Automatic Parallelisation of Rust Programs at Compile Time

Why?

- Processors gain more cores
- Writing parallel code is more difficult than sequential code
- Already got many sequential programs that we do not want to rewrite.
- Want a way to automatically convert a sequential program to take advantage of the additional cores

Why Rust?

- Rust has a unique memory management system.
- Each variable has ownership information and a lifetime.
- Access to the variable can be given by moving or borrowed the variable.
- “Guarantees thread safety”

What did I do?

- Created two rust compiler plugin
  - Linter plugin: Analyses function with macros expanded
  - Syntax extension plugin: Modifies function to run parts in parallel
- Parallelising Steps
  - Deconstruction
  - Dependency Analysis
  - Scheduling
  - Reconstruction
- Focused on “safe” statement level parallelisations

Simple Example

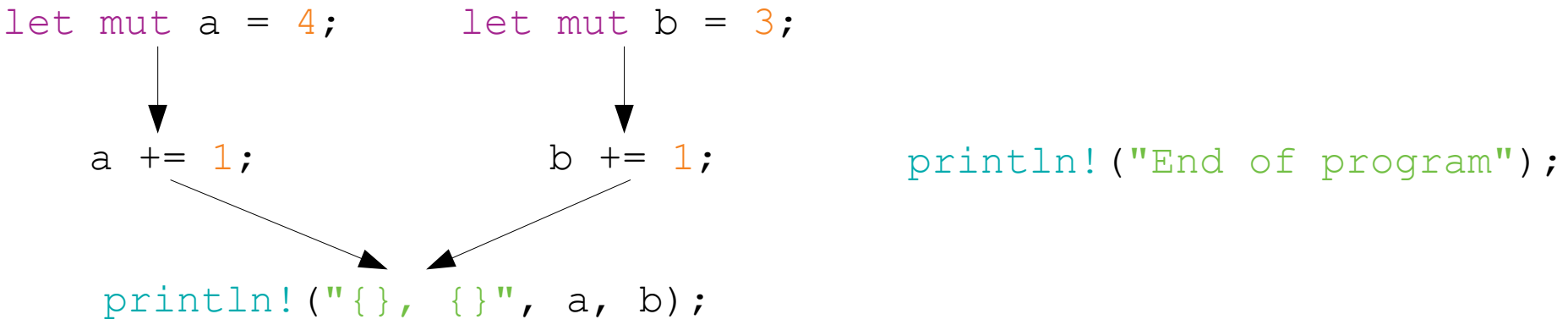
```
fn main() {
    let mut a = 4;
    let mut b = 3;
    a += 1;
    b += 1;
    println!("{}, {}", a, b);
    println!("End of program");
}
```

Deconstructor

```
fn main() {
    let mut a = 4;
    let mut b = 3;
    a += 1;
    b += 1;
    println!("{}, {}", a, b);
    println!("End of program");
}
```

```
In: {},      Out: {a}
In: {},      Out: {b}
In: {a},     Out: {a}
In: {b},     Out: {b}
In: {a, b}, Out: {a, b}
In: {},      Out: {}
```

Dependency Analysis



Scheduler

```
println!("End of program");
```

```
let mut b = 3;
b += 1;
// send b
```

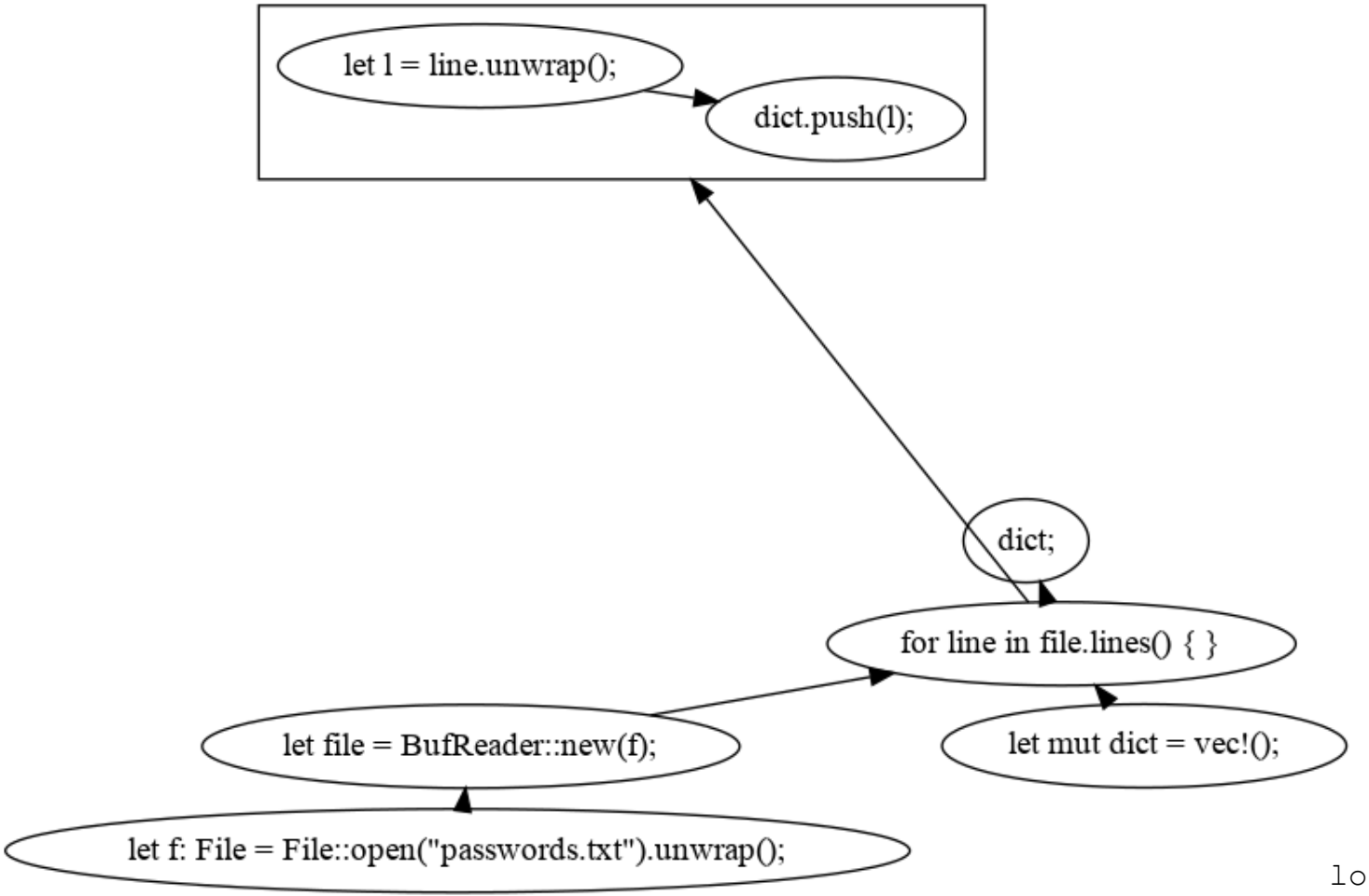
```
let mut a = 4;
a += 1;
// wait for b
println!("{}, {}", a, b);
```

```
fn main() {
    let (syncline_137_144_149_174_b_send,
         syncline_137_144_149_174_b_receive) = std::sync::mpsc::channel();
    let thread_179_206 =
        std::thread::spawn(move || { println!("End of program"); });
    let thread_106_120 =
        std::thread::spawn(move ||
                                {
                                    let mut b = 3;
                                    let return_value =
                                        {
                                            b += 1;
                                            syncline_137_144_149_174_b_send.send((b,)).unwrap()
                                        };
                                    return_value
                                });
    let return_value =
        {
            let mut a = 4;
            let return_value =
                {
                    a += 1;
                    let return_value =
                        {
                            let (b,) =
                                syncline_137_144_149_174_b_receive.recv().unwrap();
                            println!("{}, {}" , a , b);
                        };
                    return_value
                };
            return_value
        };
    thread_179_206.join().unwrap();
    thread_106_120.join().unwrap();
    return_value
}
```

What could be expanded

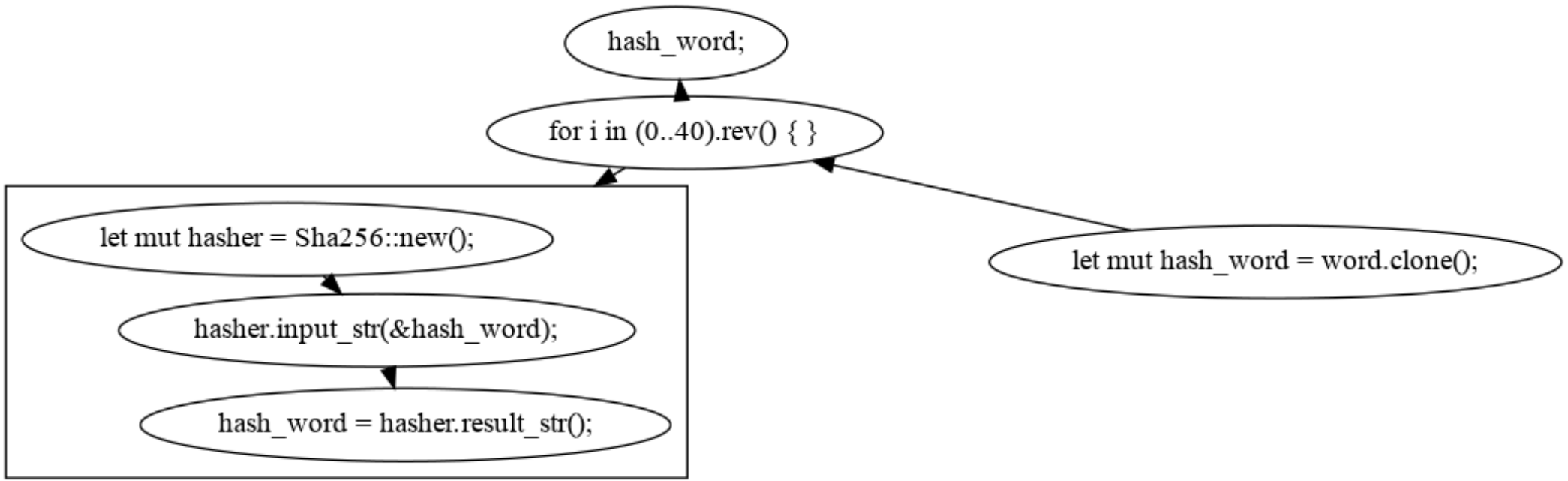
- Performance analysis
- Unsafe blocks
- External functions that rely on a global state
- Already threaded code
- Loops
- Limit number of threads
- Check for Send Trait

Password Cracker



`loadDictionary()`

Password Cracker



Password Cracker