Funnel Heap

- A Cache-Oblivious Priority Queue

Gerth Stølting Brodal   Rolf Fagerberg

BRICS

University of Aarhus

Thirteenth International Symposium on Algorithms and Computation

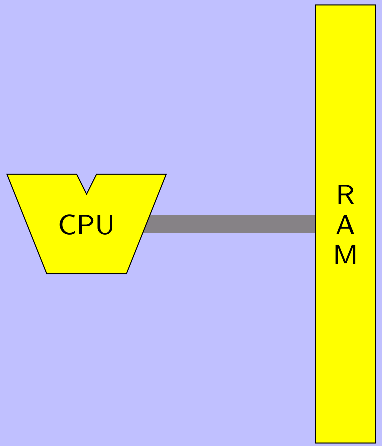
Vancouver, BC, Canada, November 22, 2002

Outline of talk

- Cache-oblivious model
- Cache-oblivious results
- Cache-oblivious priority queues
- Funnel heap
  - k - merger
  - the data structure
  - operations

The Classic RAM Model

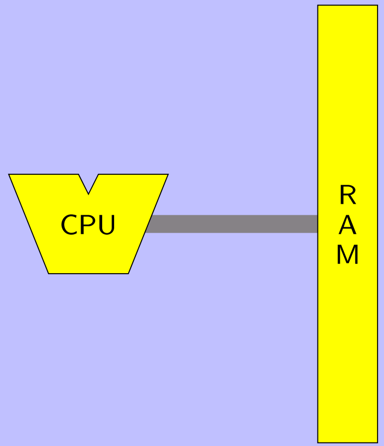
The RAM model:



Add:	$O(1)$
Mult:	$O(1)$
Mem access:	$O(1)$

The Classic RAM Model

The RAM model:

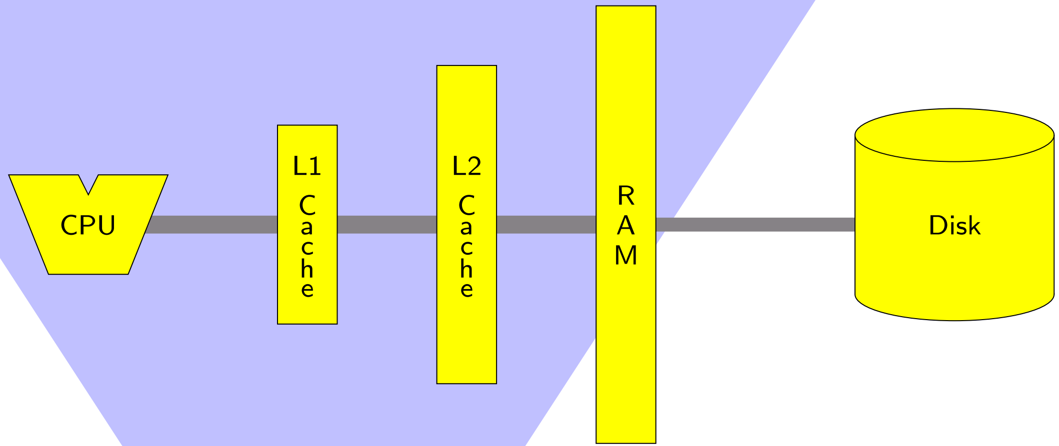


Add: $O(1)$

Mult: $O(1)$

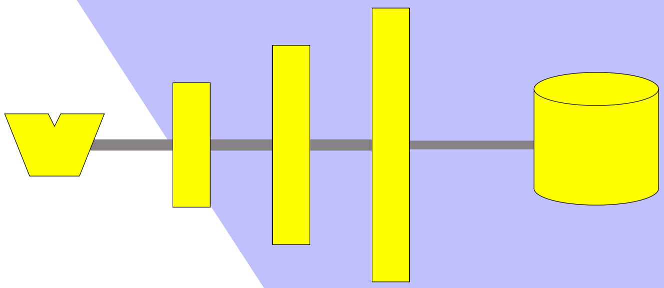
~~Mem access: $O(1)$~~

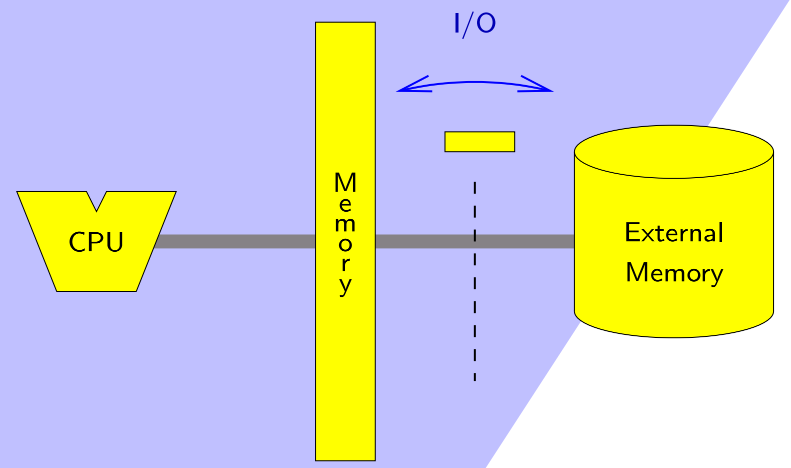
Real life:



Bottleneck: transfer between two highest memory levels in use

The I/O Model





N = problem size
M = memory size
B = I/O block size

Aggarwal and Vitter 1988

- One I/O moves B consecutive records from/to disk
- Cost: number of I/Os

$$\text{Scan}(N) = O(N/B) \qquad \text{Sort}(N) = O\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$$

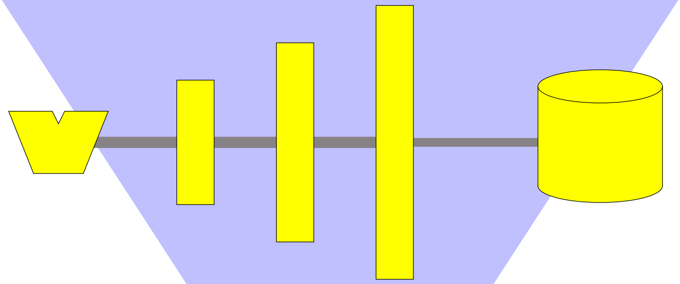
Cache-Oblivious Model

Frigo, Leiserson, Prokop, Ramachandran 1999

- Program in the RAM model
- Analyze in the I/O model (for arbitrary B and M)

Advantages

- Optimal on arbitrary level $\Rightarrow$ optimal on all levels
- B and M not hard-wired into algorithm



Cache-Oblivious Results

- Scanning $\Rightarrow$ stack, queue, selection, ...
- Sorting, matrix multiplication, FFT

  Frigo, Leiserson, Prokop, Ramachandran, FOCS'99

- Cache oblivious search trees

  Prokop 99
  Bender, Demaine, Farach-Colton, FOCS'00
  Rahman, Cole, Raman, WAE'01
  Bender, Duan, Iacono, Wu and Brodal, Fagerberg, Jacob, SODA'02

- Priority queue and graph algorithms

  Arge, Bender, Demaine, Holland-Minkley, Munro, STOC'02

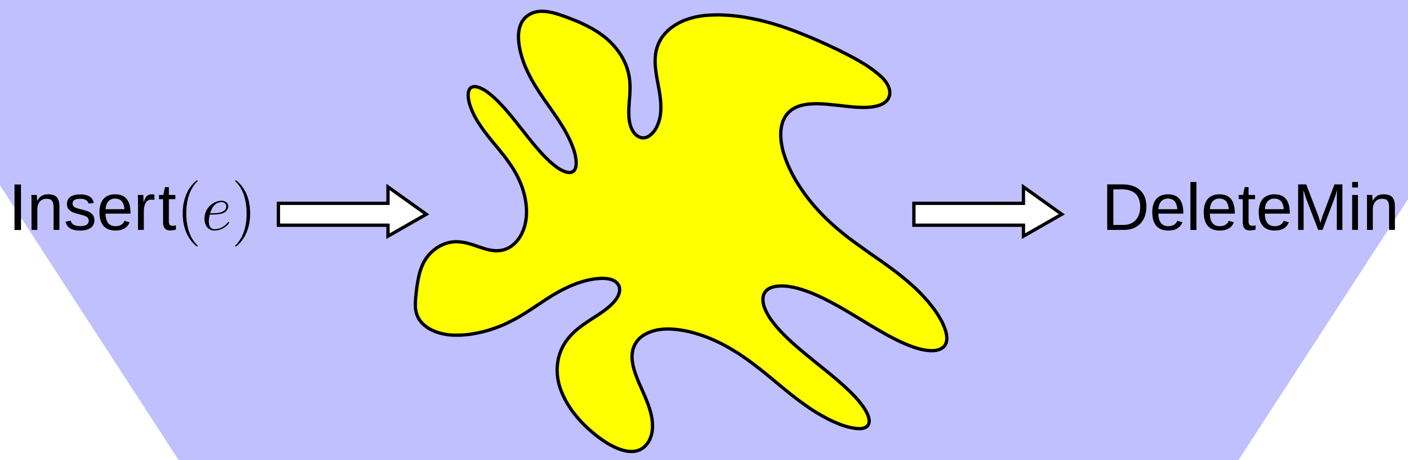
- Computational geometry

  Bender, Cole, Raman, ICALP'02
  Brodal, Fagerberg, ICALP'02

- Scanning dynamic sets

  Bender, Cole, Demaine, Farach-Colton, ESA'02

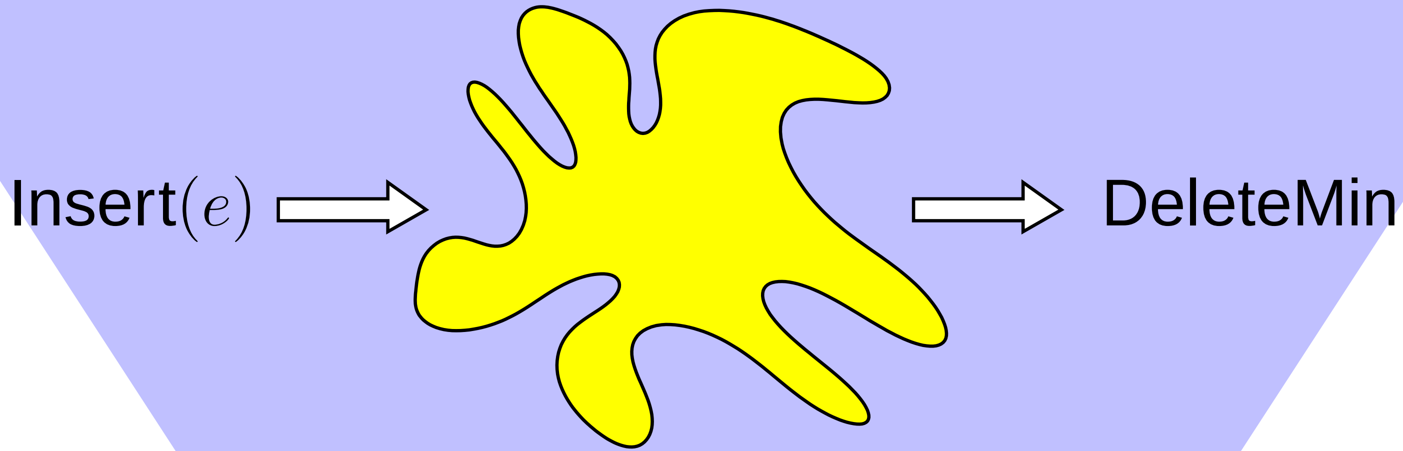
Priority Queues



Classic RAM:

- Heap: $O(\log_2 n)$ time — Williams 1964

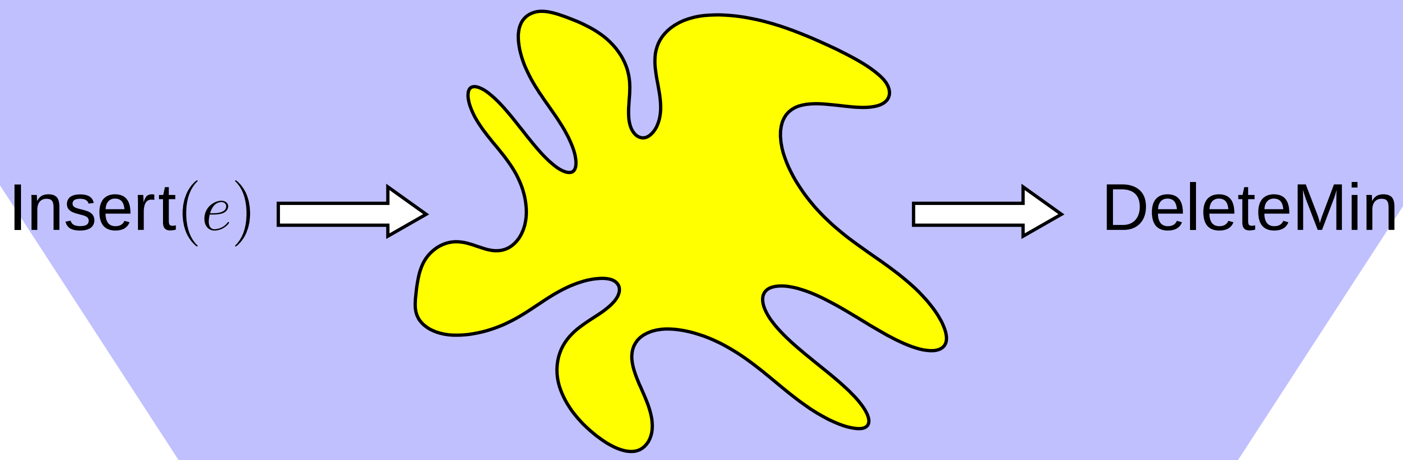
Priority Queues



Classic RAM:

- Heap: $O(\log_2 n)$ time, $O\left(\log_2 \frac{N}{M}\right)$ I/Os   Williams 1964

Priority Queues



Classic RAM:

- Heap: $O(\log_2 n)$ time, $O\left(\log_2 \frac{N}{M}\right)$ I/Os — Williams 1964

I/O model:

- Buffer tree: $O\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right) = O\left(\frac{\text{Sort}(N)}{N}\right)$ I/Os — Arge 1995

Cache-Oblivious Priority Queues

Arge, Bender, Demaine, Holland-Minkley, Munro 2002

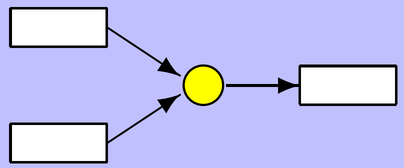
- $O\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os
  - Uses sorting and selection as subroutines
  - Requires tall cache assumption, $M \geq B^2$

Cache-Oblivious Priority Queues

- $O\left(\frac{1}{B}\log_{M/B}\frac{N}{B}\right)$ I/Os — Arge, Bender, Demaine, Holland-Minkley, Munro 2002
  - Uses sorting and selection as subroutines
  - Requires tall cache assumption, $M \geq B^2$
- Funnel heap — This talk
  - Uses only binary merging
  - Profile adaptive, i.e. $O\left(\frac{1}{B}\log_{M/B}\frac{N_i}{B}\right)$ I/Os

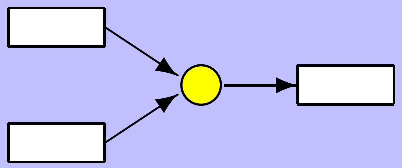
    N_i is either the size profile, max depth profile, or #insertions during the lifetime of the ith inserted element

Merge Trees

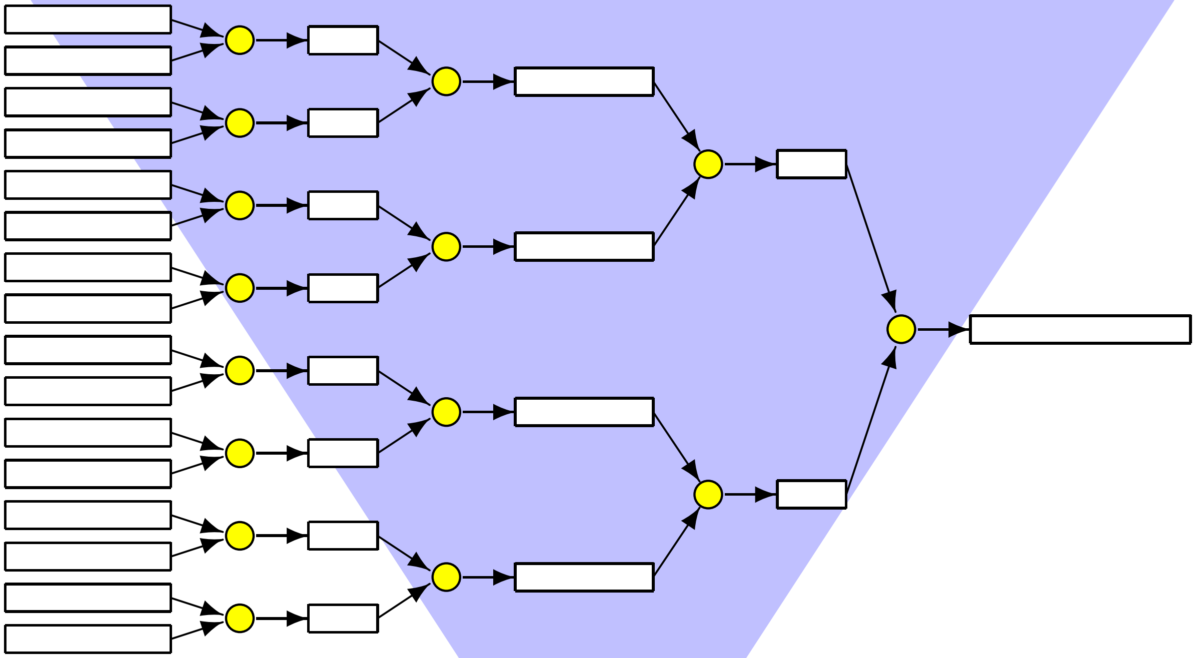


Binary merger

Merge Trees

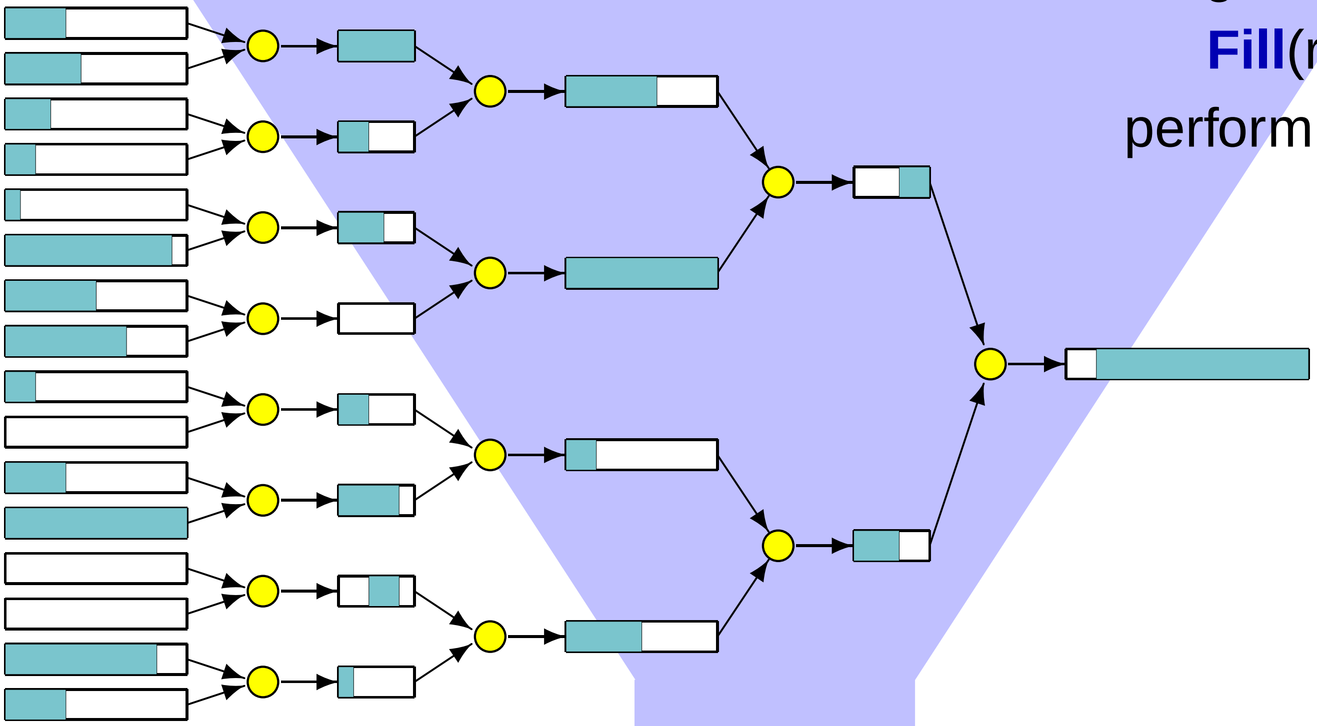


Binary merger



Merge tree

Merging Algorithm

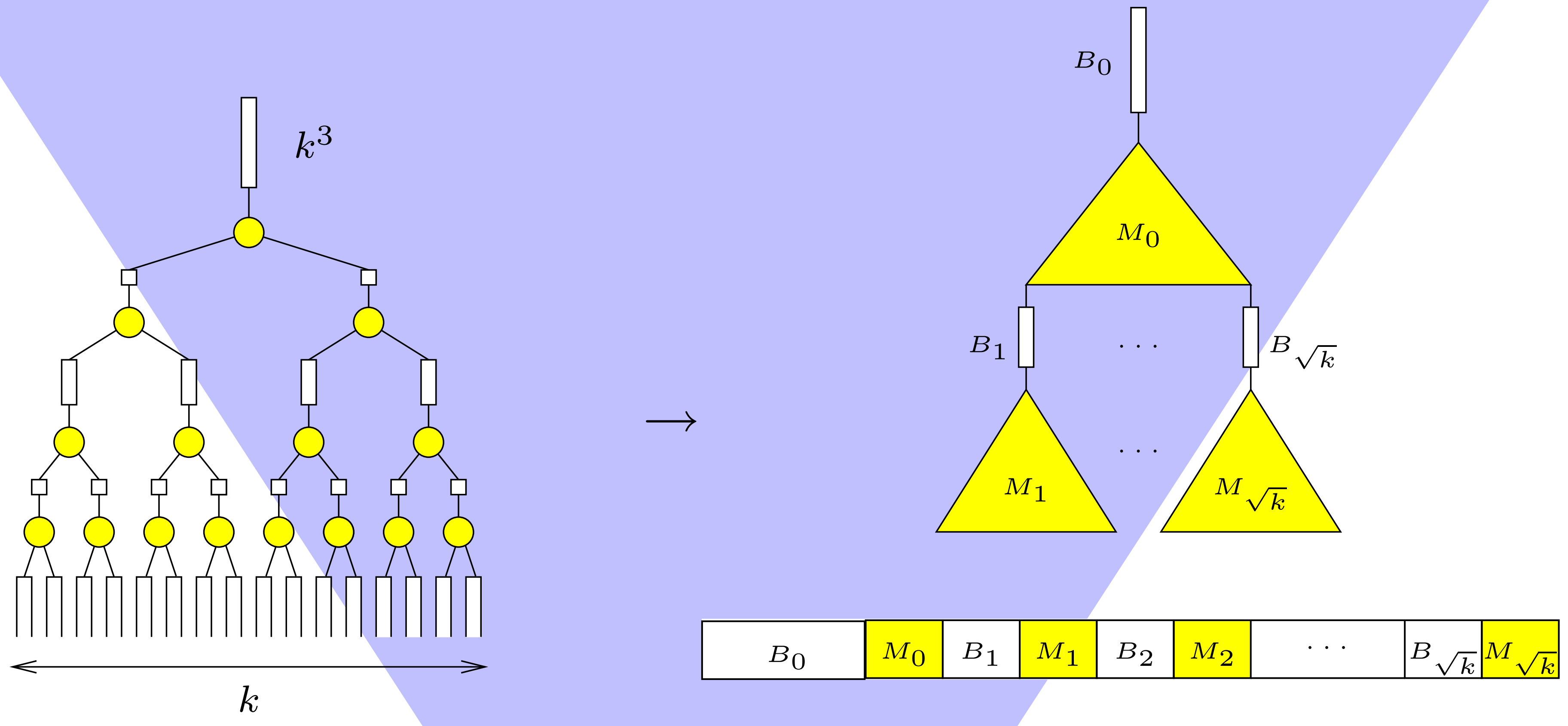


Fill(v)

---

while out-buffer not full
    if left in-buffer empty
        Fill(left child)
    if right in-buffer empty
        Fill(right child)
    perform one merge step

k - merger

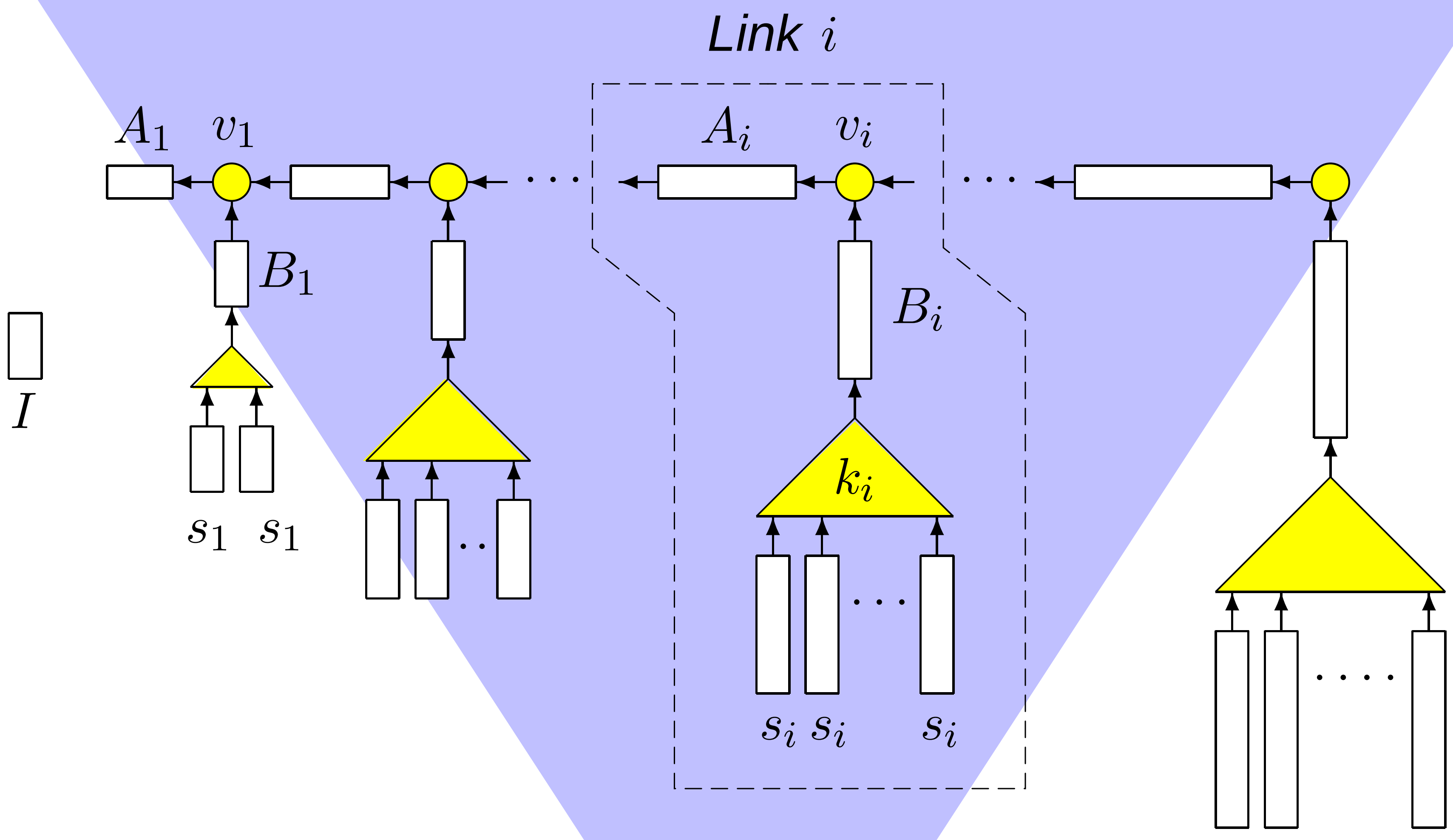


Recursive memory layout
Recursive defi nition of buffer sizes

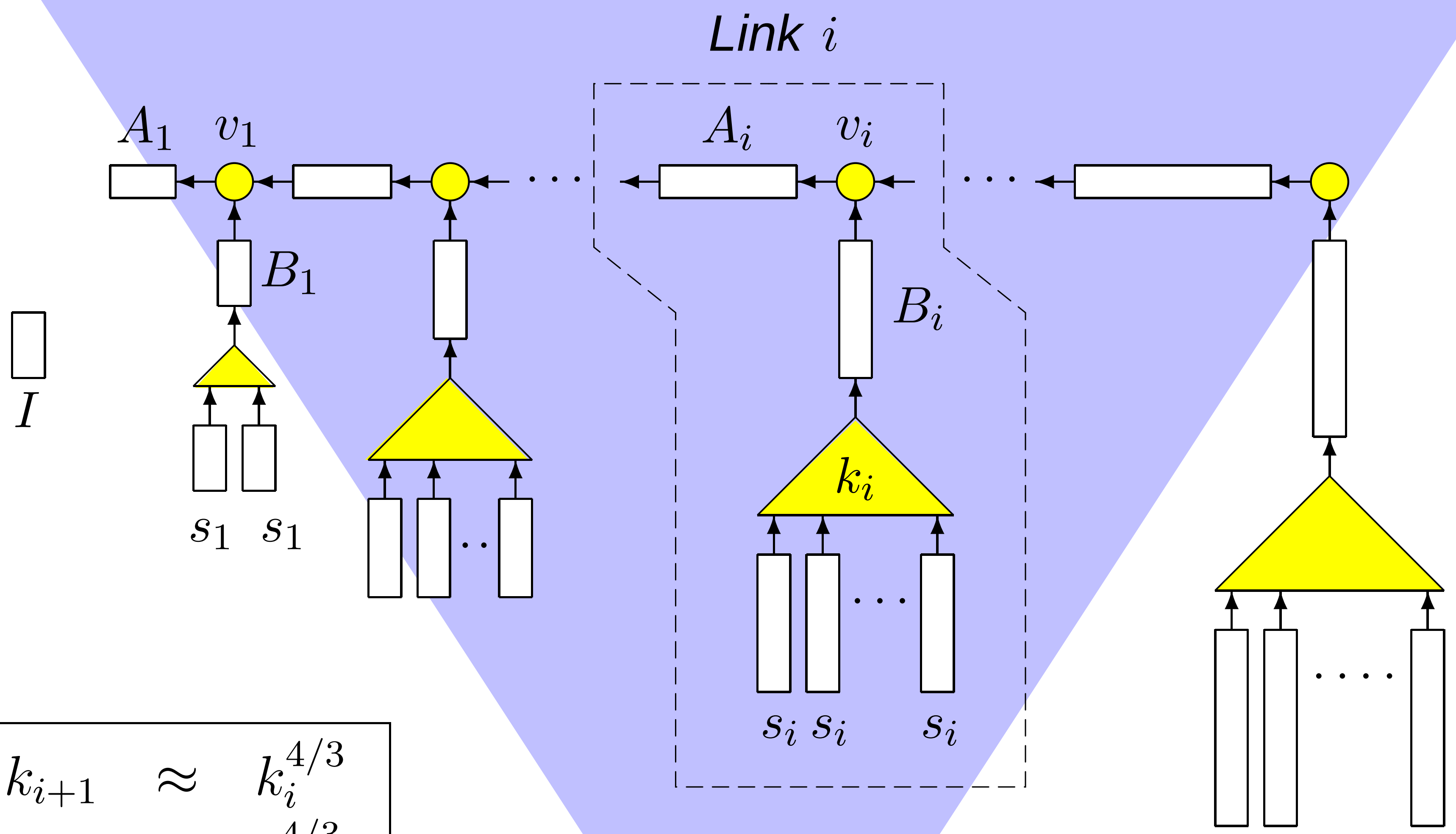
Lemma: $O(\frac{k^3}{B}\log_M(k^3)+k)$ I/Os per invocation (if $M \geq B^2$)

Frigo, Leiserson, Prokop, Ramachandran 1999
Brodal, Fagerberg 2002

The Priority Queue

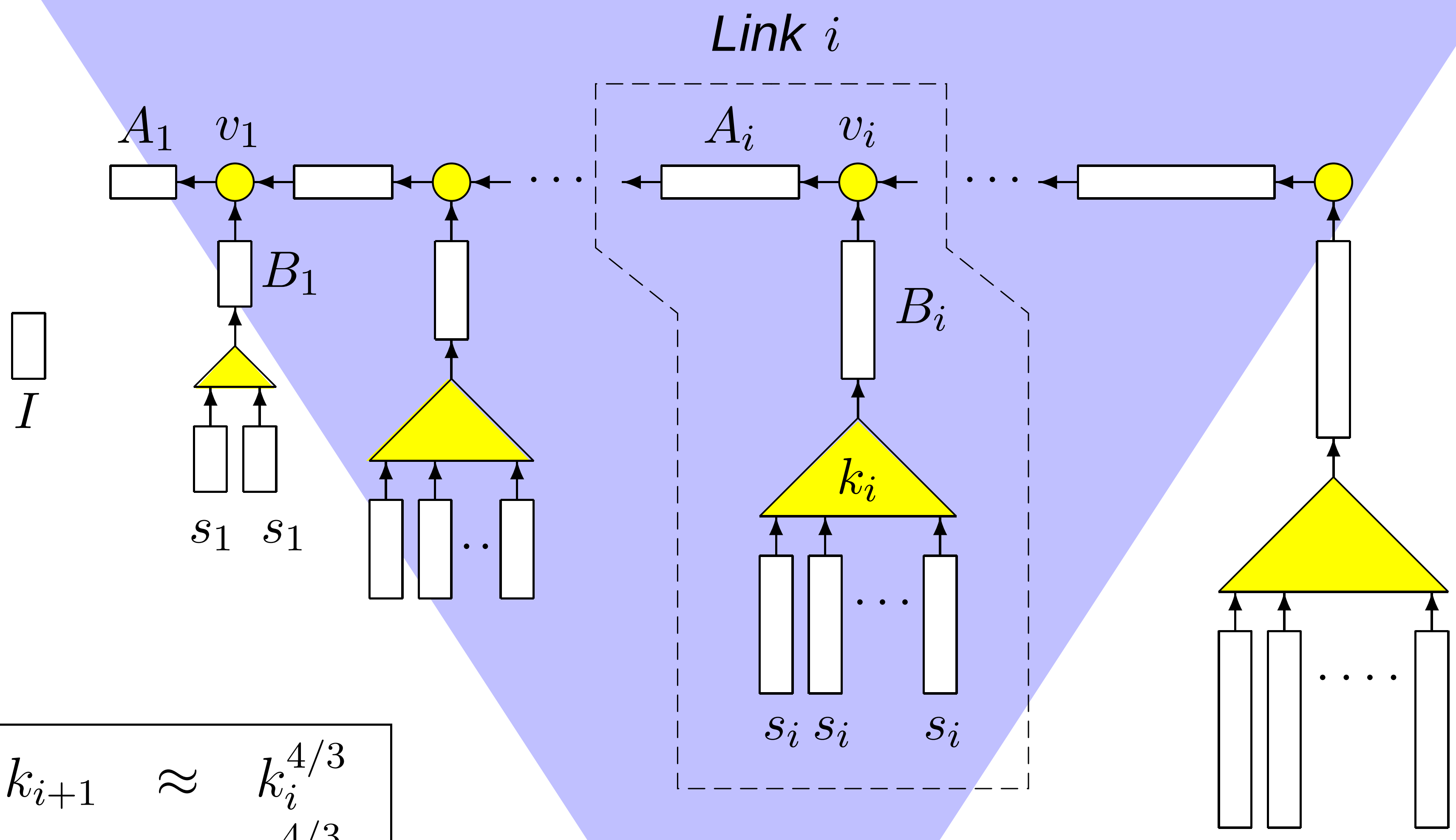


The Priority Queue



$$
\begin{aligned}
k_{i+1} &\approx k_i^{4/3} \\
s_{i+1} &\approx s_i^{4/3} \\
k_i &\approx s_i^{1/3}
\end{aligned}
$$

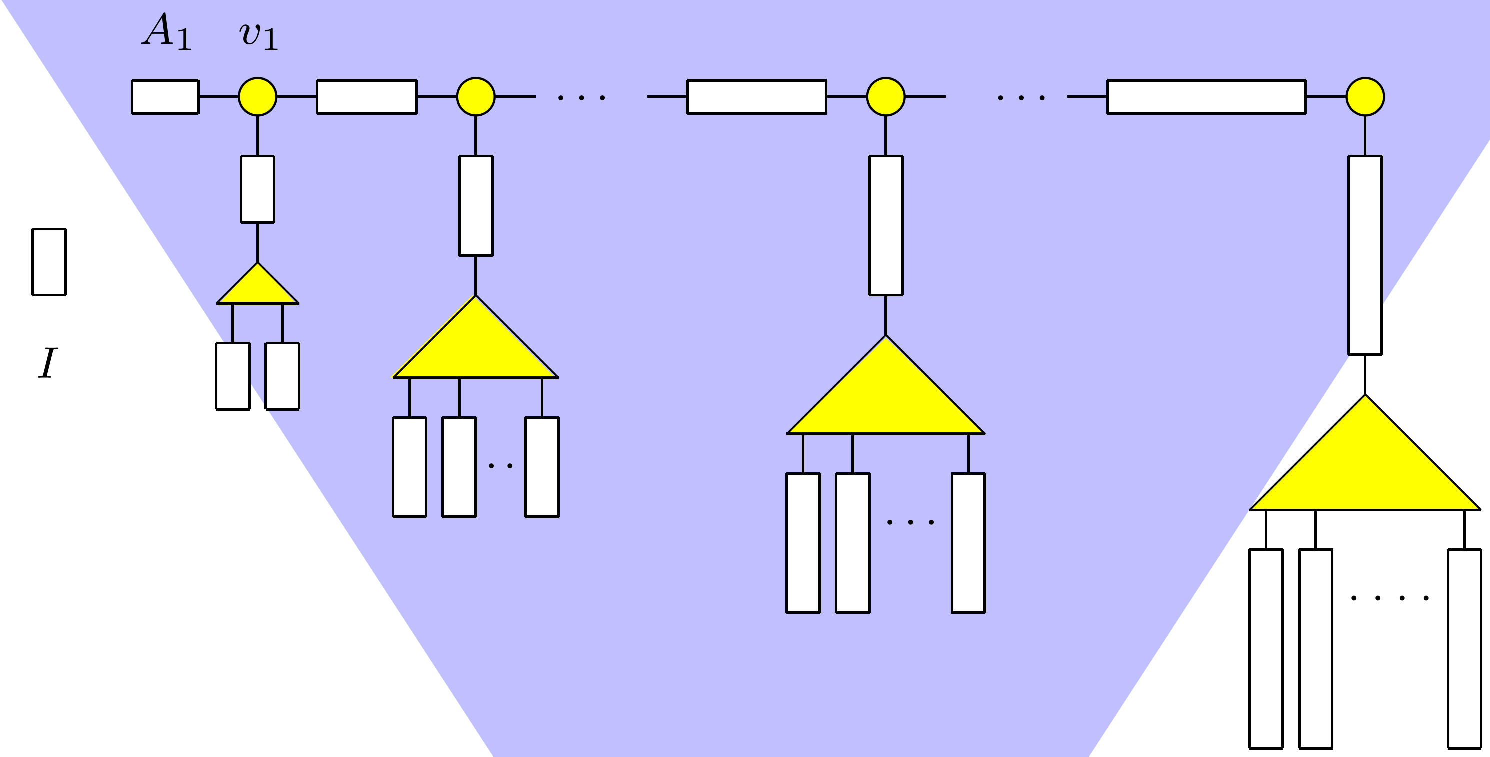
The Priority Queue



$$k_{i+1} \approx k_i^{4/3}$$
$$s_{i+1} \approx s_i^{4/3}$$
$$k_i \approx s_i^{1/3}$$

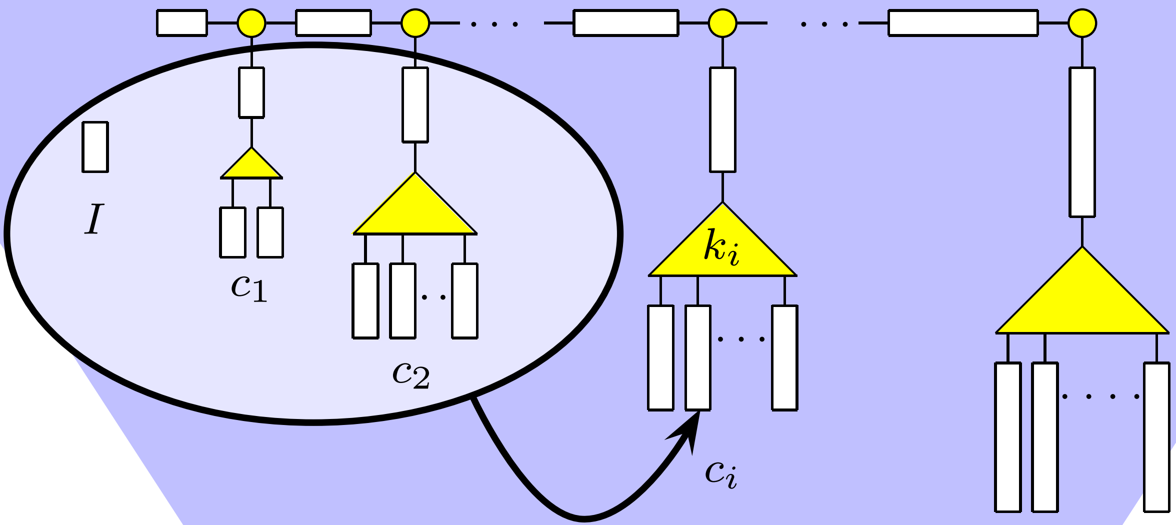
In total: A single binary merge tree

Operations — DeleteMin



- If A_1 is empty, call Fill(v_1)
- Search I and A_1 for minimum element

Operations — Insert

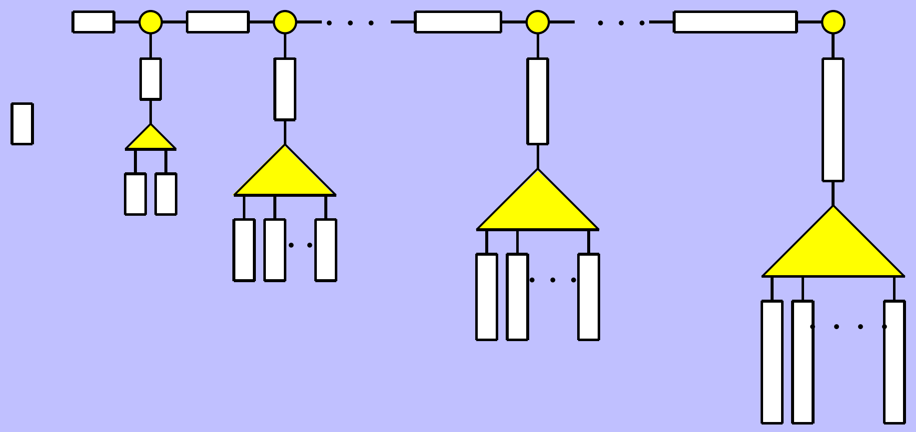


- Insert in I
- If I overflows, call Sweep(i) for first i where $c_i \leq k_i$

Sweep $\approx$ addition of one to number $c_1 c_2 .. c_i .. c_{\max}$

$$s_i = s_1 + \sum_{j=1}^{i-1} k_j s_j$$

Analysis



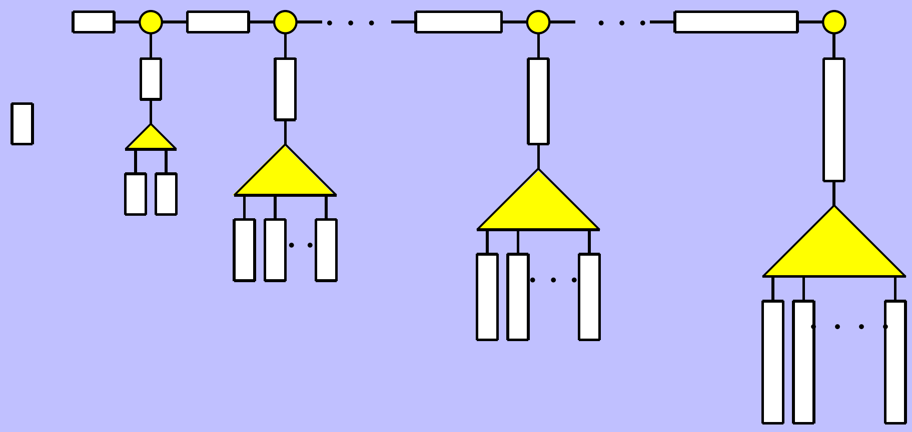
We can prove:

- Number N of insertions performed: $s_{i_{\max}} \leq N$
- Number of I/Os per Insert for link i: $O\left(\frac{1}{B}\log_{M/B} s_i\right)$
- By the doubly-exponentially growth of s_i, the total number of I/Os per Insert is

$$O\left(\sum_{k=0}^{\infty} \frac{1}{B}\log_{M/B} N^{(3/4)^k}\right) = O\left(\frac{1}{B}\log_{M/B} N\right)$$

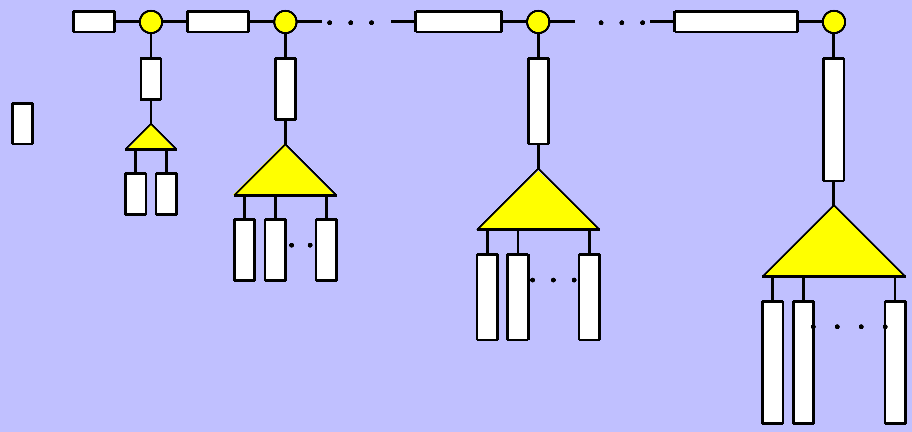
- DeleteMin is amortized for free

Profile Adaptive



- Modify Insert to apply Sweep(i) for
  - first i where $c_i \leq k_i$, or
  - link i is at most half-full, i.e. there exists S_{ij_1} and S_{ij_2} that can be merged
- $O\left(\frac{1}{B}\log_{M/B}\frac{N_i}{B}\right)$ I/Os

Conclusions



- Funnel heap - a cache-oblivious priority queue
- Profile adaptive
- Requires tall cache assumption
  (necessary requirement Brodal, Fagerberg 2002)

Open problem

- Worst-case bounds