Safe, Fast Sharing of memcached as a Protected Library

Chris Kjellqvist, Mohammad Hedayati, Michael Scott

ICPP 2020

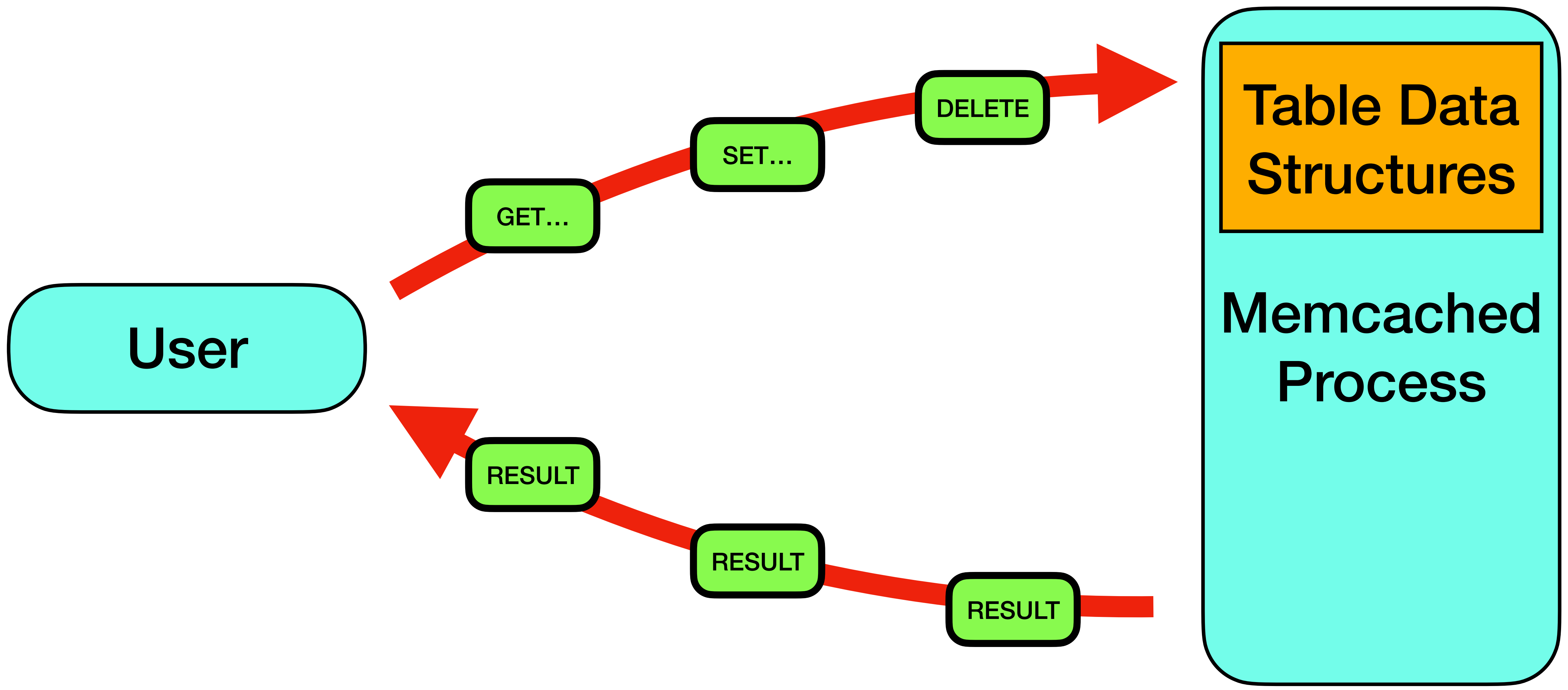
Motivation

- Memcached is a distributed data cache very commonly used by datacenter & e-commerce apps
  - Does this by caching data normally stored to disk and reducing the cost of access
- Memcached operates as an independent process that can be queried from other processes either on or off the current node
  - Queries are made by message passing over sockets
  - Sockets are necessary for communication across nodes, but very wasteful when communicating with a process on the **same** node

Motivation

Sockets

- Applications that require inter-process communication usually do so by message passing interfaces (ie sockets/pipes)
- Separate process performs request for user process. Table data is in a separate address space from the user process
- Sockets are unfortunately wasteful when the data the user desires already exists in memory
  - Require kernel intervention
  - Network protocols severely complicate the code and add the overhead of parsing on top of the already costly server communication



Motivation

- What if we put all Memcached structures necessary to let users perform their own queries in shared memory?
  - Potential 2-3x throughput speedup, 11-56x latency speedup!
  - But… Giving total control to the users is dangerous?
    - Malicious users
    - Even if the developer provides code to correctly perform operations, users don’t need to use it

1. M. Hedayati, S. Gravani, E. Johnson, J. Criswell, M. L. Scott, K. Shen, and M. Marty. Hodor: Intra-process isolation for high-throughput data plane libraries. In 2019 USENIX Annual Technical Conf. (ATC), pages 489–504, Renton, WA, July 2019.

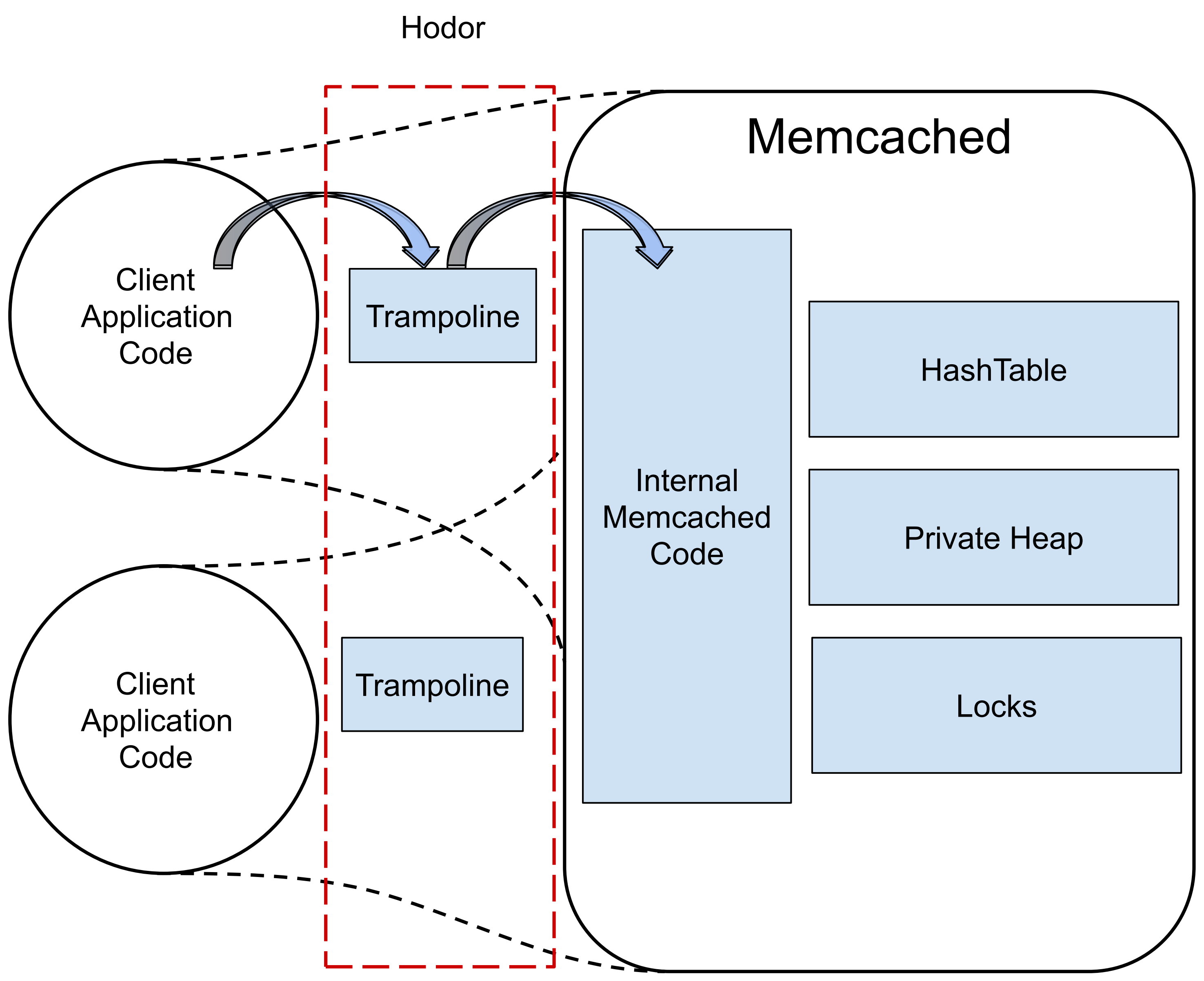
Motivation

- What if we put all Memcached structures necessary to let users perform their own queries in shared memory?
  - Potential 2-3x throughput speedup, 11-56x latency speedup!
  - But… Giving total control to the users is dangerous?
    - Malicious users
    - Even if the developer provides code to correctly perform operations, users don't need to use it
- Hodor - A mechanism for fast, safe process isolation that can be used to replace message passing [1]

1. M. Hedayati, S. Gravani, E. Johnson, J. Criswell, M. L. Scott, K. Shen, and M. Marty. Hodor: Intra-process isolation for high-throughput data plane libraries. In 2019 USENIX Annual Technical Conf. (ATC), pages 489–504, Renton, WA, July 2019.

Motivation

Hodor



- With Hodor, a memcached can allow direct access to its internal data structures by putting them directly in each user's own address space
- Very little extra code
- **Fully secure**
  - Resources are completely inaccessible outside of library calls
  - Guarantees that bounded length library calls are completed even if the process dies
- Uses Intel Protection Keys for Userspace (PKU) to enable fast switching between 'library' and 'non-library' mode
- The code that can access the shared structures is labeled by the programmer as trampolines
- Instead of remotely performing operations by passing them to another process, a user process now performs the operations itself

Motivation

Code Size

- Memcached uses sockets for intra-process and cross-node communication
  - 20% of the code is networking (~5000 LOC)
    - Very complex, allowing multiple formats (binary & ASCII) and multiple protocols (UDP & TCP)
    - Very hard to debug. Where an operation begins and ends is difficult to find

Modifications

- Integrate Hodor
- Make resources available over shared memory
  - Requires code to be position independent
- Make service bulletproof to user error

- These tasks sound expensive but surprisingly require relatively few additions

Modifications

Shared Memory

- Use Ralloc! A position independent, persistent, file-backed slab allocator [ISMM '20]
- Provides smart pointers that simplify the process of position independence
- File backing allows us to easily map the file containing all of our dynamically allocated structures to any user process that wants to use memcached
    - Hodor init routines (where the file will be mapped in) are run as root user, meaning this file is only readable/writable by the protected library and root user
- Ralloc allows protected libraries to have their own private heap! Not only can the line between user and library be confidently drawn, but it provides us other benefits too:
    - Speed, memory management, persistency

Modifications

Hodor Integration

- Mark functions available to the user as ‘trampoline’ functions

```
HODOR_FUNC_ATTR
char *
memcached_get_internal
   (const char * key, size_t key_length, size_t *value_length, uint32_t *flags,
    memcached_return_t *error){
   assert(run_once && "You must run memcached_init before calling memcached_functions");
   *error = MEMCACHED_FAILURE;
   char *buff;
   *error = pku_memcached_get(key, key_length, buff, value_length,
       flags);
   return buff;
}
HODOR_FUNC_EXPORT(memcached_get_internal, 5);
```

Modifications

Hodor Integration

- Write init function(s) that map in file as root and use PKU to protect pages

```
void memcached_init(){
  if (!run_once){
    run_once = true;
  } else return;
// map in file
  is_restart = RP_init("memcached.rpma", 2*MIN_SB_REGION_SIZE);
  int i = 0;
  void *start, *end;
  fetch_ptrs = (item**)RP_malloc(sizeof(item*)*128);
  agnostic_init();
  while (!RP_region_range(i++, &start, &end) && !server_flag){
   ptrdiff_t rp_region_len = (char*)end- (char*)start- 1;
// use PKU syscalls to protect the file
    if(pkey_mprotect(start, rp_region_len, PROT_READ | PROT_WRITE | PROT_EXEC, 1)) {
      printf("error in mprotect: %s\n", strerror(errno));
      exit(0);
    }
  }
// Mark function as an init function. Will be called before main()
} HODOR_INIT_FUNC(memcached_init);
```

Modifications

Hodor Integration

- Write init function(s) that allocate or retrieve structures from previously mapped in file

```
void assoc_init(const int hashtable_init) {
   if (hashtable_init) {
      hashpower = hashtable_init;
   }
// global variable set in previous init function that signals if structures can be fetched or allocated
   if (!is_restart){
// Use pptr<> and Ralloc allocation functions to allocate your structures the same way as malloc
      primary_hashtable_storage = (pptr<pptr<item>>*)RP_malloc(sizeof(pptr<pptr<item> >));
      assert(primary_hashtable_storage != nullptr);
      primary_hashtable = pptr<pptr<item> > ((pptr<item>*)RP_calloc(hashsize(hashpower), sizeof(pptr<item>)));
      assert(primary_hashtable != nullptr);
// Store the new root of the structure statically in the file for easily retrieval in future runs
      RP_set_root(primary_hashtable_storage, RPMRoot::PrimaryHT);
      RP_set_root(nullptr, RPMRoot::OldHT);
      for(unsigned int i = 0; i < hashsize(hashpower); ++i){
         primary_hashtable[i] = pptr<item>(nullptr);
      }
   } else {
// In this case, we have detected a previous run & can therefore retrieve structures directly from the file ready to use
      primary_hashtable_storage = (pptr<pptr<item> >*)RP_get_root<pptr<pptr<item> > >(RPMRoot::PrimaryHT);
      old_hashtable_storage = (pptr<pptr<item> >*)RP_get_root<pptr<pptr<item> > >(RPMRoot::OldHT);
   }
}
```

Modifications

Bulletproofing

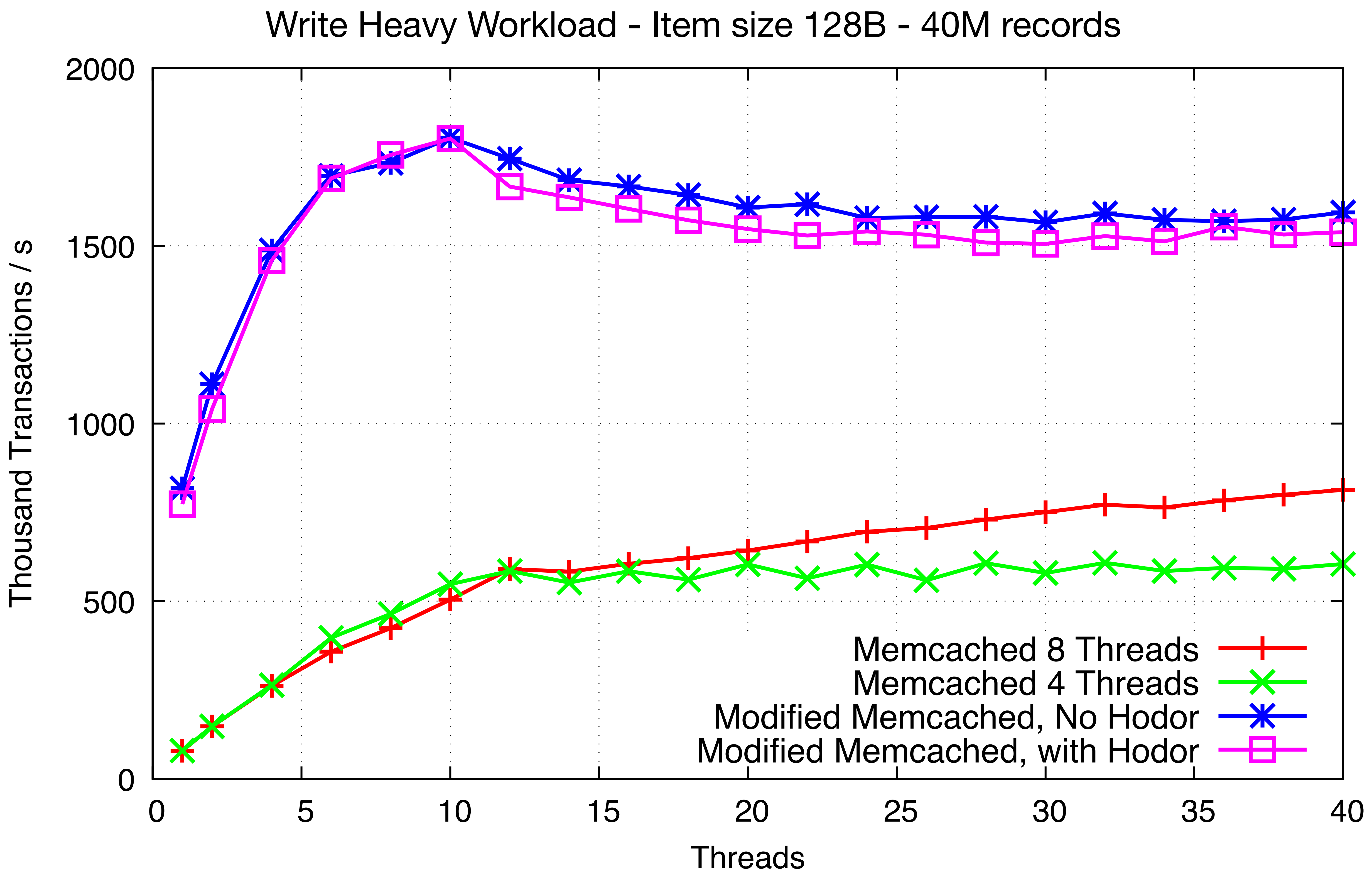
- In the same way the kernel doesn't trust user data, **neither can we!**

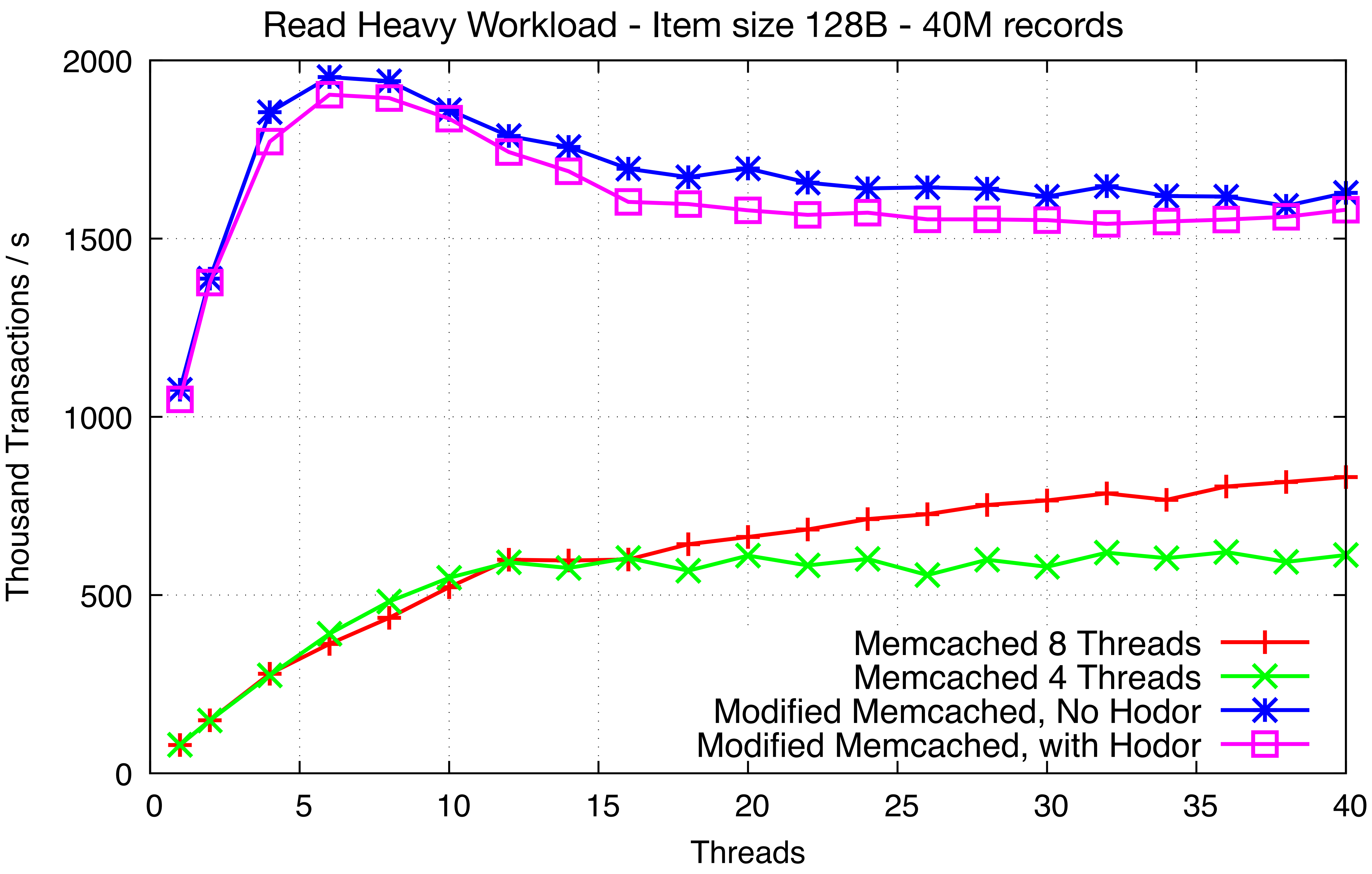
- User data may be nonsensical - Needs to be validated before use
- User threads may change data while it is in use in library - Input must be copied into user-inaccessible buffers
- Cannot trust user locations - All data (even output) must be assembled in user-inaccessible buffers and copied out to user accessible locations after all resources are released

- These changes ensure that errors can not be induced in the protected library by a malicious user

Results

- **2-3x** improved throughput
- Limiting factor is no longer networking, it's the scalability of the data structure
  - This is a much better problem to have because it can be easily* fixed!





Results

- **11-56x** improvement in latency
- Queries that have the lowest cost to actually execute see the largest speedup

	Memcached	Plib, w/ Hodor	Plib, No Hodor	Speedup
Get 128B	13 μs	0.67 μs	0.64 μs	19x
Get 5KB	13 μs	0.67 μs	0.64 μs	20x
Set 128B	13 μs	1.2 μs	1.2 μs	11x
Set 5KB	17 μs	1.5 μs	1.5 μs	11x
Delete	10 μs	0.21 μs	0.18 μs	**56x**
Increment	54 μs	1.6 μs	1.5 μs	36x

Results

- **24%** reduction in code size
  - 5200 lines due to obsolete networking code
  - 1600 lines due to slab management
  + 600 lines added for Hodor integration
  - Integrating Hodor is *much* easier than writing Memcached's socket interface
  - No need for a separate codebase for servers and clients
    - libmemcached is 14 000 lines

Conclusions

- Writing system services as protected libraries gives us
  - Better performance (2-3x throughput)
  - Less code
  - Same degree of safety/security
- Hybrid approach possible
  - Use Hodor for on-node queries and networking for off-node queries
- Future Work
  - Fast Microkernels - moving kernel functionality out into user programs
  - Persistency