Let's Build Provable Multicore Schedulers!

Redha GOUICEM

Whisper team, Sorbonne Universités, Inria, LIP6

Joint work with: Gilles MULLER, Julia LAWALL (Inria, LIP6),
Julien SOPENA (LIP6, UPMC), Baptiste LEPERS, Willy ZWAENPOEL (EPFL),
Jean-Pierre LOZI (UNICE), Nicolas PALIX (Université Grenoble-Alpes)

April 23, 2017

Context

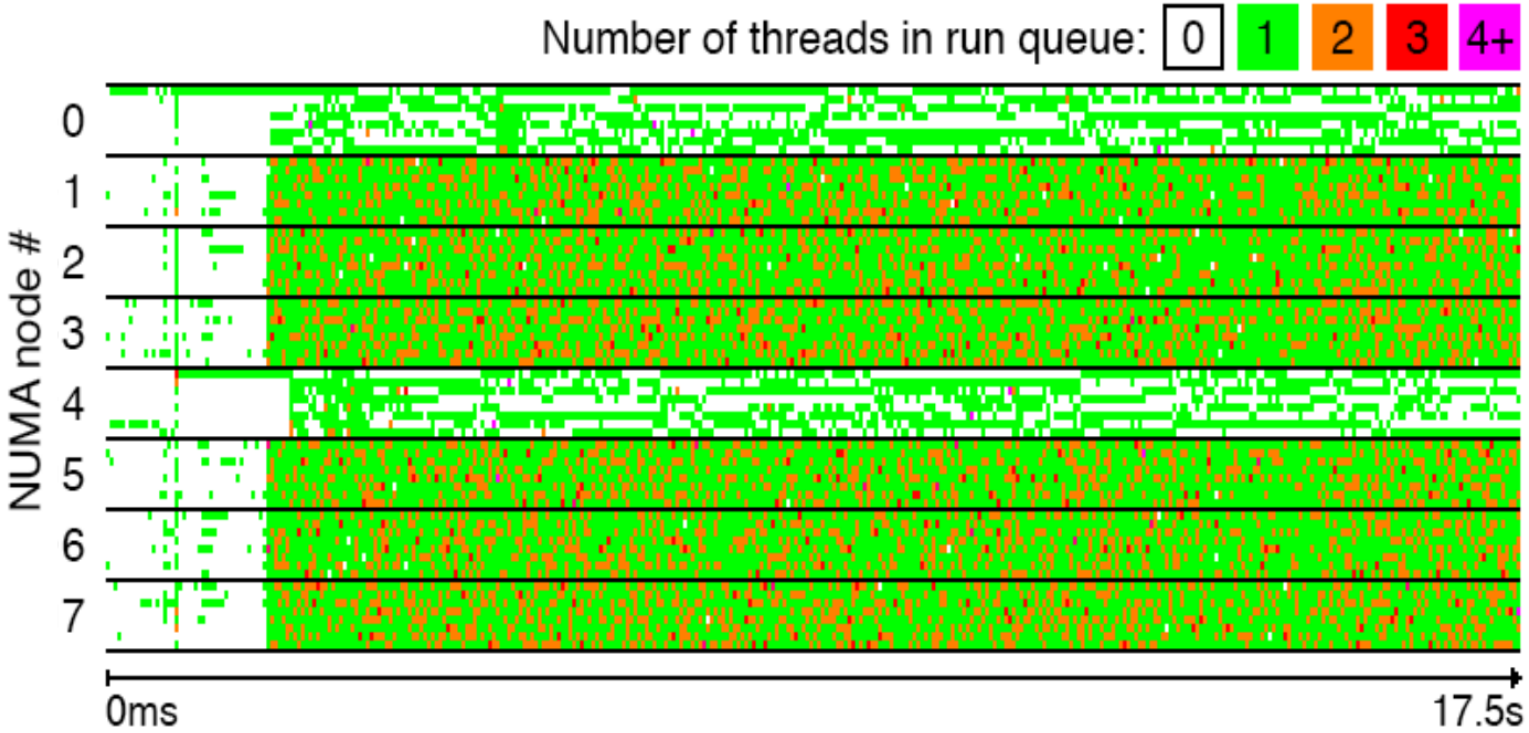
Lozi et al. [EUROSYS'16] found bugs in the Linux scheduler, eg. violation of work conservation property

Work conservation: process waiting to be scheduled $\Rightarrow$ no idle core

Context

Lozi et al. [EUROSYS'16] found bugs in the Linux scheduler, eg. violation of work conservation property

Work conservation: process waiting to be scheduled $\Rightarrow$ no idle core



Some cores are idle while others are overloaded

Problem: Finding bugs

Linux scheduler (CFS) is growing fast in terms of complexity (lines of code, variables, functions, structures)
`kernel/sched/fair.c` codebase history:

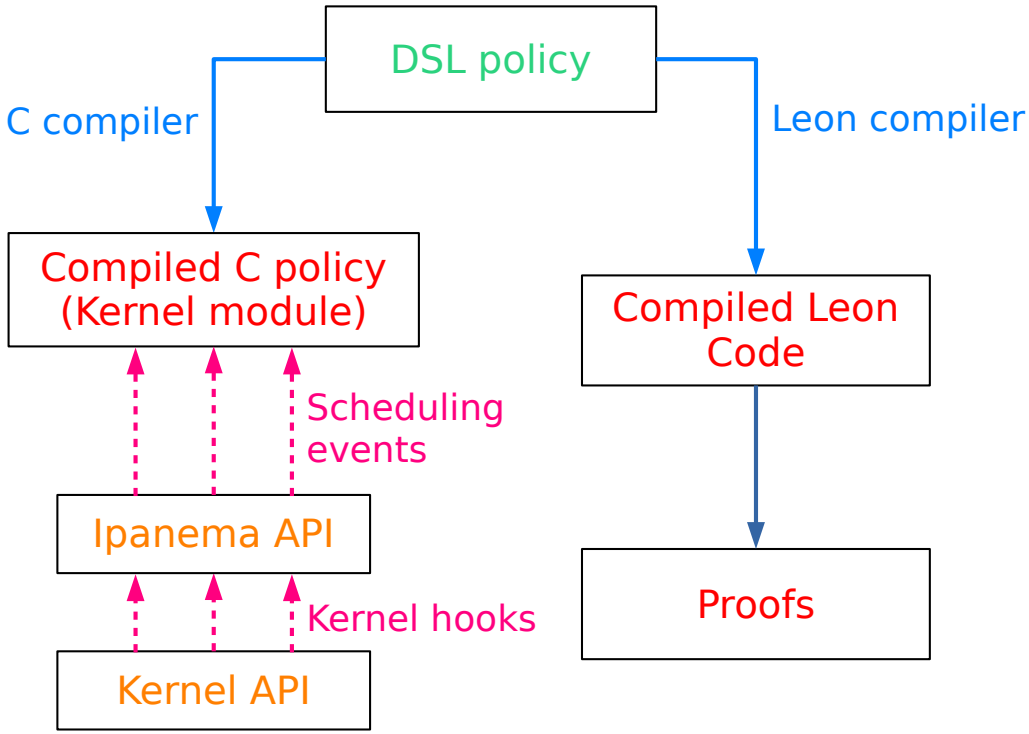
$\Rightarrow$ difficult to maintain and add features

Contribution

Ipanema, a DSL for proved multicore schedulers
Based on Bossa [HASE'05], an event-based DSL for single-core schedulers

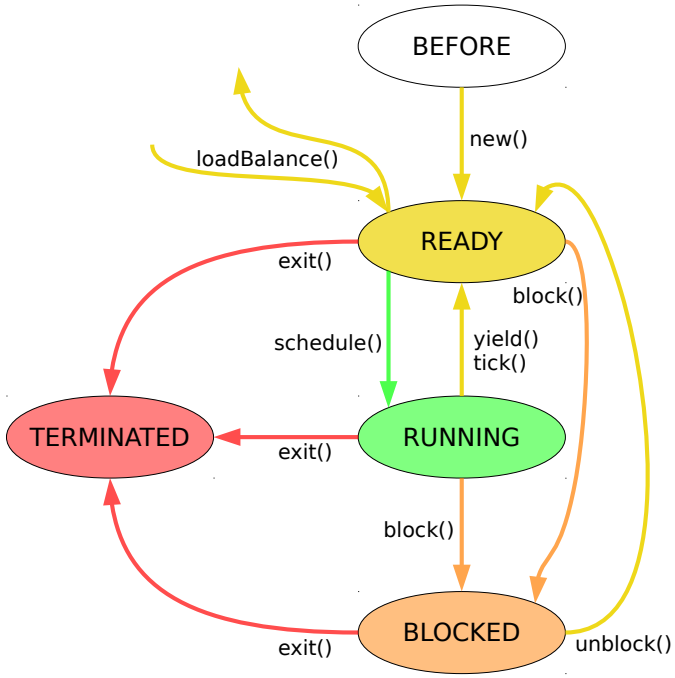
Contribution

Ipanema, a DSL for proved multicore schedulers
Based on Bossa [HASE'05], an event-based DSL for single-core schedulers



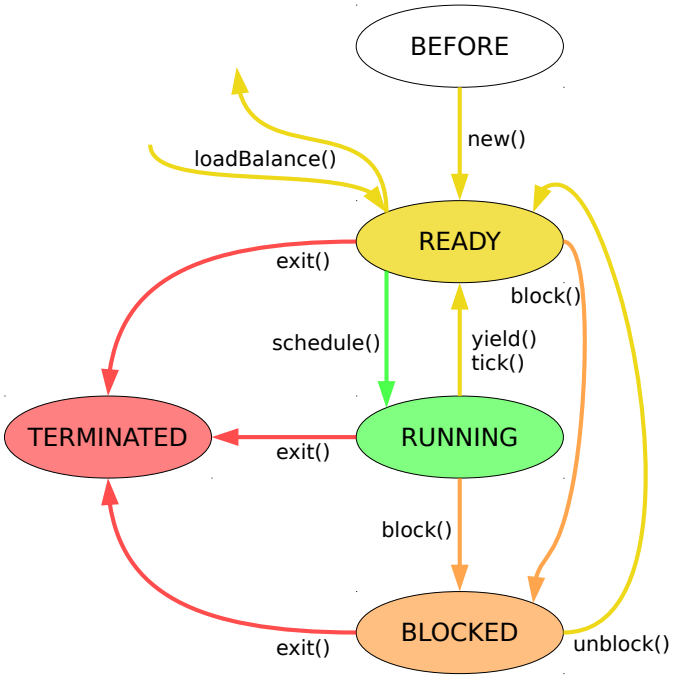
Contribution

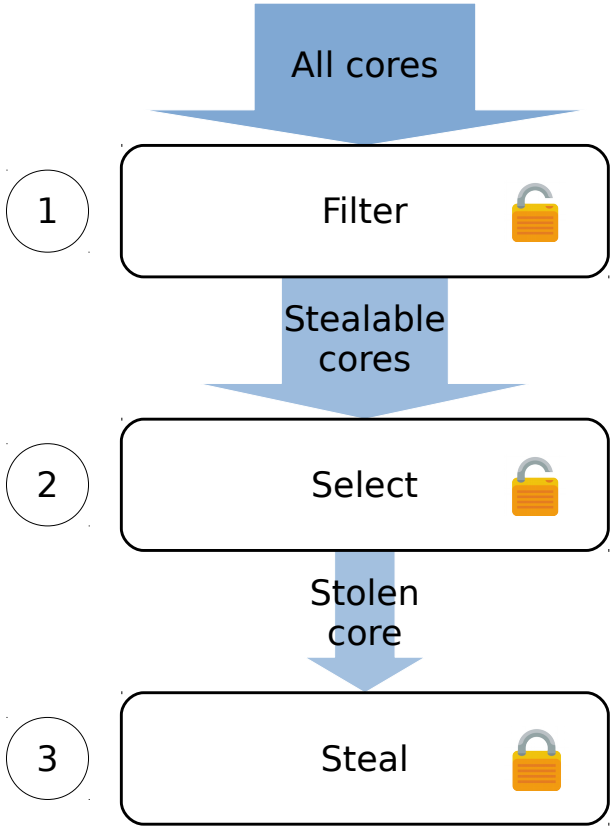
Ipanema = Bossa + load balancing + proofs



Contribution

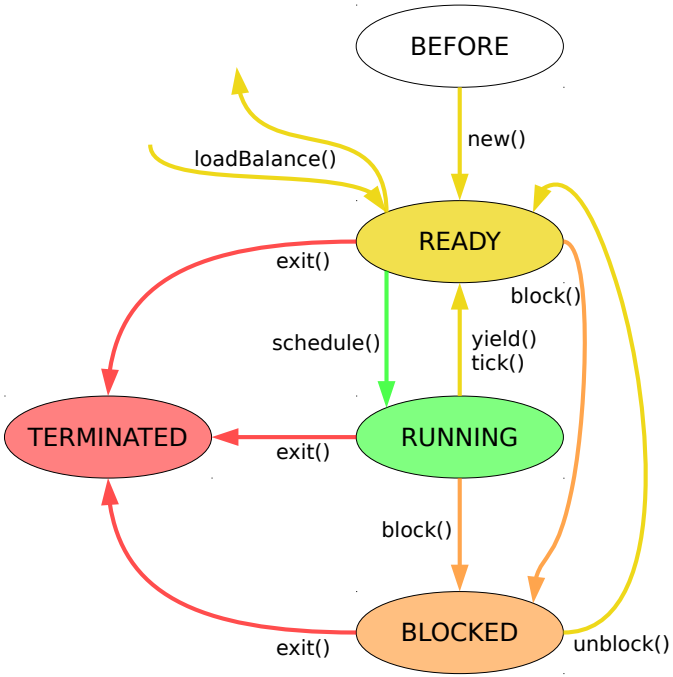
Ipanema = Bossa + load balancing + proofs

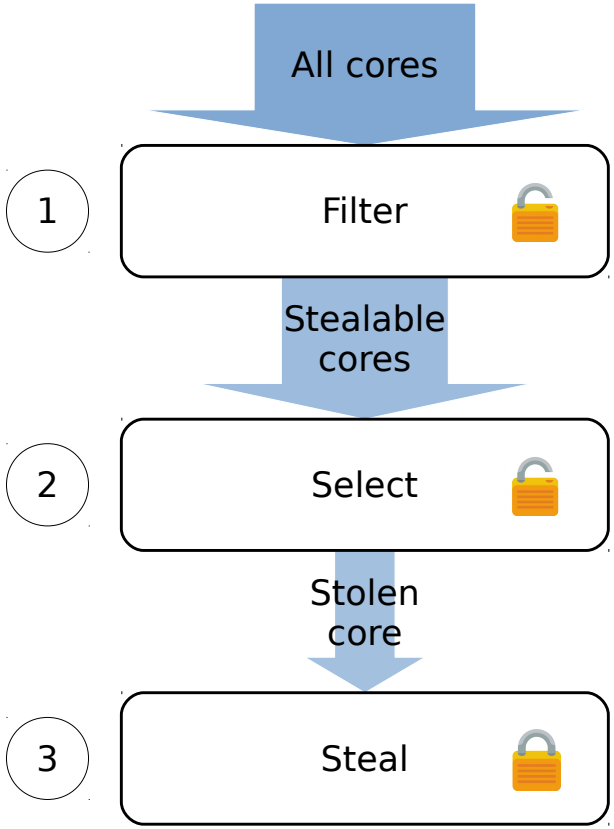




Contribution

Ipanema = Bossa + load balancing + proofs





Evaluation

Safety:

generated kernel modules do not lead to kernel crashes or failures

DSL expressivity:

can we implement "all" scheduling policies ?

Simplicity:

can "anyone" write a scheduling policy ?

Bug detection:

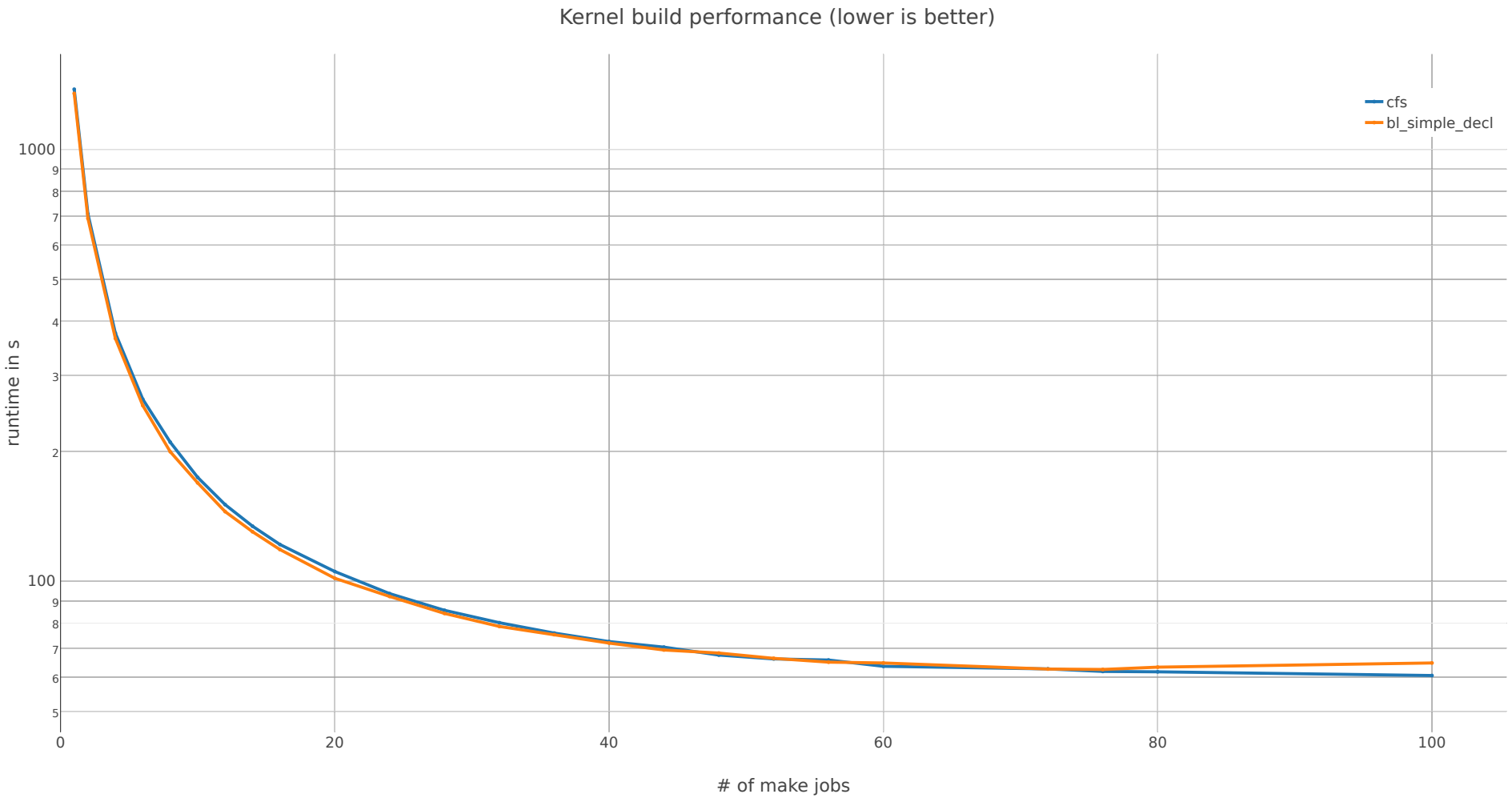
do we prevent bugs that were found/fixed in the CFS ?

Code sample

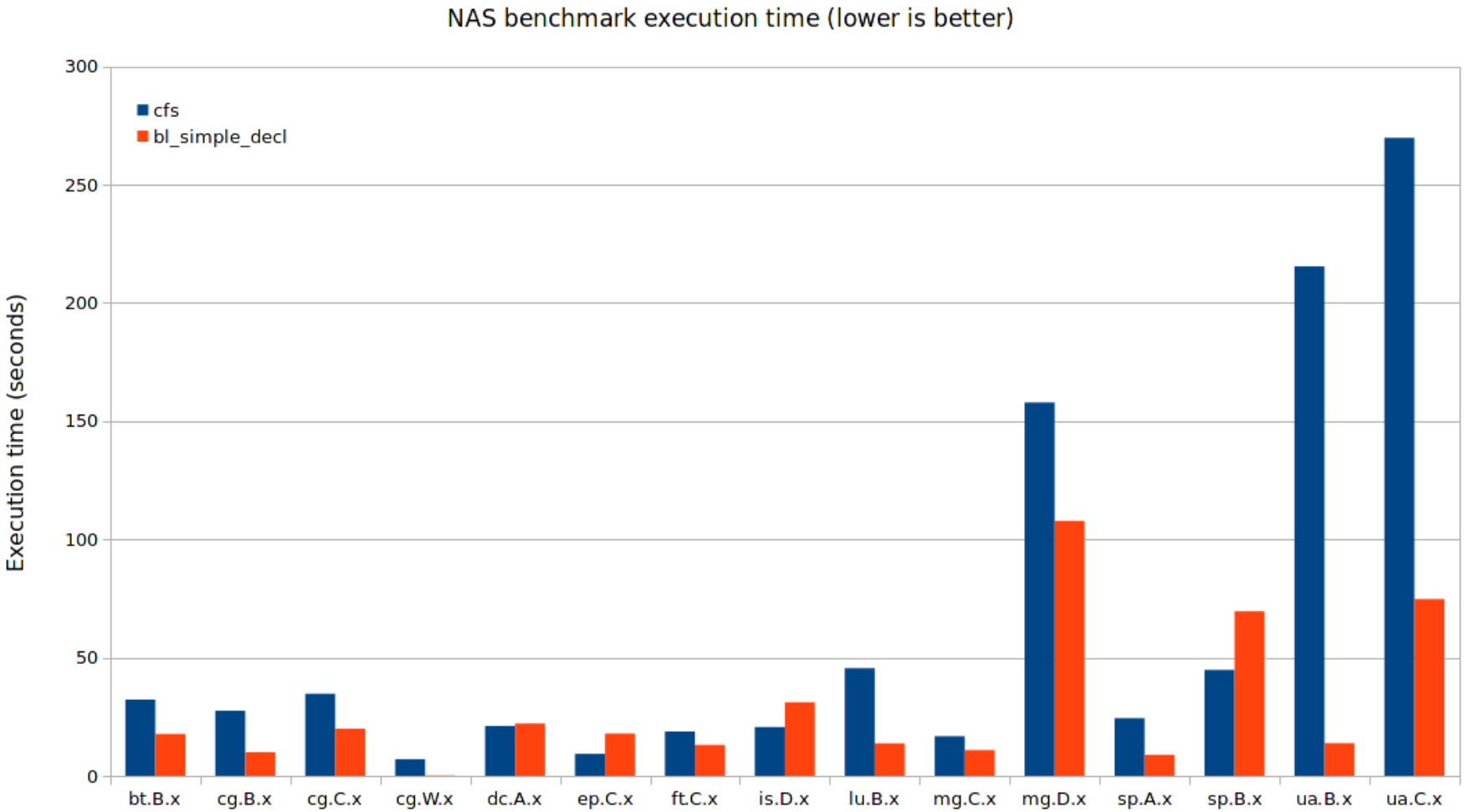
```
thread = {
  int progress, load, last_sched, curr_quanta;
}

core = {
   threads = {
      RUNNING thread current;
      shared READY set<thread> ready:order = {
         lowest progress
      };
      BLOCKED blocked;
      TERMINATED terminated;
   }
   int load = sum(ready.load) +
              valid(current) ? 1024 : 0;
}
```

```
handler (thread_event e) {
   On new {
      core c = first(cores order = { lowest load });
      e.target.load = 0;
      e.target.progress = 0;
      e.target => c.ready;
   }

   On detach {
      e.target => terminated;
   }

   On tick {
      update_progress(e.target);
      if (e.target.curr_quanta >= max_quanta) {
            update_load(e.target, max_quanta);
            e.target => ready;
      }
   }

   On schedule {
      thread t = first(ready);
      t.last_sched = now();
      t.curr_quanta = 0;
      t => current;
   }

   On unblock {
      core c = first(cores order = { lowest load });
      e.target => c.ready;
   }

   On yield {
      update_progress(e.target);
      update_load(e.target,e.target.curr_quanta);
      e.target => ready;
   }

   On block {
      update_progress(e.target);
      update_load(e.target,e.target.curr_quanta);
      e.target => blocked;
   }
}
```

Preliminary results



Preliminary results (2)



Roadmap

What we have:

- Ipanema→C compiler
- Modified Linux-kernel for Ipanema: scheduling events hooks
- Simple policies written in Ipanema run on a modified Linux
- Work conservation proof in Leon for CFS-like balancing policies

What remains to be done:

- Prove properties on policies (fairness, liveness, work conservation)
- Work on the Ipanema→Leon compiler to generate these proofs
- Evaluate

Roadmap

What we have:

- Ipanema→C compiler
- Modified Linux-kernel for Ipanema: scheduling events hooks
- Simple policies written in Ipanema run on a modified Linux
- Work conservation proof in Leon for CFS-like balancing policies

What remains to be done:

- Prove properties on policies (fairness, liveness, work conservation)
- Work on the Ipanema→Leon compiler to generate these proofs
- Evaluate