Run Generation Revisited: What Goes Up May or May Not Come Down

Shikha Singh

Joint Work with :

Michael A. Bender, Samuel McCauley, Andrew McGregor, and Hoa T. Vu

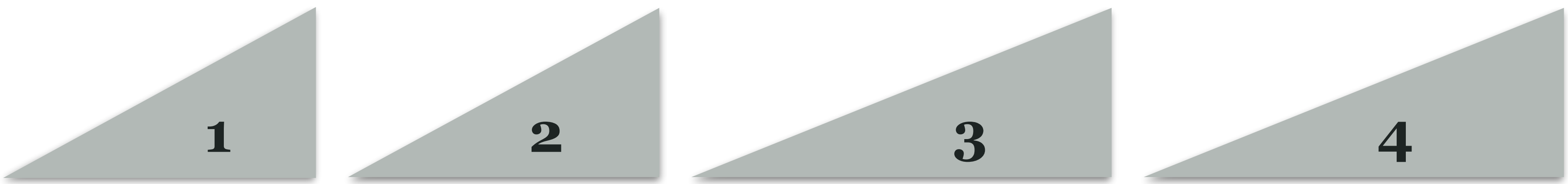
Stony Brook University

UMASS AMHERST

Run Generation Revisited: What Goes Up May or May Not Come Down

- Contiguous sequence of sorted elements in an array

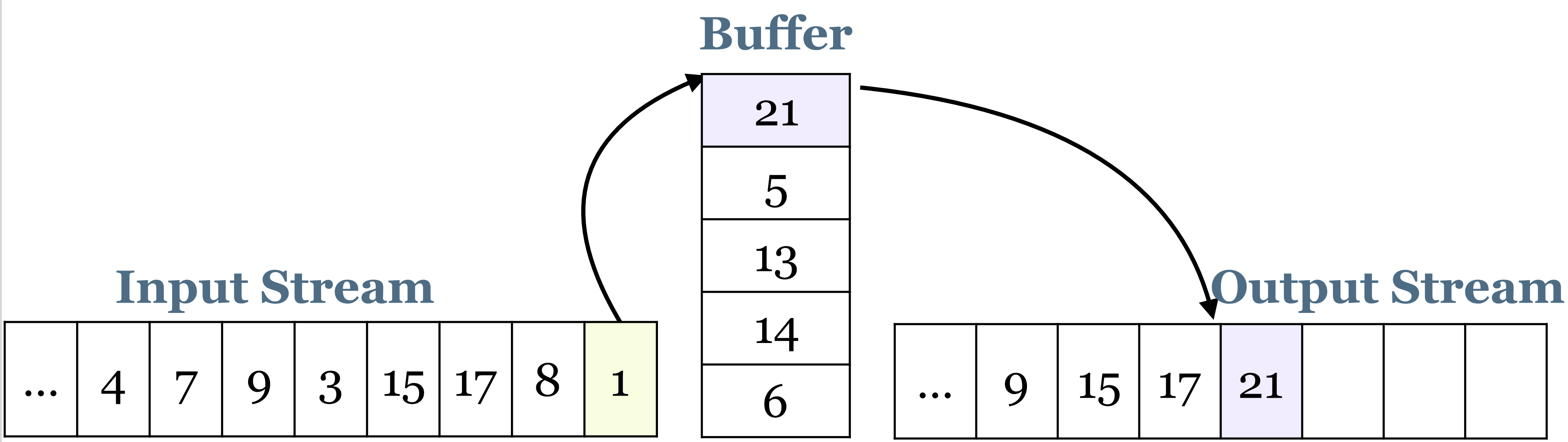
5	9	11	2	4	7	6	13	25	30	3	5	7	11



- **Number of runs**:
  - Smallest number of runs that partition the array

Run Generation Revisited: What Goes Up May or May Not Come Down

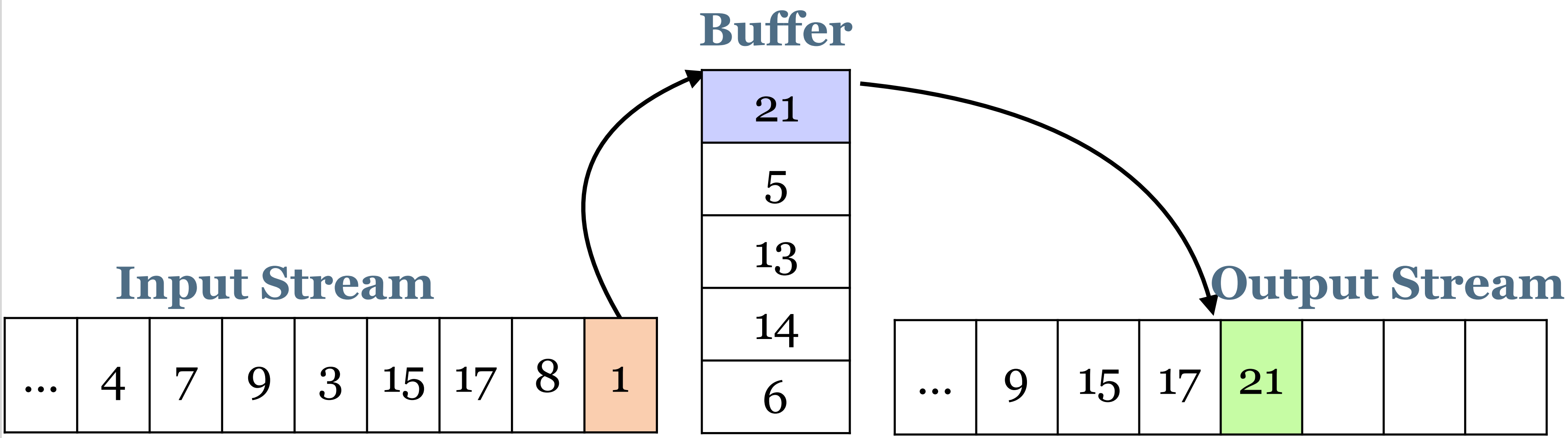
Reorder elements arriving from a (large) input stream using a (small) buffer to produce *long* runs



Run Generation Revisited: What Goes Up May or May Not Come Down

- Scan input ingesting elements into buffer
- Reorder using buffer and write to output stream

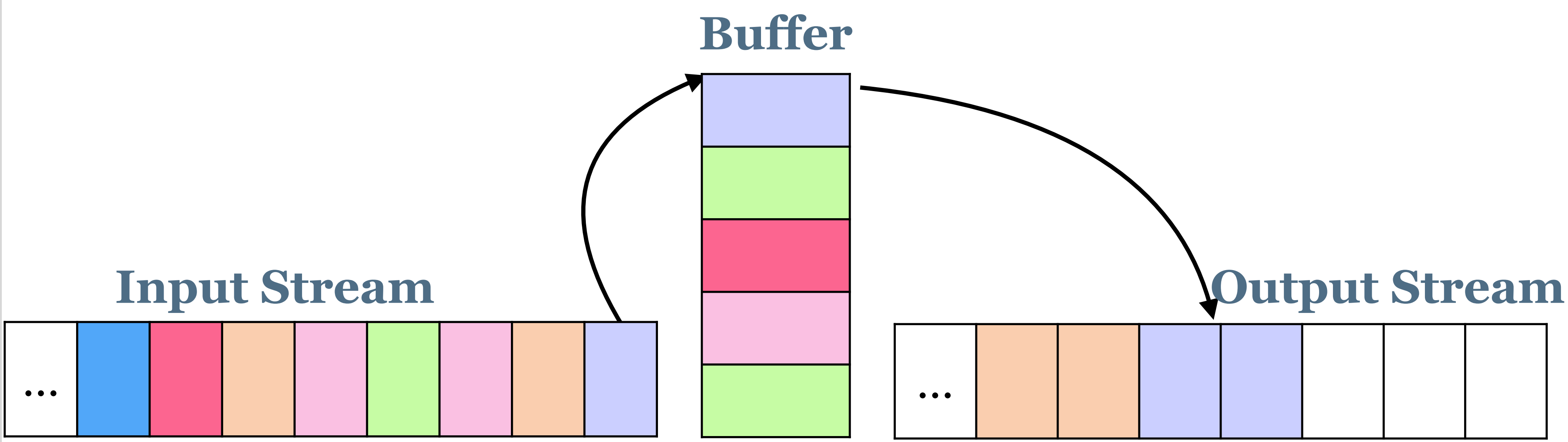
Objective: Devise a scheduling strategy for the order in which elements must be output so as to minimize the **number of runs**



Related: Reordering Buffer Management

- Ingest input elements, each of a certain *color*, into buffer
- Reorder using buffer and output element

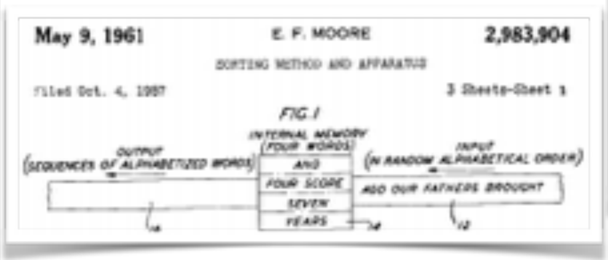
Objective: Devise a scheduling strategy for the order in which elements must be output so as to minimize the **number of color changes**



Related: Reordering Buffer Management

- Well-known scheduling problem
- Extensively studied (both online and offline case)
  - [Racke, Sohler and Westermann 2002]
  - [Asahiro, Kawahara and Miyano 2012]
  - [Avigdor-Elgrabli and Rabani 2013]
  - [Bar-Yehuda and Laserson 2007]
  - [Chan, Megow, Sitters and van Stee 2012]
  - [Englert and Westermann 2005]
  - [Im and Moseley 2013, 2015]
  - [Avigdor-Elgrabli, Im, Moseley and Rabani 2015]

Run Generation Revisited: What Goes Up May or May Not Come Down



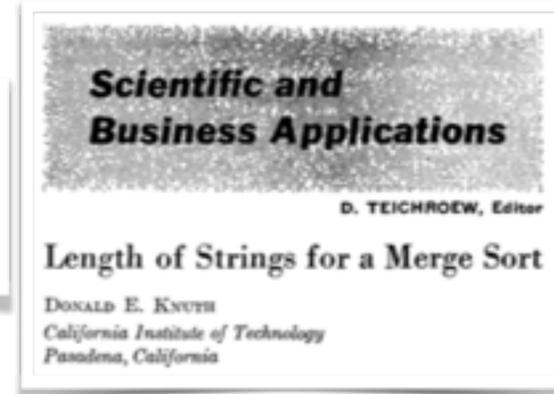
1961


Internal and Tape Sorting Using the Replacement-Selection Technique*

Martin A. Goetz

Applied Data Research, Inc., Princeton, N. J.

1963


Scientific and Business Applications

D. TEICHROEW, Editor

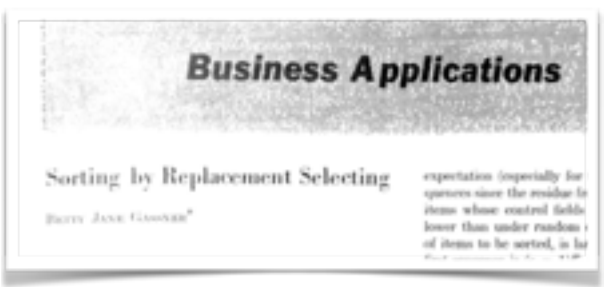
Length of Strings for a Merge Sort

Donald E. Knuth

California Institute of Technology

Pasadena, California

1963


Business Applications

Sorting by Replacement Selecting

1967


Sorting by Natural Selection

W.D. Frazer

and

C.K. Wong

IBM Thomas J. Watson Research Center

1972


Perfectly Overlapped Generation of Long Runs for Sorting Large Files*

Yen-Chun Lin

1973

- Studied in the context of *External Memory Merge Sort*

Run Generation Revisited: What Goes Up May or May Not Come Down

FAST GENERATION OF LONG SORTED RUNS FOR SORTING A LARGE FILE

Yen-Chun Lin and Yu-Ho Cheng
Dept. of Electronic Engineering
National Taiwan Institute of Technology
Taipei, Taiwan, R.O.C.

1991

Speeding up External Mergesort

LuoQuan Zheng and Per-Åke Larson *

1996

Perfectly overlapped generation of long runs on a transputer array for sorting

Yen-Chun Lin*, Horng-Yi Lai

Department of Electronic Engineering, National Taiwan Institute of Technology, P.O. Box 90-100, Taipei 106, Taiwan

Received 18 March 1996; revised 20 November 1996; accepted 9 December 1996

1997

Memory Management during Run Generation in External Sorting

Per-Åke Larson
Microsoft
PALarson@microsoft.com

Goetz Graefe
Microsoft
GoetzG@microsoft.com

1998

- Continued work to improve run length (to speed up merge)

Run Generation Revisited: What Goes Up May or May Not Come Down

IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING, VOL. 15, NO. 4, JULY/AUGUST 2003

External Sorting: Run Formation Revisited

Per-Åke Larson, *Member, IEEE Computer Society*

2003

Implementing Sorting in Database Systems

GOETZ GRAEFE

Microsoft

2006

Two-way Replacement Selection

Xavier Martinez-Palau, David Dominguez-Sal, Josep Lluis Larriba-Pey
DAMA-UPC, Departament d'Arquitectura de Computadors
Universitat Politècnica de Catalunya
Campus Nord-UPC, 08034 Barcelona
{xmartine,ddomings,larri}@ac.upc.edu

2010

External Sorting on Flash Memory Via Natural Page Run Generation

YANG LIU, ZHEN HE, YI-PING PHOEBE CHEN AND THI NGUYEN

Department of Computer Science and Computer Engineering, La Trobe University, VIC 3086, Australia

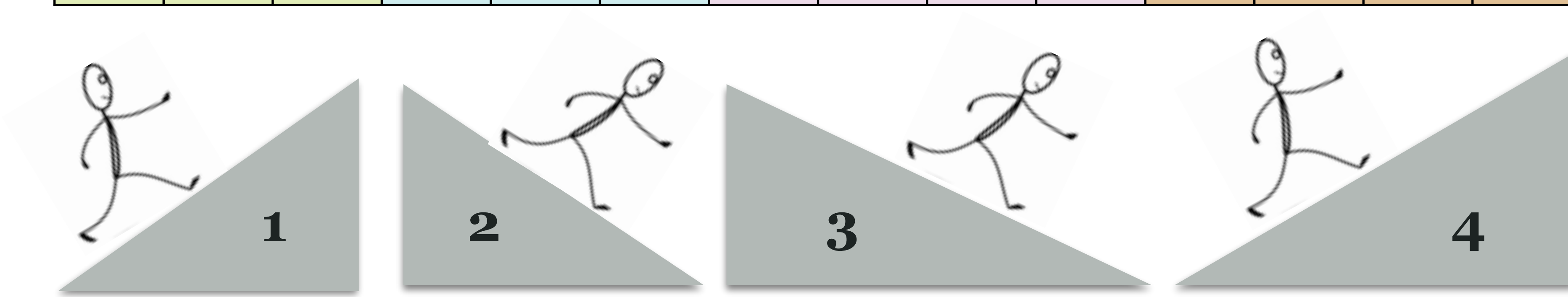
2011

- Classic Problem: Studied for over 5 decades!

Run Generation Revisited:
What Goes Up May or May Not Come Down

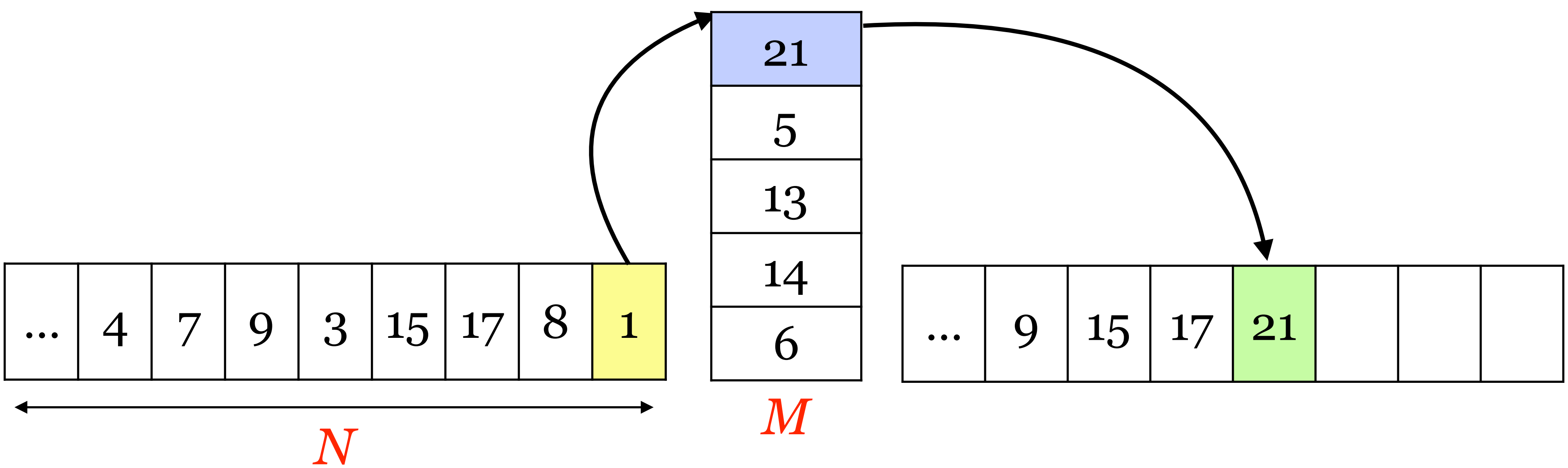
- Up Runs are monotonically increasing (sorted)
- Down Runs are monotonically decreasing (reverse sorted)

5	9	11	7	4	2	30	25	13	6	8	12	17	21



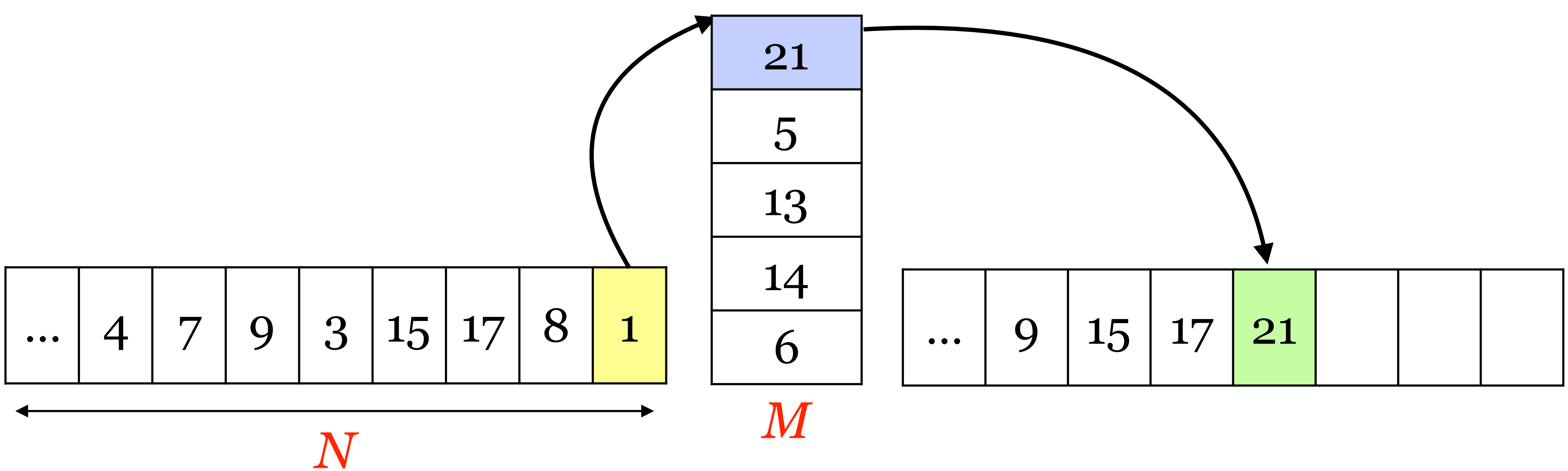
Run Generation: Problem Definition

- Input: Stream of *N* elements
- Can be stored temporarily in a buffer of size *M* < *N*
- Buffer gets full -> *write* an element to output stream
- Next element is *read* into the slot freed
- Buffer is always full (except when < *M* elements remain)



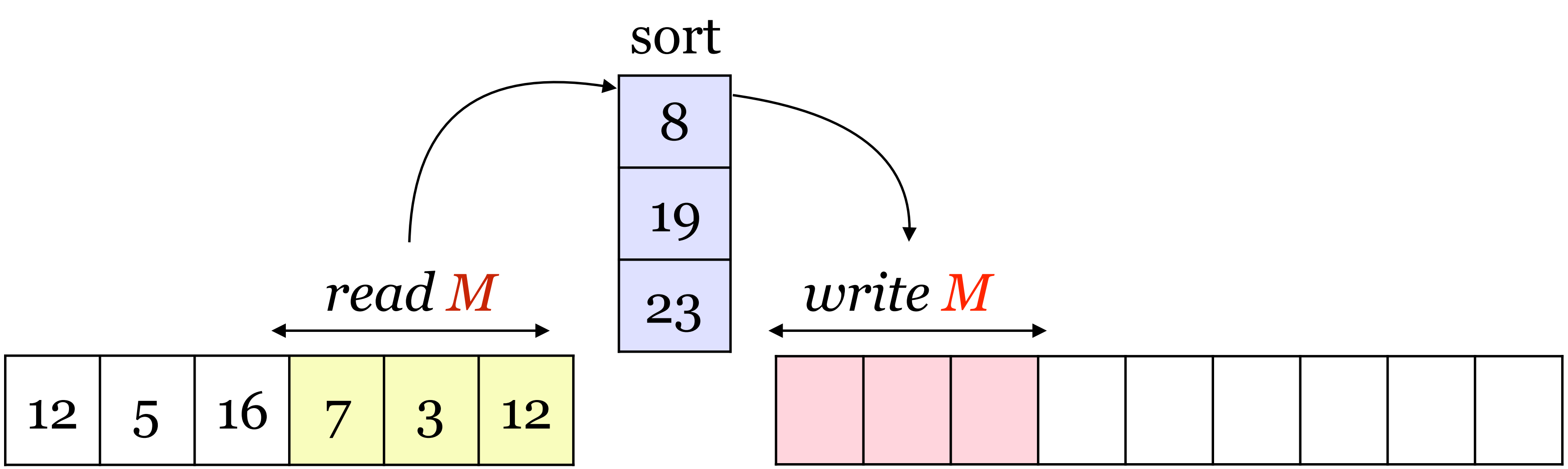
Run Generation: Problem Definition

- Schedule dictates what to eject based on
  - Contents of buffer, last element written
- Cannot arbitrarily access input or output
  - Read next-in-order from input, append to output



Naive Run Generation

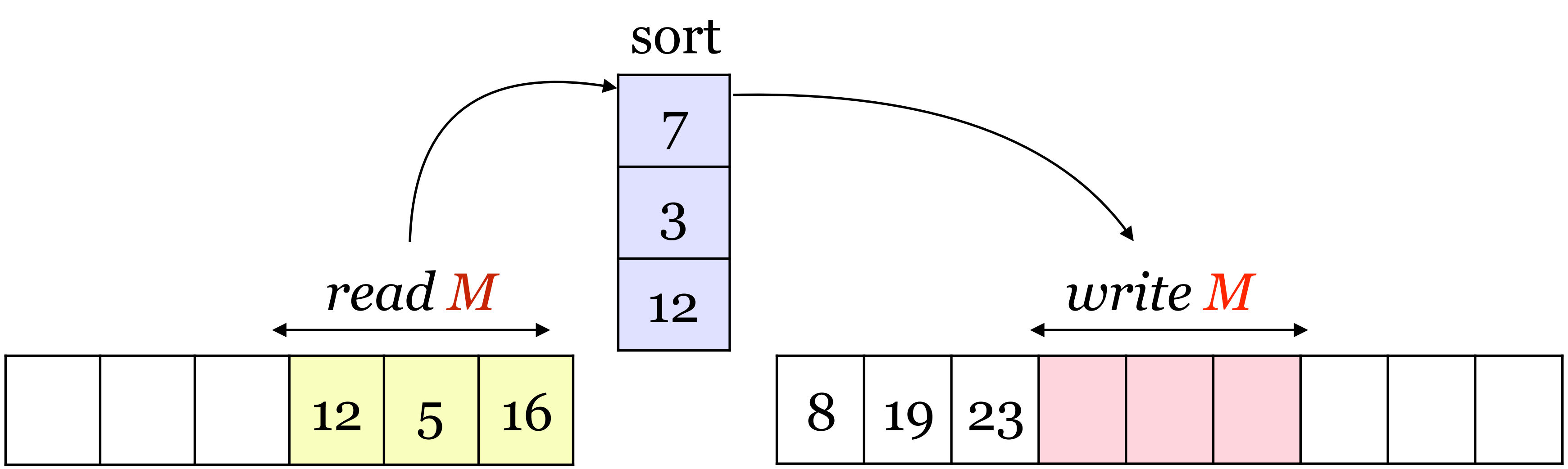
- Load M elements to the buffer
- Sort these M elements
- Output them in sorted order



Runs of length M

Naive Run Generation

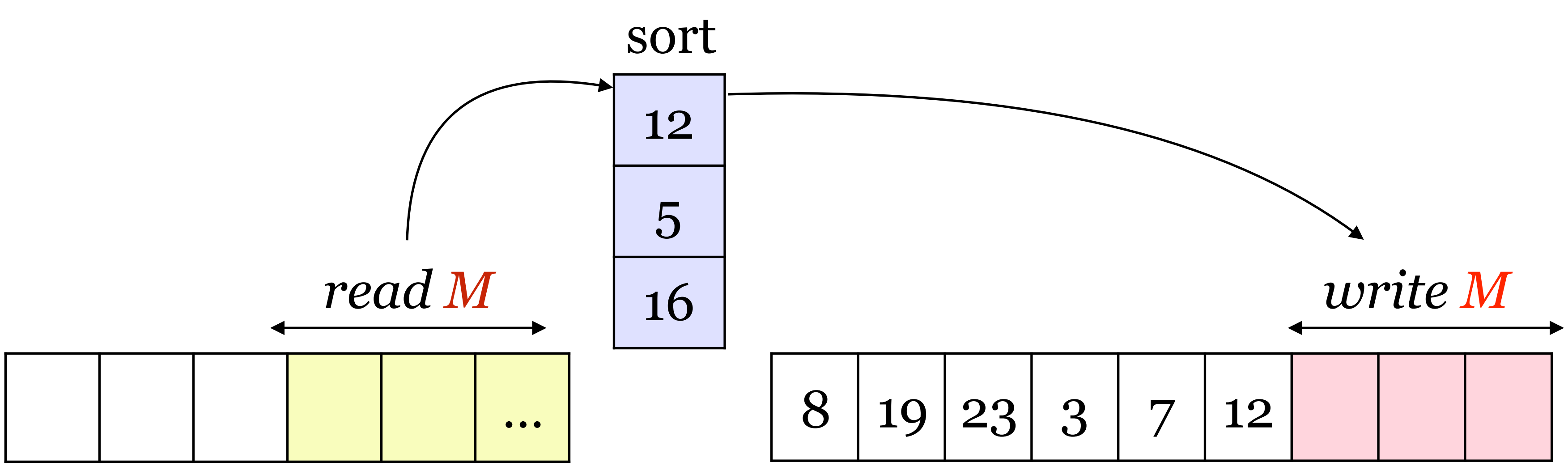
- Load *M* elements to the buffer
- Sort these *M* elements
- Output them in sorted order



Runs of length *M*

Naive Run Generation

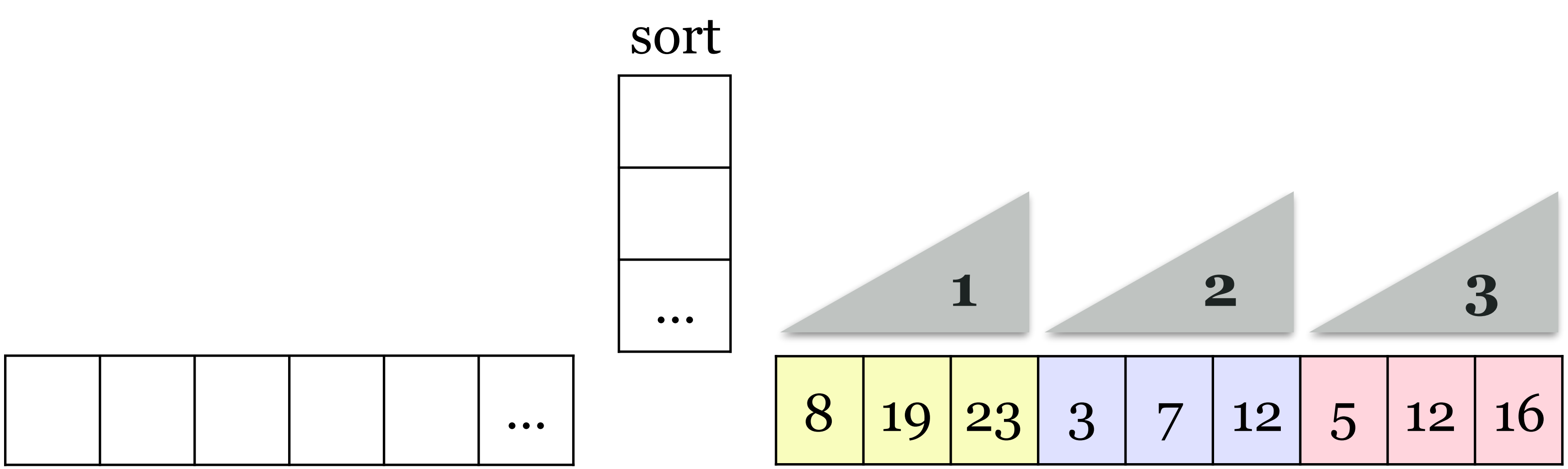
- Load *M* elements to the buffer
- Sort these *M* elements
- Output them in sorted order



Runs of length *M*

Naive Run Generation

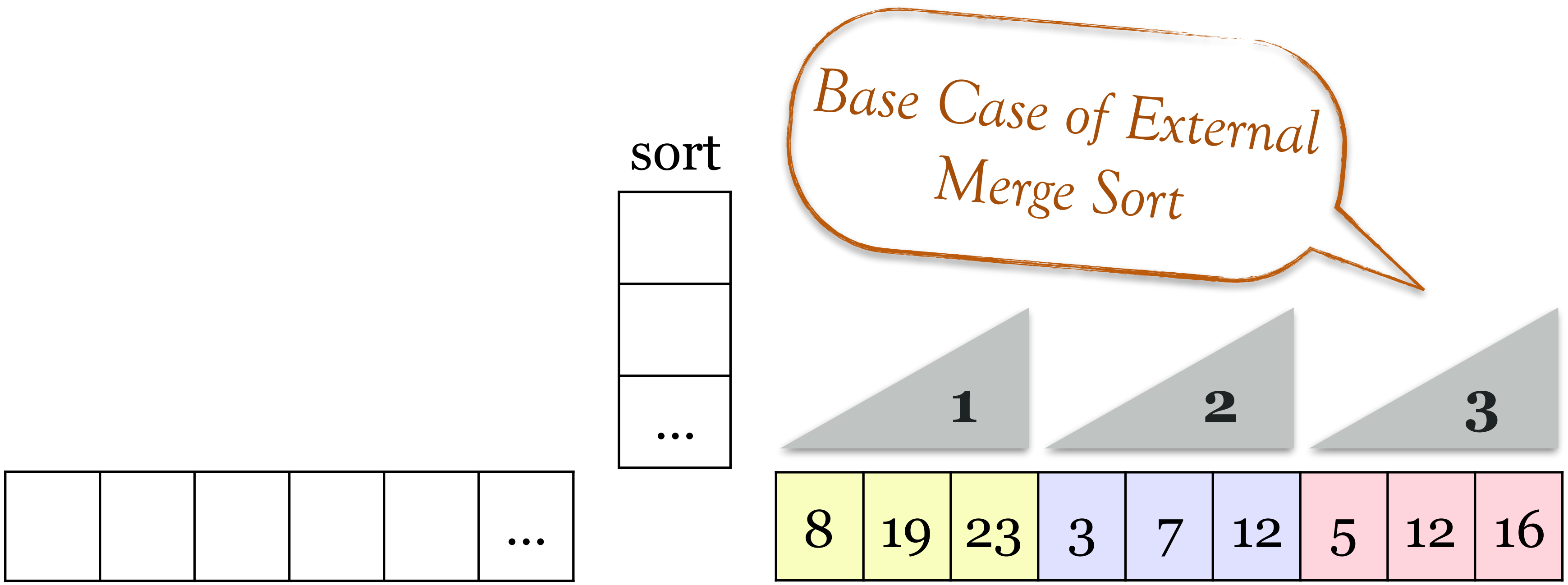
- Load M elements to the buffer
- Sort these M elements
- Output them in sorted order



Runs of length M

Naive Run Generation

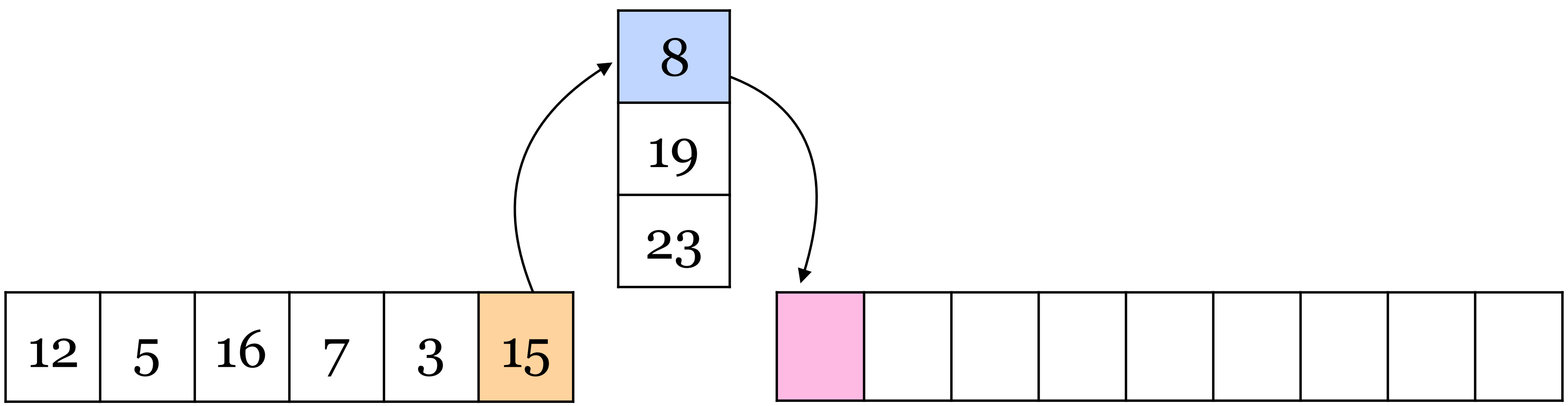
- Load M elements to the buffer
- Sort these M elements
- Output them in sorted order



Runs of length M

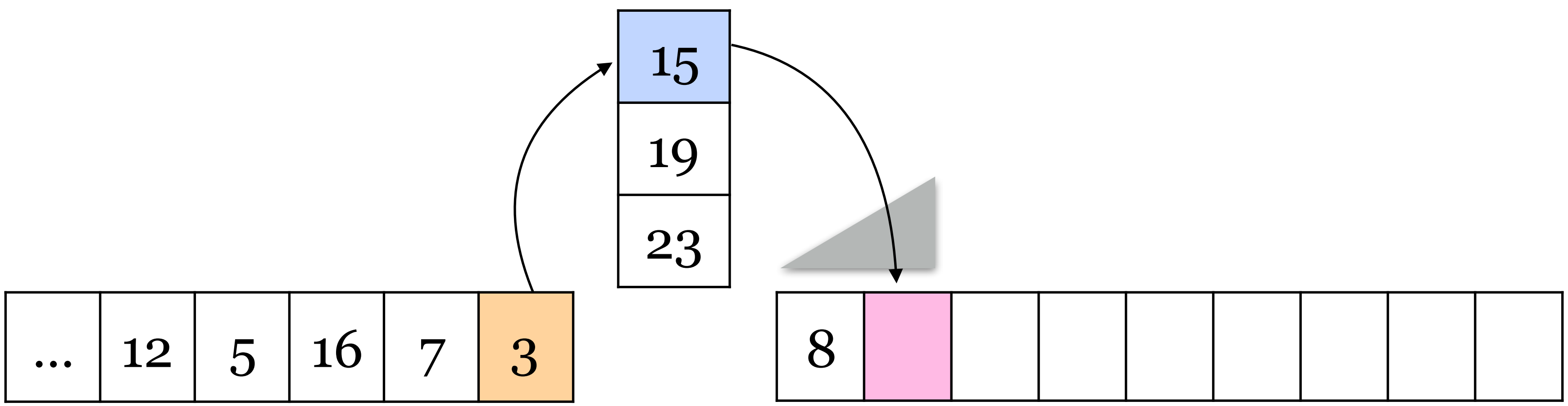
Classic Schedule: All Up Runs

- **Replacement Selection** [Goetz 63]:
  - Starting from a full buffer, output smallest element
  - Output smallest element in buffer $\geq$ the last output
  - If no such element, start a new run and continue



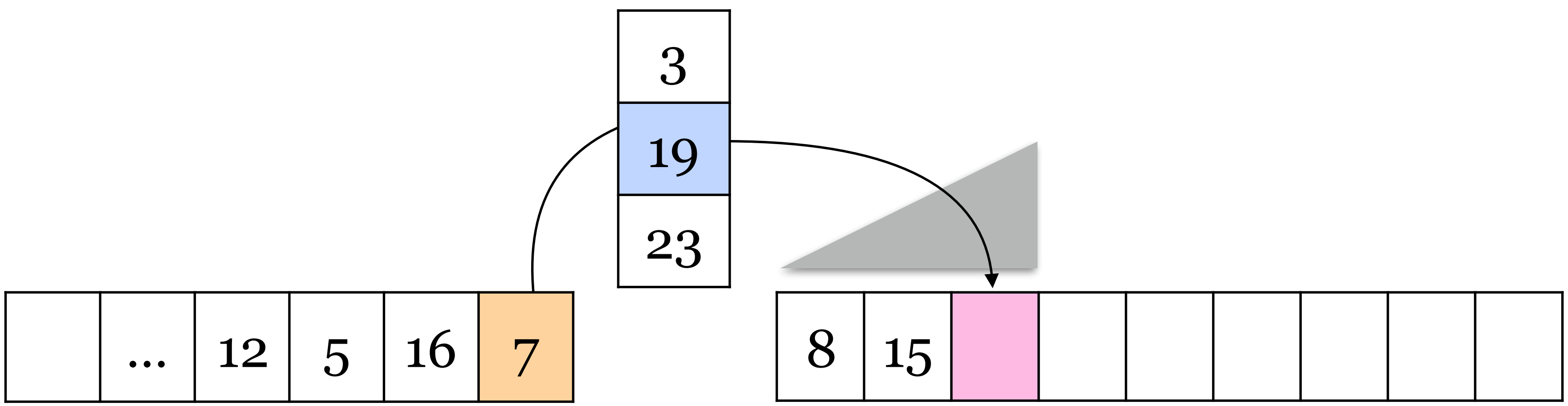
Classic Schedule: All Up Runs

- **Replacement Selection** [Goetz 63]:
  - Starting from a full buffer, output smallest element
  - Output smallest element in buffer $\geq$ the last output
  - If no such element, start a new run and continue



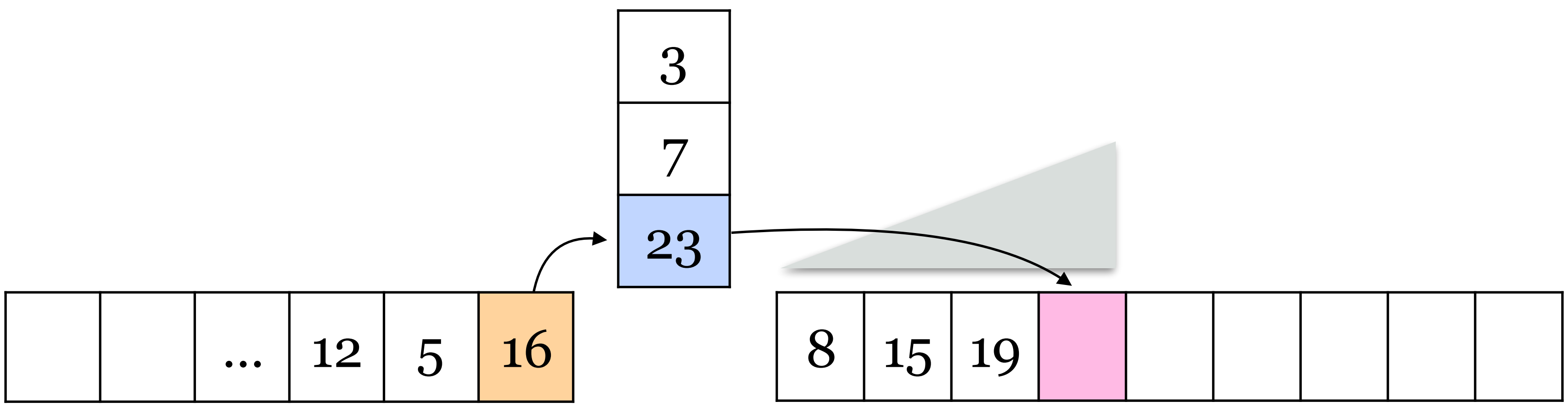
Classic Schedule: All Up Runs

- **Replacement Selection** [Goetz 63]:
  - Starting from a full buffer, output smallest element
  - Output smallest element in buffer $\geq$ the last output
  - If no such element, start a new run and continue



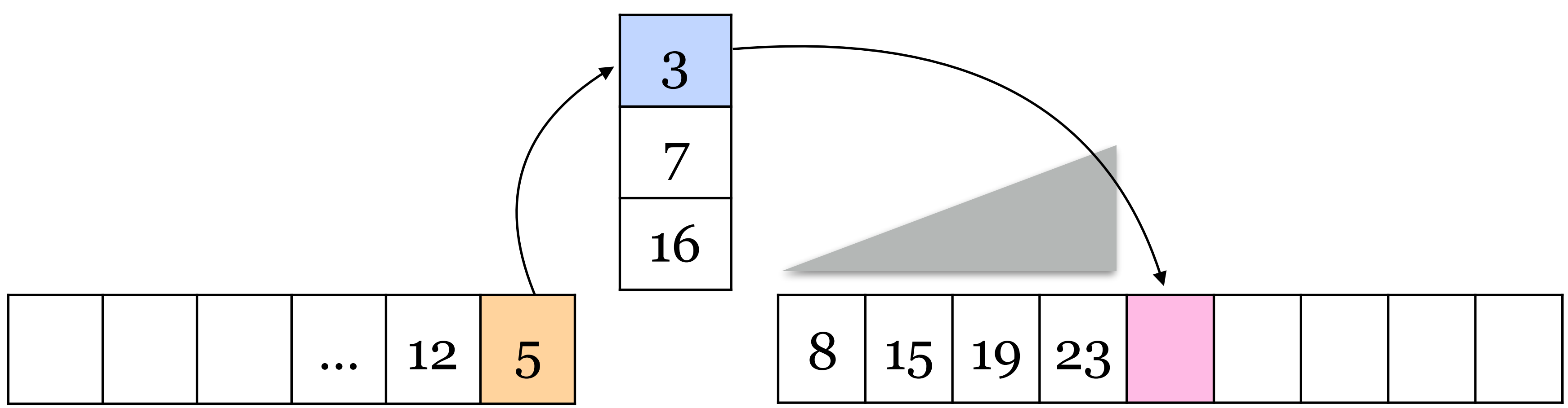
Classic Schedule: All Up Runs

- **Replacement Selection** [Goetz 63]:
  - Starting from a full buffer, output smallest element
  - Output smallest element in buffer $\geq$ the last output
  - If no such element, start a new run and continue



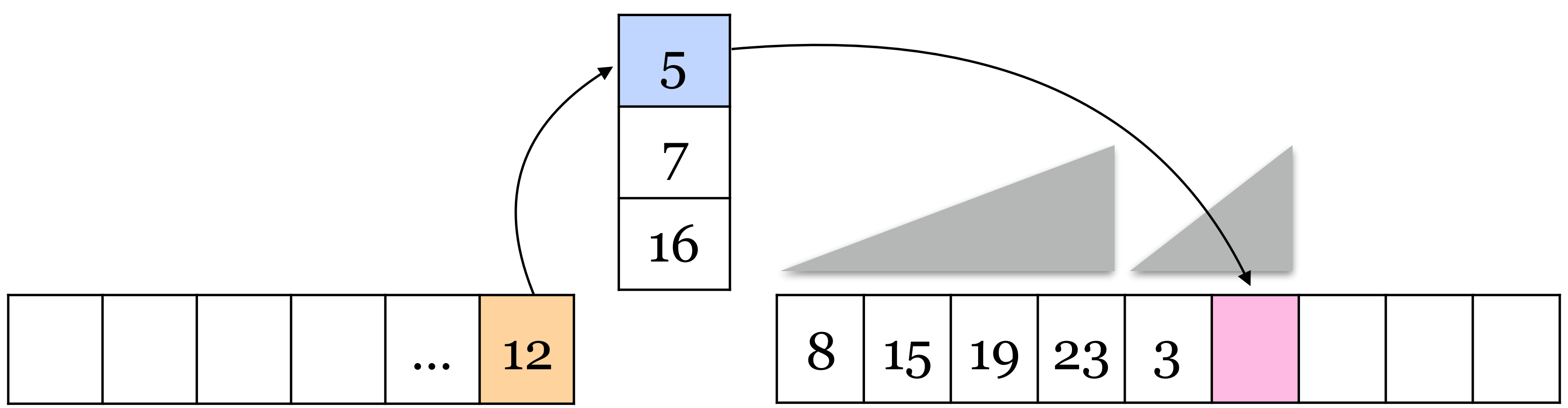
Classic Schedule: All Up Runs

- **Replacement Selection** [Goetz 63]:
  - Starting from a full buffer, output smallest element
  - Output smallest element in buffer $\geq$ the last output
  - If no such element, start a new run and continue



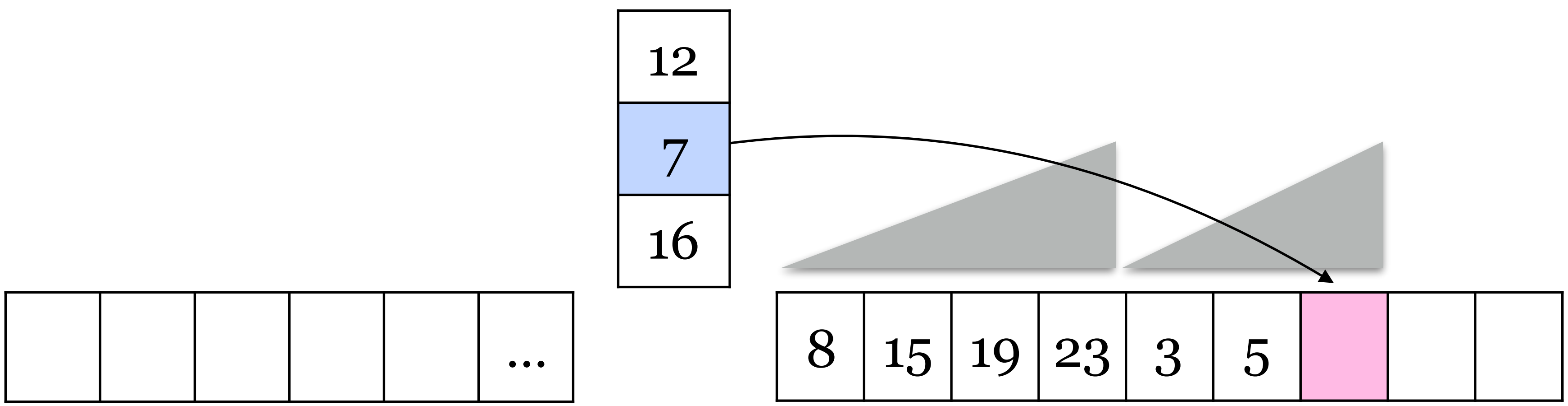
Classic Schedule: All Up Runs

- **Replacement Selection** [Goetz 63]:
  - Starting from a full buffer, output smallest element
  - Output smallest element in buffer $\geq$ the last output
  - If no such element, start a new run and continue



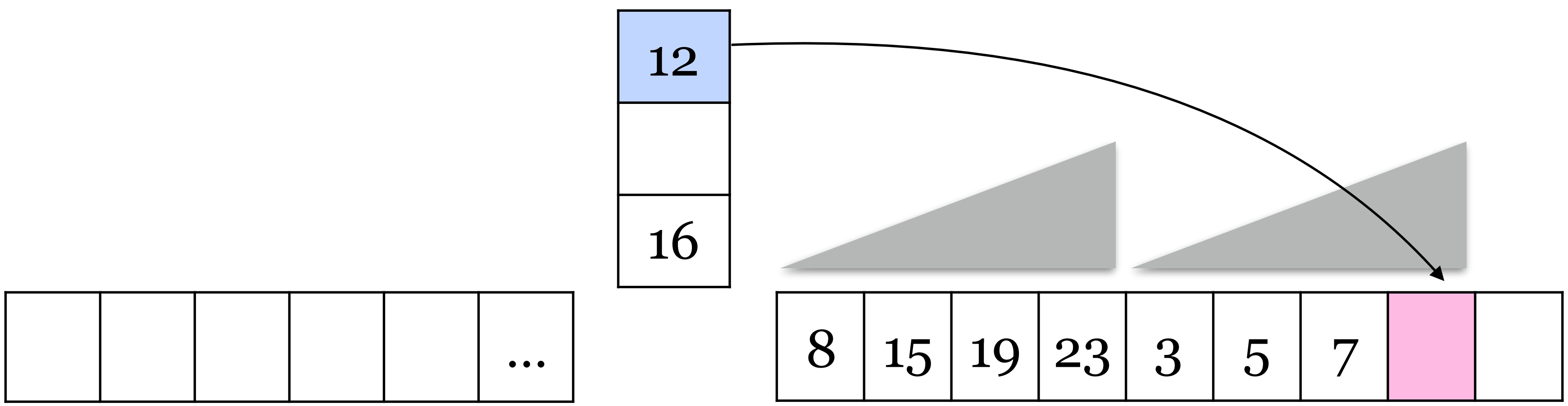
Classic Schedule: All Up Runs

- **Replacement Selection** [Goetz 63]:
  - Starting from a full buffer, output smallest element
  - Output smallest element in buffer $\geq$ the last output
  - If no such element, start a new run and continue



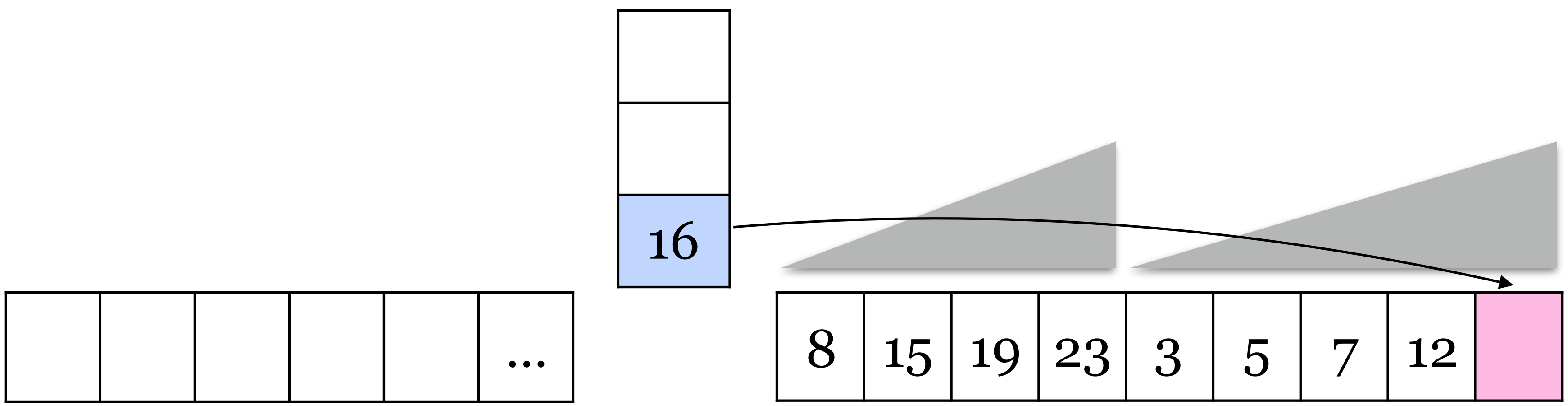
Classic Schedule: All Up Runs

- **Replacement Selection** [Goetz 63]:
  - Starting from a full buffer, output smallest element
  - Output smallest element in buffer $\geq$ the last output
  - If no such element, start a new run and continue



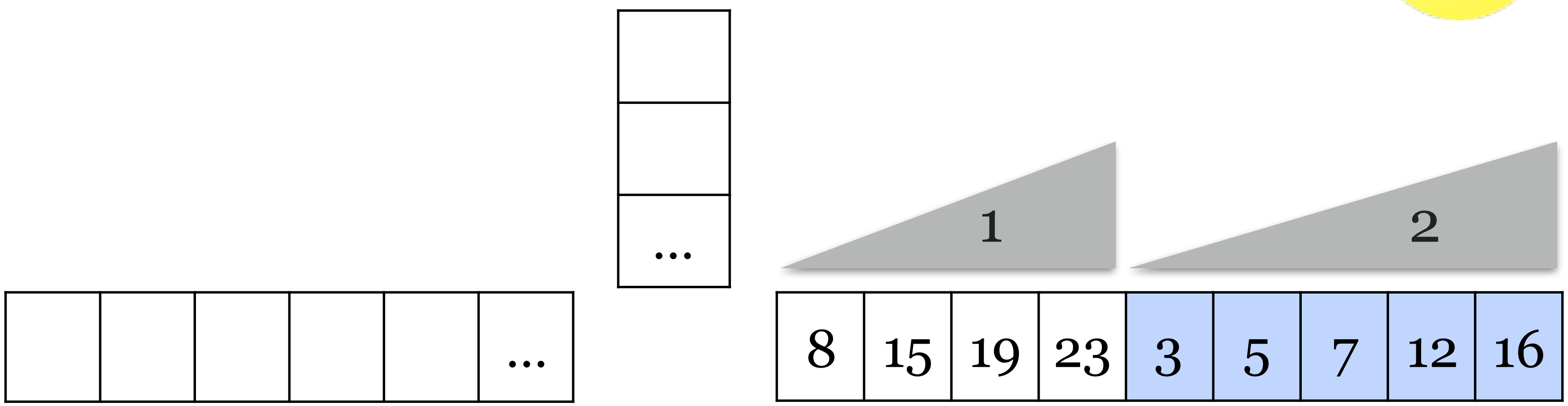
Classic Schedule: All Up Runs

- **Replacement Selection** [Goetz 63]:
  - Starting from a full buffer, output smallest element
  - Output smallest element in buffer $\geq$ the last output
  - If no such element, start a new run and continue



Classic Schedule: All Up Runs

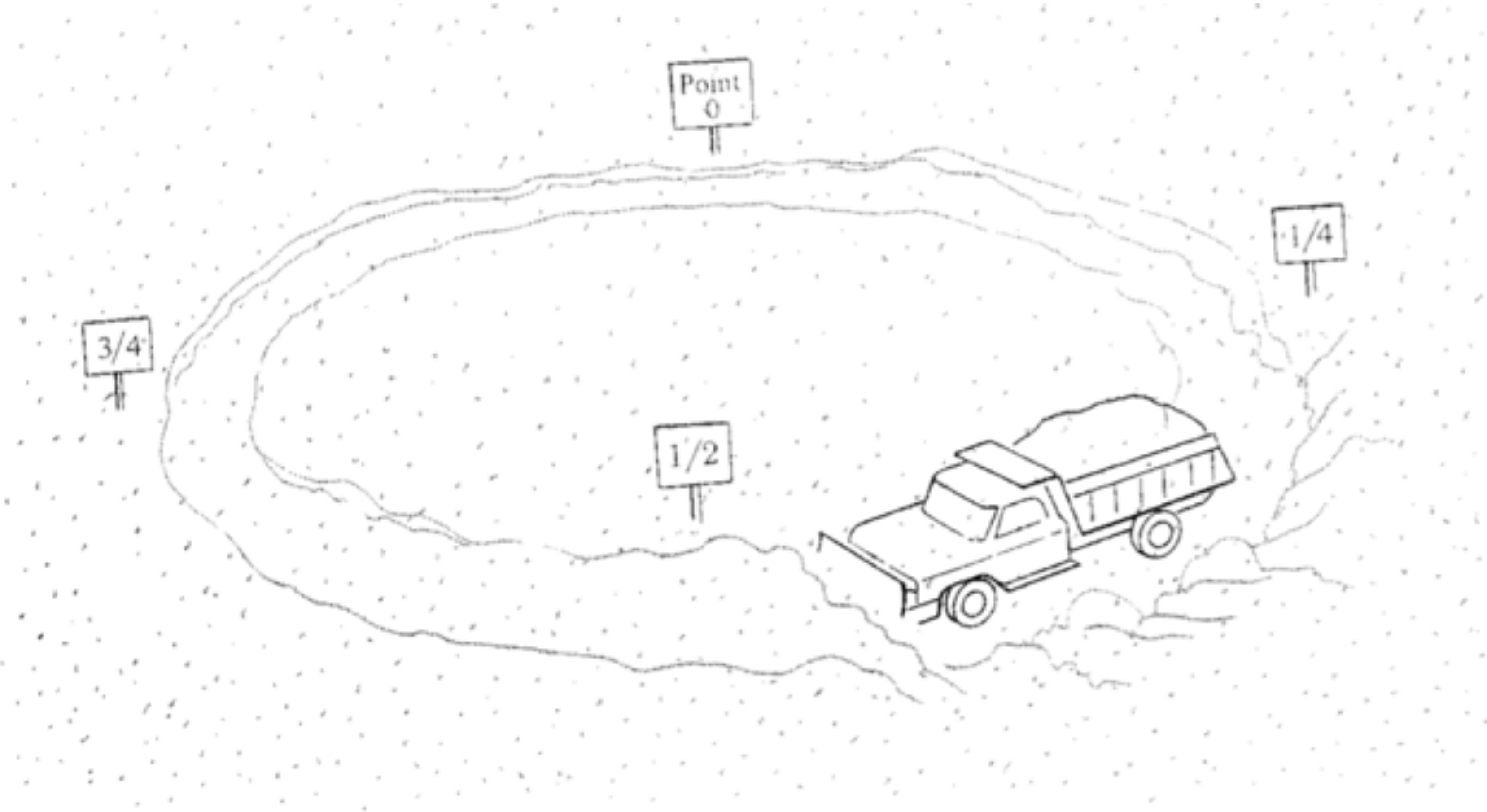
- **Replacement Selection** [Goetz 63]:
  - Starting from a full buffer, output smallest element
  - Output smallest element in buffer $\geq$ the last output
  - If no such element, start a new run and continue



Runs of length $> M$

Performance of Replacement Selection

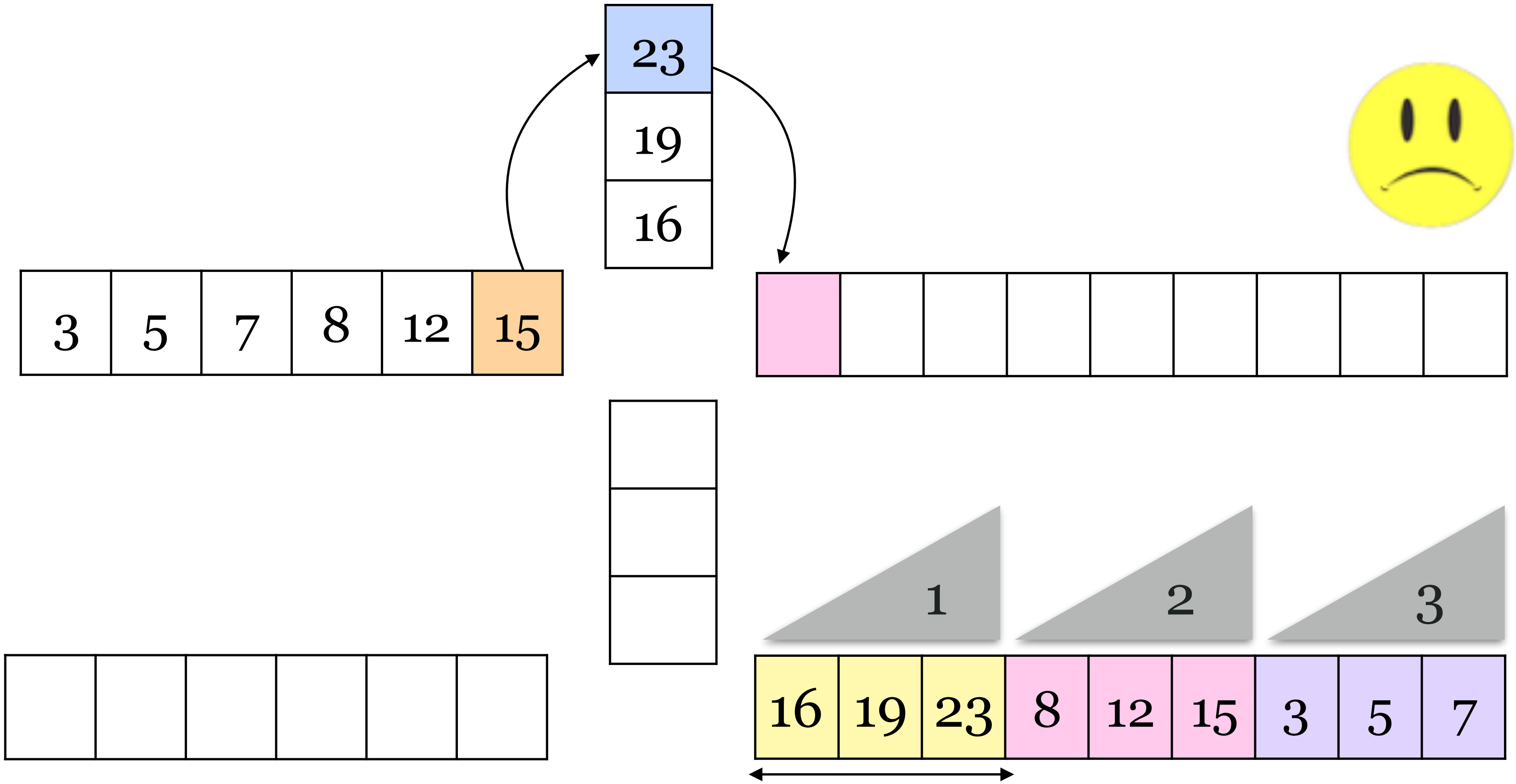
- On random data, expected length of a run is 2M



"The perpetual plow on its ceaseless cycle." - Knuth '98

Performance of Replacement Selection

- However, on inversely sorted input

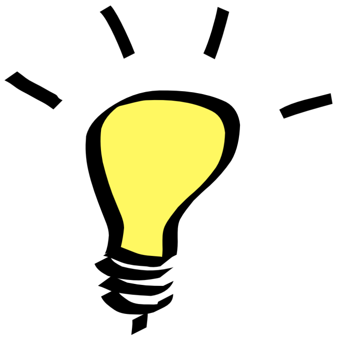


Runs of length *M* on reverse sorted input

Performance of Replacement Selection

- If the input stream is mostly increasing
  - Up runs are great

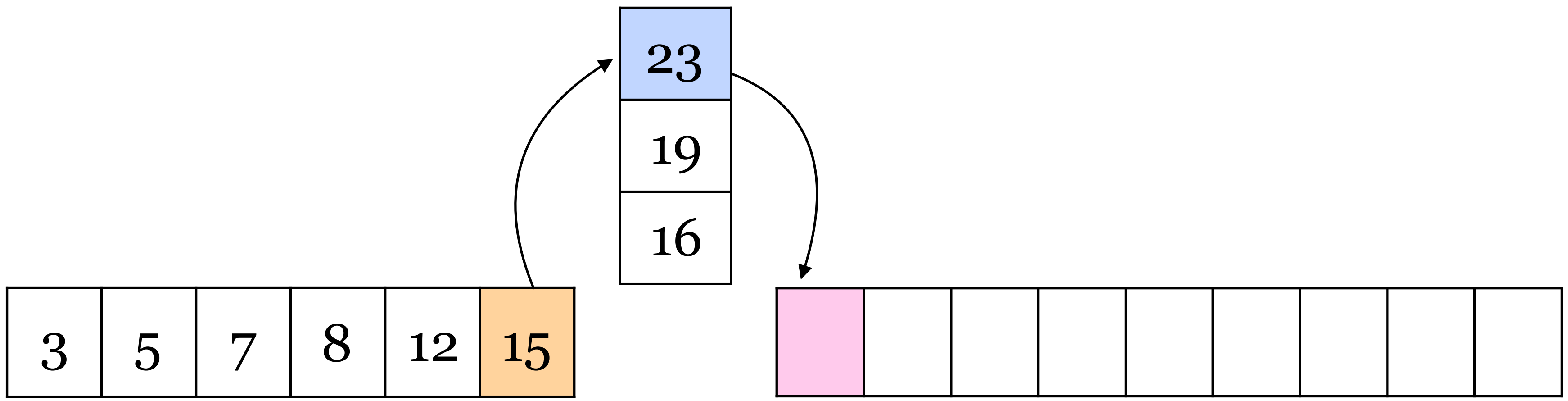
- If the input stream is mostly decreasing
  - Up runs don’t help



From the point of view of sorting (merging), the direction of runs (up or down) doesn't really matter.

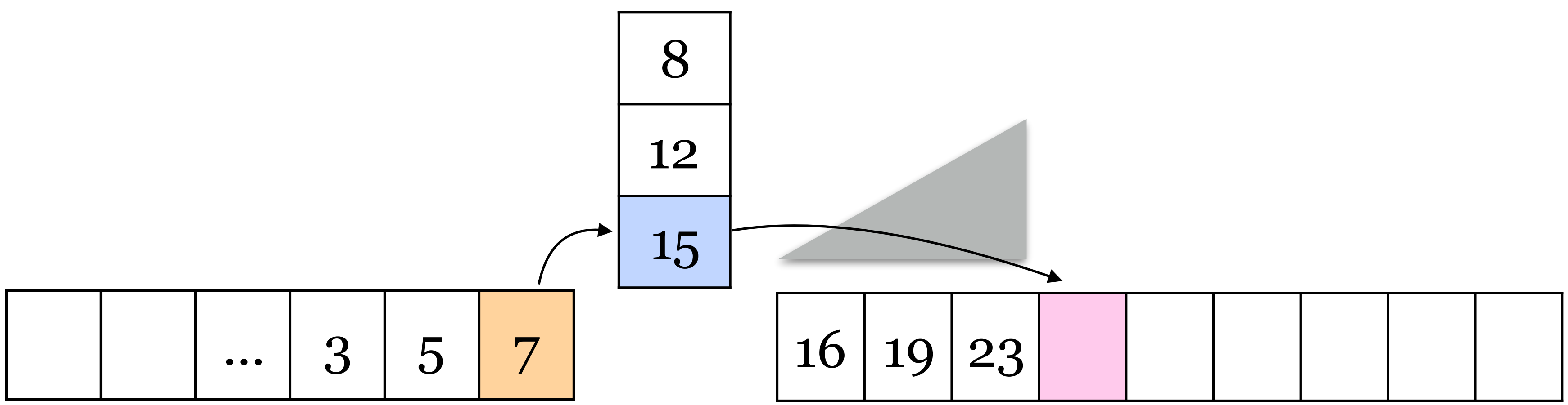
Alternating-Up-Down Schedule

- Deterministically alternate between up and down runs



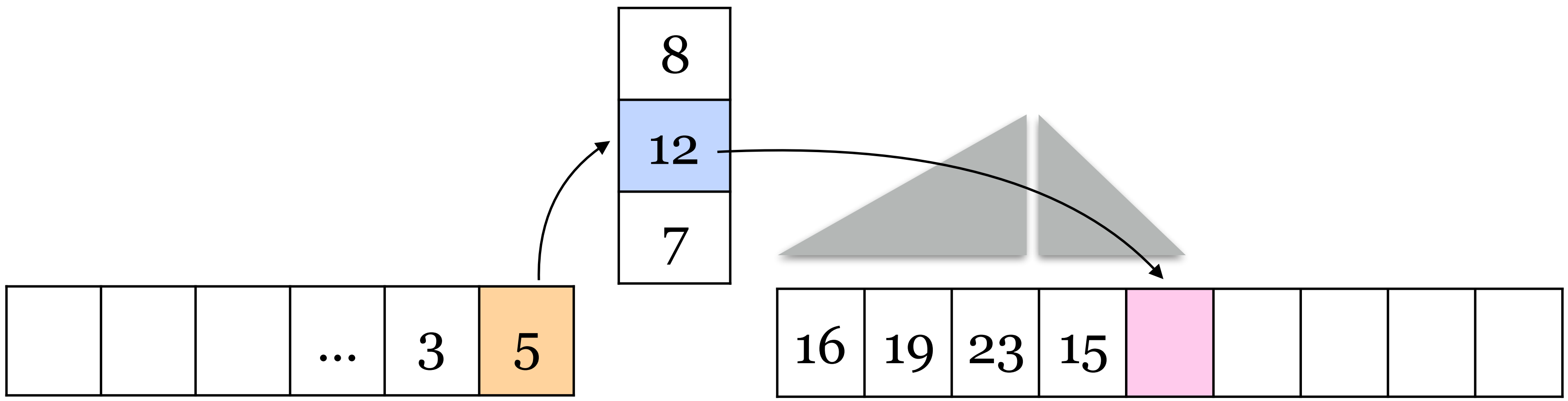
Alternating-Up-Down Schedule

- Deterministically alternate between up and down runs



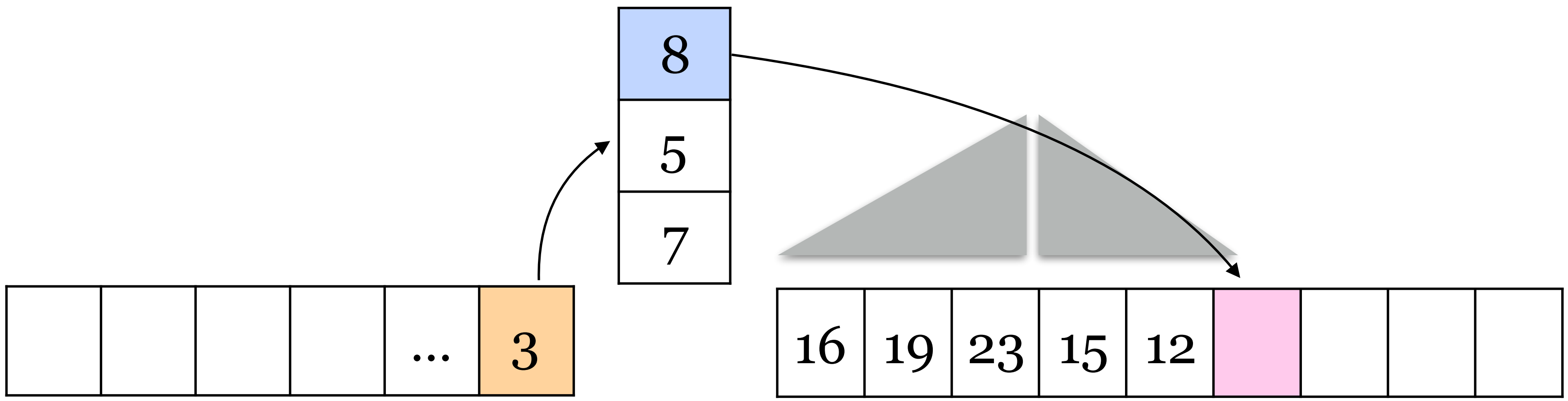
Alternating-Up-Down Schedule

- Deterministically alternate between up and down runs



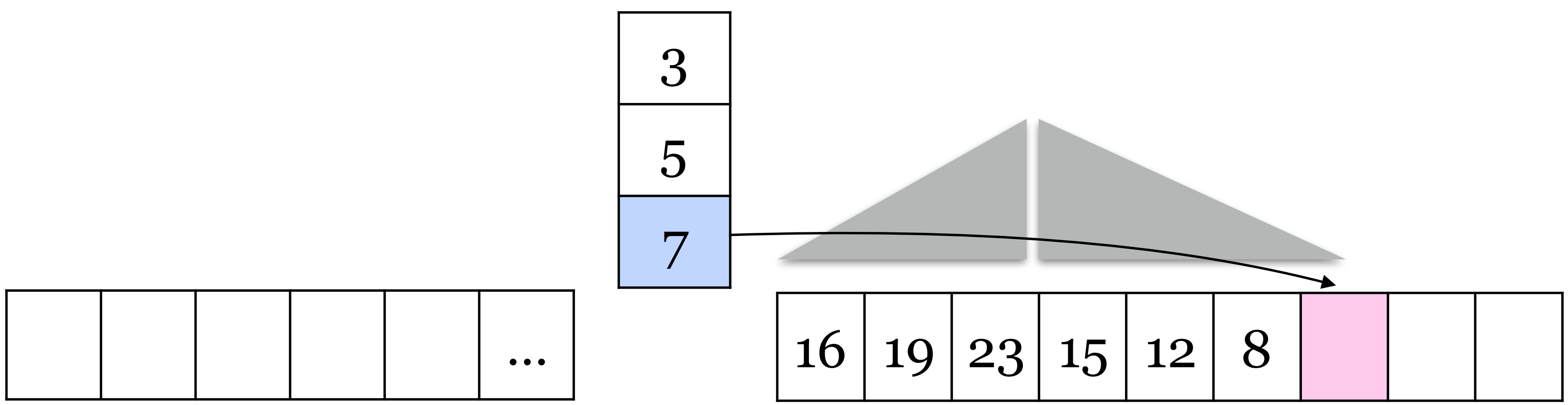
Alternating-Up-Down Schedule

- Deterministically alternate between up and down runs



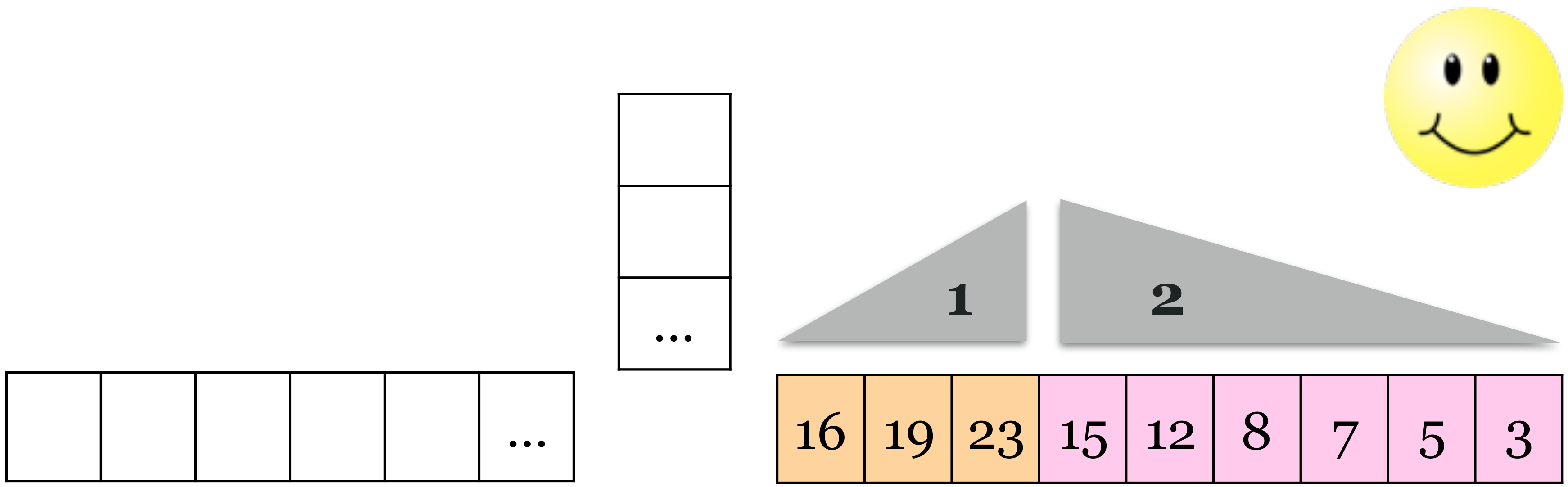
Alternating-Up-Down Schedule

- Deterministically alternate between up and down runs



Alternating-Up-Down Schedule

- Deterministically alternate between up and down runs



Runs of length $> M$

Alternating-Up-Down Schedule

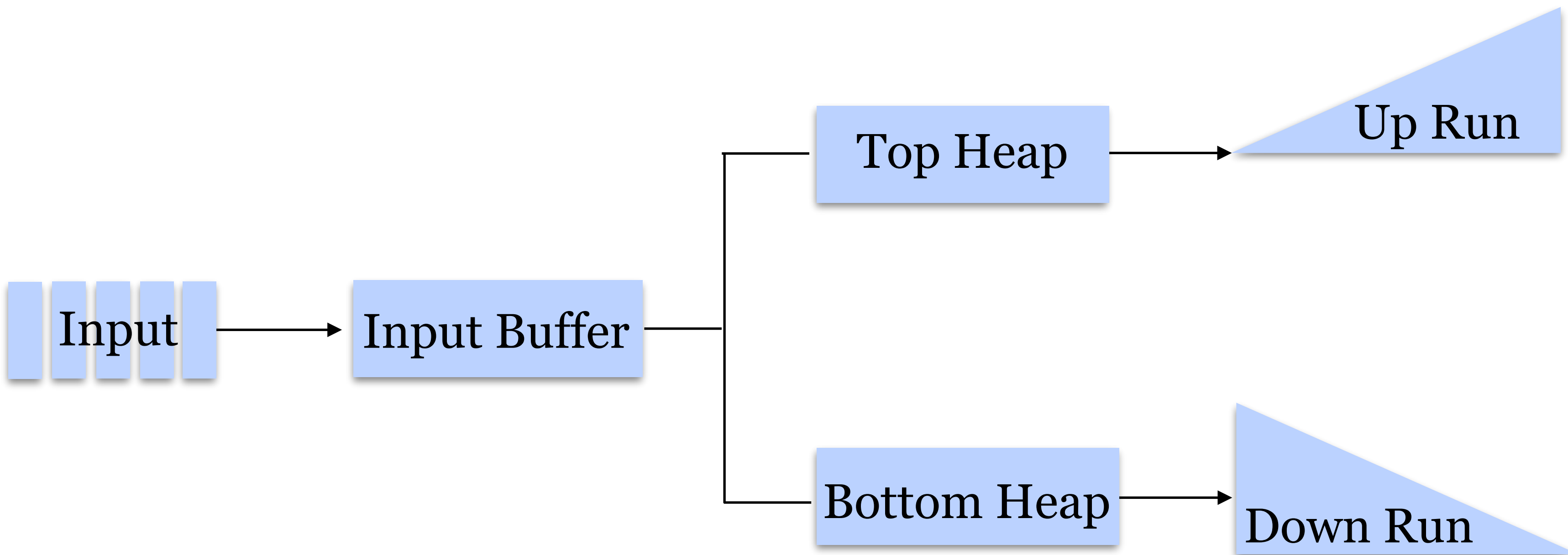
- *Is this better than replacement selection?*

Alternating-Up-Down Schedule

- *Is this better than replacement selection?*
- [Knuth 63] On random data, it is *worse*
    - Average run length is 1.5M, compared to 2M

Two-Way Replacement Selection

- [Martinez-Palau et al. VLDB 10]
  - Heuristically *choose* between an *up* and *down run*
  - Slightly better than Replacement Selection on *some* data



To run up or down, that is the question…

Our Main Contributions

- Theoretical foundation of the run generation problem
- Competitive analysis of run generation scheduling policies

" My Momma always said smart things about life and chocolates... But I need to know the theory behind it.."

Our Results

- Alternating-Up-Down Replacement Selection is
    - 2-competitive
    - Best possible
- Improve competitive ratio with *resource augmentation*
- Improve performance when input is *nearly sorted*

"My Momma always said smart things about life and chocolates... But I need to know the theory behind it.."

Summary of Our Results

Competitive Ratio	Buffer Size	Lookahead	Comments
2	**M**	**-**	**Tight**
1.5	M	3M	Tight
1.75	2M	M	Randomized
1	4M	3M	Tight
$(1+\varepsilon)$	M	N - M	Offline
1.5	2M	2M	*3-nearly sorted*
1	M	N	*5-nearly sorted*

Useful Observations

- Adding elements to an input stream cannot help
  - If *I'* is a subsequence of *I*, *OPT(I')*
- Writing extra elements (compared to OPT) doesn't hurt

WLOG

- Algorithm must always write *maximal runs*
  - Never end a run unless forced to
  - Never skip over elements

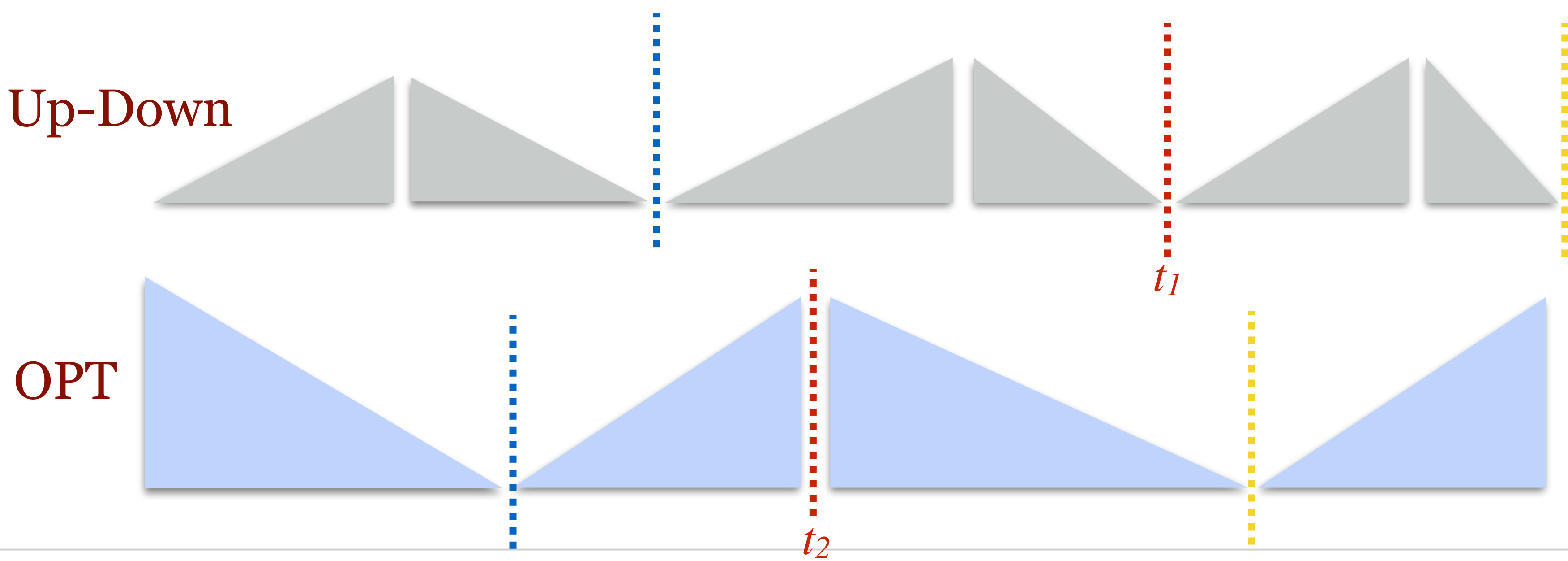
Summary of Our Results

Competitive Ratio	Buffer Size	Lookahead	Comments
2	**M**	**-**	**Tight**
1.5	M	3M	Tight
1.75	2M	M	Randomized
1	4M	3M	Tight
$(1+\varepsilon)$	M	N - M	Offline
1.5	2M	2M	*3-nearly sorted*
1	M	N	*5-nearly sorted*

Alternating-Up-Down is 2-competitive

Proof Sketch

- At each decision point, suppose OPT goes up/down
    - A maximal up and down run goes at least as far
    - Every two runs cover at least one run of OPT

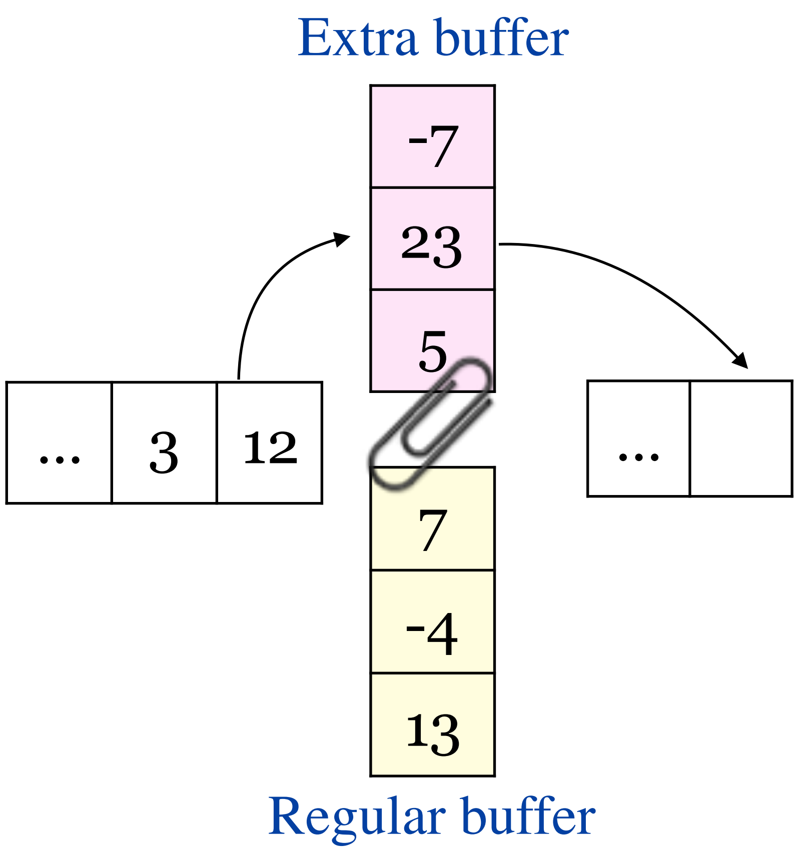


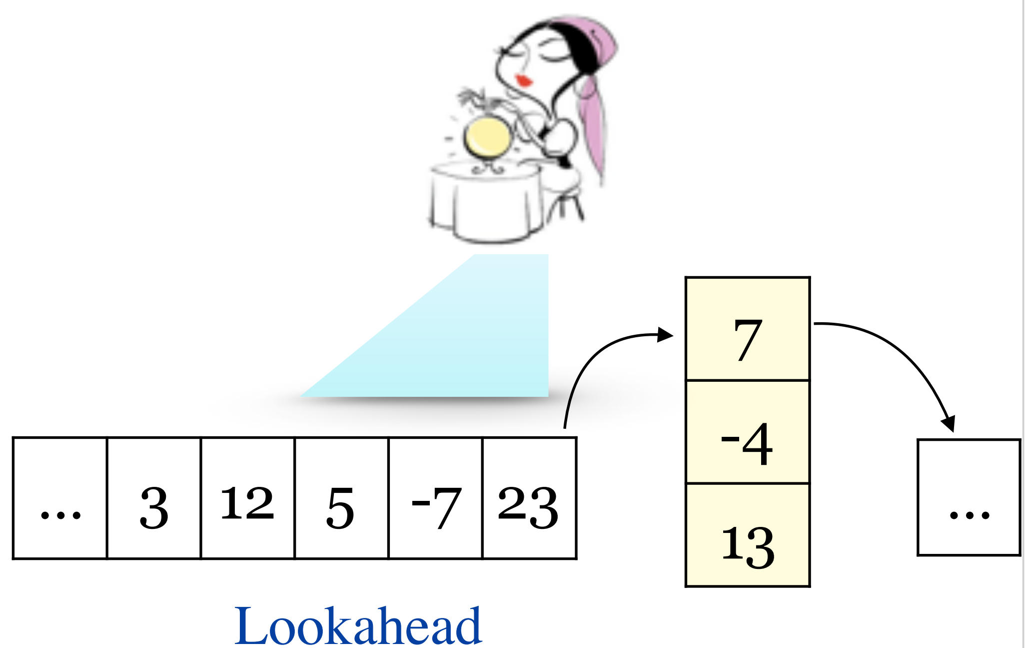
Lower Bounds

- No deterministic algorithm can do better than a 2-approx
  - Adversary switches the upcoming input wrt decision made
- No randomized algorithm can do better than a 1.5-approx
  - *Yao's minimax*

Resource Augmentation

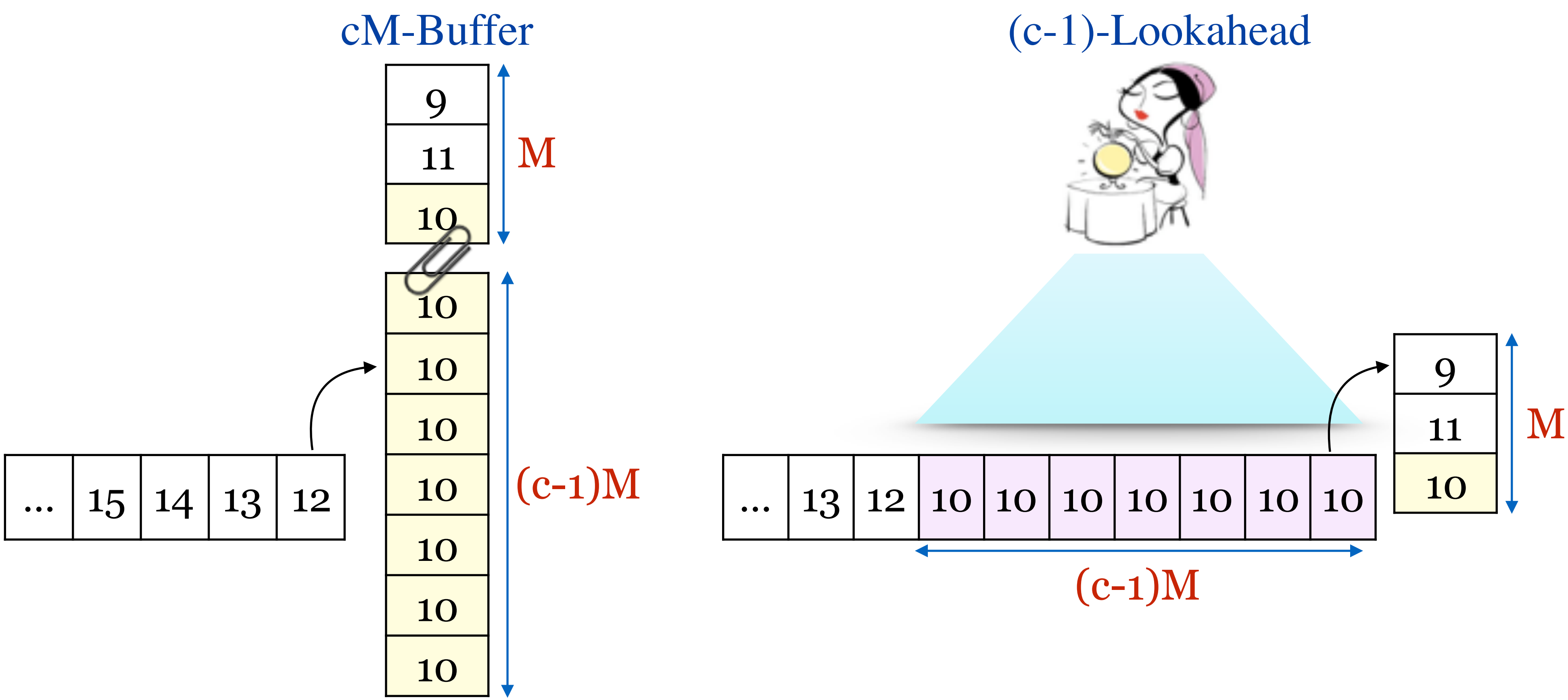
- No online algorithm can be better than a 2-approximation
  - *Can we do better with extra buffer or lookahead?*





Resource Augmentation: No Duplicates

- Resource augmentation results require uniqueness
  - *Duplicates nullify extra buffer or lookahead*

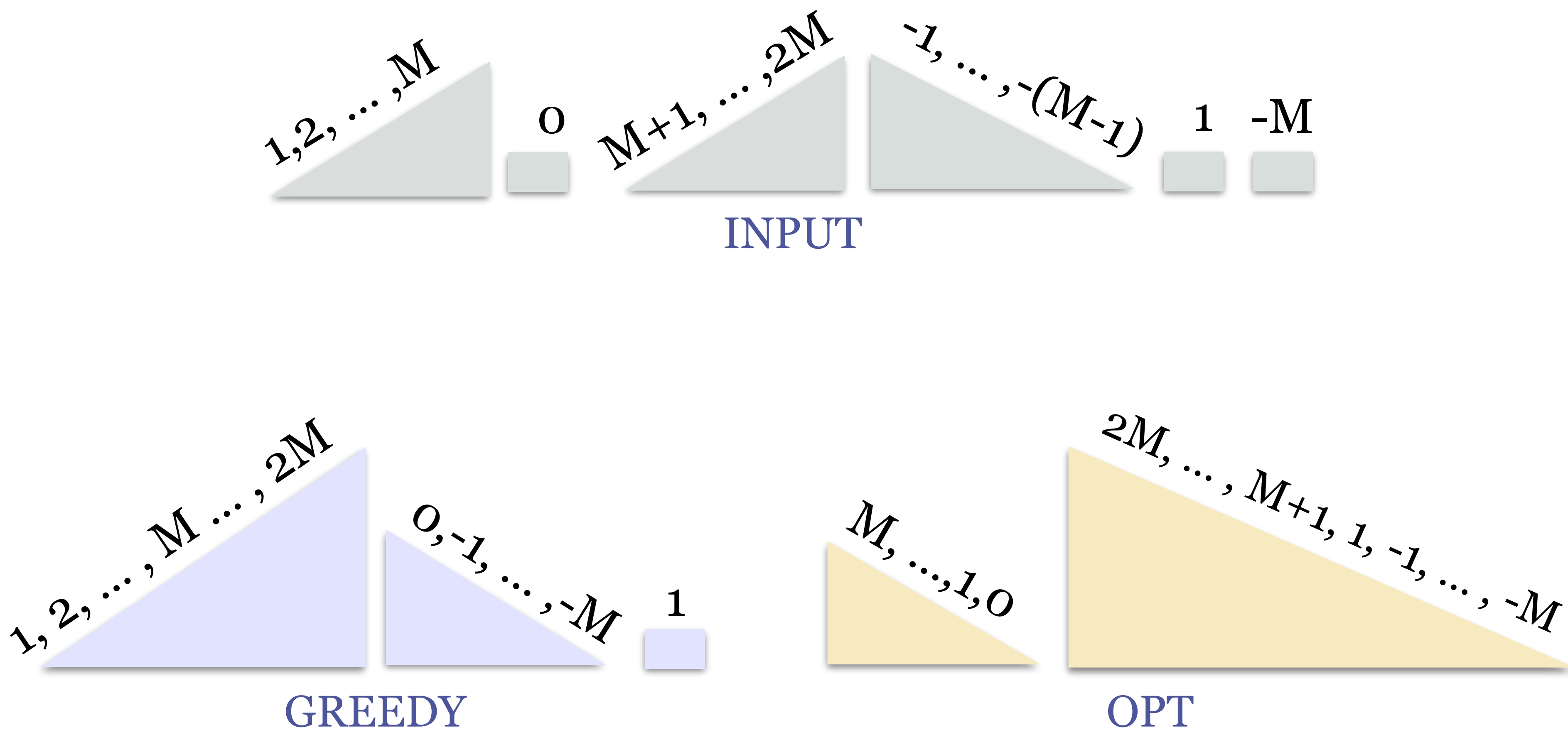


Main Idea Behind Resource Augmentation: What Would *Greedy* Do?

- Greedy chooses the longer run at every decision point
  - *Not* an online algorithm
- Greedy has some good guarantees
  - Upper bound and lower bound on run lengths

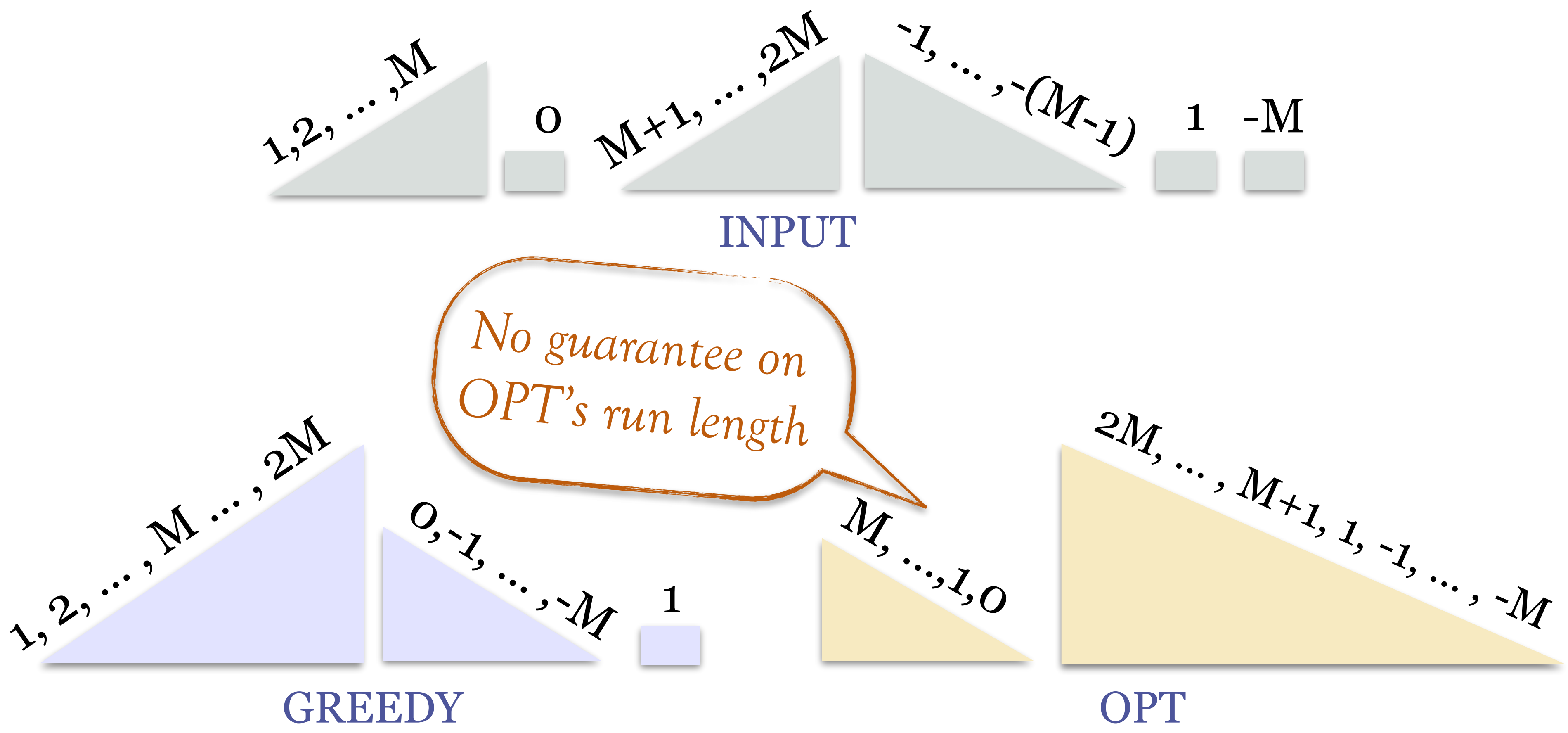
Note: Greedy is Not Optimal

- Can be as bad as 1.5 times OPT



Note: Greedy is Not Optimal

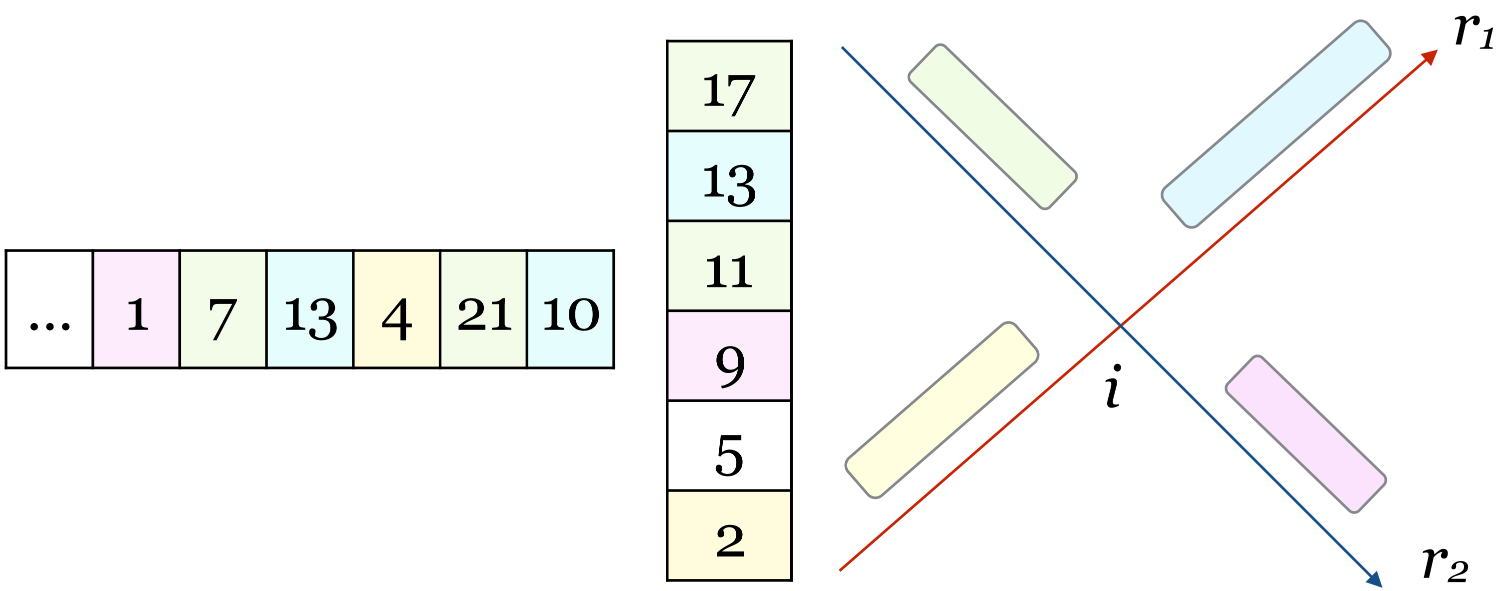
- Can be as bad as 1.5 times OPT



Greedy: How Long is the Not So Long Run?

Key Lemma

Given an input I, let r_1 and r_2 be two possible runs in opposite directions, then $|r_1| < 3M$ or $|r_2| < 3M$.



Greedy: How Long is the Not So Long Run?

Key Lemma

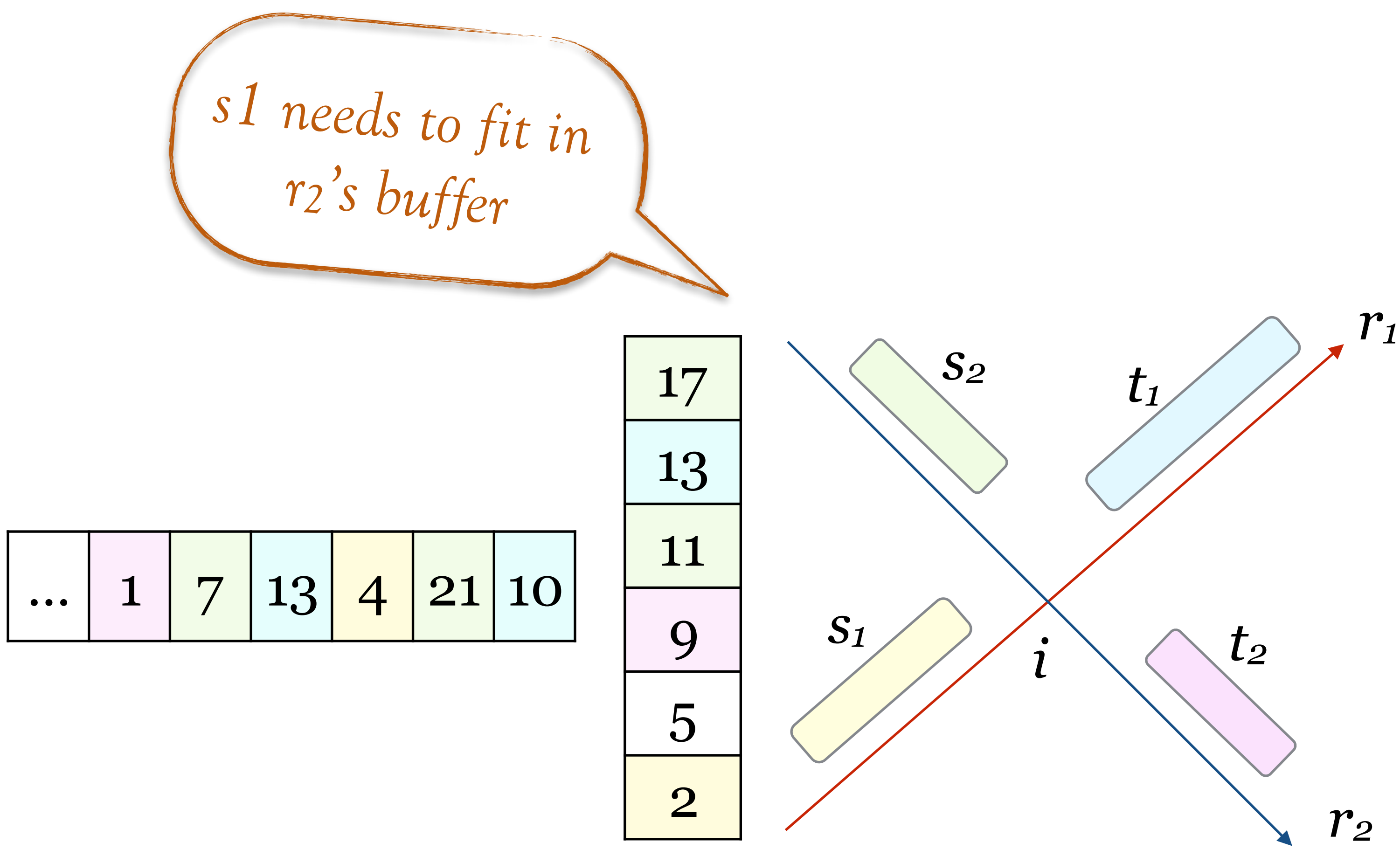
Given an input I with no duplicates, let r_1 and r_2 be two possible runs in opposite directions, then $|r_1| < 3M$ or $|r_2| < 3M$.

Take-away

- Don't have to look too far into the future to know greedy's choice

Sketchy Proof of Key Lemma

$s_1 \leq M$



Sketchy Proof of Key Lemma

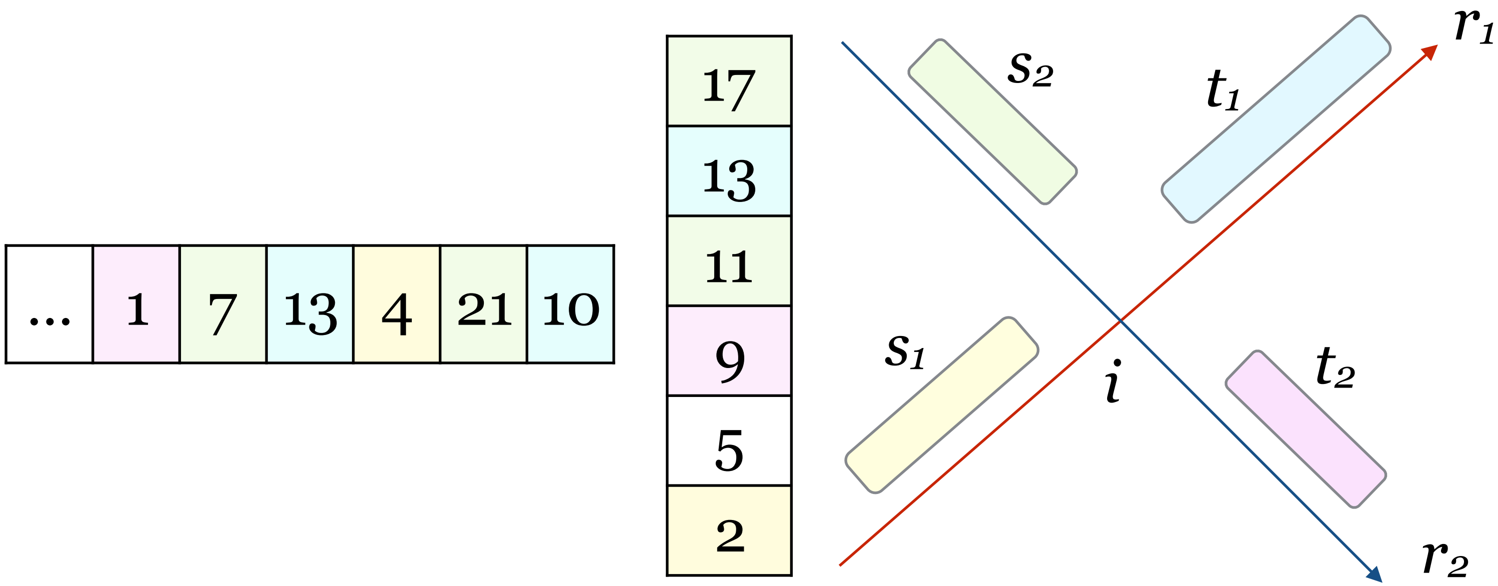
$s_1 \leq M$

$s_{2,N} + t_{1,B} \leq M$

$s_{2,N}$: Elements of s_2 not in initial buffer

$t_{1,B}$: Elements of t_1 in initial buffer

Both need to fit in r_1's buffer at i



Sketchy Proof of Key Lemma

$s_1 \leq M$

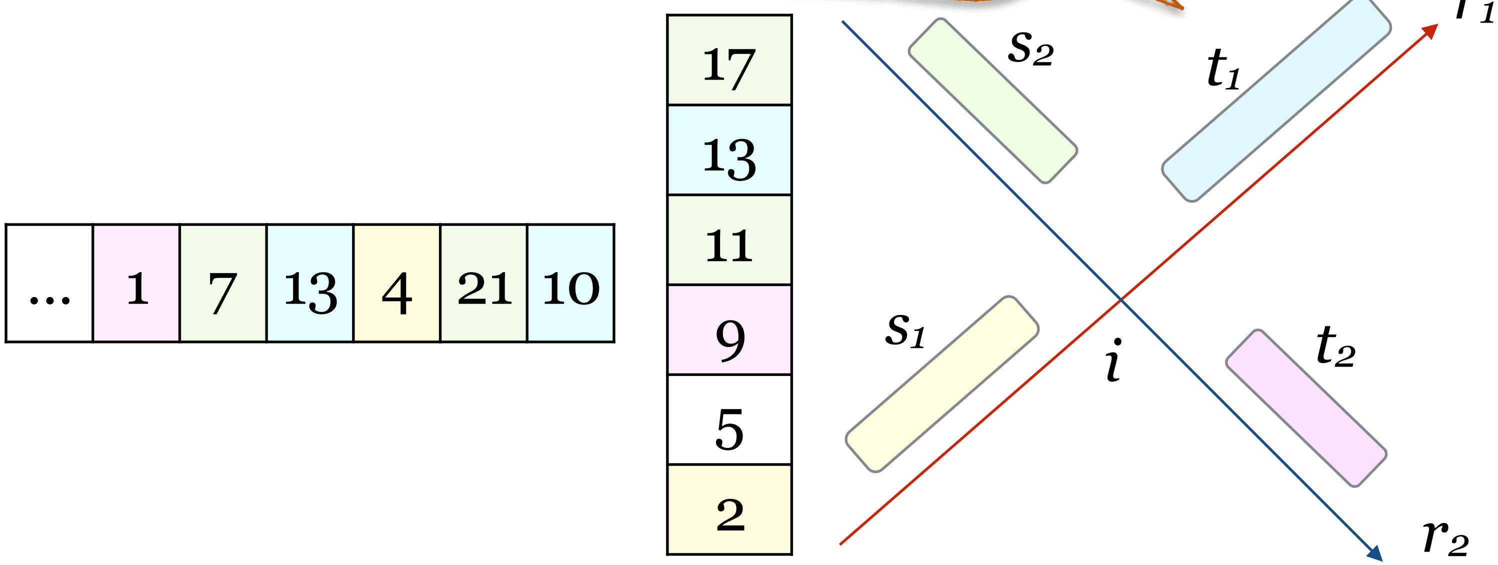
$s_{2,N} + t_{1,B} \leq M$

$s_{2,N}$: Elements of s_2 not in initial buffer

$t_{1,B}$: Elements of t_1 in initial buffer

$t_{1,i}$: Elements in r_1 and read in after i

$t_{1,i}$ cannot be included in r_2



Sketchy Proof of Key Lemma

$s_1 \leq M$

$s_{2,N} + t_{1,B} \leq M$

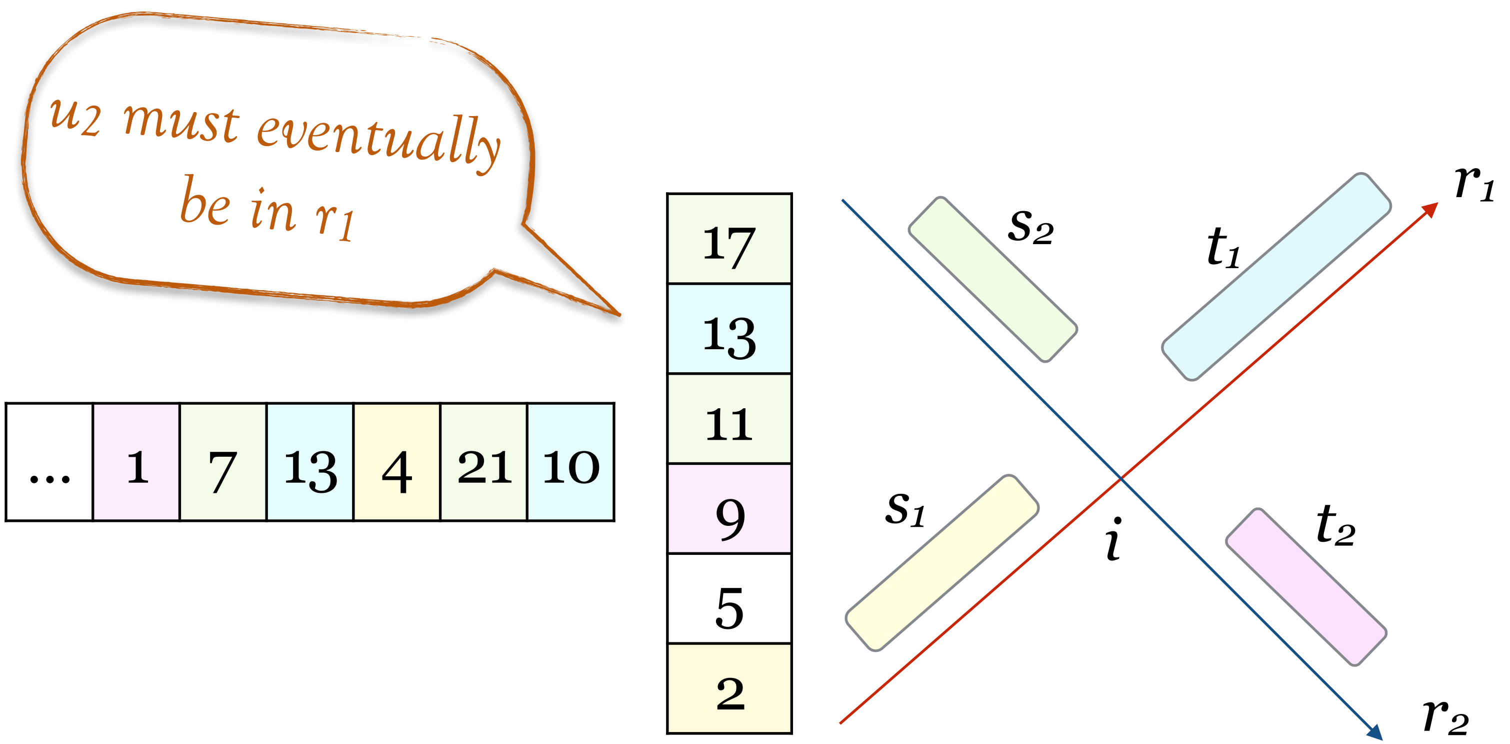
$u_2 \leq M$

$s_{2,N}$: Elements of s_2 not in initial buffer

$t_{1,B}$: Elements of t_1 in initial buffer

$t_{1,i}$: Elements in r_1 and read in after i

u_2 : Elements not in r_2 and read in before i



Sketchy Proof of Key Lemma

$s_1 \leq M$

$s_{2,N} + t_{1,B} \leq M$

$u_2 \leq M$

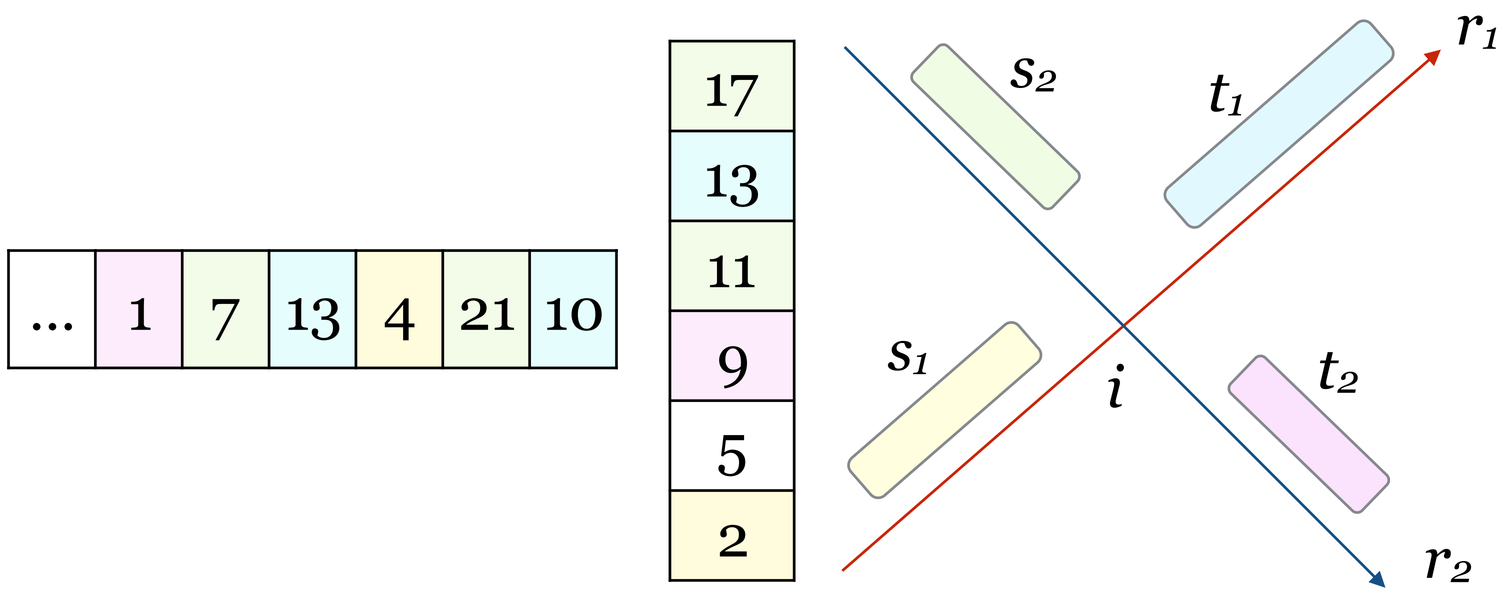
$s_{2,N}$: Elements of s_2 not in initial buffer

$t_{1,B}$: Elements of t_1 in initial buffer

$t_{1,i}$: Elements in r_1 and read in after i

u_2 : Elements not in r_2 and read in before i

$$\boxed{r_1 \leq s_1 + s_{2,N} + t_{1,B} + t_{1,i} + u_2}$$



Sketchy Proof of Key Lemma

$s_1 \leq M$

$s_{2,N} + t_{1,B} \leq M$

$u_2 \leq M$

$s_{2,N}$: Elements of s_2 not in initial buffer

$t_{1,B}$: Elements of t_1 in initial buffer

$t_{1,i}$: Elements in r_1 and read in after i

u_2 : Elements not in r_2 and read in before i

$$\boxed{r_1 \leq s_1 + s_{2,N} + t_{1,B} + t_{1,i} + u_2}$$

Weaker bound of 4M

$$\boxed{\text{If } r_1 \geq 4M \text{ then } t_{1,i} \geq M}$$

Sketchy Proof of Key Lemma

$s_1 \leq M$

$s_{2,N} + t_{1,B} \leq M$

$u_2 \leq M$

$s_{2,N}$: Elements of s_2 not in initial buffer

$t_{1,B}$: Elements of t_1 in initial buffer

$t_{1,i}$: Elements in r_1 and read in after i

u_2 : Elements not in r_2 and read in before i

$$\boxed{r_1 \leq s_1 + s_{2,N} + t_{1,B} + t_{1,i} + u_2}$$

Weaker bound of 4M

But $t_{1,i}$ needs to fit in r_2's buffer

$$\boxed{\text{If } r_1 \geq 4M \text{ then } t_{1,i} \geq M}$$

$$\boxed{r_2 < 4M}$$

Greedy: How Long is the Not So Long Run?

Key Lemma

Given an input I with no duplicates, let r_1 and r_2 be two possible runs in opposite directions, then $|r_1| < 3M$ or $|r_2| < 3M$.

Take-away

- Don't have to look too far into the future to know greedy's choice

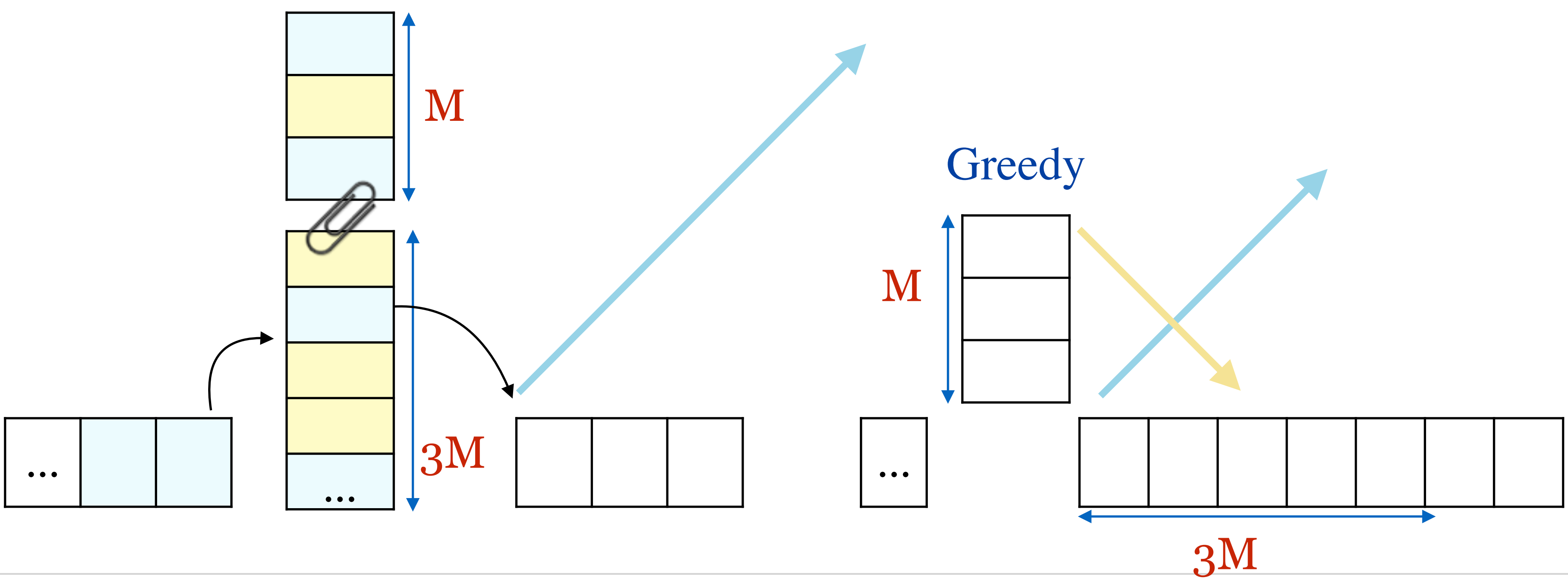
Summary of Our Results

Competitive Ratio	Buffer Size	Lookahead	Comments
2	M	-	Tight
1.5	M	3M	Tight
1.75	2M	M	Randomized
1	**4M**	**3M**	**Tight**
$(1+\varepsilon)$	M	N - M	Offline
1.5	2M	2M	*3-nearly sorted*
1	M	N	*5-nearly sorted*

Warm Up: Matching OPT with 4M buffer

Algorithm

1. Read elements until entire buffer (4M) is full
2. Determine what greedy (with M buffer) would do
3. Write a maximal run in greedy's direction



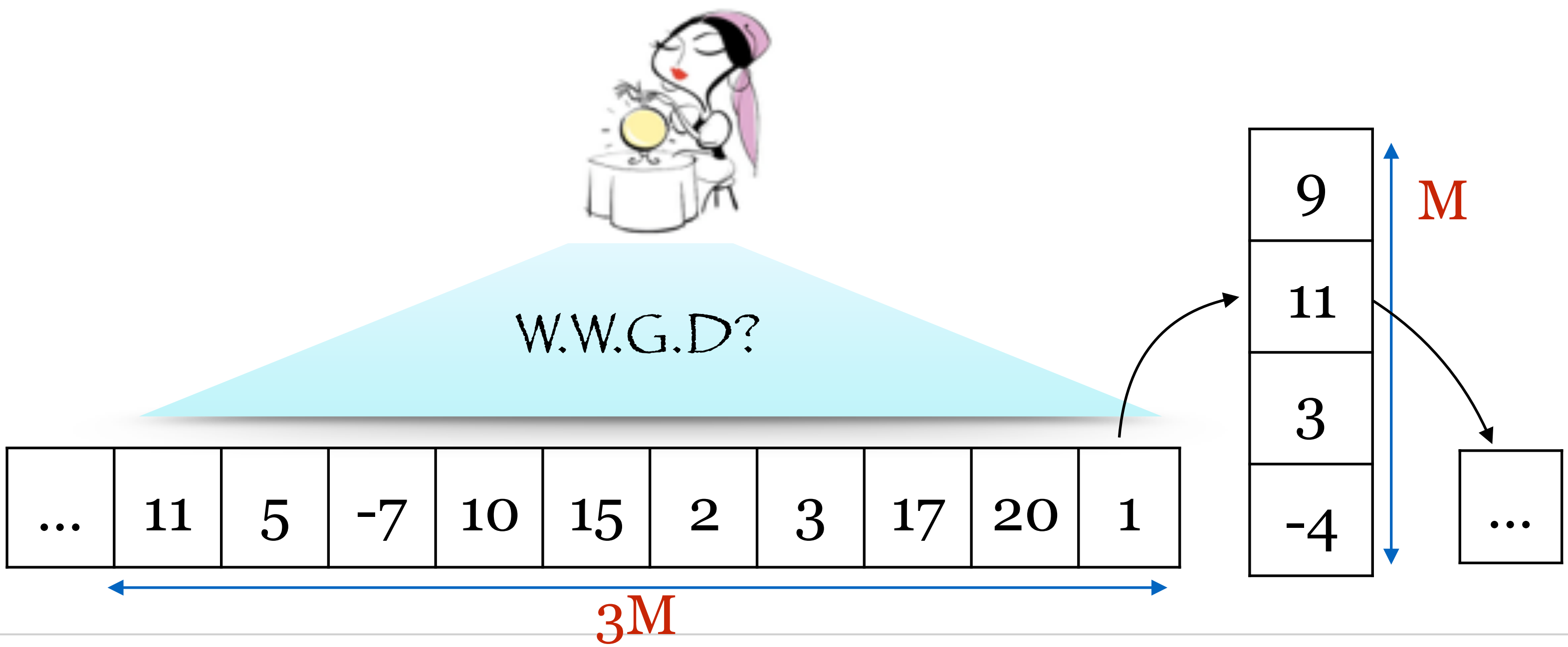
Summary of Our Results

Competitive Ratio	Buffer Size	Lookahead	Comments
2	M	-	Tight
1.5	**M**	**3M**	**Tight**
1.75	2M	M	Randomized
1	4M	3M	Tight
$(1+\varepsilon)$	M	N - M	Offline
1.5	2M	2M	*3-nearly sorted*
1	M	N	*5-nearly sorted*

Theorem: 1.5-Approximation with 4M-visibility

Algorithm

1. Determine what greedy (with M buffer) would do
2. Write a maximal run in greedy's direction
3. Write two more - in the same and opposite direction



Theorem: 1.5-Approximation with 4M-visibility

Algorithm

1. Determine what greedy (with M buffer) would do
2. Write a maximal run in greedy's direction
3. Write two more - in the same and opposite direction

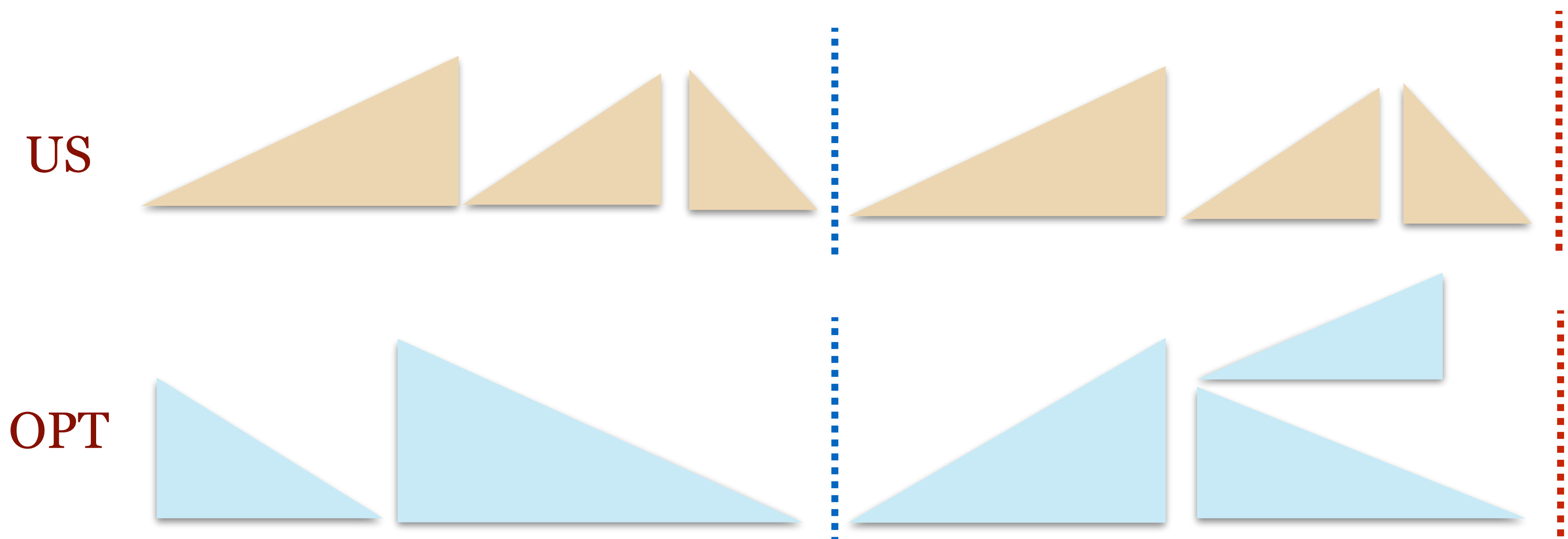
Lemma

At any decision point, if OPT chooses a non-greedy run (say down), it's next run must be in the same direction (down).

Theorem: 1.5-Approximation with 4M-visibility

Algorithm

1. Determine what greedy (with M buffer) would do
2. Write a maximal run in greedy's direction
3. Write two more - in the same and opposite direction



Lower Bound on Resource Augmentation

Almost tight

- With a buffer of size 4M-2
  - No deterministic algorithm can do better than 1.5-approx
- Above lower bound implies lower bound for 4M-2 visibility

Summary of Our Results

Competitive Ratio	Buffer Size	Lookahead	Comments
2	M	-	Tight
1.5	M	3M	Tight
1.75	**2M**	**M**	**Randomized**
1	4M	3M	Tight
$(1+\varepsilon)$	M	N - M	Offline
1.5	2M	2M	*3-nearly sorted*
1	M	N	*5-nearly sorted*

Summary of Our Results

Competitive Ratio	Buffer Size	Lookahead	Comments
2	M	-	Tight
1.5	M	3M	Tight
1.75	2M	M	Randomized
1	4M	3M	Tight
$(1+\varepsilon)$	**M**	**N - M**	**Offline**
1.5	2M	2M	*3-nearly sorted*
1	M	N	*5-nearly sorted*

Offline Run Generation Problem

- Given the input in advance, compute the policy which produces the minimum possible number of runs
- We have a PTAS
- **OPEN problem:** Polynomial time offline (exact) policy?

Summary of Our Results

Competitive Ratio	Buffer Size	Lookahead	Comments
2	M		Tight
1.5	M		ight
1.75	2M		omized
1	4M		ight
(1+ε)	M		ine
1.5	**2M**	**2M**	***3-nearly sorted***
1	**M**	**N**	***5-nearly sorted***

c-nearly sorted:
Optimal has runs of length at least cM

The Road Ahead

- Polynomial offline exact algorithm
- Does Randomization help?
- Practical speed ups
  - How can we use the new structural insights?
- Parallel instead of sequential writes?
  - Very similar to *Patience Sort*

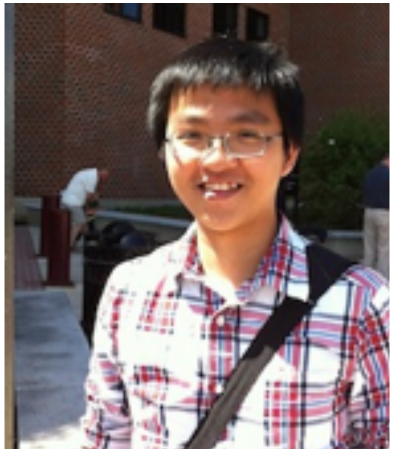
A Shout Out to the Team!

Michael

Sam

Andrew

Hoa



" And that's all I have to say about that.."