Entity Component Systems with Rust

Dzmitry Malyshau [:kvark]
Rust Toronto Meetup, 2017

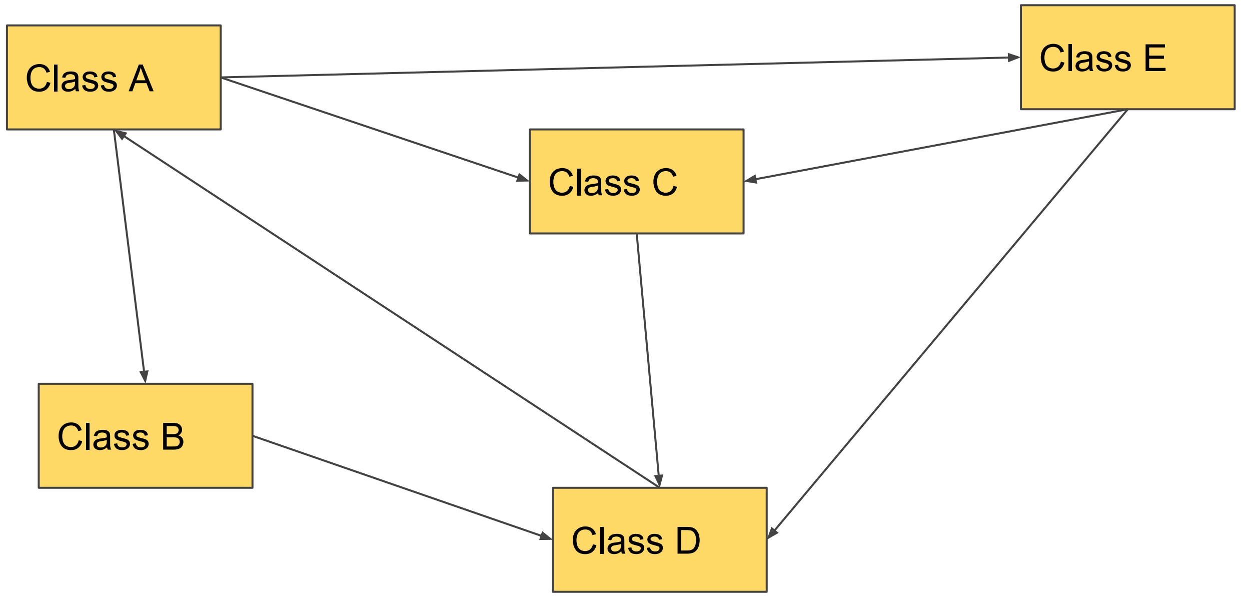
What is ECS anyway?

1. Way to design complex systems with many actors, and work with them. Much different from classical OOP, closer to functional programming.
2. A hardware-friendly way to lay out the data of such systems and process it efficiently.

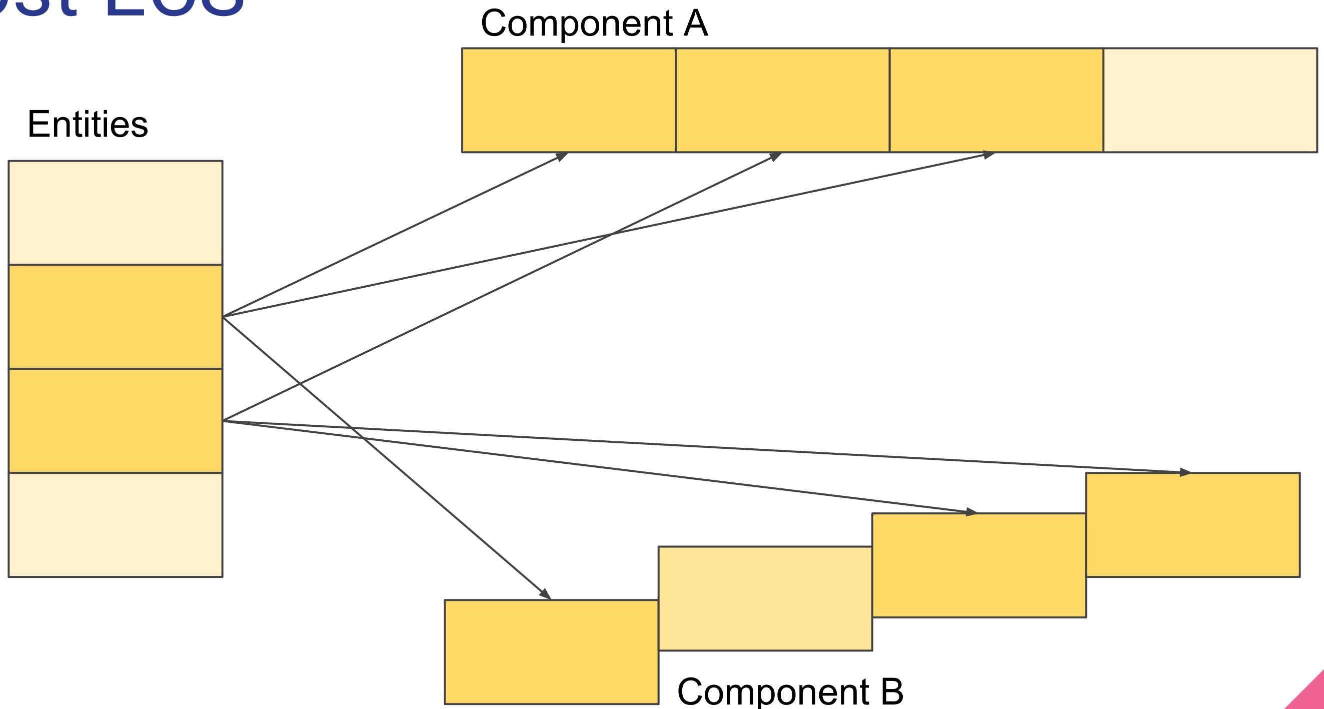
Highly popular in game development due to high complexity domain, large amounts of data, and performance constraints.

- Entity: a collection of components
- Component: a semi-independent piece of data
- System: function that processes an aspect of entities

Pre-ECS



Post-ECS



Things of trouble

- Dependent components
    - includes scene graphs, spatial trees, etc
- Component sharing
- Heterogeneous entities
    - includes rare entities, multiple domains, etc
- Non-uniform dependencies

Specs Parallel ECS

- Born 5th April 2016 (celebrating a year now!)
- Made mostly be me and Colin Sherratt
- First automatically parallel ECS in Rust
- Fastest kid in town
- Still growing...

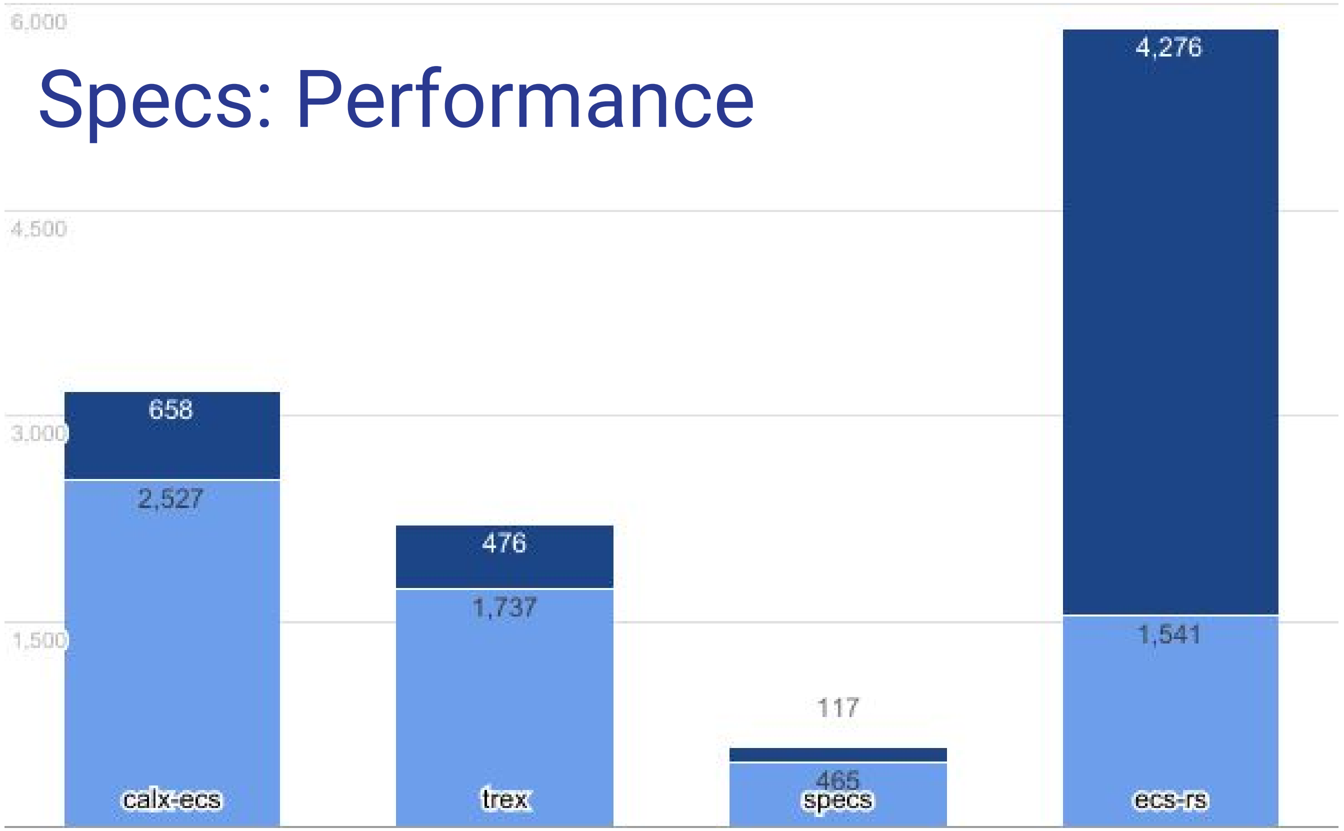
Specs: Example

```
fn run(&self, arg: specs::RunArg, time: Delta) {
    let (mut bullet, entities) = arg.fetch(|w| {
        (w.write::<Bullet>(), w.entities())
    });
    for (b, e) in (&mut bullet, entities).join() {
        b.life_time -= time;
        if b.life_time < 0.0 {
            arg.delete(e);
        }
    }
}
```

Specs: Features

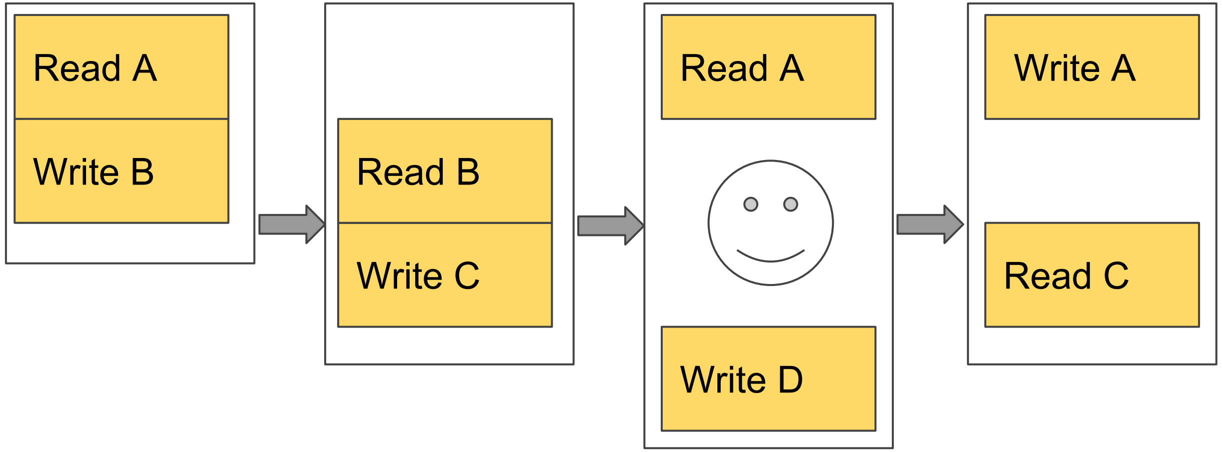
- Automatically parallel on the system level
- Component storages are abstract and independent
- Fast ad-hoc iteration over a group of components
- Asynchronous entity creation/deletion
- No dynamic dispatch, minimal overhead
- Simple traits, no macros

Specs: Performance



Specs: Ticketed locks

- Separates lock acquisition from waiting
- Improves the parallelism
- No code changes needed for the base use case
- Replaces `RwLock` of component storages



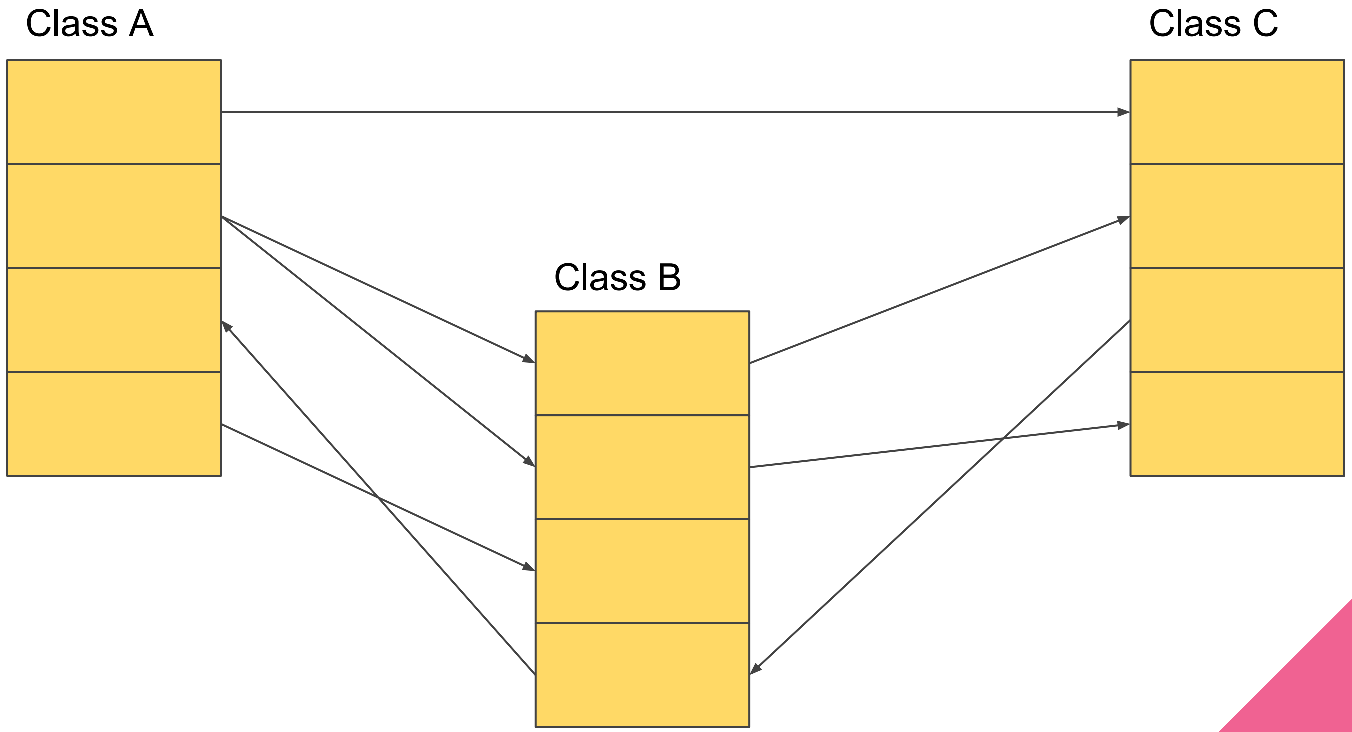
Specs: Future

- Serialization
- Better compile time enforcement
- Single-thread mode and/or lower overhead for synchronization
- Integration with a job system / inner parallelism

Beyond ECS

- What if…. components could point to other components?
- Then, EVERYTHING is a component
- Forming a Component Graph System?

Component Graph System



Froggy

- CGS implementation. NOT an ECS!
- Straightforward to use as regular OOP
- Automatic reference counting for entities
- Nice data placement
- No need to implement/fulfill any traits
- Systems are not fleshed out yet

Questions?

Links:

- https://github.com/slide-rs/specs
- https://github.com/lschmierer/ecs_bench

- https://github.com/kvark/froggy

Bonus: case studies

ECS.rs: https://github.com/jFransham/ecs-rs, last commit on on Apr 4, 2016

- avoids macros on the API side
- every entity is an AnyMap…
- somewhat verbose but safe system definitions
- static component lifetime bounds

Trex: https://github.com/rcolinray/trex, last commit on Apr 15, 2016

- macros for registering components and events
- systems register within a simulation, no parallel work

Bonus: case studies

Tiny-ecs: https://github.com/not-fl3/tinyecs, last commit on Jun 27, 2016

- actually used in Shar game
- systems define aspects* using macros
- entity creation possible almost everywhere

Ecs-rs: https://github.com/HeroesGrave/ecs-rs, last commit on Jul 1, 2016

- oldest (from 2014) and most sophisticated ECS in town
- complex macros for entities, aspects, and system definitions
- hot & cold storage types, active/passive systems

Bonus: case studies

Constellation: https://github.com/TomGillen/constellation/, updated on Jan 9, 2017

- very similar API to Specs, has parallel processing
- does not mutate in place, defers using *SystemCommandBuffer*

Calx-ecs: https://github.com/rsaarelm/calx-ecs, last commit on on Mar 4, 2017

- actually used in Magog roguelike: https://github.com/rsaarelm/magog
- fairly simple, a macro generates the world, but no systems
- built-in serialization with *serde*
- only *FnvHashMap* for components