YVR18-217: Lightweight flows for fine-grain packet order processing

(Flow Aware Scheduler)

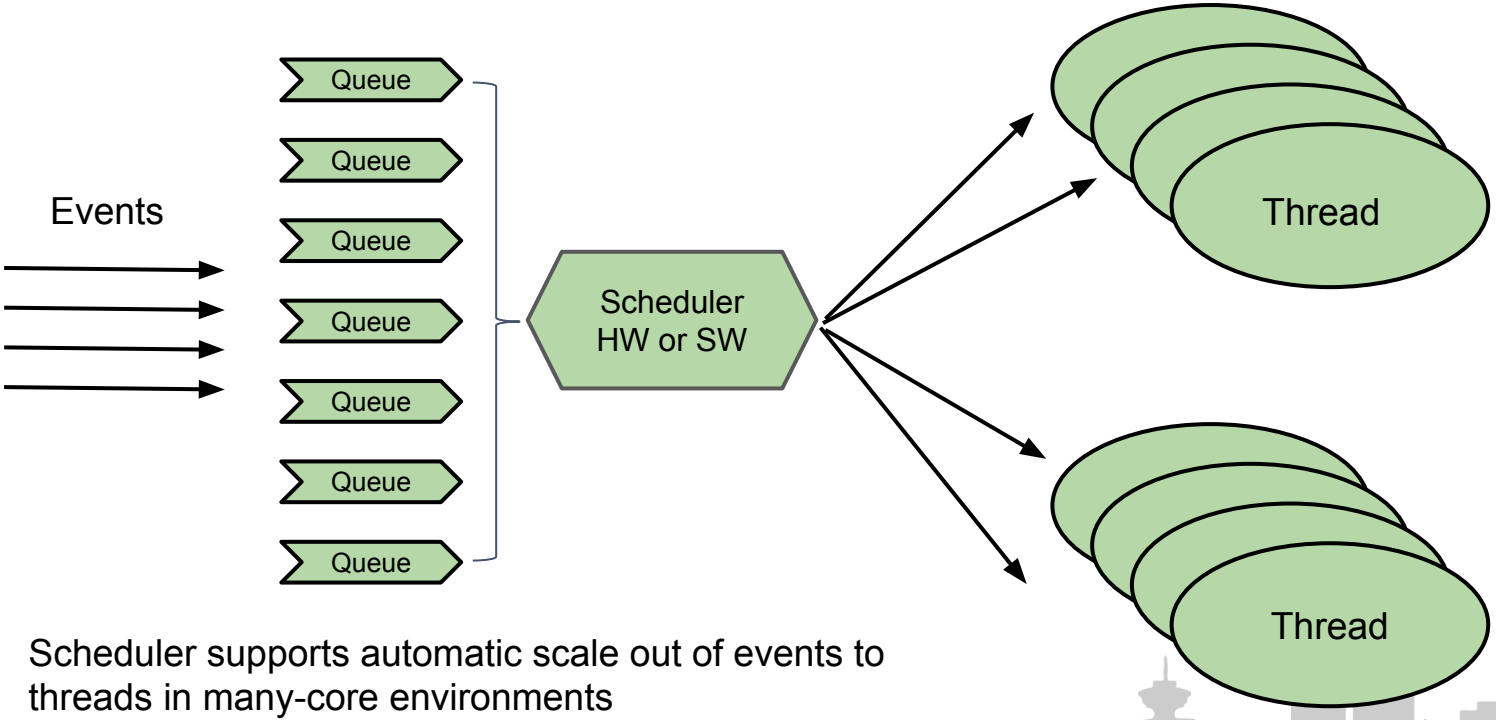
Balasubramanian Manoharan
Bill Fischofer

Disclaimer

This is work in progress. The final design and APIs shown here may change before these are released as part of ODP.

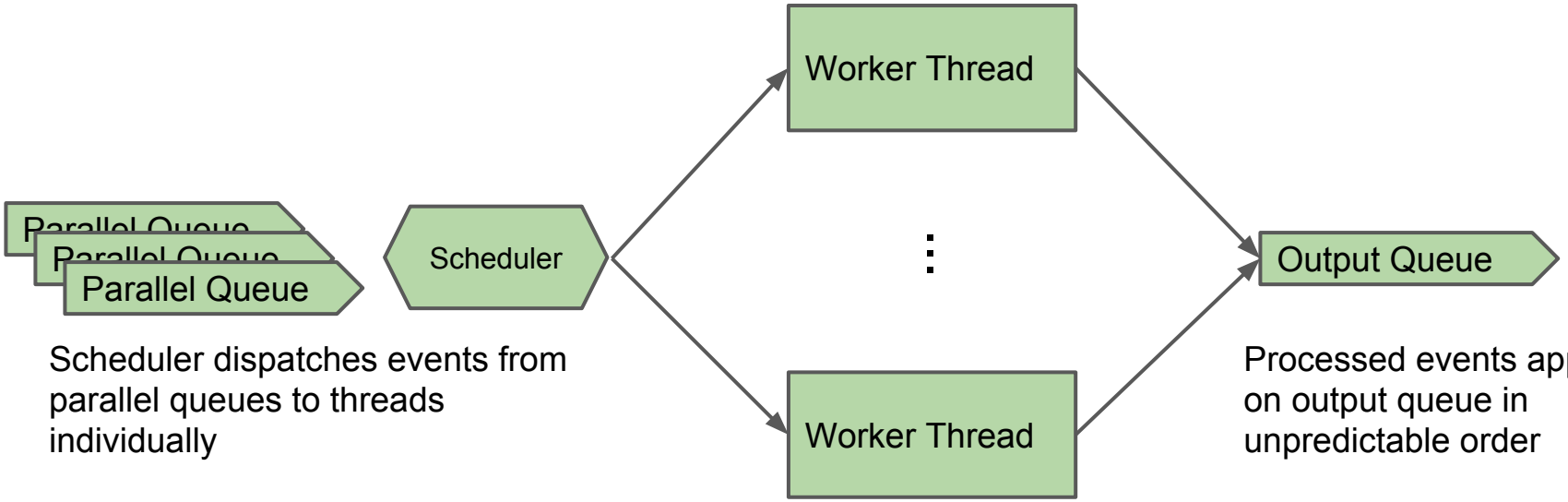
We'd like to get your feedback, either here or on the ODP mailing list (lng-odp@lists.linaro.org)

ODP Scheduler Overview



- Scheduler supports automatic scale out of events to threads in many-core environments
- Events reside on queues that the scheduler scans
- Queues provide both event sequencing and context
- Parallel, Atomic, or Ordered queues

Parallel Queue Event Processing

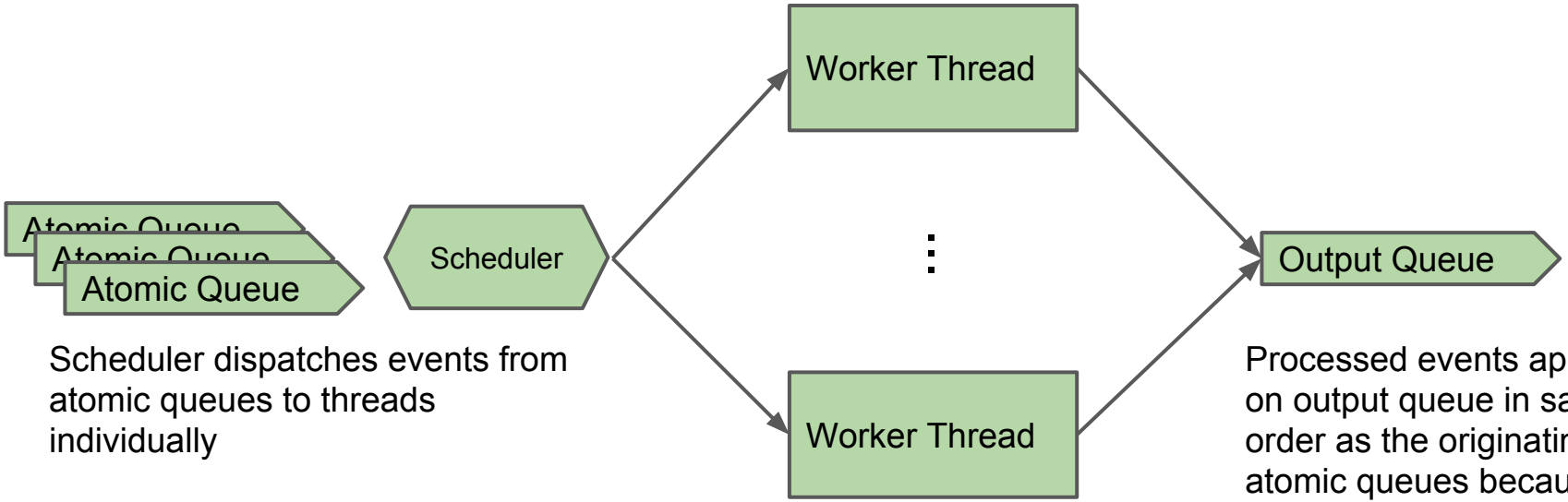


Scheduler dispatches events from parallel queues to threads individually

Worker threads process events in parallel, any synchronization needed among events is application responsibility

Processed events appear on output queue in unpredictable order

Atomic Queue Event Processing

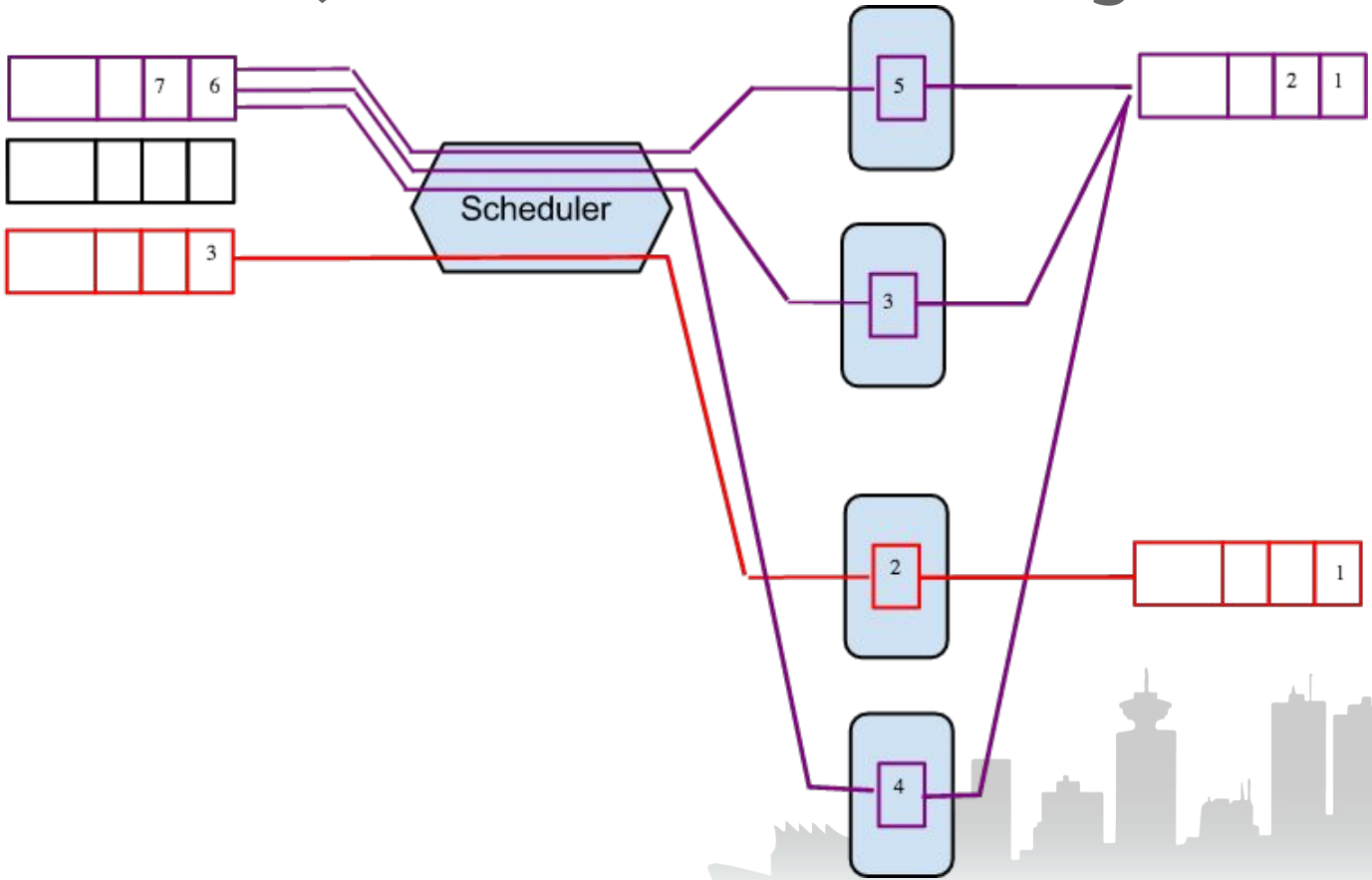


Scheduler dispatches events from atomic queues to threads individually

Worker threads process events in parallel, scheduler ensures no two threads can process events from the same atomic queue at the same time

Processed events appear on output queue in same order as the originating atomic queues because scheduler has serialized them

Ordered Queue Event Processing



Elements of ordered queues are dispatched to next “node” (HW block or thread) in parallel

Output is reordered if necessary so that output queues maintain same relative order as input queues

Simplified Worker Thread Structure

```
void worker_thread(...) {
    odp_init_local(ODP_THREAD_WORKER)  /* And other init processing */
    while (1) {
        ev = odp_schedule()                /* Get next event to be processed */
        ...process work in parallel with other threads
        odp_schedule_order_lock()          /* Enter ordered critical section */
        ...critical section processed in order
        odp_schedule_order_unlock()        /* Exit ordered critical section */
        ...additional work processed in parallel with other threads
        odp_queue_enq(queue, ev)           /* Send event to next processing stage */
    }
}
```

Optional, for ordered queues

Issues with current ODP scheduler design

Resource Issues

- Synchronization limited by the total number of queues
- Each ODP queue contains a queue context
- Creating millions of queues will create memory constraints in many environments

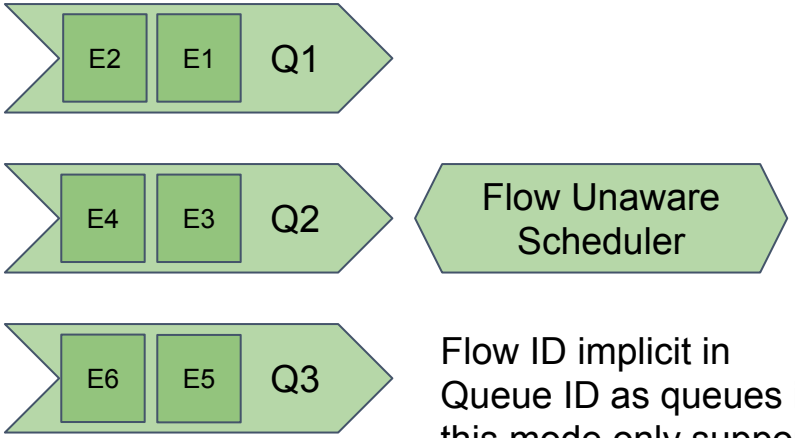
Functional Issues

- Flow concept is tied to queues
- Want to associate flows with individual packets and allow flow identity to change in response to decapsulation, decryption, etc. during processing

Lightweight Flows

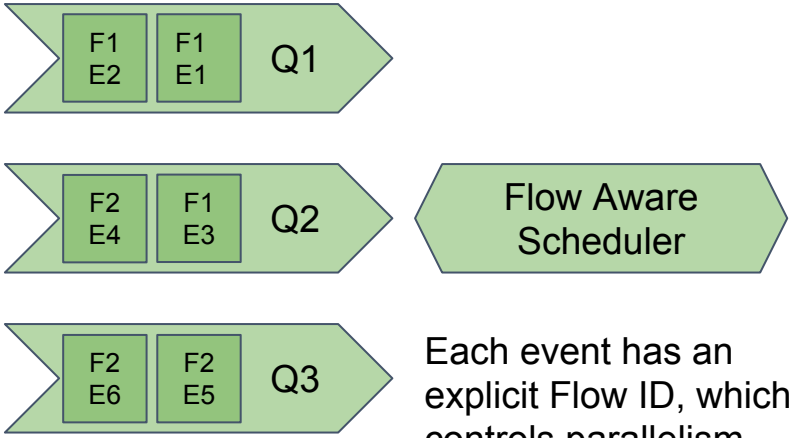
- Lightweight flows are similar to an ODP queue without a context
- Events can be assigned to a specific flow by application before enqueuing onto a scheduled queue
- Event synchronization (PARALLEL, ATOMIC, or ORDERED) is performed at the flow level
- When an event is received from the wire, Initial flow id is generated by the ODP implementation
- Event flow id is enforced only when the event is enqueued to the scheduled queue
- Supports Backward compatibility with the existing ODP scheduler

Flow Aware ODP Scheduler design



Flow ID implicit in Queue ID as queues in this mode only support one flow

All events on same queue belong to same flow



Each event has an explicit Flow ID, which controls parallelism within a queue

Queues can support multiple flows concurrently

Synchronization is controlled by QueueID || FlowID

Scheduler modifications and new APIs

Scheduler capability - `odp_schedule_capability()`

- Platform provides the supported maximum flow and queue count

Scheduler configuration - `odp_schedule_config()`

- Scheduler can be configured to be either flow-aware or (by default) flow-unaware
- Flow-unaware scheduler is same as existing ODP scheduler
- Application configures the number of queues and flow required
- Scheduler configuration has to be done before queue configuration

Scheduler modifications - continued

Scheduler start - `odp_schedule_start()`

- Scheduler start function starts receiving events
- Scheduler configuration can not be modified after start
- If scheduler is not configured, scheduler is inherently configured to default during queue configuration
  - This provides backward compatibility to existing applications

Event Flow ID Mgmt - `odp_event_flow_id(), odp_event_flow_id_set()`

- Enables event flow IDs to be queried or set
- Initial flow ID set by implementation

Flow-aware worker thread structure

```
void flow_aware_thread(...) {
      odp_init_local(ODP_THREAD_WORKER) /* And other init processing */
      while (1) {
            ev = odp_schedule()                    /* Get next event to be processed */
            flow_id = odp_event_flow_id(ev)    /* Get flow ID associated with ev (optional) */
            ...processing                          /* Process the event */
            odp_event_flow_id_set(ev, newid) /* Set new flow ID for event (optional) */
            odp_queue_enq(queue, ev)           /* Send event to next stage */
      }
}
```

Expected Benefits

Finer grained control

Better fit to many applications and resource-constrained platform environments

Better compatibility with Event concepts being introduced in DPDK

Q&A

Questions?


Linaro
connect
Vancouver 2018
Thank you!