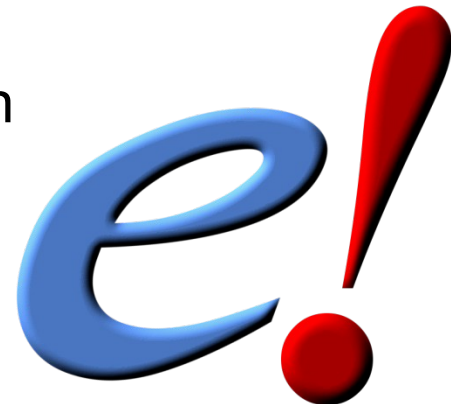




# Accumulated dataflow in eHive

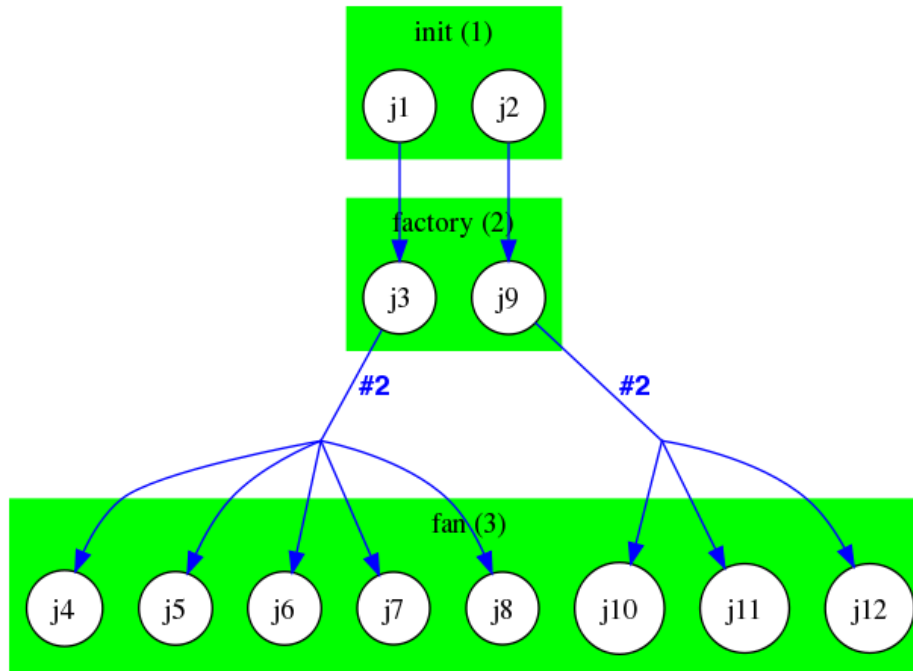
Leo Gordon



# Hive and dataflow

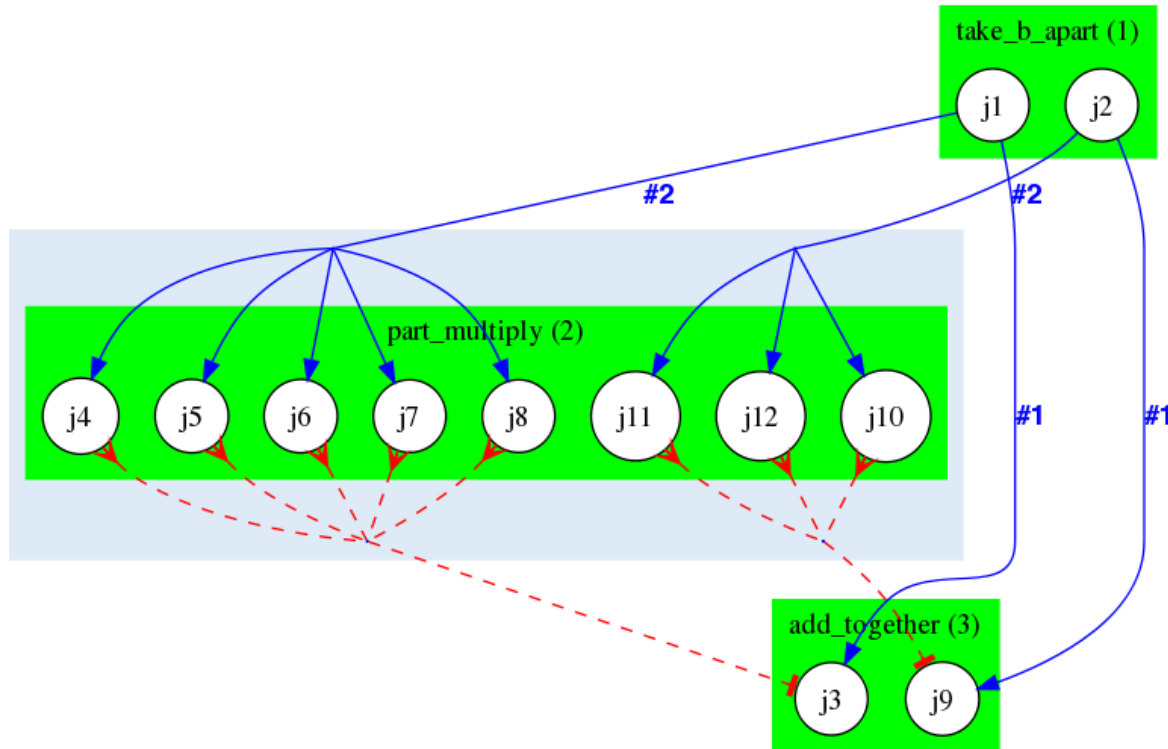
- *Hive is a DB-based system that simplifies creation of pipelines from configuration files and runs these pipelines on computer farms.*
- *Dataflow is a unified way sending parametrized events in Hive pipelines.*
- *These events are generated by jobs and can be optionally routed:*
  - *to analyses (which produces other jobs) or*
  - *to tables (which stores this data in tables).*

# Dataflow into analyses (produces jobs)



The logic of most pipelines follows that of structural programming: there is usually a single backbone/trunk of control that can spawn multiple "threads" whenever things can be done in parallel.

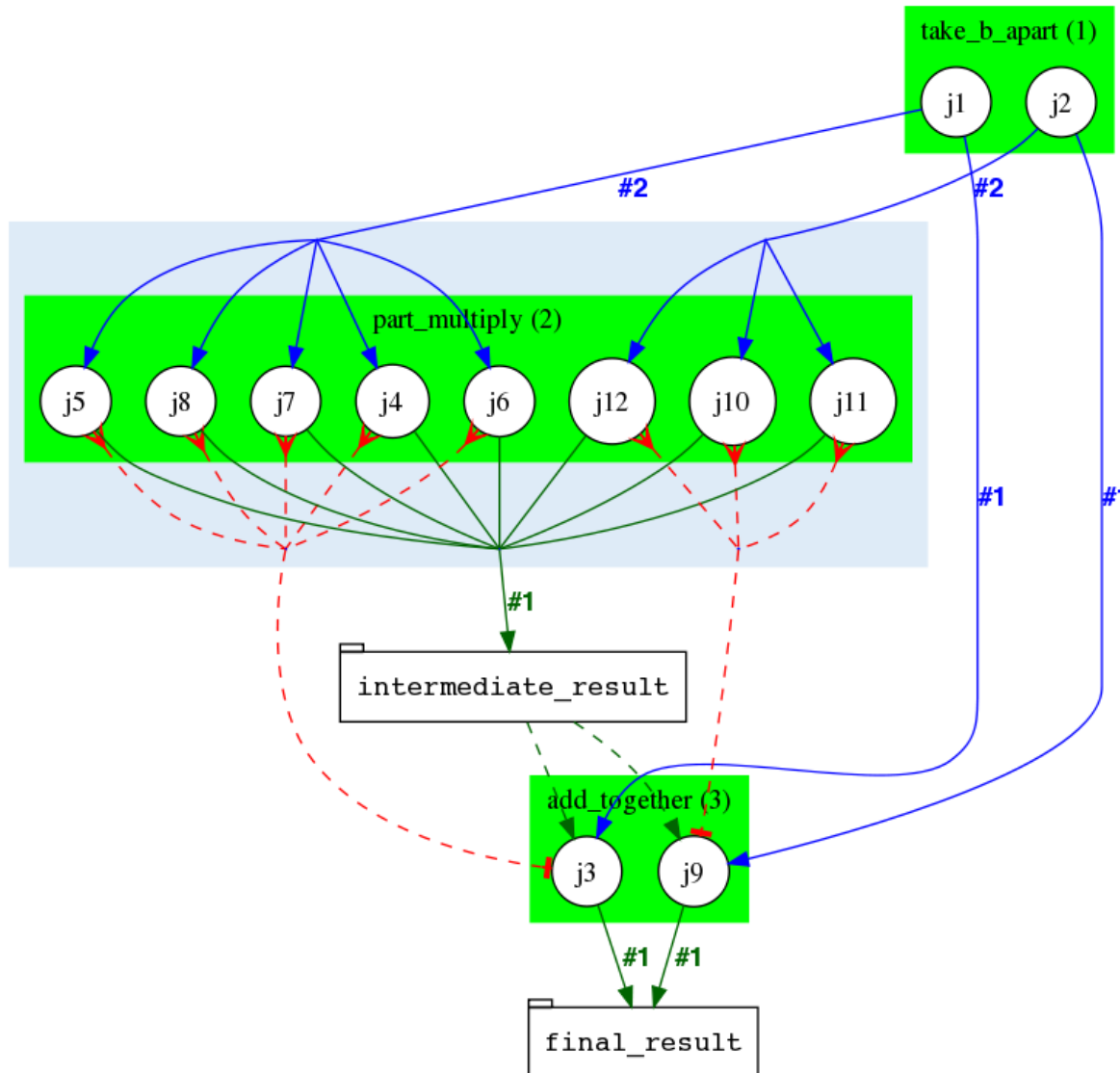
# Semaphored dataflow (allows returning control back to trunk)



For gathering these threads back we now employ a trick with semaphores: the gathering point ("funnel job") is a job that is created by the same "factory job" that spawns jobs corresponding to independent threads.

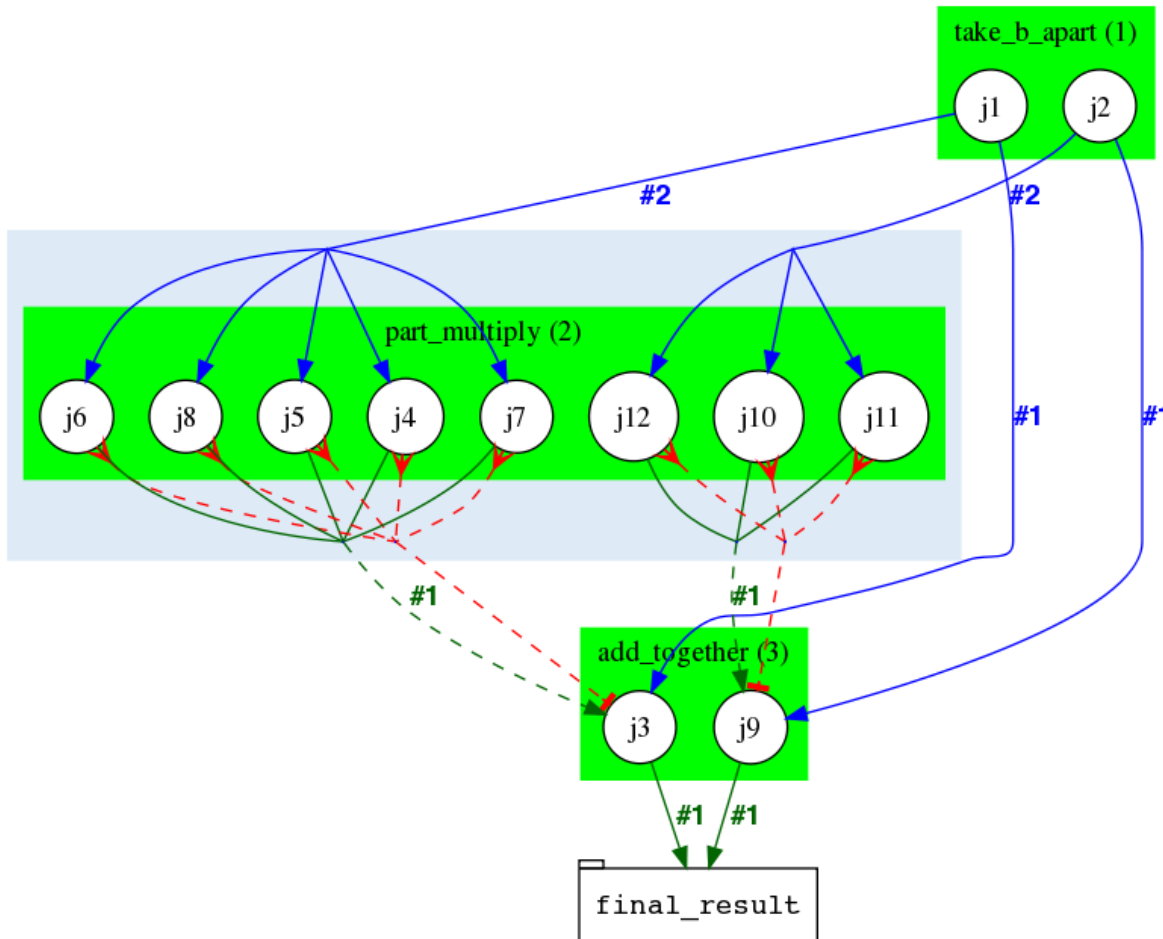
The "funnel job" is then linked to the "fan jobs" by a two-way link which blocks the funnel job from starting until all "fan jobs" are finished.

# Dataflowing into external tables (old hacky way)



In the old model an external table had to be created and fan jobs were dataflowing their data into it. Then each funnel job had to figure out which rows belonged to it.

# Accumulated dataflow (new proposed way)



- This is a new mechanism that allows semaphoring jobs to send data directly to their funnel job.
- Same dataflow API is used for sending. Existing param API is used for receiving.
- The Hive system makes sure a job only gets data that was intended for it.
- Why the name?

# Usage example:

dataflow rule from PipeConfig:

```
1 => [ ':///accu?partial_product={digit}' ]
```

Sending code (standard dataflow API call) :

```
$self->dataflow_output_id( { # here we are sending only one partial product
    'a_multiplier'      => $self->param('a_multiplier'),
    'digit'             => $self->param('digit'),
    'partial_product'   => $self->param('partial_product')
}, 1);
```

[It ends up in accu table somewhere]

Retrieving code (standard param API call) :

```
my $partial_product = $self->param('partial_product'); # an assembled hash
```

# Final remarks

- Four signatures correspond to different types of assembled data:
  - [foo] indexed/ordered array
  - [] pushed/unordered array
  - {bar} hash
  - {} multiset
- Potential storage inefficiency [trade-off]
- Untested on large datasets
- Available from Sanger-internal git repository, to be included in next CVS release



# Acknowledgements

**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**



**Quantomics**

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

