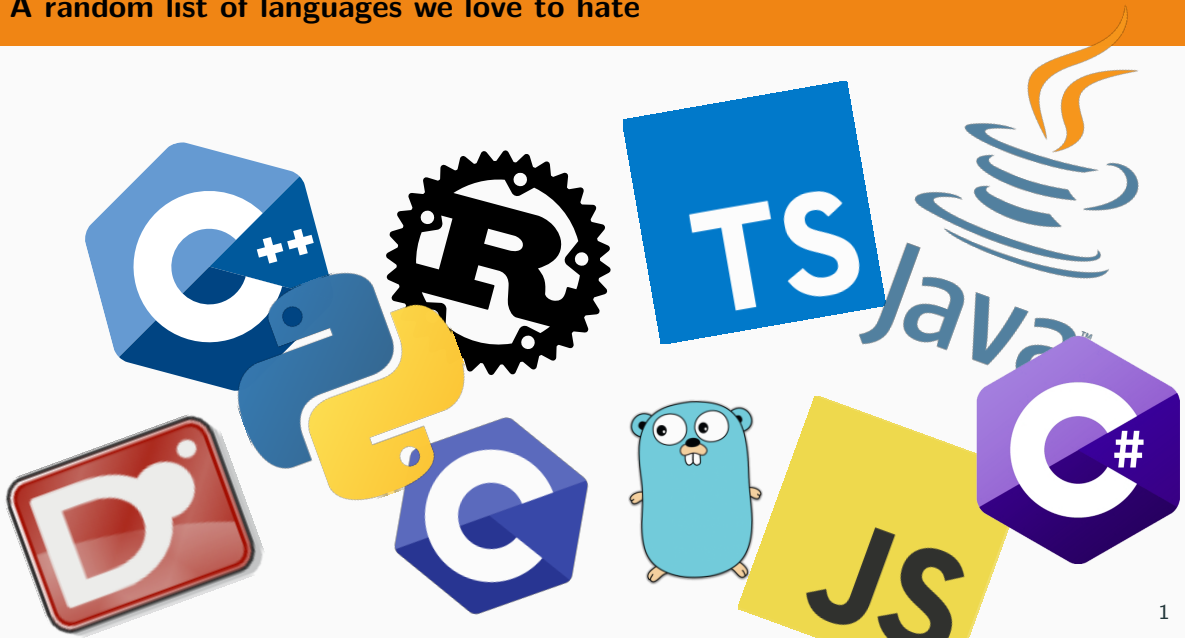
All Spreadsheets must Die

Robert Schadek

May 7, 2019

Getting started

A random list of languages we love to hate

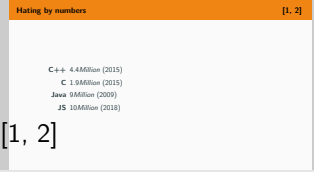


└A random list of languages we love to hate

- There are all these languages we like to hate.
- Many of them are good, many are more popular.
- Many might even be better than D in some regards.
- But we are missing something

Hating by numbers [1, 2]

C++	4.4*Million* (2015)
C	1.9*Million* (2015)
Java	9*Million* (2009)
JS	10*Million* (2018)



[1, 2]

- Many of the languages are used by more people than D is.
- Programms written in C run the world (linux).
- JS is eating everything.

These are all small fish

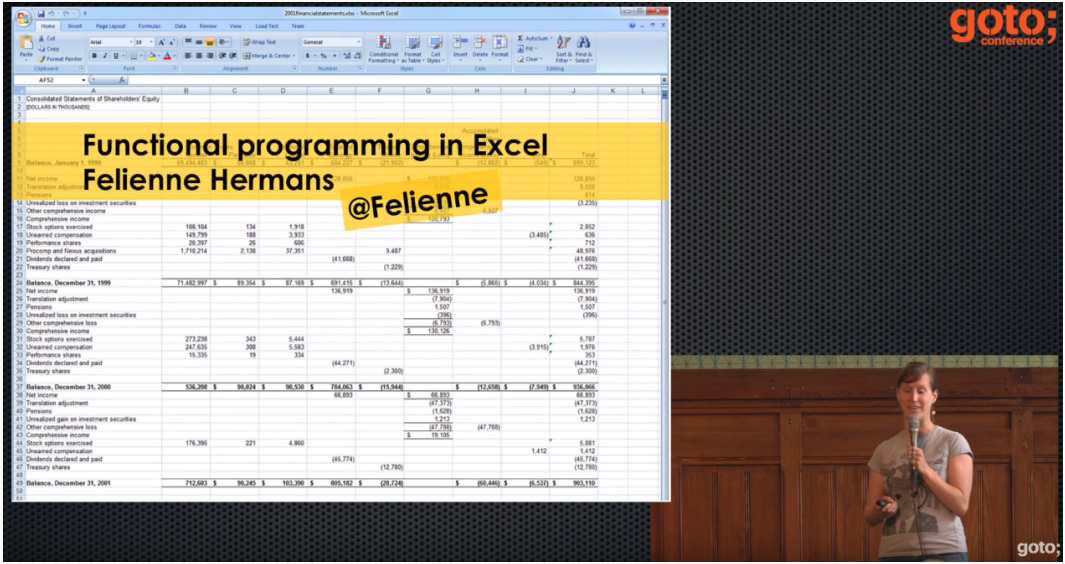
These are all small fish

Excel

$\approx$ 750 Million (2016)

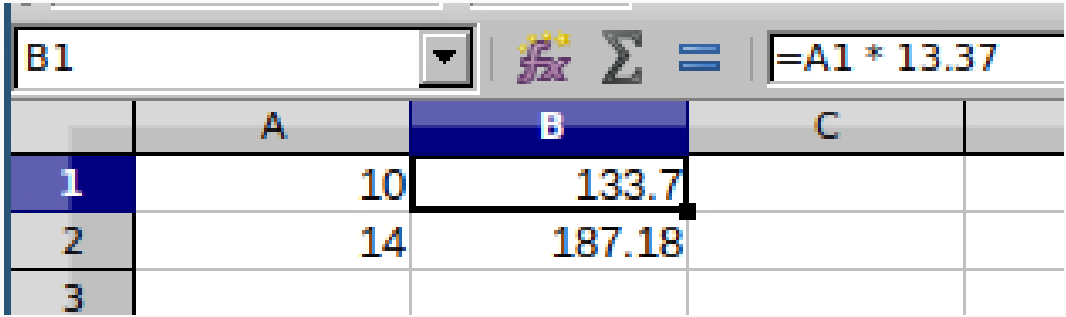

BUT EXCEL IS
NOT PROGRAMMING
imgflip.com

Oh, but it is [5]


2001financialstatements.xlsx - Microsoft Excel
Consolidated Statements of Shareholders' Equity
[DOLLARS IN THOUSANDS]
Functional programming in Excel
Felienne Hermans
@Felienne
goto; conference
goto;

A little bit of Spreadsheet bashing

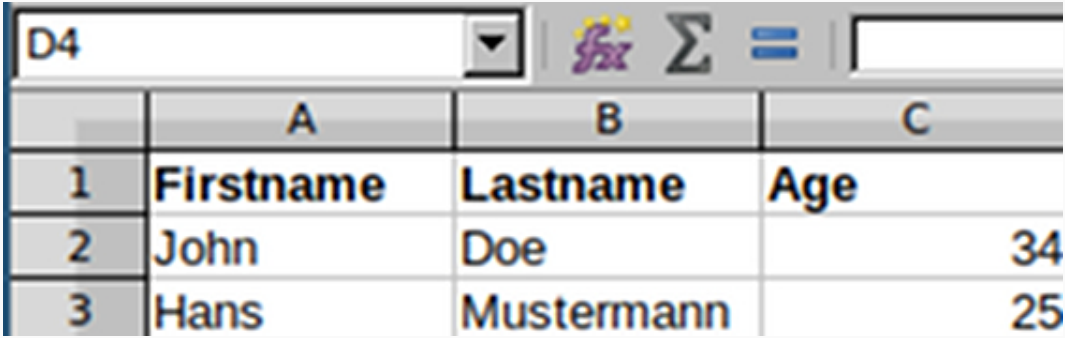
Seeing the code is difficult



Dynamic Types

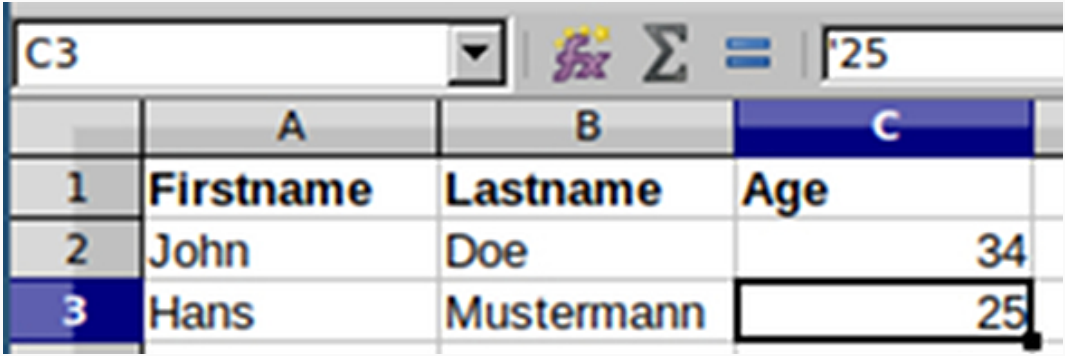
	A	B	C
1	**Firstname**	**Lastname**	**Age**
2	John	Doe	34
3	Hans	Mustermann	twenty-five
4			

Dynamic Types



	A	B	C
1	**Firstname**	**Lastname**	**Age**
2	John	Doe	34
3	Hans	Mustermann	25

Dynamic Types



git blame

git blame

lets not go there

Code refactoring

- `=SUM(1,2)`

Code refactoring

- `=SUM(1,2)`
- equal, identifier, lparen, int(1), comma, int(2), rparen

Code refactoring

- `=SUM(1,2)`
- equal, identifier, lparen, int(1), comma, int(2), rparen

- set Excel locale to de_DE

Code refactoring

- `=SUM(1,2)`
- equal, identifier, lparen, int(1), comma, int(2), rparen

- set Excel locale to de_DE
- `=SUM(1,2)`

Code refactoring

- `=SUM(1,2)`
- equal, identifier, lparen, int(1), comma, int(2), rparen

- set Excel locale to de_DE
- `=SUM(1,2)`
- equal, identifier, lparen, float(1.2), rparen

- Who cares about rvalues or named arguments.
- I want this in D.

Bits and Pieces

- Knowledge silos
- Slow
- No separation between data and code
- Access management ...

Bits and Pieces

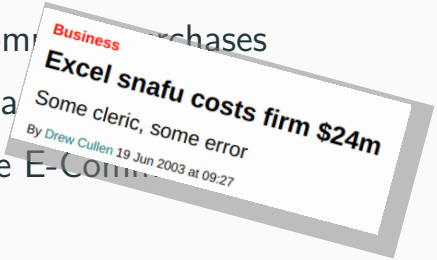
- Knowledge silos
- Slow
- No separation between data and code
- Access management ... anybody?

(Typical) Spreadsheet Lifecycle

1. Create private shopping spreadsheet
2. Show spreadsheet to college
3. Use spreadsheet for all company purchases
4. Put web frontend on spreadsheet backend
5. Pivot company to become E-Commerce company

(Typical) Spreadsheet Lifecycle

1. Create private shopping spreadsheet
2. Show spreadsheet to college
3. Use spreadsheet for all com[illegible]rchases
4. Put web frontend on sprea[illegible]
5. Pivot company to become E-Com[illegible]



(Typical) Spreadsheet Lifecycle

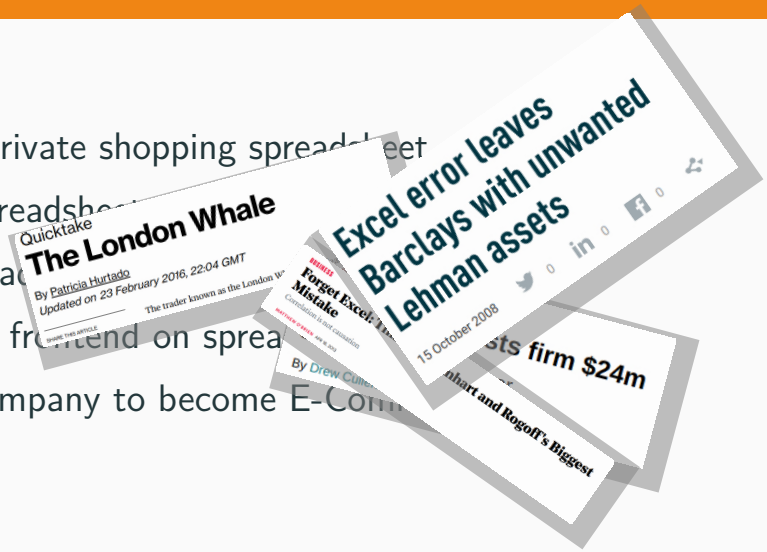
1. Create private shopping spreadsheet
2. Show spreadsh
3. Use spread chases
4. Put web frontend on sprea
5. Pivot company to become E-Com

(Typical) Spreadsheet Lifecycle

1. Create private shopping spreadsheet
2. Show spreadsh
3. Use spread chases
4. Put web fr tend on sprea
5. Pivot company to become E-Com

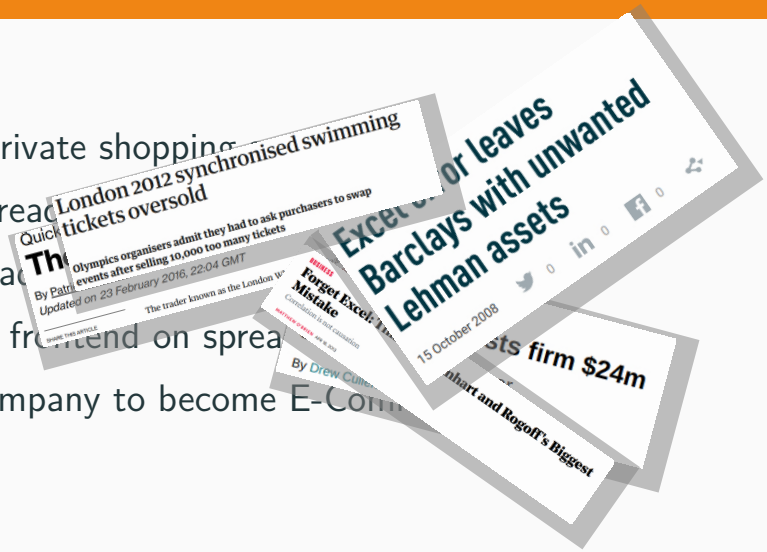
(Typical) Spreadsheet Lifecycle

1. Create private shopping spreadsheet
2. Show spreadsh
3. Use spread
4. Put web fr nd on sprea
5. Pivot company to become E-Com



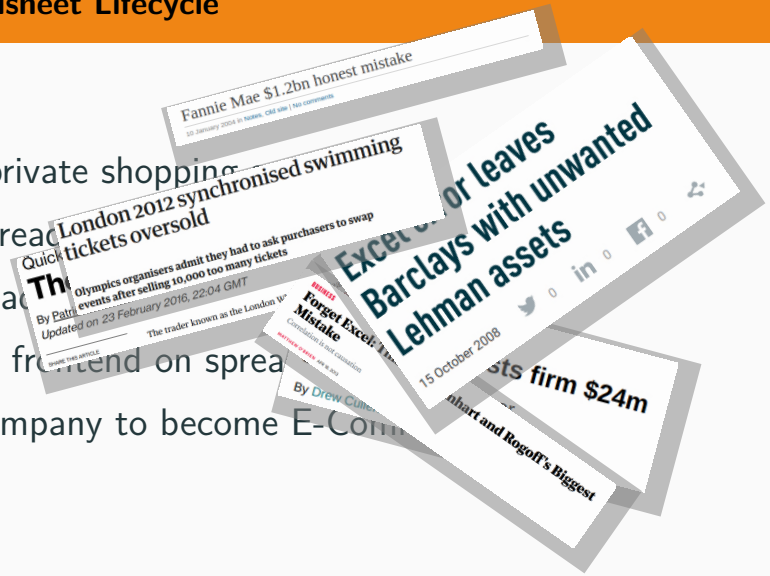
(Typical) Spreadsheet Lifecycle

1. Create private shopping
2. Show spread
3. Use spread
4. Put web frontend on sprea
5. Pivot company to become E-Com



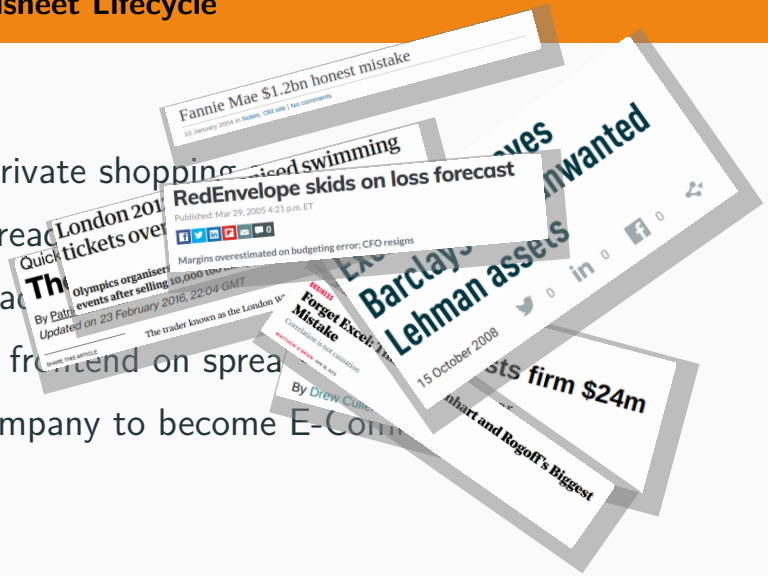
(Typical) Spreadsheet Lifecycle

1. Create private shopping
2. Show spread
3. Use spread
4. Put web frontend on sprea
5. Pivot company to become E-Com



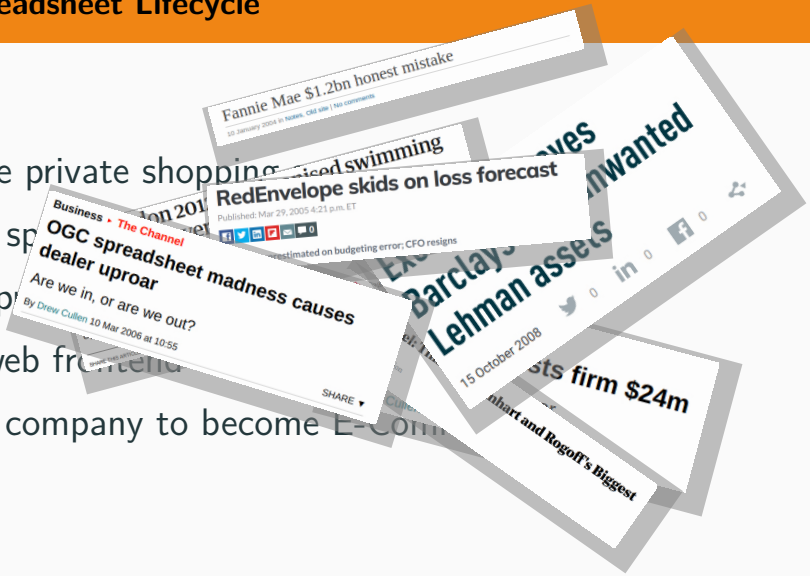
(Typical) Spreadsheet Lifecycle

1. Create private shopping [illegible]sed swimming
2. Show spread
3. Use spread
4. Put web frontend on sprea
5. Pivot company to become E-Com



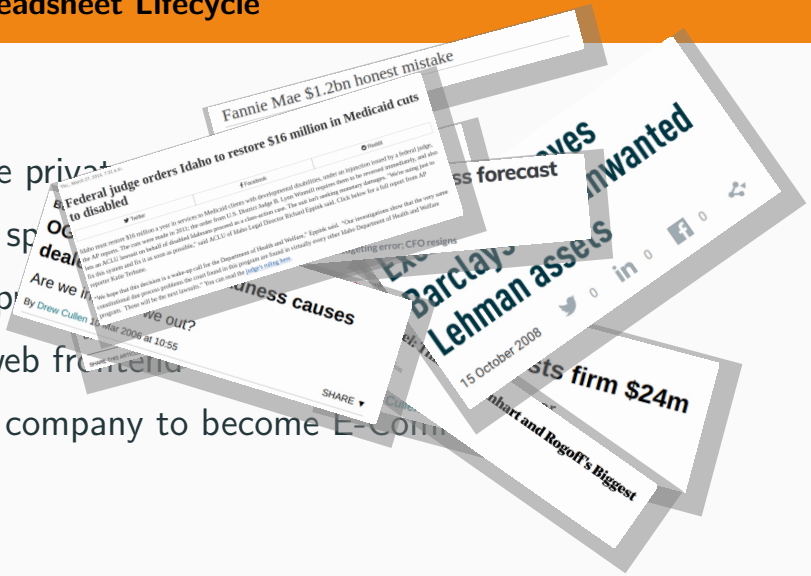
(Typical) Spreadsheet Lifecycle

1. Create private shopping [illegible] swimming
2. Show sp[illegible]
3. Use sp[illegible]
4. Put web fr[illegible]
5. Pivot company to become E-Com[illegible]



(Typical) Spreadsheet Lifecycle

1. Create priv
2. Show sp
3. Use sp
4. Put web fr
5. Pivot company to become E-Com

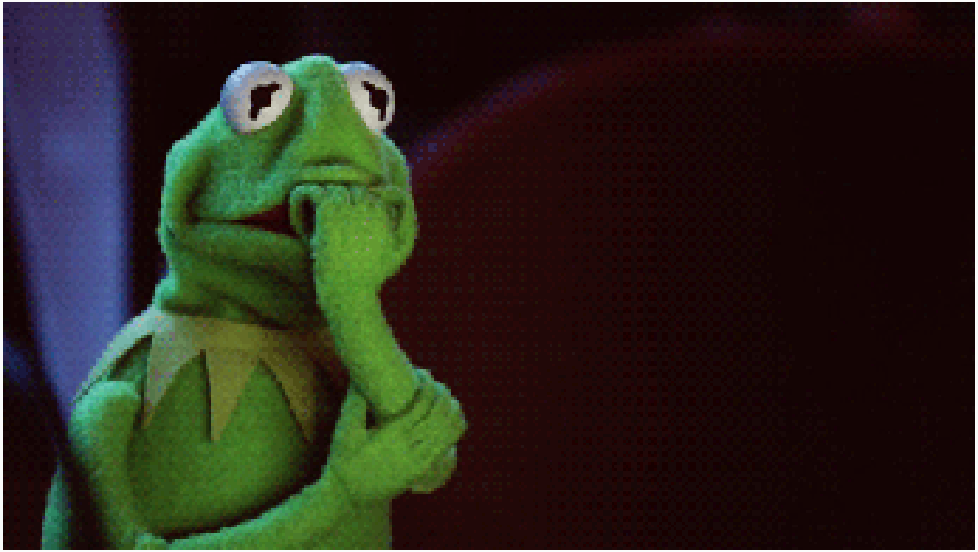


(Typical) Spreadsheet Lifecycle

1. Create pr
2. Show sp
3. Use sp
4. Put web
5. Pivot co

Spreadsheets rule the world!

How you should be feeling right now



Two assumptions going forward

1. You believe that spreadsheets rule the world.
2. You want D to rule the world instead.

How are we going to win this?

How are we going to win this?

We are not!

Lets draw up a battle plan

Lets take stock of what we have

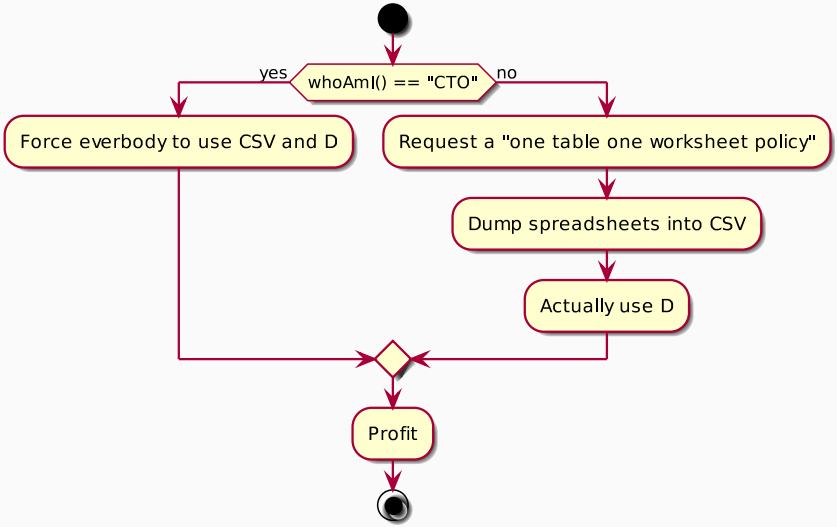
- Too many spreadsheets
- Too many tasks
- Too little man-power

Lets take stock of what we have

- Too many spreadsheets
- Too many tasks
- Too little man-power

- Millions of lines of source in different languages
- D

Possible Attack Vectors



How to work with limited man-power

Writing data to spreadsheets

Leveraging existing libraries

Writing data to spreadsheets

- Is required, people will ask for that
- Writing a somewhat feature complete xlsx writer is a huge task

Leveraging existing libraries

Writing data to spreadsheets

- Is required, people will ask for that
- Writing a somewhat feature complete xlsx writer is a huge task

- libxlsxwriter is a feature rich xlsx writer
- Wrapping it by hand,

Leveraging existing libraries

Writing data to spreadsheets

- Is required, people will ask for that
- Writing a somewhat feature complete xlsx writer is a huge task

- libxlsxwriter is a feature rich xlsx writer
- Wrapping it by hand, no way (+78000 lines of structs, enums and functions)

Leveraging existing libraries

Writing data to spreadsheets

- Is required, people will ask for that
- Writing a somewhat feature complete xlsx writer is a huge task

- libxlsxwriter is a feature rich xlsx writer
- Wrapping it by hand, no way (+78000 lines of structs, enums and functions)

- dpp to the rescue
- libxlsxwriter.d (+4000 lines)
- But it is still a C api

dpp output

```
1 void chart_axis_set_name(lxw_chart_axis*, const(char)*)
2 void chart_axis_set_name_font(lxw_chart_axis*, lxw_chart_font*)
3 void chart_axis_set_num_font(lxw_chart_axis*, lxw_chart_font*)
4 void chart_axis_set_num_format(lxw_chart_axis*, const(char)*)
5 void chart_axis_set_line(lxw_chart_axis*, lxw_chart_line*)
6 void chart_axis_set_fill(lxw_chart_axis*, lxw_chart_fill*)
7 ...
```

Semi-automatic refactoring

```
1  struct ChartAxis {
2    lxw_chart_axis* handle;
3
4    void setName(string name) {
5      chart_axis_set_name(this.handle, toStringz(name));
6    }
7
8    void setNameRange(string name, lxw_row_t row,
9        lxw_col_t col)
10   {
11     chart_axis_set_name(this.handle, toStringz(name), row,
12         col
13       );
14   }
15   ...
16 }
```

Creating fake data

Creating fake data

Problem to solve: We needed fake data with a variety of attributes.

- Name
- Address
- i18n
- ...


JS

- It hurts me to say, but just look at the packages on npmjs.

faker.js

- +160 attributes
- 39 languages

```
module["exports"] = [
  "#{prefix} #{first_name} #{last_name}",
  "#{first_name} #{nobility_title_prefix} #{last_name}",
  "#{first_name} #{last_name}",
  "#{first_name} #{last_name}",
  "#{first_name} #{last_name}",
  "#{first_name} #{last_name}"
];
```

Listing 1: locales/de/name/name.js

FakeD [7]

```
override string nameName() {
  switch(uniform(0, 6, this.rnd)) {
    case 0:
      return format!"%s %s %s"(namePrefix(), nameFirstName(),
          nameLastName());
    case 1:
      return format!"%s %s %s"(nameFirstName(), nameNobilityTitlePrefix(),
          nameLastName());
    case 2:
      return format!"%s %s"(nameFirstName(), nameLastName());
    case 3:
      return format!"%s %s"(nameFirstName(), nameLastName());
    case 4:
      return format!"%s %s"(nameFirstName(), nameLastName());
    case 5:
      return format!"%s %s"(nameFirstName(), nameLastName());
    default: assert(false);
  }
}
```

FakeD

```d
import faked;

auto f = new Faker(1337);
writeln(f.nameName());

// localized to german
f = new Faker_de(1338);
writeln(f.nameName());
```

FakeD

- Input:
  - Parser and Generator $\approx$ 1500 lines of D
  - A day of labor

FakeD

- Input:
  - Parser and Generator $\approx$ 1500 lines of D
  - A day of labor
- Output:
  - Output feature equivalent $\approx$ 70000 lines faker.js clone
  - Most changes in faker.js just require a rerun of the tool to update

FakeD

- Input:
  - Parser and Generator $\approx$ 1500 lines of D
  - A day of labor
- Output:
  - Output feature equivalent $\approx$ 70000 lines faker.js clone
  - Most changes in faker.js just require a rerun of the tool to update
- Bonus:
  - Created two PRs to faker.js fixing wrong template expansion

Taking a step back

The Wanted Output: Salary Table

Firstname	Lastname	Amount	Currency	CreatedBy
Hans	Meier	73331	USD	Ruth Ember
John	Doe	83431	GPB	Ruth Ember
Ruth	Ember	103431	EUR	Hans Meier

The starting point

```
class Employee {
  long id;
  DateTime createdAt;

  EmployeeInfo info;
  long infoId;
}

class EmployeeInfo {
  long id;
  string firstname;
  string lastname;

  Salary salary;
  long salaryId;
}
```

```
class Salary {
  long id;
  Employee createdBy;
  long createdById;

  CurrencyAmount amount;
  long amountId;
}

class CurrencyAmount {
  long id;
  double amount;

  Currency currency;
  long currencyId;
}
```

```
class Currency {
  long id;
  string name;
}
```

The vibe.d REST interface

```
1 interface Backend {
2   Employee[] getAllEmployees();
3   Employee getEmployee(long empId);
4   EmployeeInfo getEmployeeInfo(long empInfoId);
5   Salary getSalary(long salaryId);
6   CurrencyAmount getCurrencyAmount(long amountId);
7   Currency getCurrency(long currencyId);
8 }
```

The Frontend Code: Types

```ts
interface Employee {
  id: number;
  createdAt: number;

  info?: EmployeeInfo;
  infoId: number;
}

interface EmployeeInfo {
  id: number;
  firstname: string;
  lastname: string;

  salary?: Salary;
  salaryId: number;
}

interface Salary {
  id: number;
  createdBy?: Employee;
  createdById: number;

  amount?: CurrencyAmount;
  amountId: number;
}

interface CurrencyAmount {
  id: number;
  amount: number;

  currency?: Currency;
  currencyId: number;
}
```

The Frontend Code: Backend Service

```
1  import {
2      Employee, EmployeeInfo, Salary, CurrencyAmount, Currency
3  } from "model";
4
5  class Backend {
6      getAllEmployees() : Employee[] { ... }
7      getEmployee(empId: number): Employee { ... }
8      getEmployeeInfo(empInfoId: number): EmployeeInfo { ... }
9      getSalary(salaryId: number): Salary { ... }
10     getCurrencyAmount(amountId: number): CurrencyAmount { ... }
11     getCurrency(currencyId: number): Currency { ... }
12 }
```

The Frontend Code: Calling the Backend

```
this.backend.getAllEmployees().pipe(
  mergeMap((emps: Employee[]) => {
    const obs = [];
    for(const emp of emps) {
      obs.push(this.backend.getEmployeeInfo(emp.infoId)
        .pipe(map((empInfo: EmployeeInfo) => {
            const ne: Employee = {...emp, info : empInfo};
            return ne;
          })
        )
      );
    }
    return forkJoin(obs);
  }),
  mergeMap((emps: Employee[]) => {
    const obs = [];
    for(const emp of emps) {
```

The Communication

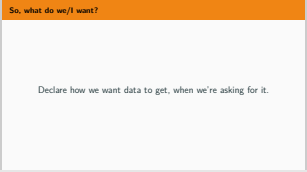
The Takeaways

- Clearly, this is unworkable
- Not plastic at all
- Just a lot of boring work

The real Takeway

So, what do we/I want?

Declare how we want data to get, when we're asking for it.



- We don't want the backend to dictate what data we can ask for.
- Going to the backend, more than once is not acceptable, bad networks and such.
- We want no Under-fetching.
- We want no Over-fetching.

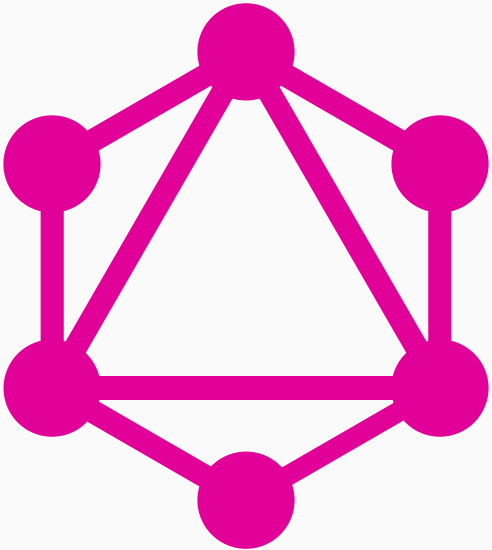
Why can't we write this?

```
{
  allEmployees {
    info {
      firstname
      lastname
      salary {
        amount
        currency {
          name
        }
        createdBy {
          info {
            firstname
            lastname
          }
        }
      }
    }
  }
}
```

Why can't we write this?

```
{
  allEmployees {
    info {
      firstname
      lastname
      salary {
        amount
        currency {
          name
        }
        createdBy {
          info {
            firstname
            lastname
          }
        }
      }
    }
  }
}
```

```
{
  allEmployees: [ {
      info: {
        firstname: "Hans",
        lastname: "Meier",
        salary: {
          amount: 73331,
          currency: {
            name: "USD"
          }
          createdBy: {
            info: {
              firstname: "Ruth",
              lastname: "Ember"
            }
          }
        }
      },
    },
    ...
```



GraphQL

```
schema {
  query: Backend
}

type Backend {
  getAllEmployees: [Employee]
}

type Employee {
  id: number!;
  createdAt: number!;

  info: EmployeeInfo;
  infoId: number!;
}
```

```
type EmployeeInfo {
  id: number!;
  firstname: String!;
  lastname: String!;
  salaryId: number!;
  salary: Salary;
}

type Salary {
  id: number!;
  createdBy: Employee;
  amountId: number!;
  amount: CurrencyAmount;
}
```

```
type CurrencyAmount {
  id: number!;
  amount: number!;
  currencyId: number!;
  currency: Currency;
}
type Currency {
  id: number!;
  name: string!;
}
```

Less code is better

```
query one {
  allEmployees {
    ...deep
  }
}

fragment names on EmployeeInfo {
  firstname
  lastname
}

fragment empInfo on Employee {
  info {
    ...names
  }
}
```

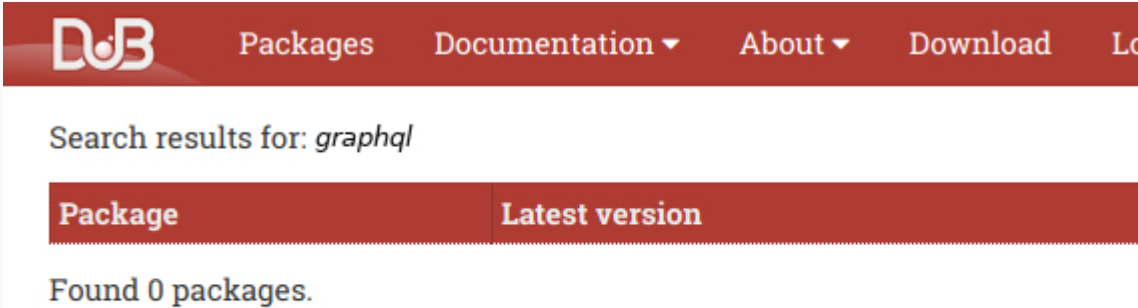
```
fragment deep on Employee {
  info {
    ...names
    salary {
      amount
      currency {
        name
      }
      createdBy {
        ...empInfo
      }
    }
  }
}
```

Introspecting Types

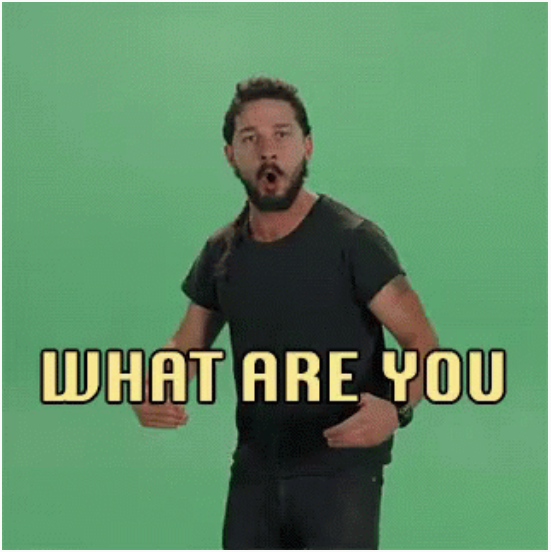
```
{
  __type(name: "Employee") {
    name
    fields {
      name
      type {
        name
        kind
        ofType {
          name
        }
      }
    }
  }
}
```

```
{
  "data": {
    "__type": {
      "name": "Employee",
      "fields": [
        {
          "name": "id",
          "type": {
            "name": null,
            "kind": "NON_NULL",
            "ofType" {
              "name": "INT"
            }
          }
        },
        {
          "name": "info",
          "type": {
            "name": "EmployeeInfo",
            "kind": "OBJECT"
```

D's GraphQL Story



JUST DO IT



GraphQLD [9]

```
1  import graphqld;
2
3  interface Query {
4    Employee[] getAllEmployees();
5  }
6
7  class Schema {
8    Query queryType;
9  }
```

GraphQLD

[9]

```
import graphqld;

interface Query {
  Employee[] getAllEmployees();
}

class Schema {
  Query queryType;
}
```

```
auto graphqld = new GraphQLD!(Schema)();

graphqld.setResolver(
  "queryType", "getAllEmployees",
  delegate(string name, Json parent,
      Json args, ref Context context) @safe
  {
    Employee[] employees = getAllEmployees();
    Json ret = Json.emptyObject();
    ret["data"] = toGraphqlJson(employees);
    return ret;
  });
```

GraphQLD

```
1  void graphqlEndpoint(HTTPServerRequest req,
2      HTTPServerResponse res)
3  {
4    string toParse = extractQuery(req);
5    auto p = Parser(Lexer(toParse));
6
7    Document d = p.parseDocument();
8    auto fv = new QueryValidator(d);
9    auto sv = new SchemaValidator!Schema(d, graphqld.schema);
10   fv.accept(d);
11   sv.accept(d);
12
13   Context con = buildContext(req);
14   Json ret = graphqld.execute(d, extractVariables(req), con);
15   res.writeJsonBody(ret);
16 }
```

GraphQLD

```
1  graphqld.setResolver("Employee", "info",
2    delegate(string name, Json parent, Json args,
3        ref Context context)
4    {
5      const id = parent["infoId"].get!long();
6      EmployeeInfo ei = getEmployeeInfo(id);
7      Json ret = Json.emptyObject();
8      ret["data"] = toGraphqlJson(ei);
9      return ret;
10   });
```

GraphQLD

- Mostly feature complete
- Some validations are missing
- $\approx$ 17000 lines
- $\approx$ 9000 lines are generated by darser
- ready for use now

Homework

Homework

Write a GraphQL backend that uses an excel spreadsheet as a database.

Conclusion

Conclusion

- Spreadsheets are a terrible programming language.

Conclusion

- Spreadsheets are a terrible programming language.
- C++ and Rust are not our main competition.

Conclusion

- Spreadsheets are a terrible programming language.
- C++ and Rust are not our main competition.

- We need to learn to use what is there.
- Use D to work smart not hard.
- Do not write the code, write the code that writes the code.
- Look at JS for inspiration.
- GraphQL 👍

The End

[1] *Infographic: C/C++ facts we learned before going ahead with CLion*. https://blog.jetbrains.com/clion/2015/07/infographics-cpp-facts-before-clion/. (Accessed on 04/08/2019).

[2] Developer Economics. *Developer Economics: State of the Developer Nation 15th Edition*. https://www.developereconomics.com/reports/state-of-the-developer-nation-15th-edition. (Accessed on 04/08/2019).

[3] Microsoft. *Build 2016 Keynote (Day 2)*. https://www.youtube.com/watch?v=bf0Rr81is6U. (Accessed on 04/08/2019).

[4] Irish Tech News. *Seven reasons why Excel is still used by half a billion people worldwide*. https://irishtechnews.ie/seven-reasons-why-excel-is-still-used-by-half-a-billion-people-worldwide/. (Accessed on 04/08/2019).

[5] Felienne Hermans. *GOTO 2016: Pure Functional Programming in Excel by Felienne Hermans*. https://www.youtube.com/watch?v=0yKf8TrLUOw. (Accessed on 04/08/2019).

[6] *Marak/faker.js: generate massive amounts of realistic fake data in Node.js and the browser*. https://github.com/marak/Faker.js/. (Accessed on 05/03/2019).

[7] *kaleidicassociates/faked: D library to create real fake data*. https://github.com/kaleidicassociates/faked. (Accessed on 05/03/2019).

[8] *GraphQL | A query language for your API*. https://graphql.org/. (Accessed on 05/03/2019).

[9] *burner/graphqld: A vibe.d library to handle the GraphQL Protocol written in the D Programming Language*. https://github.com/burner/graphqld. (Accessed on 05/03/2019).

[10] *burner/Darser: LL1 Parser Generator for D*. https://github.com/burner/Darser. (Accessed on 05/03/2019).

Encore

Reappearing UDA Pattern

```
1  struct Employee {
2    @GQLD(
3      Description("The social security number of an employee"),
4      Deprecated(IsDeprecated.yes, "To complex")
5    )
6    SocialSecurityNumber number;
7  }
```

Reappearing UDA Pattern

```
1  struct Employee {
2    @GQLD(
3      Description("The social security number of an employee"),
4      Deprecated(IsDeprecated.yes, "To complex")
5    )
6    SocialSecurityNumber number;
7  }

9  enum IsDeprecated {
10   undefined,
11   no,
12   yes
13 }
14
15 struct GQLDData {
16   Description desc;
17   Deprecated depre;
18 }
```

Reappearing UDA Pattern

```
1  struct GQLDData {
2    Description desc;
3    Deprecated depre;
4  }
5
6  GQLDData GQLD(Args...)(Args args) {
7    GQLDData ret;
8    static foreach(mem; __traits(allMembers, GQLDData)) {
9      static foreach(arg; args) {
10       static if(is(typeof(__traits(getMember, ret, mem)) ==
11           typeof(arg)))
12       {
13         __traits(getMember, ret, mem) = arg;
14       }
15     }
16   }
17   return ret;
```

Static Foreach Switch Case

```
Json ret = Json.emptyObject();
string typename = ...;
l: switch(typename) {
  static foreach(type; collectTypes!(T)) {{
    case typeToTypeName!(type): {
      ret["data"] = typeToJson!(type)();
      break l;
    }
  }}
  default: break;
}
return ret;
```

Collecting all Referenced Types

```
1  alias allTypes = collectTypes!Schema;
2
3  template collectTypesImpl(Type) {
4    import graphql.uda;
5    static if(is(Type : GQLDCustomLeaf!F, F)) {
6      alias collectTypes Impl= AliasSeq!(Type);
7    } else static if(is(Type == interface)) {
8      alias RetTypes = AliasSeq!(collectReturnType!(Type,
9          __traits(allMembers, Type)));
10     alias ArgTypes = AliasSeq!(collectParameterTypes!(Type,
11         __traits(allMembers, Type)));
12     alias collectTypesImpl = AliasSeq!(Type, RetTypes,
13         ArgTypes, InterfacesTuple!Type);
14   ....
15   } else static if(is(Type == union)) {
16     alias collectTypesImpl = AliasSeq!(Type, InheritedClasses!Type);
17   } else static if(is(Type : Nullable!F, F)) {
18     alias collectTypesImpl = .collectTypesImpl!(F);
19   ...
```

Compile-Time are long as

- $\approx$ 7000 lines
- $\approx$ 8 seconds build time

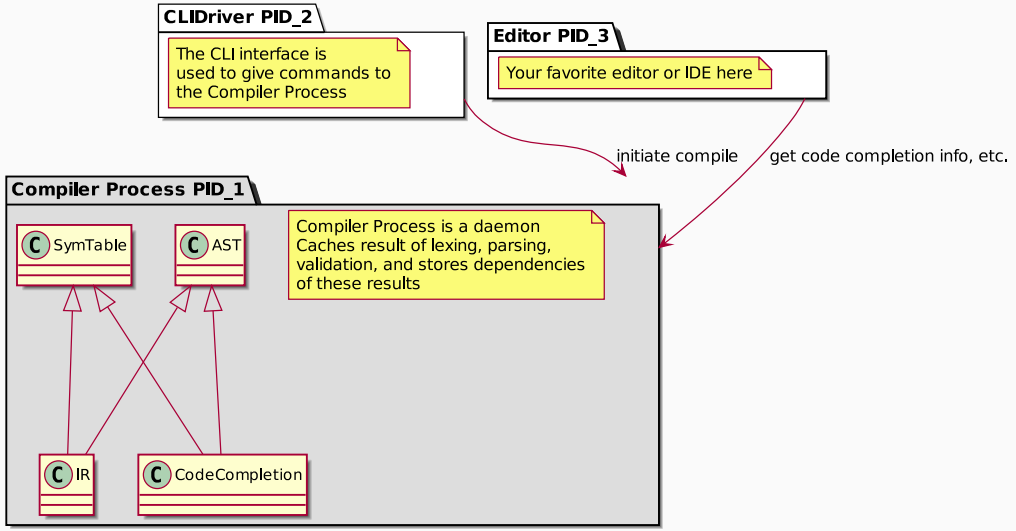
Overall Legacy Architecture

- Traditional compiler pipeline design is dated

Overall Legacy Architecture

- Traditional compiler pipeline design is dated
- We have practically unlimited memory
- Recreating the AST, IR, and ASM on every compile is extremely wasteful
- Why does code-completion and the compiler different frontends

Possible Compiler Re-arch



- Darser is a recursive descent parser generator for LL(1) grammars
- It also generates the AST and a default Visitor
- Not at CT, but as a pre-build step

- Darser is a recursive descent parser generator for LL(1) grammars
- It also generates the AST and a default Visitor
- Not at CT, but as a pre-build step

- It generates “good” error messages
- Not just a generic Node types, but names that reflect the grammar
- Inheriting from the default Visitor is trivial and powerful
- Used right now by GraphQLD

I'm out of Slides