eHive

–

Ensembl Compara Production system

Javier Herrero

Vertebrate Genomics Team

EMBL-EBI

Wellcome Trust Genome Campus

Hinxton CB10 1SD, UK

EMBL-EBI

wellcome trust sanger institute

Production System Requiements

Flexible

Fault-tolerant

Minimal supervision

High-throughput

eHive system overview

1 Database with information about

- jobs to be run
- rules about which jobs cannot be run before others
- rules about flowing the output of one job to another job

BlackBoard

Dataflow graph

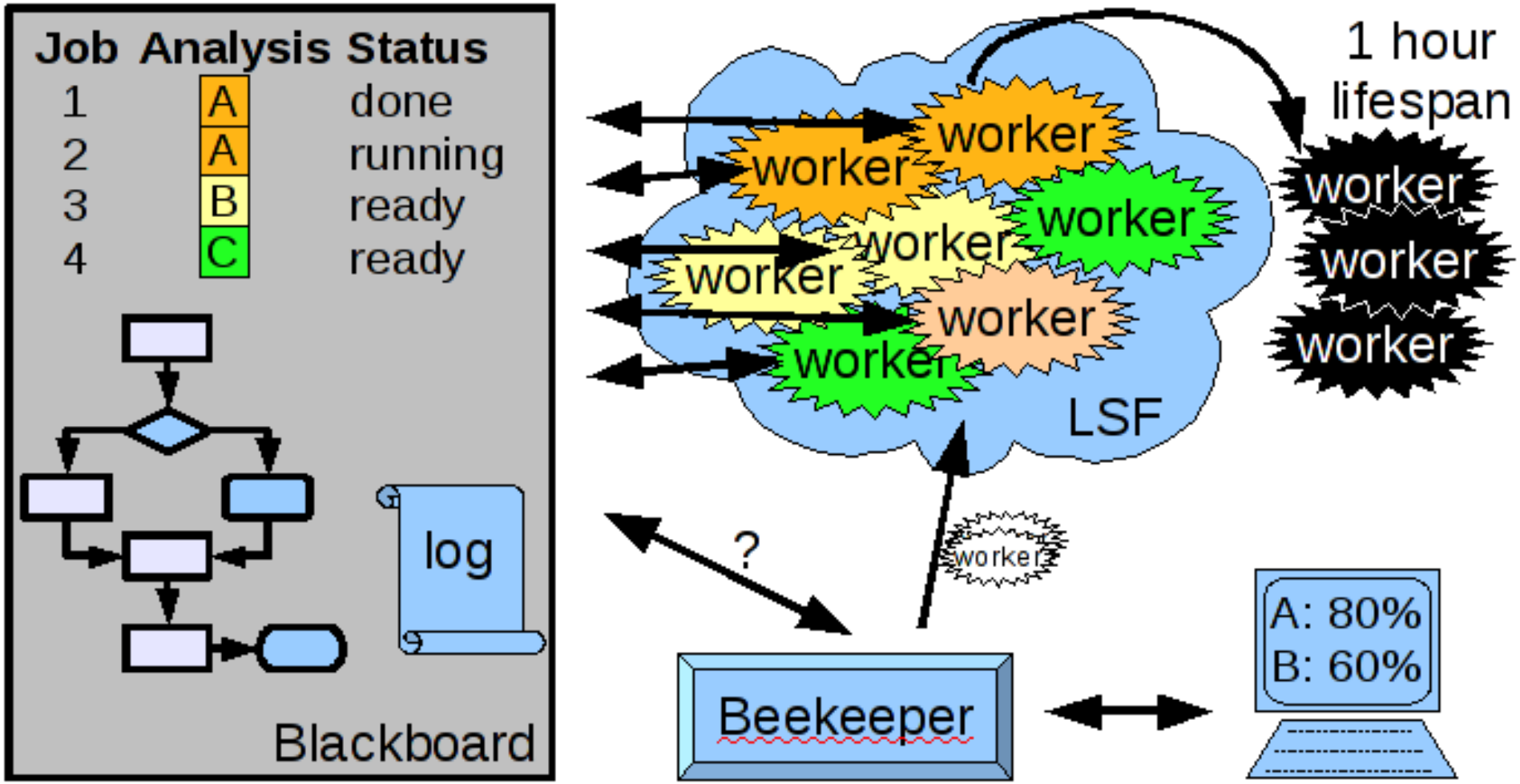
2 scripts

- runWorker.pl: takes jobs from the eHive DB; runs them; stores and/or flows the output
- beekeeper.pl: monitors the eHive DB and sends workers to the farm using LSF queuing system

Autonomous agent

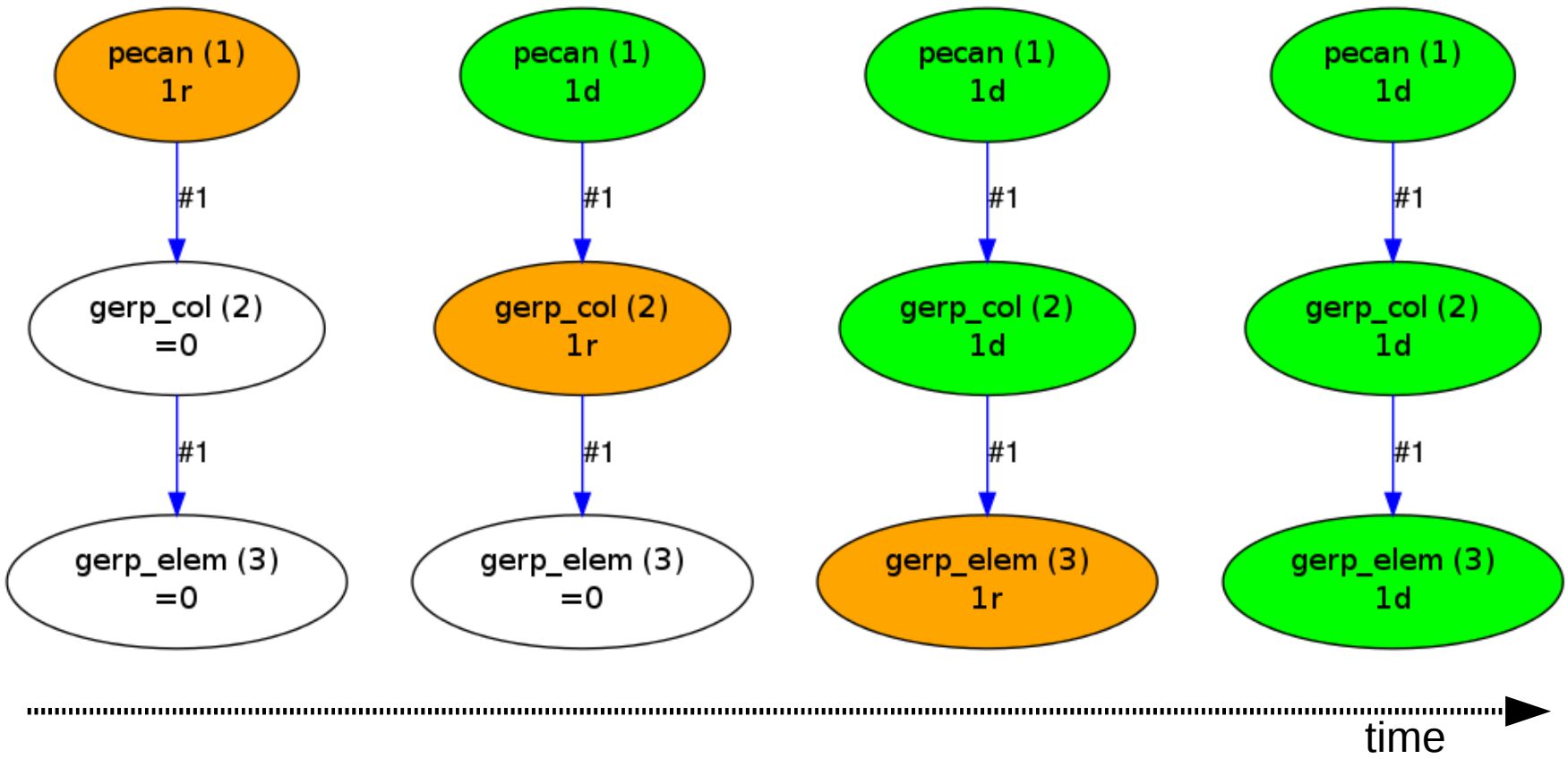
Central contoller

eHive overview



Pipeline design

A pipeline is usually defined a graph, with one job at the top that triggers new jobs when it completes.



Configuration module

The graph is defined in a configuration module. The easiest way to create your configuration module is to copy an existing one (ensembl-hive/modules/Bio/EnsEMBL/Hive/PipeConfig/).

It typically requires two sections at least:

- **sub default_options { … }**: defines defaults for this pipeline
- **sub pipeline_analyses { … }**: defines the analyses and how they are connected (the graph)

Pecan-GERP mini-pipeline

Let's design a simple pipeline to run:

- Pecan
- gerp_col
- gerp_elem

The output of *Pecan* will be used as the input for *gerp_col* and the output of *gerp_col* will be the input of *gerp_elem*.

Pecan-GERP mini-pipeline

The default_options section:

```
sub default_options {
    my ($self) = @_;
    return {
        %{ $self->SUPER::default_options() },

        'pipeline_name' => 'mini_pecan_single',
    };
}
```

Returns a hash

Inherits other defaults from the base class

The name of the pipeline. You can be creative

You could also specify other values that can be used in the pipeline

Pecan-GERP mini-pipeline

The pipeline_analyses section:

```
sub pipeline_analyses {
    my ($self) = @_;
    return [

        {   -logic_name => 'pecan',
            -module     => 'Bio::EnsEMBL::Hive::RunnableDB::SystemCmd',
            -parameters => {
                'cmd'     => 'your command goes here',
            },

        },
    ]
}
```

We need a unique name for each analysis

The SystemCmd module allows us to run command lines. There are other modules and you can create your owns.

The SystemCmd needs the parameter *cmd* which defines the command line

Pecan-GERP mini-pipeline

Fixed command:

```
{   -logic_name => 'pecan',
    -module     => 'Bio::EnsEMBL::Hive::RunnableDB::SystemCmd',
    -parameters => {
        'cmd'     => 'java -cp pecan_v0.8.jar bp.pecan.Pecan
                       -E "(HUMAN,(MOUSE,RAT));"
                       -F human.fa mouse.fa rat.fa -G pecan.mfa',
    },
},
```

This is not very convenient as it will always run exactly the same command

Pecan-GERP mini-pipeline

Parameter substitution:

```
{   -logic_name => 'pecan',
    -module     => 'Bio::EnsEMBL::Hive::RunnableDB::SystemCmd',
    -parameters => {
        'cmd'     => 'java -cp pecan_v0.8.jar bp.pecan.Pecan -E
                        "#tree_string#" -F #input_files# -G #msa_file#',
    },
    -input_ids => [
        {
            'tree_string' => '(HUMAN,(MOUSE,RAT));',
            'input_files'  => 'human.fa mouse.fa rat.fa',
            'msa_file'     => 'pecan.mfa',
        }
    ],
},
```

The parameters are the same for all the jobs of a given analysis.

Each set of input_ids will be a job. We can have several ones, But they need to be unique.

Note: Typically, each analysis gets its input from the previous one in the pipeline.

Pecan-GERP mini-pipeline

gerp_col and gerp_elem analyses:

```
{   -logic_name => 'gerp_col',
    -module     => 'Bio::EnsEMBL::Hive::RunnableDB::SystemCmd',
    -parameters => {
        'cmd'         => 'gerpcol -t tree.nw -f #msa_file# -a -e HUMAN',
    },
},

{   -logic_name => 'gerp_elem',
    -module     => 'Bio::EnsEMBL::Hive::RunnableDB::SystemCmd',
    -parameters => {
        'cmd'   => 'gerpelem -f #msa_file#.rates -c chr13 -s 32878016 -x .bed',
    },
},
```

Now we have all the analyses, but we haven't defined how they are connected

We will use dataflow rules for connecting the steps.

Pecan-GERP mini-pipeline

Dataflow rules

```
{   -logic_name => 'pecan',
    [[...snip...]]
    -flow_into => {
        1 => [ 'gerp_col' ],
    },
},

{   -logic_name => 'gerp_col',
    [[...snip...]]
    -flow_into => {
        1 => [ 'gerp_elem' ],
    },
},

{   -logic_name => 'gerp_elem',
    [[...snip...]]
},
```

Pecan jobs will flow into gerp_col

gerp_col jobs will flow into gerp_elem

See MiniPecanSingle_conf.pm

Running an eHive pipeline

You need to complete four steps:

- Write/Modify your configuration module (this is required only the 1st time)
- Configure your environment (could be saved in your .bashrc file)
  - export PATH=$PATH:~/src/ensembl-hive/scripts
  - export ENSEMBL_CVS_ROOT_DIR=~/src/
- Initialise your pipeline
  - init_pipeline.pl MiniPecanSingle_conf.pm -hive_driver sqlite -password FOO
- Run beekeeper.pl
  - beekeeper.pl -url sqlite:///username_mini_pecan_single -loop

Advanced dataflow: input_template

This allows you to rename the parameters between analyses:

```
{   -logic_name => 'pecan',
     [[...snip...]]
    -flow_into => {
        1 => { 'gerp_col' => {'input_file' => '#msa_file#'} },
    },
},

{   -logic_name => 'gerp_col',
    -module     => 'Bio::EnsEMBL::Hive::RunnableDB::SystemCmd',
    -parameters => {
        'cmd'         => 'gerpcol -t tree.nw -f #input_file# -a -e HUMAN',
    },
     [[...snip...]]
},
```

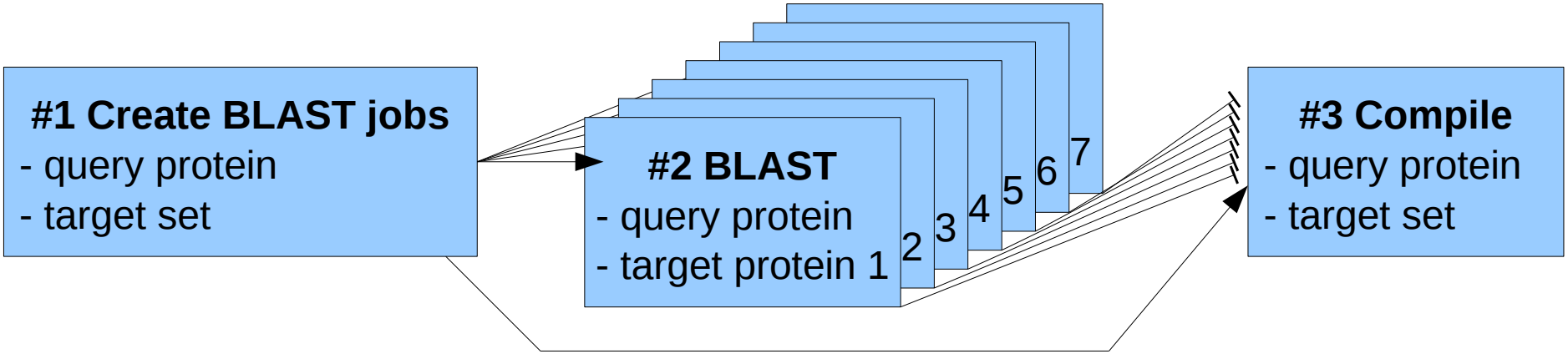
See MiniPecanSingle2_conf.pm

Advanced dataflow: semaphores

Semaphores create control rules at the job level. A given job cannot start until some jobs are done.

This is typically used for fans and funnels, where a job creates several ones and you need a job to collect all the information at the end.

- Imagine you want to find the most similar protein in a large set:



Simple semaphore

```
{   -logic_name => 'pecan',
    [[...snip...]]
    -input_ids  => [
        {
          'tree_string' => '((((HUMAN,(MOUSE,RAT)),COW),OPOSSUM),CHICKEN);',
          'input_files' => 'human.fa mouse.fa rat.fa cow.fa opossum.fa chicken.fa',
          'msa_file'    => "pecan3.mfa",
          'chr_name'    => "chr13",
          'chr_start'   => "32878016",
        },
    ],
    -flow_into => {
        '1->A' => { 'gerp_col' => {'input_file' => '#msa_file#'} },
        'A->1' => { 'gerp_elem' => {'input_file' => '#msa_file#.rates',
                                    'chr_name' => '#chr_name#',
                                    'chr_start' => '#chr_start#'}
                                  },
    },
},
```

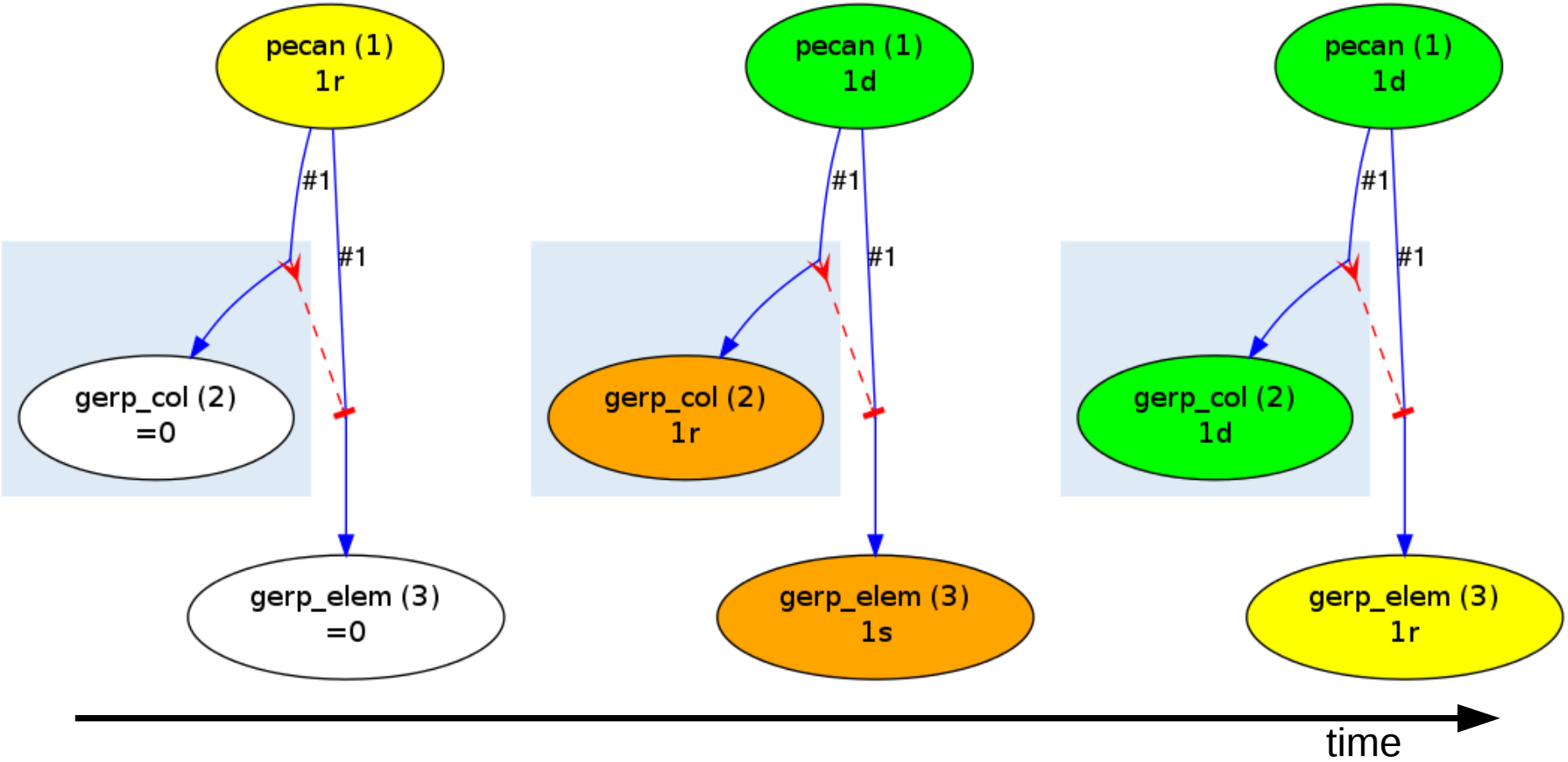
1->A: jobs flow through branch 1. These jobs will update the semaphore 'A' upon completion

A->1: jobs flow through branch 1. These jobs will wait for the corresponding gerp_col job.

See MiniPecanSingle3_conf.pm

Simple semaphore timeline

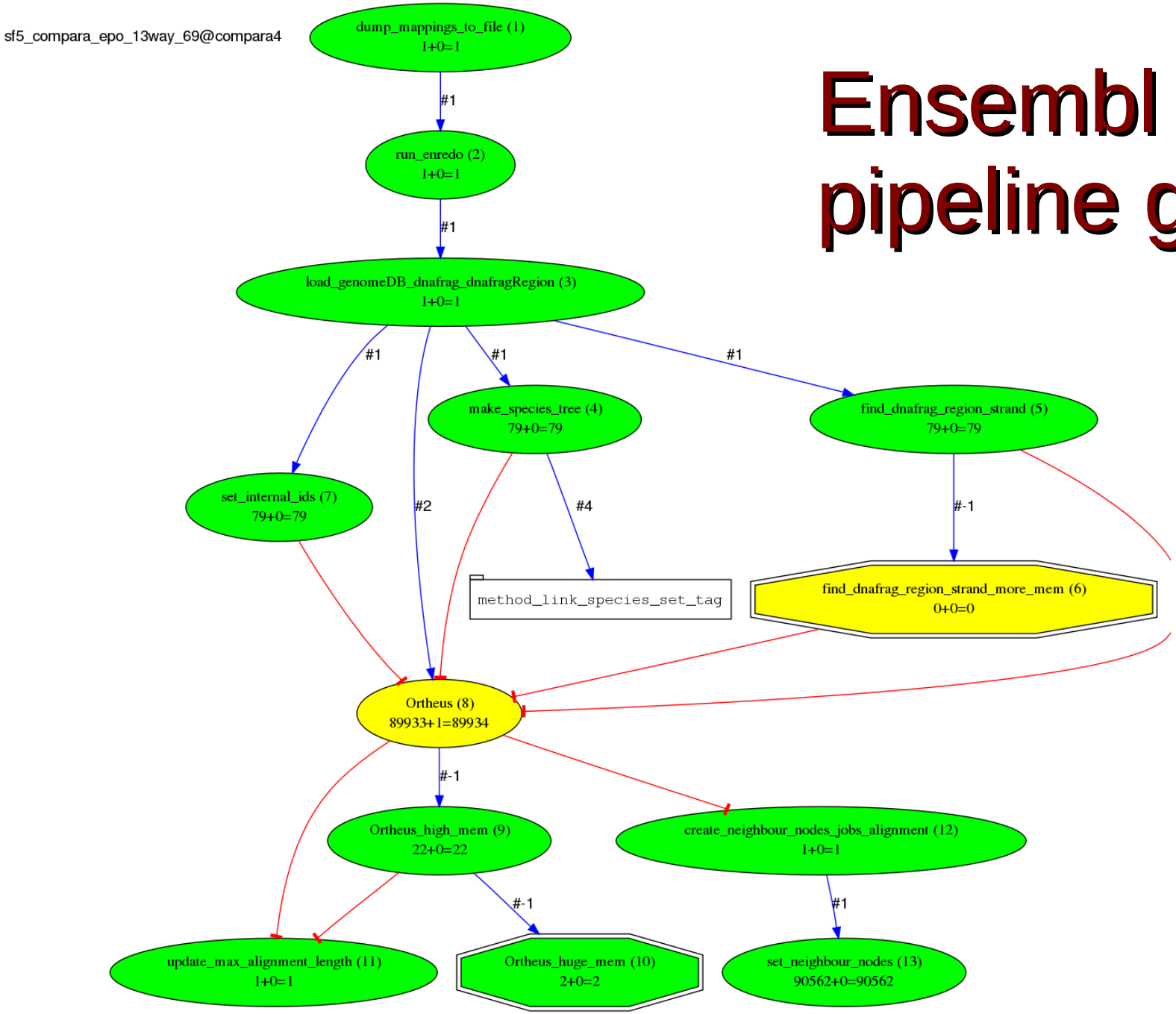
The ***pecan*** job create both the ***gerp_col*** and ***gerp_elem*** jobs, however the ***gerp_elem*** job is semaphored (s) and waits for its ***gerp_col*** job to finish.



A couple of examples of pipelines used in Ensembl Compara

These graphs represent real pipelines and show how you can build complex pipelines by combining dataflow rules, control rules and semaphores.

Ensembl EPO pipeline graph



Ensembl Family pipeline graph