



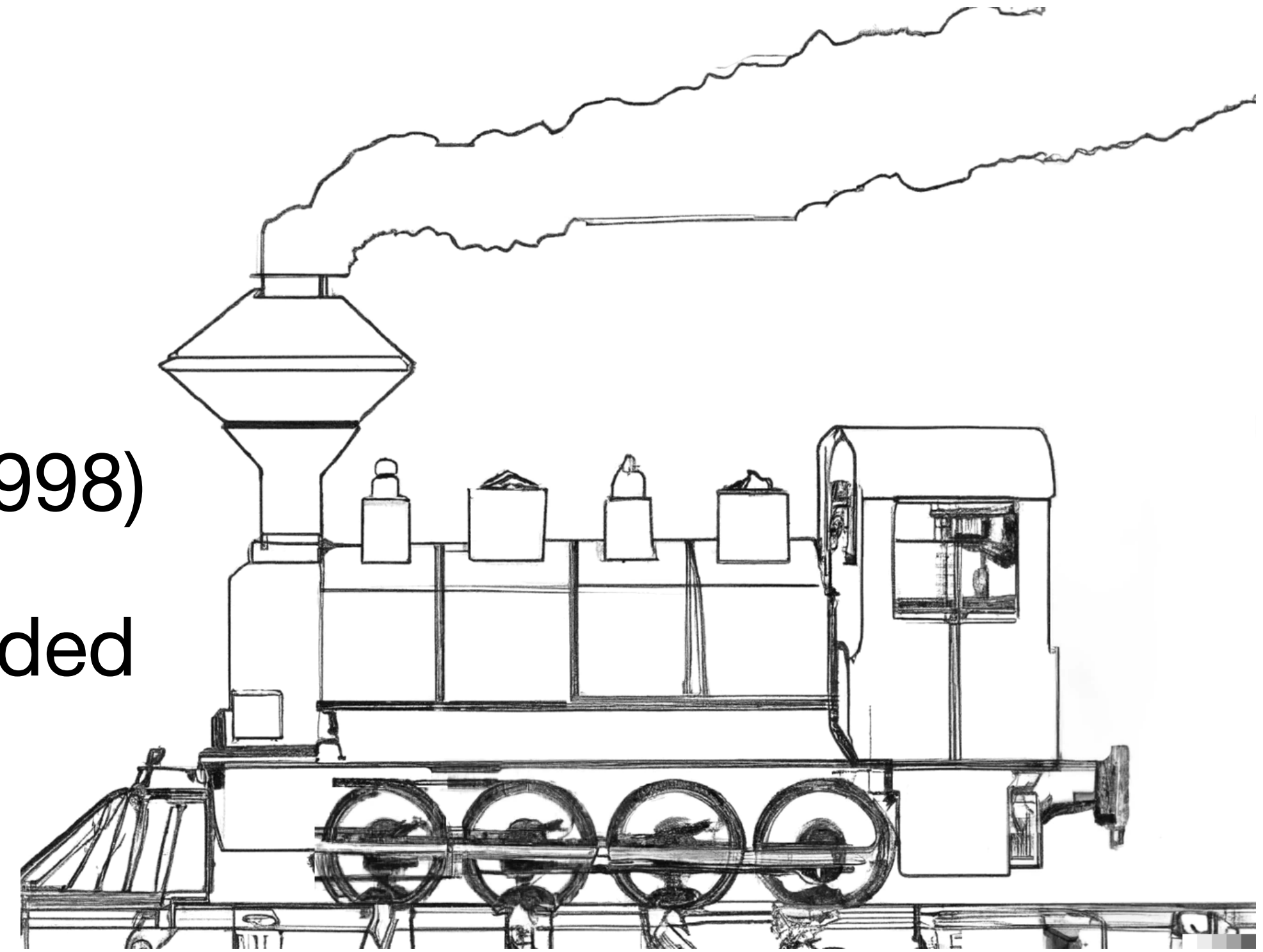
How to make BIND 9 fast(er) and more scaleable

Transition to the new century

Ondřej Surý, ISC; RIPE 90 Lisbon

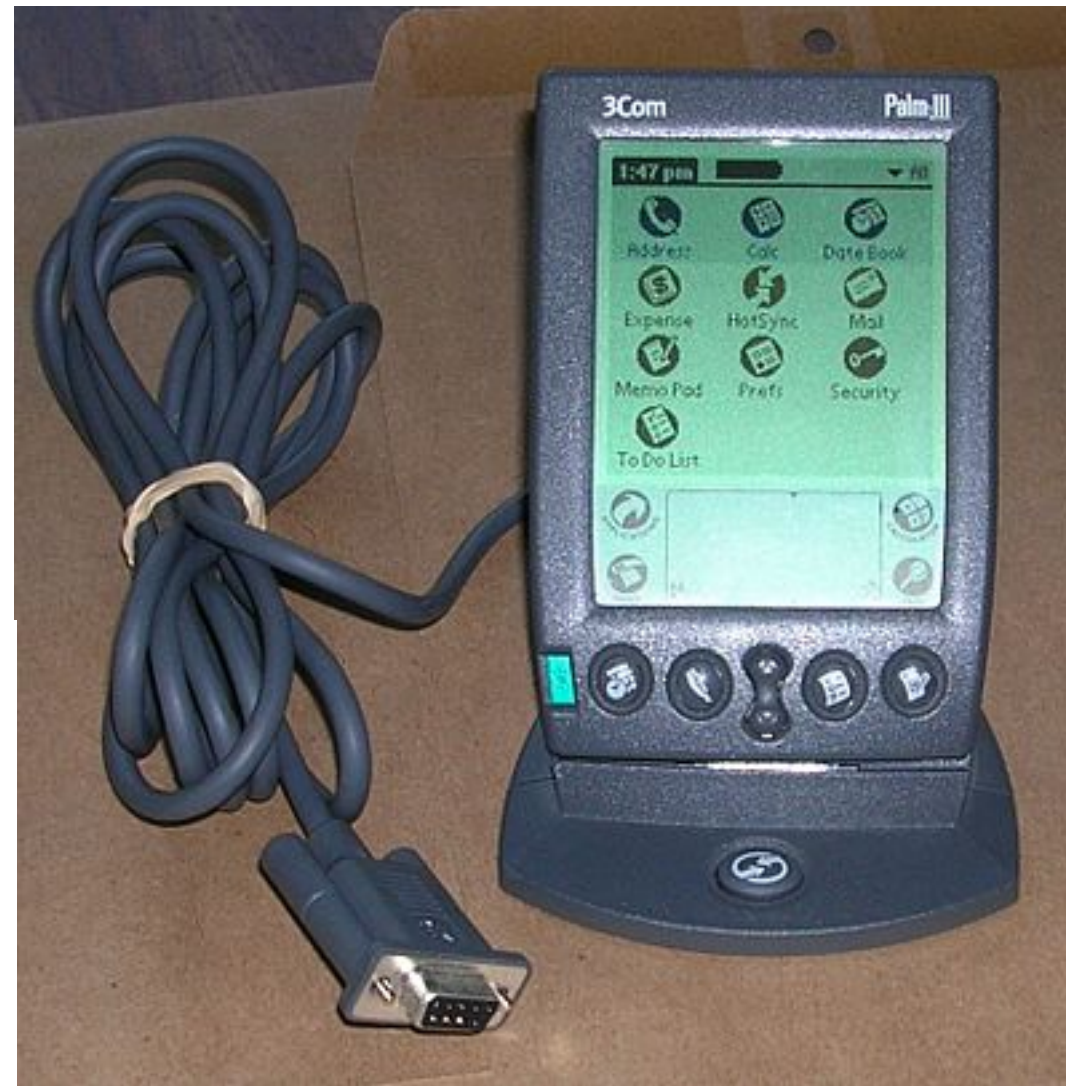
BIND 9 History

- Successor of BIND 4 and BIND 8
- Development started in the last millennium (~1998)
- Originally both single-threaded and multi-threaded
 - Own event-based cooperative scheduling
 - Uses locking (pthread_mutex)
 - Has own rwlock (allows downgrade / upgrade)



1998? How old is that?

AMD
K6
PROCESSOR
2
3DNOW!

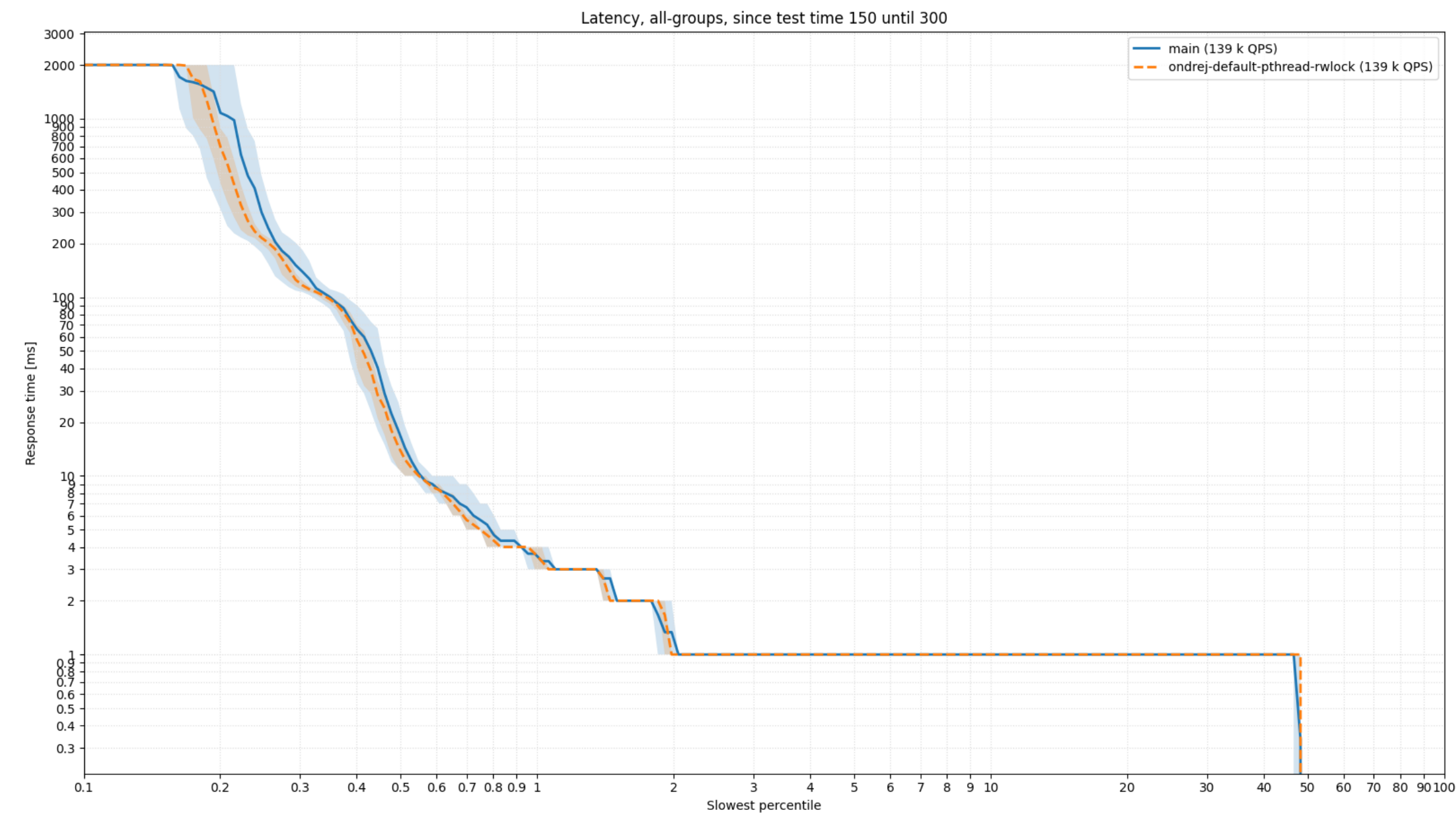
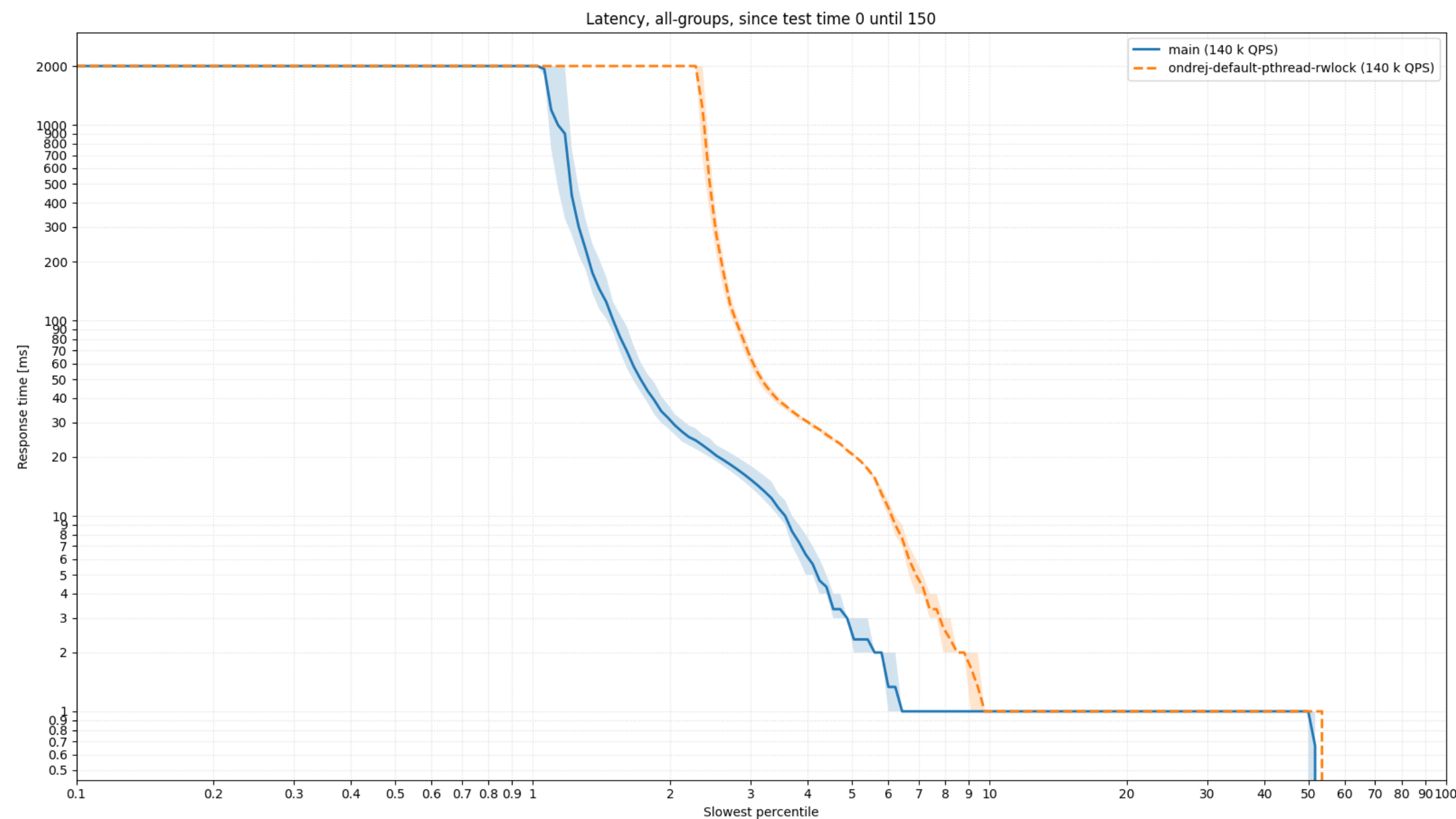


Google

BIND 9.20 Synchronization Primitives

- POSIX Locks
- Custom RW Locking
 - C-RW-WP – <http://dl.acm.org/citation.cfm?id=2442532>
 - Not 1:1 implementation of the paper
- Standard atomics

Custom RWLock – what's the effect?



What Makes Parallel Programming Hard?

McKenney, 2024: Is Parallel Programming Hard, And, If So, What Can You Do About It?



Figure 3.9: CPU Encounters Thermal Throttling

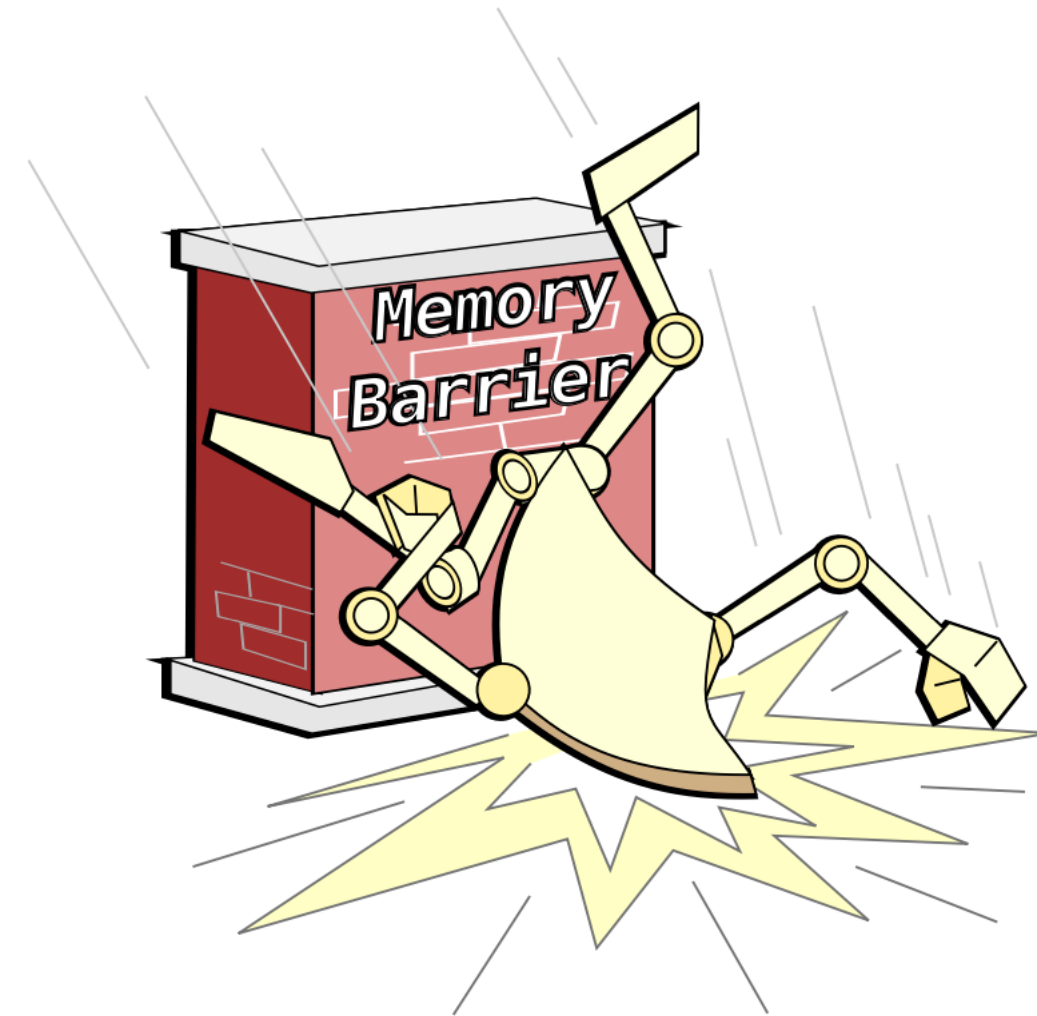


Figure 3.7: CPU Meets a Memory Barrier

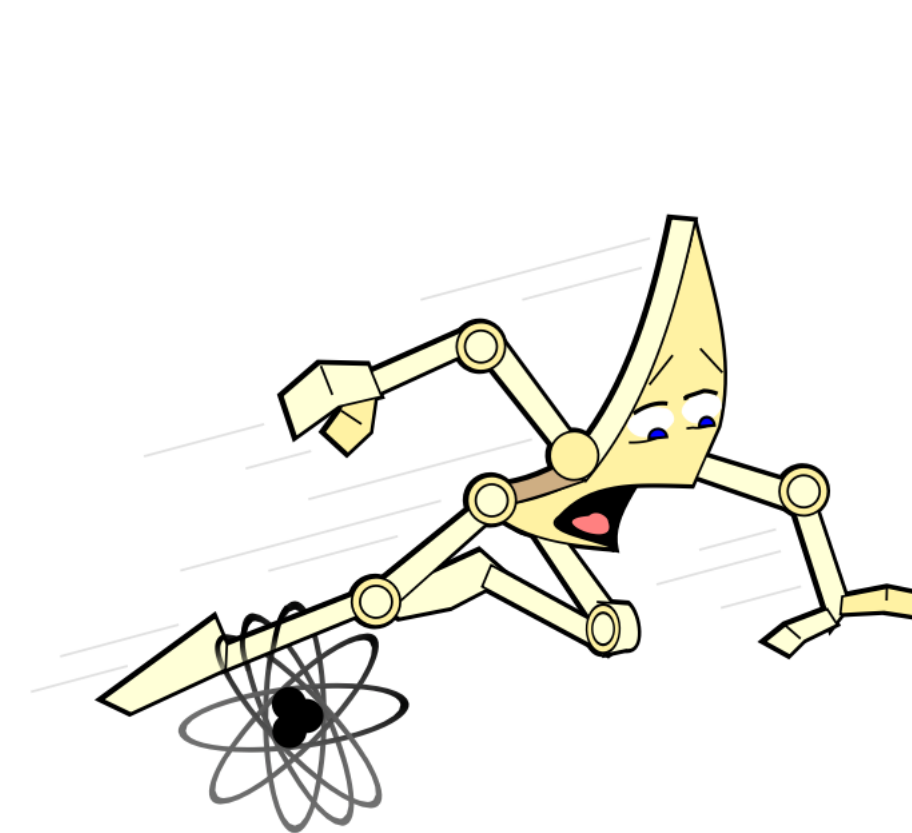
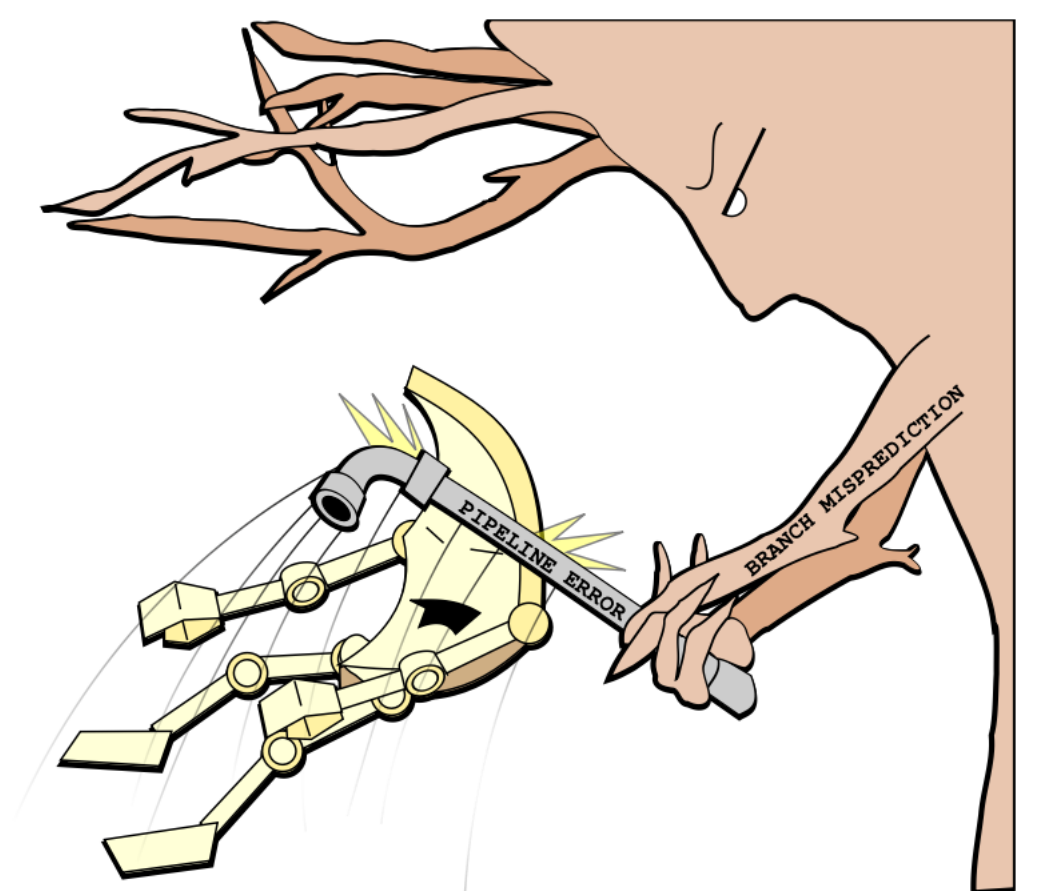


Figure 3.6: CPU Meets an Atomic Operation



CPU Meets a Pipeline Flush

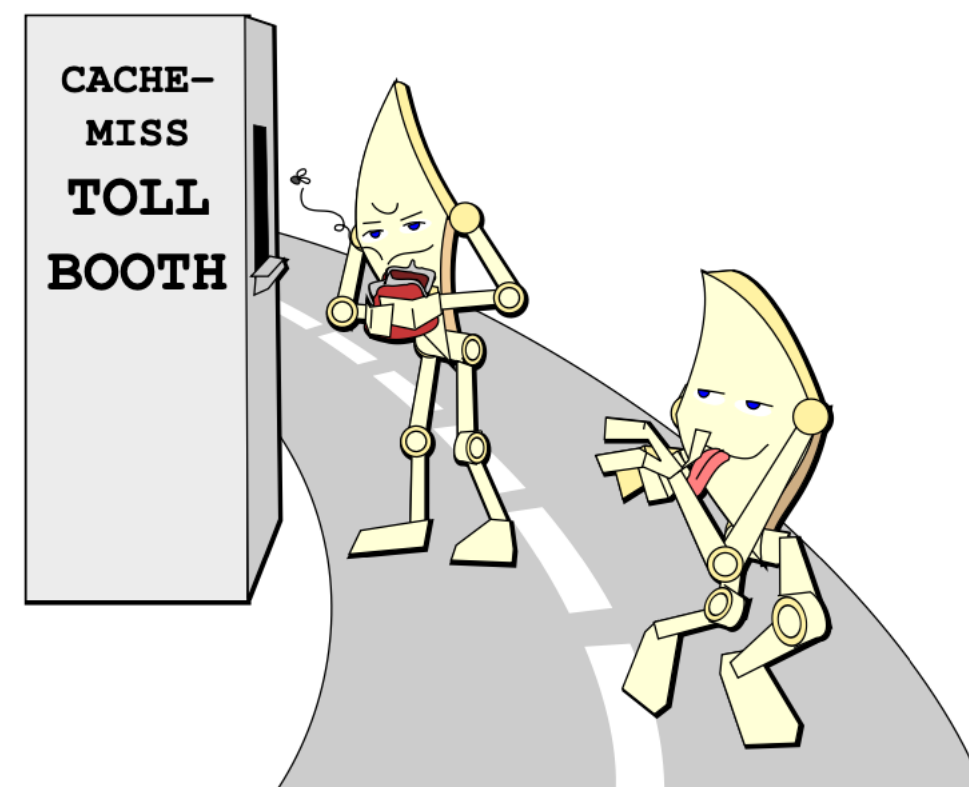


Figure 3.10: CPU Meets a Cache Miss

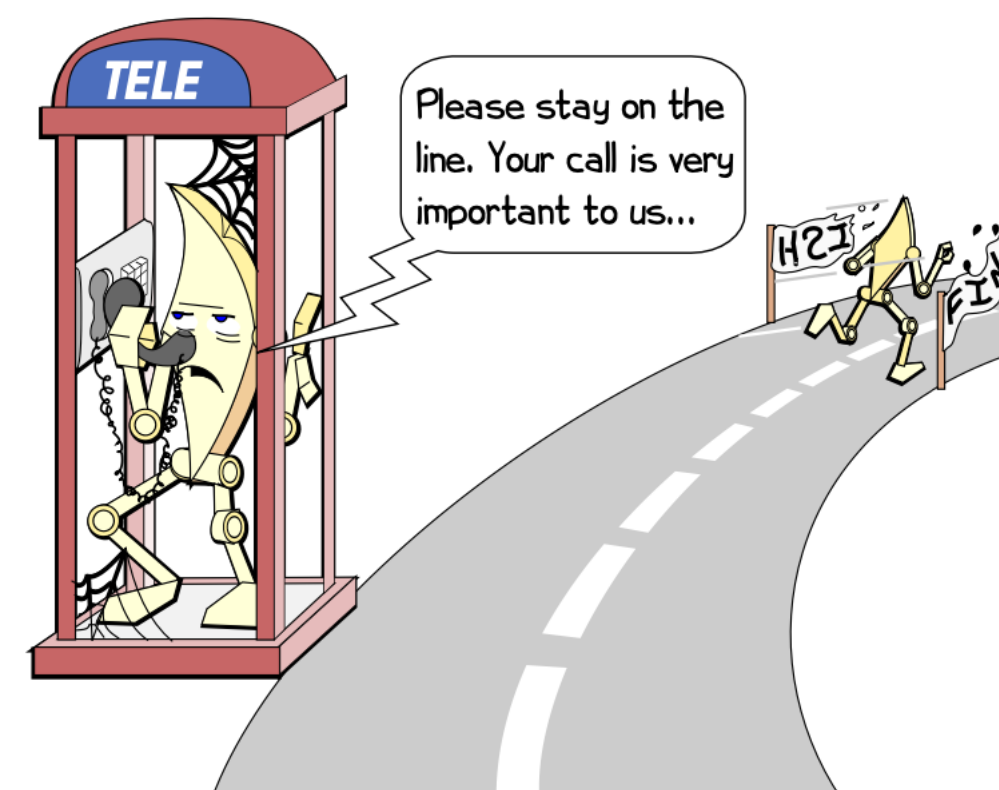


Figure 3.11: CPU Waits for I/O Completion

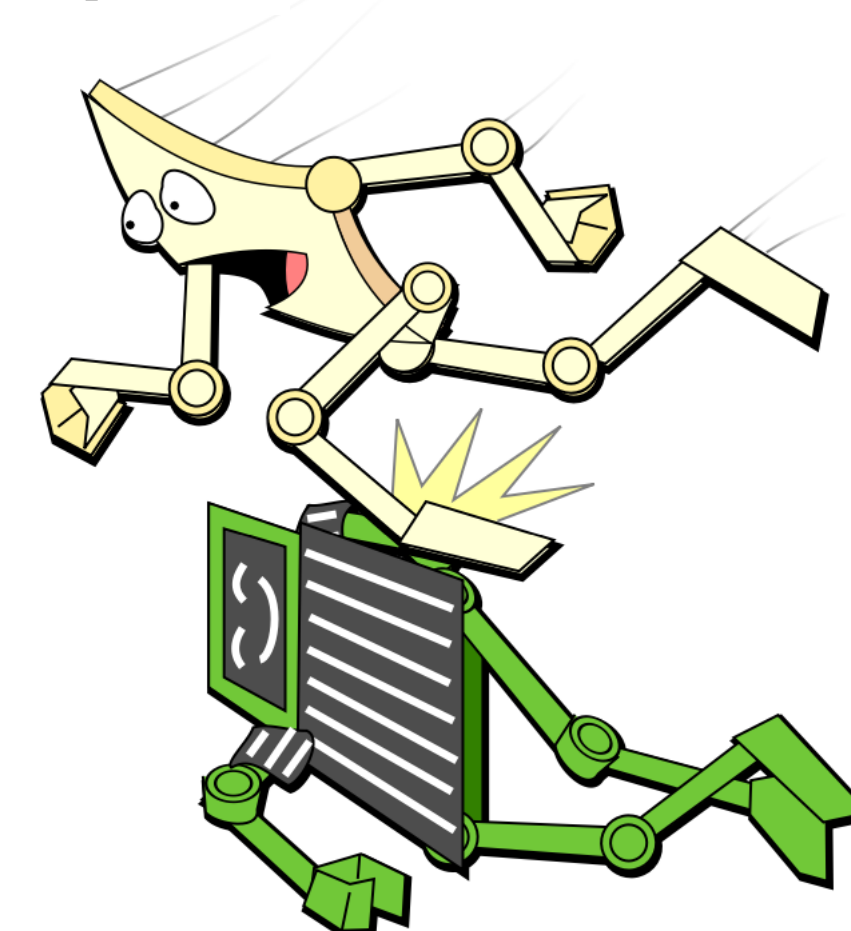
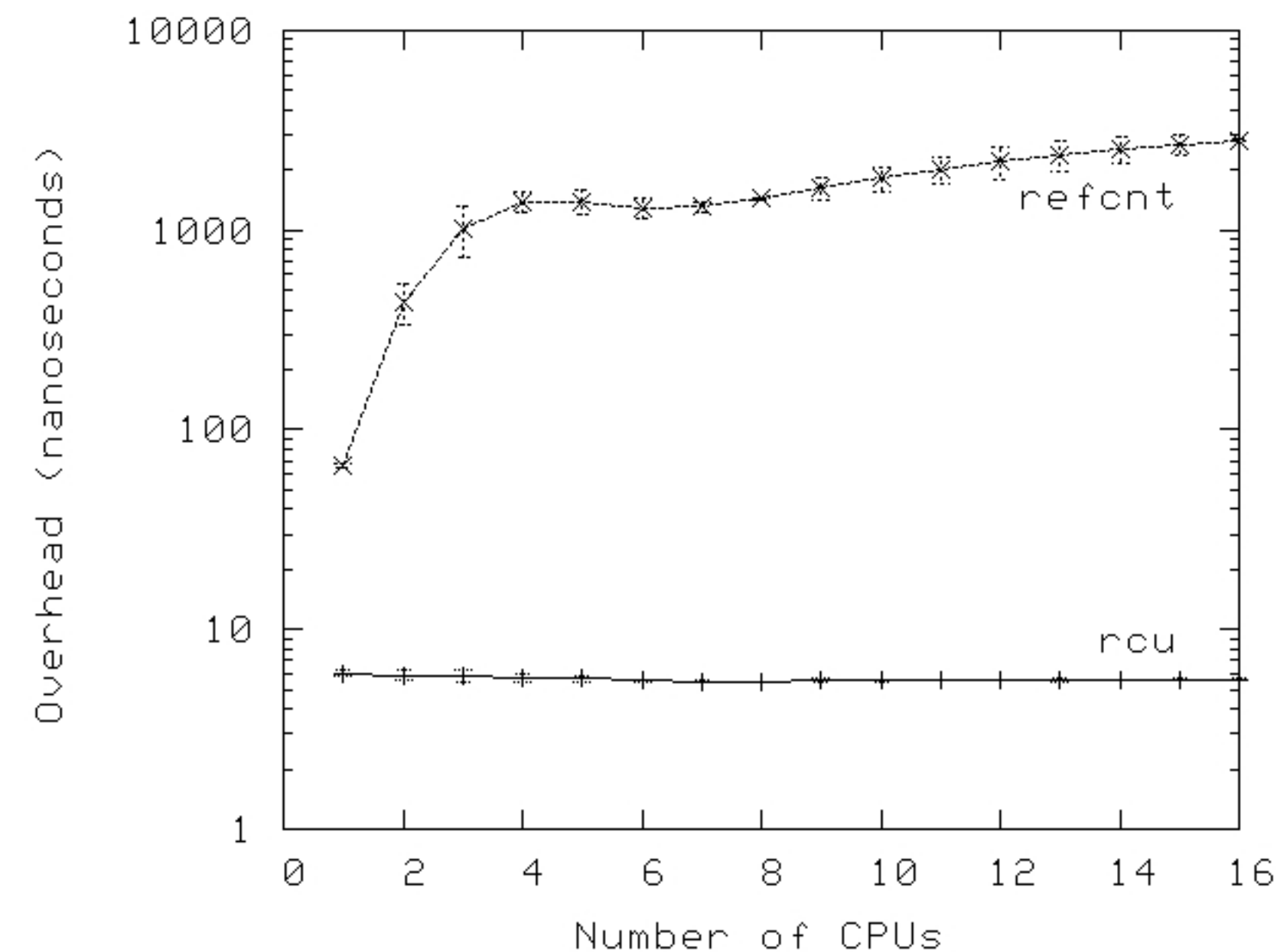
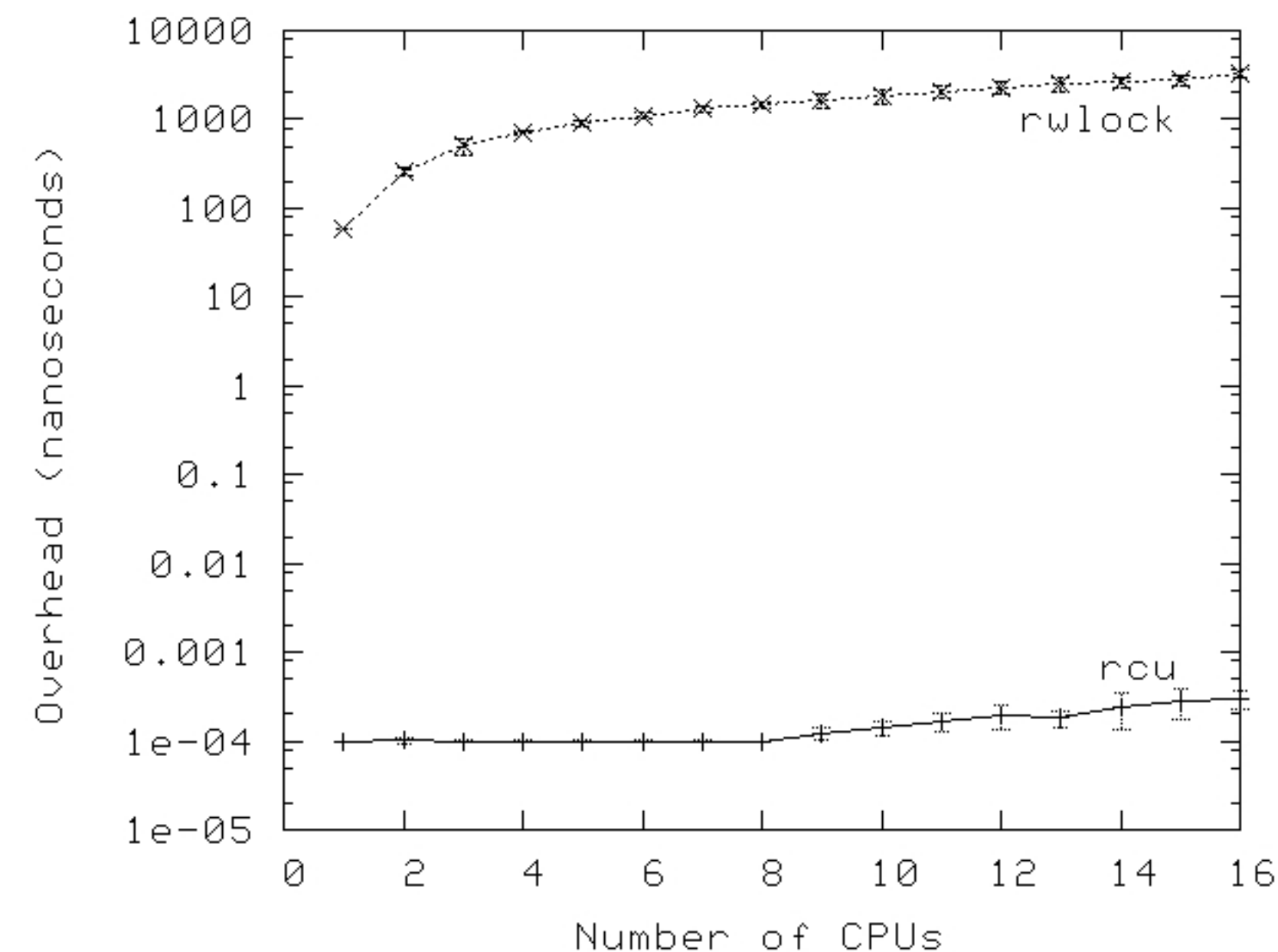
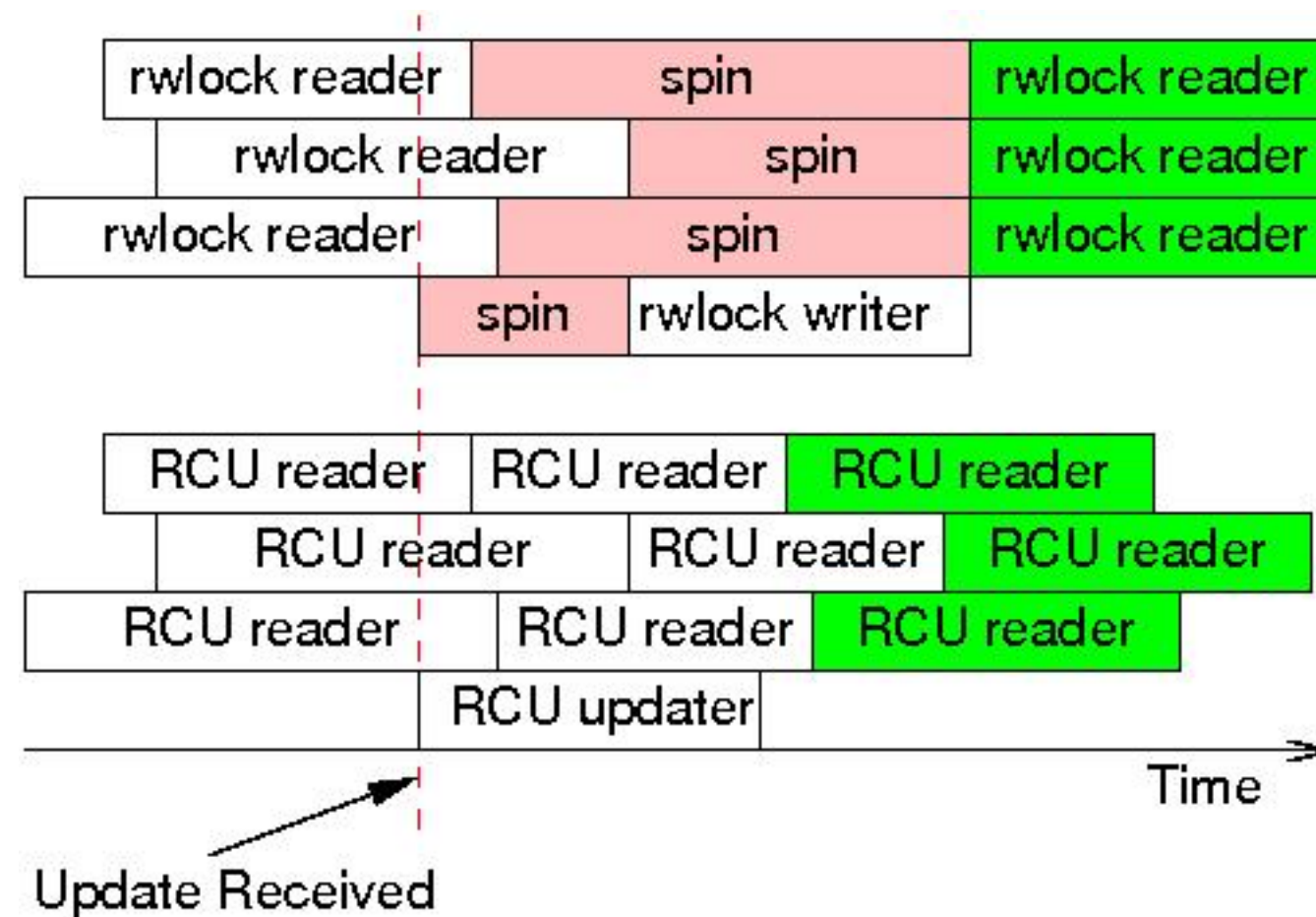
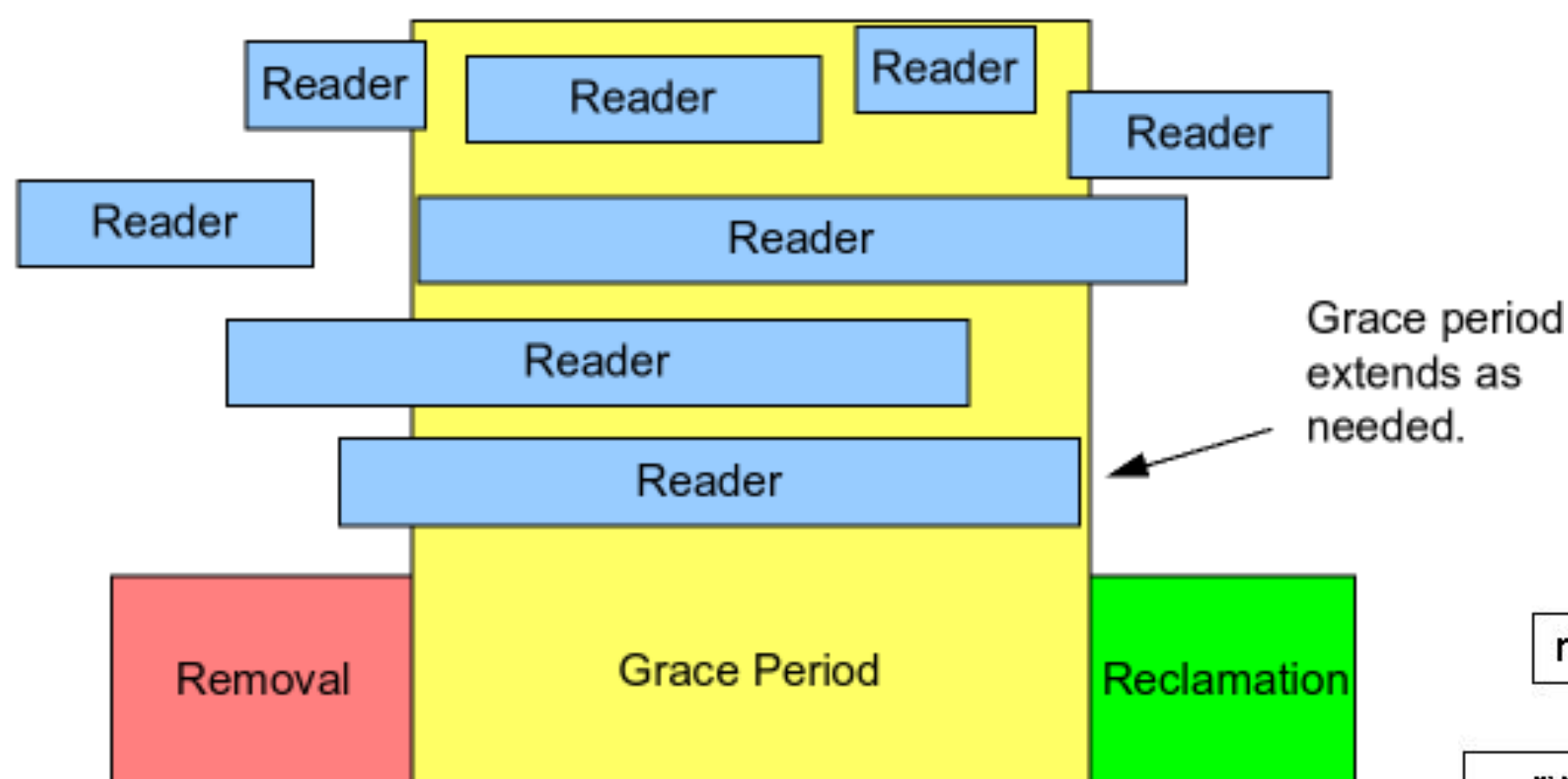


Figure 3.5: CPU Meets a Memory Reference

Read-Copy-Update

See LWN series "What is RCU, Really?"

<https://lwn.net/Articles/263130/>



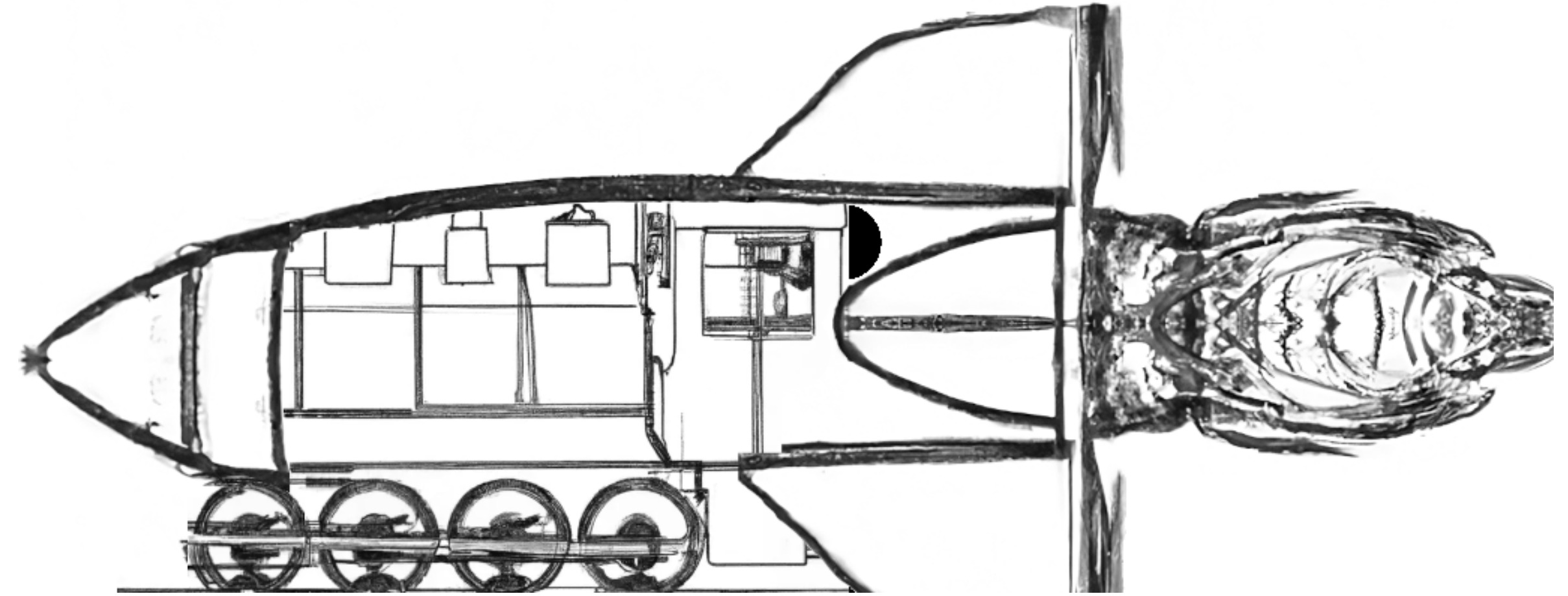
<https://www.linuxfoundation.org/webinars/unraveling-rcu-usage-mysteries>

<https://www.linuxfoundation.org/webinars/unraveling-rcu-usage-mysteries-additional-use-cases>

Userspace-RCU API

- **rcu_**: Read-Copy Update (see [doc/rcu-api.md](#))
- **cmm_**: Concurrent Memory Model
- **caa_**: Concurrent Architecture Abstraction
- **cds_**: Concurrent Data Structures (see [doc/cds-api.md](#))
- **uatomic_**: Userspace Atomic (see [doc/uatomic-api.md](#))

BIND 9.20



- Uses Userspace-RCU
- Replaced locks/rwlocks with URCU in couple places
 - Logging subsystem configuration
 - Weak-references (in couple places)
- Lock-free/wait-free data structures
 - dns_db – callbacks table
 - dns_dispatch – table of TCP connections
 - dns_badcache – temporary status information (indexed with name/type)

dns_badcache API

- Formerly bucketed locked lists (≤ 9.18) – 522 lines
 - Linear lists with dns_name hash as index
 - Lists used as LRU
- Completely lock-free (9.20+) – 435 lines
 - cds_lfht – single hashtable
 - cds_list – per-thread LRU list
 - call_rcu – for memory reclamation

BIND 9.20 event loops

- Replaced the old cooperative scheduling with per-thread libuv loops
- Simpler, but more powerful
- Several types of scheduling:
 - `isc_job` – no-locking, on-loop, very fast, no allocations, shared threadpool
 - `isc_async` – no-locking (`cds_wfcq`), off-loop, fast operations, shared threadpool
 - `isc_helper` – no-locking (`cds_wfcq`), off-loop, slow (crypto) operations, separate threadpool (introduced for KeyTrap)
 - `isc_work` – locking, `uv_work_enqueue`-based, very slow (i/o) operations, uv threadpool

BIND 9.20 K-V Database

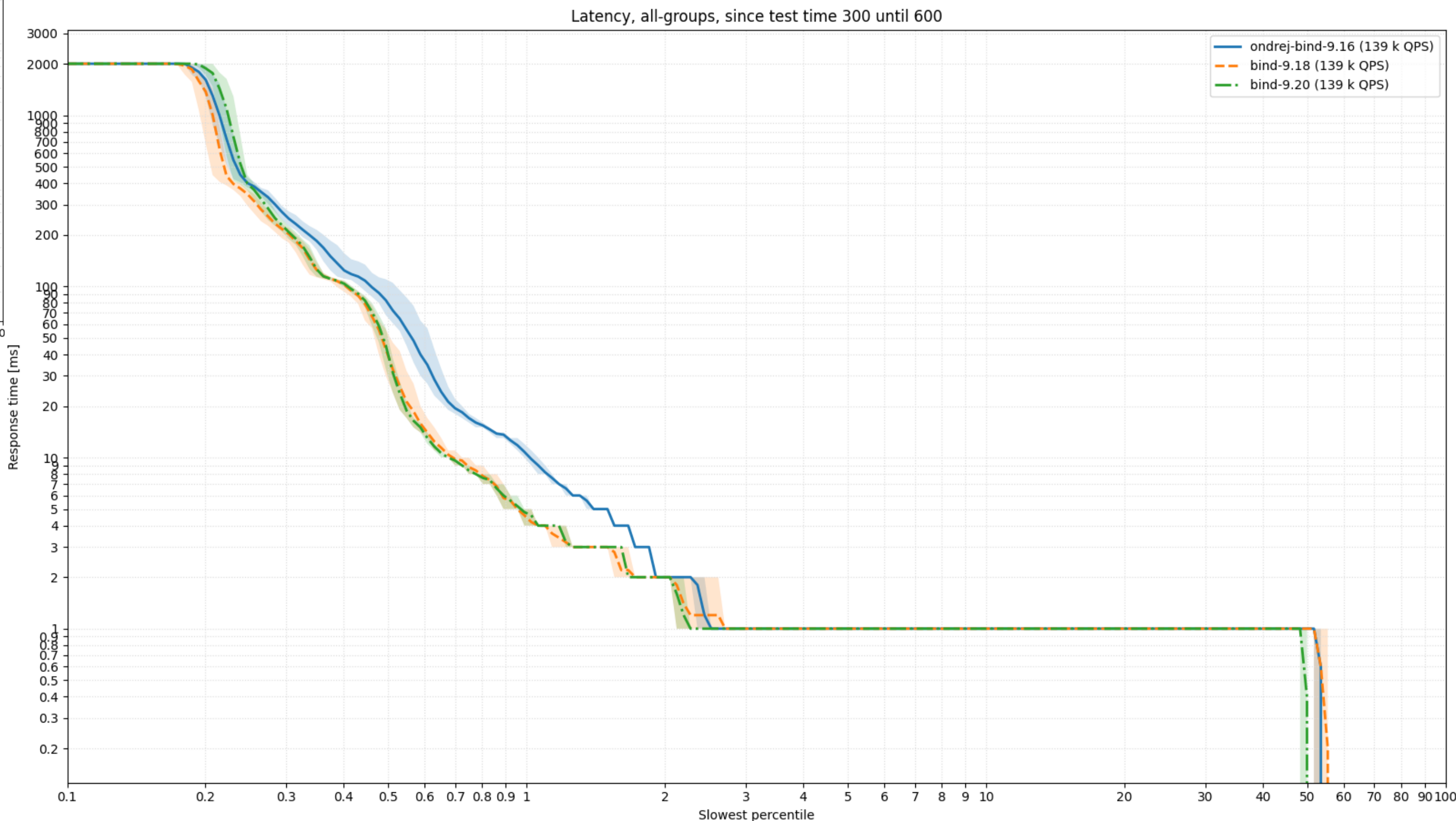
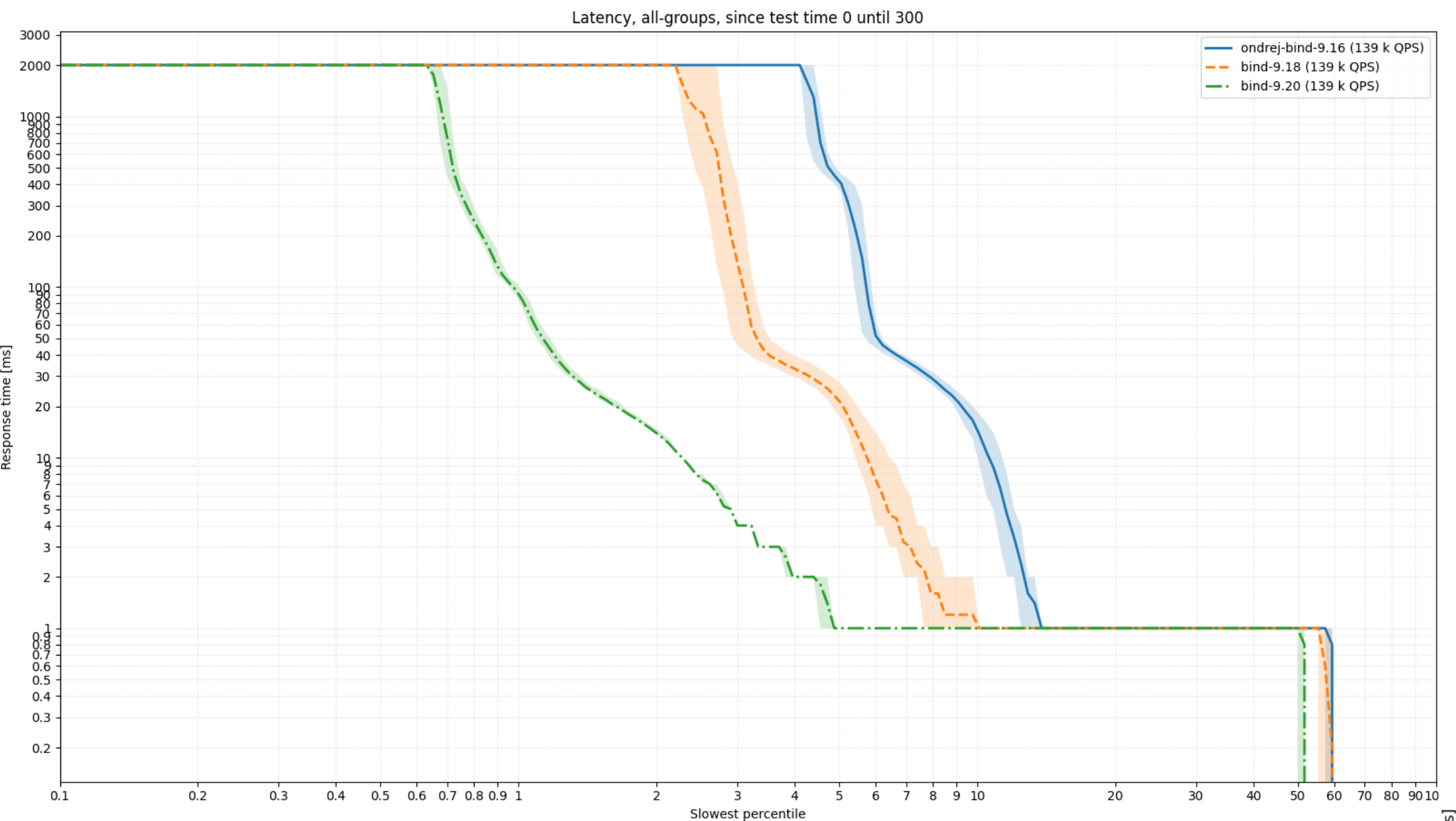
- qp-trie based database
 - <https://dotat.at/prog/qp/README.html>
 - Replaces venerable RBTDB
- qp-trie is a transactional key-value storage suitable for DNS
 - Uses RCU for reads (no locking for reading)
 - Locking for serialization of writes

BIND 9.20 QP Zone and Cache Database

- Built on top of the QP Trie
 - DNS data storage has still the "old" RBTDB design
 - Uses bucketed RWLocks for storing the DNS data
 - Database nodes are indexed using DNS names only
 - All RRTypes are stored inside the node (single-linked list of lists)
 - Types are stored horizontally
 - Versions are stored vertically
 - Data can be freed only when node has no users (references == 0)

BIND 9.20 QP Zone and Cache Database

Includes Networking, Databases and Job Scheduling changes



BIND 9 – measuring hotspots

- DTrace / SystemTap to the rescue
 - Zero-overhead trace points (enabled by in-memory rewrites)
 - Userspace probes (USDT) available in BIND 9.20+
 - RWLocks
 - event loop jobs
 - incoming transfers
 - RRL drops (contributed)
- DTrace-enabled libc available on several systems (Fedora)

SystemTap scripting

```
global mutex_wait_times
global mutex_lock_times

global lock_entry

probe process("/lib64/libc.so.6").provider("libc").mark("mutex_entry") {
    t = tid()
    c = get_cycles()
    lock_entry[t, $arg1] = c
}

probe process("/lib64/libc.so.6").provider("libc").mark("mutex_acquired")
{
    t = tid()
    c = get_cycles()
    fl = usymfileline(ustack(1))
    mutex_wait_times[fl] <<< (c - lock_entry[t, $arg1])
    lock_entry[t, $arg1] = c
}

probe process("/lib64/libc.so.6").provider("libc").mark("mutex_release") {
    t = tid()
    c = get_cycles()
    fl = usymfileline(addr)
    mutex_lock_times[fl] <<< (c - lock_entry[t, $arg1])
    delete lock_entry[t, $arg1]
}
```

- Skip other processes

```
if (pid() != target()) next;
```

- Cache fileline

```
global fileline
```

```
addr = ustack(1)
if (addr in fileline)
    fl = fileline[addr]
else {
    fl = usymfileline(addr)
    fileline[addr] = fl
}
```

SystemTap Output

```
mutex_wait_times["lib/dns/adb.c:1427"] @count=808430 @min=1209 @max=3235167 @sum=4519008117 @avg=5589
mutex_wait_times["lib/dns/adb.c:2268"] @count=808429 @min=1209 @max=1638039 @sum=4234913943 @avg=5238
mutex_wait_times["lib/dns/adb.c:1689"] @count=782902 @min=1248 @max=17523246 @sum=3809706537 @avg=4866
mutex_wait_times["lib/dns/resolver.c:1341"] @count=273231 @min=1209 @max=927108 @sum=1329818100 @avg=4867
mutex_wait_times["lib/dns/adb.c:3345"] @count=171090 @min=1209 @max=8860605 @sum=957802404 @avg=5598
mutex_wait_times["lib/dns/resolver.c:1276"] @count=98490 @min=1248 @max=6739629 @sum=480190620 @avg=4875
mutex_wait_times["lib/dns/resolver.c:2074"] @count=98490 @min=1248 @max=566319 @sum=444133482 @avg=4509

rdlock_wait_times["lib/dns/qpcache.c:2887"] @count=1247530 @min=1170 @max=9171318 @sum=5782837086 @avg=4635
rdlock_wait_times["lib/dns/qpcache.c:1655"] @count=341718 @min=1170 @max=9273810 @sum=2585709750 @avg=7566
rdlock_wait_times["lib/dns/qpcache.c:3591"] @count=326840 @min=1209 @max=4539639 @sum=1916176002 @avg=5862
rdlock_wait_times["lib/dns/qpcache.c:1424"] @count=273618 @min=1170 @max=4258917 @sum=1212226197 @avg=4430
rdlock_wait_times["lib/dns/qpcache.c:2787"] @count=246113 @min=1209 @max=4376346 @sum=1013308296 @avg=4117
rdlock_wait_times["lib/dns/qpcache.c:2824"] @count=246113 @min=1170 @max=12282699 @sum=1751056749 @avg=7114
rdlock_wait_times["lib/dns/qpcache.c:1742"] @count=236144 @min=1170 @max=8419242 @sum=1011850515 @avg=4284

wrlock_wait_times["lib/dns/qpcache.c:3619"] @count=331570 @min=1209 @max=2477631 @sum=1576478475 @avg=4754
wrlock_wait_times["lib/dns/qpcache.c:798"] @count=241068 @min=1209 @max=2768532 @sum=1792732305 @avg=7436
wrlock_wait_times["lib/dns/qpcache.c:2837"] @count=74020 @min=1248 @max=18134376 @sum=1681427982 @avg=22715
wrlock_wait_times["lib/dns/resolver.c:10243"] @count=72705 @min=1248 @max=3041337 @sum=1171274793 @avg=16109
wrlock_wait_times["lib/dns/resolver.c:6928"] @count=72688 @min=1287 @max=1784289 @sum=889544448 @avg=12237
wrlock_wait_times["lib/dns/qpcache.c:3612"] @count=3565 @min=1248 @max=1595841 @sum=75141378 @avg=21077
wrlock_wait_times["lib/dns/qpcache.c:2764"] @count=3053 @min=1326 @max=244608 @sum=35734608 @avg=11704
```

SystemTap Output (Preprocessed)

Combined							
file	type	op	count	min	max	sum	avg
lib/dns/rbt-cachedb.c:1146	rdlock	lock	1224958	8112	4264572	73916873433	60342
lib/dns/rbtddb.c:2247	rdlock	lock	3649970	7371	2861664	45036573309	12338
lib/dns/adb.c:2244	mutex	lock	2343834	7722	7401459	41789003373	17829
lib/dns/adb.c:1346	rdlock	lock	2264501	7644	9186567	30712342050	13562
lib/dns/rbtddb.c:2235	rdlock	wait	4299730	1209	2121132	28323967659	6587
lib/dns/adb.c:2280	mutex	lock	2343834	7371	1382706	21812467521	9306
lib/dns/rbtddb.c:2070	rdlock	lock	461929	24687	3009513	21711909687	47002
lib/dns/resolver.c:10335	rdlock	lock	234511	11778	4396704	19303318515	82313
lib/dns/adb.c:1289	rdlock	wait	2311775	1209	7290894	16339461372	7067
lib/dns/rbt-cachedb.c:1327	rdlock	lock	191438	9282	3035799	12092832141	63168

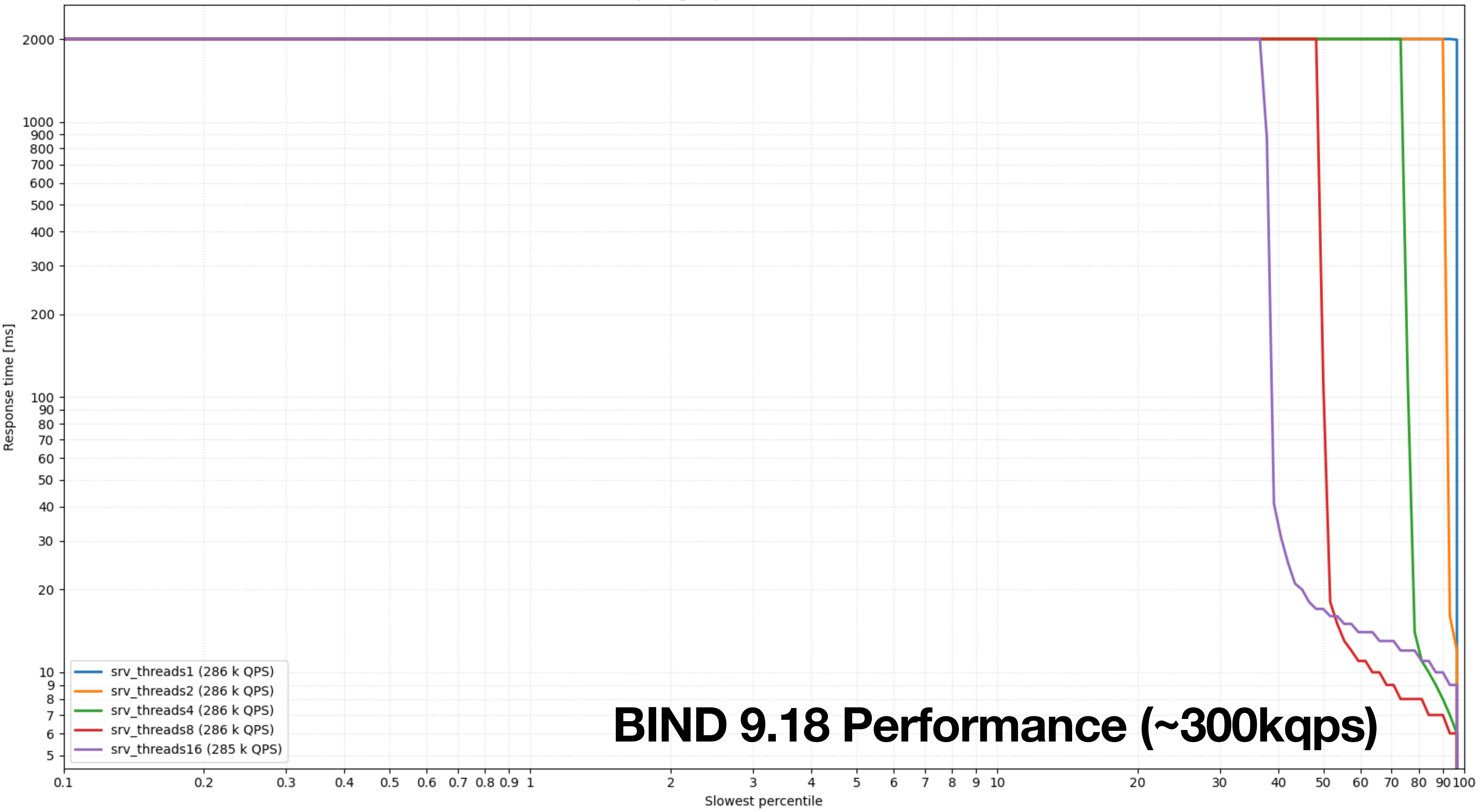
BIND 9.21 and beyond

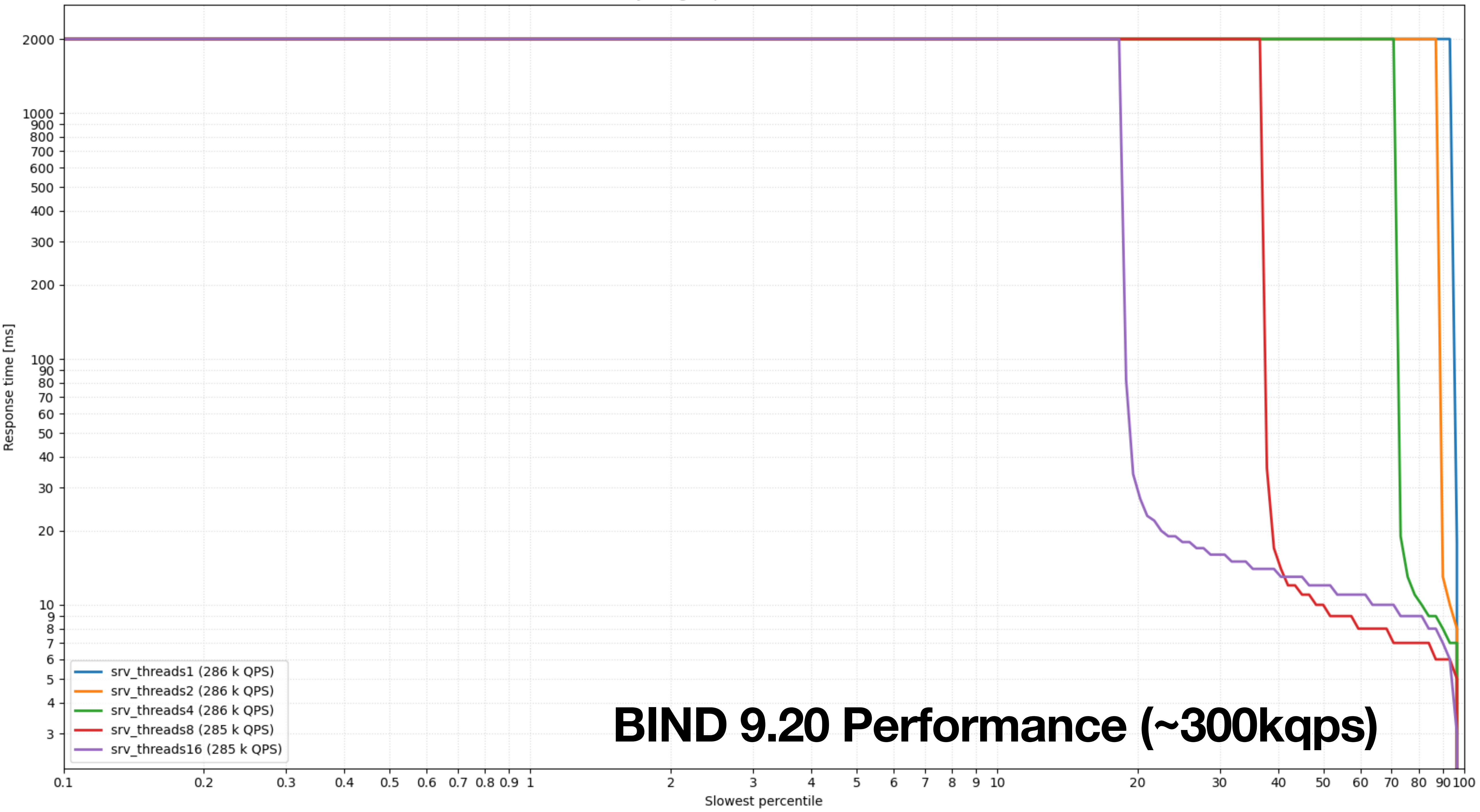
- Keep data local to threads (streamline most operations)
- Share only what needs to be shared among threads
- Convert easy and typical cases to RCU API (locking, refcounting)
 - Not everything can be converted – long-lived objects can't use only RCU
- Replace the configuration synchronization mechanism
 - Centralize the k-v storage and callbacks instead of scattered values
 - Remove the "exclusive" mode (everything needs to stop)

BIND 9.21 and beyond

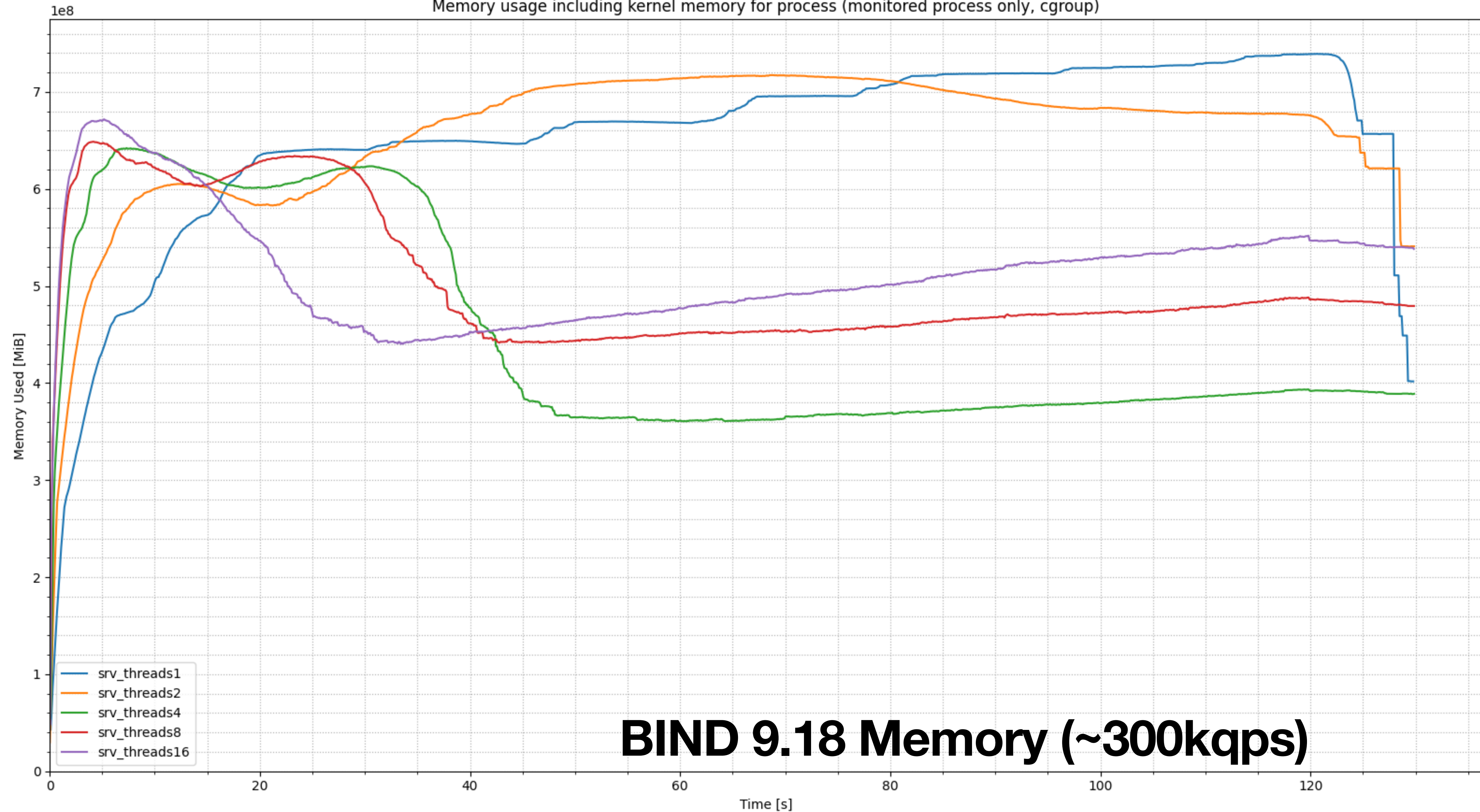
- Remove locking in QPDB (in order of difficulty)
 - Change the API to use name+type as key (in progress, not so easy)
 - Change the writes to use COW (copy-on-write) mechanism
 - Remove/replace locking on the node buckets
 - TTL-based cleaning (use skiplist instead of heaps, in progress)
 - LRU-based cleaning (quite hard, needs per-thread memory for each node)
- Replace the locking in address database (ADB) and resolver
 - Use lock-free hash-tables (easy)
 - Rewrite the LRU mechanism (hard to harder)

Is there some progress already?



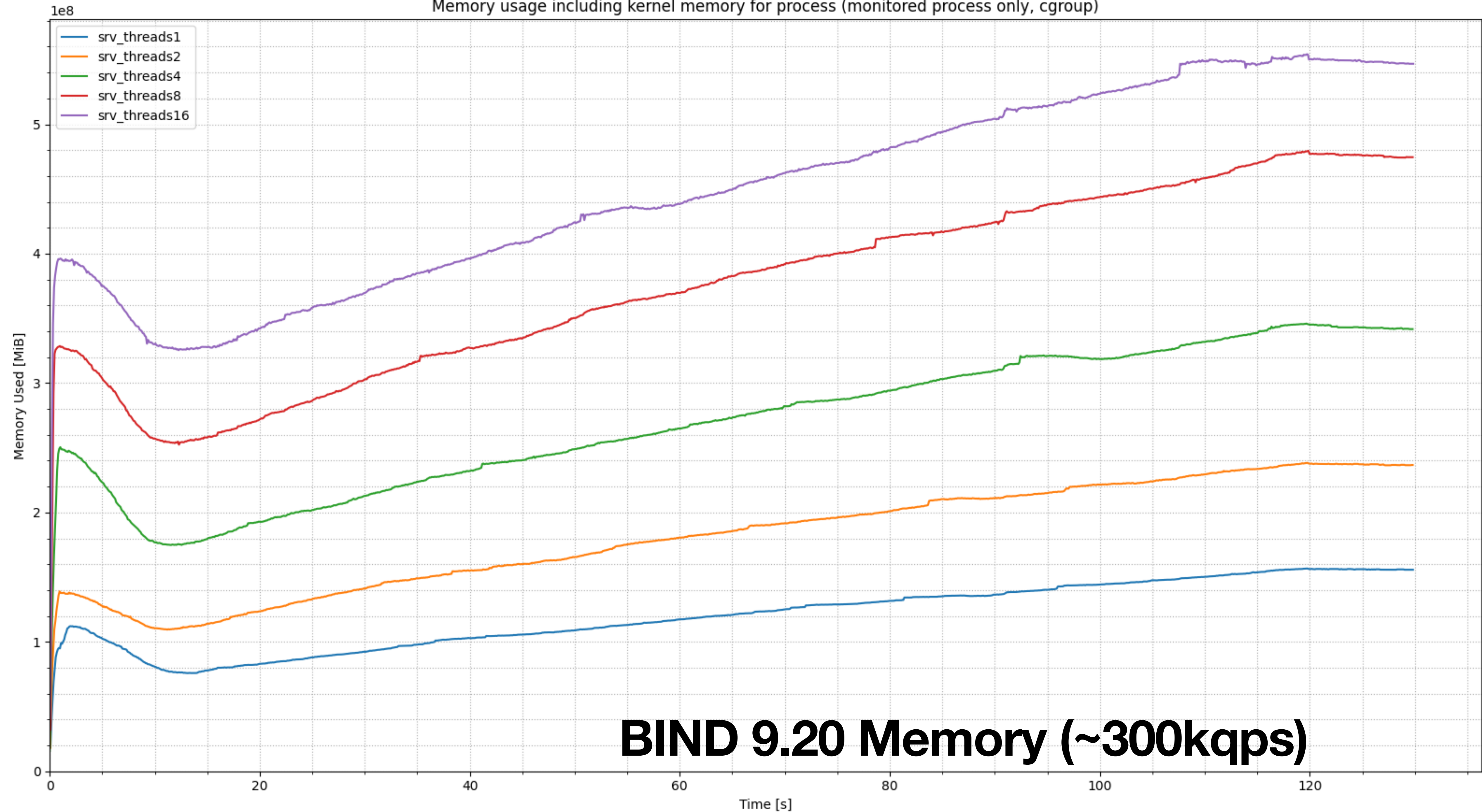


Memory usage including kernel memory for process (monitored process only, cgroup)



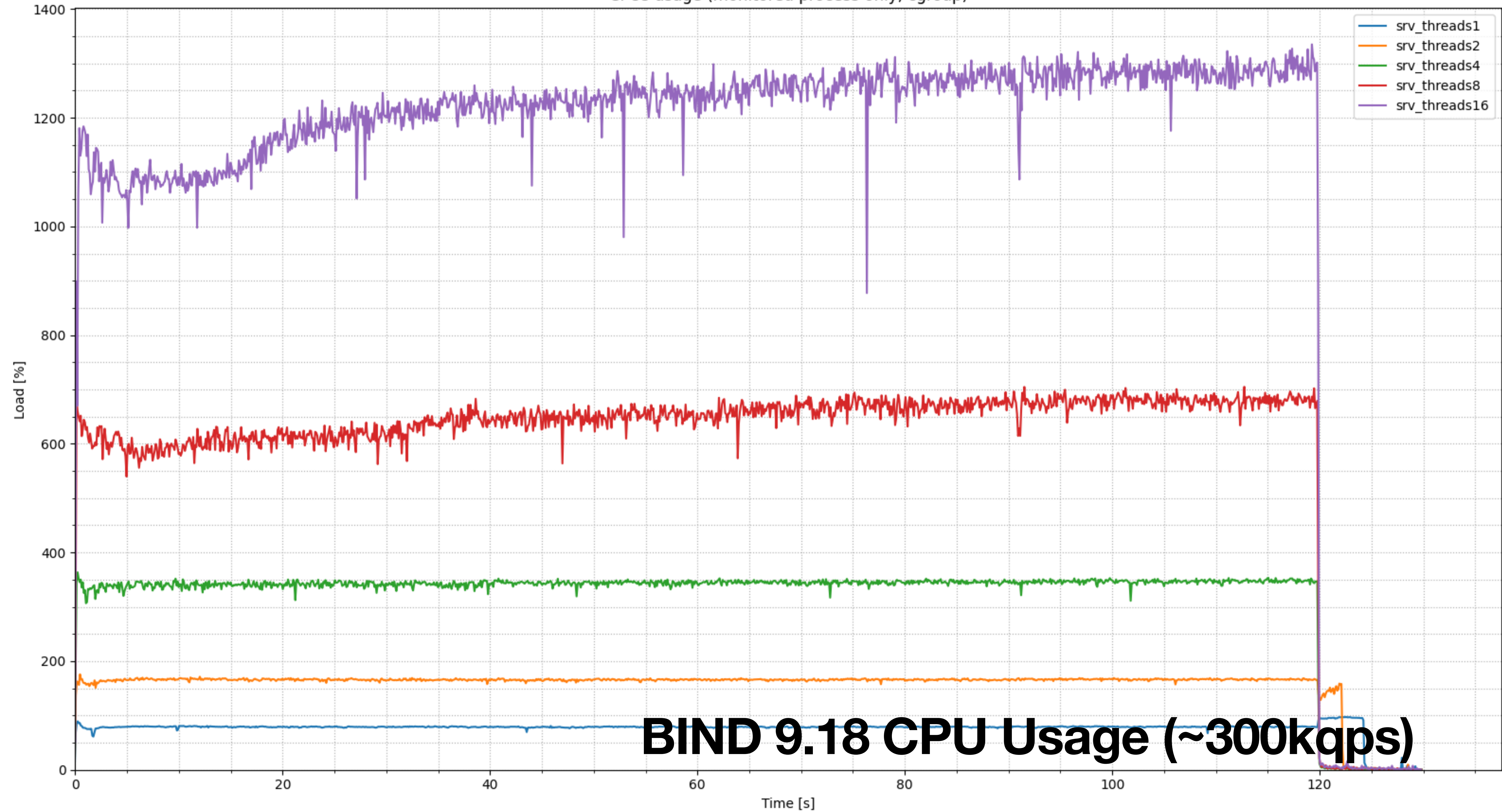
BIND 9.18 Memory (~300kqps)

Memory usage including kernel memory for process (monitored process only, cgroup)



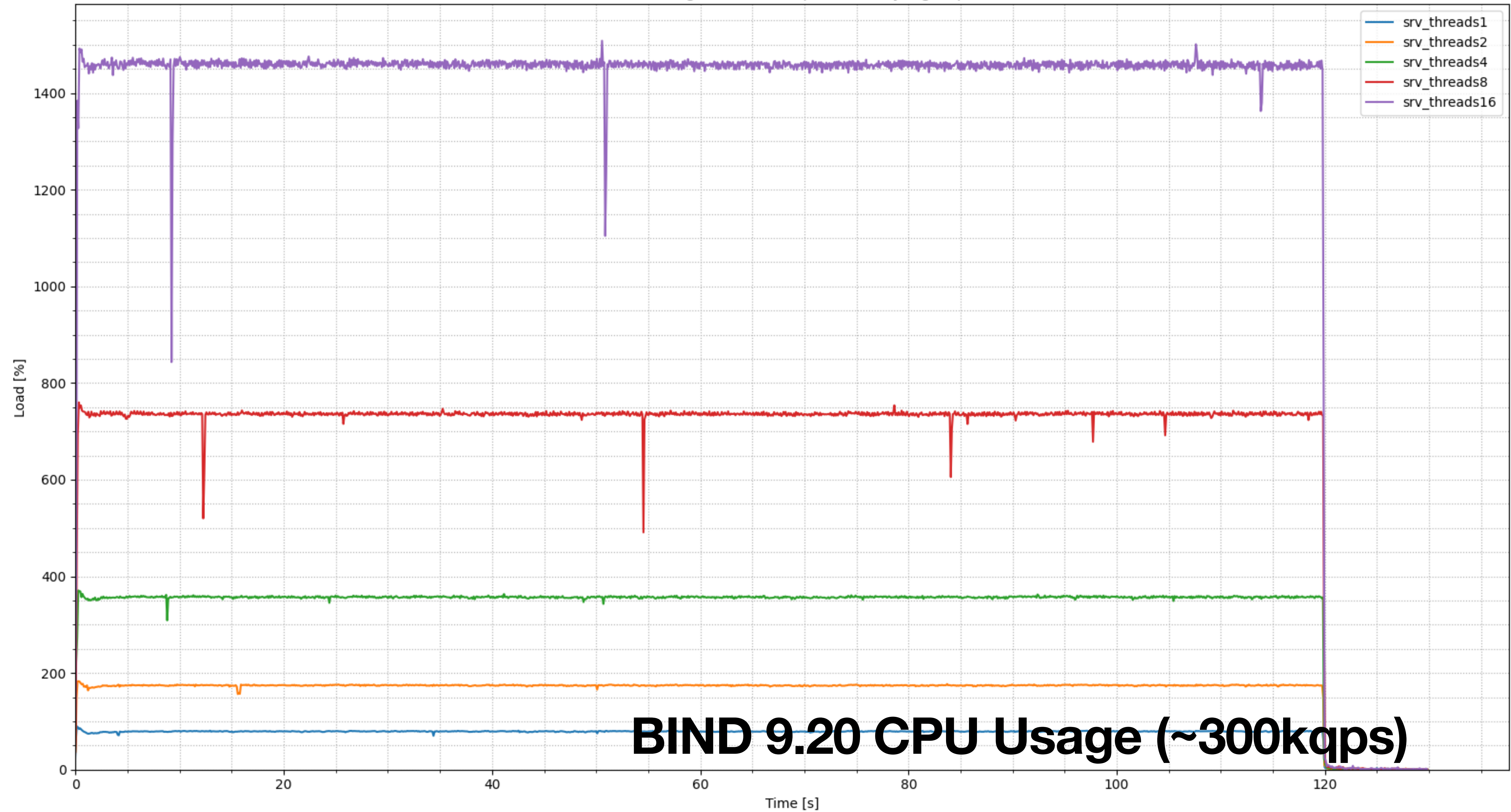
BIND 9.20 Memory (~300kqps)

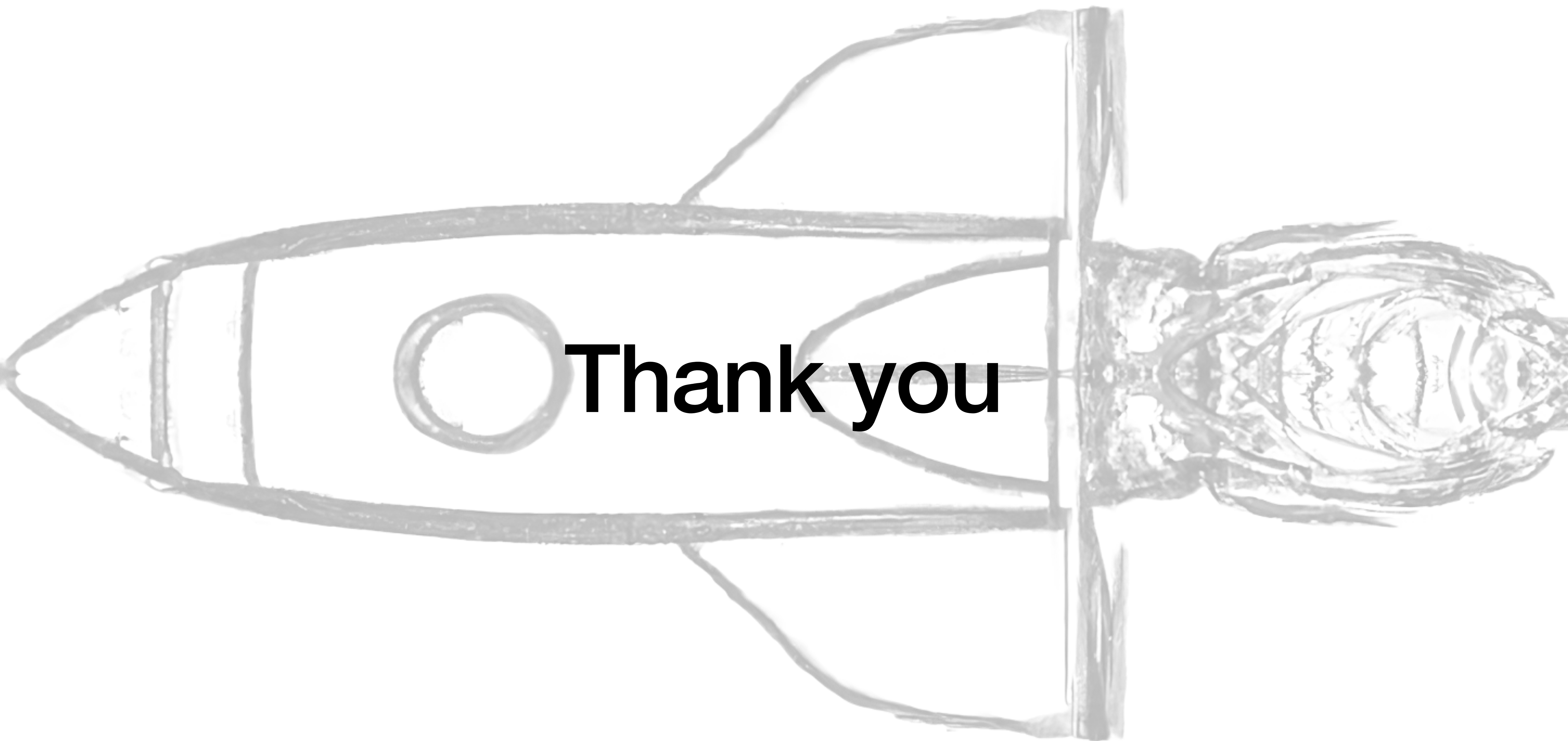
CPUs usage (monitored process only, cgroup)



BIND 9.18 CPU Usage (~300kqps)

CPUs usage (monitored process only, cgroup)





Thank you