`cache_ext`: Customizing the Page Cache with eBPF

Tal Zussman Columbia
Ioannis Zarkadas Columbia
Jeremy Carin Columbia
Andrew Cheng Columbia
Hubertus Franke IBM
Jonas Pfefferle IBM
Asaf Cidon Columbia

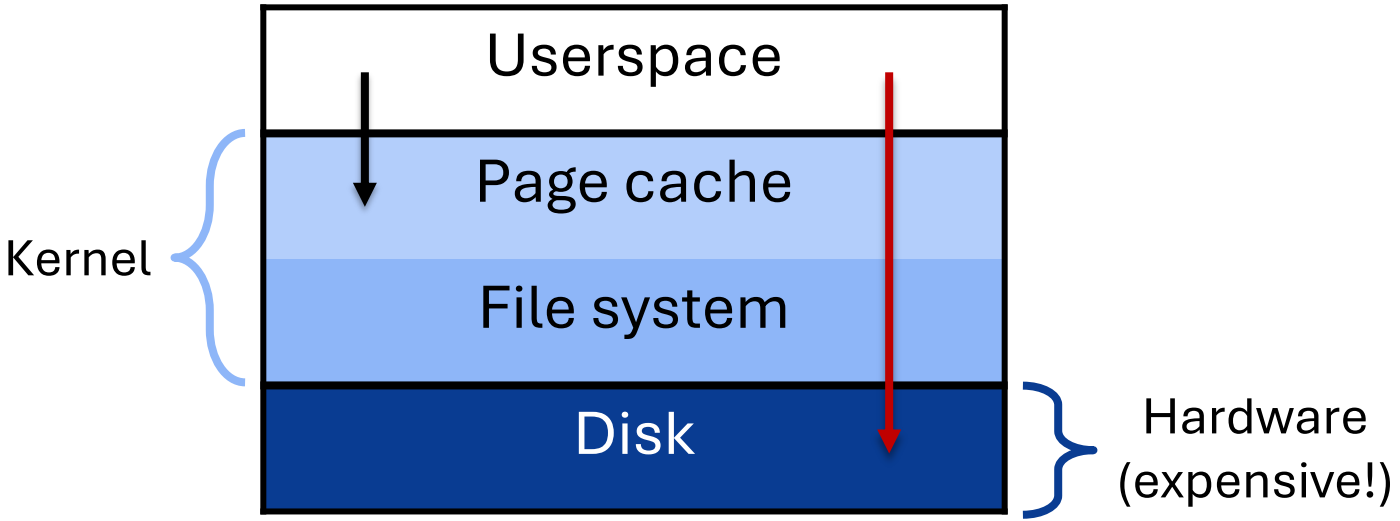
Columbia University
IBM Research

TLDR;

- The page cache doesn’t work so well with some workloads…
- cache_ext allows applications to customize page cache policies to improve performance using eBPF
- Experimentation with policies and application hints are necessary to maximize performance

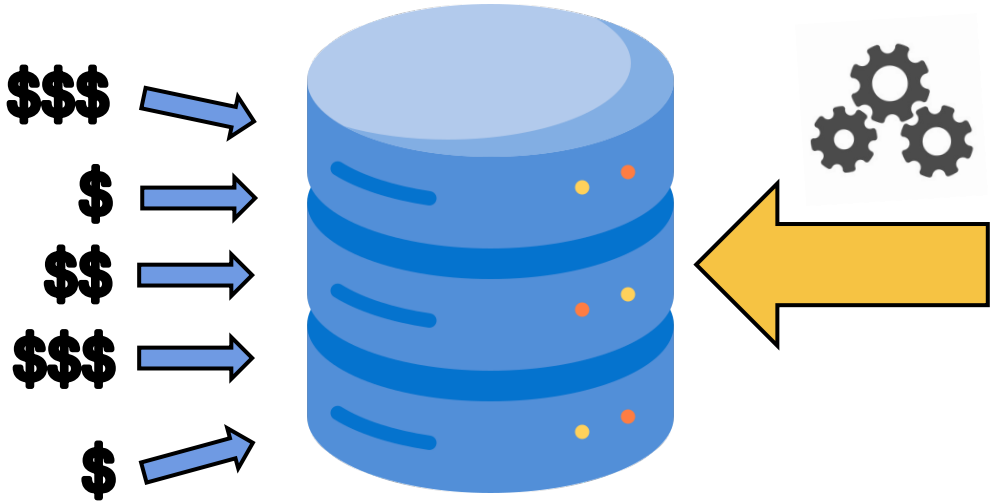
Linux page cache significantly affects performance

LRU-approximation algorithm – **doesn't work well for all workloads!**



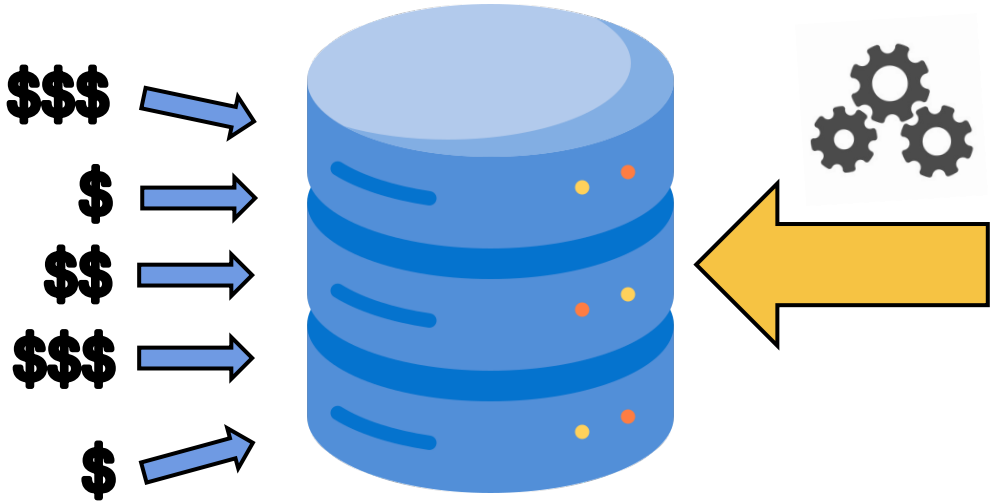
Linux page cache struggles with some workloads

- Many tiny transactions
- A few massive background tasks



Linux page cache struggles with some workloads

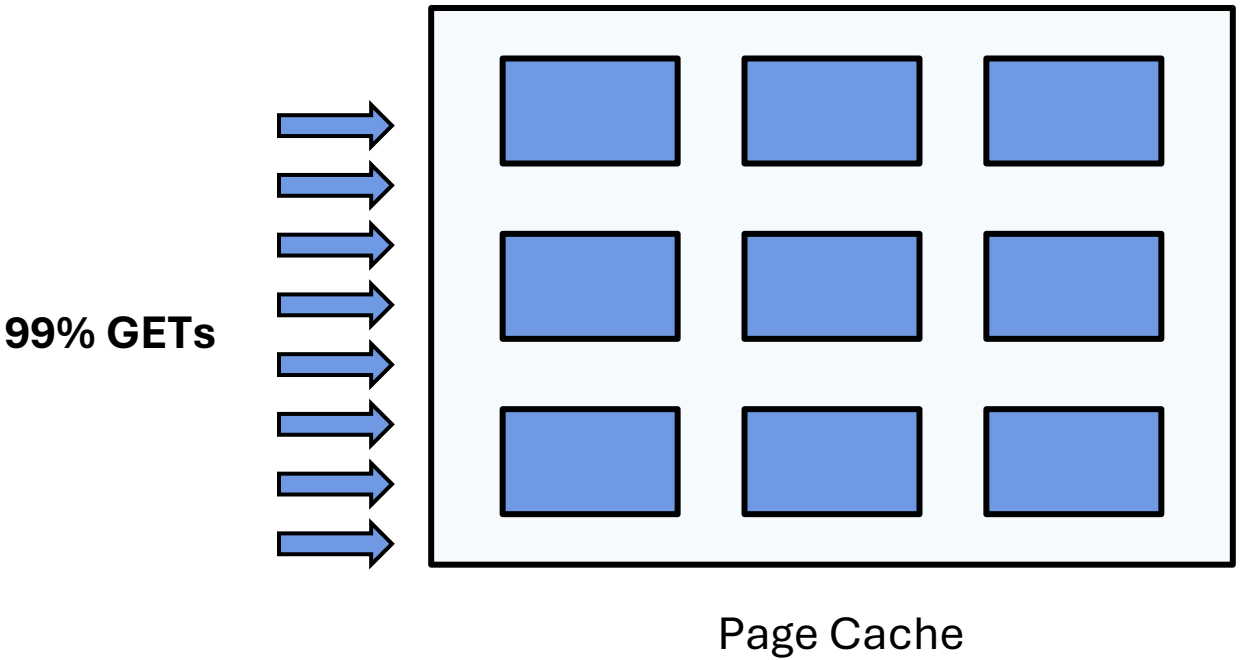
- Many tiny transactions
- A few massive background tasks



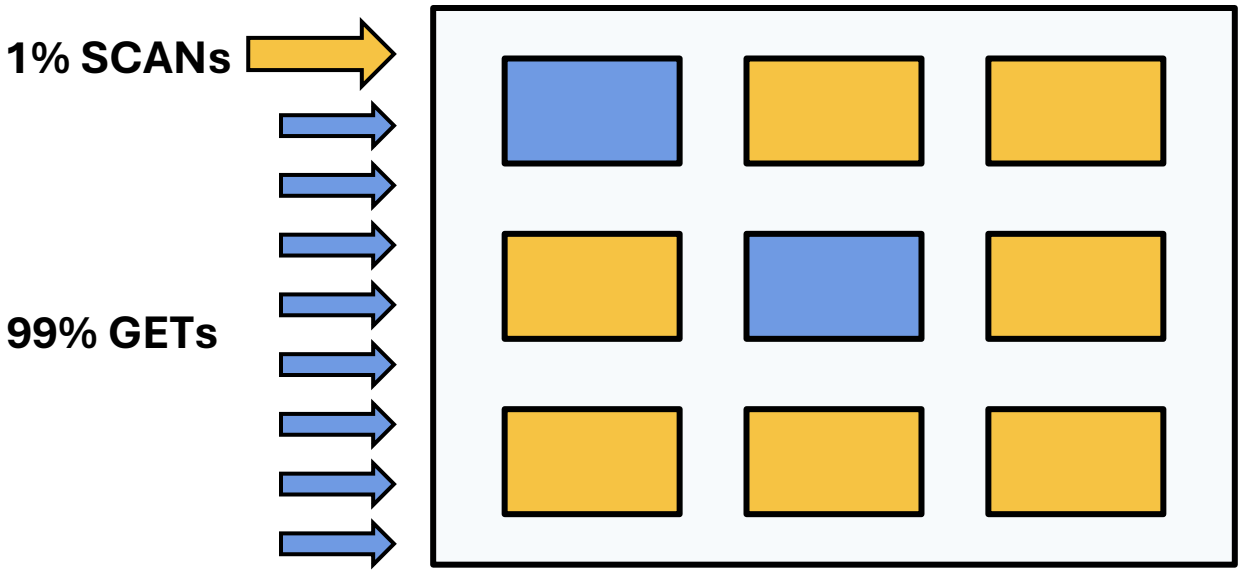
HTAP-like:
Mix of Transactional (GET) and Analytical (SCAN)

Shinjuku (NSDI ’19),
Syrup (SOSP ‘21)

Linux page cache struggles with some workloads



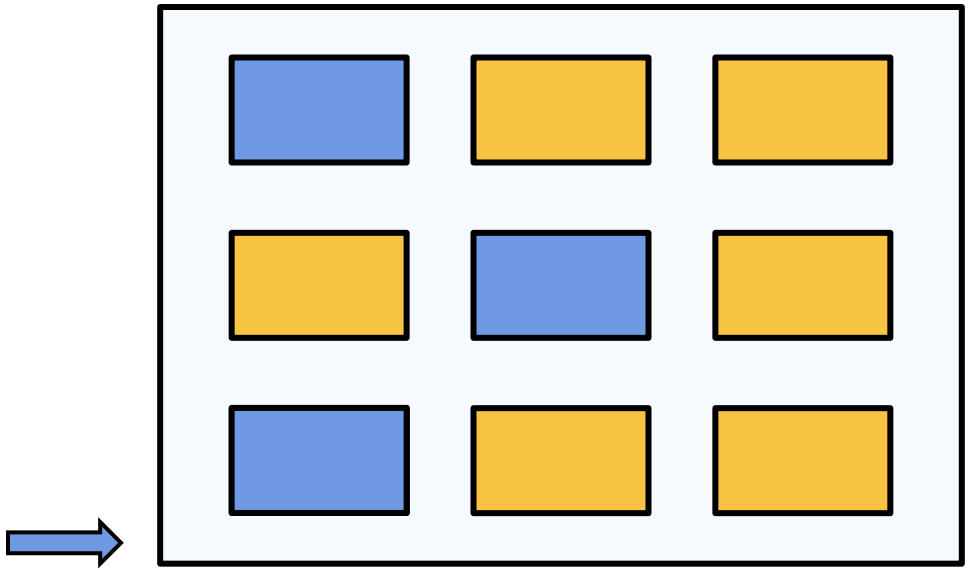
Linux page cache struggles with many workloads



Data is 99% SCAN and 1% GET

Linux page cache struggles with many workloads

We've been thrashed!



Page Cache

The page cache is not flexible enough

We are leaving performance on the table

Existing solutions

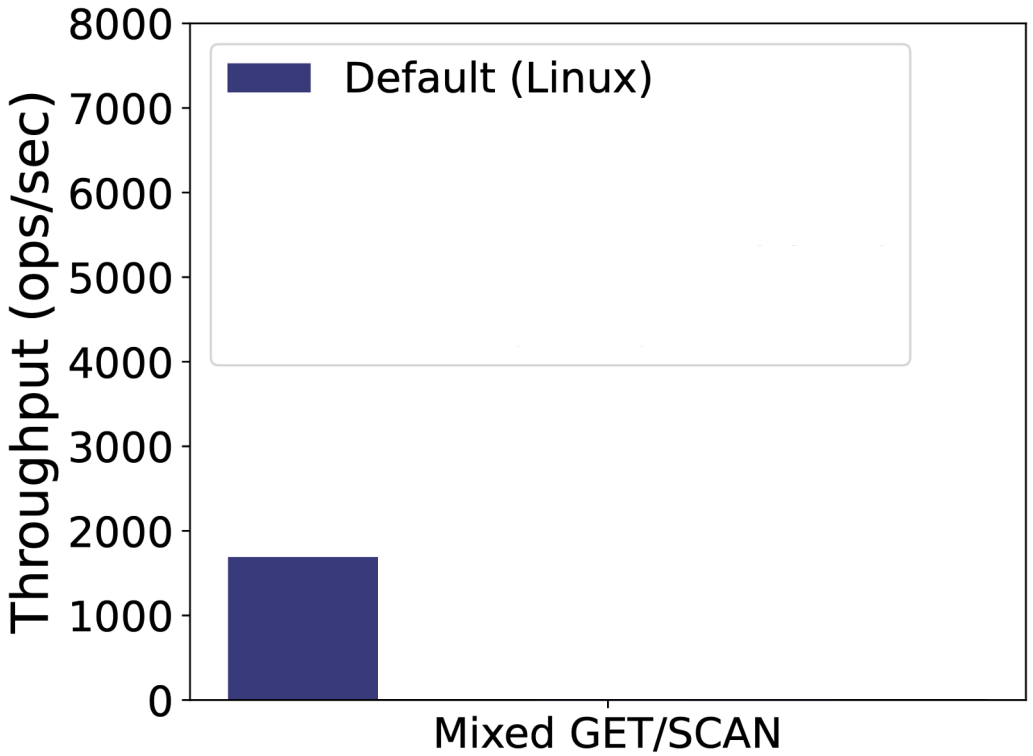
- Change the policy
- Userspace hints
- Application-level caching

Changing the policy is hard

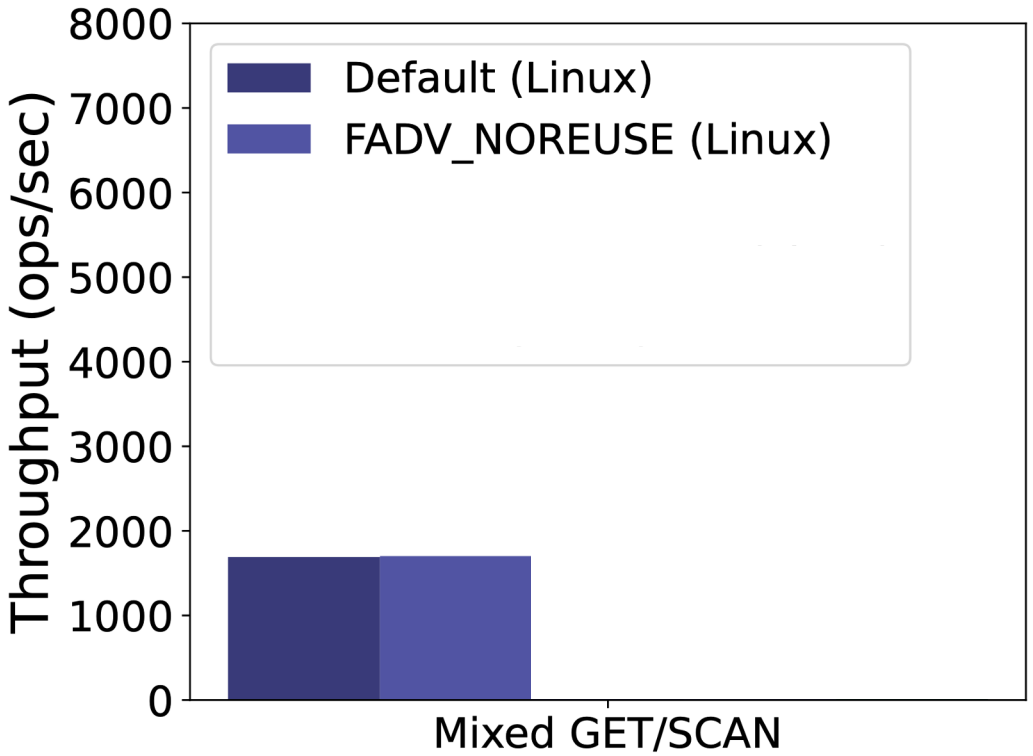
MGLRU merged in Linux v6.1 as new experimental policy

- Several years of effort
- Still not default upstream
- Global policy change

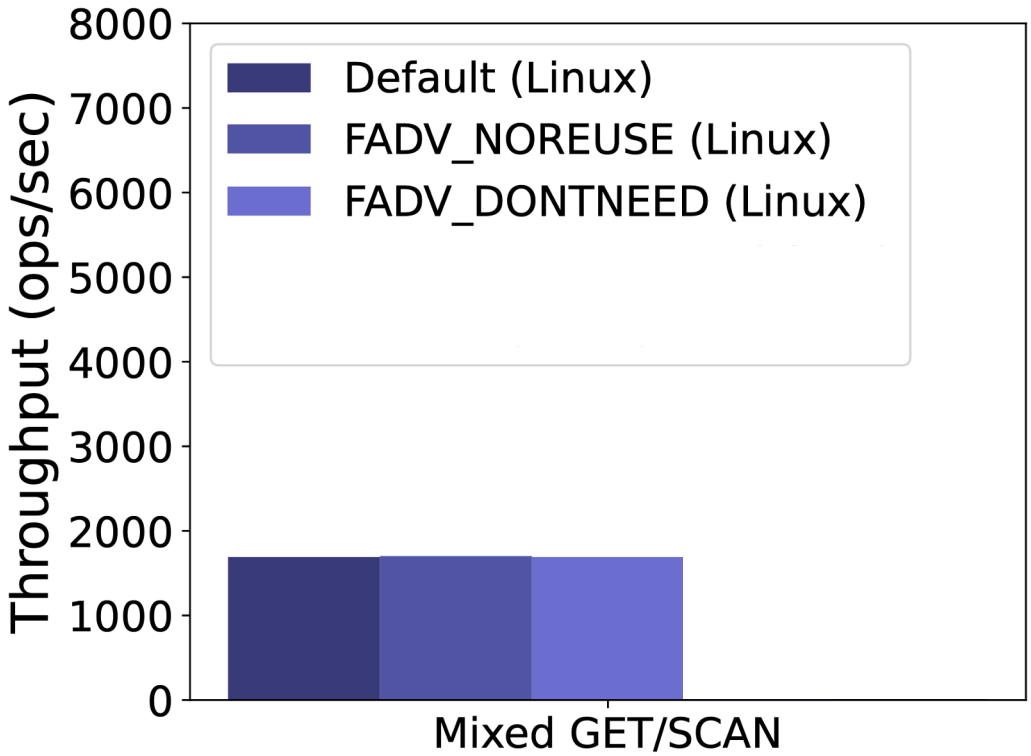
Existing hint interfaces are insufficient



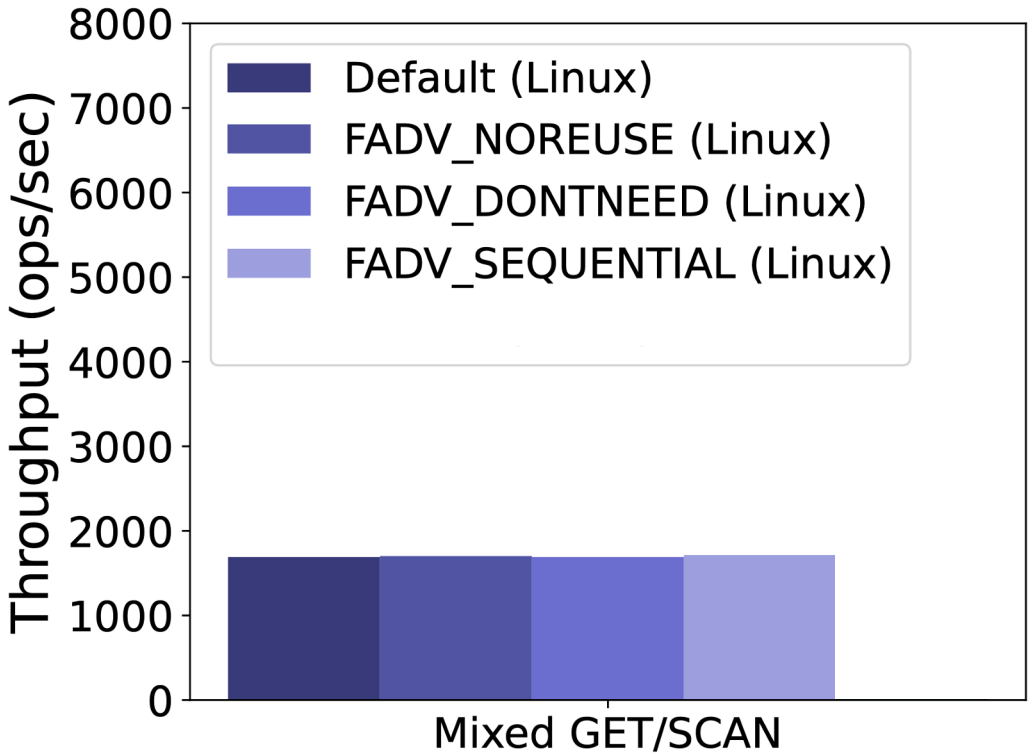
Existing hint interfaces are insufficient



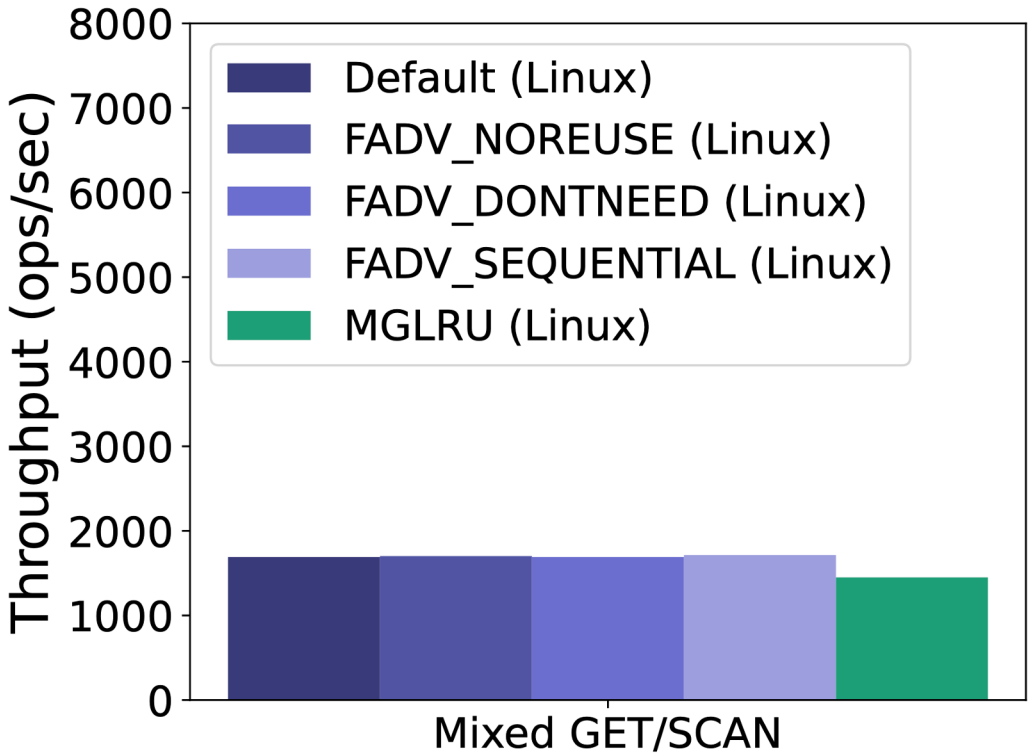
Existing hint interfaces are insufficient



Existing hint interfaces are insufficient



Existing hint interfaces are insufficient



Hints don’t work!

How can we improve page cache behavior?

	Change the policy	Userspace hints	Application-level caching
Customizability	✅	❌	✅
Simplicity	❌	✅	❌
Isolation	❌	✅	✅ / ❌

We need to change the underlying policy

Like scheduling, networking, file systems, etc.

cache_ext: Custom page cache policies

- Simplicity: No kernel changes for developers ✅
- Isolation: Per-cgroup policies ✅
- Customizability: eBPF ✅

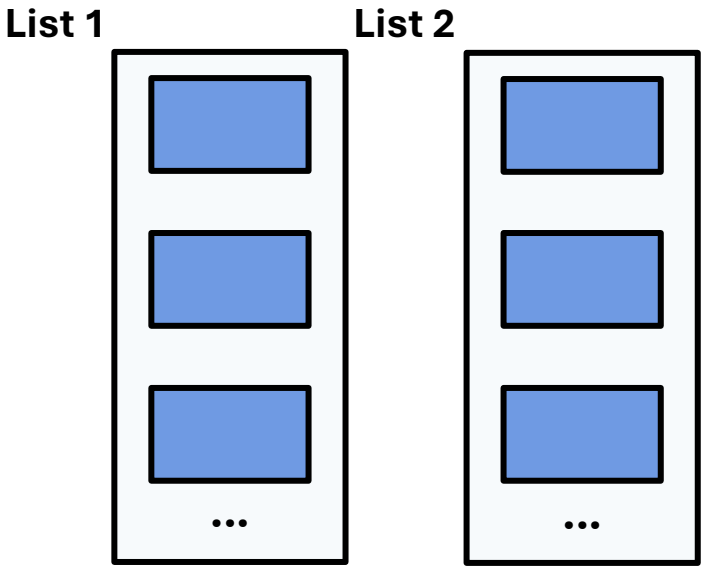
cache_ext interface

- struct_ops hooks added to page cache implementation
- ~200 LoC in page cache code (mostly batching eviction)

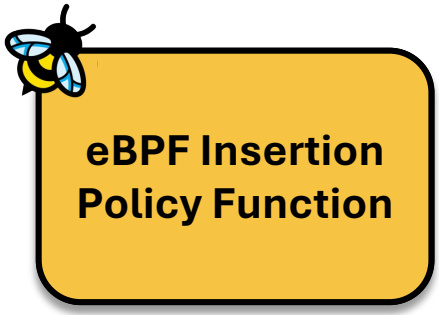
```
// Policy function hooks
struct cache_ext_ops {
    s32  (*policy_init)(struct mem_cgroup *memcg);
    // Propose folios to evict
    void (*evict_folios)(struct eviction_ctx *ctx,
         struct mem_cgroup *memcg);
    void (*folio_added)(struct folio *folio);
    void (*folio_accessed)(struct folio *folio);
    // Folio was removed: clean up metadata
    void (*folio_removed)(struct folio *folio);
    char name[CACHE_EXT_OPS_NAME_LEN];
};

struct eviction_ctx {
    u64 nr_candidates_requested; /* Input  */
    u64 nr_candidates_proposed;  /* Output */
    struct folio *candidates[32];
};
```

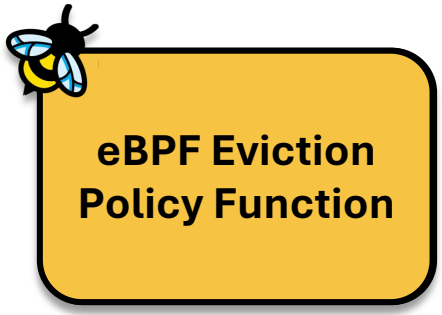
cache_ext: Simple API yields powerful policies

eBPF Eviction Lists



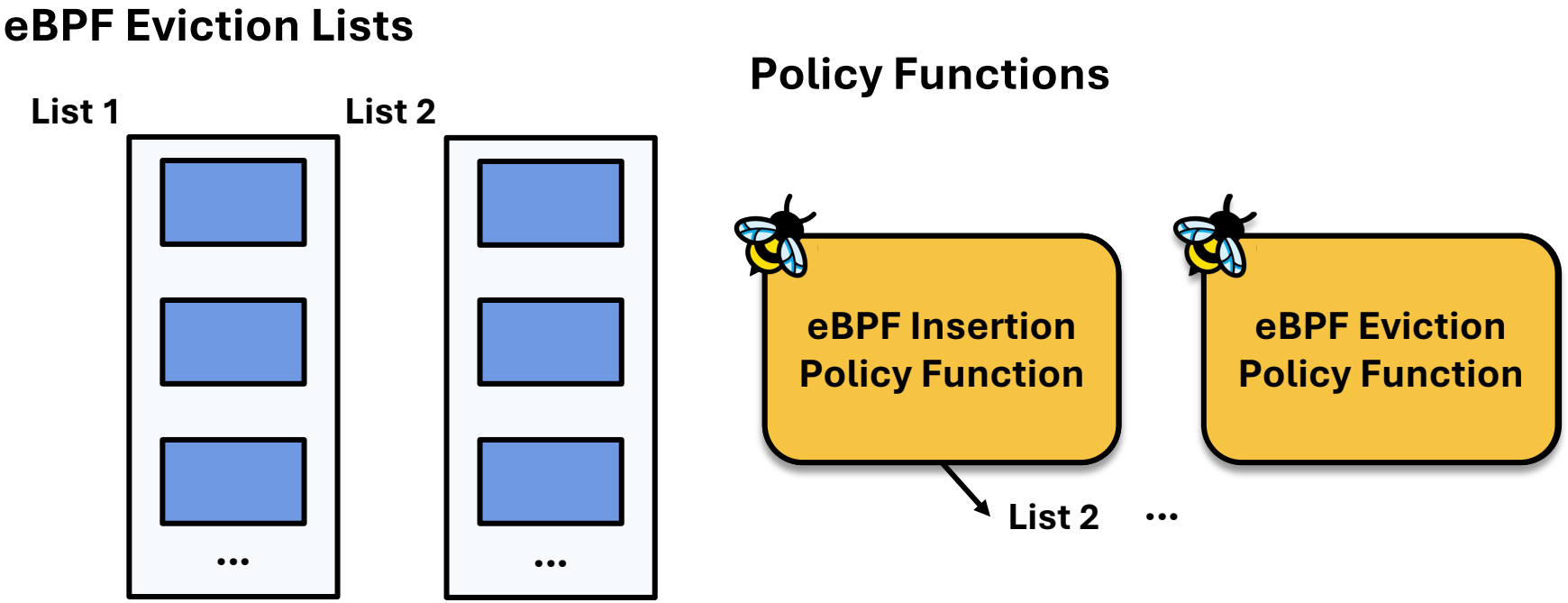
Policy Functions





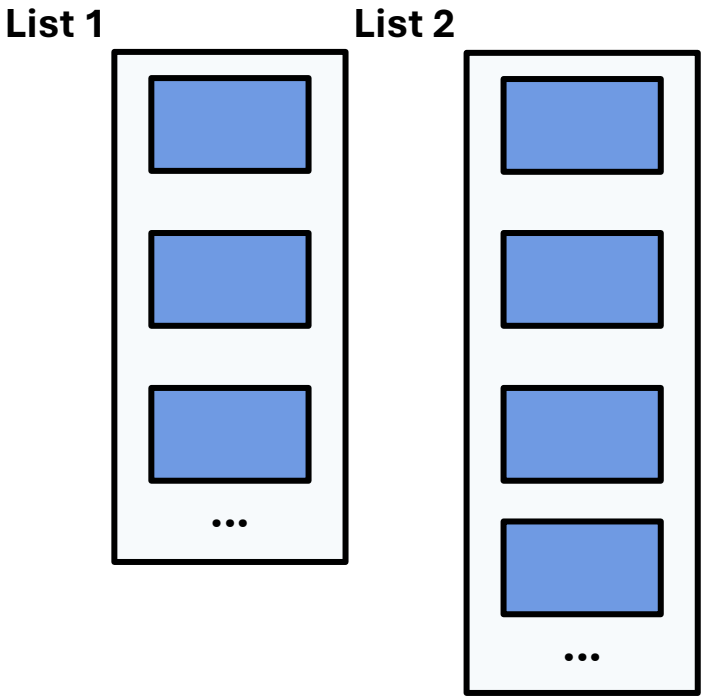
...

cache_ext: Simple API yields powerful policies

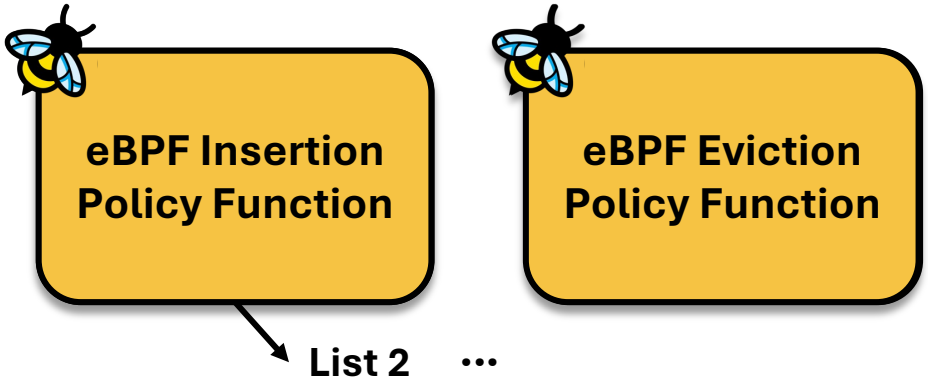


cache_ext: Simple API yields powerful policies

eBPF Eviction Lists

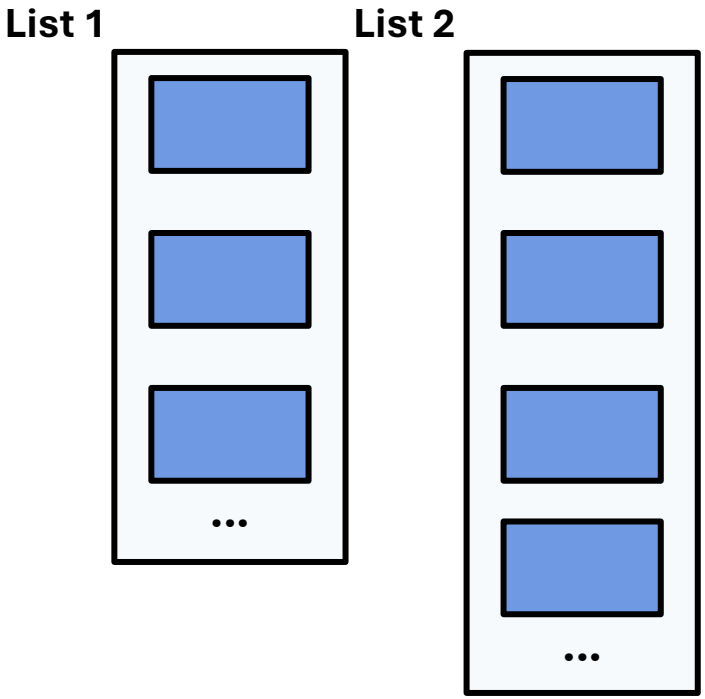


Policy Functions

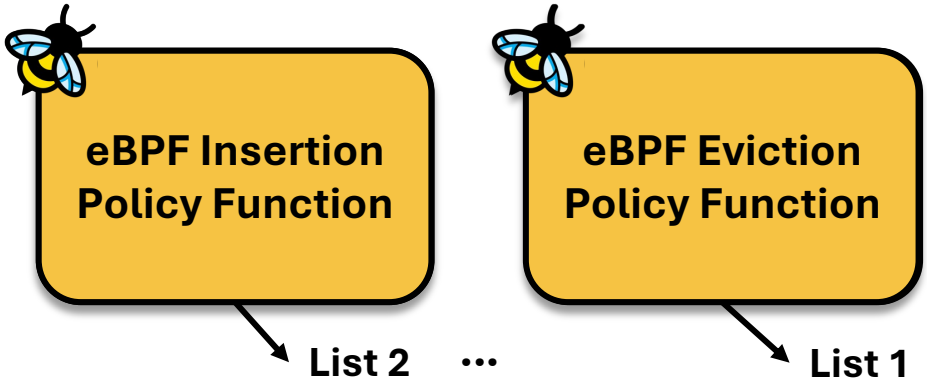


cache_ext: Simple API yields powerful policies

eBPF Eviction Lists

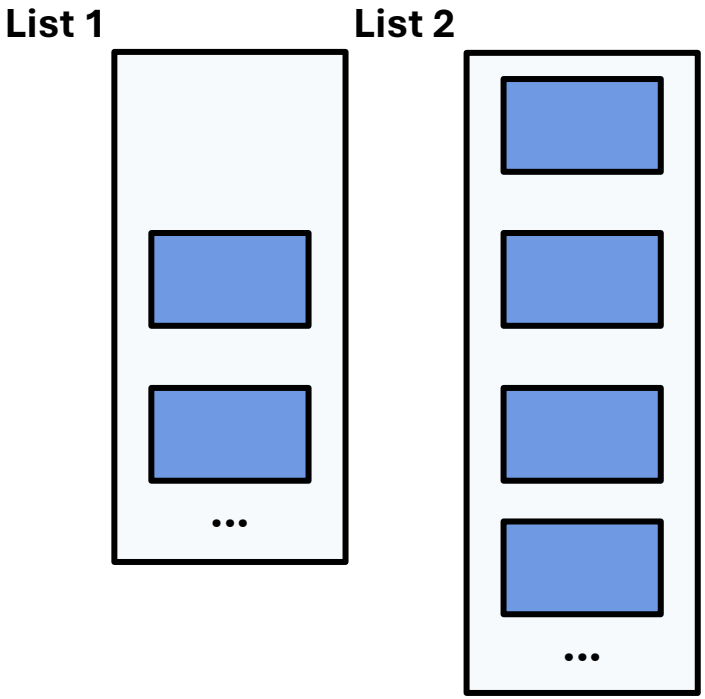


Policy Functions

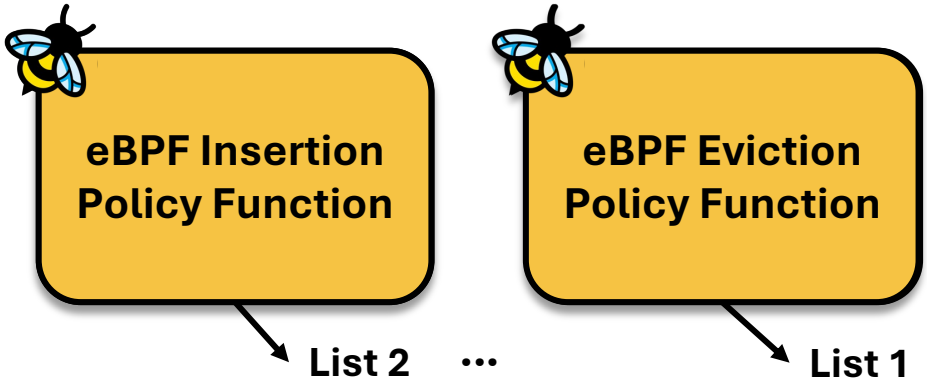


cache_ext: Simple API yields powerful policies

eBPF Eviction Lists

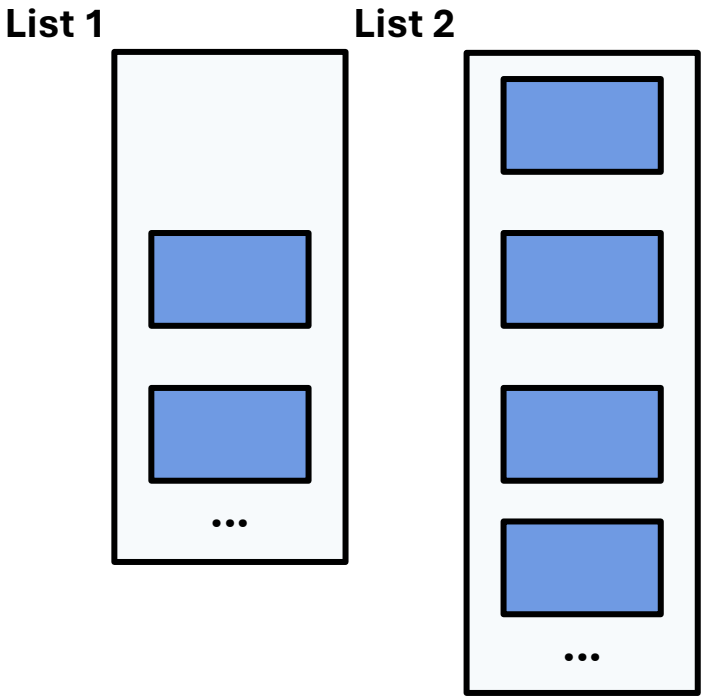


Policy Functions

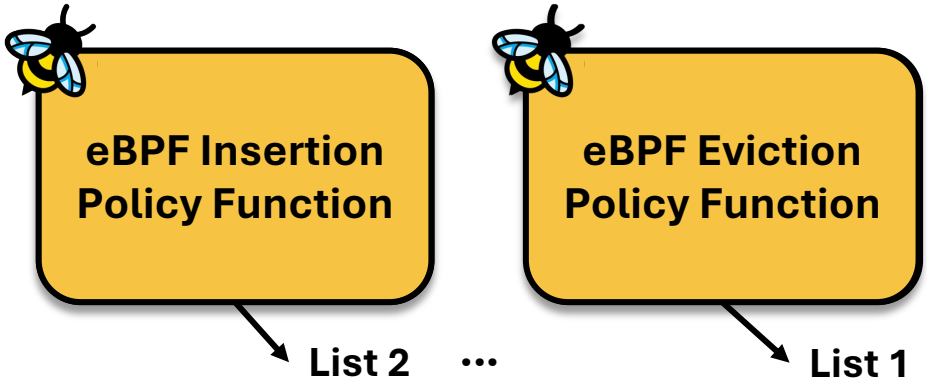


cache_ext: Simple API yields powerful policies

eBPF Eviction Lists



Policy Functions



Most policies can be implemented with lists!

cache_ext list kfuncs

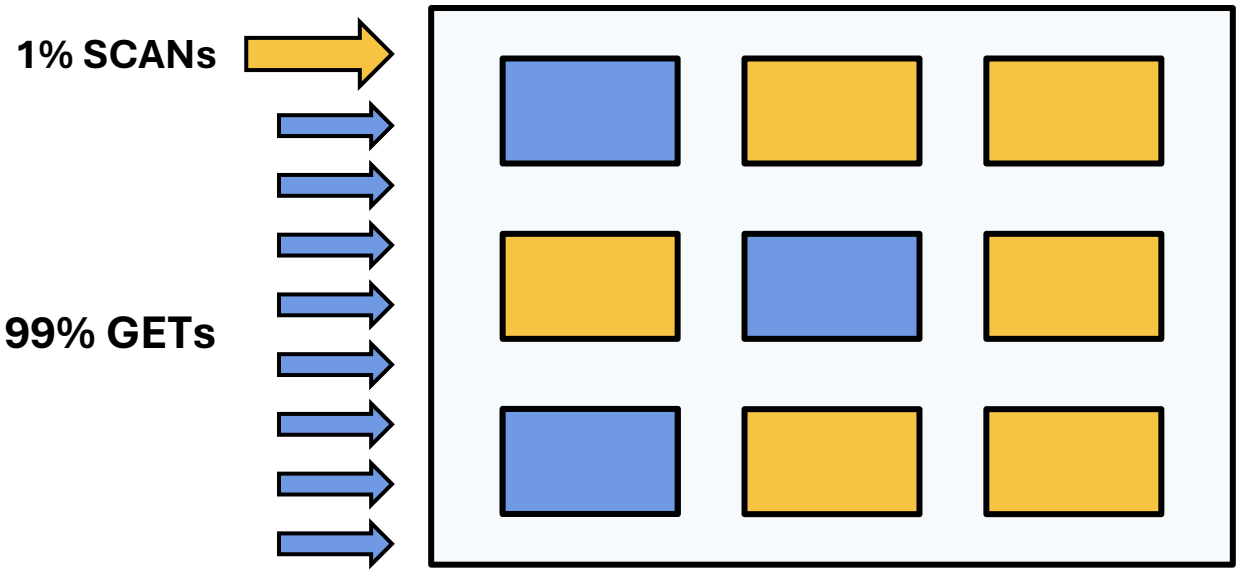
Eviction lists are similar to sched_ext DSQs

Eviction list API
`u64 list_create(struct mem_cgroup *memcg)`
`int list_add(u64 list, struct folio *f, bool tail)`
`int list_move(u64 list, struct folio *f, bool tail)`
`int list_del(struct folio *f)`

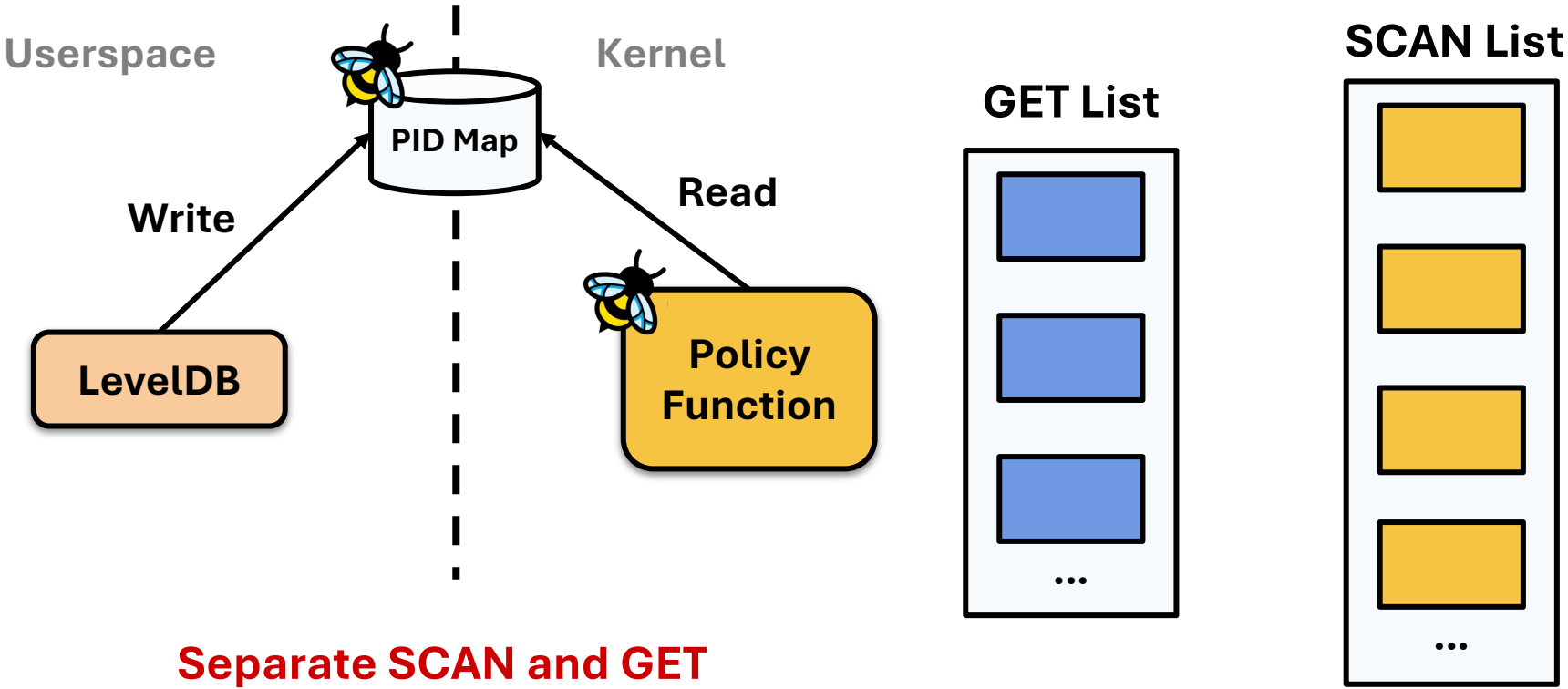
`int list_iterate(struct mem_cgroup *memcg, u64 list,`
`  s64(*iter_fn)(int id, struct folio *f),`
`  struct iter_opts *opts, struct eviction_ctx *ctx)`

Note: These kfuncs are namespaced with a cache_ext prefix

GET/SCAN doesn’t work well with default policy

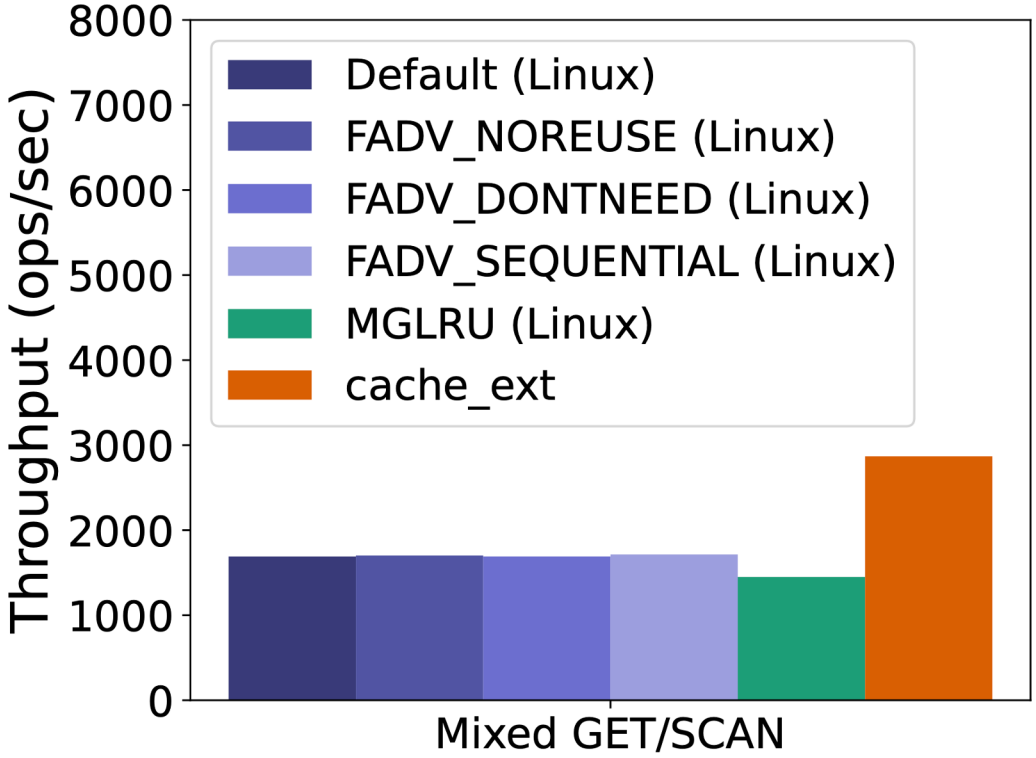


cache_ext enables application-informed policies



Separate SCAN and GET data by request PID

Application-informed policies improve performance



70% GET throughput increase

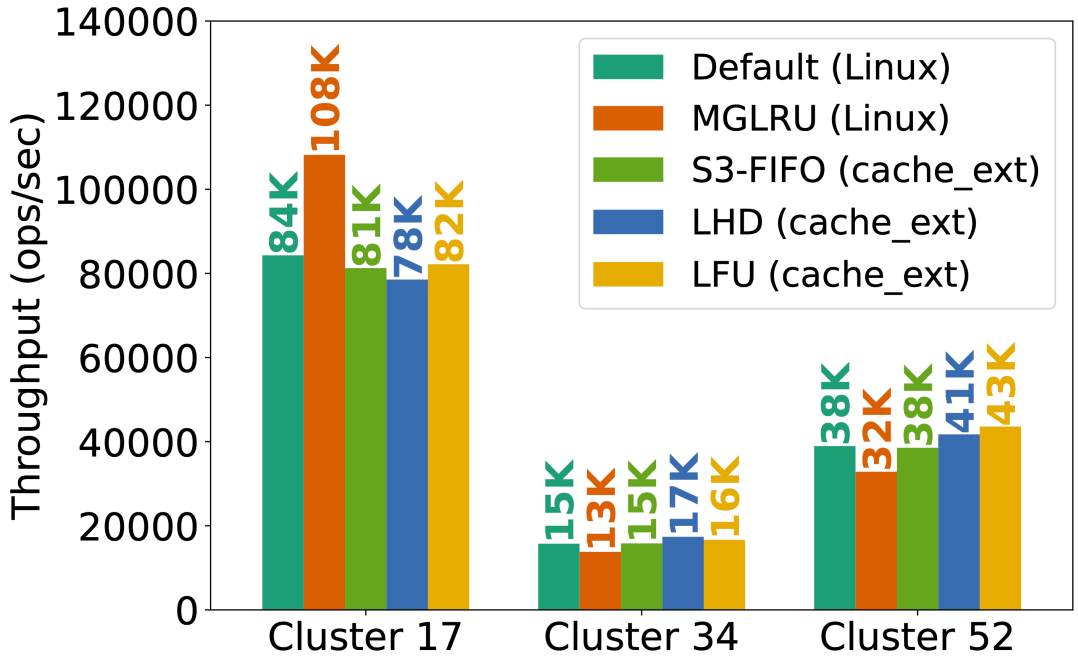
57% tail latency decrease

LFU policy

```
u64 lfu_list;
int lfu_policy_init(struct mem_cgroup *cg) {
    lfu_list = list_create(cg);
    return 0;
}
void lfu_folio_added(struct folio *folio) {
    u64 freq = 1;
    list_add(lfu_list, folio, true); // Add to tail
    bpf_map_update_elem(&freq_map, &folio, &freq);
}
void lfu_folio_accessed(struct folio *folio) {
    u64 *freq = bpf_map_lookup_elem(&freq_map, &folio);
    __sync_fetch_and_add(freq, 1); // Increment freq
}
```

```
long score_lfu(int id, struct folio *folio) {
    return bpf_map_lookup_elem(&freq_map, &folio);
}
void lfu_evict_folios(struct eviction_ctx *ctx, struct
     mem_cgroup *cg) {
    struct iter_opts opts = { /* Set scoring mode */ };
    list_iterate(cg, lfu_list, score_lfu, &opts, ctx);
}
void lfu_folio_removed(struct folio *folio) {
    bpf_map_delete_elem(&freq_map, &folio);
}
```

Customizability yields better performance



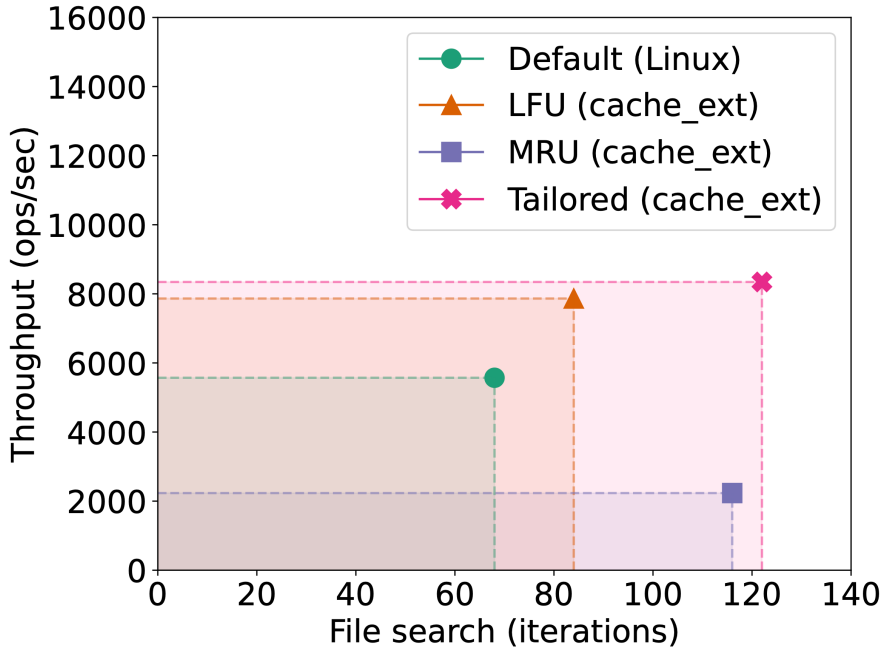
Twitter traces (OSDI '20),
S3-FIFO (SOSP '23),
LHD (NSDI '18)

Also ran with YCSB:
Up to 37% throughput improvement

No one policy performs best in all cases!

Per-cgroup policies

cgroup 1



cgroup 2

Optimal performance when each cgroup runs a policy matching its workload:

50% and 80% improvements over baseline

The case for BPF

- The perfect one-size-fits-all policy is a fantasy
- Tweak behavior for one workload without regressing other workloads
- Need to be able to experiment and specialize (app-informed policies)

Challenges / Open Questions

- Per-cgroup struct_ops
  - Discussed on-list for BPF OOM, sched_ext – solvable
- Use trusted pointers to simplify reference counting
- Live switching policies + cgroup lifetime management
- Is the eviction iteration interface powerful enough? Other data structures? Arenas?

cache_ext enables modern page cache usage

- In order to maximize performance, must be able to experiment
- Policies can be reused across applications
- Applications can have their own per-cgroup policies (multi-tenancy)

Thank you!

- cache_ext enables custom page cache policies to better match applications
- Up to 70% higher throughput and 58% lower tail latency
- More in the paper: Policies, admission filter, security, overhead…

Paper and code:

Tal Zussman
tz2294@columbia.edu

github.com/cache-ext/cache_ext