The ParaPhrase and RePhrase Projects: Programming Parallel Systems using High-Level Patterns

Kevin Hammond

University of St Andrews, Scotland

Invited Talk at PEPGUM 2015

Amsterdam, January 21st 2015

T: *@paraphrase_fp7*
E: *kh@cs.st-andrews.ac.uk*
W: `http://www.paraphrase-ict.eu`

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 Kevin Hammond St Andrews

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 Kevin Hammond St Andrews

IBM

evopro it's possible!

UNIVERSITÀ DEGLI STUDI DI TORINO ALMA UNIVERSITAS TAURINENSIS

PRQA Programming Research

scch software competence center hagenberg

UNIVERSITÀ DI PISA

All future programming will be parallel

- 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

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

An

The diffe … tions

will not ``automagically'' run t

actu

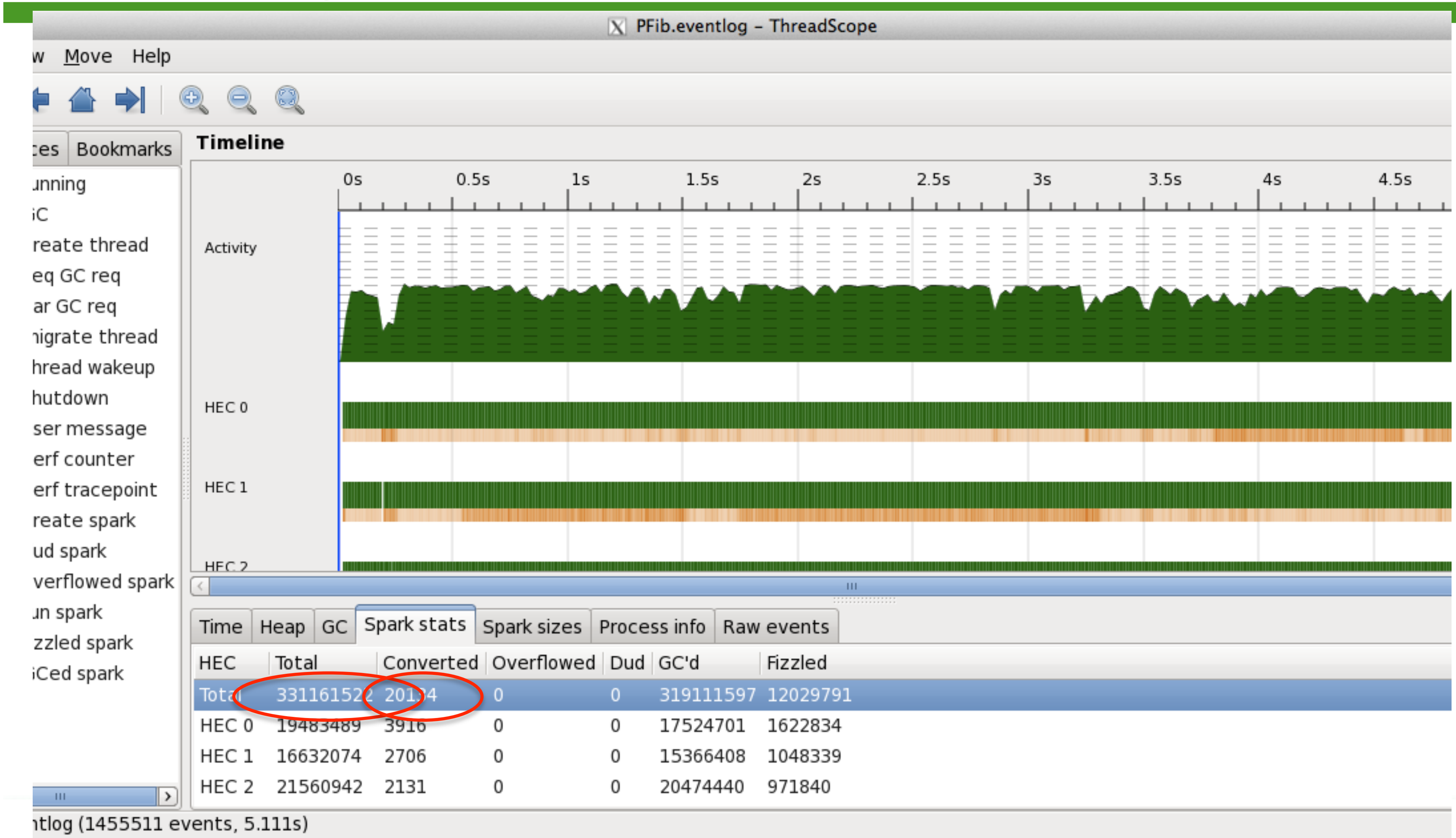
The **ONLY** important challenge in Computer Science

Intel

Also recognised as thematic priorities by EU and national bodies

Patrick Leonard, Vice President for Product Development
Rogue Wave Software

Doesn't that mean millions of threads on a megacore machine??



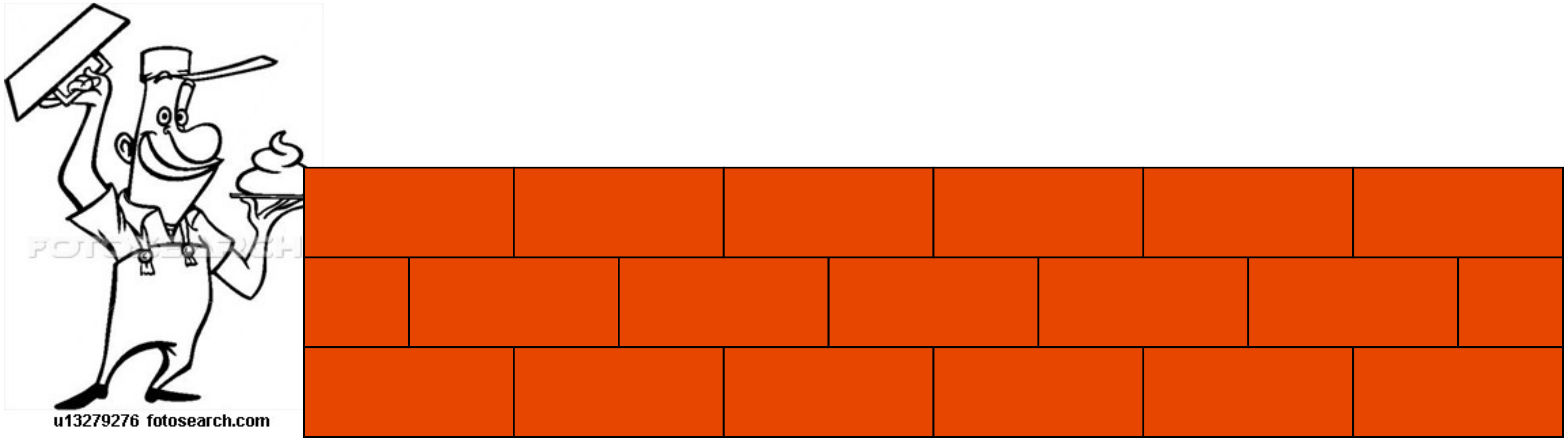
HEC	Total	Converted	Overflowed	Dud	GC'd	Fizzled
Total	331161522	20134	0	0	319111597	12029791
HEC 0	19483489	3916	0	0	17524701	1622834
HEC 1	16632074	2706	0	0	15366408	1048339
HEC 2	21560942	2131	0	0	20474440	971840

ıtlog (1455511 events, 5.111s)

What are we trying to achieve?

Parallelism and Concurrency

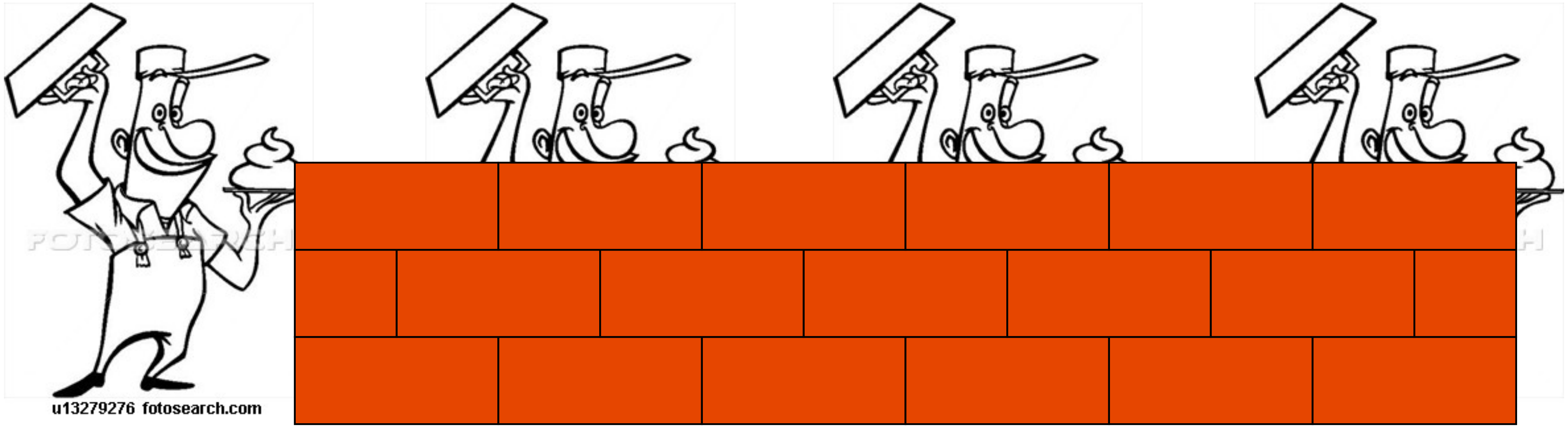
How to build a wall



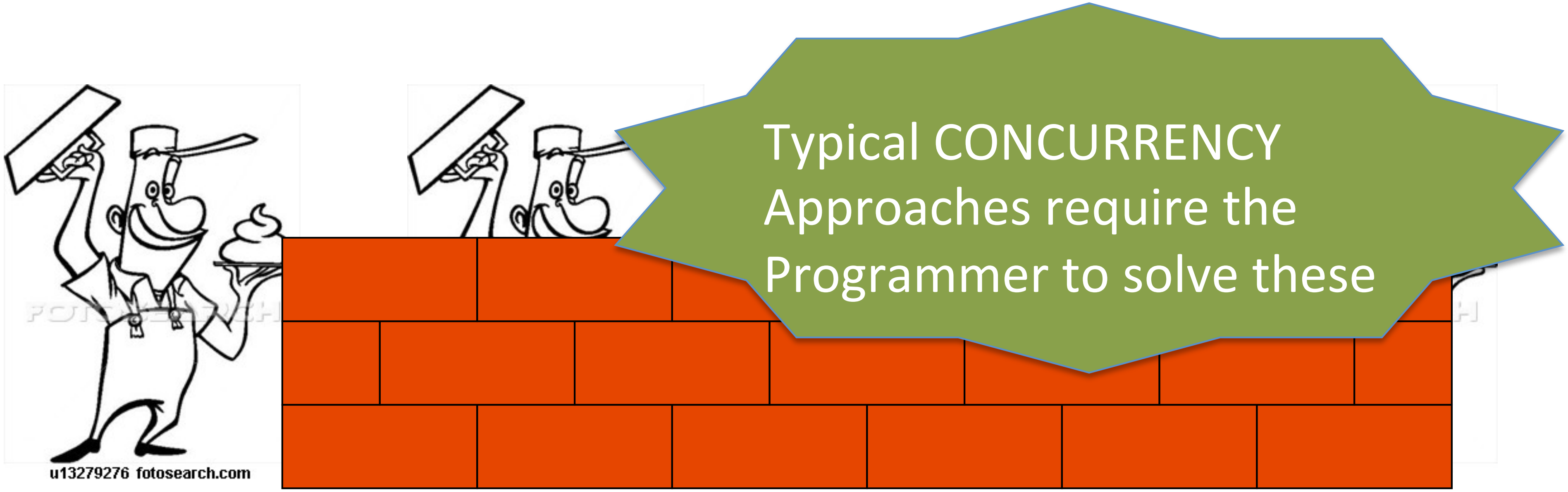
u13279276 fotosearch.com

(with apologies to Ian Watson, Univ. Manchester)

How to build a wall *faster*



How NOT to build a wall



Typical CONCURRENCY Approaches require the Programmer to solve these

u13279276 fotosearch.com

Task identification is not the only problem…

Must also consider Coordination, communication, placement, scheduling, …

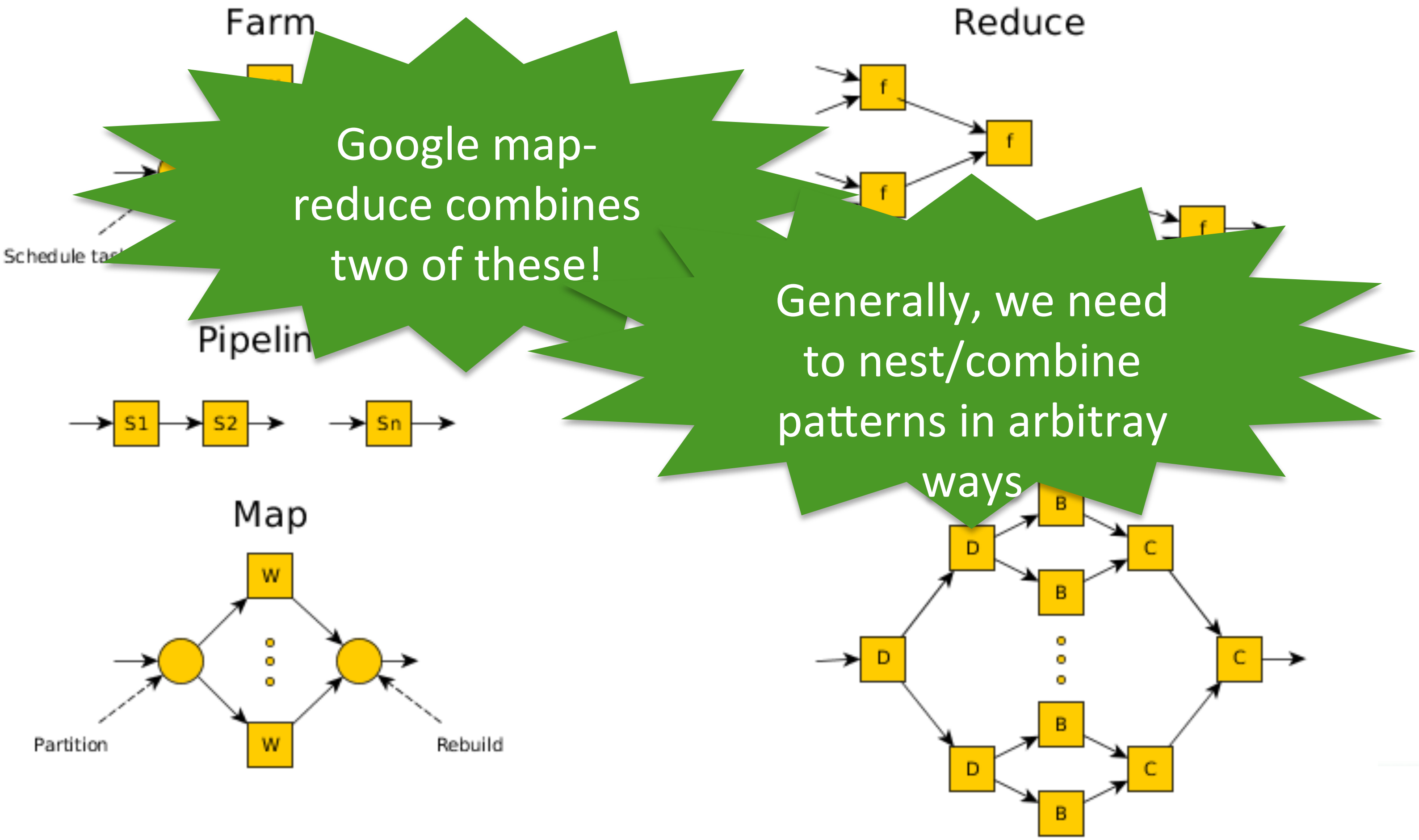
We need structure

We need abstraction

We don't need another brick in the wall

Some Common Patterns

- High-level abstract patterns of common parallel algorithms



Google map-reduce combines two of these!

Generally, we need to nest/combine patterns in arbitray ways

The ParaPhrase/RePhrase Approach

- Start bottom-up
  - identify (strongly hygienic) **COMPONENTS**
  - *using semi-automated refactoring*

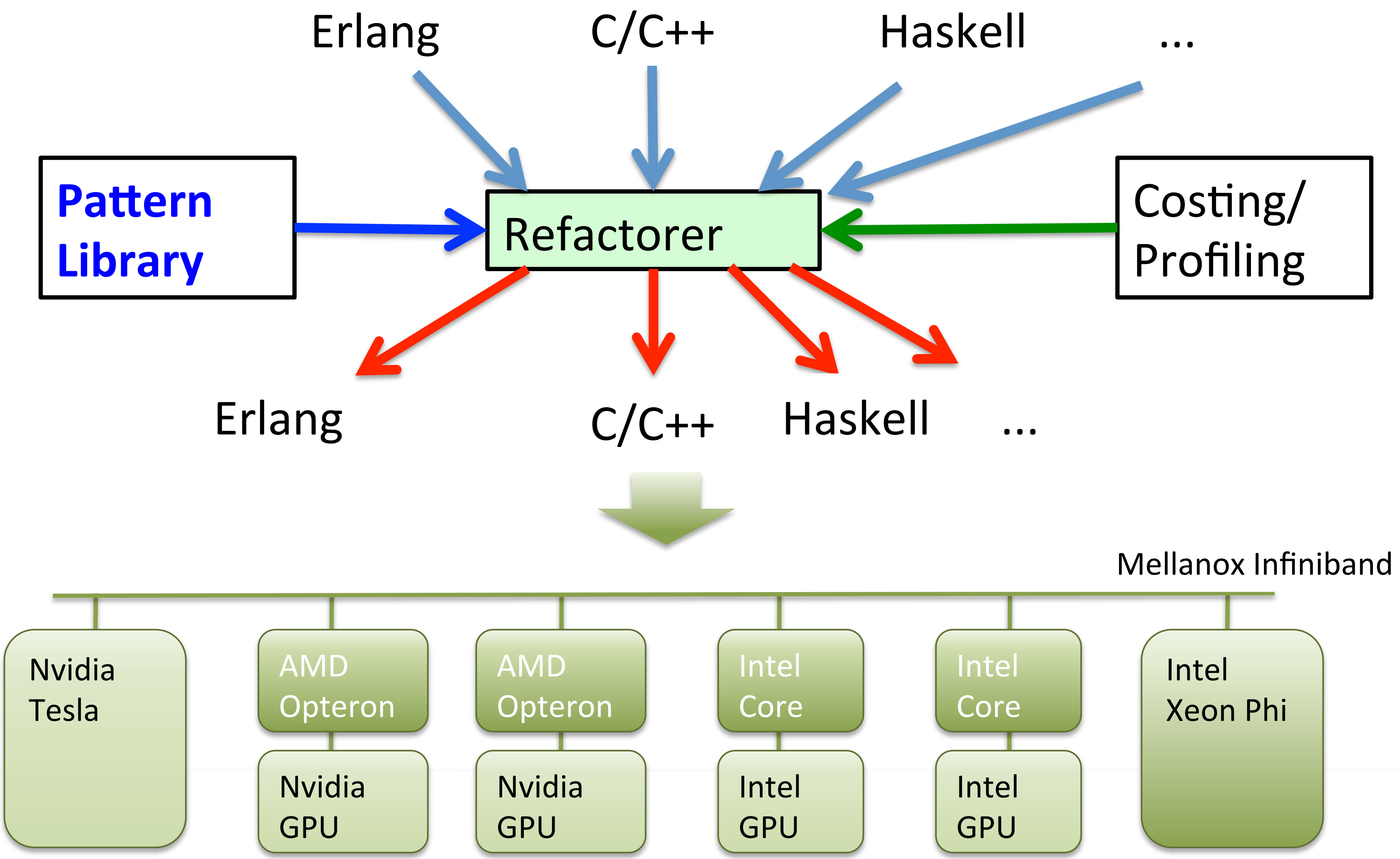
both legacy and new programs

- 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)

Components and Abstraction

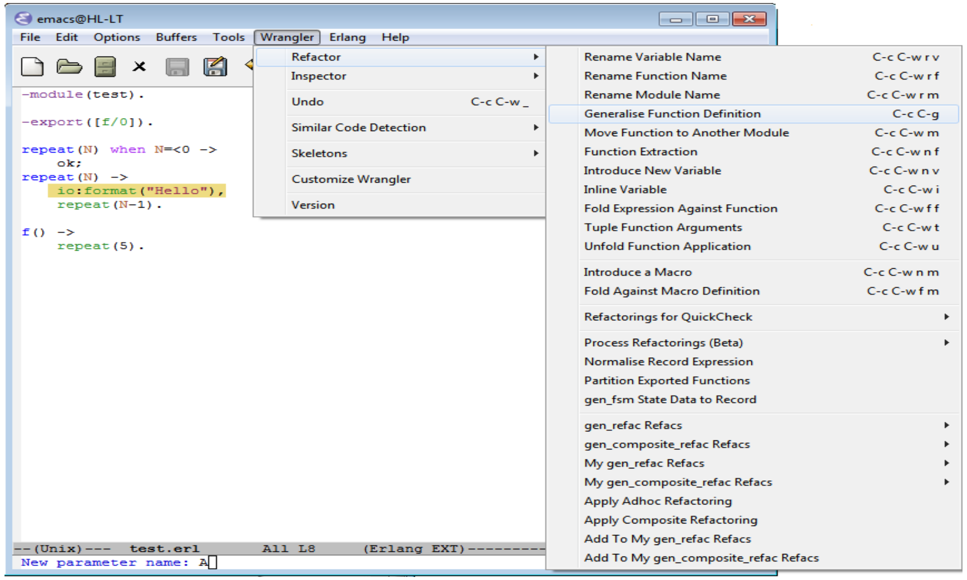
- Components give some of the advantages of functional programming
  - clean abstraction
  - pure computations, easily scheduled
  - dependencies can be exposed
- Hygiene/discipline is necessary
  - no unwanted state leakage
    (e.g. in terms of implicit shared memory state)

General Technique



Refactoring

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

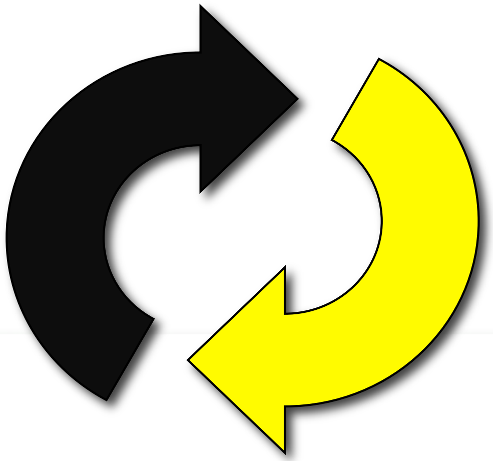


emacs@HL-LT
File Edit Options Buffers Tools Wrangler Erlang Help
Refactor
Inspector
Undo C-c C-w _
Similar Code Detection
Skeletons
Customize Wrangler
Version
Rename Variable Name C-c C-w r v
Rename Function Name C-c C-w r f
Rename Module Name C-c C-w r m
Generalise Function Definition C-c C-g
Move Function to Another Module C-c C-w m
Function Extraction C-c C-w n f
Introduce New Variable C-c C-w n v
Inline Variable C-c C-w i
Fold Expression Against Function C-c C-w f f
Tuple Function Arguments C-c C-w t
Unfold Function Application C-c C-w u
Introduce a Macro C-c C-w n m
Fold Against Macro Definition C-c C-w f m
Refactorings for QuickCheck
Process Refactorings (Beta)
Normalise Record Expression
Partition Exported Functions
gen_fsm State Data to Record
gen_refac Refacs
gen_composite_refac Refacs
My gen_refac Refacs
My gen_composite_refac Refacs
Apply Adhoc Refactoring
Apply Composite Refactoring
Add To My gen_refac Refacs
Add To My gen_composite_refac Refacs

```
-module(test).

-export([f/0]).

repeat(N) when N=<0 ->
    ok;
repeat(N) ->
    io:format("Hello"),
    repeat(N-1).

f() ->
    repeat(5).
```

```
--(Unix)---   test.erl      All L8      (Erlang EXT)---------
New parameter name: A
```

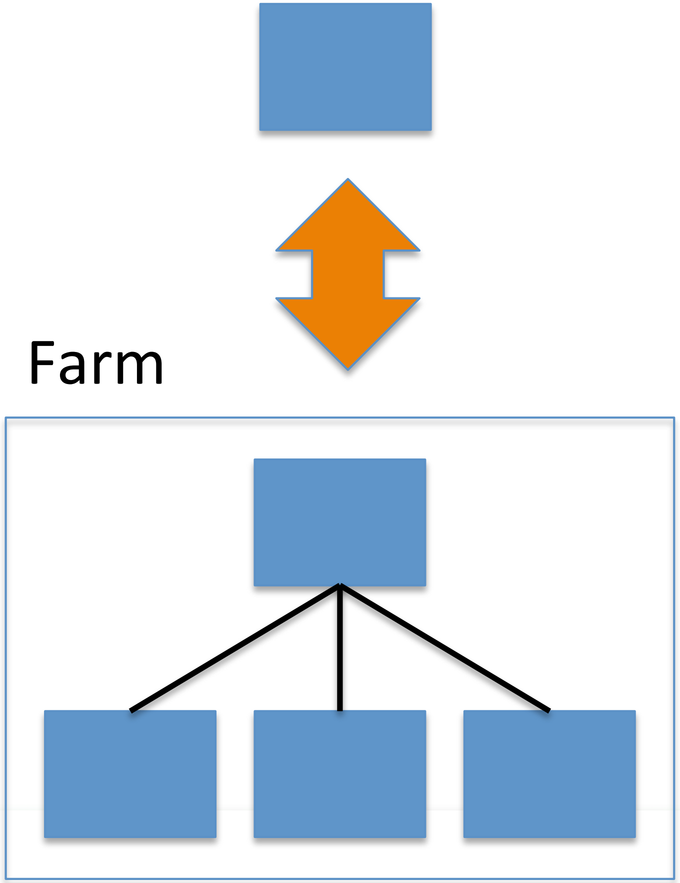
Review



Refactor

Refactoring: Farm Introduction

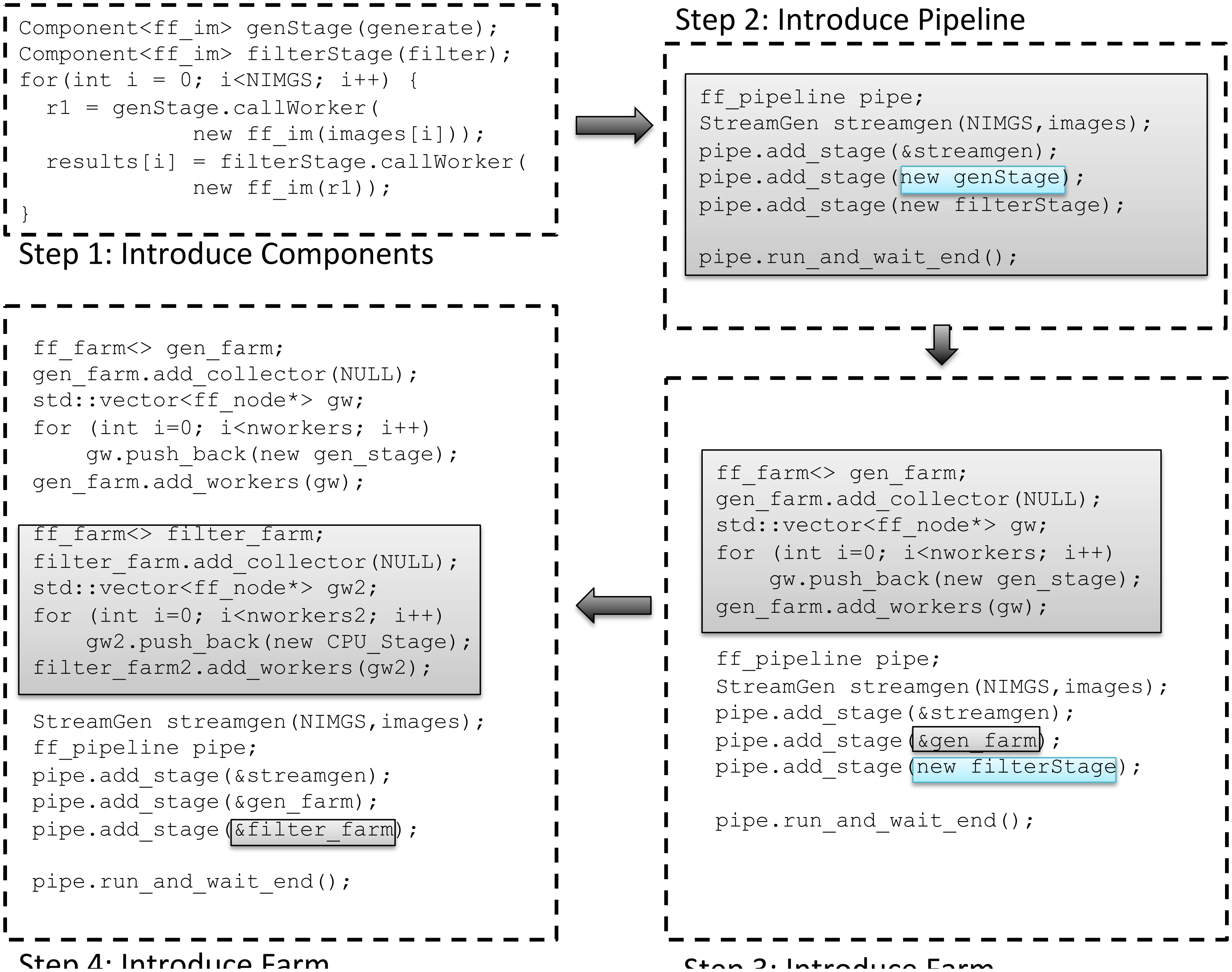
$S \equiv Farm(S)$ *farm intro/elim*



ParaPhrase Parallel C++ Refactoring

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

Image Convolution



```
Component<ff_im> genStage(generate);
Component<ff_im> filterStage(filter);
for(int i = 0; i<NIMGS; i++) {
  r1 = genStage.callWorker(
               new ff_im(images[i]));
  results[i] = filterStage.callWorker(
               new ff_im(r1));
}
```

Step 1: Introduce Components

Step 2: Introduce Pipeline

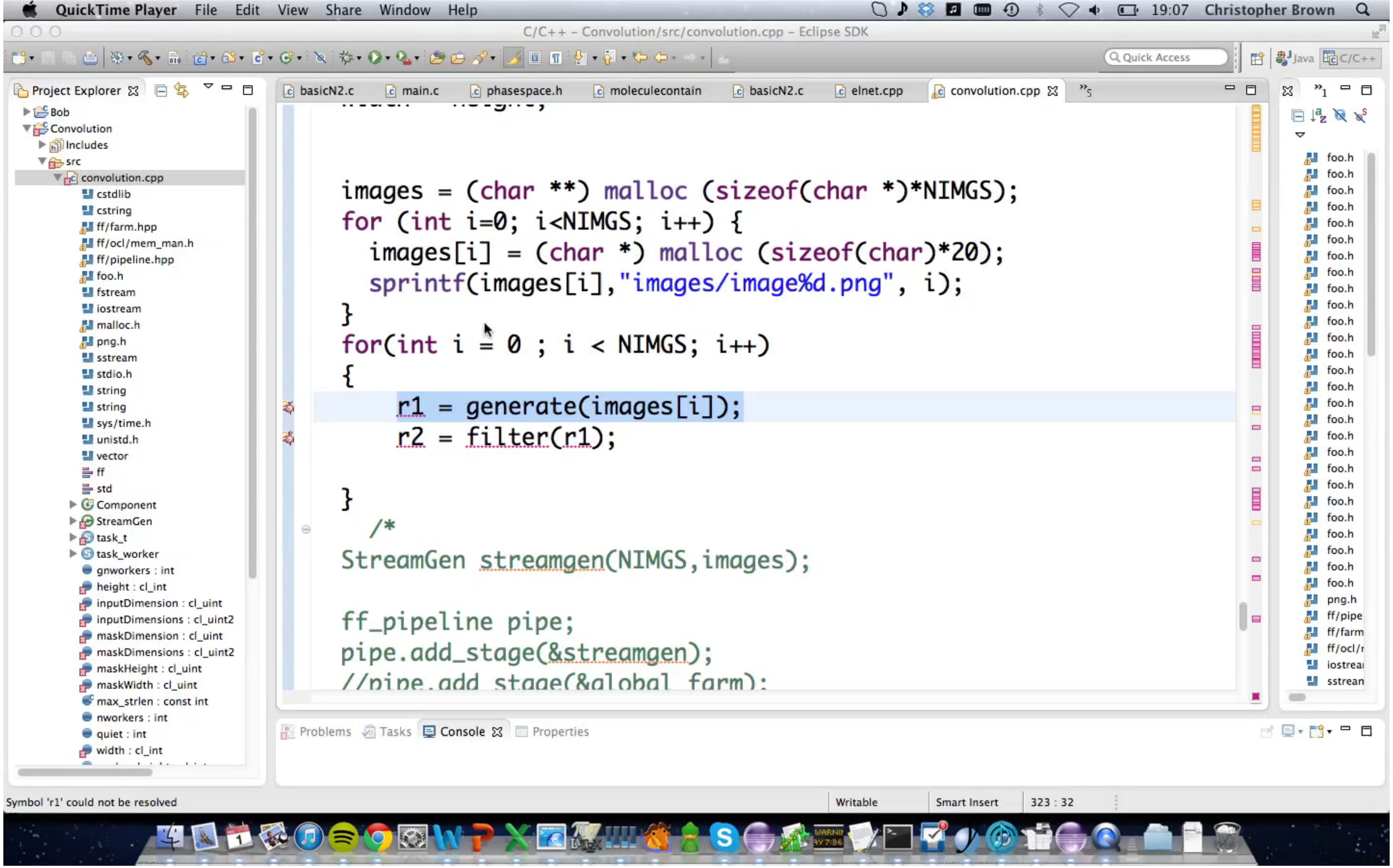
```
ff_pipeline pipe;
StreamGen streamgen(NIMGS,images);
pipe.add_stage(&streamgen);
pipe.add_stage(new genStage);
pipe.add_stage(new filterStage);

pipe.run_and_wait_end();
```

```
ff_farm<> gen_farm;
gen_farm.add_collector(NULL);
std::vector<ff_node*> gw;
for (int i=0; i<nworkers; i++)
    gw.push_back(new gen_stage);
gen_farm.add_workers(gw);

ff_pipeline pipe;
StreamGen streamgen(NIMGS,images);
pipe.add_stage(&streamgen);
pipe.add_stage(&gen_farm);
pipe.add_stage(new filterStage);

pipe.run_and_wait_end();
```

Step 3: Introduce Farm

```
ff_farm<> gen_farm;
gen_farm.add_collector(NULL);
std::vector<ff_node*> gw;
for (int i=0; i<nworkers; i++)
    gw.push_back(new gen_stage);
gen_farm.add_workers(gw);

ff_farm<> filter_farm;
filter_farm.add_collector(NULL);
std::vector<ff_node*> gw2;
for (int i=0; i<nworkers2; i++)
    gw2.push_back(new CPU_Stage);
filter_farm2.add_workers(gw2);

StreamGen streamgen(NIMGS,images);
ff_pipeline pipe;
pipe.add_stage(&streamgen);
pipe.add_stage(&gen_farm);
pipe.add_stage(&filter_farm);

pipe.run_and_wait_end();
```

Step 4: Introduce Farm

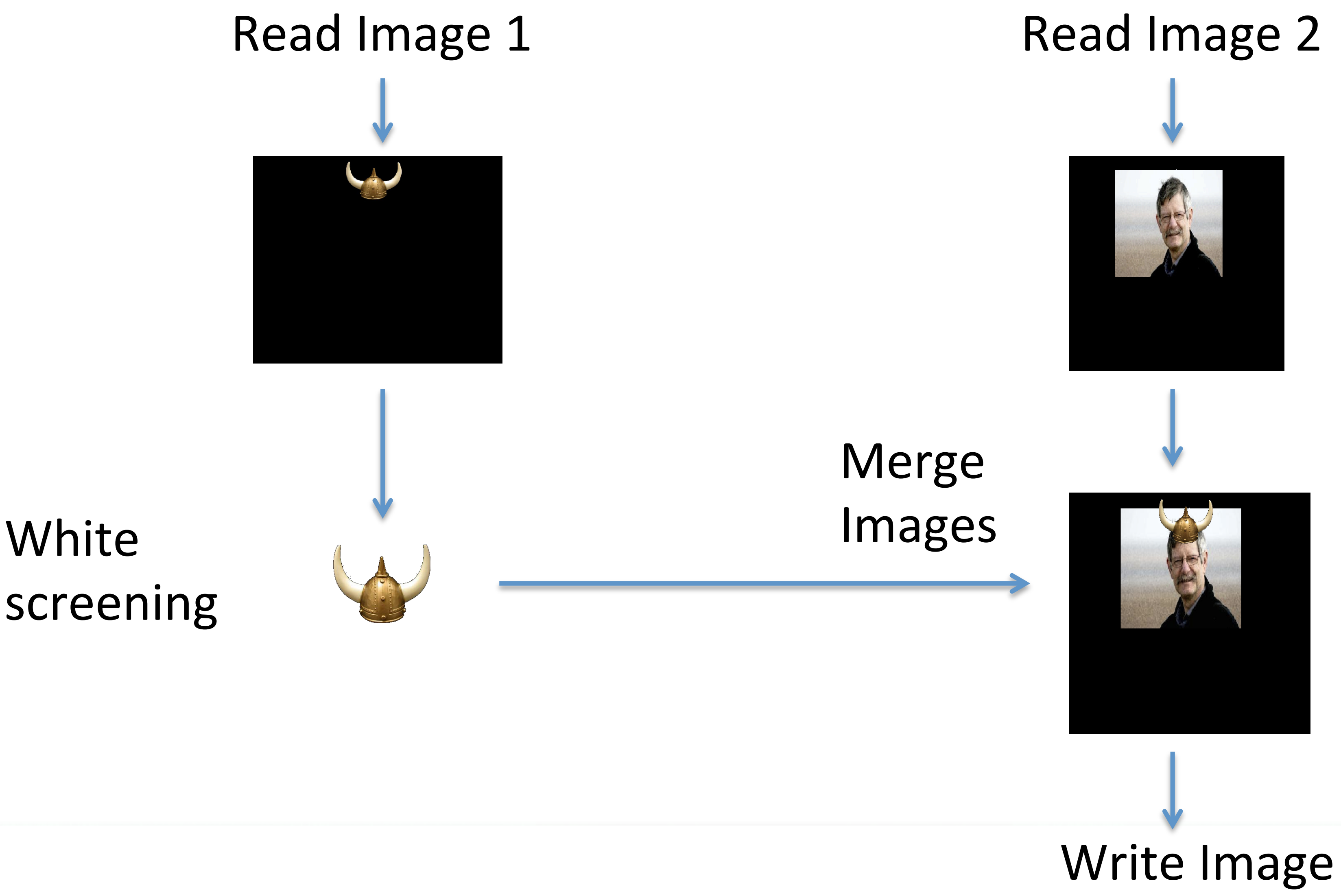


```
images = (char **) malloc (sizeof(char *)*NIMGS);
for (int i=0; i<NIMGS; i++) {
  images[i] = (char *) malloc (sizeof(char)*20);
  sprintf(images[i],"images/image%d.png", i);
}
for(int i = 0 ; i < NIMGS; i++)
{
    r1 = generate(images[i]);
    r2 = filter(r1);

}
  /*
StreamGen streamgen(NIMGS,images);

ff_pipeline pipe;
pipe.add_stage(&streamgen);
//pipe.add_stage(&global_farm);
```

Symbol 'r1' could not be resolved

Writable | Smart Insert | 323 : 32

Image Processing Example



Read Image 1

Read Image 2

White screening

Merge Images

Write Image

Basic Erlang Structure

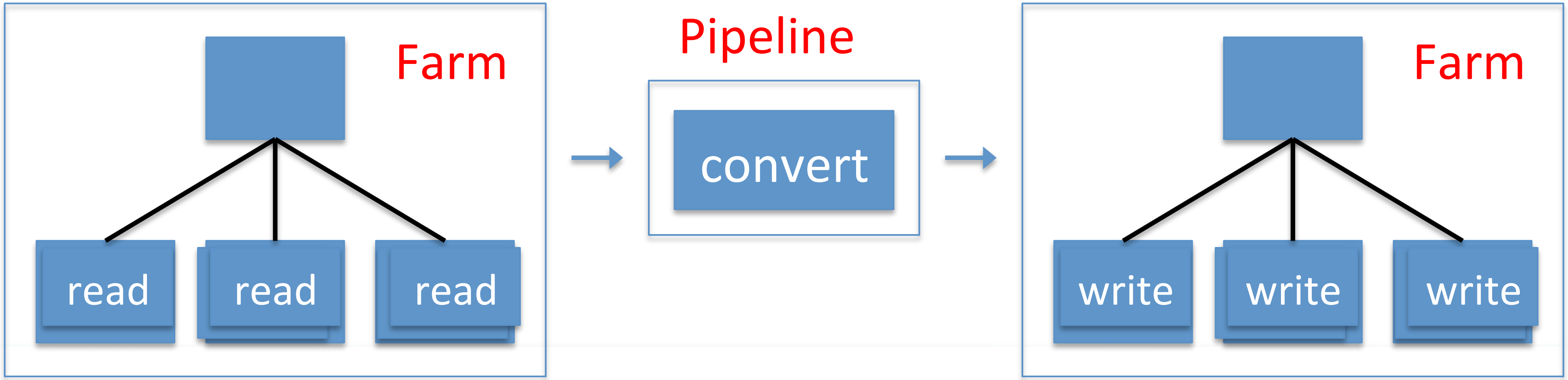
```
[ writeImage(convertMerge(readImage(X)))
                                    || X <- Images() ]

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

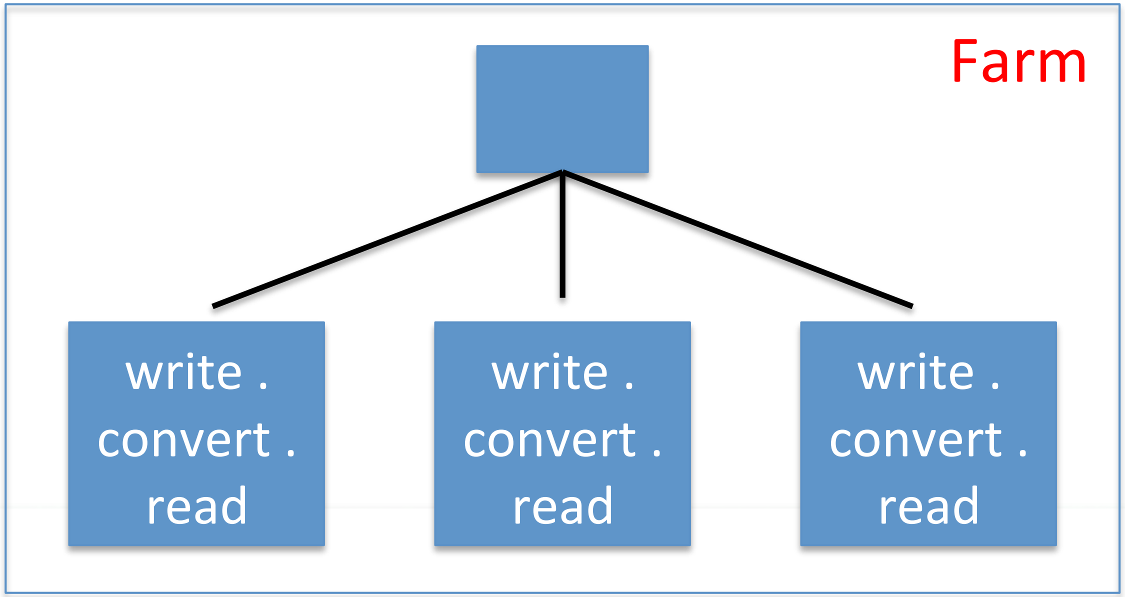


Alternative Program Structure

Sequential

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

Parallel



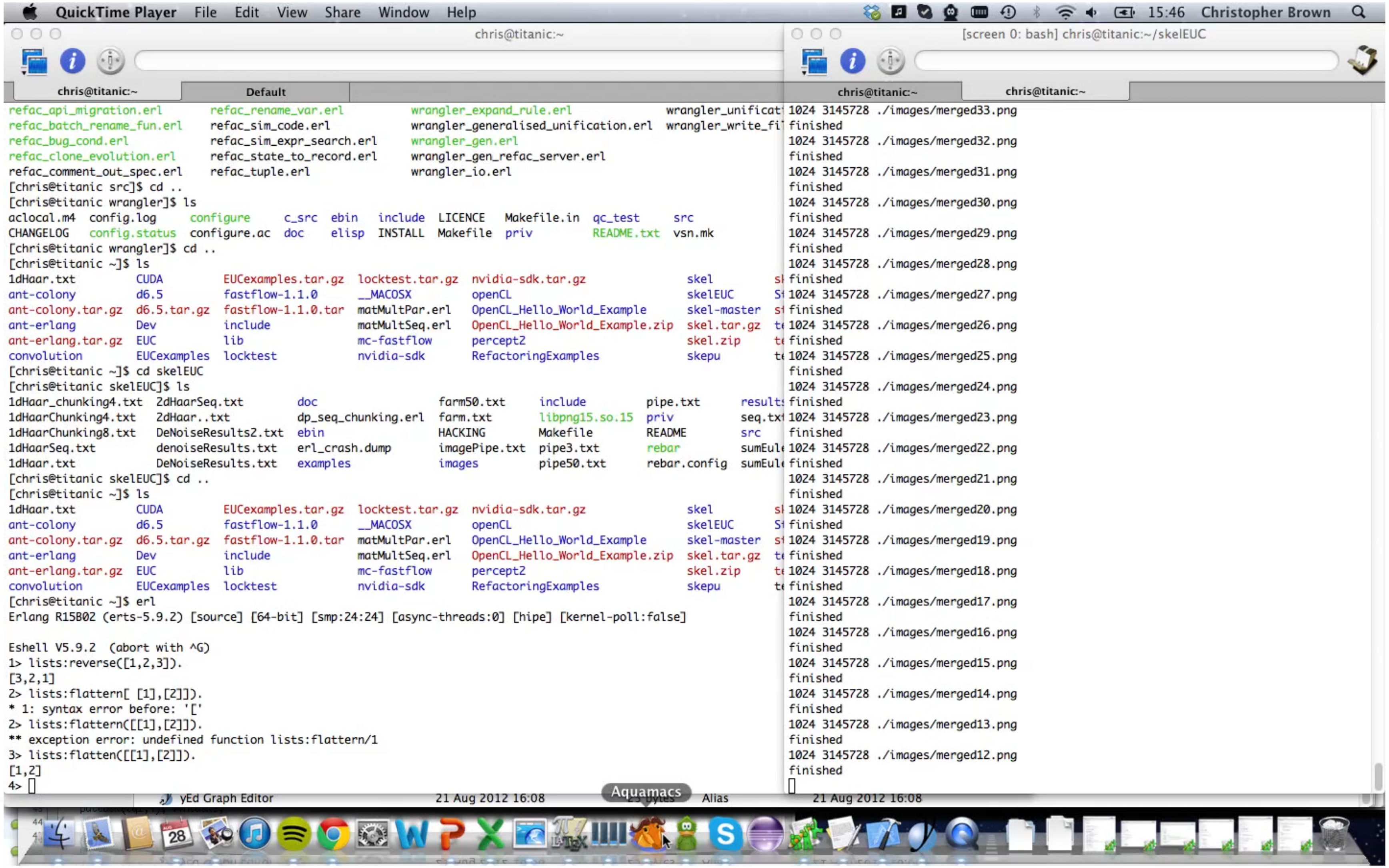
Farm

write .
convert .
read

write .
convert .
read

write .
convert .
read

Refactoring Demo

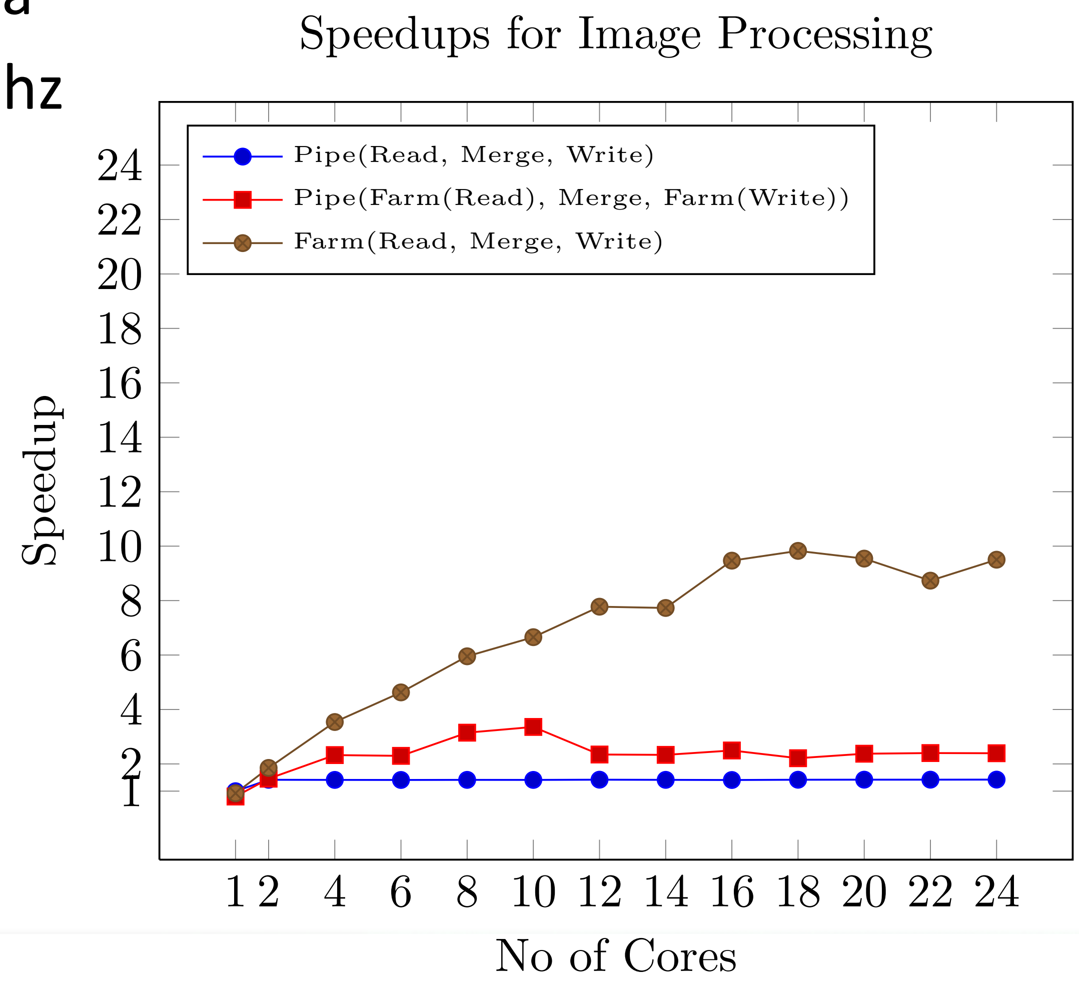


```
refac_api_migration.erl     refac_rename_var.erl         wrangler_expand_rule.erl              wrangler_unificat
refac_batch_rename_fun.erl  refac_sim_code.erl           wrangler_generalised_unification.erl  wrangler_write_fi
refac_bug_cond.erl          refac_sim_expr_search.erl    wrangler_gen.erl
refac_clone_evolution.erl   refac_state_to_record.erl    wrangler_gen_refac_server.erl
refac_comment_out_spec.erl  refac_tuple.erl              wrangler_io.erl
[chris@titanic src]$ cd ..
[chris@titanic wrangler]$ ls
aclocal.m4  config.log     configure     c_src  ebin   include  LICENCE   Makefile.in  qc_test     src
CHANGELOG   config.status  configure.ac  doc    elisp  INSTALL  Makefile  priv         README.txt  vsn.mk
[chris@titanic wrangler]$ cd ..
[chris@titanic ~]$ ls
1dHaar.txt          CUDA          EUCexamples.tar.gz  locktest.tar.gz  nvidia-sdk.tar.gz             skel         sk
ant-colony          d6.5          fastflow-1.1.0      __MACOSX         openCL                        skelEUC      Sk
ant-colony.tar.gz   d6.5.tar.gz   fastflow-1.1.0.tar  matMultPar.erl   OpenCL_Hello_World_Example    skel-master  sk
ant-erlang          Dev           include             matMultSeq.erl   OpenCL_Hello_World_Example.zip  skel.tar.gz  te
ant-erlang.tar.gz   EUC           lib                 mc-fastflow      percept2                      skel.zip     te
convolution         EUCexamples   locktest            nvidia-sdk       RefactoringExamples           skepu        te
[chris@titanic ~]$ cd skelEUC
[chris@titanic skelEUC]$ ls
1dHaar_chunking4.txt  2dHaarSeq.txt          doc                  farm50.txt     include          pipe.txt     results
1dHaarChunking4.txt   2dHaar..txt            dp_seq_chunking.erl  farm.txt       libpng15.so.15   priv         seq.txt
1dHaarChunking8.txt   DeNoiseResults2.txt    ebin                 HACKING        Makefile         README       src
1dHaarSeq.txt         denoiseResults.txt     erl_crash.dump       imagePipe.txt  pipe3.txt        rebar        sumEule
1dHaar.txt            DeNoiseResults.txt     examples             images         pipe50.txt       rebar.config sumEule
[chris@titanic skelEUC]$ cd ..
[chris@titanic ~]$ ls
1dHaar.txt          CUDA          EUCexamples.tar.gz  locktest.tar.gz  nvidia-sdk.tar.gz             skel         sk
ant-colony          d6.5          fastflow-1.1.0      __MACOSX         openCL                        skelEUC      Sk
ant-colony.tar.gz   d6.5.tar.gz   fastflow-1.1.0.tar  matMultPar.erl   OpenCL_Hello_World_Example    skel-master  sk
ant-erlang          Dev           include             matMultSeq.erl   OpenCL_Hello_World_Example.zip  skel.tar.gz  te
ant-erlang.tar.gz   EUC           lib                 mc-fastflow      percept2                      skel.zip     te
convolution         EUCexamples   locktest            nvidia-sdk       RefactoringExamples           skepu        te
[chris@titanic ~]$ erl
Erlang R15B02 (erts-5.9.2) [source] [64-bit] [smp:24:24] [async-threads:0] [hipe] [kernel-poll:false]

Eshell V5.9.2  (abort with ^G)
1> lists:reverse([1,2,3]).
[3,2,1]
2> lists:flattern[ [1],[2]]).
* 1: syntax error before: '['
2> lists:flattern([[1],[2]]).
** exception error: undefined function lists:flattern/1
3> lists:flatten([[1],[2]]).
[1,2]
4> 
```

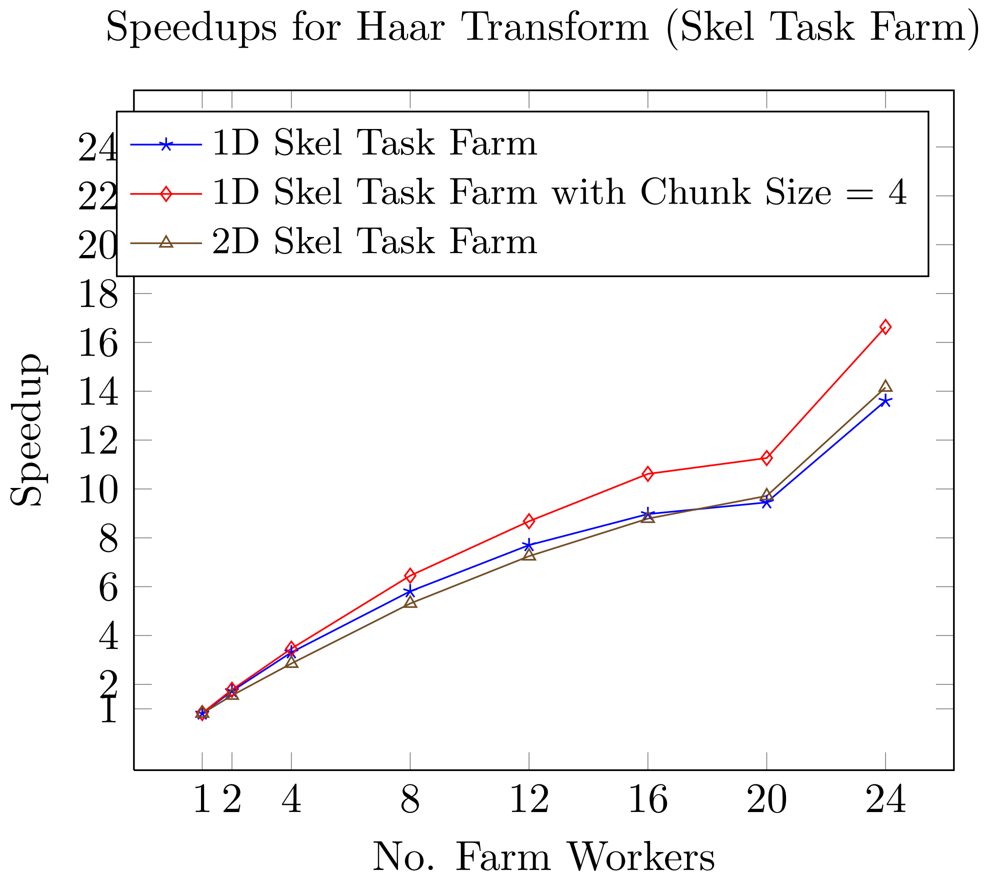
```
1024 3145728 ./images/merged33.png
finished
1024 3145728 ./images/merged32.png
finished
1024 3145728 ./images/merged31.png
finished
1024 3145728 ./images/merged30.png
finished
1024 3145728 ./images/merged29.png
finished
1024 3145728 ./images/merged28.png
finished
1024 3145728 ./images/merged27.png
finished
1024 3145728 ./images/merged26.png
finished
1024 3145728 ./images/merged25.png
finished
1024 3145728 ./images/merged24.png
finished
1024 3145728 ./images/merged23.png
finished
1024 3145728 ./images/merged22.png
finished
1024 3145728 ./images/merged21.png
finished
1024 3145728 ./images/merged20.png
finished
1024 3145728 ./images/merged19.png
finished
1024 3145728 ./images/merged18.png
finished
1024 3145728 ./images/merged17.png
finished
1024 3145728 ./images/merged16.png
finished
1024 3145728 ./images/merged15.png
finished
1024 3145728 ./images/merged14.png
finished
1024 3145728 ./images/merged13.png
finished
1024 3145728 ./images/merged12.png
finished
```

Speedup Results

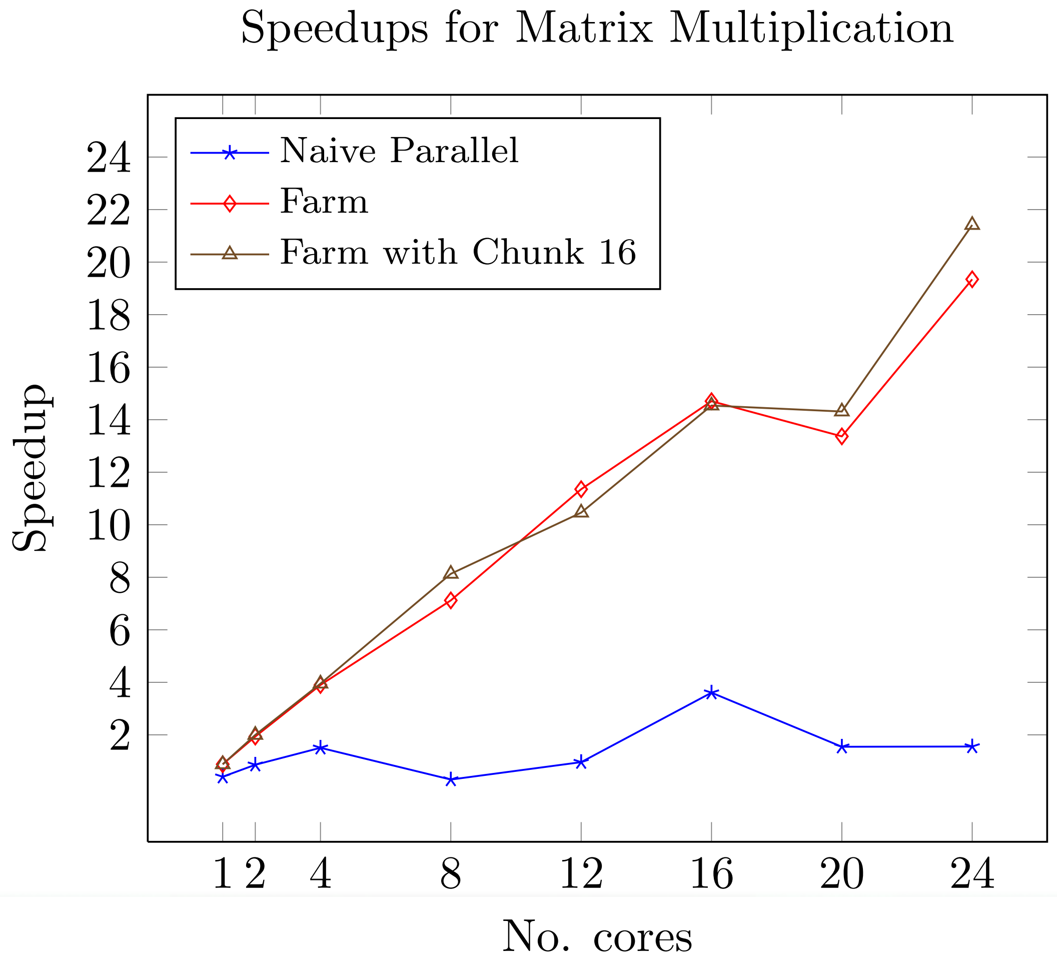
- 24 core machine at Uni. Pisa
- AMD Opteron 6176. 800 Mhz
- 32GB RAM



Speedup Results (Image Processing)

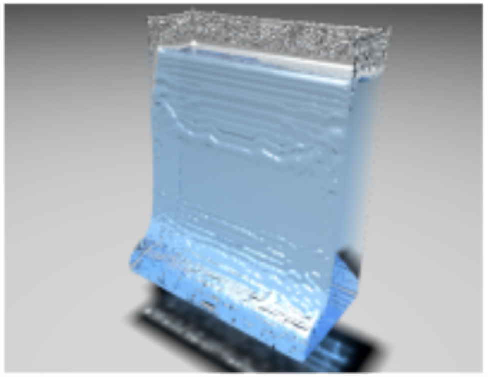


Using The Right Pattern Matters



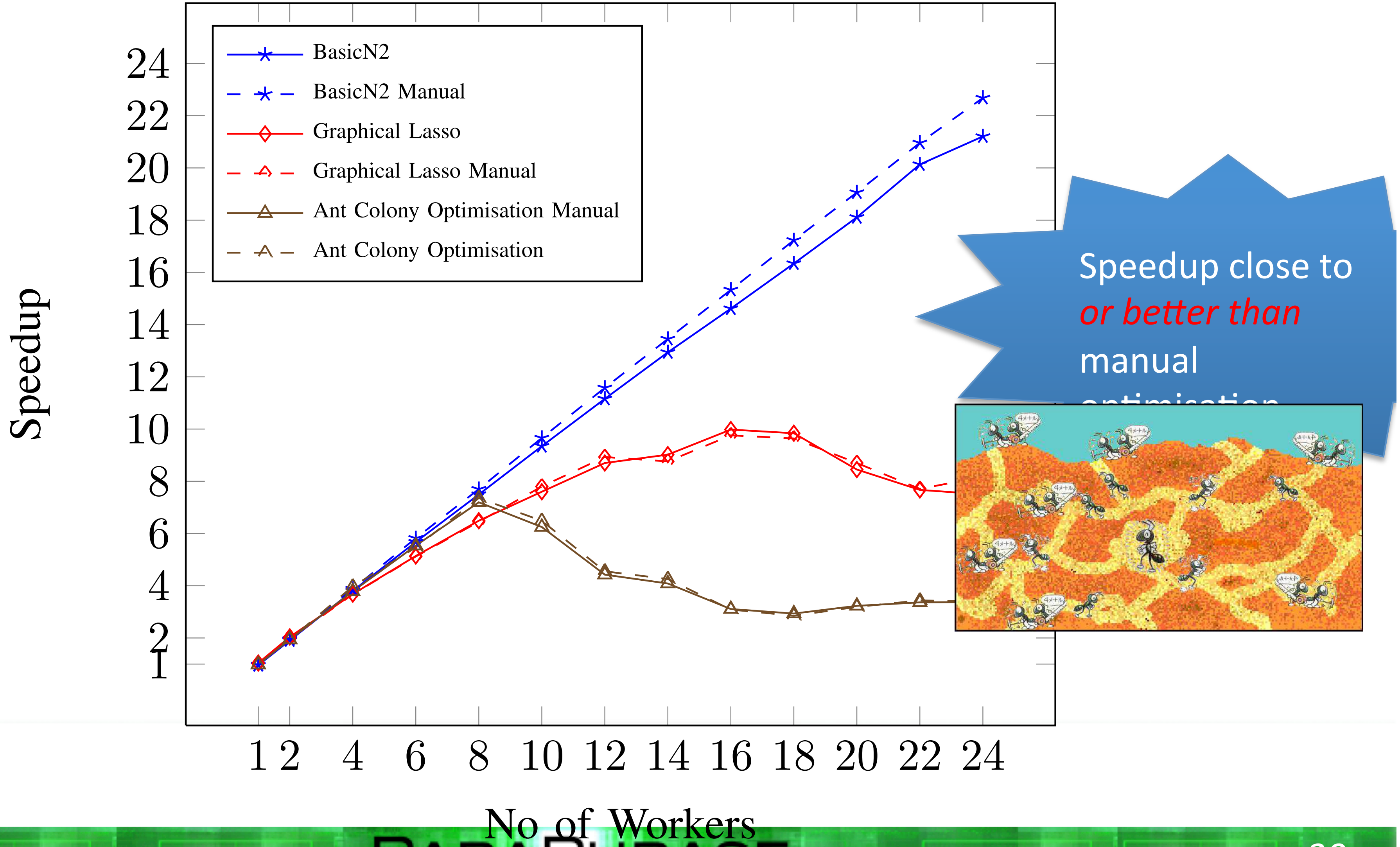
Large-Scale Demonstrator Applications

- 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)



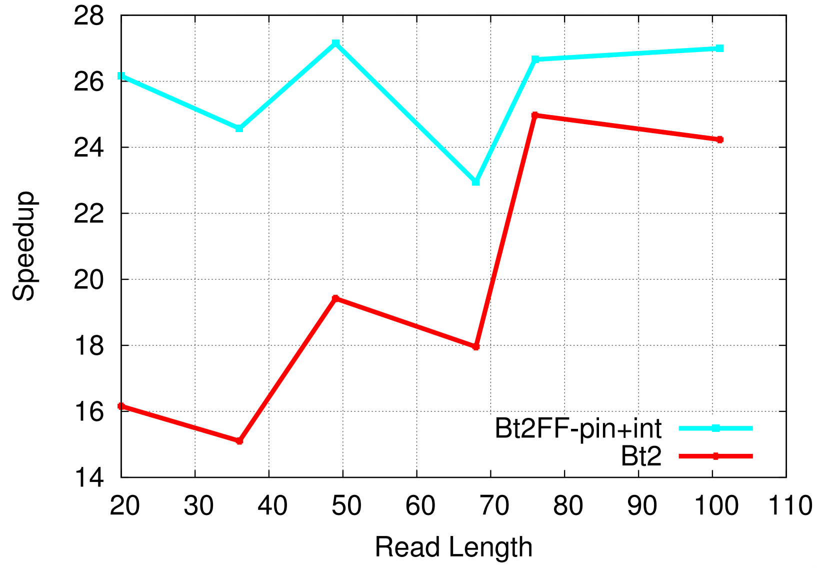
Speedup Results (demonstrators)

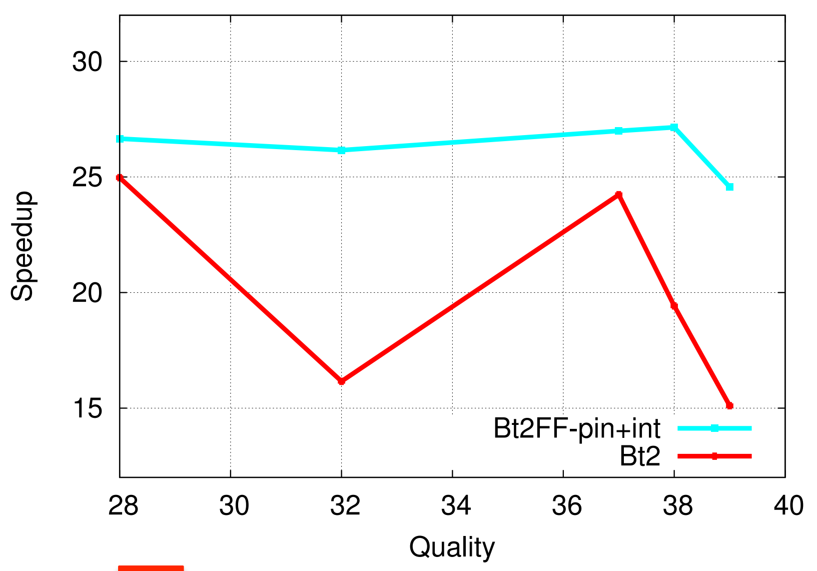
Speedups for Ant Colony, BasicN2 and Graphical Lasso



Speedup close to *or better than* manual optimisation

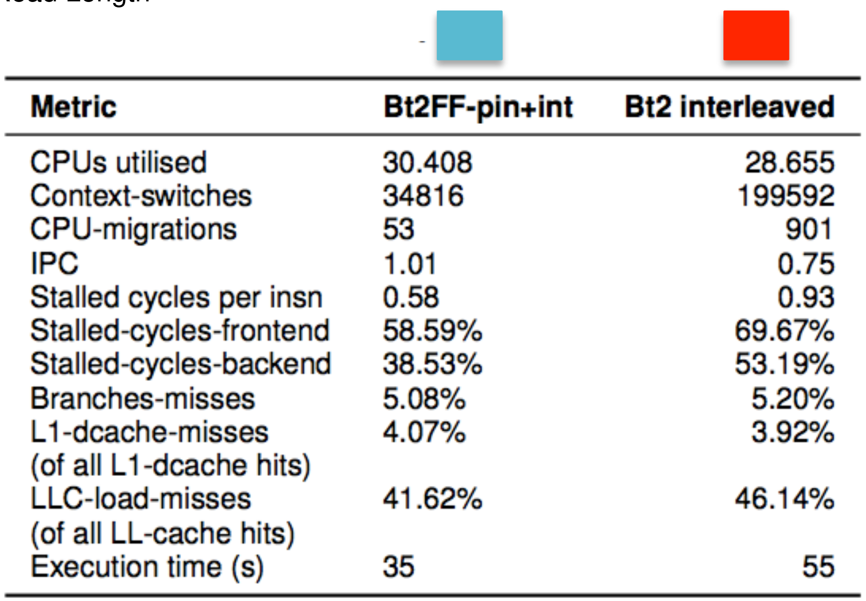
Bowtie2: most widely used DNA alignment tool





🟥 Original

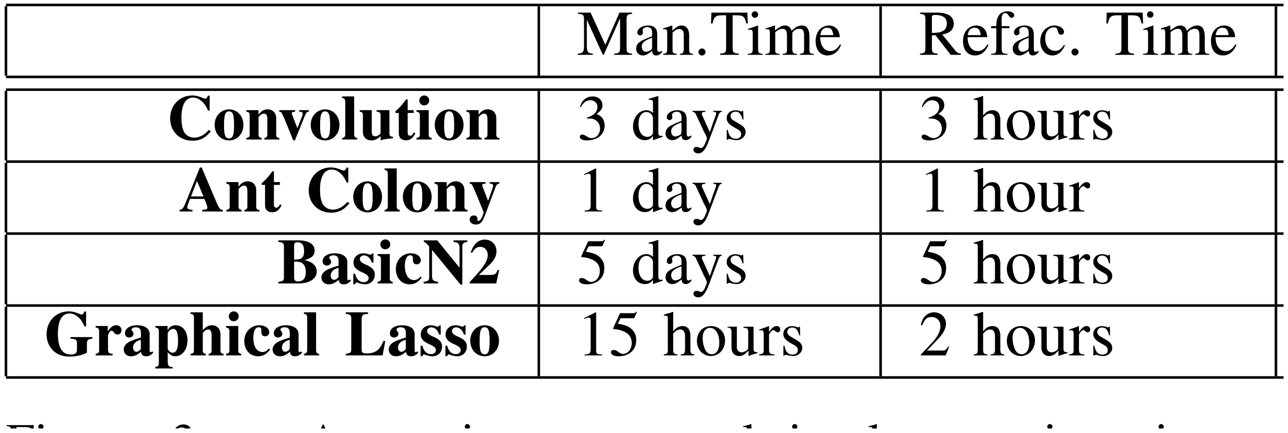
🟦 Paraphrase



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.

Comparison of Development Times



	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

Conclusions

- The manycore revolution is upon us
  - Computer hardware is changing very rapidly (more than in the last 50 years)
  - The **megacore** era is here (aka exascale, BIG data)
- Heterogeneity and energy are both important
- Most programming models are too low-level
  - concurrency based
  - need to expose mass parallelism
- Patterns and *functional programming* help with abstraction
  - millions of threads, easily controlled
- Refactoring helps with program structure

Isn’t this all just wishful thinking?

Rampant-Lambda-Men in St Andrews

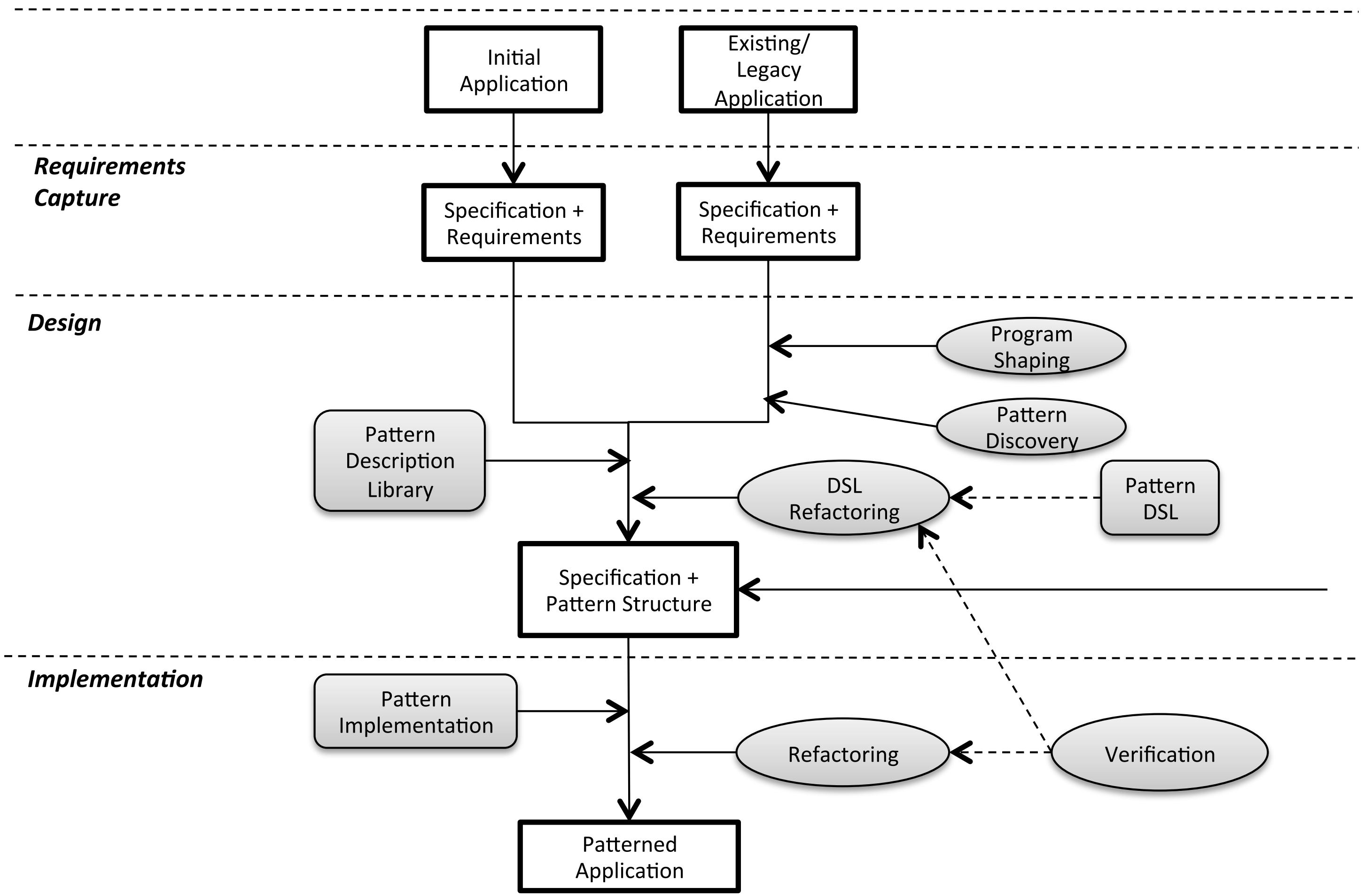
NO!

- C++11/14 has lambda functions; C++17 will have more
- Java 8 will have lambda (closures)
- Apple uses closures in Swift

Research Challenges

- How do we move software engineering into the manycore era
  - requirements, debugging, testing/verification, development methodologies, legacy codes, etc.
- Can we model parallelism formally
  - when is one program "better" than other
  - Can we prove this???
- How do we deal with the "megacore challenge"
  - scaling, heterogeneity, multiple levels
- What are the best abstractions for parallelism
  - skeletons (what skeletons?), evaluation strategies, ...
  - How do we help the programmer "think parallel"
  - What do we do if a pattern doesn't quite fit the problem
- How do we understand performance
  - visualisation, abstraction, formal reasoning, ...
- How can we analyse resource usage in parallel systems
  - Time, energy, ...
- What about tool support (e.g. refactoring)
- Can we do it all automatically??

Towards a general SE Methodology



…

Towards a general SE Methodology (2)

Further Reading

Chris Brown. Vladimir Janjic, Kevin Hammond, Mehdi Goli and John McCall
“Bridging the Divide: Intelligent Mapping for the Heterogeneous Parallel Programmer”, *Submitted to IPDPS 2015*

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

Chris B wn. Han Wolfgang Loidl and Kevin Hammond
llel Haskell Programs using Novel Refactoring Techniques”
Programming (TFP), Madrid, Spain, May 2011

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

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

Funded by

- **RePhrase (Horizon 2020), Software Engineering for Parallelism,**
  €3.7M, 2015-2018
- **ParaPhrase (EU FP7), Patterns for heterogeneous multicore,**
  €4.2M, 2011-2015
- **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
- **Islay (EPSRC), Real-time FPGA streaming implementation**
  £1.4M, 2008-2011

Some of our Industrial Connections

IBM

Software Competence Centre, Hagenberg

Erlang Solutions Ltd

Mellanox Inc.

SAP GmbH, Karlsrühe

BAe Systems

Selex Galileo

Philips Healthcare

Microsoft Research

Well-Typed LLC

IBM

Mellanox TECHNOLOGIES

PHILIPS

Erlang SOLUTIONS

SAP®

s c c h software competence center hagenberg

Roke Part of the Chemring Group

SELEX GALILEO A Finmeccanica Company

Well-Typed The Haskell Consultants

PRQA Programming Research

Microsoft Research

BAE SYSTEMS

ParaPhrase Needs You!

University of St Andrews

- 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.paraphrase-ict.eu`

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

@paraphrase_fp7