Modular Pluggable Analyses

Patrick Lam, Viktor Kuncak and

Martin Rinard

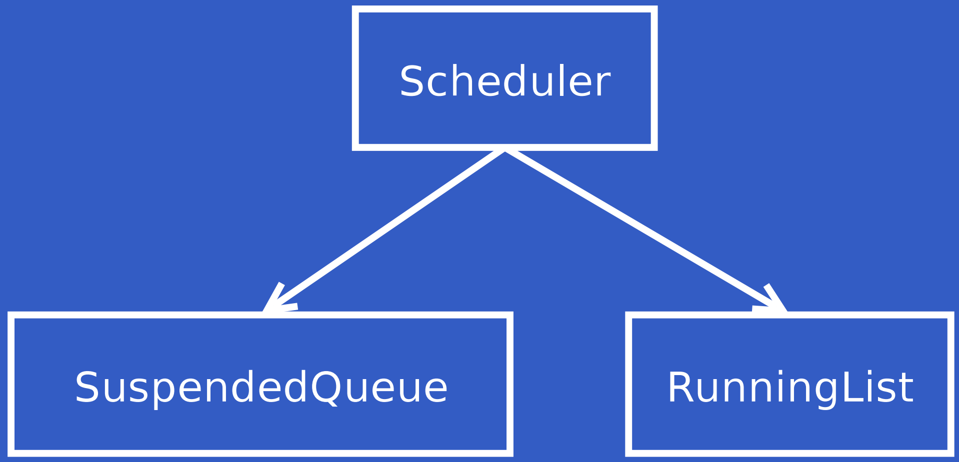
Scheduler

Two kinds of sets of objects:

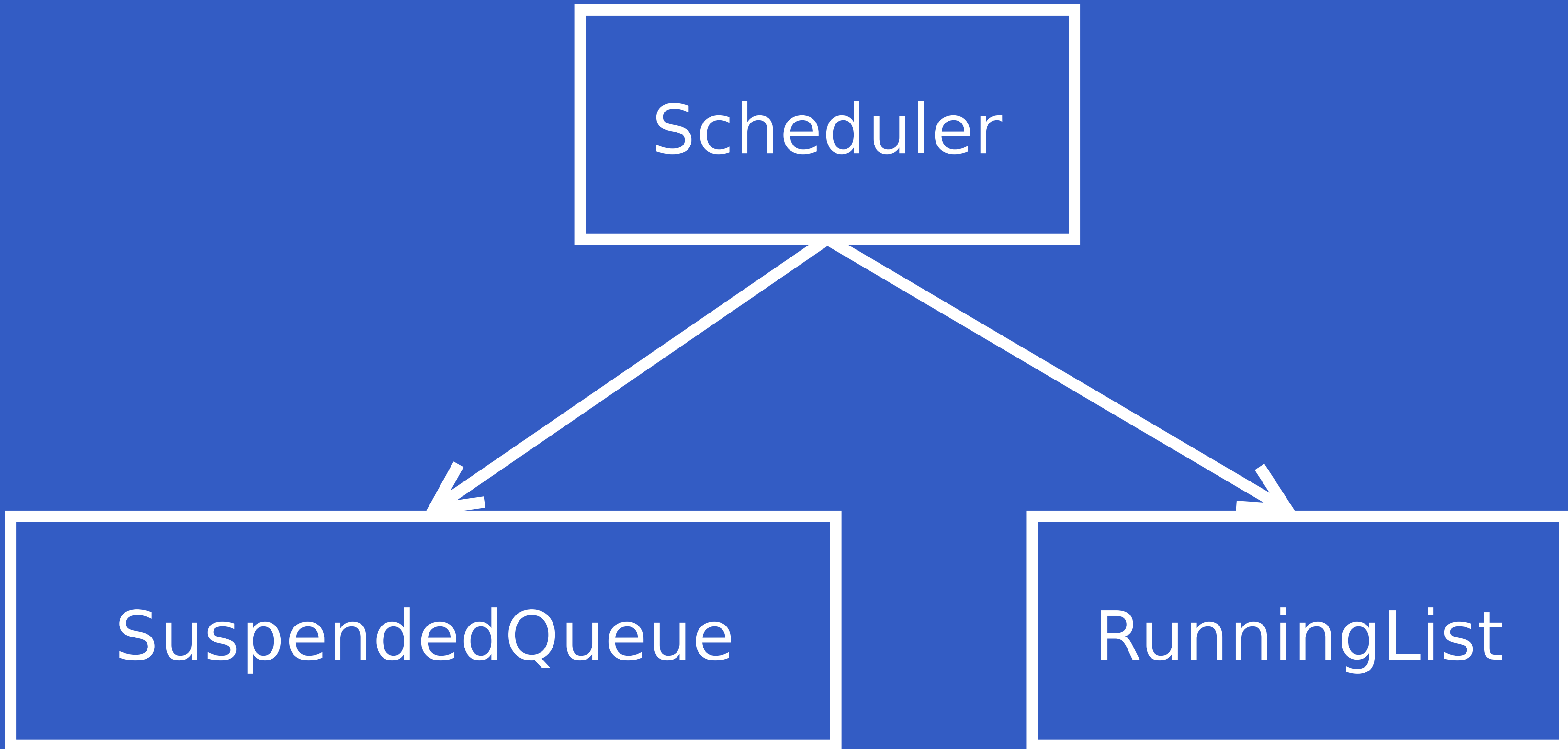
`Running`, `Suspended`

This classification is dynamically-changing: processes are suspended and woken up.

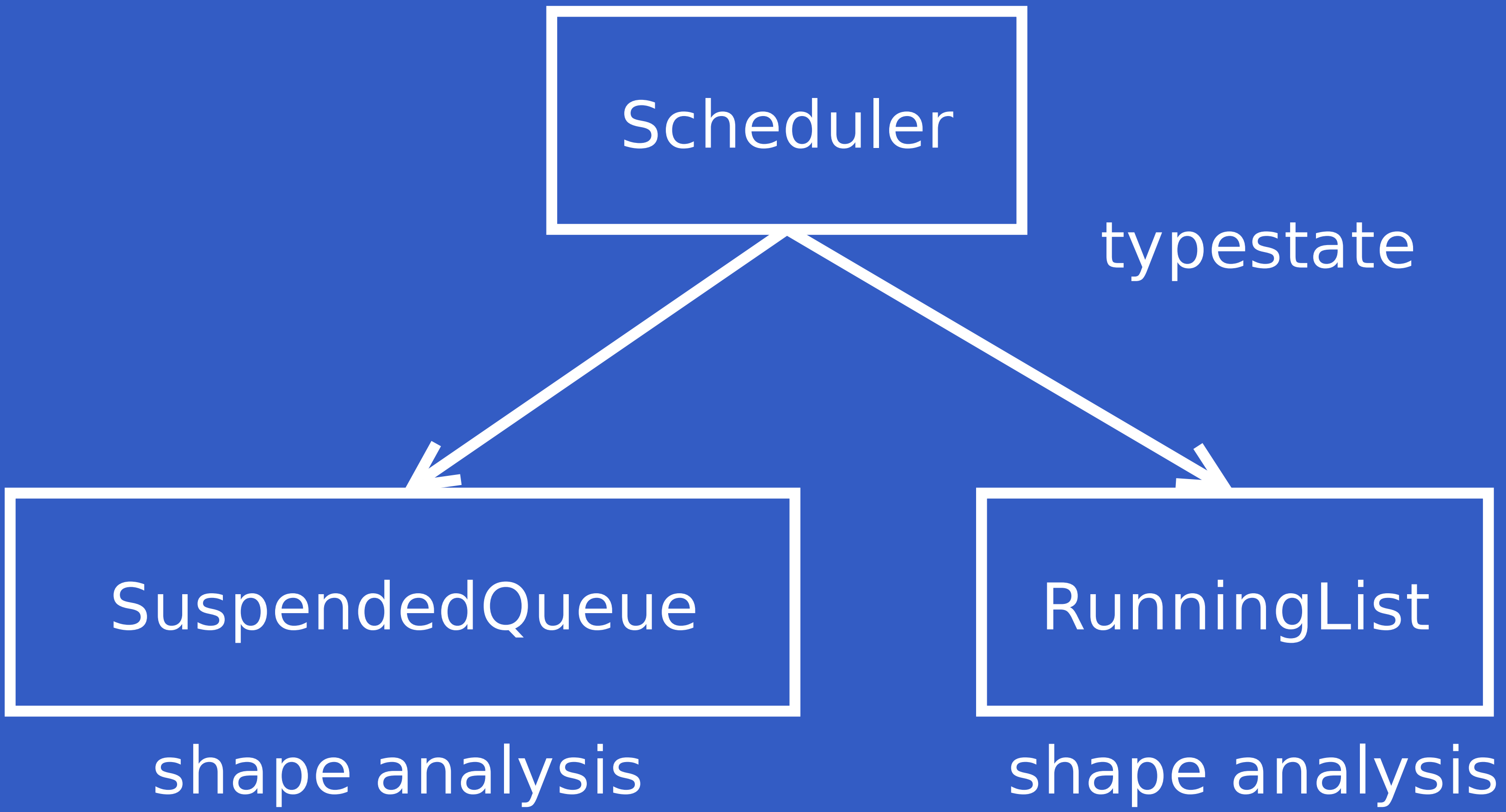
Set membership is determined in two ways: field values and pointer reachability. Our analysis checks that sets coincide.



Example



Example


Scheduler
typestate
SuspendedQueue
RunningList
shape analysis
shape analysis

Challenge: Module Interaction

Objects may be shared between modules.

How do we know that other parts of program don't break our invariants, especially through aliases?

- Allow modules to share objects, but ensure that each module refers to disjoint fields.

```
impl module SuspendedQueue {
  format Process { next:Process; priority:int; } }

impl module Scheduler {
  format Process { status:int; } }
```

Challenge: Effects

Modules have effects; must reason about them.

We use a uniform specification language: preconditions and postconditions in first-order logic on sets.

```
proc suspend(p:Process; priority:int)
  requires p in Running & card(p)=1
  modifies Running, Suspended
  ensures Suspended' = Suspended + p &
    Running' = Running - p;
```

Plugin: Flags

Use fields to determine set membership.

```
abst module Scheduler {
 use plugin "flags";
 Running = {x:Process | "x.status=2"};
 Suspended = {x:Process | "x.status=1"};
}
```

Uses a dataflow analysis over first-order boolean formulas.

Plugin: Graph Reachability

```
abst module SuspendedQueue {
  use plugin "PALE";
  InQueue = {x:Process | "root<next*>x"};

  invariant "type Process = {
     data next:Process;
     }";
  invariant "data root:Process;";
}
```

Uses MSOL over trees and loop invariants.

Experience

We have implemented a prototype system and tested computational patterns inspired by:

- compiler transformations
- CTAS
- water

Currently working on bigger examples.

Conclusion

Most discovered errors were specification errors.

Found some errors in the implementation.

At one point, we inadvertently changed the abstraction function and only partially updated the code. The tool found this error.