departamento de informática
FACULDADE DE CIÊNCIAS E TECNOLOGIA
UNIVERSIDADE NOVA DE LISBOA

Concurrency Errors (3)

lecture 24 (2020-05-26)

Master in Computer Science and Engineering

— Concurrency and Parallelism / 2019-20 —

João Lourenço <joao.lourenco@fct.unl.pt>

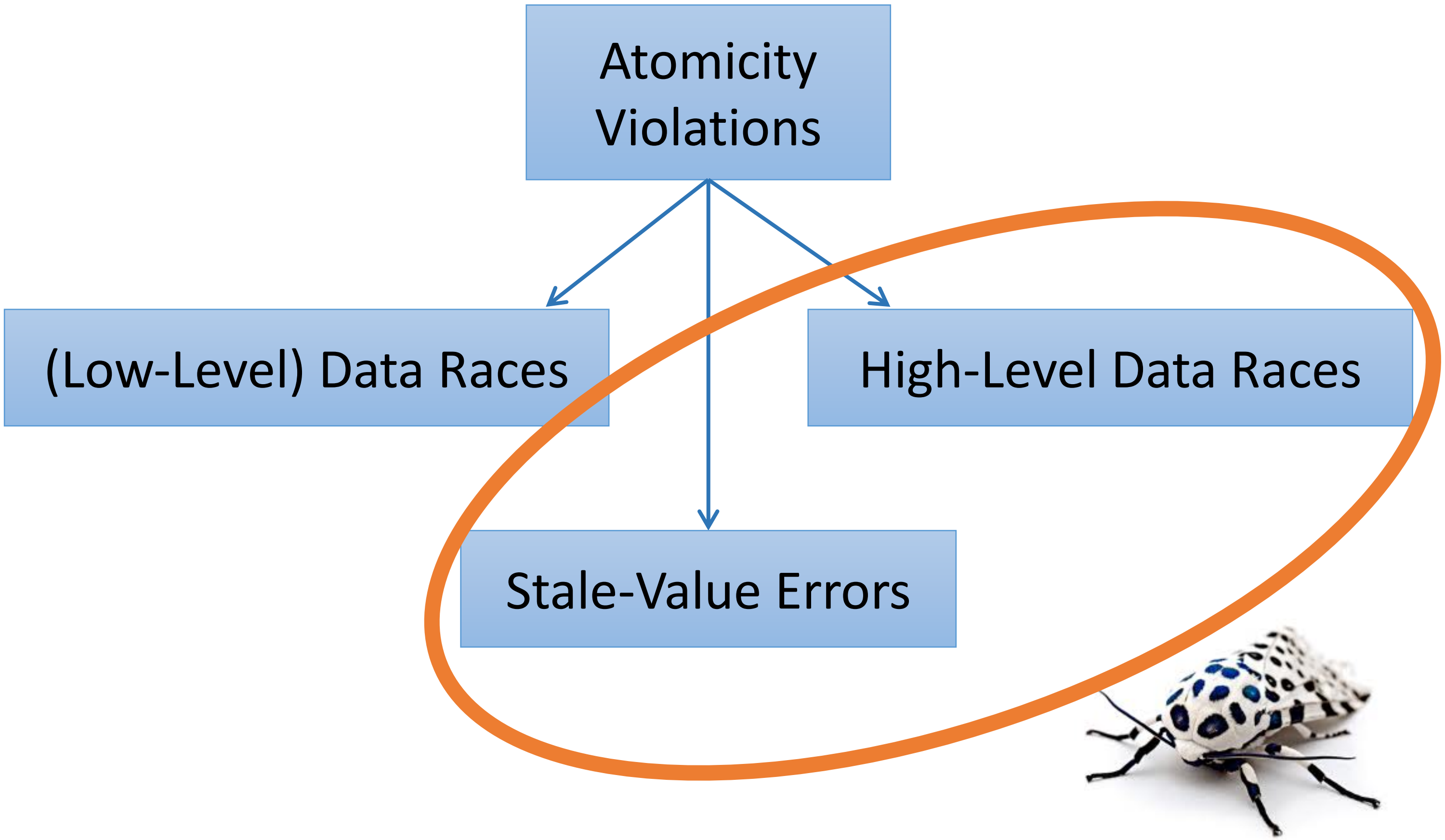
Agenda

- Concurrency Anomalies
- Assigning Semantics to Concurrent Programs
- Concurrency Errors
  - Detection of data races
  - Detection of high-level data races and stale value errors
  - Detection of deadlocks

Concurrency Errors

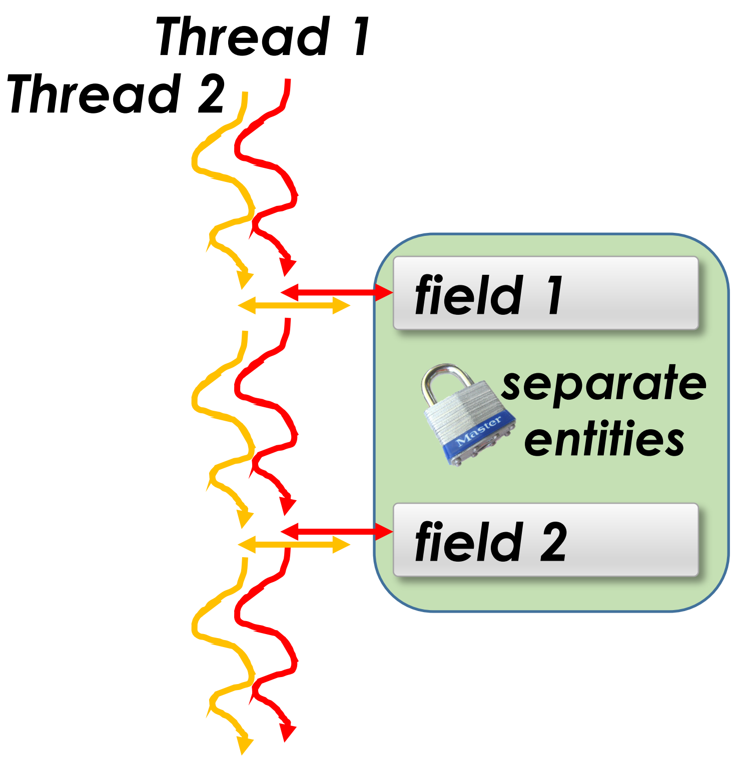
Detection of High-level Data Races and Stale-value Errors [Artho03]

Concurrency Anomalies



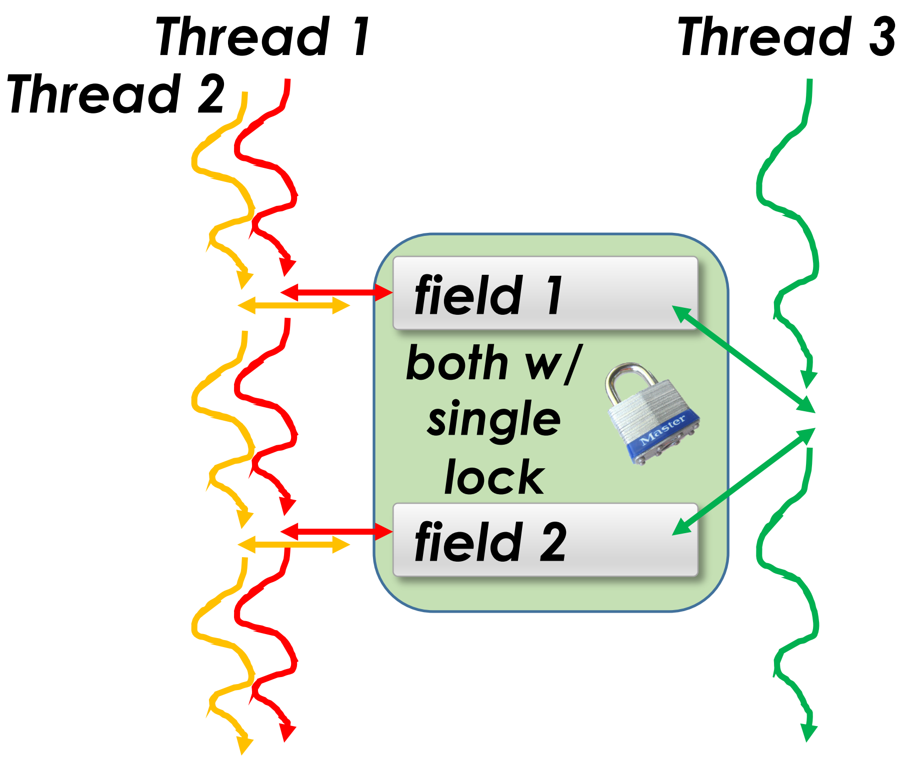
High-Level Data Race

- Wrongly defined atomic blocks



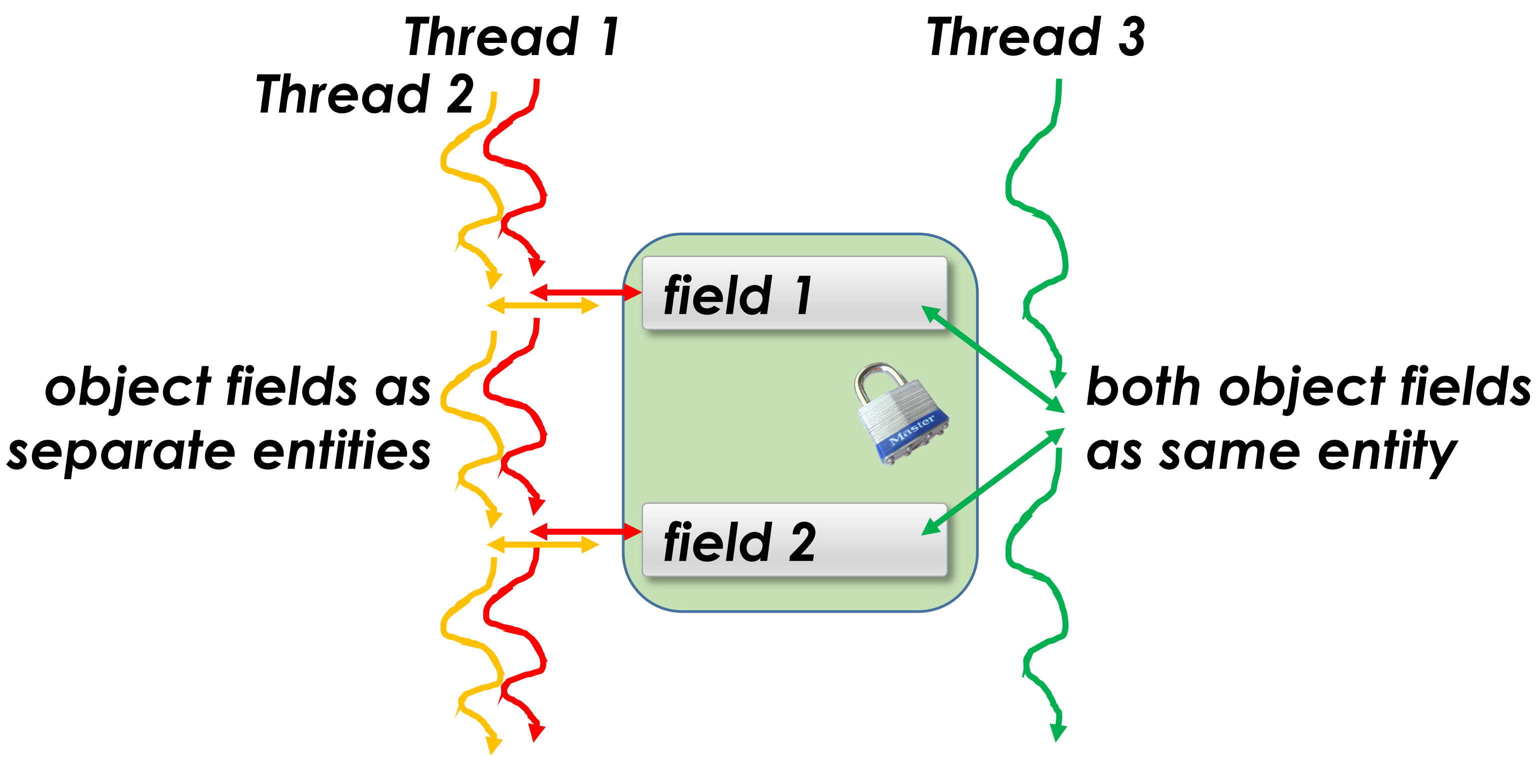
High-Level Data Race

- Wrongly defined atomic blocks



High-Level Data Race

- Wrongly defined atomic blocks



High-Level Data Races

```java
public synchronized
int getX(){
  return this.x;
}
```

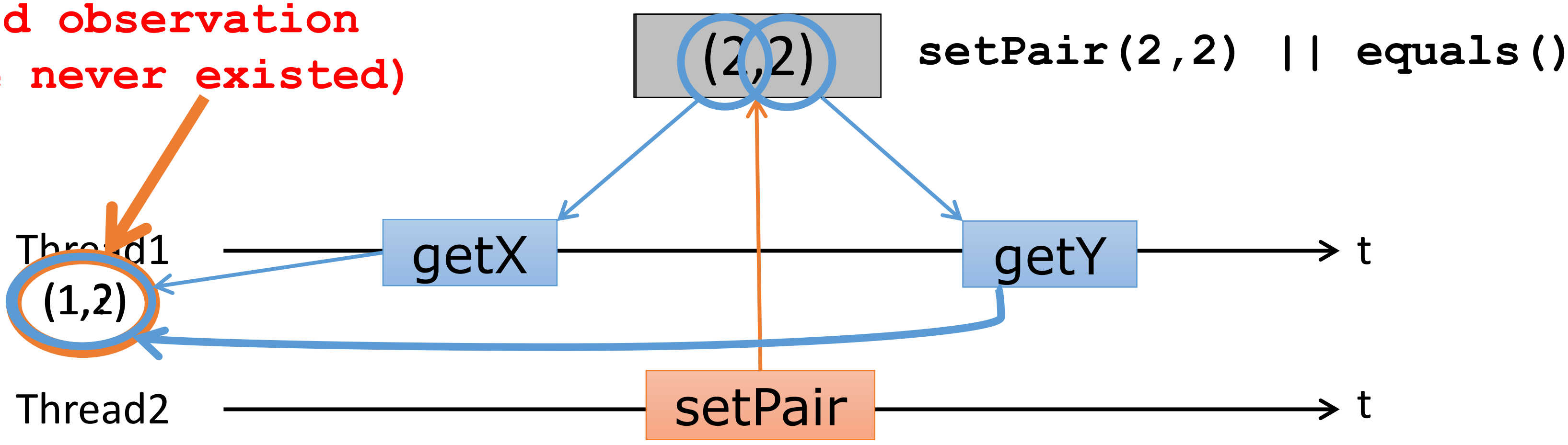
```java
public synchronized
int getY(){
  return this.y;
}
```

Shared variables: x, y

```java
// Thread 1
public boolean equals(){
  int loc_x = getX(); // synchr
  int loc_y = getY(); // synchr
  return loc_x == loc_y;
}
```

```java
// Thread 2
public synchronized
int setPair(int v1, int v2){
    x = v1;
    y = v2;
}
```

Invalid observation (state never existed)



Stale-Value Errors

- Caused by privatization of shared values

Shared variables: x, y

```
void yEqualsXTimesTwo () {
        int local = getX()   // Atomic
        // local may have a stale value
        setY(2 * local) ; // Atomic
}

private synchronized int getX() {
        return x ;
}

@Atomic
private synchronized void setY(int value) {
    y = value ;
}
```

x

write(x)?

View of an Atomic Block [Artho03]

- A view of an atomic block B — V(B) — is the set of variables accessed inside the atomic code block B

View of an Atomic Block

- A view of an atomic block B — $V(B)$ — if the set of variables accessed inside the atomic code block B
- The *read view* of B — $V_R(B) \subseteq V(B)$ — is the set of variables read inside the atomic code block B
- The *write view* of B — $V_W(B) \subseteq V(B)$ — is the set of variables written inside the atomic code block B

Views Analysis [Artho03]

- View: set of shared variables accessed atomically

```
public synchronized void incX(){
    int local = x;
    setX(local + 1)
}

public void setX(int aux){
        x = aux;
}
```

V(incX) = ?

Views Analysis [Artho03]

- View: set of shared variables accessed atomically

```
public synchronized void incX(){
    int local = x;
    setX(local + 1)
}

public void setX(int aux){
        x = aux;
}
```

Which (shared) variables are accessed and how?

Views Analysis [Artho03]

- View: set of shared variables accessed atomically

```
public synchronized void incX(){
    int local = x;
    setX(local + 1)
}

public void setX(int aux){
        x = aux;
}
```

Vars(incX) = { }

Views Analysis [Artho03]

- View: set of shared variables accessed atomically

```
public synchronized void incX(){
    int local = x;
    setX(local + 1)
}

public void setX(int aux){
        x = aux;
}
```

Vars(incX) = { } ∪ {read(X)}

Views Analysis [Artho03]

- View: set of shared variables accessed atomically

```
public synchronized void incX(){
    int local = x;
    setX(local + 1)
}

public void setX(int aux){
        x = aux;
}
```

Vars(incX) = { } ∪ {read(X)} ∪ Vars(setX)

Views Analysis [Artho03]

- View: set of shared variables accessed atomically

```
public synchronized void incX(){
    int local = x;
    setX(local + 1)
}

public void setX(int aux){
        x = aux;
}
```

Vars(incX) = { } ∪ {read(X)} ∪ Vars(setX)

Vars(setX) = { }

Views Analysis [Artho03]

- View: set of shared variables accessed atomically

```
public synchronized void incX(){
    int local = x;
    setX(local + 1)
}

public void setX(int aux){
        x = aux;
}
```

Vars(incX) = { } ∪ {read(X)} ∪ Vars(setX)

Vars(setX) = { } ∪ {write(X)}

Views Analysis [Artho03]

- View: set of shared variables accessed atomically

```
public synchronized void incX(){
    int local = x;
    setX(local + 1)
}

public void setX(int aux){
        x = aux;
}
```

Vars(incX) = { } ∪ {read(X)} ∪ Vars(setX)

Vars(setX) = {write(X)}

Views Analysis [Artho03]

- View: set of shared variables accessed atomically

```
public synchronized void incX(){
    int local = x;
    setX(local + 1)
}

public void setX(int aux){
        x = aux;
}
```

$$\mathrm{Vars}(\mathrm{incX}) = \{\ \} \cup \{\mathrm{read}(X)\} \cup \mathrm{Vars}(\mathrm{setX})$$

$$\mathrm{Vars}(\mathrm{setX}) = \{\mathrm{write}(X)\}$$

Views Analysis [Artho03]

- View: set of shared variables accessed atomically

```
public synchronized void incX(){
    int local = x;
    setX(local + 1)
}

public void setX(int aux){
        x = aux;
}
```

Vars(incX) = { } ∪ {read(X)} ∪ {write(X)}

Views Analysis [Artho03]

- View: set of shared variables accessed atomically

```
public synchronized void incX(){
    int local = x;
    setX(local + 1)
}

public void setX(int aux){
        x = aux;
}
```

Vars(incX) = {read(X), write(X)}   V(incX) = { X }

Views Analysis [Dias12]

- View: set of shared variables accessed atomically

```
public synchronized void incX(){
    int local = x;
    setX(local + 1)
}

public void setX(int aux){
        x = aux;
}
```

Vars(incX) = {read(X), write(X)}

V(incX) = { X }

V_R(incX) = { X }

V_W(incX) = { X }

Views Analysis [Dias12]

- View: set of shared variables accessed atomically

```
// Thread 1
public boolean equals(){
  int loc_x = getX(); // Atomic
  int loc_y = getY(); // Atomic
  return loc_x == loc_y;
}
```

```
// Thread 2
public synchronized
int setPair(int v1, int v2){
    x = v1;
    y = v2;
}
```

```
public synchronized int getX(){
return this.x;
}
```

```
public synchronized int getY(){
return this.y;
}
```

Views Analysis [Dias12]

```
public int getX(){
return this.x;
}
```

$V(getX) = \{ X \}$ $V_R(getX) = \{ X \}$ $V_W(getX) = \{ \}$

```
public int getY(){
return this.y;
}
```

$V(getY) = \{ Y \}$ $V_R(getY) = \{ Y \}$ $V_W(getY) = \{ \}$

```
public int setPair(int v1, int v2){
    x = v1;
    y = v2;
}
```

$V(setPair) = \{ X, Y \}$

$V_R(setPair) = \{ \}$ $V_W(setPair) = \{ X, Y \}$

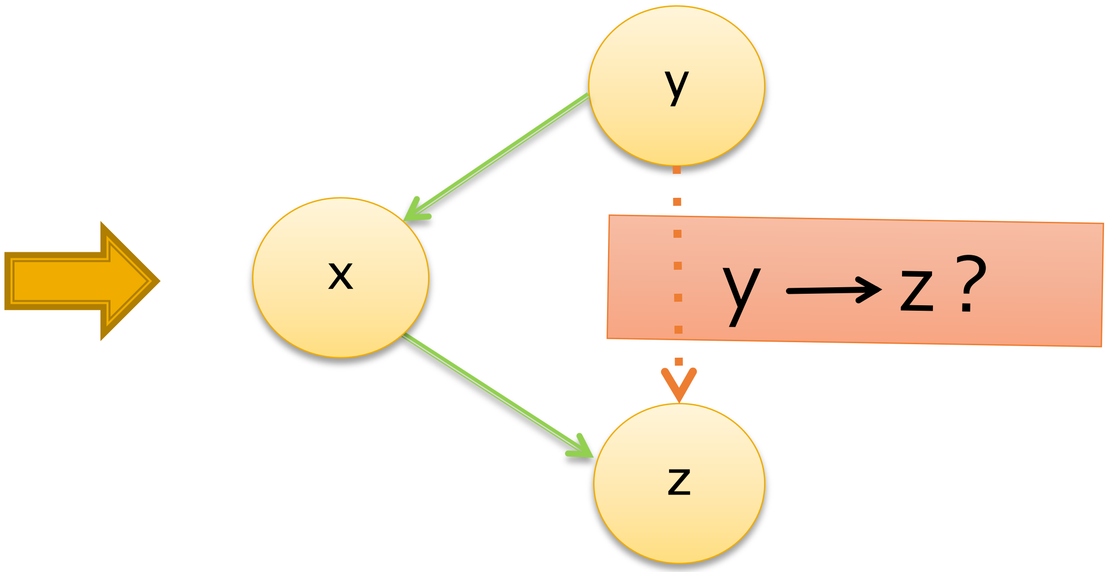
```
public boolean equals(){
  int loc_x = getX();
  int loc_y = getY();
  return loc_x == loc_y;
}
```

$V(setPair) = \{ X, Y \}$

$V_R(setPair) = \{ X, Y \}$ $V_W(setPair) = \{ \}$

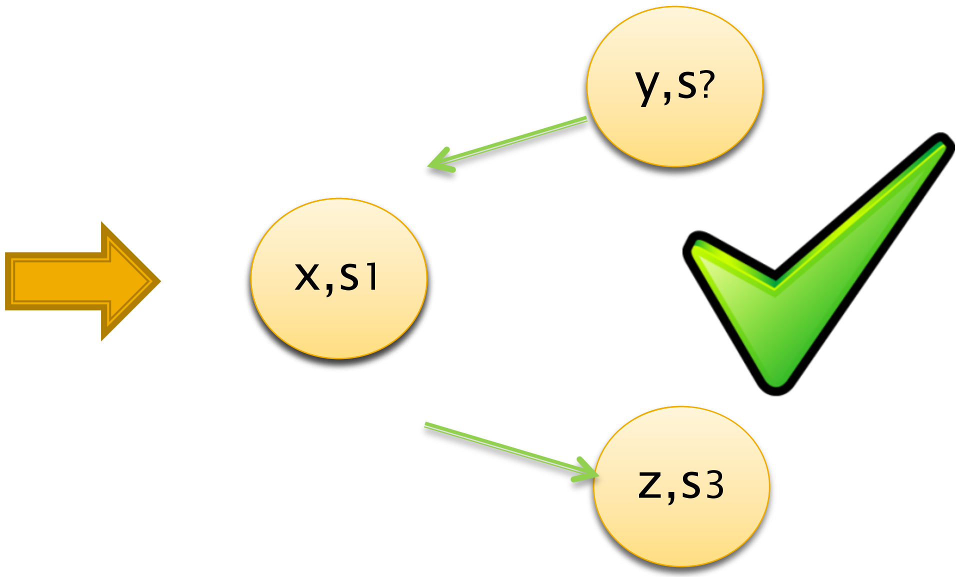
Data Dependency Analysis

```
s1: x = y;
s2: x = 2;
s3: z = x;
```



Data Dependency Analysis

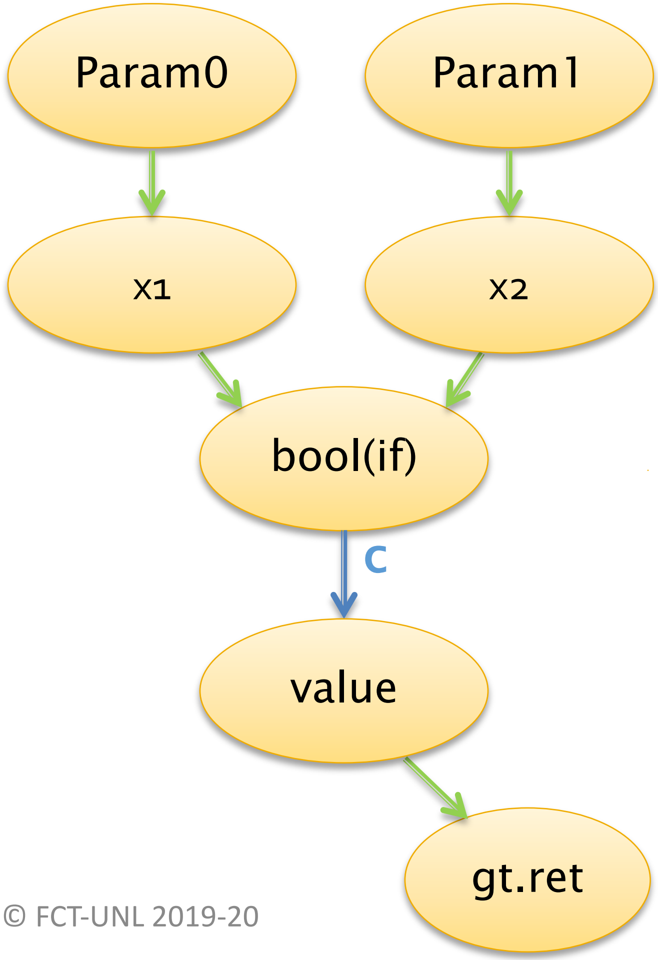
```
s1: x = y;
s2: x = 2;
s3: z = x;
```



Data Dependency Analysis

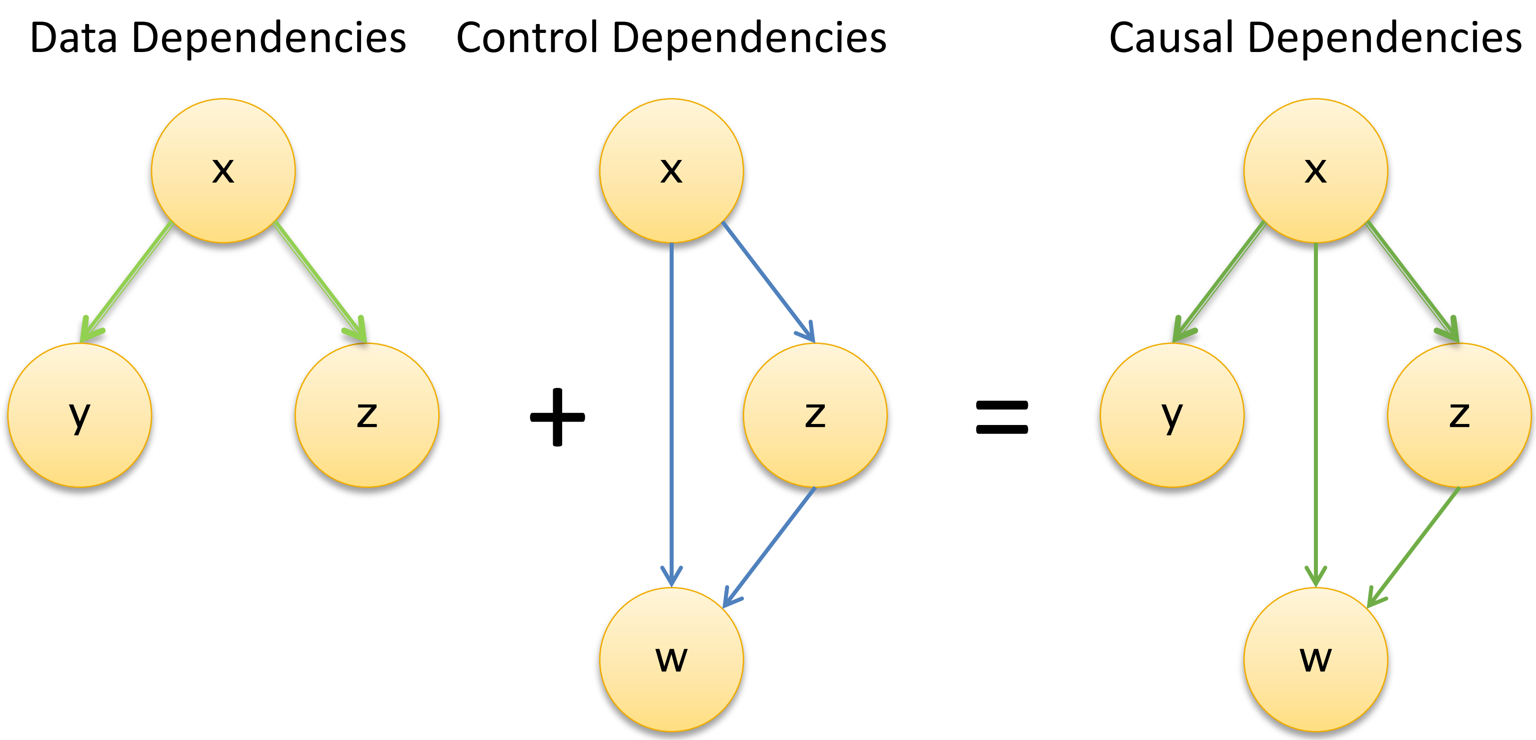
- Data Dependencies are not enough!

```
   boolean gt(int x1, int x2){
        boolean value;
s1:     if(x1>x2){
s2:            value = true;
        }else{
s3:            value = false;
        }
s4:     return value;
   }
```



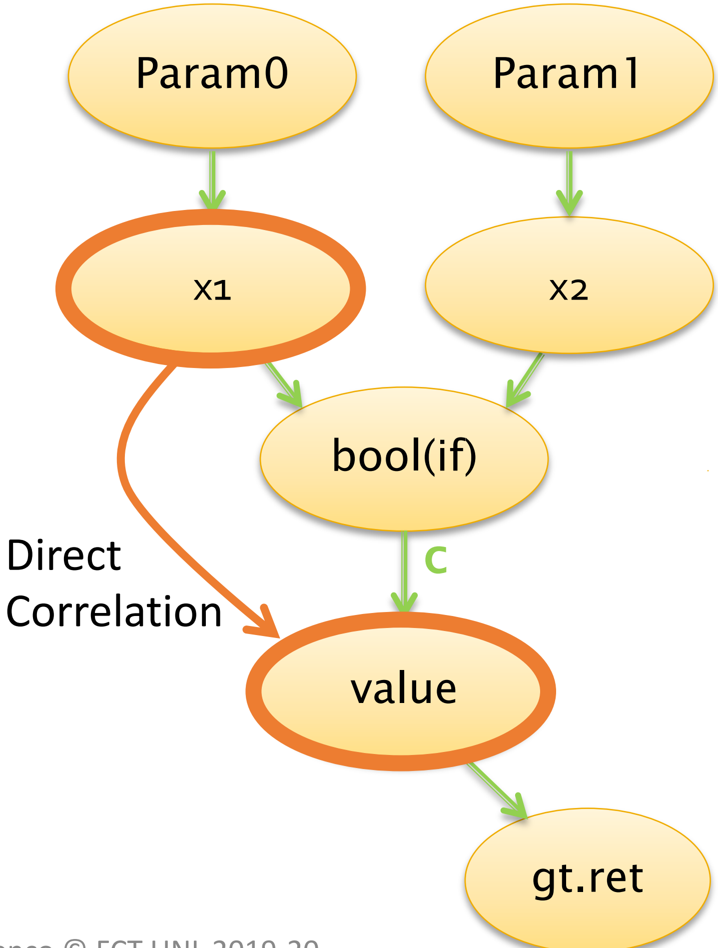
Causal Dependencies Graph

- Merge Data and Control Flow Dependencies in the Causal Dependencies Graph



Direct Correlation

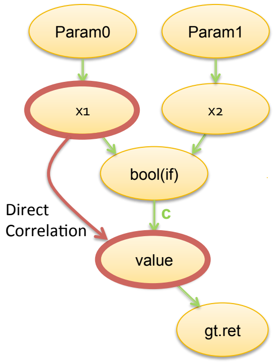
```
   boolean gt(int x1, int x2){
       boolean value;
s1:    if(x1>x2){
             value = true;
       }else{
s3:          value = false;
       }
s4:    return value;
   }
```



Variable's Correlation [Dias12]

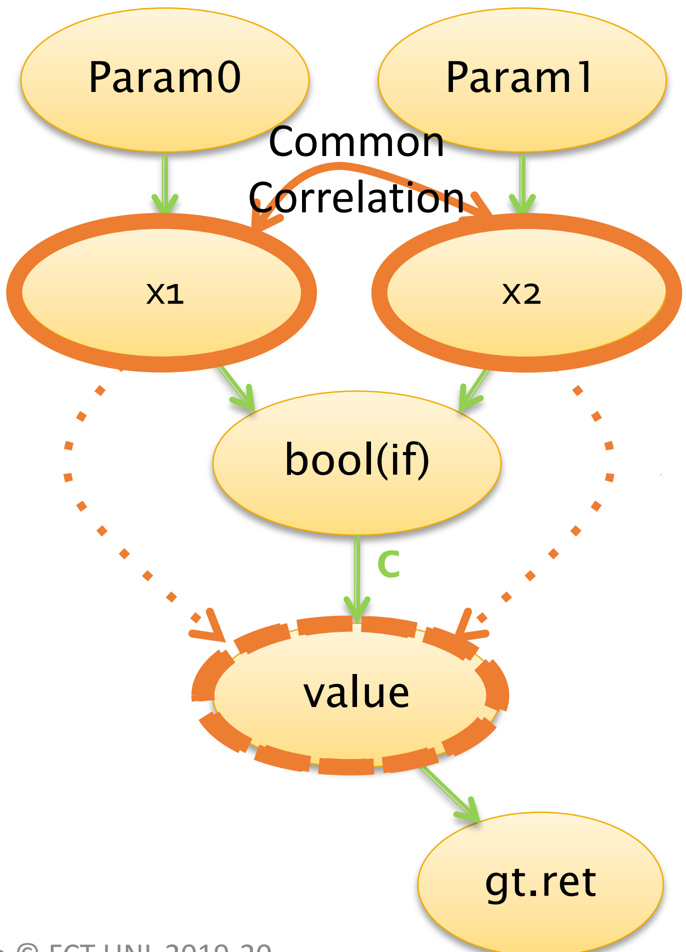
- Direct Correlation (x, y):

  There is a direct correlation between a read variable 'x' and a written variable 'y' if in the dependency graph D there is a path from 'x' to 'y'.



Common Correlation

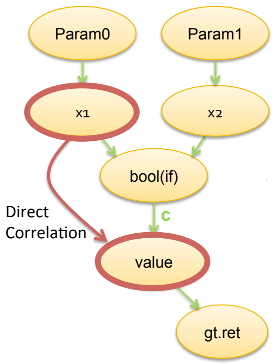
```
   boolean gt(int x1, int x2){
        boolean value;
s1:     if(x1>x2){
               value = true;
        }else{
s3:            value = false;
        }
s4:     return value;
   }
```



Variable's Correlation [Dias12]

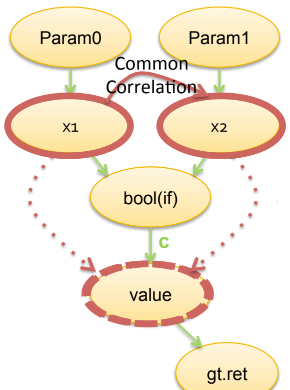
- Direct Correlation (x, y):

  There is a direct correlation between a read variable 'x' and a written variable 'y' if, in a dependency graph D, there is a path from 'x' to 'y'.



- Common Correlation (x, y):

  There is a common correlation between read variables 'x' and 'y' if, in a dependency graph D, there is a written variable 'z' such that: 'z ≠ x' and 'z ≠ y' and there are paths from 'x' to 'z' and from 'y' to 'z'.



High-Level Data Race

Thread 1

```
public synchronized
int setPair(int v1, int v2){
    x = v1;
    y = v2;
}
```

Thread 2

```
public boolean equals(){
    int loc_x = getX();  // Atomic
    int loc_y = getY();  // Atomic
    return loc_x == loc_y;
}
```

High-Level Data Race

Thread 1

```
public synchronized
int setPair(int v1, int v2){
    x = v1;
    y = v2;
}
```

Thread 2

```
public boolean equals(){
  int loc_x = getX()  // Atomic
  int loc_y = getY()  // Atomic
  return loc_x == loc_y;
}
```

Thread 1	Thread 2
V_w (setPair) = {x, y}	V_w (getX) = { }
V_r (setPair) = { }	V_r (getX) = {x}
	V_w (getY) = { }
	V_r (getY) = {y}

High-Level Data Race [Dias12]

Thread 1

```
public synchronized
int setPair(int v1, int v2){
    x = v1;
    y = v2;
}
```

Thread 2

```
public boolean equals(){
  int loc_x = getX(); // Atomic
  int loc_y = getY(); // Atomic
  return loc_x == loc_y;
}
```

Maximal $View_W$ of T1

Thread 1	Thread 2
V_w (setPair) = {x, y}	{ }
V_r (setPair) = { }	(getX) = {x}
	V_w (getY) = { }
	V_r (getY) = {y}

$View_R$ of T2

$View_R$ of T2

Data Race!

{x, y} ∩ {x} = {x}

{x, y} ∩ {y} = {y}

({x} ⊈ {y} ∧ {y} ⊈ {x}) ∧ Common Correlation ({x}, {y})

HLDR Quiz

- T1 runs **V1 = {A, B, C}** and **V2 = {A, B, C, D}**
- T2 runs **V3 = {A, B, E}** and **V4 = {B, C, F}**
- **Is there a HLDR?**

- V2 is maximal in T1 (ignore V1)
- $V2 \cap V3 = \{A, B\}$ $V2 \cap V4 = \{B, C\}$
- $\{A, B\} \subseteq \{B, C\}$ or $\{B, C\} \subseteq \{A, B\}$? No!
- Common-Correlation(A, C)?
  - Yes! High Level Data Race!
  - No! **No** High Level Data Race

Acknowledgments

- Some parts of this presentation was based in publicly available slides and PDFs
  - www.cs.cornell.edu/courses/cs4410/2011su/slides/lecture10.pdf
  - www.microsoft.com/en-us/research/people/madanm/
  - williamstallings.com/OperatingSystems/
  - codex.cs.yale.edu/avi/os-book/OS9/slide-dir/

The END