Mantid MPI

Simon Heybrock
simon.heybrock@esss.se

European Spallation Source

Motivation

Why parallelize over many nodes (processors)?

- Computation speed
- Memory (ESS: may get 10^8 events/s $\Rightarrow >$ 1 GB/s data)

Things to consider

- We have a large amount of code
  $\Rightarrow$ must minimize the amount of required code changes.
- Networks are much slower than memory access (10–100$\times$)
  $\Rightarrow$ need to minimize the amount of data moved between compute nodes.

How. . . ?

Crucial insight

Many algorithms treat all detectors/spectra independently.
In other words, in many cases:

- Workspace = list of spectra
- Algorithm = process each item in list (by itself)

$\Rightarrow$ Processing different spectra can be done on different MPI ranks

How. . . ?

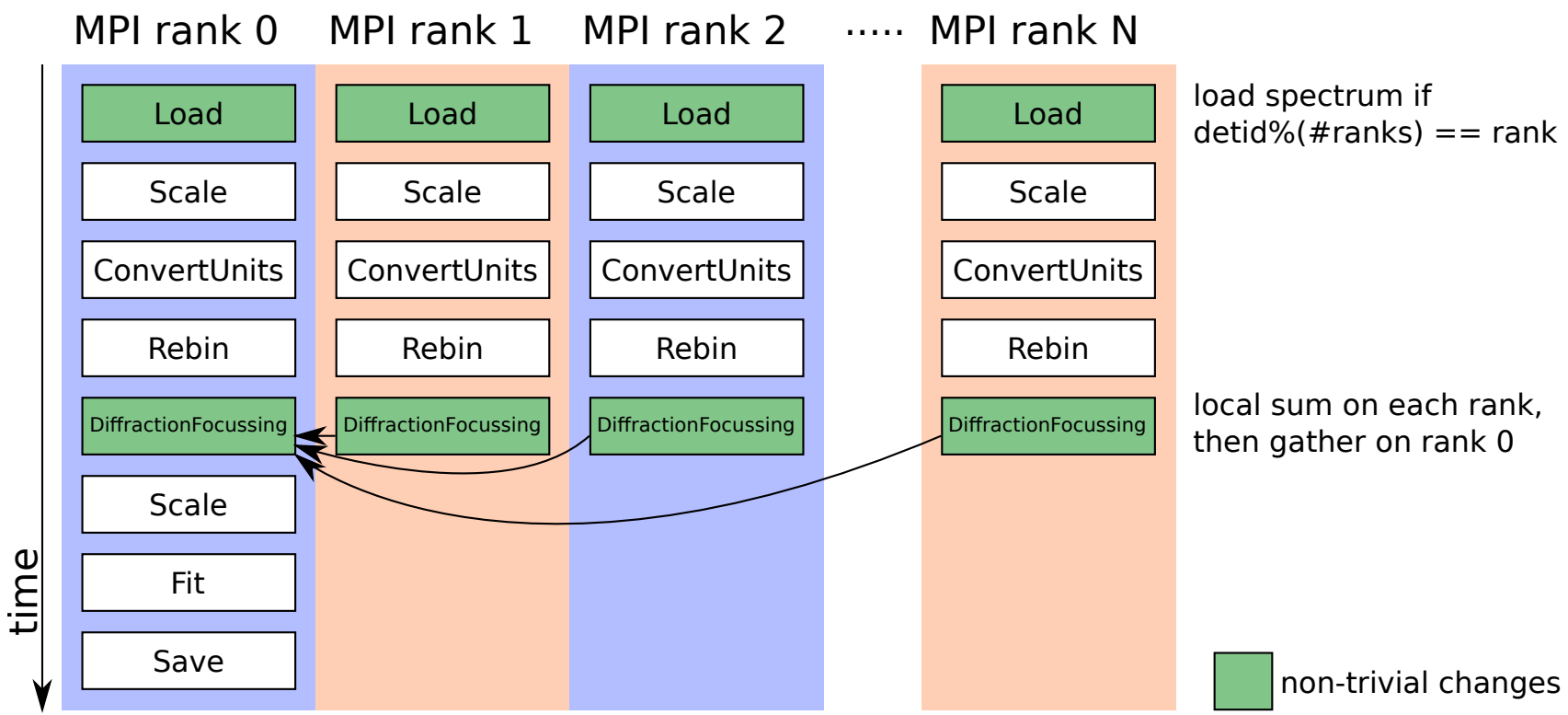
Basic idea

- 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.

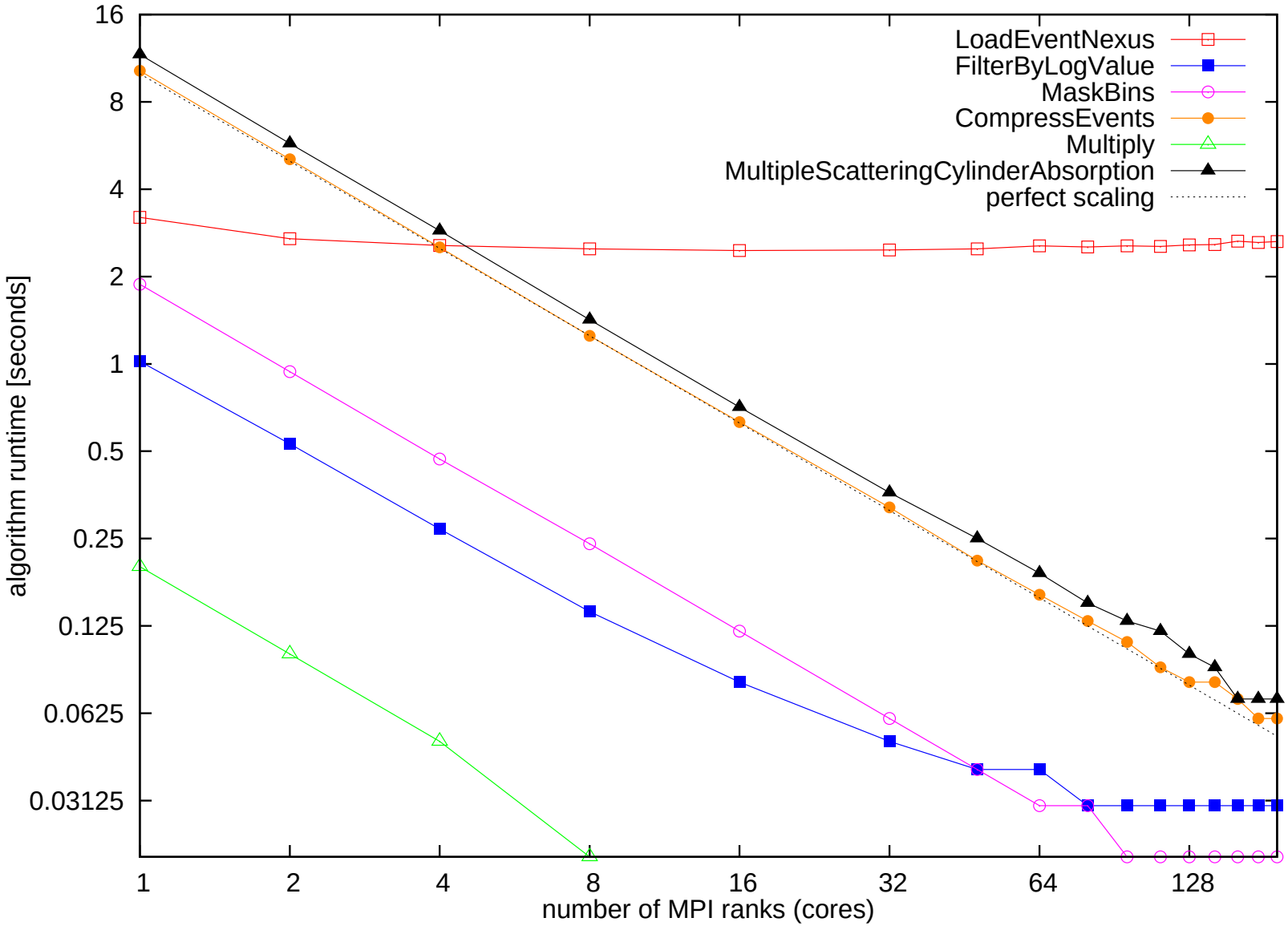
The are some Algorithms that do need to know about the other MPI ranks and exchange data with them.

[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.

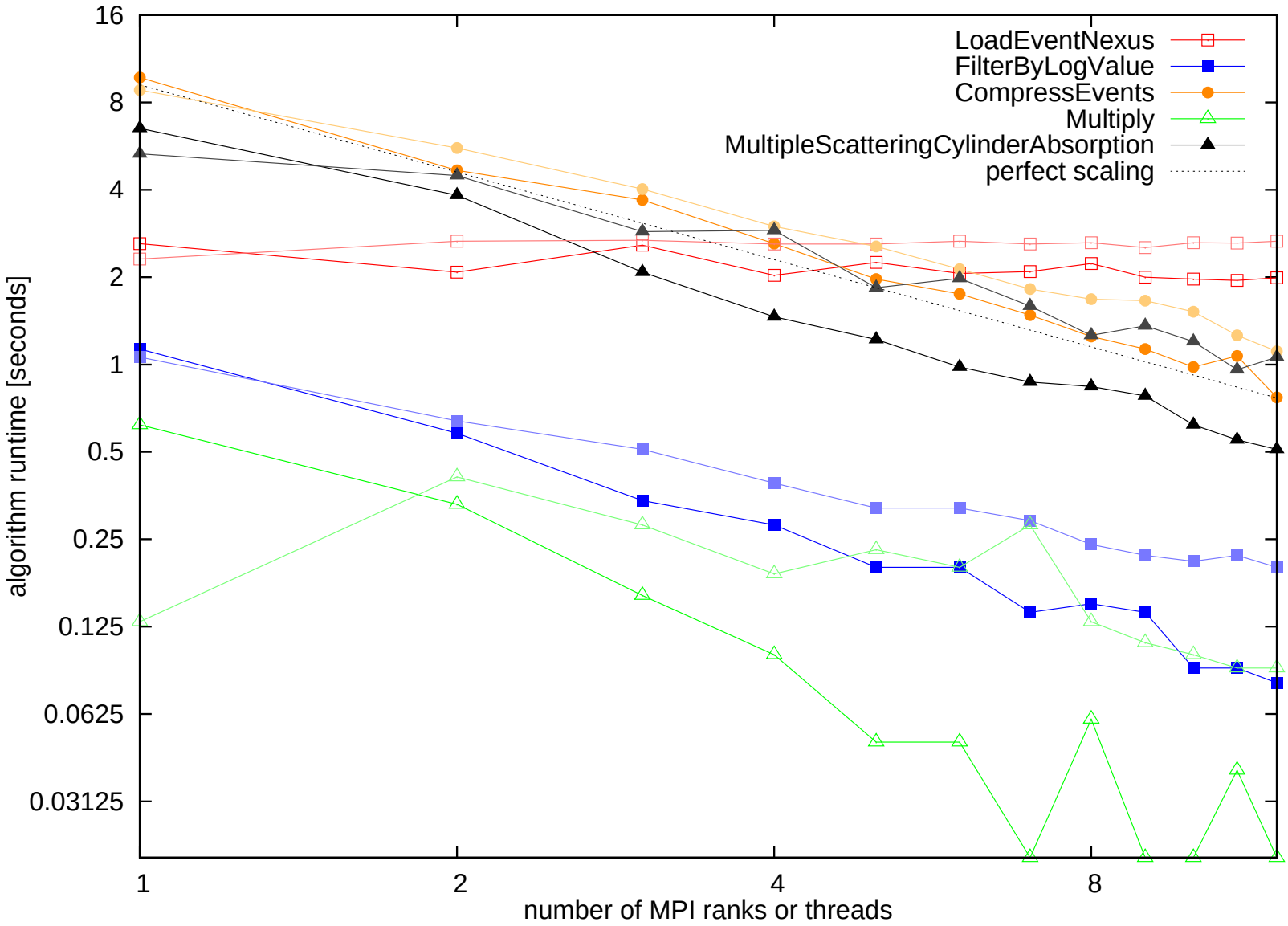
Example: Powder Diffraction



Results: SNSPowderDiffraction — strong-scaling on ESS cluster



Results: MPI vs. threading (12 core workstation)



Pros and cons (1/3)

Con: Need to modify all algorithms

- Requires (minor) modification of all algorithms.
- In many cases the only change would be the base class.

Con(?): Load balance

Neutron count in specific detector regions varies a lot between:

- instruments
- instrument configuration
- different experiments on the same instrument

$\Rightarrow$ Potential load imbalance

- Different distribution of detectors onto ranks for each case.
- Could be automated.

Pros and cons (2/3)

Pro: Not instrument specific

- Should work with all instruments with > 1 detector pixel.

Pro: Speed — increased throughput

- **Data reduction: detectors are loosely coupled $\Rightarrow$ speed**, i.e., good strategy from computational point of view.

Pro: Speed — reduced latency

- Event stream reduction: Each processor has to deal with fewer detector pixels $\Rightarrow$ decreased latency $\Rightarrow$ ESS "live" reduction

Pros and cons (3/3)

Pro: Few changes in higher level scripts

- No or minor changes to Python reduction scripts

Open questions: See below

There are several open questions where solution is not clear (yet).

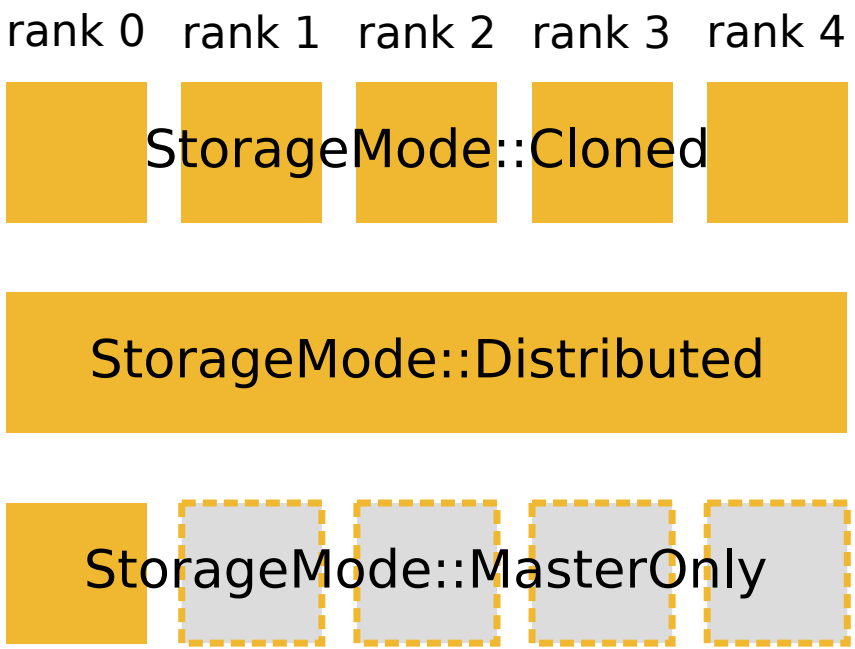
Alternative?

Slice time: rank N gets event data from $t = (N-1) \cdot \delta t$ to $N \cdot \Delta t$

- Histogram mode: more histograms, no gain from MPI
- Event mode: more spectra with fewer events
  - Probably (slightly) less efficient for most algorithms
  - Not so useful for event stream reduction

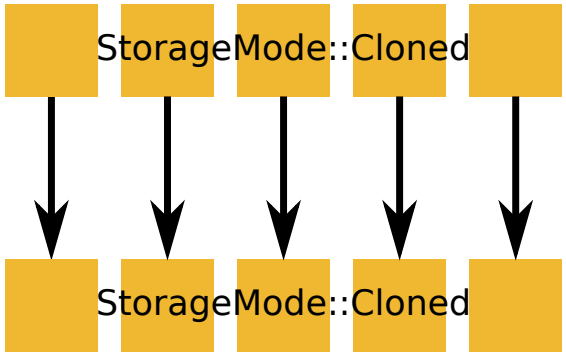
Workspace storage modes

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



Algorithm execution modes

Very useful:
Algorithm has non-virtual interface —
non-virtual `execute()` calls virtual `exec()`
⇒ add some logics to `execute()`



Simplest case

- Input workspace `StorageMode` ⇒
  Algorithm `ExecutionMode`
- Output workspace `StorageMode` =
  Input workspace `StorageMode`

No changes in specific algorithms required!

Examples:
Rebin, ConvertUnits, BinaryOperation, . . .

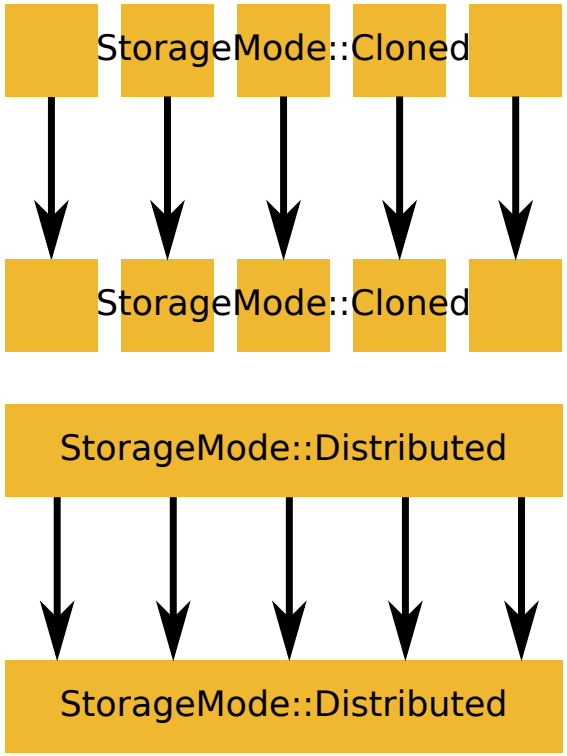
Algorithm execution modes

Very useful:
Algorithm has non-virtual interface —
non-virtual `execute()` calls virtual `exec()`
⇒ add some logics to `execute()`

Simplest case

- Input workspace `StorageMode` ⇒ Algorithm `ExecutionMode`
- Output workspace `StorageMode` = Input workspace `StorageMode`

No changes in specific algorithms required!



Examples:
Rebin, ConvertUnits, BinaryOperation, . . .

Algorithm execution modes

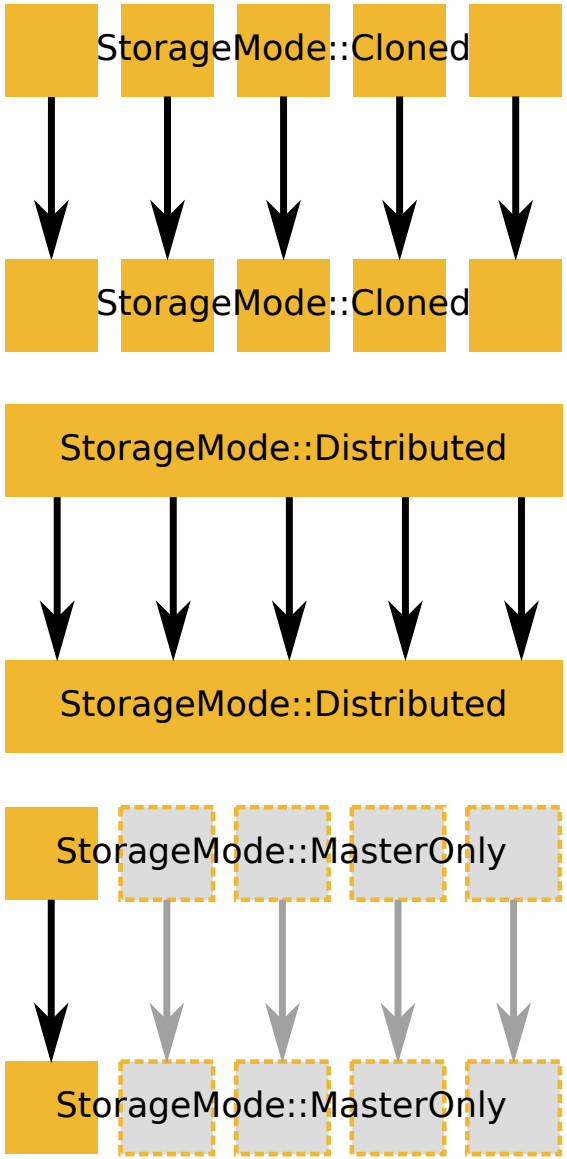
Very useful:
Algorithm has non-virtual interface —
non-virtual `execute()` calls virtual `exec()`
⇒ add some logics to `execute()`

Simplest case

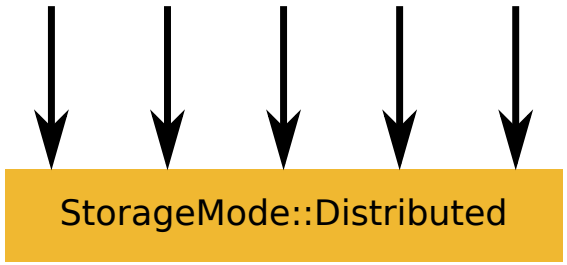
- Input workspace `StorageMode` ⇒ Algorithm `ExecutionMode`
- Output workspace `StorageMode` = Input workspace `StorageMode`

No changes in specific algorithms required!

Examples:
Rebin, ConvertUnits, BinaryOperation, . . .



Algorithm execution modes: non-trivial examples



`LoadEventNexus`

- Modified to load only subset of spectra

`SumSpectra`, `DiffractionFocussing`

- No changes: sum only within rank!
- MPI version: gather intermediate results from all ranks
- Only one spectrum left $\rightarrow$ transition to `StorageMode::MasterOnly`

`Save`

- Distributed store requires work

Algorithm execution modes: non-trivial examples

`LoadEventNexus`

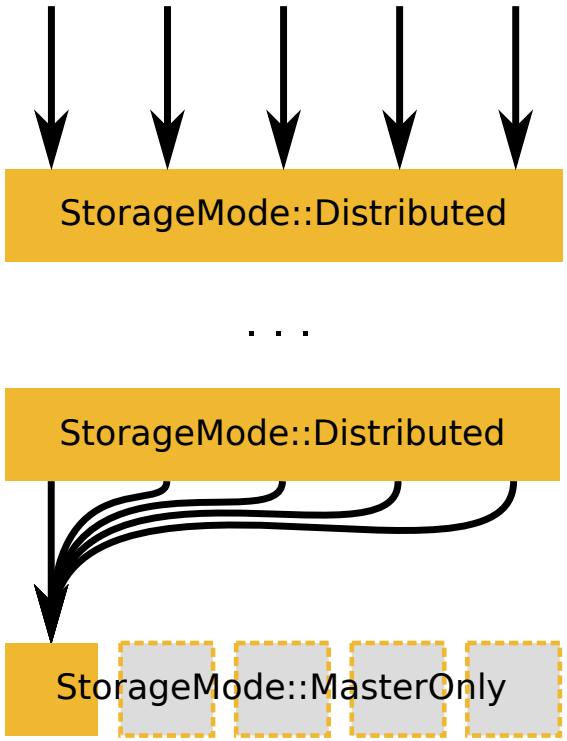
- Modified to load only subset of spectra

`SumSpectra`, `DiffractionFocussing`

- No changes: sum only within rank!
- MPI version: gather intermediate results from all ranks
- Only one spectrum left → transition to `StorageMode::MasterOnly`

`Save`

- Distributed store requires work



Algorithm execution modes: non-trivial examples

`LoadEventNexus`

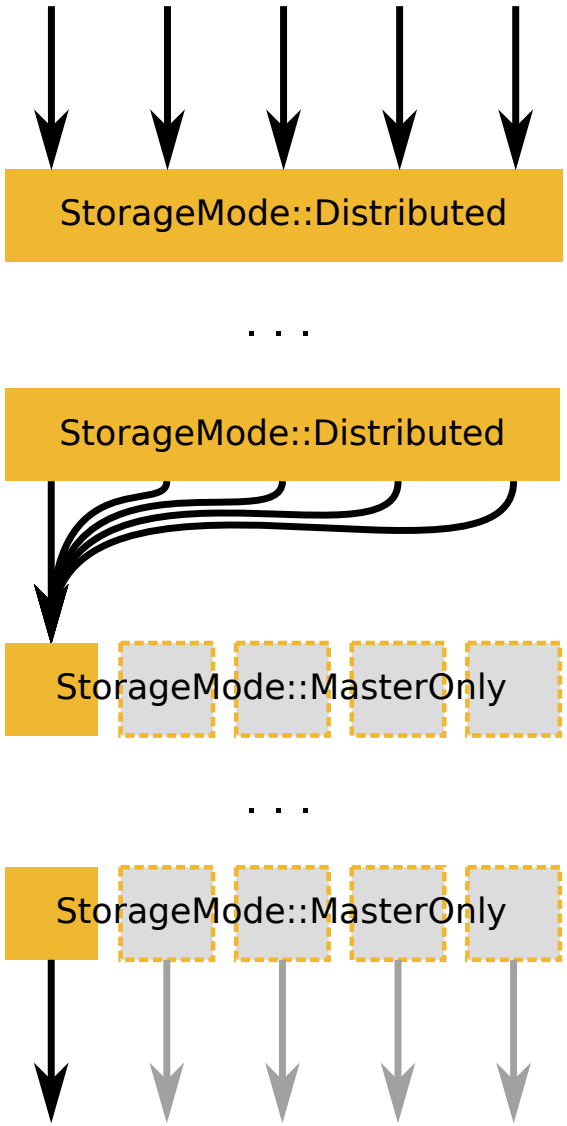
- Modified to load only subset of spectra

`SumSpectra`, `DiffractionFocussing`

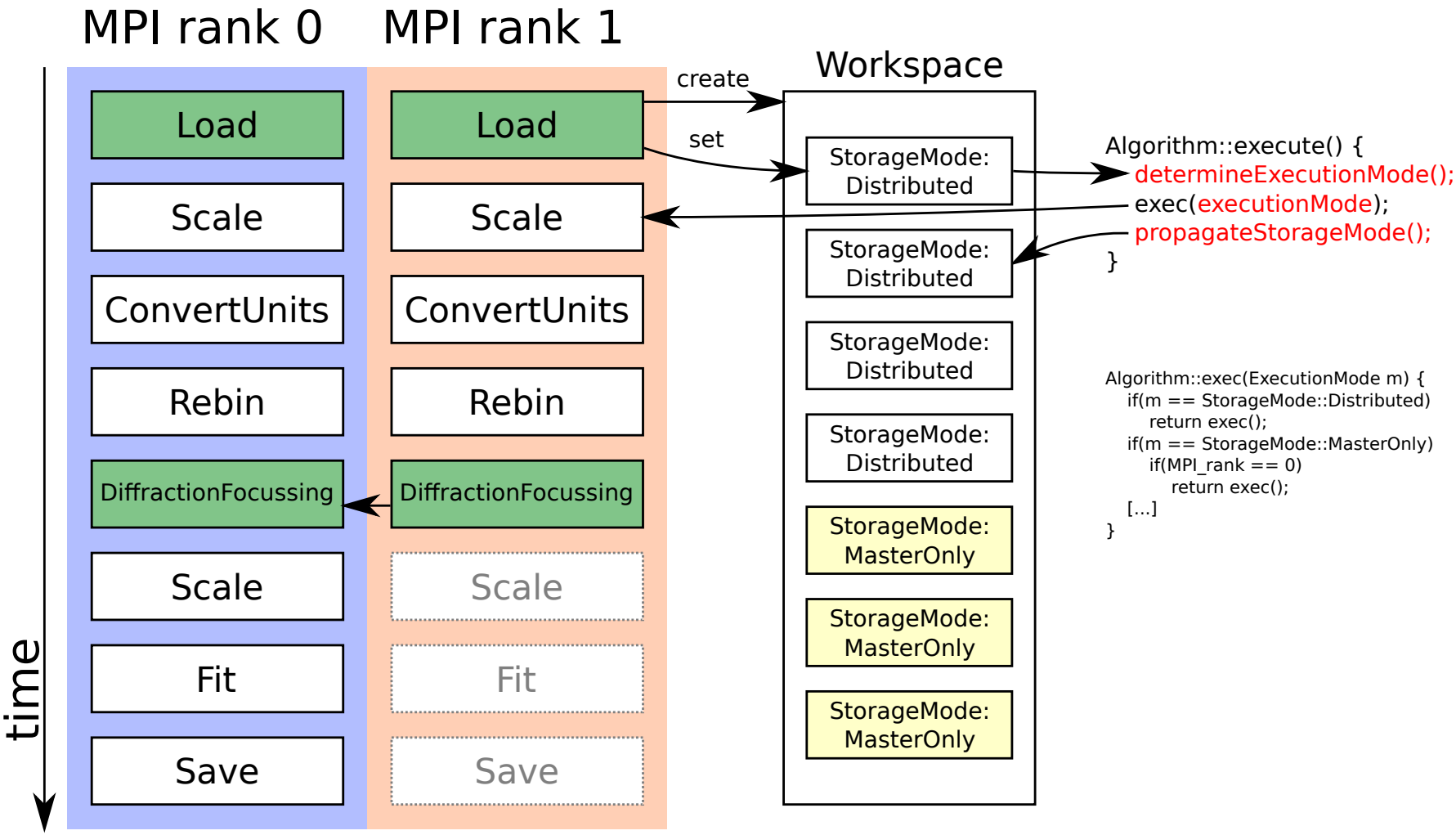
- No changes: sum only within rank!
- MPI version: gather intermediate results from all ranks
- Only one spectrum left → transition to `StorageMode::MasterOnly`

`Save`

- Distributed store requires work



Example: implementation



Open questions (1/2)

`MDWorkspace`

- Instead of splitting the detector array, cut/partition the MD volume.
- Will require significant coding effort (partial `MDGridBox`, partitioning algorithms, load balancing, `MDBox` and event redistribution, . . .).

Monitors

- ESS details unknown: event mode? high time resolution?
- Too big to clone monitor data on each MPI rank?
- Dedicated rank for monitors, how do other ranks access data?

Open questions (2/2)

Loading

- Non-live reduction — how do we efficiently read a file?
- Parallel load (HDF5), redistribute via MPI?

Visualization

How to visualize data that is scattered across many processes?

- ParaView can run with MPI back end, too — combine?
- Render on back end?
- Send relevant data to visualization server?

Integration with interactive reduction?

- Probably very difficult in the current Mantid (GUI) design?
- Not top priority, but should not be ignored completely.

Conclusions and discussion

Status?

- Basic concept seems ok, proof of concept is working.
- Needs close inspection (design review), cleanup, tests.
- Encouraging results: `SNSPowderReduction`, about 70 algorithms "ported".

Open questions

	Concept	Effort
MDWorkspace	ok on high level	medium – large
Monitors	depends	small – medium
Loading	ok on high level	medium
Visualization	unclear	medium – large
Interactive workflow	unclear	medium – large

- Most of these issues would be similar with different MPI design.