FTF | FREESCALE TECHNOLOGY FORUM 2015

AIOP: Programming Model, Enablement Software and Integration

FTF-DES-F1378

Howard Owens | Software Architects
Michael Kardonik | Software Architects
Peter Newton | Software Architects

J U N E . 2 0 1 5

freescale™

External Use

Agenda

- Why AIOP ?
- Integration models and customers
- Block diagram
- Memory model
- Task and scheduler
- Synchronization and ordering
- Frame operations
- Architecture of AIOP software
- Tools

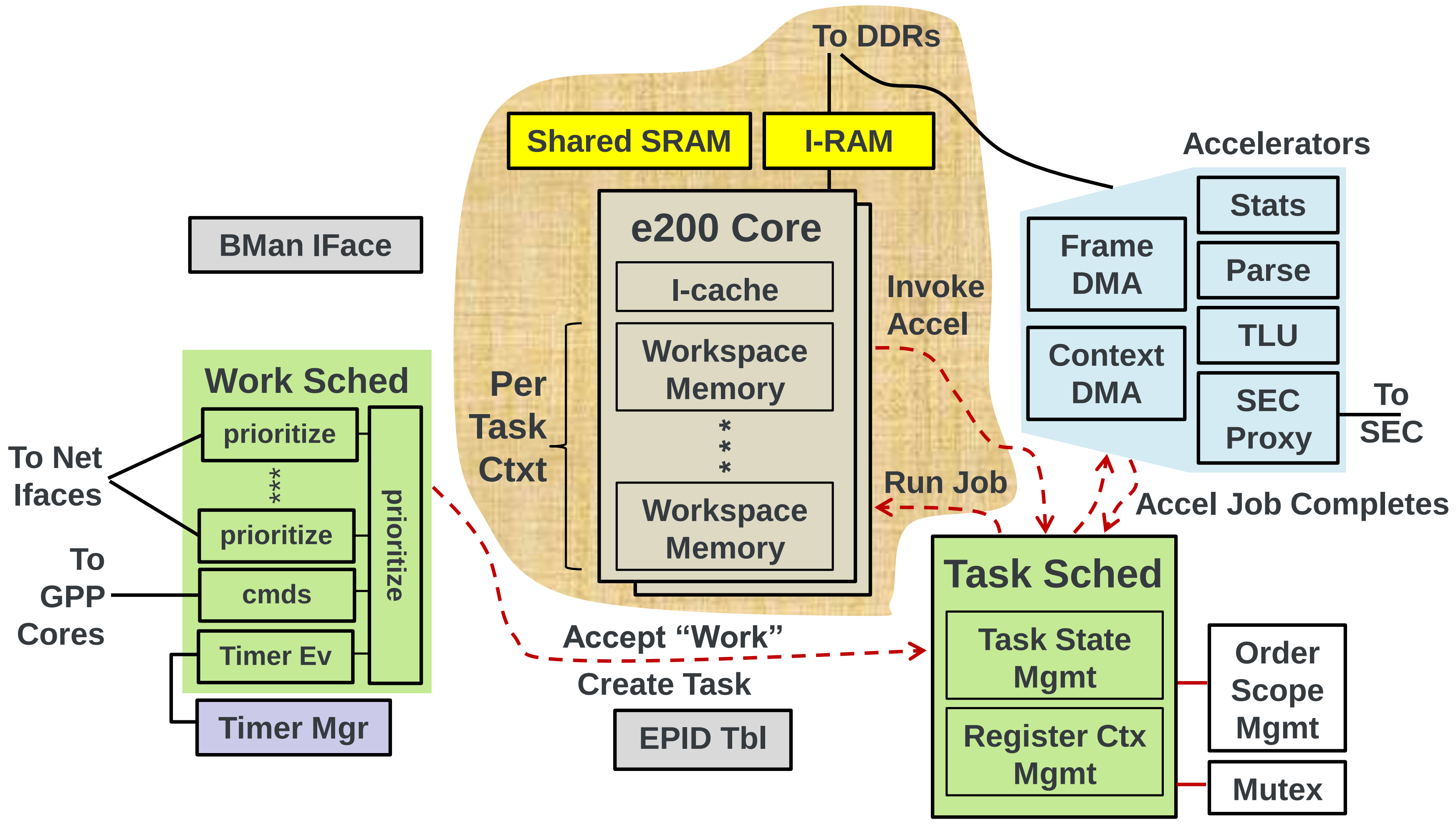
Why AIOP ?

General purpose processors sub-optimal for packet processing	AIOP approach
Low packet processing locality underutilizes cache and pipeline and suffers from high DDR latency	Specialized memory hierarchy and explicit DMA operations instead of cache allow deterministic performance.
Increasing single thread performance causes super-linear power increase	Parallelism with more “small” cores
Performance and complexity of software hiding latency of asynchronous access to accelerators	Hardware scheduler based multitasking environment hides access latency
Many cycles are wasted for standard procedures – search, parse, operations on frame, timers, statistics etc)	Hardware accelerators for common tasks: lookups, parse, frame operations, timers
Performance and complexity of software ordering and synchronization	Functions are provided by hardware scheduler

Why AIOP ?

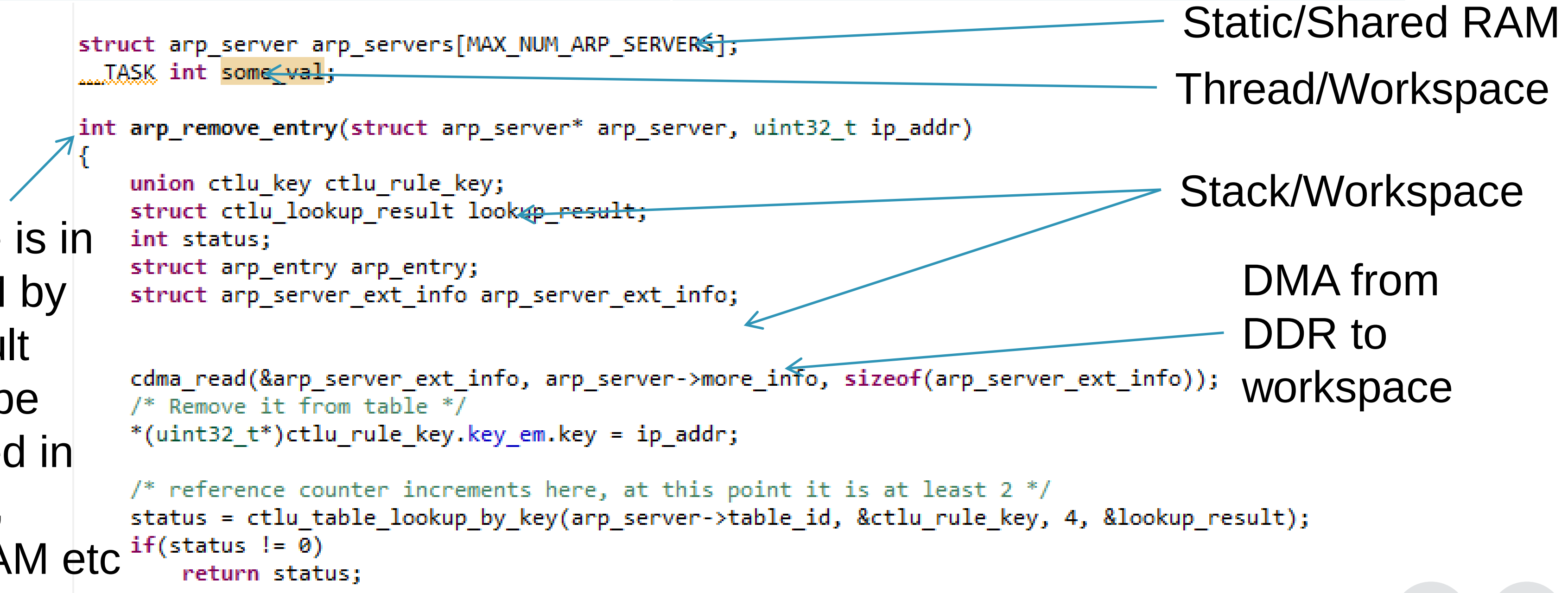
Typical NPU: efficient but not easy to use	AIOP approach
Proprietary languages (data flow oriented, assembler etc). Sometimes restricted ‘C’ is used	Standard ‘C’ language following procedural programming paradigm. Modularity is supported thru standard language methods
Tightly coupled to control processor that has intimate knowledge of NPU	Possible to build self-contained network nodes. Data/control structures are not shared enabling independence
Congestion management is complicated because of need to handle it within stages of pipeline	Congestion management is centralized into network interface that handles it before AIOP task start

AIOP *Architecture* Block Diagram



AIOP Memory Model

'C' variable	AIOP
Stack, Thread Local	Workspace memory, very fast and relatively small
Static	Shared RAM
Dynamic	DDR, Express memory, accessed with DMA



```
struct arp_server arp_servers[MAX_NUM_ARP_SERVERS];
__TASK int some_val;

int arp_remove_entry(struct arp_server* arp_server, uint32_t ip_addr)
{
    union ctlu_key ctlu_rule_key;
    struct ctlu_lookup_result lookup_result;
    int status;
    struct arp_entry arp_entry;
    struct arp_server_ext_info arp_server_ext_info;


    cdma_read(&arp_server_ext_info, arp_server->more_info, sizeof(arp_server_ext_info));
    /* Remove it from table */
    *(uint32_t*)ctlu_rule_key.key_em.key = ip_addr;

    /* reference counter increments here, at this point it is at least 2 */
    status = ctlu_table_lookup_by_key(arp_server->table_id, &ctlu_rule_key, 4, &lookup_result);
    if(status != 0)
        return status;
```

Code is in IRAM by default
Can be placed in DDR, ShRAM etc

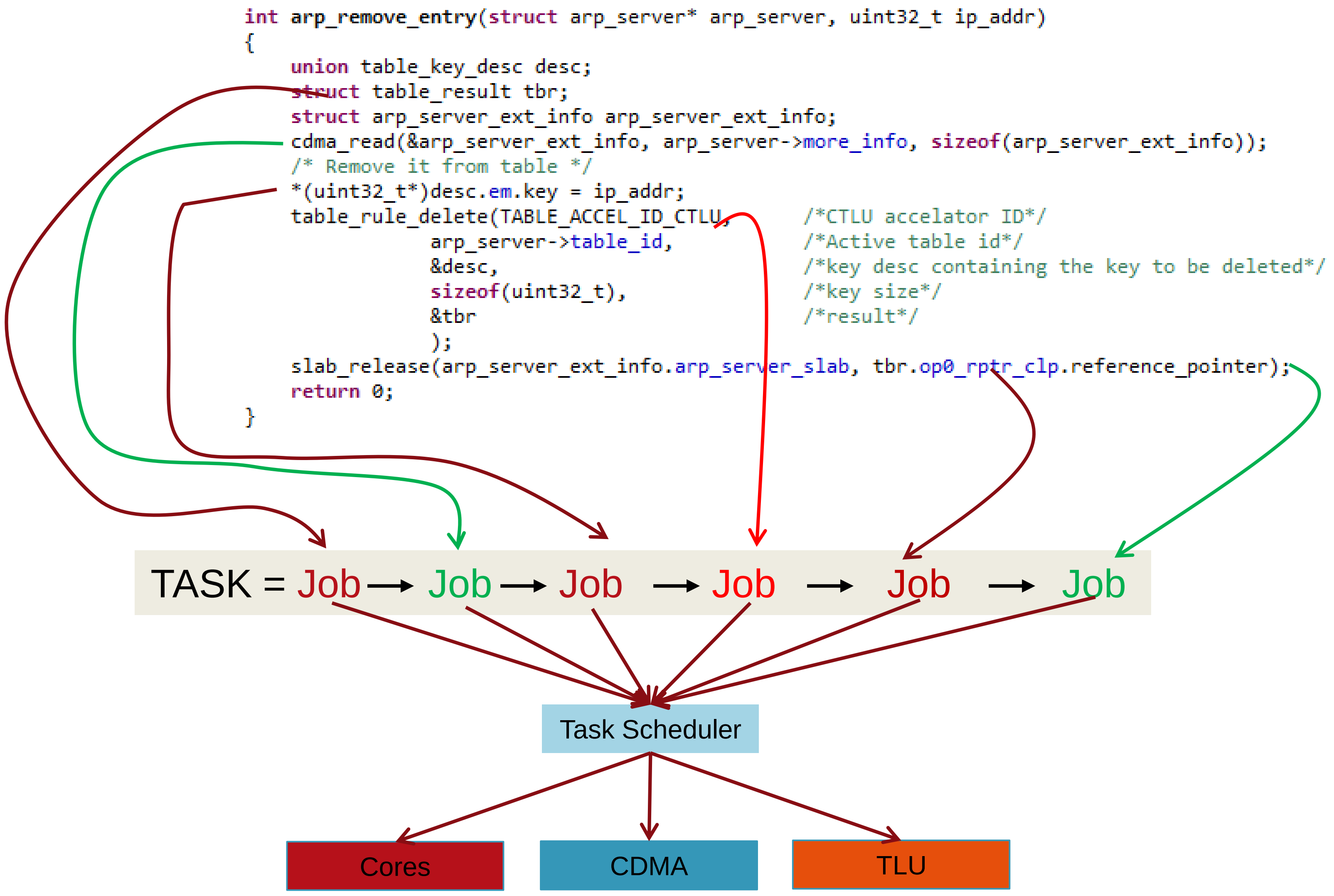
AIOP: Works, Tasks and Jobs

- Tasks are created by work scheduler to accomplish Work
- Processing elements: cores, accelerators.
- **Job – minimal schedulable part of task that are executed by processing elements**
- **Task is a sequence of Jobs that share a context**

TASK = Job → Job → Job → * * * → Job → Job

- **HW-based scheduler** causes the Jobs to be scheduled to processing elements

‘C’ sequential program and accelerators call



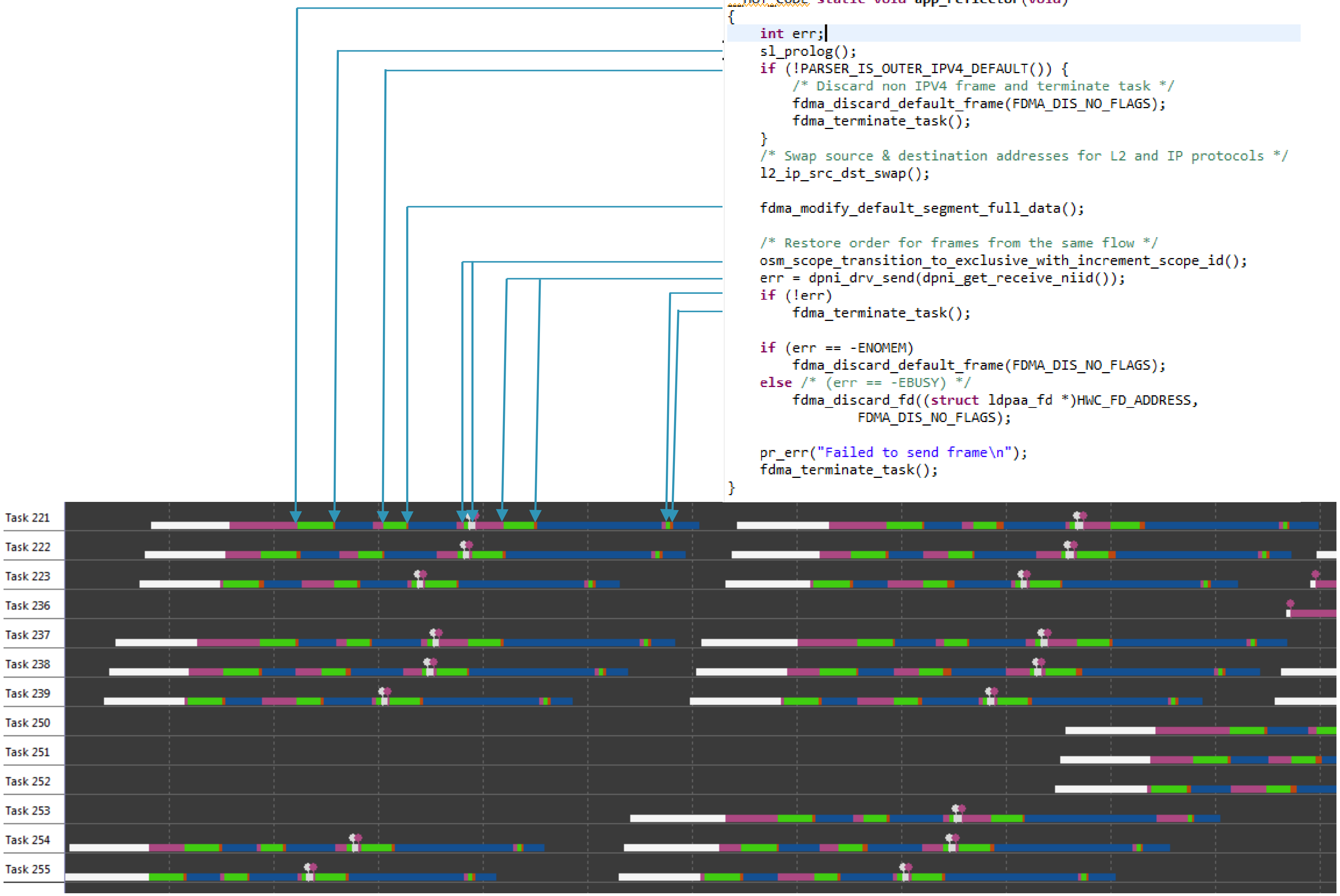
```
int arp_remove_entry(struct arp_server* arp_server, uint32_t ip_addr)
{
    union table_key_desc desc;
    struct table_result tbr;
    struct arp_server_ext_info arp_server_ext_info;
    cdma_read(&arp_server_ext_info, arp_server->more_info, sizeof(arp_server_ext_info));
    /* Remove it from table */
    *(uint32_t*)desc.em.key = ip_addr;
    table_rule_delete(TABLE_ACCEL_ID_CTLU,      /*CTLU accelator ID*/
                arp_server->table_id,            /*Active table id*/
                &desc,                           /*key desc containing the key to be deleted*/
                sizeof(uint32_t),                /*key size*/
                &tbr                             /*result*/
                );
    slab_release(arp_server_ext_info.arp_server_slab, tbr.op0_rptr_clp.reference_pointer);
    return 0;
}
```

AIOP Task Scheduler Design

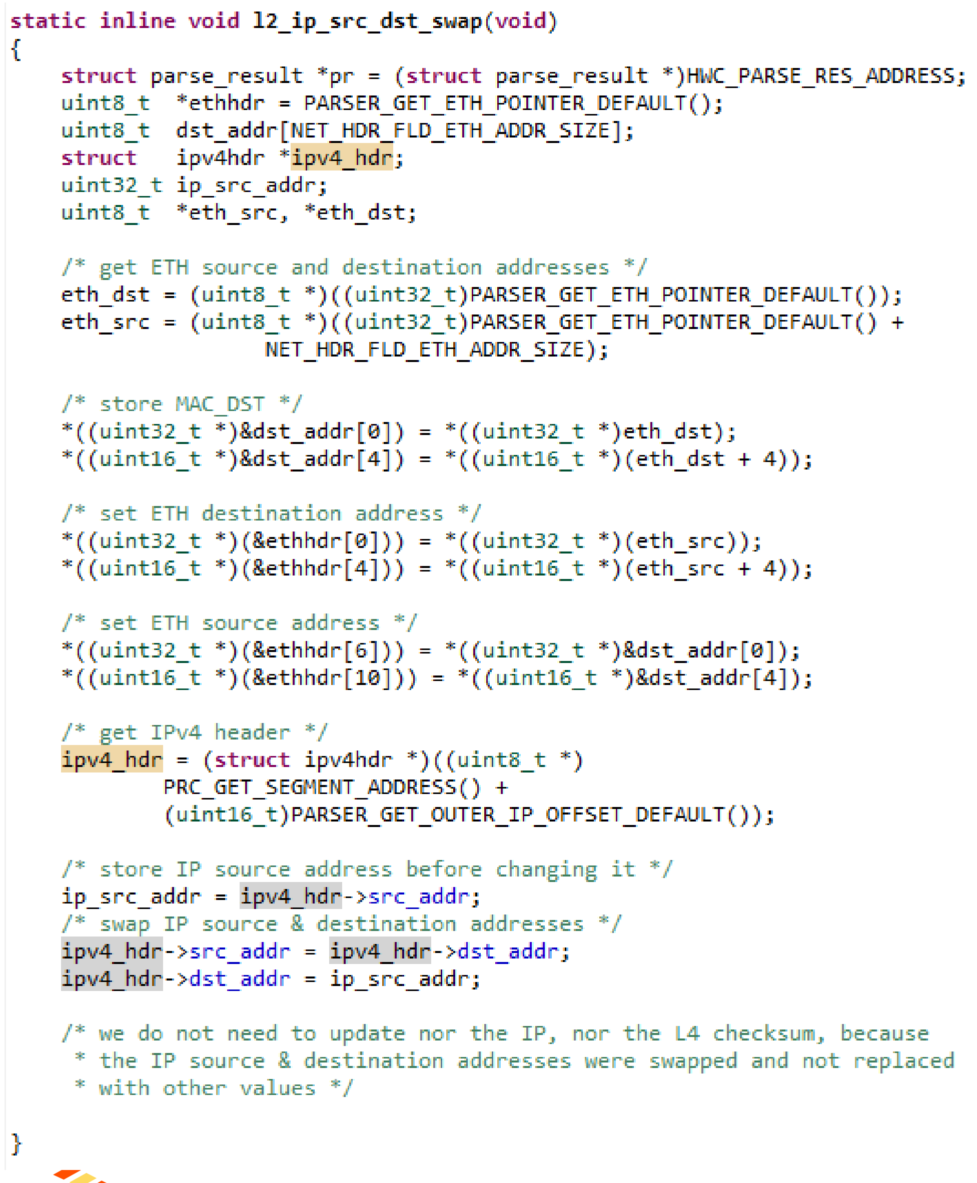
- Task scheduler is invoked on job boundaries to schedule jobs
- The AIOP scheduling and execution model is hardware based “SMP”. Jobs can execute on any processing element of right type
- Task scheduler is non-preemptive, based on a strict priority equal to task age
- Work scheduler accepts tasks according to sophisticated prioritization
- Latency of accelerator jobs is hidden by AIOP scheduling jobs from other tasks to keep processing element fully utilized
- Tasks do not know about each other; they cannot initiate or control each other directly
- Tasks do not know names or counts of processing element, and are never architecturally affine

Simple reflector code and trace

```
__HOT_CODE static void app_reflector(void)
{
    int err;
    sl_prolog();
    if (!PARSER_IS_OUTER_IPV4_DEFAULT()) {
        /* Discard non IPV4 frame and terminate task */
        fdma_discard_default_frame(FDMA_DIS_NO_FLAGS);
        fdma_terminate_task();
    }
    /* Swap source & destination addresses for L2 and IP protocols */
    l2_ip_src_dst_swap();

    fdma_modify_default_segment_full_data();

    /* Restore order for frames from the same flow */
    osm_scope_transition_to_exclusive_with_increment_scope_id();
    err = dpni_drv_send(dpni_get_receive_niid());
    if (!err)
        fdma_terminate_task();

    if (err == -ENOMEM)
        fdma_discard_default_frame(FDMA_DIS_NO_FLAGS);
    else /* (err == -EBUSY) */
        fdma_discard_fd((struct ldpaa_fd *)HWC_FD_ADDRESS,
                FDMA_DIS_NO_FLAGS);

    pr_err("Failed to send frame\n");
    fdma_terminate_task();
}
```

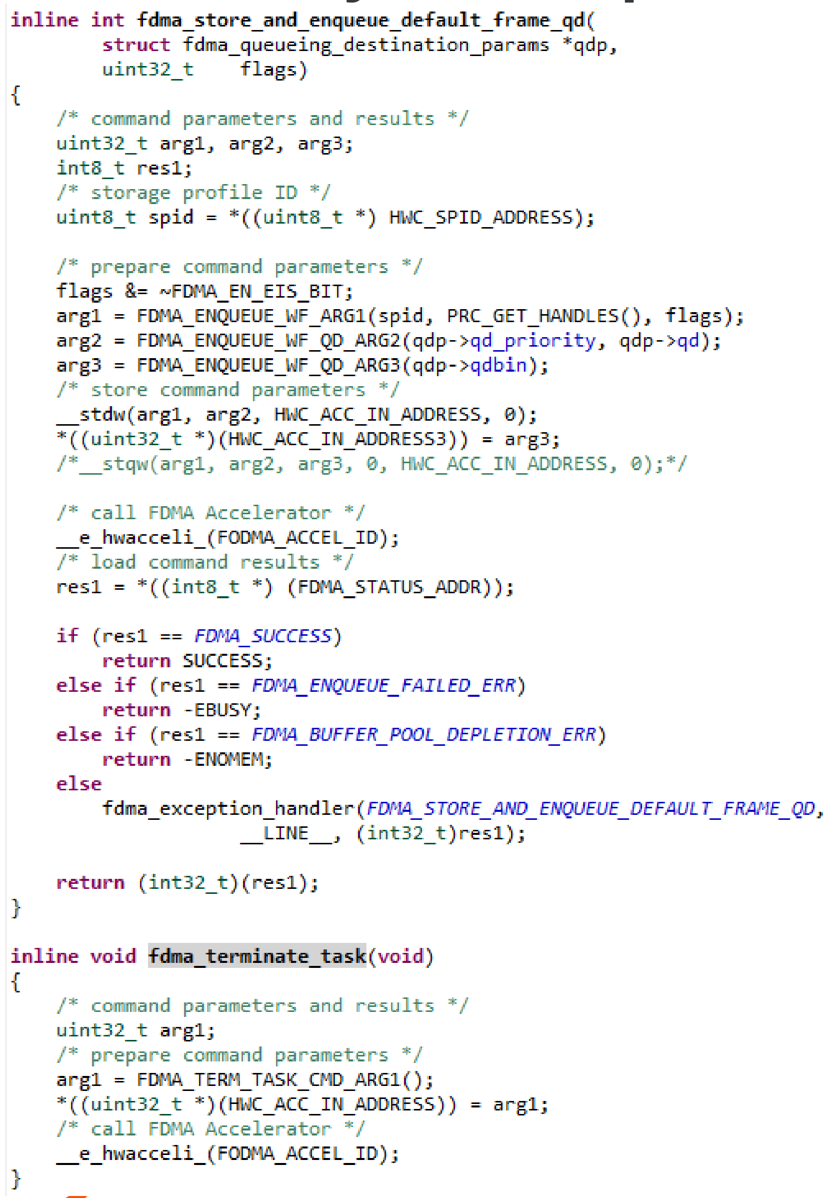


Simple reflector code and trace



```
static inline void l2_ip_src_dst_swap(void)
{
    struct parse_result *pr = (struct parse_result *)HWC_PARSE_RES_ADDRESS;
    uint8_t  *ethhdr = PARSER_GET_ETH_POINTER_DEFAULT();
    uint8_t  dst_addr[NET_HDR_FLD_ETH_ADDR_SIZE];
    struct   ipv4hdr *ipv4_hdr;
    uint32_t ip_src_addr;
    uint8_t  *eth_src, *eth_dst;

    /* get ETH source and destination addresses */
    eth_dst = (uint8_t *)((uint32_t)PARSER_GET_ETH_POINTER_DEFAULT());
    eth_src = (uint8_t *)((uint32_t)PARSER_GET_ETH_POINTER_DEFAULT() +
                NET_HDR_FLD_ETH_ADDR_SIZE);

    /* store MAC_DST */
    *((uint32_t *)&dst_addr[0]) = *((uint32_t *)eth_dst);
    *((uint16_t *)&dst_addr[4]) = *((uint16_t *)(eth_dst + 4));

    /* set ETH destination address */
    *((uint32_t *)(&ethhdr[0])) = *((uint32_t *)(eth_src));
    *((uint16_t *)(&ethhdr[4])) = *((uint16_t *)(eth_src + 4));

    /* set ETH source address */
    *((uint32_t *)(&ethhdr[6])) = *((uint32_t *)&dst_addr[0]);
    *((uint16_t *)(&ethhdr[10])) = *((uint16_t *)&dst_addr[4]);

    /* get IPv4 header */
    ipv4_hdr = (struct ipv4hdr *)((uint8_t *)
            PRC_GET_SEGMENT_ADDRESS() +
            (uint16_t)PARSER_GET_OUTER_IP_OFFSET_DEFAULT());

    /* store IP source address before changing it */
    ip_src_addr = ipv4_hdr->src_addr;
    /* swap IP source & destination addresses */
    ipv4_hdr->src_addr = ipv4_hdr->dst_addr;
    ipv4_hdr->dst_addr = ip_src_addr;

    /* we do not need to update nor the IP, nor the L4 checksum, because
     * the IP source & destination addresses were swapped and not replaced
     * with other values */

}
```

Service Layer example

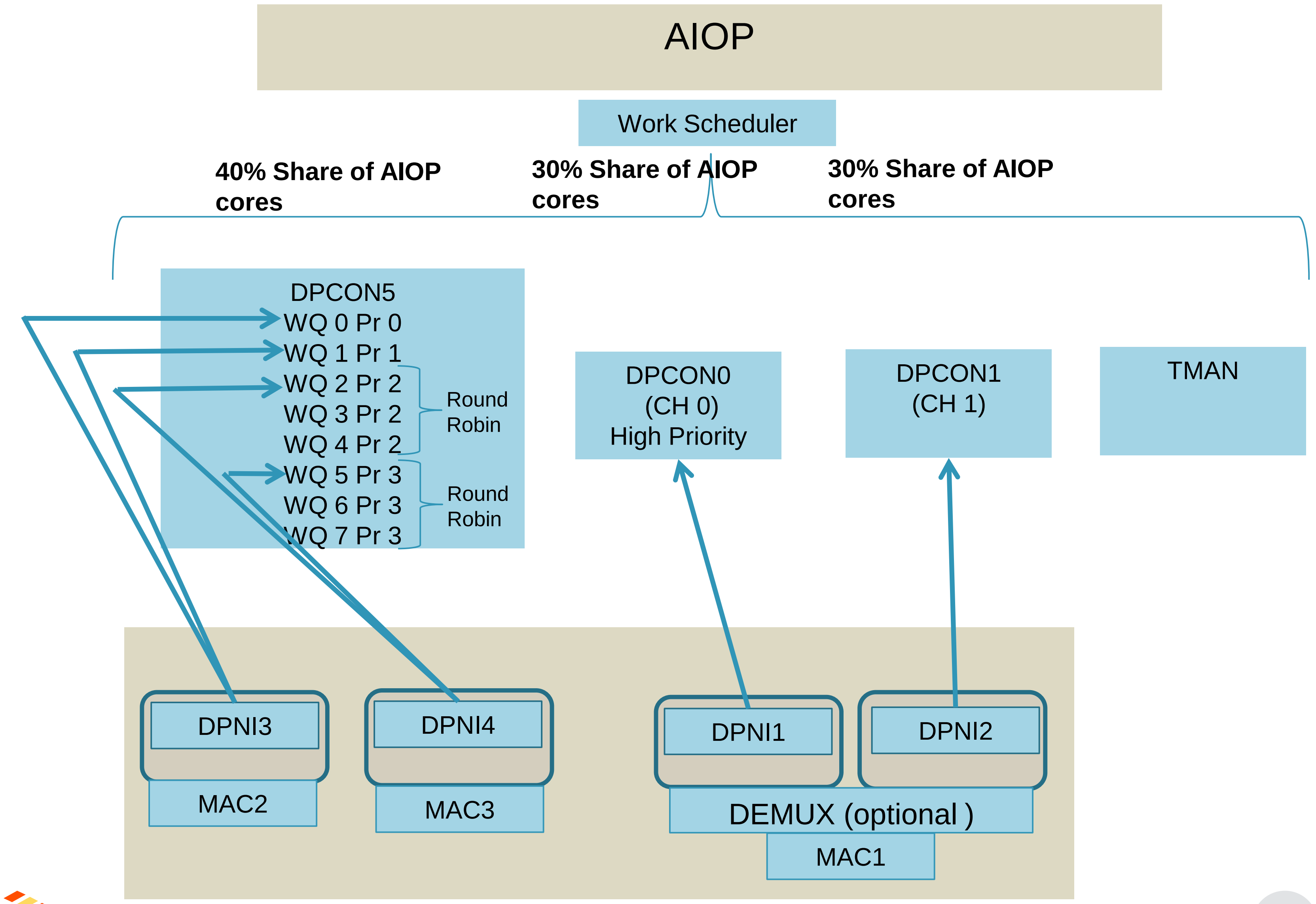


```
inline int fdma_store_and_enqueue_default_frame_qd(
        struct fdma_queueing_destination_params *qdp,
        uint32_t     flags)
{
    /* command parameters and results */
    uint32_t arg1, arg2, arg3;
    int8_t res1;
    /* storage profile ID */
    uint8_t spid = *((uint8_t *) HWC_SPID_ADDRESS);

    /* prepare command parameters */
    flags &= ~FDMA_EN_EIS_BIT;
    arg1 = FDMA_ENQUEUE_WF_ARG1(spid, PRC_GET_HANDLES(), flags);
    arg2 = FDMA_ENQUEUE_WF_QD_ARG2(qdp->qd_priority, qdp->qd);
    arg3 = FDMA_ENQUEUE_WF_QD_ARG3(qdp->qdbin);
    /* store command parameters */
    __stdw(arg1, arg2, HWC_ACC_IN_ADDRESS, 0);
    *((uint32_t *)(HWC_ACC_IN_ADDRESS3)) = arg3;
    /*__stqw(arg1, arg2, arg3, 0, HWC_ACC_IN_ADDRESS, 0);*/

    /* call FDMA Accelerator */
    __e_hwacceli_(FODMA_ACCEL_ID);
    /* load command results */
    res1 = *((int8_t *) (FDMA_STATUS_ADDR));

    if (res1 == FDMA_SUCCESS)
        return SUCCESS;
    else if (res1 == FDMA_ENQUEUE_FAILED_ERR)
        return -EBUSY;
    else if (res1 == FDMA_BUFFER_POOL_DEPLETION_ERR)
        return -ENOMEM;
    else
        fdma_exception_handler(FDMA_STORE_AND_ENQUEUE_DEFAULT_FRAME_QD,
                    __LINE__, (int32_t)res1);

    return (int32_t)(res1);
}

inline void fdma_terminate_task(void)
{
    /* command parameters and results */
    uint32_t arg1;
    /* prepare command parameters */
    arg1 = FDMA_TERM_TASK_CMD_ARG1();
    *((uint32_t *)(HWC_ACC_IN_ADDRESS)) = arg1;
    /* call FDMA Accelerator */
    __e_hwacceli_(FODMA_ACCEL_ID);
}
```

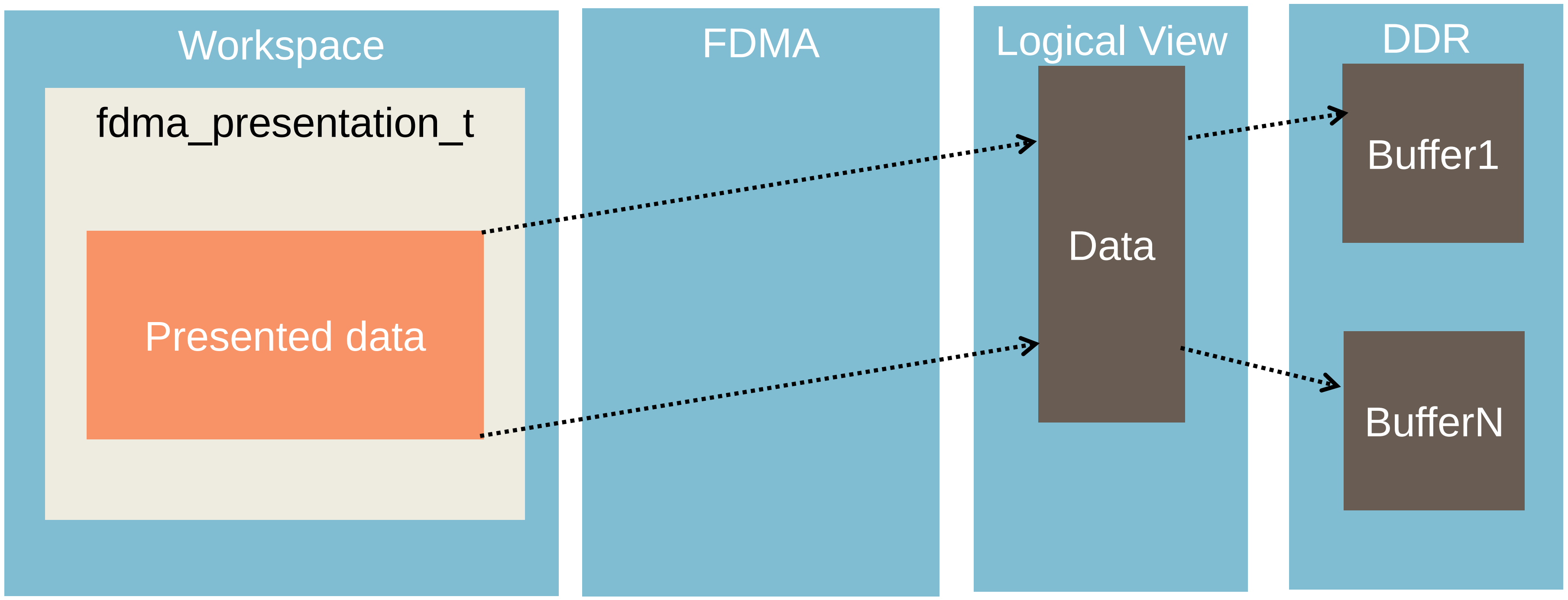
Interfaces and Work Scheduler

- **Work Scheduler role's is to accept work to AIOP**
- **Work scheduling in AIOP supports following features:**
  - Bandwidth partitioning
  - Priority of different traffic within bandwidth partition
  - High priority traffic has special channel
  - Congestion control on queues
- **Implementation:**
  - QMAN Channels for bandwidth isolation
  - Interface is connected to one channel only
  - Intra-channel (WQs) for different CoTs
  - High priority channels for emergency traffic

Work Scheduler role's is to accept work to AIOP



Operations on Frame (FDMA)



- Frame and presentation objects are created in task workspace (local, fast memory) on packet reception
- fdma_presentation_t object contains copy of specific area of frame in workspace and provides methods that allow committing changes made by user on that copy, inserting and deleting of new data. One can insert data into presentation, remove data, write data
- It is possible to have multiple presentations at the same time

Operations on Frame (FDMA)

- Create new presentation in workspace (done automatically for initial frame)

```
int32_t fdma_create_presentation(
        fdma_frame_t* const fh,
        uint16_t initial_offset,
        fdma_presentation_t* presentation)
```

- This function will commit changes that were made in workspace back to frame

```
int32_t fdma_commit_data_to_presentation(
    fdma_presentation_t* fdma_presentation,
    void *starting_address,
    uint16_t size)
```

- Copy and paste data from workspace to presentation

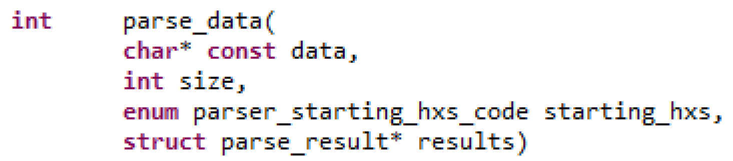
```
int32_t fdma_copy_paste_data_to_presentation(
    fdma_presentation_t *fdma_presentation,
    lcl_addr_t src_addr,
    uint16_t src_size,
    uint16_t dst_offset,
    uint16_t dst_size
)
```

- Close frame and send to frame queue

```
int32_t fdma_close_and_enqueue_frame_to_fqid(
    fdma_frame_t *frame,
    uint32_t fqid,
    uint32_t flags)
```

Operations on Frame (Parser)

- Parses the frame. The parse results updated



```
int          parse_data(
             char* const data,
             int size,
             enum parser_starting_hxs_code starting_hxs,
             struct parse_result* results)
```

DMA operations

- Write from workspace memory to external address

```
void cdma_write(
        uint64_t ext_address,
        void *ws_src,
        uint16_t size);
```

- Read from external memory to workspace address

```
void cdma_read(
        void *ws_dst,
        uint64_t ext_address,
        uint16_t size);
```

```
struct arp_server_ext_info arp_server_ext_info;

cdma_read(&arp_server_ext_info, arp_server->more_info, sizeof(arp_server_ext_info));
```

Table operations

- Lookup an entry based on key

```
int table_lookup_by_key(enum table_hw_accel_id acc_id,
            uint16_t table_id,
            union table_lookup_key_desc key_desc,
            uint8_t key_size,
            struct table_lookup_result *lookup_result);
```

Synchronization

- Mutexes
  - Mutual exclusion
- RCU
  - Garbage collection
- Ordering Scope Manager (OSM)
  - Mutual exclusion and Ordering

Mutexes

- Mutexes are used to prevent race conditions between different tasks using the same resource (e.g. updating/reading the same memory structures in external memory)

- Two types of locks
  - Several readers are allowed in mutually exclusive section and only one writer is allowed:

Updating task – only one task can update variables

```
cdma_mutex_lock_take(mac_addr_ddr, WRITE_LOCK);
cdma_write(&mac_addr, mac_addr_ddr, 6);
cdma_mutex_lock_release(mac_addr_ddr);
```

Reading task - several reading tasks may take lock simultaneously

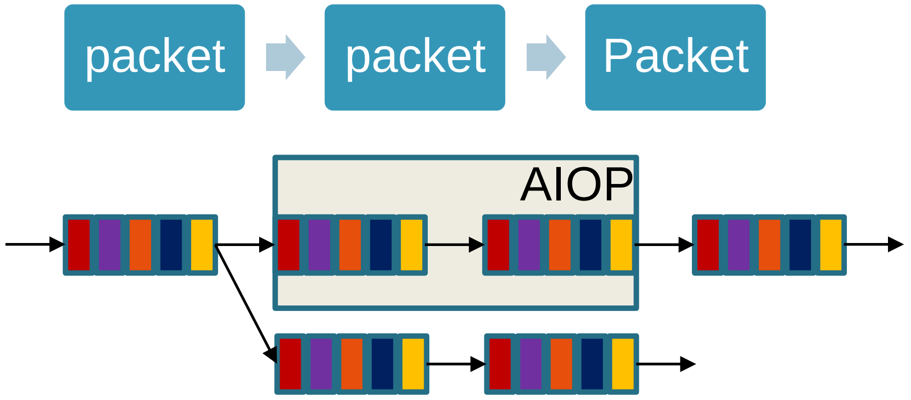
```
cdma_mutex_lock_take(mac_addr_ddr, READ_LOCK);
cdma_read (&mac_addr, mac_addr_ddr, 6);
///Do something with it
cdma_mutex_lock_release(mac_addr_ddr);
```

RCU

- rcu_read_lock(): Marks an RCU-protected data structure so that it won't be reclaimed for the full duration of that critical section. In our case, this function typically done on task creation time.
- rcu_read_unlock(): Used by a reader to inform the reclaimer that the reader is exiting an RCU read-side critical section.
- synchronize_rcu(): It blocks until all pre-existing RCU read-side critical sections on all CPUs have completed.

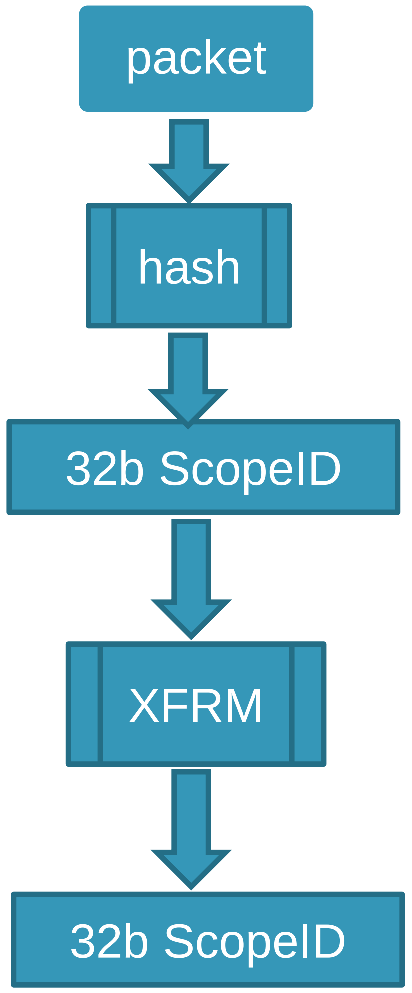
Ordering is about queues, queues cross AIOP

- Some packets pass through AIOP for near wire rate processing
- A queue within AIOP is identified by ScopeID
- AIOP applications manipulate ScopeID to enable concurrency while maintaining order
- Packets which transition through the same queues (ScopeIDs) remain ordered



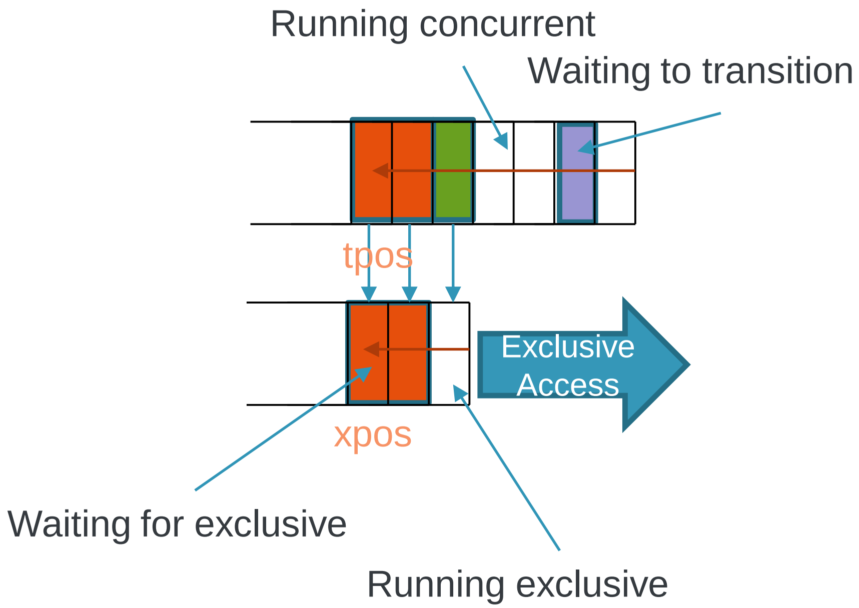
Scope construction/transformation

- Scopes can be a hash of header fields (classification)
  - Initial automatic on arrival to AIOP
- Vocabulary of transformations into new scopes from old scopes
- Or reclassify with new hash
- Software controls “steps” of a program
- Each step is a queue and as always a sequence of queues define order
- Software does not need to know the absolute value, only the step of a flow



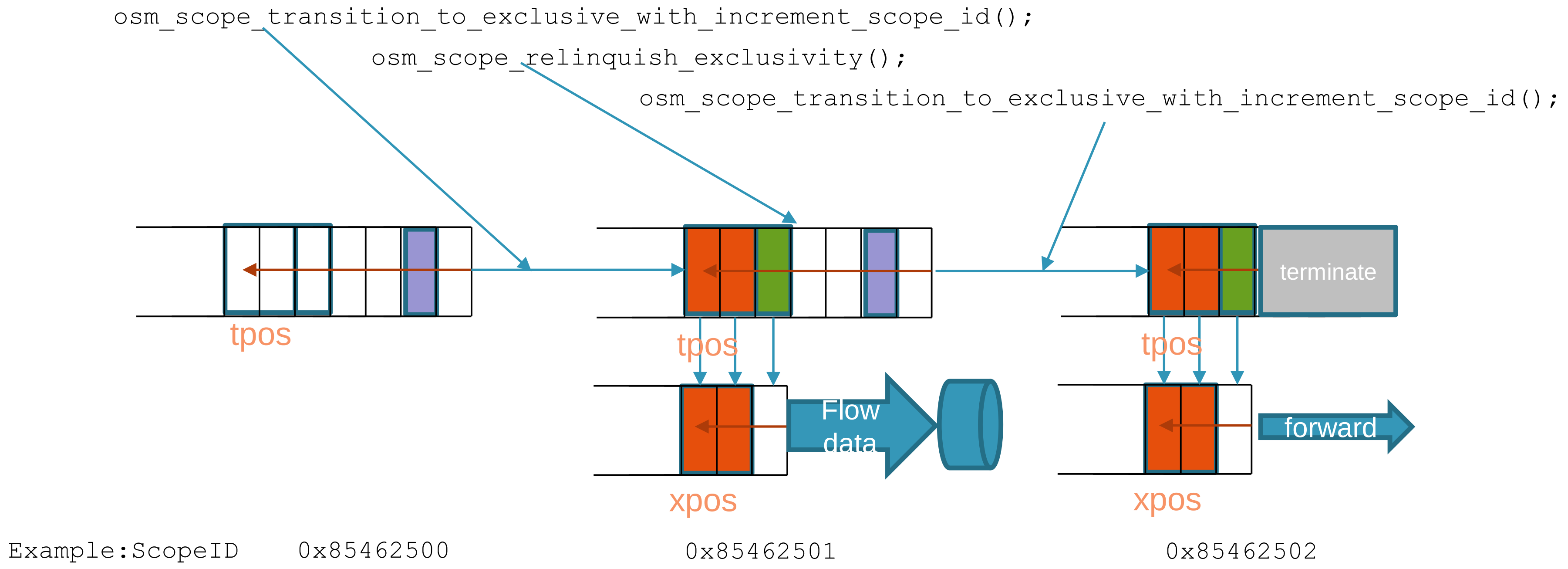
Tasks follow rules with ScopeID (queue)

- A task's position in a queue is called TPOS
- A task's position within a sub-queue for exclusivity is called XPOS
- Tasks may wait for exclusivity (red) or wait to transition to the next queue (purple)
- WX->XX->XC->WT
- All tasks not waiting are running concurrently
- One task knows it is exclusive
- Queues are always FIFO



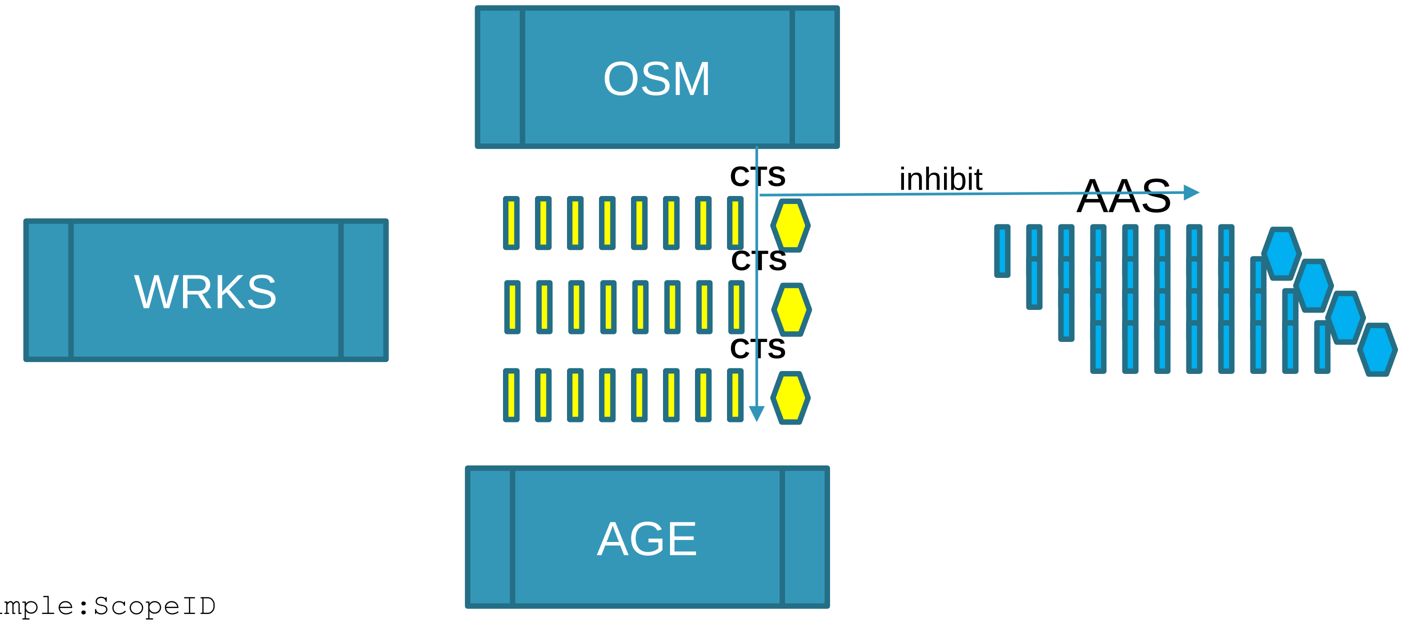
Possible application design

- Start concurrent, TLU access to gather flow data structure
- Transition to exclusive to access/modify flow data
- Relinquish exclusivity to perform remain work (stats perhaps)
- Transition to exclusive to forward packet in order


osm_scope_transition_to_exclusive_with_increment_scope_id();
osm_scope_relinquish_exclusivity();
osm_scope_transition_to_exclusive_with_increment_scope_id();
tpos
tpos
tpos
xpos
xpos
Flow data
terminate
forward
Example:ScopeID
0x85462500
0x85462501
0x85462502

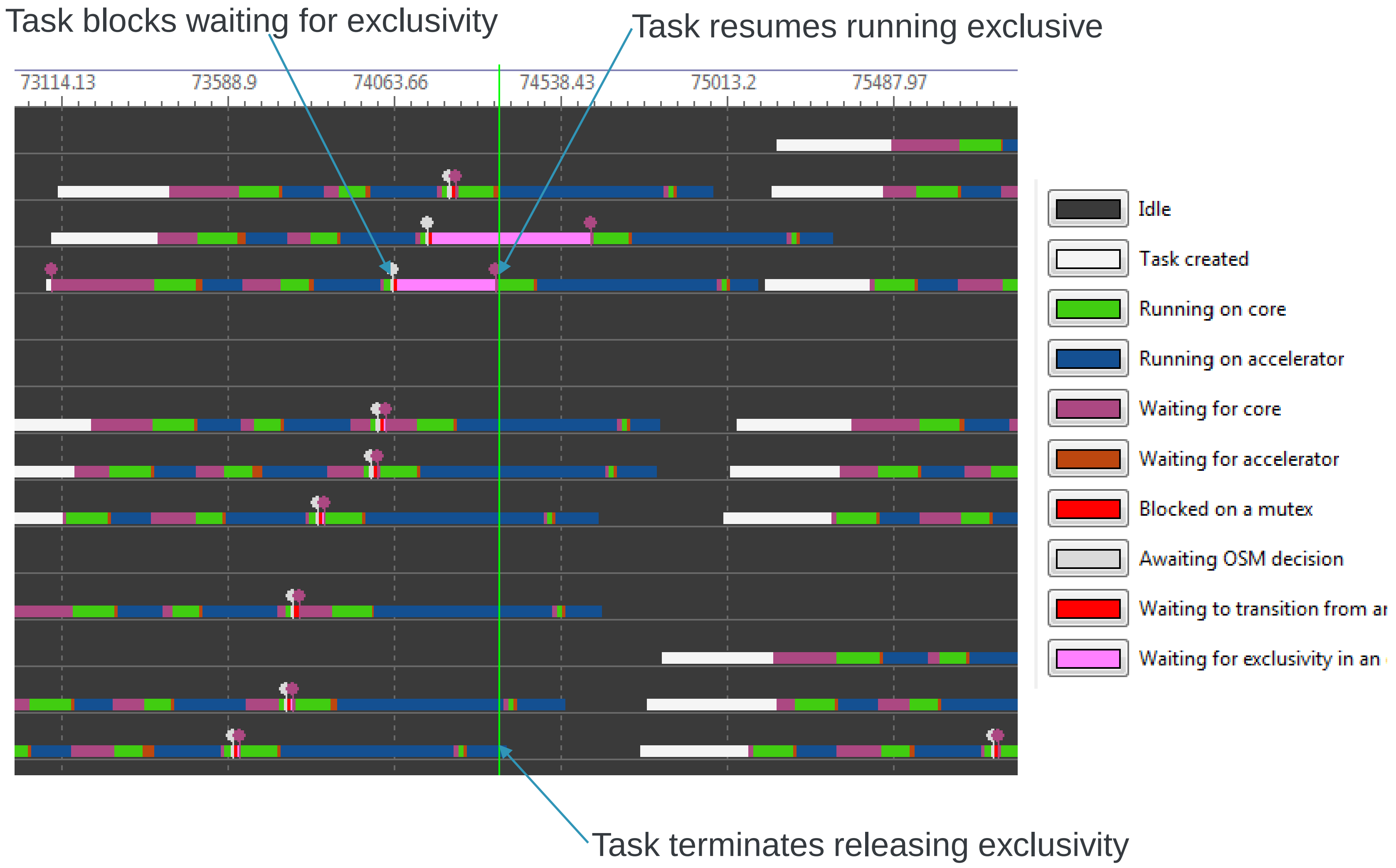
Age priority versus order constraint

- Cores have task slots, WRKS assigns packets to task slots, it is static (no migration)
- Tasks are assigned a global age tracked in a global age record
- Tasks are a sequence of software and hardware jobs to be run by cores or accelerators
- Each core has a scheduler (CTS) to schedule software jobs based on age (priority)
- Accelerators have a single scheduler to schedule hardware jobs based on age (priority)
- OSM tracks order constraints of tasks (illusion of queues) and inhibits them from being scheduled as needed (inhibits scheduling)
  - But software must inform OSM when it wants to move from (ordering) queue to queue within AIOP



Example:ScopeID

Example of Ordering Constraint in Trace



AIOP Packet Processor Details for LS2085A

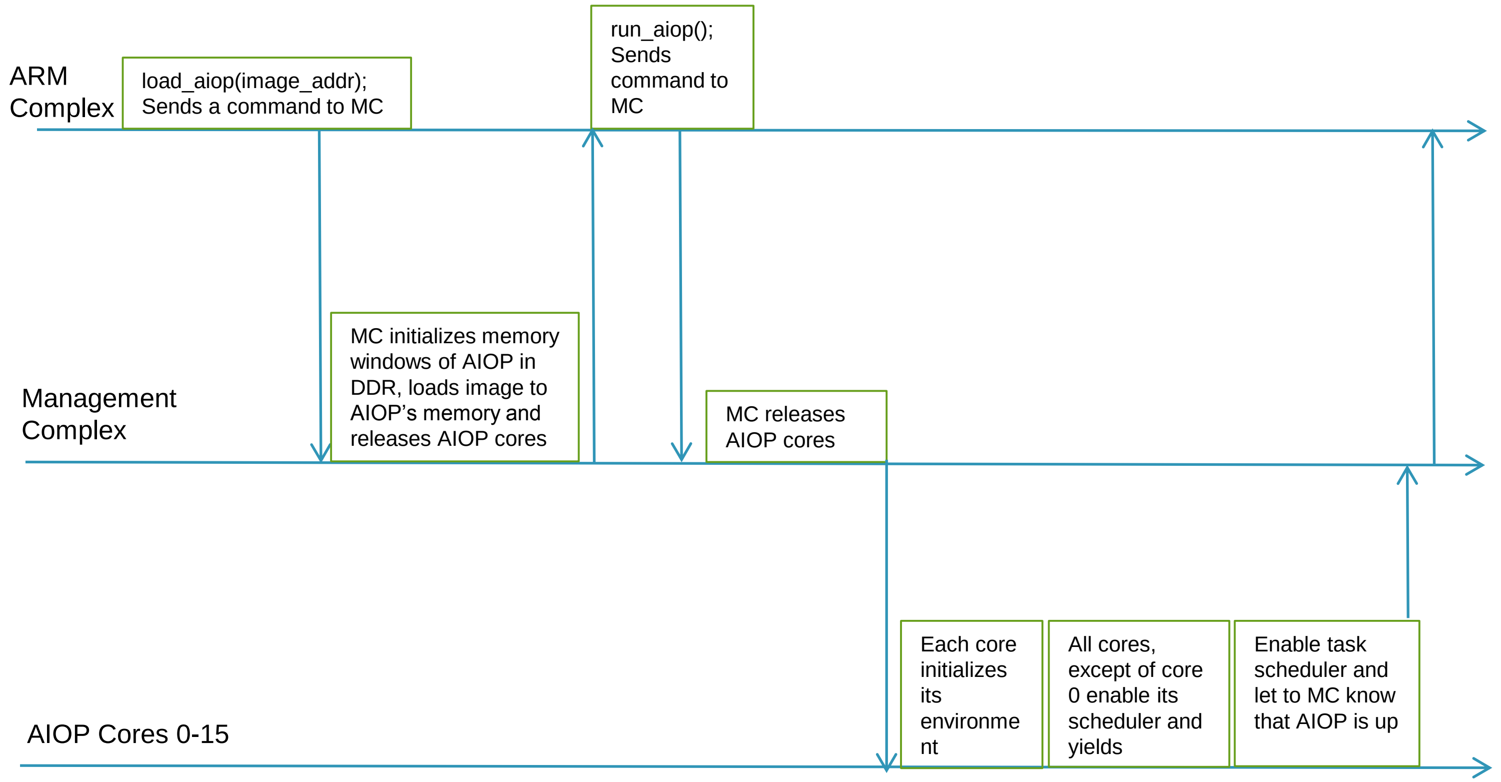
- Cores
  - 16 e200z490 32-bit cores (4 clusters of 4 cores each) with 256 hardware threads
  - 8 KB I-cache per core, 32 KB Workspace SRAM per core partitioned between up to 16 tasks
  - 128 KB shared IRAM per cluster of 4 cores
  - 256 KB Shared SRAM shared across AIOP tile
- Accelerators:
  - Table Look-up Unit, support 17MSPS LPM, 51MSPS EM and 17MSPS ACL (10K rules in PEB) operations concurrently
    - EM key size up to 124B; LPM key sizes, 4 byte EM + 4 byte LPM (IPv4) or 4byte EM + 16 byte LPM (IPv6)
    - ACL key size up to 56B
  - Parse/Classify, 30MOPS
  - Frame DMA,Context DMA, 17MPPS packet presentation/enqueue with 3 context DMA operation combined
  - Timer Manager, Millions of timer supported
  - Stats Engine, 32 in-flight stats command executed concurrently with buffers up to 2K commands.
- Schedulers:
  - Accelerator Scheduler, Work Scheduler, Ordering Scope Manager, per Core Task Scheduler

LS2085 Packet Processing Performance Goals

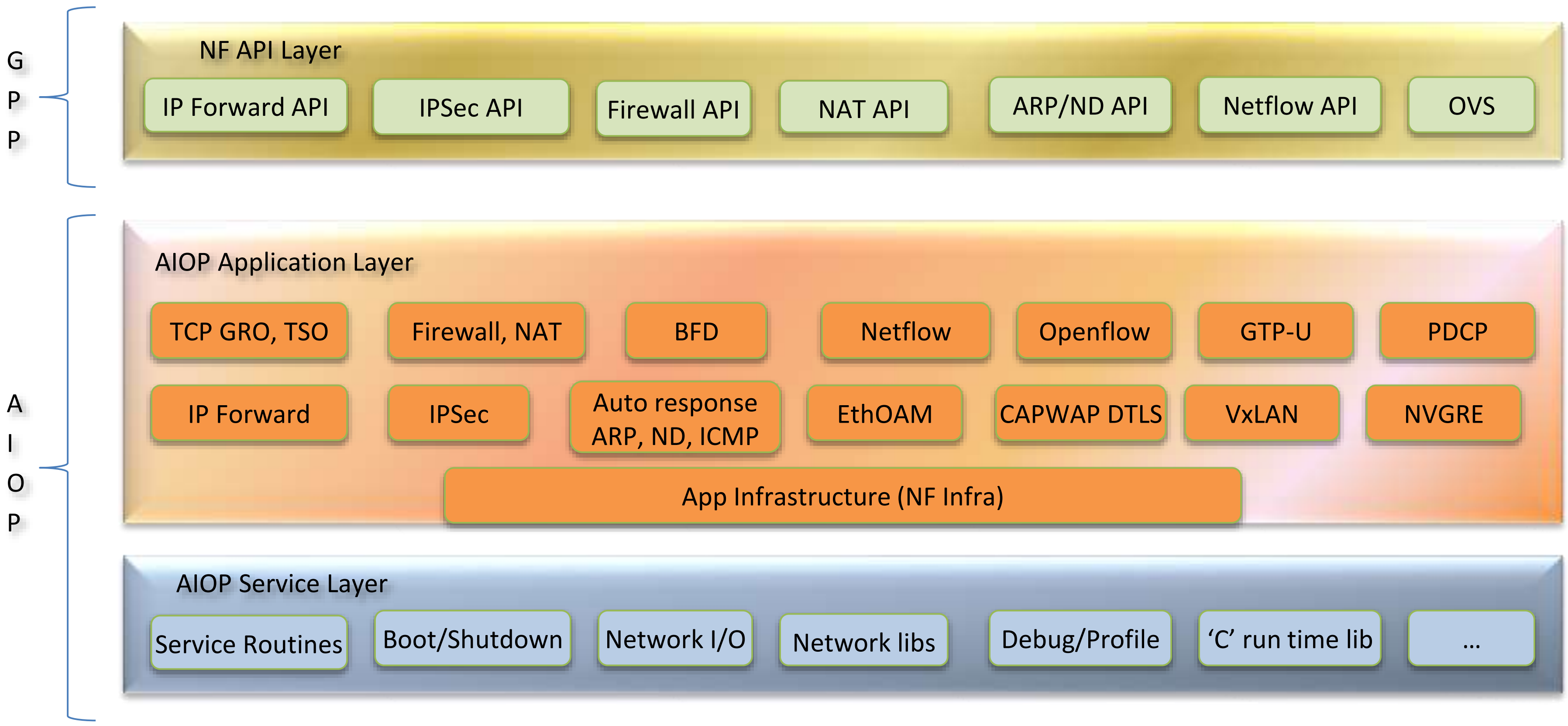
Early performance validation results confirm that LS2085r1 samples meet performance expectations for applications highlighted in green.
The other applications are currently tested in the lab.

Use-cases / Benchmarks	LS2085 Target
Complex Fwd Packet Processing • 10K Algorithmic Access Control List (ACL) Rules • 5 classification stages per frame 1. Virtual Interface Lookup (exact match - EM) 2. IP SA Spoof Check (EM) 3. Policy Based Routing – Access Control List (ACL) 4. IP Forward - Longest Prefix Match (LPM) 5. ARP Table Lookup (EM)	20Gbps @128B Packet Size (on track for production silicon)
L2 Switch – Physical	80Gbps @64B Packet Size (on track for production silicon)
Netflow (IPFIX) Packet Processing	20Gbps @ 128B
Simple IP Fwd	20Gbps @64B Packet Size (on track for production silicon)
Simple IPSec Fwd	15Gbps @ 390B

AIOP boot process

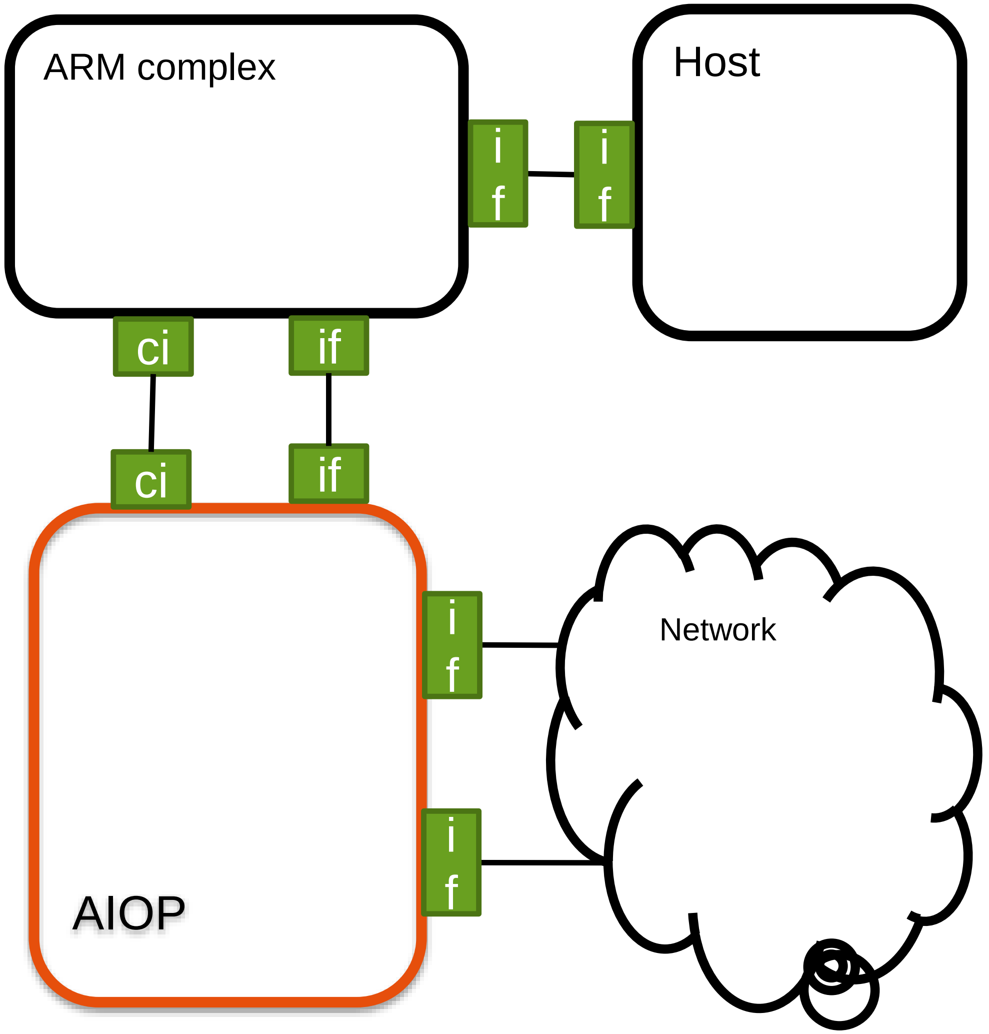

ARM Complex
load_aiop(image_addr); Sends a command to MC
run_aiop(); Sends command to MC
Management Complex
MC initializes memory windows of AIOP in DDR, loads image to AIOP's memory and releases AIOP cores
MC releases AIOP cores
AIOP Cores 0-15
Each core initializes its environme nt
All cores, except of core 0 enable its scheduler and yields
Enable task scheduler and let to MC know that AIOP is up

AIOP Software: Building Blocks

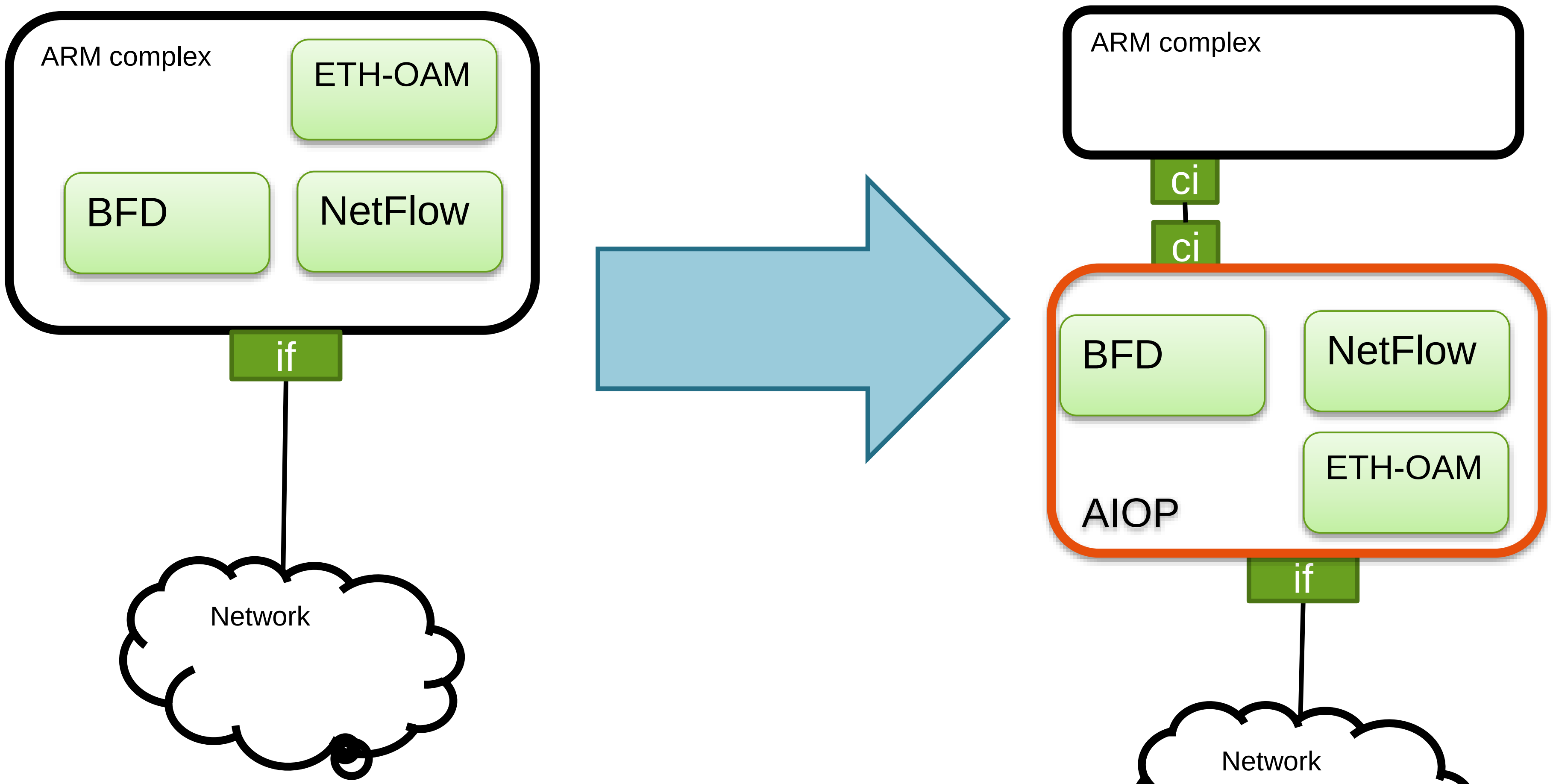


AIOP integration – depends on specific application !

- AIOP – network node with network interfaces and command control interfaces.
- AIOP is generally easier to program for efficient packet processing then GPP
- AIOP program looks and acts very similar to general purpose processor program (“sequential” ‘C’)



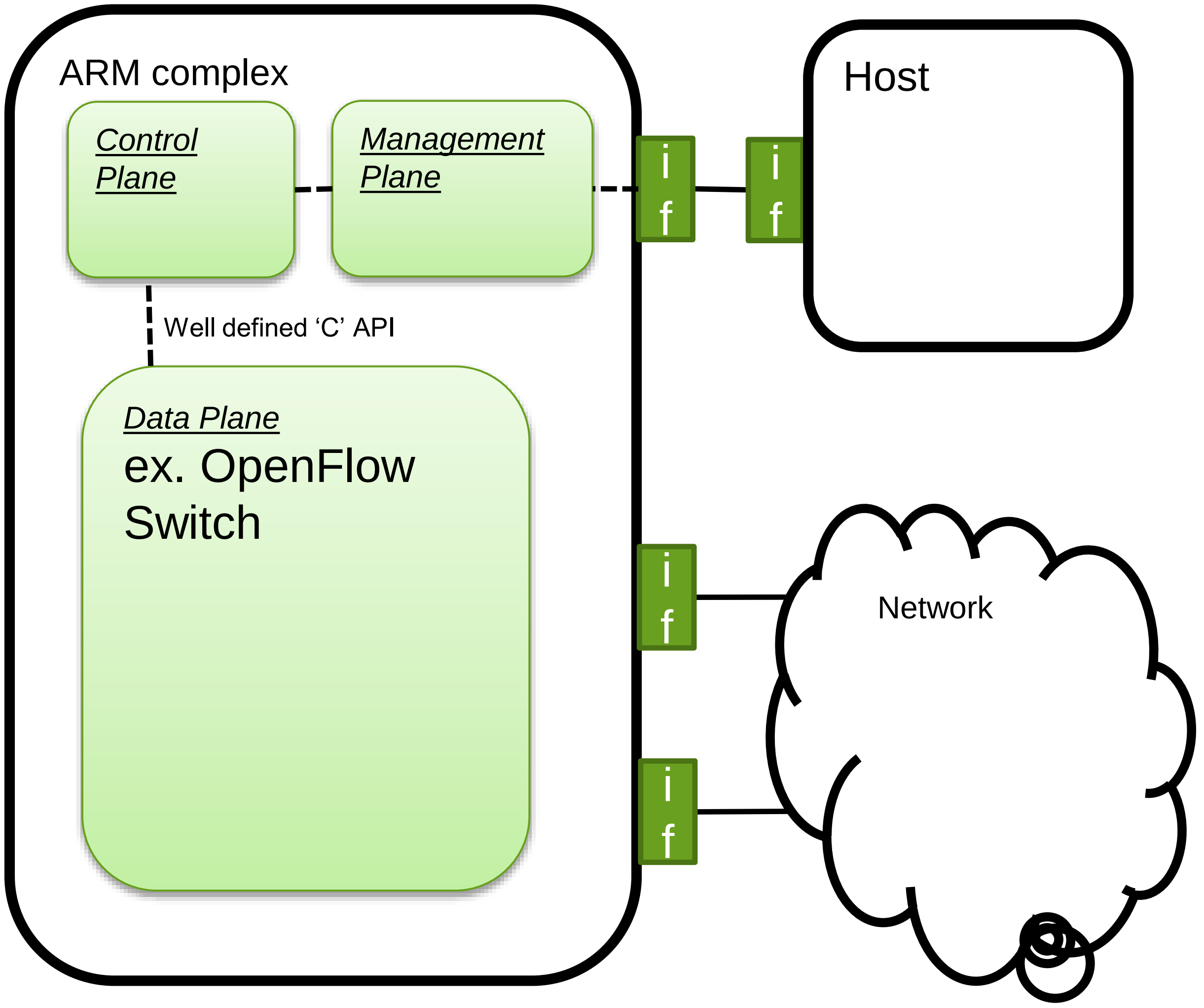
Specialized “Network Application” - Monitoring



- Some network application that is typically executing on host but is not part of normal packet processing.
- Examples include monitoring, ARP server etc.

Management/Control and Data plane Architecture

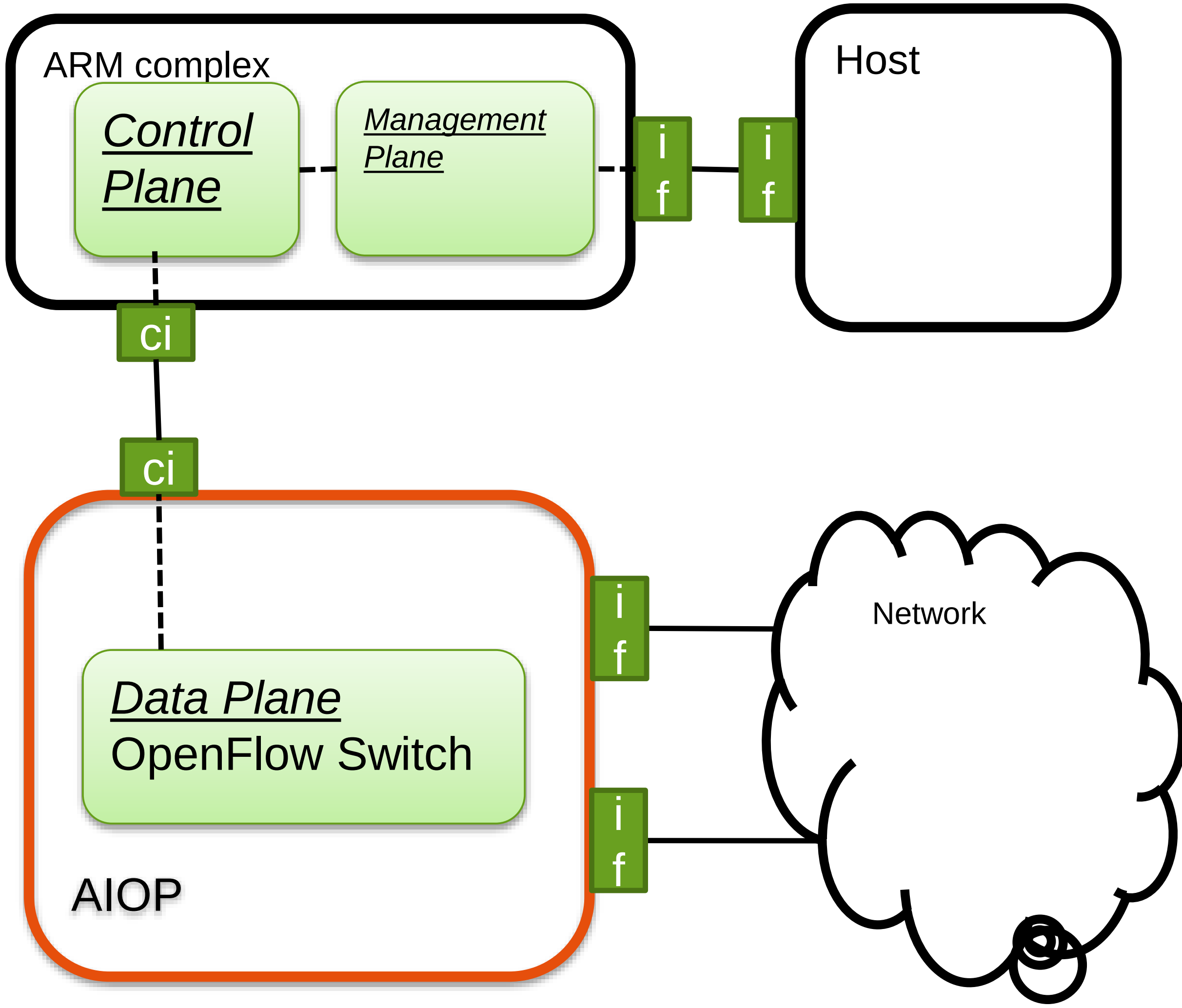
- Forwarding plane is decoupled from control plane thru well defined API
- Examples: OpenFlow Switch



#FTF2015

Management/Control in ARM and Data plane in AIOP

- AIOP is built especially to make easy and efficient implementation of data plane
- It is possible to retain the same API between Control Plane and Data Plane



AIOP Enablement options

	Use Freescale Integration	Adjust Freescale Integration	Integrate by yourself	
Use MKT ready apps	✓	✓	✓	AIOP is "black Box"
Adjust Freescale applications*	✓	✓	✓	AIOP is "white box"
Write your own	✓	✓	✓	AIOP is your own

* Freescale services organization may help with this work

www.Freescale.com