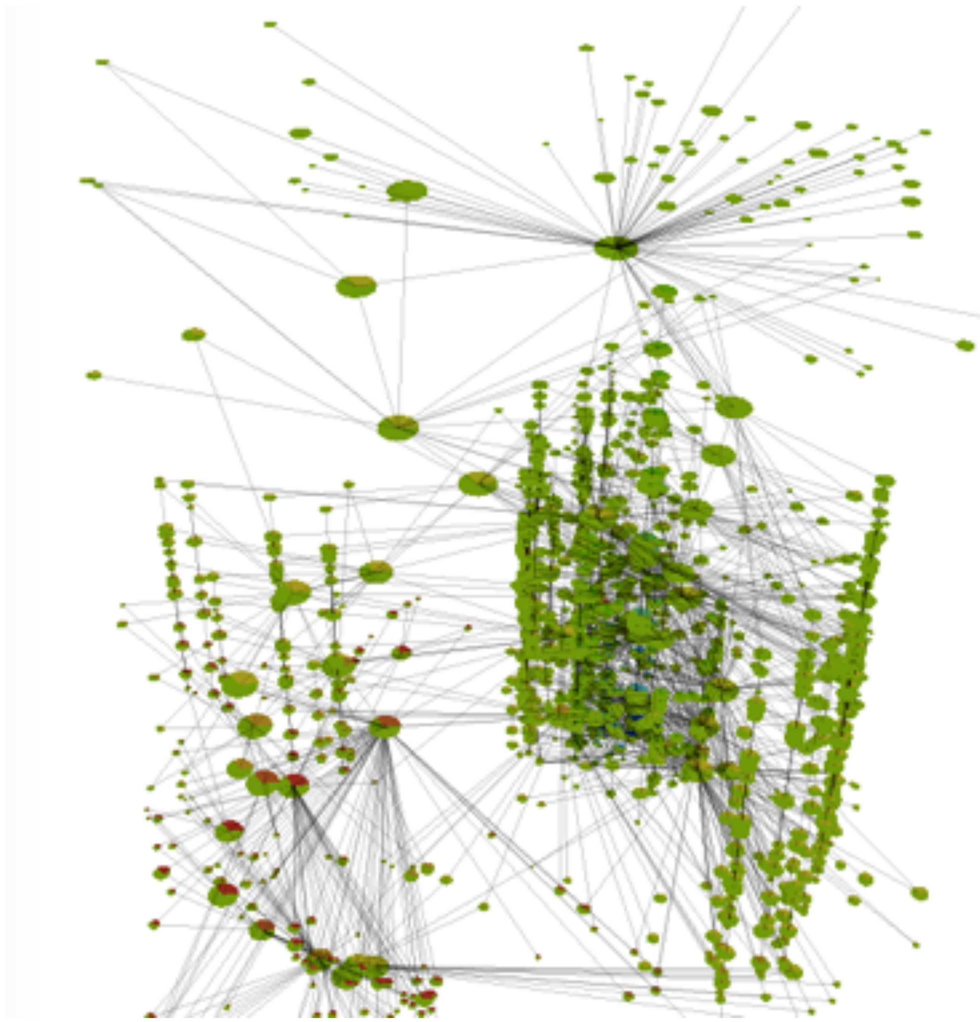
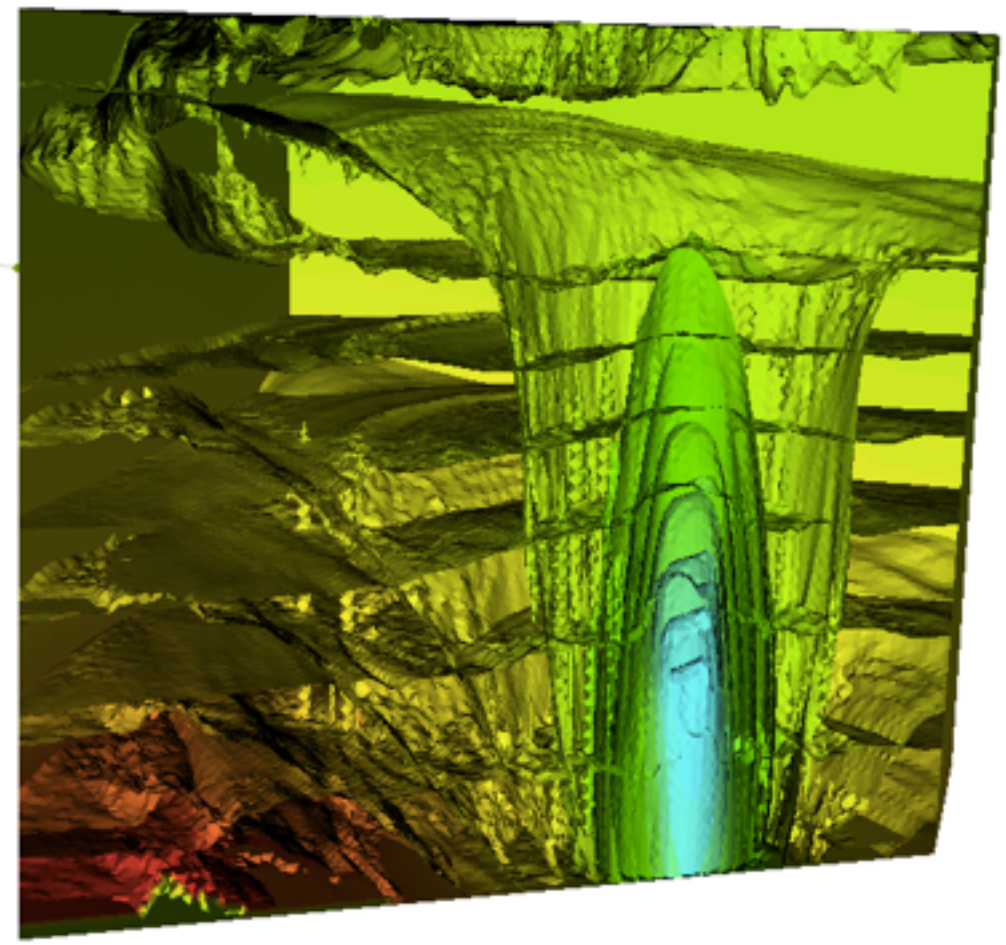
Skeletons for Parallel Scientific Computing





David Duke

Computational Science and Engineering Group
School of Computing
University of Leeds, UK

The purpose of computing is insight, not numbers.

R. Hamming, *Numerical Methods for Scientists and Engineers*, 1962.

Outline

1. Background
   - Computational science & visualization
   - Topological analysis
   - Running case study: the Joint Contour Net (JCN) and its implementation
2. Sequential implementation
3. Skeletons

---

4. Shared-memory parallelism in the Par Monad
5. Distributed parallelism
   - Eden
   - Distributed skeletons
   - Performance
   - Distributed data representation
6. Insight and conclusions
   - Strengths and weaknesses of skeletons
   - Challenges and opportunities in functional HPC

Computational science

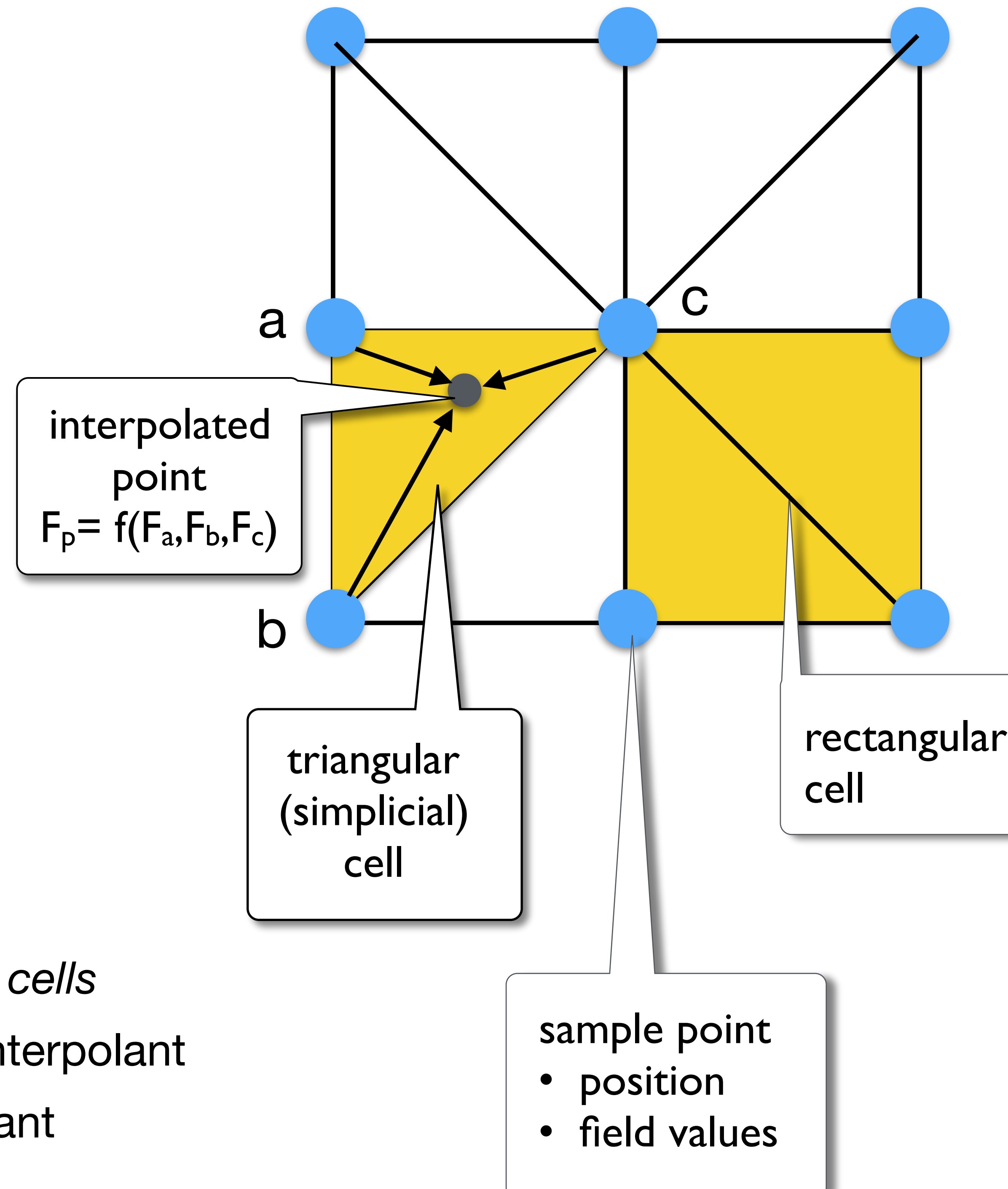
- Simulation/Sensors ⇛ {dataset} ⇛ Visualization/Analysis
- Characteristics
  - volumetric (3D) time-varying data, or hyper-volumes
  - multifields: $f :: M \subset R^{4+} \rightarrow R^n$
- Example problems:
  - nuclear scission *[LBL/FHPC14]:* 66×40×40, 2 fields [× 40]
  - star formation *[ICFP'08]:* 250×250×600, 13 fields [× 200]
  - hurricane Isabel simulation: 500×500×100, 13 fields [× 48]
  - combustion: Landge et.al., SC14: - 2025×1600×400 [× ?]
  - high-dimensional spaces: Gerber et.al. 700K samples in 10D
  - HIV capsid: Zhao et.al. (UIUC) - 64M atoms
  - ...
- Challenge - scale

Zhao et.al., Nature, 497, 2013

NCSA Blue Waters
Cray XE/XK
10 Petaflops, 300K cores, $200M

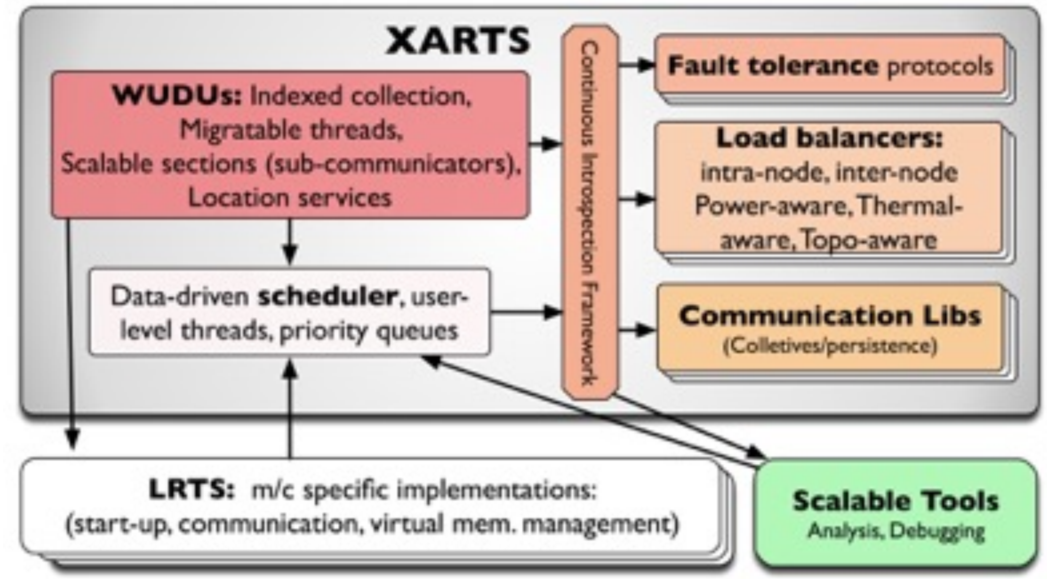
Scientific datasets

- What is a scientific dataset?
  - (discrete) set of samples ...
    - scalar data - pressure, temperature
    - vector data - velocity, curl
    - tensor data - diffusion
  - .. at points in some continuous space
    - 3D
    - 3D + time
    - higher dimensions - e.g. in optimization
- Sample points may be
  - organised on a regular grid
  - irregularly spaced
- But we also need to interpolate
  - so we divide the dataset into topological *cells*
  - simplicial (triangles/tets) -> barycentric interpolant
  - square/rectangular -> bi/trilinear interpolant
  - other subdivisions and interpolants used

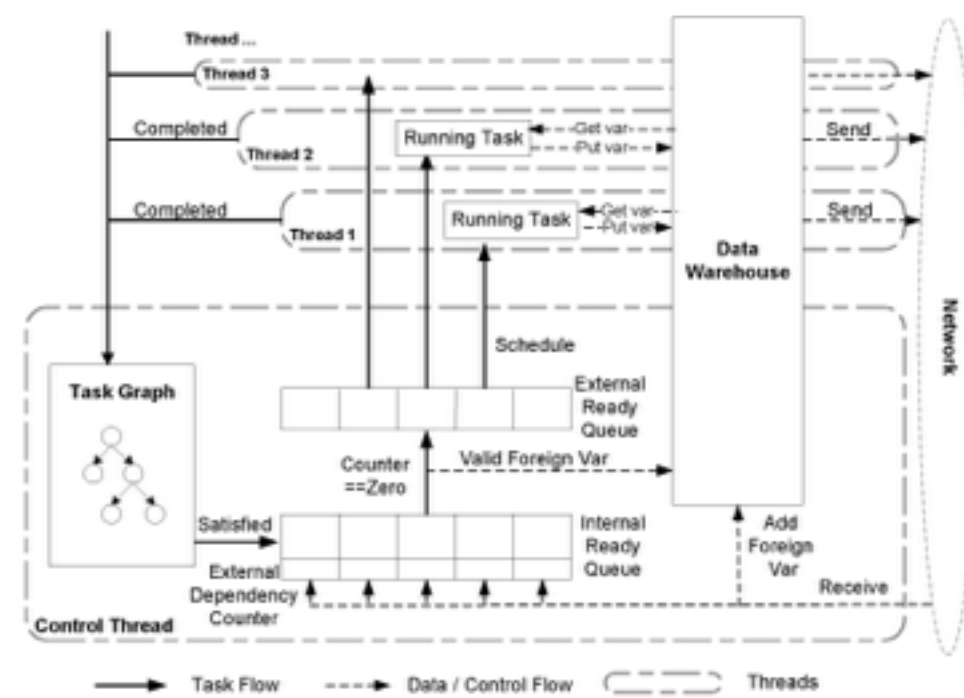


Implementation

- Parallel Run-Time Systems
  - processor virtualisation
  - migratable work and data
  - asynchrony, reflection
  - fault tolerance
- Domain-Specific Languages
- Example of platforms
  - Uintah (SCI - Utah)
  - Charm++ (UIUC)
  - VTK (Kitware - visualization)
- Imperative technologies
- Imperative thinking

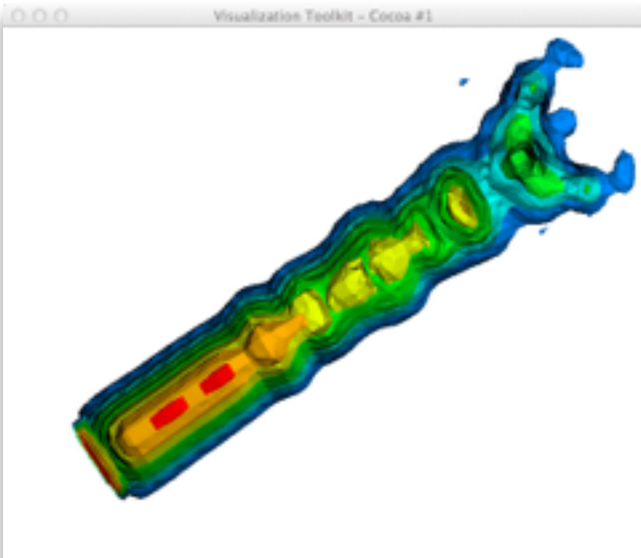


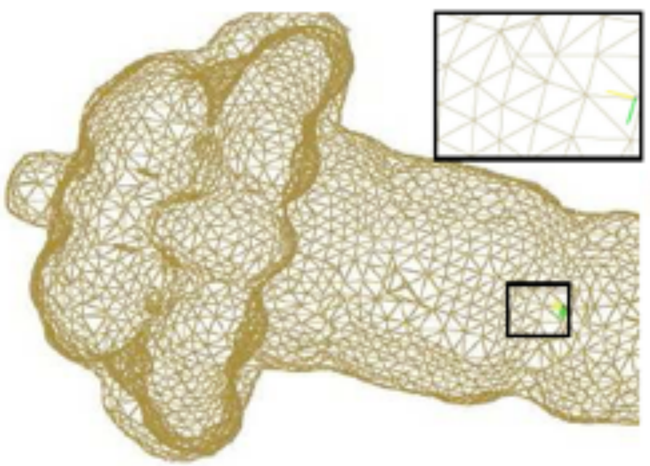
Getting Ready for Adaptive RTSs, Laxmikant Kale, Salesian conference on High-speed Computing, LANL, 2014.



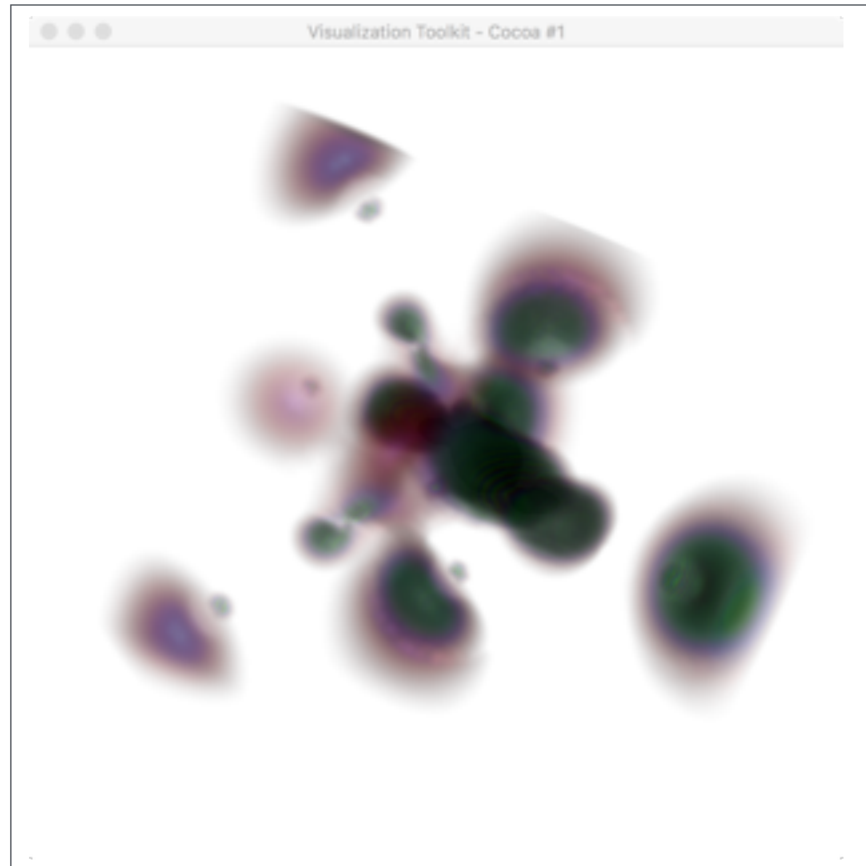
The Uintah Framework: A Unified Heterogeneous Task Scheduling and Runtime System: Meng, Humphry, Berzins, Proc. Supercomputing, 2012

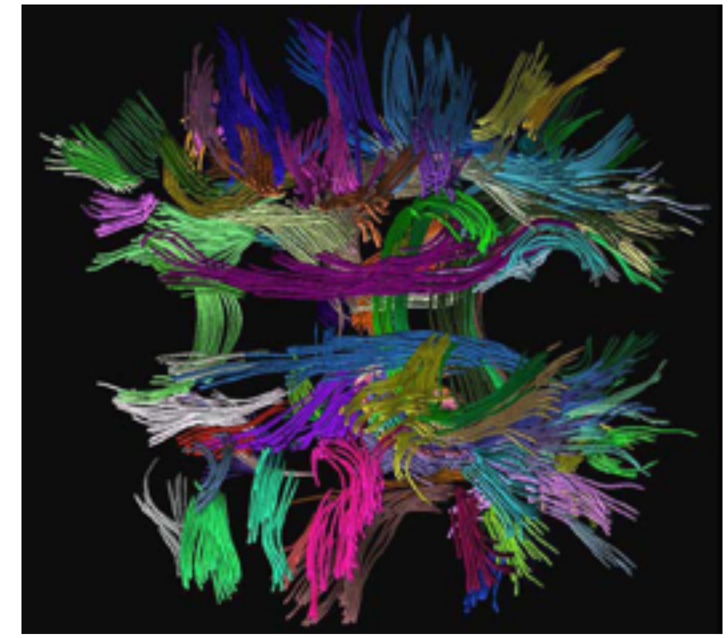
Visualization Examples



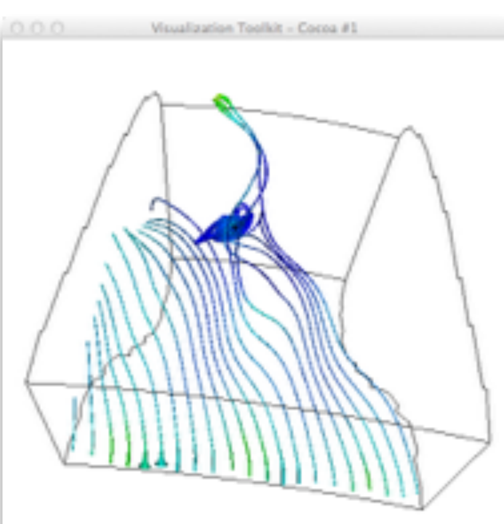


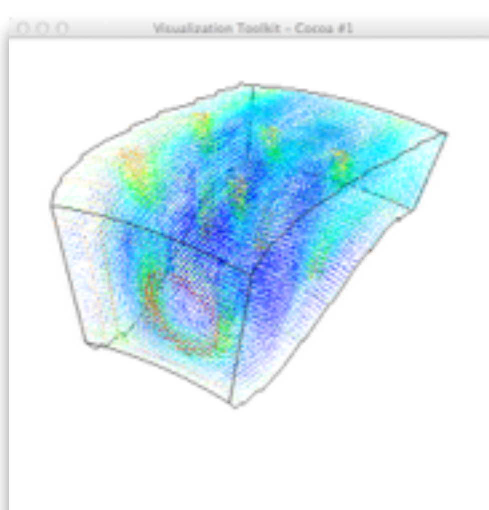
Topologically Accurate Dual Isosurfacing Using Ray Intersection: Jaya Sreevalsan-Nair, Lars Linsen, and Bernd Hamann

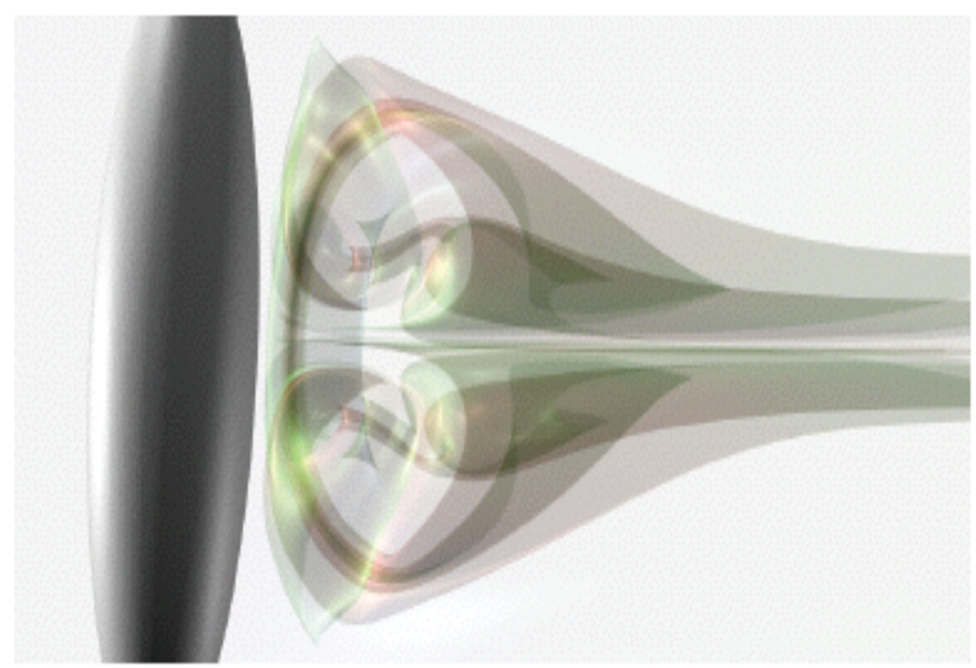




DTI Fiber Clustering in the Whole Brain, S. Zhang & D. Laidlaw, Proc. IEEE Visualization 2004.







Generation of Accurate Integral Surfaces in Time-Dependent Vector Fields, C. Garth, H. Krishnan, X. Tricoche, T. Bobach, and K.I. Joy

Challenges

- Data scale:
  - 10^{12} bytes (tera-scale), 10^{15} (peta-scale), 10^{18} (exa-scale)
  - "discovery science"
  - push for ever-greater spatial and temporal resolution
- Visualization limits:
  - 10^7 retinal cells,
  - 1 byte/cell (RGB), 300MB/s raw b/with
  - perception: effective bandwidth 10-100KB/s
  - display: 4×10^6 pixels on a good single-screen display
- Geometric & topological analysis
  - identify features
  - guide/accelerate visualization
  - quantitative summary
  - *multifields*

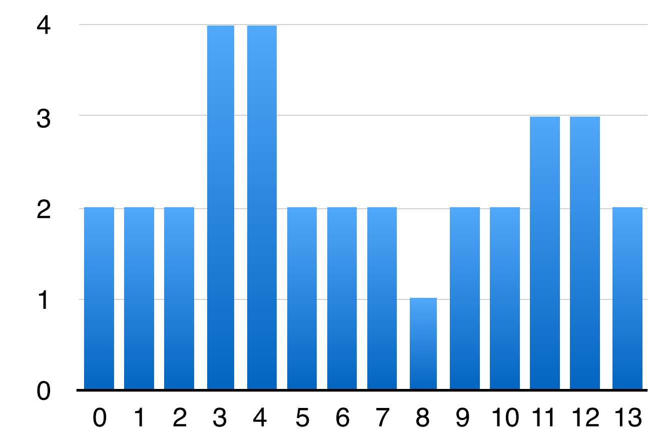
How long to eyeball
- 1 GB?
- 1 TB?
- 1 EB?

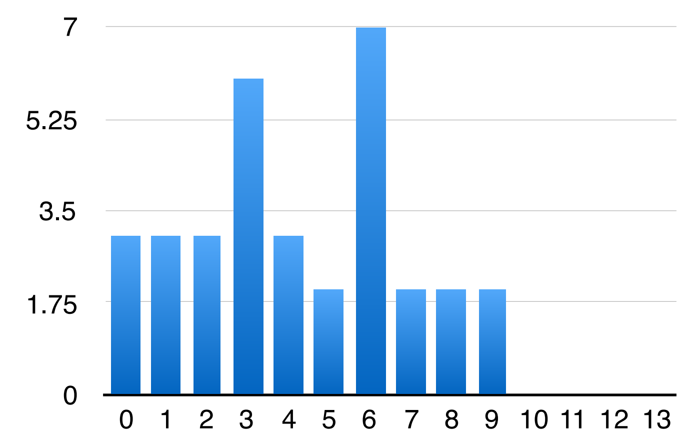
2.7 hrs
112 days
317000 years

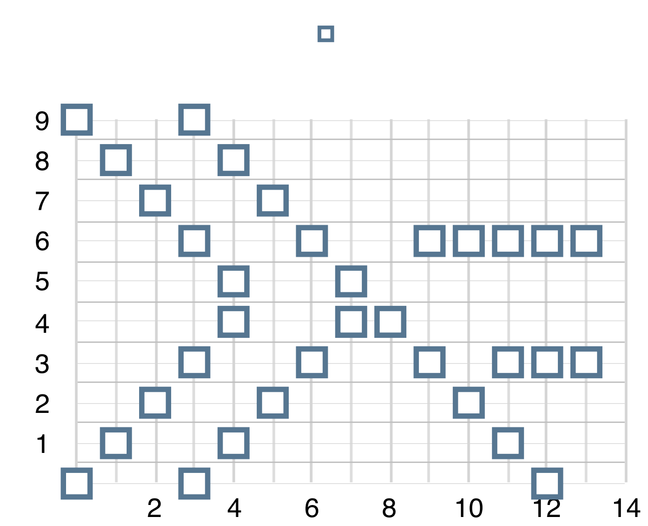
One answer: abstraction

UNIVERSITY OF LEEDS

- What if we have too many raw numbers?
  - (0,0) (0,9) (1,8) (1,1) (2,2) (2,7) (3,0) (3,3) (3,6) (3,9) (4,1) (4,4) (4,5) (4,8) (5,2) (5,7) (6,3) (6,6) (7,4) (7,5) (8,4) (9,3) (9,6) (10,2) (10,6) (11,1) (11,3) (11,6) (12,0) (12,3) (12,6) (13,3) (13,6)
- Need visual summaries!
- Simple example: histograms
  - *quantise* value range into set of discrete buckets
  - visualize the buckets
- As always, choice of visualization is critical

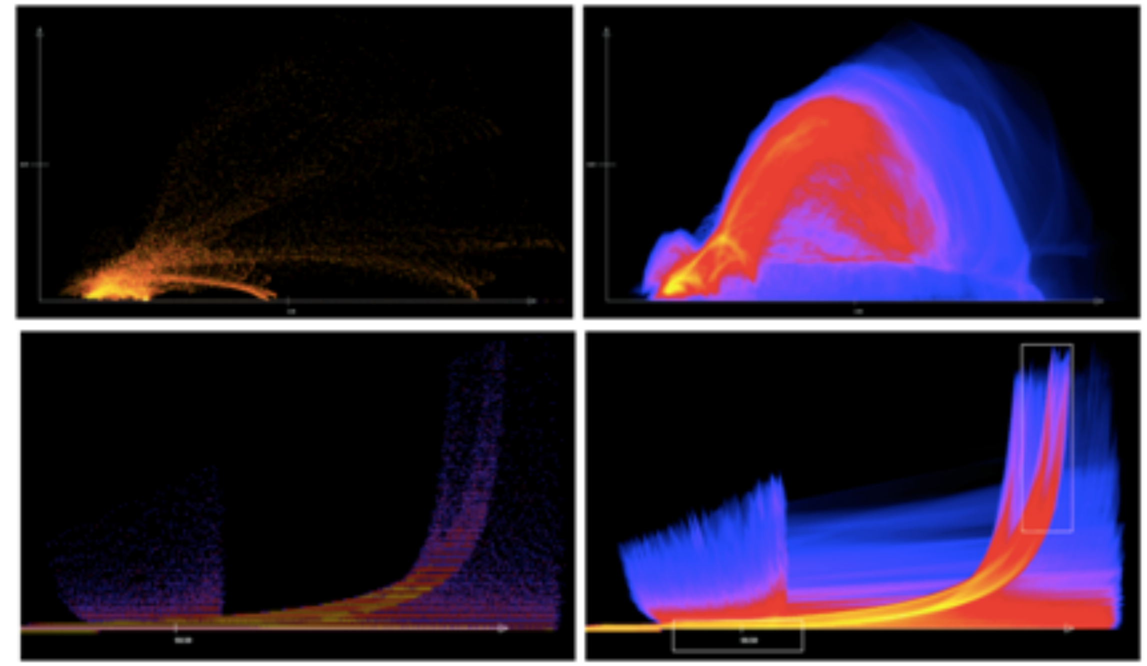






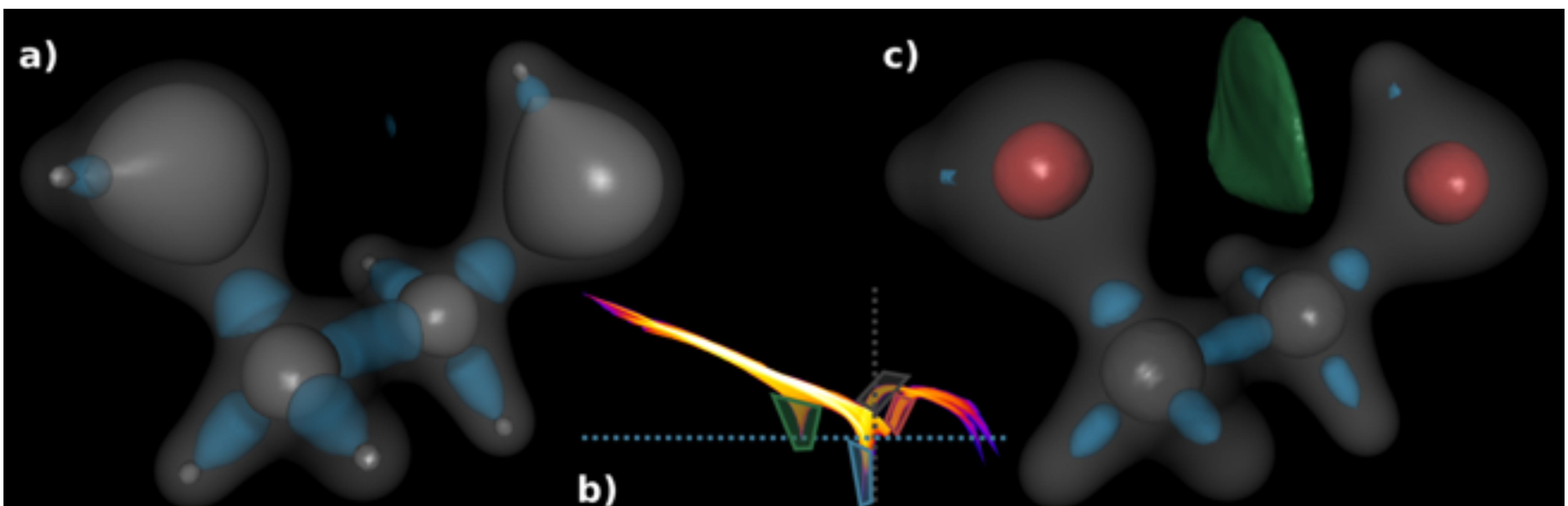
Continuous scatterplots

- Scientific data are discrete samples of a continuous field
- Discrete scatterplots miss important detail: the "bits between the samples"!
- Useful! But the result is an image, not an abstraction.



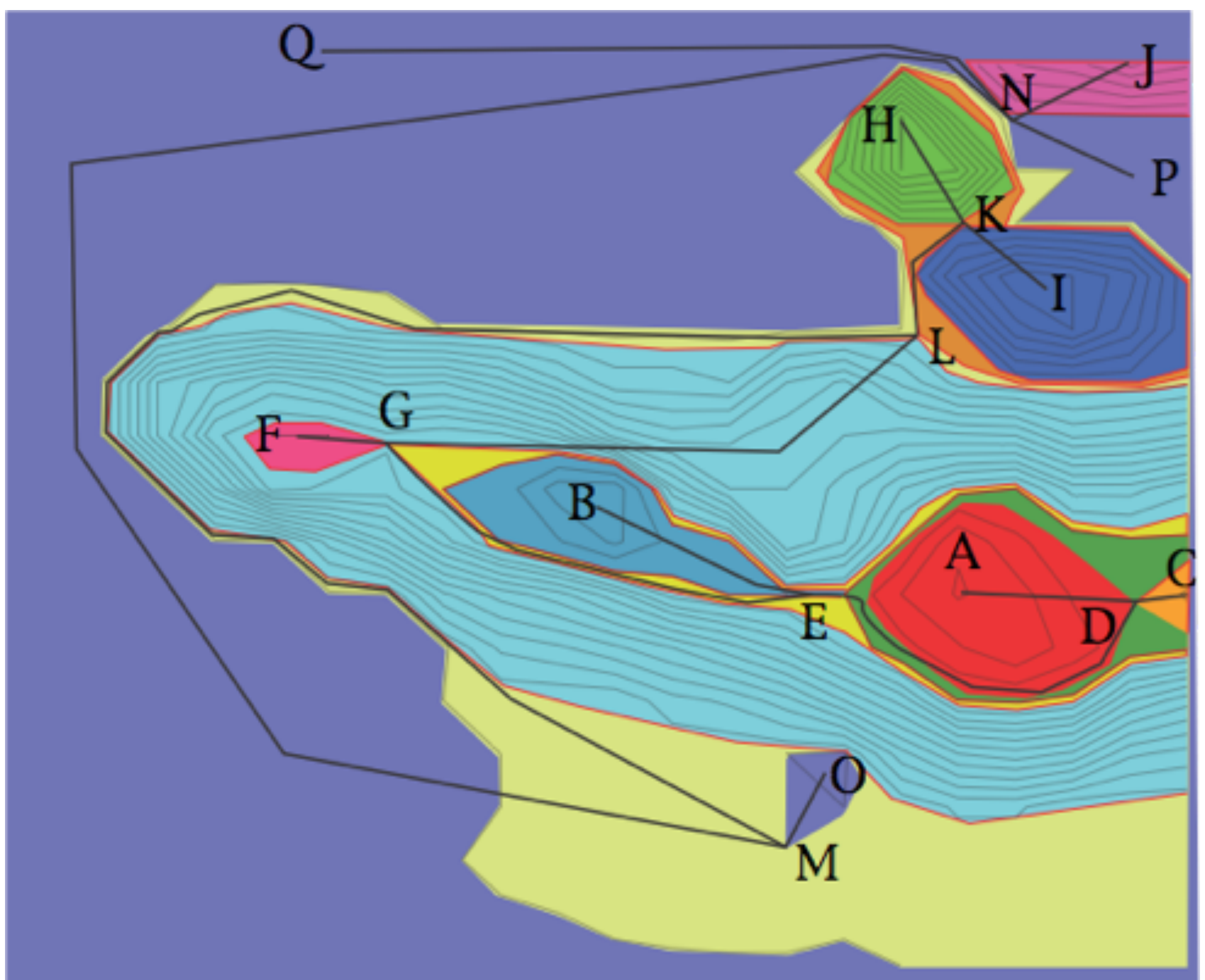
Bachthaler & Weiskopf, Continuous Scatterplots, IEEE TVCG 14(6), 2008

Carr, Geng, Tierny, Chattopadhyay & Knoll, Fiber Surfaces: Generalizing Isosurfaces to Bivariate Data, Eurovis, 2015.

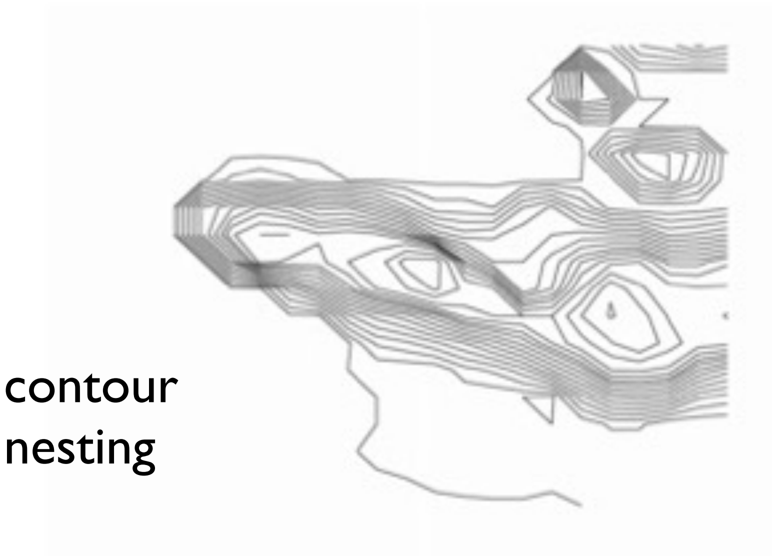


Introducing scalar topology

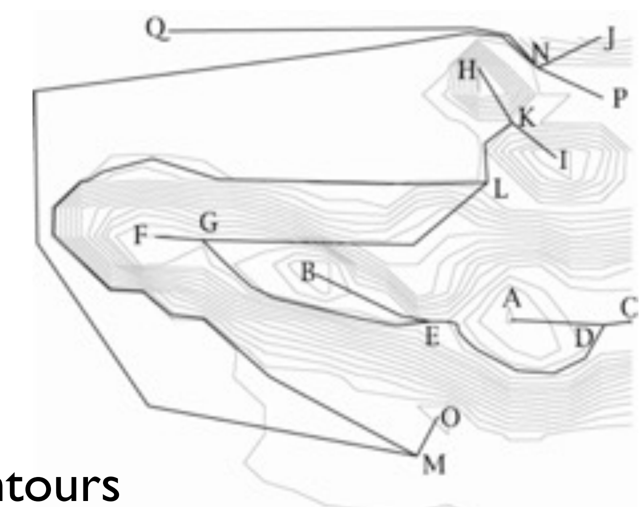
- Rubber-sheet geometry
- Captures properties unchanged under
  - continuous deformation
  - affine transformation



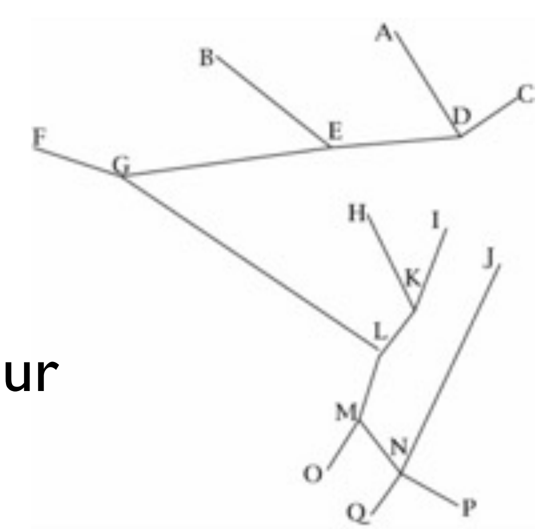
Example courtesy of Hamish Carr



contour nesting



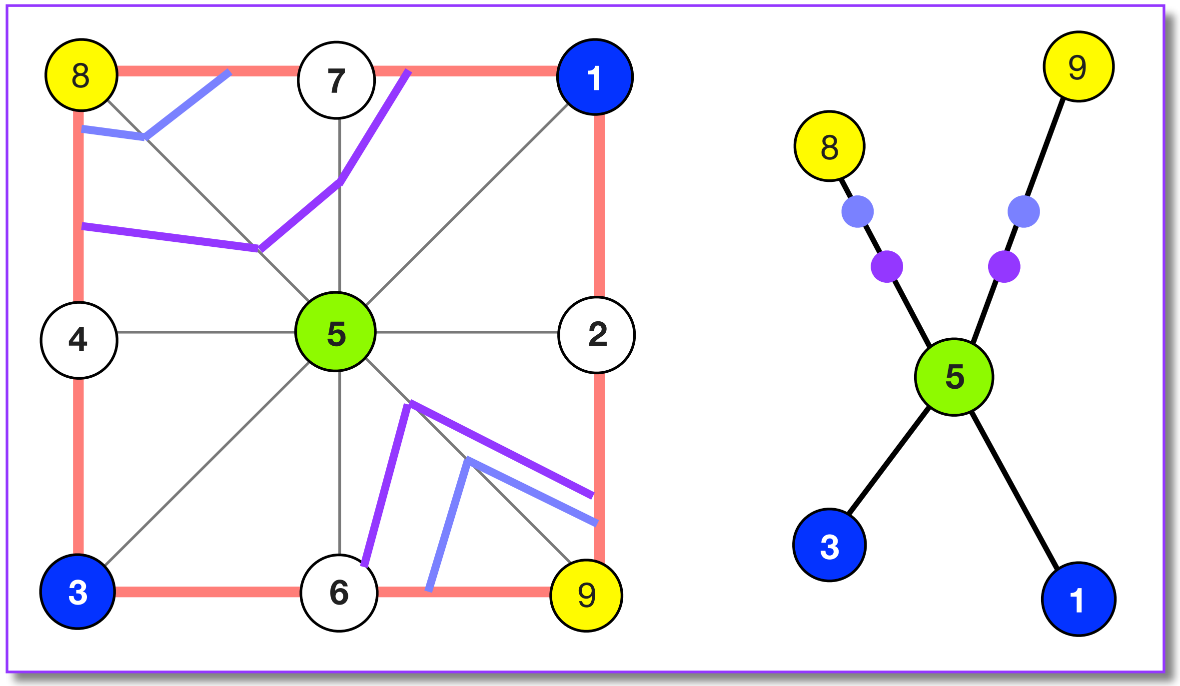
contours with paths

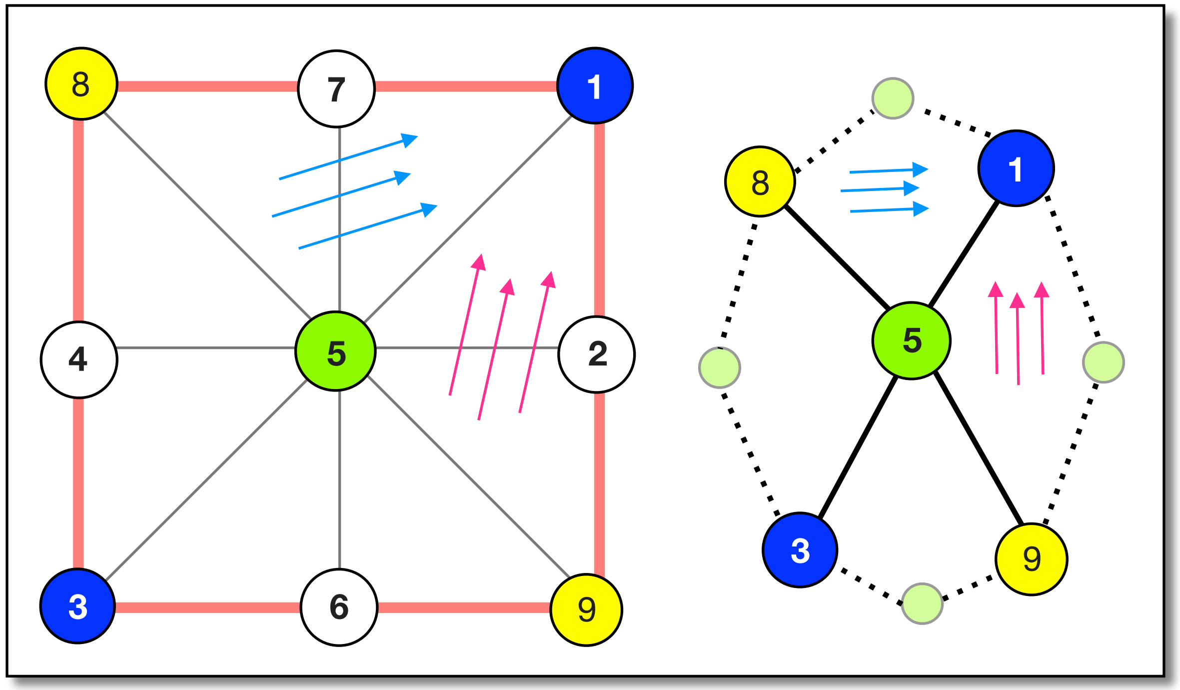


contour tree

More formally ...

- Assumptions
  - scalar field f : $M \subset R^m \rightarrow R$
  - samples at discrete points in bounded domain
  - subdivided into discrete [simplicial] cells
- Vocabulary for "interesting things" in space
  - local / global minima & maxima
  - saddles
- Segment domain into *regions* of equivalence
- **Reeb graph** (contour tree):
  - regions inside and outside a contour
  - contour connectivity
  - captures nesting of contours
- **Morse-Smale complex**:
  - watershed and inverse watershed
  - regions of equivalent gradient / flow
  - captures "diamond" boundary of regions



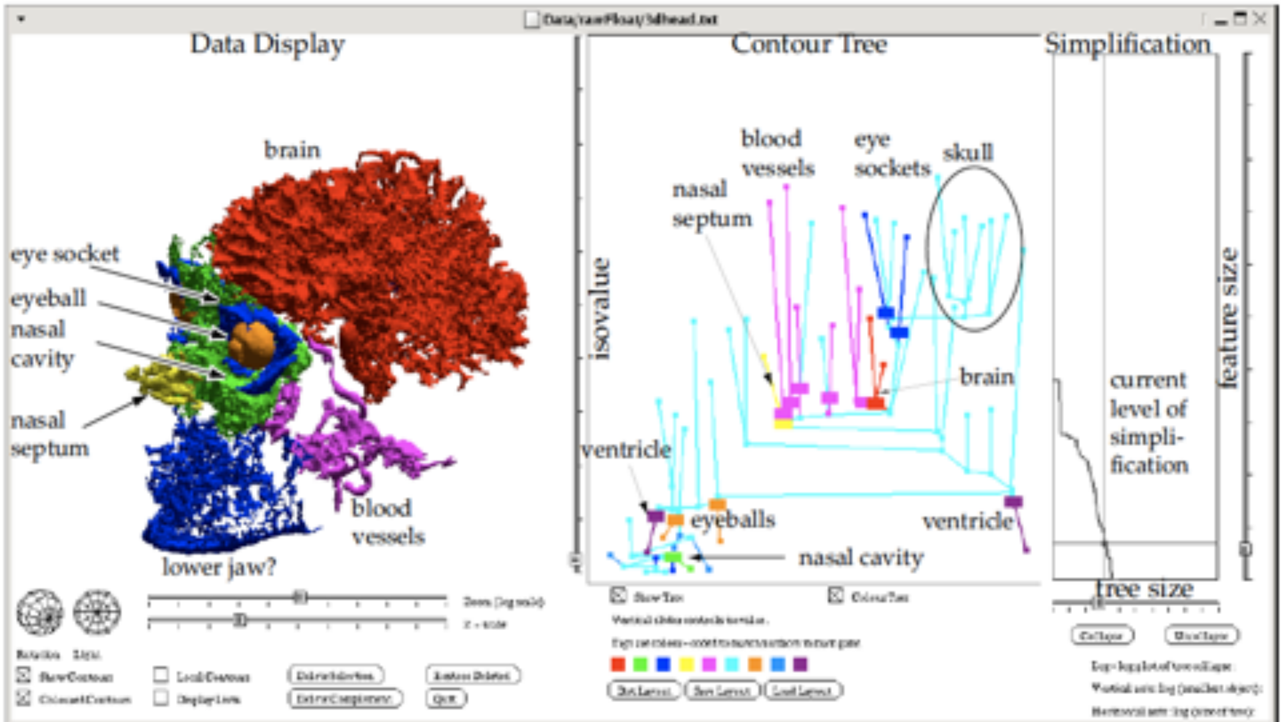


minima

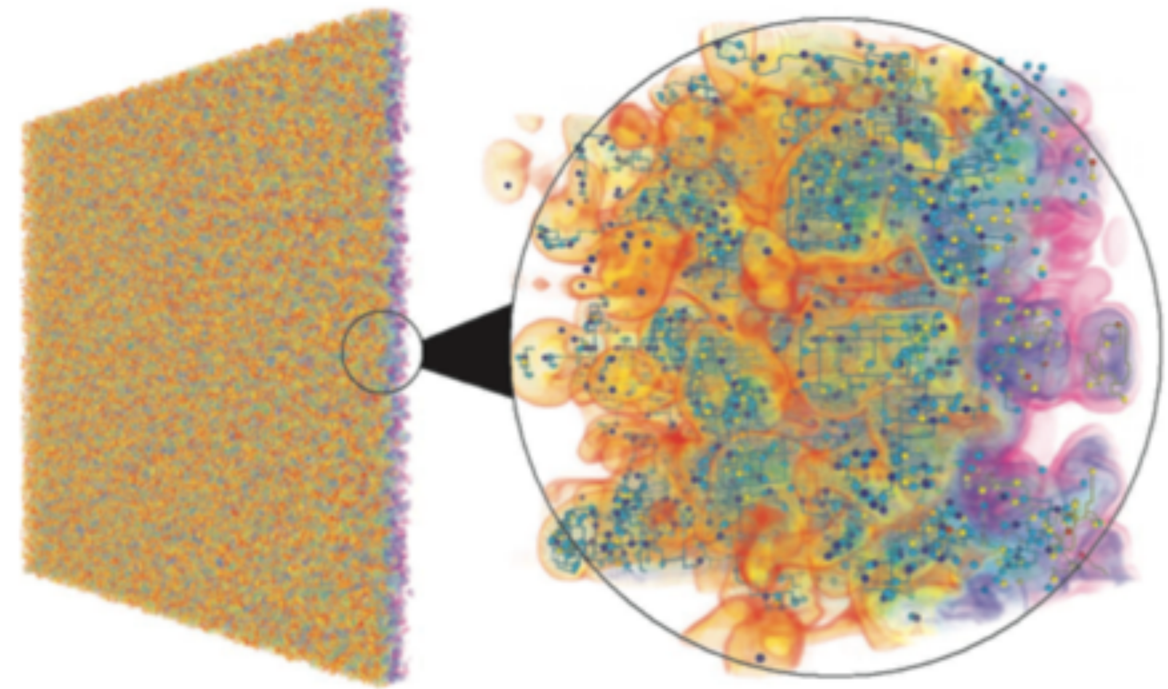
maxima

saddle

Practical examples

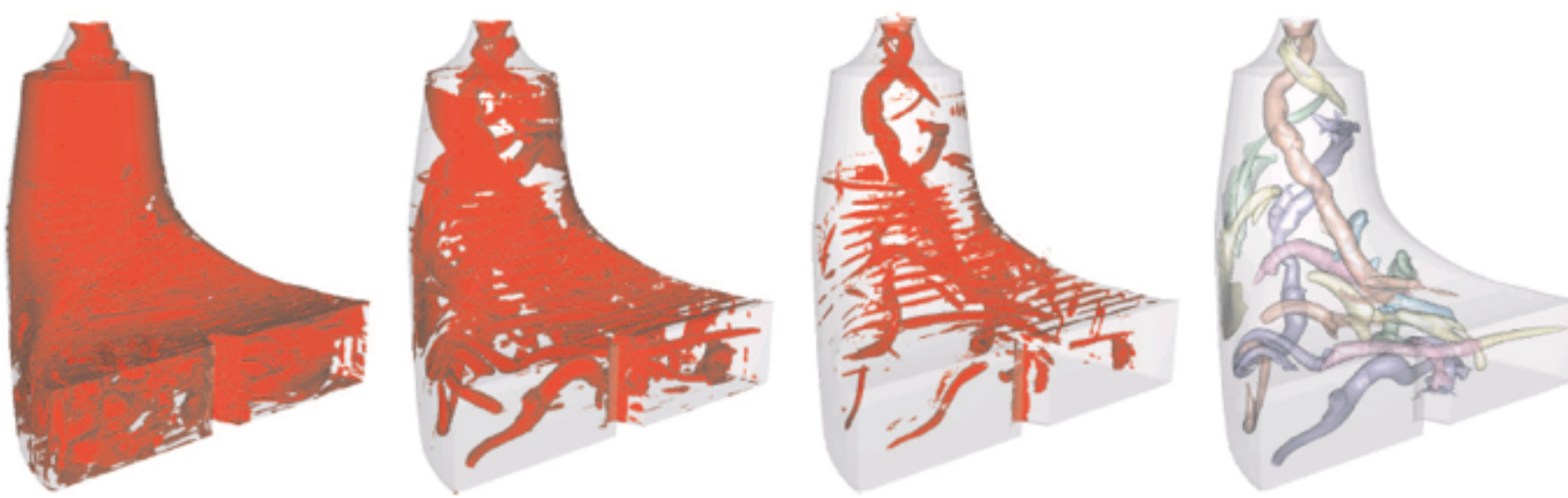


Carr, Snoeyink, & van de Panne, *Flexible isosurfaces: Simplifying and displaying scalar topology using the contour tree*, Computational Geometry, 2010.



Raleigh-Taylor instability, 1152 x 1152 x 1000

Gyulassy, Bremer, Hamann, Pascucci, A Practical Approach to Morse-Smale Complex Computation: Scalability and Generality, IEEE TVCG 14(6), 2008



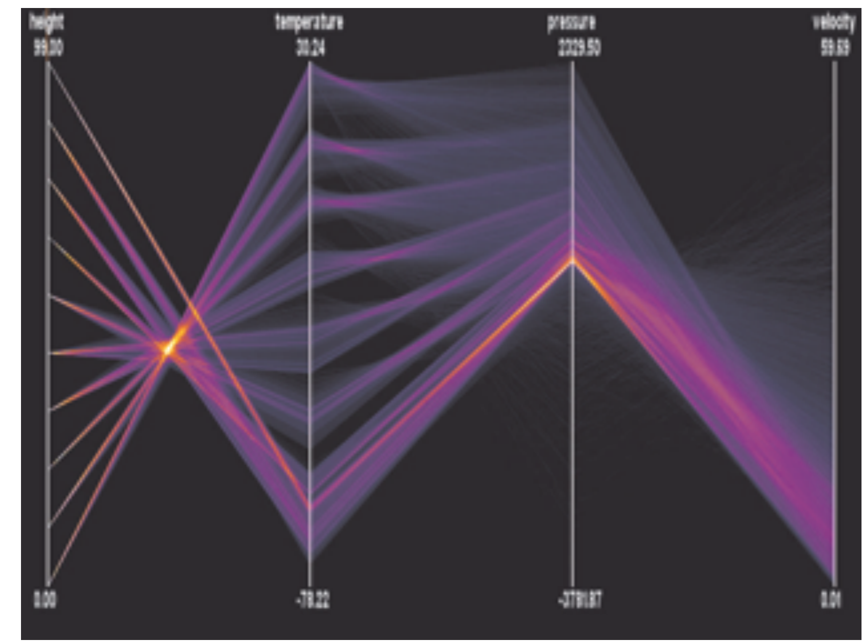
L2 isosurface from CFD simulation of a Francis turbine.

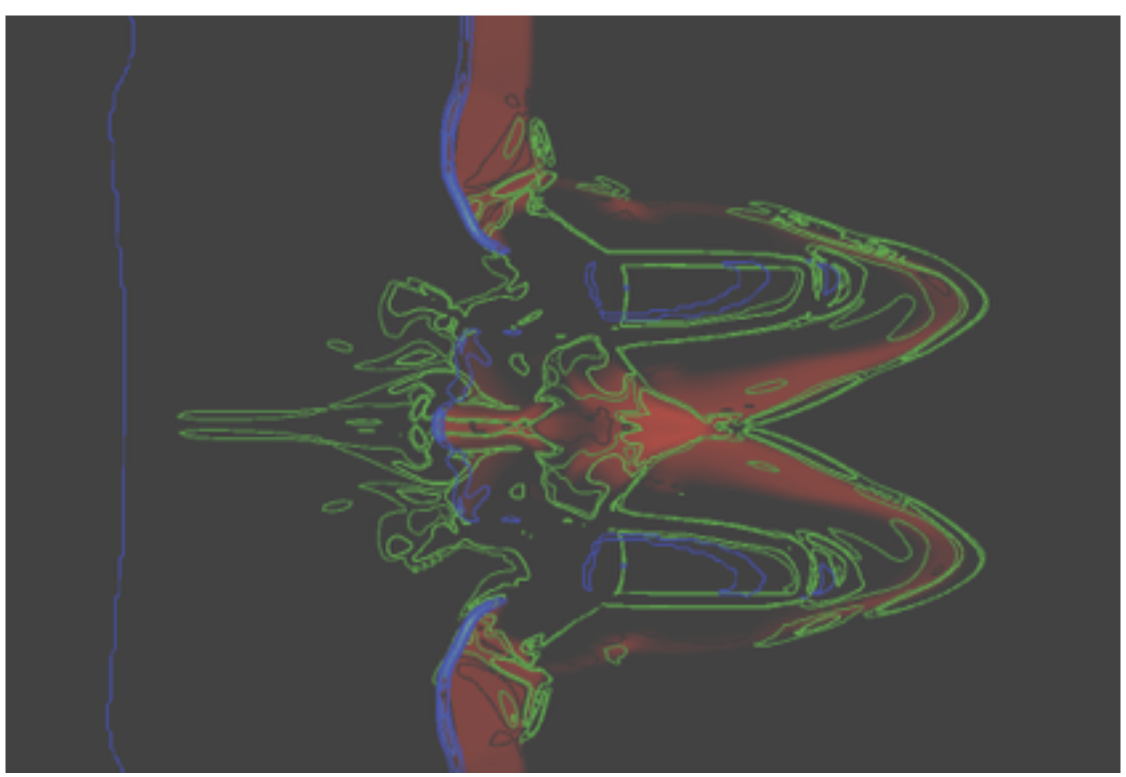
Schneider, Wiebel, Carr, Hlawitschka, and Scheuermann, Interactive Comparison of Scalar Fields Based on Largest Contours with Applications to Flow Visualization, IEEE TVCG, 14(6), 2008.

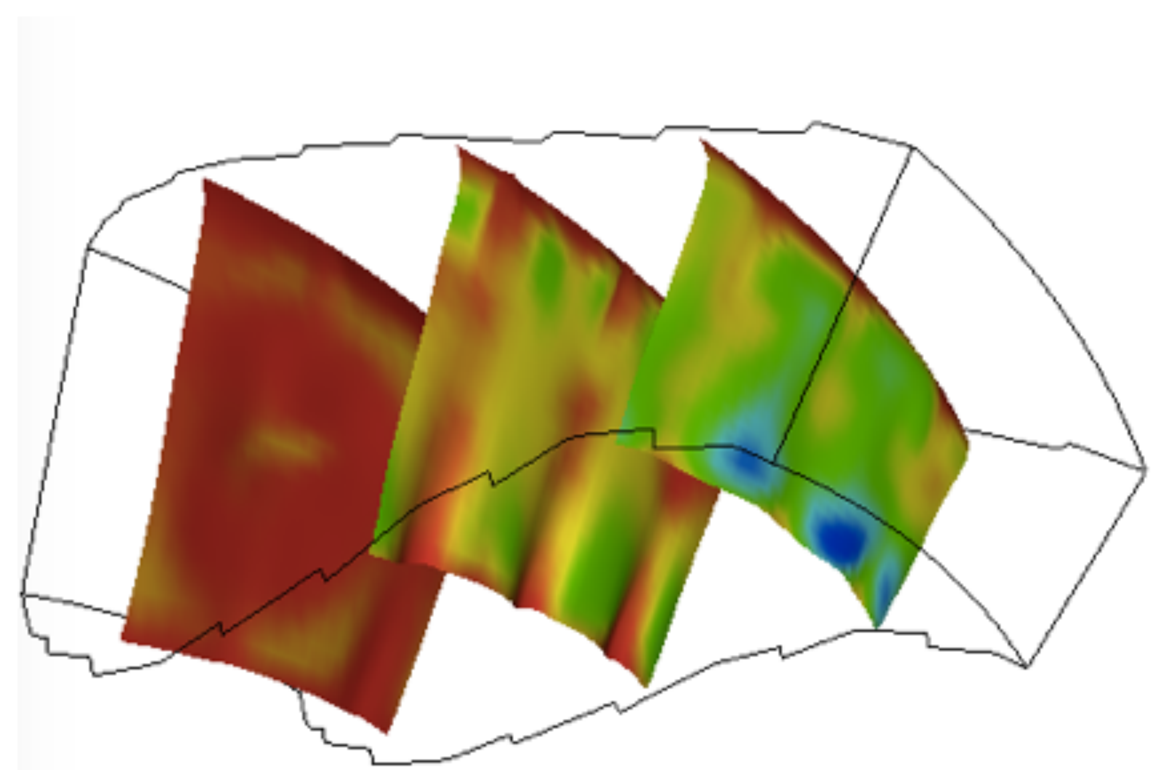
Challenge: m*ultifield* visualisation

UNIVERSITY OF LEEDS

- Understand interaction *between* properties
  - Routine in *information* visualization
  - Tufte: “Graphical excellence is nearly always multivariate”
- For scientific datasets:
  - Harder! Data pinned to points in physical space/time
  - Ad-hoc methods: overlay; probing; scatterplot matrices



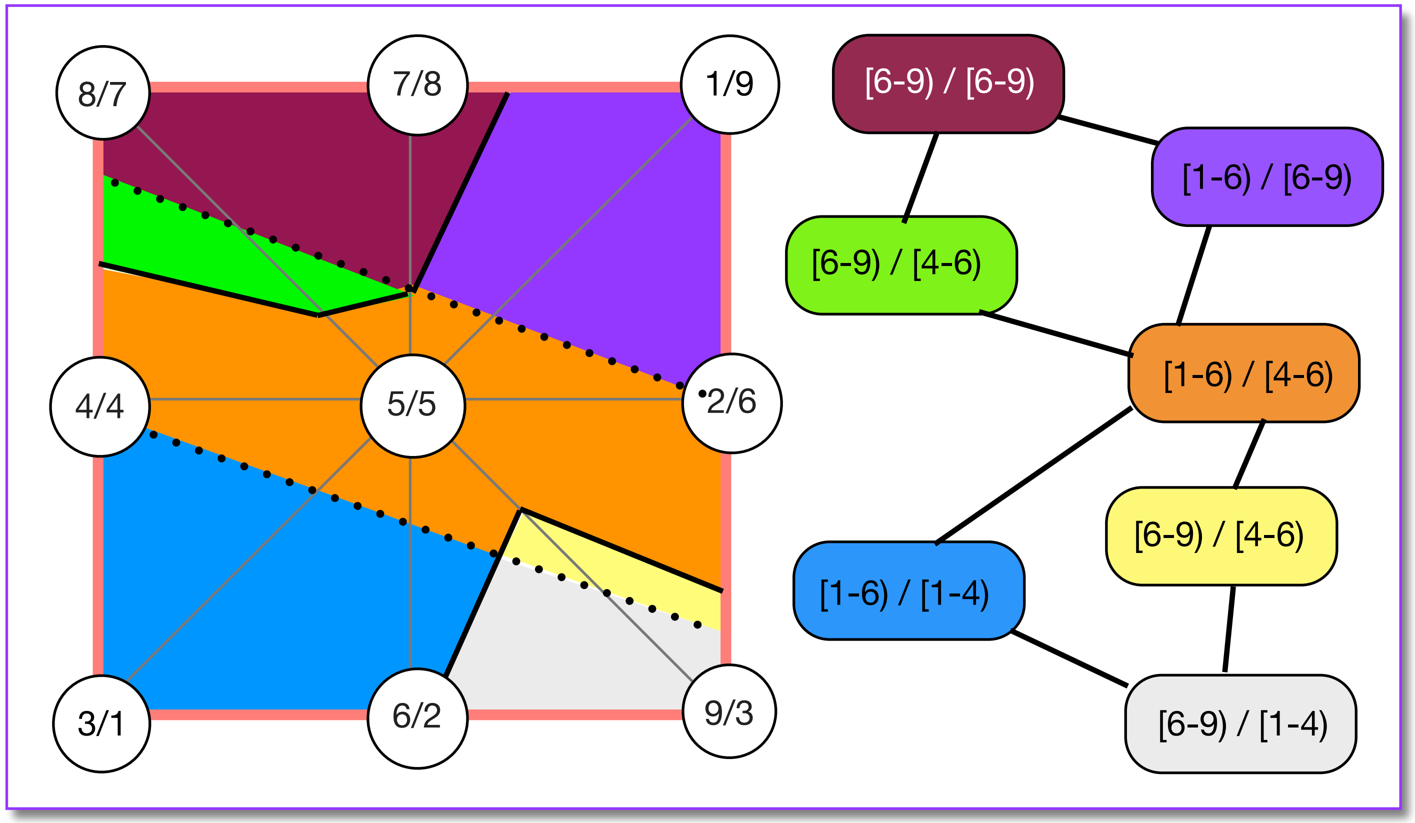




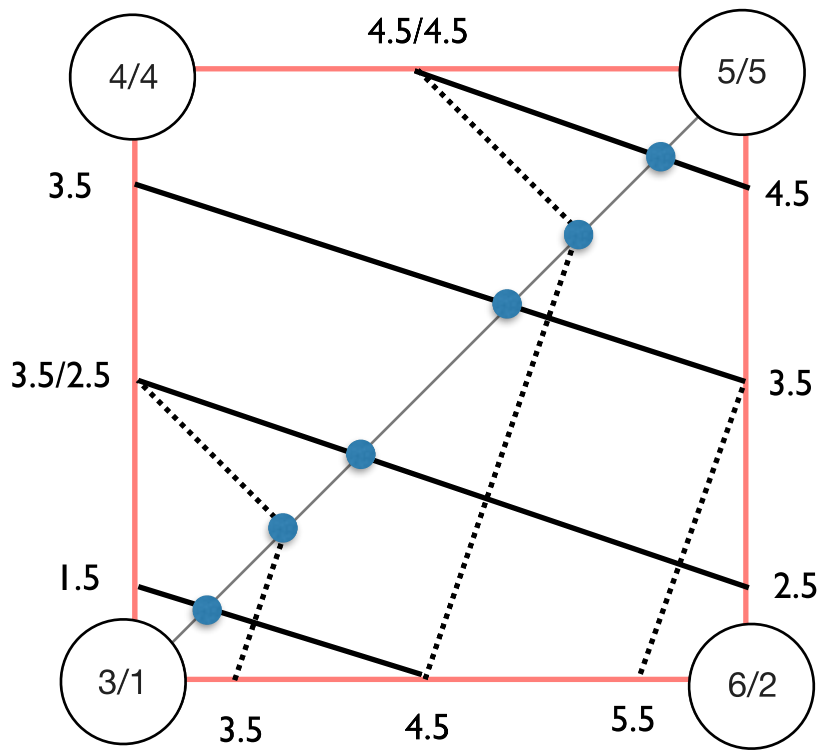
Multifield topology: Joint Contour Net

UNIVERSITY OF LEEDS

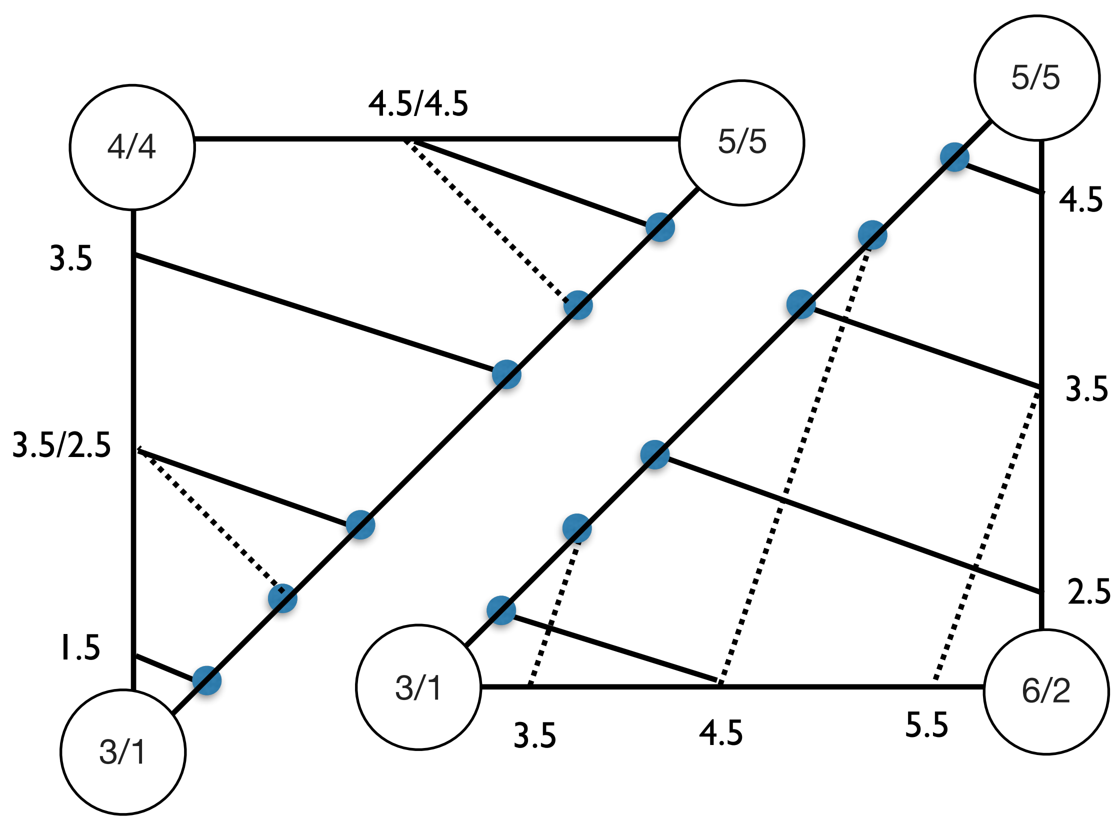
- Generalisation of the Contour Tree / Reeb Graph
    - *family* of scalar fields $f : M \subset R^m \rightarrow R^n$
- Segment domain into regions
  … based on *combinations* of field value.
- JCN
    - nodes = equivalence classes in range (intervals)
    - edges = adjacency
    - reduces to Reeb graph (Contour tree) for n=1.
- No simple concept of minimum / maximum / saddle.
- Any dimension domain.

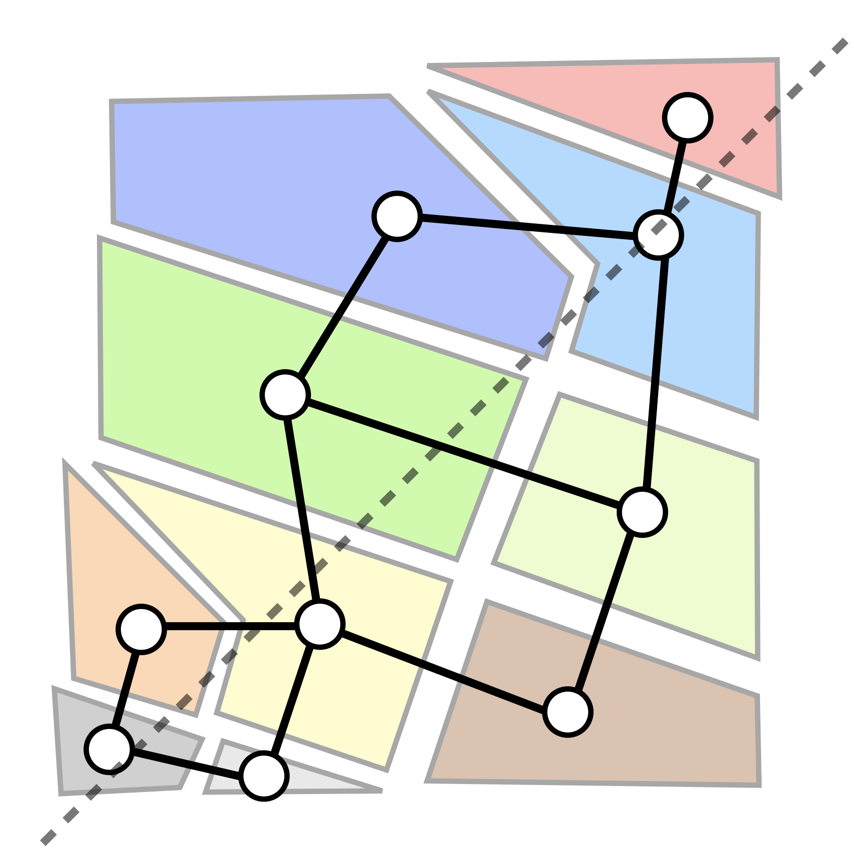


JCN construction



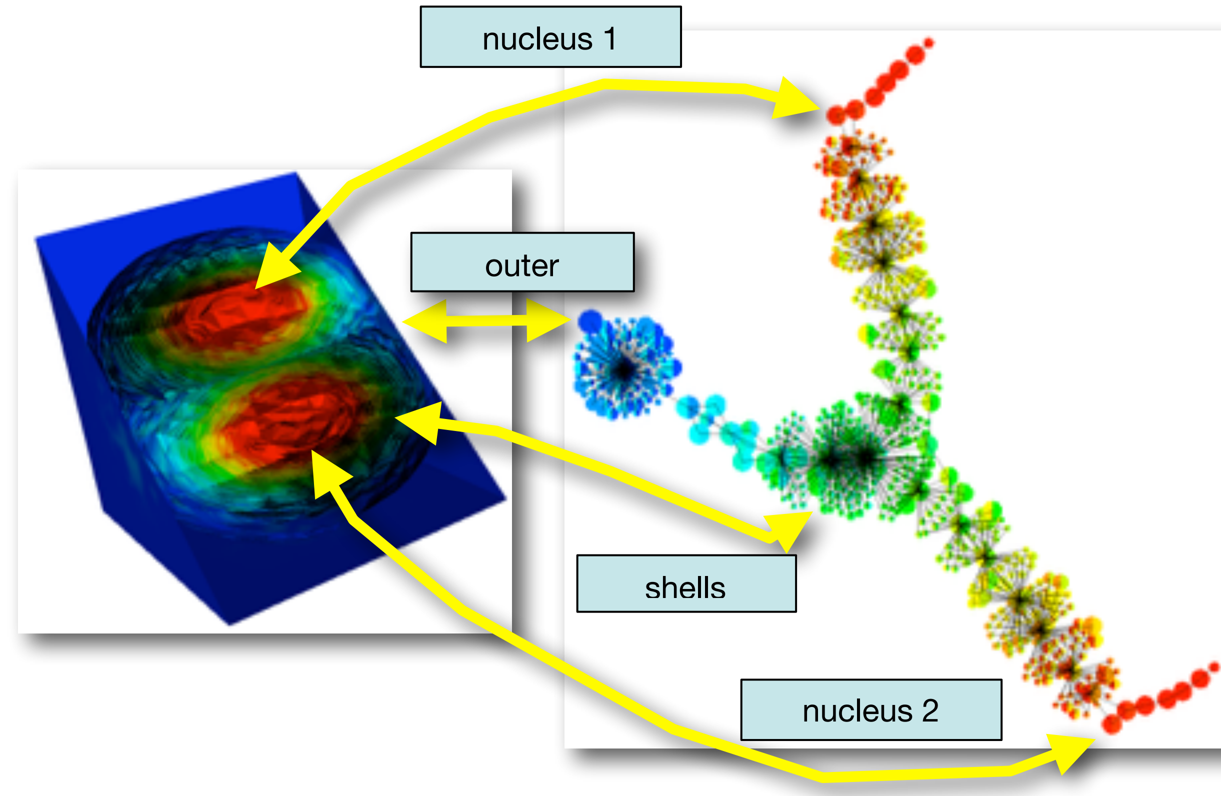

```
frags := cells
For each field d
    for each f in frags
        remove f from frags
        subdivide f against d
        insert results into frags
uf := UnionFind over frags
for each a,b in frags, a /= b
    if a,b are adjacent AND
       AND a,b have equivalent field values
    then merge a,b in uf
JCNnodes := uf classes
JCNedges := { (a,b) in JCNnodes
             | a /= b AND a,b adjacent }
```

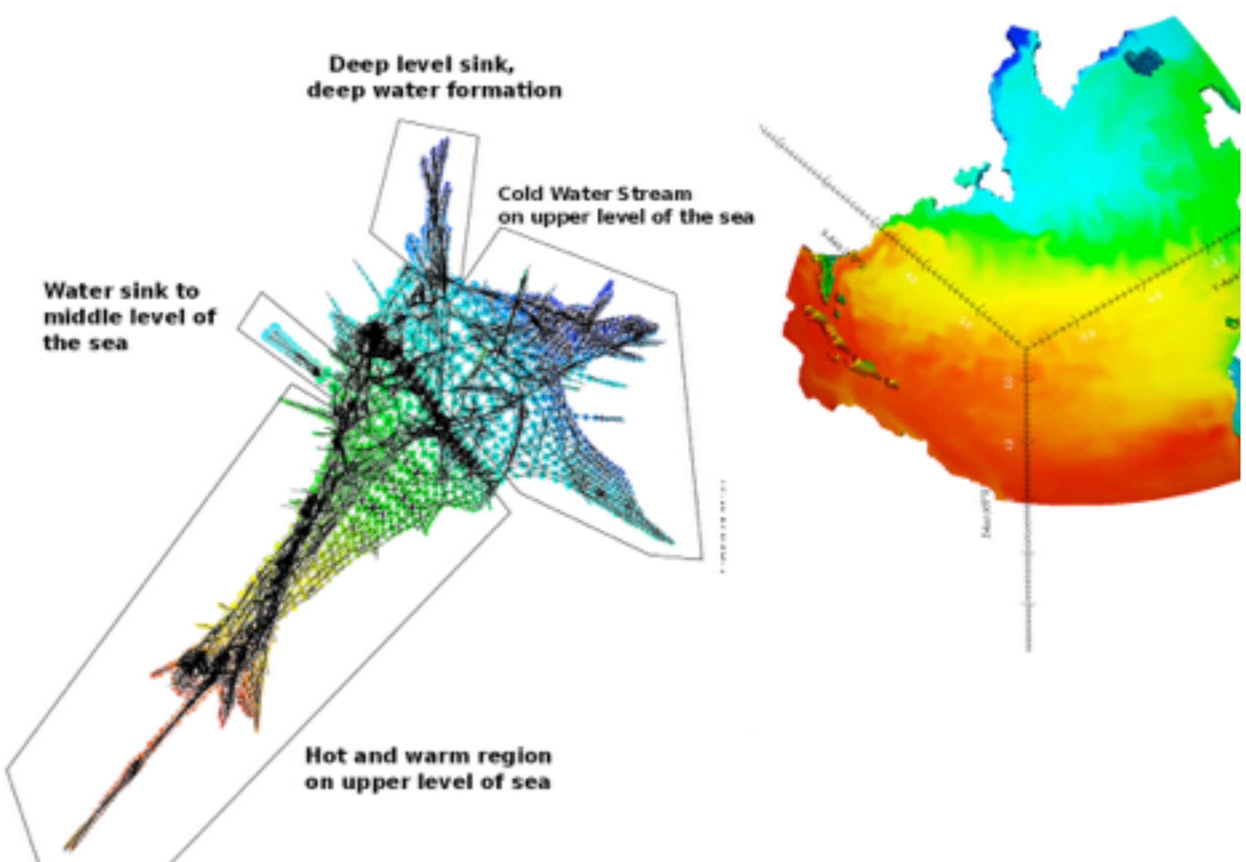


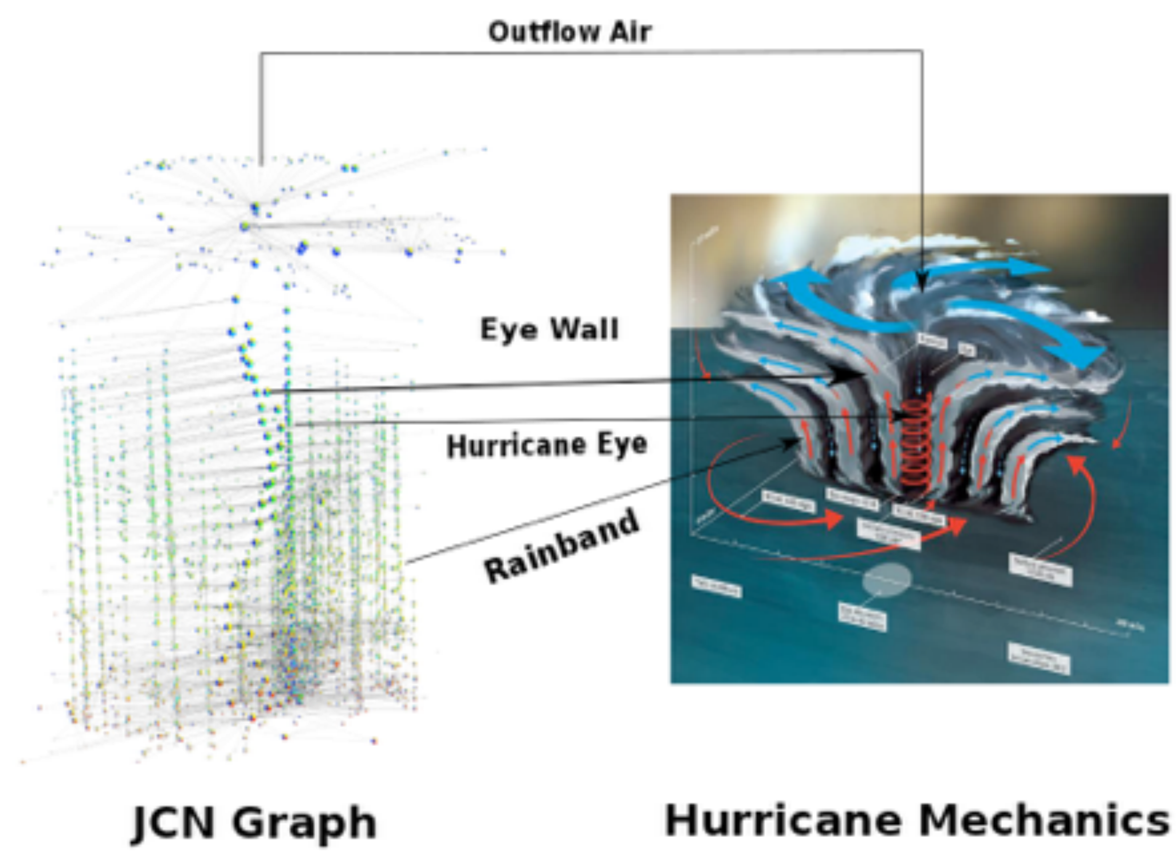


JCN examples

- Nuclear fission
  IEEE Visualization 2012
  Physical Review C 90(5) 2014
  Physical Review C 91(3) 2015
- Oceanography
  Topological Methods in Visualisation 2015
- Hurricane formation
  Computer Graphics & Visual Computing 2014
- Mathematics of singular fibers
  IEEE Visualisation 2015







Functional Programming

UNIVERSITY OF LEEDS

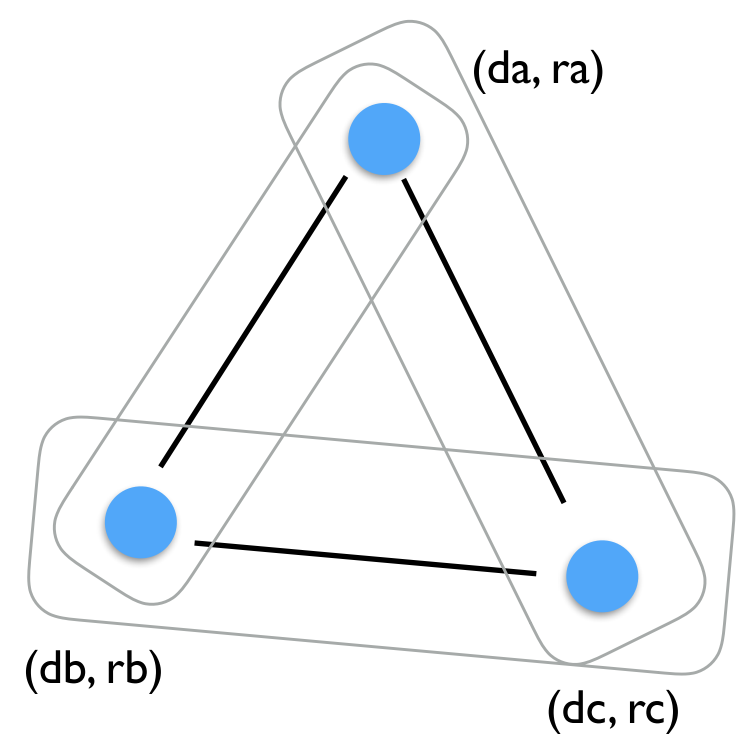
- Why?
  - unexplored territory!
  - much easier / faster / cheaper to explore implementation space
  - build a trusted *reference* implementation
    (cf. testing)
  - longer-term aims: application-quality implementation
    - in-situ visualization (laziness / streaming)
    - heterogeneous platforms (SE costs)
  - previous successes with
    - streaming (lazy marching cubes, Vis 2006)
    - DSLs for data visualization (Vis 2008, ICFP 2008)

- Why not?
  - Haskell not widely known or used in Vis / HPC
  - vis infrastructure comparatively poor (HOpenGL + ?)
  - collaborator wants insight, not beautiful code

Sequential code - types

UNIVERSITY OF LEEDS

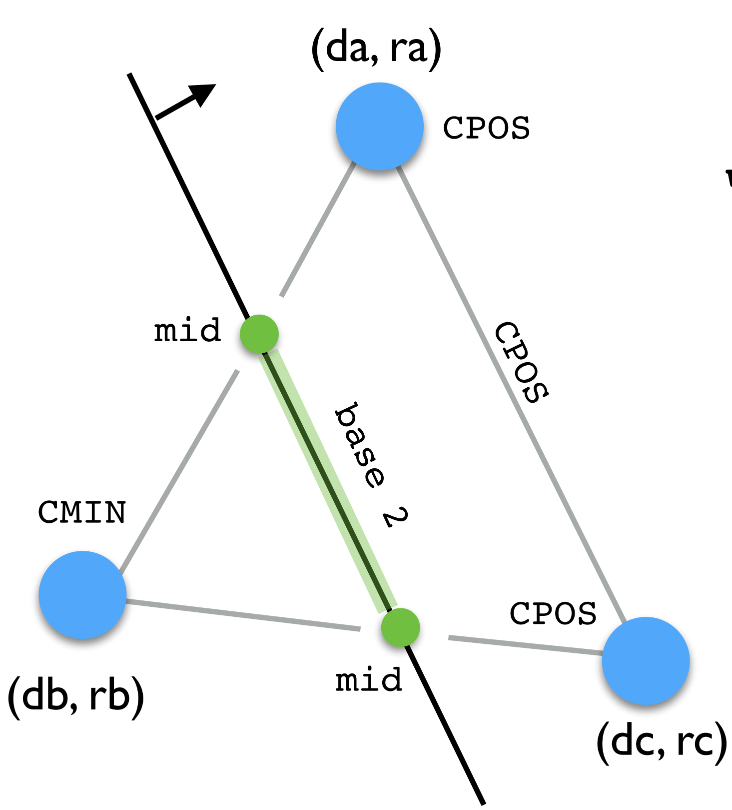
```
data Ptope d r = Ptope TopeMark !Int [Ptope d r]
               | Point TopeMark !d !r
               | Nil

data TopeMark  = None    -- no mark yet computed
               | CPOS    -- polytope is strictly above wrt cut
               | CMIN    -- polytope is strictly below cut
               | CEQ     -- polytope lies in cutting plane

data Outcome   = NonePlus     -- No facet is strictly above the cut plane.
               | NoneMinus    -- No facet is strictly below the plane.
               | Mixed        -- >=1 facet lies on either side of the plane.
               | AllEqual     -- All facets lie in the plane.

type Break d r = ([Ptope d r], [Ptope d r], [Ptope d r])

-- For testing

edge da db ra rb = Ptope None 1 [ Point None da ra
                                , Point None db rb]

tri da db dc ra rb rc
    = Ptope None 2 [ edge da db ra rb
                   , edge db dc rb rc
                   , edge da dc ra rc]
```



Sequential code - polytope cutting

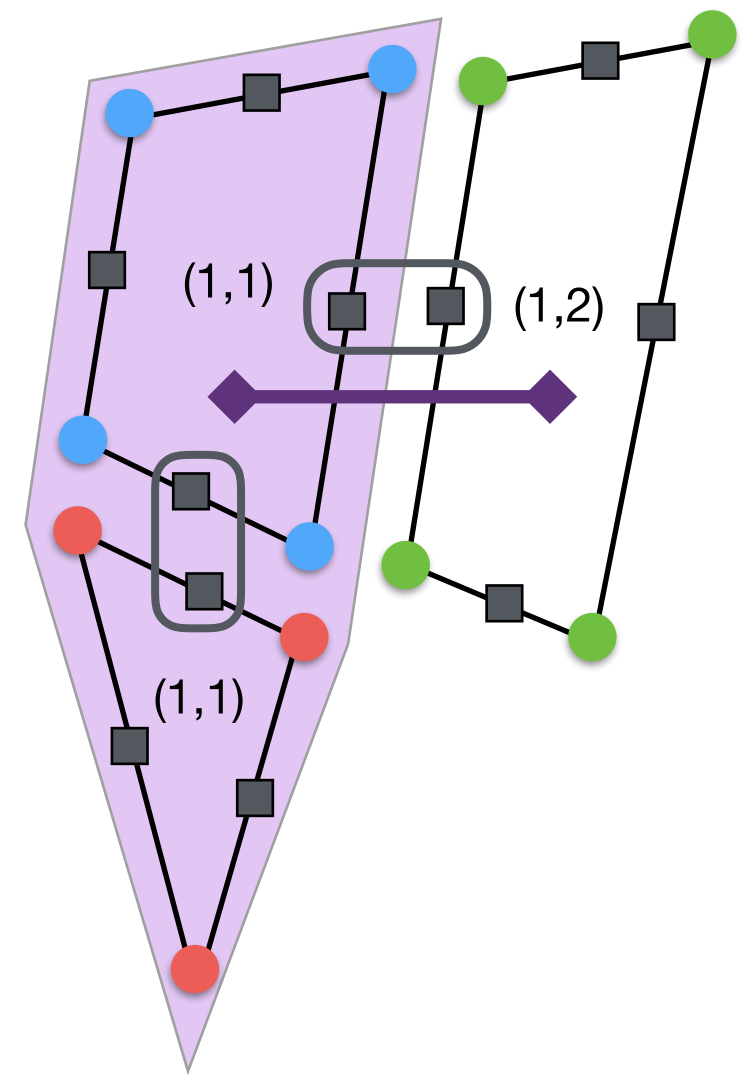
UNIVERSITY OF LEEDS

```
cut :: (Coord d, Coord r) => Int -> Double -> Break d r -> Ptope d r -> Break d r
cut !i !v (mas, eas, pas) !pt
    = case pt of

        Point _ d r -> let !c = classify v (r .! i)
                           !p = Point c d r
                       in case c of
                            CEQ  -> (  mas, p:eas,   pas)
                            CPOS -> (  mas,   eas, p:pas)
                            CMIN -> (p:mas,   eas,   pas)

        Ptope _ d ps -> case split fm fe fp of
                          AllEqual  -> (mas, (Ptope CEQ d fe) : eas, pas)
                          Mixed     -> let pl = (base d fp)
                                           ml = (base d fm)
                                       in ((Ptope CMIN d ml) : mas,  eas, (Ptope CPOS d pl) : pas)
                          NonePlus  -> ((Ptope CMIN d (fm++fe)) : mas, eas, pas)
                          NoneMinus -> (mas, eas, (Ptope CPOS d (fe++fp)) : pas)
                        where
                          (fm, fe, fp)  = foldl' (cut i v) ([],[],[]) ps
                          base 1 ps = mid : ps
                          base n ps = Ptope CEQ (n-1) [p | p <- concatMap facets ps, mark p == CEQ] : ps
                          mid = Point CEQ dp rp
                                where
                                    [Point _ da ra, Point _ db rb] = (fm++fe++fp)
                                    !ai = ra .! i
                                    !bi = rb .! i
                                    !p  = (v - ai) / (bi - ai)
                                    !dp = czipWith (\u v -> (1-p)*u + p*v) da db
                                    !rp = czipWith (\u v -> (1-p)*u + p*v) ra rb
```



Sequential code - fragment merger


```
data Fragment d r = Fragment { fragCoord  :: r
                             , fragFacets :: [d]
                             }

linkFaces :: (Coord d, Coord r)
          => FragValue s r    -- map from fragment nr to range value
          -> Int              -- number (id) of fragment containing these faces
          -> UnionFind s      -- the union-find structure
          -> KD.KDtree d Int  -- spatial search structure for fragment centers
          -> [(Int, Int)]     -- current proto-graph (edges between adjacent fragments)
          -> [d]              -- list of face center points still to process
          -> ST s (KD.KDtree d Int, [(Int, Int)])

linkFaces sb n uf faces gr []     = return (faces, gr)
linkFaces sb n uf faces gr (f:fs)
    = do case KD.find f faces of
           Nothing  -> let kd = KD.insert f n faces
                       in linkFaces sb n uf kd gr fs
           Just m   -> do { nval <- sb `getA` n
                          ; mval <- sb `getA` m
                          ; if nval == mval
                            then do { union n m uf
                                    ; let kd = KD.delete f faces
                                    ; linkFaces sb n uf kd gr fs
                                    }
                            else do { let kd = KD.delete f faces
                                    ; linkFaces sb n uf kd ((n,m):gr) fs
                                    }
                          }
```



Opportunities for parallelisation

- ... can be non-trivial to spot!

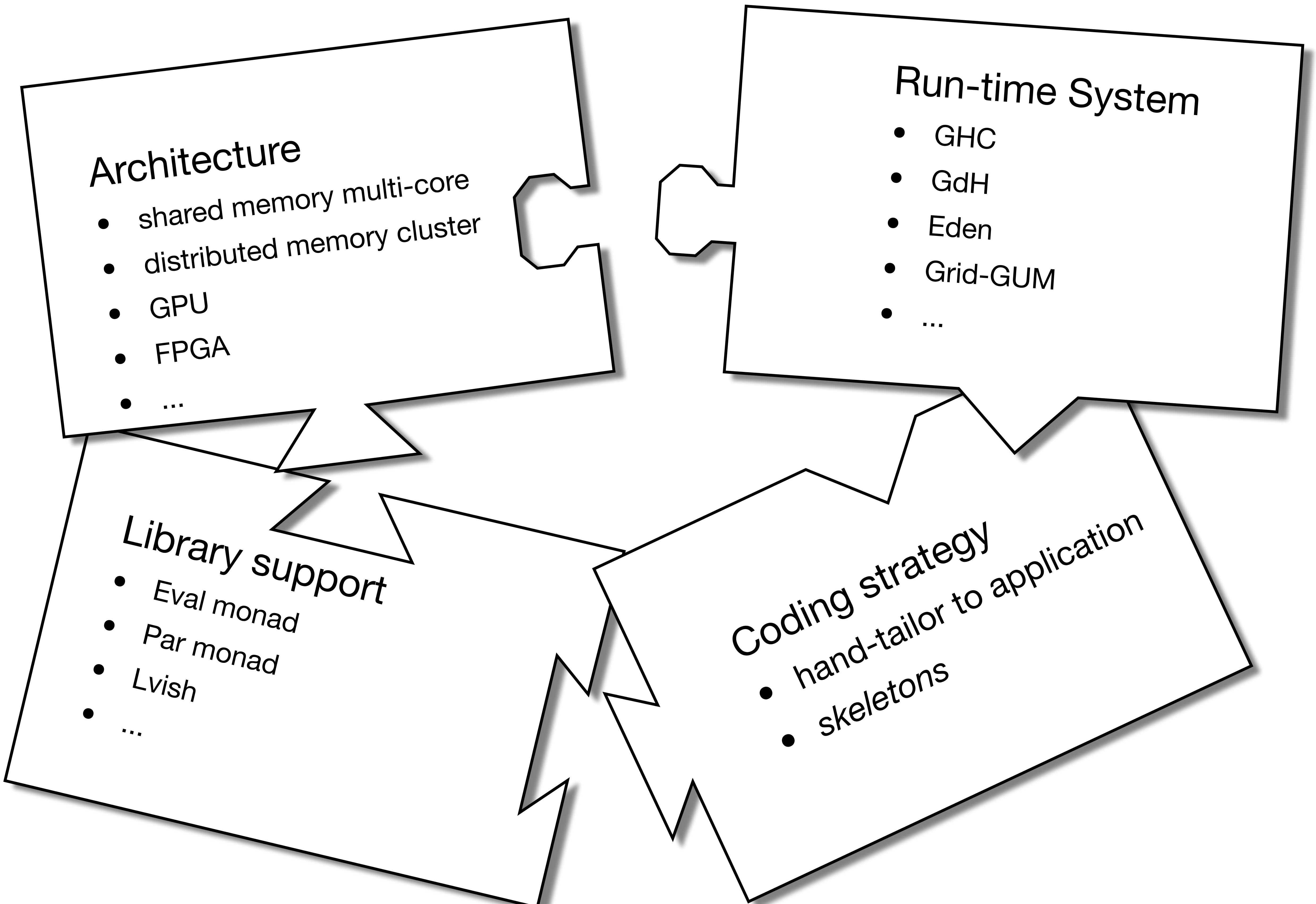
```
makeJCN :: (Coord d, Coord r) => (d, d) -> r -> r -> [Cell d r] -> JCN d r
makeJCN extent fmin swid
      = mergeFrags extent
        . concatMap (cellFrags fmin swid)

cellFrags ::  (Coord d, Coord r) => r -> r -> Cell d r -> [Fragment d r]
cellFrags fmin swid
     = map (mkfrag fmin swid) . polytopes fmin swid

mkfrag :: (Coord d, Coord r) => r -> r -> Ptope d r -> Fragment d r
mkfrag fmin swid ptps
    = let mins    = czipWith3 slab fmin swid crds
          crds    = minCorner $ rngCoords ptps
          centers = map center . ptfacets $ ptps
      in Fragment mins centers

dice :: (Coord d, Coord r) => [Ptope d r] -> (Int, [Double]) -> [Ptope d r]
dice ptopes (i, [])      = ptopes
dice ptopes (i, (v:vs))
    = below ++ dice above (i, vs)
      where
          (below, above) = diceEach ptopes ([], [])
          diceEach [] res = res
          diceEach (p:ps) (bs, as)
              = let (!m1s,!e1s,!p1s) = cut i v ([], [], []) p
                  in diceEach ps (m1s ++ bs, e1s ++ p1s ++ as)
```

Parallel Topology

- Historically challenging to parallelize topology
  - topology = *global* property
  - stubbornly serial!
  - analysis overhead already large (est. 1.2KB per point)
  - divide-and-conquer (Cole-McLaughlin & Pascucci)
  - cost of maintaining information to "zip" partial solutions

- Large-scale datasets => emphasis on distributed memory
  - communication costs dominate
  - solutions to date:
    - distribute the data structure, then query (Weber & Morozov)
    - stratify, terminate at desired resolution (Gyulassy, Landge, Pascucci et.al.)

The parallel Haskell jigsaw



Skeletons

- Algorithmic Skeletons:
  Structured Management of Parallel Computation, Cole, 1989.
- From *Loogen et.al.*
  - commonly-used patterns of parallel evaluation
  - simplify development .. can simply be used in a given application context
  - may be different implementations
  - efficiency given by a cost model

For a functional programmer, skeleton = HOF

`Par` Monad

- Deterministic and explicit parallel computation
  - Computations that fork and communicate
  - IVars for data communication
  - Thread evolution as continuation monad over a meta-scheduler state
  - Implemented using reflection of GHC RTS

- Interface

```
class NFData where ...

runPar :: Par a -> a

fork :: Par () -> Par ()

new  :: Par (IVar a)
put  :: NFData a => IVar a -> a -> Par ()
get  :: IVar a -> Par a
```

Skeletons in the `Par` monad

UNIVERSITY OF LEEDS

- Parallel map:

```
parMap :: NFData b => (a -> b) -> [a] -> Par [b]
parMap f []     = []
parMap f (a:as) = do b <- spawn $ return (f a)
                     bs <- parMap f as
                     return $ b:bs
```

- Chunked map:

```
parMapChunk :: Int -> (a -> b) -> [a] -> Par [b]
```

- Divide-and-conquer:

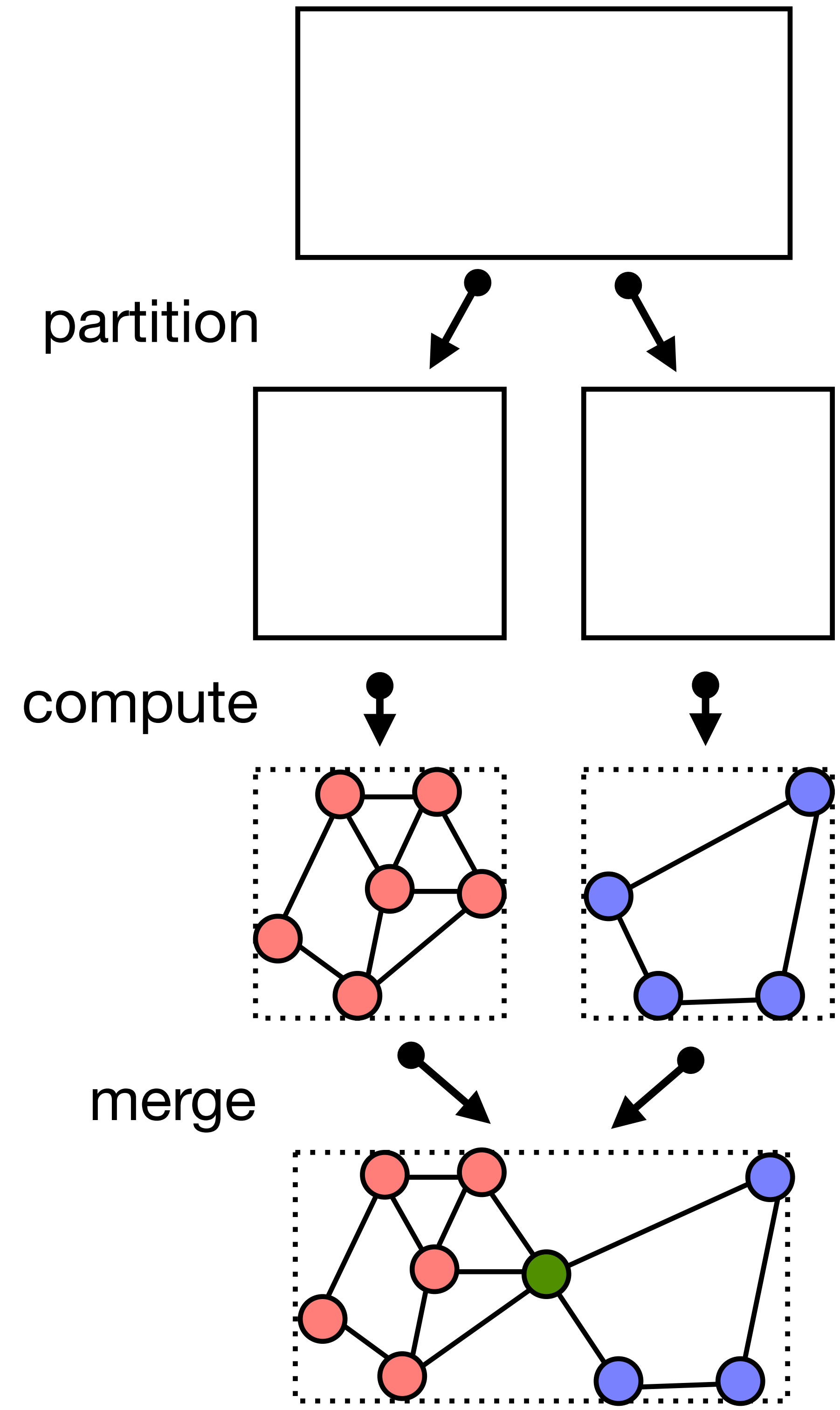
```
parDaC :: NFData s
       => (p -> Bool)     -- can the problem be simplified?
       -> (p -> (p,p))    -- subdivide problem
       -> (s -> s -> s)   -- merge two solutions
       -> (p -> s)        -- solve a "trivial" problem
       -> p               -- initial problem
       -> s               -- solution
```

Intermission

Parallelizing JCN

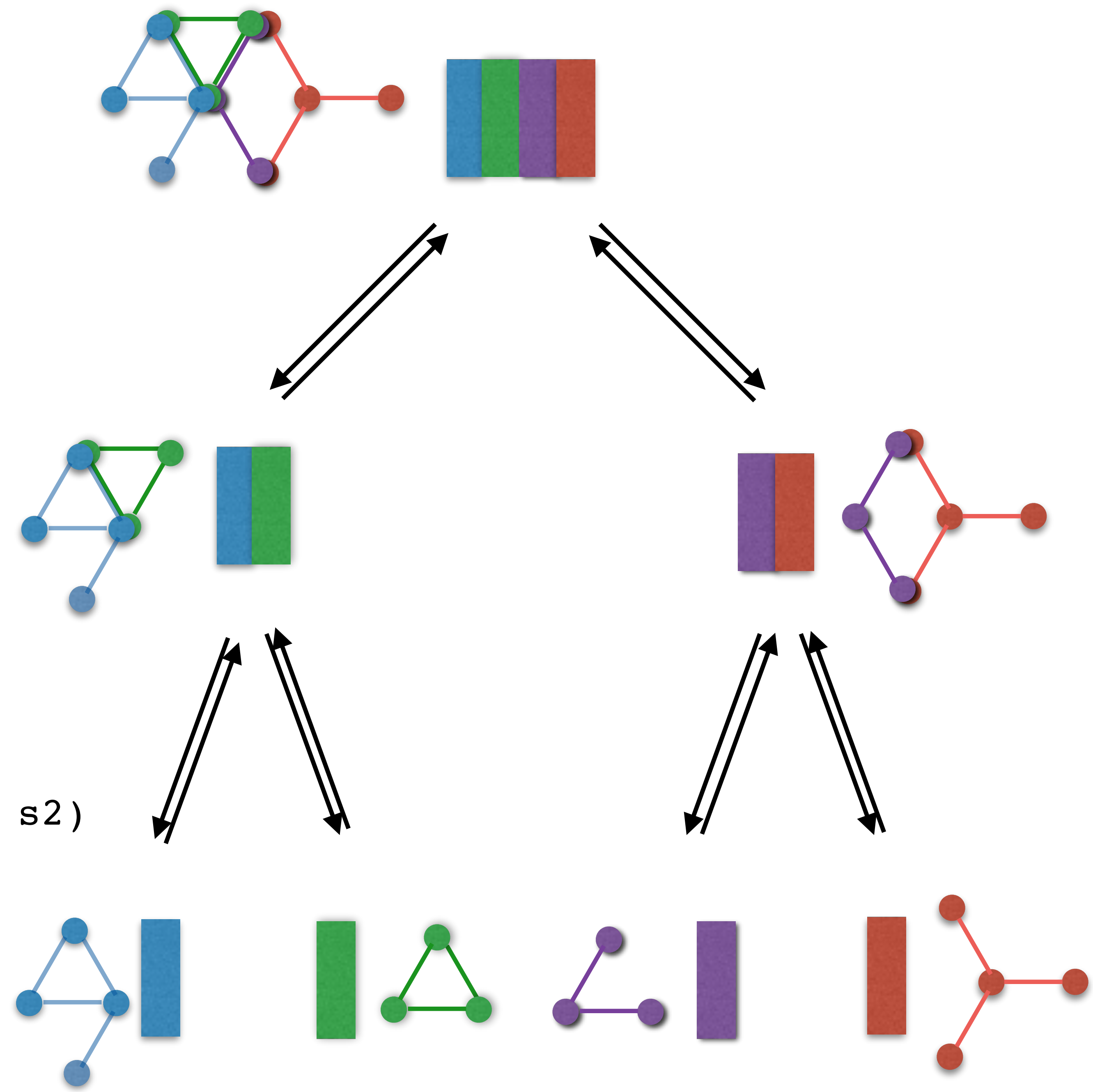
UNIVERSITY OF LEEDS

- Embarrassingly parallel
  - partition the domain
  - compute local JCNs
  - merge JCNs to generate the global structure
- Input/Output
  - Each process access the input files directly
  - The main process write the final JCN to the output file
- Merger
  - need to capture fragment adjacencies
  - solution:
    - facet mid-points are unique
    - **store in spatial search structure (KD-tree)**



Divide-and-conquer skeleton

```
parDaC :: NFData s
  => (p -> Bool)      -- is trivial?
  -> (p -> (p,p))     -- subdivide
  -> (s -> s -> s)    -- merge
  -> (p -> s)         -- compute
  -> p
  -> s

parDaC refine split merge solve problem
  = runPar $ go problem
    where go p
          | refine p
               = do let (p1, p2)  = split p
                        s1 <- spawn $ go p1
                        s2 <- spawn $ go p2
                        liftM2 merge (get s1) (get s2)
          | otherwise = return (solve p)
```



Map skeleton

UNIVERSITY OF LEEDS

- Exploit parMap

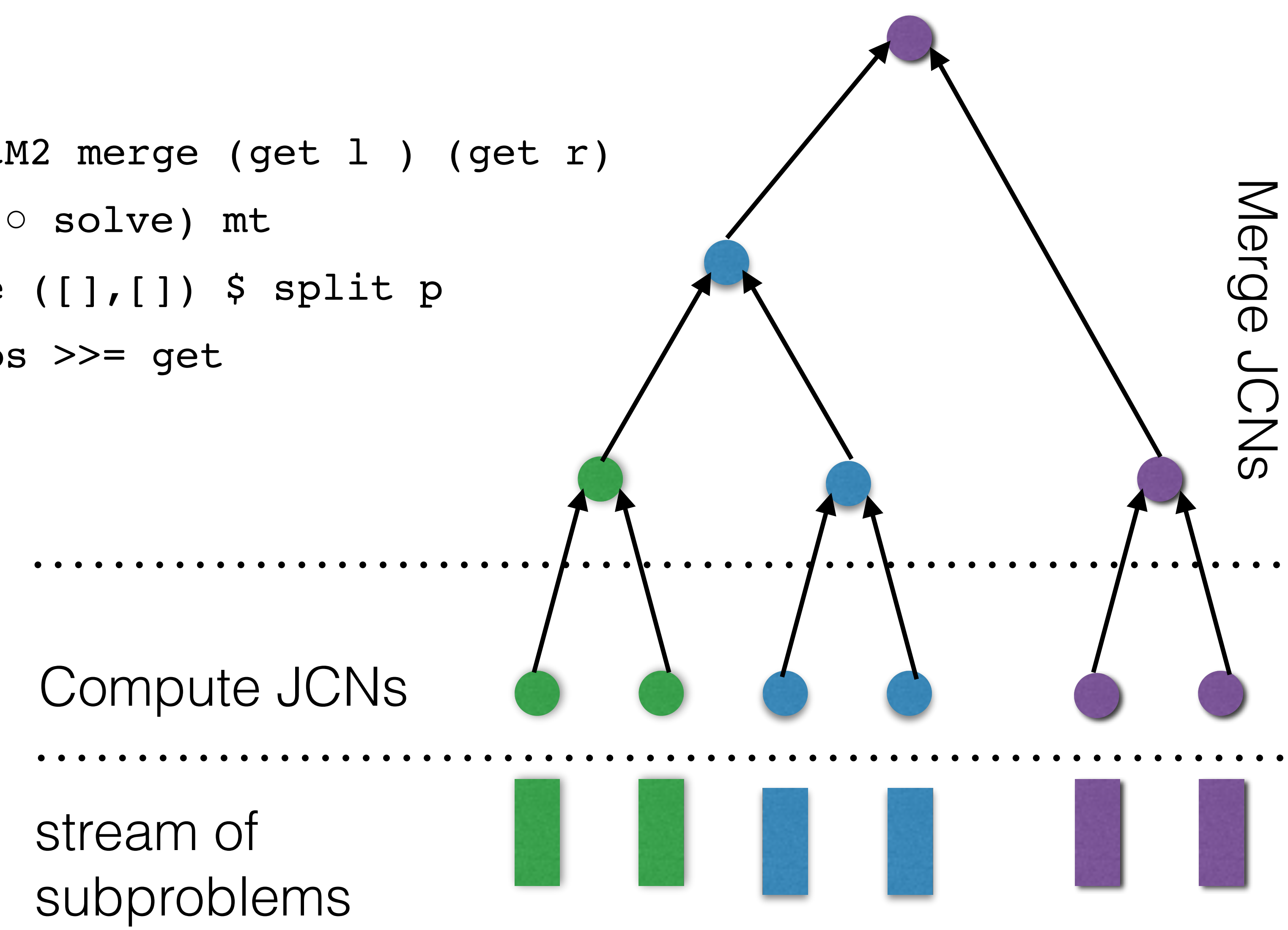
```
mapSkel :: NFData s
        => (p -> [sp])       -- split into sub-problems
        -> (sp -> ss)        -- solve a sub-problem
        -> ([ss] -> s)       -- merge sub-problem solutions
        -> p                 -- initial problem
        -> s                 -- solution

mapSkel split solve merge p
    = runPar . liftM merge . parMap solve . split $ problem
```

- parMapChunk variant

Eager skeleton

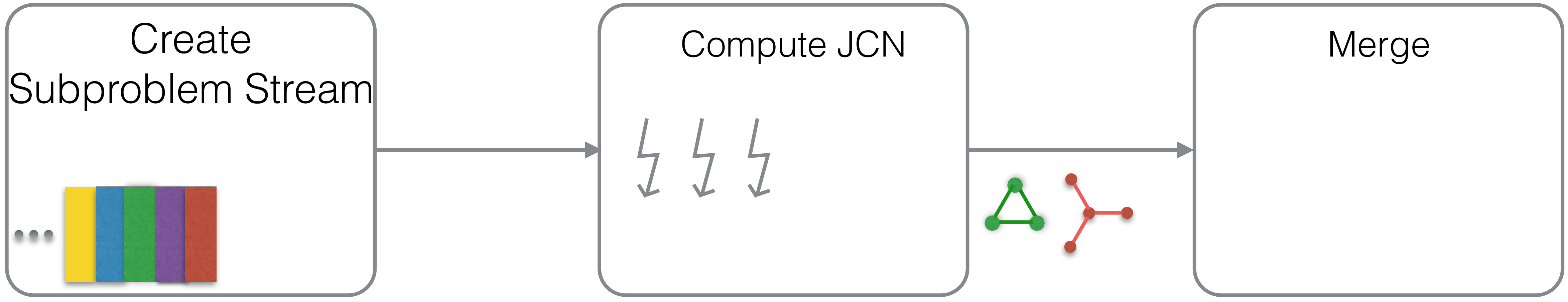
- Input divided into stream of subproblem pairs
- One thread spawned for each subproblem & merge step
- IVars communicate solutions towards the root

```
eager split solve merge p
  = runPar $ do
      let mt l r = spawn $ liftM2 merge (get l ) (get r)
      let tree = sched (return ○ solve) mt
      (mrgJobs, ) <- foldM tree ([],[]) $ split p
      finishMerger mTask mrgJobs >>= get
```



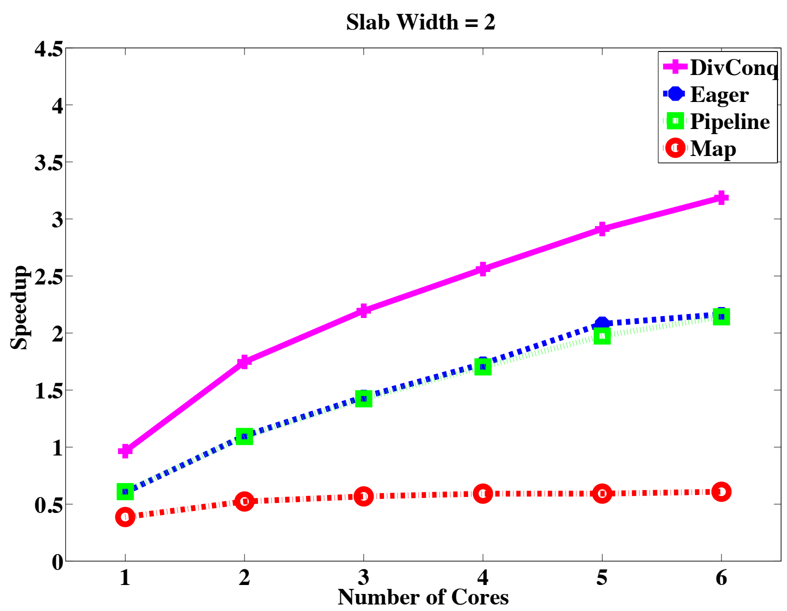
Pipeline skeleton

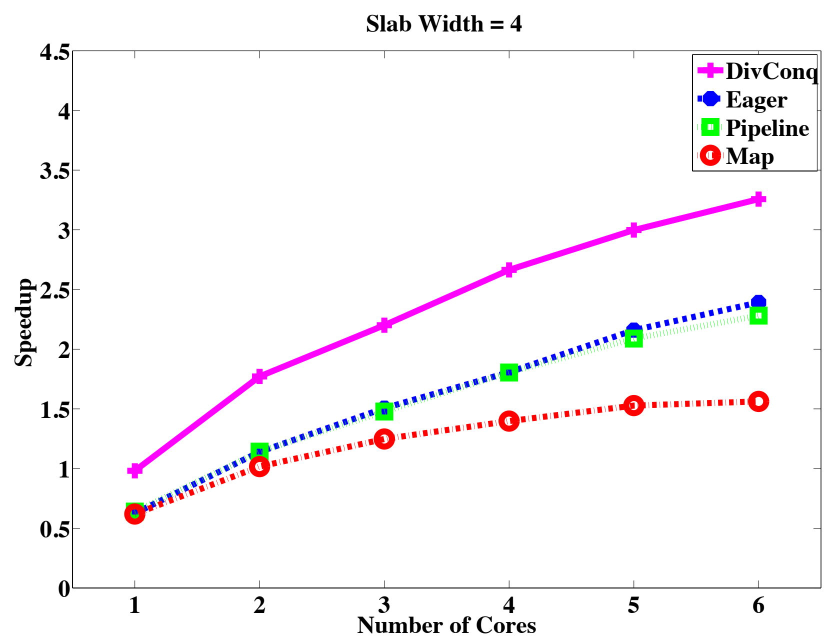
- Eager Skeleton was too eager
  - fragmentation faster than merger
  - leaves complete, building up large volume of data
- Rate-limiting mechanism
  - delay fragmentation until pending mergers complete

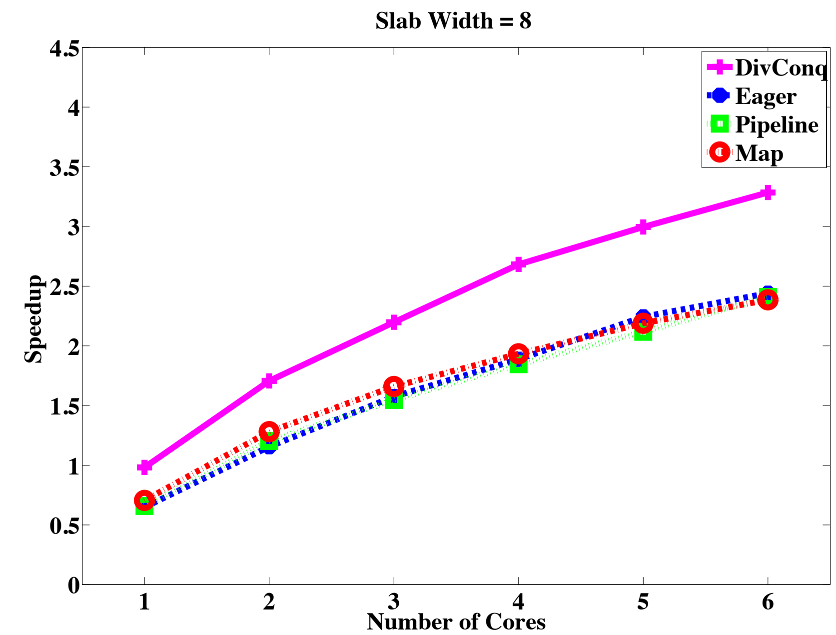


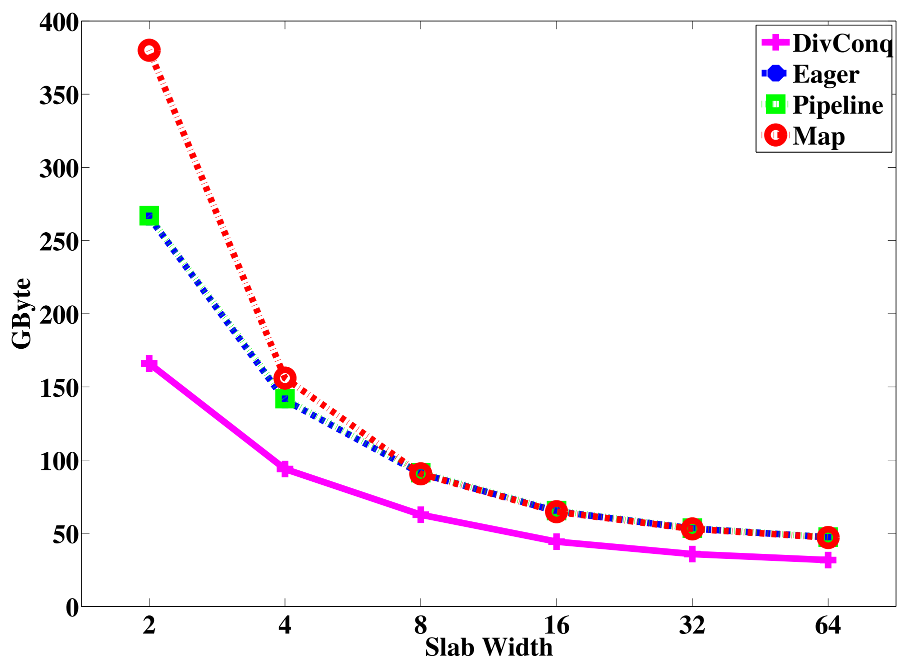
Parallel Performance

UNIVERSITY OF LEEDS

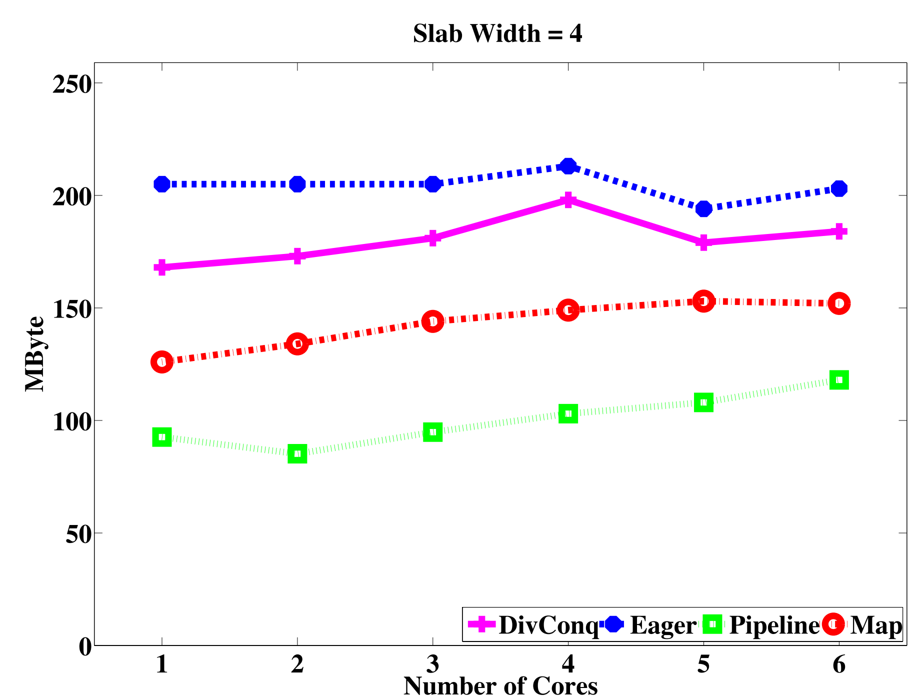








Total Memory Allocation



Maximum Memory Residency

Distributed-memory parallelism

UNIVERSITY OF LEEDS

- Why?
  - massive simulations
  - e.g. PB-scale combustion studies at SCI (Utah) and LBL
- Distributed topology
  - Distributed merge/contour tree implementations [on Cray] (Morozov & Weber, 2013)
  - Partial trees are distributed across nodes
- Platform
  - ARC1: cluster built on 32-core nodes
  - Using Eden, a distributed-memory fork of GHC
  - Eden Trace-Viewer: similar to threadscope, currently less information
- Starting point: three *Eden* skeletons
  - parallel map-reduce (disMapReduce)
  - distributed divide-and-conquer (disDC)
  - workpool

Eden language

- Eden primitive constructs:

```
class Trans a where ...

process :: (Trans a, Trans b) => (a -> b) -> Process a b
(#) :: (Trans a, Trans b) => Process a b -> a -> b
```

- Eden's RTS extends the GHC runtime system:
  - process creation (strict)
  - task placement
  - synchronisation
  - data communication
    - implicit parent-child channels
    - explicit creation and use of dynamic channels

Eden skeletons

UNIVERSITY OF LEEDS

- Map-reduce:

```
map :: (a -> b) -> [a] -> [b]

foldr :: (b -> c -> c) -> c -> [b] -> c

mapRedr :: (b -> c -> c) -> c -> (a -> b) -> [a] -> c
mapRedr rf e mf = (foldr rf e) . (map mf)
```

- Distributed map-reduce:

```
disMap :: (Trans a, Trans b) -> (a -> b) -> [a] -> [b]

disMapRedr :: (Trans a, Trans b) => (b -> b -> b) -> b -> (a -> b)->[a] -> b
disMapRedr rf e mf xs
   | noPe == 1 = mapRedr g e f xs
   | otherwise = (foldr rf e)
                 . (disMap (disMapRedr rf e mf))
                 . (splitIntoN noPe) $ xs
```

The Experimental Platform

UNIVERSITY OF LEEDS

- ARC2 : One of the HPC clusters at University of Leeds

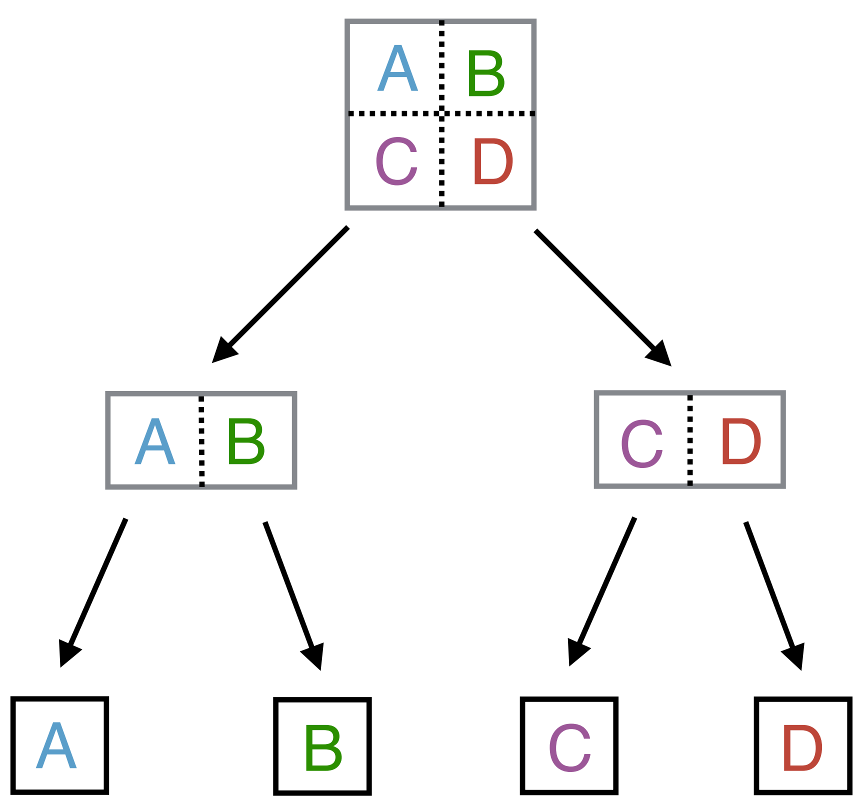
Compute	• 3040 cores
	• Each Node has
	  • *A **dual** socket with 2.6GHz **8-core*** Intel E5-2670 processors
	  • ***32GB*** of *DDR3 memory*
	  • *500 Gb* of *local Hard drive*

Storage	• ***Lustre file system*** • *Delivering 4GB/s via the InfiniBand network* • 170TB storage
Network	• All user traffic data is transferred over ***InfiniBand*** network • Gigabit for management

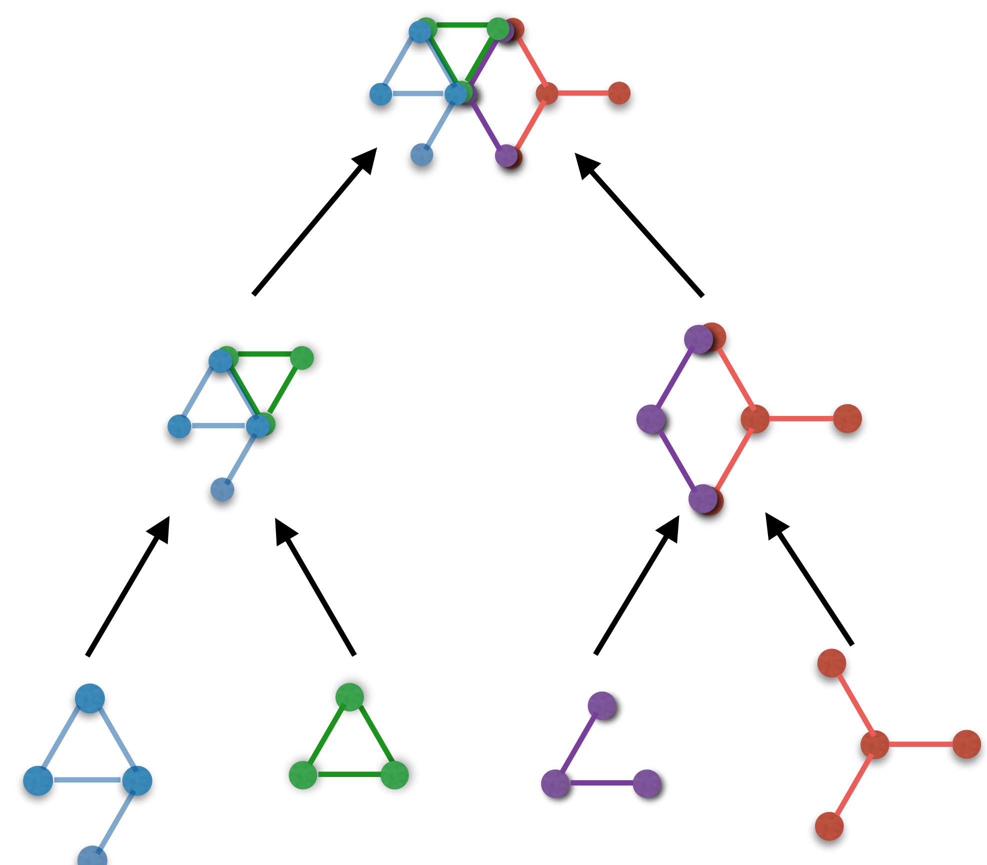
Distributed Divide and Conquer Skeleton

- The best performance on shared memory implementation

```
disDC :: (Trans a, Trans b)
      => Int                    — branching degree
      -> Places                 — ticket list
      -> (a -> Bool)            — trivial?
      -> (a -> b)               — solve
      -> (a -> [a]              — split
      -> (a -> [b] -> b)        — merge
      -> a                      — input
      -> b
```



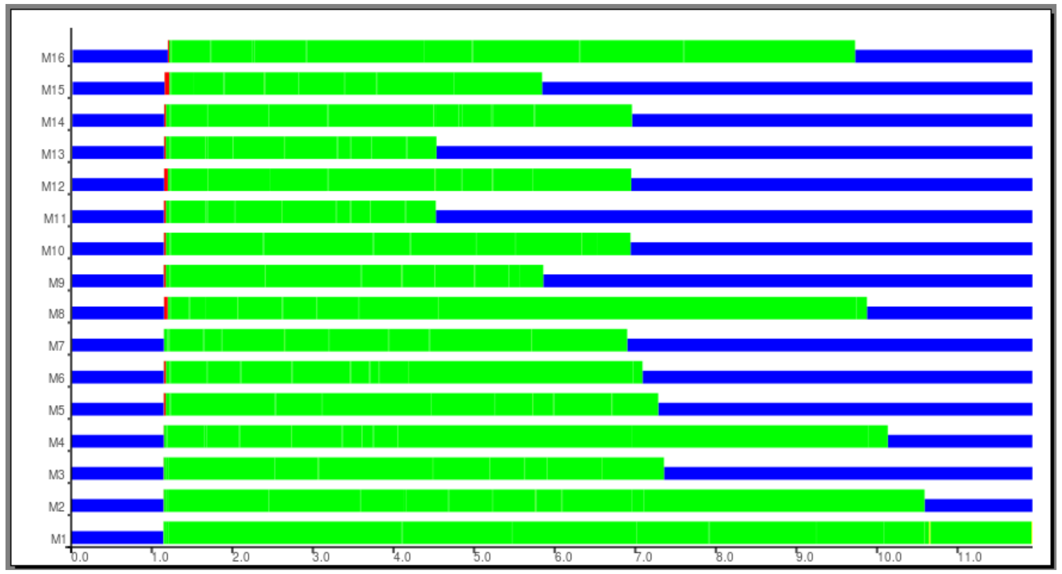
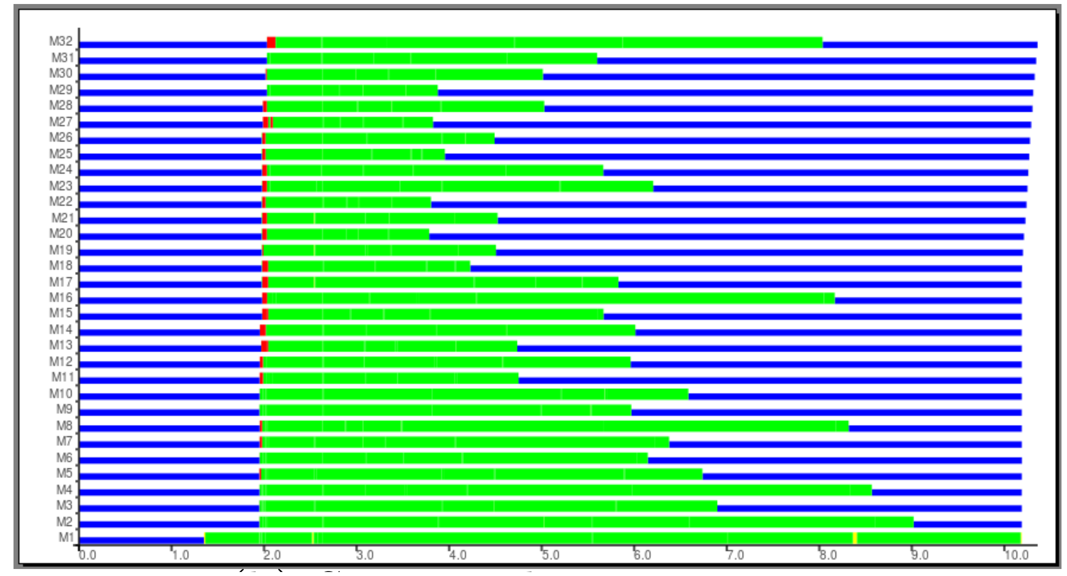
dividing the domain



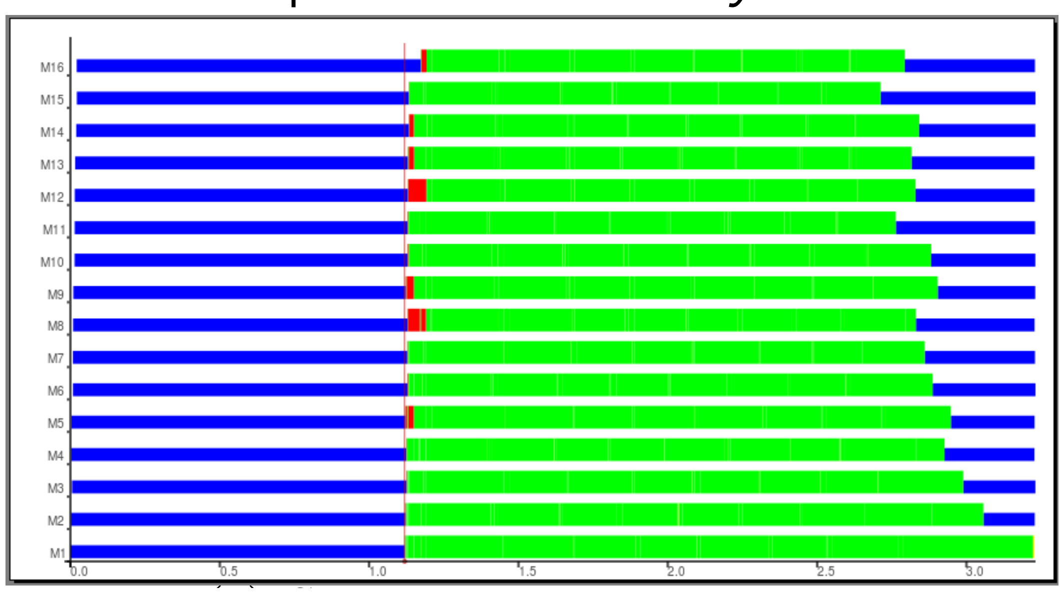
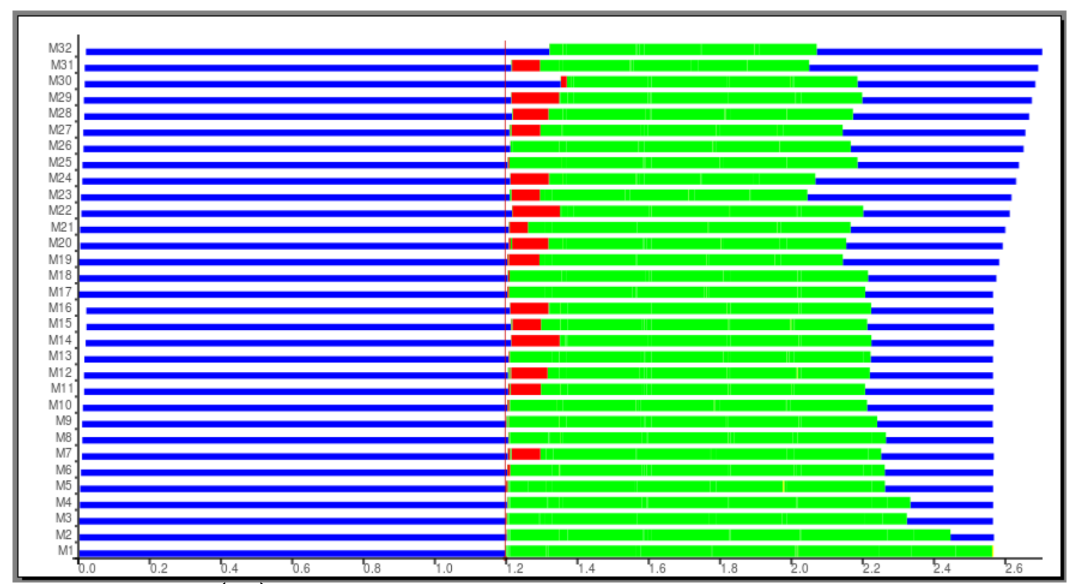
computing local JCNs and
reducing them into a global structure

Unbalanced Computation Load

- JCN Computation on *scission* dataset

- JCN Computation on *a synthetic* dataset

- JCN is a data dependent computation
- Divide and Conquer skeleton leads to an unbalance computation load

Over-decomposition

- Common strategy in computational science
  - decompose problem into N >> no. PEs
  - maintain work pool of tasks
  - scheduler "smooths over" differences from data dependencies
  - dynamic load balancing

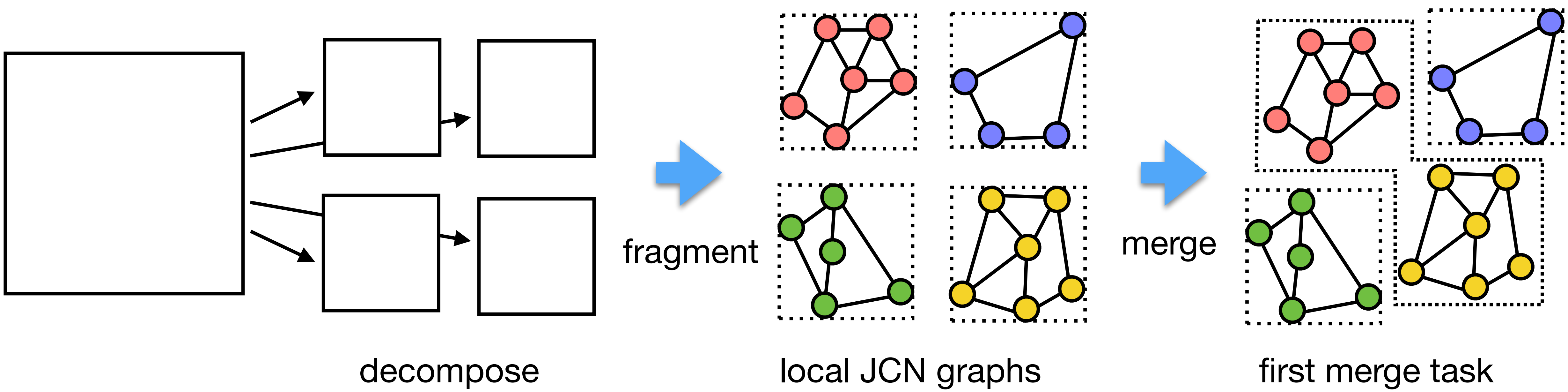
- Eden has a workpool skeleton

```
workpool :: (Trans t, Trans r)
         => Int              -- number of workers
         -> Int              -- prefetch
         -> (t -> r)         -- map function
         -> [t]              -- list of tasks
         -> [r]
```

- Could be used for both fragmentation and merger ..

Eden Workpool

- Eden library provides several extensions that integrate map and reduce functions
- BUT all allow merger of intermediate results *in arbitrary order*
- This is a problem (why?)

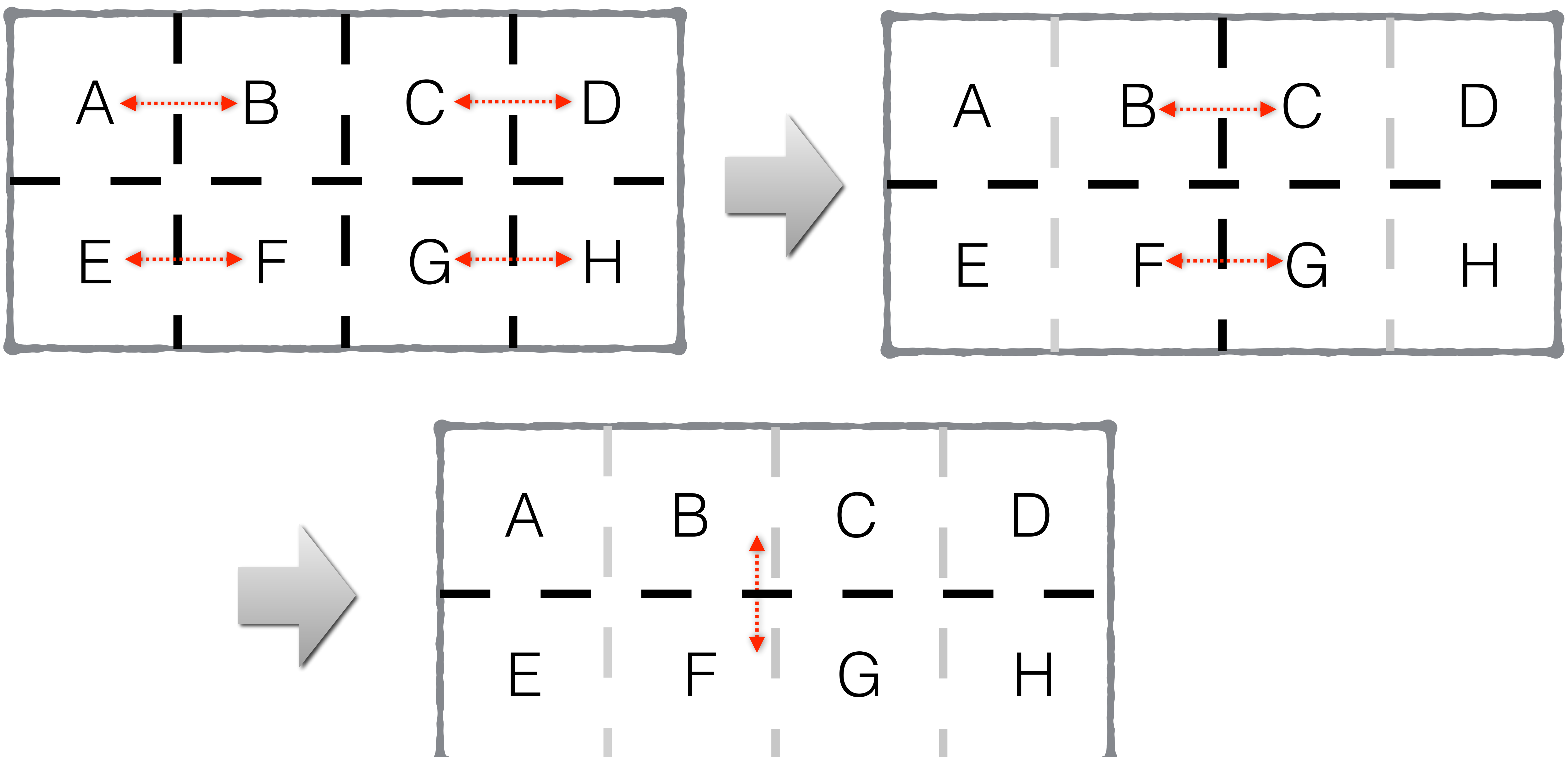


- Fortunately, strategies are just Eden functions ...
- ... so we can write our own.

Designing a new skeleton

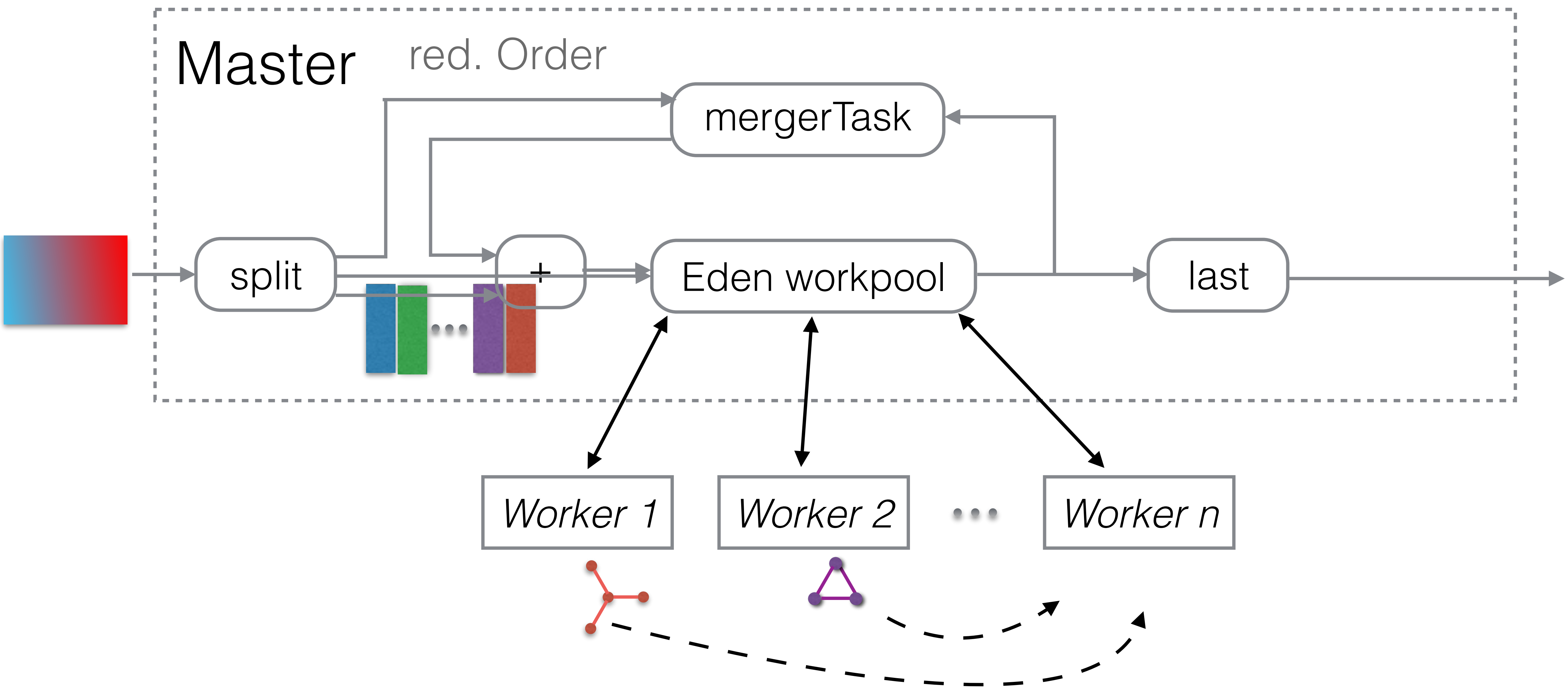
UNIVERSITY OF LEEDS

- Seek dynamic load balancing in *both* map *and* reduce stages
- Multi-level reduction of spatially-adjacent sub-problems
    - uses a *merger* function that matches intermediate results in a multi-level scheme

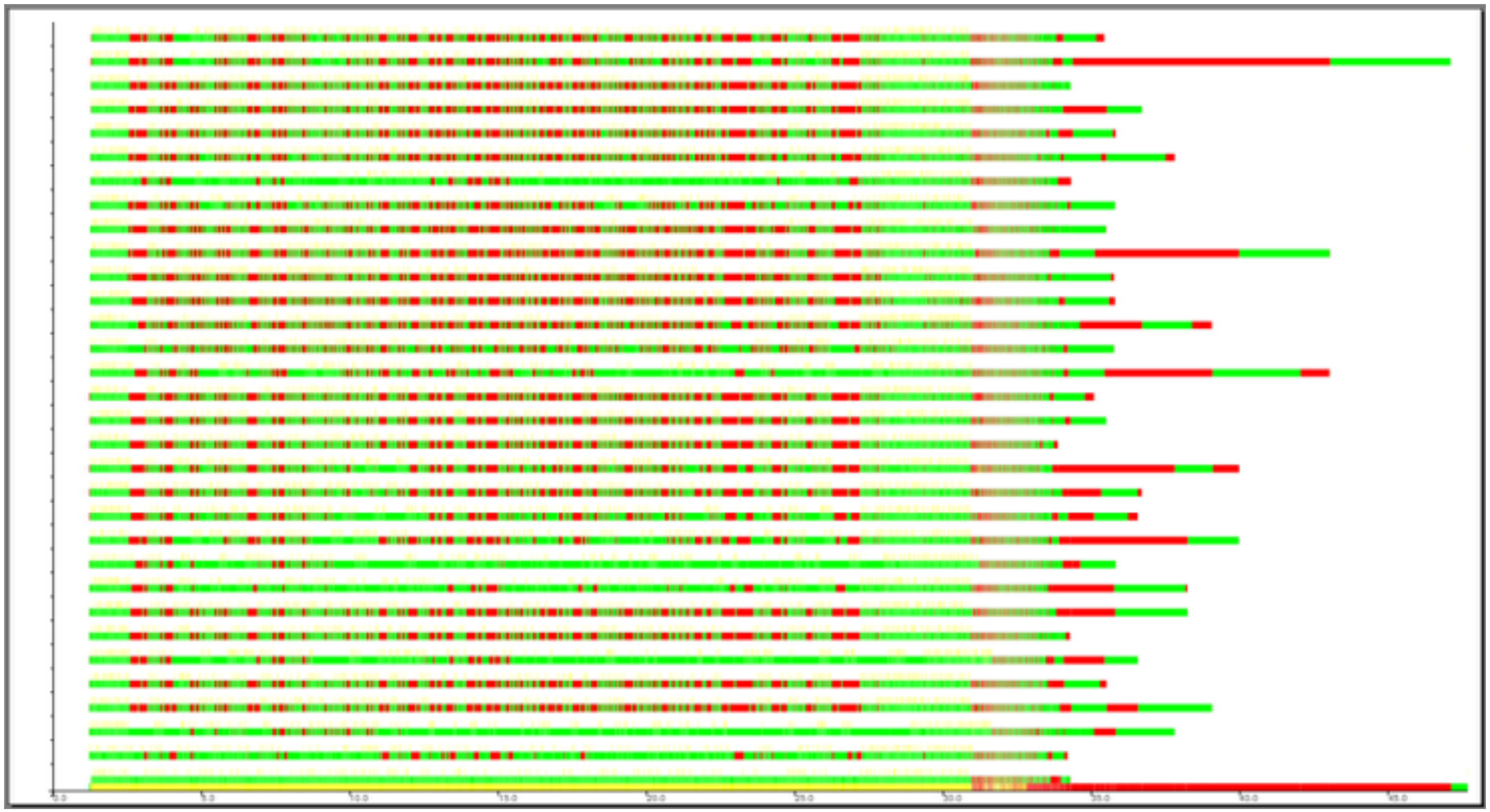


newWorkpool

- A new workpool skeleton for JCN-like computations

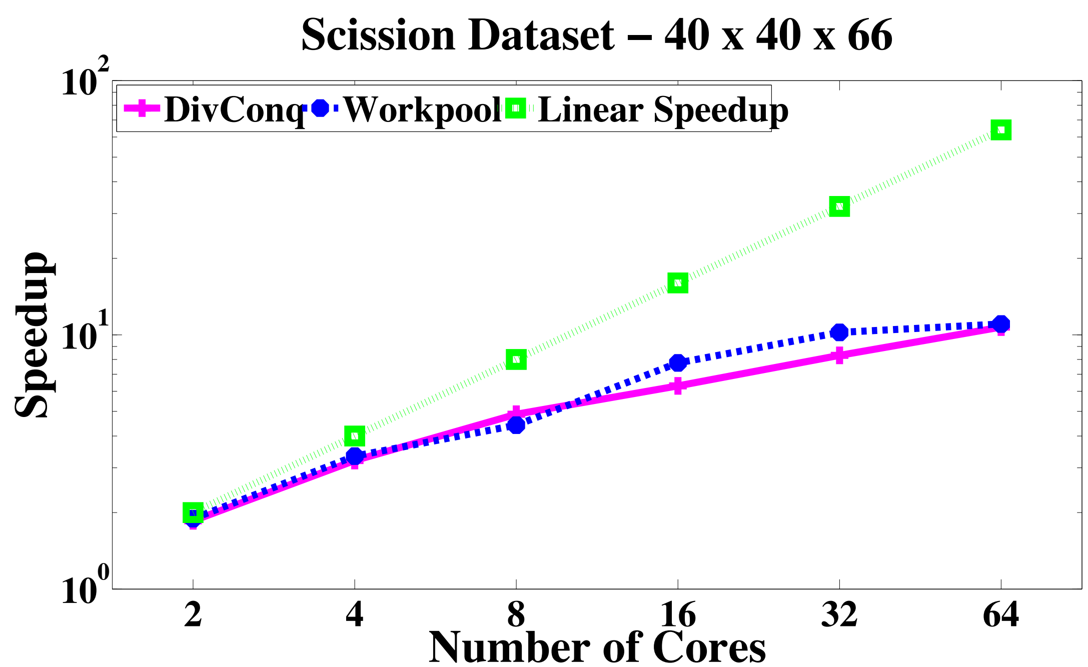


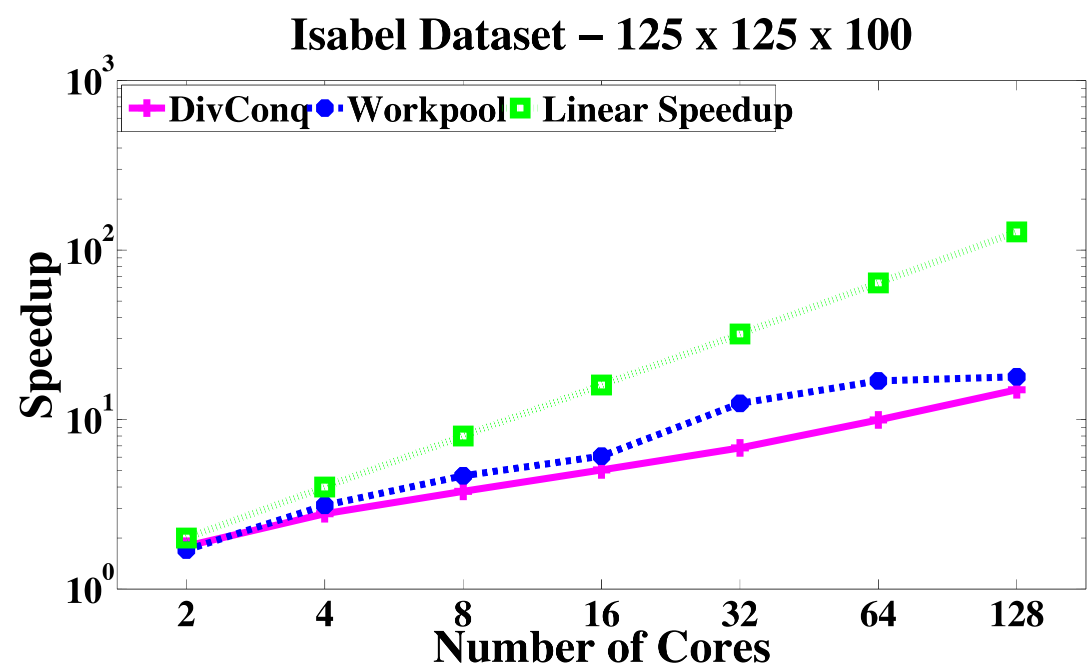
newWorkpool: run-time profile

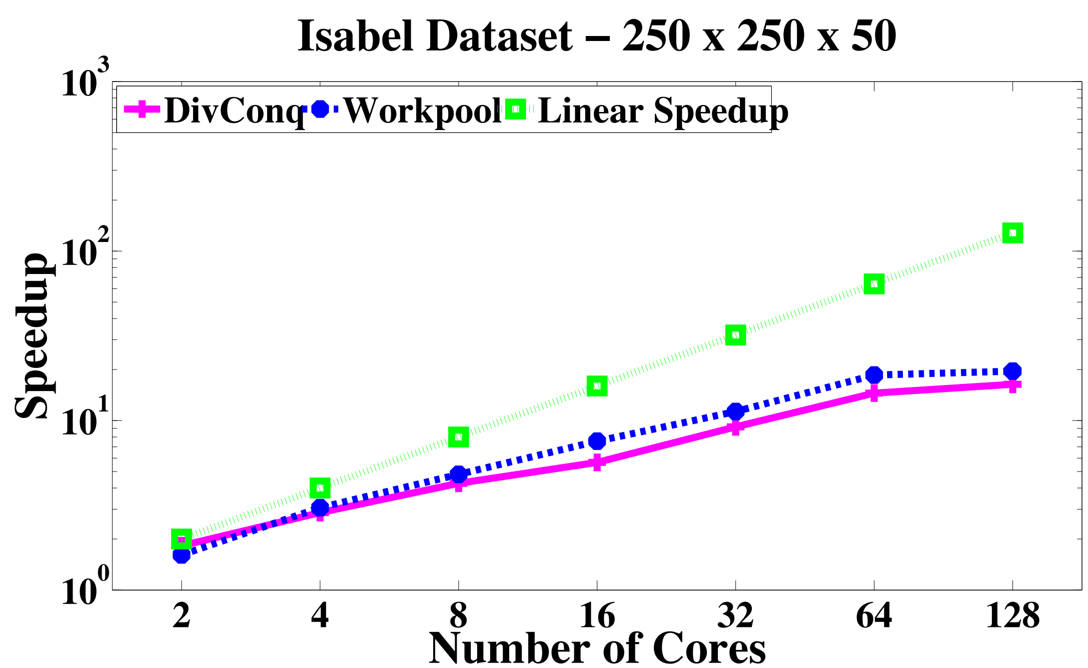


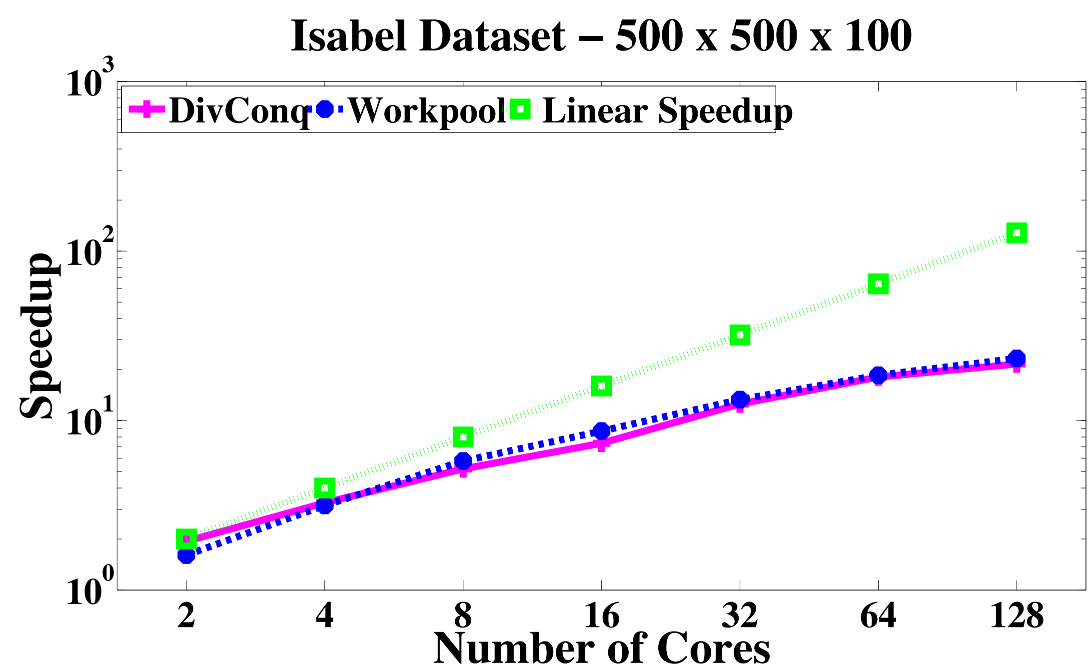
- JCN of Isabel dataset on 32 PEs

Performance of Distributed JCN



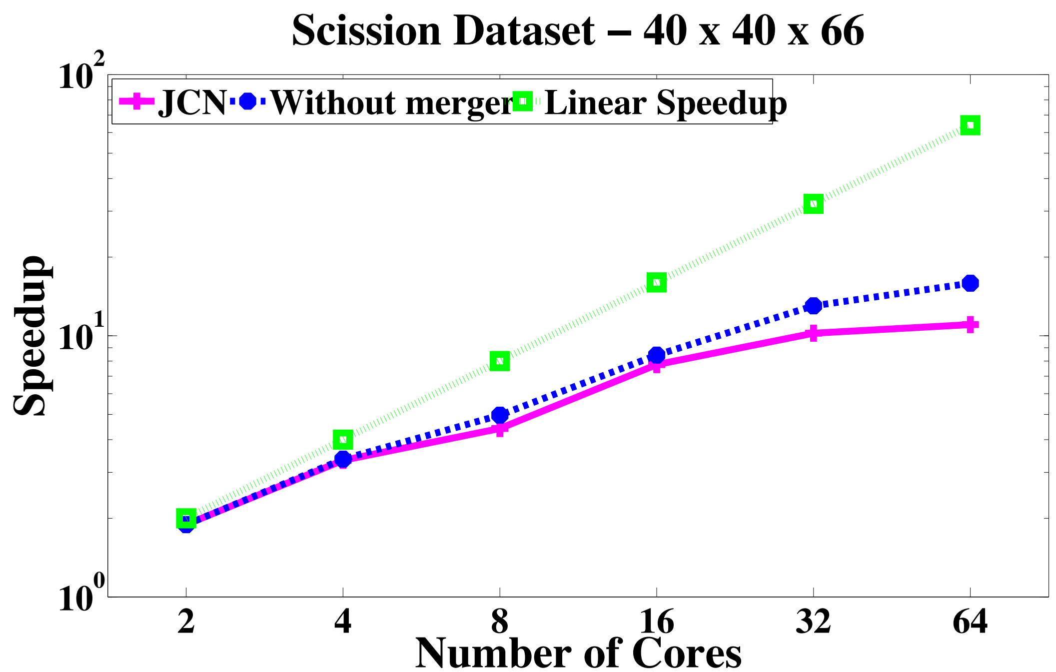


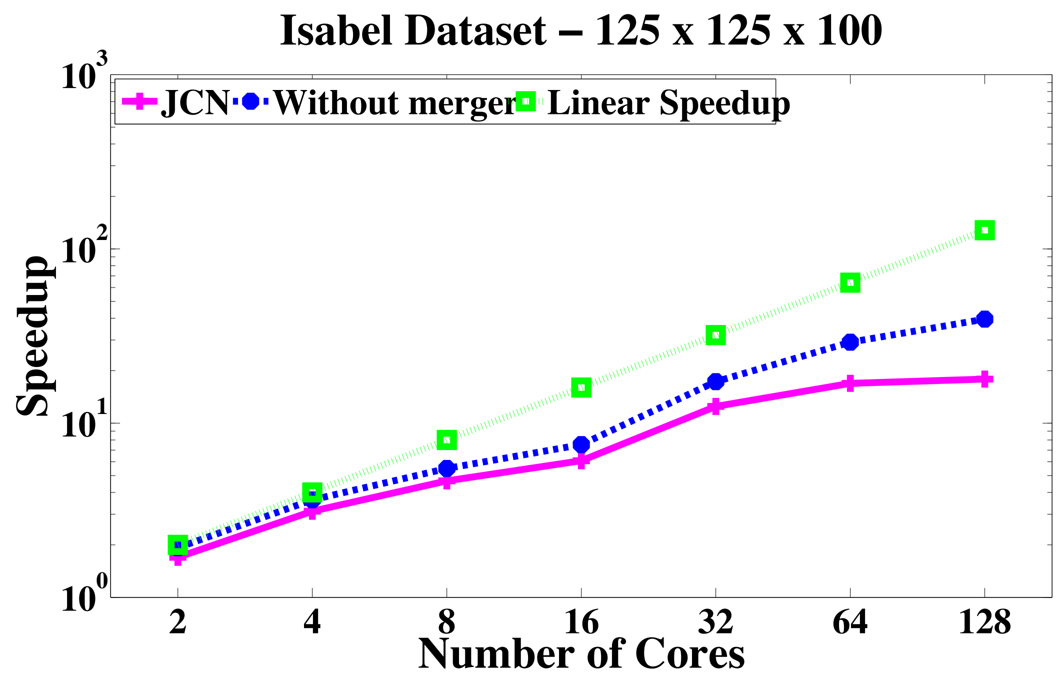


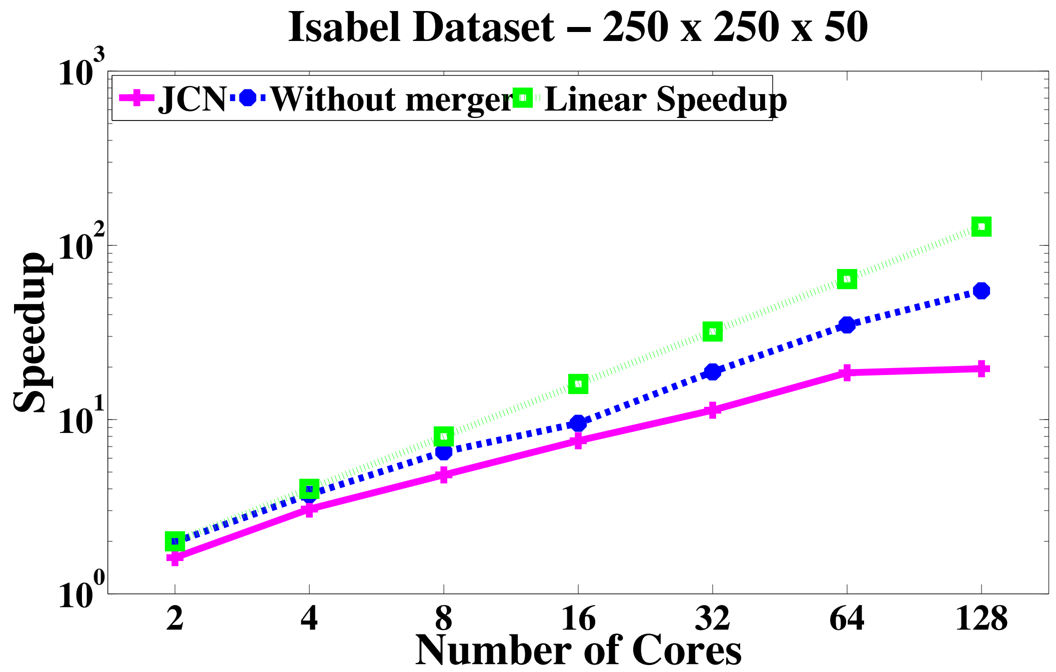


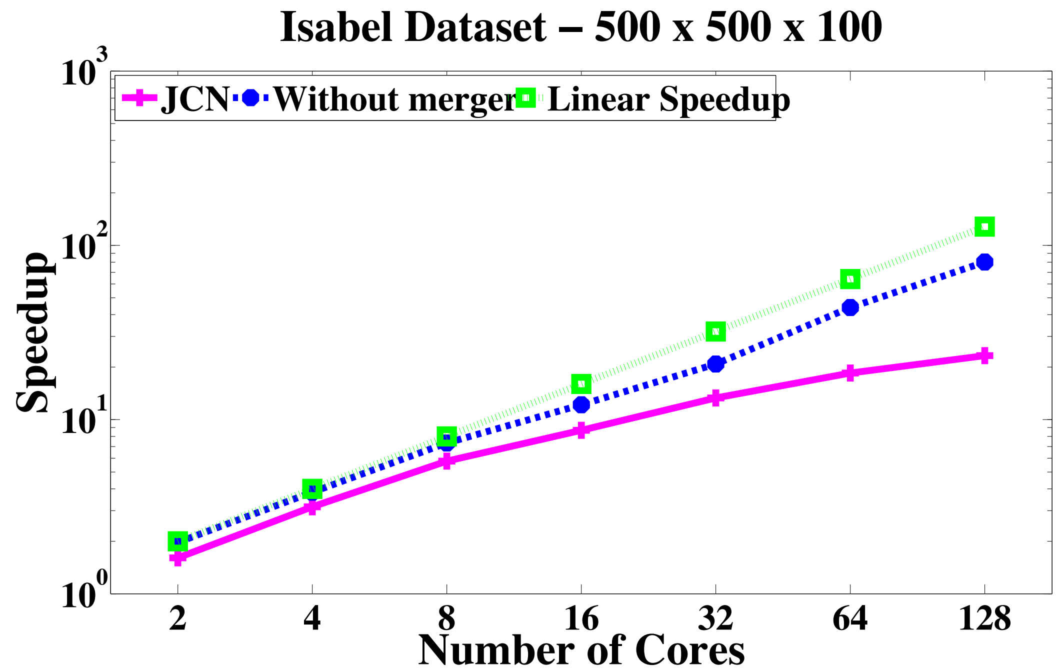
JCN without Merger

UNIVERSITY OF LEEDS



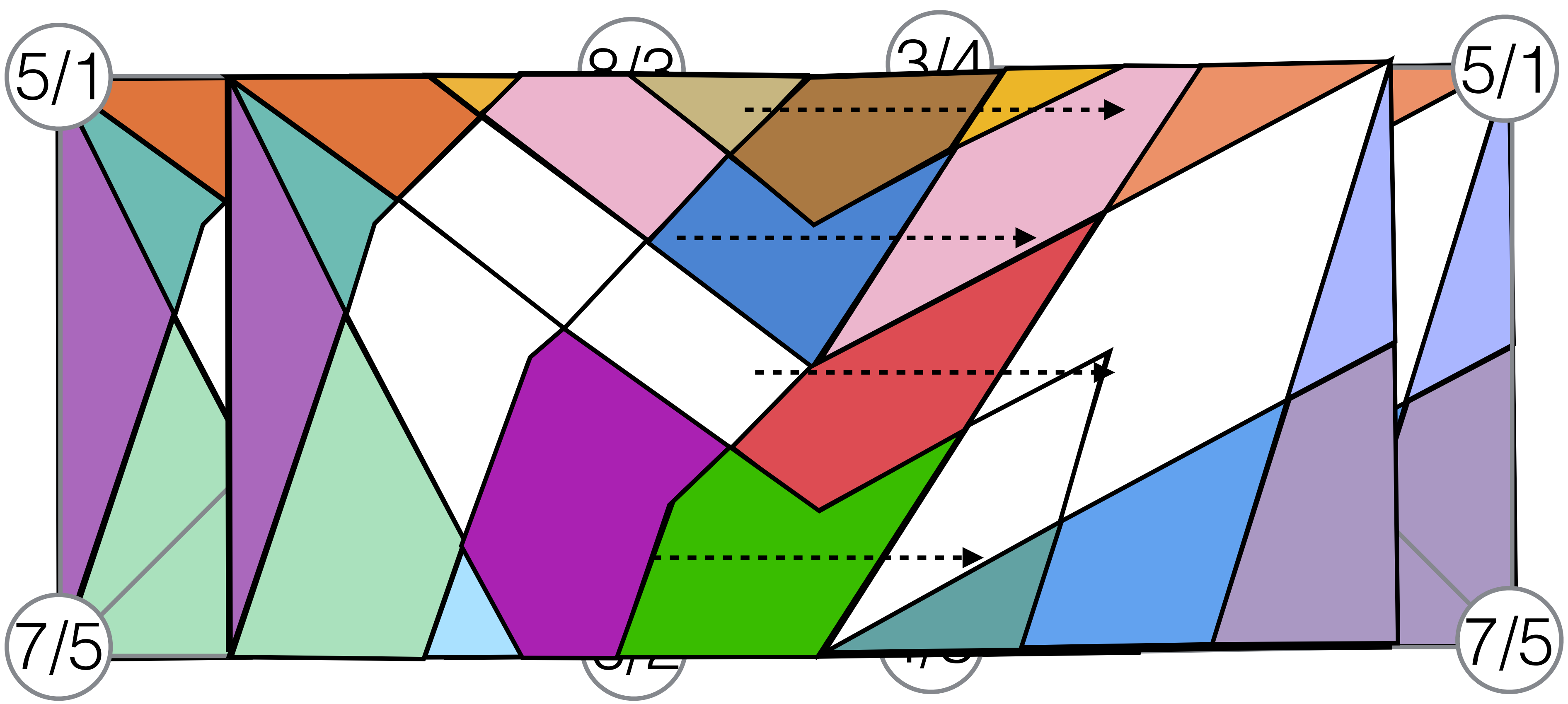






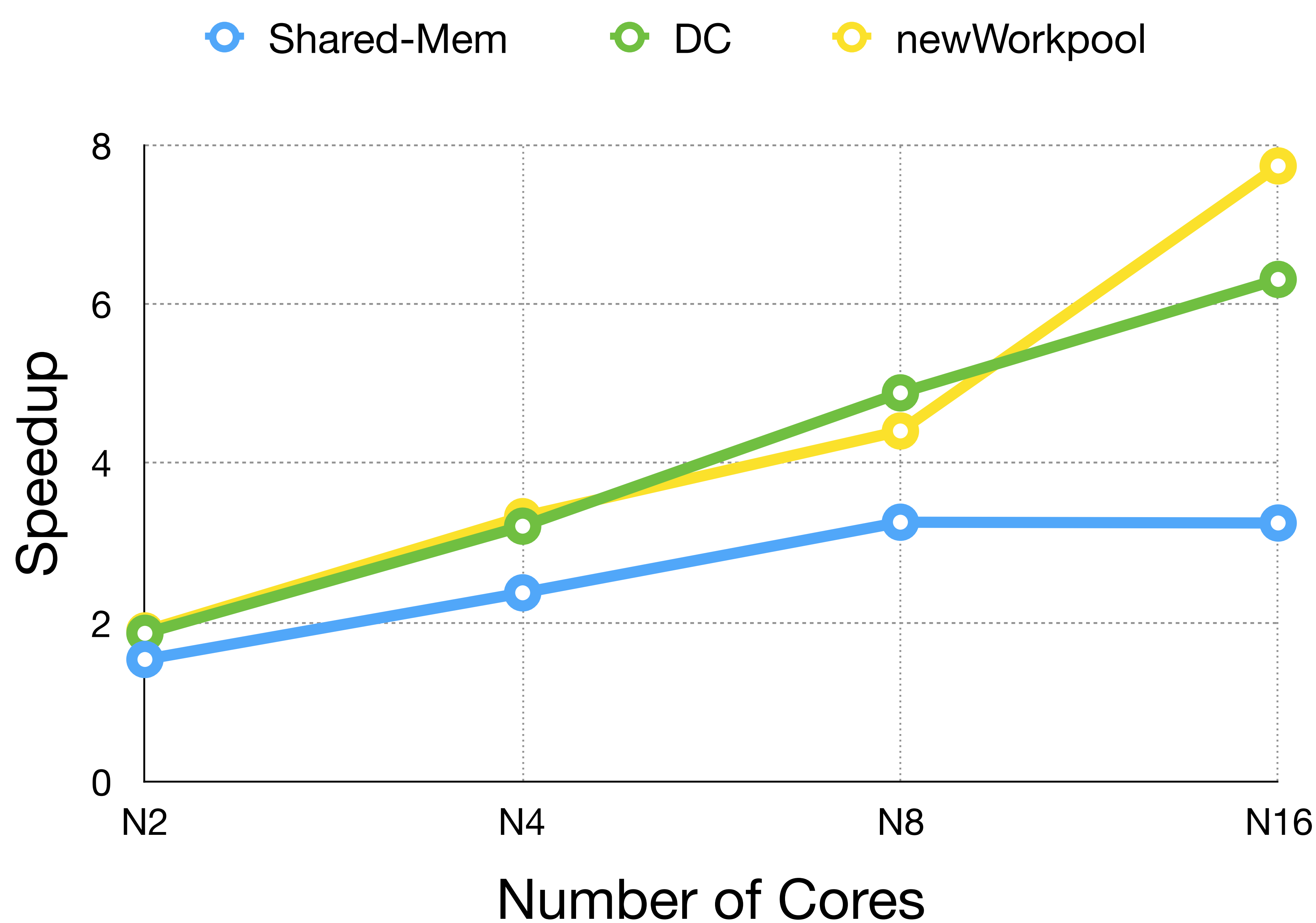
A distributed JCN representation

- **Observation:** merging JCNs involves only nodes on sub-problem boundary
- Alternative strategy:
  - distributed representation
  - incremental updates



- **Challenge**: over-decomposition gives [initial] subproblems with small interior graph

Shared & Distributed Mem Implementation



What we achieved / found ...

UNIVERSITY OF LEEDS

- Unexpected lessons on skeletons
  - skeletons can themselves be non-trivial functions
  - conveniently abstract for straightforward cases
  - *in*conveniently abstract for understanding performance
  - **can we have our cake and eat it too?**
  - **… are there sensible building blocks for skeletons?**

- Tooling still an issue
  - Peter Wortmann's enhanced profiling support invaluable - but further work needed (see Haskell Symposium 2013 paper)
  - hard to measure communication costs
  - profiling over 100's of cores
  - integrating IO into skeletons and distributed Haskell

- Contributions
  - scaled parallel JCN implementation from small to modest datasets
  - understood issues affecting further scaling
  - achieved some impact ...

Impact

- Haskell not yet on the cover of Nature ...
- ... but two *Physical Review C* papers are a start ...

PHYSICAL REVIEW C **90**, 054305 (2014)

Description of induced nuclear fission with Skyrme energy functionals: Static potential energy surfaces and fission fragment properties

N. Schunck,[1] D. Duke,[2] H. Carr,[2] and A. Knoll[3]
[1]*Physics Division, Lawrence Livermore National Laboratory, Livermore, California 94551, USA*
[2]*School of Computing, University of Leeds, Leeds, United Kingdom*
[3]*Argonne National Laboratory, Argonne, Illinois, USA*
(Received 11 November 2013; revised manuscript received 17 September 2014; published 6 November 2014)

Eighty years after its experimental discovery, a description of induced nuclear fission based solely on the interactions between neutrons and protons and quantum many-body methods still poses formidable challenges. The goal of this paper is to contribute to the development of a predictive microscopic framework for the accurate calculation of static properties of fission fragments for hot fission and thermal or slow neutrons. To this end, we focus on the ^{239}Pu(n,f) reaction and employ nuclear density functional theory with Skyrme energy densities. Potential energy surfaces are computed at the Hartree-Fock-Bogoliubov approximation with up to five collective

PHYSICAL REVIEW C **91**, 034327 (2015)

Description of induced nuclear fission with Skyrme energy functionals. II. Finite temperature effects

N. Schunck,[1] D. Duke,[2] and H. Carr[2]
[1]*Physics Division, Lawrence Livermore National Laboratory, Livermore, California 94551, USA*
[2]*School of Computing, University of Leeds, United Kingdom*
(Received 23 January 2015; published 25 March 2015)

Understanding the mechanisms of induced nuclear fission for a broad range of neutron energies could help resolve fundamental science issues, such as the formation of elements in the universe, but could have also a large impact on societal applications in energy production or nuclear waste management. The goal of this paper is to set up the foundations of a microscopic theory to study the static aspects of induced fission as a function of the excitation energy of the incident neutron, from thermal to fast neutrons. To account for the high excitation energy of the compound nucleus, we employ a statistical approach based on finite temperature nuclear density functional theory with Skyrme energy densities, which we benchmark on the ^{239}Pu(n,f) reaction. We compute

Broader issues

- So why isn't Haskell used in ..
  - .. computational science (Fortran, C/C++, Chapel, OpenCL, ...)
  - .. computer graphics & games (C++, C++, C++, C++, ...)
  - .. <insert your favourite application area>?
- Technology
  - technology only just mature
  - absence of a standard (<u>who</u> programs in Haskell'98 these days?!)
  - tools, esp. cost modelling and profiling
  - legacy code and inertia
  - different engineering challenges and <u>trade-offs</u> from success areas
- People issues
  - Your top C++ programmer leaves ... replace in days
  - Your top (HPC) Haskell programmer leaves ... replace when?
  - Project *risk* against likely *benefits*

What we achieved ...

- Contributions
  - scaled parallel JCN implementation from small to modest datasets
  - understood issues affecting further scaling
- Unexpected lessons on skeletons
  - skeletons can themselves be non-trivial functions
  - conveniently abstract for straightforward cases
  - *in*conveniently abstract for understanding performance
  - **can we have our cake and eat it too?**
  - **… are there sensible building blocks for skeletons?**
- Tooling still an issue
  - e.g. measuring communication costs, profiling over 100s of cores, IO
- Ongoing and future work
  - moving work towards GPU clusters
  - heterogeneous resources, complex memory hierarchy
  - in-situ processing
  - run on Tianhe-2 ?!

Future

- Challenges
  - Haskell / FP have made inroads in some sectors
  - Why? What do they excel at?
  - EDSLs - Obsidian, Repa, Accelerate ...
  - Heterogeneous resources, complex memory hierarchy
  - In-situ processing
  - Try running on Tianhe-2 ?!

- Current plans
  - Memory-sensitive representation
  - Games - one of the most performance-critical CS applications

Acknowledgements

UNIVERSITY OF LEEDS

- UK Engineering and Physical Sciences Research Council

EPSRC
Engineering and Physical Sciences Research Council

Multifield Extension of Topological Analysis, Grant EP/J013072/1

- Collaborators and contributors
    - Fouzhan Hosseini
    - Nicholas Schunck
    - Hamish Carr
    - Geng Zhao
    - Amit Chattopadhyay
    - Aaron Knoll
    - Jost Berthold
    - Thomas Horstmeyer
    - Hans-Wolfgang Loidl
    - Ben Lippmeier