A Parallel, Generic Sorting Library in Go

QOSORT

Shaojie Bai
Yutong Chen

QOSORT

- Portable library
- Supports generic data types
- Optimized for multi-core
- Scalable with resources

CHALLENGES

- Lack of explicit hardware scheduling
- Lack of parallel API support
- Enormity of data size
- Complications associated with generic data types

SORT IN GO

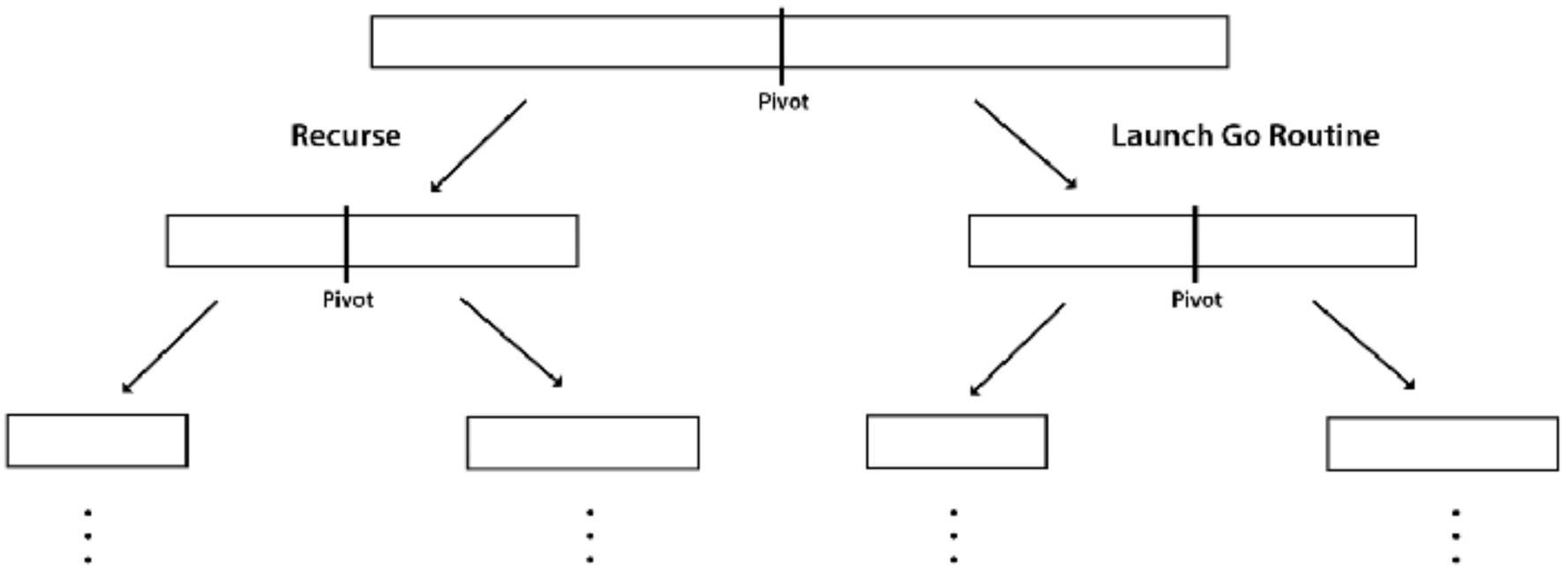
sort.Interface

```
type Interface interface {
        // Len is the number of elements in the collection.
        Len() int
        // Less reports whether the element with
        // index i should sort before the element with index j.
        Less(i, j int) bool
        // Swap swaps the elements with indexes i and j.
        Swap(i, j int)
}
```

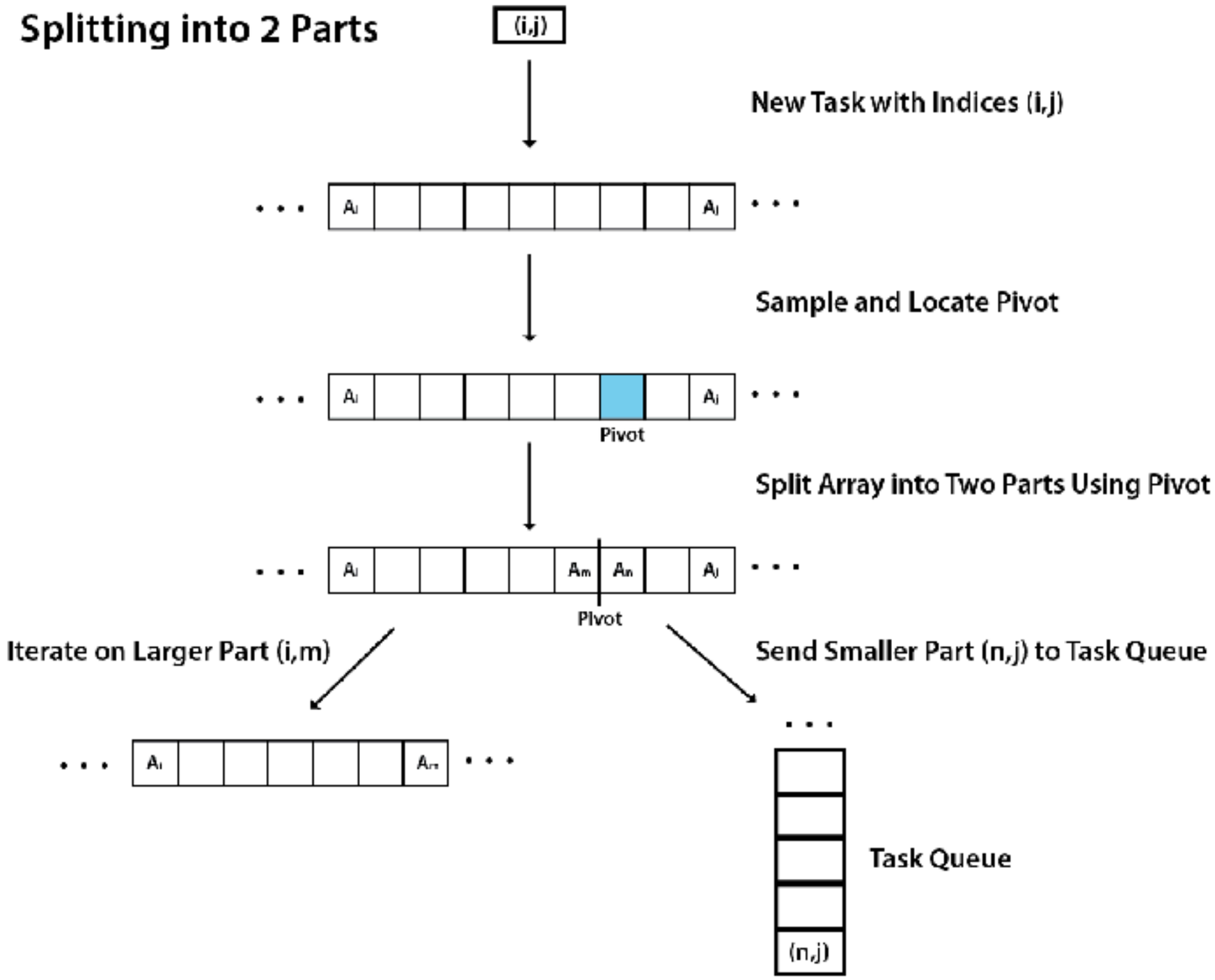
QUICKSORT

- **Quick** recap
- Two factors at stake:
  - Pivot
  - Scheduling

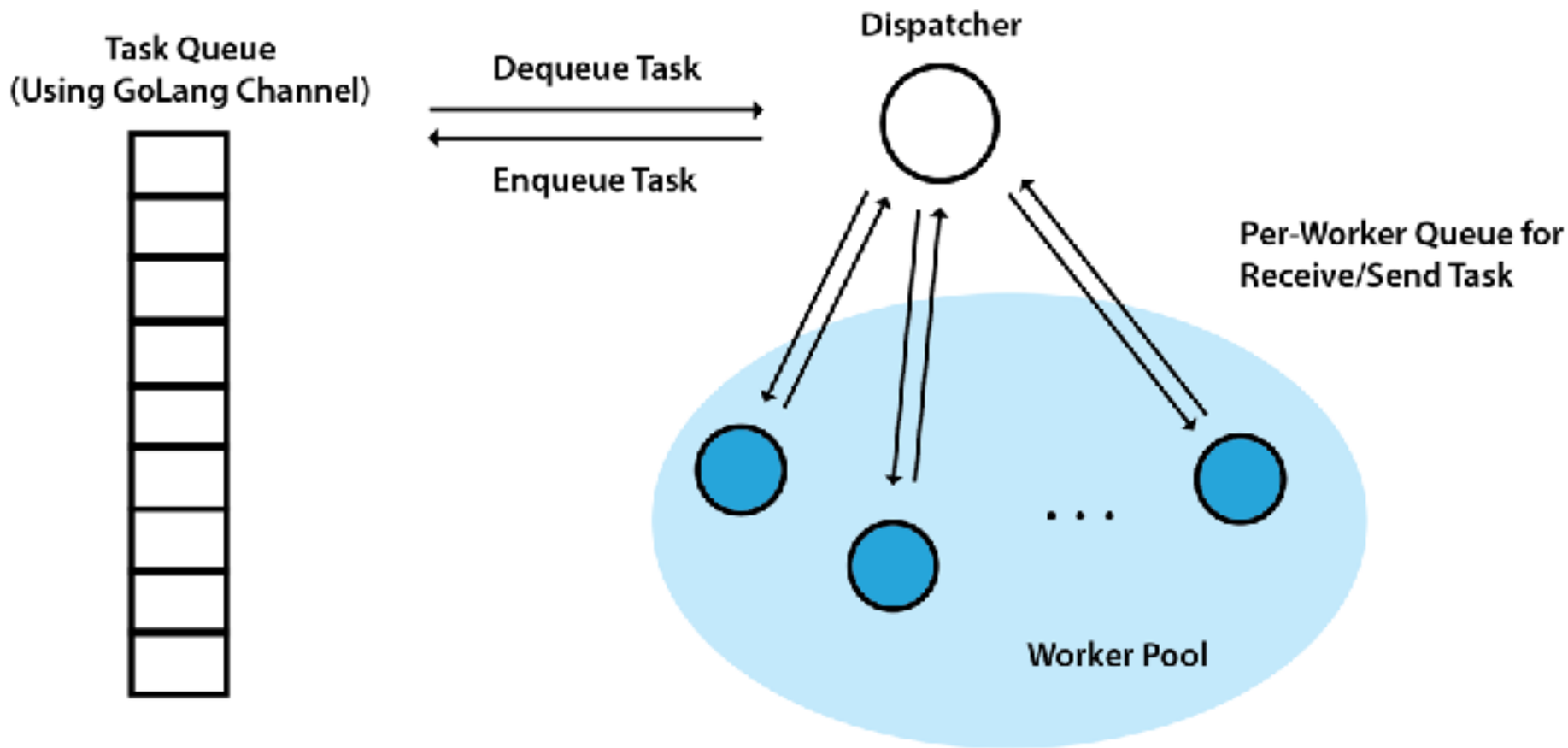
QUICKSORT V1.0 – "NAIVE"



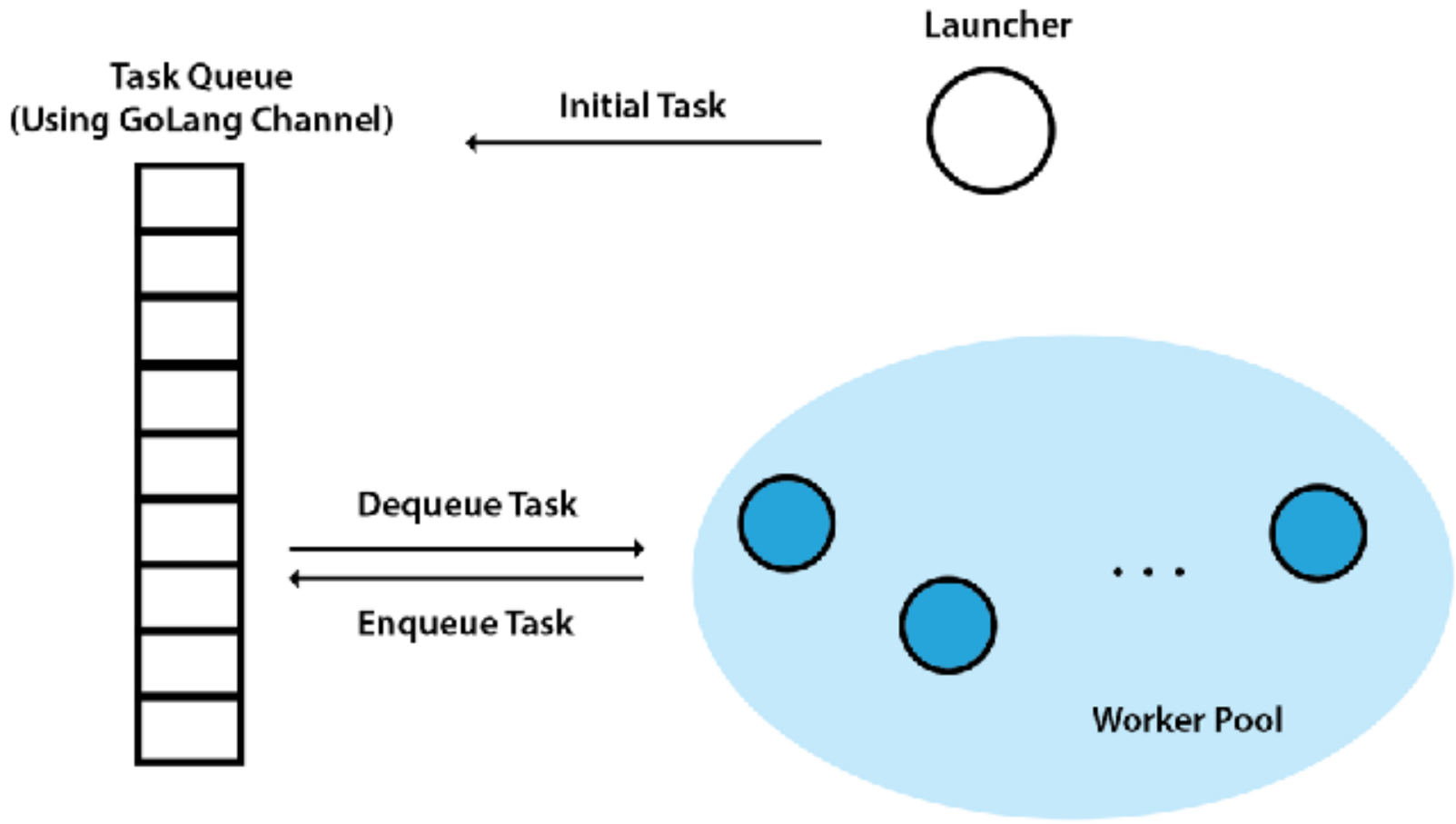
QUICKSORT TASK-BASED SCHEDULING



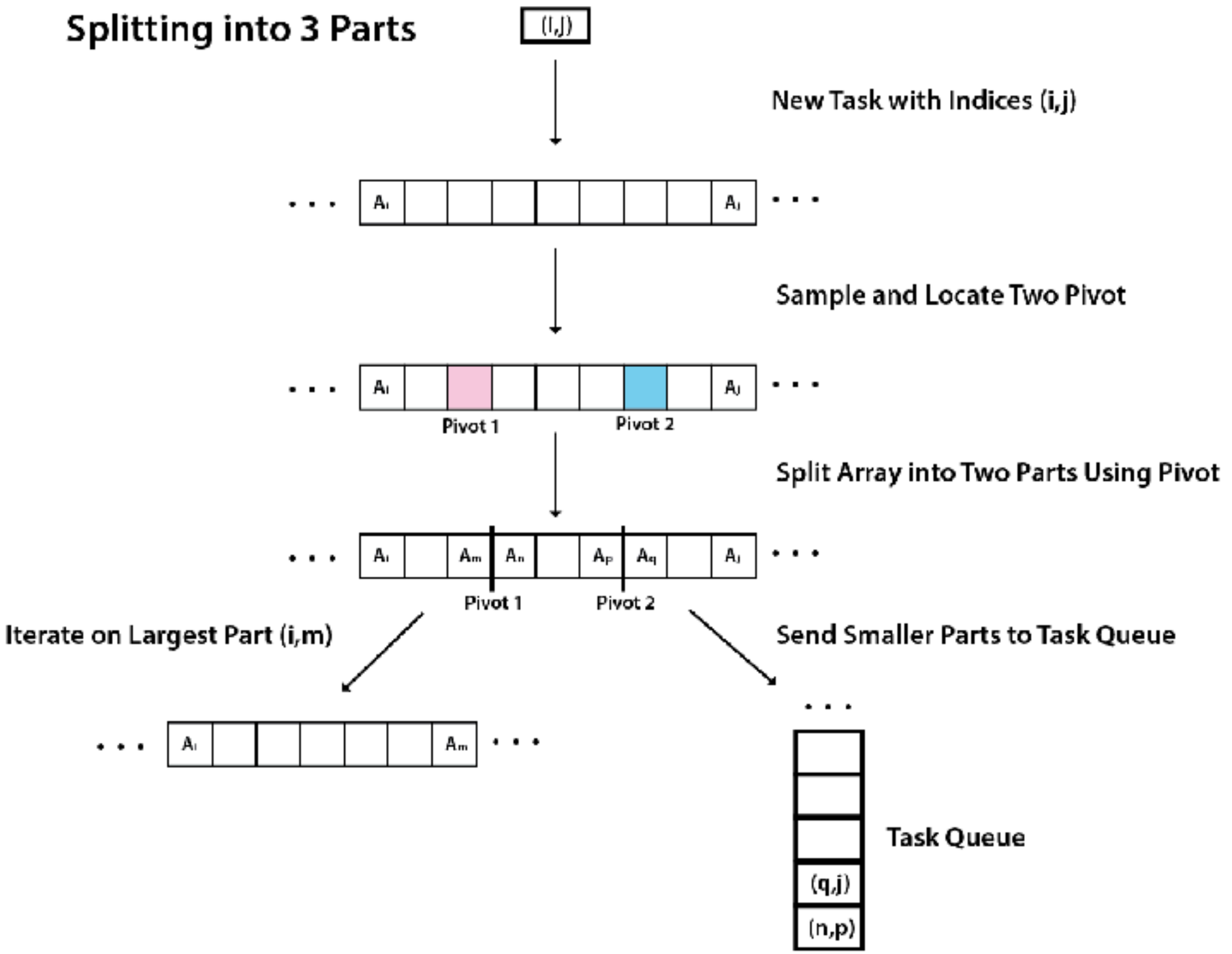
QUICKSORT V2.0 – "MASTER DISPATCH"



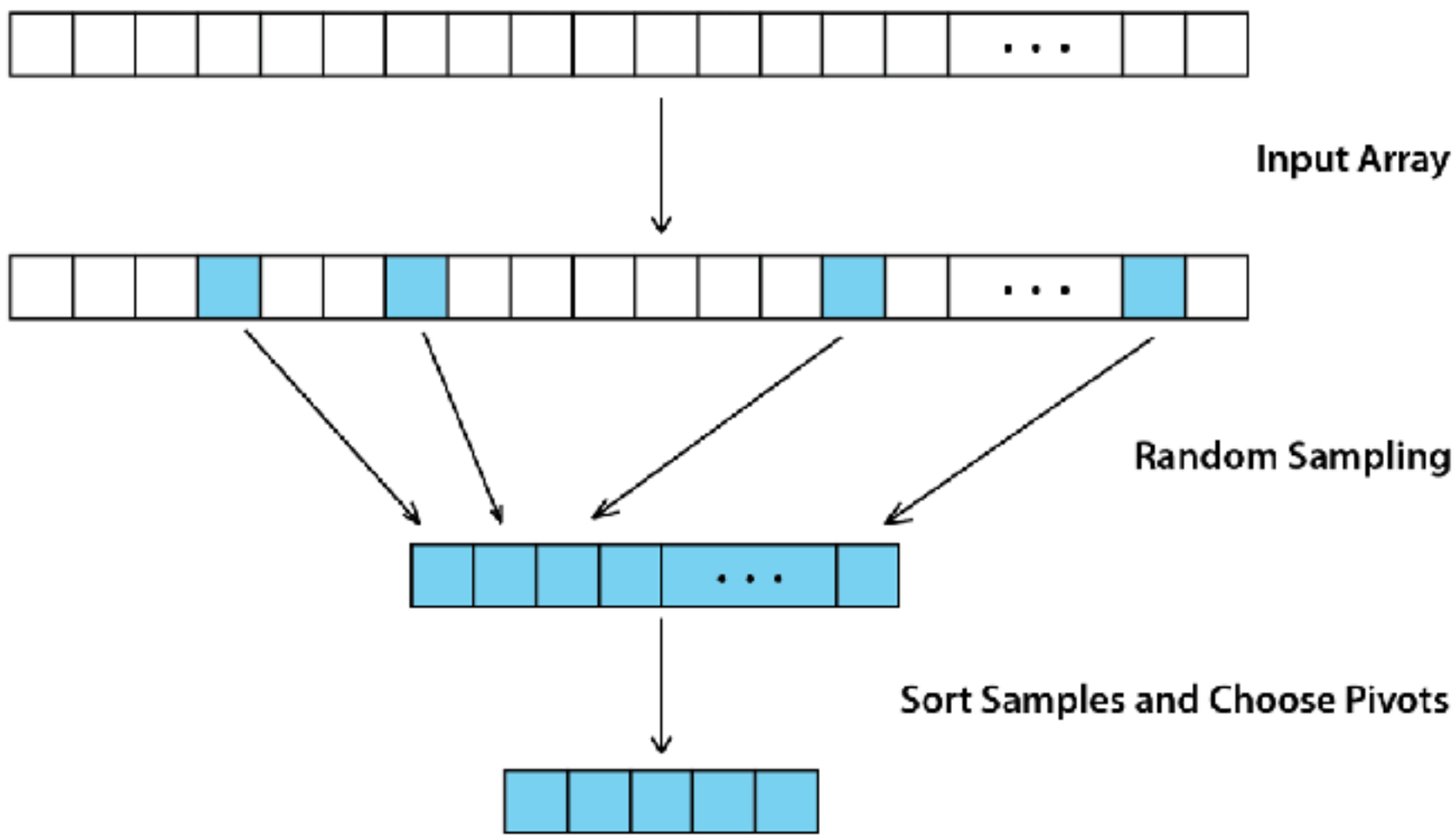
QUICKSORT V3.0 – "ONE QUEUE"



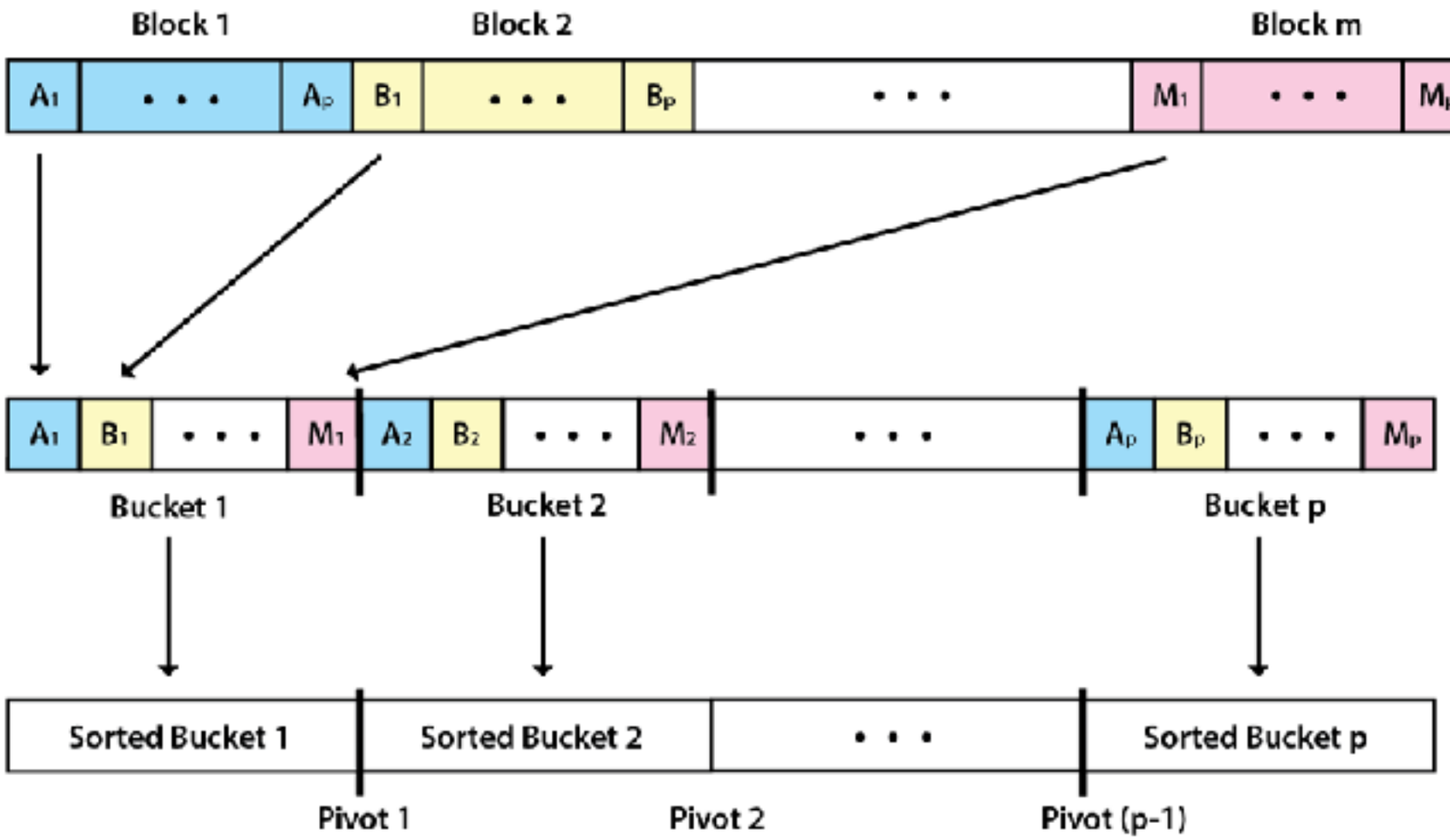
QUICKSORT V4.0 – "SPLIT-BY-3"



SAMPLESORT – SAMPLE



SAMPLESORT – BLOCK TRANSPOSE



Transpose Blocks to Buckets

Sort Each Bucket

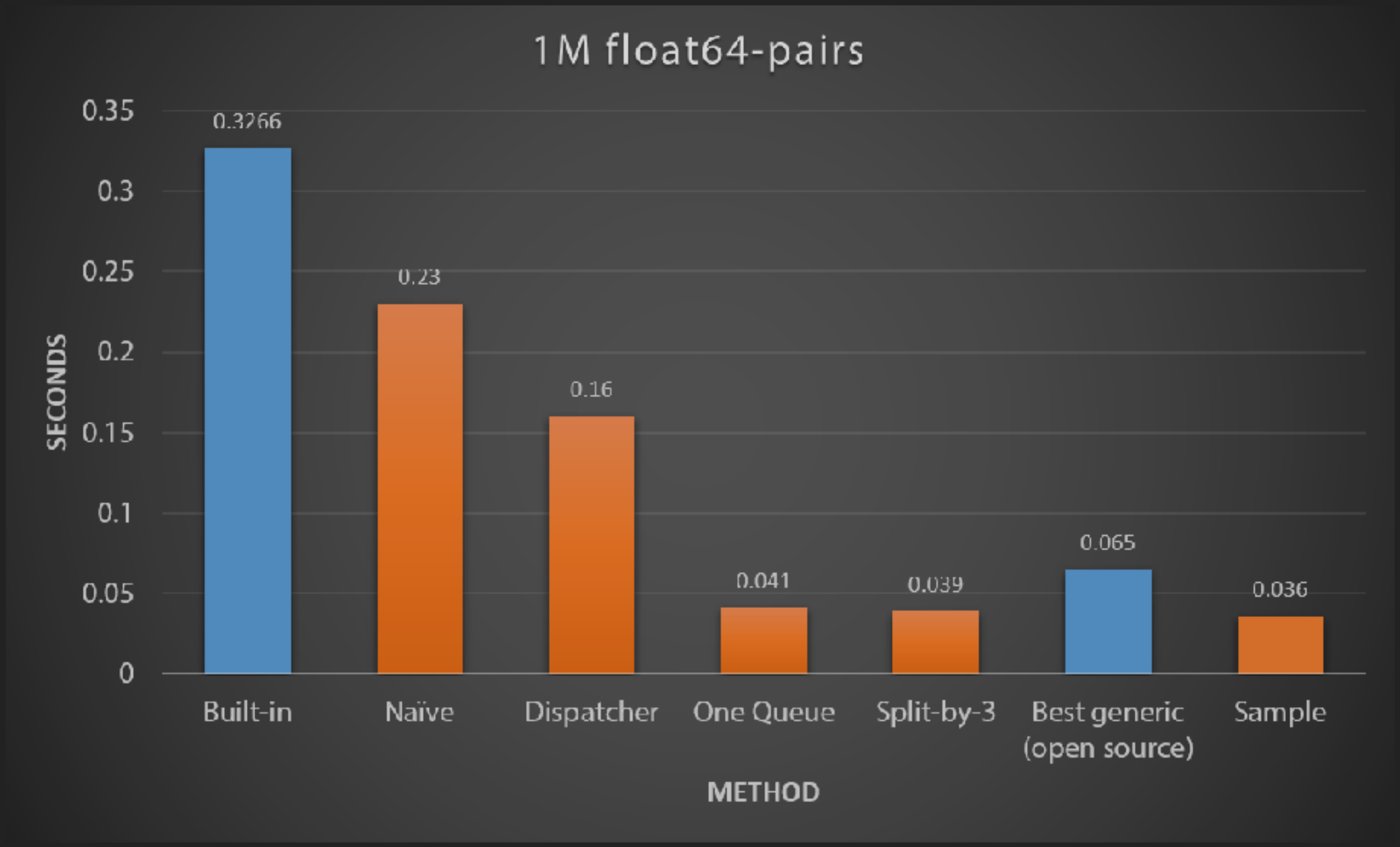
IMPROVEMENT OVER QUICKSORT

- Highly parallelizable
- Even better load balancing
- Parallelism starts early
- Spatial locality

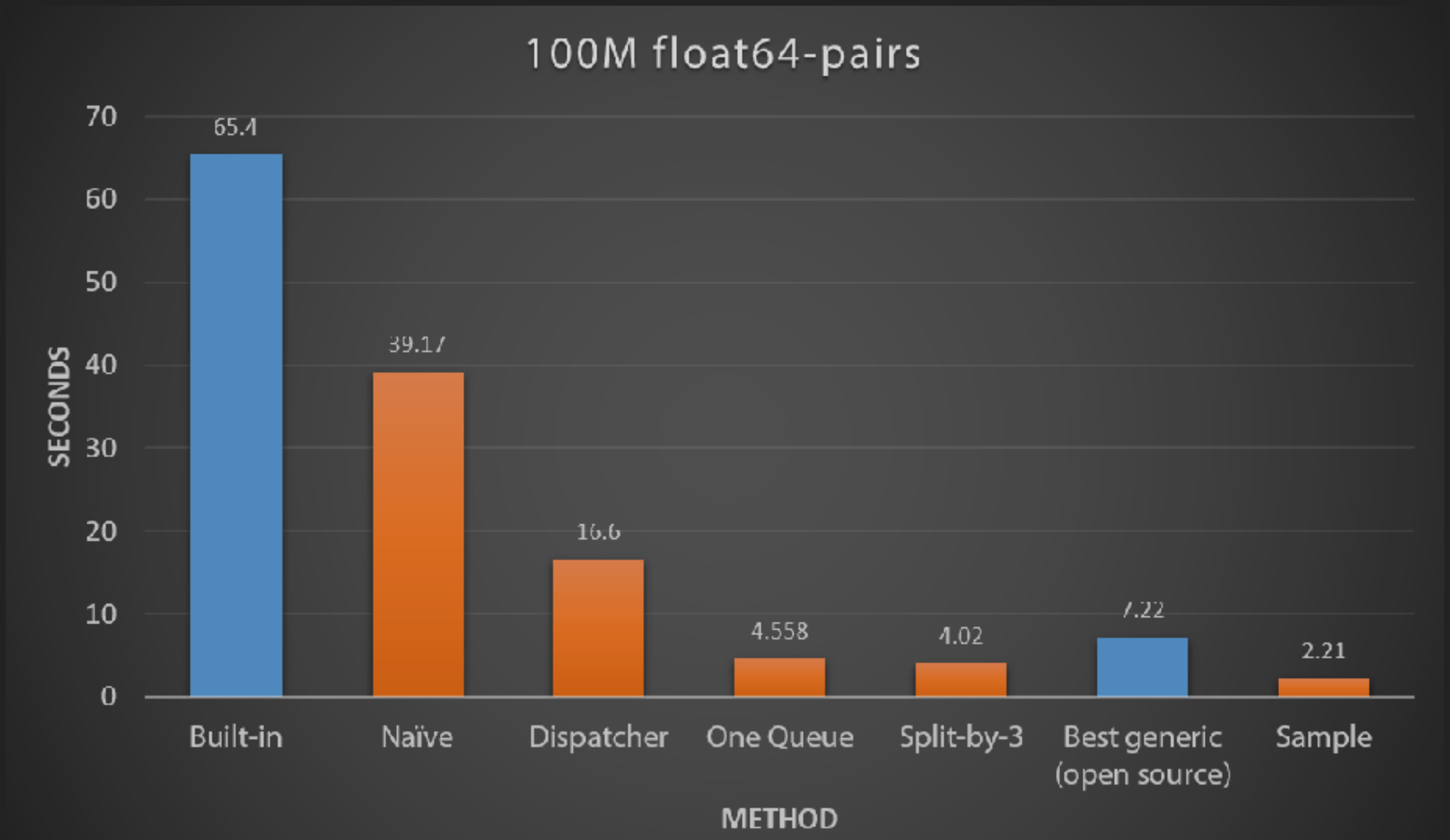
TEST SETTING

- UNIX Machine - 2x Intel(R) Xeon(R) E5-2680 v2 (20 cores)
- Go 1.4.2 Linux/AMD64
- 100,000,000 float64 key-value pairs

PERFORMANCE



PERFORMANCE (CONT'D)



QUESTIONS?

- Questions for you consideration:
  - What's the size of worker pool ?
  - "Base case" for each algorithm ?
- Why the name "QoSort" ?
- Why is Kayvon replying my comment at 2am ?!
- ...