DYNK module in SixTrack:

Changing machine properties on a turn-by-turn basis

K. Sjobak, A. Santamaria,
R.D. Maria, A. Mereghetti, H. Burkhardt

HSS meeting, Jan 19$^{\text{th}}$ 2015

Outline

1 What is DYNK?

2 Why DYNK?

3 Using DYNK

4 Implementation of DYNK

5 Conclusions & Future

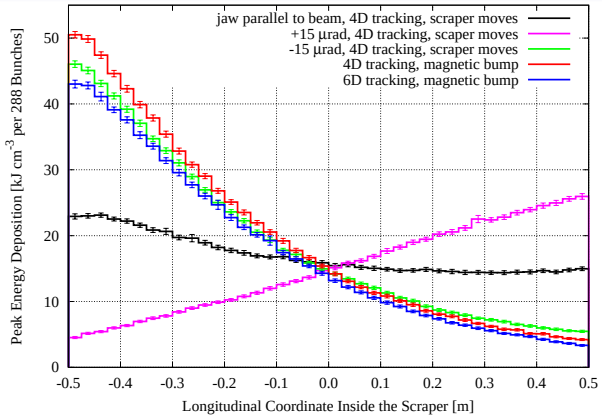
What is DYNK?

- DYNK = DYNamic Kicks
- Changing the parameters of a single element turn-by-turn
- Fully controlled by input files (new block in `fort.3`)
- Examples:
  - Crab amplitude and phase
  - Magnet strengths
- New values calculated using a simple and flexible mini-language
  - Also supports loading values from file
- Available[1] in mainline SixTrack code

[1] Next release

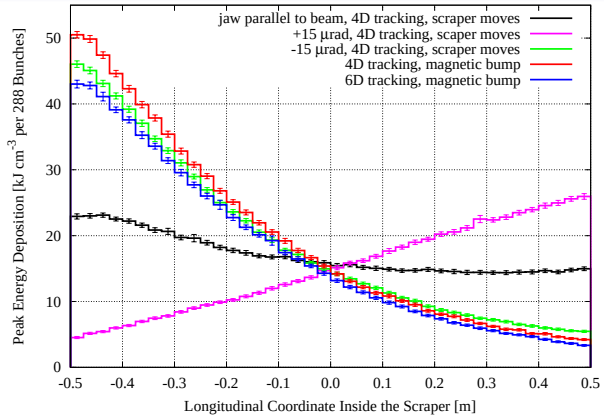
Why was DYNK written?

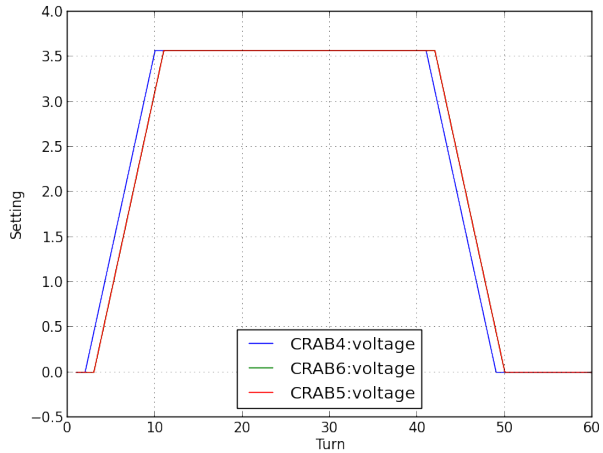
- Original purpose (A. Mereghetti):
  Magnetic bump scraping in the SPS for LIU
- Our goal:
  Simulating fast transients from crab cavities
  - Switching on/off
  - Failures
- Wanted a flexible and general tool
  - Can also be used for
    simulating fast optics changes etc.
  - Limiting proliferation of
    private versions of SixTrack

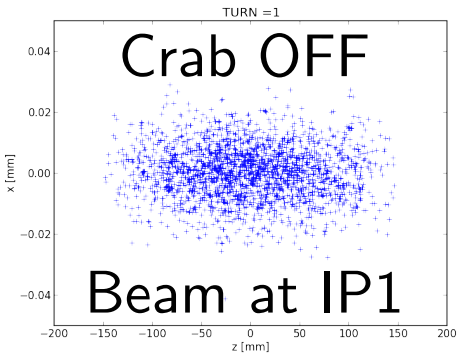


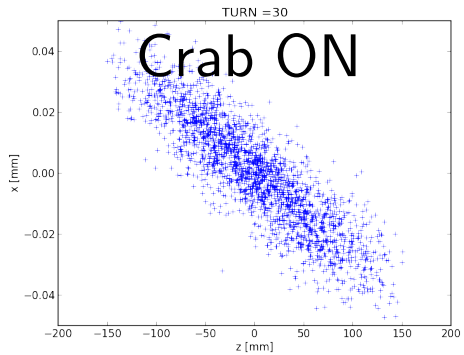
Why was DYNK written?

- Original purpose (A. Mereghetti):
  Magnetic bump scraping in the SPS for LIU
- Our goal:
  Simulating fast transients from crab cavities
  - Switching on/off
  - Failures
- Wanted a flexible and general tool
  - Can also be used for
    simulating fast optics changes etc.
  - Limiting proliferation of
    private versions of SixTrack









Using DYNK

New block in fort.3:

```
DYNK
FUN initial CONST 0.0
FUN ramp LINSEG 0 8 0 3.568626888
FUN plateau CONST 3.568626888
FUN failure LIN -0.44607836099999998 3.56862688
SET CRAB4 voltage initial 1 2 0
SET CRAB4 voltage ramp 3 10 -2
SET CRAB4 voltage plateau 11 41 0
SET CRAB4 voltage failure 42 49 -41
NEXT
```

Two main types of commands:

- `FUN <function name> <function type> arg1 arg2 arg3...`
- `SET <element name> <attribute name> <function name> <first turn> <last turn> <turn shift>`

Setting new values – WHAT?

```
SET <element name> <attribute name> <function name>
    <first turn> <last turn> <turn shift>
```

Setting new values – WHAT?

```
SET <element name> <attribute name> <function name>
    <first turn> <last turn> <turn shift>
```

Supported elements/attributes:

- Crab cavities
  - Voltage
  - Frequency
  - Phase
- Nonlinear elements (type 0 – ± 10)
  - Multipole strength K_n (before errors)
- Multipole blocks
  - Bending strength
  - Reference radius

Input in same units as in `fort.2` –
changing settings of `SINGLE ELEMENT`s

Defining functions – HOW?

```
FUN <function name> <function type> arg1 arg2 arg3...
```

- Can take any number of arguments
- May depend on each other

Defining functions – HOW?

```
FUN <function name> <function type> arg1 arg2 arg3...
```

- Can take any number of arguments
- May depend on each other
- Example function types:
  - `CONST <value>`
  - `TURN`
  - `LINSEG x1 x2 y1 y2`
  - `GET <element> <attribute>`
  - `FILELIN <filename>`
  - `RANDG <seed1> <seed2> <mu> <sigma> <mcut>`
  - `SQRT <function name>`
  - `SIN <function name>`
  - `ADD <fname1> <fname2>`

Full list of functions: `https://twiki.cern.ch/twiki/bin/view/LHCAtHome/SixTrackDoc#Syntax_of_FUN`

Defining functions – HOW?

```
FUN <function name> <function type> arg1 arg2 arg3...
```

- Can take any number of arguments
- May depend on each other
- Example function types:
  - `CONST <value>`
  - `TURN`
  - `LINSEG x1 x2 y1 y2`
  - `GET <element> <attribute>`
  - `FILELIN <filename>`
  - `RANDG <seed1> <seed2> <mu> <sigma> <mcut>`
  - `SQRT <function name>`
  - `SIN <function name>`
  - `ADD <fname1> <fname2>`

To define the function

$$A(t) = 3 + \sqrt{\sin\left(B(t)\right)}$$

where $B(t)$ is a linear segment intersecting $(t, B(t)) = (1, 0)$ and $(5, -2)$, one would write:

```
FUN B LINSEG 1 5 0 -2
FUN sinB SIN B
FUN sq SQRT sinB
FUN three CONST 3
FUN A ADD sq three
```

Full list of functions: `https://twiki.cern.ch/twiki/bin/view/LHCAtHome/SixTrackDoc#Syntax_of_FUN`

Defining functions – HOW?

```
FUN <function name> <function type> arg1 arg2 arg3...
```

- Can take any number of arguments
- May depend on each other
- Example function types:
  - `CONST <value>`
  - `TURN`
  - `LINSEG x1 x2 y1 y2`
  - `GET <element> <attribute>`
  - `FILELIN <filename>`
  - `RANDG <seed1> <seed2> <mu> <sigma> <mcut>`
  - `SQRT <function name>`
  - `SIN <function name>`
  - `ADD <fname1> <fname2>`

To define the function

$$A(t) = 3 + \sqrt{\sin\left(B(t)\right)}$$

where $B(t)$ is a linear segment intersecting $(t, B(t)) = (1, 0)$ and $(5, -2)$, one would write:

```
FUN B LINSEG 1 5 0 -2
FUN sinB SIN B
FUN sq SQRT sinB
FUN three CONST 3
FUN A ADD sq three
```

```
SET CRAB5 phase A 1 20 0
```

Full list of functions: `https://twiki.cern.ch/twiki/bin/view/LHCAtHome/SixTrackDoc#Syntax_of_FUN`

More FUN & SET examples

Damper frequency sweep

$$F(t) = \sin\left(2\pi B(t) t\right)$$

where $B(t)$ goes from 0.2 to 0.4 in 10^4 turns:

```
DYNK
FUN B LINSEG 0.2 0.4 0 10000
FUN p1 LIN 6.2831853071796 0
FUN p2 MUL p1 B
FUN F SIN p2
SET kicker1 average_ms
 ↪ 1000 11000 -1000
NEXT
```

The SET is active from turn 1000 to turn 11000, calling $F(t - 1000)$.

More FUN & SET examples

Damper frequency sweep

$$F(t) = \sin\left(2\pi B(t) t\right)$$

where $B(t)$ goes from 0.2 to 0.4 in 10^4 turns:

```
DYNK
FUN B LINSEG 0.2 0.4 0 10000
FUN p1 LIN 6.2831853071796 0
FUN p2 MUL p1 B
FUN F SIN p2
SET kicker1 average_ms
 ↪ 1000 11000 -1000
NEXT
```

The SET is active from turn 1000 to turn 11000, calling $F(t - 1000)$.

Power converter noise

Noise components at 50, 100, 300, ... Hz

$$F(t) = a_1 \sin\left(\frac{2\pi 90[\mu\mathrm{s}]}{(1/50)[\mathrm{s}]} t\right) + \cdots$$

```
FUN s1 SINF 8 0.028274
FUN s2 SINF 4 0.056549
FUN s3 SINF 2 0.169646
FUN a1 ADD s1 s2
FUN F ADD a1 s3
```

More FUN & SET examples

Damper frequency sweep

$$F(t) = \sin\left(2\pi B(t) t\right)$$

where $B(t)$ goes from 0.2 to 0.4 in 10^4 turns:

```
DYNK
FUN B LINSEG 0.2 0.4 0 10000
FUN p1 LIN 6.2831853071796 0
FUN p2 MUL p1 B
FUN F SIN p2
SET kicker1 average_ms
 ↪ 1000 11000 -1000
NEXT
```

The SET is active from turn 1000 to turn 11000, calling $F(t - 1000)$.

Power converter noise

Noise components at 50, 100, 300, ... Hz

$$F(t) = a_1 \sin\left(\frac{2\pi 90[\mu\mathrm{s}]}{(1/50)[\mathrm{s}]} t\right) + \cdots$$

```
FUN s1 SINF 8 0.028274
FUN s2 SINF 4 0.056549
FUN s3 SINF 2 0.169646
FUN a1 ADD s1 s2
FUN F ADD a1 s3
```

Random noise

Gaussian random value with $\sigma = 0.2$ and $\mu = 45.0$:

```
FUN G RANDG 12345 67890
   ↪ 45 0.2 0
```

Implementation – data structure

Data stored in common blocks defined in block `comdynk`

- FUN statements stored in one table
  - 1 row per FUN
  - Columns: Name (index in cexpr_dynk), function type, 3×free
  - Arrays available with "allocatable" memory, storing integers, reals and strings

```
integer funcs_dynk (maxfuncs_dynk,5)                !Main FUN table
integer iexpr_dynk (maxdata_dynk)                   !Free storage (integers)
double precision fexpr_dynk (maxdata_dynk)          !Free storage (doubles)
character(maxstrlen_dynk) cexpr_dynk(maxdata_dynk)!Free storage (strings)

integer nfuncs_dynk, niexpr_dynk, nfexpr_dynk, ncexpr_dynk !Allocation
```

Implementation – data structure

Data stored in common blocks defined in block `comdynk`

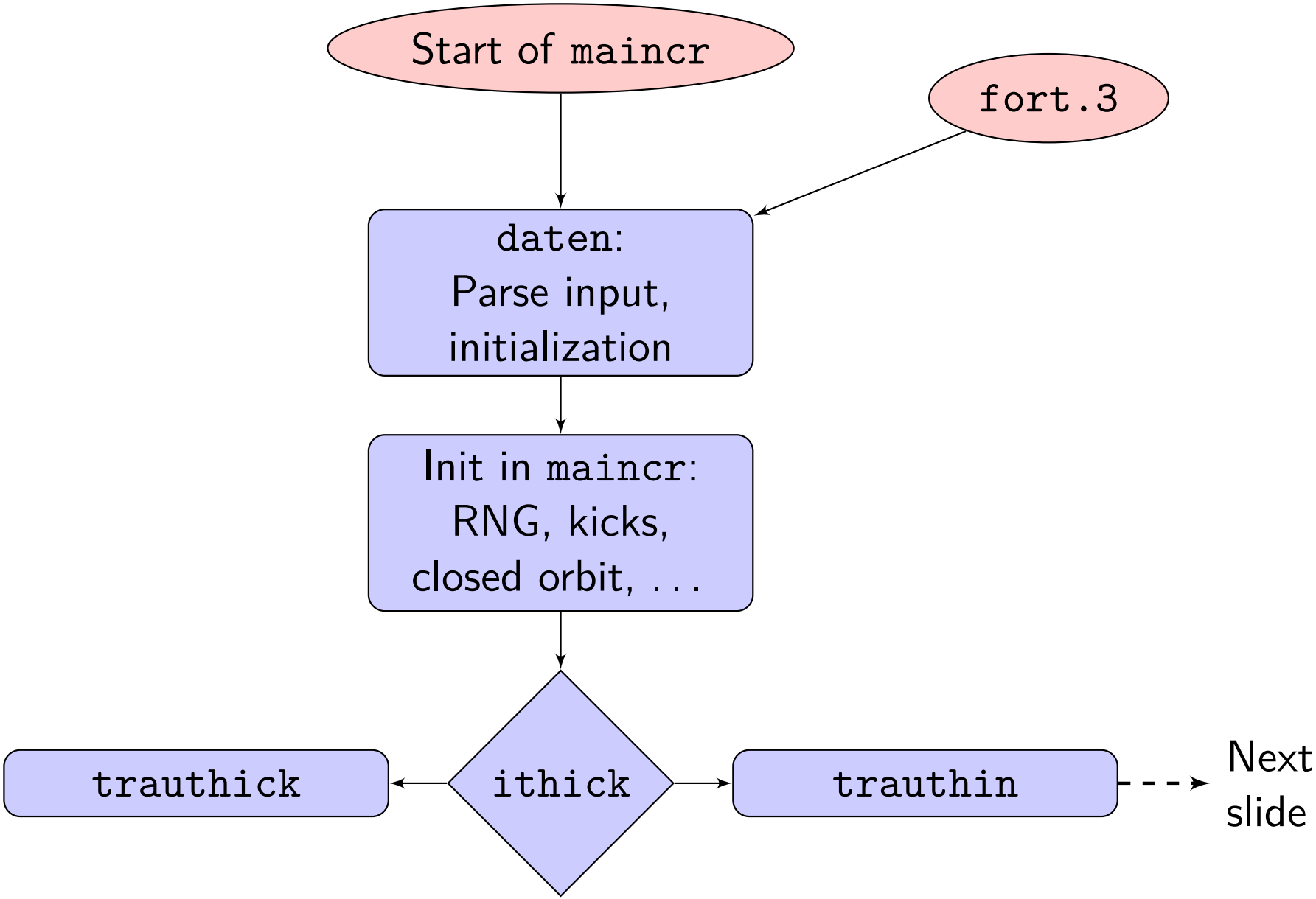
- FUN statements stored in one table
  - 1 row per FUN
  - Columns: Name (index in cexpr_dynk), function type, 3×free
  - Arrays available with "allocatable" memory, storing integers, reals and strings

```
integer funcs_dynk (maxfuncs_dynk,5)               !Main FUN table
integer iexpr_dynk (maxdata_dynk)                  !Free storage (integers)
double precision fexpr_dynk (maxdata_dynk)         !Free storage (doubles)
character(maxstrlen_dynk) cexpr_dynk(maxdata_dynk)!Free storage (strings)

integer nfuncs_dynk, niexpr_dynk, nfexpr_dynk, ncexpr_dynk !Allocation
```

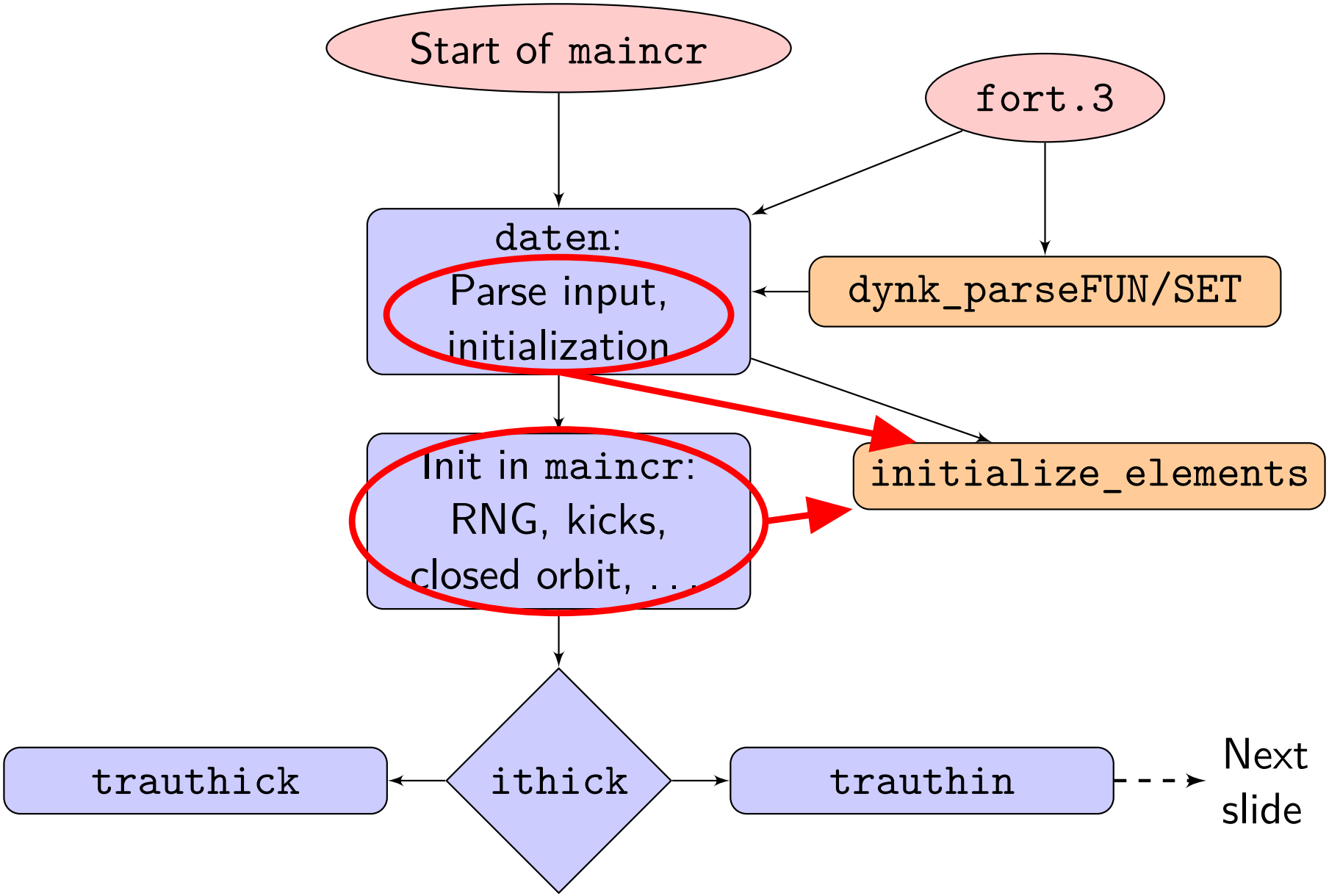
- Two similar tables for SET statements
  - Columns: Function index, turn limits, turn shift
  - Columns: Element name, attribute name
  - Also store pre-tracking values

```
integer sets_dynk(maxsets_dynk, 4)                     !  SET table (ints)
character(maxstrlen_dynk) csets_dynk (maxsets_dynk,2) !  SET table (names)
integer nsets_dynk

character(maxstrlen_dynk) csets_unique_dynk (maxsets_dynk,2) !  Store the pre-tracking
double precision fsets_origvalue_dynk(maxsets_dynk)          !  settings from fort.2
```

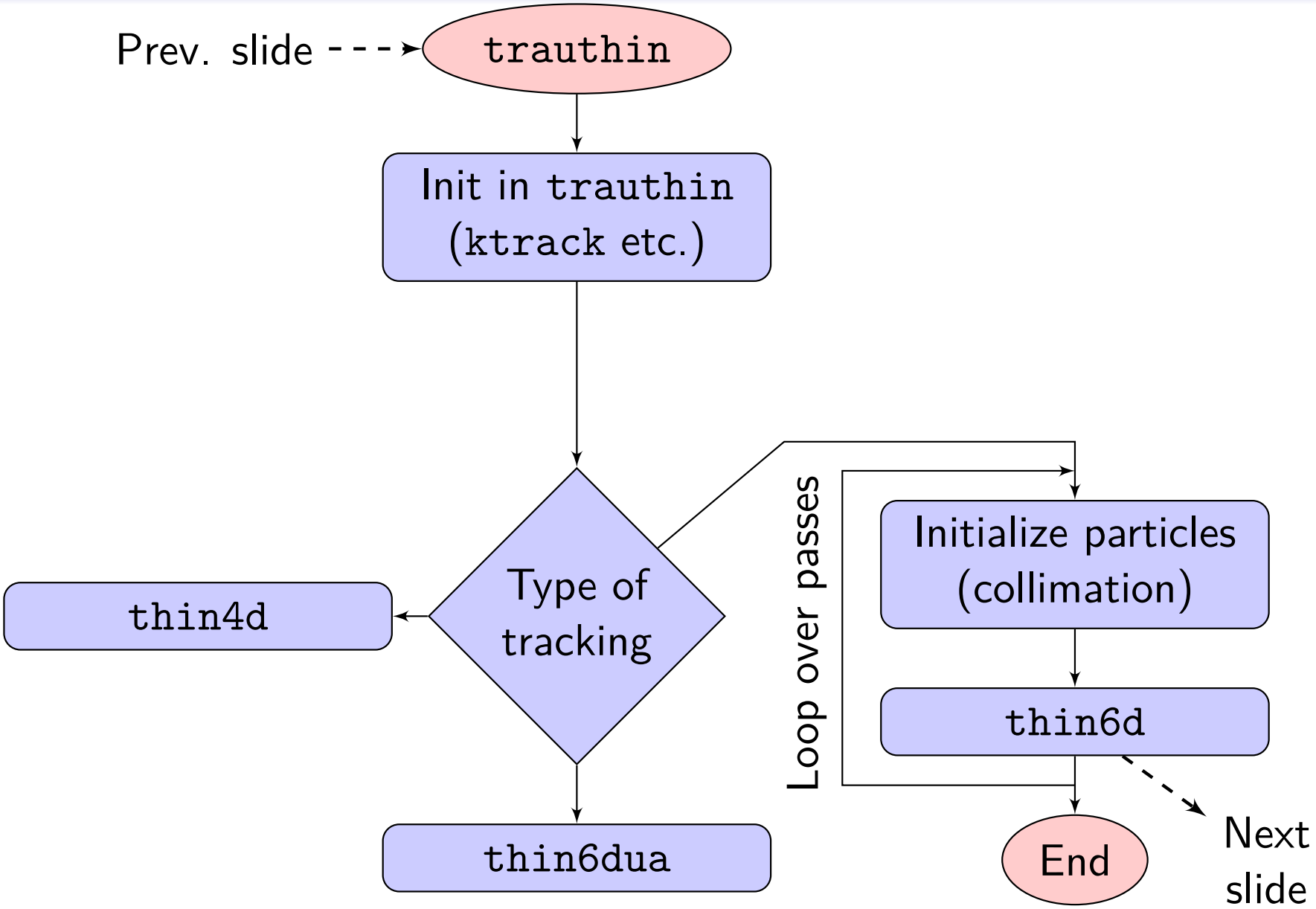
Implementation – program flow (1/3)



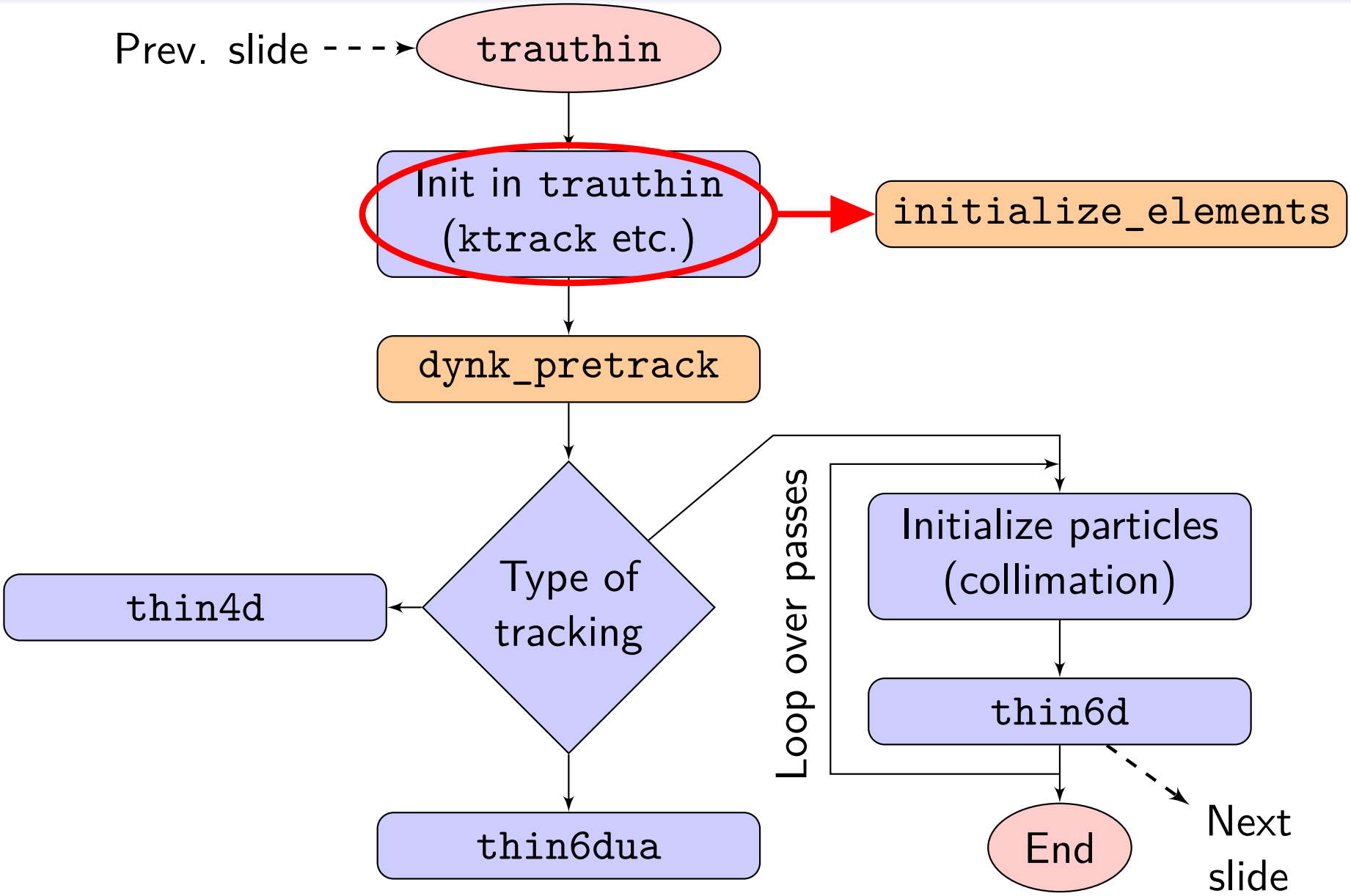
Implementation – program flow (1/3)



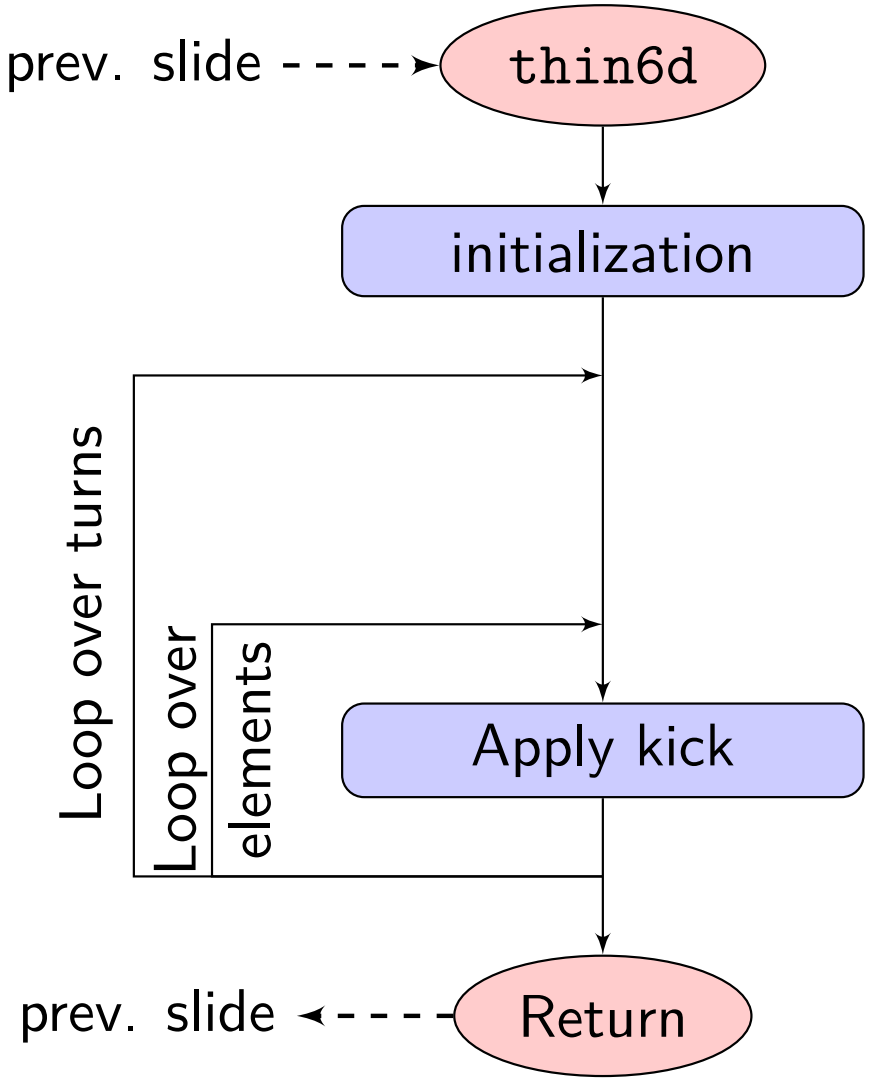
Implementation – program flow (2/3)



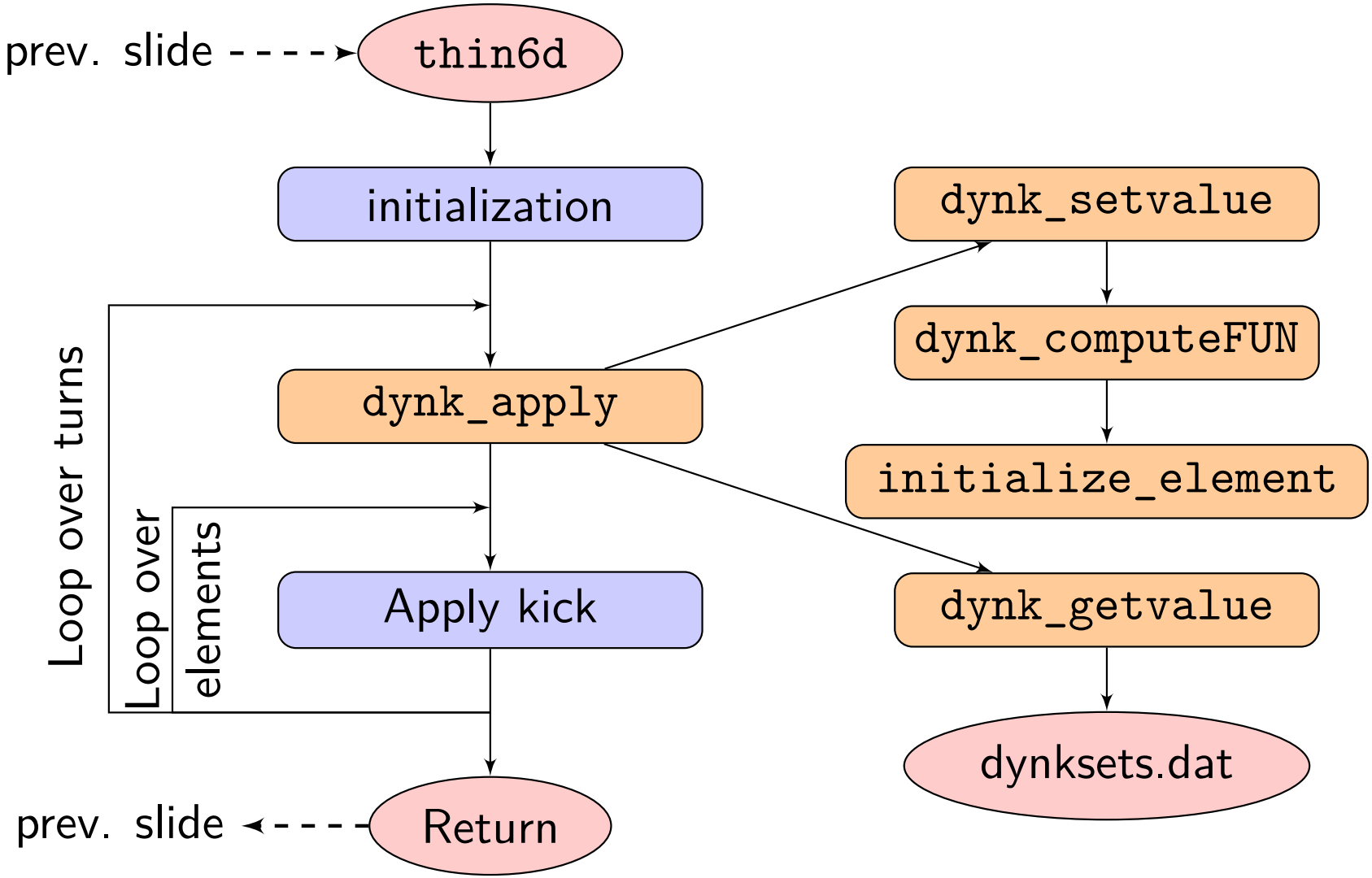
Implementation – program flow (2/3)



Implementation – program flow (3/3)



Implementation – program flow (3/3)



Implementation – Adding new FUNs

In principle

- Allocate a function index
- Add code to the functions `dynk_computeFUN` and `dynk_parseFUN`

Implementation – Adding new FUNs

In principle

- Allocate a function index
- Add code to the functions dynk_computeFUN and dynk_parseFUN

Code example – evaluation:

```
recursive double precision function
& dynk_computeFUN( funNum, turn ) result(retval)
...
select case ( funcs_dynk(funNum,2) )
...
case( 32 )
retval = sin( dynk_computeFUN(
&             funcs_dynk(funNum,3),turn) )
...
end select
end function
```

Code example – initialization: FUN SIN <function name>

```
subroutine dynk_parseFUN( getfields_fields,
& getfields_lfields,getfields_nfields )
...
select case ( getfields_fields(3)
& (1:getfields_lfields(3)) )
...
case("SIN")[a]
!  DATA: Name, Type, function index, -, -
funcs_dynk(nfuncs_dynk,1) = ncexpr_dynk
funcs_dynk(nfuncs_dynk,2) = 32
funcs_dynk(nfuncs_dynk,3) =
&    dynk_findFUNindex(getfields_fields(4)
&    (1:getfields_lfields(4)), 1)

!  NAME
cexpr_dynk(ncexpr_dynk)(1:getfields_lfields(2))
& = getfields_fields(2)(1:getfields_lfields(2))
...
end select
end subroutine
```

[a] Some boilerplate code, incl. input sanity checks, is omitted

Implementation – Adding new elements / attributes

- Add the element to `dynk_setvalue` and `dynk_getvalue`
- If the element uses data from other variables than `ed`, `ek` and `el` for kicking:
  Add code to `initialize_element`
- Sometimes ugly interactions occur...
  - Initialization spread thin throughout the code
  - Other elements depending (indirectly) on this setting

Current status & Future plans

- Most functionality in place – to be released with next version of SixTrack (v4.5.19)
  - This will also contain improvements from FLUKA team:
    Online aperture check,
    online exchange of particles SixTrack $\leftrightarrow$ FLUKA
  - New output files (DUMP, STAT, BMAT)
- Current code available from devel branch at
  `https://github.com/KFubuki/time_dependant_kicks/`
- Missing pieces:
  - Formal testing with SixDesk:
    running old tests and adding new tests
  - Checkpoint/restart
  - Output with crlibm (`lout`)
- Also interesting to look at other limitations in SixTrack
  - Output file sizes

Current status & Future plans

- Most functionality in place – to be released with next version of SixTrack (v4.5.19)
  - This will also contain improvements from FLUKA team:
    Online aperture check,
    online exchange of particles SixTrack $\leftrightarrow$ FLUKA
  - New output files (DUMP, STAT, BMAT)
- Current code available from devel branch at
  `https://github.com/KFubuki/time_dependant_kicks/`
- Missing pieces:
  - Formal testing with SixDesk:
    running old tests and adding new tests
  - Checkpoint/restart
  - Output with crlibm (`lout`)
- Also interesting to look at other limitations in SixTrack
  - Output file sizes

How DYNK will be presented – IPAC'15

- DYNK:
  "General functionality for
  turn-dependent element properties in SixTrack"
  - Same authors as this talk
- Failure scenarios w/ Crab Cavities in HL LHC
  - "Limits on failure scenarios for crab cavities in the HL-LHC"
  - Study how fast crab cavities can fail
    before it will damage the machine & detectors
  - Will calculate lossmaps in the HL-LHC for failure cases
  - Establish limits for what failures can be dealt with,
    and what can damage the machine and experiments
  - Need a good aperture model
  - Utilizes DYNK functionality –
    crab cavities already well supported

How DYNK will be presented – IPAC'15

- DYNK:
  "General functionality for
  turn-dependent element properties in SixTrack"
  - Same authors as this talk
- Failure scenarios w/ Crab Cavities in HL LHC
  - "Limits on failure scenarios for crab cavities in the HL-LHC"
  - Study how fast crab cavities can fail
    before it will damage the machine & detectors
  - Will calculate lossmaps in the HL-LHC for failure cases
  - Establish limits for what failures can be dealt with,
    and what can damage the machine and experiments
  - Need a good aperture model
  - Utilizes DYNK functionality –
    crab cavities already well supported

BACKUP SLIDES

Implementation – Main functions

`dynk_parseFUN(...)` and `dynk_parseSET(...)`

- Interprets DYNK block lines, filling data structure
- Called from `daten`

Implementation – Main functions

dynk_parseFUN(...) and dynk_parseSET(...)

- Interprets DYNK block lines, filling data structure
- Called from daten

dynk_pretrack()

- Saves original settings
- Called from trauthin and trauthck

dynk_apply(turn)

- Applies new settings
- Called from thin4d, thin6d, thin6dua, etc.

Implementation – Main functions

dynk_parseFUN(...) and dynk_parseSET(...)

- Interprets DYNK block lines, filling data structure
- Called from daten

dynk_pretrack()

- Saves original settings
- Called from trauthin and trauthck

dynk_apply(turn)

- Applies new settings
- Called from thin4d, thin6d, thin6dua, etc.

dynk_getvalue(el,att) and dynk_setvalue(el,att)

- Applies or retrieves the setting of an element/attribute

dynk_computeFUN(funNum, t)

- Computes the value of a given function
- May recursively call itself
- Called from dynk_setvalue

initialize_element(ix,lfrst)

- Initializes extra variables
- Called by daten and dynk_setvalue

General functionality for turn-dependent element properties in SixTrack

Kyrre Sjobak, Andrea Santamaria, Riccardo De Maria, Alessio Mereghetti, Helmut Burkhardt

In order to facilitate studies of how dynamically changing element attributes affect the dynamics of the beam and beam losses, the functionality for dynamic kicks (DYNK) of SixTrack has been significantly extended. This functionality can be used for the simulation of dynamic scenarios, such as when crab cavities are switched on, orbit bumps are applied, optics are changed, or failures occur. The functionality has been extended with a more general and flexible implementation, such that arbitrary time-dependent functions can be defined and applied to different attributes of families or individual elements, directly from the user input files. This removes the need for source code manipulation, and it is compatible with LHC@Home which offers substantial computing resources from volunteers. In this paper, the functionality and implementation of DYNK is discussed, along with examples of application to the HL-LHC crab cavities.

Limits on failure scenarios for crab cavities in the HL-LHC

A. Santamaria, R. Bruce, H. Burkhardt, K. Hernandez Chahin, A. Macpherson, S. Redaelli, K. Sjobak, D. Wollmann, B. Yee-Rendon

The High Luminosity (HL) LHC upgrade aims for a tenfold increase in integrated luminosity compared to the nominal LHC and for operating at a levelled luminosity of 5e34cm-2s-1, which is five times higher than the nominal LHC peak luminosity. Crab Cavities (CCs) are planned to compensate the geometric luminosity loss created by the increased crossing angle by rotating the bunch, allowing quasi head-on collisions at the Interaction Points (IP). As the CCs work by creating strong transverse kicks, a failure may have short time constants comparable to the reaction time of the machine protection system, and can produce significant betatron oscillations and fast emittance growth. We study CC failure modes and their effects on the beam by simulation with SixTrack. For this goal, the foreseen aperture and lattice has been used for tracking studies, where we use the newly added functionality of SixTrack which allows to dynamically change the attributes of the CC. This study presents the first values of the acceptable change rates for the phase and voltage of the CCs in a failure, before damage to the machine due to fast losses may occur.