University of St Andrews

SEVENTH FRAMEWORK PROGRAMME

Megacore, Megafast, Megacool? Functional Patterns of Parallelism

Kevin Hammond, Chris Brown, Vladimir Janjic

University of St Andrews, Scotland

Erlang User Conferencce, Stockholm, June 12, 2015

THE FRAMEWORK PROGRAMME FOR RESEARCH AND INNOVATION HORIZON 2020

T: *@rephrase_eu, @khstandrews*

E: *kh@cs.st-andrews.ac.uk, kevin@kevinhammond.net*

W: `http://www.paraphrase-ict.eu`
`http://www.rephrase-ict.eu`

cost EPSRC sicsa*

PARAPHRASE

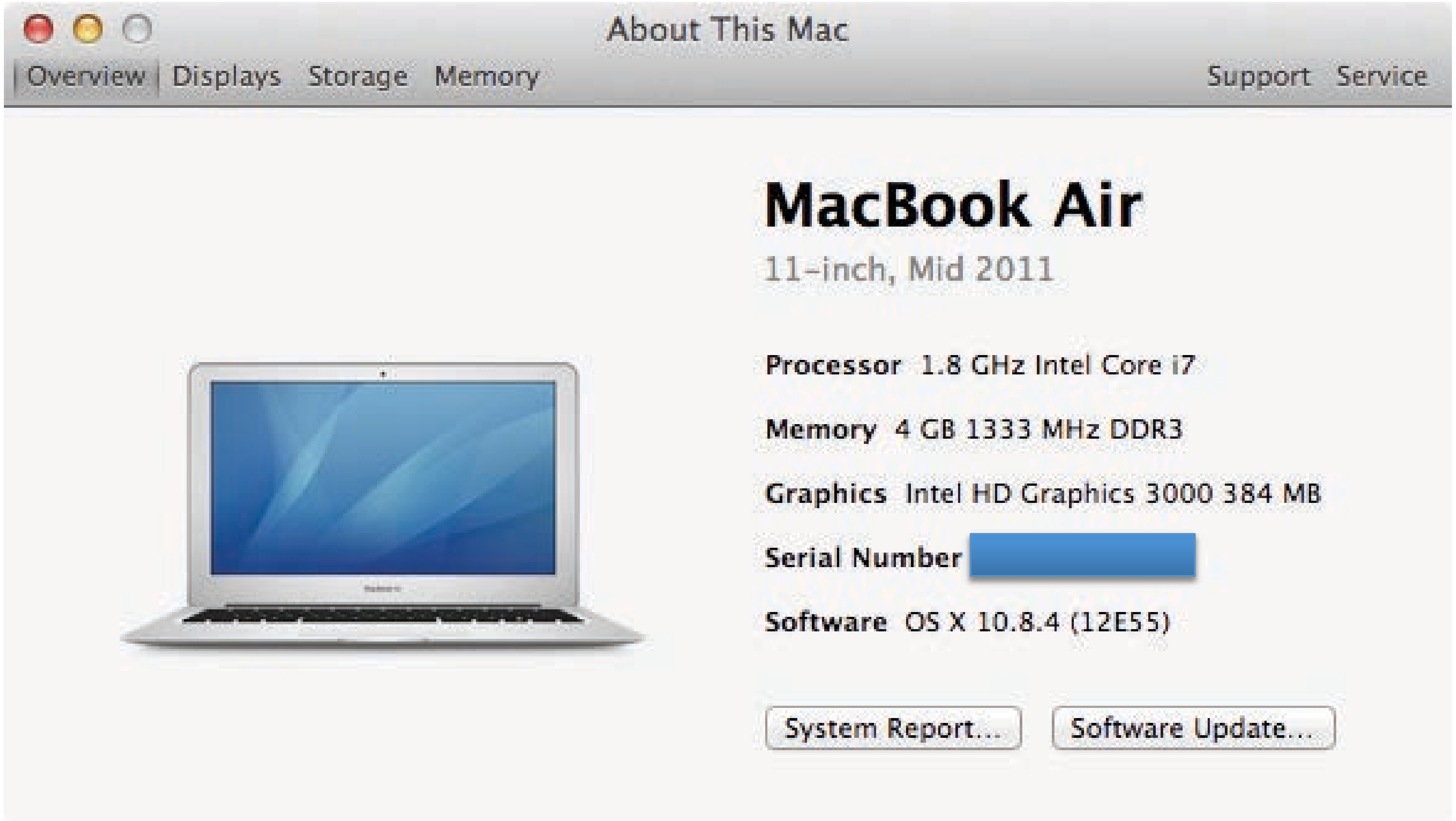
ParaPhrase Project: Parallel Patterns for Heterogeneous Multicore Systems (ICT-288570), 2011-2015, €4.2M budget

13 Partners, 8 European countries

UK, Italy, Germany, Austria, Ireland, Hungary, Poland, Israel

Coordinated by @khstandrews

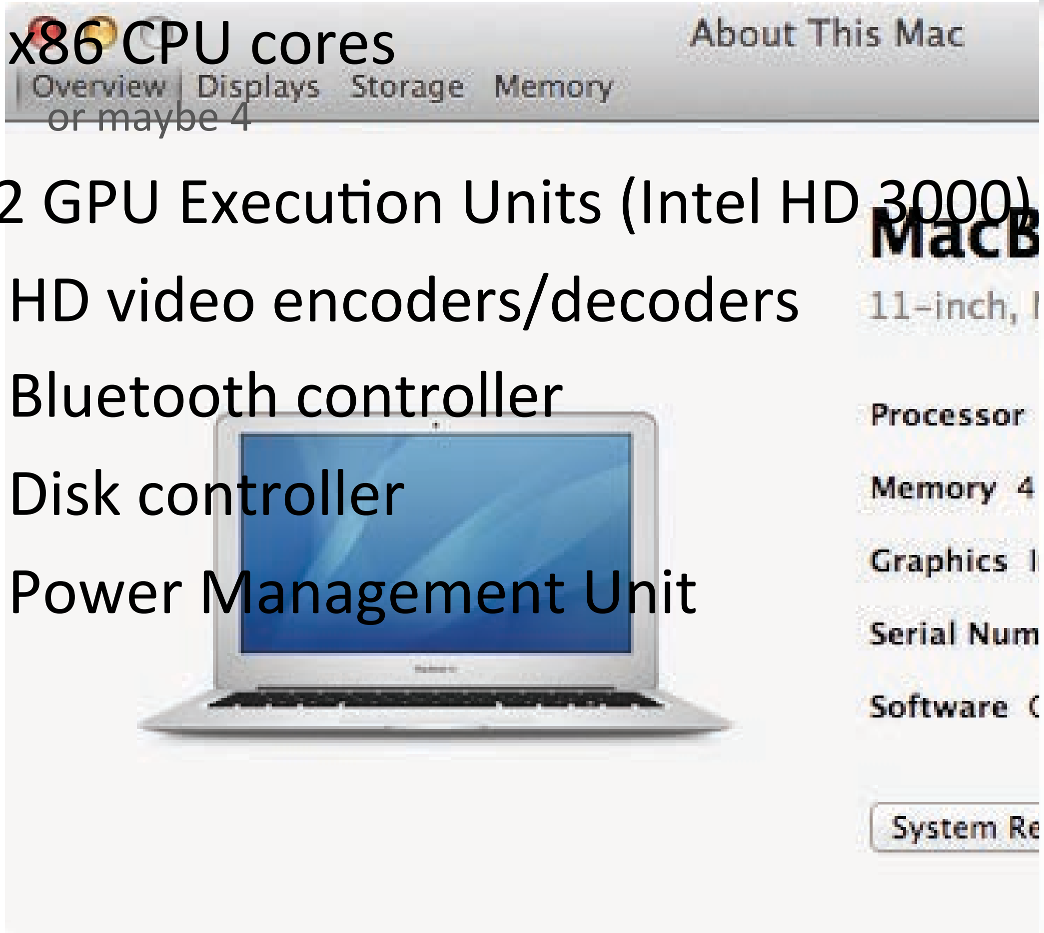
Multicore is now ubiquitous

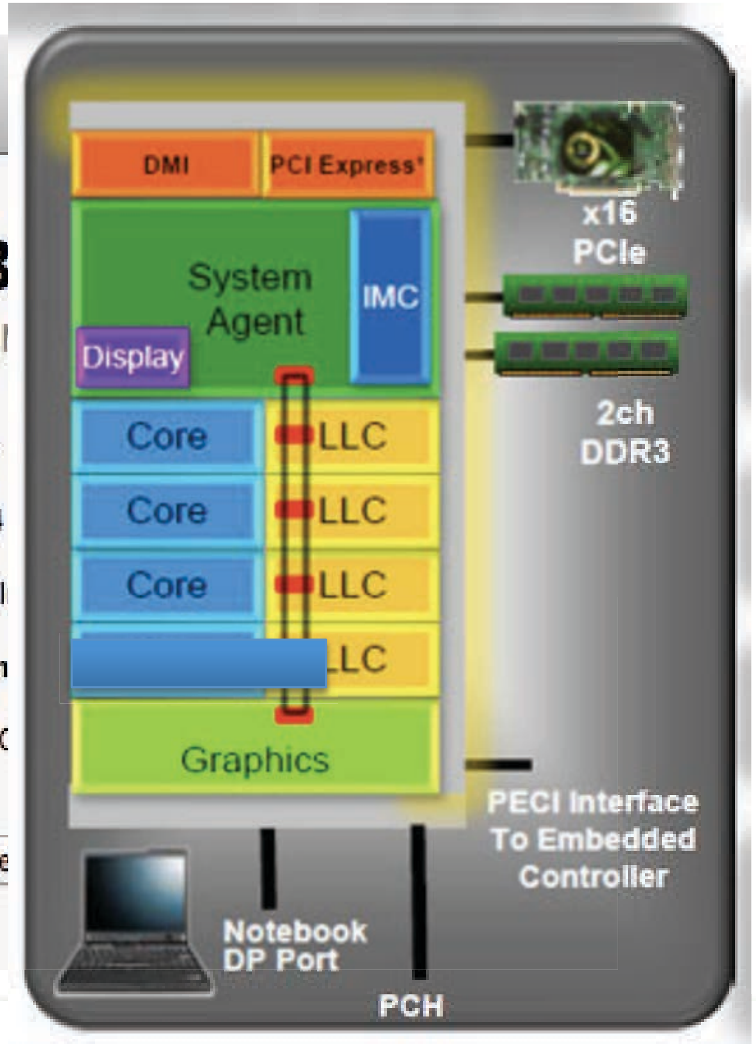


How Many Cores does my laptop have?

University of St Andrews

- 2 x86 CPU cores
  - or maybe 4
- 12 GPU Execution Units (Intel HD 3000)
- 2 HD video encoders/decoders
- 1 Bluetooth controller
- 1 Disk controller
- 1 Power Management Unit
- …





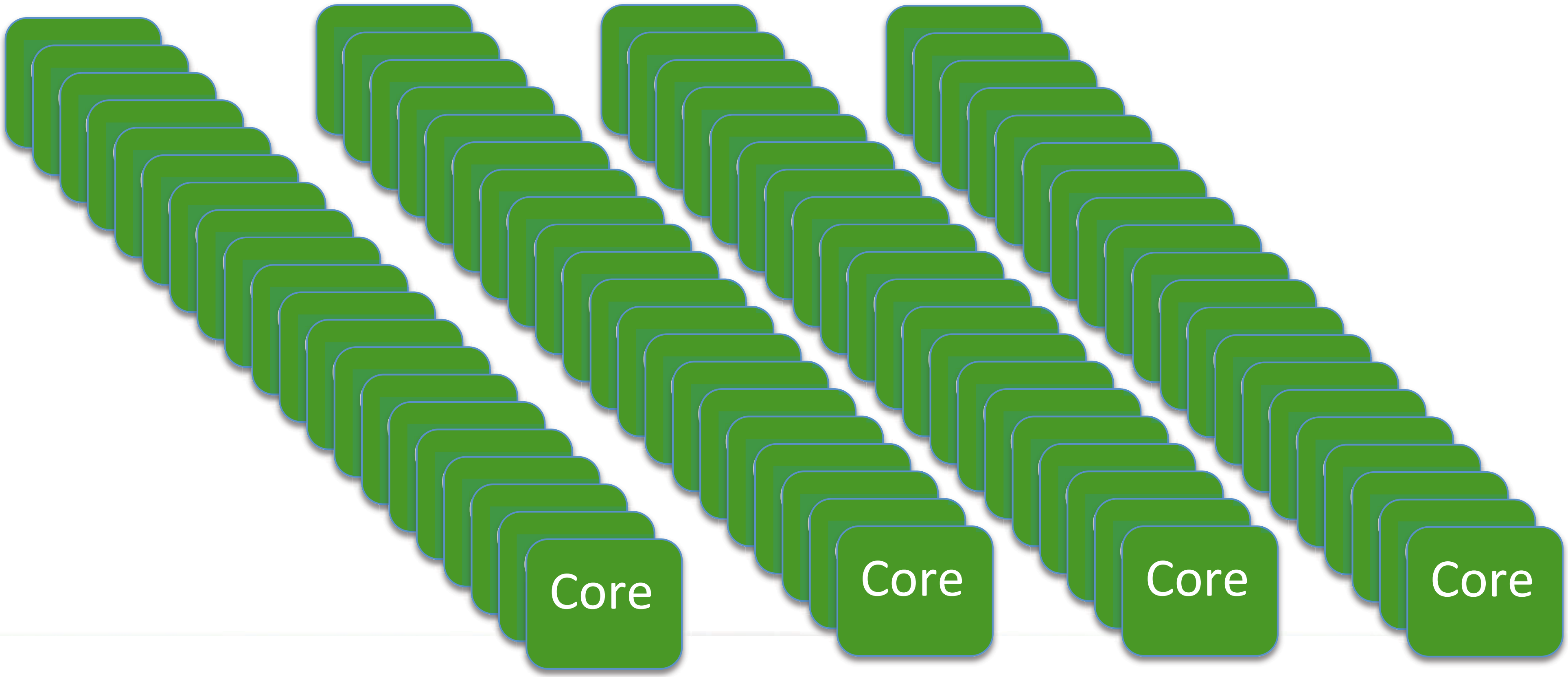
The Present:
From Multicore to Manycore

University of St Andrews

Intel Xeon Phi 7120P: 60 x86 cores (at 1.238GHz), 300W

The Future: “megacore” computers?

- ***Hundreds of thousands, or millions, of (small) cores***



What will "megacore" computers look like?

- **Nodes will be linked into systems**
  - Each nodes will have several large CPU cores
    - plus specialist manycore accelerators
    - ***Highly heterogeneous processor structure***
  - High-performance *network* to link nodes
  - Not much memory per core

- *Dealing with heterogeneity is a major problem!*
  - *Most current models are very difficult to use well*
    - *e.g. CUDA, OpenCL, ...*

- ***Exascale*** **systems will probably be heterogeneous megacores**

The Fastest Computer in the World
June 2013-date

One of the first MEGACORE computers!

Tianhe-2, Chinese National University of Defence Technology

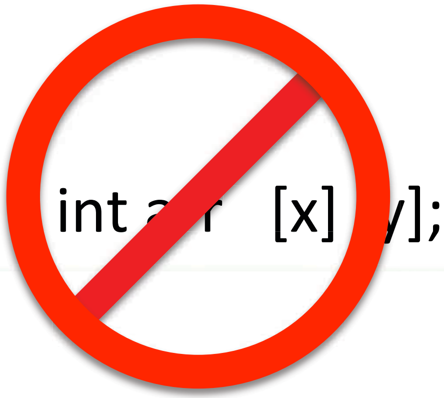
33.86 petaflops/s
16,000 Nodes; each with 2 Ivy Bridge multicores and 3 Xeon Phis
3,120,000 x86 cores in total!!!

What will future “megacore” computers look like?

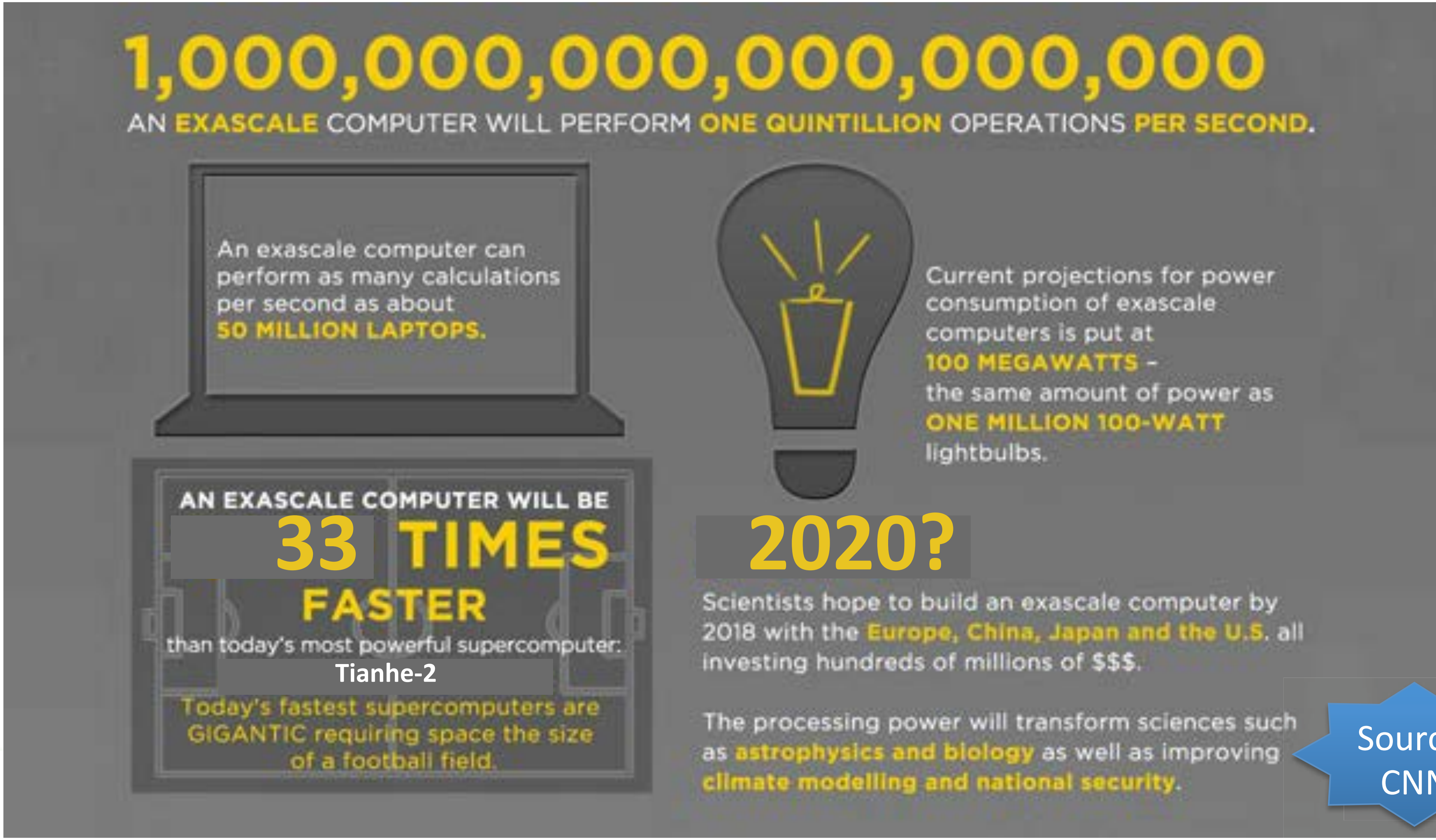
University of St Andrews

- **Probably *not* shared memory**
  - not all memory will cost the same to address
  - maybe hardware distributed shared memory
  - maybe hardware transactional memory

- ***Assuming afully shared memory will not work!***
  - *But most models make the programmer do all the work!*
    - *e.g. Partitioned Global Address Space (PGAS)*
  - *Side effects will not work!*



ExaScale Megacore Computers



Source: CNN

Is this megacool?

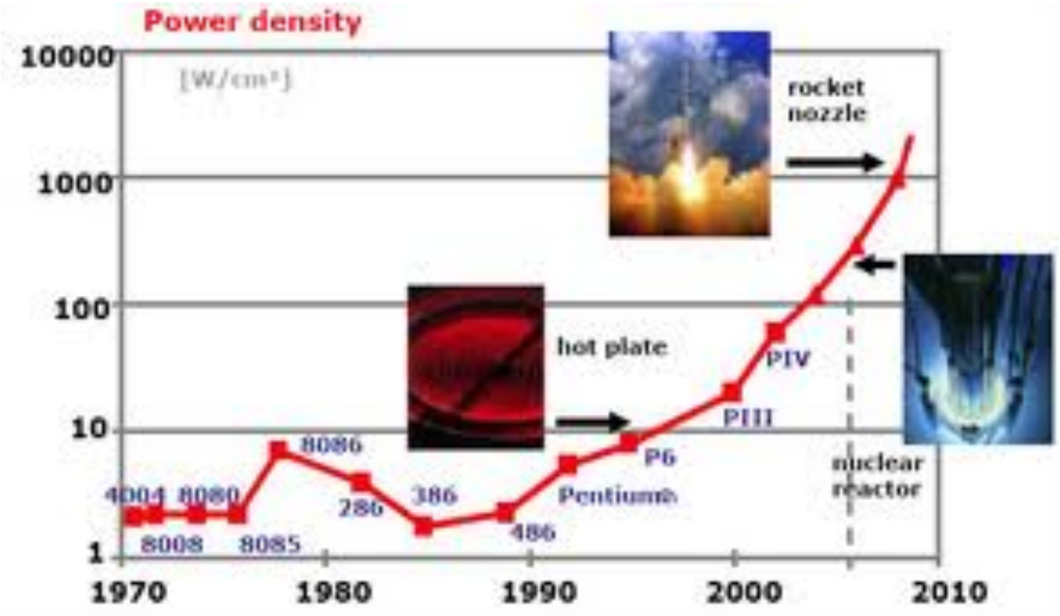
University of St Andrews

Image by Daniel Case

Or really megahot?

- Energy usage scales:
  - **linearly** with the *number of cores*
  - **cubically** with the *clock frequency*

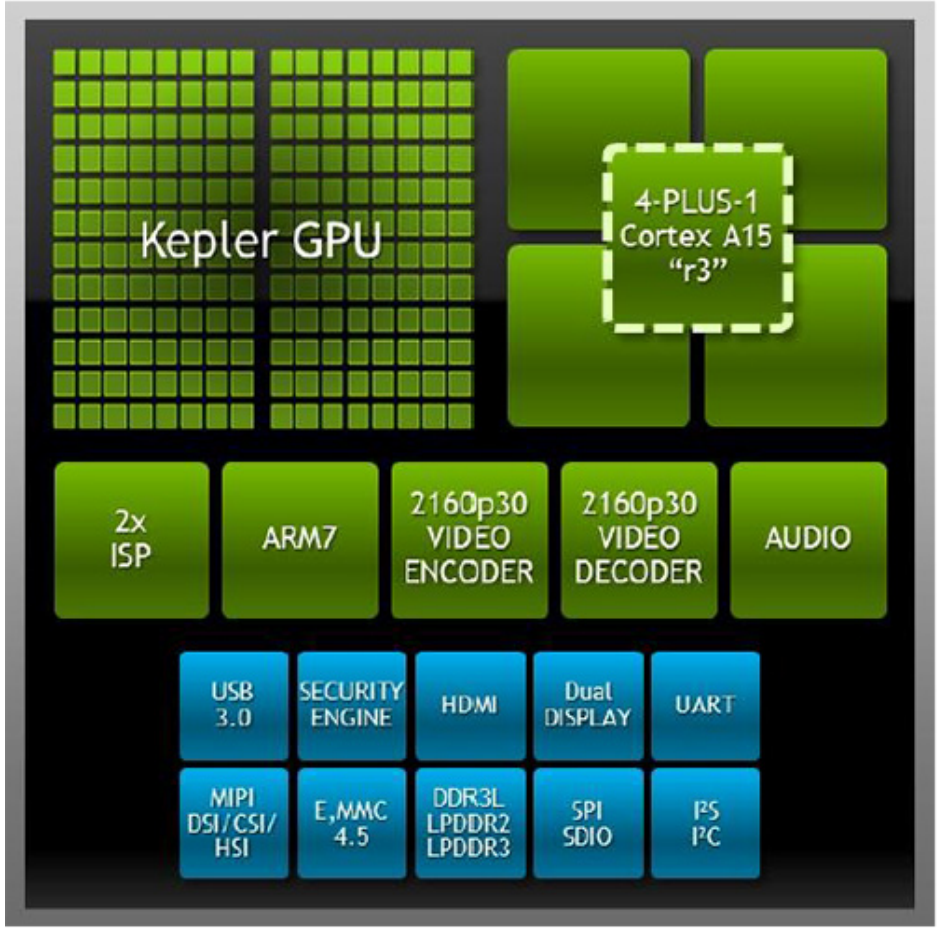
- Power density is critical
  - smaller process sizes (e.g. 22nm) need less energy
  - But a core 1/30 of the size will still consume 1/8 of the power!
  - We are reaching limits on heat dissipation!



Source: Patterson and Hennessey

- Efficient use of Energy is a **major** concern

"Embedded Supercomputing": Nvidia Tegra K1



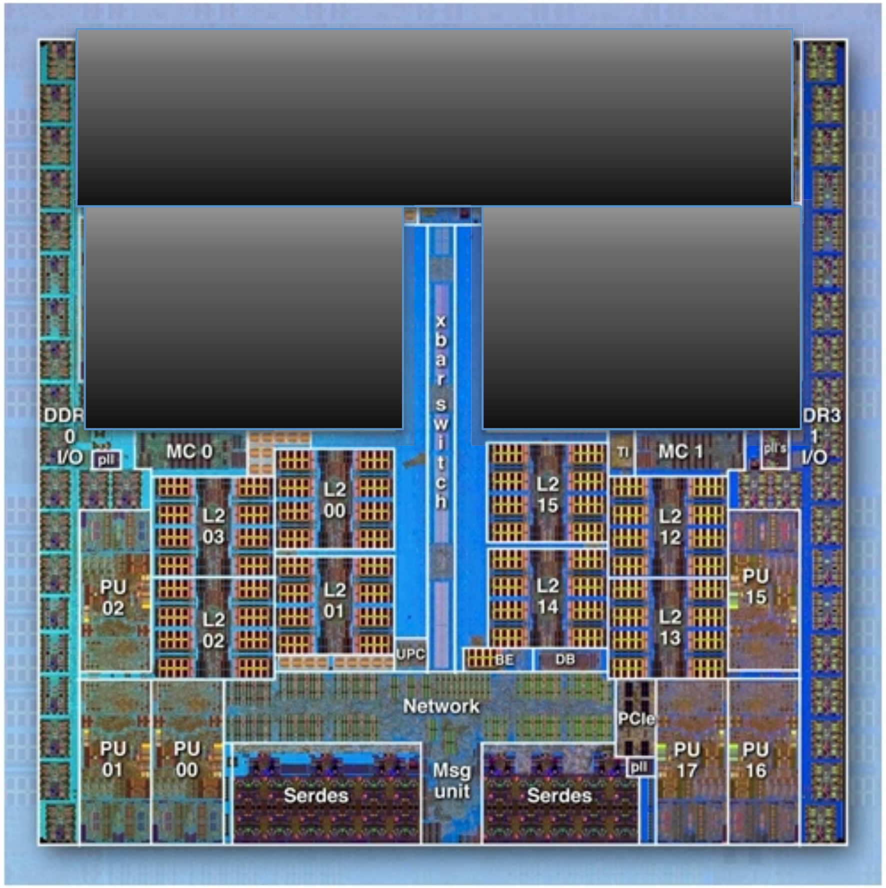
- 4 Fast ARM Cortex A15 Cores
- 1 Slower Low-Power A15 Core
  - Cores can be enabled individually, the rest are dark
- 192-core Kepler GPU
- 2 GB RAM
  - **Shared Between CPU and GPU**
- **1-5W Peak Power Usage** (60W Max)

Dark Silicon

University of St Andrews

- Not all the processor is powered
  - reduces power usage
  - (maybe not all CAN be powered!)

- Execution units are powered up when needed
  - e.g. to deal with video processing, security, etc



All future programming will be parallel

University of St Andrews

- No future system will be single-core
  - parallel programming will be essential
- It's not just about performance
  - it's also about energy usage
- If we don't solve the multicore challenge, then no other advances will matter!
  - user interfaces
  - cyber-physical systems
  - robotics
  - games
  - ...

The Manycore Challenge

University of St Andrews

"Ultimately, developers should start thinking about *tens, hundreds, and thousands* of cores *now* in their algorithmic development and deployment pipeline."

will not ``automagically'' run

actu

The **ONLY** important challenge in Computer Science

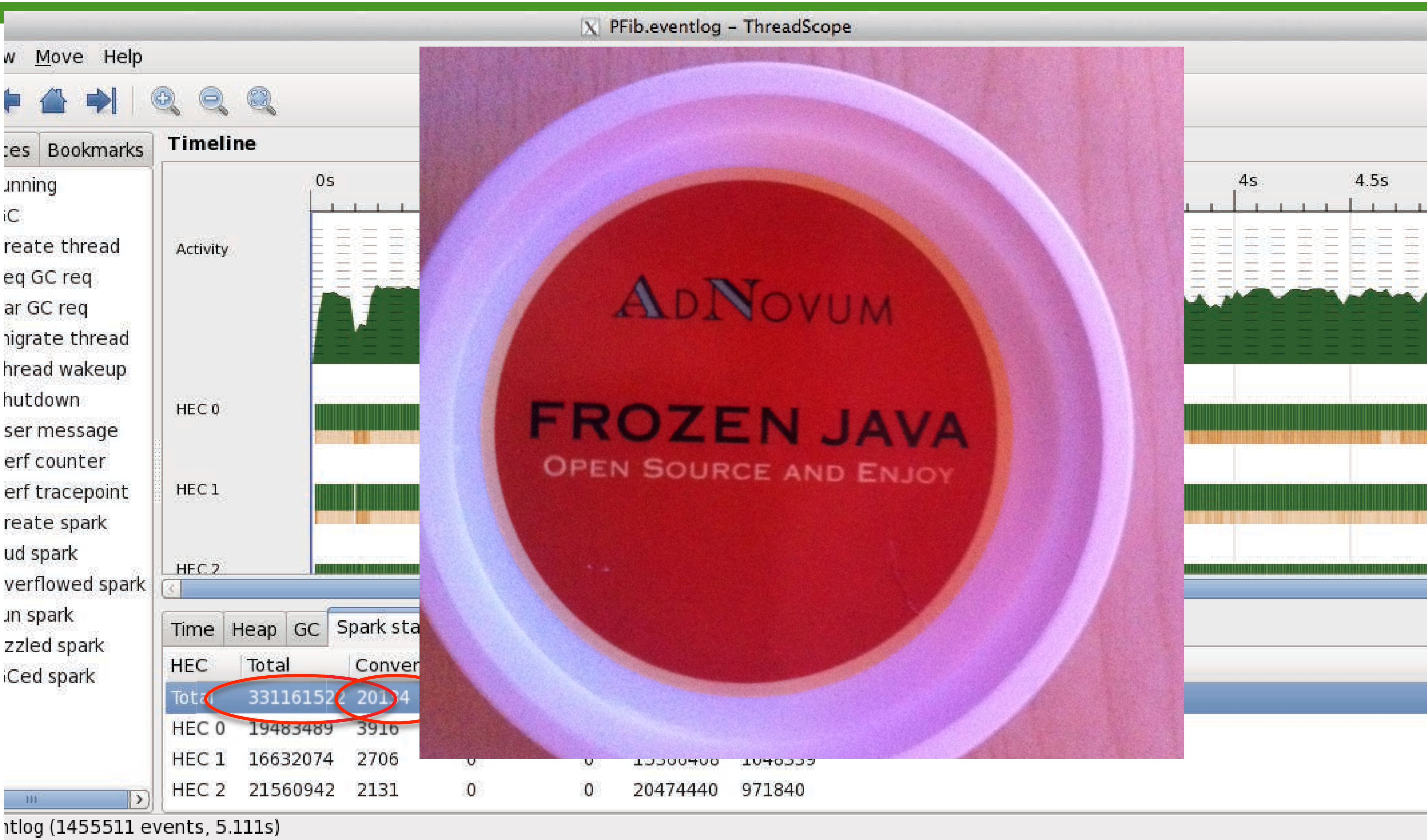
(Intel)

Also recognised as thematic priorities by EU and national funding bodies

Patrick Leonard, Vice President for Product Development
Rogue Wave Software

But Does it Scale?

What to **millions** of threads??

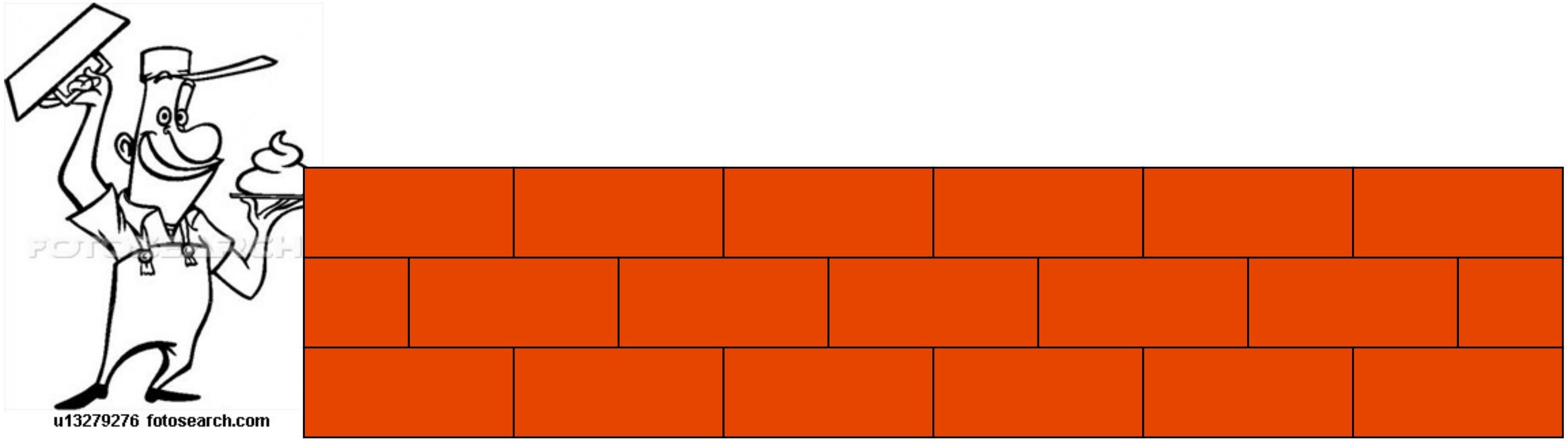


What are we trying to achieve?

University of St Andrews

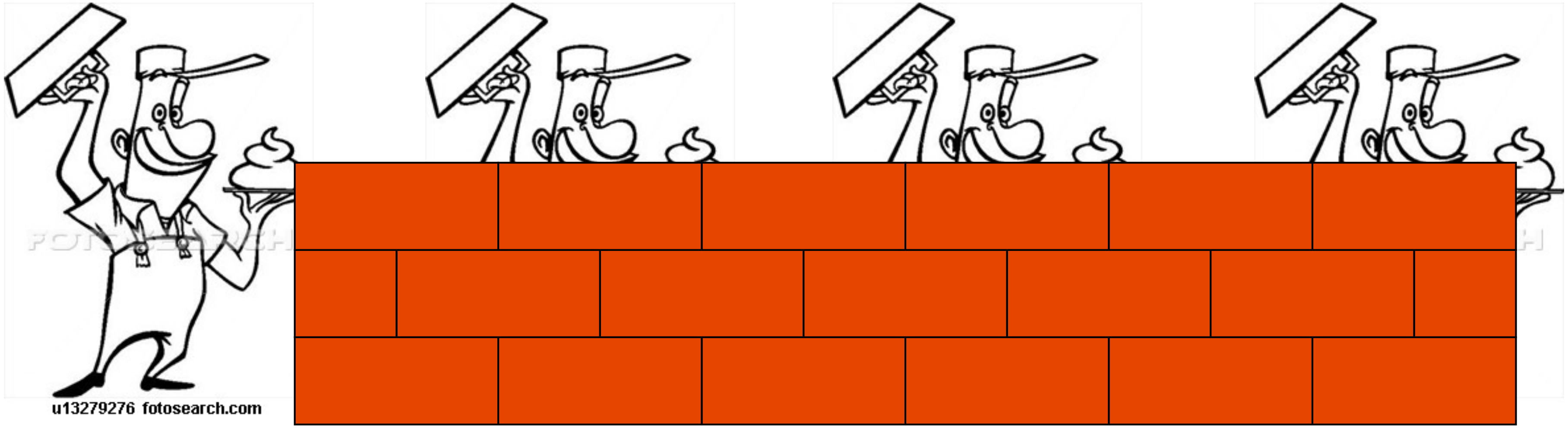
Parallelism and Concurrency

How to build a wall



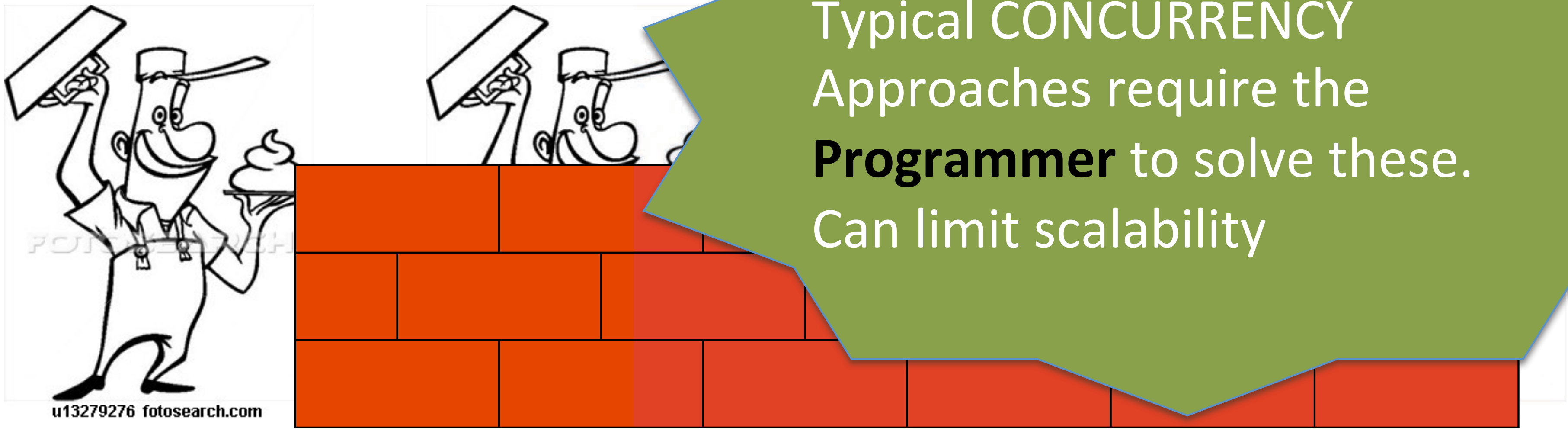
(with apologies to Ian Watson, Univ. Manchester)

How to build a wall *faster*



How NOT to build a wall

University of St Andrews



Typical CONCURRENCY Approaches require the **Programmer** to solve these.
Can limit scalability

Task identification is not the only problem…
Must also consider Coordination, communication, placement, scheduling, …

We need structure
We need abstraction

We don't need another brick in the wall

Thinking Parallel

- **Fundamentally, programmers must learn to "think parallel"**
  - this requires new *high-level* programming constructs
    - perhaps dealing with hundreds of *millions* of threads
- **You cannot program effectively while worrying about deadlocks etc.**
  - they must be eliminated from the design!
- **You cannot program effectively while fiddling with communication etc.**
  - this needs to be packaged/abstracted!

A Solution?

"The only thing that works for parallelism is functional programming"

Bob Harper, Carnegie Mellon University

Parallel Functional Programming

University of St Andrews

- Purity means no side-effects
  - Easy to find parallelism
  - Impossible for parallel processes to interfere with each other
  - **Can debug sequentially but run in parallel**
  - **Enormous** saving in effort
- Programmers concentrate on solving the problem
  - Not porting a sequential algorithm into a (ill-defined) parallel domain
- **No locks, deadlocks or race conditions!!**
- **Huge productivity gains!**

Parallelism is not Concurrency

University of St Andrews

- Concurrency is a programming abstraction
  - The ***illusion*** of independent threads of execution

- Parallelism is a hardware artefact
  - The ***reality*** of threads executing at the same time

- Concurrency is about breaking a program down into separate units of computation (conceptual)
- Parallelism is about making things happen at the same time (practical)

- A parallel program has ***thousands or millions*** of **tiny** threads
- A concurrent program has a few **huge** threads

The ParaPhrase (ParTE) Approach

University of St Andrews

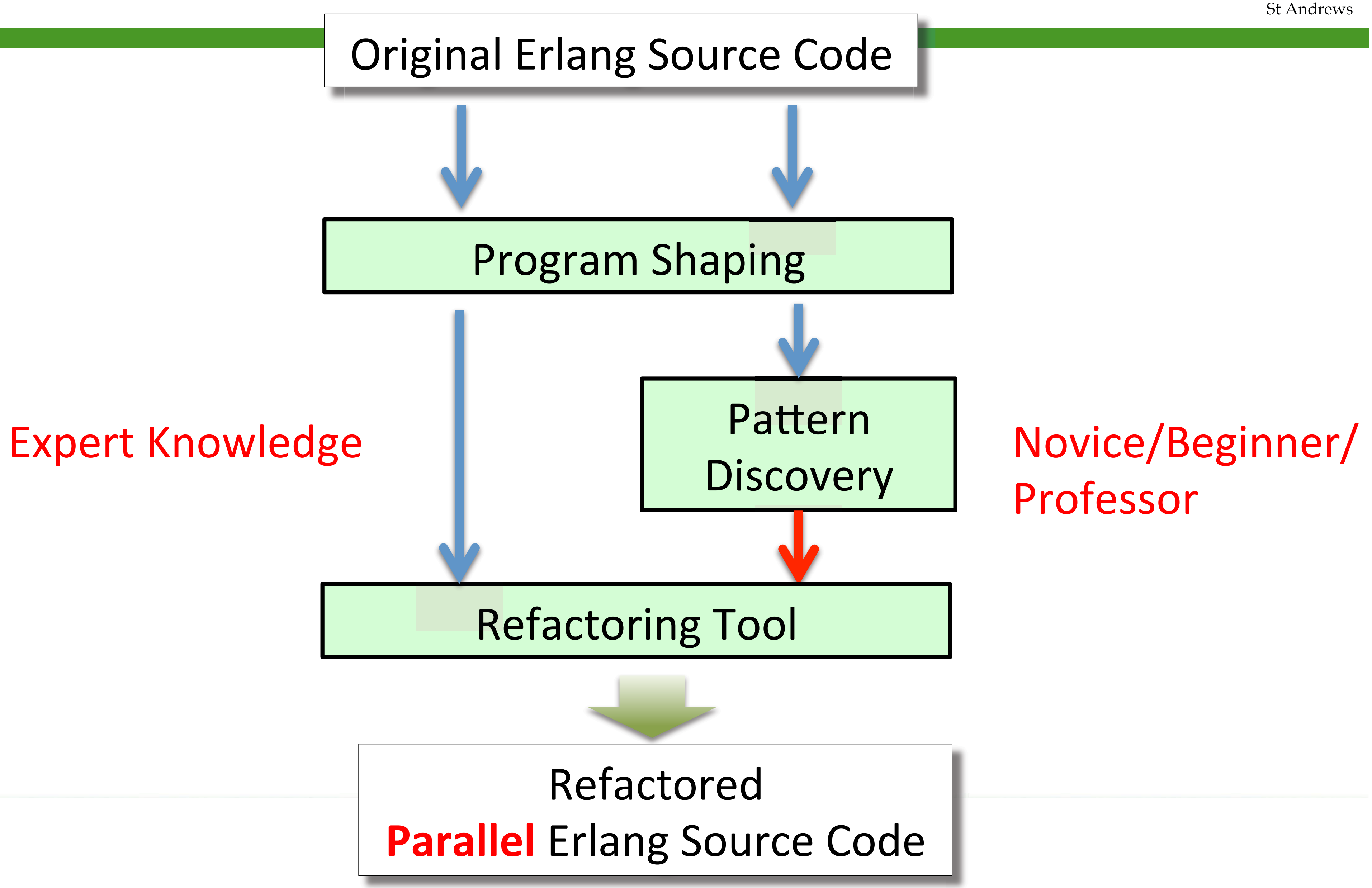
- Start bottom-up
  - identify (non-side-effecting) **COMPONENTS (BRICKS)**
  - *using semi-automated refactoring*

both legacy and new programs

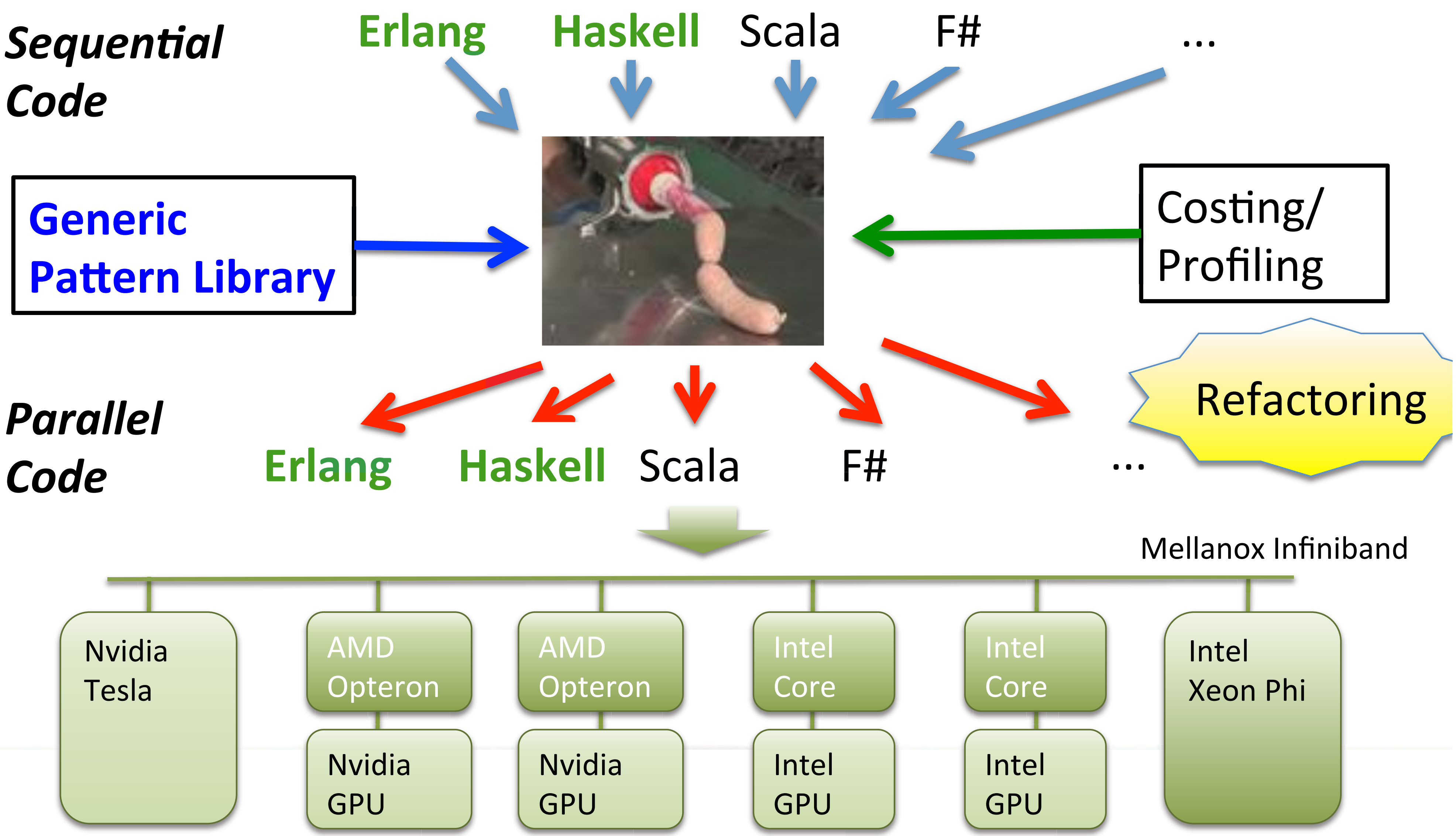
- Think about the **PATTERN** of parallelism
  - e.g. map(reduce), task farm, parallel search, parallel completion, ...
- **STRUCTURE** the components into a parallel program
  - ***turn the patterns into concrete (skeleton) code***
  - Take performance, **energy** etc. into account (multi-objective optimisation)
  - also using refactoring
- **RESTRUCTURE** if necessary! (also using refactoring)

PaRTE - General Technique

University of St Andrews

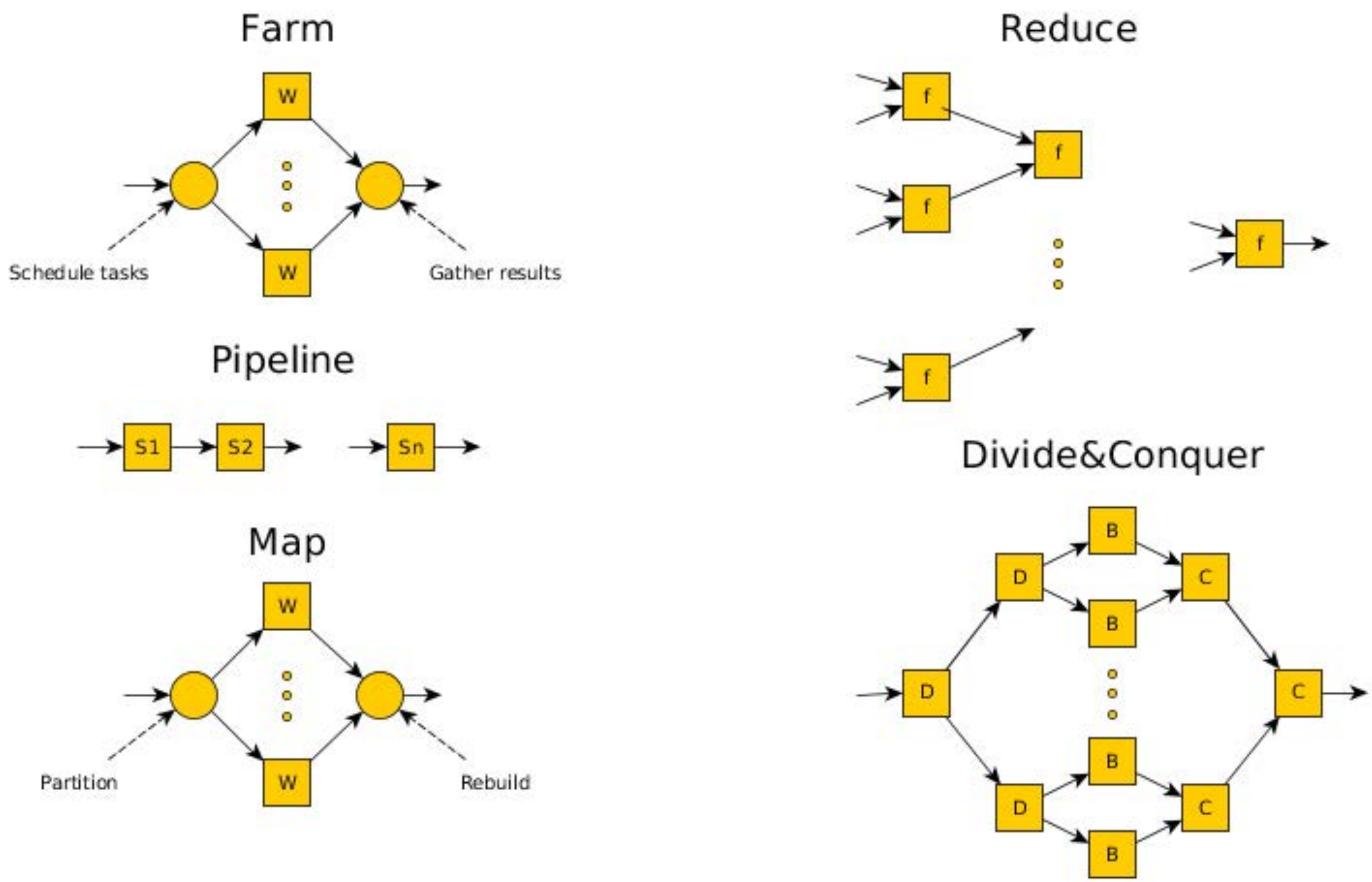


The ParaPhrase Approach



Some Common Patterns

- High-level abstract patterns of common parallel algorithms



Bricks are Functional

- We can construct a *closure* (aka a *future*) to capture some computation, e.g. in Parallel Haskell:

```
brick = ... {- an expression to evaluate in parallel -}

-- now run brick in parallel using the par construct
par brick  {- the main computation -}
```

In a strict language (Erlang, Scala, ...), you can simply turn it into a function...

```
brick [] = ...
par (fun brick [])  ...
```

Bricks are Functional

University of St Andrews

- We can construct a *closure* (aka a *future*) to capture some computation, e.g. in Parallel Haskell:

```
brick = ... {- an expression to evaluate in parallel -}

-- now run brick in parallel using the par construct
par brick  {- the main computation -}
```

> In a strict language (Erlang, Scala, ...), you can simply turn a brick into a function...
>
> *brick [] = ...*
> *par (fun brick) ...*

Parallel Patterns are Functional

- *Higher-order functions* can capture parallel patterns

```
-- warning: pidgin-Erlang follows
parmap F [X|Xs] = par (F X) (parmap F Xs)
parmap F [] = []

-- build the bricks
bricks Input = parmap makebrick Input
```

- These functions are often called *skeletons (Murray Cole, 1989)*

The *Skel* Library for Erlang

University of St Andrews

- Skeletons implement specific parallel patterns
  - Pluggable templates
- **Skel** is a new (AND ONLY!) Skeleton library in Erlang
  - map, farm, reduce, pipeline, feedback
  - instantiated using **skel:do**
- ***Fully Nestable***
- **A DSL for parallelism**

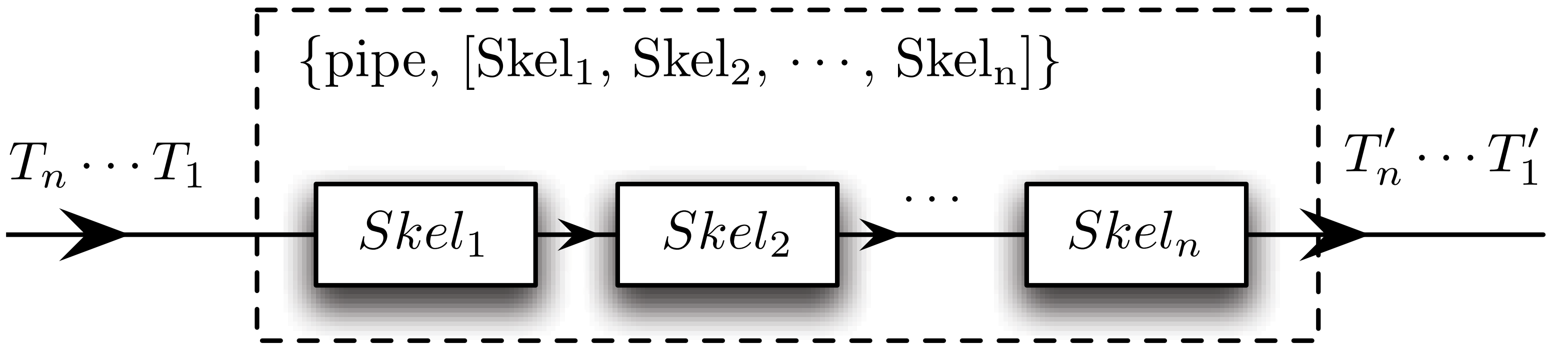
http://skel.weebly.com

https://github.com/ParaPhrase/skel

```
OutputItems = skel:do(Skeleton, InputItems).
```

Parallel Pipeline Skeleton

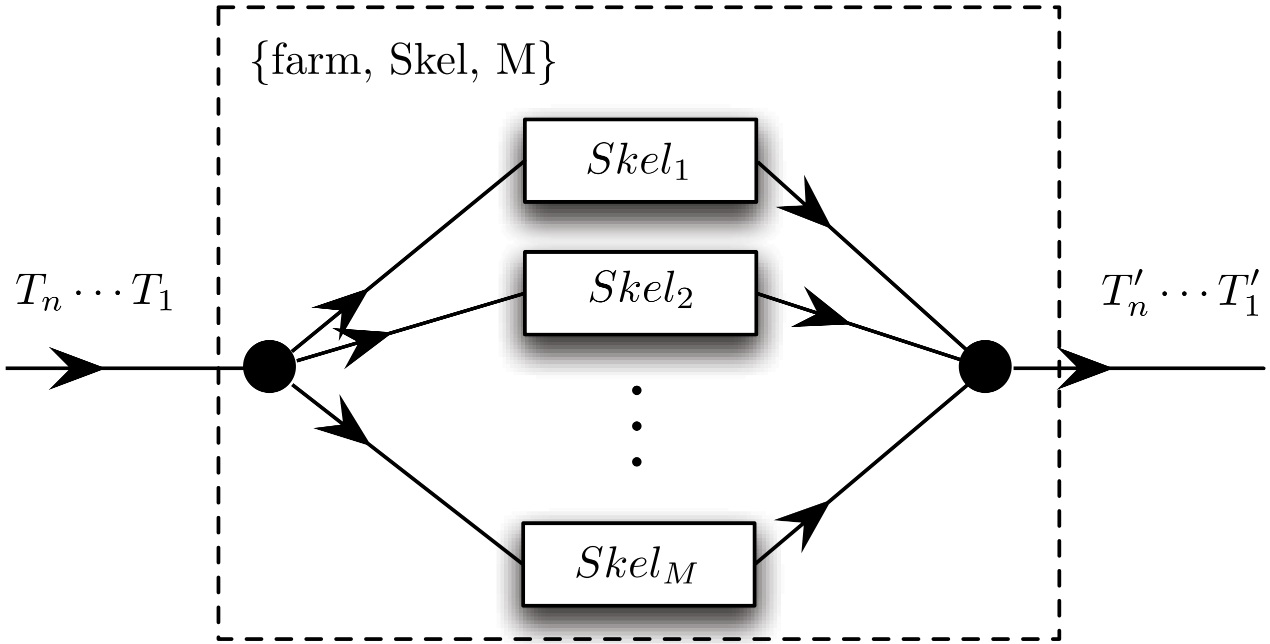
- Each stage of the pipeline can be executed in parallel
- The input and output are streams




```
skel:do([{pipe,[Skel1, Skel2,..,SkelN]}], Inputs).
```

Farm Skeleton

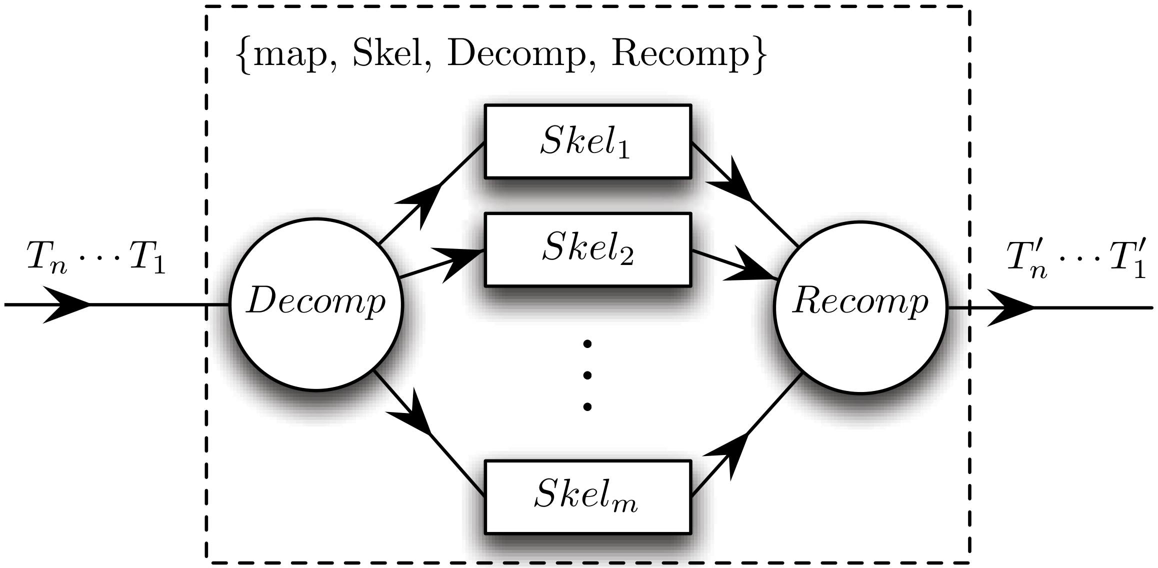
- Each worker is executed in parallel
- A bit like a 1-stage pipeline




```
skel:do([{farm, Skel, M}], Inputs).
```

Map Skeleton

University of St Andrews

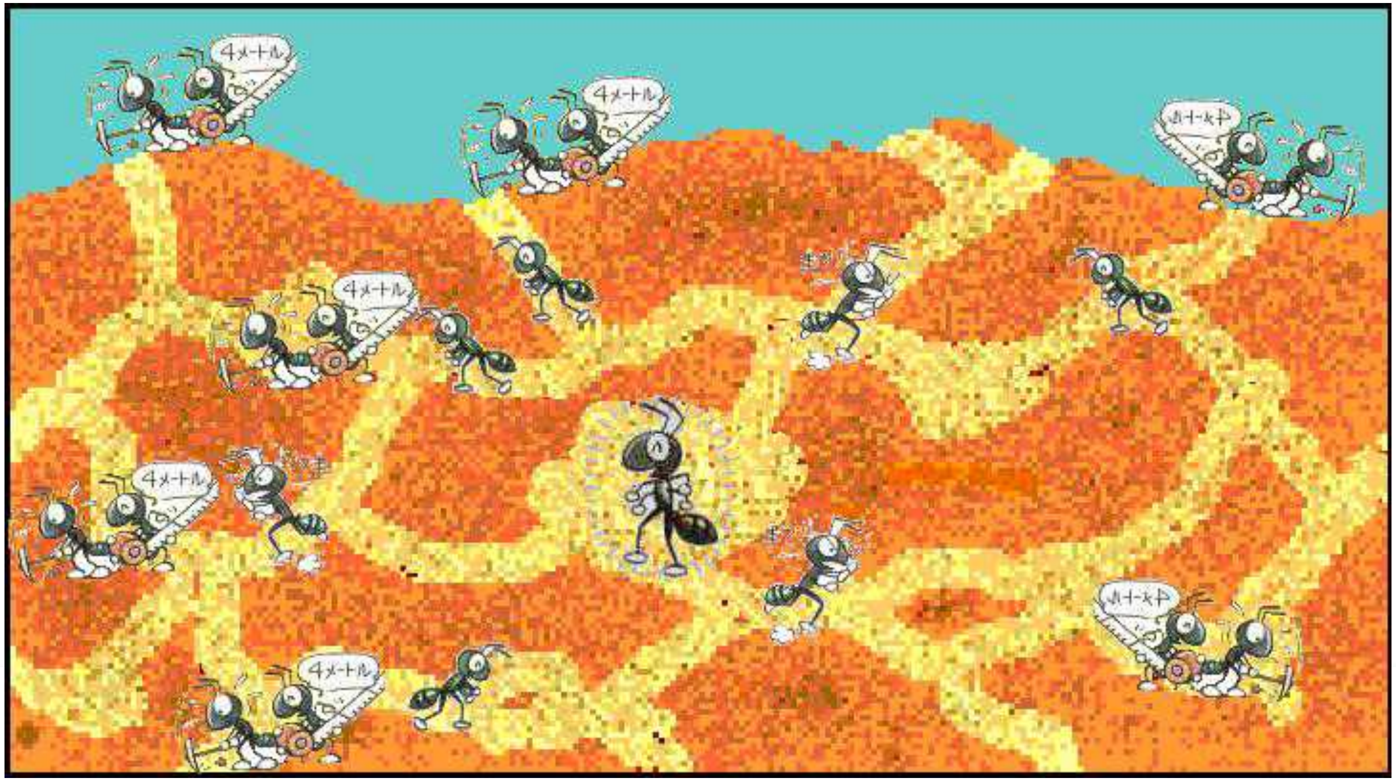



```
skel:do([{map, Skel, Decomp, Recomp}],
                                 Inputs).
```

Example: Ant Colony Optimisation

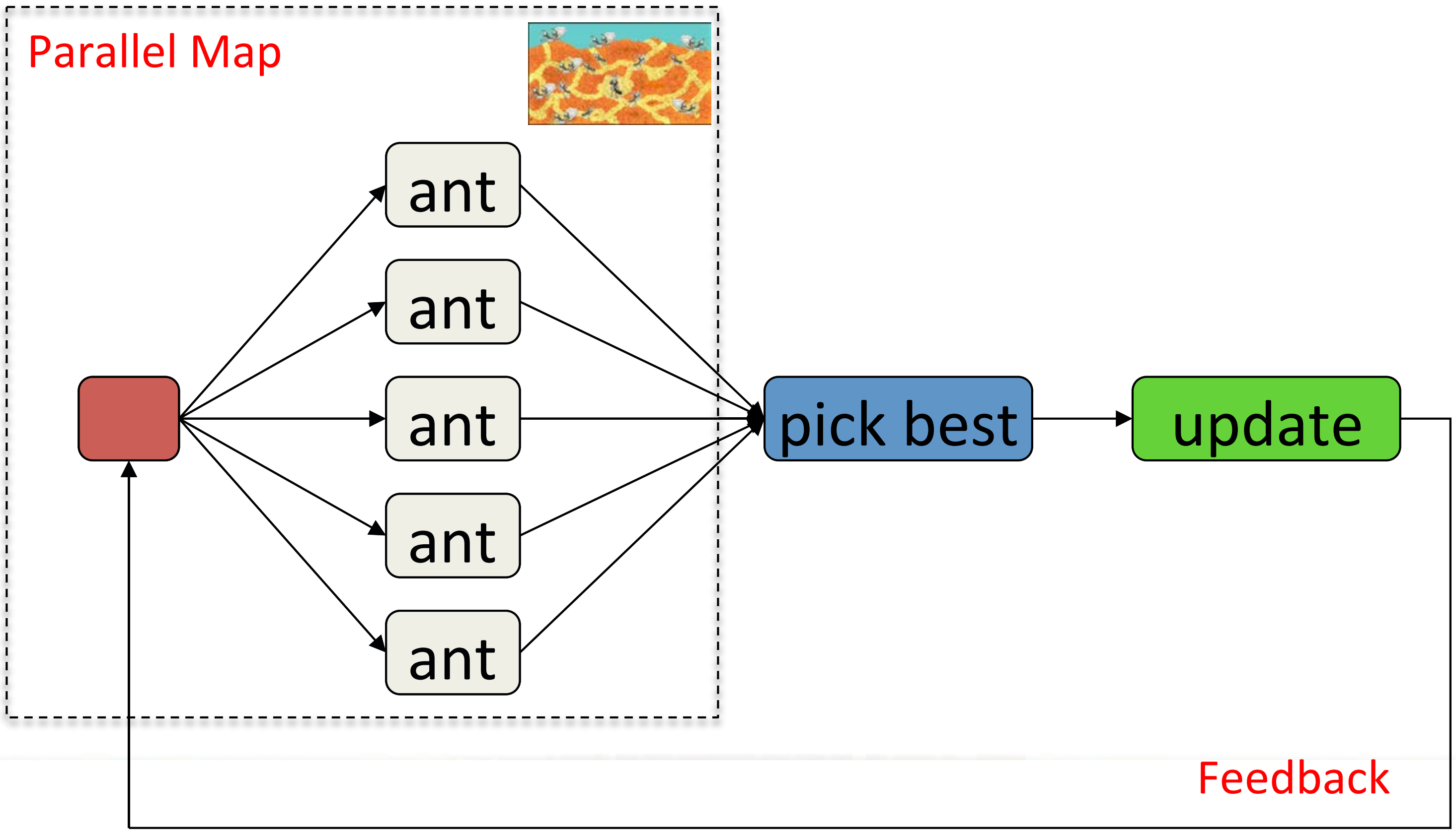
University of St Andrews

- Tries to find a good solution to a particular scheduling problem
  - find a schedule which *minimises* the time by which each job misses its deadline
- *N!* possible schedules for *N* jobs. Solving this is **NP-hard.**



Ant Colony – Parallelisation

- Ants can be parallelised with Parallel Map pattern
- Complete computation needs to be wrapped into a Feedback pattern

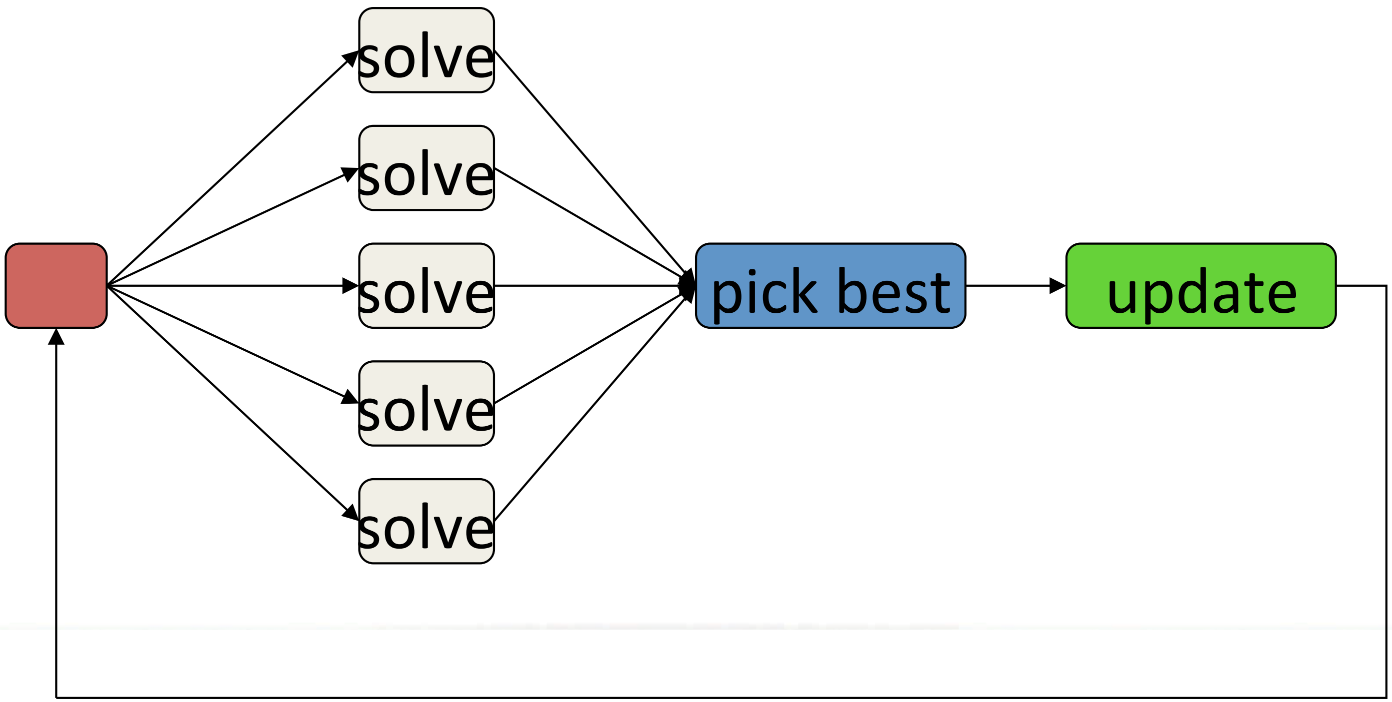


Ant Colony

University of St Andrews

- Pheromone trail in this case is a matrix where entry (i,j) is the probability that job j is i-th in the schedule

- Parallel structure of a program is

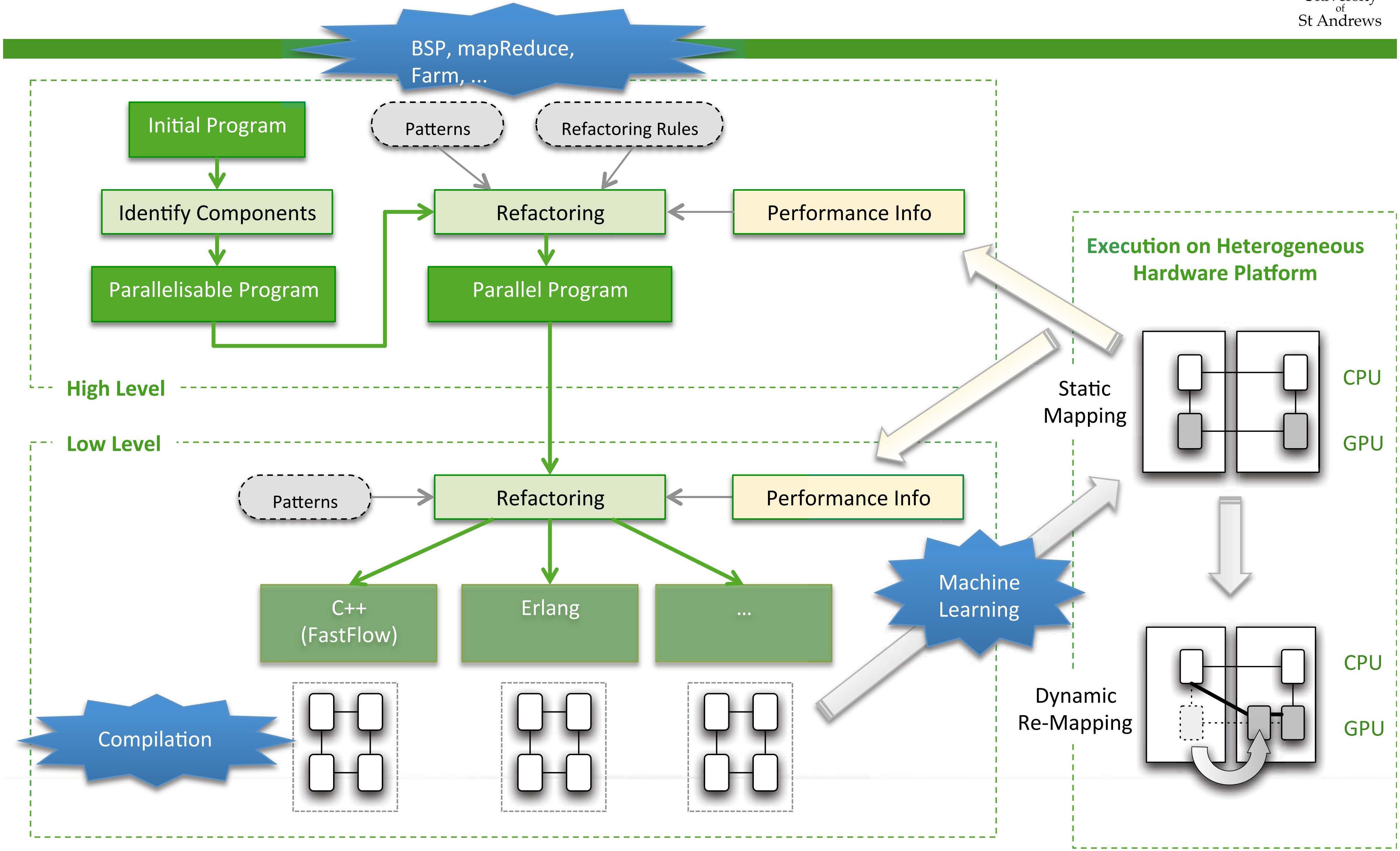


Ant Colony Code: Nested Skeletons

University of St Andrews

```
ant_colony(FName, Num_Ants, Num_Iters, NWs) ->
    {Num_Jobs, Process_Time, Weight, Deadline, Tau} = binary_init(FName),
    Task = {Num_Jobs, Process_Time, Weight, Deadline, Tau, Num_Iters},
    ChunkSize = calculate_chunk_size(Num_Ants div 64, NWs),
    InputList = create_input_list(ChunkSizes, 64, Task),
    Pipe = {pipe, [{map, [{func, fun(X) ->
                         lists:map(fun find_solution/1, X) end}],
                   fun(X) -> X end,
                   fun(X) -> X end},
                  {func, fun(X) -> pickbest_update (NWs, ChunkSize, X) end}]},
    Feedback = {feedback, [Pipe], fun(X) -> ant_feedback(X) end},
    skel:do([Feedback], [InputList]).
```

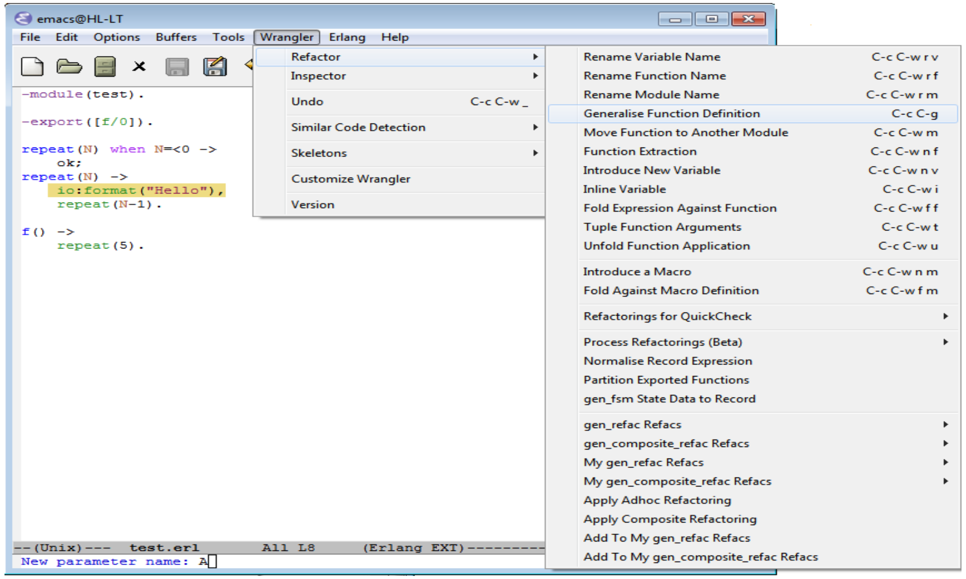
ParaPhrase Project Vision

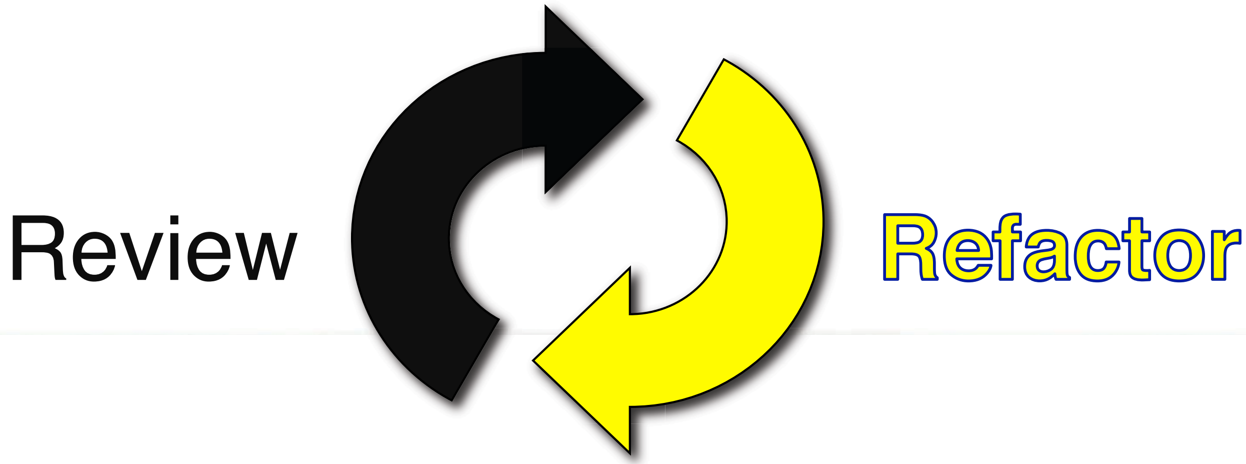


Refactoring

University of St Andrews

- Refactoring **changes** the **structure of** the **source code**
  - using well-defined rules
  - *semi-automatically under programmer guidance*





Refactoring: Farm Introduction

University of St Andrews

S	$\equiv$	$Farm(S)$	*farm intro/elim*

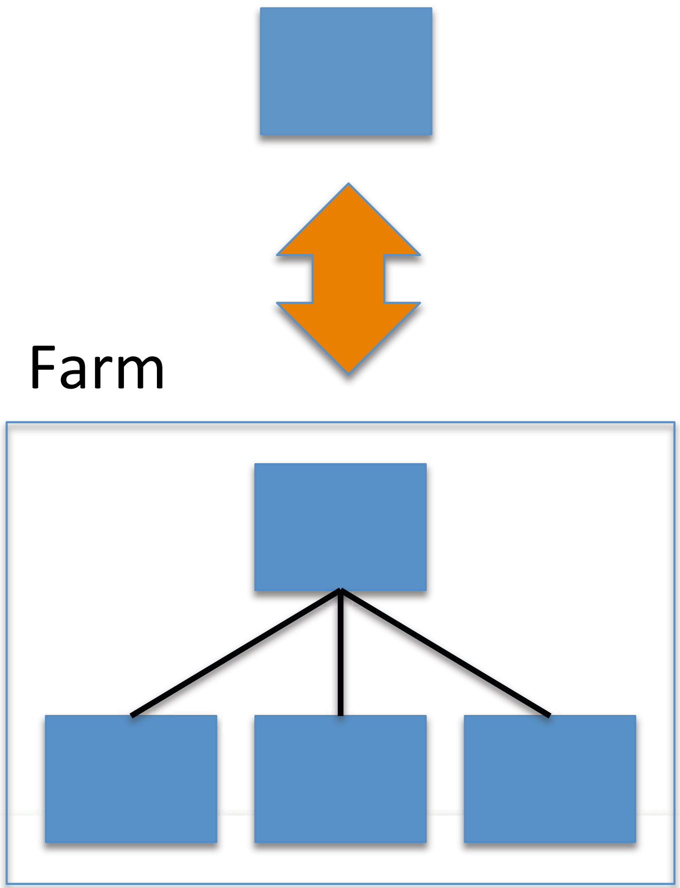
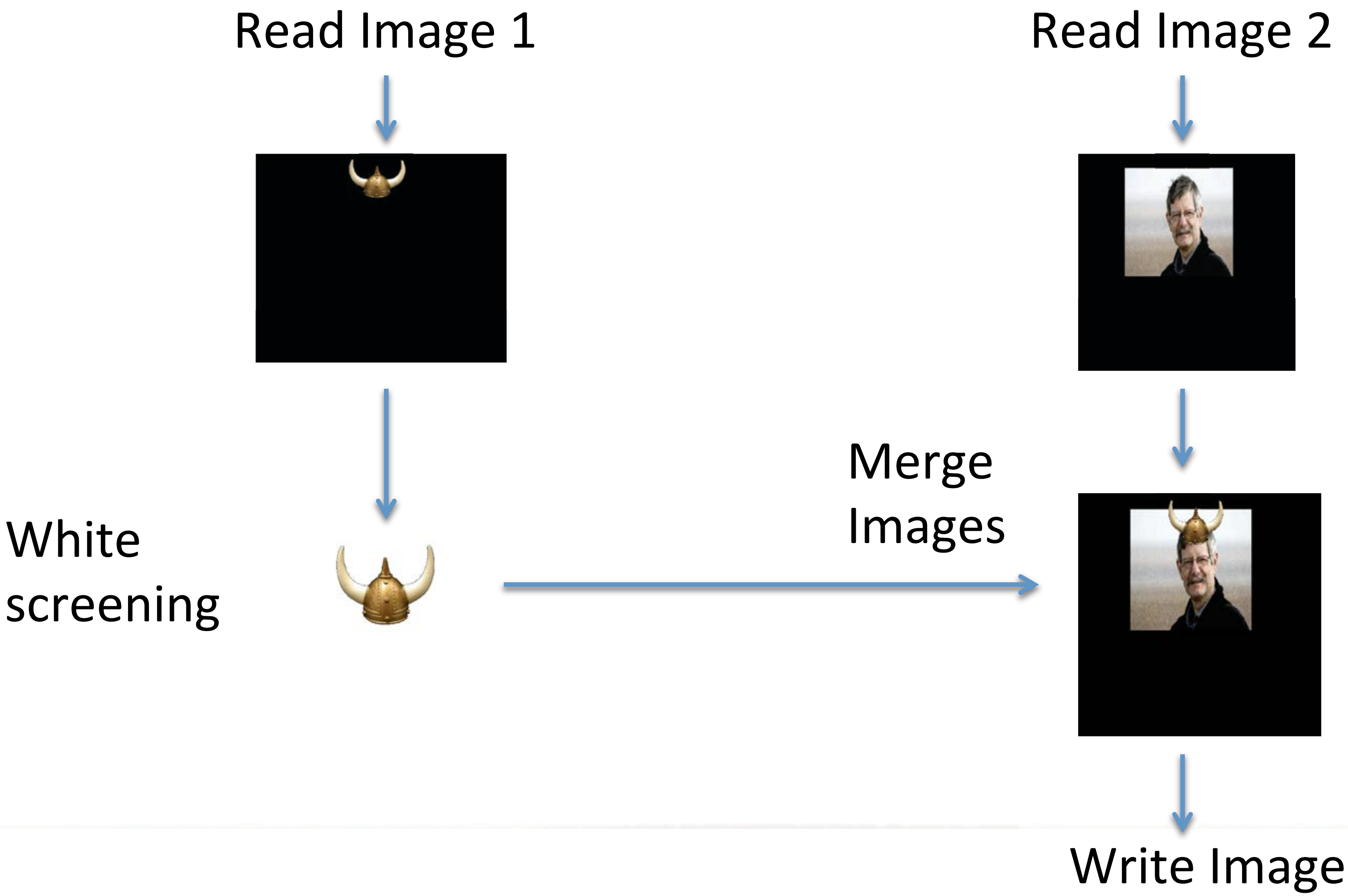


Image Processing Example



Basic Erlang Structure

University of St Andrews

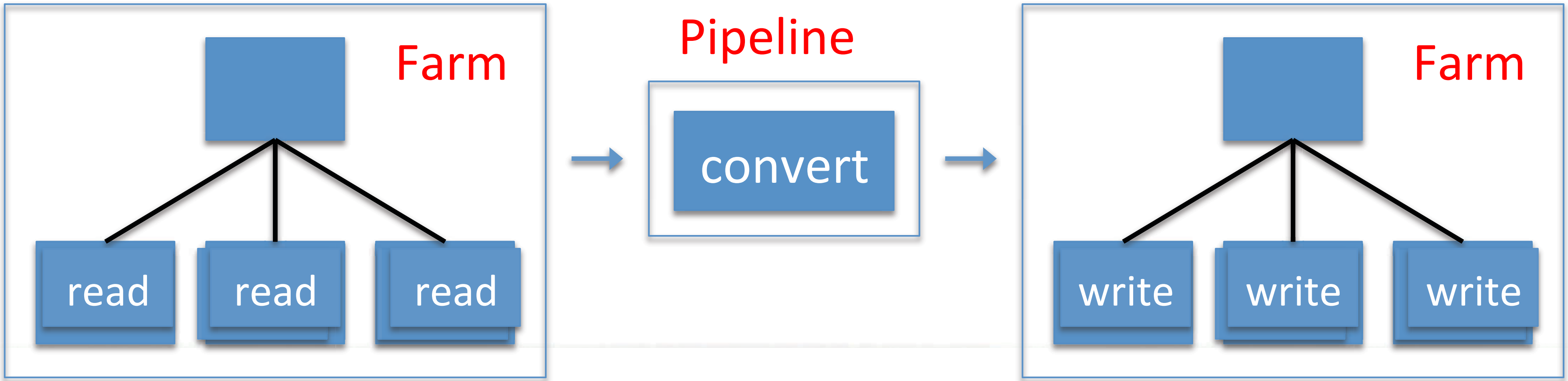
```
[ writeImage(convertMerge(readImage(X)))
                                      || X <- Images() ]

readImage({In1, in2, out) ->
      …
      { Image1, Image2, out}.

convertImage({Image1, Image2, out}) ->
      Image1P = whiteScreen(Image1),
      Image2P = mergeImages(Image1, Image2),
      {Image2P, out}.

writeImage({Image, Out}) -> …
```

Program Structure

Sequential

```
for each image, i.
        write (convert (read i))
```

Parallel



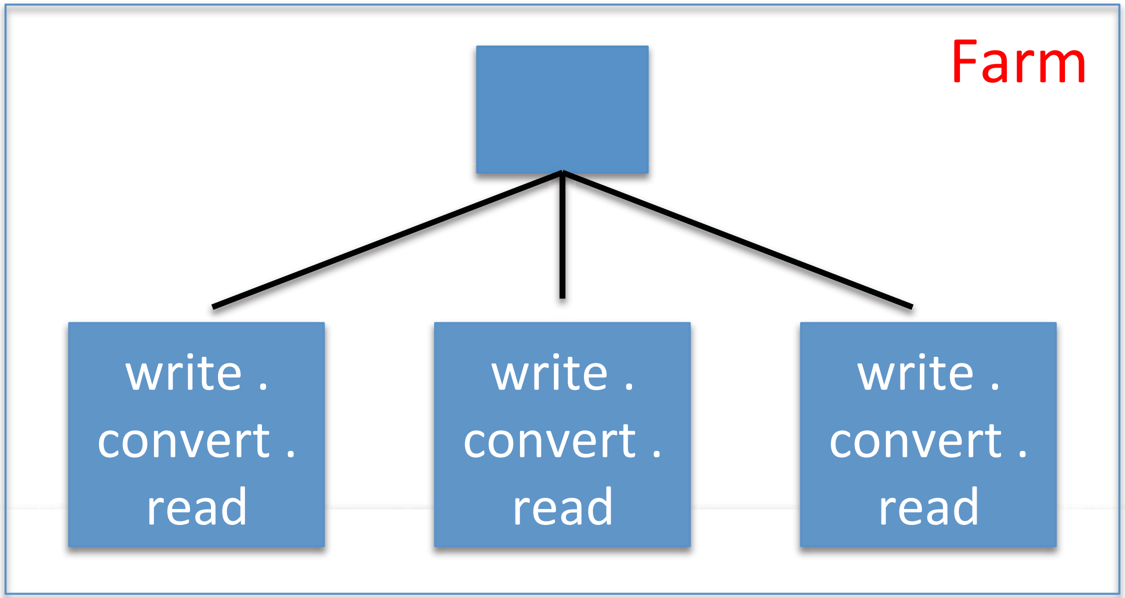
Alternative Program Structure

University of St Andrews

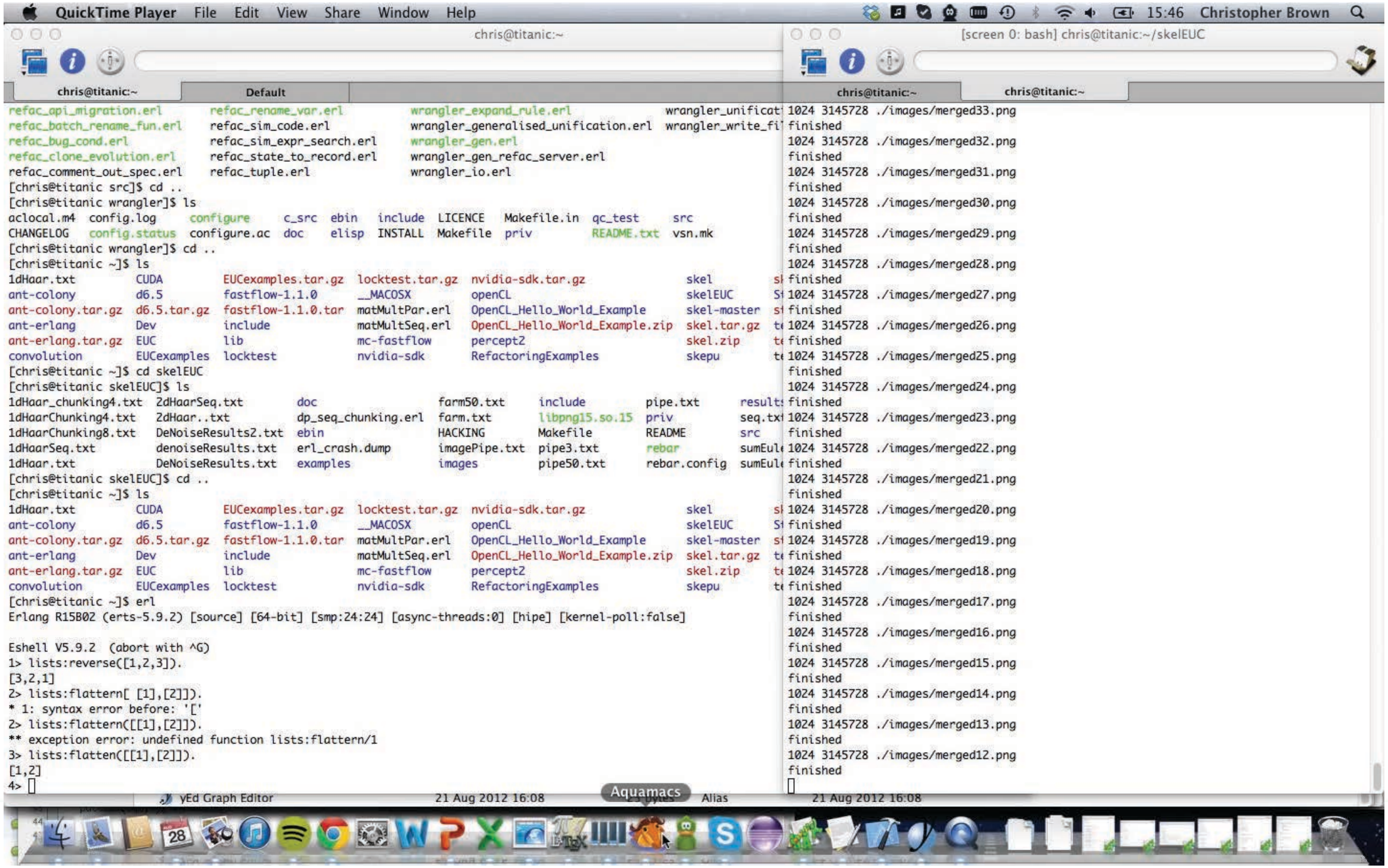
Sequential

```
for each image, i.
        write (convert (read i))
```

Parallel

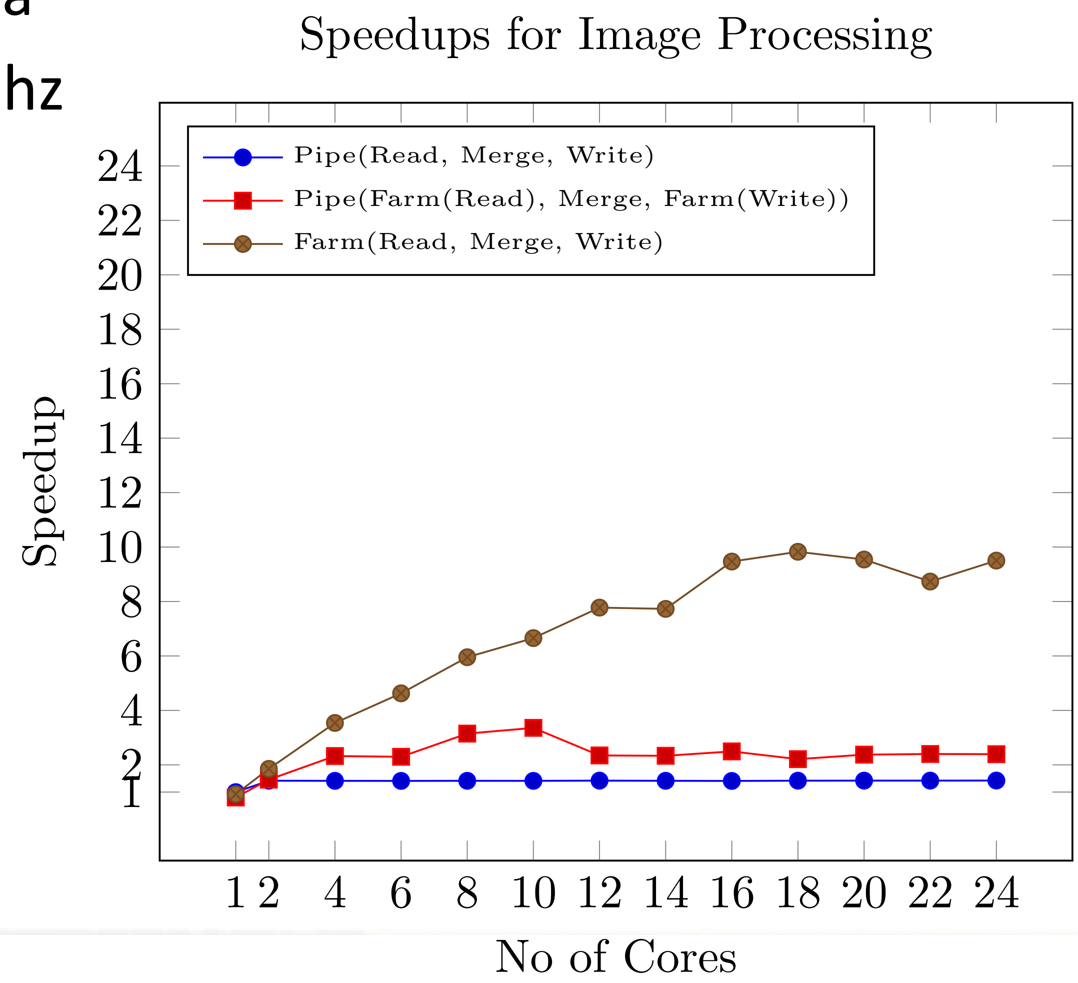


Refactoring

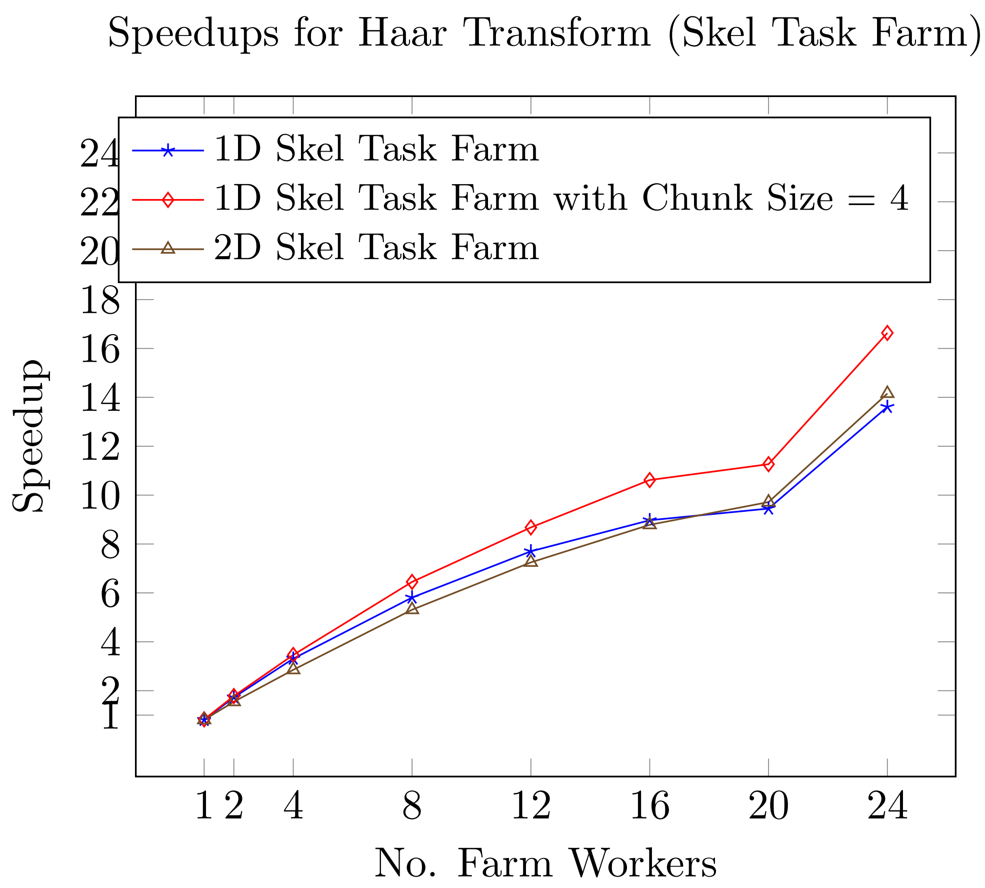


Speedup Results

- 24 core machine at Uni. Pisa
- AMD Opteron 6176. 800 Mhz
- 32GB RAM

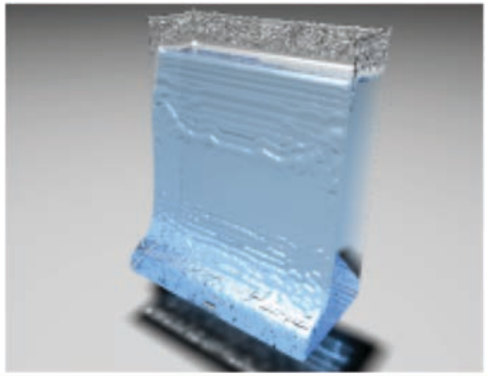


Speedup Results (Image Processing)



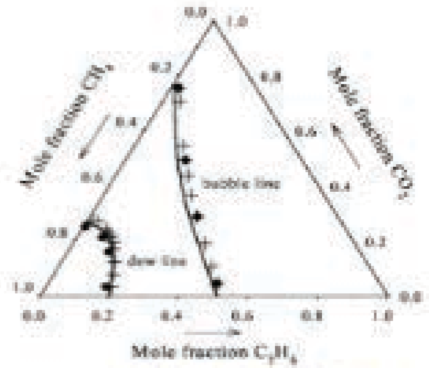
Large-Scale Demonstrator Applications

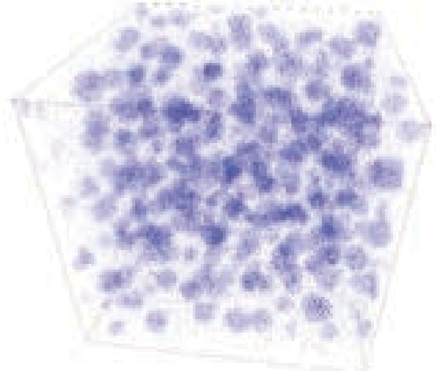
- ParaPhrase tools are being used by commercial/end-user partners
  - SCCH (SME, Austria)
  - Erlang Solutions Ltd (SME, UK)
  - Mellanox (Israel)
  - ELTESoft, Hungary (SME)
  - AGH (University, Poland)
  - HLRS (High Performance Computing Centre, Germany)

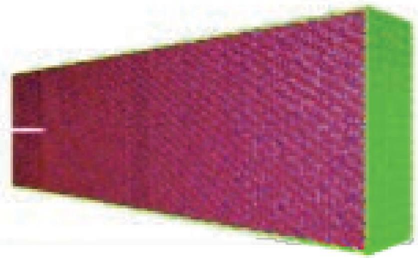


Examples: Computational Molecular Dynamics

- Simulates interactions between molecules
- Thermodynamic properties of fluids and gases
- Cultivated for basic research into HPC
- Features multiple MD data structures, algorithms and parallelization strategies
  - Allows quantitative comparisons
- Two widely used data structures with
- corresponding algorithms
  - BasicN2
  - MoleculeBlocks

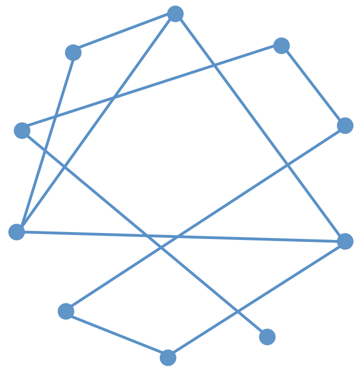






Examples: Machine Learning Methods

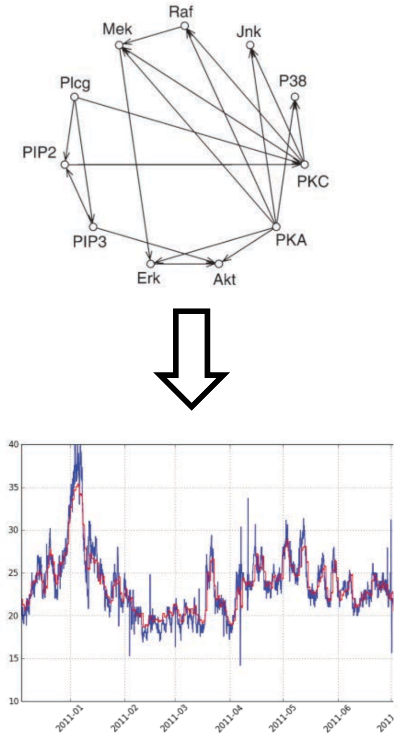
- Graphical Lasso
  - Determine direct linear influences
  - Iterative matrix inversion algorithm:
    - for each *independent components* of the matrix
    - by iteratively *solving a matrix inversion* problem:
    - for each feature
      - iteratively *solve a lasso regression*



- Ant-Colony Optmisation
  - tries to find a good solution to a particular scheduling problem
    - each job has a specified *duration* and *weight*.
    - find a sequential schedule (ie, a permutation of the jobs) which *minimises* the time by which each job misses its deadline
  - *N!* possible schedules for *N* jobs. Solving this is **NP-hard.**

Waste Water Prediction

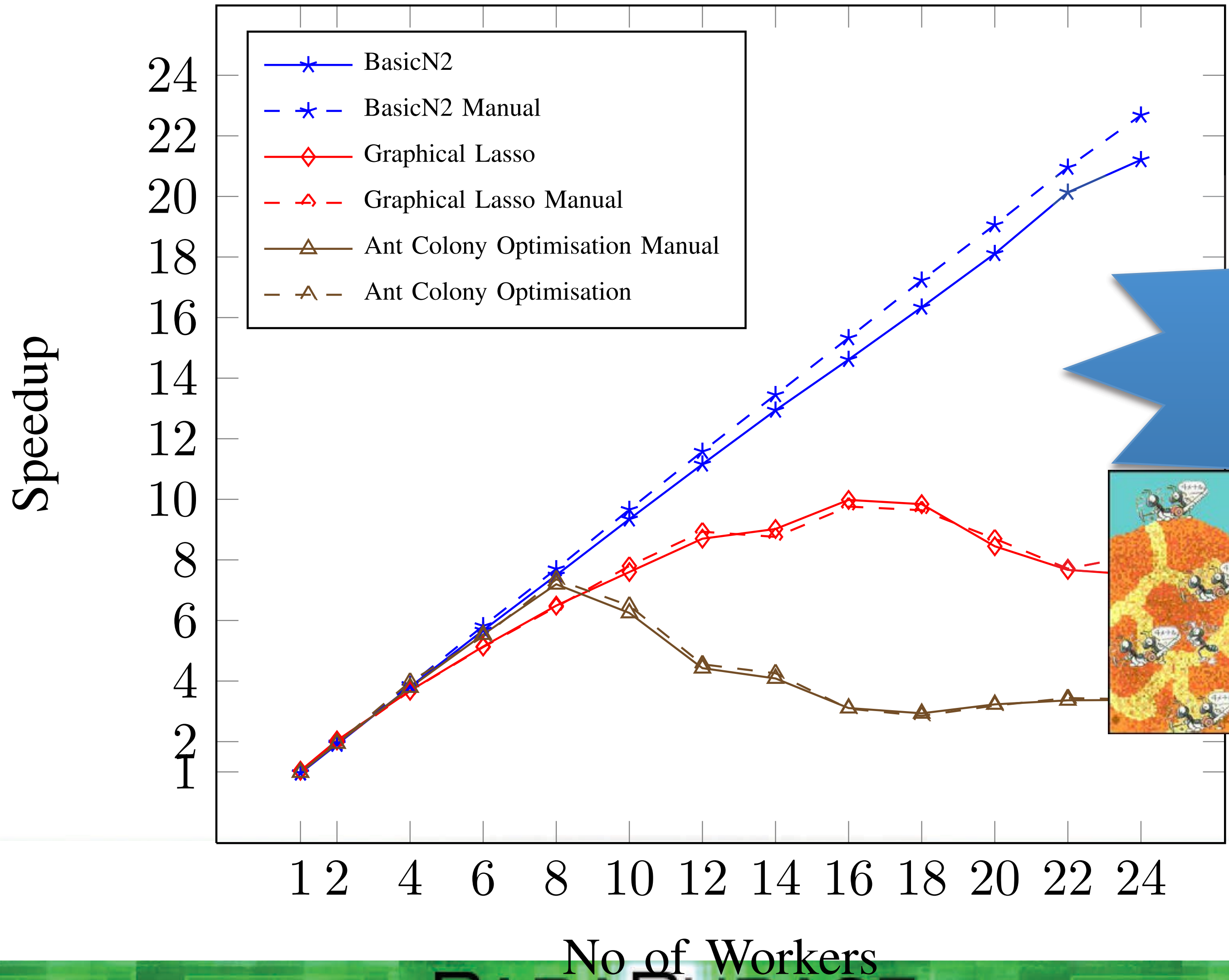
- Large industrial plant, residential and business neighborhood
- Predict total organic carbon content
- Find dependency structure
  - Robust prediction model
  - Using techniques such as graphical lasso, granger causality
- Input: throughput, chemical analyses, control parameters
  - ≈6000 features each hour, since 2.5 years



Speedup Results (demonstrators)

University of St Andrews

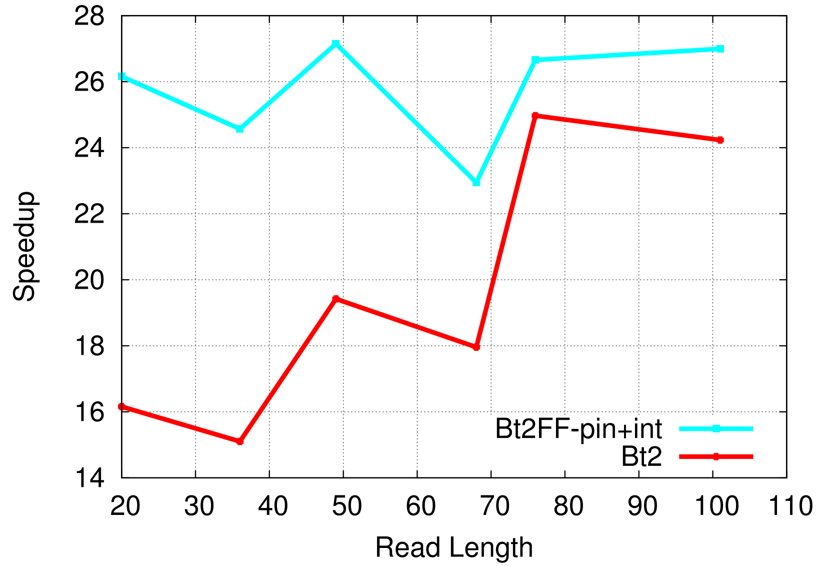
Speedups for Ant Colony, BasicN2 and Graphical Lasso

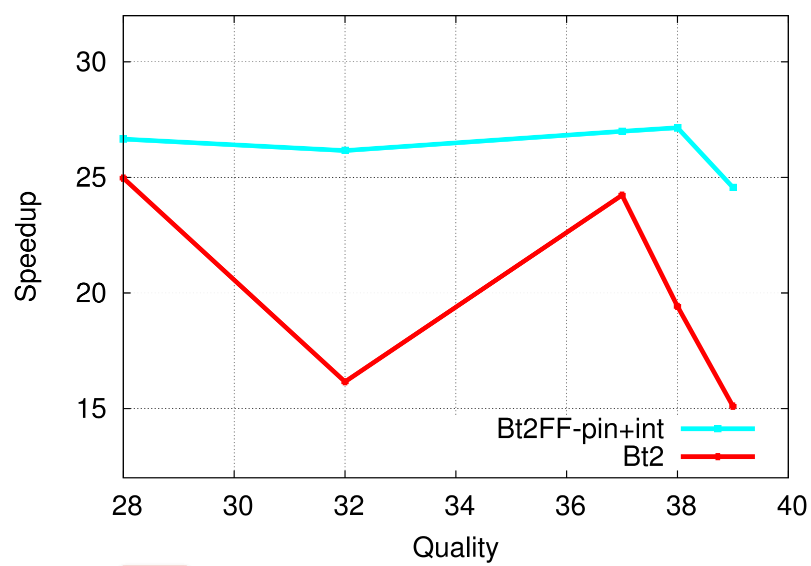


Speedup close to *or better than* manual

Bowtie2: most widely used DNA alignment tool

University of St Andrews





Original

Paraphrase

Metric	Bt2FF-pin+int	Bt2 interleaved
CPUs utilised	30.408	28.655
Context-switches	34816	199592
CPU-migrations	53	901
IPC	1.01	0.75
Stalled cycles per insn	0.58	0.93
Stalled-cycles-frontend	58.59%	69.67%
Stalled-cycles-backend	38.53%	53.19%
Branches-misses	5.08%	5.20%
L1-dcache-misses (of all L1-dcache hits)	4.07%	3.92%
LLC-load-misses (of all LL-cache hits)	41.62%	46.14%
Execution time (s)	35	55

C. Misale. Accelerating Bowtie2 with a lock-less concurrency approach and memory affinity. IEEE PDP 2014. To appear.

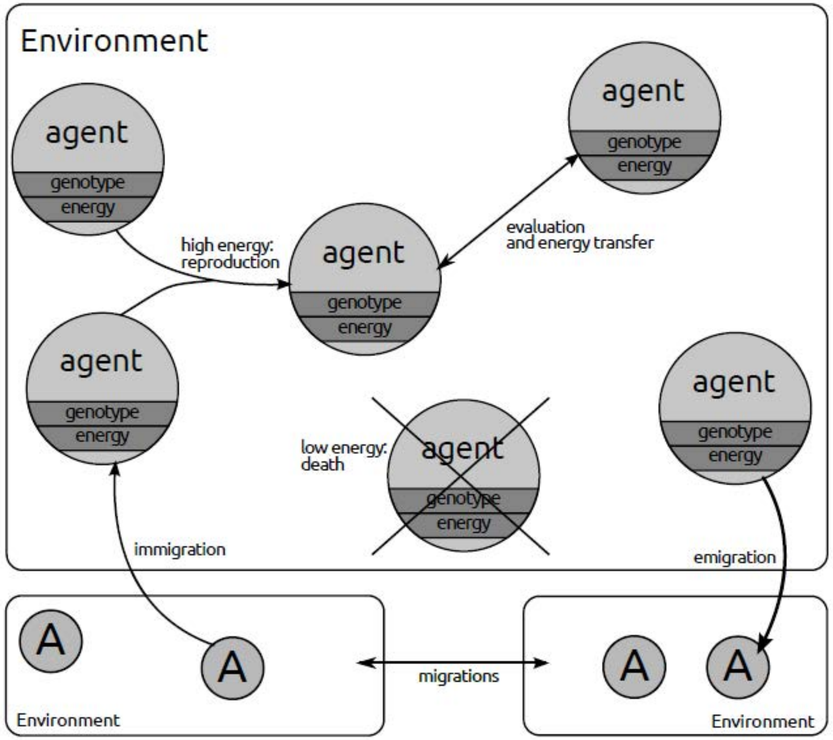
Comparison of Development Times

University of St Andrews

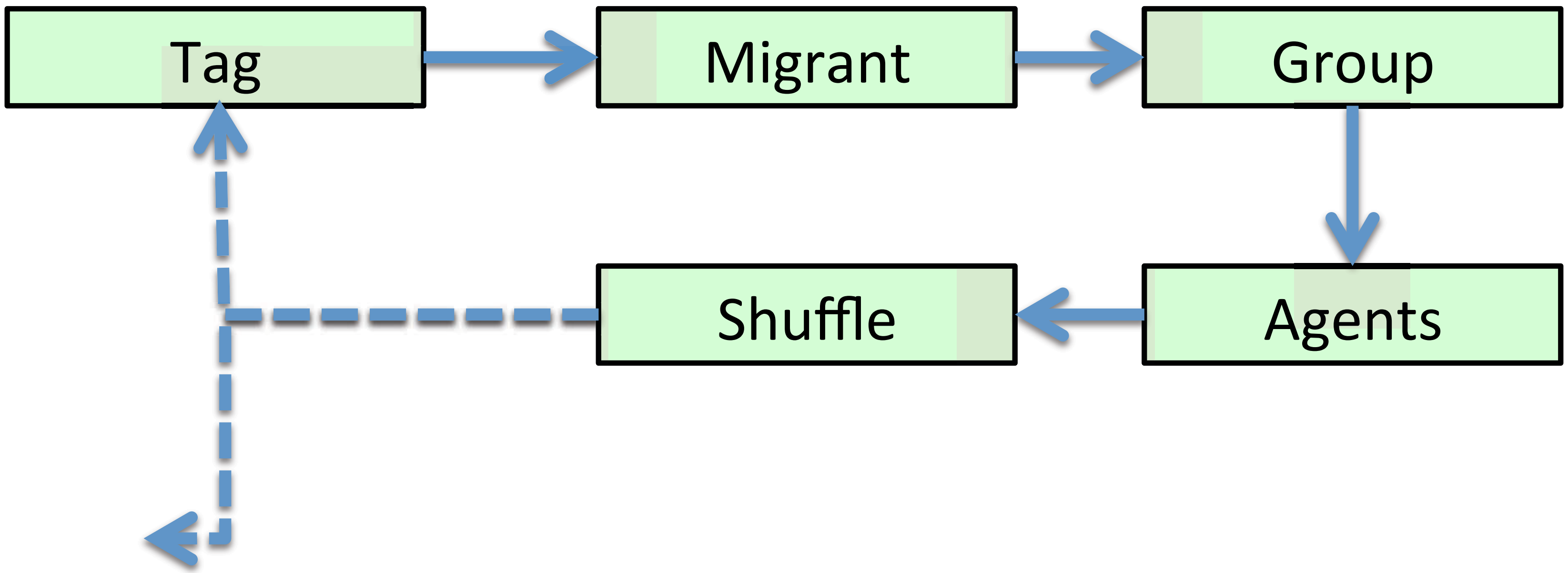
	Man.Time	Refac. Time
Convolution	3 days	3 hours
Ant Colony	1 day	1 hour
BasicN2	5 days	5 hours
Graphical Lasso	15 hours	2 hours

Evolutionary Multi-Agent Systems (EMAS)

- Meta heuristic approach for optimization
  - universal optimization algorithm (formally proven)
- Agents
  - **located on evolutionary islands**
  - perform actions (death, reproduction, migration, fight)

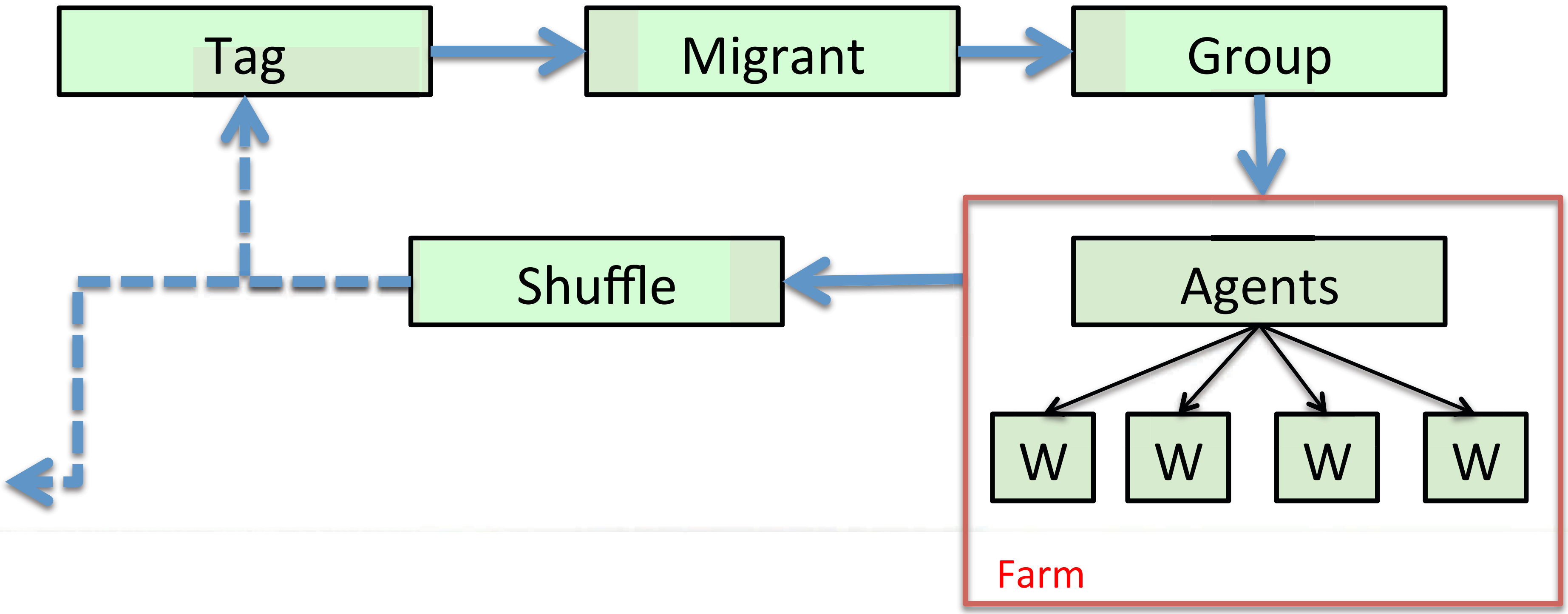


MAS – Basic Structure



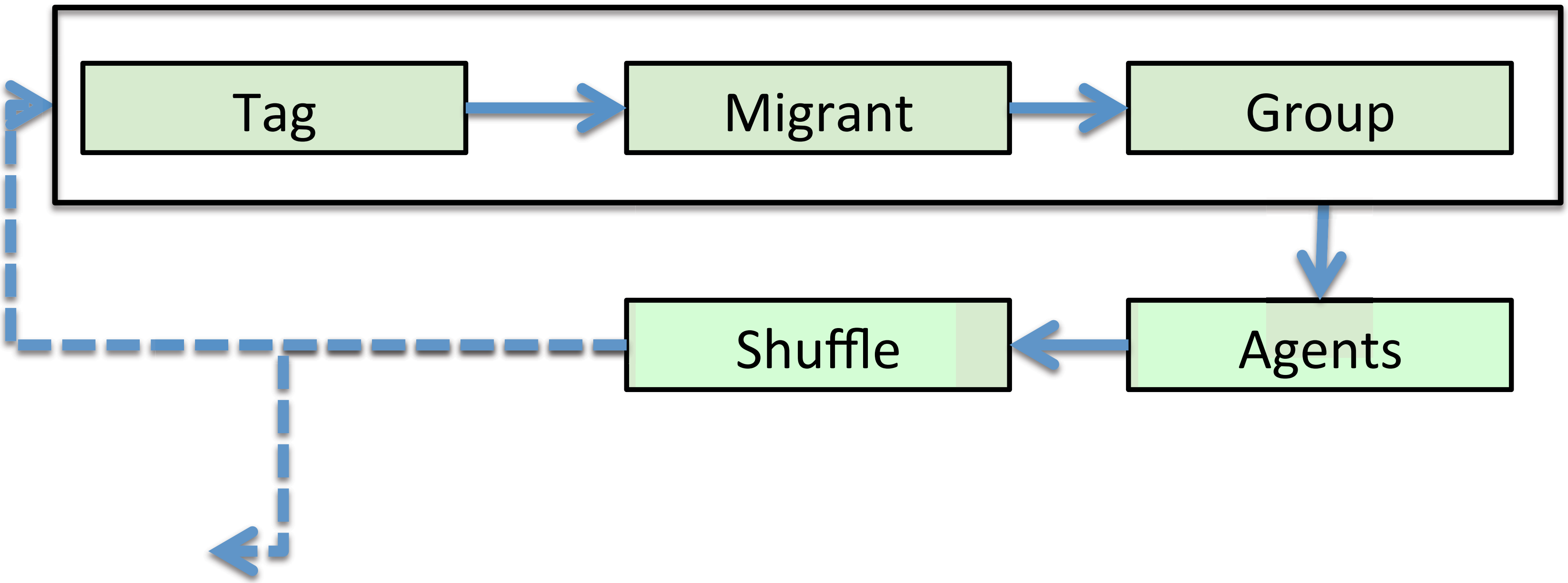
MAS – Pattern Discovery

Introduce Farm



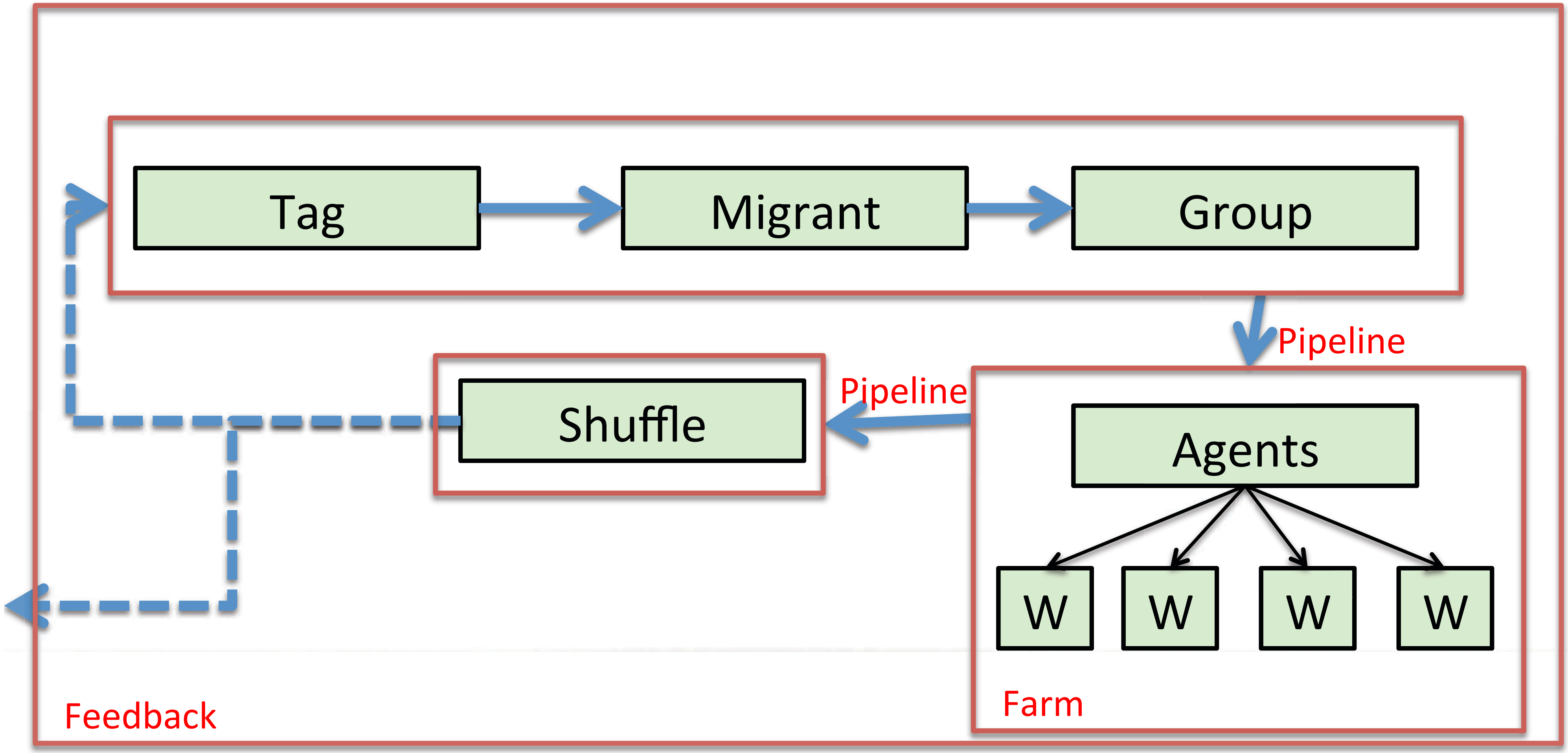
MAS – Program Shaping

Group together stages and remove dependencies

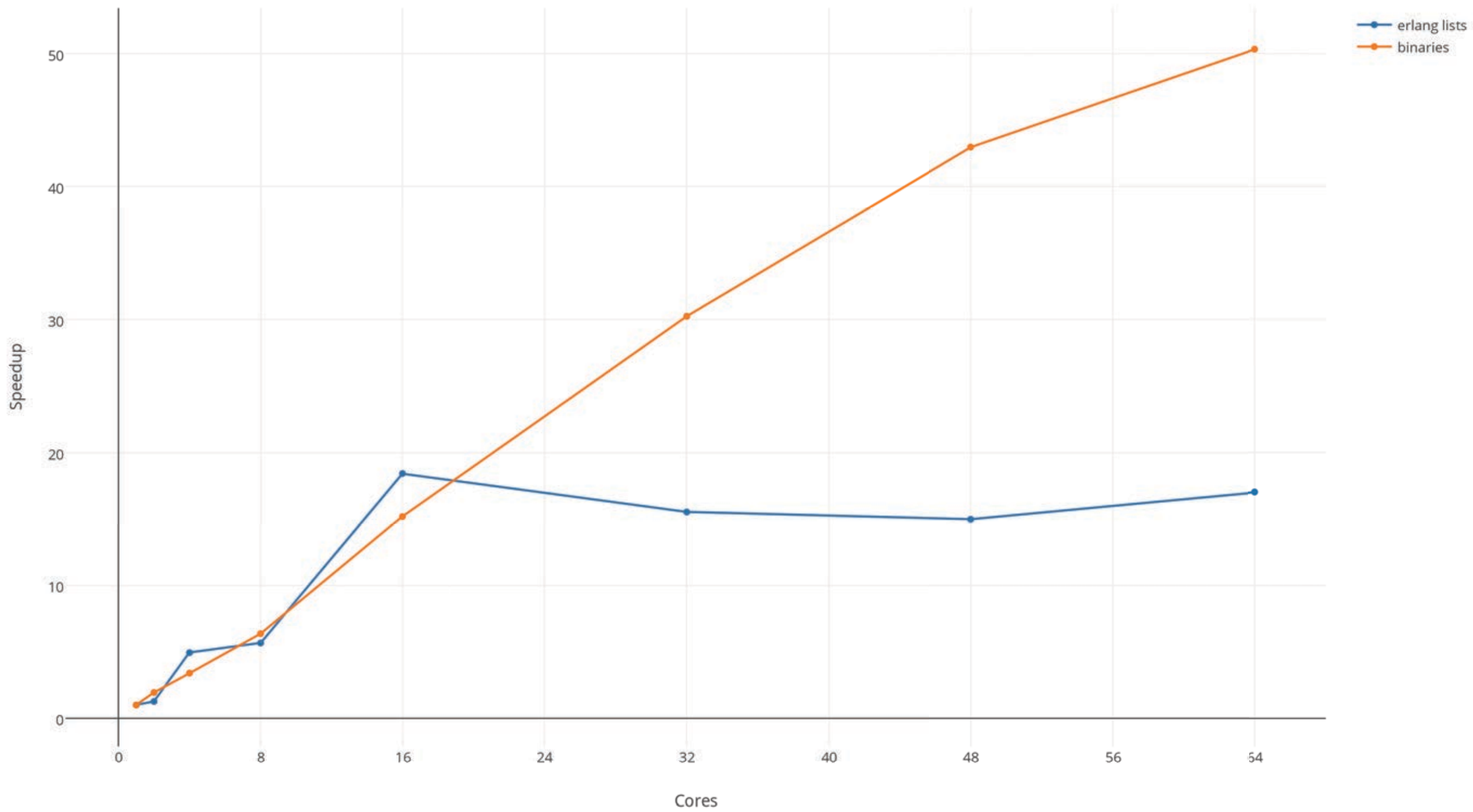


MAS – Advanced Refactoring

Feedback with Pipeline and Farm



MAS - Results


erlang lists
binaries
Speedup
0
10
20
30
40
50
0
8
16
24
32
40
48
56
64
Cores

EMAS Erlang Implementations

Sequential: The population is processed by a single process. It is split between groups of agents having the same behaviour on the same island.

Concurrent: Every agent is represented by a different process and all communication uses message-passing. Agent interactions are mediated by “meeting arenas”.

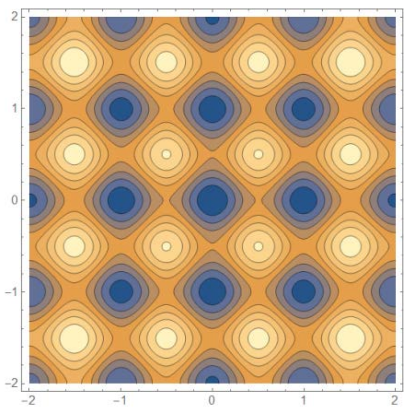
SKEL: The sequential implementation refactored into a SKEL workflow. Independent agent meetings are mapped and performed in parallel.

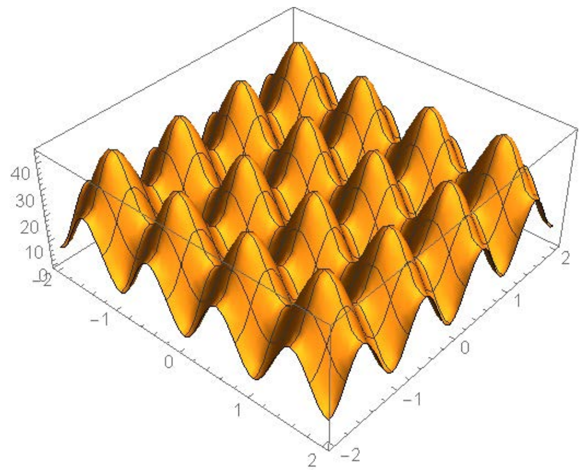
Hybrid SKEL: Every island is processed by a different process in the same way as in the sequential version.

Optimization Benchmark

- Find optimum of Rastrigin function in dimensions $n=100$
  - $f(x)=10n+\sum_{i=1}^{n}\left(x_i^2-10\cos(2\pi x_i)\right)$
  - One of classic global optimization benchmark functions

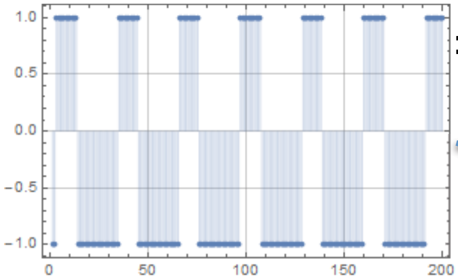
- Example: Rastrigin function in two dimensions

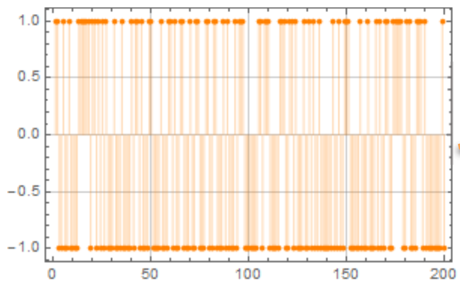


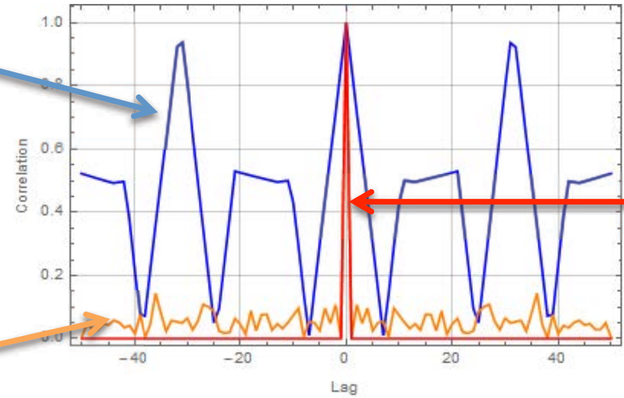


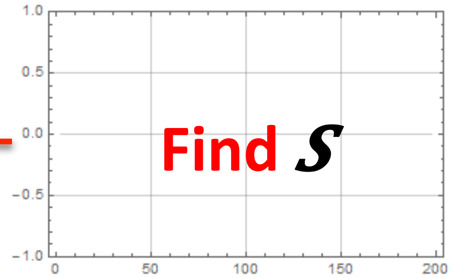
LABS

Low-Autocorrelation Binary Sequences

- $S = s_1 s_2 \ldots s_L$: binary sequence of length L and $s_i \in \{-1,+1\}$
- Aperiodic Autocorrelation with lag k : $C_k(S) = \sum_{i=1}^{L-k} s_i s_{i+k}$
-  $= \sum_{k=1}^{L-1} C_k^2(S)$ with respect to S







EMAS : Speed-Up

Rastringin Problem

Computation / Communication = Low

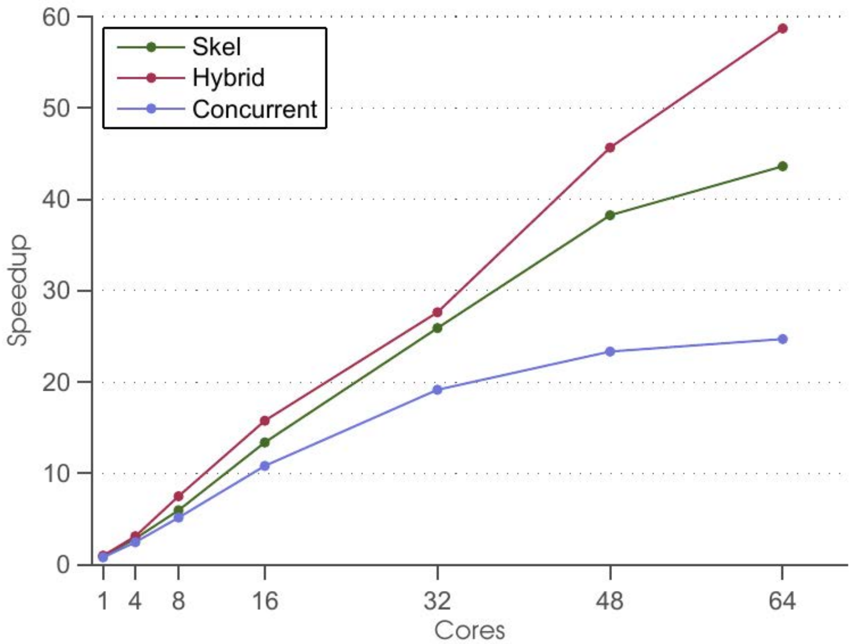


Figure 2.1: MAS versions speedup for the Rastrigin problem.

LABS Problem

Computation / Communication = High

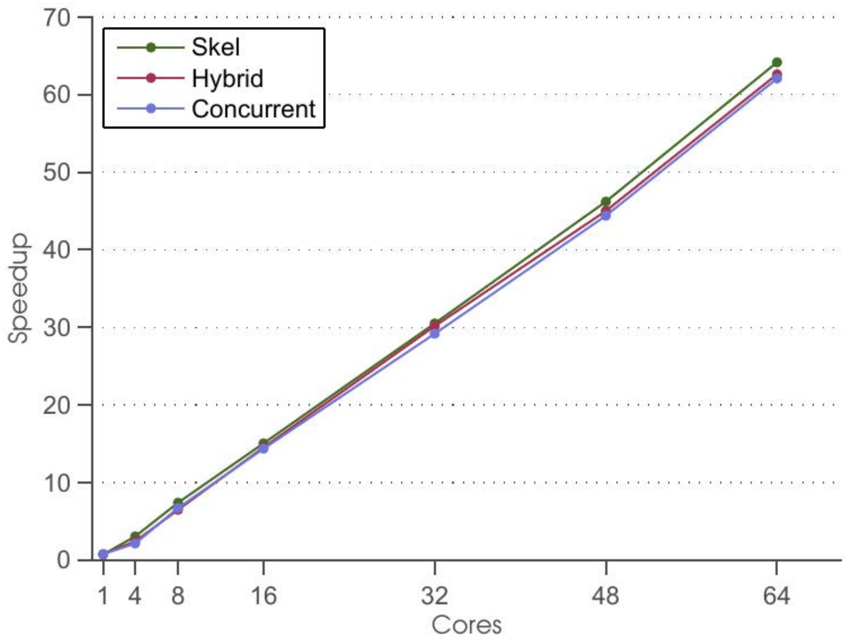


Figure 2.2: MAS versions speedup for the LABS problem.

EMAS : Coding Efficiency

- Effort for implementing the generic EMAS backends

	Lines of Code	Effort in Days
Sequential	85	10
Concurrent	353	7
SKEL	**100**	**1**

EMAS : Coding Efficiency

University of St Andrews

- Effort for implementing the generic EMAS backends

	Lines of Code	Effort in Days
Sequential	85	10
Hybrid	129	2
Concurrent	353	7
SKEL	**100**	**1**

EMAS : Coding Efficiency

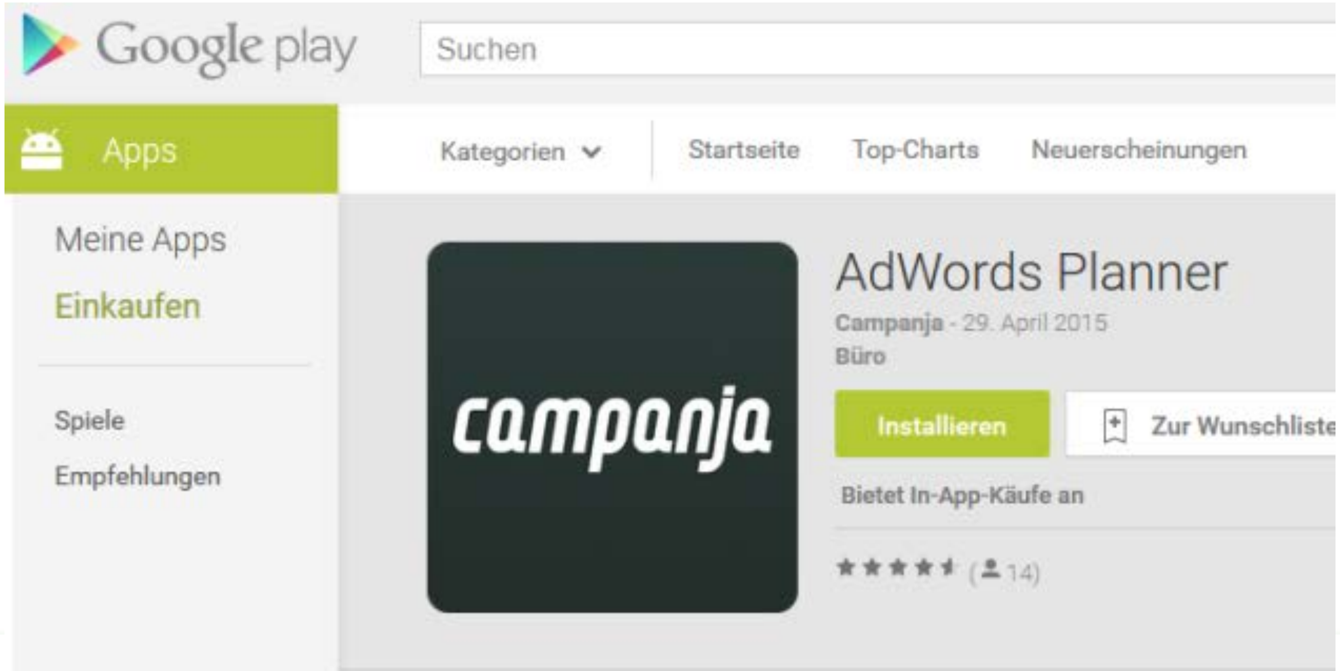
University of St Andrews

- Effort for implementing the generic EMAS backends

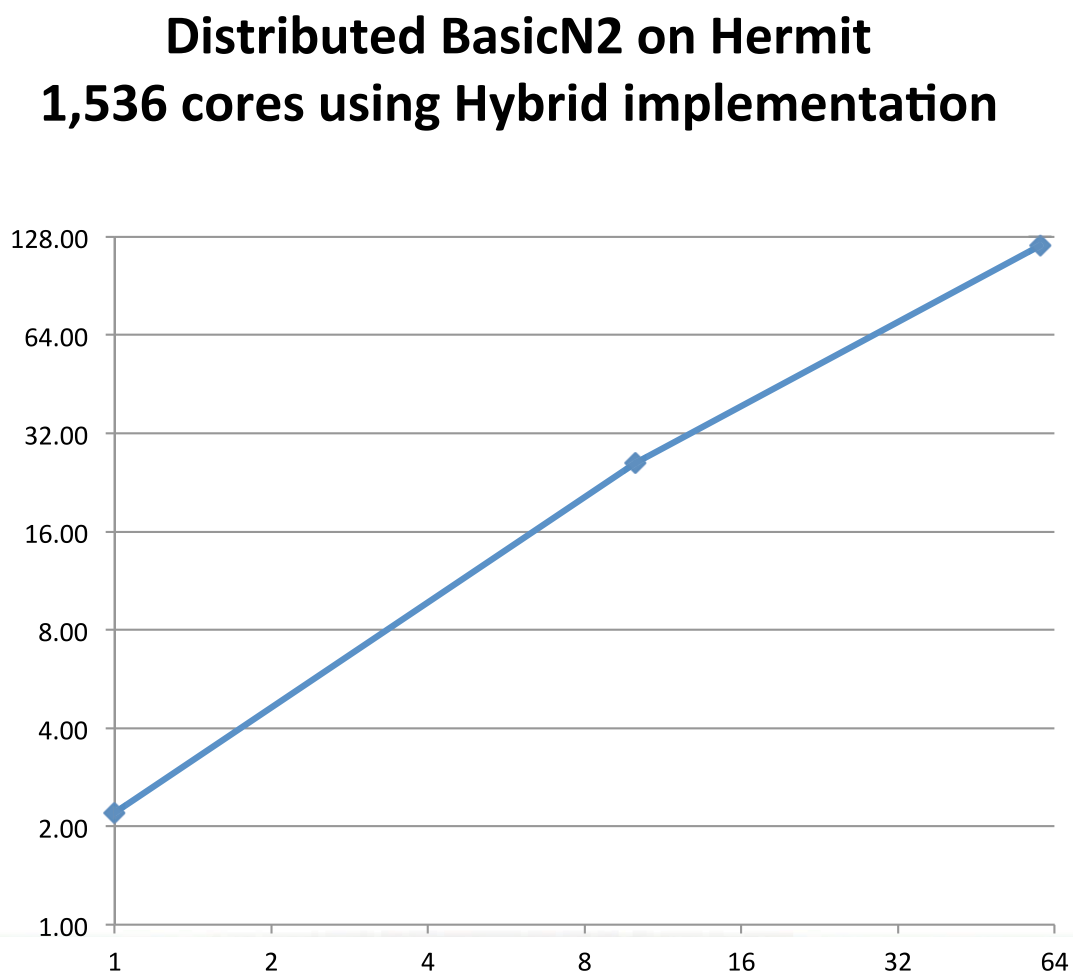
	Lines of Code	Effort in Days
Sequential	85	10
Concurrent	353	7
SKEL	**100**	**1**
Hybrid SKEL	129	2

EMAS in Production

- ParaMAS is used in **Campanja**'s product **AdWords Planner**, dealing with advertisement campaigns.
- Client available at Google Play



Well does it?



Distributed BasicN2 on Hermit
1,536 cores using Hybrid implementation
128.00
64.00
32.00
16.00
8.00
4.00
2.00
1.00
1
2
4
8
16
32
64

ParaPhrase Success

University of St Andrews

- Applications from different areas have successfully been parallelized
- Programmer productivity was significantly increased by the availability of new generic as well as domain-specific patterns
- Speedups close to the expected theoretical value
- Automatic pattern candidate discovery techniques can indeed find meaningful patterns in (Erlang) code bases
- ParaPhrase technology is used in production code
- Heterogeneous patterns provide a unified approach, rather than using different programming paradigms for parallelizing CPU and GPU codes

ParaPhrase Success

University of St Andrews

- Applications from different areas have successfully been parallelized
- Programmer productivity was significantly increased by the availability of new generic as well as domain-specific patterns
- Speedups close to the expected theoretical value
- Automatic pattern candidate discovery techniques can indeed find meaningful patterns in (Erlang) code bases
- ParaPhrase technology is used in production code
- Heterogeneous patterns provide a unified approach, rather than using different programming paradigms for parallelizing CPU and GPU codes

It's not just about large systems

- Even mobile phones are multicore
  - Samsung Exynos 5 Octa has 8 cores, 4 of which are Dark
- Performance/energy tradeoffs mean systems will be increasingly parallel
- Even embedded systems are becoming multicore and heterogeneous
  - NVidia Tegra TK1 has integrated 5 ARM CPU cores and 192-core Kepler GPU
- If we don't solve the multicore challenge, then no other computing advances will matter!

ALL Future Programming will be Parallel!

An Endorsement from a Happy Customer

University of St Andrews

Functions deal with heterogeneity (e.g. CPUs and GPUs at the same time)

University of St Andrews

- Closures can be compiled differently for different platforms.

```
-- Haskell

data Procs = CPU | GPU

brick CPU = ... parmap ...
brick GPU = ... Data.Accelerate.map ...
```

- The RTS can choose dynamically between closure types

```
brick (if  cost brick CPU < cost brick GPU then CPU else GPU) data
```

Lapedo: a Framework for Hybrid Skeletons

- Extends **Skel** for Erlang with hybrid skeletons for GPU/CPU computations
- Builds on the **CL** library for interfacing to OpenCL code
- New refactorings for:
  - introducing hybrid skeletons
  - switching between CPU/GPU implementations
  - semi-automatic code generation

```
https://github.com/ParaPhrase/skel
```

Example: Introducing Hybrid Skeleton

```
nbody(Particles,0) -> Particles;
nbody(Particles, NIters) ->
  NewParticles = lists:map (fun(X) -> nbody_cpu (X,Parts) end, Particles),
  nbody(NewParticles, NIters-1).
```

ParMapIntro Refactoring

```
nbody(Particles,0)->Particles;
nbody(Particles,NIters) ->
 Map = {map, [{seq, fun(X) -> map (fun nbody_cpu/1, X) end}],
         fun split/1,
         fun combine/1},
 NewParticles = skel:do([Map],[Particles]),
 nbody(NewParticles,NIters-1).
```

HybMapIntro Refactoring

```
iter = fun(NCPU, NGPU) ->
 Map = {map, [{seq, fun(X) -> het_map:het_dispatcher(
                              fun(Y) -> nbody_cpu(Y,Particles,0.0001) end,
                              fun(Y) -> nbody_gpu(Y,Particles,0.0001) end,
                              X) end}],
          fun(X) -> het_map:het_split(fun split/2,X,NCPU,NGPU) end,
          fun combine/1},
  Results = skel:do([Map],[Particles]),
  Result.
```

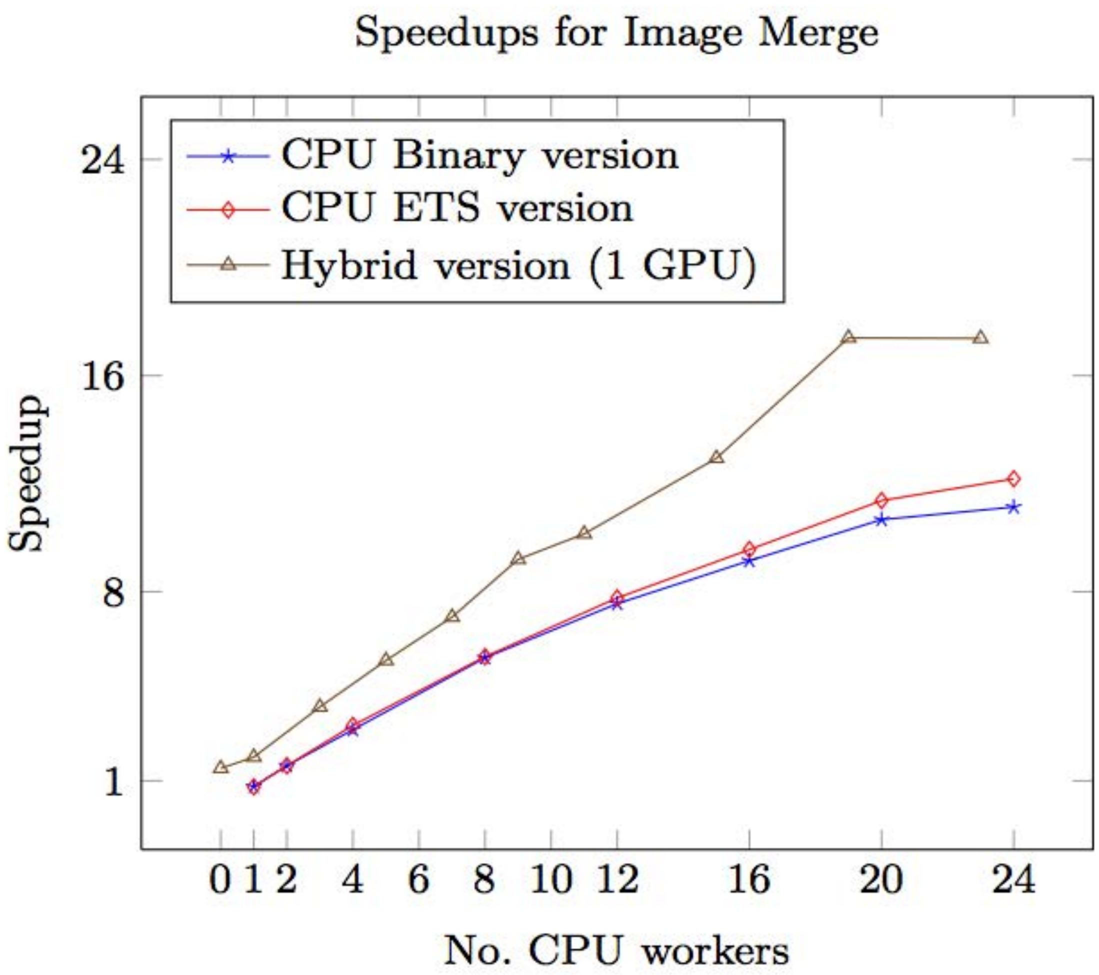
The CL Erlang Library

University of St Andrews

- Provides Simplified bindings to C OpenCL functions
- Supports data transfers between CPU and GPU and GPU kernel execution
- Basic marshalling mechanisms – from Erlang Binaries to C Arrays

```
E = clu:setup(all),

{ok,Program} = clu:build_source(E, Source),

{ok,Kernel} = cl:create_kernel(Program,
 "imageMergeKernel"),

...
```

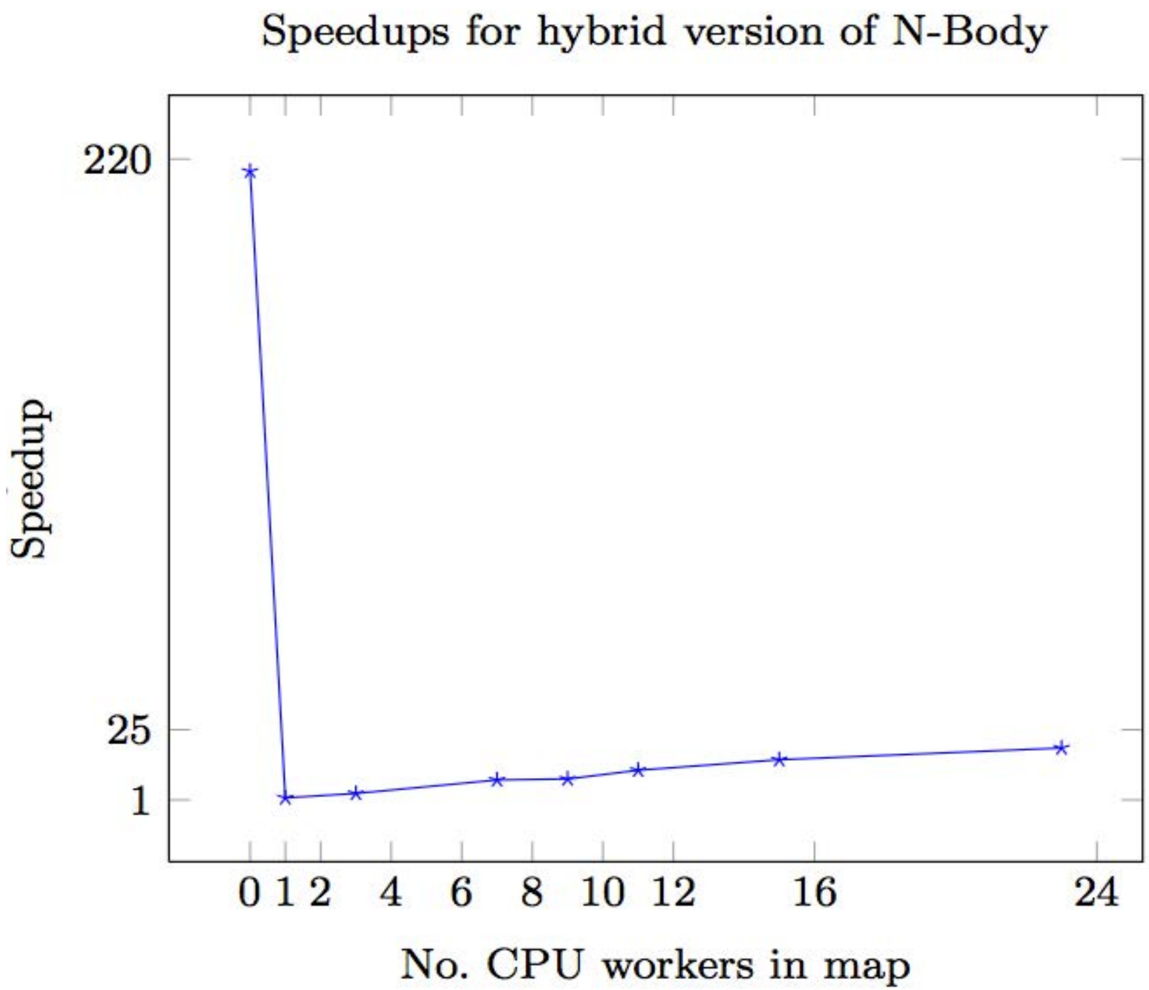
Speedups for Image Merge



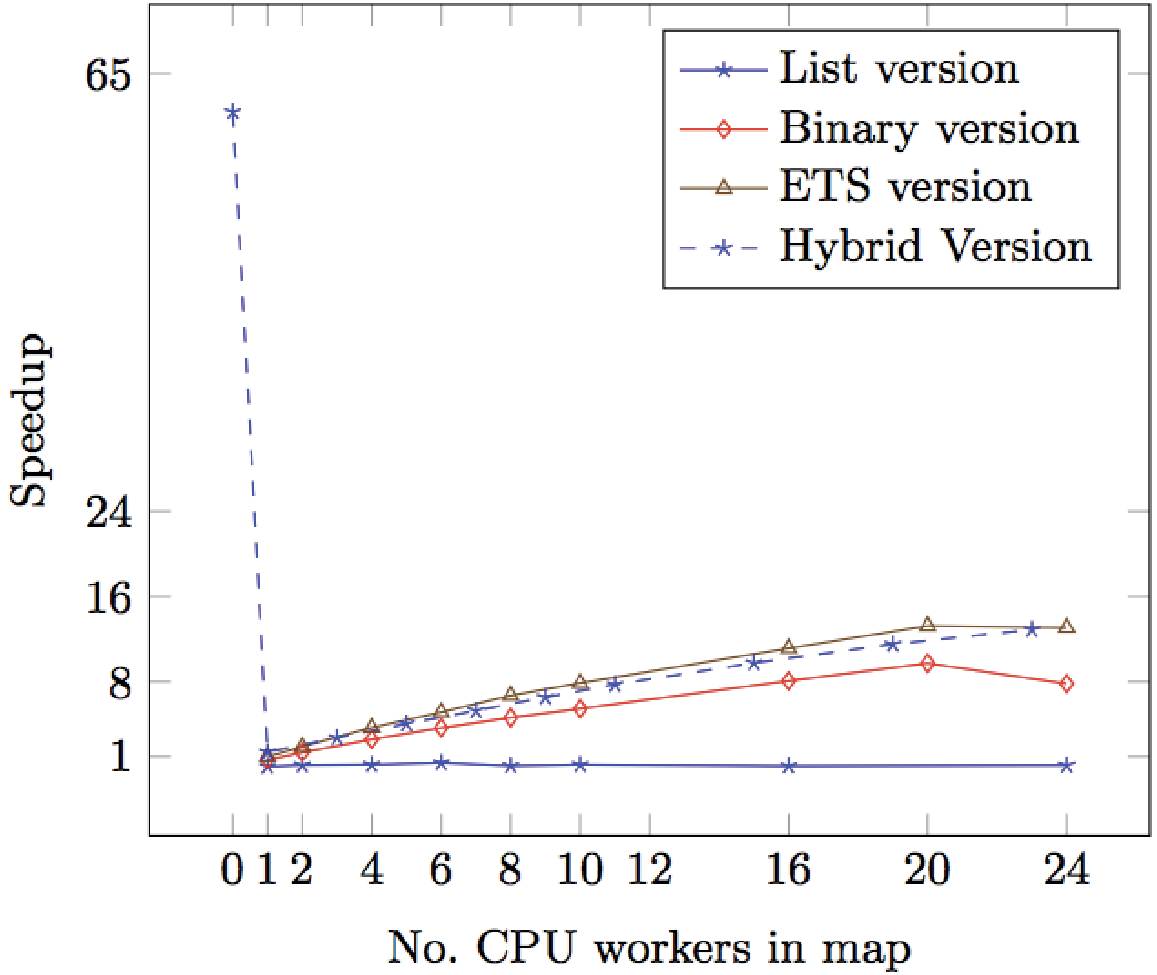
2 x AMD Opteron 6176	(24 CPU cores at 800MHz)
1 x NVidia Tesla Fermi C2050	(448 GPU cores @ 1.15GHz)

Speedups for Nbody Simulation

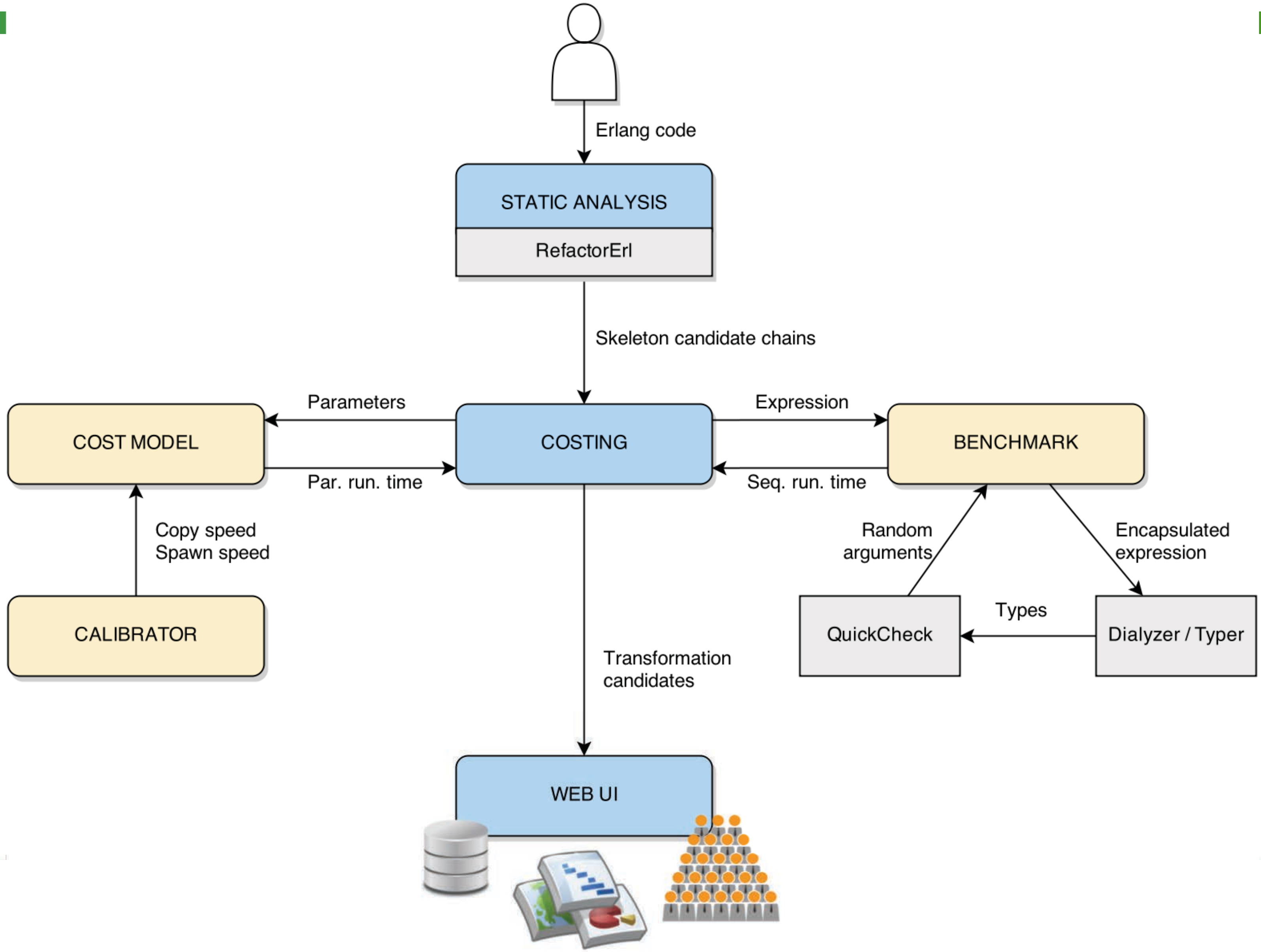
University of St Andrews



Speedups for Ant Colony Optimisation



Automatic Pattern Discovery



Automatic Pattern Discovery

localhost:34307/paraphrase/refpp_web_services/output

Pattern Candidate Browser

Transformation sequences

ID	Configuration	Module	Function	Arity	Number of workers	Expected speedup (CPU)	Expected speedup (GPU)	Recommended?
38	(!(!e9965))	matMult	theSkel	2	340	304.23	1.00	✔
41	(!(!e11501))	matMult2	theSkel	2	340	304.23	1.00	✔
46	(!(!e12819))	matrix	mult_seq_1	2	340	276.45	1.00	✔
30	(!(!e11715))	matMult2	run_all_examples	1	260	254.07	1.00	✔
53	(!(!e13496))	matrix_ex	mult_seq	2	340	245.93	1.00	✔
54	(!(!(!e13548)))	matrix_ex	mult_seq	2	340	226.05	1.00	✔
8	(!e11630)	matMult2	randmat	3	257	173.76	1.00	✔
5	(!e10101)	matMult	randmat	3	257	173.21	1.00	✔
11	(!e13256)	matrix	randmat	3	257	171.88	1.00	✔
3	(!e8681)	main	randmat	3	257	169.11	1.00	✔

< > 1 2 3 4 5 6

Chart options ▼

Details of the transformation sequence

Configuration	Location information	Program text	Number of workers	Sequential CPU time	Sequential GPU time	Parallel CPU time	Parallel GPU time	Expected speedup (CPU)	Expected speedup (GPU)	Used stream length
e12819	/home/v/work/paraphrase/repo/referl /tool/matrix/matrix.erl : {{37,16},{37,18}} - {{37, 40}, {37, 42}}	fun(C) -> multSum(R, C) end	1	0.51	0.00	0.51	0.00	1.00	1.00	1
(!e12819)	/home/v/work/paraphrase/repo/referl /tool/matrix/matrix.erl : {{37,6},{37,10}} - {{37, 49}, {37, 49}}	lists:map(fun(C) -> multSum(R, C) end, Cols)	2	5,050.16	0.00	3,080.58	0.00	1.64	1.00	10,000
(!(!e12819))	/home/v/work/paraphrase/repo/referl /tool/matrix/matrix.erl : {{36,27},{36,31}} - {{38, 18}, {38, 18}}	lists:map(fun(R) -> lists:map(fun(C) -> multSum(R, C) end, Cols) end, Rows)	170	50,501,604.68	0.00	182,678.93	0.00	276.45	1.00	10,000

Chart options ▼

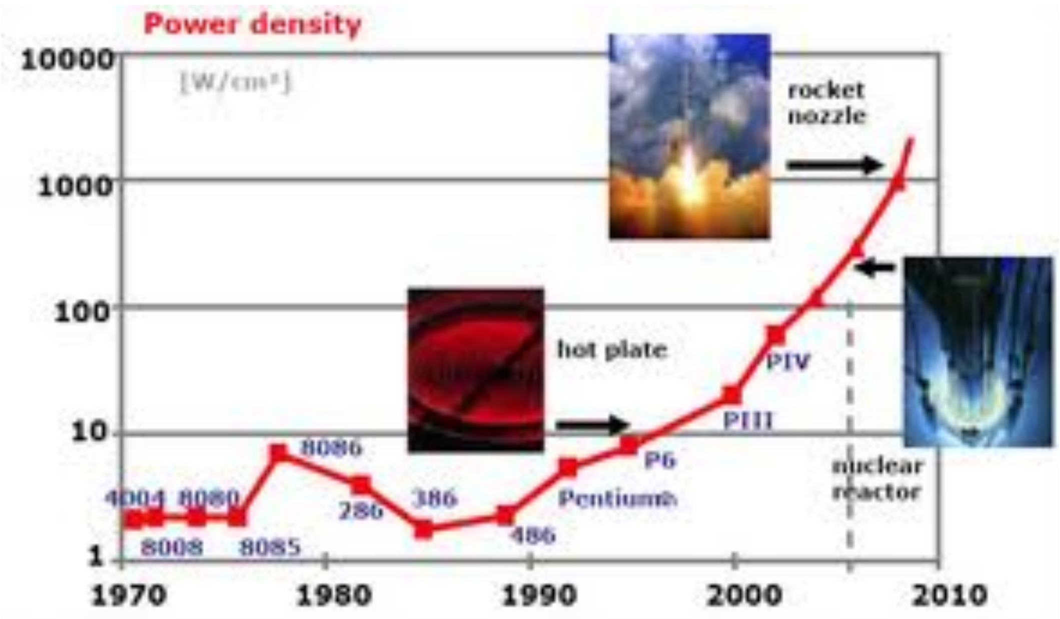
Is this megacool?

University of St Andrews

Image by Daniel Case

Or megahot?

- Energy usage scales:
  - **linearly** with the *number of cores*
  - **cubically** with the *clock frequency*
- Power density is critical
  - smaller process sizes (e.g. 22nm) need less energy
  - But a core 1/30 of the size will still consume 1/8 of the power!
  - We are reaching limits on heat dissipation!



Source: Patterson and Hennessey

- Efficient use of Energy is a major concern

How Functional Programming can Help

University of St Andrews

- Lots of small tasks are better than a few big ones
  - *can use more lower-powered cores*
  - *easy to do this with closure-based techniques*

- Functional programs can be easily parameterised
  - e.g. with energy models, performance costs

- Information can even be lifted into a type!

Source: Patterson and Hennessey

Conclusions

University of St Andrews

- The manycore revolution is upon us
  - Computer hardware is changing very rapidly (more than in the last 50 years)
  - The **megacore** era is already here! (aka exascale, BIG data)
  - **Heterogeneity** and **energy** are both important
- Most programming models are too low-level
  - concurrency based
  - unable to expose mass parallelism
- *Patterns* and *functional programming* greatly aid abstraction
  - millions of threads, easily controlled
    - easy scalability, deals with heterogeneity, can deal with dark silicon
  - (pure) closures and higher-order functions are key to unlocking megacore!

Some Open Research Challenges

- How do we deal with processor hierarchies?
- How do we allocate data to parallel hardware?
- What are the best parallel patterns to use
  - and what are the best implementations of those patterns?
  - do we need to alter patterns to include energy etc??
- How can we find instances of patterns in code?
- How do we find the best mapping to heterogeneous processors?

Conclusions (2)

University of St Andrews

- Functional programming makes it easy to introduce parallelism
  - No side effects means any computation could be parallel
  - Matches pattern-based parallelism
  - Much detail can be abstracted
- Lots of problems can be avoided
  - e.g. Freedom from Deadlock
  - Parallel programs give the same results as sequential ones!
- Automation is very important
  - Refactoring dramatically reduces development time (while keeping the programmer in the loop)
  - Machine learning is very promising for determining complex performance settings

Some of our Industrial Connections

University of St Andrews

Erlang Solutions Ltd
IBM
EvoPro Innovation
PRQA Programming Research
Roke Manor
SAP GmbH, Karlsrühe
BAe Systems
Selex Galileo
Biold GmbH, Stuttgart
Philips Healthcare
Software Competence Centre, Hagenberg
Microsoft Research
Well-Typed LLC
Mellanox Inc.

But isn't this all just wishful thinking?

NO!

- C++11/14 has lambda functions (and some other nice functional-inspired features)
- C++17 will add parallelism as well as concurrency
- Java 8 has lambda and futures
- Swift has first-class functions (and can support futures)

ParaPhrase Parallel C++ Refactoring

- Integrated into Eclipse
- Supports full C++(11) standard
- Uses strongly hygienic components
  - functional encapsulation (closures)
- Transfers our functional ideas to C++

Further Reading

Chris Brown. Marco Danelutto, Kevin Hammond, Peter Kilpatrick and Sam Elliot
“Cost-Directed Refactoring for Parallel Erlang Programs”
To appear in International Journal of Parallel Programming, 2014

John McCall, Mehdi Goli, Vladimir Janjic, Chris Brown and Kevin Hammond
“Using Machine Learning to Derive Mappings for Heterogeneous Parallel Computations”
2013 IEEE Congress on Evolutionary Computing.

Chris … Wolfgang Loidl and Kevin Hammond
… allel Haskell Programs using Novel Refactoring Techniques”
… rogramming (TFP), Madrid, Spain, May 2011

… Vladimir Janjic and Kevin Hammond
… ion Replay for Parallel Haskell Programs”
… n Functional Programming (TFP), St Andrews, UK, June 2012

Ask me for copies!
Many technical results also on the ParaPhrase web site:
free for download!

Funded by

- **RePhrase (EU H2020), Refactoring Parallel Heterogeneous Software – a Software Engineering Approach,**
  - €3.5M, 2015-2018
- **ParaPhrase (EU FP7), Patterns for heterogeneous multicore,**
  - €4.2M, 2011-2014
- **SCIEnce (EU FP6), Grid/Cloud/Multicore coordination**
  - €3.2M, 2005-2012
- **Advance (EU FP7), Multicore streaming**
  - €2.7M, 2010-2013
- **HPC-GAP (EPSRC), Legacy system on thousands of cores**
  - £1.6M, 2010-2014
- **TACLE: European Cost Action on Timing Analysis**
  - €300K, 2014-2017

University of St Andrews

SEVENTH FRAMEWORK PROGRAMME

RePhrase Project:Refactoring Parallel Heterogeneous Software – a Software Engineering Approach (ICT-644235), 2015-2018, €3.5M budget

8 Partners, 6 European countries

UK, Spain, Italy, Austria, Hungary, Israel

Coordinated by @khstandrews

IBM

evopro it's possible!

PRQA Programming Research

scch software competence center hagenberg

UNIVERSITÀ DI PISA

UNIVERSITÀ DEGLI STUDI DI TORINO ALMA UNIVERSITAS TAURINENSIS

PARAPHRASE

ParaPhrase Needs You!

- Please join our mailing list
  and help grow our user community
  - news items
  - access to free development software
  - chat to the developers
  - free developer workshops
  - bug tracking and fixing
  - Tools for both Erlang and C++

- Subscribe at

  https://mailman.cs.st-andrews.ac.uk/mailman/listinfo/paraphrase-news

- We're also looking for open source developers...

THANK YOU!

`http://www.rephrase-ict.eu`

`http://www.paraphrase-ict.eu`

`http://www.project-advance.eu`

@paraphrase_fp7, @rephrase_eu

kh@cs.st-andrews.ac.uk, kevin@kevinhammond.net