Mathematical Models of Write Amplification in FTLs

Peter Desnoyers
Northeastern University

Write Amplification

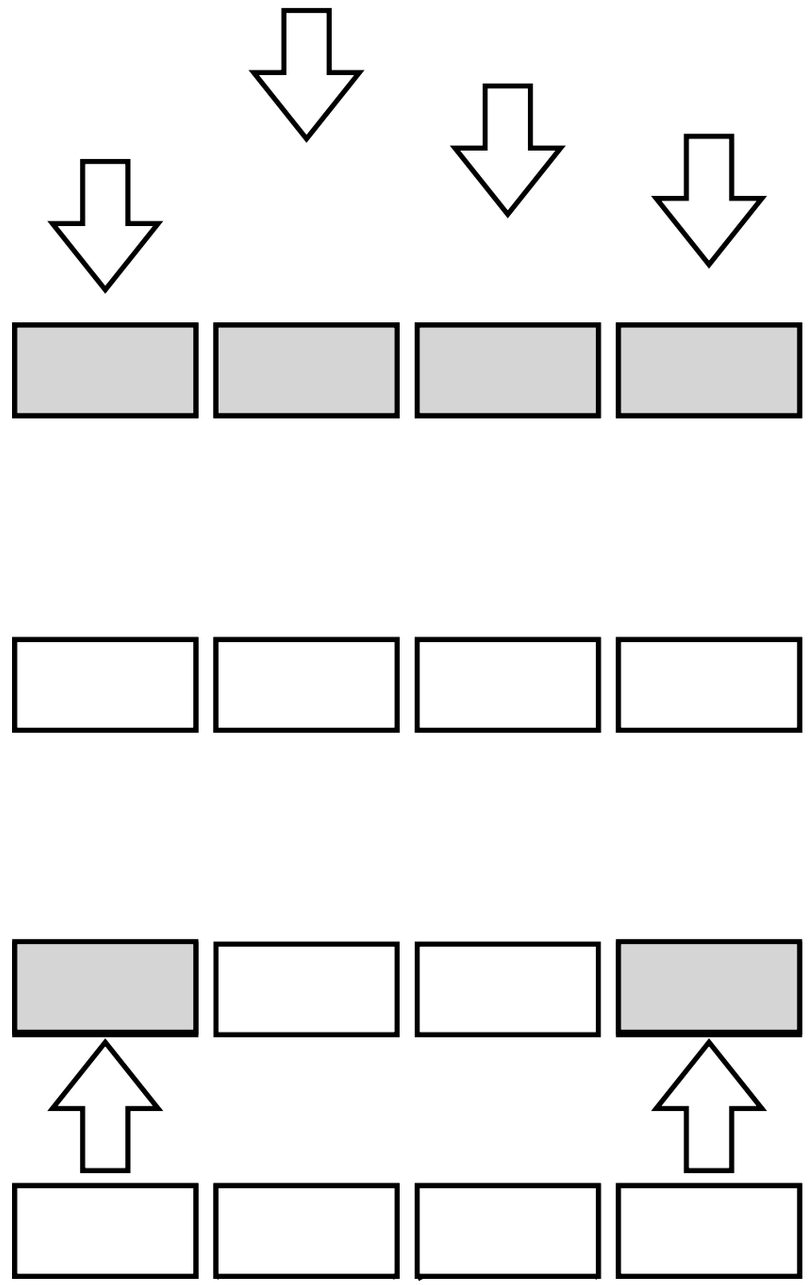
Random page writes

+

Multi-page block erase

=

Internal copying
(write amplification)



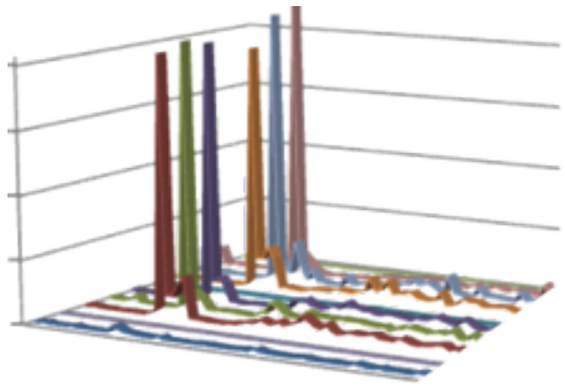
What can we do about it?

- Design Flash Translation Layers

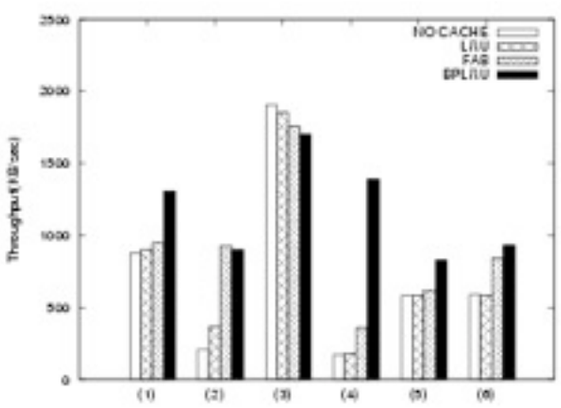
BPLRU HFTL JFTL
DFTL ComboFTL
KAST
FAST MNFTL μ-FTL
BAST Lazy-FTL
SuperBlock NFTL
A-SAST
LAST

What can we do about it?

- and test them



collect traces

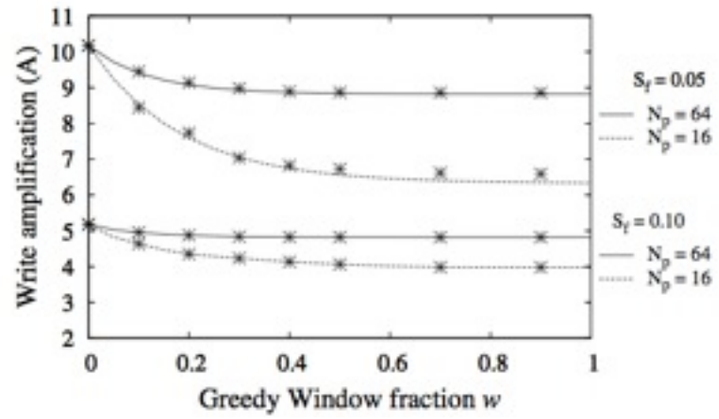


and simulate...

What's missing?

- Mathematical understanding
  - ***Why*** does FTL 1 perform better than FTL 2?
  - Can we do better? What is optimal?

$$\sum_{i=X_0}^{N_p} f_i = \sum_{i=X_0}^{N_p} \frac{k}{i} = 1$$



$$\int_{i=X_0-\frac{1}{2}}^{N_p+\frac{1}{2}} \frac{k}{i} di = 1$$

$$X_0 = \frac{1}{2} + N_p(S_f - 1)\cdot W\left(\frac{\left(e^{-\frac{1}{2N_p}-1}\left(N_p+\frac{1}{2}\right)^{1-S_f}\right)^{\frac{1}{1-S_f}}}{N_p(S_f-1)}\right)$$

What this talk is about:

- Simple flash translation layers
  - Page-mapped, no wear leveling
- Simple traffic models
  - Uniform randomly-distributed writes
- Accurate results
  - Models that tell us exactly what is happening in these simple cases.

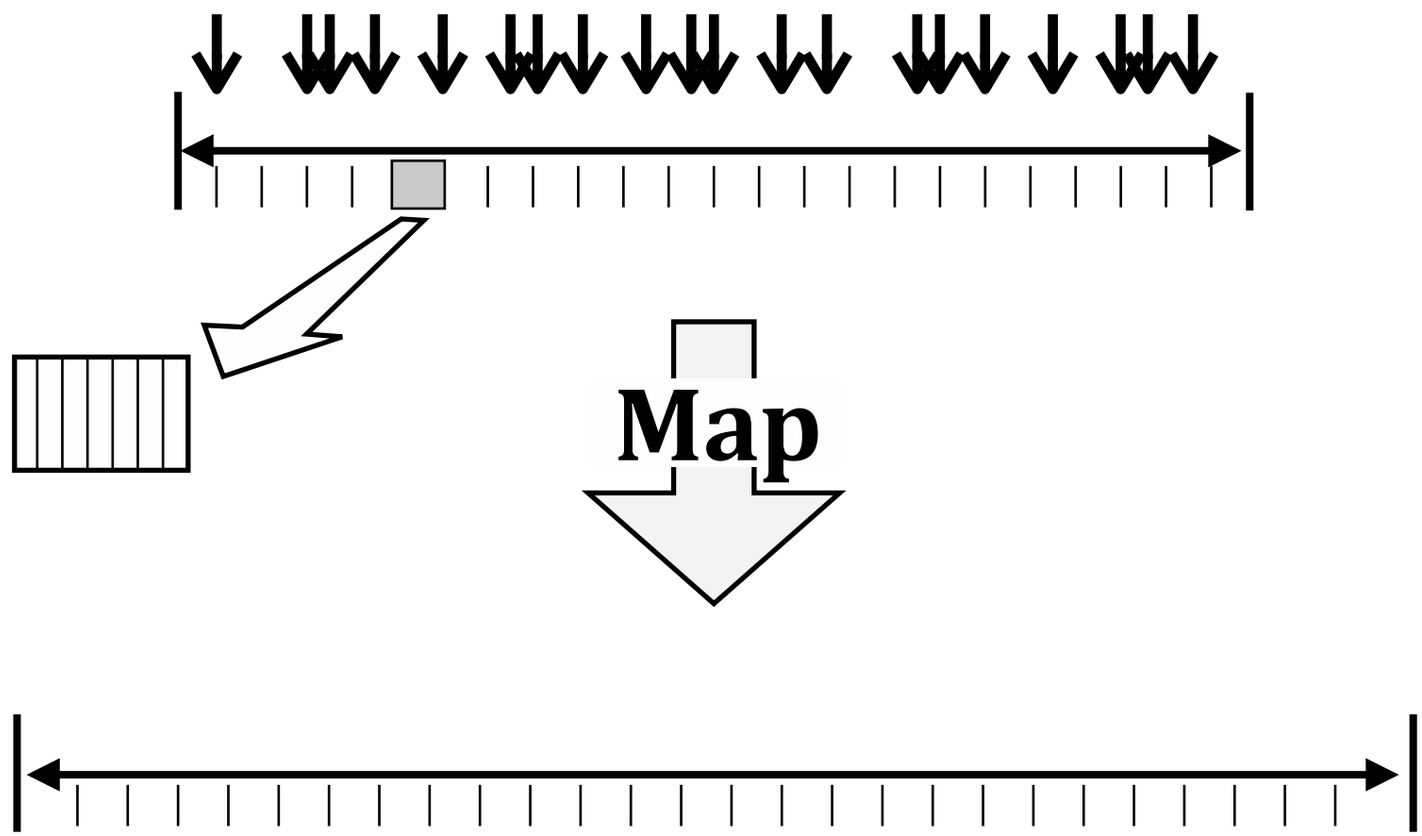
Write Amplification

- $A = \frac{\text{total internal writes}}{\text{external writes}}$

- If N_p = erase block size (in pages)
  N_{GC} = number of valid pages in garbage-collected block
- then $A = \frac{N_p}{N_p - N_{GC}}$ (Cost/Benefit)

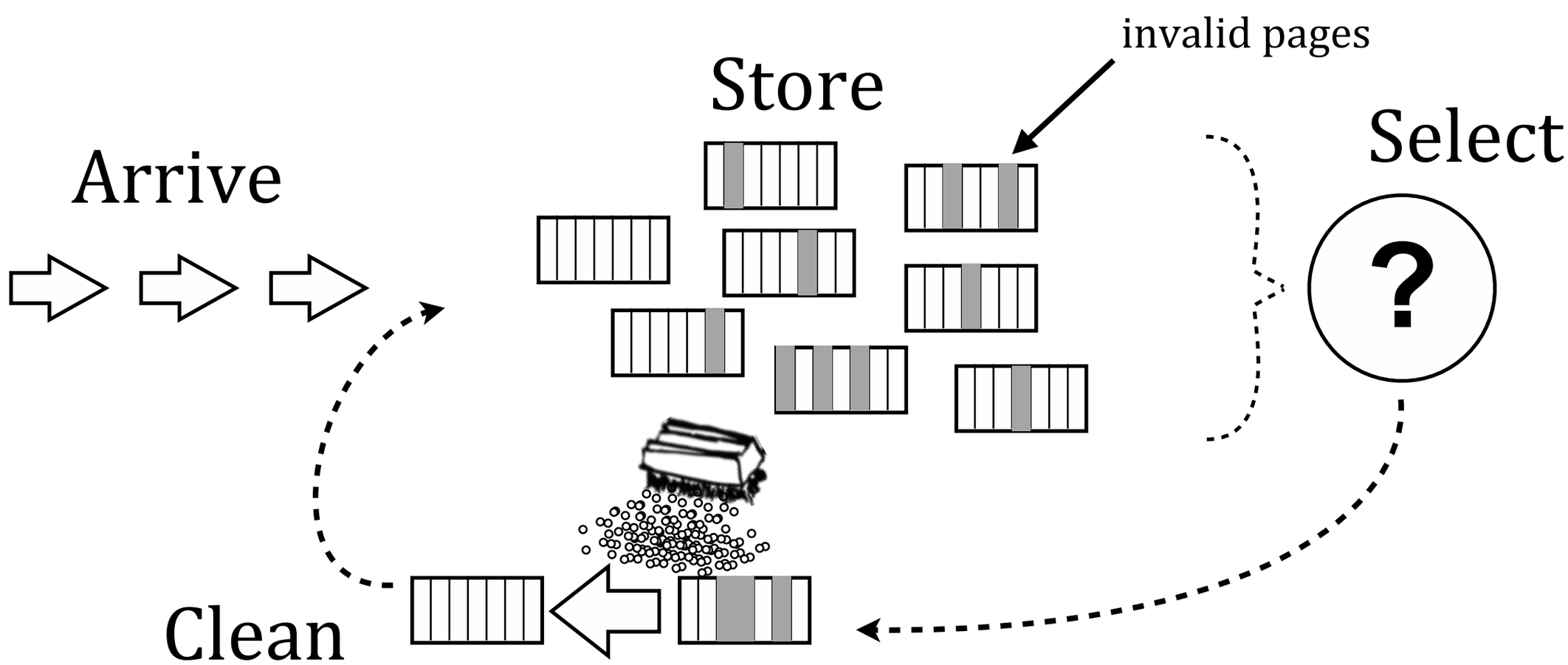
The FTL Model

- Random uniformly-distributed single-page writes
- **U** blocks of LBA space
- $\mathbf{N_p}$ pages per block
- **T** physical blocks



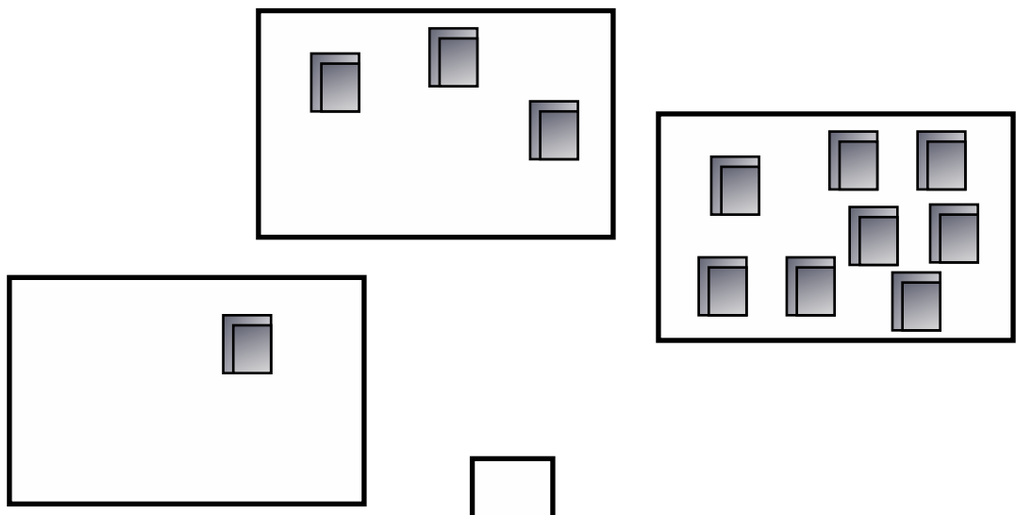
Simple cleaning model

- Writes arrive at rate 1
- Lazy cleaning (when blocks are needed for writes)
- Blocks selected for cleaning according to some rule **R**



Random Selection

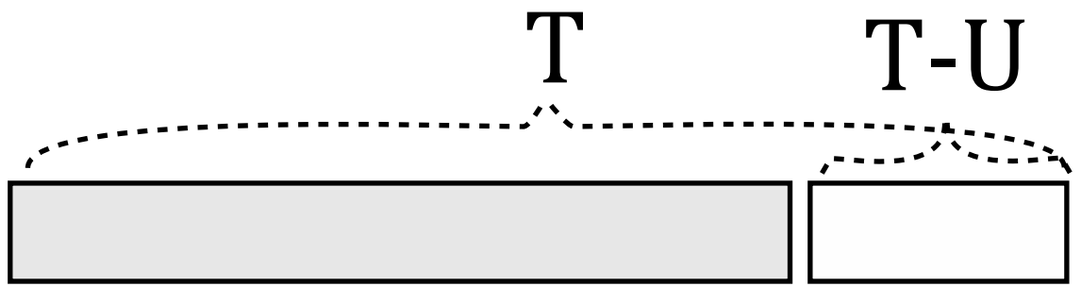
- Simplest to analyze
- Random sample sees population mean:
  - U · Np valid pages
    T · Np total pages
- Valid fraction = $\frac{U}{T}$
- Reclaimed frac. = $1 - \frac{U}{T} = S_f$



$$A = \frac{1}{S_f}$$

Measures of Free Space

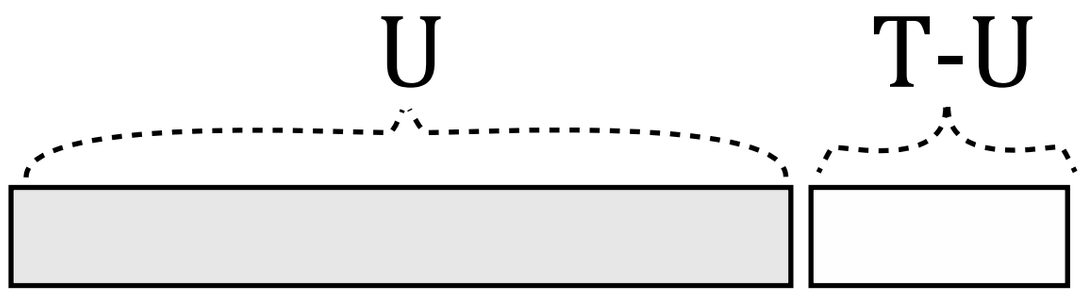
- Spare Fraction S_f



$$S_f = \frac{T - U}{T}$$

  - Spare blocks as a fraction of total blocks

- Overprovisioning factor O_f



$$O_f = \frac{T - U}{U}$$

  - Ratio of spare blocks to user-visible blocks

Typical Free Space Factors

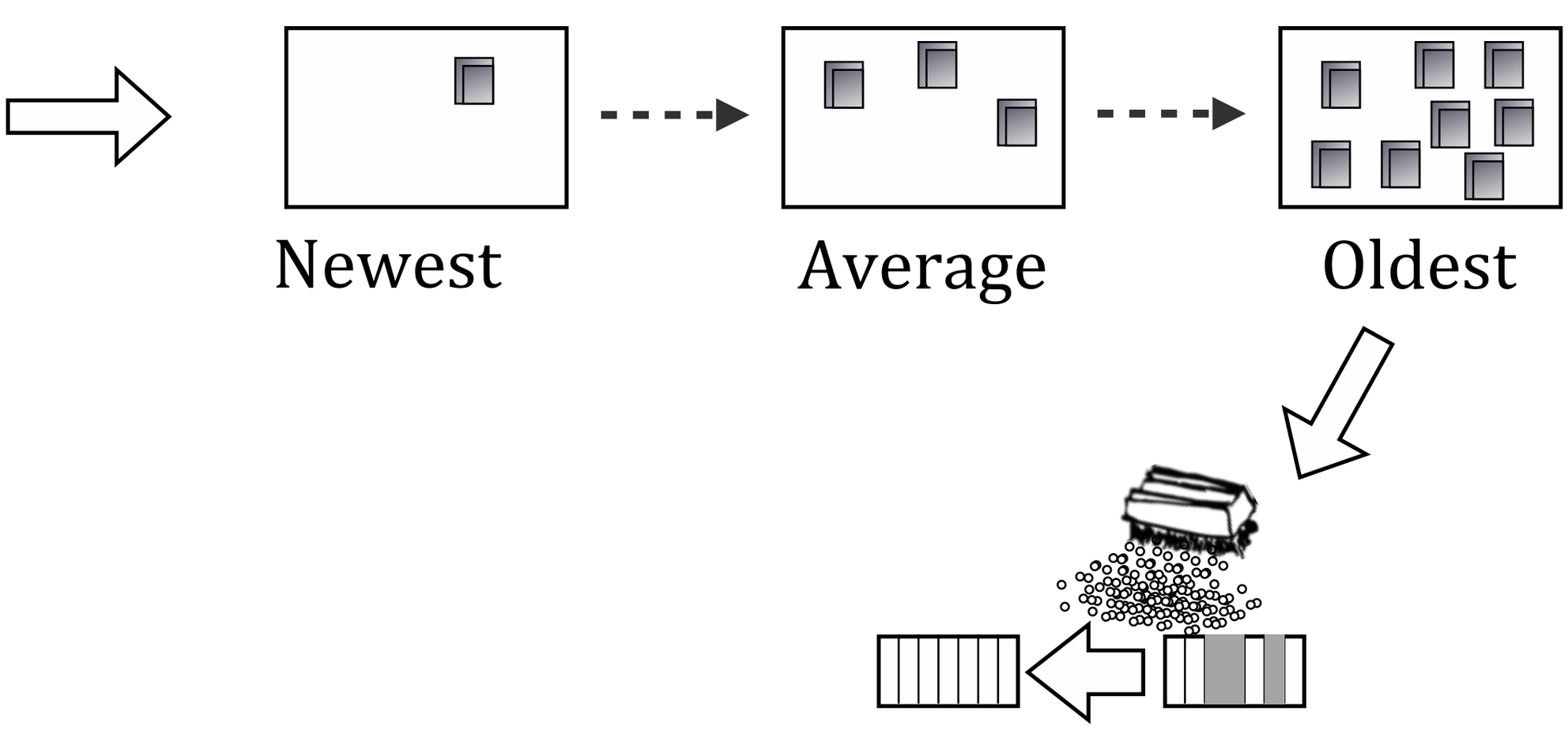
- Low-end SSDs: 7%

$$\frac{2^{30} - 10^9}{2^{30}} = 0.069$$

- Midrange: 25%
  - E.g. X25-E - 40 GiB flash, 32 GB LBA range
- High-end: ???
  - E.g. FusionIO

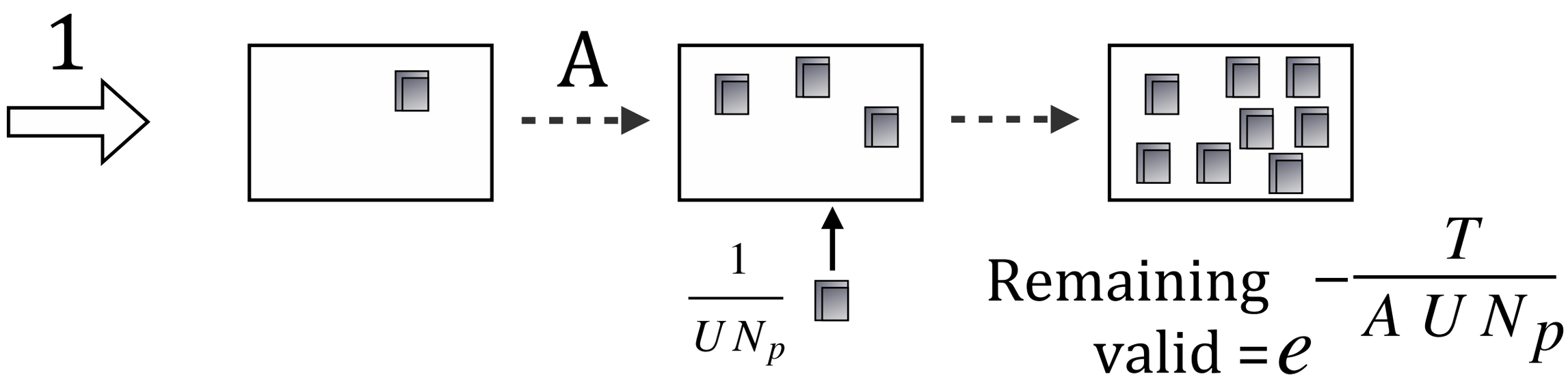
FIFO (LRU) Cleaning

- Select the oldest block to clean
  - has most expected free space



FIFO Analysis

- Assume 1 external write/sec
  - Pages are invalidated at rate $= \frac{1}{U N_p}$
- Queue moves at speed A
- Total time in queue $= \frac{T}{A}$

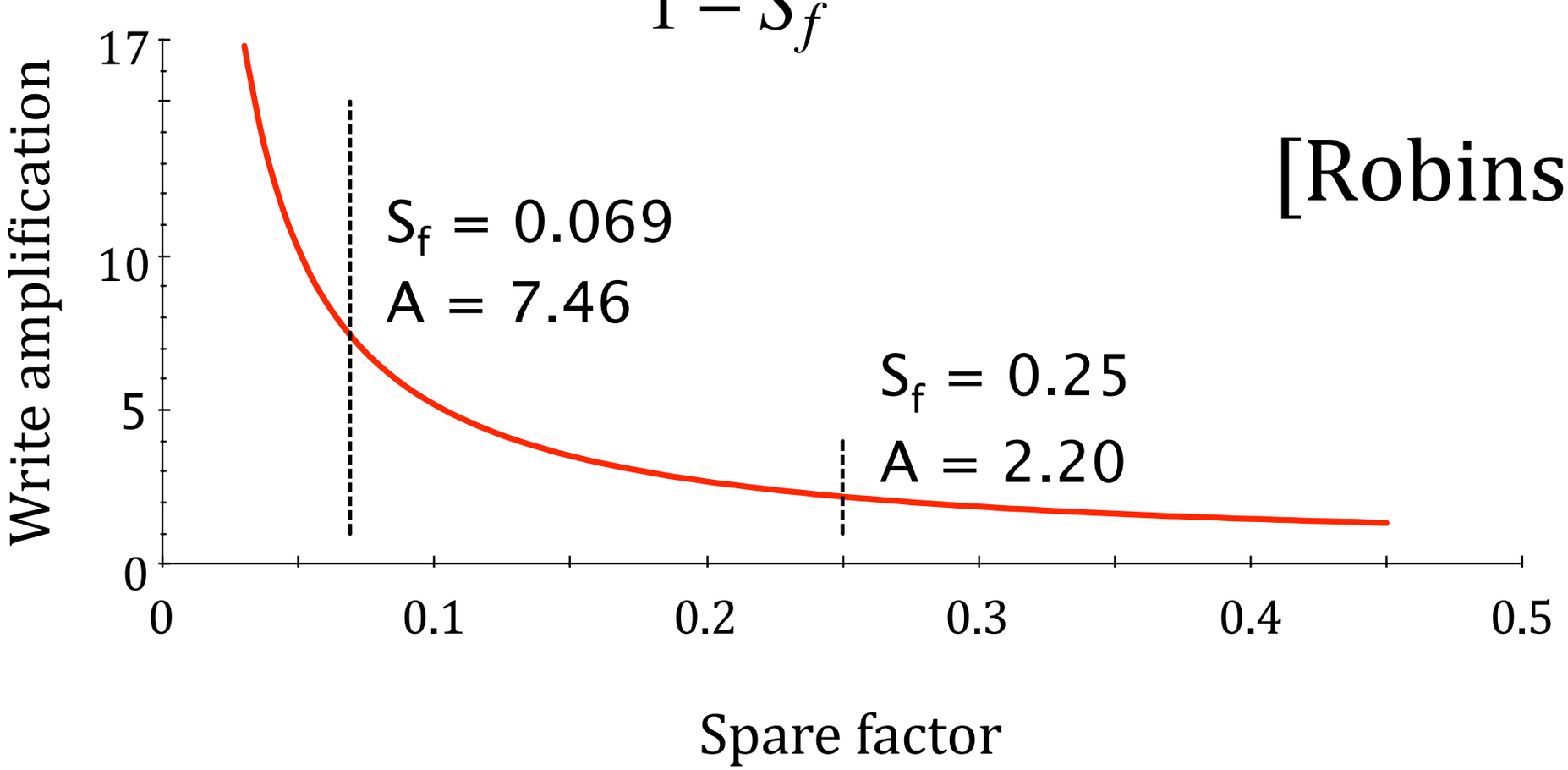


FIFO Performance

- Solution:

$$A = \frac{O_f}{W\left(O_f\left(-e^{-O_F}\right)\right) + O_f}$$

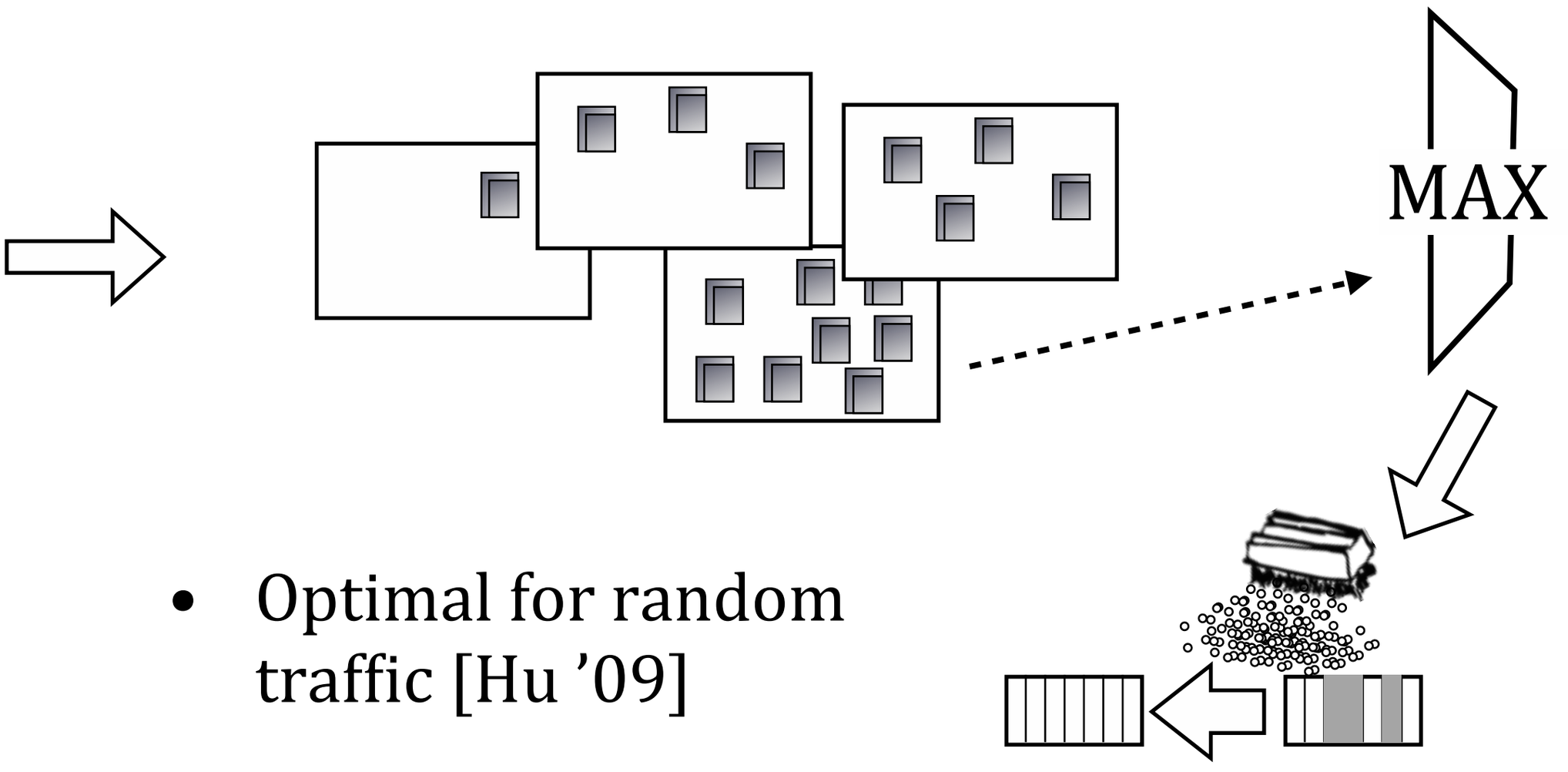
where $O_f = \dfrac{1}{1 - S_f}$, $W(x) = t \,|\, t\, e^t = x$



[Robinson '96]

Greedy Cleaning

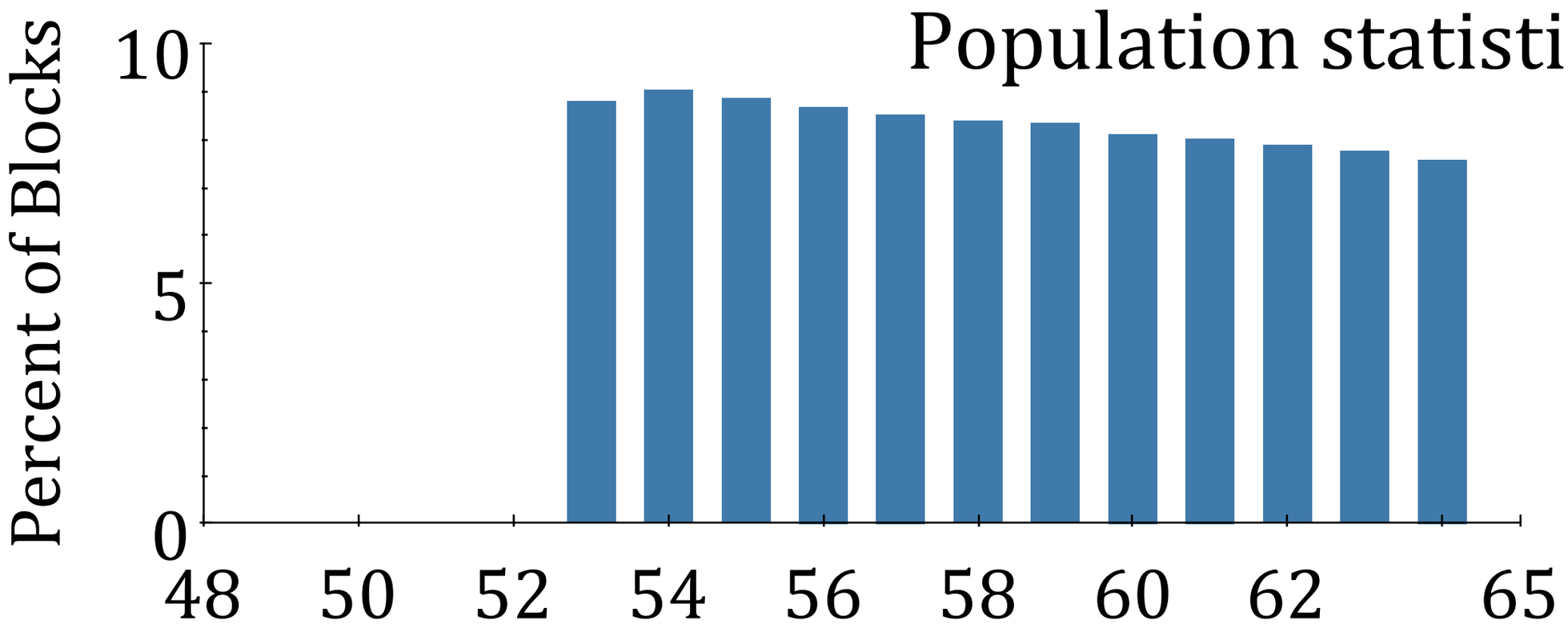
- Choose block with most free space
  - also called Cost/Benefit



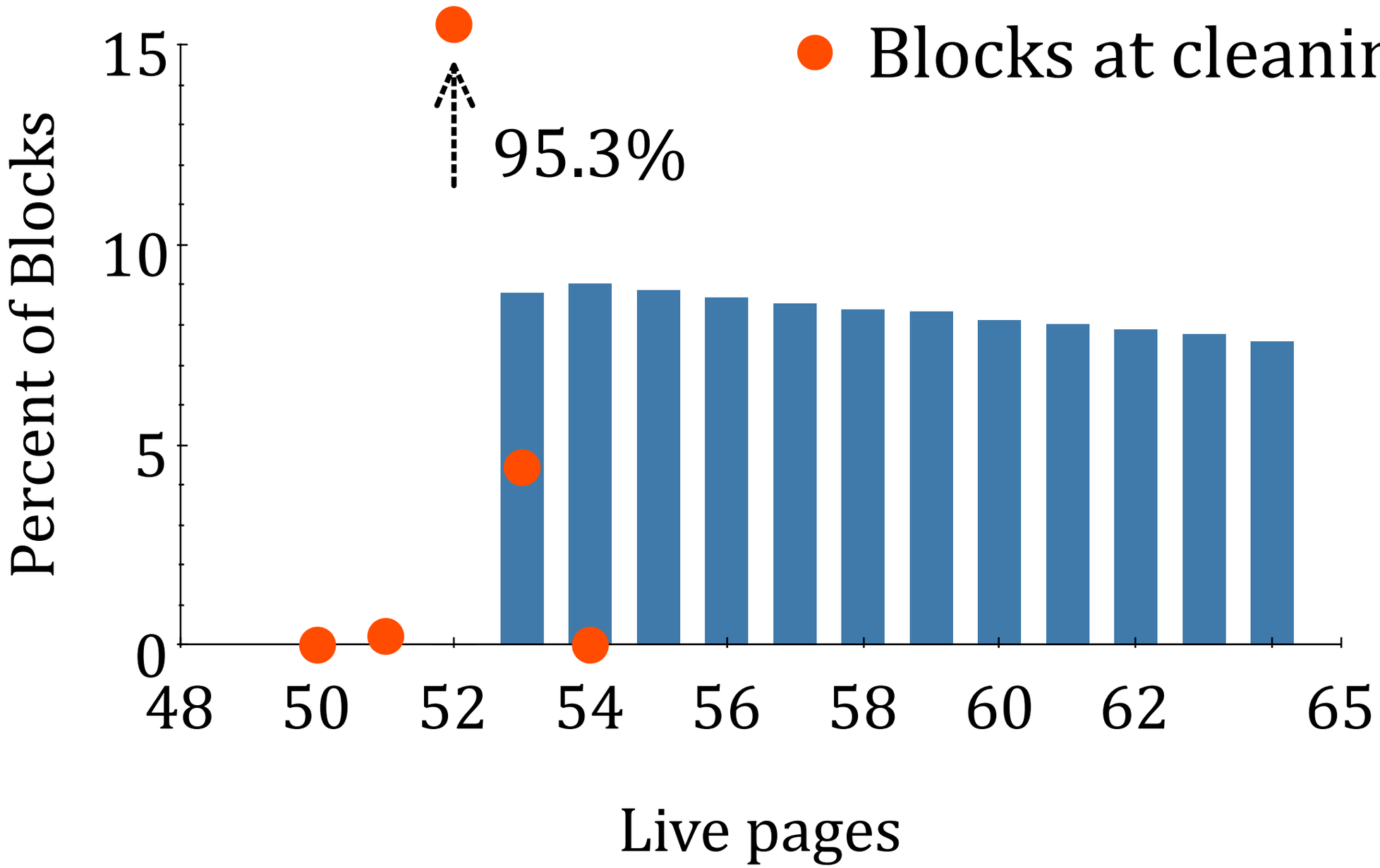
- Optimal for random traffic [Hu ’09]

Greedy Algorithm Behavior

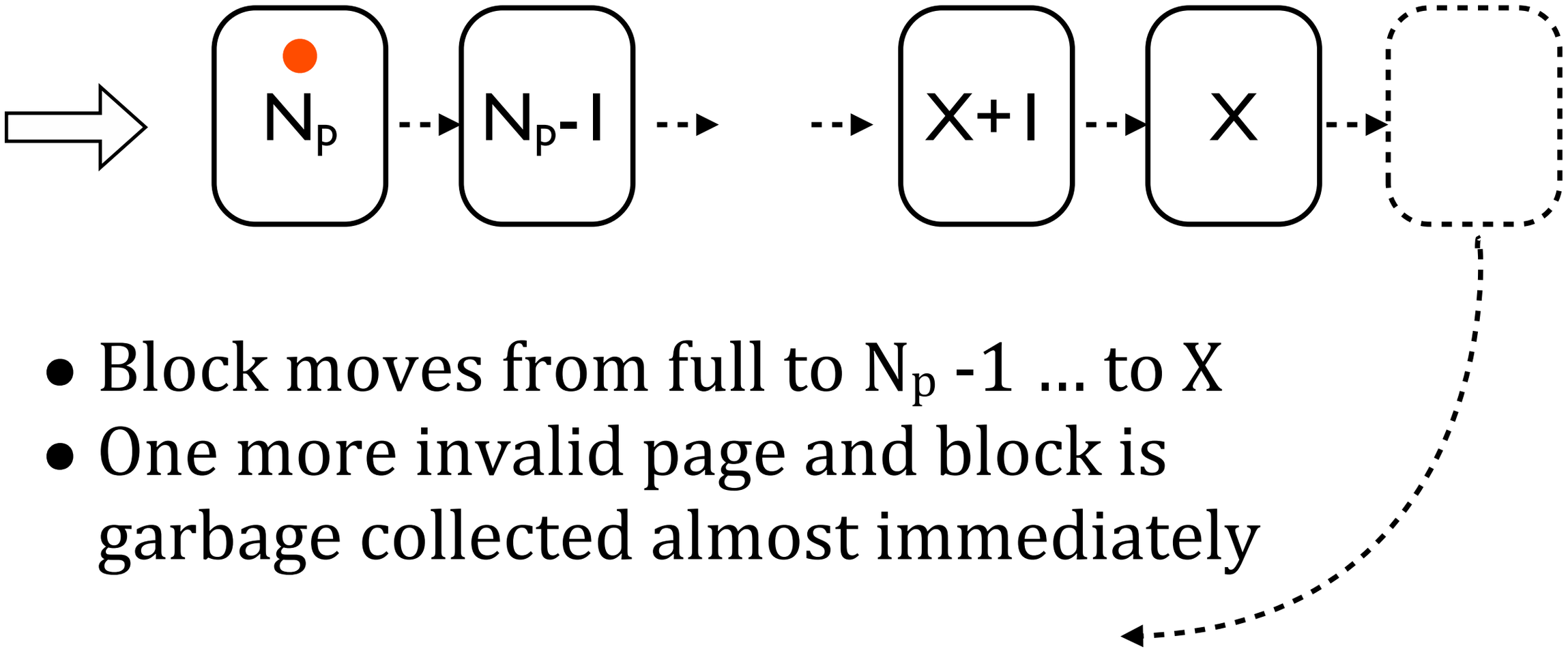
- Simulation with $S_f = 0.09$, $N_p = 64$



Greedy Algorithm Behavior



Markov Model for Greedy



- Block moves from full to N_p -1 … to X
- One more invalid page and block is garbage collected almost immediately

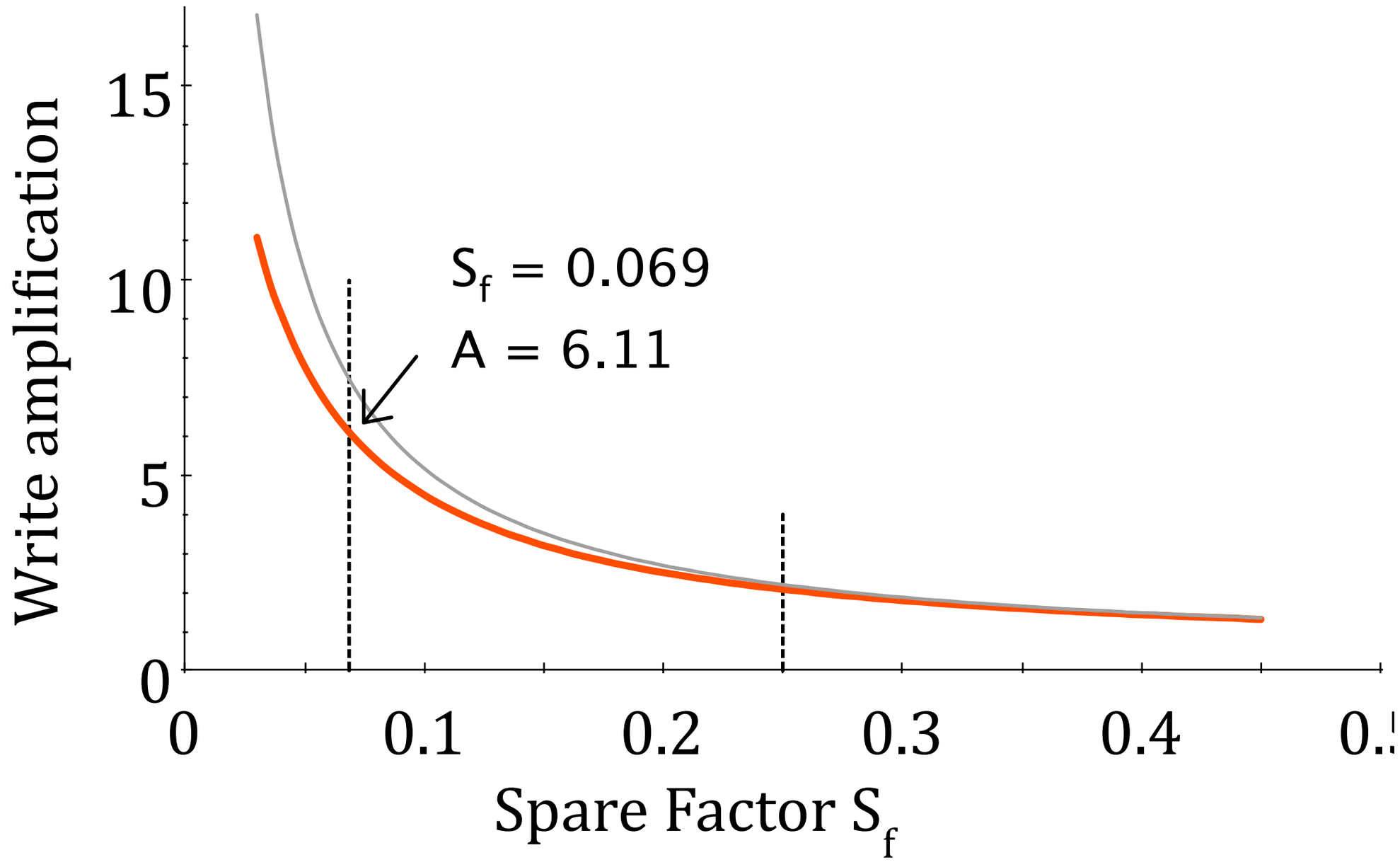
- Transition rate S→S-1 ∝ S

Greedy Cleaning Solution

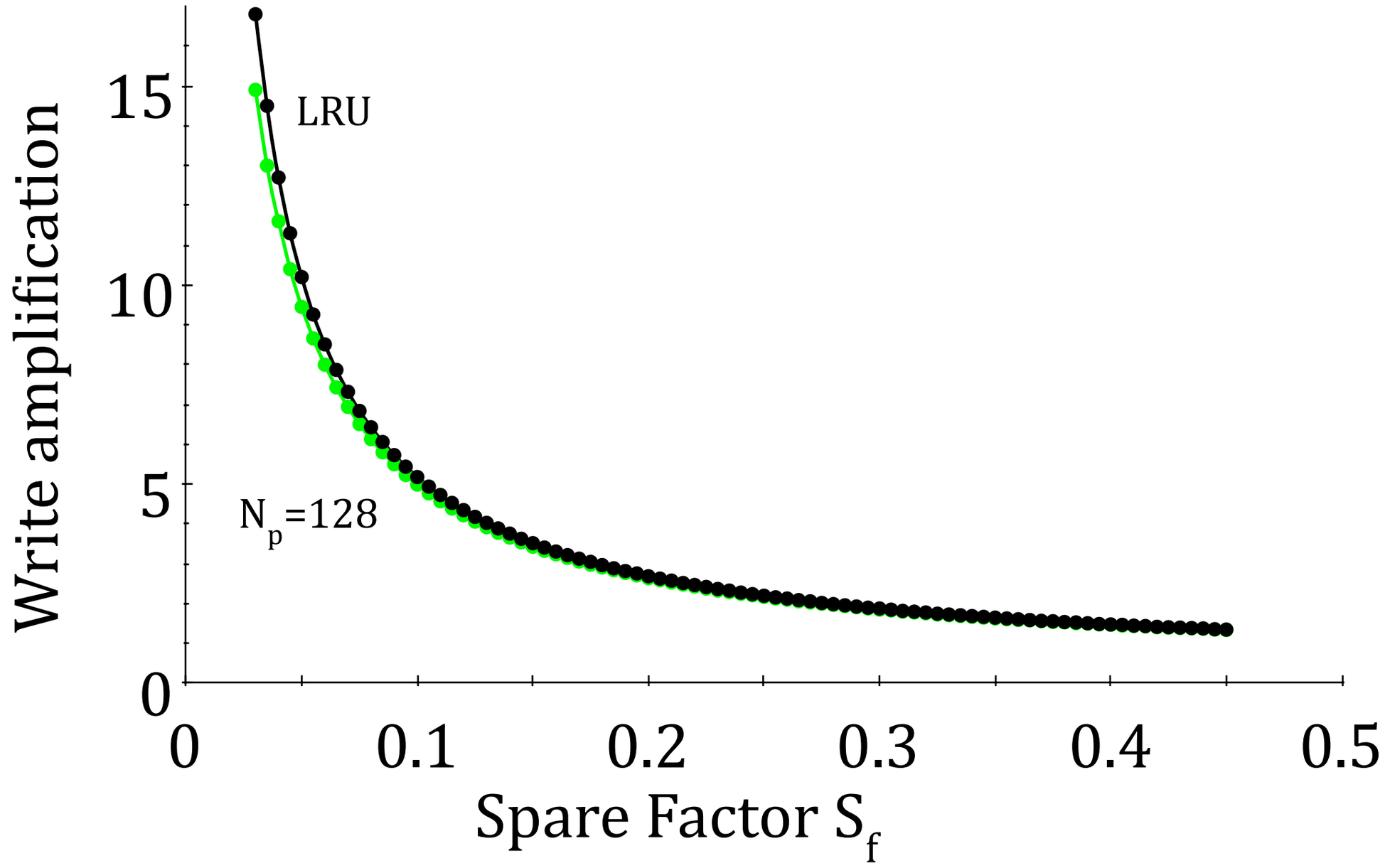
$$X_0 = \left(S_f - 1\right) N_p \, W\left(\frac{\left(e^{-\frac{1}{2\,N_p}-1}\left(N_p+\frac{1}{2}\right)^{1-S_f}\right)^{\frac{1}{1-S_f}}}{\left(S_f-1\right)N_p}\right)+\frac{1}{2}$$

$$A = \frac{N_p}{N_p - X_0 - 1}$$

Greedy Cleaning Performance



Greedy and Block Size



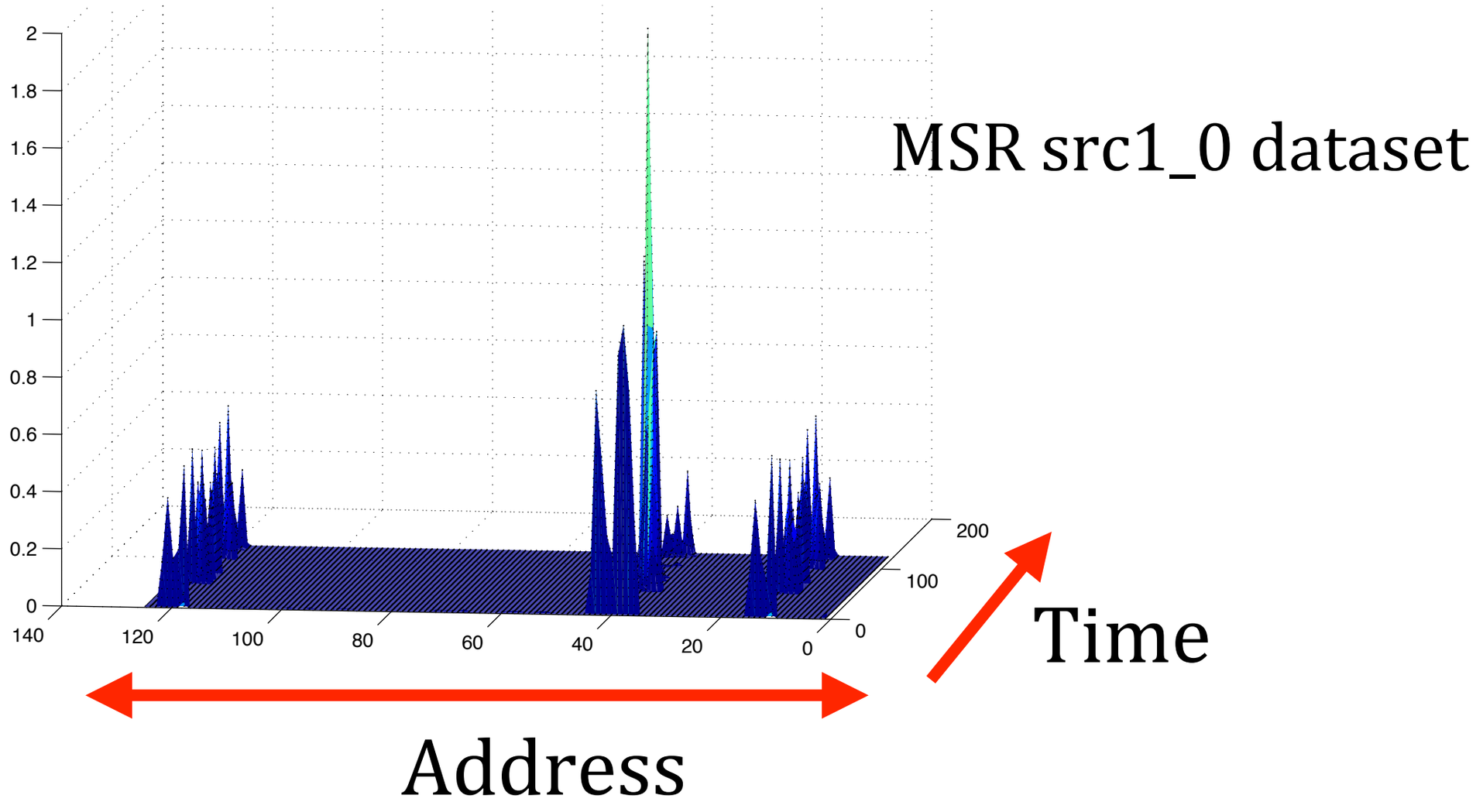
Greedy vs. LRU Cleaning

- Greedy works better when:
  1. Erase blocks are small, and
  2. There is little free space

- LRU is about the same when:
  1. Erase blocks are large (≥128) or
  2. Free space is available (≥15%)

Locality (Hot / Cold data)

- Real-world workloads aren't uniformly distributed and random



What do we know already?

- Separation of hot and cold data is related to improved performance for realistic workloads

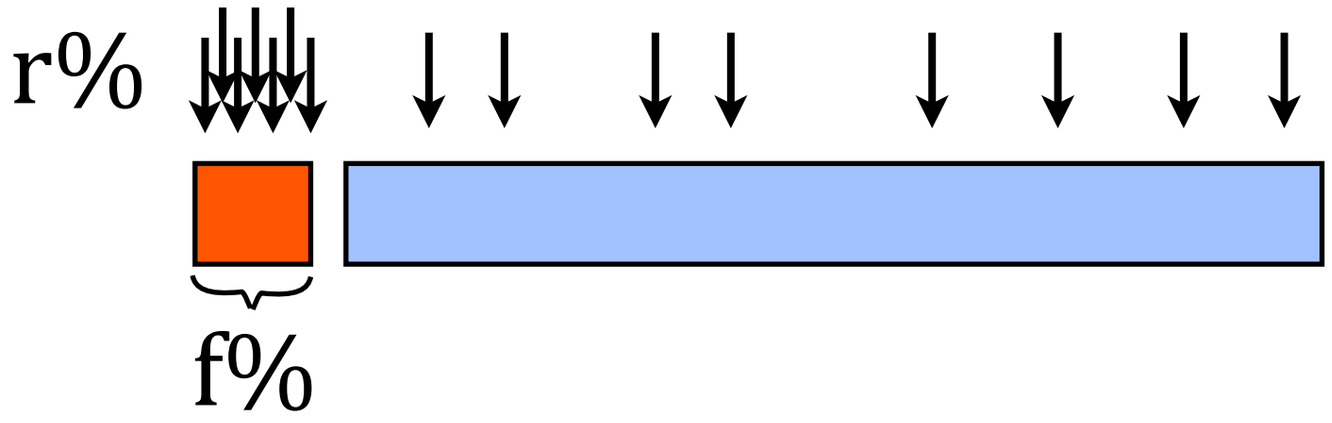
etc. Multi-hash
Cost/Age/Times
Bloom filters
Dual Pool
etc.
etc. Multi-queue

Goals

- Understand how locality impacts performance of naïve FIFO and Greedy cleaning
- Investigate how to extract performance gains when locality is present.

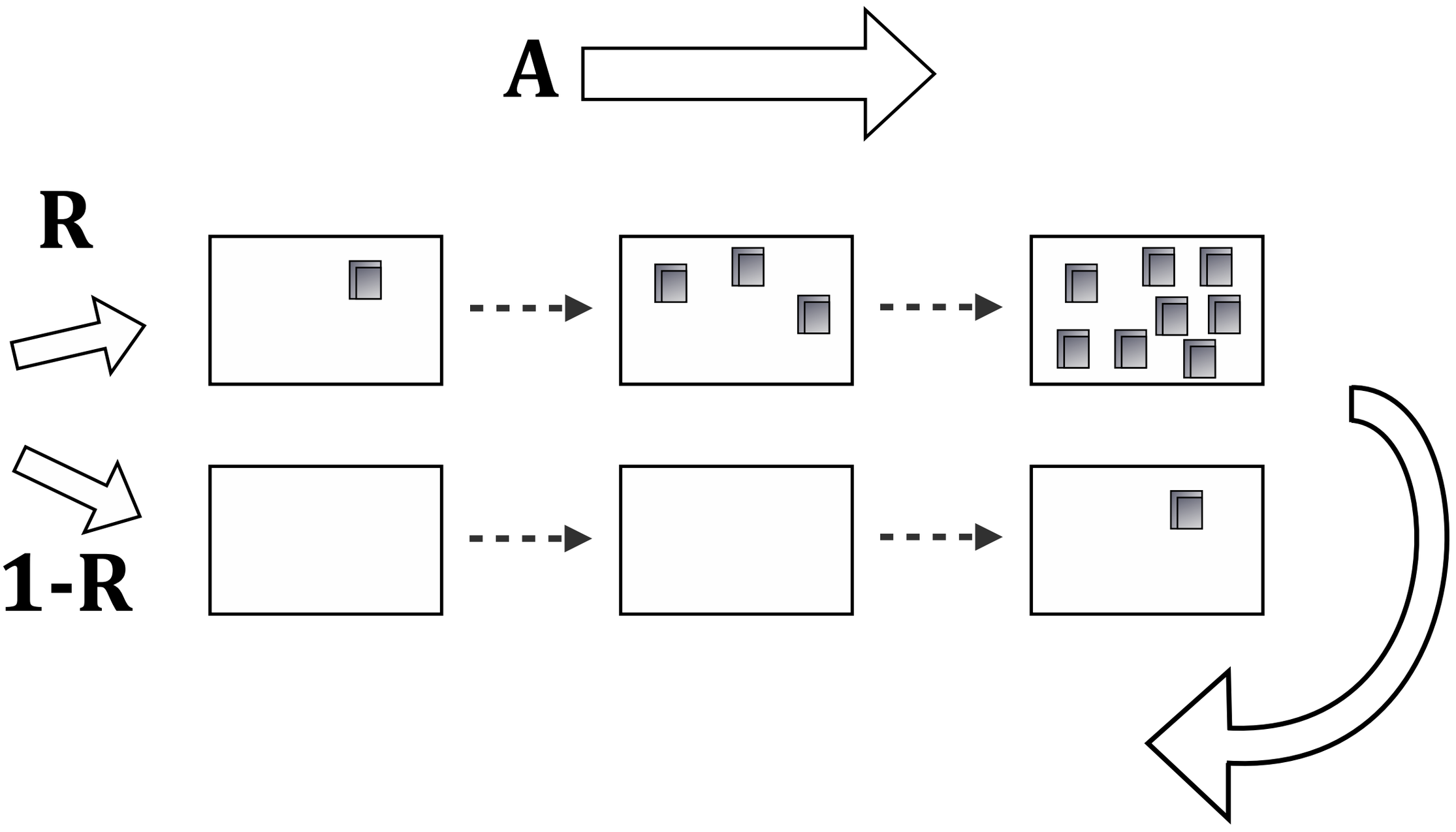
Traffic Model

- Simple hot/cold data:
  R% of writes to F% of LBA space



- E.g. 90% of writes to 10% of LBAs

FIFO Cleaning - Hot/Cold



FIFO Hot/Cold - Results

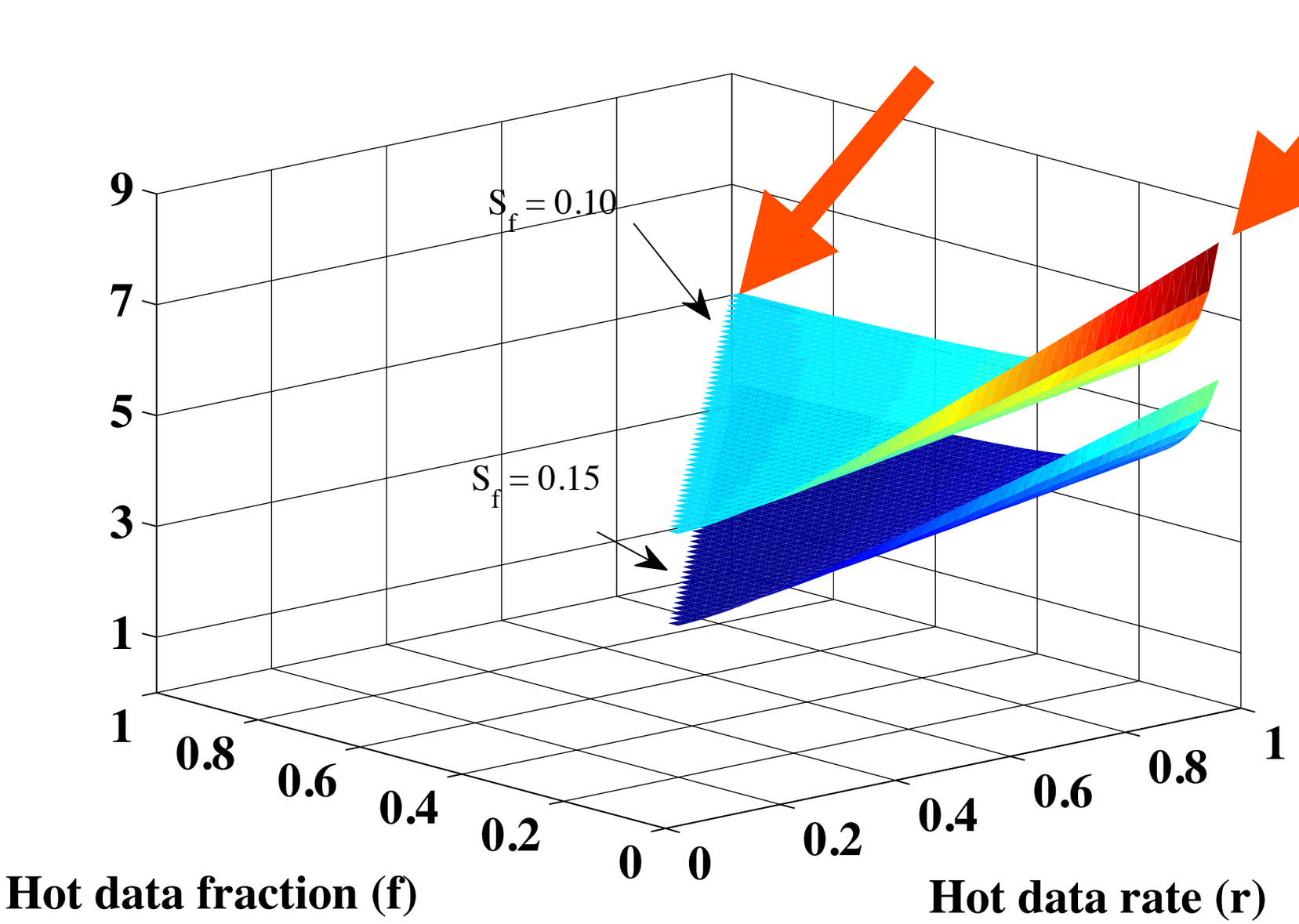
- Hot blocks move through system too slowly
- Cold blocks move through too quickly

$$A = r\,e^{-\frac{r\,O_a}{A\,f}} + \frac{1-r}{e^{\frac{(1-r)\,O_a}{A\,(1-f)}} - 1} + 1$$

- Solve numerically

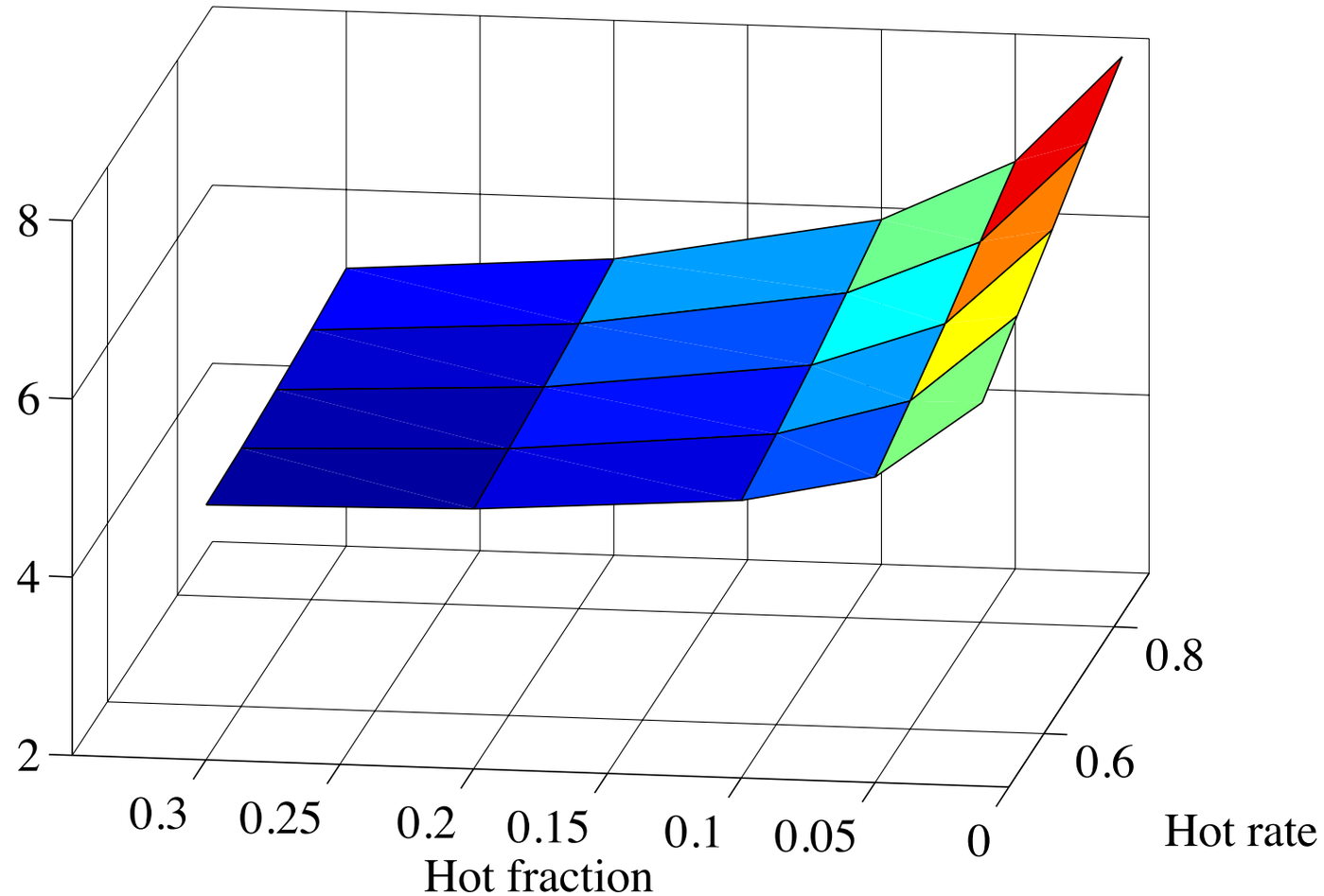
What does that mean?

almost 2x worst-case degradation



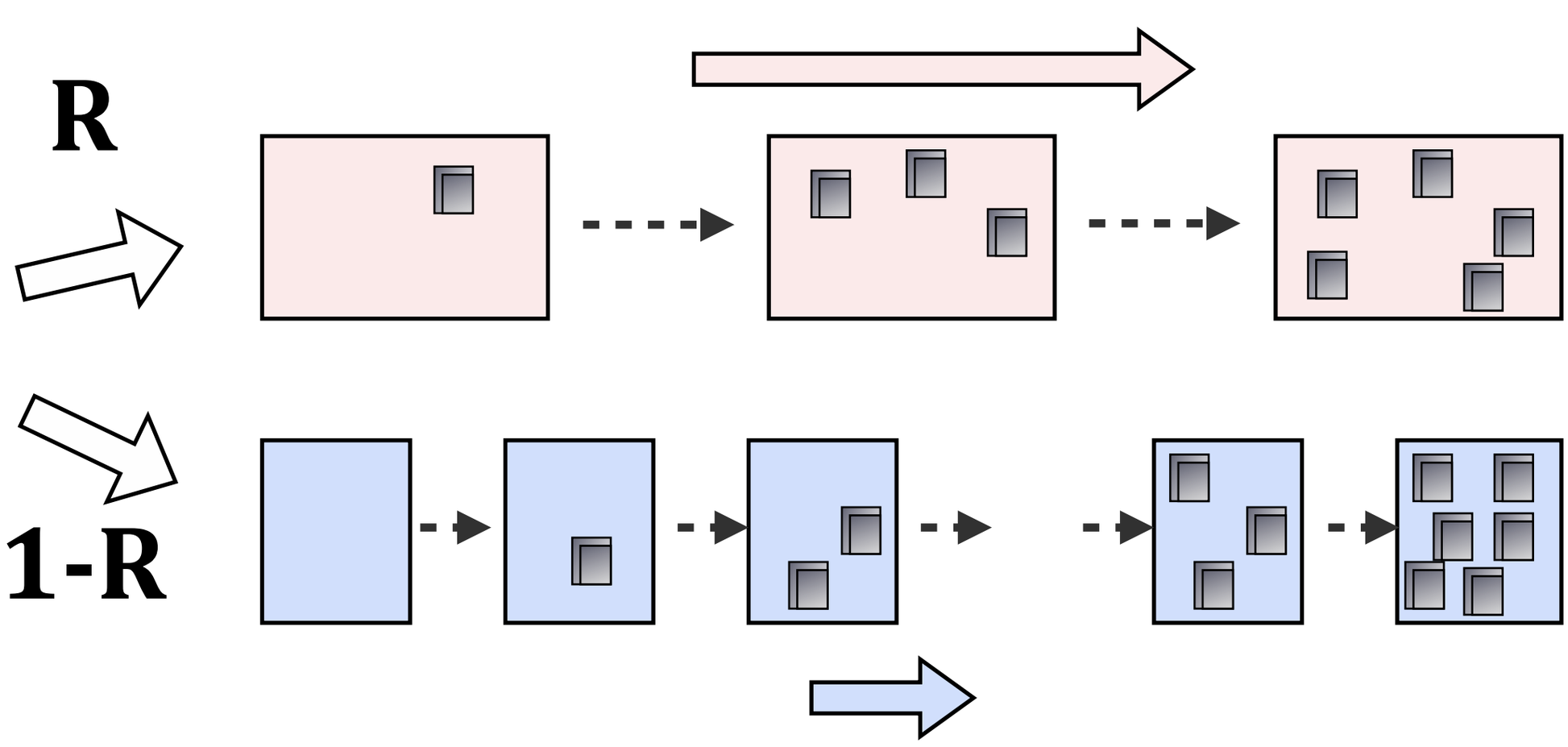
Greedy Cleaning - Hot/Cold

- Like FIFO, but more complicated
  - simulation results ($S_f = 0.10$, $N_p = 64$):



Hot/Cold Data Separation

- Assume perfect separation
- Greedy cleaning

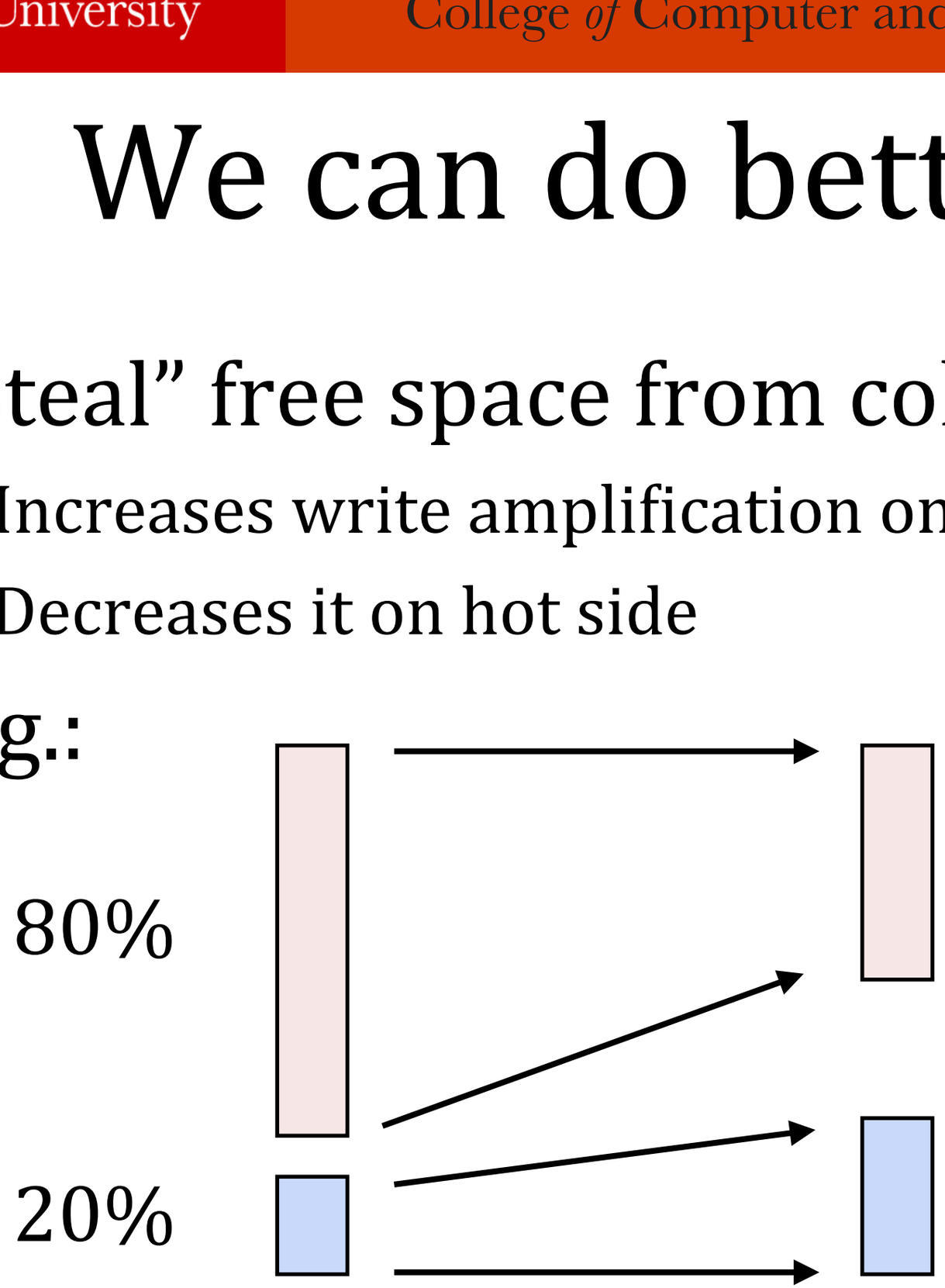


Result - original performance

- Hot and cold blocks have same number of free pages at cleaning
  - ⇒ Same write amplification
  - ⇒ Same spare ratio - since $A=f(S_f)$
  - ⇒ Same spare ratio as original uniform case
- Operates like two isolated FTLs
  - except tied together by global cleaning.

We can do better

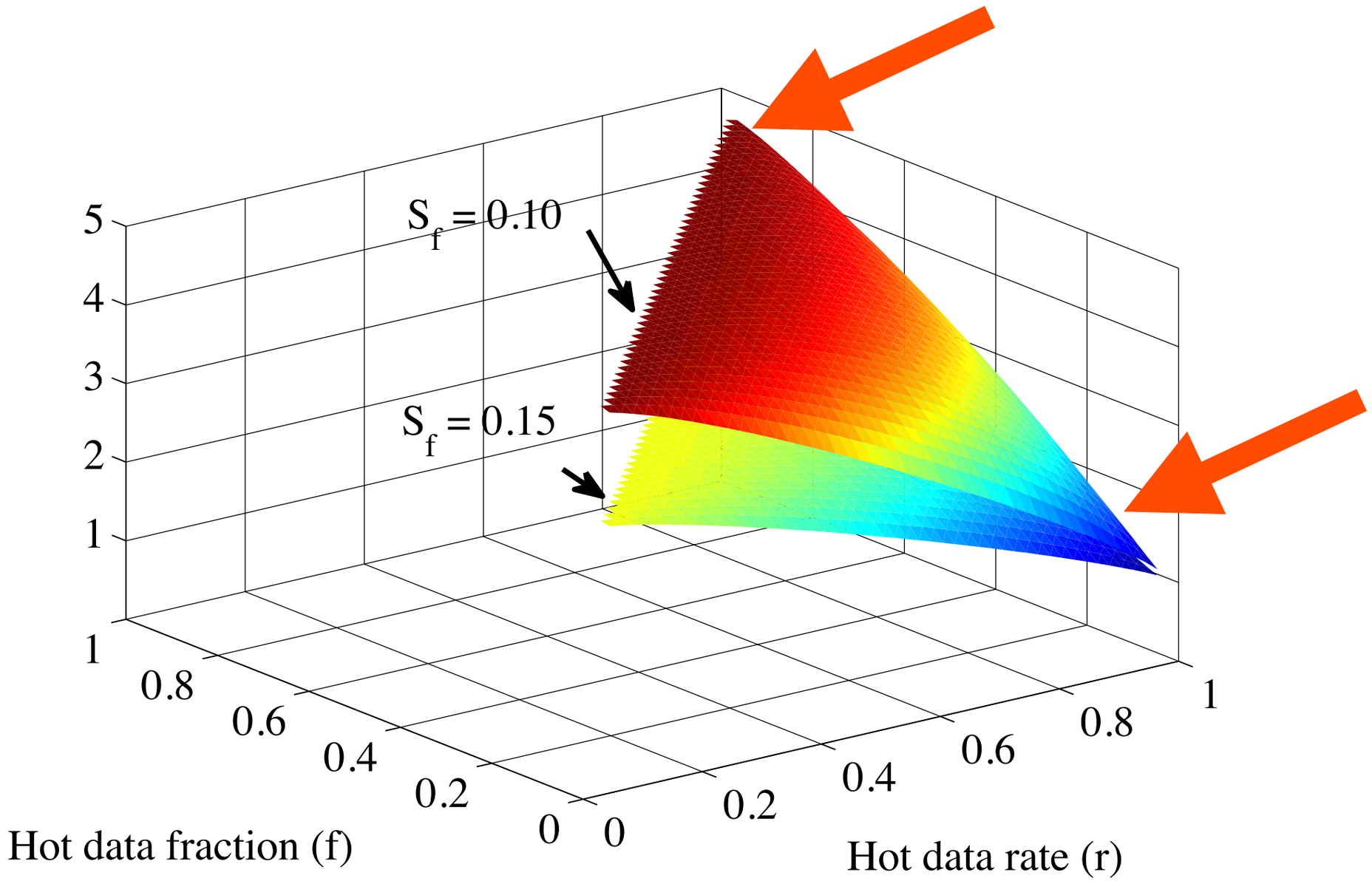
- "Steal" free space from cold blocks
  - Increases write amplification on cold side
  - Decreases it on hot side
- E.g.:



Optimal Hot/Cold FTL

- Assign free space unequally between hot and cold queues
  - Collect cold blocks with fewer free pages than hot blocks
- Artificial case
  - Find optimum free space assignment - $f(r,f,S_f)$
  - Hot/cold identification trivial for fake data
  - Choose hot/cold to clean based on imbalance from optimal free space assignment

Optimal Hot/Cold FTL



Ongoing and Future Work

- More complex traffic models
- Effect of inaccuracy in identifying hot data
- Efficient on-line algorithms for optimal division of free space
- OpenSSD implementations