Mantid on parallel computing architectures using MPI

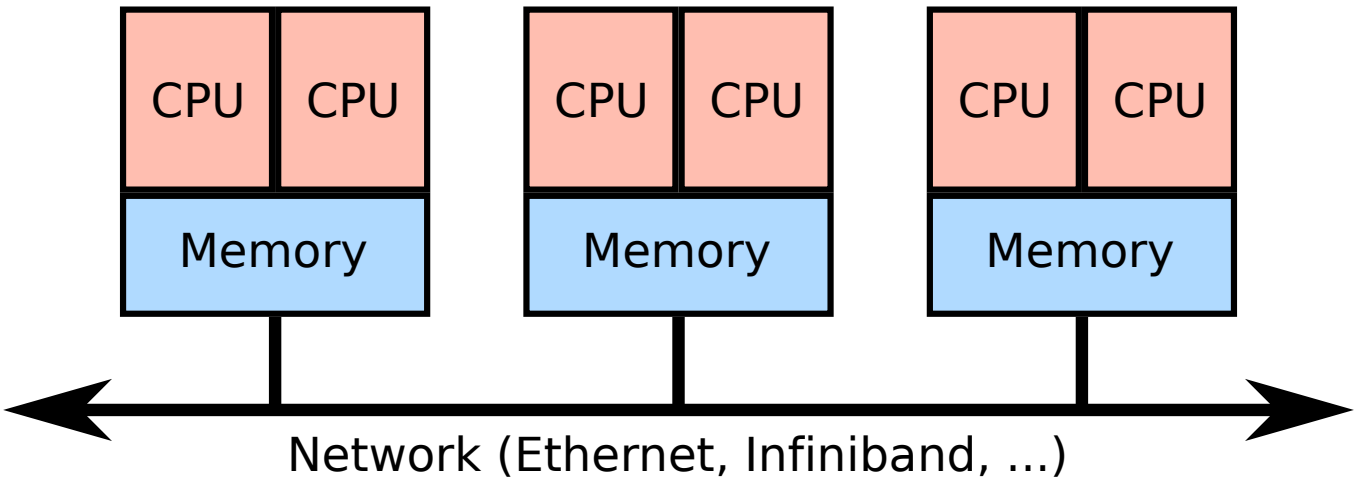
Simon Heybrock
simon.heybrock@esss.se

European Spallation Source

Motivation: Why parallelize over many processors?

Computationally challenging problems:

- Exceedingly long computation runtime on a single computer
- More memory required than available on single computer



MPI (Message Passing Interface) is a standard for libraries supporting data exchange between processes running on different computers.

Goals and approach

1. Speedup compared to current multi-threading Mantid.
2. MPI support for Mantid without being visible/explicit for user.
   - Ideally existing Python workflows should work without changes.

Technique areas with many small files

- Process individual files on each compute node.
- Not this talk!

Technique areas with large files (many events and/or spectra)

- Process a subset of spectra on each process:

Goals and approach

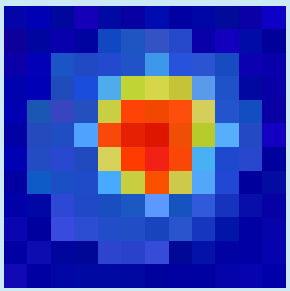
1. Speedup compared to current multi-threading Mantid.
2. MPI support for Mantid without being visible/explicit for user.
   - Ideally existing Python workflows should work without changes.

Technique areas with many small files

- Process individual files on each compute node.
- Not this talk!

Technique areas with large files (many events and/or spectra)

- Process a subset of spectra on each process:



Goals and approach

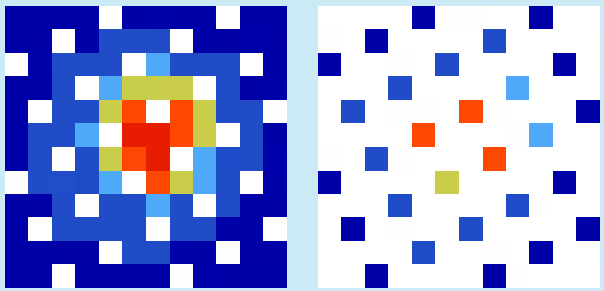
1. Speedup compared to current multi-threading Mantid.
2. MPI support for Mantid without being visible/explicit for user.
   - Ideally existing Python workflows should work without changes.

Technique areas with many small files

- Process individual files on each compute node.
- Not this talk!

Technique areas with large files (many events and/or spectra)

- Process a subset of spectra on each process:



Goals and approach

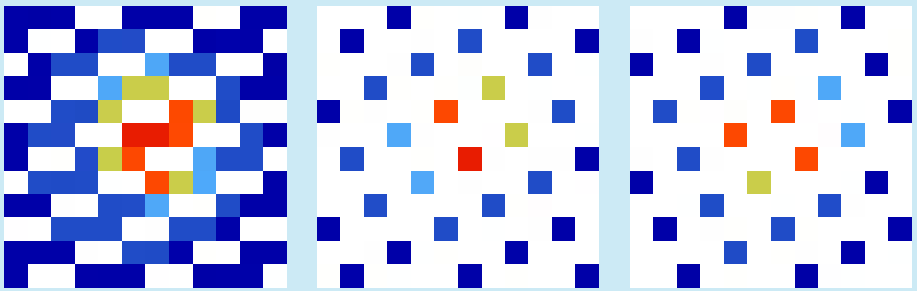
1. Speedup compared to current multi-threading Mantid.
2. MPI support for Mantid without being visible/explicit for user.
   - Ideally existing Python workflows should work without changes.

Technique areas with many small files

- Process individual files on each compute node.
- Not this talk!

Technique areas with large files (many events and/or spectra)

- Process a subset of spectra on each process:



Goals and approach

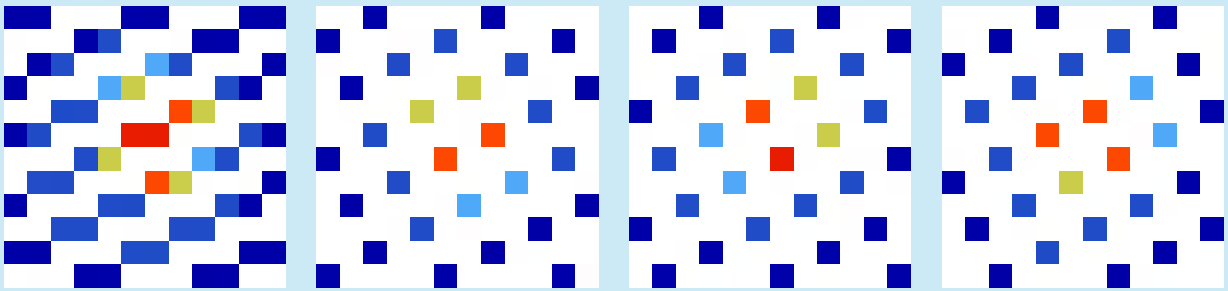
1. Speedup compared to current multi-threading Mantid.
2. MPI support for Mantid without being visible/explicit for user.
   - Ideally existing Python workflows should work without changes.

Technique areas with many small files

- Process individual files on each compute node.
- Not this talk!

Technique areas with large files (many events and/or spectra)

- Process a subset of spectra on each process:



Goals and approach

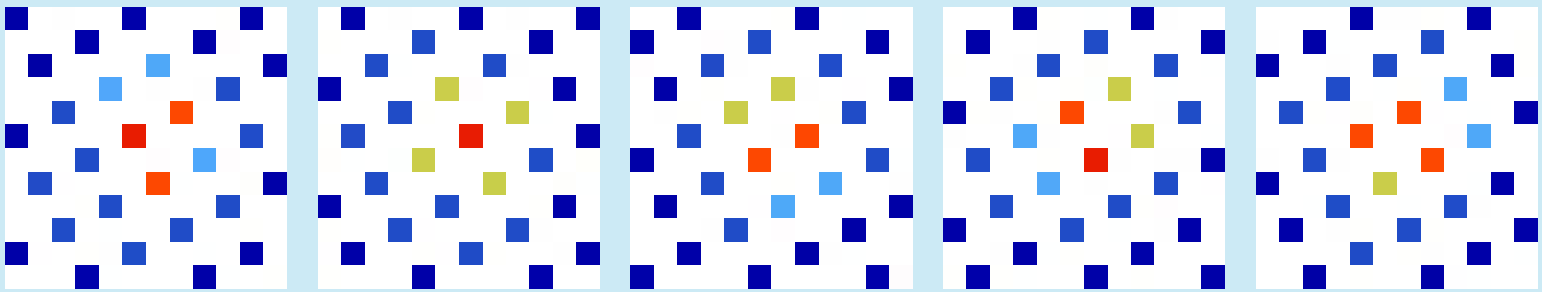
1. Speedup compared to current multi-threading Mantid.
2. MPI support for Mantid without being visible/explicit for user.
   - Ideally existing Python workflows should work without changes.

Technique areas with many small files

- Process individual files on each compute node.
- Not this talk!

Technique areas with large files (many events and/or spectra)

- Process a subset of spectra on each process:



Example: Python workflow

Consider a typical (simplified) Mantid script:

```
1  LoadEventNexus(Filename='ABCD_event.nxs',
2                 OutputWorkspace='ws')
3  Rebin(InputWorkspace='ws',
4        OutputWorkspace='ws', Params='100')
5  SumSpectra(InputWorkspace='ws',
6             OutputWorkspace='ws')
7  Rebin(InputWorkspace='ws',
8        OutputWorkspace='ws', Params='10')
9  SaveNexusProcessed(InputWorkspace='ws',
10                    Filename='result.nxs')
```

(replace `SumSpectra` with `Q1D` or `DiffractionFocussing` to obtain SANS or powder diffraction workflow)

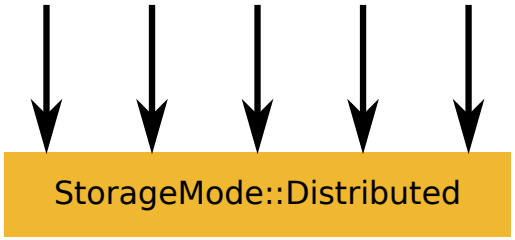
Example: Python workflow with MPI

I am a workspace

~ algorithm execution by a process

Run this script with 5 processes using MPI:

```
1  LoadEventNexus(Filename='ABCD_event.nxs',
2                 OutputWorkspace='ws')

3  Rebin(InputWorkspace='ws',
4        OutputWorkspace='ws', Params='100')

5  SumSpectra(InputWorkspace='ws',
6             OutputWorkspace='ws')

7  Rebin(InputWorkspace='ws',
8        OutputWorkspace='ws', Params='10')

9  SaveNexusProcessed(InputWorkspace='ws',
10                    Filename='result.nxs')
```



No changes to user script in this case!

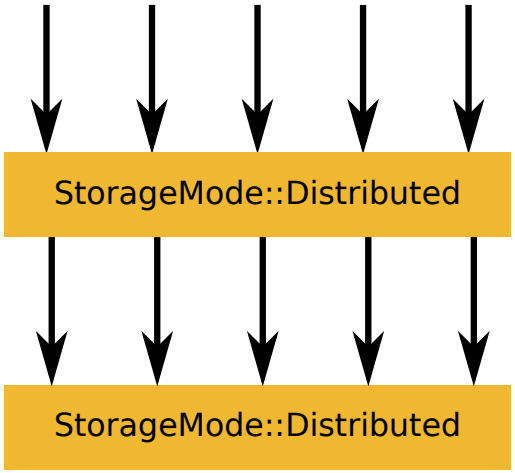
Example: Python workflow with MPI

I am a workspace

~ algorithm execution by a process

Run this script with 5 processes using MPI:

```
1  LoadEventNexus(Filename='ABCD_event.nxs',
2                 OutputWorkspace='ws')

3  Rebin(InputWorkspace='ws',
4        OutputWorkspace='ws', Params='100')

5  SumSpectra(InputWorkspace='ws',
6             OutputWorkspace='ws')

7  Rebin(InputWorkspace='ws',
8        OutputWorkspace='ws', Params='10')

9  SaveNexusProcessed(InputWorkspace='ws',
10                    Filename='result.nxs')
```



No changes to user script in this case!

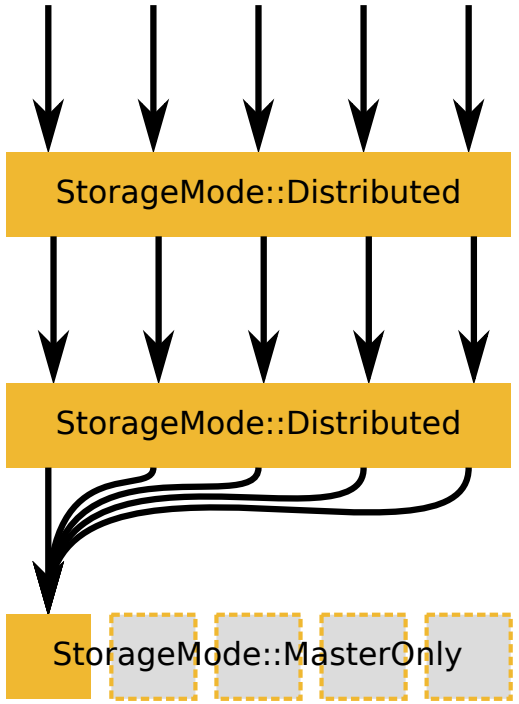
Example: Python workflow with MPI

I am a workspace

~ algorithm execution by a process

Run this script with 5 processes using MPI:

```
1  LoadEventNexus(Filename='ABCD_event.nxs',
2                 OutputWorkspace='ws')

3  Rebin(InputWorkspace='ws',
4        OutputWorkspace='ws', Params='100')

5  SumSpectra(InputWorkspace='ws',
6             OutputWorkspace='ws')

7  Rebin(InputWorkspace='ws',
8        OutputWorkspace='ws', Params='10')

9  SaveNexusProcessed(InputWorkspace='ws',
10                    Filename='result.nxs')
```



No changes to user script in this case!

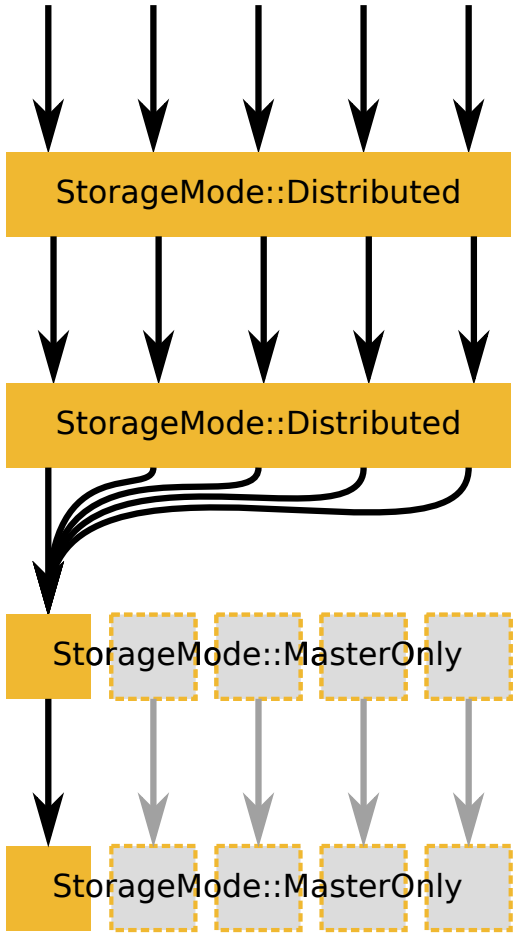
Example: Python workflow with MPI

I am a workspace

~ algorithm execution by a process

Run this script with 5 processes using MPI:

```
1  LoadEventNexus(Filename='ABCD_event.nxs',
2                 OutputWorkspace='ws')

3  Rebin(InputWorkspace='ws',
4        OutputWorkspace='ws', Params='100')

5  SumSpectra(InputWorkspace='ws',
6             OutputWorkspace='ws')

7  Rebin(InputWorkspace='ws',
8        OutputWorkspace='ws', Params='10')

9  SaveNexusProcessed(InputWorkspace='ws',
10                    Filename='result.nxs')
```



No changes to user script in this case!

Example: Python workflow with MPI

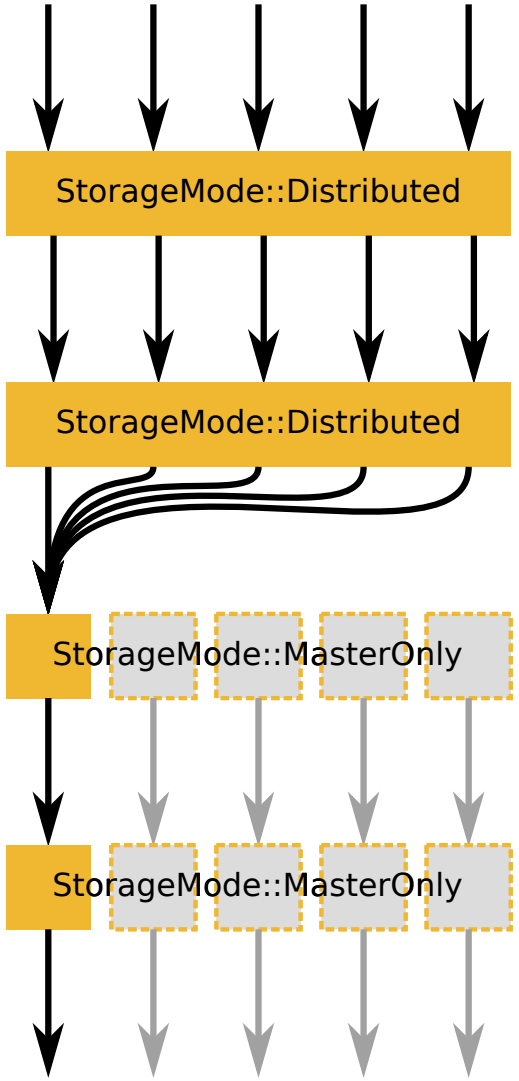
I am a workspace

~ algorithm execution by a process

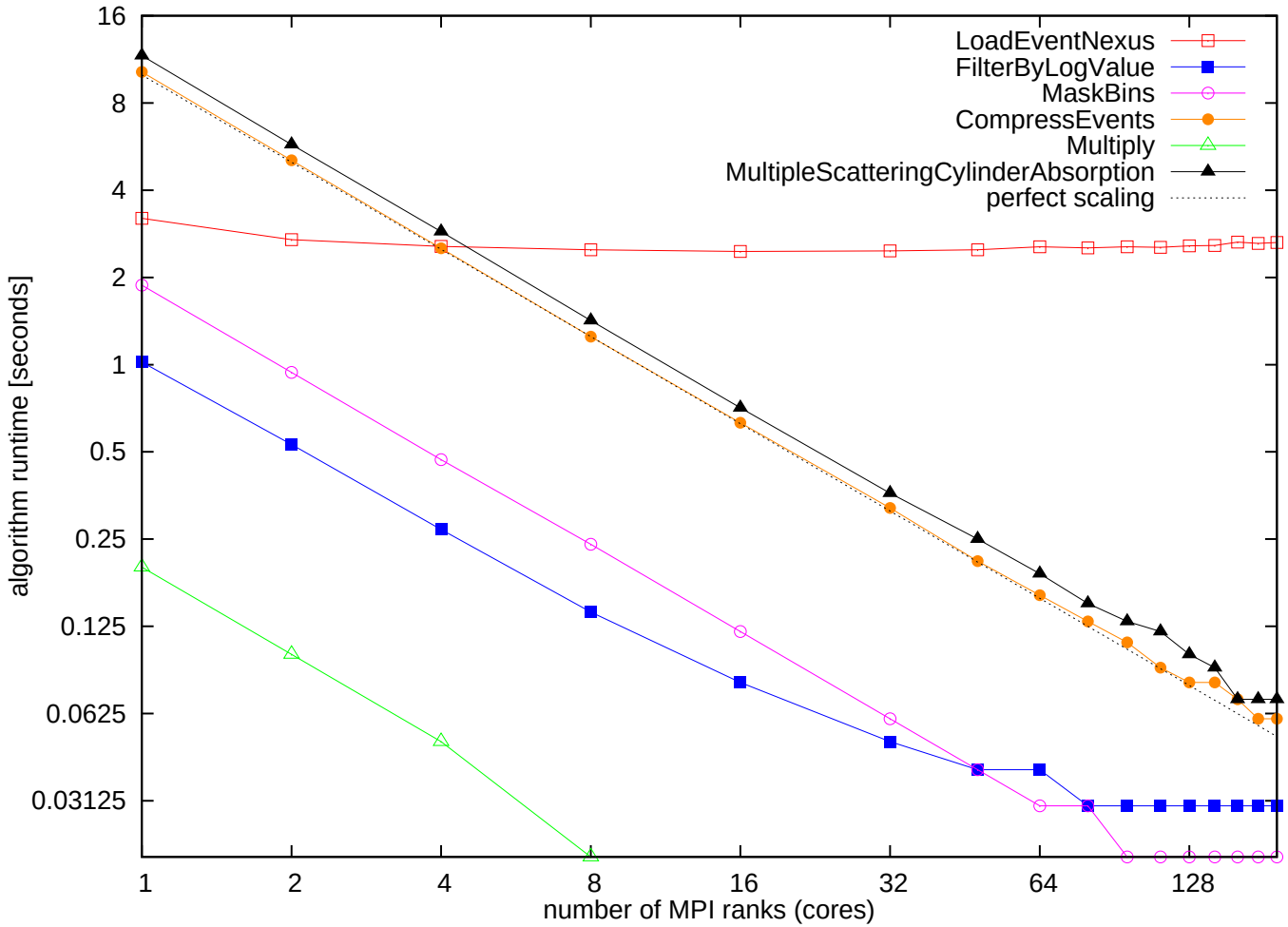
Run this script with 5 processes using MPI:

```
1  LoadEventNexus(Filename='ABCD_event.nxs',
2                 OutputWorkspace='ws')

3  Rebin(InputWorkspace='ws',
4        OutputWorkspace='ws', Params='100')

5  SumSpectra(InputWorkspace='ws',
6             OutputWorkspace='ws')

7  Rebin(InputWorkspace='ws',
8        OutputWorkspace='ws', Params='10')

9  SaveNexusProcessed(InputWorkspace='ws',
10                    Filename='result.nxs')
```

No changes to user script in this case!

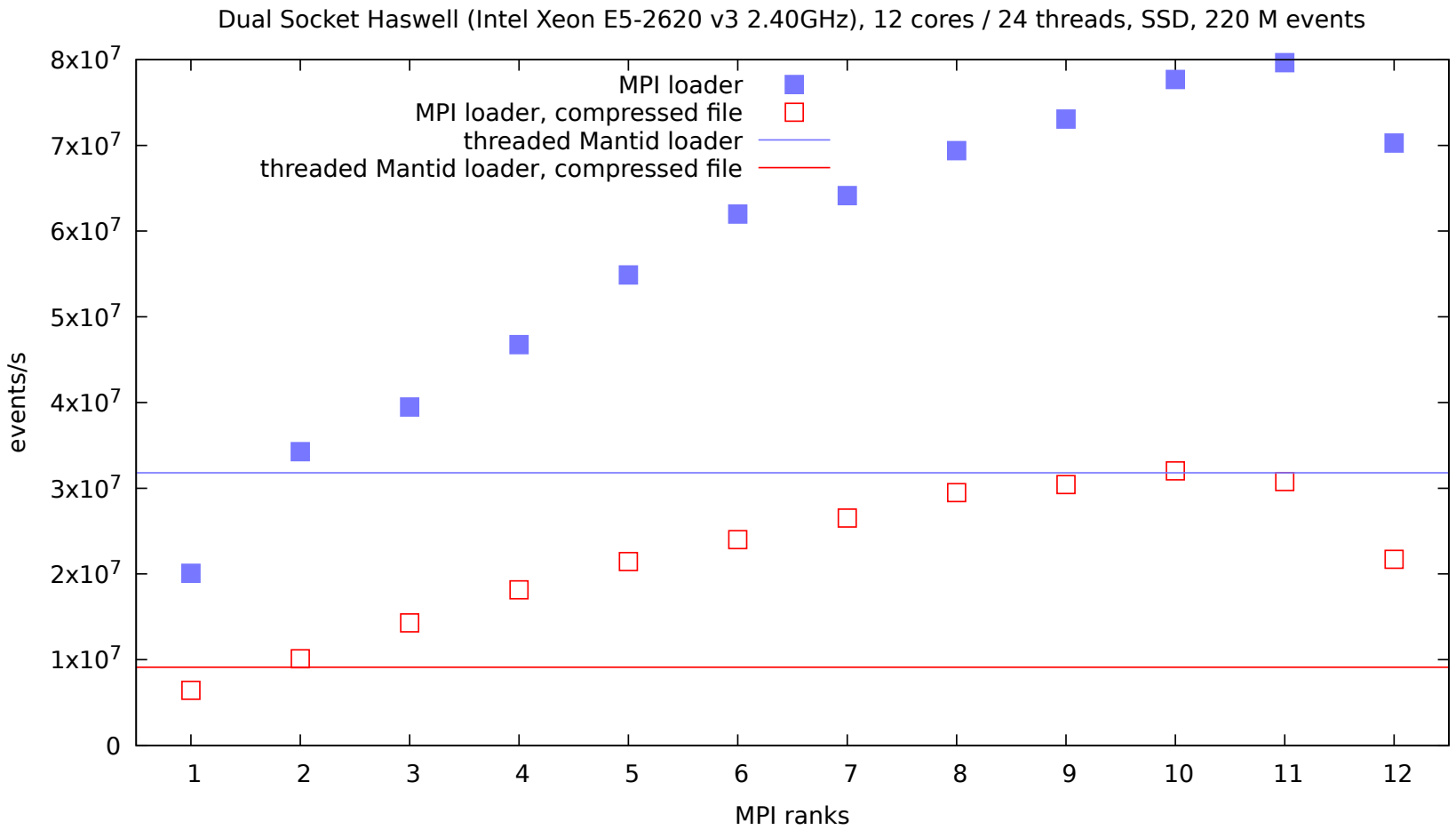


Performance scaling — results for selected algorithms



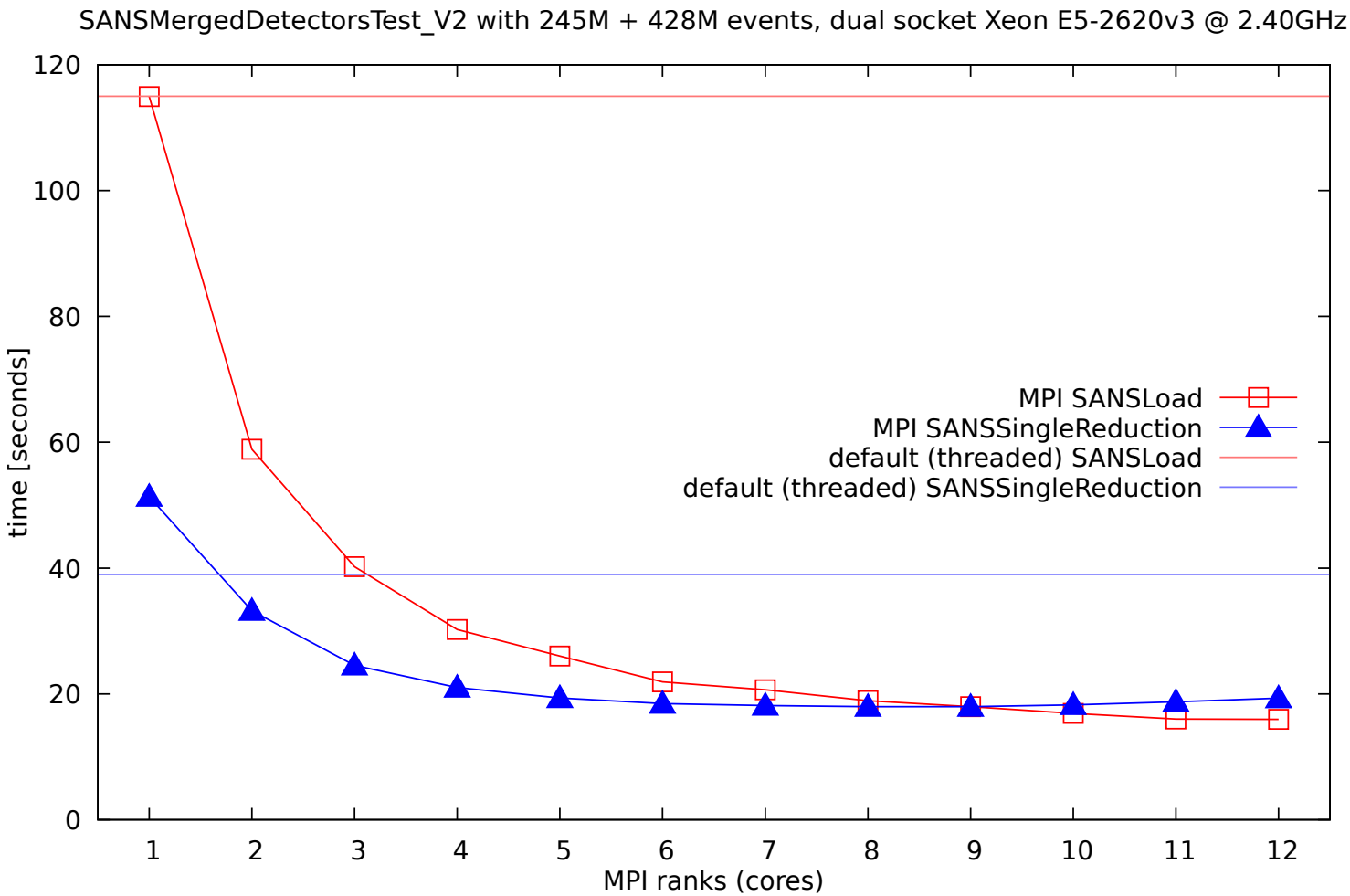
- Algorithms doing I/O generally show limited or no scaling.
- I/O-heavy workflows will exhibit limited scaling.

Optimized MPI version of `LoadEventNexus` (with Lamar Moore)



- Currently not all flavors of NeXus event files supported.
- Tests on cluster with parallel file system pending.

Performance scaling — SANS2D workflow



- Currently only limited scaling due to expensive algorithm property validation and loading of masking files.

Status and plans (1/2): Python scripts / batch reduction

`MatrixWorkspace`-based workflows

- Distributed implementation of `LoadEventNexus`
- Fair number of common algorithms supported
  - *Supported* does not imply *distributed* (see `SaveNexusProcessed` example above)
- Python workflow support without changes?
  - Works for scripts calling only algorithms.
  - Support for lower-level API calls needs more work (in combination with `MasterOnly` workspaces).
- Basic `SNSPowderRedux` and `SANS2D` workflows mostly there.

Workflows including multi-dimensional workspaces

- Under investigation (in particular `MDEventWorkspace`)

Status and plans (2/2): Live reduction and interactive workflows

Live reduction

Reduction of incoming event stream, "live" update of reduced data such as $I(Q)$.

- Should be relatively straightforward.
- Currently not clear whether/when this is required due to lower data rates.
  - Likely to be supported in the future.

Interactive workflows

Most likely no support for:

- Python console
- GUI (Mantidplot)

Distribution and documentation

- No binary release.
  - Usually compilation for specific MPI implementation is necessary, so any binary release would be highly facility- and machine-specific.
  - Not ready for production yet.
- Build instructions and detailed documentation available at `http://docs.mantidproject.org/nightly/development/AlgorithmMPISupport.html`.
  - Includes table of supported algorithms.

Conclusion

Running Mantid Python scripts with MPI

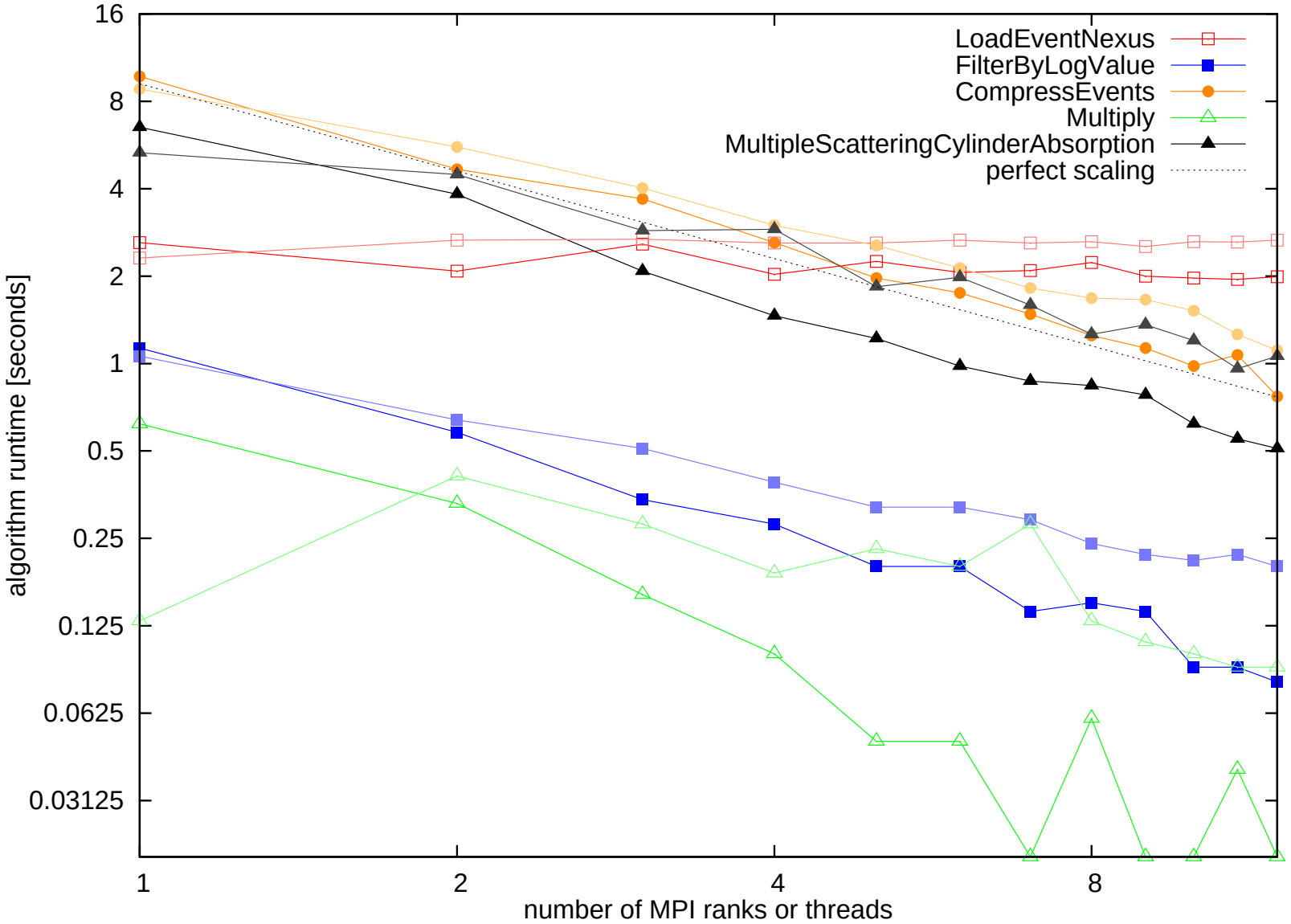
- Initial working version nearly there (for `MatrixWorkspace`).
- Good scaling of most algorithms that do not load or save files.
- (Sometimes greatly) improved workflow performance, depends on file sizes and balance between I/O and computation.
- In some cases MPI reduction is faster than threaded reduction *on the same machine*.

Next steps

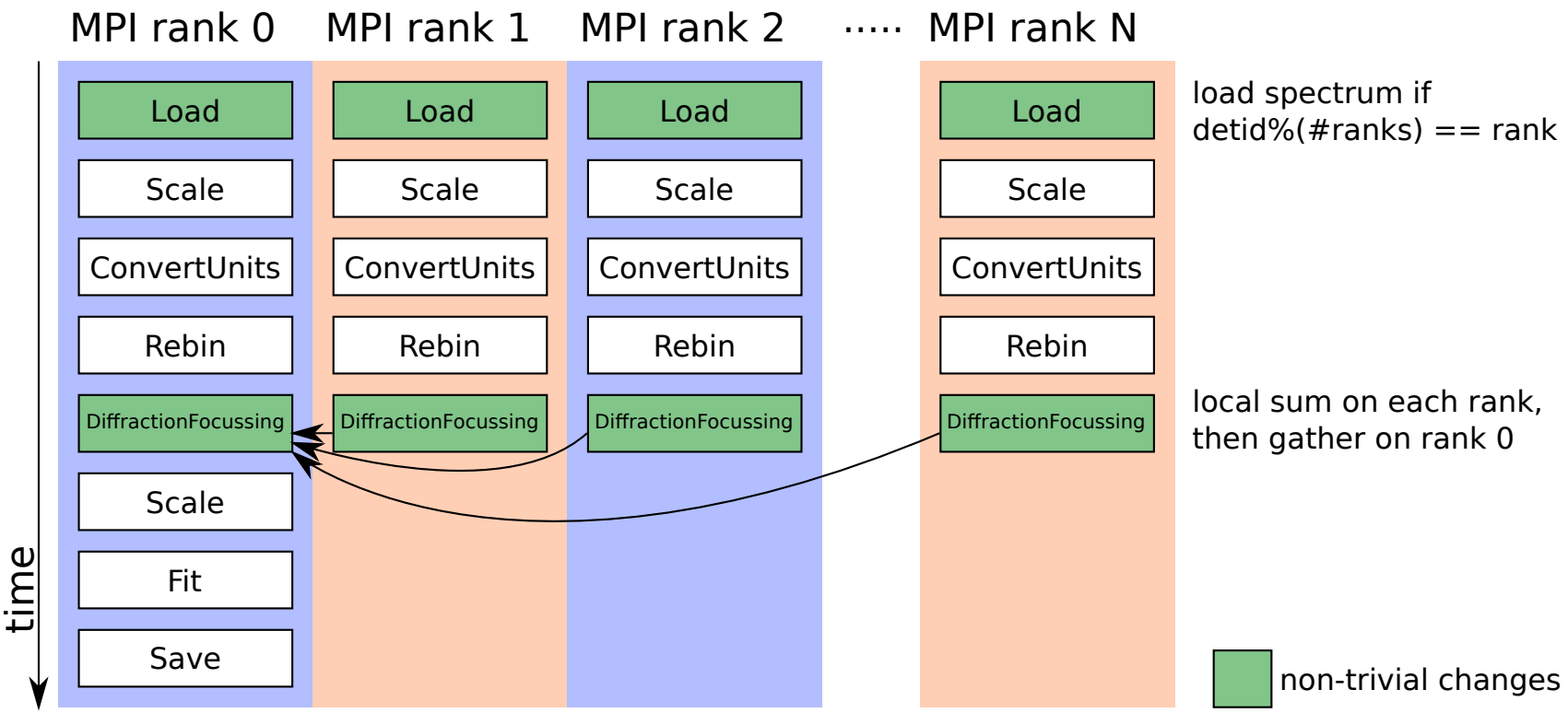
- Analyse shortcomings and limitations.
- Acceptance by other developers and power users?
- Not focussed on more optimizations or getting production ready at this stage.

Thank you for your attention!
Questions?

Backup slide: MPI vs. threading

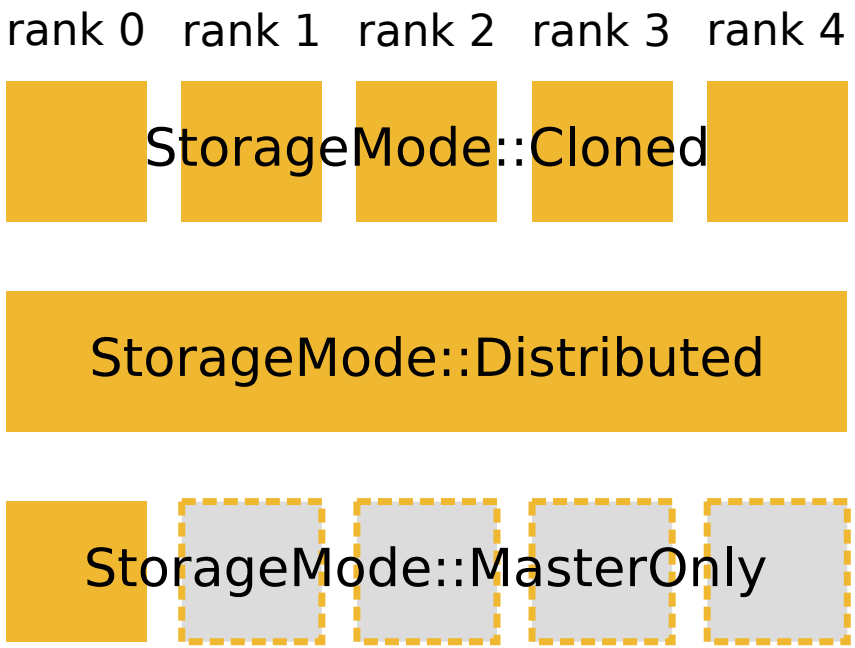


Backup slide: Powder Diffraction example



Backup slide: Workspace storage modes

- Workspace has a new field: `StorageMode`
- Algorithm behavior can be determined by `StorageMode`



Backup slide: How. . . ?

Basic idea

- Workspace stored distributed, i.e., across all MPI ranks.[1]
- Each rank executes an Algorithm. Typically Algorithm is not aware that it "sees" only parts of the detectors.
- Output of Algorithm is again a distributed Workspace.

The are some Algorithms that do need to know about the other MPI ranks and exchange data with them.

[1] Technically, there is an instance of a Workspace on each MPI rank. Logically we interpret these as a single workspace, given by the combination of all spectra from all ranks.