

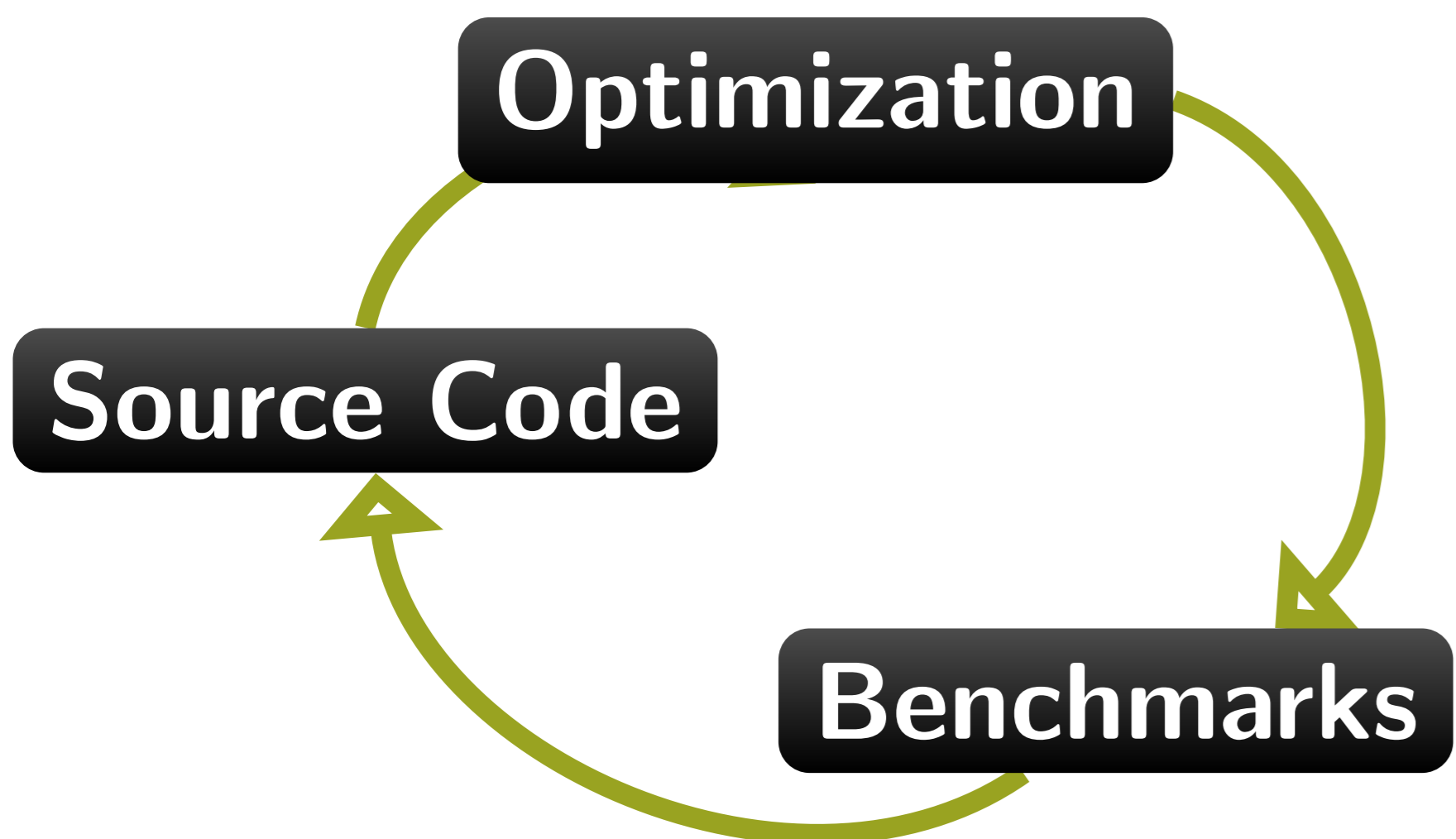
# A genetic and source-to-source approach to iterative compilation

Serge Guelton, serge.guelton@telecom-bretagne.eu  
 Institut Télécom, Télécom Bretagne, France

TELECOM  
Bretagne



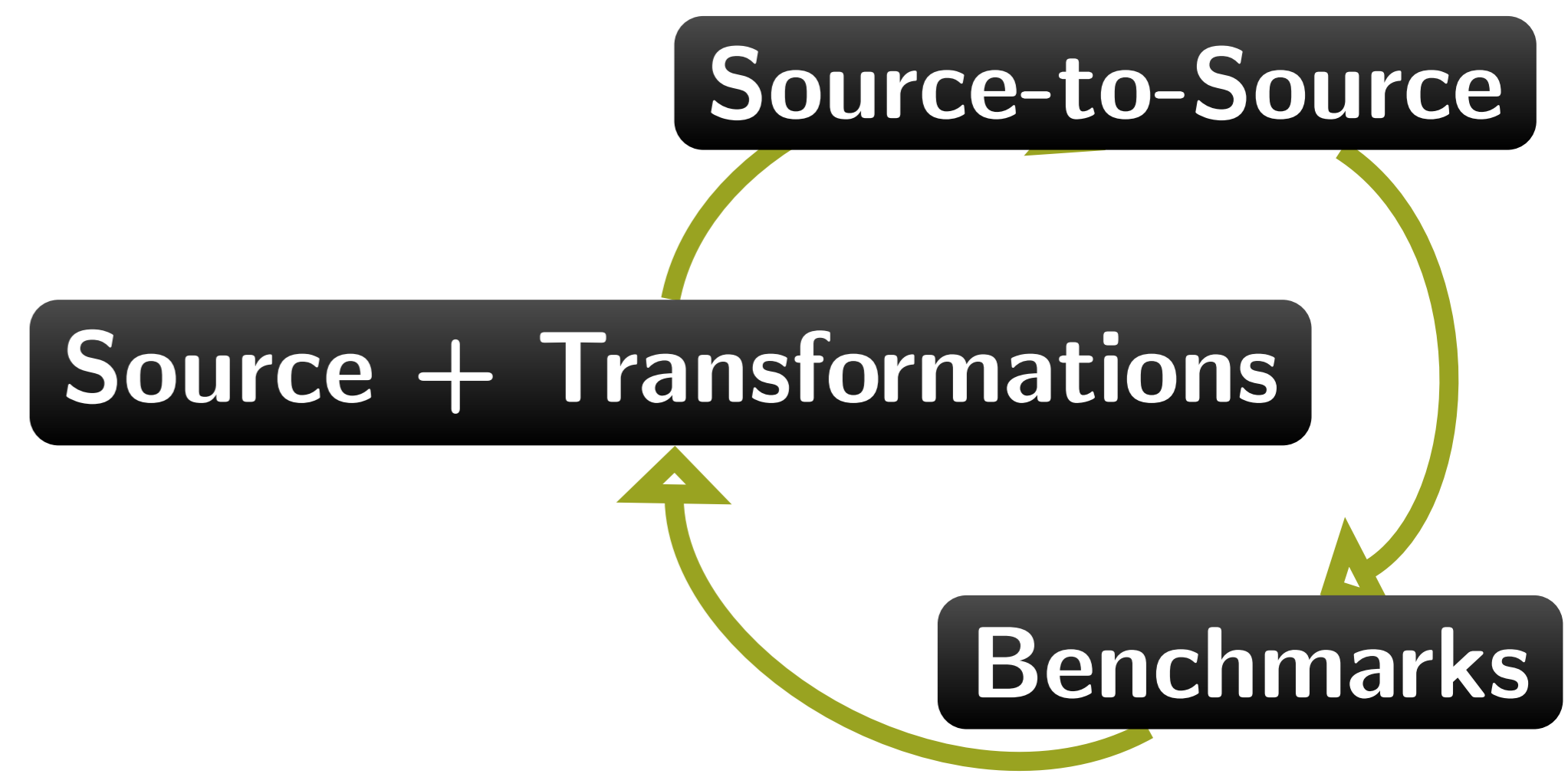
## Classical optimization Cycle



## With Source-to-source compilers

- PIPS – OSS from Mines ParisTech, 21 years old, 550 KLOC
- 242 transformations/phases
- ODL  $\Rightarrow$  introspection, persistence ...
- à la make phase chaining engine
- Linear Algebra-based analysis

## Enhanced optimization Cycle



## Automation through Python

### Automatically chain Transformations

```

from pyps import * # tool namespace
w = workspace("test.c") # load files
foo=w["foo"] # per function selection
bar=w["bar"] # bar as a 'module' type
bar.unroll(FACTOR=4) # call transformations on module
foo.inlining(CALLERS="bar",PURGE_LABELS=True)
bar.display() # display module content on stdout
w.save(indir="sample") # save transformed files
w.compile(outdir="sample",link=False) # compile them
    
```

## Various Approaches

- Transverse  $\mathbb{T}$ , the source-to-source transformation set  $\Rightarrow$  unrolling, inlining, loop distribution, scalarization ...
- $K$  is a selection factor  $\Rightarrow$  control the greedy complexity



Full:  $\mathcal{O}(|\mathbb{T}|!)$

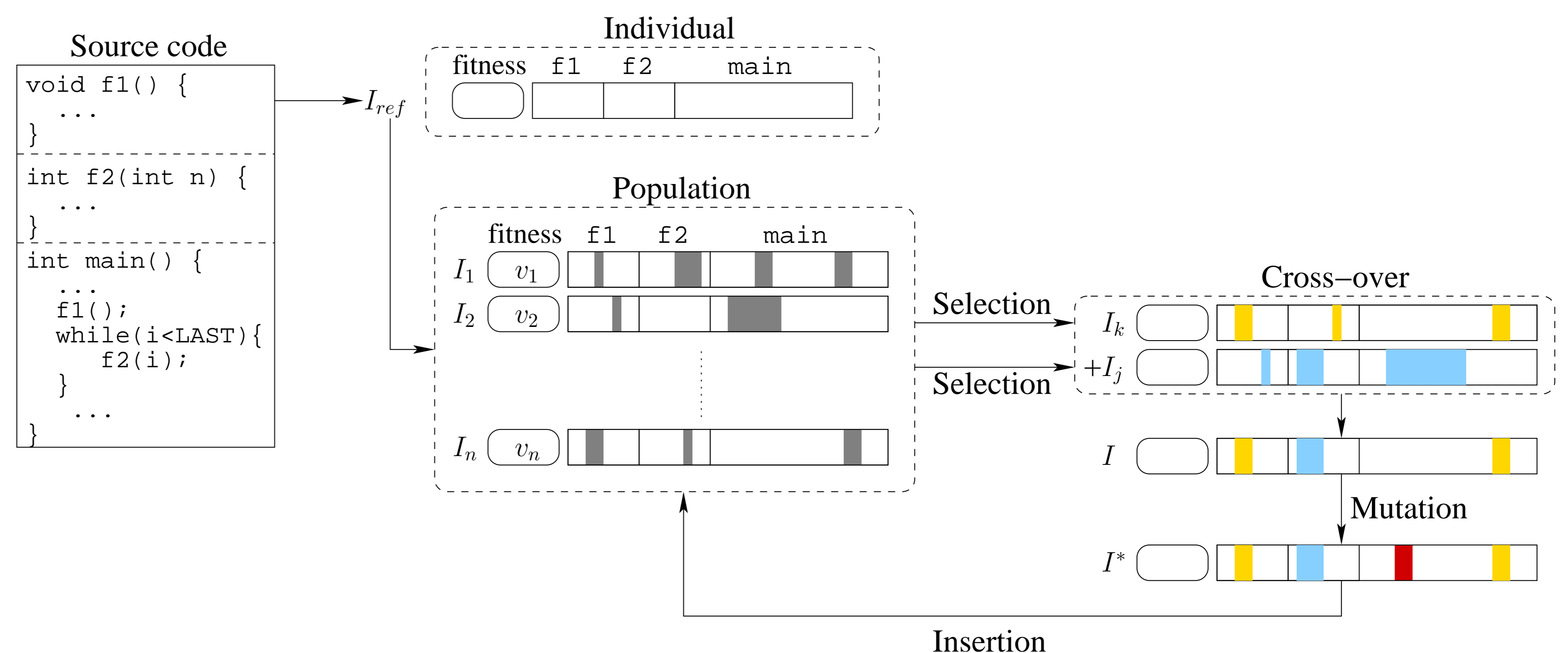


Greedy:  $\mathcal{O}(K * |\mathbb{T}|^2)$

$\Rightarrow$  Either too costly or too naive !

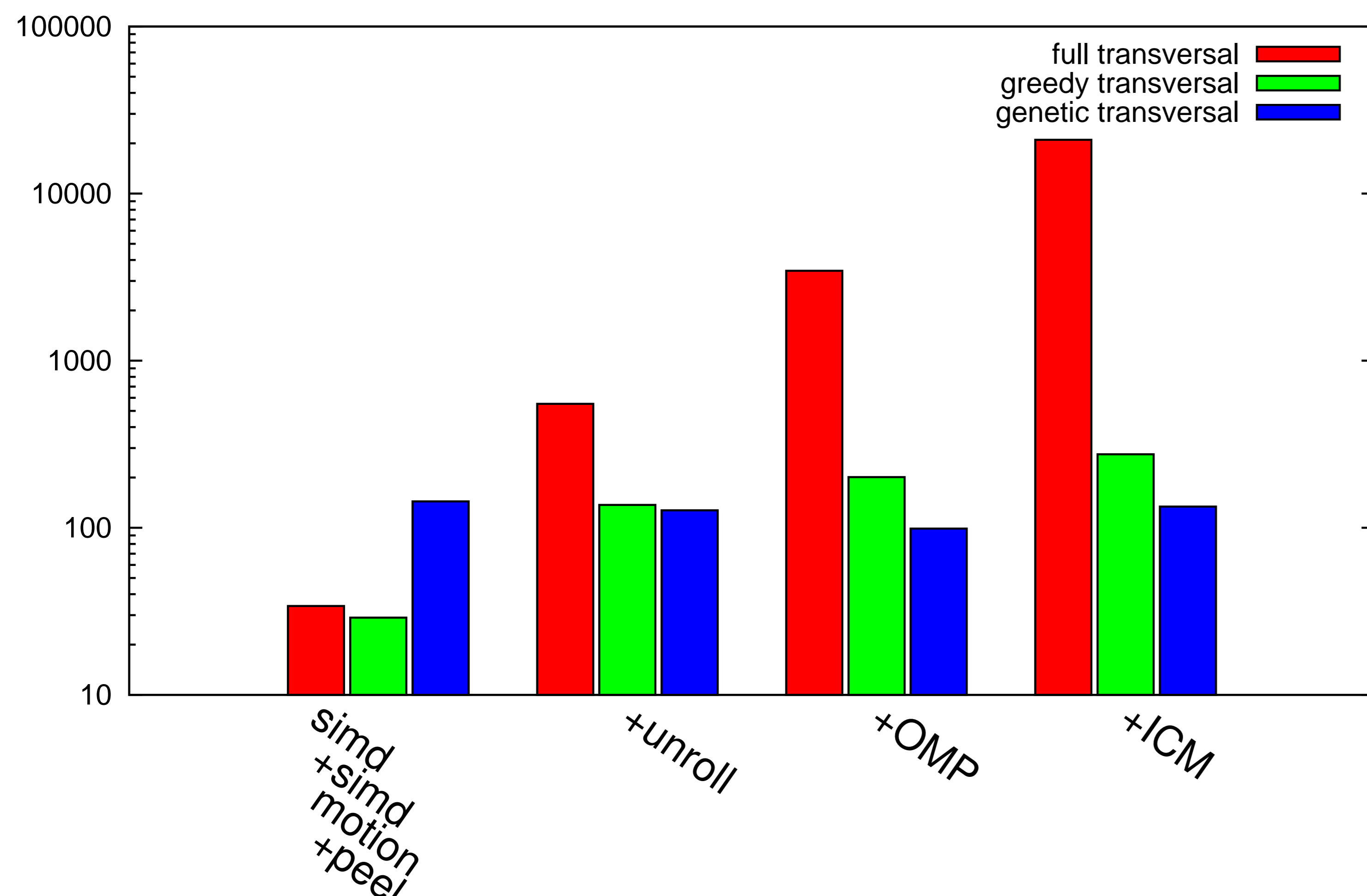
## Genetic Algorithm Based Transformation space exploration

- Search heuristic based on biological evolution
- **Individual**: source code
- **Mutation**: code transformation
- **Fitness**: execution time
- Configurable Complexity
- Avoid Greedy algorithm problem
- $\pm$  Random behavior
- Parallelization opportunity ?



## Time performance

Time taken by various iterative compilation method



## On an inner product

```

float dotprod(float b[SIZE], float c[SIZE]) {
    int i;
    float a = 0;
    for(i = 0; i < SIZE; i++)
        a += b[i]*c[i];
    return a;}
    
```

## Best Element

	ref	full	greedy	genetic
execution time (clock)	11435	8641	9870	8673

## Best transformation

simdization + simd constant motion  
 + unrolling + omp // for