Week 12 Lab Session

CS2030S AY21/22 Semester 2
Lab 14B

7 Apr 2022

Yan Xiaozhi (David)
@david_eom
yan_xiaozhi@u.nus.edu

Admin

- Contact tracing & QR code
- Lab 7 grading
  - Will finish marking by next Monday
  - Lab 7 will not be included in PE 2
- PE 2 this coming Saturday
  - Exam accounts are open from 8am today to 12pm tomorrow
  - Practice logging in using exam account
  - No extra time will be given to peeps who failed to follow login instructions

PE 2 Logging In Options

- Option 1: tunnelling through `stu`
  - `ssh -t <STU> ssh <PE>`
  - No longer recommended, as `stu` cannot handle many logins at the same time
  - Alternatively, log in to `stu` between 9.00-9.30am, what for the public password to be released
- Option 2: Using SoC VPN
  - `ssh <plabid>@peXXX.comp.nus.edu.sg`
  - Bypass `stu` altogether

Lab 7 Feedback

Grading Scheme

- 22 marks correctness, 2 marks documentation, at most -2 for styling violations
- -5 if change types of head or tail
- -1 for each raw type and abuse of `@SuppressWarnings`
- -1 for each incorrect PECS
  - `iterate` and `generate` do not need PECS
- -0.5 for each case where dynamic binding is not used
- -1 for each case where it is not “lazy” enough
- -1 for each use `Maybe::get` (unless obvious)

head & tail

- Explicit checking:
- ```
  public T head() {
    if (this.head.get().equals(Maybe.none())) {
      return this.tail.get().head();
    } else {
      return this.head.get();
    }
  }
  ```
  ```

head & tail

- ```java
  public T head() {
      return this.head.get()
          .orElseGet(() -> this.tail.get().head());
  }
  ```

- ```java
  public InfiniteList<T> tail() {
      return this.head.get()
          .map(x -> this.tail.get())
          .orElseGet(() -> this.tail.get().tail());
  }
  ```

map & filter

- ```
  public <R> InfiniteList<R> map(
      Transformer<? super T, ? extends R> mapper) {
    return new InfiniteList<R>(
      this.head.map(x -> x.map(mapper)),
      this.tail.map(l -> l.map(mapper))
    );
  }
  ```

- ```
  public InfiniteList<T> filter(
      BooleanCondition<? super T> predicate) {
    return new InfiniteList<T>(
      this.head.map(x -> x.filter(predicate)),
      this.tail.map(l -> l.filter(predicate))
    );
  }
  ```

`limit`

- If `n == 0`:
  return a `Sentinel`
- Else:
  If `head` filtered, call `limit(n)` on `tail`
  If `head` unfiltered, call `limit(n - 1)` on `tail`
- ```
  public InfiniteList<T> limit(long n) {
    if (n <= 0) { return InfiniteList.sentinel(); }
    return new InfiniteList<T>(
      this.head,
      this.tail.map(
        list -> this.head.get().map(x -> list.limit(n - 1))
                               .orElseGet(() -> list.limit(n))
      )
    );
  }
  ```
  ```

toList

- ```java
  public List<T> toList() {
    ArrayList<T> array = new ArrayList<>();
    InfiniteList<T> list = this;
    while (!list.isSentinel()) {
      list.head.get().consumeWith(array::add);
      list = list.tail.get();
    }
    return array;
  }
  ```
  ```

`takeWhile`

- `head`:
  - If `head` is not filtered and predicate is `true`, keep `head`
  - Otherwise, filter and set to `None`
- `tail`:
  - If `head` is not filtered and predicate is `false`, return a `Sentinel`
  - Otherwise, `takeWhile` on `tail`

takeWhile

- ```
  public InfiniteList<T> takeWhile(BooleanCondition<? super T> cond) {
    Lazy<Boolean> filtered = head.filter(maybe -> maybe.isNone());
    Lazy<Boolean> failTest = head.filter(maybe -> maybe.filter(cond).isNone());

    Lazy<Maybe<T>> h = filtered
      .combine(failTest, (x, y) -> !x && !y)
      .map(x -> x
            ? head.get()
            : Maybe.none());

    Lazy<InfiniteList<T>> t = filtered
      .combine(failTest, (x, y) -> !x && y)
      .map(x -> x
            ? sentinel()
            : tail.map(l -> l.takeWhile(cond)).get()
          );

    return new InfiniteList<>(h, t);
  }
  ```
  ```

reduce

- ```
  public <R> R reduce(
        R identity, Combiner<R, ? super T, R> accumulator) {
      R result = identity;
      InfiniteList<T> list = this;
      while (!list.isSentinel()) {
        final R tmp = result;
        result = list.head.get()
            .map(h -> accumulator.combine(tmp, h))
            .orElse(result);
        list = list.tail.get();
      }
      return result;
    }
  ```
  ```

Asynchronicity

Motivation

- Ways to improve computer performance
  - Using faster algorithms
    - But optimal algorithms have been found for most usage :(
  - Enhancing computer hardware
    - But Moore’s Law might not apply in the near future :(
- Another way to improve the performance: splitting workload
  - Parallel / concurrent computing

Parallel Computing

- One of the focus areas for computer science!
- Will be further explored CS2106 Operating Systems
    - Race conditions
    - Deadlocks/Livelocks

Parallel Computing

This focus area aims to give students the skills to understand parallelis
take full advantage of the latest hardware. Read more ...

Primaries

- **CS3210** Parallel Computing
- **CS3211** Parallel and Concurrent Programming
- **CS4231** Parallel and Distributed Algorithms
- **CS4223** Multi-core Architecture

Electives

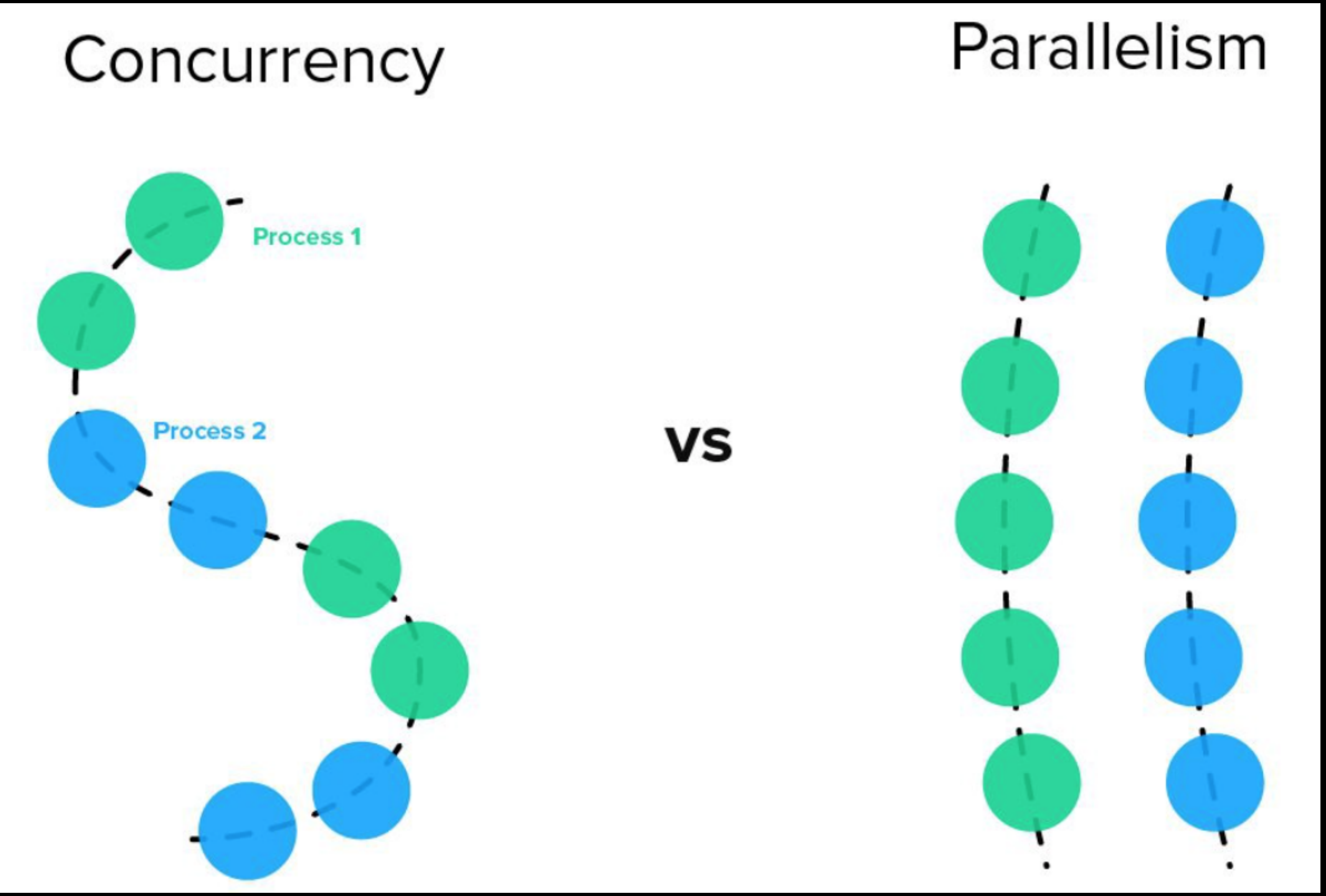
- **CS5222** Advanced Computer Architectures
- **CS5223** Distributed Systems
- **CS5224** Cloud Computing
- **CS5239** Computer System Performance Analysis
- **CS5250** Advanced Operating Systems

Parallelism - Example

- Given `n` numbers and `k` workers, find the sum of all numbers
- Sequential solution:
  - Simply add together -> `n - 1` operations
  - O(n)
- Parallel solution:
  - Divide numbers into `k` partitions
  - Each worker computes the sum of n/k numbers -> `n/k - 1` operations
  - Once every worker is done, sum up the results from the k sums -> `k - 1` operations
  - O(n/k + k)

Parallel vs Concurrent

- Concurrent: processing multiple tasks
- Parallel: processing multiple tasks **<u>at the same time</u>**
- All parallel programs are concurrent
- Not all concurrent programs are parallel



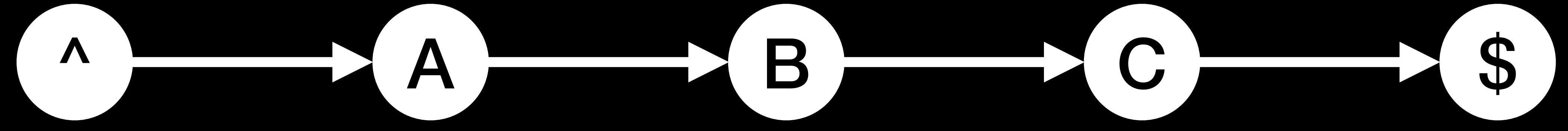
Parallel vs Concurrent

- What's the point of concurrency then?
- Useful when a thread is forced to wait
  - Waiting for input (unknown amount of time)
  - Waiting for web request (~100ms)
  - Reading from hard drive (~10ms)
- In contrast, each instructions take ~1ns
- Massive waste of computation resource and time if done without concurrency

Concurrency - Example

- Assume process makes three requests to a web server, taking 100ms, 150ms and 200ms respectively

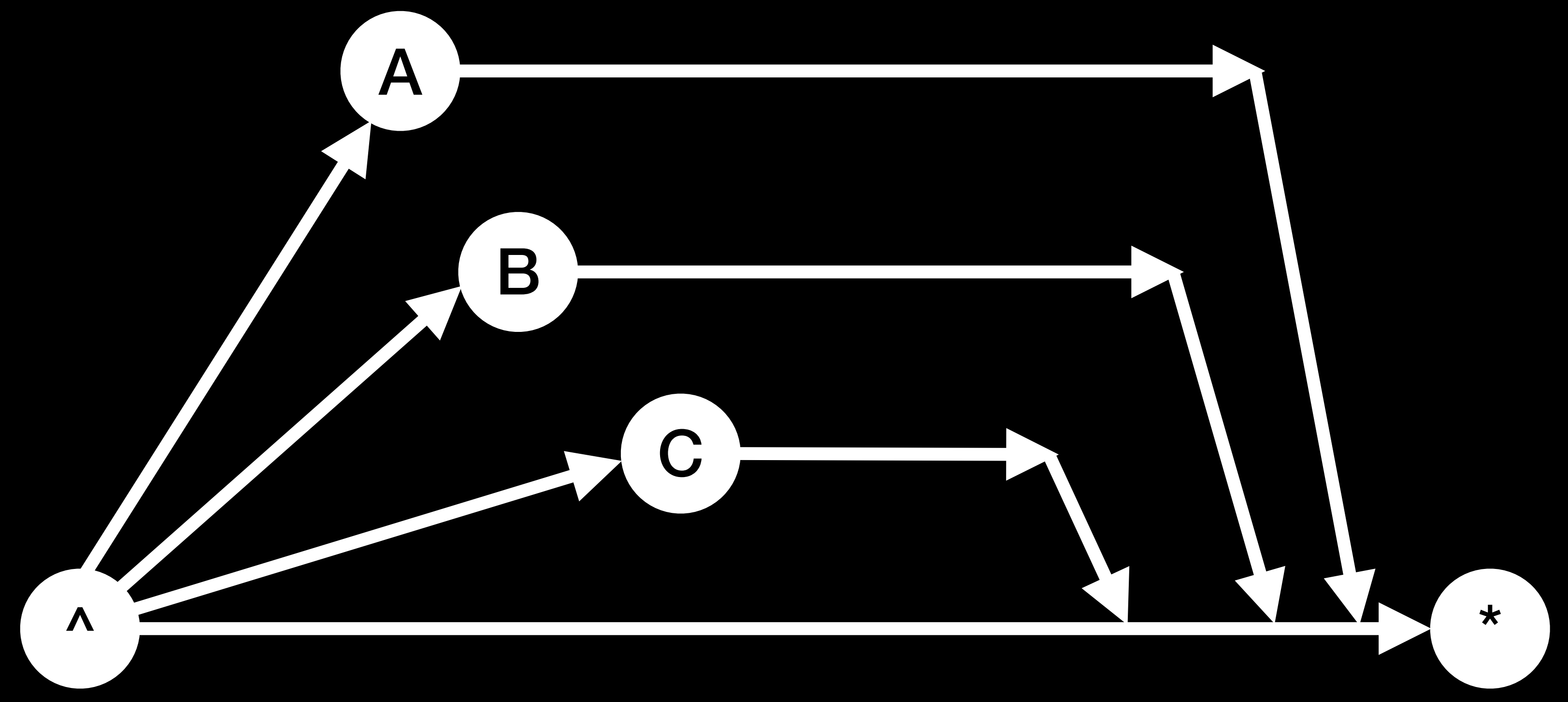
- Without concurrency:



  - A (100ms) -> B (150ms) -> C (200ms)

  - Total time taken: 450ms

- With concurrency:



  - Spawn three different threads to make queries

  - Computation not done locally, simply wait threads to return the main thread

  - Total time taken: 200ms

Problems with Parallelisation

- Not all programs can be parallelised
  - Associativity
- Overheads in parallelisation may negate the performance gain
  - Spawning new threads
  - Splitting up data
- Bottlenecks
  - Using shared data

Java Thread Pools

- Execution of program can be split into “threads” - units of work
- Each program can be split into one or more threads, each thread may be allocated one or more cores
- Java does this with thread pools
    - Java API that enables parallelism by allowing creation of threads
    - Any free allocated core will retrieve 1 thread to execute
        - If program only allocated 1 CPU core, parallelism cannot be achieved
    - Order of execution of threads cannot be guaranteed

Lab 8 Overview

Lab 8: Keep Your World Moving

- Given current stop `S` and search string `Q`, returns the list of busses serving `S` that also serves any stop with a description containing `Q`
- Queries a web API, but is is synchronous and slow as it waits for each query
-
```
for each pair (bus stop, string):
  get the bus services serving the stop
  for each bus service:
    get the bus stops served by the service
    look for matching string
```
- Your task: convert it to asynchronous queries

Lab 8: Keep Your World Moving

- Sample output:

- ```
  Search for: 16189 <-> Clementi: From 16189
  - Can take 96 to:
    - 17171 Clementi Stn
    - 17091 Aft Clementi Ave 1
    - 17009 Clementi Int
  - Can take 151 to:
    - 17091 Aft Clementi Ave 1
  - Can take 151e to:
    - 17091 Aft Clementi Ave 1
  Took 11,084ms
  ```
  ```

Lab 8: Keep Your World Moving

- Code given:
  - `BusStop` & `BusService`: encapsulates a bus stop and a bus service
  - `BusAPI`: provides interface to query the API
  - `BusSg`: implements the bus route query above
  - `BusRoutes`: encapsulates the result of a query
  - `Main`: reads from `stdin` and invokes `BusSg`'s methods, and print the result

Lab 8: Keep Your World Moving

- In BusAPI.java:
  - `response = client.send(...);`
  - Invokes Java class `HttpClient::send`, which is blocking
- Instead, change to:
  - `response = client.sendAsync(…);`
  - Triggers a sequence of required changes to make program asynchronous
  - `response` will be a `CompletableFuture<HttpResponse<T>>`

Lab 8: Keep Your World Moving

- Some of the changes include:
  - `BusAPI::getBusStopsServedBy` now returns `CF<String>`
  - `BusAPI::getBusServicesAt` now returns `CF<String>`
  - `BusStop::getBusServices` now returns `CF<Set<BusService>>`
  - `BusRoutes` now stores `CF<Set<BusStop>>`
  - `BusRoutes::description` now returns `CF<String>`
  - …

Lab 8: Keep Your World Moving

- Do NOT call `CF::join` or `CF::get` except in the final step in `main`
  - Or else your code will become synchronous
  - Only in `main`, wait for `CF`s to complete to use `allOf` or `join`
  - Then print out the description

Happy coding! 👨‍💻

To Conclude…

Final Words

- CS2030S labs are “rigid”
  - There is no absolute right solution to real-life designs
  - Apply principles and frameworks you have learnt to do what you think is the best
- There is still a lot to be learnt
  - CS2103T Software Engineering
  - CS2106 Operating Systems / CS3210 Parallel Computing
  - CS3219 Software Engineering Principles and Patterns

Thanks for the past 10 weeks
and all the best for your finals! 🍀