Profiling Postgres with Perf
pgconf.eu 2015

Andres Freund
PostgreSQL Developer & Committer
Citus Data – citusdata.com - @citusdata

http://anarazel.de/talks/pgconf-eu-2015-10-29/profilingperf.pdf

citusdata

Profiling

Analyze where a resource, e.g. time, is spent during program execution.

Sampling

- Measure a continuous progress in a discrete way
- Collecting a full “trace” would be too expensive
- Usually low overhead, depends on sampling rate
- Sampling:
  - Every …Seconds (perf's -F option)
  - Every … Events (perf's -c option)

Tracing

- Collect discrete events
- Full tracing of all events too expensive
- Full tracing of all events of a type often also too expensive
- static tracing: predefined event types
- dynamic tracing: new tracepoint at runtime

What's perf

An annoyingly named suite of linux tools

- sampling, tracing recording : `perf record`
- display recorded data: `perf report`
- show live events: `perf top`
- event counting: `perf stat`
- dynamic tracing: `perf probe`
- list events: `perf list`

Setup Perf

- Install perf:
  - debian/ubuntu: `apt-get install linux-tools`
  - Red-Hat based: `yum install perf`
- enable useful profiling for everyone:

```
sudo sysctl kernel.perf_event_paranoid=-1
sudo sysctl kernel.kptr_restrict=0
```

- make it persistent:

```
sudo tee /etc/sysctl.d/60-perf.conf <<EOF
kernel.kptr_restrict=0
kernel.perf_event_paranoid=-1
EOF
```

Prepare Applications

- Install debugging symbols

```
apt-get install libc6-dbg postgresql-9.4-dbg
debuginfo-install postgresql94
```

- Recompile with frame pointers enabled
  - framepointers allow efficient hierarchical profiling

```
./configure CFLAGS='-fno-omit-frame-pointer -ggdb -O2' …
```

  - newer debian/ubuntu packages have it enabled
  - help me lobby devrim to enable it yum.pg.o ;)

Basic Approach

- Choose Event(s) to profile. Default is 'cycles'
- perf record && perf report
- perf top

perf record -a sleep 5
perf report --tui --sort comm,dso,symbol

```
Samples: 38K of event 'cycles', Event count (approx.): 108670457028
Overhead  Command          Shared Object                                Symbol
   3.04%  postgres         postgres                                     [.] hash_search_with_hash_value
   2.60%  postgres         postgres                                     [.] _bt_compare
   2.10%  postgres         postgres                                     [.] AllocSetAlloc
   1.77%  postgres         postgres                                     [.] LWLockAcquire
   1.57%  postgres         postgres                                     [.] GetSnapshotData
   1.53%  postgres         postgres                                     [.] SearchCatCache
   1.09%  pgbench          libc-2.19.so                                 [.] vfprintf
   1.08%  postgres         postgres                                     [.] PostgresMain
   0.95%  postgres         [kernel.kallsyms]                            [k] copy_user_enhanced_fast_str
   0.93%  pgbench          pgbench                                      [.] doCustom
   0.88%  postgres         postgres                                     [.] LockAcquireExtended
   0.80%  postgres         postgres                                     [.] LockReleaseAll
   0.80%  postgres         libc-2.19.so                                 [.] vfprintf
   0.77%  pgbench          [kernel.kallsyms]                            [k] do_select
   0.74%  postgres         libc-2.19.so                                 [.] _int_malloc
   0.70%  postgres         postgres                                     [.] hash_any
For a higher level overview, try: perf report --sort comm,dso
```

perf record -a sleep 5
perf report --tui --sort comm,symbol –no-children

```
Samples: 36K of event 'cycles', Event count (approx.): 95946113235
Overhead  Command          Symbol
   3.53%  postgres         [k] copy_user_enhanced_fast_string
   3.28%  postgres         [.] hash_search_with_hash_value
   1.94%  postgres         [.] AllocSetAlloc
   1.92%  postgres         [.] LWLockAcquire
   1.61%  postgres         [.] _bt_compare
   1.49%  postgres         [.] SearchCatCache
   1.47%  postgres         [.] GetSnapshotData
   0.84%  pgbench          [.] vfprintf
   0.81%  postgres         [.] PostgresMain
   0.79%  pgbench          [.] doCustom
   0.76%  postgres         [.] LockReleaseAll
   0.75%  postgres         [.] vfprintf
   0.74%  postgres         [k] __radix_tree_lookup
For a higher level overview, try: perf report --sort comm,dso
```

perf record –call-graph lbr -a sleep 5
perf report --tui --sort comm,symbol –no-children

```
Samples: 36K of event 'cycles', Event count (approx.): 76786557367
  Overhead  Command          Shared Object                    Symbol
-    3.49%  postgres         [kernel.kallsyms]                [k] copy_user_enhanced_fa
   - copy_user_enhanced_fast_string
      - 96.94% read@plt
           FileRead
        - mdread
           - ReadBuffer_common
              + ReleaseAndReadBuffer
      + 1.95% send@plt
      + 1.04% recv@plt
+    3.18%  postgres         postgres                         [.] hash_search_with_hash
+    1.86%  postgres         postgres                         [.] LWLockAcquire
+    1.80%  postgres         postgres                         [.] AllocSetAlloc
+    1.64%  postgres         postgres                         [.] _bt_compare
For a higher level overview, try: perf report --sort comm,dso
```

perf record –call-graph lbr -a sleep 5
perf report --tui --sort comm,symbol –children

```
Samples: 7K of event 'cycles', Event count (approx.): 23261875019
  Children      Self  Command          Symbol
+   65.40%     0.00%  postgres         [.] __libc_start_main
+   65.40%     0.00%  postgres         [.] main
+   65.40%     0.00%  postgres         [.] PostmasterMain
+   65.40%     0.00%  postgres         [.] ServerLoop
+   63.89%     0.87%  postgres         [.] PostgresMain
+   23.29%     0.25%  postgres         [.] PortalRun
+   23.00%     0.30%  postgres         [.] exec_bind_message
+   22.93%     0.10%  postgres         [.] PortalRunSelect
+   22.49%     0.23%  postgres         [.] standard_ExecutorRun
+   19.87%     0.11%  postgres         [.] ExecProcNode
+   18.96%     0.16%  postgres         [.] ExecScan
+   18.25%     0.12%  postgres         [.] IndexNext
+   18.02%     0.18%  postgres         [.] index_getnext
For a higher level overview, try: perf report --sort comm,dso
```

```
SearchCatCache  /home/andres/build/postgres/dev-optimize/vpath/src/backend/postgres
                                /*
                                 * initialize the search key information
                                 */
                                memcpy(cur_skey, cache->cc_skey, sizeof(cur_skey));
  0.18    28:     lea     0x70(%r13),%rsi
                  mov     $0x24,%ecx
                  lea     -0x150(%rbp),%rdi
 24.96            rep     movsq %ds:(%rsi),%es:(%rdi)
                                cur_skey[3].sk_argument = v4;

                                /*
                                 * find the hash bucket in which to look for the tuple
                                 */
                                hashValue = CatalogCacheComputeHashValue(cache, cache->cc_nkeys,
Press 'h' for help on key bindings
```

```
SearchCatCache  /home/andres/build/postgres/dev-optimize/vpath/src/backend/postgres
                    Assert(IsTransactionState());

                    /*
                     * one-time startup overhead for each cache
                     */
                    if (cache->cc_tupdesc == NULL)
  0.53         cmpq   $0x0,0x28(%rdi)
 11.94       ↓ je     47a
             #endif

                    /*
                     * initialize the search key information
                     */
                    memcpy(cur_skey, cache->cc_skey, sizeof(cur_skey));
Press 'h' for help on key bindings
```

Looking at one Bottleneck

```
SearchCatCache(CatCache *cache, Datum v1, ..., Datum v4)
{
    ScanKeyData cur_skey[CATCACHE_MAXKEYS]; -- 288 bytes

...
    memcpy(cur_skey, cache->cc_skey, sizeof(cur_skey));
...
    switch (cache->cc_nkeys)
    {
...
        case 4:
            oneHash = DatumGetUInt32(DirectFunctionCall1(...,cur_skey[3].sk_argument));
```

Call Graph Profiling

- Sample Stack for Events
- Different methods
  - fp: efficient, default, requires compilation flag
  - lbr: efficient, requires new hardware, only hardware events, no tracepoints
  - dwarf: slow, large data, works always, requires debuginfo
- Use lbr if you can, fp otherwise, fall back to dwarf

What to record

- Everything (till ctrl-c): perf record -a
- Everyting for a while: perf record -a sleep 5
- A command: perf record somecommand
- Important options:
  - -a – systemwide profiling
  - -g / --call-graph $method – include stack in samples
  - -e event-desc1 – what event(s) to measure
  - -F # – sampling frequency
  - -f $file – store output in $file

What to show

- perf report options:
  - --children – include cost of children in sorting
  - --no-children – do not include cost of called functions
  - --sort comm,dso,symbol,… – fields to “group by”
  - --stdio // --tui // --gtk – frontend

Events

- perf list (depends on user permissions!)
- perf help list – syntax for event descriptors
- Important Hardware Events:
  - cycles (both hard & software)
  - cache-misses
  - branch-misses
  - modifiers: pp (precise), u/k (user/kernel)
- Important OS Events
  - page-faults
  - context-switch
- Fewer Hardware events in VMs (especially “cloud”)

Static Tracepoints

- Interesting Tracepoints
  - raw_syscalls:sys_enter – look at all the tracepoints
  - syscalls:sys_enter_semop – profile lwlock waits
  - syscalls:sys_enter_select – profile spinlock waits
  - block:* – block layer tracepoints
  - sched:* – scheduler tracepoints
- Require root
- A bit faster than static tracepoints
- full trace by default, use -F to sample frequent ones

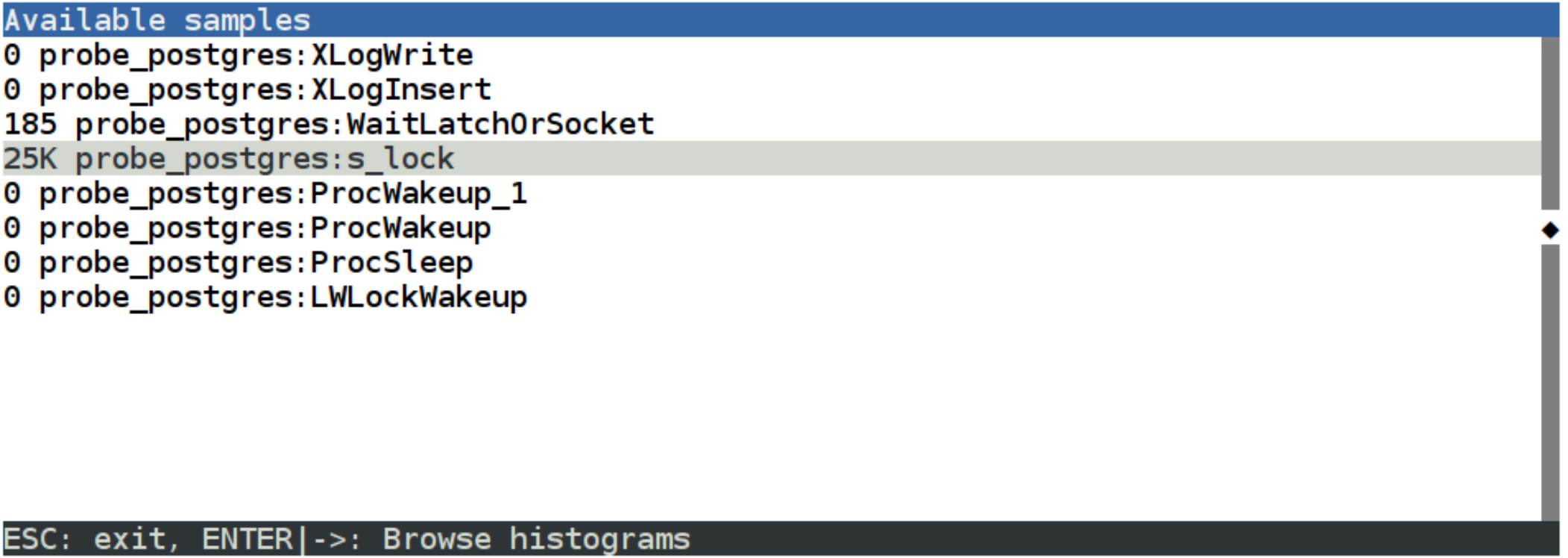
Dynamic Tracepoints

- Manage Dynamic Tracepoints
  - perf probe -l – list dynamic tracepoints
  - perf probe -x binary --add … – add tracepoint to binary
  - perf probe –del event/event*
  - perf probe -x … --line $func – show lines you can trace
- --add function/function:line/…
- Require Debug Information
- Very useful, especially for measuring contention, causes of load and such
- Multiple Matches, _1, _2, ...

Important Dynamic Tracepoints

- s_lock – unavailable spinlock
- LWLockWakeup – blocked others in lwlock
- ProcSleep – waiting for other backend, e.g. heavyweight lock
- WaitLatchOrSocket – waiting for something, client commands or e.g. a proc wakeup
- XLogInsert()

Workload #1

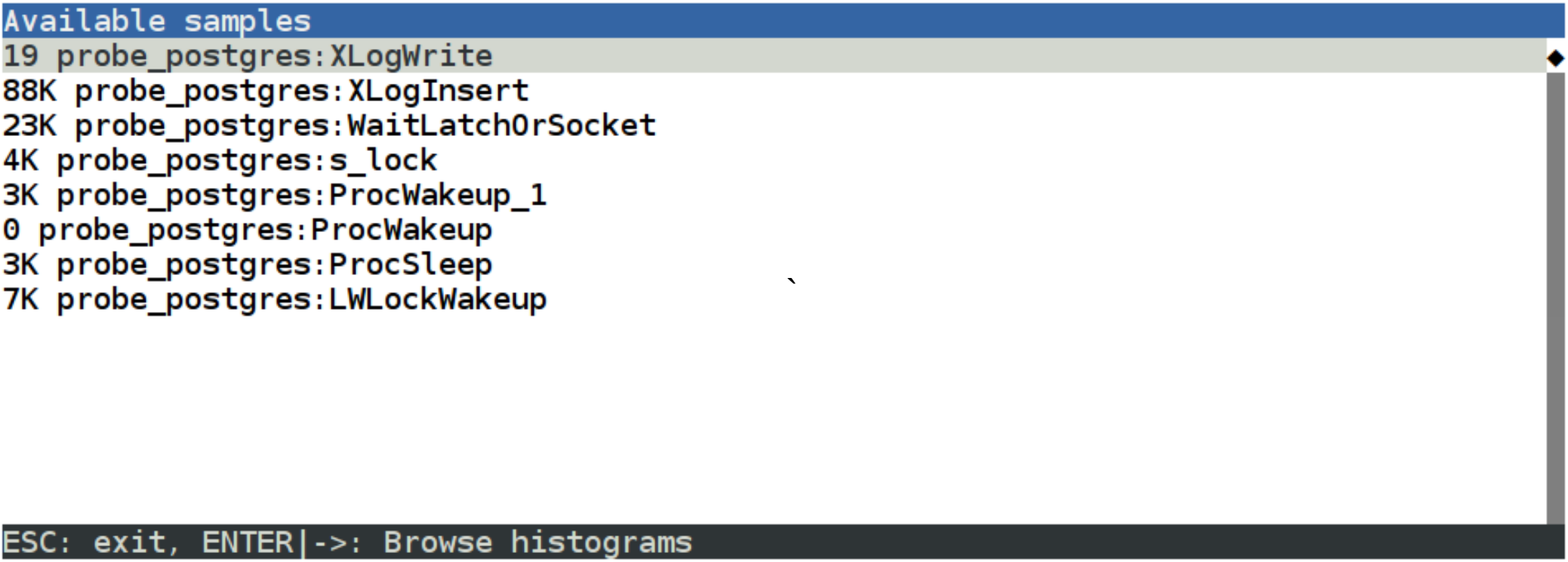

Available samples
0 probe_postgres:XLogWrite
0 probe_postgres:XLogInsert
185 probe_postgres:WaitLatchOrSocket
25K probe_postgres:s_lock
0 probe_postgres:ProcWakeup_1
0 probe_postgres:ProcWakeup
0 probe_postgres:ProcSleep
0 probe_postgres:LWLockWakeup
ESC: exit, ENTER|->: Browse histograms

Workload #1

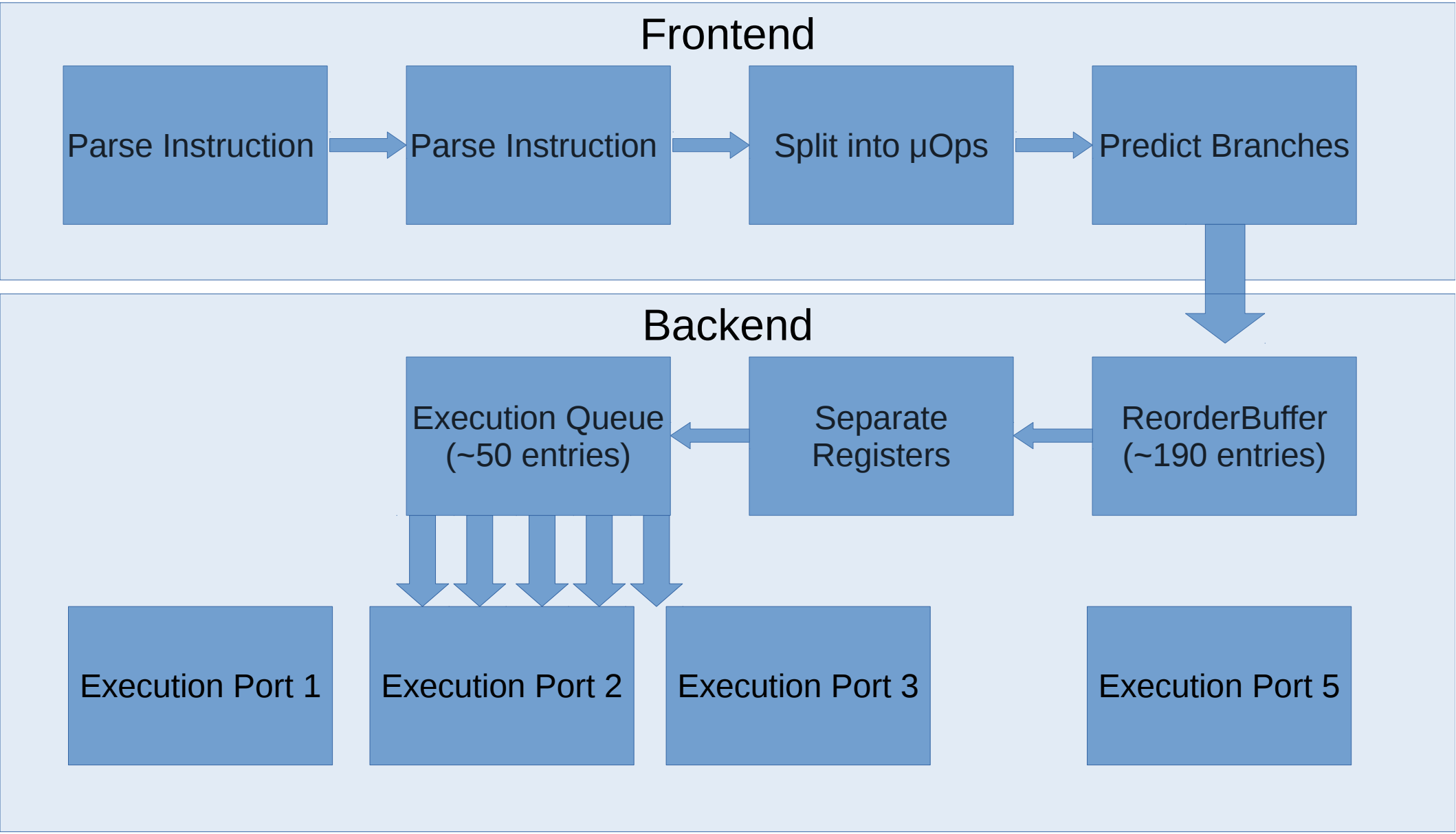
```
Samples: 25K of event 'probe_postgres:s_lock', Event count (approx.): 25916
  Overhead  Command   Shared Object  Symbol
-  100.00%  postgres  postgres       [.] s_lock
   - s_lock
      + 50.73% ReleaseAndReadBuffer
      + 49.27% ReadBuffer_common

For a higher level overview, try: perf report --sort comm,dso
```

Workload #2


Available samples
19 probe_postgres:XLogWrite
88K probe_postgres:XLogInsert
23K probe_postgres:WaitLatchOrSocket
4K probe_postgres:s_lock
3K probe_postgres:ProcWakeup_1
0 probe_postgres:ProcWakeup
3K probe_postgres:ProcSleep
7K probe_postgres:LWLockWakeup
ESC: exit, ENTER|->: Browse histograms

Quick Intro into modern CPUs



Consequences of modern CPUs

- Out-of-Order hides latencies
- Hidden latencies make profiling much harder
  - sometimes a cache miss is fata
  - most of the time a cache miss is harmless
- Independent instructions allow reordering
- Stalling the entire pipeline is extremely expensive
- Should have it's own talk

Additional Tools

- pmu-tools
  - https://github.com/andikleen/pmu-tools
  - ocperf list – show low level intel hardware events
  - toplev – look for “pipeline bottleneck”
    - highlevel, not line level profile
- flame graph generator
  - https://github.com/brendangregg/FlameGraph
  - shows profile over time in a graphical manner