Rust concurrency

Stefan Schrijvers
g.schrijvers@student.science.ru.nl

Rowan Goemans
r.goemans@student.science.ru.nl

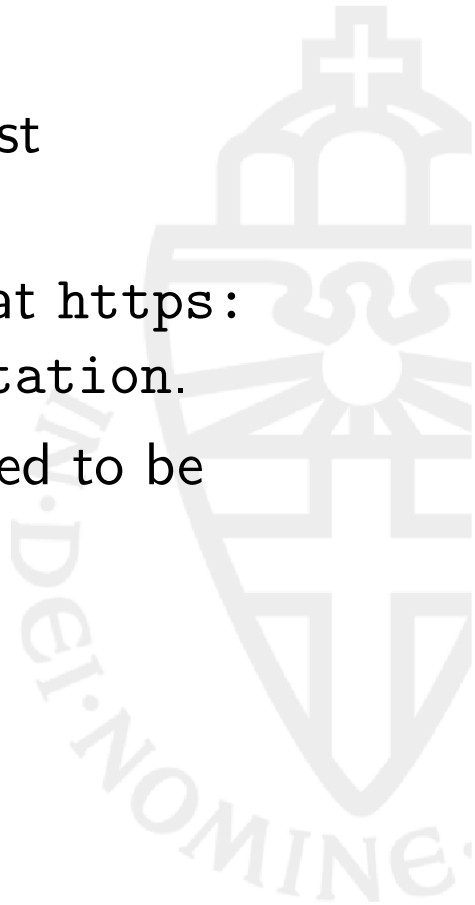
Radboud University Nijmegen

Type Theory & Coq
26th November 2020

Outline

Rust book

- We will cover Section 16 (concurrency) of the Rust Programming Language book [**?**]
- Presentation and all code examples are available at `https://github.com/rowanG077/TypeTheoryPresentation`.
- Demonstration during the presentation are designed to be followed along.



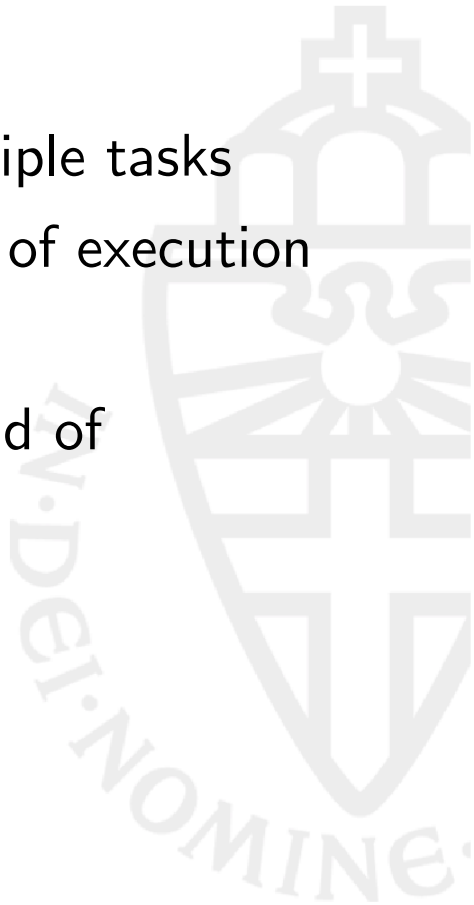
Follow along requirements

- Requirements: Cloned repository, Rust toolchain and a C compiler
  - Clone repo: `git clone git@github.com:rowanG077/TypeTheoryPresentation.git`
  - Install Rust on Linux/OSX: `curl --proto '=https' --tlsv1.2 -sSf https://sh.rustup.rs | sh`
  - Install on Windows, Download and run: `https://static.rust-lang.org/rustup/dist/i686-pc-windows-gnu/rustup-init.exe`
  - Follow onscreen procedure.

The what and why of concurrency

What is concurrency

- Definition: An application makes progress in multiple tasks
- Concrete: An application has more than 1 thread of execution
- Weaker notion of parallelism.
- Parallelism: An application has more than 1 thread of execution executing simultaneously.



Concurrency vs Parallelism

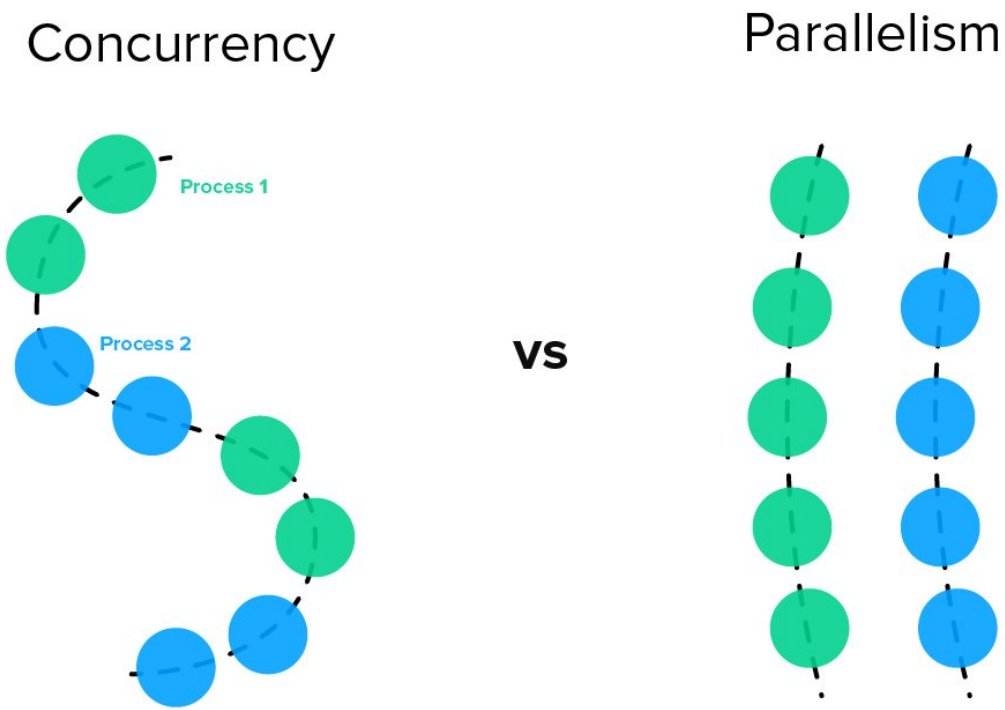
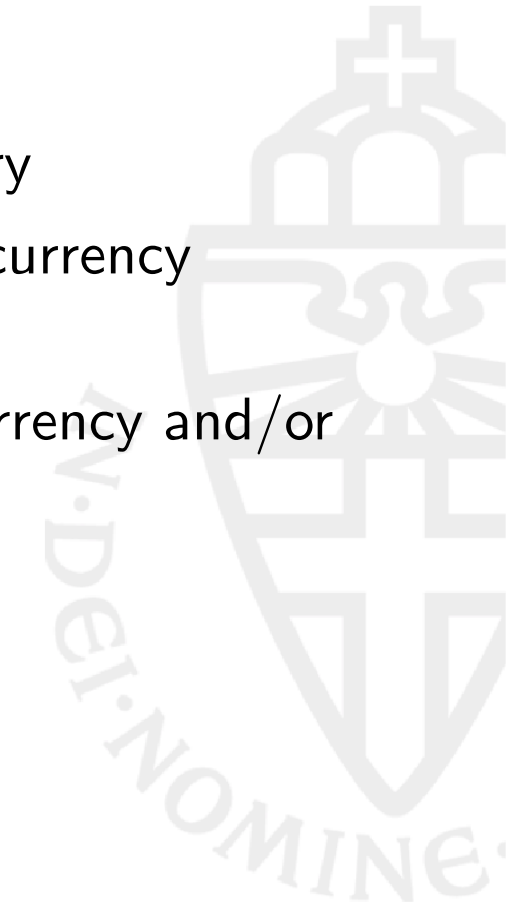


Figure: Concurrency vs Parallelism [**?**]

Concurrency vs Parallelism

- Sloppy usage of the term concurrency by industry
- Usage of concurrency usually really means "concurrency and/or Parallelism"
- When encountering "concurrency", read "concurrency and/or parallelism" during this presentation.



Why Concurrency?

- Modern computers have many cores
- Without concurrency only a single core can be used by an application
- We want to make full use of all resources of a modern computer
- Thus we need concurrency

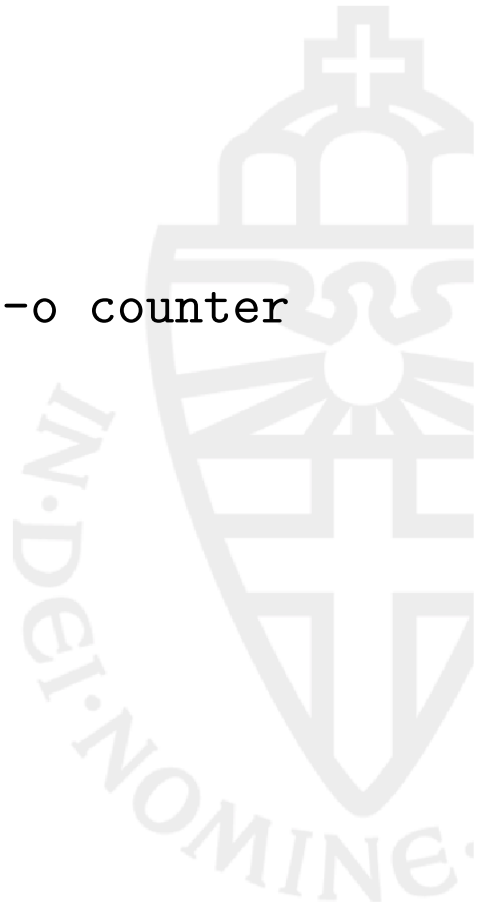
Examples of problems with concurrency

Common issues

- Race conditions
- Deadlocks
- Difficult to use correctly
- Incredibly difficult to debug

Example in C

- Demo
  - File `c-samples/counter.c`
  - Compile with `gcc -O0 -lpthread counter.c -o counter`
- Example contains race conditions
- Can give a different result each run



Common solutions and their pitfalls

- Guards (e.g. Mutex, Semaphore, Spinlocks)
- Difficult to know whether solution is correct
- Only observe result at run-time
- Rust's ownership model and type system allow for catching some concurrency issues at compile time

Recap of Rust ownership model

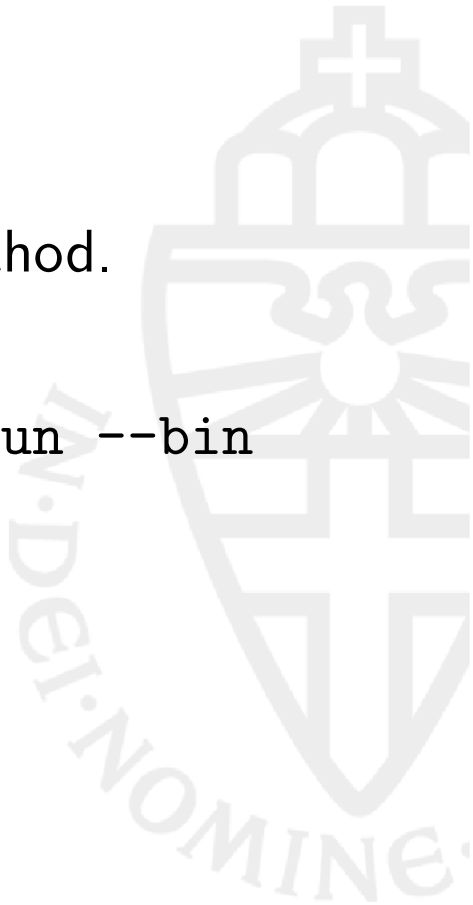
Ownership

- Rust compiler keeps track of who owns a variable
- Variable can only be used by the owner or someone who "borrows" the variable.
- Allows Rust to have static memory safety guarantees with no garbage collector.
- Short demo
  - Source is under: `rust-samples/src/borrowing.rs`
  - To compile & run: `cd rust-samples && cargo run --bin borrowing`.

Basic thread usage in Rust

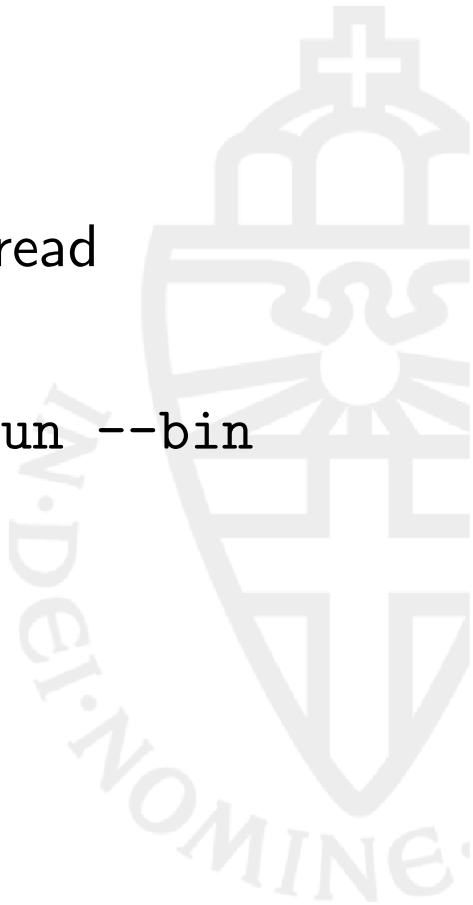
Usage

- Created with `thread::spawn`
- Can wait for a thread to finish with the `join` method.
- File `rust-samples/src/thread_basic.rs`
- Try to compile : `cd rust-samples && cargo run --bin thread_basic`



Ownership

- Previous example only used thread-local data
- More common use is to use data from another thread
- File `rust-samples/src/ownership_thread.rs`
- Try to compile : `cd rust-samples && cargo run --bin ownership_thread`



Ownership

- `v` is captured in the closure
- Closure are anonymous functions that can capture environment
- Rust infers how to capture `v` by its usage
- `println` only needs a reference
- Rust is conservative and only borrows `v`
- This is a problem since the thread does not know how long `v` is valid

Ownership

- We can force ownership by using `move`
- New thread now owns `v`
- This guarantees other threads can't modify `v`

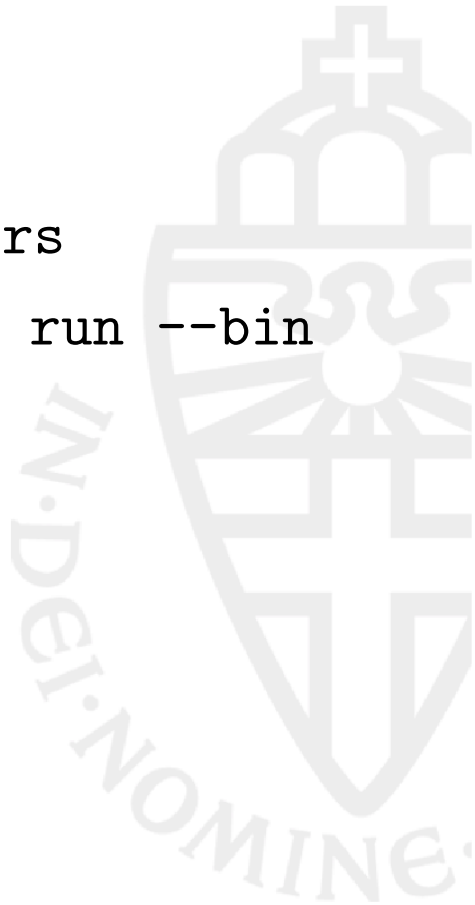
Message passing between threads

Problem

- 'Share memory by communicating instead of communicating by sharing memory'
- Possible solution: Have communication channel between threads.
- Implemented by rust via `std::sync::mpsc` module.
- `std::sync::mpsc` is a multi-producer, single-consumer mechanism
- Allow a thread to send a message over a channel to different thread.
- Moves ownership to a variable in different thread.

Demo

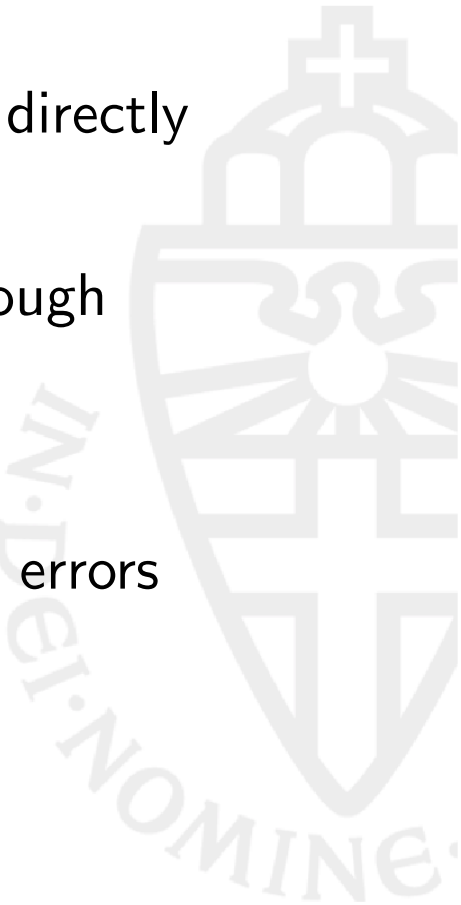
- Source is under: `rust-samples/src/channels.rs`
- To compile & run: `cd rust-samples && cargo run --bin channels`.



Shared state concurrency

Alternative communication method

- Alternative is to communicate by sharing memory directly
- This is what the C example did
- Unlike message passing single ownership is not enough
- Shared memory is similar to multiple ownership
- Results in additional complexity
- Rust's type system and ownership can help reduce errors

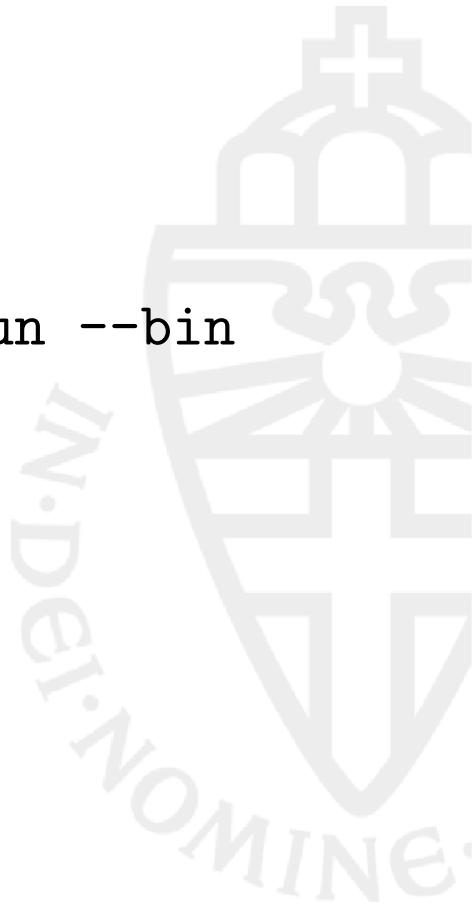


C example revisited

- Earlier example in C we had a race condition
- The example used shared memory
- Create a safe Rust version

Demo

- File `rust-samples/src/shmem_initial.rs`
- Try to compile: `cd rust-samples && cargo run --bin shmem_initial.rs`



C example revisited

- The issue is that ownership was already moved in a previous iteration
- Solution requires multiple ownership
- Rust provides various mechanisms for multiple ownership
- One solution uses reference counting, implemented by `Rc<T>`
- Attempt 2, File `rust-samples/src/shmem_rc.rs`
- Try to compile: `cd rust-samples && cargo run --bin shmem_rc.rs`

Traits

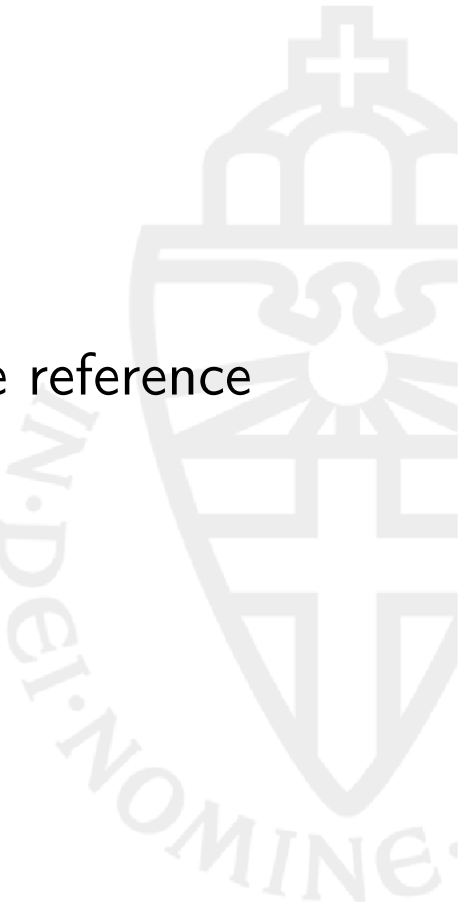
- Traits are a way to share features between types
- Provide ad-hoc polymorphism
- File `rust-samples/src/traits.rs`
- Try to compile: `cd rust-samples && cargo run --bin traits.rs`

Traits

- There are two traits for concurrency concepts embedded in Rust
- `Send`: Ownership can be transferred to another thread.
- `Sync`: Indicates that the type can be referenced from multiple threads
- `Rc<T>` does not have the `Send` trait
- This is because it doesn't guarantee that the updates to the reference count are atomic

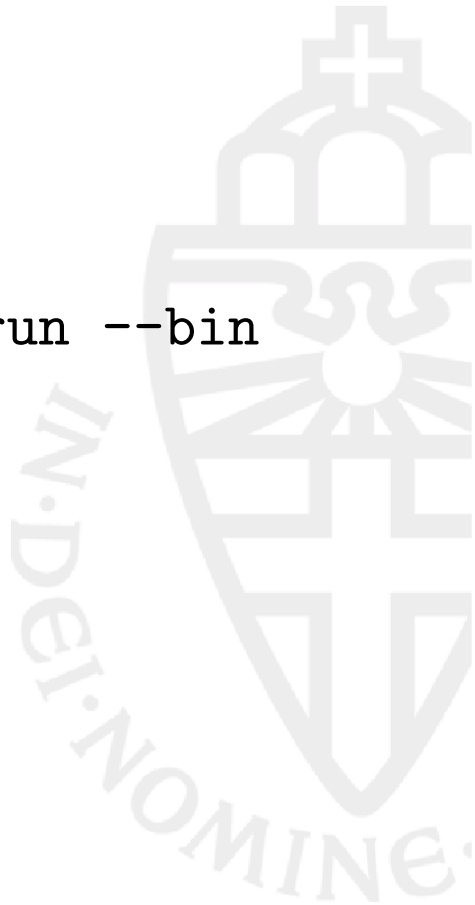
C example revisited

- Solution: `Arc<T>`
- `Arc<T>` is an atomically reference counted type
- This type guarantees that the modifications to the reference count occur atomically, i.e. are thread-safe.



Demo

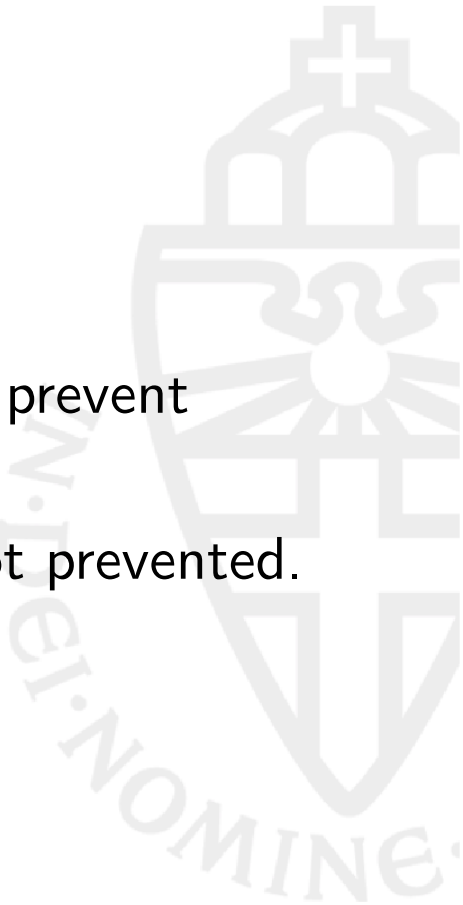
- File `rust-samples/src/shmem_arc.rs`
- Compile & run : `cd rust-samples && cargo run --bin shmem_arc.rs`



Conclusion

Conclusion

- Creation and use of threads
- Message passing
- Shared state
- Rusts ownership model and type system statically prevent data races and invalid references
- However, race condition such as dead-locks are not prevented.



References I