

McStas advanced language features

- and other important, not so well known details

Peter Willendrup^{1,5}

Emmanuel Farhi²

Erik Knudsen^{1,5}

Uwe Filges³

Kim Lefmann^{4,5}

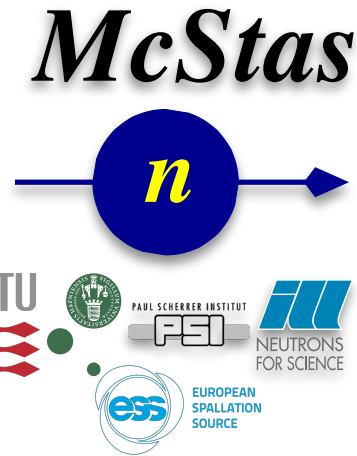
¹DTU Physics, Lyngby, Denmark

²Calcul Scientifique, Institut Laue-Langevin (ILL), Grenoble, France

³Niels Bohr Institute, University of Copenhagen, Copenhagen, Denmark

⁴Paul Scherrer Institut, Villigen, Switzerland

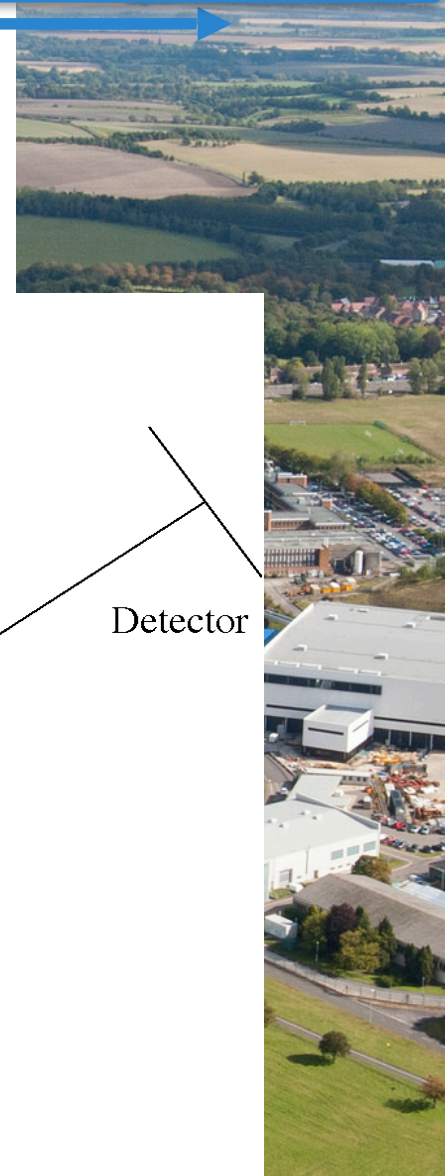
⁵ESS DMSC, Copenhagen, Denmark



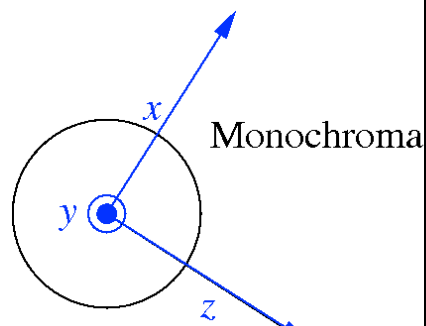
Topics

- ♦ A quick recap of the most important vocabulary
- ♦ Simulation to experiment comparison
- ♦ How to make your McStas more efficient
- ♦ McStas language macros
- ♦ Other important details





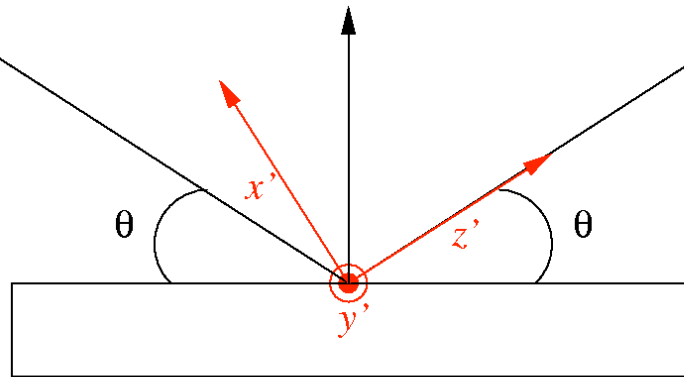
McStas: key concepts



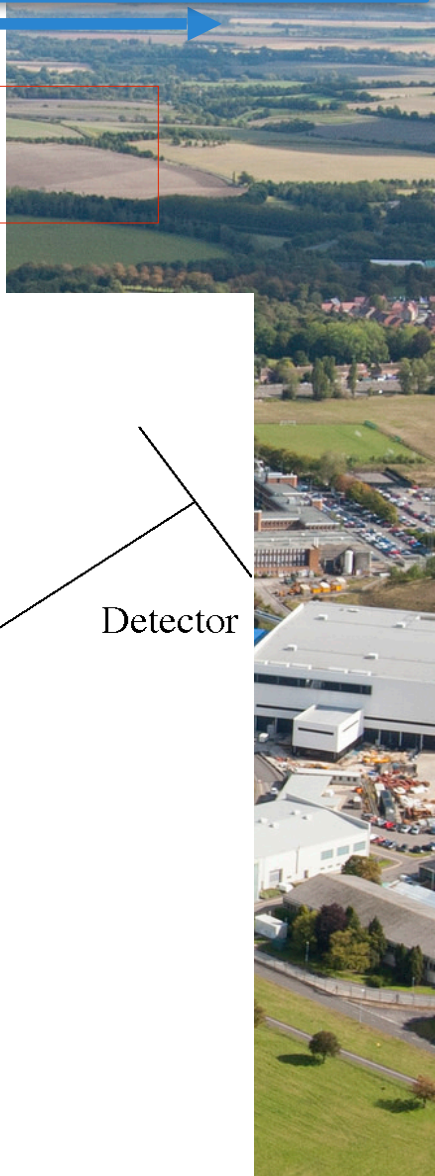
Monochromator

Neutron ray/package:
 Weight (p): # neutrons (left) in the package
 Coordinates (x,y,z)
 Velocity (v_x, v_y, v_z)
 Spin (s_x, s_y, s_z)
 Time (t)

Detector

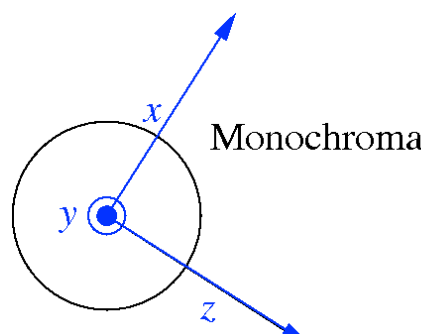


Crystal in Bragg scattering condition

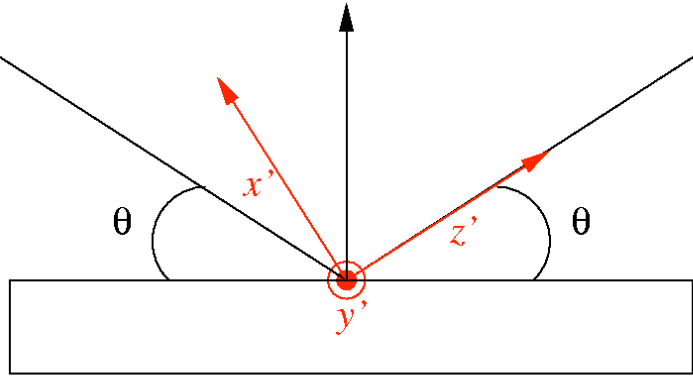


McStas: key concepts

Neutron state parameters are global variables!



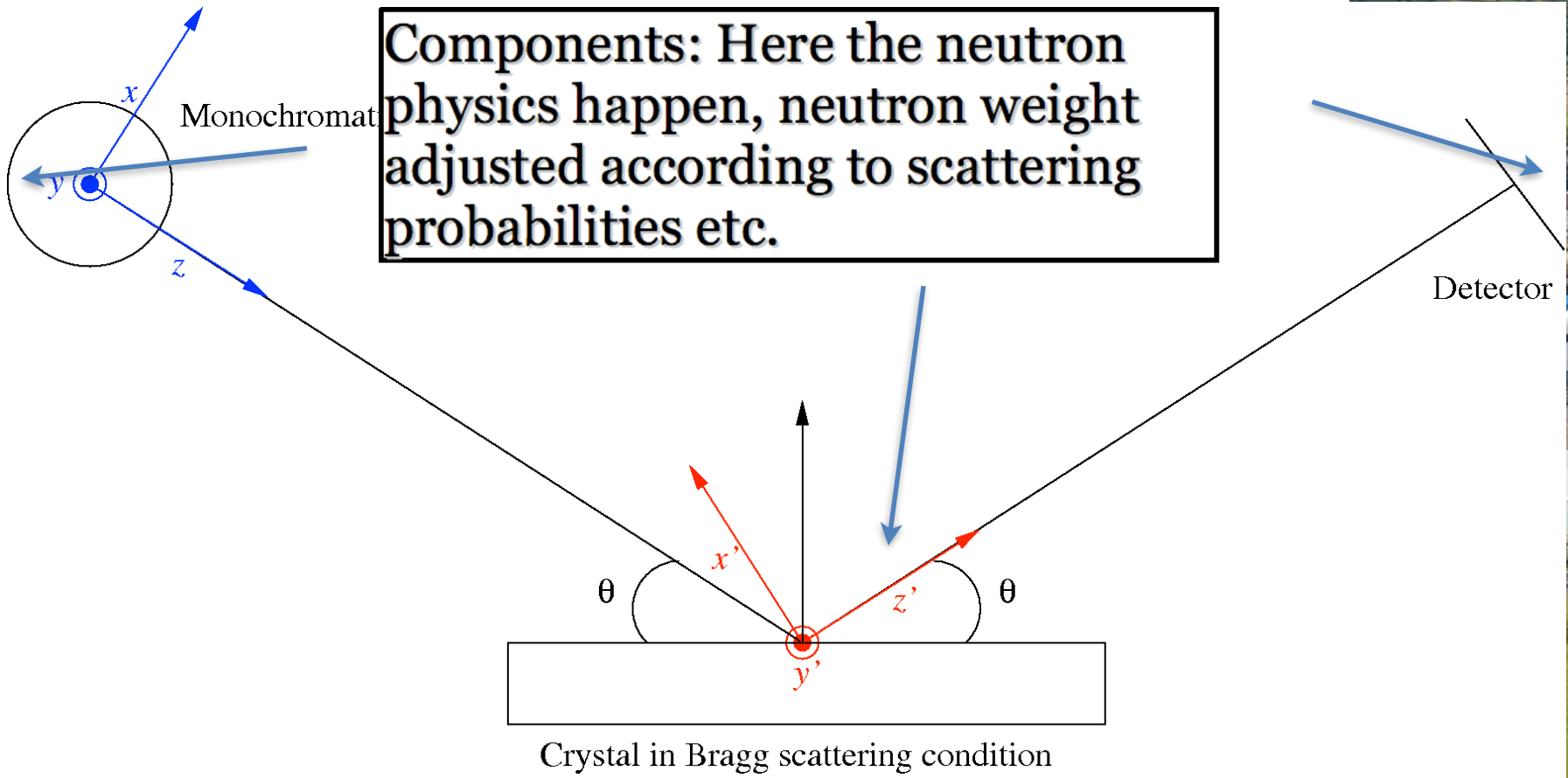
Neutron ray/package:
 Weight (p): # neutrons (left) in the package
 Coordinates (x,y,z)
 Velocity (v_x, v_y, v_z)
 Spin (s_x, s_y, s_z)
 Time (t)



Detector

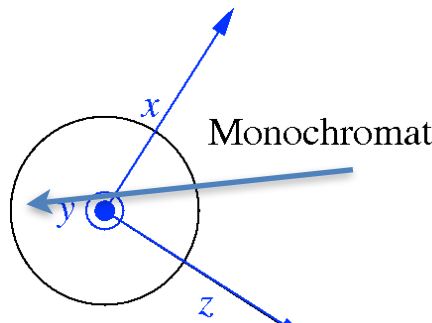


McStas: key concepts

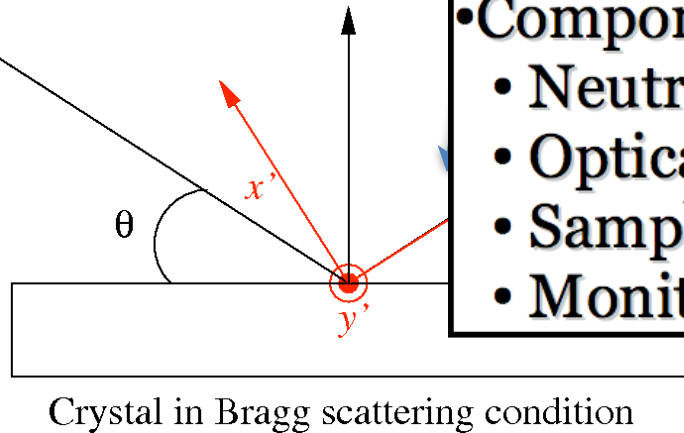




McStas: key concepts



Components: Here the neutron physics happen, neutron weight adjusted according to scattering probabilities etc.

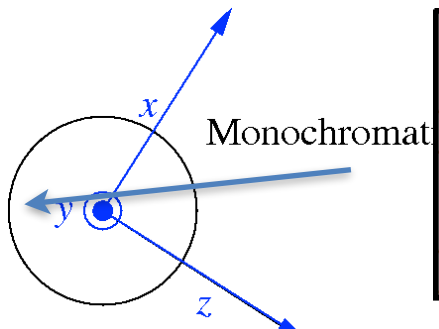
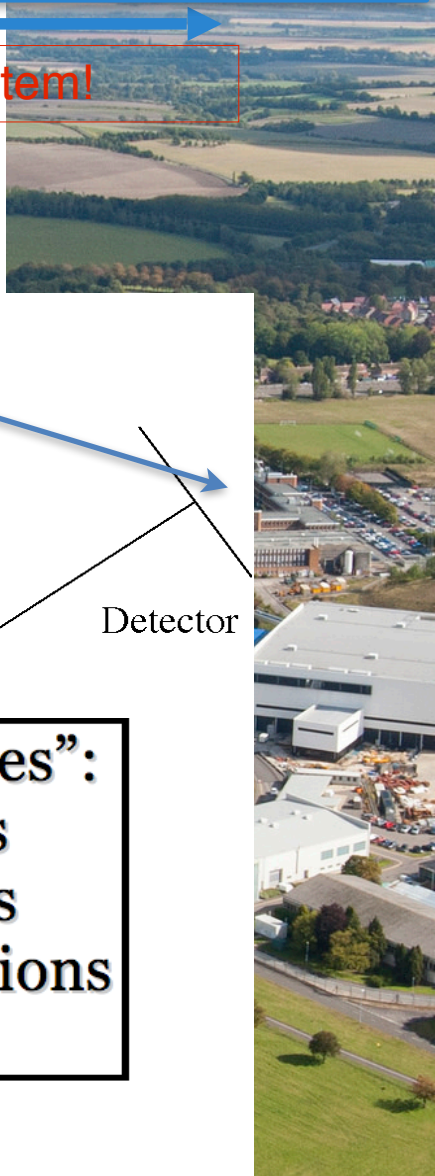


- Component “classes”:
- Neutron sources
- Optical elements
- Sample descriptions
- Monitors

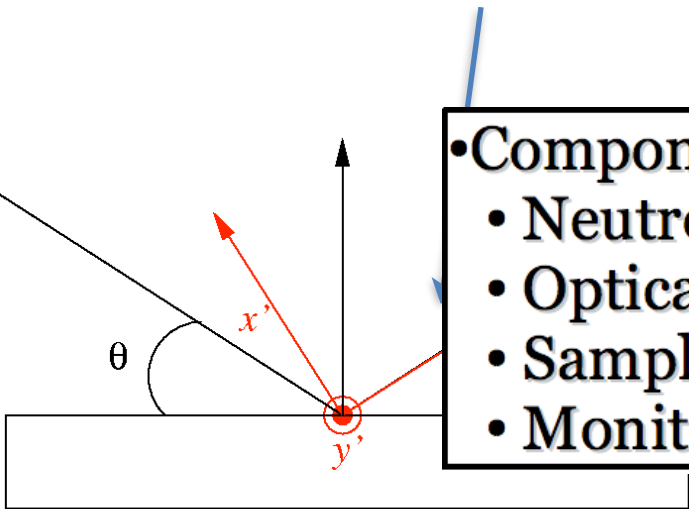
Detector

McStas: key concepts

Local, internal coordinate system!

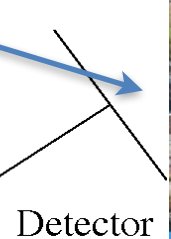


Components: Here the neutron physics happen, neutron weight adjusted according to scattering probabilities etc.



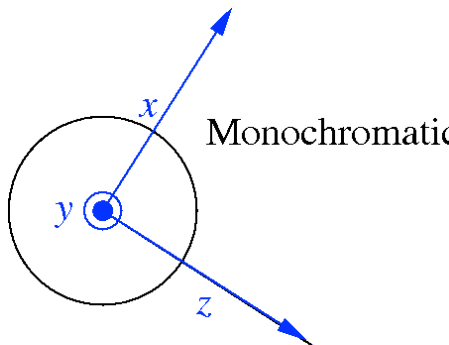
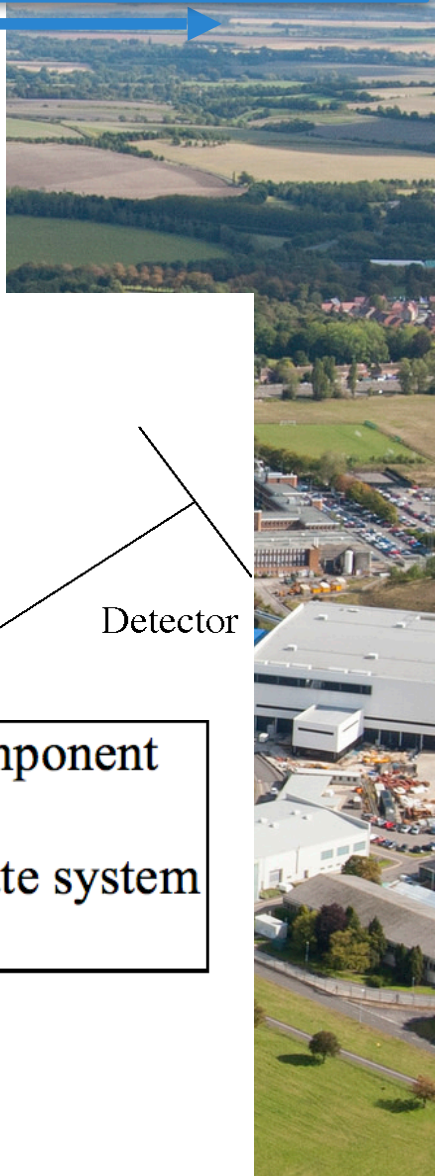
Crystal in Bragg scattering condition

- Component “classes”:
- Neutron sources
- Optical elements
- Sample descriptions
- Monitors



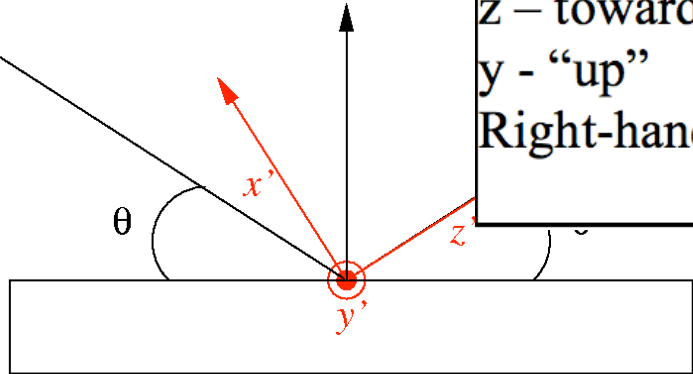
Detector

McStas: key concepts



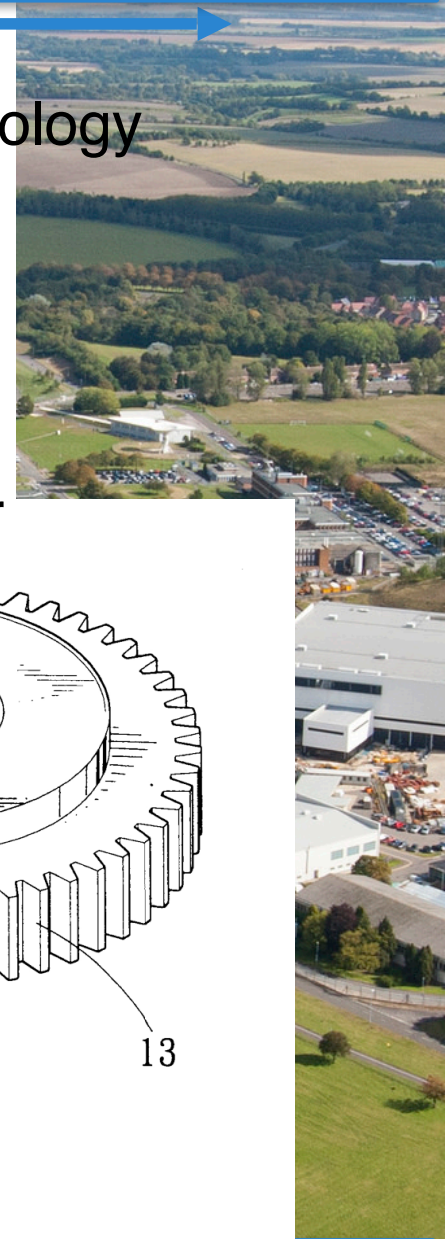
Instrument: positioning + transformation between sequential component coordinate systems, e.g. neutron source, crystal, detector.

z – towards “next” component
 y - “up”
 Right-handed coordinate system



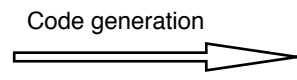
Crystal in Bragg scattering condition

Detector



Under-the-hood / inner workings

- Domain-specific-language (DSL) based on compiler technology (LeX+Yacc)

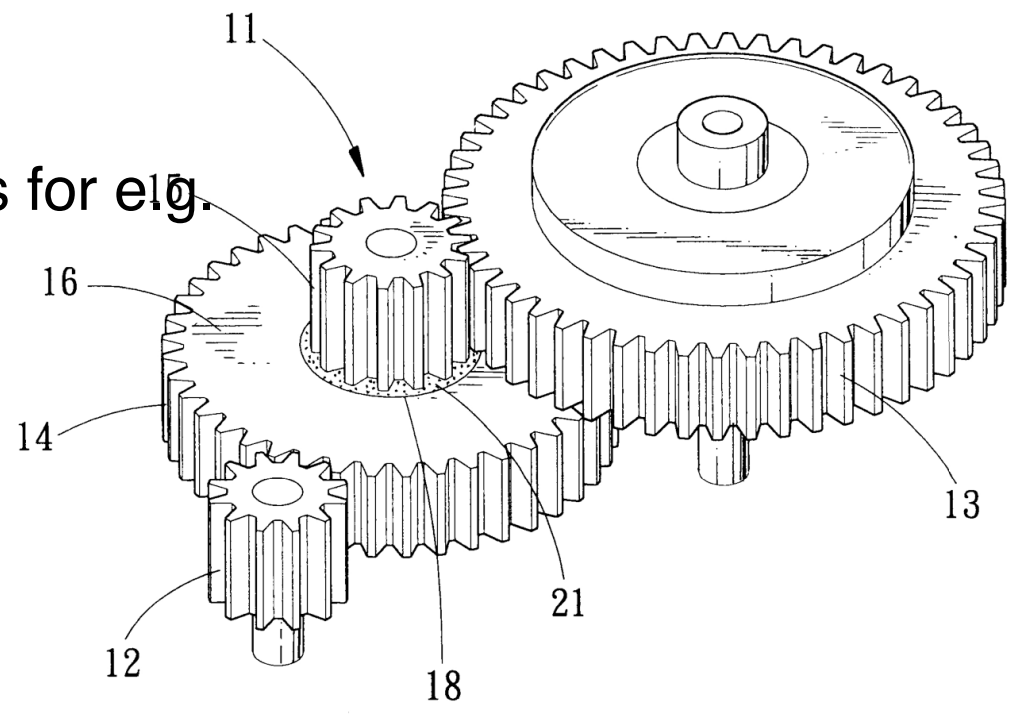


- Simple Instrument language ISO C

- Component codes realizing beamline parts (including user contribs)

- Library of common functions for e.g.

- I/O
- Random numbers
- Physical constants
- Propagation
- Precession in fields
- ...

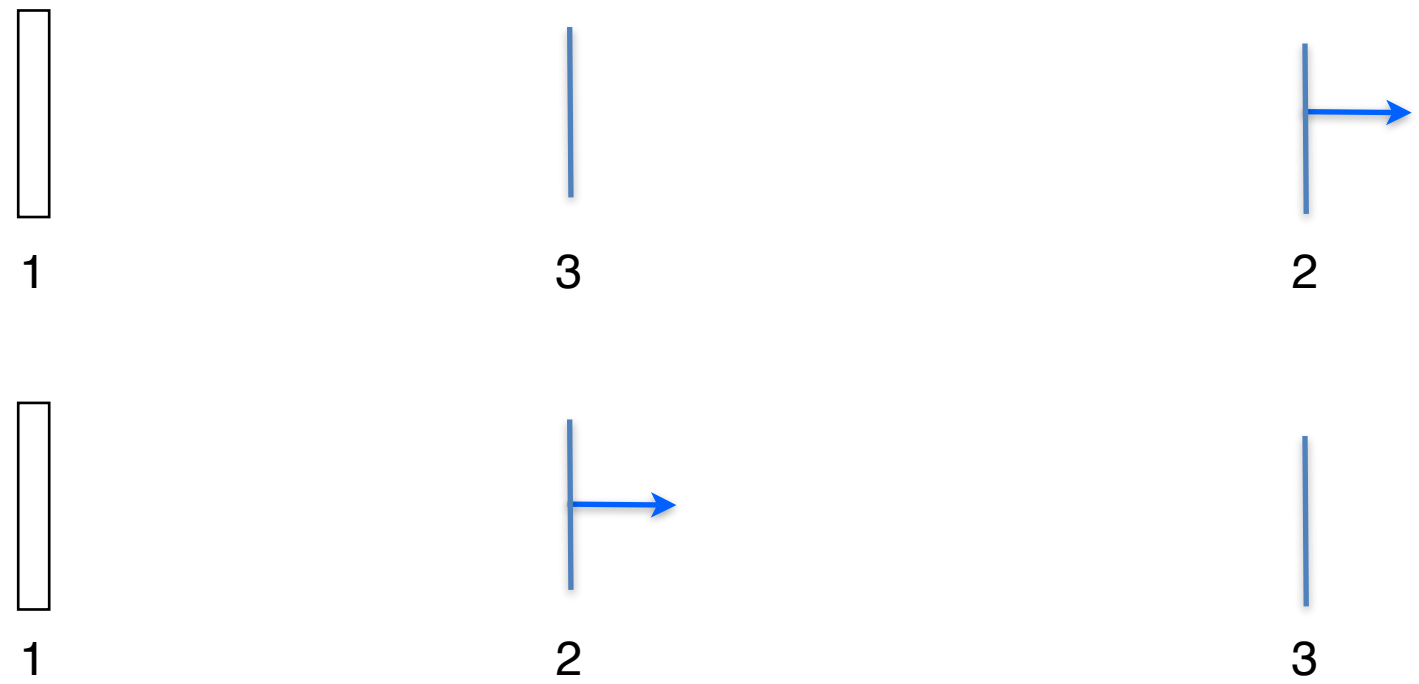


From the “list of frequent questions”

- Why do you need to have the components in this order?

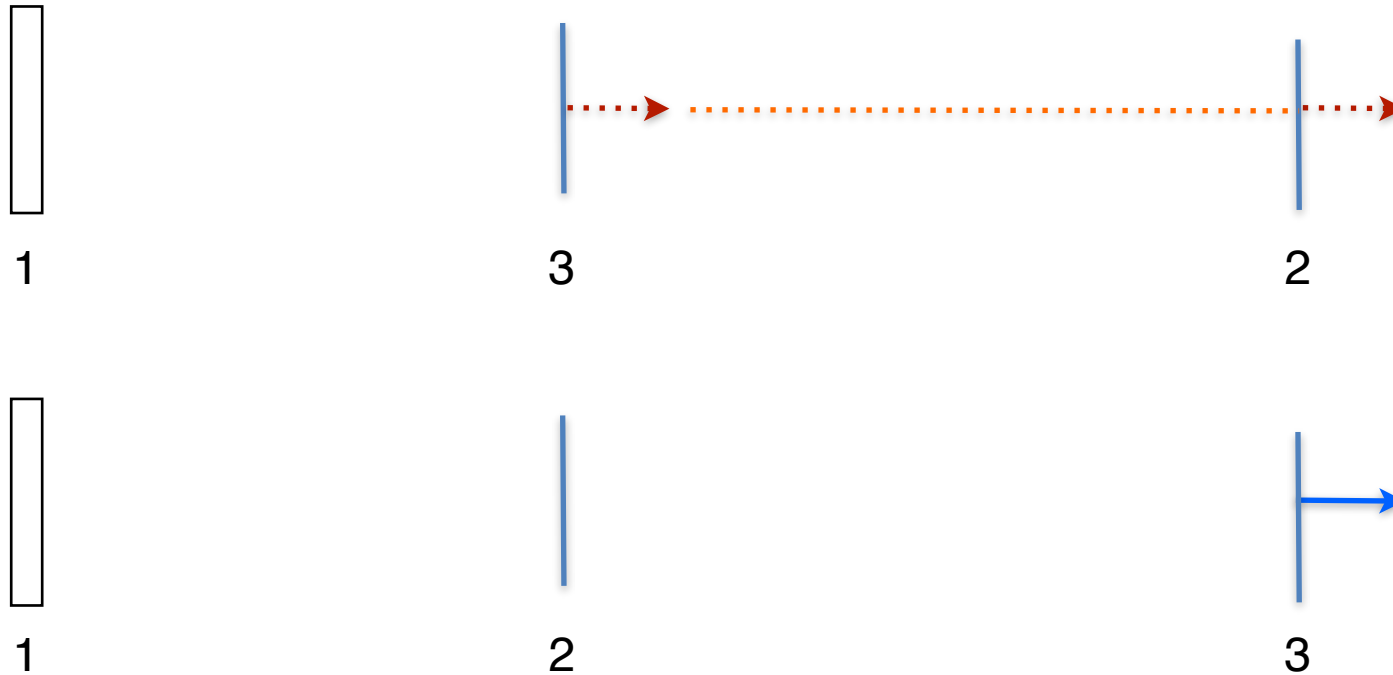


Order of components is important



Moving to first comp in the list

Order of components is important



Moving to 3rd comp in list requires “moving back in time”.
Default behavior is to ABSORB this type of neutron.
For monitors use `restore_neutron=1` in this case.
For homegrown comps use `ALLOW_BACKPROP` macro.



Further tips 1:



◆ Simulation to experiment comparison

◆

What is really the information content...?

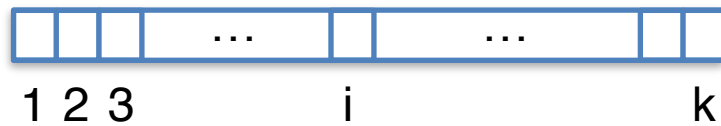


- ♦ McStas sources generally provide “intensity” in units of neutrons/s (into a chosen solid angle)
- ♦ That intensity is carried through the instrument on a discrete set of “neutron rays”



In a histogram sense

- Imagine a histogram, e.g. $I(\lambda)$



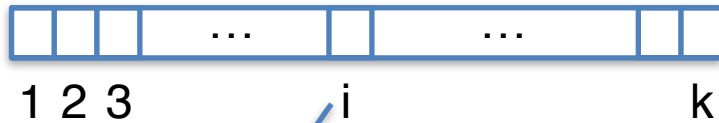
In bin i , N events each carrying a fractional intensity p_j so that

$$I = \sum_N p_j$$

- The RMS variance over that set becomes our statistical error bar E

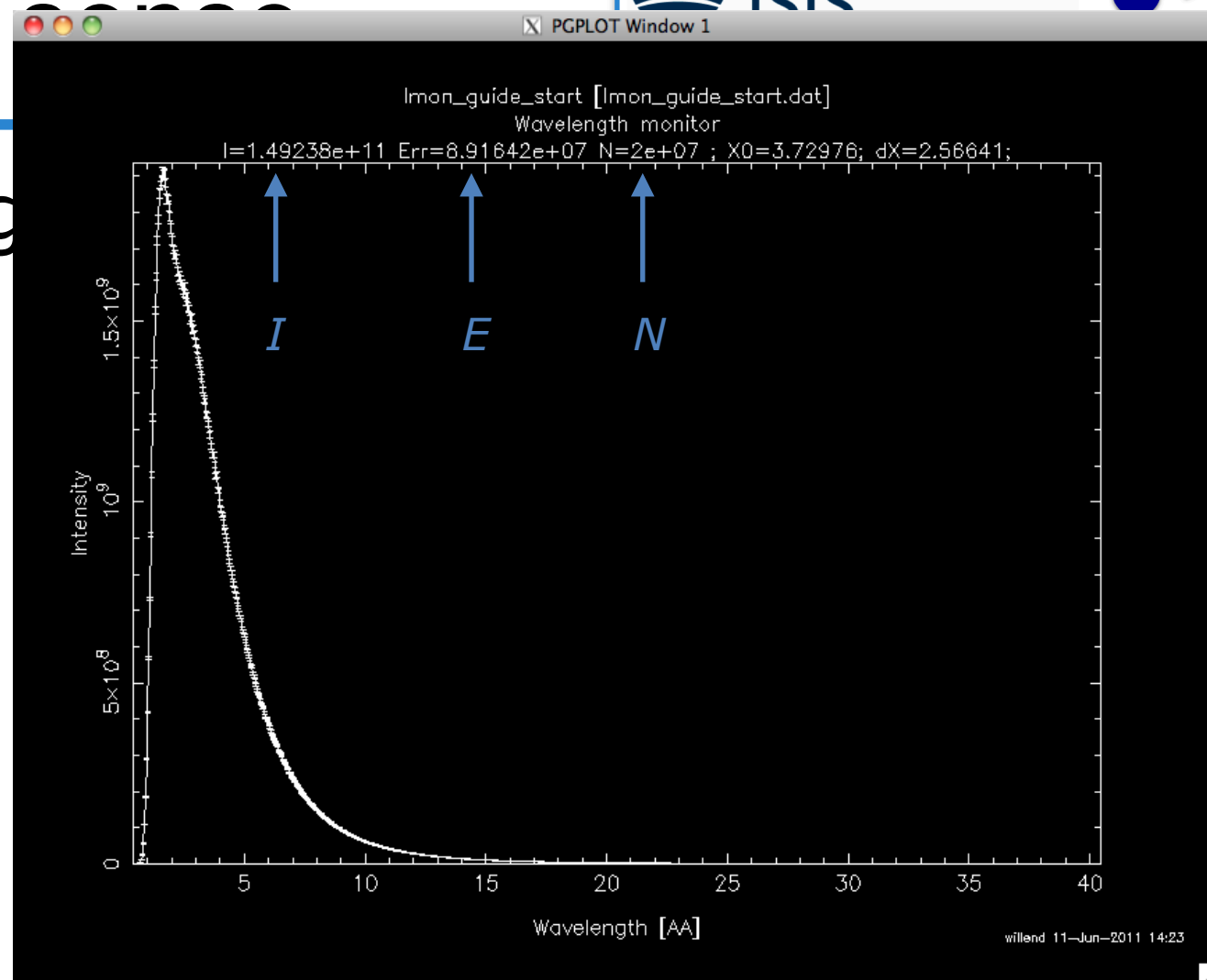
In a histogram

- Imagine a histogram



In bin i , N events each carrying a fractional intensity p_j so that

$$I = \sum_N p_j$$



- The RMS variance over that set becomes our statistical error bar E





From "Virtual experiments - the ultimate aim of neutron ray-tracing simulations", K. Lefmann et al., Journal of Neutron Research 16, 97-111 (2008)

Let n be the number of neutron rays reaching the detector, and let the rays have (different) weights, w_i . The simulated intensity is then given by

$$I = \sum_{i=1}^n w_i. \quad (1)$$

The estimate of the error on this number is calculated in the McStas manual [1], and the standard deviation is approximated by

$$\sigma^2(I) = \sum_{i=1}^n w_i^2. \quad (2)$$

In real experiments, $w_i = 1$, whence we reach $I = n$ and $\sigma(I) = \sqrt{I}$ as expected (for counts exceeding 10). Let the virtual time be denoted by t . The simulated counts during this time becomes

$$C = tI, \quad (3)$$



From "Virtual experiments - the ultimate aim of neutron ray-tracing simulations", K. Lefmann et al., Journal of Neutron Research 16, 97-111 (2008)

and its error bar estimate is

$$\sigma^2(C) = t^2 \sigma^2(I). \quad (4)$$

However, to simulate a realistic counting statistics, we must fulfill

$$\sigma_{\text{VE}}(C_{\text{VE}}) = \sqrt{C_{\text{VE}}}. \quad (5)$$

This is obtained by adding to (3) a Gaussian noise $E(\Sigma)$ of mean value zero and standard deviation Σ :

$$C_{\text{VE}} = tI + E(\Sigma). \quad (6)$$

The standard deviation for the VE becomes

$$\sigma_{\text{VE}}^2(C) = t^2 \sigma^2(I) + \Sigma^2. \quad (7)$$

Now, the requirement (5) allows us to determine Σ :

$$\Sigma^2 = tI - t^2 \sigma^2(I). \quad (8)$$

Since Σ^2 must remain positive, we reach an upper limit on t

$$t_{\text{max}} = \frac{I}{\sigma^2(I)}. \quad (9)$$



Sketch of an algorithm...

1. On a given McStas histogram
2. For the non-zero bins, calculate

$$t_{\max} = \frac{I}{\sigma^2(I)}.$$

3. The *smallest* t_{\max} defines the “maximal counting time” allowed by your statistics
4. Preferably a “background” should be added - use a “known experimental value” or an estimate...

Important points to remember

1. Your simulation will only contain elements you provided / defined
2. ... to the precision you defined
3. Answers the questions you posed
4. Background essentially only from "sample", or sample-near objects



Further tips 2:



- ♦ How to make your McStas more efficient

Onto efficiency...

- ◆ Apply focusing techniques
 - ◆ At the source (spatially, temporally, in wavelength...)
 - ◆ At the sample, if possible
- ◆ (carefully!) Apply SPLIT - but only if immediately followed by Monte Carlo choices, e.g. in sample
- ◆ Alternatively use MCPL o/i which allows repetition - beware of biases!



Onto efficiency...

- ◆ Apply focusing techniques
 - ◆ At the source (spatially, temporally, in wavelength...)
 - ◆ At the sample, if possible
- ◆ (carefully!) Apply SPLIT - but only if immediately followed by Monte Carlo choices, e.g. in sample
- ◆ Alternatively use MCPL o/i which allows repetition - beware of biases!

All of this can be considered "variance reduction" or biasing



Onto efficiency...

- ◆ Use MPI parallelisation - included in macOS install from 2.4, easy to get on Linux...
- ◆ The Intel C compiler is known to give ~factor of 2 wrt. gcc in most cases
- ◆ - Still consider if you are asking the right question if runtimes reach days/weeks...



Onto efficiency...

- ◆ Use MPI parallelisation - included in macOS install from 2.4, easy to get on Linux... (Windows: see later slide)
- ◆ The Intel C compiler is known to give ~factor of 2 wrt. gcc in most cases
- ◆ - Still consider if you are asking the right question if runtimes reach days/weeks...

Sledge-hammer / brute force!



Advanced language features

- Macros and tricks for your instrument...



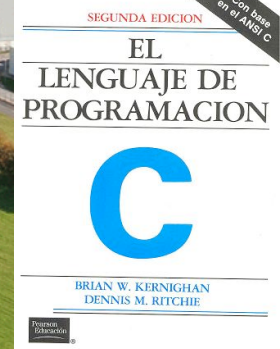
DECLARE / INITIALIZE

- | Use the DECLARE section define user variables and functions.
 - | DECLARE %{
 - | double myvar;
 - | %}

- | Use INITIALIZE for initialization of user variables and calculations.
 - | INITIALIZE %{
 - | myvar = sqrt(PI*input_var)*rand01();
 - | %}

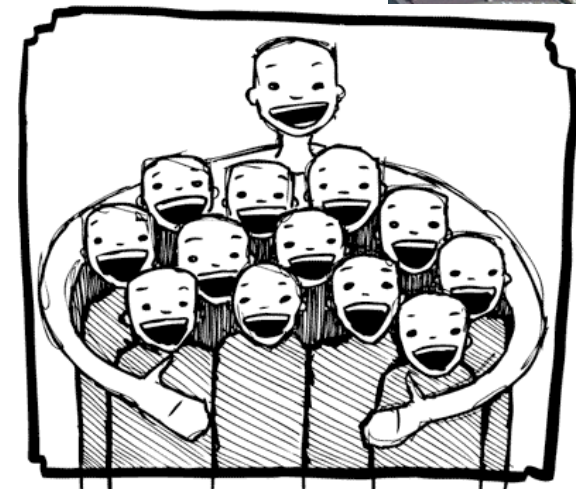
- | - Both use normal c-syntax.

- | **BEWARE:** (example) What you do in the c-style areas is c-standard, e.g. trigonometric functions from math.h use radians!
 - McStas placement specifiers work in degrees, etc...



%include

- | Instrumentfiles can include external c-code or other instrumentfiles... See the examples:
- | ILL_H15_IN6.instr:%include "monitor_nd-lib"
- | ILL_H16_IN5.instr:%include "ILL_H16.instr"
- | ILL_H25_IN22.instr:%include "ILL_H25.instr"
- | ILL_H25_IN22.instr:%include "templateTAS.instr"
- | Used in the DECLARE section



Syntax in one, complex view...



```
{SPLIT} COMPONENT name = comp(parameters) {WHEN condition}  
  AT (...) [RELATIVE [reference|PREVIOUS] | ABSOLUTE]  
  {ROTATED {RELATIVE [reference|PREVIOUS] | ABSOLUTE} }  
  {GROUP group_name}  
  {EXTEND C_code}  
  {JUMP [reference|PREVIOUS|MYSELF|NEXT] [ITERATE number_of_times | WHEN condition] }
```



SPLIT

- | Increase statistics beyond this point in the instrumentfile

- | SPLIT n MyArm = Arm()

- | AT somewhere

- | will “formulate an if-statement”:

- | for j=1:n

- | comp1

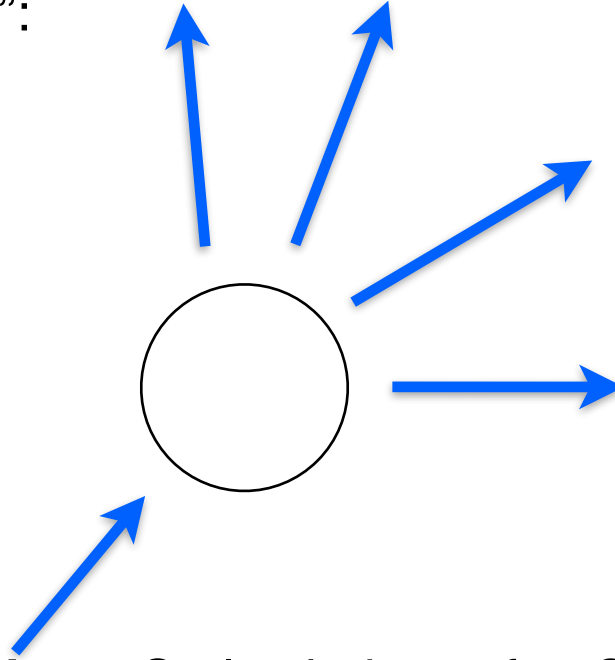
- | comp2

- | comp3

- | ...

- | end (of instrument)

- | ONLY meaningful in case of Monte Carlo choices after SPLIT point...



SPLIT

- | Increase statistics beyond this point in the instrumentfile

- | SPLIT n MyArm = Arm()

- | AT somewhere

- | will “formulate an if-statement”:

- | for j=1:n

- | comp1

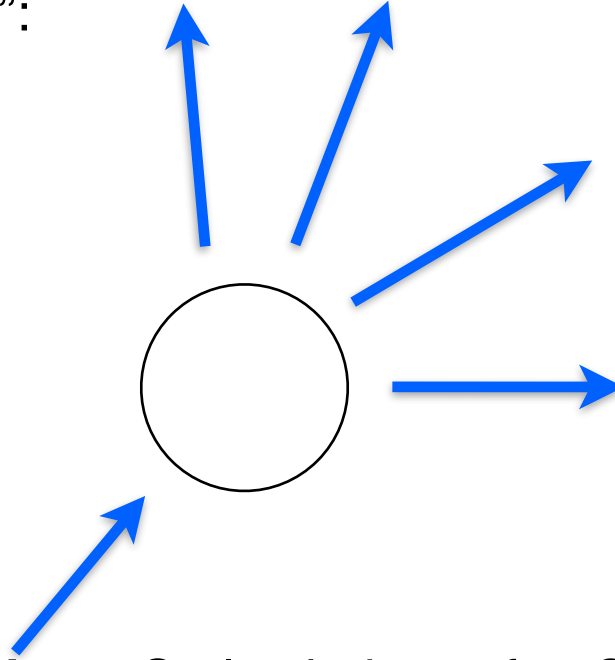
- | comp2

- | comp3

- | ...

- | end (of instrument)

- | ONLY meaningful in case of Monte Carlo choices after SPLIT point...



slight sidetrack....

Problem: McStas Single_crystal.comp “slow” for large unit cell diffraction studies

- Example: Rubredoxin

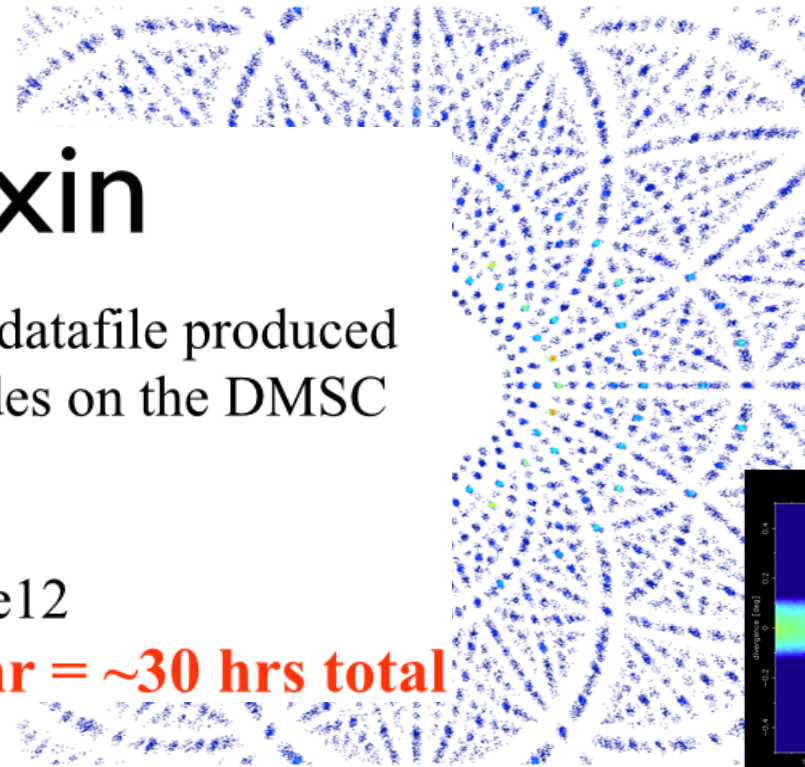
1 timebin, 1000 x,y-bins

Rubredoxin

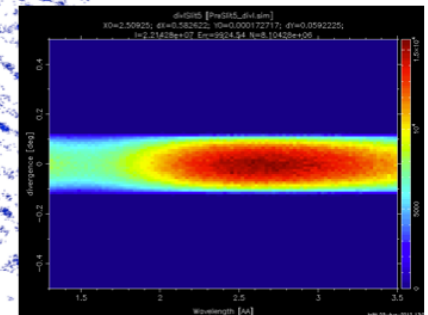
Images created from simulated datafile produced
August 20th 2012 using 25 nodes on the DMSC
cluster.

Neutron count: 1e12

Simulation time: ~10 + ~20 hr = ~30 hrs total



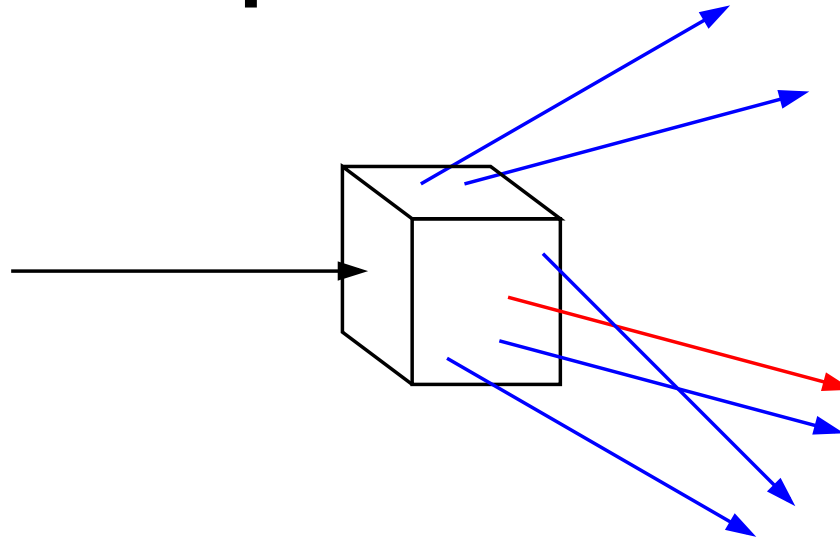
Neutroncount: 1e12
No gravitation
Xtal size: 0.5 mm
Xtal mosaicity: 12'
Detector: 50 x 50 cm flat
Detector-to-sample
distance: 20 cm
Guide length: 131 m
Guide dimensions: 9.5 cm
 $\lambda_{\min} = 1.3 \text{ \AA}$
 $\lambda_{\max} = 3.5 \text{ \AA}$
Timespan: 51.39 to 143.4 ms
Divergence = 0.2 degs



- Reflection list ~ 124 K reflections (still “small” in the PX world!!)

slight sidetrack....

Algorithm improvement: **Use incoming neutrons more efficiently - scatter each one on all possible reflections**



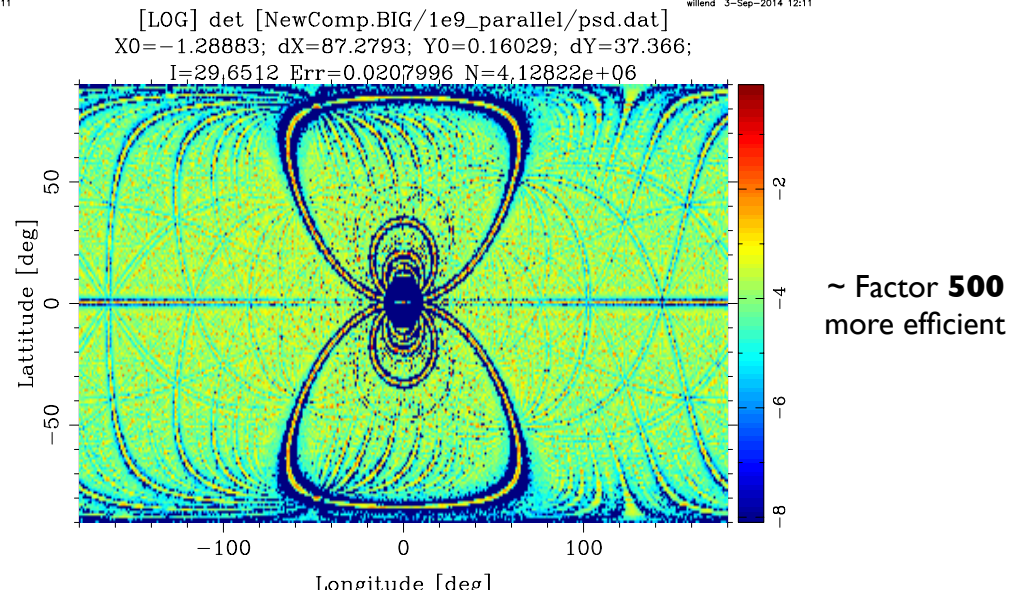
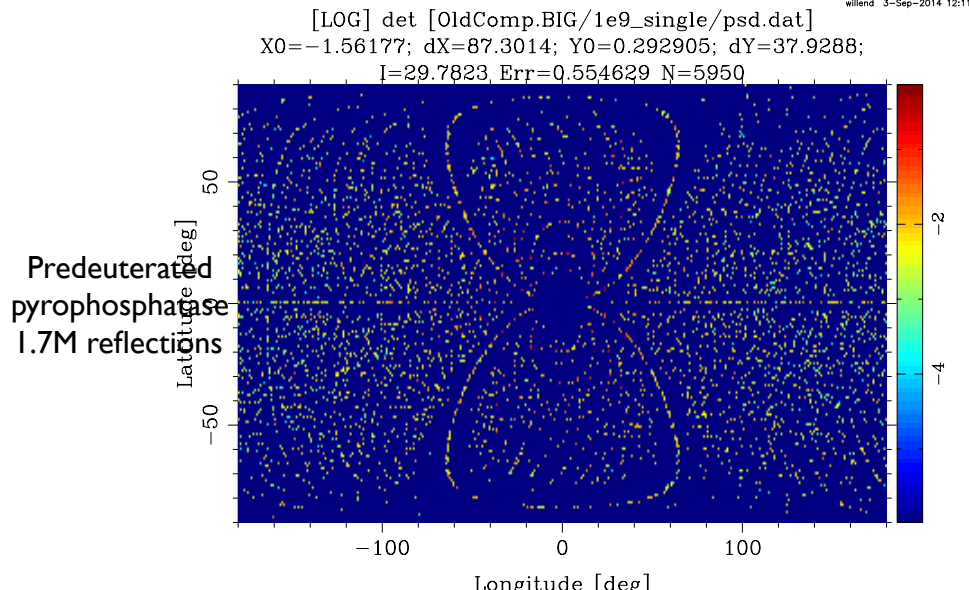
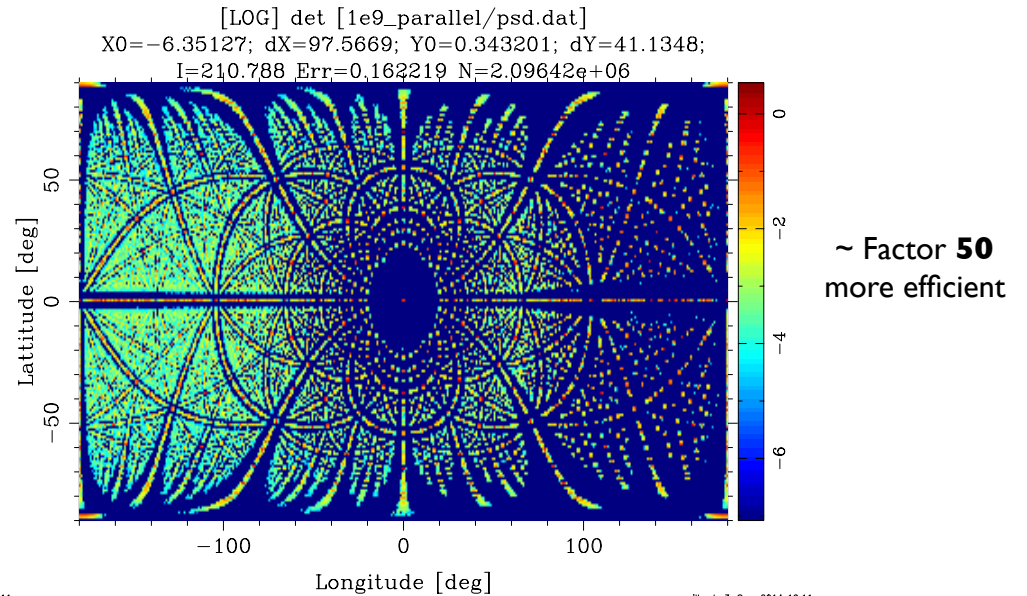
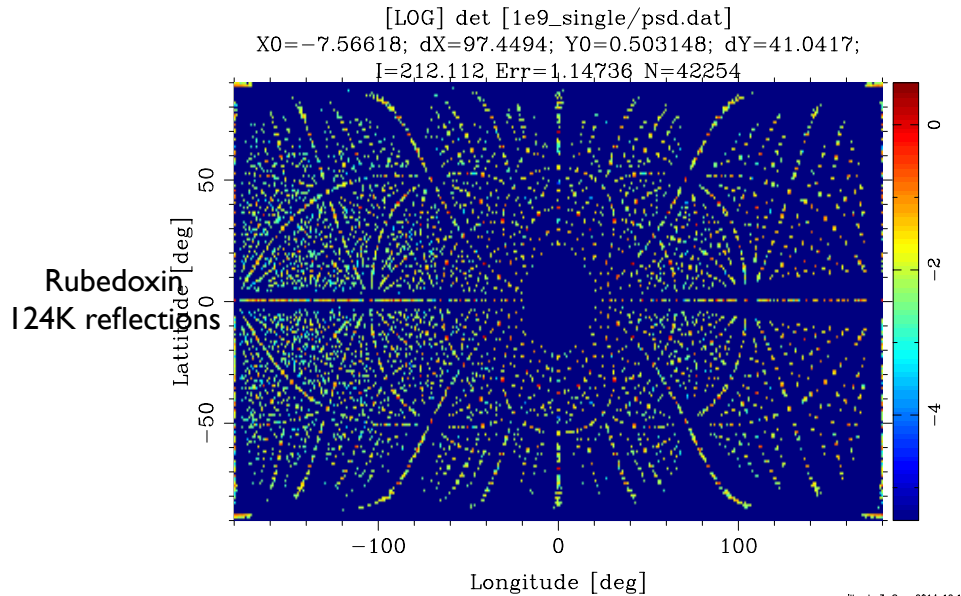
- **Red**: Original algorithm, one incoming neutron used only once
- **Blue**: Improved algorithm, each incoming neutron scattered (via SPLIT keyword) all possible times
- Component makes **estimate on average number of “active” diffraction spots** - in the case Rubredoxin this is around **50!**

slight sidetrack....

Sim data speak for themselves - 1e9 rays

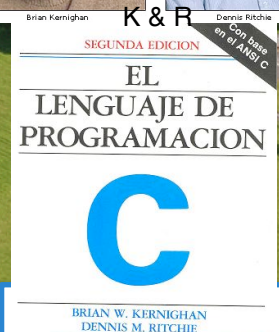
Old comp

New comp



WHEN

- | Syntax:
- | COMPONENT Mine = Yours(blah, blah)
- | WHEN (c-expression) AT (....)
- | Is very powerful when combined with EXTEND and user variables, or as a method to let input parameters select if certain components are active.
- | Example: Use EXTEND to flag if neutron was scattered on one monochromator blade or another. Then later use WHEN to only show contribution from blade N at sample position?
- | COMPONENT Mon = PSD_monitor(...)
- | WHEN (myvar==1) AT (0,0,0) RELATIVE Sample



GROUP - components working in parallel

```
COMPONENT Mono1 = Monochromator_curved(...)  
AT (0,0, -LMM) RELATIVE Cradle ROTATED (0,A1/2,0) RELATIVE Cradle  
GROUP IN6Monoks
```

```
COMPONENT Mono2 = Monochromator_curved(...)  
AT (0,0, 0) RELATIVE Cradle ROTATED (0,A2/2,0) RELATIVE Cradle  
GROUP IN6Monoks
```

- One comp after the other is “tried” in sequential order until the neutron was SCATTERED.



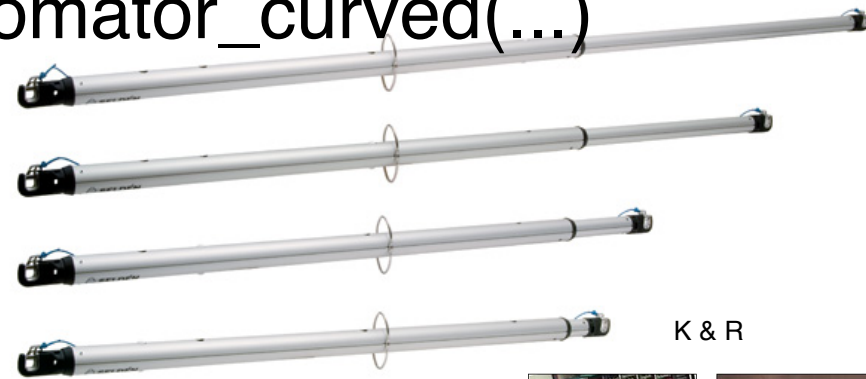
EXTEND



Enrich component behaviour using EXTEND:

COMPONENT Mono1 = Monochromator_curved(...)

```
AT (0,0, -LMM) RELATIVE Cradle ROTATED (0,A1/2,0) RELATIVE Cradle
GROUP IN6Monoks
EXTEND
%{
if (SCATTERED) { myvar = ;1 }
%}
```



...

COMPONENT Mono2 = Monochromator_curved(...)

```
AT (0,0, 0) RELATIVE Cradle ROTATED (0,A2/2,0) RELATIVE Cradle
GROUP IN6Monoks
%{
if (SCATTERED) { myvar = 2 }
%}
```



JUMP

- | A goto. Be careful. Can be used in two situations:
- | JUMP to myself
- | JUMP to an Arm

- | No coordinate transformations are applied... (Meaning that if the Arms you JUMP between do not coincide you will “move” / “reorient” the neutrons...)

- | Syntaxes:
- | COMPONENT a=b(...)
- | WHEN (expr) AT (...) JUMP somewhere

- | COMPONENT a=b(...)
- | WHEN (expr) AT (...) JUMP somewhere



JUMP

- | A goto. Be careful. Can be used in two situations:
- | JUMP to myself

BEWARE - This IS a GOTO!

- | No coordinate transformations are applied... (Meaning that if the Arms you JUMP between do not coincide you will “move” / “reorient” the neutrons...)

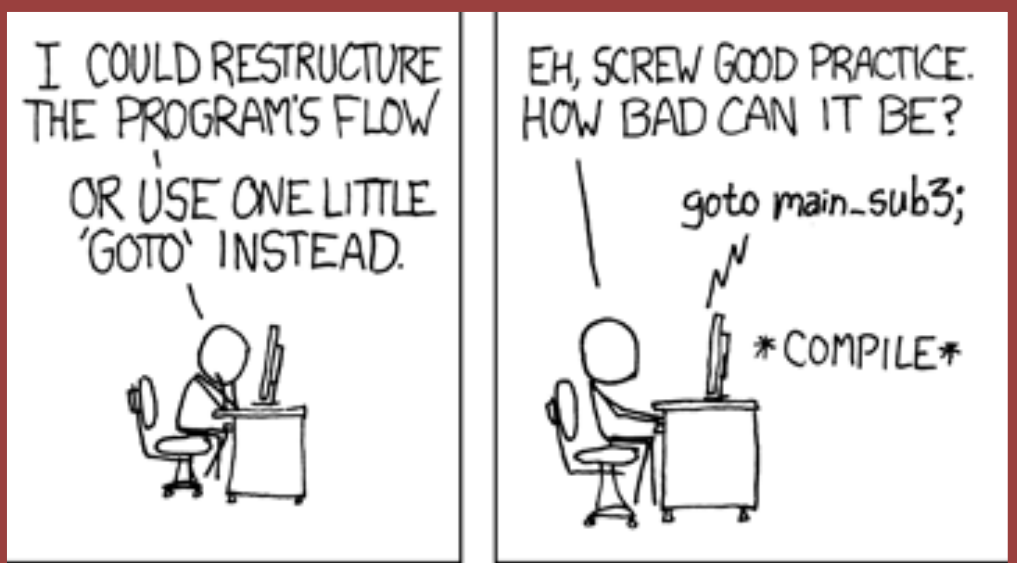
- | Syntaxes:
- | COMPONENT a=b(...)
- | WHEN (expr) AT (...) JUMP somewhere
- | COMPONENT a=b(...)
- | WHEN (expr) AT (...) JUMP somewhere



JUMP

A goto. Be careful. Can be used in two situations:

BEWARE!

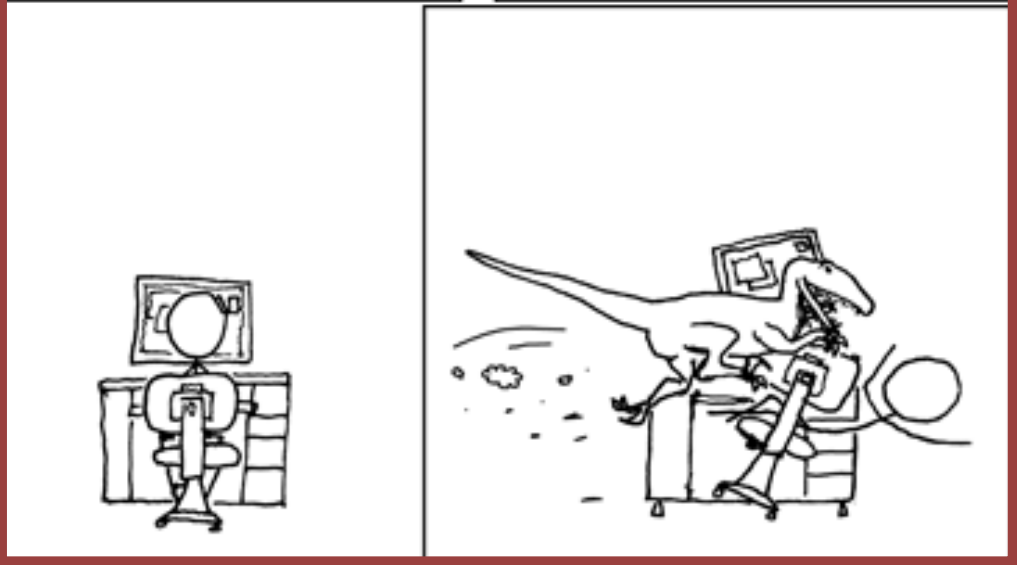


BOOM!

Meaning that if the will "move" /

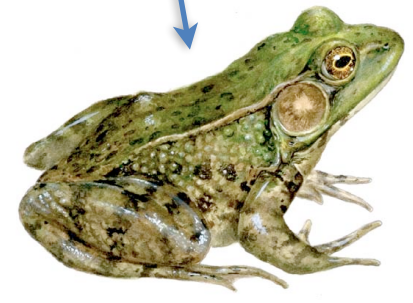
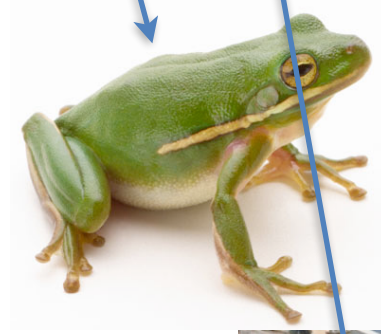
No coordi
Arms you
"reorient"

Syntaxes
COMPON
WHEN (e
COMPON
WHEN (e



COPY- inside instruments

Technology Facilities Council **McStas**
 DTU, ESS, EUROPEAN SPALLATION SOURCE
 Training April 26th-28th 2017



In instruments: (see ILL_H25.instr)

COMPONENT H25_1 = Guide_gravity(
 w1=0.03, h1=0.2, w2=0.03, h2=0.2, l=L_H25_1,
 R0=gR0, Qc=gQc, alpha=gAlpha, m=m, W=gW)
 AT (0,0,Al_Thickness+gGap) RELATIVE PREVIOUS
 ROTATED (0,Rh_H25_1,0) RELATIVE PREVIOUS

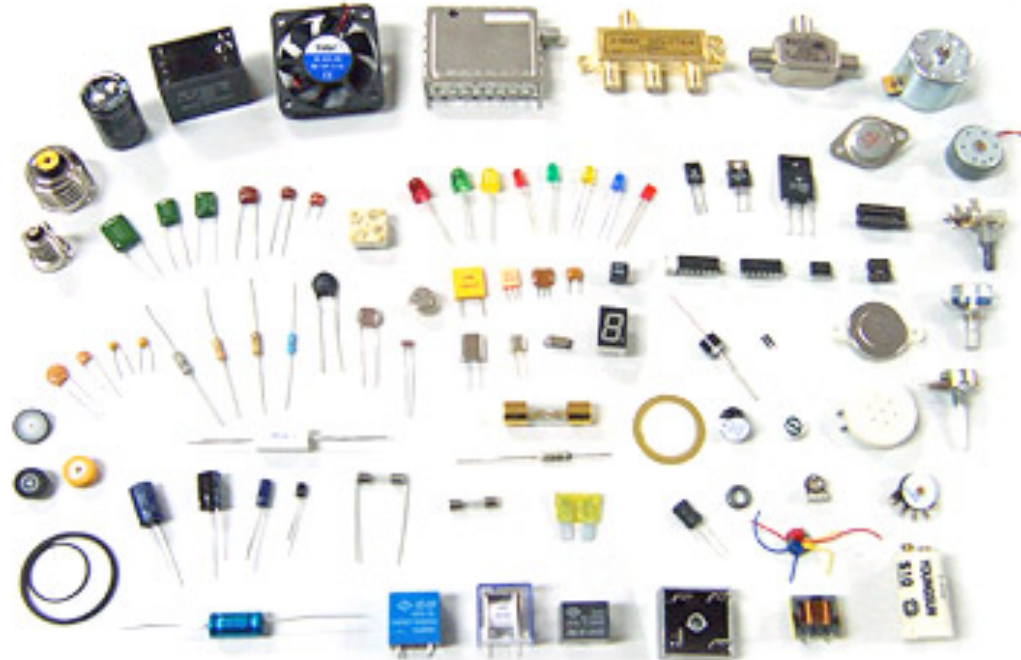
COMPONENT COPY(H25_1) = COPY(H25_1)
 AT (0,0,L_H25_1+gGap) RELATIVE PREVIOUS
 ROTATED (0,Rh_H25_1,0) RELATIVE PREVIOUS

COMPONENT COPY(H25_1) = COPY(H25_1)(W=2*gW)
 AT (0,0,L_H25_1+gGap) RELATIVE PREVIOUS
 ROTATED (0,Rh_H25_1,0) RELATIVE PREVIOUS

DTU, ESS EUROPEAN SPALLATION SOURCE

Other advanced topics

- Tricks, macros and functions for your components



COPY In components:



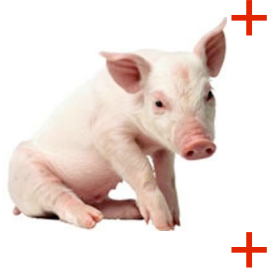
There is a heritage mechanism to create childs of existing components. These are exact duplicates of the parent component, but one may override/extend original definitions of any section.

The syntax for a full component child is

```
DEFINE COMPONENT child_name COPY parent_name
```

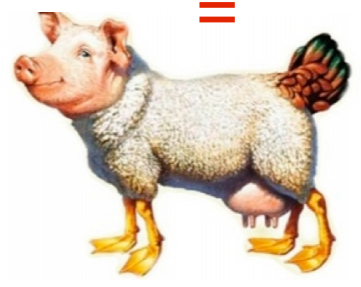
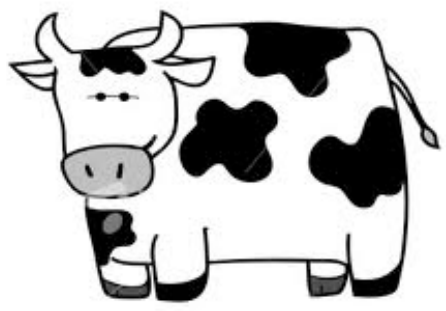
This single line will copy all parts of the *parent* into the *child*, except for the documentation header.

As for normal component definitions, you may add other parameters, DECLARE, TRACE, sections. Each of them will replace or extend (be catenated to, with the COPY/EXTEND words, see example below) the corresponding *parent* definition. In practice, you could copy a component and only rewrite some of it, as in the following example:



```
DEFINE COMPONENT child_name COPY parent_name
```

```
SETTING PARAMETERS (newpar1, newpar2)
INITIALIZE COPY parent_name EXTEND
%{
    ... C code to be catenated to the parent_name INITIALIZE ...
%}
SAVE
%{
    ... C code to replace the parent_name SAVE ...
%}
```





MPI on Windows

- ♦ Get both of the installers from
 - ♦ <https://github.com/McStasMcXtrace/McCode/tree/master/support/Win32/MSMPI>
 - ♦ - install them
- ♦ Dump this batchfile in bin dir of your McStas installation:
 - ♦ <https://github.com/McStasMcXtrace/McCode/blob/master/cmake/support/run-scripts/mpicc.bat>
- ♦ Add C:\Program Files\Microsoft MPI\Bin\ to your system PATH



NeXus libs on Windows

- Get last release from
 - <https://github.com/nexusformat/code/releases?after=4.3.1>
 - Install it
- Use the MCSTAS_CFLAGS_OVERRIDE or the GUI File->Configuration entry to specify -I and -L directives for linking
- i.e.

```
-DUSE_NEXUS -lNeXus-0  
-L"C:\Program Files (x86)\NeXus Data Format\bin"  
-L"C:\Program Files (x86)\NeXus Data Format\lib\nexus"  
-I"C:\Program Files (x86)\NeXus Data Format\include"
```



NeXus libs on macOS

- Get homebrew via <https://brew.sh>
- Install it via the recipe given
- In a terminal, issue

```
DTUs-MacBook-Pro:Downloads dtuphysics$ brew search nexusformat  
homebrew/science/nexusformat ✓  
DTUs-MacBook-Pro:Downloads dtuphysics$ brew install homebrew/science/nexusformat
```

- Use these CFLAGS either via env var or gui

```
-DUSE_NEXUS -lNeXus -L/usr/local/lib -I/usr/local/include
```

The end

