Goblint Against Auto Racing

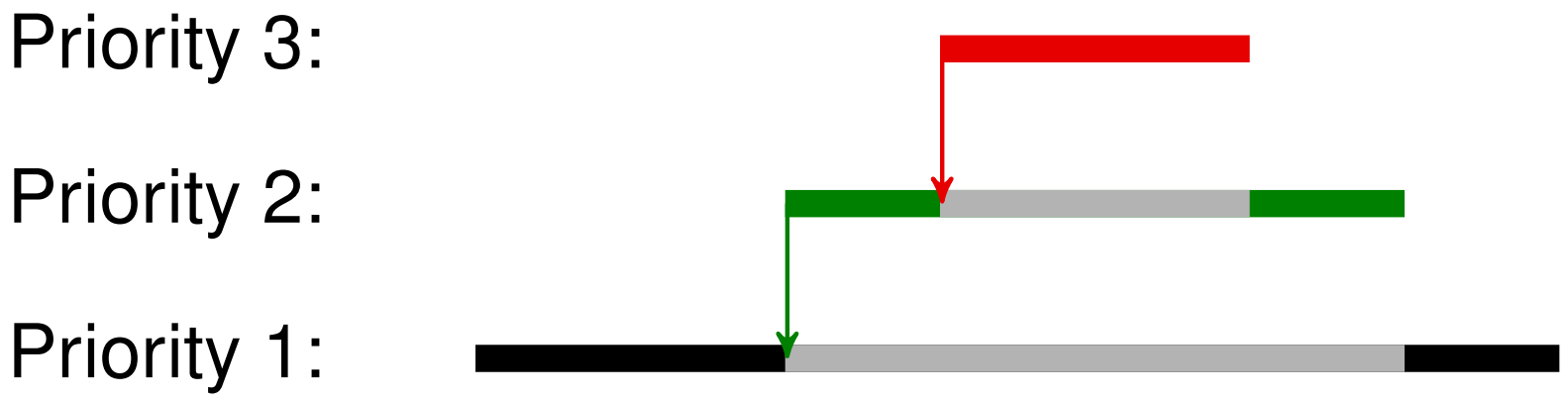
Detecting Concurrency Flaws in Interrupt-Driven Software

Vesal Vojdani

(based on Schwarz, Seidl, Vojdani, Lammich, and Müller-Olm. “Static Analysis of Interrupt-Driven Programs synchronized via the priority ceiling protocol”. POPL’11)

Theory Days at Nelijärve
February 6, 2011

Interrupt-Driven Concurrency



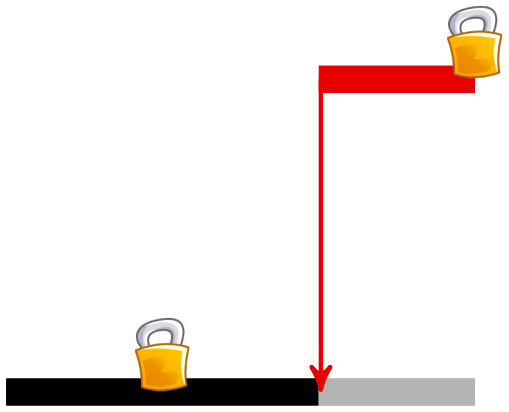
Ima let you finish, but . . .

Synchronization: Mutexes I

Priority 3:

Priority 2:

Priority 1:

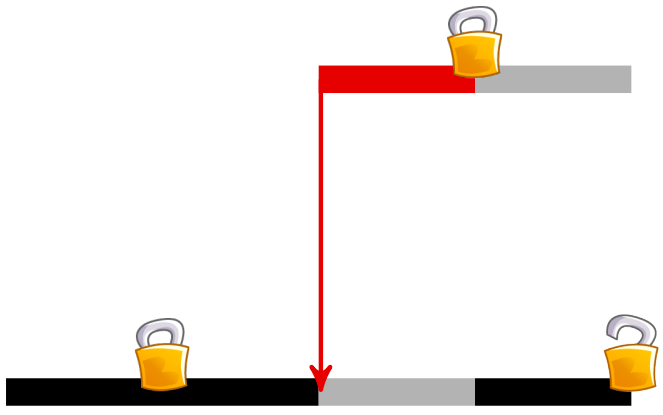


Synchronization: Mutexes I

Priority 3:

Priority 2:

Priority 1:

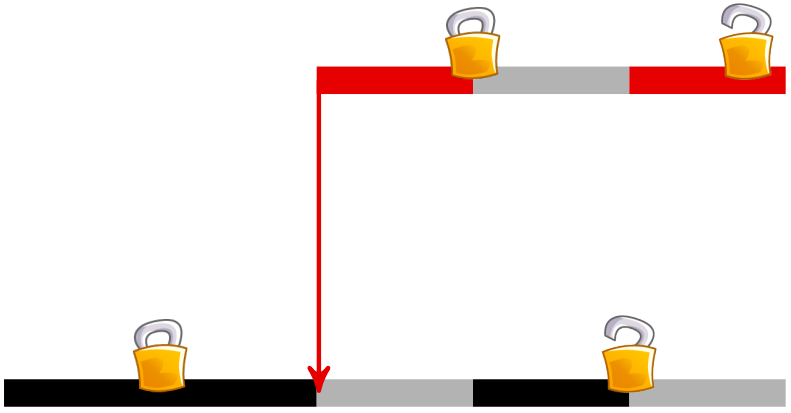


Synchronization: Mutexes I

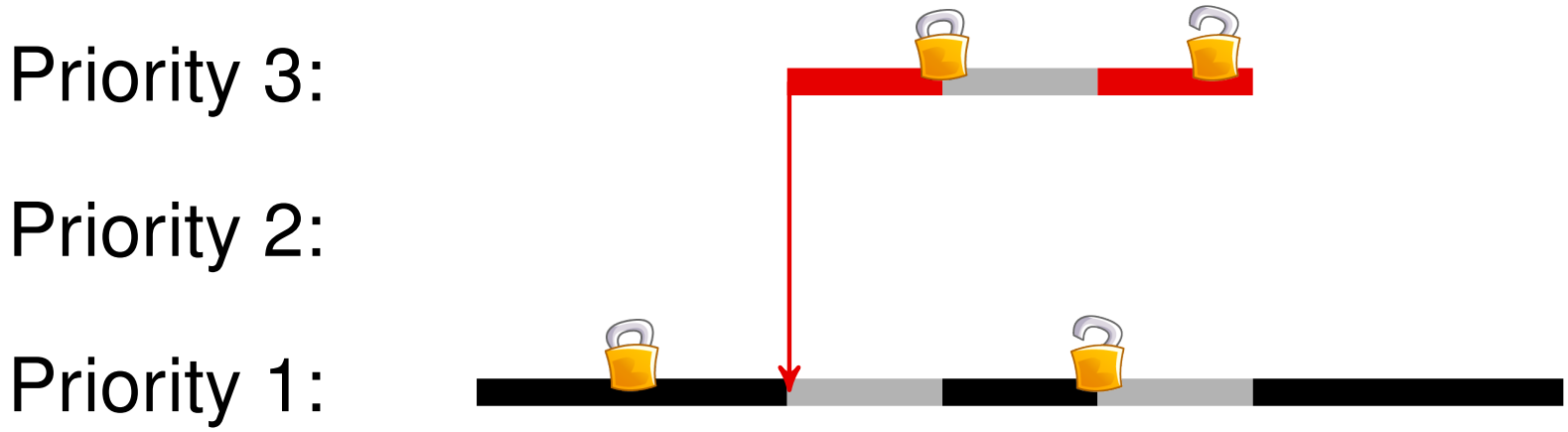
Priority 3:

Priority 2:

Priority 1:



Synchronization: Mutexes I



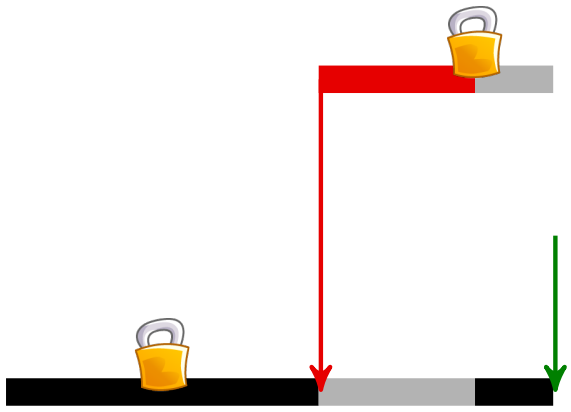
blocking $\leqslant$ critical section

Synchronization: Mutexes II

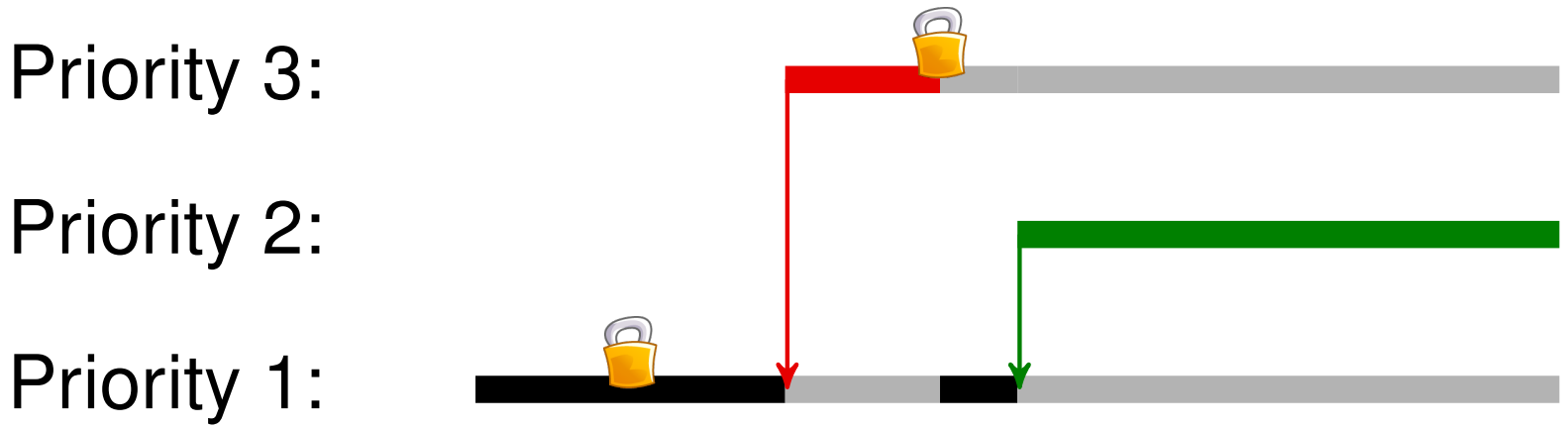
Priority 3:

Priority 2:

Priority 1:



Synchronization: Mutexes II



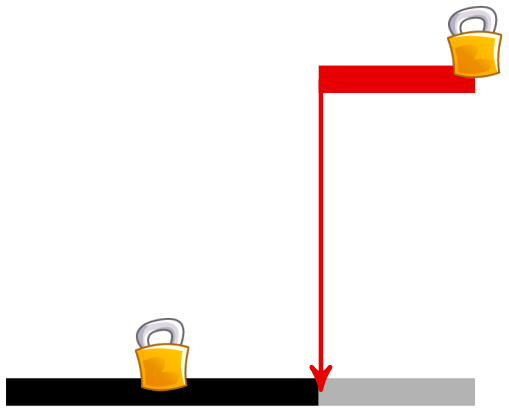
Unbounded priority inversion!

Solution: Priority Inheritance

Priority 3:

Priority 2:

Priority 1:

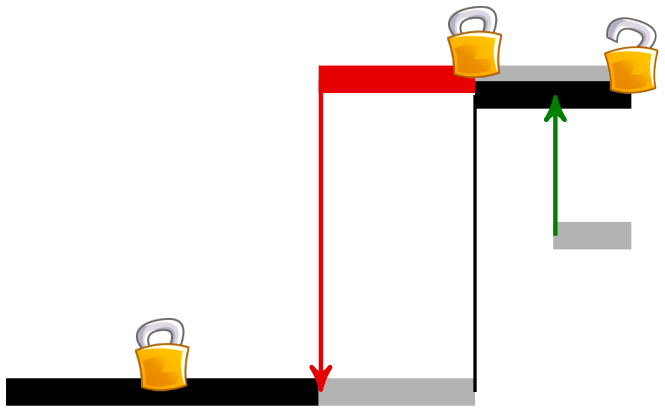


Solution: Priority Inheritance

Priority 3:

Priority 2:

Priority 1:

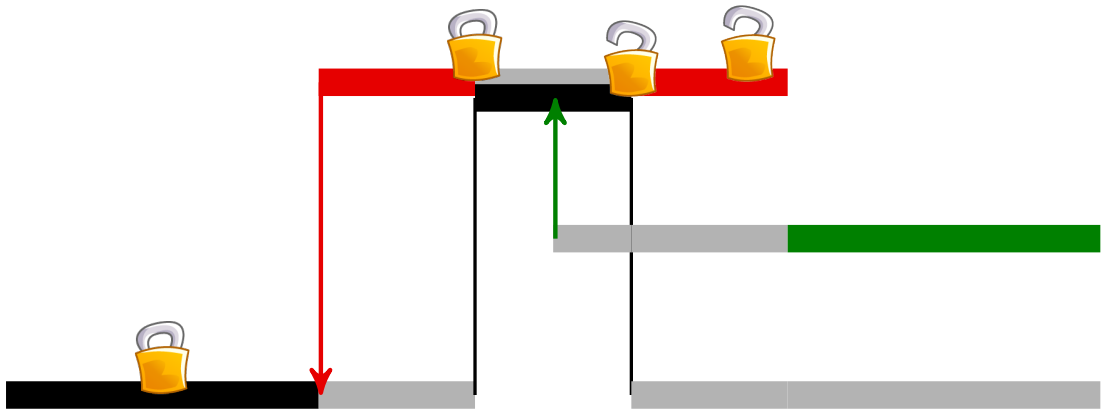


Solution: Priority Inheritance

Priority 3:

Priority 2:

Priority 1:



Again, blocking $\leqslant$ critical section

Simplification: Immediate Inheritance

Priority 3:

Priority 2:

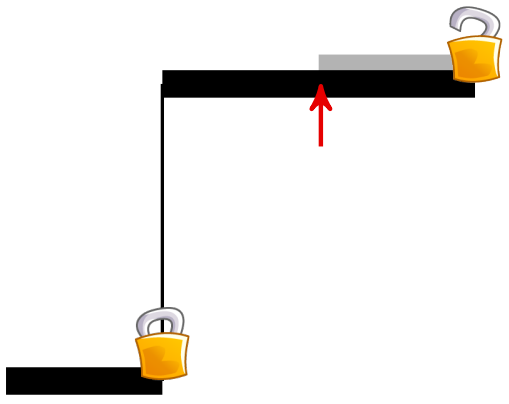
Priority 1: 

Simplification: Immediate Inheritance

Priority 3:

Priority 2:

Priority 1:

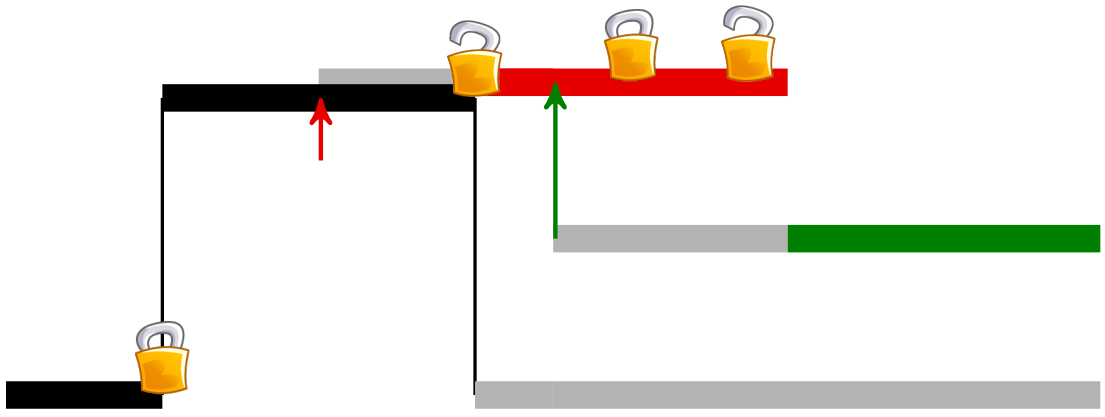


Simplification: Immediate Inheritance

Priority 3:

Priority 2:

Priority 1:



More aggressive in raising priorities;
still, same worst-case bounds on blocking.

Immediate Ceiling Priority Protocol

- $P(t)$: priority of task t.
- T_r: set of tasks acquiring resource r.
- Ceiling priority of r:

$$P(r) = \bigvee \{P(t) \mid t \in T_r\}$$

- Task t with resources R runs at priority

$$\bigvee \{P(r) \mid r \in R\} \vee P(t)$$

Key Benefits

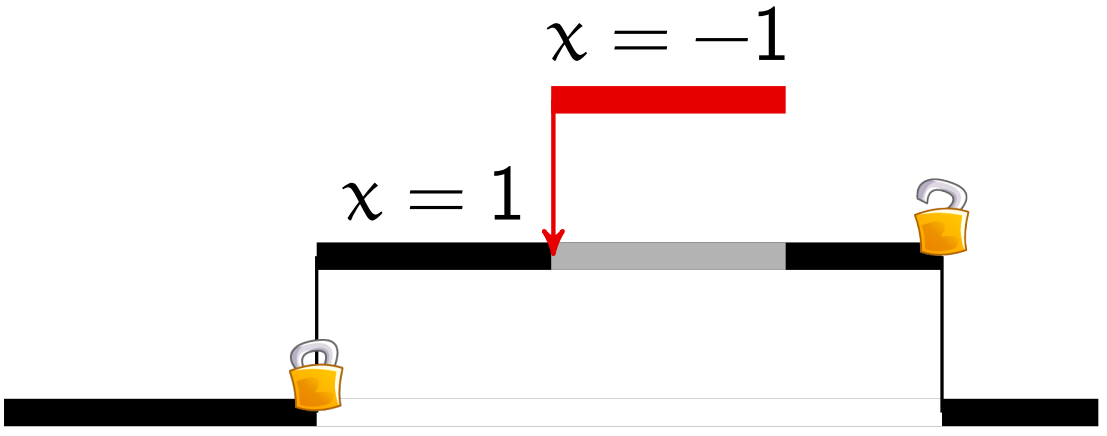
- No unbounded priority inversion.
- No deadlocks!
  - Only strictly higher-priority tasks can preempt.
  - You never acquire resources held by others.
  - If t acquires a resource r held by t', $\mathrm{P}(\mathrm{r}) \geqslant \mathrm{P}(\mathrm{t})$. (So t could not have pre-empted t'.)
- Hence, adopted by the Autosar/OSEK standard.

Some Problems: Data Race

Priority 3:

Priority 2:

Priority 1:



Value of x may be lost or messed up.

Some Problems: Transactionality

Priority 3:

Priority 2:

Priority 1:

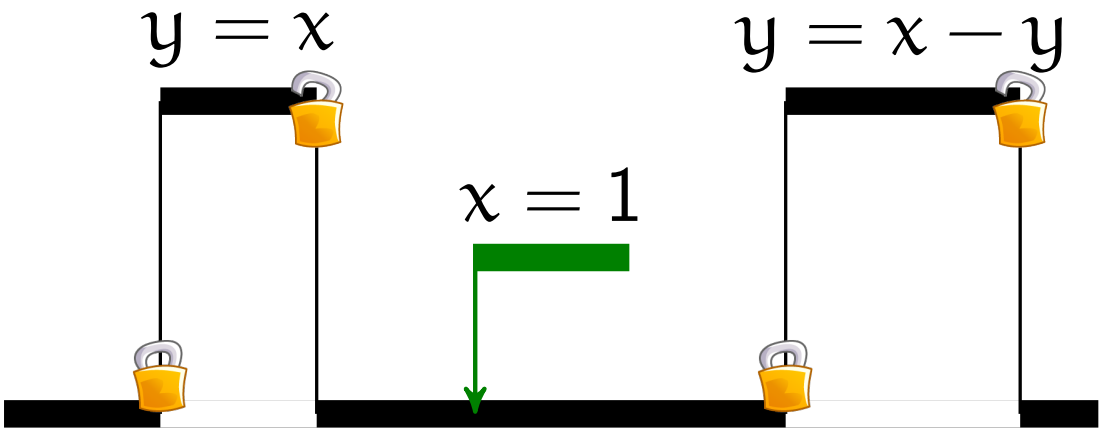
Each access is protected

Some Problems: Transactionality

Priority 3:

Priority 2:

Priority 1:



Each access is protected
but computation is not transactional.

Transactionality analysis

- Resource analysis
  - Determine sets of held resources for every program point.
  - Determine ceiling priority of every program point.
- Transactionality analysis
  - Uses information from the resource analysis.
  - Determine minimum priorities of critical sections.

Resource analysis

- Functions require context
  - Depend on resource set at the call
  - Return possibly changed resource set
- Preemptions do not influence resource sets
  - Tasks start with empty resource sets
  - Tasks terminate with empty resource sets

$\Longrightarrow$ Ordinary interprocedural analysis

Flow-independent Resources

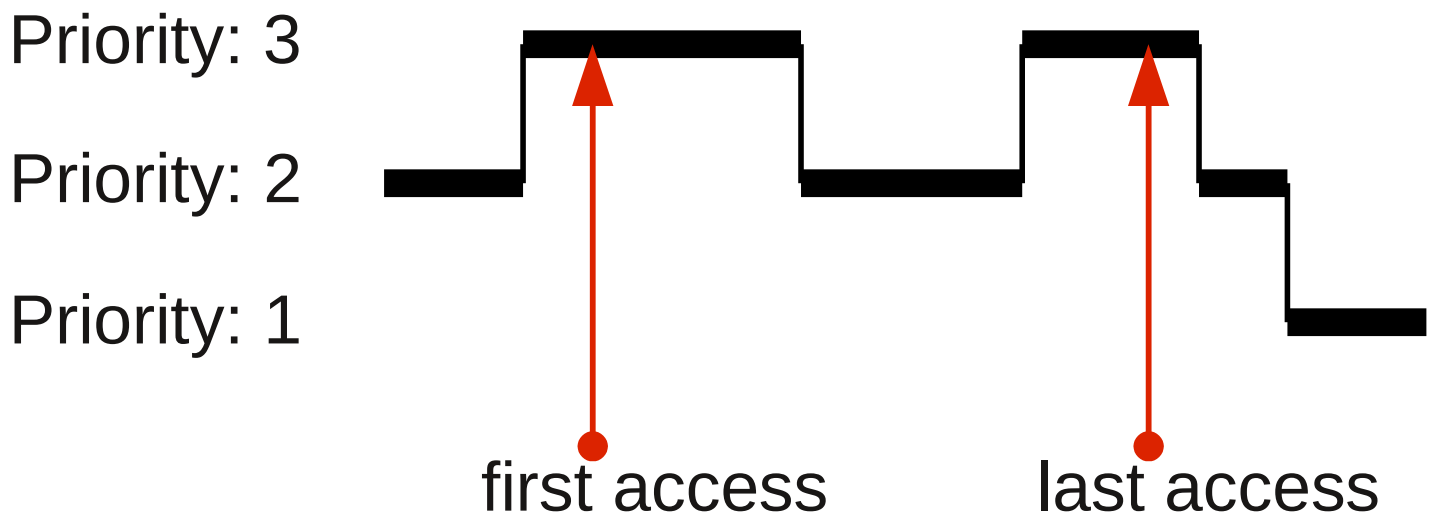
- Assume: Resource sets depend only on calling context and the current program point.
- Does not differ along different paths.

$\Longrightarrow$ Gen/kill resource analysis
efficient procedure summaries.

(Otherwise, the analysis is sound, but not complete.)

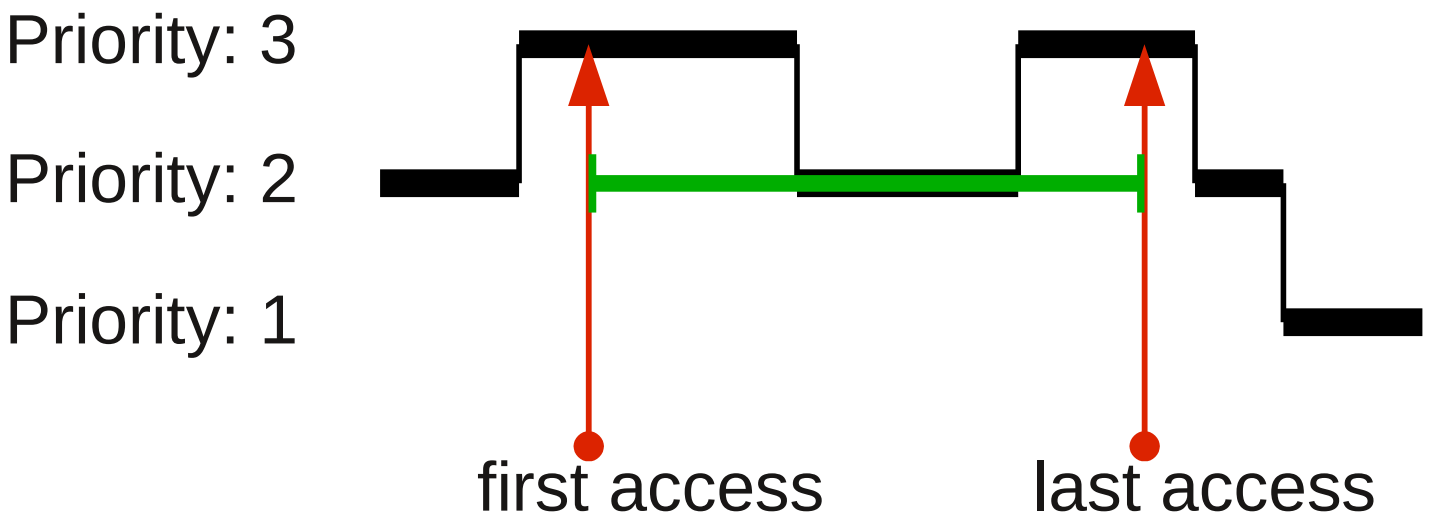
Analyzing transactionality

Four accumulated priorities



Analyzing transactionality

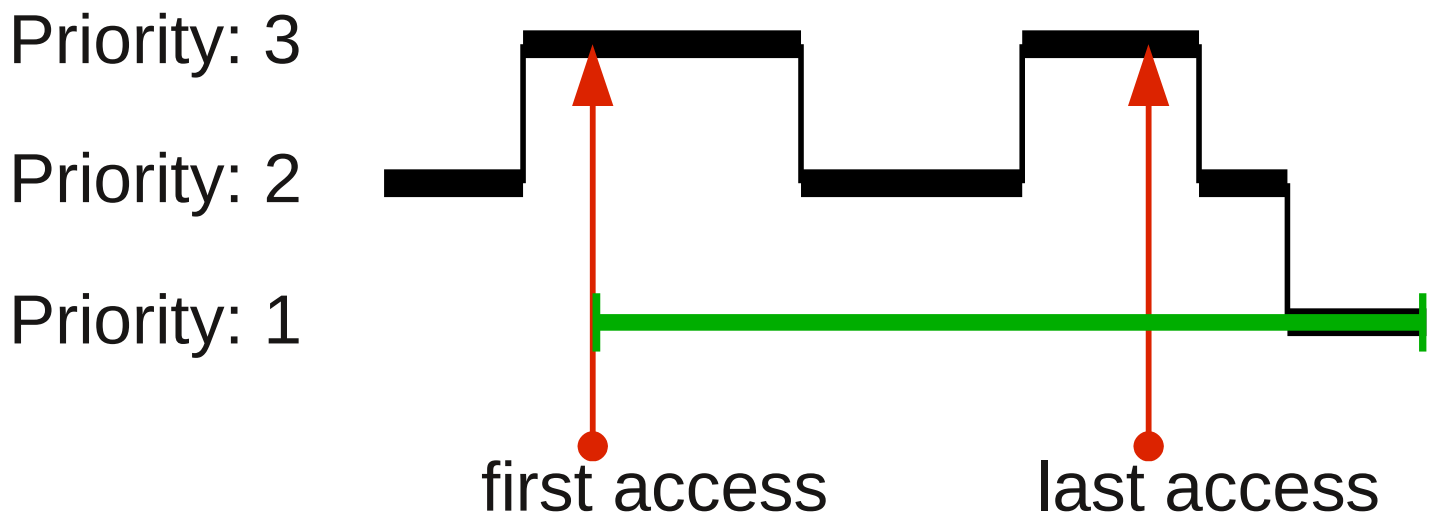
Four accumulated priorities



- Minimal priority in the critical section: 2

Analyzing transactionality

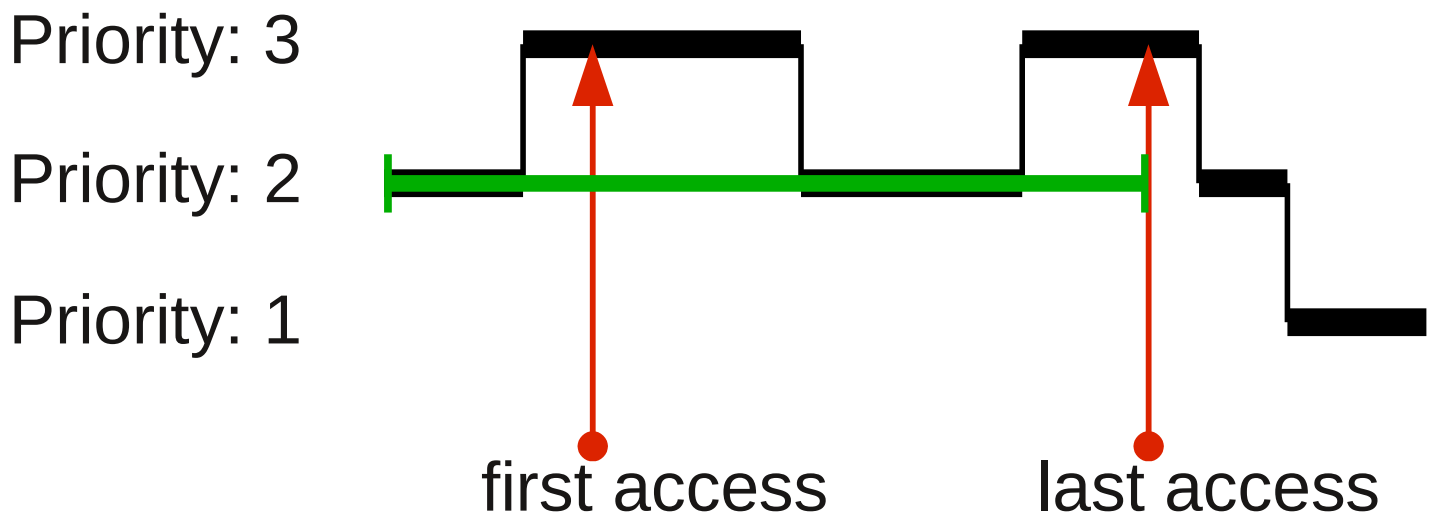
Four accumulated priorities



- Minimal priority after the first access: 1

Analyzing transactionality

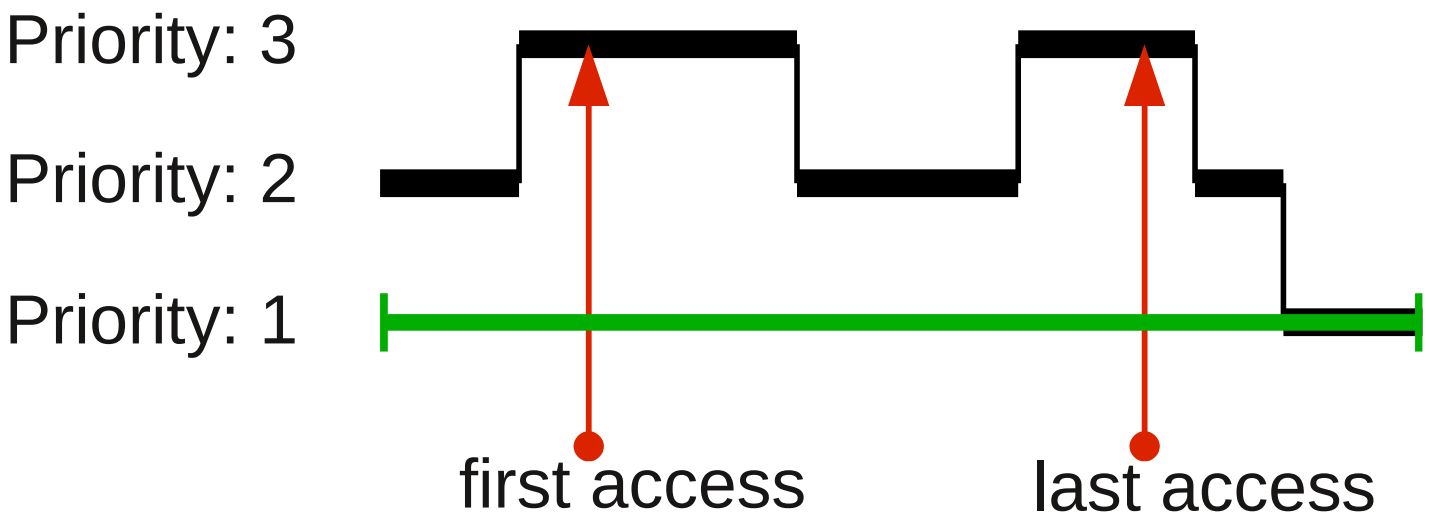
Four accumulated priorities



- Minimal priority before the last access: 2

Analyzing transactionality

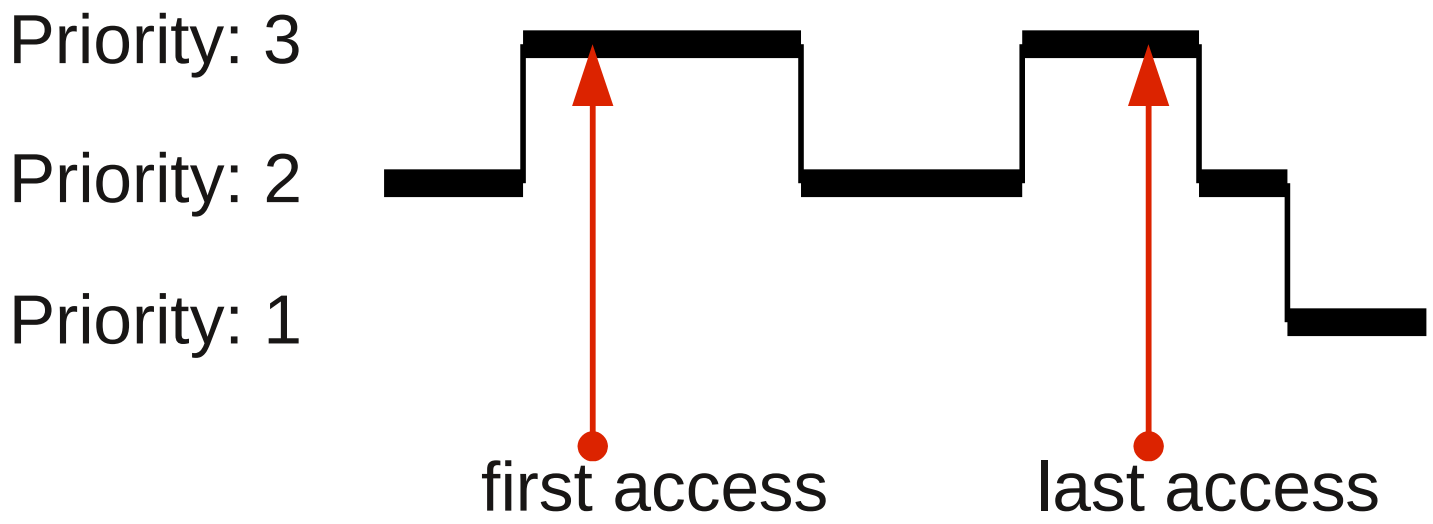
Four accumulated priorities



- Minimal priority in the whole function: 1

Analyzing transactionality

Four accumulated priorities



- (2,1,2,1)

Composition

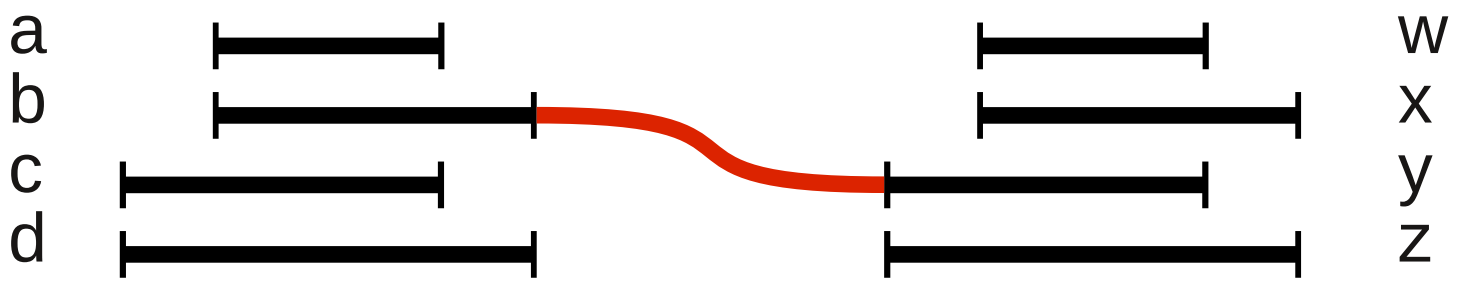
(critical, after first, before last, whole)

$$(w, x, y, z) \circ (a, b, c, d) = (y \wedge b, z \wedge b, y \wedge d, z \wedge d)$$

Composition

(critical, after first, before last, whole)

$$(w, x, y, z) \circ (a, b, c, d) = (y \wedge b, z \wedge b, y \wedge d, z \wedge d)$$



Composition

(critical, after first, before last, whole)

$$(w, x, y, z) \circ (a, b, c, d) = (y \wedge b, z \wedge b, y \wedge d, z \wedge d)$$

Composition

(critical, after first, before last, whole)

$$(w, x, y, z) \circ (a, b, c, d) = (y \wedge b, z \wedge b, y \wedge d, z \wedge d)$$

Composition

(critical, after first, before last, whole)

$$(w, x, y, z) \circ (a, b, c, d) = (y \wedge b, z \wedge b, y \wedge d, z \wedge d)$$

Effect of edges

$$[\![\text{edge}]\!]^\sharp(\mathrm{p}) = \begin{cases} (\infty, \mathrm{p}, \mathrm{p}, \mathrm{p}) & \text{accesses in edge} \\ (\infty, \infty, \infty, \mathrm{p}) & \text{no accesses in edge} \end{cases}$$

p denotes the priority at the edge with resources and calling context taken into account

Transactionality analysis

Transactionality summary:

$$[\![f]\!]^\sharp(R) \mapsto (\text{critical, first, last, whole})$$

Theorem (complexity)

Computing transactionality summaries is

- *linear in the size of the program*
- *quadratic in the number of priority levels*
- *exponential in the number of resources*

Determining transactionality

Transactionality summary:

$$[\![\mathrm{f}]\!]^\sharp(\mathrm{R}) \mapsto (\text{critical, first, last, whole})$$

Set of variables accessed by f: V

Theorem (sound & complete)

Function f *is transactional when called with resource set* R *if and only if*

$$\textit{critical} \geqslant \begin{array}{l}\textit{maximal priority of interrupts} \\ \textit{accessing a variable in } \mathrm{V}\end{array}$$

Implementation

Resource, data race and transactionality analysis have been implemented in the Goblint analyzer

- Analyzes concurrent C.
- Context sensitive.
- Based on side-effecting fixpoint solving
- Performance:

Program	Size	Time
sunroof	40966 lines	15 s

Real benchmarks with resources

Program	Size	Time
`biped_robot`	151 lines	0,02 s
`pe_test`	97 lines	0,06 s
`res_test`	74 lines	0,03 s
`tt_test`	101 lines	0,07 s
`usb_test`	140 lines	0,04 s
`pingpong`	53 lines	0,03 s
`counter`	58 lines	0,02 s

Conclusion

- Techniques for taking into account the data flow induced by the priority-driven scheduler.
- Detecting concurrency bugs, and. . .
- Adapting value analyses to OSEK programs. Example: Precise Linear Equality Analysis.
- Further Challenges: lock-free idioms . . .

Thank you for your attention!