



.NET Conf

Focus on F#

July 29, 2021

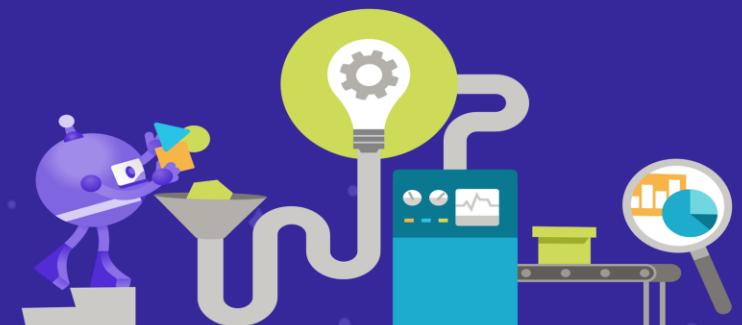
focus.dotnetconf.net

DEMETRIX



Into the Land of Biotech, with F# as an Ally

Olya Samusik | @w0lya
Software Engineer at Demetrix, Inc

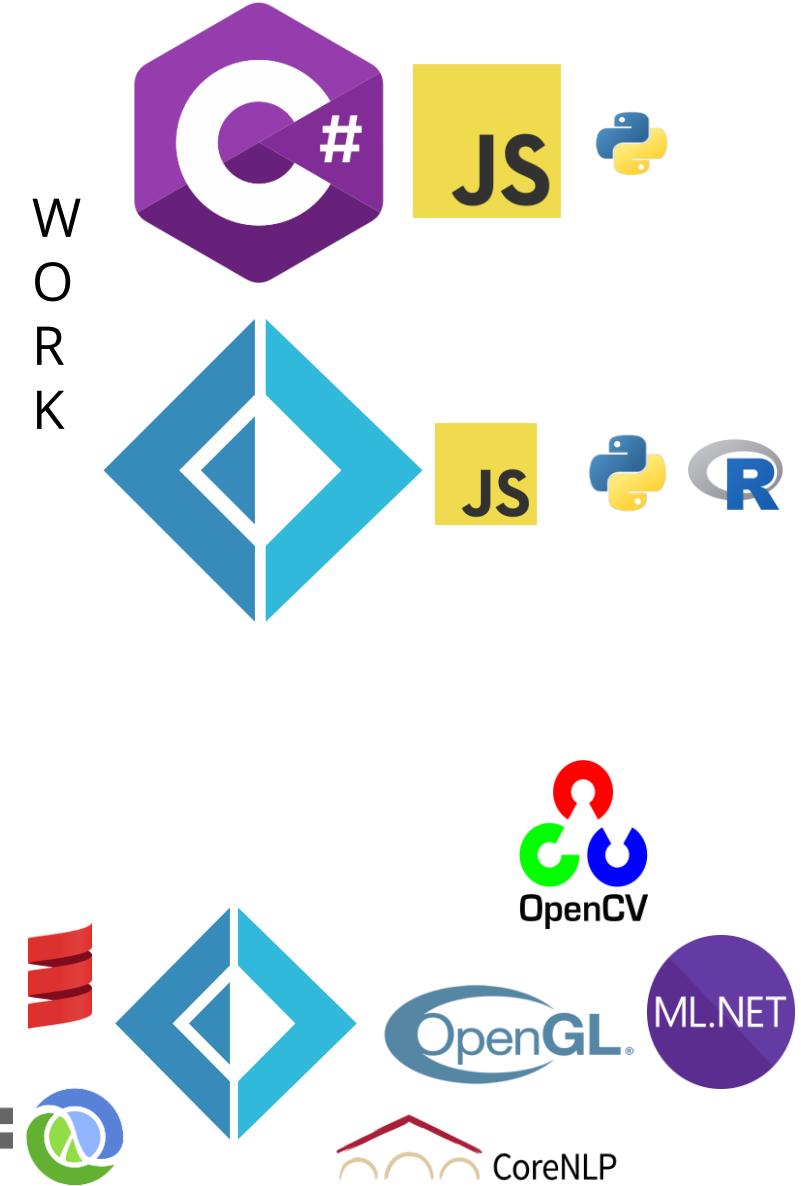


Thanks!

- Demetrix
 - Software Engineers
 - Data Scientists
 - IT
 - Scientists
- Dotnet Conf, Focus on F# organizers
 - F# Community
 - Don Syme - [Language + Type provider support](#)
 - Alfonso Garcia-Caro - [Fable and solid advice](#)
 - Dustin Morris Gorski - [Giraffe](#)
 - Maxime Mangel - [Elmish](#)
 - Postgres Npgsql maintainers - [improvements in type safety](#)
 - SAFE stack maintainers
 - Elliot Menschik, Paul Underwood - [AWS Biotech support](#)
 - Krzysztof-Cieslak - [Ionide](#)
 - Matthias Brandewinder - [Recruiting](#)
 - Chris Macklin - [Genotype Specification Language collaboration](#)
 - Cody Johnson - [Feliz.Plotly](#)
 - a lot of other people...

About me

- Software engineer
 - 9 y of (mostly) C# coding by day at work, ~5y of occasional F# coding by night
 - 2 y of full stack F# at work! And trying to wrap my head around biotech
 - computer graphics / vision | ML, data science | ...
- Contact info
 - w0lya on Twitter, GitHub, ...
 - olya@demetrix.com

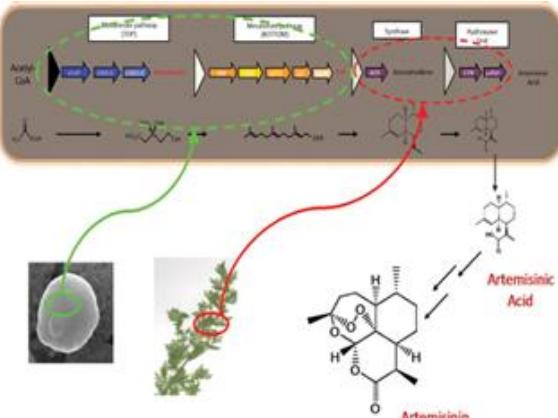
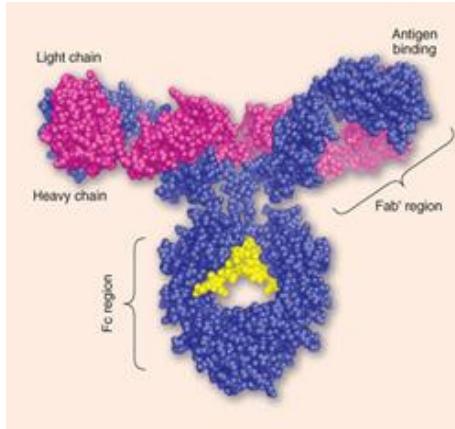


Plan

- Quick peek at biotechnology
- Demetrix, what do we do
- Challenges we face, usual / unusual
- Our technical landscape
- Software we build
- Why F#, how it helps us
- Problems and solutions
- Lessons learned, opportunities for improvement
- Q&A



Biotech - the future is now!



Prototype disk drive writing data into DNA
Catalog DNA

Therapeutic antibodies (cancer , infectious disease)

IMPOSSIBLE™

Alternative foods.
Heme molecule

BIO MASON

Biosynthesized
construction materials, e.g. cement

BOLT THREADS

Modern Meadow

Biosynthesized
sustainable materials, e.g.
leather



SEIZE THE DAY AFTER LAST NIGHT

ZBiotics® is a probiotic engineered by a team of PhD scientists who know three things: microbiology, the next-day effects of alcohol, and how microbiology can prepare our bodies to better handle alcohol.

ZBiotics - hangover prevention
(bacteria that eats acetaldehyde)

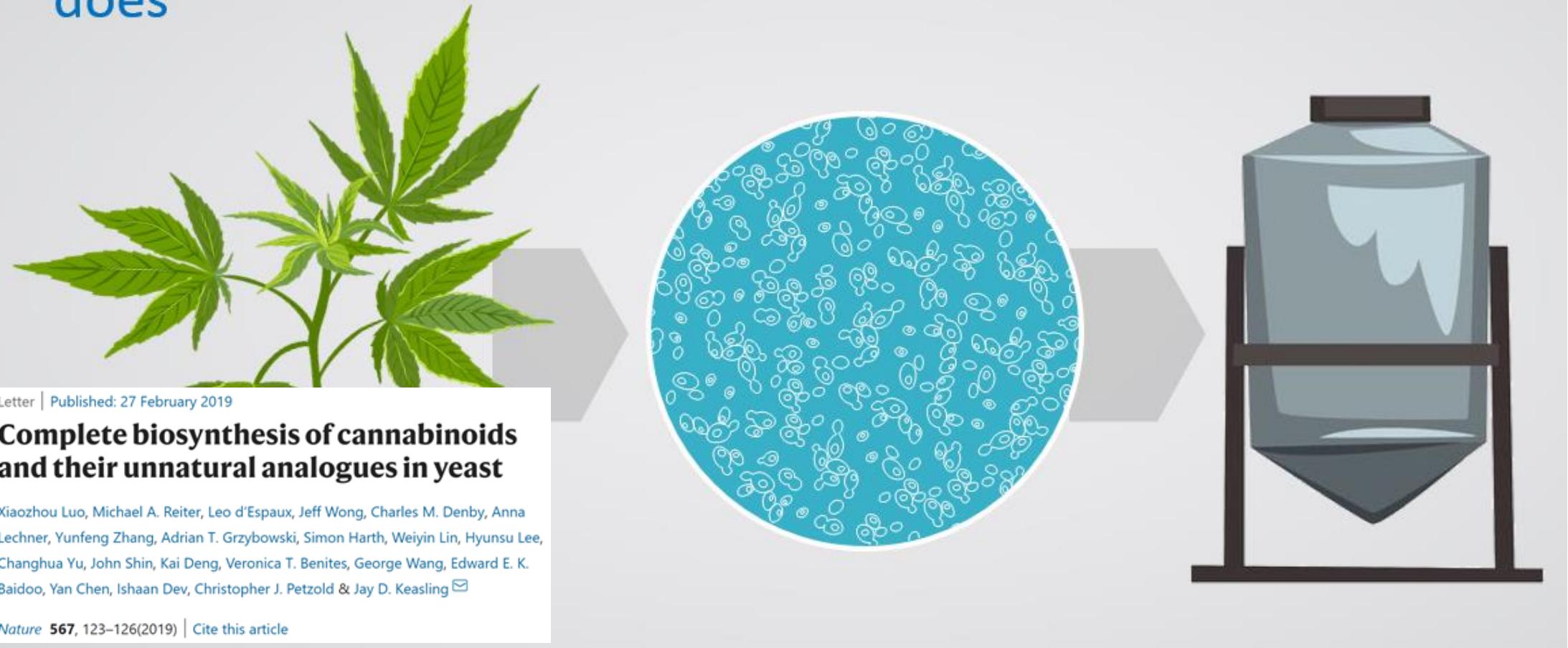


I Grew Real Spider Silk
Using Yeast

Justin Atkin, DIY enthusiast (The Thought Emporium)

What does Demetrix do?

Demetrix takes DNA sequences from plants and puts them into yeast – yeast now makes what the plant does



Panoramic view

Teams

Strain Engineering

Design and build yeast strains, aiming to optimize production

DNAOps

Make the process of Strain Engineering efficient and consistent

USP (UpStream Processing)

Engineer robust and scalable fermentation processes that allow for optimal strain performance

DSP (DownStream Processing)

Develop and support processes for extraction, separation, purification and preservation of the products

HTPE (High-Throughput Process Engineering)

Characterize, optimize and execute scaled down processes, perform strain screening

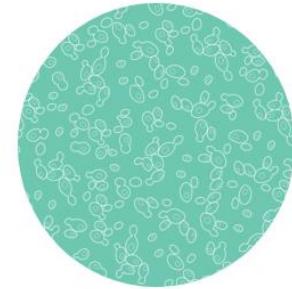
HTS (High Throughput Screening)

Auto-testing large numbers of compounds for a specific biological target

Analytics

Develop assays and process samples to get qualitative (which ones) and quantitative data (concentration) about analytes

Two avenues necessary for success



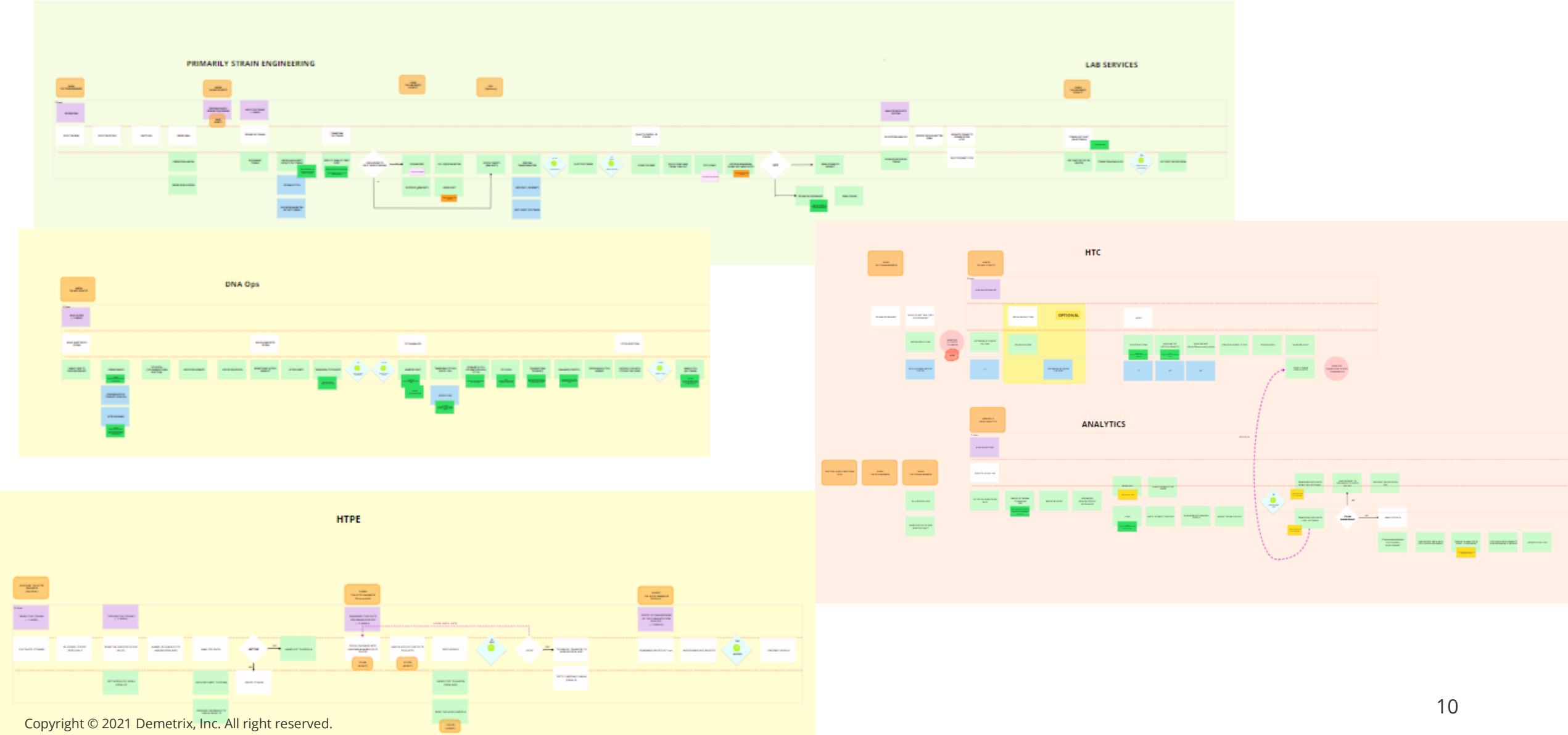
R&D/Strain Engineering

Manufacturing/Process Engineering

Let's zoom in a bit... (partial) real world view



Let's zoom in a bit... (partial) logical view. Complex!

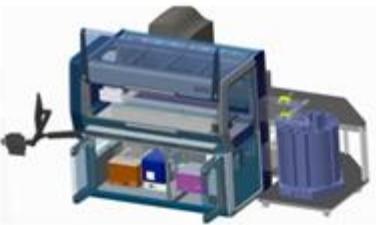


Some of the challenges we face

- Modelling multiple complex domains
- Gathering, understanding, prioritizing requirements
- Enabling maximum user engagement & involvement in building software (for their own good!)
- Addressing rapidly evolving requirements
- Gathering, structuring, making use of a lot of heterogeneous data
- Ensuring secure access for humans and robots
- Building nice UI to help users solve unconventional problems
- Being efficient regardless of being spread thin across different areas
- Web server
- Hosting
- Prototyping
- Testing
- Iterations
- CI / CD
- Maintenance
- Error handling
- Logging
- Batch, recurring tasks
- Reliable and performant data storage
- Reusing functionality, infrastructure across applications

Tech landscape

User interfaces
Data visualization
Lab / Robot data services



Auth



Hosting



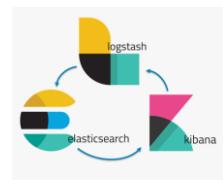
Hangfire
(batch tasks)



User services
– mail / docs



Metrics



Logging

```
1 // Yeast terpene design from Paddon et al., Wilson et al.
2 // Level 1 syntax
3 // Part definitions
4 #refgenome cnpk
5 let pGAL1_10 = gGAL1[-668:-1]
6 let trunGMR = /ATGG/ {#nabitstart} ; gGAL1[1586:-200E]
7
8 let cassette(locus,gene1, gene2) =
9   &locus.up ; ### ; !gene1 ; &pGAL1_10 ; !gene2 ; &locus[-500:-500]
10 end
11
12
13 // Locus engineering
14 uERG9 ; ### ; pMER3 ; gERG9[1:-500]
15 cassette(gLEU2,mERG8,mWD01)
16 cassette(gHIS3,mERG10,mERG12)
17 cassette(gADE1,mID11,&trunGMR)
18 cassette(gURA3,mERG13,&trunGMR)
19 cassette(gTRP1,&trunGMR,mERG20)
20
21
22
```

Compiler output



ASP.NET Core

Server

```
166
167
168 bool -> int -> GSLEditorState -> Task<Either<GSLEditorState>>
169 let compileGSLDocument overwrite (userId : int) (gslEditorState : GSLEditorState) =
170   taskEither {
171     let compilerURL = if AppConfig.Value.AuthConfig.Offline then "" else AppConfig.Value.GslCompilerServiceUrl
172
173     let! validation = GSLDocumentValidationService.validate overwrite gslEditorState.Document userId
174     let! hashedContent = hashASCIIBySHA256 gslEditorState.Document.Content
175
176     let gslEditorState =
177       { gslEditorState with
178         Document =
179           { gslEditorState.Document with
180             LastUpdated = DateTime.Now
181             HashedContent = hashedContent
182           }
183       }
184
185     let! saveStateResult = saveGSLEditorState gslEditorState
186     let compilerTask = GslCompilerService.invoke
187       compilerURL
188       (sprintf "gslDocId" gslEditorState.Document.Id)
189       gslEditorState.Document.Content
190       gslEditorState.Document.CompilerFlags
191
192     let compilerResponseResult = compilerTask > Task.runSynchronously
193   }
194 }
```

Full stack F#



PostgreSQL



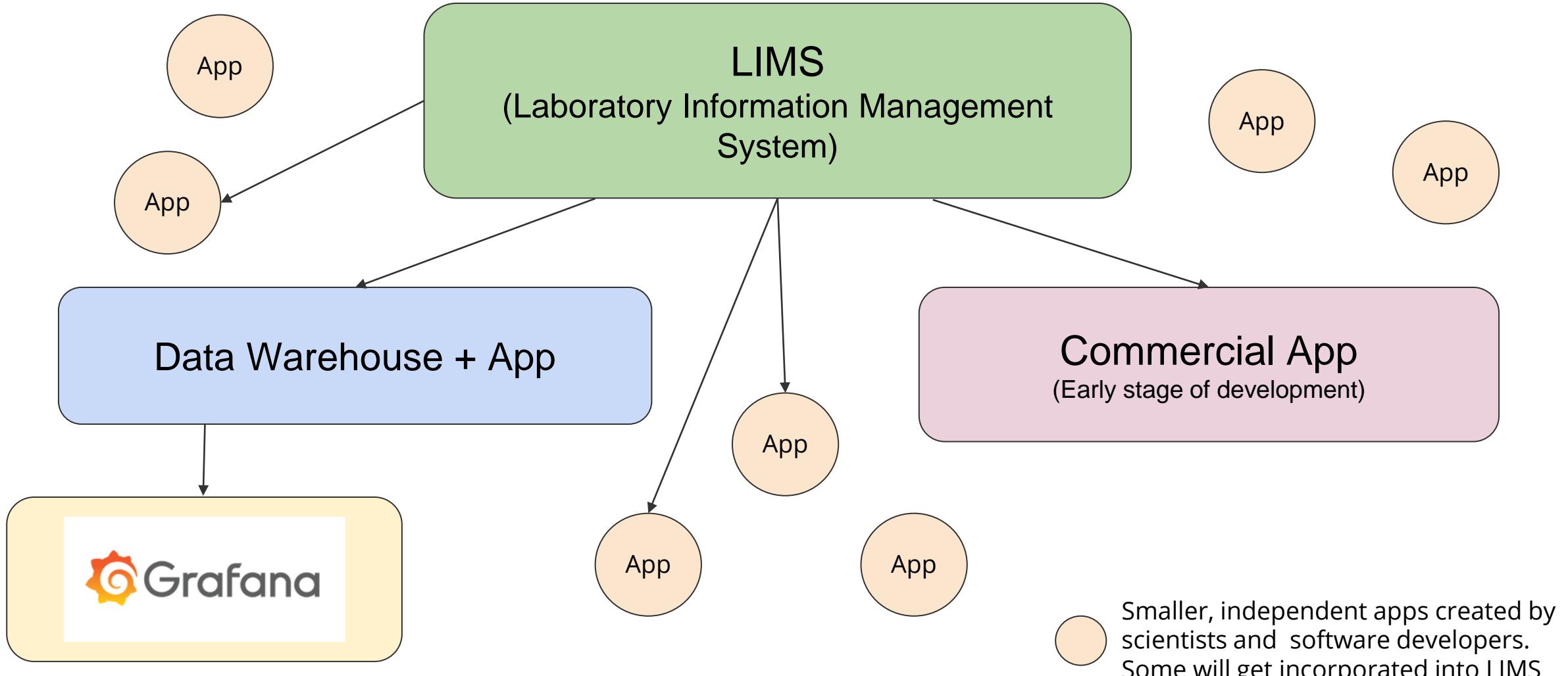
FAKE (F# make)

PAKET₁₂

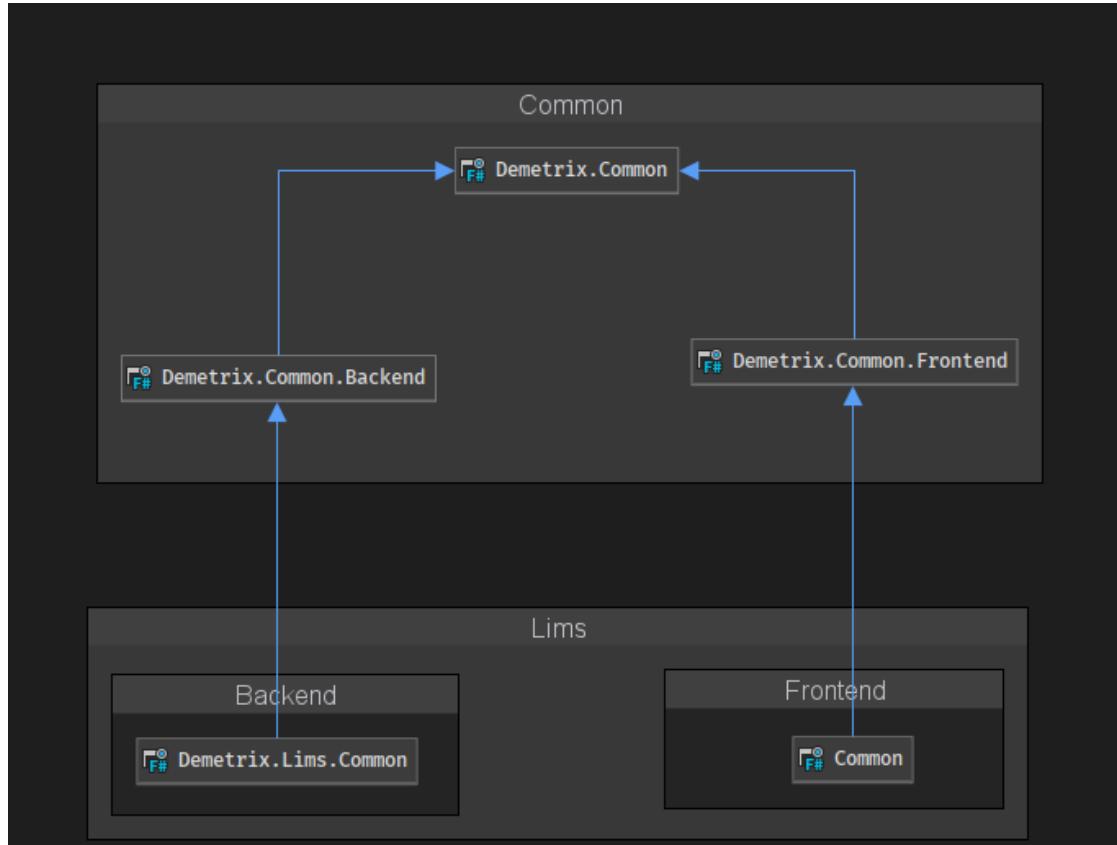


Shared
code and
data definitions

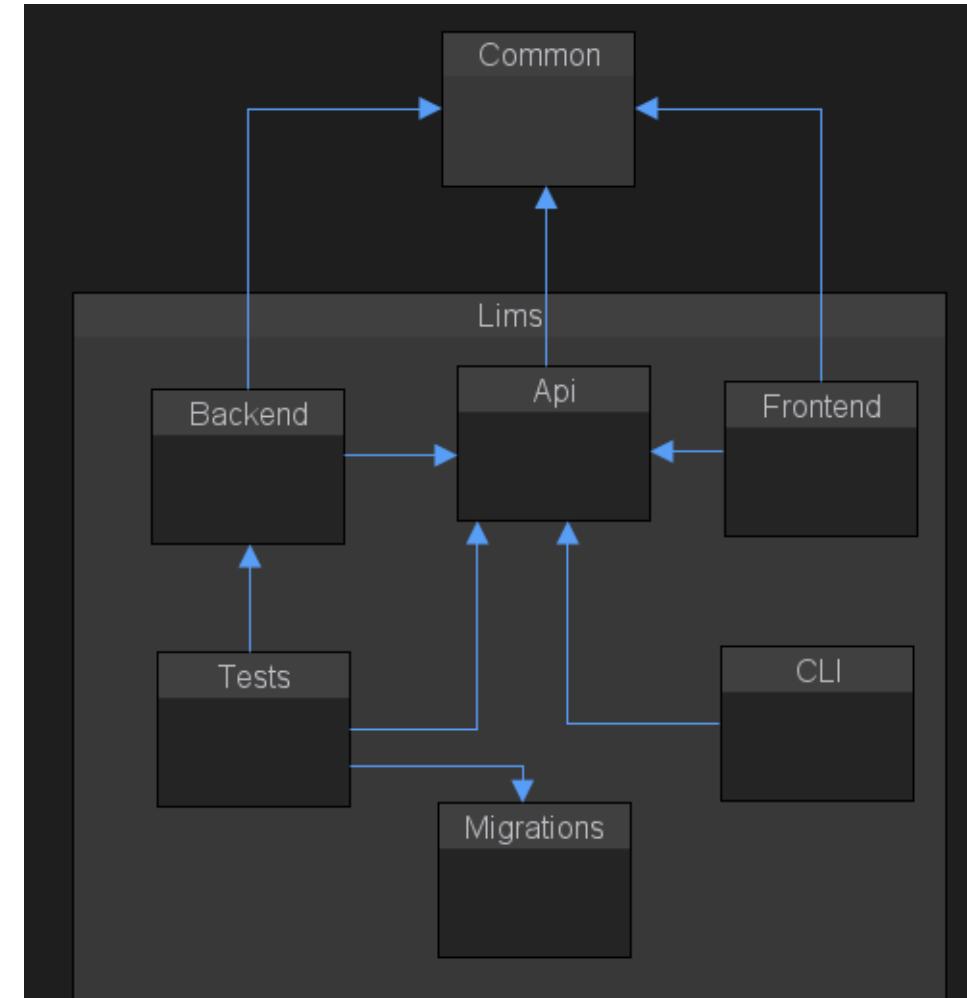
Demetrix software world



A bit about code structure



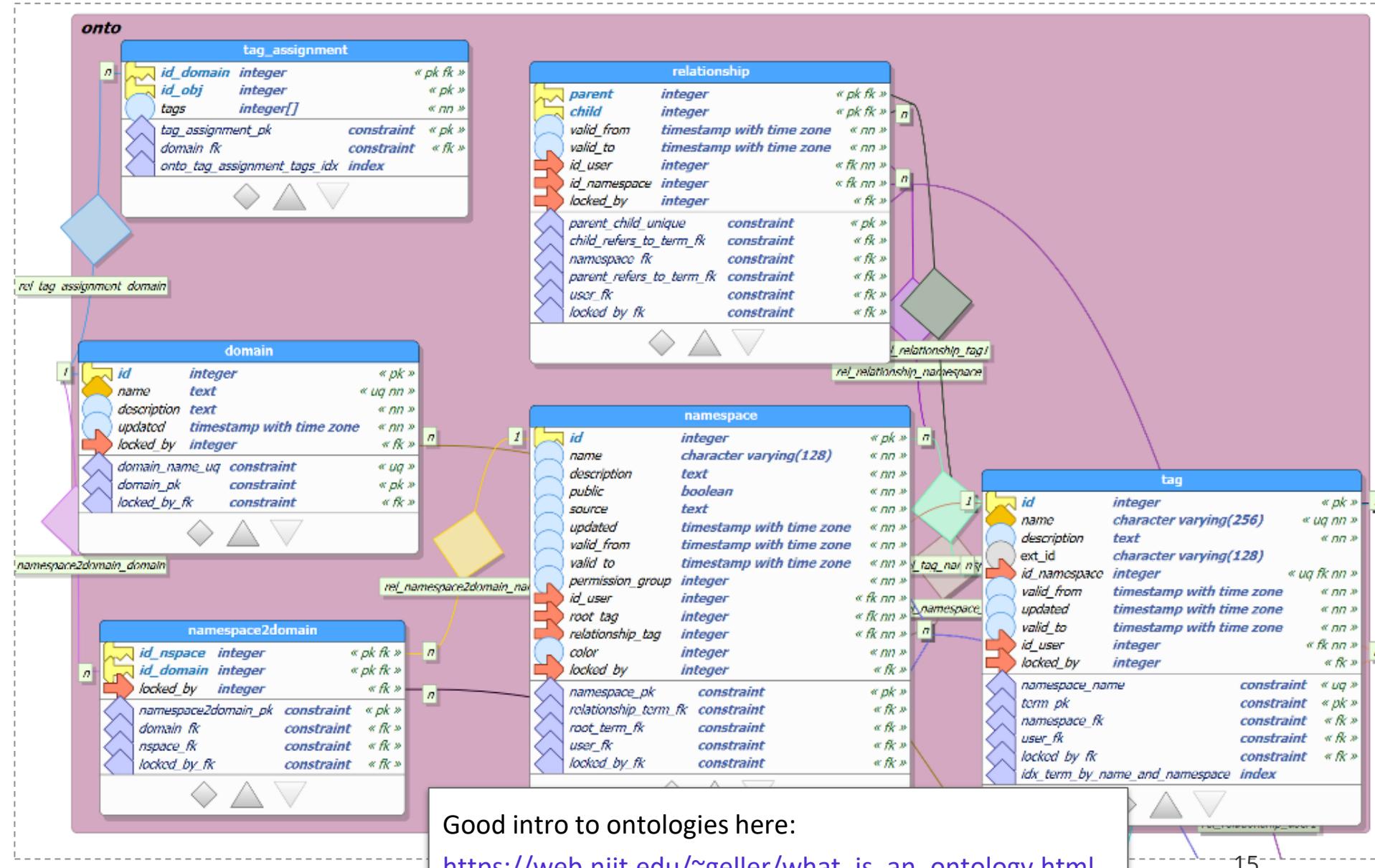
What common code sharing looks like for any of our apps



A look at an individual app and its code parts

Ontologies to represent domain metadata

- ▼ vessel_type root tag
 - 250mL Bottle
 - 500µL Assay Plate
 - 50mL Conical Tube
 - Assay media
 - Bottle 500mL
 - PC media
 - PR media
 - Vial
- ▼ chemical_vendor
 - Demetrix Media Prep
 - Elga LabWater
 - Fisher Chemical
 - Sigma-Aldrich
 - VWR
- ▼ demetrix_lab
- ▼ equipment
 - LidValet
 - Microserve
 - freezer
- ▼ liquid_transferer
 - Echo
 - Stellaluna
 - ▼ Vantage
 - Babe
 - 8channel
 - channel1
 - channel2
 - channel3



LIMS - a federated set of Fable applications

DEMETRIX Administrator ▾ Inventory ▾ DNA Building ▾ Strain Building ▾ Experiment Planner ▾ Ontology Browser Transition Tracking

WHAT'S NEW?

Demetrix Applications

-  **GSL Documents**
Add or edit GSL documents
-  **Strain Designer**
Add or edit strains
-  **Ontology Browser**
Ontology Browser
-  **Transition Tracking**
Transition Tracking
-  **Stitch Gallery**
Beta application, information may be incorrect
-  **Data Visualization**
Post-run data visualization
-  **Crop Builder**
Add or edit crops
-  **High-throughput Screening**
Experiment Planner
-  **Strain Bank Lookup**
Lookup banked strains
-  **Chemical Inventory**
Chemical Inventory
-  **Strain Gallery**
Lookup strain lineage information
-  **Hangfire Processing**
Hangfire jobs dashboard

Administrator Applications

-  **Ontology Domain**
Ontology Domain
-  **Command Line Application Downloads**
 -  [File Upload Client](#)
 -  [Robot Worklist Client](#)
-  **Barcode Manager**
Barcode Manager

DNA Designer application

DEMETRIX Administrator ▾ Inventory ▾ DNA Building ▾ Strain Building ▾ Experiment Planner ▾ Ontology Browser Transition Tracking User Olya Samusik ▾

SAVE ▶ COMPILE Crop32_RB read-only locked

// Yeast terpene design from Paddon et al, Wilson et al.
// Level 1 Syntax
// Part definitions
#refgenome cenpk
let pGAL1_10 = gGAL1[-668:-1]
let truncHMGR = /ATGG/ {#rabbitstart} ; gHMGR[1586:~200E]

let cassette(locus,gene1,gene2) =
| &locus.up ; ### ; !&gene1 ; &pGAL1_10 ; &gene2 ; &locus[-500:~500]
end

// Locus engineering
uERG9 ; ### ; pMET3 ; gERG9[1:~500]
cassette(gLEU2,mERG8,mMVD1)
cassette(gHIS3,mERG10,mERG12)
cassette(gADE1,mIDI1,&truncHMGR)
cassette(gURA3,mERG13,&truncHMGR)
cassette(gTRP1,&truncHMGR,mERG20)

Genotype Specification Language
Erin H. Wilson, Shiori Sagawa, James W. Weis, Max G. Schubert, Michael Bissell, Brian Hawthorne, Christopher D Reeves, Jed Dean, and Darren Platt*
Amyris, Inc., 5885 Hollis Street, Suite 100, Emeryville, California 94608, United States

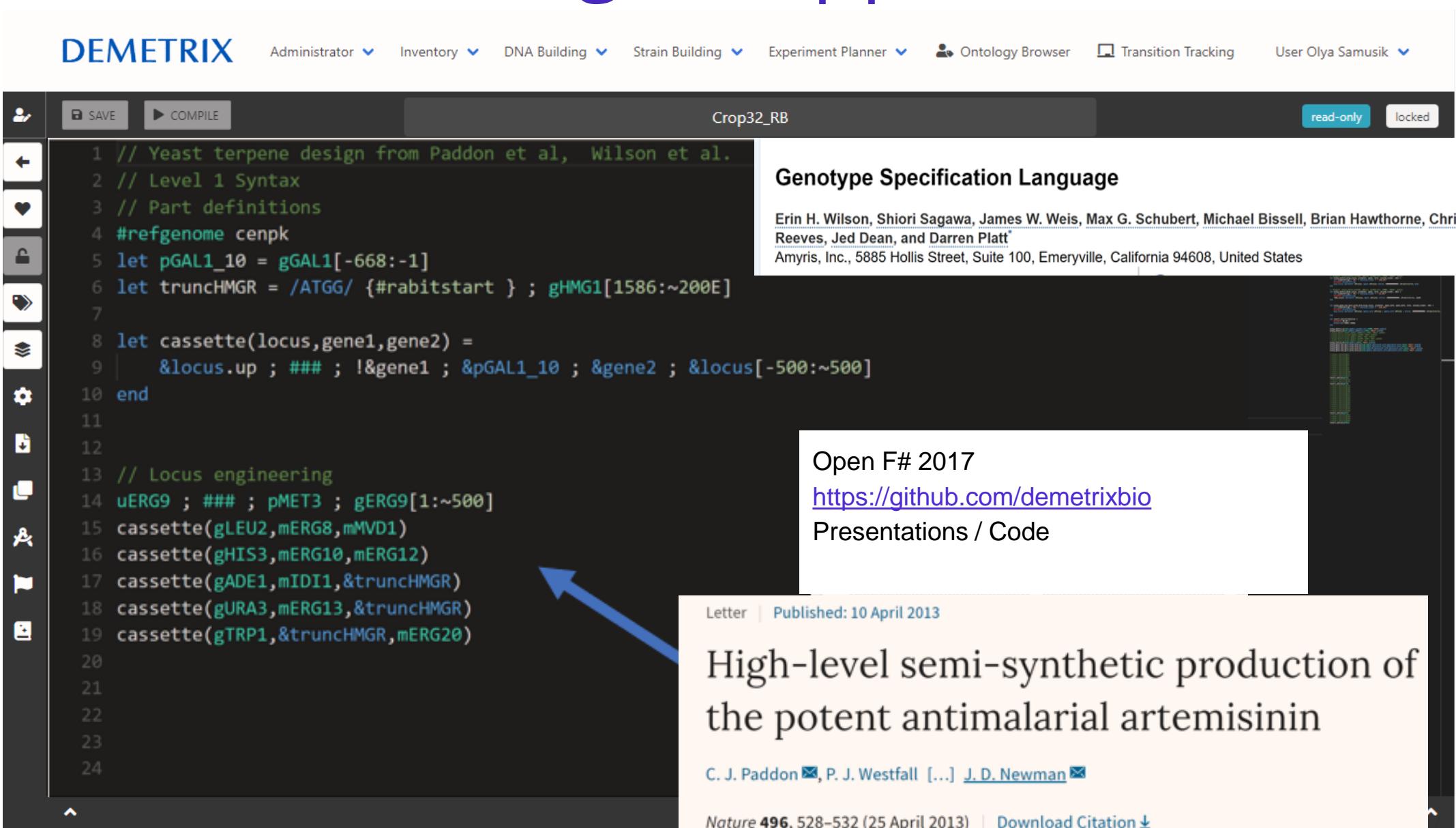
Open F# 2017
<https://github.com/demetrixbio>
Presentations / Code

Letter | Published: 10 April 2013

High-level semi-synthetic production of the potent antimalarial artemisinin

C. J. Paddon ✉, P. J. Westfall [...] J. D. Newman ✉

Nature 496, 528–532 (25 April 2013) | Download Citation ↴



DNA Building

Compiled design from previous page

Included designs

DNA parts needed for construction of design

The screenshot shows the Demetrix DNA Building software interface. At the top, there's a blue header bar with the text "Compiled design from previous page". Below it, a red arrow points from the "Included designs" section to the "DNA parts needed for construction of design" section. The "Included designs" section shows a table with one row, circled in red. The table has columns for #, Title, and Owner. The row shows #48, Title "gsl paper demo", and Owner "Darren Platt". Another red arrow points from the "DNA parts needed for construction of design" section to the "REVERSEPRIMERS" section in the main workspace. The workspace displays a grid of DNA parts labeled A through G across rows REV, PLATE, and 1. The "REVERSEPRIMERS" section shows a list of primers: REVERSEREPRIMERS, REV_PLATE_1, and REV_PLATE_2. The "Included designs" section also includes a "Statistics" panel with counts for Upstreams, Downstreams, ORFs, Primers, Kernels, Linkers, PCRs, Stitches/Stalks, and Megastitches/Plants. The "DNA parts needed for construction of design" section includes a "Crop summary" and a "Pipeline progress" section with a checklist of tasks.

DEMETRIX Build Applications Crop Editor: open_fsharp_demo_crop User Darren

#50 open_fsharp_demo_crop

GSL Documents

Search Title

#	Title	Owner
47	Crop8_20180926_Leo	Darren Platt
46	crop 8 control stitches	Darren Platt
45	crop7 final	Darren Platt
39	crop6_control_stitches	Darren Platt

Included designs

#	Title	Owner
48	gsl paper demo	Darren Platt

REVERSEPRIMERS

REV_PLATE_1

	1	2	3	4	5	6	7	8	9	10	11	12	13
A	uERU2_REV	gERG9[15~500]_S1_REV	uADE1_REV	uURA3_REV	uERG9_REV	uTRP1_REV	uHS3_REV	pMET3_REV	gGAL1[6685~1]_S1_REV	gHIS3[5005~5005]_S1_REV	gLEU2[5005~5005]_S1_REV	gLEU2[5005~5005]_S1_REV	gLEU2[5005~5005]_S1_REV
B	gURAN[5005~5005]_REV	gADE1[5005~5005]_REV	lndC1_REV	mERG20_REV	mMD1_REV	mERG12_REV	mERG13_REV	lghMG1[15865~2000]_REV	snap_a_marker2_o2_REV	snap_a_marker2_o2_REV	snap_a_marker1_o2_REV	snap_a_marker1_o2_REV	lMERG10_REV
C	lMERG8_REV	gHMG1[15865~2000]_REV											
D													
E													
F													
G													

DEMETRIX Build Applications Crop Pipeline: open_fsharp_demo_crop

#50 open_fsharp_demo_crop

Crop summary

GSL Documents

#	Title	Owner
48	gsl paper demo	Darren Platt

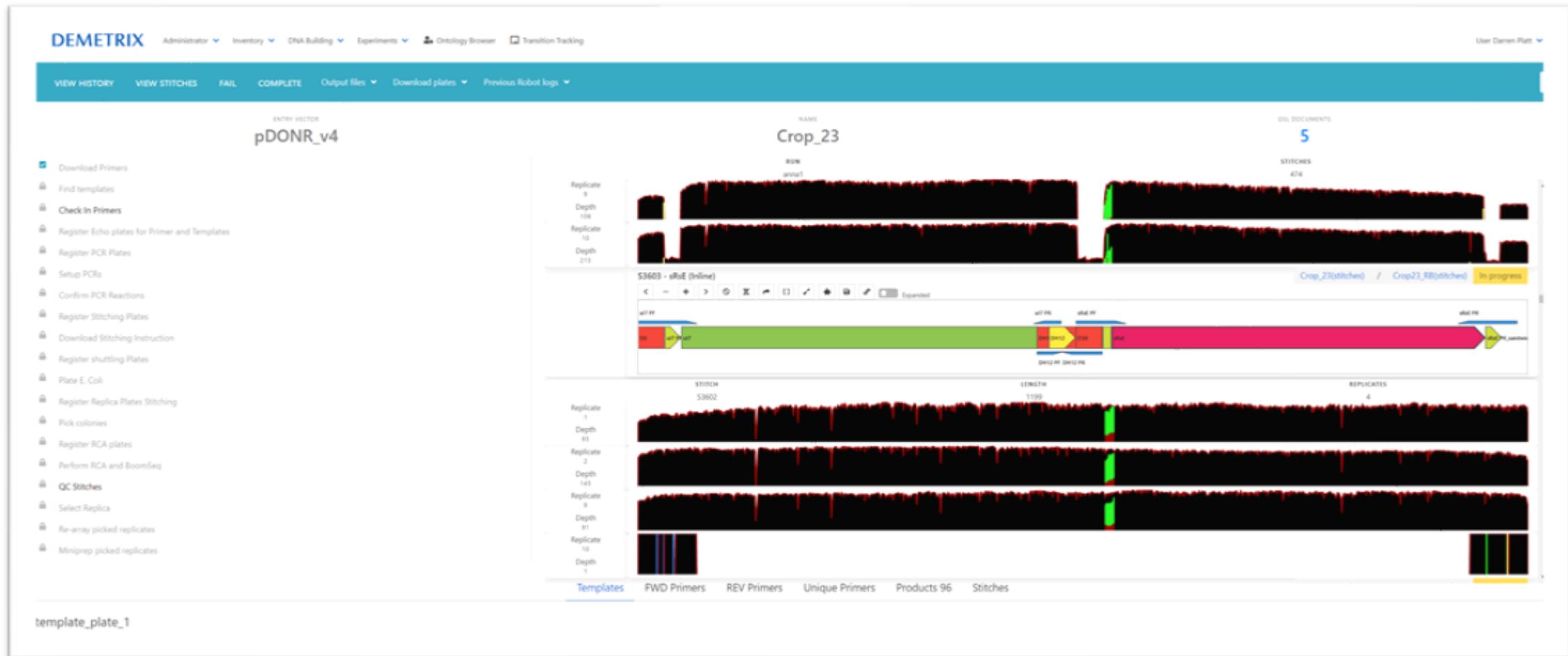
History

Date	User	Action
Today, 4:09pm	Darren Platt	Complete Register Physical Plates for PCR
Today, 4:09pm	Darren Platt	Complete Check In Primers
Today, 4:09pm	Darren Platt	Complete Receive Primers
Today, 4:09pm	Darren Platt	Complete Download Primers

Pipeline progress

- Download Primers
- Receive Primers
- Check In Primers
- Register Physical Plates for PCR
- Download PCR Instructions
- Register Plates

Constructing and then QC-ing DNA

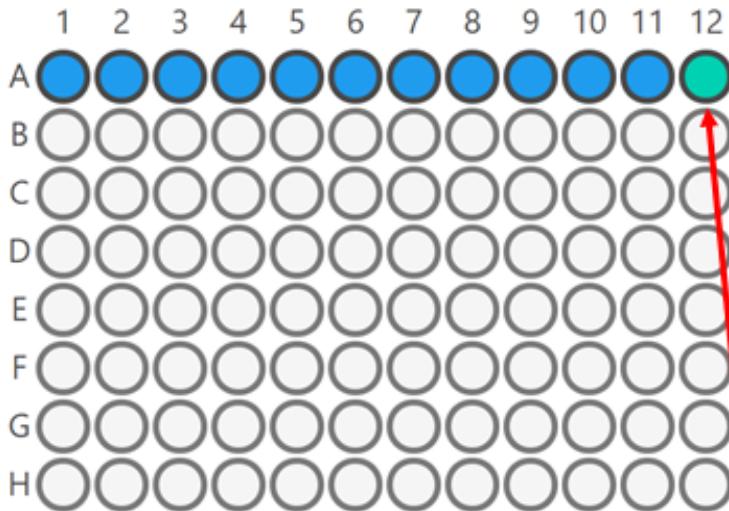


Inventory

stitch_plate_1 (plate)

There is no barcode information available for stitch_plate_1

DETAILS



gTRP1.up____!gHMG1[1586S:~200E]_!/ATGG/_rabitend_gGAL1[-668S:-1S]_mERG20_gTRP1[-500:
(stalk50)

Vessel name	gTRP1.up____!gHMG1[1586S:~200E]_!/ATGG/_rabitend_gGAL1[-668S:-1S]_mERG20_gTR
Vessel type	well
Contains	part
Barcode	
Barcode label	
Created	Today, 4:09pm
Created by	Darren Platt

Single DNA design
from example GSL

Back

Example physical
location

Virtual model of Physical World

DEMETRIX

Administrator Inventory DNA Building Experiments Ontology Browser Transition T

LOOKUP

StrainSearch Single barcode S616 Help

UPLOAD

DLabel uploading Stock C Stock B Both Strains Stock A Micronics log Search Cancel Restock

B578 (S616) B Strains Plate 7C Stock - G11 B696 (S616) B Strains Plate 10C Stock - B12

B862 (S616) B Strains Plate 9C Stock - H10 B865 (S616) B Strains Plate 10C Stock - A1

B1054 (S616) B Strains Plate 11C Stock - H10 B1150 (S616) B Strains Plate 12C Stock - H10



Experiment planning, human-robot coordination

EXPERIMENT LOOKUP	
Flavor	Find experiment
STRAIN ENGINEERING	
Excel Import	
Plans	
Submitted Experiments	
PROCESS DEVELOPMENT & ENGINEERING	
Excel Import	
Experiments	
Single Sample Submission	
Gallery Data Import	
HTS	
Week Schedule	
Day Schedule	
ANALYTICAL CHEMISTRY	
Week Schedule	
Sample Log	
Create Run	
View Runs	

Analytical Chemistry Schedule						
Monday	Tuesday	Wednesday	Thursday	Friday	Saturday	Sunday
	Today	Jul 27	Jul 28	Jul 29	Jul 30	Jul 31
▶ 1 Combined-CBVA		▶ 1 Sugars	▶ 1 Sugars	▶ 33 Combined-CBGVA		
▶ 1 DAD-CBG_Overlay		▶ 1 Hex	▶ 1 Hex	▶ 1 Sugars		
▶ 3 Combined-CBG		▶ 1 Acids-CBDA	▶ 2 Combined-CBGVA	▶ 1 Combined-CBGVA	▶ 1 Hex	
PE246	SAM57					
EXP707		SC61	PE247	SC61	SC61	EXP709
Aug 2	Aug 3	Aug 4	Aug 5	Aug 6	Aug 7	Aug 8

EXP707 Process Assay Plots																	
Planned				Executed													
▶ PC 1 (Jul 22) Barcode: 1000057920 Temp: 30 °C Media: YPD ▶ PR 1 (Jul 24) Barcode: 1000057778 Dilution: 25x Temp: 30 °C Media: Jellyfish - 1.3mM Hex-Defined-Suc/Gal/Dex ▶ EKT (Jul 26) Barcode: -- Dilution: 40x Combined-CBG (Jul 26) Barcode: -- Dilution: 25x OD (Jul 26) Barcode: -- Dilution: 10x ▶ PR 2 (Jul 24) Barcode: 1000059751 Dilution: 25x Temp: 30 °C Media: custom ▶ EKT (Jul 26) Barcode: -- Dilution: 40x Combined-CBG (Jul 26) Barcode: -- Dilution: 25x OD (Jul 26) Barcode: -- Dilution: 10x ▶ PR 3 (Jul 24) Barcode: 1000059750 Dilution: 25x Temp: 30 °C Media: custom ▶ EKT (Jul 26) Barcode: -- Dilution: 40x Combined-CBG (Jul 26) Barcode: -- Dilution: 25x OD (Jul 26) Barcode: -- Dilution: 10x Glycerol (Jul 24) Barcode: 1000059028 Dilution: 2x OD (Jul 24) Barcode: 1000060607 Dilution: 10x																	
▶ Root Barcode: 1000057920 PR (Jul 24) Barcode: 1000057778 Dilution: 25x Media: Jellyfish - 1.3mM Hex-Defined-Suc/Gal/Dex PR (Jul 24) Barcode: 1000059751 Dilution: 25x Media -- PR (Jul 24) Barcode: 1000059750 Dilution: 25x Media -- Glycerol (Jul 24) Barcode: 1000059028 Dilution: 2x OD (Jul 24) Barcode: 1000060607 Dilution: 10x																	
PC																	
Submission File	Plate	Incubation Length (days)	Temp (°C)	Media	Planned Processing Date	Processing Date	Comments	Actions									
Exp 707.xlsx	Plate: PC 1 Type: 1.8mL RWUB Barcode: 1000057920 Edit barcode	2	30	YPD	Jul 22	Jul 24		Plate map									
PR																	
Submission File	Plate	Source	Incubation Length (days)	Temp (°C)	Media	Dilution	Planned Processing Date	Processing Date	Comments	Actions							
Exp 707.xlsx	Plate: PR 1 Type: 1.8mL RWUB Barcode: 1000057778 Edit barcode	Plate: PC 1 Type: 1.8mL RWUB Barcode: 1000057920 Edit barcode	2	30	Jellyfish - 1.3mM Hex Defined Suc/Gal/Dex	25	Jul 24	Jul 24	Jellyfish - 1.3 mM	Plate map							
Exp 707.xlsx	Plate: PR 2 Type: 1.8mL RWUB Barcode: 1000059751 Edit barcode	Plate: PC 1 Type: 1.8mL RWUB Barcode: 1000057920 Edit barcode	2	30	custom Custom	25	Jul 24	Jul 24	Jellyfish - 1.6 mM	Plate map							
Exp 707.xlsx	Plate: PR 3 Type: 1.8mL RWUB Barcode: 1000059750 Edit barcode	Plate: PC 1 Type: 1.8mL RWUB Barcode: 1000057920 Edit barcode	2	30	custom Custom	25	Jul 24	Jul 24	Jellyfish - 2 mM	Plate map							
Plate Assay																	
Submission File	Plate	Source	Root	Assay	Dilution	Planned Processing Date	Processing Date	Comments	Actions								
Exp 707.xlsx	Plate: OD 1 Type: 0.4ml RWFB clear Greiner Barcode: 1000060607 Edit barcode	Plate: PC 1 Type: 1.8mL RWUB Barcode: 1000057920 Edit barcode	Plate: PC 1 Type: 1.8mL RWUB Barcode: 1000057920 Edit barcode	OD	10	Jul 24	Jul 24		Plate map								
Exp 707.xlsx	Plate: Glycerol 2 Type: 0.45ml RWUB Barcode: 1000059028 Edit barcode	Plate: PC 1 Type: 1.8mL RWUB Barcode: 1000057920 Edit barcode	Plate: PC 1 Type: 1.8mL RWUB Barcode: 1000057920 Edit barcode	Glycerol	2	Jul 24	Jul 24		Plate map								
Exp 707.xlsx	Plate: OD 3 Type: 0.45ml RWUB	Plate: PR 1 Type: 1.8mL RWUB	Plate: PC 1 Type: 1.8mL RWUB	OD	10	Jul 26	..		Plate map								

Complex user interactions - experiment designer

Edit EXP689 Plan

Submit Plan

Strain Selections

Plates

General Settings

PC Plate 1

No Barcode

2 Assays 1 PR

Settings

9 wells selected for editing.

	1	2	3	4	5	6	7	8	9	10	11	12
A	H399 T1 C2 360	H399 T1 C4 360	H399 T2 C2 360	H399 T5 C1 360	H399 T5 C3 360	H399 T6 C2 360	360	360	360	D3636 D1815 D2184 360	360	360
B	H399 T1 C2 360	H399 T1 C4 360	H399 T2 C2 360	H399 T5 C1 360	H399 T5 C3 360	H399 T6 C2 360	360	360	360	D3636 D1815 D2313 360	360	360
C	H399 T1 C2 360	H399 T1 C4 360	H399 T2 C2 360	H399 T5 C1 360	H399 T5 C3 360	H399 T6 C2 360	360	360	360	D3636 D1815 D2313 360	360	360
D	H399 T1 C2 360	H399 T1 C4 360	H399 T2 C2 360	H399 T5 C1 360	H399 T5 C3 360	H399 T6 C2 360	360	360	360	D3636 D2098 D2313 360	360	360
E	H399 T1 C3 360	H399 T2 C1 360	H399 T2 C3 360	H399 T5 C2 360	H399 T5 C1 360	H399 T6 C3 360	360	360	360	D3640 D2098 D4 360	360	360
F	H399 T1 C3 360	H399 T2 C1 360	H399 T2 C3 360	H399 T5 C2 360	H399 T5 C1 360	H399 T6 C3 360	360	360	360	D3640 D2098 D4 360	360	360
G	H399 T1 C3 360	H399 T2 C1 360	H399 T2 C3 360	H399 T5 C2 360	H399 T5 C1 360	H399 T6 C3 360	360	360	360	D3640 D2184 360 360	360	360
H	H399 T1 C3 360	H399 T2 C1 360	H399 T2 C3 360	H399 T5 C2 360	H399 T5 C1 360	H399 T6 C3 360	360	360	360	D3640 D2184 360 360	360	360

Strain Type None

Strain Notes For Well

Media Type YPD

Amount 360

Media Notes For Well

Is Media Control

Add PC Plate

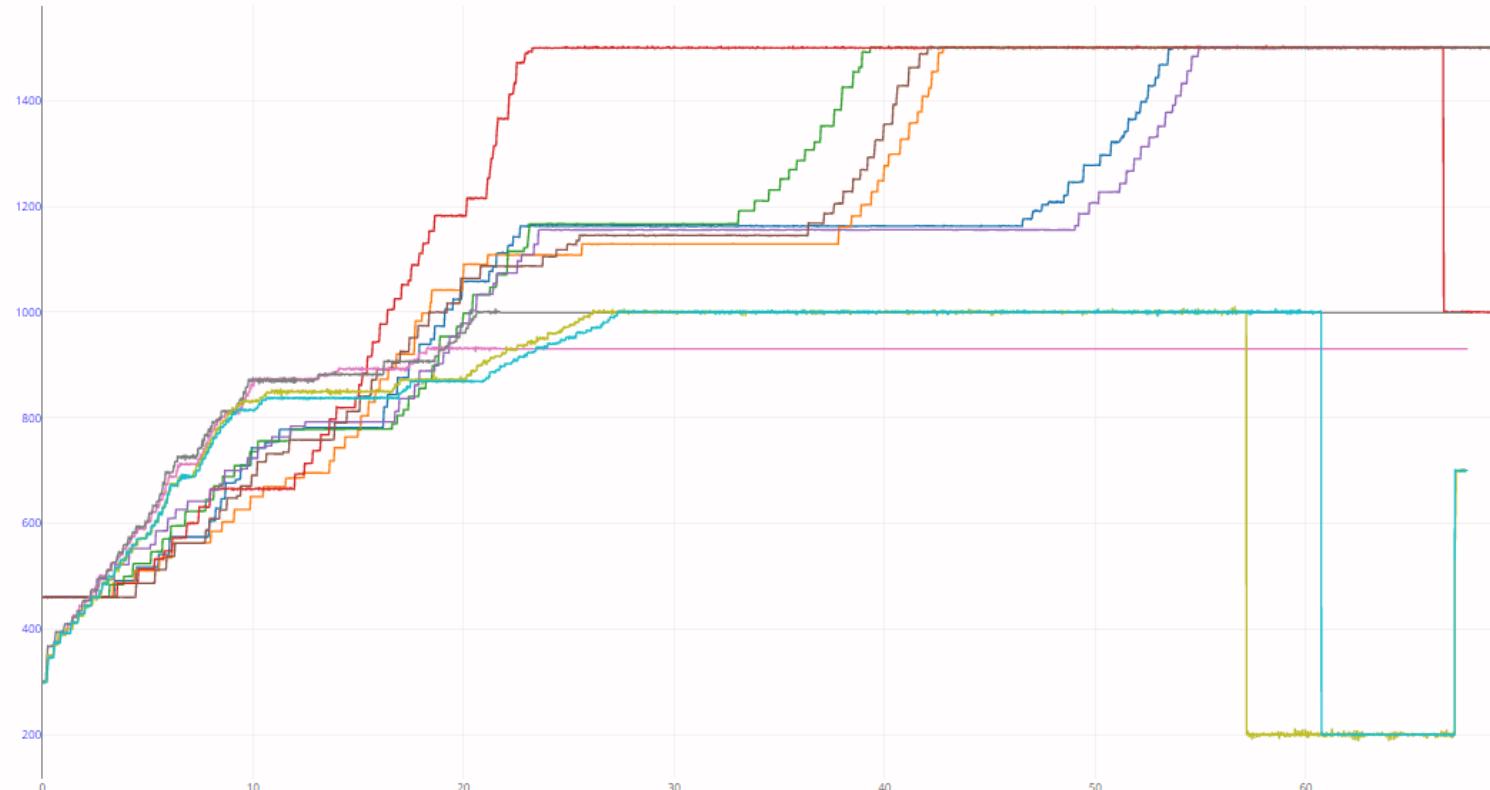
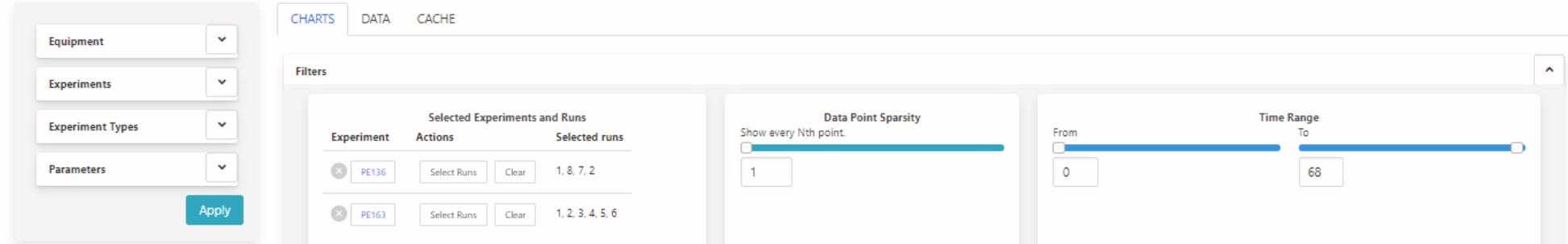
Bulk operations:

Add Assay To All PC Plates

Add PR Plate To All PC Plates

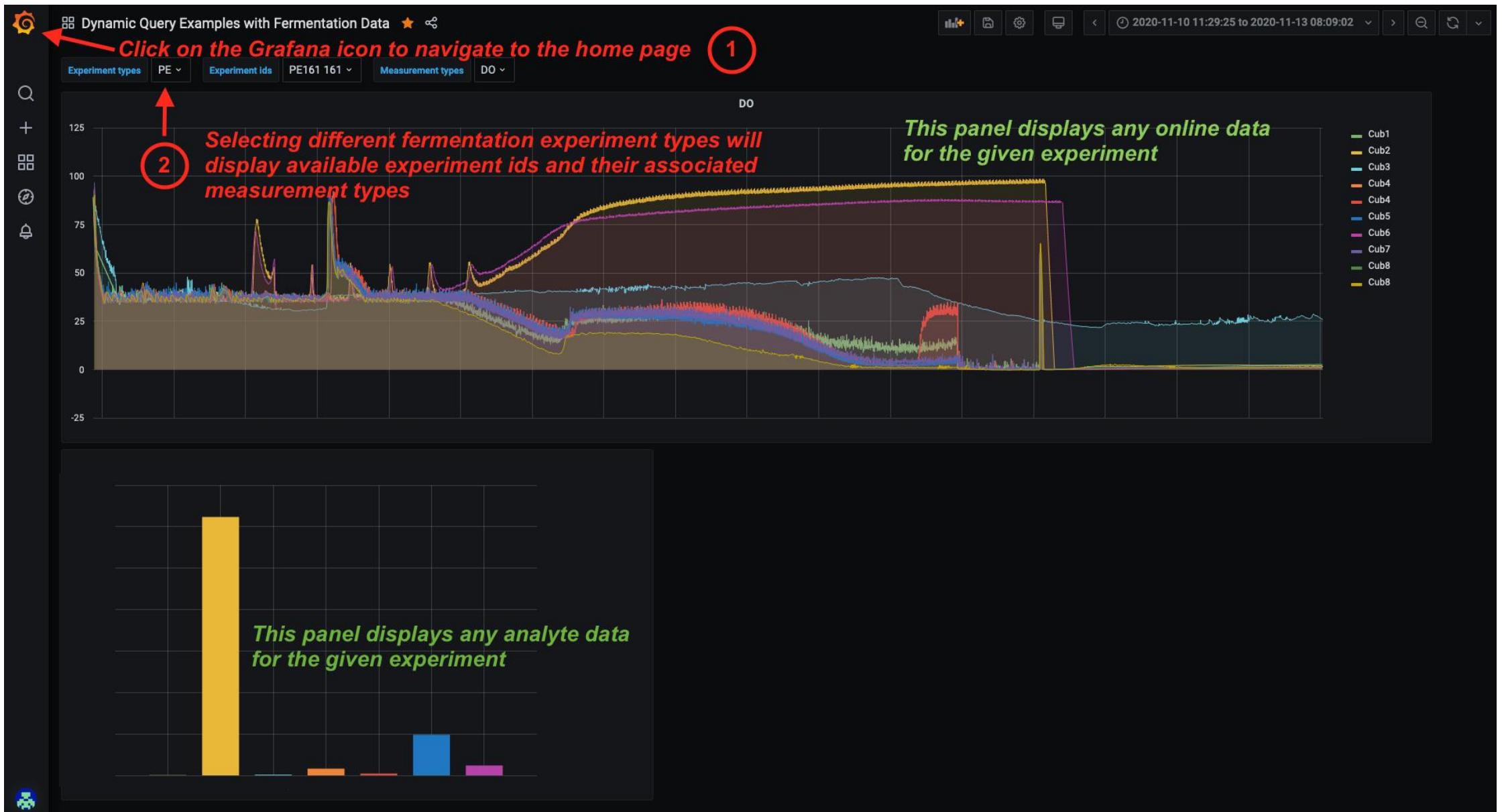
Add Assay To All PR Plates

Experimental Data Visualization



Feliz.Plotly
<https://github.com/Shmew/Feliz.Plotly>

Data Warehouse & Grafana



DASware Linter – makes reactor setup easier

- Syntax & sanity checking
- Hover provider
- Linter with superpowers: uses set points parsed from the script to predict titers using a deep learning model trained on previous fermentation runs

The screenshot shows the DASware Linter interface. On the left is a code editor window titled "dasgip_template.vb". The code is a VBScript snippet that includes an "if" statement with a variable "p". A tooltip is displayed over the variable "p", providing information about its type ("Function group: Phase"), value ("Variable to store process phase"), and access levels ("Readable: True" and "Writeable: True"). The code editor has a dark theme with syntax highlighting. On the right is a sidebar with tabs for "Manual", "Parameters", and "Model". The "Model" tab is active, showing a message "Not up to date with reactor script". At the bottom of the interface, there is a status bar with the text "Errors: 0 - Warnings: 6".

So... why F#?

What makes building software enjoyable?

- Easy domain modelling and data manipulations
- NOT spending most of my time on error hunting
- Focusing on solving business problems, not on boilerplate and ceremonies
- Convenient async and parallel programming
- Extensive, mature tooling and libraries for
 - talking to databases
 - dealing with various kinds of data
 - developing web applications
 - easier coding (e.g. IDE itself)
- Supportive tech community (especially if smaller company)



So... why F#?

Easy domain modelling and data manipulation

- Records (AND types)
- Discriminated Unions (OR-types)
- Single-case Unions
- Pattern matching
- Collection types (Sequence / List / Map / Array ...)
- Handy functions for collection processing
- Pipe operator! |>

```
module StageTopology =
    type Single =
        { VesselId : VesselId
          StageTypeTagId : TagIdentifier }

    type SingleWithChildren =
        { VesselId : VesselId
          StageTypeTagId : TagIdentifier
          Children : VesselId list }
```

1

```
type Msg =
| BarcodeInputMsg of SearchInput.Msg<Vessel>
| VesselRetrieved of Vessel
| RelatedExperimentsRetrieved of Experiment list
| ShowError of exn
```

2

```
[<Struct>]
type TagIdentifier = TagIdentifier of id : int
with
    int
    member this.Value with get() = let (TagIdentifier value) = this in value
module TagIdentifier =
    TagIdentifier -> int
    let get (TagIdentifier id) = id
```

3

```
let vesselTypes =
    if m.SelectedVesselTypes.IsEmpty then
        None
    else
        m.SelectedVesselTypes : Map<TagIdentifier,Tag>
        ▷ Seq.map (fun s -> s.Key) : seq<TagIdentifier>
        ▷ Array.ofSeq : TagIdentifier []
        ▷ fun s -> VesselType.Any(transitionRole, s ▷ Array.map TagIdentifier.get)
        ▷ Some
```

4

```
let executedTransferMap =
    plannedToExecutedAssayMap.Executed : Map<TransferPathKey,TransferPath>
    ▷ Seq.map (fun s -> s.Value) : seq<TransferPath>
    ▷ Seq.groupBy (fun s -> s.SourceBarcode) : seq<string * seq<TransferPath>>
    ▷ Seq.map (fun (key, rows) ->
        let byPath = rows ▷ Seq.map (fun s -> s.BarcodePath, s) ▷ Map.ofSeq
        key, byPath) : seq<string * Map<string [],TransferPath>>
    ▷ Map.ofSeq
```

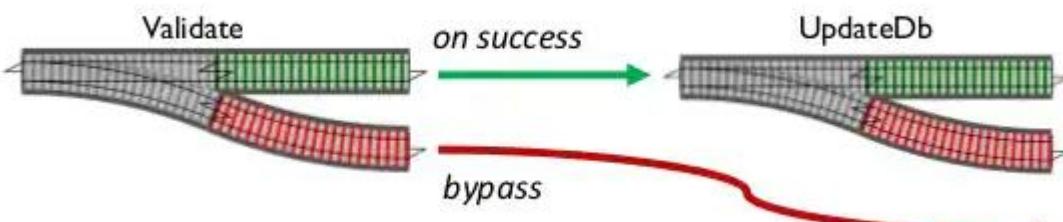
5

So... why F#?

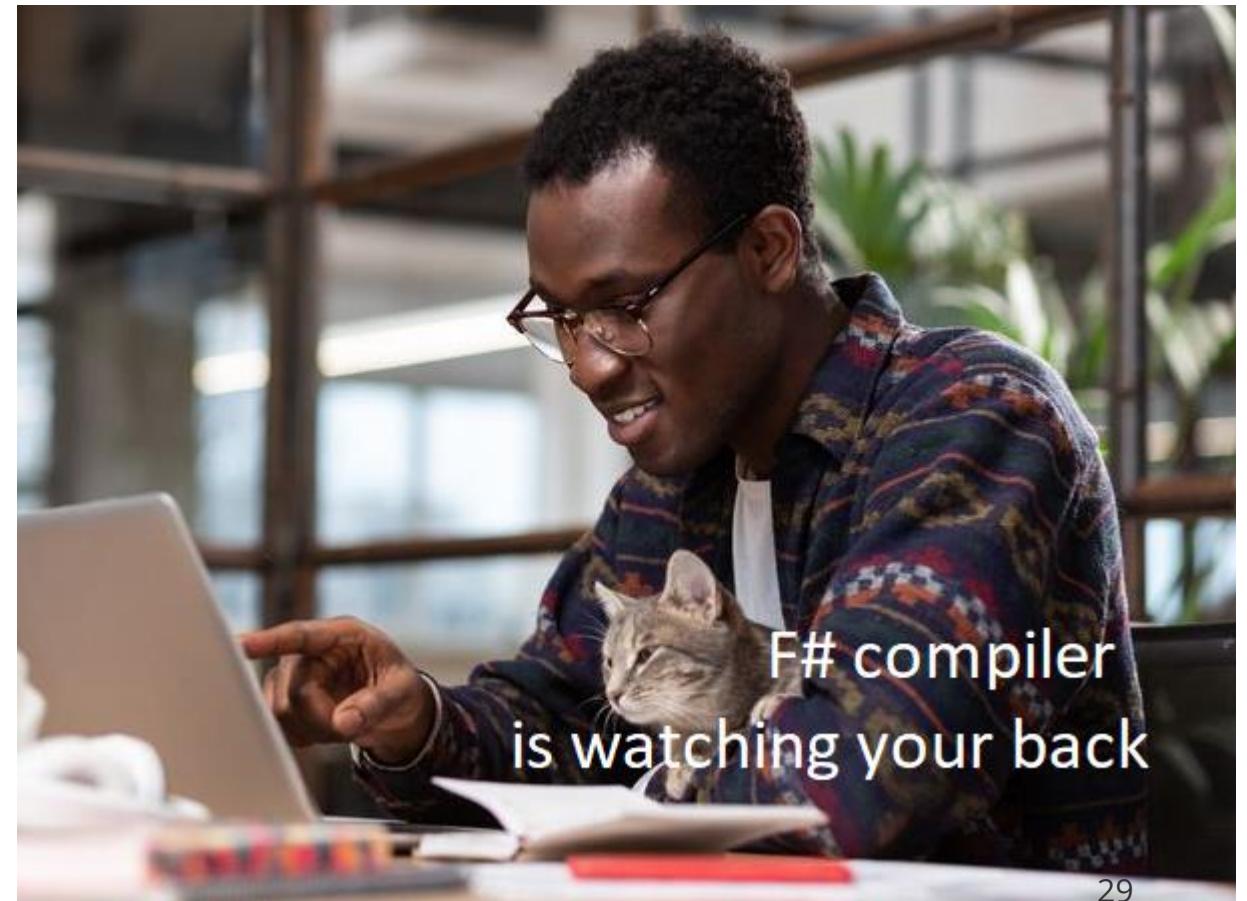


NOT spending most of your time debugging and error hunting

- Smart compiler is your friend!
 - Static typing and type inference
- Immutability by default
- Option types



- Units of measure
- Railway Oriented Programming (ROP)



So... why F#?



Focusing on solving business problems, not on boilerplate and ceremonies

- ROP-style control flow
- Declarative nature of the language
- Terseness of the language

```
type FailureMessage =
| Unknown of string
| Database of string
| Parse of string
| Validation of string
| Conflict of string
| NotFound of string
| ExceptionFailure of exn
string

member this.Message =
    match this with
    | Unknown s
    | Database s
    | Parse s
    | Validation s
    | Conflict s
    | NotFound s → s
    | ExceptionFailure e → e.Message

FailureMessage -> string
static member unwrap (failure : FailureMessage) =
    match failure with
    | FailureMessage.Database msg → msg
    | FailureMessage.Parse msg → msg
    | FailureMessage.Validation msg → msg
    | FailureMessage.Conflict msg → msg
    | FailureMessage.Unknown msg → msg
    | FailureMessage.NotFound msg → msg
    | FailureMessage.ExceptionFailure exn → exn.ToString()

type Success<'TSuccess> =
{ Data : 'TSuccess
  Warnings : string list }

type Either<'a> = Result<Success<'a>, FailureMessage>
```

```
ExperimentPK -> (unit -> 'a) -> Task<Either<Experiment>>
let private updateExperiment experimentId update =
    taskEither {
        do! ExperimentStorage.assertExperimentExists experimentId false : System.Threading.Tasks.Task<unit option>
        ▷ TaskEither.isSome (NotFound $"expId={experimentId.Value}")
        use scope = Db.createTransactionScope()
        do! update () ▷ TaskEither.ofTask
        let! experiment =
            ExperimentStorage.getExperimentById experimentId true : System.Threading.Tasks.Task<Experiment option>
            ▷ TaskEither.isSome (Validation $"expId={experimentId.Value}")
        scope.Complete()
        return experiment
    }
```



<https://github.com/demetrixbio/Plough.ControlFlow>

So... why F#?



Convenient async and parallel programming primitives

```
int -> SaveGSLDocument -> Task<Either<GSLSrcDocument>>
let saveGSLDocument userId cmd =
    taskEither {
        let (document, overwrite) =
            match cmd with
            | Strict doc -> (doc, false)
            | Overwrite doc -> (doc, true)

        let! validation = GSLDocumentValidationService.validate overwrite document userId

        let hashedContent = hashASCIIBySHA256 document.Content
        let doc =
            { document with
                LastUpdated = DateTime.Now
                HashedContent = hashedContent }

        return! GslDocumentCommandStorage.saveGSLDocument doc
    }
```

```
open System
open System.IO

let printTotalFileBytes path =
    async {
        let! bytes = File.ReadAllTextAsync(path) |> Async.AwaitTask
        let fileName = Path.GetFileName(path)
        printfn $"File {fileName} has %d{bytes.Length} bytes"
    }

[<EntryPoint>]
let main argv =
    argv
    |> Seq.map printTotalFileBytes
    |> Async.Parallel
    |> Async.Ignore
    |> Async.RunSynchronously
```

0

- Async.StartChild
 - Async.Parallel
 - Async.StartAsTask
- ...

[Async programming in F# \(Microsoft Docs\)](#)
[Writing Concurrent Programs Using F# Mailbox Processors](#)
<https://github.com/fsprojects/FSharpx.Async>

So... why F#?



Extensive, mature tooling and libraries

Data

- FSharp.Data (CSV, XML)
- Fsharp.Data.Npgsql – real time resolution of SQL / type issues (ours)
- Dapper (+ custom SQL query building)
- Dbup + custom wrapper for migrations

Web

- Giraffe / Asp.Net Core
- Fable
- Elmish
- Fulma
- SAFE stack
- Thoth
- Feliz.Plotly

Infra

- Paket
- FAKE
- Hangfire – routine batch computing
- Nunit - testing
- Argu - dealing with CLI arguments

Misc

- Jupyter Notebooks + IFsharp

IDEs

- VS Code + Ionide
- JetBrains Rider



Supportive community

- F# Twitter helps to solve both tech and employment problems
- F# Slack - very welcoming, helps F# beginners to learn and strive
- Friendly OSS project maintainers
- F# mentorship program!

Elmish 'Model-View-Update' example

```
type Model =
    { Orientation : Orientation
      FileName : string
      Content : string
      ValidationStatus : Validation.State
      ValidationMessage : string option }

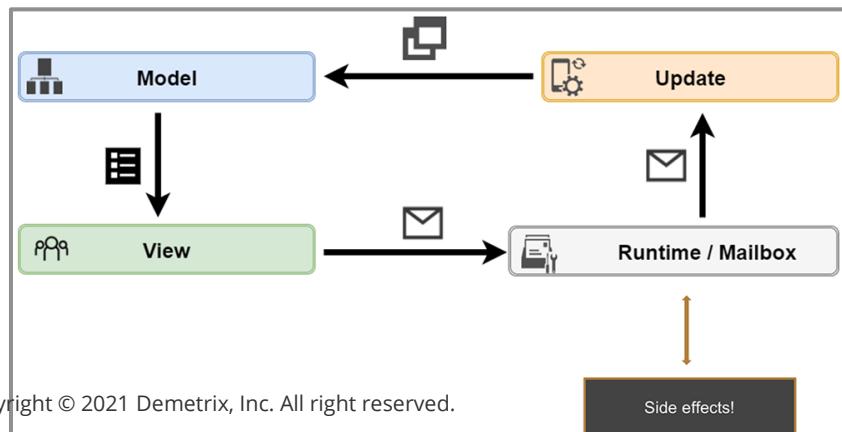
type ExternalMsg =
    | SaveFileContent of string * string
    | NoOp

type Msg =
    | SaveContent of name : string * content : string
    | ErrorReadingFile

Orientation -> Model
let init orientation =
    {
        Orientation = orientation
        FileName = ""
        Content = ""
        ValidationStatus = Validation.NotTriggered
        ValidationMessage = None
    }
```

```
Msg -> Model -> Model * Cmd<a> * ExternalMsg
let update msg model =
    match msg with
    | SaveContent (name, content) ->
        { model with FileName = name
          Content = content
          ValidationStatus = Validation.NotTriggered
          ValidationMessage = None }, Cmd.none, SaveFileContent(name, content)
    | ErrorReadingFile ->
        model, Cmd.none, NoOp
```

```
Model -> (Msg -> unit) -> ReactElement
let view model dispatch =
    Text.div [] [
        Fulma.File.file [ if model.FileName <> "" then yield Fulma.File.HasName
                           if model.Orientation = Vertical then yield Fulma.File.IsBoxed
                           yield Fulma.File.Size IsSmall
                           if model.ValidationStatus = Validation.Failed then yield Fulma.File.Color IsDanger ] [
            Fulma.File.label [] [
                yield Fulma.File.input [ GenericOption.Modifiers [ Modifier.BackgroundColor IsBlack; Modifier.TextColor IsDark ]
                                         GenericOption.Props [ OnInput (fun evt ->
                                            let file = (evt.target :> HTMLInputElement).files.[0]
                                            let name = file.name
                                            let reader = FileReader.Create()
                                            reader.onload <- fun _ ->
                                                let content = reader.result :> string
                                                (name, content) > SaveContent > dispatch
                                            reader.onerror <- (fun _ -> dispatch ErrorReadingFile)
                                            reader.readAsText file) ] ]
            yield Fulma.File.cta [
                [ Fulma.File.icon [ ] [ Icon.icon [ ] [ Fa.i [ Fa.Solid.Upload ] [ ] ] ]
                  Fulma.File.label [ ] [ str "Choose a file..." ] ]
                if model.FileName <> "" then yield Fulma.File.name [ ] [ str model.FileName ] ]
            ]
        ]
    ]
```



Web server & API example

```
type Api(client : ApiClient) =  
    OntologyNamespaceIdentifier -> TaskEither<NamespaceAndTags>  
    member x.GetNamespaceById (OntologyNamespaceIdentifier namespaceId) : TaskEither<NamespaceAndTags> =  
        client.Get <| sprintf "/api/onto/namespace/%i" namespaceId  
  
    CreateNamespace -> TaskEither<NamespaceAndTags>  
    member x.CreateNamespace (cmd : CreateNamespace) : TaskEither<NamespaceAndTags> =  
        client.Post("/api/onto/namespace/new", cmd)
```

Thin API Proxy layer



<https://github.com/demetrixbio/Plough.WebApi>

```
unit -> HttpFunc -> HttpContext -> HttpFuncResult  
let api () = subRoute "/onto" <| choose [  
    GET => requiresScope AccessScope.OntologyRead => choose [  
        route "/namespace/all" => makeJSONHandlerAsync<NamespacesAndMasterTags> getNamespacesAndMasterTags  
        routef "/namespace/%i" (makeJSONHandlerWithArgAsync<int, NamespaceAndTags> getNamespaceAndTags)  
        routef "/namespace/name/%s" (makeJSONHandlerWithArgAsync<string, NamespaceAndTags> getNamespaceAndTagsByName)  
        route "/domain" => makeJSONHandlerAsync<Domain list> getDomains  
        route "/domain/all" => makeJSONHandlerAsync<OntologyDomain list> getOntologyDomains  
        routef "/domain/name/%s" (makeJSONHandlerWithArgAsync<string, OntologyDomain> getOntologyDomainByName)  
        routef "/domain/%s/namespaces" (makeJSONHandlerWithArgAsync<string, DomainForTagAssignment> getDomainNamespacesByName)  
        routef "/tagassignment/domainnameandobj/%s/%i" (makeJSONHandlerWithArgAsync<string * int, TagAssignment> getAssignedTagsForDomainNameAndObj)  
        routef "/tagassignment/domainandobj/%i/%i" (makeJSONHandlerWithArgAsync<int * int, TagAssignment> getAssignedTagsForDomainAndObj)  
        routef "/tagassignment/domainname/%s" (makeJSONHandlerWithArgAsync<string, TagAssignment list> getAssignedTagsForDomainName)  
    ]  
    POST => requiresScope AccessScope.OntologyWrite => choose [  
        route "/namespace/new" => makeJSONHandlerWithObjAsync<CreateNamespace, NamespaceAndTags> createNamespace  
        route "/namespace/edit" => makeJSONHandlerWithObjAsync<EditNamespace, Namespace> editNamespace  
        route "/namespace/delete" => makeJSONHandlerWithObjAsync<DeleteNamespace, OntologyNamespaceIdentifier> deleteNamespace  
        route "/namespace/retire" => makeJSONHandlerWithObjAsync<RetireNamespace, Namespace> retireNamespace  
        route "/tag/new" => makeJSONHandlerWithObjAsync<CreateTag, Tag> createTag  
        route "/tag/edit" => makeJSONHandlerWithObjAsync<EditTag, Tag> editTag  
    ]
```

Dispatching endpoints + authorization + JSON wrapping

Interacting with our Database

Fsharp.Data.Npgsql

```
let getPointStructuresByNames (psNames : string []) =
    task { // like C# await
        use! connection = Db.openConnectionAsync()
        use getPointStructureIds = Db.CreateCommand<"SELECT
            p.id, p.name, p.component_types, p.component_units, p.id_user, p.created, p.updated,
            u.alias_name AS u_alias_name
        FROM measure.point_structure p
        INNER JOIN admin.user u ON p.id_user = u.id
        WHERE name = ANY(@names)", SingleRow = false>(connection)
        let! result = getPointStructureIds.TaskAsyncExecute(names = psNames)

        return result : List<FSharp.Data.Npgsql.NpgsqlConnection<...>.Commands.CreateCommand<...>,Record>
            ▷ Seq.map (fun s →
                let componentTypes = s.component_types
                let componentUnits = s.component_units
                let compDescriptions =
                    Array.zip componentTypes componentUnits:(int*int) []
                    ▷ Array.map (fun c →
                        let (cType, cUnits) = c
                        { Type = cType ▷ TagIdentifier
                          Units = cUnits ▷ TagIdentifier })
                { Id = s.id
                  Name = s.name
                  Components = compDescriptions
                  Owner = { Id = s.id_user; GivenName = s.u_alias_name }
                  Created = s.created
                  Updated = s.updated }) : seq<PointStructure>
            ▷ Seq.toList
    }
```

Dapper + QueryBuilder

```
let getChemicalPageByQuery (query : ChemicalPredicate Filter) pageIndex pageSize =
    task {
        let state : QueryInterpreter.State =
            { Template = chemicalPageSqlTemplate
              InnerJoins = Dictionary()
              LeftJoins = Dictionary()
              Parameters = Dictionary() }

        let mapSimple state = function
            | ShortName operation → mapInvariantString state operation "c.short_name"
            | InChI operation → mapInvariantString state operation "c.inchi_key"
            | IUPAC operation → mapInvariantString state operation "c.iupac_name"
            | Formula operation → mapInvariantString state operation "c.formula"

        let pageParam = [ "offset", pageIndex * pageSize ▷ box
                         "limit", box pageSize ] ▷ dict

        let select =
            QueryInterpreter.mapToSql query state mapSimple:SqlBuilder
            ▷ SqlBuilder.parametrized pageParam:SqlBuilder
            ▷ SqlBuilder.build

        let chemicalCountSql = sprintf "SELECT COUNT(1) FROM (%s) AS t;" select.Sql
        let chemicalQuerySql = sprintf "%s ORDER BY c.id DESC OFFSET @offset LIMIT @limit;" select.Sql

        use! connection = Db.openConnectionAsync()
        let! count = connection.QuerySingleAsync<int64>(chemicalCountSql, select.Parameters)
        let! chemicals = connection.QueryAsync<ChemicalProjection>(chemicalQuerySql, select.Parameters)

        return { Items = chemicals:IEnumerable<ChemicalProjection>
                  ▷ Seq.map (fun s →
                      { Chemical.Id = s.Id
                        InChI = s.InChI ▷ Option.ofObj
                        IUPAC = s.IUPAC ▷ Option.ofObj
                        Formula = s.Formula ▷ Option.ofObj
                        Description = s.Description ▷ Option.ofObj
                        State = s.State ▷ Option.ofObj ▷ Option.map Storage.stringToMatterState ▷ Option.get
                        ShortName = s.ShortName
                        SynonymGroup = s.SynonymGroup } ) : seq<Chemical>
                  ▷ List.ofSeq
                  PageSize = pageSize
                 PageIndex = pageIndex
                  TotalItemCount = Some count }
```

<https://github.com/demetrixbio/FSharp.Data.Npgsql>
<https://github.com/demetrixbio/Plough.DynamicQuery>

Room for improvement?

- Native task {} support with state machines
 - ([fsharp/fslang-suggestions/581](https://github.com/dotnet/fsharp/pull/6811)) (<https://github.com/dotnet/fsharp/pull/6811>)
 - We rely a lot on computational expressions
 - Async / Task heterogeneity
 - Excited about work that's going on to improve this whole area
- Tooling performance something we watch (and push) – shout out to all the IDE maintainers
- Native JS + Elmish model friction around state
- Scripting seems harder than it should be still (e.g. dependency loading) but that's improving
- Jupyter F# experience still not as good as Python but improving
- For novices: challenging to set things up and get started, documentation inconsistencies

Conclusions

- Biotech is a tough proving ground for a programming environment
- F# has been a great fit for developing software to manage our lab, both backend and frontend
- F# helps to handle complex, changing requirements gracefully, reduces code refactoring-PTSD
- Result is compact, type-safe, reliable code, quickly delivered that is maintainable in face of rapidly evolving requirements
- Learning curve is a bit steep for novices but manageable -- interns and scientists keep making valuable contributions to Demetrix's software



DEMETRIX

Experience the ingredients of life

Thanks for joining!
Ask your questions live on Twitter #dotNETConf

Darren's minimal shopping list for a programming language

- Strongly typed
 - reduces errors
 - reduces errors
- Semantic whitespace
 - reduces errors
 - reduces errors
- Functional first
 - reduces errors
 - reduces errors
- Option types
 - reduces errors
 - reduces errors
- Units of measure
 - reduces errors
 - reduces errors
- Type inference
 - reduces RSI, makes #1 palatable
 - fast
- Static typing
 - lazy fast
- Good parallel primitives
 - reduces development time
 - reduces development time
- Rich built in data types
 - reduces dependency
- Type providers
 - don't want to roll my own db driver
 - increases library support
- Open source
 - make CPU work for developer
- Decent library support
 - life is too short to hang out with jerks
- Language interop (*)
- Mature tooling
- Supportive community

Past Demetrix presentations in the F# world



Darren Platt
President and Head of Computing

F# for Biotech — Stranger Things edition

Darren Platt

Biotech is a broad and demanding field for software development. It combines a messy domain, rapidly changing requirements and brutal timelines for software development that stress the software development life cycle. In the middle of this controlled software explosion, we have developed a platform using F# starting from a compiler that targets life itself, through to data management systems overseeing millions of liquid transfers a year. The end product of the lab engineering is a living organism (yeast based) that can manufacture the active ingredients in the Cannabis plant via fermentation. This talk will give a user-focused view into the types of problems we have to solve, and why F# and Fable have been good companions on this journey. Also why you don't want to be the only BASIC programming in a lab full of roaming monsters.



2020

A YEAR IN PARADISE. FULL STACK F# FOR SYNTHETIC BIOLOGY

THU 27TH - 12:20 PM (TALK)

We are now about one year into implementing a largely F# environment for designing, building and analyzing engineered microorganisms. This is cloud deployed full stack F# (Fable, React, Elmish, Giraffe, Postgres) with algorithms for designing DNA, and web based systems for controlling robots and managing a lab. I will talk about what has worked well and where there are rough spots. I will touch on some areas of biology but it will be mostly about implementing a system for managing a complex manufacturing environment that is widely applicable to many industries.



2018

GSL: A COMPILER TOOL-CHAIN WHOSE TARGET ARCHITECTURE IS LIFE ITSELF

At Amyris, we are taking advantage of rapid advances in automation, biology and computing to engineer microorganisms into platforms that efficiently manufacture molecules for a wide range of industrial, consumer and pharmaceutical applications. We are currently working on hundreds of molecules and as the rate of engineering has increased, biologists are facing many of the same issues faced by early software pioneers. We have been introducing many software practices to the biology community at Amyris to increase efficiency and accuracy of genetic designs.



2019

LEARNING F# ON THE JOB AS A PROGRAMMING NOVICE

THURSDAY, SEPTEMBER 26 @ 16:00

Demetrix is a biotech start-up built on Fsharp infrastructure. We have some great Fsharp developers along with a lot of scientists and engineers that are relatively new to programming. I'm in the latter camp. I'll give some impressions and lessons learned from those of us learning F# as our first programming language.



Kate Curran
Associate Director, Automated Research Operations

<https://github.com/demetrixbio/presentations>



2017