

# Developing Applications with APR

Paul Querna  
[pquerna@apache.org](mailto:pquerna@apache.org)

July 21, 2005



<http://www.outoforder.cc/presentations/>

# A.P.R.

- Apache Portable Runtime
- Origin: httpd
- Platforms: Unix, Netware, OS/2, Windows.

Your Application

APR-Util

APR

Libraries

Operating System

# Integration Points

- Modules
  - Wrap Areas that use APR/Pools
    - C++ Objects
- Entire Application
  - Memory pools everywhere

# Layers

- Memory Allocation
- OS Portability Functions
- Library/Utility Functions (APR-Util)

# Memory Allocation.

- Memory Pools
  - No free()
  - Less overhead than Garbage Collection
  - Still requires thinking!
    - Loops
    - Long running tasks

```
apr_pool_create(&pool, NULL);
```

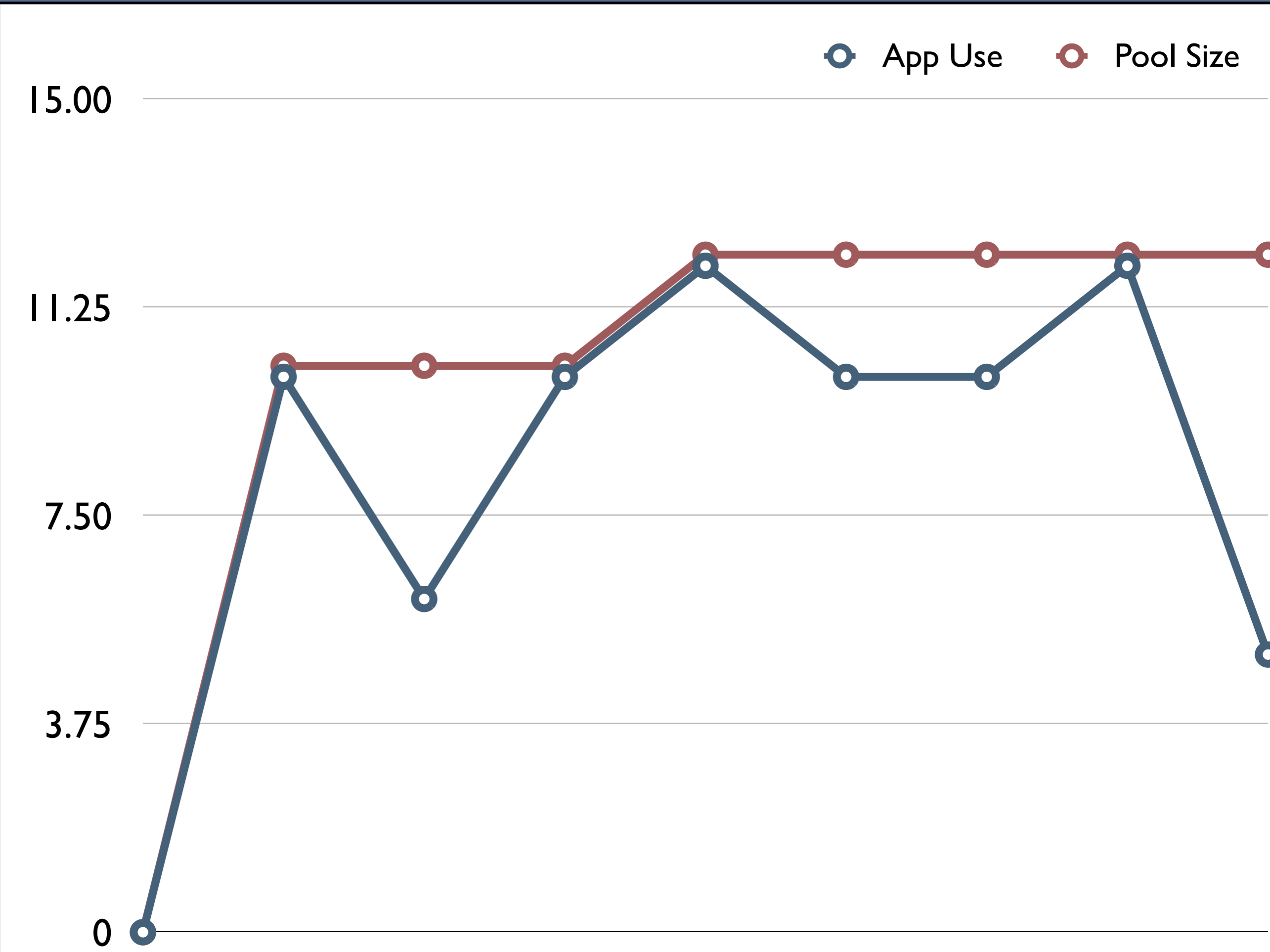
```
for (i=0; i < n; ++i) {
```

```
    do_something(pool, i);
```

```
    apr_pool_clear(pool);
```

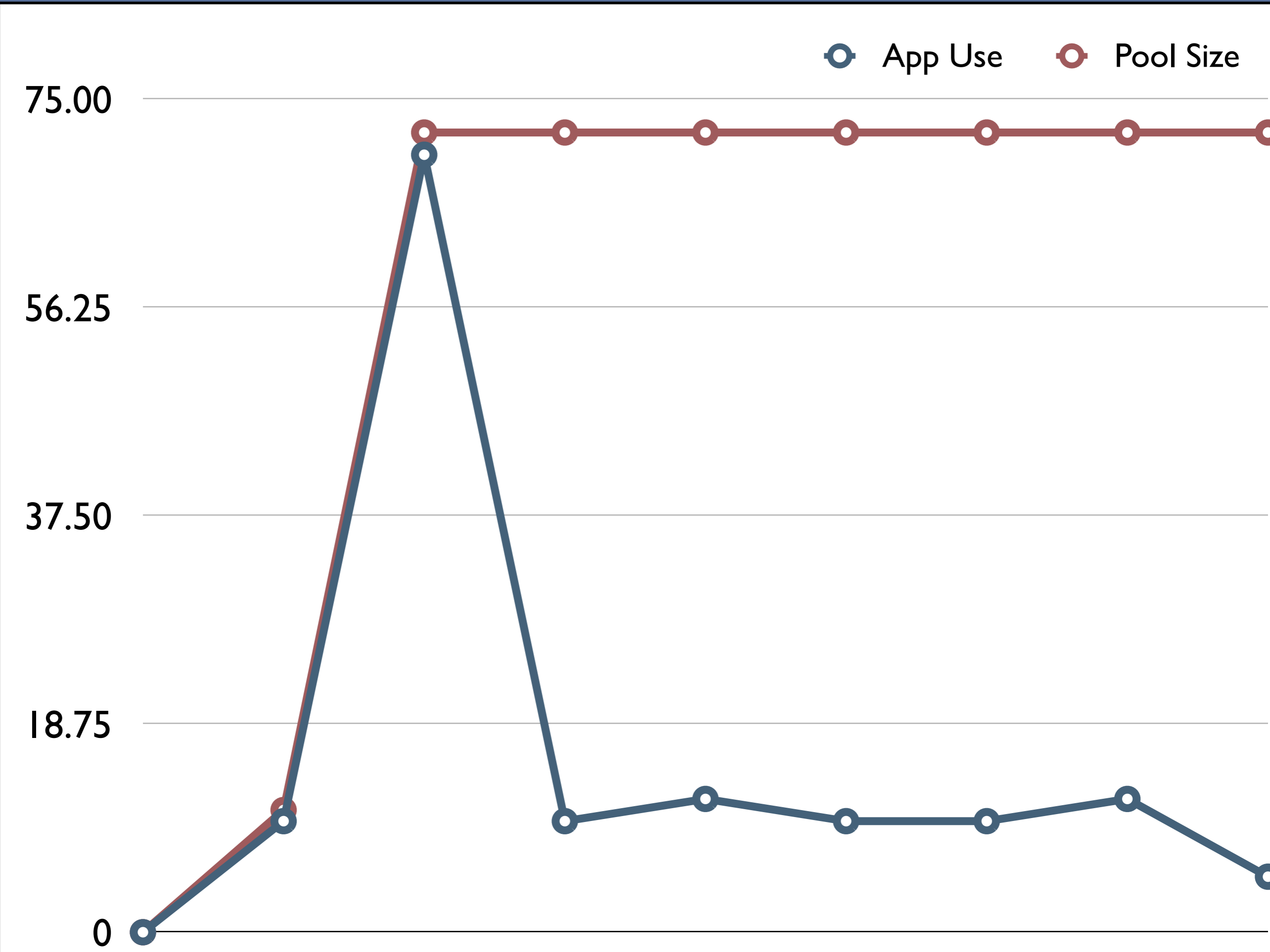
```
}
```

```
apr_pool_destroy(&pool);
```



# Not Perfect.

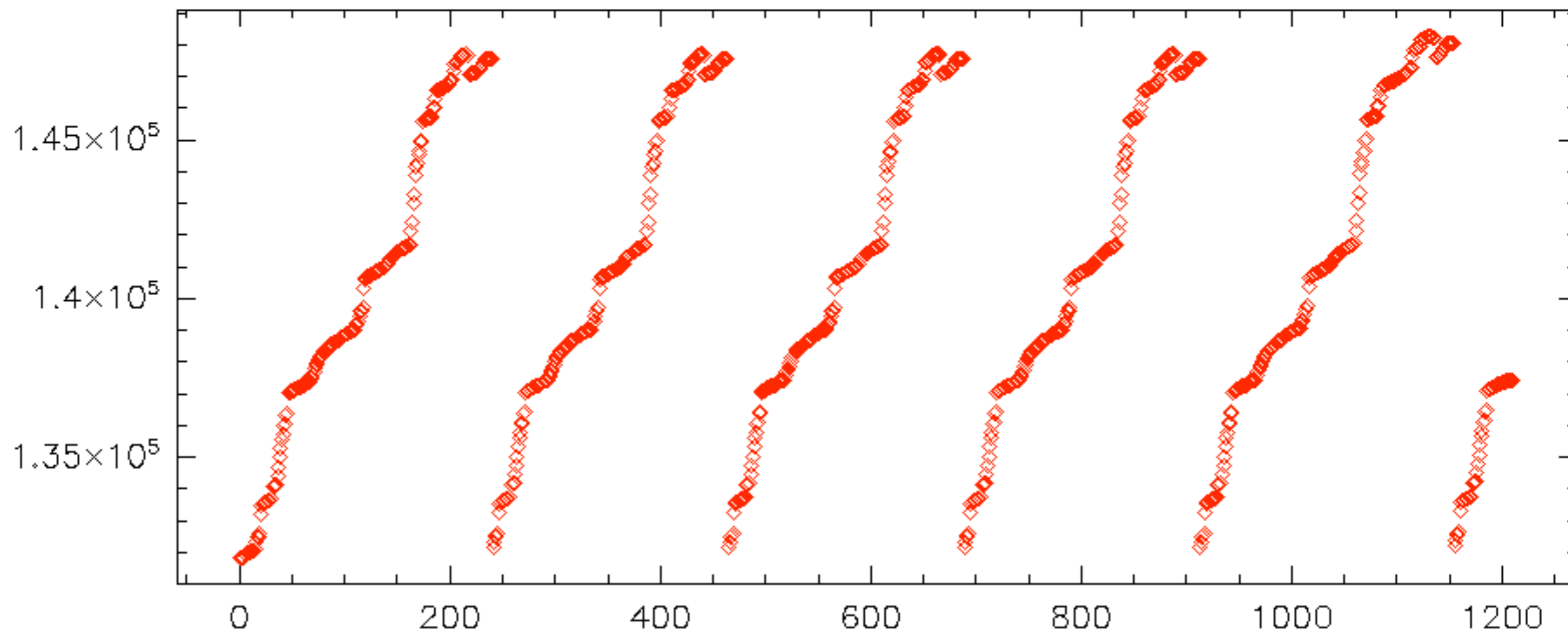
- Unbounded Memory Usage:
  - ‘leaks’
    - Pools will not free() to the OS.
  - Can be handled with sub-pools.

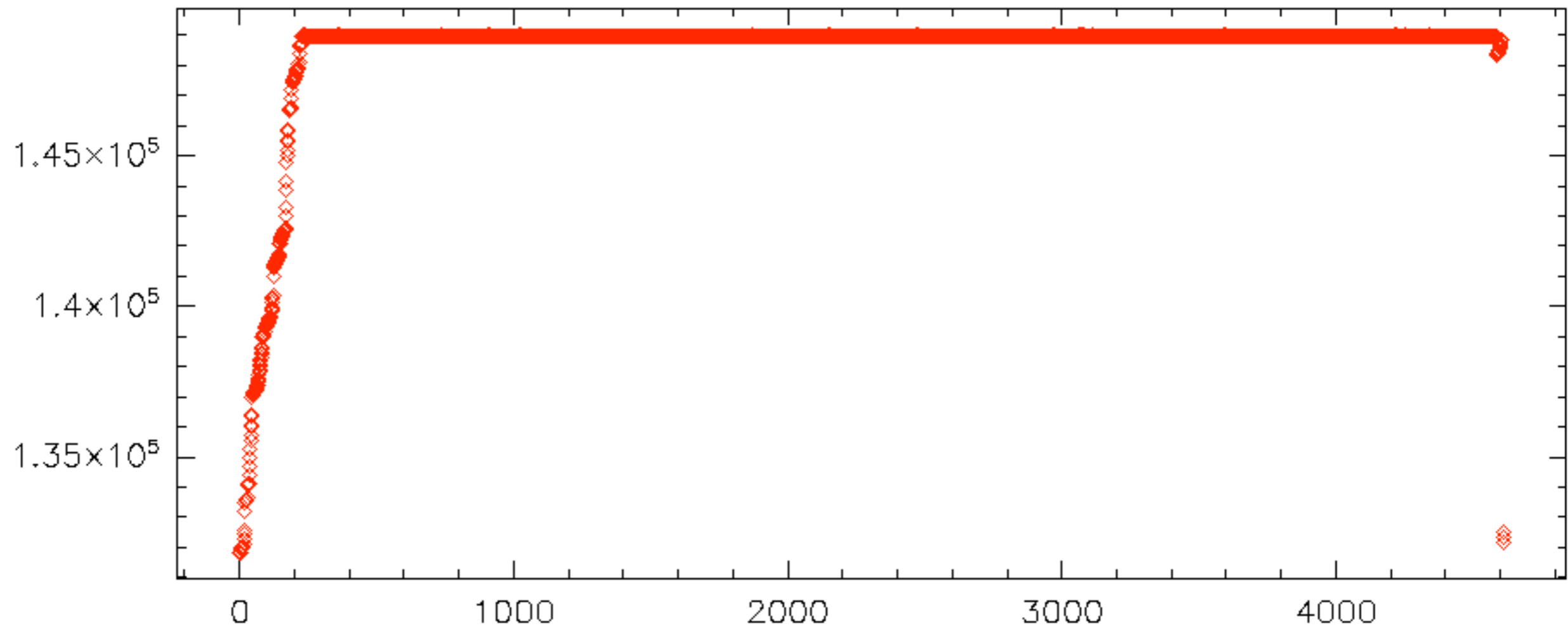


```
void use_lots_of_ram(apr_pool_t* pool)
{
    do {
        void* data = apr_palloc(pool, 500);
        ....
    }
    while(cond_true());
}
```

```
void use_lots_of_ram(apr_pool_t* pool)
{
    apr_pool_create(&subpool, pool);
    do {
        void* data = apr_palloc(subpool, 500);
        ...
    }
    while(cond_true());
    apr_pool_destroy(subpool);
}
```

```
void use_lots_of_ram(apr_pool_t* pool)
{
    apr_pool_create(&subpool, pool);
    do {
        void* data = apr_palloc(subpool, 500);
        ...
        apr_pool_clear(subpool);
    }
    while(cond_true());
    apr_pool_destroy(subpool);
}
```





# Integration Points

- Modules
  - Wrap Areas that use APR/Pools
    - C++ Objects
- Entire Application
  - Memory pools everywhere

# Simple Applications

- Few Calls to initialize APR
- Cleanup

```
#include "apr.h"
#include "apr_file_io.h"
int main(int argc, char *argv[])
{
    apr_pool_t *p;
    apr_file_t *fp;
    apr_initialize();
    atexit(apr_terminate);
    apr_pool_create(&p, NULL);
    apr_file_open_stdout(&fp, p);
    apr_file_printf(fp, "Hello World\n");
    return 0;
}
```

# Data Structures

- `apr_array_header_t`
- `apr_table_t`
- `apr_hash_t`
- APR Rings

# Arrays

- `apr_array_make()`
  - initial size
- `apr_array_push()`
  - returns pointer to new memory
- `nelts`

# Tables

- Keys & Values
- Used for HTTP Headers
- `apr_table_do`

# Hashes

- Keys & Values
- Arbitrary (void\*)

# Rings

- Double Linked List
- Optional Debugging

# apr\_status\_t

- `0 == APR_SUCCESS`
- Always use `APR_STATUS_IS_*` macros

# File IO

- read, write, writev, sendfile, seek, locking
- functions for stderr/stdout/stdin

# Network IO

- read, write, writev, sendfile
- sockaddr
- pollset
  - kqueue/epoll/event ports

# Threading

- Similar to pthreads
  - threads
  - mutex
    - thread
    - process
    - global

# APR-Util

- Everything that doesn't have a home?

# Bucket Brigades

- Complicated -- entire sessions in past AC are on them
- Buckets contain:
  - Files
  - Pipes
  - Sockets
  - Custom
- Uses `APR_RING_*`

# DBM

- iterate, fetch, insert
- must be explicitly closed -- no cleanup

# apr\_reslist

- Pooling of Resources made easy
  - Proxy
  - DBI
- Timeouts
- Creation
- Destruction

# apr\_dbd

- One API:
  - MySQL
  - Postgres
  - SQLite v2 & v3
  - Oracle (in progress)

```
apr_dbd_driver_t* driver = NULL;
apr_dbd_t* handle = NULL;
apr_dbd_get_driver(pool, "sqlite3", &driver);
apr_dbd_open(driver, pool, "db", &handle);
apr_dbd_query(driver, handle, &nrows, sql);
```

# Questions?

- <http://www.outoforder.cc/presentations/>