Cilk

An Efficient Multithreaded Runtime System

R. D. Blumofe C. F. Joerg B. C. Kuszmaul

C. E. Leiserson K. H. Randall Y. Zhou

1995

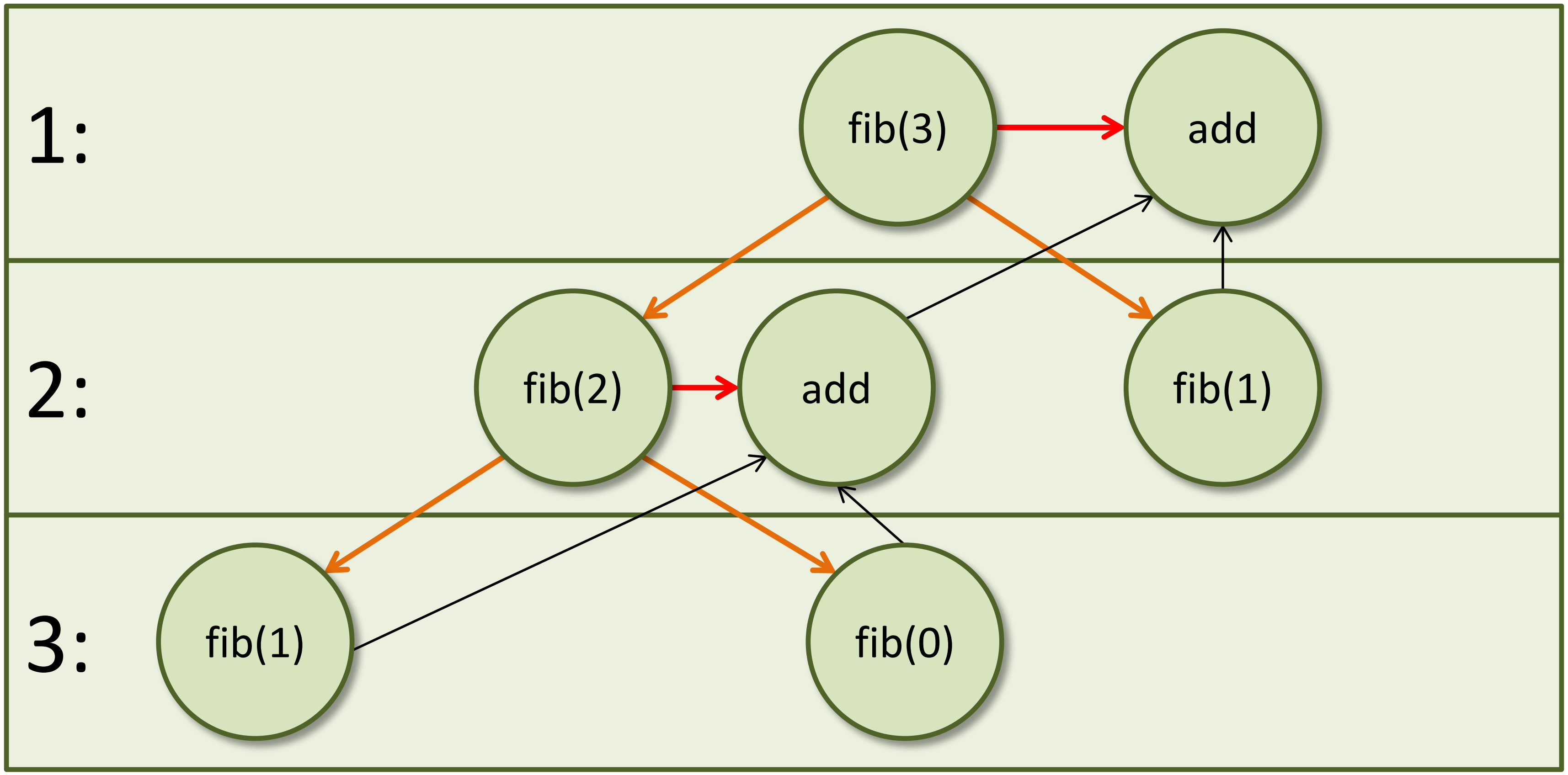
Presented by Benjamin Hess

What is Cilk

- C runtime extension
- Lightweight fork and join
  - Own scheduler
- Proofs for Performance and Space

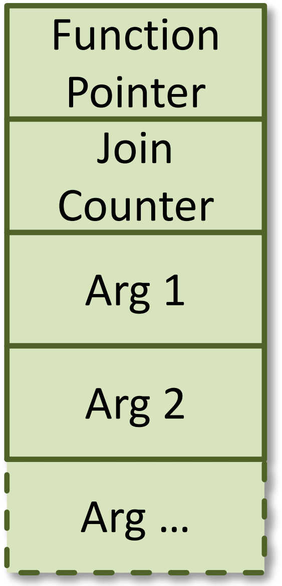
Example: fibonacci

$$fib(n) = fib(n-1) + fib(n-2)$$



Threads in Cilk

- Not a thread on the OS-level
  - Cilk Thread ≈ Task
- Represented as Closure
- Non-Blocking
- Ready or Waiting state
- Can spawn new threads
  - Children
  - Successor

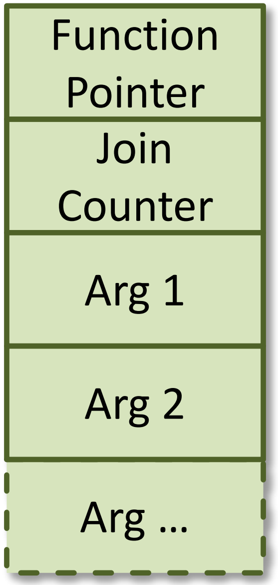


Parameter of Threads

- Parameters can be missing on creation
  - Thread starts in Waiting state

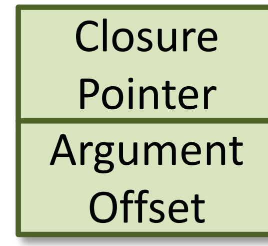
- Join Counter:
  - Number of missing arguments

- How to set those arguments?



Continuation

- Points to a missing argument



- Thread can fill in the argument
  - Decrements join counter
  - Sets argument
- If join counter is 0
  - Thread goes into ready state

Example

```
thread fib (cont int k, int n) {
  if(n<2) {
    send_argument(k, n);
  } else {
    cont int x,y;
    spawn_next sum(k, ?x, ?y);
    spawn fib(x, n-1);
    spawn fib(y, n-2);
  }
}
thread add(cont int k, int x, int y) {
  send_argument(k, x+y);
}
```

Keyword: thread

```
thread fib (cont int k, int n) {
  if(n<2) {
    send_argument(k, n);
  } else {
    cont int x,y;
    spawn_next sum(k, ?x, ?y);
    spawn fib(x, n-1);
    spawn fib(y, n-2);
  }
}
thread add(cont int k, int x, int y) {
  send_argument(k, x+y);
}
```

Denote Cilk function

Keyword: spawn

```
thread fib (cont int k, int n) {
  if(n<2) {
    send_argument(k, n);
  } else {
    cont int x,y;
    spawn_next sum(k, ?x, ?y);
    spawn fib(x, n-1);
    spawn fib(y, n-2);
  }
}
thread add(cont int k, int x, int y) {
  send_argument(k, x+y);
}
```

Spawn children thread

Keyword: spawn_next

```
thread fib (cont int k, int n) {
  if(n<2) {
    send_argument(k, n);
  } else {
    cont int x,y;
    spawn_next sum(k, ?x, ?y);
    spawn fib(x, n-1);
    spawn fib(y, n-2);
  }
}
thread add(cont int k, int x, int y) {
  send_argument(k, x+y);
}
```

Spawn successor thread

Keyword: cont

```
thread fib (cont int k, int n) {
  if(n<2) {
    send_argument(k, n);
  } else {
    cont int x,y;
    spawn_next sum(k, ?x, ?y);
    spawn fib(x, n-1);
    spawn fib(y, n-2);
  }
}
thread add(cont int k, int x, int y) {
  send_argument(k, x+y);
}
```

Denotes a continuation

Keyword: ?

```
thread fib (cont int k, int n) {
  if(n<2) {
    send_argument(k, n);
  } else {
    cont int x,y;
    spawn_next sum(k, ?x, ?y);
    spawn fib(x, n-1);
    spawn fib(y, n-2);
  }
}
thread add(cont int k, int x, int y) {
  send_argument(k, x+y);
}
```

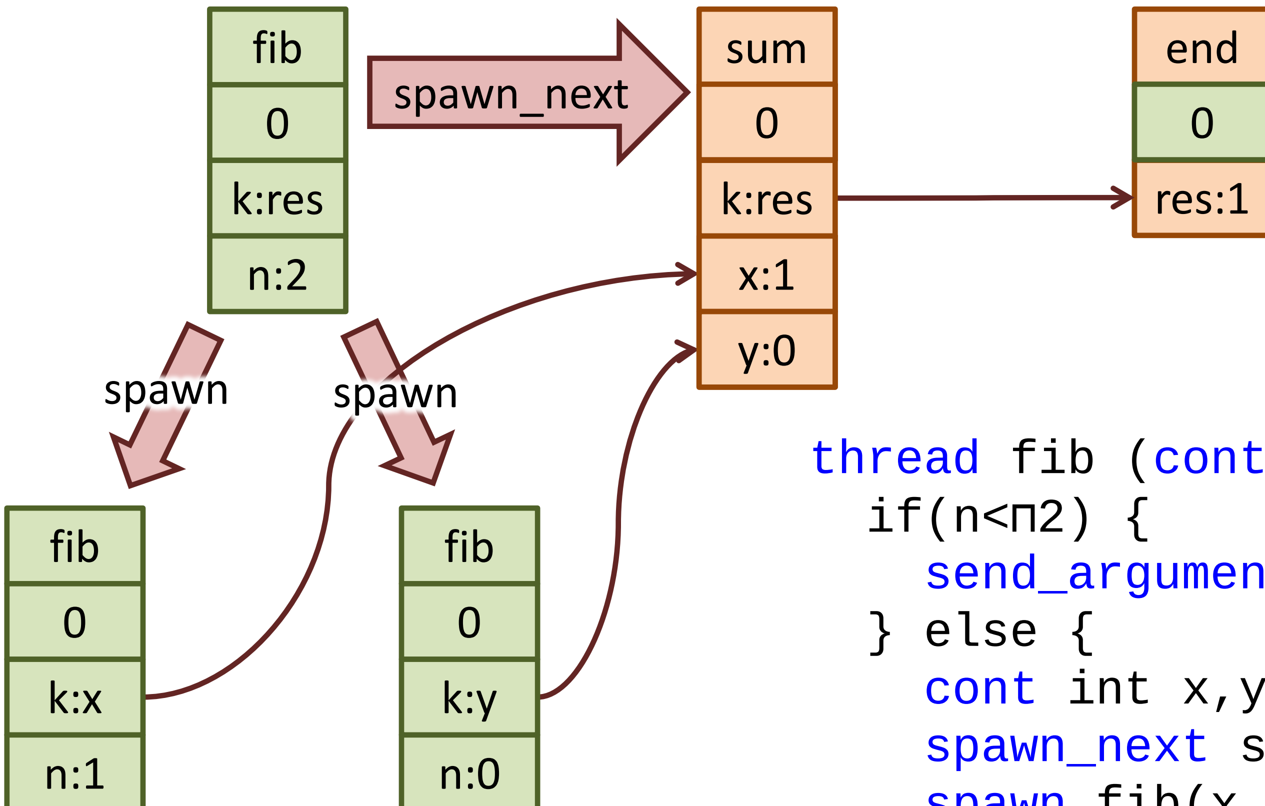
Creates a continuation

Keyword: send_argument

```
thread fib (cont int k, int n) {
  if(n<2) {
    send_argument(k, n);
  } else {
    cont int x,y;
    spawn_next sum(k, ?x, ?y);
    spawn fib(x, n-1);
    spawn fib(y, n-2);
  }
}
thread add(cont int k, int x, int y) {
  send_argument(k, x+y);
}
```

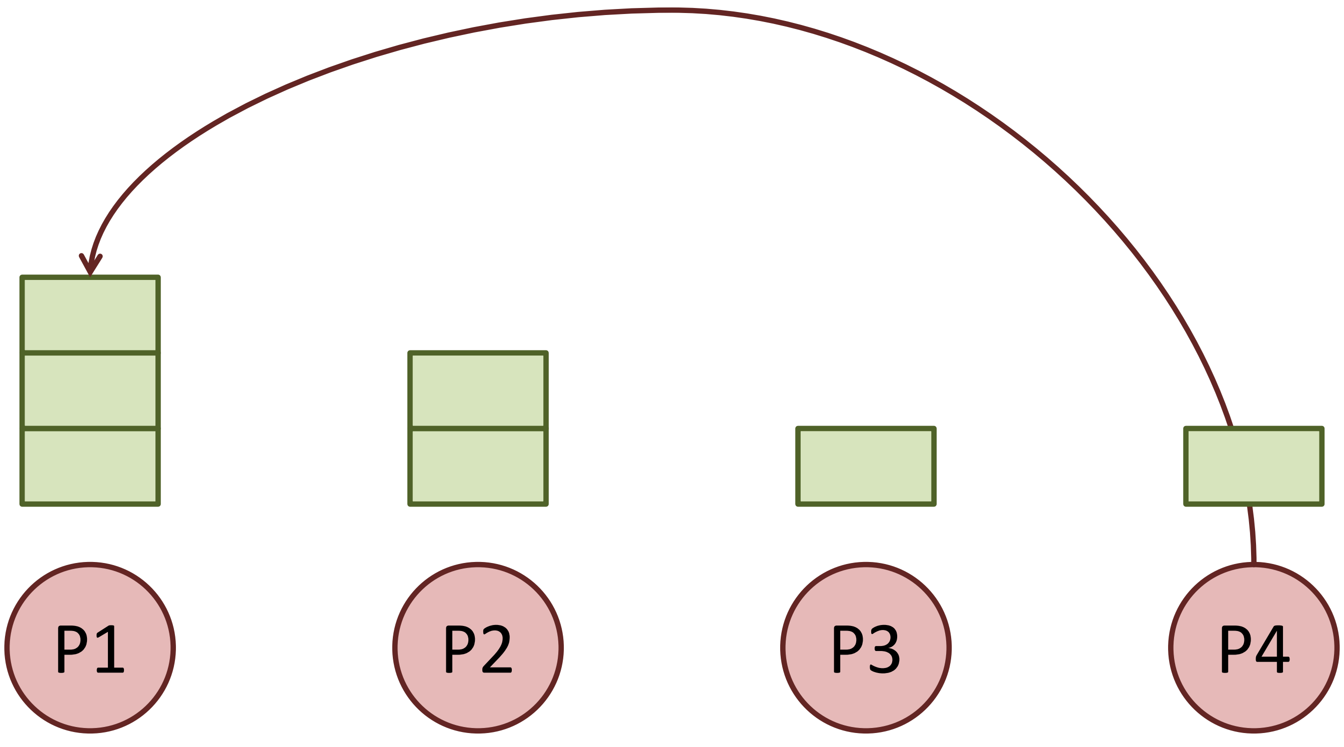
Fills a parameter in a closure

fib(res, 2)

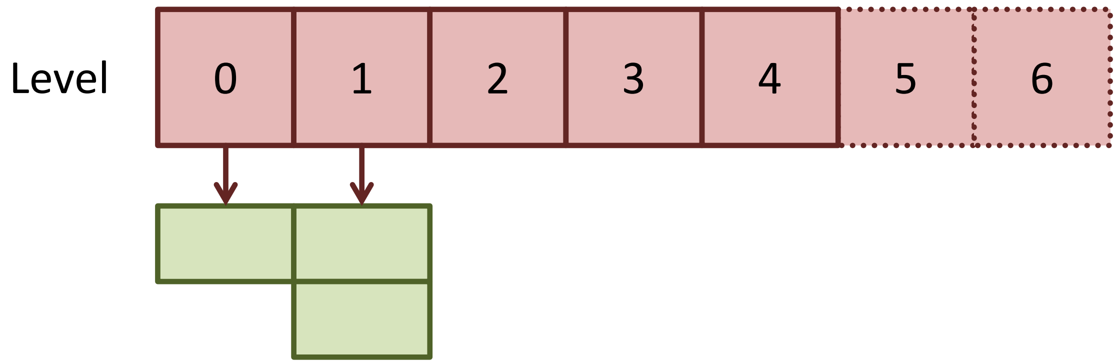



```
thread fib (cont int k, int n) {
  if(n<2) {
    send_argument(k, n);
  } else {
    cont int x,y;
    spawn_next sum(k, ?x, ?y);
    spawn fib(x, n-1);
    spawn fib(y, n-2);
  }
}
```

Workstealing Scheduler



Ready-Queue



- Array of linked lists
  - ith Element contains all ready closures of level i
- spawn → create closure for next deeper level
- spawn_next → create closure for same level

Scheduler

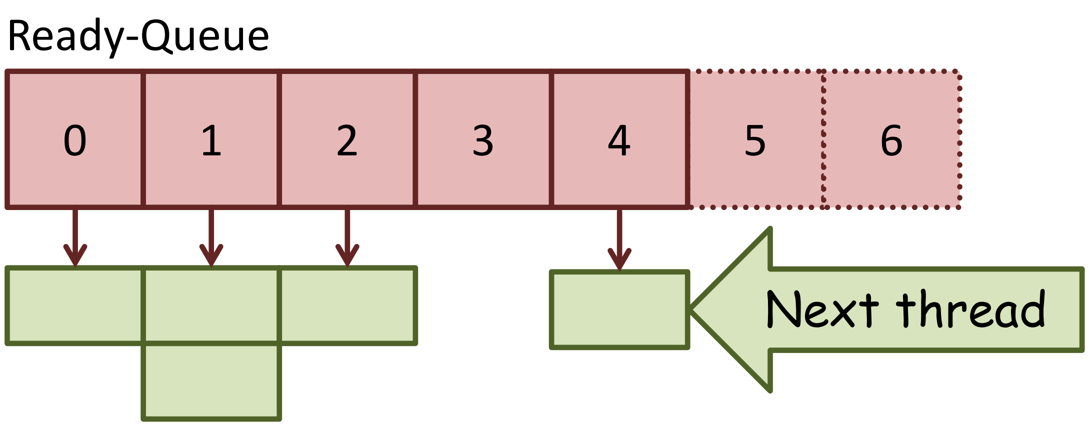
- Every Processor has own:
  - Scheduler
  - Ready-Queue

- Invoked when thread ends
  - Schedules or steals another thread

Ready-Queue NOT empty

- Get thread of the deepest level
  - Like depth first search in a graph

- No communication needed

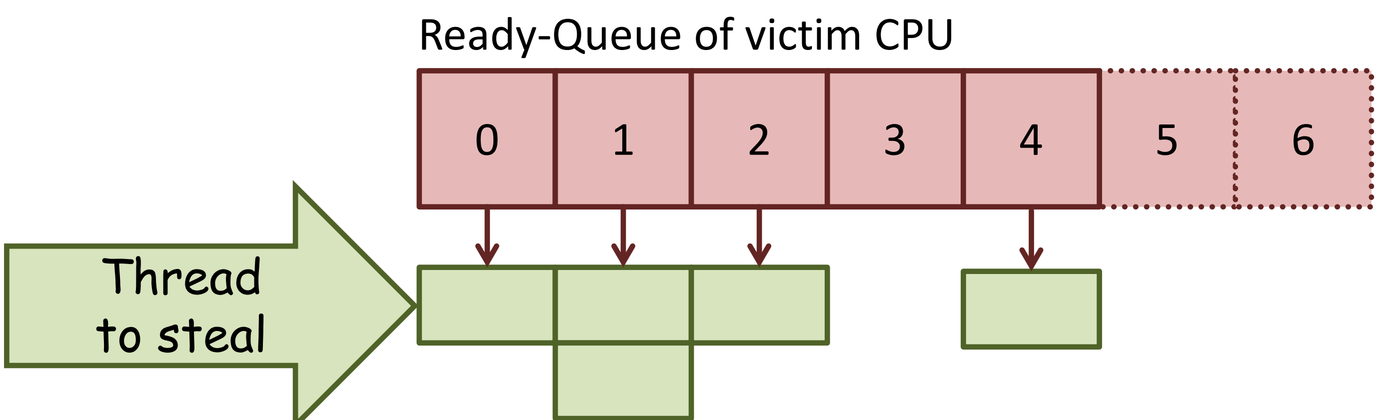


Ready-Queue empty

1. Select a random victim Processor

2. Check if ready thread is available

3. Steal thread with lowest level

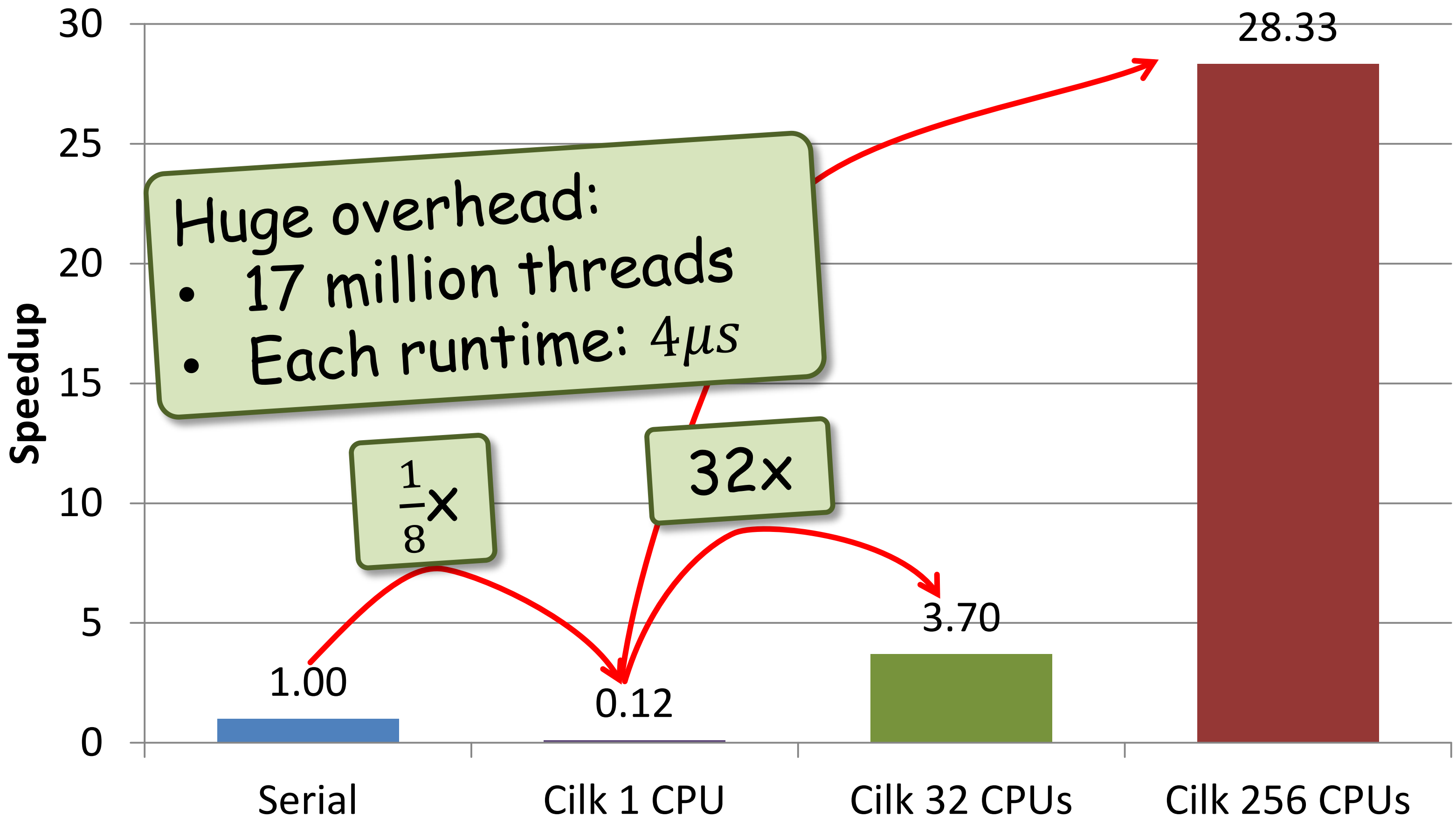


Evaluation

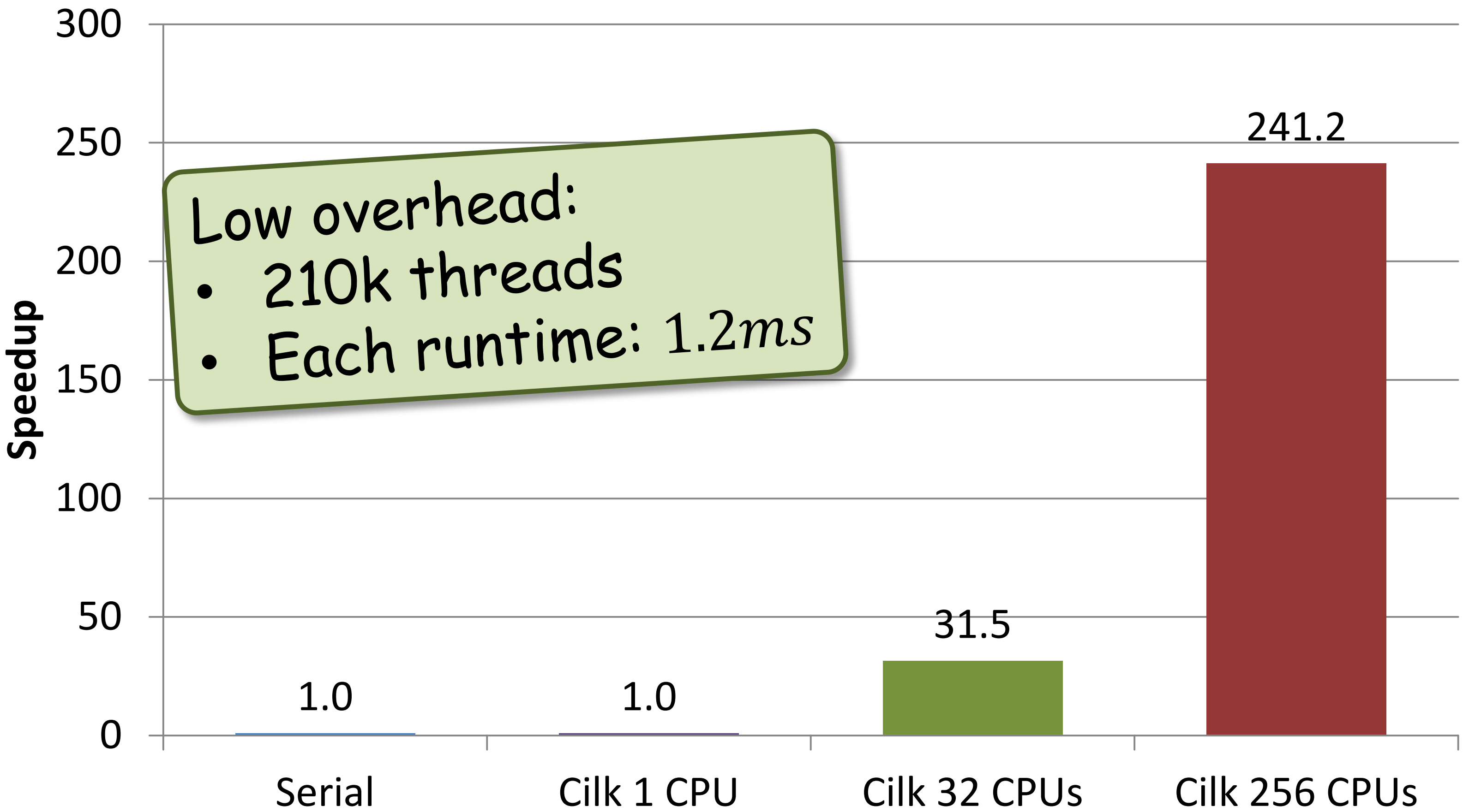
- Example Programs
  - Runtime serial
  - Runtime cilk
    - 1 CPU
    - 32 CPUs
    - 256 CPUs

- Run on CM5 supercomputer
  - 32MHz IBM SPARC CPUs

33rd Fibonacci Number

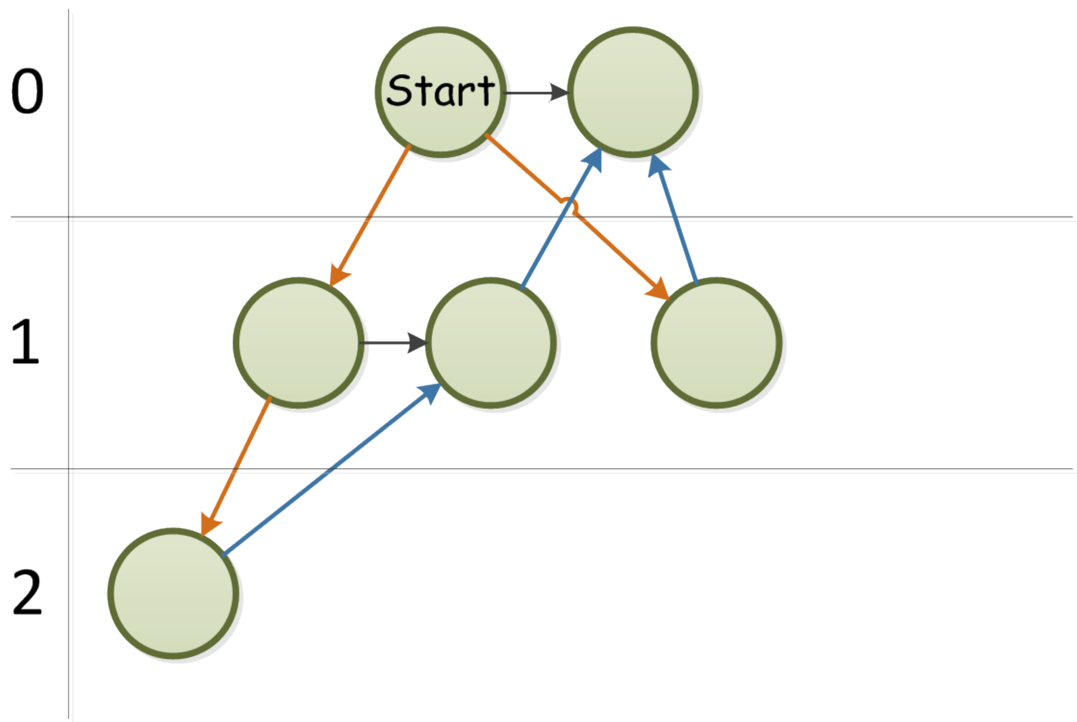


Queens on 15x15 Tiles



Proofs

- Only for Strict Cilk program
  - Only send arguments to ist parent’s successor



Proofs

1. Uses at most $S_1 * P$ space on P CPUs
   S_1: used space on 1 CPU

2. Shallowest thread is best to steal for a program with no more than one successor

3. Expected runtime on P CPUs: $\mathrm{E}\left[\mathrm{T_p}\right] = O\left(\frac{T_1}{P} + T_\infty\right)$ for a program with no more than one successor
   T_1: time on 1 CPU,
   T_∞: time on ∞ CPUs

4. Expected communication: $O(T_\infty P S_{max})$
   S_{max}: maximum closure size
   T_∞: critical path length

Conclusion

- Pro
  - Guaranteed runtime & space usage
  - Good performance
    - Critical Path short compared to total work
- Contra
  - Only suitable for tree like computations
  - Continuations confusing
  - No shared memory

Cilk^{TM} Plus

- Maintained by Intel©

- Only 3 keywords
  - Cilk_spawn
  - Cilk_sync
  - Cilk_for

- GCC 4.7 branch «cilkplus»

Questions?

Comments?