Karlsruhe Institute of Technology

Quis Custodiet Ipsos Custodes
The Java memory model

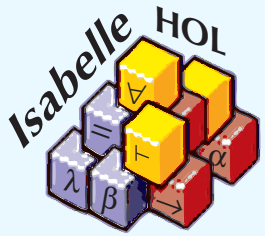
funded by DFG Ni491/11, Sn11/10

Andreas Lochbihler

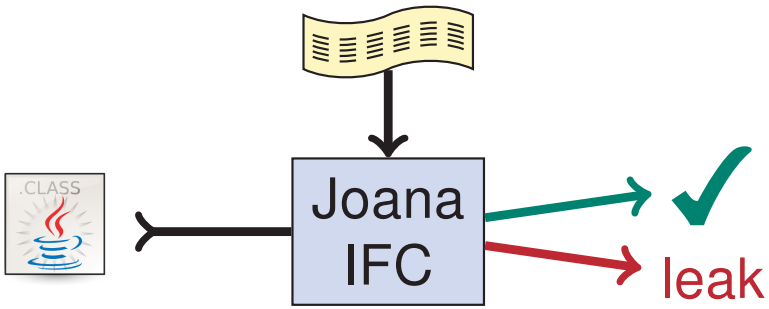
PROGRAMMING PARADIGMS GROUP

```
theorem drf:
  assumes sync: "correctly_synchronized P E"
  and legal: "legal_execution P E (E, ws)"
  shows "sequentially_consistent P (E, ws)"
using legal_wf_execD[OF legal] legal_ED[OF legal] sync
proof(rule drf_lemma)
  fix r
  assume "r ∈ read_actions E"

  from legal obtain J where E: "E ∈ E"
    and wf_exec: "P ⊢ (E, ws) √"
    and J: "P ⊢ (E, ws) justified_by J"
```

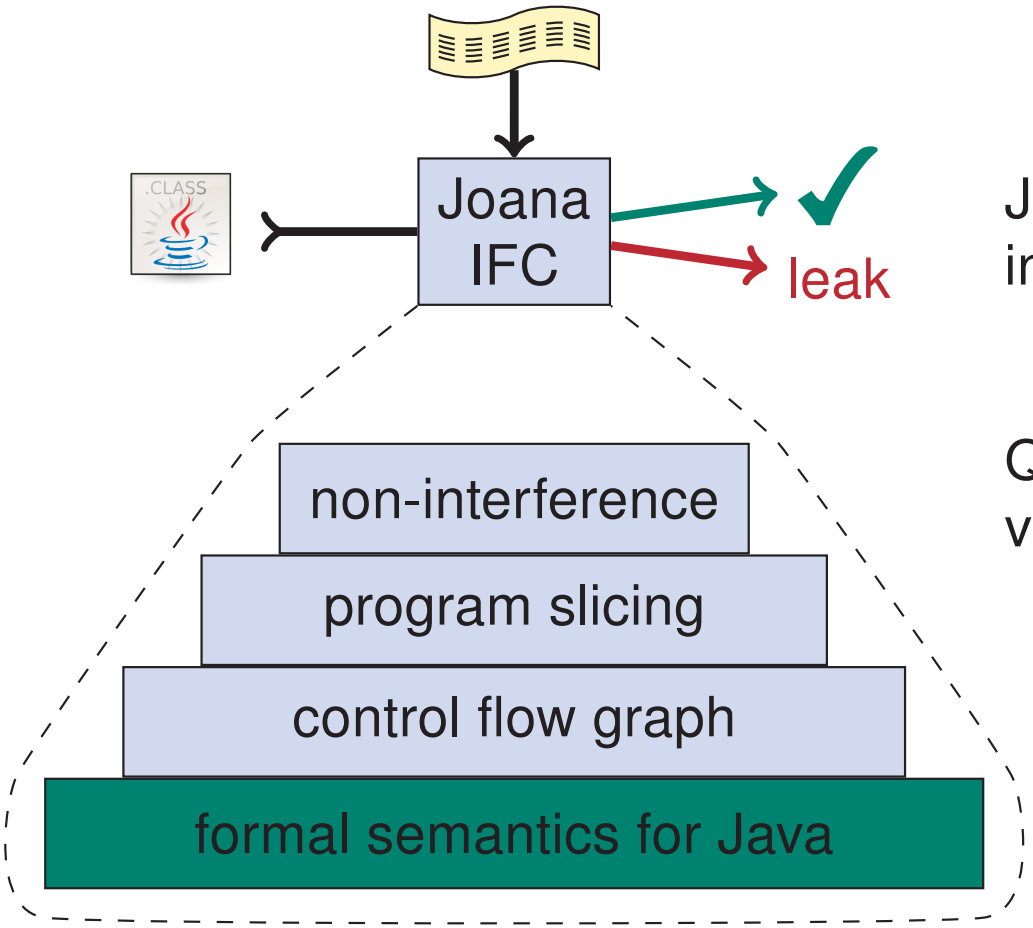


Quis custodiet ipsos custodes?



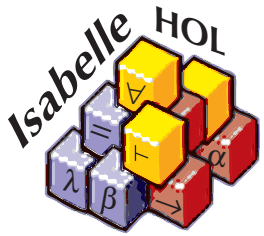
Joana:
information flow control for Java

Quis custodiet ipsos custodes?

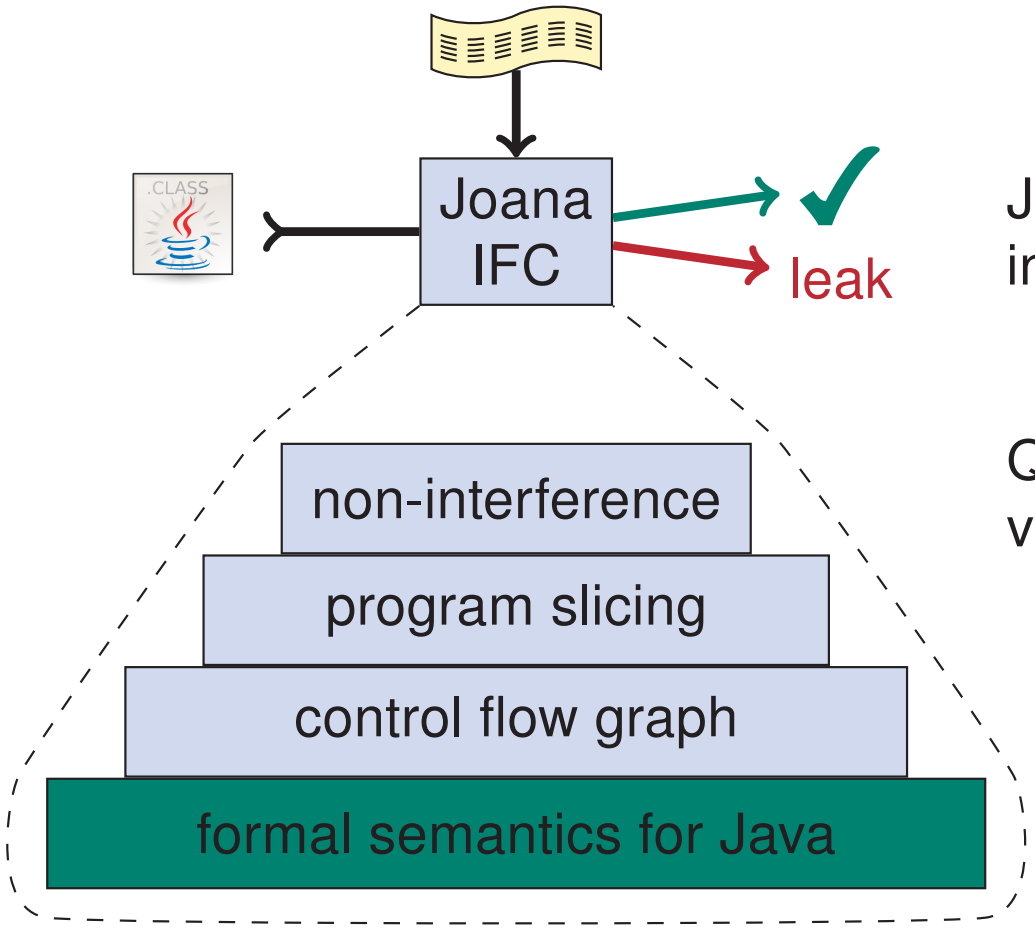


Joana:
information flow control for Java

Quis custodiet:
verify IFC algorithm

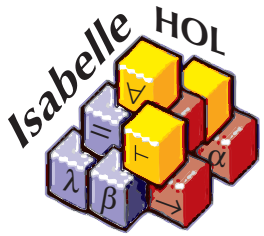


Quis custodiet ipsos custodes?



Joana:
information flow control for Java

Quis custodiet:
verify IFC algorithm



How can we include Java concurrency?

The Java memory model (JMM): beyond interleaving semantics

initially: `x = y = 0;`

`x = 1;`	`y = 2;`
`j = y;`	`i = x;`

The Java memory model (JMM): beyond interleaving semantics

Karlsruhe Institute of Technology

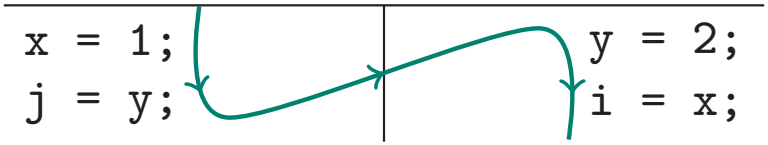
initially: `x = y = 0;`

`x = 1;`	`y = 2;`
`j = y;`	`i = x;`

interleaving semantics

	`j == 0`	`j == 2`
`i == 0`		
`i == 1`		

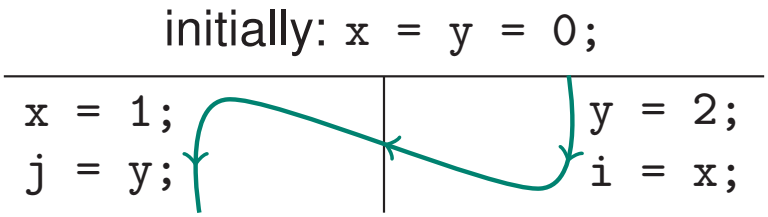
The Java memory model (JMM): beyond interleaving semantics



interleaving semantics

	j == 0	j == 2
i == 0		
i == 1	√	

The Java memory model (JMM): beyond interleaving semantics

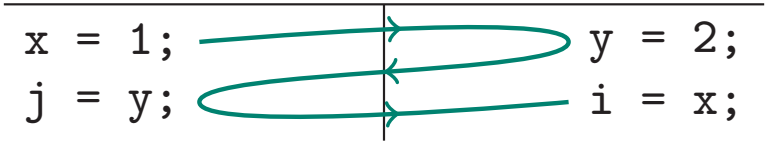


interleaving semantics

	j == 0	j == 2
i == 0		√
i == 1	√	

The Java memory model (JMM): beyond interleaving semantics

initially: `x = y = 0;`

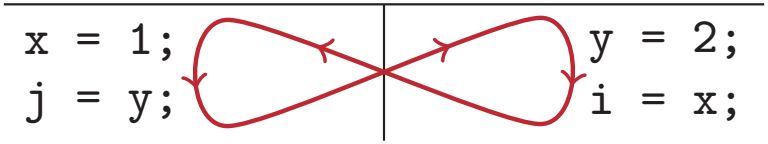


interleaving semantics

	`j == 0`	`j == 2`
`i == 0`		√
`i == 1`	√	√

The Java memory model (JMM): beyond interleaving semantics

initially: `x = y = 0;`



interleaving semantics

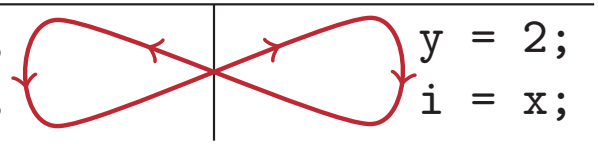
	`j == 0`	`j == 2`
`i == 0`	X	√
`i == 1`	√	√

The Java memory model (JMM): beyond interleaving semantics

Karlsruhe Institute of Technology

initially: `x = y = 0;`

`x = 1;`	`y = 2;`
`j = y;`	`i = x;`

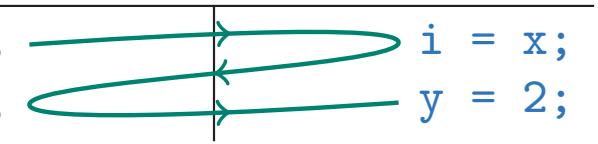


interleaving semantics

	`j == 0`	`j == 2`
`i == 0`	X	√
`i == 1`	√	√

compiler and hardware reorder statements

`j = y;`	`i = x;`
`x = 1;`	`y = 2;`



	`j == 0`	`j == 2`
`i == 0`	√	
`i == 1`		

The Java memory model (JMM): beyond interleaving semantics

Karlsruhe Institute of Technology

initially: `x = y = 0;`

`x = 1;`	`y = 2;`
`j = y;`	`i = x;`

compiler and hardware reorder statements

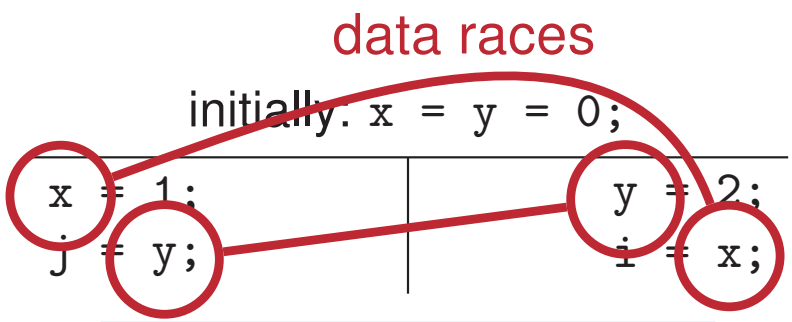
`j = y;`	`i = x;`
`x = 1;`	`y = 2;`

Java memory model

	`j == 0`	`j == 2`
`i == 0`	√	√
`i == 1`	√	√

	`j == 0`	`j == 2`
`i == 0`	√	
`i == 1`		

The Java memory model (JMM): beyond interleaving semantics



compiler and hardware reorder statements

`j = y;`	`i = x;`
`x = 1;`	`y = 2;`

Java memory model

	`j == 0`	`j == 2`
`i == 0`	√	√
`i == 1`	√	√

	`j == 0`	`j == 2`
`i == 0`	√	
`i == 1`		

Problems with data races under the JMM

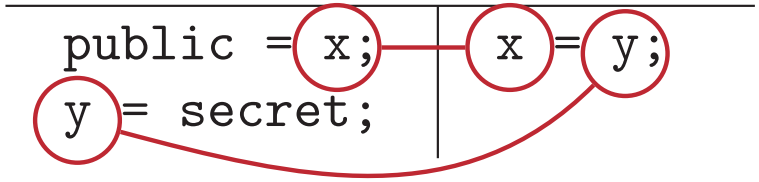
1. Data races allow time travel.

`public = x;` `y = secret;`	`x = y;`

- `r1` may contain `input()`.
- Time-sensitive analyses do not cover that.

Problems with data races under the JMM

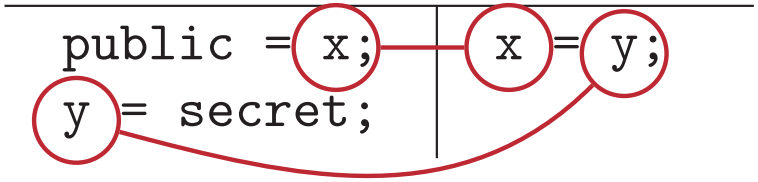
1. Data races allow time travel.



- `r1` may contain `input()`.
- Time-sensitive analyses do not cover that.

Problems with data races under the JMM

1. Data races allow time travel.



- `r1` may contain `input()`.
- Time-sensitive analyses do not cover that.

2. Values appear out of thin air.

initially: `x = y = null; b = false;`

`r1 = y;`
`x = r1;`
	`r2 = null;`
	`r3 = x;`
	`if (b) r2 = new A();`
	`else r3 = new A();`
	`y = r3;`
		`b = true;`

- `r2` and `r3` may alias,
- but typical points-to analyses compute: They never alias.

Results

Unified model of Java and the JMM

First formal link between Java and the Java memory model

Proofs about the JMM

DRF guarantee:

- Programs without data races *behave* as if executed under interleaving semantics
- Most program analyses assume interleaving semantics

$\Rightarrow$ Sound for DRF programs

Type safety:

- Java's type safety extends to the JMM, even if there are data races.
- Crucial for CFG construction (no undefined behaviour)

What is a data race?

Karlsruhe Institute of Technology

Data race:

- In an interleaved (sequentially consistent) executions,
- two accesses (at least one read) to the same non-volatile location
- that are unrelated in $\leq_{hb}$

What is a data race?

Data race:

- In an interleaved (sequentially consistent) executions,
- two accesses (at least one read) to the same non-volatile location
- that are unrelated in $\leq_{hb}$

Synchronisation via by spawning threads:

initially: `x = new Thread(); y = 0;`

`y = 1;` `x.start();`	`try { x.start();` `} catch (IllegalThreadStateException _) { r = y; }`

What is a data race?

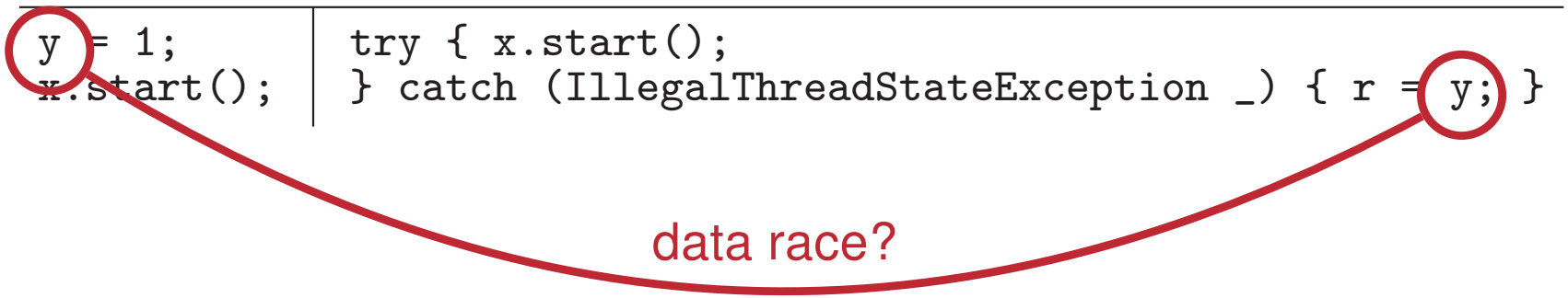
Data race:

- In an interleaved (sequentially consistent) executions,
- two accesses (at least one read) to the same non-volatile location
- that are unrelated in $\leq_{hb}$

Synchronisation via by spawning threads:

initially: `x = new Thread(); y = 0;`

`y = 1;` `x.start();`	`try { x.start();` `} catch (IllegalThreadStateException _) { r = y; }`

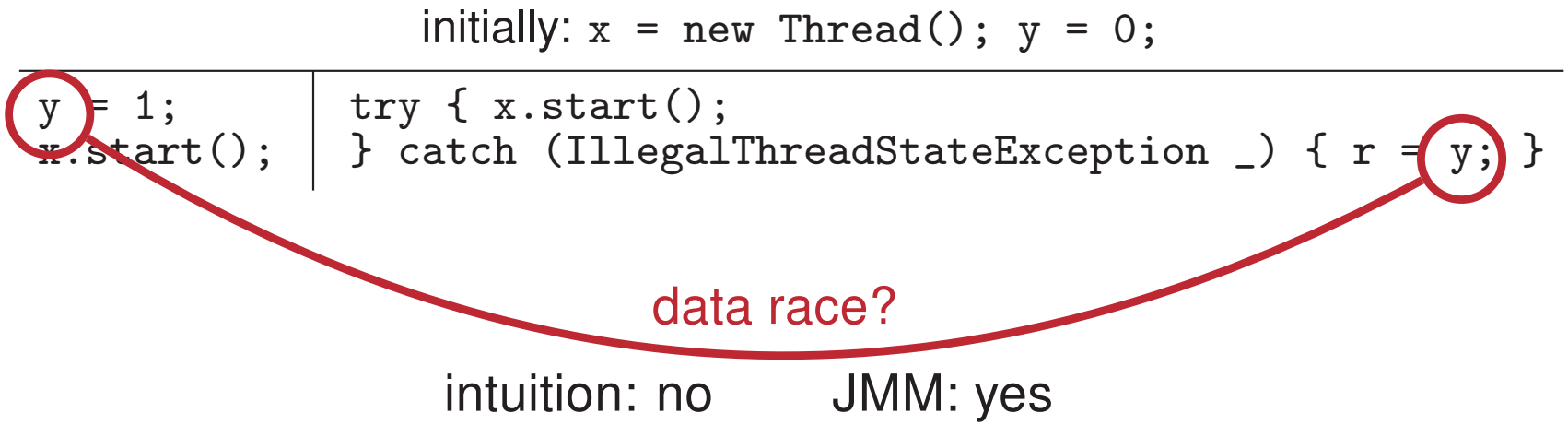


What is a data race?

Data race:

- In an interleaved (sequentially consistent) executions,
- two accesses (at least one read) to the same non-volatile location
- that are unrelated in $\leq_{hb}$

Synchronisation via by spawning threads:



What is a data race?

Data race:

- In an interleaved (sequentially consistent) executions,
- two accesses (at least one read) to the same non-volatile location
- that are unrelated in $\leq_{hb}$

Synchronisation via by spawning threads: