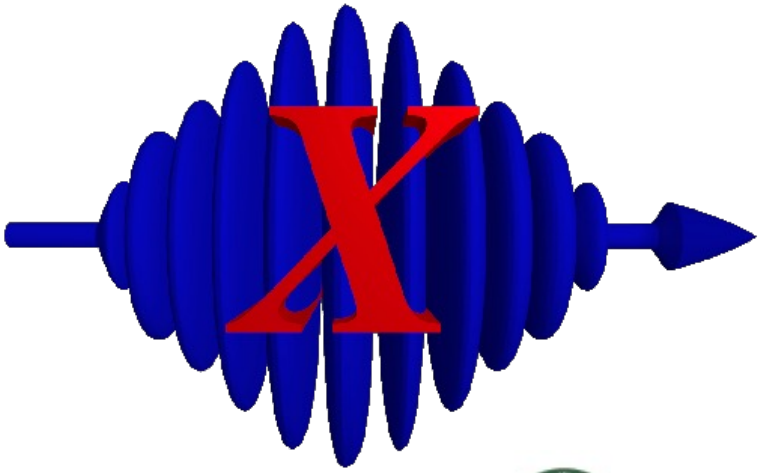
McXtrace



Advanced grammar

Peter Willendrup (pkwi@fysik.dtu.dk)

At the beam-line description level

- 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!

At the beam-line description level

- 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

At the computing/execution level

- Use **MPI** parallelisation - included in macOS install, easy to get on Linux…
- The **Intel C** compiler is known to give ~factor of 2 wrt. gcc in most cases
- Use **GPU's** (McXtrace 3.x) with Nvidia compiler.
- Still consider if you are asking the right question if runtimes reach days/weeks…

SPLIT, EXTEND, WHEN, COPY & GROUP

(… and the dangerous JUMP)

- Advanced language features:

Tips and tricks for your instrument

Syntax in one, complex view...

McXtrace

SOLEIL SYNCHROTRON

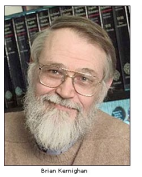
{SPLIT} COMPONENT name = comp(parameters) {WHEN condition}
AT (...) [RELATIVE [reference|PREVIOUS] | ABSOLUTE]
{ROTATED {RELATIVE [reference|PREVIOUS] | ABSOLUTE} }
{GROUP group_name}
{EXTEND C_code}
{JUMP [reference|PREVIOUS|MYSELF|NEXT] [ITERATE number_of_times
| WHEN condition] }

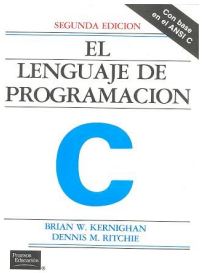
DECLARE / INITIALIZE

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

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

K & R. / GNU





- 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! - McXtrace placement specifiers work in degrees, etc...

DECLARE / INITIALIZE

Useful physics constants:

```
#define CELE      1.602176487e-19   /* [C] Elementary charge CODATA 2006*/
#define M_C       299792458         /* [m/s] speed of light CODATA 2006*/
#define E2K       0.506773091264796 /* Convert k[1/AA] to E [keV] (CELE/(HBAR*M_C)*1e-10)*1e3 */
#define K2E       1.97326972808327  /*Convert E[keV] to k[1/AA] (1e10*M_C*HBAR/CELE)/1e3 */
#define RE        2.8179402894e-5   /*[AA] Thomson scattering length*/
```

plus e.g. DEG2RAD, RAD2DEG, and these math constants

```
( #  define PI 3.14159265358979323846 )
 #  define M_PI PI
 #  define M_PI_2 M_PI/2.0
 #  define M_PI_4 M_PI/4.0
 #  define M_1_PI 1.0/M_PI
 #  define M_2_PI 2*M_1_PI
 #  define M_2_SQRTPI 2/sqrt(M_PI)
 #  define M_SQRT2 sqrt(2)
 #  define M_SQRT1_2 sqrt(1/2)
 # endif
```

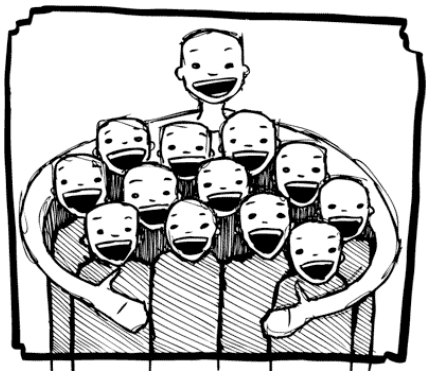
plus anything you can imagine in terms of trigonometric functions from C <math.h>
(beware, these take radians as input - as opposed to our ROTATED statements)

%include

- Instrumentfiles can include external c-code or other instrumentfiles... (These are examples from McStas)

- 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



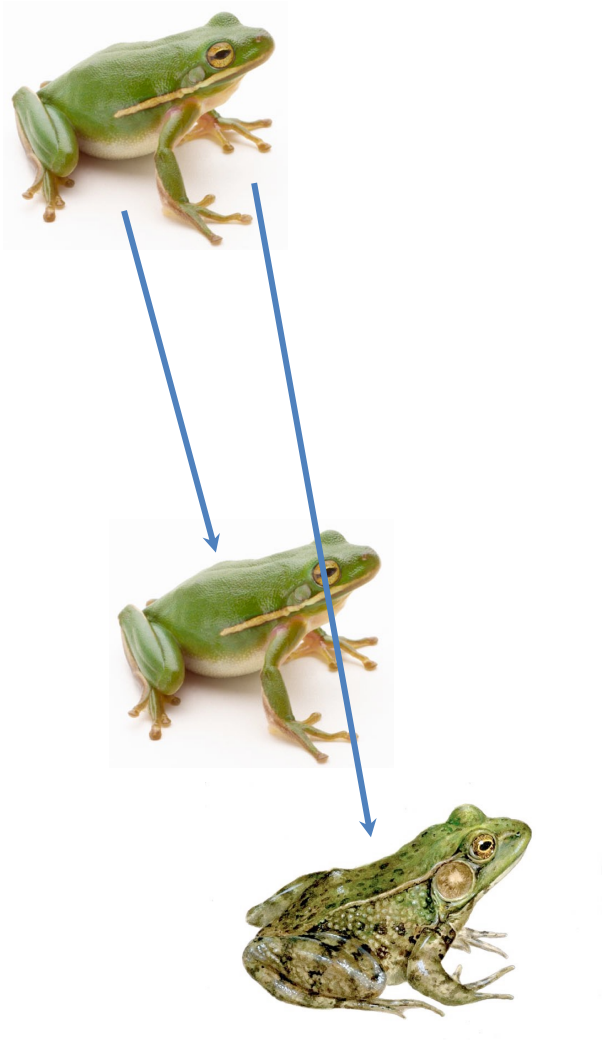
COPY- inside instruments

- In instruments: (see e.g. 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 **MYcopy** = **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



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 particle is "tried" in sequential order until the it becomes SCATTERED.

WHEN

- Syntax:

COMPONENT Mine = Yours(blah, blah)

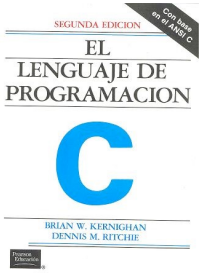
WHEN (c-expression) AT (....)

- 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 X-ray 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(...)

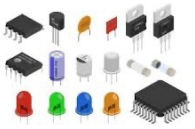
WHEN (myvar==1) AT (0,0,0) RELATIVE Sample




Brian Kernighan


Dennis Ritchie

EXTEND



McXtrace

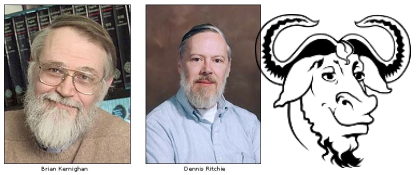
SOLEIL SYNCHROTRON

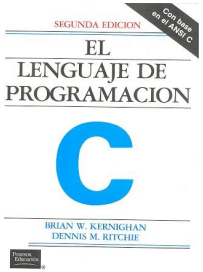
- Enrich component behaviour using EXTEND:

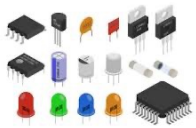
```
COMPONENT Mono1 = Monochromator_curved(...)
AT (0,0, -LMM) RELATIVE Cradle ROTATED (0,A1/2,0) RELATIVE Cradle
GROUP IN6Monoks
EXTEND %{
  if (SCATTERED) { myvar = 1; }
%}

...
COMPONENT Mono2 = Monochromator_curved(...)
AT (0,0, 0) RELATIVE Cradle ROTATED (0,A2/2,0) RELATIVE Cradle
GROUP IN6Monoks
EXTEND
%{
  if (SCATTERED) { myvar = 2 ;}
%}
```

K & R. / GNU







Combined example: Decompose multiple scattering from Single_crystal

DECLARE %{

double multiple_scatt;

%}

…

COMPONENT Crystal = Single_crystal(… order=0 …)

AT (0,0,0) RELATIVE somewhere

EXTEND %{

multiple_scatt=SCATTERED;

%}

…

COMPONENT PSD_single=PSD_monitor(…)

WHEN (multiple_scatt==1) AT (0,0,0) RELATIVE somewhere_else

COMPONENT PSD_multiple=PSD_monitor(…)

WHEN (multiple_scatt > 1) AT (0,0,0) RELATIVE somewhere_else

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 X-rays...)

Syntaxes:

- COMPONENT a=b(...)
- **WHEN** (expr) AT (...) **JUMP** arm_somewhere

- COMPONENT a=b(...)
- **WHEN** (expr) AT (...) **JUMP** myself

JUMP

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

- **JUMP** to myself
- **JUMP** to an Arm

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 X-rays...)

Syntaxes:

- COMPONENT a=b(...)
- **WHEN** (expr) AT (...) **JUMP** arm_somewhere

- COMPONENT a=b(...)
- **WHEN** (expr) AT (...) **JUMP** myself

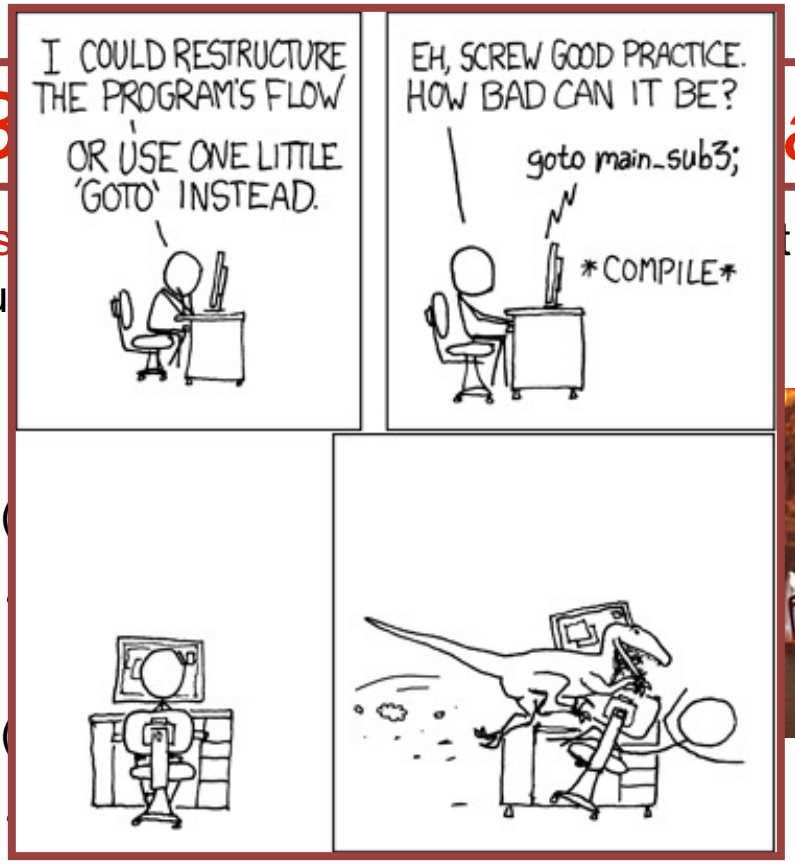
JUMP

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

- **JUMP** to myself
- **JUMP** to an Arm

B a GOTO!

- No coordinate trans t if the Arms you JUMP between do not coincide you

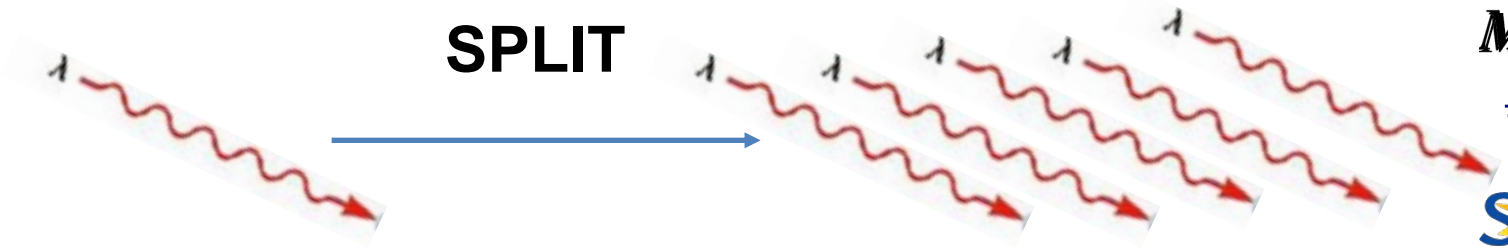


Syntaxes:

- COMPONENT a=b(
- **WHEN** (expr) AT (...

- COMPONENT a=b(
- **WHEN** (expr) AT (...

SPLIT



Increase statistics beyond this point in the instrumentfile

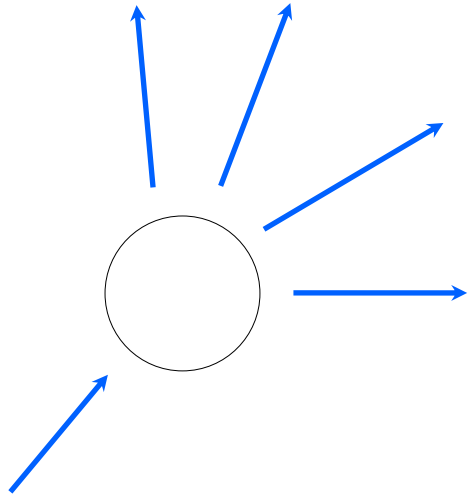
- **SPLIT** n MyArm = Arm()
- AT somewhere

will “formulate an if-statement”:

for j=1:n

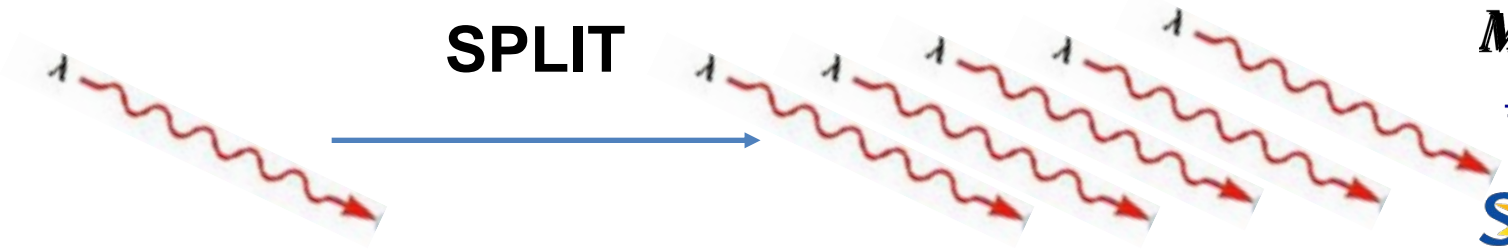
- comp1
- comp2
- comp3
- ...

end (of instrument)



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

SPLIT



Increase statistics beyond this point in the instrumentfile

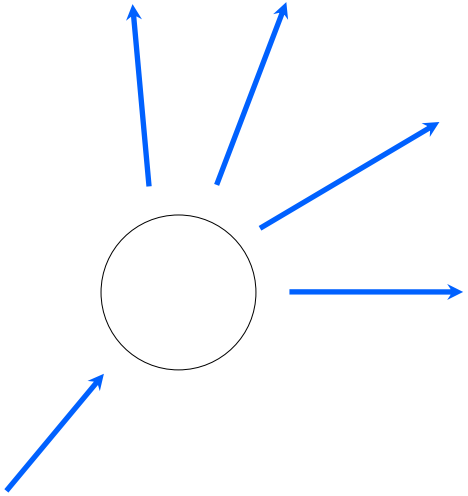
- **SPLIT** n MyArm = Arm()
- AT somewhere

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...

Works very well together with e.g. monochromators, Single_crystal, PowderN