

How to calculate an election result in your bedroom

by Michael Sproul



Senate Elections



- Senate composition is decided *per state*, with preferential voting
- Largest state is NSW with 5M voters

STV Preferential Voting

- You number candidates in order of preference
- An algorithm elects popular candidates, knocks out unpopular ones
- Lots of candidates, lots of ballots and complicated rules!

The AEC's dirty secret

The Australian Electoral Commission keeps their code secret!

... but all the data is available as CSV!

... and the voting algorithm is defined in legislation!

Why Rust?

- Memory usage
 $5 \cdot 10^6$ voters \times 10^2 candidates \times 4 bytes \approx 2GB RAM
- Speed
As fast as C/C++ without segfaults
- Types
Easy refactoring, controlled mutability, ADTs (enums)



Naive Number Crunching

Example:

Candidate A has 100,000 votes

Quota = 80,000 votes

Elect Candidate A and transfer each vote to the next round with value $20,000/100,000 = 1/5$

Logically, 4/5ths of each vote have been “used up”

Naive Number Crunching

What do we do with votes that are already fractional?

Naive Solution: Multiply their value by the new transfer value

E.g.

Candidate B has 80,000 $\frac{1}{5}$ votes

Apply transfer value of $(1/5)/80,000 = 1/400,000$

Votes from previous round now have value $1/5 * 1/400,000 = 1/2,000,000$

Naive Number Crunching

Uh oh!

- Irreducible fractional votes with **3000 digits in numerator and denominator**
- Standard `num` crate too slow, switched to `gmp` and `ramp`
- 20+ hours to run, 8+ gigabytes of memory used!

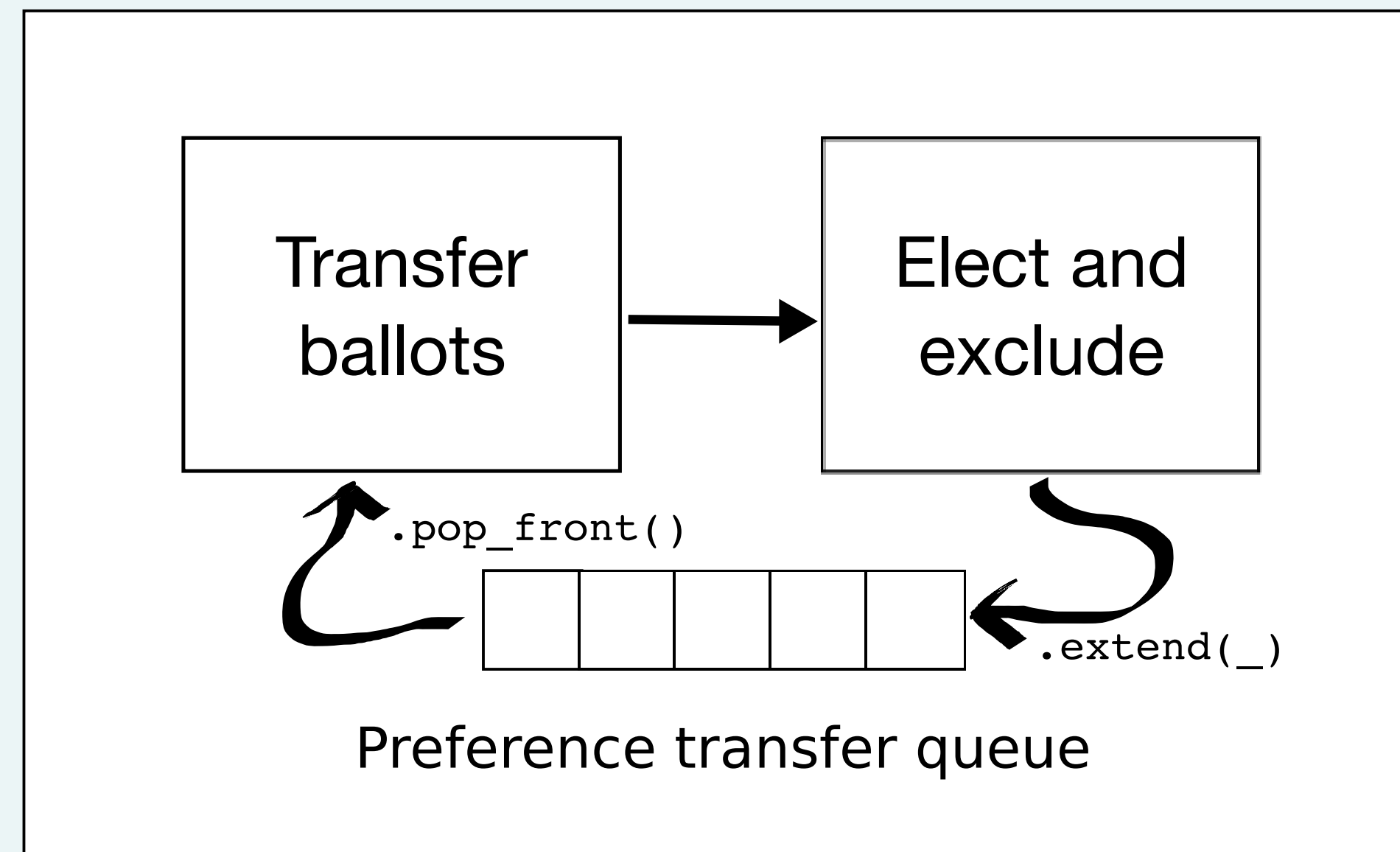
Reading the Legislation

- Round vote counts to integers
- “Aggregate” transfer values to avoid multiplying them:
$$t = (\text{num_votes} - \text{quota}) / (\text{num_ballots})$$
- Downside: more complex, more sensitive to order of operations

System Architecture



Main algorithm, round i



Core Data Structures

```
pub struct VoteMap<'a> {  
    info: HashMap<CandidateId, VoteInfo<'a>>,  
    candidates: &'a CandidateMap  
}  
  
struct VoteInfo<'a> {  
    votes: Int,  
    ballots: TransferMap<'a>,  
    eliminated: bool,  
}  
  
pub type TransferMap<'a> = BTreeMap<Frac, Vec<&'a mut Ballot>>;
```

Results

- Replicated the official result for all 8 states and territories
- For the largest state (NSW):
 - 4,705,270 votes
 - 20 seconds to run
 - ~1GB peak memory usage
- 1021 lines of code

Links

- My code: github.com/michaelsproul/aus_senate
- Another implementation: github.com/grahame/dividebatur

Image Credits

- *Rust Logo* by Mozilla, CC-BY <https://www.rust-lang.org/en-US/legal.html>
- *Australia Map* by Lokal_Profil, CC-BY-SA-2.5 https://commons.wikimedia.org/wiki/File:Australia_map,_States.svg
- *CSV Icon* made by Freepik from www.flaticon.com

Thank you!