Benchmark code: user.git.corp.google.com/jopra/steel

Entity Component Systems for Compilers

Reimagining the Abstract Syntax Tree

or

“How not to build a compiler”

J Pratt
they/them

Arcs (Sydney)
Raksha

jopra@
Cypher1

What on earth is an ECS?

Entity Component System

Used in Game development

Alternative to Object Oriented Programming

What do I mean by ECS?

Entity

A unique object (e.g. a character)

What do I mean by ECS?

Entity **Component**

A group of properties / values
(e.g. A character’s items, skills, etc.)

What do I mean by ECS?

Entity Component **System**

A structured way to handle the components

e.g.
A renderer,
A physics simulator,
A damage system

What do I mean by ECS?

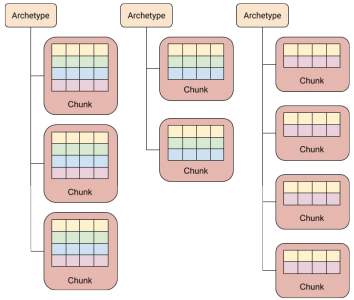
Let’s compare to OOP

What do I mean by ECS?

Extra terminology:

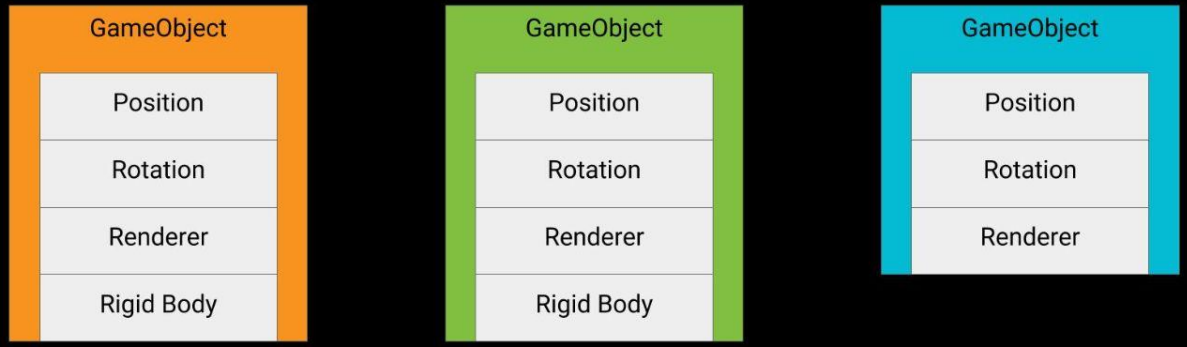
Archetype = A group of components

System = A thing that processes entities

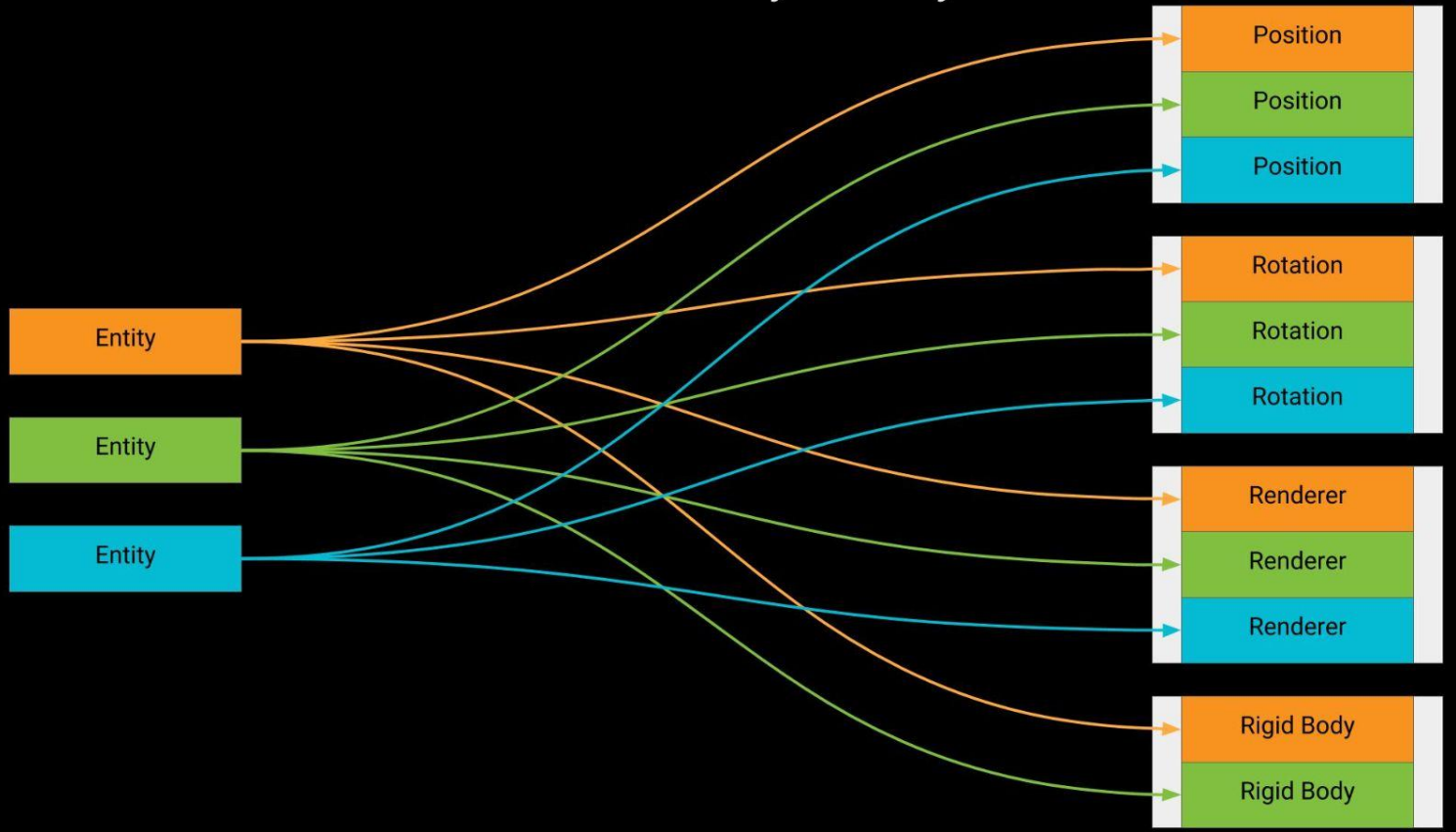


Chunk = Group of co-located instances of the same component/archetype

OOP → A GameObject is a container.



ECS → An entity is a key.



unity

What is an ECS like?

1. Systems operate over components
2. Components are co-located in memory
3. Look up time for each component is small (normally O(1))
4. Systems perform the same (or similar) actions on each component

Why is ECS used in Game development?

1. Separates concerns
2. Systems can operate independently
    a. Systems can operate in parallel*
    b. Systems can run at different times
3. Cache & Memory locality
4. Avoiding branches

Sources:
https://rams3s.github.io/blog/2019-01-09-ecs-deep-dive/ <- This one is probably the best imo
https://www.richardlord.net/blog/ecs/why-use-an-entity-framework.html
https://www.gamedevs.org/uploads/data-driven-game-object-system.pdf
https://web.archive.org/web/20171030021158/http://entity-systems-wiki.t-machine.org/
http://gameprogrammingpatterns.com/component.html

*separable systems: i.e. those using non-overlapping sets of components and global data

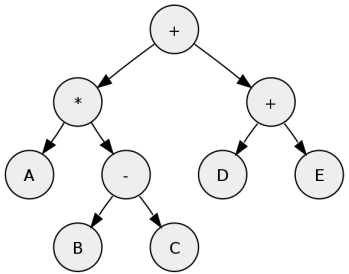
How is program data stored in compilers?

Abstract Syntax Tree!

AST for short

What do I mean by AST?

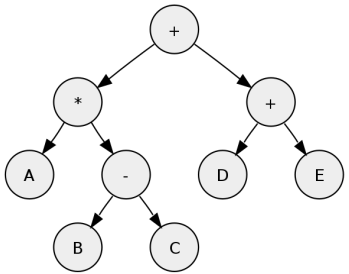
Abstract Syntax Tree



What do I mean by AST?

Abstract vs Concrete

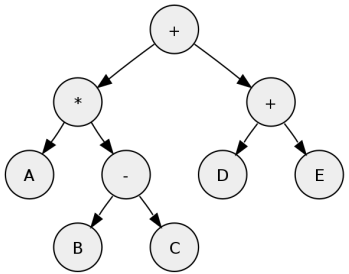
We’re happy to throw away some information



What do I mean by AST?

Abstract **Syntax** vs Semantics

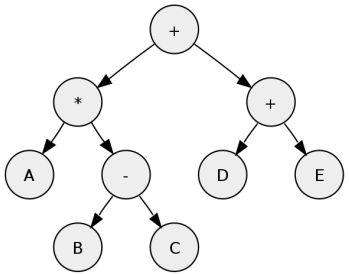
This doesn’t (normally) model the runtime behaviour



What do I mean by AST?

Abstract Syntax **Tree** vs …..Graph?

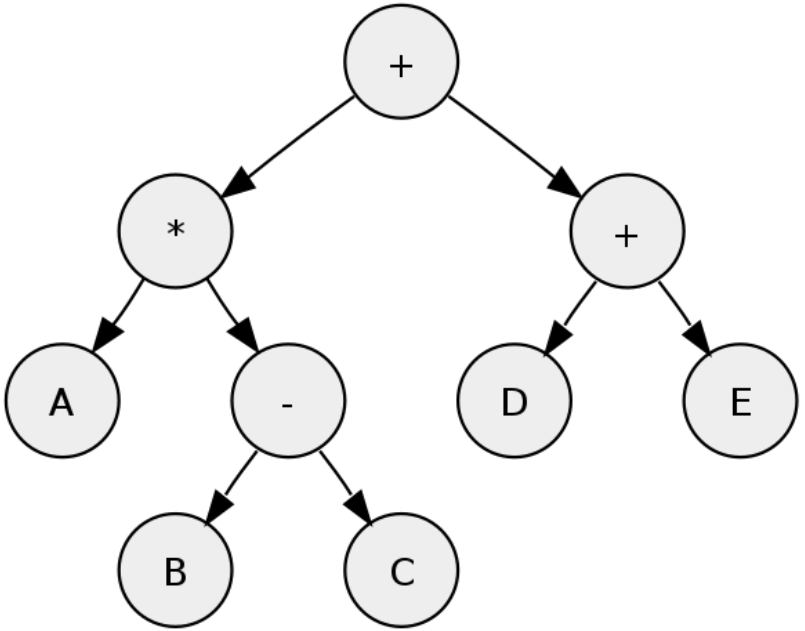
They (typically) only branch out…



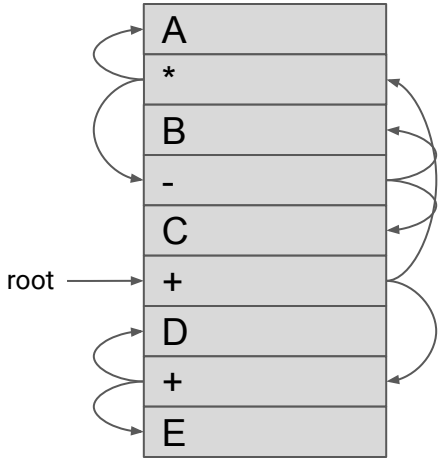
How is program data stored in compilers?

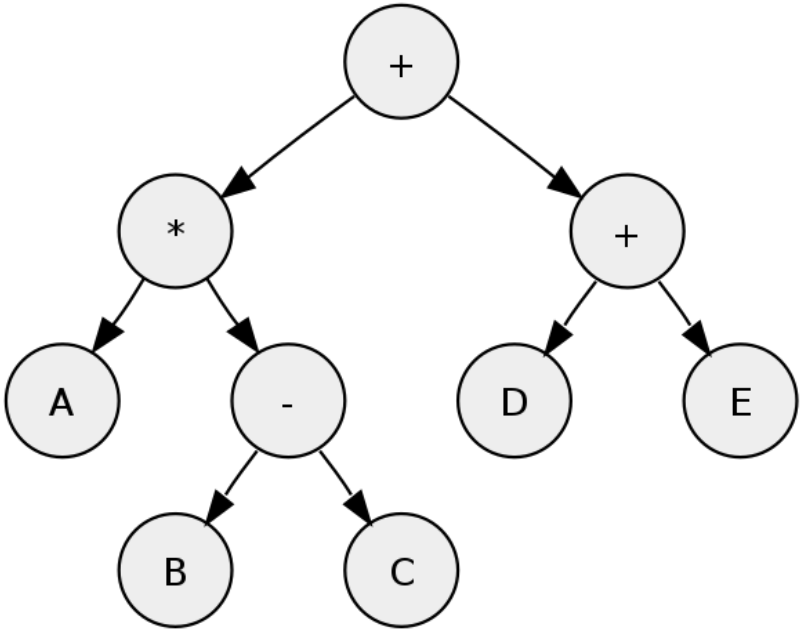
e.g.

```
A*(B-C)+D+E
```

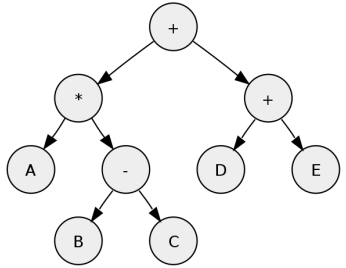


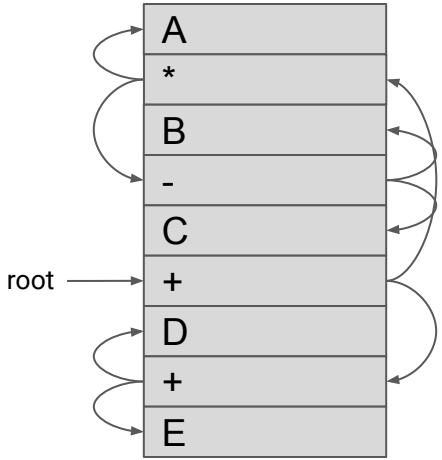
How is program data stored in compilers?





How is program data stored in compilers?



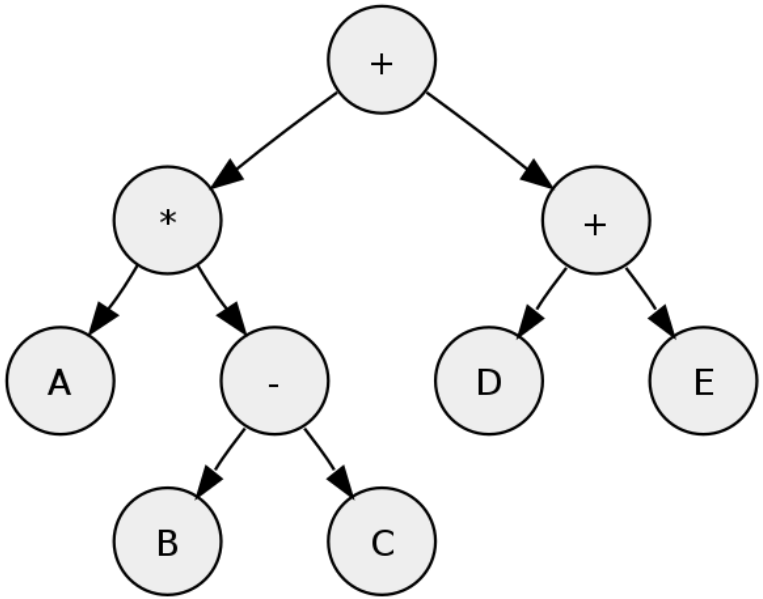


Nodes

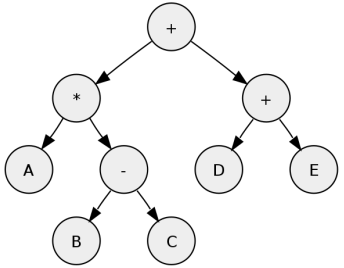
root

0: Symbol “A”
1: Operator “*”, Children(0, 2)
2: Symbol “B”
3: Operator “-”, Children(2, 4)
4: Symbol “C”,
5: Operator “+”, Children(2, 4)
6: Symbol “D”
7: Operator “+”
8: Symbol “E”

What would this look like in an ECS?



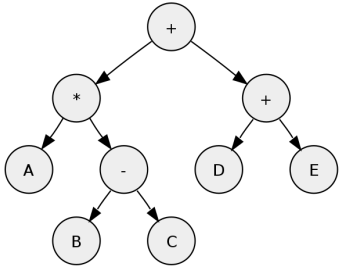
What would this look like in an ECS?



IsSymbol

A
B
C
D
E

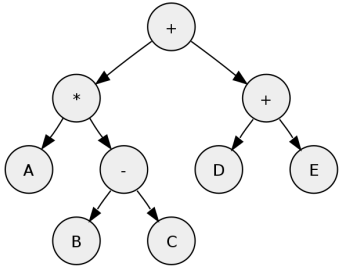
What would this look like in an ECS?



IsSymbol

0: A
1: B
2: C
3: D
4: E

What would this look like in an ECS?



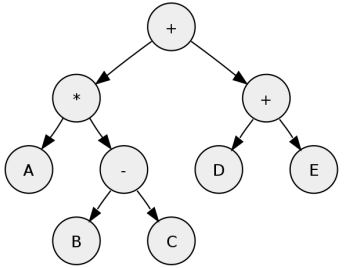
IsSymbol

0: A
1: B
2: C
3: D
4: E

Entities

0: IsSymbol 0
2: IsSymbol 1
4: IsSymbol 2
6: IsSymbol 3
8: IsSymbol 4

What would this look like in an ECS?



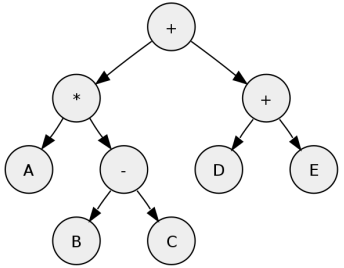
IsSymbol

0: E0, A
1: E2, B
2: E4, C
3: E6, D
4: E8, E

Entities

0: IsSymbol 0
2: IsSymbol 1
4: IsSymbol 2
6: IsSymbol 3
8: IsSymbol 4

What would this look like in an ECS?



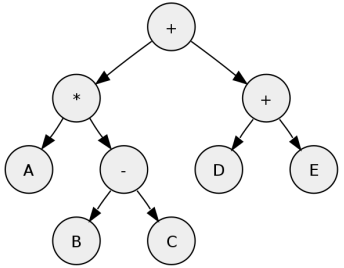
IsSymbol

0: E0, A
1: E2, B
2: E4, C
3: E6, D
4: E8, E

Entities

0: IsSymbol 0
1: IsOperator 0
2: IsSymbol 1
3: IsOperator 1
4: IsSymbol 2
5: IsOperator 2, IsRoot
6: IsSymbol 3
7: IsOperator 3
8: IsSymbol 4

What would this look like in an ECS?



IsOperator

*
-
+
+

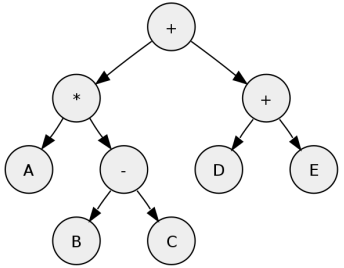
IsSymbol

0: E0, A
1: E2, B
2: E4, C
3: E6, D
4: E8, E

Entities

0: IsSymbol 0
1: IsOperator 0
2: IsSymbol 1
3: IsOperator 1
4: IsSymbol 2
5: IsOperator 2, IsRoot
6: IsSymbol 3
7: IsOperator 3
8: IsSymbol 4

What would this look like in an ECS?



IsOperator

0: *
1: -
2: +
3: +

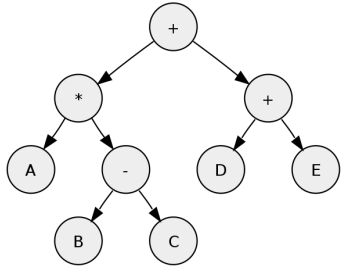
IsSymbol

0: E0, A
1: E2, B
2: E4, C
3: E6, D
4: E8, E

Entities

0: IsSymbol 0
1: IsOperator 0
2: IsSymbol 1
3: IsOperator 1
4: IsSymbol 2
5: IsOperator 2, IsRoot
6: IsSymbol 3
7: IsOperator 3
8: IsSymbol 4

What would this look like in an ECS?



IsOperator

0: E1, *
1: E3, -
2: E5, +
3: E7, +

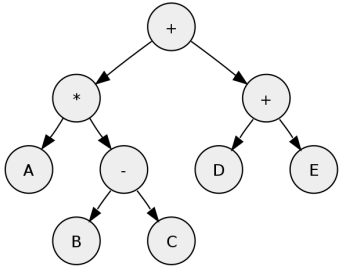
IsSymbol

0: E0, A
1: E2, B
2: E4, C
3: E6, D
4: E8, E

Entities

0: IsSymbol 0
1: IsOperator 0
2: IsSymbol 1
3: IsOperator 1
4: IsSymbol 2
5: IsOperator 2, IsRoot
6: IsSymbol 3
7: IsOperator 3
8: IsSymbol 4

What would this look like in an ECS?



IsOperator

0: E1, *, E0, E3
1: E3, -, E2, E4
2: E5, +, E1, E7
3: E7, +, E6, E8

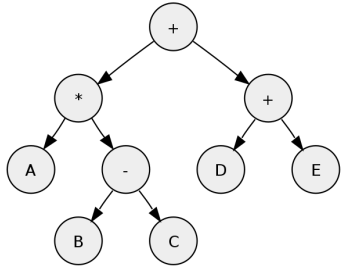
IsSymbol

0: E0, A
1: E2, B
2: E4, C
3: E6, D
4: E8, E

Entities

0: IsSymbol 0
1: IsOperator 0
2: IsSymbol 1
3: IsOperator 1
4: IsSymbol 2
5: IsOperator 2, IsRoot
6: IsSymbol 3
7: IsOperator 3
8: IsSymbol 4

What would this look like in an ECS?



IsOperator

0: E1, *, E0, E3
1: E3, -, E2, E4
2: E5, +, E1, E7
3: E7, +, E6, E8

IsSymbol

0: E0, A
1: E2, B
2: E4, C
3: E6, D
4: E8, E

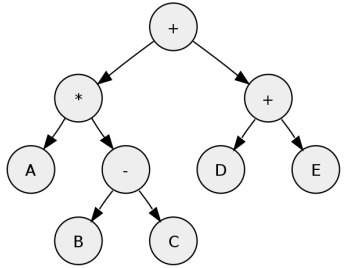
IsValue

….
_: E_, 123
…

Entities

0: IsSymbol 0
1: IsOperator 0
2: IsSymbol 1
3: IsOperator 1
4: IsSymbol 2
5: IsOperator 2, IsRoot
6: IsSymbol 3
7: IsOperator 3
8: IsSymbol 4

What would this look like in an ECS?



IsOperator

0: E1, *, E0, E3
1: E3, -, E2, E4
2: E5, +, E1, E7
3: E7, +, E6, E8

IsSymbol

0: E0, A
1: E2, B
2: E4, C
3: E6, D
4: E8, E

IsValue

….
_: E_, 123
…

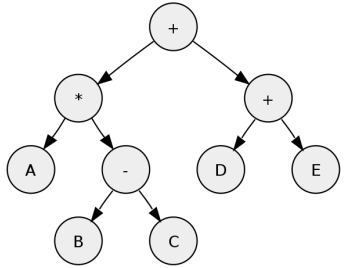
SourceLocation

….
_: E_, “test.rs” L5:C3
…

Entities

0: IsSymbol 0
1: IsOperator 0
2: IsSymbol 1
3: IsOperator 1
4: IsSymbol 2
5: IsOperator 2, IsRoot
6: IsSymbol 3
7: IsOperator 3
8: IsSymbol 4

How is program data stored in compilers?



AST

Nodes

root →

0: Symbol “A”
1: Operator “*”, Children(0, 2)
2: Symbol “B”
3: Operator “-”, Children(2, 4)
4: Symbol “C”,
5: Operator “+”, Children(2, 4)
6: Symbol “D”
7: Operator “+”
8: Symbol “E”

ECS

IsOperator

0: E1, *, E0, E3
1: E3, -, E2, E4
2: E5, +, E1, E7
3: E7, +, E6, E8

IsSymbol

0: E0, A
1: E2, B
2: E4, C
3: E6, D
4: E8, E

IsValue

….
_: E_, 123
…

SourceLocation

….
_: E_, “test.rs” L5:C3
…

Entities

0: IsSymbol 0
1: IsOperator 0
2: IsSymbol 1
3: IsOperator 1
4: IsSymbol 2
5: IsOperator 2, IsRoot
6: IsSymbol 3
7: IsOperator 3
8: IsSymbol 4

Why might we want ECS in Compilers?

1. Separates concerns
2. Systems can operate independently
    a. Systems can operate in parallel*
    b. Systems can run at different times
3. Cache & Memory locality
4. Avoiding branches

What can ECS do that might work in Compilers?

Anything that can be written as repeated passes over related nodes

- This seems plausible for Static analysis!
  So:
  - Optimization!
  - Type checking!
  - Dataflow checking!
  - Code generation!

A lot of what a modern compiler spends time on!

Does it actually work?

Well…

I don’t know.

Let’s write some benchmarks!

Steel: An attempt at ECS for compilers

- A small expression language
- Just enough program to do optimization on
- …
- Okay, maybe also a few fun features… just to keep life interesting

Steel: An attempt at ECS for compilers

Features

- Parser
  - Precedence:
    `i.e. 1+2*3 = 2*3+1 = 6`
  - Keyword arguments: `f(x=3)`
  - Positional arguments: `f(3)`
- Pretty printer
- Optimizer
  - Does constant folding
    e.g. `1+2 => 3`
- Interpreter
  - So many (two) types!
    - BuiltInFunction
    - i64
- and a random code generator

Steel: An attempt at ECS for compilers

Features

- Implemented in AST **AND** ECS
- 100%* Rust 🦀
- **Has Benchmarks**

*(ignore the data collection scripts)

Steel: An attempt at ECS for compilers

Demo?

Steel: An attempt at ECS for compilers

Benchmarks

- Timed operations over generated programs of arbitrary size
- Used two different storage backends with an abstraction layer
  - AST
  - ECS
- Programs were ‘trivially optimizable’

  E.g.

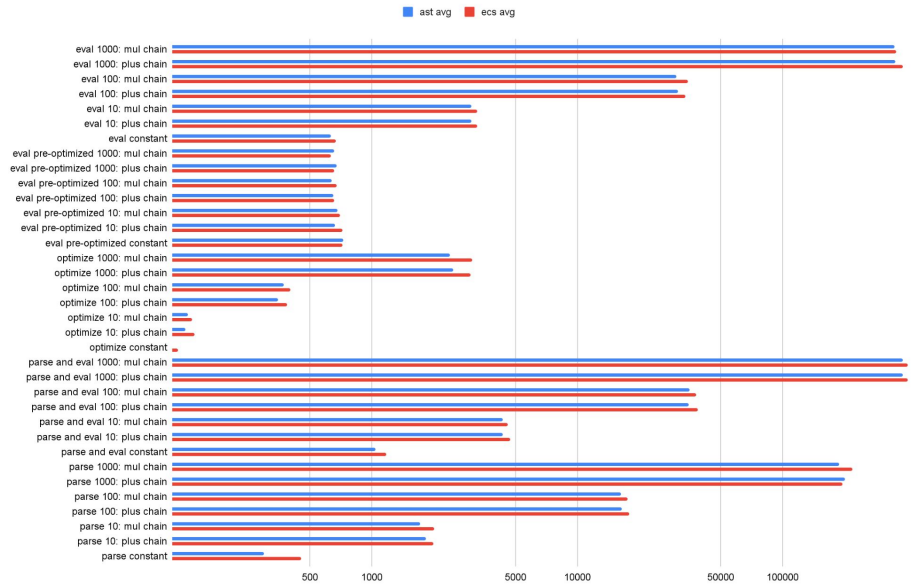
  1+2+3+4……+1000

  1*2*3*4……*1000

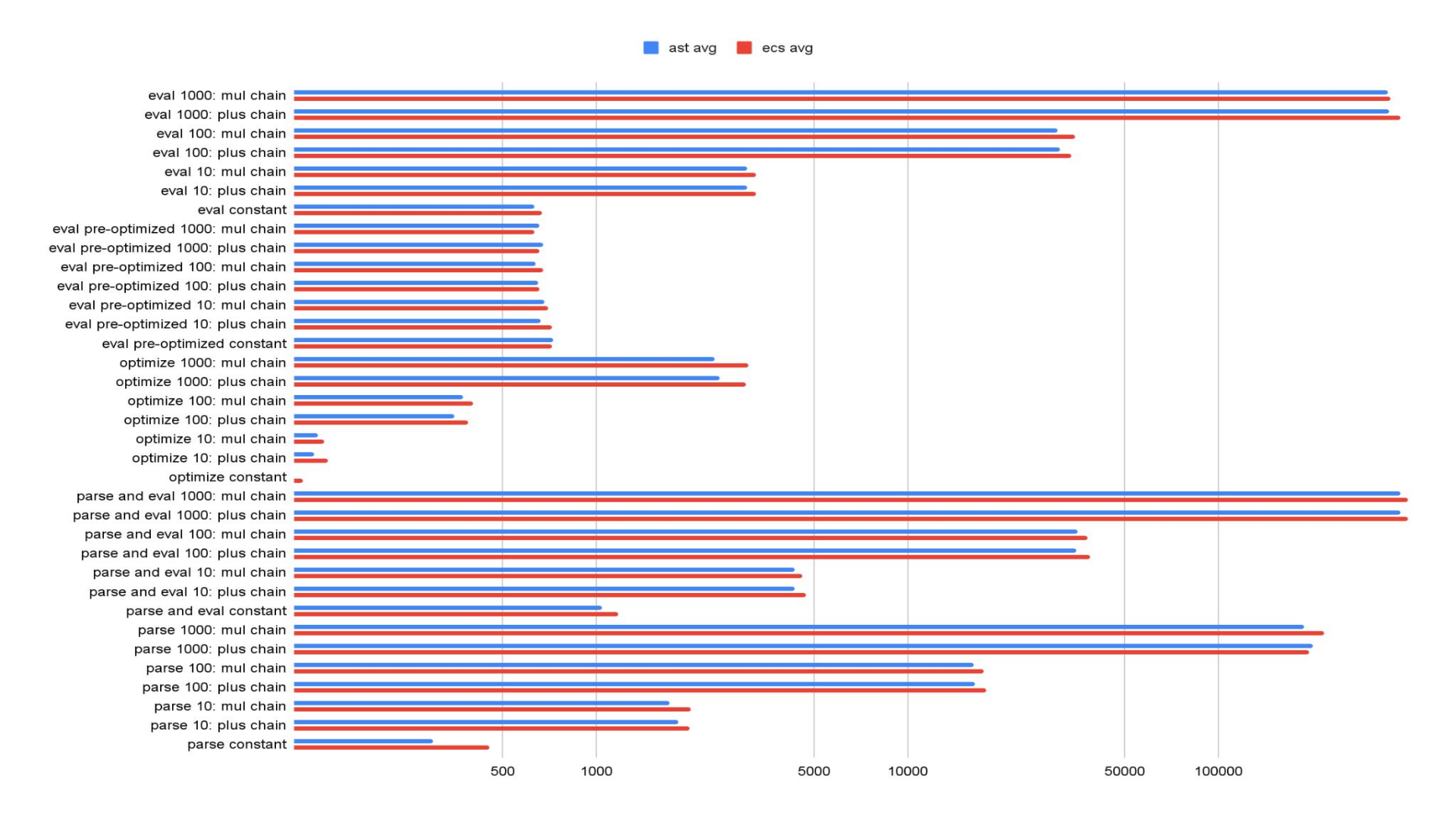
Steel: An attempt at ECS for compilers

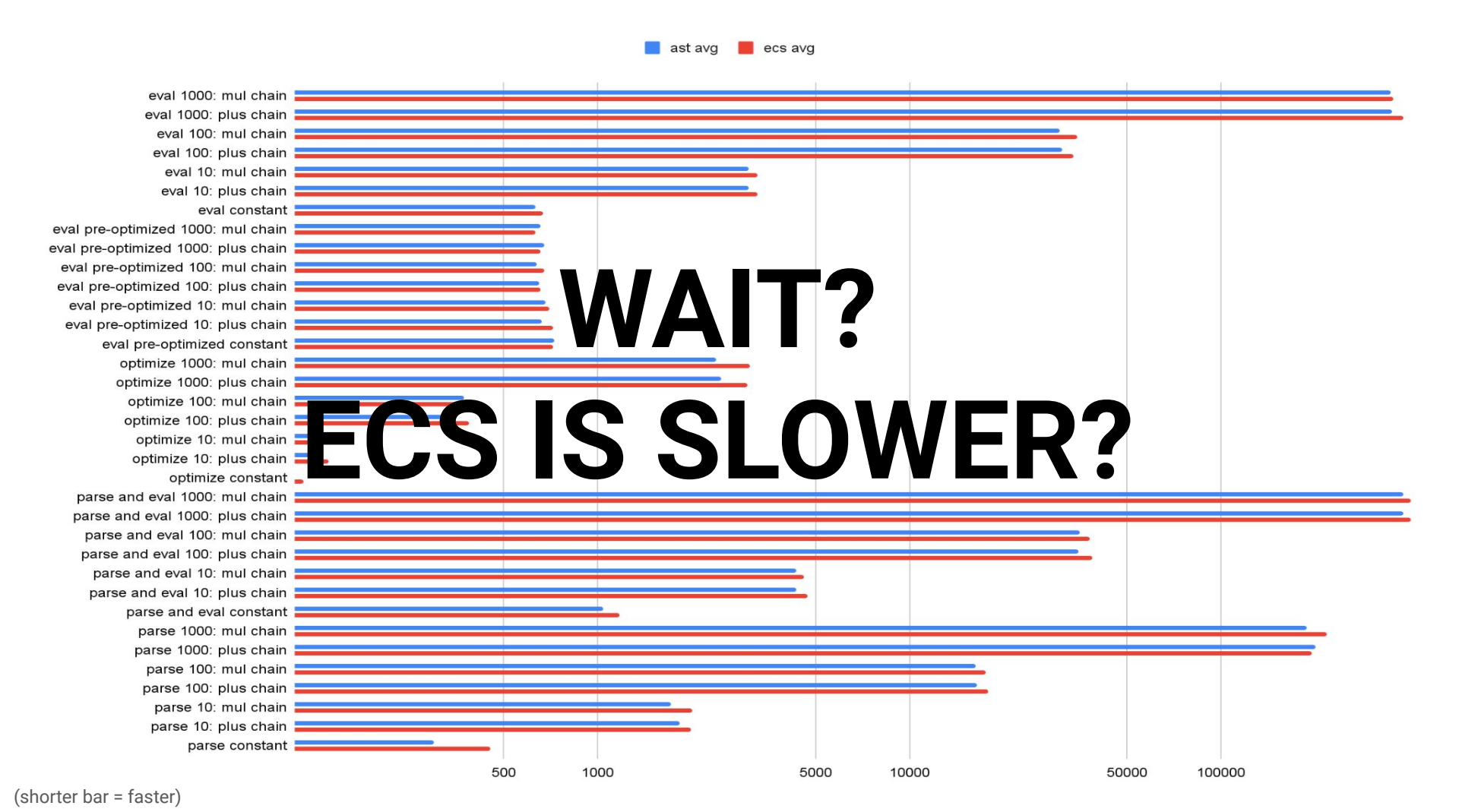
Benchmark results

(shorter bar = faster)



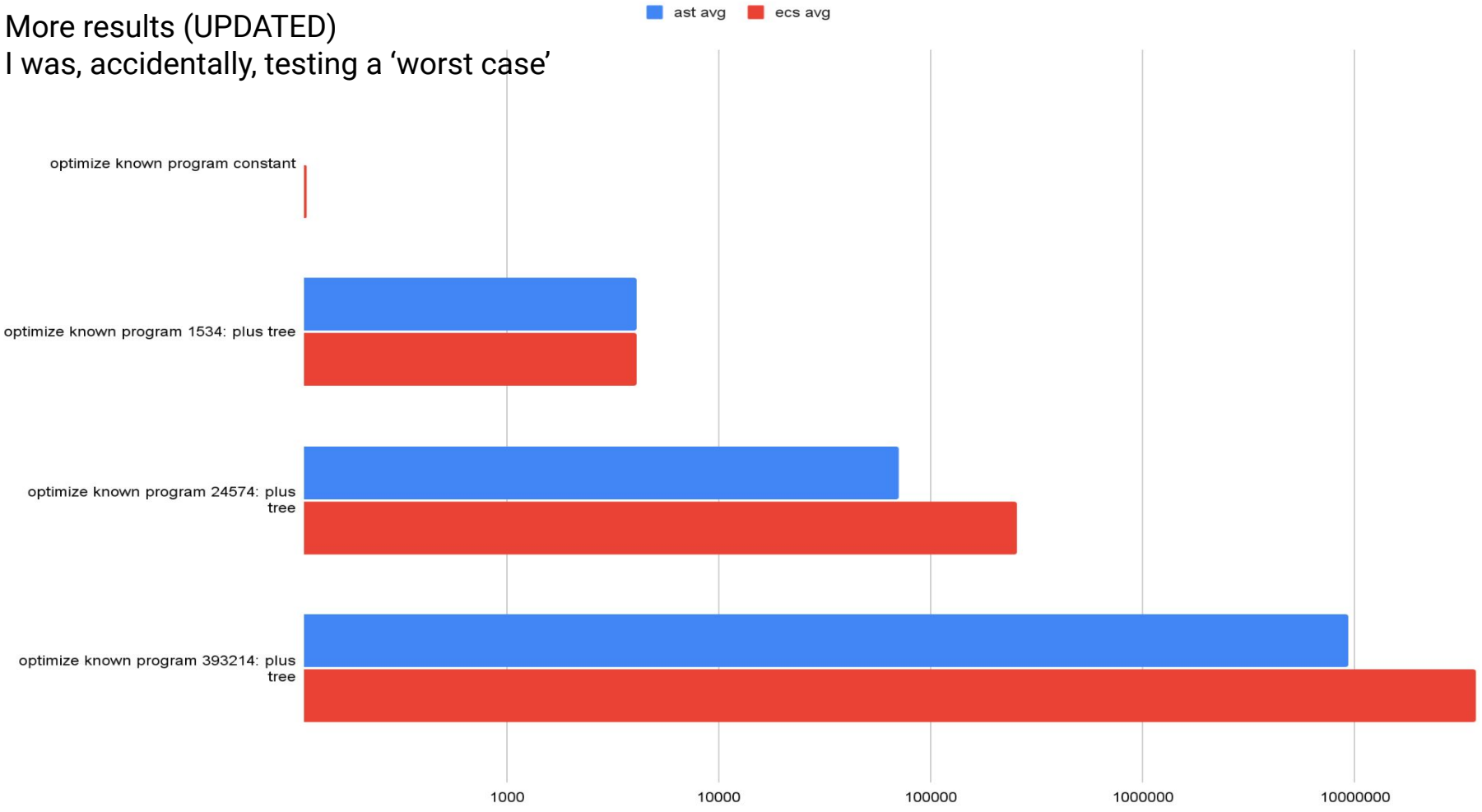
See the appendices for an update


ast avg
ecs avg
eval 1000: mul chain
eval 1000: plus chain
eval 100: mul chain
eval 100: plus chain
eval 10: mul chain
eval 10: plus chain
eval constant
eval pre-optimized 1000: mul chain
eval pre-optimized 1000: plus chain
eval pre-optimized 100: mul chain
eval pre-optimized 100: plus chain
eval pre-optimized 10: mul chain
eval pre-optimized 10: plus chain
eval pre-optimized constant
optimize 1000: mul chain
optimize 1000: plus chain
optimize 100: mul chain
optimize 100: plus chain
optimize 10: mul chain
optimize 10: plus chain
optimize constant
parse and eval 1000: mul chain
parse and eval 1000: plus chain
parse and eval 100: mul chain
parse and eval 100: plus chain
parse and eval 10: mul chain
parse and eval 10: plus chain
parse and eval constant
parse 1000: mul chain
parse 1000: plus chain
parse 100: mul chain
parse 100: plus chain
parse 10: mul chain
parse 10: plus chain
parse constant
500
1000
5000
10000
50000
100000

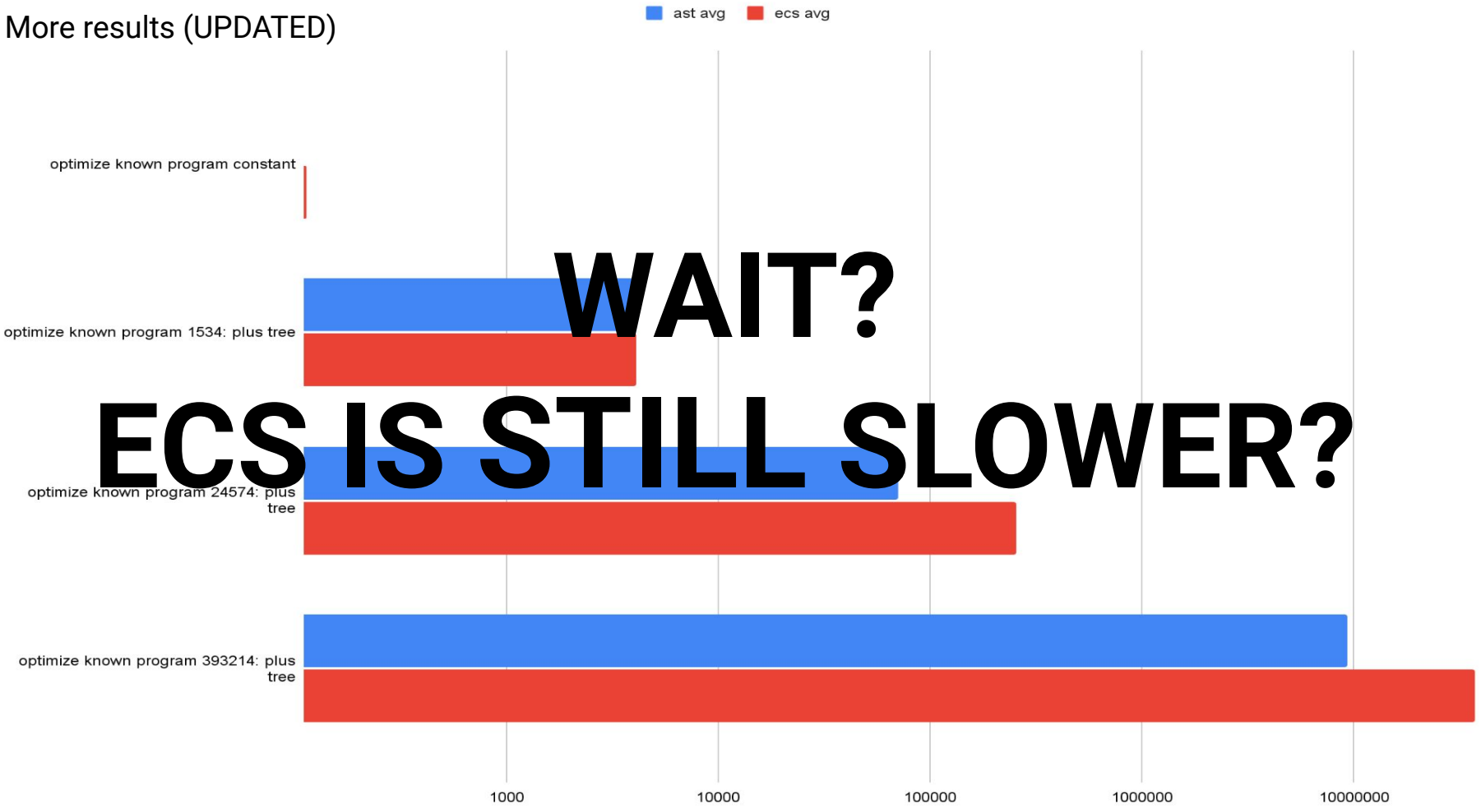


(shorter bar = faster)

More results (UPDATED)
I was, accidentally, testing a ‘worst case’



More results (UPDATED)



WAIT?

ECS IS STILL SLOWER?

Steel: What could be happening here?

‘Sins’ I committed when preparing this data

I.e. Places where mistakes may have been made.

- Used `bash` to process the output of Rust’s benchmark data
  - because it there was so much
  - because manually converting the data into a sheet/excel was way too much work
- Used sheets to convert units with a handwritten function
  - because sheets doesn’t understand small units of time

Steel: What could be happening here?

- My code could be really really bad

Steel: What could be happening here?

- My code **IS** suboptimal
- My benchmarks could be bad
- My ECS implementation **IS** slow
- My AST implementation… could be **fast???**

Future work

Future work: Move on?

Maybe AST is good actually?

Future work: Fix it?

- Try other ECS implementations
  - Bevy
  - Specs
- Try different modeling
- Maybe the way I’m using ECS is poor
  - Hmm…

Future work: Maybe the way I’m using ECS is poor

- The optimizer is a big *“Find and Replace”*
- We have good *“Find and Replace”* algorithms!

Future work: Maybe the way I'm using ECS is poor

We have good *"Find and Replace"* algorithms

- Regex??? (Not great for structured data)
- Techniques from Databases? (i.e. optimizing SQL queries)
- RETE
  - Uses graphs of candidates
  - Avoids re-doing work
  - Might be a good fit
- RETE is used by a bunch of different (non-compiler) tools already
  - Datalog
  - Souffle
- Maybe I should try building a compiler with these?
- Others?

Future work: Collaboration!

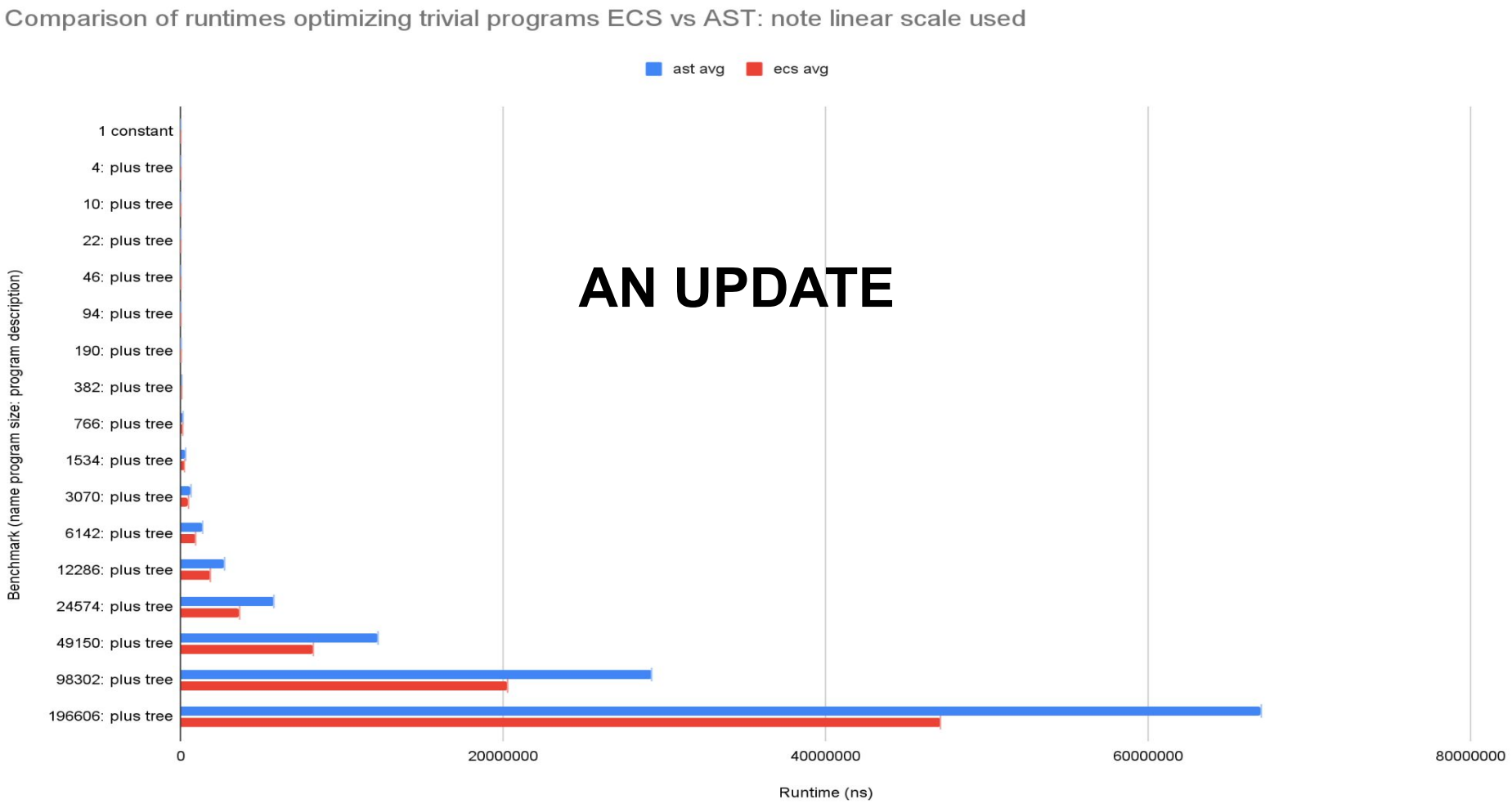
- Open source Steel
- Write a blog post?
- Write a paper?

Contributions & Collaborations welcome

:D

Thanks!

Continue on for appendices!


Comparison of runtimes optimizing trivial programs ECS vs AST: note linear scale used
ast avg
ecs avg
AN UPDATE
Benchmark (name program size: program description)
1 constant
4: plus tree
10: plus tree
22: plus tree
46: plus tree
94: plus tree
190: plus tree
382: plus tree
766: plus tree
1534: plus tree
3070: plus tree
6142: plus tree
12286: plus tree
24574: plus tree
49150: plus tree
98302: plus tree
196606: plus tree
0
20000000
40000000
60000000
80000000
Runtime (ns)

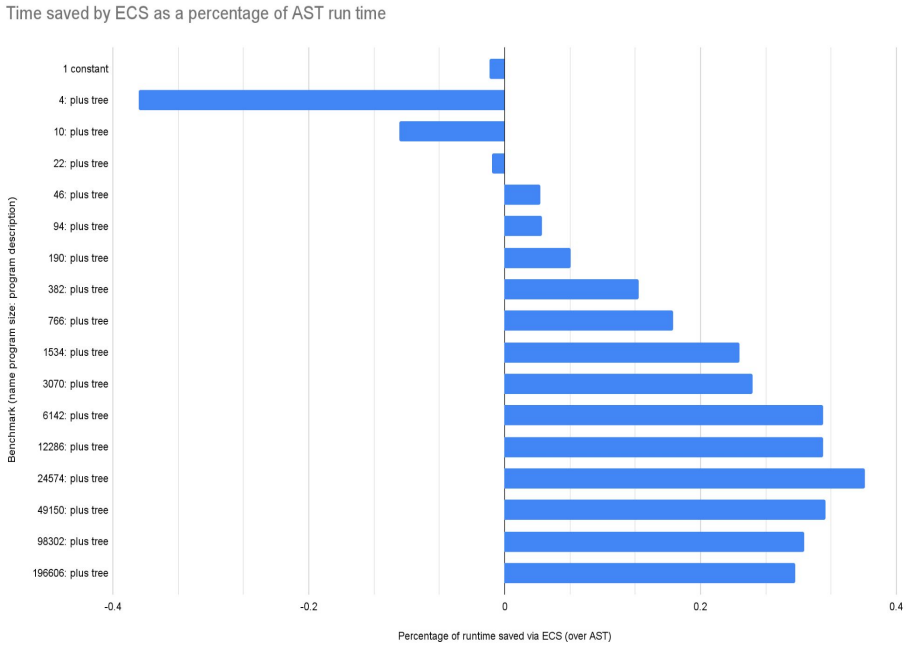
Improvements

Fixes:

- Split `Symbol` and `Operator` tables (like in slides)
- Store `‘left’` and `'right’` with calls
  (separate from the names args)
- Fetch Values and Operators from program data,
  Don’t use a hashmap for bookkeeping

Results:

- On small programs ECS’s constant factors outweigh the benefits :(
- On large programs ECS can achieve ~20%-38% performance improvement :)



Improvements

Takeaway:

- Pre-computation has to be **FASTER** than memory
- HashMaps are awesome, but dense, purpose built data structures are generally better
- Parser is 97% of my benchmark time, even using a library like nom
    - At this point I **should** probably stop

The following talk was **FAR** more helpful than anything else I watched or read:

- A Practical Guide to Applying Data-Oriented Design (by Zig creator: Andrew Kelley)

Rust tools I used (or wanted to)

CLI tools

- https://crates.io/crates/bacon/1.1.4 // A watch command
- sccache - crates.io: Rust Package Registry // A distributed compilation cache

Useful libraries

- https://crates.io/crates/static_assertions // static checks to avoid making mistakes
- https://crates.io/crates/memoize // caching/memoizing
- https://github.com/salsa-rs/salsa // incremental computation
- https://github.com/Geal/nom // makes parsers easy
- https://docs.rs/typed-arena/latest/typed_arena/ // Makes contiguous memory easier (comparison)

ECS frameworks

- Bevy // Comes recommended and with a great set of game dev tools
- https://github.com/amethyst/specs

Other compiler projects & tools

Tools

- https://github.com/jzimmerman/langcc // Tools for generating compilers
- Rust! - https://www.rust-lang.org/ // My recommendation for a programming language

Compiler Projects

- Cypher1/tako // My hobby project (very WIP)
- google-research/raksha // My day job

Approaches and algorithms

Equality Saturation

- Equality Saturation: a New Approach to Optimization *
- https://rosstate.org/publications/eqsat/RossThesis.pdf
- https://arxiv.org/pdf/2004.03082.pdf

RETE algorithm

- Wiki: https://en.wikipedia.org/wiki/Rete_algorithm
- Rete Algorithm - an overview | ScienceDirect Topics
- UNSW: The RETE Algorithm

Index mistakes are easy to make!

Typed ptrs & indexes can help!

Keeping type information can avoid mixing up indexes.

Code sample: here
Gist: Typed Index and Arena impl

```rust
use std::marker::PhantomData;

#[derive(Clone, Copy, Eq, PartialOrd, Ord)]
struct TypedIndex<T> {
    index: usize,
    ty: PhantomData<T>,
}

impl<T> std::fmt::Debug for TypedIndex<T> {
    fn fmt(&self, f: &mut std::fmt::Formatter) -> std::fmt::Result {
        write!(f, "{}_index({:?})", std::any::type_name::<T>(), self.index)
    }
}

impl<T> PartialEq for TypedIndex<T> {
    fn eq(&self, other: &Self) -> bool {
        self.index == other.index
    }
}

impl<T> TypedIndex<T> {
    fn new(index: usize) -> Self {
        Self { index, ty: PhantomData }
    }
}

fn main() {
    let args: Vec<String> = std::env::args().collect();
    let x = args.len();
    let ind1: TypedIndex<i32> = TypedIndex::new(x);
    let ind2: TypedIndex<i32> = TypedIndex::new(x+1);
    dbg!(std::mem::size_of::<TypedIndex<i32>>() == std::mem::size_of::<usize>());
    dbg!(ind1, ind2);
    dbg!(ind1 == ind2);
}
```

Thanks again!