# Parallel Horace – design principles

## User interaction.

1. A user expected to continue to use Horace as he does now, i.e. typing or scripting Horace commands.
2. If appropriate powerful machine or parallel cluster is available to a user, user would enable parallel capabilities by issuing a simple command. (e.g. **hpc on**), which spawns execution of the time consuming operations in parallel.
3. The parallel Horace’s features are configured using Herbert configuration classes and available for advanced users to configure and fine-tune. A simplified default version of the framework should run without any users configuring.
4. User may or may not have access to parallel computing toolbox and distributed Matlab server. The access to these resources improves user’s calculation capabilities, but the basic access to parallel Horace resources expected to be independent on these resources.
5. The commands which may take long time and will benefit from parallelization are **gen\_sqw cut\_sqw**, **unit operations**, **tobyfit.fit** ,**sqw\_eval** and **symmetrise**.

## Technical opportunities, constrains and choices

1. All Horace algorithms benefiting from parallelization can be summarized by the following pseudocode:

*while* condition(Data):

*For* i=1:N\_iterations

Results(i) = Do\_something(Data,i)

*End*

condition,summary =

reduce\_results\_process\_condition(Data,Results)

*end* (condition)

*return* summary

where expensive operations of interest on Data are in fact the operations over the pixels of one or the group of sqw objects. The sqw objects expected not fit the memory. The *Do\_something* operation can be efficiently executed independently on each pixel or small group of pixels while *reduce\_results\_process\_condition* request interprocess communications. The condition and summary variables can be easy fit into memory of a single node and exchanged through MPI interface.

The sufficient way for parallelizing such job would be division of the Data in *For* loop among MPI workers and the usage of MPI communications to gather Results and condition on a head MPI node.

1. A MPI job can always and only be executed in the form:

>> mpiexec n\_workers the\_mpi\_program

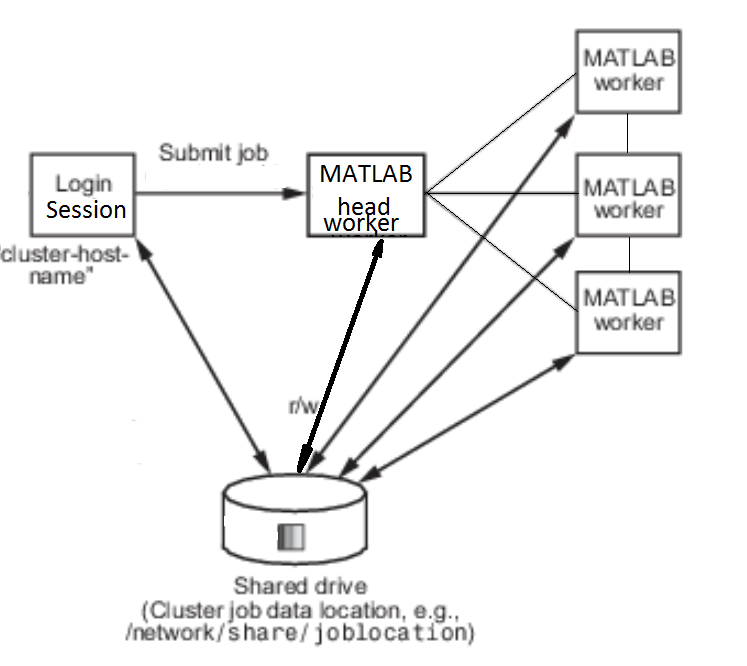
1. Matlab supports MPI jobs through **spmd** block only:

*smpd*

code\_to\_run(lab\_num)

*end* (smpd)

1. The Matlab server configuration necessary and sufficient to run Horace MPI jobs in 1) above has the form:



1. Matlab parallel computing toolbox and distributed Matlab server provide range of classes and functions (blocks) to deliver system independent operations. The blocks we are decided to use are :
   1. parcluster (parpool?) -- provides interface to physical resources and job control operations
   2. MPI commands (e.g. labnum, labsend, labreceive etc), working in a spmd block only and providing low level communication and synchronization routines between different Matlab workers.
   3. SPMD blocks itself, hiding the process of task splitting and job dispatching.

We are going to wrap these blocks into custom classes with common interface, to provide independence between Horace and presence of parallel computing toolbox and distributed server.

## Desighn choices.