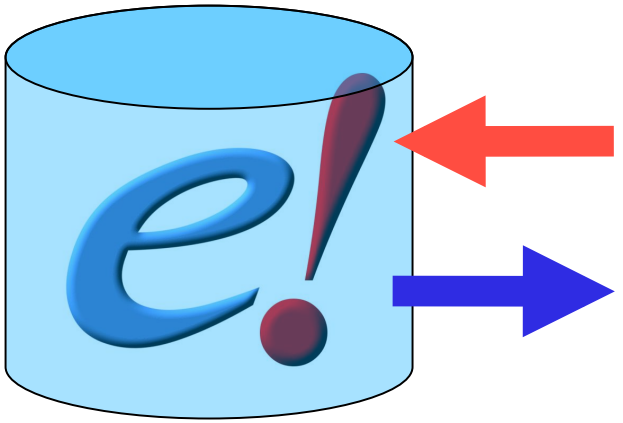
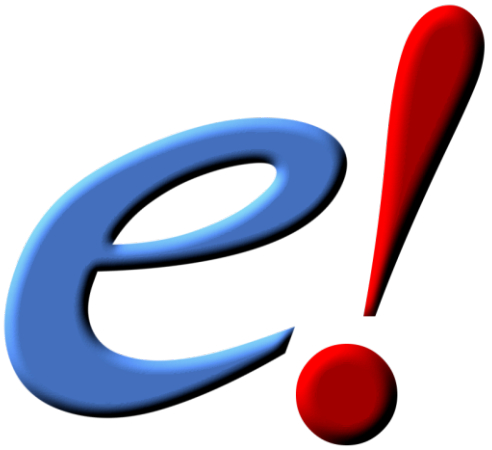
Ensembl Core API



EMBL-EBI

wellcome trust sanger institute



Ensembl Databases

- Ensembl stores the data in several databases:
  - 1 core DB per species (genome, genes, xref...)*
  - 1 variation DB per species (SNPs, strains, individuals...)
  - 1 single compara DB for all the comparative genomics data
  - and many more...

(*) It is possible to store more than 1 species per core DB (e.g. bacterias)

Exercise:

- Connect to the ensembldb.ensembl.org database and look at the list of databases:

```
$> mysql -u anonymous -h ensembldb.ensembl.org –P 3306
mysql> show databases;
```

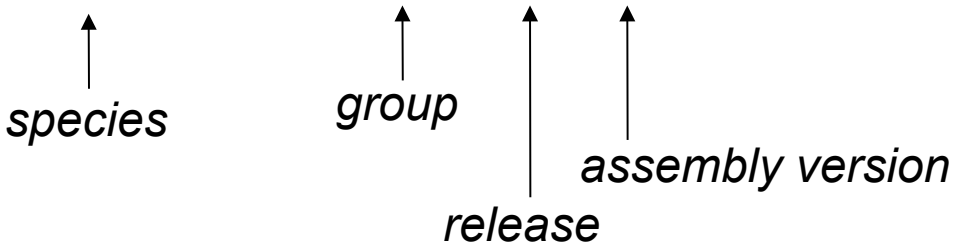
The Ensembl Core databases

The Ensembl Core databases store:

- genomic sequence
- assembly information
- gene, transcript and protein models
- cDNA and protein alignments
- cytogenetic bands, markers, repeats, CpG islands etc.
- external references

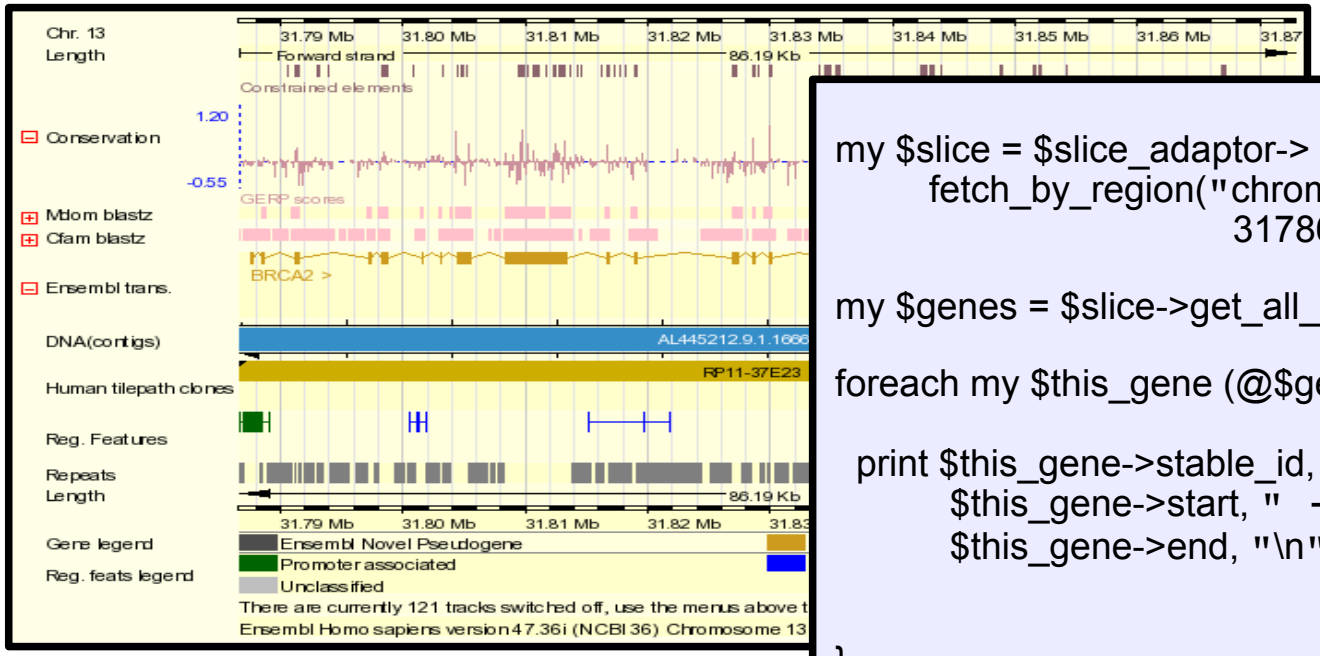
Database name convention:

homo_sapiens_core_74_37



The Perl API

- Written in Object-Oriented Perl.
- Used to retrieve data from and to store data in Ensembl databases.
- Foundation for the Ensembl Pipeline and Ensembl Web interface.




```
my $slice = $slice_adaptor->
        fetch_by_region("chromosome", "13",
                                31786617, 31872809);

my $genes = $slice->get_all_Genes();

foreach my $this_gene (@$genes) {

  print $this_gene->stable_id, ": ",
          $this_gene->start, "  -  ",
          $this_gene->end, "\n";


}
```

Perl Objects (1)

An object is an instance of a class

- $woofy is an instance of the class “dog”

It has attributes and methods

- $woofy->{'breed'} # “greyhound”
- $woofy->speak( ) # “woof, woof, I am greyhound”

The class defines all the attributes and methods, but each object has its own copy of the attributes:

- $woofy->{'birthday'} # 25 Apr 2012
- $laika->{'birthday'} # 20 May 2010

The methods use the attributes:

- $woofy->get_age( ) # 9 months
- $laika->get_age( ) # 2 years and 8 months

Perl Objects (2)

Abstraction

- We don't need to know how the information is stored in order to use the object. The data structure can change, but the methods can remain the same

Encapsulation

- Information and methods are stored together.

Inheritance

- Classes can inherit attributes and methods from other classes. For instance, the class “dog” can inherit from the class “animal” which can inherit from the class “living organism”

Polymorphism

- A child class can redefine a method from a parent class to better suit its needs.

Perl Objects in the Ensembl API

We will avoid accessing the attributes directly, we will use the methods

Methods are Perl subroutines typically accessed from the object only

Methods are called using the “arrow” (->) operator:

- $dog->speak( );

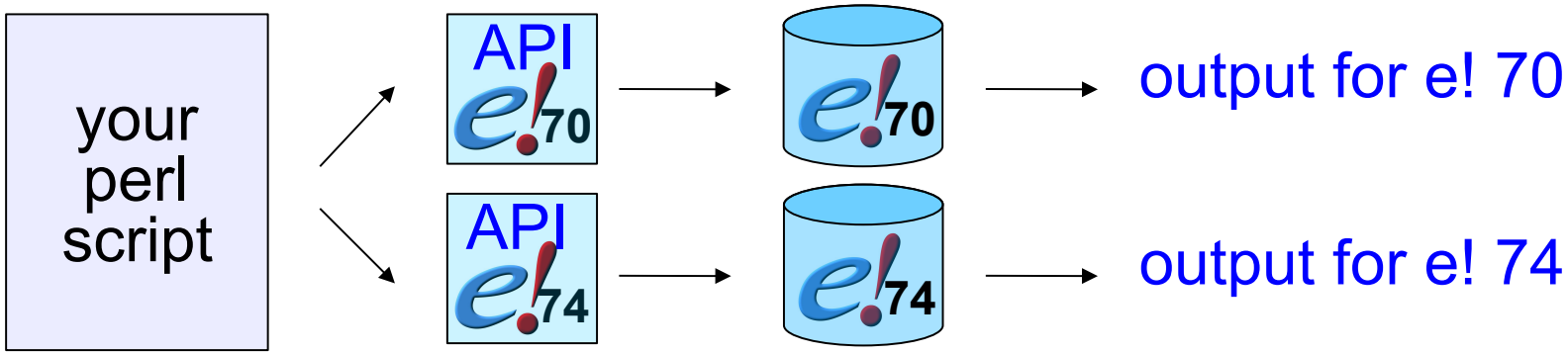
Many methods are getter/setter. They can be used to get or to set a value. For instance:

- GET (no arg.): my $gene_name = $gene->name( );
- SET (new value): $gene->name(“ENSG00000000123152”);

Why use an API?

- Uniform method of access to the data.
- Avoid writing the same thing twice: reusable in different systems.
- Reliable: lots of hard work, testing and optimisation already done.
- Insulates developers to underlying changes at a lower level (i.e. the database).

- NB: API version **must** match database version. Old scripts using the API *should* continue working with a newer API!



Documentation & Help

- Installation instructions, web-browsable version of the POD (Perldoc) and tutorial

  http://www.ensembl.org/info/docs/api/index.html

- Inline Perl POD (Plain Old Documentation)
- ensembl-dev mailing list:

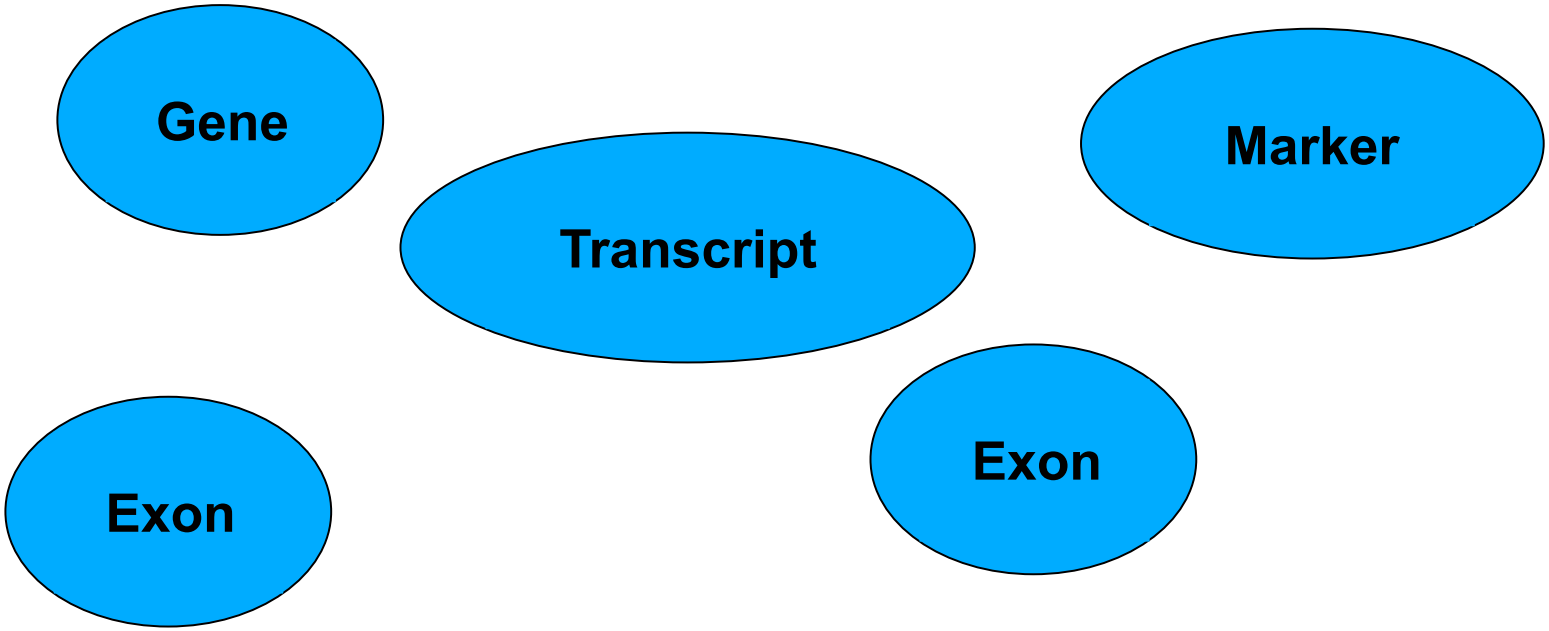
  http://www.ensembl.org/info/about/contact/mailing.html

- Ensembl helpdesk:

  helpdesk@ensembl.org

Data Objects

- Information is obtained from the API in the form of *Data Objects*.
- A *Data Object* represents a piece of data that is (or can be) stored in the database.



Object Adaptors

- *Object Adaptors* are factories for *Data Objects*.
- *Data Objects* are retrieved from and stored in the database using *Object Adaptors*.
- Each *Object Adaptor* is responsible for creating objects of only one particular type. For instance:
  - The GeneAdaptor is used to fetch Gene objects
  - The SliceAdaptor is used to fetch Slice objects
- *Object Adaptor* *fetch*, *store*, and *remove* methods are used to retrieve, save, and delete information in the database.
- 2 types of methods:
  - **fetch_by_....** returns 1 object (or undef)
  - **fetch_all_by_...** returns a ref. to an array of objects (or ref. to an empty array)

Arrays and references in Perl

- An array (AKA list) contains several values:

```
my @array = ("one", "two", "three");
print join(" -- ", $array[0], $array[1], $array[2]), "\n";
one -- two -- three
print join(" -- ", @array), "\n";
one -- two -- three
```

- A reference to an array can be dereferenced to get the values:

```
my $array_ref = ["one", "two", "three"];
print join(" -- ", @$array_ref), "\n";
one -- two -- three
```

- References to arrays are widely used in the Ensembl API because it is much more efficient to use them to return values

Database Adaptors

Database Adaptors are Object Adaptor factories

Database Adaptors are used to connect to a single database

There is 1 core database per species. You need several Database Adaptors to get data for different species

Ensembl core modules

- Name space for the modules:
  - Object modules start with Bio::EnsEMBL
    - Bio::EnsEMBL::Gene for gene objects
    - Bio::EnsEMBL::Slice for slice objects
  - ObjectAdaptors start with Bio::EnsEMBL::DBSQL
    - the GeneAdaptor is a Bio::EnsEMBL::DBSQL::GeneAdaptor
    - the SliceAdaptor is a Bio::EnsEMBL::DBSQL::SliceAdaptor
- To get the documentation about a particular module:
  - perldoc Bio::EnsEMBL::Slice
  - http://www.ensembl.org/info/docs/api/index.html
- Naming conventions for the methods:
  - For retrieving objects from the ObjectAdaptors:
    - fetch_by_ and fetch_all_by_
  - For retrieving Objects attributes:
    - get_Object and get_all_Objects

The Registry

- Needed to tell your script which databases to use
- Enables you to specify configuration scripts (advanced mode)

load_registry_from_db *or* **load_registry_from_url**

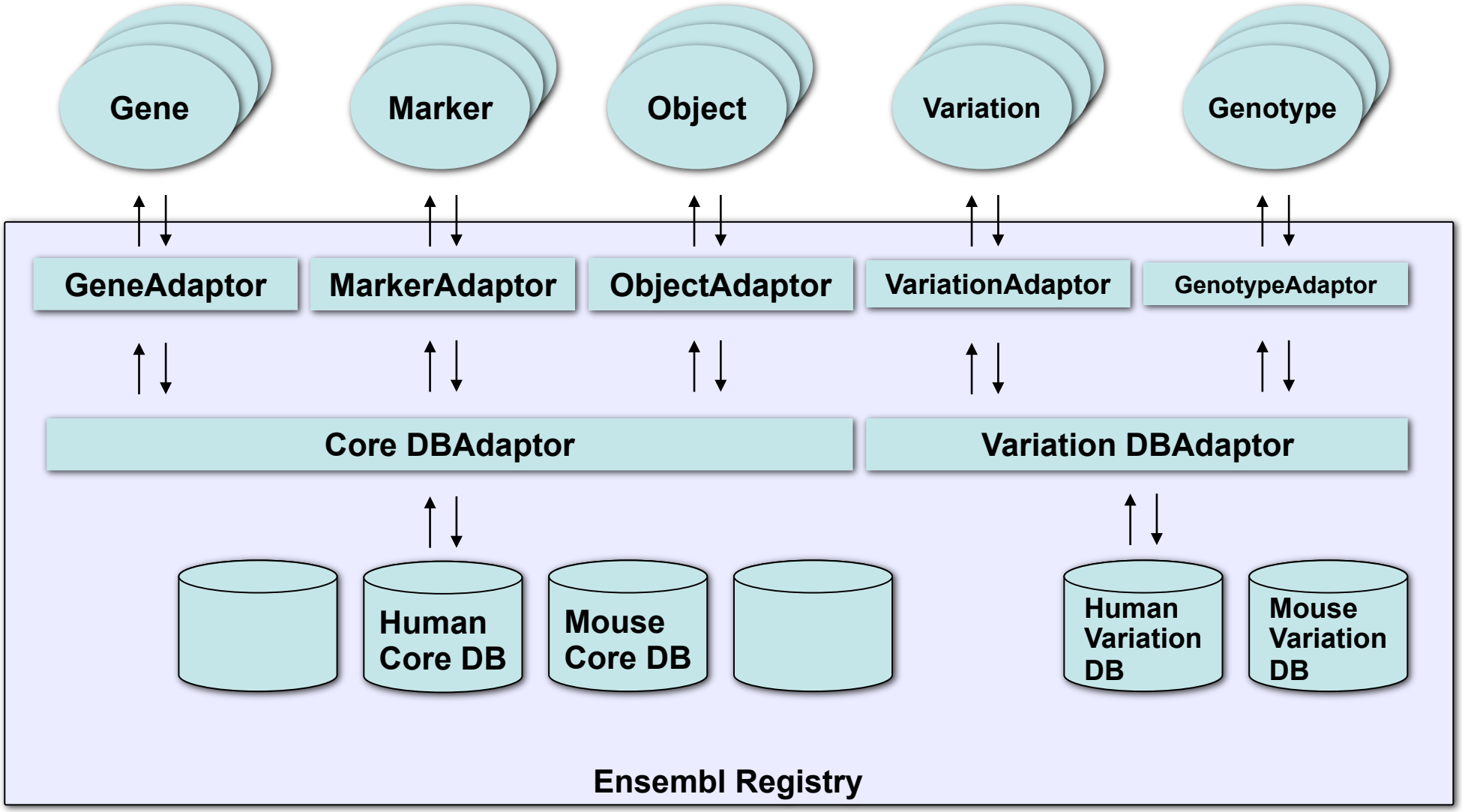
- automatic configuration of the Registry
- loads all the databases for the correct version of this API
  - Ensures you use the correct API and Database
  - Lazy loads so no database connections are made until requested

```
use Bio::EnsEMBL::Registry;

my $reg = "Bio::EnsEMBL::Registry";

$reg->load_registry_from_url('mysql://anonymous@ensembldb.ensembl.org');
```

System architecture



Exercise 1

a) **Create a script that uses the function load_registry_from_db to load all databases into the registry and print the names of the databases loaded.**

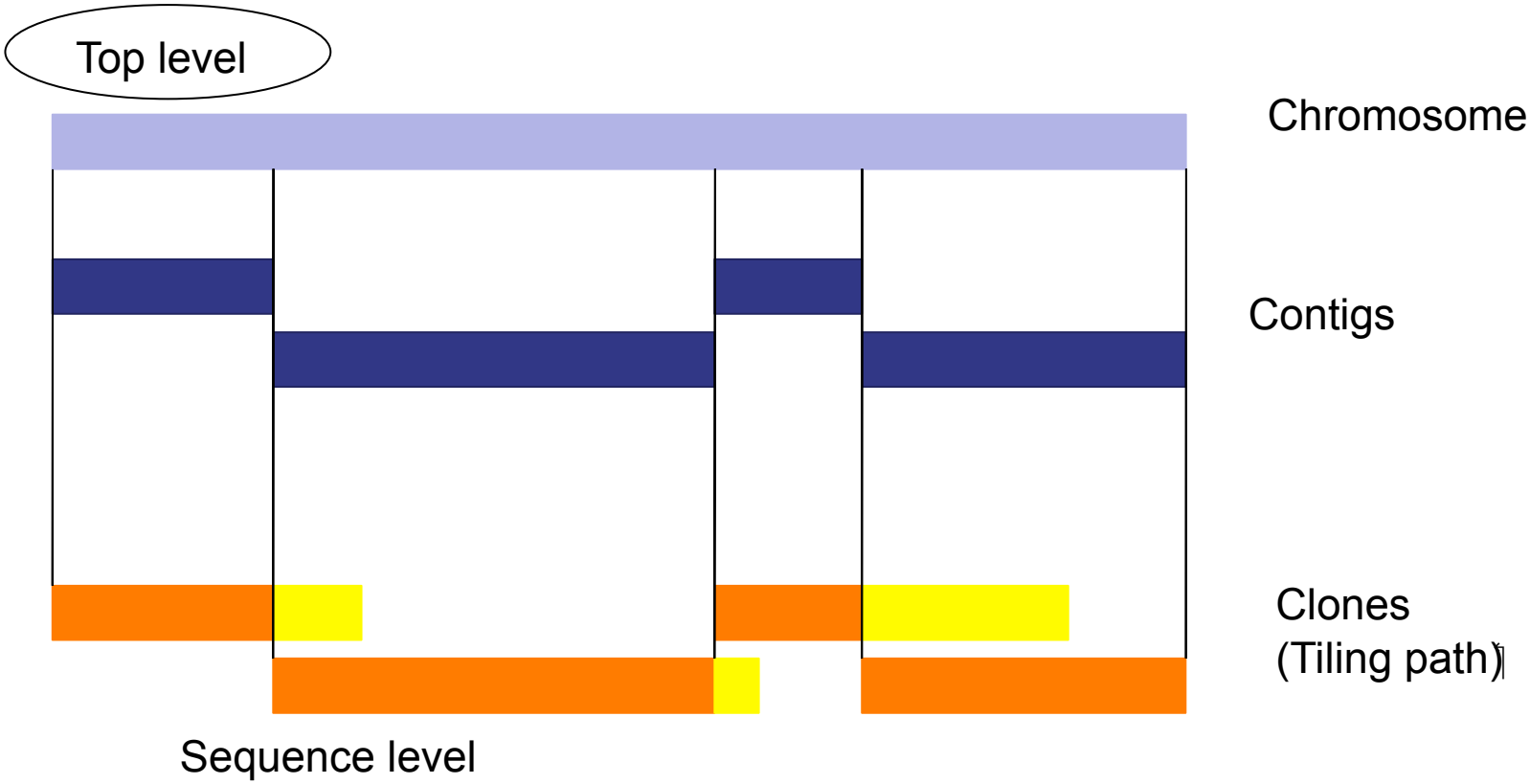
Sequence Regions

Sequences stored in Ensembl are associated with Sequence Regions (Bio::EnsEMBL::SeqRegion objects)

Sequence Regions are linked to a distinct hierarchy of Coordinate Systems

Coordinate Systems

- Ensembl stores features and sequence in a number of coordinate systems.
- Example coordinate systems: chromosome, clone, scaffold, contig



Code Example

```
#   Obtain all coordinate systems for human

use Bio::EnsEMBL::Registry;

my $registry = 'Bio::EnsEMBL::Registry';

$registry->load_registry_from_db(
    -host => 'ensembldb.ensembl.org',
    -user => 'anonymous'
);

my $coordsystem_adaptor = $registry->get_adaptor( 'Human', 'Core',
   'CoordSystem' );

my $coordsystems = $coordsystem_adaptor->fetch_all;

while ( my $coordsystem = shift @{$coordsystems} ){
    print $coordsystem->name, "\t", $coordsystem->version, "\n";
}
```

Code Example

OUTPUT:

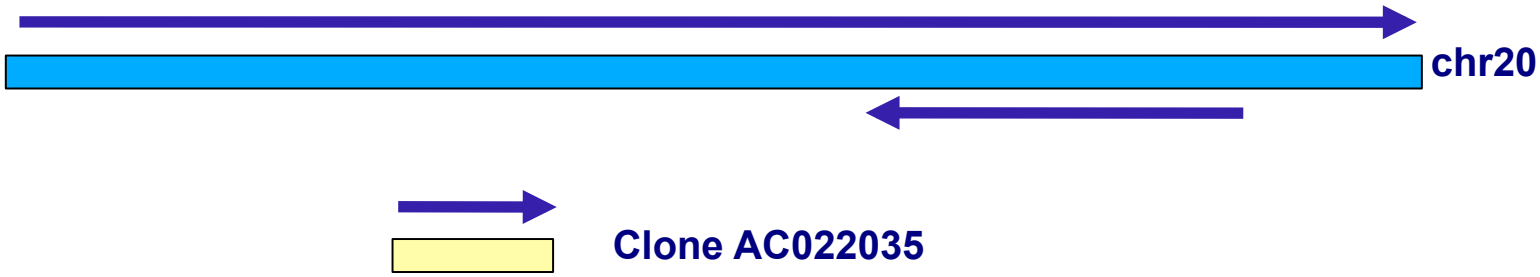
```
chromosome  GRCh37
supercontig
clone
contig
chromosome  NCBI36
chromosome  NCBI35
chromosome  NCBI34
lrg
```

} Old assemblies, used for mapping

Locus-Reference Genes, used in clinical tests

Slices

- A *Slice* Data Object represents an arbitrary region of a genome, a slice of a Sequence Region.
- *Slices* are not directly stored in the database.
- A *Slice* is used to request sequence or features from a specific region in a specific coordinate system.



Code Example

```
# Obtain a slice covering the entire human Y chromosome

my $slice_adaptor = $registry->get_adaptor( 'Human', 'Core', 'Slice' );

my $slice = $slice_adaptor->fetch_by_region( 'chromosome', 'Y' );

print
  "Coord system:\t", $slice->coord_system_name,
  "\nSeq region:\t", $slice->seq_region_name,
  "\nStart:\t\t", $slice->start,
  "\nEnd:\t\t", $slice->end,
  "\nStrand:\t\t", $slice->strand,
  "\nSlice:\t\t", $slice->name, "\n";
```

Code Example

OUTPUT:

```
Coord system:   chromosome
Seq region:     Y
Start:          1
End:                59373566
Strand:         1
Slice:          chromosome:GRCh37:Y:1:59373566:1
```

Ensembl Perl script design

- Always:
  - Load the registry
- Which features (genes, repeats, SNPs, etc.) are in my particular region of interest?
  - Get the SliceAdaptor
  - Fetch the Slice for you region of interest
  - Get the features from your Slice
- What do we know about a particular gene (or any other feature)?
  - Get the GeneAdaptor
  - Fetch your Gene of interest
  - Get more details about the gene:
    - Gene structure (transcripts, exons, translations)
    - Annotations: GO xrefs, HGNC symbols, etc.
    - Features in the same region -> get the Slice for the Gene!

My first Ensembl script: Hello Slice!

```
use warnings;
use strict;

#Load the registry
use Bio::EnsEMBL::Registry;
my $reg = "Bio::EnsEMBL::Registry";
$reg->load_registry_from_url(
   'mysql://anonymous@ensembldb.ensembl.org');

# Get the Slice Adaptor for human
my $slice_adaptor = $reg->get_adaptor("Homo sapiens", "core", "Slice");

# Fetch a slice
my $slice = $slice_adaptor->fetch_by_region(
   "chromosome", "13", 31_000_001, 31_000_100);

# Print its name
print "Hello slice: ", $slice->name, "\n";
```

Always start a script with these!

Mind the single quotes!

Species_name, db_type, Object_type

Coord_system, name, start, end

Coordinates can have underscores for readability

Exercise 2

a) **Fetch the sequence of chromosome 20 from 100,000 to 110,000**

b) **Fetch a slice for the ENSG00000101266 gene**

c) **Determine the number of genes on the first 10MB of chromosome 20.**

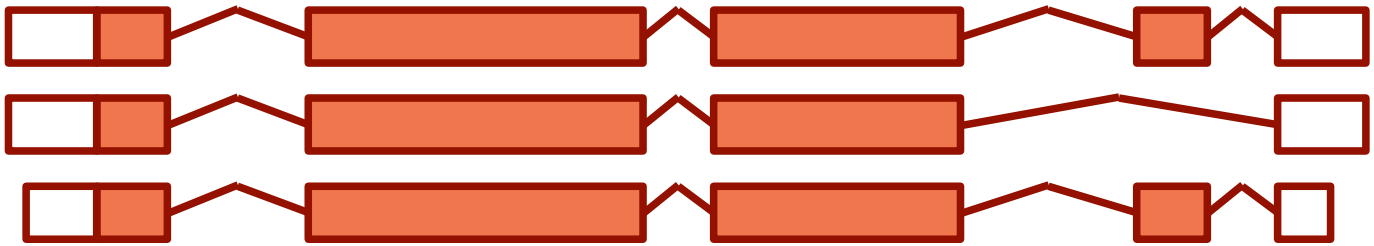
Genes, Transcripts and Exons

Genes, Transcript and Exons are Feature Data Objects

A Gene is a grouping of Transcripts which share any (partially) overlapping Exons

A Transcript is a set of Exons

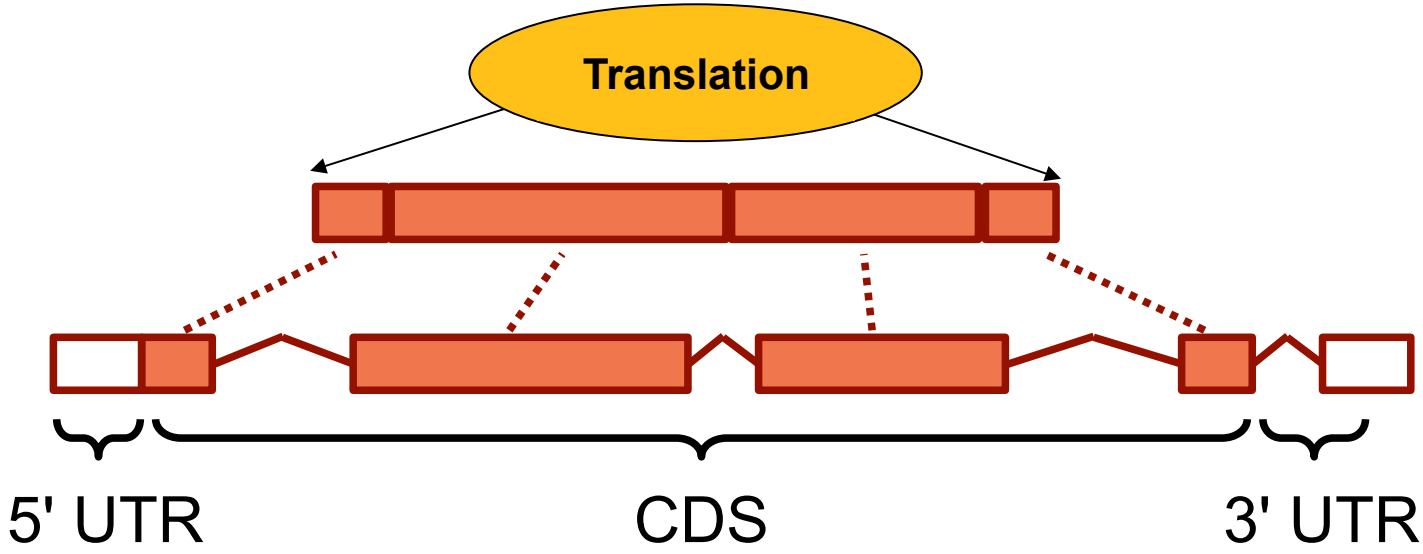
Introns are not explicitly defined in the database



Translations

- Translations are not Features.
- A Translation object defines the UTR and CDS of a Transcript.
- Peptides are not stored in the database, they are computed on the fly using Transcript objects.

- NB: ncRNA don't have a translation!



Biotypes

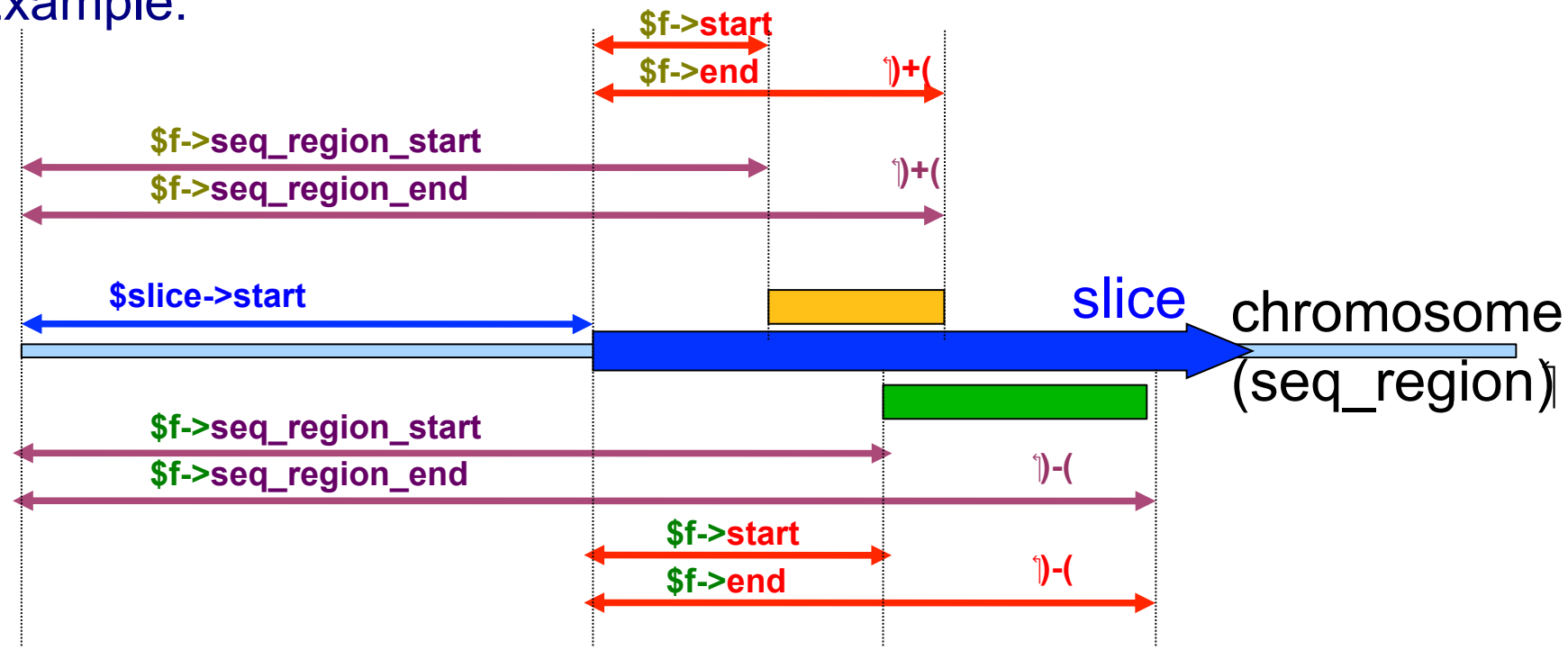
- Genes are classified by biotype

protein_coding
pseudogene
processed_transcript
snRNA
miRNA
snoRNA
lincRNA
misc_RNA
scRNA_pseudogene
Mt_tRNA_pseudogene
rRNA
IG_V_gene
rRNA_pseudogene
IG_V_pseudogene
tRNA_pseudogene
LRG_gene

IG_J_gene
snRNA_pseudogene
snoRNA_pseudogene
TR_V_gene
IG_D_gene
Mt_tRNA
TR_V_pseudogene
IG_C_gene
polymorphic_pseudogene
miRNA_pseudogene
TR_J_gene
IG_C_pseudogene
IG_J_pseudogene
misc_RNA_pseudogene
TR_C_gene
Mt_rRNA

Features and coordinates

- All the Features (Gene, Exon, RepeatFeature, SNP, etc.) have a slice, start, end and strand values that locate the Feature on the genome.
- start, end and strand refer to the slice while seq_region_start, seq_region_end and seq_region_strand refer to the whole chromosome
- Example:



Exercise 3

a) **Fetch the gene with id 'ENSG00000101266'**

b) **Get all the transcripts and proteins for the above gene**

c) **Get the list of the different biotypes for the genes obtained in Exercise 2c**

External references

External References (Xrefs) are cross references to identifiers from other databases, e.g. HGNC, WikiGenes, UniProtKB/Swiss-Prot, RefSeq, MIM etc. etc.

External References can be on the gene, transcript or protein level

Code Example

```
# Obtain external references for Ensembl gene ENSG00000139618

my $gene = $gene_adaptor->fetch_by_stable_id( 'ENSG00000139618' );

my $gene_xrefs = $gene->get_all_DBEntries;

print "Xrefs on the gene level: \n\n";
foreach my $gene_xref( @{$gene_xrefs} ){
  print $gene_xref->dbname, ":", $gene_xref->display_id, "\n";
}

my $all_xrefs = $gene->get_all_DBLinks;

print "\nXrefs on the gene, transcript and protein level: \n\n";
foreach my $all_xref( @{$all_xrefs} ){
  print $all_xref->dbname, ":", $all_xref->display_id, "\n";
}
```

Code Example

Output:

```
Xrefs on the gene level:

OTTG:OTTHUMG00000017411
HGNC:BRCA2
DBASS3:BRCA2
HGNC_curated_gene:BRCA2

Xrefs on the gene, transcript and protein level:

OTTG:OTTHUMG00000017411
HGNC:BRCA2
DBASS3:BRCA2
HGNC_curated_gene:BRCA2
shares_CDS_and_UTR_with_OTTT:OTTHUMT00000046000
UCSC:uc001uub.1
CCDS:CCDS9344.1
RefSeq_dna:NM_000059.3
HGNC:BRCA2
UniGene:Hs.34012
HGNC_curated_transcript:BRCA2-001
EntrezGene:BRCA2
WikiGene:BRCA2
MIM_MORBID:#114480
MIM_MORBID:#137800
MIM_MORBID:#155255
MIM_MORBID:#155720
MIM_MORBID:#176807
MIM_MORBID:#194070
MIM_MORBID:#227650
MIM_MORBID:#260350
MIM_GENE:*600185
MIM_MORBID:#605724
MIM_MORBID:#612555
MIM_MORBID:#613029
MIM_MORBID:#613347
RefSeq_peptide:NP_000050.2
Uniprot/SPTREMBL:A1YBP1_HUMAN
EMBL:DQ897648
protein_id:ABI74674.1
EMBL:AL445212
Uniprot/SPTREMBL:B2ZAH0_HUMAN
EMBL:EU625579
protein_id:ACD01217.1
EMBL:AL137247
Uniprot/SPTREMBL:Q8IU64_HUMAN
EMBL:AY151039
protein_id:AAN28944.1
EMBL:AF489725
protein_id:AAN61409.1
EMBL:AF489726
protein_id:AAN61410.1
EMBL:AF489727
protein_id:AAN61411.1
EMBL:AF489728
protein_id:AAN61412.1
EMBL:AF489729
protein_id:AAN61413.1
EMBL:AF489730
protein_id:AAN61414.1
EMBL:AF489731
protein_id:AAN61415.1
EMBL:AF489732
protein_id:AAN61416.1
EMBL:AF489733
protein_id:AAN61417.1
EMBL:AF489734
protein_id:AAN61418.1
EMBL:AF489735
protein_id:AAN61419.1
EMBL:AF489736
protein_id:AAN61420.1
EMBL:AF489737
protein_id:AAN61421.1
EMBL:AF489738
protein_id:AAN61422.1
Uniprot/SPTREMBL:Q8IU77_HUMAN
EMBL:AF507079
protein_id:AAN61426.1
EMBL:AF507080
protein_id:AAN61427.1
EMBL:AF507081
protein_id:AAN61428.1
EMBL:AF507082
protein_id:AAN61429.1


Etc.etc.
```

get_all_DBLinks vs get_all_DBEntries

- get_all_DBLinks and get_all_DBEntries return the Xref (cross-references) for this entry.
- Xrefs are associated with:
  - gene (example: HGNC name)
  - transcript (example: affy probe mapping)
  - translation (example GO annotation)
- **get_all_DBEntries** returns the Xrefs for this object only
- **get_all_DBLinks** returns the Xrefs for this object plus all the Xrefs for all the underlying ones (transcripts and corresponding translations)

Exercise 4

a) **Fetch the ensembl gene for BRCA2**

b) **Fetch the ensembl genes for SNORA1**

c) **Print out all other linked database entries for the gene BRCA2.**

More, more, many more things

- Core:
  - Ancestral repeats
  - Mappers & Assembly mappers
  - ESTs, Genscan predictions, ncRNA, Vega genes, CCDS
  - Protein alignments
  - Xrefs, GO annotations
- Variation:
  - SNPs
  - CNVs
  - Personal genomes and species strains
- Functional genomics
  - Array probe mappings
  - ChIP-chip, ChIP-seq, regulatory features
- Comparative genomics
  - Genomic alignments
  - Protein homologies

Acknowledgements


Andy


Stephen


Kieron


Magali

D48–D55 *Nucleic Acids Research, 2013, Vol. 41, Database issue* *Published online 30 November 2012*
doi:10.1093/nar/gks1236

Ensembl 2013

Paul Flicek[1,2,*], Ikhlak Ahmed[1], M. Ridwan Amode[2], Daniel Barrell[2], Kathryn Beal[1], Simon Brent[2], Denise Carvalho-Silva[1], Peter Clapham[2], Guy Coates[2], Susan Fairley[2], Stephen Fitzgerald[1], Laurent Gil[1], Carlos García-Girón[2], Leo Gordon[1], Thibaut Hourlier[2], Sarah Hunt[1], Thomas Juettemann[1], Andreas K. Kähäri[2], Stephen Keenan[1], Monika Komorowska[1], Eugene Kulesha[1], Ian Longden[1], Thomas Maurel[1], William M. McLaren[1], Matthieu Muffato[1], Rishi Nag[2], Bert Overduin[1], Miguel Pignatelli[1], Bethan Pritchard[2], Emily Pritchard[1], Harpreet Singh Riat[2], Graham R. S. Ritchie[1], Magali Ruffier[1], Michael Schuster[1], Daniel Sheppard[2], Daniel Sobral[1], Kieron Taylor[1], Anja Thormann[1], Stephen Trevanion[2], Simon White[2], Steven P. Wilder[1], Bronwen L. Aken[2], Ewan Birney[1], Fiona Cunningham[1], Ian Dunham[1], Jennifer Harrow[2], Javier Herrero[1], Tim J. P. Hubbard[2], Nathan Johnson[1], Rhoda Kinsella[1], Anne Parker[2], Giulietta Spudich[1], Andy Yates[1], Amonida Zadissa[2] and Stephen M. J. Searle[2]

[1]European Bioinformatics Institute, Wellcome Trust Genome Campus, Hinxton Cambridge CB10 1SD, UK and [2]Wellcome Trust Sanger Institute, Wellcome Trust Genome Campus, Hinxton, Cambridge CB10 1SA, UK

wellcome trust

EMBL

National Human Genome Research Institute

BBSRC bioscience for the future

European Commission
Framework Programme 7

SEVENTH FRAMEWORK PROGRAMME

GEN2PHEN

BLUEPRINT epigenome

Quantomics
From Sequence to Consequence :
Tools for the Exploitation of Livestock Genomes