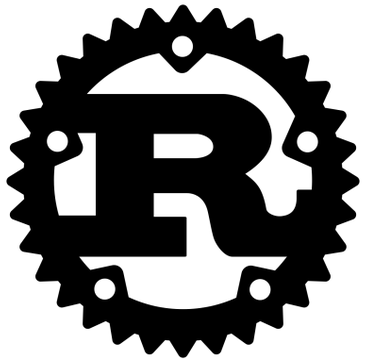


MODELLI DI SIMULAZIONE BASATI SU AGENTI CON RUST

Primo Relatore:
Prof. Alberto Negro

Secondo Relatore:
Carmine Spagnuolo

INDICE

ISISLab

AGENT BASED MODEL

- Analisi di sistemi reali complessi
- ABM composto da
  - Agenti
  - Relazioni
  - Regole

ISISLab

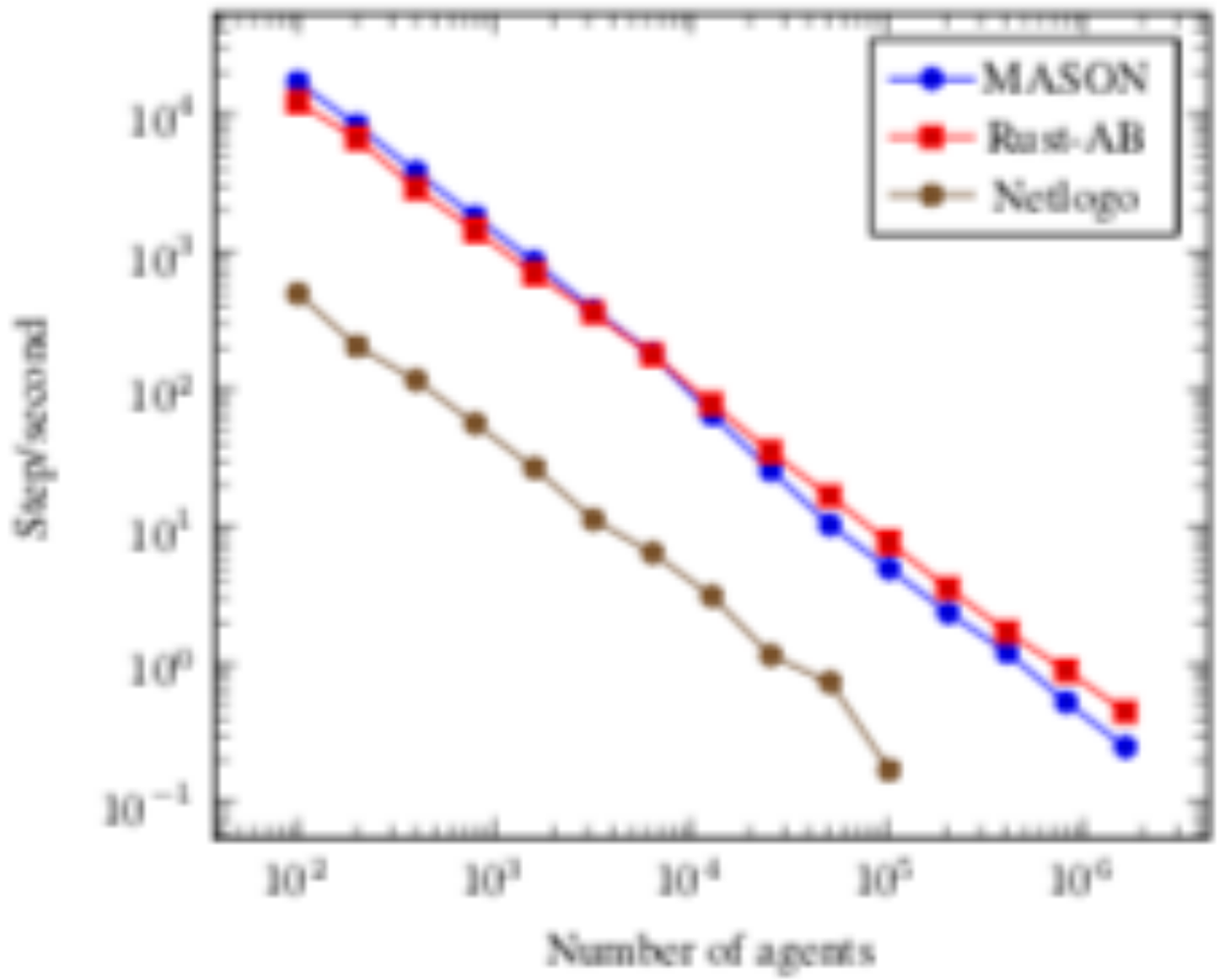
PERCHÉ USARE RUST PER UN ABM

- Performance simili a quelle del C
- Linguaggio pensato per sistemi complessi e soluzioni embedded, ma molto più ad alto livello rispetto ad altri competitor
- Safe memory model
- Caratteristiche OOP

RUST-AB

- Libreria ready-to-use
- L'architettura è ispirata dal design di MASON per facilitarne l'utilizzo

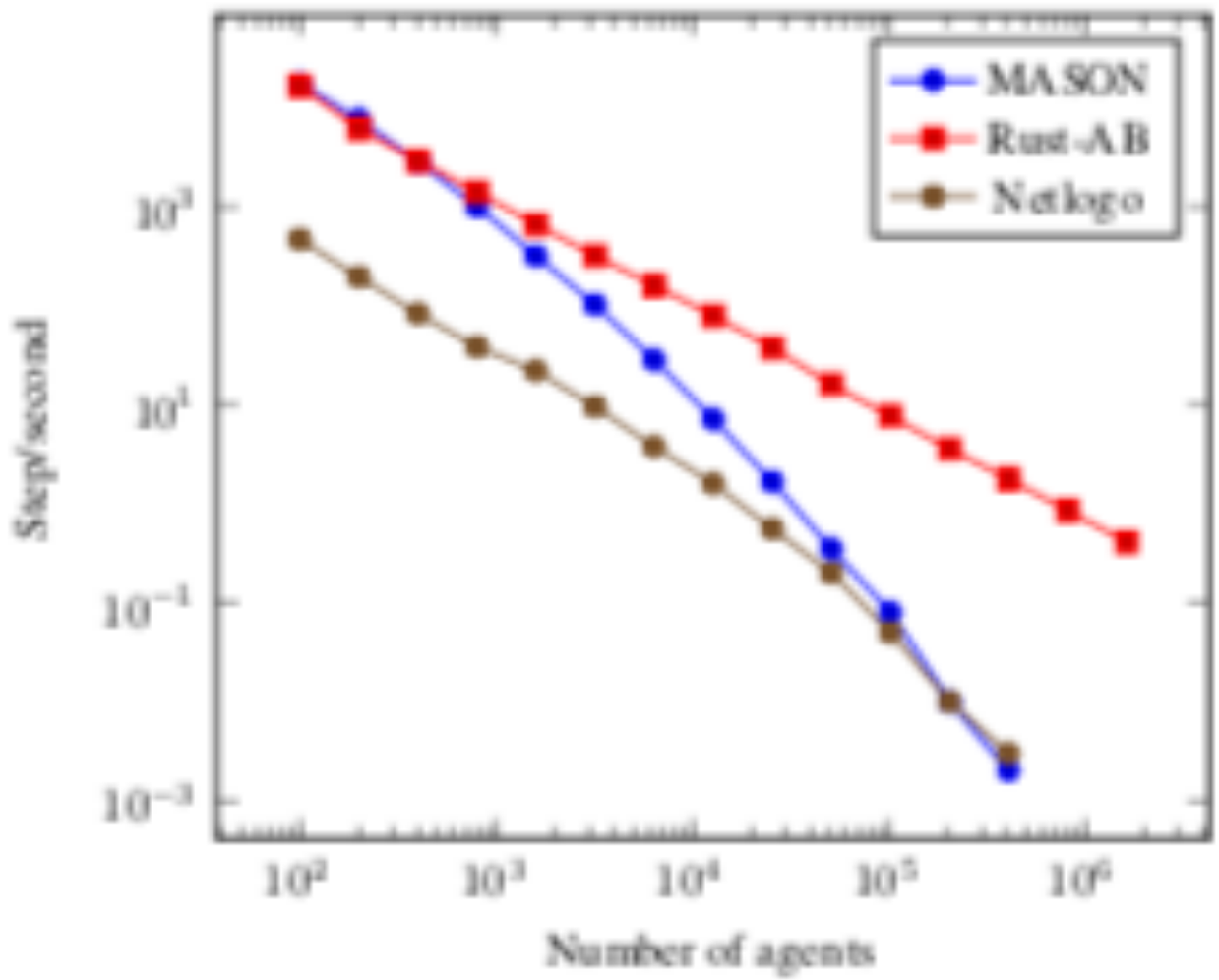
RUST-AB VS MASON



(a) Constant Agents Density

RUST-AB VS MASON

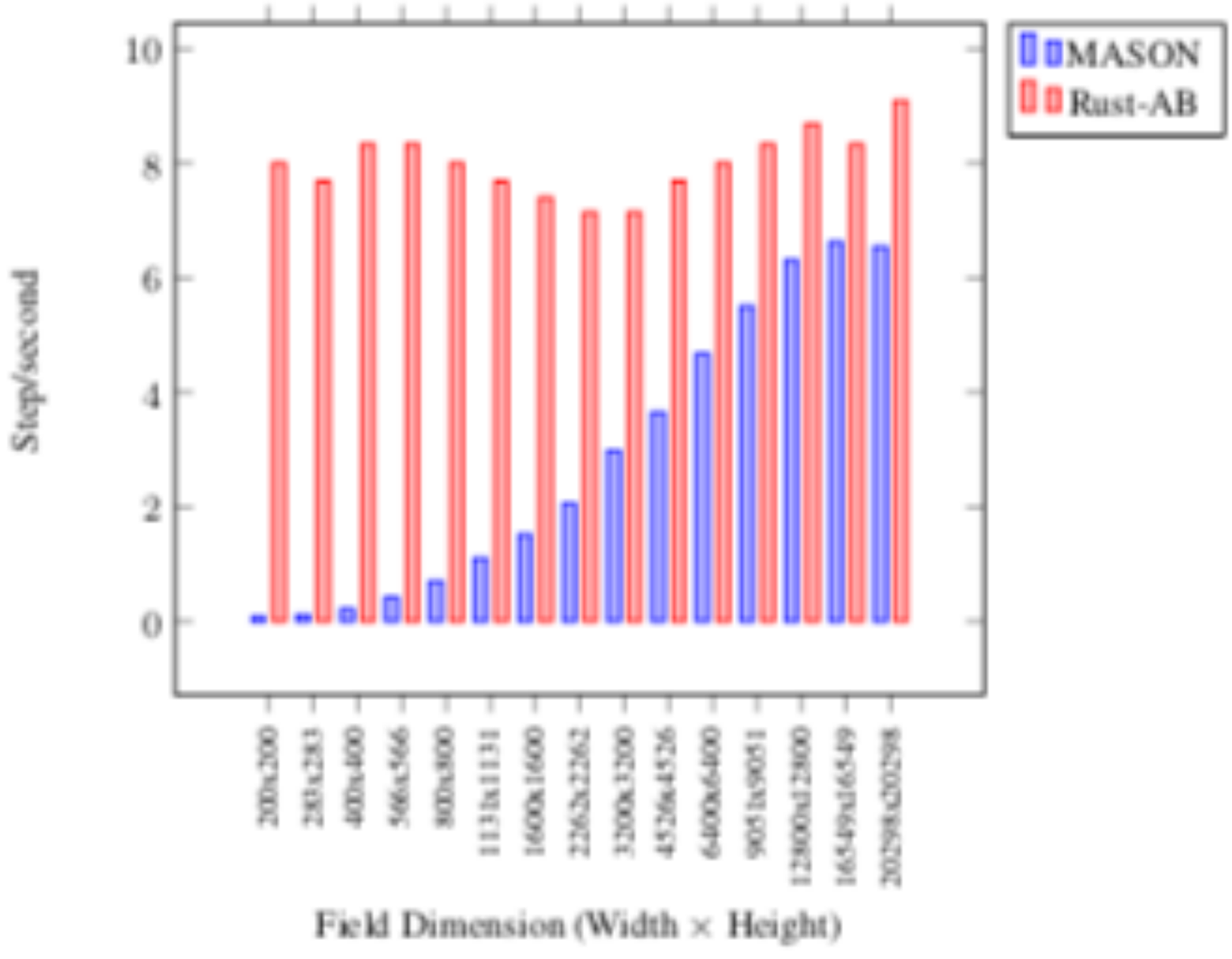
RUST-AB VS MASON



(b) Constant Field Size

RUST-AB VS MASON

RUST-AB VS MASON



ISISLab

ARCHITETTURA

- Agent
  - Trait
    - qualsiasi struct può implementare *Agent*
    - bisogna implementare la funzione *step* per poter effettuare i passi di simulazione

ARCHITETTURA

```rust
pub trait Agent {
    fn step(&mut self);
}
```

ARCHITETTURA

- AgentImpl
  - struct che agisce come un wrapper sull'agente, fornendogli un id univoco e un boolean per lo scheduling

ARCHITETTURA

```rust
#[derive(Clone, Debug)]
pub struct AgentImpl<A: Agent + Clone> {
    pub id: u32,
    pub agent: A,
    pub repeating: bool,
}
```

ARCHITETTURA

- Priority
  - Definisce una struttura per gestire la priorità di un agente

ARCHITETTURA

```rust
#[derive(Clone, Debug)]
pub struct Priority{
    pub time:f64,
    pub ordering:i64
}
```

ARCHITETTURA

ARCHITETTURA

```rust
impl Ord for Priority {
    fn cmp(&self, other: &Priority) -> Ordering {
        if self.time < other.time {return Ordering::Greater;}
        if self.time > other.time {return Ordering::Less;}
        if self.ordering < other.ordering {return Ordering::Greater;}
        if self.ordering > other.ordering {return Ordering::Less;}
        Ordering::Equal
    }
}
```

ARCHITETTURA

- Schedule
  - Un agente è schedulabile se implementa i Trait Agent e Clone
- Priority queue in cui gli agenti vengono schedulati secondo tempo e priorità

ARCHITETTURA

```rust
pub struct Schedule<A:'static + Agent + Clone + Send>{
    pub step: usize,
    pub time: f64,
    pub events: PriorityQueue<AgentImpl<A>,Priority>,
}
```

ARCHITETTURA

ARCHITETTURA

```rust
pub fn schedule_once(&mut self, agent: AgentImpl<A>,the_time:f64, the_ordering:i64) {
    self.events.push(agent, Priority{time: the_time, ordering: the_ordering});
}

pub fn schedule_repeating(&mut self, agent: A, the_time:f64, the_ordering:i64) {
    let mut a = AgentImpl::new(agent);
    a.repeating = true;
    let pr = Priority::new(the_time, the_ordering);
    self.events.push(a, pr);
}
```

ISISLab

ARCHITETTURA

- Location
  - Definisce un Real2D per gestire la posizione degli agenti
- Trait Location2D che definisce la struttura su cui il campo opera

ARCHITETTURA

```rust
#[derive(Clone, Default, Debug, Copy)]
pub struct Real2D {
    pub x: f64,
    pub y: f64,
}
```

ARCHITETTURA

ARCHITETTURA

```rust
pub trait Location2D {
    fn get_location(self) -> Real2D;
    fn set_location(&mut self, loc: Real2D);
}
```

ARCHITETTURA

- Field2D
  - E’ una matrice sparsa che permette di modellare una simulazione in uno spazio 2D

ARCHITETTURA

```rust
#[derive(Clone)]
pub struct Field2D<A: Location2D + Clone + Hash + Eq + Display + Copy> {
    pub findex: HashMap<A, Int2D>,
    pub fbag: HashMap<Int2D, Vec<A>>,
    pub fpos: HashMap<A, Real2D>,
    pub width: f64,
    pub heigth: f64,
    pub discretization: f64,
    pub toroidal: bool,
}
```

ARCHITETTURA

ARCHITETTURA

```
pub fn set_object_location(&mut self, object: A, pos: Real2D) {…
}
pub fn get_neighbors_within_distance(&self, pos: Real2D, dist: f64) -> Vec<A> {…

}
pub fn get_objects_at_location(&self, pos: Real2D) -> Vec<&A>{…
}
pub fn num_objects(&self) -> usize {…
}
pub fn num_objects_at_location(&self, pos: Real2D) -> usize {…
}
pub fn get_object_location(&self, obj: A) -> Option<&Real2D> {…
}
```