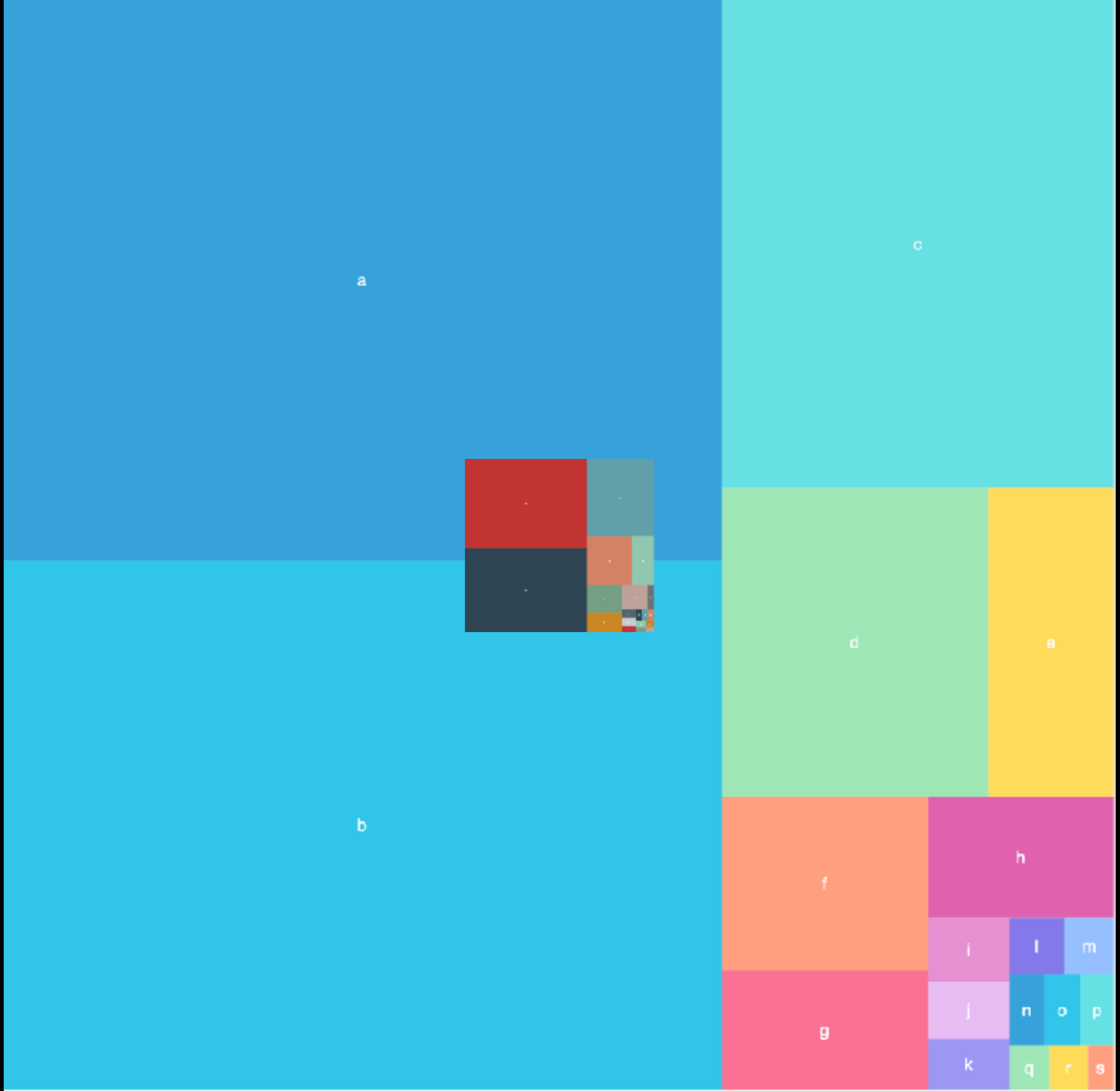
Cache Promote

Optimizing Cache Hit Rate and Disk Churn
for Big Working Set Sizes

Miles Libbey


a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s


a
b
c
d
e
f
g
h
i
j
k
l
m
n
o
p
q
r
s

Phenomenal Cosmic Power!!
Itty Bitty Living Space

Big Working Set, Small Cache

- Disk churn increases — SSD life concerns
- Popular objects churned out of cache —> Cache hit rate suffers

Cache Promote Plugin

Don’t just cache everything the origin tells you to

Cache Promote Plugin

Don’t tell me how to live my life

Decision Policies

- Random Chance
- Number of hits

Random Chance

- Popular objects get many tries
- 1 hit wonders, get 1 shot — Some make it in.

Number of hits

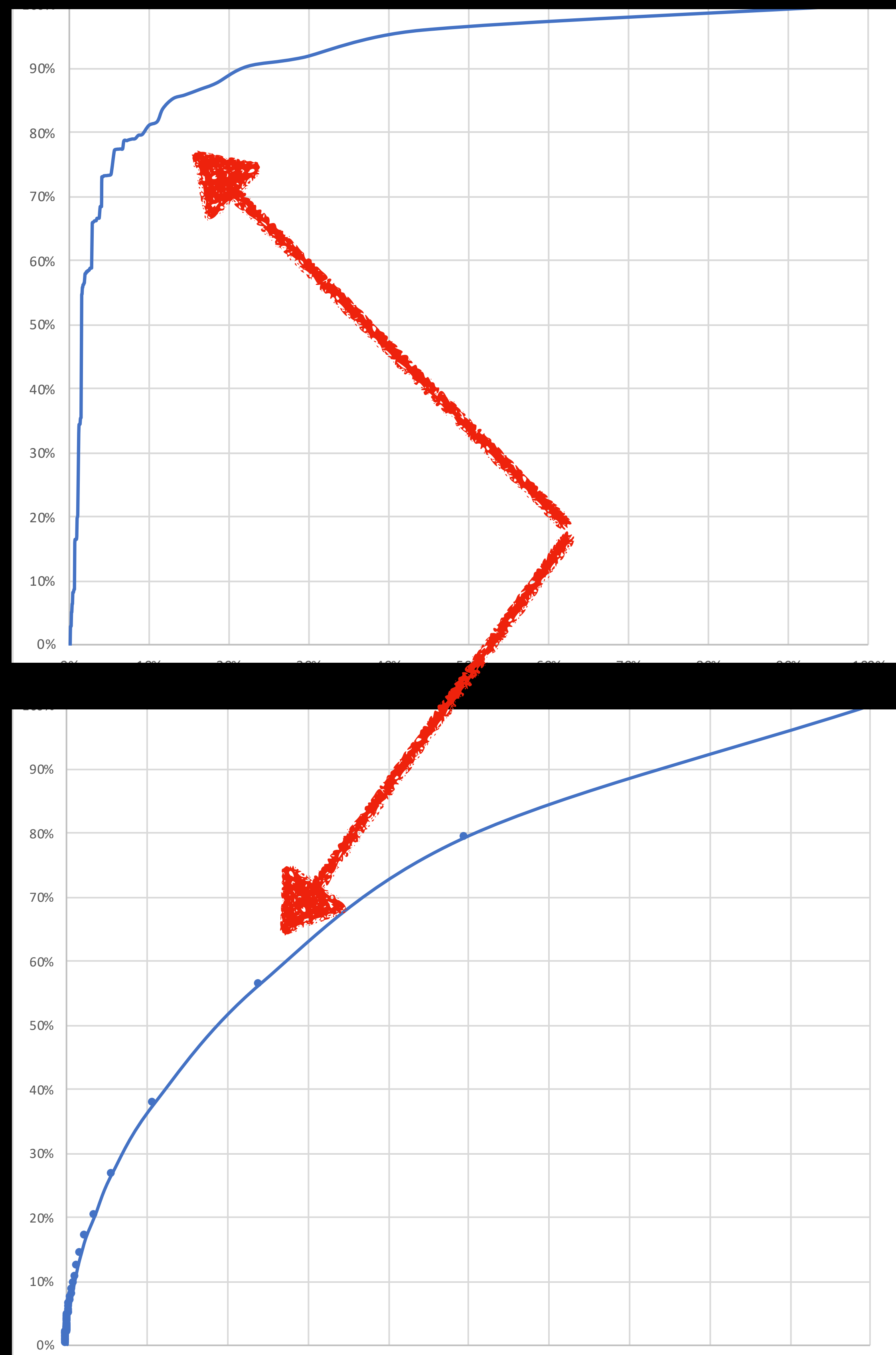
- Track each inbound url, with the number of hits its gotten
- Table is limited to N urls — “Bucket Size”
- Table eviction policy — Least Recently Used
- When the url gets X hits, allow it into cache


SOLD

But, what should I use for the number of hits?


OMG IT'S R2D2!
I LOVED HIM IN
STAR TREK

Lets try the “knee of the curve”

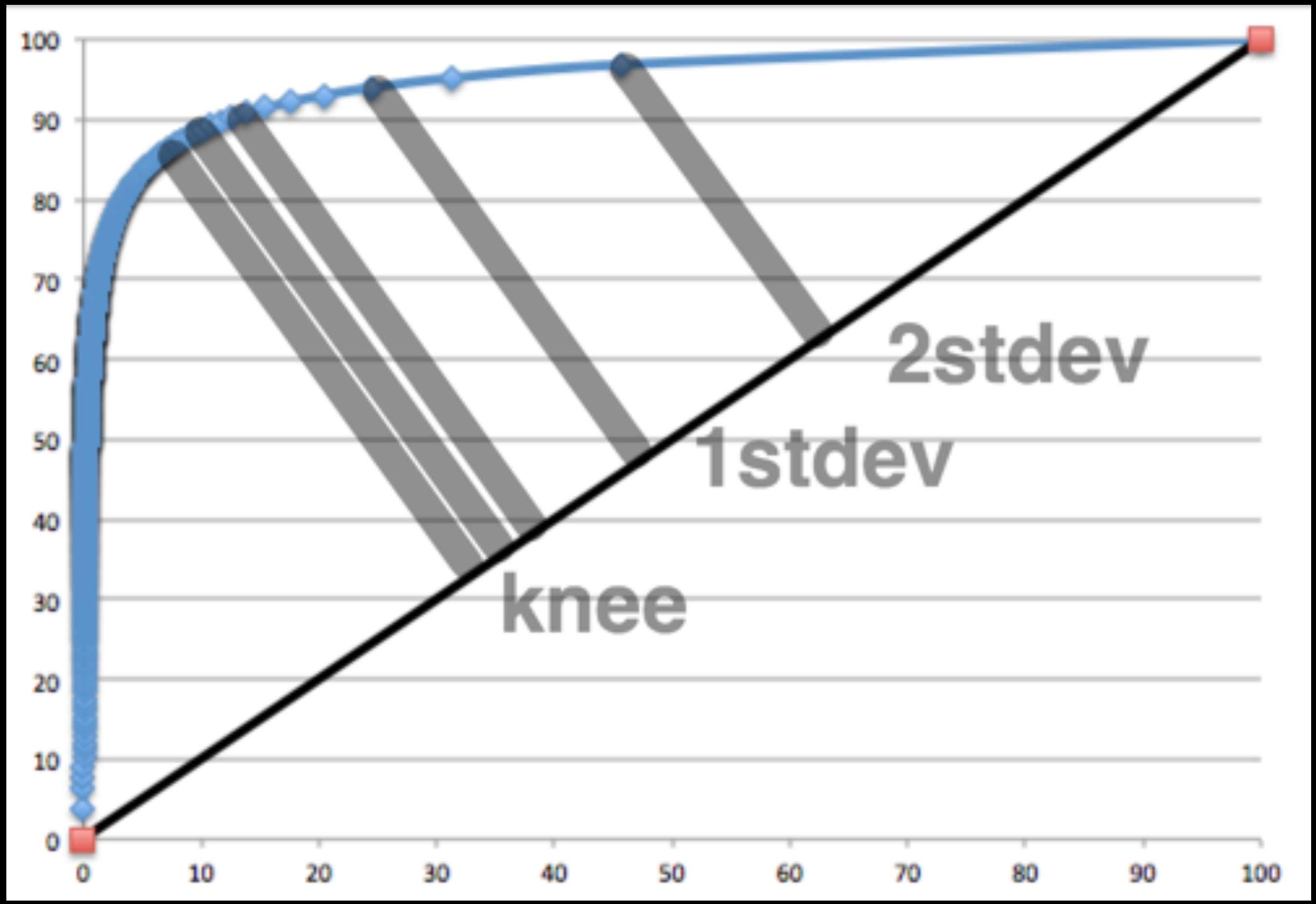


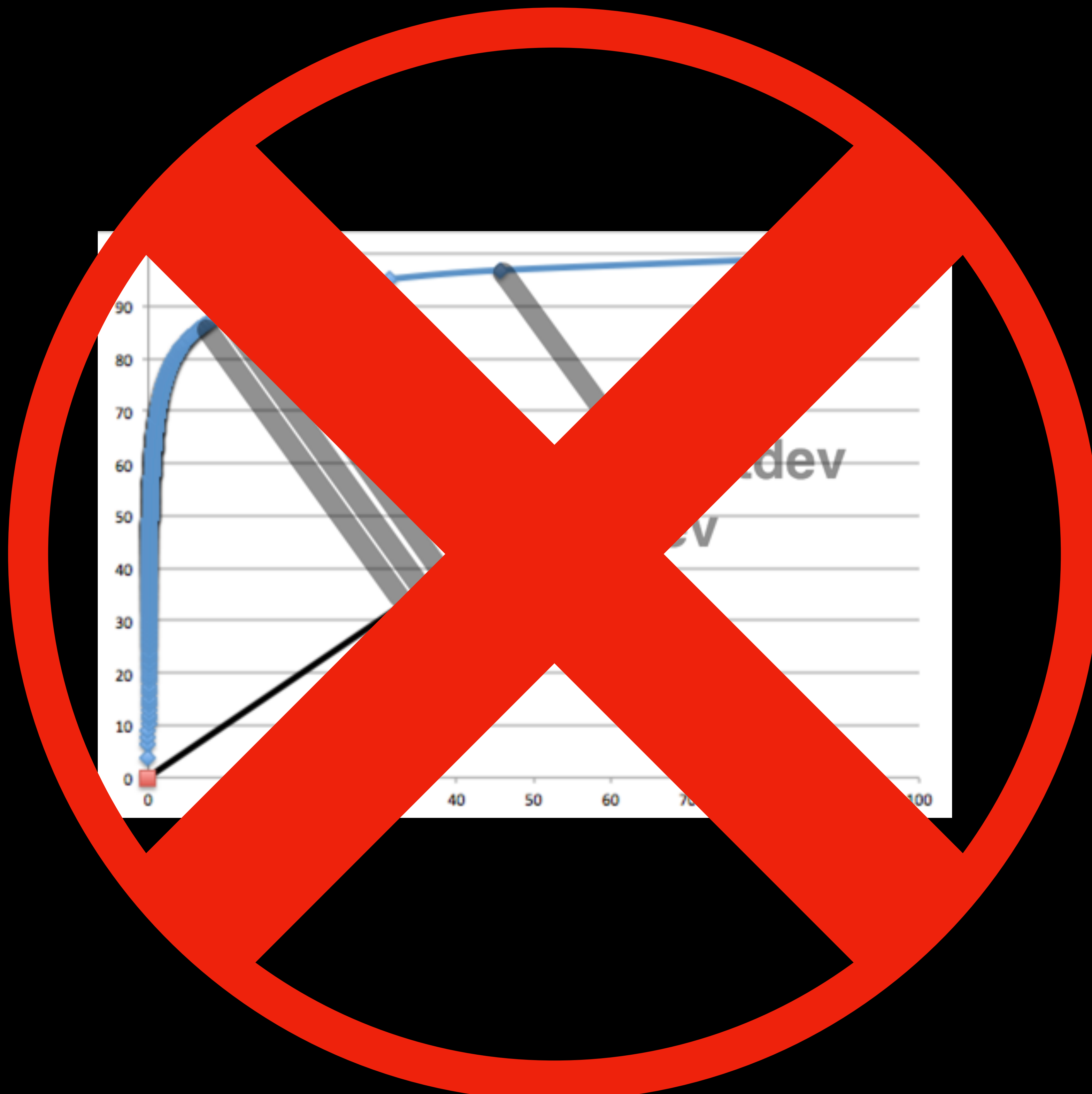
Lets try the “knee of the curve”

Data Gathering … for each property on a cache node

- Group URLs by number of hits
- Get the sum of their file sizes
- Sort by the number of hits descending
- Find the Cumulative File Size and Requests for each row

	A	B	C	D	E	F	G	H	I
1	**Hits**	**#urls**	**Urls (Cum.)**	**Total Reqests**	**Requests (Cum.)**	**Total Size (GB)**	**Cache Size (GB)**	**Bandwidth**	**Bandwidth (Cum.)**
2	2444	1	1	2444	2444	0.01115	0.0111509	27.2527	27.2527
3	2294	1	2	2294	4738	0.01072	0.0218669	24.5827	51.8354
4	1772	1	3	1772	6510	0.00884	0.0307101	15.6701	67.5055
5	1714	1	4	1714	8224	0.00930	0.0400113	15.9423	83.4478
6	1712	1	5	1712	9936	0.00847	0.0484776	14.4942	97.942
7	1650	1	6	1650	11586	0.00934	0.0578146	15.4061	113.348
76	22	258	1623	5676	80202	4.48084	20.4589	98.5786	916.444
77	20	372	1995	7440	87642	5.31028	25.7692	106.206	1022.65
78	18	515	2510	9270	96912	8.68765	34.4568	156.378	1179.03
79	16	808	3318	12928	109840	12.13733	46.5941	194.197	1373.22
80	14	1173	4491	16422	126262	17.89602	64.4902	250.544	1623.77
81	12	2078	6569	24936	151198	26.38207	90.8722	316.585	1940.35
82	10	3956	10525	39560	190758	61.04712	151.919	610.471	2550.82
83	8	10000	20525	80000	270758	130.78821	282.708	1046.31	3597.13
84	6	25035	45560	150210	420968	295.99959	578.707	1776	5373.13
85	4	48635	94195	194540	615508	545.31410	1124.02	2181.26	7554.38
86	2	95736	189931	191472	806980	977.62673	2101.65	1955.25	9509.64


100
90
80
70
60
50
40
30
20
10
0
0
10
20
30
40
50
60
70
80
90
100
2stdev
1stdev
knee



90
80
70
60
50
40
30
20
10
0
0
40
50
60
100

No knees?

- Cache Hit change wasn’t repeatable
- Churn wasn’t repeatable
- Why?
  - Didn’t consider the cache size
  - Optimized each individual property, not the total cache


EASTPAK
EASTPAK
EASTPAK
EASTPAK
EASTPAK
EASTPAK

Knapsack problem

From Wikipedia, the free encyclopedia

The **knapsack problem** or **rucksack problem** is a problem in combinatorial optimization: Given a set of items, each with a weight and a value, determine the number of each item to include in a collection so that the total weight is less than or equal to a given limit and the total value is as large as possible. It derives its name from the problem faced by someone who is constrained by a fixed-size knapsack and must fill it with the most valuable items.

The problem often arises in resource allocation where there are financial constraints and is studied in fields such as combinatorics, computer science, complexity theory, cryptography, applied mathematics, and daily fantasy sports.

The knapsack problem has been studied for more than a century, with early works dating as far back as 1897.[1] The name "knapsack problem" dates back to the early works of mathematician Tobias Dantzig (1884–1956),[2] and refers to the commonplace problem of packing the most valuable or useful items without overloading the luggage.

- Best combination of things to put in your knapsack
- Maximize value [Bandwidth]
- Without going over [cache size]

Almost…

- Doesn’t quite fit classic problem
    - Assumes item independence

Customize it

- Use the same data files from knees
- Test “all” the combinations
- First line from 1st file,
    - with first line of 2nd file …
        - with first line of last
        - with second line of last
    - …
- Add the bandwidths together, and track bandwidth records
    - Final record is the answer
- Ignore all combinations where the sum of the cache sizes is too big

A few optimizations — reduce the combinations

- Only look at big properties
- Aggregate the onsie-twosie lines together — to get some minimal cache size per line

	A	B	C	D	E	F	G	H	I
1	Hits	#urls	Urls (Cum.)	Total Reqests	Requests (Cum.)	Total Size (GB)	Cache Size (GB)	Bandwidth	Bandwidth (Cum.)
2	2444	1	1	2444	2444	0.01115	0.0111509	27.2527	27.2527
3	2294	1	2	2294	4738	0.01072	0.0218669	24.5827	51.8354
4	1772	1	3	1772	6510	0.00884	0.0307101	15.670[illegible]	67.5055
5	1714	1	4	1714	8224	0.00930	0.0400113	15.942[illegible]	83.447[illegible]
6	1712	1	5	1712	9936	0.00847	0.0484776	14.4942	97.942
7	1650	1	6	1650	11586	0.00934	0.0578146	15.4061	113.348
76	22	258	1623	5676	80202	4.48084	20.4589	98.5786	916.444
77	20	372	1995	7440	87642	5.31028	25.7692	106.206	[illegible]
78	18	515	2510	9270	96912	8.68765	34.4568	156.378	1179.03
79	16	808	3318	12928	109840	12.13733	46.5941	194.197	1373.22
[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]	[illegible]

Group these!

And these!

Buckets/Table Size

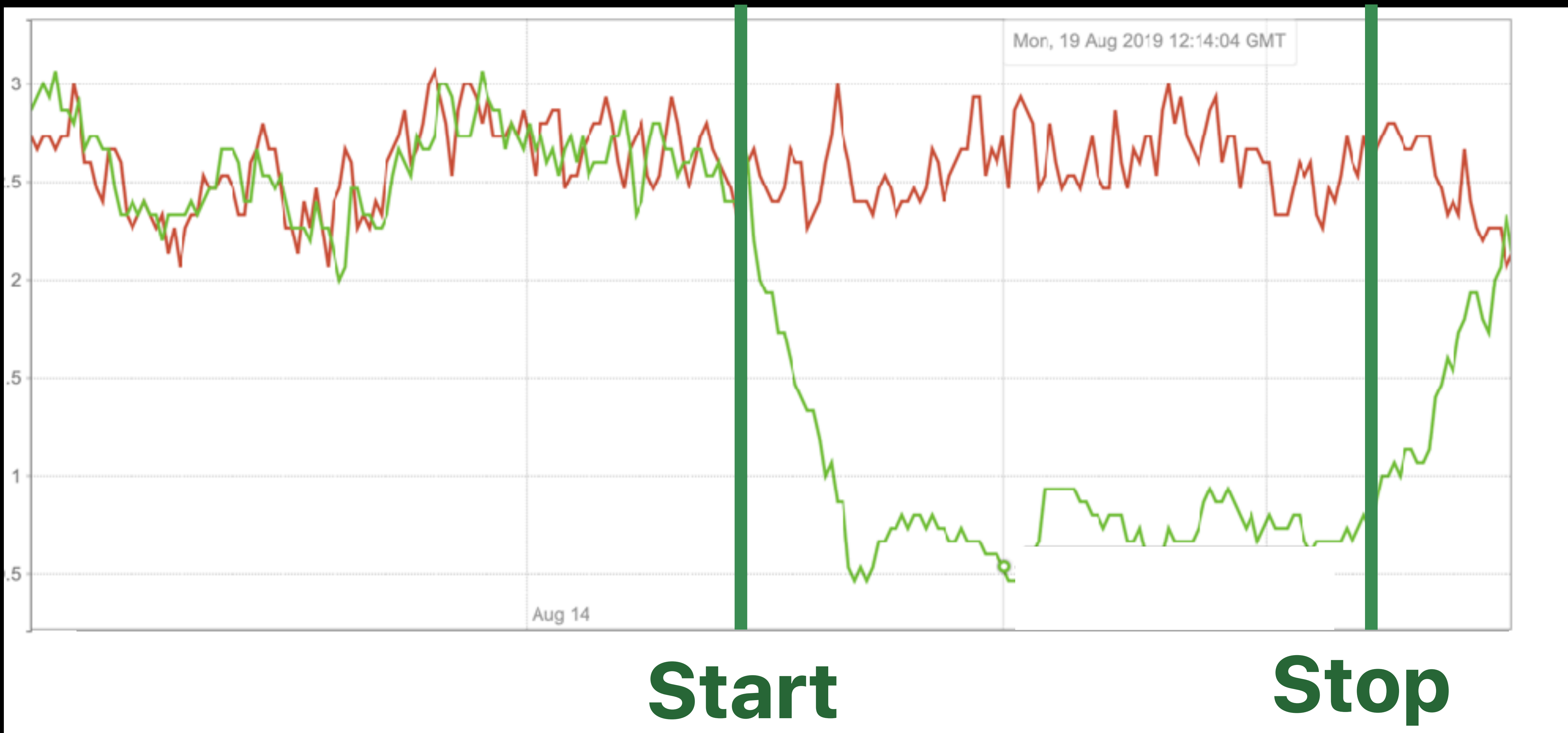

OMG IT'S R2D2!
I LOVED HIM IN
STAR TREK

Perhaps the number of unique urls the property sees in the desired churn time?

Experiment

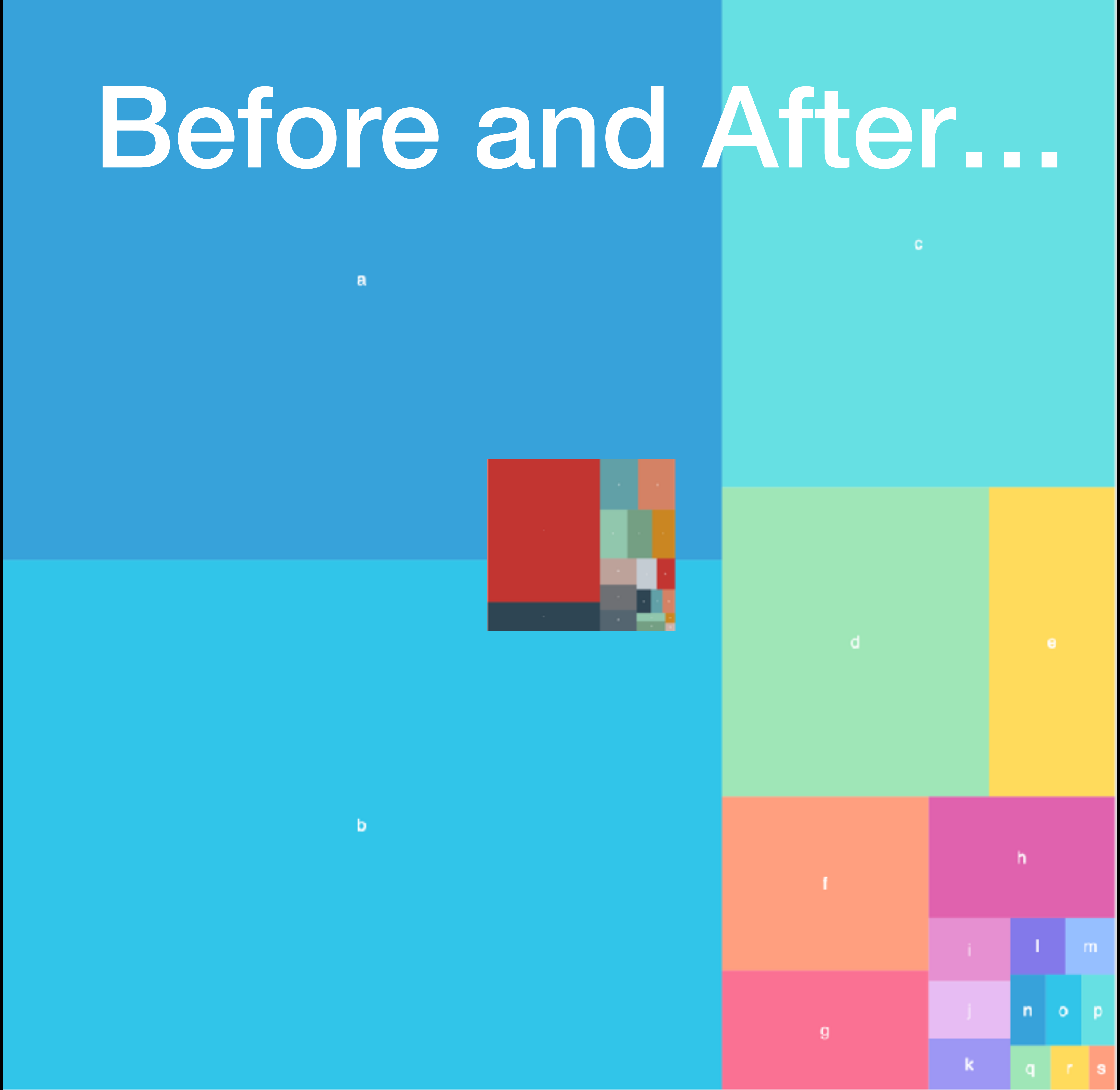
- Apply the settings to one machine
- Don’t apply the settings to its brother with the same traffic

Disk Churn dropped …

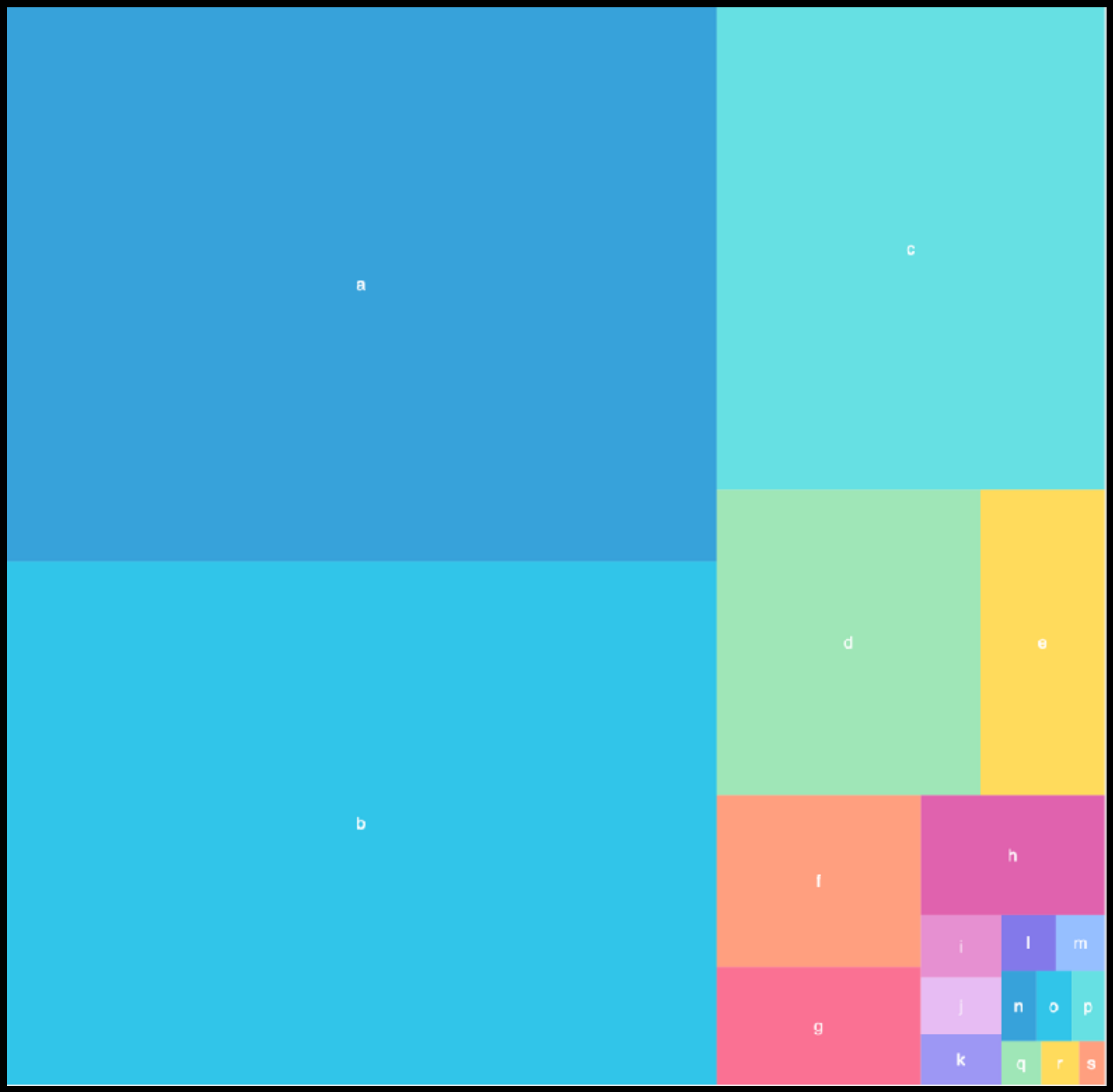


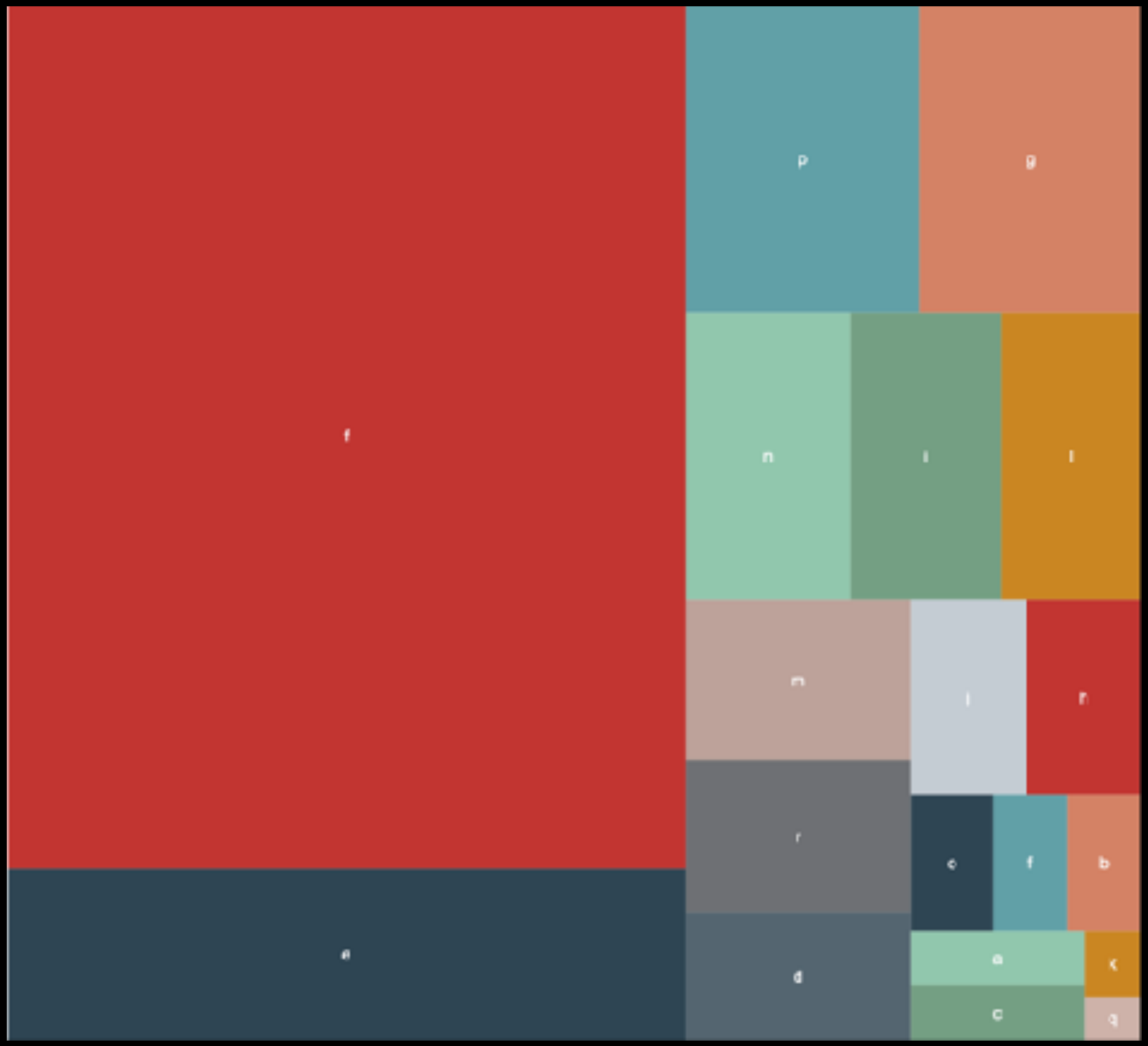
Bandwidth Cache Hit Rate increased

Before and After…

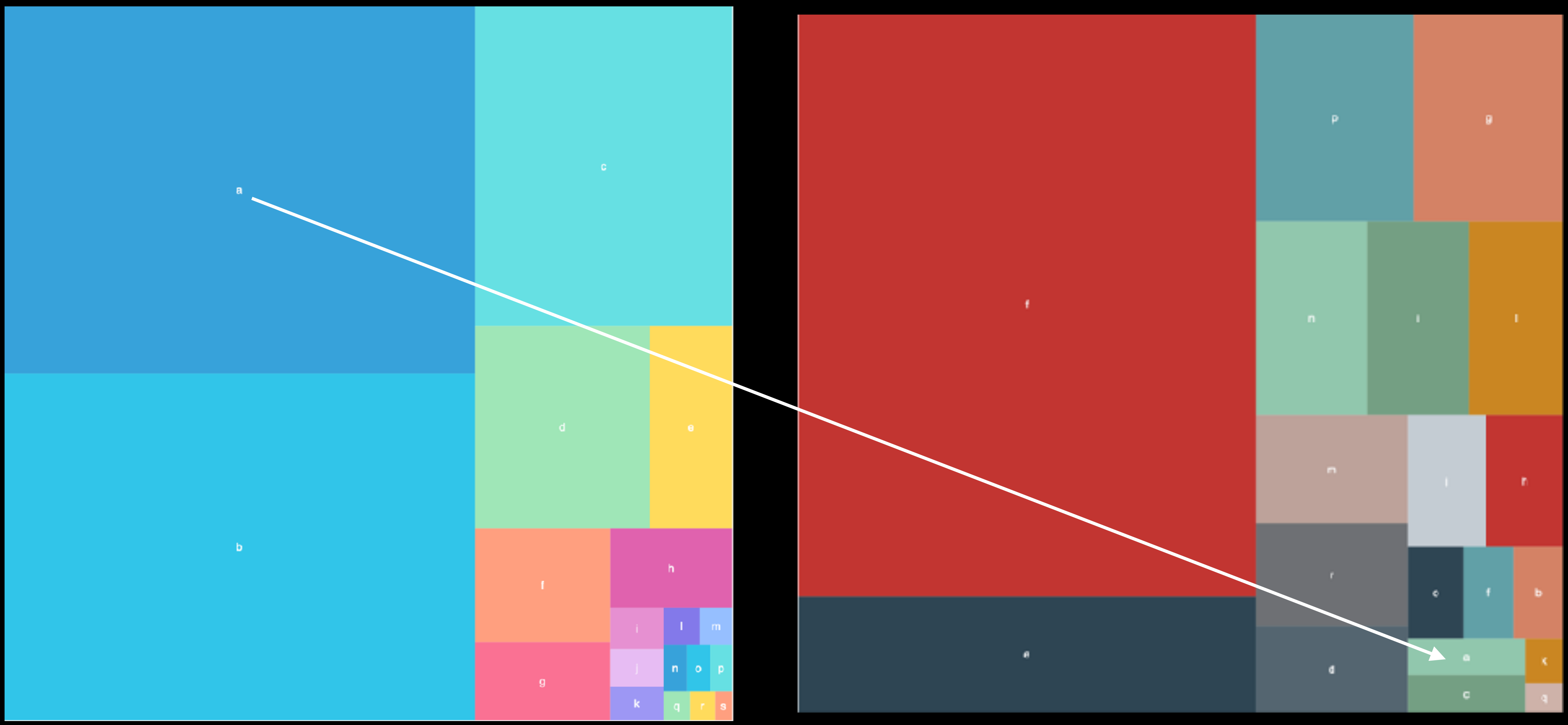


Before and After...

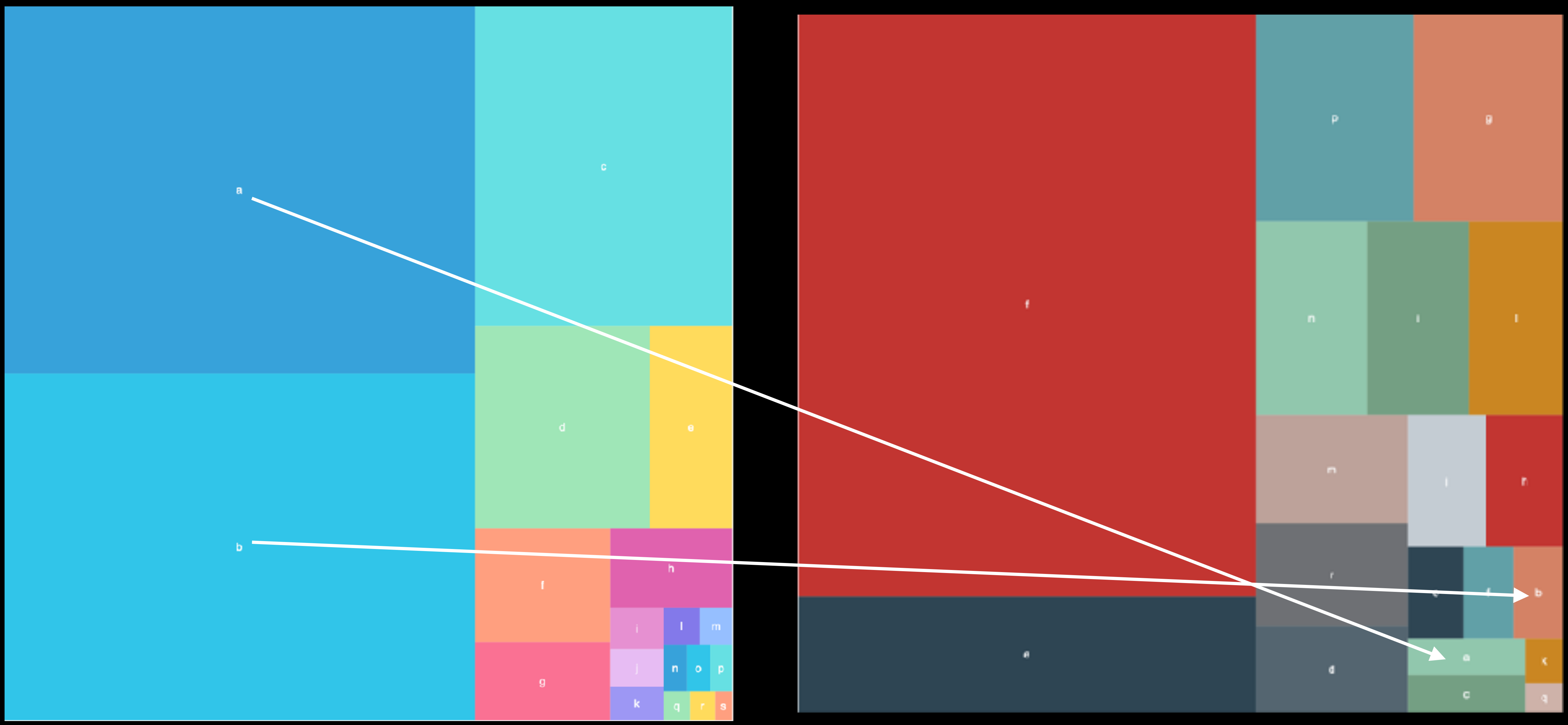




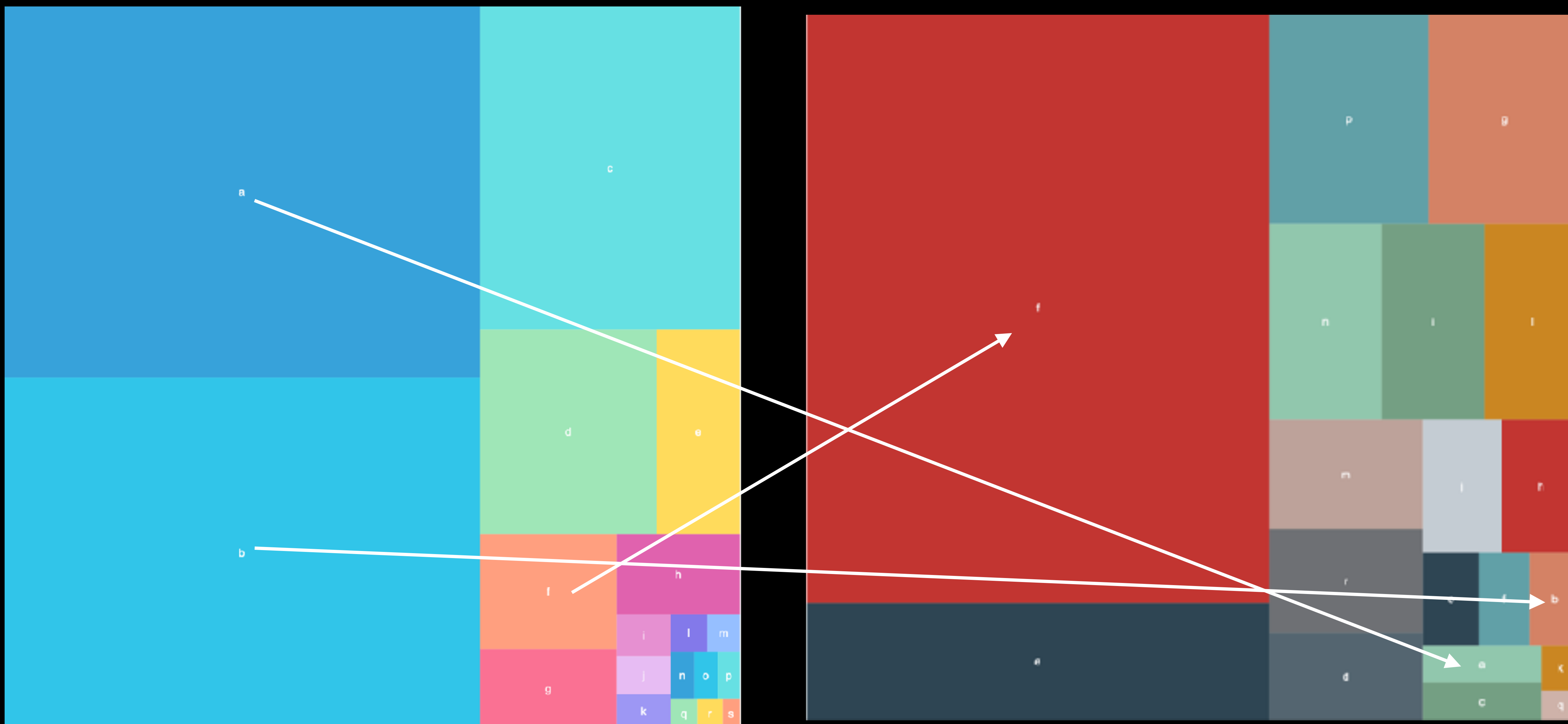
Before and After…



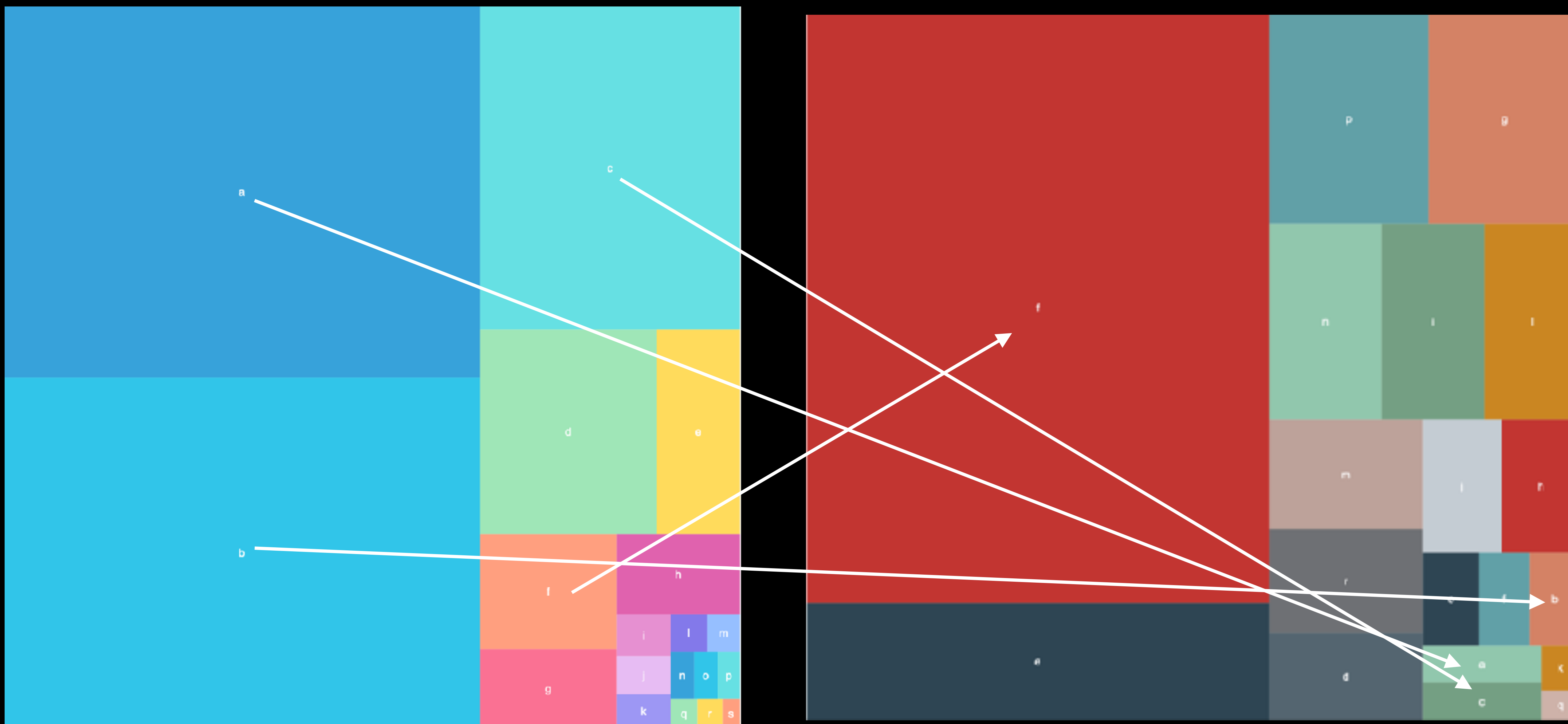
Before and After...



Before and After…



Before and After…



Future Work

- Bucket Size
- How to define a “Big” property
- Would a different eviction algorithm for the tracking table be better?