# Optimising the compilation of Petri net models

Łukasz Fronc and Franck Pommereau
{fronc,pommereau}@ibisc.univ-evry.fr

SUMo (at PETRI NETS), June 21th, 2011


# Outlines

Domain:

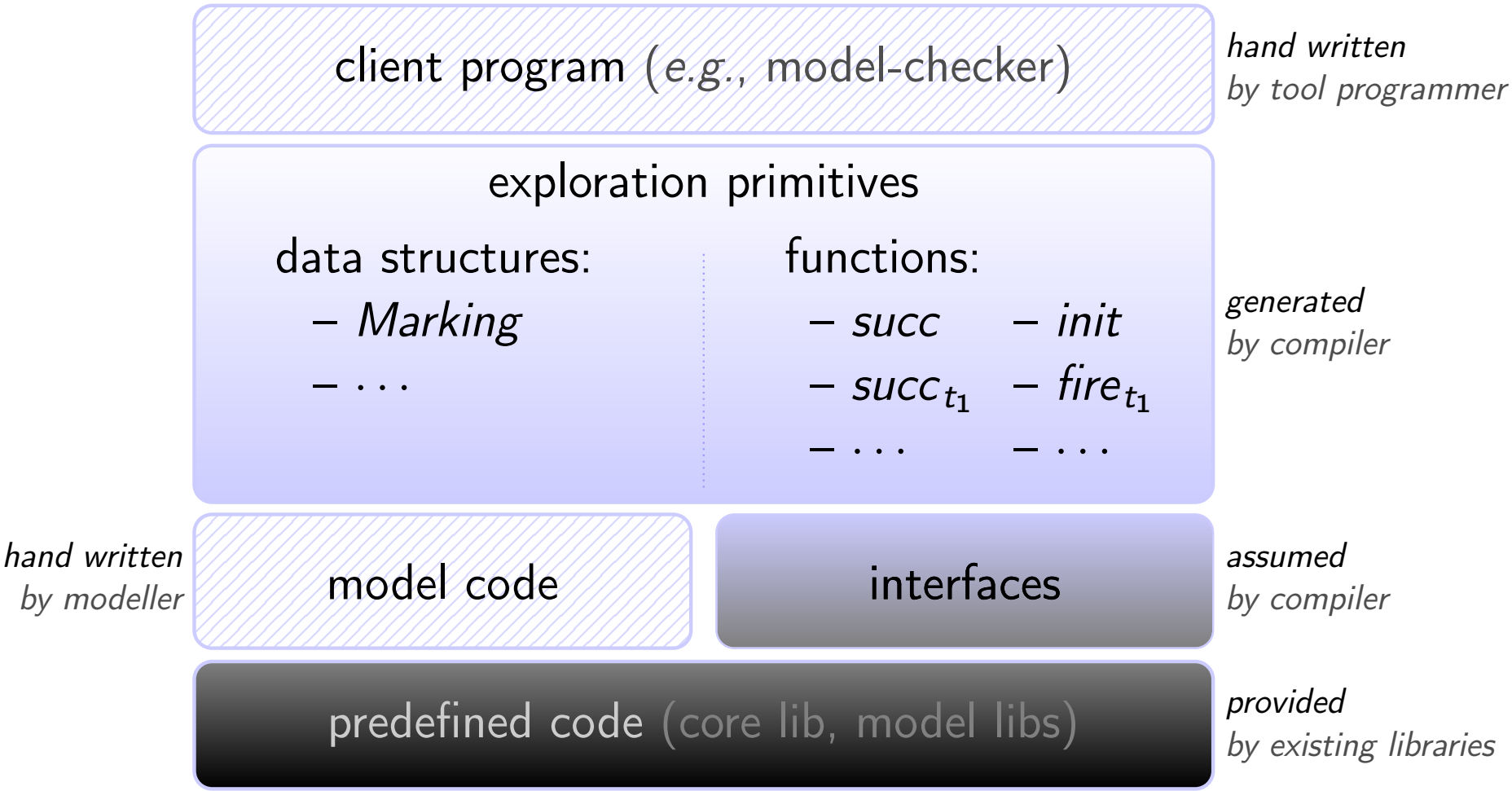
- coloured Petri nets
  - colour domain is Python
  - infinite place types allowed
  - output arcs do computation (arbitrary Python expressions)
  - input arcs bind variables (with pattern matching)
- explicit model-checking (or simulation)

Goal: accelerate transition firing

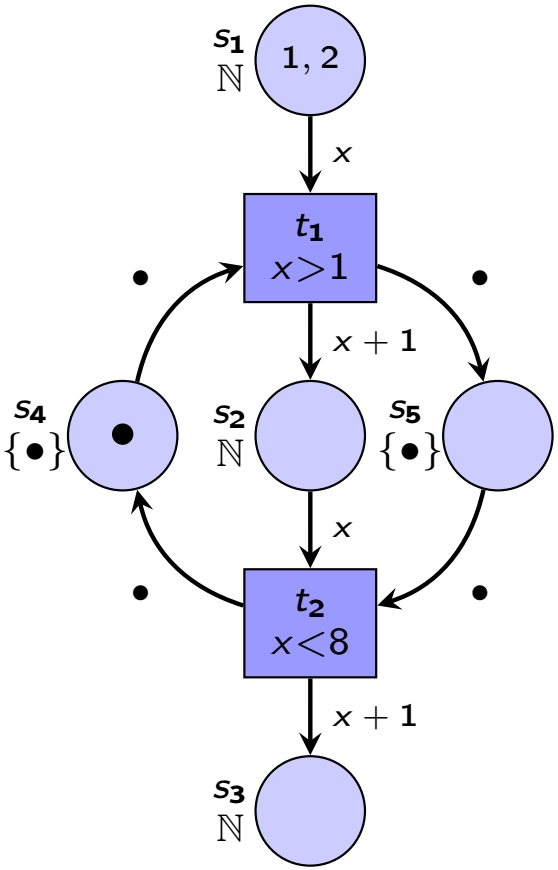
- use model compilation
  - used by Helena, Spin, ...
  - remove many data structures (remain: markings, functions)
  - produce simple and efficient code (model-specific)
- exploit model-dependent optimisations

# Contents

# Produced code



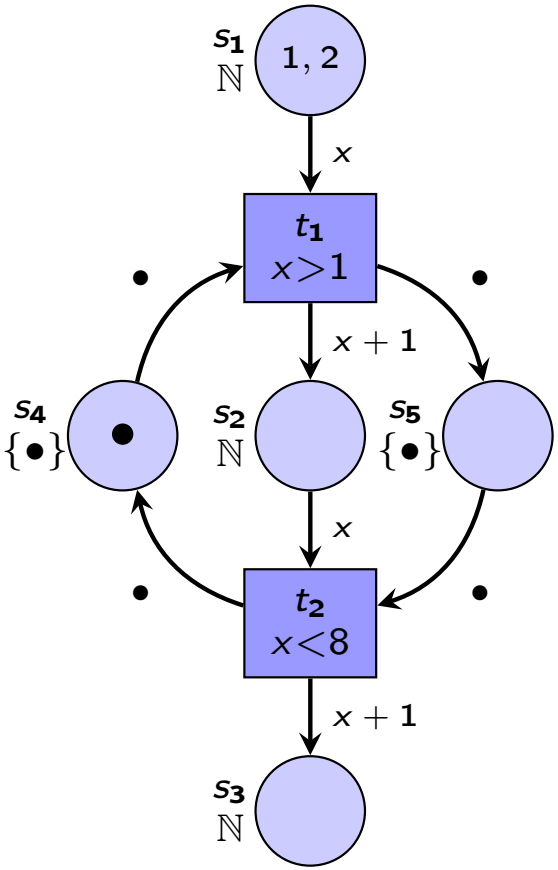
# Interpreting successor algorithm




**succ** : $M$ : *Marking*, $t$ : *Transition* $\rightarrow$ *MarkingSet*

*next* $\leftarrow \emptyset$
**for** $b$ **in** $modes(M, t)$ **do**
  *next* $\leftarrow$ *next* $\cup \{fire(M, t, b)\}$
**endfor**
**return** *next*

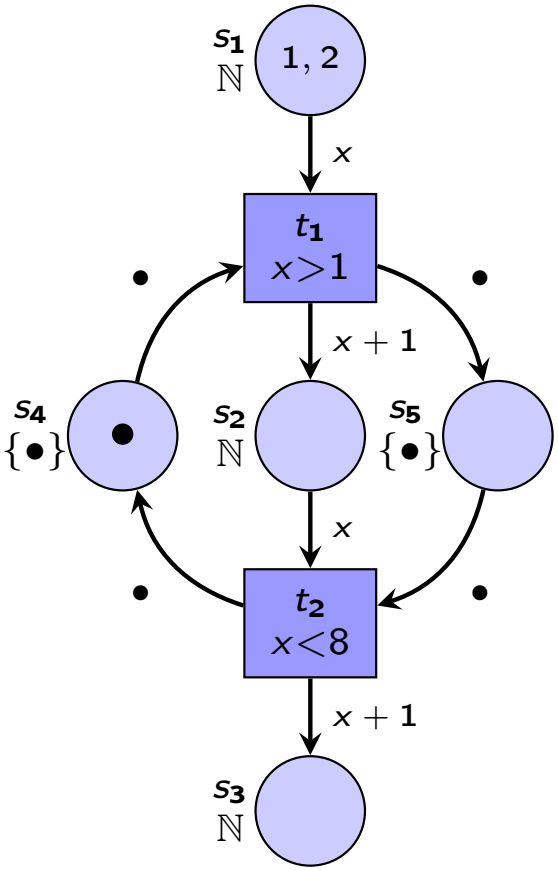
# Compiled successor algorithm




**succ$_{t_1}$** : $M$ : *Marking* $\rightarrow$ *MarkingSet*

$next \leftarrow \emptyset$
**for** $x$ **in** $M(s_1)$ **do**
  **for** $token_{s_4}$ **in** $M(s_4)$ **do**
    **if** $x > 1$ **then**
      $next \leftarrow next \cup \{fire_{t_1}(M, x)\}$
    **endif**
  **endfor**
**endfor**
**return** *next*

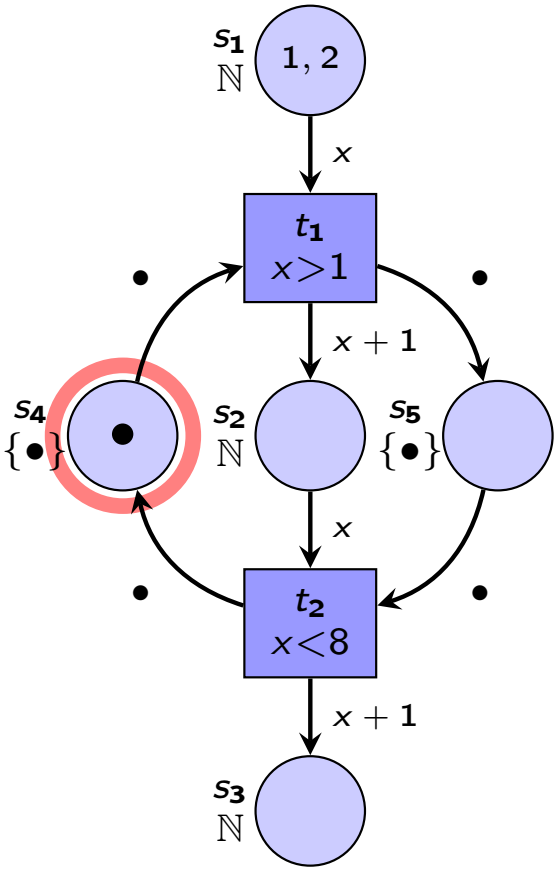
# Example of successor algorithm optimisations




**succ$_{t_1}$** : $M$ : *Marking* $\rightarrow$ *MarkingSet*

$next \leftarrow \emptyset$
**for** $x$ **in** $M(s_1)$ **do**
  **for** $token_{s_4}$ **in** $M(s_4)$ **do**
    **if** $x > 1$ **then**
      $next \leftarrow next \cup \{fire_{t_1}(M, x)\}$
    **endif**
  **endfor**
**endfor**
**return** *next*

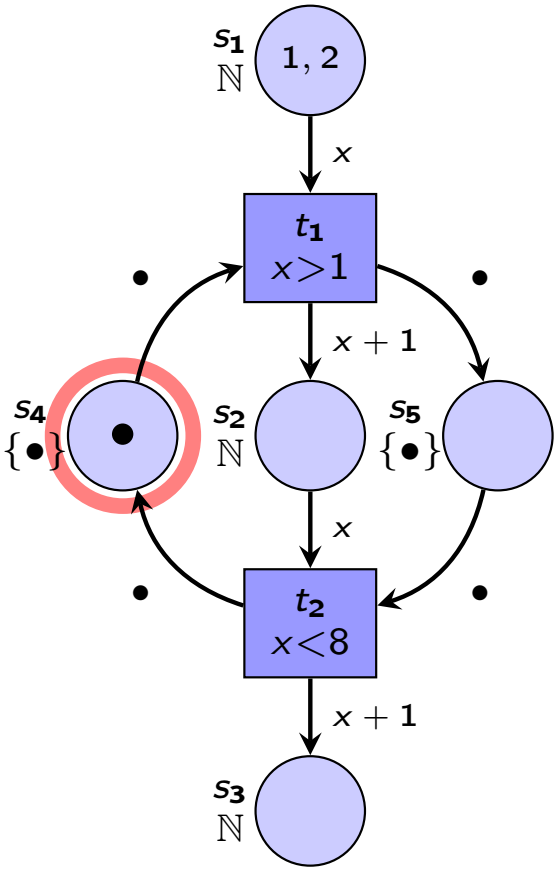
# Example of successor algorithm optimisations




**succ$_{t_1}$** : $M$ : *Marking* $\rightarrow$ *MarkingSet*

$next \leftarrow \emptyset$
**for** $x$ **in** $M(s_1)$ **do**
  **for** $token_{s_4}$ **in** $M(s_4)$ **do**
    **if** $x > 1$ **then**
      $next \leftarrow next \cup \{fire_{t_1}(M, x)\}$
    **endif**
  **endfor**
**endfor**
**return** *next*

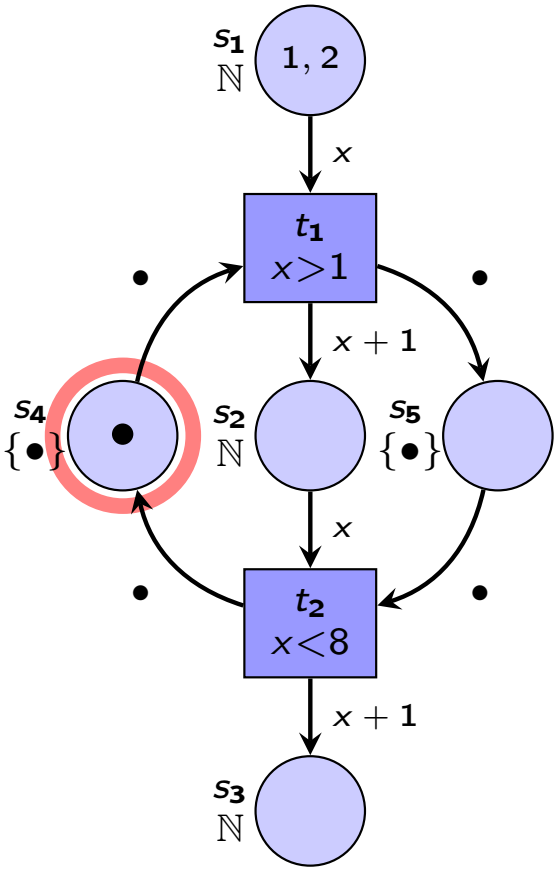
# Example of successor algorithm optimisations




**succ$_{t_1}$** : $M$ : *Marking* $\rightarrow$ *MarkingSet*

```
next ← ∅
for x in M(s1) do
    for token_s4 in M(s4) do
        if x > 1 then
            next ← next ∪ {fire_t1(M, x)}
        endif
    endfor
endfor
return next
```

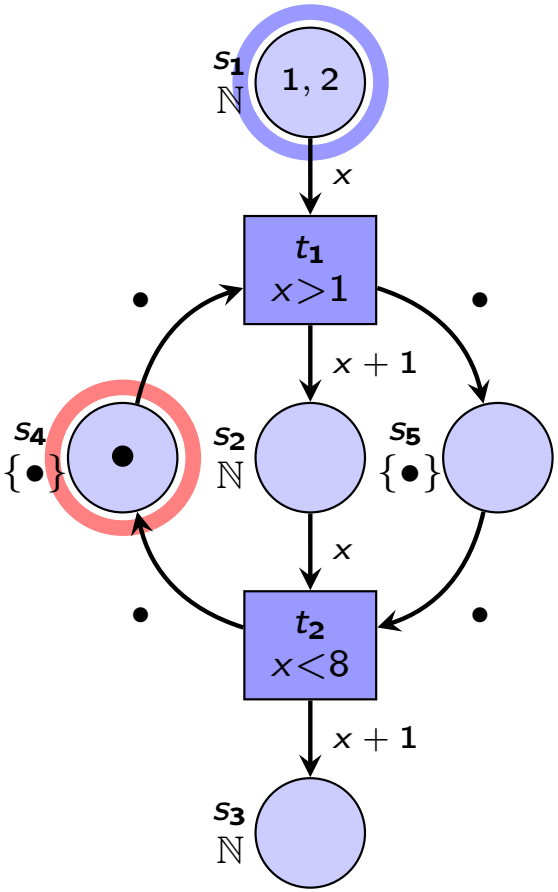
# Example of successor algorithm optimisations




**succ$_{t_1}$** : $M$ : *Marking* $\rightarrow$ *MarkingSet*

```
next ← ∅
for x in M(s1) do
    if notempty(M(s4)) then
        if x > 1 then
            next ← next ∪ {fire_t1(M, x)}
        endif
    endif
endfor
return next
```

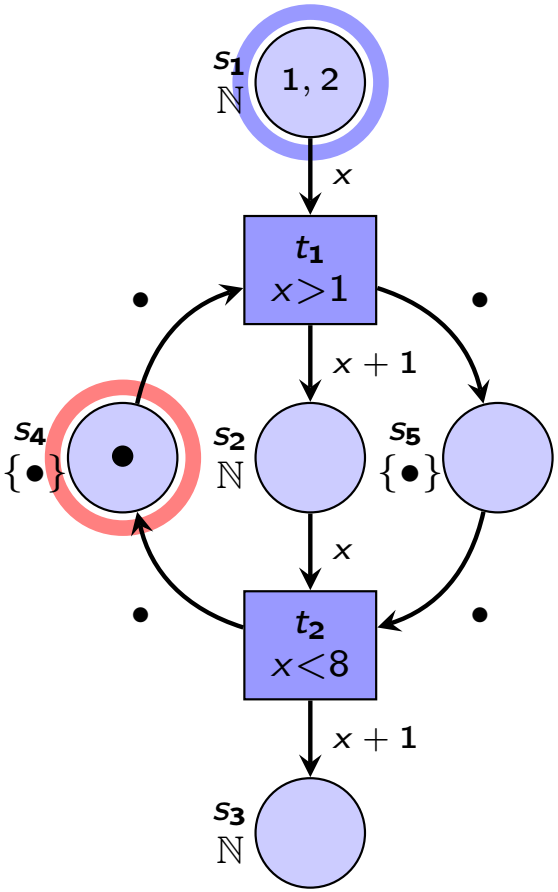
# Example of successor algorithm optimisations




**succ$_{t_1}$** : $M$ : *Marking* $\rightarrow$ *MarkingSet*

```
next ← ∅
for x in M(s1) do
    if notempty(M(s4)) then
        if x > 1 then
            next ← next ∪ {fire_t1(M, x)}
        endif
    endif
endfor
return next
```

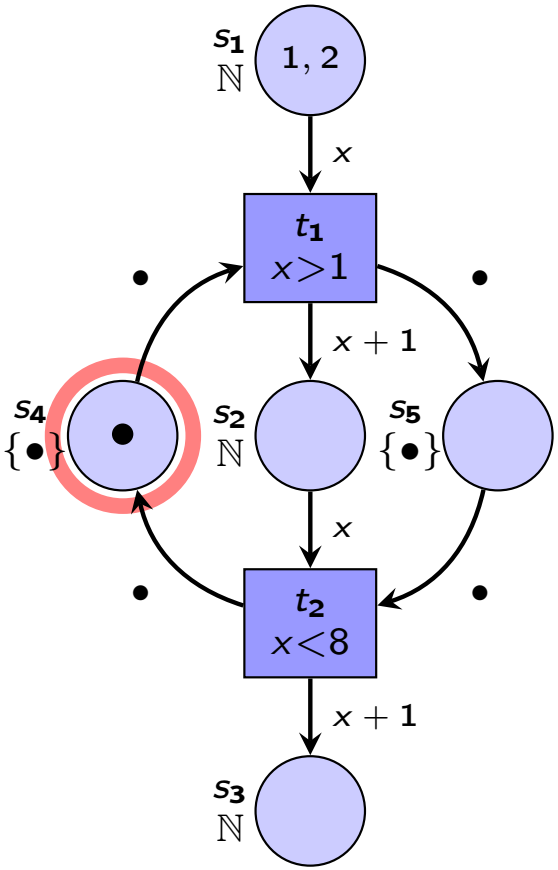
# Example of successor algorithm optimisations




**succ$_{t_1}$** : $M$ : *Marking* $\rightarrow$ *MarkingSet*

```
next ← ∅
if notempty(M(s4)) then
    for x in M(s1) do
        if x > 1 then
            next ← next ∪ {fire_t1(M, x)}
        endif
    endfor
endif
return next
```

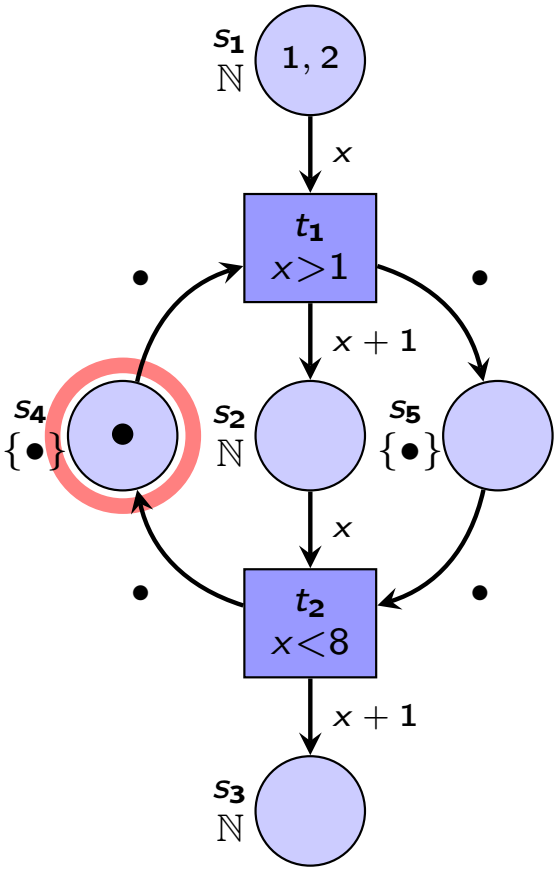
# Example of successor algorithm optimisations




**succ$_{t_1}$** : $M$ : *Marking* $\rightarrow$ *MarkingSet*

```
next ← ∅
if notempty(M(s4)) then
    for x in M(s1) do
        if x > 1 then
            next ← next ∪ {fire_t1(M, x)}
        endif
    endfor
endif
return next
```

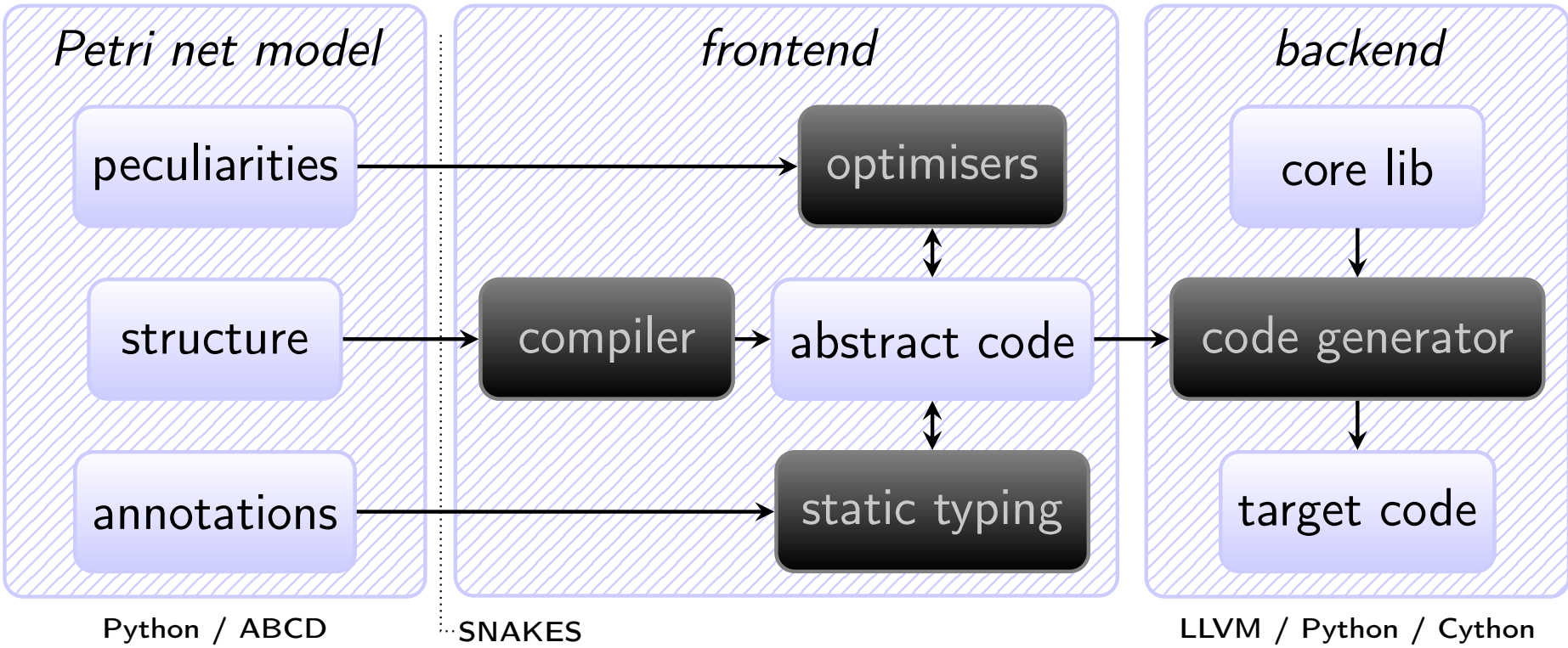
# Example of successor algorithm optimisations




**succ$_{t_1}$** : $M$ : *Marking* $\rightarrow$ *MarkingSet*

```
next ← ∅
if M(s4) then
    for x in M(s1) do
        if x > 1 then
            next ← next ∪ {fire_t1(M, x)}
        endif
    endfor
endif
return next
```

# Compiler architecture




python powered

# Experimental results

We compare:

our prototype using the Cython backend

most efficient, as expressive as Python

with Helena model checker

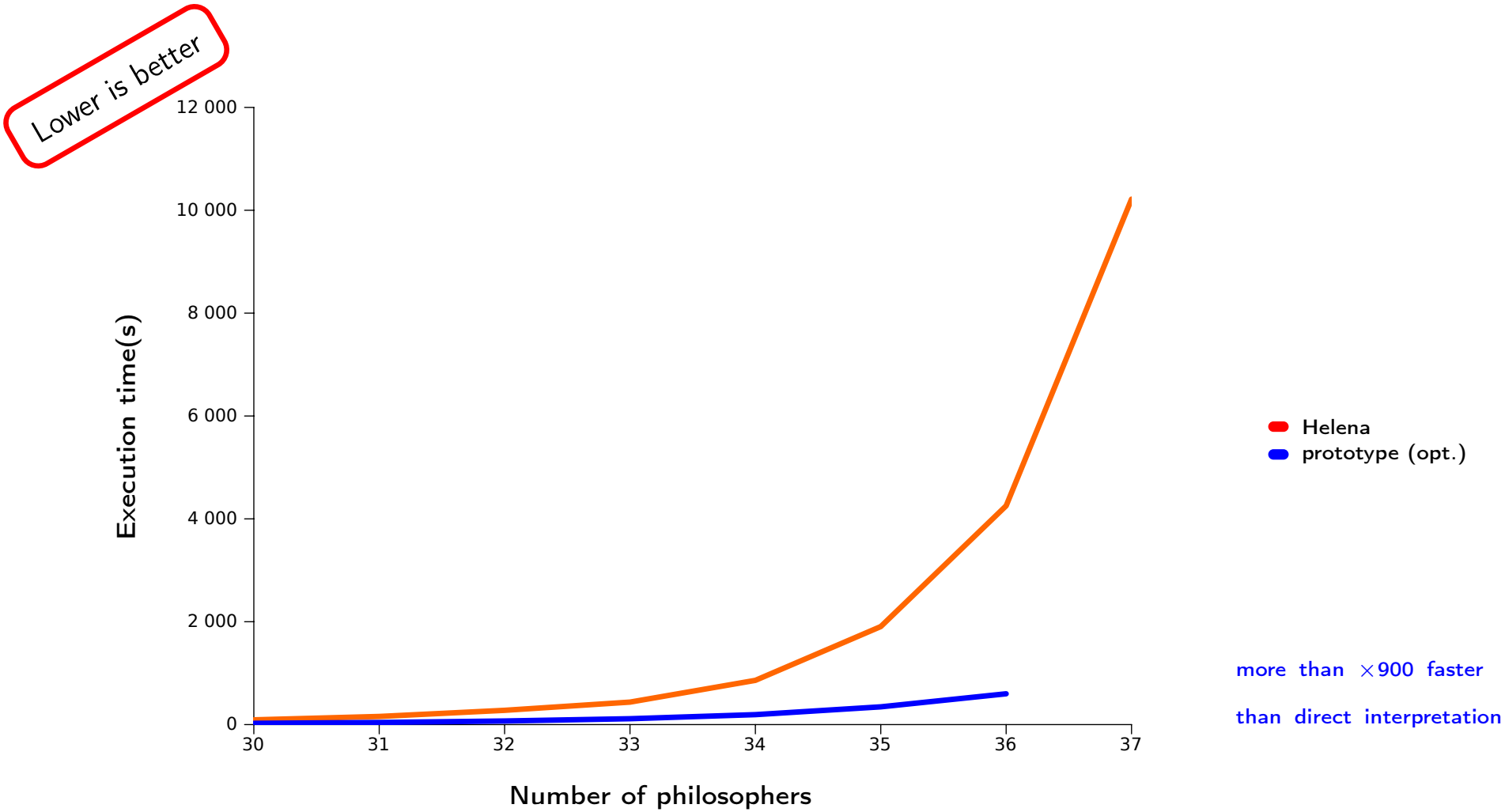
static reductions disabled

Three test cases:

- ◇ dining philosophers — one bounded Petri net
- ◇ a railroad crossing model — colored Petri net
- ◇ a security protocol model — "highly" colored Petri net

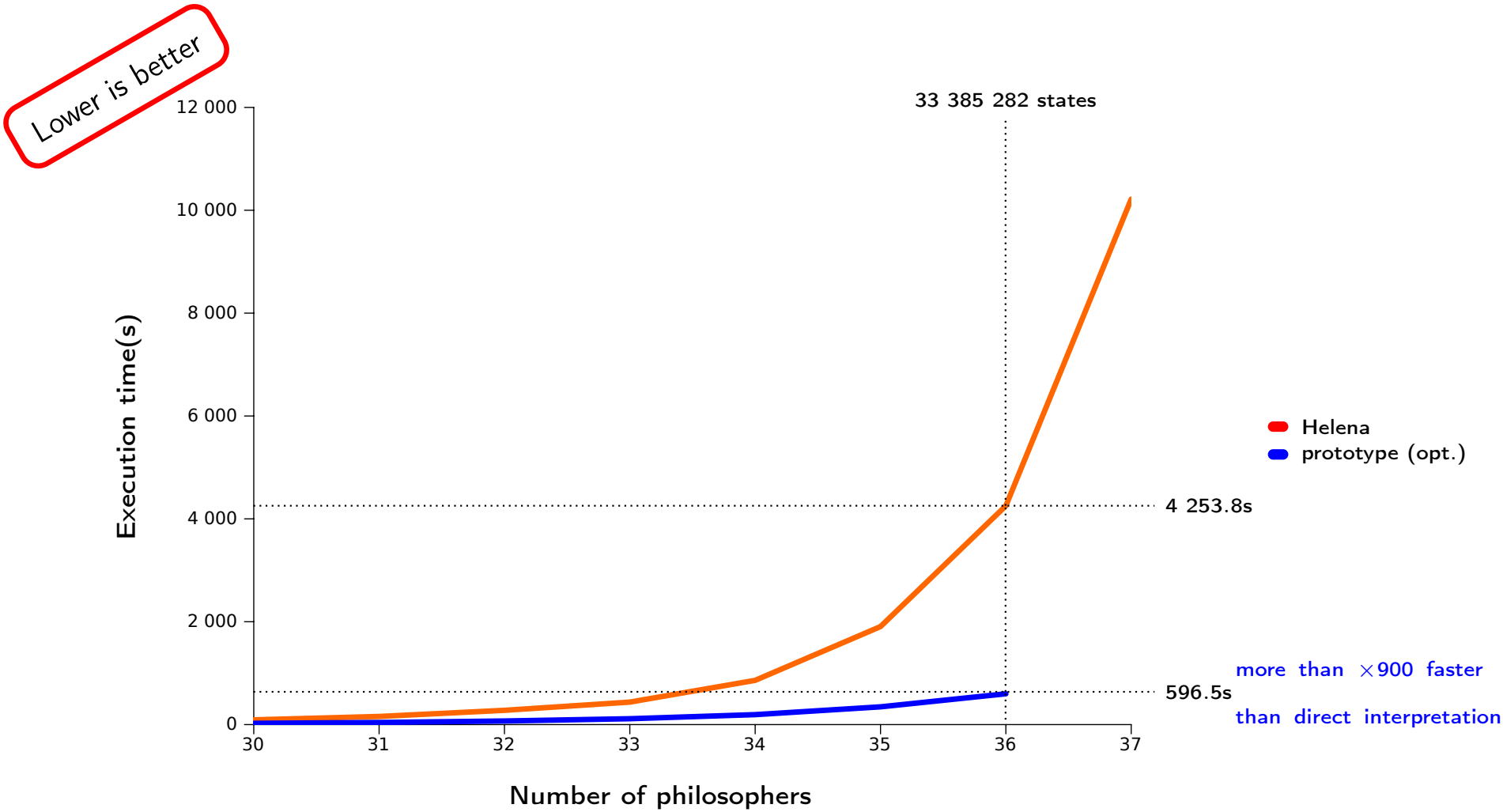
# Dining Philosophers model

## *vs* Helena



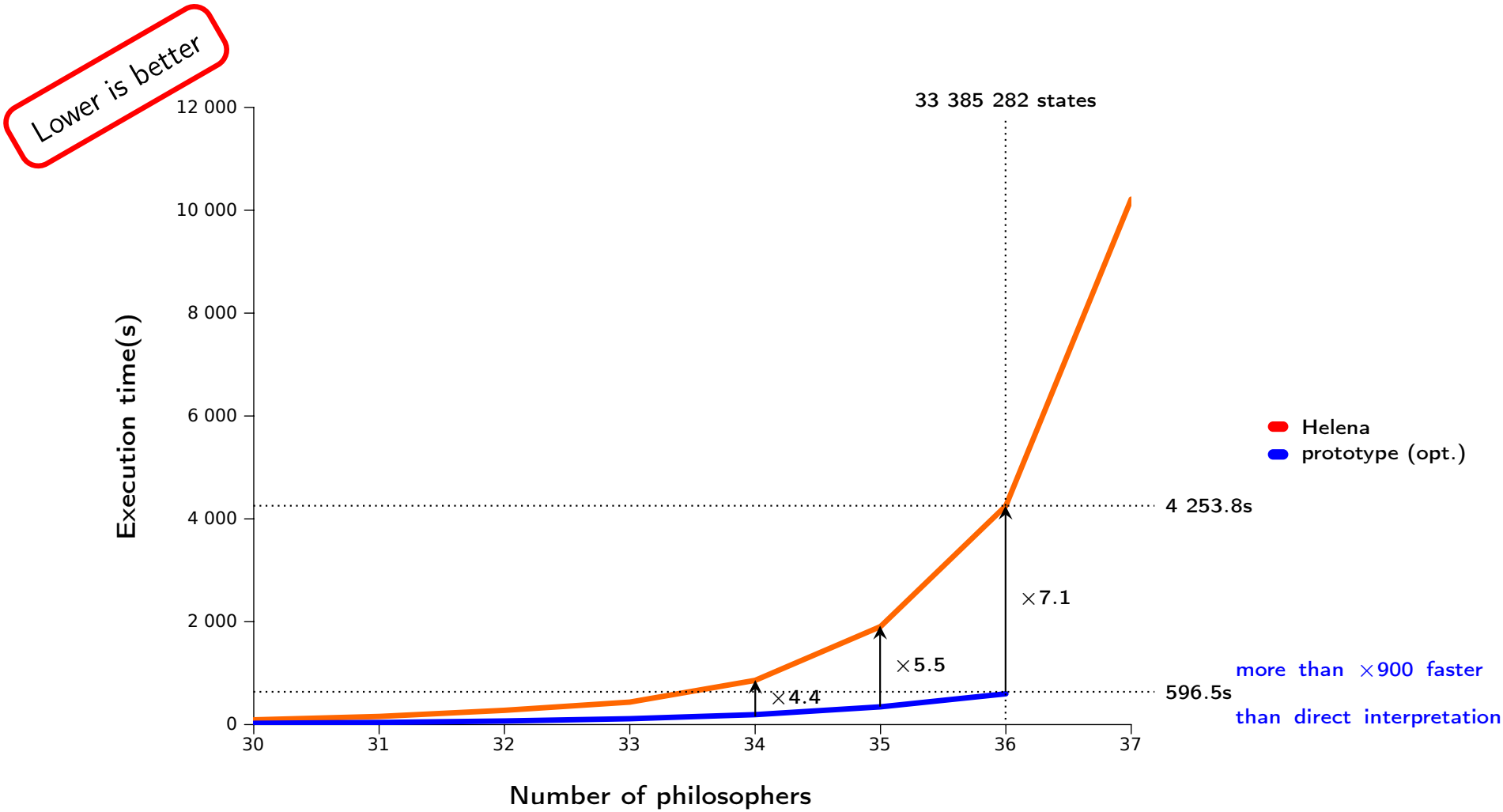
# Dining Philosophers model

## *vs* Helena



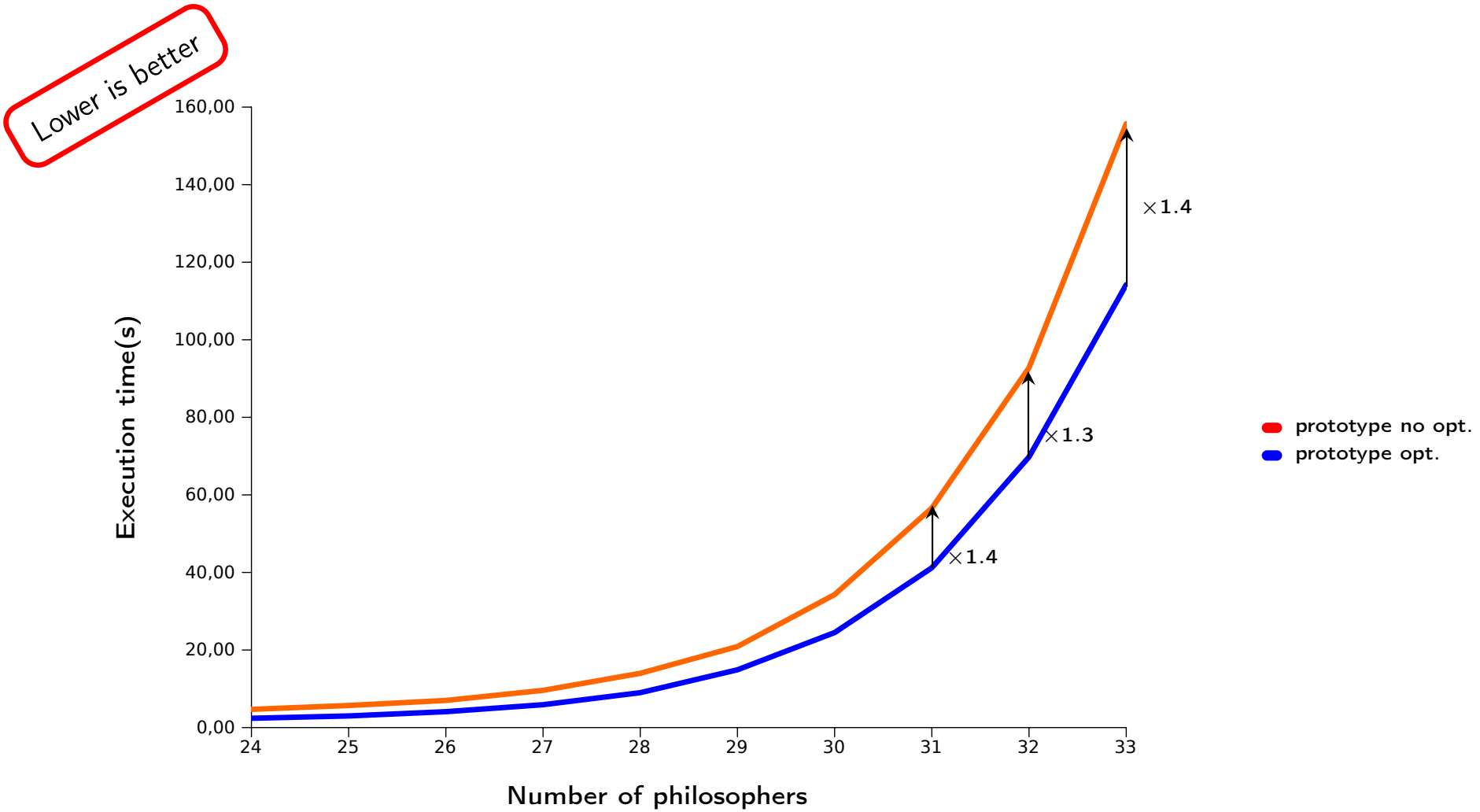
# Dining Philosophers model

## *vs* Helena



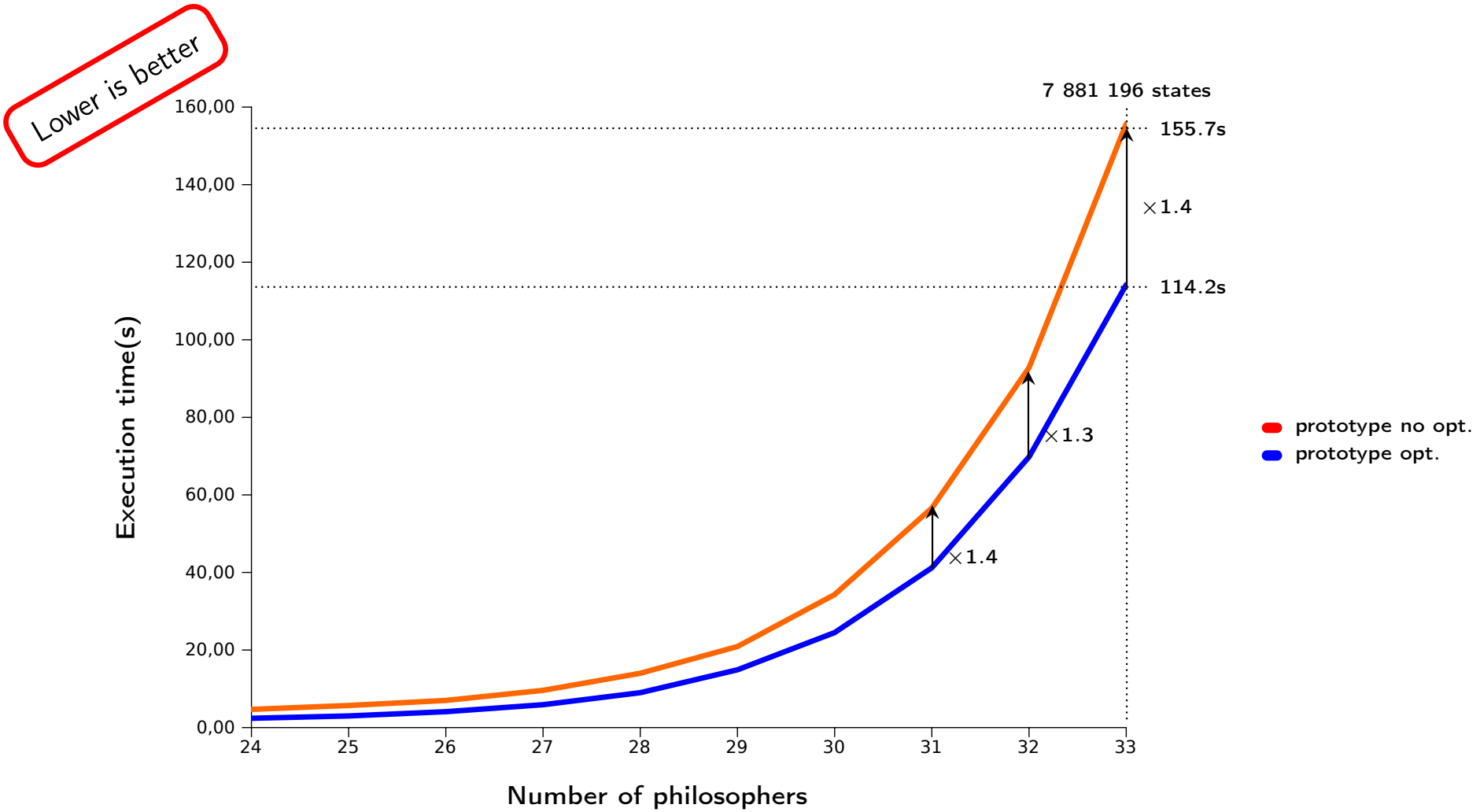
# Dining Philosophers model

## *vs* unoptimised



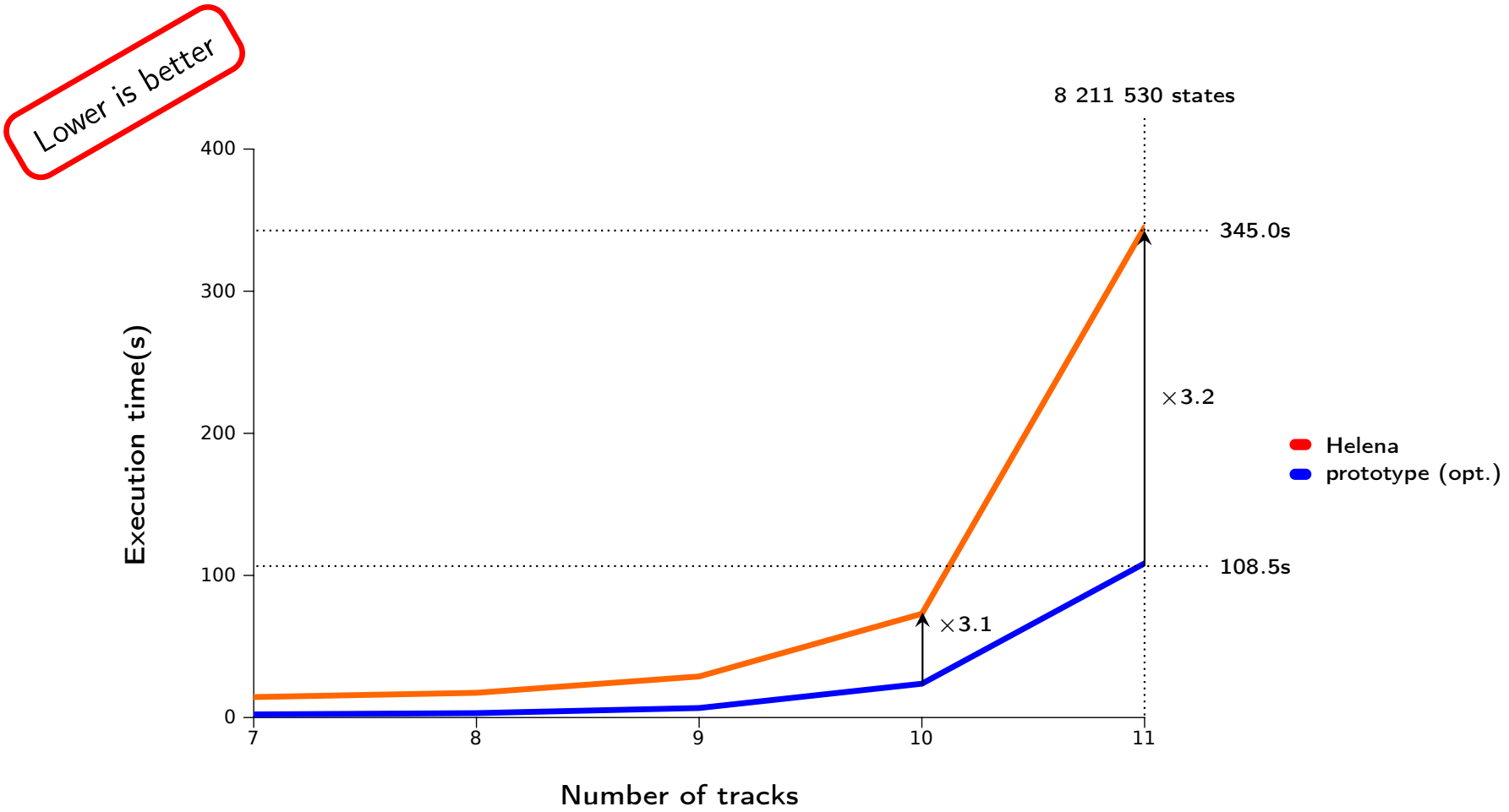
# Dining Philosophers model

## *vs* unoptimised



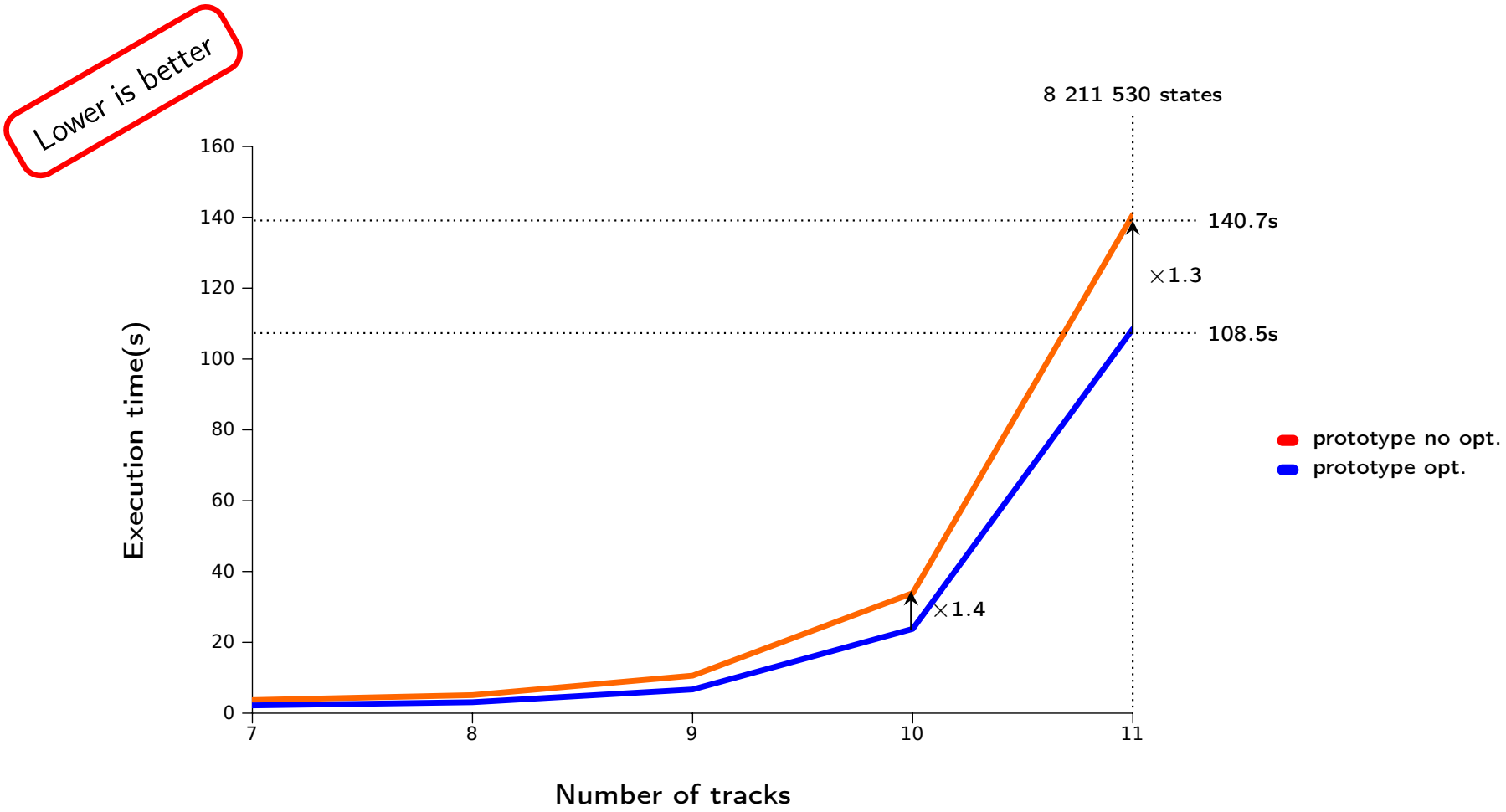
# Railroad crossing model

## *vs* Helena



# Railroad crossing model

## *vs* unoptimised



# A security protocol model

- Needham-Schroeder public key cryptographic protocol
- embeds about 350 lines of Python code Dolev-Yao attacker
  - easy to implement with a full-featured language like Python

# A security protocol model

- Needham-Schroeder public key cryptographic protocol
- embeds about 350 lines of Python code Dolev-Yao attacker
  - easy to implement with a full-featured language like Python

similar model in Helena [Bouroulet 2006]

- SNAKES and Helena run in comparable times
- Python backend 10 times faster than SNAKES
  - we show now: Python *vs* Cython

# A security protocol model

- only 1234 states
- much time spent inside embedded code — cannot be optimised
- compilation is expensive on small models

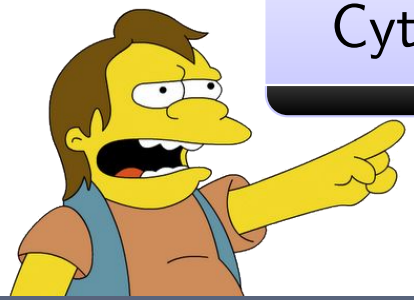
# A security protocol model

- only 1234 states
- much time spent inside embedded code — cannot be optimised
- compilation is expensive on small models
  - our Python backend is faster than the Cython one

# A security protocol model

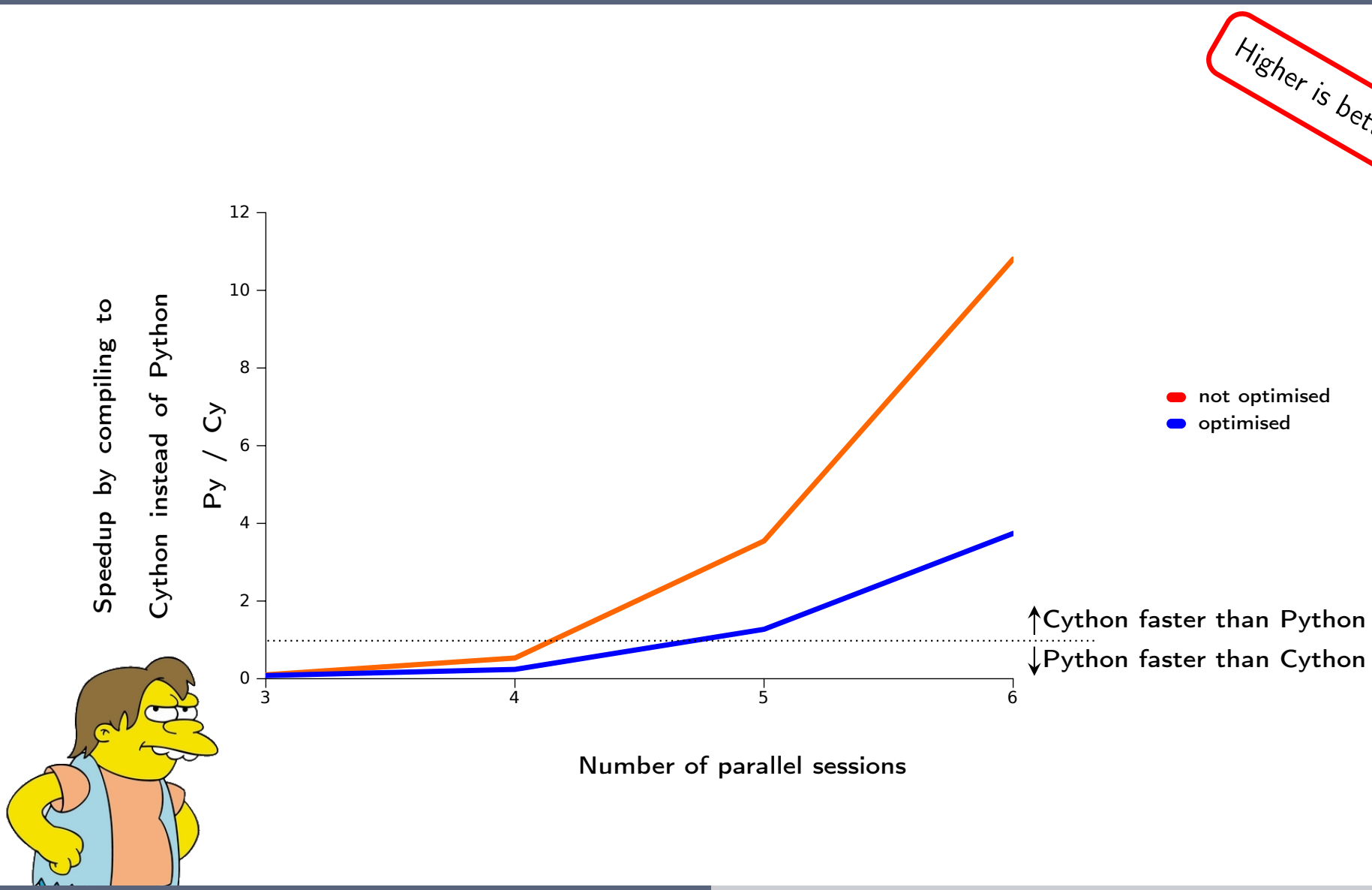
- only 1234 states
- much time spent inside embedded code — cannot be optimised
- compilation is expensive on small models
- our Python backend is faster than the Cython one

| language | not optimised | optimised | speedup |
|---|---|---|---|
| Python | 0.07s 4.93s 5.00s | 0.07s 4.11s 4.18s | 1.20 |
| Cython | 2.36s 3.14s 5.50s | 2.02s 2.82s 4.84s | 1.14 |



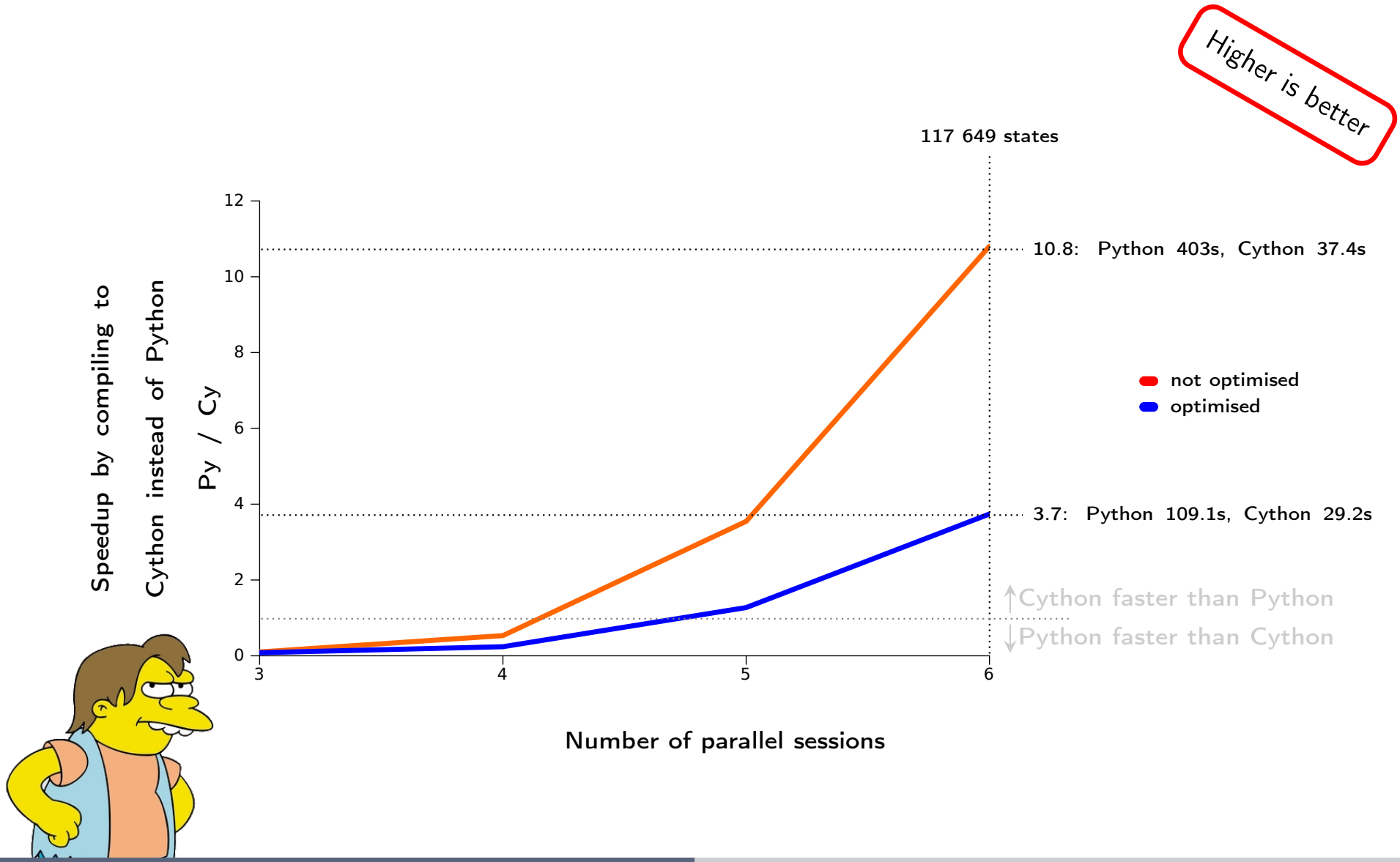
# A security protocol model

## without the Dolev-Yao attacker



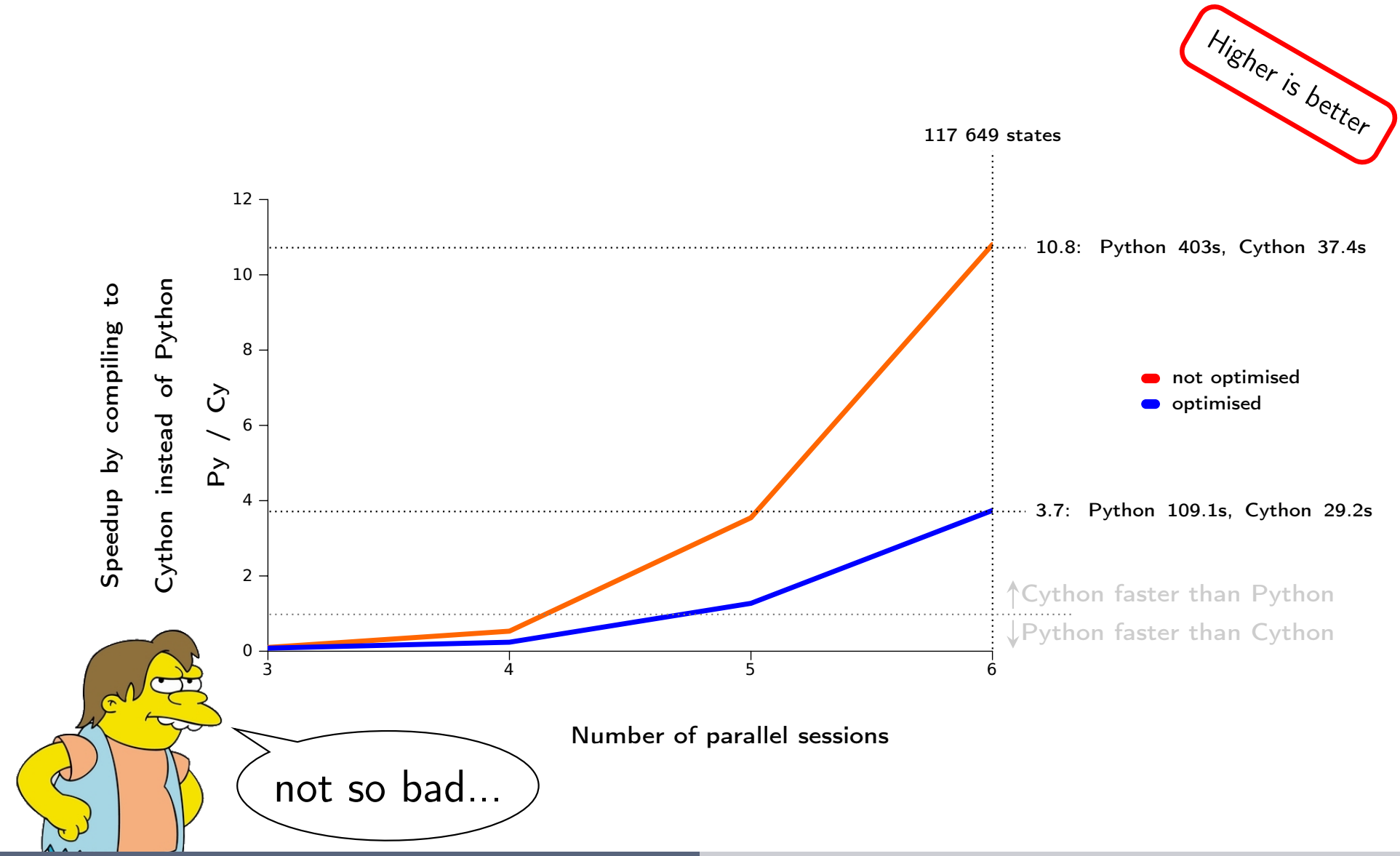
# A security protocol model

## without the Dolev-Yao attacker



# A security protocol model

## without the Dolev-Yao attacker



# Conclusion

- simple model peculiarities — boundaries, place types
  - compile/compute faster — than Helena and than ourselves
  - may be known by construction — *i.e.*, for free
- relevance shown by experimental results

  optimisations: speedup up to $\times 2$ faster — avg. $\times 1.4$

  over $\times 900$ faster than direct interpretation — w.r.t. SNAKES

## What's next?

- more case studies
- integrate into Helena, interface with SPOT
  - real model-checking
  - optimisations: stubborn sets, states compression, ...