# PostgreSIEVE: Implementing SIEVE in the Postgres Bufferpool

Ryan Crosier, Will Garlington, Jackson Gilstrap

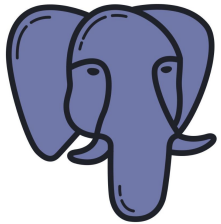
# Motivation

SIEVE is Simpler than LRU

ACEing the Bufferpool Management

Compared new policy to existing policies against a web cache workload

Aimed at addressing asymmetry in the buffer manager in an industry system

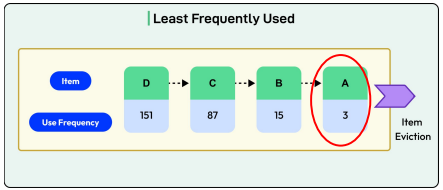


Postgres

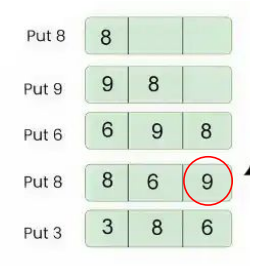
+ SIEVE

# Background

## Primitive eviction policies

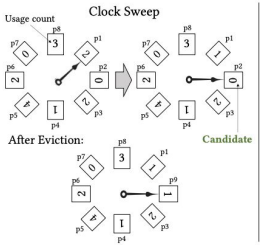



LRU




FIFO


## Fine tuned eviction policies




ARC

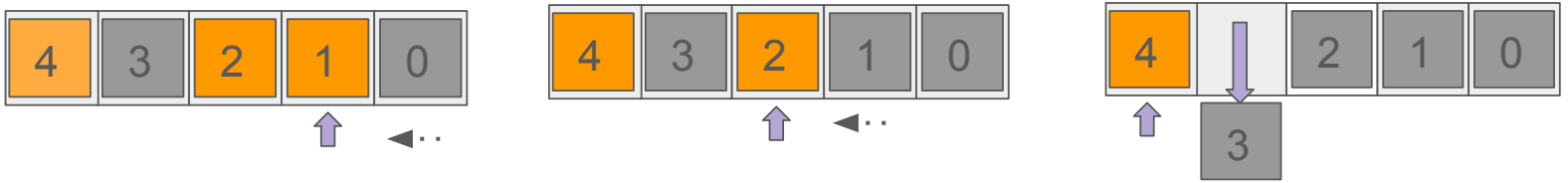
```
. . . [   B1  <-[     T1     <-!->      T2   ]-> B2   ] . .
      [ . . . . [ . . . . . . . ! . .^. . . . ] . . . . ]
                [   fixed cache size (c)    ]
```

Tradeoff

Implementation ⟺ Performance

SIEVE

# What is SIEVE




**Primary benefit - Quick Demotion**

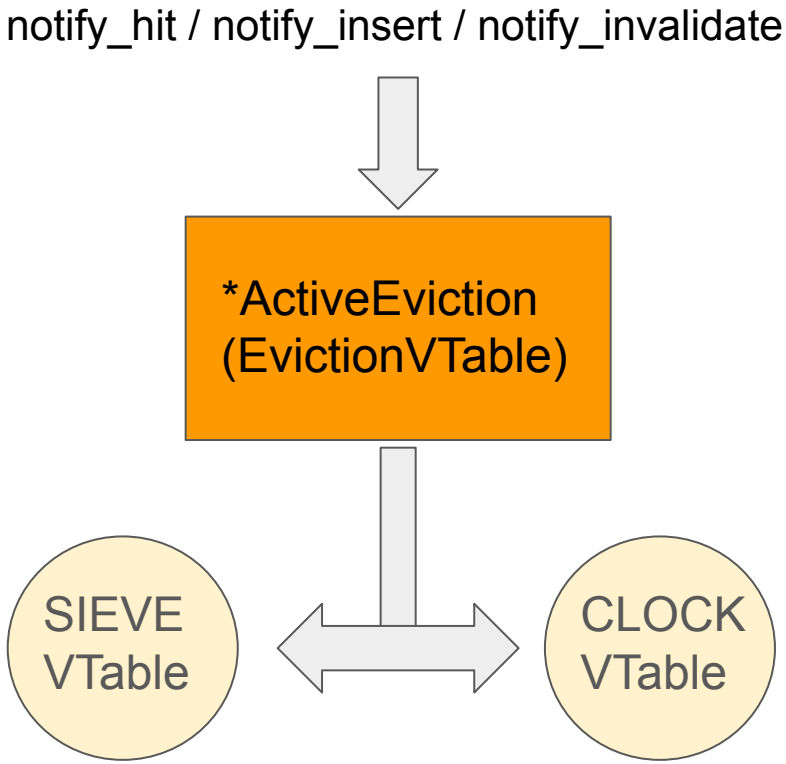
Unlike LRU which keeps unpopular items in the cache for extended periods of time, SIEVE marks unpopular items for eviction at the first possible chance so that in the next round it is removed from cache

# Software Architecture

**Strategy Common & Eviction Table**: Two structs which work in tandem to facilitate a common eviction architecture for policy extension.

**StrategyCommon** - Defines activeStrategy and controls locking and other metadata

**EvictionTable**: Defines common methods for eviction. Buffer eviction policies implement EvictionVTable and function calls are routed to policy functions.



# SIEVE Implementation

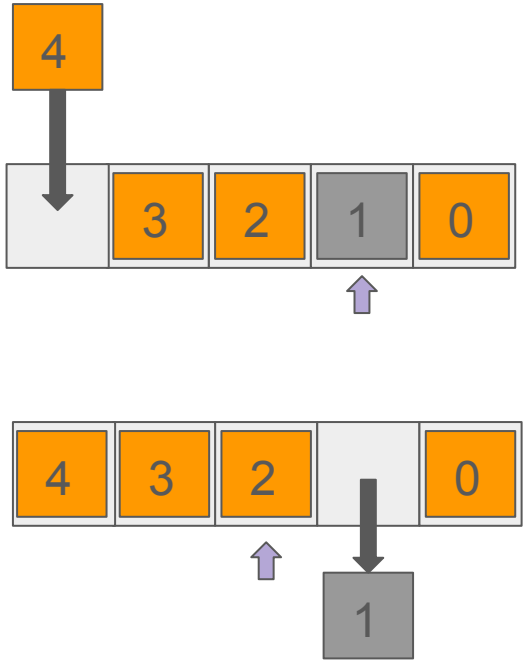
- Simulate a list as fixed array
- **NBuffers** is set → array can be static

notify_insert:

- New buffer(id) is inserted at head of list (or re-inserted)
- Absolute index vals are updated

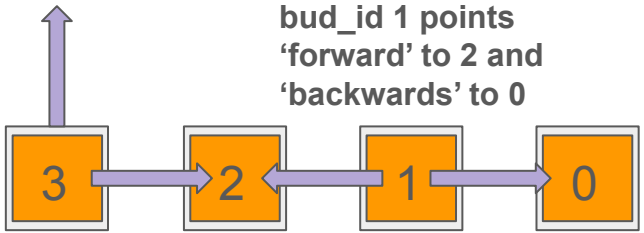
get_buffer:

- Victim is selected via SIEVE policy (hand advances until unused buffer is found)
- Additional checks are performed to see if buffer is pinned externally
- Buffer is unlinked from list and sieve_hand is advanced

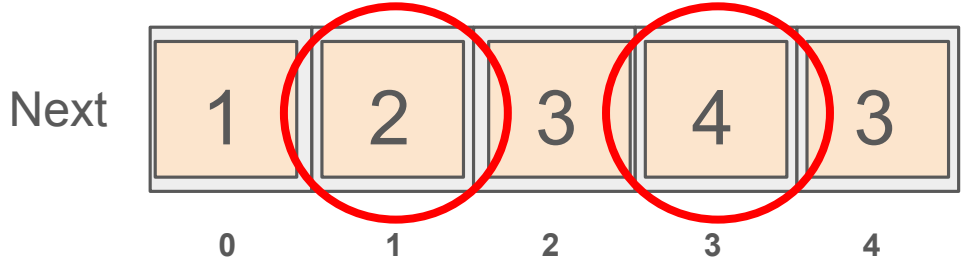


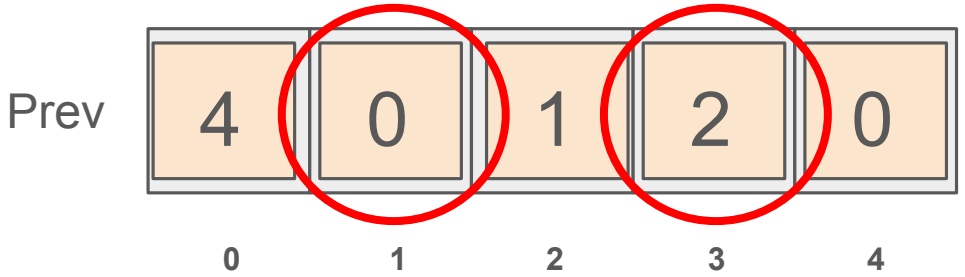
# Linked List Details

Next points to NBUFF, so this node is at the head of the list




Prev points to index 2, so bud_id 2 is 'down' the list



# Experimental Setup

Hardware

- NVME and SATA SSDs
- 6-core 3.7/4.2 GHz CPU
- 32GB DDR4 Memory
- Debian-based linux environment

Postgres

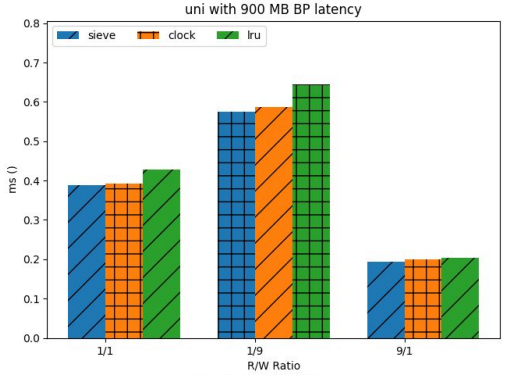
- Data stored on NVME SSD
- WAL stored on a separate SATA SSD

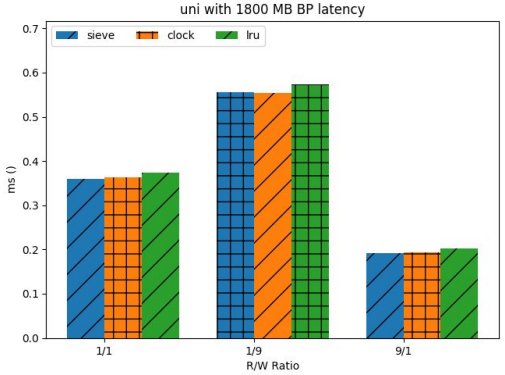
# Experimental Methodology

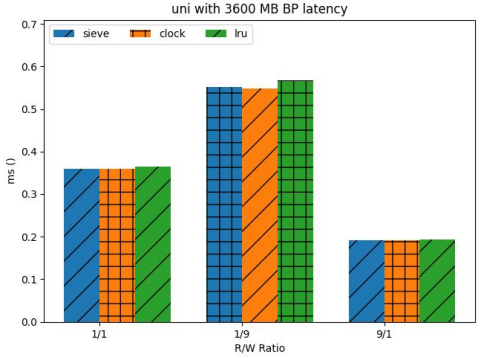
- **pgbench** used for synthetic benchmarking (as in ACE paper)
- We use different transactions scripts to simulate point queries and updates
- We weight scripts in benchmarks to vary **workload skew**
- We draw from uniform and zipfian distributions to vary **data skew**
  - A zipfian parameter of about 1.01 gives us a 90/10 locality
- We run each test for **3 minutes**
- 12 clients and 6 threads used unless otherwise stated
- We disable autovacuuming and vacuum manually after each test

# Results: SIEVE Slightly Outperforms with Low Memory

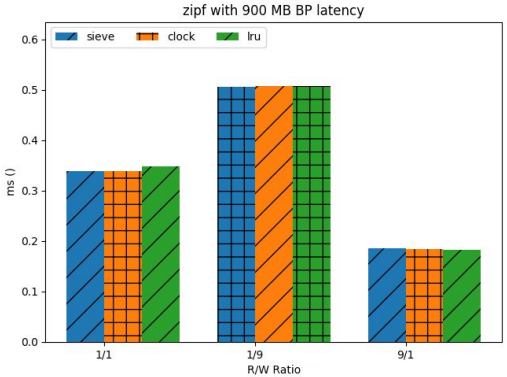
Uniform Workload

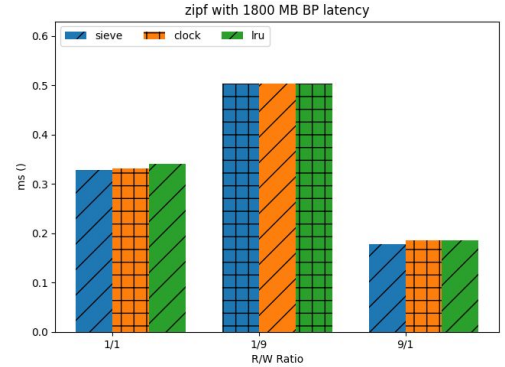


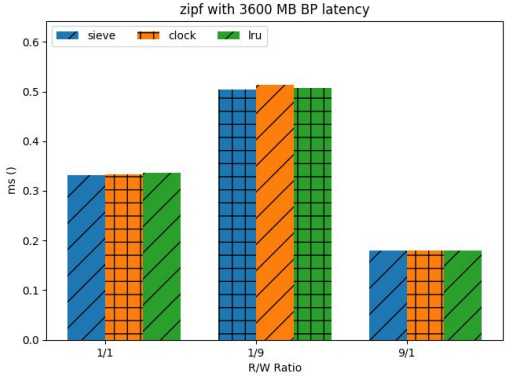




Skewed Workload

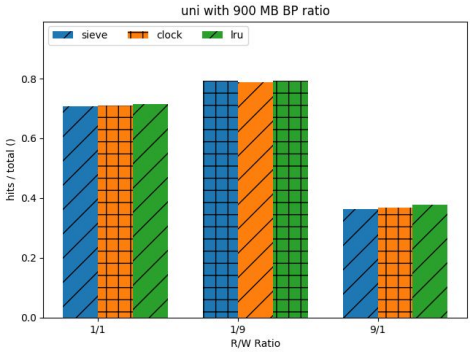


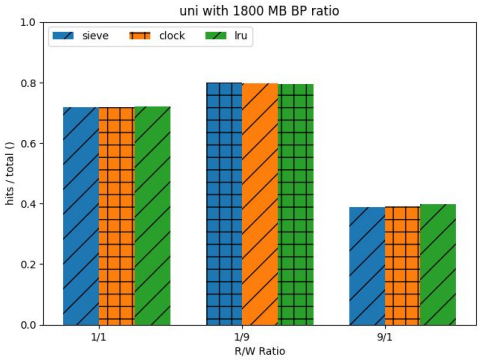



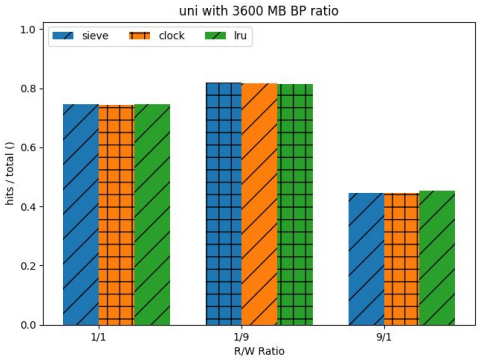




# Results: Similar Cache Hit Ratios

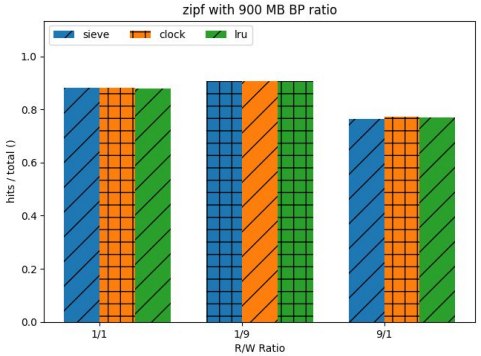
Uniform Workload

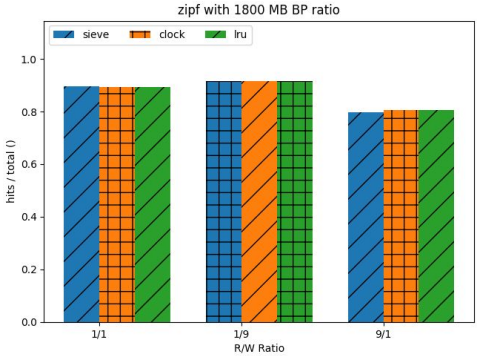


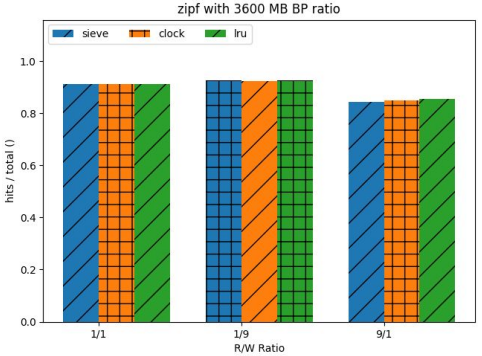



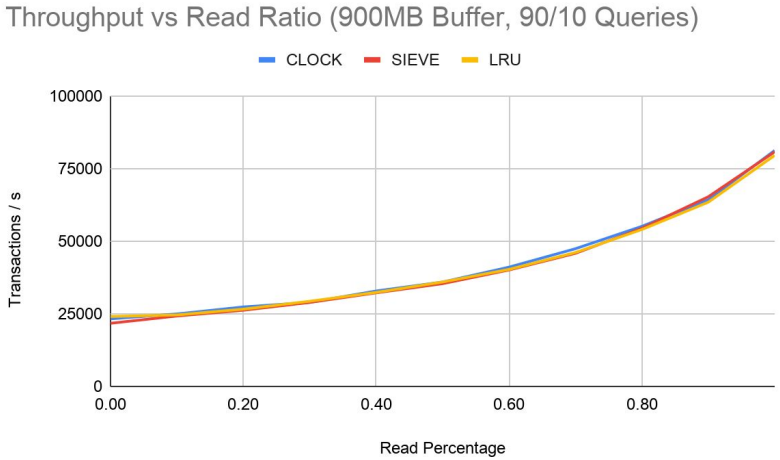

Skewed Workload

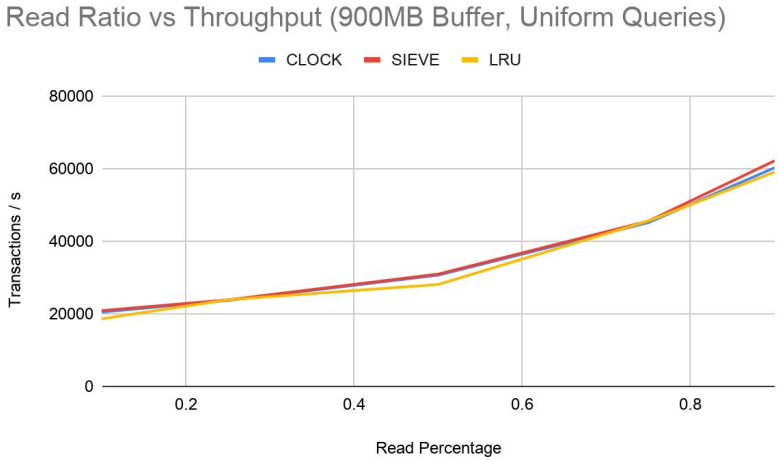


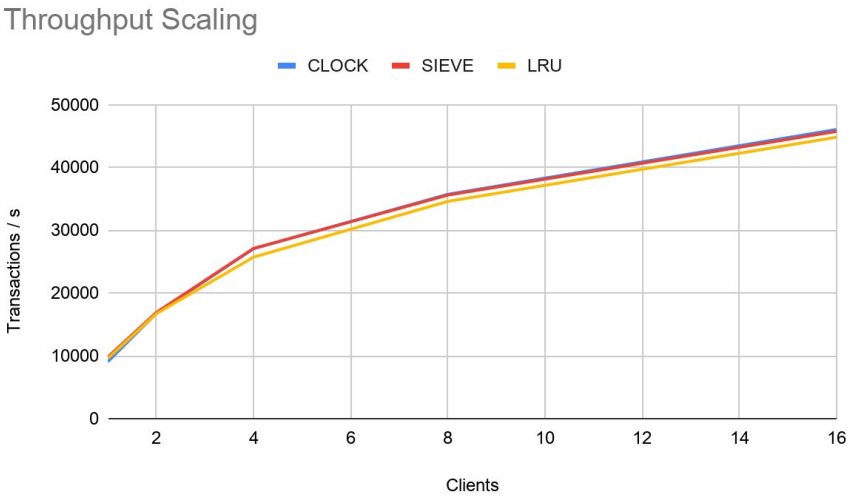




# Results: SIEVE Matches CLOCK



# Results: Client Scaling Falls Behind



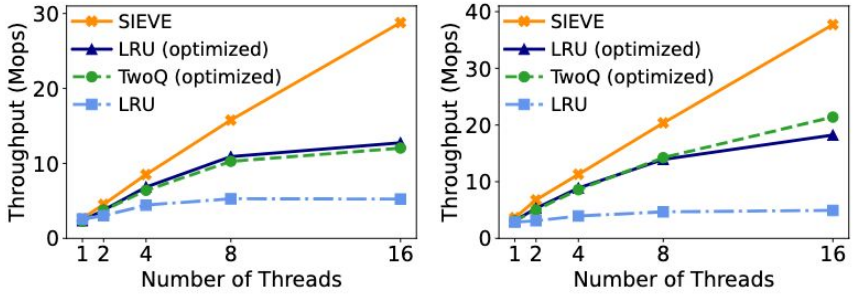
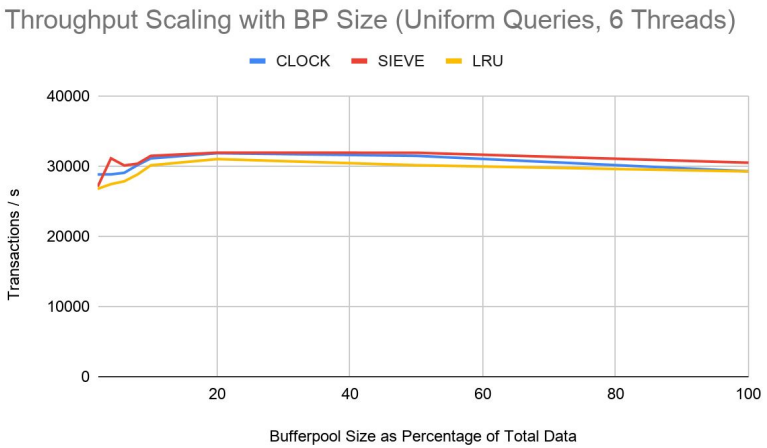



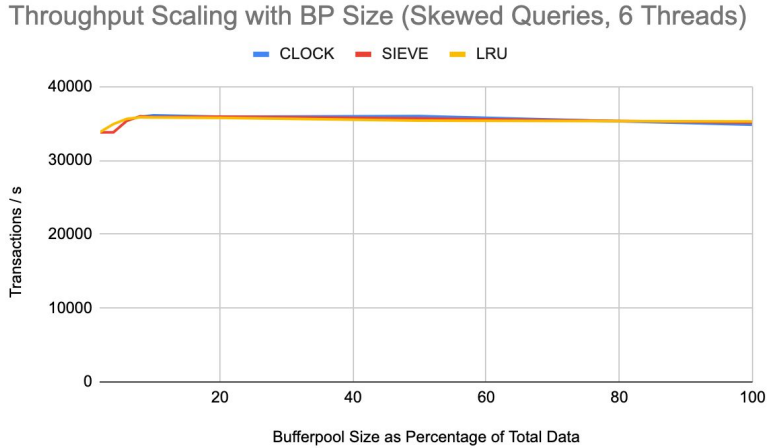

**Figure 6:** Throughput scaling with CPU cores on two KV-cache workloads.

**Theory:** Cache performance is more impactful on KV stores than on relational queries.

**Bottom Line:** These performance gains require the workload to be very I/O bound

# Results: BP Size has minimal effects on mixed workloads




**Problem:** Even though our cache hit rate increases, our throughput does not! Things we tried to ameliorate: varying thread count, turning off fsync…

**Theory:** Our synthetic workloads do not benefit greatly from increased cache size beyond a certain point.

# Conclusion and Future Work

SIEVE excels at the KV-Store workloads presented in the original paper, but does not alter performance significantly in synthetic benchmarks in a relational DB. This can be attributed to the increased complexity of systems around the cache.

Future Work:

- Investigate the performance impacts of caching algorithms in SQL over NoSQL systems
- Investigate the performance of SIEVE under scan-heavy workloads