BigQueue
A Big, Fast and Persistent Queue

- by William
http://bulldog2011.github.com/

Feature Highlights

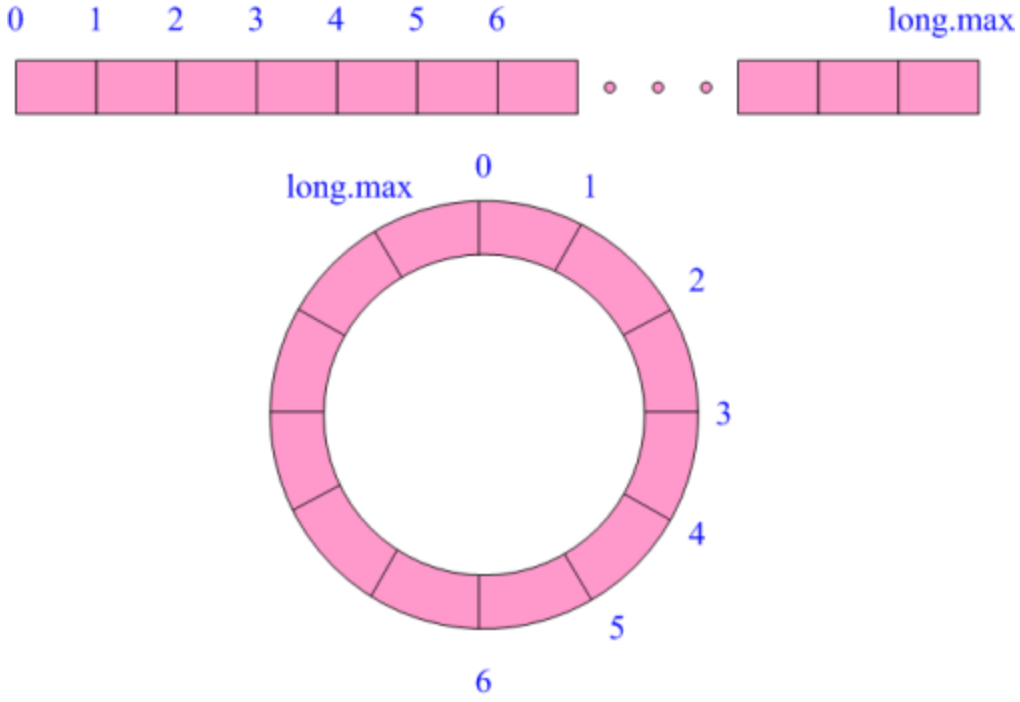
- **Fast**
  - enqueue and dequeue are close to O(1) direct memory access.
- **Big**
  - Only limited by available disk space.
- **Persistent**
  - Data is persisted on disk and is crash resistant.
- **Reliable**
  - OS will be responsible for message persistence even your process crashes.
- **Realtime**
  - Produced messages will be immediately visible to consumers
- **Flexible Queue Semantics**
  - Consume once queue, fanout queue, can even consume by index
- **Memory-efficient**
  - Automatic pagging & swapping algorithm, only most recently accessed data is kept in memory.
- **Thread-safe**
  - Multiple threads can concurrently enqueue and dequeue without data corruption.
- **Simple & Light-weight**
  - Current library jar is less than 40K.

Performance Highlights

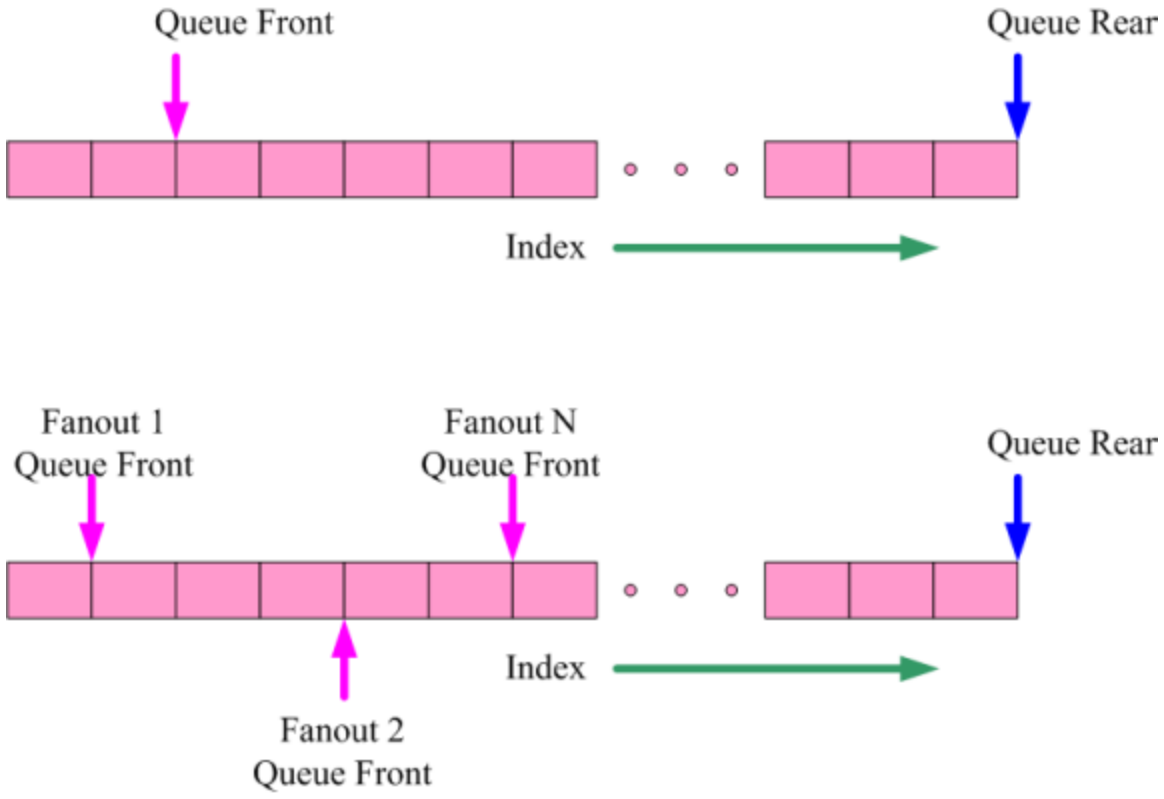
- In concurrent producing and consuming case, average throughput is around ***166MBps***.
- In sequential producing then consuming case, average throughput is around ***333MBps***.

Design – Logical View

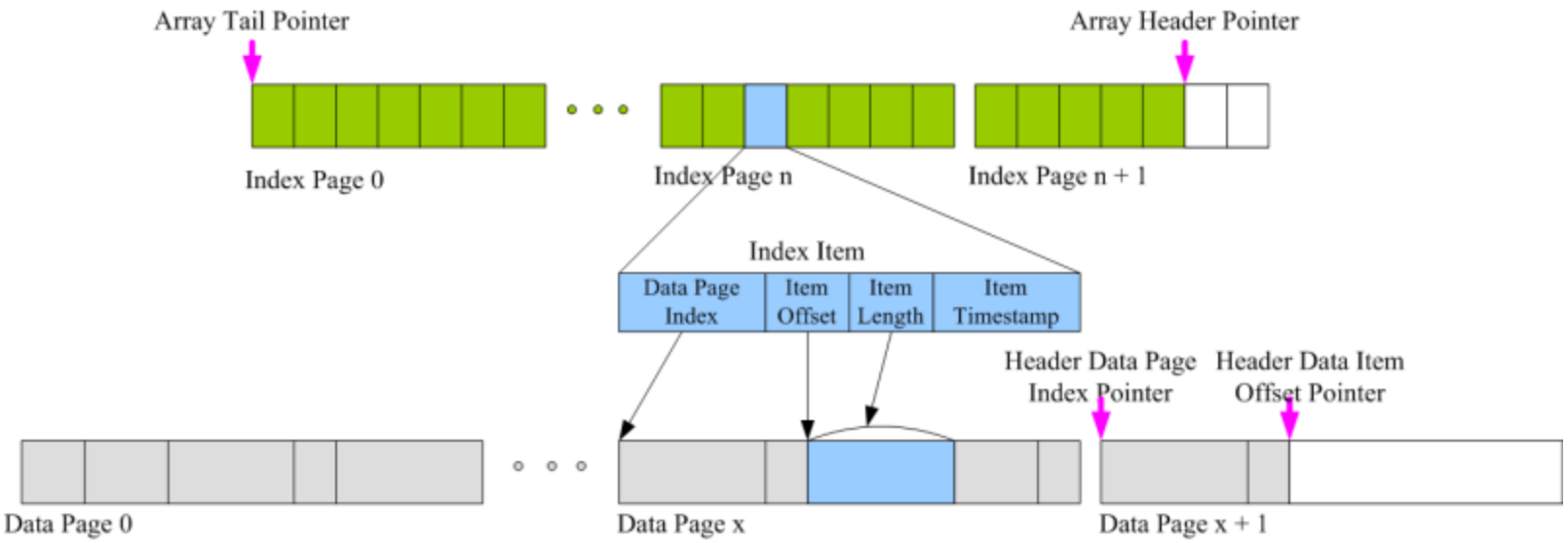
- Looks just like a big array or a circular array



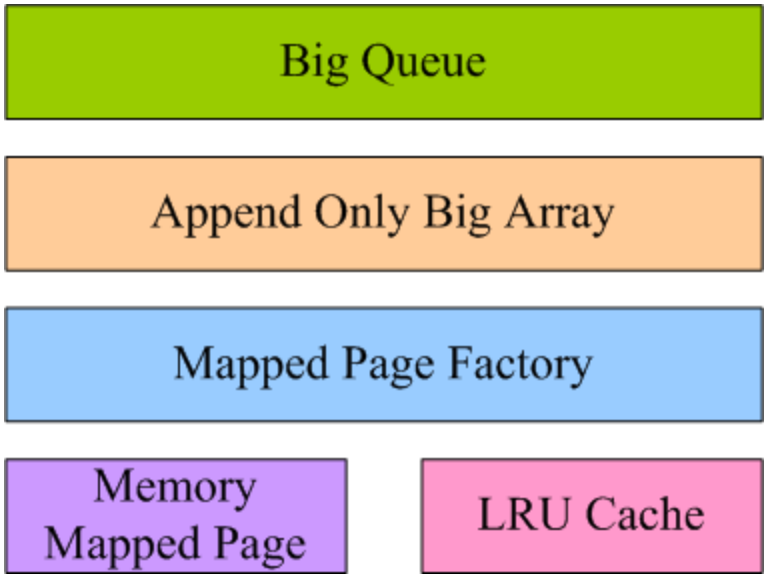
Design – Consume Once and Fanout Semantics Support



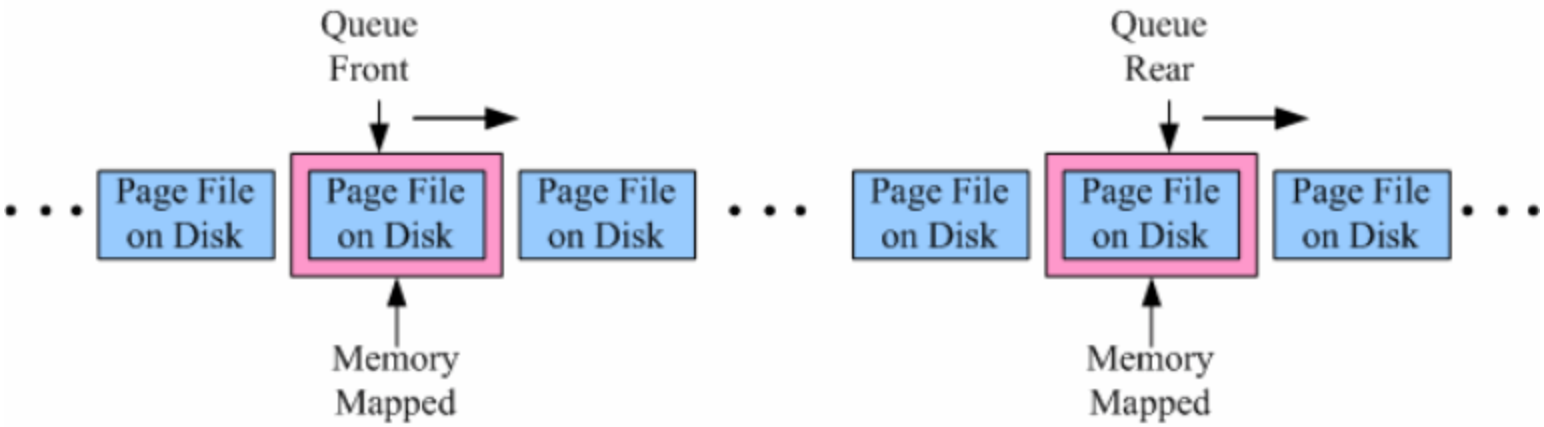
Design – Physical View : Paged Index + Data File



Design – Components View



Design – Dynamic View : Memory Mapped Sliding Window



Concurrency

- Produce(or append) is synchronized in the queue implementation
- Consume(or read) is already thread-safe

Simple Interface

- Creation

```
IBigQueue bigQueue = new BigQueueImpl("d:/bigqueue/tutorial",
"demo");
```

- Enqueue

```
for(int i = 0; i < 10; i++) {
    String item = String.valueOf(i);
    bigQueue.enqueue(item.getBytes());
}
```

- Dequeue

```
for(int i = 0; i < 5; i++) {
  String item = new String(bigQueue.dequeue());
}
```

- Peek

```
byte[] data = bigQueue.peek();
```

Fanout Queue

- Creation

```
IFanOutQueue foQueue = new FanOutQueueImpl("d:/tutorial/fanout-queue", "demo");
```

- Enqueue

```
for(int i = 0; i < 10; i++) {
   String log = "log-" + i;
   foQueue.enqueue(log.getBytes());
}
```

- Fanout 1 Dequeue

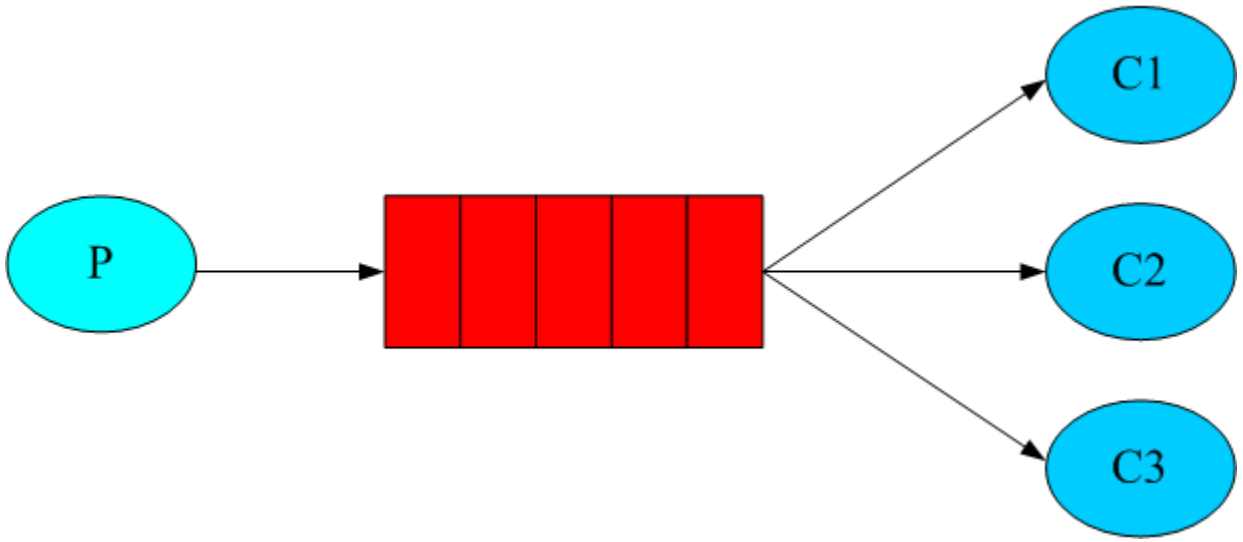
```
String fanoutId1 = "realtime";
while(!foQueue.isEmpty(fanoutId1)) {
    String item = new String(foQueue.dequeue(fanoutId1));
    System.out.println(item);
}
```

- Fanout 2 Dequeue

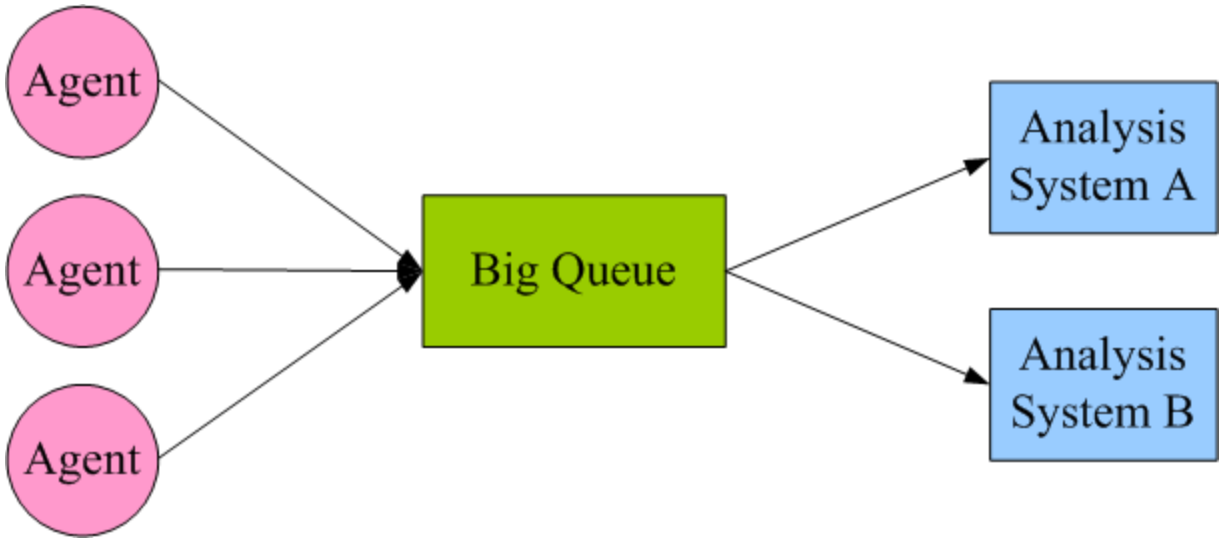
```
String fanoutId2 = "offline";
while(!foQueue.isEmpty(fanoutId2)) {
    String item = new String(foQueue.dequeue(fanoutId2));
    System.out.println(item);
}
```

Fanout 1 and Fanout 2 consuming are independent

Fanout Queue Semantics



Use Case 1 :
Log Collecting & Consuming



Use Case 2 : Big Data Sorting

- Queue Only Algorithm:
  1. Put all data into a source queue
  2. Build a queueOfSortedQueues by dividing and sorting the source queue
  3. Merge sort the queueOfSortedQueues
  4. The last one left in the queueOfSortedQueues is the final sorted queue

Source, Samples, Docs and Tutorials

https://github.com/bulldog2011/bigqueue

Other Alternatives

- Apache ActiveMQ
  - http://activemq.apache.org
- RabbitMQ
  - http://www.rabbitmq.com/
- ZeroMQ
  - http://www.zeromq.org
- Kestrel
  - https://github.com/robey/kestrel
- Apache Kafka
  - http://kafka.apache.org