Managing and Analyzing Big-Data in Genomics

Sebastien Mondet[1], **Ashish Agarwal**[12],
Paul Scheid[1], Aviv Madar[1], Richard Bonneau[12],
Jane Carlton[1], Kristin C. Gunsalus[1]

1 Center for Genomics and Systems Biology, Dept of Biology, New York University

2 Courant Institute of Mathematical Sciences, New York University

IBM Programming Languages Day 2012

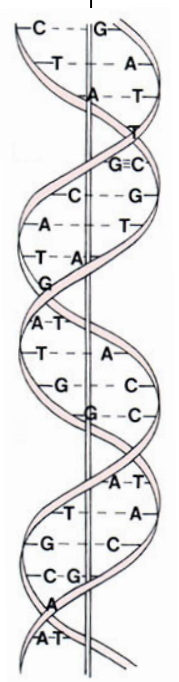
Outline

1. Biology, Genetics, Big Data™, HPC
2. A DSL Approach To The Software Stack
3. OCaml, Ocsigen, … Experience Report

Introduction
Genomics & Sequencing

Let's over-simplify:

```
type sample  (** Some extract of formerly-living material *)
type library  (** Carefully prepared bunch of
                  short DNA fragments *)

val lab_tech:
  sample -> protocol -> library real_world_monad

type base = A | C | G | T

type read = base list
  (** one read  <  one previous DNA fragment *)

val sequencing:
  library -> reagents -> (read list) real_world_monad
```



Introduction
Genomics & Sequencing

How it looks like once on the computational side:

```
@SEQ 1
AATAGTAAATCCATTTGTTCAACTCACAGTTTGATTTGGGGTTCAAAGCAGTATCGATCA
+
!''*(((***+))%%%++)(%%%%).1***-+*''))**55CCF>>>>>>CCCCCCC65

@SEQ 2
GATTTGGGGTTCAAAGCAGTATCGATCAAATAGTAAATCCATTTGTTCAACTCACAGTTT
+
(>CCC(**!>>CCC''*((*%%).1**+))%%''))**5+)(5CC%+%%*-+*F>>>C65
```

c.f. wikipedia:FASTQ_format.

Introduction
Bioinformatics

Then, a branch of computer science takes over:

Alignment [BWA09], [Bowtie09]

```
List.map reads ~f:(fun r -> String.clever_find genome r)
```

Assembly [Assembly10]

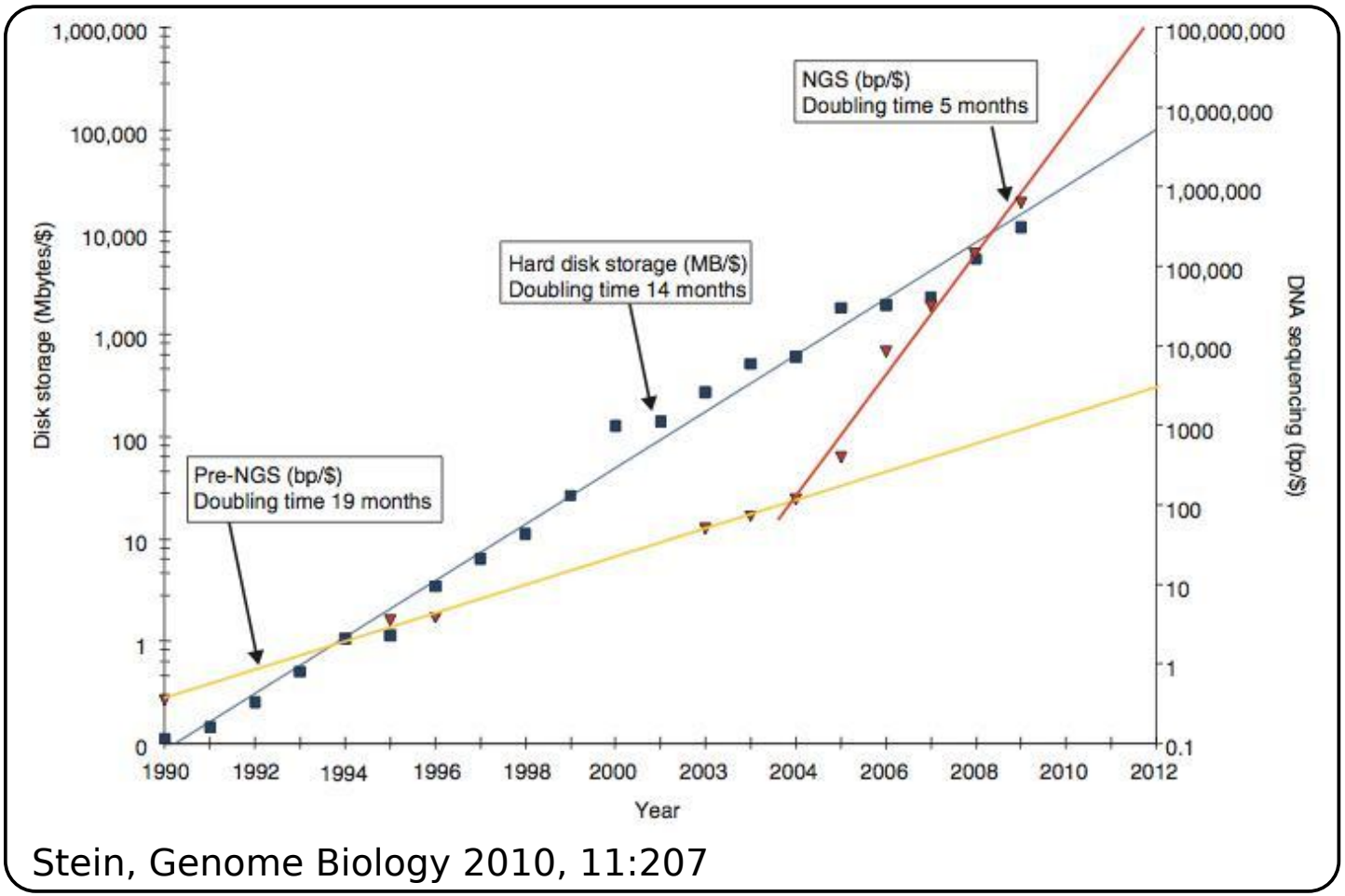
```
val assemble: read list -> genome
```

Annotation [Annot09]

Et cetera.

Introduction

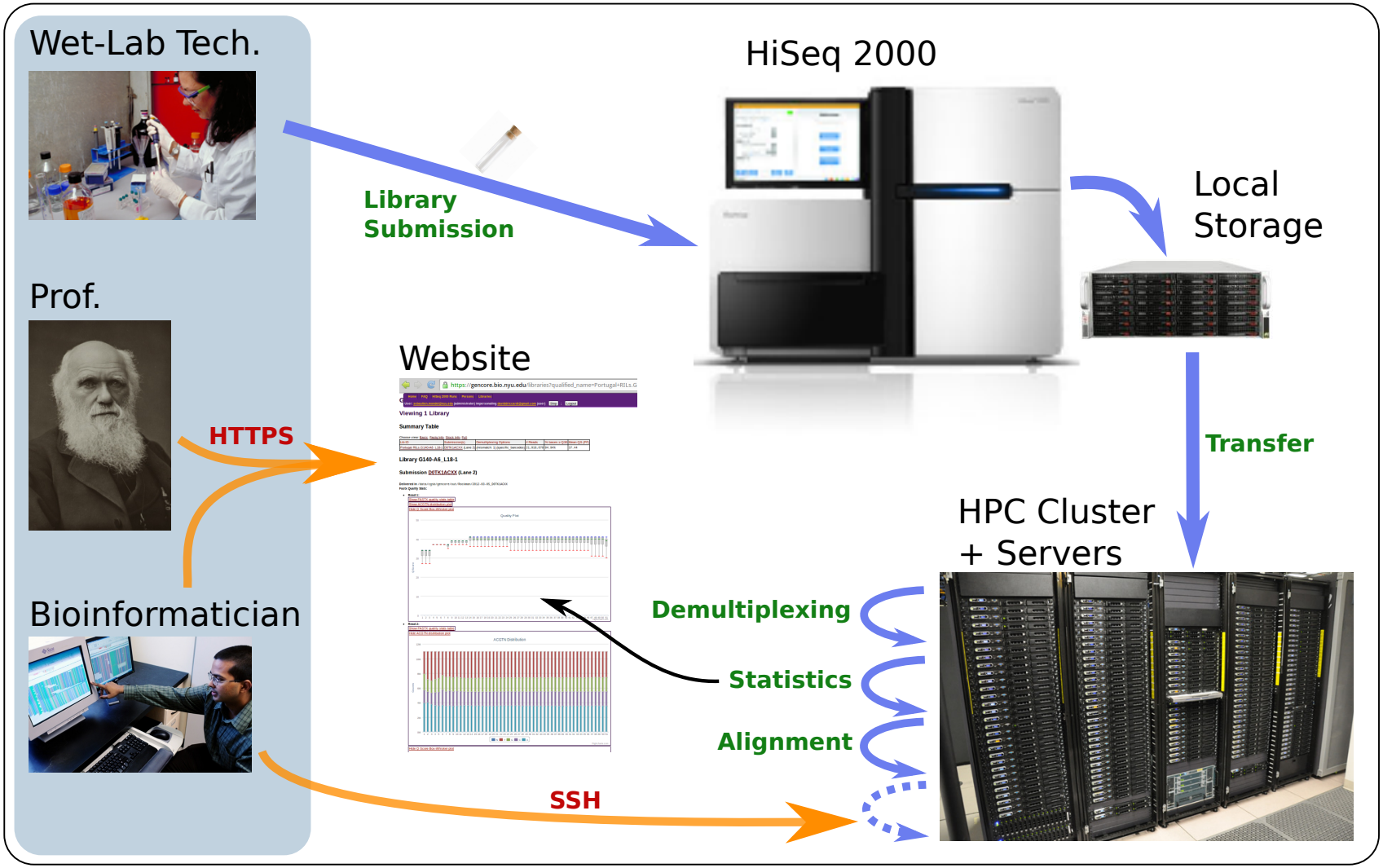
Numbers — Moore's Law of Biology



Stein, Genome Biology 2010, 11:207

Introduction

Our Process



The "Layout" DSL
Why?

Need a persistence layer:

- Database → meta-data.
- File Structure → big data.

Need a more high-level representation:

- Virtual file-system: *just* a cache.
- Ensure coherency: DB tables Vs OCaml code Vs File-system.
- SQL is not portable, weakly typed, verbose.
- Quick and safe migrations.

The "Layout" DSL
The DSL Approach

Define a Domain Specific Language:

- "Our" concepts: functions, records, …
- Better typing: enumerations, non-null by default, file-system elements …
- Manageable size.

The "Layout" DSL
The DSL Approach

Then, from one single descriptive file:

- Generate the "right" SQL queries & File-system paths.
- Generate well typed OCaml code for accessing the data.
- Generate pretty graphs.
- Handle migrations peacefully.
- Provide an *introspection-like* API.
- Generate safety and consistency checking functions.
- Well-formed backups.

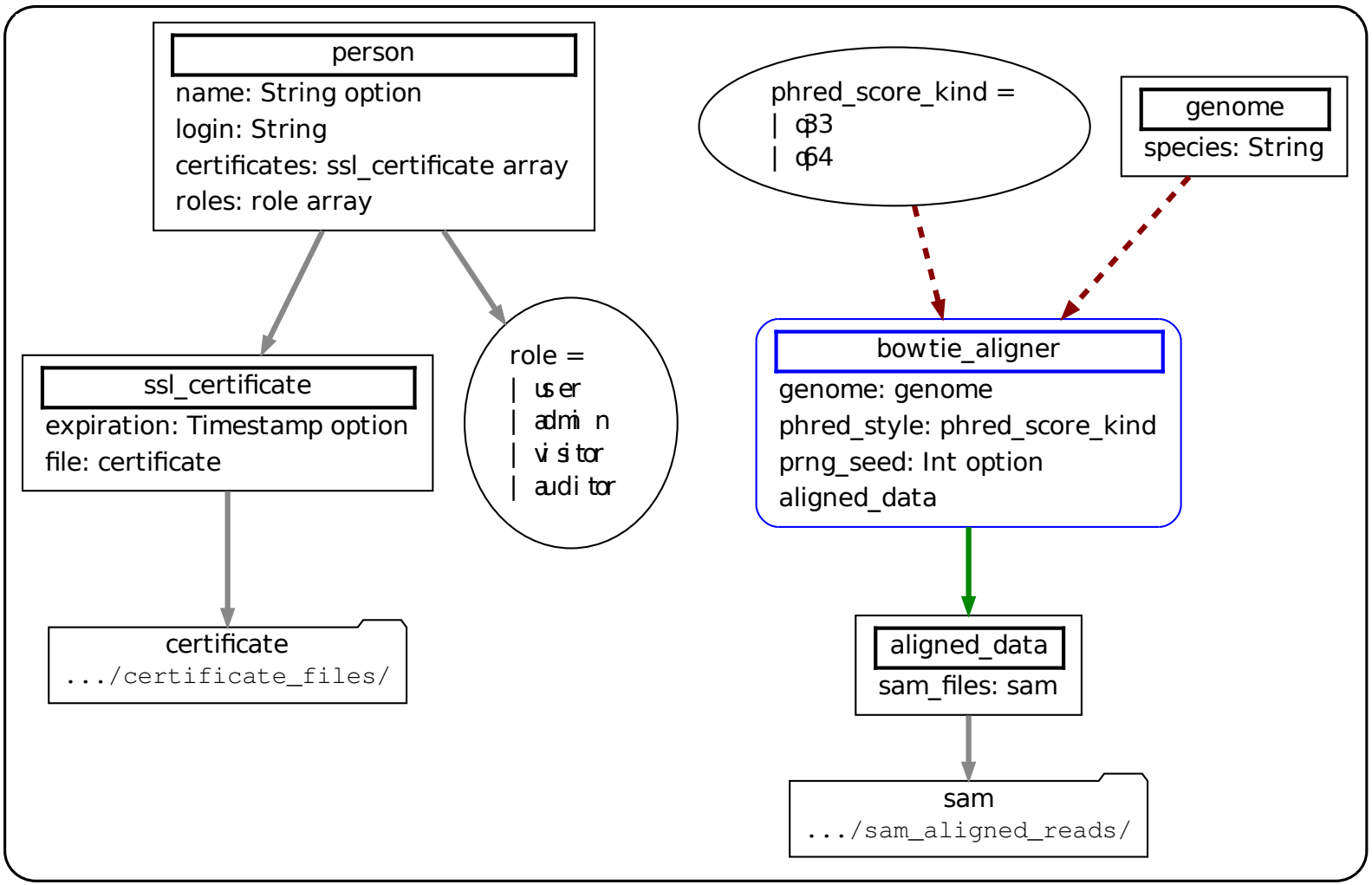
The "Layout" DSL

An Example — The Source

```
(volume certificate certificate_files)
(record ssl_certificate
        (expiration timestamp option)
        (file certificate))
(enumeration role user admin visitor auditor)
(record person
        (name string option)
        (login string)
        (certificates ssl_certificate array)
        (roles role array))
(* ... *)
(function aligned_data bowtie_aligner
          (genome genome)
          (phred_style phred_score_kind)
          (prng_seed int option))
```

The "Layout" DSL

An Example — The Pretty Graph



The "Layout" DSL

An Example — Generated Code

```
module Enumeration_role = struct
(** The type of {i role} items. *)
type t = [‘user | ‘admin | ‘visitor | ‘auditor] with sexp
let to_string : t -> string = function
| ‘user -> ”user”
| ‘admin -> ”admin”
| ‘visitor -> ”visitor”
| ‘auditor -> ”auditor”
let of_string_exn: string -> t = function
(* ... *)

let of_string s =
  try Ok (of_string_exn s) with e -> Error s
```

The "Layout" DSL

An Example — Generated Code

```
module Record_person = struct
type pointer = { id: int}  with sexp
type value = {
  name : string option;
  login : string;
  certificates : Record_ssl_certificate.pointer array;
  roles : Enumeration_role.t array} with sexp
type t = {
  g_id : int;
  g_created : Timestamp.t;
  g_last_modified : Timestamp.t;
  g_value: value} with sexp
```

The "Layout" DSL

An Example — Generated Code

```
module Function_bowtie_aligner = struct
type pointer = { id: int}  with sexp
type evaluation = {
  genome : Record_genome.pointer;
  phred_style : Enumeration_phred_score_kind.t;
  prng_seed : int option} with sexp
type t = {
  g_id : int;
  g_result : Record_aligned_data.pointer option;
  g_inserted : Timestamp.t;
  g_started : Timestamp.t option;
  g_completed : Timestamp.t option;
  g_status : Enumeration_process_status.t;
  g_evaluation: evaluation} with sexp
```

The "Layout" DSL

An Example — Generated Code

```
val add_value: ?expiration: Timestamp.t -> file:File_system.
    pointer ->
      (Record_ssl_certificate.pointer,
        [> `Layout of Layout.error_location
            * Layout.error_cause ]) Flow.t

let add_value ?expiration ~file ~dbh =
  let v = { expiration; file; } in
  let work_m =
    let query =
      Sql_query.add_value_sexp ~record_name:"ssl_certificate"
        (Record_ssl_certificate.sexp_of_value v) in
    Backend.query ~dbh query
    >>| Sql_query.single_id_of_result
    >>= (function Some id -> return {id} |
        None -> error (`wrong_add_value))
  in
  bind_on_error work_m (fun e ->
    error (`Layout ((`Record "ssl_certificate": error_location),
                    (e : error_cause))))
```

The "Layout" DSL

An Example — Generated Code

```
module Bowtie_aligner = struct
  let add_evaluation ~genome ~phred_style ?prng_seed ~dbh =
    (* ... *)
  let get ~dbh pointer =
    (* ... *)
  let get_all ~dbh =
    (* ... *)
  let set_started ~dbh p =
    (* ... *)
  let set_failed ~dbh p =
    (* ... *)
  let set_succeeded ~dbh ~result p =
    (* ... *)
```

The "Layout" DSL

An Example — Migrations & Backups

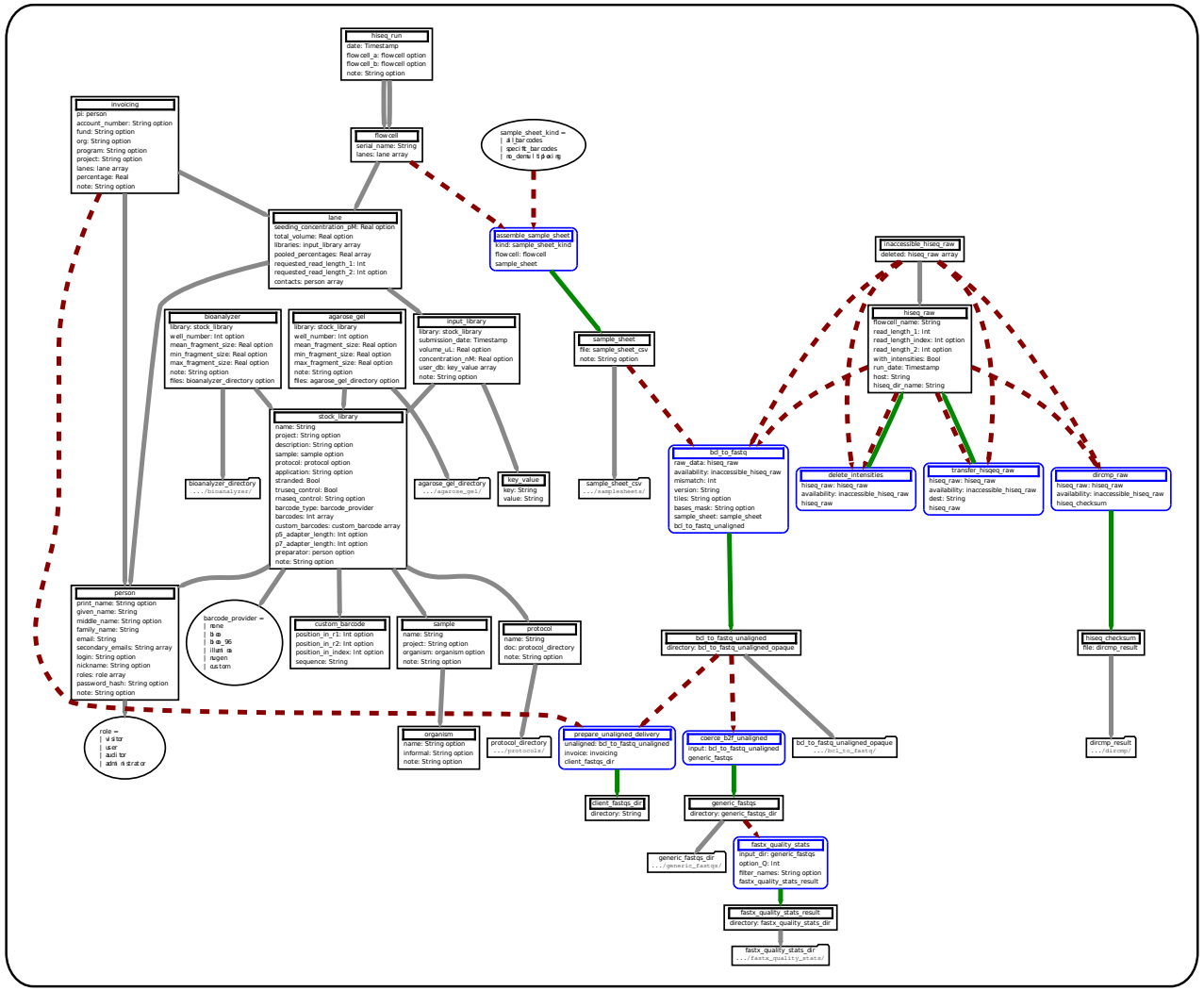
Write

```
val migrator: S-Expression -> S-Expression
```

```
$ hitscore  dump-to-file backup_v42

$ ./migrator backup_v42 backup_v43

$ hitscore wipe-out-database
$ hitscore init-database
$ hitscore load-file backup_v43

$ hitscore verify-layout
```

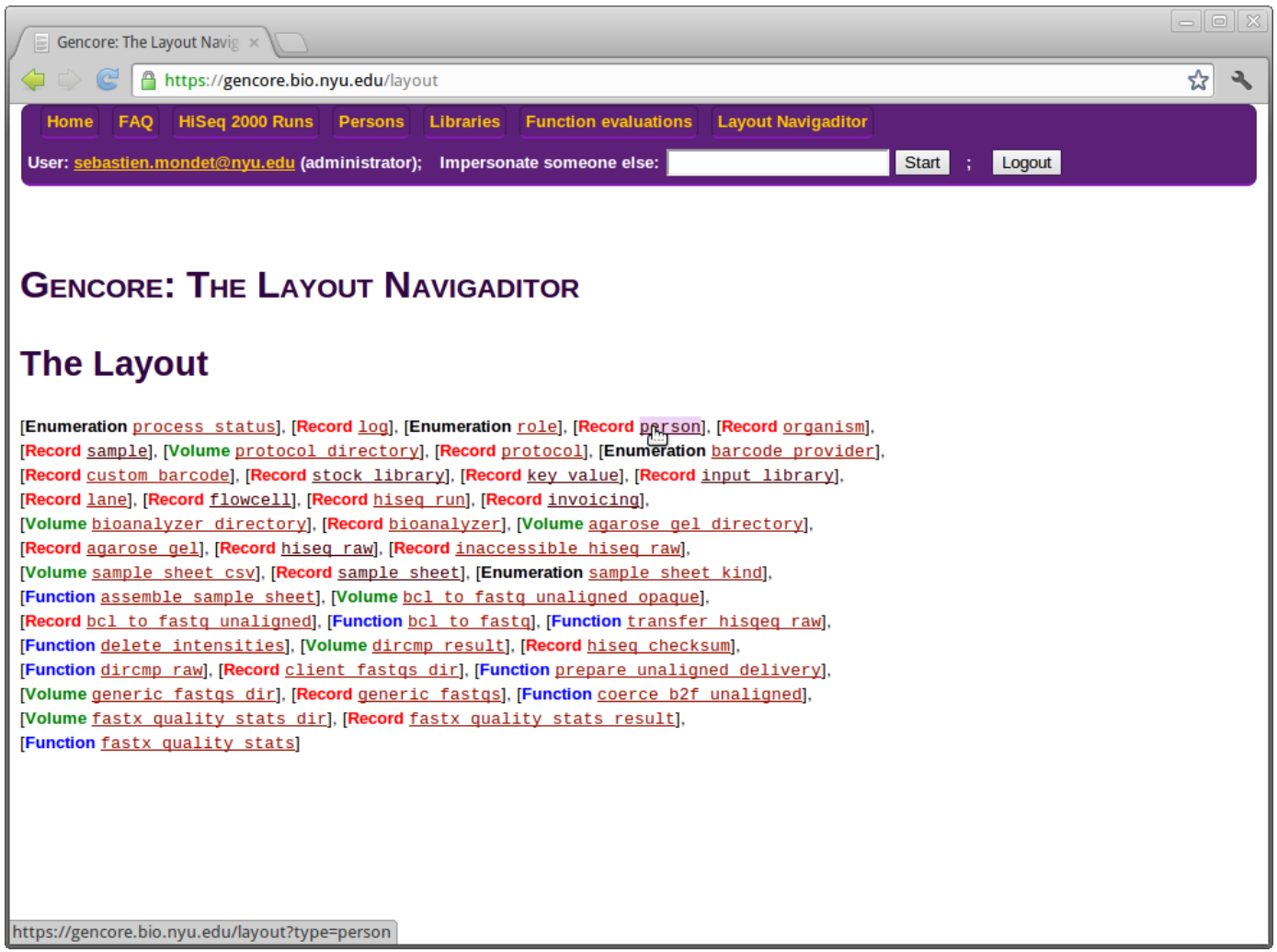
The “Layout” DSL

Our Current Layout



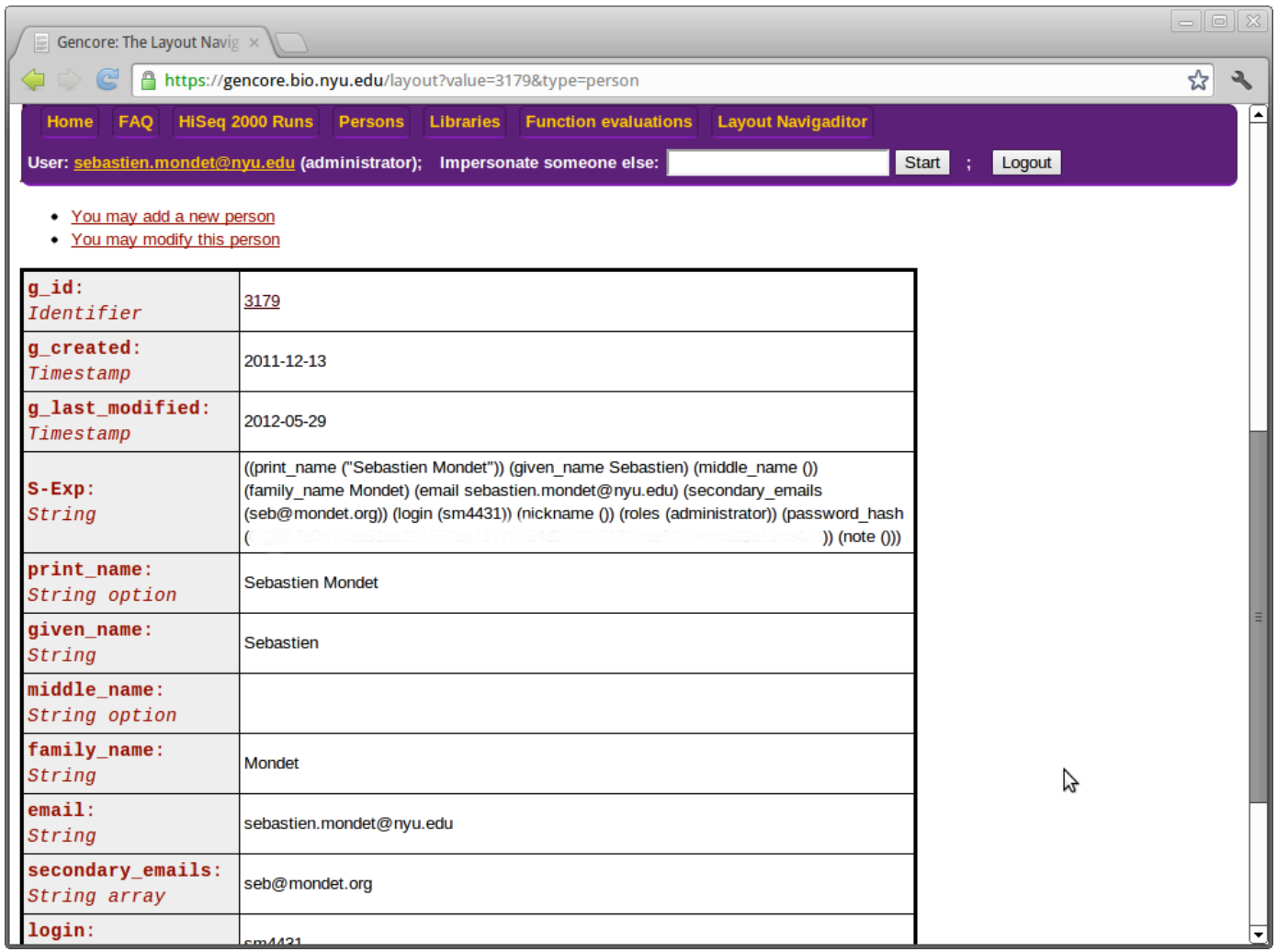
The "Layout" DSL

Introspection-like API



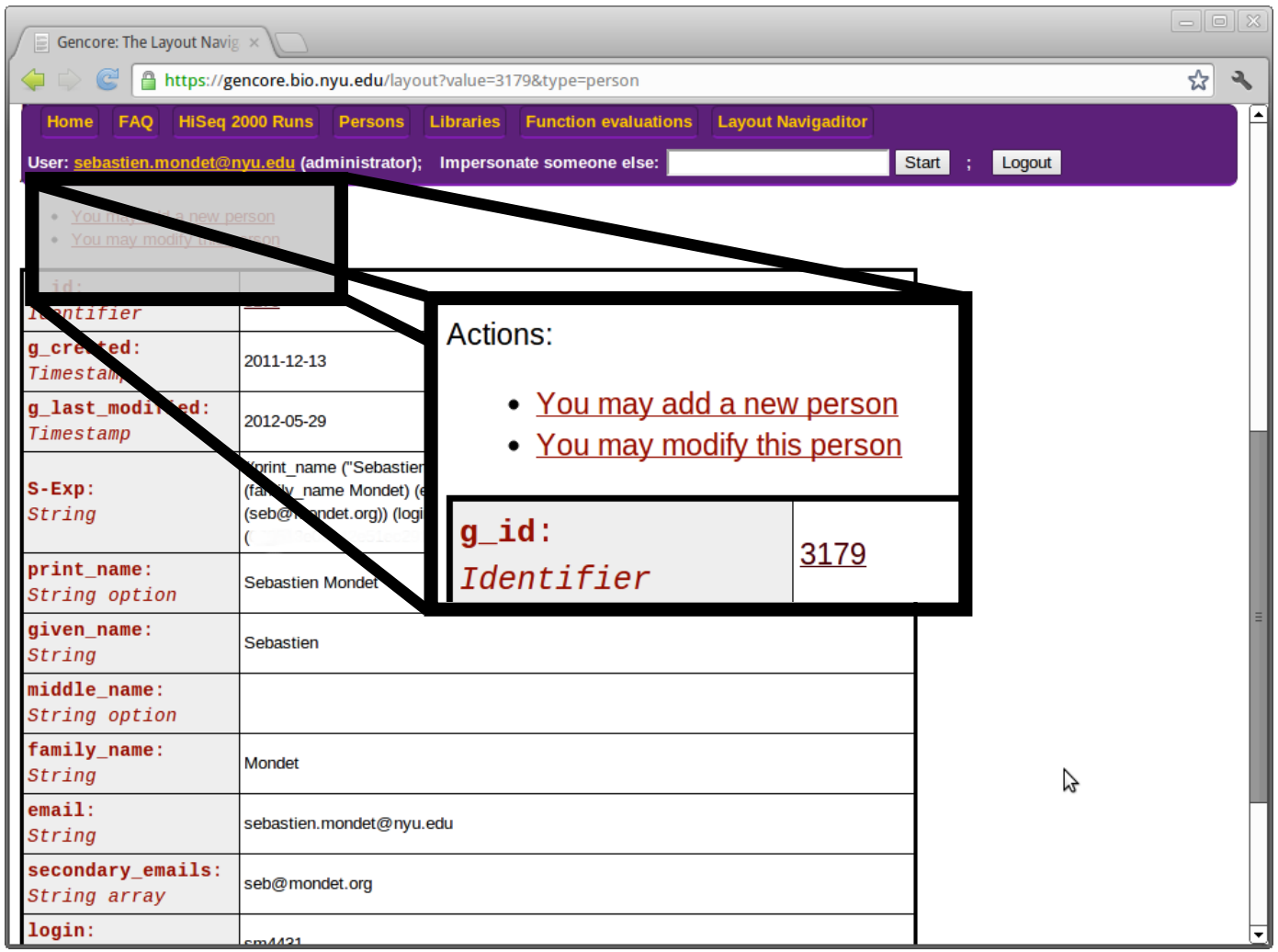
The "Layout" DSL

Introspection-like API



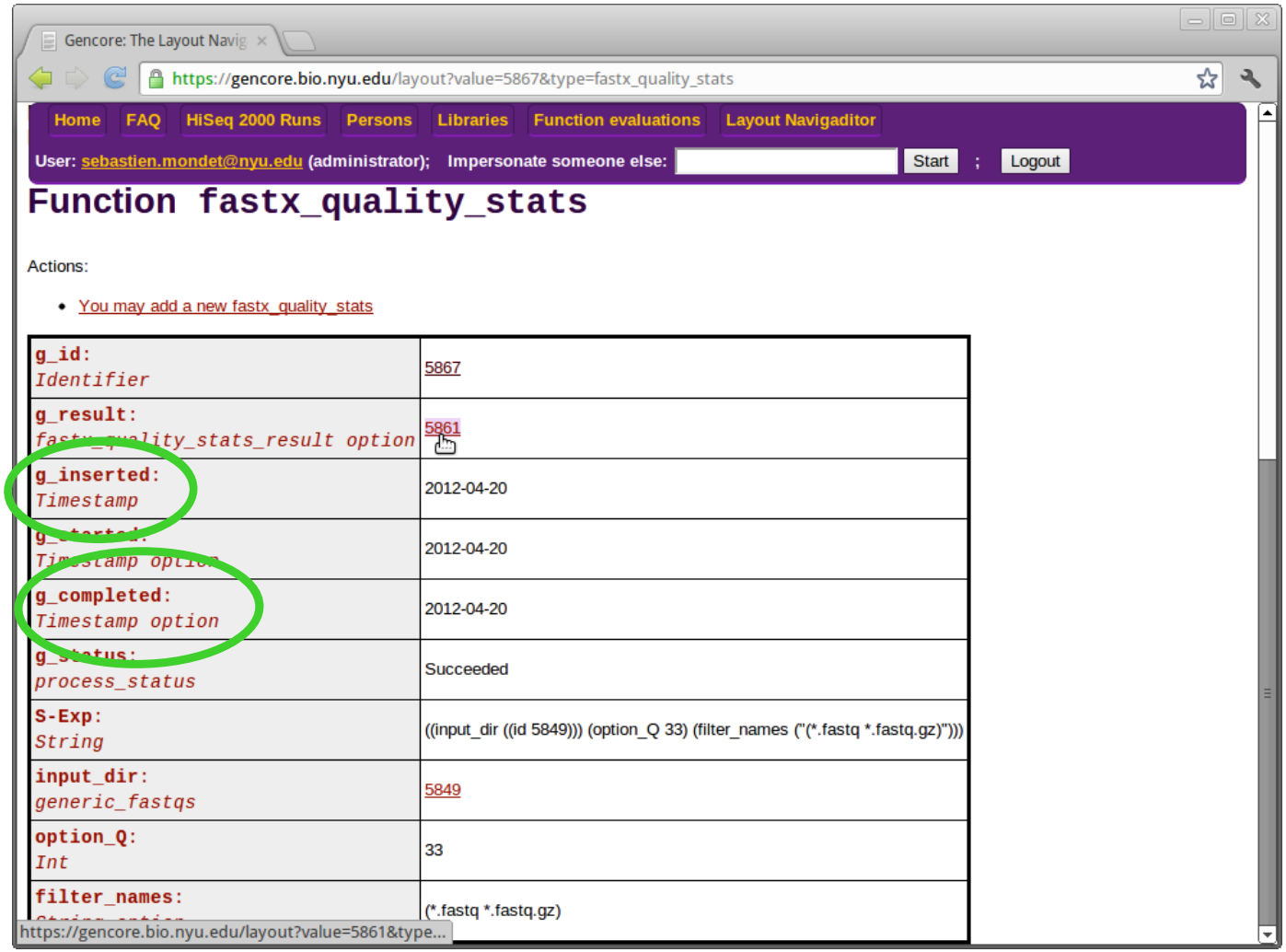
The "Layout" DSL

Introspection-like API



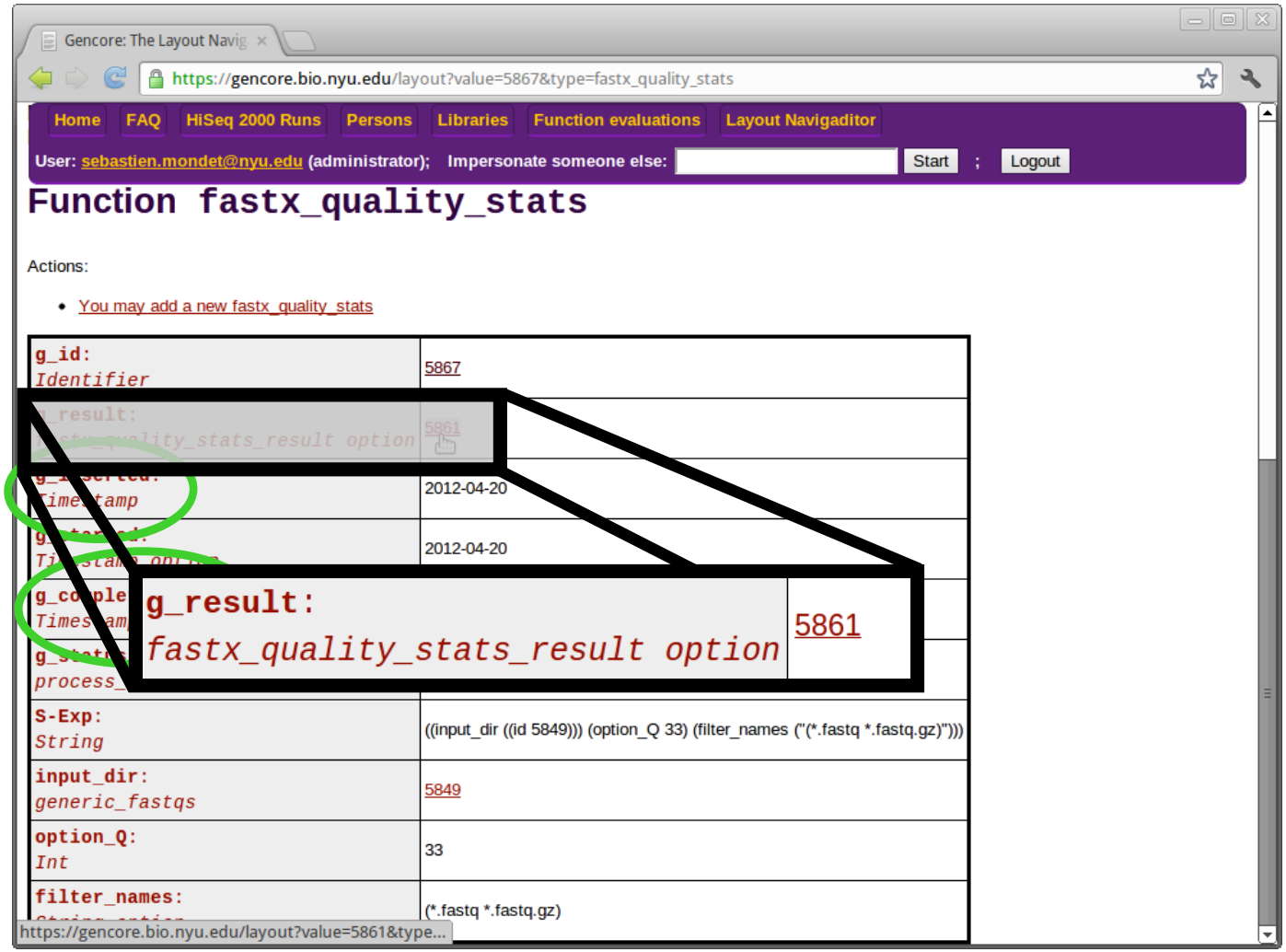
The "Layout" DSL

Introspection-like API



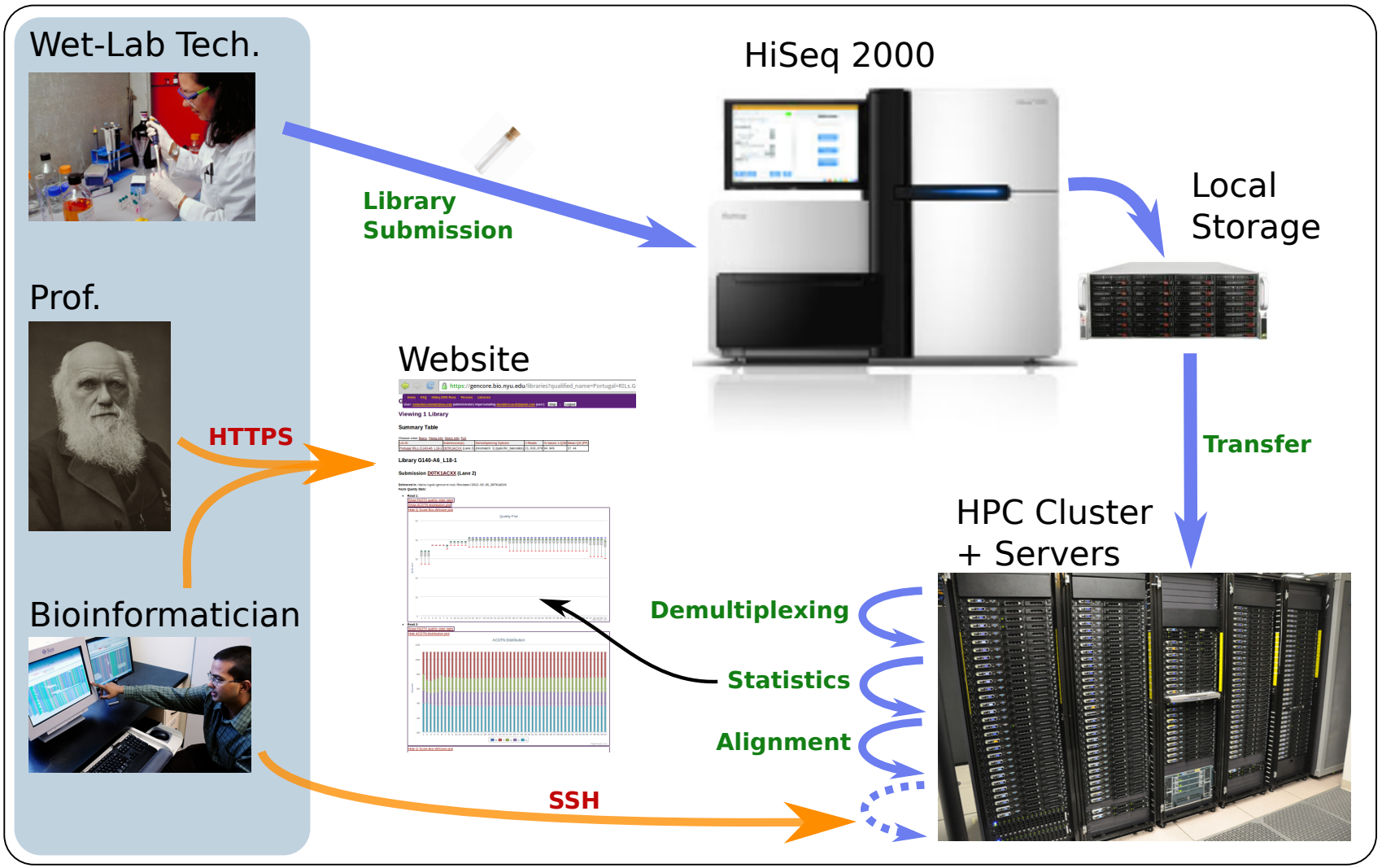
The "Layout" DSL

Introspection-like API



The "Layout" DSL

It's Everywhere



OCaml in Genomics
Experience Report

We use OCaml everywhere with:

- Jane St Core & Batteries
- Lwt
- Ocsigen
- Biocaml
- PG'OCaml, XMLM, Csv, Sqlite, The Cryptokit

OCaml and Asynchronous I/O

Lwt Is Also Everywhere

Writing application-servers & Web-services.

∧

Preemptive threads + shared memory + human beings = ☠ ☣ ☢

⇒

Lwt:

- Light-weight threads — Monadic non-preemptive I/O
- Don't block — Don't get preempted
- `ocsigen.org/lwt/`

OCaml and Asynchronous I/O

Lwt Is Also Everywhere

```
perform_some_complex_io x y z
>>= fun result ->
(* toy "shared mutable state" example: *)
let aux = !global_a in
global_a := !global_b;
global_b := result
return ()
```

OCaml and Asynchronous I/O

Lwt Is Also Everywhere

We actually don't like exceptions:

- Embed a Result/Error monad → *Flow monad*
- Use polymorphic variants for the "error side" (extensible at will + exhaustive pattern check)

```
module type Flow = sig
  type ('ok, 'err) monad
  val bind : ('ok, 'err) monad -> ('ok -> ('oknext, 'err) monad)
       -> ('oknext, 'err) monad
  val return : 'ok -> ('ok, 'any) monad
  val error : 'err -> ('any, 'err) monad
end
```

OCaml and Asynchronous I/O

An Example — Error Management

```
let f i =
  match i with
  | 0 -> return ()
  | 1 -> error ‘its_one
  | 2 -> error ‘its_two
  | n -> error (‘its_a_lot n)
```

```
val f :
  int -> (unit, [> ‘its_a_lot of int | ‘its_one | ‘its_two ])
       Flow.t
```

OCaml and Asynchronous I/O

An Example — Error Management

```
bind_on_error (f 42)
  (function
  | `its_one -> eprintf "One!\n"; return ()
  | `its_a_lot n -> eprintf "A lot: %d\n"; return ())
```

```
        Characters 51-59:
      | `its_one -> eprintf "One!\n"; return ()
        ^^^^^^^^
Error: This pattern matches values of type [< `its_a_lot of 'a |
     `its_one ]
       but a pattern was expected which matches values of type
         [> `its_a_lot of int | `its_one | `its_two ]
       The first variant type does not allow tag(s) `its_two
```

OCaml In The Wild

It's Super-Cool

OCaml is not only well-typed:

- Industrial-Strength (core and libraries/frameworks).
- Hackability (Bypass tools, extend build-system).
- Objects and Polymorphic Variants.
- The Future (Coq).

It's being improved on:

- The Marketing.
- The Programmer's Toolkit.

Ocsigen
The Way To Go

A concentrate of awesomeness:

- **Well-Typed** web-programming
  ⇒ HTML5 **and** services **and** client code !
- `Eliom_output.Caml.register*` ! (⇔ RPC-like programming)
- Choices do not get on your way
  DB design, templating, "there is more than one way" …
- Statically linked native webserver/application.
- `js_of_ocaml`: great but still limited by JS/DOM.
  Something is missing there …

Thanks
Any Questions?

Contacts:

- Ashish Agarwal
  `ashish.agarwal@nyu.edu`
  `http://ashishagarwal.org/`
- Sebastien Mondet
  `sebastien.mondet@nyu.edu`
  `http://seb.mondet.org`

References I

- `[BWA09]` Heng Li, and Richard Durbin ; *Fast and accurate short read alignment with Burrows-Wheeler transform.* Bioinformatics Volume 25 Issue 14, 2009.
- `[Bowtie09]` Ben Langmead, Cole Trapnell, Mihai Pop, and Steven L Salzberg ; *Ultrafast and memory-efficient alignment of short DNA sequences to the human genome*. Genome Biol. Volume 10 Issue 3, 2009.
- `[Assembly10]` Jason R. Miller, Sergey Koren, and Granger Sutton ; *Assembly algorithms for next-generation sequencing data.* Genomics. 2010 June; 95(6): 315–327, 2010.

References II

- [Annot09] Peter Bakke, Nick Carney, Will DeLoache, Mary Gearing, Kjeld Ingvorsen, Matt Lotz, Jay McNair, Pallavi Penumetcha, Samantha Simpson, Laura Voss, Max Win, Laurie J. Heyer, and A. Malcolm Campbell ; *Evaluation of Three Automated Genome Annotations for Halorhabdus utahensis*. PLoS ONE 4(7): e6291, 2009.

The Flow Monad
Why and What

Cooperative (or not) Threads:

- Be able to use *Lwt*, *Async*, or standard preemptive threading.
  ⇒ Functor over an I/O monad.

Don't like exceptions:

- Need a **Result Monad** (a.k.a. "error monad")
  Already using a monad ⇒ Monad Transformer

```
module type Result_IO_monad = sig
  type ('ok, 'err) monad
  val bind : ('ok, 'err) monad -> ('ok -> ('oknext, 'err) monad)
       -> ('oknext, 'err) monad
  val return : 'ok -> ('ok, 'any) monad
  val error : 'err -> ('any, 'err) monad
end
```

The "Layout" DSL

An Example — Generated Code

```
with_database configuration (fun ~dbh ->
  let layout = Classy.make dbh in
  layout#library#all
  >>= map_sequential ~f:(fun lib -> lib#preparator#get)
  >>| List.dedup
  >>= map_sequential ~f:(fun prep_person ->
    prep_person#set_roles (`preparator :: prep_person#roles))
  >>= fun _ ->
  return ())
```