Thread Scaling of ROOT's Serialization

Dr Christopher Jones *FNAL*

ROOT Serialization

ROOT serialization can be used separate from ROOT I/O

- Implemented using TBufferFile class

Want to understand threading limitations

- What happens when multiple threads are doing serialization concurrently?

Serialization Test Strategy

Used a very simple class: std::vector<Thing>

```
class Thing {
    int a;
};
```

Create the container once per job

Different measurements use different number of elements: 10 and 1000

Launch N threads

Vary N from 1 to 32

Each thread processes independently

Long loop with each iteration doing 1 serialization of the container

Keep all cores of the machine busy

Run enough jobs concurrently to fill the machine's 32 cores

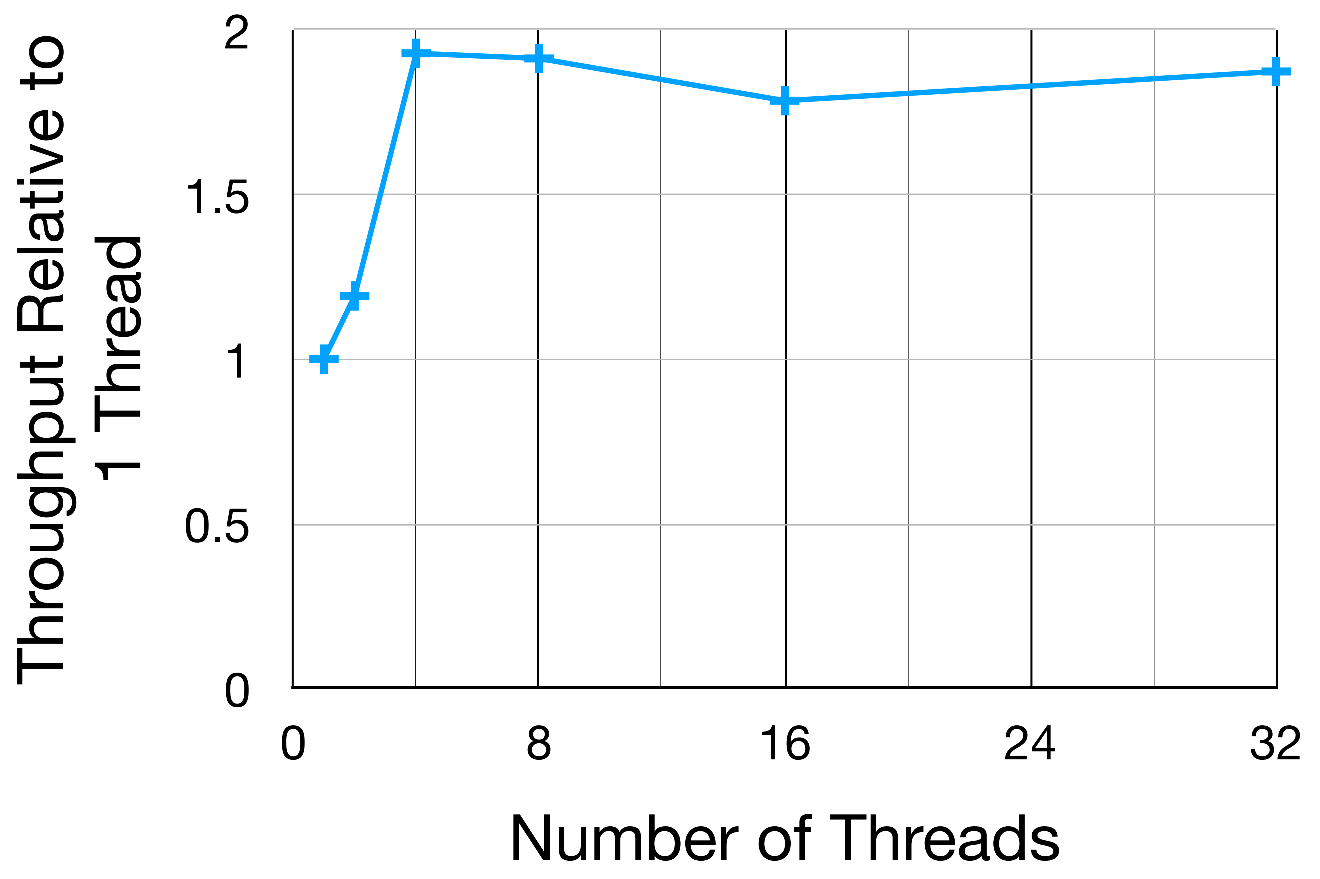
Machine Used

AMD Opteron(tm) Processor 6128

4 CPUs

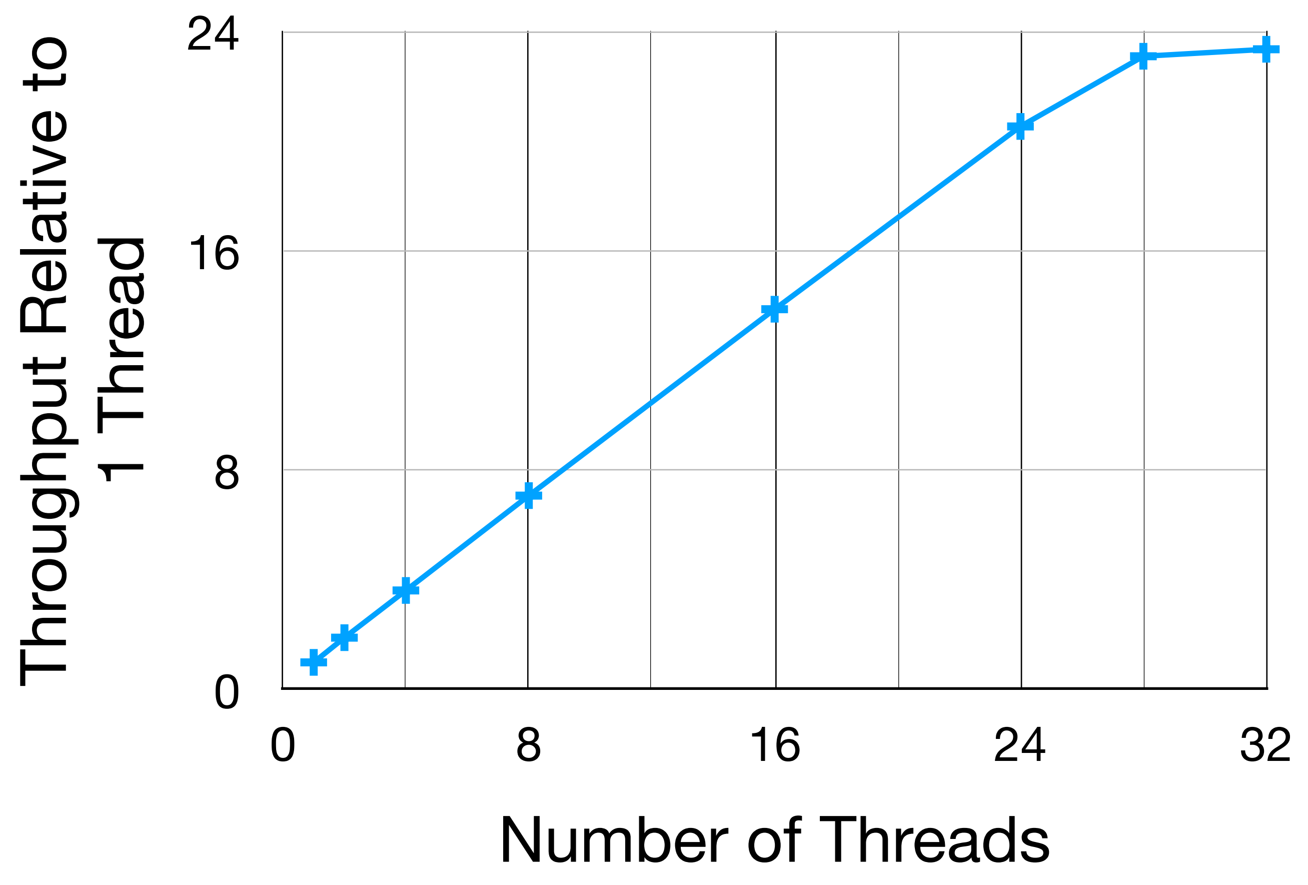
8 Cores per CPU

10 Items per Container



Jobs quickly hit serialization bottleneck

1000 Items per Container



Linear scaling up to ~24 threads

Synchronization barrier appears to be per container, not per item

Investigation

Philippe Canal found the cause of the serialization

Frequent setting of an atomic value

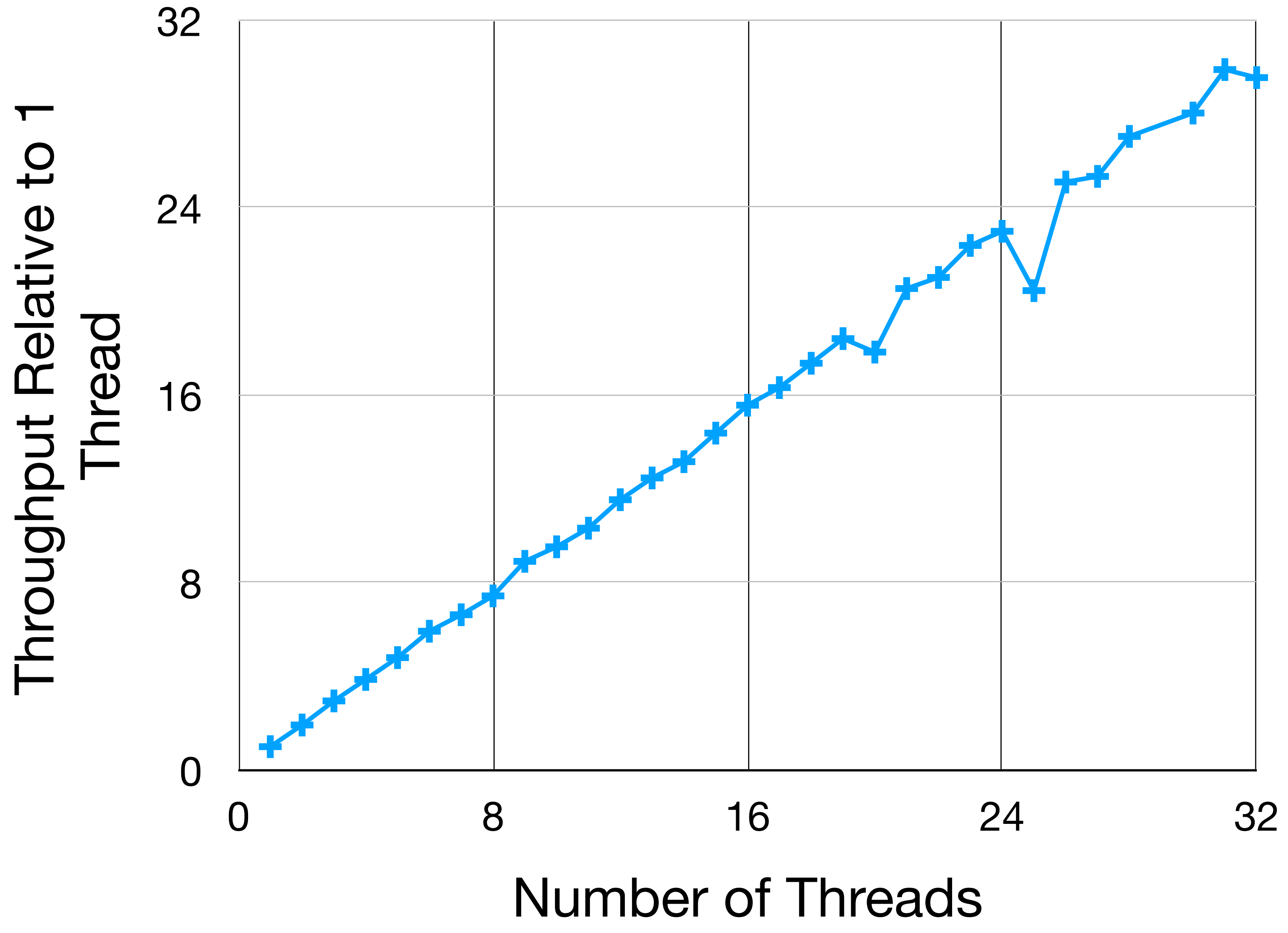
```
Version_t GetClassVersion() const {
      fVersionUsed = kTRUE;
      return fClassVersion; }
```

Changed to

```
Version_t GetClassVersion() const {
      if (!fVersionUsed.load())
         fVersionUsed = kTRUE;
      return fClassVersion; }
```

After Fix: 10 Items per Container

Scales well with number of threads



Deserialization Test Strategy

Use same container as serialization strategy

Serialize the container once per job into a buffer

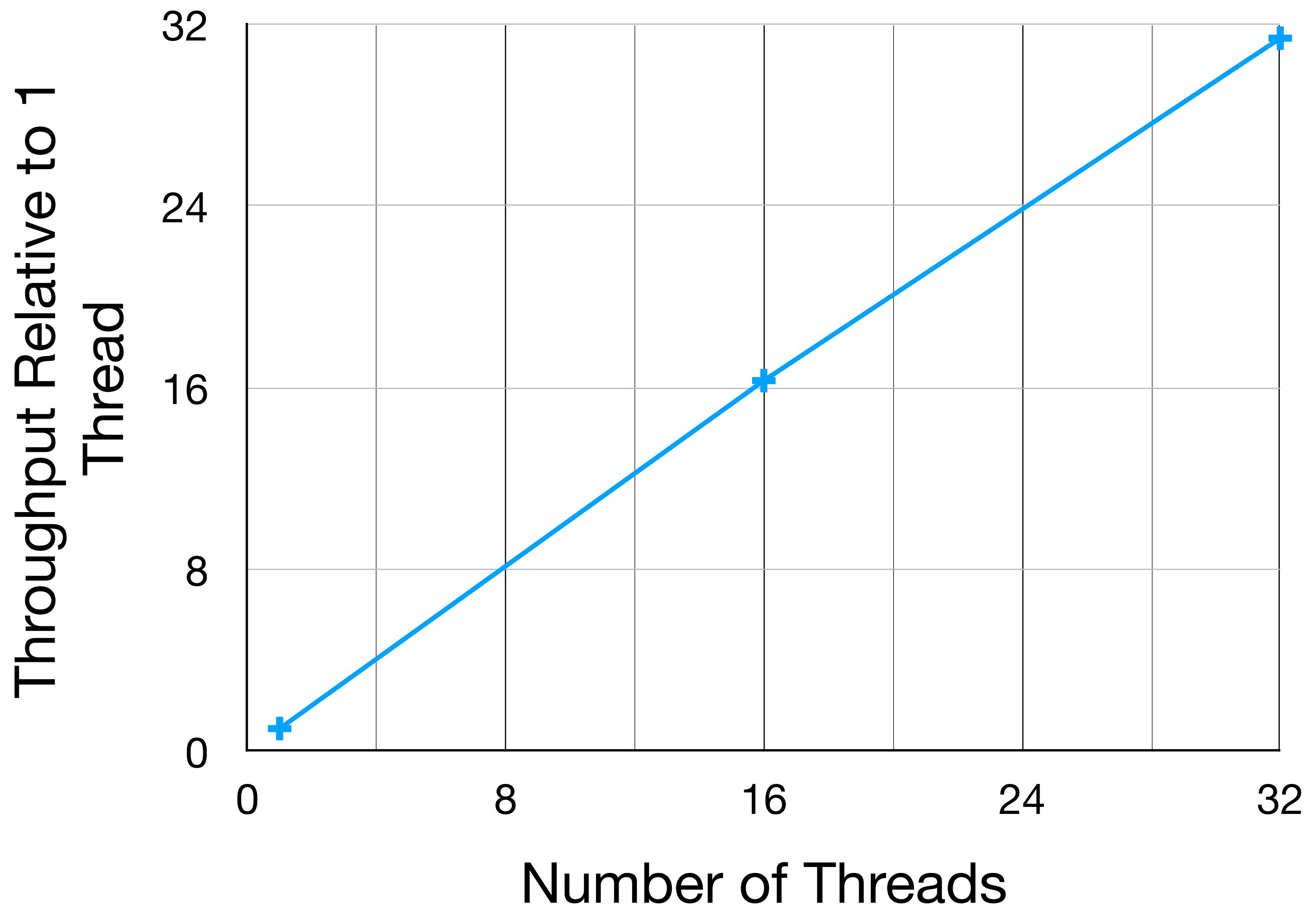
Launch N threads
- Vary N from 1 to 32

Each thread processes independently
- Long loop with each iteration doing 1 deserialization of the buffer

Keep all cores of the machine busy
- Run enough jobs concurrently to fill the machine's 32 cores

10 Items per Container

Shows perfect scaling


Throughput Relative to 1 Thread
Number of Threads