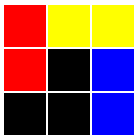
Narzędzia do analizy dużych zestawów danych

Krzysztof Miernik



Zakład Fizyki Jądrowej
Wydział Fizyki

Seminaria ZFJ

Według tytułów z archiwum (2016-2020)

Seminaria ZFJ

Według tytułów z archiwum (2016-2020)

- Eksperymentalne: 37
  ”Badanie własności neutrono-nadmiarowych nuklidów w okolicy podwójnie-magicznego jądra 132Sn”
- Teoretyczne: 27
  ”Reakcje nadciekłych jąder atomowych w świetle teorii funkcjonału gęstości”

Seminaria ZFJ

Według tytułów z archiwum (2016-2020)

- Eksperymentalne: 37
  ”Badanie własności neutrono-nadmiarowych nuklidów w okolicy podwójnie-magicznego jądra 132Sn”
- Teoretyczne: 27
  ”Reakcje nadciekłych jąder atomowych w świetle teorii funkcjonału gęstości”
- Detektory, układy: 19
  ”PARIS calorimeter - idea, status and first experiments”

Seminaria ZFJ

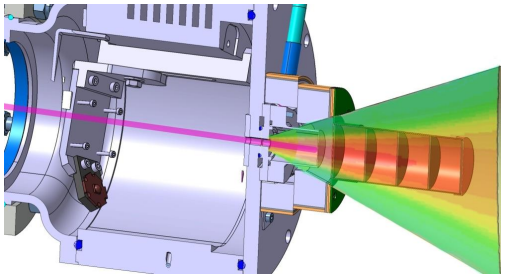
Według tytułów z archiwum (2016-2020)

- Eksperymentalne: 37
  ”Badanie własności neutrono-nadmiarowych nuklidów w okolicy podwójnie-magicznego jądra 132Sn”
- Teoretyczne: 27
  ”Reakcje nadciekłych jąder atomowych w świetle teorii funkcjonału gęstości”
- Detektory, układy: 19
  ”PARIS calorimeter - idea, status and first experiments”
- Inne: 8
  ”Polska energetyka jądrowa - fakty i mity”
- Komputery, programowanie: 1
  ”Uczenie maszynowe w fizyce subatomowej”

Eksperyment ν-Ball w ALTO

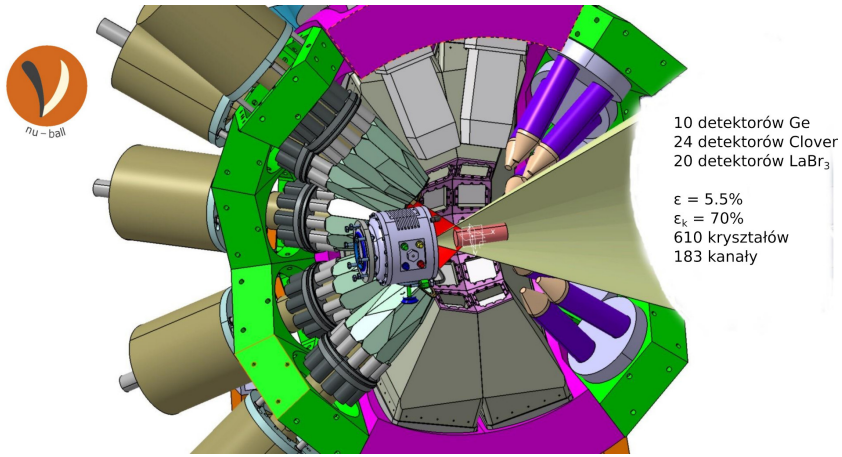
- Źródło neutronów LICORNE

  Wiązka ^{7}Li 200 nA Tarcza ^{2}H Strumień 10^{8} n/cm^{2}/s



  Tarcza ^{238}U lub ^{232}Th

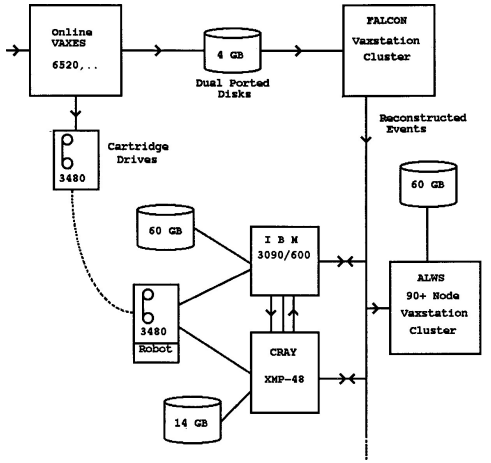
- Układ ν-Ball



- Pulsowana wiązka neutronów (400 ns)
- 4 tygodnie pomiarów z ^{238}U i 3 tygodnie z ^{232}Th
- beztriggerowy system zbierania danych
- 130 i 100 TB danych w formacie ”raw” $\rightarrow$ ok. 8-10 TB danych po utworzeniu zdarzeń

Rola komputerów

- W 1990-tych eksperyment ALEPH produkował TB danych (na eksperyment)

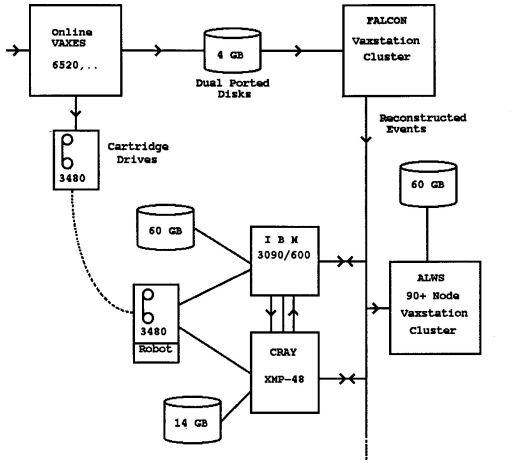


M.J. Corden et al. Comp. in Phys. 6 (1992)
IBM 3090-600: 6 rdzeni, 150 MIPS, 512 MB RAM

Cray X-MP: 4 rdzenie, 800 MFLOPS

Rola komputerów

- W 1990-tych eksperyment ALEPH produkował TB danych (na eksperyment)



M.J. Corden et al. Comp. in Phys. 6 (1992)
IBM 3090-600: 6 rdzeni, 150 MIPS, 512 MB RAM

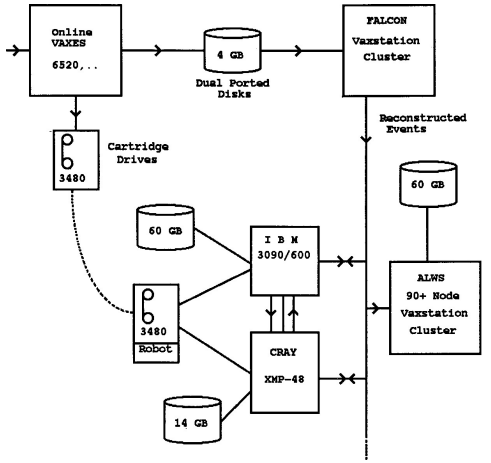
Cray X-MP: 4 rdzenie, 800 MFLOPS

- Dzisiaj ATLAS produkuje GB / s i PB na eksperyment
- W tym czasie procesor, dysk, pamięć w typowym komputerze również wzrosły 1000 razy

4 rdzenie, 125 MIPS, 100 GFLOPS, 8 GB RAM

Rola komputerów

- W 1990-tych eksperyment ALEPH produkował TB danych (na eksperyment)

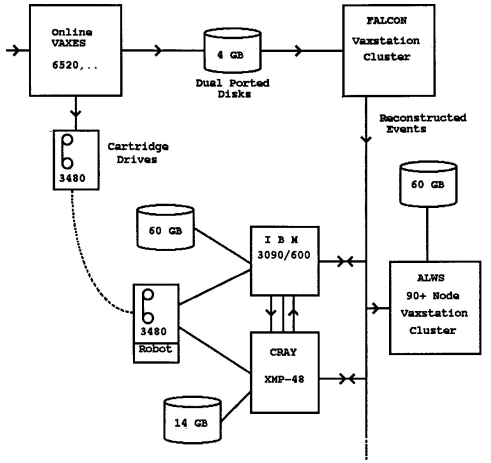


M.J. Corden et al. Comp. in Phys. 6 (1992)
IBM 3090-600: 6 rdzeni, 150 MIPS, 512 MB RAM
Cray X-MP: 4 rdzenie, 800 MFLOPS

- Dzisiaj ATLAS produkuje GB / s i PB na eksperyment
- W tym czasie procesor, dysk, pamięć w typowym komputerze również wzrosły 1000 razy
  4 rdzenie, 125 MIPS, 100 GFLOPS, 8 GB RAM
- Eksperymenty w fizyce niskich energii także zaczęły rosnąć: więcej detektorów, kanałów,

Rola komputerów

- W 1990-tych eksperyment ALEPH produkował TB danych (na eksperyment)

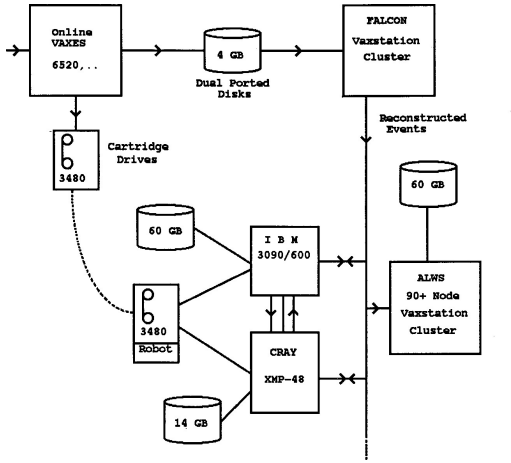


M.J. Corden et al. Comp. in Phys. 6 (1992)
IBM 3090-600: 6 rdzeni, 150 MIPS, 512 MB RAM
Cray X-MP: 4 rdzenie, 800 MFLOPS

- Dzisiaj ATLAS produkuje GB / s i PB na eksperyment
- W tym czasie procesor, dysk, pamięć w typowym komputerze również wzrosły 1000 razy
  4 rdzenie, 125 MIPS, 100 GFLOPS, 8 GB RAM
- Eksperymenty w fizyce niskich energii także zaczęły rosnąć: więcej detektorów, kanałów,
- Największą rewulocją jest elektronika cyfrowa - tryb beztriggerowy, przebiegi sygnałów

Rola komputerów

- W 1990-tych eksperyment ALEPH produkował TB danych (na eksperyment)

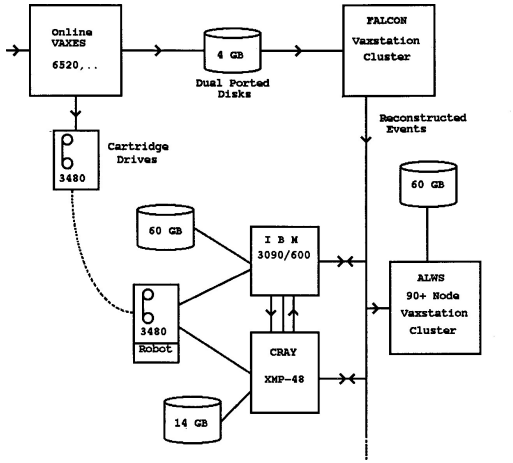


M.J. Corden et al. Comp. in Phys. 6 (1992)
IBM 3090-600: 6 rdzeni, 150 MIPS, 512 MB RAM
Cray X-MP: 4 rdzenie, 800 MFLOPS

- Dzisiaj ATLAS produkuje GB / s i PB na eksperyment
- W tym czasie procesor, dysk, pamięć w typowym komputerze również wzrosły 1000 razy
  4 rdzenie, 125 MIPS, 100 GFLOPS, 8 GB RAM
- Eksperymenty w fizyce niskich energii także zaczęły rosnąć: więcej detektorów, kanałów,
- Największą rewulocją jest elektronika cyfrowa - tryb beztriggerowy, przebiegi sygnałów
- Mamy do dyspozycji ”centrum obliczeniowe” równe temu, które było uznawane za szczytowe osiągnięcia w fizyce 30 lat temu

Rola komputerów

- W 1990-tych eksperyment ALEPH produkował TB danych (na eksperyment)

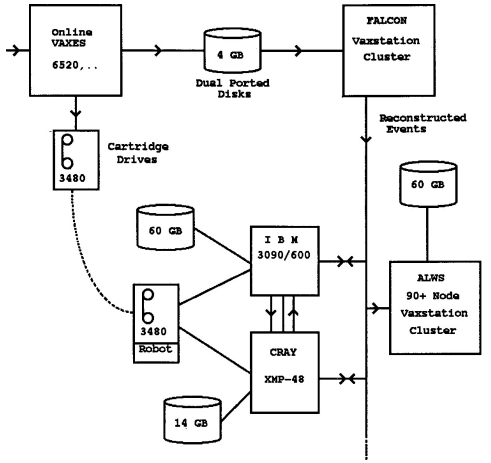


M.J. Corden et al. Comp. in Phys. 6 (1992)
IBM 3090-600: 6 rdzeni, 150 MIPS, 512 MB RAM
Cray X-MP: 4 rdzenie, 800 MFLOPS

- Dzisiaj ATLAS produkuje GB / s i PB na eksperyment
- W tym czasie procesor, dysk, pamięć w typowym komputerze również wzrosły 1000 razy
  4 rdzenie, 125 MIPS, 100 GFLOPS, 8 GB RAM
- Eksperymenty w fizyce niskich energii także zaczęły rosnąć: więcej detektorów, kanałów,
- Największą rewulocją jest elektronika cyfrowa - tryb beztriggerowy, przebiegi sygnałów
- Mamy do dyspozycji ”centrum obliczeniowe” równe temu, które było uznawane za szczytowe osiągnięcia w fizyce 30 lat temu
- Ale zwykle robimy to sami lub w niewielkich grupach . . .

Rola komputerów

- W 1990-tych eksperyment ALEPH produkował TB danych (na eksperyment)



M.J. Corden et al. Comp. in Phys. 6 (1992)
IBM 3090-600: 6 rdzeni, 150 MIPS, 512 MB RAM
Cray X-MP: 4 rdzenie, 800 MFLOPS

- Dzisiaj ATLAS produkuje GB / s i PB na eksperyment
- W tym czasie procesor, dysk, pamięć w typowym komputerze również wzrosły 1000 razy
  4 rdzenie, 125 MIPS, 100 GFLOPS, 8 GB RAM
- Eksperymenty w fizyce niskich energii także zaczęły rosnąć: więcej detektorów, kanałów,
- Największą rewulocją jest elektronika cyfrowa - tryb beztriggerowy, przebiegi sygnałów
- Mamy do dyspozycji ”centrum obliczeniowe” równe temu, które było uznawane za szczytowe osiągnięcia w fizyce 30 lat temu
- Ale zwykle robimy to sami lub w niewielkich grupach . . .
- Analiza danych często staje się ”wąskim gardłem” w eksperymentach. Czy potrafimy poradzić sobie z nadmiarem danych i wykorzystać możliwości komputerów?

Rozwój komputerów


IBM 1620: 100 kHz


Z3: 10 Hz

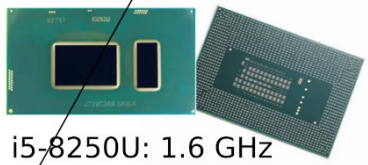

Odra 1003: 250 kHz


ZX Spectrum: 3.5 MHz


PC-AT: 8 MHz

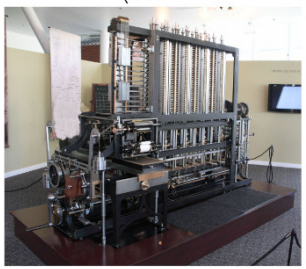
AMD K6: 166 MHz

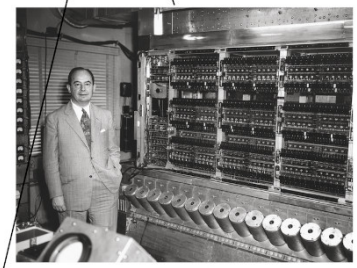

Intel Atom: 1.7 GHz

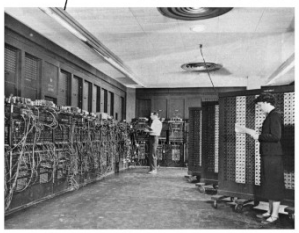

i5-8250U: 1.6 GHz

1822 1837 1943 1945 1949 1959 1963 1965 1971 1982 1984 1985 1992 1997 2003 2009

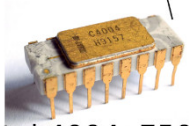
1940 1950 1960 1970 1980 1990 2000 2010 2020


Maszyna różnicowa


EDVAC: 5 kHz


ENIAC: 5 kHz


PDP8e: 667 kHz

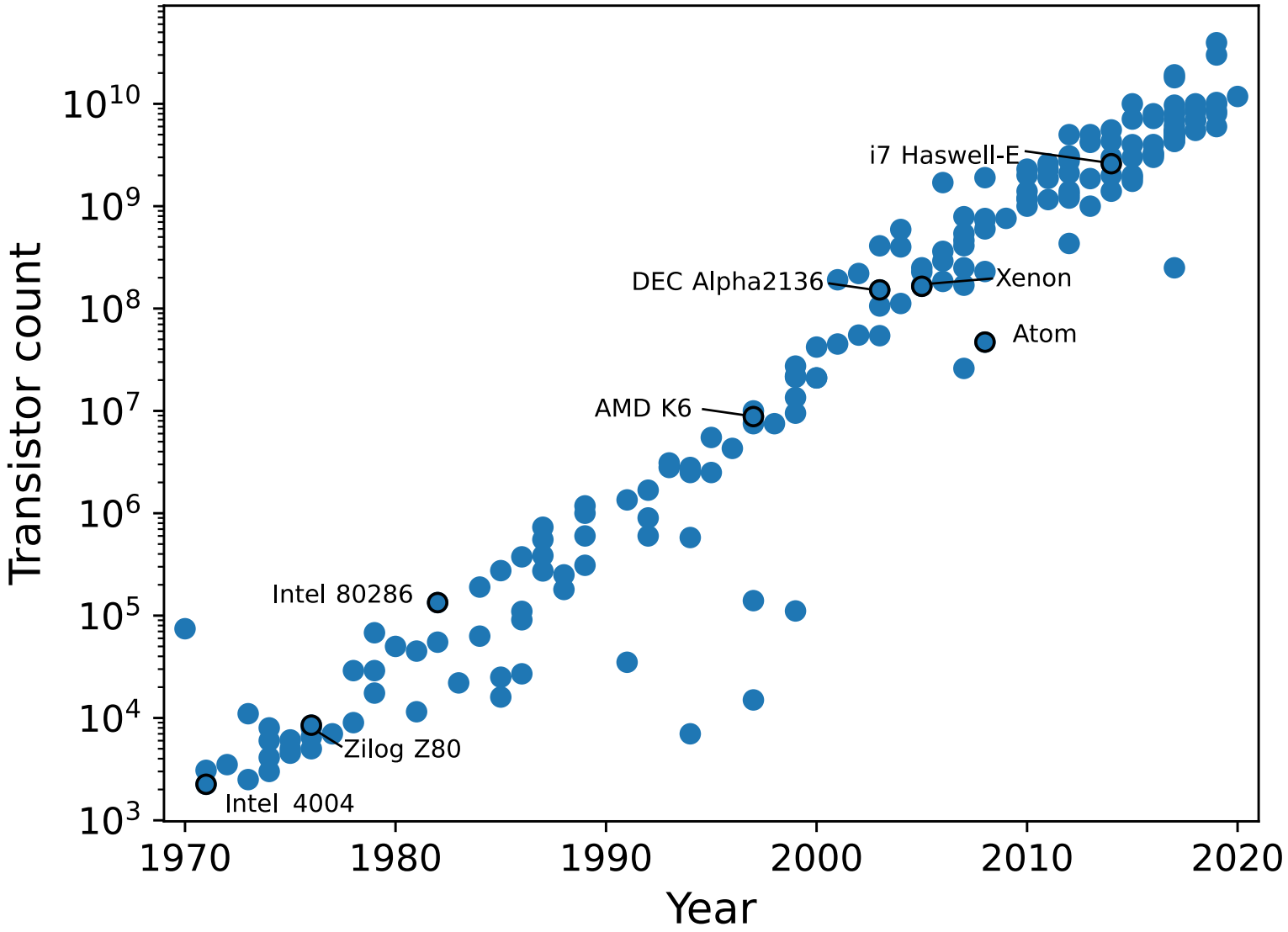

Intel 4004: 750 kHz

i386 DX: 16 MHz

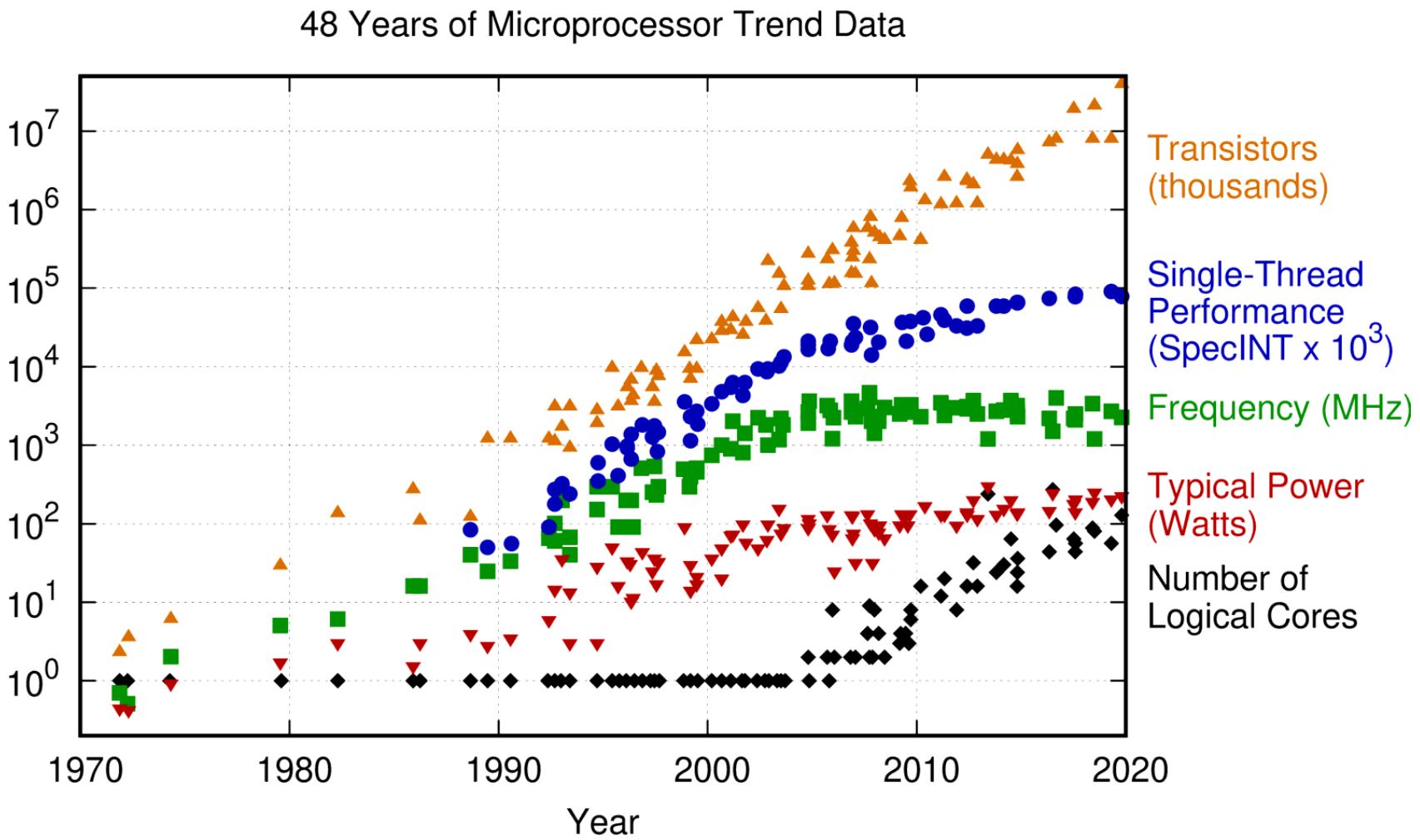

DEC Alpha: 150 MHz

AMD Opteron: 1.4 GHz

Liczba tranzystorów



Trendy w rozwoju



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

Narzędzia analizy danych

Lista życzeń

Narzędzia analizy danych

Lista życzeń

- Względnie proste do opanowania: używające popularnego języka programowania lub zbliżonej składni

Narzędzia analizy danych

Lista życzeń

- Względnie proste do opanowania: używające popularnego języka programowania lub zbliżonej składni
- Otwarte źródła

Narzędzia analizy danych

Lista życzeń

- Względnie proste do opanowania: używające popularnego języka programowania lub zbliżonej składni
- Otwarte źródła
- Często procedurę analizy opracowujemy podczas analizy samej w sobie (nie znamy wszystkich założeń a priori) i jest to proces iteracyjny

Narzędzia analizy danych

Lista życzeń

- Względnie proste do opanowania: używające popularnego języka programowania lub zbliżonej składni
- Otwarte źródła
- Często procedurę analizy opracowujemy podczas analizy samej w sobie (nie znamy wszystkich założeń a priori) i jest to proces iteracyjny $\rightarrow$ język dynamiczny

Narzędzia analizy danych

Lista życzeń

- Względnie proste do opanowania: używające popularnego języka programowania lub zbliżonej składni
- Otwarte źródła
- Często procedurę analizy opracowujemy podczas analizy samej w sobie (nie znamy wszystkich założeń a priori) i jest to proces iteracyjny $\rightarrow$ język dynamiczny
- Mile widziany język ogólnego przeznaczenia

Narzędzia analizy danych

Lista życzeń

- Względnie proste do opanowania: używające popularnego języka programowania lub zbliżonej składni
- Otwarte źródła
- Często procedurę analizy opracowujemy podczas analizy samej w sobie (nie znamy wszystkich założeń a priori) i jest to proces iteracyjny $\rightarrow$ język dynamiczny
- Mile widziany język ogólnego przeznaczenia
- Czytanie i zapisywanie różnych formatów danych, ...
- Histogramy, wykresy, ...
- Dopasowywanie funkcji, analiza statystyczna, funkcje matematyczne ...

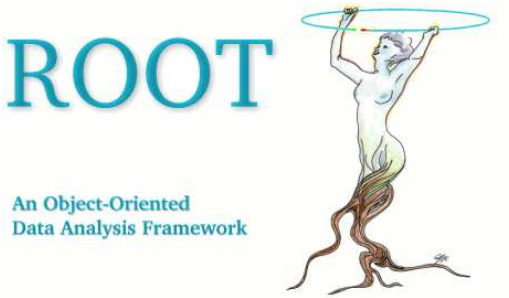
Narzędzia analizy danych

Lista życzeń

- Względnie proste do opanowania: używające popularnego języka programowania lub zbliżonej składni
- Otwarte źródła
- Często procedurę analizy opracowujemy podczas analizy samej w sobie (nie znamy wszystkich założeń a priori) i jest to proces iteracyjny $\rightarrow$ język dynamiczny
- Mile widziany język ogólnego przeznaczenia
- Czytanie i zapisywanie różnych formatów danych, ...
- Histogramy, wykresy, ...
- Dopasowywanie funkcji, analiza statystyczna, funkcje matematyczne ...
- *Wydajność!*

CERN ROOT

- Następca PAW - Physics Analysis Workstation (1986-2014, Fortran)
- CERN ROOT (1994-, C++): kompletne środowisko, obiektowa biblioteka, interpreter (Cling)



- Zawiera format plików (.root) i danych (*tree*), struktury danych (histogramy), narzędzia do prezentacji (wykresy), analizy (dopasowywanie, statystyka) i wiele innych
- Interfejs dla Pythona (>= 6.22) i Ruby
- Dobrze ugruntowana pozycja w fizyce wysokich energii i jądrowej

Programowanie obiektowe (OOP)

Programowanie obiektowe (OOP)

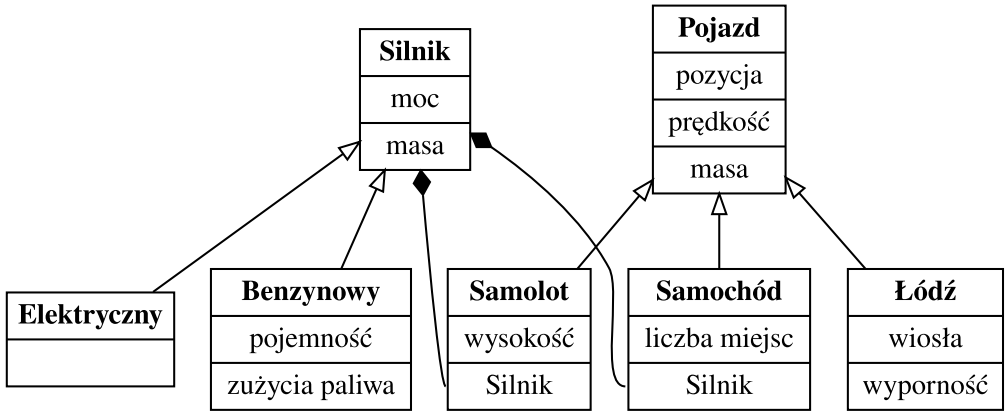
- Podejście obiektowe jest próbą opisu skomplikowanych systemów za pomocą abstrakcji zrozumiałej dla człowieka.

Programowanie obiektowe (OOP)

- Podejście obiektowe jest próbą opisu skomplikowanych systemów za pomocą abstrakcji zrozumiałej dla człowieka.
- Tworzymy zamknięte byty - obiekty, odpowiadające ”fizycznym” obiektom, z którymi się komunikujemy i które mogą się komunikować między sobą.
- Każdy obiekt należy do pewnej klasy i ma swoje zmienne i procedury prywatne i publiczne.
- Prywatną strukturę obiektów można modyfikować, uzupełniać i poprawiać, o ile nie wpływa to na strukturę publiczną.

Programowanie obiektowe (OOP)

- Podejście obiektowe jest próbą opisu skomplikowanych systemów za pomocą abstrakcji zrozumiałej dla człowieka.
- Tworzymy zamknięte byty - obiekty, odpowiadające ”fizycznym” obiektom, z którymi się komunikujemy i które mogą się komunikować między sobą.
- Każdy obiekt należy do pewnej klasy i ma swoje zmienne i procedury prywatne i publiczne.
- Prywatną strukturę obiektów można modyfikować, uzupełniać i poprawiać, o ile nie wpływa to na strukturę publiczną.
- Obiekty mogą dziedziczyć (łączyć się hierachicznie) po bardziej ogólnych klasach (relacja: x jest y) lub mogą zawierać (kompozycja) inne obiekty (relacja: x zawiera y)



Programowanie obiektowe (2)

- Pierwszym językiem obiektowym był Simula (1962) choć pewne koncepcje pojawiały się już wcześniej

Programowanie obiektowe (2)

- Pierwszym językiem obiektowym był Simula (1962) choć pewne koncepcje pojawiały się już wcześniej
- Jednym z najbardziej wpływowych języków był Smalltalk (1972)

Programowanie obiektowe (2)

- Pierwszym językiem obiektowym był Simula (1962) choć pewne koncepcje pojawiały się już wcześniej
- Jednym z najbardziej wpływowych języków był Smalltalk (1972)
- W latach 90-tych szczególnie popularność OOP zyskało dzięki C++ (1985), a następnie Javie (1995)

Programowanie obiektowe (2)

- Pierwszym językiem obiektowym był Simula (1962) choć pewne koncepcje pojawiały się już wcześniej
- Jednym z najbardziej wpływowych języków był Smalltalk (1972)
- W latach 90-tych szczególnie popularność OOP zyskało dzięki C++ (1985), a następnie Javie (1995)
- W ostatnich latach tendencja jest jednak do odchodzenia od czysto obiektowego paradygmatu na rzecz łączenia go np. z podejściem proceduralnym

Programowanie obiektowe (2)

- Pierwszym językiem obiektowym był Simula (1962) choć pewne koncepcje pojawiały się już wcześniej
- Jednym z najbardziej wpływowych języków był Smalltalk (1972)
- W latach 90-tych szczególnie popularność OOP zyskało dzięki C++ (1985), a następnie Javie (1995)
- W ostatnich latach tendencja jest jednak do odchodzenia od czysto obiektowego paradygmatu na rzecz łączenia go np. z podejściem proceduralnym
- OOP kładzie zbyt duży nacisk na typy (klasy) zamiast na algorytmy i struktury danych,

Programowanie obiektowe (2)

- Pierwszym językiem obiektowym był Simula (1962) choć pewne koncepcje pojawiały się już wcześniej
- Jednym z najbardziej wpływowych języków był Smalltalk (1972)
- W latach 90-tych szczególnie popularność OOP zyskało dzięki C++ (1985), a następnie Javie (1995)
- W ostatnich latach tendencja jest jednak do odchodzenia od czysto obiektowego paradygmatu na rzecz łączenia go np. z podejściem proceduralnym
- OOP kładzie zbyt duży nacisk na typy (klasy) zamiast na algorytmy i struktury danych,
- Piętrowe, wielokrotne dziedziczenie nie tylko jest nieefektywne, ale powoduje trudności w zrozumieniu i używaniu bibliotek oraz modyfikowaniu kodu

Programowanie obiektowe (2)

- Pierwszym językiem obiektowym był Simula (1962) choć pewne koncepcje pojawiały się już wcześniej
- Jednym z najbardziej wpływowych języków był Smalltalk (1972)
- W latach 90-tych szczególnie popularność OOP zyskało dzięki C++ (1985), a następnie Javie (1995)
- W ostatnich latach tendencja jest jednak do odchodzenia od czysto obiektowego paradygmatu na rzecz łączenia go np. z podejściem proceduralnym
- OOP kładzie zbyt duży nacisk na typy (klasy) zamiast na algorytmy i struktury danych,
- Piętrowe, wielokrotne dziedziczenie nie tylko jest nieefektywne, ale powoduje trudności w zrozumieniu i używaniu bibliotek oraz modyfikowaniu kodu
- Zalecane jest stosowanie przede wszystkim kompozycji (Composition over inheritance - ”Desing Patterns” (1994))

Programowanie obiektowe (2)

- Pierwszym językiem obiektowym był Simula (1962) choć pewne koncepcje pojawiały się już wcześniej
- Jednym z najbardziej wpływowych języków był Smalltalk (1972)
- W latach 90-tych szczególnie popularność OOP zyskało dzięki C++ (1985), a następnie Javie (1995)
- W ostatnich latach tendencja jest jednak do odchodzenia od czysto obiektowego paradygmatu na rzecz łączenia go np. z podejściem proceduralnym
- OOP kładzie zbyt duży nacisk na typy (klasy) zamiast na algorytmy i struktury danych,
- Piętrowe, wielokrotne dziedziczenie nie tylko jest nieefektywne, ale powoduje trudności w zrozumieniu i używaniu bibliotek oraz modyfikowaniu kodu
- Zalecane jest stosowanie przede wszystkim kompozycji (Composition over inheritance - ”Desing Patterns” (1994))
- ”Chcesz po prostu banana, ale dostajesz goryla trzymającego banana, i jeszcze całą dżunglę.” (Joe Armstrong)

Programowanie obiektowe (2)

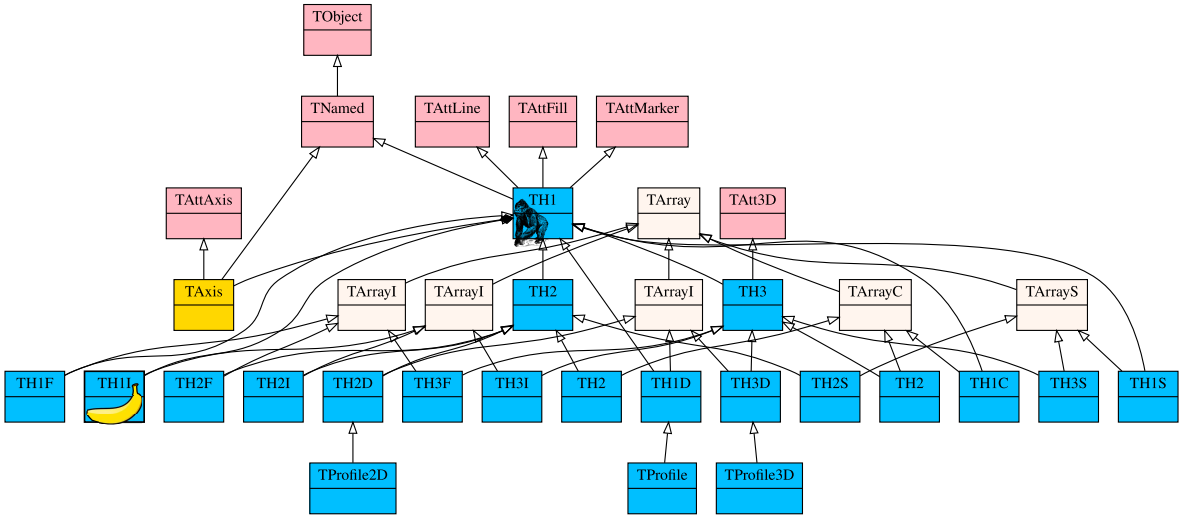
- Pierwszym językiem obiektowym był Simula (1962) choć pewne koncepcje pojawiały się już wcześniej
- Jednym z najbardziej wpływowych języków był Smalltalk (1972)
- W latach 90-tych szczególnie popularność OOP zyskało dzięki C++ (1985), a następnie Javie (1995)
- W ostatnich latach tendencja jest jednak do odchodzenia od czysto obiektowego paradygmatu na rzecz łączenia go np. z podejściem proceduralnym
- OOP kładzie zbyt duży nacisk na typy (klasy) zamiast na algorytmy i struktury danych,
- Piętrowe, wielokrotne dziedziczenie nie tylko jest nieefektywne, ale powoduje trudności w zrozumieniu i używaniu bibliotek oraz modyfikowaniu kodu
- Zalecane jest stosowanie przede wszystkim kompozycji (Composition over inheritance - ”Desing Patterns” (1994))
- ”Chcesz po prostu banana, ale dostajesz goryla trzymającego banana, i jeszcze całą dżunglę.” (Joe Armstrong)

Banan i goryl

- Jednowymiarowy histogram (zliczenia całkowite) - TH1I

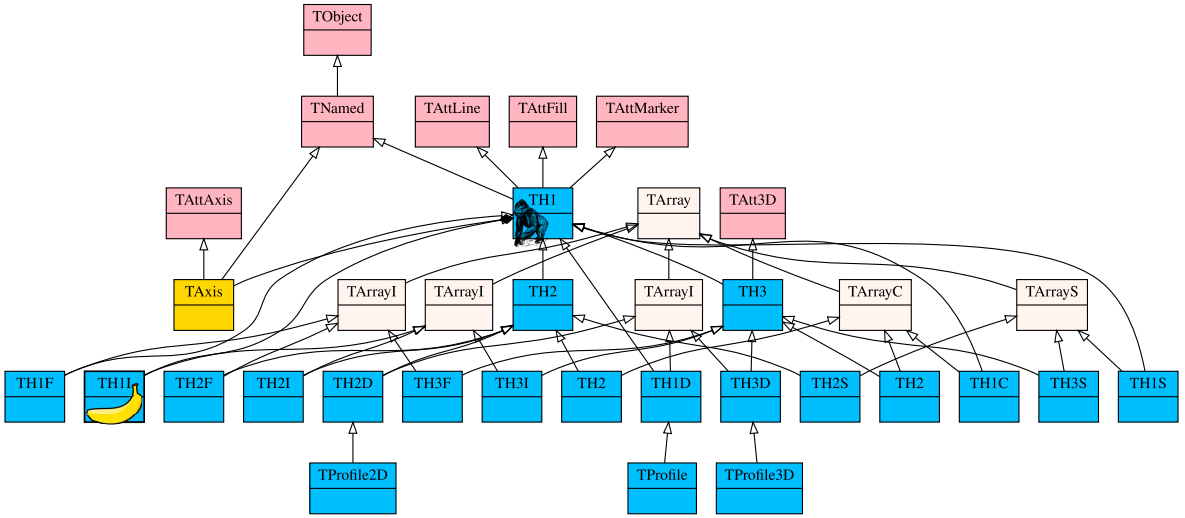
Banan i goryl

- Jednowymiarowy histogram (zliczenia całkowite) - TH1I

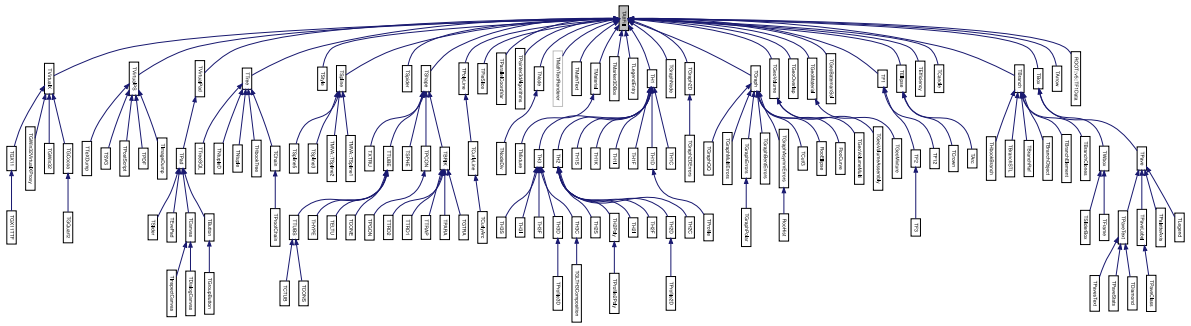


Banan i goryl

- Jednowymiarowy histogram (zliczenia całkowite) - TH1I

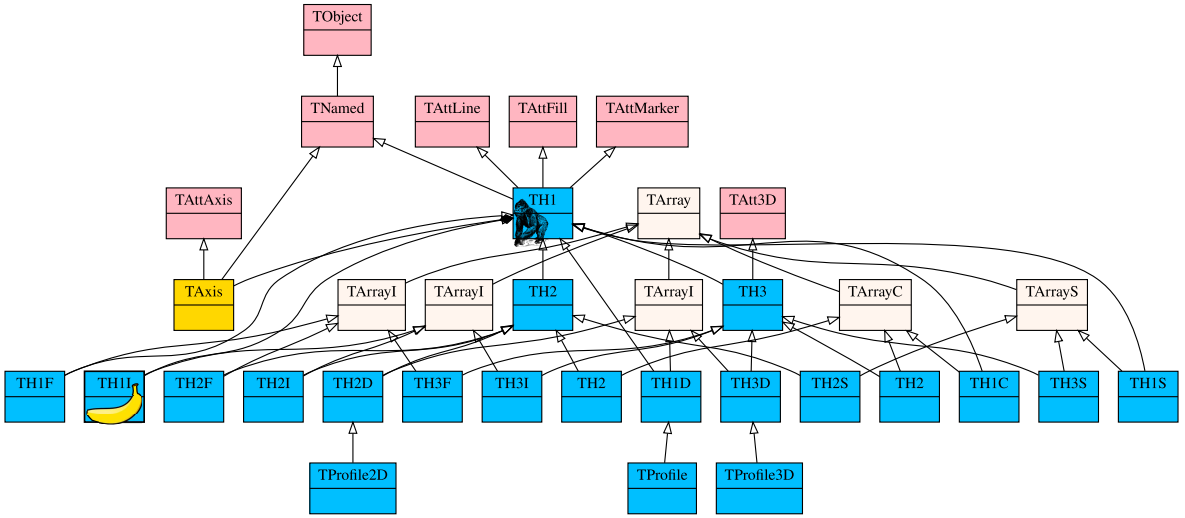


- Niemal każdy obiekt dziedziczy po klasach TAttLine, TAttFill, TAttMarker (krój, wypełnienie i kształt znacznika na wykresie)!

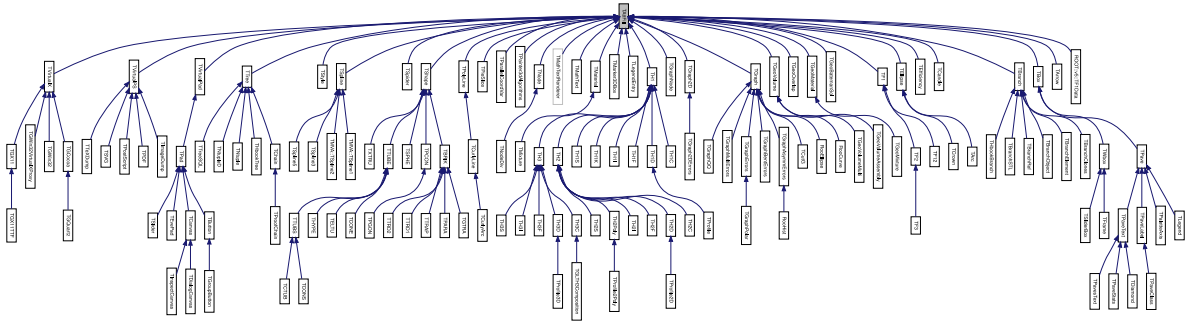


Banan i goryl

- Jednowymiarowy histogram (zliczenia całkowite) - TH1I



- Niemal każdy obiekt dziedziczy po klasach TAttLine, TAttFill, TAttMarker (krój, wypełnienie i kształt znacznika na wykresie)!



- W przypadku histogramu (TH1), funkcji (TF1) można to jeszcze zrozumieć. Ale jaki jest związek struktury danych TTree z kształtem linii? Albo backendu grafiki TVirtualX?

Banan i goryl i dżungla

Przy tak skomplikowanej strukturze nie pomaga uboga dokumentacja

```
Long64_t TTree::Project (const char* hname, const char* varexp,
                         const char* selection = "",
                         Option_t* option = "",
                         Long64_t  nentries = kMaxEntries,
                         Long64_t firstentry = 0 )virtual

Make a projection of a tree using selections.
Depending on the value of varexp (described in Draw) a 1-D, 2-D, etc.,
projection of the tree will be filled in histogram hname.
Note that the dimension of hname must match with the dimension of varexp.

Definition at line 7385 of file TTree.cxx.
```

Problem?

Skoro działa to nie ma problemu?

CERN ROOT

Zalety:

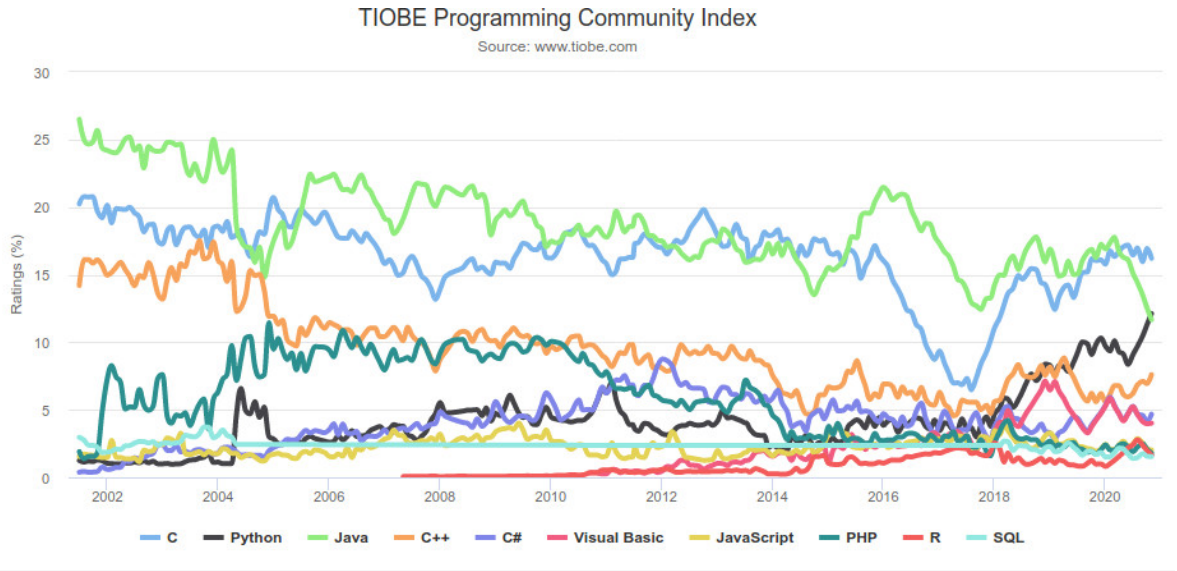
- Działa
- Dobrze znany w fizyce wysokich energii i jądrowej
- Dopasowany do naszych wymagań
- Zawiera wszystkie potrzebne elementy
- Szybki i wydajny

Wady:

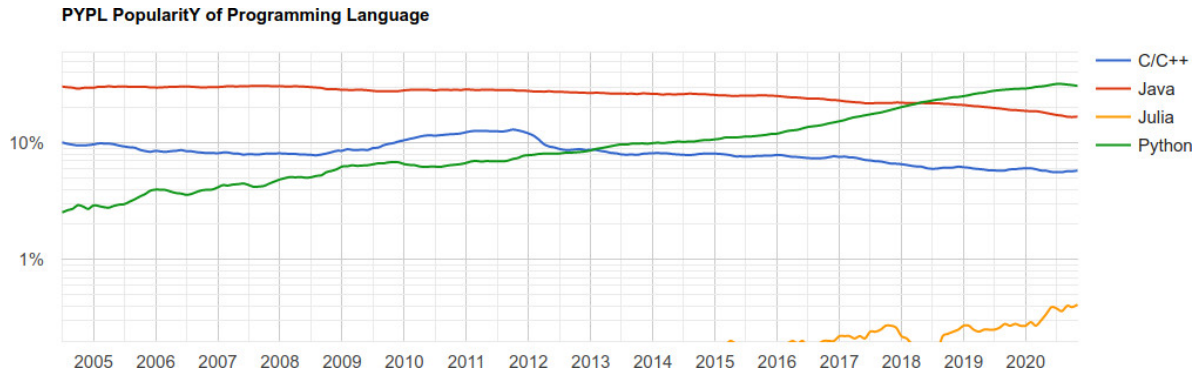
- Monolityczna struktura (wszystko albo nic)
- Uboga dokumentacja
- Stosunkowo trudny do nauki
- Stosunkowo mała grupa użytkowników
- Nieznany i nieużywany w szerszym świecie
- Oparty na C++

Popularność języków

- Indeks TIOBE (popularność wyszukiwania, liczba stron)

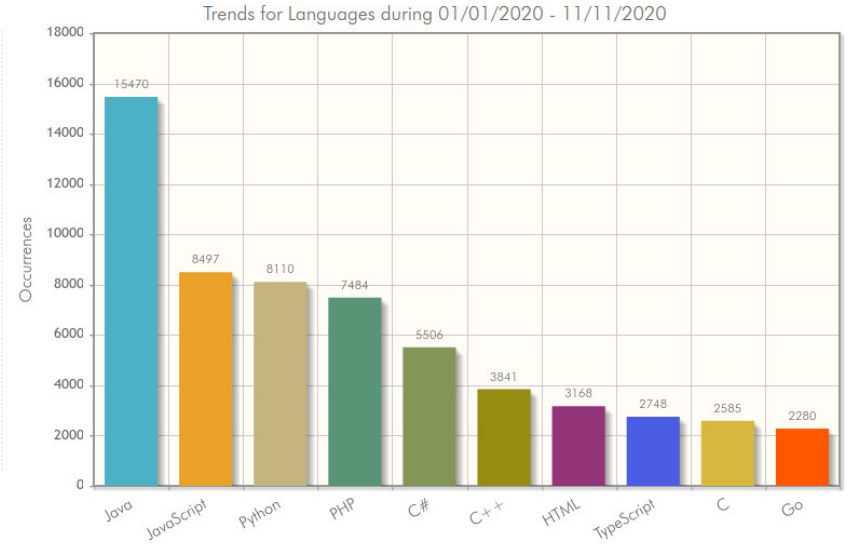


- PYPL (popularność wyszukiwania poradników)

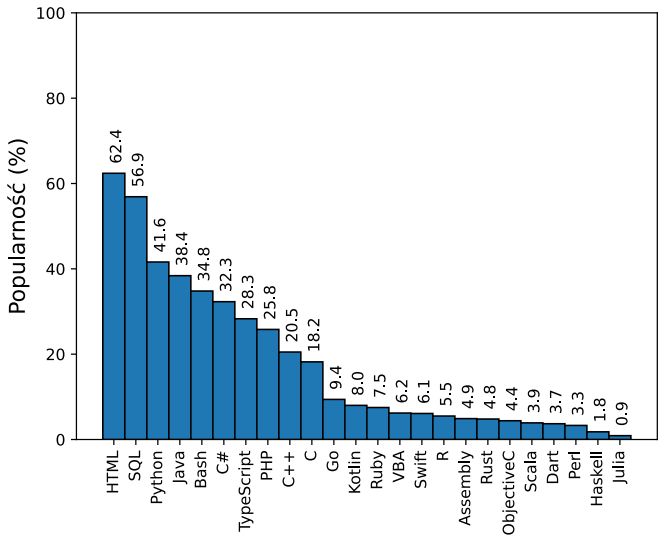


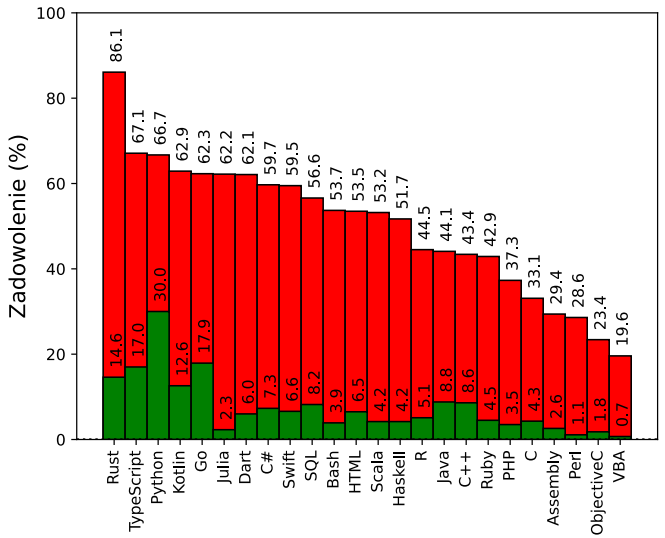
Popularność języków (2)

- Trendy Skills (liczba ogłoszeń o pracę)



- Stackoverflow (ankieta wśród programistów)





Popularność języków (3)

Dlaczego powinno nas to interesować? Większa popularność to:

- Więcej materiałów, podręczników, porad
- Lepszy rozwój języka (poprawki, rozszerzenia)
- Większe zainteresowanie studentów, doktorantów, post-doców
- Większe możliwości nabycia przydatnych umiejętności

Popularność języków (3)

Dlaczego powinno nas to interesować? Większa popularność to:

- Więcej materiałów, podręczników, porad
- Lepszy rozwój języka (poprawki, rozszerzenia)
- Większe zainteresowanie studentów, doktorantów, post-doców
- Większe możliwości nabycia przydatnych umiejętności

Analiza dużych zbiorów danych to popularny temat, dynamicznie rozwijający się wraz ze wzrostem możliwości mikrokomputerów.
Czego używają inni w tej dziedzinie?

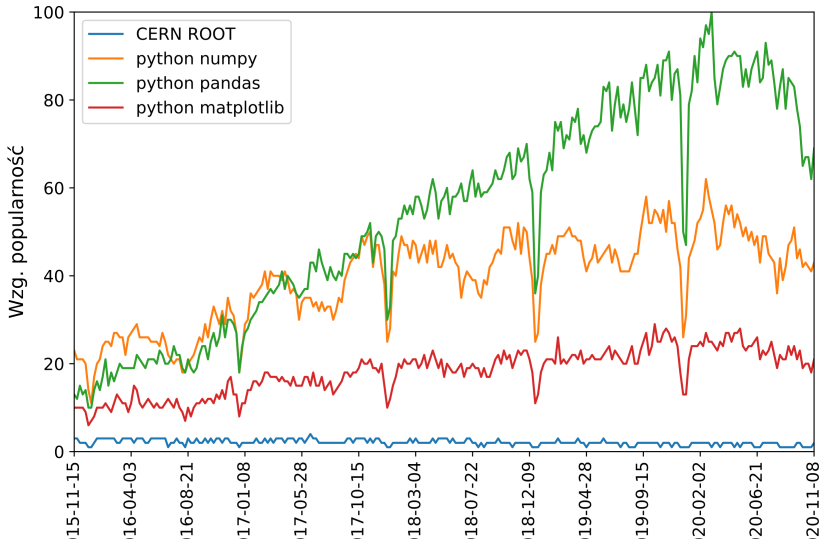
Popularność języków (3)

Dlaczego powinno nas to interesować? Większa popularność to:

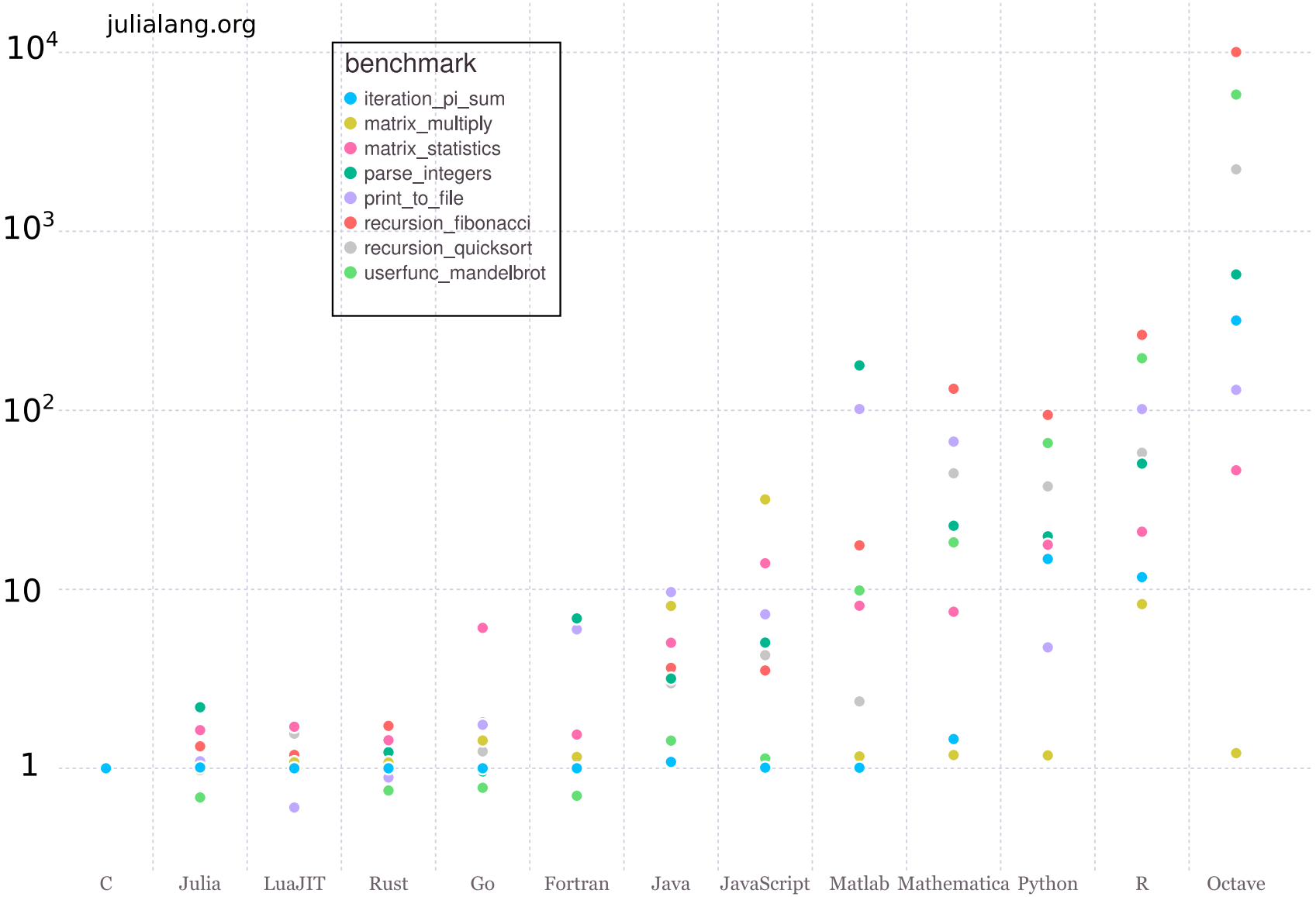
- Więcej materiałów, podręczników, porad
- Lepszy rozwój języka (poprawki, rozszerzenia)
- Większe zainteresowanie studentów, doktorantów, post-doców
- Większe możliwości nabycia przydatnych umiejętności

Analiza dużych zbiorów danych to popularny temat, dynamicznie rozwijający się wraz ze wzrostem możliwości mikrokomputerów.
Czego używają inni w tej dziedzinie?

- *Python* (66%)
- R (47%) - problem: wydajność
- SQL (33%) - problem: nieprzydatny do naszych celów
- Java (12%) - problem: mało rozbudowane biblioteki do analizy
- C++ (7%)
- Scala (4%) - problem: rozszerzenie Javy, niszowy język
- *Julia* (2%) - nowy język (2012) wart uwagi



Porównanie języków



Język Python

- Stworzony w 1991 przez Guido van Rossuma
- Nazwa pochodzi od „Monty Python Flying Circus”
- Jest językiem wysokiego poziomu ogólnego przeznaczenia

Język Python

- Stworzony w 1991 przez Guido van Rossuma
- Nazwa pochodzi od „Monty Python Flying Circus”
- Jest językiem wysokiego poziomu ogólnego przeznaczenia
- Założenia projektowe (wybór z ZEN of Python - PEP20)
  - Explicit is better than implicit.
  - Simple is better than complex.
  - Flat is better than nested.
  - Sparse is better than dense.
  - Readability counts.
  - Special cases aren’t special enough to break the rules.
  - Although practicality beats purity.
  - Now is better than never.
  - There should be one – and preferably only one – obvious way to do it.

Język Python

- Stworzony w 1991 przez Guido van Rossuma
- Nazwa pochodzi od „Monty Python Flying Circus”
- Jest językiem wysokiego poziomu ogólnego przeznaczenia
- Założenia projektowe (wybór z ZEN of Python - PEP20)
  - Explicit is better than implicit.
  - Simple is better than complex.
  - Flat is better than nested.
  - Sparse is better than dense.
  - Readability counts.
  - Special cases aren’t special enough to break the rules.
  - Although practicality beats purity.
  - Now is better than never.
  - There should be one – and preferably only one – obvious way to do it.
- Programowanie proceduralne, obiektowe i funkcyjne
- Język interpretowany, typowanie dynamiczne i słabe
- Automatyczne zarządzanie pamięcią

Język Python

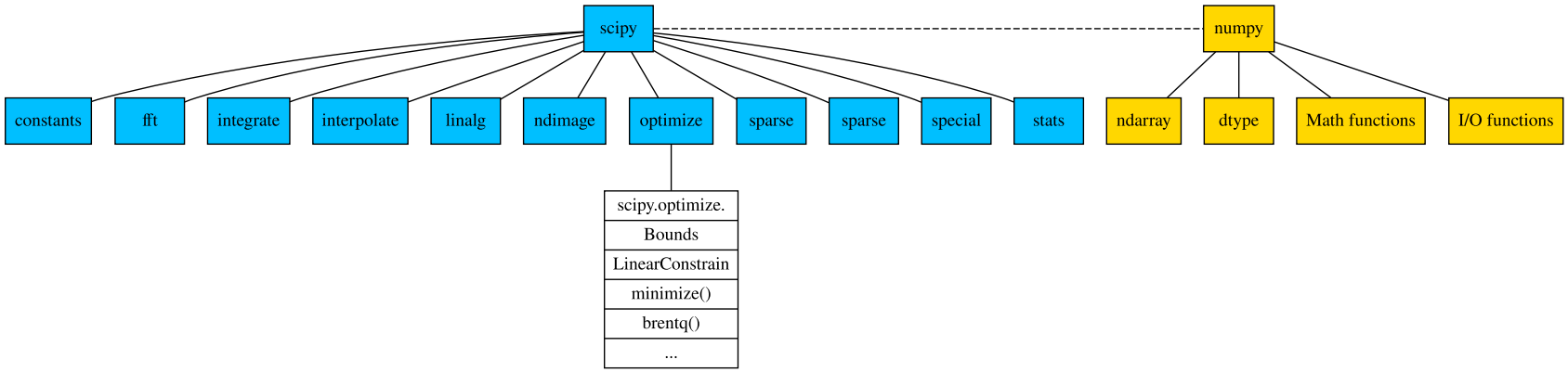
- Stworzony w 1991 przez Guido van Rossuma
- Nazwa pochodzi od „Monty Python Flying Circus”
- Jest językiem wysokiego poziomu ogólnego przeznaczenia
- Założenia projektowe (wybór z ZEN of Python - PEP20)
  - Explicit is better than implicit.
  - Simple is better than complex.
  - Flat is better than nested.
  - Sparse is better than dense.
  - Readability counts.
  - Special cases aren’t special enough to break the rules.
  - Although practicality beats purity.
  - Now is better than never.
  - There should be one – and preferably only one – obvious way to do it.
- Programowanie proceduralne, obiektowe i funkcyjne
- Język interpretowany, typowanie dynamiczne i słabe
- Automatyczne zarządzanie pamięcią
- Bardzo bogata biblioteka standardowa (batteries included)
- Bogate biblioteki (i są łatwe do instalowania)

Krótki przegląd bibloteki standardowej

- *argparse* - interfejs linii poleceń (parametry)
- *datetime* - operacje związane z czasem i kalendarzem
- *distutils* - narzędzia do instalowania i tworzenia pakietów Pythona
- *doctest* - testowanie dokumentacji
- *email*
- *html*
- *http*
- *math* - funkcje matematyczne
- *multiprocessing* - przetwarzanie wielowątkowe
- *os* - interakcje z systemem operacyjnym
- *random* - generatory liczb losowych
- *re* - wyrażenia regularne
- *smtpd* - serwer SMTP
- *smtplib* - klient SMTP
- *socketserver* - operacje sieciowe niskiego poziomu
- *ssl* - OpenSSL
- *sqlite3* - interfejs bazy danych SQLite
- *timeit* - pomiary wydajności programu
- *tkinter* - Tk GUI
- *unittest* - testowanie programu
- *xml*
- *zipfile*

Analiza danych

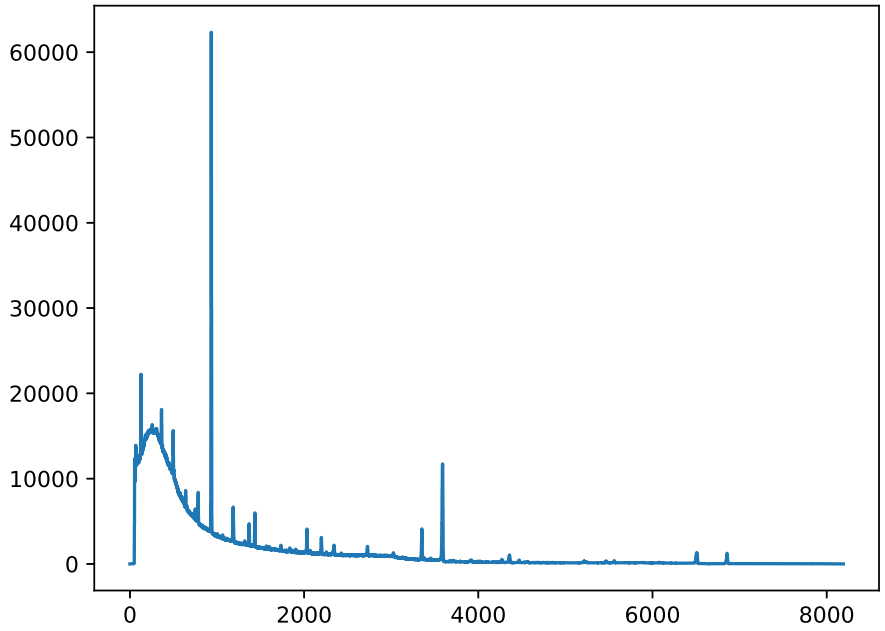
- *numpy* - biblioteka numeryczna
- *matplotlib* - bogata biblioteka do tworzenia wykresów
- *ipython/jupyter* - rozszerzony interaktywny interpreter REPL w konsoli / przeglądarce
- *SymPy* - obliczenia symboliczne
- *Pandas* - struktury danych i ich analiza
- *scipy* - scientific Python = wszystko powyżej + dodatkowe narzędzia



Przykład

Wczytaj dane tekstowe (widmo) i narysuj wykres

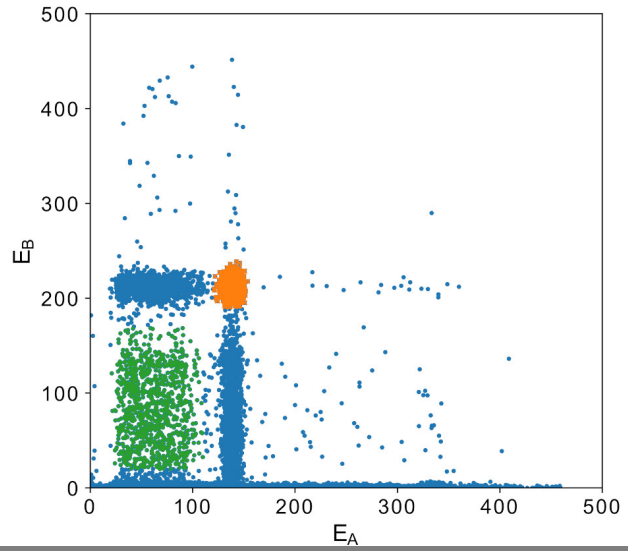
```
import numpy
import matplotlib.pyplot as plt

data = numpy.loadtxt('dane.txt')
x = numpy.arange(data.shape)
plt.plot(x, data, ds='steps-mid')
plt.show()
```

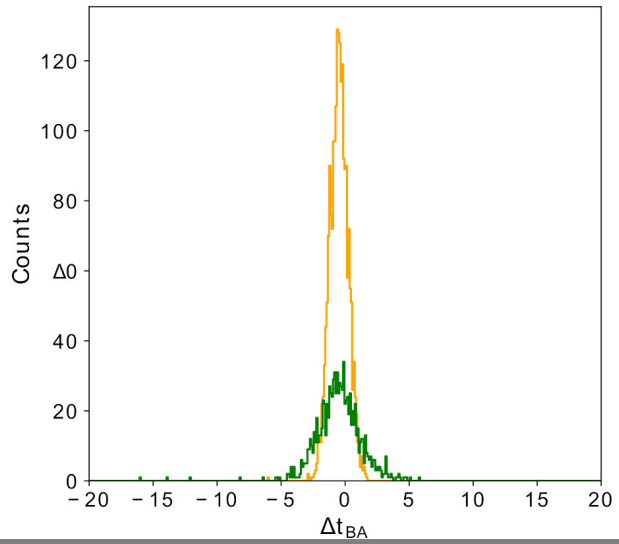


Przykład

Wczytaj zdarzenia, wybierz ich podzbiory i narysuj różnice czasowe

```
import pandas
import matplotlib.pyplot as plt

df = pandas.read_csv('events.txt', delim_whitespace=True, comment='#',
                         names=['EA', 'EB', 'tA', 'tB'])

sel1 = df[df.EA > 120][df.EA < 160][df.EB > 190][df.EB < 240]
sel2 = df[df.EA > 20][df.EA < 110][df.EB > 20][df.EB < 170]

(...)

bins, edges = numpy.histogram(sel1.tA - sel1.tB, bins=400,
                                  range=(-20, 20))
plot.plot(edges[:-1], bins, ds='steps-mid')
bins, edges = numpy.histogram(sel2.tA - sel2.tB, bins=400,
                                  range=(-20, 20))
plt.plot(edges[:-1], bins, ds='steps-mid')
```

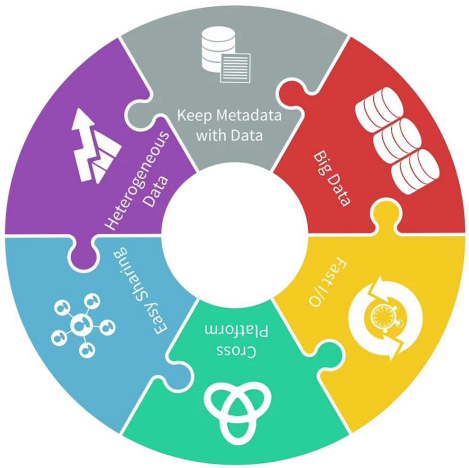




HDF5 (1)

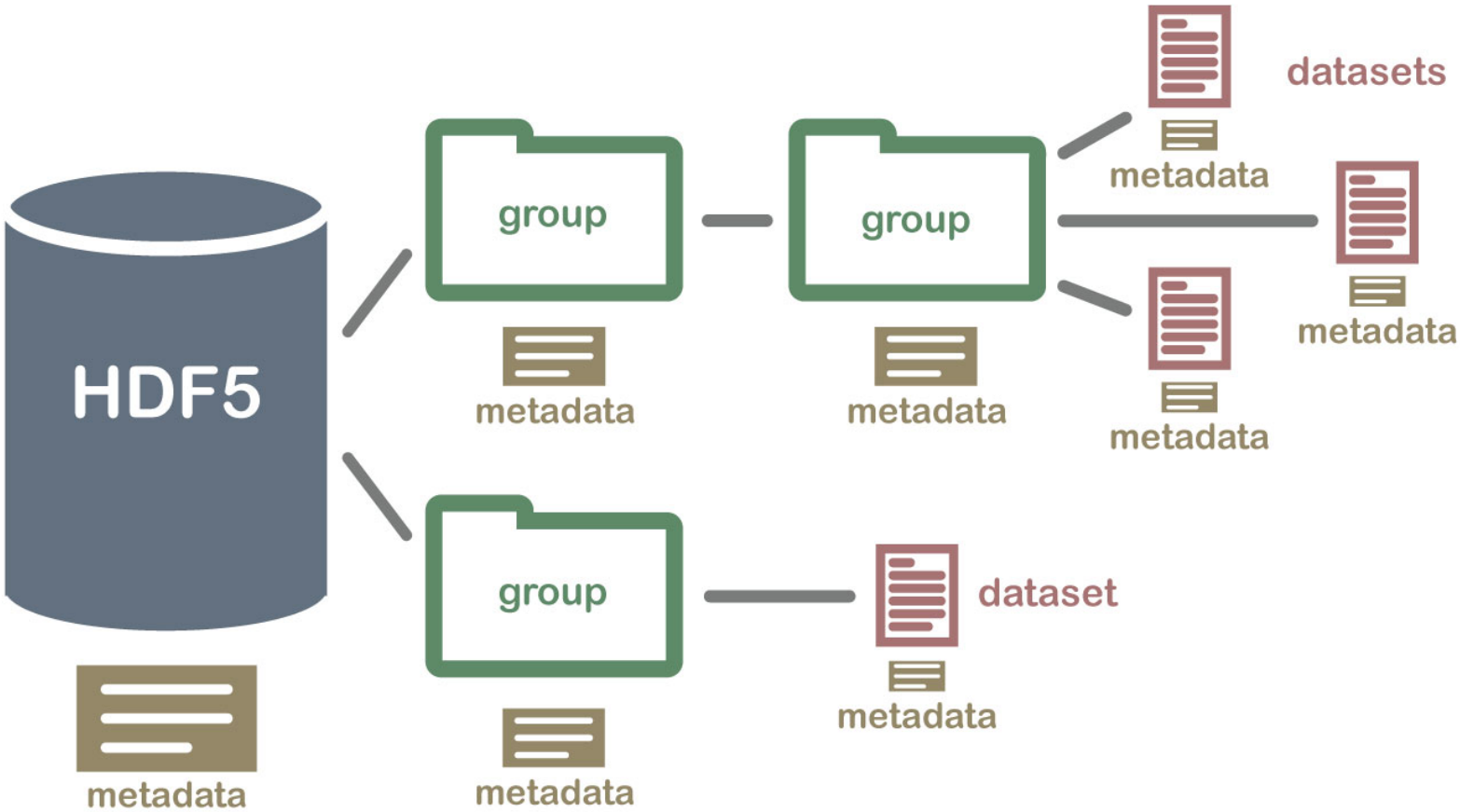
Hierarchical Data Format

- Rozwój formatu rozpoczęty w 1987
- Obecna wersja HDF5 (5-1.10.7, 09.2020)
- Biblioteki dla języków: C, C++, Fortran, Java, Python, Matlab, R, Mathematica, ...
- Używany m.in przez NASA, DOE, LBNL, banki, przemysł (Ford, Airbus, ...)
- Elastyczny format pozwalający na reprezentacje dowolnych danych
- Brak limitów rozmiaru przechowywanych danych
- Otwartoźródłowy kod



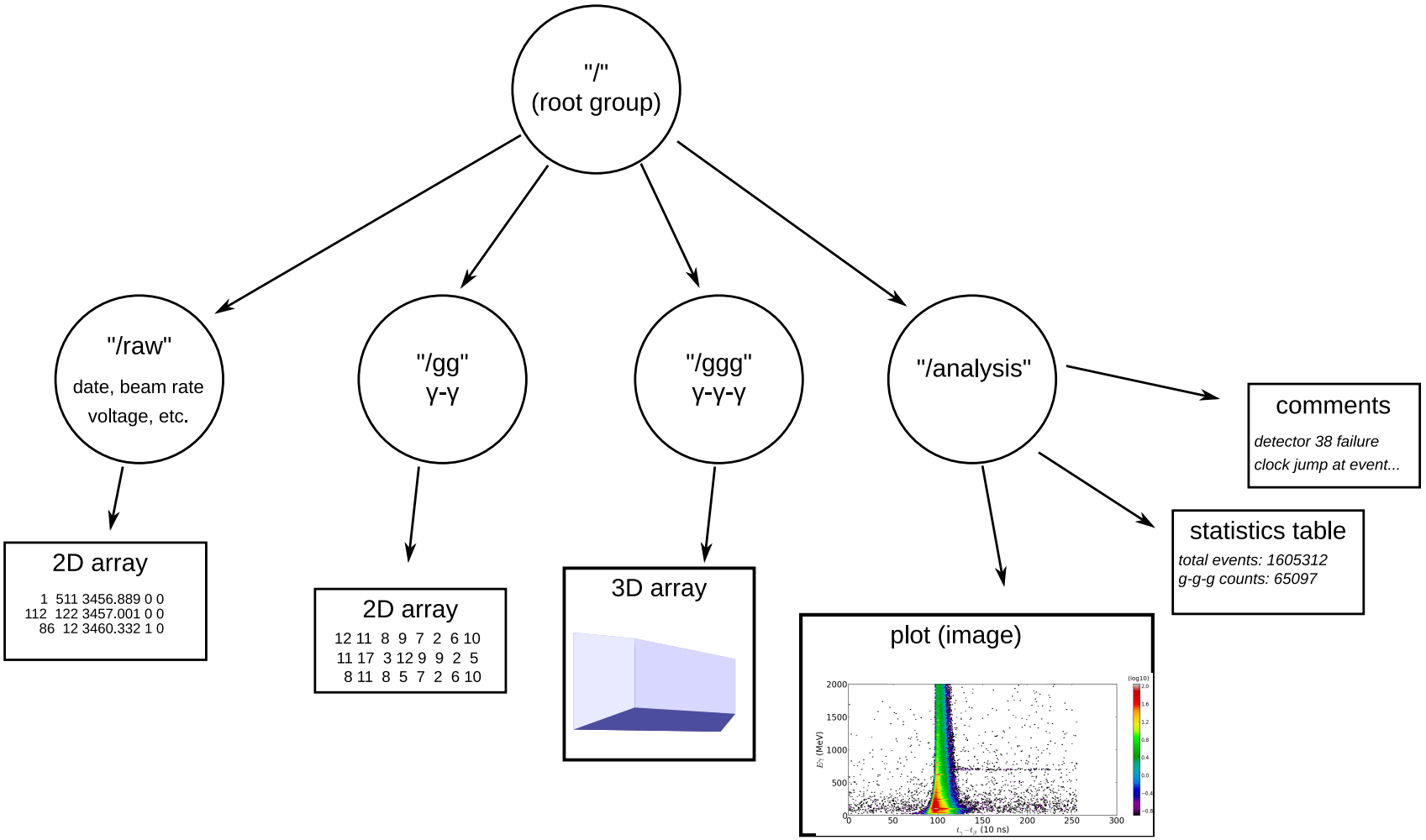
HDF5 (2)

- Dataset: n-wymiarowa tablica elementów o pewnym typie danych + meta-dane
- Group: ”katalog” zawierający inne grupy lub zbiory danych + metadane



HDF5 (2)

- Dataset: n-wymiarowa tablica elementów o pewnym typie danych + meta-dane
- Group: ”katalog” zawierający inne grupy lub zbiory danych + metadane



HDF5 (3)

- Kod w Pythonie ładujący dane z grupy i zbioru danych

```
import h5py

fin = h5py.File('name.h5', 'r')
data = fin['group_name/dataset_name']

# Analyze, plot, etc...

fin.close()
```

HDF5 (3)

- Kod w Pythonie ładujący dane z grupy i zbioru danych

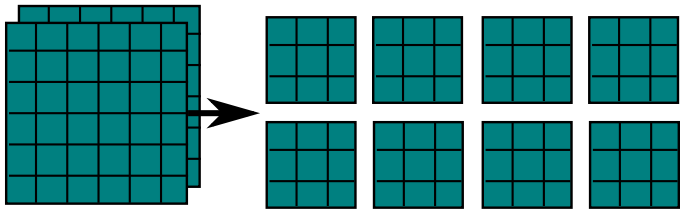
```
import h5py

fin = h5py.File('name.h5', 'r')
data = fin['group_name/dataset_name']

# Analyze, plot, etc...

fin.close()
```

- Kod tworzący plik, grupę i zapisujący dane

```
import h5py

fout = h5py.File('name.h5', 'a')
group = fout.create_group('events')
matrix = group.create_dataset('data', (10000, 3), dtype='u4')

# Reading/Writing supports fancy indexing
# matrix[:, 0] = ...
# matrix[:,::2] = ...

fout.close()
```

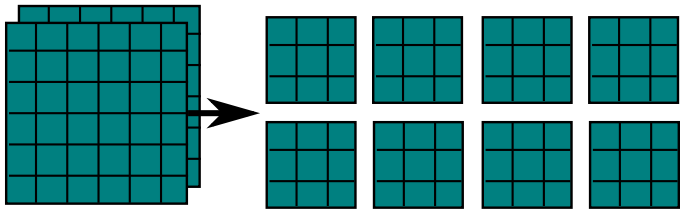
Użyteczne własności (1)

- Segmentowane przechowywanie (chunks)
  - Dane są przechowywane w porcjach o wybranym rozmiarze i mogą być wydajnie ładowane do pamięci
  - Jest to całkowicie przeźroczyste dla użytkownika, ale iteracja w ”złym” kierunku powoduje niską wydajność

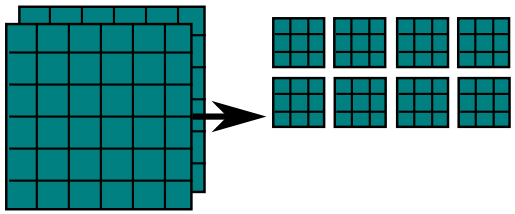


Użyteczne własności (1)

- Segmentowane przechowywanie (chunks)
  - Dane są przechowywane w porcjach o wybranym rozmiarze i mogą być wydajnie ładowane do pamięci
  - Jest to całkowicie przeźroczyste dla użytkownika, ale iteracja w ”złym” kierunku powoduje niską wydajność

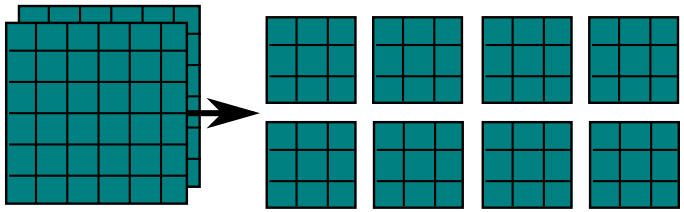


- Kompresja na bieżąco (różne algorytmy)
  - Przeźroczysta dla użytkownika
  - Zmniejsza rozmiar pliku, ale zwiększa czas dostępu

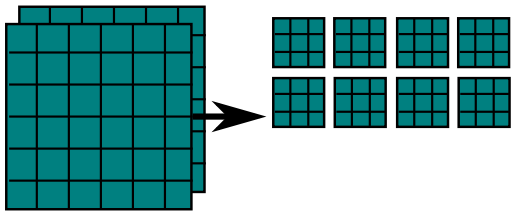


Użyteczne własności (1)

- Segmentowane przechowywanie (chunks)
  - Dane są przechowywane w porcjach o wybranym rozmiarze i mogą być wydajnie ładowane do pamięci
  - Jest to całkowicie przeźroczyste dla użytkownika, ale iteracja w ”złym” kierunku powoduje niską wydajność



- Kompresja na bieżąco (różne algorytmy)
  - Przeźroczysta dla użytkownika
  - Zmniejsza rozmiar pliku, ale zwiększa czas dostępu



- Przykład

```
matrix = file.create_dataset('g', (D, D, D), chunks=(1, D, D),
                             dtype='u4', compression='gzip')
```

Użyteczne własności (2)

- Rozmiar danych ograniczony pojemnością dysku (rozszerzalne na żądanie)

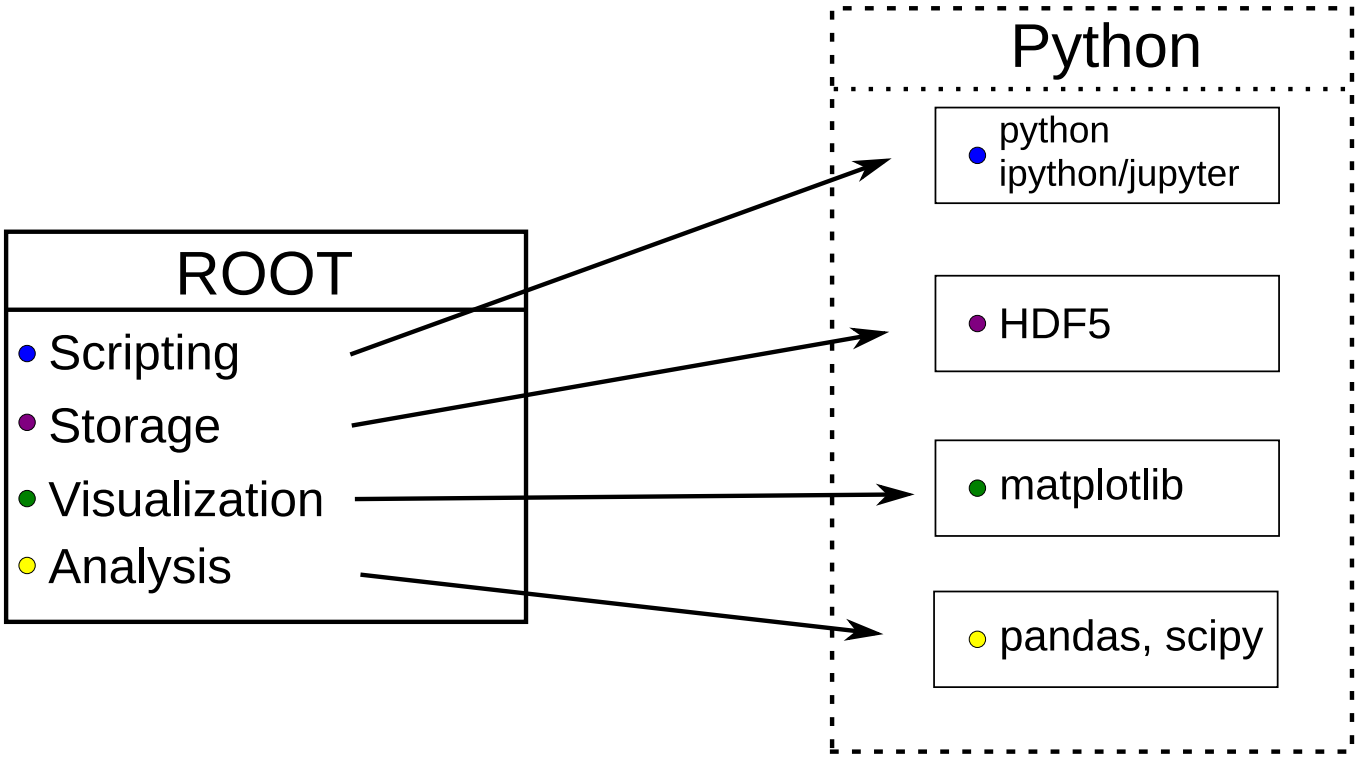
Użyteczne własności (2)

- Rozmiar danych ograniczony pojemnością dysku (rozszerzalne na żądanie)
- Single Writer Multiple Readers (SWMR)
  - Wiele czytających procesów
  - Mamy gwarancje, że plik jest w stanie umożliwiającym poprawne czytanie
  - Proces piszący dodaje dane normalnie
  - Czytający proces może sprawdzić czy pojawiły się nowe dane
  - Możliwe do wykorzystania w systemach on-line

Użyteczne własności (2)

- Rozmiar danych ograniczony pojemnością dysku (rozszerzalne na żądanie)
- Single Writer Multiple Readers (SWMR)
  - Wiele czytających procesów
  - Mamy gwarancje, że plik jest w stanie umożliwiającym poprawne czytanie
  - Proces piszący dodaje dane normalnie
  - Czytający proces może sprawdzić czy pojawiły się nowe dane
  - Możliwe do wykorzystania w systemach on-line
- Przetwarzanie równoległe (MPI)
  - Operacje kolektywne (np. tworzenie grupy) - wspólne dla wszystkich procesów
  - Operacje niezależne (np. czytanie danych) - możliwe do zrównolegnienia

Root vs Python



Język Julia

- Język powstał w 2012 (obecna wersja 1.5) na MIT
- Celem było stworzenie języka wysokiego rzędu o wysokiej wydajności
- Jest językiem ogólnego przeznaczenia, ale przede wszystkim jest tworzony pod kątem analizy numerycznej i analizy danych
- Używa kompilacji JIT, przygotowany jest do obliczeń współbieżnych
- Interpreter REPL
- Możliwość bezpośredniego wołania funkcji w C i Fortranie
- Ma bogate biblioteki do wizualizacji, analizy danych, uczenia maszynowego, naukowe (np. BioJulia, AstroJulia)
- Obok C, C++ i Fortranu użyty do obliczeń w skali PetaFLOPS (projekt Celeste)

Julia - przykład

Wybór ciekawych rozwiązań

Julia - przykład

Wybór ciekawych rozwiązań

- Zmienne UTF-8 δ `= 0.00001,` $\sum$`(x,y) = x + y`

Julia - przykład

Wybór ciekawych rozwiązań

- Zmienne UTF-8 `δ = 0.00001, ∑(x,y) = x + y`
- Zwektoryzowane operatory i funkcje

```
A = [1, 2, 3]
B = A.^2
C = sin.(A)
```

Julia - przykład

Wybór ciekawych rozwiązań

- Zmienne UTF-8 δ `= 0.00001,` $\sum$`(x,y) = x + y`
- Zwektoryzowane operatory i funkcje

```
A = [1, 2, 3]
B = A.^2
C = sin.(A)
```

- Domyślne mnożenie `x -> x^2 + 2x - 1` (tu w funkcji anonimowej)

Julia - przykład

Wybór ciekawych rozwiązań

- Zmienne UTF-8 `δ = 0.00001`, `∑(x,y) = x + y`
- Zwektoryzowane operatory i funkcje

```
A = [1, 2, 3]
B = A.^2
C = sin.(A)
```

- Domyślne mnożenie `x -> x^2 + 2x - 1` (tu w funkcji anonimowej)
- Deklaracje zmiennych i funkcji bez typu i z typem (wydajność!)

```
function f(x)
    T = 0
    ...
end
```

```
function f(x::Array{Float64, 1})::Float64
    T = Int64(0)
    ...
end
```

Julia - przykład

Wybór ciekawych rozwiązań

- Zmienne UTF-8 `δ = 0.00001`, `∑(x,y) = x + y`
- Zwektoryzowane operatory i funkcje

```
A = [1, 2, 3]
B = A.^2
C = sin.(A)
```

- Domyślne mnożenie `x -> x^2 + 2x - 1` (tu w funkcji anonimowej)
- Deklaracje zmiennych i funkcji bez typu i z typem (wydajność!)

```
function f(x)
    T = 0
    ...
end
```

```
function f(x::Array{Float64, 1})::Float64
    T = Int64(0)
    ...
end
```

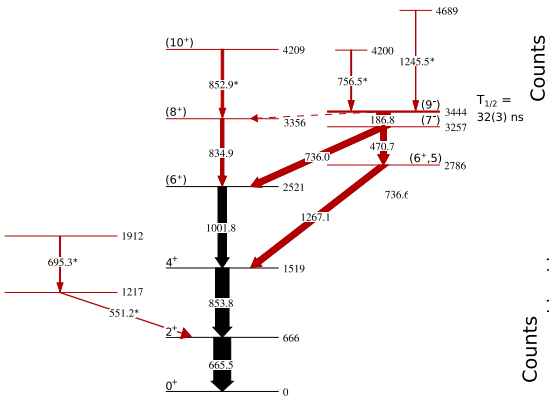
- Przykład wczytania pliku i stworzenia wykresu widma

```
import HDF5
import Plots

data_file = HDF5.h5open("data_file.h5", "r")
A = HDF5.read(data_file["/group/dataset/"])
Plots.plot(A)
```

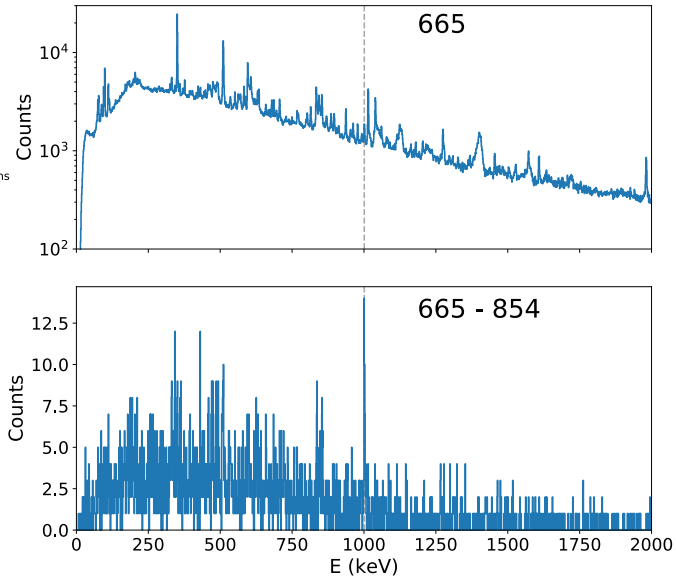
Budowa macierzy 3D

Budowa macierzy 3D

W eksperymencie rozszczepieniowym macierze $\gamma - \gamma - \gamma$ są koniecznością w przypadku badania koincydencji

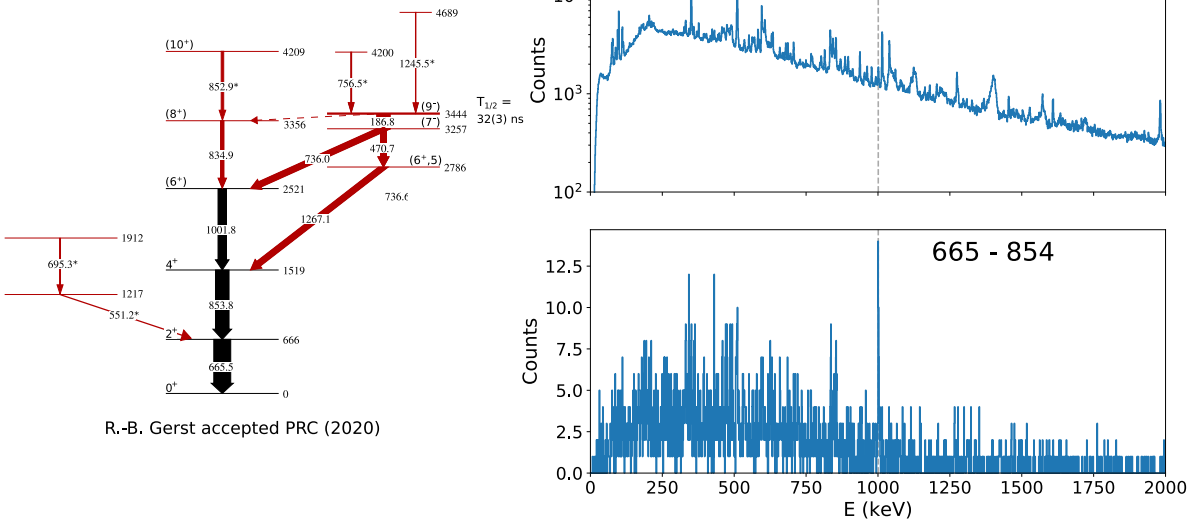


R.-B. Gerst accepted PRC (2020)



Budowa macierzy 3D

W eksperymencie rozszczepieniowym macierze $\gamma - \gamma - \gamma$ są koniecznością w przypadku badania koincydencji

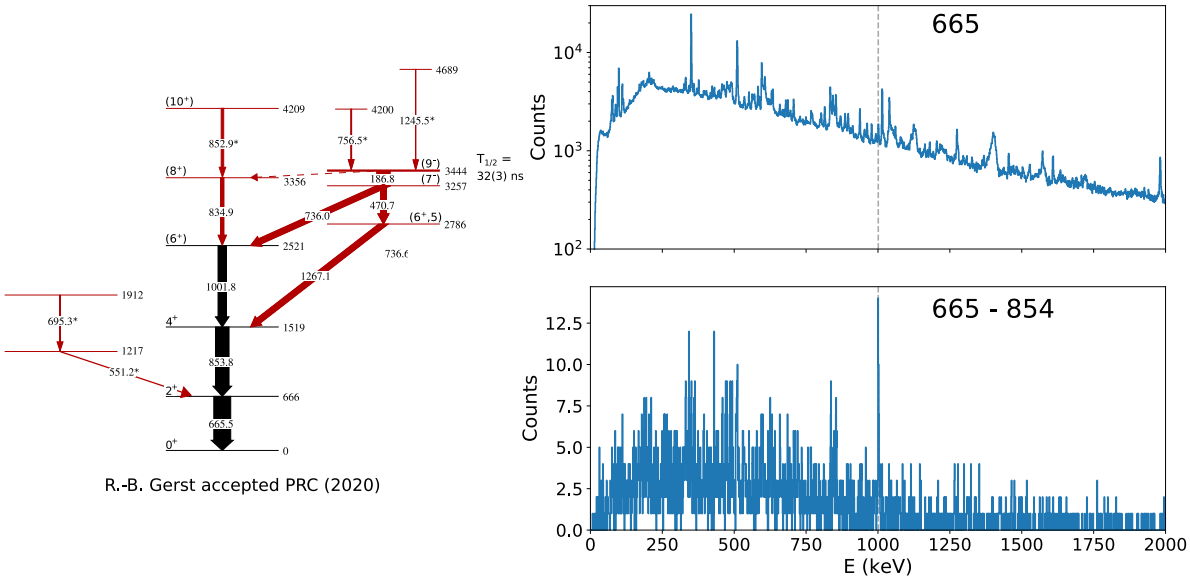


R.-B. Gerst accepted PRC (2020)

- Wersja A (bezpośrednio)
  - Skanowanie danych, zapisywanie koincydencji γ-γ-γ do histogramu 3D
  - Wolne skanowanie
  - Szybkie bramkowanie

Budowa macierzy 3D

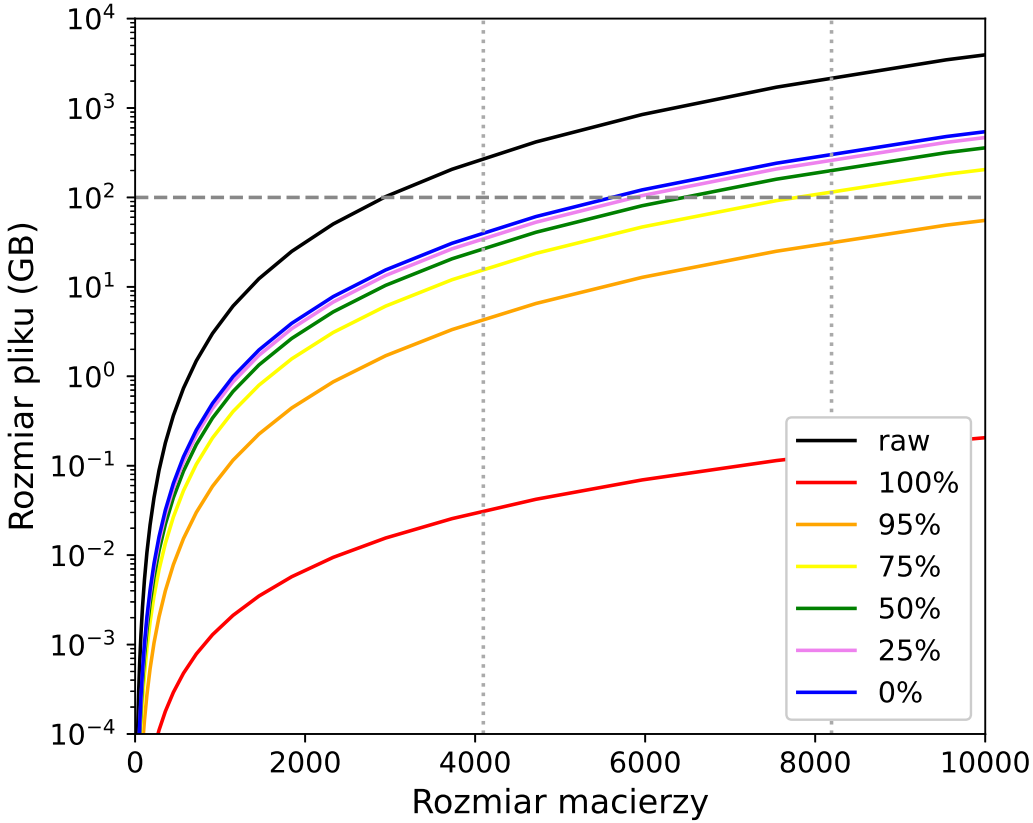
W eksperymencie rozszczepieniowym macierze $\gamma - \gamma - \gamma$ są koniecznością w przypadku badania koincydencji



R.-B. Gerst accepted PRC (2020)

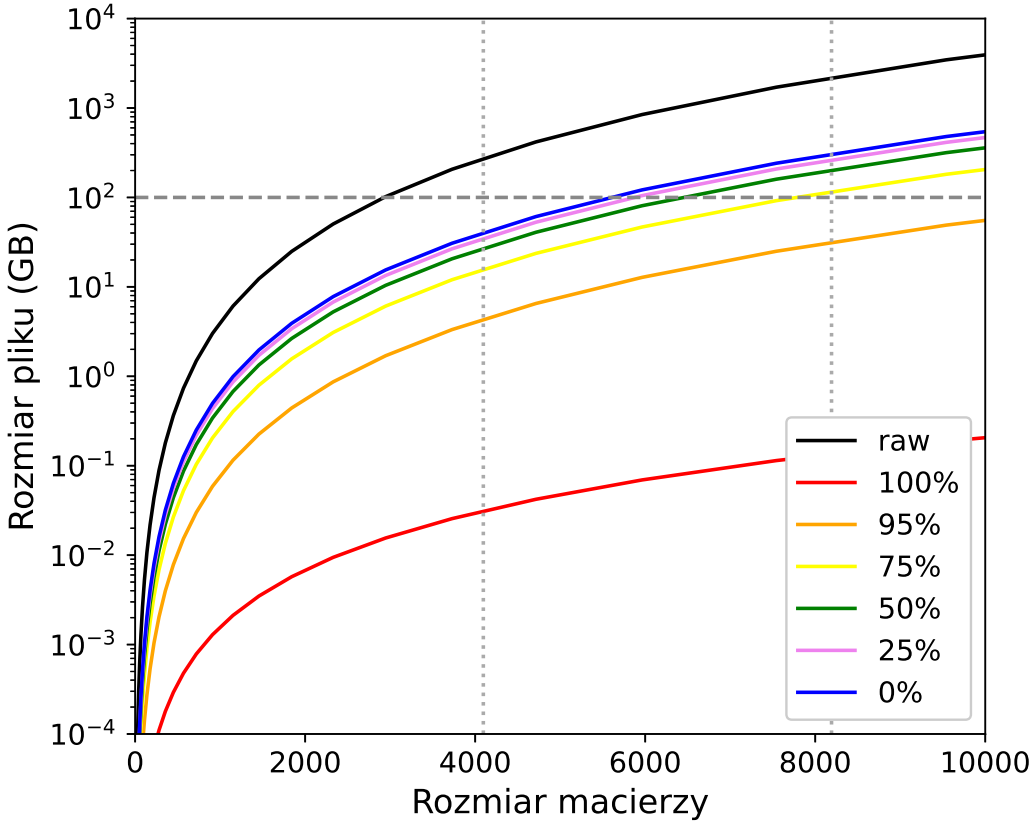
- Wersja A (bezpośrednio)
  - Skanowanie danych, zapisywanie koincydencji γ-γ-γ do histogramu 3D
  - Wolne skanowanie
  - Szybkie bramkowanie
- Wersja B (lista zdarzeń)
  - Skanowanie danych, zapisywanie listy zdarzeń γ-γ-γ
  - Szybsze skanowanie
  - Wolne bramkowanie

Macierze 3D (A)



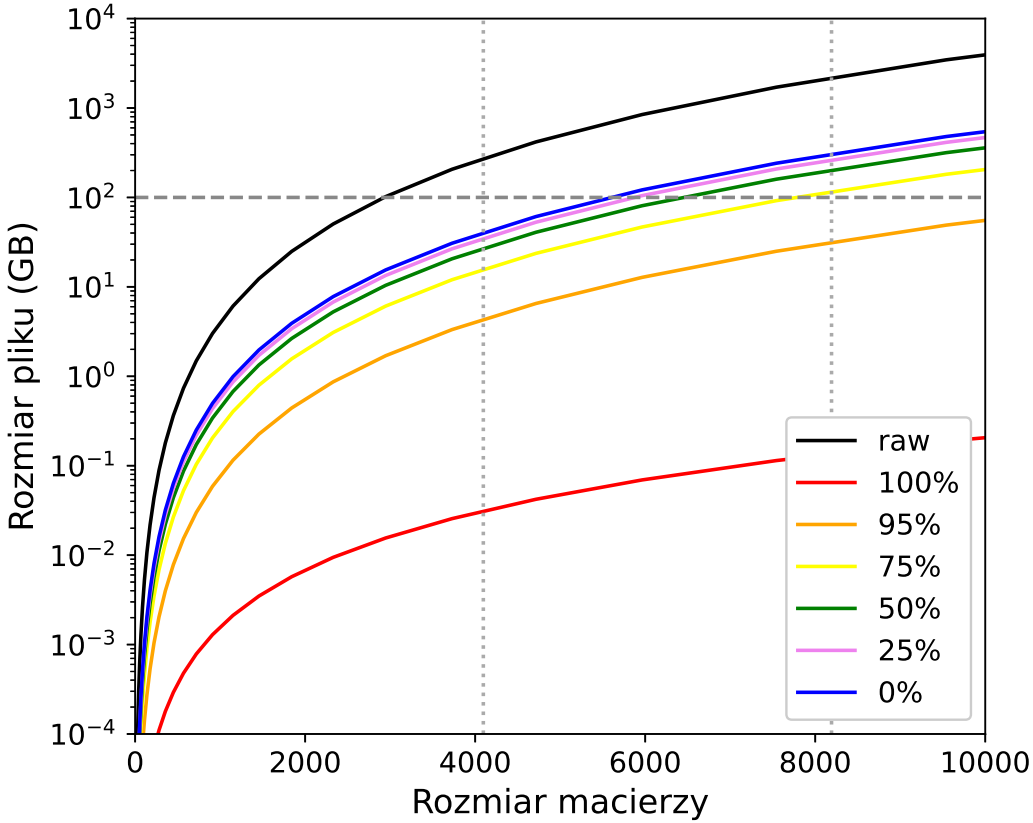
- Macierz $\gamma\gamma\gamma$ z eksperymentu ν-Ball

Macierze 3D (A)



- Macierz $\gamma\gamma\gamma$ z eksperymentu ν-Ball
- Czas tworzenia pliku $\sim$20 dni.

Macierze 3D (A)

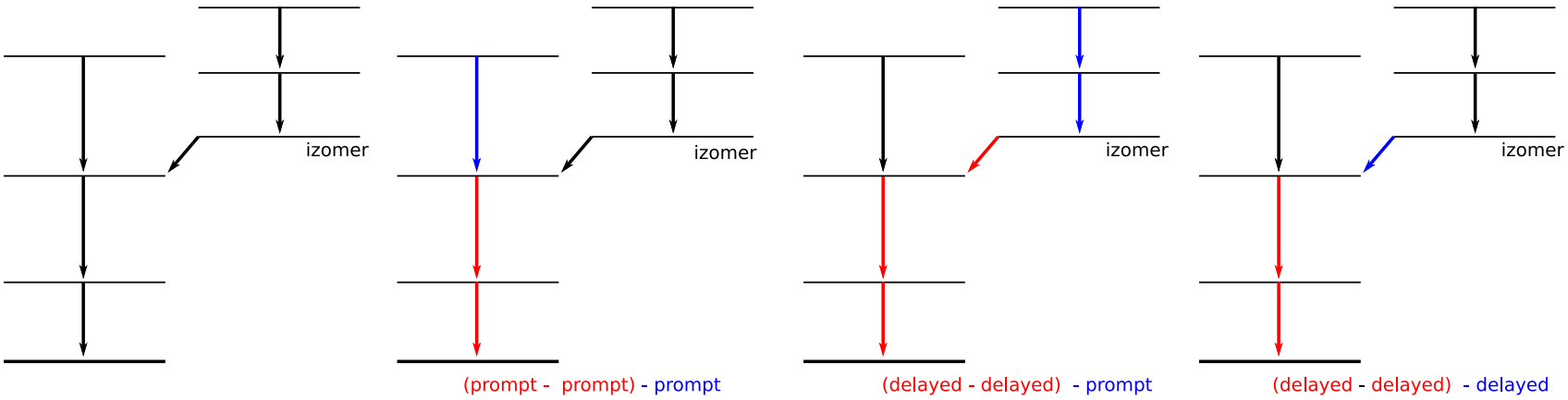


- Macierz $\gamma\gamma\gamma$ z eksperymentu ν-Ball
- Czas tworzenia pliku $\sim$20 dni.
- Stały czas bramkowania/rzutowania (sekundy)

Macierze 3D (A)

Liczba macierzy:

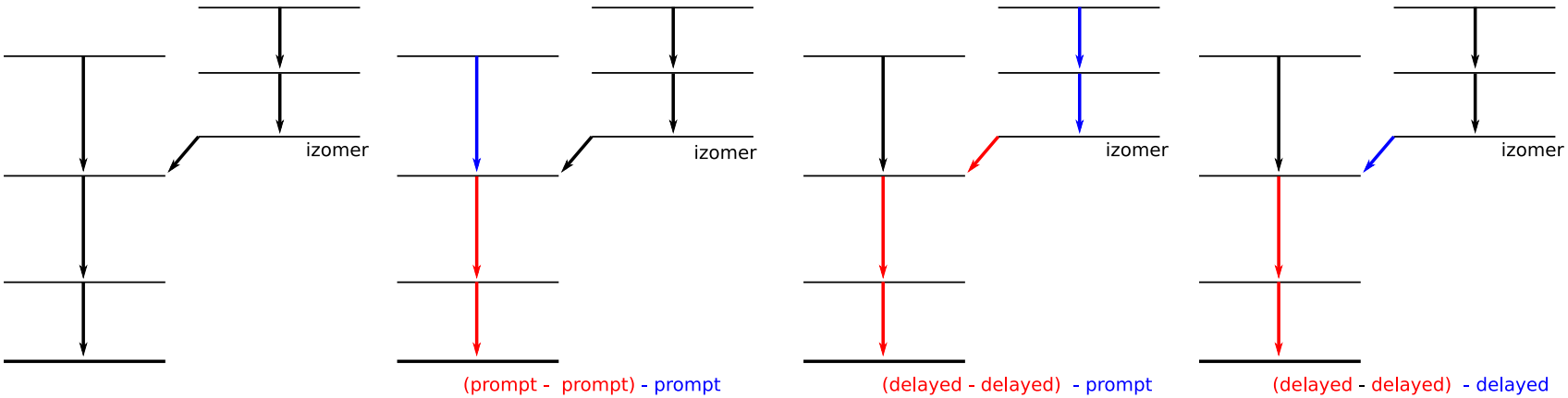
- Typy koincydencji czasowych: 3



Macierze 3D (A)

Liczba macierzy:

- Typy koincydencji czasowych: 3

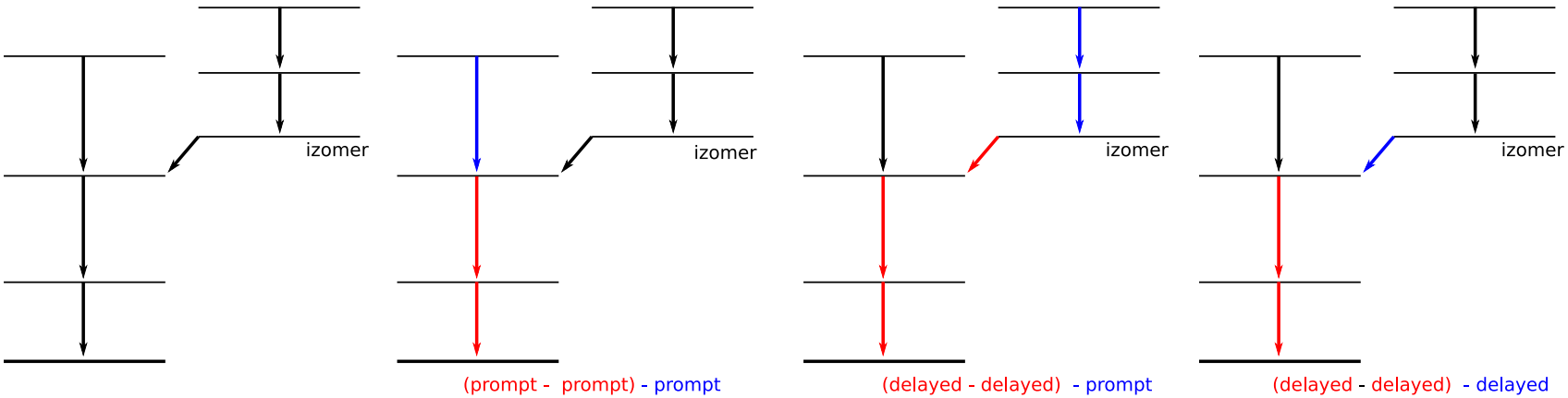


- Typy detektorów (Ge-Ge-Ge, Ge-Ge-La, Ge-La-La): 3

Macierze 3D (A)

Liczba macierzy:

- Typy koincydencji czasowych: 3



- Typy detektorów (Ge-Ge-Ge, Ge-Ge-La, Ge-La-La): 3
- Krotności (wszystkie, >4, >5): 3

Wszystkie kombinacje: $3^3 = 27$ macierzy po 20-30 GB.

Musimy dobrze znać warunki przed tworzeniem macierzy!

Macierze 3D (B)

- Plik z listą zdarzeń $\gamma\gamma\gamma$

Macierze 3D (B)

- Plik z listą zdarzeń $\gamma\gamma\gamma$
- Czas tworzenia pliku $\sim$2 dni

Macierze 3D (B)

- Plik z listą zdarzeń $\gamma\gamma\gamma$
- Czas tworzenia pliku $\sim$2 dni
- Czas bramkowania $\sim$10 minut

Macierze 3D (B)

- Plik z listą zdarzeń $\gamma\gamma\gamma$
- Czas tworzenia pliku $\sim$2 dni
- Czas bramkowania $\sim$10 minut
- Struktura danych: E0, E1, E2, T0, T1, T2, Hit pattern

Macierze 3D (B)

- Plik z listą zdarzeń $\gamma\gamma\gamma$
- Czas tworzenia pliku $\sim$2 dni
- Czas bramkowania $\sim$10 minut
- Struktura danych: E0, E1, E2, T0, T1, T2, Hit pattern
- Energia w *10 eV*, czas w *ps*, Hit pattern to 3-bitowa liczba (Ge = 1, LaBr = 0)

Macierze 3D (B)

- Plik z listą zdarzeń $\gamma\gamma\gamma$
- Czas tworzenia pliku $\sim$2 dni
- Czas bramkowania $\sim$10 minut
- Struktura danych: E0, E1, E2, T0, T1, T2, Hit pattern
- Energia w *10 eV*, czas w *ps*, Hit pattern to 3-bitowa liczba (Ge = 1, LaBr = 0)
- Zdarzenia są podzielone na grupy o kolejnych krotnościach (3, 4, 5, . . .)

```
GROUP "/" {
   GROUP "GeGeGe" {
      DATASET "3" {
         DATATYPE  H5T_STD_U32LE
         DATASPACE  SIMPLE { ( 414983607, 7 ) / ( H5S_UNLIMITED, 7 ) }
         DATA {
         (0,0): 27470, 33193, 14813, 34768, 42624, 72480, 7,
         (1,0): 27470, 33193, 8729, 34768, 42624, 1037712, 7,
         (2,0): 27470, 14813, 8729, 34768, 72480, 1037712, 7,
            ...
            }
        }
    }
}
```

Dane ALTO

Przykład bramkowania danych - Python

```
(...)

dataset = group['GeGeGe/{}'.format(m)]
submatrix = numpy.array(dataset[left_pos:right_pos, :])

sel = submatrix[submatrix[:, 0] >= gate_z[0] * E_unit]
sel = sel[sel[:, 0] < gate_z[1] * E_unit]
sel = sel[sel[:, 1] >= gate_y[0] * E_unit]
sel = sel[sel[:, 1] < gate_y[1] * E_unit]
gg, edges = numpy.histogram( sel[:, 2] / E_unit, bins=D, range=[0, D])

(...)
```

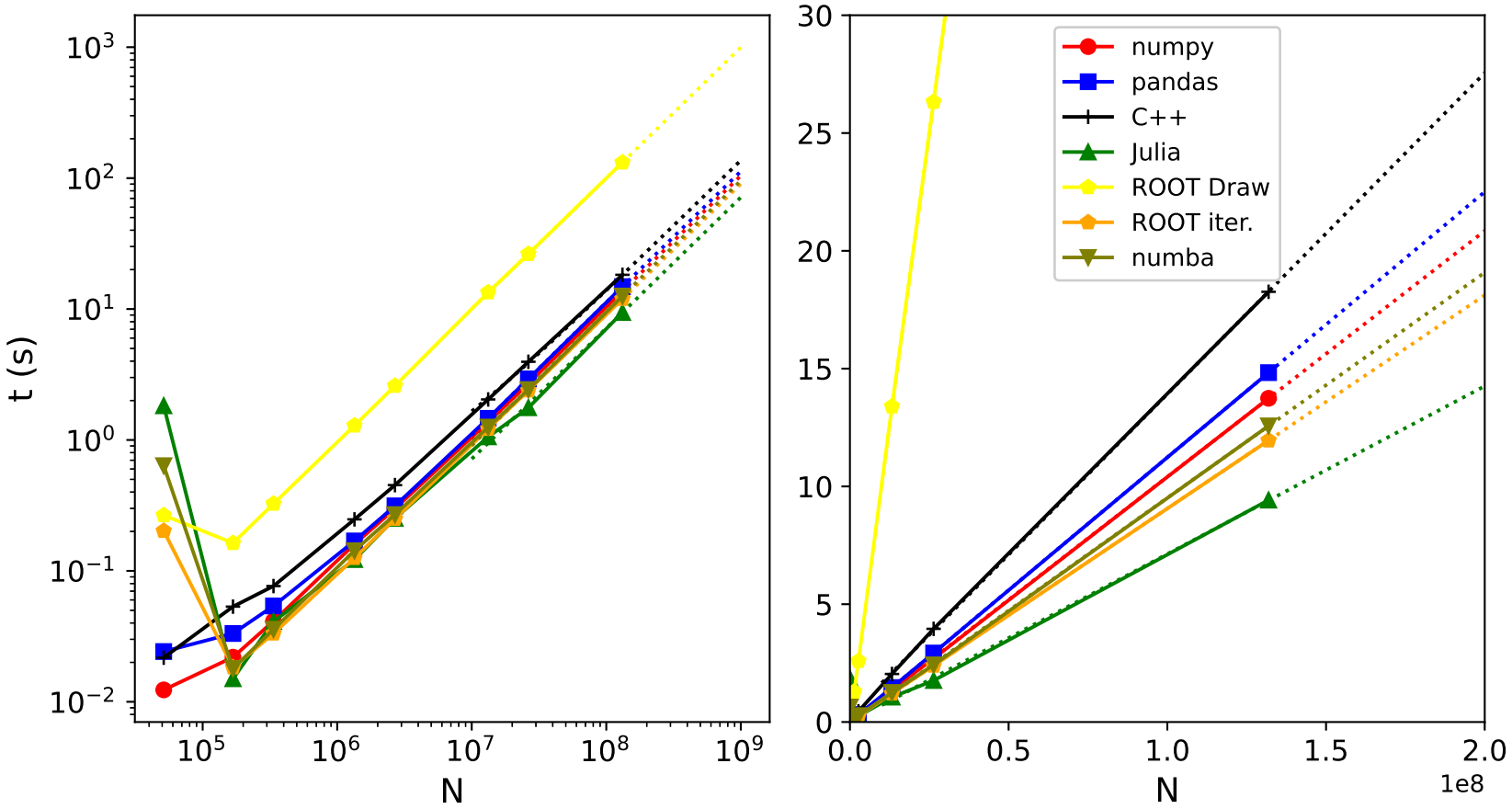
Przykład bramkowania danych - Julia

```
data = dataset[:, left_pos:right_pos]

df = DataFrame(data',
            ["E0", "E1", "E2", "t0", "t1", "t2", "pattern"])

sub = df[(gate_y[1] .<= df[:, 2] .< gate_y[2])
    .& (gate_z[1] .<= df[:, 1] .< gate_z[2]), :]
h = fit(Histogram, sub[:, 3], edges)
matrix += h.weights
```

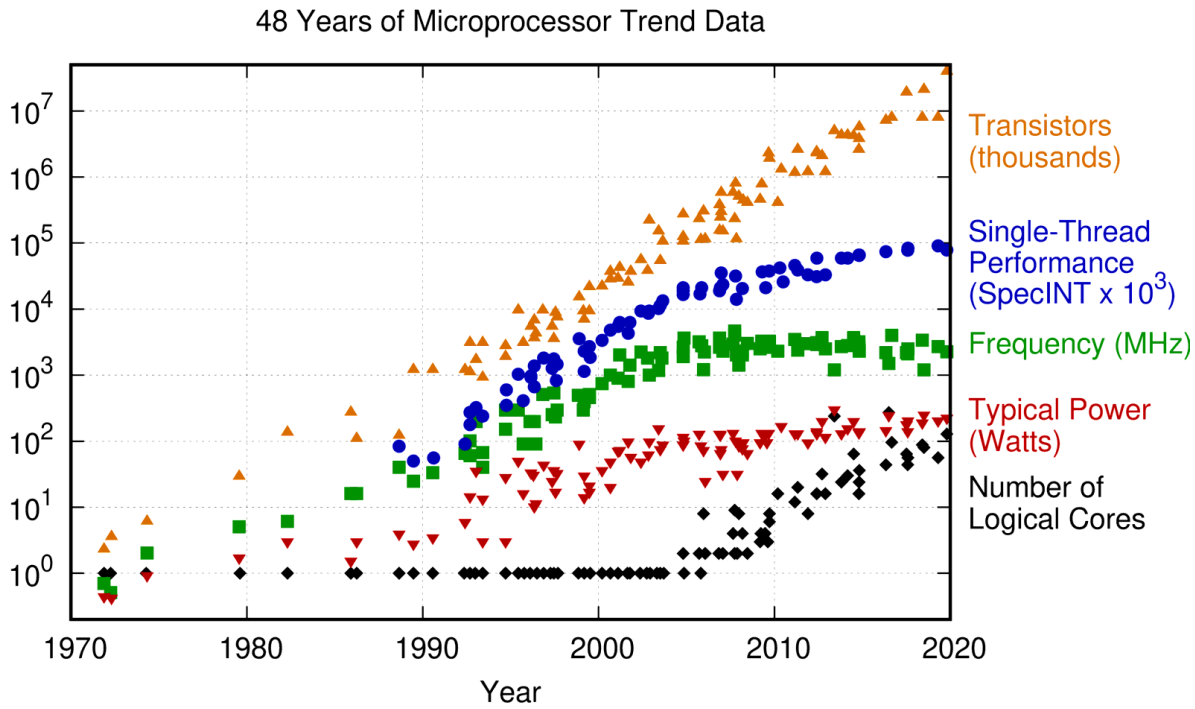
Test wydajności

Zadanie testowe dla kodów

- Odczytać plik z listą zdarzeń we własnym formacie (.bin, .root, .h5)
- Znaleźć zdarzenia dla każdego z 16 detektorów
- Stworzyć histogram (widmo energii)

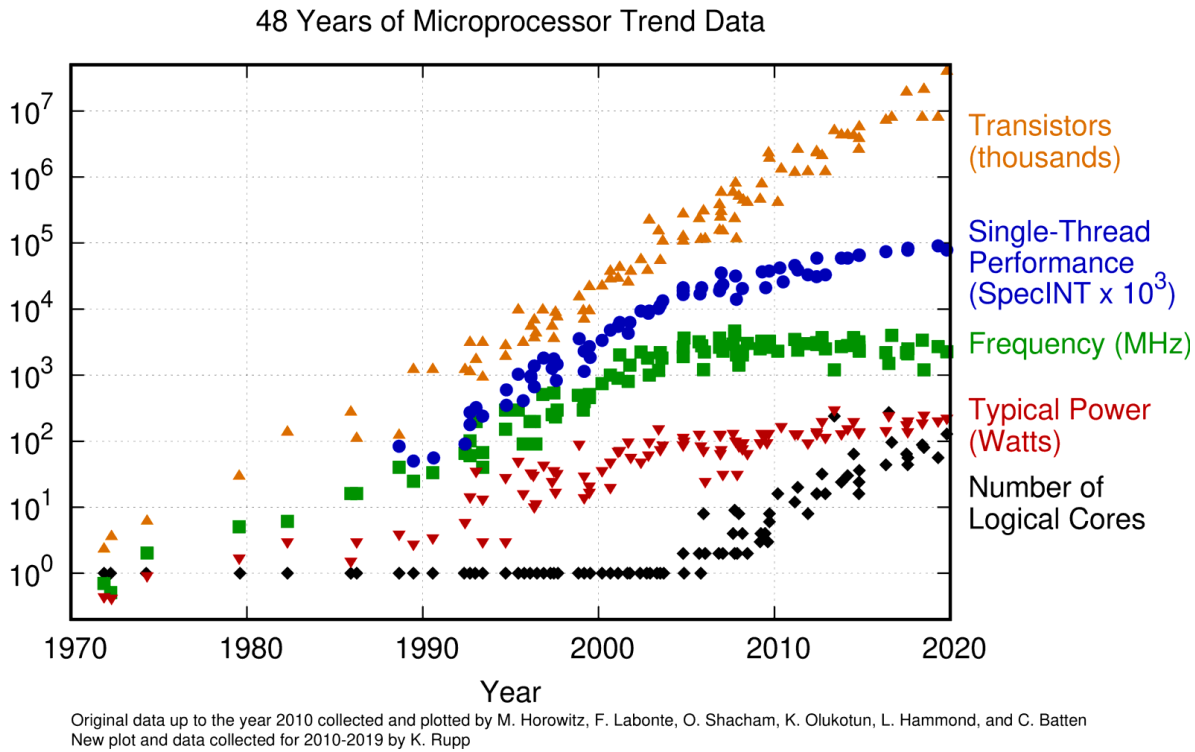


Aktualne trendy



Original data up to the year 2010 collected and plotted by M. Horowitz, F. Labonte, O. Shacham, K. Olukotun, L. Hammond, and C. Batten
New plot and data collected for 2010-2019 by K. Rupp

Aktualne trendy



- Przetwarzanie wielowątkowe
- Kompilacja Just-In-Time (JIT)

Wielowątkowość

Wielowątkowość

Przetwarzanie wielowątkowe

- Wielowątkowość współbieżna (multithreading) - wykorzystanie jednego fizycznego rdzenia, ale wykorzystanie możliwości jednoczesnego wykonywania różnych operacji przez CPU

Wielowątkowość

Przetwarzanie wielowątkowe

- Wielowątkowość współbieżna (multithreading) - wykorzystanie jednego fizycznego rdzenia, ale wykorzystanie możliwości jednoczesnego wykonywania różnych operacji przez CPU
  - \+ Wspólna pamięć, szybki dostęp
  - \+ GUI
  - \+ Operacje I/O
  - \- Użycie tylko jednego CPU

Wielowątkowość

Przetwarzanie wielowątkowe

- Wielowątkowość współbieżna (multithreading) - wykorzystanie jednego fizycznego rdzenia, ale wykorzystanie możliwości jednoczesnego wykonywania różnych operacji przez CPU
  - \+ Wspólna pamięć, szybki dostęp
  - \+ GUI
  - \+ Operacje I/O
  - \- Użycie tylko jednego CPU
- Wieloprocesorowość (multiprocessing) - wykorzystanie więcej niż jednego fizycznego CPU

Wielowątkowość

Przetwarzanie wielowątkowe

- Wielowątkowość współbieżna (multithreading) - wykorzystanie jednego fizycznego rdzenia, ale wykorzystanie możliwości jednoczesnego wykonywania różnych operacji przez CPU
  - \+ Wspólna pamięć, szybki dostęp
  - \+ GUI
  - \+ Operacje I/O
  - \- Użycie tylko jednego CPU
- Wieloprocesorowość (multiprocessing) - wykorzystanie więcej niż jednego fizycznego CPU
  - \+ Wiele CPU
  - \- Kosztowna operacja tworzenia nowego procesu
  - \- Oddzielne przestrzenie pamięci, konieczność komunikacji

Wielowątkowość

Przetwarzanie wielowątkowe

- Wielowątkowość współbieżna (multithreading) - wykorzystanie jednego fizycznego rdzenia, ale wykorzystanie możliwości jednoczesnego wykonywania różnych operacji przez CPU
  - \+ Wspólna pamięć, szybki dostęp
  - \+ GUI
  - \+ Operacje I/O
  - \- Użycie tylko jednego CPU
- Wieloprocesorowość (multiprocessing) - wykorzystanie więcej niż jednego fizycznego CPU
  - \+ Wiele CPU
  - \- Kosztowna operacja tworzenia nowego procesu
  - \- Oddzielne przestrzenie pamięci, konieczność komunikacji
- Większość języków programowania wspiera przetwarzanie wielowątkowe, ale na różnych poziomach

Wielowątkowość - Python

- Python - niski poziom: *multithreading, multiprocessing*

```
class Decoder(multiprocessing.Process):

    def __init__(self, ...)):
        (...)
    def run(self):
        (...)

data_queue = multiprocessing.Queue()
decoders = []
for i in n:
    decoders.append(Decoder(data_queue))
    decoders[-1].start()
```

Wielowątkowość - Python

- Python - niski poziom: *multithreading, multiprocessing*

```
class Decoder(multiprocessing.Process):

    def __init__(self, ...)):
        (...)
    def run(self):
        (...)

data_queue = multiprocessing.Queue()
decoders = []
for i in n:
    decoders.append(Decoder(data_queue))
    decoders[-1].start()
```

- Python - Wysoki poziom - *numba*

```
@jit(parallel=True)
def test(x):
    T = 0
    n = x.shape[0]
    for i in prange(n):
        if x[i] > 0.5:
            T += 1
    return T / n
```

Wielowątkowość - Julia

- Julia - niski poziom

```
module Test
export process
(...)
end

addprocs(p)
@everywhere include("./Test.jl")
w = []
for i in 1:p
    push!(w, @spawnat :any Test.process(k))
end
```

Wielowątkowość - Julia

- Julia - niski poziom

```
module Test
export process
(...)
end

addprocs(p)
@everywhere include("./Test.jl")
w = []
for i in 1:p
    push!(w, @spawnat :any Test.process(k))
end
```

- Julia - wysoki poziom
  - Programowanie wielowątkowe

```
T = 0
@thread for i in 1:n
    if x[i] > 0.5
        T += 1
end
```

Wielowątkowość - Julia

- Julia - niski poziom

```
module Test
export process
(...)
end

addprocs(p)
@everywhere include("./Test.jl")
w = []
for i in 1:p
    push!(w, @spawnat :any Test.process(k))
end
```

- Julia - wysoki poziom
  - Programowanie wielowątkowe

```
T = 0
@thread for i in 1:n
    if x[i] > 0.5
        T += 1
end
```

  - Programowanie rozproszone i wieloprocesorowe

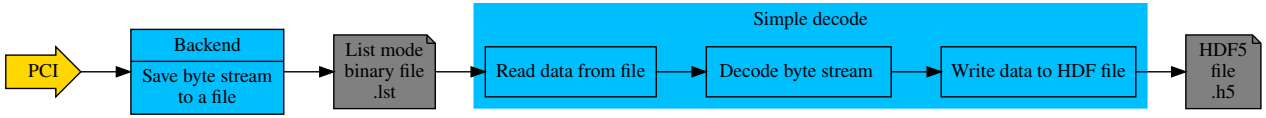
```
T = @distributed (+) for i = 1:n
    if x[i] > 0.5
        1
    end
end
```

Wielowątkowość - przykład

- Program do odkodowywania formatu z elektroniki cyfrowej Pixie

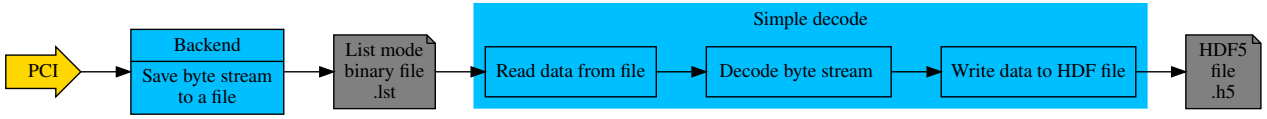
Wielowątkowość - przykład

- Program do odkodowywania formatu z elektroniki cyfrowej Pixie
- Wersja jednowątkowa

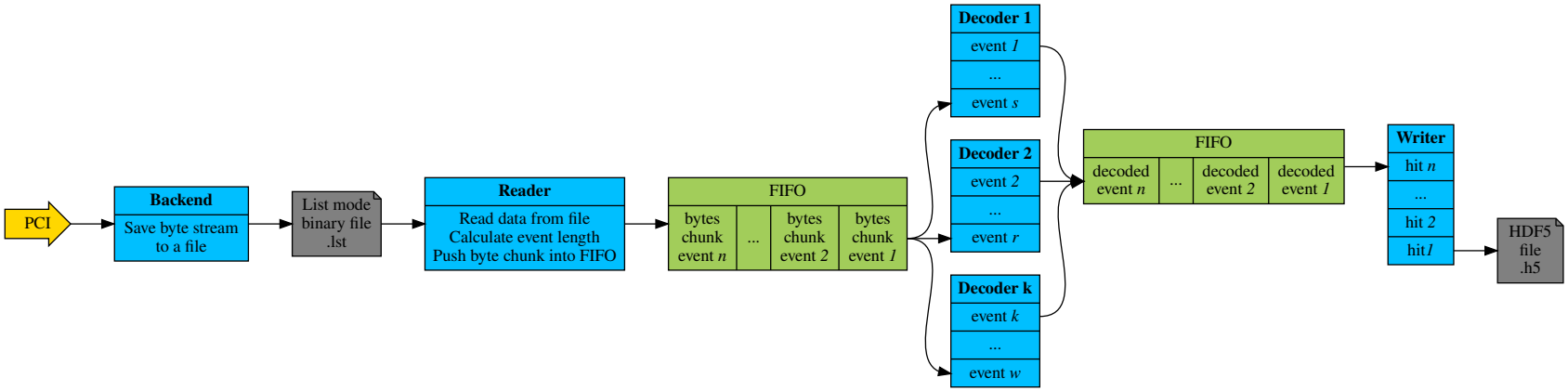


Wielowątkowość - przykład

- Program do odkodowywania formatu z elektroniki cyfrowej Pixie
- Wersja jednowątkowa

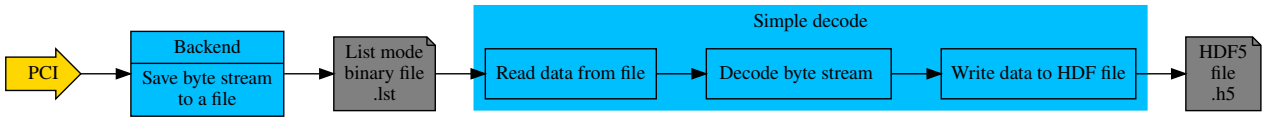


- Wersja wieloprocesorowa

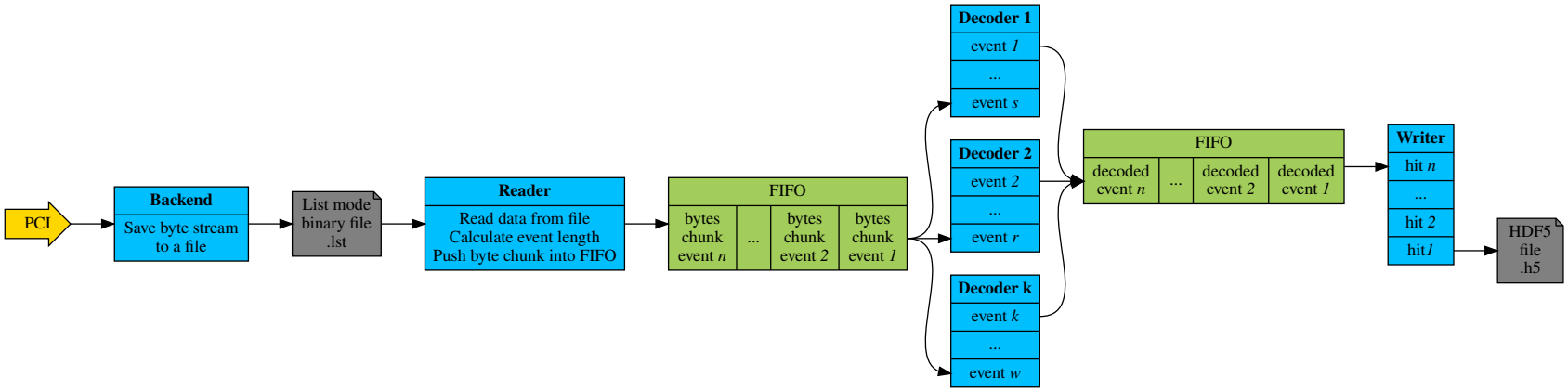


Wielowątkowość - przykład

- Program do odkodowywania formatu z elektroniki cyfrowej Pixie
- Wersja jednowątkowa



- Wersja wieloprocesorowa



- Wyniki

Wersja	Analiza tylko on-board	Z analizą sygnałów
python (1)	130 kHz	26 kHz
python - mp (2)	22 kHz	3 kHz
python - mp (4)	19 kHz	4.5 kHz
julia (1)	460 kHz	420 kHz

Just-in-time

Just-in-time compilation

Just-in-time

Just-in-time compilation

- Kompilacja podczas wykonywania programu (zamiast przed), interpreter lub maszyna wirtualna śledzi kod i kompiluje odpowiednie fragmenty
- Dzięki temu otrzymujemy zalety języków interpretowanych (dynamiczność) oraz kompilowanych (szybkość)
- Ceną jest czas rozgrzewki (kompilacji)

Just-in-time

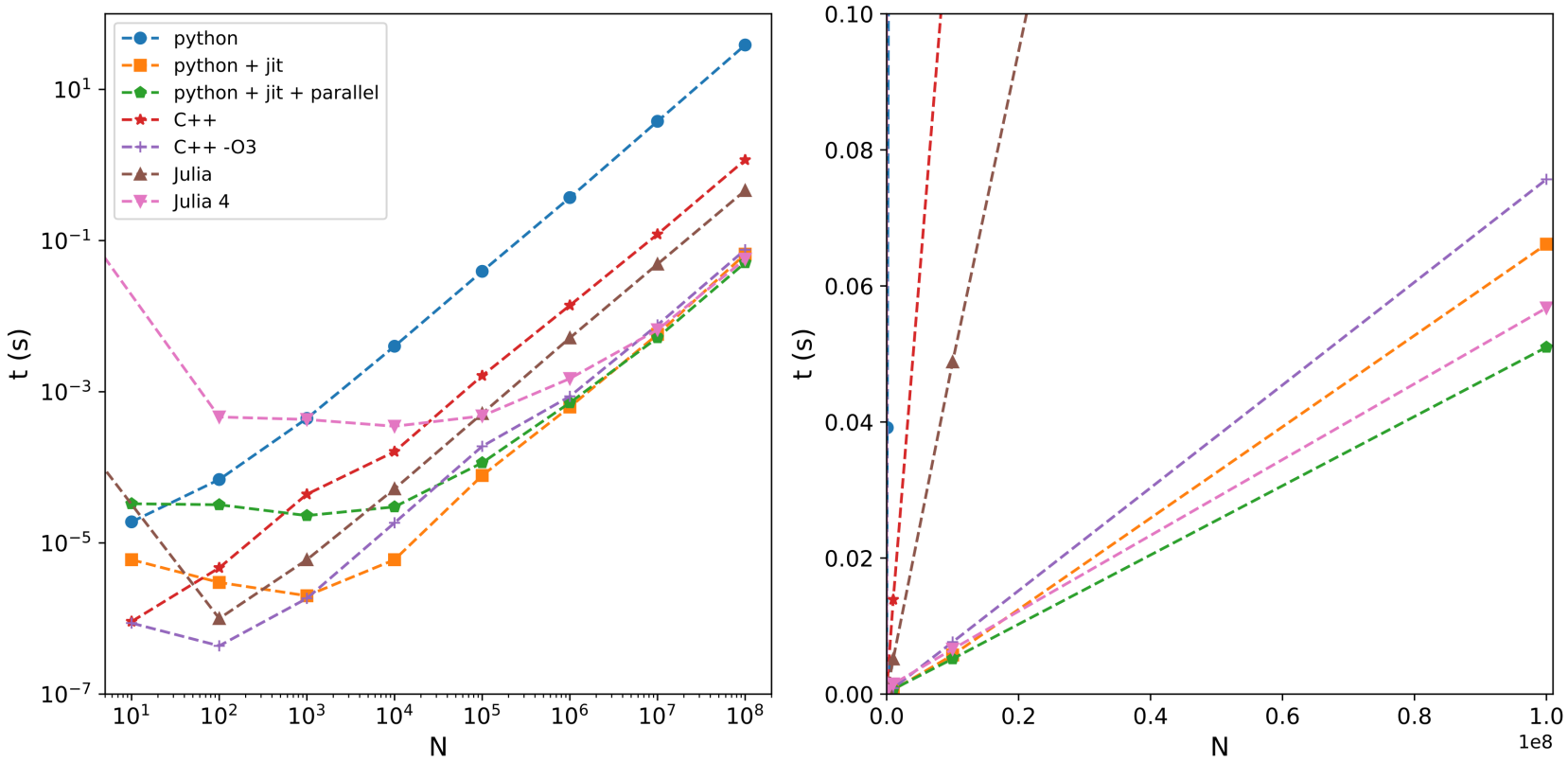
Just-in-time compilation

- Kompilacja podczas wykonywania programu (zamiast przed), interpreter lub maszyna wirtualna śledzi kod i kompiluje odpowiednie fragmenty
- Dzięki temu otrzymujemy zalety języków interpretowanych (dynamiczność) oraz kompilowanych (szybkość)
- Ceną jest czas rozgrzewki (kompilacji)
- Python - numba

```
from numba import jit, prange

@jit
def test(x):
    T = 0
    n = x.shape[0]
    for i in prange(n):
        if x[i] > 0.5:
            T += 1
    return T / n
```

Test wydajności

Zadanie: w losowym wektorze N elementowym policz ile jest wyrazów > 0.5



Podsumowanie

- Wraz z rosnącymi możliwościami komputerów mamy do dyspozycji coraz większe możliwości obliczeniowe
- Ich wykorzystanie przekracza możliwości pisania kodów od podstaw
- Warto sięgać po sprawdzone języki programowania oraz biblioteki
- Fizyka wysokich energii i jądrowa to nie jedyne dziedziny przetwarzające duże zbiory danych
- Dobrze jest przyglądać się postępom w innych dziedzinach i aktualnym trendom w informatyce
- Wykorzystanie popularnych narzędzi pozwala zaoszczędzić czas, a naszym współpracownikom (studenci, doktoranci, post-doc) zyskać umiejętności cenne poza naszą dziedziną