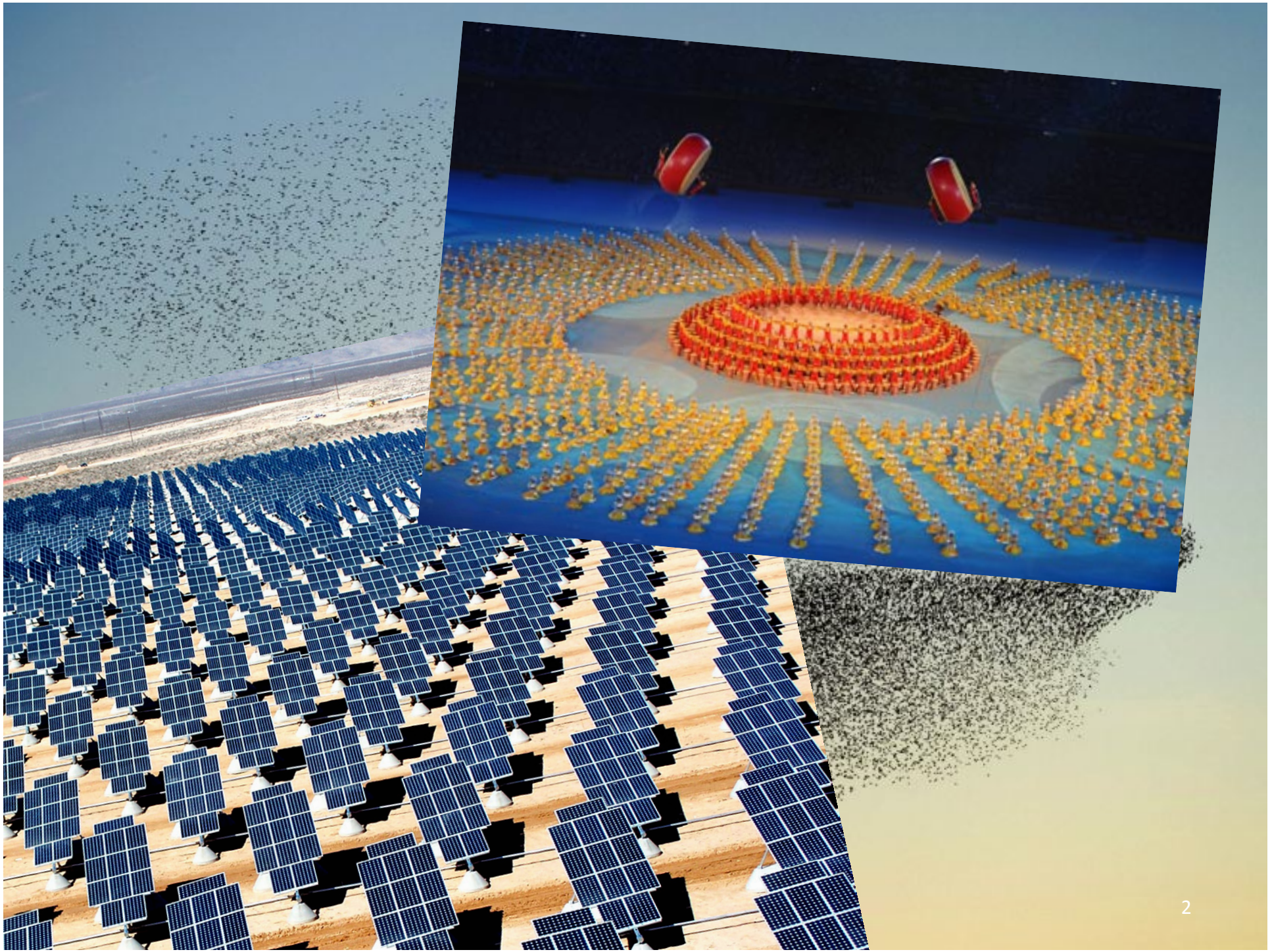
HERIOT WATT UNIVERSITY

The Powers of Array Programming

AiPL – Edinburgh

20.8.2014

Sven-Bodo Scholz



Everything is an Array

HERIOT WATT UNIVERSITY

Think Arrays!

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

Everything is an Array

HERIOT WATT UNIVERSITY

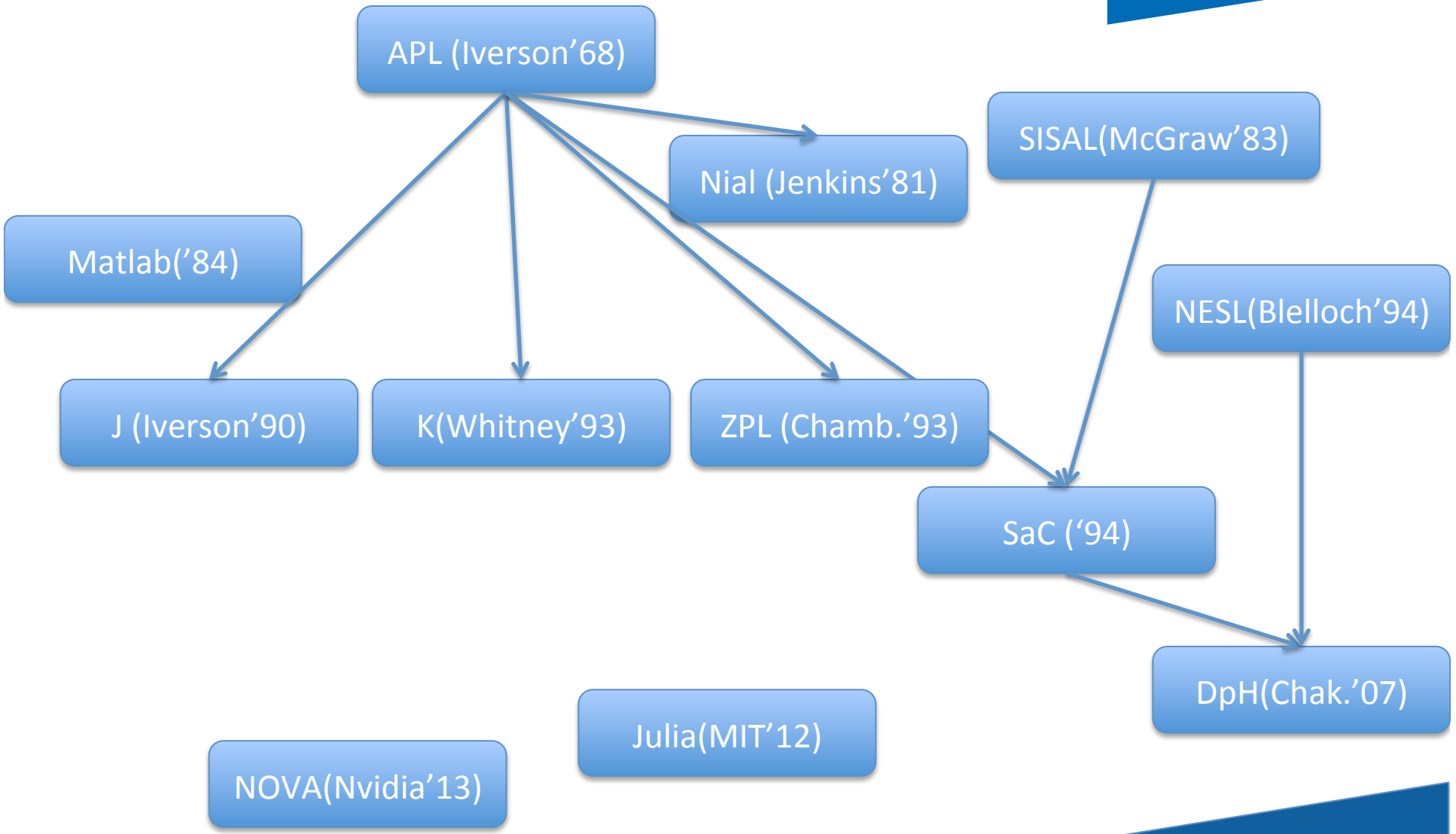
Think Arrays!

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

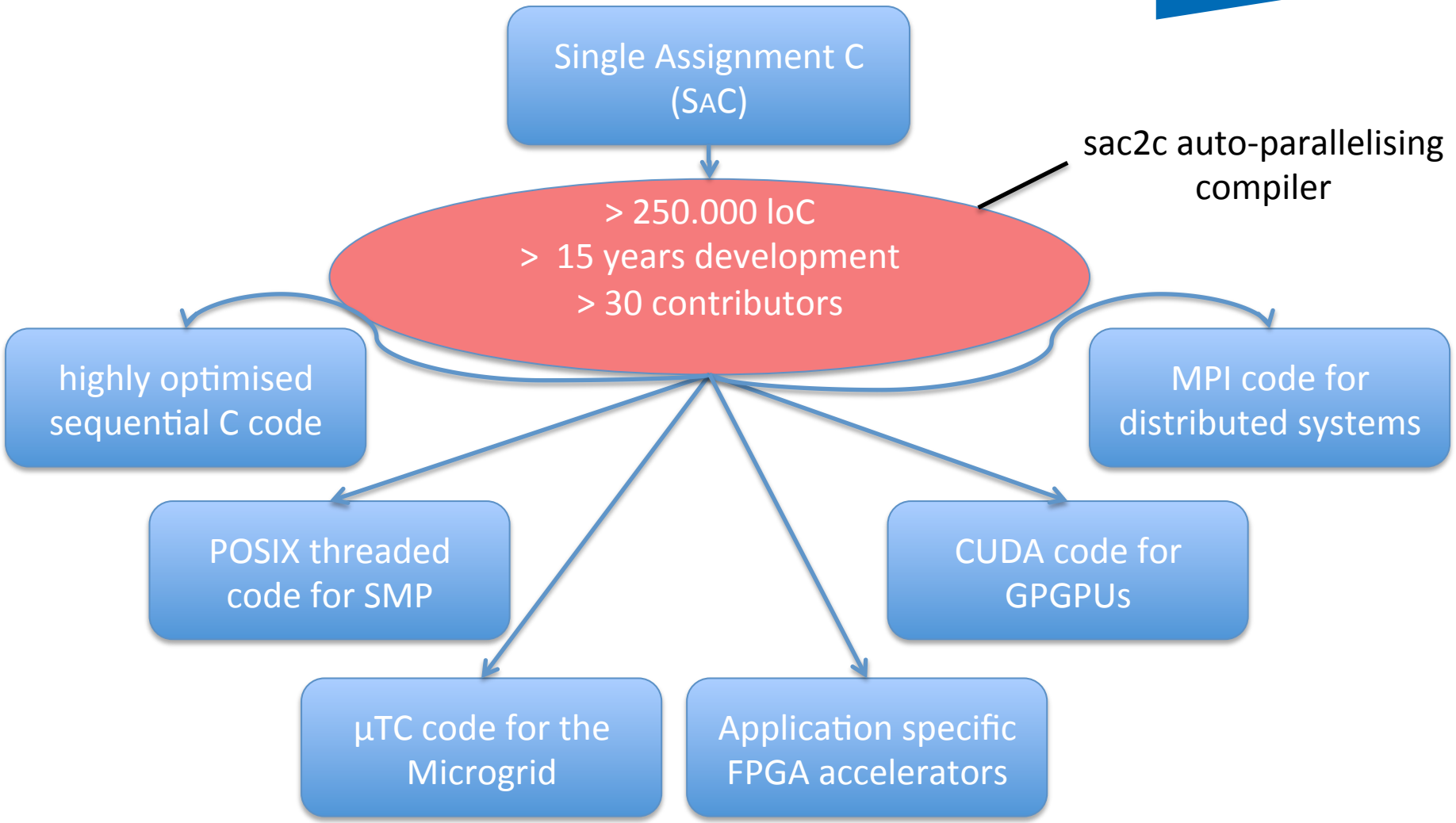
- Even scalars are arrays.
- Any operation maps arrays to arrays.
- Even iteration spaces are arrays

- Graphs are arrays.
- Streams are arrays.
- ...

Some History



The Big Picture



SAC: HP² Driven Language Design

HIGH-PRODUCTIVITY

- easy to learn
  - C-like look and feel
- easy to program
  - Matlab-like style
  - OO-like power
  - FP-like abstractions
- easy to integrate
  - light-weight C interface

&

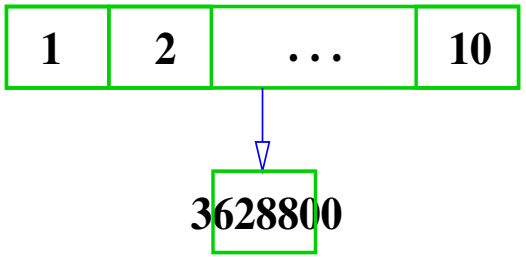
HIGH-PERFORMANCE

- no frills
  - lean language core
- performance focus
  - strictly controlled side-effects
  - implicit memory management
- concurrency apt
  - data-parallelism at core

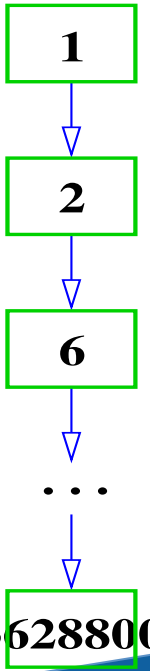
Data-Parallelism, First Steps

Formulate algorithms in space rather than time!

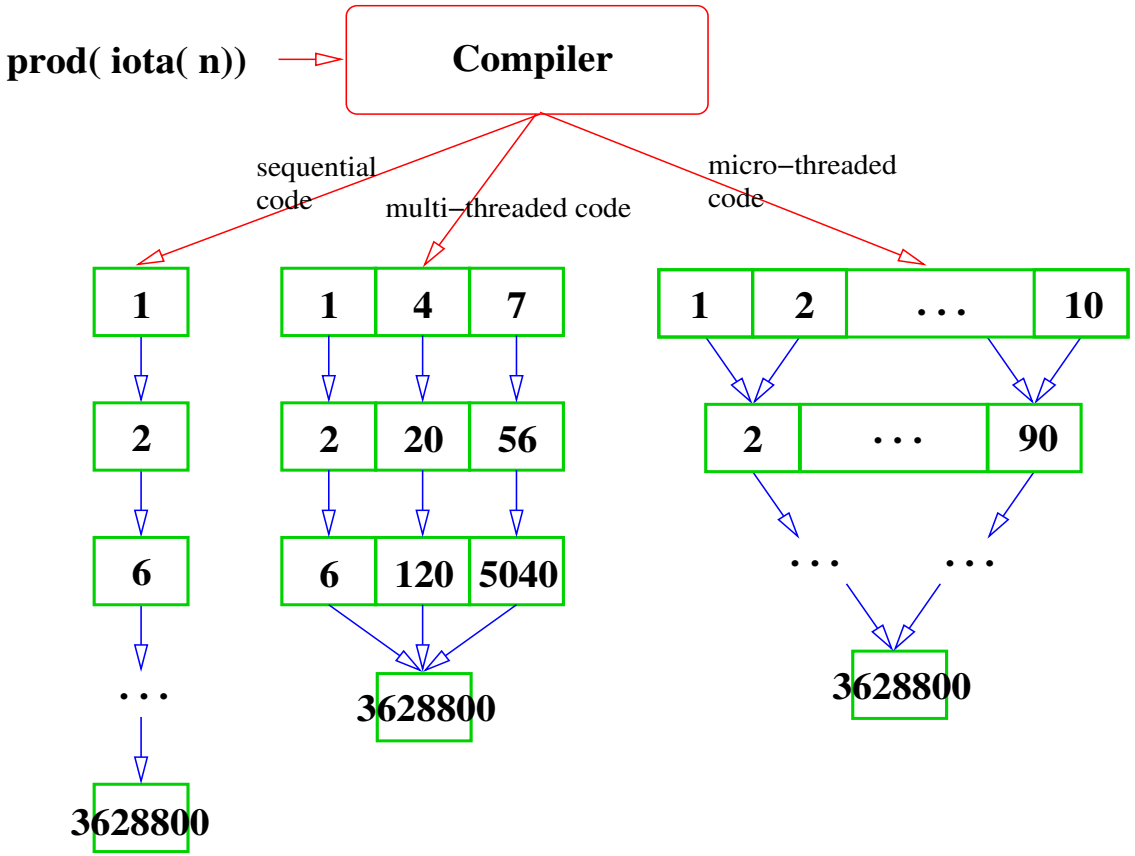
```
prod = prod( iota( 10)+1)
```




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

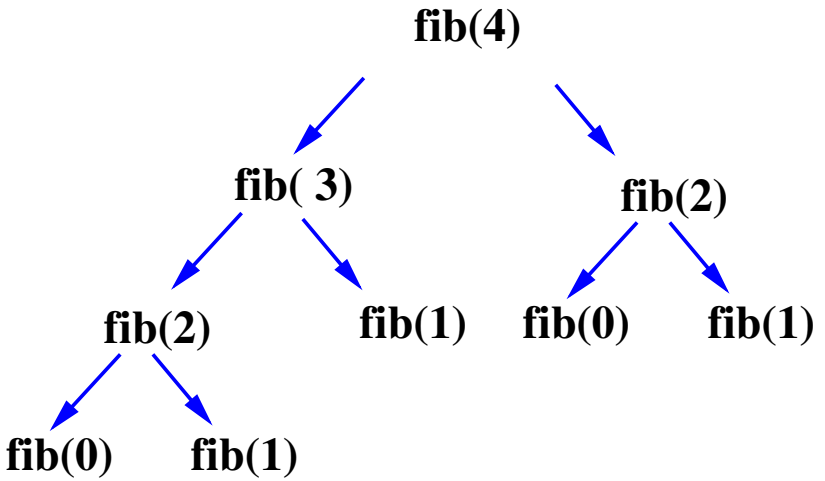


Why is Space Better than Time?



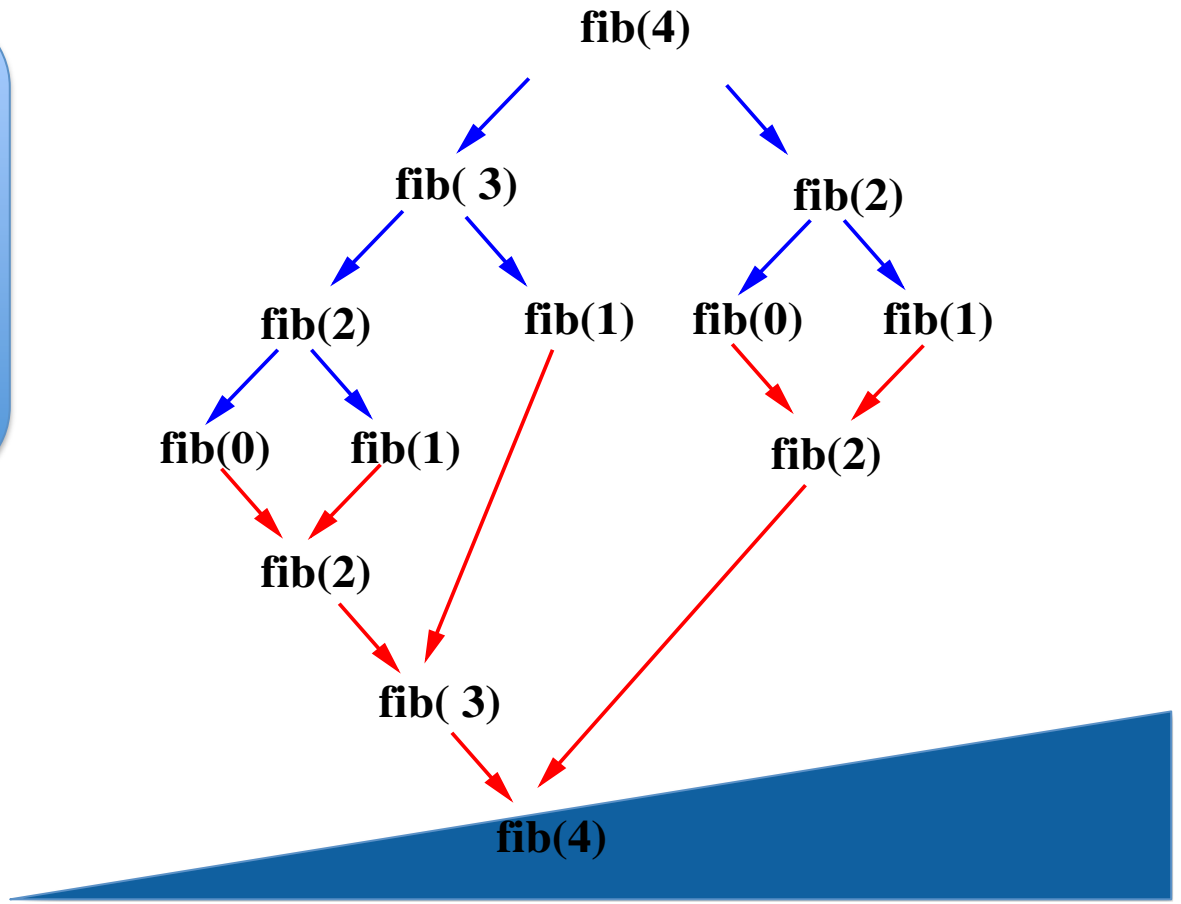
Another Example: Fibonacci Numbers

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



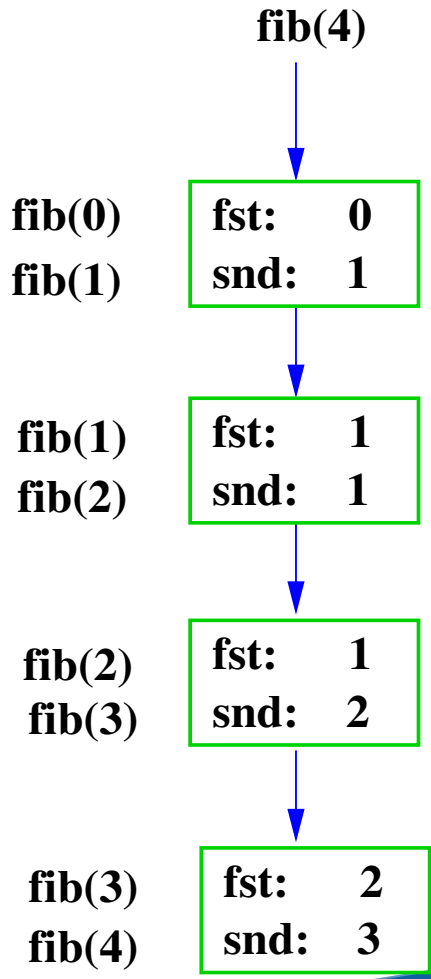
Another Example: Fibonacci Numbers

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



Fibonacci Numbers – now linearised!

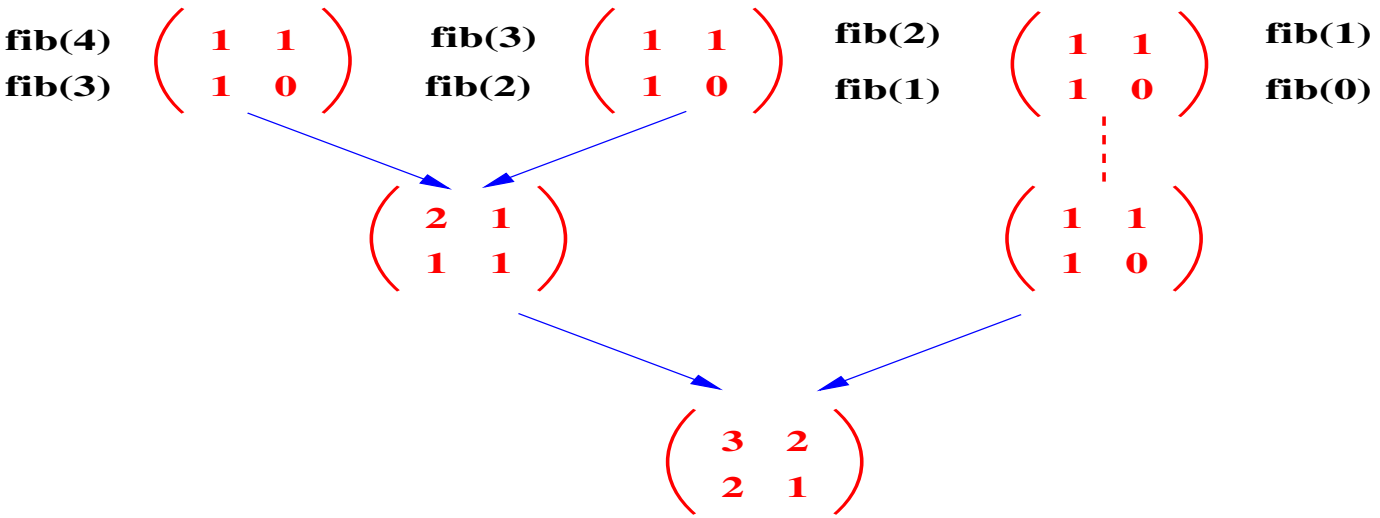
```
if( n== 0)
  return fst;
else
  return fib( snd, fst+snd,
n-1)
```



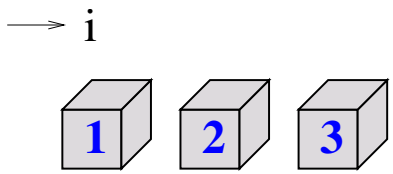
Fibonacci Numbers – now data-parallel!

HERIOT WATT UNIVERSITY

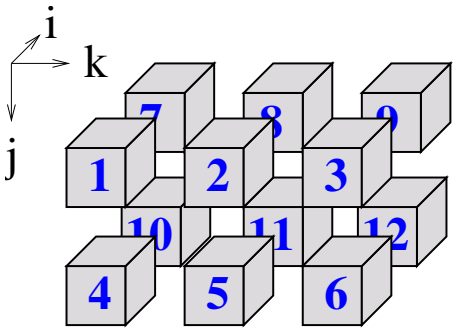
```
matprod( genarray( [n], [[1, 1], [1, 0]])) [0,0]
```



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]

Basic Operations

HERIOT WATT UNIVERSITY

```
dim(42) == 0
dim( [1,2,3]) == 1
dim([[1, 2, 3],
     [4, 5, 6]] ) == 2


shape( 42) == []
shape( [1, 2, 3]) == [3]
shape( [[1, 2, 3],
        [4, 5, 6]] ) == [2, 3]

a = [[1, 2, 3],
     [4, 5, 6]];
a[[1,0]] == 4
a[[1]] == [1,5,6]
a[[]] == a
```

The Usual Suspects

```
a = [1,2,3];
b = [4,4,2];

a+b == [5,6,5];
a<=b == [true, true, false];
sum(a) == 6
...
```

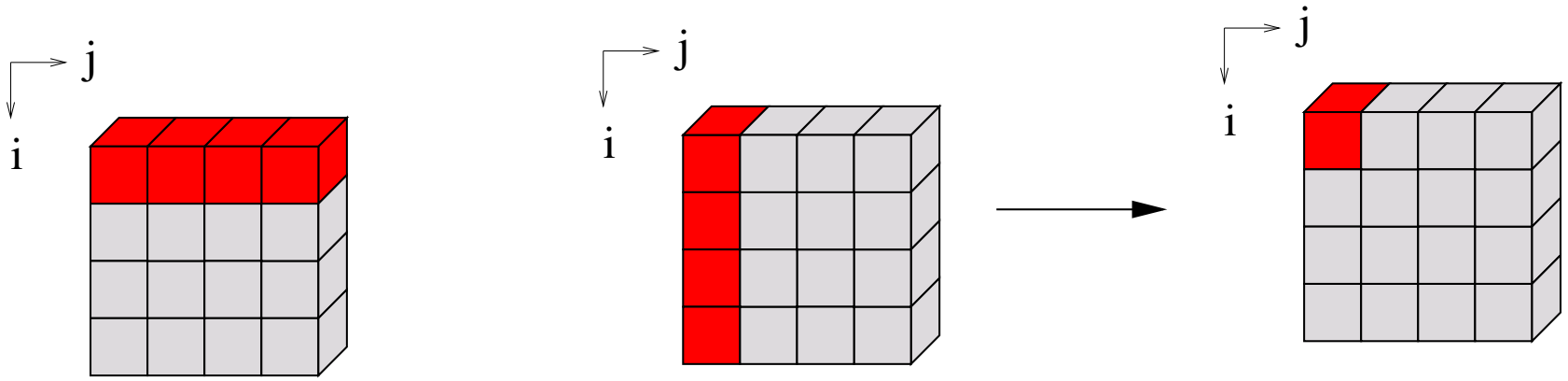
Matlab/ APL stuff

```
a = [[1,2,3],
     [4,5,6]];

take( [2,1], a) == [[1],
                    [4]]
take( [1], a) == [[1,2,3]] != [1,2,3]
take( [], a) == a

take( [-1, 2], a) == [[4,5]]

...
```

Set Notation

```
{ iv -> a[iv] + 1 } == a + 1

{ [i,j] -> mat[[j,i]] } == transpose( mat)

{ [i,j]->(i==j? mat[[i,j]]: 0)}
```

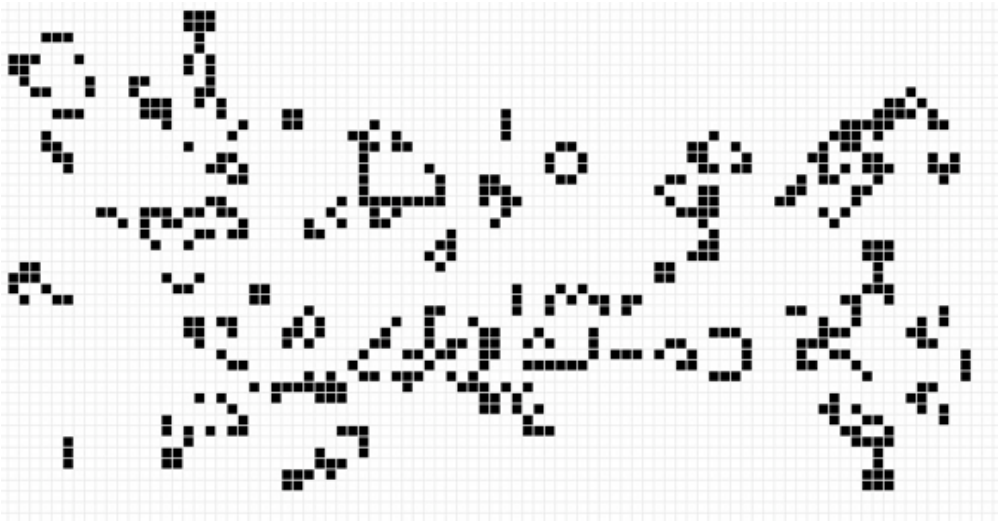
Example: Matrix Multiply



$$(AB)_{i,j} = \sum_k A_{i,k} * B_{k,j}$$

```
{ [i,j] -> sum( A[[i,.]]  * B[[.,j]]) }
```

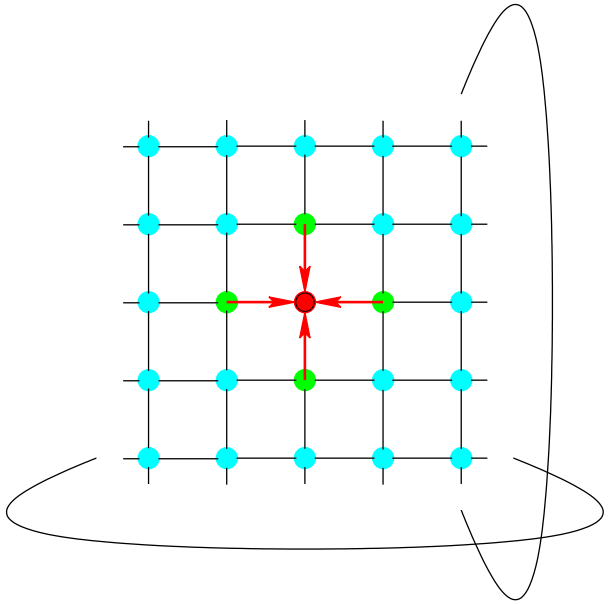
Example: Game Of Life



```
a = with {
      ( . < iv < .) {
        nbs = sum (  tile( [3,3], iv-1, a));
        state = toi( (nbs == 3) || ((a[iv] ==1) && (nbs == 4)));
      } : state;
    } : modarray( a);
```

Example: Relaxation

$$\begin{pmatrix} 0 & 1/8 & 0 \\ 1/8 & 4/8 & 1/8 \\ 0 & 1/8 & 0 \end{pmatrix}$$



```
weights = [ [0d, 1d, 0d], [1d, 4d, 1d], [ 0d, 1d, 0d]] / 8d
mat = ...
res = { [i,j] -> sum(
        { iv -> weights[iv] * rotate( iv-1, mat)}
                        [[...,i,j]] ) };
```

Index-Free Combinator-Style Computations

HERIOT WATT UNIVERSITY

L2 norm:

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

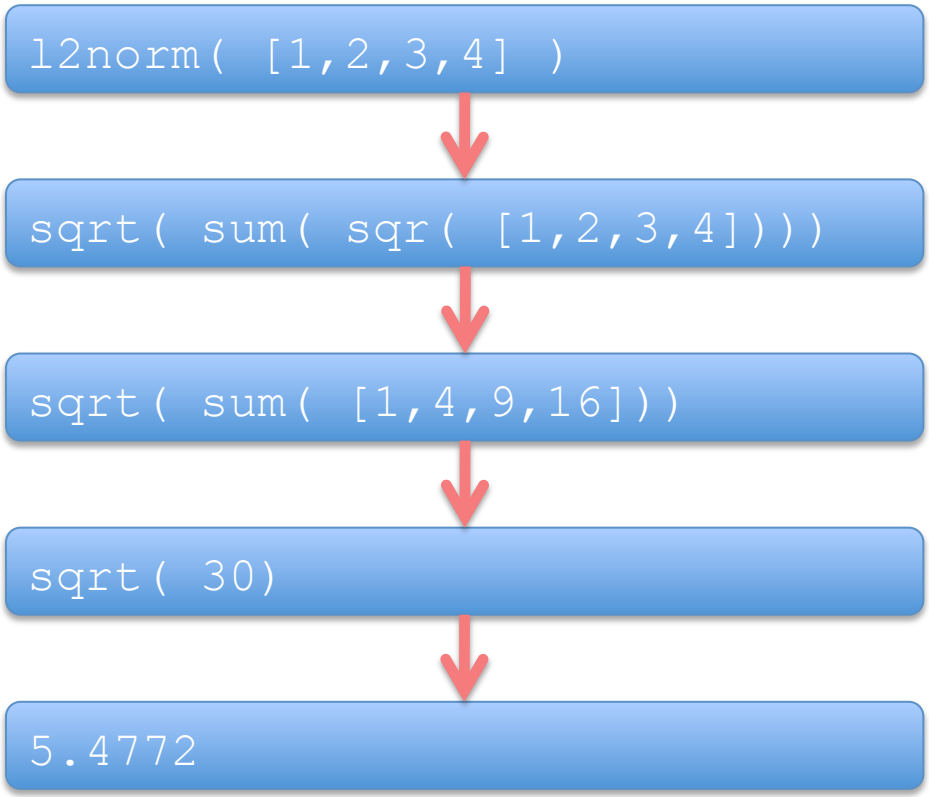
Convolution step:

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

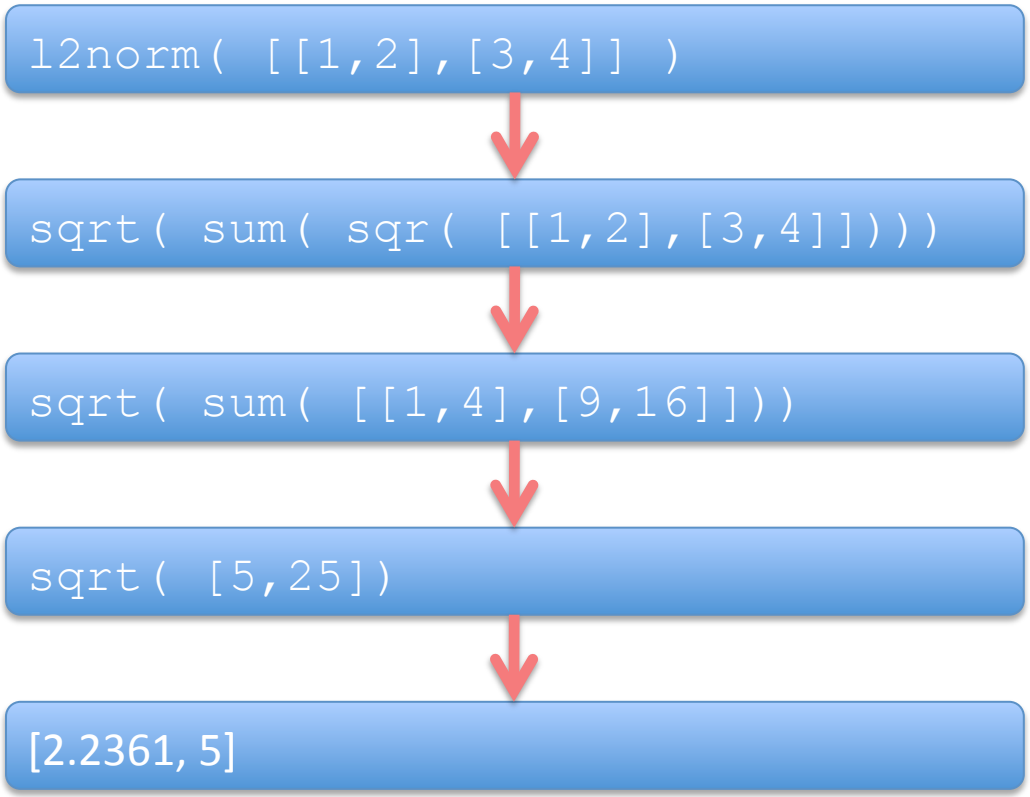
Convergence test:

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

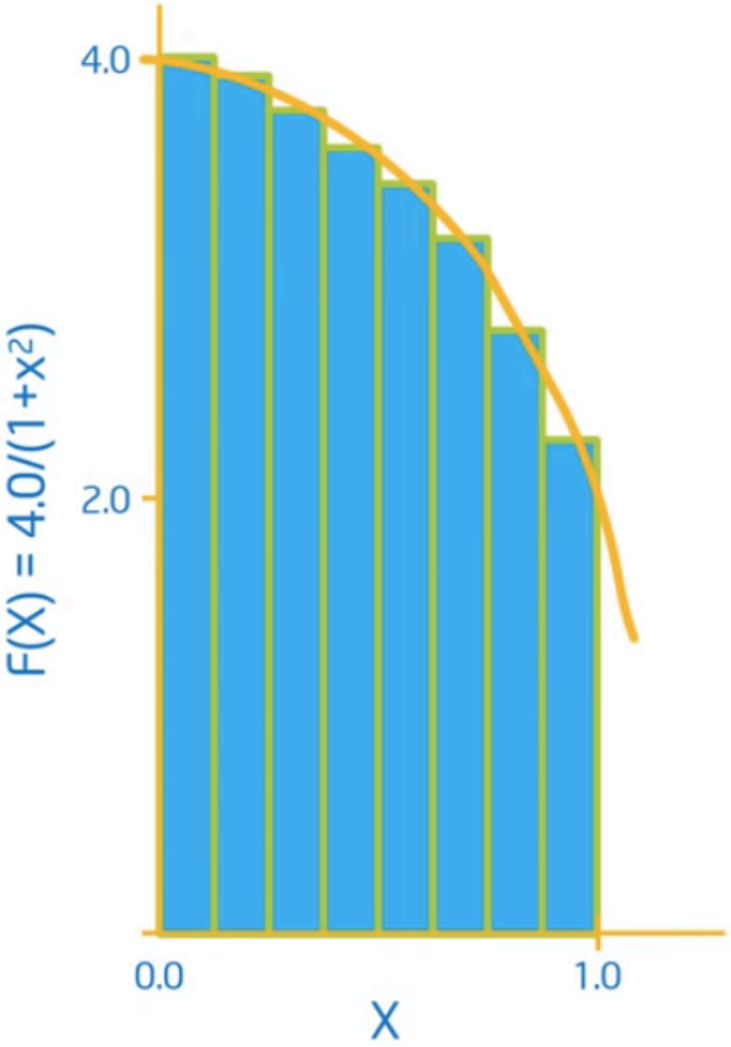
Shape-Invariant Programming



Shape-Invariant Programming



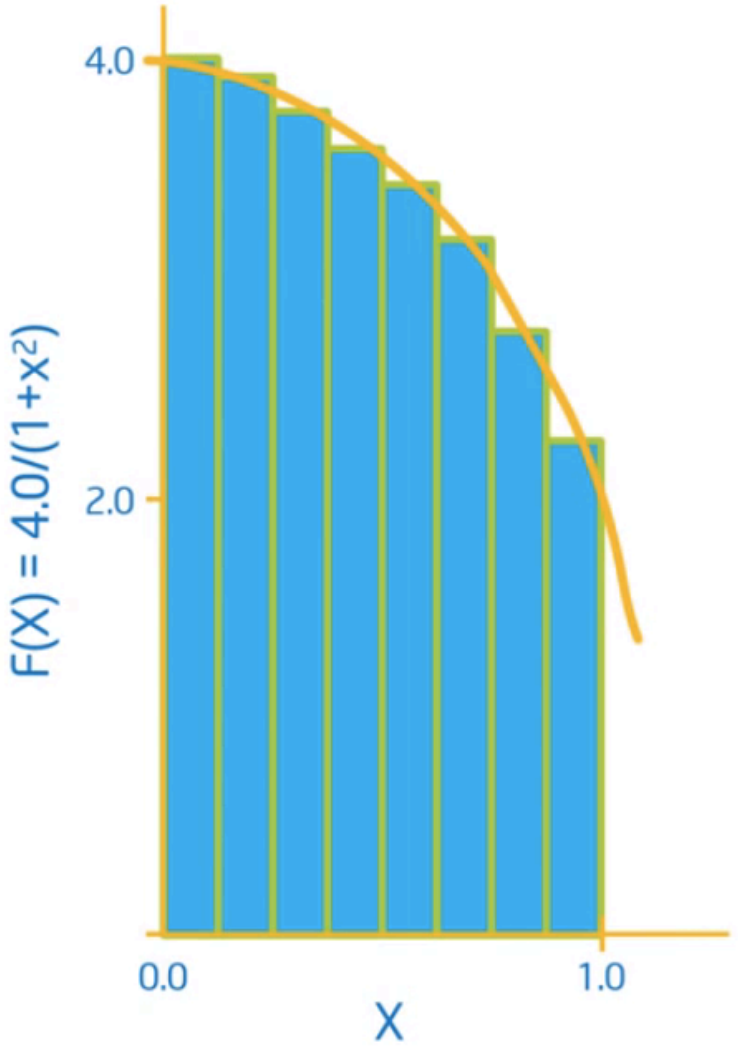
Computation of π



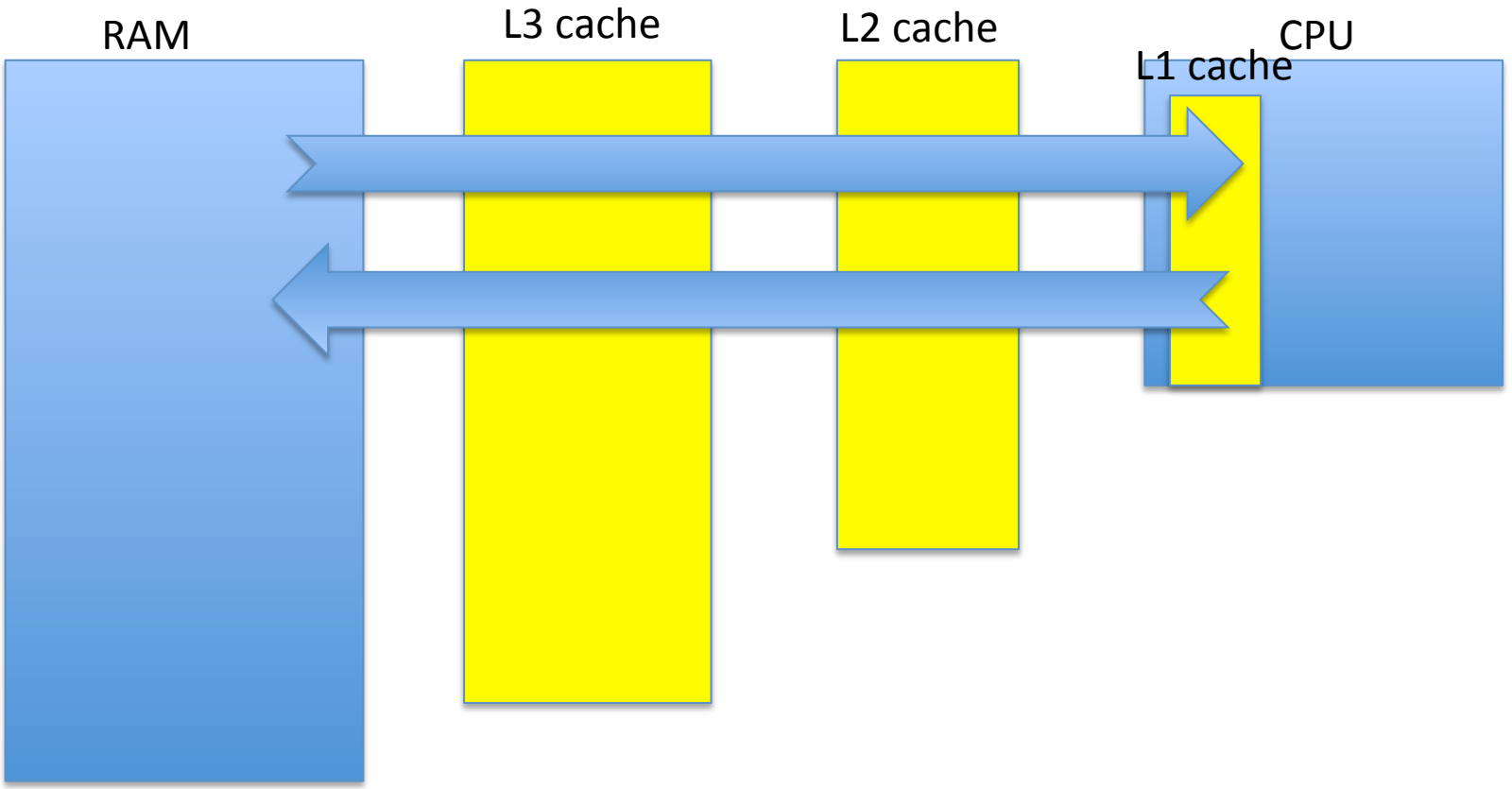
$$\int_0^1 \frac{4.0}{(1+X^2)}$$

Computation of π

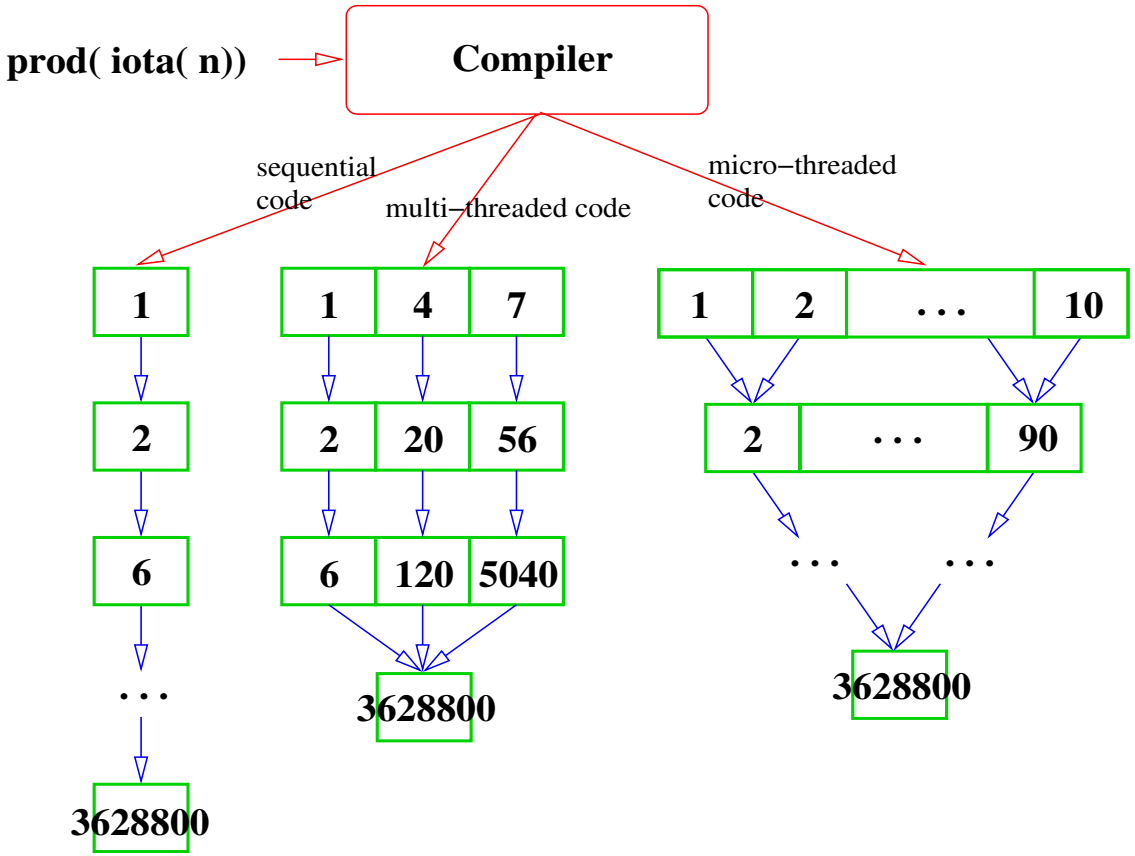
```
double f( double x)
{
   return 4.0 / (1.0+x*x);
}

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);
}
```



Arrays and Streams



Target-Specific Streaming



Streaming as High-Level Code Transformation


```
a = reshape( [M,N], iota( M*N));
b = reshape( [N,K], iota( N*K));
res = matmul(a, b);
```

```
sa = reshape( [2,M/2,N], a);
sb = reshape( [N,2,K/2], b);
res3tp = { [i,j] -> matmul( sa[i,.,.], sb[.,j,.]) };
res3 = reshape( [M,K], { [a,b,c,d] -> res3tp[ a,c,b,d] } );

print( all( res == res3));
```

```
sa = reshape( [2,M/2,N], a);
sb = b;
res2 = reshape( [M,K], { [i] -> matmul( sa[i,.,.], sb) });

print( all( res == res2));
```

…. Then streaming boils down to…..

implementing one or more axes of the array in time rather than in space!

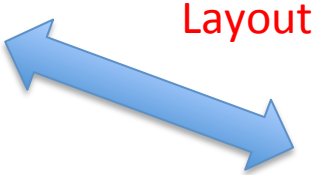
What if we add infinite axes??

Alternative Layouts - Vectorisation

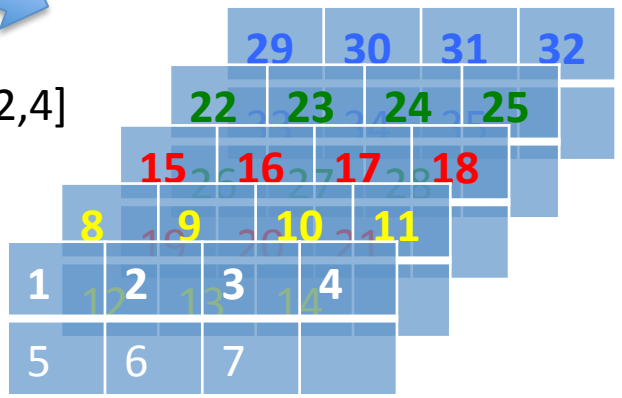
1	2	3	4	5	6	7
8	9	10	11	12	13	14
15	16	17	18	19	20	21
22	23	24	25	26	27	28
29	30	31	32	33	34	35
36	37	38	39	40	41	42

Shape: [6,7]

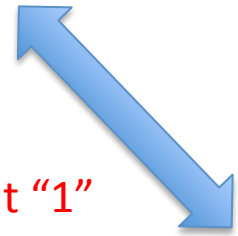
Layout "2"



Shape: [6,2,4]

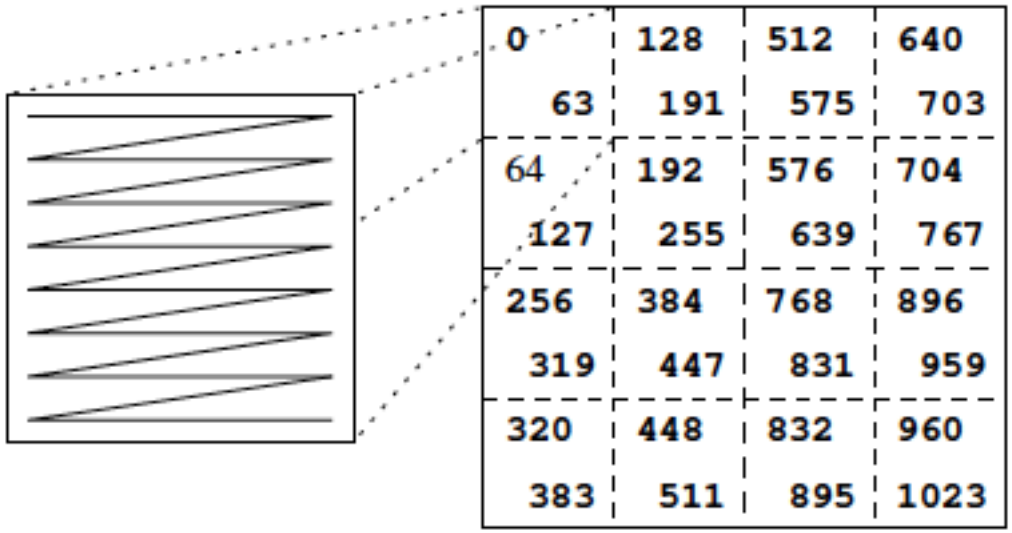


Layout "1"



Shape: [2,7,4]

Alternative Layouts – Morton Order



Reshape from [32,32] into [4,4,8,8]…….

Programming Array-Style offers:

HERIOT WATT UNIVERSITY

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

try it out: www.sac-home.org