where $k = \min\{j : f(j) \leq u\}$ where we assume f(current epoch)=0.

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:*
  i. 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 $f(k) \geq v > u$ and hence ext(u) will be adopted by k'-th epoch for some k'>k.

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)≥v>u and hence ext(u) will be adopted by k'-th epoch for some k'>k.
- *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 $f(k) \geq v > u$ and hence ext(u) will be adopted by k'-th epoch for some k'>k.
- *Lemma:* If there are no bad patterns, every ins(u) and ext(u) pair get adopted by the same epoch.
- *Algorithm:* Using fingerprinting to check:

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

Finishing Up

Finishing Up

- *Some steps to removing assumptions:*

  i. Within buffered epoch, rearrange terms such that all inserts follow a series of increasing extracts: ensures all extracts are adopted by previous epochs.

  ii. Keep track of number of times epoch k adopts ins(u) while $f(k)<u$ and while $f(k)=u$ separately and whether epoch k has ever adopted more ext(u)'s than ins(u)'s.

Finishing Up

- *Some steps to removing assumptions:*
   1. Within buffered epoch, rearrange terms such that all inserts follow a series of increasing extracts: ensures all extracts are adopted by previous epochs.
   2. Keep track of number of times epoch k adopts ins(u) while f(k)<u and while f(k)=u separately and whether epoch k has ever adopted more ext(u)'s than ins(u)'s.
- *Thm:* There exists a $O(\sqrt{N} \log N)$ space algorithm with $O(\log N)$ amortized update time for recognizing PQ.

Finishing Up

- *Some steps to removing assumptions:*

  i. Within buffered epoch, rearrange terms such that all inserts follow a series of increasing extracts: ensures all extracts are adopted by previous epochs.

  ii. Keep track of number of times epoch k adopts ins(u) while $f(k)<u$ and while $f(k)=u$ separately and whether epoch k has ever adopted more ext(u)'s than ins(u)'s.

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

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

I. Memory Checking

II. Lower Bounds

III. Augmented Indexing

II. Lower Bounds

Augmented Index

Augmented Index

- Many space lower bounds in data stream model are based on reductions from communication complexity.

Augmented Index

$x \in \{0,1\}^n$

$y \in \{0,1\}^{k-1}$, $k \in [n]$, $c \in \{0,1\}$

- Many space lower bounds in data stream model are based on reductions from communication complexity.
- *Augmented Index:* Alice has $x \in \{0,1\}^n$ and Bob has a prefix $y \in \{0,1\}^{k-1}$ of x and $c \in \{0,1\}$. Bob wants to check if $c = x_k$.

Augmented Index

$x \in \{0,1\}^n$

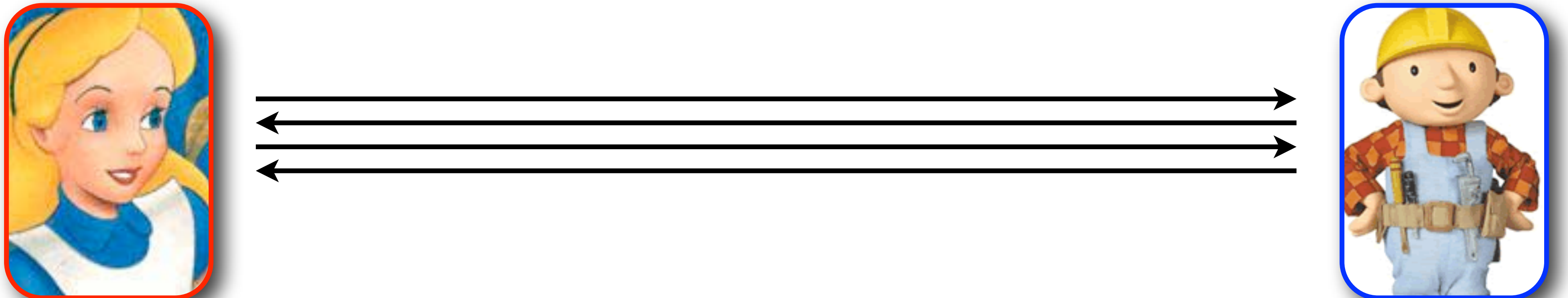
$y \in \{0,1\}^{k-1}$, $k \in [n]$, $c \in \{0,1\}$

- Many space lower bounds in data stream model are based on reductions from communication complexity.

- *Augmented Index:* Alice has $x \in \{0,1\}^n$ and Bob has a prefix $y \in \{0,1\}^{k-1}$ of x and $c \in \{0,1\}$. Bob wants to check if $c = x_k$.

- *Thm:* Any 1/3-error, one-way protocol from Alice to Bob for AI_n requires $\Omega(n)$ bits sent. [Miltersen et al. JCSS '98]

Augmented Index



$x \in \{0,1\}^n$ $y \in \{0,1\}^{k-1}, k \in [n], c \in \{0,1\}$

- Many space lower bounds in data stream model are based on reductions from communication complexity.
- *Augmented Index:* Alice has $x \in \{0,1\}^n$ and Bob has a prefix $y \in \{0,1\}^{k-1}$ of x and $c \in \{0,1\}$. Bob wants to check if $c = x_k$.
- *Thm:* Any 1/3-error, one-way protocol from Alice to Bob for AI_n requires $\Omega(n)$ bits sent. [Miltersen et al. JCSS '98]
- Our main result concerns multi-way protocols but we'll cover the relevance to $DYCK_2$ and PQ first...

Multi-player Augmented Index

Multi-player Augmented Index


x^1


y^1, k^1, c^1


x^2


y^2, k^2, c^2


x^3


y^3, k^3, c^3

- We now have 2m players $A_1, \ldots, A_m, B_1, \ldots, B_m$ where each A_i and B_i have an instance (x^i, k^i, c^i) of AI_n

Multi-player Augmented Index

- We now have 2m players $A_1, \ldots, A_m, B_1, \ldots, B_m$ where each A_i and B_i have an instance (x^i, k^i, c^i) of AI_n
- Want to determine if any of the AI instances are false using private messages communicated in the order

$$A_1 \rightarrow B_1 \rightarrow A_2 \rightarrow B_2 \rightarrow \ldots \rightarrow A_m \rightarrow B_m \rightarrow A_m \rightarrow A_{m-1} \rightarrow \ldots \rightarrow A_1$$

Multi-player Augmented Index

x^1 y^1, k^1, c^1 x^2 y^2, k^2, c^2 x^3 y^3, k^3, c^3 x^3 x^2 x^1

- We now have 2m players $A_1, \ldots, A_m, B_1, \ldots, B_m$ where each A_i and B_i have an instance (x^i, k^i, c^i) of AI_n
- Want to determine if any of the AI instances are false using private messages communicated in the order

$$A_1 \to B_1 \to A_2 \to B_2 \to \ldots \to A_m \to B_m \to A_m \to A_{m-1} \to \ldots \to A_1$$

- *Thm:* Any 1/3-error, p-round protocol for $\text{MULTI-AI}_{m,n}$ needs $ps = \Omega(\min m, n)$ where s is max message length.

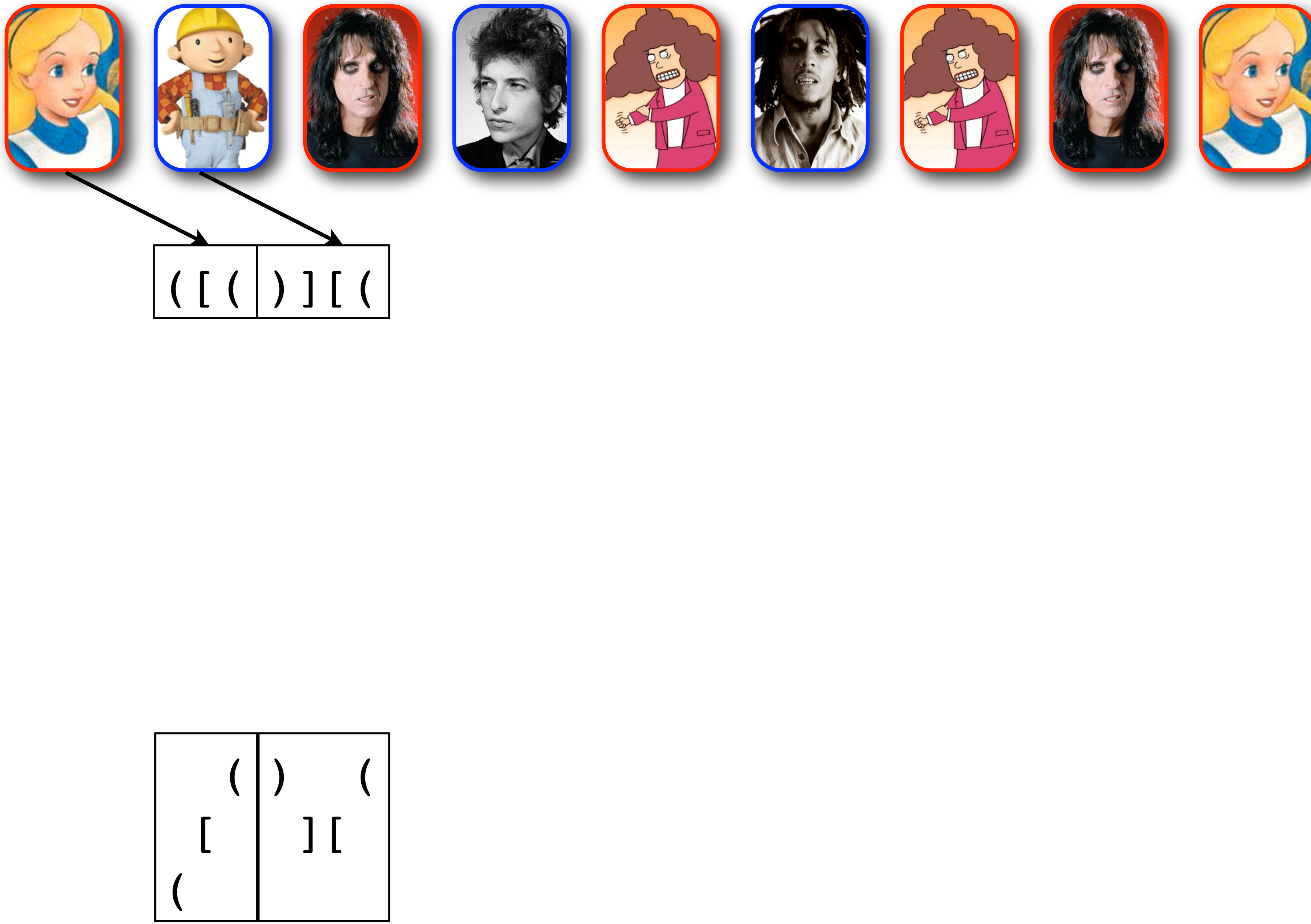
Reduction to Dyck

Reduction to Dyck

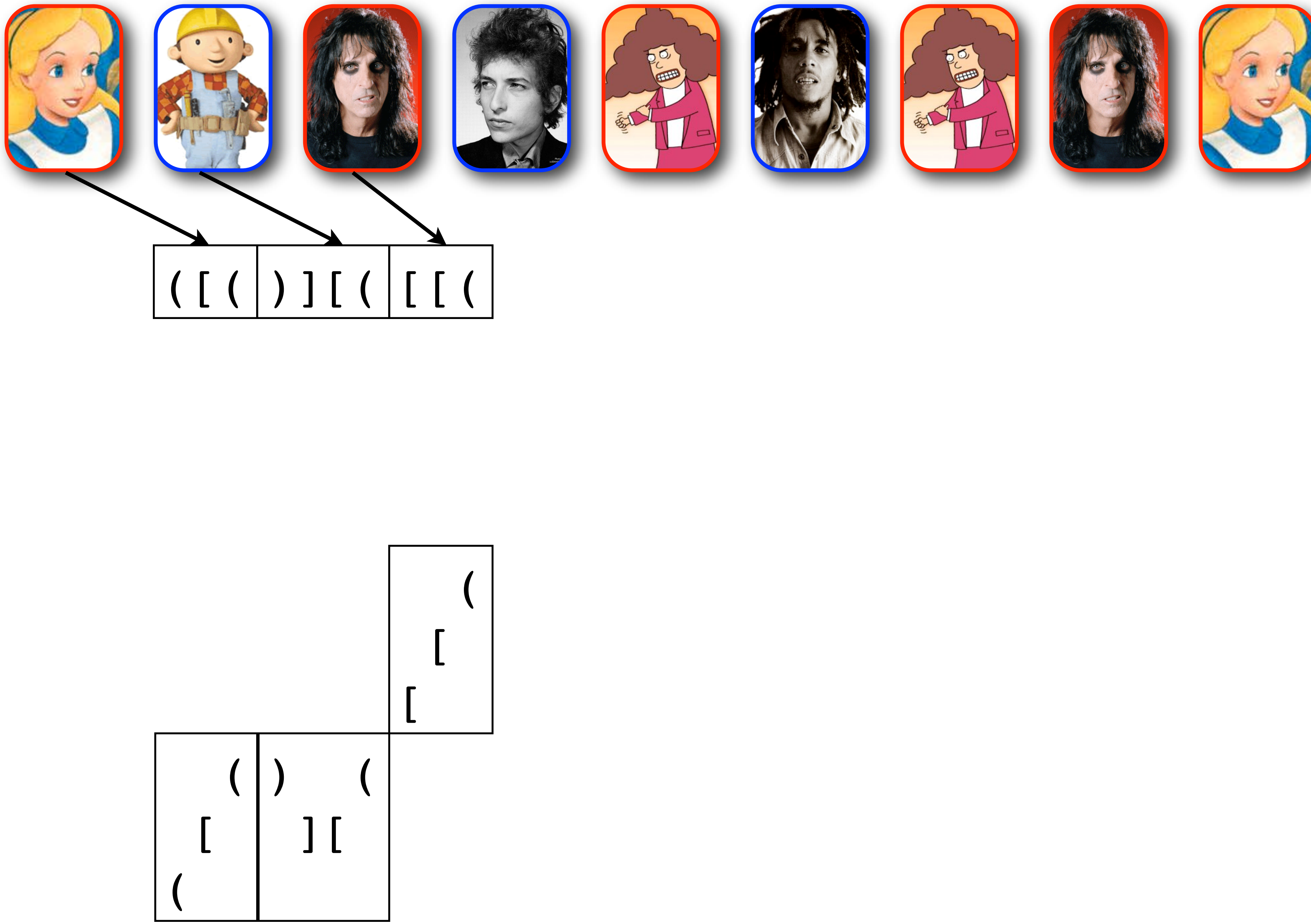
([(

(
[
(

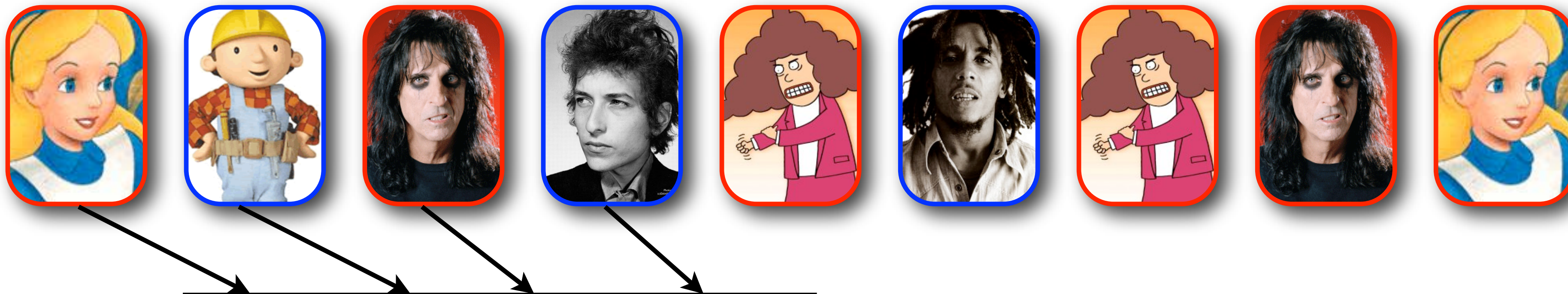
Reduction to Dyck



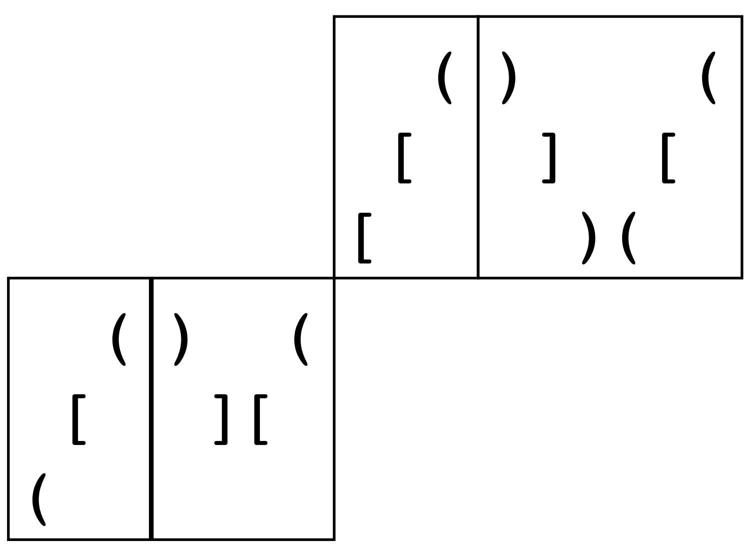
Reduction to Dyck



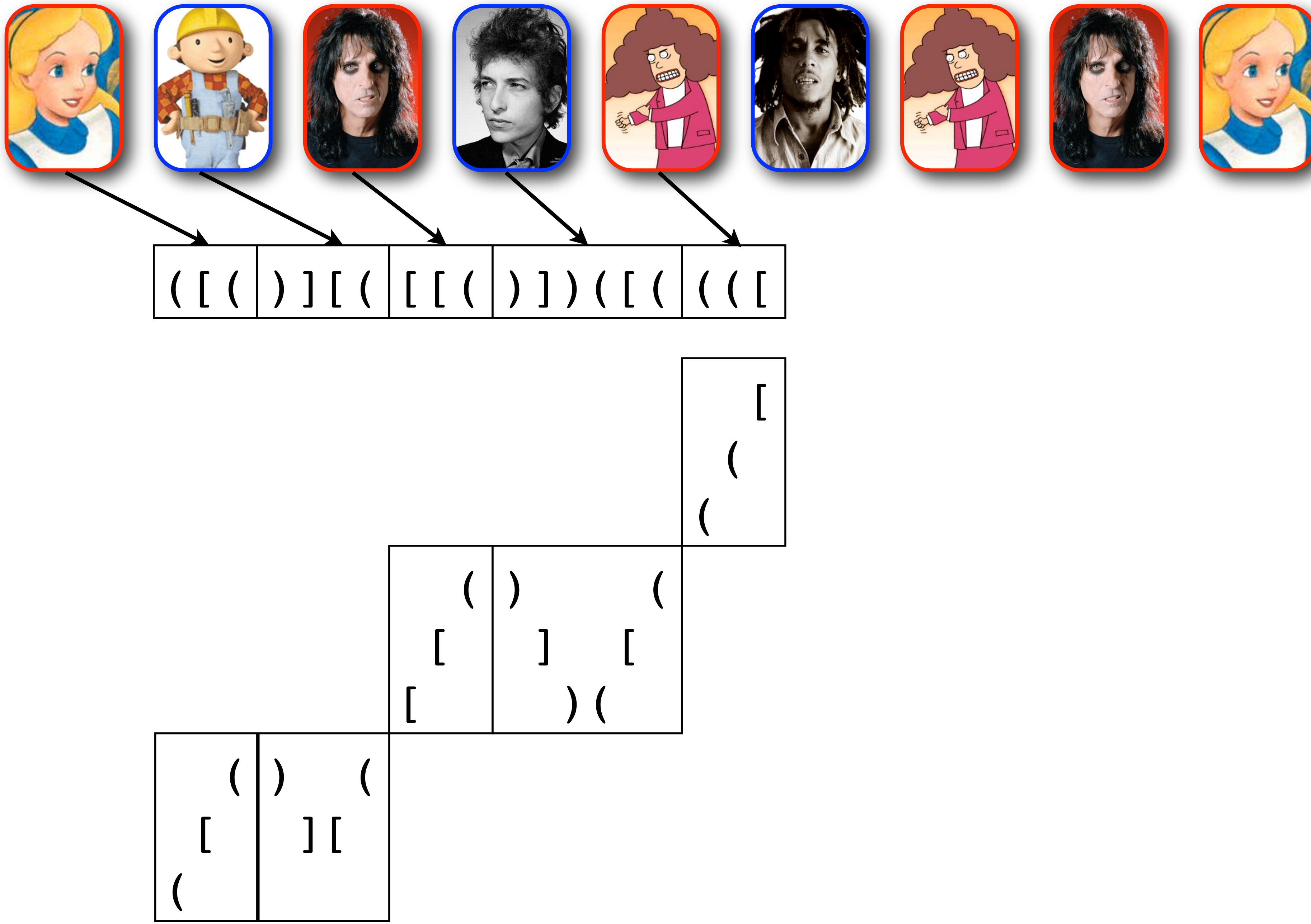
Reduction to Dyck



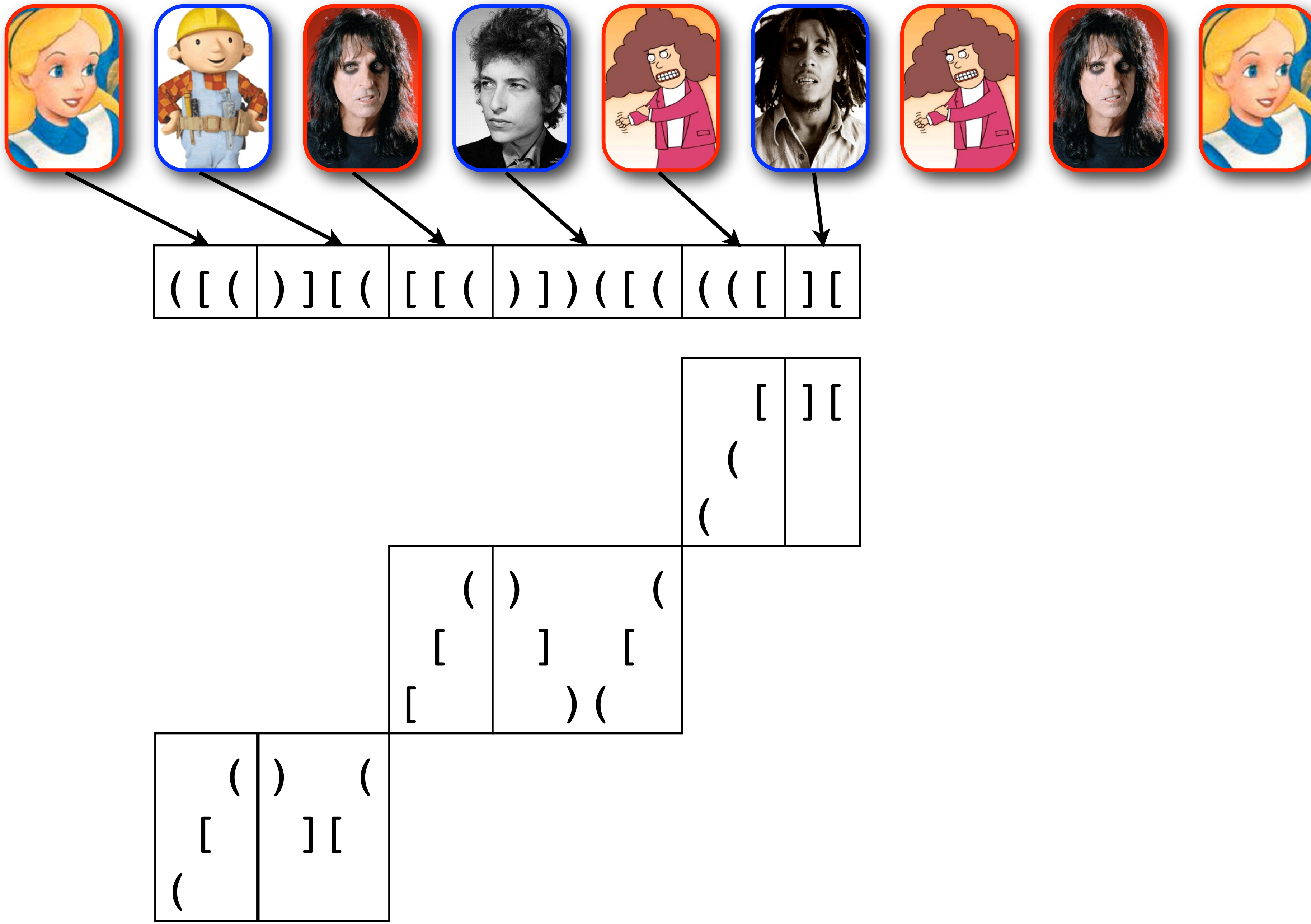
([(	)] [(	[[(	)]) ([(



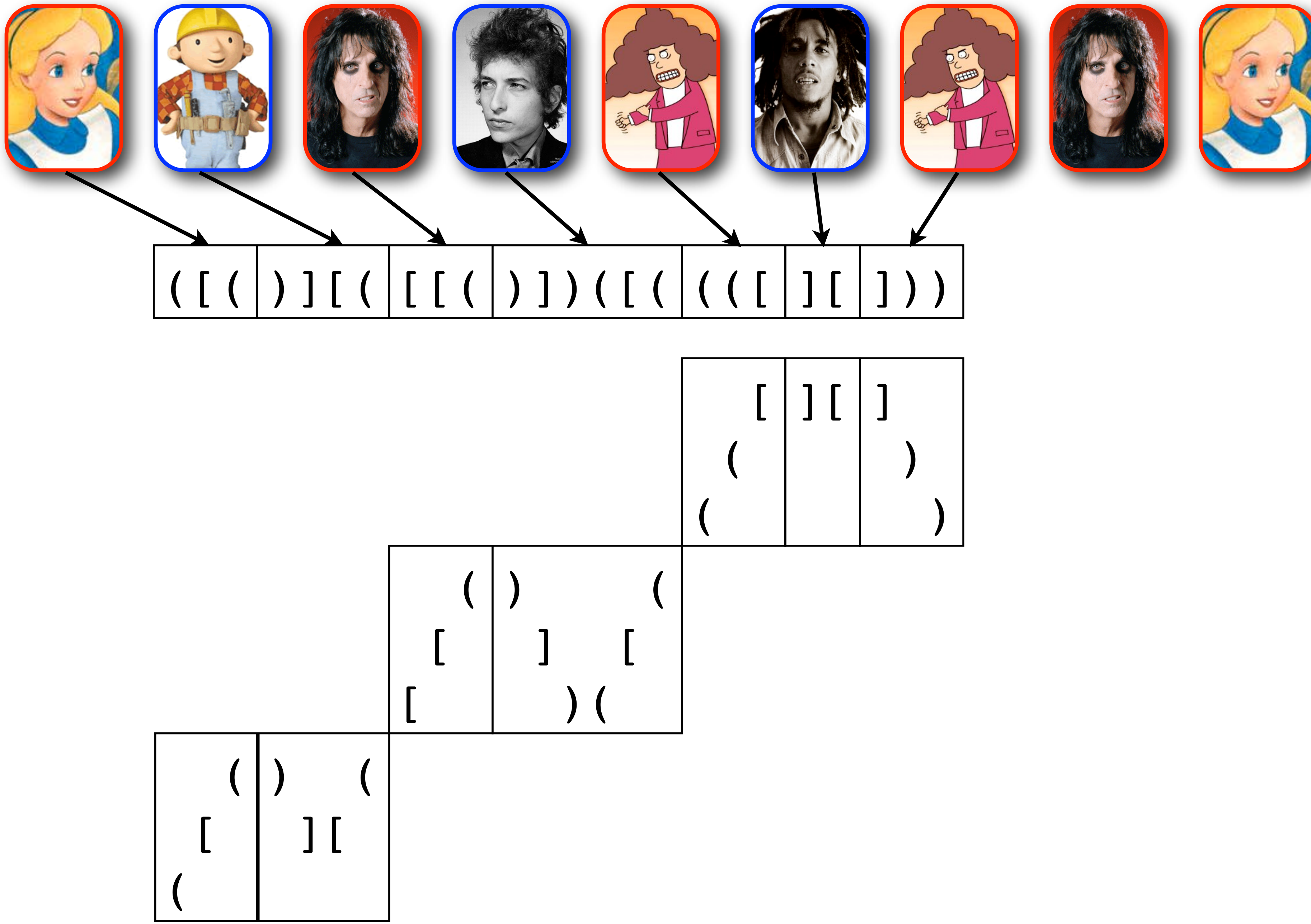
Reduction to Dyck



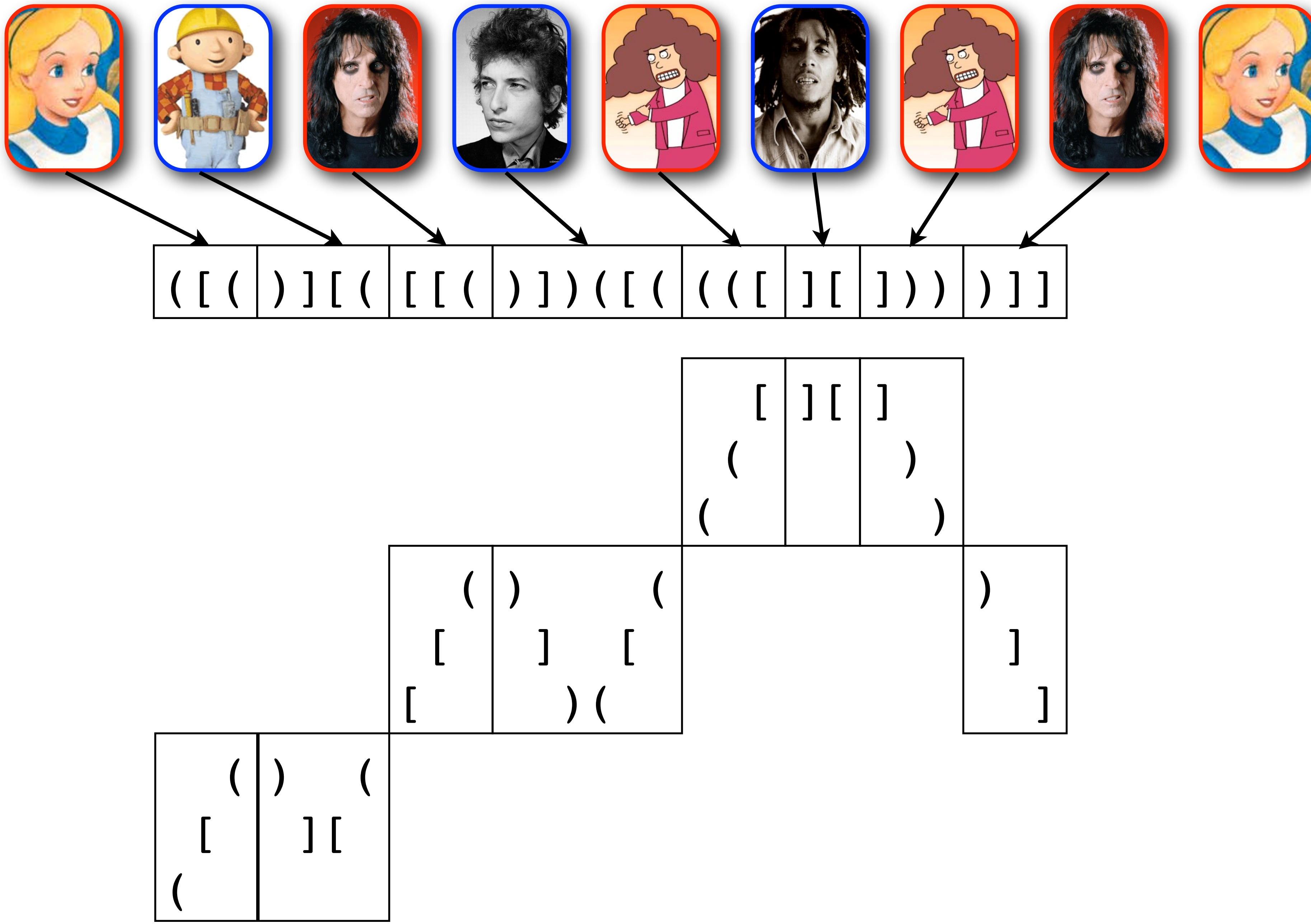
Reduction to Dyck



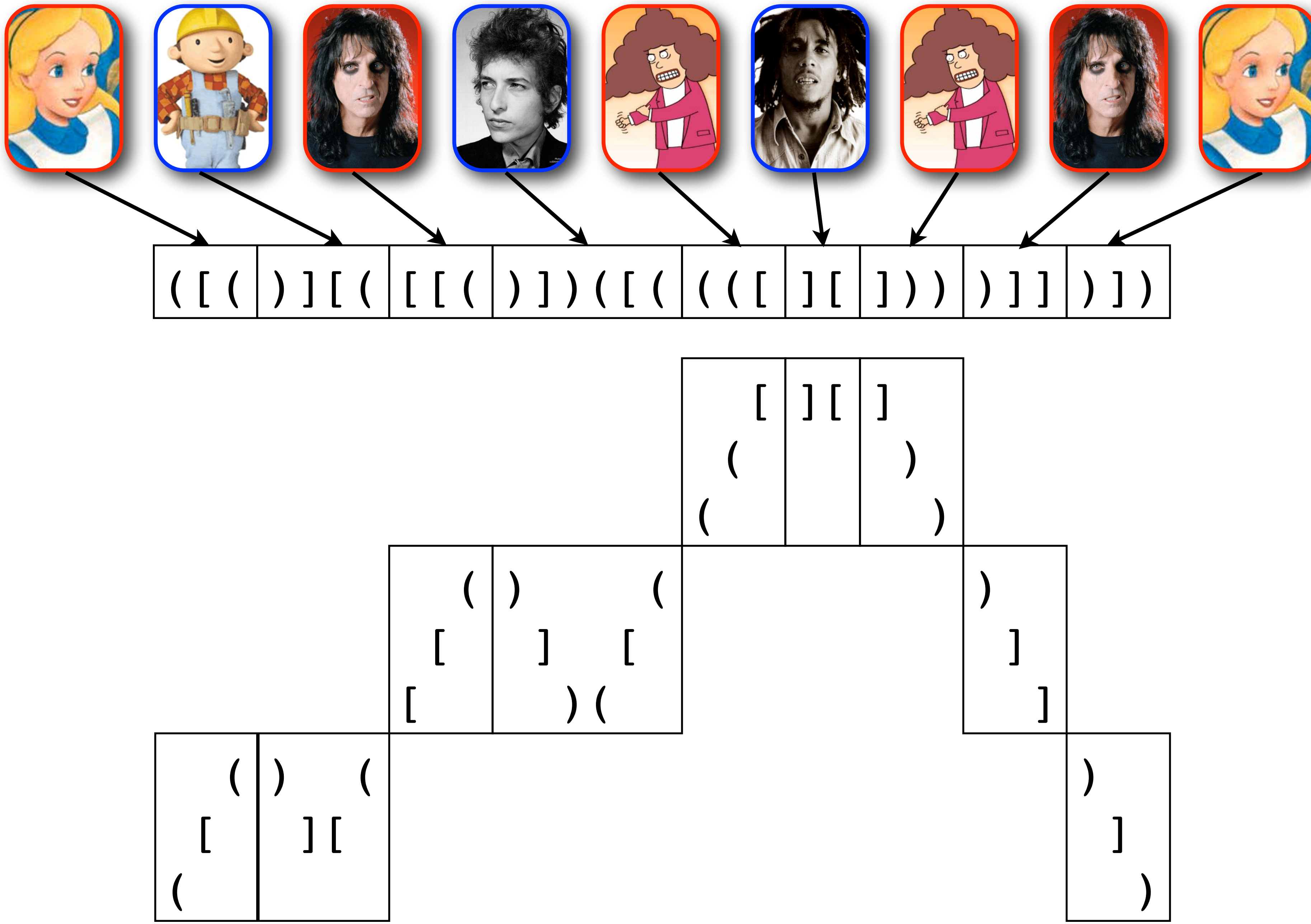
Reduction to Dyck



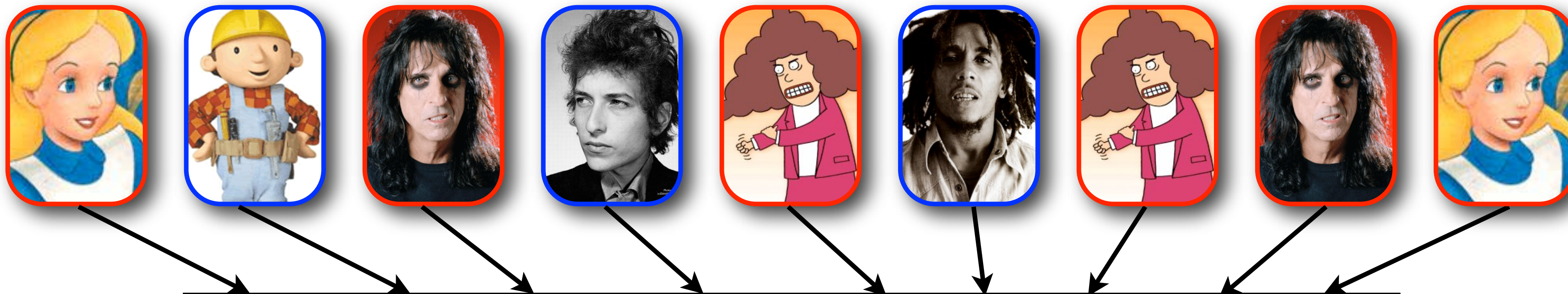
Reduction to Dyck



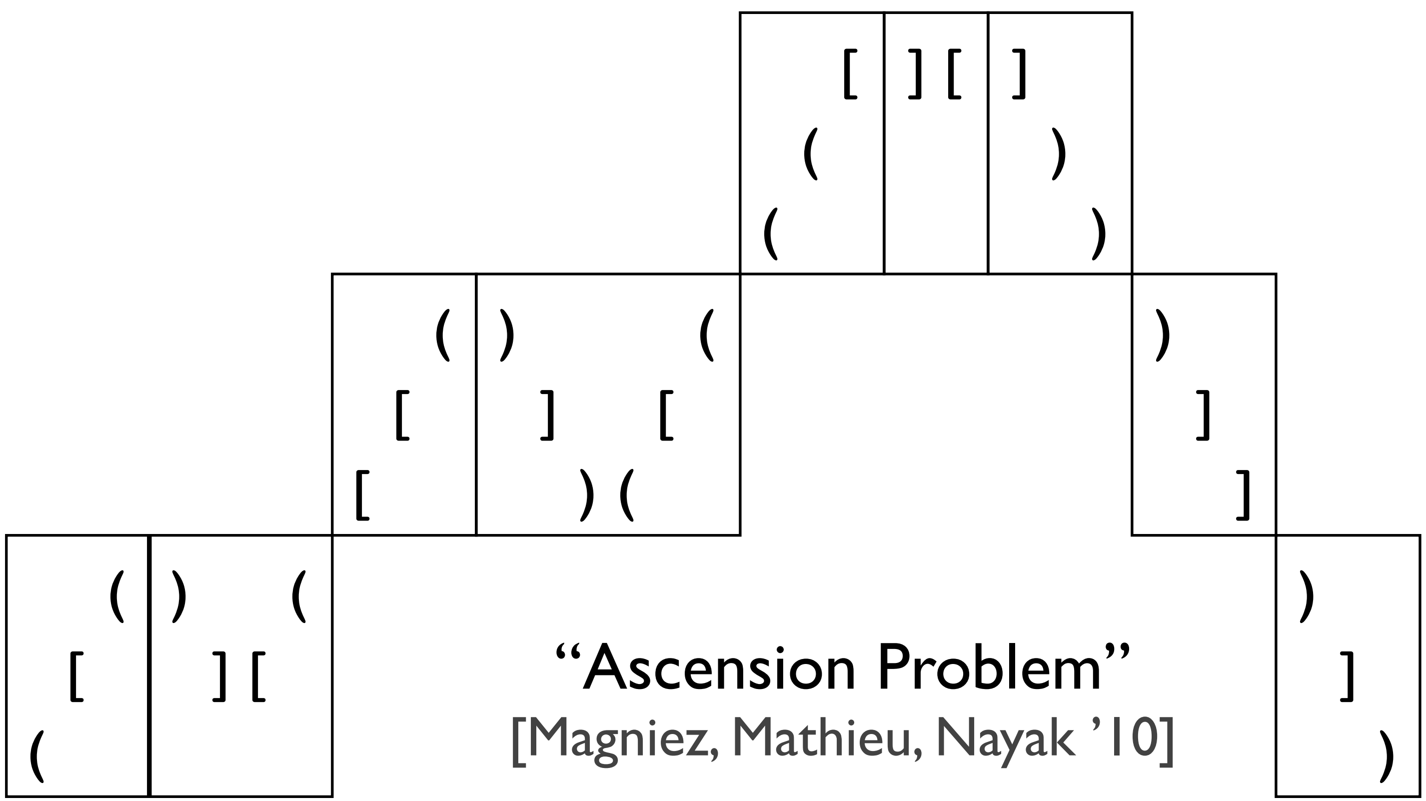
Reduction to Dyck



Reduction to Dyck



([(	)] [(	[[(	)]) ([(	(([	] [	]))	)]]	)])



"Ascension Problem"

[Magniez, Mathieu, Nayak '10]

Reduction to Dyck

Reduction to Dyck

- *Thm:* A constant-pass, algorithm for DYCK_2 that fail with probability at most 1/3 requires $\Omega(\sqrt{N})$ space.

Reduction to Dyck

- *Thm:* A constant-pass, algorithm for DYCK_2 that fail with probability at most 1/3 requires $\Omega(\sqrt{N})$ space.
- *Proof:*

  i. Let **A** be a p-pass stream algorithm using s space.

Reduction to Dyck

- *Thm:* A constant-pass, algorithm for DYCK$_2$ that fail with probability at most 1/3 requires $\Omega(\sqrt{N})$ space.

- *Proof:*

  i. Let **A** be a p-pass stream algorithm using s space.

  ii. Use **A** to construct a p-round protocol for MULTI-AI$_{\sqrt{N},\sqrt{N}}$ where max message is s-bits: Each player simulates **A** on it's part of input using Magniez et al. reduction and forwards memory state to next player.

Reduction to Dyck

- *Thm:* A constant-pass, algorithm for DYCK_2 that fail with probability at most 1/3 requires $\Omega(\sqrt{N})$ space.
- *Proof:*
  - *i.* Let **A** be a p-pass stream algorithm using s space.
  - *ii.* Use **A** to construct a p-round protocol for $\text{MULTI-AI}_{\sqrt{N},\sqrt{N}}$ where max message is s-bits: Each player simulates **A** on it's part of input using Magniez et al. reduction and forwards memory state to next player.
  - *iii.* Therefore s is $\Omega(\sqrt{N})$ as required.

Lower Bounds Summary

Lower Bounds Summary

- ***Thm:*** Any constant pass algorithm for recognizing PQ or $DYCK_2$ requires a $\Omega(\sqrt{N})$ space.

Lower Bounds Summary

- ***Thm:*** Any constant pass algorithm for recognizing PQ or $DYCK_2$ requires a $\Omega(\sqrt{N})$ space.
- ***Consequences:***

    i. Multiple forward passes have no significant advantage for recognizing the languages considered.

    ii. One forward pass + one reverse pass is exponentially more powerful than two forward passes.

I. Memory Checking

II. Lower Bounds

III. Augmented Indexing

III. Augmented Indexing

Information Complexity

[Chakrabarti, Shi, Wirth, Yao '01]

Information Complexity

[Chakrabarti, Shi, Wirth, Yao '01]

- ***Entropy and Mutual Information:***

$$
\begin{aligned}
H(X) &= -\Sigma \Pr[X = x] \lg \Pr[X = x] \\
H(X|Y) &= -\Sigma \Pr[X = x, Y = y] \lg \Pr[X = x | Y = y]
\end{aligned}
$$

Information Complexity

[Chakrabarti, Shi, Wirth, Yao '01]

- *Entropy and Mutual Information:*

$$
\begin{aligned}
H(X) &= -\Sigma \Pr[X = x] \lg \Pr[X = x] \\
H(X|Y) &= -\Sigma \Pr[X = x, Y = y] \lg \Pr[X = x | Y = y] \\
I(X;Y) &= H(X) - H(X|Y) \;=\; H(Y) - H(Y|X) \\
I(X;Y|Z) &= H(X|Z) - H(X|Y,Z)
\end{aligned}
$$

Information Complexity

[Chakrabarti, Shi, Wirth, Yao '01]

- *Entropy and Mutual Information:*

$$\begin{aligned} H(X) &= -\Sigma \Pr[X = x] \lg \Pr[X = x] \\ H(X|Y) &= -\Sigma \Pr[X = x, Y = y] \lg \Pr[X = x | Y = y] \\ I(X;Y) &= H(X) - H(X|Y) \;=\; H(Y) - H(Y|X) \\ I(X;Y|Z) &= H(X|Z) - H(X|Y,Z) \end{aligned}$$

- *Information cost method:* Consider mutual information between random input for a communication problem and the communication transcript:

$$I(\text{transcript}; \text{input})$$

Information Complexity

[Chakrabarti, Shi, Wirth, Yao '01]

- *Entropy and Mutual Information:*

$$\begin{aligned} H(X) &= -\Sigma \Pr[X = x] \lg \Pr[X = x] \\ H(X|Y) &= -\Sigma \Pr[X = x, Y = y] \lg \Pr[X = x | Y = y] \\ I(X;Y) &= H(X) - H(X|Y) \;=\; H(Y) - H(Y|X) \\ I(X;Y|Z) &= H(X|Z) - H(X|Y,Z) \end{aligned}$$

- *Information cost method:* Consider mutual information between random input for a communication problem and the communication transcript:

$$I(\text{transcript}; \text{input}) \leq \text{length of transcript}$$

Information Complexity

[Chakrabarti, Shi, Wirth, Yao '01]

- *Entropy and Mutual Information:*

$$
\begin{aligned}
H(X) &= -\Sigma \Pr[X = x] \lg \Pr[X = x] \\
H(X|Y) &= -\Sigma \Pr[X = x, Y = y] \lg \Pr[X = x | Y = y] \\
I(X;Y) &= H(X) - H(X|Y) \; = \; H(Y) - H(Y|X) \\
I(X;Y|Z) &= H(X|Z) - H(X|Y,Z)
\end{aligned}
$$

- *Information cost method:* Consider mutual information between random input for a communication problem and the communication transcript:

$$I(\text{transcript}; \text{input}) \leq \text{length of transcript}$$

- Can restrict to partial transcript and subsets of input: useful for proving direct-sum arguments.

Information Complexity of AI_n

Information Complexity of AI_n

- ***Defn:*** Let P be a protocol for AI_n using public random string R. Let T be the transcript and (X, K, C)~ξ. Define

$$
\begin{aligned}
\mathrm{icost}^{A}_{\xi}(\mathrm{P}) &= I(T : X \mid K, C, R) \\
\mathrm{icost}^{B}_{\xi}(\mathrm{P}) &= I(T : K, C \mid X, R)
\end{aligned}
$$

Information Complexity of AI_n

- *Defn:* Let P be a protocol for AI_n using public random string R. Let T be the transcript and (X, K, C)~ξ. Define

$$\begin{aligned} \mathrm{icost}^{A}_{\xi}(P) &= I(T : X \mid K, C, R) \\ \mathrm{icost}^{B}_{\xi}(P) &= I(T : K, C \mid X, R) \end{aligned}$$

- *Thm:* Let P be a randomized protocol for AI_n with error 1/3 under the uniform distribution μ. Then,

$$\mathrm{icost}^{A}_{\mu_0}(P) = \Omega(n) \quad \text{or} \quad \mathrm{icost}^{B}_{\mu_0}(P) = \Omega(1)$$

where μ_0 is μ conditioned on $X_K=C$.

MULTI-$AI_{m,n}$ versus AI_n

MULTI-$AI_{m,n}$ versus AI_n

- ***Defn:*** Let Q be a protocol for MULTI-$AI_{m,n}$ using public random string R. Let T be transcript and $(X^i,K^i,C^i)_{i\in[m]}\sim\xi$.

$$\mathrm{icost}_\xi(Q) = I(T_m : K^1, C^1, \ldots, K^m, C^m \mid X^1, \ldots, X^m, R)$$

where T_m is the set of messages sent by B_m.

MULTI-AI$_{m,n}$ versus AI$_n$

- ***Defn:*** Let Q be a protocol for MULTI-AI$_{m,n}$ using public random string R. Let T be transcript and $(X^i,K^i,C^i)_{i\in[m]}\sim\xi$.

$$\text{icost}_\xi(Q) = I(T_m : K^1, C^1, \ldots, K^m, C^m \mid X^1, \ldots, X^m, R)$$

where T_m is the set of messages sent by B_m.

- ***Thm (Direct Sum):*** If there exists a p-round, s-bit, ε-error protocol Q for MULTI-AI$_{m,n}$ then there exists a p-round, ε-error randomized protocol P for AI$_n$ where

i. Alice sends at most ps bits

ii. $m \cdot \text{icost}^B_{\mu_0}(P) \leq \text{icost}_{\mu_0^{\otimes m}}(Q)$

Putting it all together...

Putting it all together...

- *Thm:* Any p-round, s-bit, 1/3-error protocol Q for MULTI-$AI_{m,n}$ requires $ps=\Omega(\min m,n)$.

Putting it all together...

- *Thm:* Any p-round, s-bit, 1/3-error protocol Q for MULTI-$AI_{m,n}$ requires ps=Ω(min m,n).
- *Proof:*

  i. By direct sum theorem, there exists ε-error, p-pass protocol P for AI_n such that:

$$p \cdot s \geq \text{icost}_{\mu_0^{\otimes m}}(Q) \geq m \cdot \text{icost}^{B}_{\mu_0}(P)$$

$$p \cdot s \geq \text{icost}^{A}_{\mu_0}(P)$$

Putting it all together...

- *Thm:* Any p-round, s-bit, 1/3-error protocol Q for MULTI-$\mathrm{AI}_{m,n}$ requires ps=Ω(min m,n).
- *Proof:*

   i. By direct sum theorem, there exists ε-error, p-pass protocol P for AI_n such that:

$$p \cdot s \geq \mathrm{icost}_{\mu_0^{\otimes m}}(Q) \geq m \cdot \mathrm{icost}^B_{\mu_0}(P)$$

$$p \cdot s \geq \mathrm{icost}^A_{\mu_0}(P)$$

   ii. By information complexity of AI_n

$$\max(m \cdot \mathrm{icost}^B_{\mu_0}(P), \mathrm{icost}^A_{\mu_0}(P)) = \Omega(\min(m, n))$$

Summary

Memory Checking: Sub-linear space recognition of various data-structure transcript languages is possible without annotation!

Theory of Stream Computation: Forward + reverse pass can be much more useful than many forward passes!

Further Work: Annotations, stream language recognition, ...

Thanks!