

Intro to Parallel and Concurrent Programming

Work Stealing Schedulers



[WashU CSE 2301](#)

Prof. [Dennis Cosgrove](#)

#27: Thu, Dec 04, 2025

Reminder: Friday 10-Noon Office Hours January 110



Course Evals

<https://one.washu.edu/task/all/course-evals>

Worksheet

```
public static double hypotenuse(double a, double b) {
    Lock lock = new ReentrantLock();
    double[] sum = {0.0};
    Future<Void> future = void_fork(() -> {
        lock.lock();
        try {
            sum[0] += a * a;
        } finally {
            lock.unlock();
        }
    });
    sum[0] += b * b;
    join(future);
    return Math.sqrt(sum[0]);
}
```

Worksheet

```
public static double hypotenuse(double a, double b) {
    Lock lock = new ReentrantLock();
    double[] sum = {0.0, 0.0};
    sum[0] += b * b;
    Future<Void> future = void_fork(() -> {
        lock.lock();
        try {
            sum[0] += a * a;
        } finally {
            lock.unlock();
        }
    });
    join(future);
    return Math.sqrt(sum[0]);
}
```

Data Race

```
public static double hypotenuse(double a, double b) {  
    Lock lock = new ReentrantLock();  
    double[] sum = {0.0};  
    Future<Void> future = void_fork(() -> {  
        lock.lock();  
        try {  
            sum[0] += a * a;  
        } finally {  
            lock.unlock();  
        }  
    });  
    sum[0] += b * b;  
    join(future);  
    return Math.sqrt(sum[0]);  
}
```

Data Race

```
public static double hypotenuse(double a, double b) {  
    Lock lock = new ReentrantLock();  
    double[] sum = {0.0};  
    Future<Void> future = void_fork(() -> {  
        lock.lock();  
        try {  
            sum[0] += a * a;  
        } finally {  
            lock.unlock();  
        }  
    });  
    sum[0] += b * b;  
    join(future);  
    return Math.sqrt(sum[0]);  
}
```

Safe

```
public static double hypotenuse(double a, double b) {
    Lock lock = new ReentrantLock();
    double[] sum = {0.0};
    Future<Void> future = void_fork(() -> {
        lock.lock();
        try {
            sum[0] += a * a;
        } finally {
            lock.unlock();
        }
    });
    lock.lock();
    try {
        sum[0] += b * b;
    } finally {
        lock.unlock();
    }
    join(future);
    return Math.sqrt(sum[0]);
}
```


synchronized

```
public static double hypotenuse(double a, double b) {  
    Lock lock = new ReentrantLock();  
    double[] sum = {0.0};  
    Future<Void> future = void_fork(() -> {  
        synchronized (sum) {  
            sum[0] += a * a;  
        }  
    });  
    synchronized (sum) {  
        sum[0] += b * b;  
    }  
    join(future);  
    return Math.sqrt(sum[0]);  
}
```

share mutable data is the problem

```
public double hypotenuse(double a, double b) {  
    Future<Double> a2Future = fork(() -> {  
        return a * a;  
    });  
    double b2 = b * b;  
    double a2 = join(a2Future);  
    return Math.sqrt(a2 + b2);  
}
```

not enough work

```
public double hypotenuse(double a, double b) {  
    return Math.sqrt(a*a + b*b);  
}
```

just use the hypot method :)

```
public double hypotenuse(double a, double b) {  
    return Math.hypot(a, b);  
}
```

AAA

Synthesis: Stealing is good!



S&Q: Why was the prep so long?

This was a long lecture... could we get some sort of summary/rewording of what work stealing is?

You might not want to take CSE 425.

S&Q: Does the library that we use for this course use work stealing? Have we been stealing parents or children all along and we just haven't known about it?

Work Stealing: Why Pablo Halpern Now?

- starts with review of course (but critically different)
- ends w/ how it works

S&Q: So essentially, parallelism is just stealing work from other tasks? Is that correct?

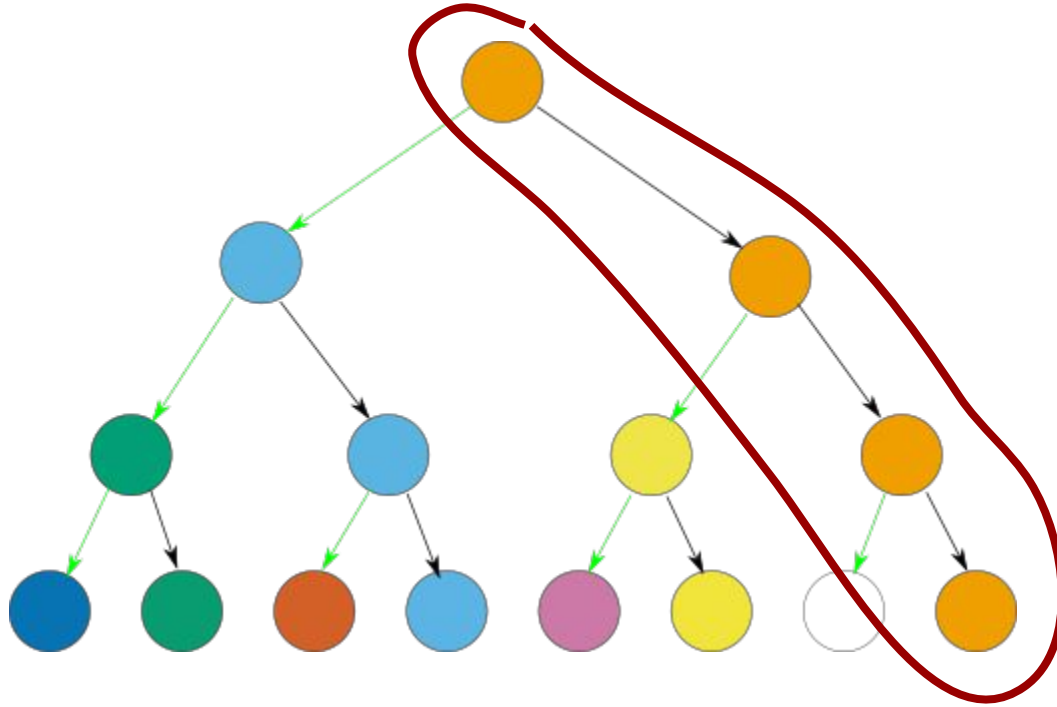
fork leaves work behind for others to steal

S&Q: How can we make sure that work that should be done in the same thread is not stolen?

write it sequentially

by definition, when you say something can run in parallel, it can run in parallel. if it cannot run in parallel, do not fork.

S&Q: What is the continuation work in a parallel for loop ?



S&Q: How does the same work stealing concepts apply in Java? Is there any package to deal with that?

[Executors.newWorkStealingPool\(\)](#)

note: undoubtedly child stealing

S&Q: This might be slightly off-topic, but what's your take on the choice between speed and power consumption? Mr. Halpern mentions that multi-core computation is more power-efficient than single-core. But if a task could be done much faster on a single-core, why should we sacrifice speed for power efficiency?

Power

Power \sim Capacitance * Voltage² * Frequency

Maximum Frequency is capped by Voltage

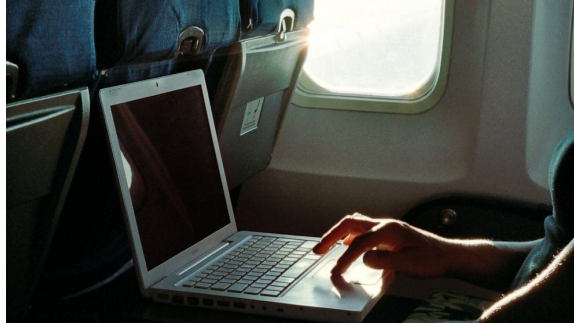
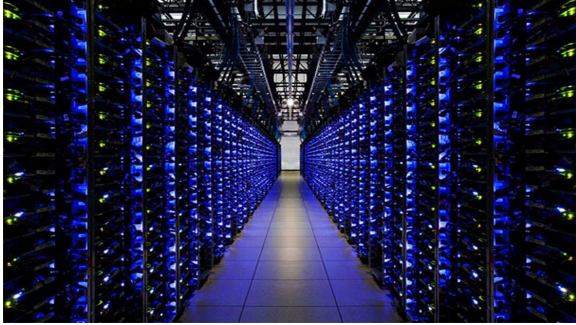
Power is proportional to Frequency³

8 Power == 8 Power

2 GHz	1 GHz
	1 GHz
	1 GHz
	1 GHz
	1 GHz
	1 GHz
	1 GHz
	1 GHz

- $2^3 = (1^3 + 1^3 + 1^3 + 1^3 + 1^3 + 1^3 + 1^3 + 1^3) = 8$

Power is a big deal



not to mention: with **power** comes **heat**



Goal 1: greedy level performance

randomized work stealing gets us close to greedy

Greedy Scheduler

“In a nutshell, a greedy scheduler is a scheduler in which no processor is idle if there is more work it can do.”

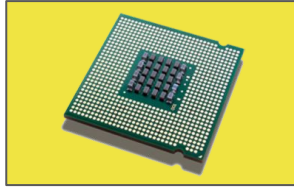
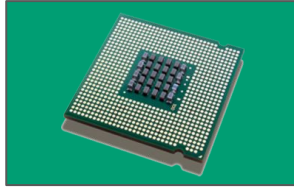
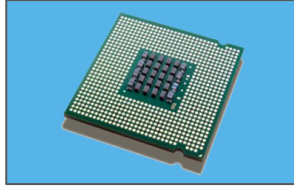
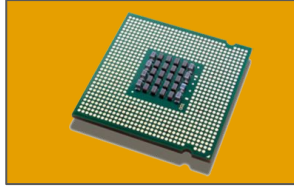
- Guy E. Blelloch

<http://www.cs.cmu.edu/~guyb/>

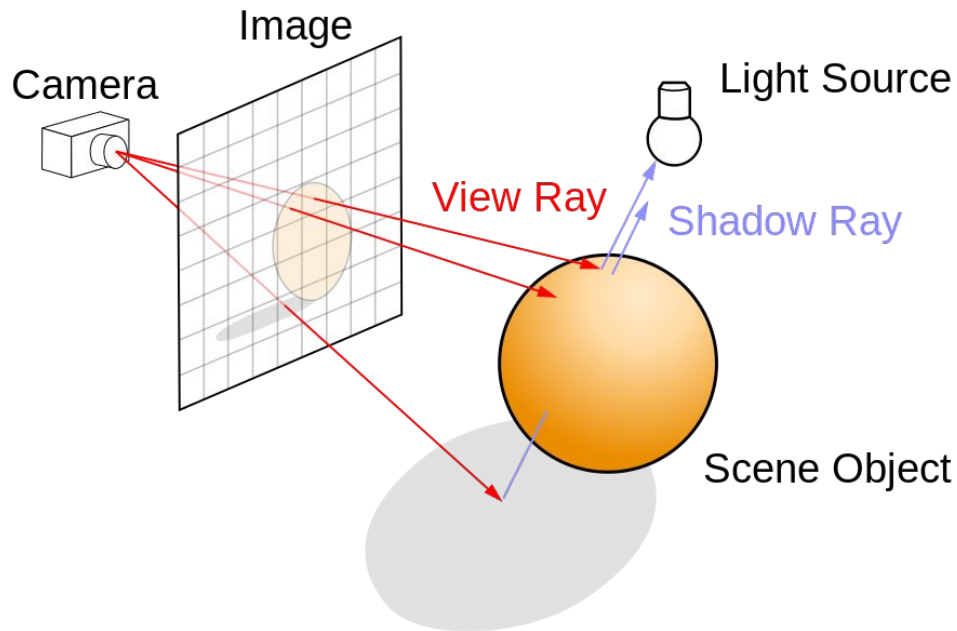


Goal 2: make fork low overhead

Imagine 4 Processor Machine



Ray tracing



[https://en.wikipedia.org/wiki/Ray_tracing_\(graphics\)](https://en.wikipedia.org/wiki/Ray_tracing_(graphics))

Pixel-planes: parallel ray tracing since 1980



<http://www.cs.unc.edu/techreports/85-001.pdf>

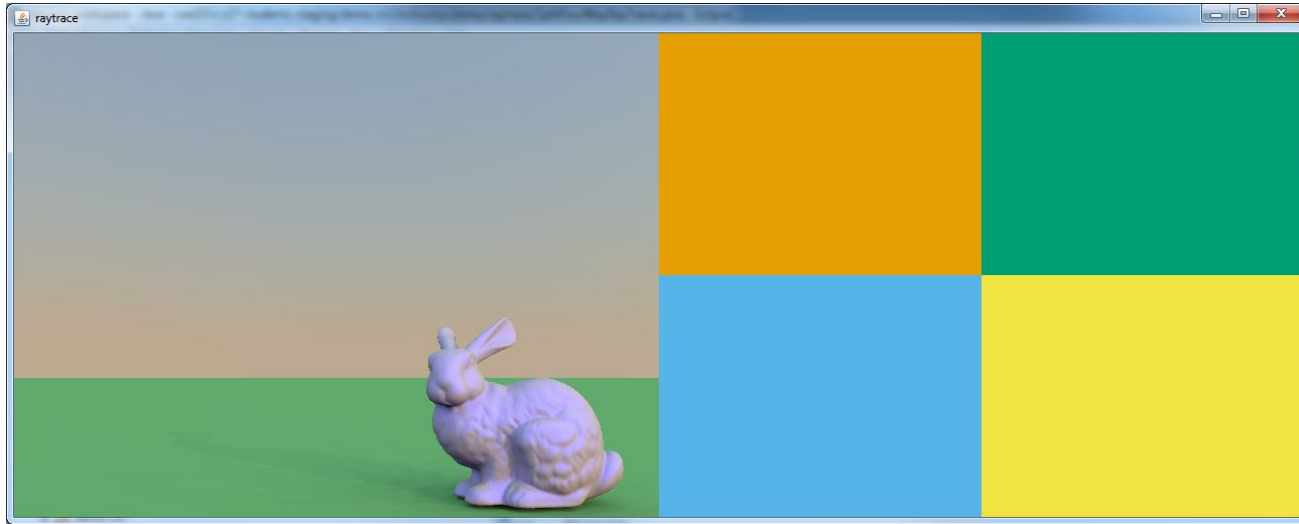
<http://www.cs.unc.edu/~pxfl/history.html>

“Pxpl2’s screen resolution was 4x64 pixels, and only 16 bits of memory per pixel, and was only able to display a few polygons per second.”

Demo SplitFourWayRayTracer



Command+Shift+T “SplitFourWayRayTracer”

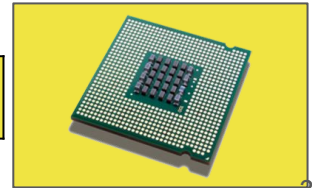
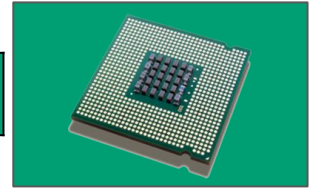
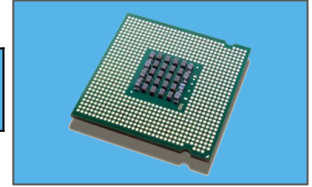
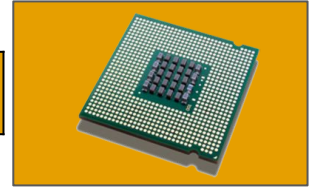


Schedulers

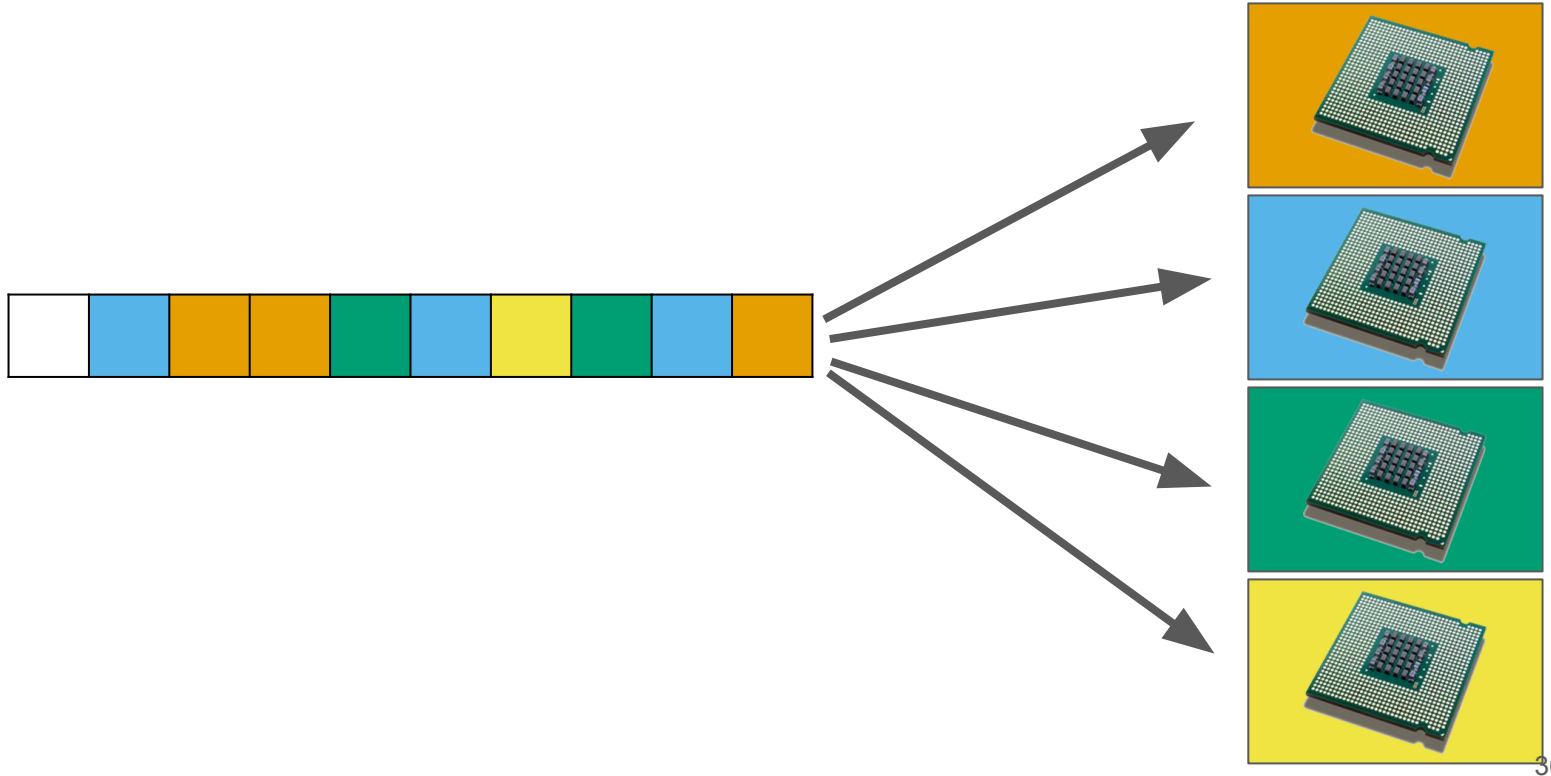
static scheduler

allocate the perfect amount of work to each processor

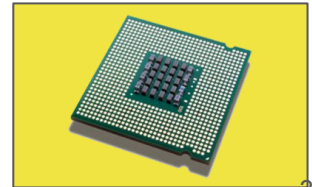
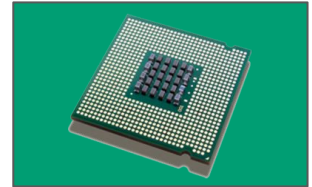
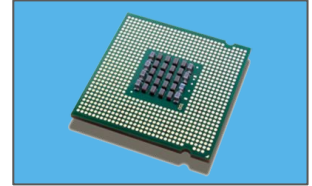
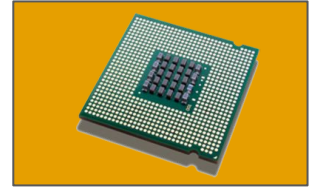
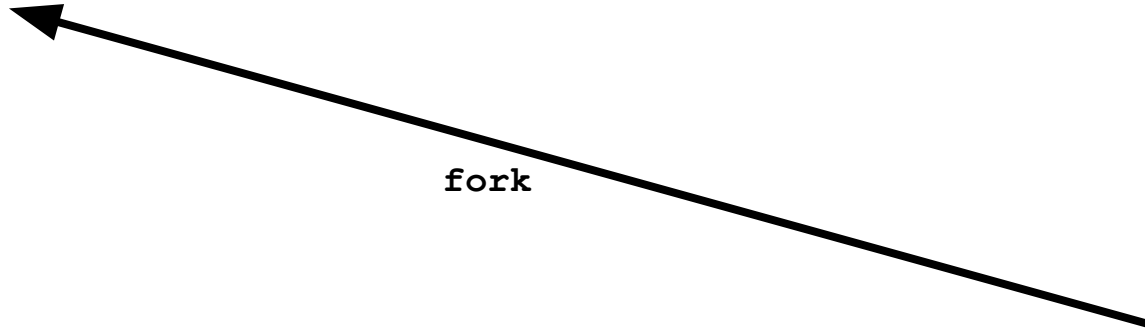
raytracing split attempts to do this but fails



Centralized Work Queue



Centralized Work Queue



Centralized Work Queue

Positives

- balances workload well

Drawbacks

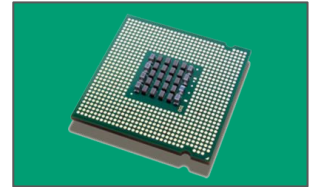
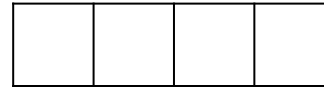
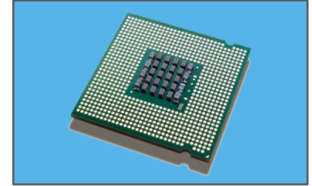
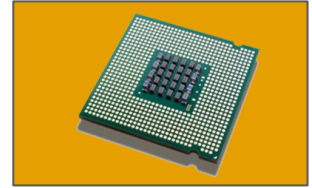
- shared mutable data
- unbounded

Good for big tasks

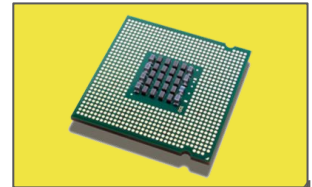
*S&Q: Can we go over child and parent stealing
some more? Confused about the deque.*

Distributed Work Queues

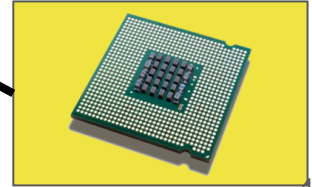
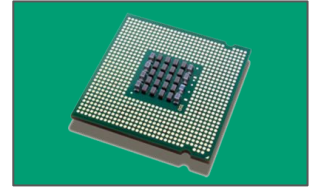
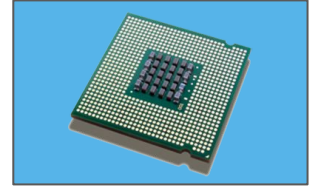
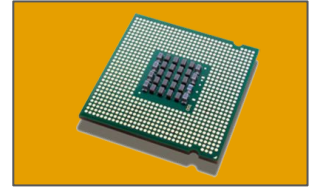
Work Sharing



fork

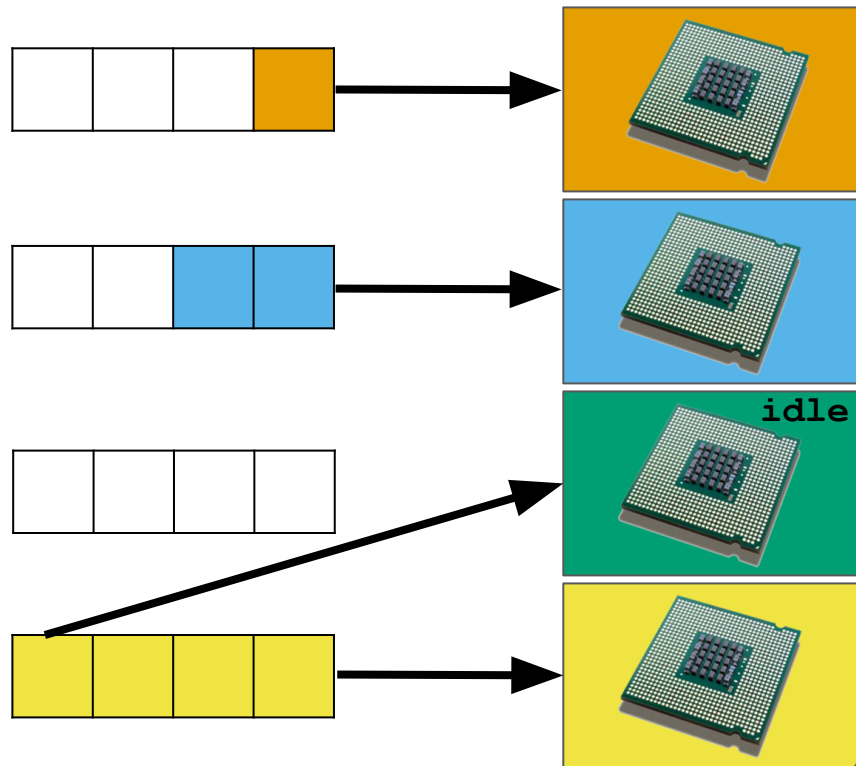


Work Sharing



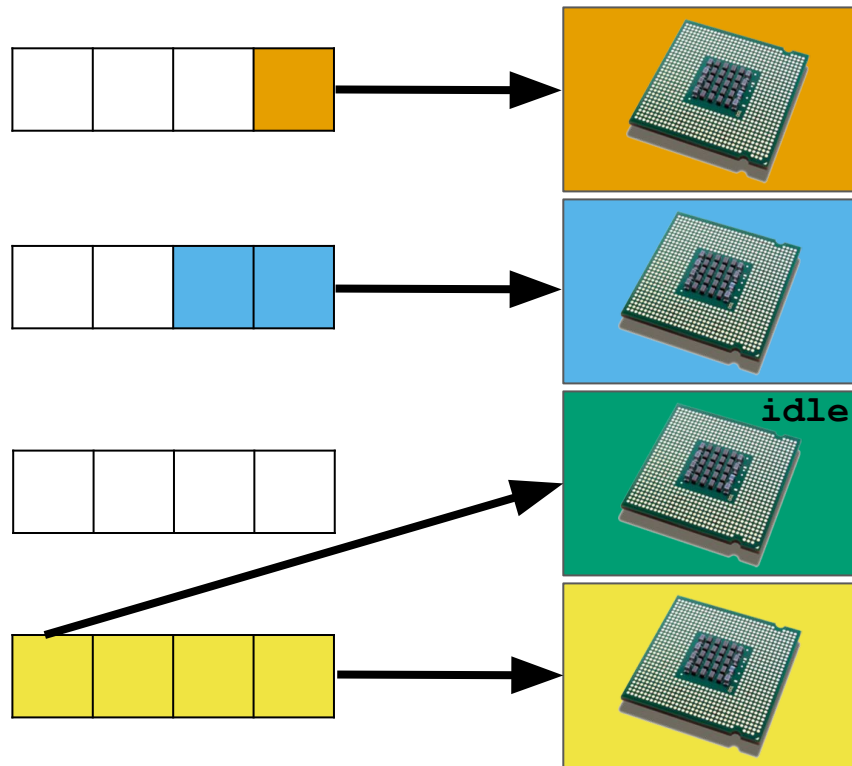
fork

Work Stealing



Work Stealing: how it works

- fork-> add to tail of your deque (double-ended queue)
- idle?
 - steal from random other's head
- at finish?
 - steal from your own tail
 - opposite end of what others are stealing (less contention)
 - it's very likely what you need to finish
 - likely cache "warm"



S&Q: I'm not sure if I just wasn't understanding the application, but I was a bit confused about how exactly work can steal from itself?

processor: *"Anyone want to go right? I'm going left"*

processor: ...

processor: *"Okay, I'm back. Do I still have work on my queue?"*

if so, steal the work from myself.

otherwise, steal the work from someone else.

Properties

- when balanced- you steal from yourself (nice)
- keep fork as cheap as possible
- allow steal to be expensive, if necessary (offload onto idle processor)

S&Q: Can you give examples of child steal and continuation steal? I don't quite understand the difference.

```
future = void_fork {  
    sum upper  
}  
sum lower  
join(future)
```

child stealing leaves **upper** behind to be stolen

continuation stealing leaves **lower** behind to be stolen

S&Q: In the video, Halpern mentions that he encourages programmers not to use thread local storage - I've used TLS in other classes and don't see too many issues with it; why might Halpern discourage it?

The continuation may be in another Thread!

demo ThreadLocalDemo

```
ThreadLocal<Integer> threadLocal = new ThreadLocal<>();
threadLocal.set(42);
System.out.println("a: " + threadLocal.get());
System.out.println("b: " + threadLocal.get());
future = void_fork(() -> {
    System.out.println("c: " + threadLocal.get());
});
System.out.println("d: " + threadLocal.get());
join(future)
System.out.println("e: " + threadLocal.get());
```


S&Q: What is the advantage of child stealing over continuation stealing?

S&Q: Why wouldn't we always do this?

I will interpret “*this*” as “*continuation (work) stealing*”.

child stealing vs continuation stealing

refers to which path you take

- child stealing: (leave fork for someone else)
 - + less special access required (library/jar only could do it)
 - + thread local works
 - + more predictable (continuation same task (cache, memory))
- continuation stealing (leave continuation for someone else)
 - - requires compiler support
 - + queue bounded
 - + mad crazy efficient to fork (push work on steal)
 - + looks like sequential on one processor

S&Q: Are there other reasons to choose one strategy over the other? It seems there would be times that child-stealing is better, otherwise why would anyone write that way?

Need to modify the compiler to get continuation stealing to work.

S&Q: Would you ever use child stealing in practice if continuation stealing is better theoretically?

Don't have access to the compiler? Child stealing

S&Q: I'm still unsure of why child stealing is much worse than continuation. It's kinda gone over but I do not understand what was going on there.

continuation stealing can have ridiculously low overhead

S&Q: All of this seems pretty intuitive to me. Designing this system from the ground up might have been hard but if we already are aware of the concepts of parallel programming it doesn't seem to hard to grasp. Is there some crazy efficiency or algorithm I am missing here

one important goal is to keep the overhead down

holy smokes does continuation stealing do this well

S&Q: How close to approximations of greedy schedulers come to achieving run time of that of non greedy schedulers? Is the difference comparable, or would being able to find truly greedy schedulers quickly be game changing?

another is [provably good performance bounds](#)

S&Q: Can you go over some of the different circumstances when we would prefer centralized work stealing queues vs. child stealing vs. continuation stealing?

S&Q: Are continuations implemented in Java?

I'll assume “continuation stealing”.

My implementation leverages `ExecutorService`.

One could build continuation stealing, but it would require modifying the Java byte code (which could be done with AspectJ).

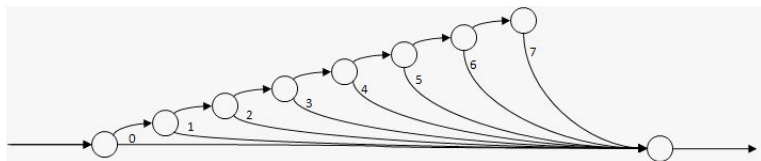
*** How is work stealing performance affected?

```
futures = [N]
for int i=0; i<N; i++
    futures[i] = void_fork
                doSomething
for future in futures
    join future
```

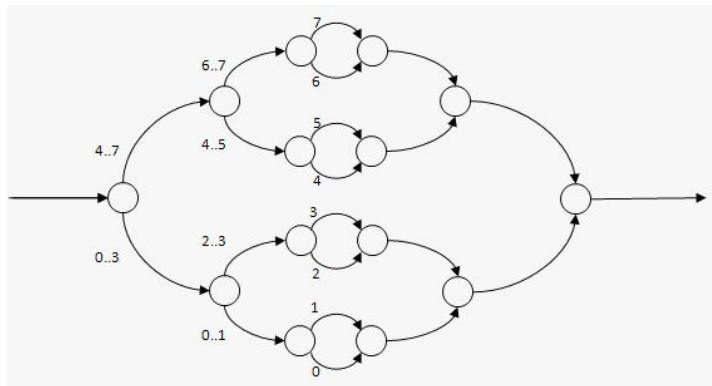
```
join_void_fork_loop 0, N, (i)->
    doSomething
```

Computation Graphs

```
futures = [N]
  for int i=0; i<N; i++
    futures[i] = void_fork
      doSomething
  for future in futures
    join future
```

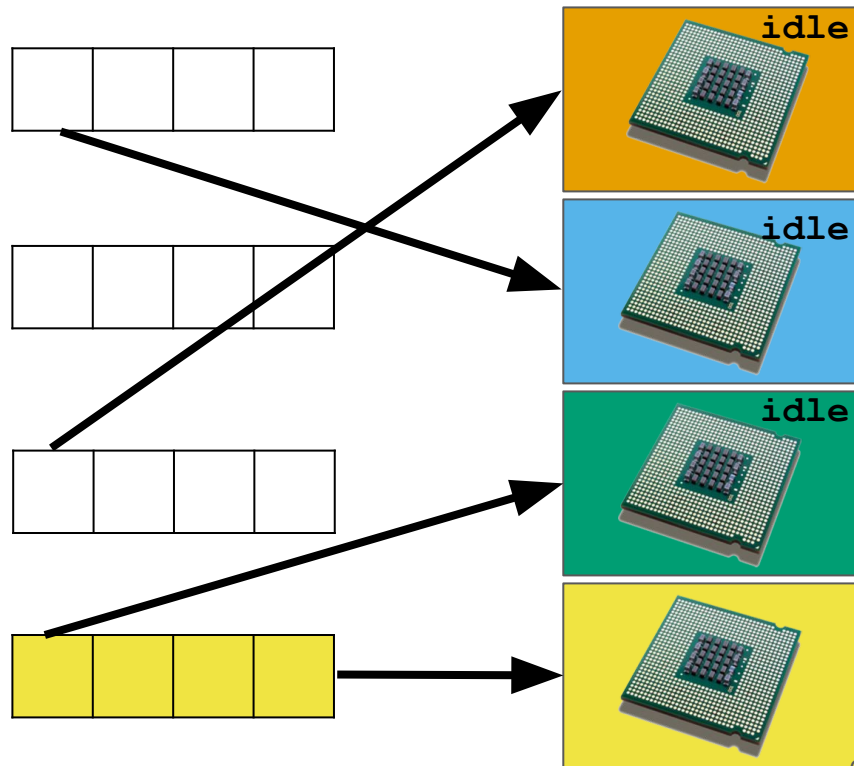
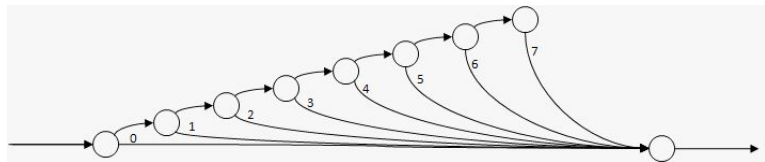


```
join_void_fork_loop 0, N, (i)->
  doSomething
```



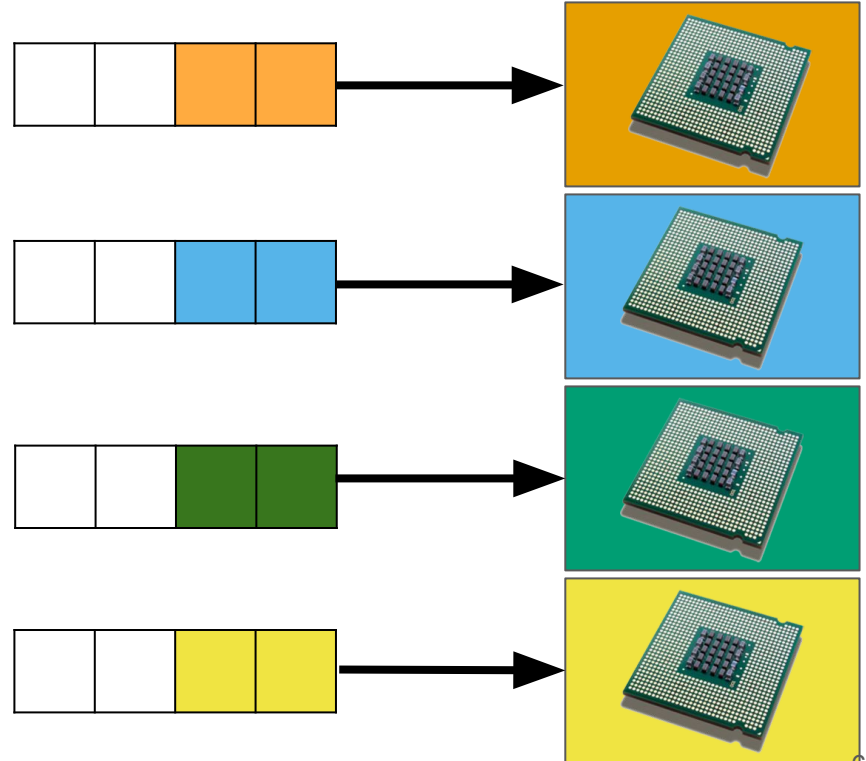
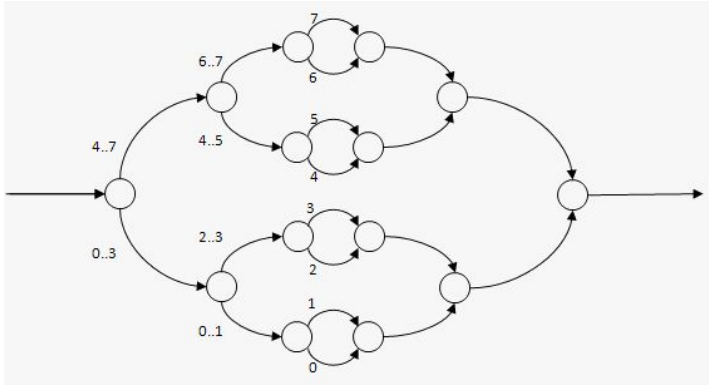
Work Queues

```
futures = [N]
  for int i=0; i<N; i++
    futures[i] = void_fork
      doSomething
  for future in futures
    join future
```



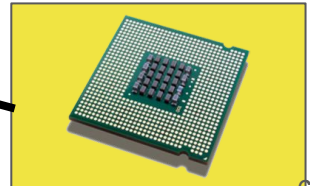
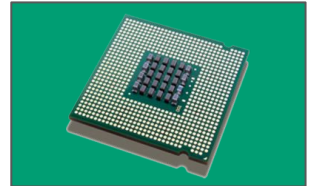
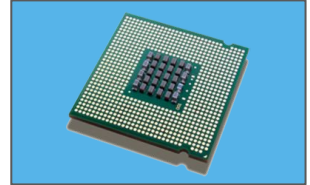
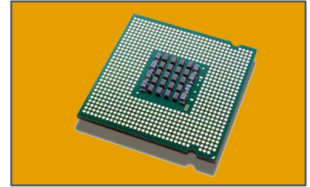
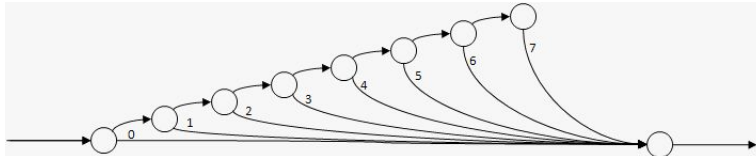
Work Queues

```
join_void_fork_loop 0, N, (i) ->  
    doSomething
```

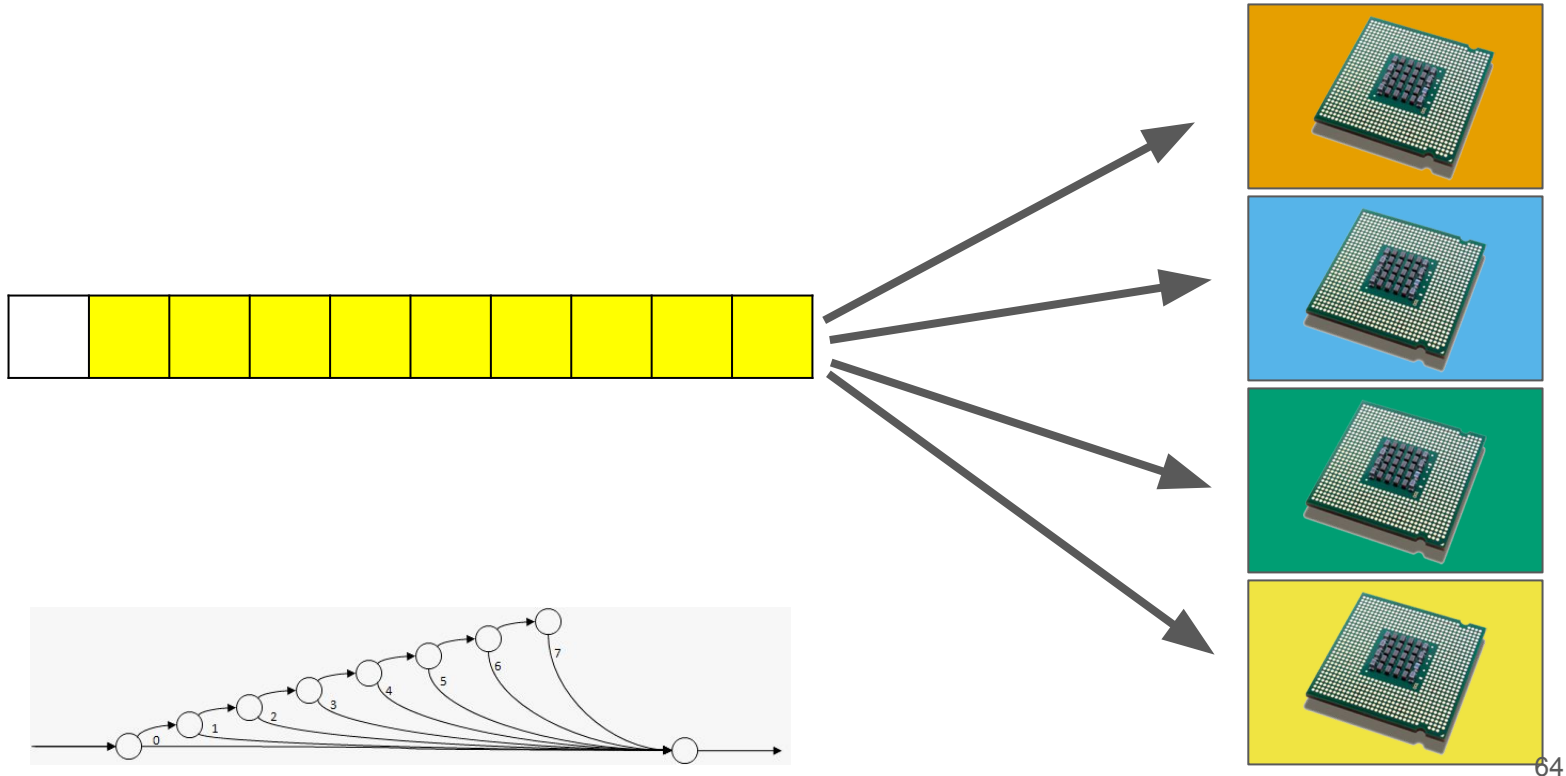


*** How is centralized queue performance affected?

```
futures = [N]
for int i=0; i<N; i++
    futures[i] = void_fork
        doSomething
for future in futures
    join future
```

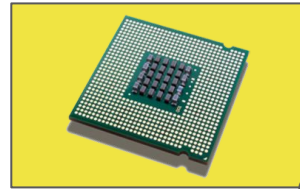
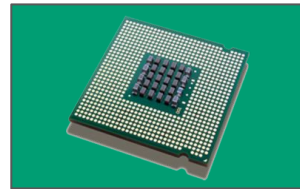
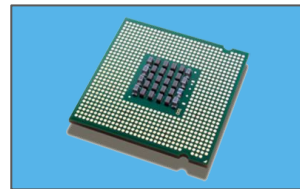
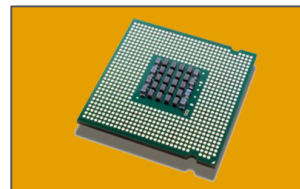
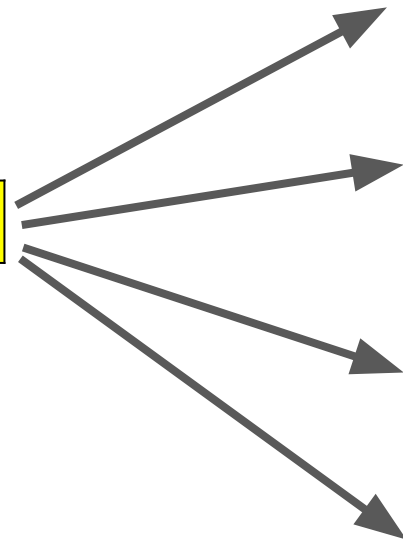
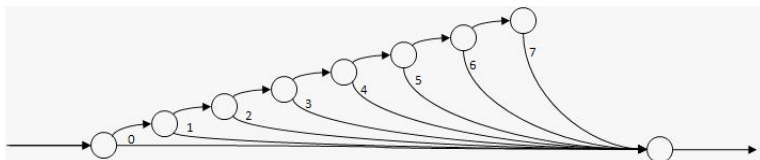


Centralized Work Queue



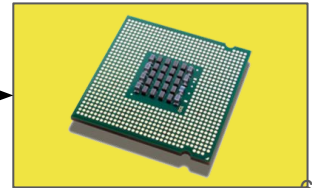
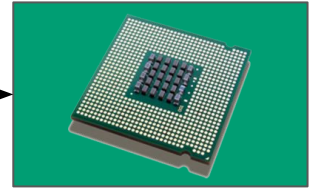
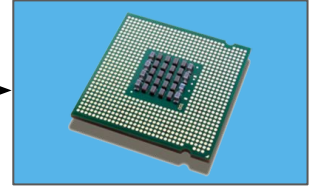
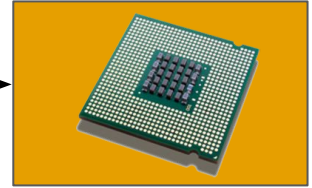
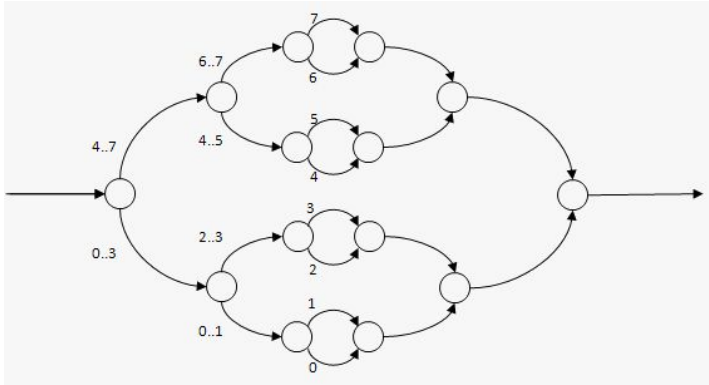
Either Way, Let `join_void_fork_loop` decide

```
join_void_fork_loop 0, N, (i) ->  
    doSomething
```



Either Way, Let `join_void_fork_loop` decide

```
join_void_fork_loop 0, N, (i) ->  
    doSomething
```



Remember Scheduler Can Do What It Thinks Is Best

fork says this **can** run in parallel, not that it will

S&Q: Is the underlying parallelism from cpp different from java?

How would this compare to parallel programming in other languages?

No.

Hopefully, the lessons you learn here will be applicable to almost any setting

S&Q: Does the programmer need to 'do' anything in work-stealing? Or is the idea just to program in order to take advantage of the scheduling strategy?

The idea behind fork/join, async/finish, spawn/sync is to allow you to express what can run in parallel and let the system worry about the details.

S&Q: I never thought much about considering cache utilization when dealing with parallelization (like with distributed work queues) -- is this something we should always consider when writing a parallel program?

Performance. Performance. Performance.

There are exceptions, but Performance.

Group WarmUp: Scheduler Client

```
public class SchedulerClient {  
    private static void output(Scheduler scheduler, int N) {  
  
        throw new NotImplementedException();  
  
    }  
  
    public static void main(String[] args) throws InterruptedException, ExecutionException {  
        ExecutorServiceScheduler scheduler = new ExecutorServiceScheduler();  
        output(scheduler, 10);  
        scheduler.joinAll();  
    }  
}
```

Minimalist: There Isn't Even A Join

```
public interface Scheduler {  
    void void_fork(Runnable runnable);  
}
```

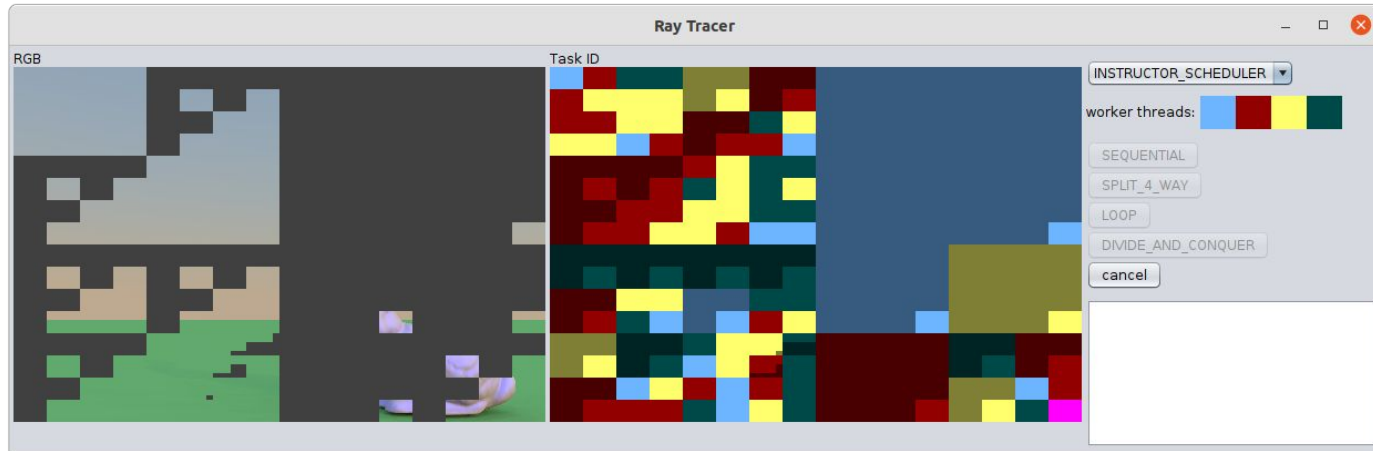

Group WarmUp: SequentialRayTracer

```
public class SequentialRayTracer implements RayTracer {  
    @Override  
    public void rayTrace(RayTraceContext context, Scheduler scheduler) {  
  
        throw new NotImplementedException();  
  
    }  
}
```

RayTraceContext

```
public interface RayTraceContext {  
    int width();  
    int height();  
    void mark(int xMin, int yMin, int xMaxExclusive, int yMaxExclusive);  
    void markAndRender(int xMin, int yMin, int xMaxExclusive, int  
yMaxExclusive);  
}
```

Parallel Ray Tracer Exercise

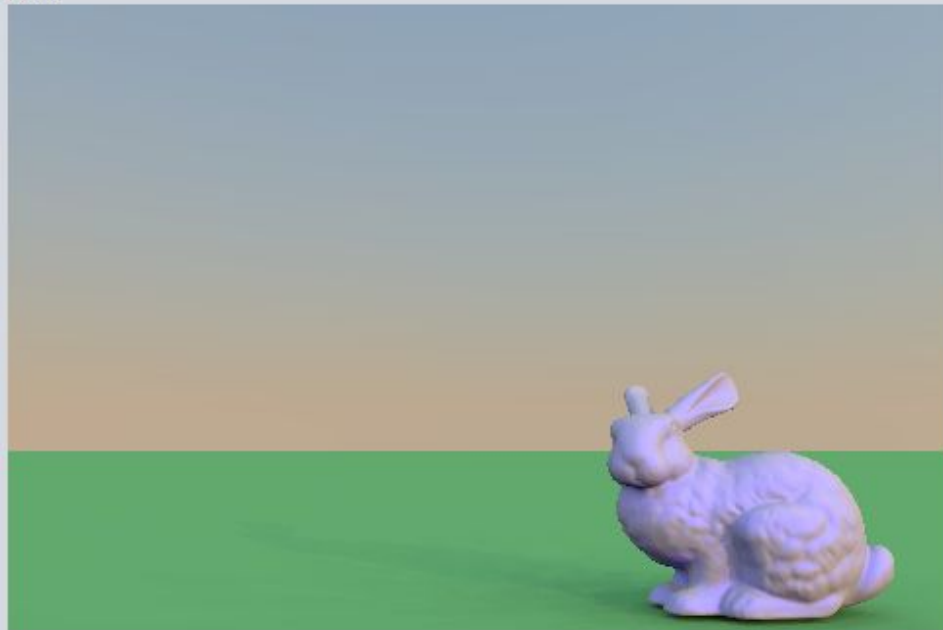


SplitFourWayRayTracer

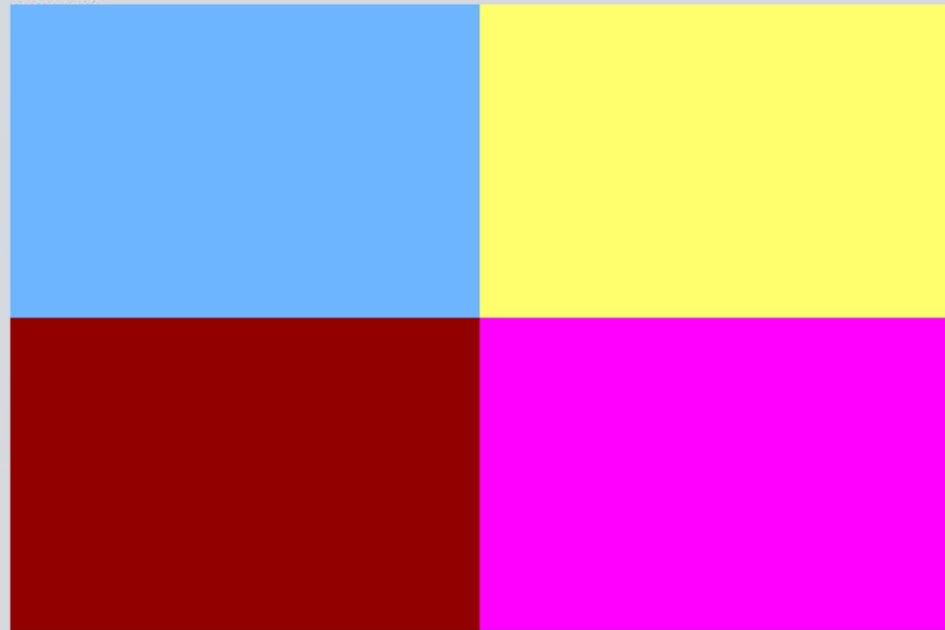
```
public class SplitFourWayRayTracer implements RayTracer {  
    @Override  
    public void rayTrace(RayTraceContext context, Scheduler scheduler) {  
        throw new NotImplementedException();  
    }  
}
```

SplitFourWayRayTracer

RGB



Task ID

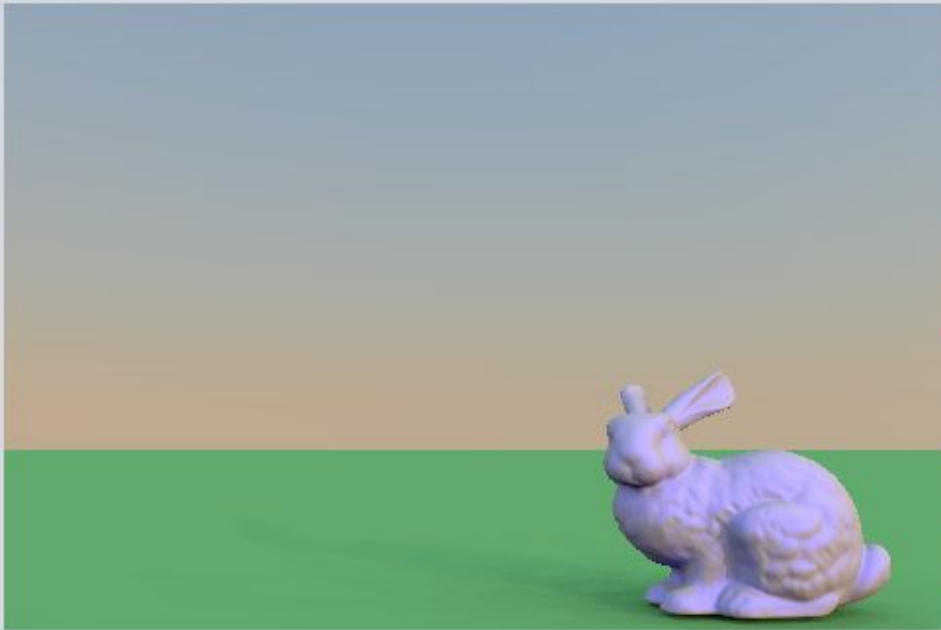


LoopRayTracer

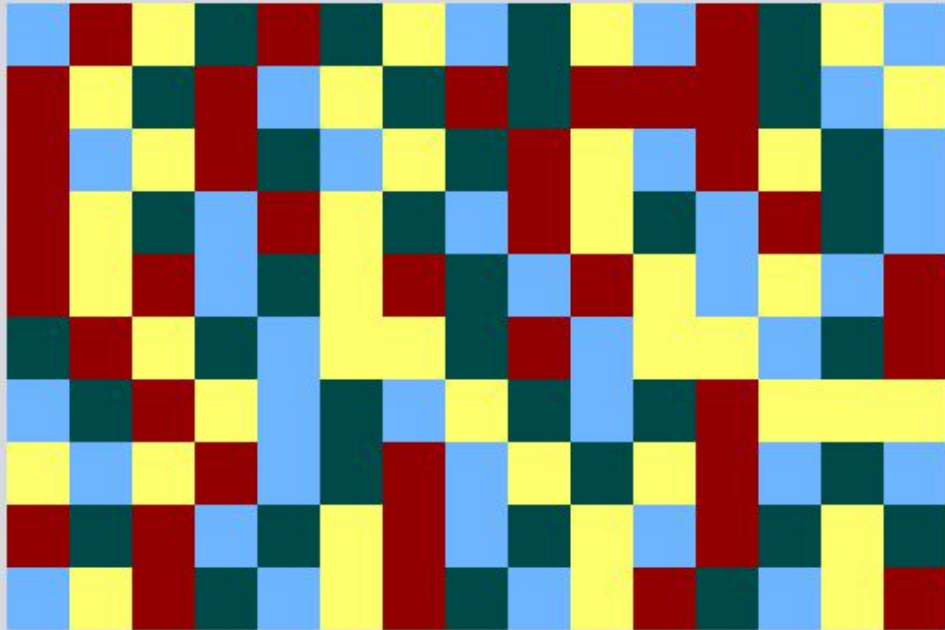
```
public class LoopRayTracer implements RayTracer {  
    public LoopRayTracer(Function<RayTraceContext, List<Section>> sectionsCreator) {  
        throw new NotImplementedException();  
    }  
    public Function<RayTraceContext, List<Section>> sectionsCreator() {  
        throw new NotImplementedException();  
    }  
    @Override  
    public void rayTrace(RayTraceContext context, Scheduler scheduler) {  
        throw new NotImplementedException();  
    }  
}
```

LoopRayTracer

RGB



Task ID

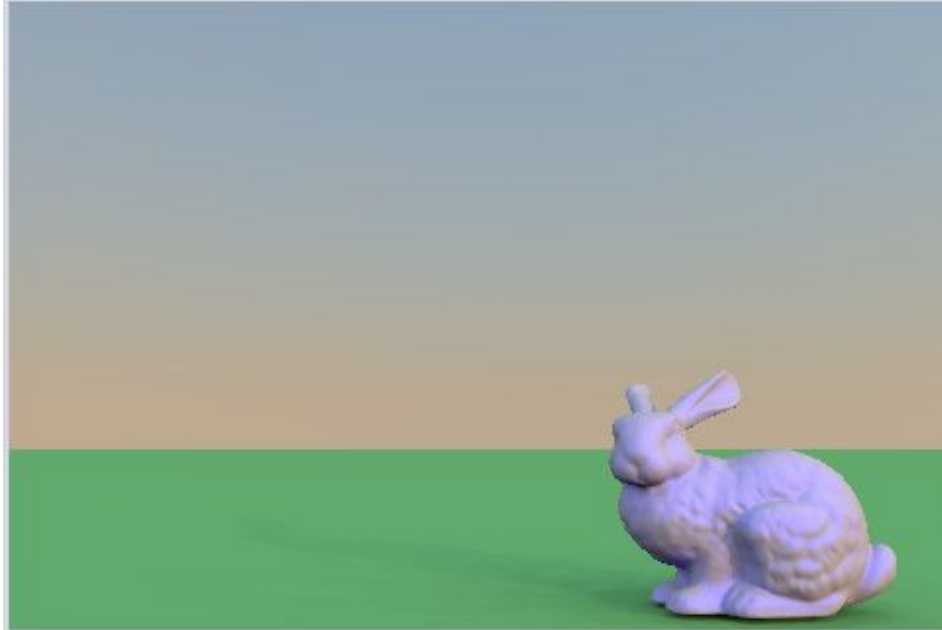


DivideAndConquerRayTracer

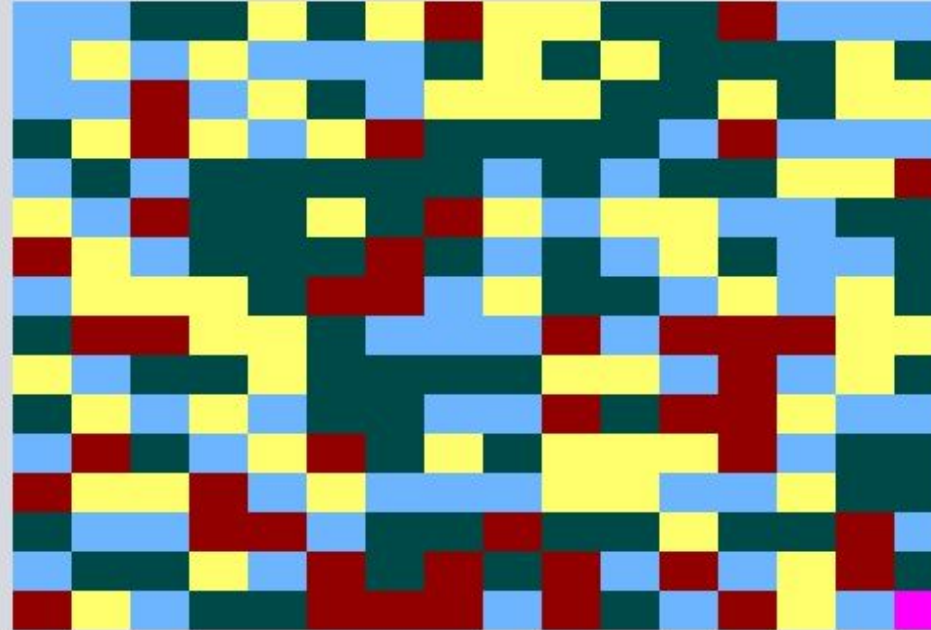
```
public class DivideAndConquerRayTracer implements RayTracer {
    public DivideAndConquerRayTracer(BiPredicate<Integer, Integer> thresholdPredicate) {
        throw new NotImplementedException();
    }
    public BiPredicate<Integer, Integer> thresholdPredicate() {
        throw new NotImplementedException();
    }
    private void rayTraceKernel(RayTraceContext context, Scheduler scheduler, int xMin, int yMin, int
xMax, int yMax) {
        throw new NotImplementedException();
    }
    @Override
    public void rayTrace(RayTraceContext context, Scheduler scheduler) {
        rayTraceKernel(context, scheduler, 0, 0, context.width(), context.height());
    }
}
```


DivideAndConquerRayTracer

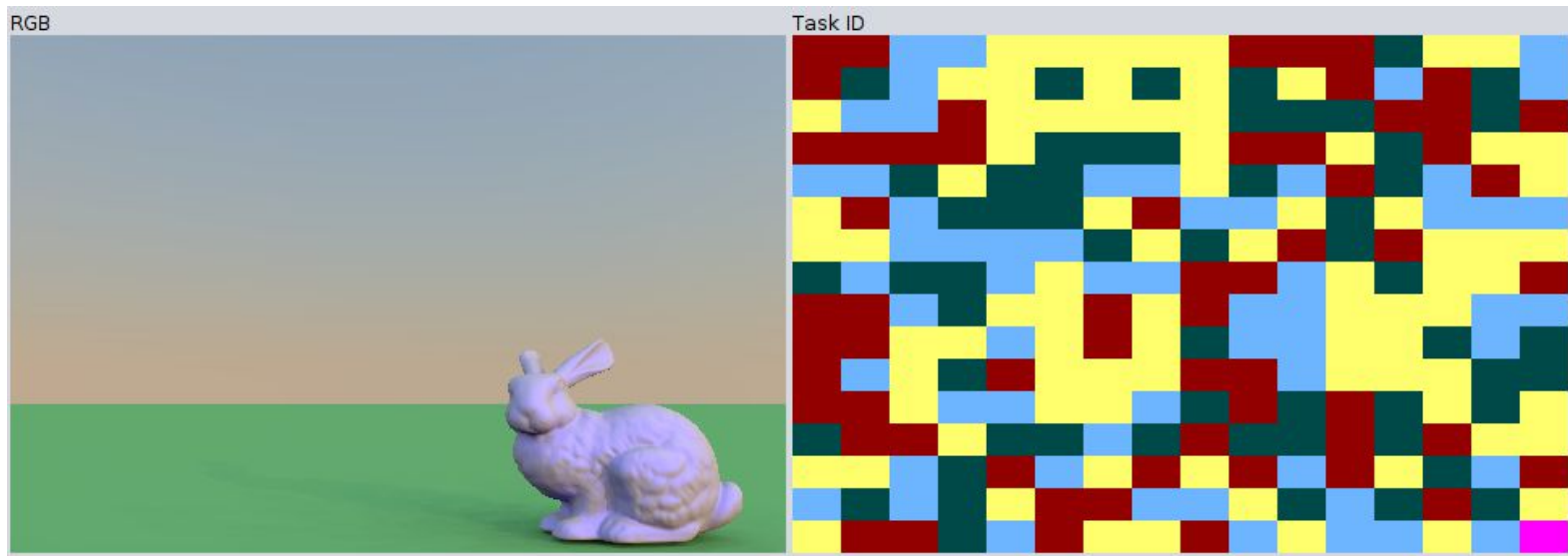
RGB

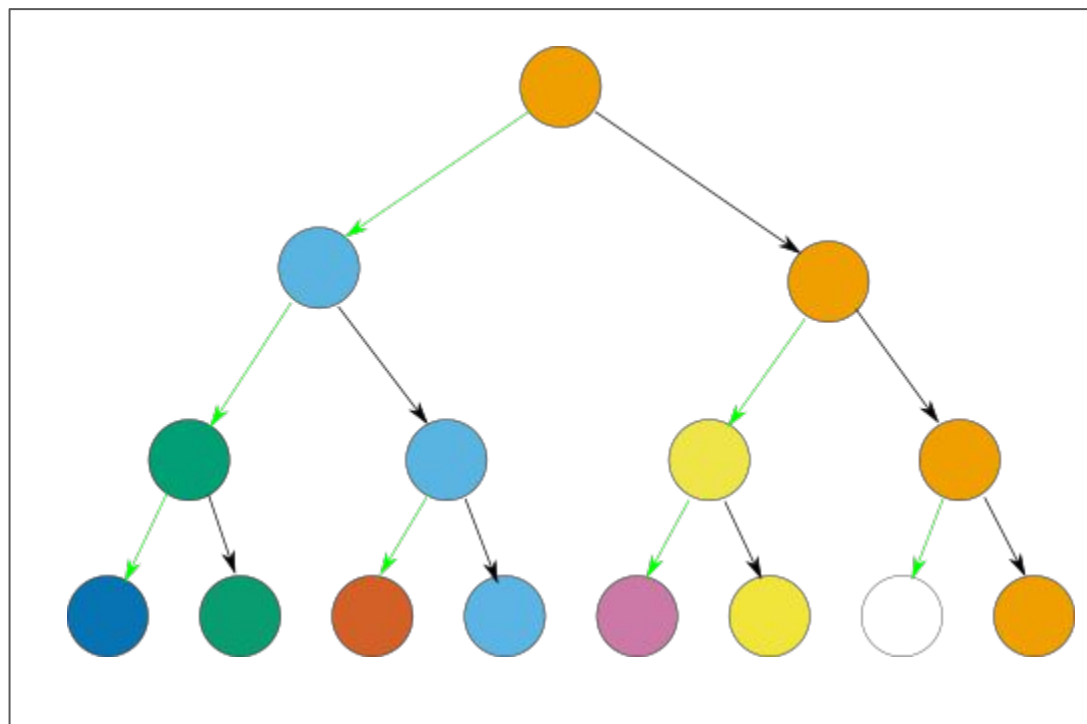


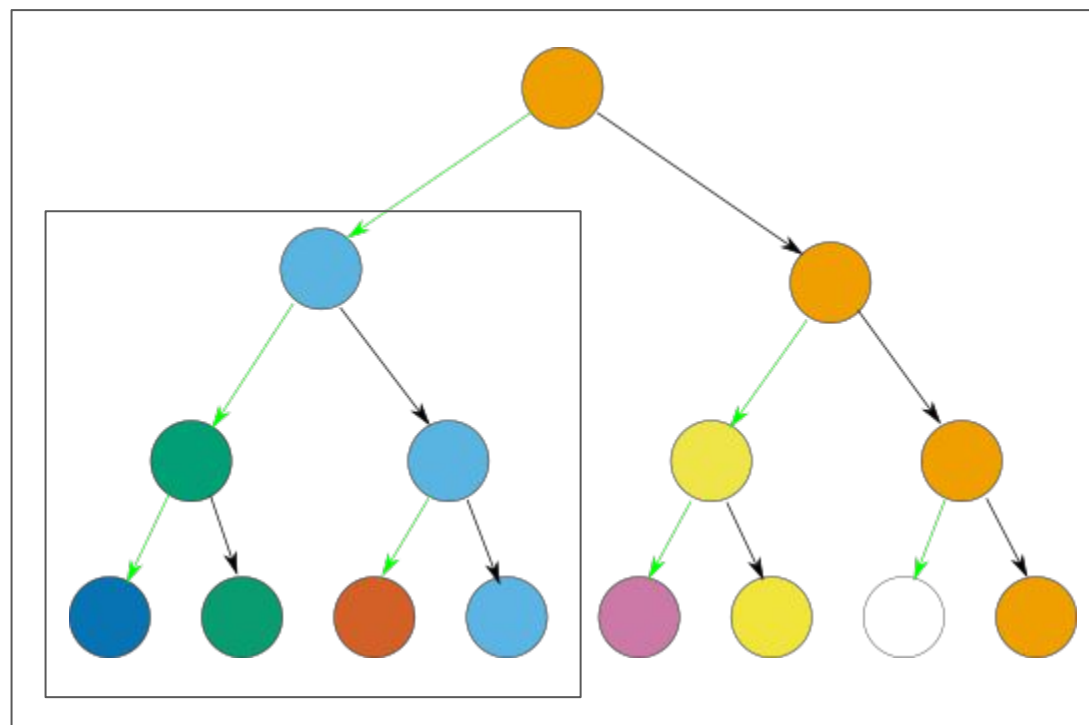
Task ID

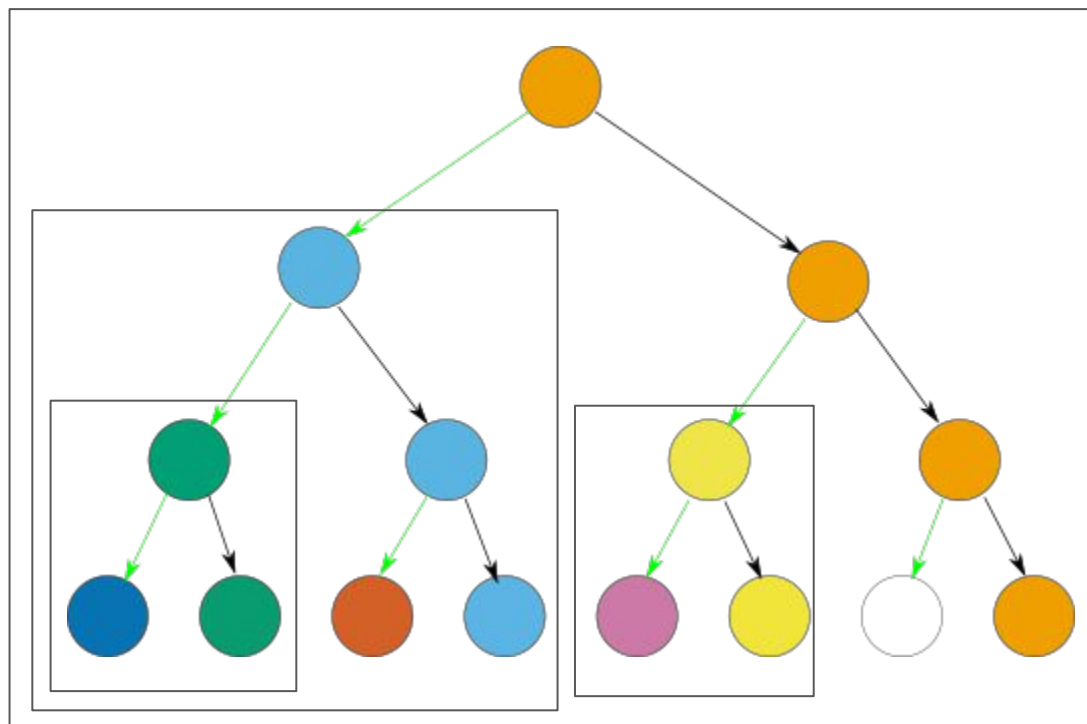


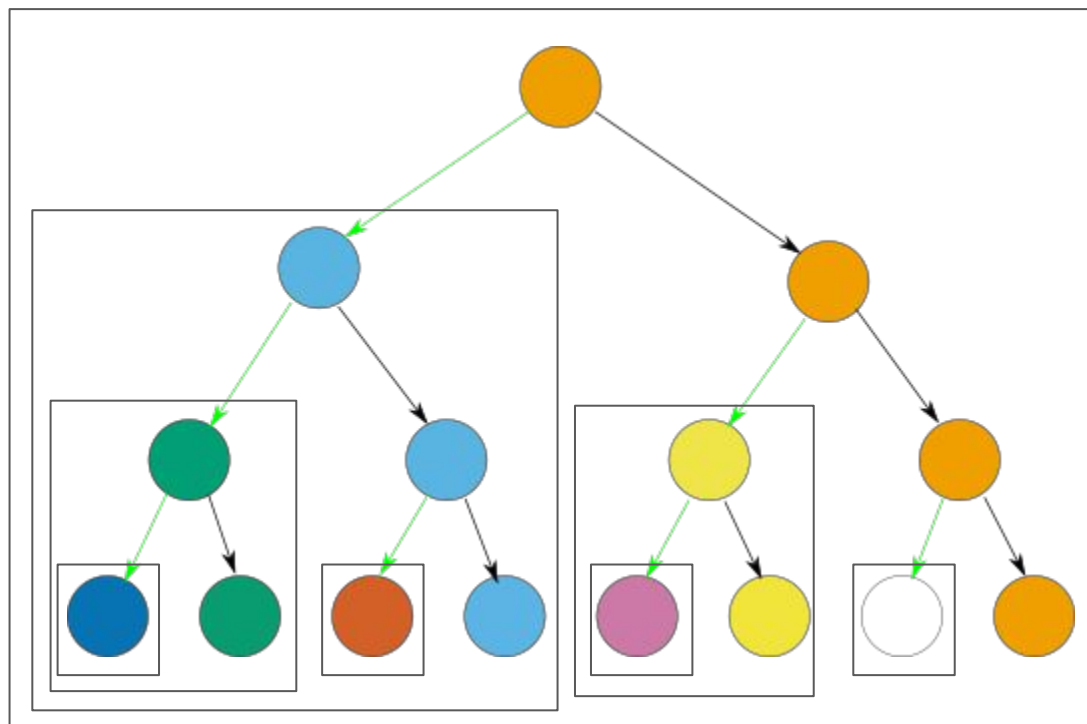
Why Magenta?











Centralized Work Queue Challenge

not required

CentralizedWorkQueueScheduler

```
public class CentralizedWorkQueueScheduler implements Scheduler {  
    public CentralizedWorkQueueScheduler(CentralizedWorkQueue centralizedWorkQueue) {  
        throw new NotImplementedException();  
    }  
    public CentralizedWorkQueue centralizedWorkQueue() {  
        throw new NotImplementedException();  
    }  
    @Override  
    public void void_fork(Runnable runnable) {  
        throw new NotImplementedException();  
    }  
}
```

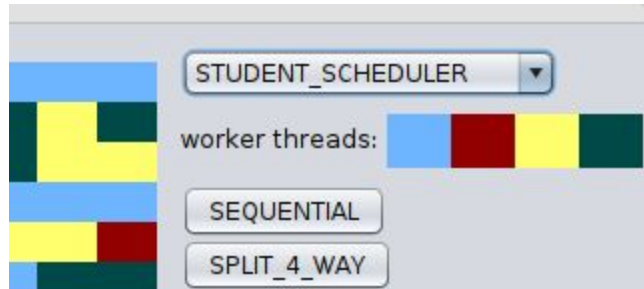

WorkerThread

```
public class WorkerThread extends ThreadWithId {  
    public WorkerThread(CentralizedWorkQueue queue, int id) {  
        throw new NotImplementedException();  
    }  
    @Override  
    public int id() {  
        throw new NotImplementedException();  
    }  
    @Override  
    public void run() {  
        throw new NotImplementedException();  
    }  
}
```

WorkerThreads

```
public class WorkerThreads {  
    public static WorkerThread[] createAll(CentralizedWorkQueue queue, int n) {  
        throw new NotImplementedException();  
    }  
    public static void startAll(WorkerThread[] workerThreads) {  
        throw new NotImplementedException();  
    }  
    public static void joinAll(WorkerThread[] workerThreads) throws InterruptedException {  
        throw new NotImplementedException();  
    }  
}
```

Test With RayTraceViz



S&Q: Instead of randomly stealing from another worker, would it make sense to steal from the worker that has the most tasks left?

S&Q: What happens during non-stalling join vs stalling join?

S&Q: What is context switching? The term seems to have come up many times during the lecture.

S&Q: What exactly is a cactus stack? How does it differ from other types of stacks? When are cactus stacks practically used?

S&Q: If we create a data structure like a queue for parallel programming, won't that effectively turn it into a sequential program

effective: I liked doing the sequential solutions to problems and then following up with the parallel implementation. This allowed me to see how the code is similar and different and see which parts of the code can be changed to parallel and which can't.

can improve: It would be nice if the worksheet/missing attendances were updated so we could know how many we have done and what our grade looks like during the semester.

I agree

S&Q: Is there a reason we use Fork Join, as opposed to CILK? Is it because the class is 200 level, and Java is easier compared to lower level C/C++?

If 131 was in C/C++ we would undoubtedly be using cilk

can improve: Probably not feasible this is in terms of prereqs, but I would also like to see how these concepts are applied in languages other than Java with maybe some optional assignments exploring C++, etc.

can improve: Spending more time on parallel processing in **other languages** would be useful. For example, I'd love to apply concepts we've learned to other languages I'll be programming with in the future.

can improve: I'm biased because I code in it, but I do think it would be fun to have a C++ day like near the end or something, but in terms of like any like important changes, I can't think of any. This class overall was really good and well taught!

I think this is a legit good idea. Deal with the installation though???

can improve: More specific write ups and lecture slides, to help with completing assignments. Though going to lecture gives you a great start to completing assignments, sometimes key information can be forgotten, and the slides don't really explain the diagrams or the images.

can improve: Many of the assignments are very object-oriented, which isn't a bad thing. However, there are some where the OOP-centric nature makes the general parallel programming concepts less understandable (at least, in my experience).

I had not thought of this. Certainly all of the specific Java syntax has always been on my radar. But OOP is new.

can improve: This course can improve by providing more fun examples and group interactivity for specific difficult concepts!

can improve: Can we do C++ parallel version ?

can improve: I think learning parallel **in other languages would be cool** as well (like maybe 1 assignment in python or something similar). I would also have liked it if we talked a bit more about how you know how many resources you have (for example how many processors is expected to have on a computer, a server, etc..)

Memory Visibility in Java

[synchronized](#) and [explicit Locks](#) guarantee “[happens before](#)” (which includes memory visibility).



[volatile](#) keyword is also useful

fork, join



```
double hypotenuse_via_sync(double a, double b) {  
    double a2 = cilk_spawn square(a);  
    double b2 = square(b);  
    cilk_sync;  
    return sqrt(a2 + b2);  
}
```


fork loop



```
void square_array(int* dst, int* src, int length) {  
    #pragma cilk grainsize 2  
    cilk_for(int i=0; i<length; ++i) {  
        dst[i] = square(src[i]);  
    }  
}
```

data race



```
int shared_mutable_value[1];

void repeatedly_increment(int v) {
    for(int i=0; i<v; ++i) {
        ++shared_mutable_value[0];
    }
}

int complete_increment(int left_amount, int right_amount) {
    shared_mutable_value[0] = 0;
    cilk_spawn repeatedly_increment(left_amount);
    repeatedly_increment(right_amount);
    cilk_sync;
    return shared_mutable_value[0];
}
```

mutual exclusion



```
int shared_mutable_value[1];
pthread_mutex_t lock;

void repeatedly_increment(int v) {
    pthread_mutex_lock(&lock);
    for(int i=0; i<v; ++i) {
        ++shared_mutable_value[0];
    }
    pthread_mutex_unlock(&lock);
}

int complete_increment(int left_amount, int right_amount) {
    shared_mutable_value[0] = 0;
    cilk_spawn repeatedly_increment(left_amount);
    repeatedly_increment(right_amount);
    cilk_sync;
    return shared_mutable_value[0];
}
```

Intel Thread Building Blocks (TBB)



OpenMP

OpenMP[®]

start and join



```
def hypotenuse(a, b):  
    a_value = multiprocessing.Value('d', a)  
    def task():  
        a_value.value = square(a_value.value)  
  
    process = multiprocessing.Process(target=task)  
    process.start()  
    b2 = square(b)  
    process.join()  
    a2 = a_value.value  
    return math.sqrt(a2 + b2)  
  
print(hypotenuse(3.0, 4.0))
```

parallel map



```
def square_all(xs):  
    # with statement automatically cleans up like try with resources in Java  
    with multiprocessing.Pool() as pool:  
        squares = pool.map(square, xs)  
        return squares  
  
print(square_all(range(1, 17)))
```

data race



```
def repeatedly_increment(amount, x):
    for i in range(amount):
        x.value += 1

def complete_increment(left_amount, right_amount):
    shared_mutable_value = multiprocessing.Value('i', 0)
    def task():
        repeatedly_increment(left_amount, shared_mutable_value)

    process = multiprocessing.Process(target=task)
    process.start()
    repeatedly_increment(right_amount, shared_mutable_value)
    process.join()
    return shared_mutable_value.value
```

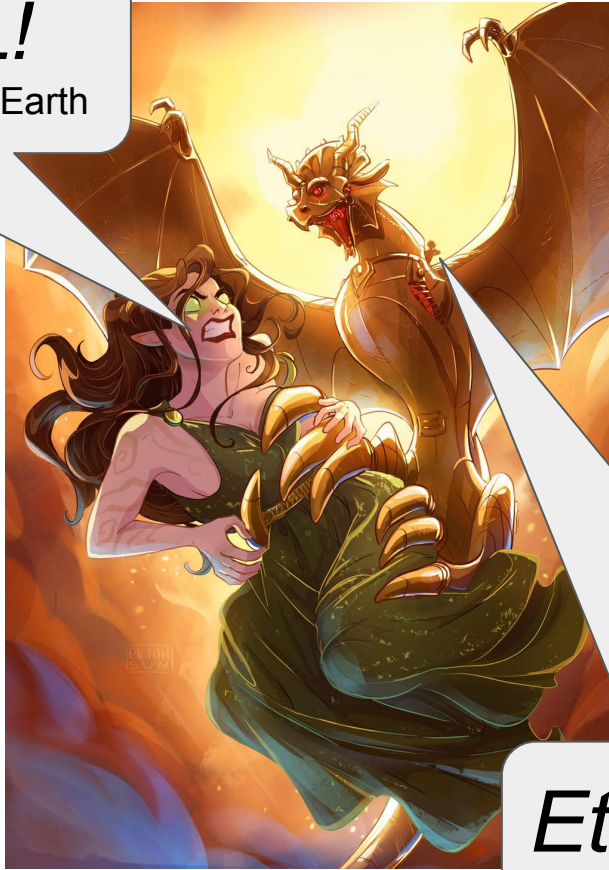

mutual exclusion



```
def repeatedly_increment (amount, x):
    for i in range (amount):
        x.value += 1
def complete_increment (left_amount, right_amount):
    lock = multiprocessing.Lock()
    shared_mutable_value = multiprocessing.Value( 'i', 0)
    def task():
        lock.acquire()
        try:
            repeatedly_increment(left_amount, shared_mutable_value)
        finally:
            lock.release()
    process = multiprocessing.Process( target=task)
    process.start()
    lock.acquire()
    try:
        repeatedly_increment(right_amount, shared_mutable_value)
    finally:
        lock.release()
    process.join()
    return shared_mutable_value.value
```

I AM ETERNAL!

-- Gaea "Race Conditions" Earth



Eternally annoying!

-- Leo "Bad Boy Supreme" Valdez

Go: Communicate to Share Data (not the other way around)



Tony Hoare Paper (1978): [Communicating Sequential Processes](#)

[goroutines and channels](#)

Haskell: Parallel and Concurrency



since all functions are pure, you can simply compile your program and tell it to run in parallel at the command line!

Monads like IO (for example, print to the console) are special and have concurrency support.

```
hypotenuse a b = sqrt (a*a + b*b)
main = do
    print(hypotenuse 3.0 4.0)
```

Rust: Fearless Concurrency



Ownership and Borrowing (of memory) is fundamental to the language

Same compile time memory checks for security serve to prevent sharing and mutation (data race free for free!)

Ruby 3 Ractors

<https://ruby-doc.org/core-3.0.0/Ractor.html>

<https://blog.appsignal.com/2022/08/24/an-introduction-to-ractors-in-ruby.html>

Limited object sharing (deeply frozen (immutable) Objects allowed)

- push: send and receive
- pull: yield and take

It is hard to describe just how lenient Ruby is, yet to support parallel the designers have reasonably constrained Ractors



You on Ruby

effective: I liked that there were no big projects in the class, and that instead we got to dip our toes in a variety of problem types and techniques. I felt I was able to learn more this way.



effective: I liked the exercising but maybe instead of so many maybe bigger ones.

can improve: A bit more support for those who are fresh out of 131 maybe data structures took a bit to grasp.

hopefully, more students will take 2301 earlier from here. that will understandably require more support.

can improve: I think that encouraging people to take this class **earlier in college** would help. Taking it as a senior, there were many times I wished it was harder/more complex. I don't know what extra things to add to the course to make it harder, but I'm sure they exist and I felt like it wasn't challenging enough to be super interesting.

can improve: I think it could improve by having less exercises so that students are not easily behind in the course. It's very easy to get stuck when like only a few tests do not pass.

can improve: I found it hard to connect my understanding of the material to the exercises sometimes. Syntax for fork-join and Optionals were usually pretty confusing to me.

- lambdas are unavoidable (could explain them better, though)
- Optionals are dying

can improve: Sometimes I found the lecture **hard to follow** because it went by **so fast**. Some real-life applications (e.g. chess and poker) were also hard to follow just because I wasn't familiar with them. There isn't much I'd change honestly.

can improve: I think I would've benefited from the lectures more if they were a bit more structured. Sometimes it felt like we were jumping between topics and answering student prep questions, and I felt like I didn't always know how to apply what we were talking about in lecture to what we were learning.

can improve: Feel like the lectures can be a bit distracting sometimes. Maybe have a recap at the end or at the beginning of each

can improve: Sometimes I found the lecture hard to follow because it went by so fast. Some real-life applications (e.g. chess and poker) were also hard to follow just because I wasn't familiar with them.

can improve: more real world examples maybe?

can improve: I personally think there were a lot of assignments due within a hard timeframe. I love this course. The only improvement I have is the deadlines. I really like the idea of having all the assignments in by the end of the semester with a "guideline" and recommended workflow.

can improve: I think this course could maybe improve by having less assignments, but maybe longer and more intricate ones. Or maybe let you miss one or two without penalty.

can improve: By not adding back exams-- not having exams for this class was amazing, and I don't feel like they would add much to be completely honest. The concepts we learn early on naturally come back up while working on more complex problems, so testing whether we know them almost seems unnecessary since we don't really have a choice-- we have to understand them or else we can't do the work.

S&Q: Although I do not enjoy exams they do help me maintain information long term so it would be nice to have an exam during this class.

*can improve: more professor made lectures online
would be nice*

can improve: Maybe make the homework instructions more detailed and clear on the wiki?

can improve: Perhaps lectures could be longer rather than having work periods at the end.

can improve: I honestly don't think the synthesis/question for the prep was super necessary, but that might have just been because I was not utilizing it correctly. I think my questions got cleared up in class 99% of the time anyways, and it kind of just felt like pointless work to fill out those fields on canvas. I think the synthesis/question made more sense for in class, but it felt weird to have to do it twice, once at home and once in class.

can improve: I think some of the custom data types used in exercises were slightly difficult to understand and did not always feel like they were critical parts of the ideas underlying the exercise. However, this was good practice for understanding an existing code base (very industry-relevant).

can improve: If there are videos that go over classes and methods that are already provided for us and what and how they work it would be helpful, because I don't understand the Java Doc all the time and even if I do there are times I don't understand how the methods need to be invoked.

can improve: Include small crossword puzzles every month

can improve: The only criticism I have is sometimes it's confusing to navigate all the different custom classes before you do an assignment to get the lay of the land, but it's kind of unavoidable if you want to have engaging assignments.

can improve: This course could improve by offering more guided practice on debugging parallel code, especially around race conditions, deadlocks, and subtle synchronization bugs. Sometimes you fail the test cases, you cannot tell which part of the code does not meet the course's expectations just based on the error description, and may result in modifying some other non-essential parts.

can improve: I mentioned this to professor cosgrove, but I think there needs to be more incentive associated with warmups. I found them to be really helpful once I started I doing them, and I think that they would really enhance our understanding of the material if we were pushed to actually do them.

can improve: I think there should be coding time in class. There were a couple classes where we would do a small coding assignment and then regroup to see the solution, and I think this helped my learning a lot because it is easy to just listen to the slides and get a good sense of what is going on, but I think it misses the implementation aspect sometimes, though I don't know if this is a limiting factor here. Also remove extensions.

can improve: This is something minor (and perhaps a personal preference), but all the assignments being stored together in the same place in IntelliJ made finding necessary files pretty tedious. I'm unsure what a solution to this would look like (that doesn't require a lot of work), but I figured I'd mention it here.

can improve: Walking through more examples of new concepts in class.

can improve: A more updated wiki.

- *can improve: The wiki page.*
- *can improve: I feel like sometimes the program descriptions are not clear enough that I do not really understand what the homework wants me or my program to do.*
- *can improve: I think there were a few assignments where the directions could sometimes be a bit vague. I think making sure that all of the exercise directions are clear could improve the course.*
- *can improve: Some of the assignment pages didn't have a ton of information on them and that made it tough to figure out exactly what I needed to do the assignments and what it was looking for. I think a little more information would help us figure it out better on our own.*

can improve: I think some times the questions included in the slides got me more confused then before.

can improve: The lecturing in class could be a little more direct and clear.

can improve: Lectures were a little all over the place.

- I agree.

can improve: I don't think the worksheets were that helpful and just made information more confusing.

effective:

Great TAs

Very interesting topics

Cool S&Q videos

Learned way more Java than I thought I would.

The Other Side



can improve: removing some basic Java programming assignments and adding more assignments about concurrent programming.

trying to make it pay off:

- powers of 2 iterable -> scan
- non-thread safe hashmap -> concurrent hash map
- non-thread-safe stack => atomic stack.

still it is a lot, and understandably frustrating

28 Days Later



- can improve: The 28 day deadline allowed me to get really behind (which was my own fault but still stressful).
- can improve: I think there should be no late days grace periods for exercises and maybe late coupons can be offered for students to use if necessary. It's so easy to fall behind now, and the TAs and instructors end up with a lot of grading all at once.



- effective: Letting students do work mostly at their own pace.

Wiki

- can improve: This course can improve if there was more context on some of the Wiki pages. Sometimes we would be confused on the directions of an exercise because the wiki was a little vague.
- can improve: There is potential improvement to be made in select spots of the Wiki that are leftover from previous semesters or filenames.



- instructions on the assignments are at times vague, but as is the challenges we should conquer as CS students.

note: I think we are in the non-productive zone on the lack of clarity front which should be improved

I Failed To Ease Up

- can improve: *I feel like towards the end of the semester I felt more stressed out with the quantity of work. Maybe allotting more time for some of the bigger assignments might reduce that stress*



- effective: *I think that this course was effective in gradually ramping up the difficulty in assignments. This allowed me to gain the skills and confidence to do the next assignment as the semester went on.*

More Exam Like Assessment

can improve: I think there should be some sort of assessment because otherwise procrastinators like me don't study. For example, making in-class quiz grades.

can improve: Maybe put the exam back? but a coding exam

~~Final Exam Prep~~

Too much abstraction

can improve: Too much abstraction, you have mentioned that this is due to testing, and to prevent students from approaching problems in unnecessary ways to create more problems. I understand that it makes it easier to control, but it is also a hassle to have to familiarize ourselves with the way assignments for the class are laid out especially in the first few weeks. I think a better approach to this would be to have an FAQ or "things to look out for" page for the assignments rather than abstracting too much. I also think the assignment wiki pages can use a lot more explaining as I have found myself in TA hours a lot of times not for not knowing how to do things, but because it was not clear what I was expected to do. This also has a lot to do with the abstraction. Toward the end of the semester, there were more videos explaining what to do and how to approach the assignments in the assignment wiki pages which were super helpful.

*can improve: Not much to say -- possibly make
S&Q's weekly?*

Course Evals

<https://registrar.wustl.edu/washu-course-evaluations/>

shout out: Finn Voichick - I went to middle and high school with his younger brother. It was cool and kind of surprising to see his name on a lot of these assignments.

so much of what is good in this course comes from student suggestions

Anonymous Feedback

CSE 231s Anonymous General Feedback

[Sign in to Google](#) to save your progress. [Learn more](#)

* Required

What would you like to say? *

Your answer

Submit

[Clear form](#)

Motivation For This Course

Moore's Law

Gordon Moore, co-founder of Intel, 1965

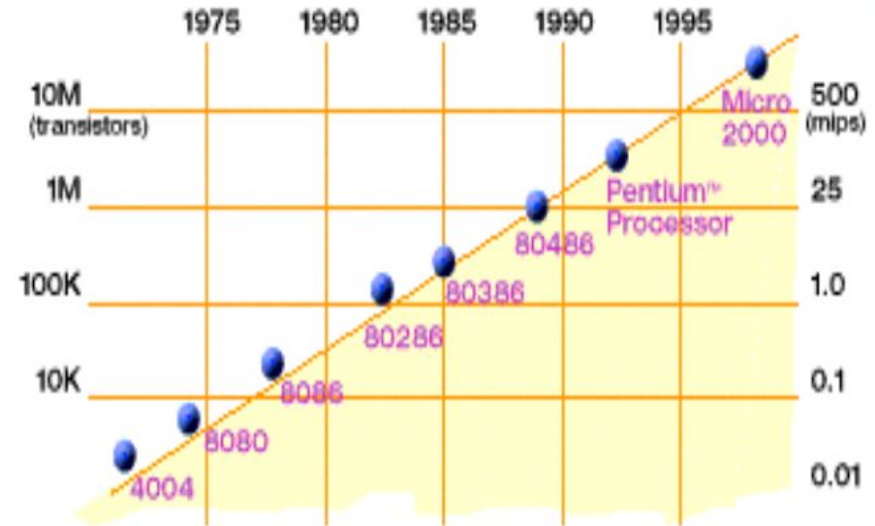
"# of transistors double every 2 years"

50 years of awesomeness

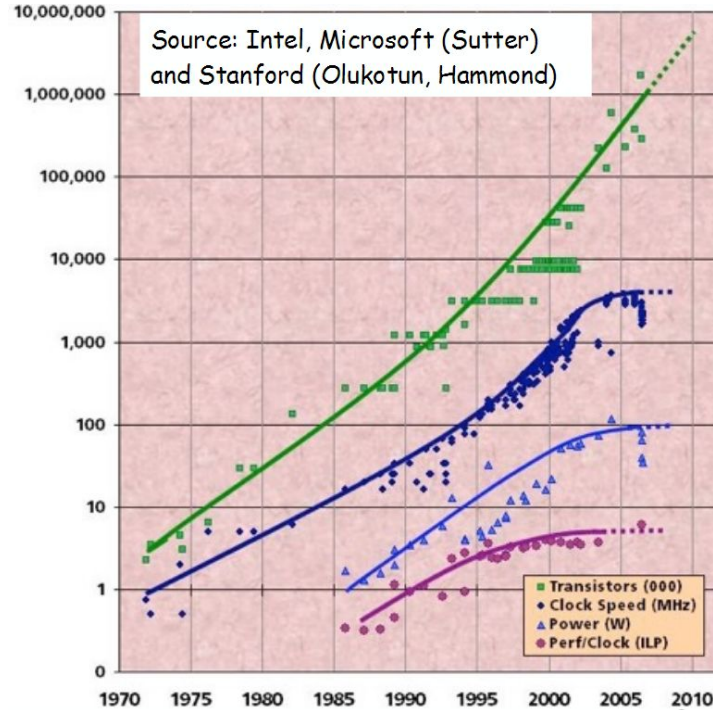
1958

Note the Log 10 Scale

Imagine if this happened to other industries: transportation



It was nice while it lasted



What is Parallel Programming?

“using multiple processors in parallel to solve problems ***more quickly*** than with a single processor and/or ***with less energy***” - *Sarkar, Imam*

Power

Power \sim Capacitance * Voltage² * Frequency

Maximum Frequency is capped by Voltage

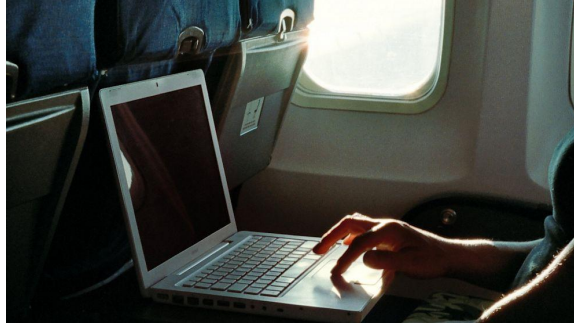
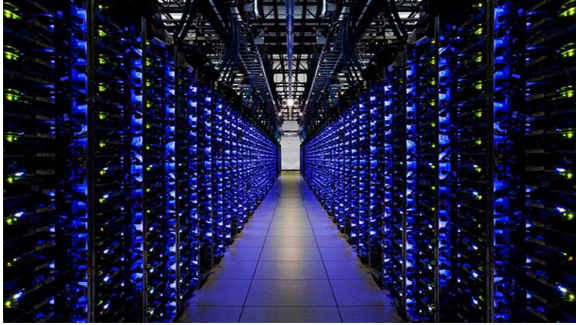
Power is proportional to Frequency³

8 Power == 8 Power

2 GHz	1 GHz
	1 GHz
	1 GHz
	1 GHz
	1 GHz
	1 GHz
	1 GHz
	1 GHz

- $2^3 = (1^3 + 1^3 + 1^3 + 1^3 + 1^3 + 1^3 + 1^3 + 1^3) = 8$

Power is a big deal



“But I heard parallel programming is hard...”

It is unavoidable.

Parallelism and Concurrency

Parallel programming is about using additional computational resources to produce an answer faster.

Concurrent programming is about correctly and efficiently controlling access by multiple threads to shared resources.

(source: <https://homes.cs.washington.edu/~dijg/teachingMaterials/spac/sophomoricParallelismAndConcurrency.pdf>)

*** What Is A Race Condition?

*** How Can You Avoid Race Conditions?

Doctor, It Hurts When I Do This

Don't do that.

Shared Mutable Data Is The Problem

Going To Agrawalandia Is A Solution



What If You Can't Find Your Way To Agrawalandia?

North–South railway aka Reunification Express



Locks Are Not Magic



Q&S: Where To Go Next

What would be the next class to take after this course?

Are there some other courses offered at WashU that sort of continues from what we have learned in 231 so far?

can improve: Personally, I would love more 347 type theory. I know 549 does this but I think this course could have a bit more of that. Also, it could be a good idea to have 231 and 549 merge into "2-part courses"? Like this course is "parallel programming 1" and 549 is "parallel programming 2".

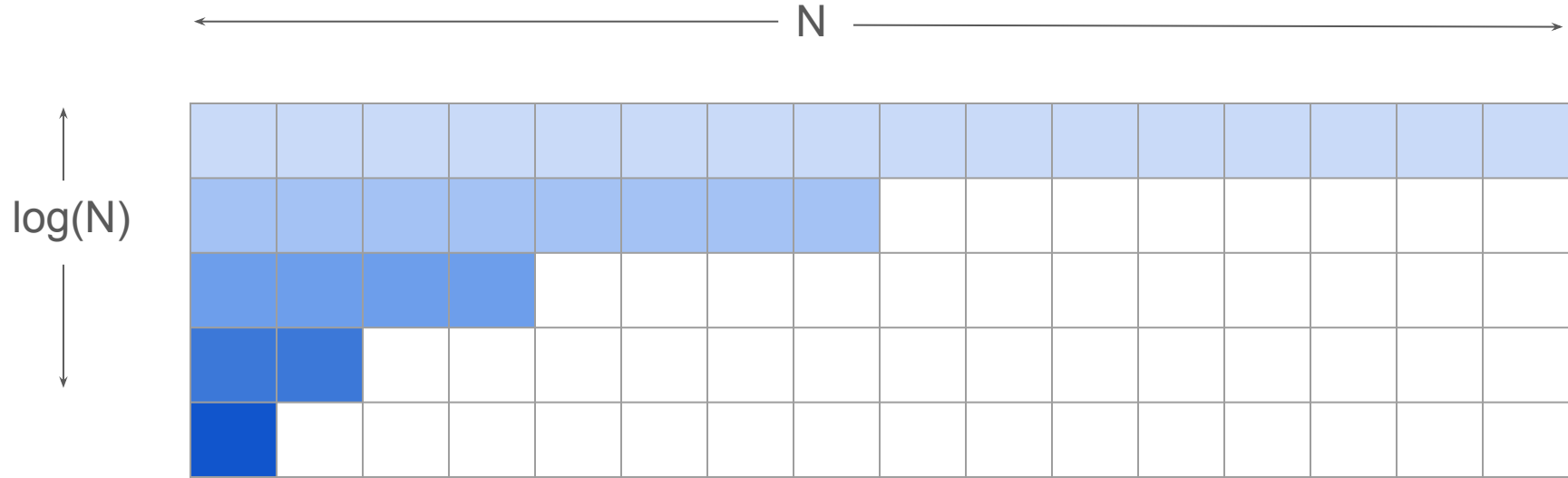
CSE 549T Theory of Parallel Systems

“Parallel merge combine’s tree isn’t $2 \log(n)$ high. It is $9/16$ at each level. What is wrong with you?!?”

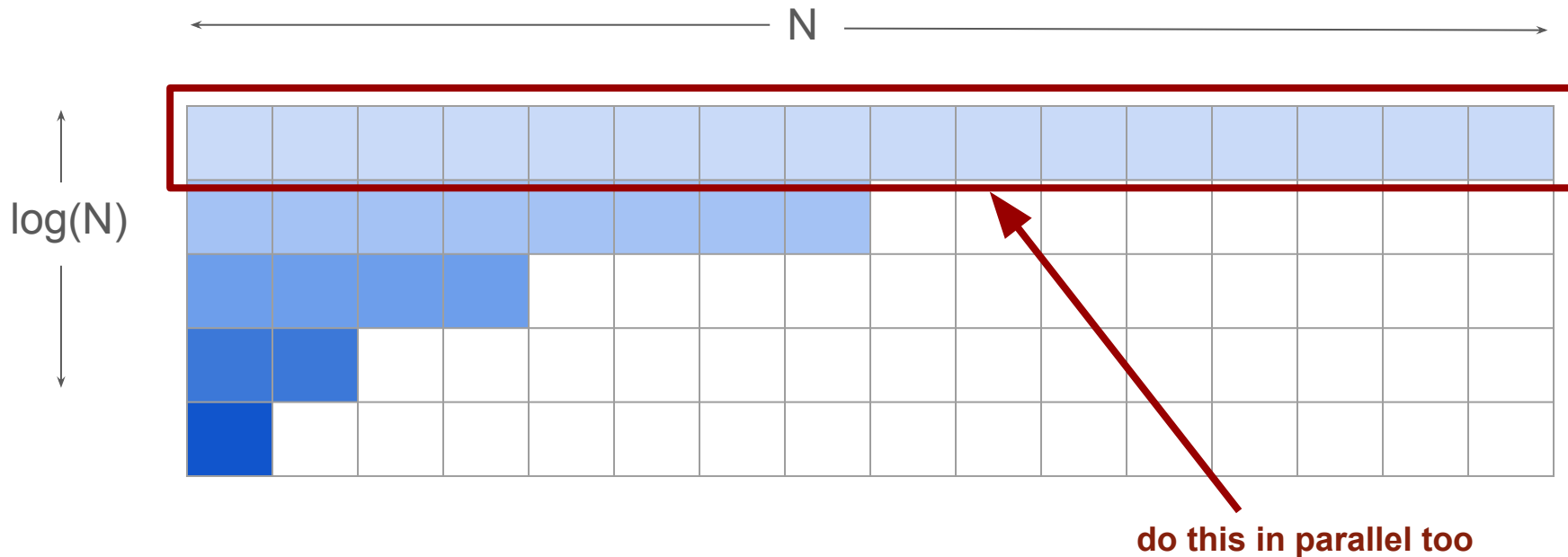
- Heavily Paraphrased Prof. Agrawal
at my hand waving $\log^k(n)$ merge sort “proof”



Merge Sort CPL $2N$ (for sequential combine)



Sort CPL $\log^k(n)$ (for parallel combine)



<http://www.classes.cec.wustl.edu/~cse341/web/handouts/lecture07.pdf>

https://classes.engineering.wustl.edu/cse231/core/index.php?title=MergeSort_Parallel_Combiner

CSE 539S: Concepts in Multicore Computing

“The only way to develop skill is to struggle.”

- Heavily Paraphrased Prof. Lee

On the way:

CSE 361: Introduction to Systems Software



CSE 532S: Advanced Multiparadigm Software Development

Intensive focus on advanced design and implementation of concurrent and distributed system software in C++.

Topics covered include concurrency and synchronization features and software architecture patterns.

Prerequisites: CSE 332S or graduate standing and strong familiarity with C++; and CSE 422S

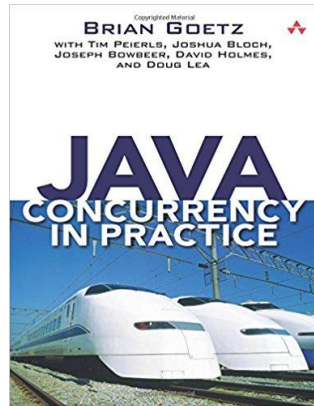
Prof. Gill



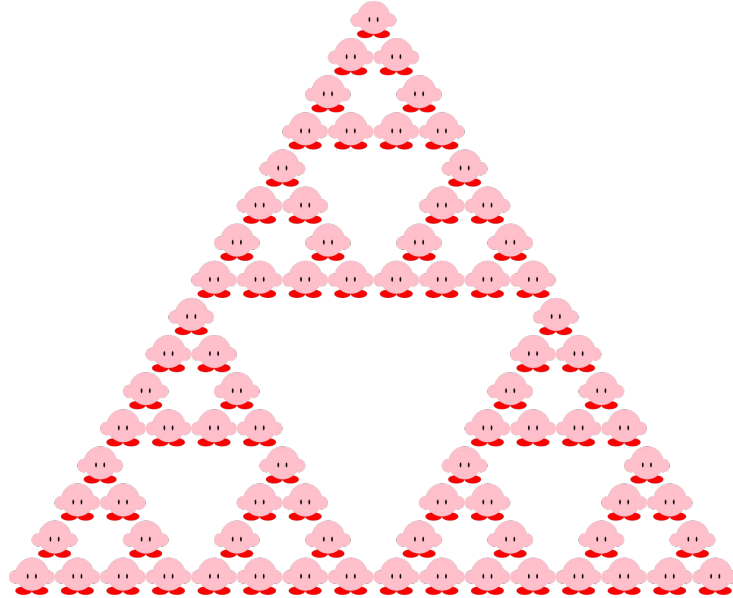
Java Concurrency In Practice

<https://smile.amazon.com/Java-Concurrency-Practice-Brian-Goetz/dp/0321349601>

- Brian Goetz, Java Language Architect, Sun now Oracle



shout out: KIRBINSKI FOREVER!!!!!!!!!!!!



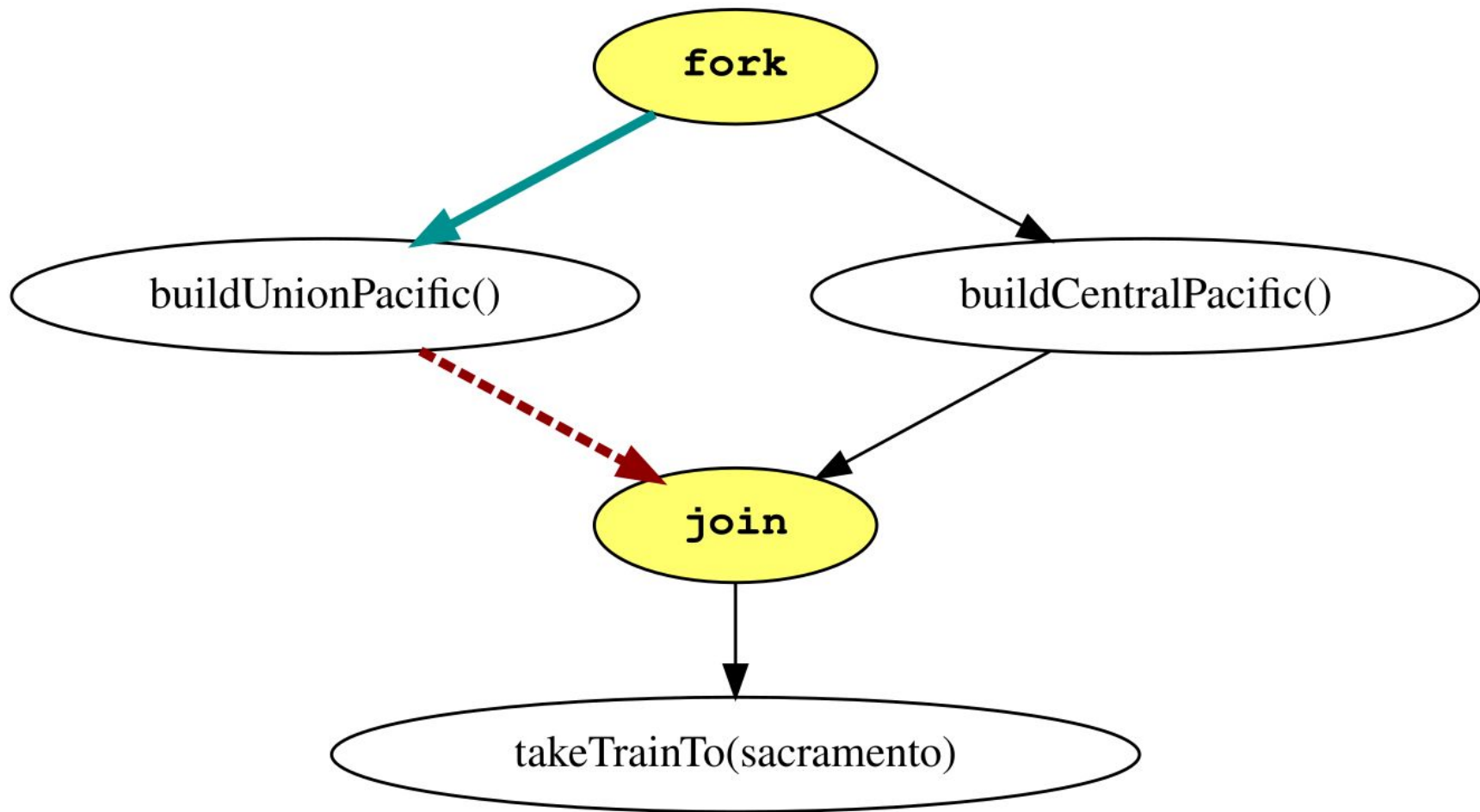
credit for Kirby to Claire Pan

can improve: More office hours could be nice

Reach out to me if you would like to be a TA next semester

AllTestSuite





I snatch'd one half out of the jaws of death.

Twelfth Night. Act III, Scene IV

map task

	A	D	E	F	G	H	I	J	K	L	...	Z
1. All's Well That Ends Well	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
2. As You Like It	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
3. Comedy of Errors	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
4. Love's Labour's Lost	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
5. Measure for Measure	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
6. Merchant of Venice	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
7. Merry Wives of Windsor	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
8. Midsummer Night's Dream	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
9. Much Ado about Nothing	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
10. Taming of the Shrew	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
11. Tempest	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
12. Twelfth Night	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
13. Two Gentlemen of Verona	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
14. Winter's Tale	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
15. Antony and Cleopatra	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
16. Coriolanus	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
17. Cymbeline	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
18. Hamlet	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
19. Julius Caesar	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
20. King Lear	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
21. Macbeth	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
22. Othello	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
23. Romeo and Juliet	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
24. Timon of Athens	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
25. Titus Andronicus	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>
26. Troilus and Cressida	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	Dict<K,A>	...	Dict<K,A>

Cowards die many times before their deaths; The valiant never taste of death but once.

Julius Caesar. Act II, Scene II

Thanks for Persevering () -> { }

1. [Half & Half Nucleobase Count](#)
2. [Race Condition: Image Batch](#)
3. [Stack](#)
4. [ForkLoop Images](#)
5. [Ranges](#)
6. [Coarsening \(N-Way Split\) Nucleobase Count](#)
7. [ForkLoop 2d](#)
8. [Parallel N-Queens](#)
9. [Divide and Conquer Nucleobase Count](#)
10. [Merge Sort](#)
11. [Floodfill](#)
12. [Fibonacci](#)
13. [NotThreadSafeHashTable](#)
14. [Higher-Order Functions: Map and Reduce](#)
15. [Mapper<E,K,V>](#)
16. [AccumulatorCombinerReducer<V,A,R>](#)
17. [MutualFriends MR App](#)
18. [Bottleneck MapReduce Framework](#)
19. [Matrix MapReduce Framework \(Part A\)](#)
20. [Matrix MapReduce Framework \(Part B\)](#)
21. [Cholera MR App](#)
22. [Chess MR App](#)
23. [Windbag MR App](#)
24. [Powers Of 2 Iterable](#)
25. [Parallel Scan](#)
26. [Pack](#)
27. [Quicksort](#)
28. [Concurrent Stack](#)
29. [Ordered Locks](#)
30. [All Or Nothing Locks](#)
31. [ConcurrentHashTable](#)
32. [Atomicity Races](#)
33. [Atomic Stack](#)
34. [K-mer Balance](#)
35. [K-mer Counting](#)
36. [Legged Races](#)
37. [Iterative Averaging](#)
38. [Iced Cakes Pipeline](#)
39. [Thread and Executor Service](#)
40. [Parallel Raytracer](#)








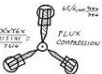

Thanks for showing up and participating

One Last Thought, Optional “Prep”

Teach Yourself Programming in Ten Years

<http://norvig.com/21-days.html>



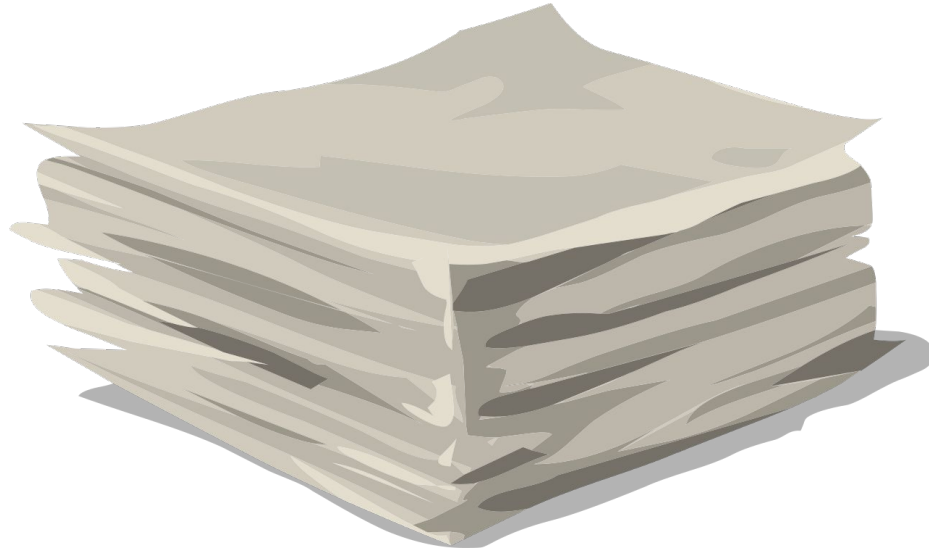
<p>Days 1 - 10 Teach yourself variables, constants, arrays, strings, expressions, statements, functions,...</p> 	<p>Days 11 - 21 Teach yourself program flow, pointers, references, classes, objects, inheritance, polymorphism,</p> 	<p>Days 22 - 697 Do a lot of recreational programming. Have fun hacking but remember to learn from your mistakes.</p> 
<p>Days 698 - 3648 Interact with other programmers. Work on programming projects together. Learn from them.</p> 	<p>Days 3649 - 7781 Teach yourself advanced theoretical physics and formulate a consistent theory of quantum gravity.</p> 	<p>Days 7782 - 14611 Teach yourself biochemistry, molecular biology, genetics,...</p> 
<p>Day 14611 Use knowledge of biology to make an age-reversing potion.</p> 	<p>Day 14611 Use knowledge of physics to build flux capacitor and go back in time to day 21.</p> 	<p>Day 21 Replace younger self.</p> 

As far as I know, this is the easiest way to "Teach Yourself C++ in 21 Days".

Reflect, Synthesize, Ask, and Turn In Worksheets

Note: you are responsible for getting the worksheets to the us.

Do not leave a pile on the end of your table and hope.



shout out: Everyone! Everyone I encountered or asked for help was always encouraging!

I agree. WashU students give me hope.

shout out: My group members

shout out: Shoutout the TAs for the help!