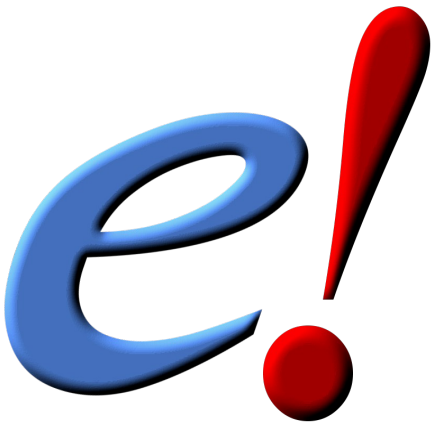
eHive Workshop

part 2: How to create pipelines (configuration files)

Leo Gordon

EMBL-EBI

wellcome trust sanger institute



Before we begin:

- Please make sure you have correct setup from part 1:
  - You are running *screen -RD* on one of farm login nodes
  - ensembl (core) and ensembl-hive repositories are installed and up-to-date:

    $ cvs -q update -dP
  - $ENSEMBL_CVS_ROOT_DIR points to the directory holding the checkouts
  - Slides are here:
    $ENSEMBL_CVS_ROOT_DIR/ensembl-hive/docs/presentations/HiveWorkshop_Sept2013
  - $PERL5LIB includes ensembl/modules and ensembl-hive/modules
  - $PATH includes ensembl-hive/scripts
  - You have a username+password on our training MySQL server

    (set them in $EHIVE_* environment variables for the duration of the course)

- Do we need breaks?

Modularity of pipelines. PipeConfigs & Runnables

- A Hive pipeline is defined in a PipeConfig file which references one or more Runnable modules.

- Many tasks can be solved by using “universal” Runnables provided by the Hive (SystemCmd, SqlCmd, JobFactory, Dummy), but sometimes you have to write your own application-specific Runnables.

- We shall learn to use universal Runnables before making our own
(however it may be the opposite of what you do in practice)

- Hive’s “universal” Runnables live here:

  $ENSEMBL_CVS_ROOT_DIR/ensembl-hive/modules/Bio/EnsEMBL/Hive/RunnableDB/

- Hive’s PipeConfig files live here:

  $ENSEMBL_CVS_ROOT_DIR/ensembl-hive/modules/Bio/EnsEMBL/Hive/PipeConfig/

They are written in a subset of Perl.

The simplest pipeline : AnyCommands_conf.pm

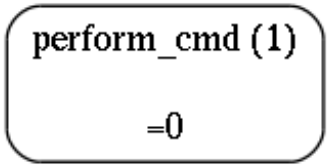
- Open the file - this is the smallest PipeConfig possible:

```
use base ('Bio::EnsEMBL::Hive::PipeConfig::HiveGeneric_conf');      #  or subclass

sub pipeline_analyses {
    return [
        {   -logic_name    => 'perform_cmd',
            -module        => 'Bio::EnsEMBL::Hive::RunnableDB::SystemCmd',
        },
    ];
}
```

```
$ init_pipeline.pl Bio::EnsEMBL::Hive::PipeConfig::AnyCommands_conf
```

```
$ generate_graph.pl -url $EHIVE_URL -out any_c_empty.png
```

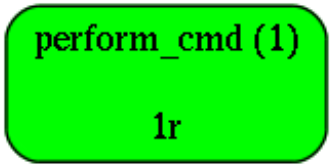


or open http://guihive.internal.sanger.ac.uk:8080/ instead

Seeding and running AnyCommands_conf.pm

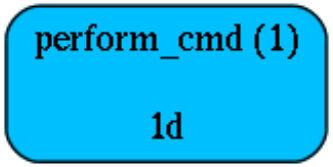
- No jobs - we will have to seed them:

```
$ seed_pipeline.pl -url $EHIVE_URL -logic_name perform_cmd \
    -input_id '{"cmd" => "echo Hello, world"}'
```



- and run:

```
$ runWorker.pl -url $EHIVE_URL
```



- Practical to a certain extent (analysis_capacity, batch_size, resources)

Analysis-wide parameters and substitution

- We can define values for old parameters and create new ones:

```
sub pipeline_analyses {
    return [
        {   -logic_name     => 'perform_cmd',
            -module         => 'Bio::EnsEMBL::Hive::RunnableDB::SystemCmd',
            -parameters => {
                "cmd" => "gzip #filename# ",
            },
        },
    ];
}
```

"cmd" defined on the analysis level

created a new parameter that can be defined by jobs

- Exercise:
  seed a few jobs and run them (you can copy some compressible files from ~lg4/work/pdfs)

- Automated seeding of jobs?

```
for filename in `find pdfs/ -name '*.pdf'` ; do
    seed_pipeline.pl -url $EHIVE_URL \
        -logic_name perform_cmd -input_id "{ 'filename' => '$filename' }";
done
```

Factories and dataflow : CompressFiles_conf.pm

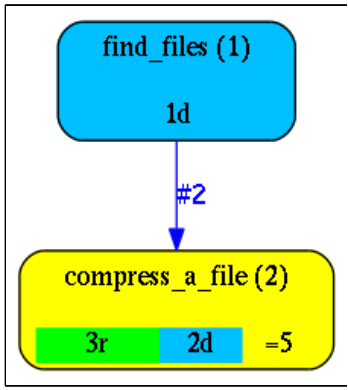
- Dataflow Rules can be used to make Jobs seed other Jobs
- Factory is an Analysis whose only aim is to seed other Jobs, create a “fan”, turn time into space
- Higher level input

```
sub pipeline_analyses {
    return [
        {   -logic_name => 'find_files',
            -module     => 'Bio::EnsEMBL::Hive::RunnableDB::JobFactory',
            -parameters => {
                'inputcmd'     => 'find #directory# -type f ',
                'column_names' => [ 'filename' ],
            },
            -flow_into => {
                2 => [ 'compress_a_file' ],
            },
        },
        {   -logic_name    => 'compress_a_file',
            -module        => 'Bio::EnsEMBL::Hive::RunnableDB::SystemCmd',
            -parameters    => {
                'cmd'       => 'gzip #filename#',
            },
            -analysis_capacity => 4,
        },
    ];
}
```

created a new higher level parameter

what to call the output

where the output should go



Dataflow conventions

- Each Dataflow Event is a pair (branch_number, hash_of_parameters+).
- In our example the 'find_files' job that we seed with

```
{ 'directory' => 'pdfs' }
```

generates the following Dataflow Events:

```
#2, { 'filename' => 'pdfs/first.pdf' }
#2, { 'filename' => 'pdfs/second.pdf' }
. . .
#2, { 'filename' => 'pdfs/last.pdf' }
#1, { 'directory' => 'pdfs' }
```

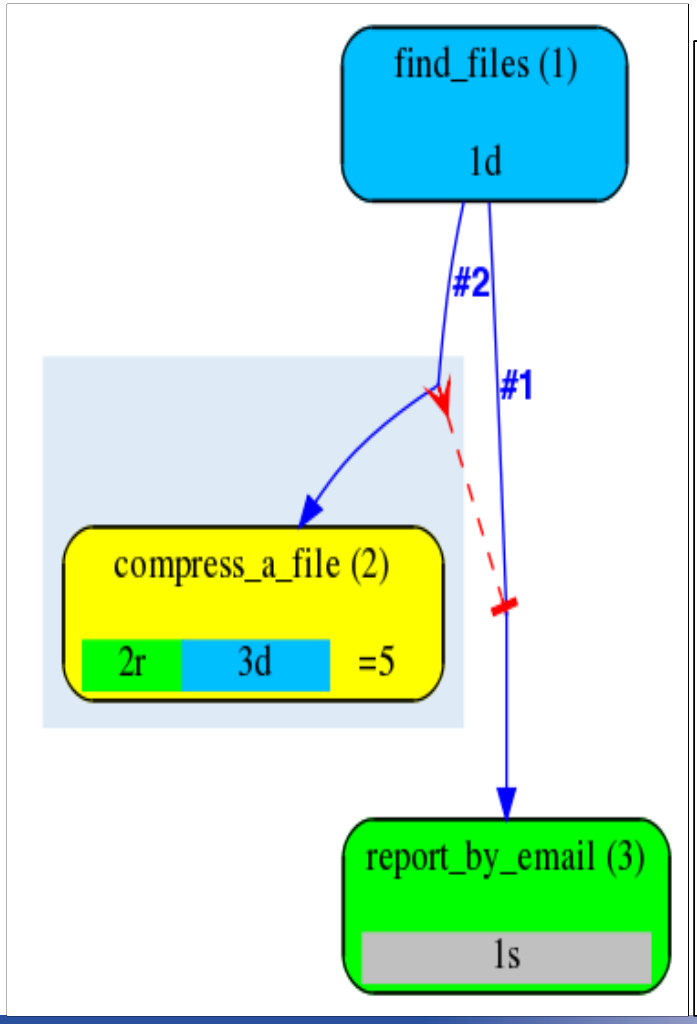
The "fan"

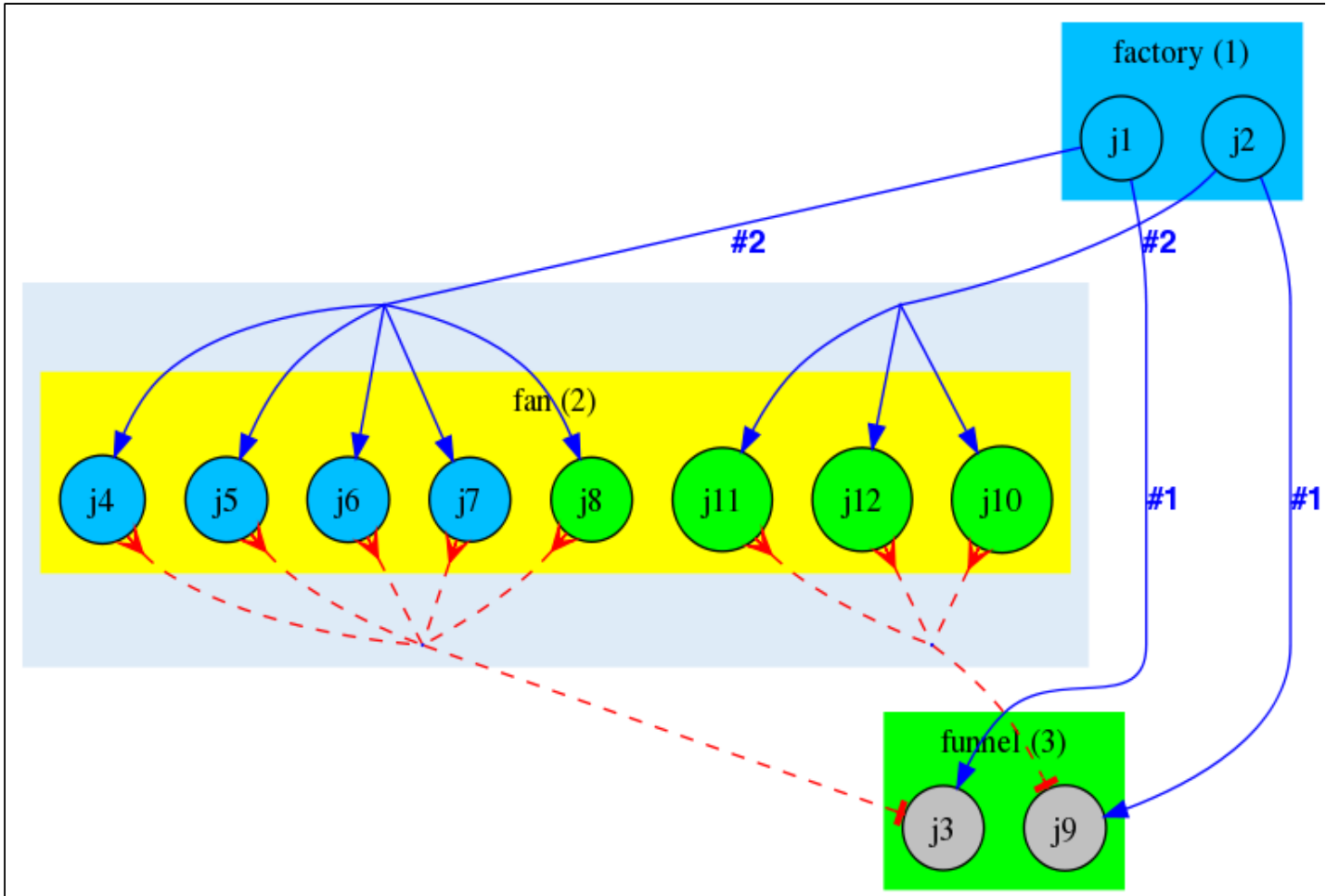
"autoflow" event,
helps to bind analyses consecutively

- "Reserved" branch numbers that have their own meaning (similar to UNIX file descriptors):
  - 1 is almost always present, it is the "continuation" after Job is 'DONE'
  - 2 is used by many Factory Runnables to emit the "fan" (of Jobs, etc)
  - -1 : "postmortem dataflow after MEMLIMIT" on LSF
  - -2 : "postmortem dataflow after RUNLIMIT" on LSF
  - 3, 4, 5... : unreserved, can be used for anything
- Each Runnable has its own set of branch_numbers that it may emit Dataflow Events into.
  Check its documentation or code to make sure the dataflow you are wiring is live.
  What happens if it's not?
- How to regain the single thread of control?

Regaining single thread of control: semaphored dataflow

- Built-in mechanism for converging individual threads back together.
- Based on semaphores that can block an individual job by a set of prerequisite jobs.





Semaphored dataflow in a PipeConfig

- Creating a funnel Analysis : let the pipeline send us a notification:

```
{       -logic_name    => 'report_by_email',
        -module        => 'Bio::EnsEMBL::Hive::RunnableDB::NotifyByEmail',
        -meadow_type => 'LOCAL',  # NB: farm nodes may not support sendmail
        -parameters    => {
            'email'   => $ENV{'USER'} . '@sanger.ac.uk',    # what if it's wrong?
            'subject' => 'pipeline has finished',
            'text'    => 'done compressing files in #directory#',
        },
},
```

- Linking two rules together happens in the emitting Analysis:

```
-flow_into => {
    '2->A' => [ 'compress_a_file' ],
    'A->1' => [ 'report_by_email' ],
}
```

creates a group of jobs that control a semaphore

creates one job that is controlled by the semaphore

- Try running.
  (solution: CompressFiles2_conf)
- A break?

Parameters and their implicit propagation

- Exercise 1:
  How do we set a default directory name for analysis 1?

- Exercise 2:
  Introduce another parameter `'only_files'` to define the wildcard pattern for filenames we want to compress

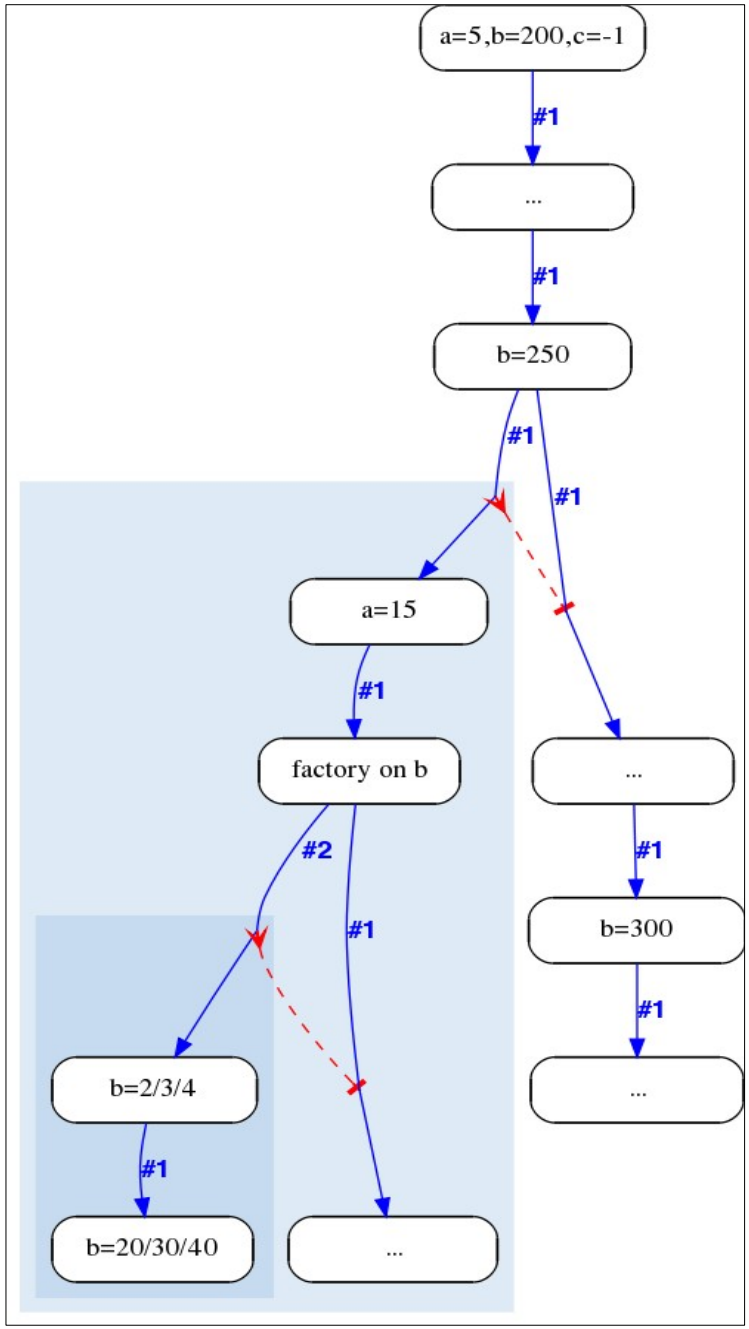
- What if we wanted to pass something to analysis 2 directly?
  Say, if we want the same analysis to act as decompressor:
  `'cmd' => 'gzip #gzip_flags# #filename#'`
  and then set `'gzip_flags'` to `'-d'` somehow/somewhere?

- First, pass the parameter to the job being seeded, then propagate it

- Explicit propagation using templates vs implicit propagation

How implicit parameter propagation works

```
{
    my $a=5; my $b=200 ;my $c=-1;
    …
    $b=250 ;
    …
        {
            my $a=15;
            for my $b (2,3,4) {
                $b *= 10;
            }
            …
        }
    …
    $b=300;
}
```



Using implicit parameter propagation

- What if we wanted to pass something to analysis 2 directly?
  Say, if we want the same analysis to act as decompressor:

```
'cmd'        => 'gzip #gzip_flags# #filename#'
```

- Implicit parameter propagation mechanism is off by default, switch it on using:

```
sub hive_meta_table {

    my ($self) = @_;
    return {
        %{$self->SUPER::hive_meta_table},
        'hive_use_param_stack'  => 1,
    };
}
```

- Can you now propagate 'gzip_flags' to analysis 2?
  (solution: CompressFiles3_conf)

Capturing data : another role of JobFactory

- Both SystemCmd and SqlCmd only run your command, no output is captured in any structured way.
  - So SqlCmd is usually used to INSERT, DELETE, UPDATE, CREATE/ALTER/DROP TABLE, but not with SELECT.

- JobFactory is not specifically creating Jobs - it simply transforms streams of “things” into Dataflow Events that may be converted into Jobs, stored in database tables, or accumulated. It is the wiring that defines what happens next.

```
{   -logic_name    => 'pre_compress_size',
    -module        => 'Bio::EnsEMBL::Hive::RunnableDB::JobFactory',
    -parameters    => {
        'inputcmd'      => "wc -c #filename# | sed -e 's/^ *//' ",
        'delimiter'     => ' ',
        'column_names'  => [ 'orig_size', 'orig_filename' ],
    },
},
```

- Insert and wire it correctly.
  - How to check whether we have captured anything?
  - How to pass it outside?

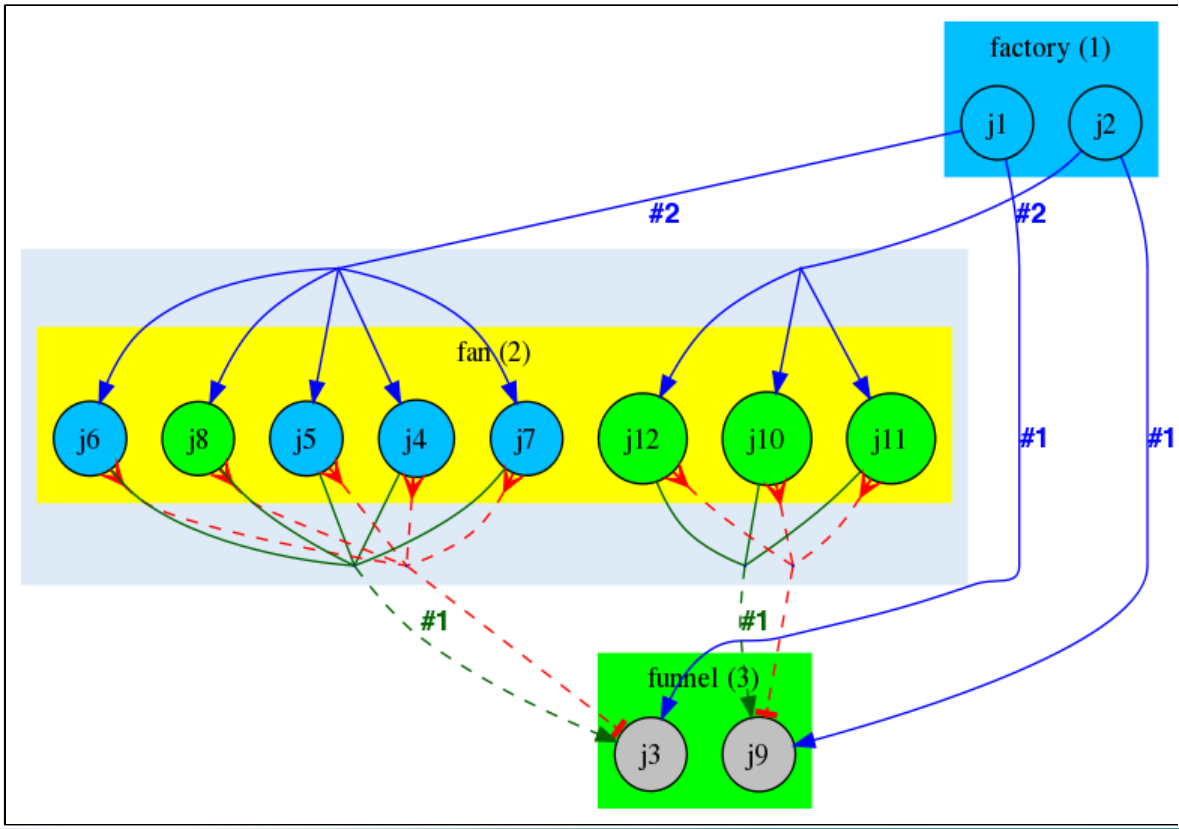
Accumulating data from a semaphore group

- How do we pass the data from the box into the funnel?
- The data can be passed from any job within the box into the correct funnel Job
- Different structures or combinations can be accumulated (hashes, arrays, piles, multisets)
- pseudo-Analysis names as targets for Dataflow (with or without templates).

- `':////accu?hash_name={key_name}'`
- `':////accu?array_name=[index_name]'`
- `':////accu?pile_name=[]'`
- `':////accu?multiset_name={}'`

- see LongMult_conf for example.

Flow the data into accu (which branch?).



Advanced parameter substitution : expressions

- What if we want to compute a value of #alpha#+1 rather than just a string?

```
'alpha_plus_one' => '#expr( #alpha#+1 )expr#'
```

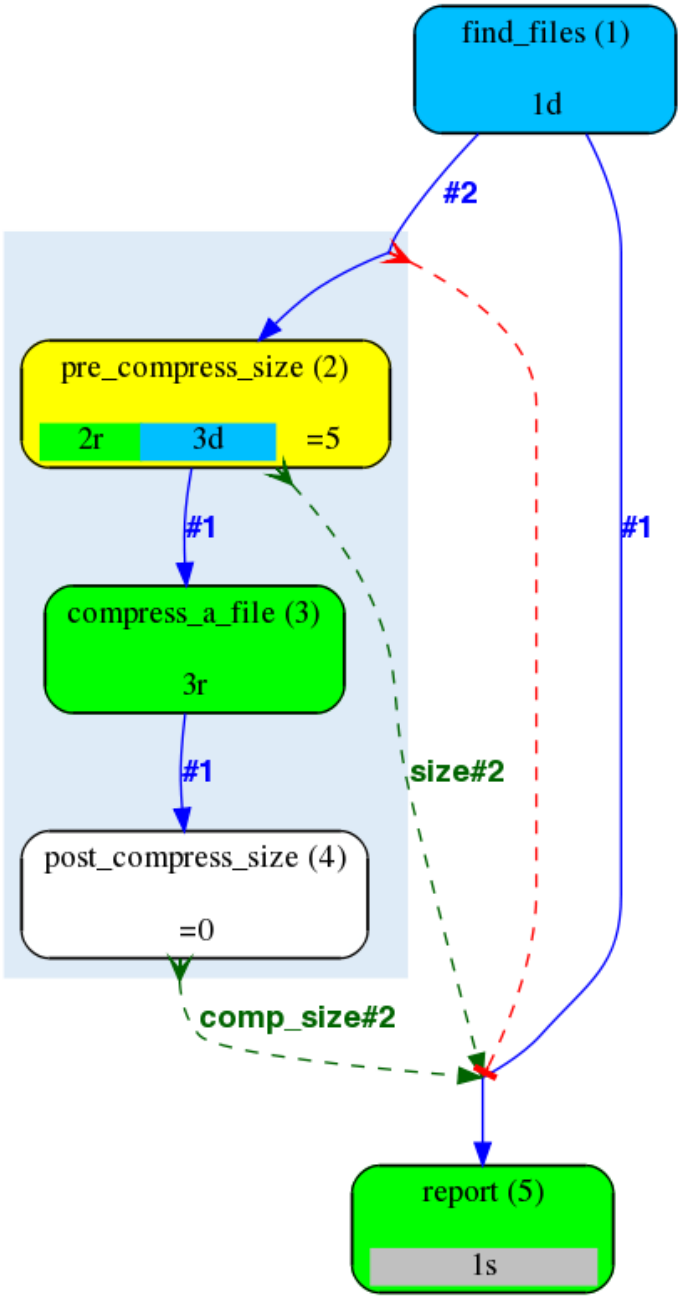
- Any Perl expression can be evaluated as follows:
  - first, #alpha# will be text-substituted with the value of alpha parameter
  - then the resulting string will be evaluated
  - put a space between dollar and the name ($ beta) if you want standard Perl interpretation of the variable
  - put curly braces around #alpha# if you want to dereference a reference:
    - @{ #array_ref# }
    - %{ #hash_ref# }

- We can flatten accumulated structures (that are not scalars) into scalars using #expr()expr# . For example,

```
'min_comp_size' => '#expr(min values %{#comp_size#})expr#',
'max_comp_size' => '#expr(max values %{#comp_size#})expr#',
'text'    => 'compressed sizes between #min_comp_size# and #max_comp_size#',
```

Exercise: accumulation + substitution

- Let's put it all together:
  - Factory on a directory to dataflow single filenames
  - compute their sizes and accumulate them
  - compress the files
  - compute the compressed sizes and accumulate
  - funnel flattens the accumulated structures and emails you the report
- Solution: CompressFiles4_conf



Questions?

Acknowledgements

Matthieu Muffato and Miguel Pignatelli

Current and previous members of Compara team

All users of eHive system for testing, feedback and ideas

Paul Flicek, Steve Searle and the entire Ensembl Team

Funding

wellcome trust

EMBL

National Human Genome Research Institute

BBSRC bioscience for the future

European Commission Framework Programme 7

SEVENTH FRAMEWORK PROGRAMME

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

GEN2PHEN

BLUEPRINT epigenome