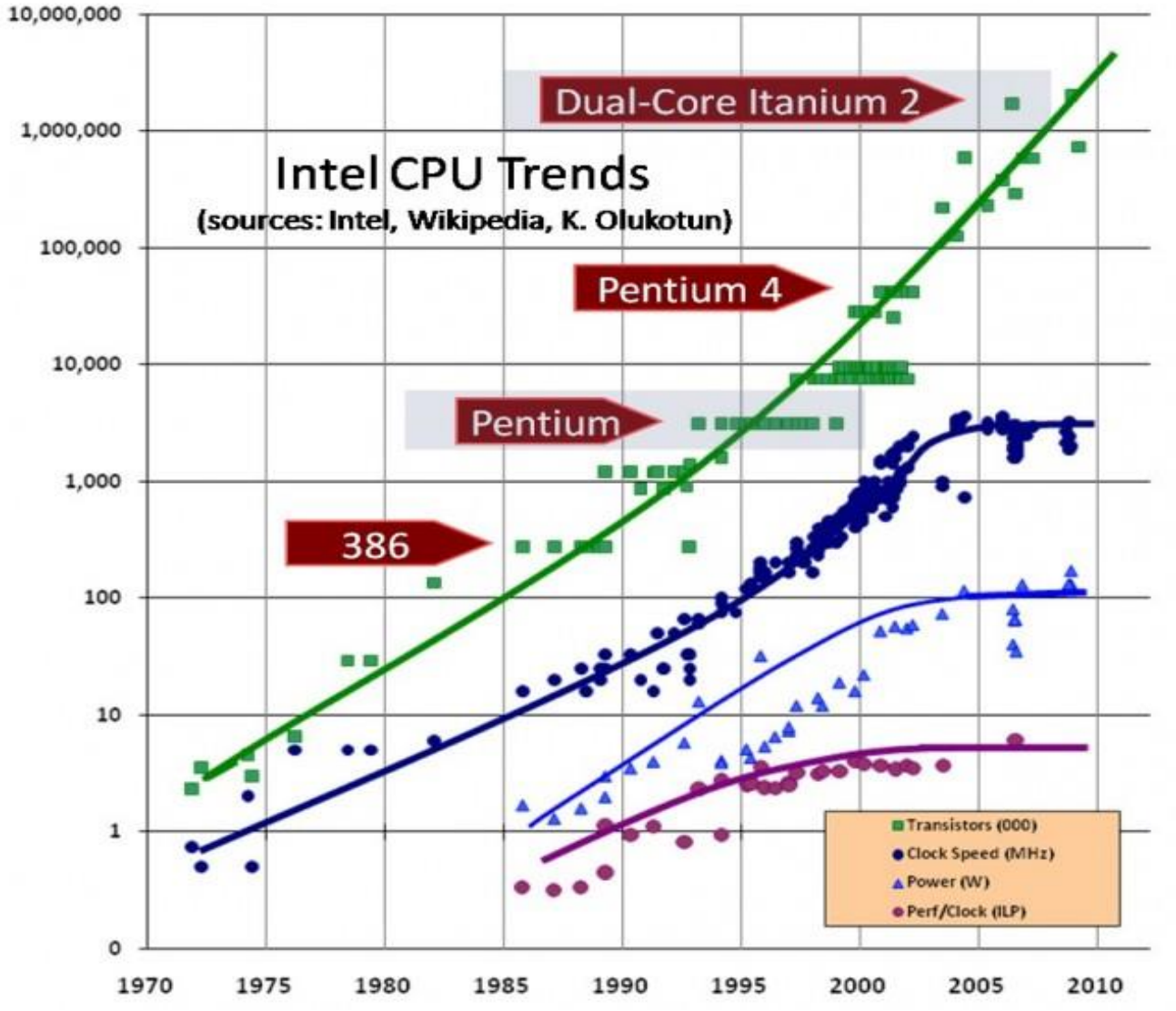
Jak ujarzmić hydrę
czyli
programowanie równoległe w Javie

dr hab. Piotr Bała, prof. UW
ICM Uniwersytet Warszawski

Prawo Moore’a

- Ekonomicznie optymalna liczba tranzystorów w układzie scalonym zwiększa się w kolejnych latach zgodnie z trendem wykładniczym (podwaja się w niemal równych odcinkach czasu).
- Autorstwo tego prawa przypisuje się Gordonowi Moore'owi, jednemu z założycieli firmy Intel - 1965 r.
- Prawo Moore’a zostało rozszerzone na inne parametry komputerów (np. szybkość)

Prawo Moore'a



Współczesne komputery

- Thiane-2
- 3 200 000 korów (jednostek obliczeniowych)

- Hydra
- 5 200 korów

Przetwarzanie równoległe

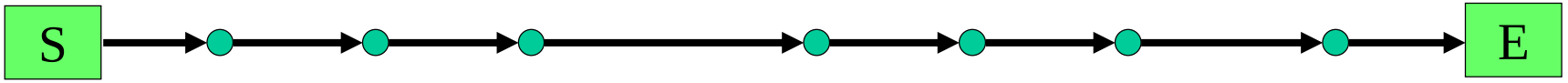
- Robotnik może wykopać 20m rowu w ciągu 10 godzin.
- W jakim czasie ten rów wykona 10 robotników?

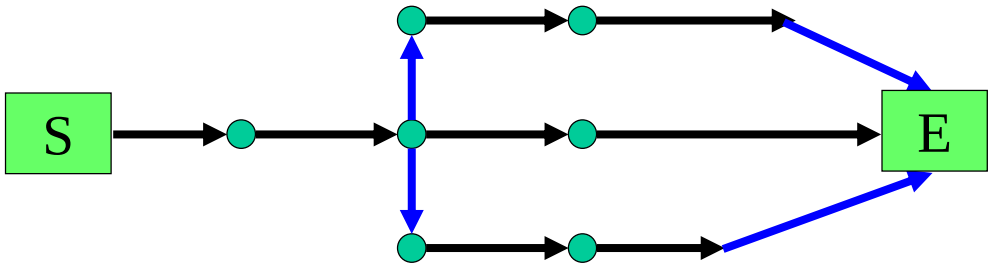
- 10 godz : 10 = 1 godz

- Co się stanie jeżeli do wykopania mamy studnię o głębokości 10 m?

Komputer równoległy

Klasyczny komputer Turinga



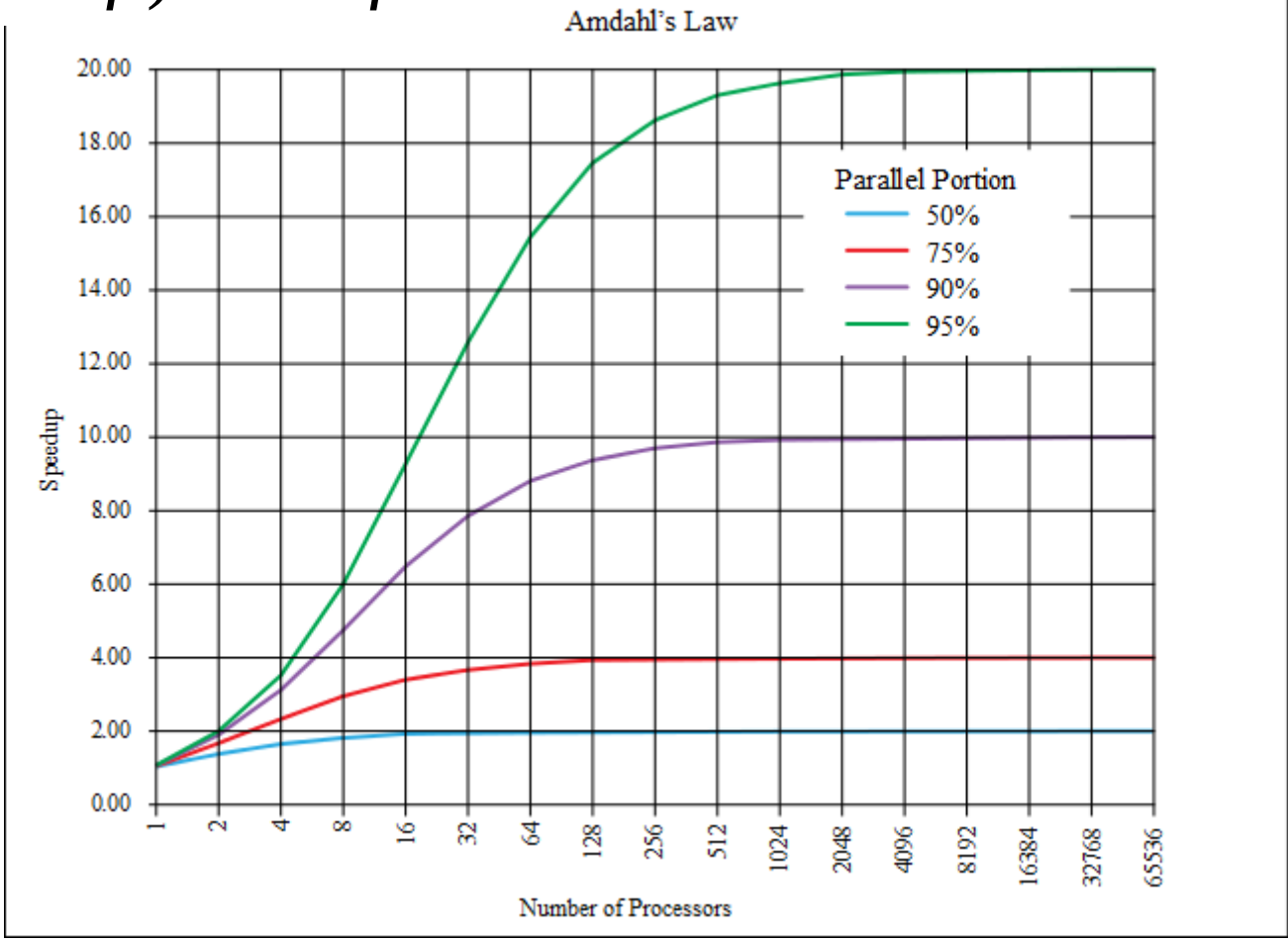


Komputer równoległy

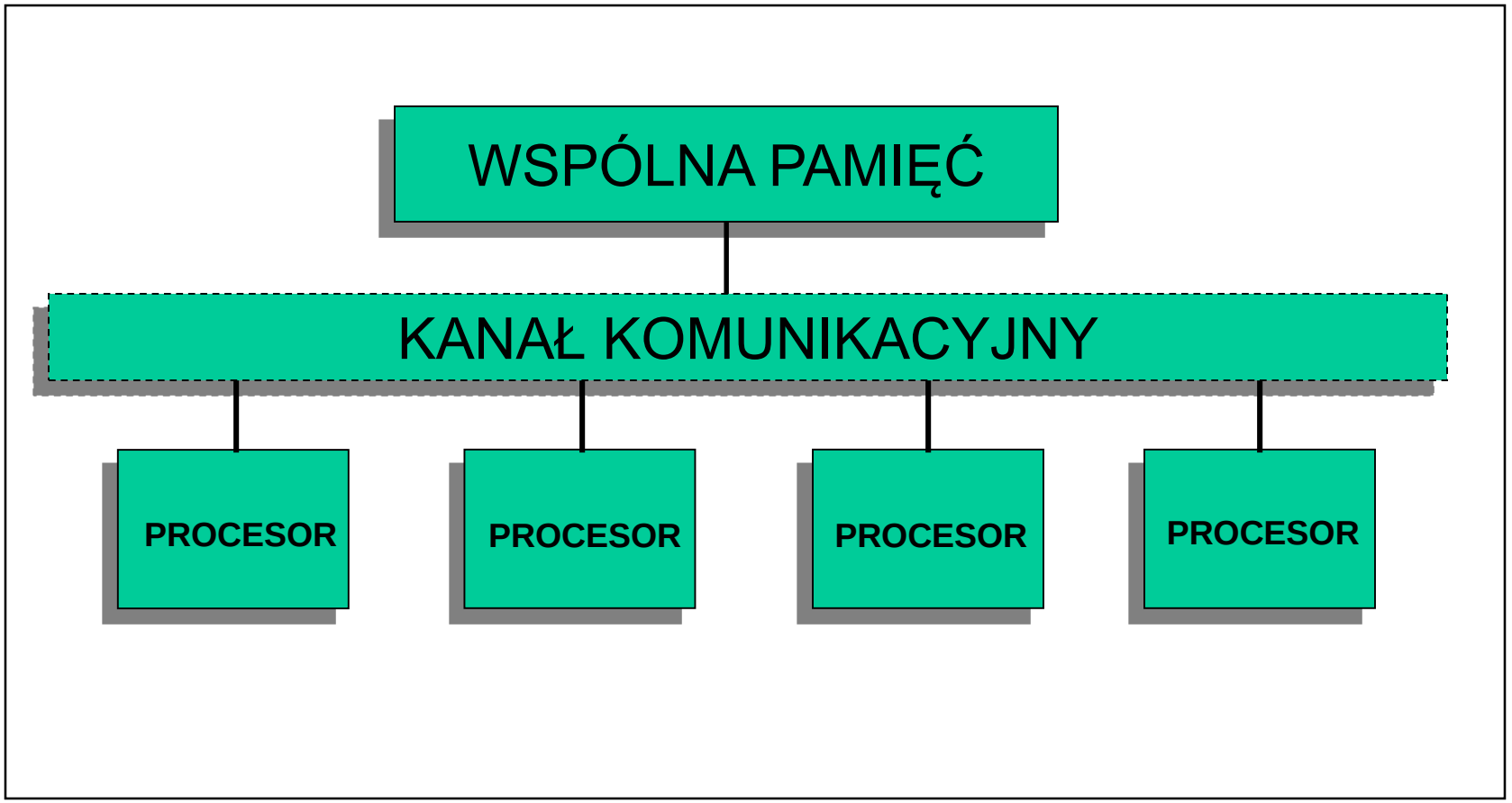
Prawo Amdahl'a

$$\psi \equiv S(p) \leq \frac{1}{f + (1-f)/n} \leq \frac{1}{f}$$

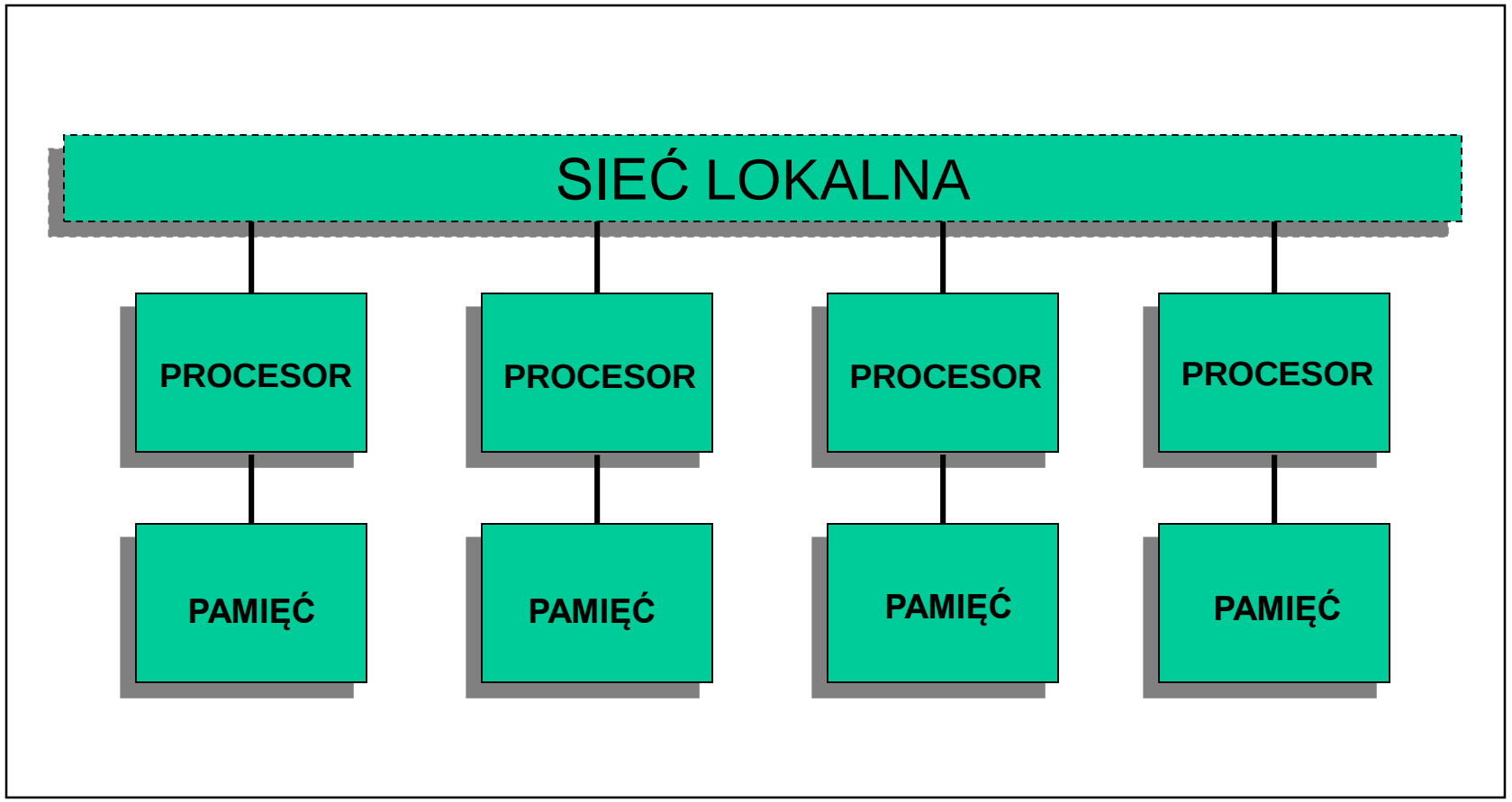
S(p) - przyspieszenie
f – część sekwencyjna programu
n – liczba procesorów (korów)



System z pamięcią wspólną



System z pamięcią rozproszoną



Programowanie równoległe

- Współczesne komputery zbudowane są z dziesiątek, tysięcy czy setek tysięcy procesorów.
- Procesory posiadają wiele jednostek obliczeniowych (korów).
- Karty graficzne i inne koprocesory mają dziesiątki i setki jednostek obliczeniowych.
- Programowanie równoległe jest kluczowe dla współczesnych zastosowań.
- Oprogramowanie na komputery dużej mocy jest pisane głownie w C i FORTRANie.
- Programowanie równoległe wykorzystuje tradycyjne modele programowania: MPI i OpenMP.

MPI - Message-Passing Interface

- Historia:
  - MPI-1.0 (1992),
  - MPI-2.0 (1997),
  - MPI-3.0 (2012)
- C, C++, Fortran, Java (wrapery, JNI), Python, . . .
- Duża liczba parametrów wywołania.
- Komunikacja na zasadzie wysyłania komunikatów.
- Łatwo wpaść w zakleszczenia, które są trudne do debugowania.

OpenMP – Open Multi-Processing

- Historia:
  - 1.0 (1997),
  - 2.5 (2005),
  - 4.0 (2013)
- Dyrektywy kompilatora:

  #pragma omp <directive> [clause]

- Zaprojektowany do pracy na komputerach o pamięci wspólnej.
- Zrównoleglanie oparte na zrównoleglaniu pojedynczych pętli
  - Wprowadza dodatkową synchronizację
  - Trudne zarządzanie rozmieszczeniem danych w pamięci

Noew podejście do programowania równoległego

- Potrzebne są nowe rozwiązania: algorytmy równoległe
- Potrzebne nowe sposoby programowania (języki lub sposoby wyrażania równoległości)
- Nowe języki programowania:
  - X10 (IBM)
  - Click (Intel)
  - XscalableMP (Fujitsu, RIKEN)
- Nowy model programowania równoległego: PGAS: Partitioned Global Address Space

PGAS – Partitioned Global Address Space

Podstawy:

- Dane lokalne (tylko dla danego procesora) i współdzielone
- Zmienne globalne – widoczne dla wszystkich procesorów
- Programista nie zajmuje się szczegółami komunikacji
- Jednostronna komunikacja

Podstawowo operacje:

- synchronizacja (bariera)
- get
  - procesor pobiera dane z procesora j
- put
  - procesor wysyła dane do procesora j

Języki PGAS

- http://www.pgas.org

- Co-Array Fortran (CAF)
- Unified Parallel C (UPC)
- Titanium *(Java dialect)*
- X10
- Chapel
- Fortress
- PCJ

PCJ - Parallel Computations in Java

Biblioteka dla Javy rozwijana w ICM

- **pcj.icm.edu.pl**

Paradygmat:

- partitioned global address space (PGAS)

Podstawowe własności

- Nie wymaga modyfikacji JVM (wirtualnej maszyny Javy).
- Dostępna dla wszystkich systemów z Java7
  - *np. IBM Java 1.7 dla Power7*
- Wykorzystuje Java SE 7 (NIO, SDP, . . .)
- Pracuje z Java SE 8, Java SE 9
- Nie wymaga dodatkowych bibliotek.

PCJ – Parallel Computations in Java

Podstawowa funkcjonalność:

- Synchronizacja wątków
- Pobieranie danych (get)
- Wysyłanie danych (put)

Dodatkowe funkcjonalności:

- Rozgłaszanie zmiennych
- Monitorowanie zmiany wartości zmiennych
- Równoległe I/O
- Grupy wątków

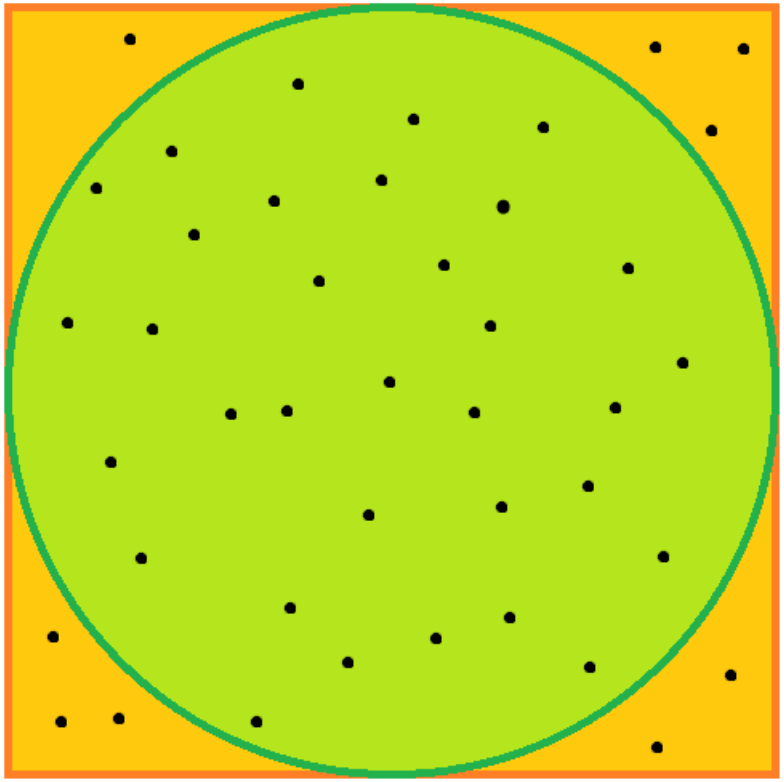
PCJ – Hello world

```
import org.pcj.*
public class PcjHelloWorld extends Storage
                                    implements StartPoint {

    @Override
    public void main() {
        System.out.println("Hello!");
    }

    public static void main(String[] args) {
        String[] nodes = new String[]{"localhost", "localhost"};
        PCJ.deploy(PcjHelloWorld.class,
                        PcjHelloWorld.class, nodes);
    }
}
```

PCJ - podstawy

```
double c;
if (PCJ.myId()==0) c =(double) PCJ.get(3, "a");

if (PCJ.myId()==0) PCJ.put(3, "a", 5.0);

if (PCJ.myId()==0) PCJ.broadcast("a", 5.0);

public static void PCJ.barrier();

public static int PCJ.threadCount()
```

PCJ – Wyznaczanie π metodą Monte Carlo



$$\pi \approx \frac{4 \times \text{inCirclePoints}}{\text{totalPoints}}$$

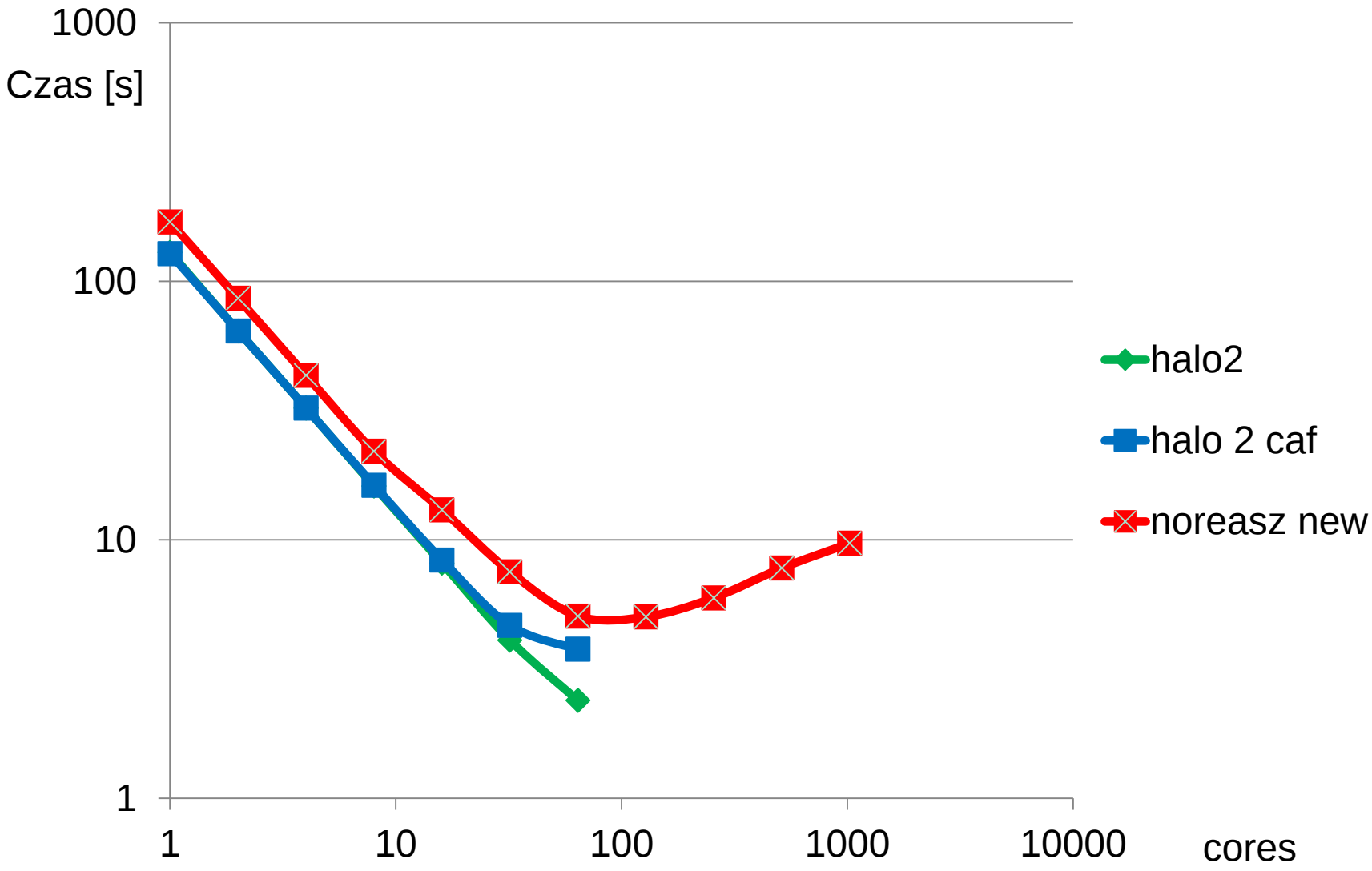
PCJ – Wyznaczanie π metodą Monte Carlo

```
Random random = new Random ();
long nAll = 1 _280_000_000 ;
long n = nAll / PCJ.threadCount ();
long myCircleCount = 0;
for ( long i = 0; i < n; ++i) {
   double x = 2.0 * random.nextDouble () - 1.0;
   double y = 2.0 * random.nextDouble () - 1.0;
   if ((x * x + y * y) <= 1.0) {
        myCircleCount ++;
    }
}
PCJ.putLocal ("count", myCircleCount );
PCJ.barrier ();
```

PCJ – Redukcja (sumowanie częściowych wyników z różnych procesorów)

```
@Shared double a
FutureObject aL[] = new FutureObject[PCJ.threadCount()];
double a0 = 0.0;
  if (PCJ.myId() == 0) {
       for (int p = 0; p < PCJ.threadCount(); p++) {
           aL[p] = PCJ.getFutureObject(p, "a");
       }
       for (int p = 0; p < PCJ.threadCount(); p++) {
           a0 = a0 + (double) aL[p].get();
         }
      }
```

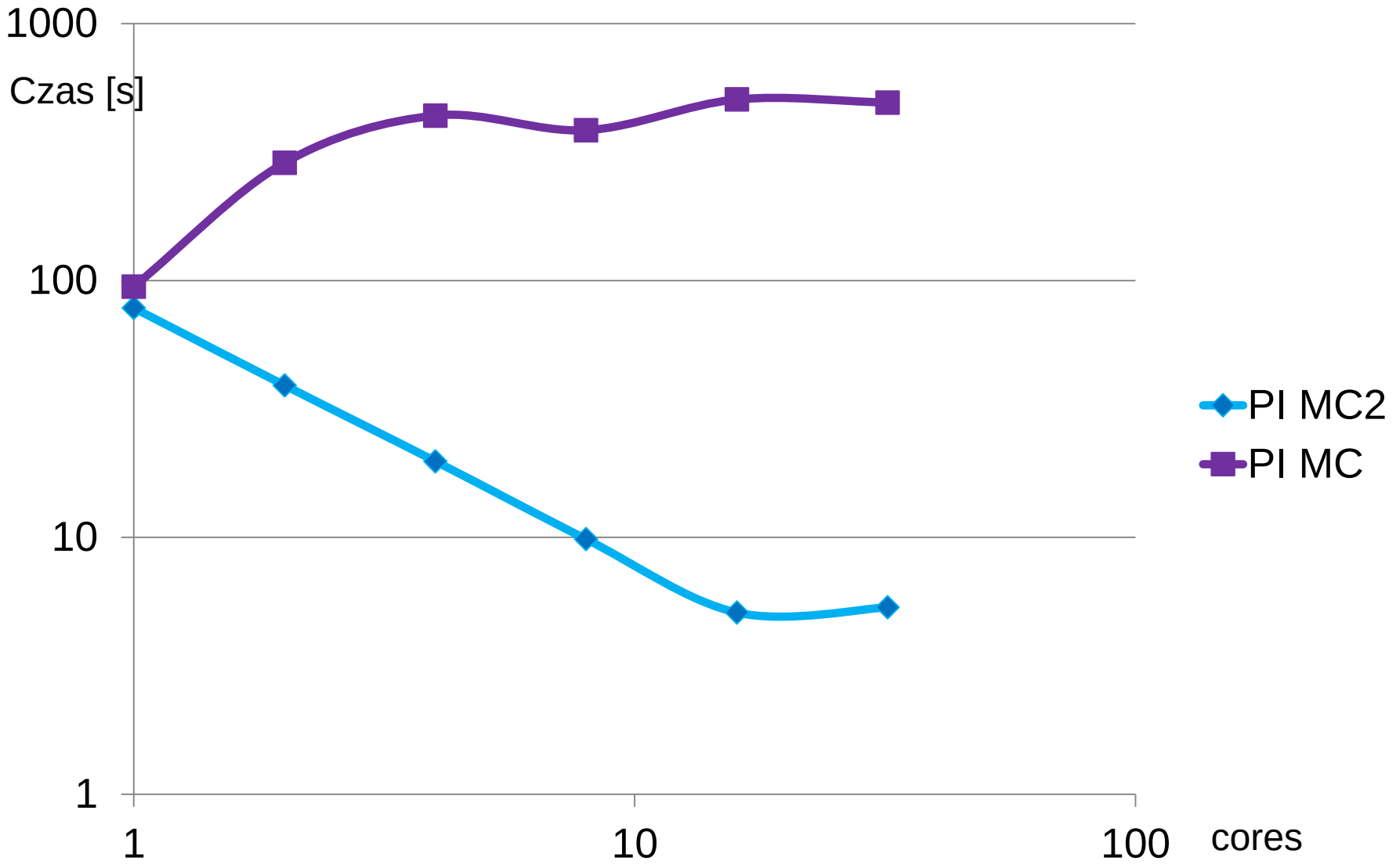
PCJ - Wyznaczanie π metodą Monte Carlo



PCJ – Wyznaczanie π metodą Monte Carlo (Math.random() zamiast new Random)

```
Random random = new Random ();
long nAll = 1 _280_000_000 ;
long n = nAll / PCJ . threadCount ();
long myCircleCount = 0;
for ( long i = 0; i < n; ++i) {
   double x = 2.0 * Math.random() - 1.0;
   double y = 2.0 * Math.random() - 1.0;
   if ((x * x + y * y) <= 1.0) {
        myCircleCount ++;
    }
}
PCJ.putLocal ("count", myCircleCount );
PCJ.barrier ();
```

PCJ – PI Calculations using Monte Carlo



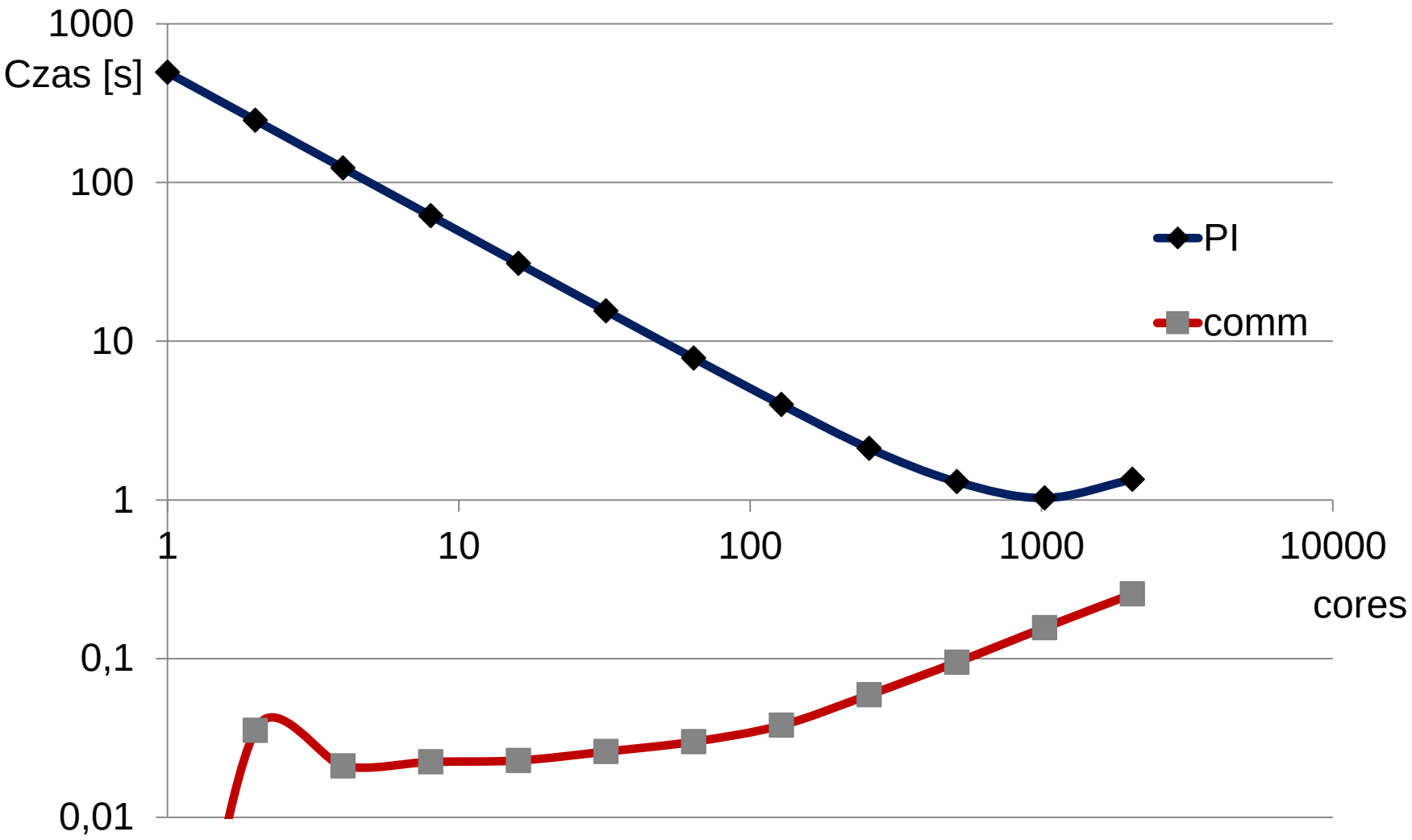
PCJ – PI Calculations using integral

- Przybliżenie wartości π za pomocą sumy (przybkiżającej całkę, czyli pole pok krzywą)

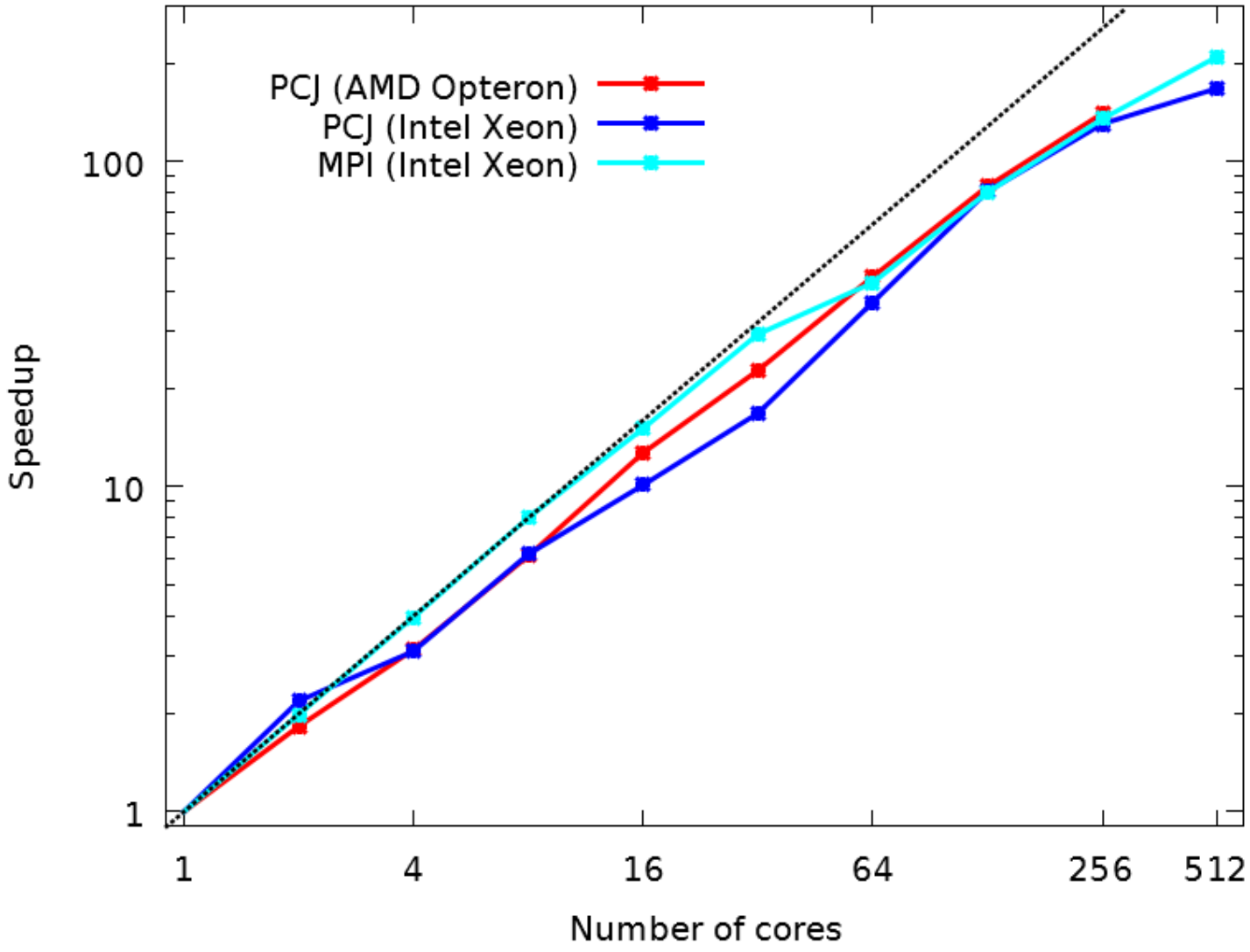
$$\pi = \int_0^1 \frac{4}{1+x^2}\,\mathrm{d}x \approx \sum_{i=1}^{N} \frac{4}{1+\left(\frac{i-\frac{1}{2}}{N}\right)^2}$$

- Sumowanie rozdzielone pomiędzy wątki.
- Każdy z wątków wyznacza swoją część sumy.
- Redukcja – posumowanie wyników częściowych z różnych procesorów.

PCJ – PI Calculations using integral



PCJ – Raytracing



MapReduce (wyznaczanie średniego wieku) - Java

- **Java**

```
long sum = 0;
for ( User user : users ) {
   um += user.getAge ();
}
double average = (double) sum / users.size ();
```

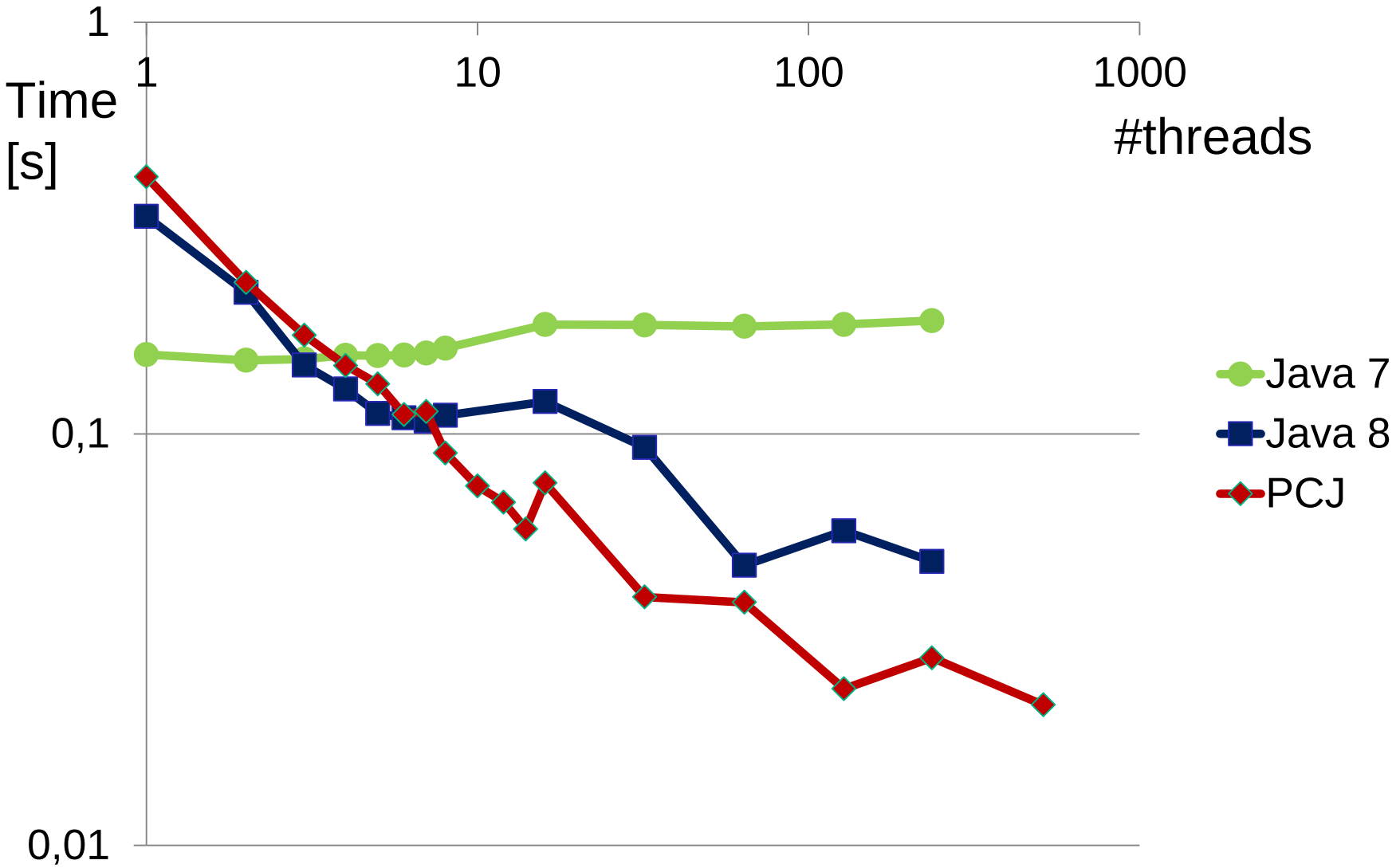
- **Java 8 parallel streams**

```
long sum = users.parallelStream ()
                .map (u -> ( long ) u. getAge ())
                .reduce ( Long :: sum )
                .get ();
double average = (double) sum / users.size ();
```

MapReduce - PCJ

```
@Shared  long sum ;
@Shared  int usersCount ;
...
myUsers = loadUsers( PCJ.myId ());
long s = 0;
for ( User u : myUsers ) {
      s += u. getAge ();
}
PCJ.putLocal ("sum", s);                    // The same fpr size
PCJ.barrier ();
s = pcj_reduce ("sum");
double average = (double) s / count ;
```

PCJ performance – MapReduce algorithm



pcj.icm.edu.pl

Piotr Bała (ICM UW)

współpraca (doktoranci):
implementacja biblioteki PCJ:
Marek Nowicki (WMiI UMK)

testy, przykłady, wykorzystanie biblioteki PCJ:
Łukasz Górski (WMiI UMK)
Magdalena Ryczkowska (WMiI UMK)
Michał Szynkiewicz (WMiI UMK)