Victoria University
of Wellington, New Zealand
Te Whare Wananga o te
Upoko o te Ika a Maui
Aotearoa

SWEN221: Software Development

15: Testing II

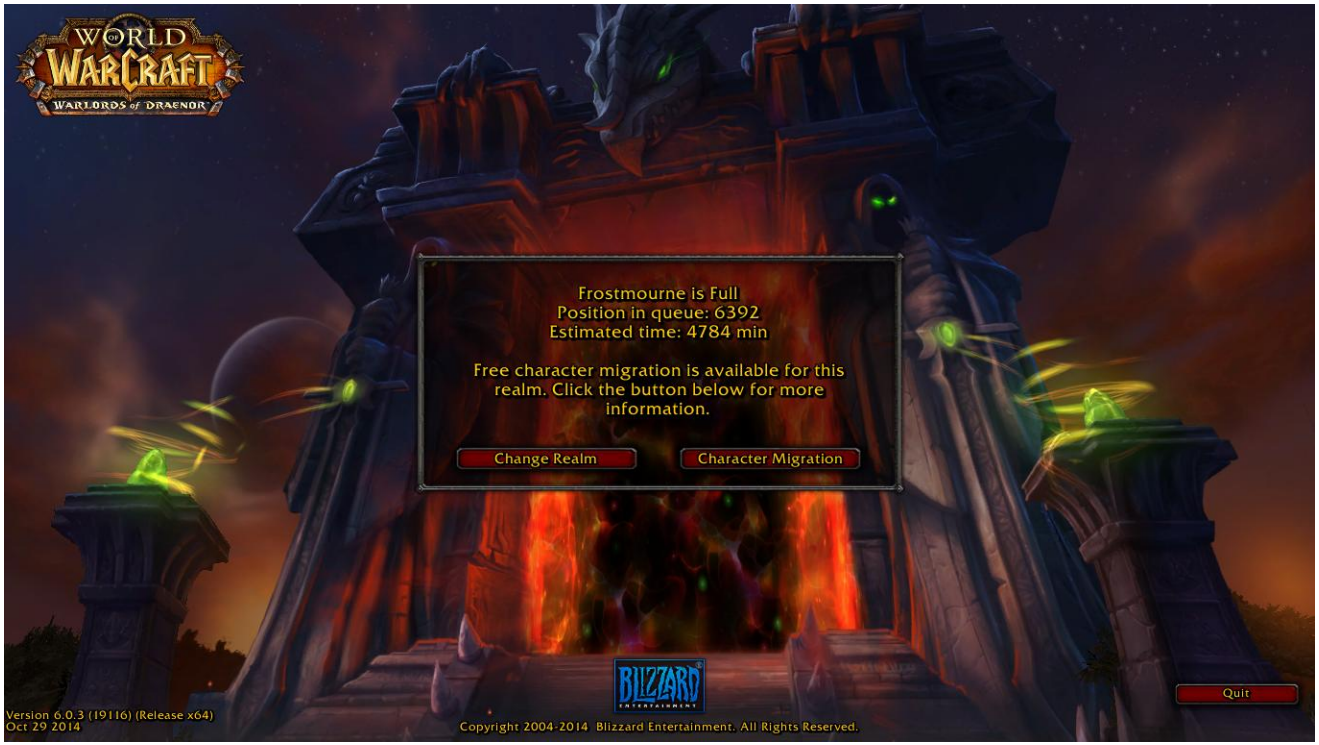
Why Testing

- We all make mistakes
  - Some may be hazardous

Why Testing

- A system may work well when only one person is using it, but not when hundreds of people are using

screenshot not from tonight



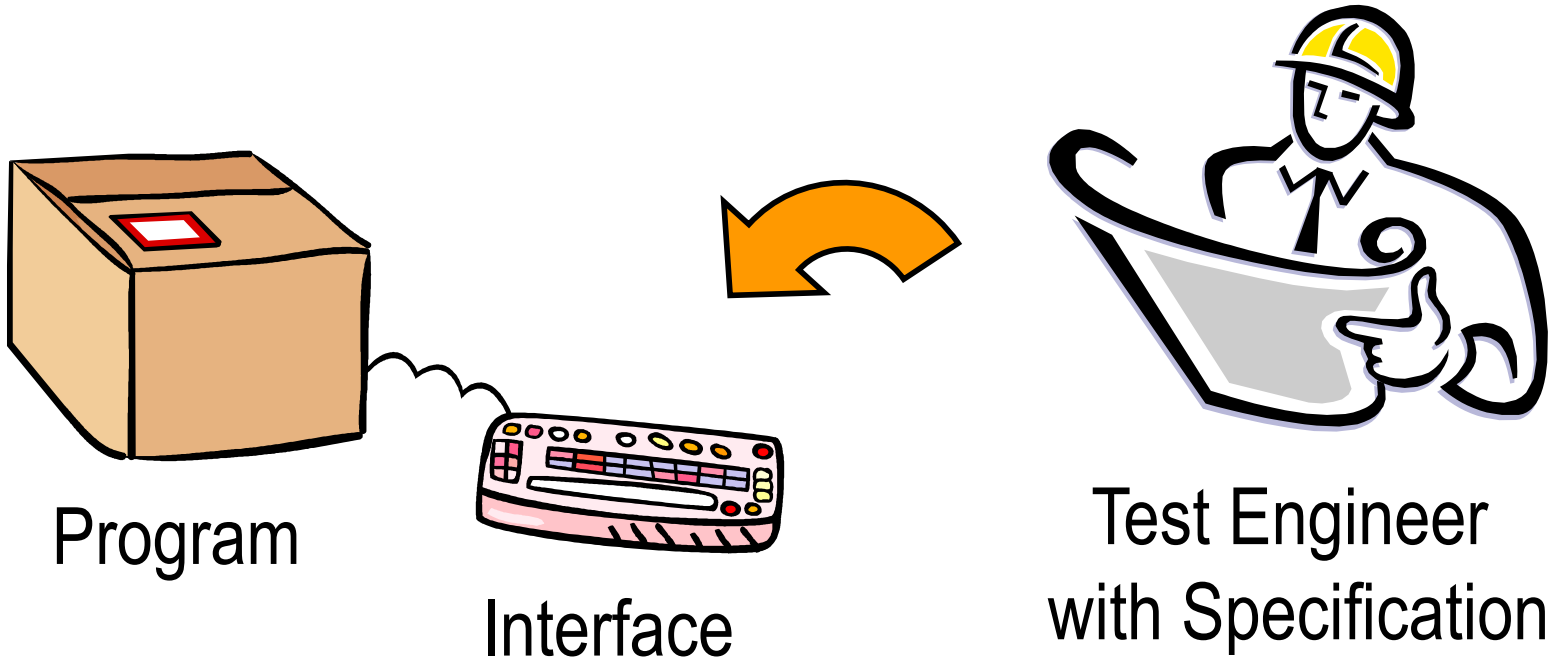
Why Testing

- A user may do some silly things that break the system

Why Testing

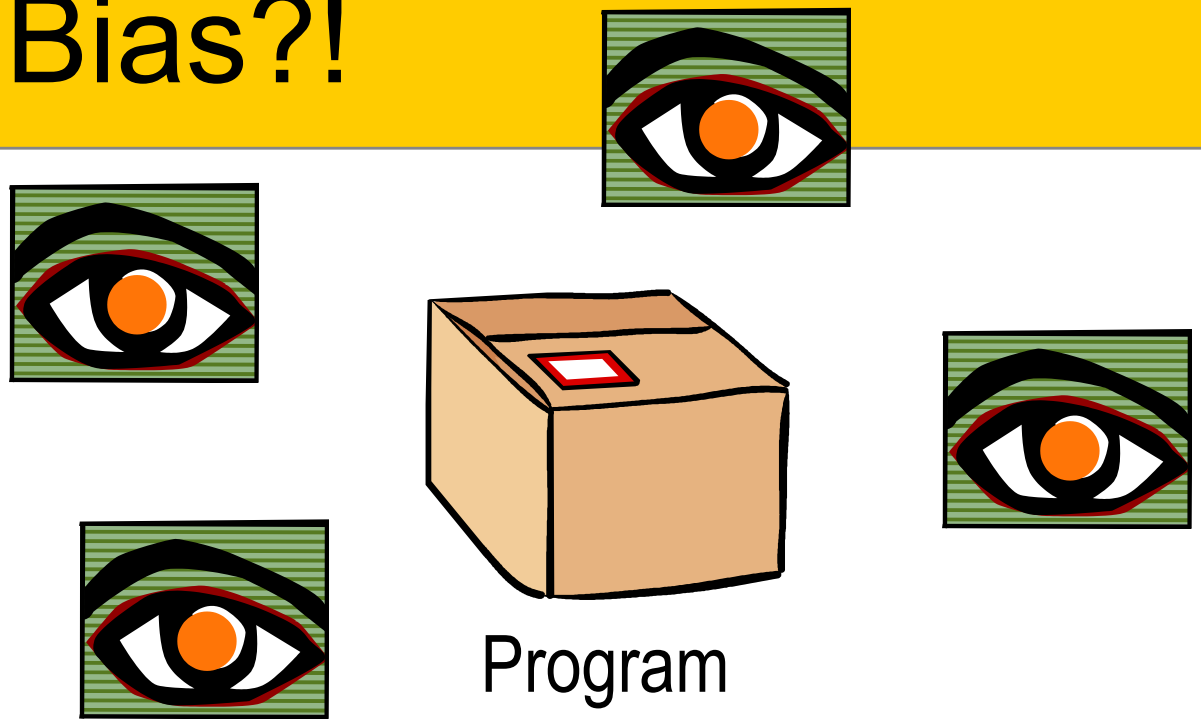
- Different browsers, devices, OS, …

Black-box Testing



- Testing *without knowledge of implementation*
  - Test cases generated directly from specification
  - Gives unbiased approach
  - Robust to implementation changes

What Bias?!



- Biases introduced by programmer:
  - Programmer may misinterpret specification
    - This misinterpretation may be repeated in his test
  - Programmer may "believe" a particular part is well-coded
    - He/she might omit tests because of this to save time
  - Programmer unlikely to represent target audience
    - What he/she finds acceptable others may not
  - Bottom-line: more eyeballs = greater chance of finding problems

Black-Box Testing (cont'd)

- Test **typical inputs**
  - Values that your program is likely to encounter
  - E.g. single pawn move for ChessView
- Test **boundary conditions**
  - Values at edges of valid input domain
  - E.g. off-by-one error:

```
int nextDay(int day) {
 // 1 <= day <= 7
 if(day > 7) { return 1; }
 else { return day + 1; }
}
```

Quiz: Find Good Boundary Tests

```
class TableRow<T> {
 private List<T> rows;

 public TableRow() { this.rows = new ArrayList<T>(); }

 public TableRow(List<T> rows) { this.rows = rows; }

 public T get(int index) { return rows.get(index); }

/**
  * Copy elements from this TableRow into parameter to
  */
 void copy(List<T> to) {
  for(int i=0; i < rows.size(); i++) {
   to.add(rows.get(i));
}}}
```

Quiz: Find Good Boundary Tests

- The tester only sees:
- A class `TableRow<T>`
- It represents a row as a `List`
- It has a constructor
  - `TableRow(List<T> rows)`
- It has two methods
  - Get the element from an index:
    `get(int index)`
  - Copy this row to the end of another list:
    `copy(List<T> to)`

Can It Pass?

```
@Test public void test1() {
 List<String> list = new ArrayList<>(Arrays.asList(”s1"));
 TableRow<String> r = new TableRow<String>(list);
 r.copy(null);
 assertEquals(r.get(0), "s1");
}
```

Can It Pass?

```
@Test public void test1() {
 List<String> list = new ArrayList<>(Arrays.asList(”s1"));
 TableRow<String> r = new TableRow<String>(list);
 r.copy(null);
 assertEquals(r.get(0), "s1");
}
```

No, NullPointerException

Can It Pass?

```
@Test public void test2() {
 List<String> list = new ArrayList<String>();
 TableRow<String> r = new TableRow<String>(list);
 list.add("s1");
 assertEquals(r.get(0), "s1");
}
```

Can It Pass?

```
@Test public void test2() {
 List<String> list = new ArrayList<String>();
 TableRow<String> r = new TableRow<String>(list);
 list.add("s1");
 assertEquals(r.get(0), "s1");
}
```

Yes

Can It Pass?

```
@Test public void test3() {
 List<String> list = new ArrayList<>(Arrays.asList(”s1"));
 TableRow<String> r = new TableRow<String>(list);
 r.copy(list);
 assertEquals(r.get(1), "s1");
}
```

Can It Pass?

```
@Test public void test3() {
 List<String> list = new ArrayList<>(Arrays.asList(”s1"));
 TableRow<String> r = new TableRow<String>(list);
 r.copy(list);
 assertEquals(r.get(1), "s1");
}
```

No, infinite loop

Can It Pass?

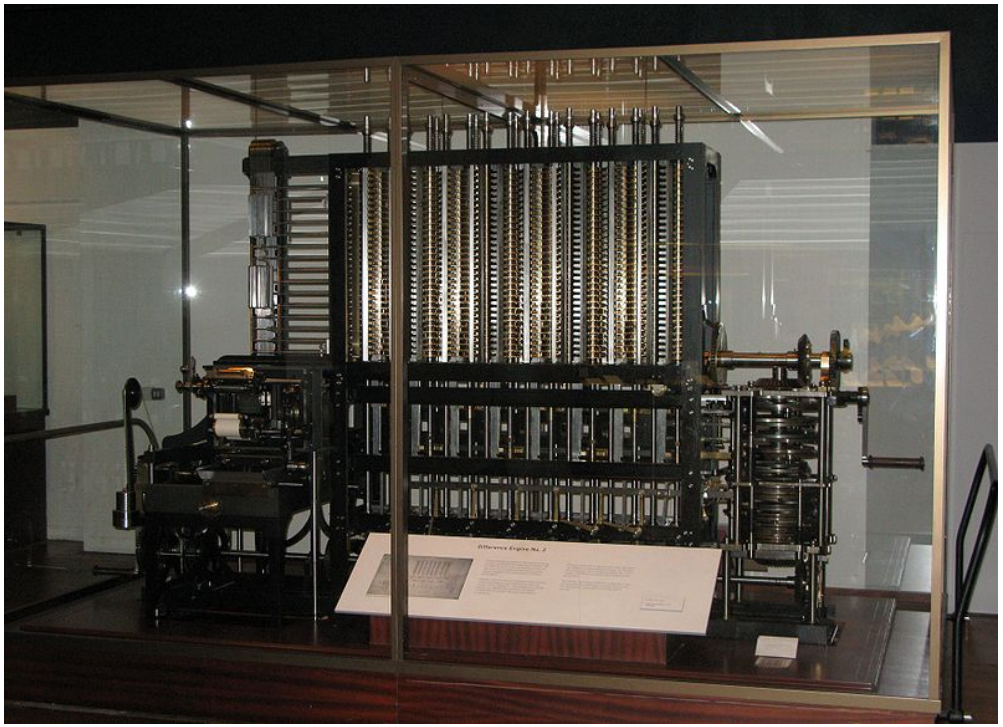
```
@Test public void test4() {
 List<String> list = new ArrayList<String>();
 list.add(null);
 TableRow<String> r = new TableRow<String>(list);
 r.copy(list);
 assertEquals(r.get(1), null);
}
```

Can It Pass?

```
@Test public void test4() {
 List<String> list = new ArrayList<String>();
 list.add(null);
 TableRow<String> r = new TableRow<String>(list);
 r.copy(list);
 assertEquals(r.get(1), null);
}
```

No, infinite loop

White-Box Testing (A.K.A. Glass-Box)



- Testing *with complete knowledge of implementation*
  - Test cases generated by looking at program code
  - Aim to reach high-degree of *code coverage*
  - Gives potentially biased approach
  - Not robust to implementation changes

White-Box testing

```
int sumSmallest(List<Integer> v1, List<Integer> v2) {
 // sum smallest list
 int r = 0;
 if(v1.size() < v2.size()) {
   for(int i=0; i < v1.size(); i++) { r += v1.get(i); }
 } else {
   for(int i=0; i < v2.size(); i++) { r += v2.get(i); }
 }
 return r;
}
```

- What's wrong with these test cases?
  - (v1=[1,5,4,3], v2=[4,2,3])
  - (v1=[4], v2=[5])
  - (v1=[5], v2=[])

White-Box testing

```
int sumSmallest(List<Integer> v1, List<Integer> v2) {
 // sum smallest list
 int r = 0;
 if(v1.size() < v2.size()) {
   for(int i=0; i < v1.size(); i++) { r += v1.get(i); }
 } else {
   for(int i=0; i < v2.size(); i++) { r += v2.get(i); }
 }
 return r;
}
```

- What's wrong with these test cases?
  - (v1=[1,5,4,3], v2=[4,2,3])
  - (v1=[4], v2=[5])
  - (v1=[5], v2=[])

Code-Coverage

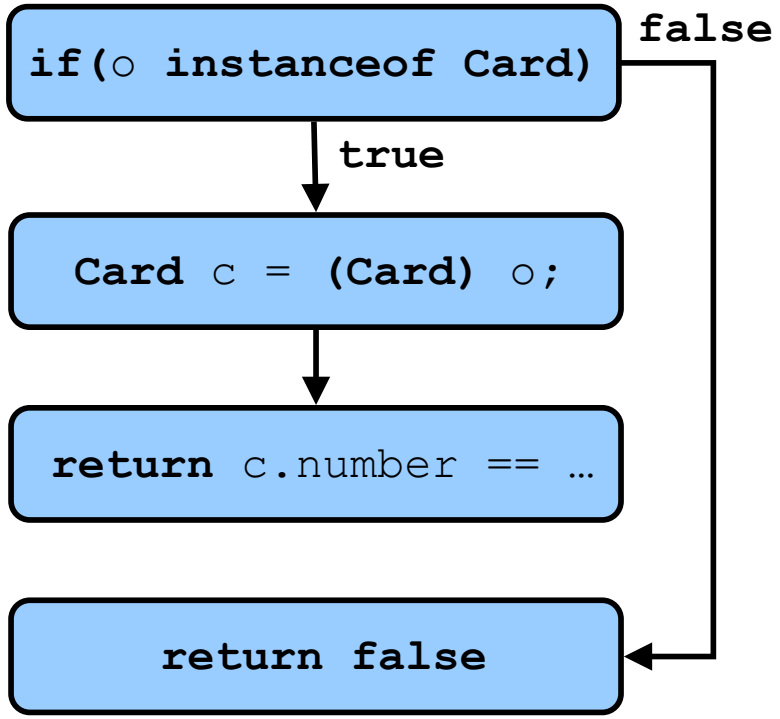
- Want test cases to cover X% of code
  - E.g. > 85% of code covered by tests
  - But, how to measure code coverage?

- Example *Coverage Criteria*:
  - **Function Coverage:** number of methods invoked / # methods
  - **Statement Coverage**: number of statements executed / # statements
  - **Branch Coverage**: number of branches where both true and false side tested / # branches

- Calculating Code Coverage
  1. Select Criteria
  2. Construct Control-Flow Representation (next slide)
  3. Mark nodes Executed Based on Tests
  4. Compute Coverage

Control-Flow Graph

```
public boolean equals(Object o) {
 if(o instanceof Card) {
   Card c = (Card) o;
   return c.number == number
          && c.suit == suit;
  }
  return false;
}
```



- Computing Coverage
  - Must be clear what counts and what doesn't
  - Requires precise representation of program
  - **Control-Flow Graph (CFG)** useful here
  - Nodes represent statements
  - Edges represent branching

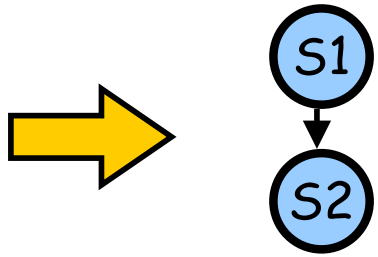
Control-Flow Graph (cont'd)

- Unit Statements
  - No branching (i.e. only one way through)
  - One node in CFG for each of these
  - E.g. assignment, method call, return, etc

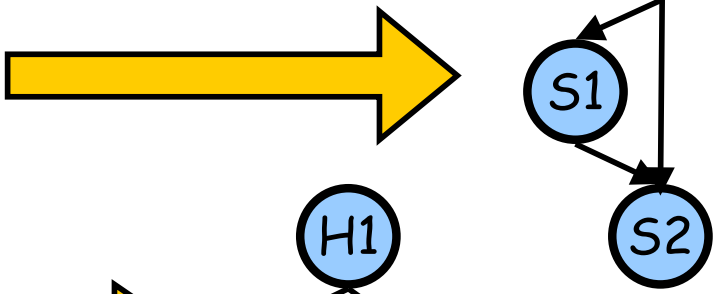
- Branching Statements
  - Cause branches
  - One node in CFG for "header"
  - E.g. ifs, for/while loops, etc

CFG Construction Examples

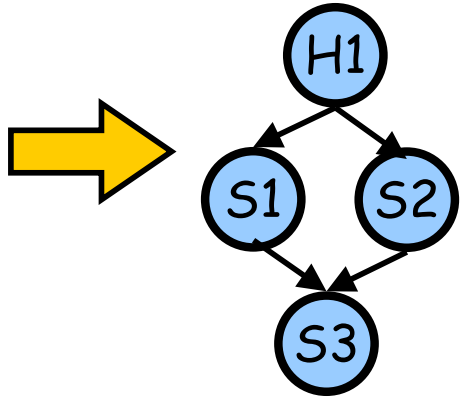
```
S1 ; S2
```



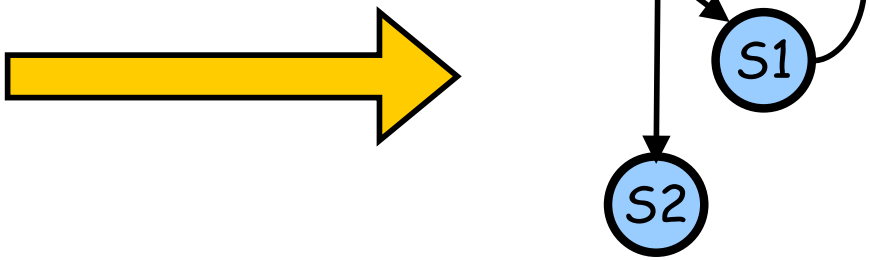

```
if(…) { S1 }
S2
```




```
if(…) { S1 } else { S2 }
S3
```




```
while(…) { S1 }
S2
```



Draw CFG For Them!

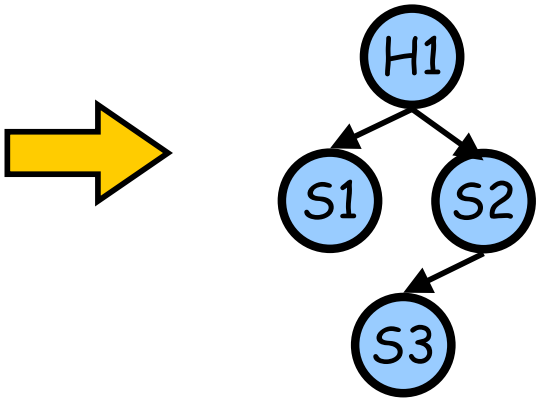
```
if(…) { return; }
else { S2 }
S3
```

```
if(…) { S1 ; S2 }
S3
```

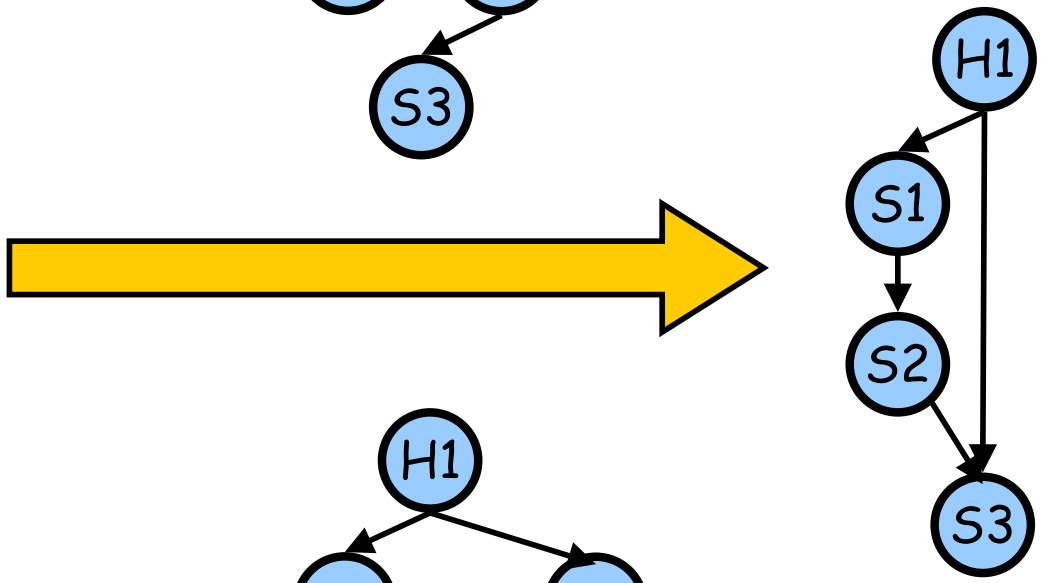
```
if(…) { S1 }
else if(…) { S2 }
else { S3 }
S4
```

Draw CFG For Them!

```
if(…) { return; }
else { S2 }
S3
```




```
if(…) { S1 ; S2 }
S3
```




```
if(…) { S1 }
else if(…) { S2 }
else { S3 }
S4
```