MANCHESTER 1824

The University of Manchester

DFScala: High Level Dataflow Support for Scala

Daniel Goodman, Salman Khan, Chris Seaton,
Yegor Gusgov, Behram Khan,
Mikel Luján and Ian Watson

Workshop on Data-Flow Execution Models for Extreme Scala Computing
23 September 2012
Minneapolis

TERAFLUX

EPSRC
Engineering and Physical Sciences Research Council

The Teraflux project is funded by the European Commission Seventh Framework Programme under grant agreement number 249013. Chris Seaton is an EPSRC funded student. Mikel Luján is a Royal Society University Research Fellow.

MANCHESTER 1824

The University of Manchester

Context

- Processors with thousands cores
- Simplifying the architecture by minimising cache coherency
- Teraflux is looking at hardware architectures combining dataflow and transactional memory
- This work is looking at implementing the programming model for non-specialist, industrial programmers

MANCHESTER 1824

The University of Manchester

Why Dataflow?

- Simple programming model
- Implicit parallelism
- Deterministic by default
- No shared-state
- No need for cache coherency

Basics of Scala

- Martin Odersky, EPFL, 2003
- JVM language – similar look and feel, semantics, object model, concurrency model to Java
- Adds anonymous functions, pattern matching, type inference
- Data structures are immutable by default
- Several parallelism models – threads, actors, parallel collections

MANCHESTER 1824

The University of Manchester

Why Scala?

- Pragmatic, high-productivity
- Strong static typing, but dynamic feel through type inference
- Existing Java infrastructure and libraries
- Motivates and facilitates a functional approach, but doesn't enforce it
- A familiar language for many programmers

MANCHESTER 1824

The University of Manchester

DFScala Model

- Data-driven
- Coarse-grained
- Nodes are functions
- Graph is dynamic
- Graph is statically checked for type correctness
- Graphs can be nested
- Avoid pitfalls of parallelism, make dataflow simpler

MANCHESTER 1824

The University of Manchester

DFScala Library

- Dataflow nodes are Scala functions
- Type-safety using inference
- Nodes are runnable when the function's arguments are available
- Java thread pool
- Shared-state scheduler using software transactional memory

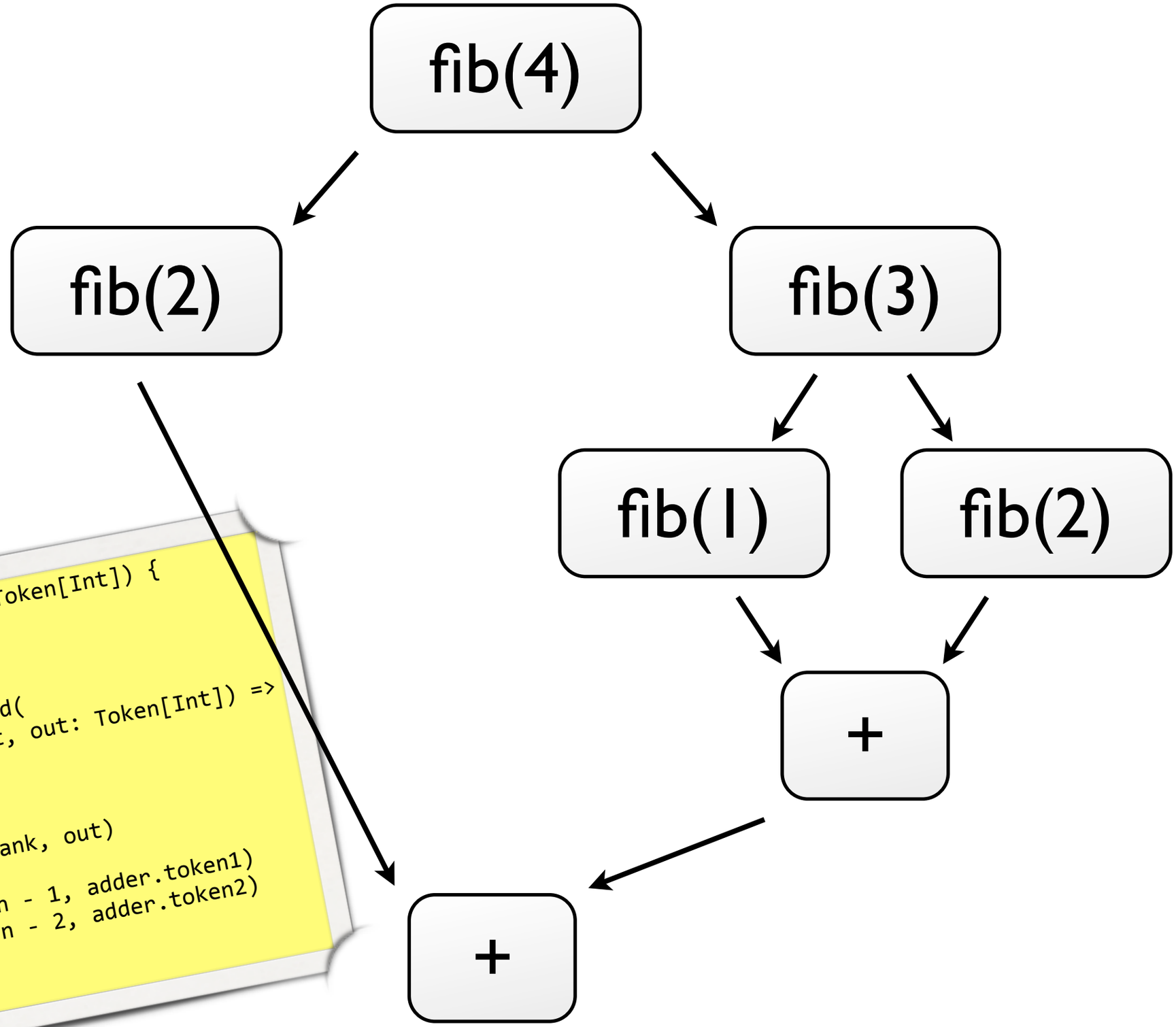
```
def fib(n : Int) : Int =
  if (n <= 2)
    1
  else
    fib(n-1) + fib(n-2)
```

```
def fib(n : Int, out: Token[Int]) {
  if (n <= 2)
    out(1)
  else {
    val adder = thread(
      (a: Int, b: Int, out: Token[Int]) =>
        out(a + b)
    )

    adder(Blank, Blank, out)

    thread(fib _, n - 1, adder.token1)
    thread(fib _, n - 2, adder.token2)
  }
}
```

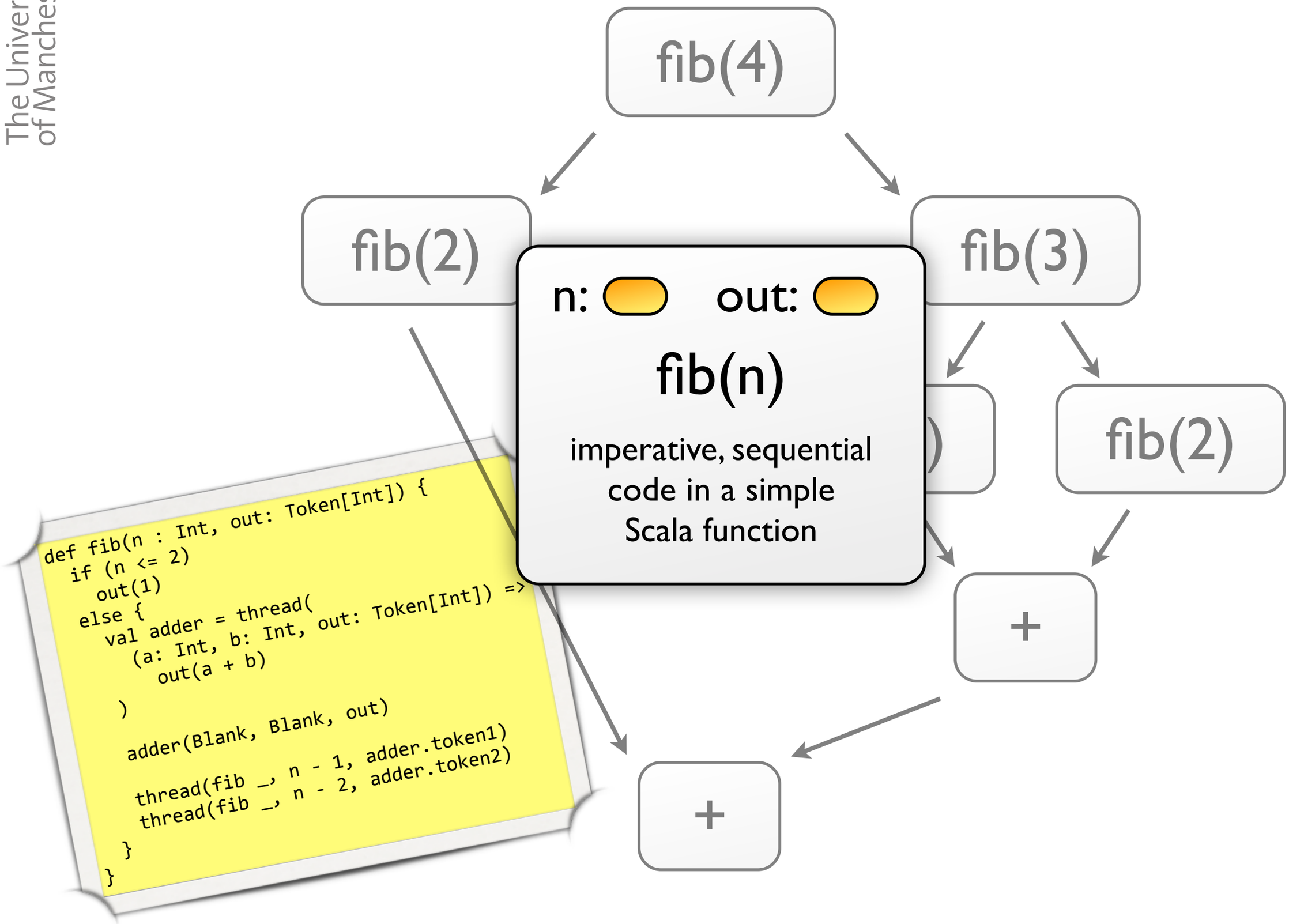
```
def fib(n : Int, out: Token[Int]) {
  if (n <= 2)
    out(1)
  else {
    val adder = thread(
      (a: Int, b: Int, out: Token[Int]) =>
        out(a + b)
    )

    adder(Blank, Blank, out)

    thread(fib _, n - 1, adder.token1)
    thread(fib _, n - 2, adder.token2)
  }
}
```

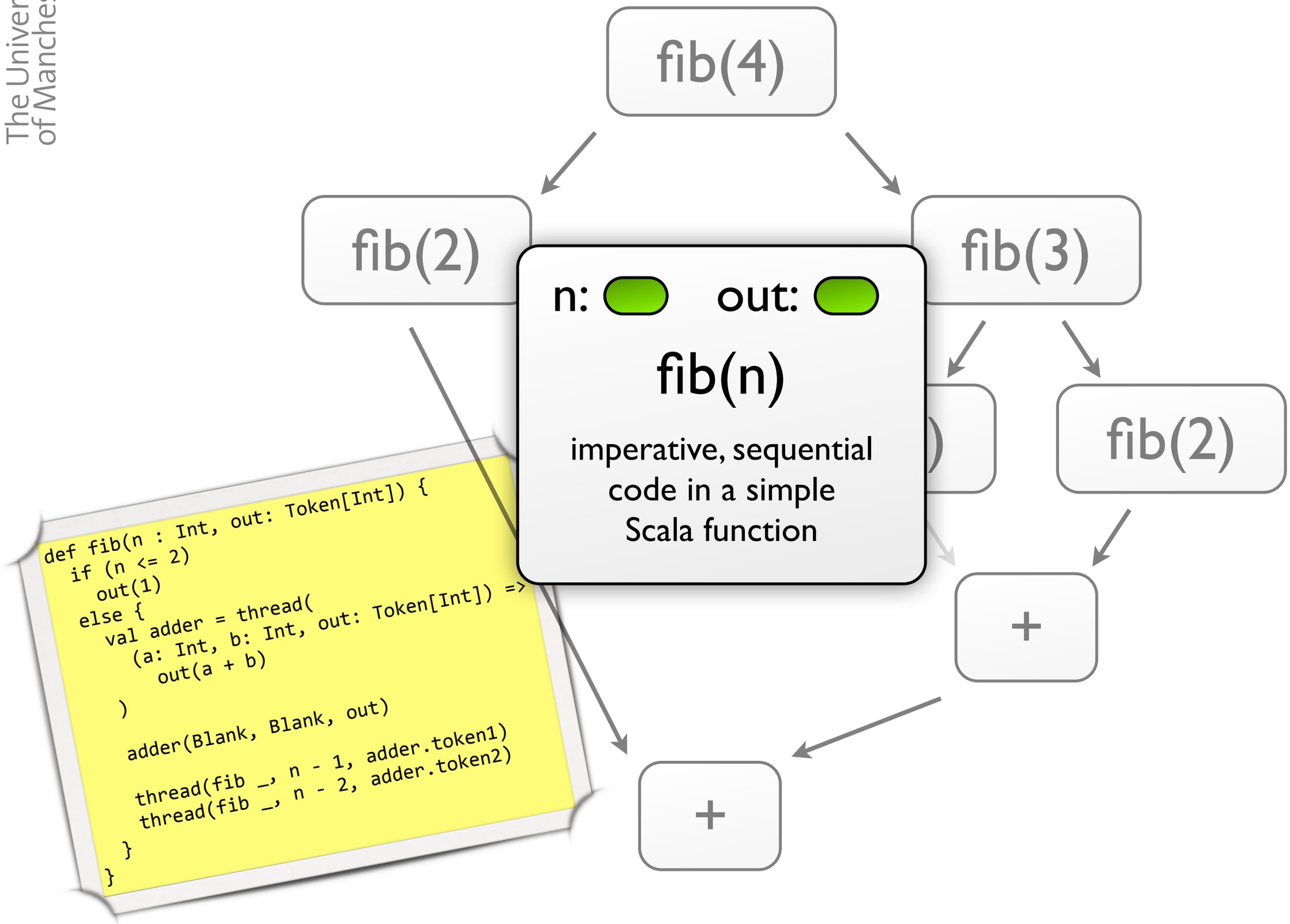
```
def fib(n : Int, out: Token[Int]) {
  if (n <= 2)
    out(1)
  else {
    val adder = thread(
      (a: Int, b: Int, out: Token[Int]) =>
        out(a + b)
    )

    adder(Blank, Blank, out)

    thread(fib _, n - 1, adder.token1)
    thread(fib _, n - 2, adder.token2)
  }
}
```

```
def fib(n : Int, out: Token[Int]) {
  if (n <= 2)
    out(1)
  else {
    val adder = thread(
      (a: Int, b: Int, out: Token[Int]) =>
        out(a + b)
    )

    adder(Blank, Blank, out)

    thread(fib _, n - 1, adder.token1)
    thread(fib _, n - 2, adder.token2)
  }
}
```




```
def fib(n : Int, out: Token[Int]) {
  if (n <= 2)
    out(1)
  else {
    val adder = thread(
      (a: Int, b: Int, out: Token[Int]) =>
        out(a + b)

    )

    adder(Blank, Blank, out)

    thread(fib _, n - 1, adder.token1)
    thread(fib _, n - 2, adder.token2)
  }
}
```




fib(4)
fib(2)
fib(3)
n:
out:
fib(n)
imperative, sequential code in a simple Scala function
fib(2)
+
+
def fib(n : Int, out: Token[Int]) {
if (n <= 2)
out(1)
else {
val adder = thread(
(a: Int, b: Int, out: Token[Int]) =>
out(a + b)
)
adder(Blank, Blank, out)
thread(fib _, n - 1, adder.token1)
thread(fib _, n - 2, adder.token2)
}
}

Benefits

- Programmers only need to worry about functions and the usual function boundaries
- Use existing arguments instead of I-structures
- No mutable state
- Graph is statically checked for type safety
- Can be used with existing code and libraries

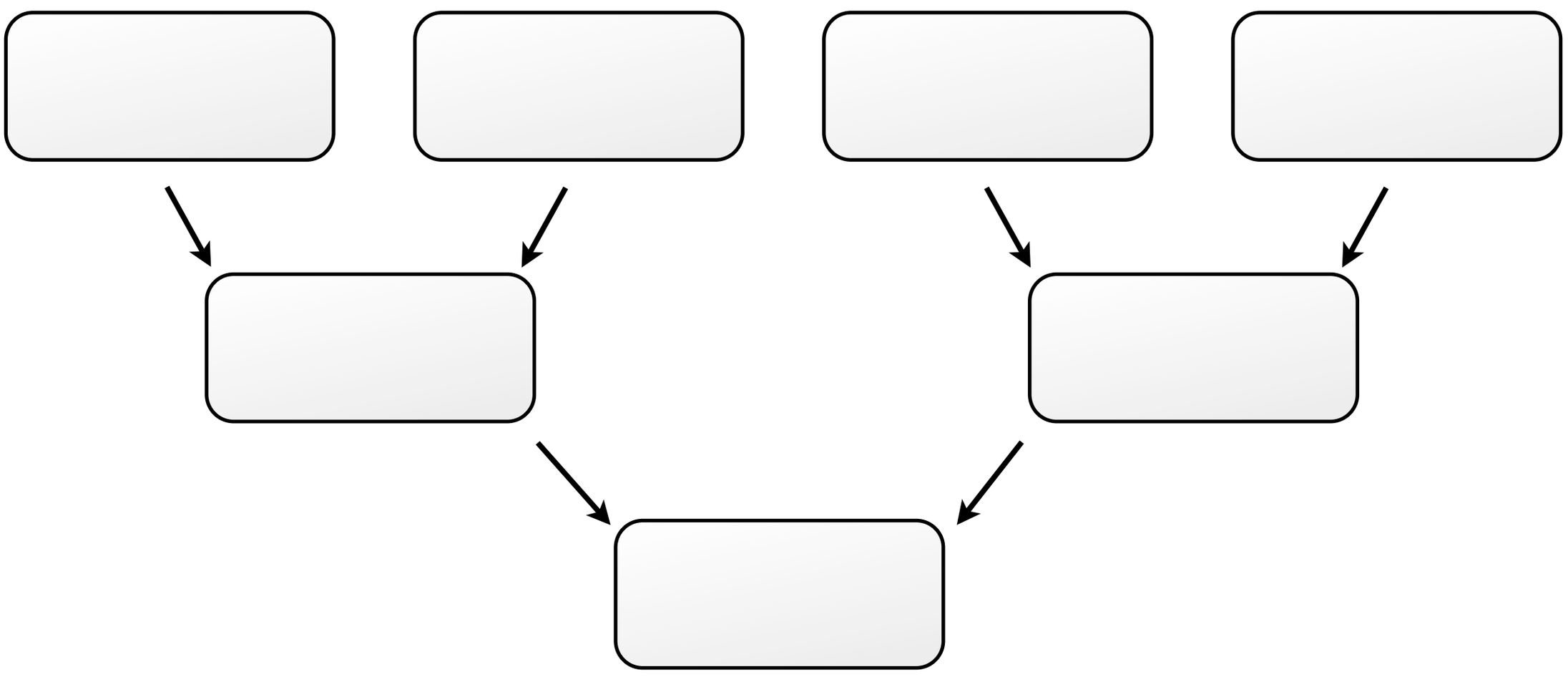
MANCHESTER 1824

The University of Manchester

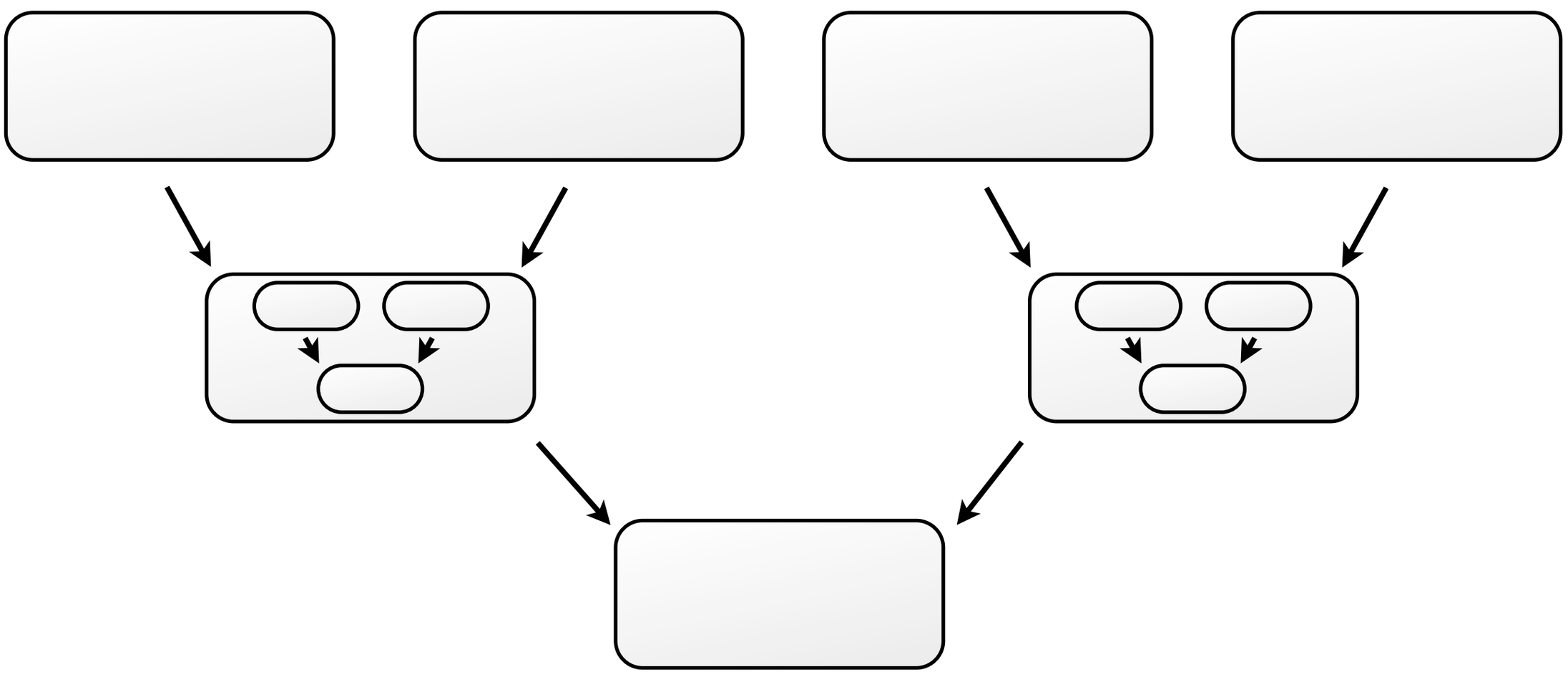
Nested Graphs

- DFScala by default does not allow for any parallelism within nodes
- Each function forming a node is entirely sequential and runs from start to finish without pauses
- As an explicit exception, dataflow graphs can be nested within nodes

Nested Graphs



Nested Graphs



Compare to Akka

- Dataflow library for Scala
- Part of a framework for large, distributed, concurrent applications (think Erlang)
- Released after we started work
- Based on the Oz model
- New concepts – dataflow variables (I-structures)
- Concurrency within functions using continuations

MANCHESTER 1824

The University of Manchester

Compare to Akka

```
def fib(n: Int): Int = {
  val r, a, b = Promise[Int]()
  flow { r << a() + b() }
  flow { a << fib(n - 1) }
  flow { b << fib(n - 2) }

  return r()
}
```

Compare to CNC-Scala

- Port of C++ Intel Concurrent Collections to Scala
- Also have a focus on 'parallelism oblivious developers' and productivity
- Parallelism is within collections
- Each collection element is an I-structure, dependencies between them form the dataflow graph
- Also concurrency in functions using continuations

MANCHESTER 1824

The University of Manchester

Comparisons

	DFScala	Akka	CNC
Dataflow	✓	✓	✓
Scala	✓	✓	✓
Dynamic	✓	✓	✓
Task parallel	✓	✓	✓
Pure Scala	✓	✓	
Special dataflow variables		✓	✓

Summary

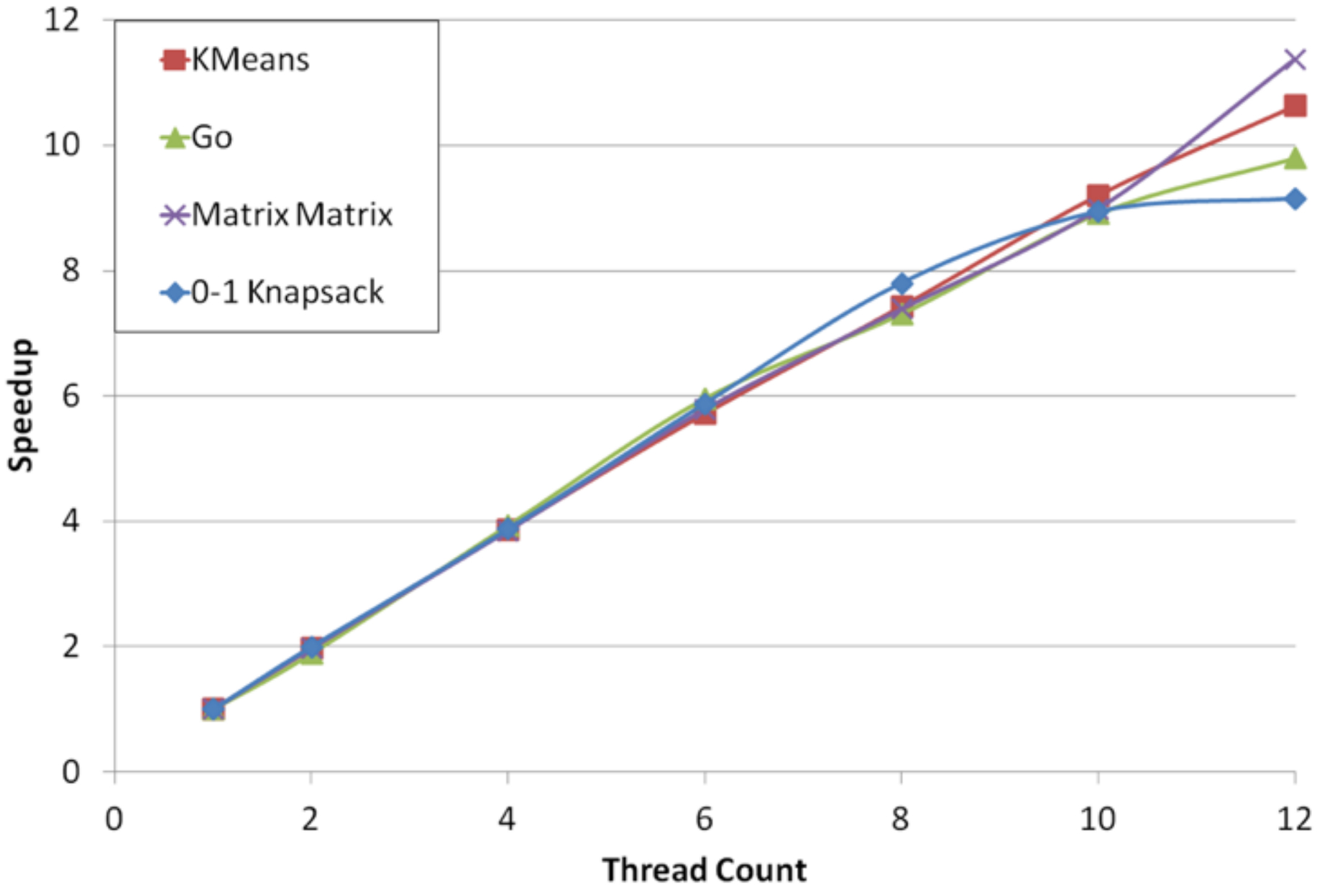
- Focus on programmability
- Coarse-grained
- Nodes are written as Scala functions
- Edges are normal function arguments
- Real software, plus tooling, available today

MANCHESTER 1824

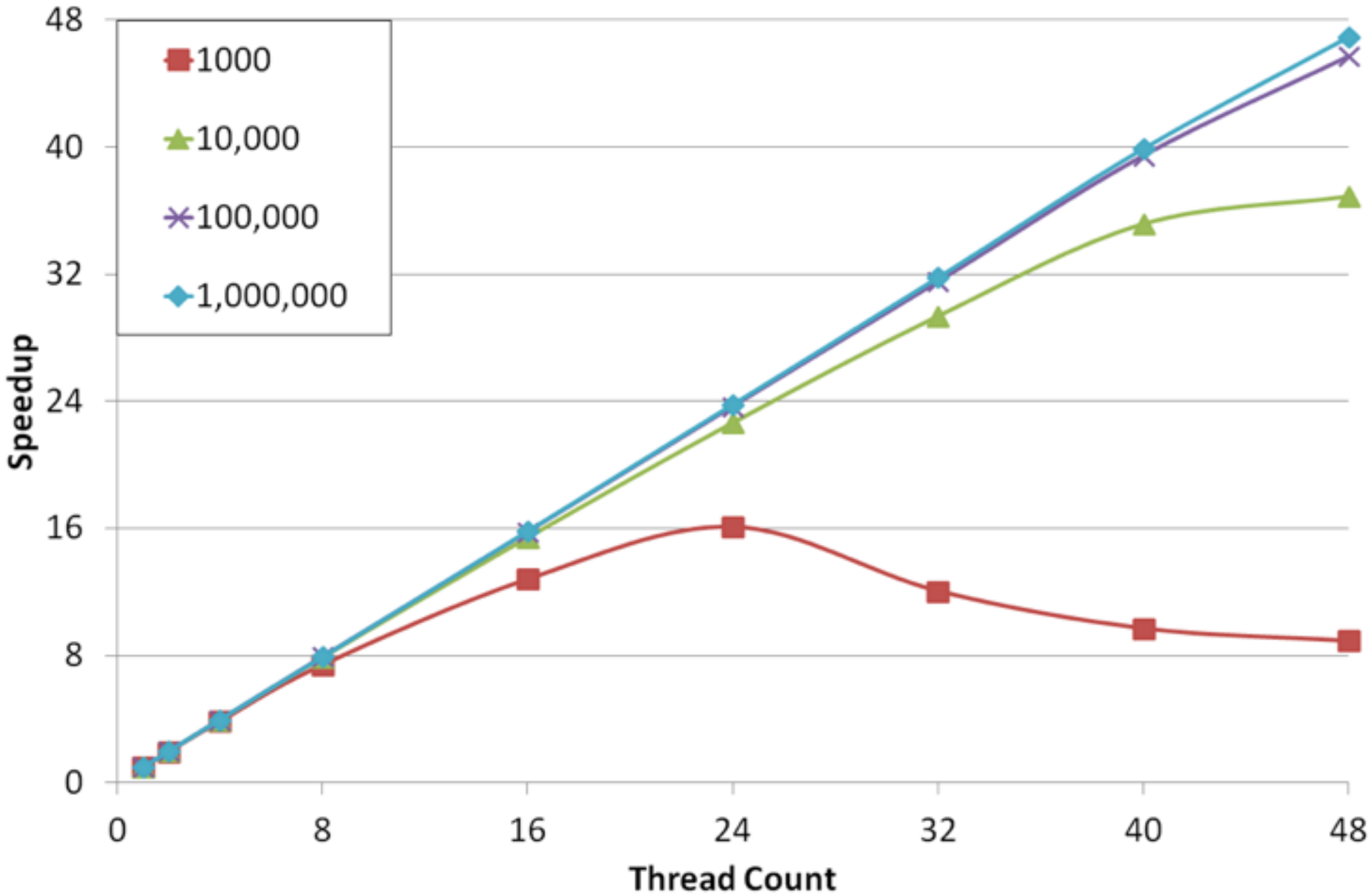
The University of Manchester

Benchmarks

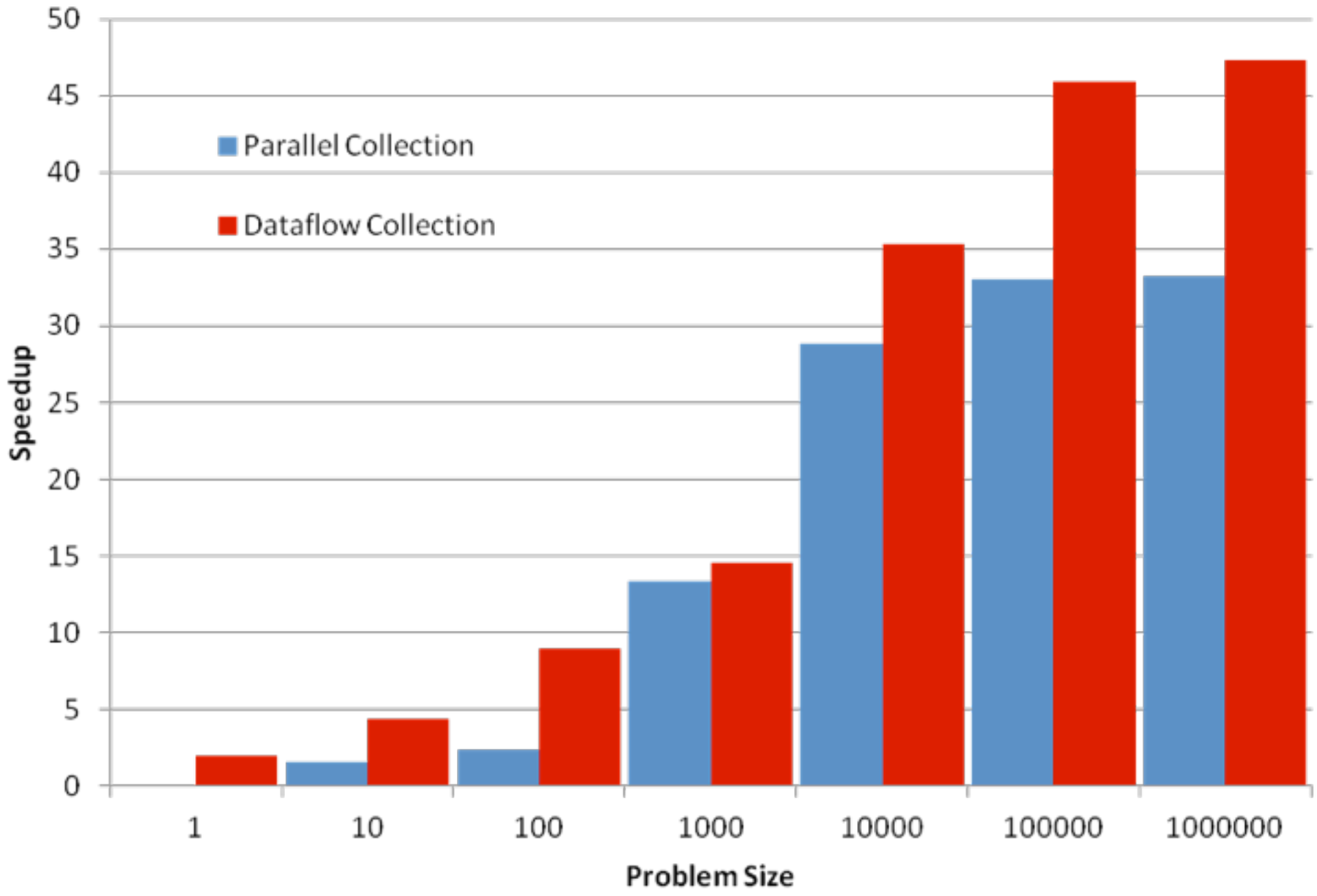
- Matrix Multiplication
- KMeans
- 0-1 Knapsack
- Go
- Parallel collections



Benchmark Speedup, 2x 6-core AMD Opteron



Collections Speedup, 4x 12-core AMD Opteron



Collections Compared to Scala Parallel Collections, 4x 12-core AMD Opteron

Tooling

- Pure Scala library
- Loggers for analysing runtime performance
- Graphical debugger

MANCHESTER 1824

The University of Manchester

DFScala Scope

Search Google or enter an address

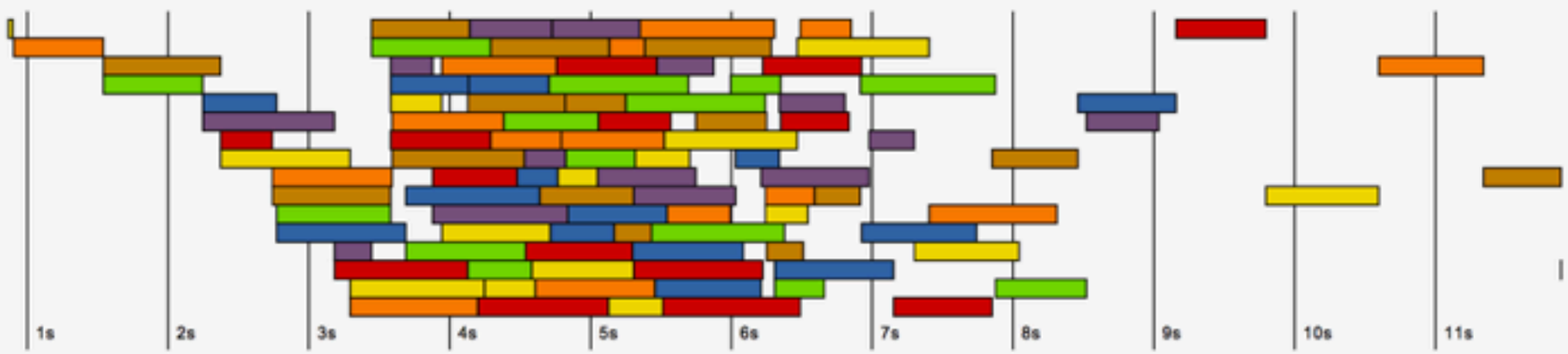
Reader

DFScala Scope | example | Connect | Example | Table | Graph

Util 34.54%

dataflow program finished
11.83s #2 finished
11.83s #2 started
11.83s #4 finished
11.83s #4 to #2 arg 1
11.28s #4 started
11.28s #10 finished
11.28s #10 to #4 arg 1
10.54s #10 started
10.54s #13 finished
10.54s #13 to #10 arg 1
9.74s #13 started
9.74s #34 finished
9.74s #34 to #13 arg 1

Thread	Parent	Received	Sent	Children	Created	Started	Finished	Duration	State
#1	root	root	#3	#2 #3	0.74s	0.81s	0.84s	0.03s	finished
#2	#1	#4			0.81s	11.83s	11.83s	0.00s	finished
#3	#1	#1	#5 #6	#4 #5 #6	0.81s	0.84s	1.48s	0.64s	finished
#4	#3	#7 #10	#2		1.48s	11.28s	11.83s	0.55s	finished
#5	#3	#3	#11 #12	#10 #11 #12	1.48s	1.48s	2.31s	0.83s	finished
#6	#3	#3	#8 #9	#7 #8 #9	1.48s	1.48s	2.18s	0.70s	finished
#7	#6	#19 #16	#4		2.18s	7.24s	7.96s	0.74s	finished
#8	#6	#6	#17 #18	#16 #17 #18	2.18s	2.19s	2.71s	0.52s	finished
#9	#6	#6	#20 #21	#19 #20 #21	2.18s	2.19s	3.12s	0.93s	finished
#10	#5	#22 #13	#4		2.31s	10.54s	11.28s	0.74s	finished



MANCHESTER 1824

The University of Manchester

Available Today

http://apt.cs.man.ac.uk/projects/TERAFLUX/DFScala/

or Google for 'manchester dfscala'

- Currently used in research into combining dataflow with transactional memory
- Research into irregular parallelism

MANCHESTER 1824

The University of Manchester

DFScala in a Nutshell

- Focus on programmability
- Coarse-grained
- Nodes are written as Scala functions
- Edges are normal function arguments
- Graphs are statically checked for type-safety
- Real software, plus tooling, available today
- Empirical results show it is scalable and can be applied to make a high performance collections library