Virtual Earthquake and seismology Research Community e-science environment in Europe
Project 283543 – FP7-INFRASTRUCTURES-2011-2 – www.verce.eu – info@verce.eu

DISPEL Introduction

Malcolm Atkinson

Data-Intensive Research Group

University of Edinburgh

VERCE @ University of Liverpool, 3 September 2012

Outline

- Data Intensive
  - What is it?
  - Why use it?
- DISPEL
  - What is it?
  - Why design it?
  - Is it different?
- A simple example
- Streaming matters
- Summary and Conclusions

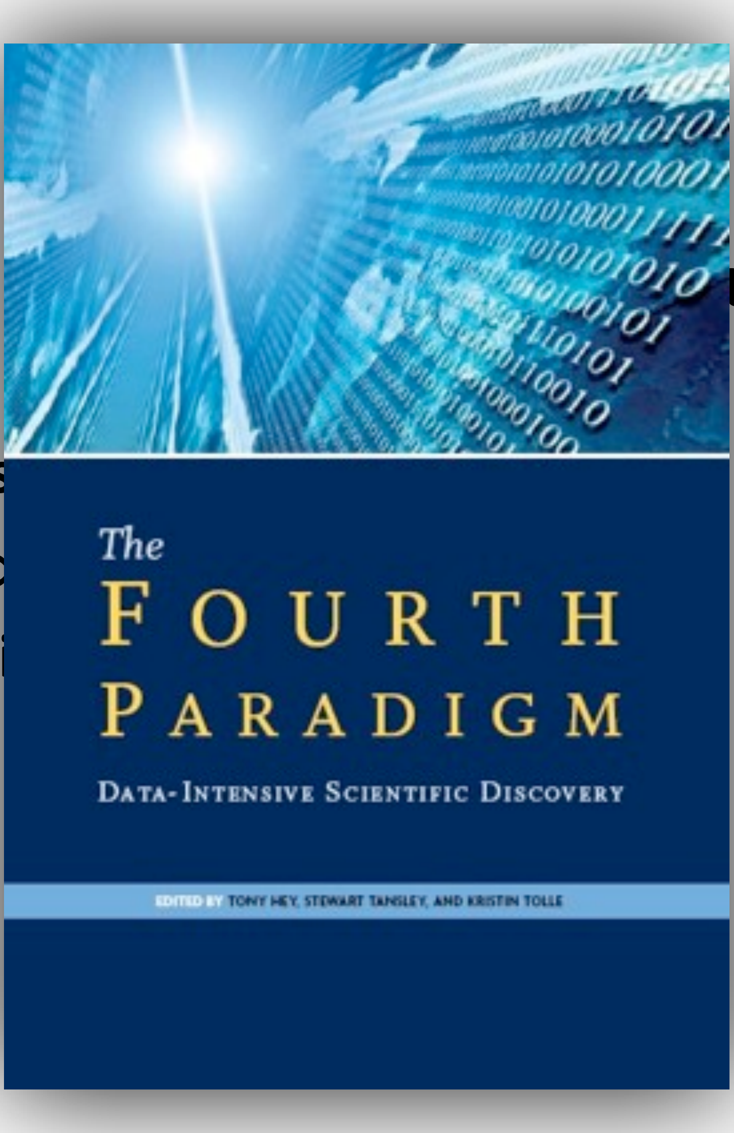
picture from Erica Salmon
Cornish Coast Path;
where I call home

Data-Intensive Thinking

Gray’s Laws of Data Engineering

Jim Gray:

- Scientific comp **ta**
- Need **scale-out**
- Take the **analys**
- Start with “**20 c**
- Go from “**worki**



The FOURTH PARADIGM

DATA-INTENSIVE SCIENTIFIC DISCOVERY

EDITED BY TONY HEY, STEWART TANSLEY, AND KRISTIN TOLLE

From: Alex Szalay, JHU

Defining “Data-Intensive”

- Generally
  - *A computational task is data-intensive if you have to think hard about an aspect of data handling to make progress*
    - distribution, permissions and rules of use, complexity, heterogeneity, rate of arrival, unstructured or changing structure, long tail of small and scattered instances, size of data, number of users
    - *invariably in combination*
- Quantitatively
  - The computation’s Amdahl numbers are close to 1
    - CPU operations : bits transferred in or out of memory
    - 1000 CPU operations : 1 I/O operation
  - Total volumes expensive to store
  - Total requests/unit time hard to accommodate
  - Data transport too slow or expensive

Data-Intensive Strategies 1

- Use commodity components and low power
  - So that you can afford a lot of them
  - Balanced for data-intensive work
  - Treat memory bandwidth as a scarce resource
- Data & computation as close together as possible
  - in the processor cache in fewest steps & not disrupted
- Work on small chunks of data
  - as small as logically possible
  - a column of a table
  - a row of a table
  - a file
  - data unbundled, in computational format & compressed
- Once data is close to a processor do all you can with it
  - multiple derivatives in one pass
  - pipelining
  - re-use of intermediate data, caching and forwarding
- *Use catalogues and indexes to avoid revisiting large-volumes of data randomly*

widely relevant

Data-Intensive Strategies 2

- Exploit very large scale parallelism and distribution
  - many subtasks at modest rate per task in large numbers
  - NOT tightly coupled parallelism!!!
  - distribution for availability, ownership & persistence
  - proximity to data sources or destinations for speed
  - good enough unbiased answers by careful sampling
- Replicate
  - for more parallelism and for durable persistence
- Most data WORM (Write Once Read Many)
  - or WORN (Write Once Read Never) - automatically eliminate or clean up
- Updates local and mostly append (mostly non-Transactional)
- Coordination & Catalogue DBs
  - distributed shared structures
  - just enough synchronisation
- Fine-grained local protection & authorisation
- Statistical and quantised accounting

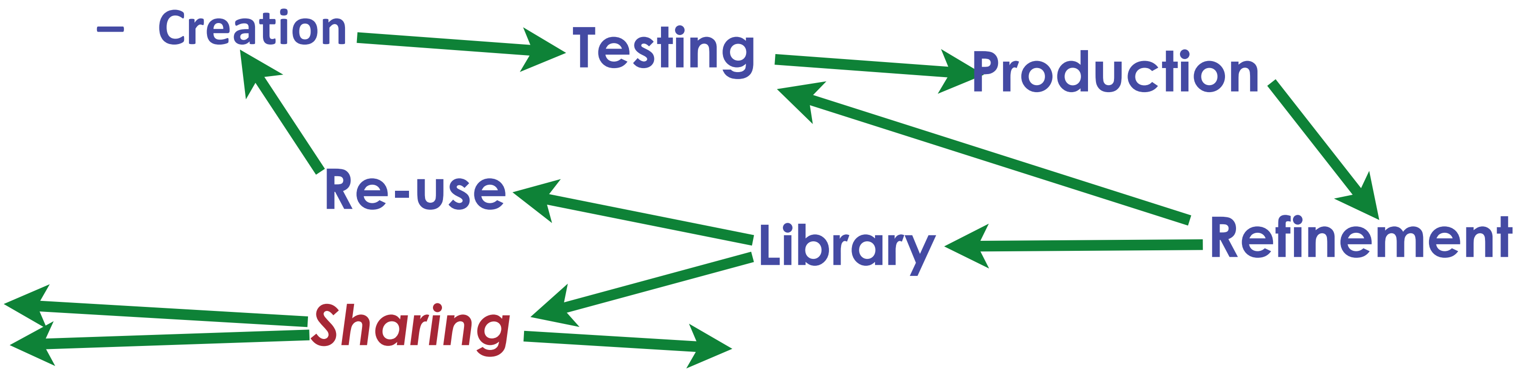
widely relevant

Data-Intensive Strategies 3

- High-level notations for describing methods /composing tasks
  - *with well-developed optimised transformations before execution*
  - query languages: SQL/AQL, (Xquery &SPARQL), ...
  - workflow languages: Kepler, Pegasus, DISPEL, ...
  - MapReduce: PigLatin, ZigZag, ...

widely relevant

- Providers + Community + User definition of (libraries of) tasks
  - your signal processing, geophysics & data-presentation steps
  - your existing code & preferred languages
- Support for the *workflow lifetime*: new research objects



- Creation

Testing

Production

Refinement

Library

Re-use

Sharing

DISPEL

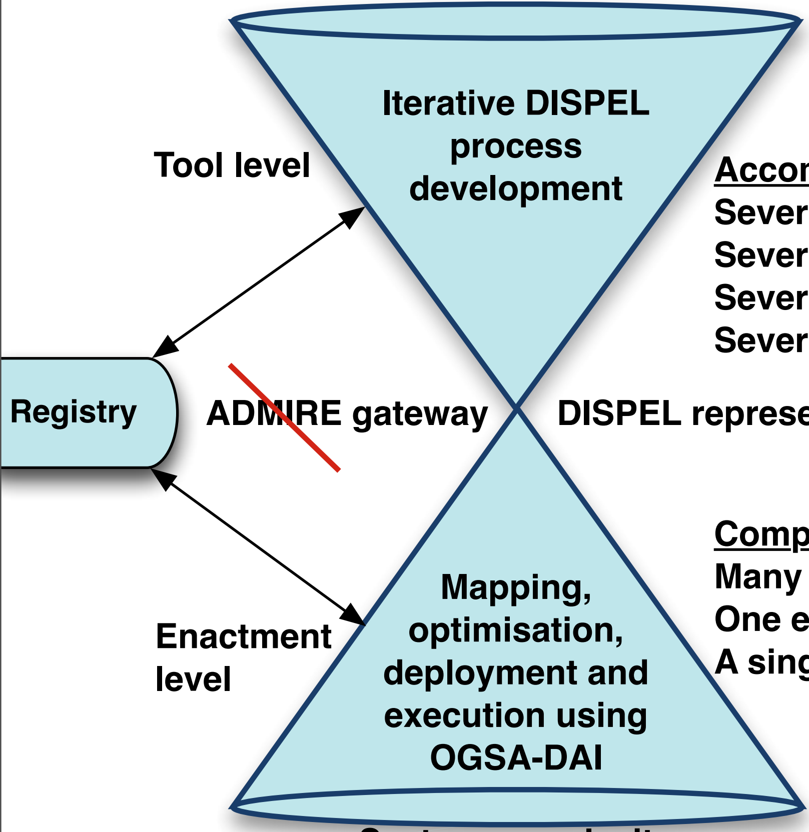
Data-Intensive Process Engineering Language

- A language for constructing data-flow graphs
  - Nodes are processing elements
  - Arcs are data-flow paths
- A language for generating data-flow patterns
  - Functions hide detail of graphs
  - Functions generate graphs
- A language for *discussing* data-flow engineering
  - Designed to be read and written by humans
  - As well as by programs
  - Supports validation and optimisation

designed to encourage data-intensive thinking

Domain experts: seismologists

User and application diversity


Iterative DISPEL process development
Tool level
Registry
~~ADMIRE~~ gateway
DISPEL representation
Enactment level
Mapping, optimisation, deployment and execution using OGSA-DAI

System complexity

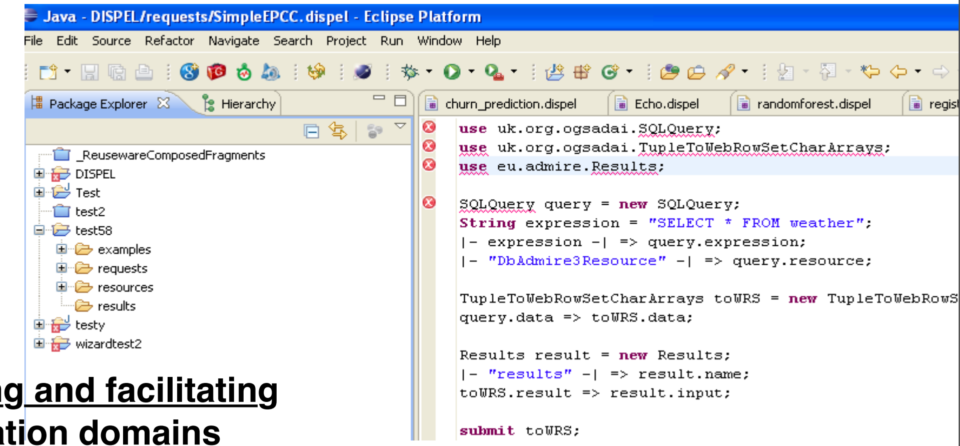
Accommodating and facilitating

Several application domains
Several tool sets
Several process representations
Several working practices

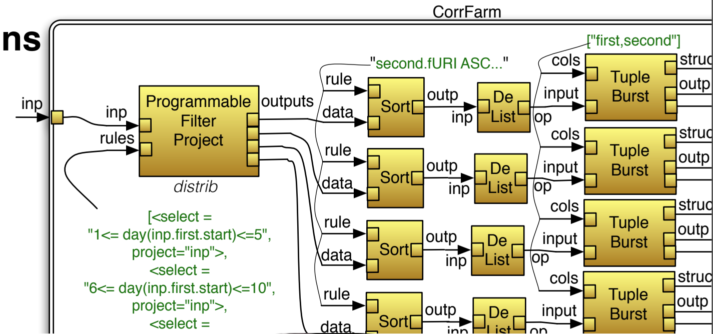
Data-analysis experts

Composing and providing

Many autonomous resources
One enactment mechanism
A single platform



```
use uk.org.ogsadai.SQLQuery;
use uk.org.ogsadai.TupleToWebRowSetCharArrays;
use eu.admire.Results;

SQLQuery query = new SQLQuery;
String expression = "SELECT * FROM weather";
|- expression -| => query.expression;
|- "DbAdmire3Resource" -| => query.resource;

TupleToWebRowSetCharArrays toWRS = new TupleToWebRowS
query.data => toWRS.data;

Results result = new Results;
|- "results" -| => result.name;
toWRS.result => result.input;

submit toWRS;
```

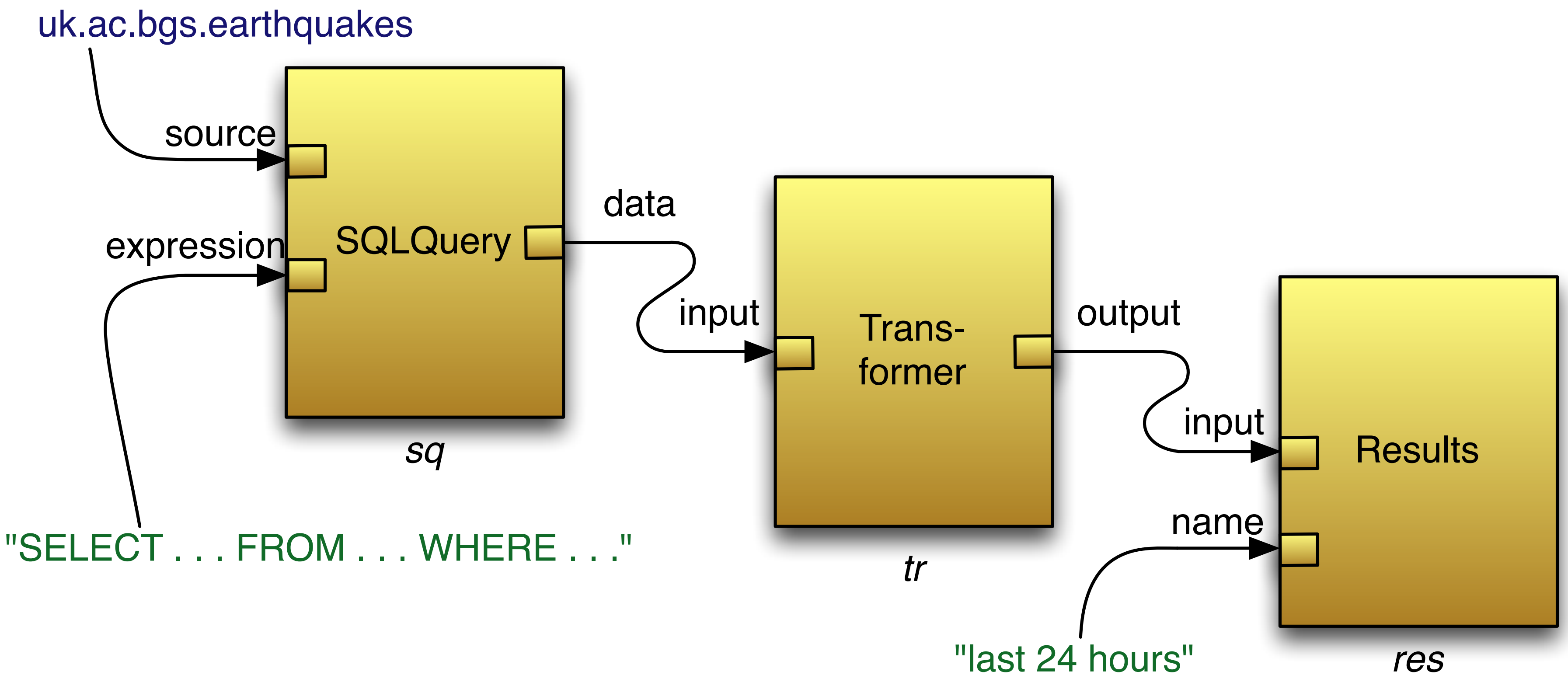


EDIM1

Data-intensive engineers

Iraklis will talk about the Registry later

A simple DISPEL graph

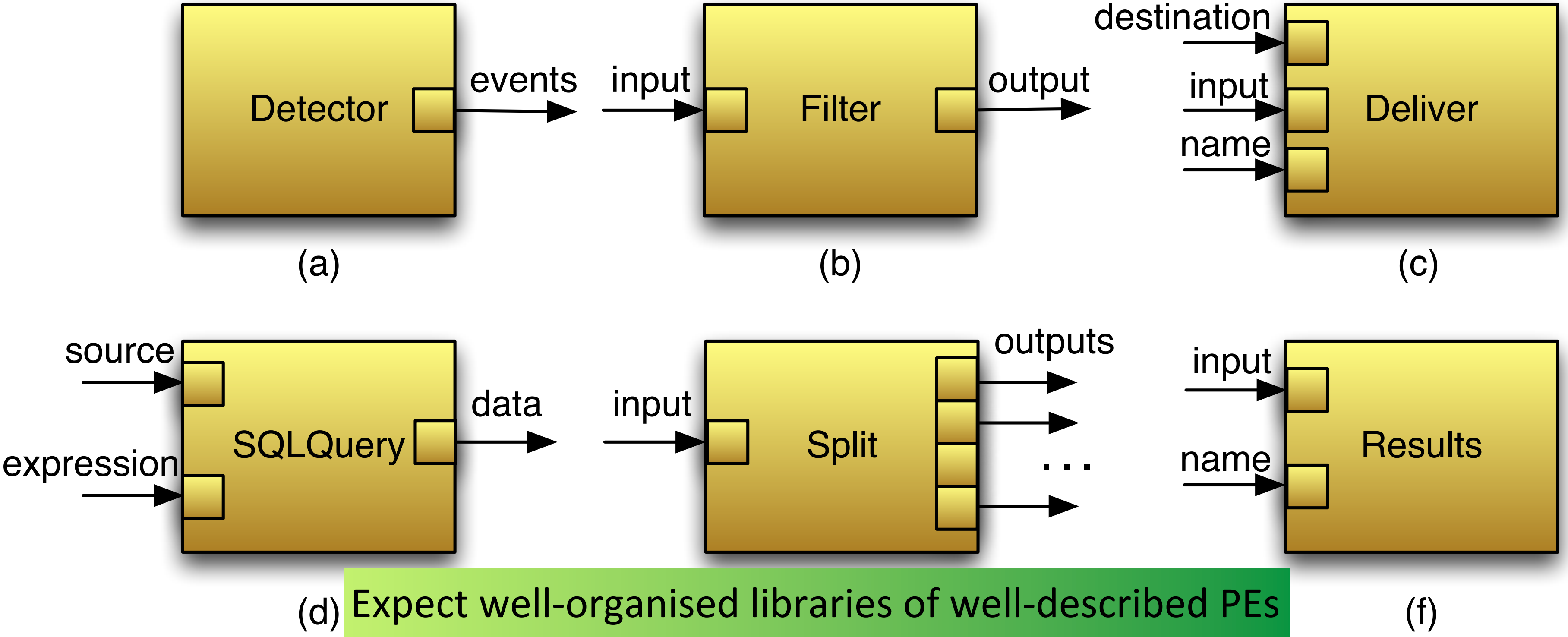


The DISPEL to Generate it

```
package book.examples.seismology {                //set working context
  use dispel.db.SQLQuery;                         //import PE SQLQuery
  use book.examples.seismo.Transform;             //import PE Transform
  use dispel.lang.Results;                        //import PE Results

  SQLQuery sq = new SQLQuery;                     // new instance of SQLQuery
  Transform tr = new Transform;                   // new instance of Transform
  Results res = new Results;                      // new instance of Results

  sq.data => tr.input;                            // set up data flow from sq to tr
  tr.output => res.input;                         // set up data flow from tr to res
  |- "uk.ac.bgs.earthquakes" -| => sq.source;     // URI of source of data
  |- "SELECT ... FROM ... WHERE ..." -| => sq.expression;    //query gets traces
  |- "last 24 hours" -| => res.name;              //name of results for user

  submit res;                                     // submit for enactment
}
```

Paul will talk about this next

Processing Elements

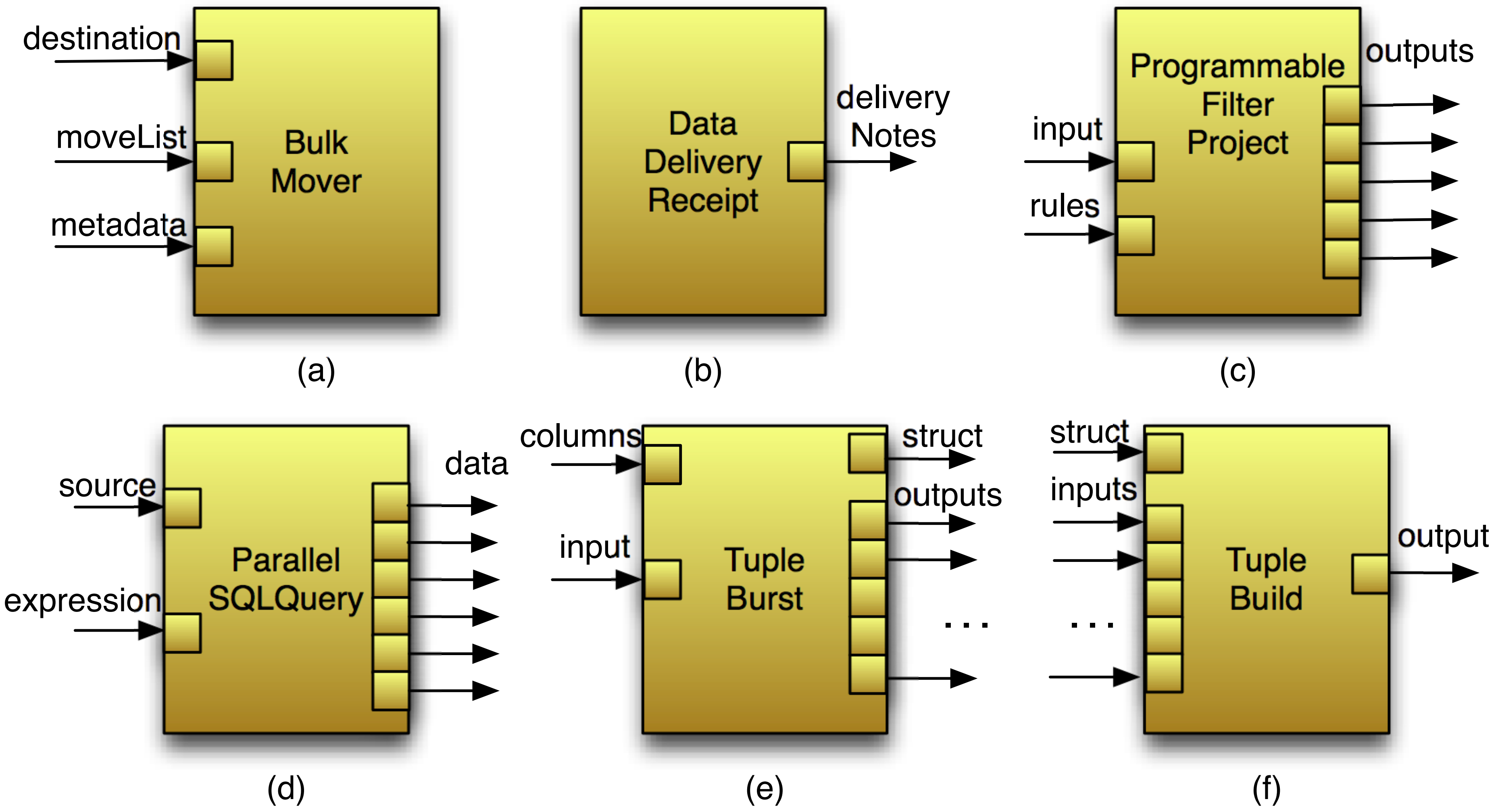


Expect well-organised libraries of well-described PEs

Description:

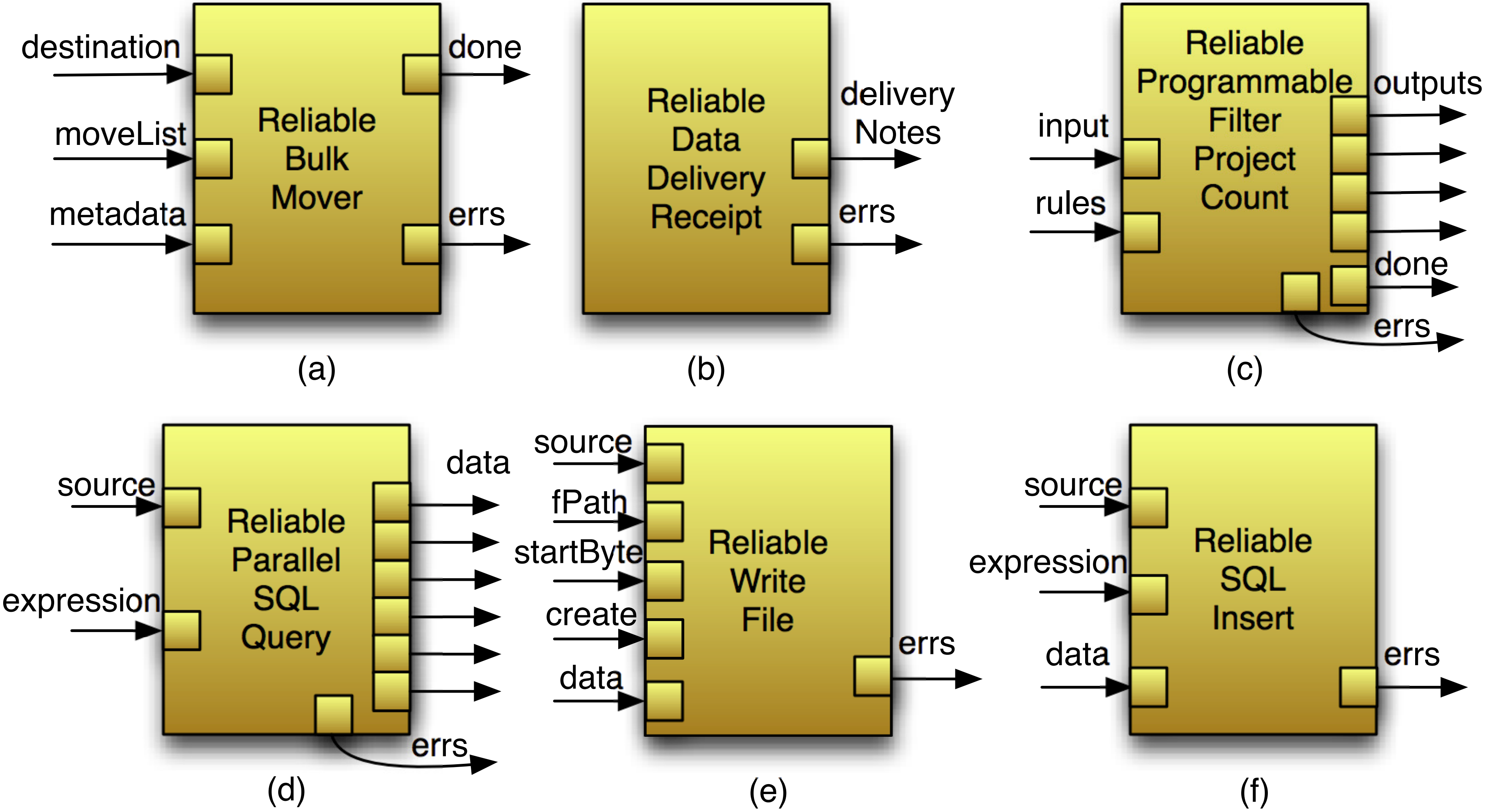
- names, inputs, outputs
- formats & meaning of each input and output
- auto-iteration behaviour, termination & errors
- optimisation properties
- use, relationships and logical properties

Enhanced Processing Elements



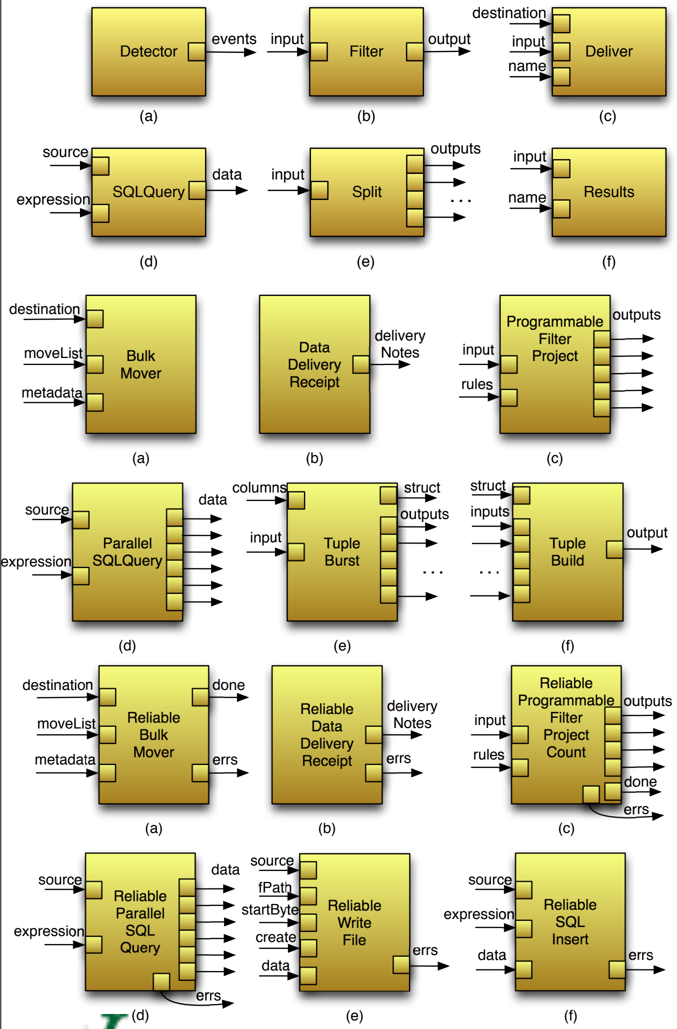
not all implemented yet

Advanced Processing Elements



not all implemented yet

Processing Element Libraries



- Good libraries are needed
  - -Generic libraries
  - -Domain specific libraries
- KNMI building VERCE libraries
  - -*Hard work today*
- Need harnesses & tools
  - -To quickly wrap existing libraries
  - -To quickly deploy new algorithms

Amy builds these

Alessandro and Luca will talk about these and this work tomorrow

Functions

- Algorithms to generate graphs
  - parametric variation
  - patterns
  - parameters
  - subgraphs

Expect well-organised libraries of well-described Functions

Description:

- names, type signature, description, ...

- Abstraction and Optimisation
  - smart methods for common patterns
  - hiding pattern implementation for stability
  - late evaluation would permit contextual

we return to functions after Paul's talk

Enactment Model

1. DISPEL language processing
   - 1.1. Validation & import from registry
   - 1.2. Format & meaning mis-match handling
   - 1.3. Interpretation to generate graph
2. Graph optimisation & mapping
   - 2.1. Re-ordering & parallelisation
   - 2.2. Identification of where to do the work
   - 2.3. Selection of PE implementations & instances
   - 2.4. Partitioning into co-located subgraphs
3. Deployment & initialisation
4. Execution, Monitoring & Clean up

Preparing to do the data processing

Doing the data processing

Amy will talk about this later

DAGMan

Condor
High Throughput Computing

VisTrails

meandre

swift

Kepler

MOTEUR

pegasus

Taverna

Trident

TRIANA

Microsoft Research

ASKALON

KNIME

Ian J. Taylor, Ewa Deelman, Dennis B. Gannon, and Matthew Shields.
Workflows for e-Science: Scientific Workflows for Grids. Springer London, 2007.

DISPEL is Different 1

- Spanning Distributed *Independent* Hosts
  - Fragments of one workflow can run in different regimes
  - Different security models
  - Different file systems
  - Different DBMS
  - Different Operating Systems
  - Different DISPEL implementations
- Agnostic about Size & Scale
  - Processing Elements of any size
  - Data values in streams of any size
  - Streams of any length
  - Graphs of any size

Can process data much larger than local storage

DISPEL is Different 2

- Patterns & Pattern Composition
  - Functions define & generate patterns
  - Higher-order functions compose patterns
  - Functions can be refined to optimise
- Component-Description Driven
  - Rich description of components
  - Capturing logical properties
  - Collecting component-builders’ hints
- Restricted language for workflow longevity
  - Only hints and *no* WF-definition time concrete mappings
  - Late mapping potentially permits optimisation, *for the system as it is at execution time* - usually much different from definition time!

Streaming

Why Stream

- Couple data-processing steps
  - locally or across networks
- Handle continuous or very large data
  - incrementally by auto-iteration on units
  - for any scale of unit
- Provide an opportunity to cancel
  - as soon as partial results show problems
- Accelerated processing
  - overlapping processing & transmission steps
  - potentially multiple steps in same cache

G H I J K L M N

F

E

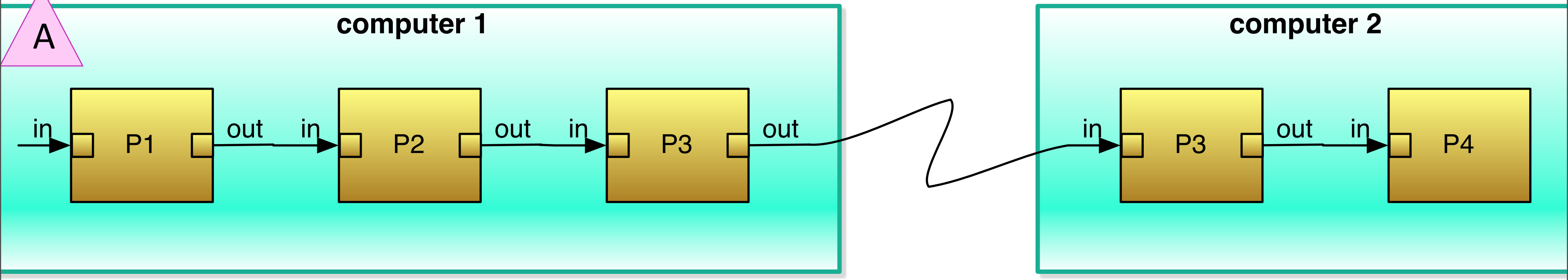
D

C

B

A

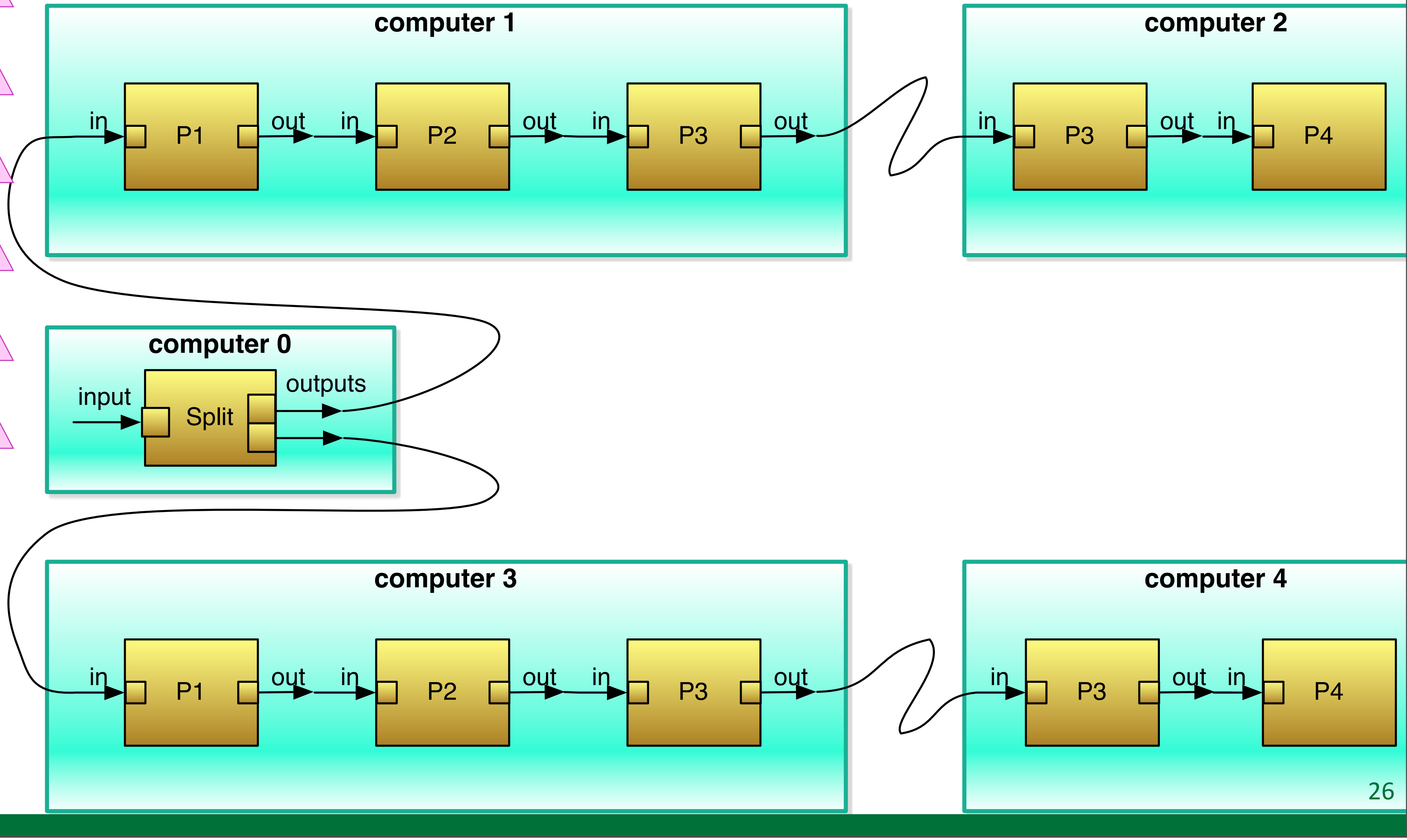
A Single Stream



H I J K L M N O P

G F E D C B A

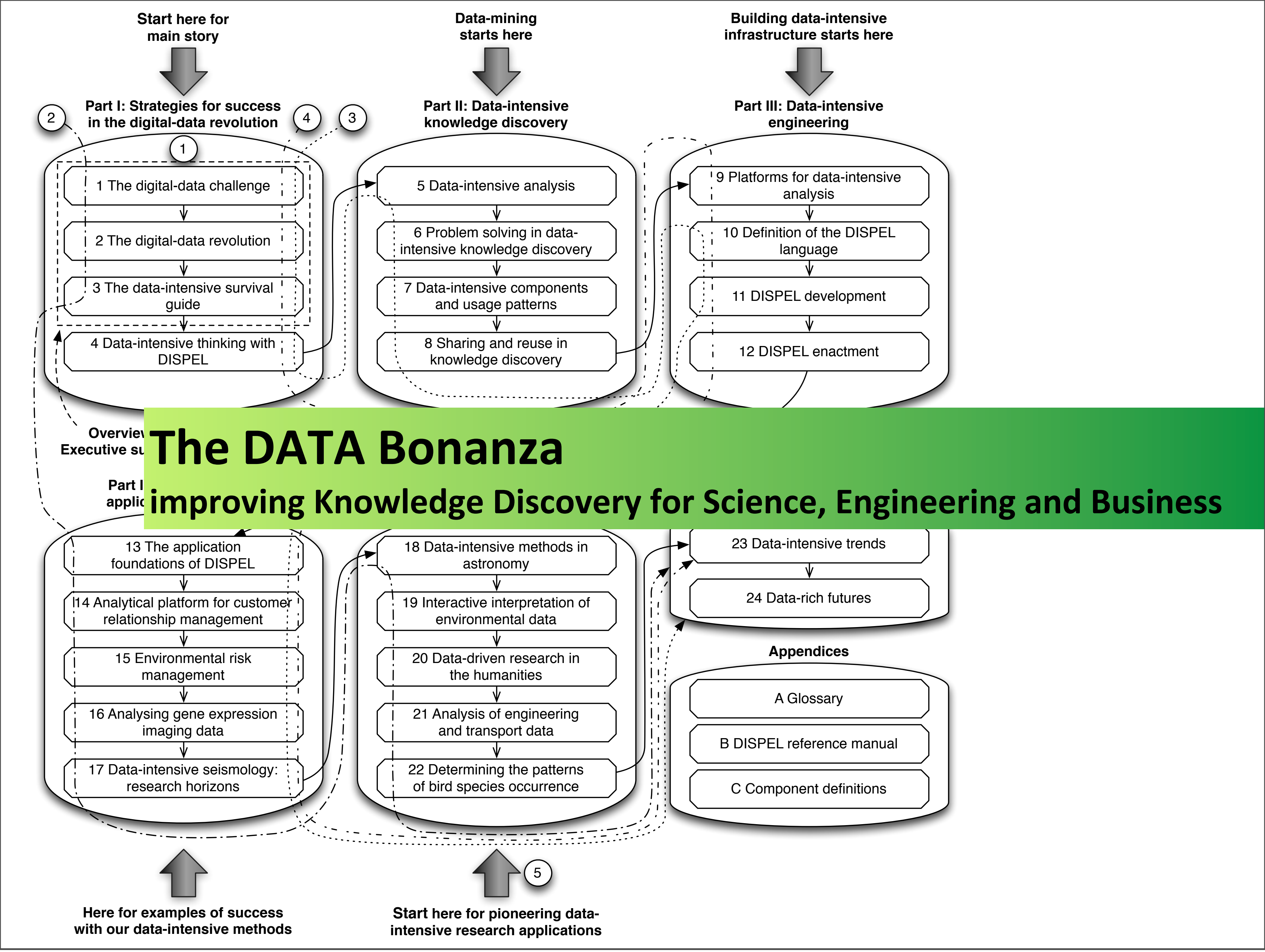
Parallel Streams



Summary and Conclusions

The DATA Bonanza

improving Knowledge Discovery for Science, Engineering and Business



Start here for main story

Data-mining starts here

Building data-intensive infrastructure starts here

Part I: Strategies for success in the digital-data revolution

1 The digital-data challenge

2 The digital-data revolution

3 The data-intensive survival guide

4 Data-intensive thinking with DISPEL

Part II: Data-intensive knowledge discovery

5 Data-intensive analysis

6 Problem solving in data-intensive knowledge discovery

7 Data-intensive components and usage patterns

8 Sharing and reuse in knowledge discovery

Part III: Data-intensive engineering

9 Platforms for data-intensive analysis

10 Definition of the DISPEL language

11 DISPEL development

12 DISPEL enactment

Overvie

Executive su

Part I

applic

13 The application foundations of DISPEL

14 Analytical platform for customer relationship management

15 Environmental risk management

16 Analysing gene expression imaging data

17 Data-intensive seismology: research horizons

18 Data-intensive methods in astronomy

19 Interactive interpretation of environmental data

20 Data-driven research in the humanities

21 Analysis of engineering and transport data

22 Determining the patterns of bird species occurrence

23 Data-intensive trends

24 Data-rich futures

Appendices

A Glossary

B DISPEL reference manual

C Component definitions

Here for examples of success with our data-intensive methods

Start here for pioneering data-intensive research applications

Summary

- DISPEL is an experimental data-intensive language
  - draws on workflows & database query internals
  - auto-iteration over values flowing through connections
  - agnostic about value sizes - implementation challenge
  - controlled access to system information
  - optimisation based on description & operation
  - distributed termination protocol
- Several years of experience
  - seven different application domains
- Differences
  - functional pattern handling
  - multi-scale streams
  - restricted information to permit platform evolution
- Status
  - two implementations: to OGSA-DAI & to Java
  - much still to do to fully explore the ideas

Today’s Programme

• Introduction	Malcolm Atkinson ✓
• DISPEL language	Paul Martin
• DISPEL enactment	Amy Krause
• DISPEL functions	Malcolm Atkinson
• Registry	Iraklis Klampanos

?

www.verce.eu

research.nesc.ac.uk/node/828

www.ogsadai.org.uk

epcc

omii-uk

www.omii.ac.uk

VERCE
www.verce.eu

VERCE

OGSA-DAI

Picture
composition
by
Luke Humphry
based on prior
art by Frans Hals