The Use of Java in Online Event Building and Recording at Jefferson Lab

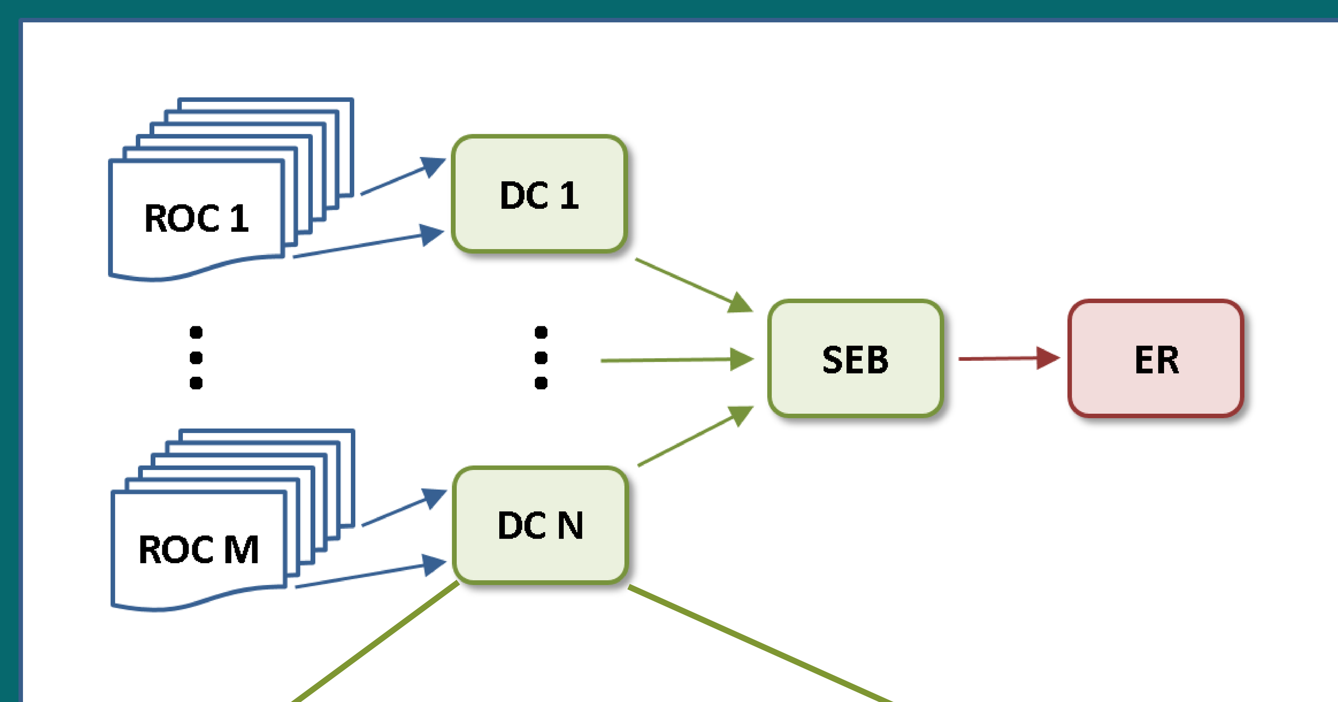
Carl Timmer, D. Abbott, W. Gu, V. Gyurjyan, G. Heyes, E. Jastrzembski, B. Moffit
Thomas Jefferson National Accelerator Facility, Newport News, VA 23606
timmer@jlab.org

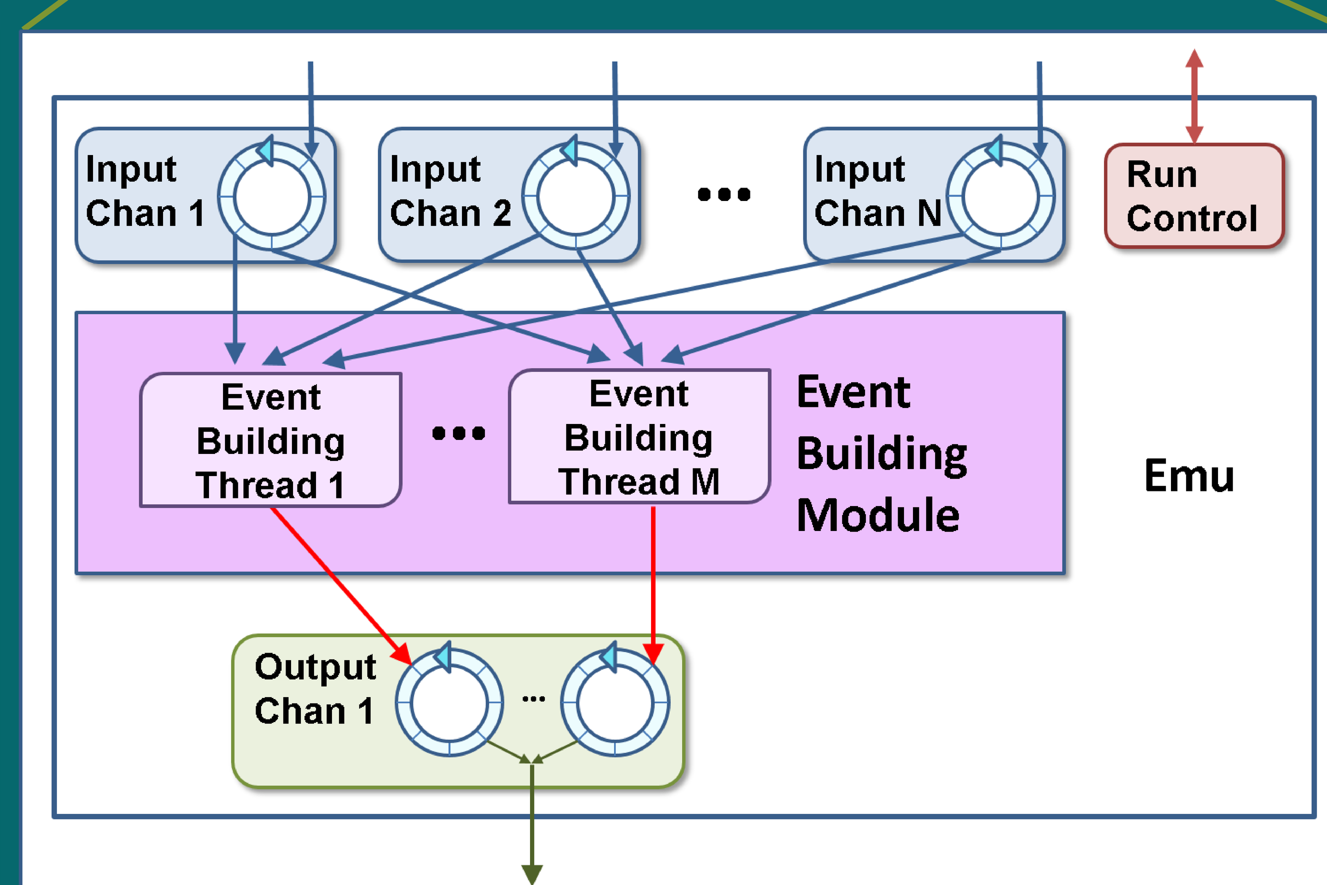
References
http://lmax-exchange.github.io/disruptor

Why JAVA?

- Highly portable
- Short development time
- Easily comprehensible
- Easily maintainable
- Fairly fast I/O
- Event building requires little compute intensive code
- Excellent IDEs and profiling tools available at no cost

DAQ layout





Good programming techniques

- Avoid locks
- Minimize creation of objects
- Reuse objects
- Avoid queues like the plague
- Replace queues with Disruptor's ring buffers.

Don't use queues

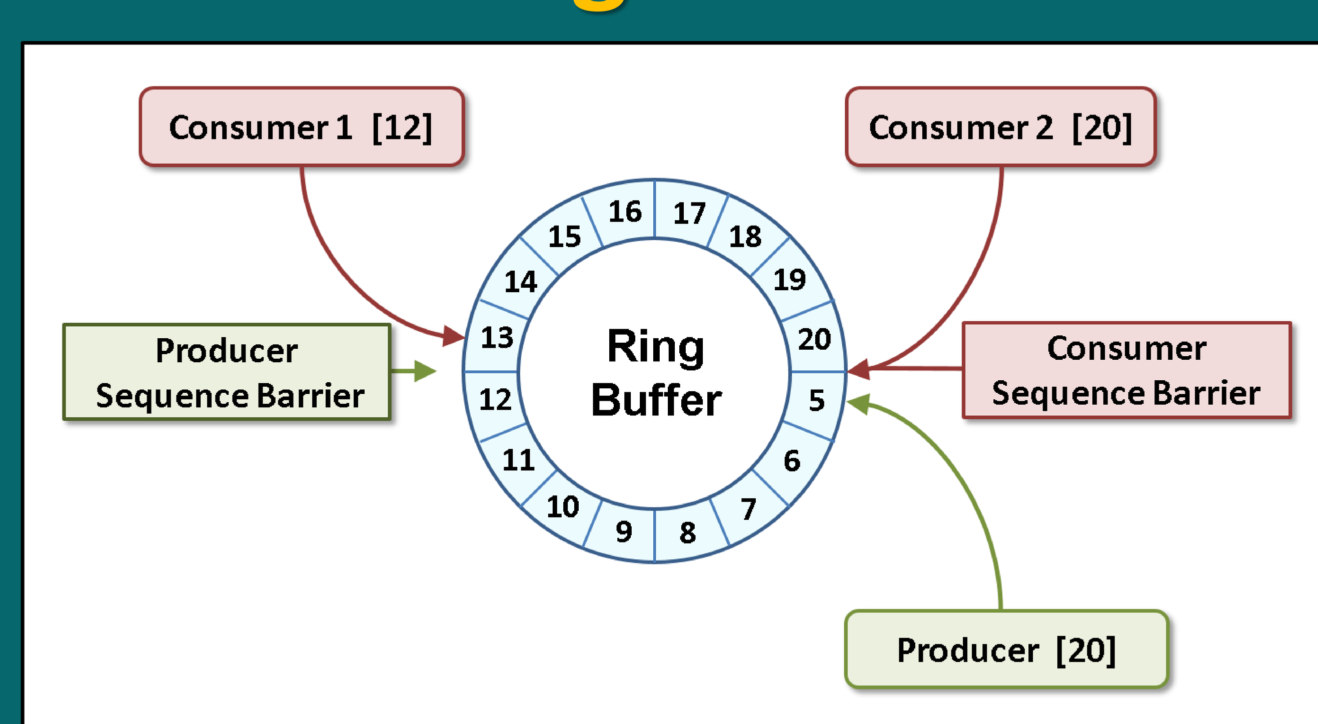
- Either full or empty. When empty, producers and consumers contend. When full, all consumers contend. All operations contend over setting size.
- Entries must be allocated, inserted, removed, and garbage collected
- Head, tail, and size often on same cache-line
- Backing by linked-list is slow and garbage generating.
- Under heavy load, queues may fill causing application to slow so entries live longer and are copied to older generation space. Eventual collection from old generation is expensive and may lead to "stop-the-world" pauses to unfragment memory.

	Array Blocking Queue (ns)	Disruptor (ns)
Min Latency	145	29
Mean Latency	33,000	52
99% less than	2,100,000	128
99.99% less than	4,200,000	8,200
Max Latency	5,100,000	176,000

Don't use locks

Method	Time (ms)
Single thread	300
Single thread with volatile write	4,700
Single thread with CAS	5,700
Two threads with CAS	30,000
Single thread with lock	10,000
Two threads with lock	224,000

Ring buffer



EB Performance

- Simulating a DC: 12 ROCs at 100 MB/s (1.2 GB/s) to 1 EB is easily handled with 4 cores.
- Simulating an SEB: 5 ROCs at 350 MB/s each (1.75 GB/s) to 1 EB is handled with 6 cores.
- Old Queues: **> 40% of cpu time**
- New Ring buffers: **0.6% of cpu time**

Use Disruptor's ultra-fast ring buffer

- Ring entries allocated once as array and are containers which are never added or removed. Arrays work well with caching.
- For single producer, access has *NO* locks, CAS or contention. Uses only memory barriers.
- Written to eliminate false sharing (cache friendly)
- Operates in a "batch" mode, in which consumer asks for next entry, but gets access to all available entries without any more concurrency operations.

Consumer waiting strategies

- 1) spin, 2) spin then yield, 3) spin, yield, then sleep, 4) timeout, 5) block and spin on waking, 6) spin, block, then spin on waking (spin-block)
- Each ring created with 1 strategy
- Different strategies can greatly affect performance
- For simulation with 11 ROCs at 32 MB/s each to 1 DC, spin-yield used 15.8 cores, spin-block used 2.7 cores and performed better!