Anti-Persistence on Persistent Storage: History-Independent Sparse Tables and Dictionaries

Michael A. Bender, Jonathon W. Berry[*], **Rob Johnson**
Thomas M. Kroeger[*], Samuel McCauley, Cynthia A. Phillips[*]
Bertrand Simon[‡], Shikha Singh, David Zage[†]

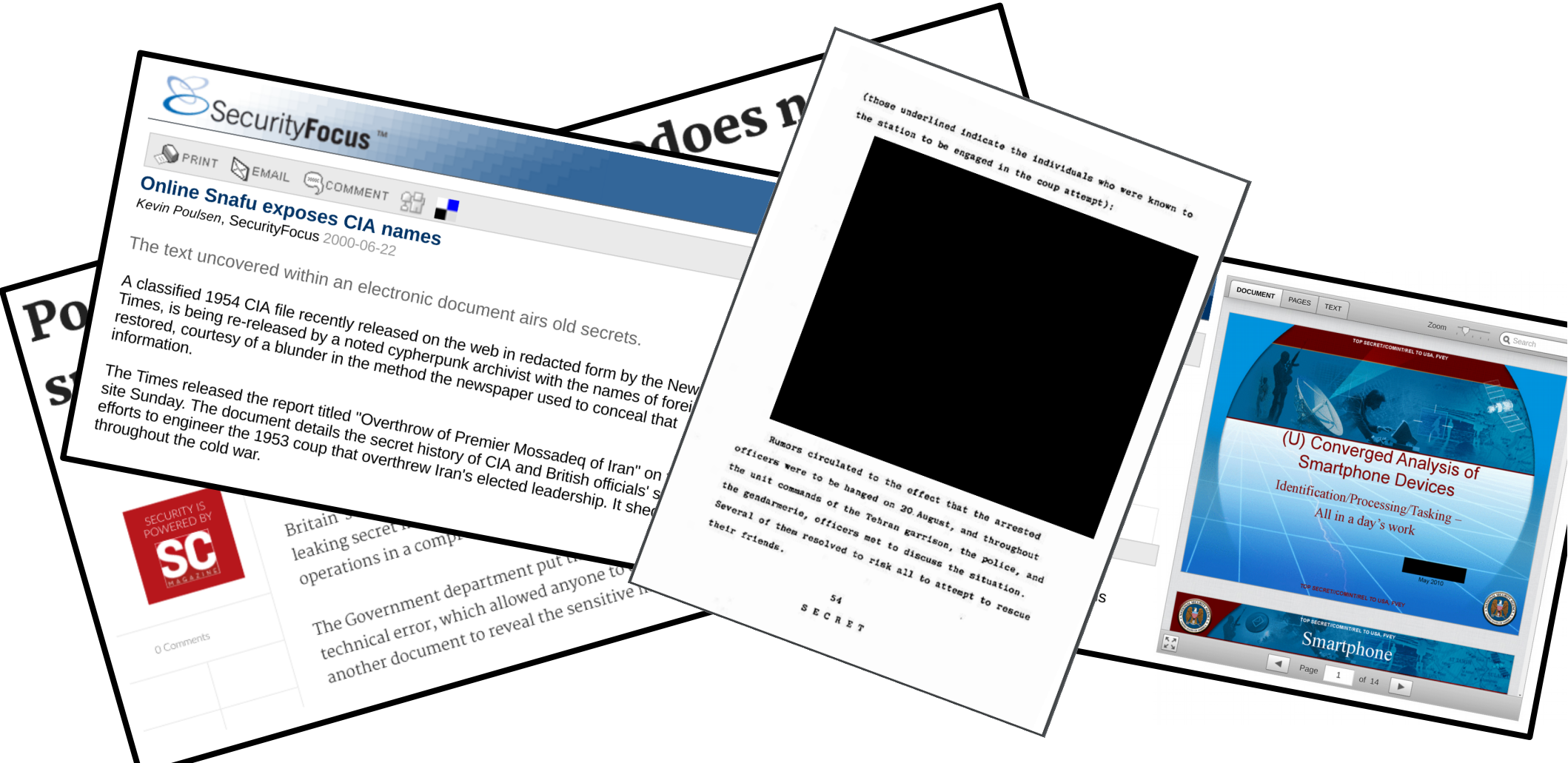
Stony Brook University

[*]Sandia Labs

[‡]Stony Brook University &
Ecole Normale Supérieure de Lyon

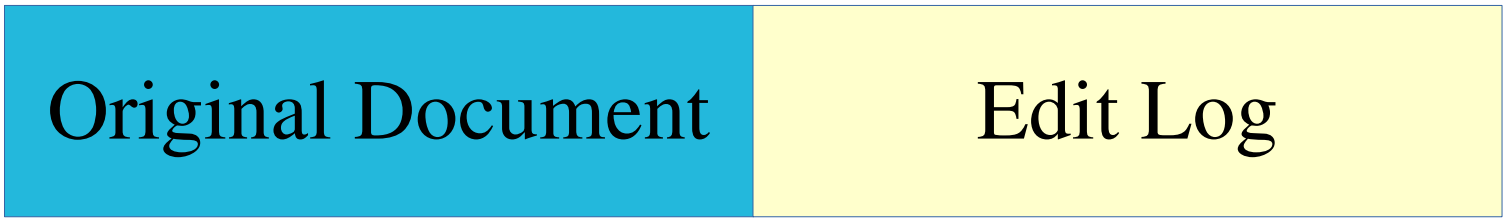
[†]Sandia Labs & Intel

History Can Be as Important as Content



Systems sacrifice security for I/O efficiency

- Example: Microsoft Word “Fast Save” appends edit log to document

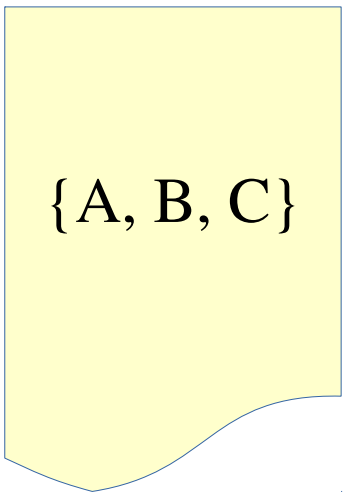


- Adversary can recover old versions of the document
- 10-30% of Word documents online have “Fast Save” data

History-Independent Data Structures [Naor & Teague '01]

[Blelloch & Golovin '07] [Buchbinder & Petrank '03] [Bajaj, Chakrabati, Sion '15] [Bajaj & Sion '13] [Molnar, Kohno, Sastry, Wagner '06] [Moran, Naor, Segev '07] [Naor, Segev, Wieder '08] [Roche, Aviv, Choi '15] [Tzouramanis '12] [Golovin '08, '09, '10]

- Bit representation reveals no additional info about past states of the data structure
- Example:



Observer cannot infer sequence of operations leading to current state

1. Insert A
2. Insert B
3. Insert C
4. Insert D
5. Delete D

1. Insert C
2. Insert B
3. Insert A

This Paper: I/O-Efficient History-Independent Data Structures

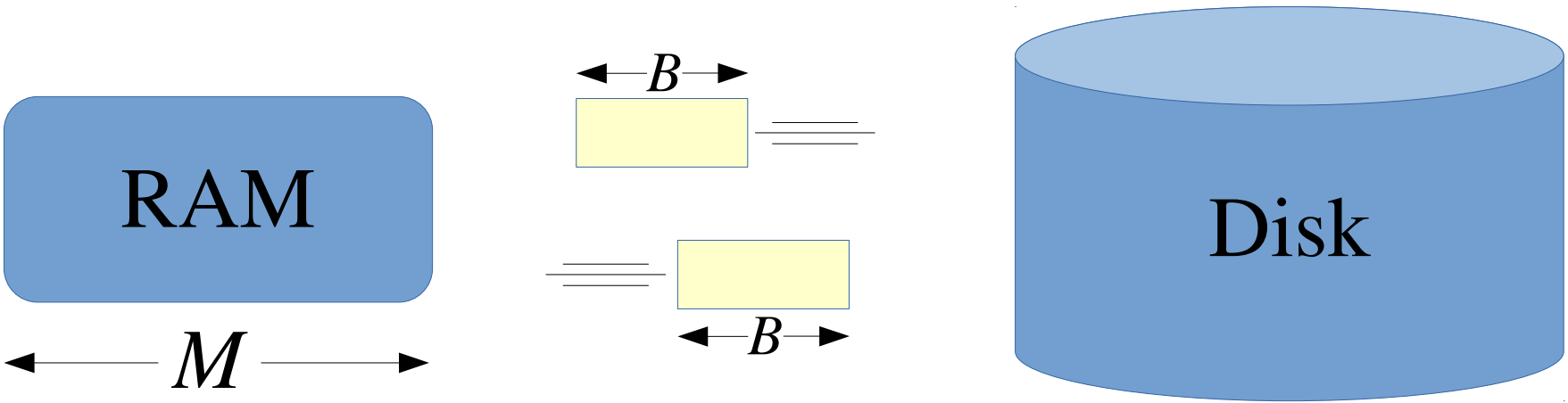
- Three history-independent data structures
  - Packed Memory Array (PMA)
  - Cache-oblivious B-tree
  - External-memory skip list
- Same computational and I/O complexities as non-HI versions

This Paper: I/O-Efficient History-Independent Data Structures

- Three history-independent data structures
  - Packed Memory Array (PMA)
  - Cache-oblivious B-tree
  - External-memory skip list
- Same computational and I/O complexities as non-HI versions

Disk Access Machine (DAM) Model [Aggarwal & Vitter '88]

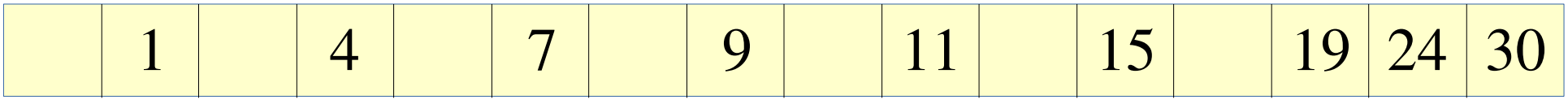
- Data is transferred in blocks between RAM and disk, and time is measured in terms of block transfers
  - Time bounds parameterized by block size ***B***, memory size ***M***, data size ***N***
- Accessing cached blocks costs 0 time, uncached blocks cost 1



Packed Memory Arrays [Itai, Konheim, Rodeh '81] [Bender, Demaine, Farach-Colton '00] [Katriel '02] [Willard '82]

- Maintain a dynamic array in physical order
- Leave gaps for future insertions
  - Amortized insertion/deletion I/O cost: $O\left(\frac{\log^2 N}{B}\right)$
  - Better than B-tree $O(\log_B N)$ when $\log N < B / \log B$
- Space: $O(N)$
- Lookups: $O(\log_B N)$ Range queries of k items: $O(k/B)$ I/Os

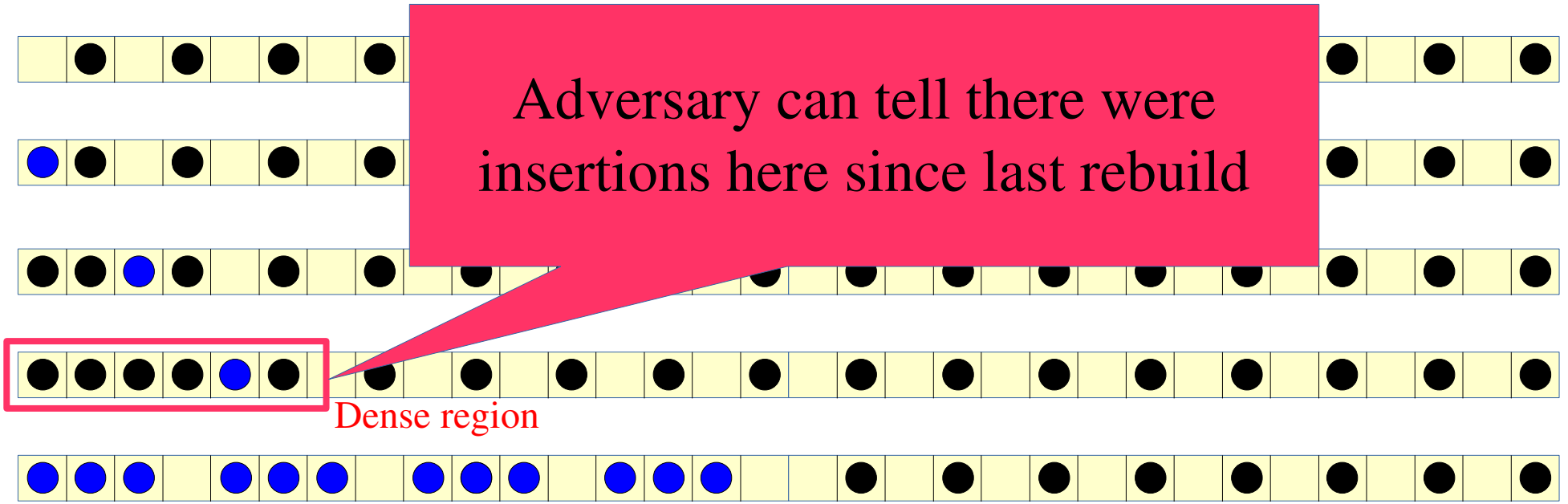
Useful building block for other data structures, e.g. cache-oblivious B-trees



Physical array

Current PMAs are Inherently History Dependent

- PMAs redistribute elements when a region gets too “dense”



The Challenge of Building a HI PMA

Inserts

Rebuilds

Elements Evenly Distributed

A History-Independent PMA

1. Initial element layout
2. Handling insertions/deletions

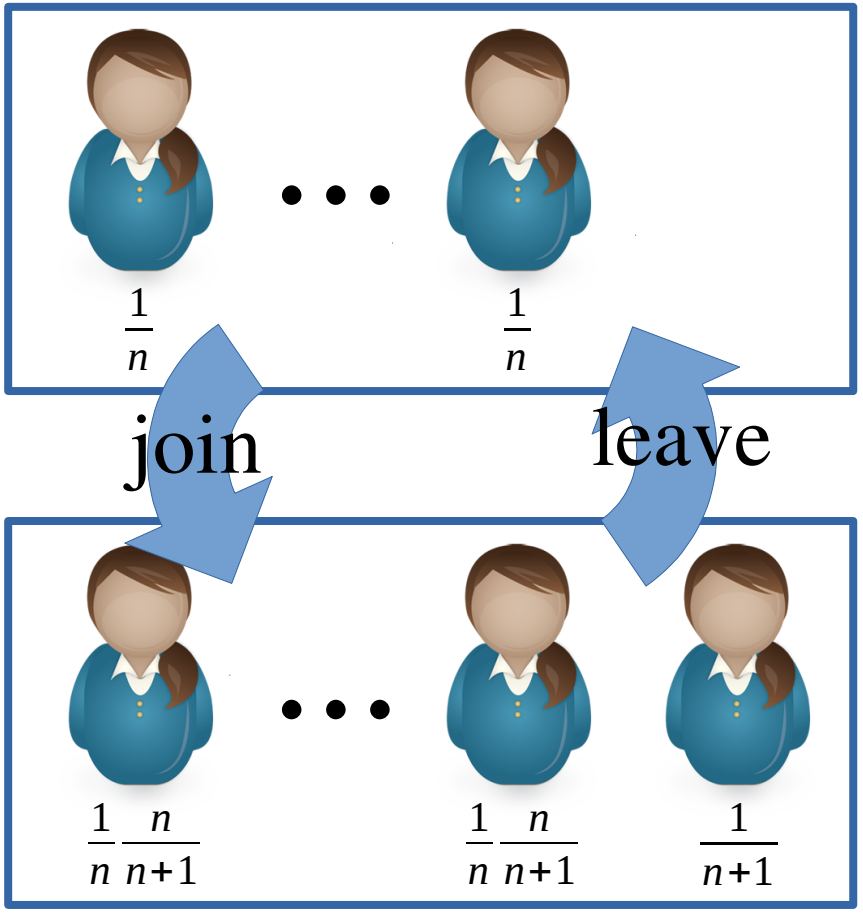
Reservoir Sampling

Reservoir Sampling with Joins & Leaves [Vitter '85]

- Two goals:
  - Maintain a club leader uniformly randomly from all current club members
  - Make leader changes rare as members join and leave

1. Elect new member w/ prob $1/(n+1)$
2. Elect new leader when leader leaves

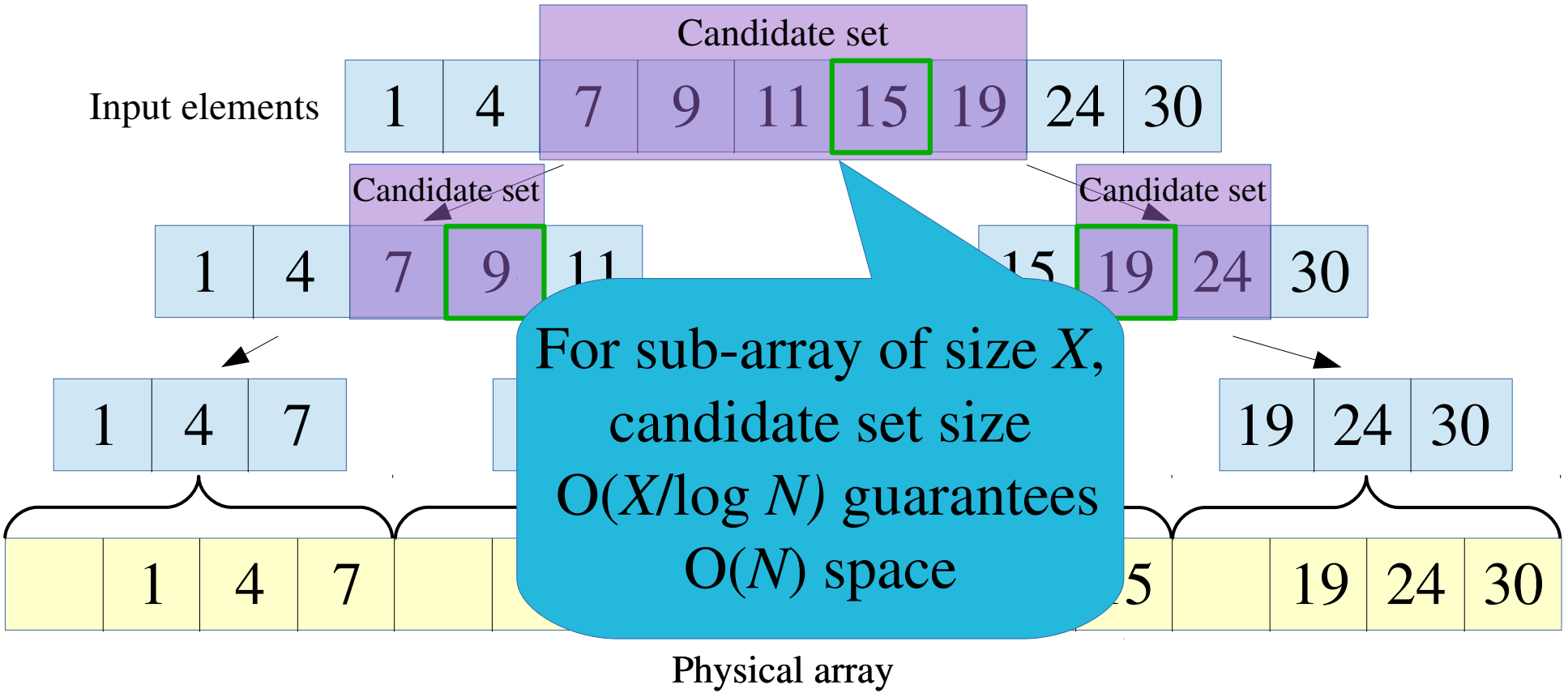
Prob[leader changes] $\approx 1/n$



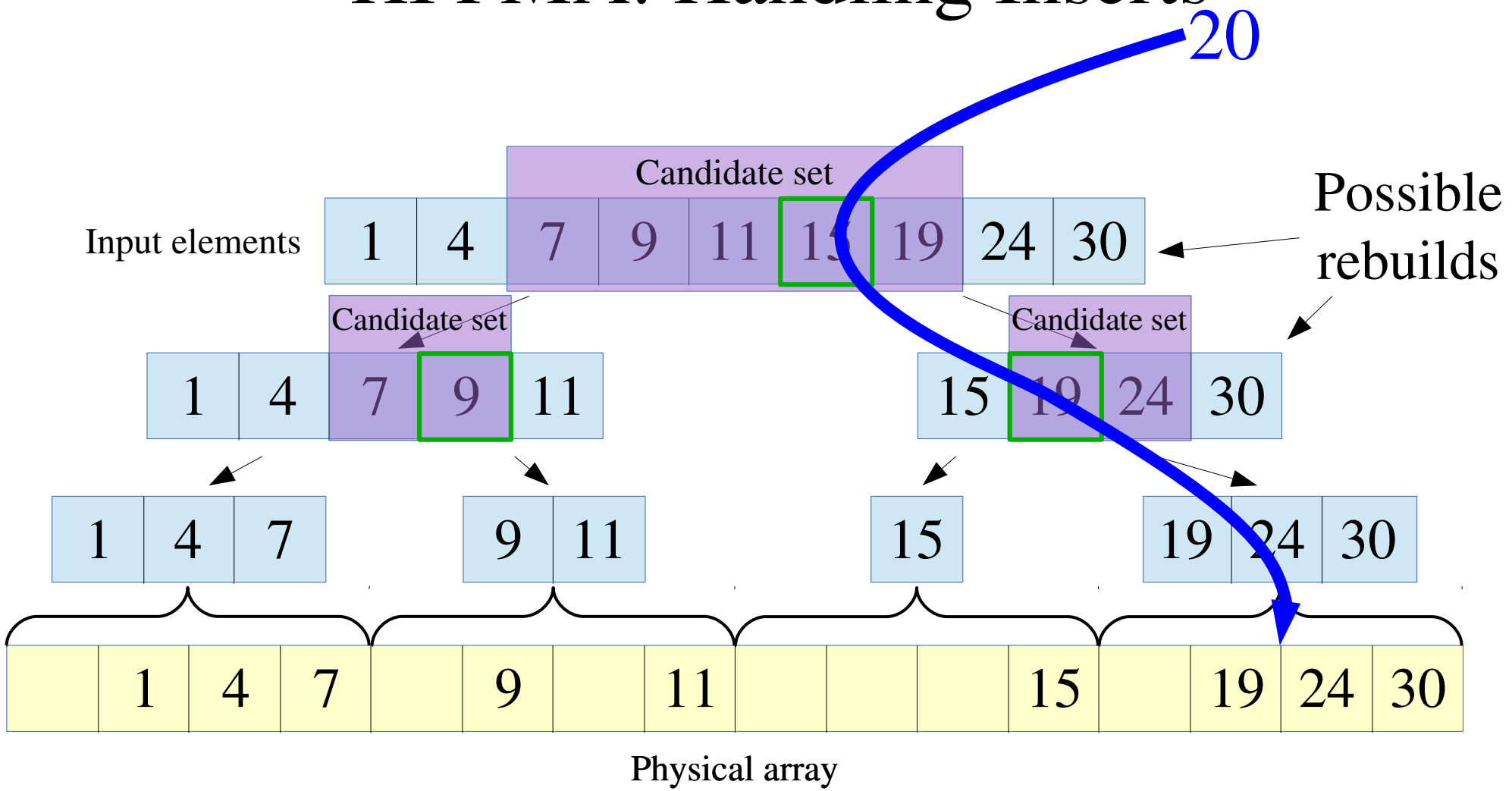
HI PMA: Building from Scratch

Idea: Use reservoir sampling to select where to split elements

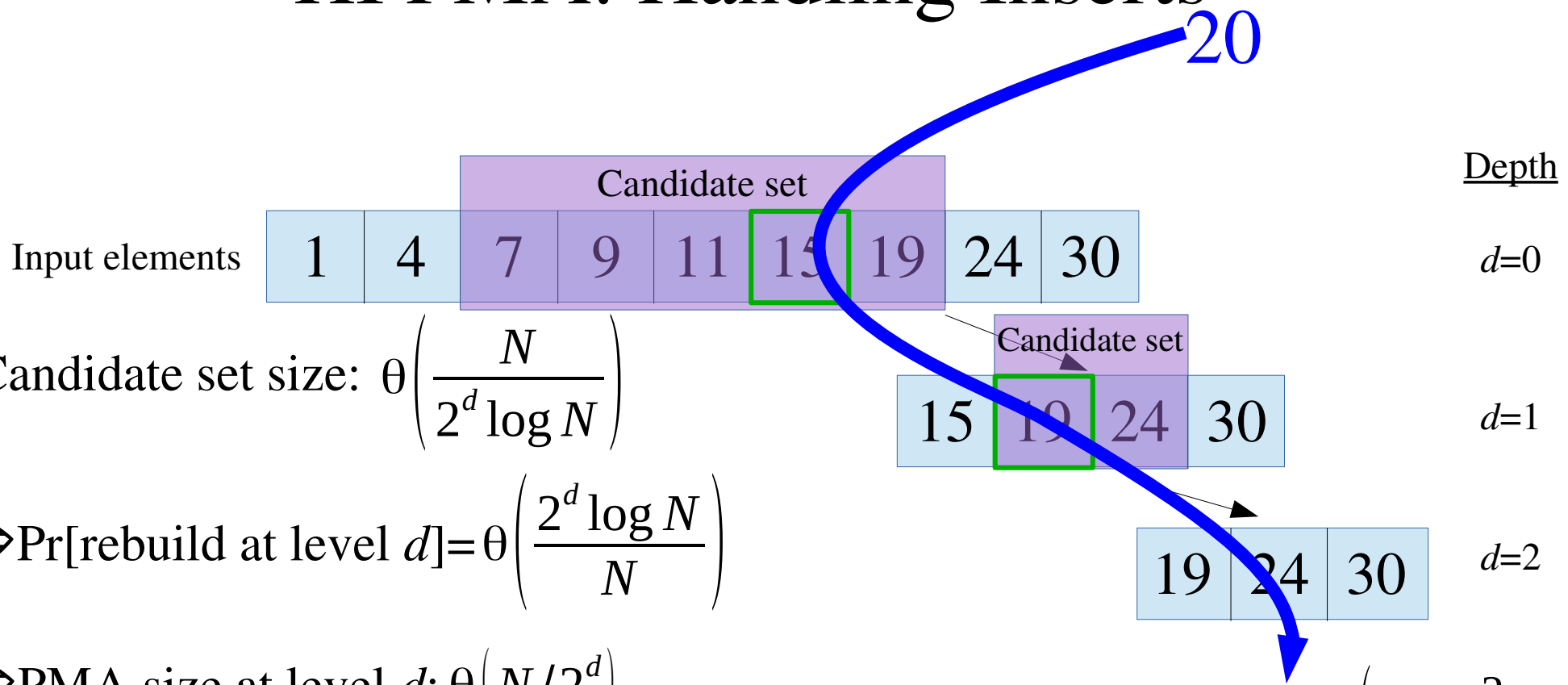
Rebuilding a PMA of N elements takes $O(N)$ work among physical slots



HI PMA: Handling Inserts



HI PMA: Handling Inserts



Candidate set size: $\theta\left(\frac{N}{2^d \log N}\right)$

$\Rightarrow$Pr[rebuild at level d]$=\theta\left(\frac{2^d \log N}{N}\right)$

$\Rightarrow$PMA size at level d: $\theta\left(N/2^d\right)$

$\Rightarrow$E[work for rebuild at level d]$=\theta\left(\log N\right)$

E[insert cost]$=\theta\left(\log^2 N\right)$

Conclusions

- We can have history independence and I/O efficiency
  - HI packed memory array
  - HI cache-oblivious B-tree
  - HI skip list
- Same amortized complexities as non-HI versions w.h.p.
- Opportunity: applications
  - Secure delete in file systems
  - Privacy-preserving documents
  - Mobile devices
  - Correctness testing
  - Concurrency