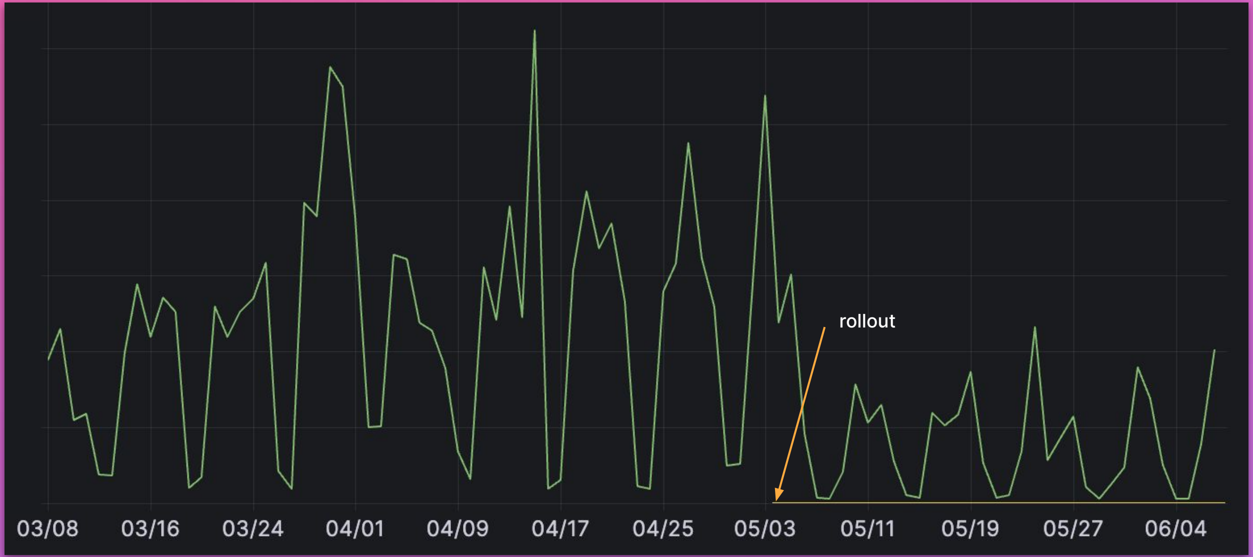
Grafana Labs

cache me if you can

how Grafana Labs scaled up their memcached **42x** and **improved reliability**

danny kopping / sr. software engineer

bottom line upfront



- ✅ **65% reduction** in object storage reqs
- ✅ vastly **improved reliability**
- ✅ **~2% overall TCO reduction**
- 🤨 **no change** in performance

but first...

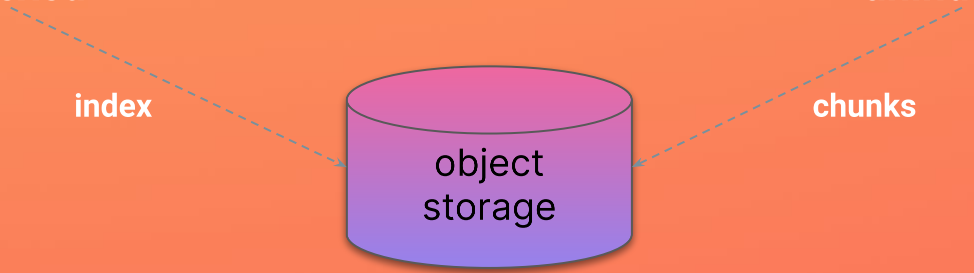
loki internals

how does Loki work?

`2023-06-16T06:09:05.123456789Z`	`{app="nginx", env="dev"}`	`192.168.1.1 [06/16/2023:06:09:05 +0000] "GET / HTTP/1.1" 200`
timestamp	**labels/selectors**	**content**
with nanosecond precision	key-value pairs	log line

indexed (timestamp, labels/selectors)

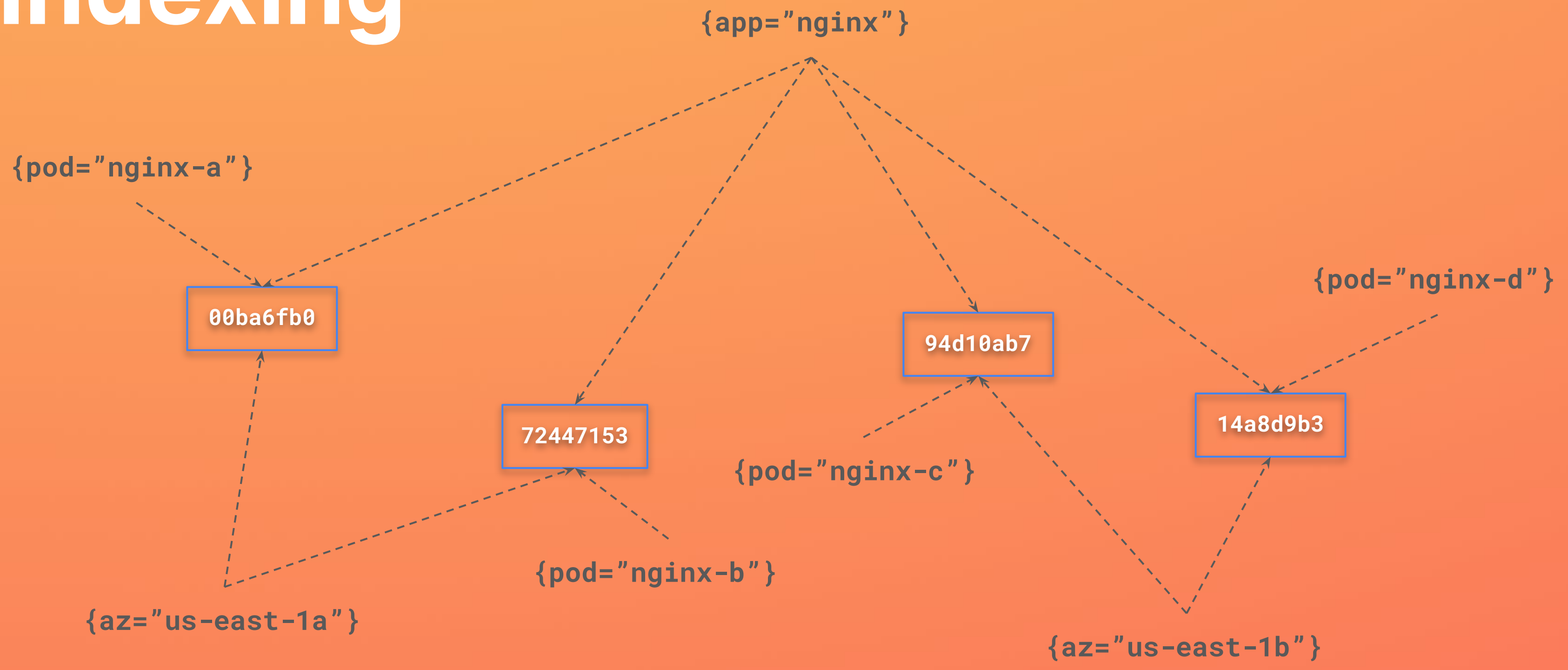
unindexed (content)



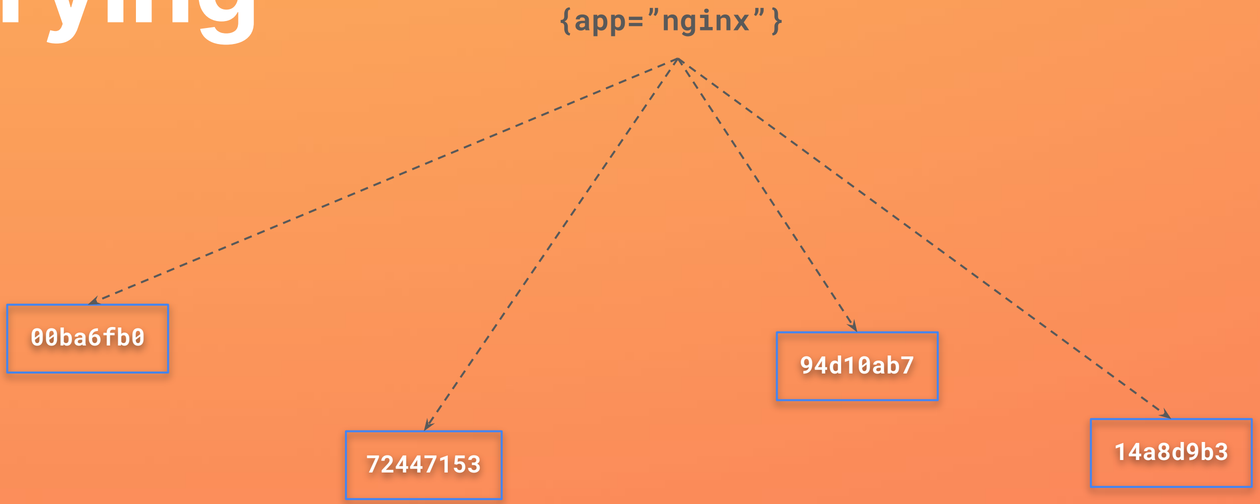
ingestion


logs
ingester
chunks
2023-06-16 11:02.123456789
2023-06-16 11:12.123456789
{app="nginx", pod="nginx-a", az="us-east-1a"}
192.168.1.1 - - [06/16/2023:06:09:05 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.74.0"
192.168.1.2 - - [06/16/2023:06:09:06 +0000] "GET /index.html HTTP/1.1" 200 612 "-" "wget/1.21.1"
192.168.1.3 - - [06/16/2023:06:09:07 +0000] "GET /about.html HTTP/1.1" 200 396 "-" "Safari/15.4"
192.168.1.4 - - [06/16/2023:06:09:08 +0000] "GET /contact.html HTTP/1.1" 200 299 "-" "Firefox/92.0"
192.168.1.5 - - [06/16/2023:06:09:09 +0000] "GET /blog/index.html HTTP/1.1" 200 824 "-" "Chrome/92.0.4515.131"
192.168.1.6 - - [06/16/2023:06:09:10 +0000] "GET /blog/post-1.html HTTP/1.1" 200 543 "-" "Opera/12.15"
192.168.1.7 - - [06/16/2023:06:09:11 +0000] "GET /blog/post-2.html HTTP/1.1" 200 456 "-" "Edge/102.0.1245.153"
192.168.1.8 - - [06/16/2023:06:09:12 +0000] "GET /blog/post-3.html HTTP/1.1" 200 369 "-" "IE/11.0"
2023-06-16 11:02.123456789
2023-06-16 11:12.123456789
{app="nginx", pod="nginx-b", az="us-east-1a"}
192.168.1.1 - - [06/16/2023:06:09:13 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.74.0"
192.168.1.2 - - [06/16/2023:06:09:14 +0000] "GET /index.html HTTP/1.1" 200 612 "-" "wget/1.21.1"
192.168.1.3 - - [06/16/2023:06:09:15 +0000] "GET /about.html HTTP/1.1" 200 396 "-" "Safari/15.4"
192.168.1.4 - - [06/16/2023:06:09:16 +0000] "GET /contact.html HTTP/1.1" 200 299 "-" "Firefox/92.0"
192.168.1.5 - - [06/16/2023:06:09:17 +0000] "GET /blog/index.html HTTP/1.1" 200 824 "-" "Chrome/92.0.4515.131"
192.168.1.6 - - [06/16/2023:06:09:18 +0000] "GET /blog/post-1.html HTTP/1.1" 200 543 "-" "Opera/12.15"
192.168.1.7 - - [06/16/2023:06:09:19 +0000] "GET /blog/post-2.html HTTP/1.1" 200 456 "-" "Edge/102.0.1245.153"
192.168.1.8 - - [06/16/2023:06:09:20 +0000] "GET /blog/post-3.html HTTP/1.1" 200 369 "-" "IE/11.0"
2023-06-16 11:02.123456789
2023-06-16 11:12.123456789
{app="nginx", pod="nginx-c", az="us-east-1b"}
192.168.1.1 - - [06/16/2023:06:09:05 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.74.0"
192.168.1.2 - - [06/16/2023:06:09:06 +0000] "GET /index.html HTTP/1.1" 200 612 "-" "wget/1.21.1"
192.168.1.3 - - [06/16/2023:06:09:07 +0000] "GET /about.html HTTP/1.1" 200 396 "-" "Safari/15.4"
192.168.1.4 - - [06/16/2023:06:09:08 +0000] "GET /contact.html HTTP/1.1" 200 299 "-" "Firefox/92.0"
192.168.1.5 - - [06/16/2023:06:09:09 +0000] "GET /blog/index.html HTTP/1.1" 200 824 "-" "Chrome/92.0.4515.131"
192.168.1.6 - - [06/16/2023:06:09:10 +0000] "GET /blog/post-1.html HTTP/1.1" 200 543 "-" "Opera/12.15"
192.168.1.7 - - [06/16/2023:06:09:11 +0000] "GET /blog/post-2.html HTTP/1.1" 200 456 "-" "Edge/102.0.1245.153"
192.168.1.8 - - [06/16/2023:06:09:12 +0000] "GET /blog/post-3.html HTTP/1.1" 200 369 "-" "IE/11.0"
2023-06-16 11:02.123456789
2023-06-16 11:12.123456789
{app="nginx", pod="nginx-d", az="us-east-1b"}
192.168.1.1 - - [06/16/2023:06:09:05 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.74.0"
192.168.1.2 - - [06/16/2023:06:09:06 +0000] "GET /index.html HTTP/1.1" 200 612 "-" "wget/1.21.1"
192.168.1.3 - - [06/16/2023:06:09:07 +0000] "GET /about.html HTTP/1.1" 200 396 "-" "Safari/15.4"
192.168.1.4 - - [06/16/2023:06:09:08 +0000] "GET /contact.html HTTP/1.1" 200 299 "-" "Firefox/92.0"
192.168.1.5 - - [06/16/2023:06:09:09 +0000] "GET /blog/index.html HTTP/1.1" 200 824 "-" "Chrome/92.0.4515.131"
192.168.1.6 - - [06/16/2023:06:09:10 +0000] "GET /blog/post-1.html HTTP/1.1" 200 543 "-" "Opera/12.15"
192.168.1.7 - - [06/16/2023:06:09:11 +0000] "GET /blog/post-2.html HTTP/1.1" 200 456 "-" "Edge/102.0.1245.153"
192.168.1.8 - - [06/16/2023:06:09:12 +0000] "GET /blog/post-3.html HTTP/1.1" 200 369 "-" "IE/11.0"
00ba6fb0
72447153
94d10ab7
14a8d9b3
object storage

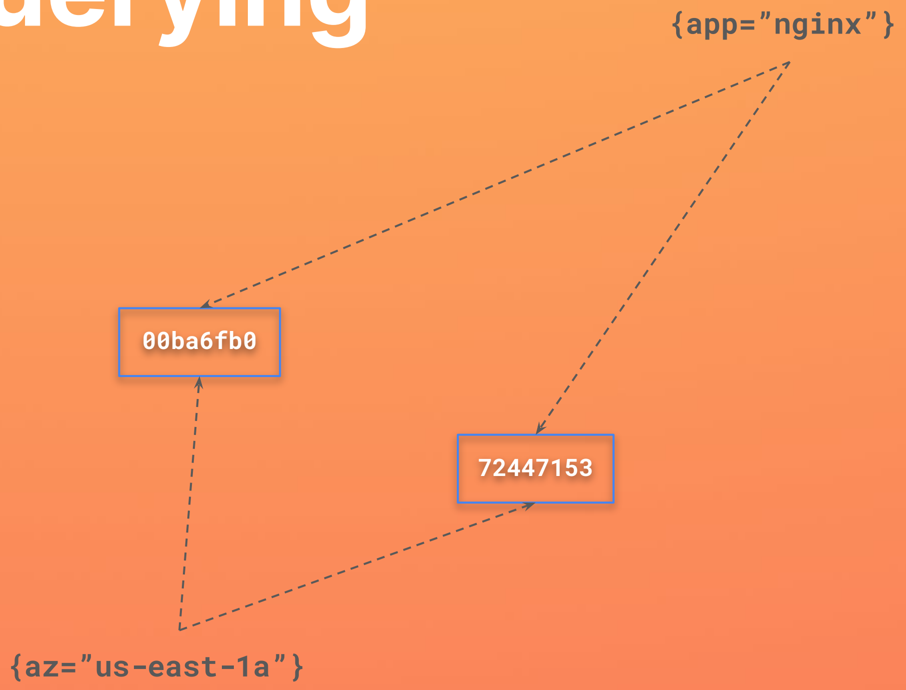
indexing


{app="nginx"}
{pod="nginx-a"}
00ba6fb0
72447153
94d10ab7
14a8d9b3
{pod="nginx-d"}
{pod="nginx-c"}
{pod="nginx-b"}
{az="us-east-1a"}
{az="us-east-1b"}

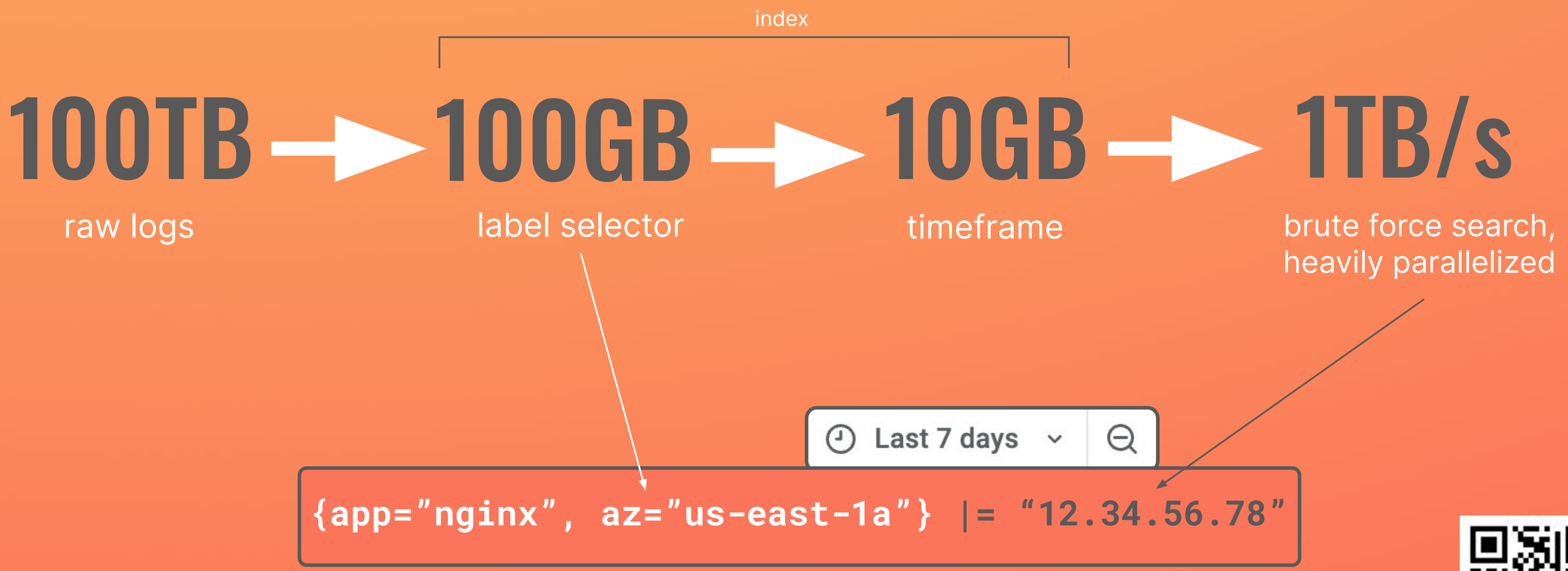
querying



querying



fast queries



slow queries

100TB → 100GB → 10GB → 1TB/s

raw logs — label selector — timeframe — brute force search, heavily parallelized

Last 30 days UTC

```
{app=~".+"} |= "12.34.56.78"
```

Grafana Cloud

+

rate-limiting

rate-limiting

⇒ decreased throughput
⇒ slower queries
⇒ frustrated users
⇒ SLO budget burn alerts
⇒ operator toil
⇒ "the incident"

two choices:

query less or **cache more**

choice A:

query less

improve labels
accelerated "needle-in-haystack"
smarter query engine
chunk compaction

the problem:

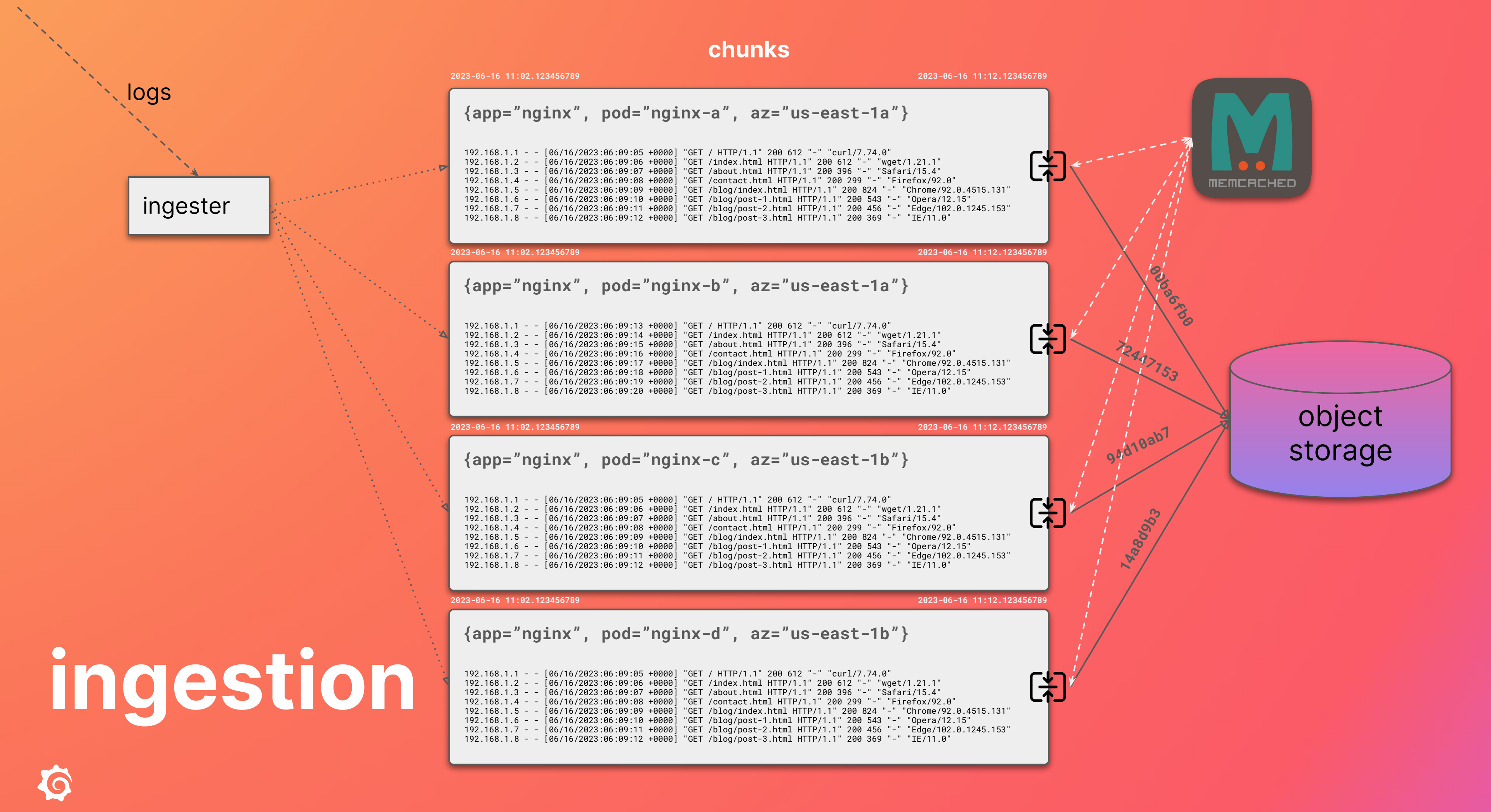
this shit takes time

choice B:

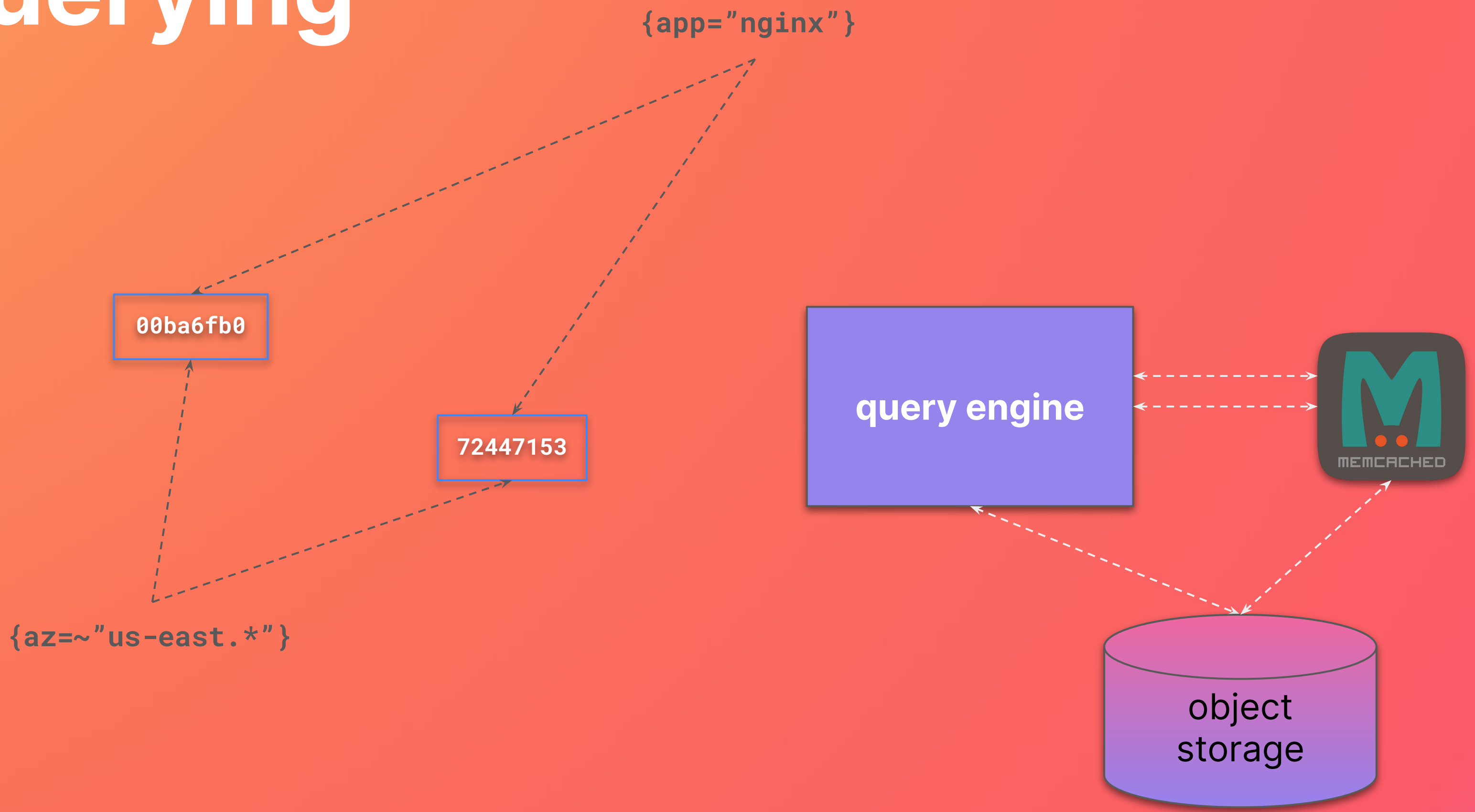
cache more

buy time to do the *right thing*™
easier lever to pull
shorter delivery timeline

caching


chunks
logs
ingester
2023-06-16 11:02.123456789
2023-06-16 11:12.123456789
{app="nginx", pod="nginx-a", az="us-east-1a"}
192.168.1.1 - - [06/16/2023:06:09:05 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.74.0"
192.168.1.2 - - [06/16/2023:06:09:06 +0000] "GET /index.html HTTP/1.1" 200 612 "-" "wget/1.21.1"
192.168.1.3 - - [06/16/2023:06:09:07 +0000] "GET /about.html HTTP/1.1" 200 396 "-" "Safari/15.4"
192.168.1.4 - - [06/16/2023:06:09:08 +0000] "GET /contact.html HTTP/1.1" 200 299 "-" "Firefox/92.0"
192.168.1.5 - - [06/16/2023:06:09:09 +0000] "GET /blog/index.html HTTP/1.1" 200 824 "-" "Chrome/92.0.4515.131"
192.168.1.6 - - [06/16/2023:06:09:10 +0000] "GET /blog/post-1.html HTTP/1.1" 200 543 "-" "Opera/12.15"
192.168.1.7 - - [06/16/2023:06:09:11 +0000] "GET /blog/post-2.html HTTP/1.1" 200 456 "-" "Edge/102.0.1245.153"
192.168.1.8 - - [06/16/2023:06:09:12 +0000] "GET /blog/post-3.html HTTP/1.1" 200 369 "-" "IE/11.0"
2023-06-16 11:02.123456789
2023-06-16 11:12.123456789
{app="nginx", pod="nginx-b", az="us-east-1a"}
192.168.1.1 - - [06/16/2023:06:09:13 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.74.0"
192.168.1.2 - - [06/16/2023:06:09:14 +0000] "GET /index.html HTTP/1.1" 200 612 "-" "wget/1.21.1"
192.168.1.3 - - [06/16/2023:06:09:15 +0000] "GET /about.html HTTP/1.1" 200 396 "-" "Safari/15.4"
192.168.1.4 - - [06/16/2023:06:09:16 +0000] "GET /contact.html HTTP/1.1" 200 299 "-" "Firefox/92.0"
192.168.1.5 - - [06/16/2023:06:09:17 +0000] "GET /blog/index.html HTTP/1.1" 200 824 "-" "Chrome/92.0.4515.131"
192.168.1.6 - - [06/16/2023:06:09:18 +0000] "GET /blog/post-1.html HTTP/1.1" 200 543 "-" "Opera/12.15"
192.168.1.7 - - [06/16/2023:06:09:19 +0000] "GET /blog/post-2.html HTTP/1.1" 200 456 "-" "Edge/102.0.1245.153"
192.168.1.8 - - [06/16/2023:06:09:20 +0000] "GET /blog/post-3.html HTTP/1.1" 200 369 "-" "IE/11.0"
2023-06-16 11:02.123456789
2023-06-16 11:12.123456789
{app="nginx", pod="nginx-c", az="us-east-1b"}
192.168.1.1 - - [06/16/2023:06:09:05 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.74.0"
192.168.1.2 - - [06/16/2023:06:09:06 +0000] "GET /index.html HTTP/1.1" 200 612 "-" "wget/1.21.1"
192.168.1.3 - - [06/16/2023:06:09:07 +0000] "GET /about.html HTTP/1.1" 200 396 "-" "Safari/15.4"
192.168.1.4 - - [06/16/2023:06:09:08 +0000] "GET /contact.html HTTP/1.1" 200 299 "-" "Firefox/92.0"
192.168.1.5 - - [06/16/2023:06:09:09 +0000] "GET /blog/index.html HTTP/1.1" 200 824 "-" "Chrome/92.0.4515.131"
192.168.1.6 - - [06/16/2023:06:09:10 +0000] "GET /blog/post-1.html HTTP/1.1" 200 543 "-" "Opera/12.15"
192.168.1.7 - - [06/16/2023:06:09:11 +0000] "GET /blog/post-2.html HTTP/1.1" 200 456 "-" "Edge/102.0.1245.153"
192.168.1.8 - - [06/16/2023:06:09:12 +0000] "GET /blog/post-3.html HTTP/1.1" 200 369 "-" "IE/11.0"
2023-06-16 11:02.123456789
2023-06-16 11:12.123456789
{app="nginx", pod="nginx-d", az="us-east-1b"}
192.168.1.1 - - [06/16/2023:06:09:05 +0000] "GET / HTTP/1.1" 200 612 "-" "curl/7.74.0"
192.168.1.2 - - [06/16/2023:06:09:06 +0000] "GET /index.html HTTP/1.1" 200 612 "-" "wget/1.21.1"
192.168.1.3 - - [06/16/2023:06:09:07 +0000] "GET /about.html HTTP/1.1" 200 396 "-" "Safari/15.4"
192.168.1.4 - - [06/16/2023:06:09:08 +0000] "GET /contact.html HTTP/1.1" 200 299 "-" "Firefox/92.0"
192.168.1.5 - - [06/16/2023:06:09:09 +0000] "GET /blog/index.html HTTP/1.1" 200 824 "-" "Chrome/92.0.4515.131"
192.168.1.6 - - [06/16/2023:06:09:10 +0000] "GET /blog/post-1.html HTTP/1.1" 200 543 "-" "Opera/12.15"
192.168.1.7 - - [06/16/2023:06:09:11 +0000] "GET /blog/post-2.html HTTP/1.1" 200 456 "-" "Edge/102.0.1245.153"
192.168.1.8 - - [06/16/2023:06:09:12 +0000] "GET /blog/post-3.html HTTP/1.1" 200 369 "-" "IE/11.0"
MEMCACHED
00ba6fb0
72417153
94d10ab7
14a8d9b3
object storage
ingestion

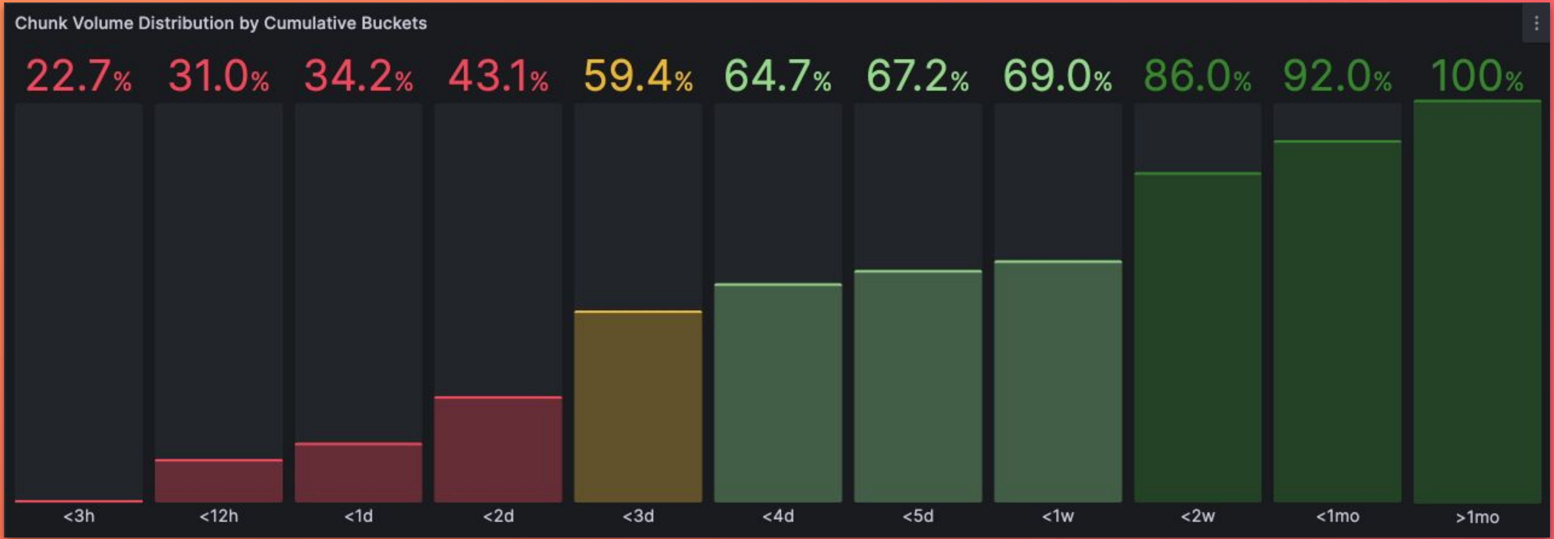
querying


{app="nginx"}
00ba6fb0
72447153
{az=~"us-east.*"}
query engine
MEMCACHED
object
storage

recency bias

global

recency bias



loki-prod3

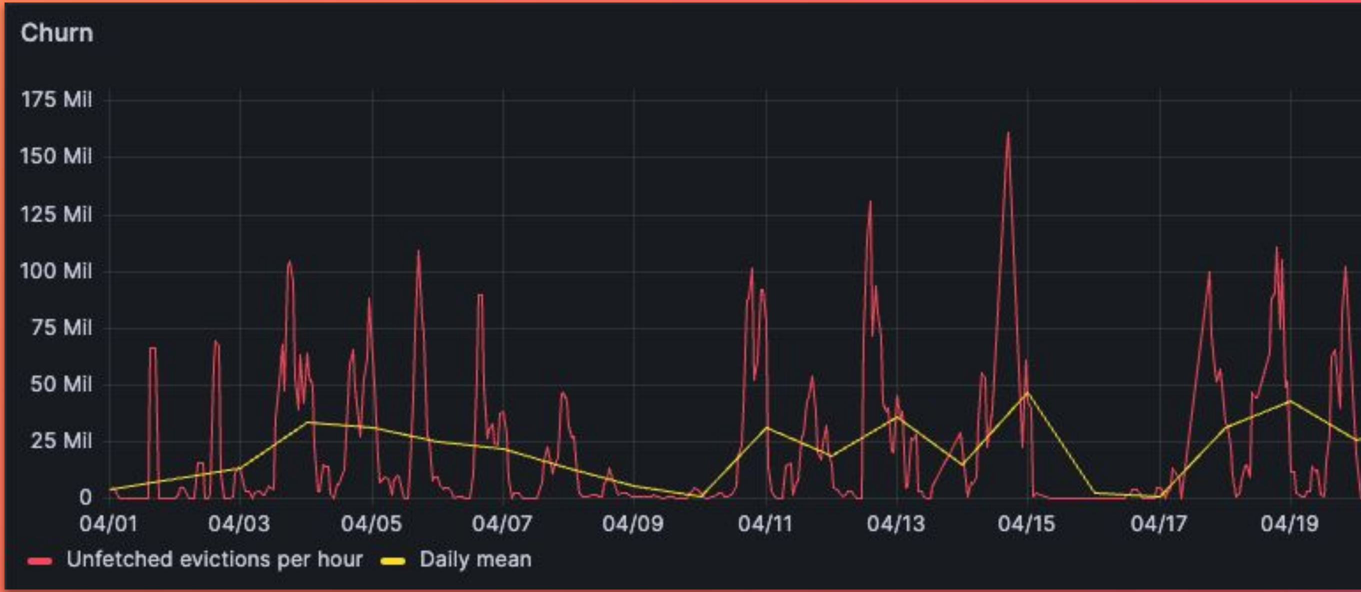
success metric:

hit rate

looks kinda... *good*, right?

problem:

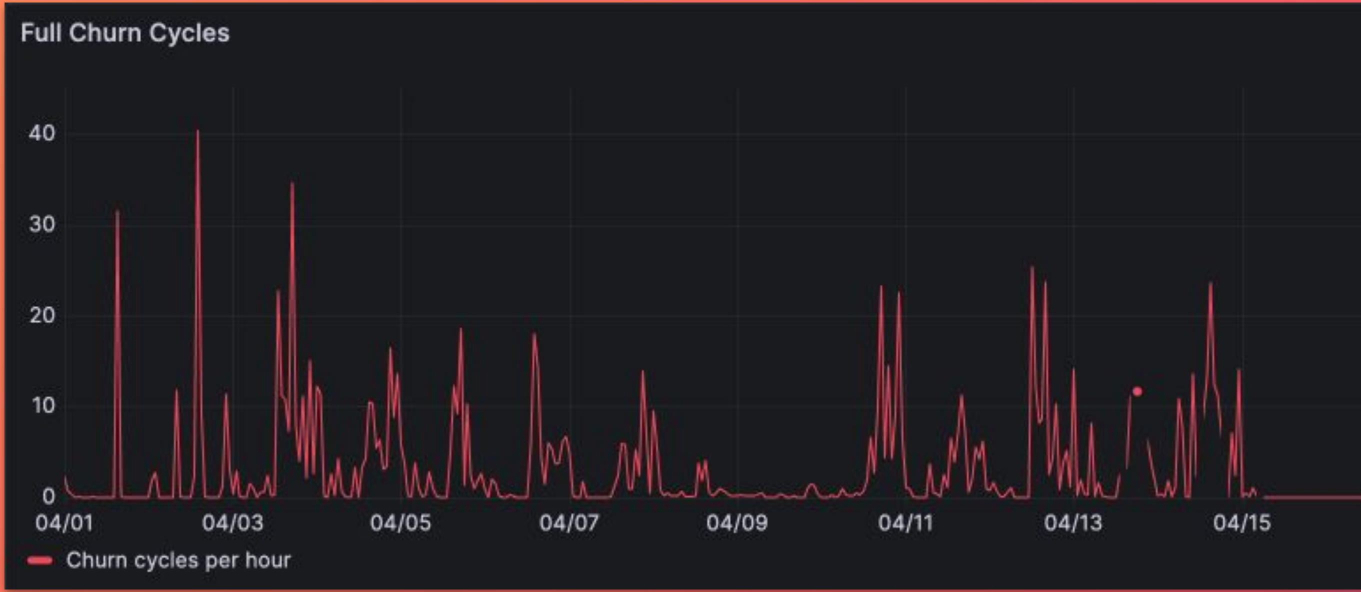
churn



items being evicted before being fetched **even once!**

problem:

churn



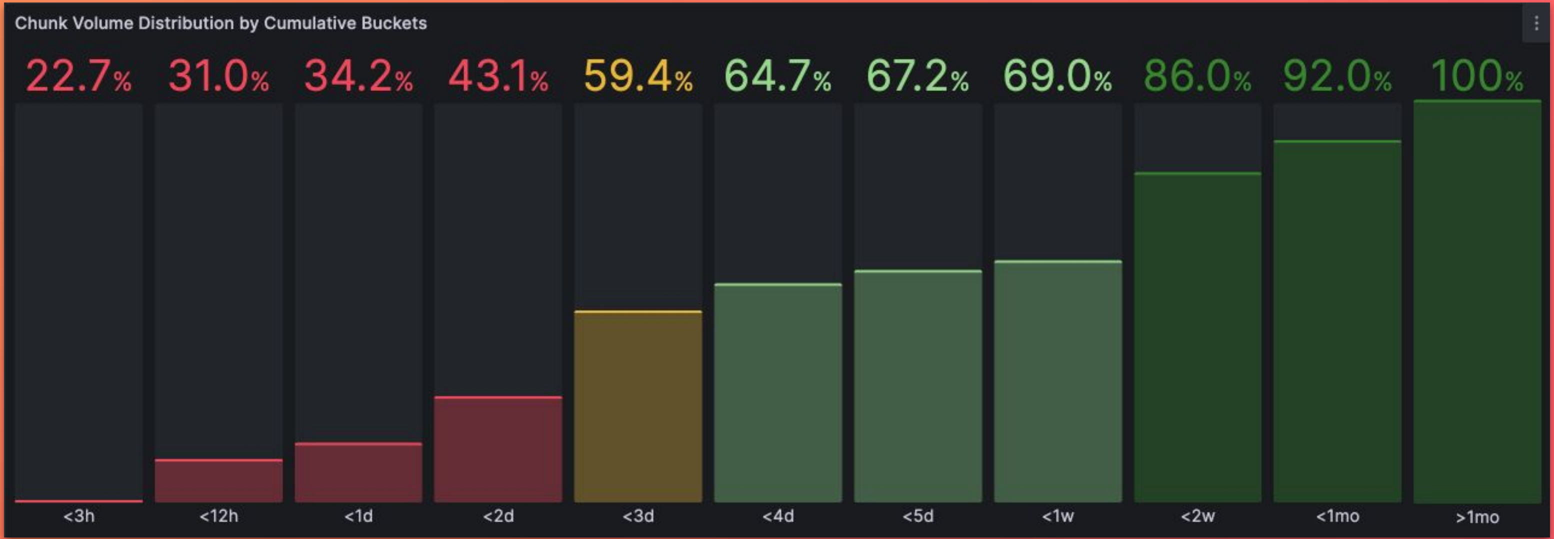
full cache replacements per hour

how much cache do we need?

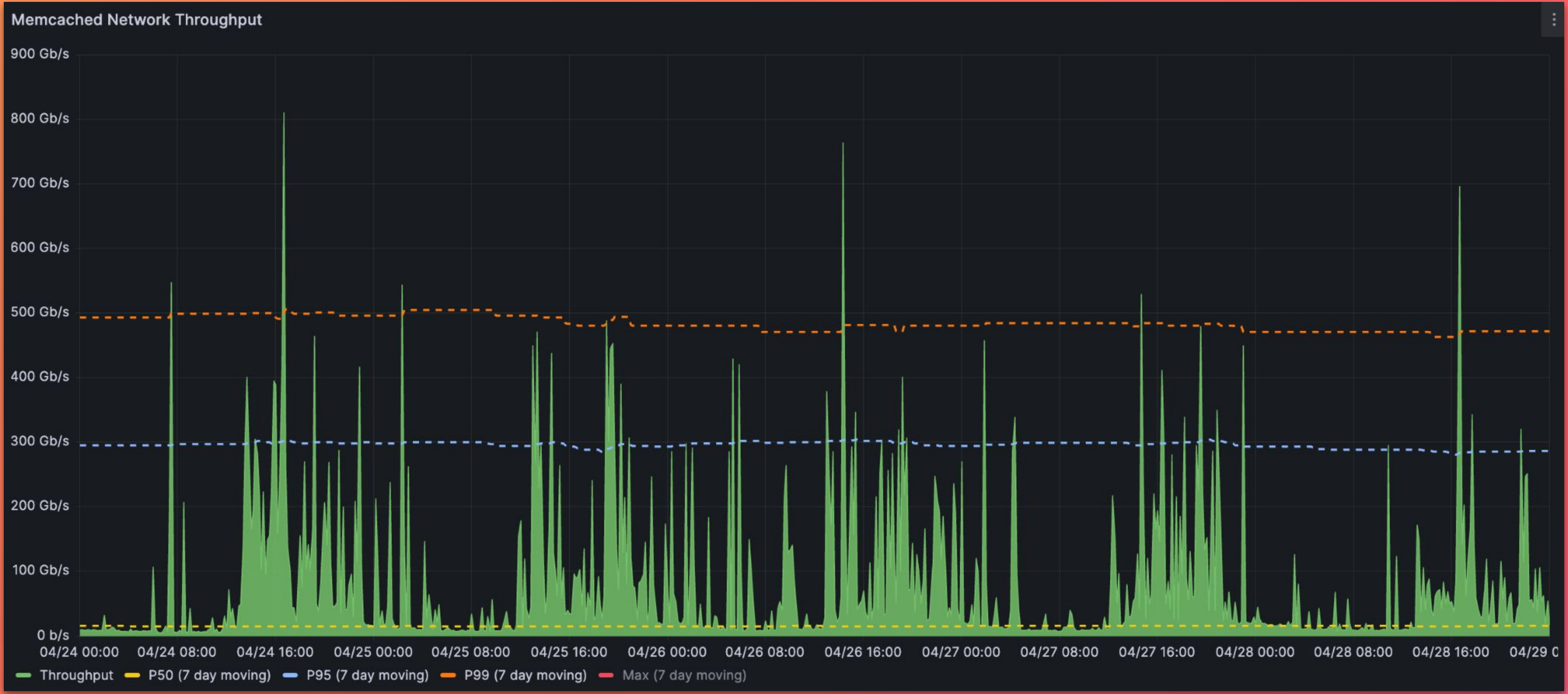
200 instances @ **1 vCPU, 6GB RAM**

total capacity: ~1.2TB RAM
running on shared
`n2-standard-32` nodes

how much cache do we need?



how much cache do we need?



~50TB
cache

~500Gbps
throughput

the challenge:

can we do it **cost-effectively,**

whilst **maintaining performance,**

in an **operationally-familiar way?**

200 instances @ **1 vCPU, 6GB RAM**

total capacity: ~1.2TB RAM
running on shared
n2-standard-32 nodes

```
CPU: <list> / 2 / <cpus> = ~$17.72/vCPU/month
RAM: <list> / 2 / <gb>   = ~$4.43/GB/month
```

200 vCPUs ⇒ $3544
1200 GB RAM ⇒ $5316
⇒ **$8860 per month**

200 replicas @ **1 vCPU, 6GB RAM**

Total capacity: ~1.2TB RAM
running on shared
n2-standard-32 nodes

CPU: <list> / 2 / <cpus> = ~$17.72/vCPU/month

RAM: ~$4.43/GB/month

200 vCPUs ⇒ $3544
1200 GB RAM ⇒ $5316
⇒ **$8860 per month**

the goal:

drive down cost per GB of cache

use memcached …with SSDs!

memcached "extstore"

the industry's best-kept secret

Home About Downloads Blog Mailing List Wiki Bugs

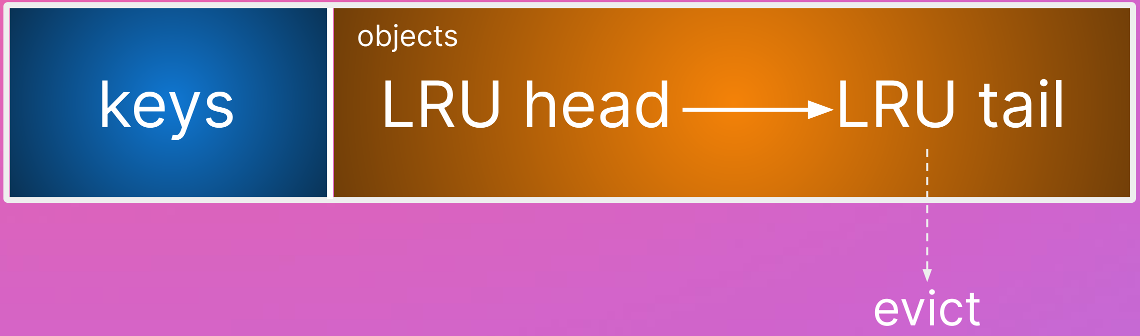
Caching beyond RAM: the case for NVMe - Dormando (June 12th, 2018)

Caching architectures at every layer of the stack embody an implicit tradeoff between performance and cost. These tradeoffs however are constantly shifting: new inflection points can emerge alongside advances in storage technology, changes in workload patterns, or fluctuations in hardware supply and demand.

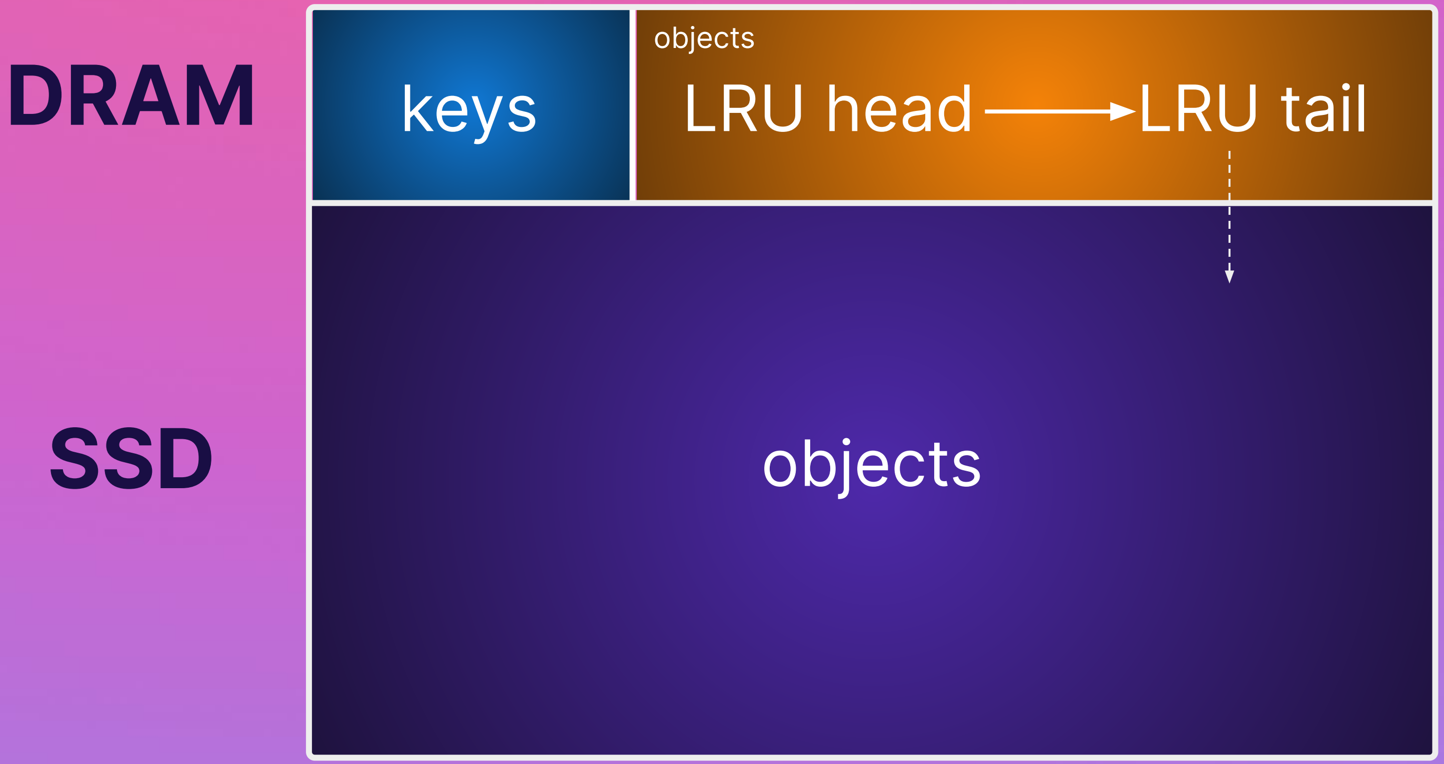
In this post we explore the design ramifications of the increasing cost of RAM on caching systems. While RAM has always been expensive, DRAM prices have risen by over 50% in 2017, and high densities of RAM involve multi-socket NUMA machines, bloating power and overall costs. Concurrently, alternative storage technologies such as Flash and Optane continue to improve. They have specialized hardware interfaces, consistent performance, high density, and relatively low costs. While there is increasing economic incentive to explore offloading caching from RAM onto NVMe or NVM devices, the implications for performance are still not widely understood.

We will explore these design implications in the context of Memcached, a distributed, simple, cache-focused key/value store. For a quick overview, see the about page or the story tutorial.

DRAM



extstore



using extstore

```
memcached <other flags>
   --extended
   ext_path=/mnt/disks/ssd0/datafile:345G
```

using extstore

```
memcached <other flags>
  --extended
  ext_path=/mnt/disks/ssd0/datafile:345G,
           /mnt/disks/ssd1/datafile:345G,
           /mnt/disks/ssd2/datafile:345G...
```

SSDs in cloud?

no "disk over network"
no multi-tenancy
physically attached to hypervisor

Google Cloud

"Local SSDs"

- **375GB** each @ **$30**/month
- add up to **24 SSDs** to most machine types
- **660MB/s reads, 350MB/s writes**

“Instance Storage”

- varies by instance type
- included in cost

~~ttl~~ ttv: time-to-value

2 weeks!

33 instances @ **6 vCPU, 5GB RAM, 4 SSDs**

total capacity: ~50TB SSD, 528Gbps
running on *dedicated*
n2-highcpu-8 nodes (16Gbps each)

```
CPU: <list> / 2 / <cpus>   = ~$13.08/vCPU/month
RAM: <list> / 2 / <gb>     = ~$13.08/GB/month
SSD: $30 per disk (375GB) = $0.08/GB/month
```

198 vCPUs ⇒ $2590
165 GB RAM ⇒ $2158
132 SSDs ⇒ $3960

⇒ **$8708 per month**

33 instances @ 6 vCPU, 5GB RAM,

Total capacity: ~50TB SSD,
running on dedicated
n2-highcpu-8 nodes

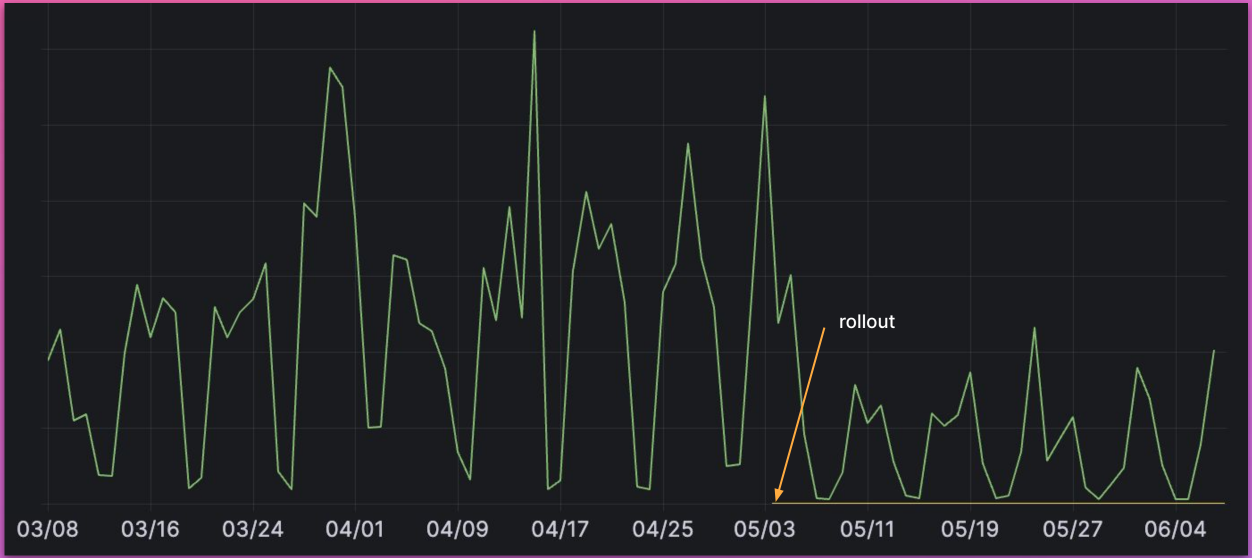
SSD: = $0.08/GB/month

200 instances @ **1 vCPU, 6GB RAM**

total capacity: ~1.2TB RAM
running on shared
`n2-standard-32` nodes

```
CPU: <list> / 2 / <cpus> = ~$17.72/vCPU/month
RAM: <list> / 2 / <gb>   = ~$4.43/GB/month
```

200 vCPUs ⇒ $3544
1200 GB RAM ⇒ $5316
⇒ **$8860 per month**



- ✅ **65% reduction** in object storage reqs
- ✅ vastly **improved reliability**
- ✅ **~2% overall TCO reduction**
- 🤨 **no change** in performance

Google Cloud Storage

Amazon S3

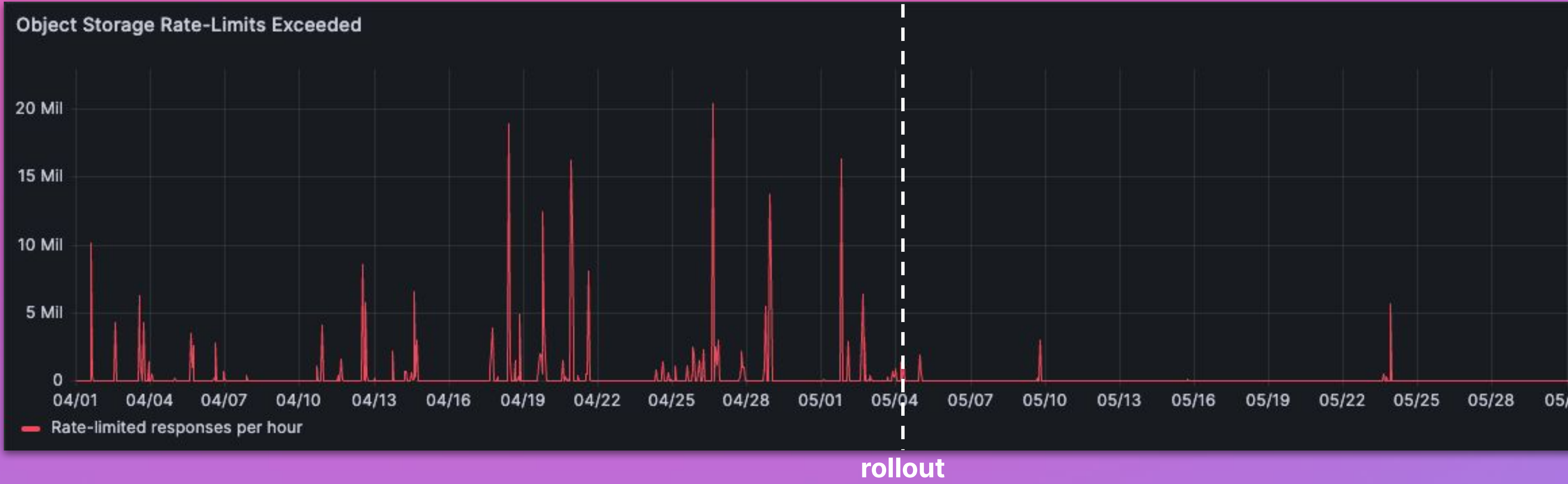
object storage

$5 per million writes
$0.005 per 1000

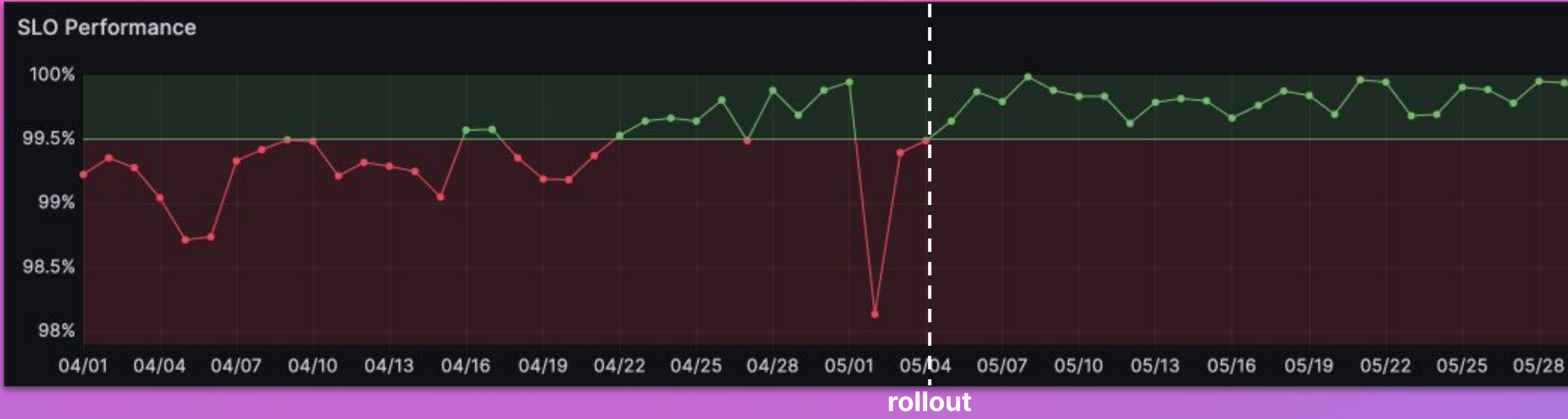
$0.4 per million reads
$0.0004 per 1000

no charge for bandwidth within the same region

rate-limits


Object Storage Rate-Limits Exceeded
20 Mil
15 Mil
10 Mil
5 Mil
0
04/01
04/04
04/07
04/10
04/13
04/16
04/19
04/22
04/25
04/28
05/01
05/04
05/07
05/10
05/13
05/16
05/19
05/22
05/25
05/28
Rate-limited responses per hour
rollout

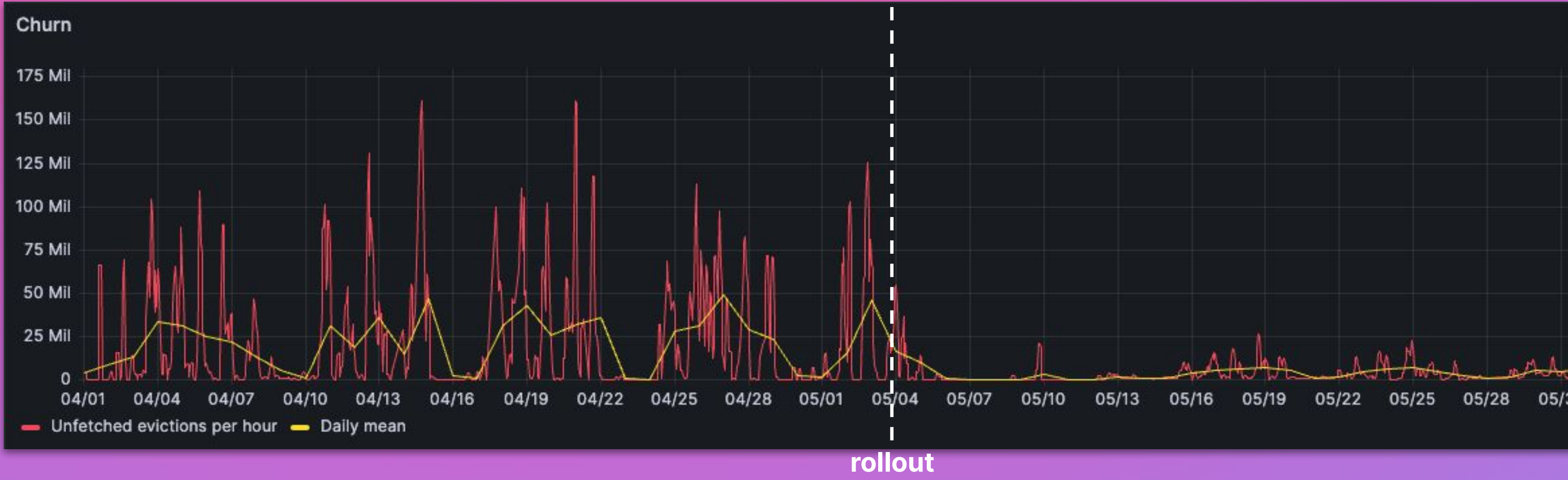
SLO (query throughput)



hit rate


Cache Hit Rate
100.0%
90.0%
80.0%
70.0%
60.0%
50.0%
04/01
04/04
04/07
04/10
04/13
04/16
04/19
04/22
04/25
04/28
05/01
05/04
05/07
05/10
05/13
05/16
05/19
05/22
05/25
05/28
05/31
Hit rate
Daily mean
rollout

churn rate


Churn
175 Mil
150 Mil
125 Mil
100 Mil
75 Mil
50 Mil
25 Mil
0
04/01
04/04
04/07
04/10
04/13
04/16
04/19
04/22
04/25
04/28
05/01
05/04
05/07
05/10
05/13
05/16
05/19
05/22
05/25
05/28
Unfetched evictions per hour
Daily mean
rollout

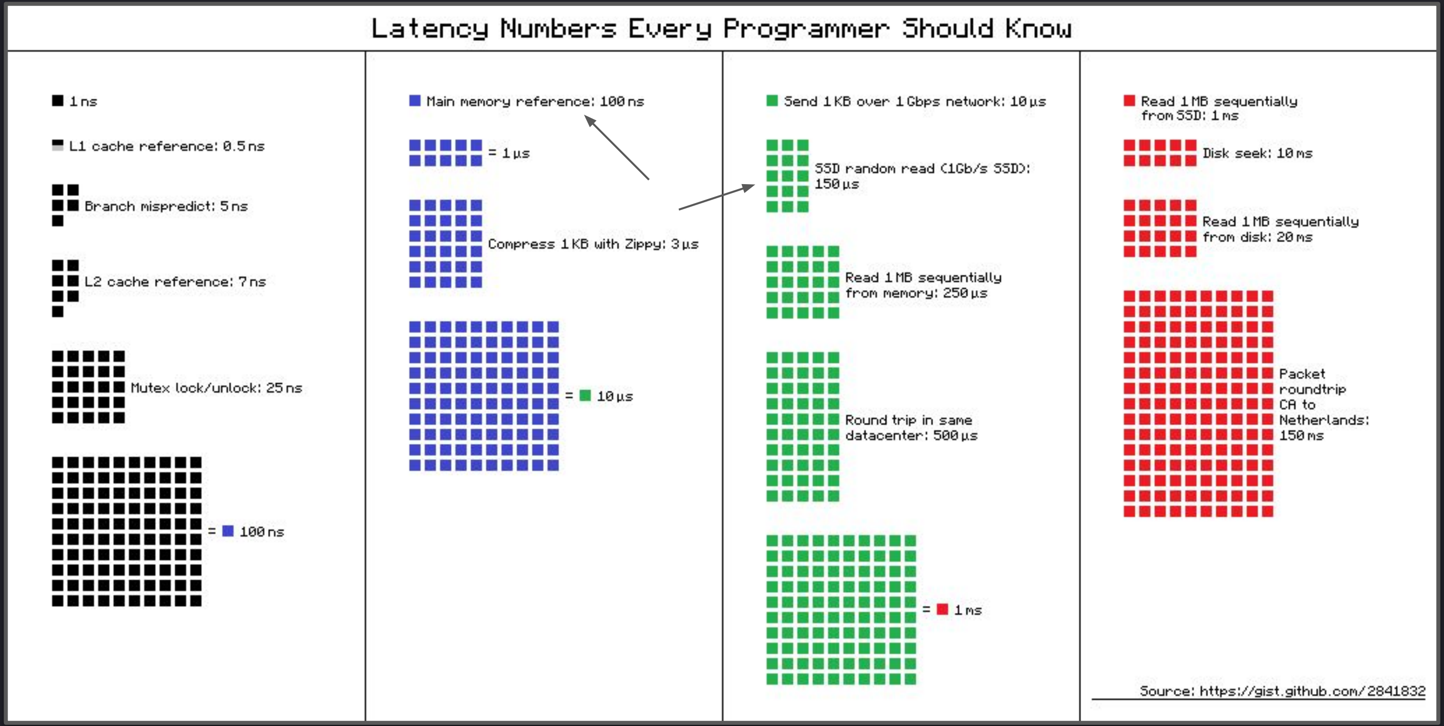
cache effectiveness

query performance

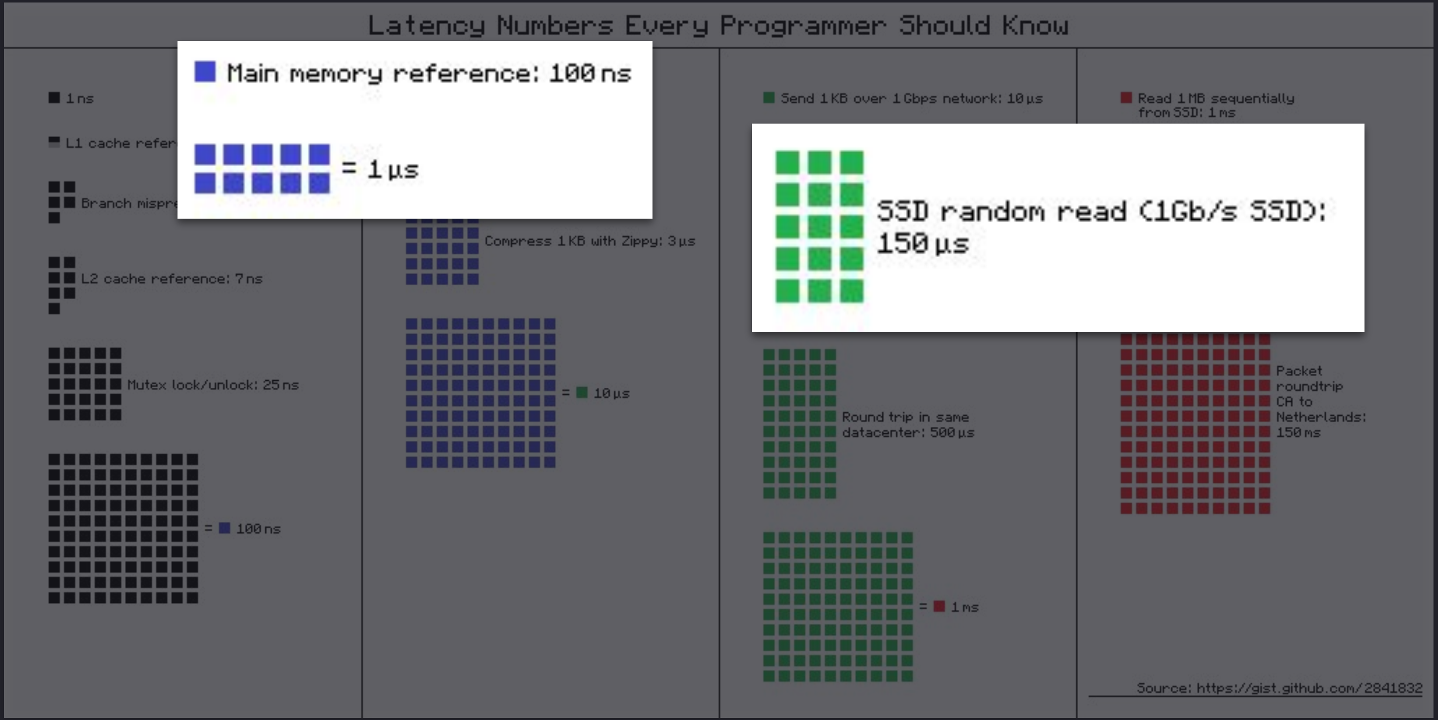
and then everything was perfect... the end

trade-offs

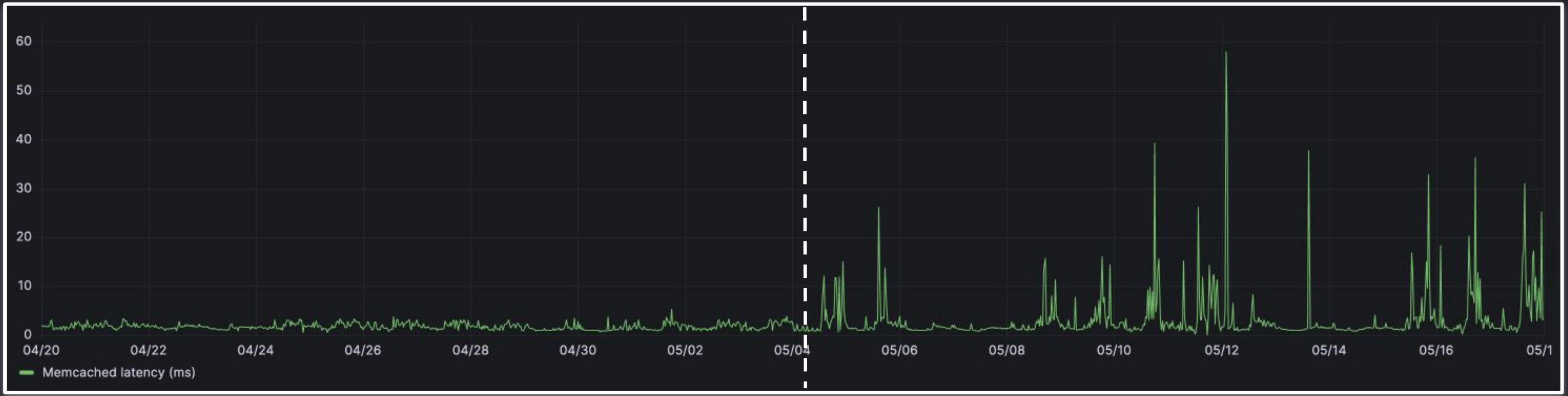
latency


Latency Numbers Every Programmer Should Know
1 ns
L1 cache reference: 0.5 ns
Branch mispredict: 5 ns
L2 cache reference: 7 ns
Mutex lock/unlock: 25 ns
= 100 ns
Main memory reference: 100 ns
= 1 µs
Compress 1 KB with Zippy: 3 µs
= 10 µs
Send 1 KB over 1 Gbps network: 10 µs
SSD random read (1Gb/s SSD): 150 µs
Read 1 MB sequentially from memory: 250 µs
Round trip in same datacenter: 500 µs
= 1 ms
Read 1 MB sequentially from SSD: 1 ms
Disk seek: 10 ms
Read 1 MB sequentially from disk: 20 ms
Packet roundtrip CA to Netherlands: 150 ms
Source: https://gist.github.com/2841832

latency


Latency Numbers Every Programmer Should Know
Main memory reference: 100 ns
= 1 µs
1 ns
L1 cache refer
Branch mispre
L2 cache reference: 7 ns
Mutex lock/unlock: 25 ns
= 100 ns
Compress 1 KB with Zippy: 3 µs
= 10 µs
Send 1 KB over 1 Gbps network: 10 µs
SSD random read (1Gb/s SSD): 150 µs
Round trip in same datacenter: 500 µs
= 1 ms
Read 1 MB sequentially from SSD: 1 ms
Packet roundtrip CA to Netherlands: 150 ms
Source: https://gist.github.com/2841832

latency



rollout

latency

250
200
150
100
50
0
04/20
04/22
04/24
04/26
04/28
04/30
05/02
05/04
05/06
05/08
05/10
05/12
05/14
05/16
05/1
P50 Memcached latency (ms)
P50 Store latency (ms)
rollout

durability vs volatility

Warning: The performance gains from local SSDs require certain trade-offs in availability, durability, and flexibility. Because of these trade-offs, Local SSD storage is **not** automatically replicated and **all data on the local SSD may be lost** if the VM stops for any reason. See Local SSD data persistence for details.

	What happens to your data?
-initiated instance lifecycle events	
The instance is rebooted	The data persists
The instance is stopped	The data does not persist
The instance is hibernated	The data does not persist
The instance is terminated	The data does not persist
The instance type is changed	The data does not persist *
A Windows AMI is created from the instance	The data does not persist in the created AMI **
An EBS-backed AMI is created from the instance	The data does not persist in the created AMI **
An instance store-backed AMI is created from the instance	The data persists in the AMI bundle uploaded to Amazon S3 ***
User-initiated OS events	
A shutdown is initiated	The data does not persist †
A restart is initiated	The data persists
AWS scheduled events	
Instance stop	The data does not persist
Instance reboot	The data persists
System reboot	The data persists
Instance retirement	The data does not persist
Unplanned events	

disk-related issues

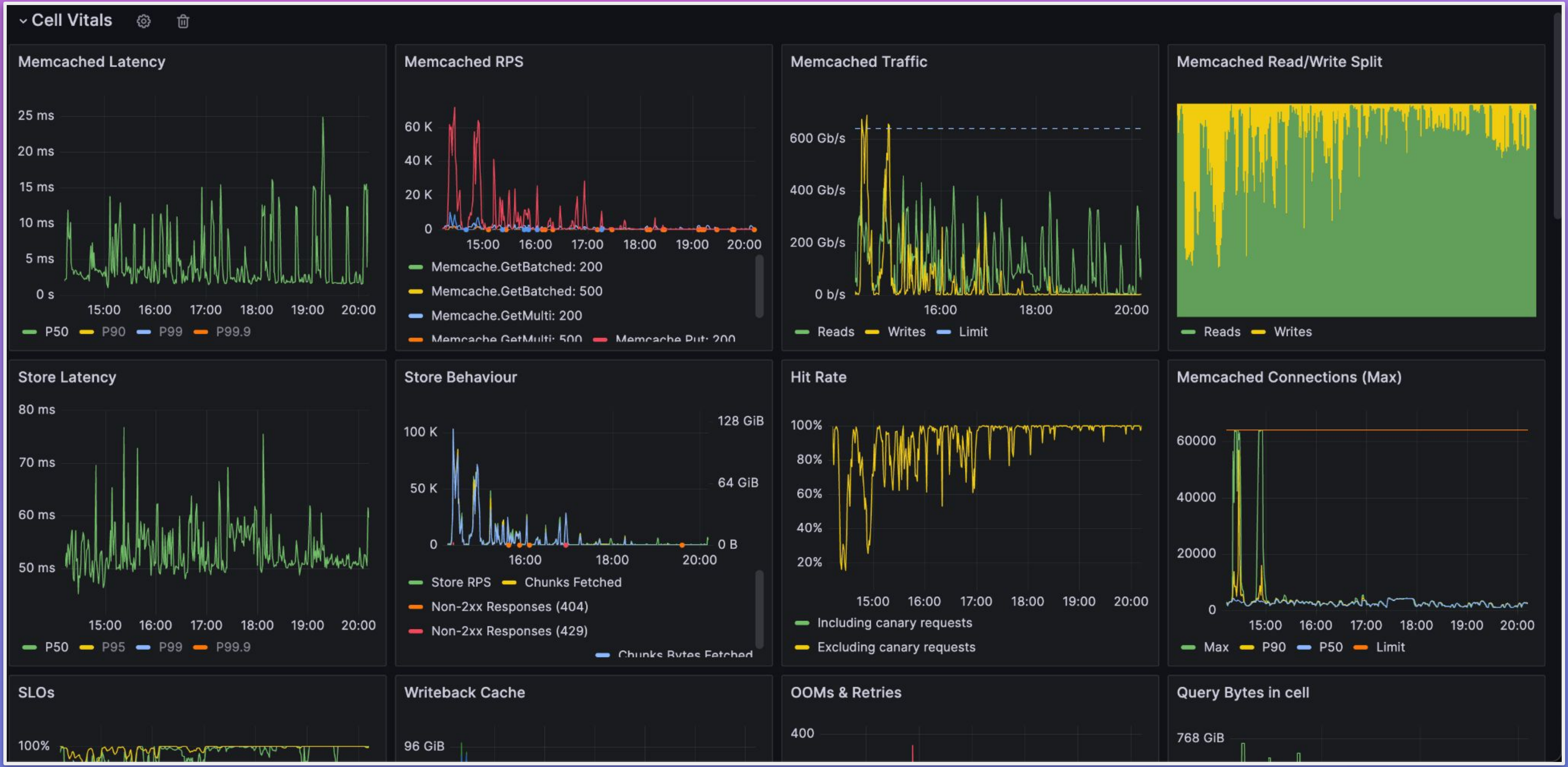
- disk fills up
- disk has bad sectors
- disk ages, performs poorly

solution:

kill the disk!

observability

- **prometheus**
  - memcached-exporter
  - node-exporter
- **alertmanager**
  - disk full alert
  - disk latency alert
  - memcached latency alert
- **grafana & loki** (shocker!)


Cell Vitals
Memcached Latency
25 ms
20 ms
15 ms
10 ms
5 ms
0 s
15:00 16:00 17:00 18:00 19:00 20:00
P50 P90 P99 P99.9
Memcached RPS
60 K
40 K
20 K
0
15:00 16:00 17:00 18:00 19:00 20:00
Memcache.GetBatched: 200
Memcache.GetBatched: 500
Memcache.GetMulti: 200
Memcache.GetMulti: 500
Memcache.Put: 200
Memcached Traffic
600 Gb/s
400 Gb/s
200 Gb/s
0 b/s
16:00 18:00 20:00
Reads Writes Limit
Memcached Read/Write Split
Reads Writes
Store Latency
80 ms
70 ms
60 ms
50 ms
15:00 16:00 17:00 18:00 19:00 20:00
P50 P95 P99 P99.9
Store Behaviour
100 K
50 K
0
128 GiB
64 GiB
0 B
16:00 18:00 20:00
Store RPS
Chunks Fetched
Non-2xx Responses (404)
Non-2xx Responses (429)
Chunks Bytes Fetched
Hit Rate
100%
80%
60%
40%
20%
15:00 16:00 17:00 18:00 19:00 20:00
Including canary requests
Excluding canary requests
Memcached Connections (Max)
60000
40000
20000
0
15:00 16:00 17:00 18:00 19:00 20:00
Max P90 P50 Limit
SLOs
100%
Writeback Cache
96 GiB
OOMs & Retries
400
Query Bytes in cell
768 GiB

implications for the future of cloud databases?

- cost is becoming compelling
- capacious disks at near-DRAM speeds
- higher network throughput
    - getting higher as network is offloaded from CPU

Grafana Labs

shoutout to
Ed & Alan!

thanks!
questions?

check out the
blog post for
more details

@dannykopping

why not Memorystore / Elasticache?

Memorystore for Memcached

37 x Cache

vCPU usage: 135,050 vCPU hours

RAM usage: 40,515,000 GiB hours

Location: us-central1

USD 367,336.00

Total Estimated Cost: USD 367,336.00 per 1 month

Estimate Currency

USD - US Dollar

Configure Amazon ElastiCache Info

Cluster settings Info

Nodes

500

Nodes can't be more than 250.

cache.m6g.8xlarge

Selected Instance:

cache.m6g.8xlarge

vCPU: 32

Memory: 103.68 GiB

Network Performance: 12 Gigabit

Utilization (On-Demand only)

Elasticache Instances will be always at 100% utilization, to stop incurring char

100

Total Upfront cost: 0.00 USD

Total Monthly cost: 865,780.00 USD

Show Details