Rules of Thumb

- Most sequences with significant similarity over their entire lengths are homologous.
- Matches that are > 50% identical in a 20-40 aa region occur frequently by chance.
- Distantly related homologs may lack significant similarity. Homologous sequences may have few absolutely conserved residues.
- A homologous to B & B to C $\Rightarrow$ A homologous to C.
- Low complexity regions, transmembrane regions and coiled-coil regions frequently display significant similarity without homology.
- Greater evolutionary distance implies that length of a local alignment required to achieve a statistically significant score also increases.

Perl: Practical Extraction & Report Language

- Created by Larry Wall, early 90s
- Portable, "glue" language for interfacing C/Fortran code, WWW/CGI, graphics, numerical analysis and much more
- Easy to use and extensible
- OOP support, simple databases, simple data structures.
- From interpreted to compiled
- high-level features, and relieves you from manual memory management, segmentation faults, bus errors, most portability problems, etc, etc.
- Competitors: Python, Tcl, Java

Perl Features

- Perl – many features
  - Bit Operations
  - Pattern Matching
  - Subroutines
  - Packages & Modules
  - Objects
  - Interprocess Communication
  - Threads , Process control
  - Compiling

BioPerl

- Routines for handling biosequence and alignment data.
- Why? Human Genome Project: Same project, same data. different data formats! Different input formats. Different output formats for comparable utility programs.
- BioPerl was useful to interchange data and meaningfully exchange results. “Perl Saved the Human Genome Project”
- Many routine tasks automated using BioPerl.
- String manipulations (string operations: substring, match, etc.; handling string data: names, annotations, comments, bibliographical references; regular expression operations)
- Modular: modules in any language

Managing a Large Project

- Devise a common data exchange format.
- Use modules that have already been developed.
- Write Perl scripts to convert to and from common data exchange format.
- Write Perl scripts to “glue” it all together.

Miscellaneous

- pTk – to enable building Perl-driven GUIs for X-Window systems.
- BioJava
- BioPython
- The BioCORBA Project provides an object-oriented, language neutral, platform-independent method for describing and solving bioinformatics problems.

Perl: Examples

```
#!/usr/bin/perl -w
# Storing DNA in a variable, and printing it out

# First we store the DNA in a variable called $DNA
$DNA = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';

# Next, we print the DNA onto the screen
print $DNA;

# Finally, we'll specifically tell the program to
  exit.
exit;  #perl1.pl
```

Perl: Strings

```
#!/usr/bin/perl -w
$DNA1 = 'ACGGGAGGACGGGAAAATTACTACGGCATTAGC';
$DNA2 = 'ATAGTGCCGTGAGAGTGATGTAGTA';
# Concatenate the DNA fragments
$DNA3 = "$DNA1$DNA2";
print "Concatenation 1):\n\n$DNA3\n\n";
# An alternative way using the "dot operator":
$DNA3 = $DNA1 . $DNA2;
print "Concatenation 2):\n\n$DNA3\n\n";
# transcribe from DNA to RNA; make rev comp; print;
$RNA = $DNA3; $RNA =~ s/T/U/g;
$rev = reverse $DNA3; $rev =~ tr/AGCTacgt/TCGAtgca/;
print "$RNA\n$rev\n";
exit;  #perl2.pl
```

Perl: arrays

```
#!/usr/bin/perl -w
# Read filename & remove newline from string
$protFile = <STDIN>; chomp $protFile;
# First we have to "open" the file
unless (open(PROTEINFILE, $protFile) {
  print "File $protFile does not exist"; exit;}
# Each line becomes an element of array @protein
@protein = <PROTEINFILE>;
print @protein;
# Print line #3 and number of lines
print $protein[2], "File contained ", scalar
  @protein, " lines\n";
# Close the file.
close PROTEINFILE;
exit; #perl3.pl
```

Perl: subroutines

```
#!/usr/bin/perl -w
# using command line argument
$dna1 = $ARGV[0]; $dna2 = $ARGV[1];
# Call subroutine with arguments; result in $dna
$dna = addACGT($dna1, $dna2);
print "Add ACGT to $dna1 & $dna2 to get $dna\n\n";
exit;
###### addACGT: concat $dna1, $dna2, & "ACGT". ######
sub addGACAGT {
    my($dnaA, $dnaB) = @_; my($dnaC) = $dnaA.$dnaB;
    $dnaC .= ‘GACAGT';
    return $dnaC;
} #perl4.pl
```

BioPerl Modules

- **Bio::PreSeq**, module for reading, accessing, manipulating, analyzing single sequences.
- **Bio::UnivAln**, module for reading, parsing, writing, slicing, and manipulating multiple biosequences (sequence multisets & alignments).
- **Bio::Struct**, module for reading, writing, accessing, and manipulating 3D structures.
- Support for invoking **BLAST** & other programs.
- Download URL [http://www.bioperl.org/Core/Latest/]
- Tutorial [http://www.bioperl.org/Core/Latest/bptutorial.html]
- Course [http://www.pasteur.fr/recherche/unites/sis/formation/bioperl/index.html]

BioPerl Sequence Object

```
$seqobj->display_id(); # readable id of sequence
$seqobj->seq(); # string of sequence
$seqobj->subseq(5,10); # part of the sequence as a string
$seqobj->accession_number(); # if present, accession num
$seqobj->moltype(); # one of 'dna','rna','protein'
$seqobj->primary_id(); # unique id for sequence independent
                       # of its display_id or accession number
```

Sequence Formats in BioPerl

```
#! /local/bin/perl -w

use strict;
use Bio::SeqIO;
my $in  = Bio::SeqIO->newFh ( -file   => '<seqs.html',
                              -format => 'swiss' );
my $out = Bio::SeqIO->newFh ( -file   => '>seqs.fasta',
                              -format => 'fasta' );

print $out $_ while <$in>;

exit; #bioperl1.pl
```

Sequence Formats in BioPerl

```
#! /local/bin/perl -w
use strict;
use Bio::SeqIO;
my $in  = Bio::SeqIO->new ( -file   => 'seqs.html', -format => 'swiss' );
my $out = Bio::SeqIO->new ( -file   => 'seqs.fas', -format => 'fasta' );

while ($seq = $in->next_seq()) {
    $accNum = $seq->accession_number();
    print "Accession# = $accNum\n";
    $out->write_seq($seq);
}

exit; #bioperl2.pl
```

BioPerl

```
#!/usr/bin/perl -w
# define a DNA sequence object with given sequence
$seq = Bio::Seq->new('-seq'=>'actgtggcgtcaact',
   '-desc'=>'Sample Bio::Seq object',
   '-display_id' => 'somethingxxx',
   '-accession_number' => 'accnumxxx',
   '-alphabet' => 'dna' );
$gb = new Bio::DB::GenBank();

$seq = $gb->get_Seq_by_id('MUSIGHBA1'); #returns Seq object
$seq = $gb->get_Seq_by_acc('AF303112'); #returns Seq object
# this returns a SeqIO object :
$seqio = $gb->get_Stream_by_batch([ qw(J00522 AF303112)]));
exit; #bioperl3.pl
```

Sequence Manipulations

```
#!/local/bin/perl -w
use Bio::DB::GenBank;
$gb = new Bio::DB::GenBank();
$seq1 = $gb->get_Seq_by_acc('AF303112');
$seq2=$seq1->trunc(1,90);
$seq2 = $seq2->revcom();
print $seq2->seq(), "\n";
$seq3=$seq2->translate;
print $seq3->seq(), "\n";
exit; #bioperl4.pl
```

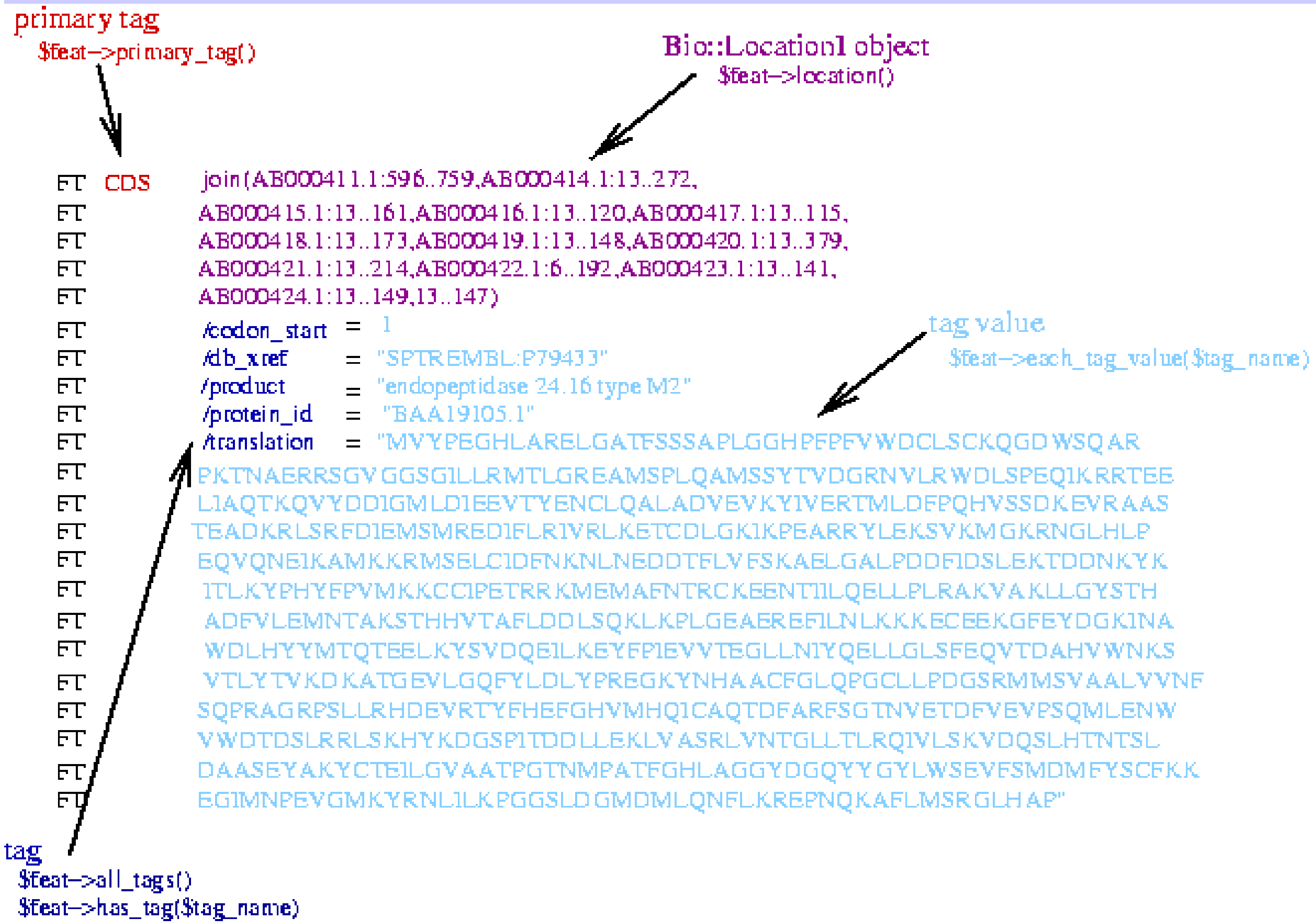
BioPerl:Download GenBank Sequences

```
#!/local/bin/perl -w

use Bio::DB::GenBank;

my $gb = new Bio::DB::GenBank(
    -retrievaltype=>'tempfile', -format=>'Fasta');

my ($seq) = $seq = $gb->get_Seq_by_id("5802612");
print $seq->id, "\n";
print $seq->desc(), "Sequence: \n";
print $seq->seq(), "\n";
exit; #bioperl5.pl
```

BioPerl: Structure

```
use Bio::Structure::IO;
$in = Bio::Structure::IO->new(-file => "inputfilename" , '-
    format' => 'pdb');
$out = Bio::Structure::IO->new(-file =>
    ">outputfilename" , '-format' => 'pdb');
# note: we quote -format to keep older perl's from
    complaining.
while ( my $struc = $in->next_structure() ) {
    $out->write_structure($struc);
    print "Structure ",$struc->id," number of models: ",
          scalar $struc->model,"\n";
}
```

Sequence Features

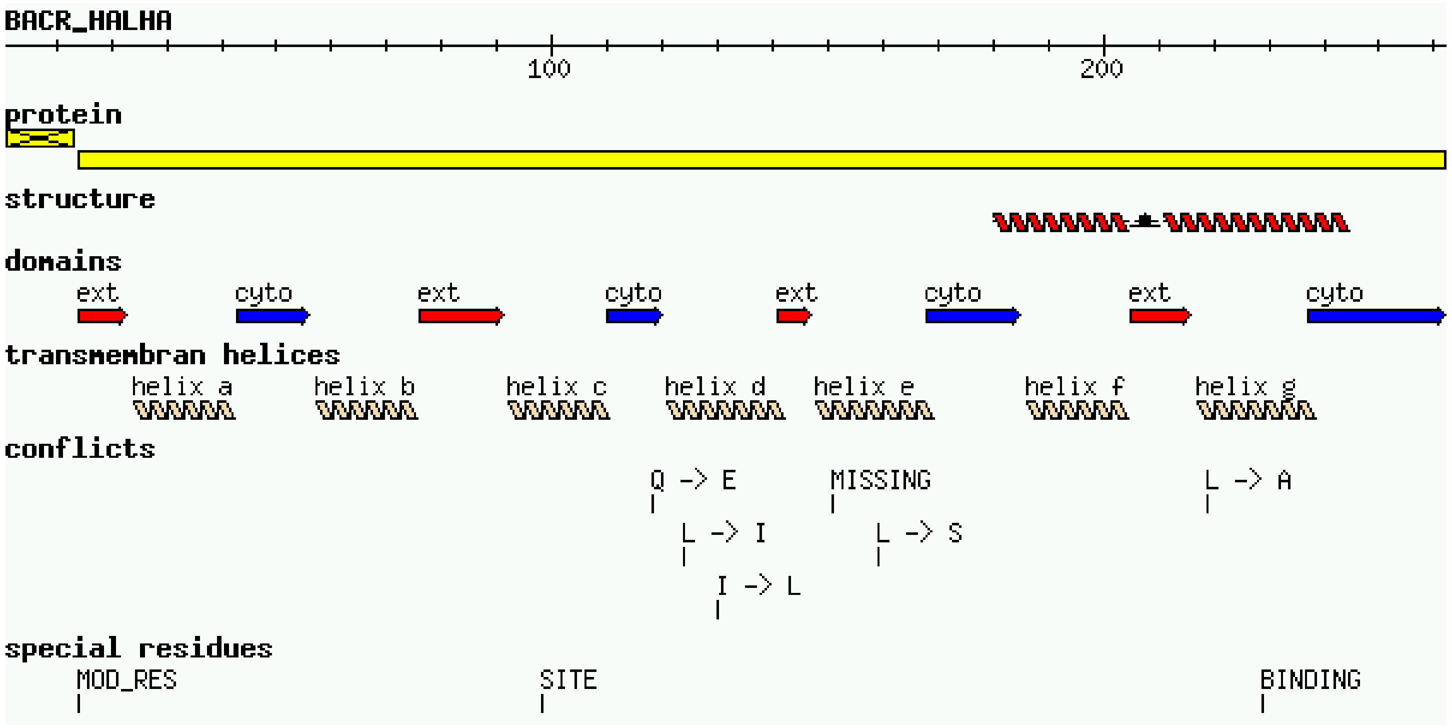


```
FT   CDS       join(AB000411.1:596..759,AB000414.1:13..272,
FT             AB000415.1:13..161,AB000416.1:13..120,AB000417.1:13..115,
FT             AB000418.1:13..173,AB000419.1:13..148,AB000420.1:13..379,
FT             AB000421.1:13..214,AB000422.1:6..192,AB000423.1:13..141,
FT             AB000424.1:13..149,13..147)
FT             /codon_start  =  1
FT             /db_xref      =  "SPTREMBL:P79433"
FT             /product      =  "endopeptidase 24.16 type M2"
FT             /protein_id   =  "BAA19105.1"
FT             /translation  =  "MVYPEGHLARELGATFSSSAPLGGHPFPFVWDCLSCKQGDWSQAR
FT             PKTNAERRSGVGGSGILLRMTLGREAMSPLQAMSSYTVDGRNVLRWDLSPEQIKRRTEE
FT             LIAQTKQVYDDIGMLDIEEVTYENCLQALADVEVKYIVERTMLDFPQHVSSDKEVRAAS
FT             TEADKRLSRFDIEMSMREDIFLRIVRLKETCDLGKIKPEARRYLEKSVKMGKRNGLHLP
FT             EQVQNEIKAMKKRMSELCIDFNKNLNEDDTFLVFSKAELGALPDDFIDSLEKTDDNKYK
FT             ITLKYPHYFPVMKKCCIPETRRKMEMAFNTRCKEENTIILQELLPLRAKVAKLLGYSTH
FT             ADFVLEMNTAKSTHHVTAFLDDLSQKLKPLGEAEREFILNLKKKECEEKGFEYDGKINA
FT             WDLHYYMTQTEELKYSVDQEILKEYFPIEVVTEGLLNIYQELLGLSFEQVTDAHVWNKS
FT             VTLYTVKDKATGEVLGQFYLDLYPREGKYNHAACFGLQPGCLLPDGSRMMSVAALVVNF
FT             SQPRAGRPSLLRHDEVRTYFHEFGHVMHQICAQTDFARFSGTNVETDFVEVPSQMLENW
FT             VWDTDSLRRLSKHYKDGSPITDDLLEKLVASRLVNTGLLTLRQIVLSKVDQSLHTNTSL
FT             DAASEYAKYCTEILGVAATPGTNMPATFGHLAGGYDGQYYGYLWSEVFSMDMFYSCFKK
FT             EGIMNPEVGMKYRNLILKPGGSLDGMDMLQNFLKREPNQKAFLMSRGLHAP"
```

BioPerl: Seq and SeqIO

```
use Bio::SeqIO;
$seqin = Bio::SeqIO->new(-format =>'EMBL', -file=>‘f1');
$seqout= Bio::SeqIO->new(-format =>'Fasta',-file=>'>f1.fa');
while((my $seqobj = $seqin->next_seq())) {
    print "Seq: ", $seqobj->display_id, ", Start of seq ",
            substr($seqobj->seq,1,10),"\n";
    if( $seqobj->moltype eq 'dna') {
            $rev = $seqobj->revcom;
            $id = $seqobj->display_id();
            $id = "$id.rev";
            $rev->display_id($id);
            $seqout->write_seq($rev); } #end if
    foreach $feat ( $seqobj->top_SeqFeatures() ) {
            if( $feat->primary_tag eq 'exon' ) {
               print STDOUT "Location ",$feat->start,":",
               $feat->end," GFF[",$feat->gff_string,"]\n";}
    } # end foreach
} # end while
exit; #bioperl6.pl
```

BioPerl Graphics Objects



textx2.pl can create such a graphics object from a SWISS-PROT file.

BioPerl Sequence Analysis Tools

```
$seq_stats = Bio::Tools::SeqStats->new(-seq=>$seqobj);
$seq_stats->count_monomers();
$seq_stats->count_codons();
$weight = $seq_stats->get_mol_wt($seqobj);

$pat = 'T[GA]AA...TAAT';
$pattern = new Bio::Tools::SeqPattern(-SEQ =>$pat, -TYPE
    =>'Dna');
$pattern->expand;
$pattern->revcom;
$pattern->alphabet_ok;
```

BioPerl Restriction Enzymes

- Locating restriction enzyme cutting sites:
  - RestrictionEnzyme object ;
  - data for over 150 restriction enzymes built in.
  - Access list of available enzymes using available_list()
- Restriction sites can be obtained by cut_seq().
- Adding an enzyme not in the default list is easy.

Restriction Enzymes example

```
#!/local/bin/perl -w

$re=new Bio::Tools::RestrictionEnzyme('-name'=>'EcoRI');
@sixcutters = $re->available_list(6);

$re1 = new Bio::Tools::RestrictionEnzyme(-name=>'EcoRI');
# $seqobj is the Seq object for the dna to be cut
@fragments = $re1->cut_seq($seqobj);

$re2 = new Bio::Tools::RestrictionEnzyme('-NAME' =>'EcoRV--
    GAT^ATC', '-MAKE' =>'custom');

exit;
```

Alignment Object

```
#! /local/bin/perl -w
use strict;
use Bio::AlignIO;
my $inform = shift @ARGV || 'clustalw';
my $outform = shift @ARGV || 'fasta';
my $in = Bio::AlignIO->newFh ( -fh => \*STDIN,
    -format => $inform );
my $out = Bio::AlignIO->newFh ( -fh => \*STDOUT, -
    format => $outform );

print $out $_ while <$in>;
exit;
```

Alignment Object

```
#! /local/bin/perl -w
use strict;
use Bio::AlignIO;
my $in = new Bio::AlignIO ( -file =>, $ARGV[0], -format => 'clustalw'
    );
my $aln = $in->next_aln();
print " all seqs same length: ",($aln->is_flush()) ? "yes" : "no", "\n";
print "alignment length: ", $aln->length(), "\n";
printf "identity: %.2f %%\n", $aln->percentage_identity();
printf "identity of conserved columns: %.2f %%\n",
    $aln->overall_percentage_identity();
```

BioPerl: Pairwise Sequence Alignment

```
use Bio::Tools::pSW;

$factory = new Bio::Tools::pSW( '-matrix'
   => 'blosum62.bla', '-gap' => 12, '-ext' =>
   2, );

$factory->align_and_show($seq1, $seq2,
   STDOUT);
```

BioPerl: Running BLAST

```
# This program only shows how to invoke BLAST and store the result
use Bio::SeqIO;
use Bio::Tools::Run::RemoteBlast;
my $Seq_in = Bio::SeqIO->new (-file => $ARGV[0], -format => 'fasta');
my $query = $Seq_in->next_seq();
my $factory = Bio::Tools::Run::RemoteBlast->new( '-prog' => 'blastp',
    '-data' => 'swissprot', _READMETHOD => "Blast" );
my $blast_report = $factory->submit_blast($query);
my $result = $blast_report->next_result;
while( my $hit = $result->next_hit()) {
    print "\thit name: ",
    $hit->name(), " significance: ", $hit->significance(), "\n";
}
# There are programs on the bioperl website that can help you automatically
    parse the information returned by BLAST.
```

BioPerl: Multiple Sequence Alignment

```
@params = ('ktuple' => 2, 'matrix' => 'BLOSUM');
$factory =
Bio::Tools::Run::Alignment::Clustalw->new(@params);
$aln = $factory->align(\@seq_array);


foreach $seq ( $aln->eachSeq() )
   {
        print $seq->seq(), "\n";
}
```

BioPerl: Structure

- Ability to store and manipulate structures.
- Modules: Atom, Chain, Residue, Model, Entry, IO
- Atom
  - new, x, y, z, xyz, residue, element,
- Chain, Residue
- Entry
  - Add_model, chain, add_chain, residue, add_residue, get_residue, add_atom, get_atoms, conect, get_atom_by_serial, seqres, …
- Model