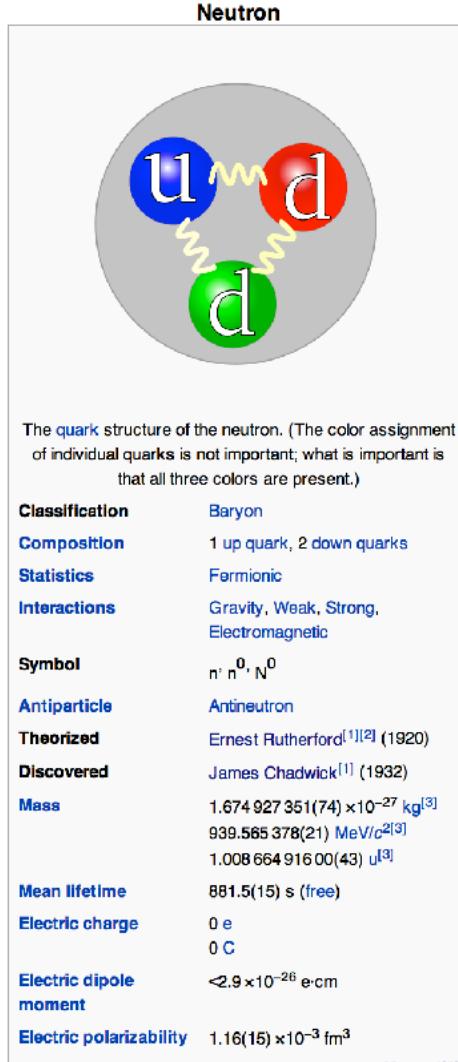


McStas Introduction and general concepts

Agenda

- A (very) brief introduction to McStas
- Components of neutron instruments
- How McStas works under the hood
- Components and instruments
- A demo

McStas: Neutrons, Monte Carlo & ray-tracing



Life time: $\tau_{1/2} = 890s$
 Mass: $m = 1.675 \times 10^{-27} \text{ kg}$
 Charge: $Q = 0$
 Spin: $s = \hbar/2$
 Magnetic moment: $\mu/\mu_n = -1.913$

$$E = \frac{1}{2}mv^2 = \frac{\hbar^2k^2}{2m}, \quad \lambda = 2\pi/k$$

$$E = 81.81 \cdot \lambda^{-2} = 2.07 \cdot k^2 = 5.23 \cdot v^2$$

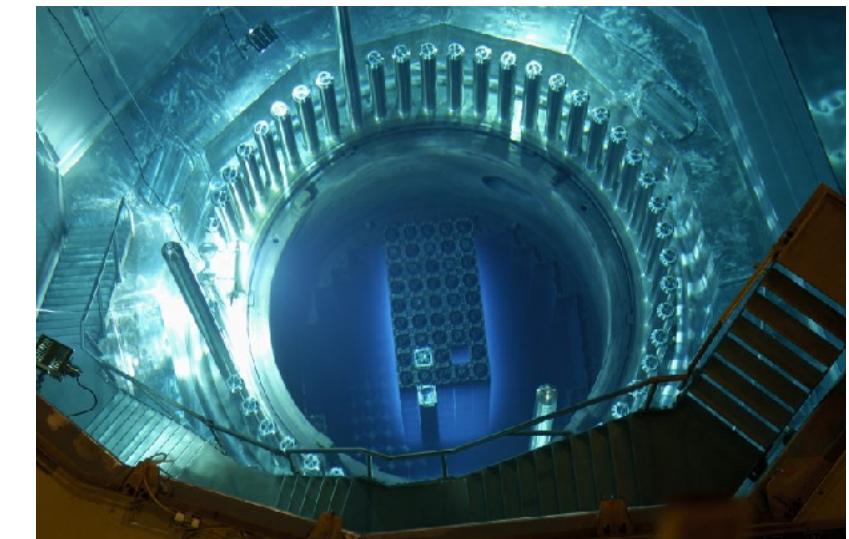
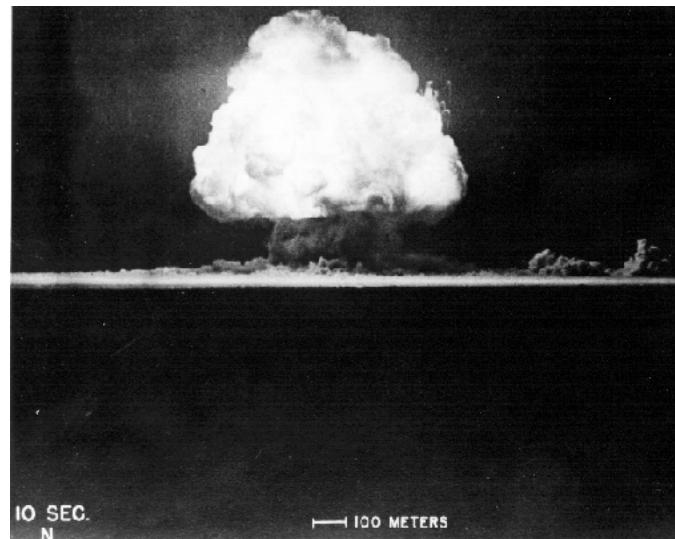
Subatomic particle discovered by Sir James Chadwick in 1932



	Energy	Wavelength	n-Wavevector	Velocity	Frequency
cold neutrons:	$E = 1 \text{ meV}$ $E = 5 \text{ meV}$	$\lambda = 9.0446 \text{ \AA}$ $\lambda = 4.0449 \text{ \AA}$	$k = 0.6947 \text{ 1/\AA}$ $k = 1.5534 \text{ 1/\AA}$	$v = 437 \text{ m/s}$ $v = 978 \text{ m/s}$	$v = 0.2418 \text{ THz}$ $v = 1.2090 \text{ THz}$
thermal neutrons:	$E = 25 \text{ meV}$ $E = 50 \text{ meV}$	$\lambda = 1.8089 \text{ \AA}$ $\lambda = 1.2791 \text{ \AA}$	$k = 3.4734 \text{ 1/\AA}$ $k = 4.9122 \text{ 1/\AA}$	$v = 2187 \text{ m/s}$ $v = 3093 \text{ m/s}$	$v = 6.045 \text{ THz}$ $v = 12.090 \text{ THz}$

Monte Carlo techniques

- Los Alamos has since then developed and perfected many different monte carlo codes leading to what is today known as the codes MCNP5 and MCNPX
- State of the art is MCNP6 that features numerous (even exotic) particles
- MCNP was originally Monte Carlo Neutron Photon, later N-Particle
- Mainly used for high-energy particle descriptions in weapons, power reactors and routinely used for estimating dose rates and needed shielding
- Not much focus on crystalline / ordered material and coherent scattering of neutrons due to the focus on high energies



Examples of Monte Carlo programs, ...)

random choices are made.

Light ray-tracing: PoV-RAY and others ...

*Nuclear reactor simulations etc. (general particle transport):
MCNP, Tripoli, GEANT4, FLUKA*

Neutron Monte Carlo ray-tracing propagation:

McStas <www.mcstas.org>, Vitess, Restrax, NISP, IDEAS, McVine

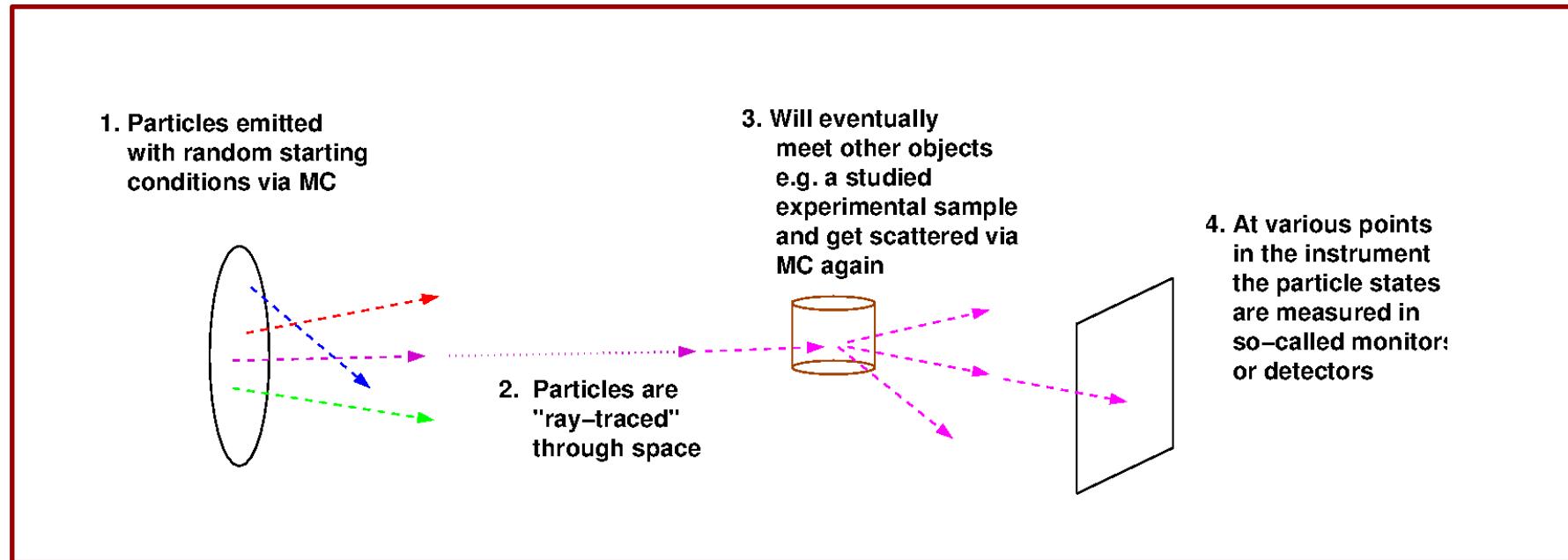
Neutrons are described as (\mathbf{r} , \mathbf{v} , \mathbf{s} , t), and are transported along instrument models.

Propagation simply uses Newton rules, incl. gravitation.

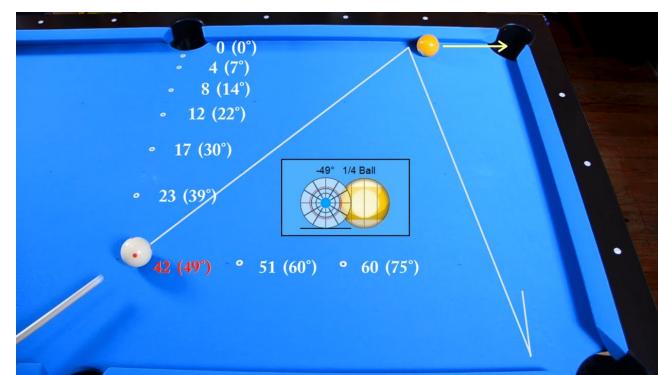
X-ray tracing

Shadow, McXtrace, RAY, ...

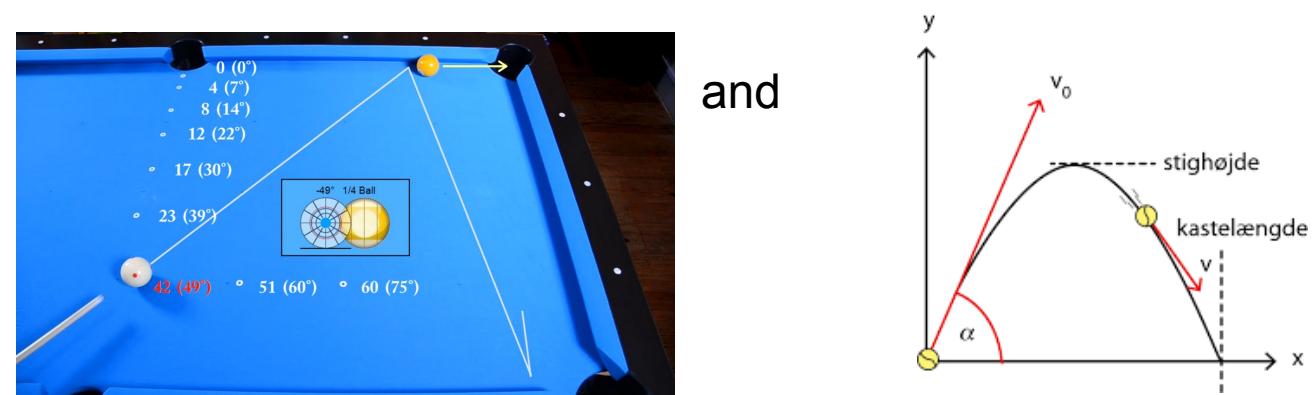
In the big picture, McStas is this...



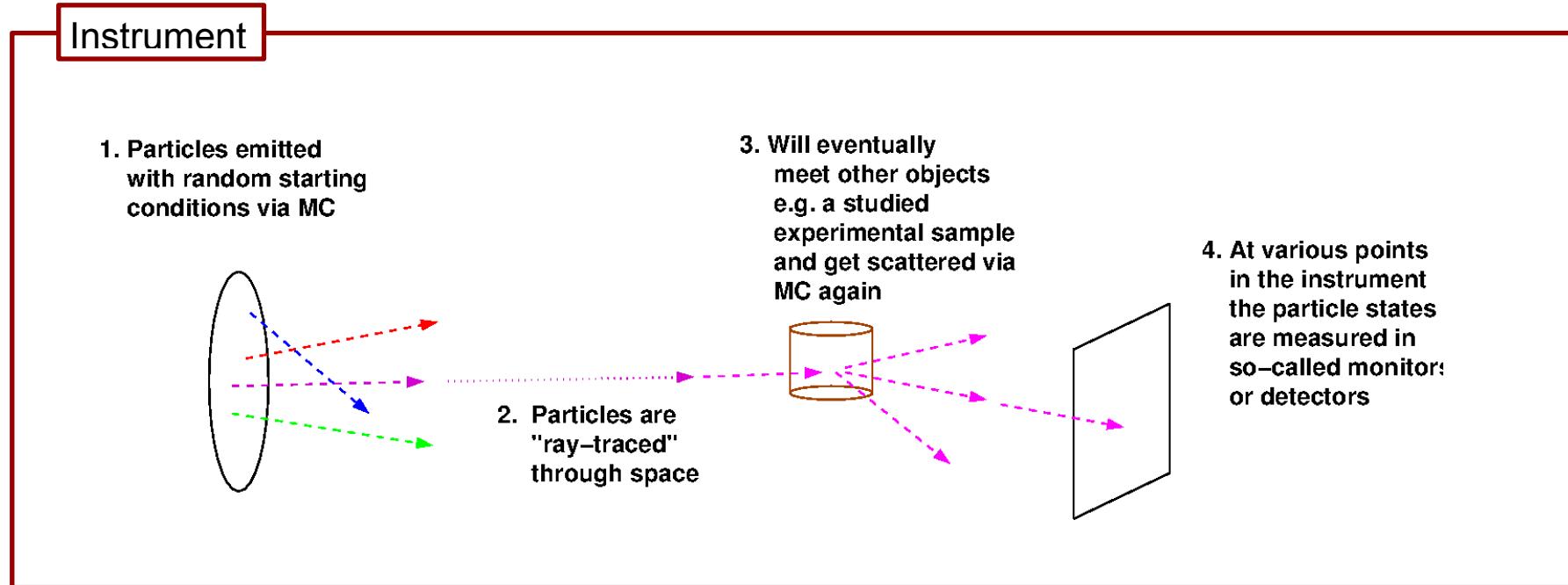
- Classical Newtonian mechanics, i.e.
- (independent, particles though...)



and

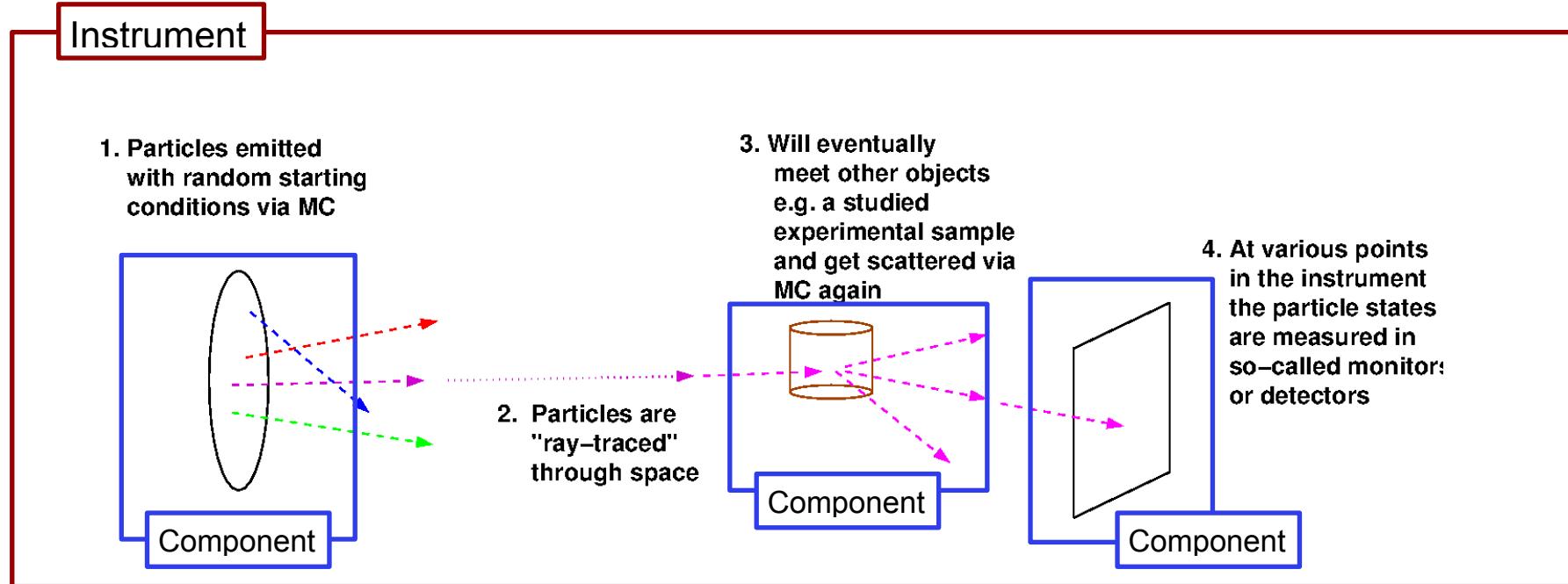


In the big picture, McStas is this...



The instrument defines our “lab coordinate system”

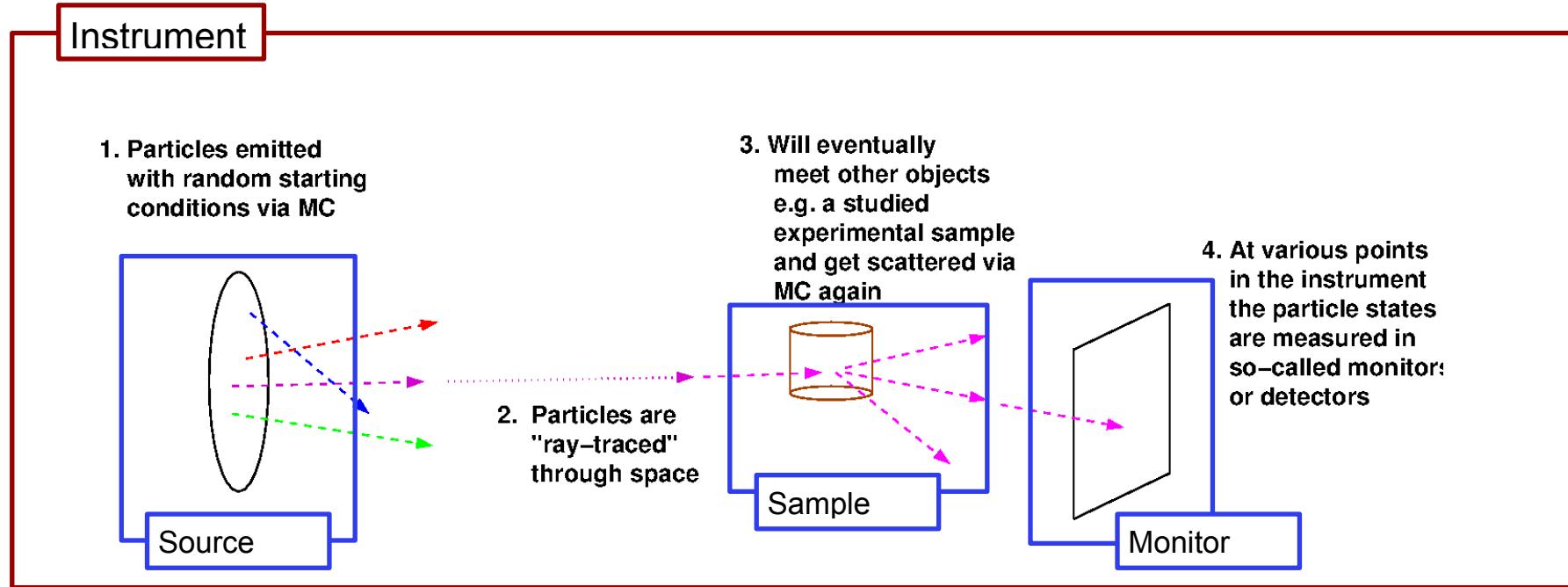
In the big picture, McStas is this...



The instrument defines our “lab coordinate system”

The components define devices or features available in our instrument

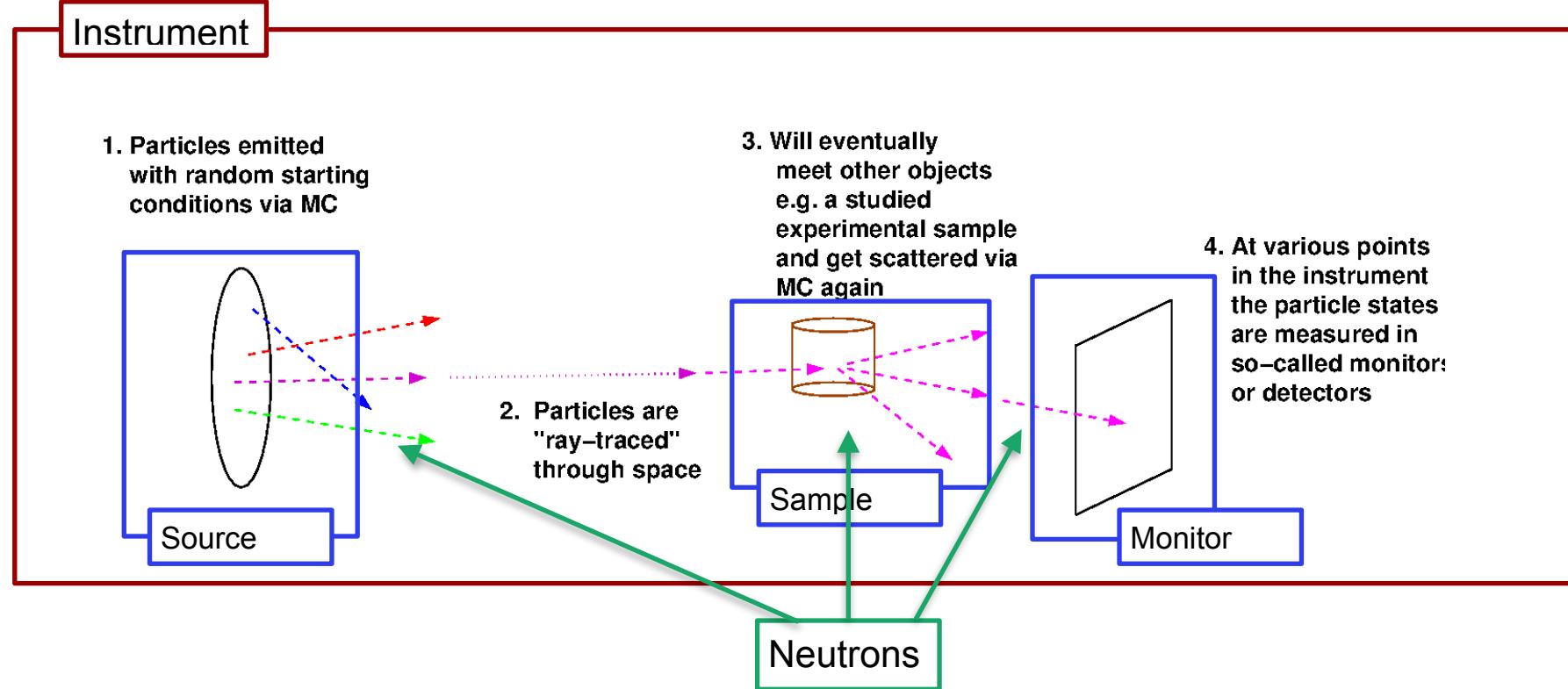
In the big picture, McStas is this...



The instrument defines our “lab coordinate system”

The components define devices or features available in our instrument - they have different function

In the big picture, McStas is this...

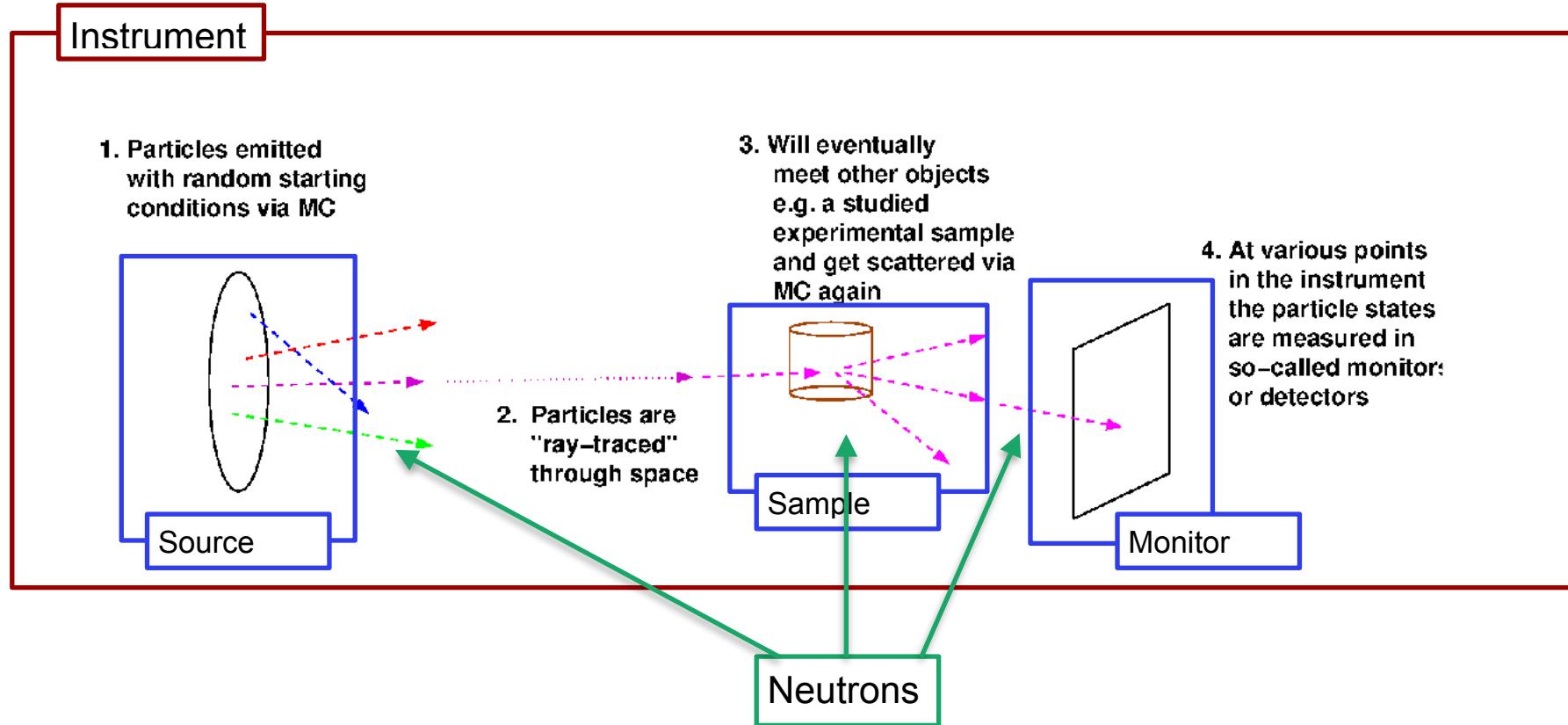


The instrument defines our “lab coordinate system”

The components define devices or features available in our instrument - they have different function

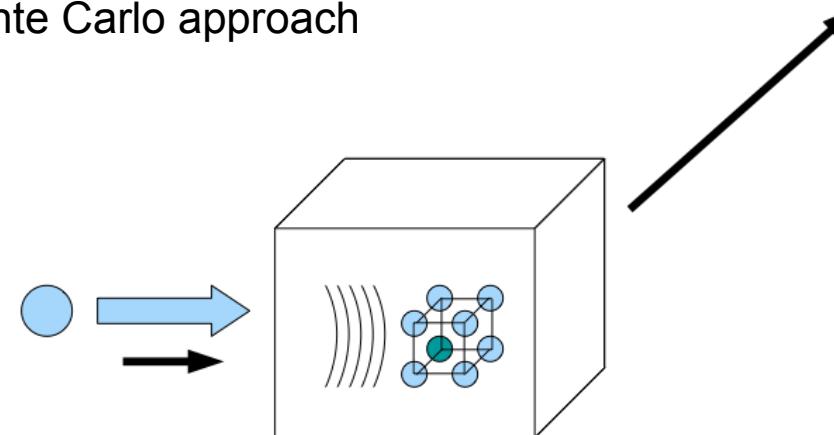
Neutron particles are passed on from one component to the next, changing state under way

McStas is a Monte Carlo ray-tracer



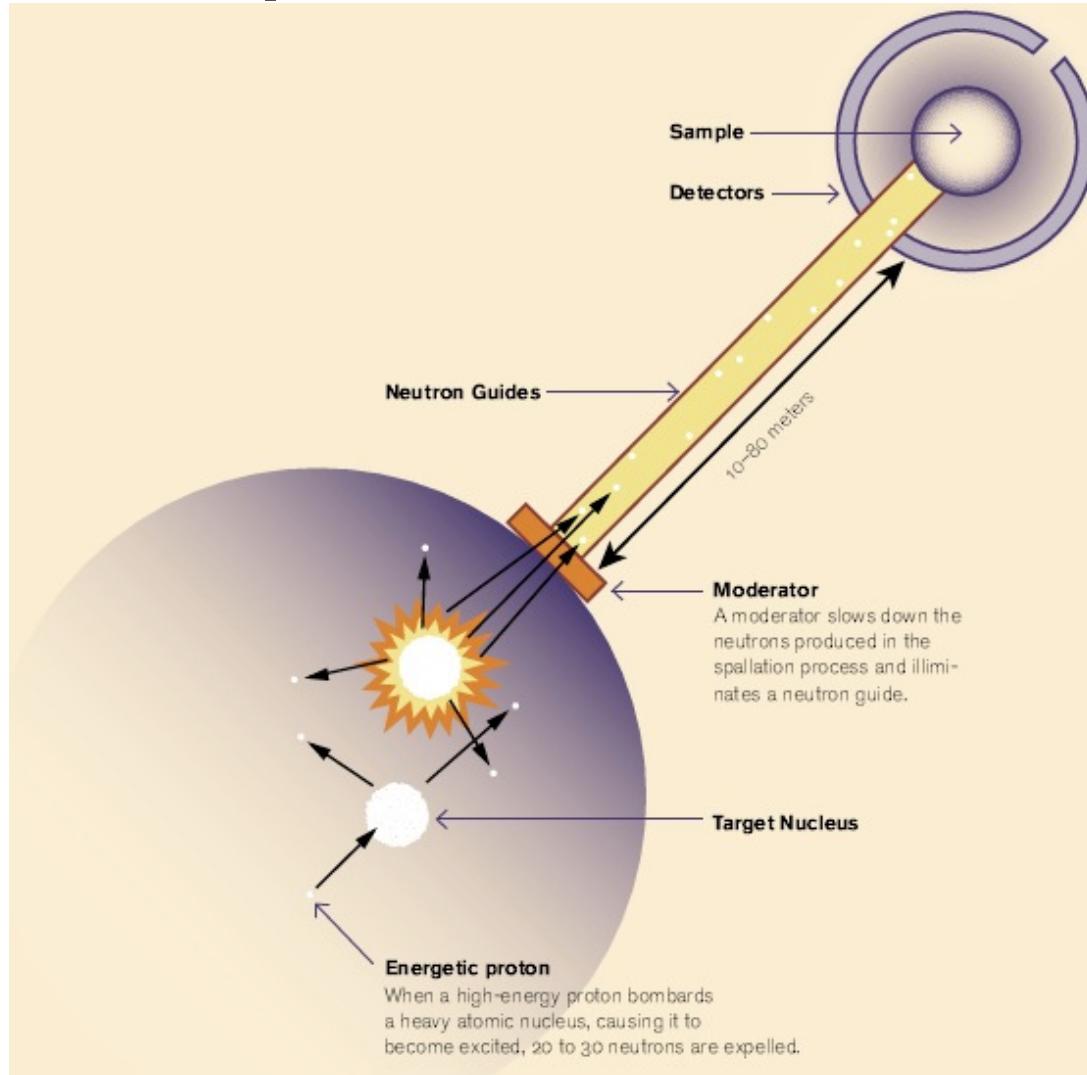
Elements of Monte-Carlo raytracing

- Instrument Monte Carlo methods implement coherent scattering effects
- Uses deterministic propagation where this can be done
- Uses Monte Carlo sampling of “complicated” distributions and stochastic processes and multiple outcomes with known probabilities are involved
 - I.e. inside scattering matter
- Uses the particle-wave duality of the neutron to switch back and forward between deterministic ray tracing and Monte Carlo approach

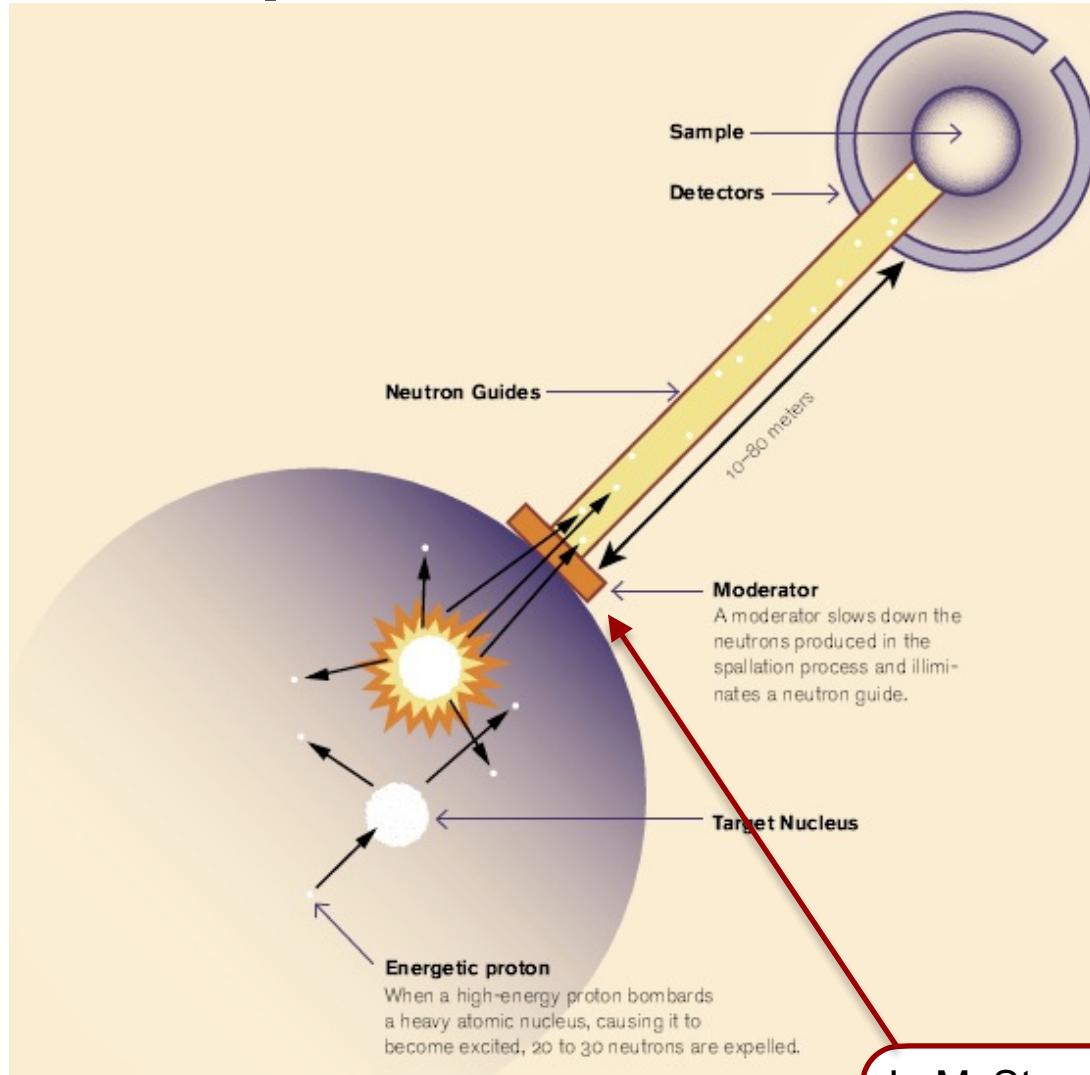


- Result: A realistic and efficient transport of neutrons in the thermal and cold range

Components of neutron instruments

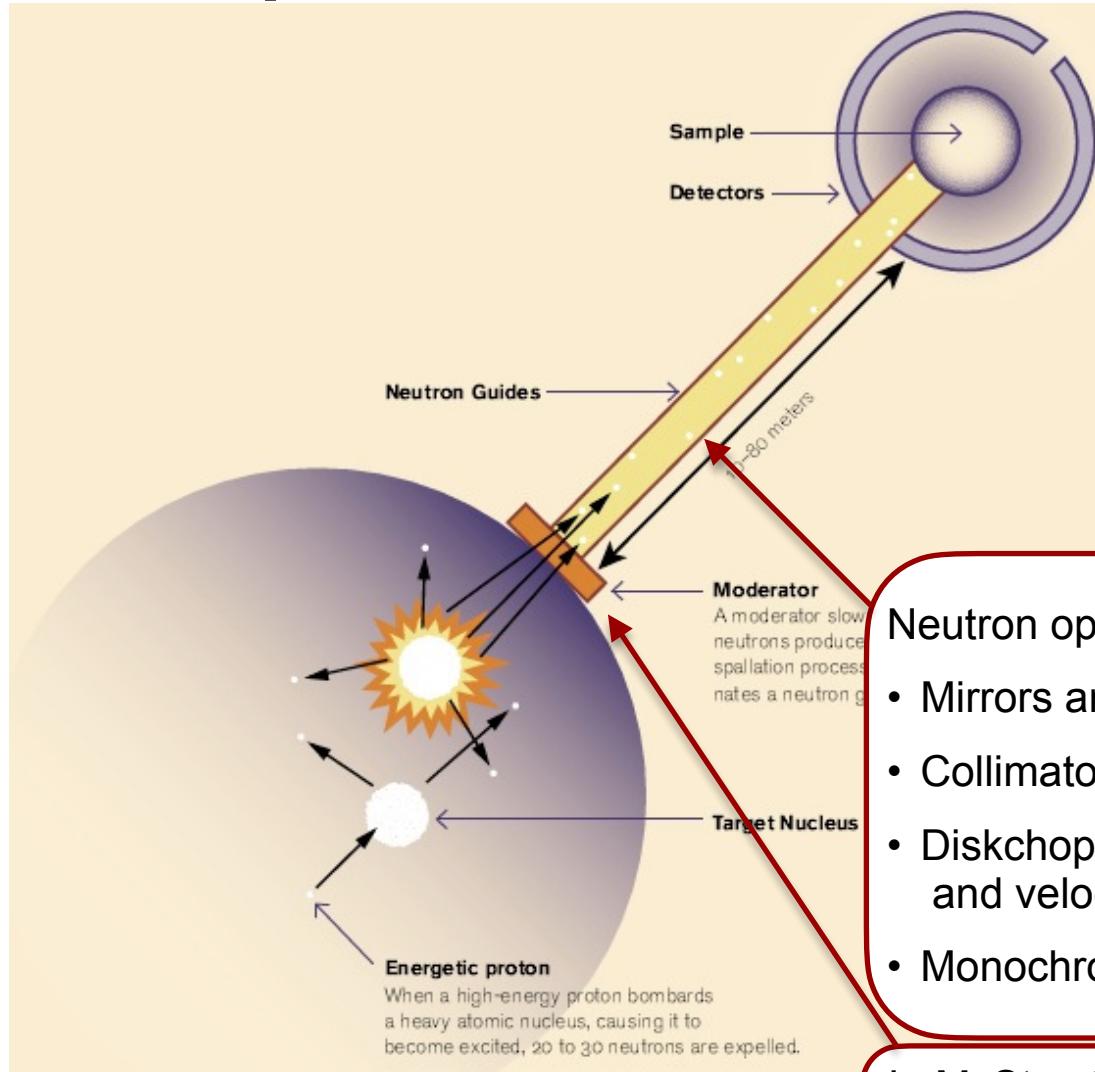


Components of neutron instruments



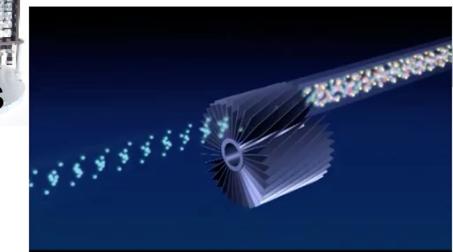
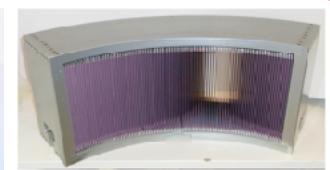
In McStas the moderator is the “source”

Components of neutron instruments



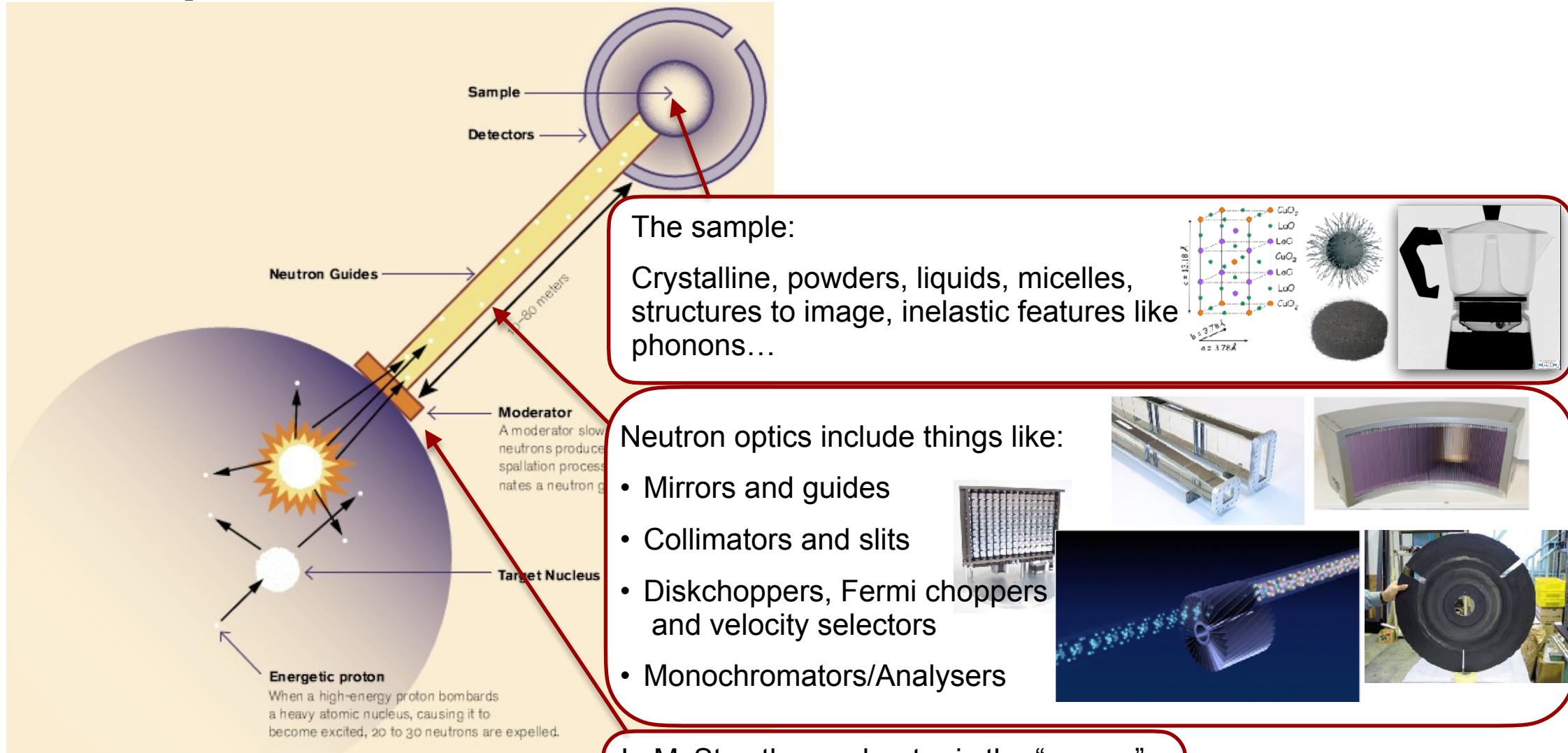
Neutron optics include things like:

- Mirrors and guides
- Collimators and slits
- Diskchoppers, Fermi choppers and velocity selectors
- Monochromators/Analysers

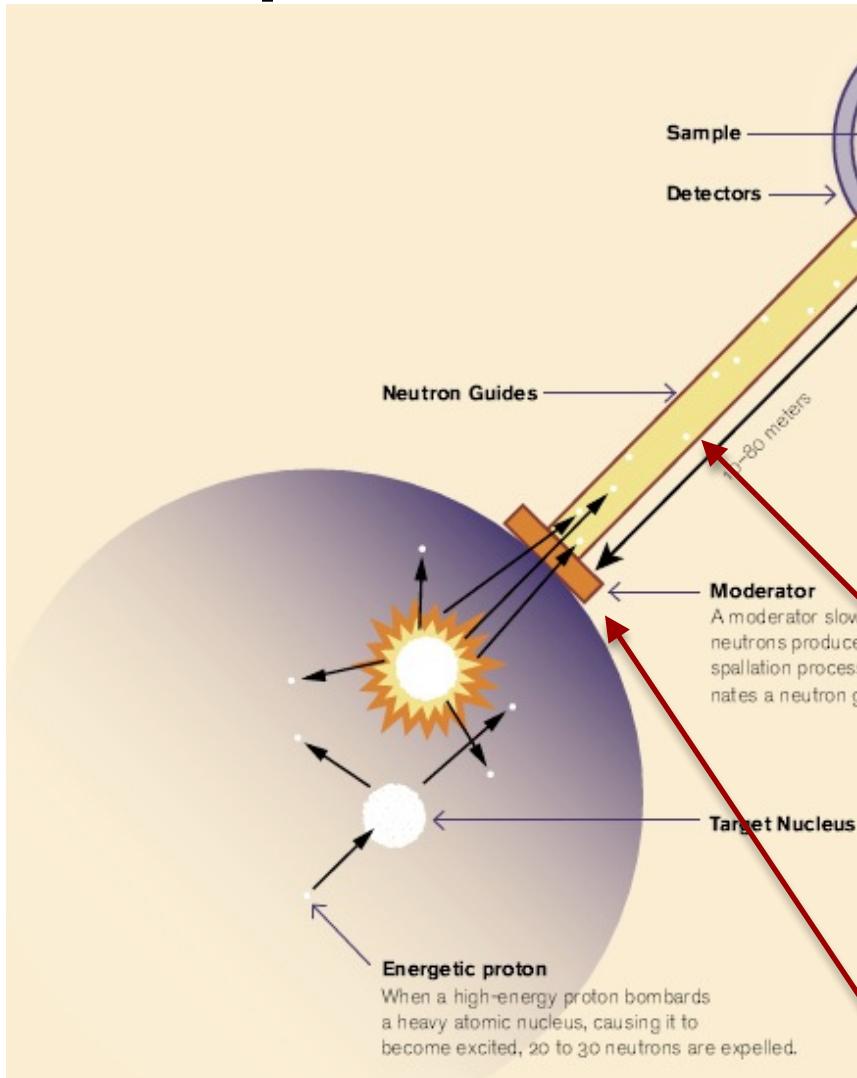


In McStas the moderator is the “source”

Components of neutron instruments



Components of neutron instruments



Energetic proton
When a high-energy proton bombards a heavy atomic nucleus, causing it to become excited, 20 to 30 neutrons are expelled.

Target Nucleus

Moderator
A moderator slows neutrons produced by the spallation process. It generates a neutron guide.

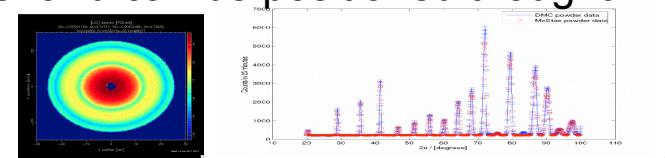
Neutron Guides

Sample

Detectors

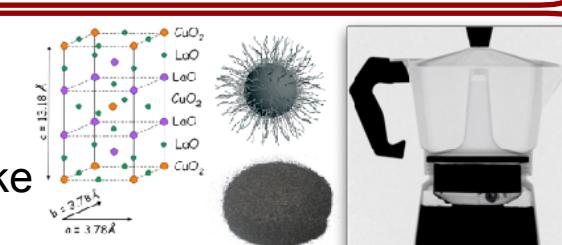
1-80 meters

Detectors are “monitors” in McStas. Mostly they act as “perfect probes” and can be positioned thought your instrument gathering 1D/2D/ event lists...



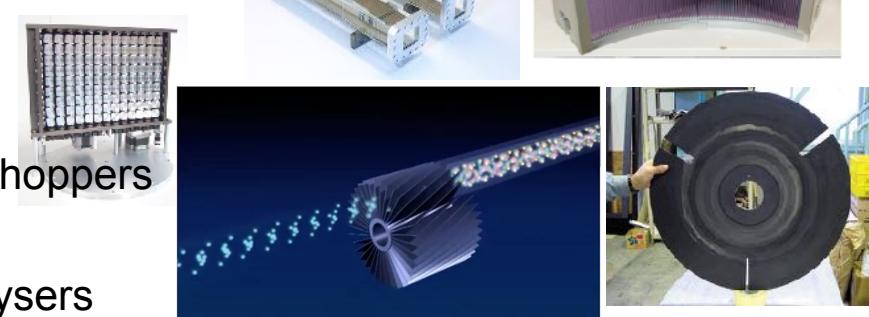
The sample:

Crystalline, powders, liquids, micelles, structures to image, inelastic features like phonons...



Neutron optics include things like:

- Mirrors and guides
- Collimators and slits
- Diskchoppers, Fermi choppers and velocity selectors
- Monochromators/Analysers



In McStas the moderator is the “source”

McStas Introduction

- Flexible, general simulation utility for neutron scattering experiments.
- Original design for **Monte carlo Simulation of triple axis spectrometers**
- Developed at DTU Physics, ILL, PSI, Uni CPH, ESS DMSC
- V. 1.0 by K Nielsen & K Lefmann (1998) RISØ
- Currently ~2-3 people full time plus students and user-contributions

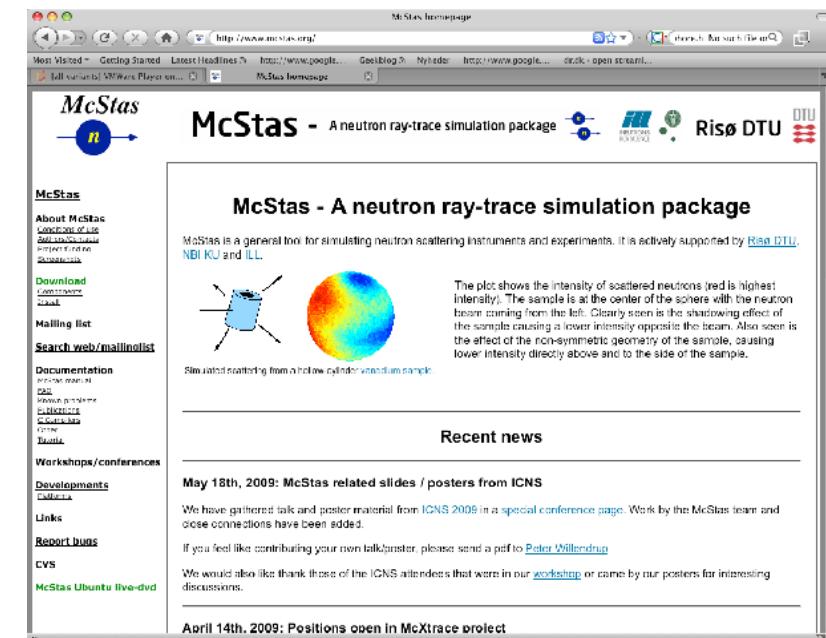
Project website at

<http://www.mcstas.org>

mcstas-users@mcstas.org mailinglist



GNU GPL license
Open Source



The screenshot shows the McStas homepage. On the left is a sidebar with links like "About McStas", "Documentation", "Workshops/conferences", and "Recent news". The main content area features a title "McStas - A neutron ray-trace simulation package" and a sub-section "McStas - A neutron ray-trace simulation package". It includes a plot titled "Simulated scattering from a hollow cylinder vanadium sample" showing intensity distribution. Below the plot, there's a section for "Recent news" with entries for May 18th, 2009, and April 14th, 2009.

McXtrace Introduction

Main Page – McXtraceWiki

Most Visited ▾ Getting Started Latest Headlines ▾ http://www.google... Geekblog ▾ Nyheder http://www.google... dr.dk ▾ open streami... Log in / create account

[article](#) [discussion](#) [edit](#) [history](#)

Main Page

McXtrace

[edit]

McXtrace - Monte Carlo Xray ray-tracing is a joint venture by

Risø DTU DTU ESRF JJ X-RAY Danish Science Design Engineering Production

Funding from NABIIT, DSF and the above parties.

Our code will be based on technology from 

For information on our progress, please subscribe to our [user mailinglist](#).
<mailto:webmaster@mcxtrace.org>

This page was last modified 13:15, 25 February 2009. This page has been accessed 2,049 times. Privacy policy About McXtraceWiki Disclaimers Powered By MediaWiki

- Synergy, knowledge transfer, shared infrastructure

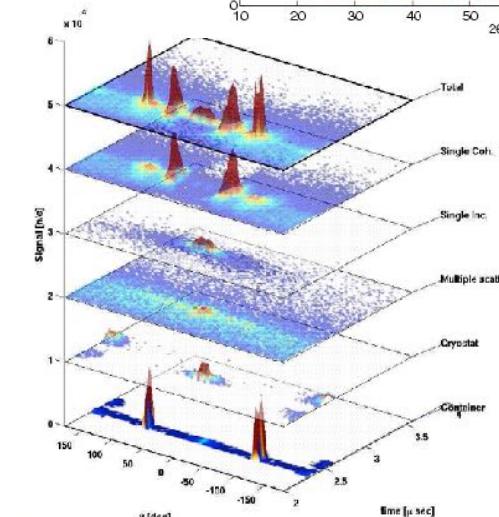
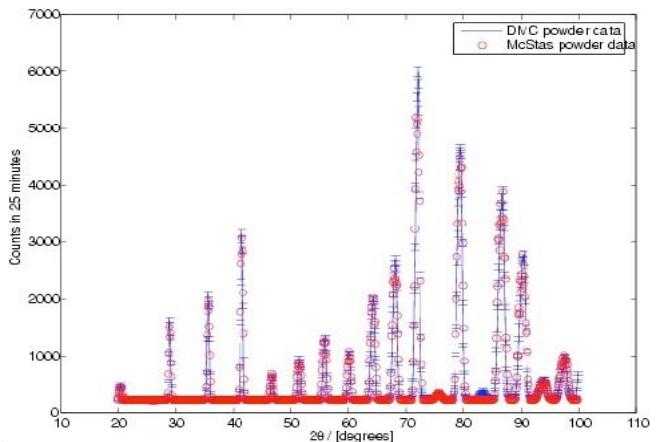
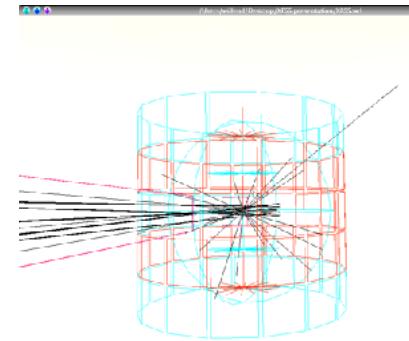
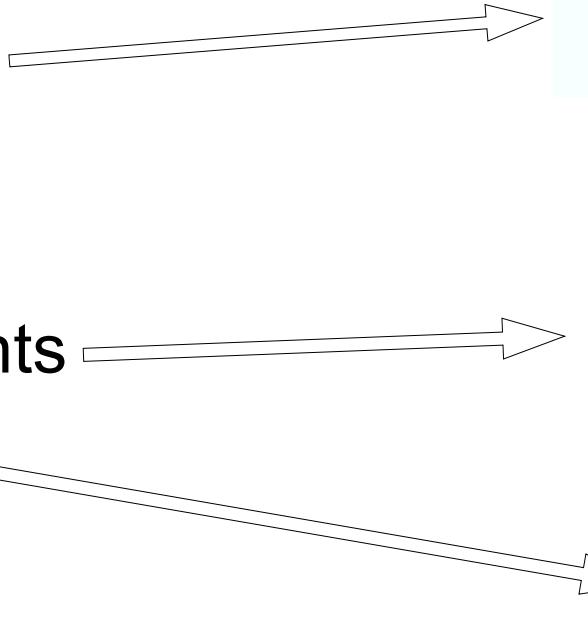
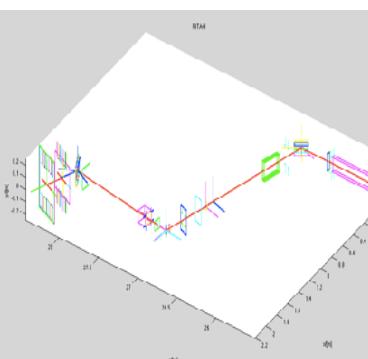
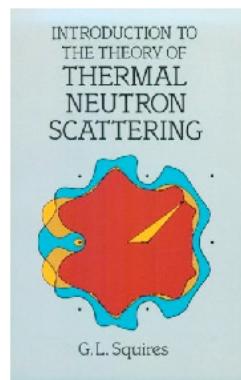
Used in many places



What is McStas used for?

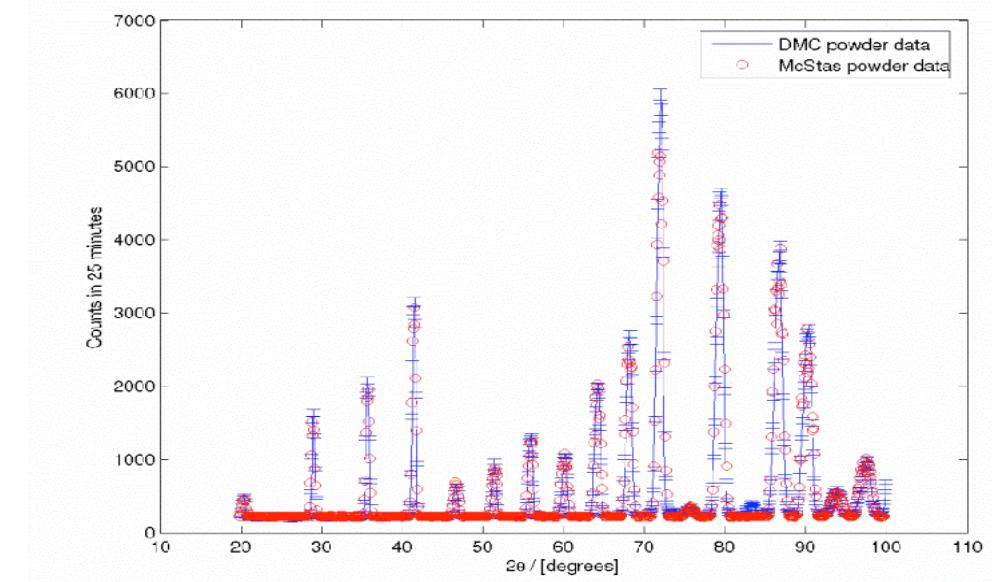
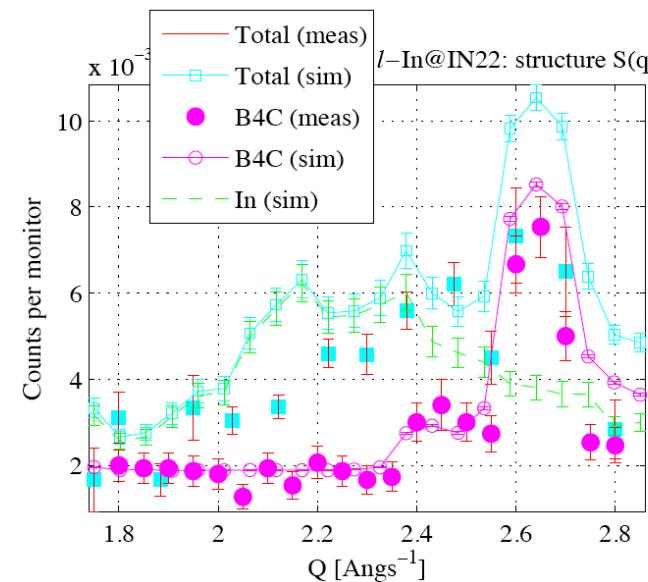
- Instrumentation
- Planning
- Construction
- Virtual experiments
- Data analysis
- Teaching

(KU, DTU)



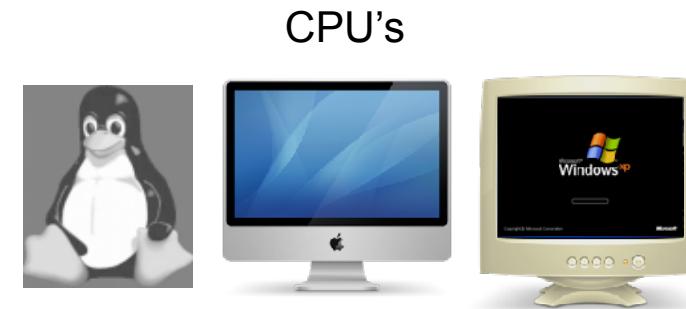
Reliability - cross comparisons

- Much effort has gone into this
- Here: simulations vs. exp. at powder diffract. DMC, PSI
- The bottom line is
- McStas agree very well with other packages (NISP, Vitess, IDEAS, RESTRAX, ...)
- Experimental line shapes are within 5%
- Absolute intensities are within 10%
- Common understanding: McStas and similar codes are reliable



McStas overview

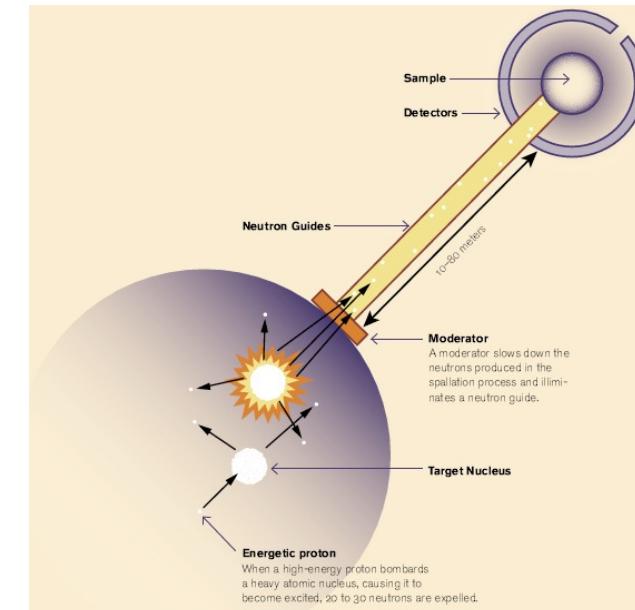
- Portable code (Unix/Linux/Mac/Windoze)
- Ran on everything from iPhone to 1000+ node cluster!



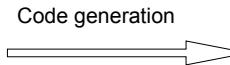
+ NVIDIA GPU's

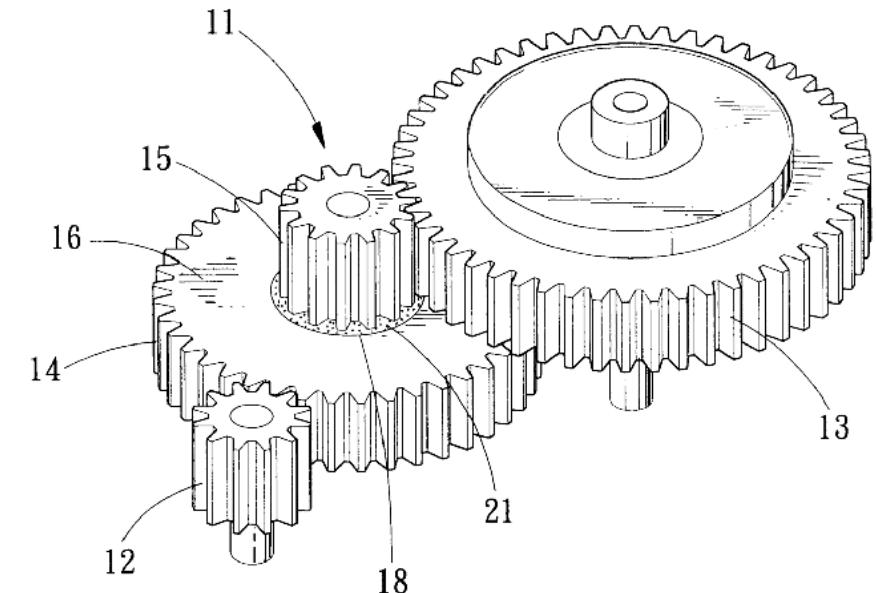


- 'Component' files (~100) inserted from library
 - Sources
 - Optics
 - Samples
 - Monitors
 - If needed, write your own comps
- DSL + ISO-C code gen.



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
 - ...

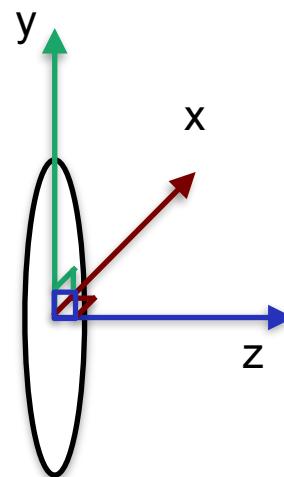


Implementation

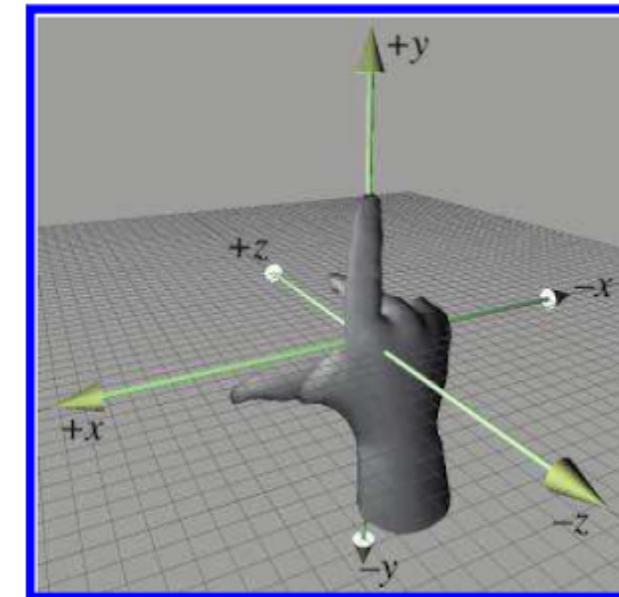
- Three levels of source code:
 - Instrument file (All users)
 - Component files (Some users)
 - ANSI c code (no users)

Placing components - source

- One of the first components in your instrument is typically a source, which has a coordinate system like this....



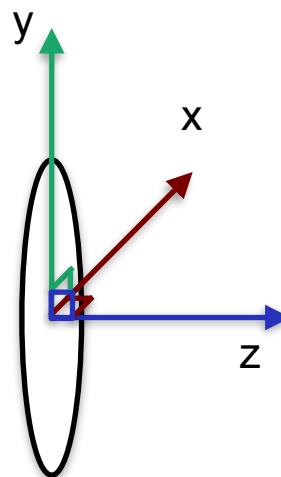
- z is along neutron beam direction
- y is vertical
- x at an angle of 90° wrt. z,y



Right-handed
coordinate system

Placing components - source

- Often the source coordinate system coincides with the “lab” coordinate system, denoted ABSOLUTE in McStas language, i.e.

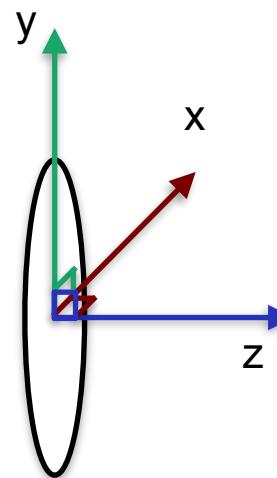


- COMPONENT Source = Source_simple(...)
AT (0,0,0) ABSOLUTE

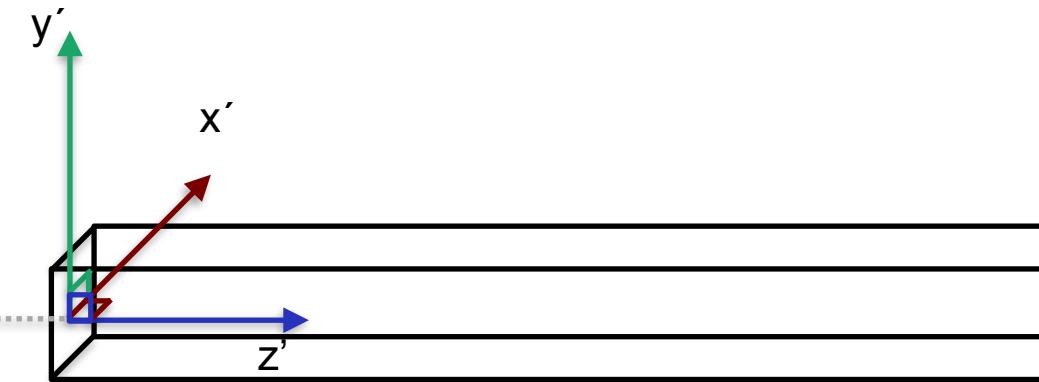
Placing further components - RELATIVE

Placing further components is done by order of

1. Location, i.e



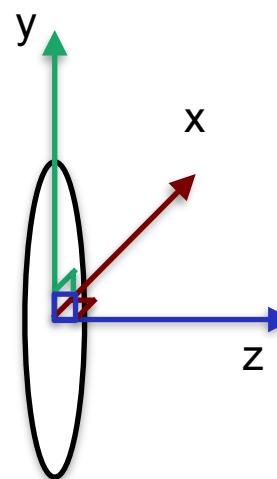
```
COMPONENT Source = Source_simple(...)  
AT (0,0,0) ABSOLUTE
```



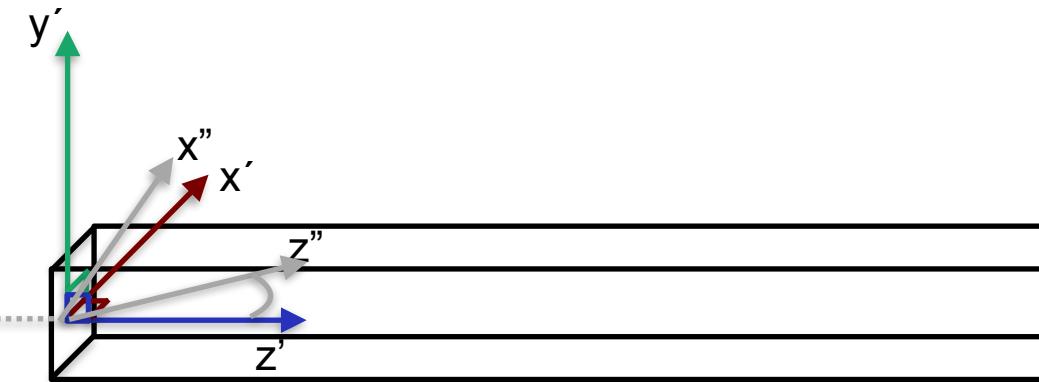
```
COMPONENT Guide = Guide(...)  
AT (0,0,1) RELATIVE Source
```

Placing further components - RELATIVE

Placing further components is done by order of
2. Rotation, i.e



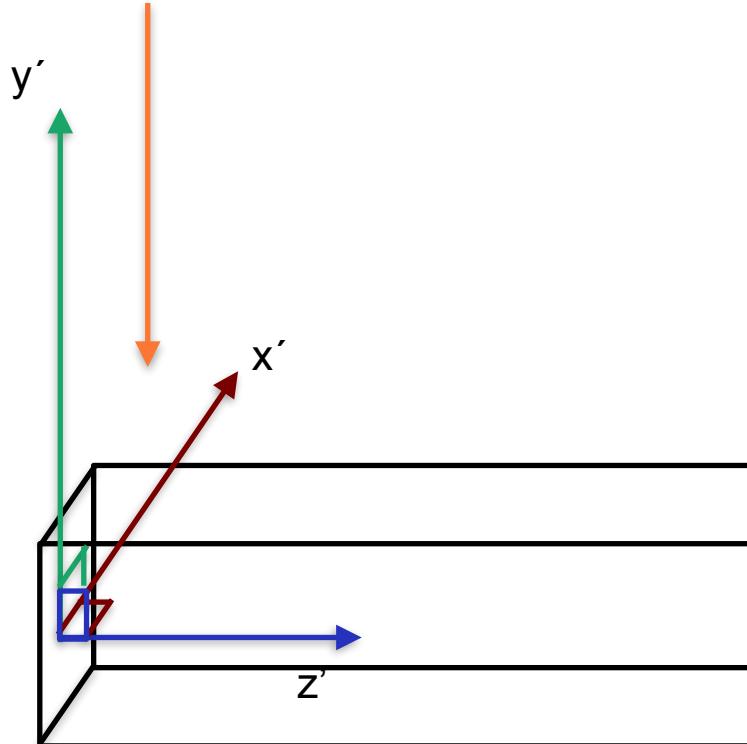
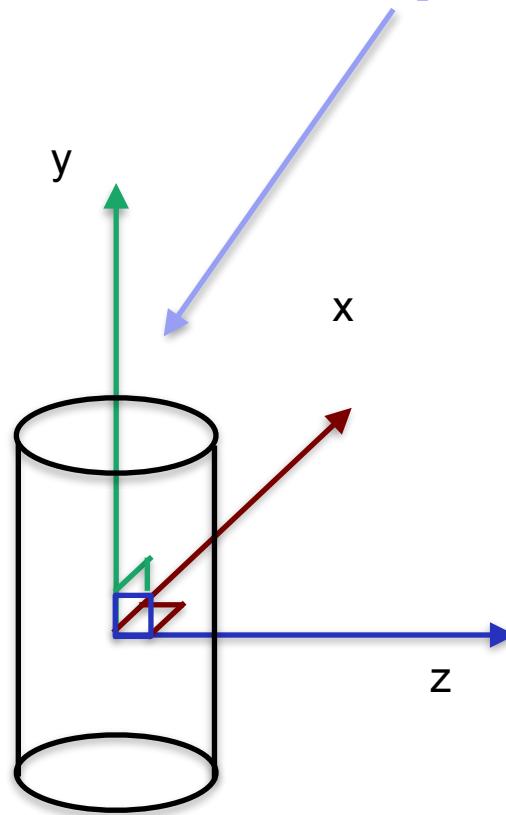
COMPONENT Source = Source_simple(...)
AT (0,0,0) ABSOLUTE



COMPONENT Guide = Guide(...)
AT (0,0,1) RELATIVE Source
ROTATED (0,30,0) RELATIVE Source

(Reference labels can also be PREVIOUS or PREVIOUS+1 etc.)

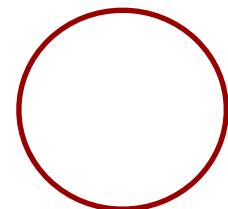
Components often have their origin at the centre of mass, i.e. for samples ... but not for neutron guides



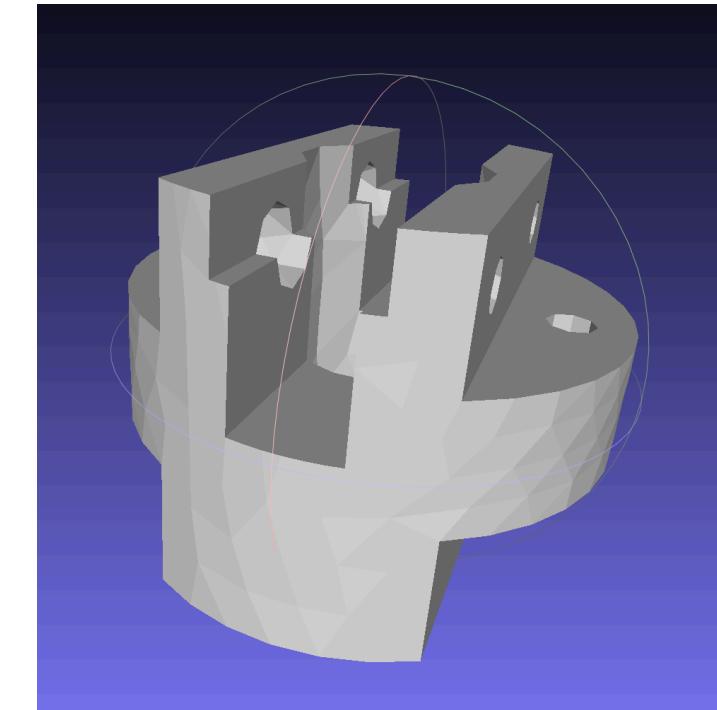
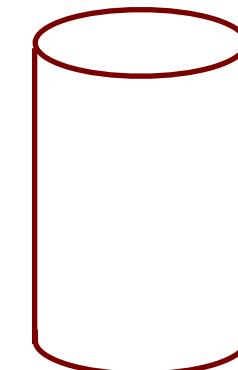
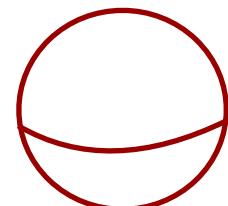
Generally speaking, the component author can choose
the meaningful coordinate system for the given problem!
- The McStas system takes care of the transformation between them....

Component geometries are typically simple objects... But some have polygon-description of the surface

2D



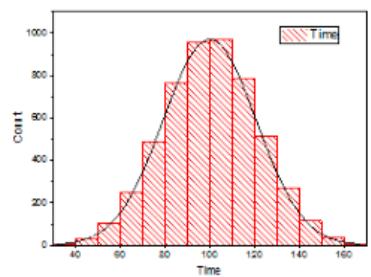
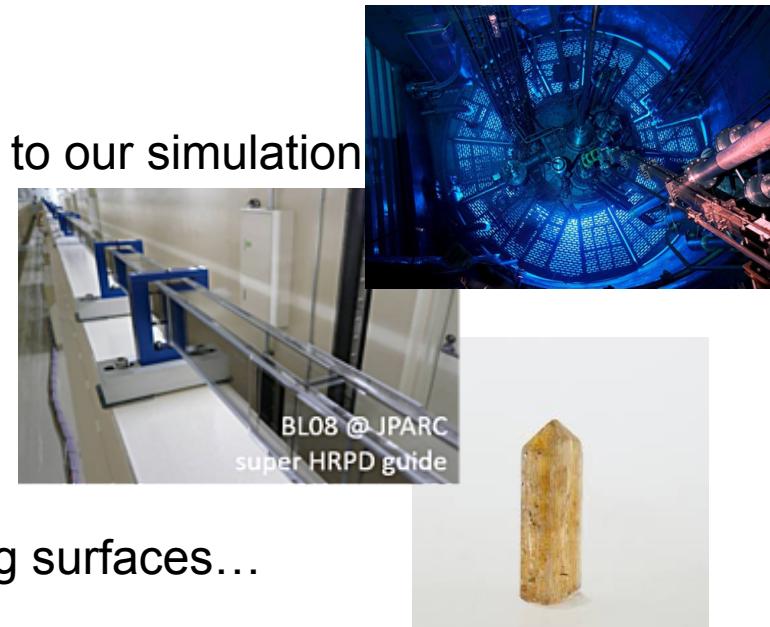
3D



Component classes

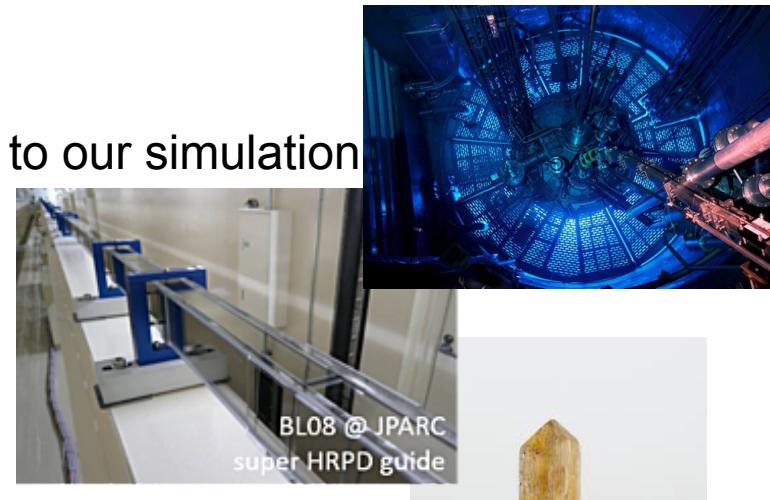
- Sources - these define MC starting conditions / “inject” neutrons to our simulation
- Optics - used to tailor properties of the neutron beam
 - Examples are mirrors, guides, choppers, collimators, slits, ...
- Samples - “matter” of some form
 - Powders, single crystals, liquids, micelles in solution, reflecting surfaces...
- Monitors - may probe the state of the neutron beam and store histograms / event lists
- Misc, obsolete
 - “Other stuff” and “Old stuff”

*Other
Stuff*



Component classes

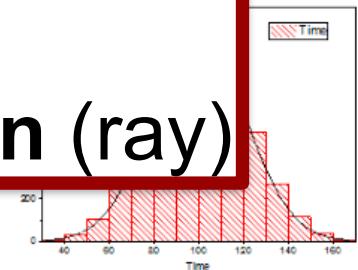
- Sources - these define MC starting conditions / “inject” neutrons to our simulation
- Optics - used to tailor properties of the neutron beam
 - Examples are mirrors, guides, choppers, collimators, slits, ...
- Samples - “matter” of some form
 - Powders, single crystals, liquids, micelles in solution, reflecting surfaces



- Monitors - may probe
- Misc, obsolete
 - “Other stuff” and “Old stuff”

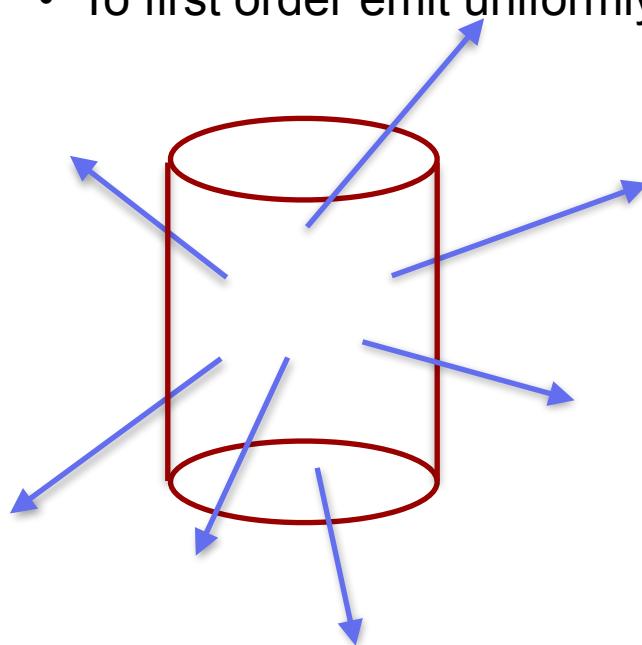
Common to all components:
They set, manipulate/interact with
or measure the **state of the neutron (ray)**

*Other
Stuff*

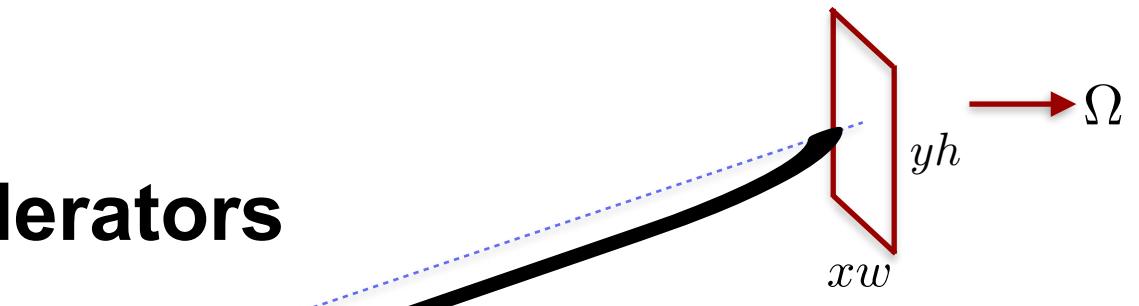
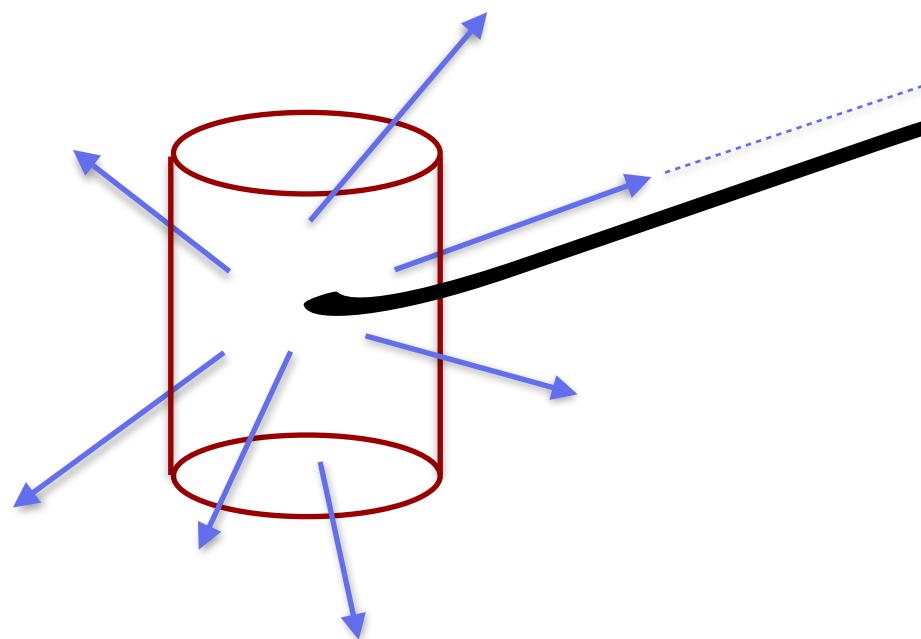


Neutron sources, i.e. moderators

- To first order emit uniformly into 4π steradian

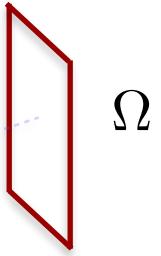


Neutron sources, i.e. moderators



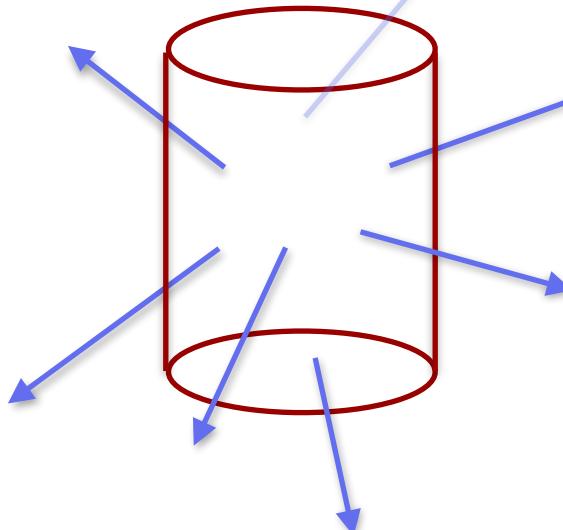
E.g. the beamport?

- Generally we are interested in the input to a single instrument, characterised by a certain solid angle Ω , often corresponding to a rectangle $xw \times yh$ at a distance $dist$ from the source



Neutron sources, i.e. moderators

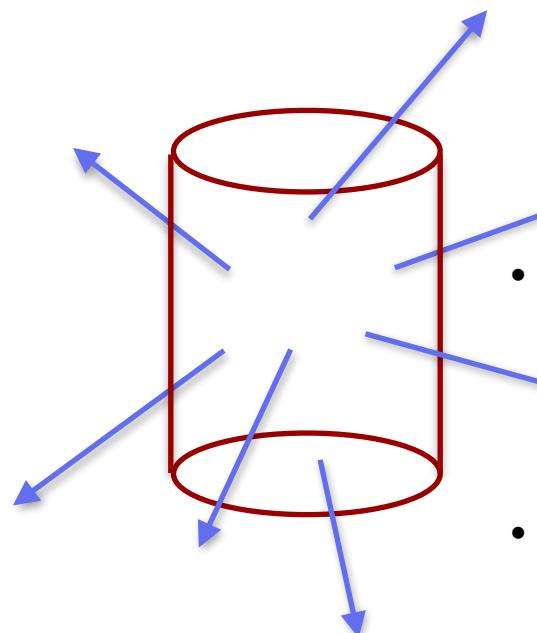
- The emission intensity into our chosen solid angle Ω can be a function of wavelength, time (pulsed sources) and possibly point of origin on the source surface



$I(\lambda)$	$[n/s/str]$
$I(\lambda, t)$	$[n/s/str]$
$I(\lambda, t, \vec{r})$	$[n/s/str]$

- The emission of particles into the solid angle Ω is in fact an integration and leads to a simulated “intensity” of I_Ω $[n/s]$
- In McStas, that integrated intensity is partitioned over a given set of particle rays referred to as **ncount**, -n or --ncount
- The default **ncount** is 1e6 rays

Neutron sources, i.e. moderators



- Our neutron rays are emitted randomly, sampling Ω and all variables of the source “spectrum”, i.e. wavelength, time and area
- assigning neutron weights p such that

$$\sum_{j=1}^{\text{ncount}} p_j = \int_{d\lambda, dt, d\vec{r}} I_\Omega(\lambda, t, \vec{r})$$



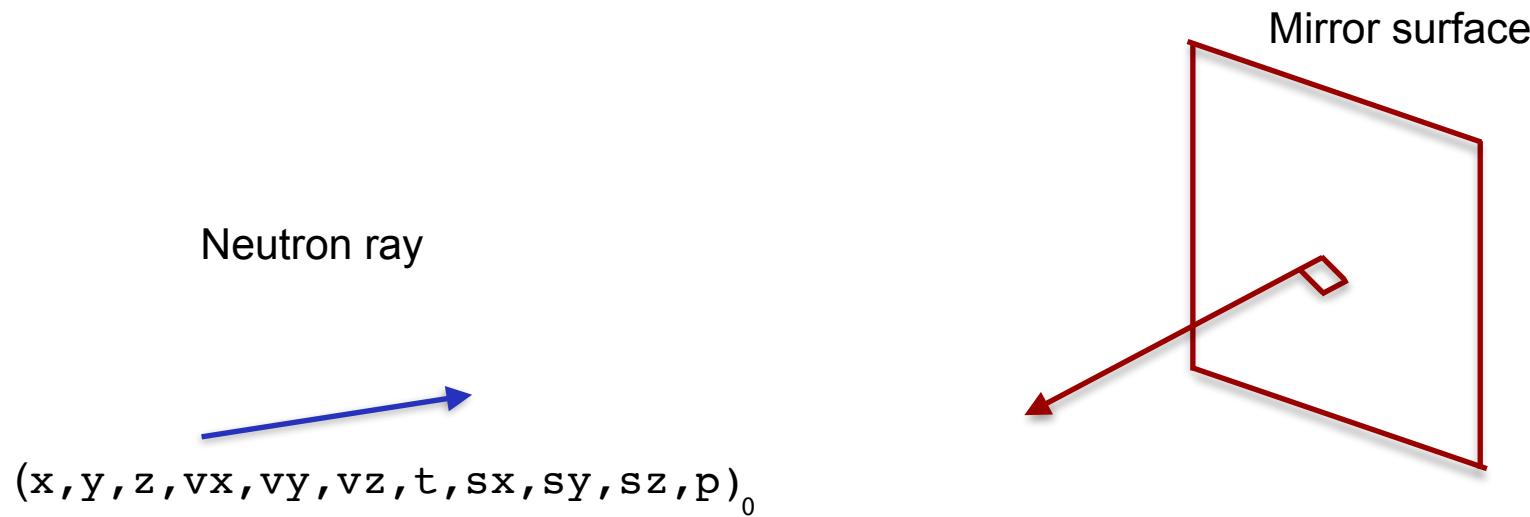
Neutron rays in McStas - what are they?

- Defining the neutron starting conditions imply setting:
 - The **starting point** on the surface, i.e. \vec{r} (in the code variables x, y, z)
 - The **direction** into Ω and our λ/E_{kin} (in the code variables vx, vy, vz)
 - The **starting time** (in the code the variable t)
 - The initial **intensity** / weight of the neutron ray (in the code the variable p)
 - If needed the initial **polarisation** (in the code the variables sx, sy, sz)

Neutron ray in McStas:	
Location	x, y, z
Velocity	vx, vy, vz
Time	t
Polarisation.	sx, sy, sz
Intensity	p

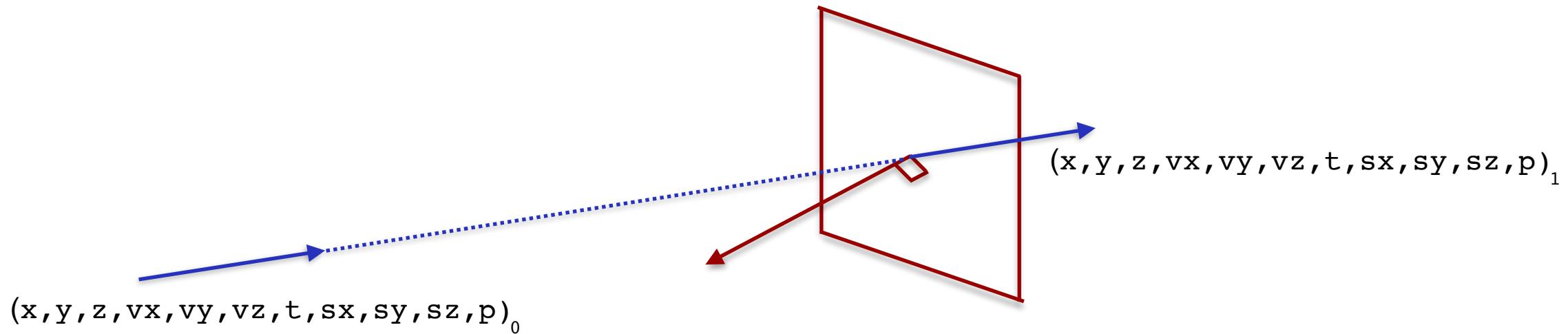
Neutron (ray)-matter interaction 1: reflecting surface

- 1 starting situation



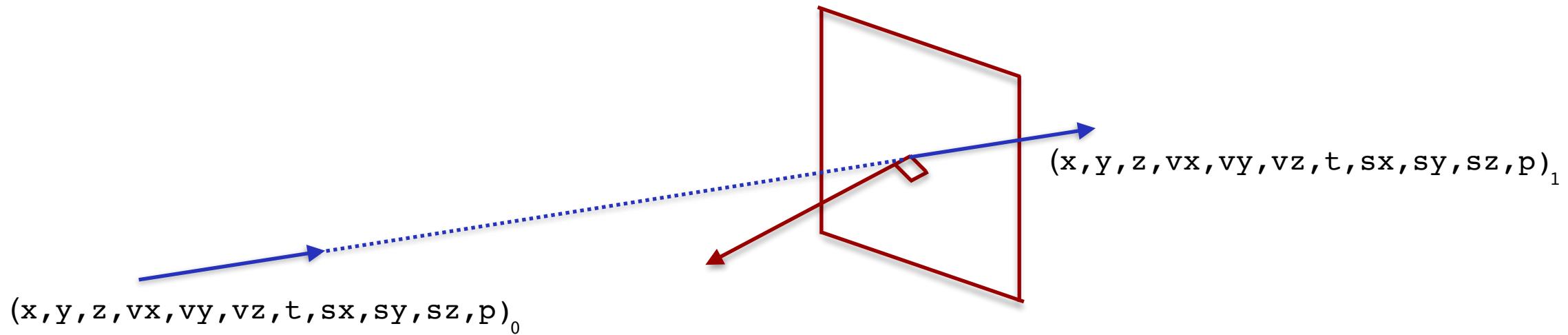
Neutron (ray)-matter interaction 1: reflecting surface

- 2. Propagate to the mirror surface



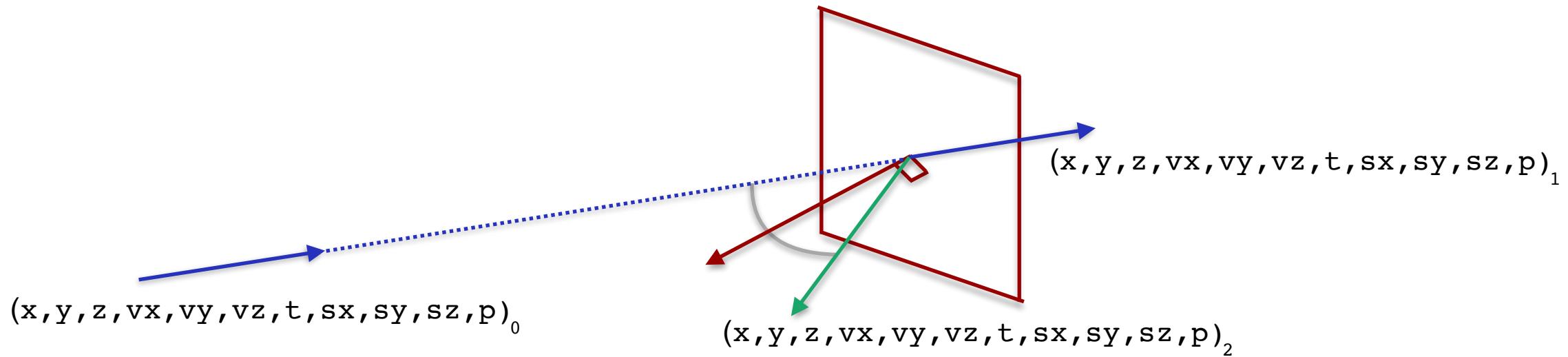
Neutron (ray)-matter interaction 1: reflecting surface

- 3. Checks (are we on surface, what is probability of reflection etc.)



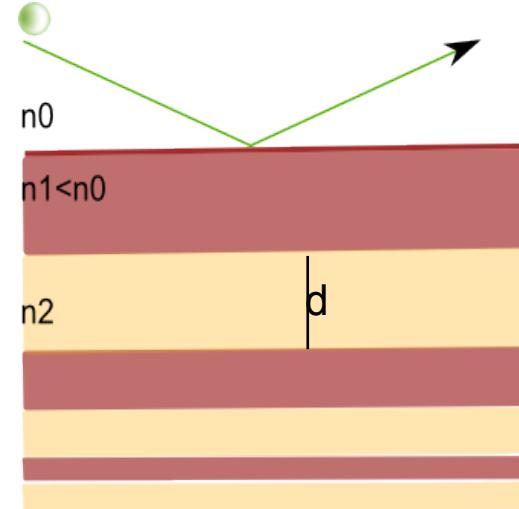
Neutron (ray)-matter interaction 1: reflecting surface

- 4. Reflect



Weight of final ray is adjusted according to reflectivity, see next slide

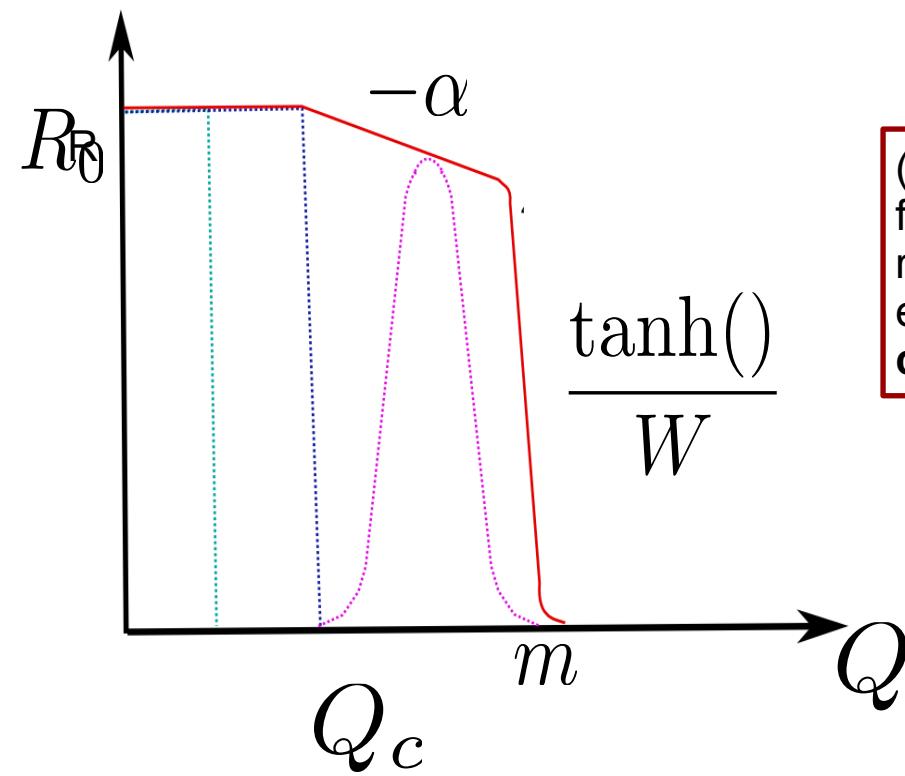
Parametrisation of reflectivity on mirrors etc.



$$V = \frac{2\pi\hbar^2}{m} bN \quad \sin\theta < \sqrt{\frac{mV}{2\pi^2\hbar^2}}\lambda$$

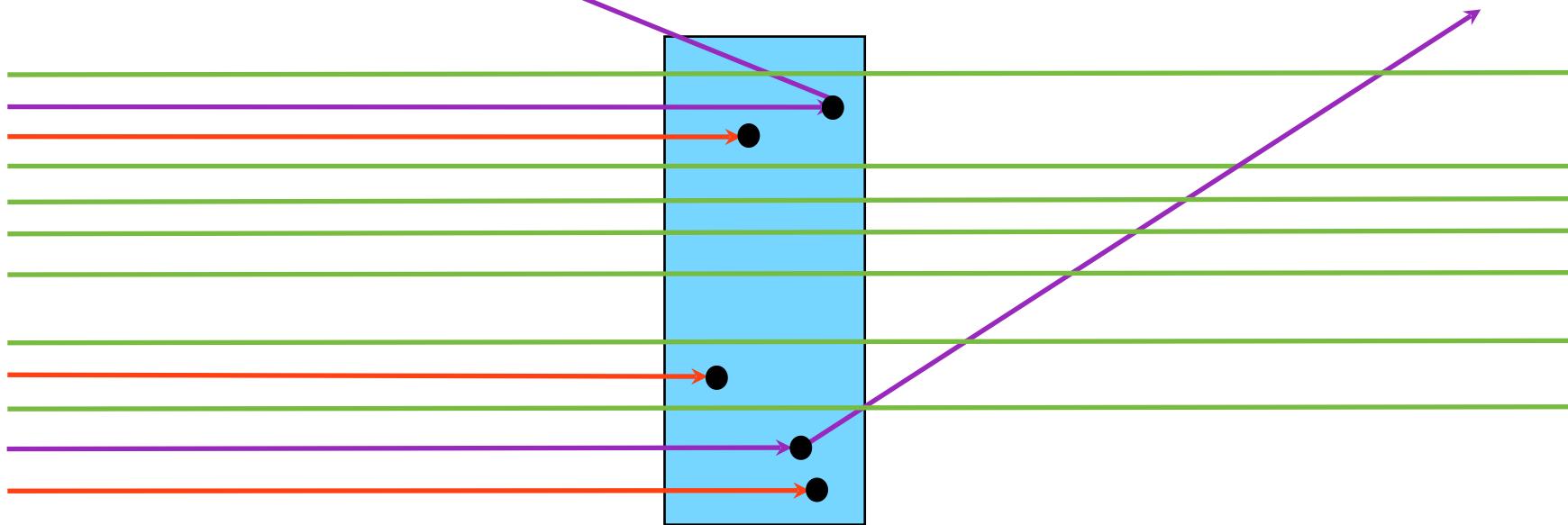
$$m = \frac{\theta_{mirror}}{\theta_{Ni}}$$

$$R_0 \cdot \left(1 - \frac{\tanh(Q - mQ_c)}{W}\right) \cdot (1 - \alpha(Q - Q_c))$$



(i.e. Q is calculated for given neutron, reflectivity encoded in **changed p value**)

Neutron (ray)-matter interaction in General



A neutron hitting a sample can be:
absorbed, **transmitted**, or **scattered**

Samples

For a **non-thin** sample the probabilities for **absorption**, **transmission** or **scattering** are given by

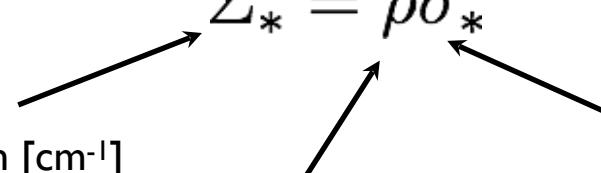
$$p_A = (1 - e^{-\Sigma_T t})(\Sigma_A / \Sigma_T)$$

$$p_S = (1 - e^{-\Sigma_T t})(\Sigma_S / \Sigma_T)$$

$$p_T = 1 - p_S - p_A = e^{-\Sigma_T t}$$

$$\Sigma_* = \rho \sigma_*$$

$\mathbf{t} = \text{sample thickness}$



```

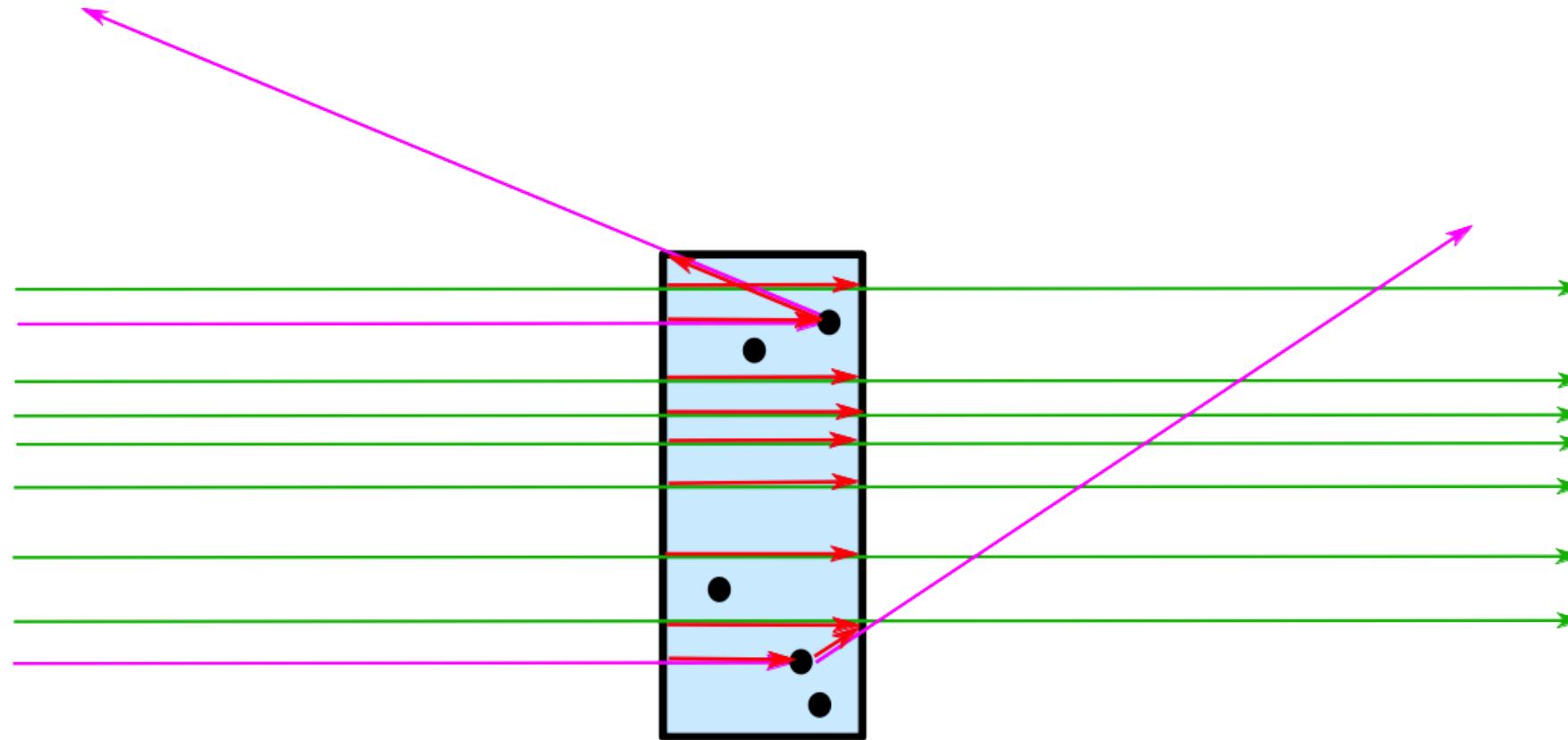
graph TD
    A[macroscopic cross section [cm⁻¹]] --> B["Σ* = ρσ*"]
    C[number density [atoms/cm³]] --> B
    D["microscopic cross section [barn/atom]"] --> B
    E["1 barn = 10⁻²⁴ cm²"]
  
```

macroscopic cross section [cm⁻¹]

number density [atoms/cm³]

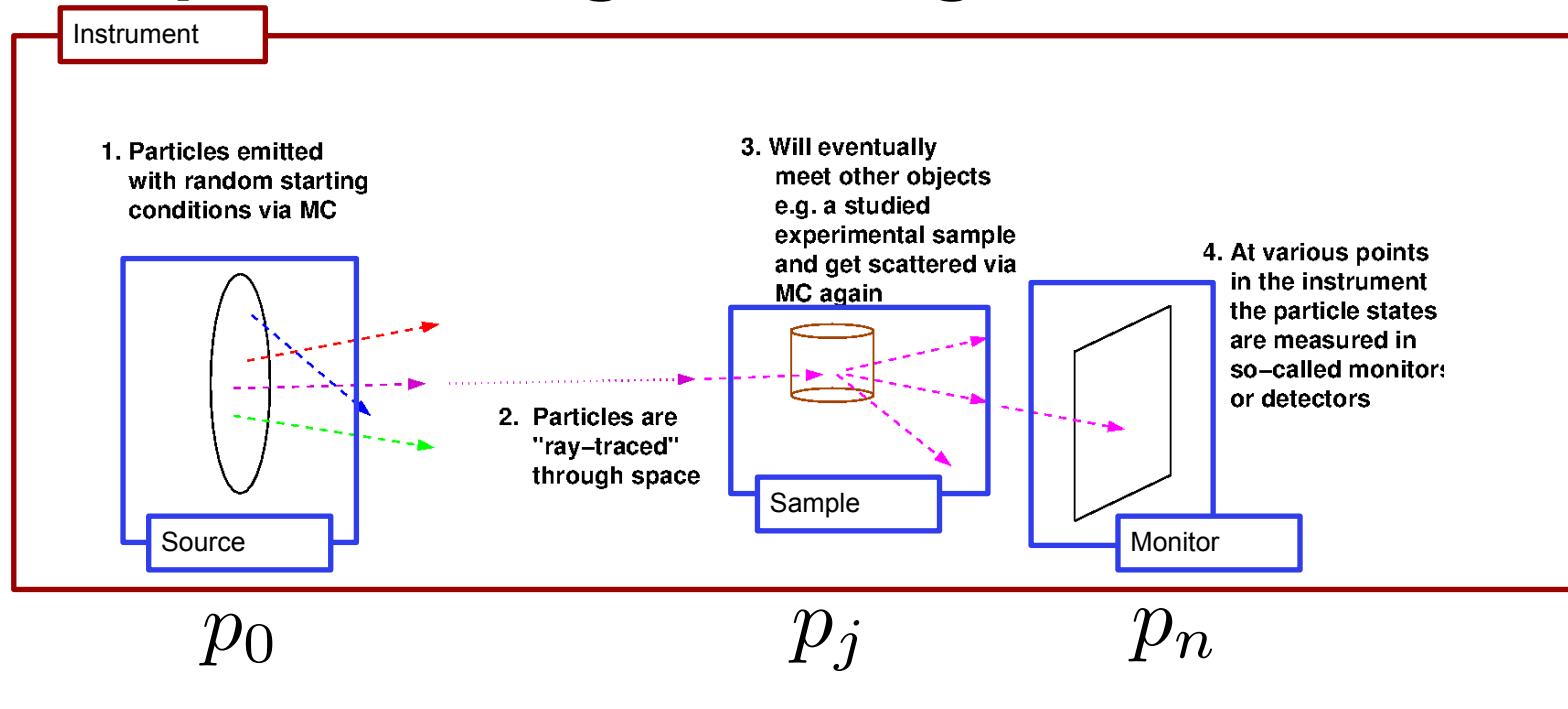
microscopic cross section [barn/atom]
1 barn = 10⁻²⁴cm²

Samples/Matter interaction in General in McStas



A neutron ray hitting a sample can be:
transmitted+absorption, or scattered+absorption

Transport of weight through the instrument...



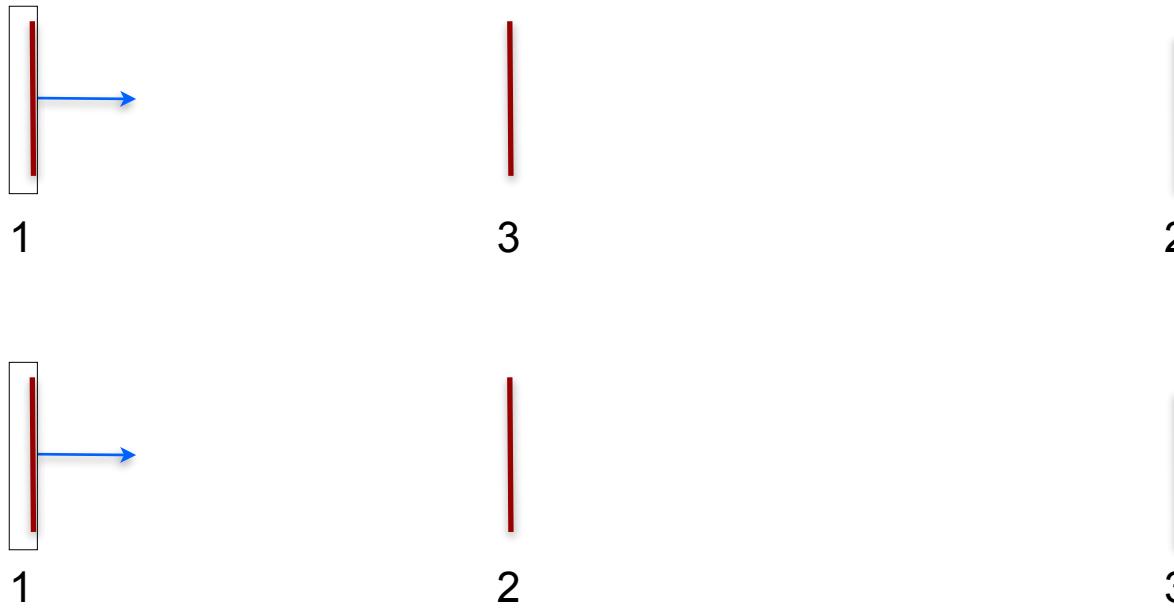
$$p_j = p_0 \prod_{k=1}^j w_k$$

The weight multiplier of the j 'th component, w_j , is calculated by the probability rule $f_{MC,b}w_j = P_b$ where P_b is the physical probability for the event "b", and $f_{MC,b}$ is the probability that the Monte Carlo simulation selects this event.

In case of "branching", i.e. multiple outcomes, it is clear that

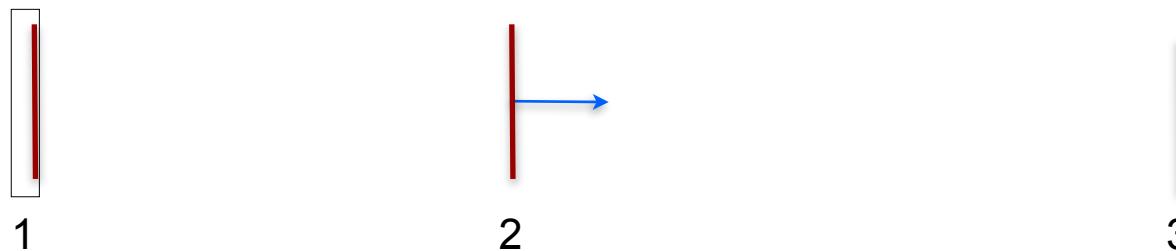
$$\sum_b f_{MC,b} = 1$$

To first order, McStas is linear and follows sequence of components in your file...



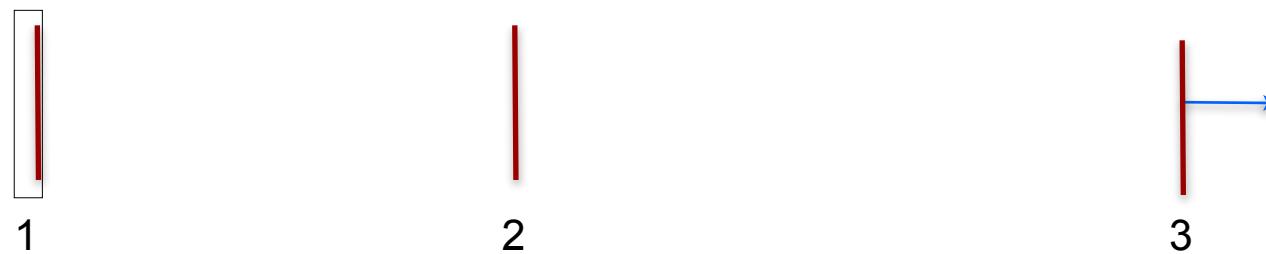
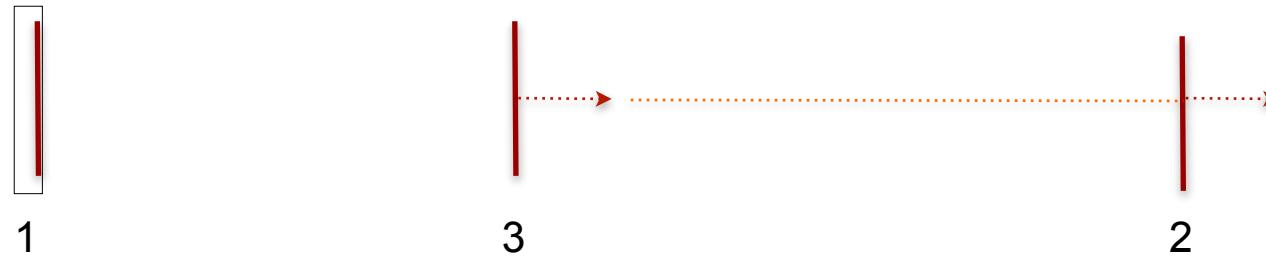
Starting at the source

To first order, McStas is linear and follows sequence of components in your file...



Moving to first comp in the list

To first order, McStas is linear and follows sequence of components in your file...



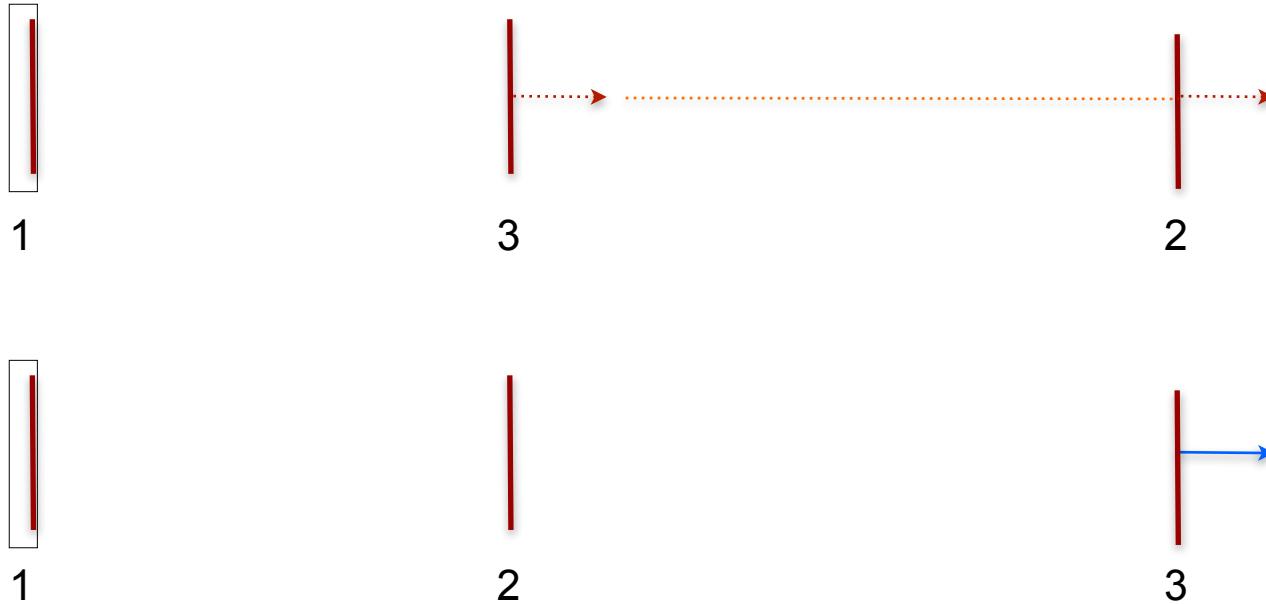
Moving to 3rd comp in list requires “moving back in time”.

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.

To first order, McStas is linear and follows sequence of components in your file...



The order of components is important,
and in general overlaps should be avoided!

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.

Instrument file

```
DEFINE INSTRUMENT My_Instrument(DIST=10)

/* Here comes the TRACE section, where the actual      */
/* instrument is defined as a sequence of components.  */
TRACE

/* The Arm() class component defines reference points and orientations */
/* in 3D space.                                                       */
COMPONENT Origin = Arm()
    AT (0,0,0) ABSOLUTE

COMPONENT Source = Source_simple(
    radius = 0.1, dist = 10, xw = 0.1, yh = 0.1, E0 = 5, dE = 1)
    AT (0, 0, 0) RELATIVE Origin

COMPONENT Emon = E_monitor(
    filename = "Emon.dat", xmin = -0.1, xmax = 0.1, ymin = -0.1,
    ymax = 0.1, Emin = 0, Emax = 10)
    AT (0, 0, DIST) RELATIVE Origin

COMPONENT PSD = PSD_monitor(
    nx = 128, ny = 128, filename = "PSD.dat", xmin = -0.1,
    xmax = 0.1, ymin = -0.1, ymax = 0.1)
    AT (0, 0, 1e-10) RELATIVE Emon

/* The END token marks the instrument definition end */
END
```

Written by you!

Component file

```

*****+
* Mcstas, neutron ray-tracing package
* Copyright 1997-2002, All rights reserved
* Risoe National Laboratory, Roskilde, Denmark
* Institut Laue Langevin, Grenoble, France
*
* Component: Source_flat
* %I
* Written by: Kim Lefmann
* Date: October 30, 1997
* Modified by: KL, October 4, 2001
* Modified by: Emmanuel Farhi, October 30, 2001. Serious bug corrected.
* Version: $Revision: 1.22 $
* Origin: Risoe
* Release: McStas 1.6
*
* A circular neutron source with flat energy spectrum and arbitrary flux
* %D
* The routine is a circular neutron source, which aims at a square target
* centered at the beam (in order to improve MU-acceptance rate). The angular
* divergence is then given by the dimensions of the target.
* The neutron energy is uniformly distributed between E0-dE and E0+dE.
*
* Example: Source_flat(radius=0.1, dist=2, xw=.1, yh=.1, E0=14, dE=2)
* %P
* radius: (m) Radius of circle in (x,y,0) plane where neutrons
*          are generated.
* dist: (m) Distance to target along z axis.
* xw: (m) Width(x) of target
* yh: (m) Height(y) of target
* E0: (meV) Mean energy of neutrons.
* dE: (meV) Energy spread of neutrons.
* Lambda0 (AA) Mean wavelength of neutrons.
* dLambda (AA) Wavelength spread of neutrons.
* flux (1/(s*cm**2*s)) Energy integrated flux
*
* %E
*****+
DEFINE COMPONENT Source_simple
DEFINITION PARAMETERS ()
SETTING PARAMETERS (radius, dist, xw, yh, E0=0, dE=0, Lambda0=0, dLambda=0, flux=1)
OUTPUT PARAMETERS ()
STATE PARAMETERS (x, y, z, vx, vy, vz, t, s1, s2, p)
DECLARE
|{
    double pmul, pdir;
}
INITIALIZE
|{
    pmul=flux*PI*1e4*radius*radius/mcget_ncount();
}

```

```

TRACE
|(
double chi,E,Lambda,v,r, xf, yf, rf, dx, dy;

t=0;
z=0;

chi=2*PI*rand01();                                /* Choose point on source */
r=sqrt(rand01())*radius;                          /* with uniform distribution. */
x=r*cos(chi);
y=r*sin(chi);

randvec_target_rect(&xf, &yf, &rf, &pdir,
    0, 0, dist, xw, yh, ROT_A_CURRENT_COMP);

dx = xf-x;
dy = yf-y;
rf = sqrt(dx*dx+dy*dy+dist*dist);

p = pdir*pmul;

if(Lambda0==0) {                                    /* Choose from uniform distribution */
    E=E0+dr*randpm1();
    v=sqrt(E)*SE2V;
} else {
    Lambda=Lambda0+dLambda*randpm1();
    v = K2V*(2*PI/Lambda);
}

vz=v*dist/rf;
vy=v*dy/rf;
vx=v*dx/rf;
}

MCDISPLAY
|(
    magnify("xy");
    circle("xy",0,0,0, radius);
)

END

```

Written by developers
and possibly you!

Generated c-code

```

/* Automatically generated file. Do not edit.
 * Format: ANSI C source code
 * Creator: McStas <http://neutron.risoe.dk>
 * Instrument: My_Instrument.instr (My Instrument)
 * Date: Sat Apr 9 15:27:56 2005
 */

/* THOUSANDS of lines removed here.... */

/* TRACE Component Source. */
SIG_MESSAGE("Source (Trace)");
mcDEBUG_COMP("Source")
mccoordschange(mcpoSource, mcrotrSource,
    &mcnlx, &mcnly, &mcnlz,
    &mcnlvx, &mcnlvy, &mcnlvz,
    &mcnlt, &mcnlsx, &mcnlsy);
mcDEBUG_STATE(mcnlx, mcnly, mcnlz, mcnlvx, mcnlvy, mcnlvz, mcnlt, mcnlsx, mcnlsy, mcnlp)
#define x mcnlx
#define y mcnly
#define z mcnlz
#define vx mcnlvx
#define vy mcnlvy
#define vz mcnlvz
#define t mcnlt
#define s1 mcnlsx
#define s2 mcnlsy
#define p mcnlp
STORE_NEUTRON(2,mcnlx, mcnly, mcnlz, mcnlvx,mcnlvy,mcnlvz,mcnlt,mcnlsx,mcnley, mcnlsz, mcnlp);
mcScattered=0;
mcNCounter[2]++;
#define mccompcurname Source
#define mccompcurindex 2
{
    /* Declarations of SETTING parameters. */
MCNUM radius = mccSource_radius;
MCNUM dist = mccSource_dist;
MCNUM xv = mccSource_xv;
MCNUM yh = mccSource_yh;
MCNUM EO = mccSource_EO;
MCNUM dr = mccSource_dE;
MCNUM Lambda0 = mccSource_Lambda0;
MCNUM dLambda = mccSource_dLambda;
MCNUM flux = mccSource_flux;
#line 58 "Source_simple.comp"
{
    double chi,E,Lambda,v,r, xf, yf, rf, dx, dy;

t=0;
z=0;

chi=2*PI*rand01();                                /* Choose point on source */
r=sqrt(rand01())*radius;                          /* with uniform distribution. */
x=r*cos(chi);
y=r*sin(chi);

randvec_target_rect(&xf, &yf, &rf, &pdix,
    0, 0, dist, xv, yh, ROT_A_CURRENT_COMP);
}

```

Written by mcstas!

McStas is a (pre)compiler!

Input is .comp and .instr files +
runtime functions for e.g. random
numbers

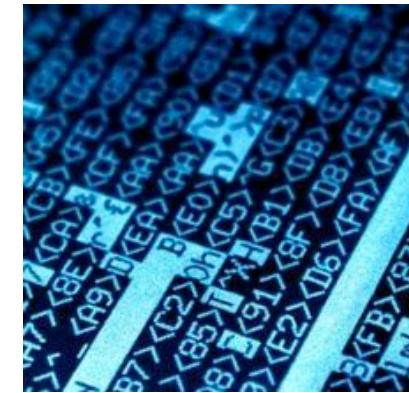
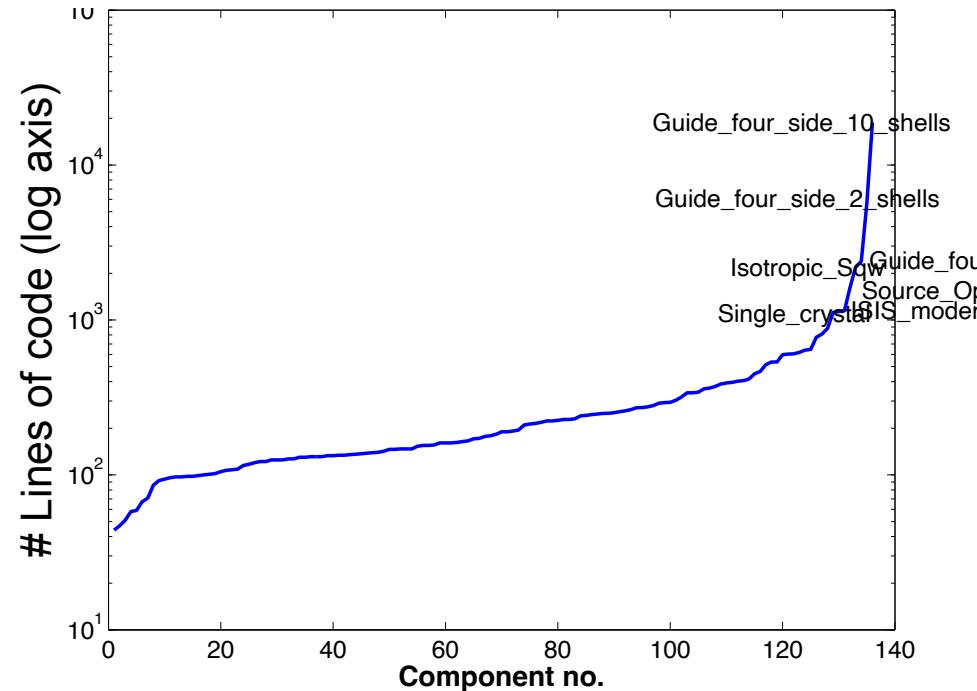
Output is a single c-file, which can
be compiled using e.g. gcc.

Can take input arguments if
needed.

Writing new comps or understanding existing is not that complex...

- Check our long list of components and look inside... Most of them are quite simple and short... Statistics:

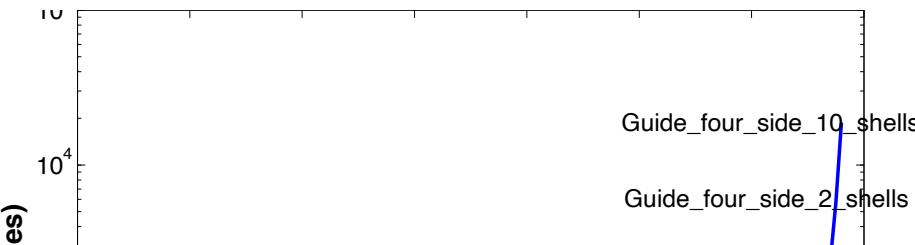
Number of lines of code per component - 240 comps in total



Writing new comps or understanding existing is not that complex...

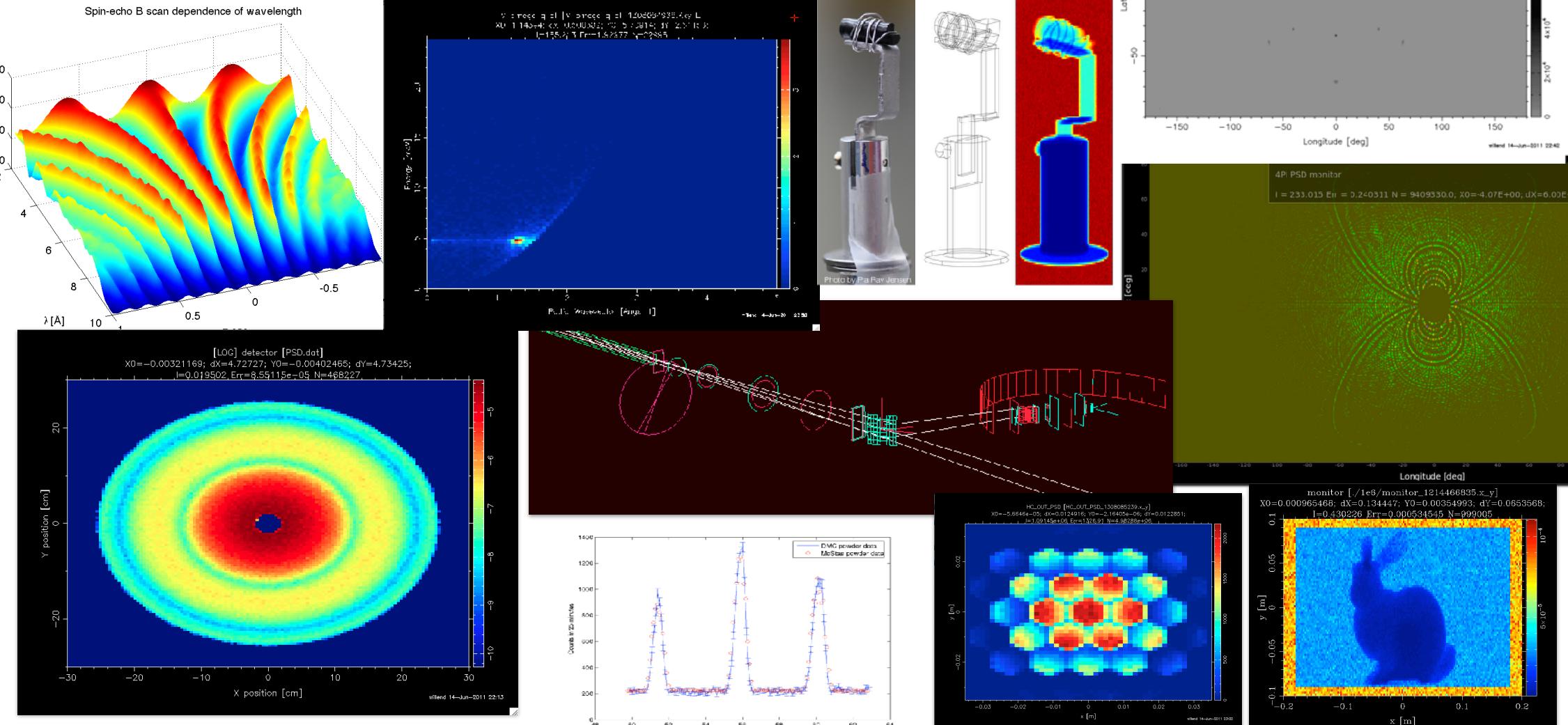
- Check our long list of components and look inside... Most of them are quite simple and short... Statistics:

Number of lines of code per component - 240 comps in total



- Well-developed community support
 - 30-40% of existing and new additions are from users
 - No direct refereeing of the code, but these requirements:
 - At least one test-instrument
 - Meaningful documentation headers (in-code docs)
 - Contributions go in dedicated contrib/ section of library

Example suite: ~252 instruments





**THIS IS NOT
THE END
IT'S JUST
THE BEGINNING**

