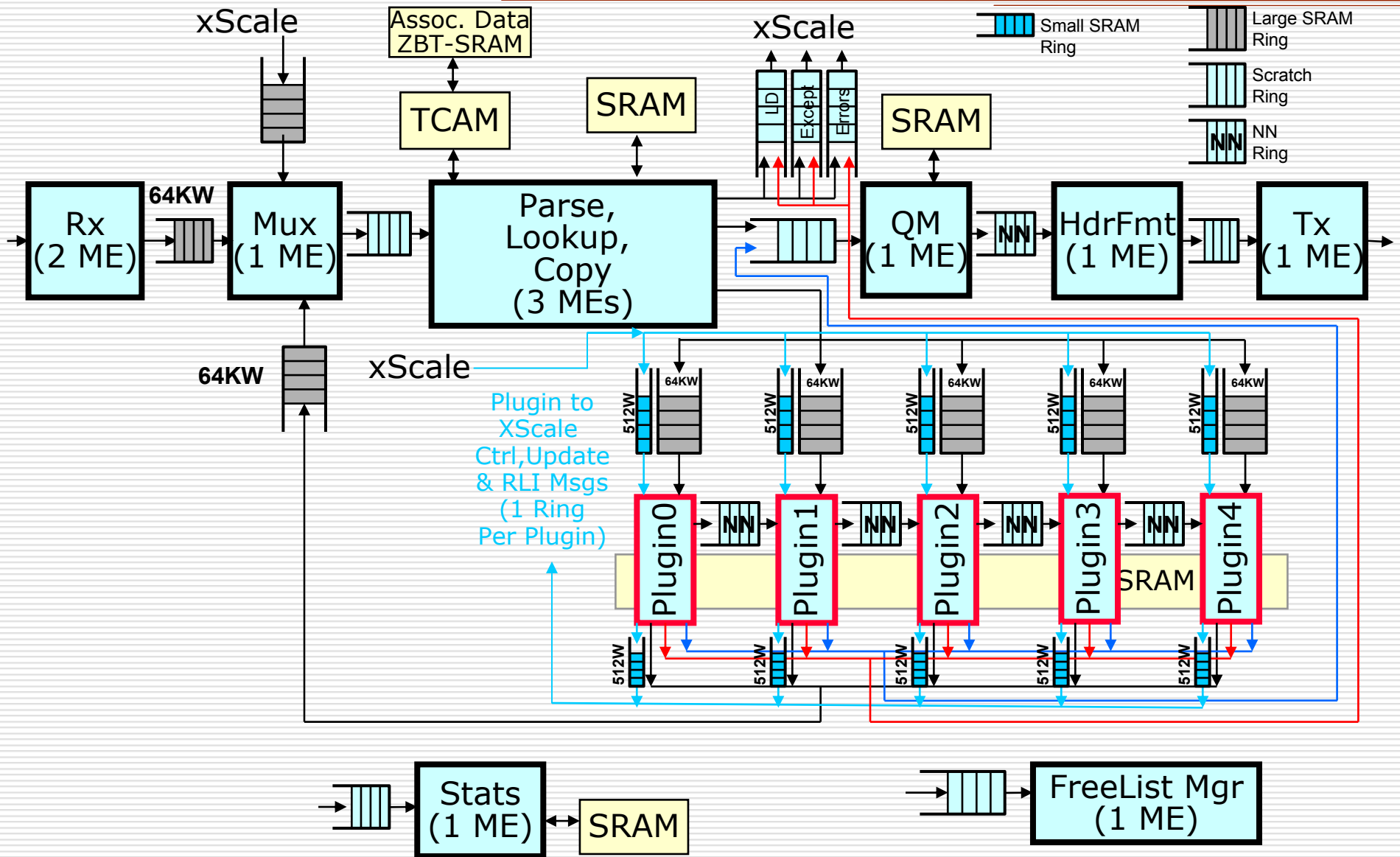


ONL NP Router

Plugins

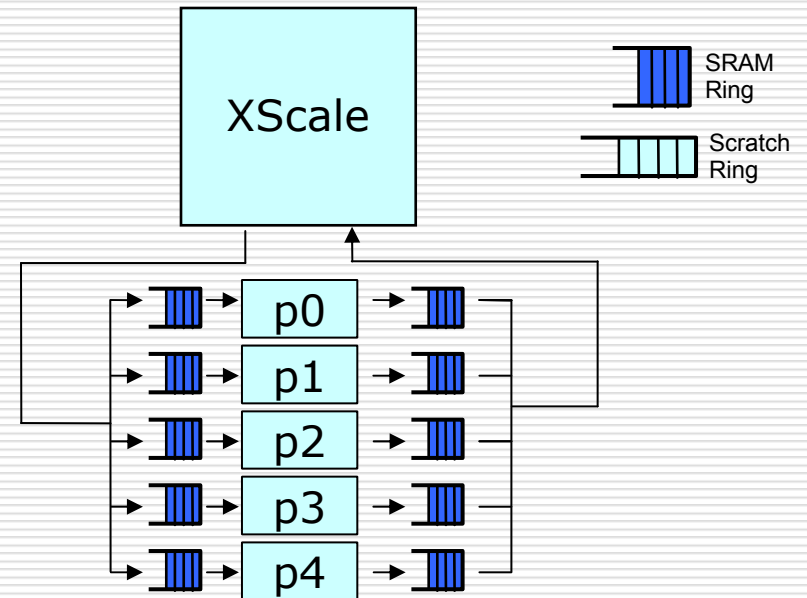
Shakir James, Charlie Wiseman,
Ken Wong, John DeHart
{scj1, cgw1, kenw, jdd}@arl.wustl.edu

ONL NP Router

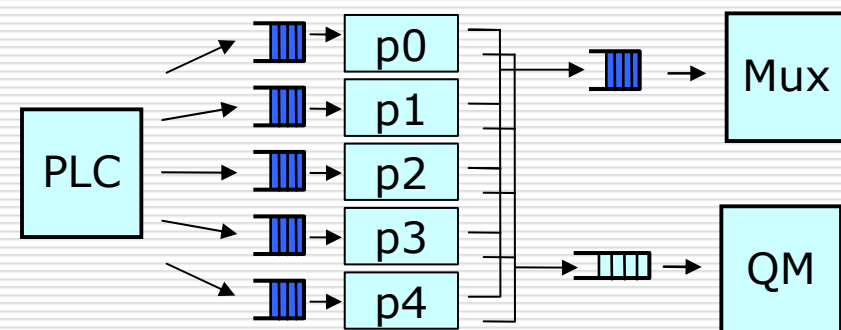


Design Review: Plugins

- Control path
 - » XScale \leftrightarrow Plugins



- Data path
 - » Plugins \rightarrow {Mux, QM}



Outline

- Overview
- My first plugin
- My second plugin
- Framework internals
- Core plugins
- Conclusion

Overview

Plugin Framework

```
graph TD; A[Plugin Framework] --> B[Base Plugin]; A --> C[API];
```

Base Plugin

- `handle_init_user()`
- `handle_pkt_user()`
- `handle_msg_user()`
- `handle_callback_user()`

API

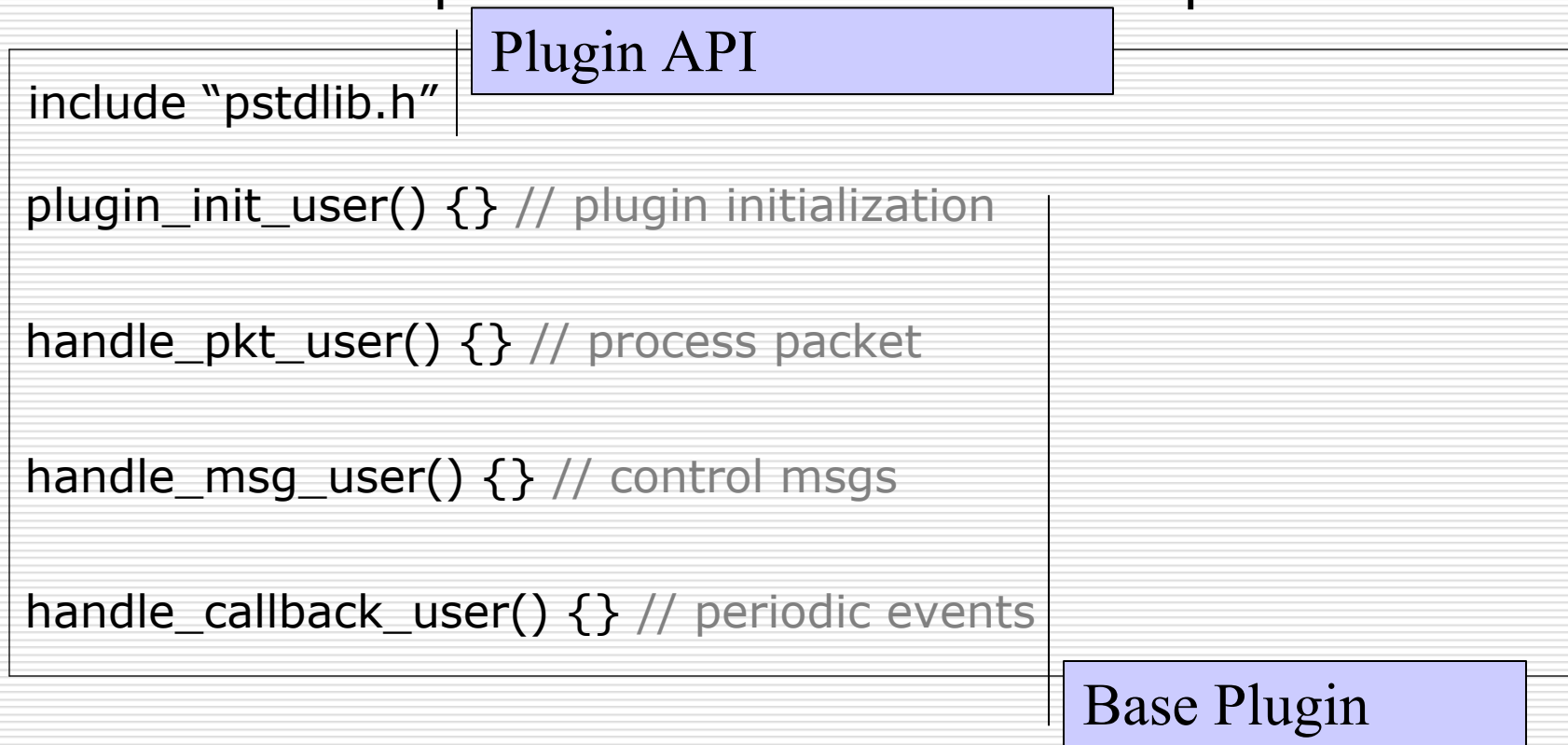
- constants
- macros
- functions

Goal: Provide a simple API that “lowers the barrier” to entry for new IXP programmer.

My First Plugin

- Counter

- » Increment a pkt counter and forward packet



My First Plugin

- Counter

- » Increment a pkt counter and forward packet

```
include "pstdlib.h"  
  
...  
handle_pkt_user() // process packet  
{  
    pcount_inc(0); // increment local counter  
}  
  
...
```

My Second Plugin

- Stats

- » Increment ICMP, UDP, and TCP counters

```
handle_pkt_user() // process packet
{
    buf_handle = pkt_get_buf_handle(); //get buf handle
    ipv4_hdr = pkt_get_ipv4_hdr(buf_handle); // read IP Hdr from DRAM

    switch(ipv4_hdr.protocol) {
        case PROTO_ICMP:
            pcount_inc(0);
            break;
    }
}
```

Framework Internals

- Framework components
 - » Base Plugin
 - » Plugin API (Plugin Std Library – pstdlib)

- Base Plugin
 - » Dispatch loop: `main()`
 - » Plugin initialization: `plugin_init()`
 - » Packet processing: `handle_pkt()`
 - » Control messages: `handle_msg()`
 - » Callbacks: `callback()`

Framework Base Plugin: main()

- Dispatch loop: main()

```
plugin_init()
while (1)
    switch(_ctx())
        case 0, 1, 2, 3, 4, 5: handle_packet();
        case 6: callback();
        case 7: handle_msg();
```

- Preprocessor directives:

- » FIRST_PACKET_THREAD=**0**
- » LAST_PACKET_THREAD=**5**
- » MESSAGE_THREAD=**6**
- » CALLBACK_THREAD=**7**
- » DL_ORDERED

Framework Base Plugin: plugin_init()

- Plugin initialization: plugin_init()

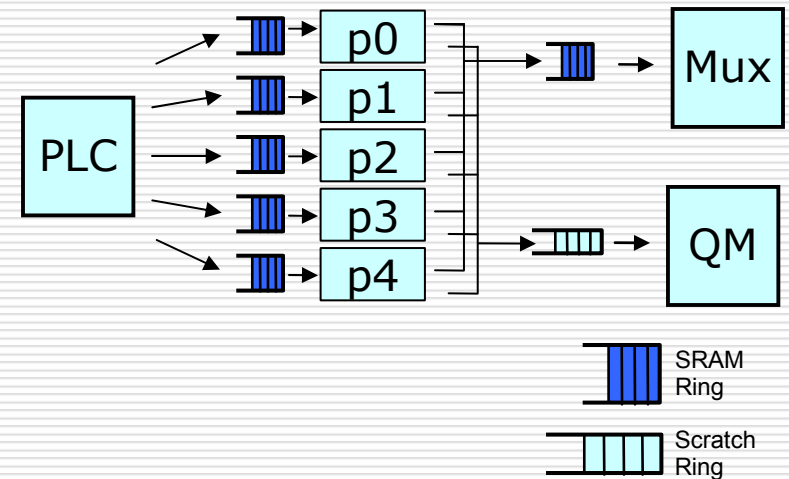
```
... ONL plugin initialization ...  
plugin_init_user(); // user hook
```

- ONL plugin initialization
 - » Local ME state (data/ctrl ring)
 - PACKET_IN_RING_0, MESSAGE_IN_RING_0
- User hook: plugin_init_user()
 - » default is stub
 - » E.g.: init vars, set timer, change (default) nextBlock

Framework Base Plugin: handle_pkt

- Packet processing: handle_packet()

```
dl_source_packet();  
  
default_format_out_data();  
  
handle_pkt_user(); // user hook  
  
dl_sink_packet();
```



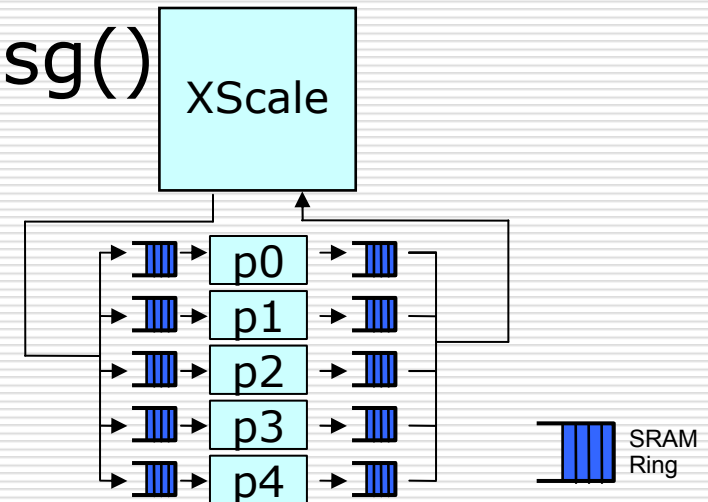
- dl_source_packet(), dl_sink_packet()
 - » uses ME state (plugin_init)

- User hook: handle_pkt_user()

Framework Base Plugin: handle_msg()

- Control messages: handle_msg()

```
dl_source_message();  
  
handle_msg_user(); //user hook  
  
dl_sink_message();
```



- dl_source_message(), dl_sink_message()
 - » uses ME state (plugin_init)
 - » mutex_lock: multiple SRAM reads per msg possible
- User hook: handle_msg_user()
 - » Handle control msg from RLI
 - » Send configuration updates to XScale

Framework Base Plugin: callback()

- Callbacks: callback()

```
sleep(timeout);  
handle_callback(); // user hook
```

- sleep()

- » timeout in cycles (ME's operate at 1.4 GHz)
- » Override default timeout (1000) in plugin_init_user()

- User hook: handle_callback_user()

- » Non-packet arrival driven processing: delay plugin

Framework Plugin API: pstdlib

- Framework components
 - » Base Plugin
 - » Plugin API (Plugin Std Library: pstdlib)

- Plugin API: pstdlib
 - » Constants
 - » Macros
 - » Functions

Plugin API functions -Current

- Stats - read/write stats counters
 - » PCNTR_INC(int), plugin_cntr_inc(int, int) , ...
 - » PCNTR_ADD(int, int), plugin_cntr_add(int, int, int)
- Packet forwarding and dropping,
 - » pkt_set_out_to_MUX() // set dINextBlock
 - » pkt_set_out_to_QM() // set dINextBlock
 - » pkt_drop() // drop packet
- Packet access/modification
 - » pkt_get_buf_handle() // get next buffer handle
 - » pkt_get_ipv4_hdr() // read pkt data from DRAM

Plugin API functions –Coming Soon

- Packet access/modification
 - » `pkt_get_udp_hdr()` // read udp hdr from DRAM
 - » `pkt_get_tcp_hdr()` // read tcp hdr from DRAM
 - » `pkt_get_icmp_hdr()` // read icmp hdr from DRAM
- Allocate a new pkt
- Copy pkts
- Read Queue Length

Plugin API functions –Coming Later

- These will be added eventually (possibly as needed):
 - » Configuration reads
 - get_queue_threshold
 - get_queue_quantum
 - » Configuration updates
 - add_filter
 - add_route
 - update_queue_threshold
 - update_queue_quantum
 - » Queues of packets with-in the plugin
 - » Wrap intrinsic functions (Clock, CRC, local CAM access)

Core Plugins

- null
 - » Forward packets
- count
 - » Increment pkt counter and forward pkt
- nstats
 - » Increment ICMP, UPD and TCP pkt counter
 - » Drop pkt

Conclusion

- Did we achieve our goal?
 - » Provide a simple API that “lowers the barrier” to entry for new IXP programmer
- What’s next?
 - » Add Core Plugins
 - dumphdr, multicast, stringSub, pdelay, dropdelay
 - » Expand Plugin API
 - More functions based on course assignments
 - » Hardware Testing
 - » Inter-plugin communication?

TODO for cgw1,kenw,scj1

- Control path
- Core Plugins
 - » multicast
 - forward pkt copies to ports indicated by destination vector
 - » stringSub
 - Replace all occurrence of the string "HELLO" with "adieu"
 - » pdelay
 - delay pkt
 - » dropdelay
 - delay pkt with probability m/n , and drop with probability $(n-m)/n$
 - » dumphdr
 - Save IP header fields and transport header fields
 - Return header fields through control msg