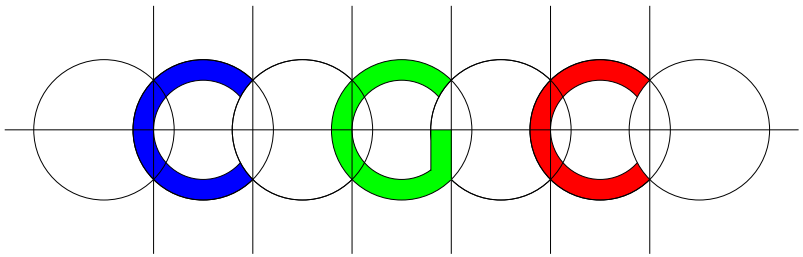
Simple Randomized Mresgerot

Jeff Vitter

Duke University and

INRIA Sophia-Antipolis

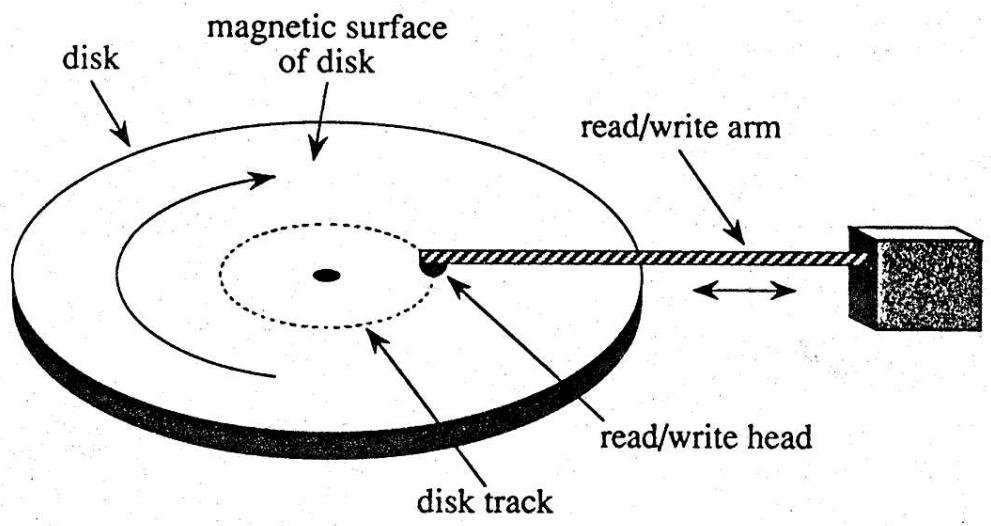


Center for Geometric Computing

http://www.cs.duke.edu/CGC/

June 1999

Magnetic Disk Drives as Secondary Memory



- ☆ I/O Crisis! 10^6 times slower access than registers.
- ☆ Time for rotation $\approx$ Time for seek.
- ☆ Amortize search time by large block transfer so that
  Time for rotation $\approx$ Time for seek $\approx$ Time to transfer data.
- ☆ Parallel disks.

Trends

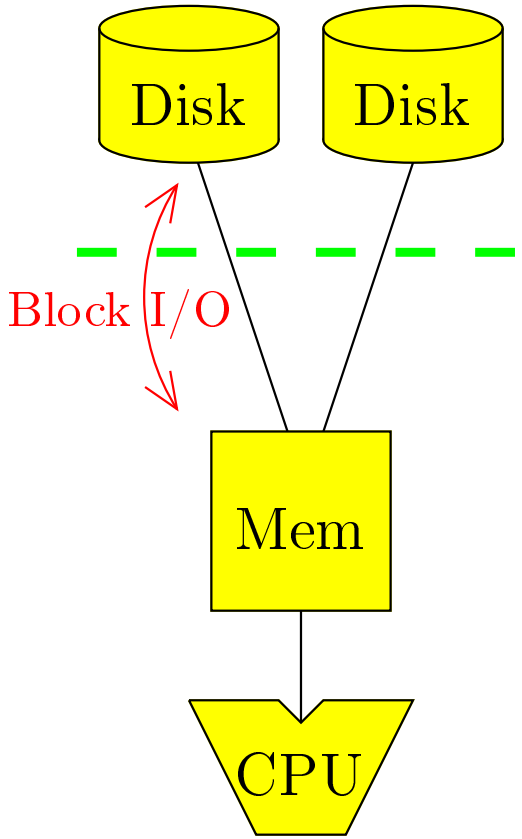
[Dahlin 96]:

Parameter	Yearly Improvement Rate
Disk Latency	10%
Disk Bandwidth	20%
Processor Speed	55%
RAM Bandwidth	40%
RAM Capacity/Cost	45%

- ☆ Performance gap is increasing.
- ☆ RAM Capacity/Cost doubling every 22–23 months, but users doubling data storage every 5 months. [AUS98]
- ☆ Users frequently reprocess data in entirety. [AUS98]
- ☆ I/O Bottleneck.

Parallel Disk Model

[Aggarwal & Vitter 88], [Vitter & Shriver 90, 94]



N = problem data size.

M = size of internal memory.

B = size of disk block.

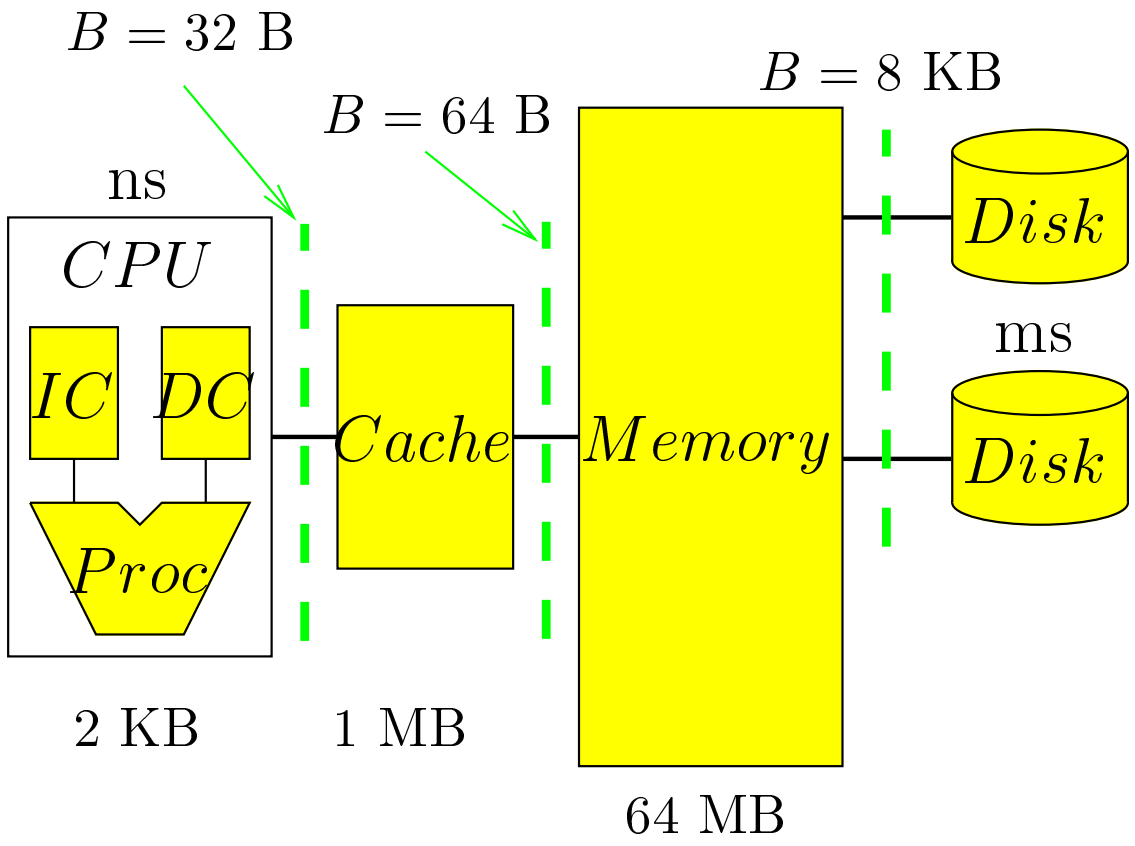
D = number of independent disks.

P = number of CPUs.

Notational convenience (in units of blocks):

$n = \frac{N}{B}, \quad m = \frac{M}{B}.$

A "Real" Machine



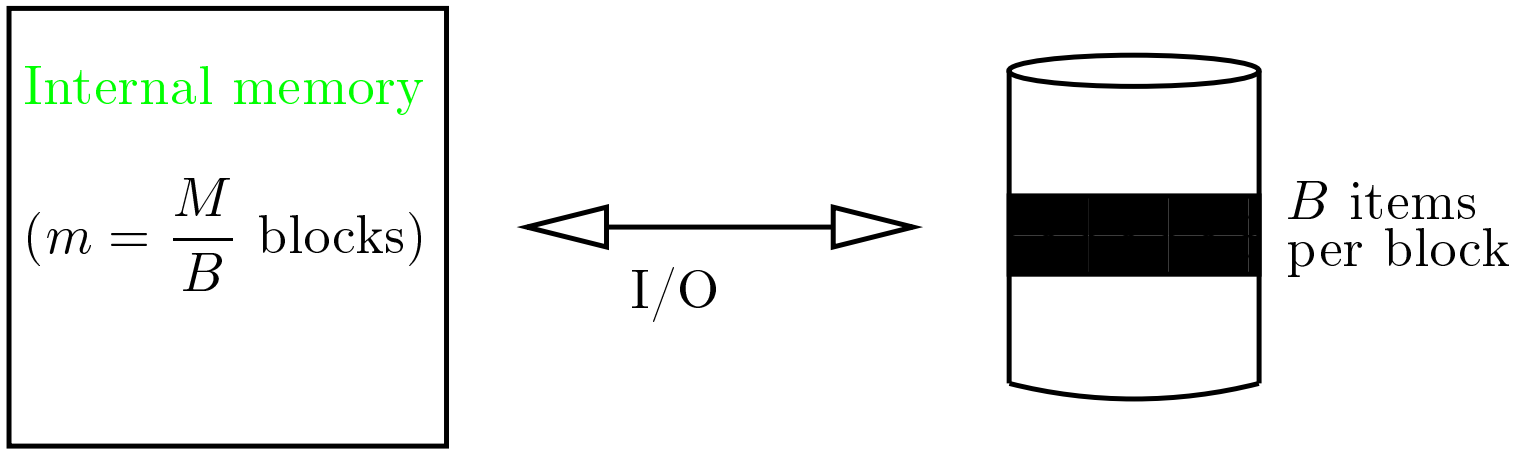
Outline of Talk

- ☆ Single Disk Model, $D = 1$.
  - Lower Bound on sorting.
  - Single Disk Mergesort.
- ☆ Parallel Disk Model, $D > 1$.
  - Difficulties of Parallelization.
  - Previous Approaches.
  - Simple Randomized Mergesort (SRM) and its analysis.
- ☆ Implementation of SRM.
- ☆ Dynamic Memory Model.
  - Lower Bound on sorting.
  - Optimal memory-adaptive mergesort
  - Analysis with potential functions
  - Related results.

Fundamental Bounds

- ☆ Batched problems [AV88], [VS90,VS94]:
  - Scanning (touch problem): $\Theta\left(\frac{N}{DB}\right) = \Theta\left(\frac{n}{D}\right)$
  - Sorting:
    $$\Theta\left(\frac{N}{DB}\frac{\log\frac{N}{B}}{\log\frac{M}{B}}\right) = \Theta\left(\frac{N}{DB}\log_{M/B}\frac{N}{B}\right) = \Theta\left(\frac{n}{D}\log_m n\right)$$
  - Permuting: $\Theta\left(\min\left\{\frac{N}{D}, \frac{n}{D}\log_m n\right\}\right)$
- ☆ Sorting is key subroutine for many problems [CGGTVV95], [AKL95], ...
  - Graph problems $\asymp$ Permutation
  - Computational Geometry $\asymp$ Sorting
- ☆ Online problems:
  - Searching: $\Theta(\log_B N + z)$

I/O Lower Bound for Sorting [AgV88]



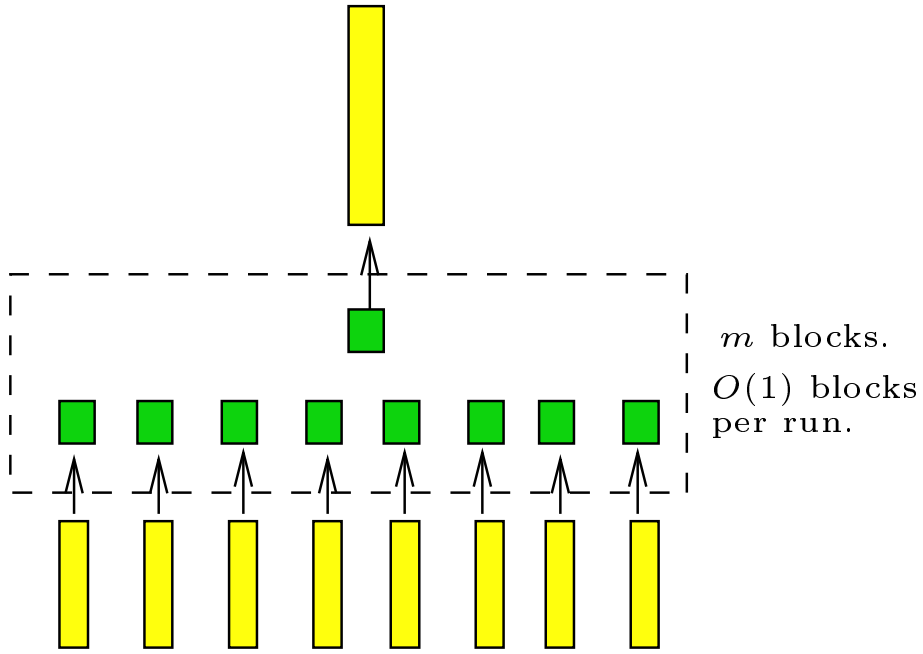
For most m, n, B values, a routing argument gives the bound below.
A comparison model argument works for all m, n, B values.

$$\#\text{ possible outcomes to comparisons per I/O} = \begin{cases} B! \times \binom{M}{B} & \text{reading unread block.} \\ \binom{M}{B} & \text{reading dirty block.} \end{cases}$$

$$(B!)^{N/B} \binom{M}{B}^T \geq N! \implies T = \Omega(n \log_m n) = \Omega\left(n \frac{\log n}{\log m}\right).$$

For $D > 1$, divide lower bound by D.

Mergesort with a Single Disk, $D = 1$



Merge $\Theta(m)$ runs together at a time.

☆ Merge $R = \Theta(m)$ or $\Theta(\sqrt{m})$ runs at a time

$\Longrightarrow \lceil \log_m n_0 \rceil$ merge passes

☆ Each merge pass takes $O(n)$ I/Os

$\Longrightarrow$ Total of $O(n \log_m n)$ I/Os for sorting (optimal).

Disk Striping: $D = 5$ disks, block size $B = 2$

☆ Layout for data:

	$\mathcal{D}_0$	$\mathcal{D}_1$	$\mathcal{D}_2$	$\mathcal{D}_3$	$\mathcal{D}_4$
stripe 0	0 1	2 3	4 5	6 7	8 9
stripe 1	10 11	12 13	14 15	16 17	18 19
stripe 2	20 21	22 23	24 25	26 27	28 29
stripe 3	30 31	32 33	34 35	36 37	38 39

☆ Disk striping involves using the D disks in lock step

$\Longrightarrow$ Substitute $B \leftarrow DB$ in single-disk algorithm.

☆ Single-disk I/O bound $\Theta\left(\frac{N}{B}\log_{M/B}\frac{N}{B}\right)$ becomes

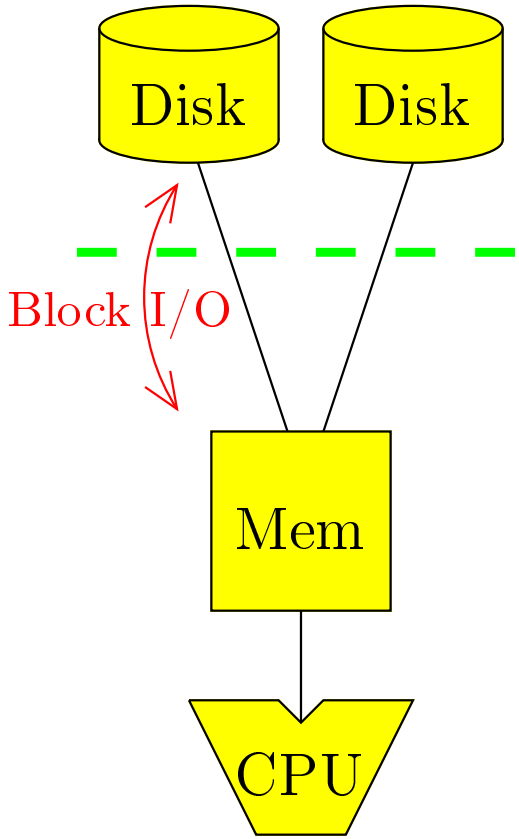
$\Theta\left(\frac{N}{DB}\log_{M/DB}\frac{N}{DB}\right) = \Theta\left(\frac{n}{D}\log_{m/D}\frac{n}{D}\right)$

☆ Ratio with optimal bound $\Theta\left(\frac{n}{D}\log_m n\right)$ is $\approx \frac{\log m}{\log \frac{m}{D}}$ when $D \approx m$.

☆ To get an optimal sorting algorithm, *use disks independently!*

Parallel Disk Model

[Aggarwal & Vitter 88], [Vitter & Shriver 90, 94]



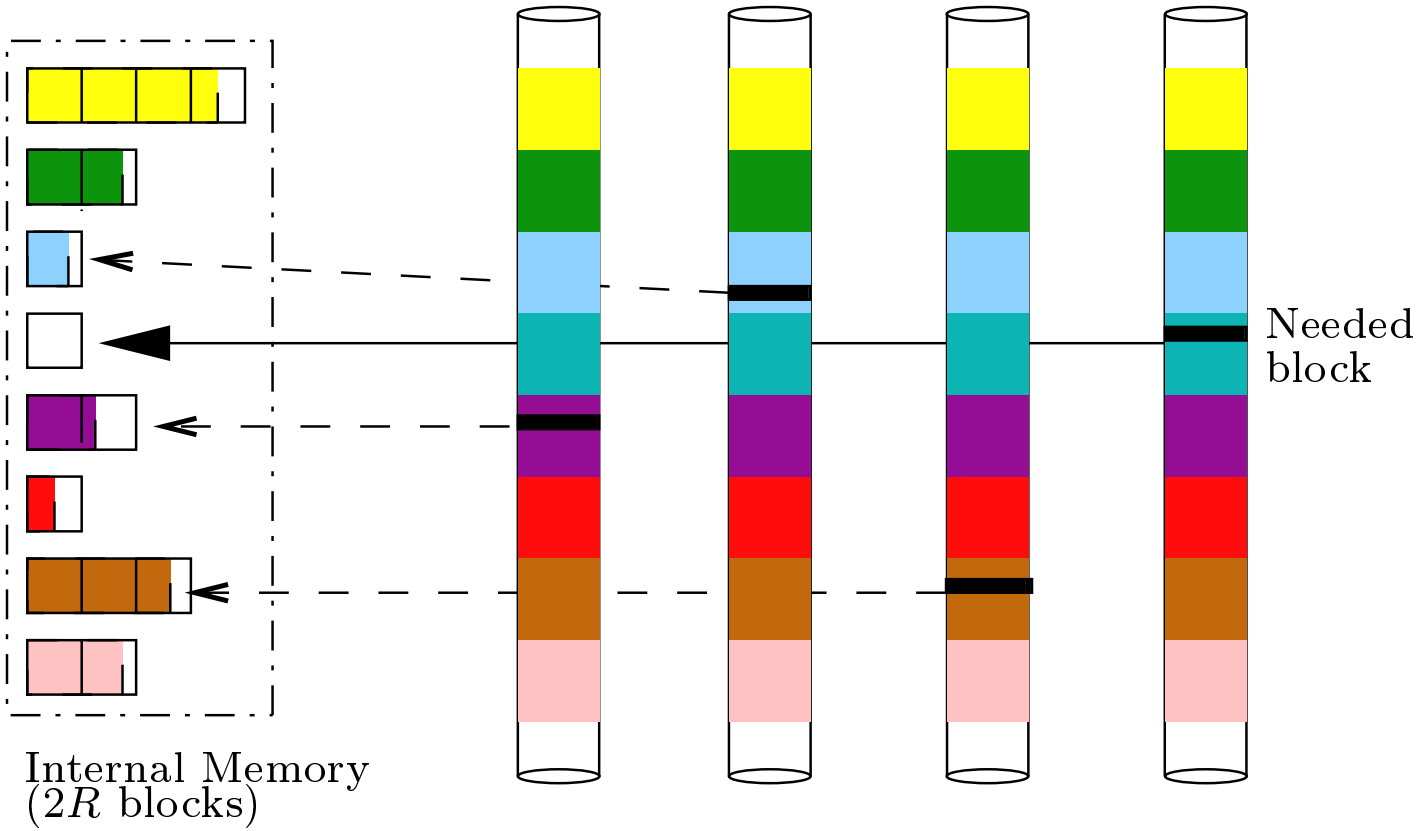
- ⋆ D = number of independent disks.
- ⋆ Goal:
  Design computation to transfer $\Theta(D)$ blocks in each I/O (one per disk).
- ⋆ Optimal Sorting: $\Theta\left(\frac{n}{D}\log_m n\right)$ I/Os.
- ⋆ Desired Mergesort Performance:

$$\begin{aligned} \#\text{ merge passes} &= \Theta\left(\log_m n\right); \\ \#\text{I/Os per merge pass} &= \Theta\left(\frac{n}{D}\right). \end{aligned}$$

Some Optimal Parallel Disk Sorting Algorithms

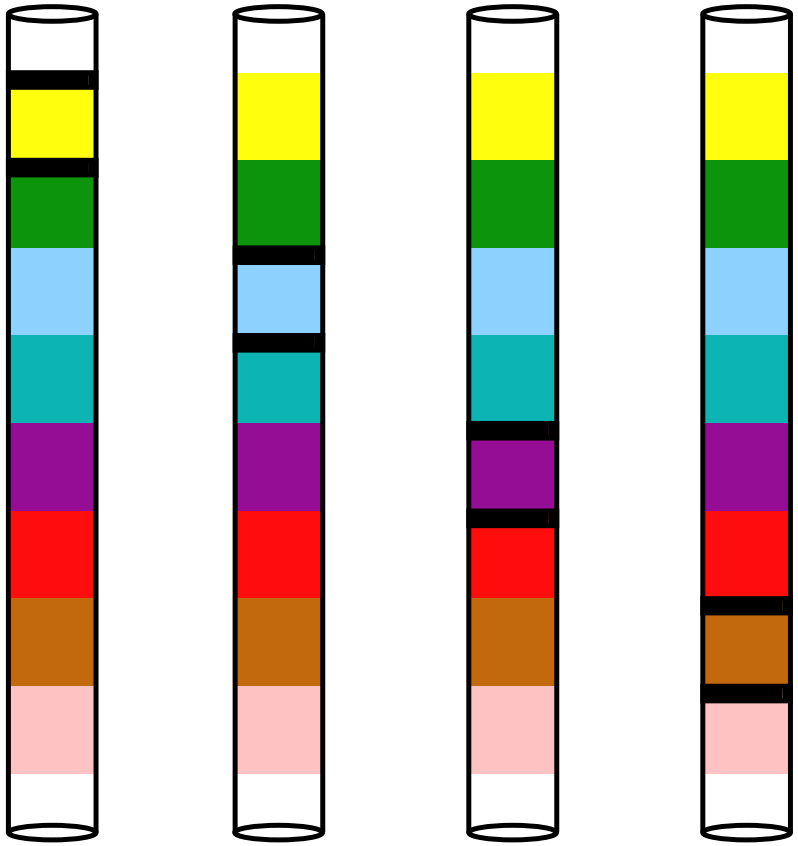
- ☆ Distribution Sort [Vitter& Shriver, 90]
- ☆ Greed Sort [Nodine& Vitter, 91]
- ☆ Balance Sort [Nodine & Vitter, 93]
- ☆ Share Sort [Aggarwal & Plaxton, 94]
- ☆ All are somewhat complicated, most are not very practical.
- ☆ Wanted:
  A provably efficient and simple
  parallelization of single-disk mergesort.

Difficulty of merging $R = \Theta(m)$ runs on D disks



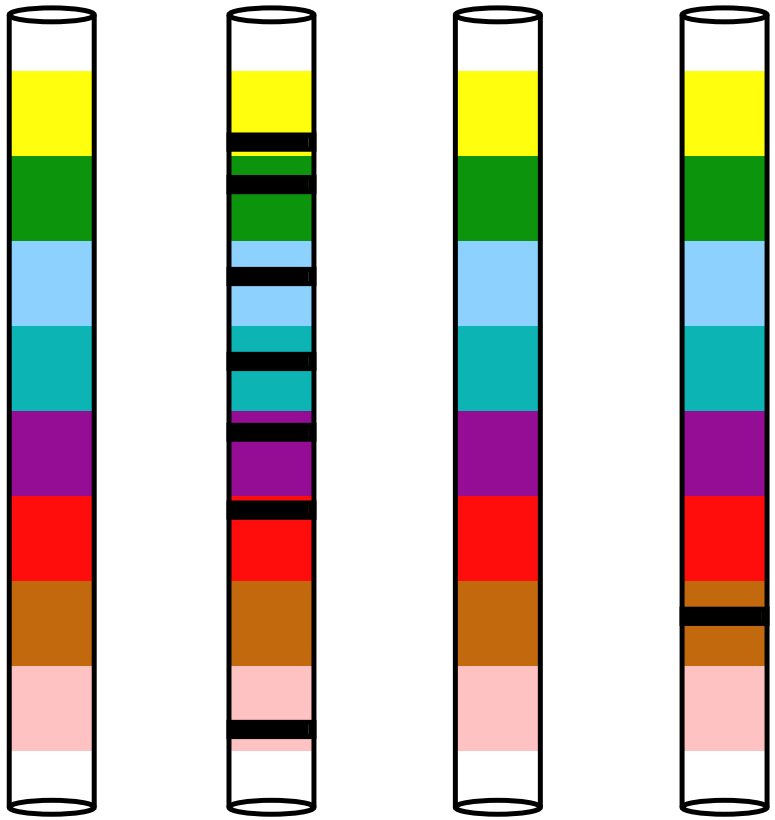
- ☆ Each read: One needed block, $D-1$ prefetched blocks for future use.
- ☆ Bounded memory size inhibits prefetching.
- ☆ Performance critical question: How many reads required to bring in the "next" $R = \Theta(m)$ blocks? Ideally, R/D.

Staggered Layout: $R = 8$, $D = 4$.



- ☆ Each run is striped, but the starting disk of the runs are staggered.
- ☆ Initially, # I/Os need to read in the next R leading blocks
  = Max Occupancy (of leading blocks) on any disk
  = R/D, as desired.
- ☆ But balance can quickly deteriorate.

$R = 8$, $D = 4$: Imbalance can set in.



Maximum Occupancy (of leading blocks on disks) is $7 \gg R/D$

$\Longrightarrow$ # I/Os needed to load 8 blocks $= 7$.

Gilbreath Principle

☆ Can achieve perfect balance for merging two runs, $R = 2$:

Run 1:

A B C D

E F G H

I J K L

. . .

Run 2:

D C B A (striped in reverse order)

H G F E

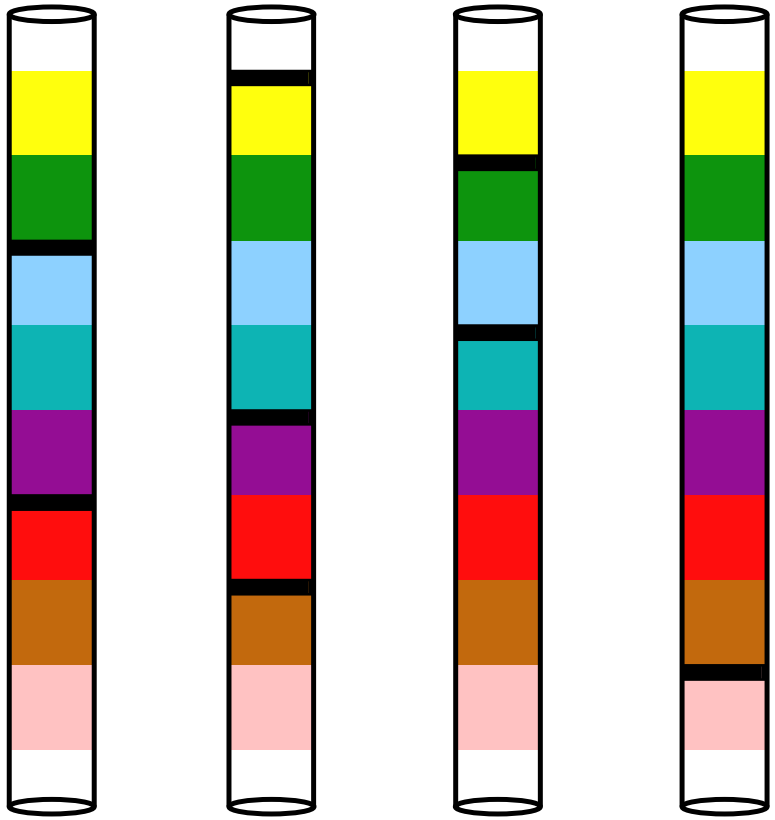
L K J I

. . .

☆ Reduces necessary buffer space by half.

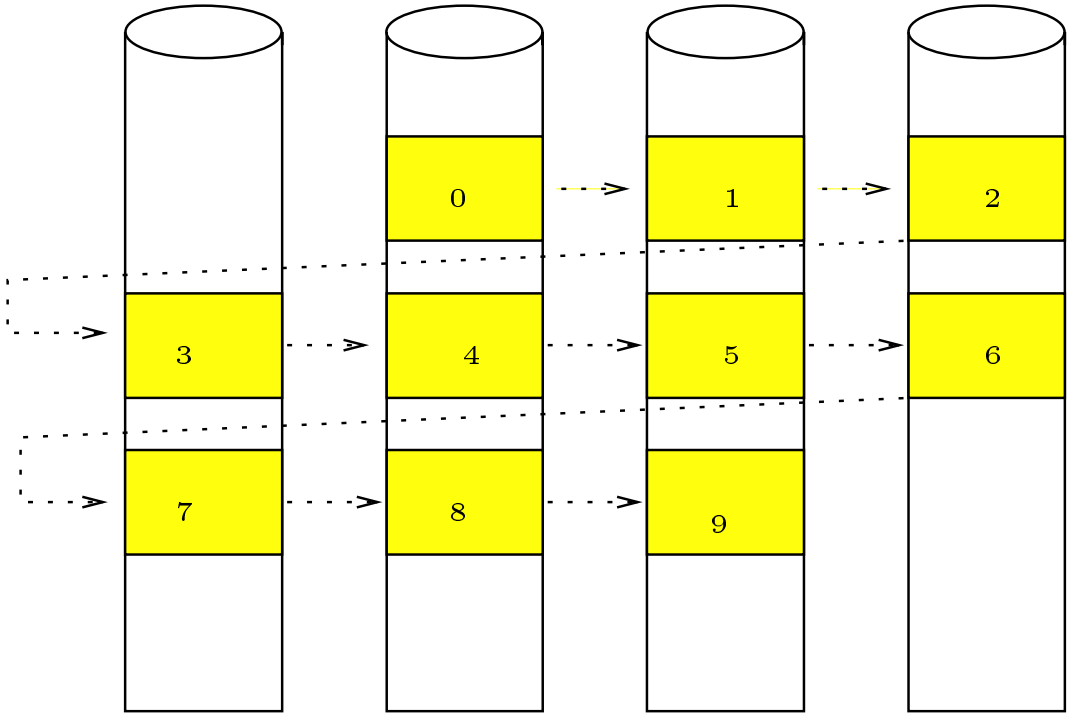
☆ Cannot be generalized to $R > 2$.

Simple Randomized Mergesort (SRM) [BGV 97]



- ☆ Each run is striped starting at a randomly chosen disk.
- ☆ *At any time*, the disk containing the leading block of any run is uniformly random.

Forecasting Information in an SRM run



Forecasting Information in the input blocks of a run:

Implanted in block i is the largest key of block $i + D - 1$.

When block i is in memory, the time at which the merge needs block $i + D$ can be predicted.

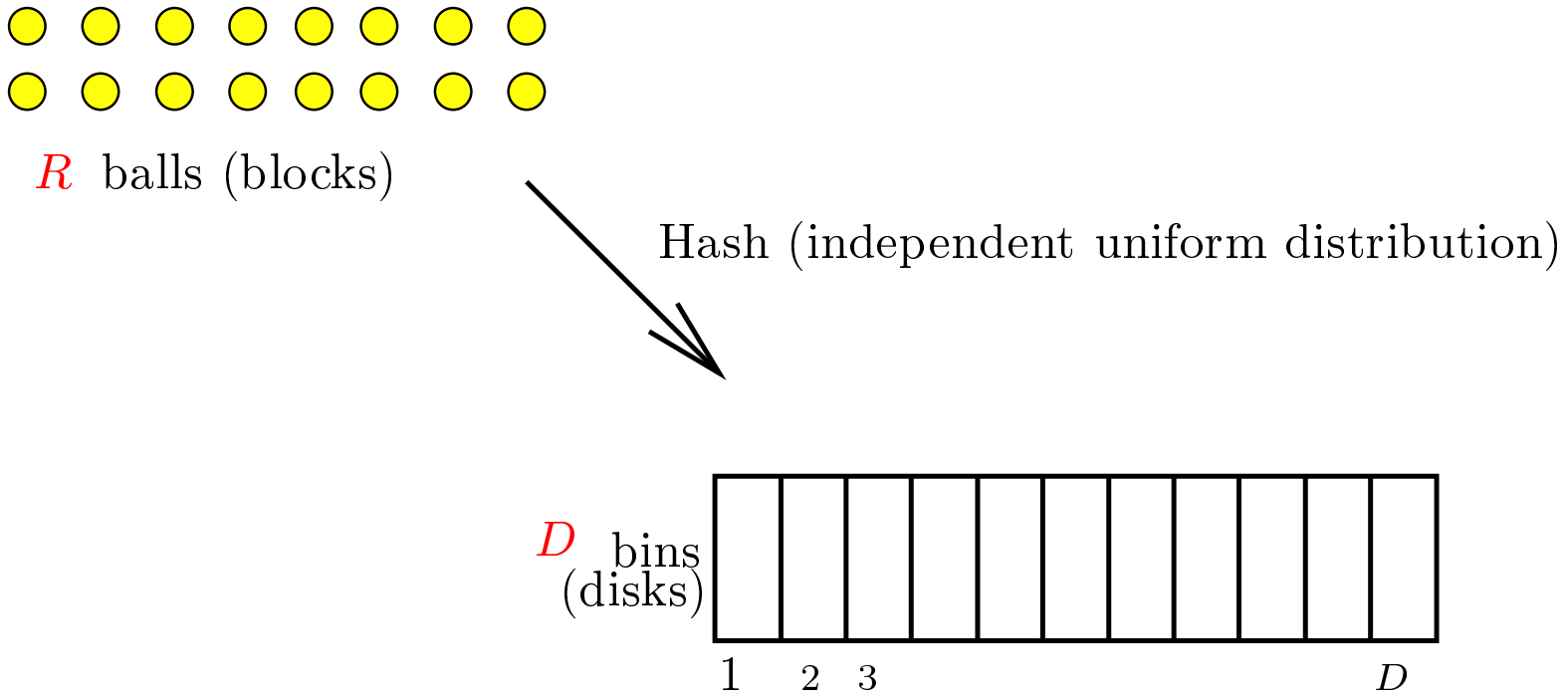
SRM: *Forecast and Flush (FF)*

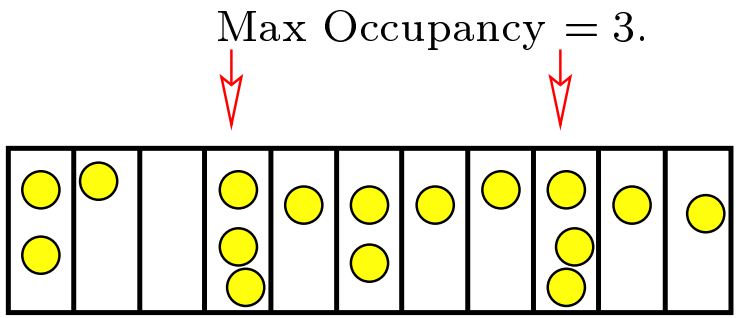
- ☆ Writes occur at full D-disk parallelism.
- ☆ SRM implants forecasting info in each input block.
- ☆ FF buffer management:
  - If # Free Blocks is $D - f$, Flush f "largest" blocks.
  - Forecast the "smallest" block from each disk.
  - Read in the "smallest" block from each disk.
- ☆ Analysis: If $E[\text{MaxOcc}_{SRM}]$ is the average Max Occupancy of the next R blocks,

$$E[\#\text{reads}_{SRM}] = E[\text{MaxOcc}_{SRM}] \times \frac{n}{R} \times \lceil \log_R(n/m) \rceil$$

Classical Maximum Bucket Occupancy

$$E[\text{Classical Max Occupancy}] \sim \begin{cases} \dfrac{\ln D}{\ln \ln D} \cdot \dfrac{R}{D} & \text{if } \dfrac{R}{D} = 1 \\ c \cdot \dfrac{R}{D} & \text{if } \dfrac{R}{D} = \Theta(\log D) \\ \dfrac{R}{D} & \text{if } \dfrac{R}{D} \gg \log D \end{cases}$$


R balls (blocks)
Hash (independent uniform distribution)
D bins
(disks)
1
2
3
D


Max Occupancy = 3.

Occupancies with Dependency

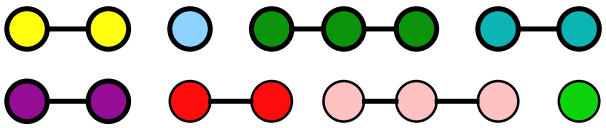
Conjecture: Let n_i =size of ith chain.

$E[\text{MaxOcc}_d(n_1, n_2, \ldots, n_{R'})]$

$\leq \quad E[\text{MaxOcc}_d(n_1, n_2, \ldots, n_{R'-1}, n_{R'} - 1, 1)]$

$\leq \quad E[\text{Classical Max Occupancy}].$

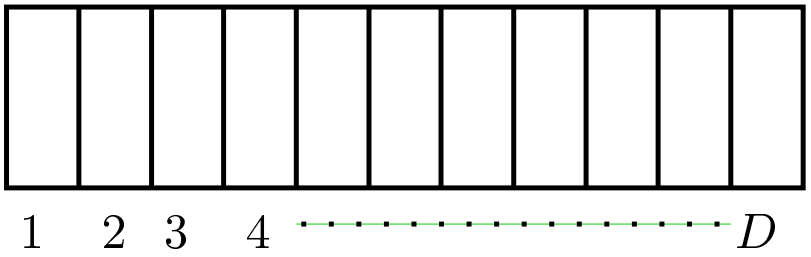
We can prove the (*harder !?!?*) conjecture if the balls of each chain are in random order rather than consecutive.

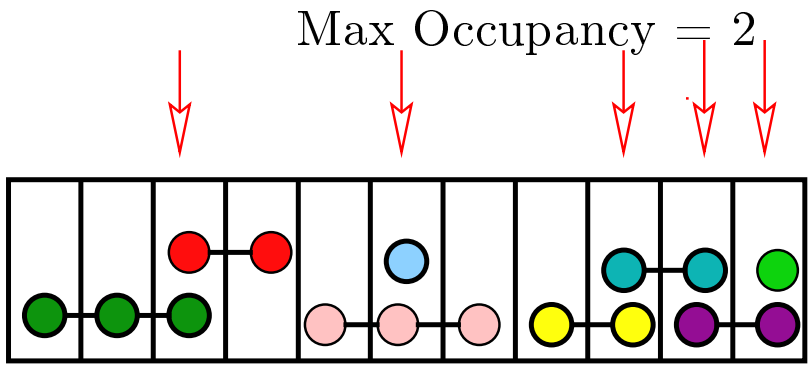


R' chains
containing R balls

Starting bin of each chain
is uniformly random.

D bins
(disks)




Max Occupancy = 2

I/O Performance of SRM, $R = m/2$

Asymptotically,

$$E[\#\text{reads}_{SRM}] \leq \begin{cases} \dfrac{\ln D}{k \ln \ln D} \cdot \dfrac{n}{D} \log_m n & \text{if } \dfrac{m}{2D} \approx k. \\ c \cdot \dfrac{n}{D} \log_m n & \text{if } \dfrac{m}{2D} = \Theta(\log D). \\ \dfrac{n}{D} \log_m n & \text{if } \dfrac{m}{2D} \gg \log D. \end{cases}$$

Simulation of I/O performance ratio $\dfrac{\text{IO}_{\text{SRM}}}{\text{IO}_{\text{DSM}}}$ for $m \approx (2k+4)D$:

SRM is better than striping!

	$D = 5$	$D = 10$	$D = 50$
$k = 5$	0.56	0.47	0.37
$k = 10$	0.61	0.52	0.40
$k = 50$	0.71	0.63	0.51

Other Aspects

- ☆ Probabilistic analysis required getting around dependancies.
- ☆ Same technique and analysis for a simple randomized (multi-way) distribution. Application in parallel disk distribution sort.
- ☆ Forecast and Flush technique has been used in a competitive parallel prefetching algorithm for certain request sequences, and may have other applications.
- ☆ Algorithm can be modified to be theoretically optimal for all parameter values using technique of redundant writing [Sanders, Egner, Korst 99].

Implementation of SRM

- ☆ Implanting forecasting information.
- ☆ FF buffer management:
  - If # Free Blocks is $D - f$, Flush f "largest" blocks.
  - Forecast the "smallest" block from each disk.
  - Read in the "smallest" block from each disk.
- ☆ As stated, Forecasting requires D priority queues each containing R keys at any time.
- ☆ Flushing requires maintaining order among prefetched blocks in memory.

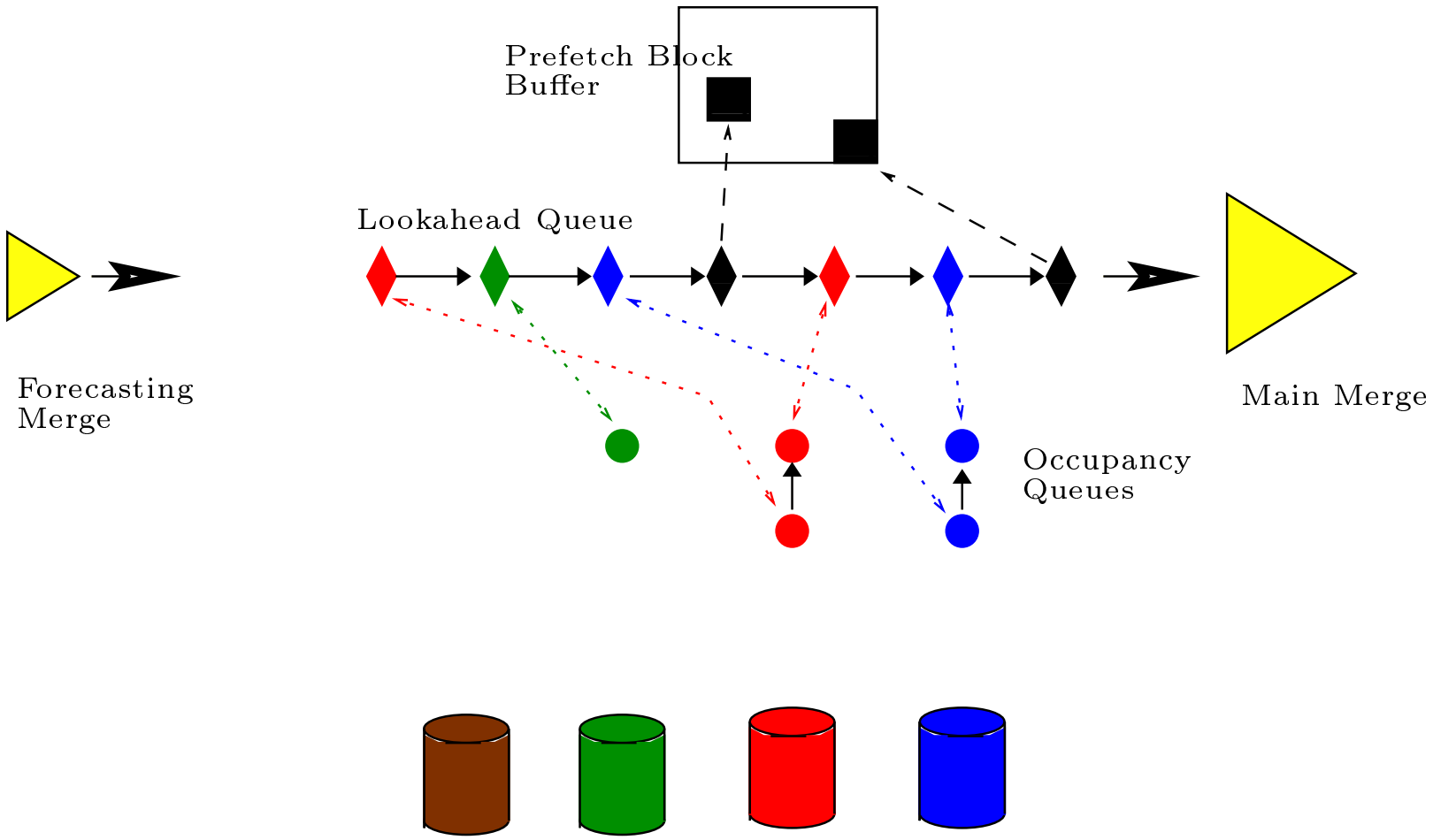
Simplifying the implementation of FF

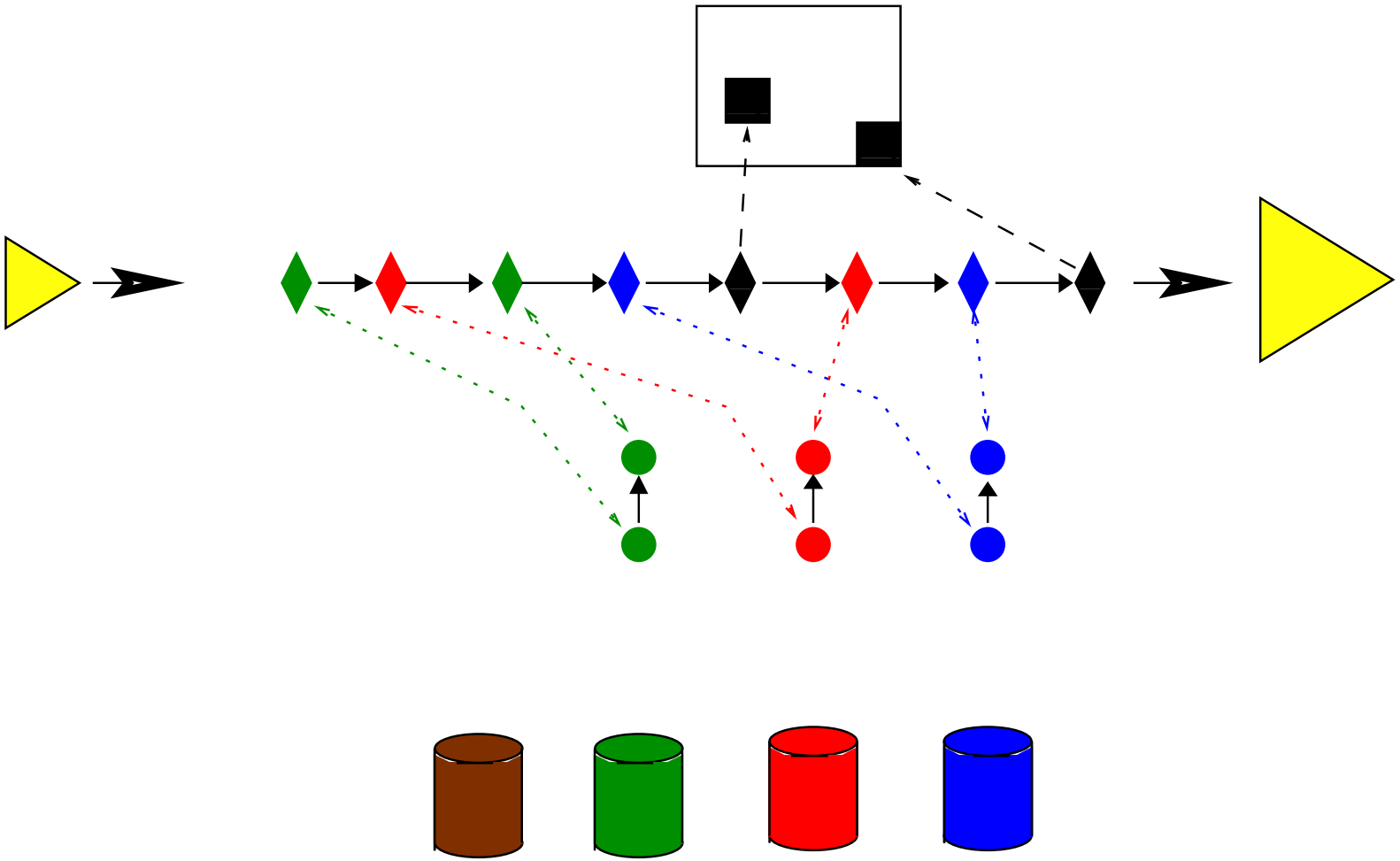
- ☆ If I is item size,

$$\text{Size of forecasting information} = \frac{1}{BI} \times \text{Input file size.}$$

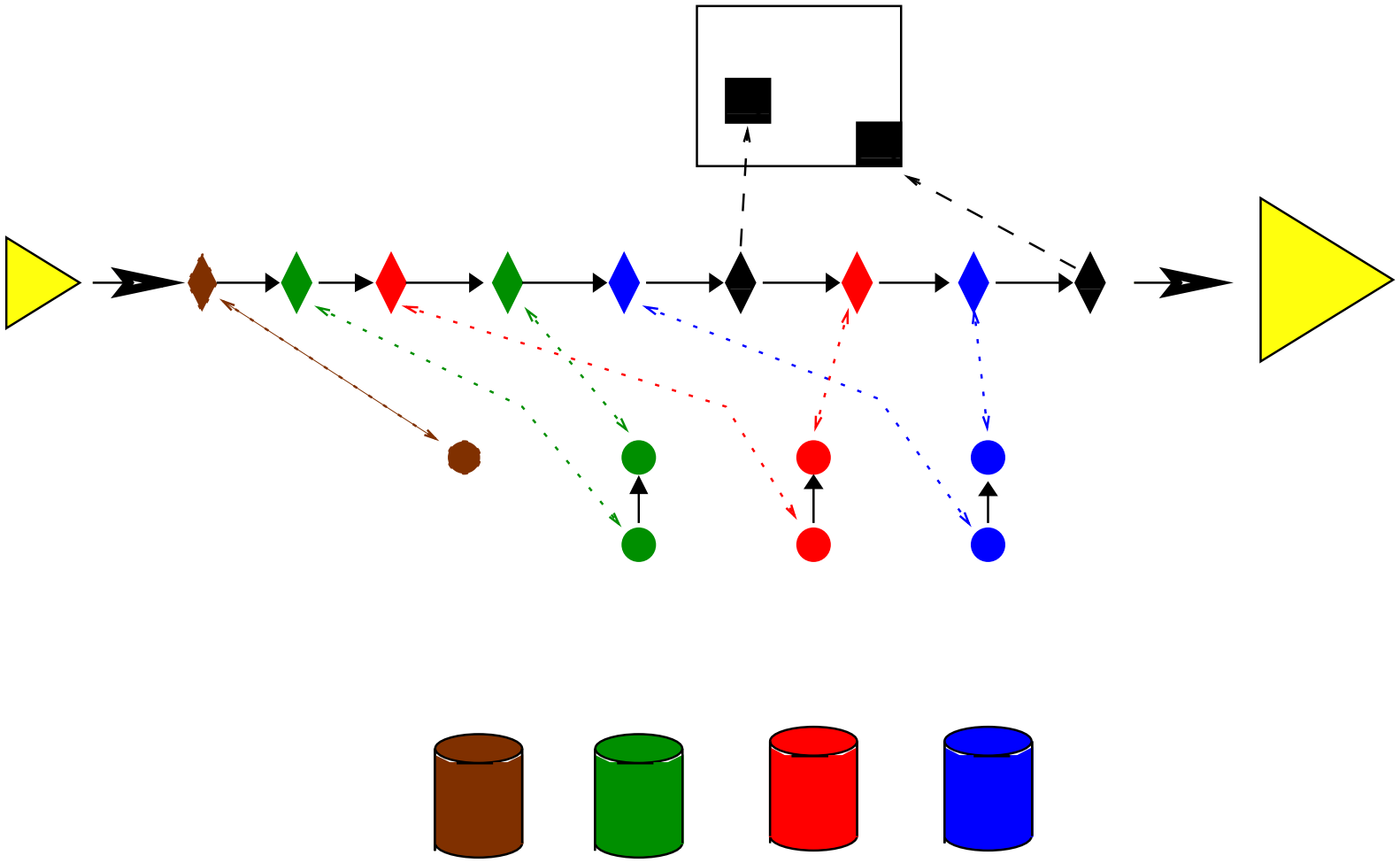
- ☆ Our approach
  - *Don't implant* forecasting information in run blocks.
  - Store forecasting keys of a run in a separate file.
  - Using the forecasting keys during SRM requires a (much) smaller-scale R-way auxiliary merge of forecasting files.
- ☆ Requires only one priority queue with R keys in it.
- ☆ Automatically orders prefetched blocks and flushing is easy.

Simplified Implementation of FF


Prefetch Block
Buffer
Lookahead Queue
Forecasting
Merge
Main Merge
Occupancy
Queues

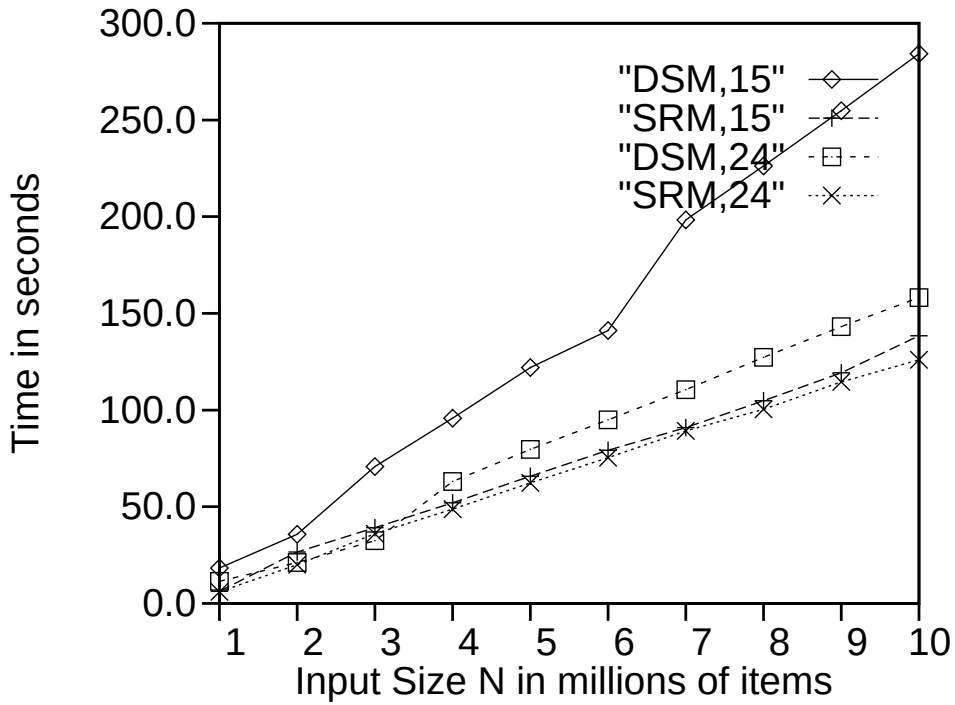


Simplified Implementation of FF



Practical Performance

- ☆ SRM outperforms DSM by 25–50% in running time.
- ☆ Merge passes of SRM are slower, but fewer than those of DSM.
- ☆ Overhead ratio for SRM is very close to 1 in practice.



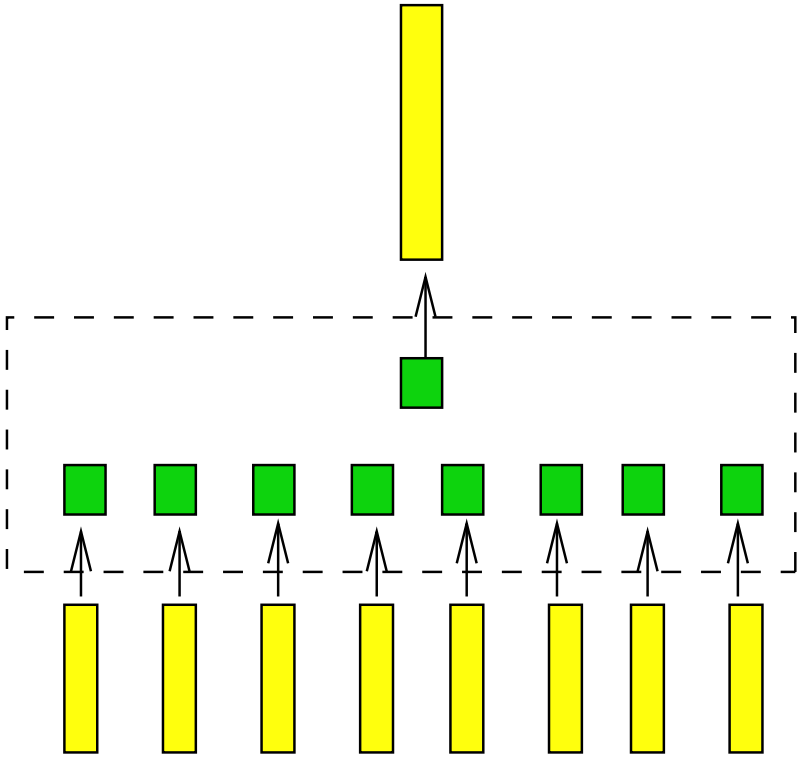
Other Aspects

- ☆ Implementation of SRM and DSM was in the TPIE system; TPIE's functionality had to be extended to suport parallel disk operations.
- ☆ Several internal memory and other optimizations were programmed, and play a significant role in performance improvement.
- ☆ Parallel I/O was performed using the *mmb* memory-map system developed at Duke.

The Need for Memory-Adaptive EM Algorithms.

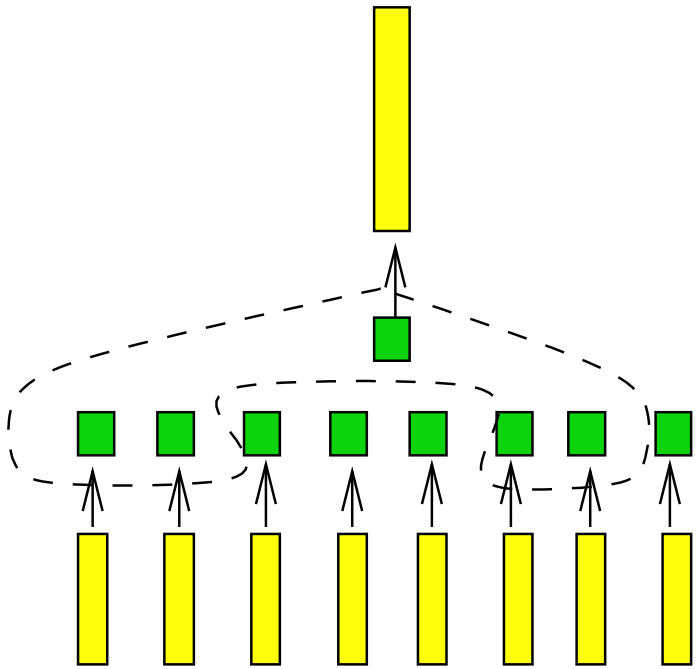
- ☆ Traditional EM algorithms assume *fixed memory* allocation.
- ☆ Problem:
  - OS/DBMS can *dynamically* change memory allocation.
  - EM applications exhibit *thrashing*.
- ☆ Solution:
  EM algorithms that *adapt online* to memory fluctuations.
- ☆ All prior work has been exclusively empirical:
  - Memory-Adaptive Hash Join (Zeller& Gray, Pang et al.)
  - Pang et al., 1995: Non-optimal memory-adaptive sort.
  - Zhang and Larson, 1997: Memory-adaptive sort, works only for very restricted kinds of fluctuations.

Why Traditional EM Algorithms Thrash.



Merging 8 runs using 9 internal memory blocks

Why Traditional EM Algorithms Thrash.



Merging 8 runs using 5 internal memory blocks:
Leading blocks of 4 runs are out of memory

- ☆ If m drops to less than 8 but merge-order remains 8, worst case cost is one I/O per element output by merge.
- ☆ Solution: Reorganize computation; ie, change merge-order in response to change in m.

Dynamic Memory Environment

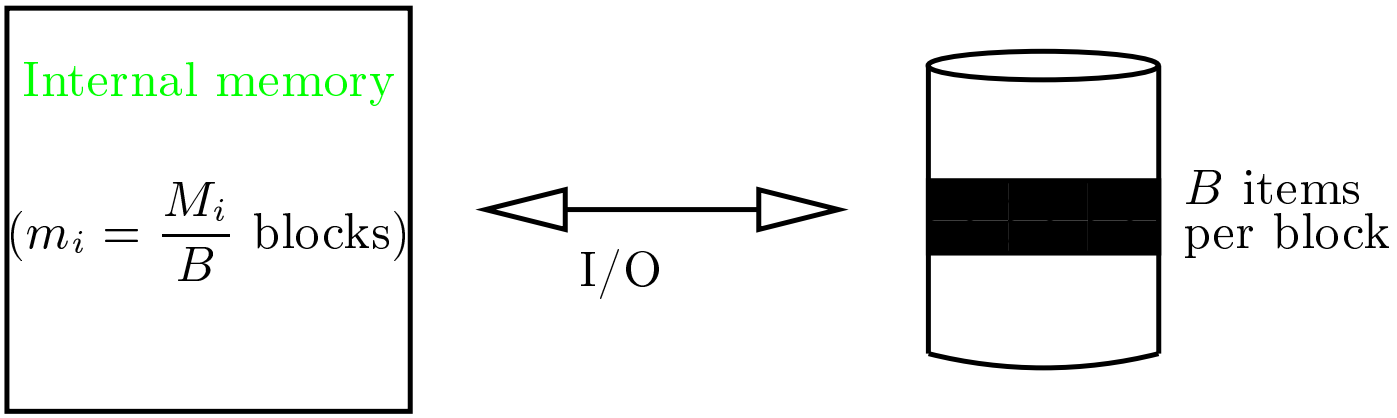
- ☆ EM algorithm is allocated m memory blocks by the OS/DBMS for an unspecified amount of time.
- ☆ When OS/DBMS wants to change the allocation of m, it first allows EM algorithm to carry out m I/Os ("Reaction time"). Then it changes m.
- ☆ We use a simplified "constant factor approximation" of this model.

Simple Model for Memory-Adaptive EM Algorithms

- ☆ EM algorithm $\mathcal{A}$ is allocated memory in an allocation sequence $\sigma = m_1, m_2, m_3, \ldots$ of *allocation phases.*
- ☆ OS/DBMS determines σ in an online adversarial manner.
- ☆ ith phase: Algorithm owns m_i blocks of memory for $2m_i$ I/Os.
- ☆ EM algorithm must adapt to allocation sequence.
- ☆ Suppose that $\mathcal{A}$ solves problem $\mathcal{P}$ during σ.
- ☆ $\mathcal{A}$ is dynamically optimal for $\mathcal{P}$ iff
  - No other algorithm $\mathcal{A}'$ can solve problem $\mathcal{P}$ more than a constant number of times during σ.

Dynamic Memory Lower Bound for Sorting

ith phase:



Use comparison model:

$$\#\text{ possible outcomes to comparisons per I/O} = \begin{cases} B! \times \binom{M_i}{B} & \text{reading unread block.} \\ \binom{M_i}{B} & \text{reading dirty block.} \end{cases}$$

$$(B!)^{N/B} \prod_i \binom{M_i}{B}^{2m_i} \geq N! \implies \sum_i 2m_i \log m_i = \Omega(n \log n).$$

Resource Consumption of Sorting

Sorting algorithm completes in ℓ phases

$\Longrightarrow \quad \sum_{i=1}^{\ell} 2m_i \log m_i = \Omega(n \log n)$.

- ☆ Resource Consumption of an I/O in phase i is

  $\log m_i$

- ☆ Algorithm is *dynamically optimal* iff

  Total Resource Consumption (RC)$= O(n \log n)$.

A Framework for Memory-Adaptive Mergesort

☆ Run Formation

- Phase i $\Longrightarrow$ Generate a run of length m_i blocks.
- Number of runs in $\mathcal{Q}$ is $n_0 \leq n$. (Very often, $n_0 \ll n$.)
- Total Resource Consumption

$$\begin{aligned} \mathrm{RC}_{\mathrm{run_formation}} &= O(\#\mathrm{I/Os} \times \text{Max cost of each I/O}) \\ &= O(n \log m_{\max}) \end{aligned}$$

☆ Merging Stage

- Memory-adaptive merging routine $\mathcal{M}$.
- Repeat: Merge R runs from $\mathcal{Q}$, append output run to $\mathcal{Q}$.

Resource Consumption Requirement for Merging

$$\begin{aligned} \mathrm{RC}_{\mathrm{sort}} &= O\left(\mathrm{RC}_{\mathrm{run_formation}} + \frac{\log n_0}{\log R}\mathrm{RC}_{\mathrm{pass}}\right) \\ &= O\left(n \log m_{\max} + \frac{\log n_0}{\log R}\mathrm{RC}_{\mathrm{pass}}\right) \end{aligned}$$

For dynamic optimality,

- ☆ $\mathrm{RC}_{\mathrm{pass}} = O(n \log R)$.
- ☆ $R = \Omega(m_{\max}^c)$.

Previous Approach (Pang et al.)

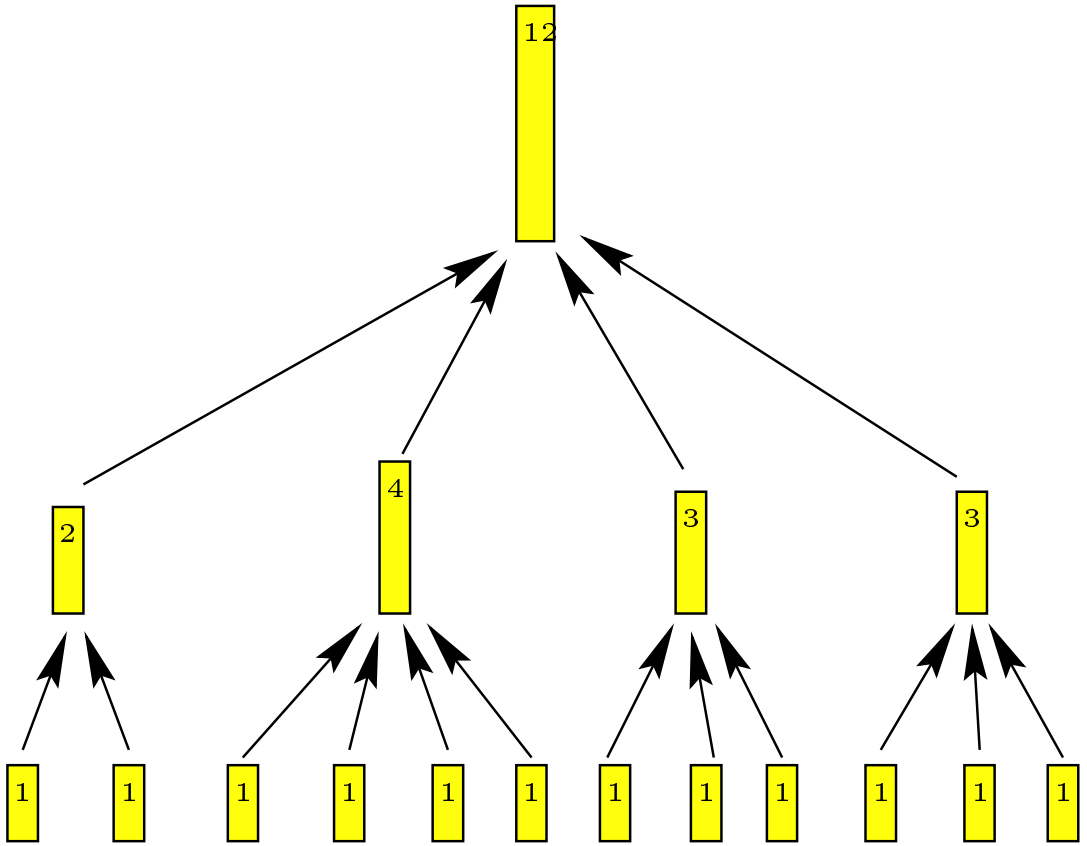
- ☆ Provably Inefficient approach.
- ☆ "Nemesis" allocation sequence can force binary merging, almost throughtout the execution, even when m_i is large.
- ☆ Thus, there is inefficient usage of resources.
- ☆ We show that

$$\mathrm{RC}_{\mathrm{pass}} = \Omega(n(\log R)(\log m_{\max})).$$

Our optimal approach

- ☆ 'Global" merge: Evolving "composition" of several individual merge operations of varying merge orders.
- ☆ Data structures and techniques to make each merge operation dynamically reorganizable and pre-emptible.
- ☆ Invariant:
  In an allocation phase of size m_i, process a merge operation with merge order $\hat{\rho} \geq \sqrt{m_i}$.
- ☆ Consequence of Invariant and techniques:
  Resource Consumption $\mathrm{RC}_{\mathrm{pass}}$ is always within a constant factor of *Merge Potential*.

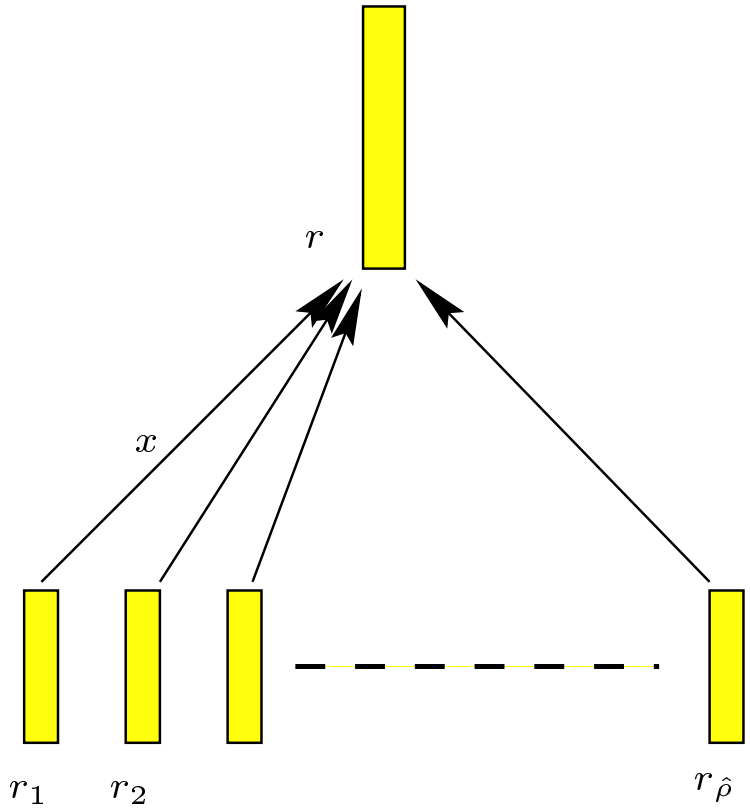
Merge Potential Ψ



Rank $S(r)$ of run r is # of input runs producing r.

- $\Psi(x) = \frac{1}{B} \lg \mathbf{S}(r)$, where r is run containing item x.
- Merge Potential $\Psi = \sum_x \Psi(x)$.
- $\Psi = 0$ (at the start) $\Longrightarrow$ $\Psi = n' \lg R$ (at the end)

Increasing Merge Potential



$$S(r) = \sum_i S(r_i)$$

☆ Item x "elevated" from run r_1 to run r $\Longrightarrow$ $\Delta\Psi(x) = \frac{1}{B} \lg \frac{\mathbf{S}(r)}{\mathbf{S}(r_1)}$.

☆ $\mathbf{S}(r_1) = \mathbf{S}(r_2) = \ldots \mathbf{S}(r_{\hat{\rho}}) \Longrightarrow \mathbf{S}(r) = \hat{\rho}\mathbf{S}(r_i)$

$\Longrightarrow$ $\Delta\Psi = \lg \hat{\rho}$, per output block.

Invariant

- ☆ Recall: $\mathcal{M}$ merges $R \approx m_{\max}$ runs containing a total of n' blocks.
- ☆ Invariant during a typical phase of size m_i:
  - By definition, $\Delta\text{RC} = 2m_i \log m_i$.
  - $\mathcal{M}$ ensures that $\approx m_i$ blocks of a $\hat{\rho} = \Omega(\sqrt{m_i})$-way merge operation are output, so that

$$\begin{aligned} \Delta\Psi &= \Omega(m_i \log m_i) \\ \Longrightarrow \Delta\text{RC} &= O(\Delta\Psi). \end{aligned}$$

- ☆ On termination, $\text{RC}_{\text{merge}} = O(\Psi) = O(n' \log R)$
  $\Longrightarrow \quad \text{RC}_{\text{pass}} = O(n \log R)$

Aspects

- ☆ Various external memory data structures and techniques are required for the scheme to work efficiently.
- ☆ Lower Bounds for problems related to sorting and matrix multiplication (and related problems).
- ☆ Sorting algorithm was used to get dynamically optimal algorithms for permuting, permutation networks, FFT.
- ☆ Dynamically Optimal memory-adaptive version of a buffer tree.
- ☆ Techniques applicable via sorting and buffer trees to many other applications.
- ☆ Dynamically optimal matrix multiplication algorithm.

Conclusions

- ☆ Practical benefits from using parallel disks independently.
- ☆ SRM outperforms DSM by 25–50% in running time.
- ☆ Merge passes of SRM are slower, but fewer than those of DSM.
- ☆ Overhead ratio for SRM is very close to 1 in practice.
- ☆ Work planned on distribution sort based on SRM ideas.
- ☆ Currently looking at analysis of cache issues in internal memory.
- ☆ Practical implementations must deal with changing memory allocation. Interesting theoretical questions.