EPSRC Centre for Doctoral Training in
Pervasive Parallelism

Algorithmic Skeletons:

The **future** of parallel programming.

Martin Rüfenacht
Chris Cummins

THE UNIVERSITY of EDINBURGH

informatics

EPSRC
Pioneering research
and skills

The Papers

[1] M. Cole

"**Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming**",

Parallel Computing, vol. 30, no. 3, pp. 389–406, Mar. 2004.

[2] M. Leyton and J. M. Piquer

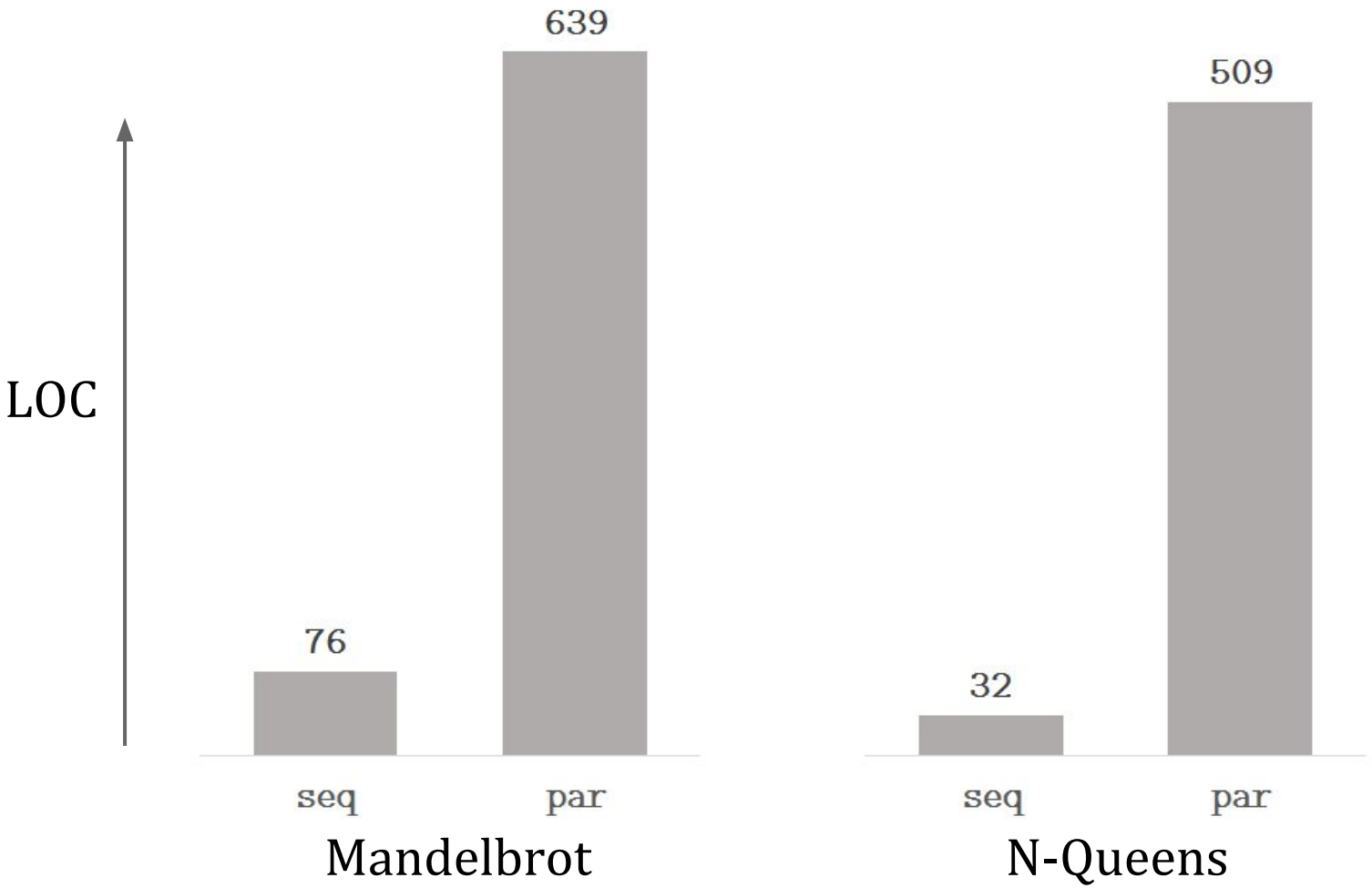
"**Skandium: Multi-core Programming with Algorithmic Skeletons**",

2010 18th Euromicro Conference on Parallel, Distributed Network-based Processing, pp. 289–296, Feb. 2010.

A brief overview of Skeletons

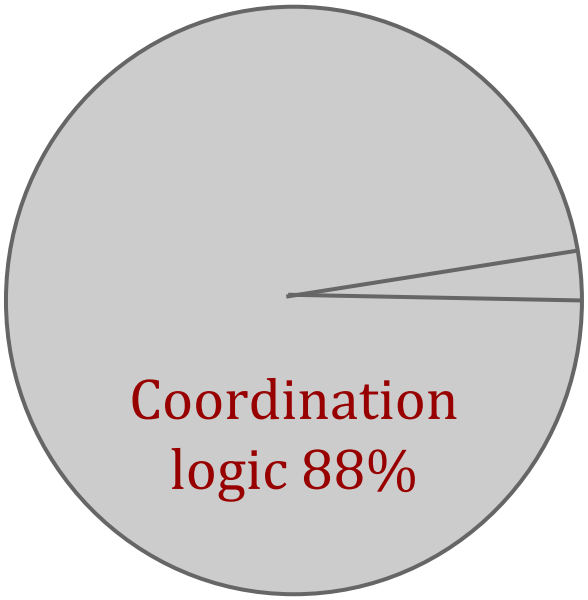
1. **What is the problem?**
2. How do Skeletons attempt to solve it?
3. How well do Skeletons achieve this?

Writing parallel software is impossibly time consuming.

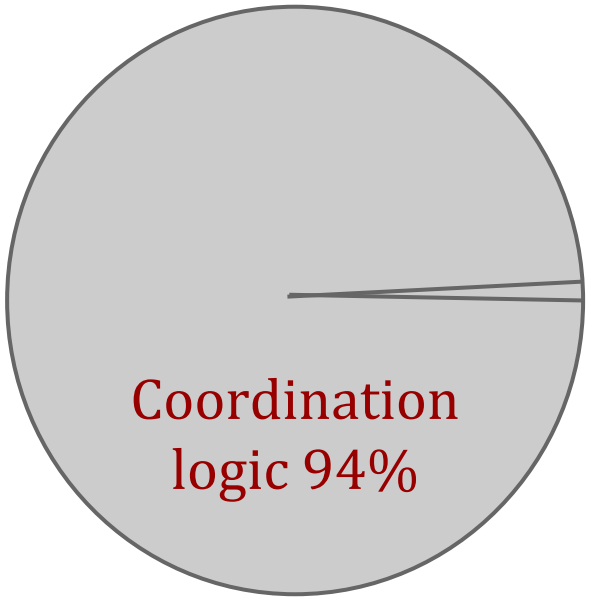


Sources:
http://rosettacode.org/
http://wili.cc/blog/mandelbrot-mpi.html
http://penguin.ewu.edu/~trolfe/MpiQueen/MpiQueen.c

How much of that is actually relevant to the task?



Mandelbrot



N-Queens

A brief overview of Skeletons

1. What is the problem?
2. **How do Skeletons attempt to solve it?**
3. How well do Skeletons achieve this?

Abstracting generic patterns

```
def merge_sort(A):
    if cant_split(A):
        return A
    else:
        B[] = divide(A)
        map(merge_sort, B)
        return combine(B)
```

Identify the problem domain

```
def merge_sort(A):
    if cant_split(A):
        return A
    else:
        B[] = divide(A)
        map(merge_sort, B)
        return combine(B)
```

Abstract the problem domain

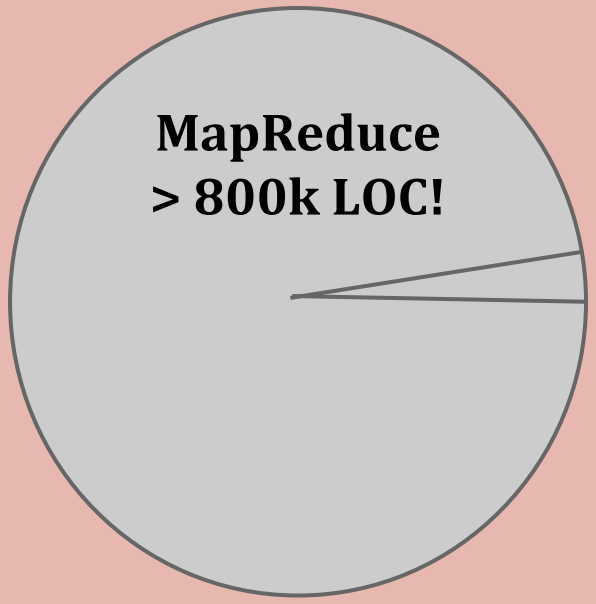
```
def divide_and_conquer(A, f):
    if f.cant_split(A):
            return f.solve(A)
    else:
            B[] = f.divide(A)
            map(divide_and_conquer, B)
            return f.combine(B)
```

Create re-usable patterns

```
def merge_sort(A):
    return divide_and_conquer(A, f1)

def max_subarray(A):
    return divide_and_conquer(A, f2)

def n_queens(A):
    return divide_and_conquer(A, f3)

// ... etc
```

Separate the pattern from the problem.

Skeletal domain



Problem domain

Web indexing.

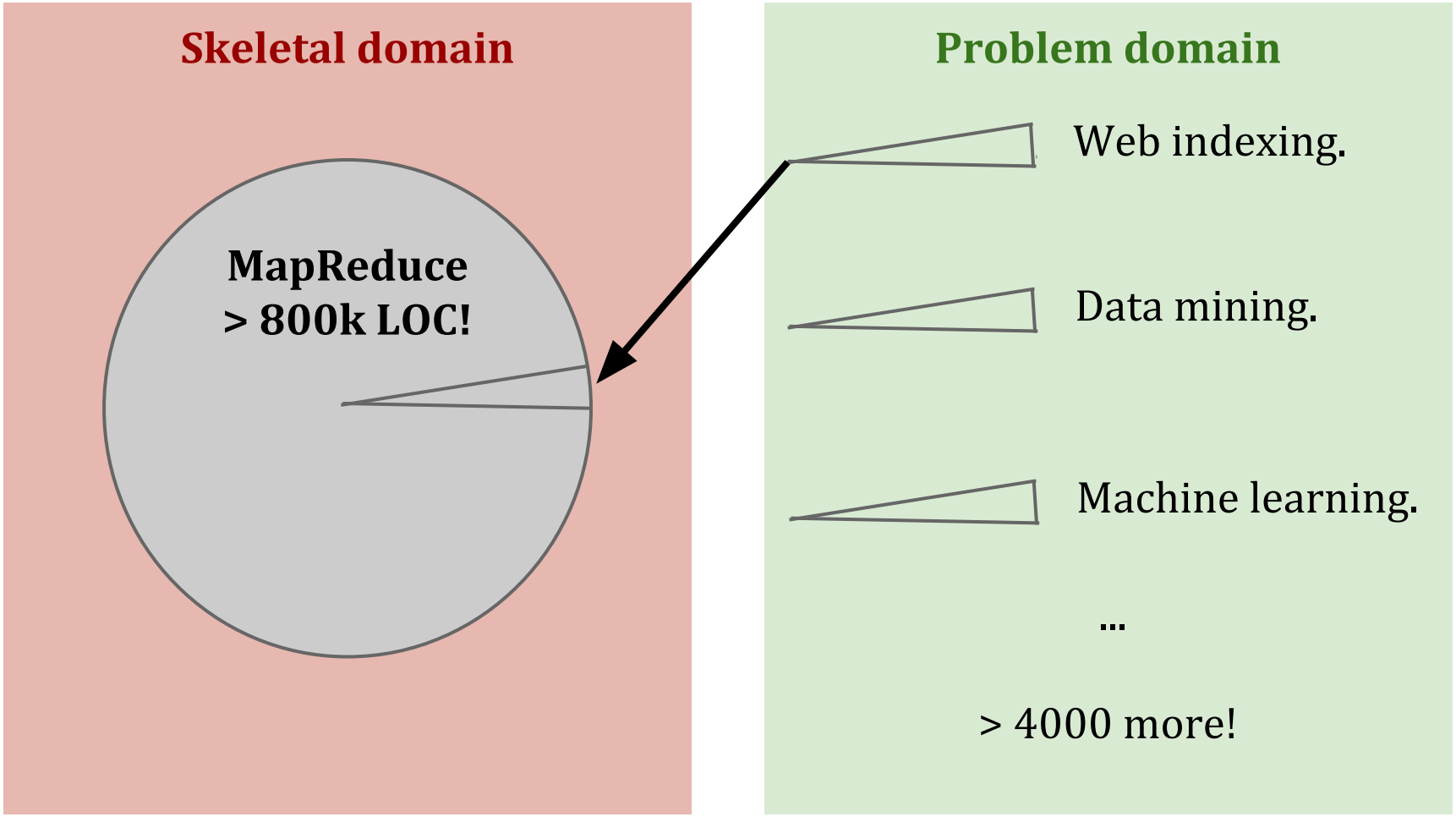
Data mining.

Machine learning.

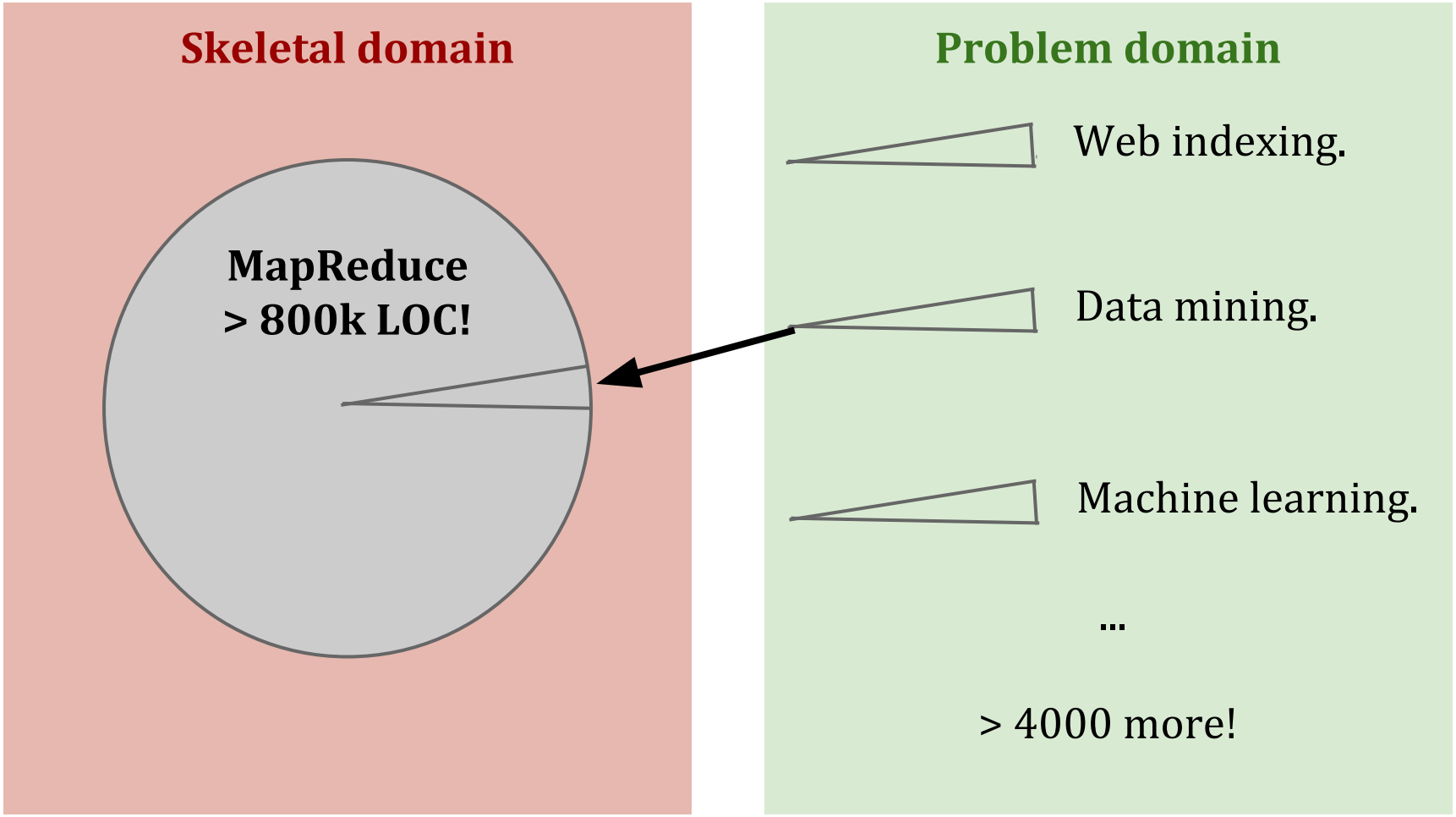
...

> 4000 more!

Separate the pattern from the problem.

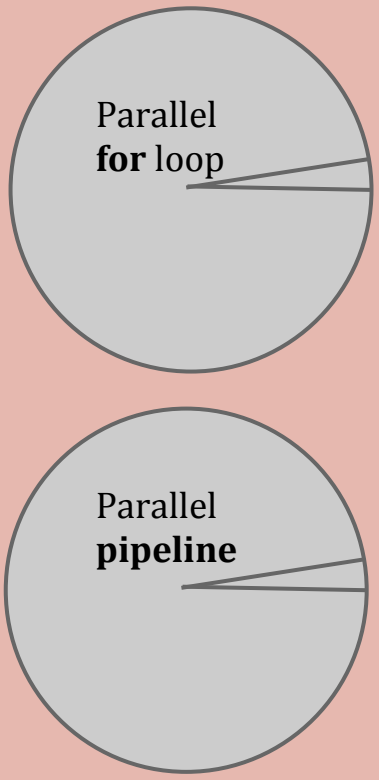


Separate the pattern from the problem.



Separate the pattern from the problem.

Skeletal domain



Intel TBB
> 115k LOC!

Problem domain

Sorting function.

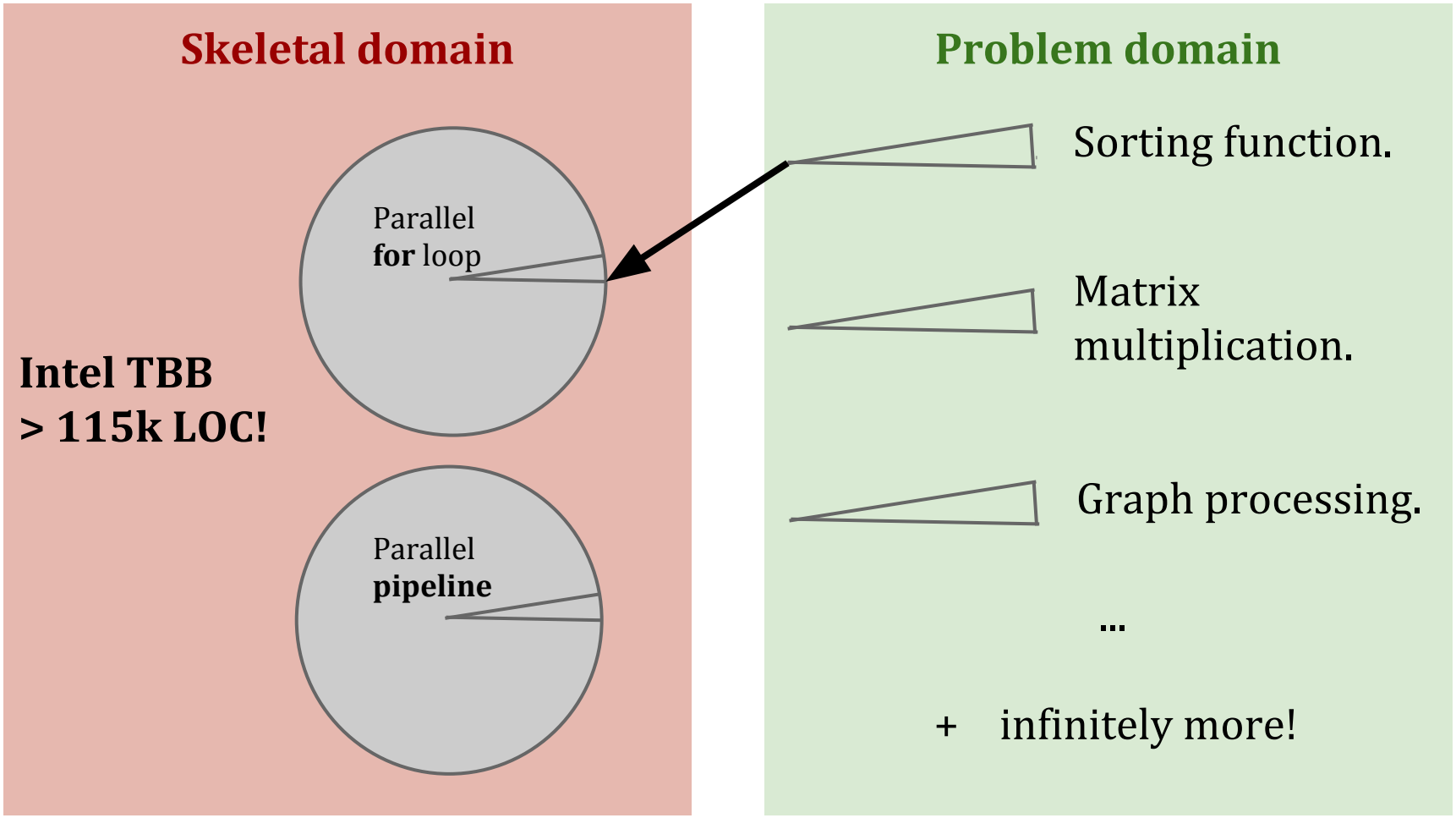
Matrix multiplication.

Graph processing.

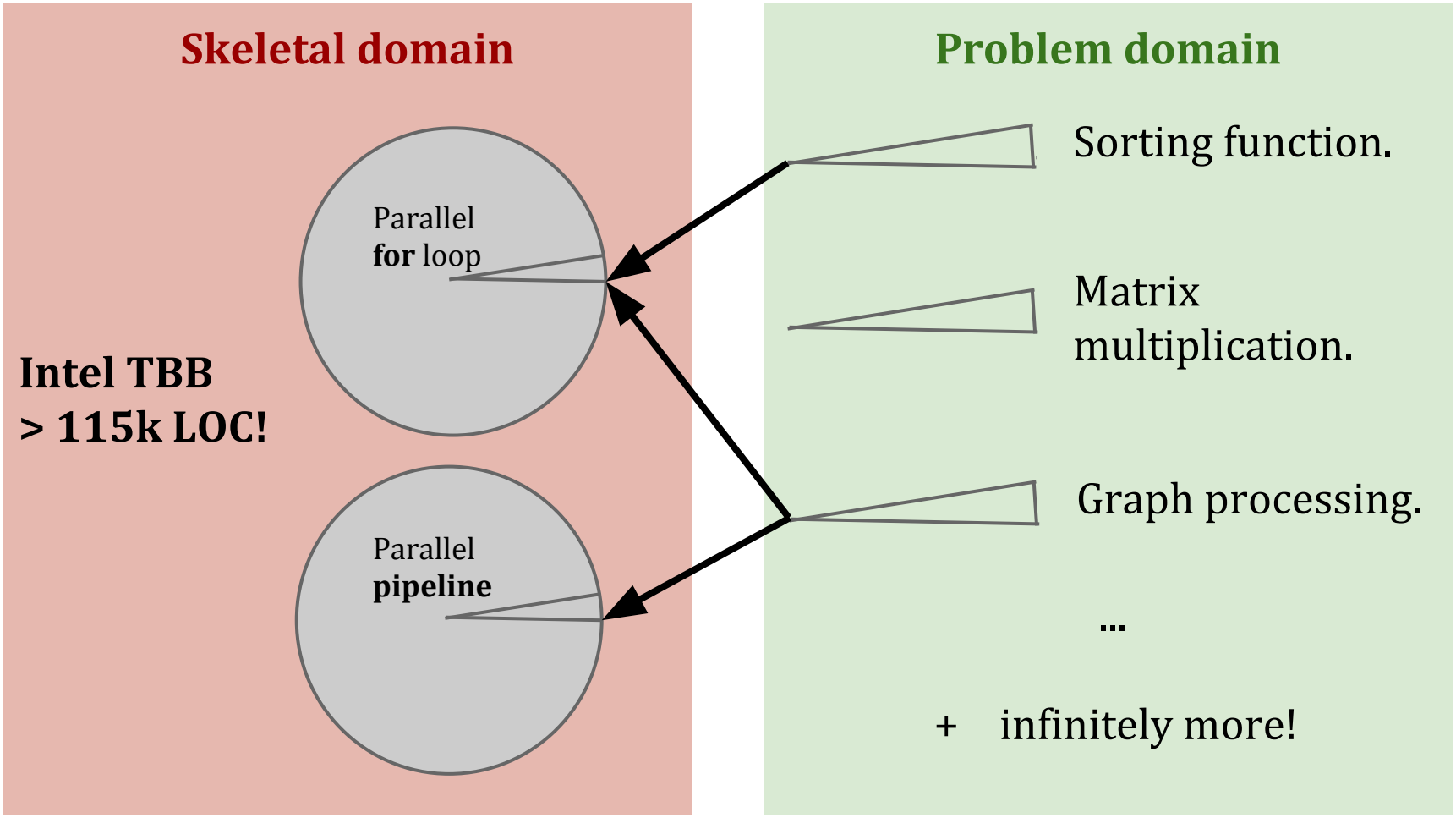
...

\+ infinitely more!

Separate the pattern from the problem.



Separate the pattern from the problem.



A brief overview of Skeletons

1. What is the problem?
2. How do skeletons attempt to solve it?
3. **How well do they achieve this?**

Skeletons in academia

Skeleton frameworks:

Lithium, Muskel, ASSIST, CO2P3S, Calcium, Skandium, Eden, SKElib, eSkel, HDC, HOC-SA, PAS, EPAS, JaSkel, Mallba, Marrow, Muesli, P3L, SkIE, SBASCO, SCL, SkePU, SKiPPER, QUAFF, SkeTo, Skil, T4P, …..

Multiple implementations, with different goals.

Skeletons in academia

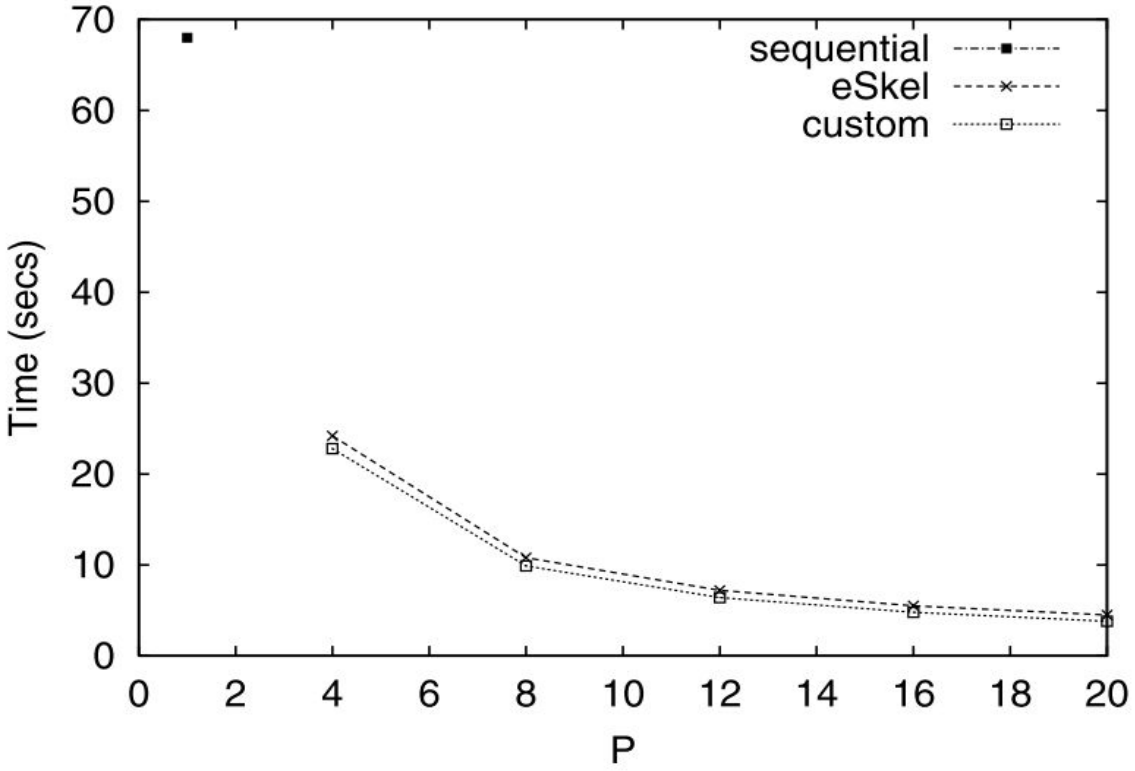
Skeleton frameworks:

Lithium, Muskel, ASSIST, CO2P3S, Calcium, Skandium, Eden, SKElib, eSkel, HDC, HOC-SA, PAS, EPAS, JaSkel, Mallba, Marrow, Muesli, P3L, SkIE, SBASCO, SCL, SkePU, SKiPPER, QUAFF, SkeTo, Skil, T4P, …..

Multiple implementations, with different goals.

eSkel [1]

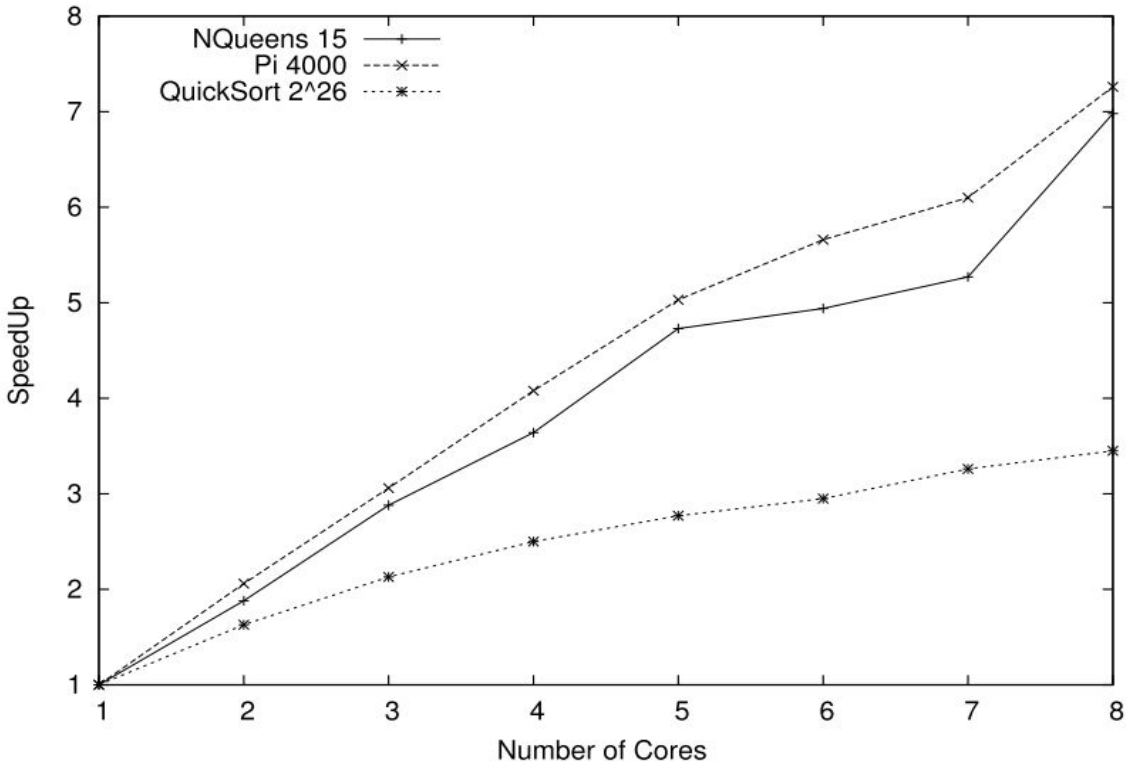
1. Imperative
2. Built on MPI
3. Distributed



Comparison against **hand-tuned**.
Multiple imp. of **same** benchmark.

Skandium [2]

1. Obj. Oriented
2. Built on Java
3. Multi-core



Comparison against **self**.
Imp. of **multiple** benchmarks.

Why the differences in approach?

eSkel [1] published **2 months** before Intel started producing **slower** chips with **more cores**.

Parallel programming is no longer for the **elite**.

Today: *"All computers are parallel"*

- *Structure Parallel Programming (2008)*

Algorithmic Skeletons

Programs increase in **complexity** exponentially when parallelised.

Algorithmic Skeletons address this problem and provide:

- **Simpler** programs.
- Greater **code reuse**.
- Improved **performance**.
- Improved **optimisations**.

The Papers

[1] M. Cole

"**Bringing skeletons out of the closet: a pragmatic manifesto for skeletal parallel programming**",

Parallel Computing, vol. 30, no. 3, pp. 389–406, Mar. 2004.

[2] M. Leyton and J. M. Piquer

"**Skandium: Multi-core Programming with Algorithmic Skeletons**",

2010 18th Euromicro Conference on Parallel, Distributed Network-based Processing, pp. 289–296, Feb. 2010.