## Motivation

**Manual job handling**

- Manual job submissions
- Manual reruns on job or system failure
- Painful pipeline and data updates

## Motivation

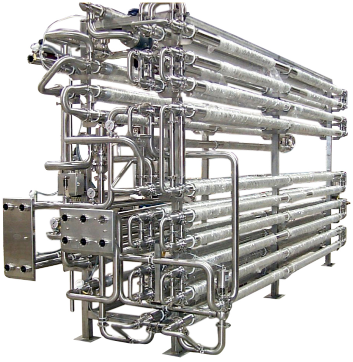
**Manual job handling**

- Manual job submissions
- Manual reruns on job or system failure
- Painful pipeline and data updates

**Intelligent pipeline**

- Automatic job re/submissions
- Automatic queue and memory limits
- Automatically resume unfinished steps

## Motivation

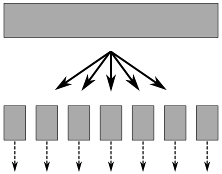


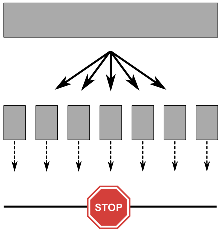
**Nice and Shiny...?**

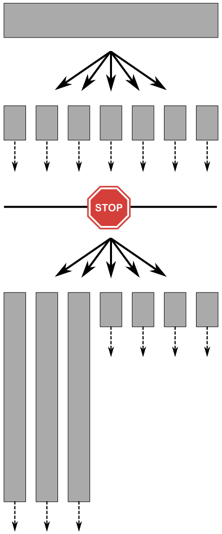
## Motivation

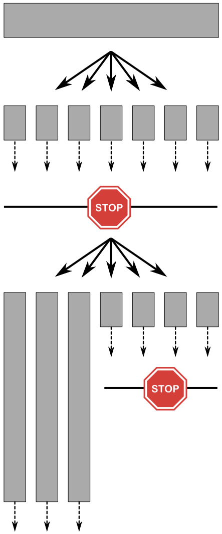
Tenerife, 2002

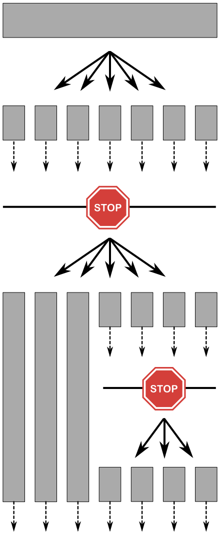
**... or a complete mess?**

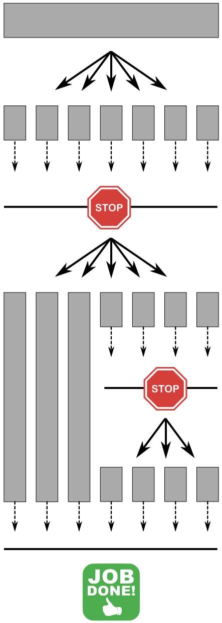



STOP


STOP


STOP
STOP


STOP
STOP


STOP
STOP
JOB DONE!

## Runners: Lightweight pipeline framework

No dependencies

- very easy to install
- very fast to run

Portability - runner pipelines can be executed in various environments

- on compute farms (LSF, SLURM)
- single multiprocessor machine
- local single CPU mode (useful for debugging or when the farm is down)

Pipelines divided into steps

- when interrupted, only unfinished steps are repeated
- error recovery from one-time errors (power down, temporary IO errors)

Parallel programming for runners is easy and natural

Chaining of multiple pipelines

## Runner code example (pseudocode)

```
# Run ten thousand jobs in paralel
for (i=0; i<10,000; i++)
{
    method     = 'my_function';         # The function to execute
    checkpoint = '/path/to/file.i';     # If exists, the step is done
    params     = (i,'some data');       # Pass arbitrary data
    spawn(method,checkpoint,params);    # This schedules the jobs
}
wait();  # This executes the tasks in parallel and waits for the results

# All tasks finished, the program can continue,
# possibly spawning more jobs
...

print "Mission accomplished!\n";
all_done();
```

## Runner code example (functional perl code)

```perl
# Run ten thousand jobs in paralel
for (my $i=0; $i<10_000; $i++)
{
    my $method     = 'my_function';             # The function to execute
    my $checkpoint = "/path/to/file.$i";        # If exists, the step is done
    my @params     = ($i,'some data');          # Pass arbitrary data
    $self->spawn($method,$checkpoint,@params); # This schedules the jobs
}
$self->wait;  # This executes the tasks in parallel and waits for the results

# All tasks finished, the program can continue,
# possibly spawning more jobs
...

print "Mission accomplished!\n";
$self->all_done;
```

## Runner usage example

```
# Create a sample config file
run-my-pipeline +sampleconf > my.conf

# Edit the config if not happy with the defaults
vi my.conf

# Run in daemon mode, check jobs every 5 minutes, send an email when done.
run-my-pipeline +config my.conf +loop 300 +mail usr@cool.edu -o outdir
```

## Runner installation

```
# Get the code
cd $HOME
git clone git://github.com/VertebrateResequencing/vr-runner.git

# Set the paths
export PATH="$HOME/vr-runner/scripts:$PATH"
export PERL5LIB="$HOME/vr-runner/modules:$PERL5LIB"

# Run a toy runner pipeline, first locally, then on the farm
run-test-simple +local
run-test-simple
```

## Cross-platform portability

```
# Run in daemon mode on LSF compute farm
run-pipeline +config my.conf +loop 300 -o outdir

# Run locally
run-pipeline +config my.conf +loop 300 -o outdir +local

# Run on a single multi-processor machine
run-pipeline +config my.conf +loop 300 -o outdir +js mpm

# Run in the SLURM environment
run-pipeline +config my.conf +loop 300 -o outdir +js slurm
```

## Runner options

```
Runner.pm arguments:
    +help              Summary of commands
    +config <file>     Configuration file
    +js <platform>     Job scheduler: LSF, MPM, SLURM
    +kill              Kill all running jobs
    +local             Do not submit jobs to LSF, but run serially
    +loop <int>        Run in daemon mode with <int> seconds sleep intervals
    +mail <address>    Email when the runner finishes
    +maxjobs <int>     Maximum number of simultaneously running jobs
    +retries <int>     Maximum number of retries.
    +sampleconf        Print a working configuration example
```