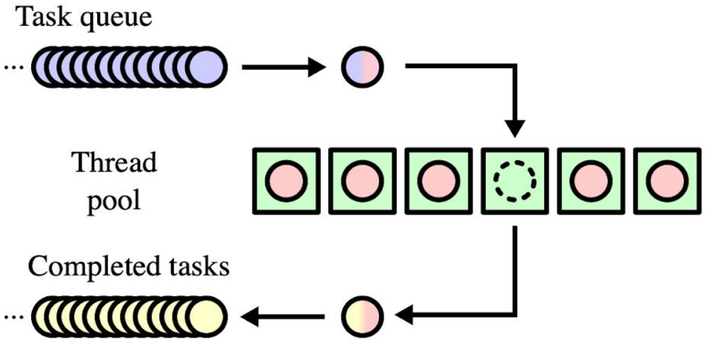
ThreadPool

Harshit Roy
UT Austin

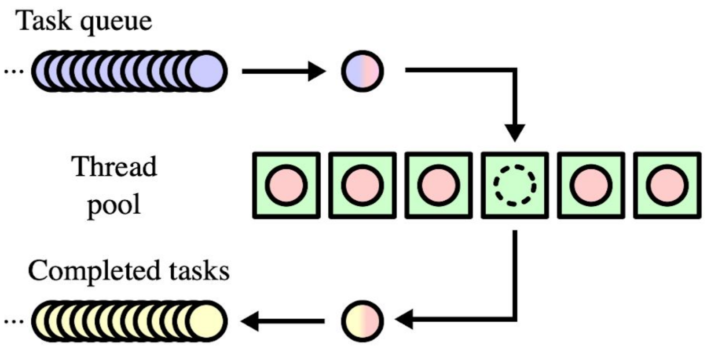


BOOST ASIO

- **Threads** fetch tasks from a **central queue**
- **Scheduling policies:**
  - **FIFO** (First-In, First-Out)
  - **Round-robin** (load-balanced)

- Uses **epoll** to monitor I/O readiness without blocking threads
- One 1 thread can wait on Linux **epoll** fd, thus limiting scalability

- **No dynamic decisions** - least overhead



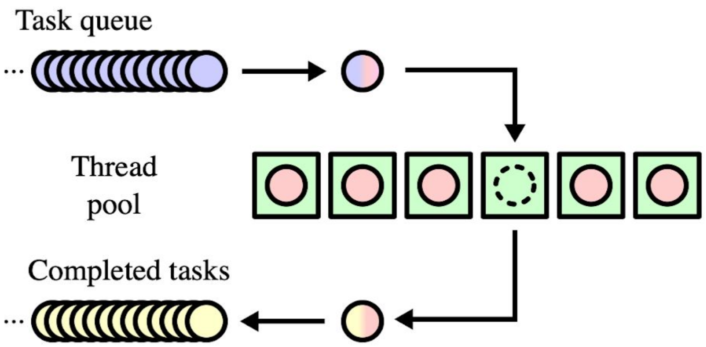
Intel TBB

No central task queue — improves scalability & reduces contention

Cilk-style Work Stealing

- Work: LIFO (Last-In, First-Out)
  - Great for locality and cache reuse
- Steal: FIFO (First-In, First-Out)
  - Load Balancing, oldest tasks are most independent
- LIFO + FIFO
  - A sweat spot

Dynamic Decision - results in overhead



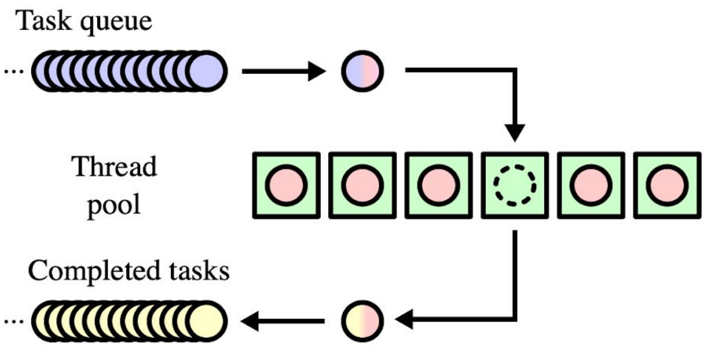
BS ThreadPool

No central task queue — improves scalability & reduces contention

Dynamic Scheduling - high overhead

Similar to Intel TBB for Compute and IO tasks

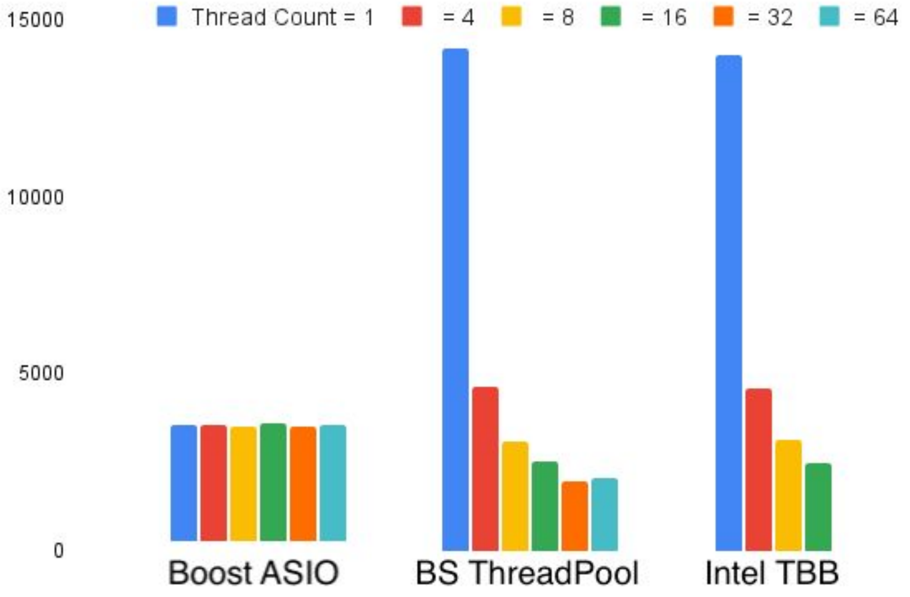
Inefficient in handling mixed workloads (compute + I/O)



I/O-Intensive Workload: File Read/Write Throughput

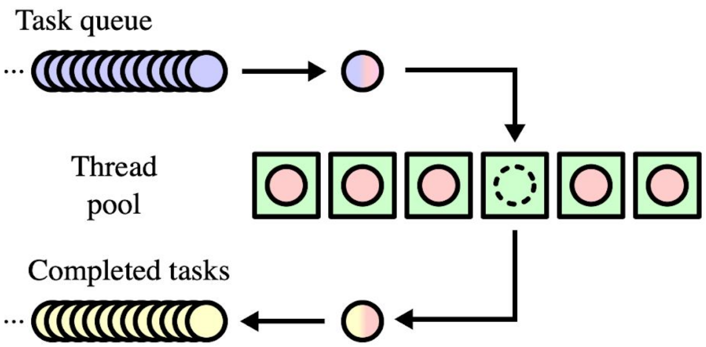
Creates 50 tasks (NUM_TASKS = 50)

- Each task:
  - Writes a 100 MB file to disk filled with 'A'
  - Then reads the same file back into memory



Boost ASIO

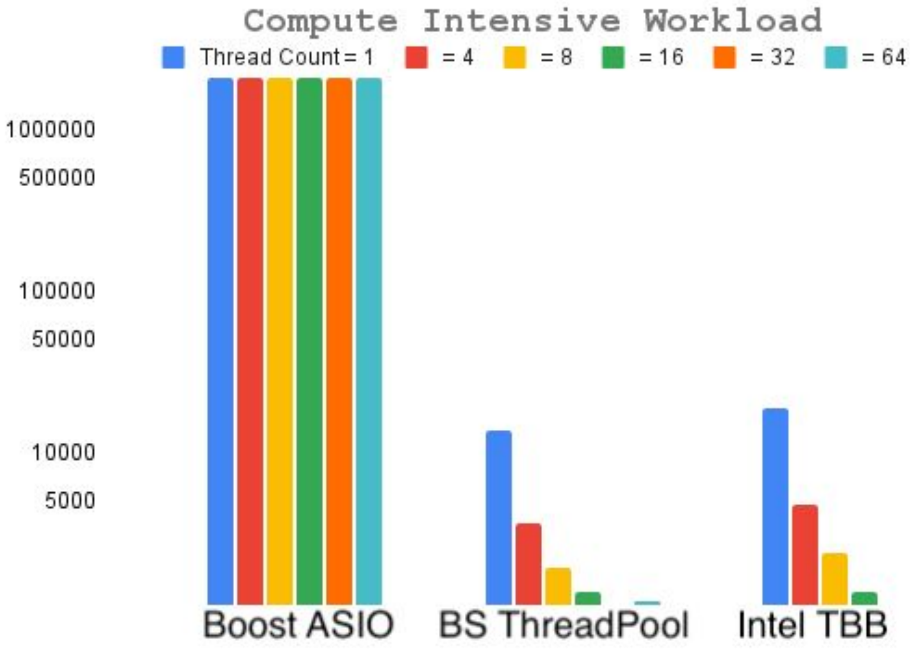
- Designed for async IO tasks
- Scaling is limited



Compute-Intensive Workload

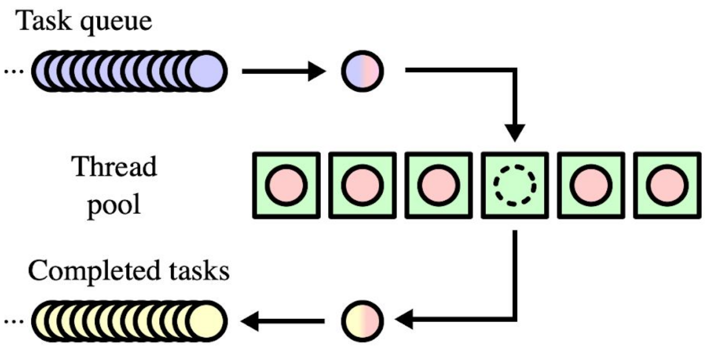
Creates three square matrices:

- A and B are filled with random integers from 1 to 10
- C is initialized to all 0s, to store the result of A × B



Boost ASIO

- Not designed for CPU-intensive tasks
- Scaling is limited

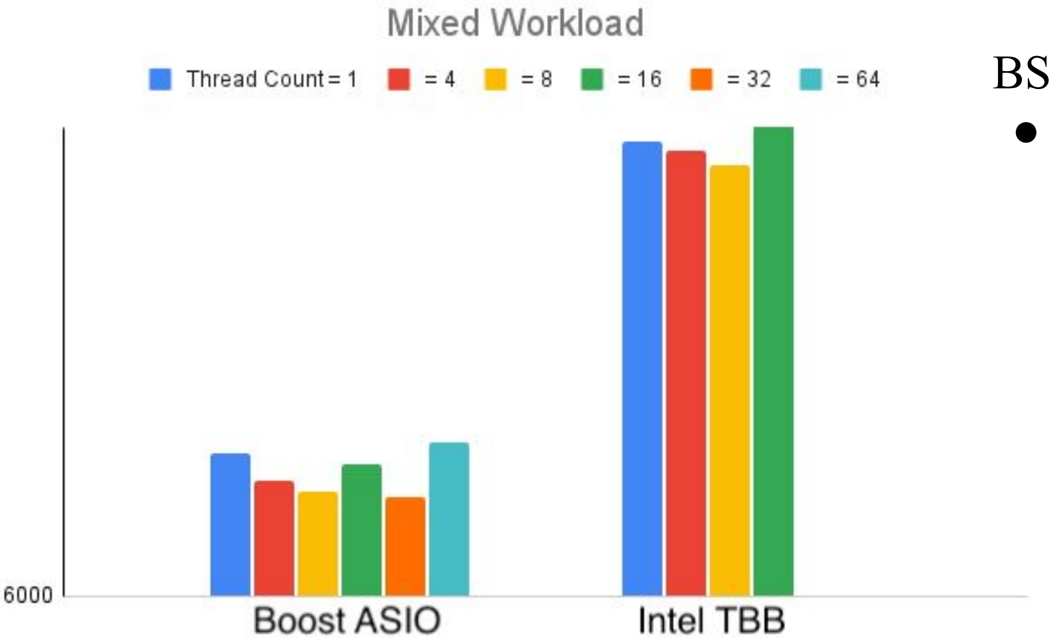


Mixed Workload (Compute + IO Workload)

Splits data into chunks of 1 million integers.

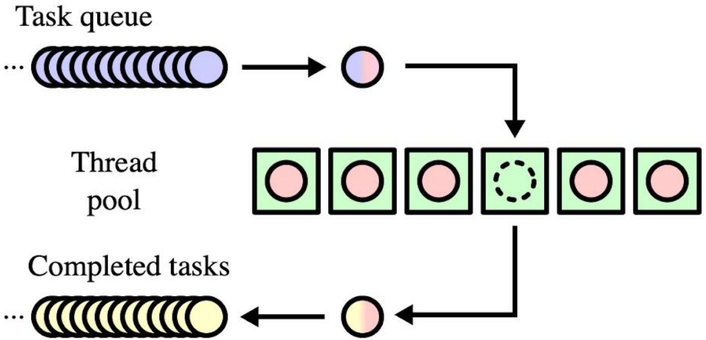
Each chunk is independently: extracted, sorted, saved to a separate chunk file

These can later be **merged** later



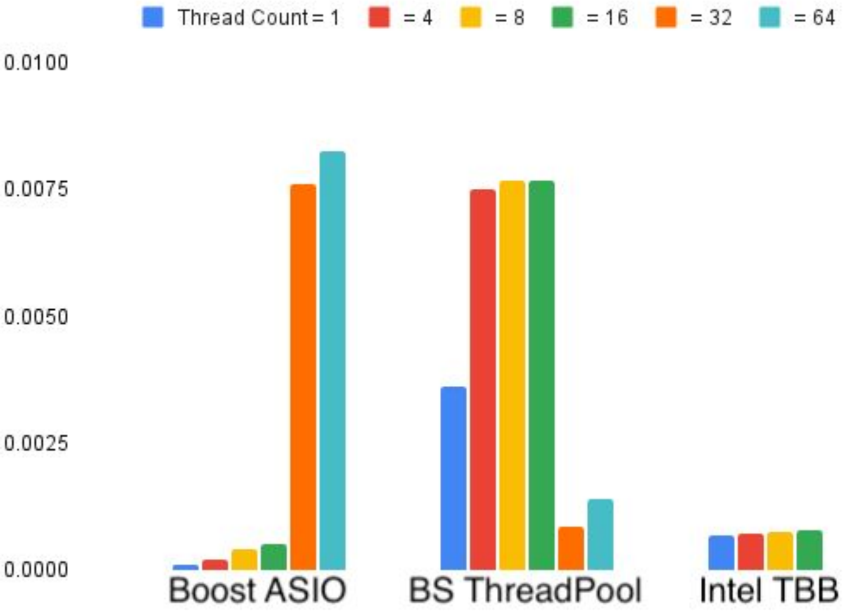
BS ThreadPool

- Not designed for mixed workload



Thread Pool Overhead

Task scheduling overhead
Time taken to assign no-op tasks to threads.

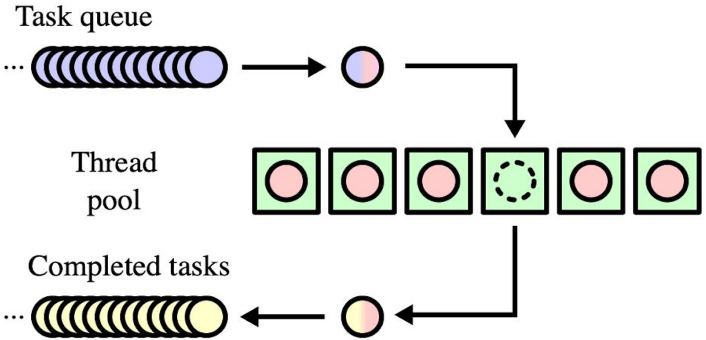


BS ThreadPool

- High overhead

Boost ASIO

- Least overhead



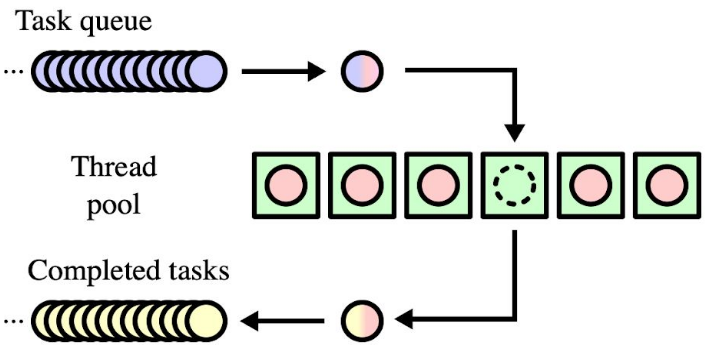
Thread Scheduling

CPU-intensive and I/O Intensive tasks are submitted concurrently.

None of the thread pools support non-blocking I/O

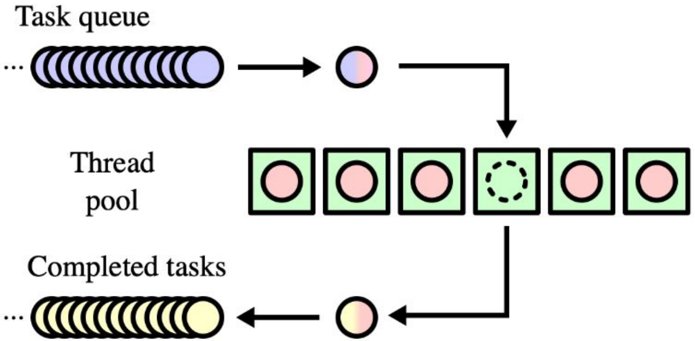
E.g. No of threads = 4, 5 I/O and CPU Task submitted concurrently

```
I/O Task 0 started at 399985 ms, ended at 411874 ms, duration: 11.88 sec
CPU Task 1 started at 399985 ms, ended at 418241 ms, duration: 18.25 sec
CPU Task 2 started at 399985 ms, ended at 418291 ms, duration: 18.30 sec
CPU Task 0 started at 399985 ms, ended at 418368 ms, duration: 18.38 sec
I/O Task 1 started at 411874 ms, ended at 424708 ms, duration: 12.83 sec
I/O Task 3 started at 418370 ms, ended at 430772 ms, duration: 12.40 sec
I/O Task 2 started at 418244 ms, ended at 430896 ms, duration: 12.65 sec
CPU Task 3 started at 418293 ms, ended at 436642 ms, duration: 18.34 sec
I/O Task 4 started at 430772 ms, ended at 442698 ms, duration: 11.92 sec
CPU Task 4 started at 424708 ms, ended at 442922 ms, duration: 18.21 sec
```



Questions?

https://github.com/royharshit/threadpool_benchmark


Task queue
Thread pool
Completed tasks