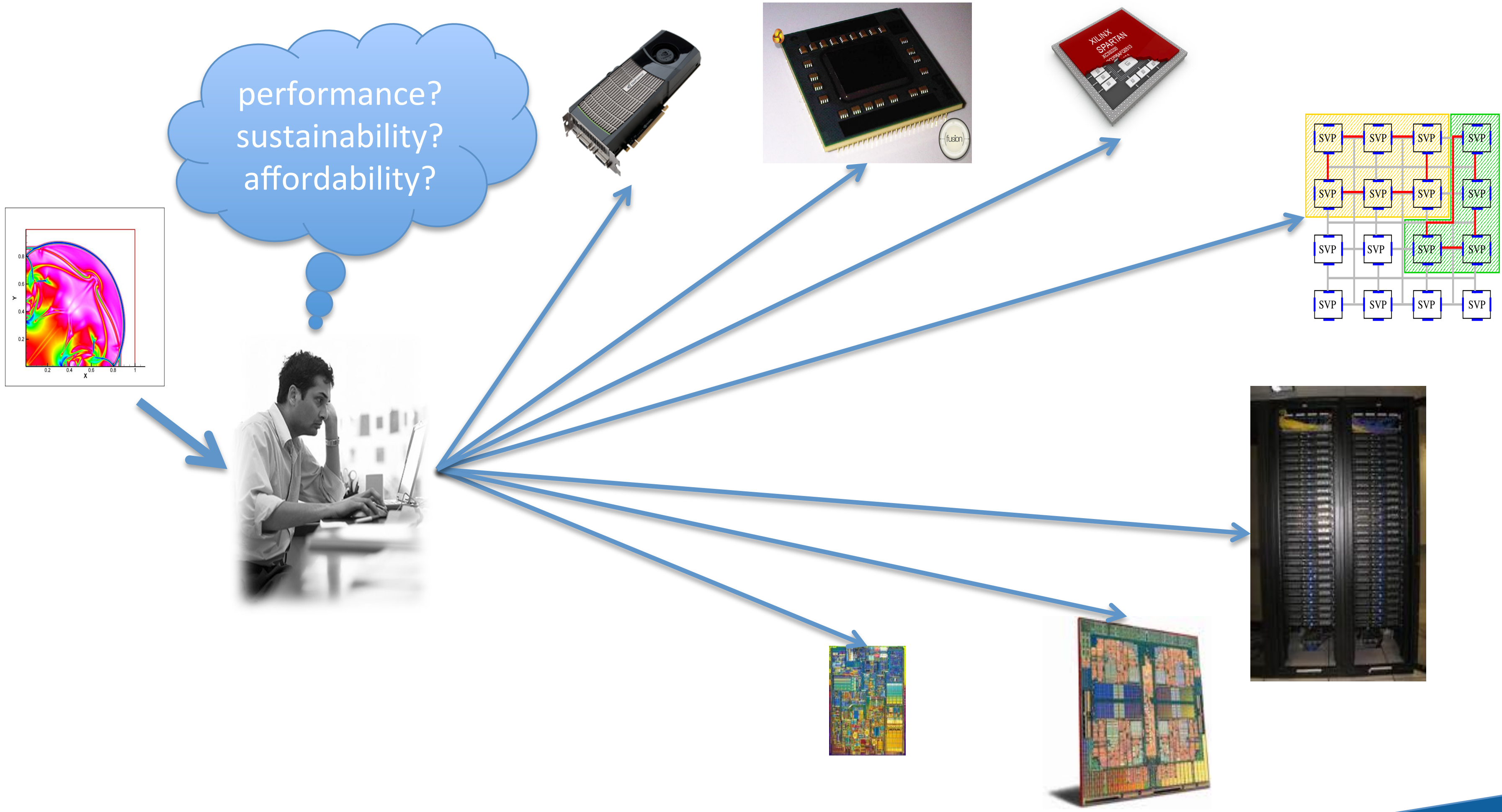
Data-Parallel Programming using SaC lecture 1

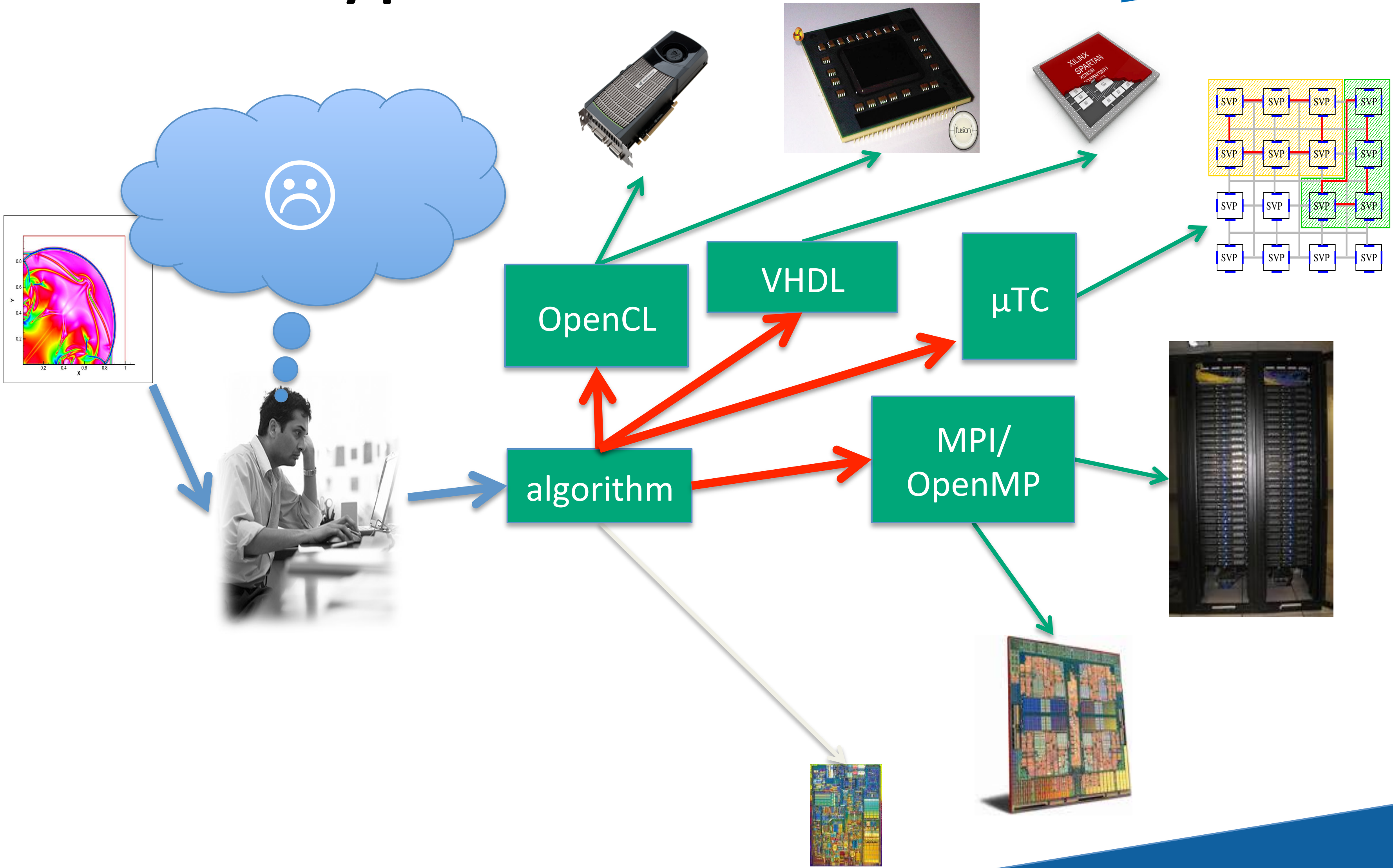
F21DP Distributed and Parallel Technology

Sven-Bodo Scholz

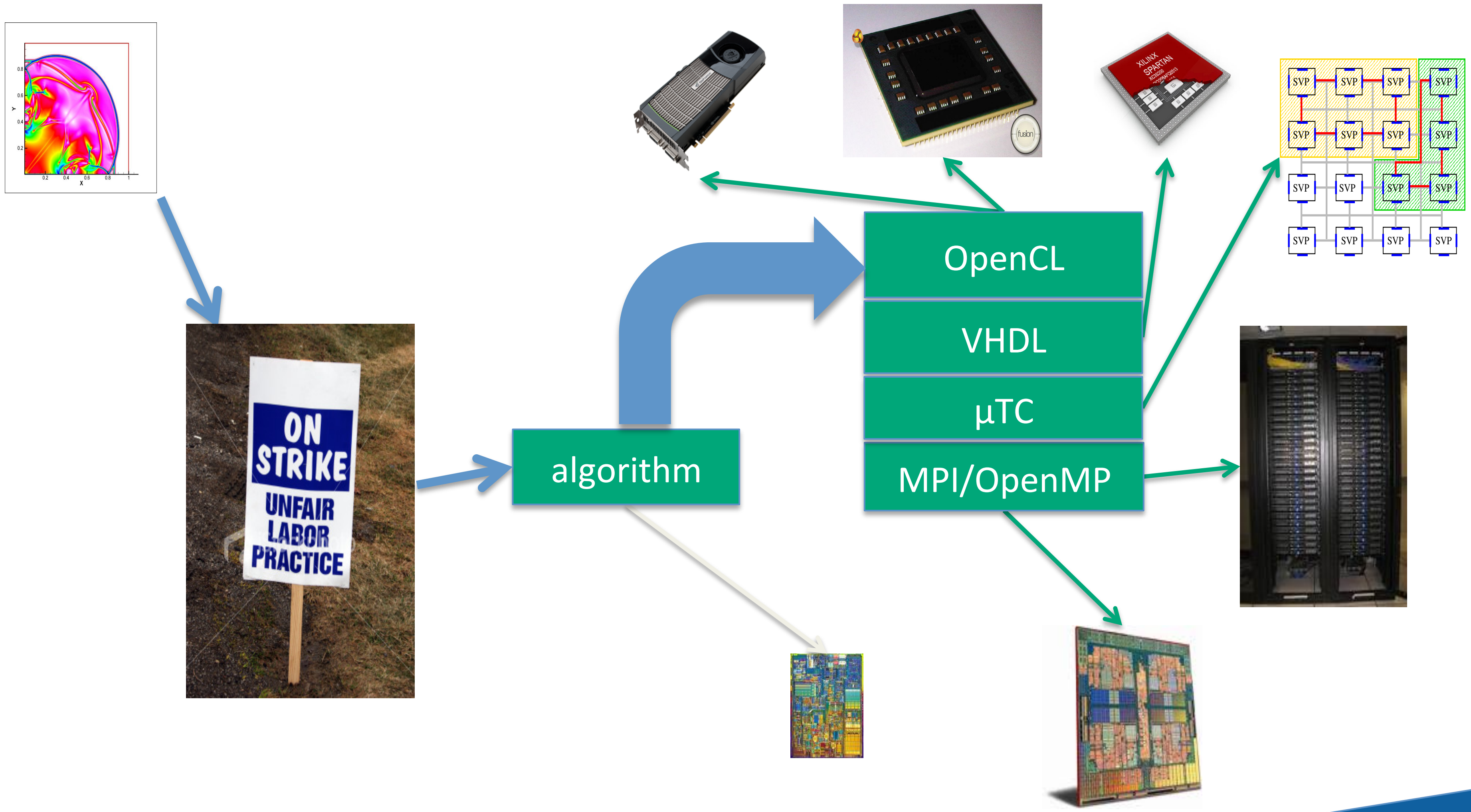
The Multicore Challenge



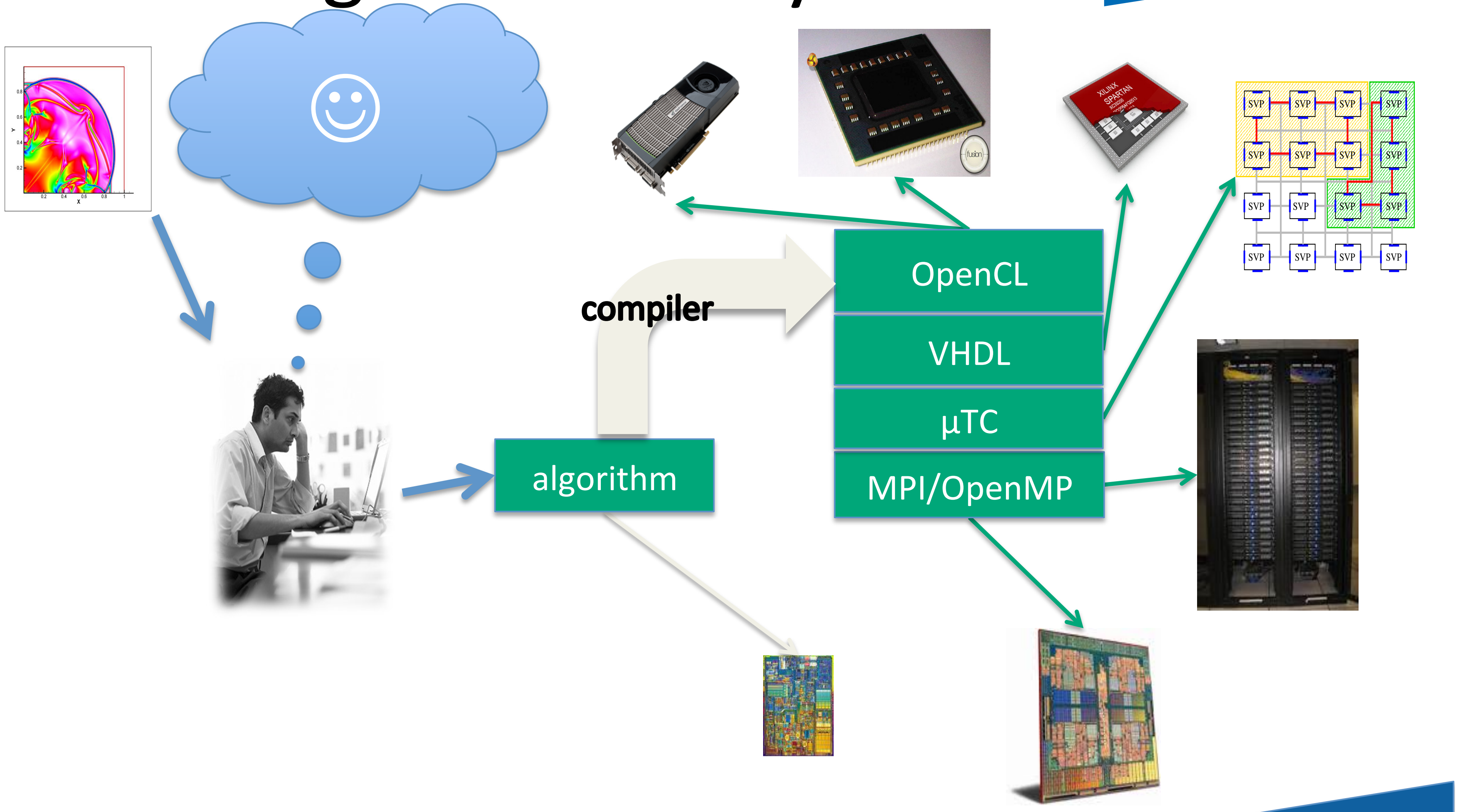
Typical Scenario



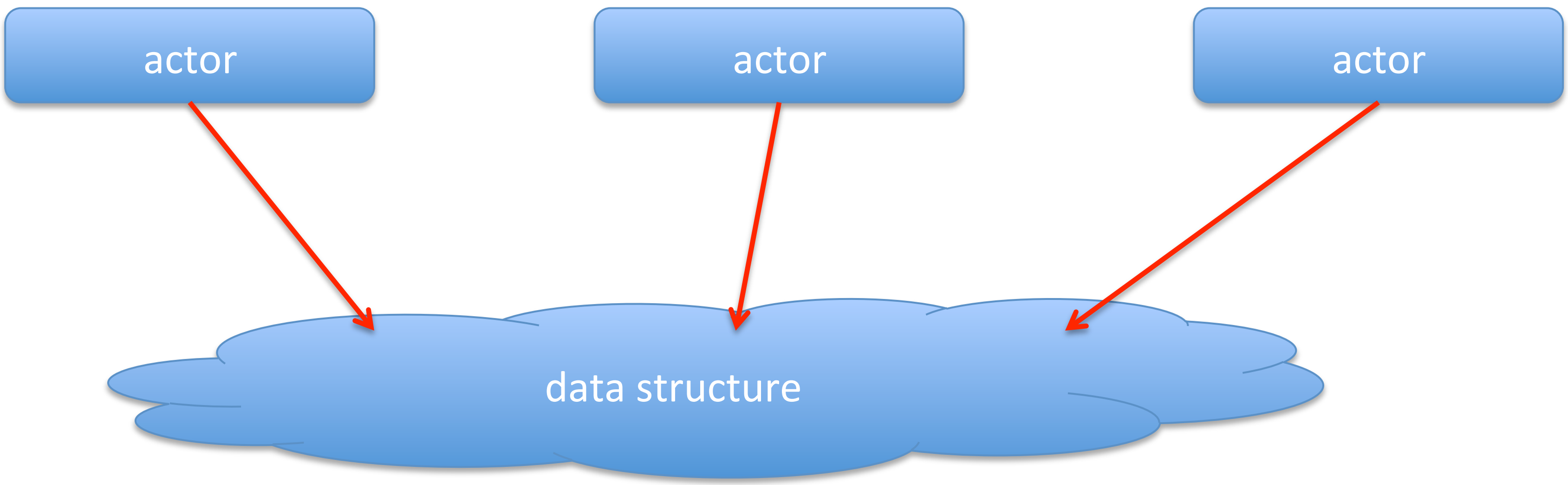
Tomorrow’s Scenario



The High-Portability Vision



What is Data-Parallelism?

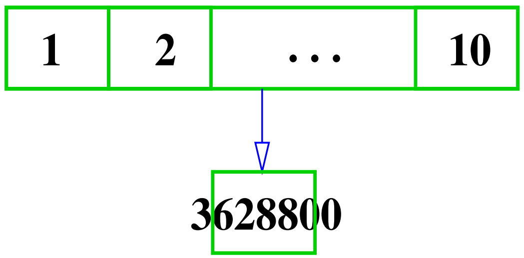


concurrent operations on a single data structure

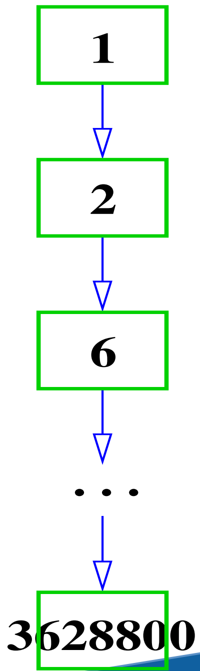
Data-Parallelism, More Concretely

Formulate algorithms in space rather than time!

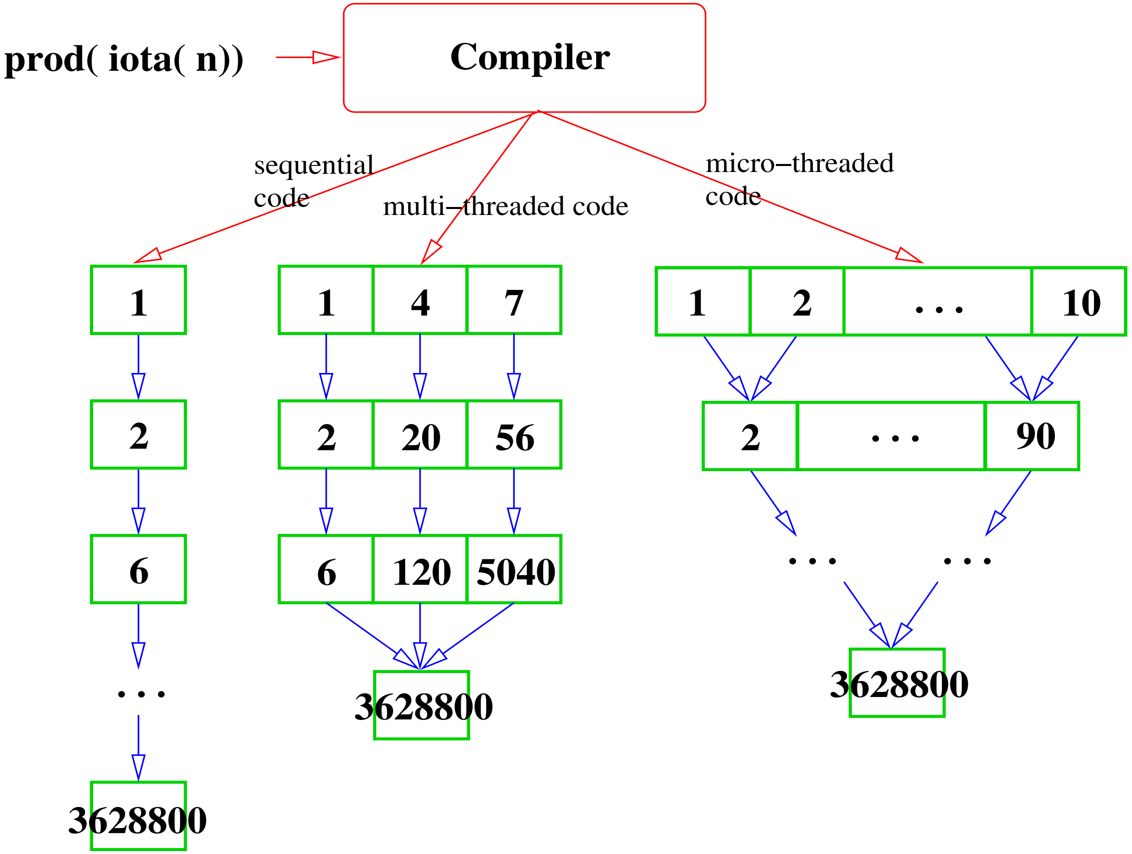
```
prod = prod( iota( 10)+1)
```




```
prod = 1;
for( i=1; i<=10; i++) {
  prod = prod*i;
}
```

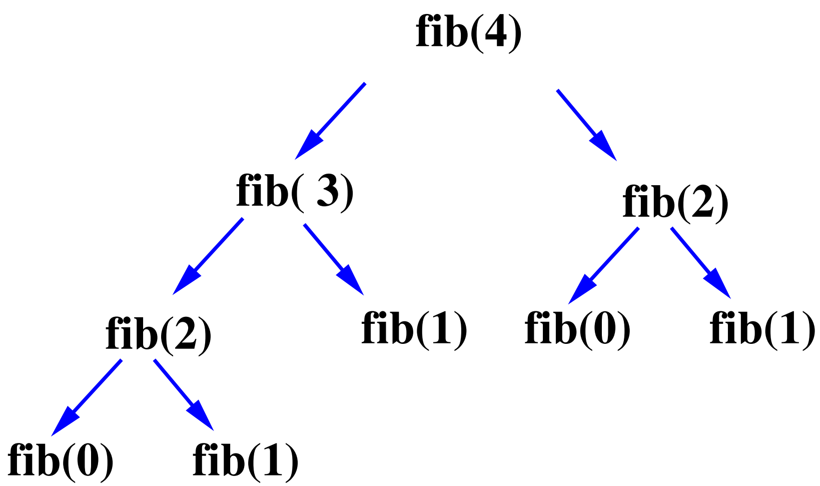


Why is Space Better than Time?



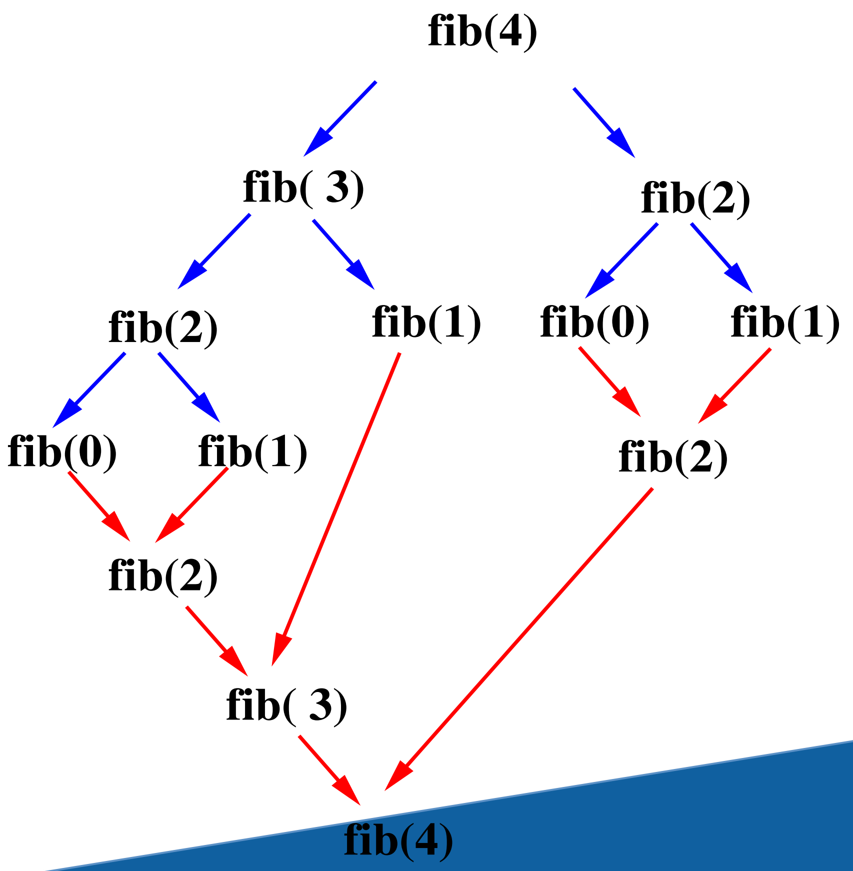
Another Example: Fibonacci Numbers

```
if( n<=1)
  return n;
} else {
  return  fib( n-1) + fib( n-2);
}
```



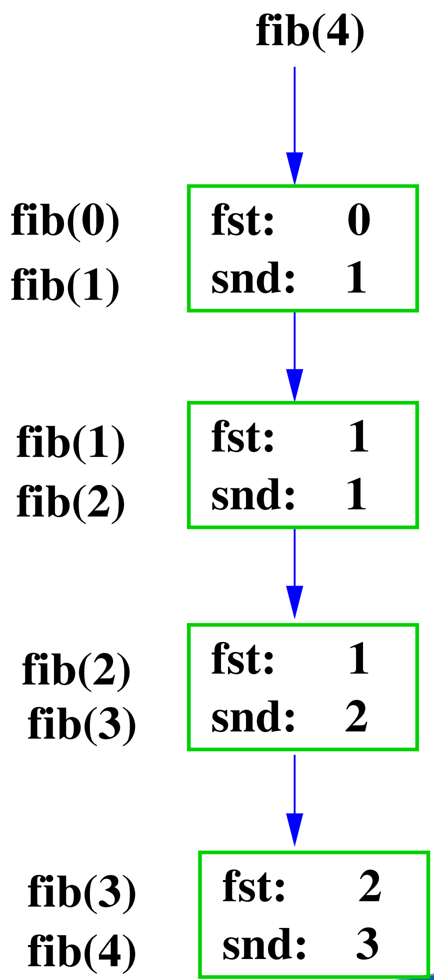
Another Example: Fibonacci Numbers

```
if( n<=1)
  return n;
} else {
  return  fib( n-1) + fib( n-2);
}
```



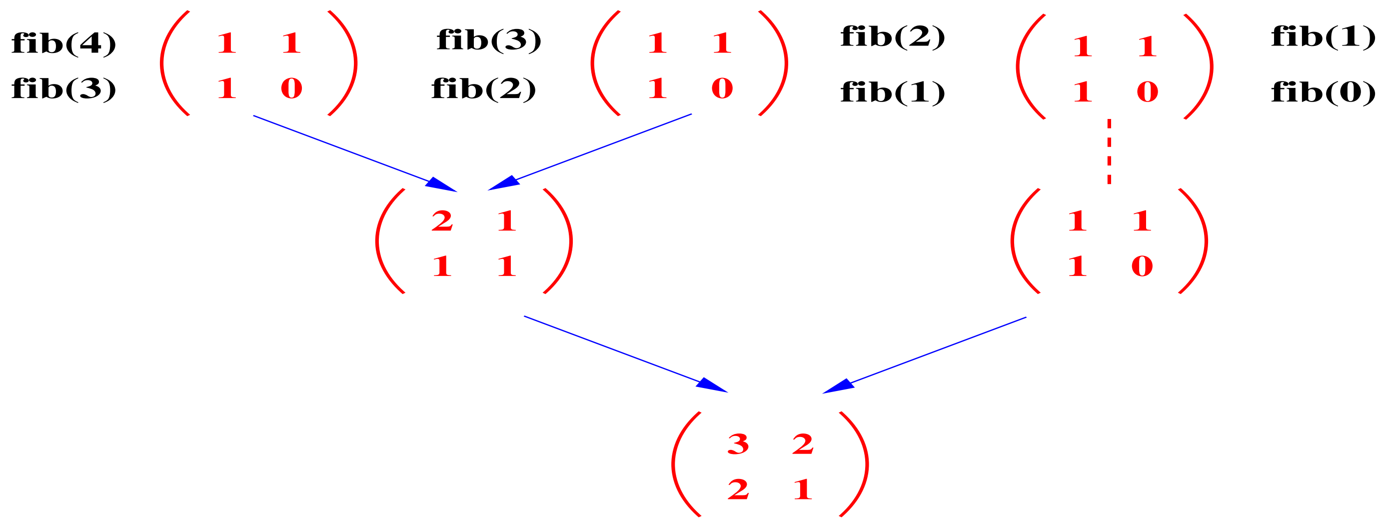
Fibonacci Numbers – now linearised!

```
if( n== 0)
  return fst;
else
  return fib( snd, fst+snd,
n-1)
```



Fibonacci Numbers – now data-parallel!

```
matprod( genarray( [n], [[1, 1], [1, 0]])) [0,0]
```



Everything is an Array

Think Arrays!

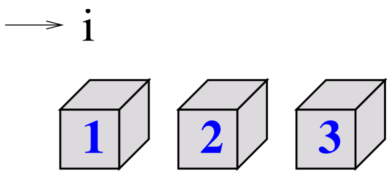
- Vectors are arrays.
- Matrices are arrays.
- Tensors are arrays.
- .... are arrays.

Everything is an Array

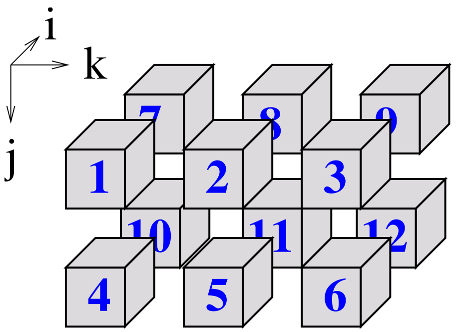
Think Arrays!

- Vectors are arrays.
- Matrices are arrays.
- Tensors are arrays.
- .... are arrays.
- Even scalars are arrays.
- Any operation maps arrays to arrays.
- Even iteration spaces are arrays

Multi-Dimensional Arrays



shape vector: [3]
data vector: [1, 2, 3]



shape vector: [2, 2, 3]
data vector: [1, 2, 3, ..., 11, 12]

42

shape vector: [ ]
data vector: [42]

Index-Free Combinator-Style Computations

L2 norm:

```
sqrt( sum( square( A)))
```

Convolution step:

```
W1 * shift(-1, A) + W2 * A + W1 * shift( 1, A)
```

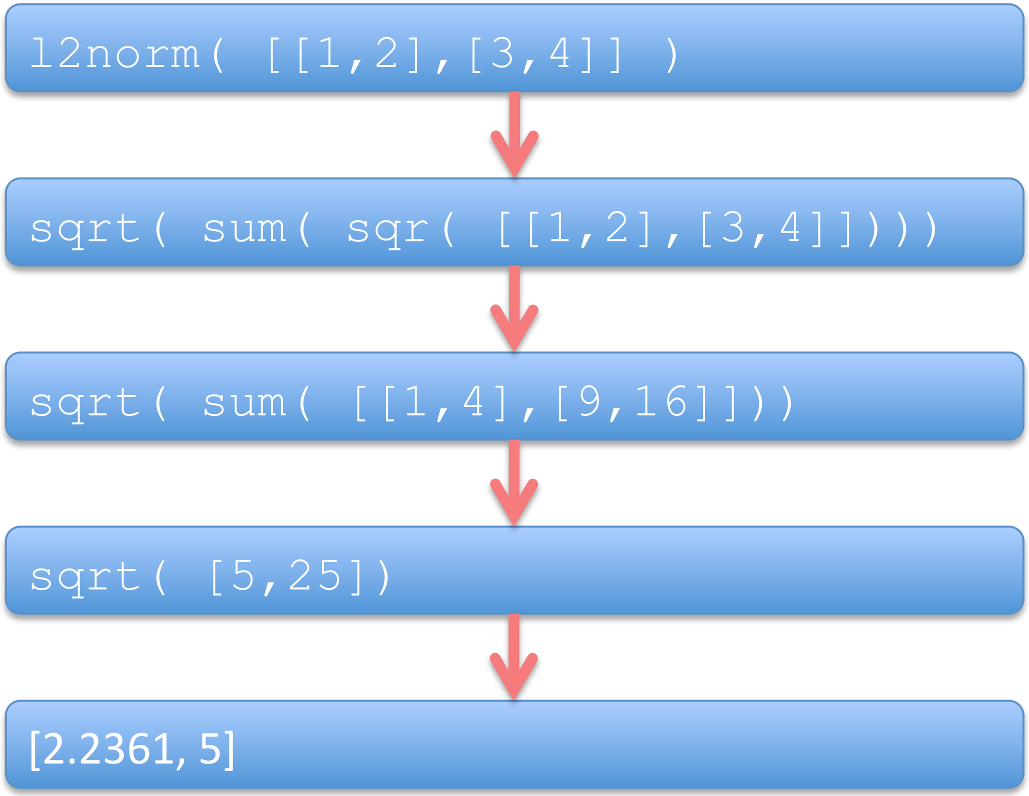
Convergence test:

```
all( abs( A-B) < eps)
```

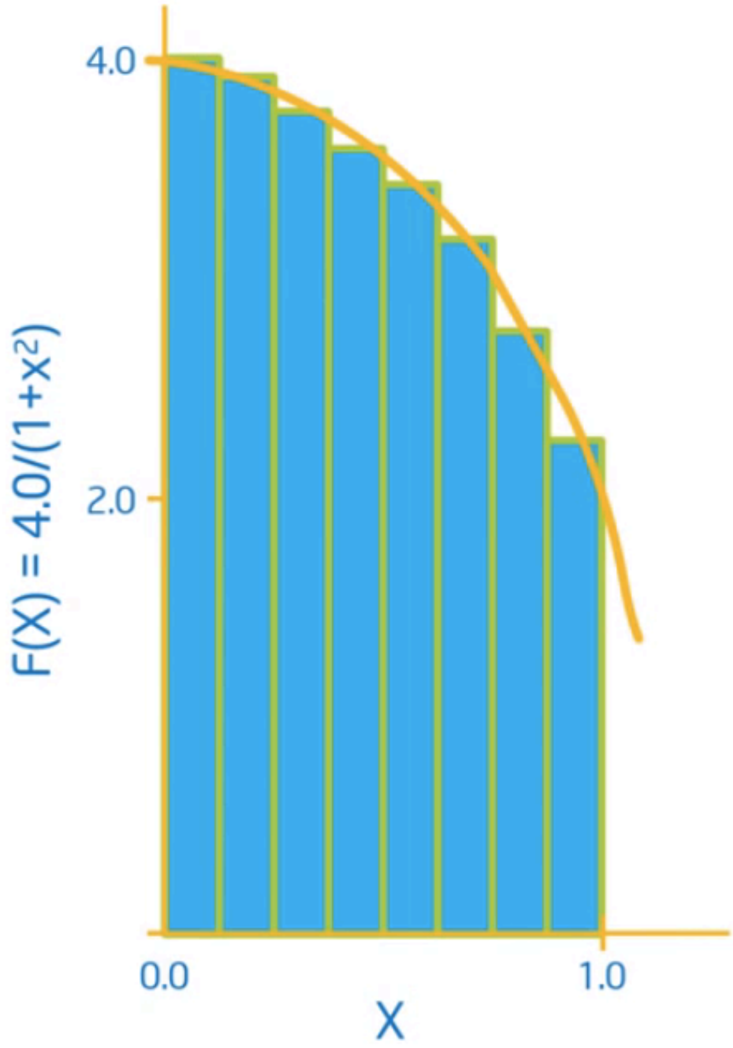
Shape-Invariant Programming

```
l2norm( [1,2,3,4] )
↓
sqrt( sum( sqr( [1,2,3,4])))
↓
sqrt( sum( [1,4,9,16]))
↓
sqrt( 30)
↓
5.4772
```

Shape-Invariant Programming



Computation of π



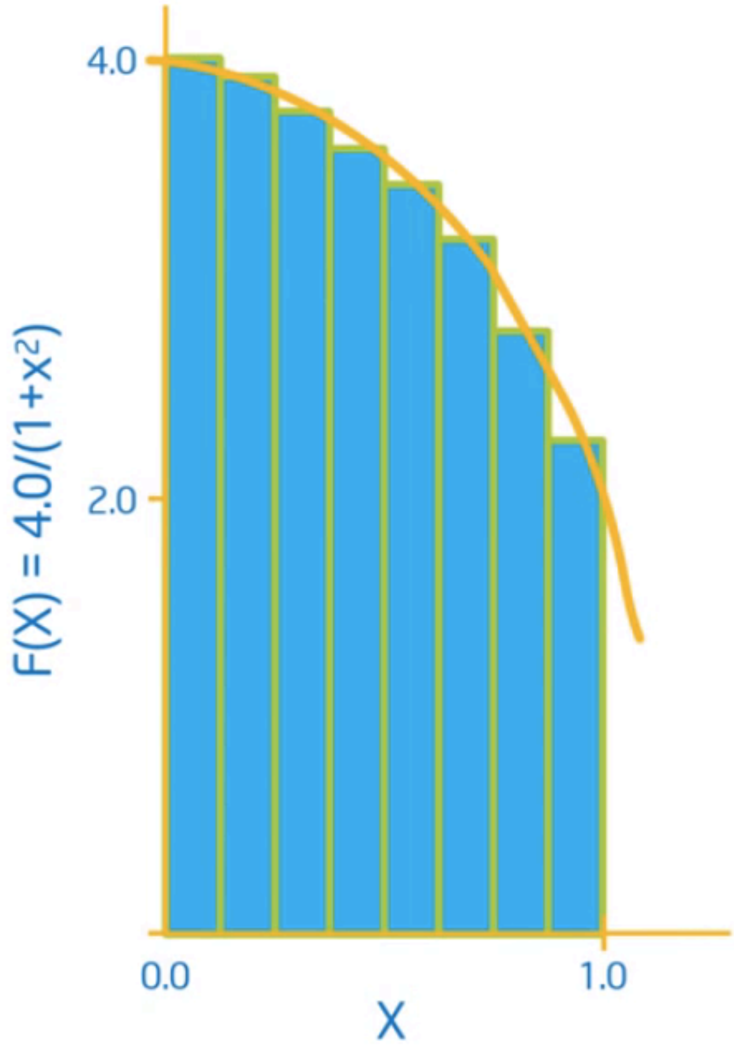
$$\int_0^1 \frac{4.0}{(1+X^2)}$$

Computation of π

```
double f( double x)
{
   return 4.0 / (1.0+x*x);
}

int main()
{
  num_steps = 10000;
  step_size = 1.0 / tod( num_steps);
  x = (0.5 + tod( iota( num_steps))) * step_size;
  y = { iv-> f( x[iv])};
  pi = sum( step_size * y);

  printf( " ...and pi is: %f\n", pi);
  return(0);
}
```



Programming in a Data-Parallel Style - Consequences

- much less error-prone indexing!
- combinator style
- increased reuse
- better maintenance
- easier to optimise
- huge exposure of concurrency!