CSE502: Foundations of Parallel Programming

Lecture 09: Types of Work-Stealing

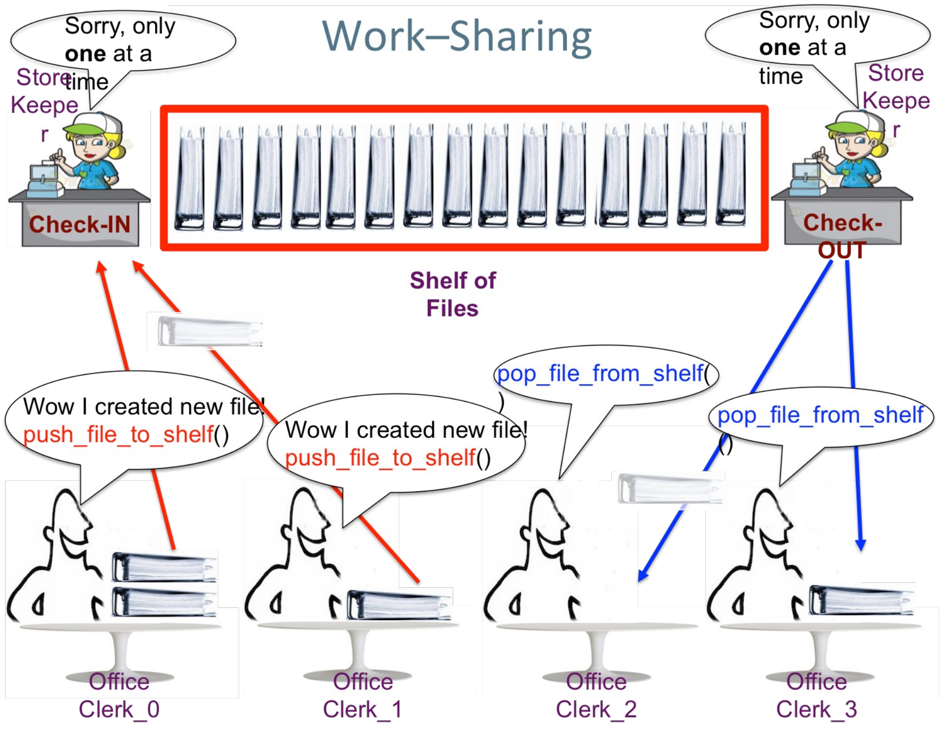
Vivek Kumar

Computer Science and Engineering

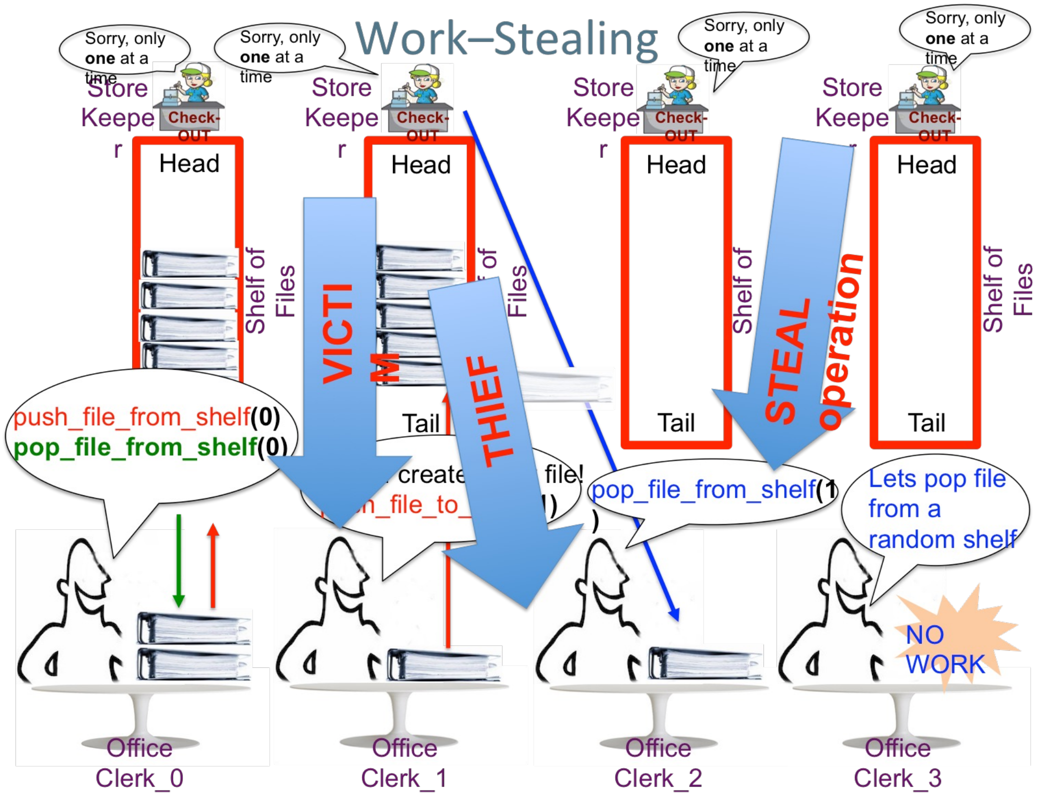
IIIT Delhi

vivekk@iiitd.ac.in

Last Class – Task Scheduling Paradigms



Work-Sharing



Work-Stealing

Today's Class

- Types of work-stealing scheduling
  - Work-first
  - Help-first
- Quiz-2

Types of Work-Stealing

- Work-first
  - Cilk, X10, TryCatchWS
- Help-first
  - Habanero-C library (HClib), Java fork/join

Types of Work-Stealing

```
1. finish {
2.     async S1;
3.     //continuation of S1
4.     async S2;
5.     //continuation of S2
6.     S3;
7. }
```

Work-first

With single worker, program execution using work-first policy is similar to serial execution

```
start_finish();
push_task_to_runtime(Line_3+);
S1;
if(Line_3+_stolen) return;
push_task_to_runtime(Line_5+);
S2;
if(Line_5+_stolen) return;
S3;
end_finish();
```

Help-first

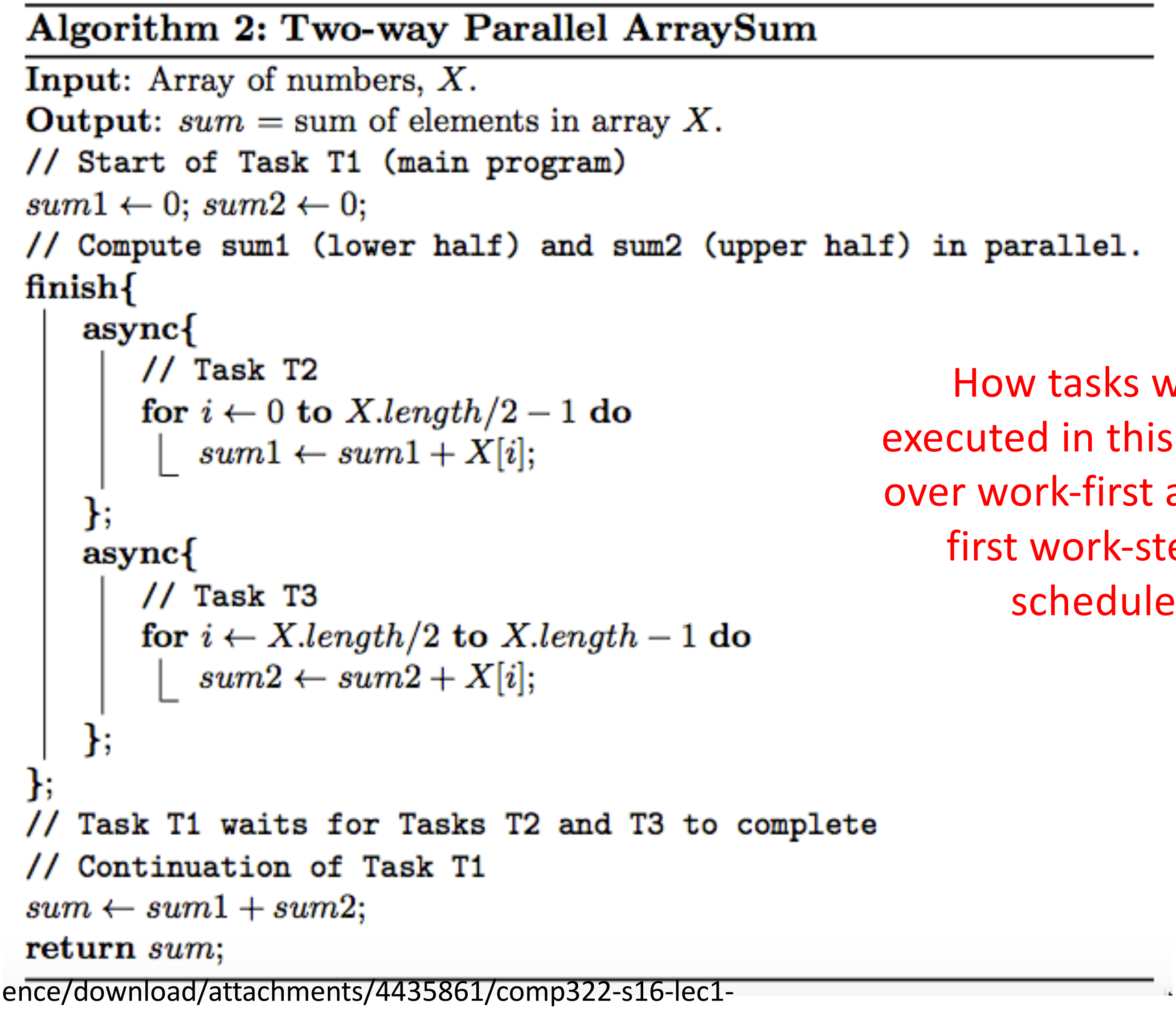
```
start_finish();
push_task_to_runtime(S1);
push_task_to_runtime(S2);
S3;
end_finish();
```

Points to ponder

- What task is getting pushed to deque
  - Continuation in W.F.
  - “async” in H.F.
- When victim becomes a thief
  - When immediate continuation is stolen in W.F.
  - When all asyncs are stolen in H.F.

Source: Work-first and help-first scheduling policies for async-finish task parallelism, Guo et. al., IPDPS 2009

Parallel Array Sum using async and finish Constructs



Algorithm 2: Two-way Parallel ArraySum

```
Input: Array of numbers, $X$.
Output: $sum$ = sum of elements in array $X$.
// Start of Task T1 (main program)
$sum1 \leftarrow 0$; $sum2 \leftarrow 0$;
// Compute sum1 (lower half) and sum2 (upper half) in parallel.
finish{
    async{
        // Task T2
        for $i \leftarrow 0$ to $X.length/2 - 1$ do
            $sum1 \leftarrow sum1 + X[i]$;
    };
    async{
        // Task T3
        for $i \leftarrow X.length/2$ to $X.length - 1$ do
            $sum2 \leftarrow sum2 + X[i]$;
    };
};
// Task T1 waits for Tasks T2 and T3 to complete
// Continuation of Task T1
$sum \leftarrow sum1 + sum2$;
return $sum$;
```

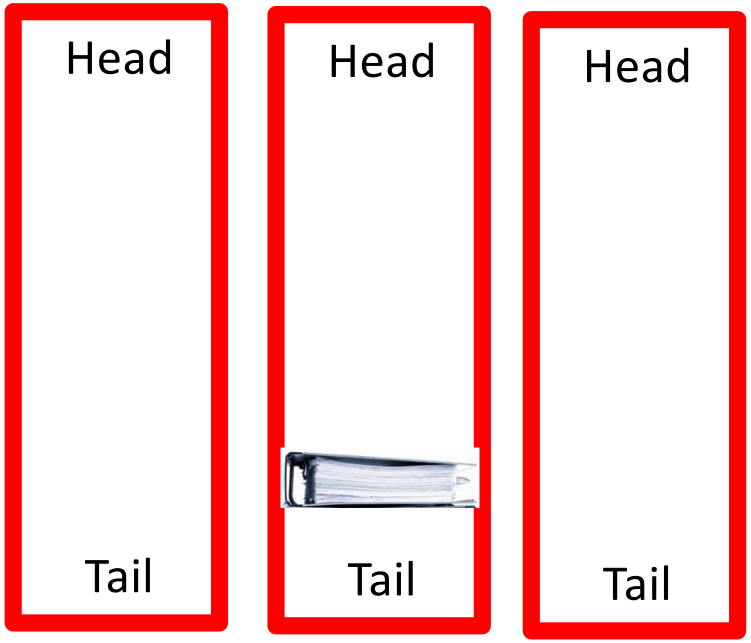
How tasks will be executed in this program over work-first and help-first work-stealing scheduler?

Source: https://wiki.rice.edu/confluence/download/attachments/4435861/comp322-s16-lec1-slides.pdf?version=1&modificationDate=1452732285045&api=v2

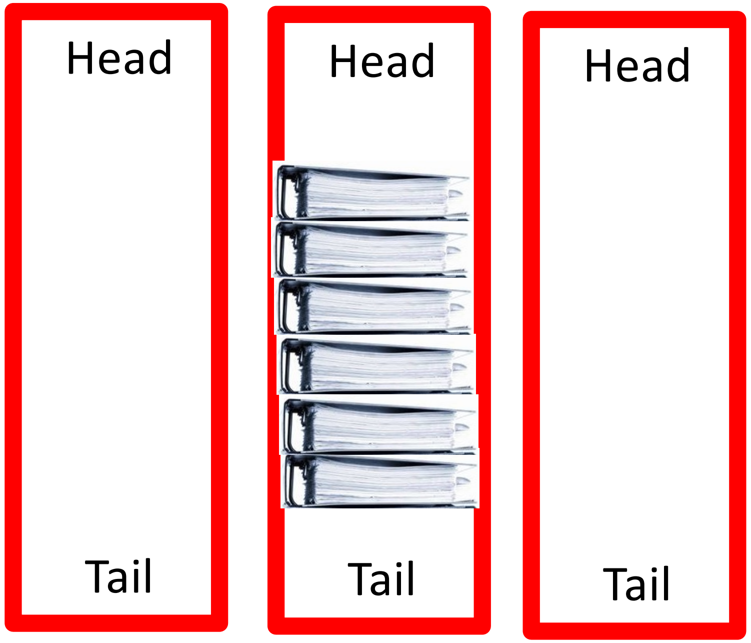
Types of Work-Stealing

Does it affect steal operations?

```
finish {
   for(int i=0; i<N; i++) {
      async S; // S does not spawn any async
   }
}
```



Work-first: at any given time there will be just one task available for stealing. New task will be generated only after the first one is stolen, leading to serialized steals. This will become scalability bottleneck with large number of workers



Help-first: plenty of tasks available for stealing as all the tasks are created upfront.

Types of Work-Stealing

- Does it affect context switches?
  - Work-first
    - Every steal will triggers a context switch of the victim
  - Help-first
    - Every task is executed after a context switch

Source: https://wiki.rice.edu/confluence/download/attachments/5212761/comp322-s11-lec11-slides-v1.pdf?version=1&modificationDate=1325608569404&api=v2

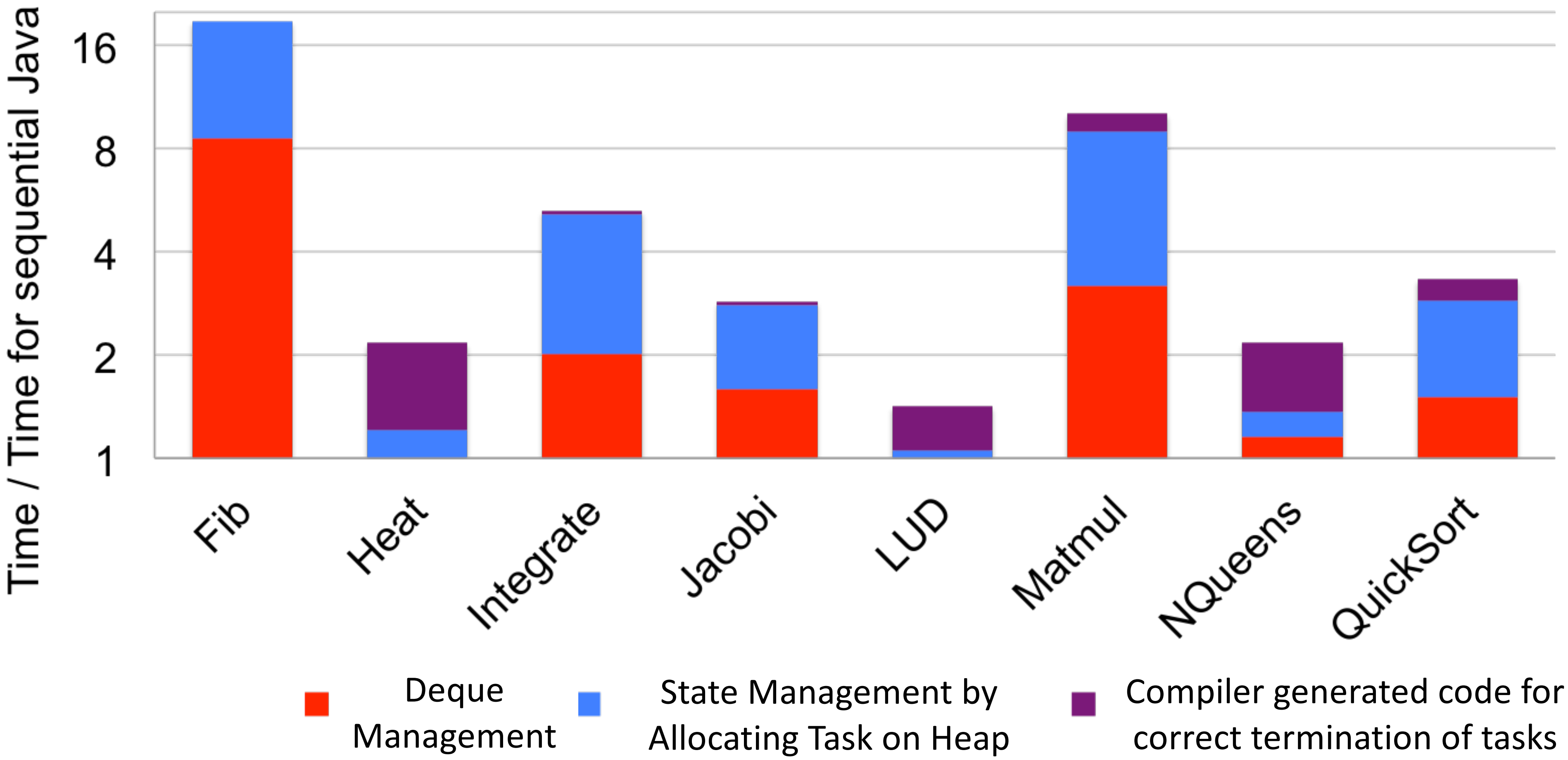
Work-Stealing Overheads

- As side-effects, work-stealing schedulers incurs some overheads
  - Deque management
    - Push
    - Pop
    - Steal
      - Insignificant overheads as steals are infrequent
  - State management
    - Allocating tasks on heap
      - Can we control this by using granularity control?
  - Code transformations in case of compiler based implementation of work-stealing

But how much of overheads??

Quick Glance Over Work-Stealing Overheads

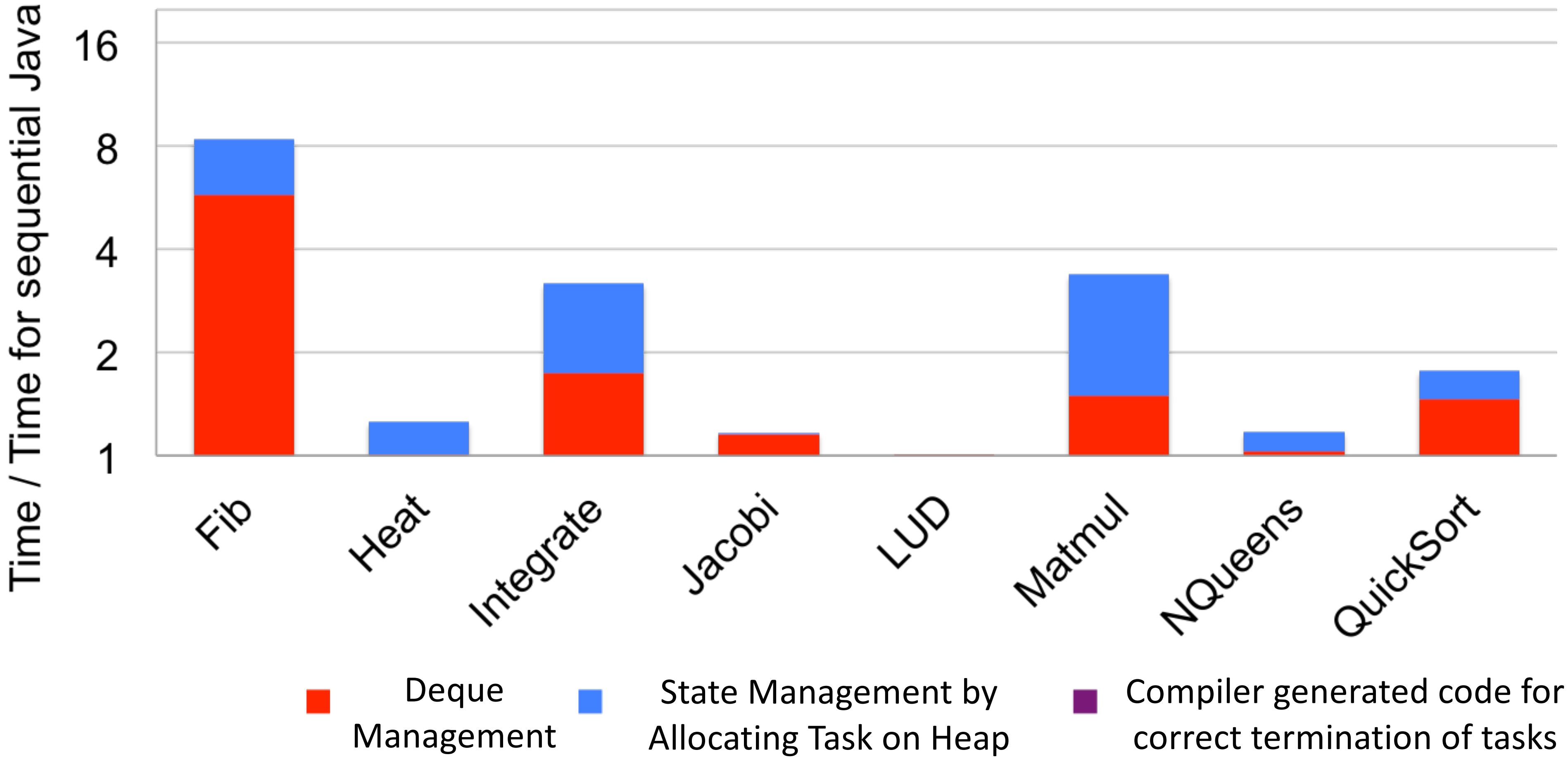
X10 Language from IBM (Compiler Based **Work-First** Implementation)



Source: Work-stealing without the baggage, Kumar et. al., OOPSLA 2012 © Vivek Kumar

Quick Glance Over Work-Stealing Overheads

Java Fork/Join Framework (Library Based **Help-First** Implementation)



Source: Work-stealing without the baggage, Kumar et. al., OOPSLA 2012

Next Class

- Memoization, Loop-level Parallelism, False Sharing

Reading Materials

- Work-first and help-first scheduling policies for async-finish task parallelism, Guo et. al. IPDPS 2009
  - http://www.cs.rice.edu/~yguo/pubs/PID824943.pdf