Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

Beams Department

MAD-NG – *a standalone multiplatform tool for non-linear optics design and optimisation*.

14th International Computational Accelerator Physics Conference.

Laurent Deniau

CERN-BE/ABP

2-5 October 2024 – Seeheim-Jugenheim

MAD-NG's features

- **Long term design: easy to use and extend.**
  - ➡ **Flexible language ➠ fast, simple, and general purpose scripting language.**
    - ~70% of the code is written in the Lua(JIT) scripting language, ~30% in C and C++.
  - ➡ **Flexible technologies ➠ self-contained, all-in-one and modular application.**
    - Single "download & run" binary application, no dependencies (requires Gnuplot for plotting).
  - ➡ **Efficient & Portable technologies ➠ embeds a Tracing Just in Time compiler.**
    - Same results everywhere (LNX, OSX, WIN), extensive unit tests (>8000) and examples.
    - Extremely simple and fast Foreign Function Interface (FFI) to C, C++, Fortran, etc…
  - ➡ **Easy to extend & support ➠ embeds an online profiler and debugger.**
    - Adding and debugging new elements with new physics take less than a day.
- **5D & 6D physics using high-order differential algebra and symplectic integrators.**
    - Combined physics, combined elements, combined misalignments, local & global frames, slicing.
    - True RBend, Exact SBend (curved multipoles), Solenoid, Fringe fields for ALL, Patches, etc…
    - Physics & Maths in Lua and C/C++, **performance is x10-x80 faster than MADX-PTC.**
- **Open source software.**
  - ➡ License GPLv3, User manual (~200p, covers <25%), Lua Manual (29p).
  - ➡ Releases & Manual https://cern.ch/mad/releases/madng/
  - ➡ Online Manual https://cern.ch/mad/releases/madng/html/
  - ➡ Repository https://github.com/MethodicalAcceleratorDesign/MAD

MAD-NG's features

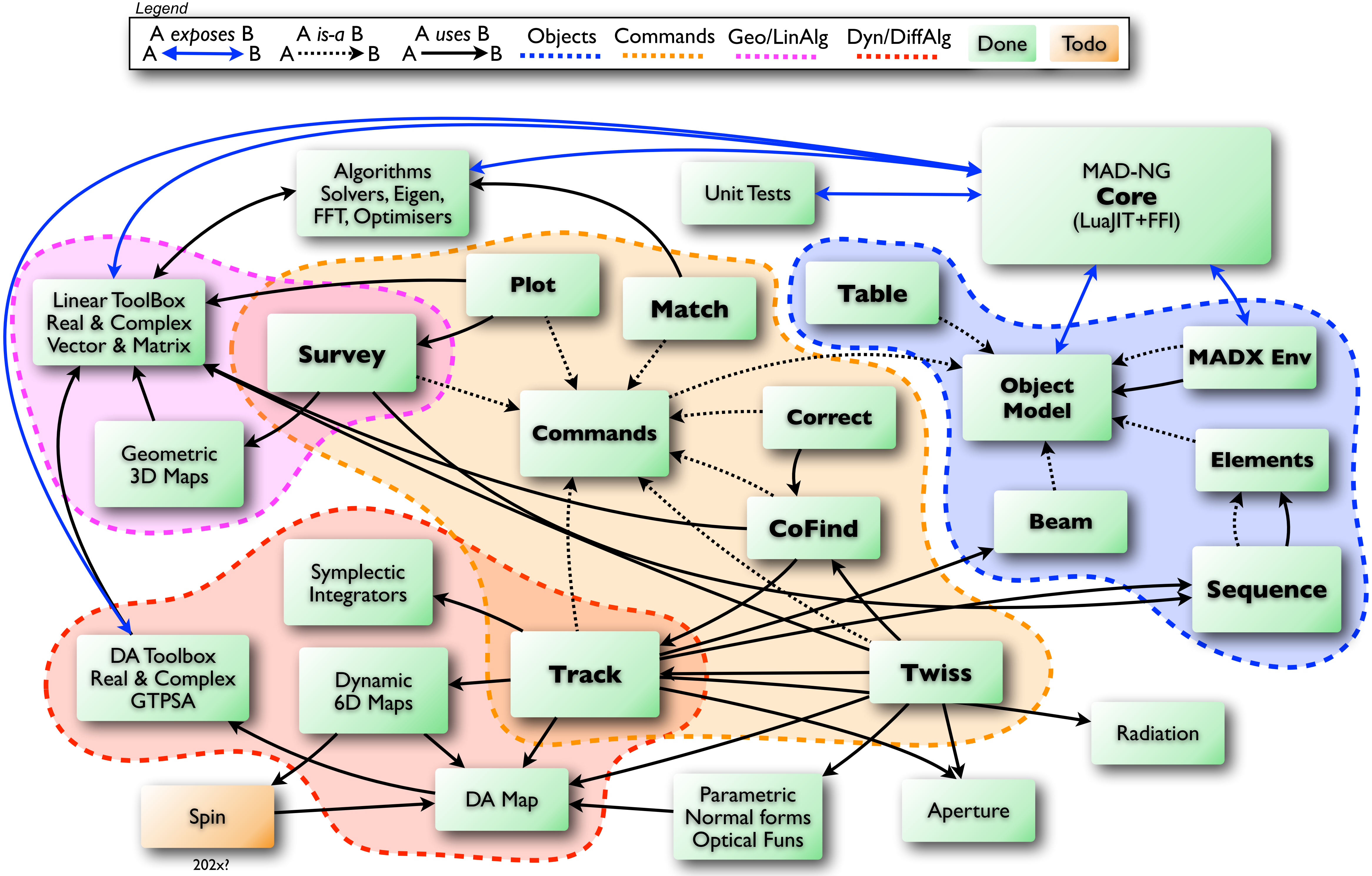
Development started in 2016

- **Long term design: easy to use and extend.**
  - ➡ **Flexible language** ➠ **fast, simple, and general purpose scripting langu**
    - ~70% of the code is written in the Lua(JIT) scripting language, ~30% in C and C++.
  - ➡ **Flexible technologies** ➠ **self-contained, all-in-one and modular application.**
    - Single "download & run" binary application, no dependencies (requires Gnuplot for plotting).
  - ➡ **Efficient & Portable technologies** ➠ **embeds a Tracing Just in Time compiler.**
    - Same results everywhere (LNX, OSX, WIN), extensive unit tests (>8000) and examples.
    - Extremely simple and fast Foreign Function Interface (FFI) to C, C++, Fortran, etc…
  - ➡ **Easy to extend & support** ➠ **embeds an online profiler and debugger.**
    - Adding and debugging new elements with new physics take less than a day.
- **5D & 6D physics using high-order differential algebra and symplectic integrators.**
    - Combined physics, combined elements, combined misalignments, local & global frames, slici
    - True RBend, Exact SBend (curved multipoles), Solenoid, Fringe fields for ALL, Patc
    - Physics & Maths in Lua and C/C++, **performance is x10-x80 faster than MADX-**

First Twiss on LHC B1 & B2 in 2018

- **Open source software.**
  - ➡ License GPLv3, User manual (~200p, covers <25%), Lua Manual (29p).
  - ➡ Releases & Manual https://cern.ch/mad/releases/madng/
  - ➡ Online Manual https://cern.ch/mad/releases/madng/html/
  - ➡ Repository https://github.com/MethodicalAcceleratorDesign/MAD

MAD-NG ecosystem



Legend

A *exposes* B A ⟷ B
A *is-a* B A ┈┈▶ B
A *uses* B A ──▶ B
Objects
Commands
Geo/LinAlg
Dyn/DiffAlg
Done
Todo

MAD-NG **Core** (LuaJIT+FFI)

Unit Tests

Algorithms Solvers, Eigen, FFT, Optimisers

Linear ToolBox Real & Complex Vector & Matrix

Geometric 3D Maps

Plot

Match

Survey

Commands

Correct

CoFind

Table

Object Model

MADX Env

Elements

Beam

Sequence

Symplectic Integrators

Track

Twiss

Radiation

DA Toolbox Real & Complex GTPSA

Dynamic 6D Maps

DA Map

Spin

202x?

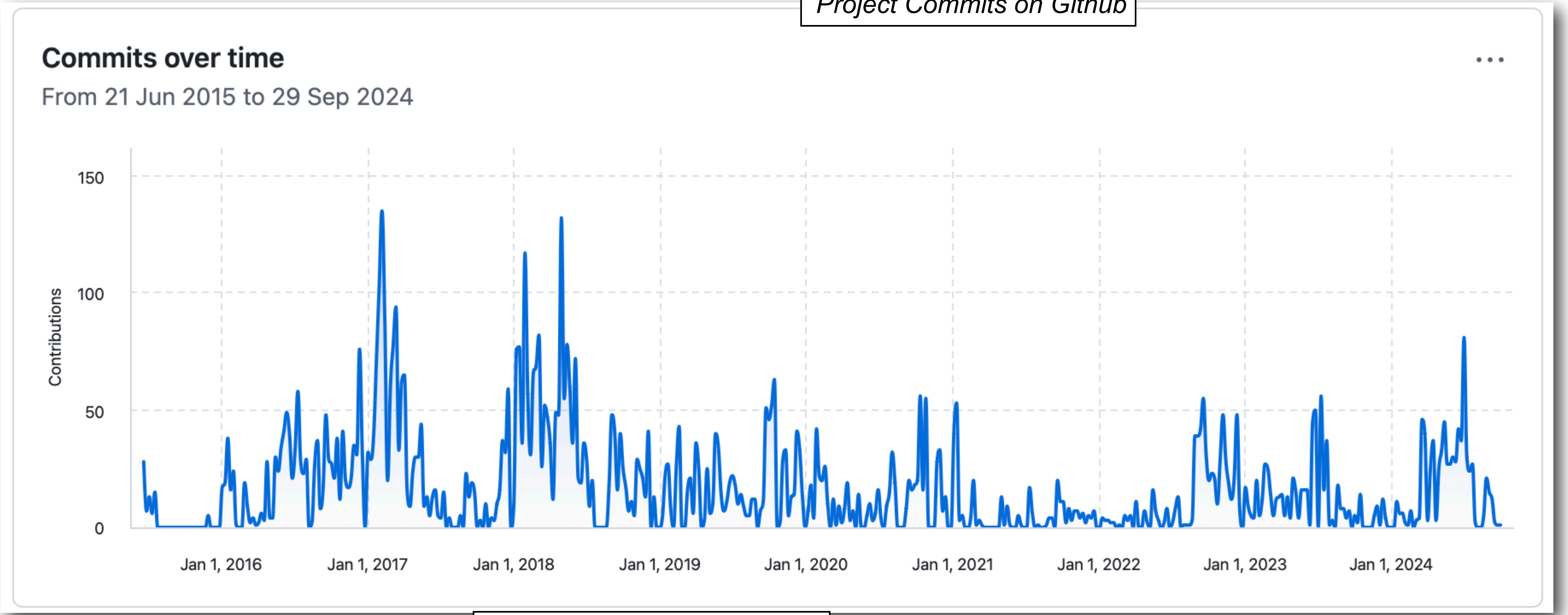
Parametric Normal forms Optical Funs

Aperture

MAD-NG development

Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

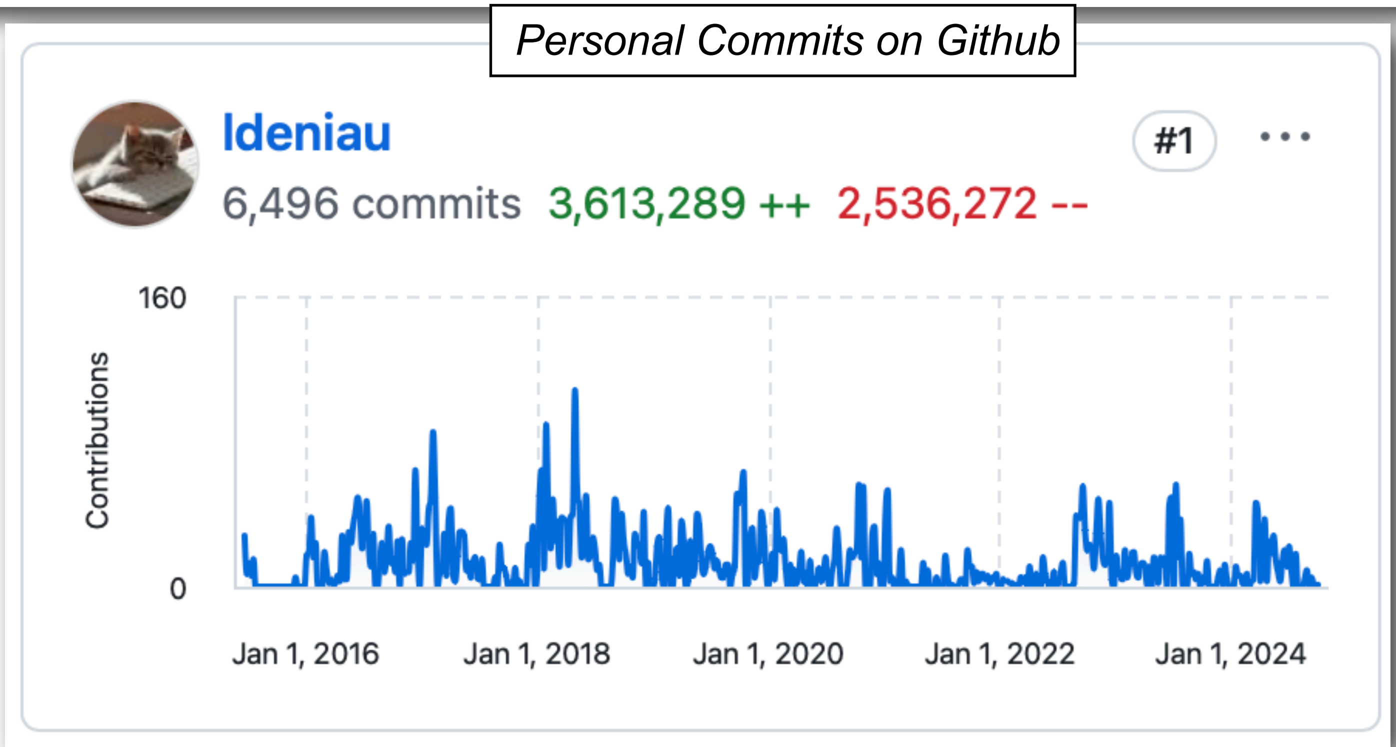
Project Commits on Github



Commits over time

From 21 Jun 2015 to 29 Sep 2024

Personal Commits on Github



ldeniau #1

6,496 commits 3,613,289 ++ 2,536,272 --

~60000 lines of code
~20000 lines in C/C++

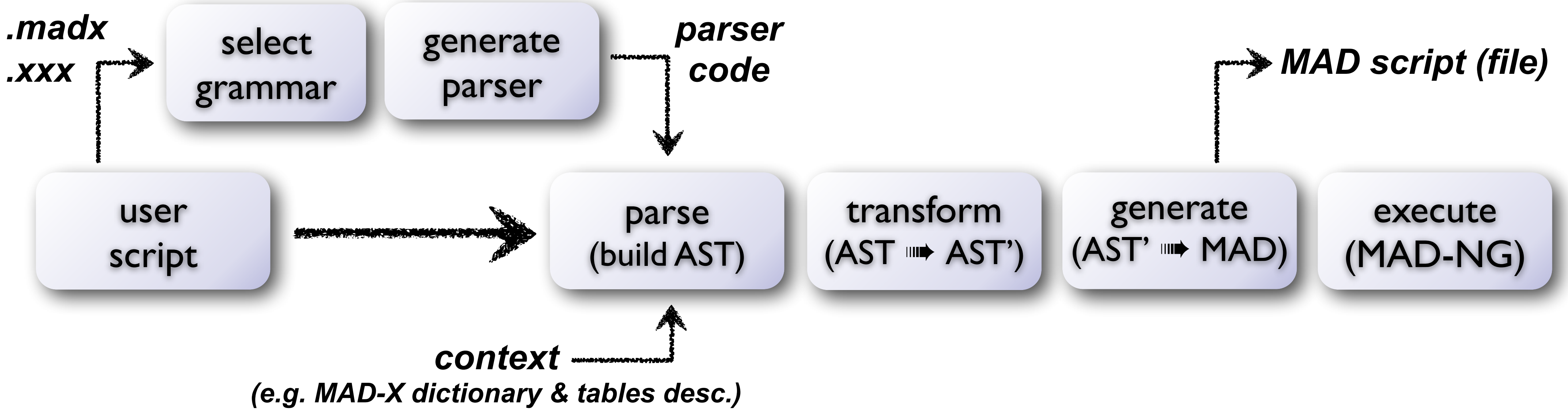
PART I
Design

MAD-X (MAD8) to MAD-NG: *files import & sharing*

- MAD-NG **loads and converts** MAD-X and MAD8 sequences, elements and variables, ***including deferred expressions***, **on-the-fly** into the MADX workspace (a MAD-NG context that emulates MAD-X global workspace) and optionally save conversion to files.

```
! convert MAD-X files on need, save to MAD file (disk), load to MADX workspace (memory)
  MADX:load('lhc_as-built.seq'        , 'lhc_as-built.mad')
  MADX:load('opticsfile.22_ctpps2'    , 'opticsfile.22_ctpps2.mad')
  MADX:load('FCCee_z_213_nosol_18.seq', 'FCCee_z_213_nosol_18.mad')
```

- MAD-NG embeds technologies to **parse arbitrary language** that can be **described with PEG** (parser expression grammar > RegEx) to generate AST (abstract syntax tree), and apply transformations and/or evaluations, translating > 400 000 lines/sec.



.madx
.xxx

select grammar

generate parser

parser code

MAD script (file)

user script

parse (build AST)

transform (AST ➠ AST')

generate (AST' ➠ MAD)

execute (MAD-NG)

context
(e.g. MAD-X dictionary & tables desc.)

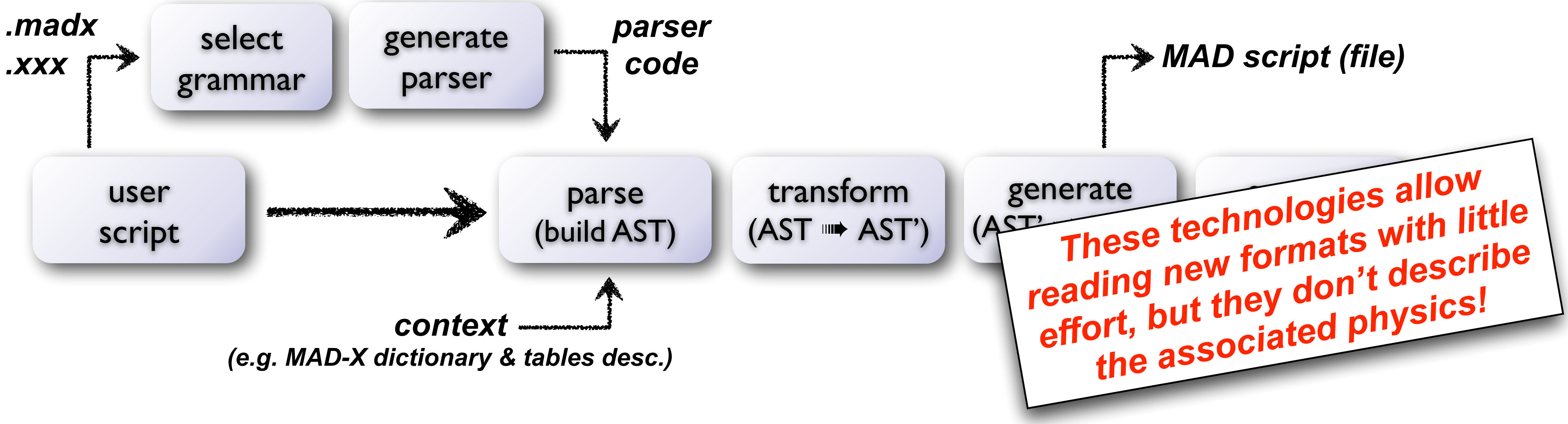
- MAD-NG can load MAD-X as a shared library (not as a subprocess like CPyMad) with direct fast access to MAD-X sequences, elements and variables, running at the speed of MAD-X itself (i.e. MAD-NG & FFI are faster than MAD-X).

MAD-X (MAD8) to MAD-NG: *files import & sharing*

- MAD-NG **loads and converts** MAD-X and MAD8 sequences, elements and variables, ***including deferred expressions***, **on-the-fly** into the MADX workspace (a MAD-NG context that emulates MAD-X global workspace) and optionally save conversion to files.

```
! convert MAD-X files on need, save to MAD file (disk), load to MADX workspace (memory)
  MADX:load('lhc_as-built.seq'         , 'lhc_as-built.mad')
  MADX:load('opticsfile.22_ctpps2'     , 'opticsfile.22_ctpps2.mad')
  MADX:load('FCCee_z_213_nosol_18.seq', 'FCCee_z_213_nosol_18.mad')
```

- MAD-NG embeds technologies to **parse arbitrary language** that can be **described with PEG** (parser expression grammar > RegEx) to generate AST (abstract syntax tree), and apply transformations and/or evaluations, translating > 400 000 lines/sec.



These technologies allow reading new formats with little effort, but they don't describe the associated physics!

- MAD-NG can load MAD-X as a shared library (not as a subprocess like CPyMad) with direct fast access to MAD-X sequences, elements and variables, running at the speed of MAD-X itself (i.e. MAD-NG & FFI are faster than MAD-X).

Sequences & elements: *flexible optics design*

- **Lattices definition as simple as in MAD-X but more flexible** *(syntax is very close)*
  - ➡ Sequences are **containers** (e.g. access elements) that can store arbitrary objects.
    - E.g. can store their **beam** or their own list of **knobs**.
  - ➡ Elements are **containers** (e.g. access attributes) that can store arbitrary objects.
  - ➡ Sequence can include **subsequences**, **beam lines, elements** (and **subelements**).
  - ➡ **Operator overloading** (+, −, ×) allows to create *sequences* with the flexibility of *lines*.
  - ➡ Names are optional and can be non-unique with support for *relative* or *absolute* counts.
    - Positions 'AT' can be absolute or relative 'FROM' names with absolute or relative counts.
- **Manage arbitrary number of sequences to model an entire accelerators complex.**
  - ➡ **Shared sequences**, e.g. LHC B1 & B2.
    - Provides few sharing policies and name mangling.
  - ➡ **Chained sequences**, e.g. Linac4+PSB+PS+SPS+LHC.
  - ➡ **Conditionally chained sequences** (e.g. RaceTrack).
    - Based on special ***s-link* element**
    - Conditions for transition and lattice connections are performed through arbitrary user-defined functions.

SPS in MAD-X

```
SPS:     LINE = (6*SUPER);
SUPER:   LINE = (7*P44,INSERT,7*P44);
INSERT:  LINE = (P24,2*P00,P42);
P00:     LINE = (QF,DL,QD,DL);
P24:     LINE = (QF,DM,2*B2,DS,PD);
P42:     LINE = (PF,QD,2*B2,DM,DS);
P44:     LINE = (PF,PD);
PD:      LINE = (QD,2*B2,2*B1,DS);
PF:      LINE = (QF,2*B1,2*B2,DS);
```

```
pf     = bline {qf,2*b1,2*b2,ds}
pd     = bline {qd,2*b2,2*b1,ds}
p24    = bline {qf,dm,2*b2,ds,pd}
p42    = bline {pf,qd,2*b2,dm,ds}
p00    = bline {qf,dl,qd,dl}
p44    = bline {pf,pd}
insert = bline {p24,2*p00,p42}
super  = bline {7*p44,insert,7*p44}
SPS    = sequence 'SPS' {6*super}
```

SPS in MAD-NG

Sequences & elements: *flexible optics design*

- **Lattices definition as simple as in MAD-X but more flexible** *(syntax is very close)*
  - ➡ Sequences are **containers** (e.g. access elements) that can store arbitrary obj
    - E.g. can store their **beam** or their own list of **knobs**.
  - ➡ Elements are **containers** (e.g. access attributes) that can store arbi
  - ➡ Sequence can include **subsequences**, **beam lines, elements** (and **subelements**).
  - ➡ **Operator overloading** (+, −, ×) allows to create *sequences* with the flexibility of *lines*.
  - ➡ Names are optional and can be non-unique with support for *relative* or *absolute* counts.
    - Positions ‘AT' can be absolute or relative ‘FROM’ names with absolute or relative counts.

Unified definitions of lines and sequences with some extensions

- **Manage arbitrary number of sequences to model an entire accelerators complex.**
  - ➡ **Shared sequences**, e.g. LHC B1 & B2.
    - Provides few sharing policies and name mangling.
  - ➡ **Chained sequences**, e.g. Linac4+PSB+PS+SPS+LHC.
  - ➡ **Conditionally chained sequences** (e.g. RaceTrack).
    - Based on special ***s-link*** **element**
    - Conditions for transition and lattice connections are performed through arbitrary user-defined functions.

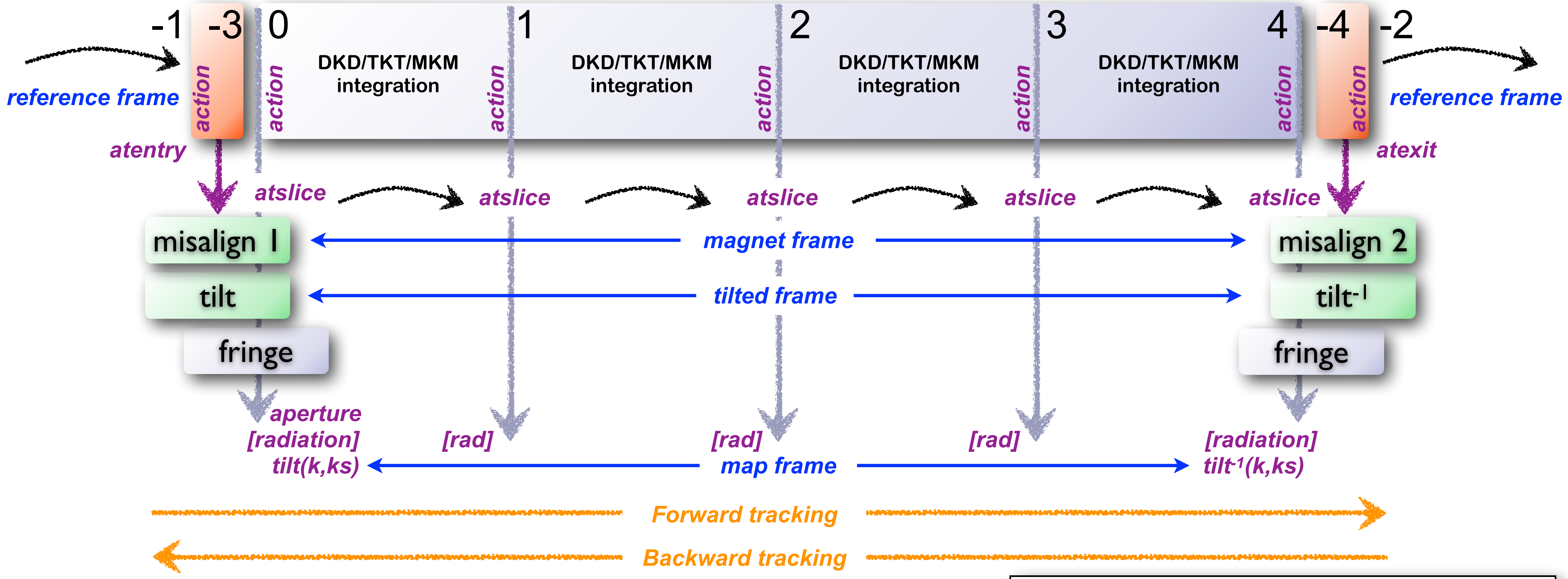
SPS in MAD-X

```
SPS:    LINE = (6*SUPER);
SUPER:  LINE = (7*P44,INSERT,7*P44);
INSERT: LINE = (P24,2*P00,P42);
P00:    LINE = (QF,DL,QD,DL);
P24:    LINE = (QF,DM,2*B2,DS,PD);
P42:    LINE = (PF,QD,2*B2,DM,DS);
P44:    LINE = (PF,PD);
PD:     LINE = (QD,2*B2,2*B1,DS);
PF:     LINE = (QF,2*B1,2*B2,DS);
```

```
pf     = bline {qf,2*b1,2*b2,ds}
pd     = bline {qd,2*b2,2*b1,ds}
p24    = bline {qf,dm,2*b2,ds,pd}
p42    = bline {pf,qd,2*b2,dm,ds}
p00    = bline {qf,dl,qd,dl}
p44    = bline {pf,pd}
insert = bline {p24,2*p00,p42}
super  = bline {7*p44,insert,7*p44}
SPS    = sequence 'SPS' {6*super}
```

SPS in MAD-NG

Element tracking: *frames, slices & actions*



```
atentry(elm, mflw,  sdir)
misalgn(elm, mflw,  sdir)
tilt   (ang, mflw,  sdir)
fringe (elm, mflw,  sdir)  DKD/TKT/MKM
integr (elm, mflw,     1, thick, kick)
fringe (elm, mflw, -sdir)      atslice
tilt   (ang, mflw, -sdir)
misalgn(elm, mflw, -sdir)
atexit (elm, mflw, -sdir)
```

- **Slicing** can be uniform or arbitrary (array, function).
- **Subelements** (thick or thin) can be inserted at arbitrary relative or absolute positions inside the parent element.
- **Installing** elements in sequence automatically (user choice) insert them as subelement upon collision.
- **Misalignments** (elements vs sequences) restore the frames on exit. *Permanent* misalignments (element property) use **patches**. Survey can consider misalignments (user choice) for global motion inside elements.

Actions: *customise Survey, Track, Cofind & Twiss…*

- **Actions are functions**
  - ➡ MAD-NG functions are *first class lexical closures* (fun & env) and can do everything…
    - i.e. **high order functions that can receive and return multiple arguments.**
  - ➡ actions kinds: `atentry`, `atslice`, `atexit`, *`ataper`*, *`atsave`*, *`atdebug`*.
  - ➡ **mechanism to customise or extend commands** (e.g. **Twiss** with **Track** and **Cofind**).
- Actions can be **combined** with combinators (and selectors).
  - chain(f_1,f_2) ➠ f_1() **;** return f_2().
  - **a**chain(f_1,f_2) ➠ return f_1() **and** f_2().
  - **o**chain(f_1,f_2) ➠ return f_1() **or** f_2().
  - compose(f_1,f_2) ➠ return f_1(f_2()).
  - ftrue, ffalse, fnil.
- Actions can be **selected** by *selectors*:
  - ➡ Selectors are functions to enable/disable actions based on some particular criteria e.g. slices number or any other user-defined criteria.
    24 predefined selectors: atall, atentry, atbegin, atbody, atbound, atend, atexit, atmid, atcore, atstd, actionat, etc…
- **Actions are triggered by Survey and Track engines during tracking**
  - ➡ actions are `chained` so they are independent from each other.
  - ➡ default for *ataper*: check for aperture *at slice* 0 (**titled frame**).
  - ➡ default for *atsave*: save data *at exit* (**reference frame**).

Actions: *customise Survey, Track, Cofind & Twiss…*

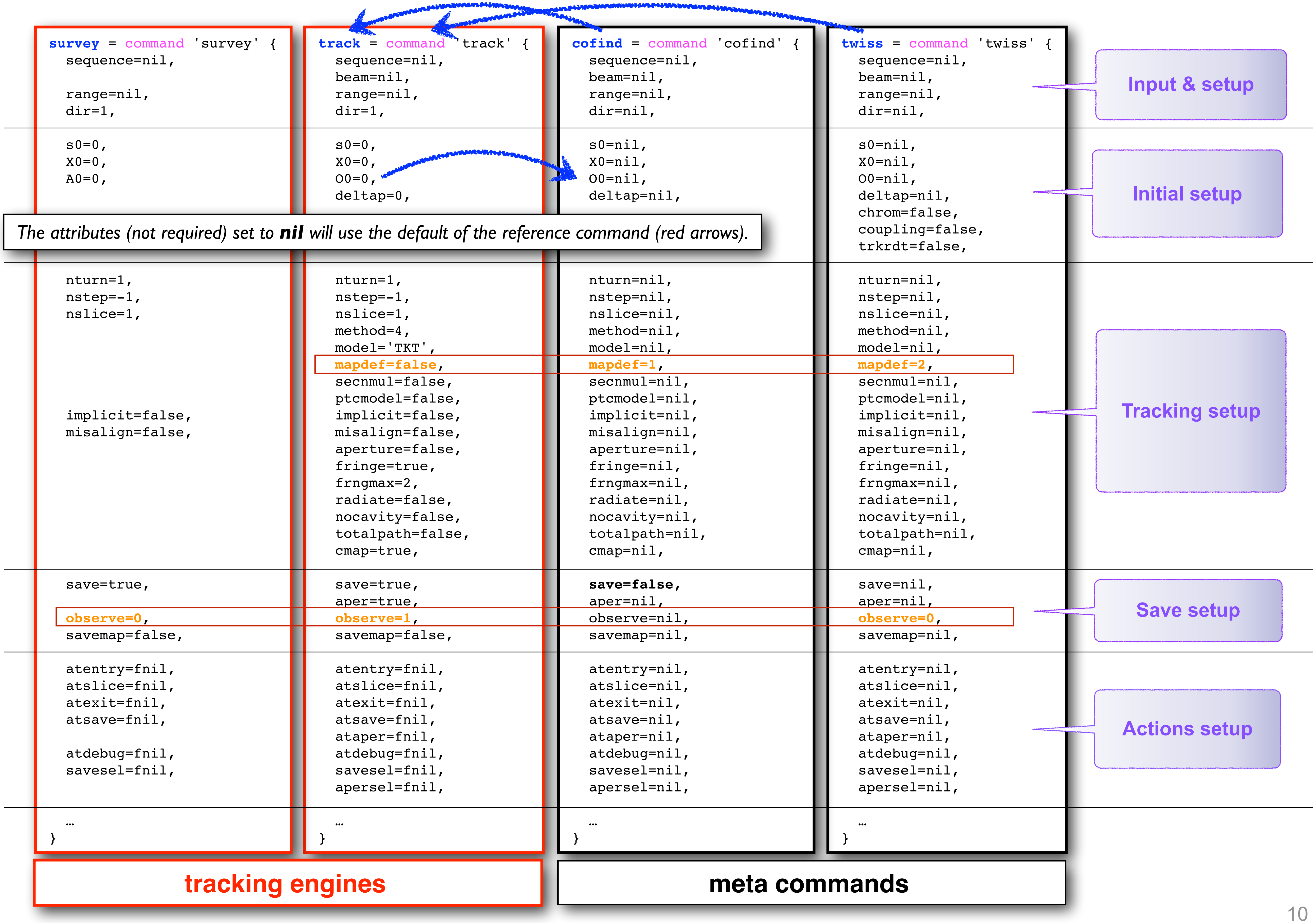
- **Actions are functions**
  - ➡ MAD-NG functions are *first class lexical closures* (fun & env) and can do everything…
    - i.e. **high order functions that can receive and return multiple arguments.**
  - ➡ actions kinds: `atentry`, `atslice`, `atexit`, *`ataper`*, *`atsa`*
  - ➡ **mechanism to customise or extend comm**
- Actions can be **combined** with combinators
  - chain(f_1,f_2) ➠ f_1() **;** return f_2().
  - **a**chain(f_1,f_2) ➠ return f_1() **and** f_2().
  - **o**chain(f_1,f_2) ➠ return f_1() **or** f_2().
  - compose(f_1,f_2) ➠ return f_1(f_2()).
  - ftrue, ffalse, fnil.

Actions are lambda functions extending Survey and Track tracking engines:
- radiation and aperture check are actions
- optic calculation (twiss) are actions
- saving data to MTable (TFS) are actions
- connecting sequences for parallel tracking
- replace, extend or wrap computations
- add extra physics locally or globally
- add multi-particules or DA maps physics
- etc…

- Actions can be **selected** by *selectors*:
  - ➡ Selectors are functions to enable/disable actions based on some particular criteria e.g. slices number or any other user-defined criteria.

    24 predefined selectors: atall, atentry, atbegin, atbody, atbound, atend, atexit, atmid, atcore, atstd, actionat, etc…
- **Actions are triggered by Survey and Track engines during tracking**
  - ➡ actions are `chained` so they are independent from each other.
  - ➡ default for *ataper*: check for aperture *at slice* 0 (**titled frame**).
  - ➡ default for *atsave*: save data *at exit* (**reference frame**).

Unified interfaces: *arbitrary complex combination*



```
survey = command 'survey' {
  sequence=nil,

  range=nil,
  dir=1,

  s0=0,
  X0=0,
  A0=0,





  nturn=1,
  nstep=-1,
  nslice=1,








  implicit=false,
  misalign=false,








  save=true,

  observe=0,
  savemap=false,

  atentry=fnil,
  atslice=fnil,
  atexit=fnil,
  atsave=fnil,

  atdebug=fnil,
  savesel=fnil,


  …
}
```

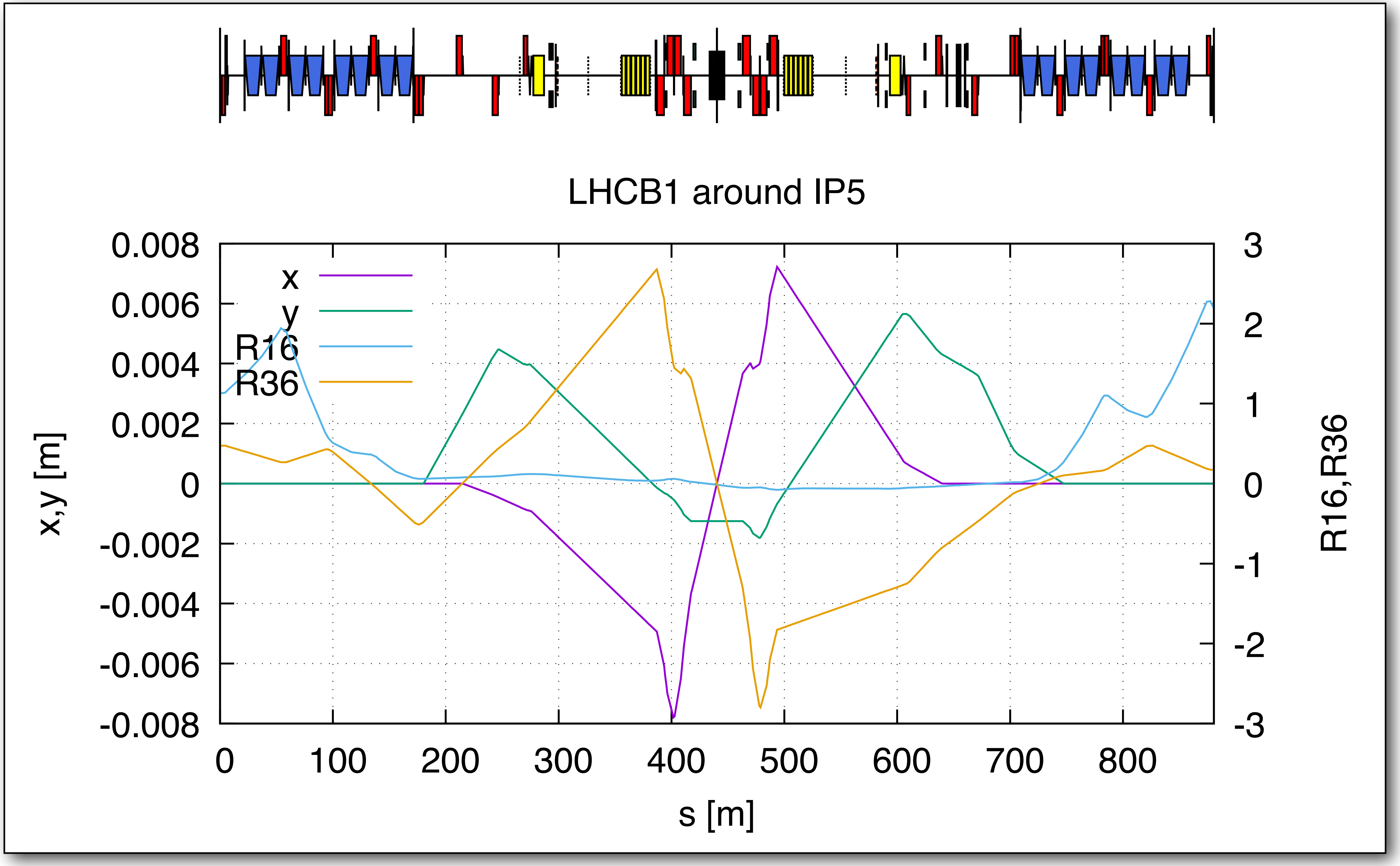
```
track = command 'track' {
  sequence=nil,
  beam=nil,
  range=nil,
  dir=1,

  s0=0,
  X0=0,
  O0=0,
  deltap=0,




  nturn=1,
  nstep=-1,
  nslice=1,
  method=4,
  model='TKT',
  mapdef=false,
  secnmul=false,
  ptcmodel=false,
  implicit=false,
  misalign=false,
  aperture=false,
  fringe=true,
  frngmax=2,
  radiate=false,
  nocavity=false,
  totalpath=false,
  cmap=true,

  save=true,
  aper=true,
  observe=1,
  savemap=false,

  atentry=fnil,
  atslice=fnil,
  atexit=fnil,
  atsave=fnil,
  ataper=fnil,
  atdebug=fnil,
  savesel=fnil,
  apersel=fnil,

  …
}
```

```
cofind = command 'cofind' {
  sequence=nil,
  beam=nil,
  range=nil,
  dir=nil,

  s0=nil,
  X0=nil,
  O0=nil,
  deltap=nil,




  nturn=nil,
  nstep=nil,
  nslice=nil,
  method=nil,
  model=nil,
  mapdef=1,
  secnmul=nil,
  ptcmodel=nil,
  implicit=nil,
  misalign=nil,
  aperture=nil,
  fringe=nil,
  frngmax=nil,
  radiate=nil,
  nocavity=nil,
  totalpath=nil,
  cmap=nil,

  save=false,
  aper=nil,
  observe=nil,
  savemap=nil,

  atentry=nil,
  atslice=nil,
  atexit=nil,
  atsave=nil,
  ataper=nil,
  atdebug=nil,
  savesel=nil,
  apersel=nil,

  …
}
```

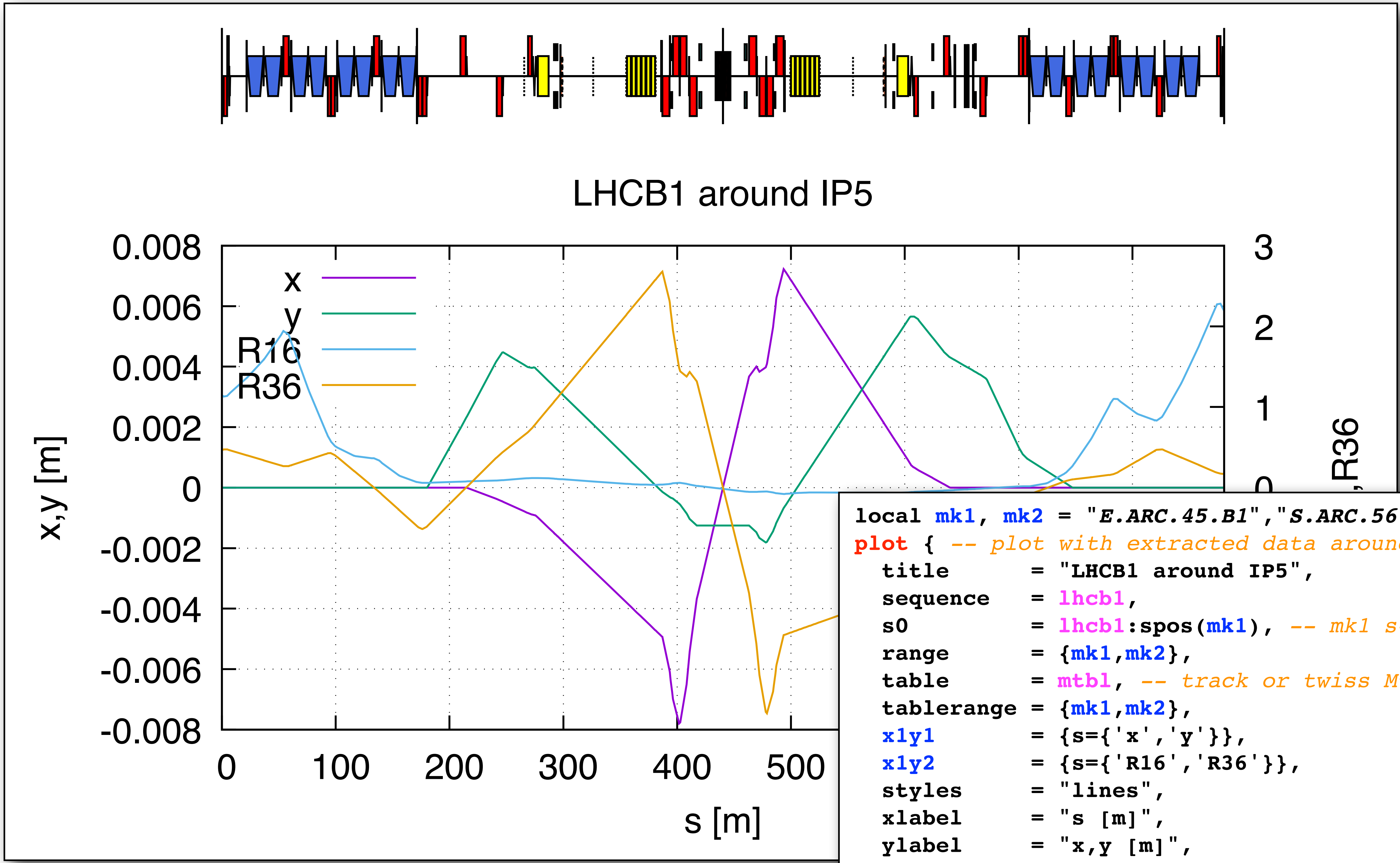
```
twiss = command 'twiss' {
  sequence=nil,
  beam=nil,
  range=nil,
  dir=nil,

  s0=nil,
  X0=nil,
  O0=nil,
  deltap=nil,
  chrom=false,
  coupling=false,
  trkrdt=false,

  nturn=nil,
  nstep=nil,
  nslice=nil,
  method=nil,
  model=nil,
  mapdef=2,
  secnmul=nil,
  ptcmodel=nil,
  implicit=nil,
  misalign=nil,
  aperture=nil,
  fringe=nil,
  frngmax=nil,
  radiate=nil,
  nocavity=nil,
  totalpath=nil,
  cmap=nil,

  save=nil,
  aper=nil,
  observe=0,
  savemap=nil,

  atentry=nil,
  atslice=nil,
  atexit=nil,
  atsave=nil,
  ataper=nil,
  atdebug=nil,
  savesel=nil,
  apersel=nil,

  …
}
```

*The attributes (not required) set to **nil** will use the default of the reference command (red arrows).*

Survey, Track & Twiss plot: *LHC B1 around IP5*

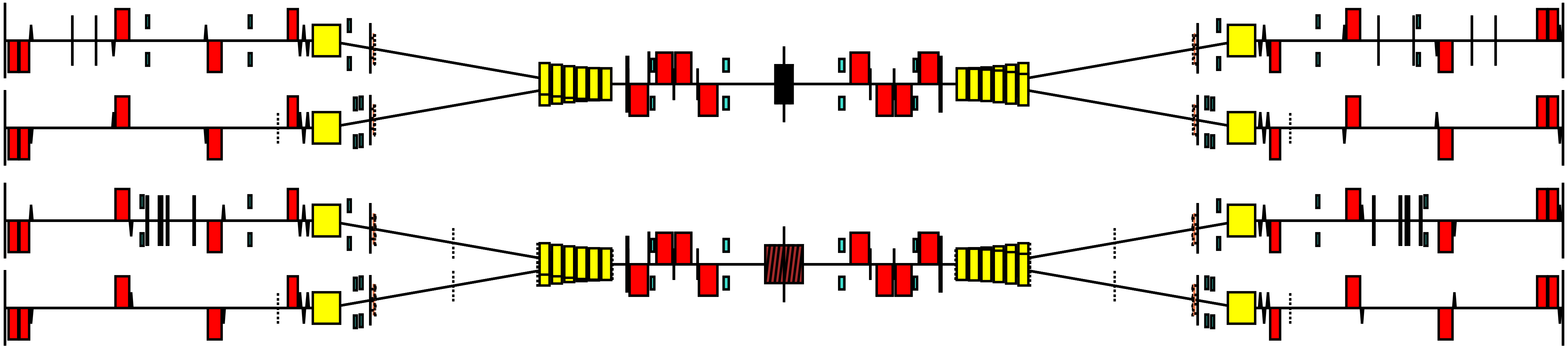


Survey, Track & Twiss plot: *LHC B1 around IP5*



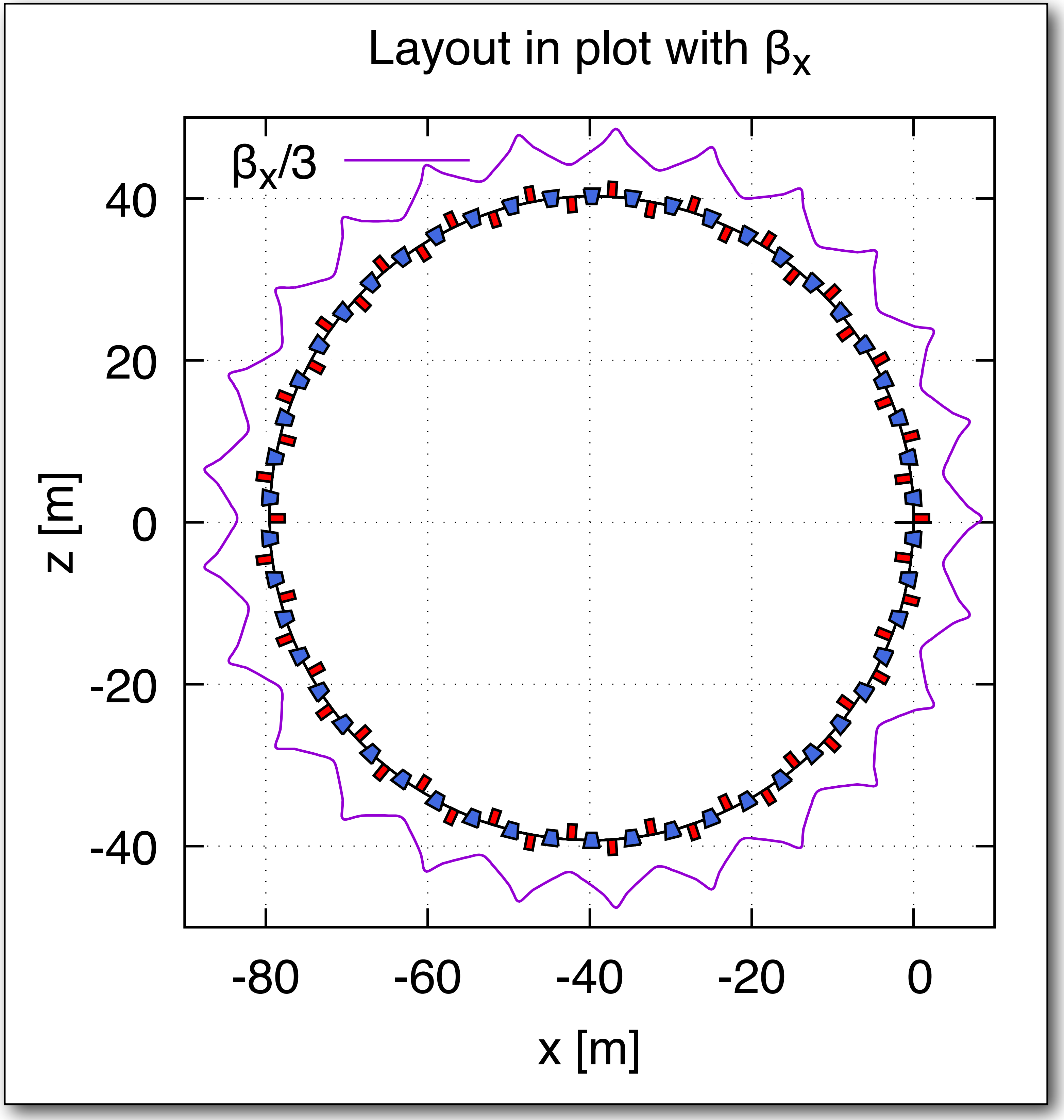
```
local mk1, mk2 = "E.ARC.45.B1","S.ARC.56.B1"
plot { -- plot with extracted data around IP5
  title       = "LHCB1 around IP5",
  sequence    = lhcb1,
  s0          = lhcb1:spos(mk1), -- mk1 s-position
  range       = {mk1,mk2},
  table       = mtbl, -- track or twiss MTable
  tablerange = {mk1,mk2},
  x1y1        = {s={'x','y'}},
  x1y2        = {s={'R16','R36'}},
  styles      = "lines",
  xlabel      = "s [m]",
  ylabel      = "x,y [m]",
  y2label     = "R16,R36",
  fontsize    = 14,
  output      = "plots/orbit_lhcb1_ip5_da.pdf",
--scrdump     = "plots/orbit_lhcb1_ip5.gp",
}
```

Layout plot: *LHC B1 & B2 at IP1 & IP5*



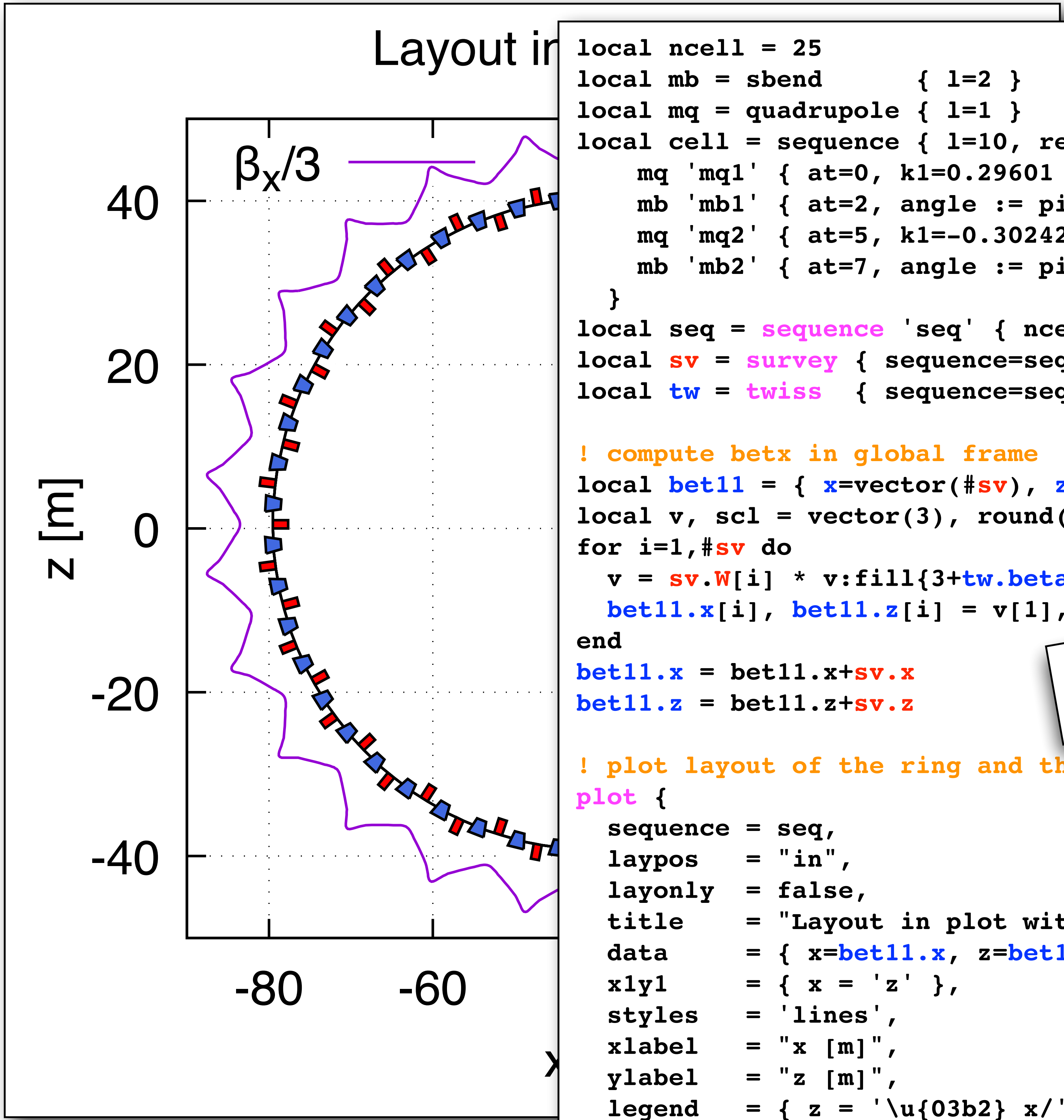
```
plot {
  sequence = { lhcb1, lhcb2, lhcb1, lhcb2 },
  range    = { ! Ranges for the 4 sequences above
    {“E.DS.L1.B1","S.DS.R1.B1"},{"E.DS.L1.B2","S.DS.R1.B2"},
    {"E.DS.L5.B1","S.DS.R5.B1"},{"E.DS.L5.B2","S.DS.R5.B2"},
  },
  laydisty = {
    lhcb2[“E.DS.L1.B2"].mech_sep,        ! Second bline y-shift [m]
    -0.4,                                ! Third  bline y-shift [m]
    -0.4 + lhcb2[‘E.DS.L5.B2'].mech_sep ! Fourth bline y-shift [m]
  },
}
```

Plot survey & twiss: β_x *around the ring*



Laurent Deniau, CERN BE/ABP, 1211 Geneva, laurent.deniau@cern.ch

Plot survey & twiss: β_x *around the ring*



```
local ncell = 25
local mb = sbend      { l=2 }
local mq = quadrupole { l=1 }
local cell = sequence { l=10, refer='entry',
    mq 'mq1' { at=0, k1=0.29601        },
    mb 'mb1' { at=2, angle := pi/ncell },
    mq 'mq2' { at=5, k1=-0.30242       },
    mb 'mb2' { at=7, angle := pi/ncell },
  }
local seq = sequence 'seq' { ncell*cell, beam=beam }
local sv = survey { sequence=seq, nslice=5, save="atstd", mapsave=true }
local tw = twiss  { sequence=seq, nslice=5, save="atstd" }

! compute betx in global frame
local bet11 = { x=vector(#sv), z=vector(#sv) }
local v, scl = vector(3), round(tw.beta11:max()/5)
for i=1,#sv do
  v = sv.W[i] * v:fill{3+tw.beta11[i]/scl, 0, 0}
  bet11.x[i], bet11.z[i] = v[1], v[3]
end
bet11.x = bet11.x+sv.x
bet11.z = bet11.z+sv.z

! plot layout of the ring and the betx
plot {
  sequence = seq,
  laypos   = "in",
  layonly  = false,
  title    = "Layout in plot with \u{03b2}_x",
  data     = { x=bet11.x, z=bet11.z },
  x1y1     = { x = 'z' },
  styles   = 'lines',
  xlabel   = "x [m]",
  ylabel   = "z [m]",
  legend   = { z = '\u{03b2}_x/'..scl },
}
```

Plots are gnuplot based, .gp script can be generated for further customisation.

PyMAD-NG: *Python integration (e.g. for XSuite)*

Python script

PyMAD-NG doc from Joshua Gray

```
from pymadng import MAD

-- create an instance of MAD-NG
madng = MAD()

-- send script to MAD-NG
madng.send('''
   MADX:load("lhc.seq"    , "lhc.mad"   ) -- load LHC B1 & B2 highly configurable through ~30000 deferred expr.
   MADX:load("optics.madx", "optics.mad") -- load optics, e.g. variables & strengths involved in deferred expr.
   MADX:load("knobs.madx" , "knobs.mad" ) -- load knobs,  e.g. crossing-angle setup for collision optics

    local damap, twiss, pymad in MAD
    local lhcb1 in MADX

    -- list of octupolar RDTs
    local rdts = {"f4000", "f3100", "f2020", "f1120"}

    -- create phase-space damap at 4th order
    local X0 = damap {nv=6, mo=4}

    -- twiss with RDTs along the ring
    local mtbl = twiss {sequence=lhcb1, X0=X0, trkrdt=rdts}

    -- send selected columns of MTable to Python
    pymad:send{mtbl.s, mtbl.beta11, mtbl.beta22, mtbl.f4000, mtbl.f3100, mtbl.f2020, mtbl.f1120}

    -- send entire MTable to Python
    pymad:send(mtbl)
''')

-- receive selected columns of MTable from MAD-NG as Numpy arrays (vectors)
s, beta11, beta22, f4000, f3100, f2020, f1120 = madng.recv()

-- receive entire MTable from MAD-NG and convert it to Pandas DataFrame
mtbl = madng.recv().to_df()
```

MAD-NG script

PART II
Physics

MAD-NG physics I – tracking engines

- **5D-6D PTC physics using differential algebra and symplectic integrators.**
  - **combined physics & elements, slicing & frames, easy to extend, etc…**
  - **x10-30 faster than MADX-PTC for TPSA tracking.**
- **Survey: geometrical tracking**
  - Survey supports **multi-turns**, **ranged** and step-by-step **forward, backward** and **reverse** geometrical tracking.
  - Support **exact** misalignments, **permanent** misalignments, and patches.
  - Output MTable (TFS) fully compatible with Track for combining observable points *(smooth plots, slicing, actions, sub-elements, combining **local & global frame**, etc…)*
- **Track: dynamical tracking**
  - Track supports **multi-particles** or **multi-damaps**, **multi-turns**, **ranged** and step-by-step **forward**, **backward** and **reverse** dynamical tracking of **charged** particles to **arbitrary differential order** with an arbitrary number of **parameters** (few hundreds).
  - Support **exact** misalignments, **permanent** misalignments, combined **multipoles** & field errors **for all elements,** and **patches** (frame changes).
  - **Symplectic integrators up to 8th (12th) order** on 5D and 6D phase space *(PTC-like exact=true, time=true, totalpath e.g. for thick RF)*.
  - Support both **thick** and **thin** lens models, **radiation** (including photons tracking), **fringe fields** for all elements, **mutable particles** (multiple beams), **exact patches** (translations, rotations & time-energy), weak-strong beam-beam, any aperture shape.
  - Output MTable (TFS) fully compatible with Survey for combining observable points.

MAD-NG physics II – meta commands

Cofind: fix point search

- Meta command that extends **Track** with actions and run a Newton-based optimiser iteratively calling **Track** with either (user-choice):
  - 1st order DA map (TPSAs)
  - 13 particles with 2nd order central finite differences (less stable)

  to obtain the Jacobian.

Twiss: normal form tracking

- Meta command that extends **Track** with actions to compute optics on-the-fly and fill the twiss MTable (extended track MTable) by running the following commands:
  - **Cofind** iterates over Track to find a closed orbit.
  - **Track** DA maps on the closed orbit to obtain high order one-turn map (OTM).
  - **Normal** to obtain the linear & non-linear normal forms from the OTM.
  - **Track** normalising forms and compute optical functions and RDTs.
- Computes coupled linear and non-linear optical functions, tunes, chromaticities, Generating Functions (RDTs), Hamiltonian Terms, synchrotron integrals, compaction factor, phase slip factor, gamma transition, Montaigue functions, etc…

Correct: orbit correction

- Provides few algorithms (e.g. SVD, Micado) and many options to correct the orbit using Beam Position Monitors (BPM) and Correctors (H-V Kickers).

MAD-NG physics III – optimisation and analysis

- **Match: highly configurable optimiser**
  - On the model of MAD-X `use_macro` approach, but with arbitrary user's setups & runs.
  - Provides all kinds of local & global, linear & non-linear, optimiser (~20 algorithms).
  - Very flexible, highly configurable with many **physics-oriented** contraints and objectives, i.e. not just a penalty-function to minimise.
- **Normal: parametric normal forms & analysis**
  - Provides linear and **non-linear parametric normal forms** on high order DA maps to compute RDTs and perform analysis. Can be triggered at observable points only to speed up the matching of local constraints.
    Non-linear normal forms and RDTs tracking are ***x50-80 faster than MADX-PTC*****,** fast enough to train **machine learning models** that we use extensively at CERN.

GTPSA in a nutshell

BE Beams Department

- Generalised Truncated Power Series Algebra

IPAC 2015

2017-2018

Github MAD

➡ Multivariate Taylor polynomials of order n in ℝ & ℂ.

➡ Powerful tool for solving differential equations (e.g. motion equations).

TPSA coefficients

1 variable x at order n in the *neighbourhood* of the point a in the domain of the function f :

$$T_f^n(x;a) = f(a) + f'(a)(x-a) + \frac{f''(a)}{2!}(x-a)^2 + \ldots + \frac{f^{(n)}(a)}{n!}(x-a)^n = \sum_{k=0}^{n} \boxed{\frac{f_a^{(k)}}{k!}}(x-a)^k$$

convergence of the remainder (i.e. truncation error):

$$\lim_{n\to\infty} R_f^n(x;a) = \lim_{n\to\infty} f(x) - T_f^n(x;a) = 0$$

f(x) is an analytic function, $T_f^n(x;a)$ *is a polynomial approximation nearby* a *with radius of convergence* h*:* $\min_{h>0} \lim_{n\to\infty} R_f^n(a \pm h; a) \neq 0.$

2 variables (x,y) at order 2 nearby (a,b):

$$= f^{(1)}_{(a,b)}(x-a, y-b)$$

$$T_f^2(x,y;a,b) = f(a,b) + \boxed{\left.\frac{\partial f}{\partial x}\right|_{(a,b)}(x-a) + \left.\frac{\partial f}{\partial y}\right|_{(a,b)}(y-b)} + \ldots$$

homogeneous polynomials

f must not depend on the integration path

$$+\frac{1}{2!}\left(\boxed{\left.\frac{\partial^2 f}{\partial x^2}\right|_{(a,b)}(x-a)^2 + \left.\boxed{2\frac{\partial^2 f}{\partial x \partial y}}\right|_{(a,b)}(x-a)(y-b) + \left.\frac{\partial^2 f}{\partial y^2}\right|_{(a,b)}(y-b)^2}\right)$$

$$= f^{(2)}_{(a,b)}(x-a, y-b)$$

v variables X at order n nearby A:

TPSA coefficients

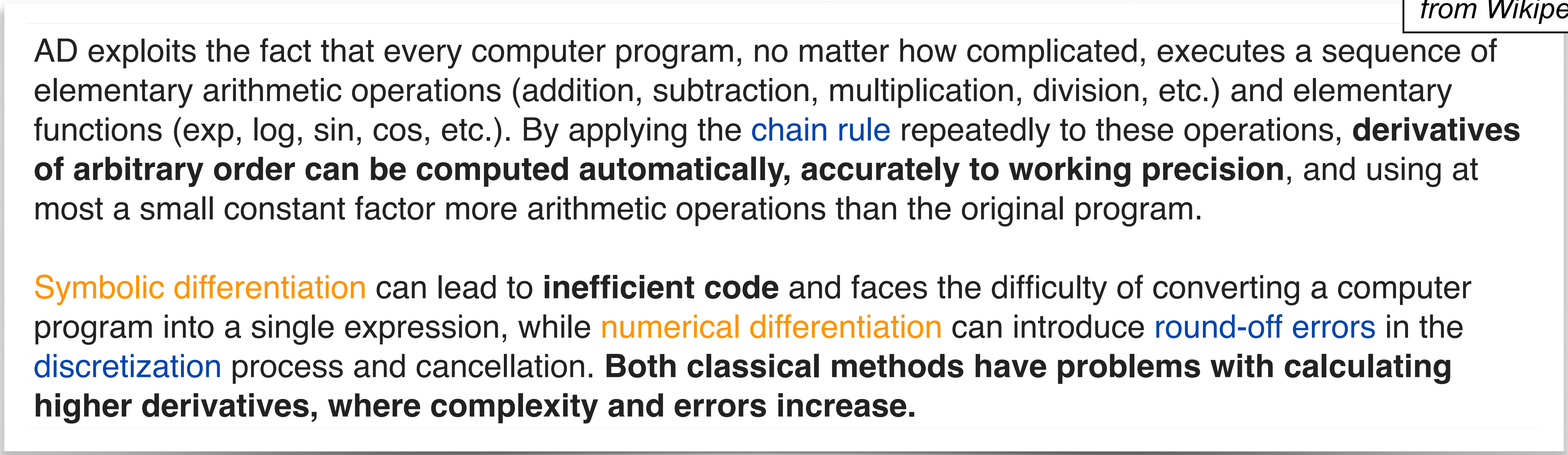
$$T_f^n(X;A) = \sum_{k=0}^{n} \frac{f_A^{(k)}}{k!}(X;A)^k = \sum_{k=0}^{n} \boxed{\frac{1}{k!}} \sum_{\boxed{|\vec{m}|=k}} \left.\boxed{\binom{k}{\vec{m}} \frac{\partial^k f}{\partial X^{\vec{m}}}}\right|_A (X;A)^{\vec{m}} \quad \text{with} \quad \boxed{\binom{k}{\vec{m}}} = \frac{k!}{c_1!\,c_2!\ldots c_v!}$$

monomials of order k

multinomial

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ Differential algebra (DA) is computed using **automatic differentiation** (AD).



from Wikipedia

AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.

Symbolic differentiation can lead to **inefficient code** and faces the difficulty of converting a computer program into a single expression, while numerical differentiation can introduce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors? (Something that we never do…)*
  - ➡ If they are used as *functions* (e.g. evaluated), instead of *DA* (e.g. track, twiss).
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution, e.g. MADX.

$$T_f^n(x;a+h) = \sum_{k=0}^{n} \frac{f_{a+h}^{(k)}}{k!}(x-a-h)^k\,;\quad f(a+h) \approx \underbrace{\sum_{k=0}^{n}\frac{f_a^{(k)}}{k!}h^k}_{T_f^n(a+h;a)}\,;\quad f_{a+h}^{(k)} \approx \frac{\mathrm{d}^k T_f^n(x;a)}{\mathrm{d}x^k}(a+h)$$

Matrix codes don't do better!

order n is constant
order n-1 is linear in h

Accuracy of TPSA (myths and legends)

- GTPSA are **exact** to machine precision, **no** approximation for orders 0..n
  - ➡ Differential algebra (DA) is computed using **automatic differentiation** (AD).

from Wikipedia

> AD exploits the fact that every computer program, no matter how complicated, executes a sequence of elementary arithmetic operations (addition, subtraction, multiplication, division, etc.) and elementary functions (exp, log, sin, cos, etc.). By applying the chain rule repeatedly to these operations, **derivatives of arbitrary order can be computed automatically, accurately to working precision**, and using at most a small constant factor more arithmetic operations than the original program.
>
> Symbolic differentiation can lead to **inefficient code** and faces the di g a computer program into a single expression, while numerical differentiation can i uce round-off errors in the discretization process and cancellation. **Both classical methods have problems with calculating higher derivatives, where complexity and errors increase.**

Functions of TPSAs ≠ TPSAs as functions
exact ≠ approximate

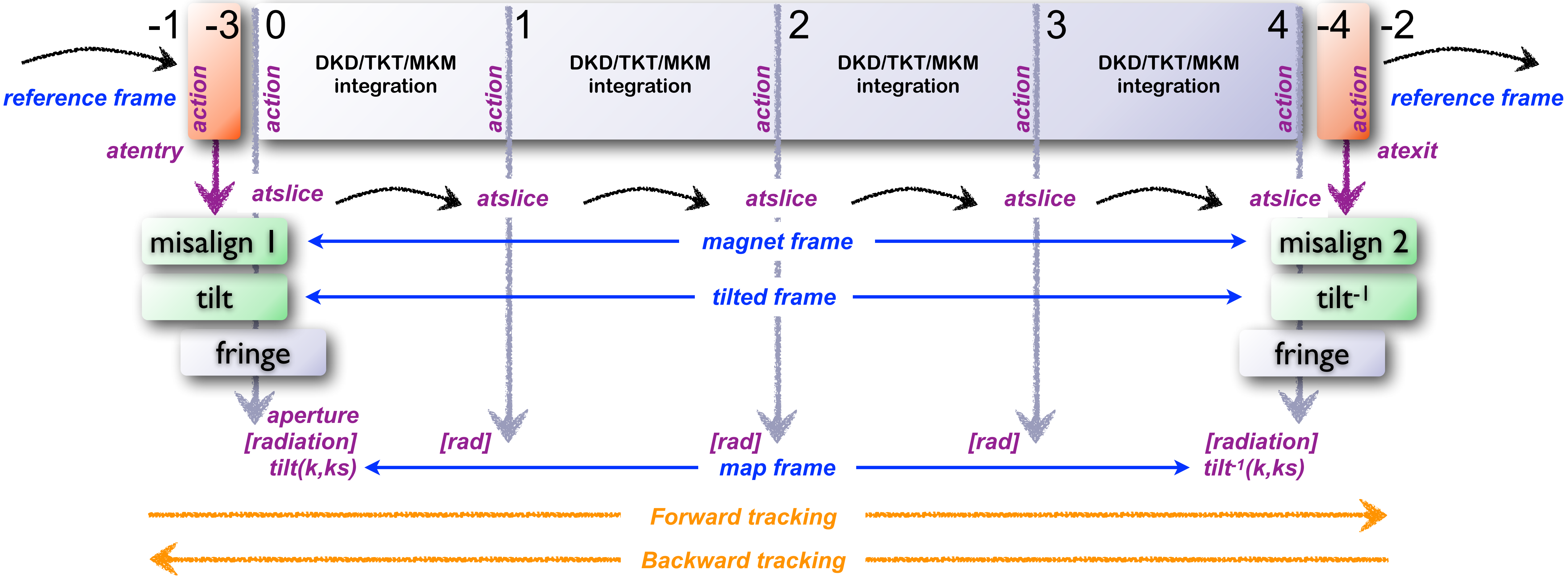
- MAD-NG includes a complete toolbox (i.e. module) to handle DA using AD…
  - ➡ users have full access to GTPSA and DAmaps from the scripting language.
  - ➡ users can manipulate DAmaps stored in the MTable or the MFlow returned by Track.
- *So when DAmap/TPSA introduce errors? (Something that we never do…)*
  - ➡ If they are used as *functions* (e.g. evaluated), instead of *DA* (e.g. track, twiss).
  - ➡ High orders of $T_f^n(x;a)$ are used to interpolate at the new position by substitution, e.g. MADX.

$$T_f^n(x;a+h) = \sum_{k=0}^{n} \frac{f_{a+h}^{(k)}}{k!}(x-a-h)^k\,;\qquad f(a+h) \approx \underbrace{\sum_{k=0}^{n} \frac{f_a^{(k)}}{k!}h^k}_{T_f^n(a+h;a)}\,;\qquad f_{a+h}^{(k)} \approx \frac{\mathrm{d}^k T_f^n(x;a)}{\mathrm{d}x^k}(a+h)$$

Matrix codes don't do better!

order n is constant
order n-1 is linear in h

Element tracking: *physics selection*


-1 -3 0 1 2 3 4 -4 -2

reference frame

action

DKD/TKT/MKM integration

atentry

atexit

atslice

misalign 1

misalign 2

magnet frame

tilt

tilted frame

tilt-1

fringe

aperture

[radiation]

tilt(k,ks)

[rad]

map frame

tilt-1(k,ks)

Forward tracking

Backward tracking

- **When entering** an element (before slice -1), the Track engine delegates to the element's the responsibility to select its own physics amongst a catalogue of **physics maps** and **symplectic integrators**.
- This element-dependent selection is based on the *tracking context*, the *element attributes* retrieved and **their values when entering**, *i.e. can vary during tracking*.

Simplified element Tracking engine

```
atentry(elm, mflw,  sdir)
misalgn(elm, mflw,  sdir)
tilt   (ang, mflw,  sdir)
fringe (elm, mflw,  sdir)   DKD/TKT/MKM
integr (elm, mflw,     1, thick, kick)
fringe (elm, mflw, -sdir)      atslice
tilt   (ang, mflw, -sdir)
misalgn(elm, mflw, -sdir)
atexit (elm, mflw, -sdir)
```

- It selects the **track**ing engine (the box), the **DKD**, **TKT** or **MKM model**, the **integr**ator scheme and its **order**, the "**thick**" and "**kick**" maps to be integrated, the "**fringe**" map, and runs the element **track**ing engine with this setup.

Elements physics: *from Lua to C++*

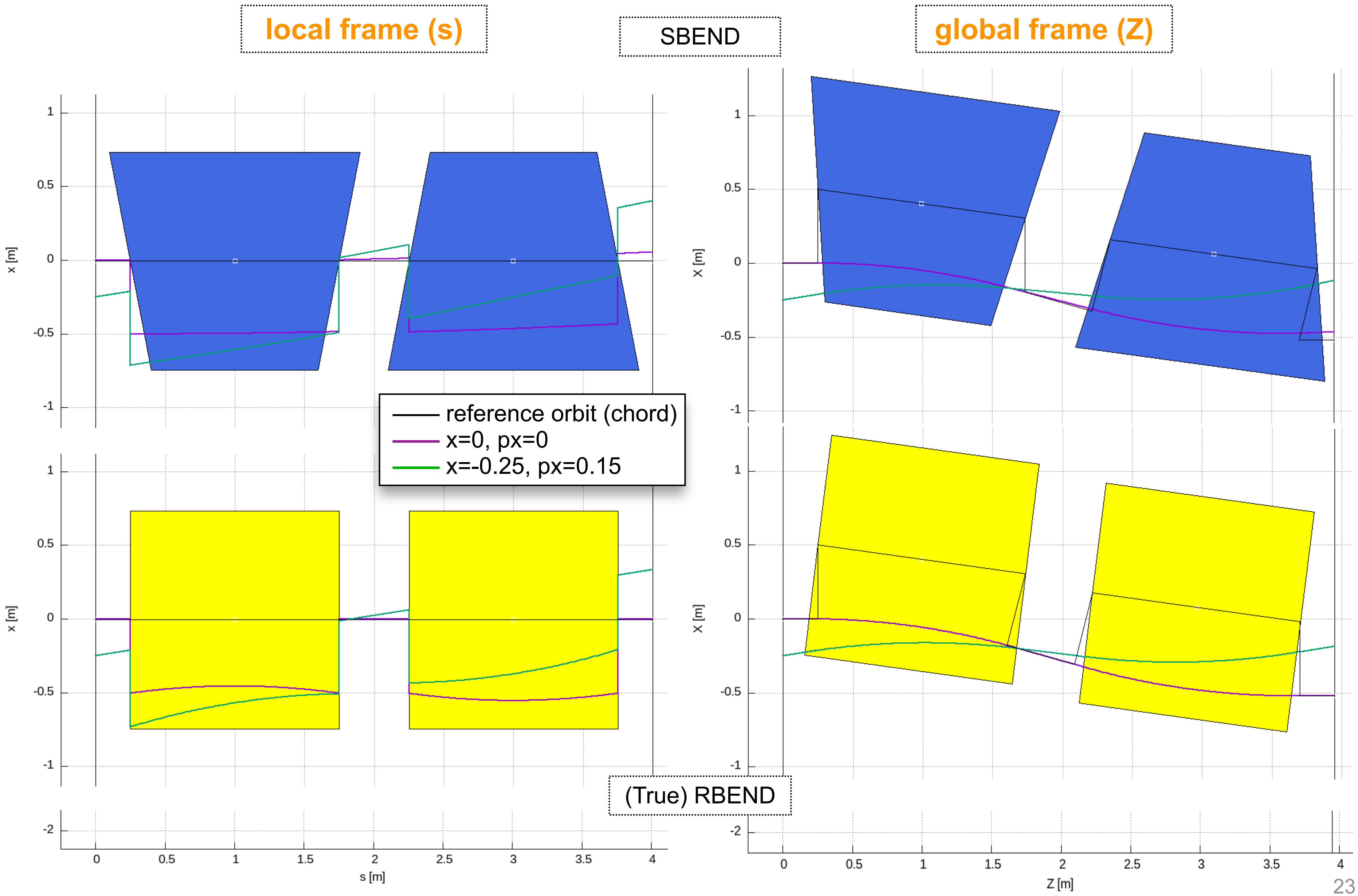
```
local function curex_drift (elm, m, lw, istp)

  local ld = (m.eld or m.el)*lw
  local ang, rho = m.eh*m.el*lw*m.edir, 1/m.eh*m.edir
  local ca, sa, sa2 = cos(ang), sin(ang), sin(ang/2)
  local beta = m.beam.beta – cache value of beta

  for i=1,m.npar do
    local x, px, y, py, t, pt in m[i]

    local dpp1 = 1 + 2/beta*pt + pt^2
    local   pz = sqrt(dpp1 - px^2 - py^2)
    local  _pz = 1/pz
    local  pxt = px*_pz
    local _ptt = 1/(ca - sa*pxt)
    local  pst = (x+rho)*sa*_pz*_ptt

    m[i].x  = (x + rho*(2*sa2^2 + sa*pxt))*_ptt
    m[i].px = ca*px + sa*pz
    m[i].y  = y + pst*py
    m[i].t  = t - pst*(1/beta+pt) + (1-m.T)/beta*ld
  end

end
```

```
template <typename M,         – type of map flow
          typename T=M::T,  – type of variable
          typename P=M::P,  – type of parameter
          typename R=M::R>  – type of parameter reference
inline void curex_drift (cflw<M> &m, num_t lw, int istp)
{
  P ld  = (fval(m.eld) ? R(m.eld) : R(m.el))*lw;
  P ang = R(m.eh)*R(m.el)*lw*m.edir, rho = 1/R(m.eh)*m.edir;
  P ca  = cos(ang), sa = sin(ang), sa2 = sin(ang/2);


  FOR(i,m.npar) {
    M p(m,i);

    T dpp1 = 1 + 2/m.beta*p.pt + sqr(p.pt);
    T   pz = sqrt(dpp1 - sqr(p.px) - sqr(p.py));
    T  _pz = 1/pz;
    T  pxt = p.px*_pz;
    T _ptt = 1/(ca - sa*pxt);
    T  pst = (p.x+rho)*sa*_pz*_ptt;

    p.x  = (p.x + rho*(2*sqr(sa2) + sa*pxt))*_ptt;
    p.px = ca*p.px + sa*pz;
    p.y += pst*p.py;
    p.t -= pst*(1/m.beta+p.pt) - (1-m.T)/m.beta*ld;
  }

}
```

Map for the exact drift in a curved frame,
e.g. selected by the sector bend for DKD.

- **C++ vs Lua code**
  - ➡ Both are simple to read: extensive use of **operator overloading**.
  - ➡ Both are fully polymorphic: same code for particles, DA maps, and ***parametric DA maps***.
  - ➡ Performances: GTPSA C++ classes use **better memory management** than Lua's garbage collector, resulting in a ***speed improvement of x7***.
  - ➡ cmap=true/false: both codes can be used for crosscheck or **rapid development**.

Local & Global Frames: *combining misalignments*



local frame (s)

SBEND

global frame (Z)

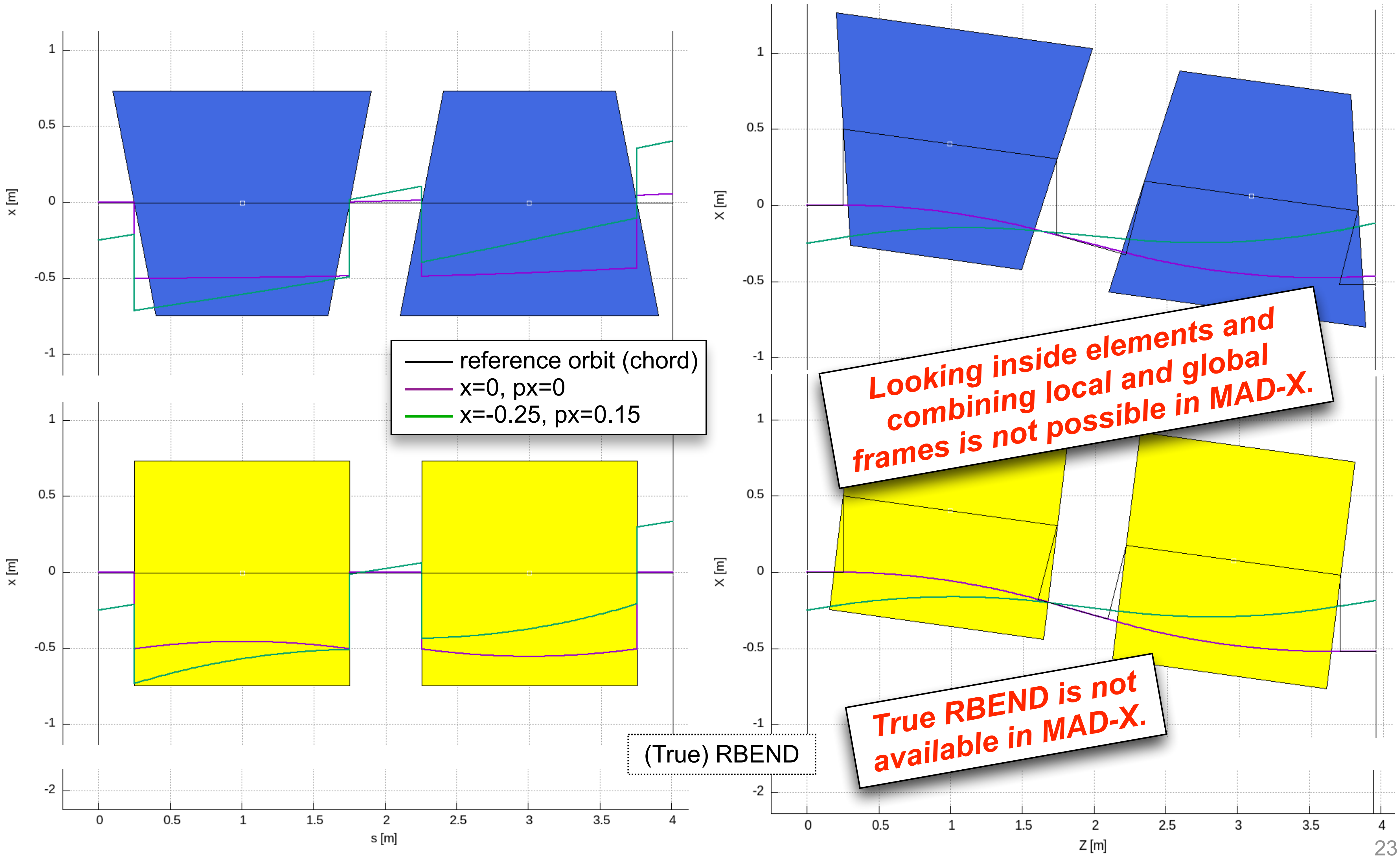
(True) RBEND

Local & Global Frames: *combining misalignments*

local frame (s)

SBEND

global frame (Z)



(True) RBEND

Looking inside elements and combining local and global frames is not possible in MAD-X.

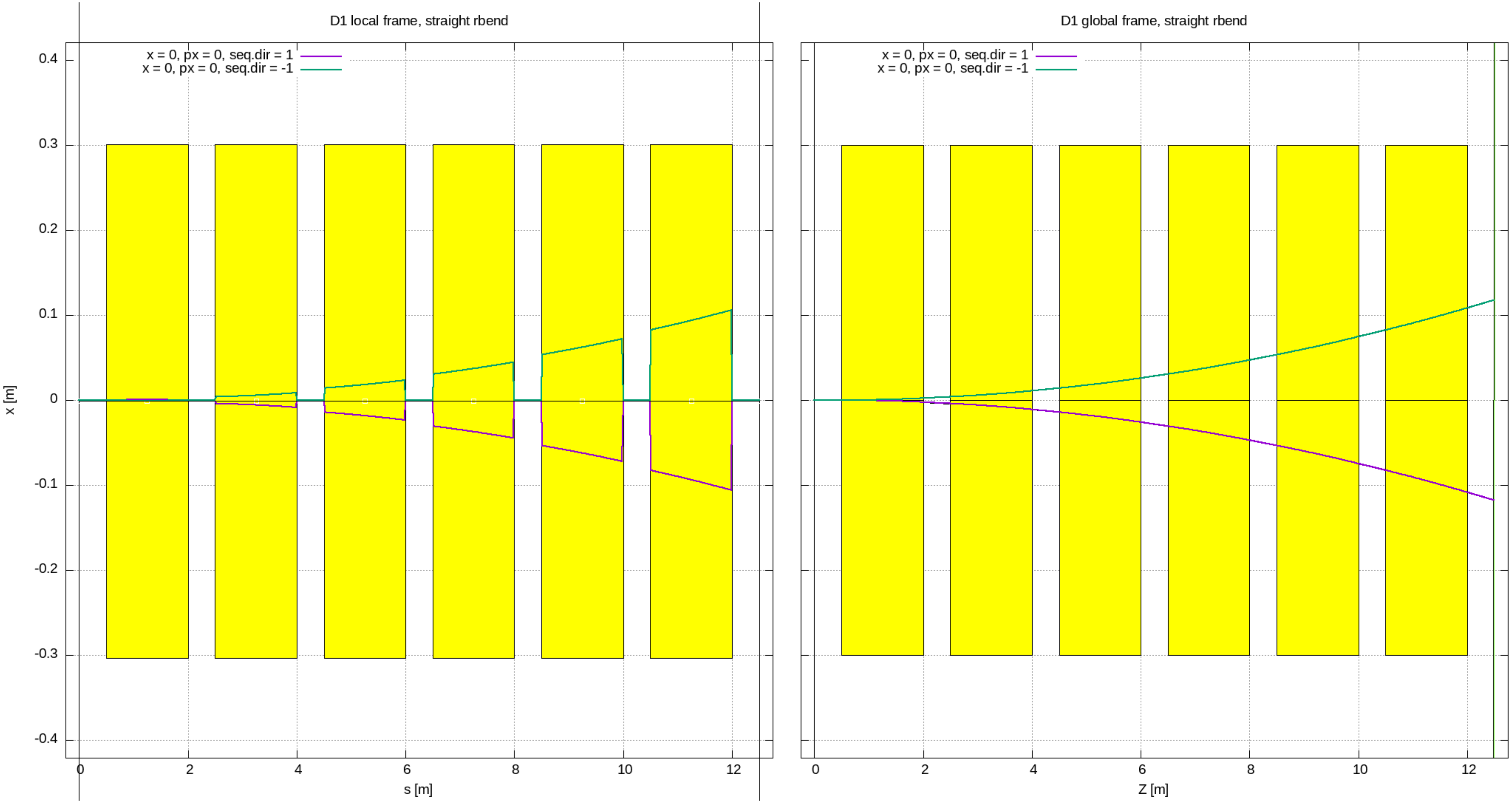
True RBEND is not available in MAD-X.

LHC D1 with True RBends

local frame (s)

RBEND (L_C) (PTC true_rbend)

global frame (Z)



- reference orbit
- seq1.dir=1, x=0, px=0
- seq2.dir=-1, x=0, px=0

Laurent Deniau, CERN BE/ABP, 1211 Geneva 23, laurent.deniau@cern.ch

PART III
Applications

MAD-NG studies: LHC coupling with param. maps

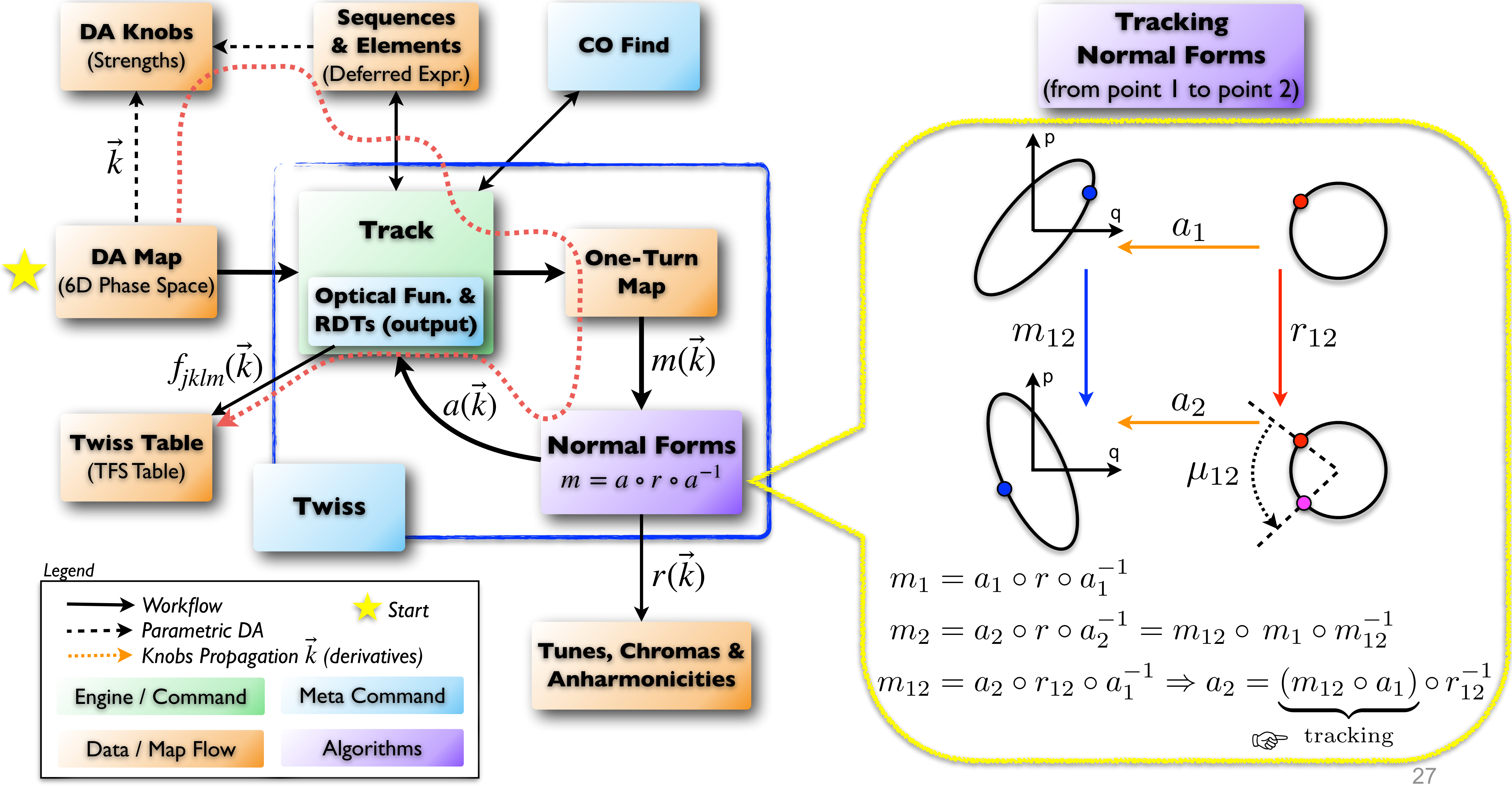
```
local X0 = damap {nv=6, mo=2, np=4, po=1,
                  pn={‘sk1r','sk2r','sk3r','sk4r'}}

local mjac = { ---> variables & knobs
  { var='x'  ,'0010001','00100001','001000001','0010000001' }, --     |
  { var='x'  ,'0001001','00010001','000100001','0001000001' }, --     |
  { var='px','0010001','00100001','001000001','0010000001' }, --     v
  { var='px','0001001','00010001','000100001','0001000001' }, -- constraints
}
```

```
-- set knobs: scalar + TPSA -> TPSA
MADX.sk1r = MADX.sk1r + X0.sk1r
MADX.sk2r = MADX.sk2r + X0.sk2r
MADX.sk3r = MADX.sk3r + X0.sk3r
MADX.sk4r = MADX.sk4r + X0.sk4r

match {
  command := track {sequence=lhcb1, X0=X0, savemap=true},

  jacobian = \t,_,jac => -- gradient not used, fill only Jacobian
    jac:setrow(1.. 8, t['S.DS.L2.B1'].__map:getm(mjac) )
    jac:setrow(9..16, t['E.DS.L2.B1'].__map:getm(mjac) )
  end,

  variables = { rtol=1e-6, -- 1 ppm
    { name='sk1r', var=‘MADX.sk1r’ },
    { name='sk2r', var=‘MADX.sk2r’ },
    { name='sk3r', var=‘MADX.sk3r’ },
    { name='sk4r', var=‘MADX.sk4r’ },
  },

  equalities = {
    { name=’S.R11.x', expr = \t -> t['S.DS.L2.B1'].__map.x :get’0010' },
    { name=’S.R12.x', expr = \t -> t['S.DS.L2.B1'].__map.x :get’0001' },
    { name=’S.R21.x’, expr = \t -> t[’S.DS.L2.B1'].__map.px:get'0010' },
    { name=’S.R22.x', expr = \t -> t[’S.DS.L2.B1'].__map.px:get'0001' },

    { name=’E.R11.x', expr = \t -> t['E.DS.L2.B1'].__map.x :get’0010' },
    { name=’E.R12.x', expr = \t -> t['E.DS.L2.B1'].__map.x :get’0001' },
    { name=’E.R21.x’, expr = \t -> t[‘E.DS.L2.B1'].__map.px:get'0010' },
    { name=’E.R22.x', expr = \t -> t[‘E.DS.L2.B1'].__map.px:get'0001' },
  },

  objective = { fmin=1e-12 },
  maxcall=100, info=2
}

-- unset knobs: restore scalar values from TPSA
MADX.sk1r = MADX.sk1r:get0()
MADX.sk2r = MADX.sk2r:get0()
MADX.sk3r = MADX.sk3r:get0()
MADX.sk4r = MADX.sk4r:get0()
```

Timing summary and links to codes:
MAD-X using R matrix **1 min 55s**
MAD-NG using R matrix **21s**
MAD-NG using R matrix & knobs **10s**
MADX-PTC using alphas-betas **~ 50 min**

MAD-NG's Parametric Differential Maps Principle

Track a **high-order differential algebra** (DA) map on the closed orbit (optionally) equipped with **parameters** (knobs) to obtain the **one-turn map** m, then compute the **closed non-linear normal form** $m = a \circ r \circ a^{-1}$ and track the normalising map a to extract the **optical functions** (α, β, μ, etc.) and the **resonant driving terms (RDTs)** along the lattice.



$$m_1 = a_1 \circ r \circ a_1^{-1}$$

$$m_2 = a_2 \circ r \circ a_2^{-1} = m_{12} \circ m_1 \circ m_{12}^{-1}$$

$$m_{12} = a_2 \circ r_{12} \circ a_1^{-1} \Rightarrow a_2 = \underbrace{(m_{12} \circ a_1)}_{\text{☞ tracking}} \circ r_{12}^{-1}$$

MAD-NG studies: RDTs for HL-LHC

```
-- HL-LHC setup
MADX:load("hllhc_saved.seq", "hllhc_saved.mad")
MADX.lhcb1.beam = beam {particle="proton", energy=450}
MADX.lhcb2.beam = beam {particle="proton", energy=450}
MADX.lhcb2.dir  = -1 -- bv = -1


-- list of RDTs
local rdts = {"f4000", "f3100", "f2020", "f1120"}

-- loop over lhcb1 and lhcb2
for _,lhc in ipairs{MADX.lhcb1, MADX.lhcb2} do

  -- create phase-space damap at 4th order
  local X0 = damap {nv=6, mo=4}

  -- compute RDTs along HL-LHC
  local mtbl = twiss {sequence=lhc, X0=X0, trkrdt=rdts}

  -- plot RDTs along HL-LHC
  plot_rdt(mtbl, rdts)

end -- end of loop
```

4th-order DA map

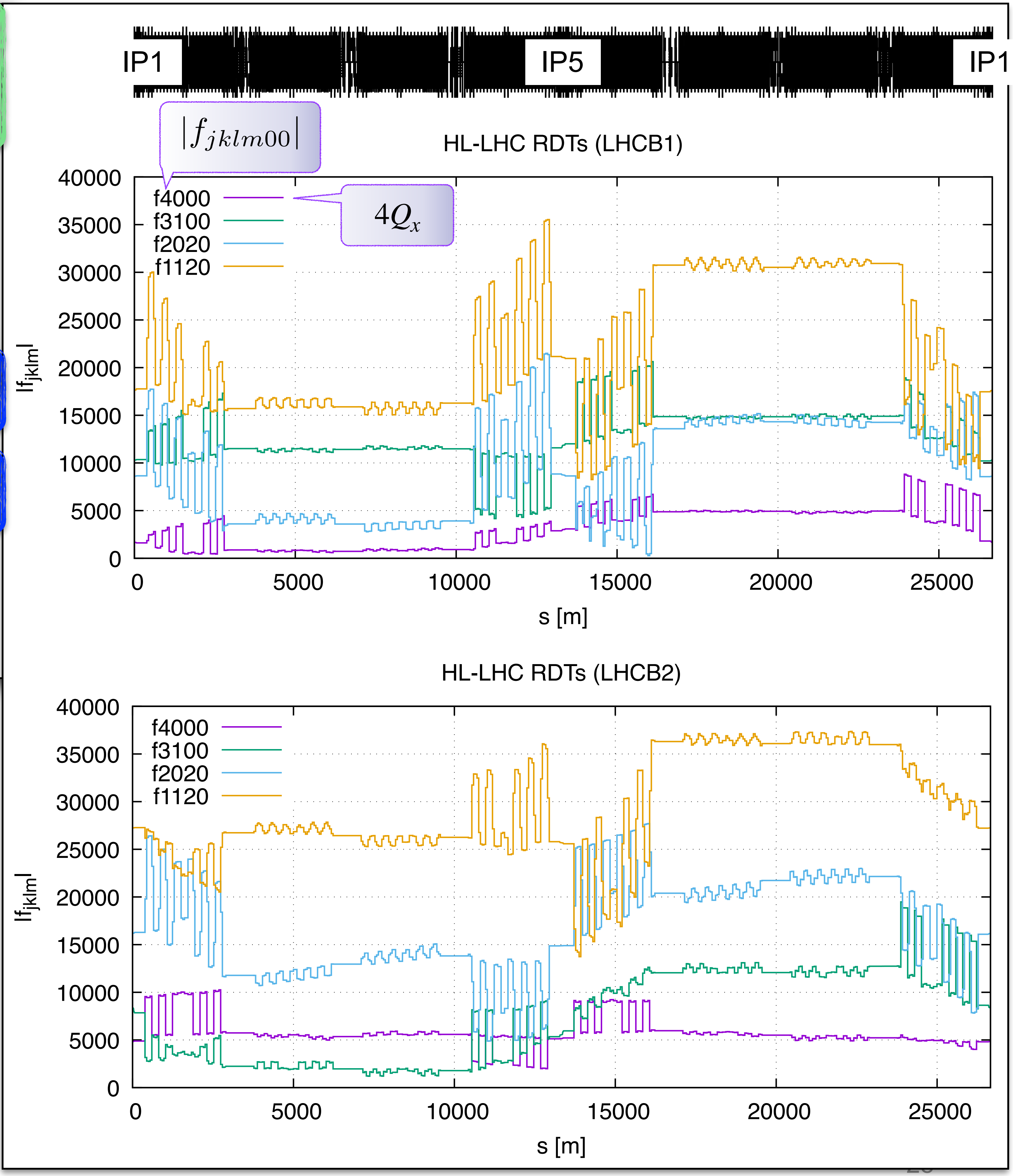
$$f_{jklm} = \frac{h_{jklm}}{1 - e^{2\pi i[(j-k)\nu_x + (l-m)\nu_y]}}$$

$$\text{Resonances: } N = (j-k)Q_x + (l-m)Q_y$$

$$\text{Spectral lines: } H(1+k-j, m-l)$$

$$V(k-j, 1+m-l)$$

These 2 RDTs' plots take
32 sec in MAD-NG
40 min in MADX-PTC


HL-LHC RDTs (LHCB1)

HL-LHC RDTs (LHCB2)

MAD-NG studies: Parametric RDTs for HL-LHC

```
-- HL-LHC setup
MADX:load("hllhc_saved.seq", "hllhc_saved.mad")
MADX.lhcb1.beam = beam {particle="proton", energy=450}
MADX.lhcb2.beam = beam {particle="proton", energy=450}
MADX.lhcb2.dir  = -1 -- bv = -1

-- list of knobs for both sequences
local knbs = { LHCB1 = {'ksf1.a45b1','ksf2.a45b1'},
               LHCB2 = {'ksf1.a45b2','ksf2.a45b2'} }

-- list of RDTs: 4Qx w 1st and 2nd derivatives vs knobs
local rdts = {“f40000000", “f40000010", "f40000020",
              “f40000011”, “f40000001", “f40000002",}

-- loop over lhcb1 and lhcb2
for _,lhc in ipairs{MADX.lhcb1, MADX.lhcb

  -- select knobs
  local knb = knbs[lhc.name]

  -- create phase-space damap at 6th order (mo=4th+po)
  local X0 = damap {nv=6, mo=6, np=#knb, po=2, pn=knb}

  -- set knobs: scalar + TPSA -> TPSA
  for _,k in ipairs(knb) do MADX[k] = MADX[k]+X0[k] end

  -- compute RDTs along HL-LHC
  local mtbl = twiss {sequence=lhc, X0=X0, trkrdt=rdts }

  -- plot RDTs along HL-LHC
  plot_rdt(mtbl, rdts)

end -- end of loop
```

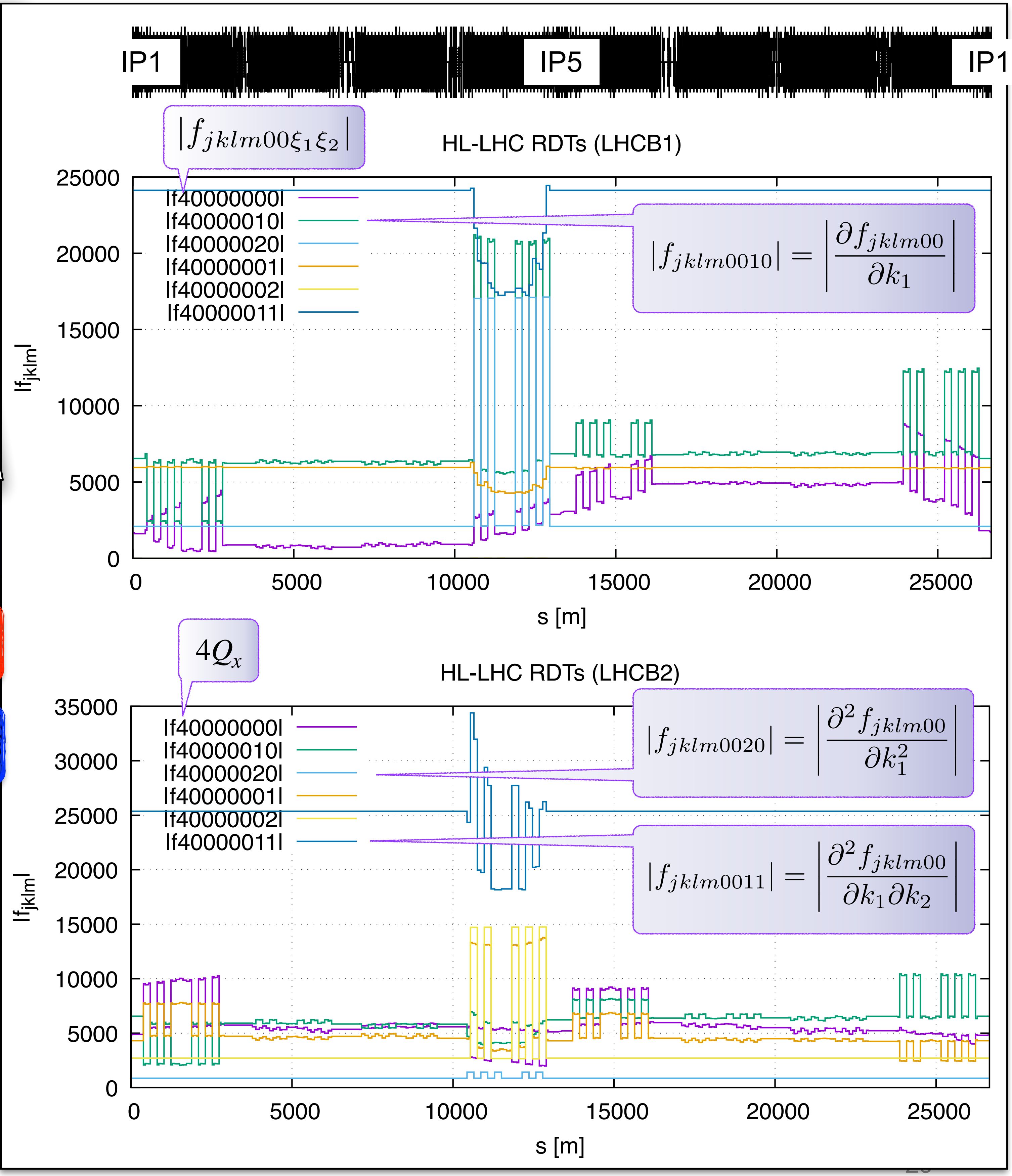
k_1

k_2

6th-order parametric DA map

$$f_{jklmno\xi_1\xi_2} = \frac{\partial^{\xi_1+\xi_2} f_{jklmno}}{\partial k_1^{\xi_1} \partial k_2^{\xi_2}}$$

$$\text{DA map order} = j+k+l+m+n+o+\sum_i \xi_i$$



MAD-NG studies: lower RDTs for LHC@injection

Loading LHC Sequences & Optics (1)

```
MADX:load'lhc_seq.madx'
MADX:load'inj_optics.madx'
MADX.lhcb1.beam = beam {particle='proton', energy=450}
MADX.lhcb2.beam = beam {particle='proton', energy=450}
MADX.lhcb2.dir  = -1 -- set LHCB2 as reversed
```

Building Parametric DA Map (2)

```
local prms = { -- param./knob names (strings)
  -- 16 strengths of trim quadrupoles families
  'kqtf.a12b1', 'kqtf.a23b1', ···, 'kqtf.a81b1',
  'kqtd.a12b1', 'kqtd.a23b1', ···, 'kqtd.a81b1',
  -- 16 strengths of octupoles families
  'kof.a12b1' , 'kof.a23b1' , ···, 'kof.a81b1',
  'kod.a12b1' , 'kod.a23b1' , ···, 'kod.a81b1',
}
```

32 Circuit Knobs 16 MQT + 16 MO

```
-- DA map representing parametric phase-space
local X0 = damap {nv=6, mo=5, np=#prms, po=1, pn=prms}
```

5th-order parametric DA map

```
-- convert scalars to GTPSAs within MADX env.
for _,knb in pairs(prms) do
  MADX[knb] = MADX[knb] + X0[knb]
end
```

Use strengths as DA map parameters

Parametric Normal Forms & Setup (3)

```
-- function to compute non-linear normal forms
local function get_nf (lhc, X0)
  local _, mflw = track {sequence=lhc, X0=X0}
  return normal(mflw[1]):analyse("all")
end

-- save reference values
local nf = get_nf(X0, MADX.lhcb1)
local q1ref   = nf:q1{1}
local q2ref   = nf:q2{1}
local q1jref  = nf:anhx{1,0}
local q2jref  = nf:anhy{0,1}
```

Twiss-like RDTs @ IP1 (faster for single point)

A solution is found by: MAD-NG in 3 min MADX-PTC in 45 min (using finite difference approx.)

Optimizing RDTs (4 & 5)

```
match {
  -- compute non-linear normal forms
  command := get_nf(), -- returns nf used below

  -- compute Jacobian from parametric maps
  jacobian = \nf,_,J =>
    for k=1,32 do -- fill [10x32] J matrix
      J:set(1,k, nf q1{1,k} or 0)
      J:set(2,k, nf q2{1,k} or 0)
      J:set(3,k, nf anhx{1,0,0,k})
      J:set(4,k, nf anhy{0,1,0,k})
      J:set(5,k, nf gnfu{"2002",k}.re)
      J:set(6,k, nf gnfu{"2002",k}.im)
      J:set(7,k, nf gnfu{"4000",k}.re)
      J:set(8,k, nf gnfu{"4000",k}.im)
      J:set(9,k, nf gnfu{"0040",k}.re)
      J:set(10,k,nf gnfu{"0040",k}.im)
    end
  end,

  -- variables in MADX env. to use as knobs
  variables = {
    {name=prms[1] , var='MADX[prms[1]]' },
    ···,
    {name=prms[32], var='MADX[prms[32]]'},
  },

  -- target constraints as equalities to zero
  equalities = {
    {name = 'q1'    , expr = \nf -> nf:q1{1} - q1ref},
    {name = 'q2'    , expr = \nf -> nf:q2{1} - q2ref},
    {name = 'q1j1'  , expr = \nf -> nf:anhx{1,0} - q1jre
    {name = 'q2j2'  , expr = \nf -> nf:anhy{0,1} - q2jre
    {name = 'f2002r', expr = \nf -> nf:gnfu{"2002"}.re  - 0},
    {name = 'f2002i', expr = \nf -> nf:gnfu{"2002"}.im) - 0},
    {name = 'f4000r', expr = \nf -> nf:gnfu{"4000"}.re) - 0},
    {name = 'f4000i', expr = \nf -> nf:gnfu{"4000"}.im) - 0},
    {name = 'f0040r', expr = \nf -> nf:gnfu{"0040"}.re) - 0},
    {name = 'f0040i', expr = \nf -> nf:gnfu{"0040"}.im) - 0},
  },
} -- close match
```

Jacobian [10x32] filled derivatives vs. knobs used by optimiser

$$\frac{\partial Q_x}{\partial k_i}, \frac{\partial Q_y}{\partial k_i}, \frac{\partial^2 Q_x}{\partial J_x \partial k_i}, \frac{\partial^2 Q_y}{\partial J_y \partial k_i}$$

$$\frac{\partial f_{2002}}{\partial k_i}, \frac{\partial f_{4000}}{\partial k_i}, \frac{\partial f_{0040}}{\partial k_i}$$

32 knobs to vary

10 constraints to satisfy

Invariant

$$Q_x, Q_y, \frac{\partial Q_x}{\partial J_x}, \frac{\partial Q_y}{\partial J_y}$$

Minimise

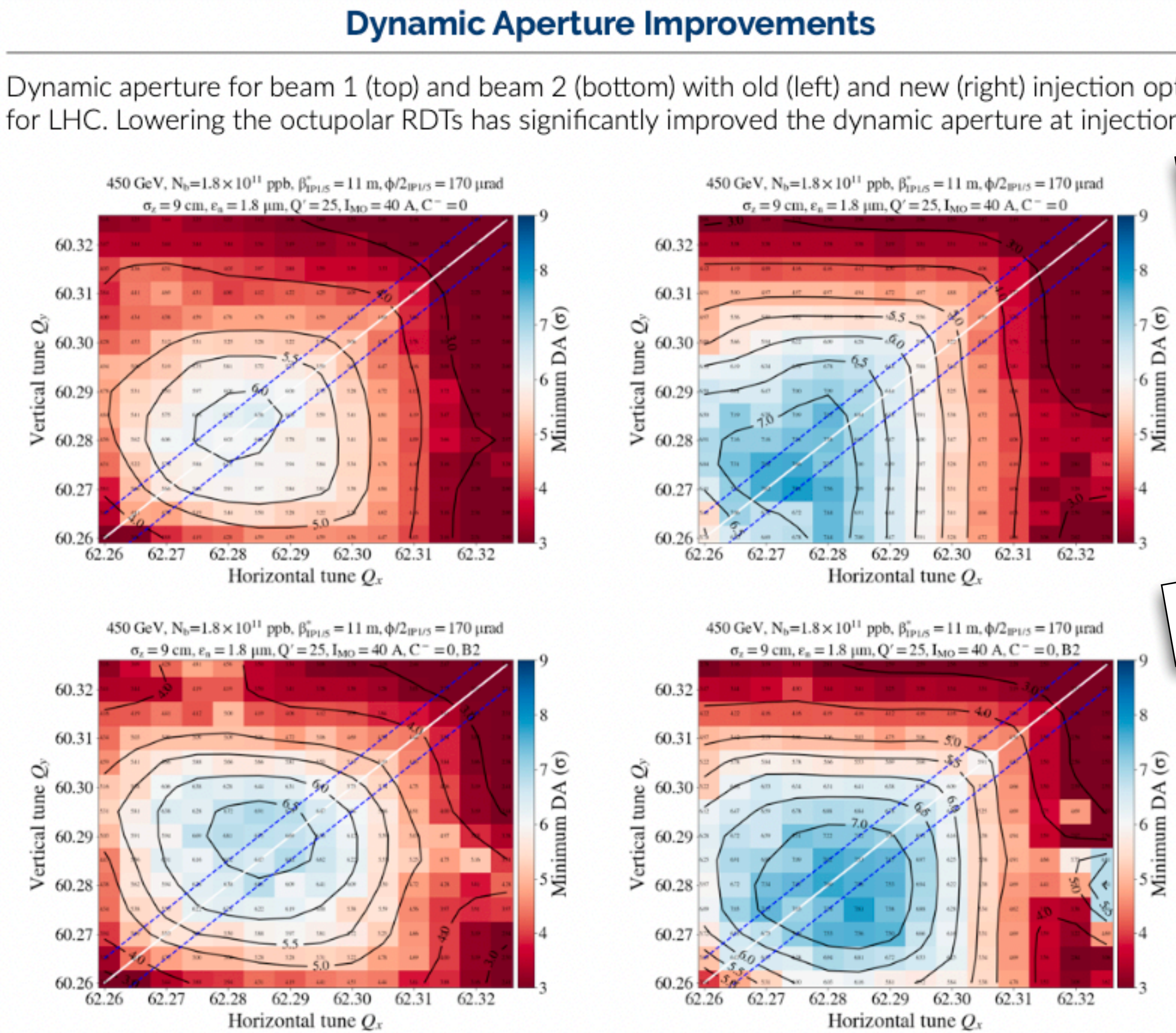
f_{2002}

f_{4000}

f_{0040}

Match has the same structure as in MAD-X with “use_macro”

MAD-NG studies: DA increase for LHC@injection



Dynamic Aperture Improvements

Dynamic aperture for beam 1 (top) and beam 2 (bottom) with old (left) and new (right) injection optics for LHC. Lowering the octupolar RDTs has significantly improved the dynamic aperture at injection.

Courtesy S. Kostoglou

Beam lifetime x3 @ injection

MAD-NG studies: *“hand made” dynap (tunes)*

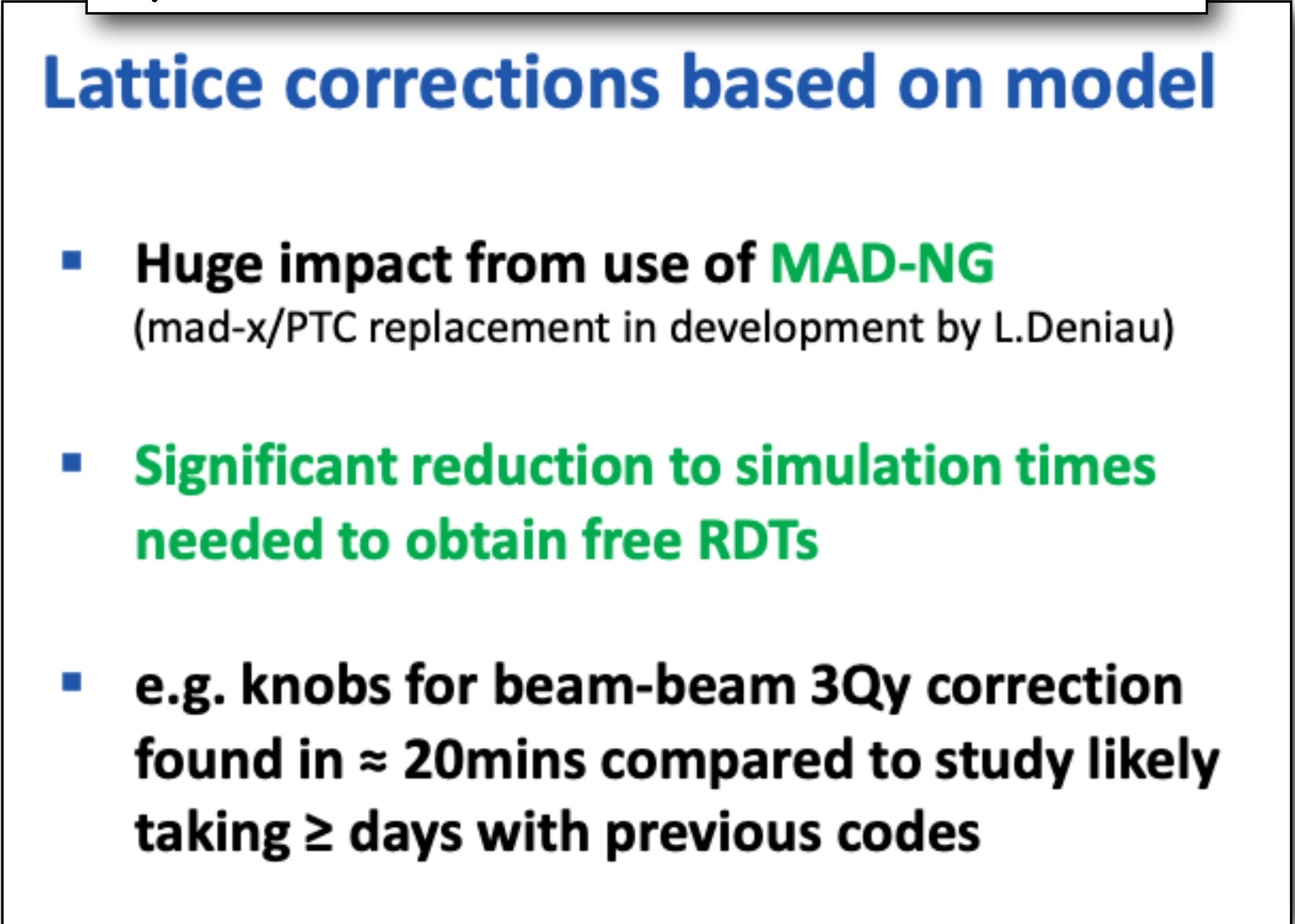
```
-- LHC setup
MADX:load("lhc_saved.seq", "lhc_saved.mad")
local lhcb1 in MADX
lhcb1.beam = beam {particle="proton", energy=450}

-- run twiss for tunes comparison
local tw = twiss {sequence=lhcb1}

-- track setup
local np  = 64          -- number of particles
local nt  = 1024        -- number of turns
local rho = 1e-6        -- amplitude  [m]
local ang = 90/(np-1) -- angle step [deg]
local X0  = {}          -- list of particles

-- create list of np particles (populate the phase space)
for i=1,np do
  X0[i] = {x=rho*cos(rad(ang*(i-1))),
           y=rho*sin(rad(ang*(i-1))), px=0,py=0,t=0,pt=0}
end

-- track np particles to collect turn-by-turn oscillations
local tk = track {sequence=lhcb1, nturn=nt, X0=X0}
assertf(tk.lost == 0, "unexpected %d particle(s) lost", tk.lost)

-- reshape coordinates for matrix operations
local xn = tk.x:copy():reshape(nt, np)
local yn = tk.y:copy():reshape(nt, np)

-- compute np real 1D-FFT, one for each particle (columns)
local xf = xn:rfft'col'
local yf = yn:rfft'col'

-- compute sum of amplitudes for each frequency (rows)
local qx = xf:sumabs'row':real()
local qy = yf:sumabs'row':real()

-- find max amplitude indexes (tunes)
local _, xi = qx:iminmax()
local _, yi = qy:iminmax()

-- print results
io.write("TUNES")
io.write("Qx_ref = ", tw.q1, ", Qx = ", (xi-1)*0.5/(#qx-1), "\n")
io.write("Qy_ref = ", tw.q2, ", Qy = ", (yi-1)*0.5/(#qy-1), "\n")
```

Results

```
TUNES
Qx_ref = 62.275051, Qx = 0.275390625
Qy_ref = 60.295050, Qy = 0.294921875

TIMINGS
Track: 146s
Other: < 1s

= 2.2ms / particle / turn
= 450 LHC_turns / s / particle
  (40 min for 1 000 000 turns)

Not so bad for a dynamic scripting language!
```

MAD-NG studies: *Testimonials*

Optics Measurements and Corrections Team



Lattice corrections based on model

- **Huge impact from use of MAD-NG**
  (mad-x/PTC replacement in development by L.Deniau)
- **Significant reduction to simulation times needed to obtain free RDTs**
- **e.g. knobs for beam-beam 3Qy correction found in ≈ 20mins compared to study likely taking ≥ days with previous codes**

OMC@LMC, E. Maclean
28/08/2024

Other Studies

CLIC final focus:
beam size optimisation including high order contributions (up to 7).

FCC-ee Q’, Q”, Q’”:
sextupole families optimisation, combining layouts vs tunnel.

PS, PSB:
model improvements for combined function magnets, RDTs.

GTPSA & Lie algebra:
used by some other codes.

PS Nonlinear studies
O. Naumenko, 27/09/2024

*The **measurement agrees** remarkably well with the PS model and the RDTs calculated by MAD-NG. The chromaticities predicted by MAD-NG also match well with the **chroma measurement** I have performed.*

Conclusions

- MAD-NG is reaching the end of its development process, **release 1.0** planned by end 2024.

- MAD-NG is now used in many CERN studies and machine optimisation and proved to be **accurate and efficient** to solve complex non-linear problems.

- MAD-NG was designed to **design methodically** new machines, where it should shine soon.

- MAD-NG **parametric normal forms** is a powerful tool to help understanding the sensitivity of quantities vs parameters (strengths, length, position, misalignments, cross-talk, etc…).

- Key features of MAD-NG vs MAD-X:
  - Better code architecture and structure, and much better physics.
  - Highly flexible and extensible for the physics (new features take a day).
  - Less surprise when **combining features** (e.g. slicing & misalign & frame & field errors).
  - Support backtracking, charged particles, parallel sequences, reversed sequences, etc…
  - Main stream programming language for scripting (save user time!) & **many toolboxes**.
  - Mature technologies, syntax error, backtrace, debugger, profiler, JIT (save user time!).