\texttt{\textasciitilde-serde} / \texttt{mem_dbg} / \texttt{sux} / \texttt{dsi-bitstream} / \texttt{webgraph}:

A Rust ecosystem for large-graph processing

Tommaso Fontana, \textbf{Sebastiano Vigna}, Stefano Zacchiroli

Partially supported by project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NGEU, and by project ANR COREGRAPHIE, grant ANR-20-CE23-0002 of the French Agence Nationale de la Recherche
The WebGraph Framework

• An open-source framework for compressed representation of graphs

• One of the most long-lived projects of this kind (>20 years!)

• Hundreds of publications in major conferences and journals using it (>1500 references)

• In 2011 news went around the world: Facebook had four degrees of separation

• The measurement was performed at Facebook using WebGraph (at that time, 721M nodes, 69B links, just 211GB!)
Moving to Rust

- A high-performance, safe language
- Memory safe (as Java), but with zero-cost abstractions
- Arrays as large as memory allows
- Fine-grained access to OS facilities (memory mapping)
- Lazy iterators
- Moving to Rust required porting a number of ideas
**ε-serde**

- **ε-copy**
- Like zero-copy, without the limitations
- Unlike abomonation, does not change the memory (e.g., you can map immutable files into memory)
- Unlike Zerovec, no impact on performance, and you use standard structures
- Unlike rkiv, the structure you deserialize is the structure you serialize, and no impact
- Requires collaboration from the underlying struct

```rust
use epserde::prelude::*;

fn main() -> anyhow::Result<()> {  
    let s = vec![0; 1000];
    s.store("foo.eps")?;
    let t = <Vec<i32>>::mmap("foo.eps", Flags::RANDOM_ACCESS)?;  
    Ok(())
}
```
- High-performance memory-occupancy detector
- Leverages on \( \varepsilon \)-serde's notion of zero-copy data to avoid iterating on collections
- Additionally, prints memory layouts

<table>
<thead>
<tr>
<th>Field</th>
<th>Size</th>
<th>Percentage</th>
</tr>
</thead>
<tbody>
<tr>
<td>size</td>
<td>815 B</td>
<td>100.00%</td>
</tr>
<tr>
<td>capacity</td>
<td>1215 B</td>
<td></td>
</tr>
</tbody>
</table>

- 985 B 100.00% • `example::Struct<example::TestEnum, example::Data<alloc::vec::Vec<u8>>>`
  - 16 B 1.62% **a**: `example::TestEnum`
    - Variant: Unnamed
      - 8 B 0.81% 0: usize
      - 1 B 0.10% 1: u8
  - 823 B 83.55% **b**: `example::Data<alloc::vec::Vec<u8>>`
    - 724 B 73.50% **a**: `alloc::vec::Vec<u8>`
    - 64 B 6.50% **b**: `alloc::vec::Vec<i32>`
    - 35 B 3.55% **c**: `(usize, alloc::string::String)`
  - 27 B 2.74% **deep_size**: `alloc::string::String`
  - 8 B 0.81% **size_of**: `alloc::string::String`
  - 8 B 0.81% **mem_size**: `alloc::string::String`
  - 138 B 14.01% **s**: `std::collections::hash::set::HashSet<usize>`
sux

- Succinct data structures
- Partial port of sux (C++ project) and Sux4J (Java project)
- There are some existing crates (some porting the projects above)
- We provide compositional constructor for mix-and-match between ranking and selection structures
- Mainly used for the Elias–Fano representation of monotone sequences (e.g., pointers into records)
dsi-bitstream

- High-performance bit streams
- Read/write data by word (settable)
- Supports little and big endian files
- Instantaneous codes for compression: Elias $\gamma$, Golomb, etc.
- Flexible architecture and benchmarks to tune to your hardware (use decoding tables or not?)
- A $\gamma$ code read in less than 2ns (for data with the intended distribution)
webgraph

- Rust port of the Java version
- Uses dsi-bitstream for instantaneous codes, sux for pointers into the bitstream
- On the Software Heritage graph (34 billion nodes, 517 billion arcs) a BFS visit is three time faster (3h)
- Unbelievably better ergonomics WRT Java
- Still in development on Github, soon into crates.io
- Composition-based labeling
- Lender- (rather than Iterator-) based architecture for iterators that depend on the graph state