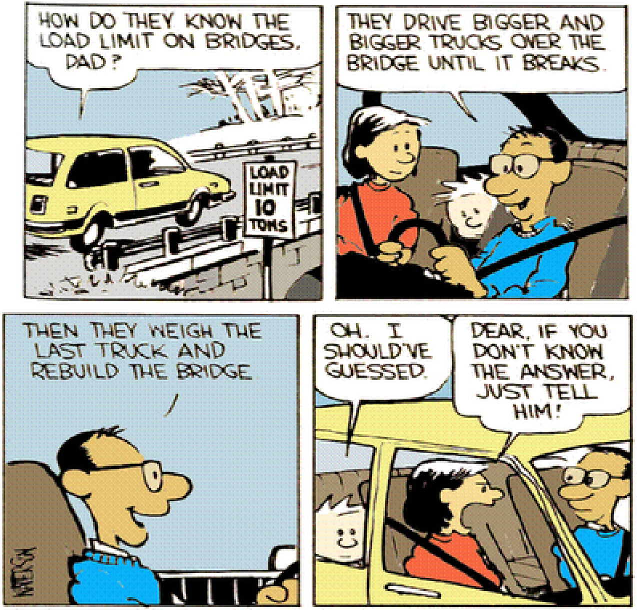
Access Control and Privacy Policies (9)

Email: christian.urban at kcl.ac.uk
Office: S1.27 (1st floor Strand Building)
Slides: KEATS (also homework is there)


HOW DO THEY KNOW THE LOAD LIMIT ON BRIDGES, DAD?
LOAD LIMIT 10 TONS
THEY DRIVE BIGGER AND BIGGER TRUCKS OVER THE BRIDGE UNTIL IT BREAKS.
THEN THEY WEIGH THE LAST TRUCK AND REBUILD THE BRIDGE.
OH. I SHOULD'VE GUESSED.
DEAR, IF YOU DON'T KNOW THE ANSWER, JUST TELL HIM!
WATTERSON

Old-Fashioned Eng. vs. CS

bridges:

engineers can "look" at a bridge and have a pretty good intuition about whether it will hold up or not (redundancy; predictive theory)

code:

programmers have very little intuition about their code; often it is too expensive to have redundancy; not "continuous"

Dijkstra on Testing

> "Program testing can be a very effective way to show the presence of bugs, but it is hopelessly inadequate for showing their absence."

unfortunately attackers exploit bugs (Satan's computer vs Murphy's)

Dijkstra: shortest path algorithm, dining philosophers problem, semaphores

Proving Programs to be Correct

Theorem: There are infinitely many prime numbers.

Proof ...

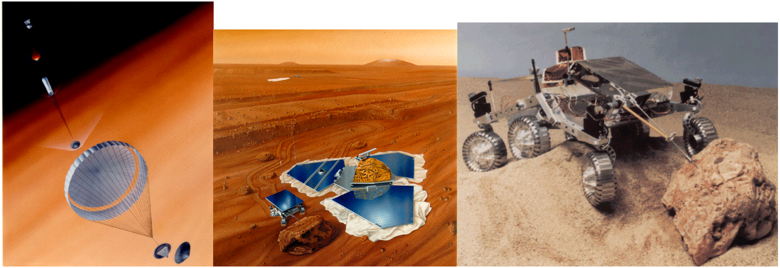
similarly

Theorem: The program is doing what it is sup+ed to be doing.

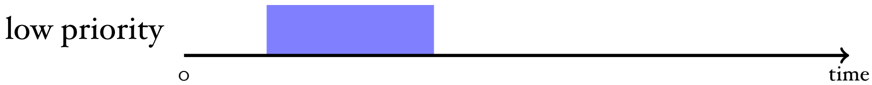
Long, long proof ...

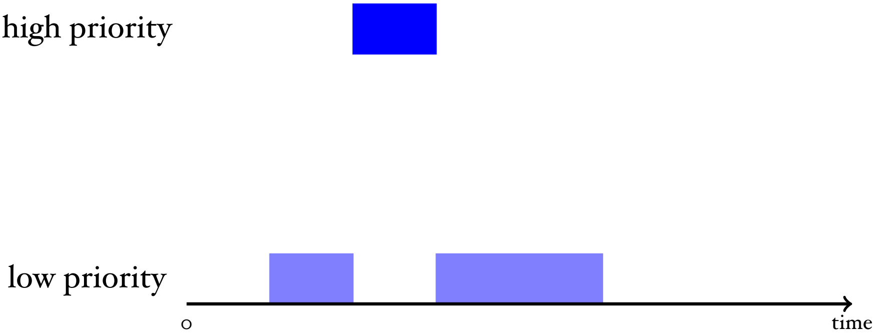
This can be a gigantic proof. The only hope is to have help from the computer. 'Program' is here to be understood to be quite general (protocol, OS,...).

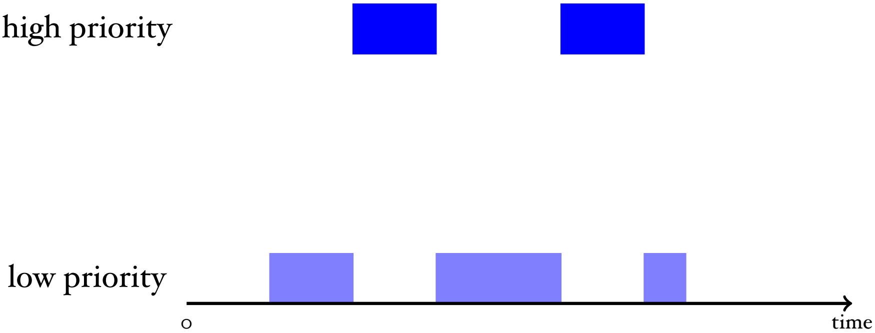
Mars Pathfinder Mission 1997



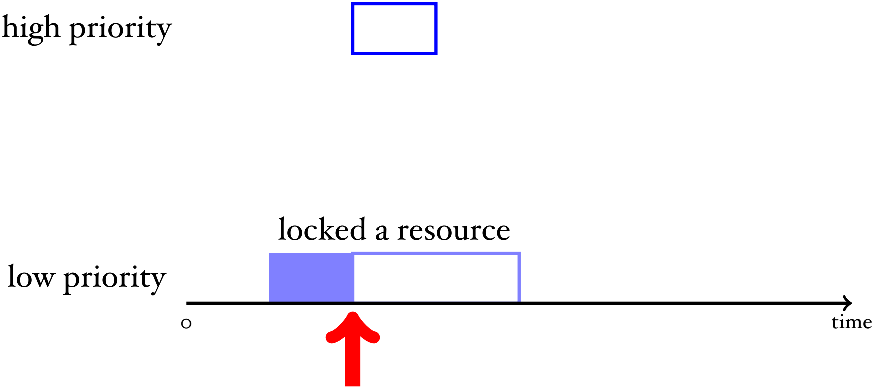
- despite NASA's famous testing procedures, the lander crashed frequently on Mars
- a scheduling algorithm was not used in the OS


low priority
0
time

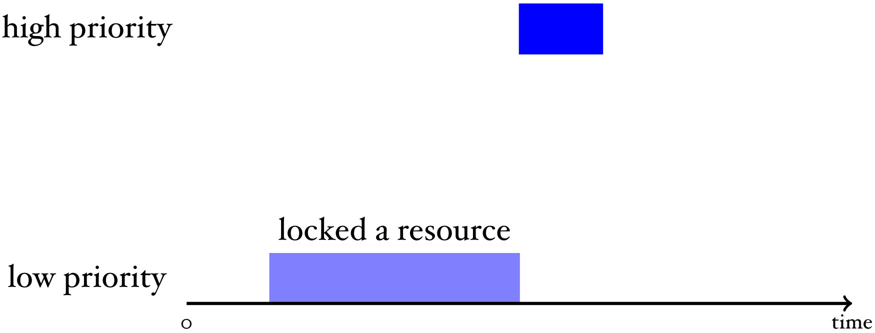

high priority
low priority
0
time



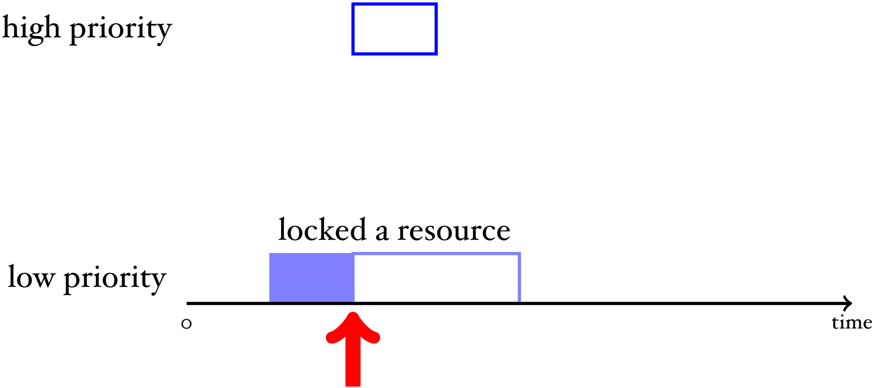
Scheduling: You want to avoid that a high priority process is staved indefinitely.



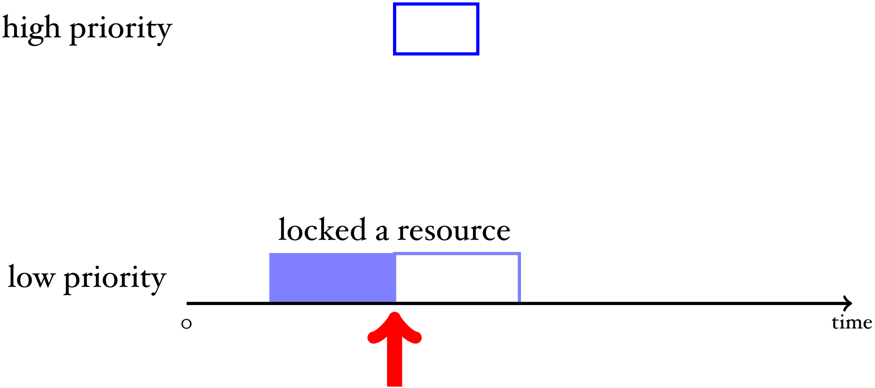
Scheduling: You want to avoid that a high priority process is staved indefinitely.



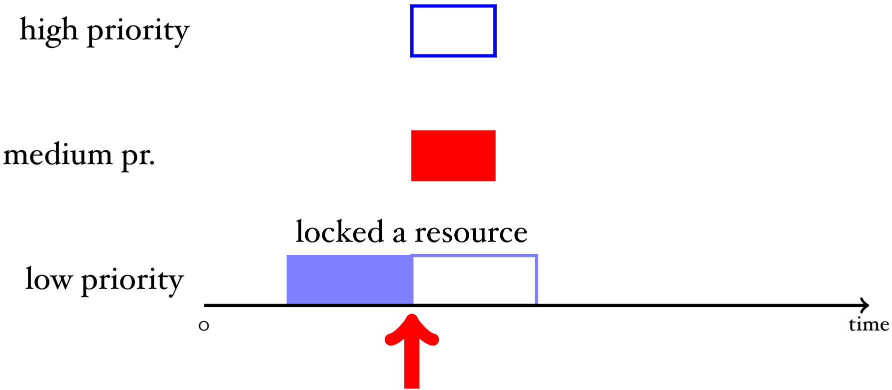
Scheduling: You want to avoid that a high priority process is staved indefinitely.



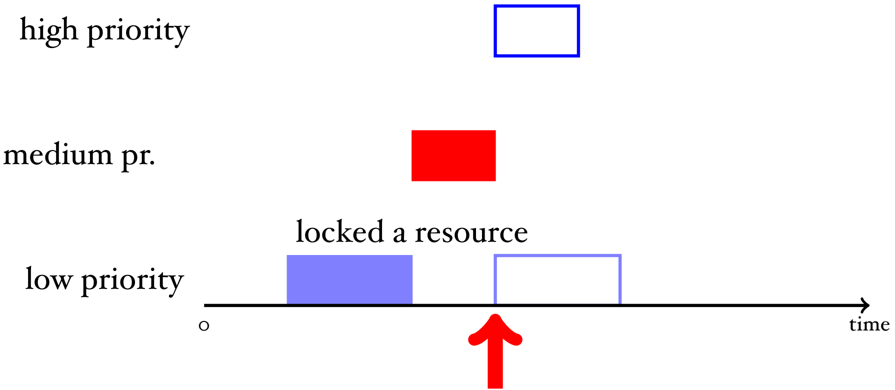
Scheduling: You want to avoid that a high priority process is staved indefinitely.



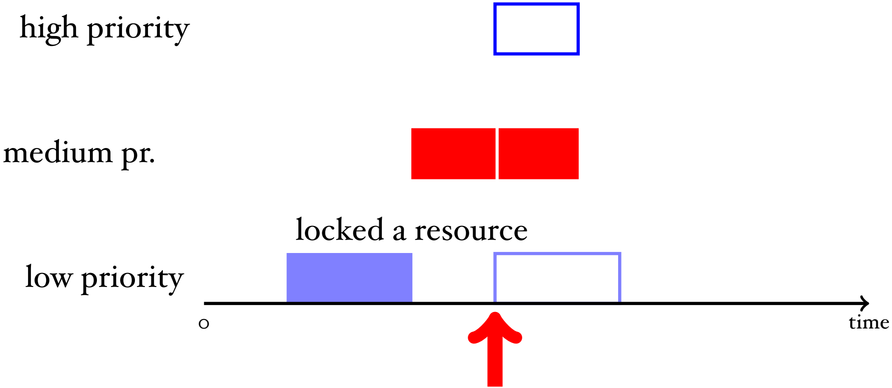
Scheduling: You want to avoid that a high priority process is staved indefinitely.



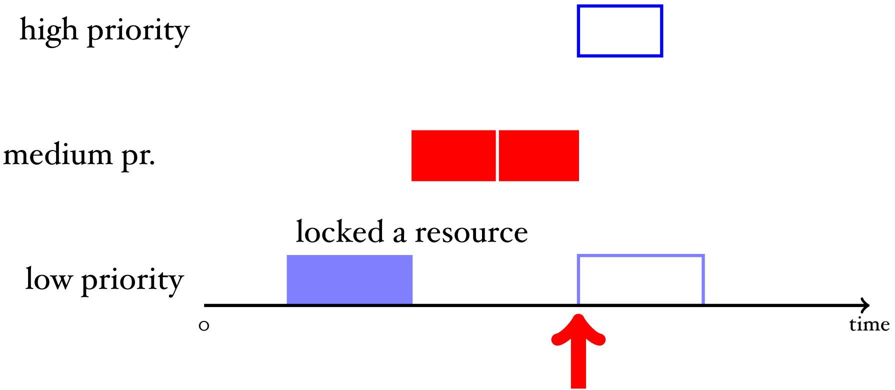
Scheduling: You want to avoid that a high priority process is staved indefinitely.



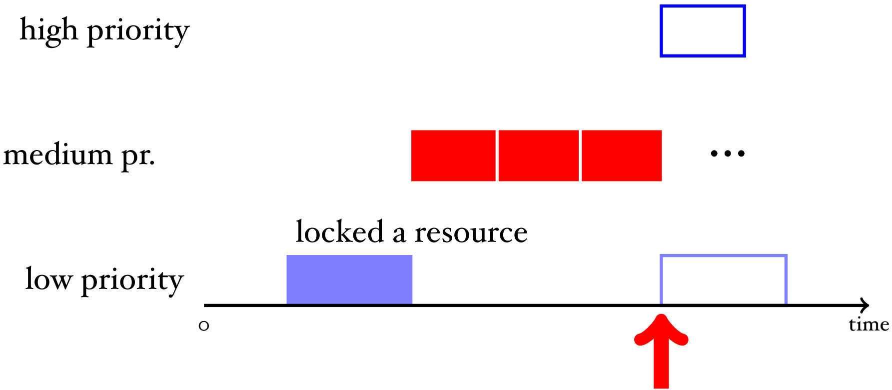
Scheduling: You want to avoid that a high priority process is staved indefinitely.



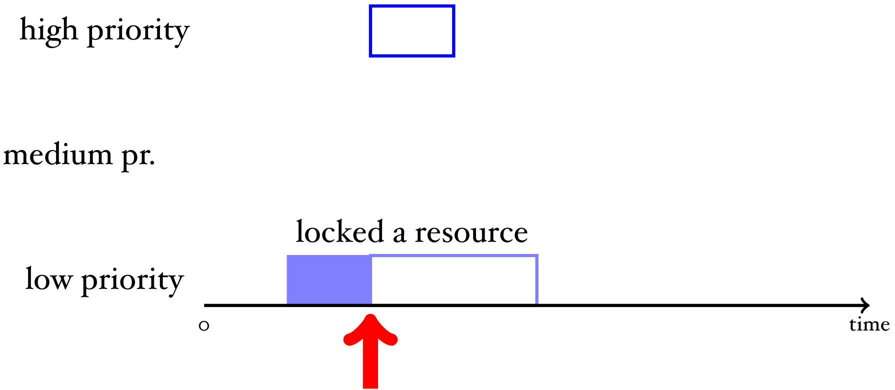
Scheduling: You want to avoid that a high priority process is staved indefinitely.



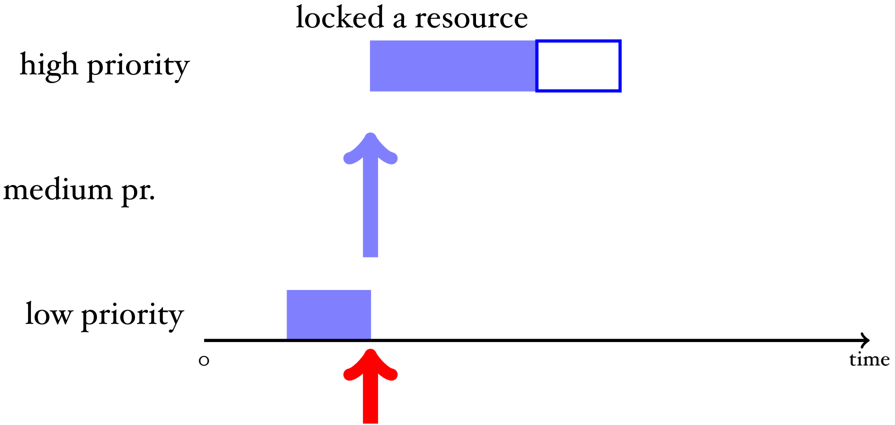
Scheduling: You want to avoid that a high priority process is staved indefinitely.



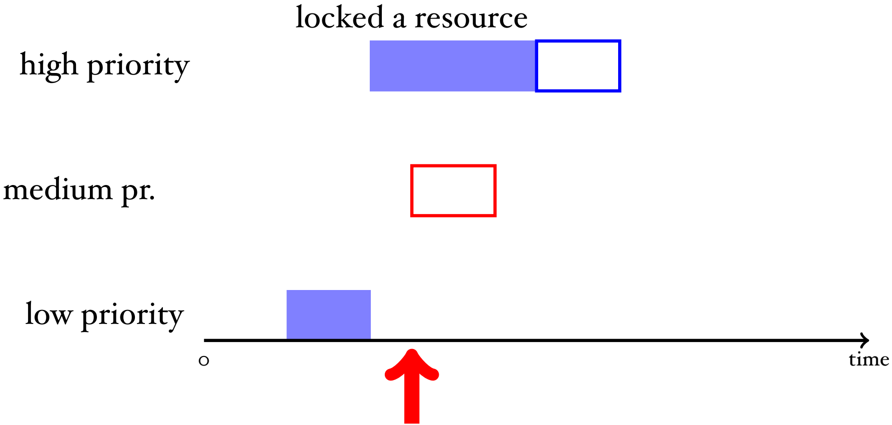
Scheduling: You want to avoid that a high priority process is staved indefinitely.



Scheduling: You want to avoid that a high priority process is staved indefinitely.



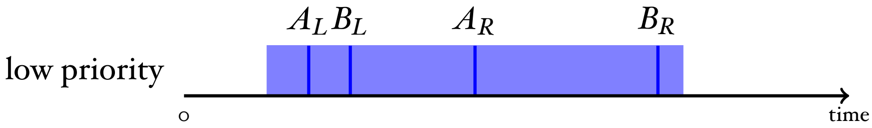
Scheduling: You want to avoid that a high priority process is staved indefinitely.

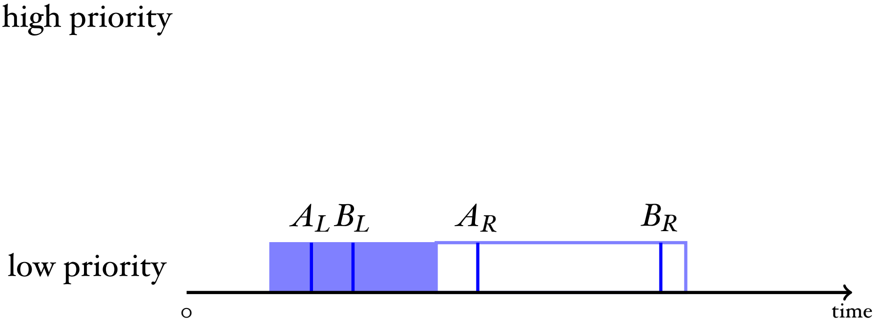


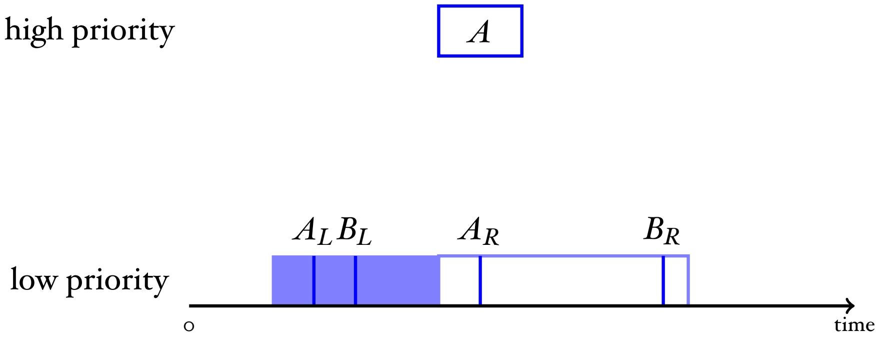
Scheduling: You want to avoid that a high priority process is staved indefinitely.

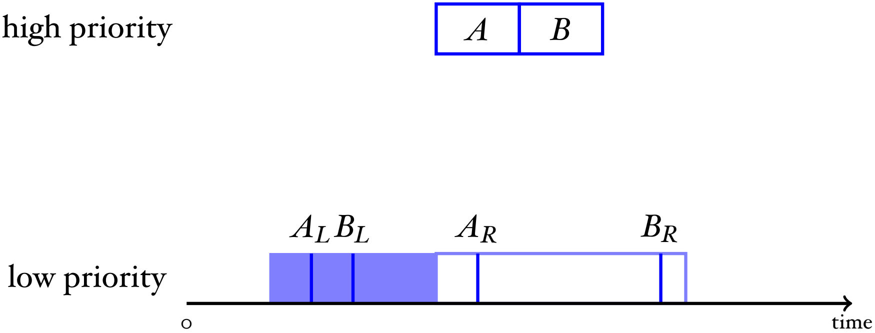
Priority Inheritance Scheduling

- Let a low priority process L temporarily inherit the high priority of H until L leaves the critical section unlocking the resource.
- Once the resource is unlocked L returns to its original priority level.

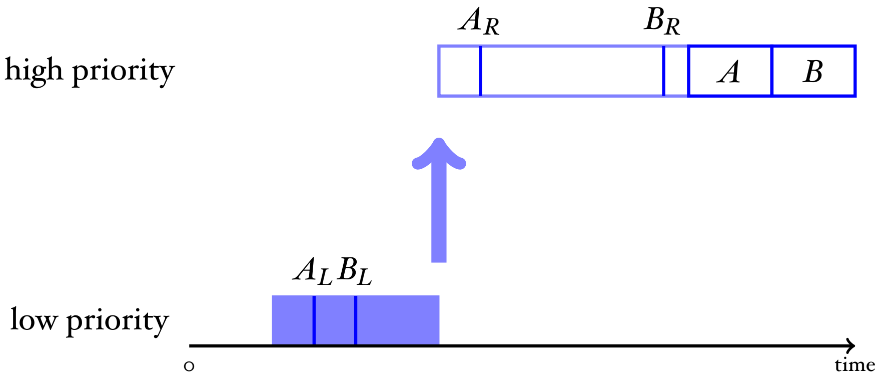

A_L B_L
A_R
B_R
low priority
0
time

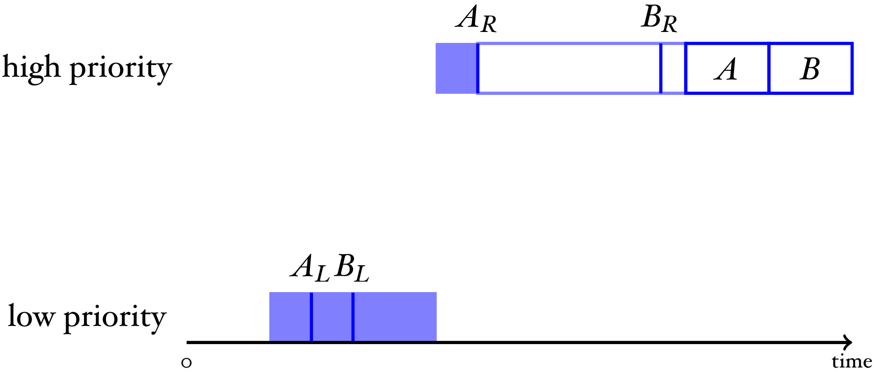

high priority
low priority
A_L B_L
A_R
B_R
0
time

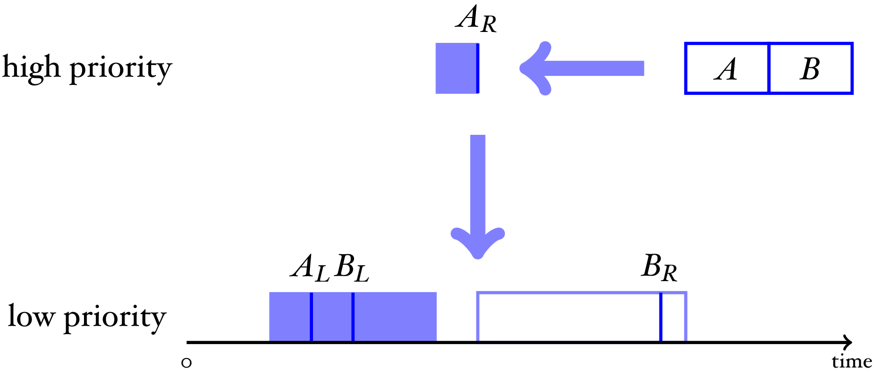

high priority
A
low priority
A_L B_L
A_R
B_R
0
time

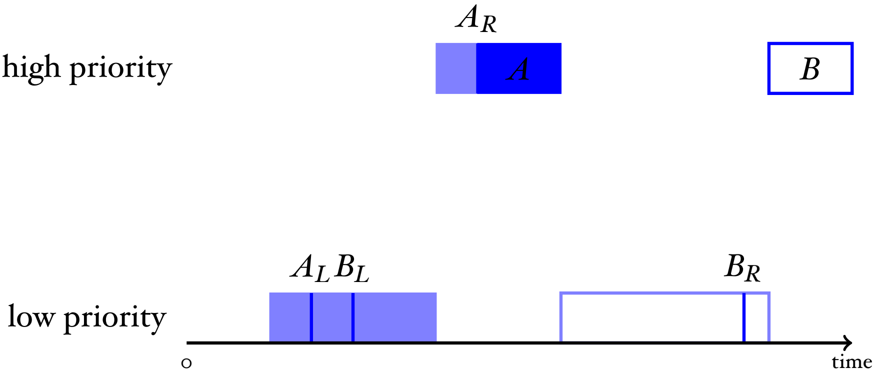


high priority
A
B
low priority
A_L B_L
A_R
B_R
0
time

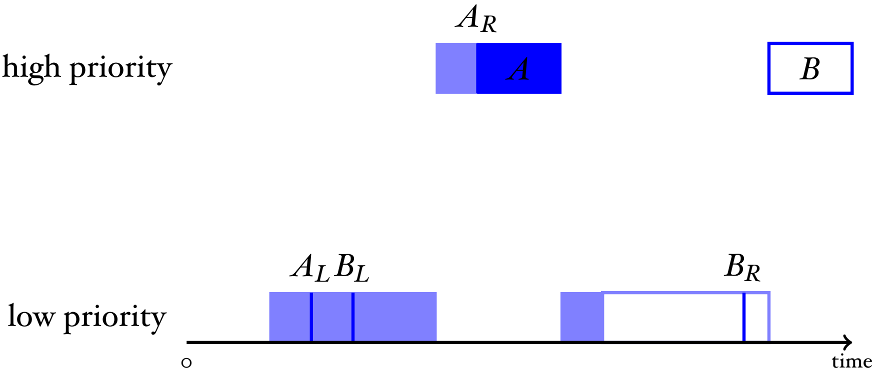

high priority
A_R
B_R
A
B
A_L B_L
low priority
0
time

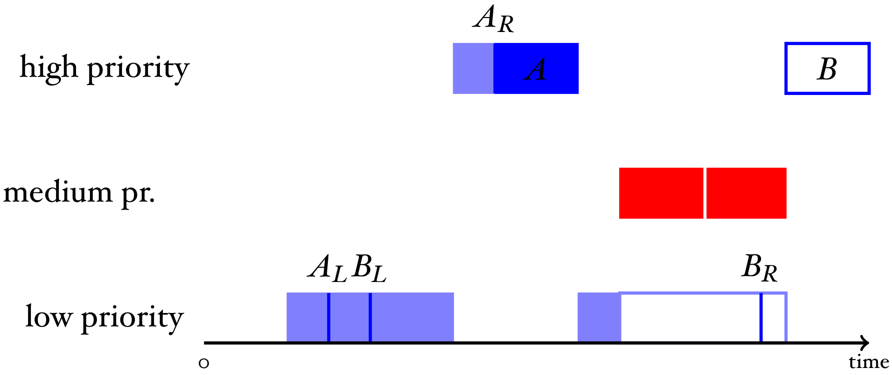

A_R
B_R
high priority
A
B
A_L B_L
low priority
0
time

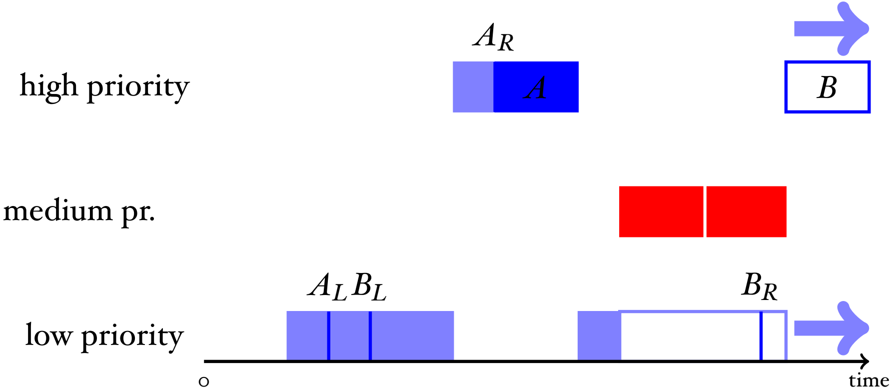

A_R
high priority
A
B
A_L B_L
B_R
low priority
0
time



A_R
high priority
A
B
A_L B_L
B_R
low priority
0
time


A_R
high priority
A
B
A_L B_L
B_R
low priority
0
time


A_R
high priority
A
B
medium pr.
A_L B_L
B_R
low priority
0
time



Scheduling: You want to avoid that a high priority process is staved indefinitely.

Priority Inheritance Scheduling

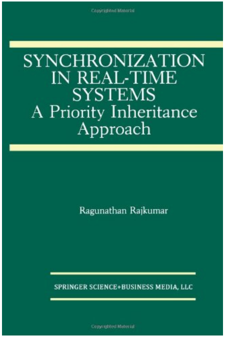
- Let a low priority process L temporarily inherit the high priority of H until L leaves the critical section unlocking the resource.

- Once the resource is unlocked L returns to its original priority level. **BOGUS**

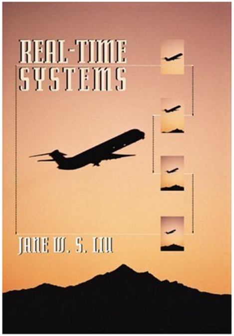
Priority Inheritance Scheduling

- Let a low priority process L temporarily inherit the high priority of H until L leaves the critical section unlocking the resource.
- Once the resource is unlocked L returns to its original priority level. **BOGUS**
- ...L needs to switch to the highest remaining priority of the threads that it blocks.

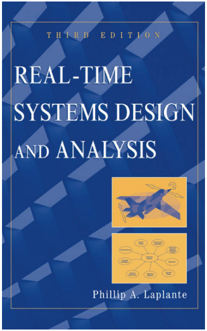
this error is already known since around 1999



- by Rajkumar, 1991
- *"it resumes the priority it had at the point of entry into the critical section"*



- by Jane Liu, 2000
- *"The job J_l executes at its inherited priority until it releases R; at that time, the priority of J_l returns to its priority at the time when it acquires the resource R."*
- gives pseudo code and totally bogus data structures
- interesting part "*left as an exercise*"



- by Laplante and Ovaska, 2011 ($113.76)
- *"when [the task] exits the critical section that caused the block, it reverts to the priority it had when it entered that section"*

Priority Scheduling

- a scheduling algorithm that is widely used in real-time operating systems
- has been "proved" correct by hand in a paper in 1983
- but this algorithm turned out to be incorrect, despite its "proof"

Priority Scheduling

- a scheduling algorithm that is widely used in real-time operating systems
- has been "proved" correct by hand in a paper in 1983
- but this algorithm turned out to be incorrect, despite its "proof"

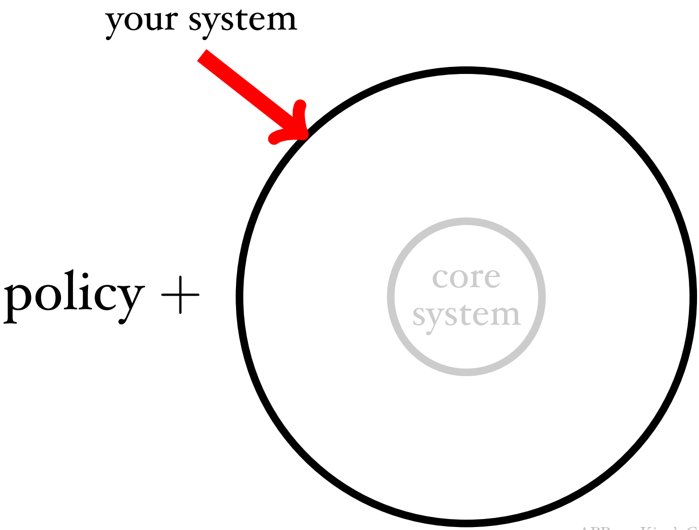
- we corrected the algorithm and then **really** proved that it is correct
- we implemented this algorithm in a small OS called PINTOS (used for teaching at Stanford)
- our implementation was much more efficient than their reference implementation

Design of AC-Policies

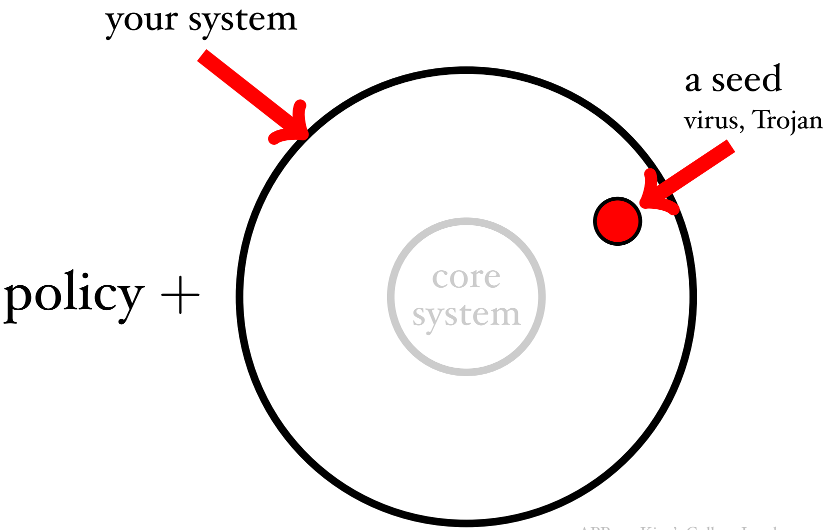
"what you specify is what you get but not necessarily what you want..."

main work by Chunhan Wu (PhD-student)

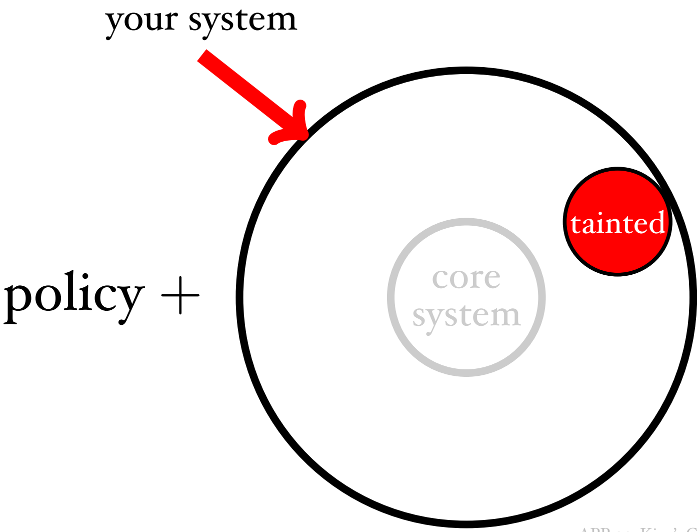
Testing Policies



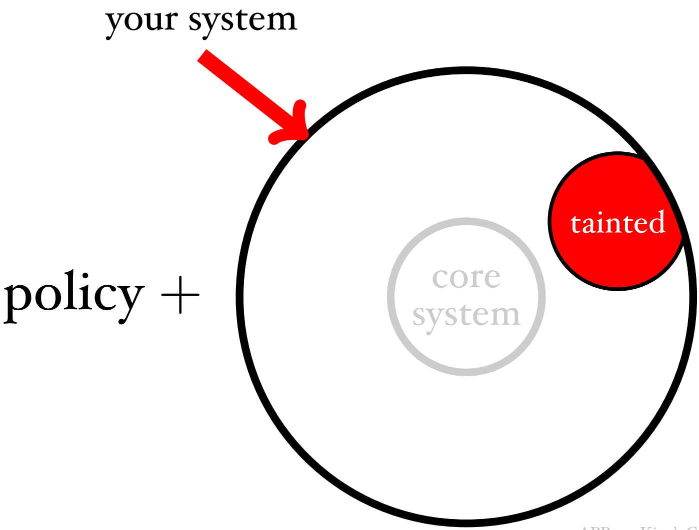
Testing Policies



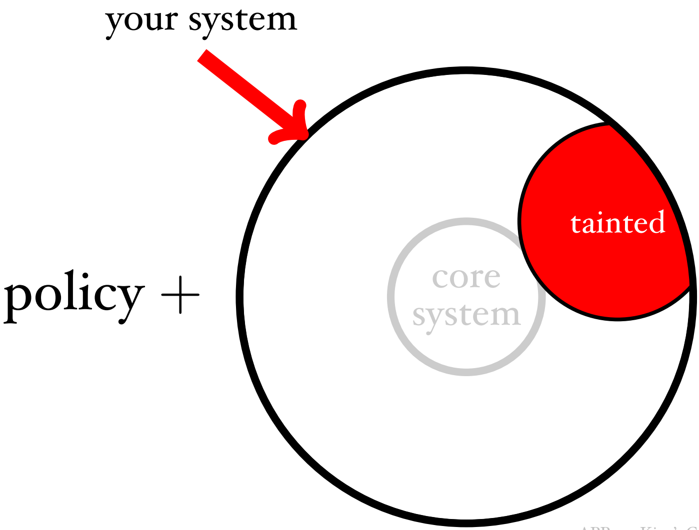
Testing Policies



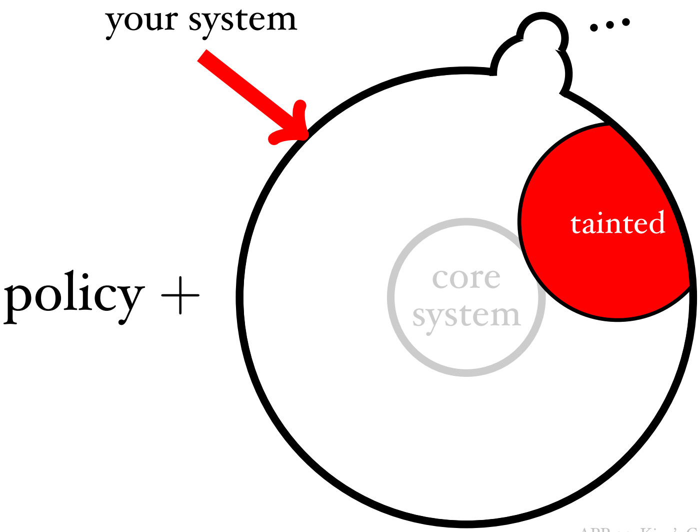
Testing Policies



Testing Policies



Testing Policies



A Sound and Complete Test

- working purely in the *dynamic world* does not work – infinite state space
- working purely on *static* policies also does not work – because of over approximation
  - sup+e a tainted file has type *bin* and
  - there is a role r which can both read and write *bin*-files

A Sound and Complete Test

- working purely in the *dynamic world* does not work – infinite state space
- working purely on *static* policies also does not work – because of over approximation
  - sup+e a tainted file has type *bin* and
  - there is a role r which can both read and write *bin*-files
  - then we would conclude that this tainted file can spread

A Sound and Complete Test

- working purely in the *dynamic world* does not work – infinite state space
- working purely on *static* policies also does not work – because of over approximation
  - sup+e a tainted file has type *bin* and
  - there is a role r which can both read and write *bin*-files
  - then we would conclude that this tainted file can spread
  - but if there is no process with role r and it will never been created, then the file actually does not spread

A Sound and Complete Test

- working purely in the *dynamic world* does not work – infinite state space
- working purely on *static* policies also does not work – because of over approximation
  - sup+e a tainted file has type *bin* and
  - there is a role r which can both read and write *bin*-files
  - then we would conclude that this tainted file can spread
  - but if there is no process with role r and it will never been created, then the file actually does not spread
- our solution: take a middle ground and record precisely the information of the initial state, but be less precise about every newly created object.

Big Proofs in CS

Formal proofs in CS sound like science fiction?
Completely irrelevant! Lecturer gone mad?

Big Proofs in CS

Formal proofs in CS sound like science fiction?
Completely irrelevant! Lecturer gone mad?

- in 2008, verification of a small C-compiler
  - "if my input program has a certain behaviour, then the compiled machine code has the same behaviour"
  - is as good as `gcc -O1`, but much less buggy
- in 2010, verification of a micro-kernel operating system (approximately 8700 loc)
  - 200k loc of proof
  - 25 - 30 person years
  - found 160 bugs in the C code (144 by the proof)

Goal

Remember the Bridges example?

- Can we look at our programs and somehow ensure they are secure/bug free/correct?

Goal

Remember the Bridges example?

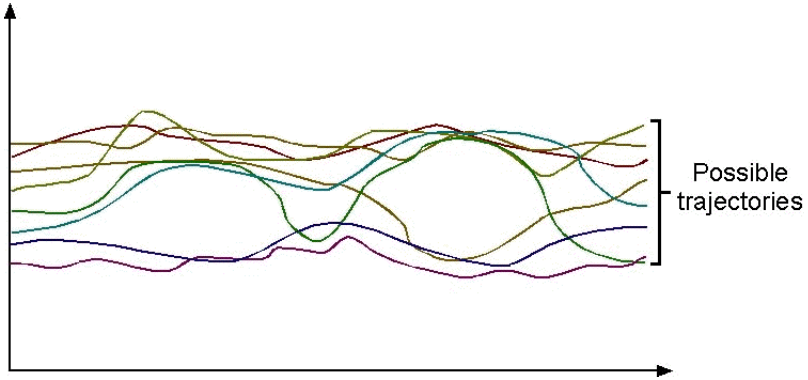
- Can we look at our programs and somehow ensure they are secure/bug free/correct?
- Very hard: Anything interesting about programs is equivalent to halting problem, which is undecidable.

Goal

Remember the Bridges example?

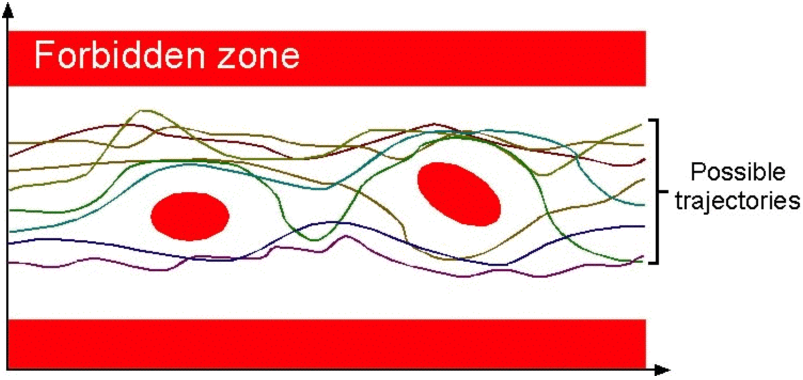
- Can we look at our programs and somehow ensure they are secure/bug free/correct?
- Very hard: Anything interesting about programs is equivalent to halting problem, which is undecidable.
- Solution: We avoid this "minor" obstacle by being as close as +sible of deciding the halting problem, without actually deciding the halting problem. $\Rightarrow$ static analysis

What is Static Analysis?

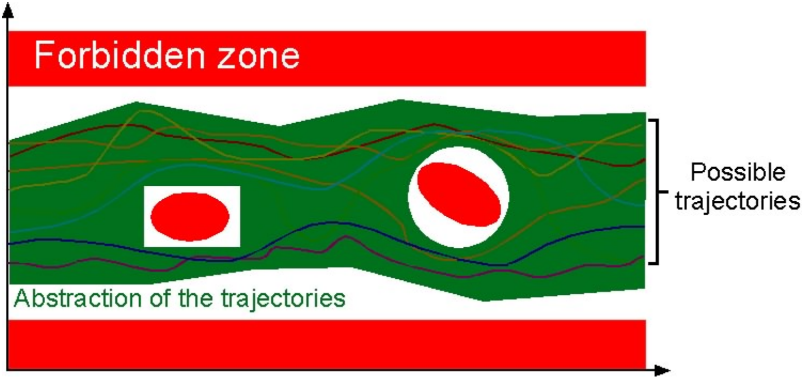


- depending on some initial input, a program (behaviour) will “develop” over time.

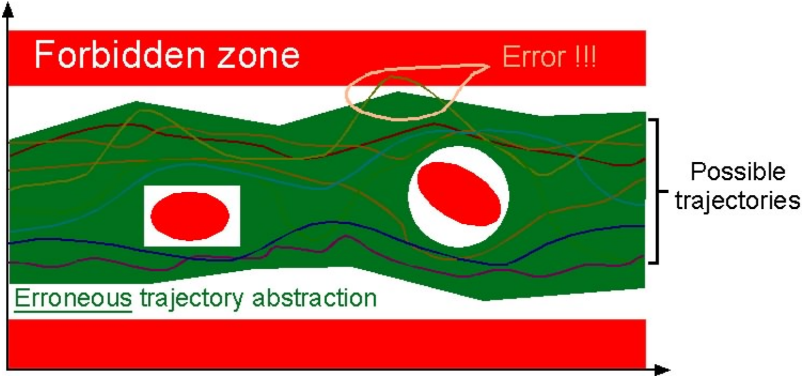
What is Static Analysis?



What is Static Analysis?

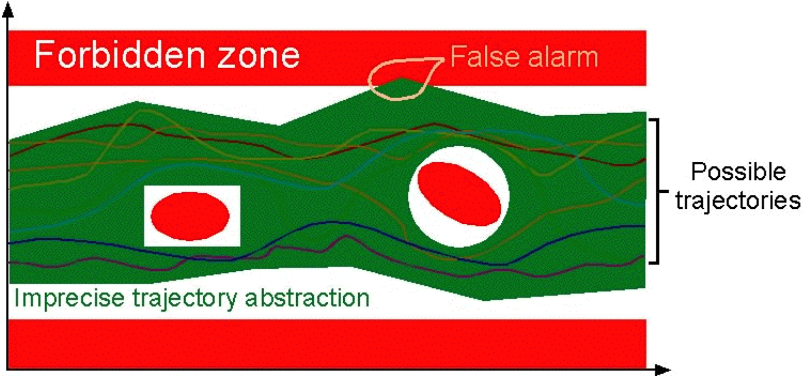


What is Static Analysis?



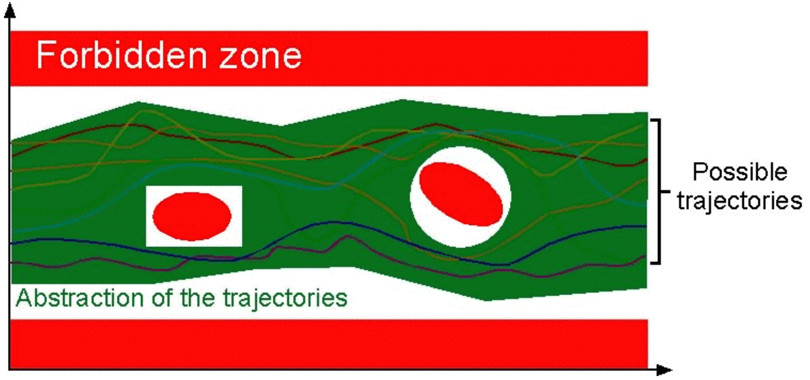
- to be avoided

What is Static Analysis?



- this needs more work

What is Static Analysis?



Concrete Example: Sign-Analysis

$$
\begin{array}{rcl}
\langle Exp\rangle & ::= & \langle Exp\rangle \texttt{ + } \langle Exp\rangle \\
 & | & \langle Exp\rangle \texttt{ * } \langle Exp\rangle \\
 & | & \langle Exp\rangle \texttt{ = } \langle Exp\rangle \\
 & | & \langle num\rangle \\
 & | & \langle var\rangle \\
\langle Stmt\rangle & ::= & \langle label\rangle \texttt{ :} \\
 & | & \langle var\rangle \texttt{ := } \langle Exp\rangle \\
 & | & \text{jump? } \langle Exp\rangle\ \langle label\rangle \\
 & | & \text{goto } \langle label\rangle \\
\langle Prog\rangle & ::= & \langle Stmt\rangle \ldots
\end{array}
$$

Concrete Example: Sign-Analysis

$$
\begin{array}{rcl}
\langle Exp \rangle & ::= & \langle Exp \rangle \texttt{ + } \langle Exp \rangle \\
& | & \langle Exp \rangle \texttt{ * } \langle Exp \rangle \\
& | & \langle Exp \rangle \texttt{ = } \langle Exp \rangle \\
& | & \langle num \rangle \\
& | & \langle var \rangle \\
\langle Stmt \rangle & ::= & \langle label \rangle \texttt{ :} \\
& | & \langle var \rangle \texttt{ := } \langle Exp \rangle \\
& | & \text{jump? } \langle Exp \rangle \ \langle label \rangle \\
& | & \text{goto } \langle label \rangle \\
\langle Prog \rangle & ::= & \langle Stmt \rangle \ \ldots
\end{array}
$$

```
      a := 1
      n := 5
top:  jump? n = 0 done
      a := a * n
      n := n + -1
      goto top
done:
```

Concrete Example: Sign-Analysis

$$
\begin{array}{rcl}
\langle Exp\rangle & ::= & \langle Exp\rangle + \langle Exp\rangle \\
& | & \langle Exp\rangle * \langle Exp\rangle \\
& | & \langle Exp\rangle = \langle Exp\rangle \\
& | & \langle num\rangle \\
& | & \langle var\rangle \\
\langle Stmt\rangle & ::= & \langle label\rangle : \\
& | & \langle var\rangle := \langle Exp \\
& | & \text{jmp?}\ \langle Exp\rangle\ \langle \\
& | & \text{goto}\ \langle label\rangle \\
\langle Prog\rangle & ::= & \langle Stmt\rangle \ldots
\end{array}
$$

```
      n := 6
      m1 := 0
      m2 := 1
top:  jmp? n = 0 done
      tmp := m2
      m2 := m1 + m2
      m1 := tmp
      n := n + -1
      goto top
done:
```

Eval

$$
\begin{aligned}
[n]_{env} &\stackrel{\text{def}}{=} n \\
[x]_{env} &\stackrel{\text{def}}{=} env(x) \\
[e_1 + e_2]_{env} &\stackrel{\text{def}}{=} [e_1]_{env} + [e_2]_{env} \\
[e_1 * e_2]_{env} &\stackrel{\text{def}}{=} [e_1]_{env} * [e_2]_{env} \\
[e_1 = e_2]_{env} &\stackrel{\text{def}}{=} \begin{cases} 1 & \text{if} \quad [e_1]_{env} = [e_2]_{env} \\ 0 & \text{otherwise} \end{cases}
\end{aligned}
$$

```
def eval_exp(e: Exp, env: Env) : Int = e match {
  case Num(n) & n
  case Var(x) & env(x)
  case Plus(e1, e2) & eval_exp(e1, env) + eval_exp(e2, env)
  case Times(e1, e2) & eval_exp(e1, env) * eval_exp(e2, env)
  case Equ(e1, e2) &
    if (eval_exp(e1, env) == eval_exp(e2, env)) 1 else 0
}
```

A program

```
      a := 1
      n := 5
top:  jmp? n = 0 done
      a := a * n
      n := n + -1
      goto top
done:
```

Some snippets

```
"""   a := 1
      n := 5
top:  jmp? n = 0 done
      a := a * n
      n := n + -1
      goto top
done:
```

```
top:  jmp? n = 0 done
      a := a * n
      n := n + -1
      goto top
done:
```

```
done:
```

Eval for Stmts

Let sn be the snippets of a program

$$
\begin{array}{lcl}
[\texttt{nil}]_{env} & \stackrel{\text{def}}{=} & env \\
[\texttt{Label}(l:) :: rest]_{env} & \stackrel{\text{def}}{=} & [rest]_{env} \\
[x \texttt{:=} e :: rest]_{env} & \stackrel{\text{def}}{=} & [rest]_{(env[x:=[e]_{env}])} \\
[\texttt{jmp?}\ e\ l :: rest]_{env} & \stackrel{\text{def}}{=} & \begin{cases} [sn(l)]_{env} & \text{if} \quad [e]_{env} = 1 \\ [rest]_{env} & \text{otherwise} \end{cases} \\
[\texttt{goto}\ l :: rest]_{env} & \stackrel{\text{def}}{=} & [sn(l)]_{env}
\end{array}
$$

Start with $[sn(\text{""})]_{\varnothing}$

Eval in Code

```
def eval(sn: Snips) : Env = {
  def eval_stmts(sts: Stmts, env: Env) : Env = sts match {
    case Nil & env
    case Label(l)::rest & eval_stmts(rest, env)
    case Assign(x, e)::rest &
      eval_stmts(rest, env + (x -> eval_exp(e, env)))
    case Jump(b, l)::rest &
      if (eval_exp(b, env) == 1) eval_stmts(sn(l), env)
      else eval_stmts(rest, env)
    case Goto(l)::rest & eval_stmts(sn(l), env)
  }

  eval_stmts(sn(""), Map())
}
```

The Idea

```
      a := 1
      n := 5
top:  jmp? n = 0 done
      a := a * n
      n := n + -1
      goto top
done:
```

⇒

```
      a := '+'
      n := '+'
top:  jmp? n = '0' done
      a := a * n
      n := n + '-'
      goto top
done:
```

Replace all constants and variables by either `+`, `-` or `0`. What we want to find out is what the sign of `a` and `n` is (they are always positive).

Sign Analysis?

e_1	e_2	$e_1 + e_2$
−	−	−
−	0	−
−	+	−, 0, +
0	x	x
+	−	−, 0, +
+	0	+
+	+	+

e_1	e_2	$e_1 * e_2$
−	−	+
−	0	0
−	+	−
0	x	0
+	−	−
+	0	0
+	+	+

$$
\begin{array}{lcl}
[n]_{env} & \stackrel{\text{def}}{=} & \begin{cases} \{+\} & \text{if } n > 0 \\ \{-\} & \text{if } n < 0 \\ \{0\} & \text{if } n = 0 \end{cases} \\
[x]_{env} & \stackrel{\text{def}}{=} & env(x) \\
[e_1 + e_2]_{env} & \stackrel{\text{def}}{=} & [e_1]_{env} \oplus [e_2]_{env} \\
[e_1 * e_2]_{env} & \stackrel{\text{def}}{=} & [e_1]_{env} \otimes [e_2]_{env} \\
[e_1 = e_2]_{env} & \stackrel{\text{def}}{=} & \{0, +\}
\end{array}
$$

```
def aeval_exp(e: Exp, aenv: AEnv) : Set[Abst] = e match {
  case Num(0) => Set(Zero)
  case Num(n) if (n < 0) => Set(Neg)
  case Num(n) if (n > 0) => Set(Pos)
  case Var(x) => aenv(x)
  case Plus(e1, e2) =>
    aplus(aeval_exp(e1, aenv), aeval_exp(e2, aenv))
  case Times(e1, e2) =>
    atimes(aeval_exp(e1, aenv), aeval_exp(e2, aenv))
  case Equ(e1, e2) => Set(Zero, Pos)
}
```

Sign Analysis

- We want to find out whether `a` and `n` are always positive?
- After a few optimisations, we can indeed find this out.
  - `if` returns `+` or `0`
  - branch into only one dircection if you know
  - if x is `+`, then $x +$ `-1` cannot be negative
- What is this good for? Well, you do not need underflow checks (in order to prevent buffer-overflow attacks).