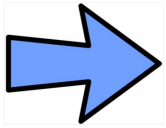
CSE502: Foundations of Parallel Programming

Lecture 05: Introduction to Dynamic Task Creation and Termination

Vivek Kumar
Computer Science and Engineering
IIIT Delhi
vivekk@iiitd.ac.in

Today's Class



Concurrency platforms

- Task creation and termination using `async-finish` statements
- Quiz-1

Pthread Implementation of Fibonacci

```c
#include <inttypes.h>
#include <pthread.h>
#include <stdio.h>
#include <stdlib.h>

uint64_t fib(uint64_t n) {
  if (n < 2) {
    return n;
  } else {
    uint64_t x = fib(n-1);
    uint64_t y = fib(n-2);
    return (x + y);
  }
}

typedef struct {
  uint64_t input;
  uint64_t output;
} thread_args;

void *thread_func(void *ptr) {
  uint64_t i =
    ((thread_args *) ptr)->input;
  ((thread_args *) ptr)->output = fib(i);
  return NULL;
}
```

```c
int main(int argc, char *argv[]) {
  pthread_t thread;
  thread_args args;
  int status;
  uint64_t result;

  if (argc < 2) { return 1; }
  uint64_t n = strtoul(argv[1], NULL, 0);
  if (n < 30) {
    result = fib(n);
  } else {
    args.input = n-1;
    status = pthread_create(&thread,
                            NULL,
                            thread_func,
                            (void*) &args);
    // main can continue executing
    if (status != NULL) { return 1; }
    result = fib(n-2);
    // Wait for the thread to terminate.
    status = pthread_join(thread, NULL);
    if (status != NULL) { return 1; }
    result += args.output;
  }
  printf("Fibonacci of %" PRIu64 " is %" PRIu64 ".\n",
         n, result);
  return 0;
}
```

What are the issues in this program?

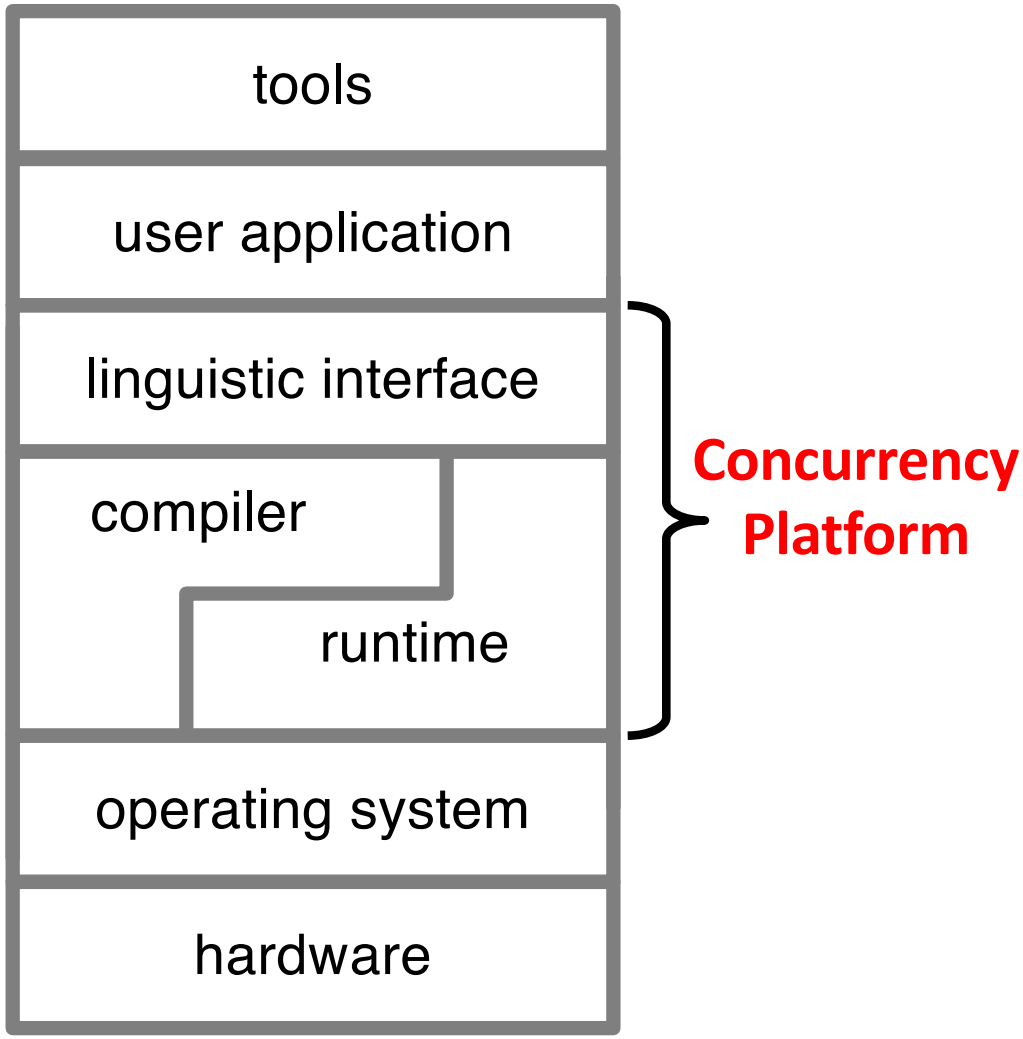
Source: http://classes.engineering.wustl.edu/cse539/web/lectures/lec01_intro.pdf

Issues with Pthreads

Overhead	The cost of creating a thread >10^4 cycles ⇒ coarse-grained concurrency. (Thread pools can help.)
Scalability	Fibonacci code gets at most about 1.5 speedup for 2 cores. Need a rewrite for more cores.
Modularity	The Fibonacci logic is no longer neatly encapsulated in the fib() function.
Code Simplicity	Programmers must marshal arguments (shades of 1958!) and engage in error-prone protocols in order to load-balance.

Source: http://classes.engineering.wustl.edu/cse539/web/lectures/lec01_intro.pdf

Concurrency Platforms

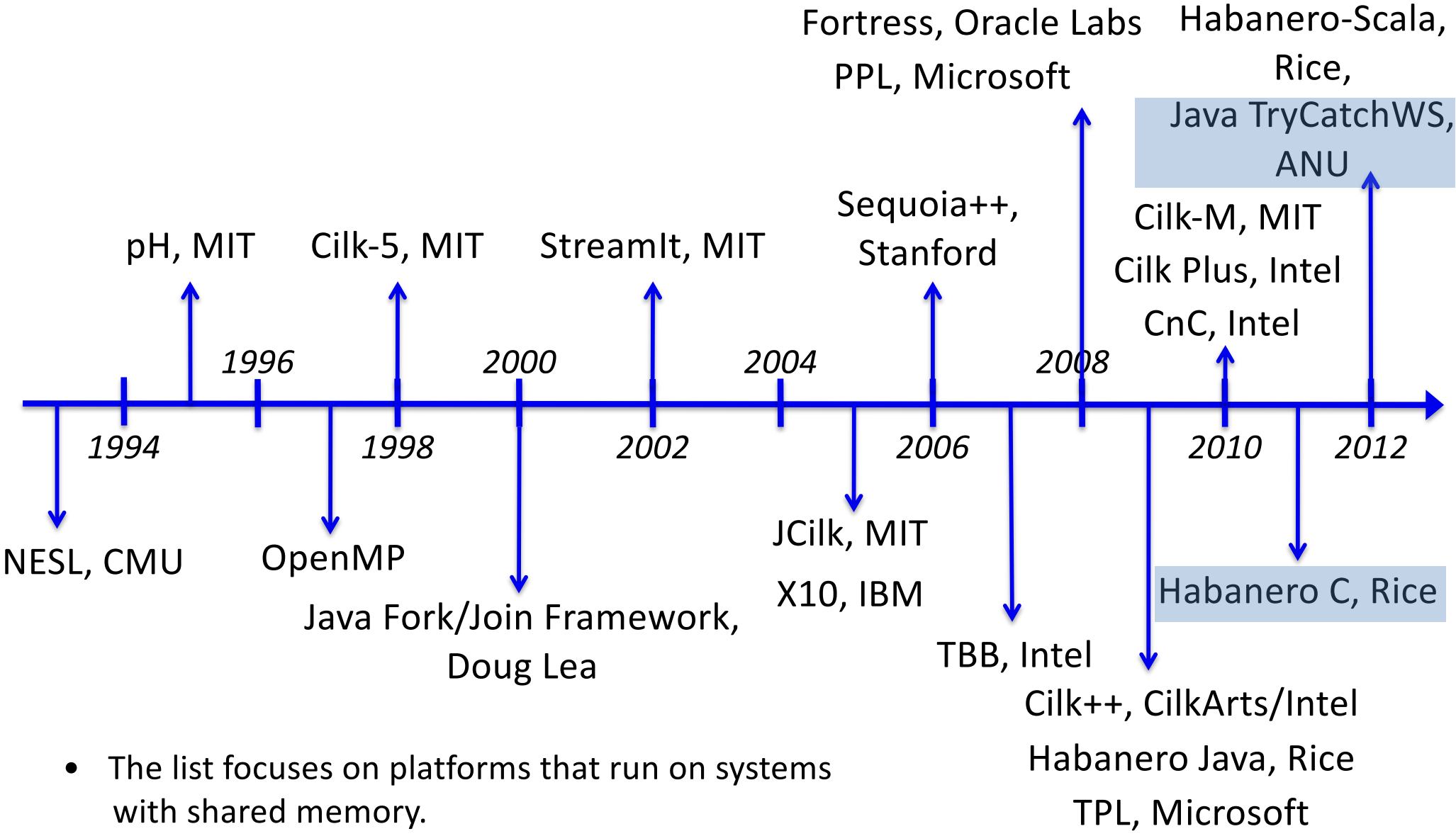


A concurrency platform should provide:

- an interface for specifying the ***logical parallelism*** of the computation;
- a runtime layer to automate scheduling and synchronization; and
- guarantees of performance and resource utilization competitive with hand-tuned code.

Source: http://classes.engineering.wustl.edu/cse539/web/lectures/lec01_intro.pdf

Modern Concurrency Platforms*



- The list focuses on platforms that run on systems with shared memory.

Source: http://classes.engineering.wustl.edu/cse539/web/lectures/lec01_intro.pdf

Today's Class

- Concurrency platforms
- Task creation and termination using `async-finish` statements
- Quiz-1

Types of Tasks

- Synchronous
  - Blocks until the task execution is complete
- Asynchronous (async)
  - Doesn’t blocks for the task to complete its execution

Async-Finish to your Sunday Tasks

```
finish {
        async { Complete your FPP assignment   }
        async { Wash your clothes in washing machine  }
}
finish {
        async { Watch movies on laptop  }
        async { Talk to father
                Talk to mother  }
        async { Buy fruits online using your smartphone }
        async { Make your bed  }
}
```

Post on Facebook that you are done with all your tasks!

Async-Finish to your Sunday Tasks

Post on Facebook that you are done with all your tasks!

Complete your FPP assignment

Wash your clothes in washing machine

Watch movies on laptop

Talk to father

Buy fruits online using your smartphone

Talk to mother

Make your bed

Async-Finish to your Sunday Tasks

Complete your FPP assignment

Wash your clothes in washing machine

Watch movies on laptop

Talk to father

Talk to mother

Buy fruits online using your smartphone

Make your bed

Applying statement reordering

Post on Facebook that you are done with all your tasks!

Async-Finish to your Sunday Tasks

```
finish {
        async { Complete your FPP assignment   }
        async { Wash your clothes in washing machine  }
}
finish {
        async { Watch movies on laptop  }
        async { Talk to father
                Talk to mother  }
        async { Buy fruits online using your smartphone }
        async { Make your bed  }
}
```

Post on Facebook that you are done with all your tasks!

Async-Finish to your Sunday Tasks

```
finish {
    async { Wash your clothes in washing machine }
    async { Complete your FPP assignment }
}
finish {
    async { Watch movies on laptop }
    async { Talk to father
            Talk to mother }
    async { Buy fruits online using your smartphone }
    async { Make your bed }
}
```

Applying statement reordering

Post on Facebook that you are done with all your tasks!

Async and Finish Statements for Task Creation and Termination (Pseudocode)

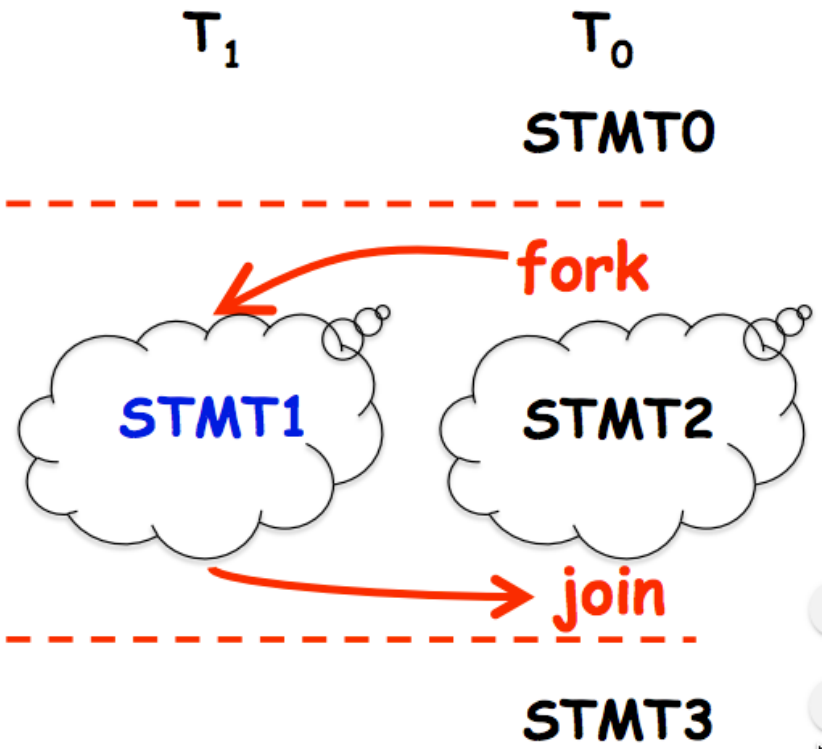
async S

- Creates a new child task that executes statement S

finish S

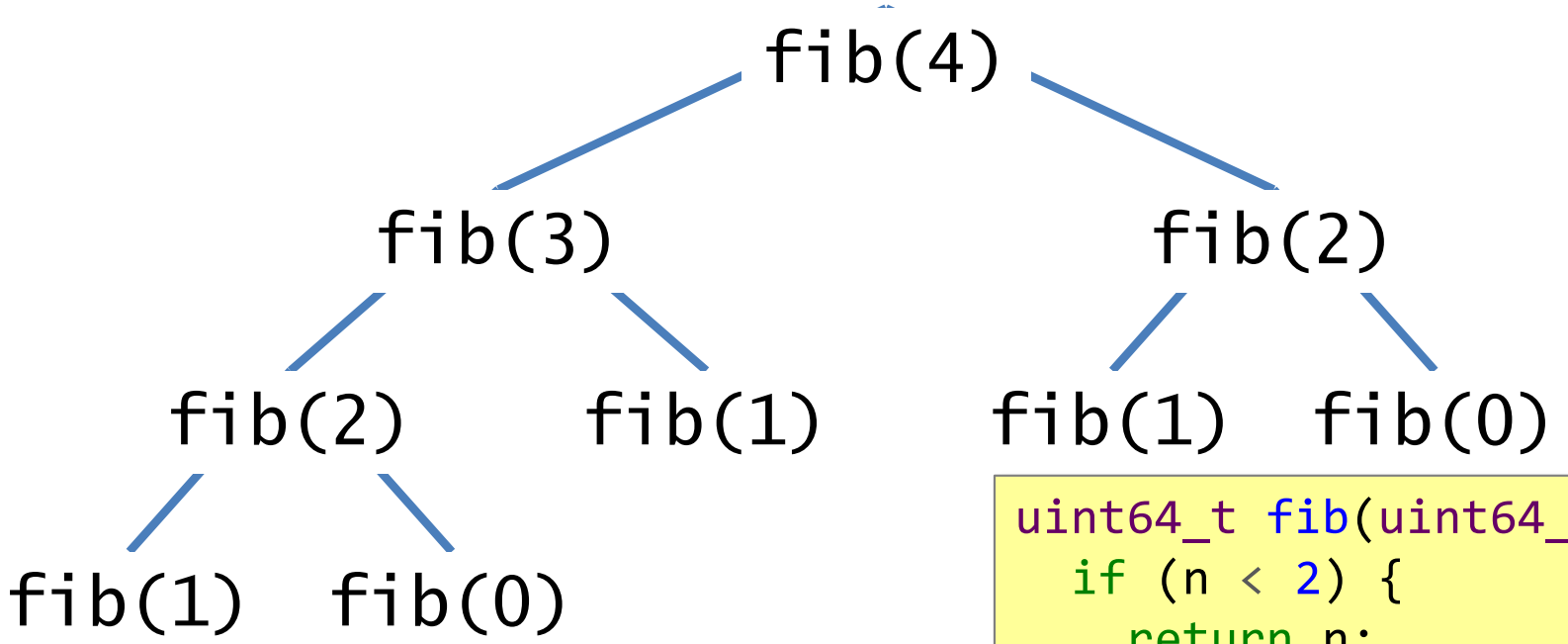
- Execute S but wait until all async in S's scope have terminated

```
// T0 (Parent task)
STMT0;
finish {    //Begin finish
  async {
    STMT1; //T1 (Child task)
  }
  STMT2;    //Continue in T0
            //Wait for T1
}           //End finish
STMT3;      //Continue in T0
```



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

How to Parallelize Fibonacci using async and finish Constructs ?




```
uint64_t fib(uint64_t n) {
  if (n < 2) {
    return n;
  } else {
    uint64_t x = fib(n-1);
    uint64_t y = fib(n-2);
    return (x + y);
  }
}
```

Source: http://classes.engineering.wustl.edu/cse539/web/lectures/lec01_intro.pdf

Serial Elision

- This is the *serial equivalence* of the async-finish based parallel program
  - Obtained by removing all async and finish constructs
  - **High productivity** – In most cases, async-finish can be simply added in any sequential algorithm (without any significant changes) to get the corresponding parallel version of the algorithm

```
uint64_t fib(uint64_t n) {
  if (n < 2) {
    return n;
  } else {
    uint64_t x, y;
    finish {
      async { x = fib(n-1); }
      y = fib(n-2);
    }
    return (x + y);
  }
}
```

```
uint64_t fib(uint64_t n) {
  if (n < 2) {
    return n;
  } else {
   uint64_t x, y;
   {
     x = fib(n-1);
     y = fib(n-2);
   }
   return (x + y);
  }
}
```

Next Class (Saturday)

- Tutorial on Habanero-C (HClib) usage
  - Important for upcoming labs
- Ideal parallelism and revisit computation graph

Acknowledgements

- Several of the slides used in this course are borrowed from the following online course materials:
  - Course COMP322, Prof. Vivek Sarkar, Rice University
  - Course COMP 422, Prof. John Mellor-Crummey, Rice University
  - Course CSE539S, Prof. I-Ting Angelina Lee, Washington University in St. Louis
- Contents are also borrowed from following sources:
  - “Introduction to Parallel Computing” by Ananth Grama, Anshul Gupta, George Karypis, and Vipin Kumar. Addison Wesley, 2003
  - https://computing.llnl.gov/tutorials/parallel_comp/
  - https://images.google.com/