History and Adoption of Programming Languages in NHEP

Jim Pivarski

Princeton University – IRIS-HEP

February 8, 2202

This talk is a historical overview of widely used programming languages in NHEP, focusing on the motivations for each language's adoption.

This talk is a historical overview of widely used programming languages in NHEP, focusing on the motivations for each language's adoption.

There have always been physicists at the edge, trying out new languages, but most physicists only use one or two. The field as a whole changes slowly.

This talk is a historical overview of widely used programming languages in NHEP, focusing on the motivations for each language's adoption.

There have always been physicists at the edge, trying out new languages, but most physicists only use one or two. The field as a whole changes slowly.

Thesis: (1) change motivated more by "pain points" than incremental benefits,

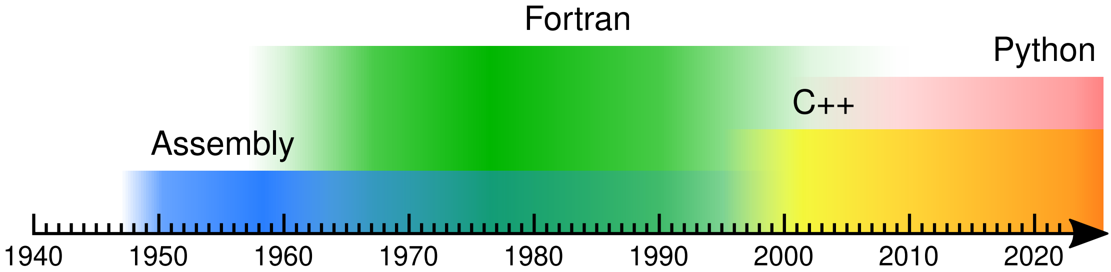
This talk is a historical overview of widely used programming languages in NHEP, focusing on the motivations for each language's adoption.

There have always been physicists at the edge, trying out new languages, but most physicists only use one or two. The field as a whole changes slowly.

Thesis: (1) change motivated more by "pain points" than incremental benefits,

(2) though each of these transitions happened in a unique way.

Three major transitions (so far)



Adoption of

Fortran: immediate; for syntax and portability; no infrastructure to replace

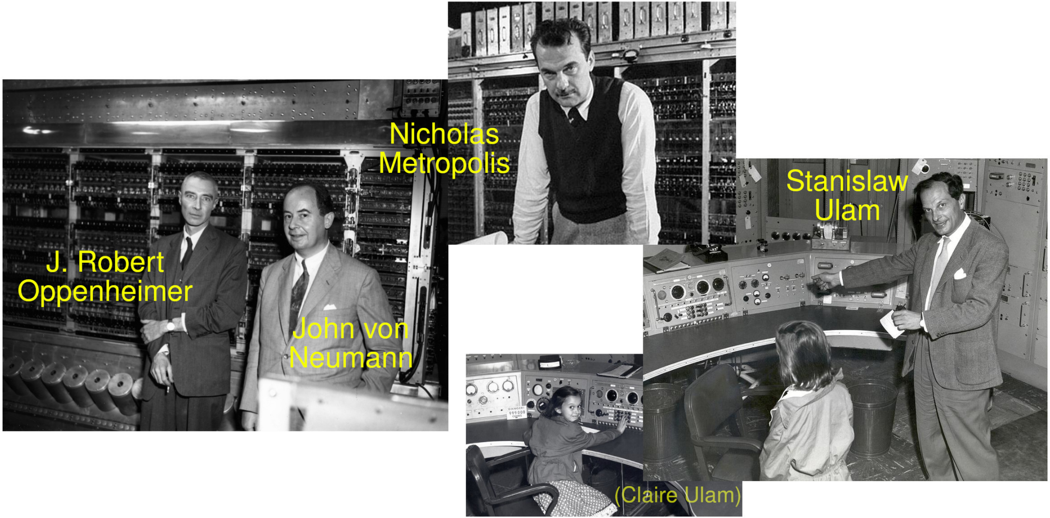
C++: long overdue; for data structures; replaced infrastructure in a burst

Python: slowly overtook its alternatives; for interactivity; different niche

Part 1: Fortran

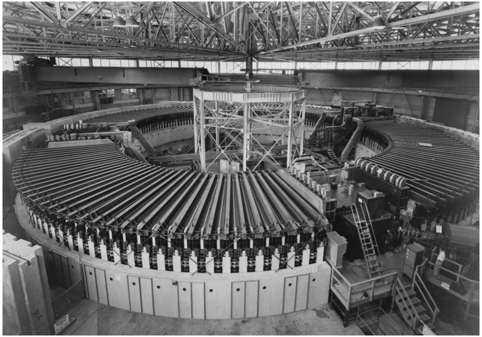
NHEP was an early adopter of digital computers

One of the very first applications was Monte Carlo (neutron transport).

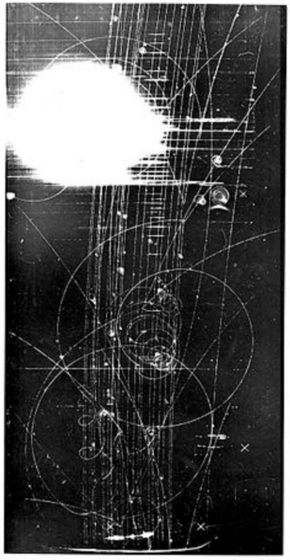


And so was data analysis

Luis Alvarez's group at the Bevatron: $2M bubble chamber, $0.2M IBM 650.

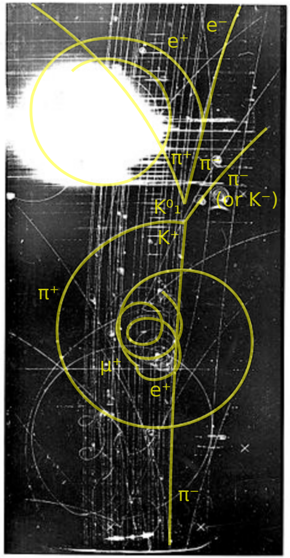


The problem was the same then as it is now



Unlabeled photos come out of the detector.

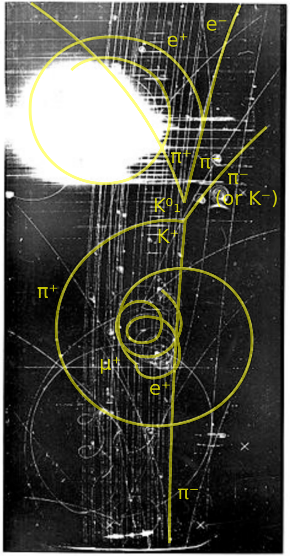
The problem was the same then as it is now



Unlabeled photos come out of the detector.

Labeling them turns them into quantities to compute.

The problem was the same then as it is now



Unlabeled photos come out of the detector.

Labeling them turns them into quantities to compute.

THE MORE EVENTS THE BETTER!!!

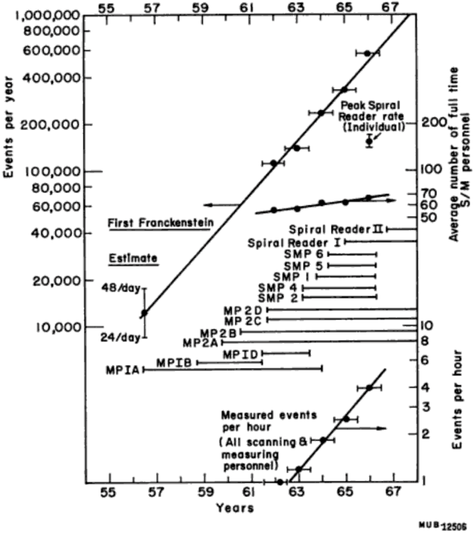
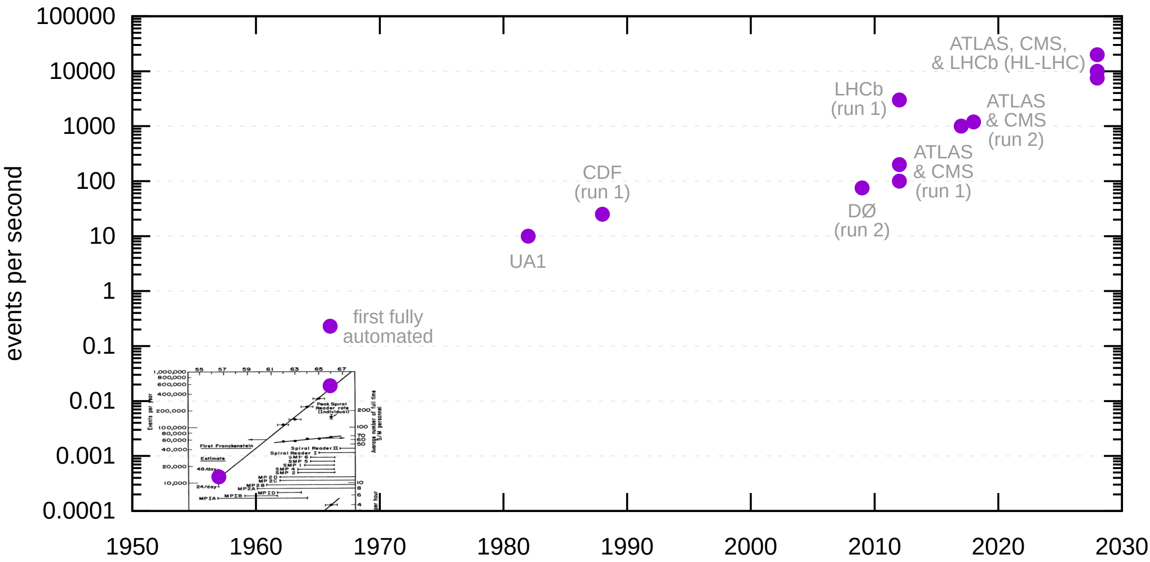


Fig. 9.
Measuring Rates.

The problem was the same then as it is now



Identifying tracks was beyond the capabilities of software

So they invented special input devices to streamline data-entry.

Identifying tracks was beyond the capabilities of software

Madeleine (née Goldstein)
Isenberg, UCLA class of ’65

“We scanners would review each frame of film, and per the brief instructions we had been given, looked for any ‘unusual activity.’

“The scanner had to use both hands, a joystick in each, and turn them clockwise or anti-clockwise, to align a double crosshair cursor at several sequential positions on a track.”

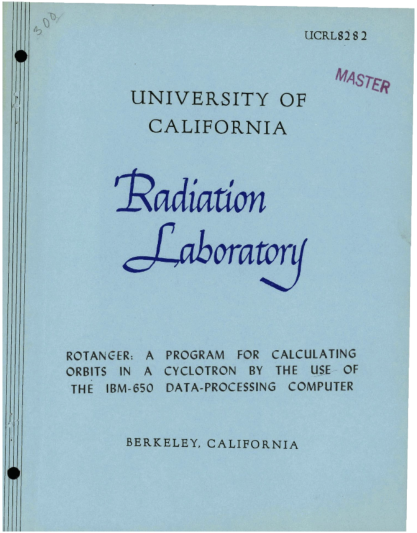
Identifying tracks was beyond the capabilities of software

Madeleine (née Goldstein) Isenberg, UCLA class of ’65

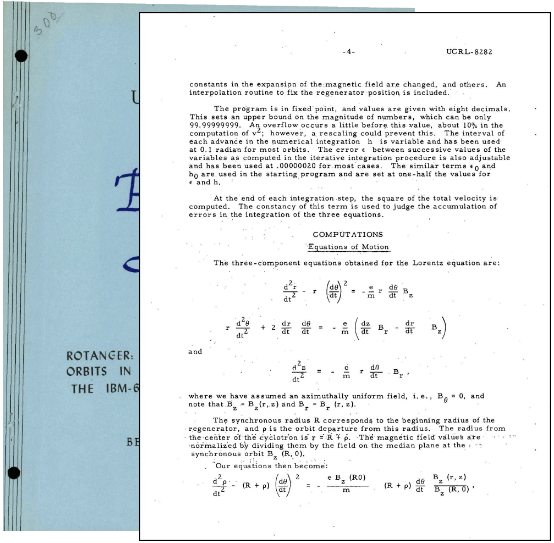
“A quick but firm tap on the foot-pedal punched the coordinate values onto an IBM card that had been fed into the keypunch machine.

“The precious stack of IBM cards were passed to the physicists, who would then process the data in the existing IBM processors, using software that would calculate the best fit for these coordinates, and thereby mathematically simulate the curvature of the track.”

At first, the software was written in assembly (example from 1958)



At first, the software was written in assembly (example from 1958)



constants in the expansion of the magnetic field are changed, and others. An interpolation routine to fix the regenerator position is included.

The program is in fixed point, and values are given with eight decimals. This sets an upper bound on the magnitude of numbers, which can be only 99.99999999. An overflow occurs a little before this value, about 10% in the computation of v^2; however, a rescaling could prevent this. The interval of each advance in the numerical integration h is variable and has been used at 0.1 radian for most orbits. The error ϵ between successive values of the variables as computed in the iterative integration procedure is also adjustable and has been used at .00000020 for most cases. The similar terms ϵ_0 and h_0 are used in the starting program and are set at one-half the values for ϵ and h.

At the end of each integration step, the square of the total velocity is computed. The constancy of this term is used to judge the accumulation of errors in the integration of the three equations.

COMPUTATIONS

Equations of Motion

The three-component equations obtained for the Lorentz equation are:

$$\frac{d^2r}{dt^2} - r\left(\frac{d\theta}{dt}\right)^2 = -\frac{e}{m}\, r\, \frac{d\theta}{dt} B_z$$

$$r\frac{d^2\theta}{dt^2} + 2\frac{dr}{dt}\frac{d\theta}{dt} = -\frac{e}{m}\left(\frac{dz}{dt} B_r - \frac{dr}{dt} B_z\right)$$

and

$$\frac{d^2z}{dt^2} = -\frac{e}{m}\, r\, \frac{d\theta}{dt} B_r ,$$

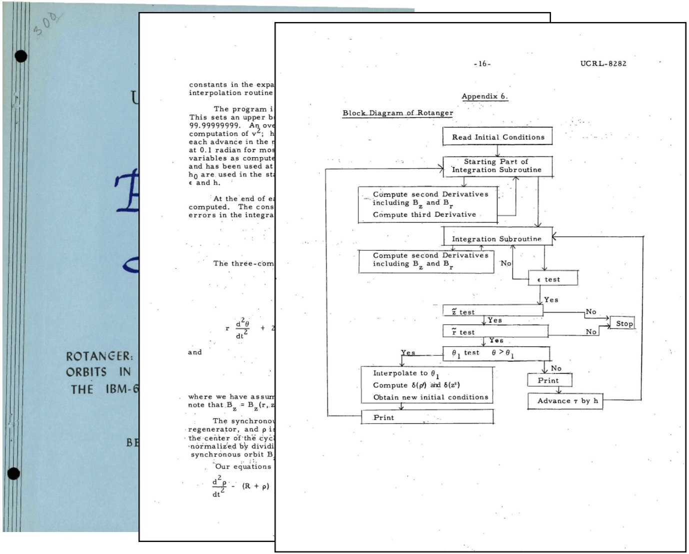
where we have assumed an azimuthally uniform field, i.e., $B_\theta = 0$, and note that $B_z = B_z(r, z)$ and $B_r = B_r(r, z)$.

The synchronous radius R corresponds to the beginning radius of the regenerator, and ρ is the orbit departure from this radius. The radius from the center of the cyclotron is $r = R + \rho$. The magnetic field values are normalized by dividing them by the field on the median plane at the synchronous orbit $B_z(R, 0)$.

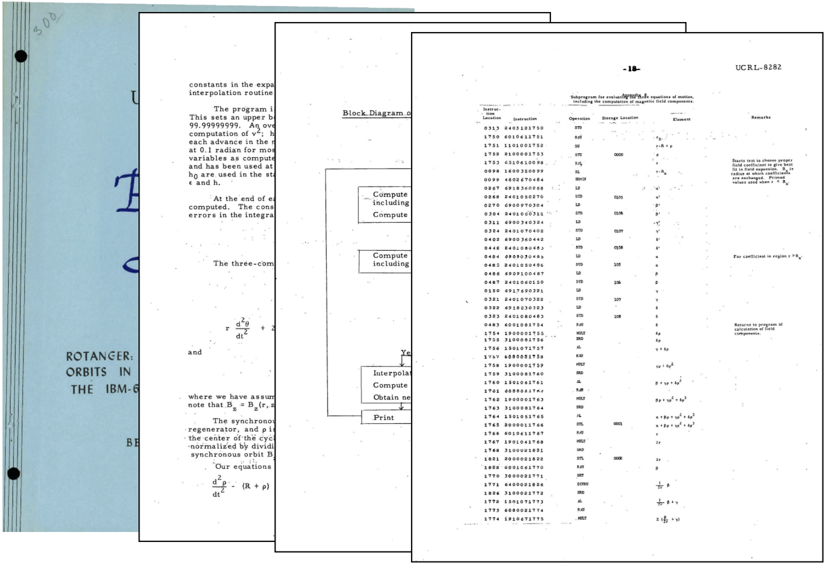
Our equations then become:

$$\frac{d^2\rho}{dt^2} - (R+\rho)\left(\frac{d\theta}{dt}\right)^2 = -\frac{e\, B_z(R0)}{m}(R+\rho)\frac{d\theta}{dt}\frac{B_z(r, z)}{B_z(R, 0)},$$

At first, the software was written in assembly (example from 1958)



At first, the software was written in assembly (example from 1958)



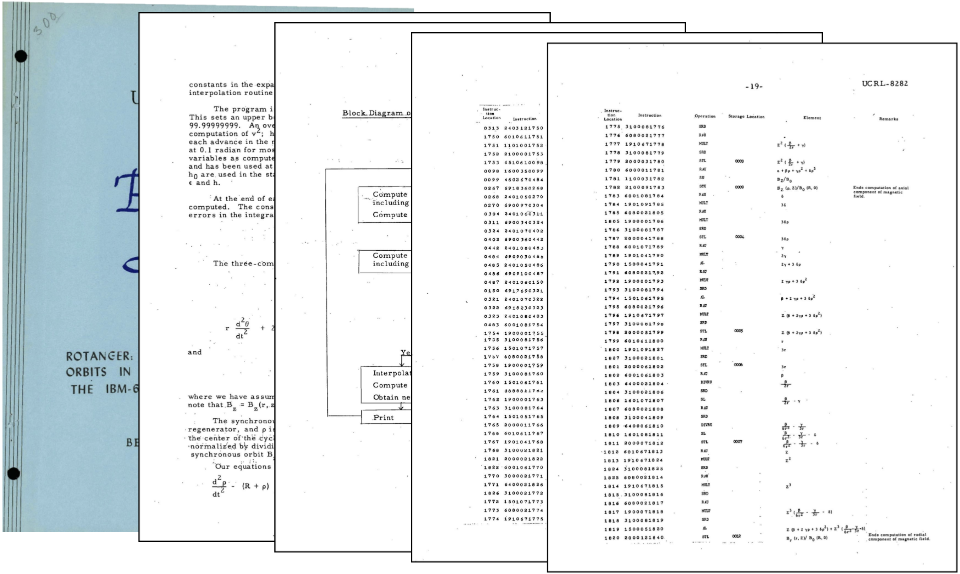
-18- UCRL-8282

Appendix B

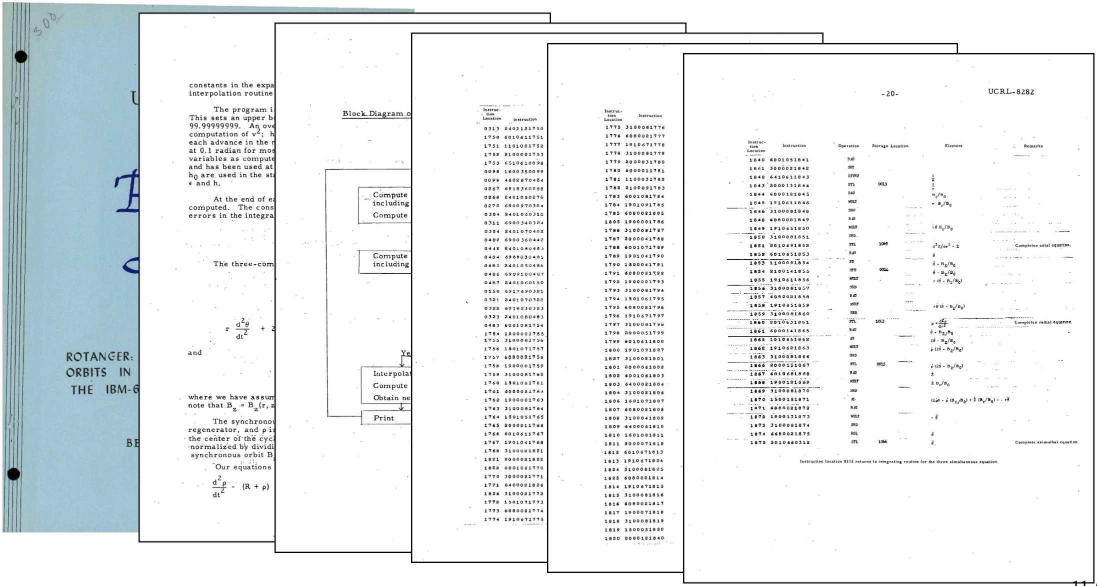
Subprogram for evaluating the three equations of motion, including the computation of magnetic field components.

Instruction Location	Instruction	Operation	Storage Location	Element	Remarks
0313	2403121750	STD			
1750	6010611751	RAU		r_0	
1751	1101001752	SU		$r-R=\rho$	
1752	2100001753	STU	0000	ρ	
1753	6510610098	RAL		r	Starts test to choose proper field coefficient to give best fit in field expansion. R_x is radius at which coefficients are exchanged. Primed values used when $r < R_x$.
0098	1600350099	SL		$r-R_x$	
0099	4602670484	BRMIN			
0267	6918360268	LD		α'	
0268	2401050270	STD	0105	α'	
0270	6900970304	LD		β'	
0304	2401060311	STD	0106	β'	
0311	6900340324	LD		γ'	
0324	2401070402	STD	0107	γ'	
0402	6900360442	LD		δ'	
0442	2401080483	STD	0108	δ'	
0484	6909030485	LD		α	For coefficient in region $r > R_x$.
0485	2401050486	STD	105	α	
0486	6909100487	LD		β	
0487	2401060150	STD	106	β	
0150	6917690321	LD		γ	
0321	2401070322	STD	107	γ	
0322	6918230323	LD		δ	
0323	2401080483	STD	108	δ	
0483	6001081754	RAU		δ	Returns to program of calculation of field components.
1754	1900001755	MULT		$\delta\rho$	
1755	3100081756	SRD		$\delta\rho$	
1756	1501071757	AL		$\gamma + \delta\rho$	
1757	6080021758	RAU			
1758	1900001759	MULT		$\gamma\rho + \delta\rho^2$	
1759	3100081760	SRD			
1760	1501061761	AL		$\beta + \gamma\rho + \delta\rho^2$	
1761	6080021762	RAU			
1762	1900001763	MULT		$\beta\rho + \gamma\rho^2 + \delta\rho^3$	
1763	3100081764	SRD			
1764	1501051765	AL		$\alpha + \beta\rho + \gamma\rho^2 + \delta\rho^3$	
1765	2000011766	STL	0001	$\alpha + \beta\rho + \gamma\rho^2 + \delta\rho^3$	
1766	6010611767	RAU		r	
1767	1901041768	MULT		2r	
1768	3100021821	SRD			
1821	2000021822	STL	0002	2r	
1822	6001061770	RAU		β	
1770	3000021771	SRT			
1771	6400021826	DIVRU		$\frac{1}{2r}\beta$	
1826	3100021772	SRD			
1772	1501071773	AL		$\frac{1}{2r}\beta + \gamma$	
1773	6080021774	RAU			
1774	1910671775	MULT		$z(\frac{\beta}{2r} + \gamma)$	

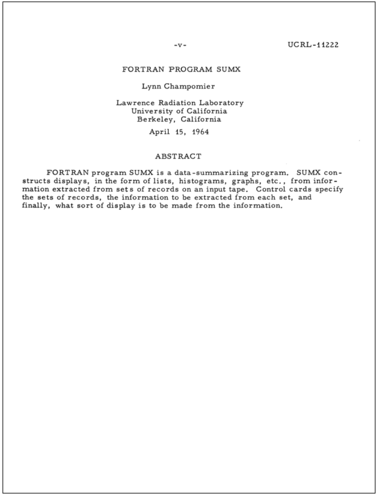
At first, the software was written in assembly (example from 1958)



At first, the software was written in assembly (example from 1958)



But it was quickly replaced by Fortran (SUMX from 1964)



-v- UCRL-11222

FORTRAN PROGRAM SUMX

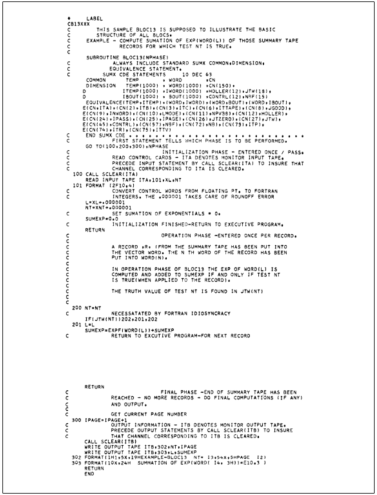
Lynn Champomier

Lawrence Radiation Laboratory
University of California
Berkeley, California

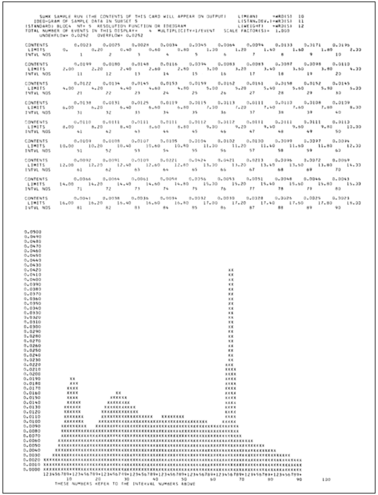
April 15, 1964

ABSTRACT

FORTRAN program SUMX is a data-summarizing program. SUMX constructs displays, in the form of lists, histograms, graphs, etc., from information extracted from sets of records on an input tape. Control cards specify the sets of records, the information to be extracted from each set, and finally, what sort of display is to be made from the information.

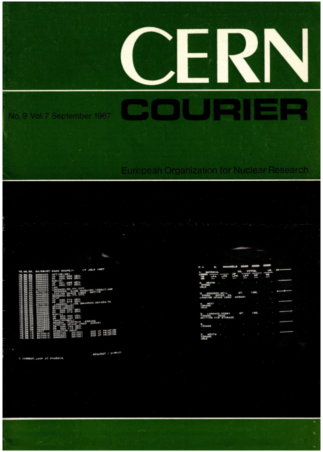


```
*     LABEL
CB13XXX
C          THIS SAMPLE BLOC13 IS SUPPOSED TO ILLUSTRATE THE BASIC
C          STRUCTURE OF ALL BLOCS.
C      EXAMPLE - COMPUTE SUMATION OF EXP(WORD(L)) OF THOSE SUMMARY TAPE
C                RECORDS FOR WHICH TEST NT IS TRUE.
C
      SUBROUTINE BLOC13(NPHASE)
C              ALWAYS INCLUDE STANDARD SUMX COMMON,DIMENSION,
C             EQUIVALENCE STATEMENT.
C           SUMX CDE STATEMENTS      10 DEC 63
      COMMON      TEMP        , WORD       ,CN
      DIMENSION   TEMP(1000) , WORD(1000) ,CN(150),
     D            ITEMP(1000) ,IWORD(1000) ,HOLLER(12),JTW(18),
     D            IBOUT(1000) , BOUT(1000) ,CONTRL(12),NBF(15)
      EQUIVALENCE(TEMP,ITEMP),(WORD,IWORD),(WORD,BOUT),(WORD,IBOUT),
     E(CN,ITA),(CN(2),ITB),(CN(3),ITC),(CN(6),ITTAPE),(CN(8),JGOOD),
     E(CN(9),INWORD),(CN(10),LMODE),(CN(11),NPV38),(CN(12),HOLLER),
     E(CN(24),IPASS),(CN(25),IPAGE),(CN(26),JTZERO),(CN(27),JTW),
     E(CN(45),CONTRL),(CN(57),NBF),(CN(72),NB),(CN(73),ITP),
     E(CN(74),ITR),(CN(75),ITTV)
C     END SUMX CDE . . . . . . . . . . . . . . . . . . . . . . . . . .
C              FIRST STATEMENT TELLS WHICH PHASE IS TO BE PERFORMED.
      GO TO(100,200,300),NPHASE
C                             INITIALIZATION PHASE - ENTERED ONCE / PASS.
C              READ CONTROL CARDS - ITA DENOTES MONITOR INPUT TAPE.
C              PRECEDE INPUT STATEMENT BY CALL SCLEAR(ITA) TO INSURE THAT
C              CHANNEL CORRESPONDING TO ITA IS CLEARED.
  100 CALL SCLEAR(ITA)
      READ INPUT TAPE ITA,101,XL,NT
  101 FORMAT (2F10.4)
C              CONVERT CONTROL WORDS FROM FLOATING PT. TO FORTRAN
C              INTEGERS. THE .000001 TAKES CARE OF ROUNOFF ERROR
      L=XL+.000001
      NT=XNT+.000001
C              SET SUMATION OF EXPONENTIALS = 0.
      SUMEXP=0.0
C              INITIALIZATION FINISHED-RETURN TO EXECUTIVE PROGRAM.
      RETURN
C                             OPERATION PHASE -ENTERED ONCE PER RECORD.
C
C              A RECORD .R. (FROM THE SUMMARY TAPE HAS BEEN PUT INTO
C              THE VECTOR WORD. THE N TH WORD OF THE RECORD HAS BEEN
C              PUT INTO WORD(N).
C
C              IN OPERATION PHASE OF BLOC13 THE EXP OF WORD(L) IS
C              COMPUTED AND ADDED TO SUMEXP IF AND ONLY IF TEST NT
C              IS TRUE(WHEN APPLIED TO THE RECORD).
C
C              THE TRUTH VALUE OF TEST NT IS FOUND IN JTW(NT)
C
C
  200 NT=NT
C              NECESSATATED BY FORTRAN IDIOSYNCRACY
      IF(JTW(NT))202,201,202
  201 L=L
      SUMEXP=EXPF(WORD(L))+SUMEXP
C              RETURN TO EXCUTIVE PROGRAM-FOR NEXT RECORD




      RETURN
C                             FINAL PHASE -END OF SUMMARY TAPE HAS BEEN
C              REACHED - NO MORE RECORDS - DO FINAL COMPUTATIONS (IF ANY)
C              AND OUTPUT.
C
C              GET CURRENT PAGE NUMBER
  300 IPAGE=IPAGE+1
C              OUTPUT INFORMATION - ITB DENOTES MONITOR OUTPUT TAPE.
C              PRECEDE OUTPUT STATEMENTS BY CALL SCLEAR(ITB) TO INSURE
C              THAT CHANNEL CORRESPONDING TO ITB IS CLEARED.
      CALL SCLEAR(ITB)
      WRITE OUTPUT TAPE ITB,302,NT,IPAGE
      WRITE OUTPUT TAPE ITB,303,L,SUMEXP
  302 FORMAT(1H1,5X,19HEXAMPLE-BLOC13  NT= I3,54X,5HPAGE  I2)
  303 FORMAT(10X,24H  SUMMATION OF EXP(WORD( I4, 3H))=E10.3 )
      RETURN
      END
```



CERN Courier special issues on computing

September 1967



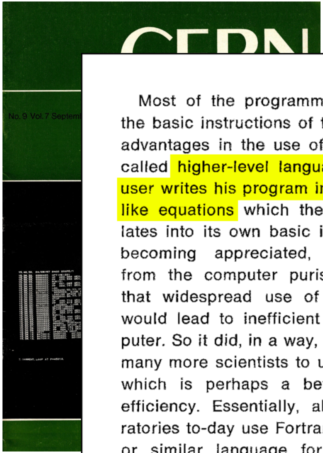
Fortran mentioned 3 times

March 1972

Fortran mentioned 18 times

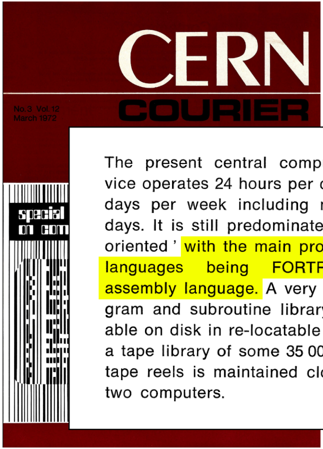
CERN Courier special issues on computing

September 1967



Most of the programming was done in the basic instructions of the computer, but advantages in the use of Autocode (a so-called higher-level language in which the user writes his program in text and algebra-like equations which the computer translates into its own basic instructions) were becoming appreciated, amidst protests from the computer purists who believed that widespread use of such languages would lead to inefficient use of the computer. So it did, in a way, but it also allowed many more scientists to use the computer, which is perhaps a better criterion for efficiency. Essentially, all research laboratories to-day use Fortran, Algol, Autocode or similar language for scientific work.

March 1972



The present central computing service operates 24 hours per day, seven days per week including most holidays. It is still predominately ' batch-oriented ' with the main programming languages being FORTRAN and assembly language. A very large program and subroutine library is available on disk in re-locatable form, and a tape library of some 35 000 labelled tape reels is maintained close to the two computers.

Part 1 conclusions: adoption of Fortran

NHEP adopted Fortran for data analysis in the first years after its release in 1956.

Part 1 conclusions: adoption of Fortran

NHEP adopted Fortran for data analysis in the first years after its release in 1956.

"Algebra-like equations" (i.e. **FOR**mula-**TRAN**slation) was an obvious benefit.

Part 1 conclusions: adoption of Fortran

NHEP adopted Fortran for data analysis in the first years after its release in 1956.

“Algebra-like equations” (i.e. **FOR**mula-**TRAN**slation) was an obvious benefit.

Also portability: assembly programs only work on a single model of computer.

Part 1 conclusions: adoption of Fortran

NHEP adopted Fortran for data analysis in the first years after its release in 1956.

"Algebra-like equations" (i.e. **FOR**mula-**TRAN**slation) was an obvious benefit.

Also portability: assembly programs only work on a single model of computer.

Why Fortran and not something else (e.g. ALGOL)?

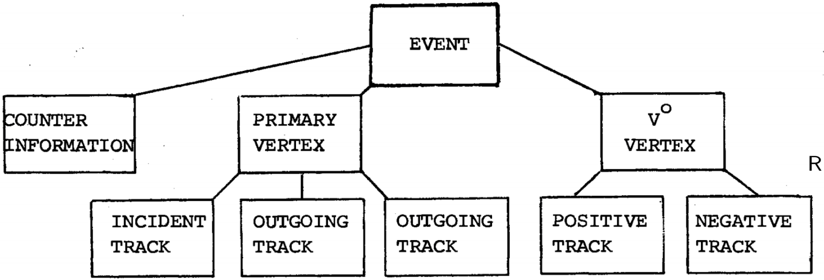
It was IBM's product, and most labs were buying IBM computers.

Part 2: C++

Fortran lacked an essential feature for NHEP

Example: High Energy Physics events are made up of vertices, every vertex has tracks associated to it. Also, to each event is associated a bank of information concerning electronic counter information to be used later. Assume the event to be a two-prong with an associated V^o.

The pictorial graph for this event information is then

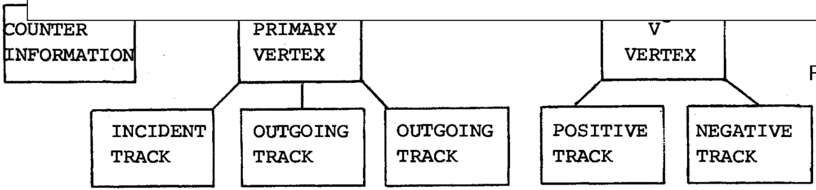


R.K. Böck, *Initiation to Hydra* (1974)

Fortran lacked an essential feature for NHEP

__Example:__ High Energy Physics events are made up of vertices, every vertex has tracks associated to it. Also, to each event is associated a bank of information concerning electronic counter information to be used later. Assume the event to be a two-prong with an associated V^{o}.

This wasn't a part of Fortran until 1991.
Physicists created *libraries* for tree-like data.



R.K. Böck, *Initiation to Hydra* (1974)

Fortran lacked an essential feature for NHEP

P. Lebrun and A. Kreymer, *High Level Language Memory Management on Parallel Architectures* (1989)

Years ago, the need for "pointer based" FORTRAN packages, such as HYDRA or BOS, later on ZBOOK [1], ZEBRA [2] or YBOS, became a necessity to efficiently manage the user heap space on a tight fixed size physical memory, in order to support large applications, such as code management systems or histogram packages.[1] Later on, these memory management systems were used not only to allocate dynamically space in a fixed size heap space, but also as a tool to organize and manage sensibly a complicate set of structures describing a detector, or an elementary particle collision. These packages are the essential building blocks for HEP data bases. They were designed to run mainly on single CPU systems and ignored entirely the existence of virtual memory available within FORTRAN through - system dependent ! - system calls.

Fortran lacked an essential feature for NHEP

inside a program. Logical relations between banks are expressed by including the address of one bank in the link-table of an other bank. For example, all tracks of a vertex-point are linked together by each track pointing to the next. Such a *data-structure* contains not only the numeric information but also logical information about the object it describes.

The program modularity is achieved by organizing the program into processors each having a well defined task. This task is *entirely* describable as a transformation applied to a data-structure in the dynamic store : some banks provide the input data to the processor and some contain the desired results. For a given application, a steering program is written to coordinate the operations of the processors needed. Any processor consists of at least one FORTRAN subroutine, its operation being invoked by transferring control to this subroutine. As a matter of internal organization, the processor may be divided up into the primary and several secondary subroutines. The programming of a processor has to observe certain conventions in order to be compatible with the HYDRA system. Precisely these conventions, which are the same through the whole program (indeed through all HYDRA programs) are responsible for the easy documentation and the good readability of the program.

The processors are supported by the HYDRA system. Its services are requested with CALL statements much like the services of the FORTRAN system which are part of the definition of the basic language. In this sense, the HYDRA system is an extension of the FORTRAN language to provide - primarily - dynamic memory management facilities. Some languages contain these facilities in their basic definition, but the HYDRA-FORTRAN combination has two important advantages — the execution speed is that of a normal FORTRAN program, with very little overhead for the HYDRA system, and FORTRAN is a commonly accepted language. Because of the need for machine independence (so that the same programs can be used on a variety of computers) the processors for the new bubble chamber program, as well as the HYDRA system packages, have been written in ANSI FORTRAN which is the internationally accepted minimum requirement expected from anybody's compiler.

The bubble chamber programs of the HYDRA form will come into operation in 1972. They should help to tear down the walls that have sometimes threatened, on the data handling side, to separate physicists from computer specialists, or bubble chamber groups from each other and from physicists using other techniques.

R.K. Böck and J. Zoll, *Central Computers in the Analysis*, CERN Courier No. 3, Vol. 12, March 1972

Beyond Fortran, data structures were a common language feature

COBOL (1959)

```
01 Point.
   05 x     pic 9(3).
   05 y     pic 9(3).
```

Simula (1962)

```
Class Point (x, y);
  Integer x, y;
  ! define attributes...
  ! define methods...
End of Point;
```

PL/I (1964)

```
define structure
  1 point,
     2 x integer,
     2 y integer;
```

ALGOL-68 (1968)

```
MODE POINT = STRUCT(
   INT x,
   INT y
);
```

Pascal (1970)

```
type Point = record
    x, y: integer;
end;
```

C (1972)

```
typedef struct Point {
  int x;
  int y;
} Point;
```

Beyond Fortran, data structures were a common language feature

Paul Kunz, *Physics Analysis Tools* (1991)

The C programming language is much better than FORTRAN for both data structures and configuration control. Shown in Figure 4 are some segments of C code that one might use in dealing with a track entity. Note the expressive power of the language in that access to variables is by full name. Also, the C language deals directly with the dynamic memory allocation of such structures since the memory allocation functions are part of its standard library. Finally, there's nothing lost in using a symbolic debugger because structures are part of the language, thus known to the existing debugger.

Clearly, C is much better at handling data structures then FORTRAN plus some additional package. Although many large collaborations have discussed abandoning FORTRAN, none has done so (yet). One reason they stayed with FORTRAN is reluctance to learn a new language, which is ironic since in each case they had to learn a big system to complement FORTRAN.

```
struct track {
     float px;
     float py;
     float pz;
};
...    ...    ...
struct track **mctrack;
...    ...    ...
px = mctrack[it]->px;
```

Figure 4. Segments of C code.

Beyond Fortran, data structures were a common language feature

Paul Kunz, *Physics Analysis Tools* (1991)

The C programming language is much better than FORTRAN for both data structures and configuration control. Shown in Figure 4 are some segments of C code that one might use in dealing with a track entity. Note the expressive power of the language in that access to variables is by full name. Also, the C language deals directly with the dynamic memory allocation of such structures since the memory allocation functions are part of its standard library. Finally, there's nothing lost in using a symbolic debugger because structures are part of the language, thus known to the existing debugger.

Clearly, C is much better at handling data structures then FORTRAN plus some additional package. Although many large collaborations have discussed abandoning FORTRAN, none has done so (yet). One reason they stayed with FORTRAN is reluctance to learn a new language, which is ironic since in each case they had to learn a big system to complement FORTRAN.

```
struct track {
     float px;
     float py;
     float pz;
};
...    ...    ...
struct track **mctrack;
...    ...    ...
px = mctrack[it]->px;
```

Figure 4. Segments of C code.

In the first 15 years of CHEP (1985–2000), similar suggestions were made for ALGOL, PL/I, Pascal, Ada, Eiffel, Objective C, Java, and of course C++.

Fortran did finally add data structures

Fortran-90 (1991)

```
type Point
    integer :: x
    integer :: y
end type Point
```

Fortran did finally add data structures

René Brun, *Technologies, Collaborations and Languages: 20 Years of HEP Computing* (2012)

Fortran-90 (1991)

```
type Point
    integer :: x
    integer :: y
end type Point
```

One major stumbling block in the move to FORTRAN 90 was the question of Input/Output. With ZEBRA, we had a simple way to describe data structures (banks) built out of basic types (typically integers and floats). Because FORTRAN 90 supported derived data types, it was theoretically possible to implement the most complex data structures that we used to model with ZEBRA. In particular ZEBRA was able to write and read these data structures from machine independent files.

> The need for introspection to deal with derived data types doomed the efforts to move to FORTRAN 90. It was going to be as hard (if not more) with C++, but we did not know this at the time.

Using FORTRAN 90, it appeared pretty hard to make a general implementation equivalent to ZEBRA without parsing the data type description in the FORTRAN 90 modules. In fact, we encountered the same problem later with C++, but we naively ignored at that time how much work it was to implement a data dictionary or reflection system describing at run time the type of objects. Mike Metcalf was aware of the problem and we reported this to a special session of the FORTRAN committee at CERN in 1992. As most members in the committee had no experience with this problem and thought that this was a database problem and not a language problem, the enhancements that we were expecting in the language did not happen.

So actually, NHEP needs:

1. rich data structures in the programming language,
2. serialization of those structures to/from disk,
3. read compatibility for old data versions (schema evolution),
4. mapping between persistent data and language's structures (which can be implemented with type-introspection).

So actually, NHEP needs:

1. rich data structures in the programming language,
2. serialization of those structures to/from disk,
3. read compatibility for old data versions (schema evolution),
4. mapping between persistent data and language's structures (which can be implemented with type-introspection).

HYDRA/ZEBRA/etc. approached these as *a single* problem.

So actually, NHEP needs:

1. rich data structures in the programming language,
2. serialization of those structures to/from disk,
3. read compatibility for old data versions (schema evolution),
4. mapping between persistent data and language's structures (which can be implemented with type-introspection).

HYDRA/ZEBRA/etc. approached these as *a single* problem.

In Java, (1–2) is a language problem, (3–4) is for databases.

So actually, NHEP needs:

1. rich data structures in the programming language,
2. serialization of those structures to/from disk,
3. read compatibility for old data versions (schema evolution),
4. mapping between persistent data and language's structures (which can be implemented with type-introspection).

HYDRA/ZEBRA/etc. approached these as *a single* problem.

In Java, (1–2) is a language problem, (3–4) is for databases.

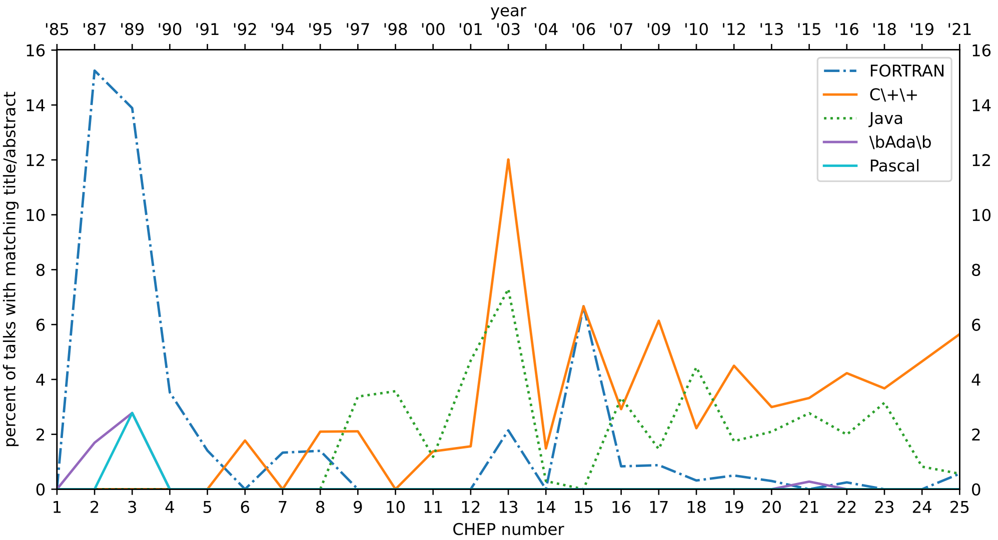
Object databases focus on (4). . .

So it wasn't just a matter of adding another language to the mix.

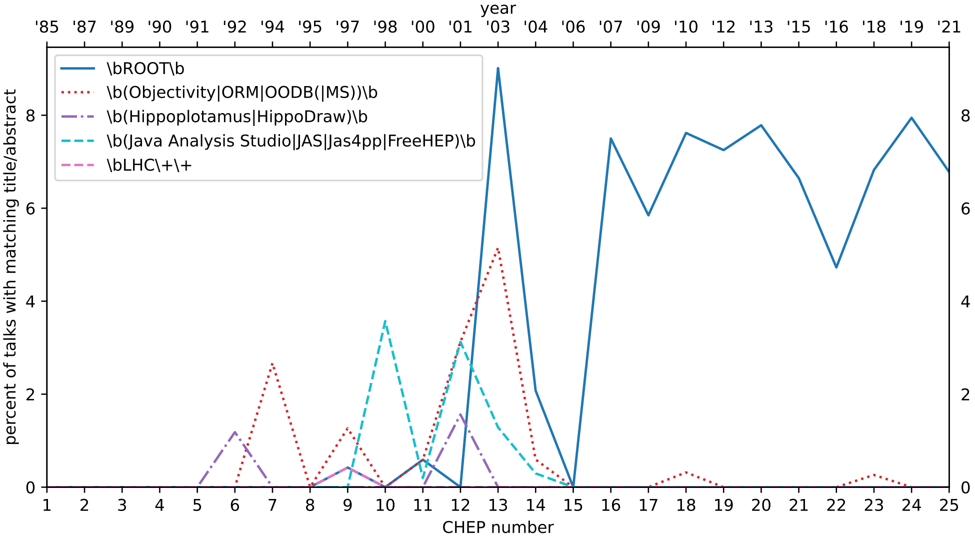
The whole infrastructure, including I/O, had to change.

You only want to do that once!

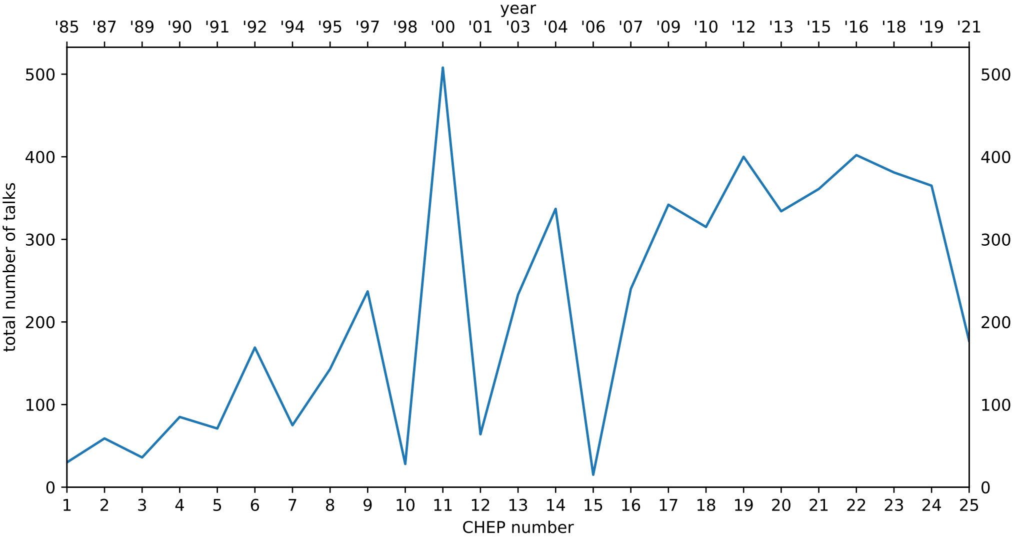
Programming languages in CHEP title/abstract regex matches



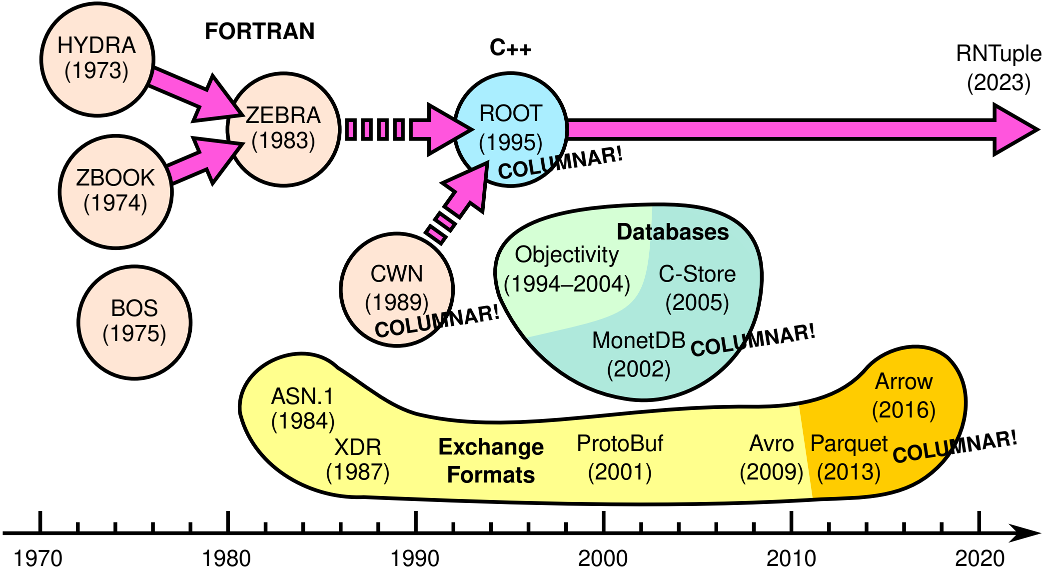
Data infrastructure in CHEP title/abstract regex matches



Total number of talks (denominator): dips are small CHEPs



ROOT I/O is custom, but was more advanced than alternatives



Some advances had to be rediscovered outside NHEP

Dremel: Interactive Analysis of Web-Scale Datasets (Google, 2010)

storage and reduce CPU cost due to cheaper compression. Column stores have been adopted for analyzing relational data [1] but to the best of our knowledge have not been extended to nested data models. The columnar storage format that we present is supported by many data processing tools at Google, including MR, Sawzall [20], and FlumeJava [7].

In this paper we make the following contributions:

- We describe a novel columnar storage format for nested data. We present algorithms for dissecting nested records into columns and reassembling them (Section 4).

Columnar, nested data was a ROOT feature 15 years earlier.

Part 2 conclusions: adoption of C++

NHEP adoption of C++ (or similar) was held back by the fact that we had unique infrastructure.

Part 2 conclusions: adoption of C++

NHEP adoption of C++ (or similar) was held back by the fact that we had unique infrastructure.

It's important to have data structures in the language, but also on disk.

Part 2 conclusions: adoption of C++

NHEP adoption of C++ (or similar) was held back by the fact that we had unique infrastructure.

It's important to have data structures in the language, but also on disk.

Our solutions were both *more advanced* and *less modular* than others.

Part 2 conclusions: adoption of C++

NHEP adoption of C++ (or similar) was held back by the fact that we had unique infrastructure.

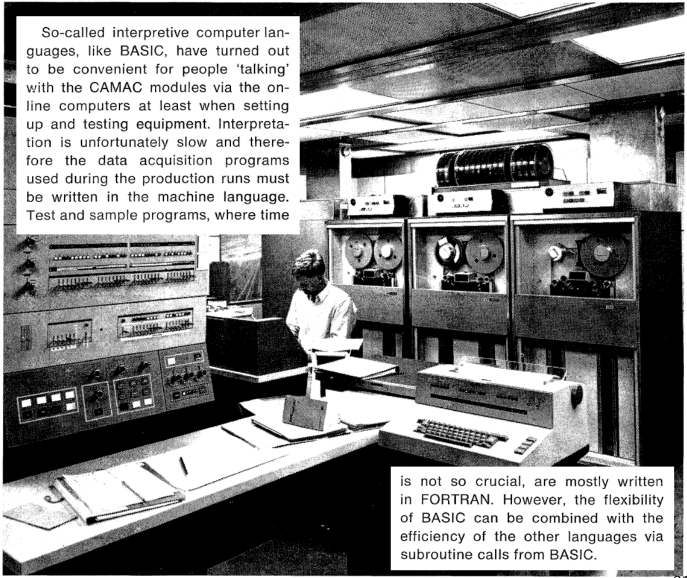
It's important to have data structures in the language, but also on disk.

Our solutions were both *more advanced* and *less modular* than others.

Many options considered in late 1990's; C++ with ROOT I/O became dominant.

Part 3: Python

End-stage data analysis benefits from interactivity



So-called interpretive computer languages, like BASIC, have turned out to be convenient for people 'talking' with the CAMAC modules via the on-line computers at least when setting up and testing equipment. Interpretation is unfortunately slow and therefore the data acquisition programs used during the production runs must be written in the machine language. Test and sample programs, where time is not so crucial, are mostly written in FORTRAN. However, the flexibility of BASIC can be combined with the efficiency of the other languages via subroutine calls from BASIC.

NHEP has a history of custom solutions for interactivity: SPEAKEASY, Minuit, PAW, KUIP, CINT, Cling. . .

NHEP has a history of custom solutions for interactivity: SPEAKEASY, Minuit, PAW, KUIP, CINT, Cling. . .

But the number of industry solutions is also vast.

Interactive mode languages [edit]

Interactive mode languages act as a kind of shell: expressions or statements can be entered one at a time, and the result of their evaluation is seen immediately. The interactive mode is also termed a read–eval–print loop (REPL).

- APL
- BASIC (some dialects)
- Clojure
- Common Lisp
- Dart (with Observatory or Dartium's developer tools)
- ECMAScript
  - ActionScript
  - ECMAScript for XML
  - JavaScript
  - JScript
  - Source
- Erlang
- Elixir (with iex)
- F#
- Fril
- GAUSS
- Groovy
- Haskell (with the GHCi or Hugs interpreter)
- IDL
- J
- Java (since version 9)
- Julia
- Lua
- MUMPS (an ANSI standard general purpose language)
- Maple
- Mathematica (Wolfram language)
- MATLAB
- ML
- OCaml
- Perl
- PHP
- Pike
- PostScript
- Prolog
- Python
- PROSE
- R
- REBOL
- Rexx
- Ruby (with IRB)
- Scala
- Scheme
- Smalltalk (anywhere in a Smalltalk environment)
- S-Lang (with the S-Lang shell, slsh)
- Speakeasy
- Swift
- Tcl (with the Tcl shell, tclsh)
- Unix shell
- Windows PowerShell (.NET-based CLI)
- Visual FoxPro

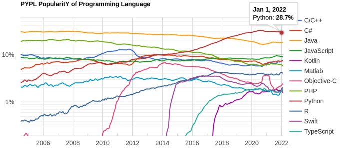
In recent years, the industry has consolidated on Python

Python is currently leading every "most popular programming language" index.

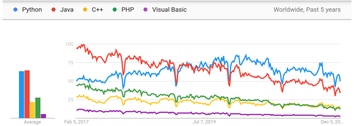
Tiobe

Jan 2022	Jan 2021	Change	Programming Language	Ratings	Change
1	3	^	Python	13.58%	+1.86%
2	1	v	C	12.44%	-4.94%
3	2	v	Java	10.66%	-1.30%
4	4		C++	8.29%	+0.73%
5	5		C#	5.68%	+1.73%
6	6		Visual Basic	4.74%	+0.90%
7	7		JavaScript	2.09%	-0.11%
8	11	^	Assembly language	1.85%	+0.21%

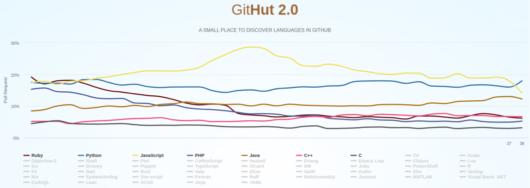
PYPL



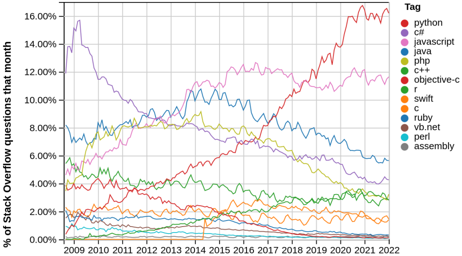
Google Trends



GitHut

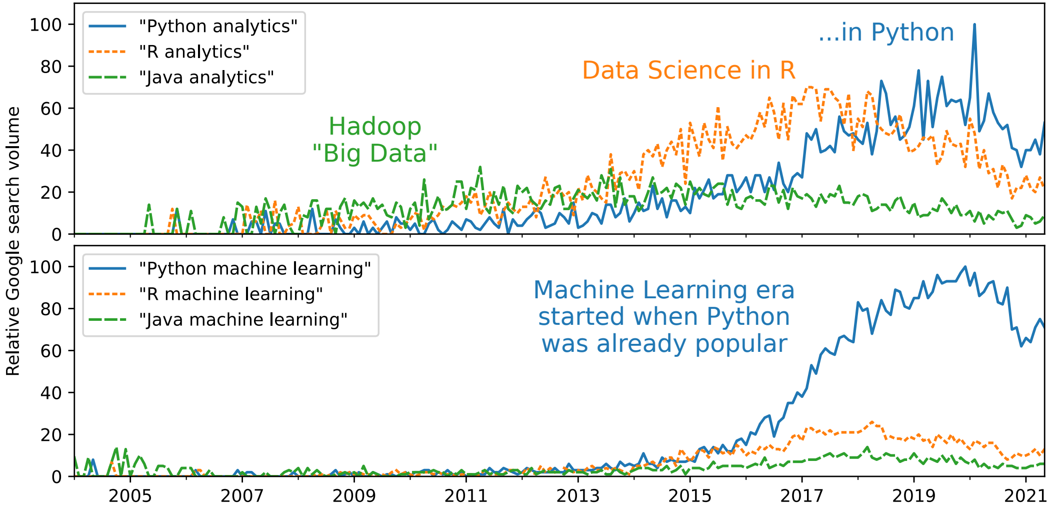


StackOverflow

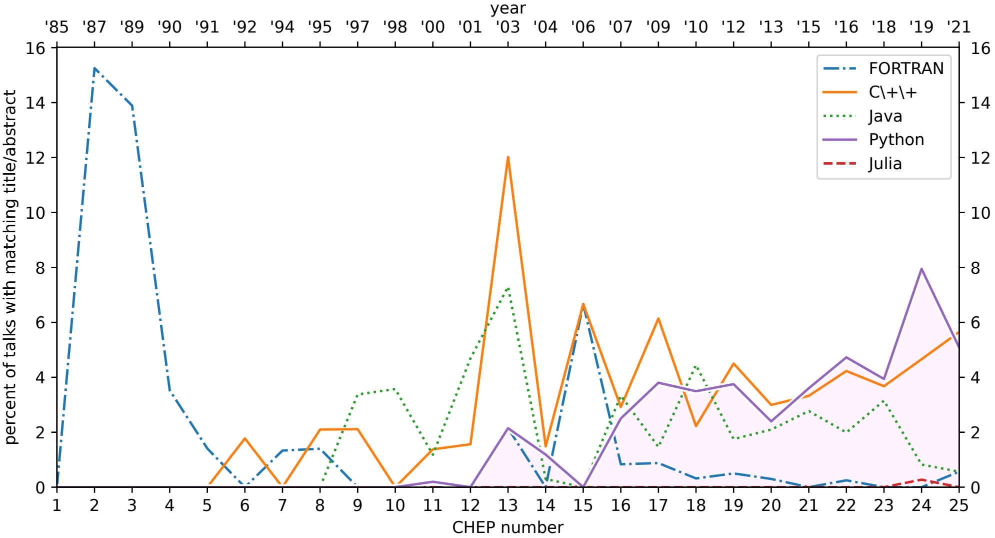


Including data analytics and especially machine learning

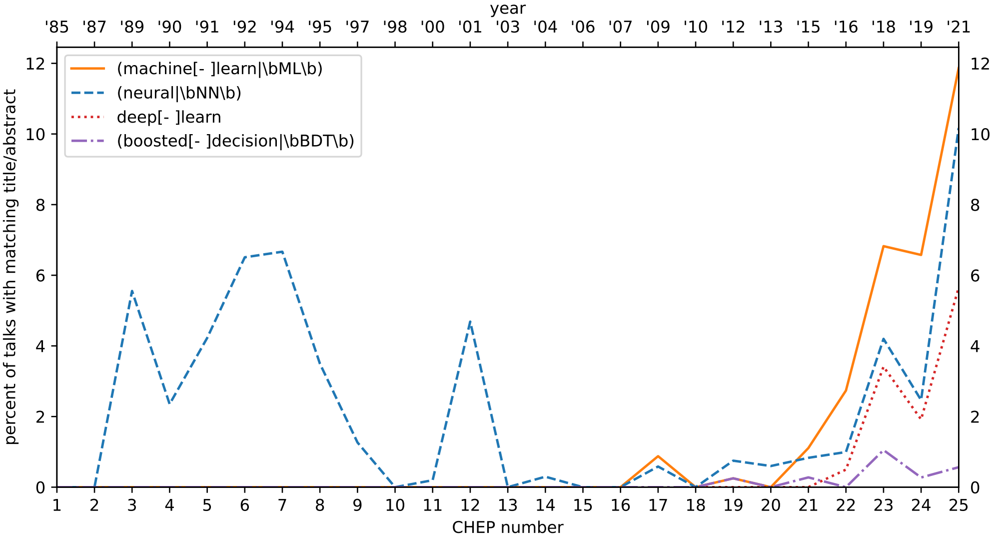
Correlated words in Google searches (Google Trends).



Python use has also been rising—steadily—in NHEP



Since *before* the return of machine learning



Emerging Standard ? Python as "Software Glue"

- **Clear trend towards Python**
  - Used by: ATLAS (Athena),CMS, D0, LHCb (Gaudi), SND,...
  - Used by: Lizard/Anaphe, HippoDraw, JAS (Jython)...
  - Architecturally, scripting is "just another service"
  - ROOT is the exception to the "Python rule"
    - CINT interpreter plays a central role
    - Developers and users seem happy
- **Python is popular with developers...**
  - Rapid prototyping; gluing together code
  - (Almost) auto-generation of wrappers (SWIG)
- **...but acceptance by users not yet proven**
  - Another language to learn, syntax,...

"Summary of Track 2: Data Analysis and Visualisa
Lucas Taylor, Northeastern U. CHEP 01, Beijing, 3-7 S

Note: PyROOT introduced in 2004 (v4.00/04).

Early applications: gluing compiled modules together

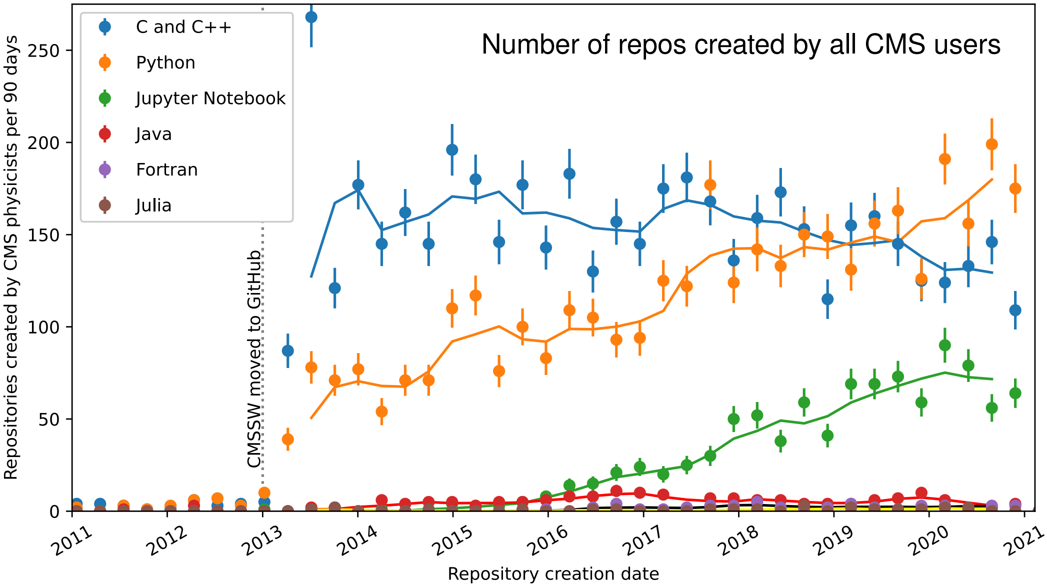
Stephan Lammel, *Computing models of major HEP experiments: DØ and CDF*, 1997

> DØ has made the decision to move all large software projects to C++. Their framework approach has a set of modules that execute sequentially, each having a specific task. The glue that holds the individual software packages together will be an interpreted script system. The main task of this framework is to "guide" data between the various modules/packages ... prototype framework based on the Python scripting language has been developed and is ready for use.

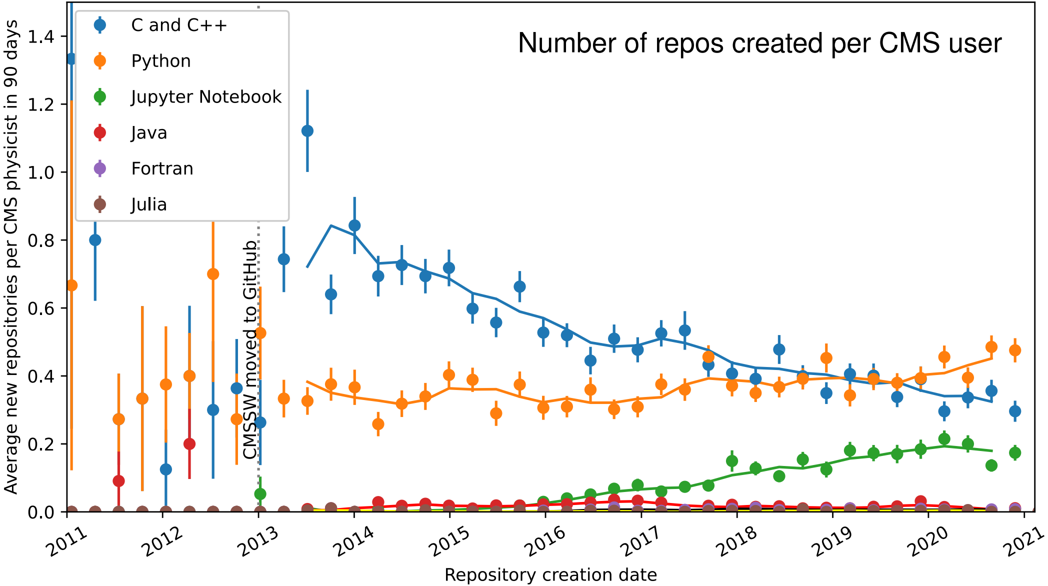
Jeff Templon, *Python as an Integration Language*, SPAG-1998-02, 1998

> Most of the code would be written in C or C++, but the integration would be done through Python. This enables the uninitiated to make simple modifications to the analysis which were perhaps not thought of by the authors; all the neophyte needs to know is how the interfaces work. On the other hand, it will force the code authors to make the analysis subsystems independent of each other (one of the big problems with the current code), and will encourage rigorous testing of subunits.

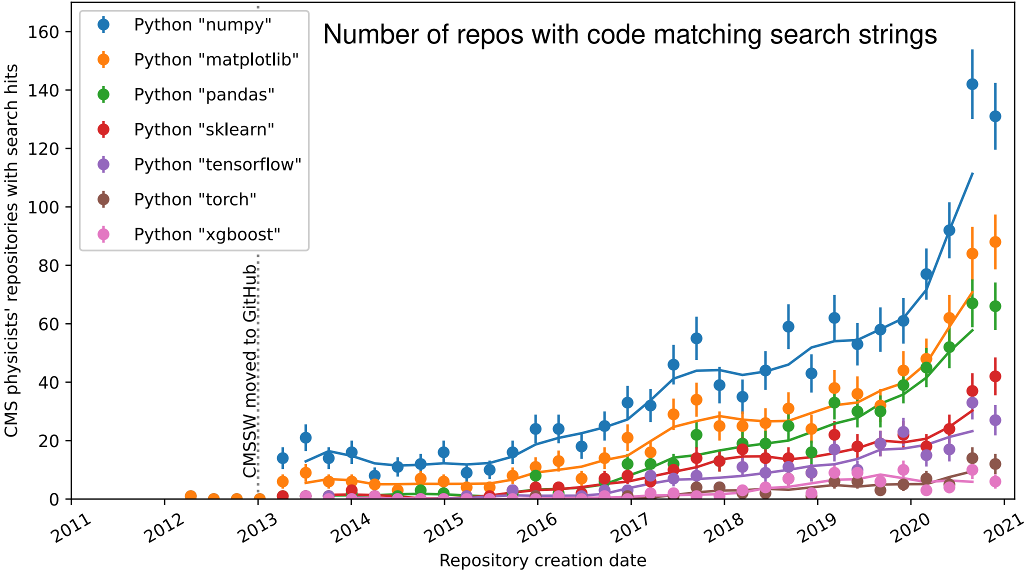
Trends in code written by CMS users (in GitHub)



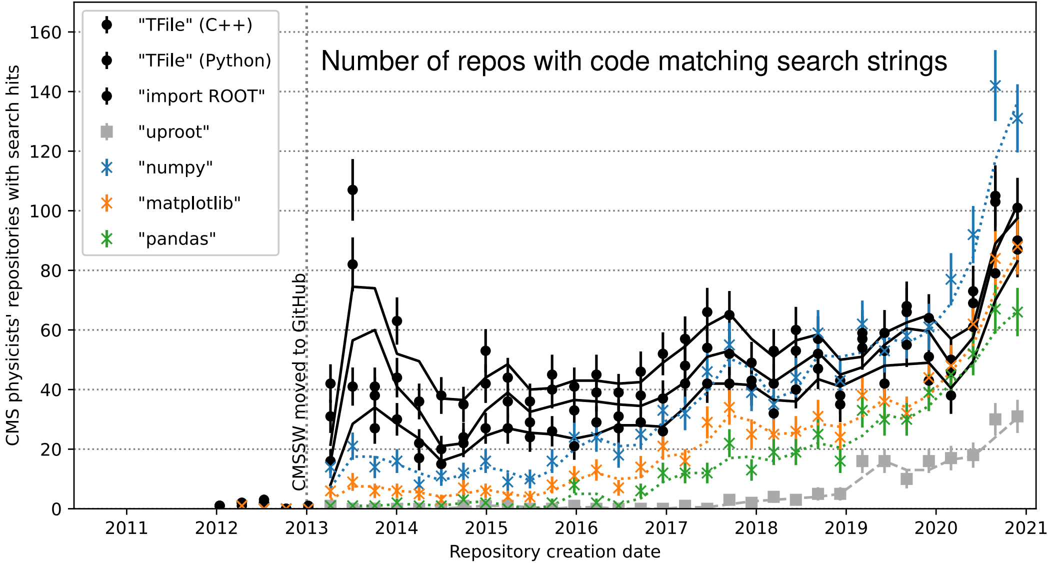
Trends in code written by CMS users (in GitHub)



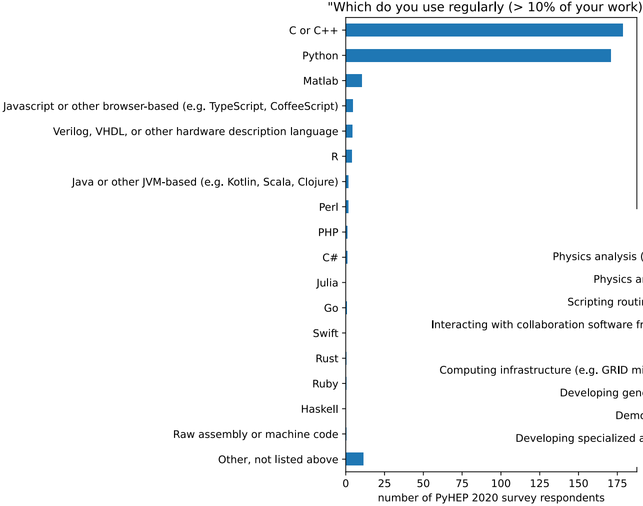
Trends in code written by CMS users (in GitHub)

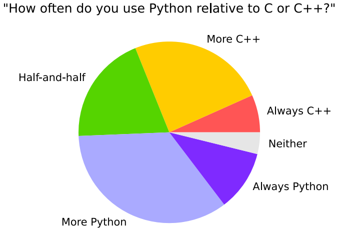


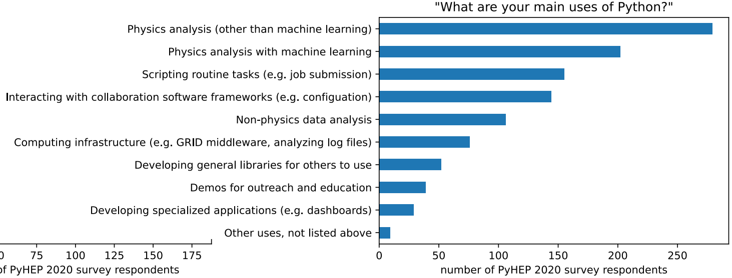
Trends in code written by CMS users (in GitHub)



Survey responses from PyHEP 2020 attendees ($N = 406$)







Part 3 conclusions: adoption of Python

NHEP adoption of Python started long before machine learning and columnar analysis trends.

Part 3 conclusions: adoption of Python

NHEP adoption of Python started long before machine learning and columnar analysis trends.

Interactive/fluid programming has always been a need, and has traditionally been met by a variety of alternatives.

Part 3 conclusions: adoption of Python

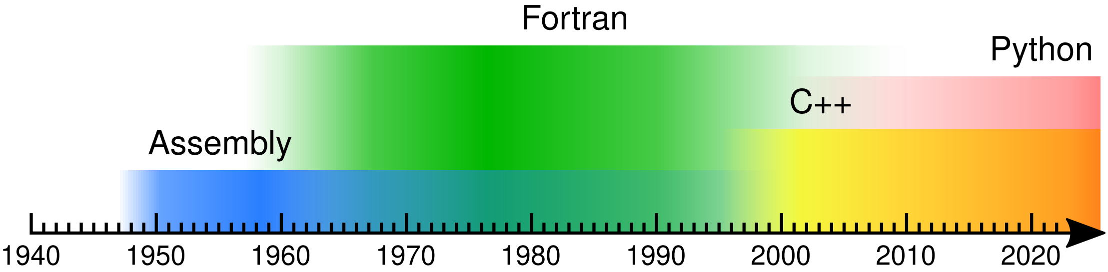
NHEP adoption of Python started long before machine learning and columnar analysis trends.

Interactive/fluid programming has always been a need, and has traditionally been met by a variety of alternatives.

What's new is the consolidation on one language, Python, and an increase in how much analysis logic can be "driven" from the interactive language.

Concluding conclusions

Concluding conclusions



Adoption of

Fortran: immediate; for syntax and portability; no infrastructure to replace

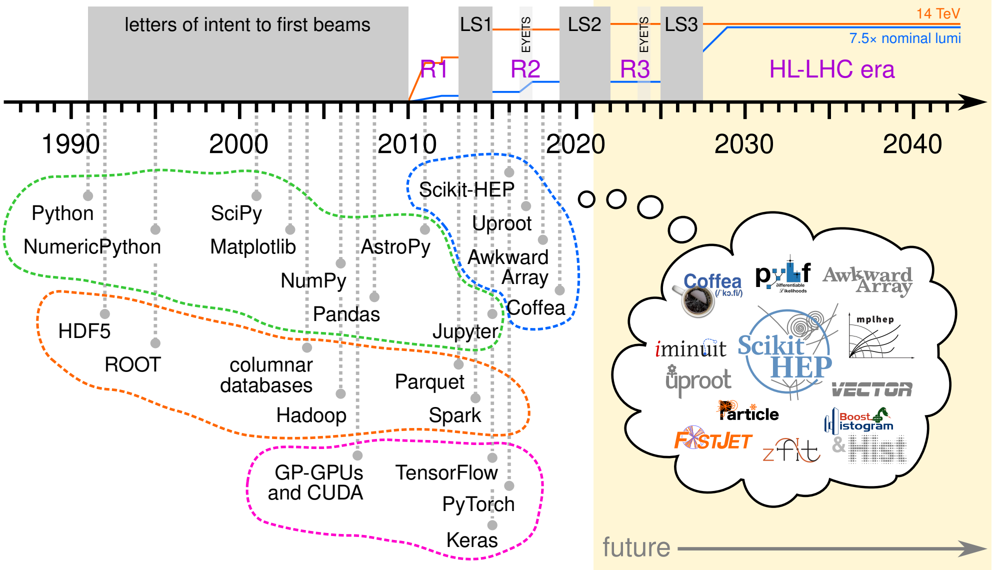
C++: long overdue; for data structures; replaced infrastructure in a burst

Python: slowly overtook its alternatives; for interactivity; different niche

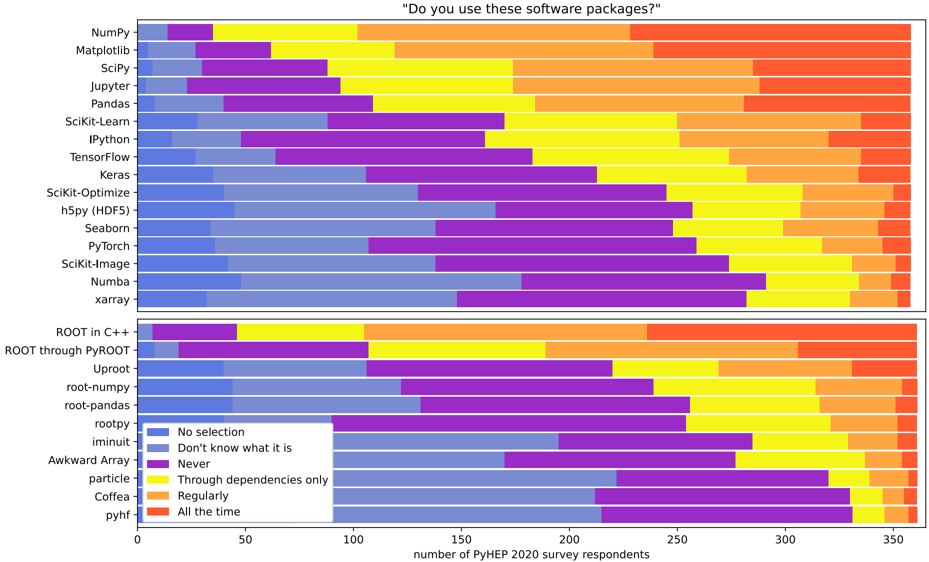
Julia? slowly mix in among the Python and C++ until it's all that's left?

Backup

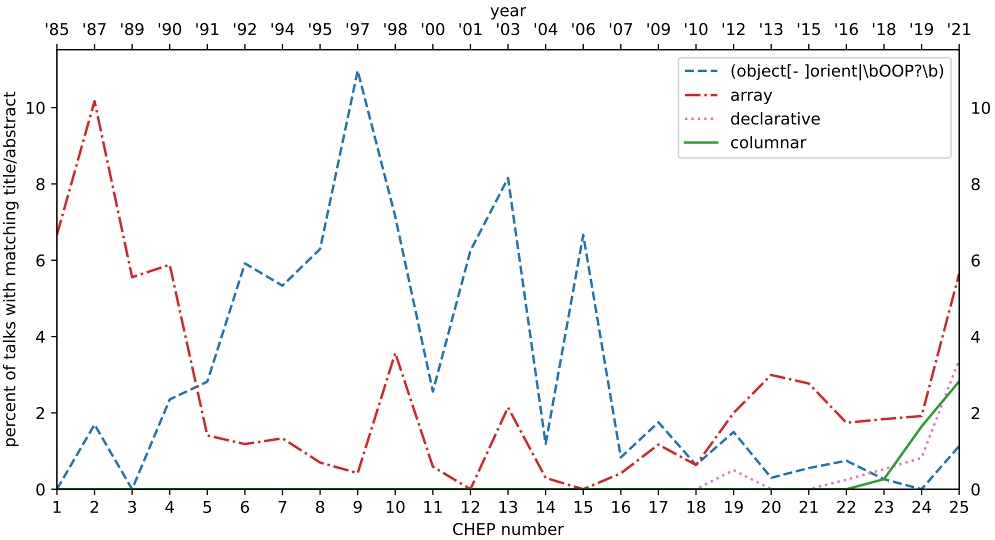
Python ecosystem has had as much time to evolve as the LHC



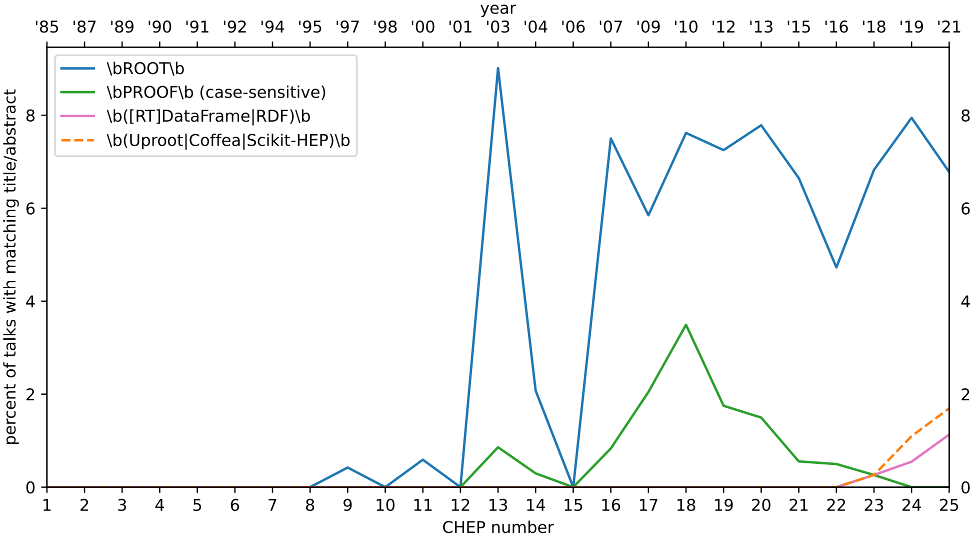
Widespread familiarity with data science tools (PyHEP survey)



More CHEP history



More CHEP history



More CHEP history