

Sanders: Algorithms for Storage Servers

1



Algorithms for Scalable Storage Servers

Peter Sanders

MPII

with

Julie Cain, Roman Dementiev, Sebastian Egnér, David Hutchinson, Jan Korst,

Martin Skutella, Naveen Sivadasan, Jeff Vitter, Berthold Vöcking,

Nick Wormald

Overview

- Motivation
- Models
- Writing
- Reading
- Adapt to more realistic models
- Virtualization of inhomogeneous disks



An Experiment:

	Nov. 2003	March 2002	
Kilobyte	104 000	67 100	
Megabyte	468 000	256 000	
Gigabyte	853 000	582 000	
Terabyte	163 000	67 400	> “Quicksort” or “spanning tree”
Petabyte	21 900	8 740	> “Integer linear programming”, . . .

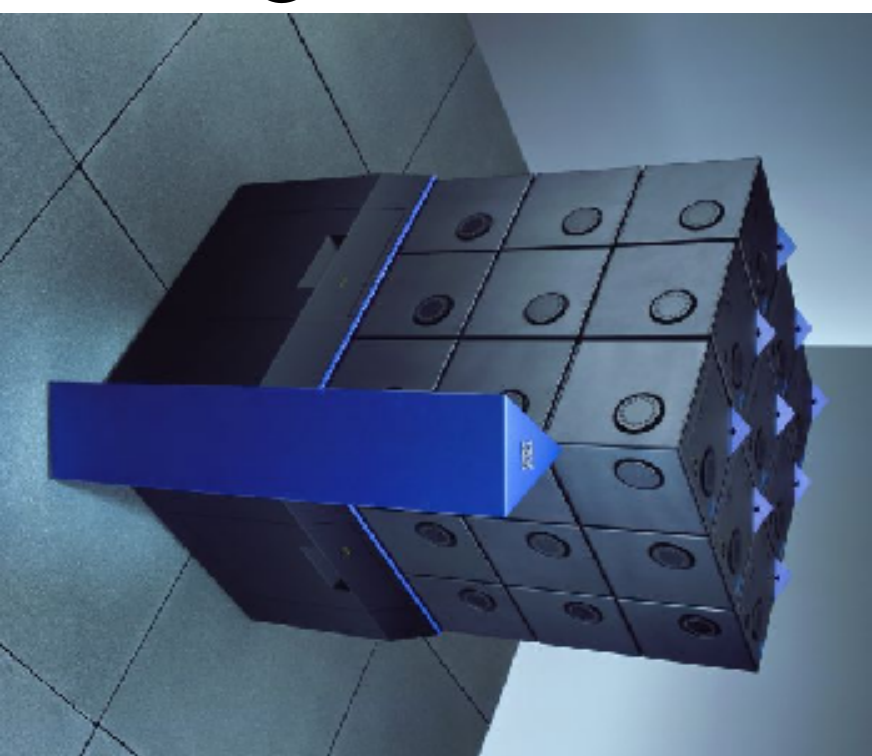
Reason: technological development +

Applications with unsatiable data hunger

(Video-On-Demand, Data-Mining, Web-Servers, Virtual Reality, GIS, . . .)

One Possible System

- up to $10 \times 10 \times 10 \times 12$ 2.5inch disks
- commodity parts
- no wiring
(cooling & power bus, capacitive coupling for communication)
- water cooled, very small floor space
- add arbitrary bricks; never remove them

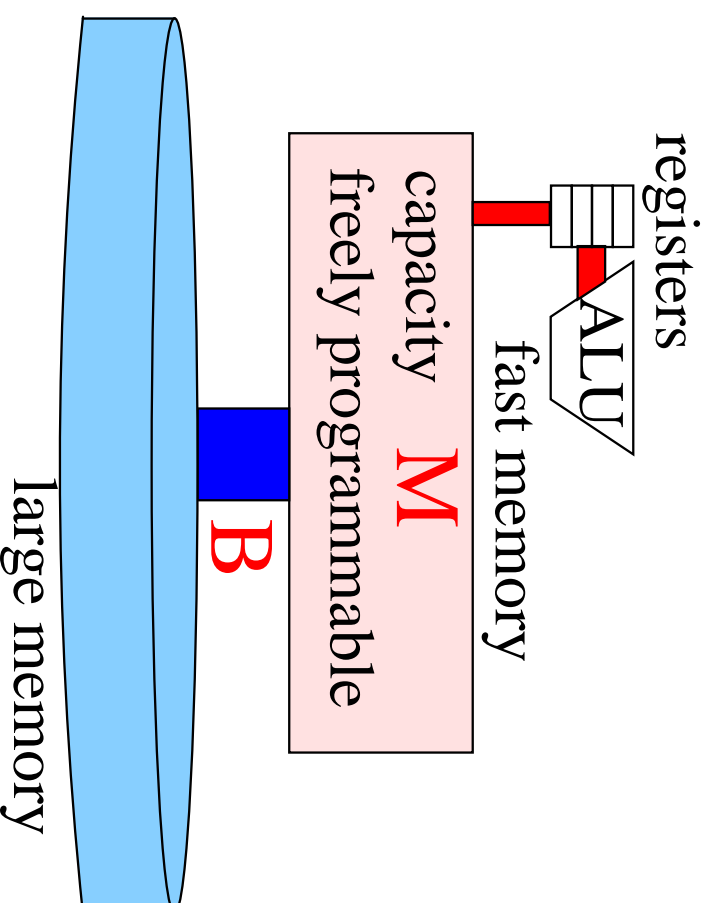


The Secondary Memory Model

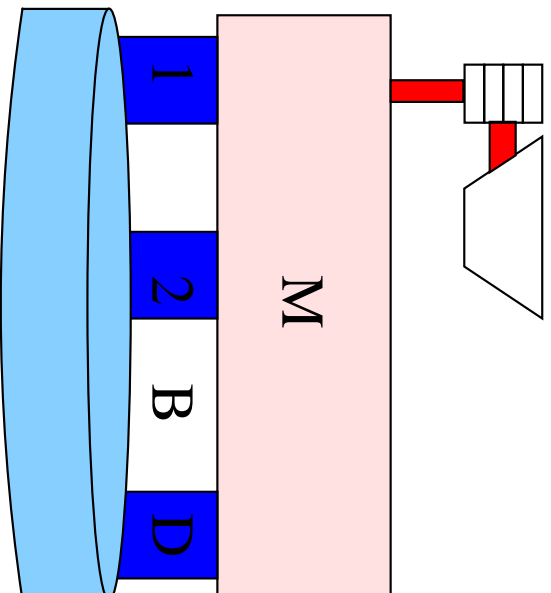
M : Fast memory of size M

B : Block size

Analysis: count block accesses

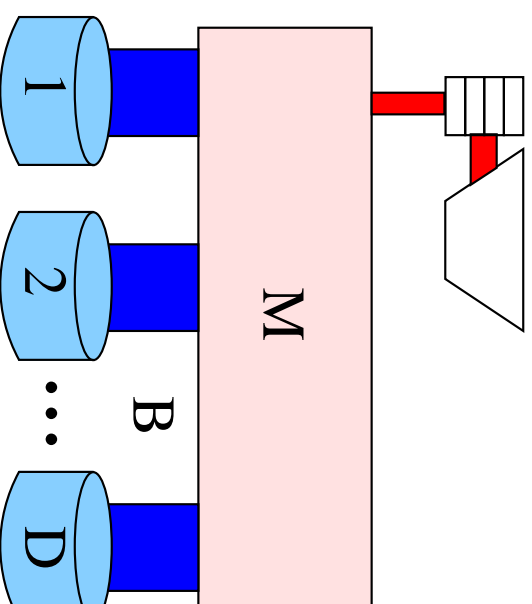


Parallel Disks



multi-head modell

[Aggarwal Vitter 88]



independent disks

[Vitter Shriver 94]

I/O Step := Access one block for each disk or head

Idea: load balancing = **Emulation** of the multi-head model.

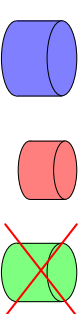
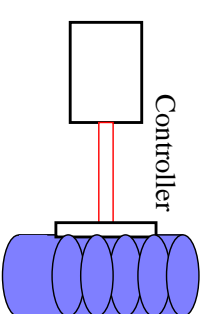
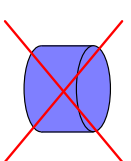
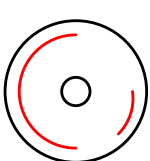
writing → offline reading → reading → generalize model

More Realistic Models

- Variable **block lengths**
- Disk failures**
- Communication bottlenecks**
- Asynchronous** accesses from several applications
- Inhomogenous disks

Goal:

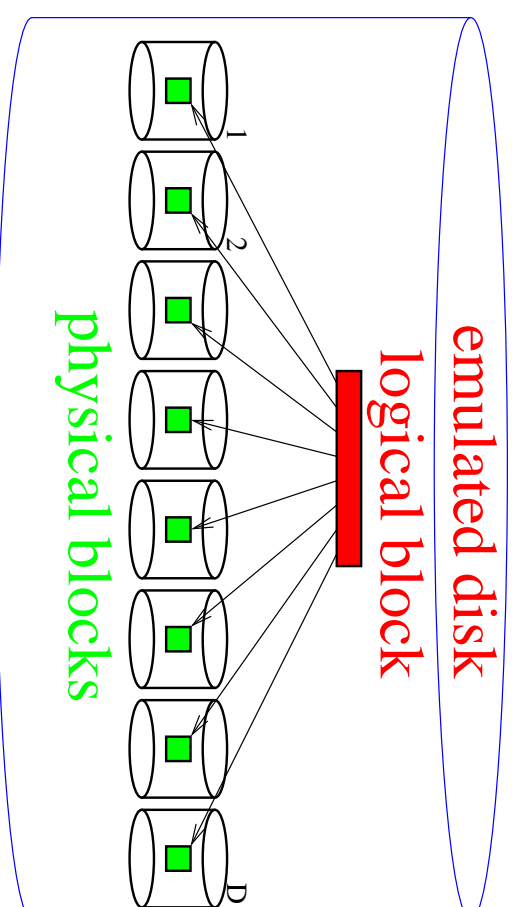
simple programming model + performance guarantees on **complex hardware** model



Striping, a Solution?

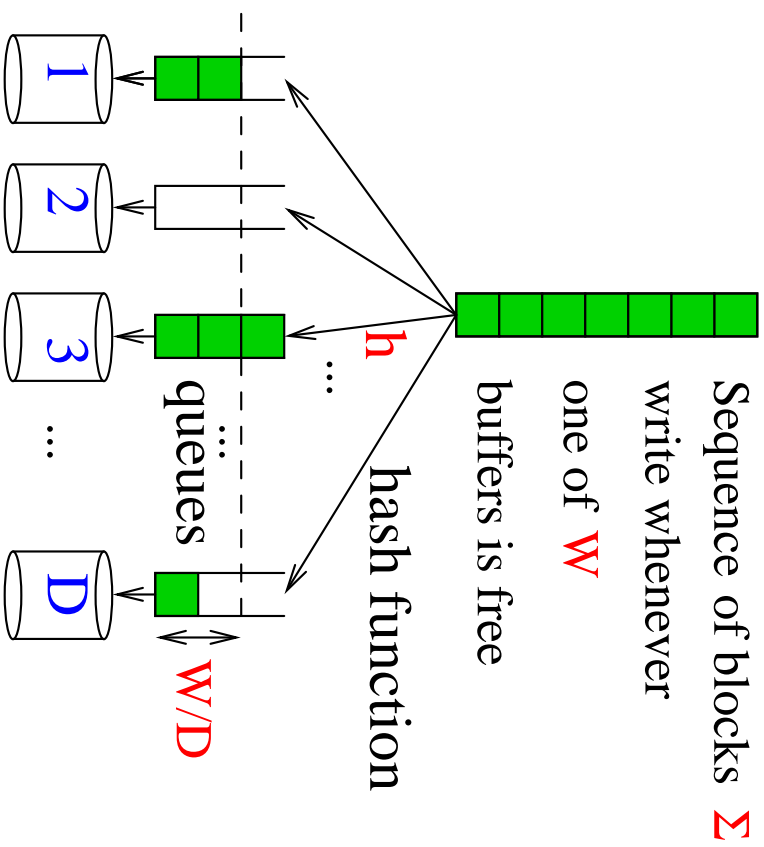
[Salem-GarciaMolina 86, Patterson-Gibson-Katz 88]

- Option for commercial **RAID-Arrays**
- + Good for **sequential** scan of a few large files
- Often algorithms need many block buffers.
 $D \times B = 256 \times 256\text{KB}$ gives logical blocs of **64MB** ...
(e.g. sorting, video-on-demand)
- Often **large blocks are useless**, e.g. transaction processing



Buffered Writing

[Sanders-Egner-Korst SODA00, Hutchinson-Sanders-Vitter ESA 01]



Theorem: For random placement

efficiency $1 - \mathcal{O}(D/W)$ is achieved.

Theorem: Buffered Writing is **optimal**

...
But

how good is optimal?

Analysis: apply **queuing theory** to a **throttled** algorithm.
Use negative association of queue sizes to prove sharp concentration of the sum of the queue sizes.

Optimal Offline Prefetching

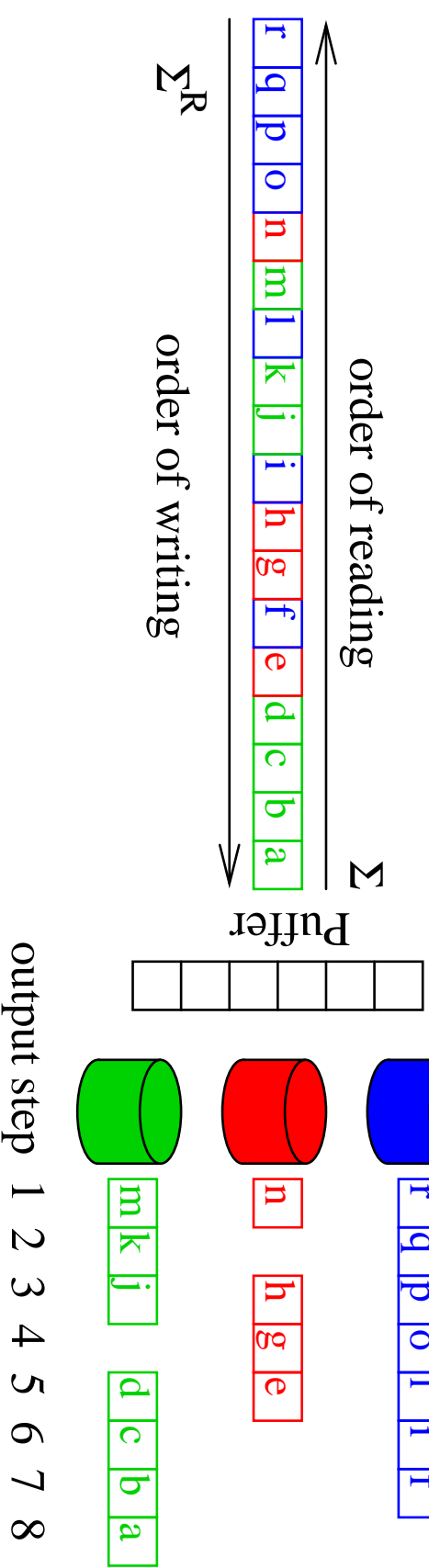
Theorem:

For buffer size W :

\exists (offline) **prefetching** schedule for Σ with T fetch steps



\exists (online) **write** schedule for Σ^R with T output steps.



Optimal Offline Prefetching

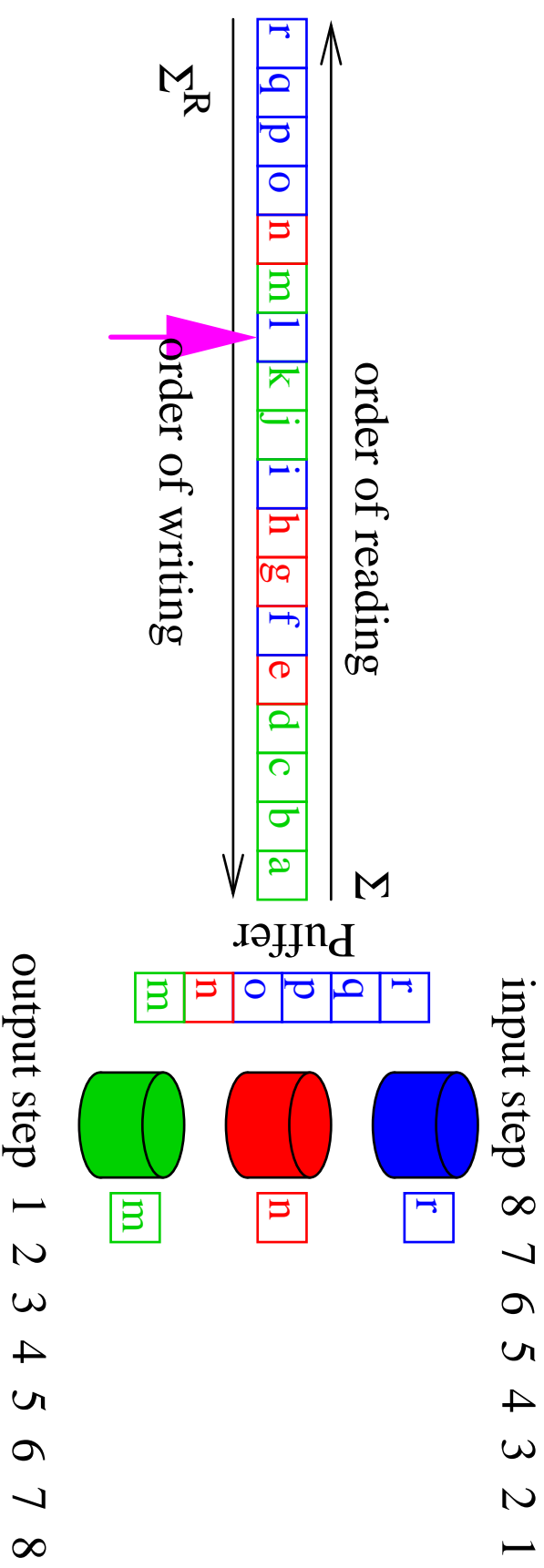
Theorem:

For buffer size W :

\exists (offline) **prefetching** schedule for Σ with T fetch steps



\exists (online) **write** schedule for Σ^R with T output steps.



Optimal Offline Prefetching

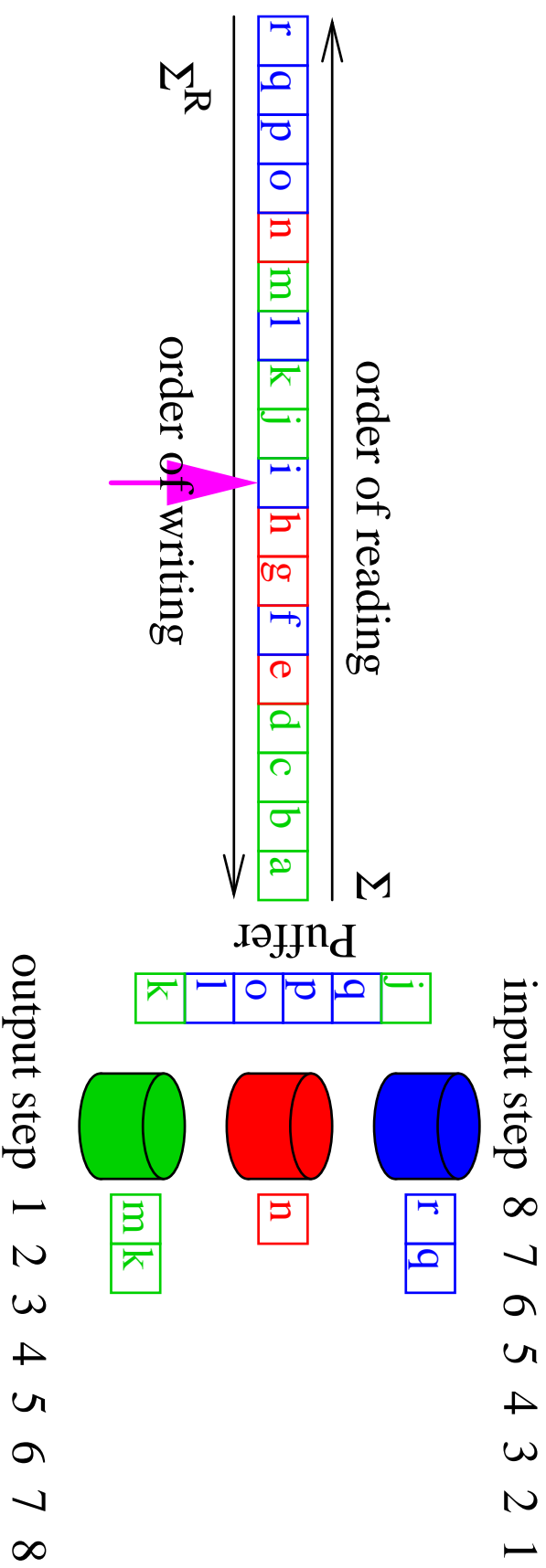
Theorem:

For buffer size W :

\exists (offline) **prefetching** schedule for Σ with T fetch steps



\exists (online) **write** schedule for Σ^R with T output steps.



Optimal Offline Prefetching

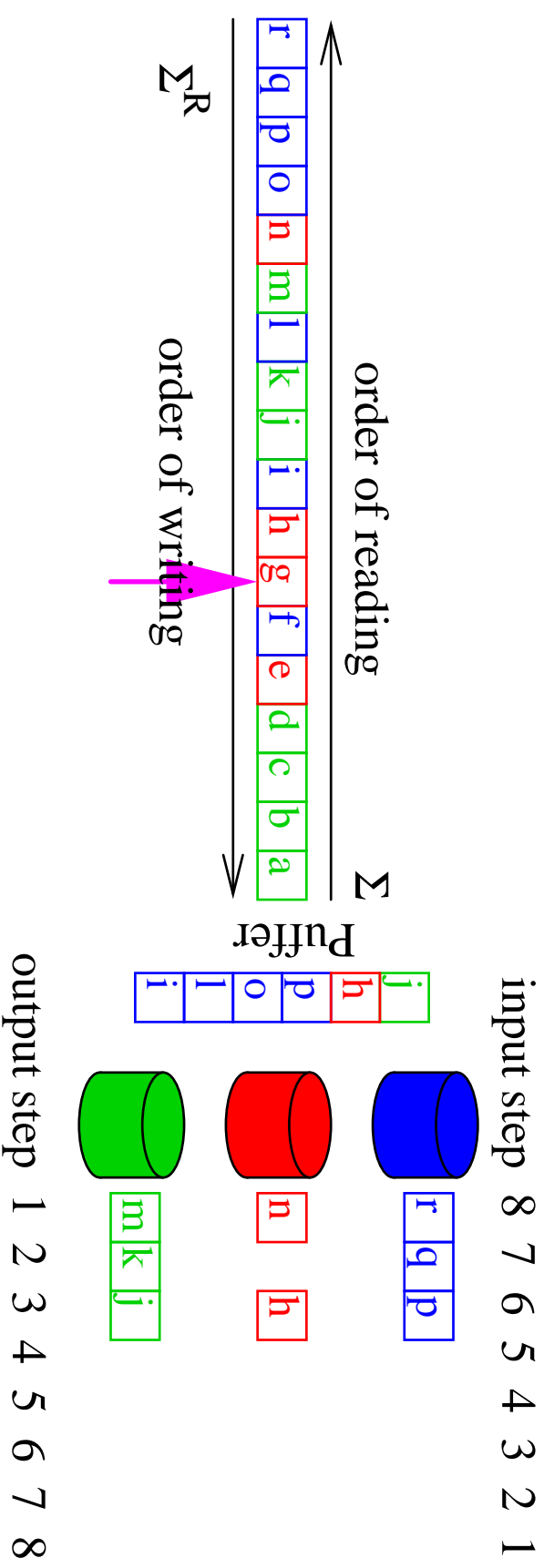
Theorem:

For buffer size W :

\exists (offline) **prefetching** schedule for Σ with T fetch steps

\Leftrightarrow

\exists (online) **write** schedule for Σ^R with T output steps.



Optimal Offline Prefetching

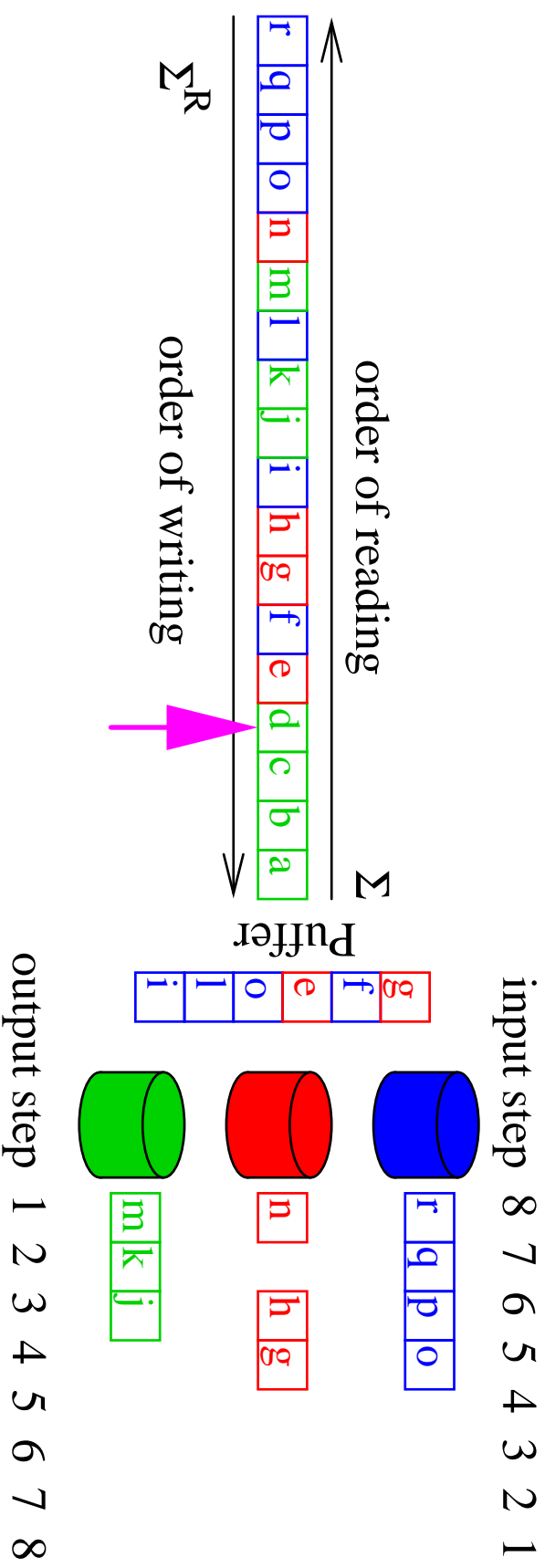
Theorem:

For buffer size W :

\exists (offline) **prefetching** schedule for Σ with T fetch steps



\exists (online) **write** schedule for Σ^R with T output steps.



Optimal Offline Prefetching

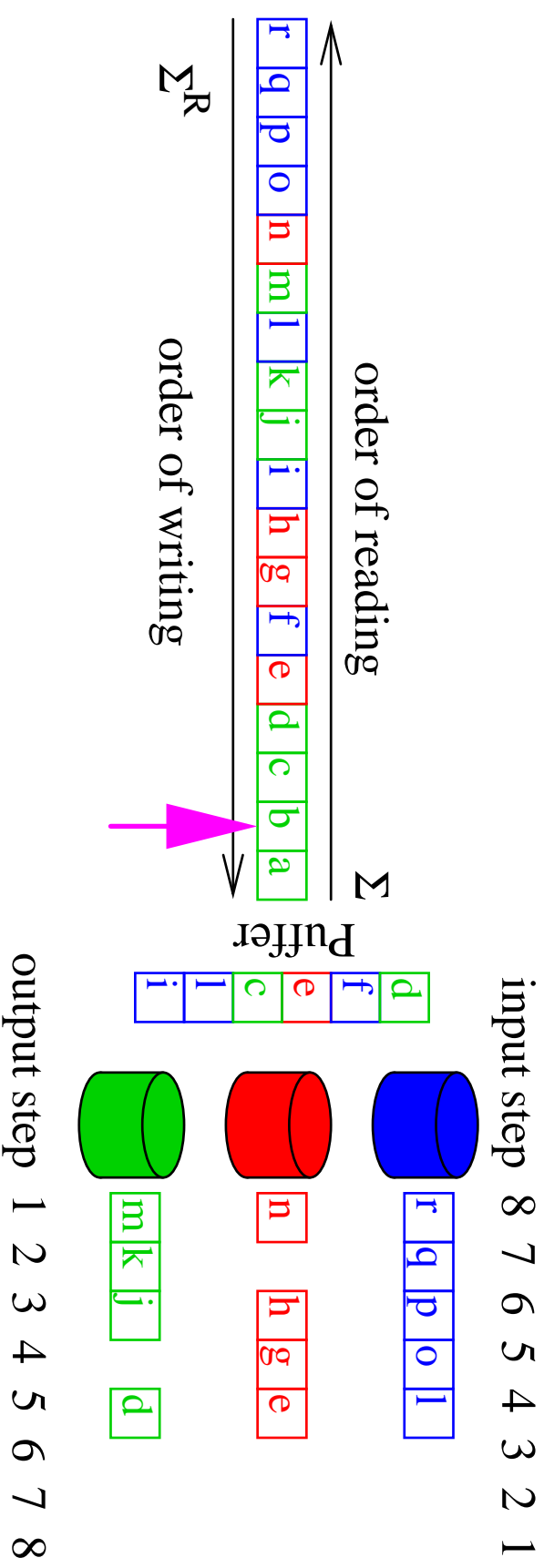
Theorem:

For buffer size W :

\exists (offline) **prefetching** schedule for Σ with T fetch steps



\exists (online) **write** schedule for Σ^R with T output steps.



Optimal Offline Prefetching

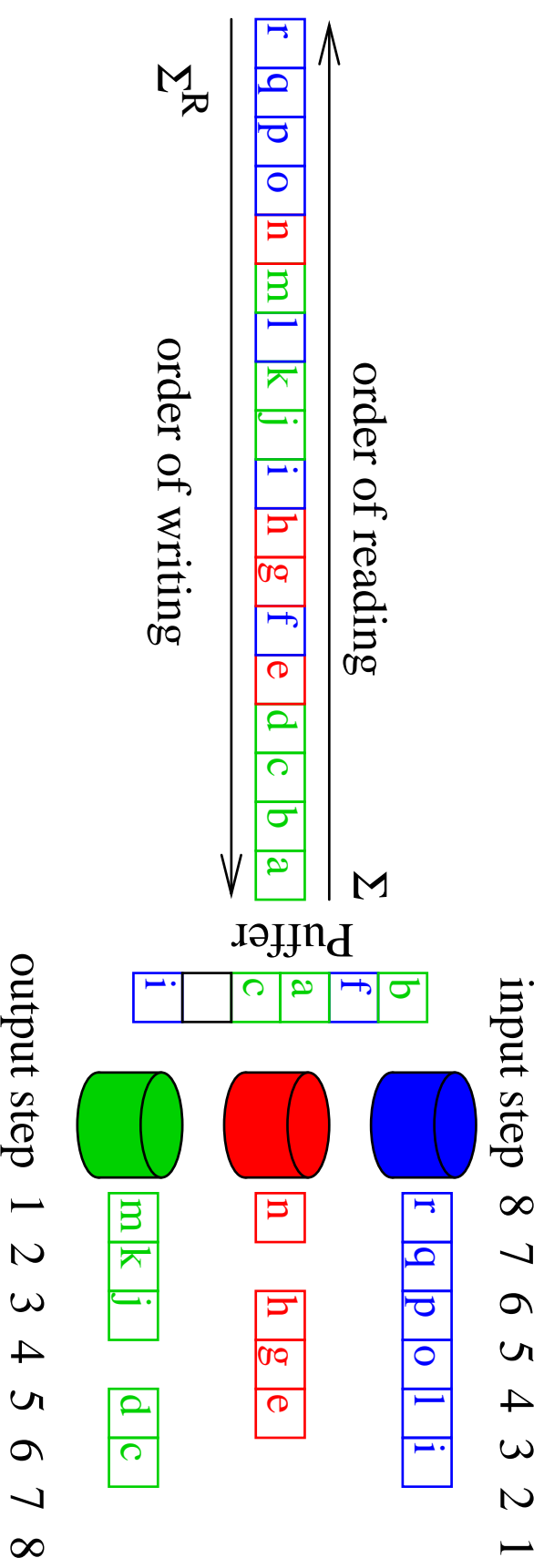
Theorem:

For buffer size W :

\exists (offline) **prefetching** schedule for Σ with T fetch steps

\Leftrightarrow

\exists (online) **write** schedule for Σ^R with T output steps.



Optimal Offline Prefetching

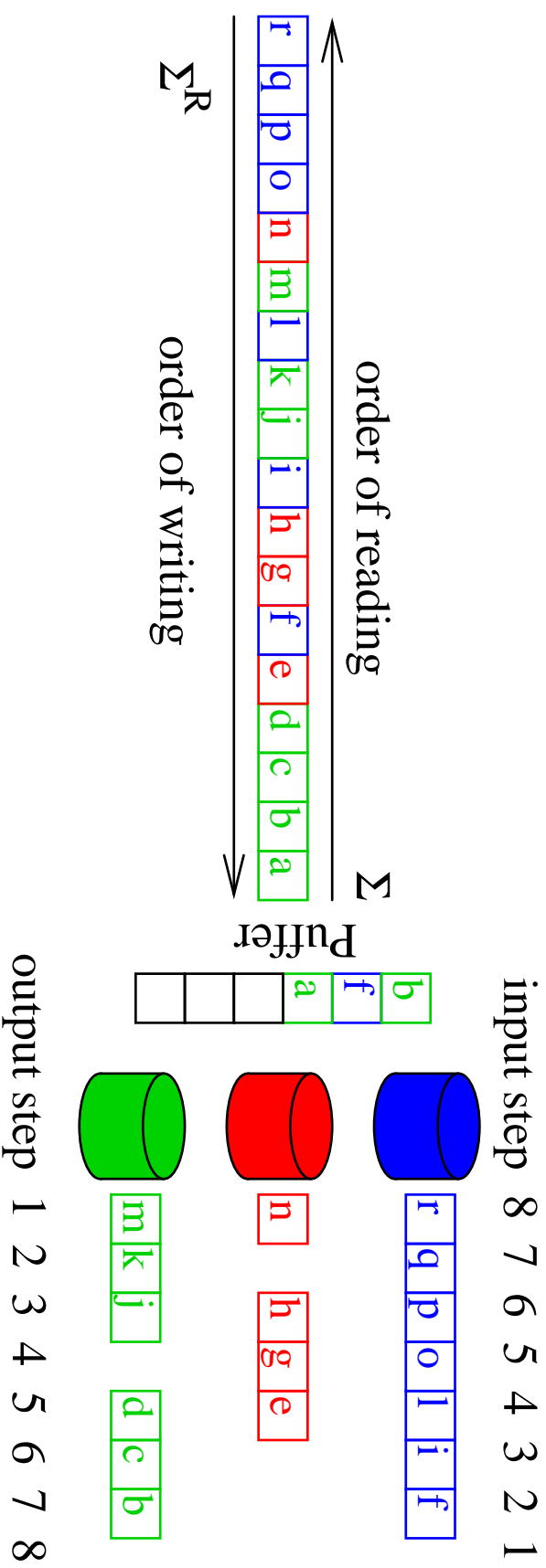
Theorem:

For buffer size W :

\exists (offline) **prefetching** schedule for Σ with T fetch steps

\Leftrightarrow

\exists (online) **write** schedule for Σ^R with T output steps.



Optimal Offline Prefetching

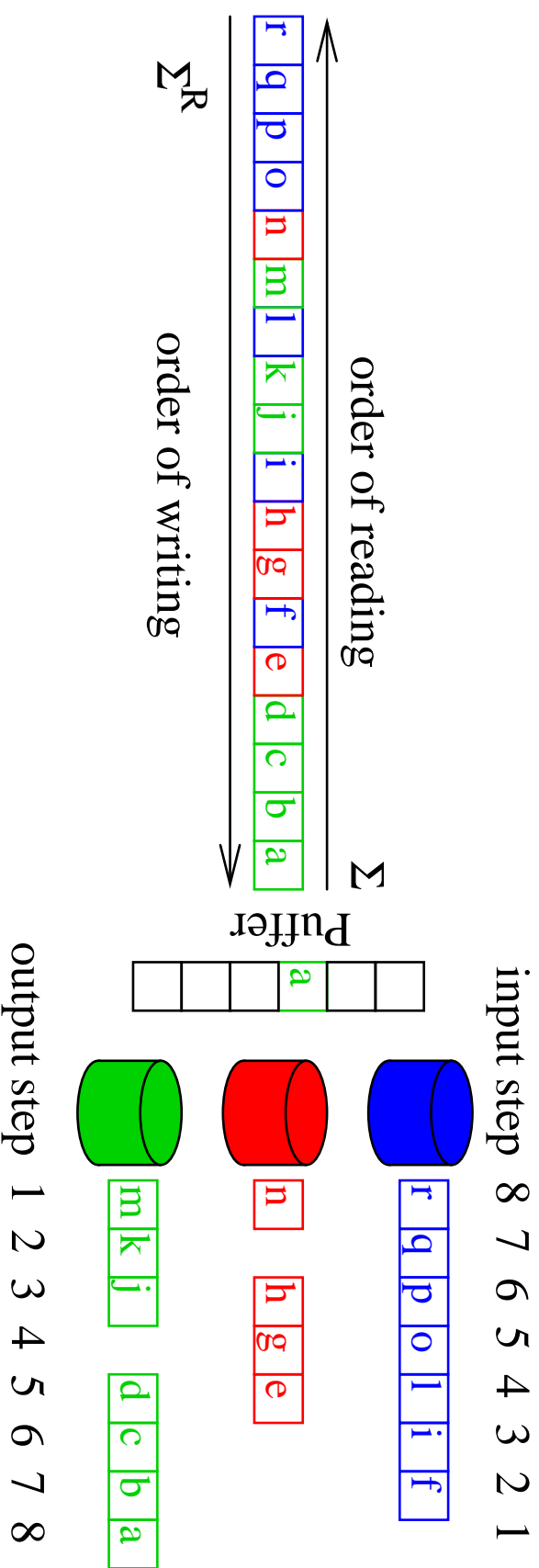
Theorem:

For buffer size W :

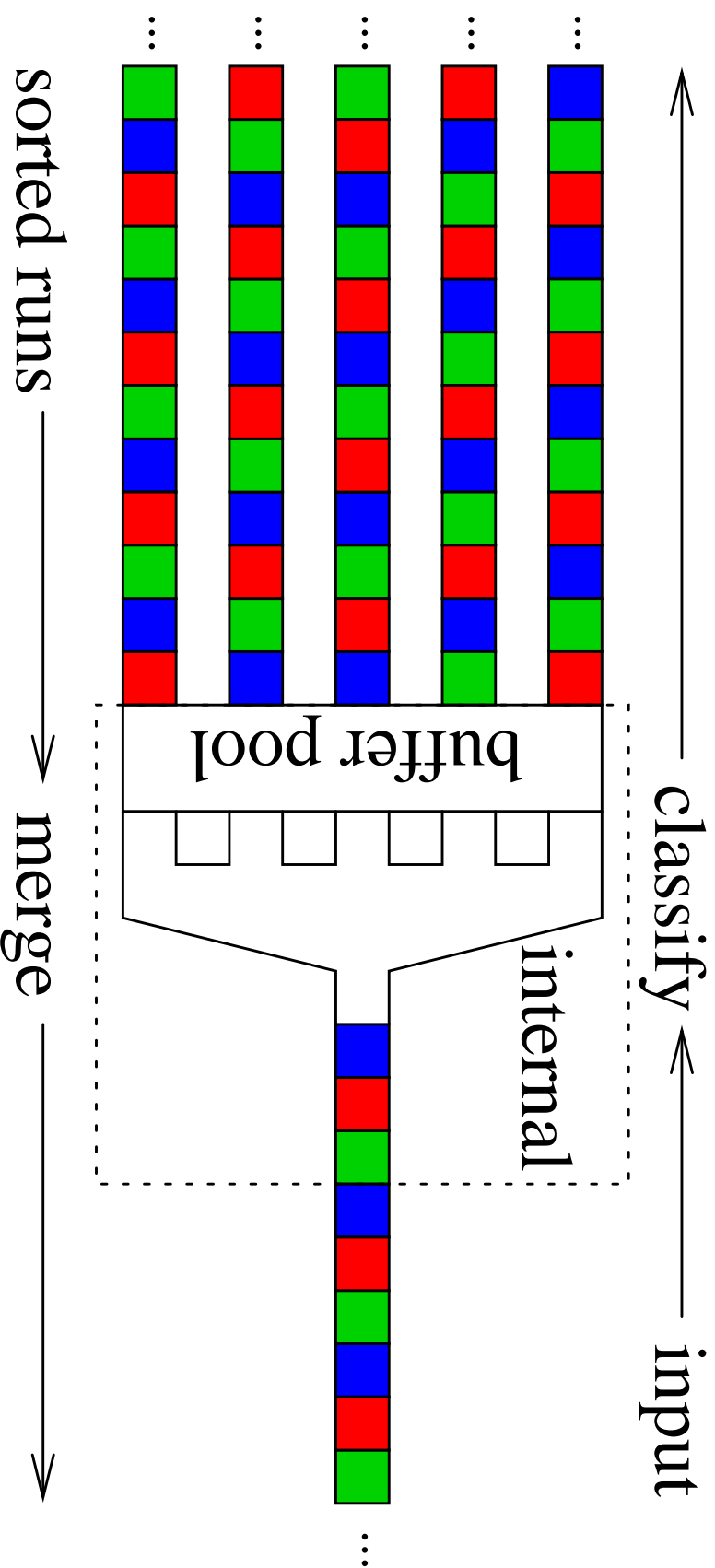
\exists (offline) **prefetching** schedule for Σ with T fetch steps



\exists (online) **write** schedule for Σ^R with T output steps.



Application: Sorting



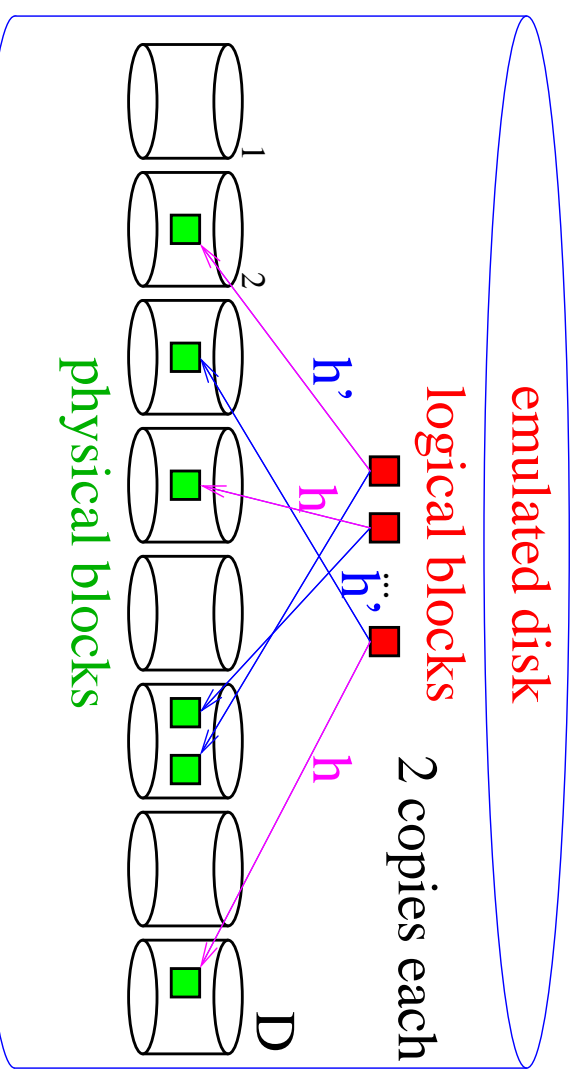
Storage related ingredients: buffered writing, optimal prefetching, **randomized**

cyclic allocation (\approx do **both** striping and randomization) [Vitter-Hutchinson 00]

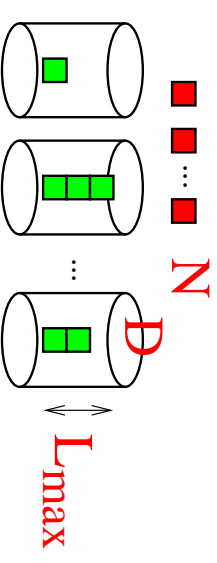
Reading Online: Random Duplicate Allocation



- Idea: more flexibility by **redundant** storage
- Two **copies** for each **log. block**
- Two hash functions h und h'



arbitrary request set $\xrightarrow{h, h'}$ random targets
 planning problem: read N blocks from D disks.



which copies to read for minimizing the number of I/O steps L_{\max} ?

Graph Theoretic Interpretation of RDA

Allocation Graph: G_a ,

undirected random multigraph
with N requests (edges)

Schedule Graph: G_s ,

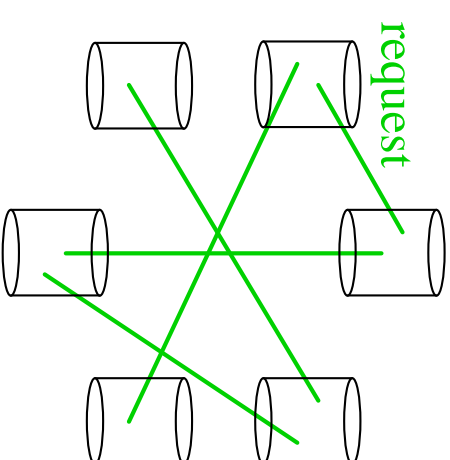
G_a with oriented edges

Maximal Load: $L_{\max}(G_s) := \max_{i=1}^D$ outdegree(i)

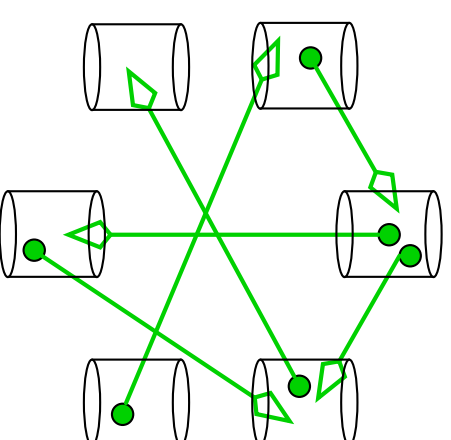
Optimal Schedule: Schedule with

$$L_{\max} = L_{\max}^*(G_a) := \min_{G_s} L_{\max}(G_s)$$

Allocation Graph



Schedule Graph



Previous Work on RDA

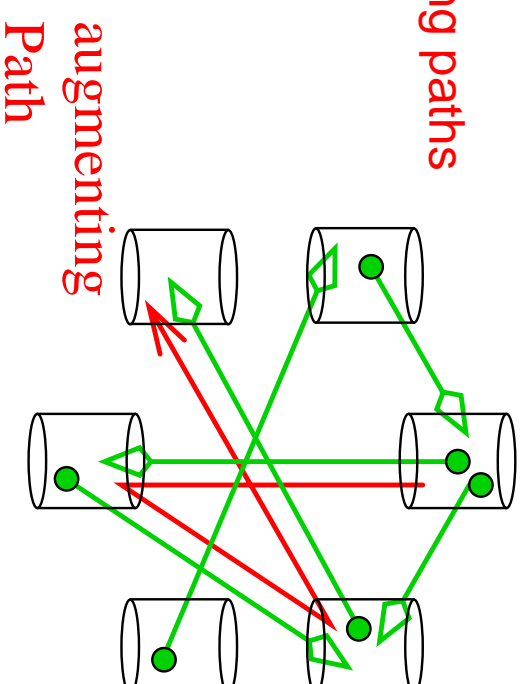
“Local Strategies”: $\mathcal{O}(\lceil N/D \rceil)$

[Karp-Luby-Meyer adH 92, . . . , Pittel-Spencer-Wormald 96] and

$N/D + \Theta(\log \log D)$ [BCSV 00]

Experiments: Mostly for video applications, and simple scheduling strategies

Global Optimization: [Korst 97] Find **augmenting paths**



How Good is Global Optimization?

It is **optimal**.

Theorem 1. *When no augmenting paths exist $L_{\max} = L_{\max}^*$.*

How Good Is Optimal?

[Sanders-Egner-Korst-SODA00]

Theorem 2. $L_{\max}^* \leq \left\lceil \frac{N}{D} \right\rceil + 1$ *whp.*

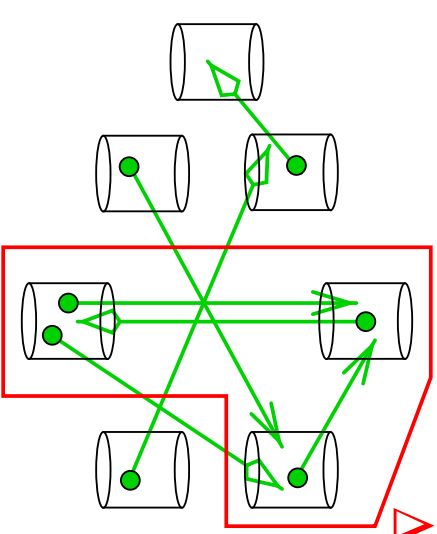
Proof Idea

Let $L_\Delta := |\{\text{requests } (u, v) : u \in \Delta \wedge v \in \Delta\}|$

be the **unavoidable** Load of a disk set Δ .

Starting Point: Optimal schedules are “witnessed”

by highly loaded sets.



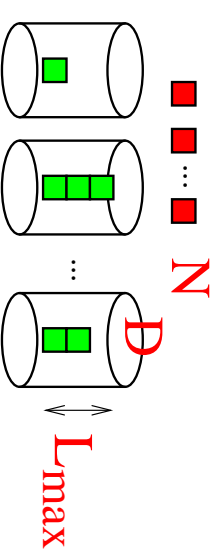
Lemma 3. $L_{\max}^* = \max_{\emptyset \neq \Delta \subseteq \{1..D\}} \left\lceil \frac{L_\Delta}{|\Delta|} \right\rceil$.

Now, it remains to show that the occurrence of subsets with

$L_\Delta > x = |\Delta| \cdot (\lceil N/D \rceil + 1)$ is improbable

$$\mathbb{P}[\exists \Delta : L_\Delta > x] \leq \sum_{|\Delta|=1}^D \binom{D}{|\Delta|} \mathbb{P}[L_\Delta > x]$$

L_Δ is $\mathcal{B}\left(N, \frac{|\Delta|^2}{D^2}\right)$ **binomially** distributed



It “only” remains to evaluate a probabilistic bound.

Refinements

Fast Scheduling

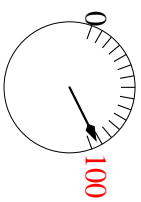
$$L_{\max} \leq \left\lceil \frac{N}{D} \right\rceil + 1 \text{ in time } \mathcal{O}(D \log D) \text{ for } N = \mathcal{O}(D).$$

Towards Linear Time

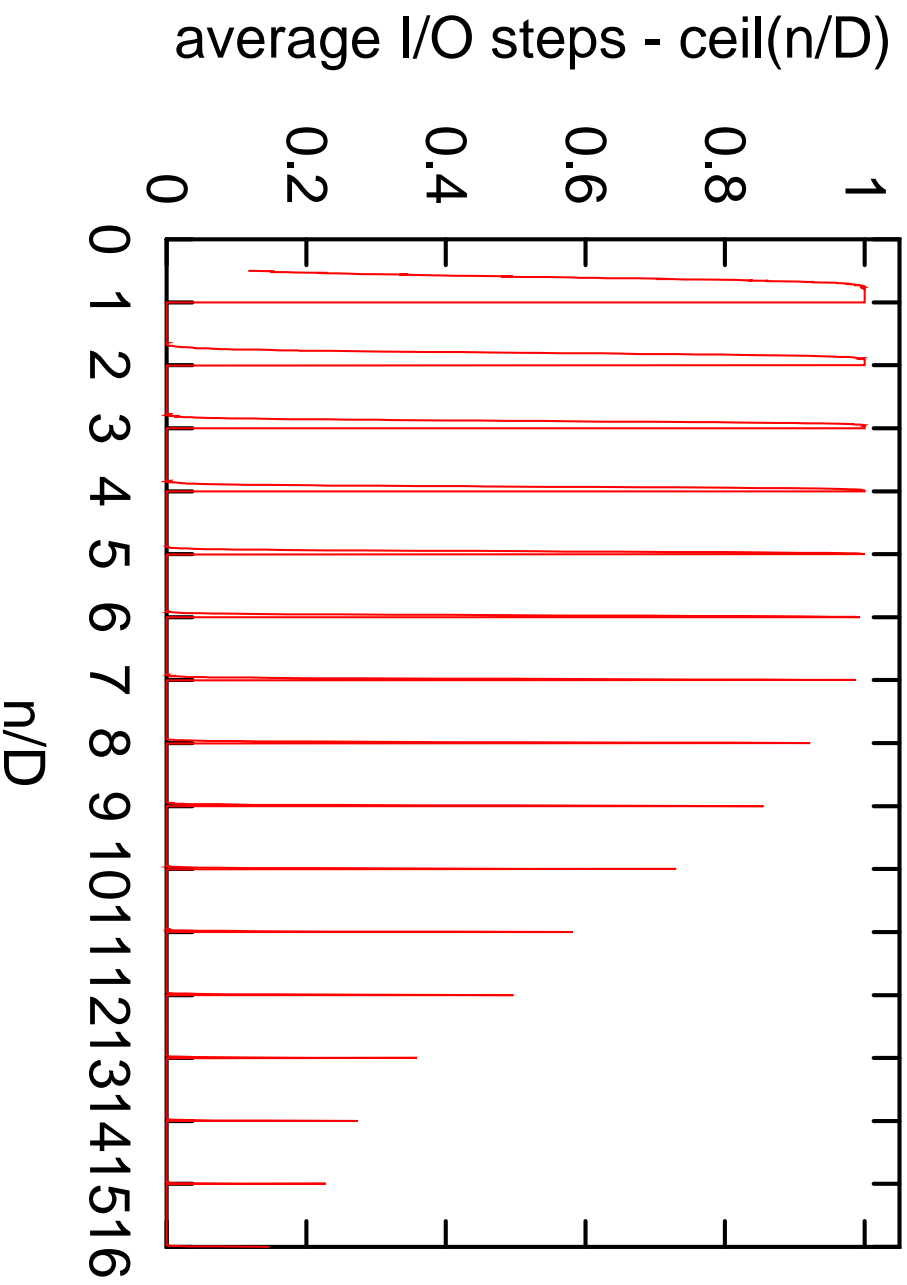
[Cain-Sanders-Wormald 04]

Idea: least loaded disks volunteer for committing one request

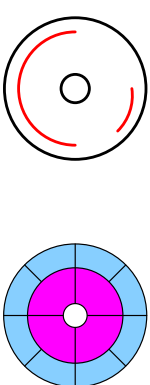
Efficiency near 100%:



$L_{\max} = \lceil N/D \rceil$ is achieved, when N is not too close to a multiple of D



Variable Block Size, Zones:



Unsplittable Blocks: NP-complete

Splittable Blocks: Mlog sizes $l_i \in [0, 1]$. Behavior like splittable unit size

block with $N = \sum_i l_i$

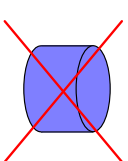
Disk Failures:

Place copies on **different** disks.

One disk fails \rightsquigarrow

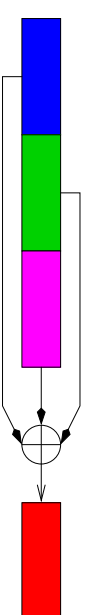
Throughput \approx **intact** system with $D - 1$ disks

More later



Reducing Redundancy

$2 \rightsquigarrow 1 + \frac{1}{r}$ if we accept r times larger logical blocks

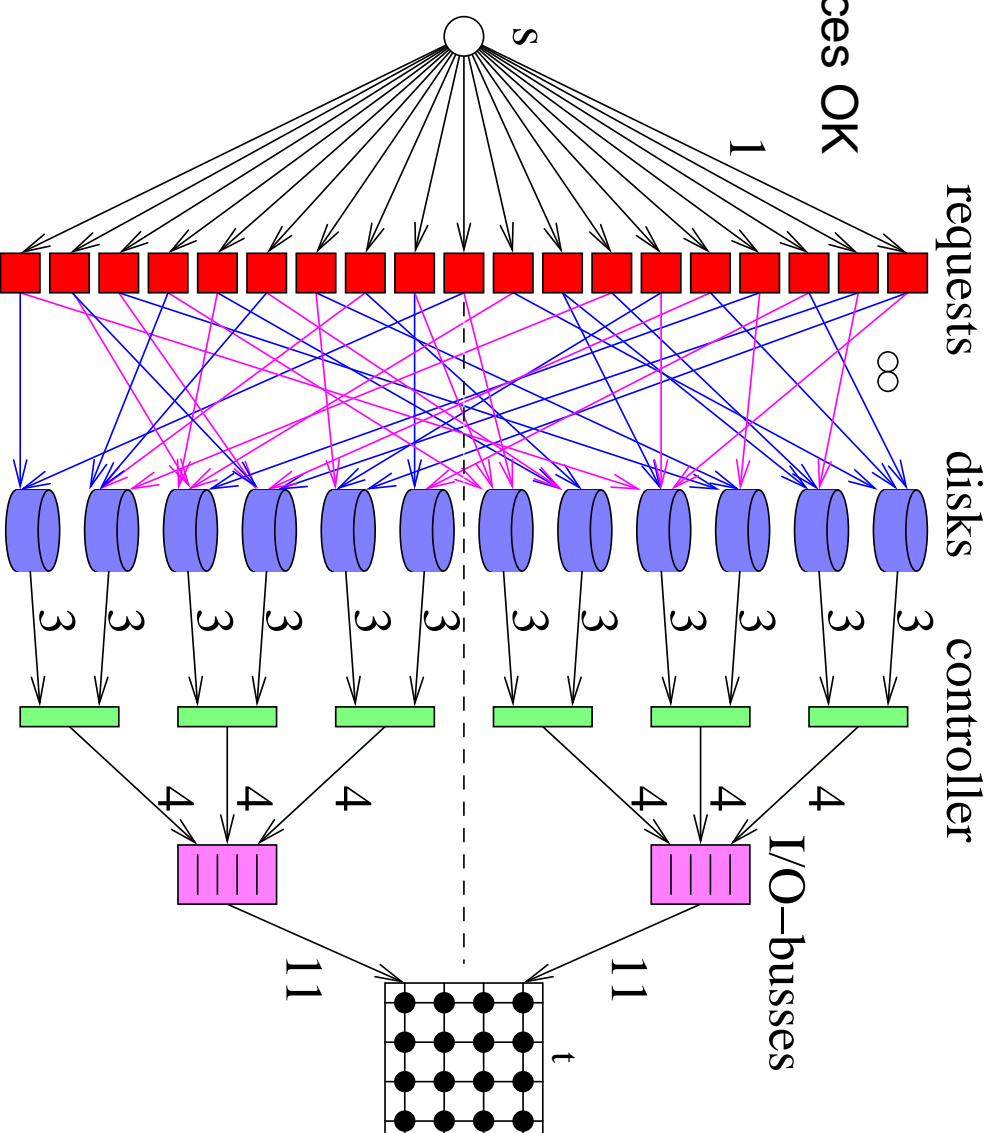


Communication Bottlenecks

- **Optimal Schedules** for arbitrary

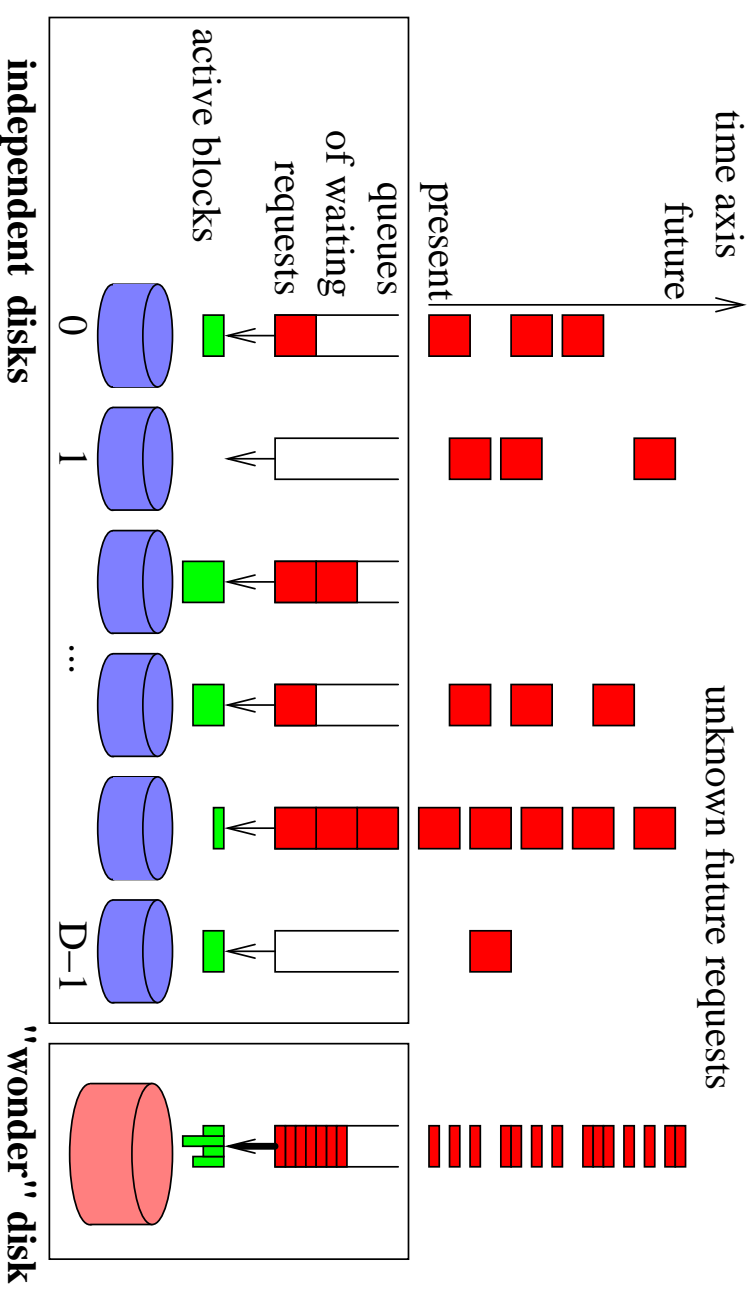
acyclic networks

- **Inhomogeneous devices OK**



Asynchronous External Memory

- Unit size blocks (Refinement: unpredictable service times)
- Delay = 1 + queue length

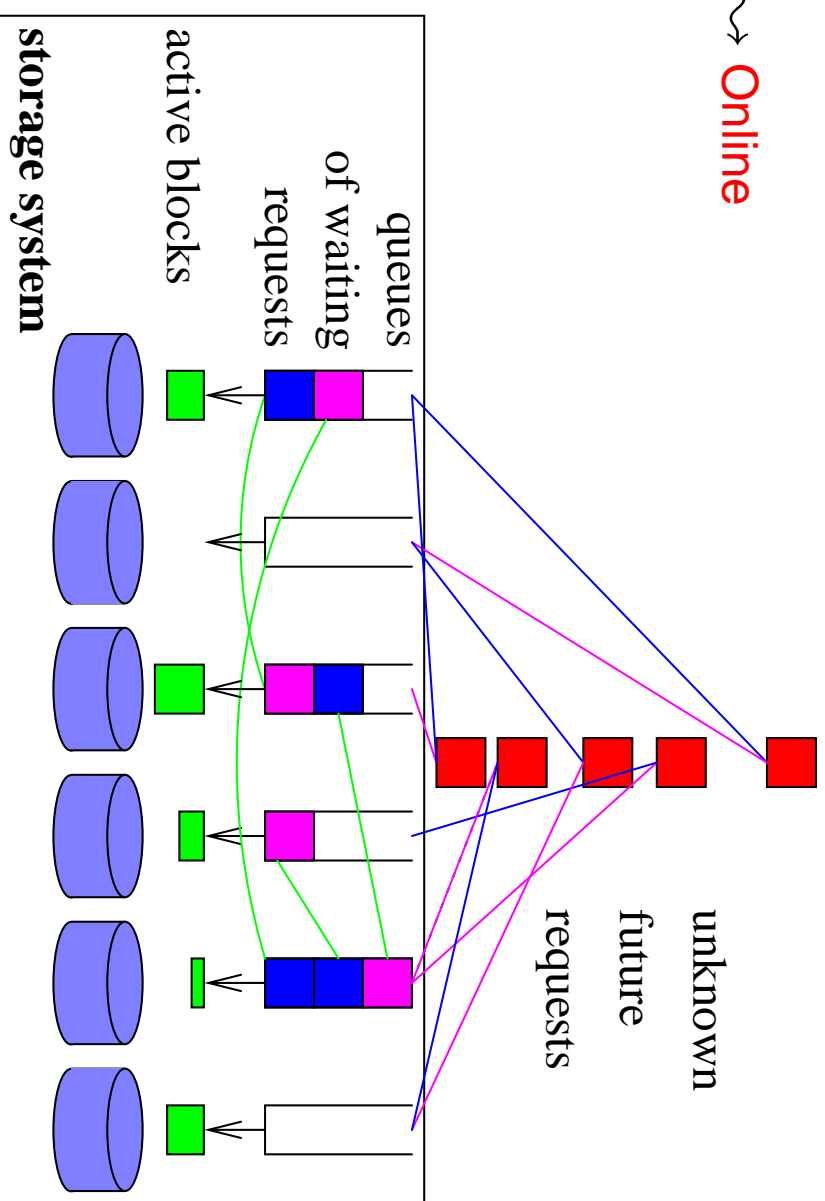


Asynchronous Access

[Sanders-SPAA00]

General purpose memory system:

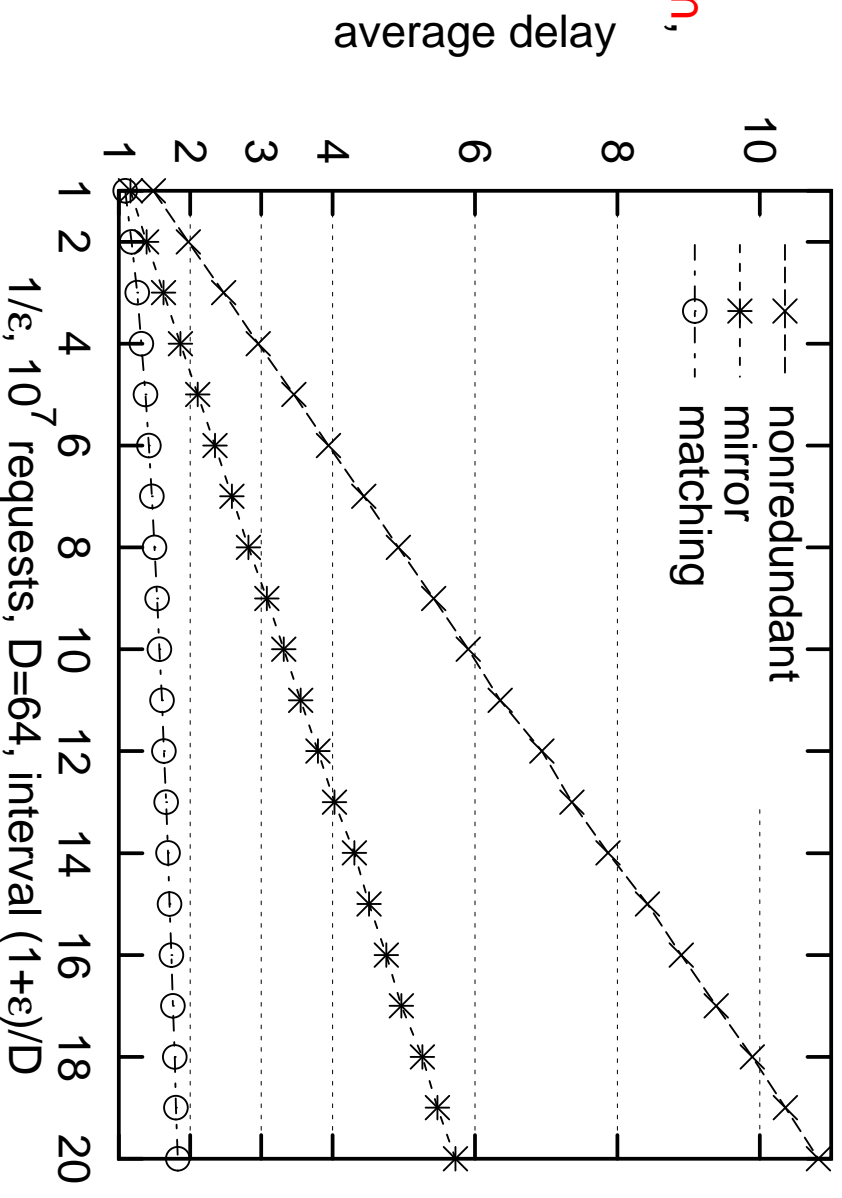
- Batches \rightsquigarrow **stream of requests**
- Maximum load L_{\max} \rightsquigarrow **distribution of delays**
- Offline (batched) \rightsquigarrow **Online**



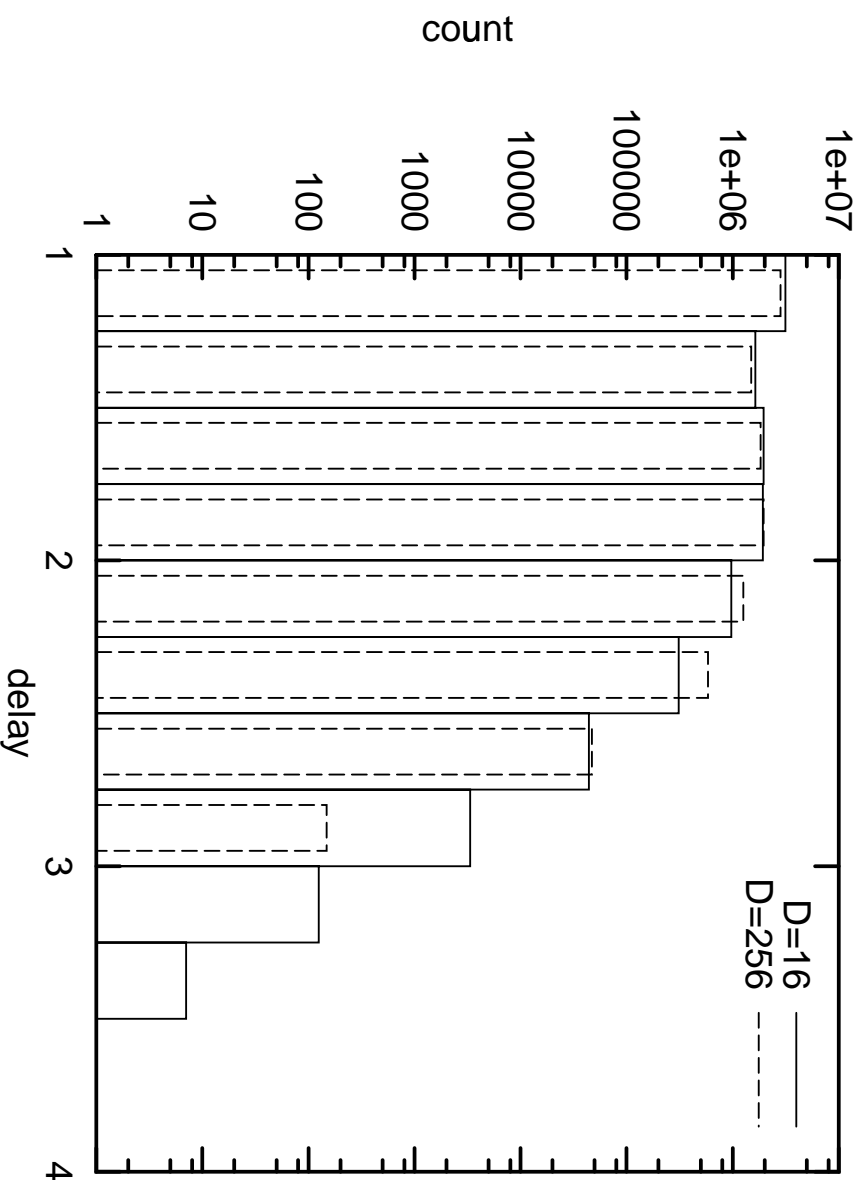
Asynchronous: Algorithmic Approach

- Mapping waiting Request → time slot on a disk (bipartite matching).
- Update for each request (about $10\mu\text{s}$ per access)

- “Optimal” if
 - no other requests come
 - Complete analysis is **open**,
- even for greedy alg.
 [Vvedenskaya-et-al.,
 Mitzenmacher 96]

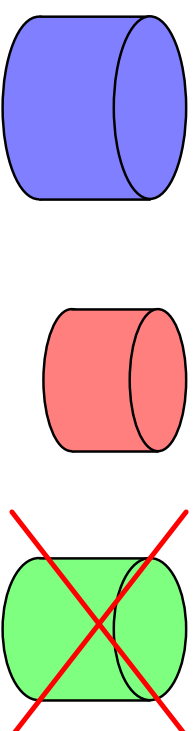


Delay Distribution for Matching



- Scheduling quality **improves** with larger D
- Maximum delay close to average delay
- \rightsquigarrow good for soft real time applications

Virtualization: Handling Inhomogenous Disks



What we want

1. **Add disks** of different size incrementally
2. Tolerate disk **failures**, i.e., remove disks
3. Complete use of disk **capacities**
4. Good **load balancing** with respect to maximal throughput
5. Low reconfiguration costs

Simplification: Ignore disk speeds. We (falsely) assume that capacity and bandwidth grow proportionally. (Alternative: ignore capacity that cannot be sustained under full load)

Volumes

Map physical blocks to

$D' \gg D$ volumes

use **random permutation**

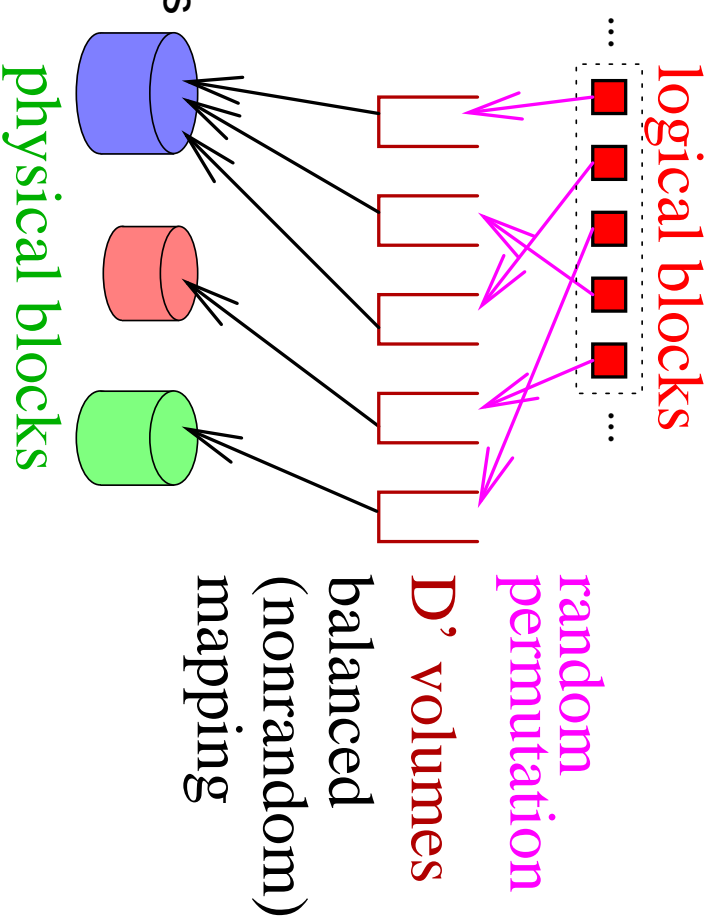
$iD'..(i+1)D' - 1$

→ volume $0..D' - 1$

Map volumes **systematically** to disks

no. volumes on a disk

is proportional to its capacity

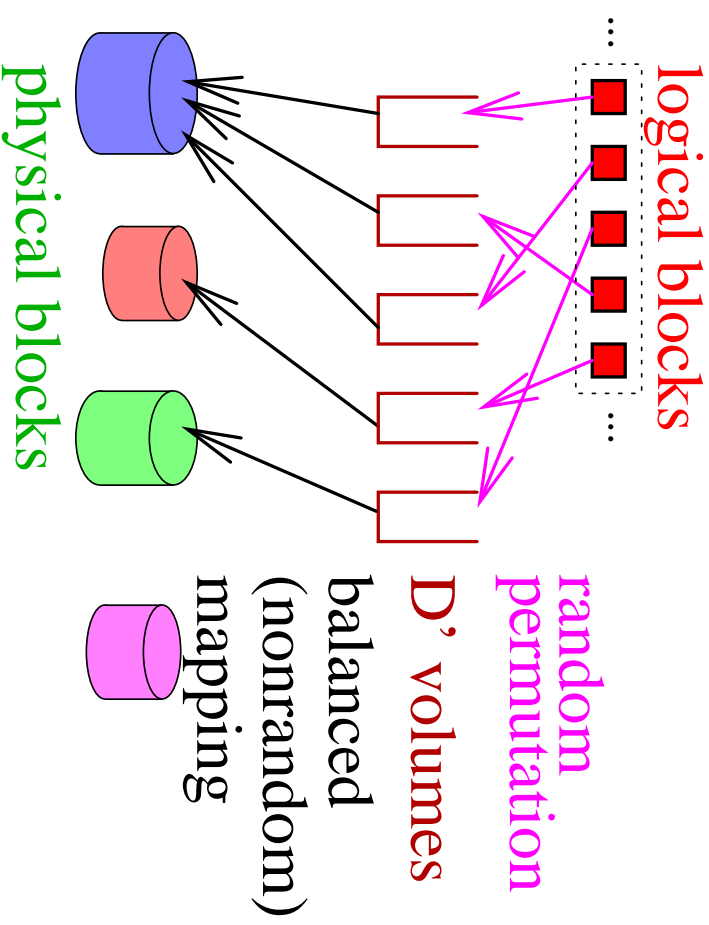


Adding a Disk

Move volumes from disks

with highest load

↪ moves only data for new disk



Tolerating Component Failures

Partition system into two virtual servers

A and *B*

with about the same disk capacity.

One copy to each virtual server.

Anything may fail

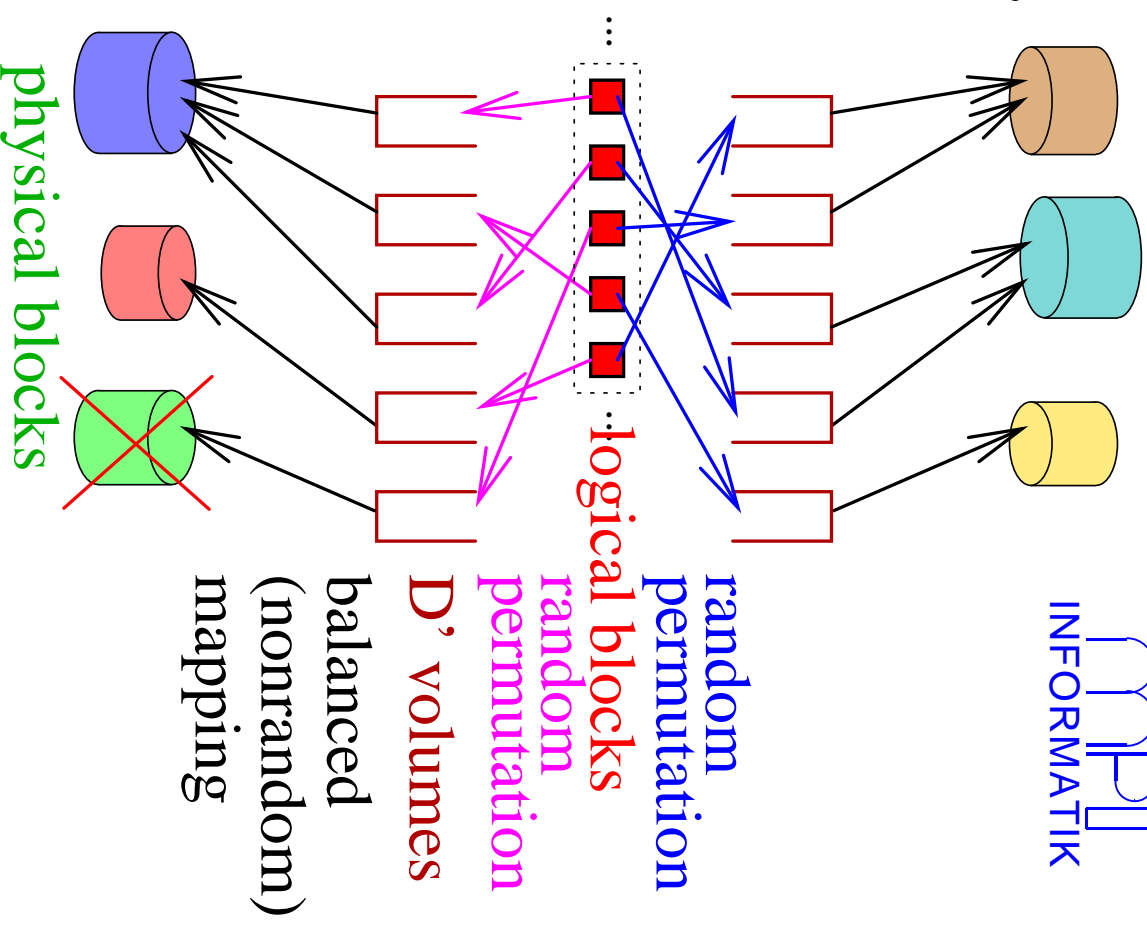
that does not affect both servers

Move affected volumes to

existing disks (**virtual spares**),

greedily, to least loaded ones

use all disks in parallel for migration

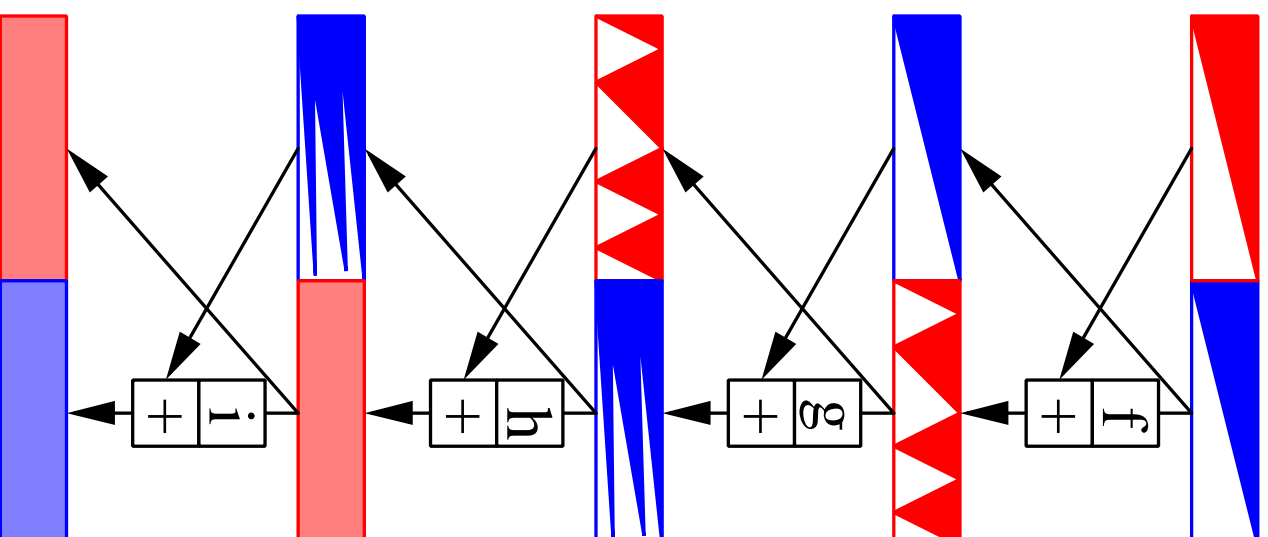


Feistel Permutations

f , g , h , and i

are hash functions

easily invertible!



Some Open Problems of Theoretical Interest

- Performance guarantees for **any** reasonable **asynchronous** RDA scheduler, e.g. expected delay as a function of arrival rate.
- **Verification**
- What can adaptive data migration do for us?
 - Optimize for capacity **and** throughput, i.e., put frequently needed data on fast disks
 - Save energy, e.g., **Massive Arrays of Idle Disks**
- **Communication** in dynamically changing storage area networks
 - Network reconfiguration
 - Planning for data migration
 - Locality versus load balancing

Some orthogonal (?) issues ignored here: Caching, . . .