Numerical Weather Prediction

Facing the Future with C++

T. Quintino, F. Rathgeber, W. Deconinck,
B. Raoult, M. Fuentes
ECMWF

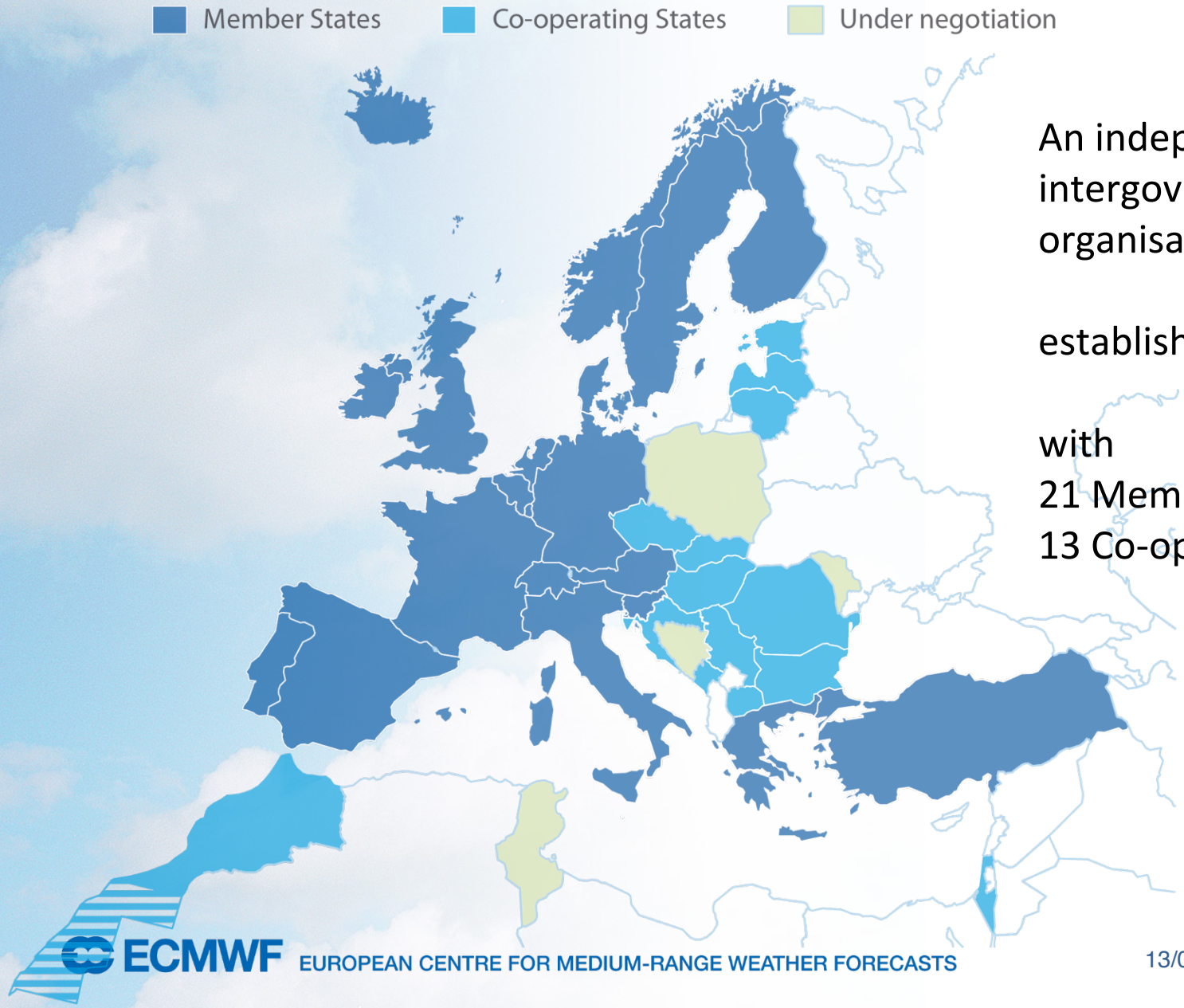
C++Now 2015, May 13 2015

Overview

- ECMWF
- Numerical Weather Prediction
- Challenges Ahead
- Leaning from the Past
- Preparing the Future

ECMWF

ECMWF



An independent intergovernmental organisation

established in 1975

with
21 Member States
13 Co-operating States

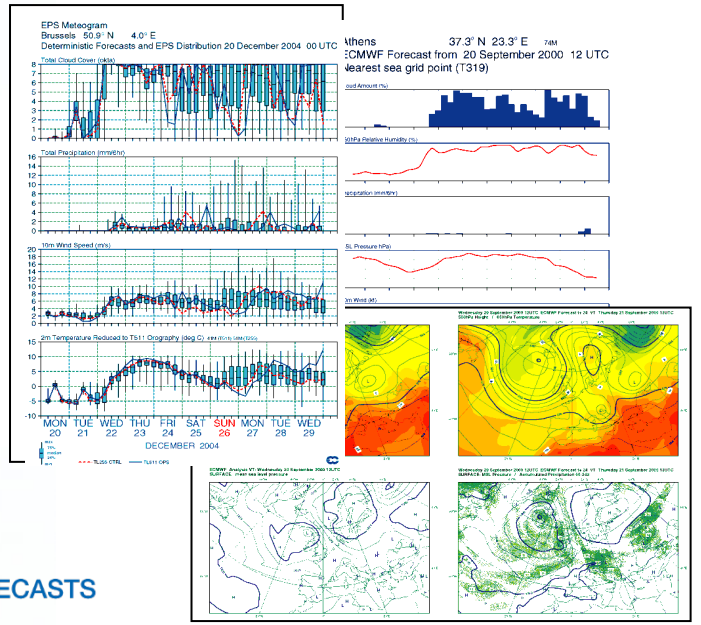
Who are we and what do we do?

Weather Forecasts	We produce global weather forecasts
Medium-Range	Up to 15 days ahead. Our products also include monthly and seasonal forecasts and we collect and store meteorological data.

Reading, United Kingdom

What do we have to achieve this?

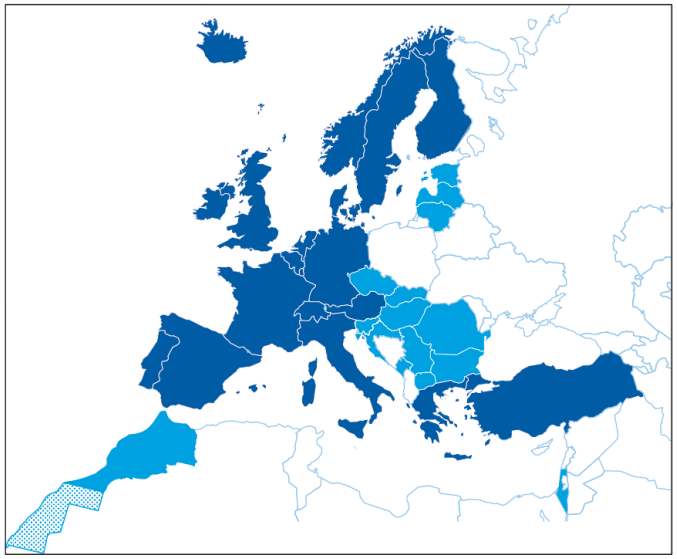
People	About 260 staff, specialists and contractors
Equipment	State-of-the-art supercomputers and data handling systems
Budget	£50 million per year



Global observation system

Global numerical weather forecasts

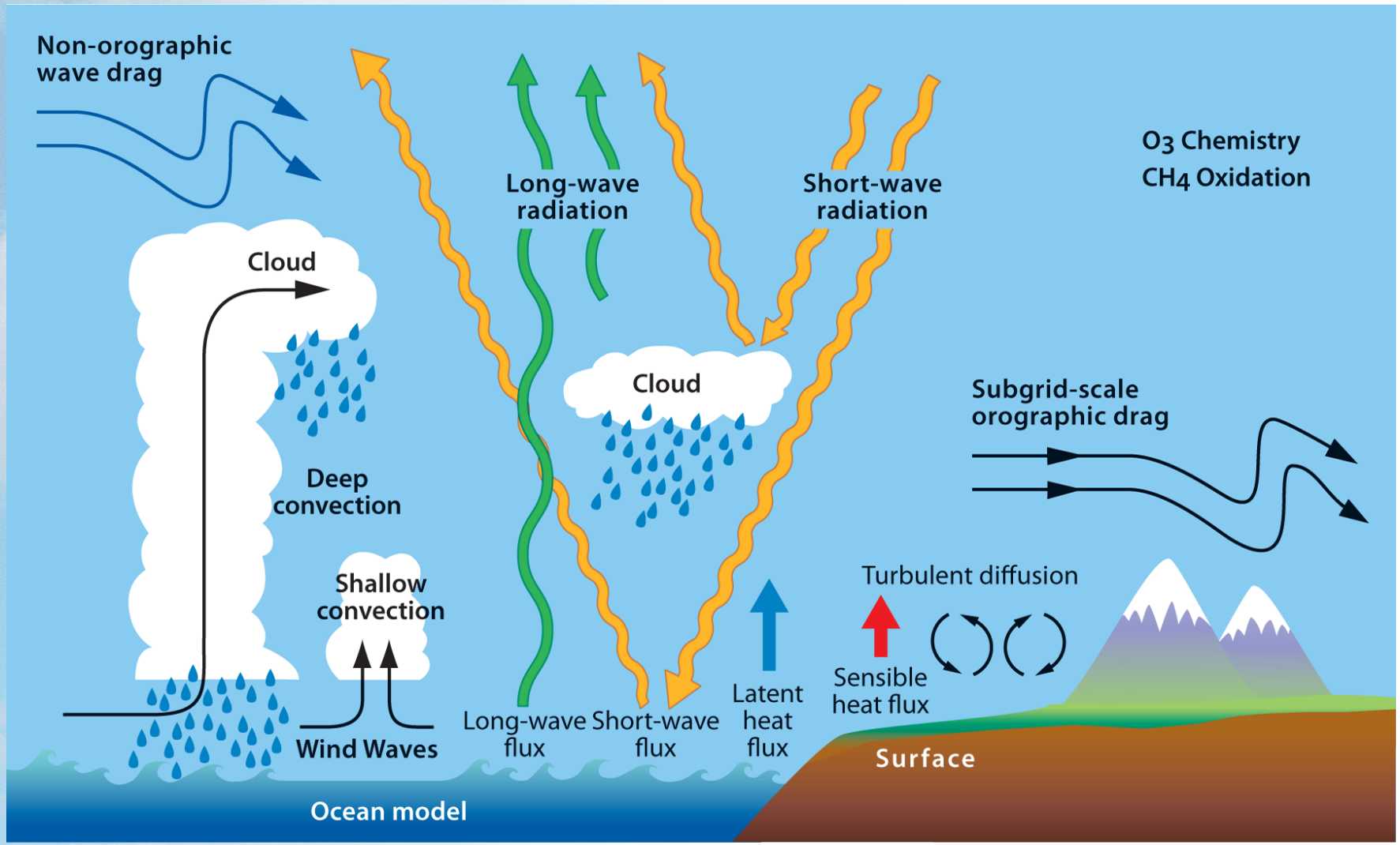
National weather services



Users

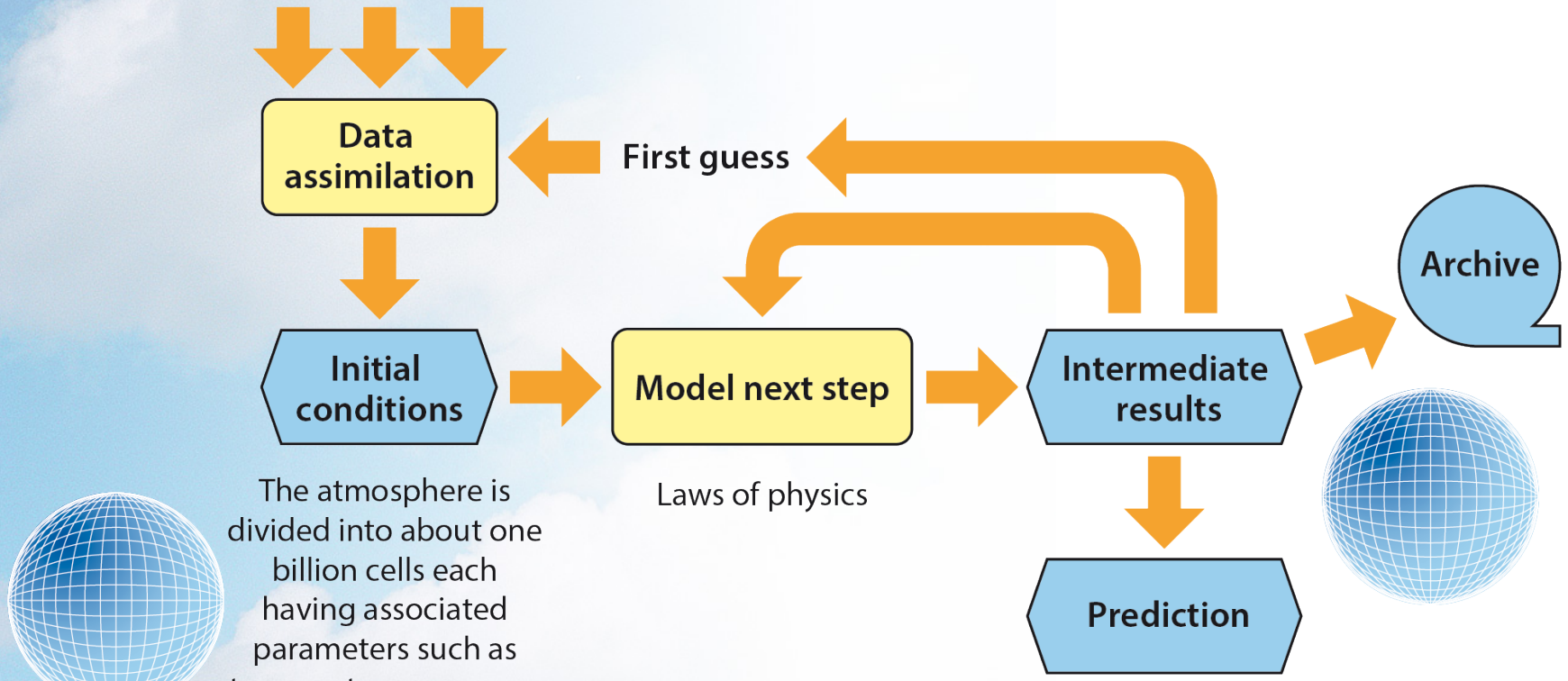
Numerical Weather Prediction

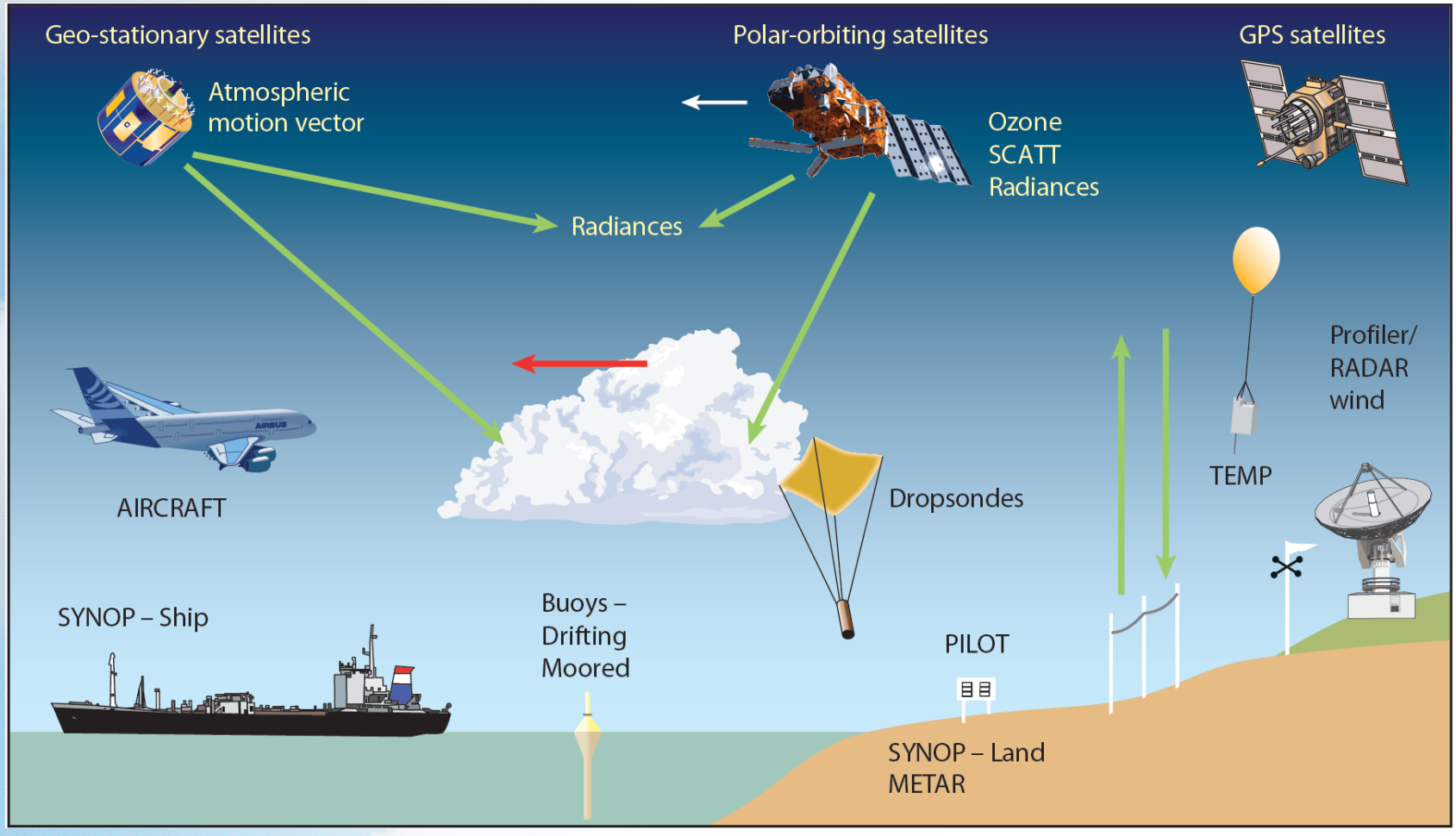
Model


Non-orographic wave drag
Long-wave radiation
Short-wave radiation
O3 Chemistry
CH4 Oxidation
Cloud
Cloud
Subgrid-scale orographic drag
Deep convection
Shallow convection
Turbulent diffusion
Sensible heat flux
Latent heat flux
Long-wave flux
Short-wave flux
Wind Waves
Surface
Ocean model

A basic description of our models

Approximately 20 million observations



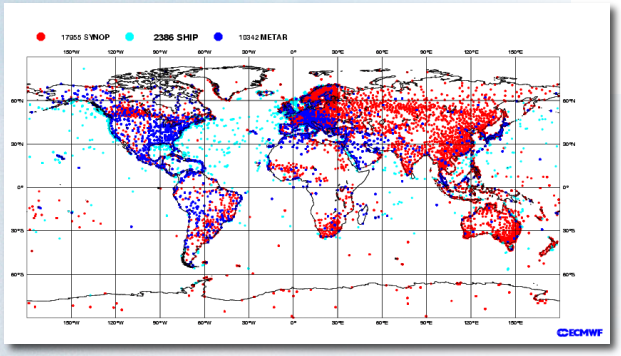


Data sources for the ECMWF Meteorological Operational System

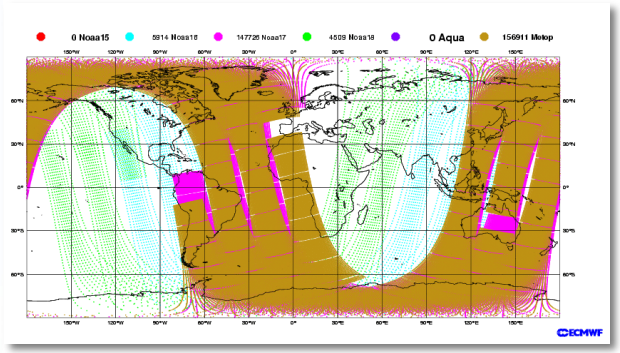
Major assimilated datasets

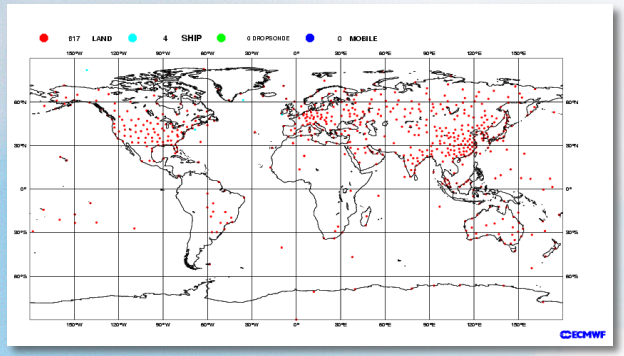
Receive 300 million observations from 130 sources daily.

Surface stations

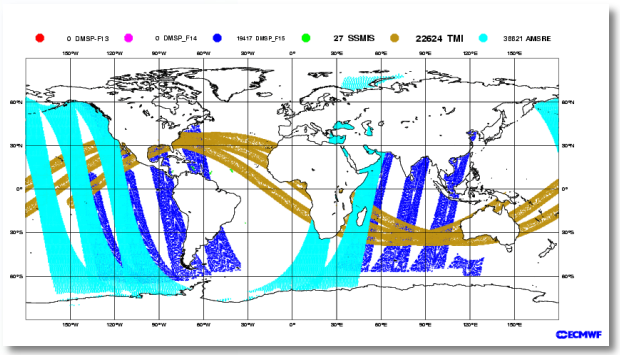


Polar, infrared



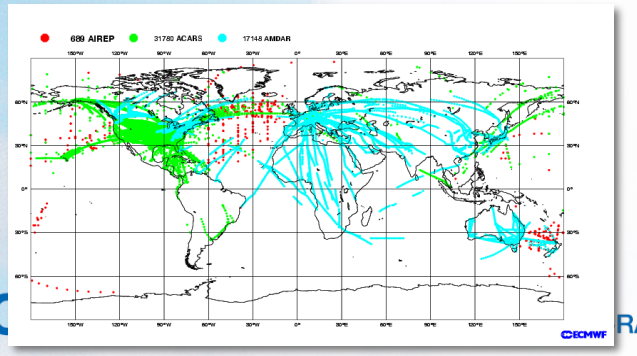


Radiosonde balloons

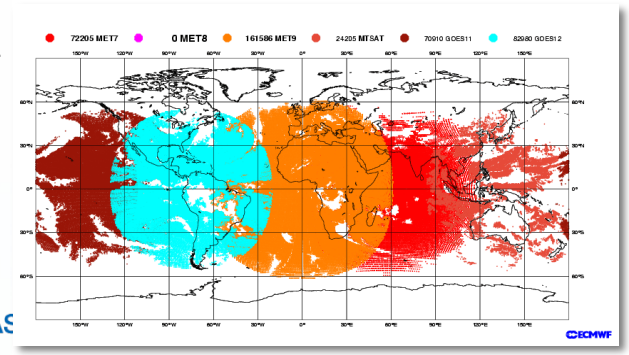


Polar, microwave

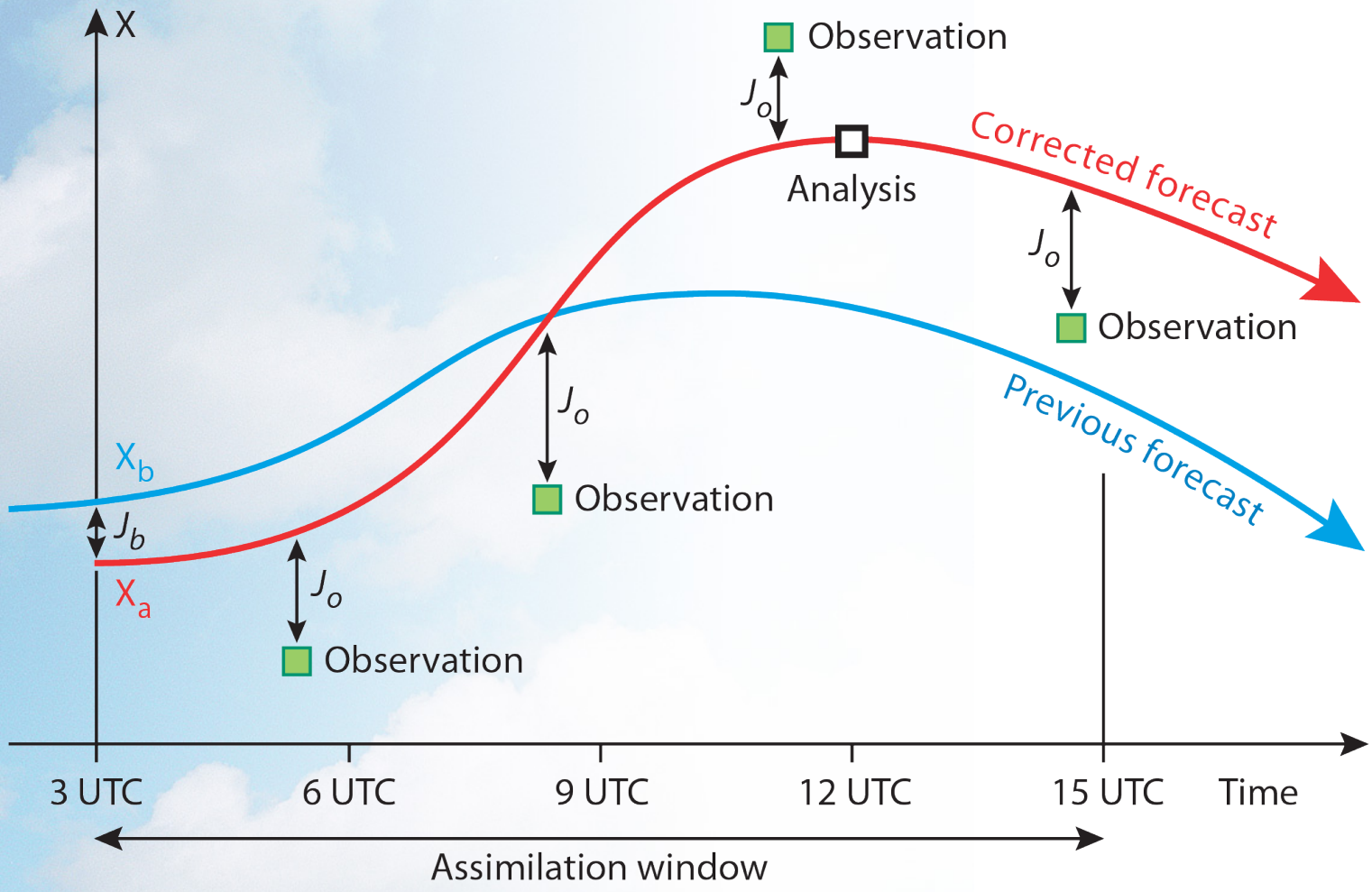
Aircraft



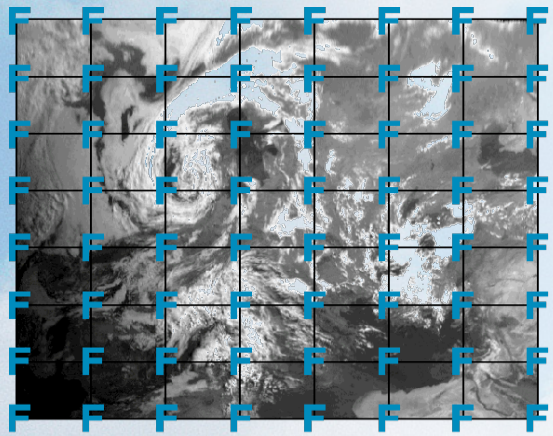
Geostationary, IR



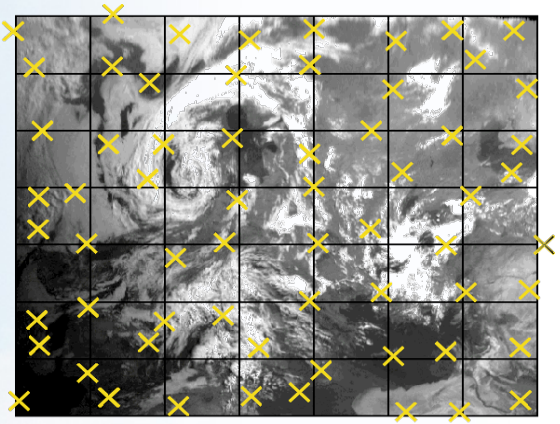
4D-Var Assimilation



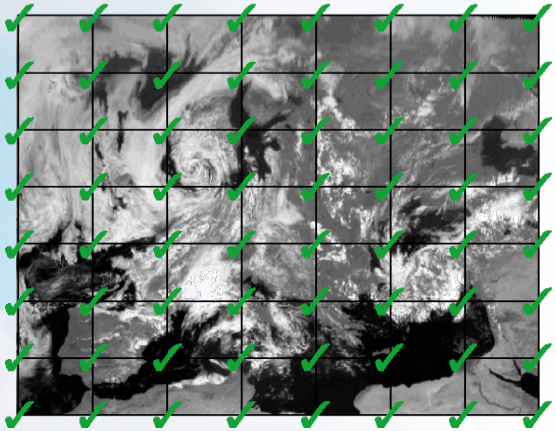
Data Assimilation



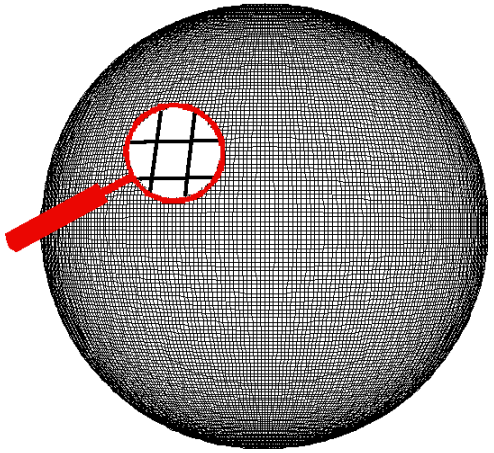
short-range forecast



measurements



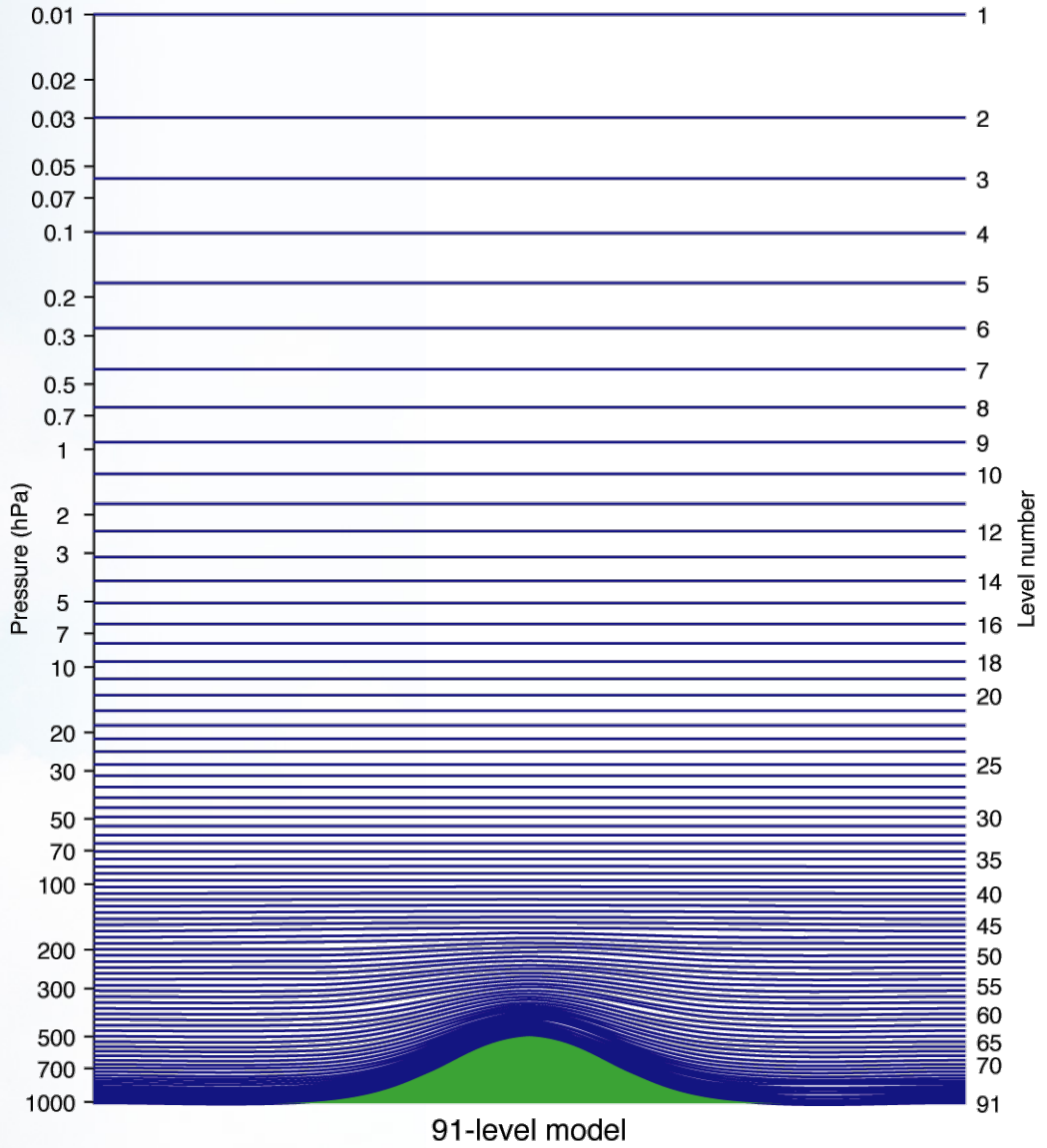
corrected representation of the atmosphere



Model Levels

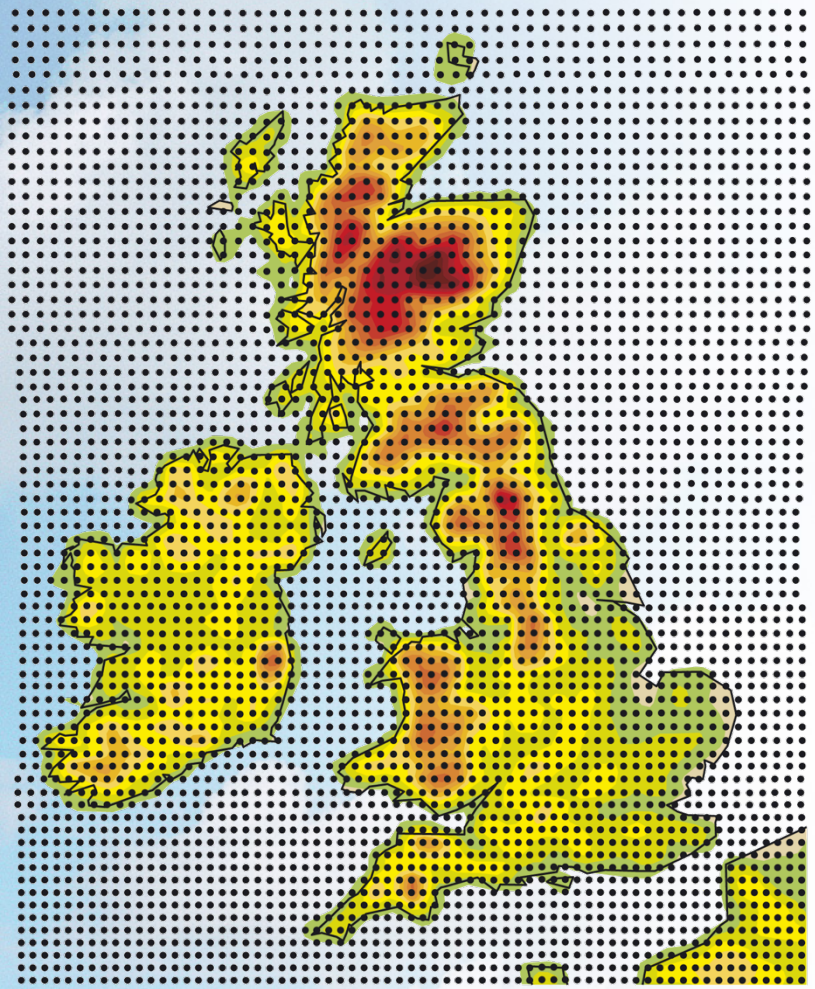
Current operational:
137 levels

Alt. ≈40.000m

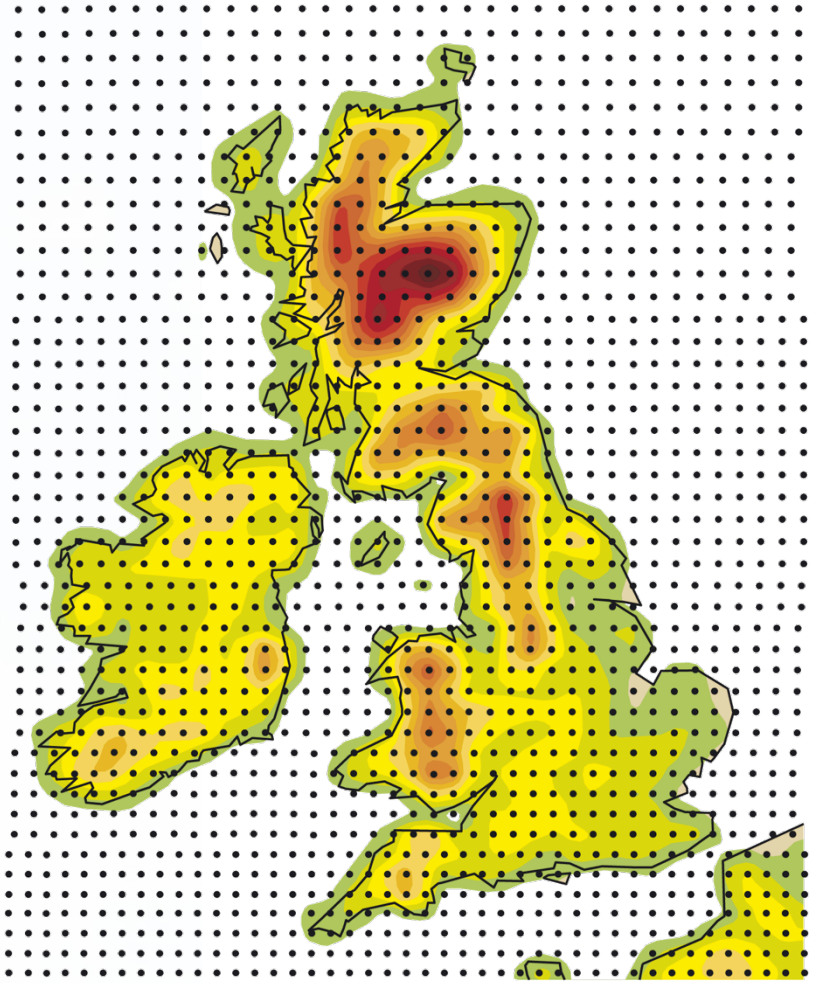


Operational model grid

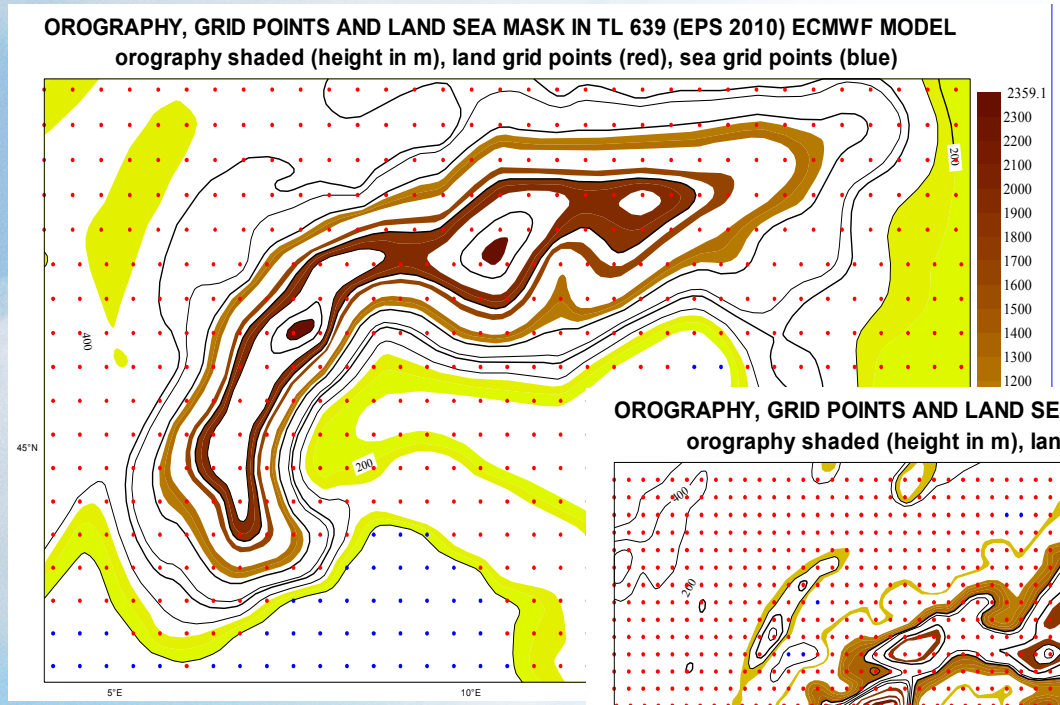
High Resolution (16km)

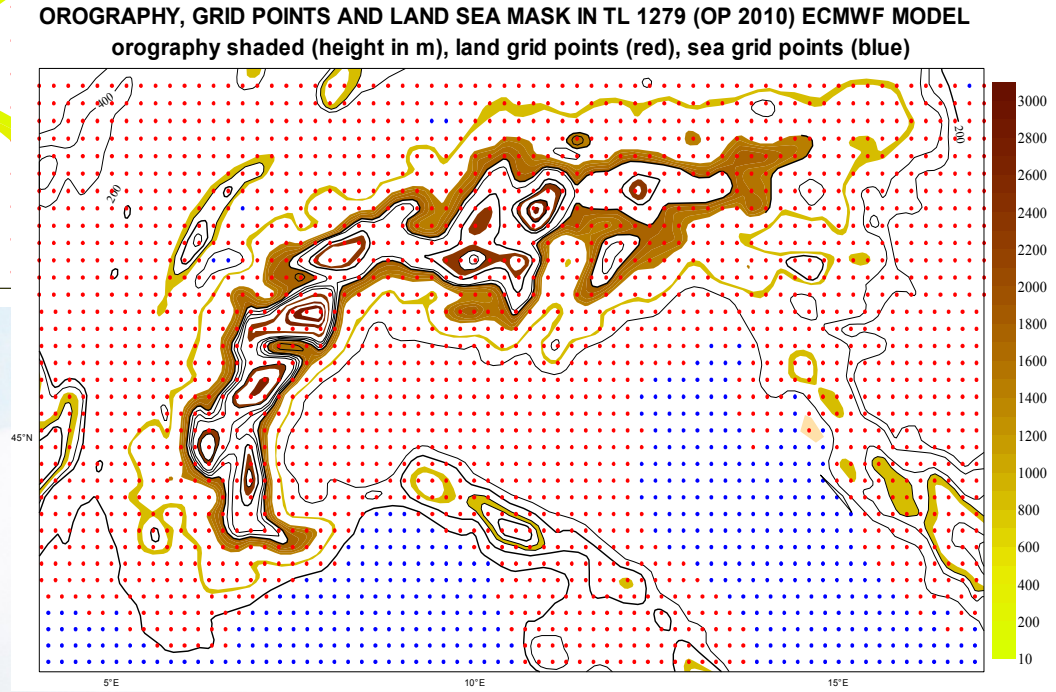


ENS (32Km)



Grid Effect


OROGRAPHY, GRID POINTS AND LAND SEA MASK IN TL 639 (EPS 2010) ECMWF MODEL
orography shaded (height in m), land grid points (red), sea grid points (blue)
2359.1
2300
2200
2100
2000
1900
1800
1700
1600
1500
1400
1300
1200
400
200
45°N
5°E
10°E


OROGRAPHY, GRID POINTS AND LAND SEA MASK IN TL 1279 (OP 2010) ECMWF MODEL
orography shaded (height in m), land grid points (red), sea grid points (blue)
3000
2800
2600
2400
2200
2000
1800
1600
1400
1200
1000
800
600
400
200
10
400
200
45°N
5°E
10°E
15°E

Meteorological Fields

Fields:

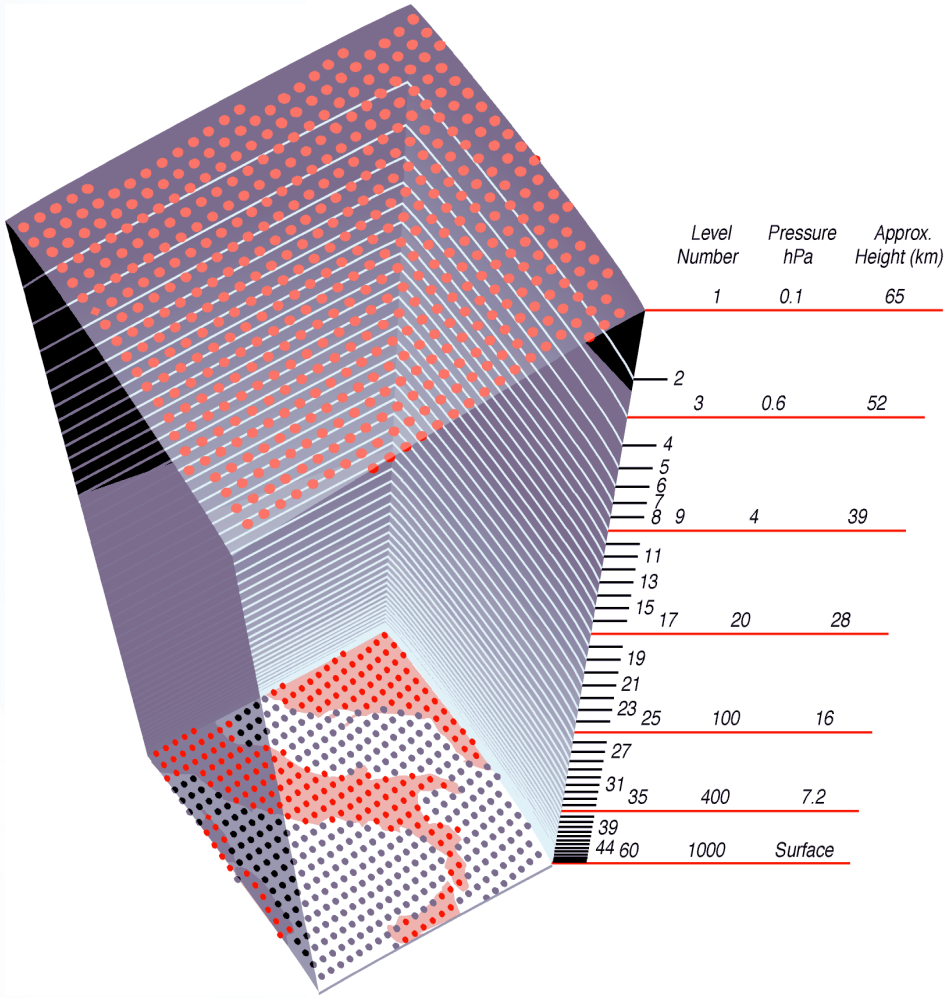
2M grid points x 137 levels

287M values, **per variable**

Operational models produce:

- 13 millions fields daily
- Totalling 14 TB/day

Time critical window: 1h !!!



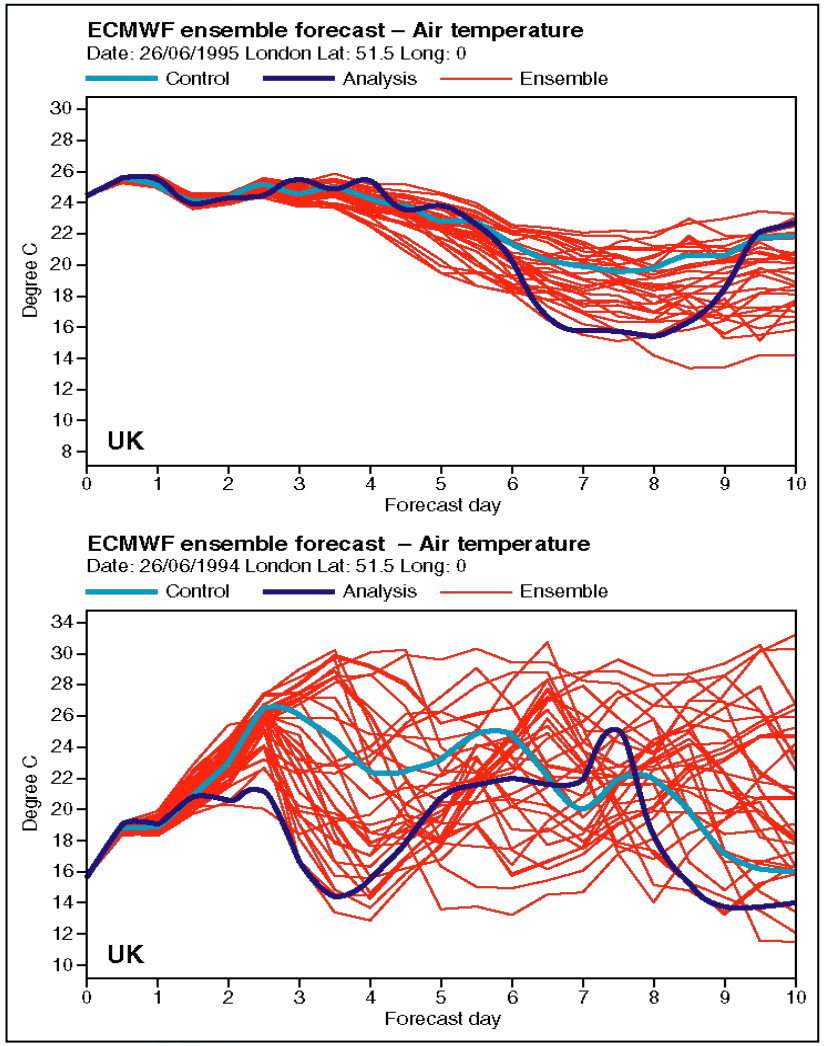
Chaos and weather prediction

The atmosphere is a chaotic system

- Small errors can grow to have major impact (butterfly effect)
- Can never perfectly measure the state of the atmosphere
- Limits weather prediction to a week or so ahead

Ensemble Forecasts

- Parallel set of forecasts from very slightly different initial conditions and model formulation
- Assess uncertainty of today's forecast



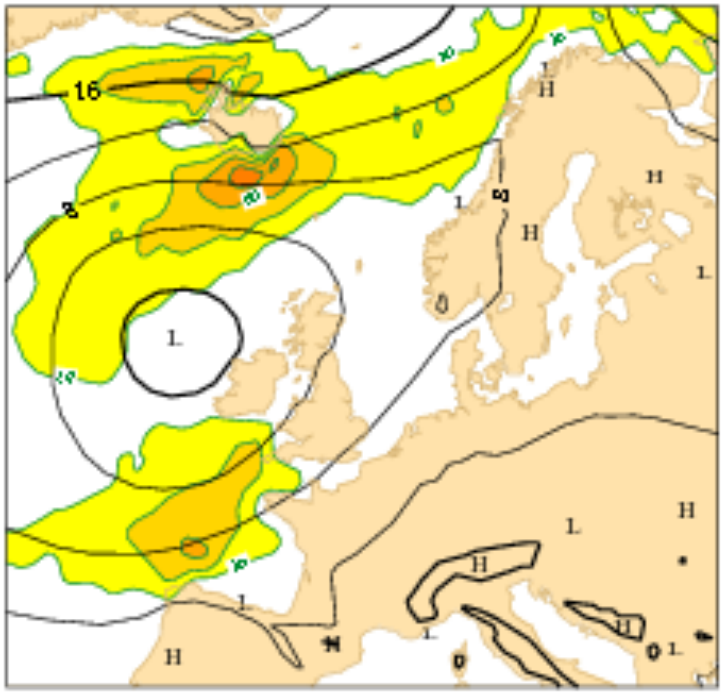
Why run an ensemble prediction system?

Basic idea

- Taking account of uncertainty
- Forecasting forecast skill

Forecasting benefits

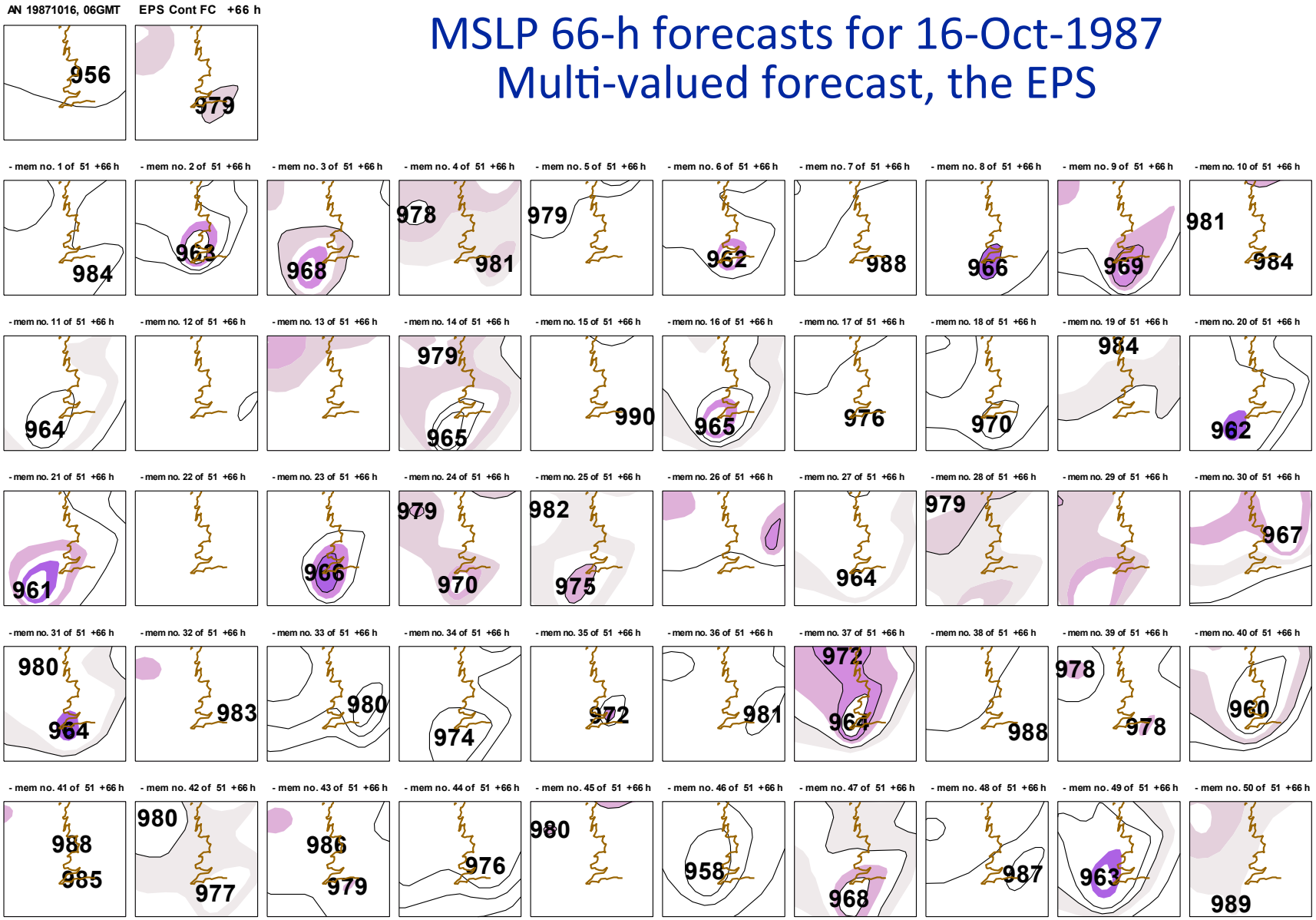
- Assess uncertainty of today's forecast
- Provide alternative forecast scenarios
- Highlight the predictable (large-scale) component and the risk for a less likely but significant (small-scale) event



Continuing challenges

- Forecasting extreme events
- Extending the forecast range

MSLP 66-h forecasts for 16-Oct-1987
Multi-valued forecast, the EPS



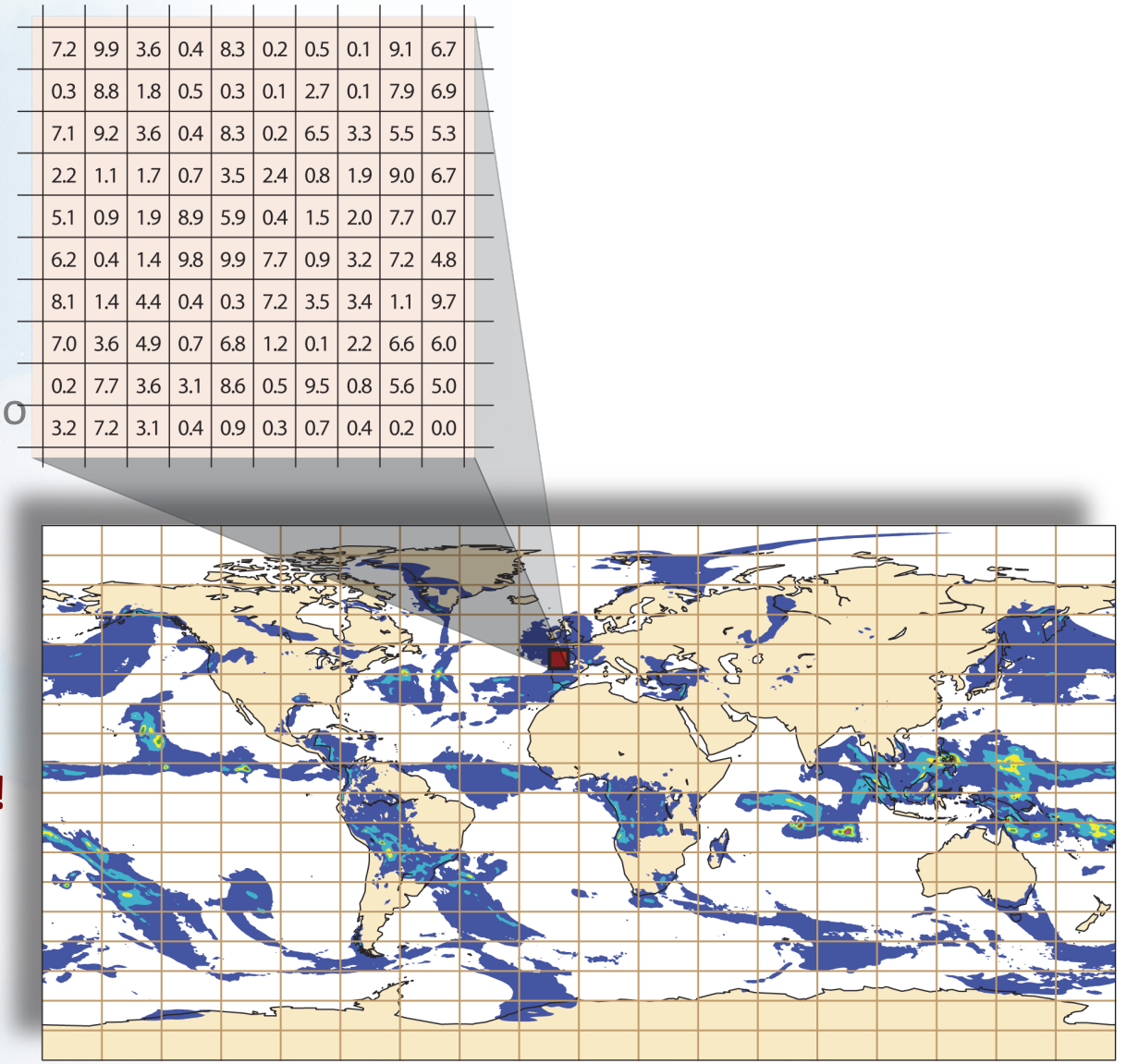
Product Generation

77 million products disseminated ever day, totalling 6 TB.

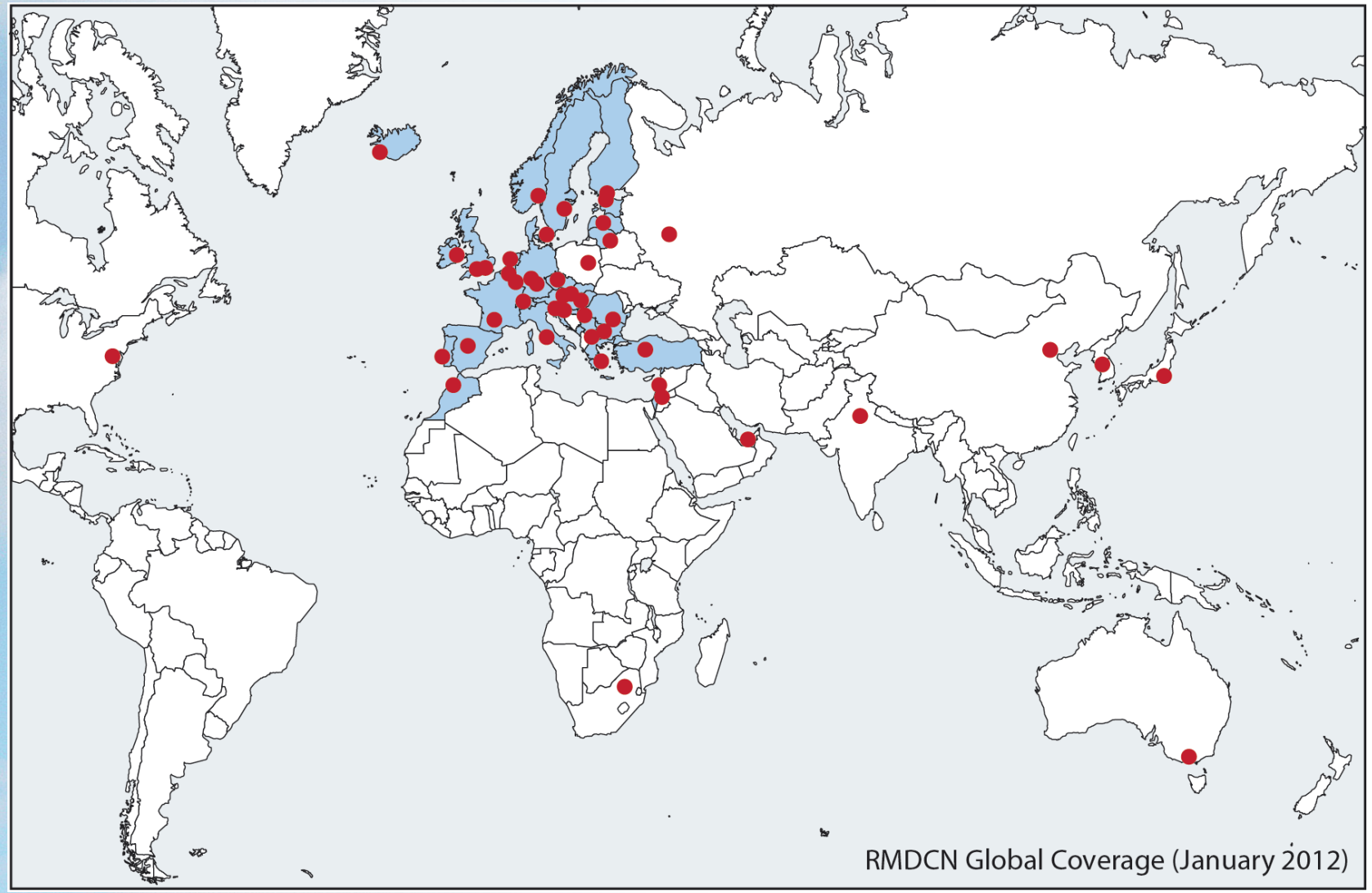
Interpolate output fields into **user required** grids

Product generation is also subject to a **dissemination schedule…**

Time critical window: 1h !!!



RMDCN Connections



Products also served via web

visualisation services

(demo)

Challenges Ahead

Scalable Algorithms

Platform Uncertainty

Big Data

New Algorithms Challenge

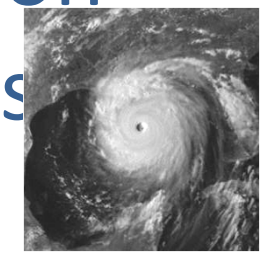
To perform better forecasts:

- Assimilate better & more observations
- Keep increasing model resolution
- Add modeling of new physical phenomena
- Add more ensemble members

More computations

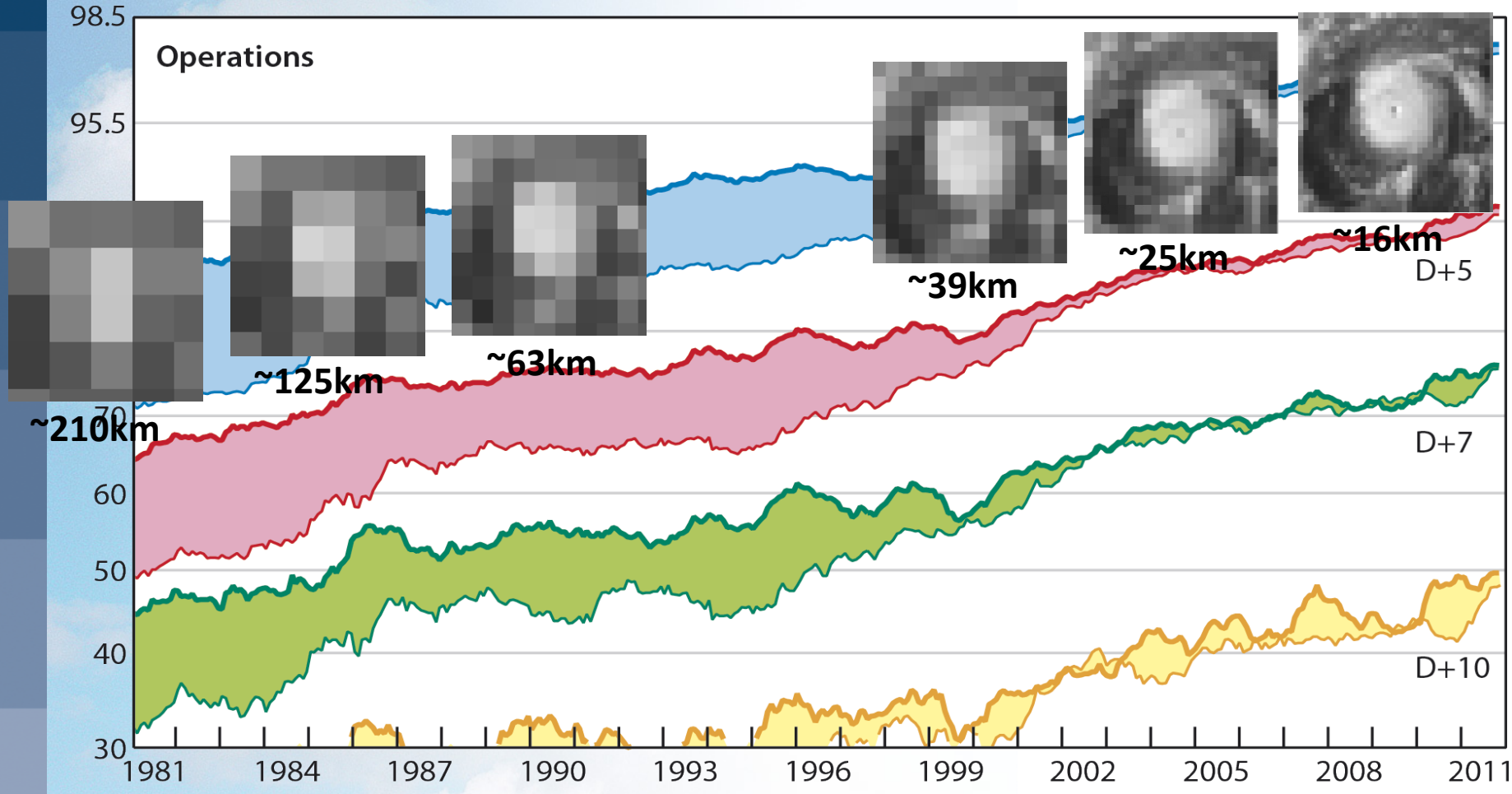
All run within a critical 1h time window

Evolution of ECMWF scores comparison northern and southern hemispheres



Anomaly correlation of 500 hPa height forecasts

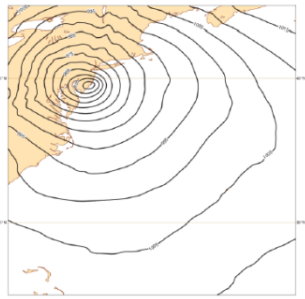
Northern hemisphere Southern hemisphere



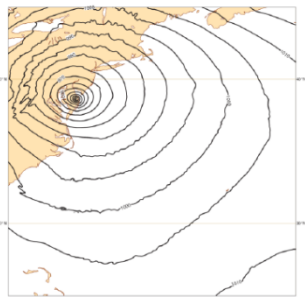
Benefits of High Resolution

Mean sea-level pressure

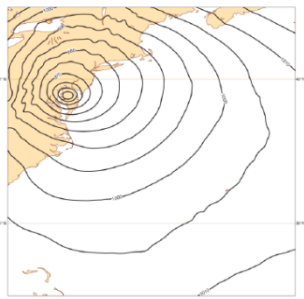
AN 30 Oct



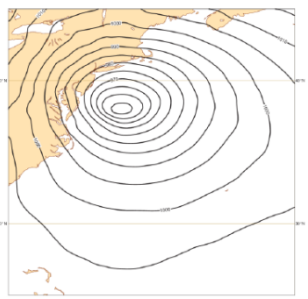
5d FC T3999



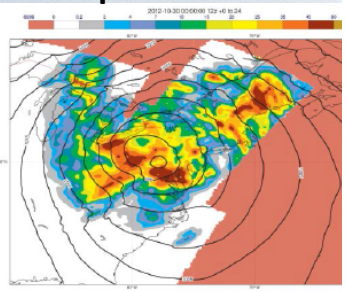
5d FC T1279



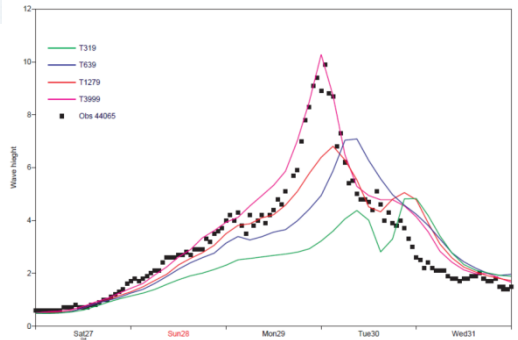
5d FC T639



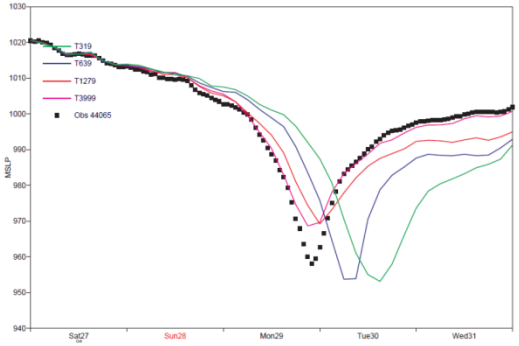
Precipitation: NEXRAD 27 Oct



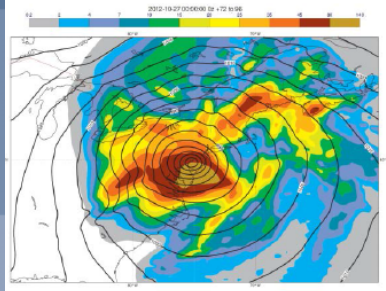
3d FC: Wave height



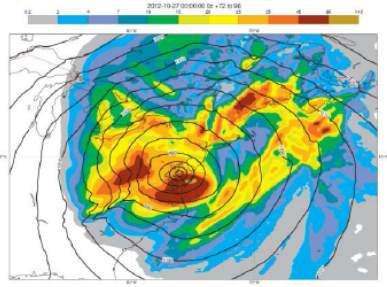
Mean sea-level pressure



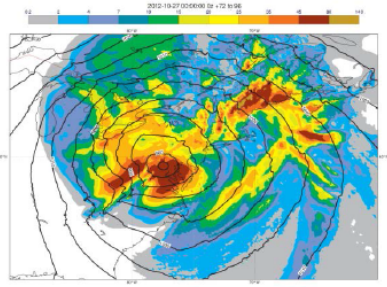
4d FC T639



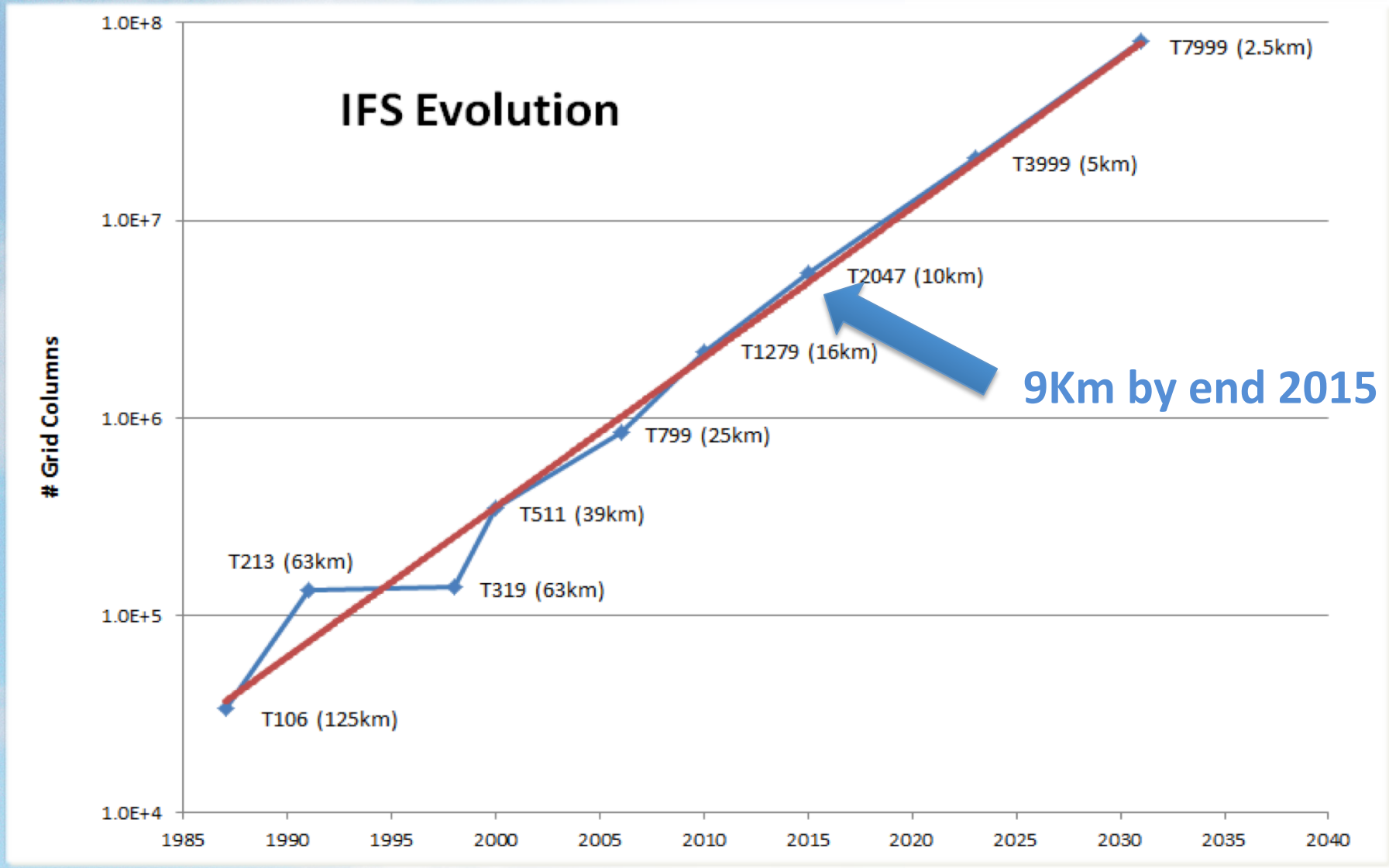
4d FC T1279



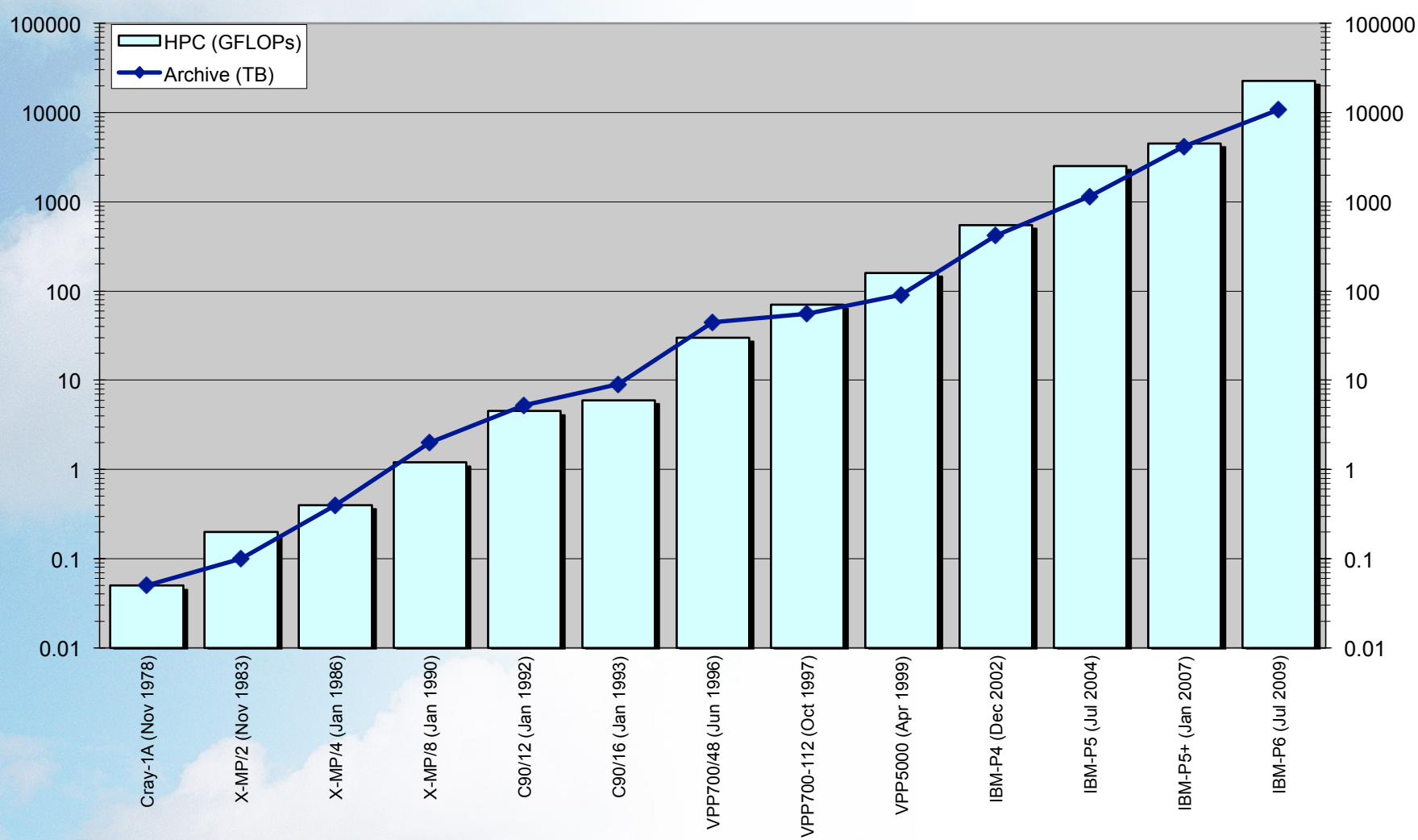
4d FC T3999



Planned Resolution Upgrades



Archive size vs. Supercomputer power



Archival rate has always been **strongly** correlated with computational power

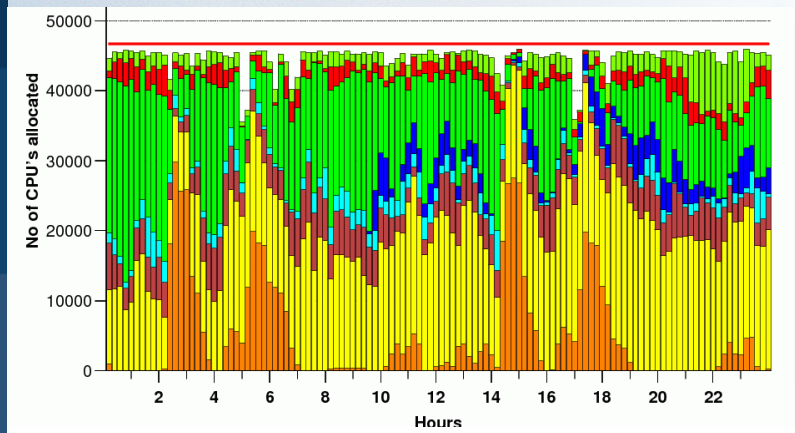
Resolution Upgrades

Resolution	Grid size	Grid Points	Field Size (in memory)
T319	62.5 km	204 k	1.6 MB
T511	39 km	524 k	4 MB
T799	25 km	1.2 M	9.6 MB
T1279	16 km	2.1 M	16.8 MB
T2047	10 km	8.4 M	67.2 MB
T3999	5 km	20 M	160 MB
T7999	2.5 km	80 M	640 MB

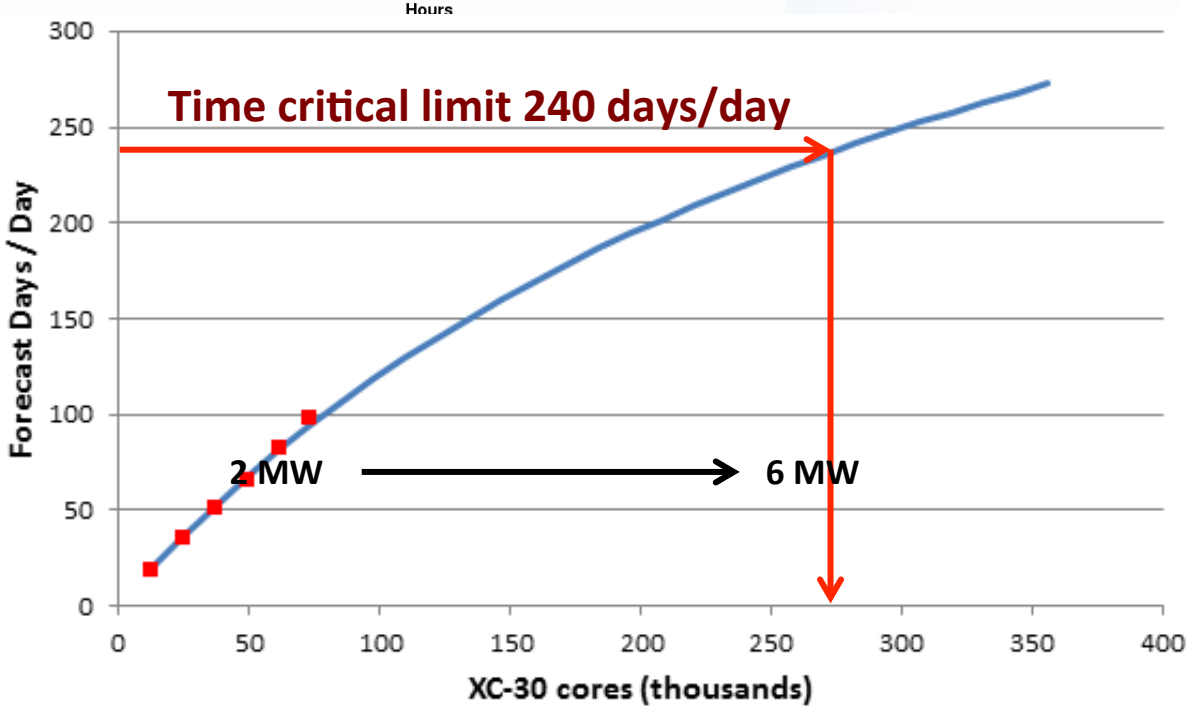
As memory per core diminishes (think GPU's and Intel Phi) …

… this may have serious implications on the data processing software!

Power Challenge



→ 25% of HPC used for operations
→ 10% of operations for HRES forecast
→ 2.5% of HPC



→ 2.5 km* forecast on Cray XC-30 (84,000 cores) = 2 MW

→ 240 days/day require 270,000 cores = 6 MW

→ x 10** = 60 MW ?

* planned for 2025
** scaling from HRES to full HPC

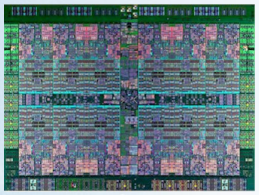
Platform Uncertainty Challenge

- How to keep up with the **changing** hardware landscape?
- How to **best** use the new heterogeneous systems?
  - CPU's
  - GPU's
  - Accelerators
- **With very limited human resources…**

Hardware Landscape

Proliferation of HPC Architectures

Intel Xeon



OpenPOWER™

IBM Power8

NVidia Tesla

AMD FirePro

Intel Xeon Phi

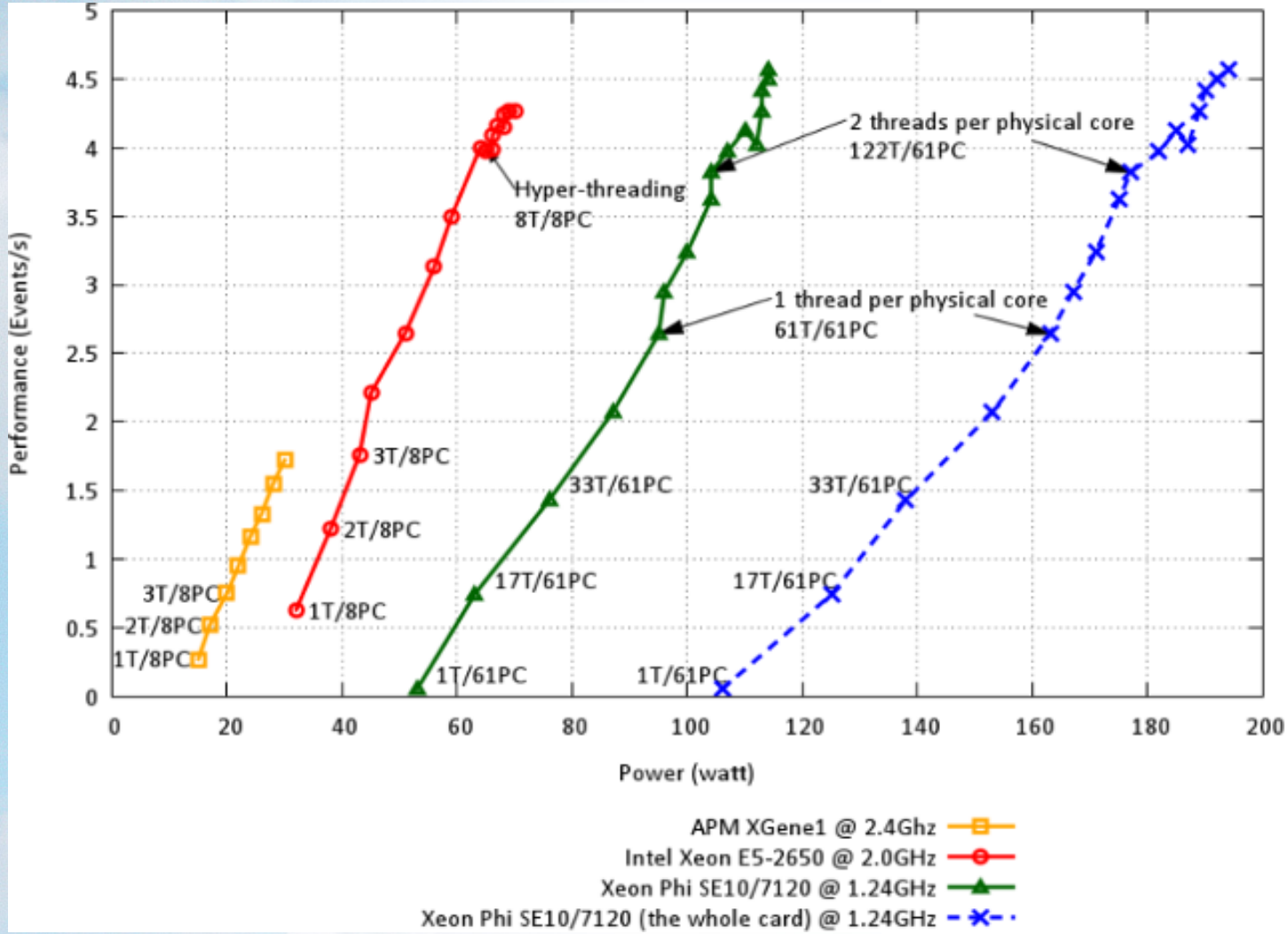
NEC SX-ACE Aurora

ThunderX ARM

X-Gene ARM

Power Matters

"Heterogeneous High Throughput Scientific Computing with APM X-Gene and Intel Xeon Phi", Adburachmanov et. al., 2014



Hypothetical Solution

- Need flexibility?
- Need performance?
- Code easier to maintain?
- A modern language?

We have a solution…

C++

But …

HPC Platform Restrictions

- **POSIX**
- Only supported languages:
  - **Fortran**
  - **C99**
  - Recently **C++ (but not C++11)**
  - Shell (Ksh) + Perl **(but not Python)**
- Most of our scientists know
  - **Fortran**
  - Python (maybe)
  - Basic shell scripting

C++ Support on HPC

- Support for C++ in HPC platforms is … **patchy!**
- Some HPC compilers aren't fully C++98 and choke on heavily template code:
  - Eigen library for linear algebra
  - Parts of Boost
    - serialization
    - Proto
- Not to mention **C++11**!

Change on the way?

Community & Vendors start talking about C++

Scientists are reluctant to use C++

Possible explanations (**my view**):

- Fortran is what they know
- Fortran is very fast for numeric computations
- C++ was (initially) perceived as slow:
  - hidden constructors
  - misuse of classes
  - **restrict** keyword

  Mostly corrected now (e.g. work of T. Veldhuizen)
- **But** C++ is still perceived as …
  - complex (templates are a common complaint)
  - less portable (than C or Fortran)

Big Data Challenge

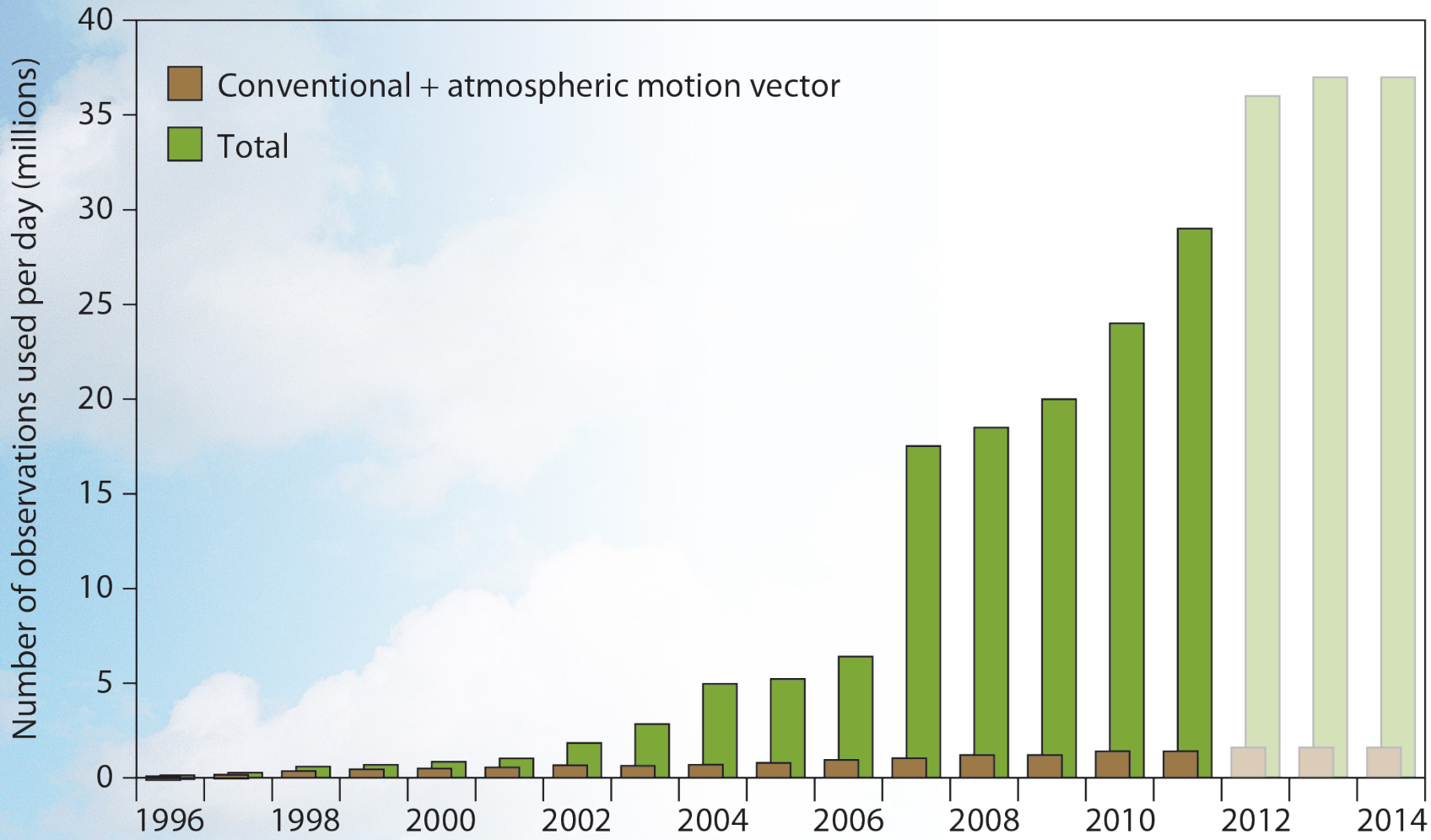
"Big Data is high volume, high velocity, and/or high variety information assets that require new forms of processing to enable enhanced decision making, insight discovery and process optimization."

"3D Data Management: Controlling Data Volume, Velocity and Variety", D. Laney, Gartner, 2001

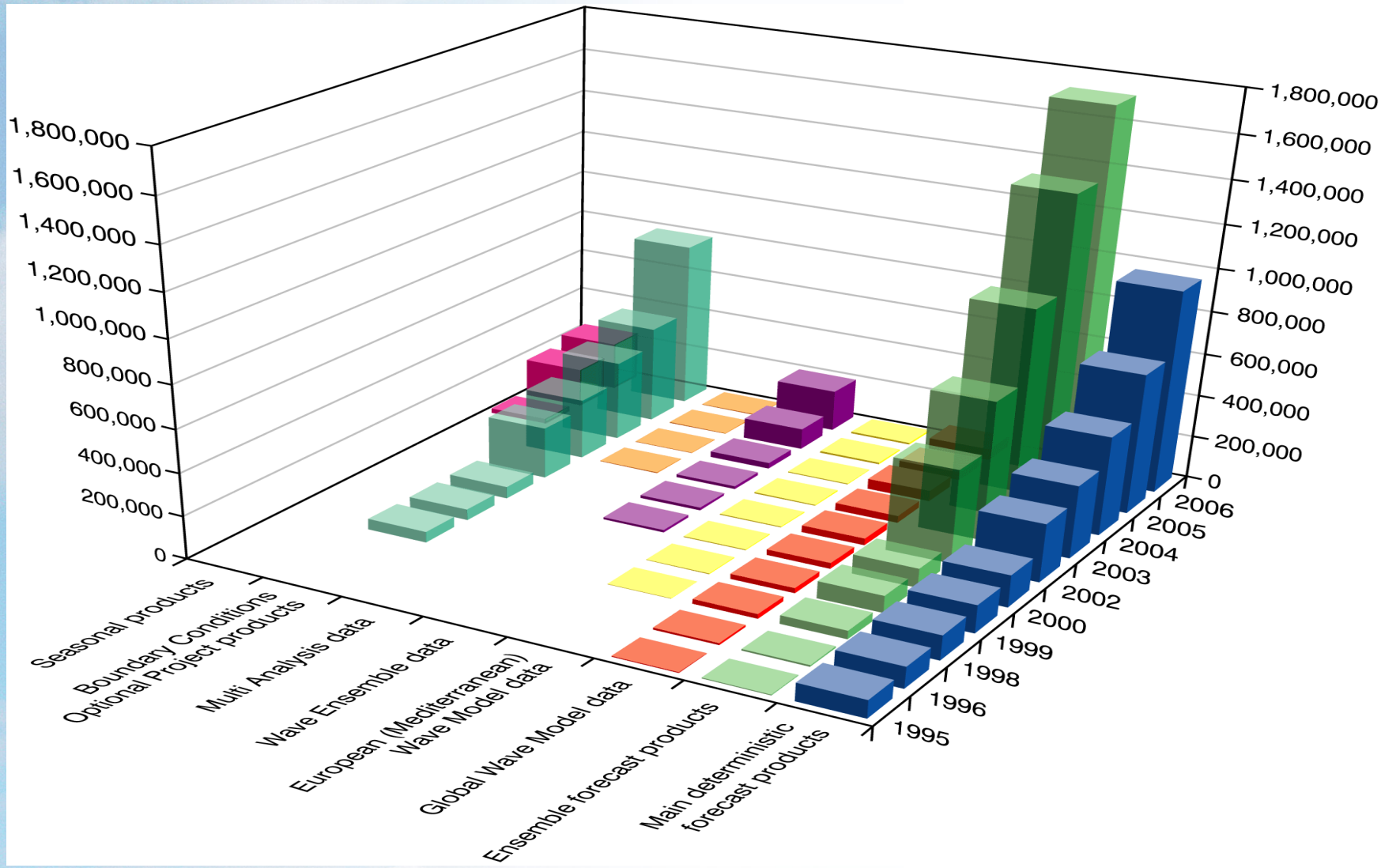
The 3 V's of Big Data

V is for Volume: Observations

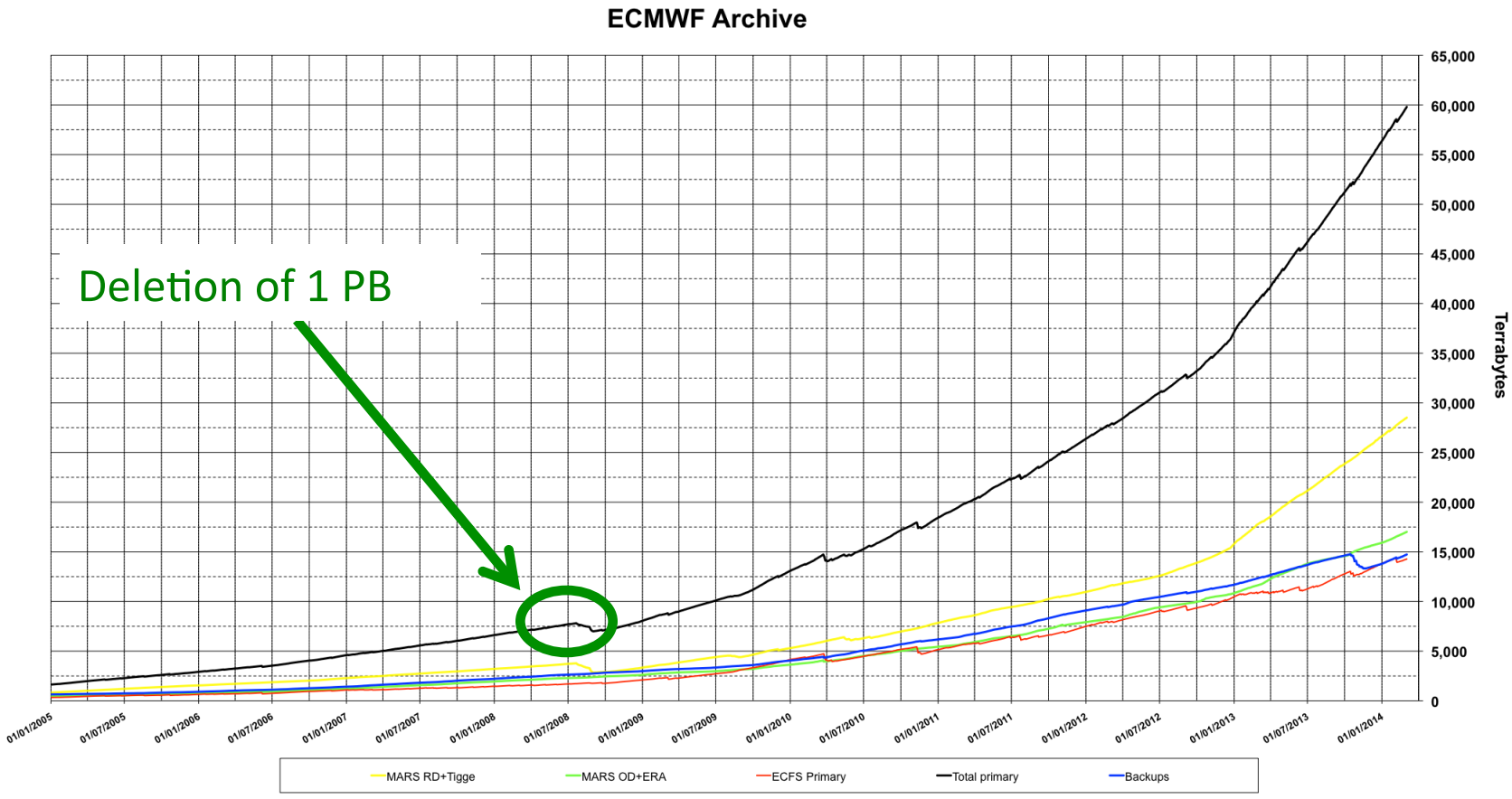
Increase of satellite data usage



V is for Volume: Products

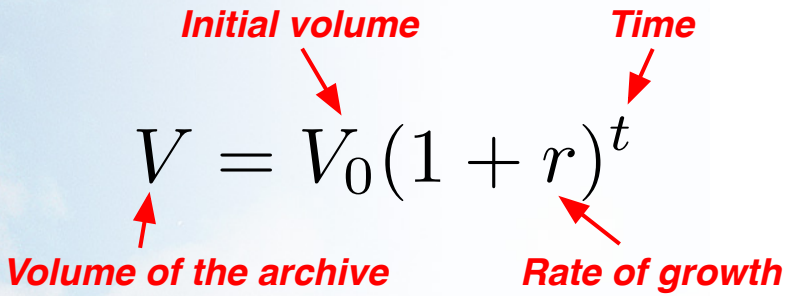


V is for Volume: Archive



V is for Velocity

- ECMWF's archive grows exponentially:

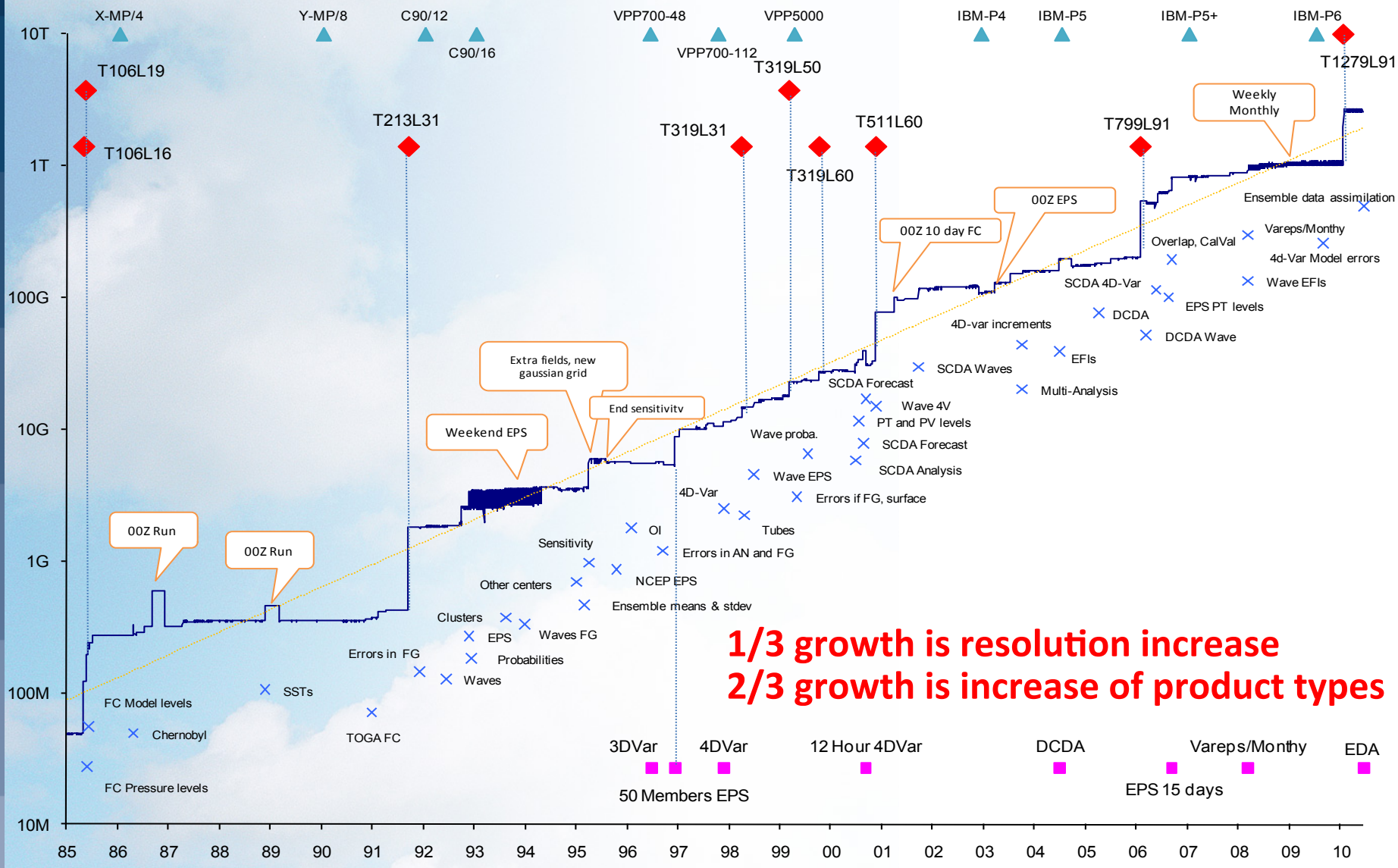


  - r is around 0.5, which is a 50% increase per year
  - The rate of added data also grows exponentially at the same rate!

$$\frac{\partial V_0(1+r)^t}{\partial t} = V_0 log(1+r)(1+r)^t = A_0(1+r)^t$$

- In 1995, the size of the archive was increasing at a rate of 14 TB/year.
- In 2015, the size of the archive increases at a rate of 100 TB/day

V is for Variety



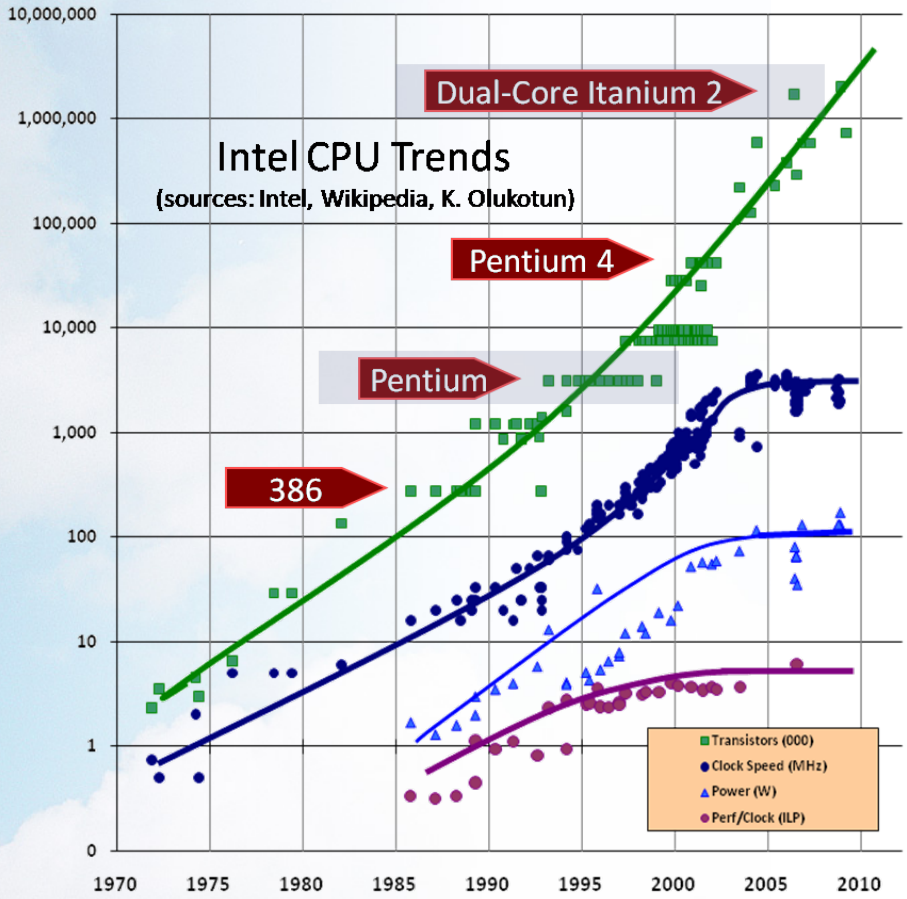
1/3 growth is resolution increase
2/3 growth is increase of product types

Nothing of this is new
We have always dealt with Big Data…

What changed?

CPU Power Growth

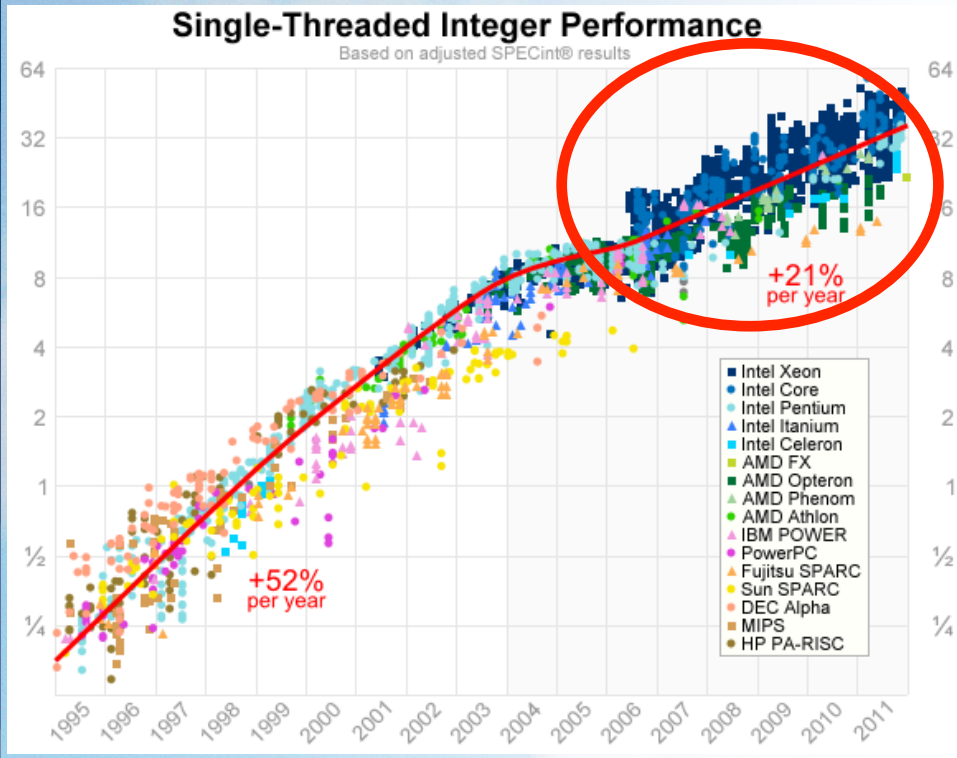
"The Free Lunch is Over". H. Sutter, Dr. Dobb's Journal, 30(3), March 2005

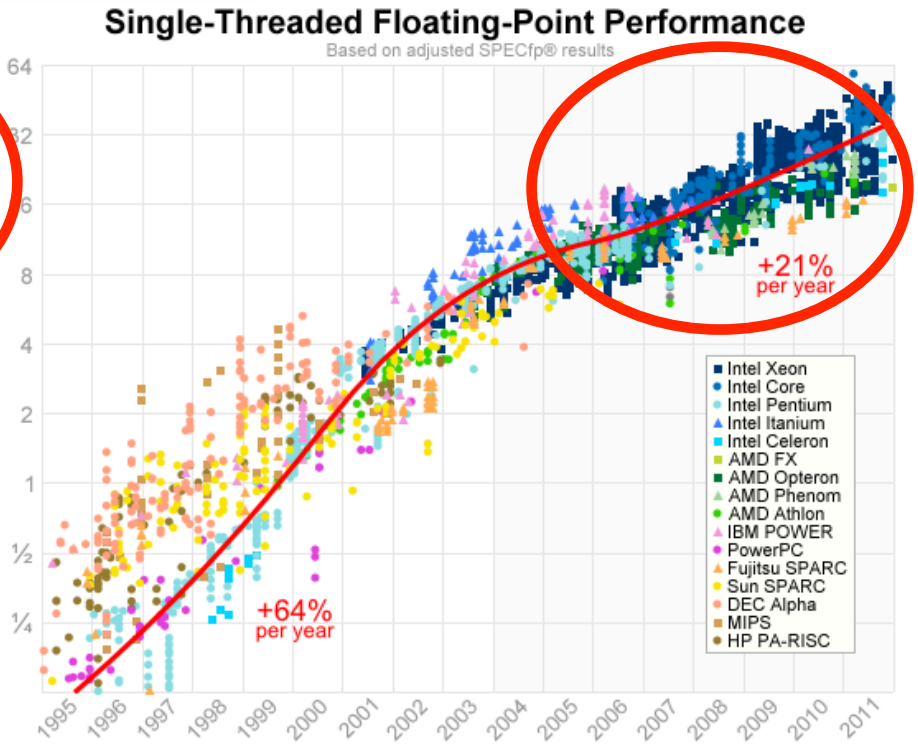


But what about "real" performance?

CPU Performance Growth (single-threaded)

"A Look Back at Single-Threaded CPU Performance", J. Pershing Feb 2012





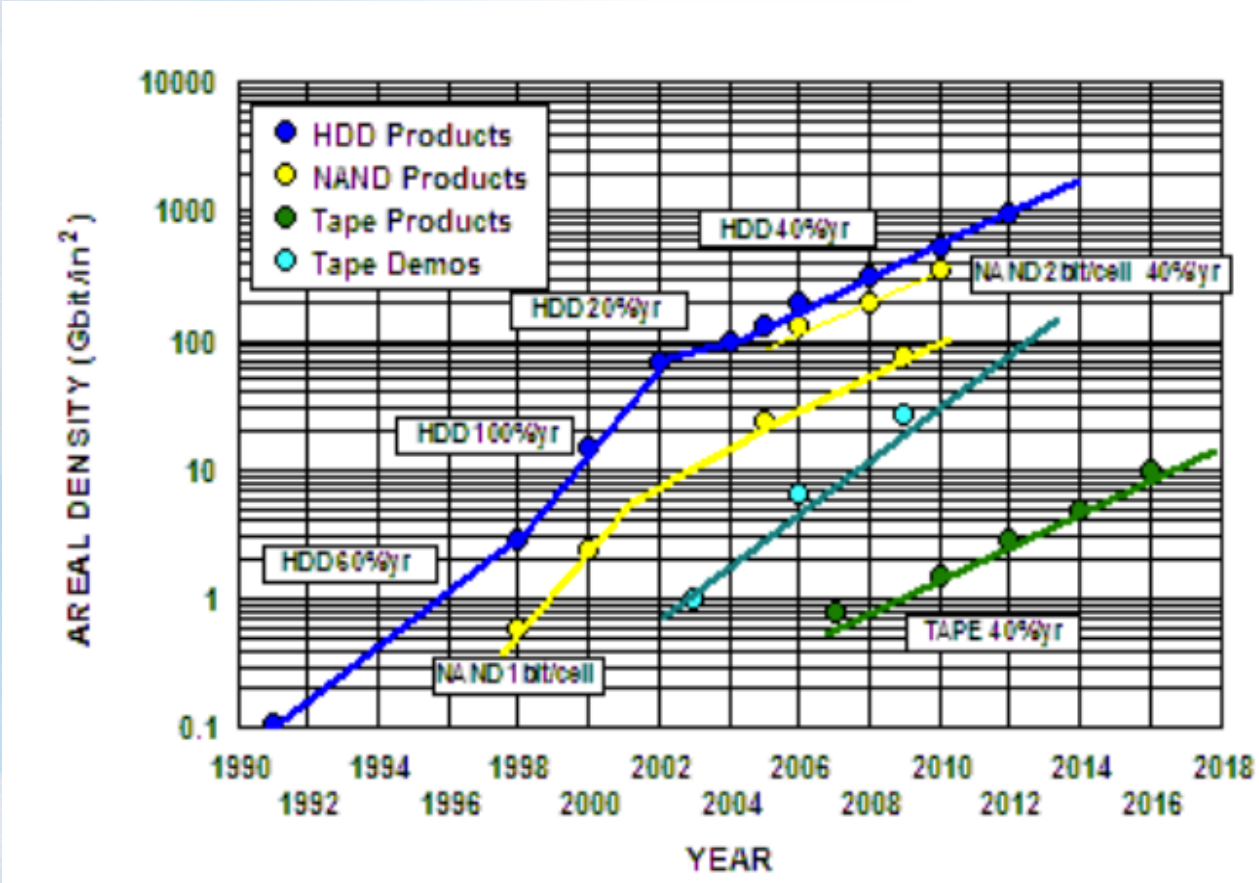
More registers, vector units, branch prediction …

but also harder to achieve!

Deeper memory hierarchy, out-of-order execution …

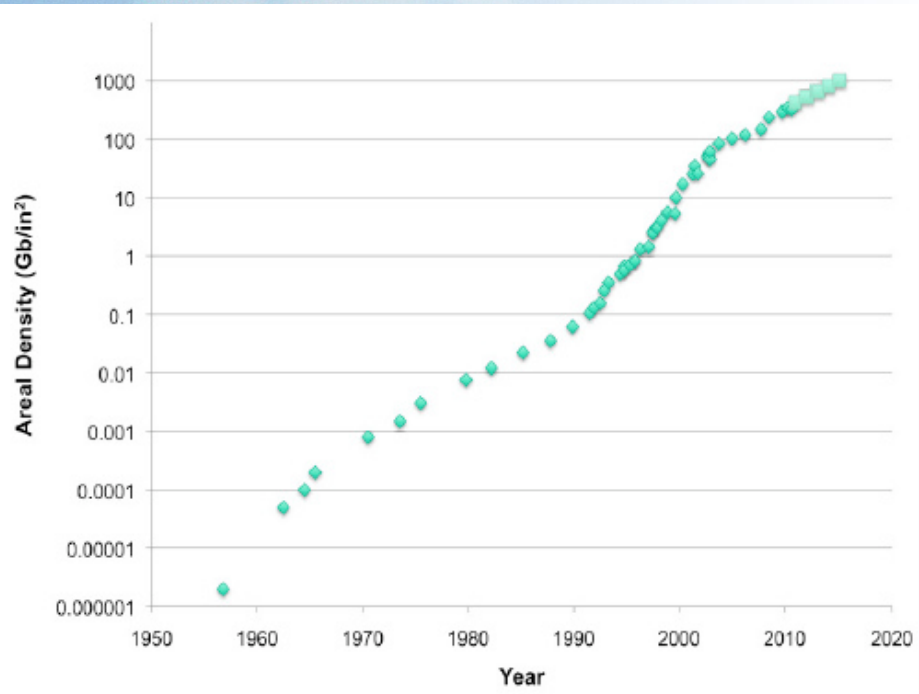
Storage Density Growth
Multiple Technologies

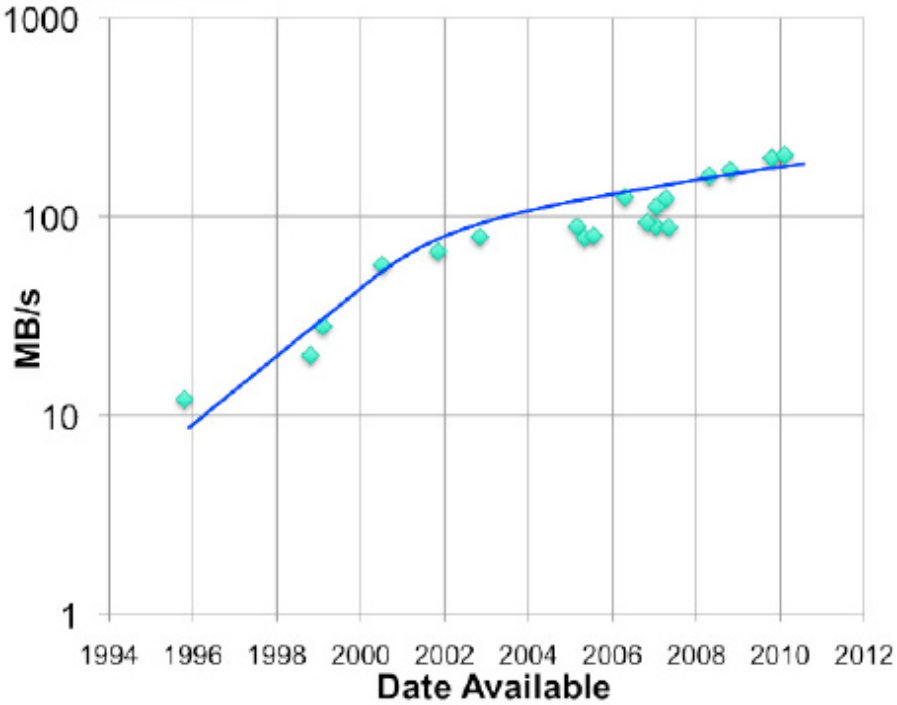
"Tape based magnetic recording: technology landscape comparisons with hard disk drive and flash roadmaps", R. Fontana et al, IBM Research Division, 2011



HDD Storage Growth

"GPFS Scans 10 Billion Files in 43 minutes". R. Freitas, et al. IBM Research Division, 2011





Volume is linearly proportional to area density

Recently follows 25-40% CAGR…

… but transaction rate hasn't kept up!

This means that we may have the capacity, but maybe not the bandwidth …

What does it imply?

- **"No Free Lunch"** ➔ Improve our software
  - NWP models have been parallel and concurrent for many decades
- Optimize **Data Movement**
- Develop new **Algorithms** that explore…
  - (More) Concurrent computations
  - Data locality
  - Computational intensity (CPU usage/MB transferred)
- Software must cope with changes – **Flexibility**
  - Best use of new hardware
  - Unknown future for parallel platforms

Learning from the Past

MARS

B. Raoult, M. Fuentes

Meteorological Archival and Retrieval System

A managed archive, **not a file system**

- Users not aware of data location
- Express access in meteorological terms

Data is kept **forever**:

- As data is accumulated, datasets become more useful
- Deleting old data in an exponentially growing archive is meaningless

Consists of 3 layers:

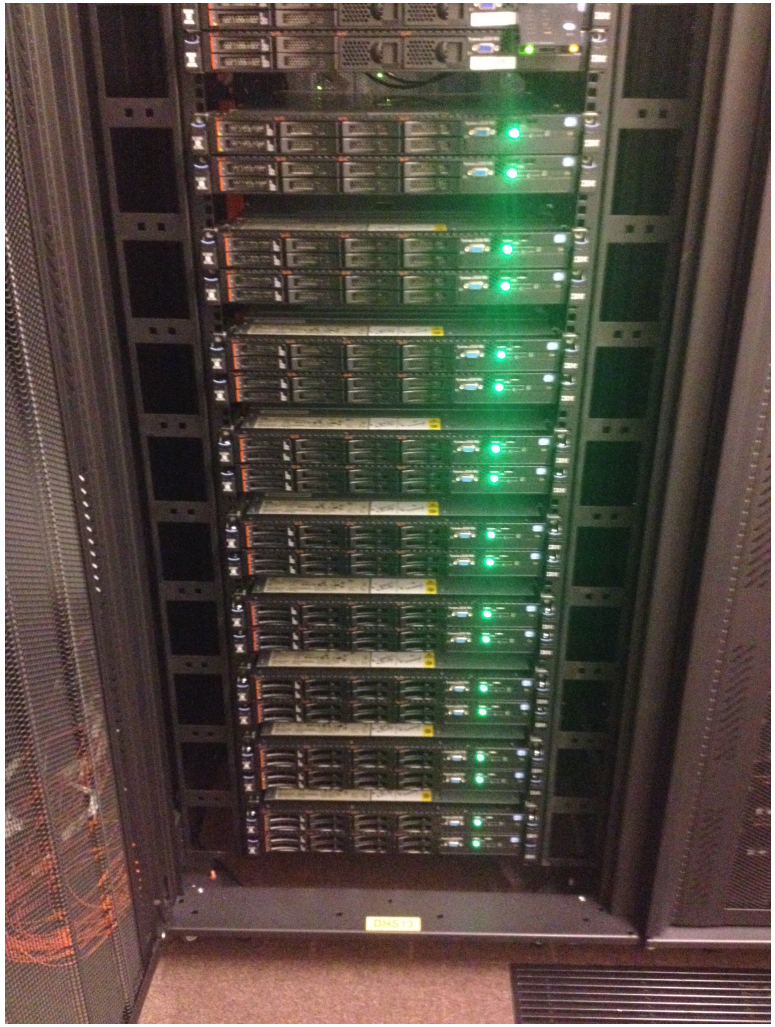
- Cache in the HPC (~80% hit ratio)
- Servers HDD cache (~80% hit ratio)
- HPSS Tape system

Meteorological Archival and Retrieval System

Fully distributed

- 15 servers for metadata and data movers
- 80 PB primary archive
- 1.5 PB of disk cache (2.5%)
- 110 billion fields in 8.5 million files (2014)
- 200 million objects (2014)
- 100 TB added daily
- 100 TB retrieved per day
- 2 million requests per day

- Fully written in C++
- Since 1995
- Index is Object Oriented DB
- Persisted in filesystem



MARS 2011 Migration

- From monolythic to distributed
- From AIX to Linux
- From Big Endian to Little Endian
- Metadata indexing 40PB
- Job 2 years in the planning

- Degraded service for a couple of hours
- Noone noticed...

A meteorological language

- retrieve,
  date = 20110101/to/20110131,
  parameter = temperature/geopotential,
  type = forecast,
  step = 12/to/240/by/12,
  levtype = pressure levels,
  levels = 1000/850/500/200,
  grid = 2/2,
  area = -10/20/10/0

- This request:
  - represents 31*2*20*4 = 4960 fields
  - defines an interpolation followed by cropping

Indirection is the key to scalability

Preparing the Future

Atlas

Hermes

Aims

- Big data challenge (scalability)
- Adaptable solution (flexibility)
- Code evolution (maintainability)

Lets use C++ (98)

Atlas

W. Deconinck, T. Quintino

Challenges: New Algorithms & Platform Uncertainty

Current IFS Model

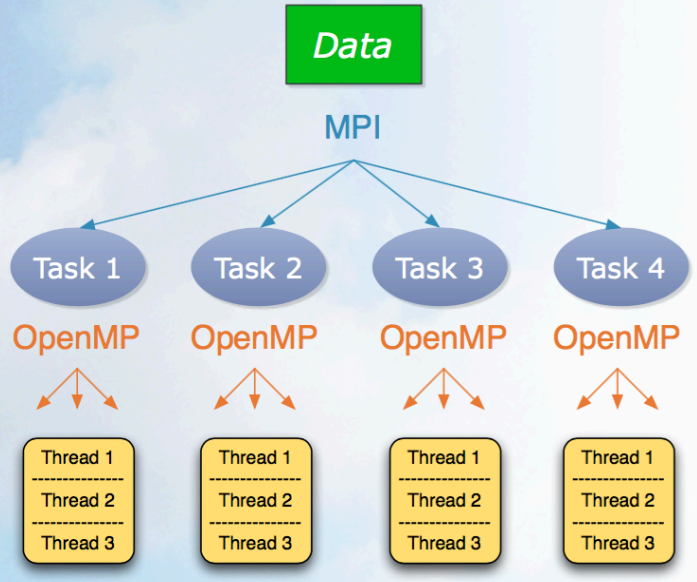
- 2M lines of Fortran code
- 9000 routines / modules
- Hybrid MPI + OpenMP
- Fixed data-structure
- Well organized but…
- … lots of global data.
- Impractical to rewrite
- Fortunately most global operations are located in the “dynamical core”.

Atlas

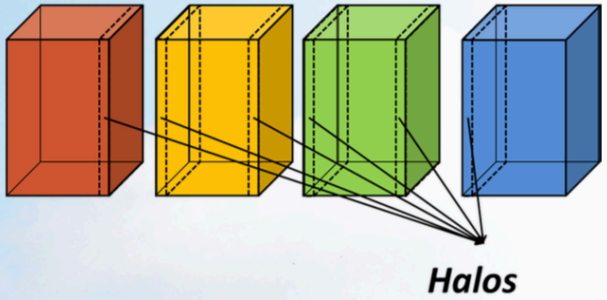
- Framework for parallel, dynamic data structures
- Supporting multiple types of grids
- Fully written in **C++**
- Providing Fortran 2003 interfaces
- Basis to develop **scalable** dynamical core

Atlas capabilities

Parallelisation



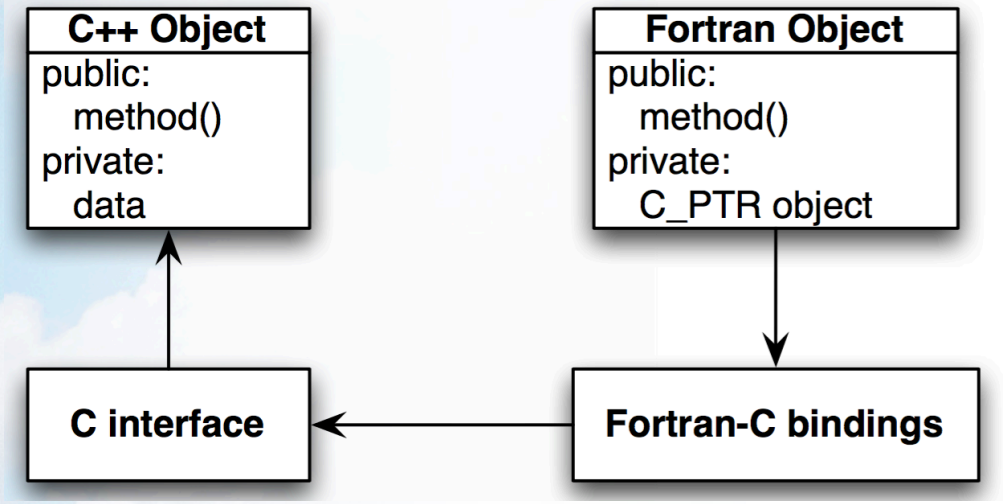
Local computations in every subdomain



Same Equal-Area Domain decomposition
(T255 – 96 partitions)

So what about Fortran?

- Keep Fortran for numeric algorithms
- C++ for …
  - All data & memory management
  - Message passing
- For every C++ class -> Fortran Derived Type
- Use Fortran ISO C Bindings (F2003)



C++ contains implementation

```
// C++ implementation
class Field {
public:
        Field(int size) { data_.resize(size); }
        double* data() { return &data_[0]; }
private:
        vector<double> data_;
}

// C interface – to be called from Fortran
Field* atlas__new_Field (int size) {
        return new Field( size );
}
void atlas__Field__access_data(Field* this,
                               double& data,
                               int& s){
        data = this->data();
        size = this->size();
}
```

WARNING: Fortran code ahead!

Fortran/C interface

```
! Fortran Derived type , contains no data , only methods F03
TYPE :: Field_type
PRIVATE
TYPE(C_PTR) :: object = C_NULL_PTR
CONTAINS
  PROCEDURE, PUBLIC :: access_data => Field__access_data
END TYPE

! Method calls C function, which uses C++ implementation
SUBROUTINE Field__access_data(this,data)
  CLASS(Field_type), intent(in) :: this
  TYPE(C_PTR) :: cptr_data
  REAL(JPRB), POINTER :: data
  INTEGER :: size
  ! This is the C function
  CALL atlas__Field__access_data(this%object ,cptr_data ,size)
  ! Converts the C pointer to a fortran pointer to array of doubles
  CALL C_F_POINTER(cptr_data ,data ,(/size/))
END SUBROUTINE
```

Fortran Program

```
PROGRAM main

  TYPE(Field_type) :: field
  REAL(JPRB), POINTER :: field_data(:)
  INTEGER :: j

  ! We can ask the field for access to its data
  CALL field%access_data(field_data)

  ! Now we can use the data as any other Fortran array
  DO j=1,size(field_data)
    field_data(j) = 0.
  ENDDO

END PROGRAM
```

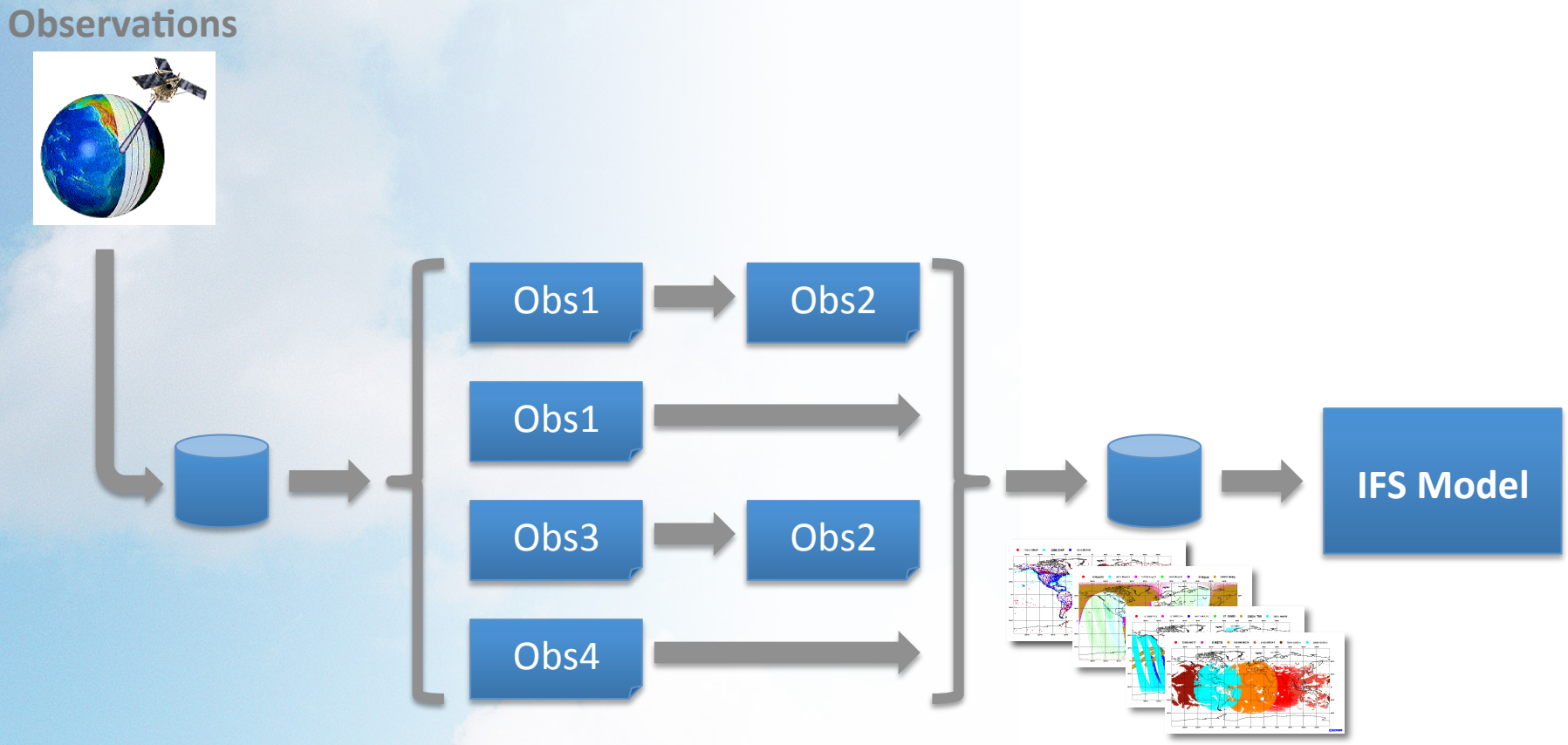
- Furthermore, we provide another higher level API to hide Fortran 2003.

Hermes

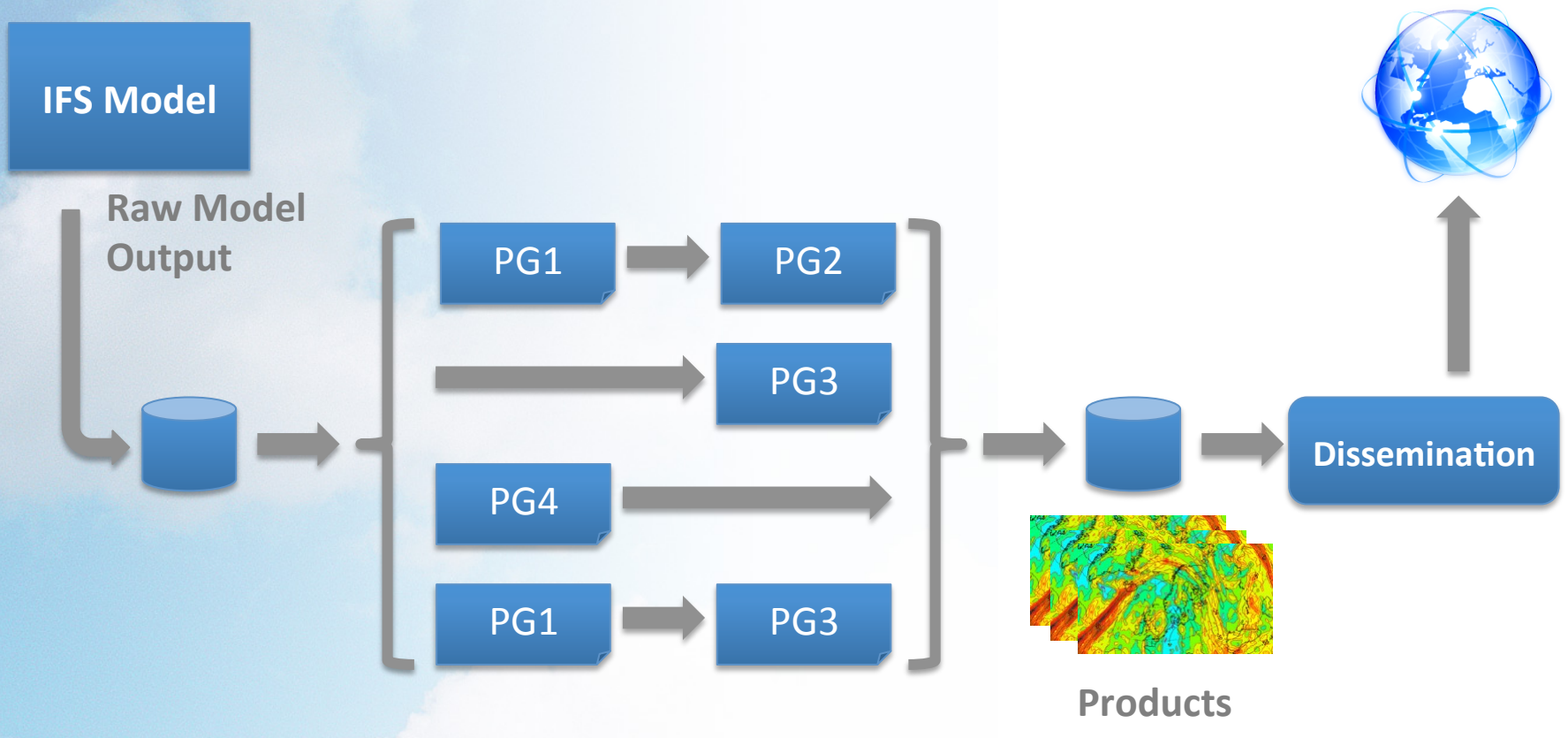
T. Quintino, B. Raoult, F. Rathgeber

Challenges: Big Data & Platform Uncertainty

A look at the data-chain



A look at the data-chain



Observations & Fields: Some Similarities

- Both processed in pipelines
- Both are parallel…
  - Concurrent (Threads / OpenMP)
  - Distributed (MPI)

How does it **differ** from an atmospheric Model?

- Parallel, but little or no synchronisation
- No time evolution
- "Embarrassingly parallel" (nearly)

Possible: *Automatic Parallelisation*

Requirements

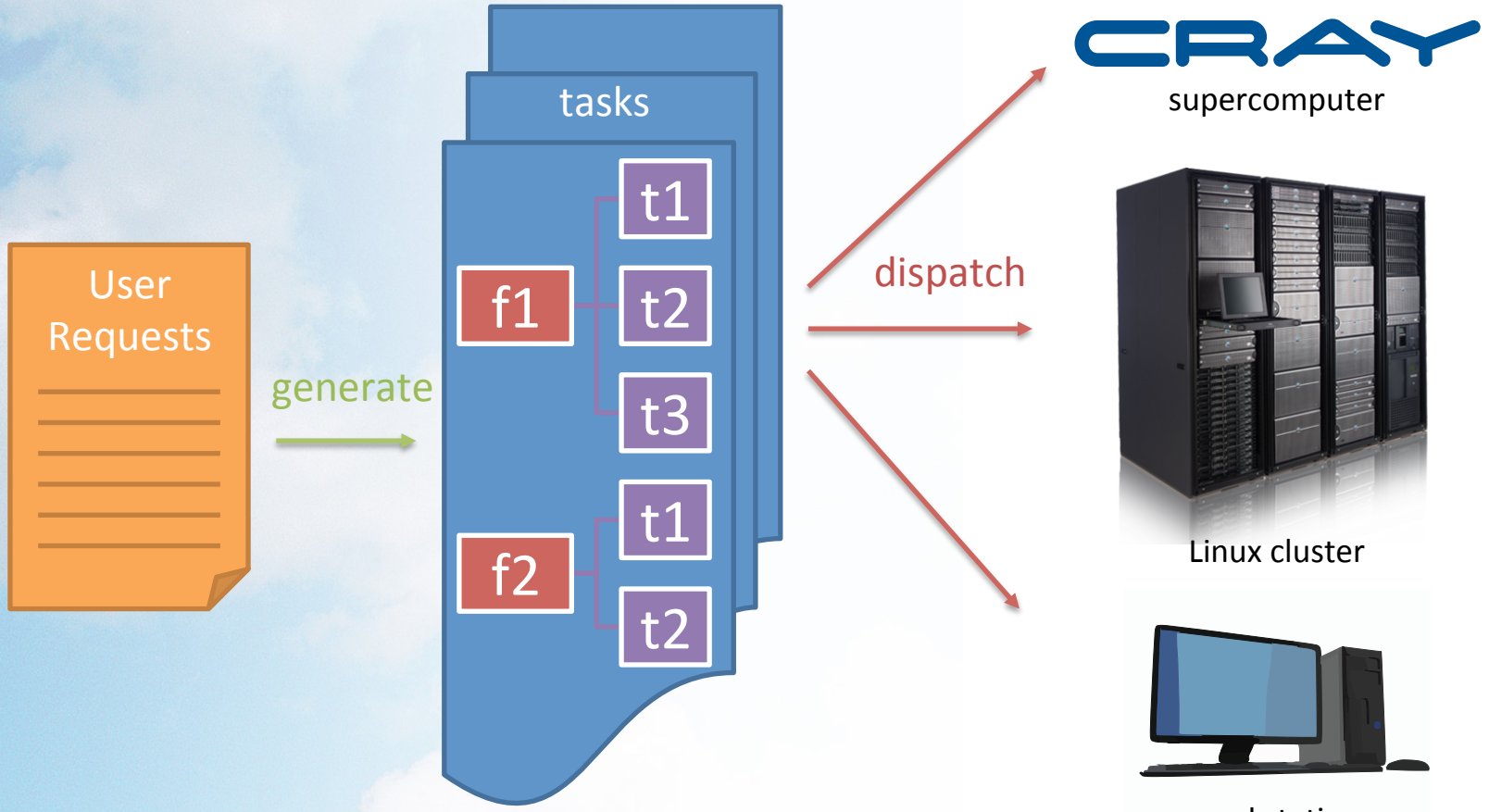
- Way to define computations
- Efficient parallel computation
- Way to accept data and deliver results
- Flexibility for future architectures
- Extendable and understandable by scientists

Solution

Indirection is key to Scalability

Decouple **what** to compute
from **how** and **where** to compute

Use Case: Product Generation

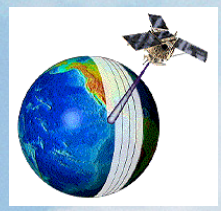



```
map ( interpolate,
      list ( f1, f2, ... ) )
```

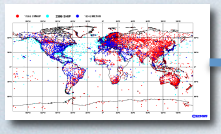
Frontend (define)

Backend (compute)

Use Case: Observation Filters

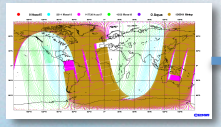


observations



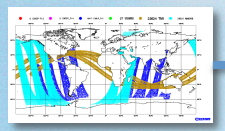
map (f, list (…))

f



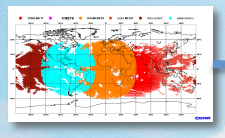
map (g•f, list (…))

f → g



map (g, list (…))

g



map (h•g•f, list (…))

f → g → h

dispatch

supercomputer

Linux cluster

workstation

Frontend (define)

Backend (compute)

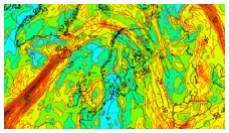
Use Case: Workstation Interpolation & Visualisation

workstation

Fields
T8000
(N x 670MB)

dispatch

Fields
N640
(N x 10MB)



Interpolation cluster

```
map ( interpolate,
      list ( … ) )
```

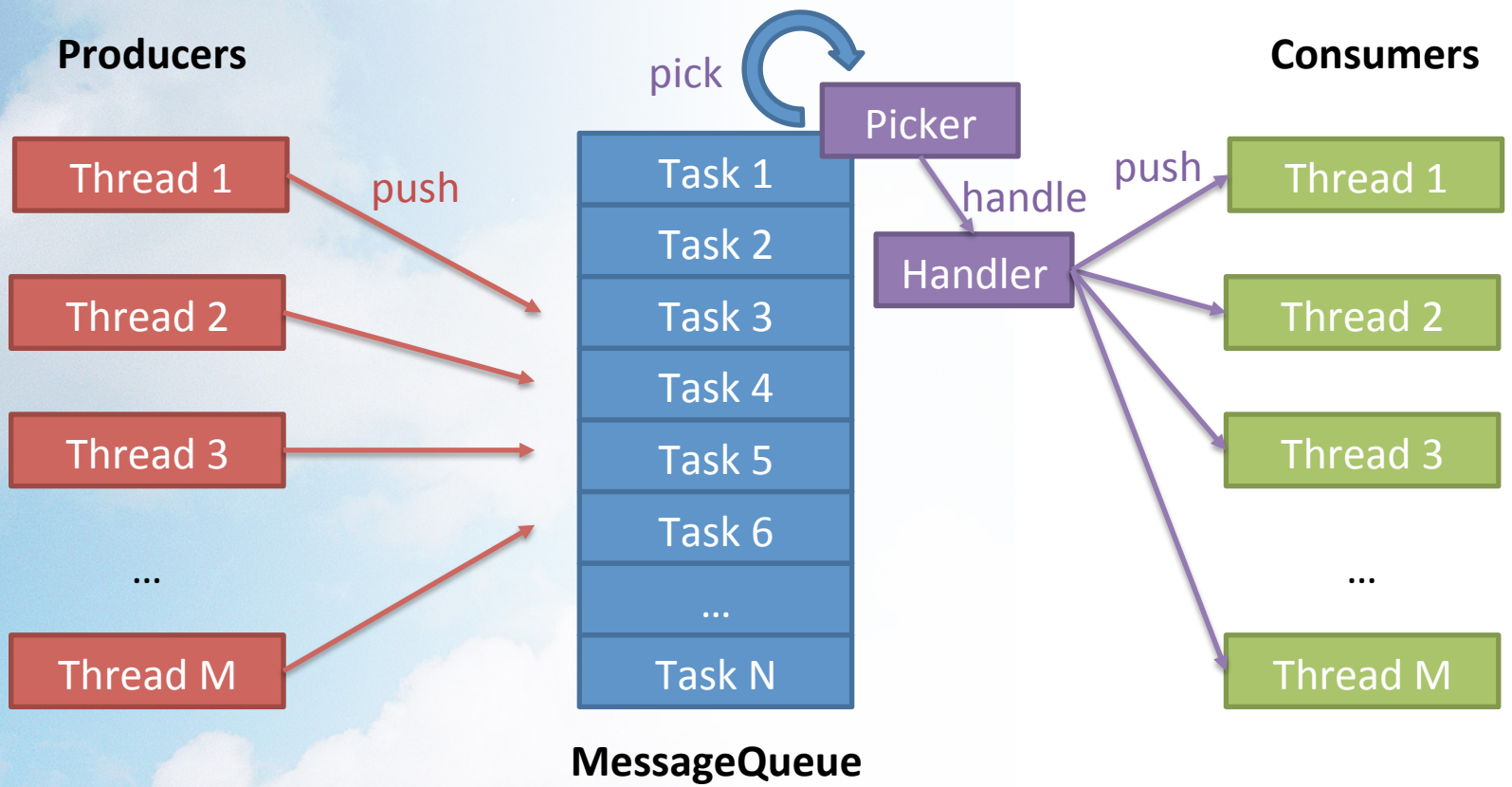
Frontend (define)

Backend (compute)

Hermes

- DSL to define computations
- Frontend for sending requests and data
- Broker to dispatch work to PU's
- Multiple computational back-ends:
  - CPU
  - CUDA (Linear Algebra)
  - Intel MKL (Linear Algebra)
  - *Intel TBB ??*
  - *HPX ??*

Dispatcher


Producers
Thread 1
Thread 2
Thread 3
...
Thread M
push
pick
Picker
Task 1
Task 2
Task 3
Task 4
Task 5
Task 6
...
Task N
MessageQueue
handle
Handler
push
Consumers
Thread 1
Thread 2
Thread 3
...
Thread M

Expression Engine

- Prototype DSEL
- Embedded in C++
- *Plug-ins in Fortran (not yet)*
- Computations as expression trees
- Hierarchy of expressions:
  - terminals (value semantics)
  - functions (pure operations)

Expression Engine

- Inspired by functional language **Haskell**
- Expressions immutable
- **Functions pure**
- No variables, only constants
- **Lazy** evaluation
- Optimiser: find most efficient path to evaluate expression
  - Pattern matching (e.g. linear combination)
  - Collapse expression tree into terminal
  - Detect common sub-expressions
  - *Automatic parallelisation*

Expression

```
typedef std::shared_ptr<Expression> ExpPtr;
typedef std::vector< ExpPtr > args_t;

class Expression {
public:
  ...
  Expression(const args_t& args);
  size_t arity() const { return args_.size(); }
  ...
  ExpPtr self();
  ExpPtr eval() const { Scope s; return ExpPtr(optimise()->evaluate(s)) };
  ExpPtr eval(Scope& s) const {  return ExpPtr(optimise()->evaluate(s)) };
  ...
  // used to bind lambda parameters
  virtual ExpPtr resolve(Scope&) const { return self(); }
  virtual ExpPtr optimise(size_t depth) const { return self(); }
  template<typename T> shared_ptr<T> as() { ... }
private:
  virtual ExpPtr evaluate(Scope&) const = 0;
  virtual ExpPtr cloneWith(args_t&) const = 0;
  virtual std::string signature() const = 0;
};
```

Values

```
class Value : public Expression {
protected:
  Value( const args_t& args ) : Expression(args) {}
private:
  virtual ExpPtr evaluate(Scope&) const { return self(); }
};


class Real : public Value {
public:
  typedef real_t value_t;
  ...
  Real(ExpPtr e);
  Real(const real_t& v);
  ...
  value_t value() const { return v_; }
  static value_t extract (const ExpPtr& e) { return e->as<Real>()->value();}
  ...
private:
  virtual std::string signature() const { return "r"; }
};
```

Bool, Integer, Real, String, List, Vector, Matrix

Functions

```
class Function: public Expression {
public:

  typedef ExpPtr (*func_t) (Scope&,const args_t&);
  typedef std::map< key_t, func_t > dispatcher_t;

  virtual ExpPtr evaluate(Scope&) const;
  virtual ExpPtr optimise(size_t depth) const;

  virtual std::string signature() const { return signatureArguments(args()); }

protected:
  Function( const args_t& args ) : Expression(args) {}
  // returns something like "func(r,v,r,v)"
  std::string signatureArguments(const args_t& args) const {...}

private:
  virtual ExpPtr evaluate(Scope&) const { return self(); }
};
```

Functions (2)

```
ExpPtr Function::evaluate(Scope& ctx) const {

  // create temporary args
  args_t tmp = args();
  for(size_t i = 0; i < tmp.size(); ++i) {
    tmp[i] = args(i, ctx, true)->self(); // evaluate the arguments
  }

  std::string sig = signatureArguments(tmp); // new signature

  dispatcher_t& d = dispatcher();  // find dispatching function
  dispatcher_t::iterator itr = d.find(sig);
  if(itr == d.end())  { ... fail ... }

  return ((*itr).second)( ctx, tmp ); // dispatch
}

ExpPtr Function::optimise(size_t depth) const {
  // revert to generic optimizer if Expression doesn't optimize itself
  return Optimiser::apply(self(), depth);
}
```

Functions (3)

```
class Map : public Function {
public:
  Map(ExpPtr func, ExpPtr lst );

private:
  ...
  ExpPtr Map::evaluate(Scope& ctx) const {

    ExpPtr f  = args(0, ctx, false); // get the function, don't evaluate
    ExpPtr ls = args(1, ctx, true);  // get the list and evaluate if necessary

    const List::value_t& list = List::extract(ctx, ls);

    List::value_t res;
    res.reserve(list.size());
    for( size_t i = 0; i < list.size(); ++i ) {
       ExpPtr e = list[i]->resolve(ctx)->eval(ctx); // evaluate each element
       ExpPtr v = f->eval(e);
       res.push_back( v );
    }
    return ExpPtr(new List(res));
  }
  ...
};
```

Functions (4)

Map
Reduce
Merge
Take
ZipWith
IfElse
Call
Lambda

Filter
Bind
UnaryOperators
- neg,sqrt,abs

BinaryOperators
- mul,add,sub,div
- mod,min,max

Predicates

Examples

```
auto a = real(2.);
auto b = real(4.);
auto x = vector({1.,2.,3.});
auto y = vector(3, 5.);

auto c = add(a,b);

std::cout << *c << std::endl; // Add(Real(2), Real(4))

auto e = add(mul(c, x) , mul(b, y));

c = c->eval();

std::cout << *c << std::endl; // Real(6)

// Add(Mul(Add(Real(2), Real(4)), Vector(1, 2, 3)), Mul(Real(4), Vector(5, 5, 5)))
std::cout << *e << std::endl;

auto opt = e->optimise(true);

std::cout << opt->signature() << std::endl; // optimised to "Linear(r,v,r,v)"

auto f1 =  reduce(mul(), list(a, x, x))->eval();   // Vector(2, 8, 18)
```

Examples

class **Xpr** provides value sematics and convinience operators

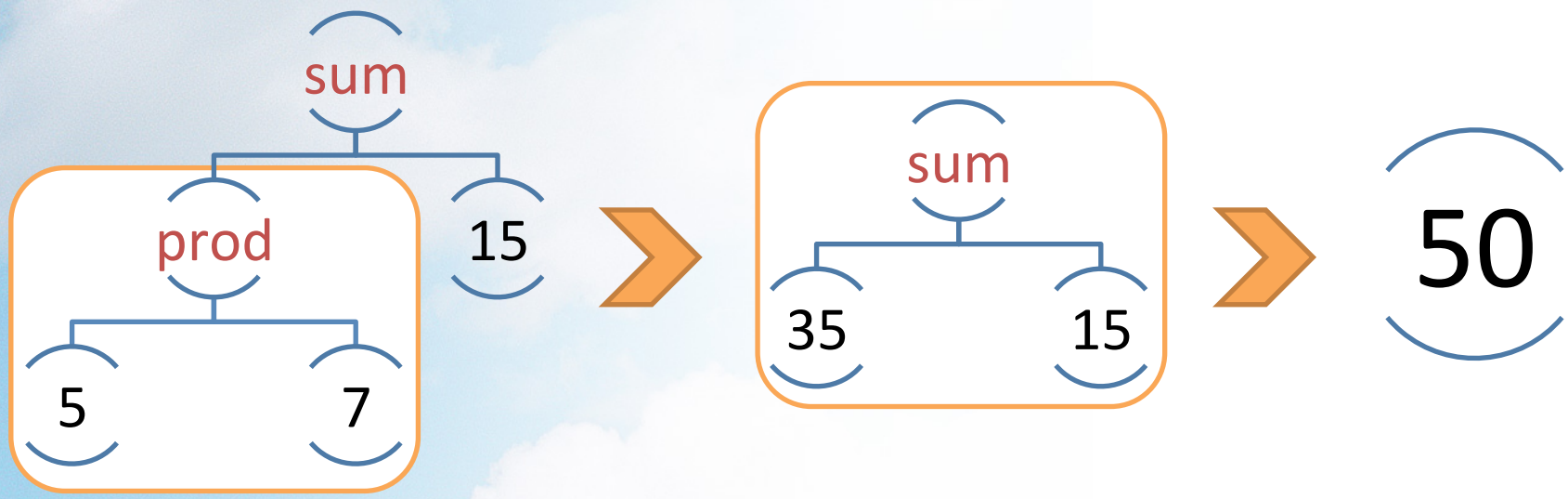
```
// lambdas

Xpr twice = call(lambda("x", Xpr(2.0) * Xpr("x"))); // define 2*x

Xpr l = list(a,b,c);

Xpr r = map(twice, l);

std::cout << r() << std::endl; // List(Real(4), Real(8), Real(12))

// currying

Xpr F = lambda("i", call( lambda("j", Xpr("i") * Xpr("j")))); // define i*j

Xpr H = call(F, Xpr( 2. ));    // curry to i*2

Xpr G = H( 3. );               // curry to 3*2

std::cout << G() << std::endl; // Real(6)
```

Examples

```
// Define the Y-combinator

Xpr Y = lambda("f", call(Xpr("f"), Xpr("f")));

// lookup table for first 10 elements of series
Xpr table = list( Xpr(0.), Xpr( 1.), Xpr( 1.), Xpr( 2.), Xpr( 3.), Xpr(5.),
                  Xpr(8.), Xpr(13.), Xpr(21.), Xpr(34.), Xpr(55.) );

// non-recursive fibonacci function
Xpr fib = lambda("f",
                 call(lambda("n",
                             ifelse(
                                 Xpr("n") > Xpr(10.),
                                 Xpr(call(Xpr("f"),Xpr("f"), Xpr("n”)-Xpr(1.0))) +
                                 Xpr(call(Xpr("f"),Xpr("f"), Xpr("n”)-Xpr(2.0))),
                                 take( Xpr("n"), table )
                                 )
                              )
                       )
                  );

Xpr fibonnaci = call(Y, fib); // applying the Y-combinator

std::cout << fibonnaci(20.) << std::endl; // Real(6765)
```

Arithmetic Expression

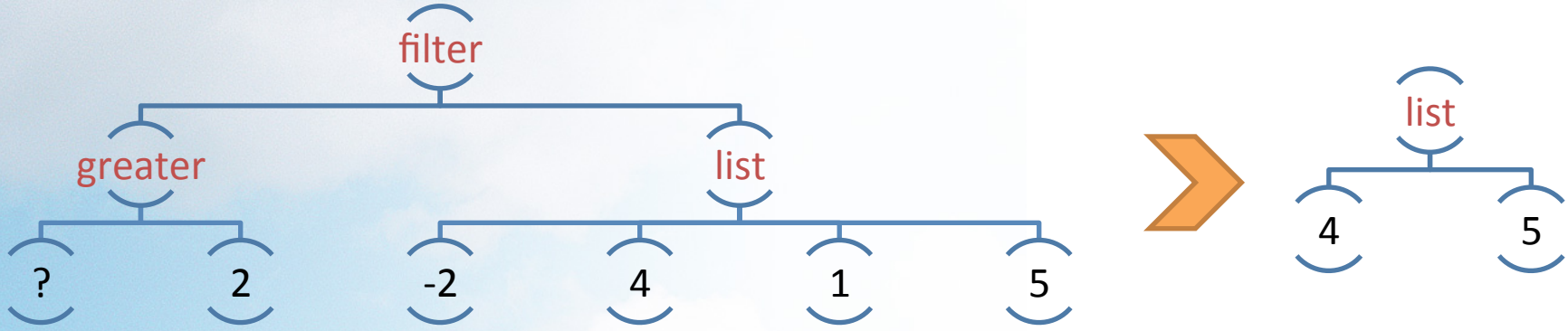
```
5 * 7 + 15 == add ( prod ( 5, 7 ), 15 )
```



constant folding and propagation

Filter Expression

```
filter ( greater ( ?, 2 ), list ( -2, 4, 1, 5 ) )
```



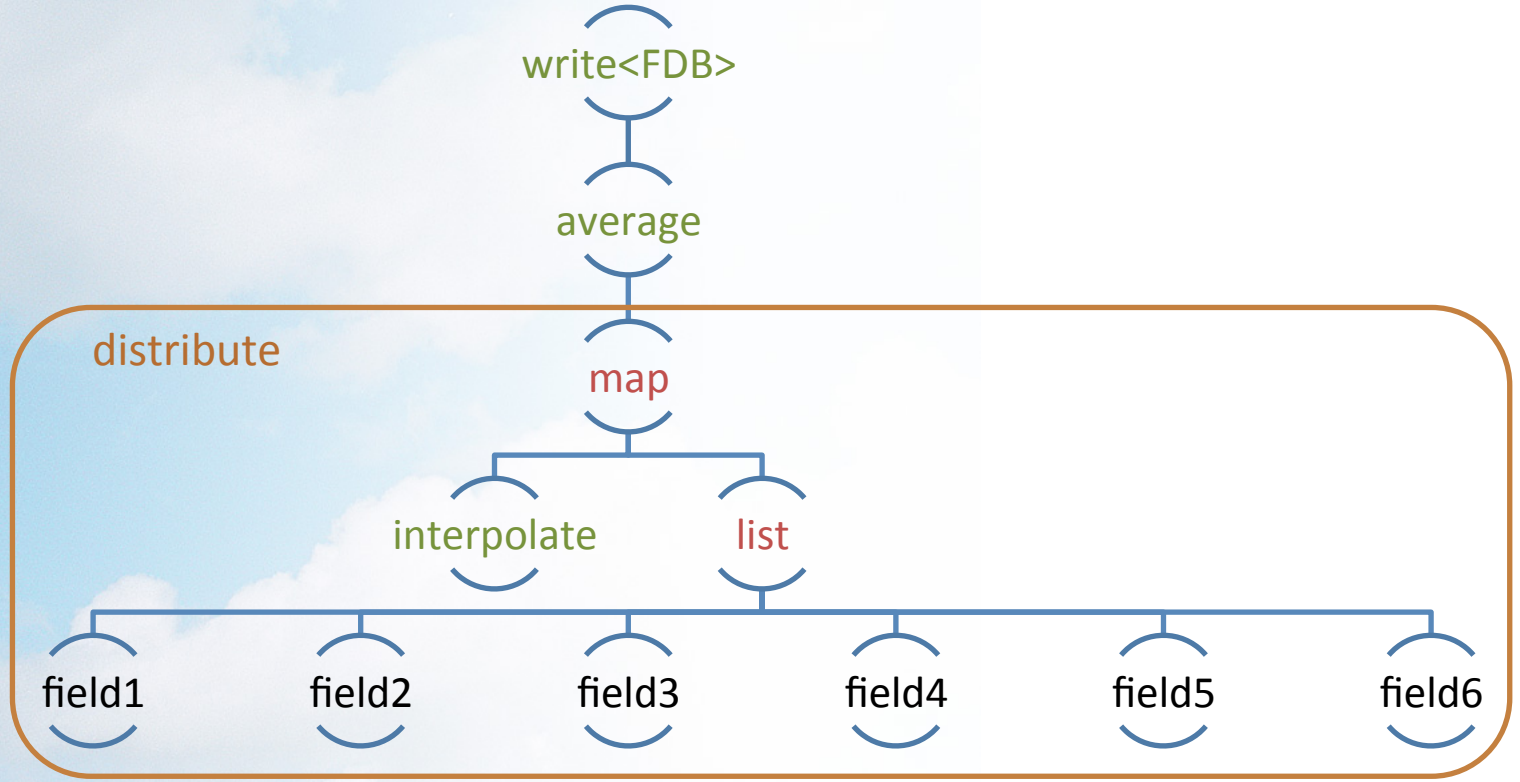
expression elimination

Future work: Map fusion

```
map ( h, map ( g, map ( f, list ( … ) ) ) )
```

Future work
Distributed interpolation of fields

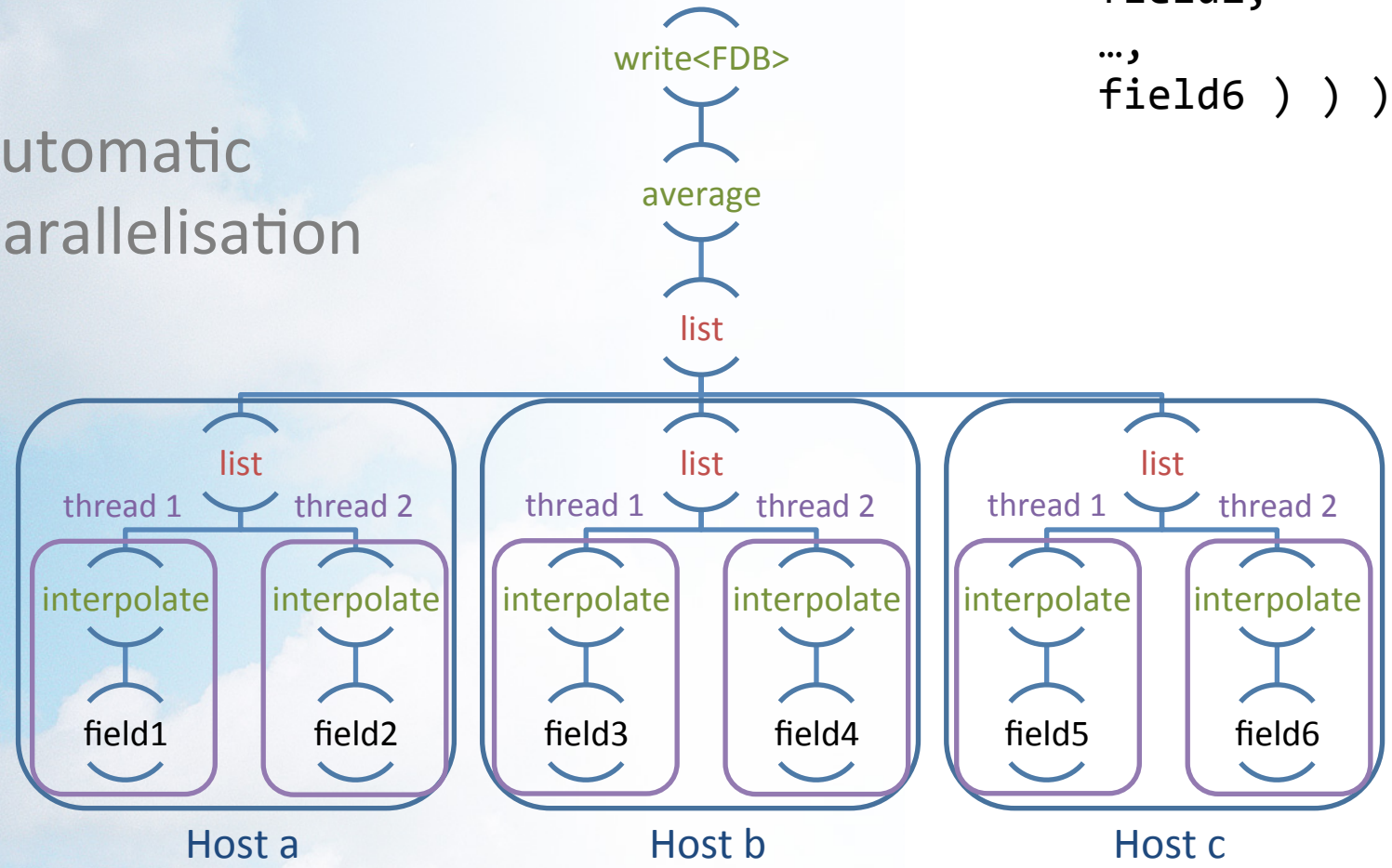
```
write<FDB> ( average ( map ( interpolate, list ( field1,
                                                 field2,
                                                 …,
                                                 field6 ) ) ) )
```



Future work
Distributed interpolation of fields

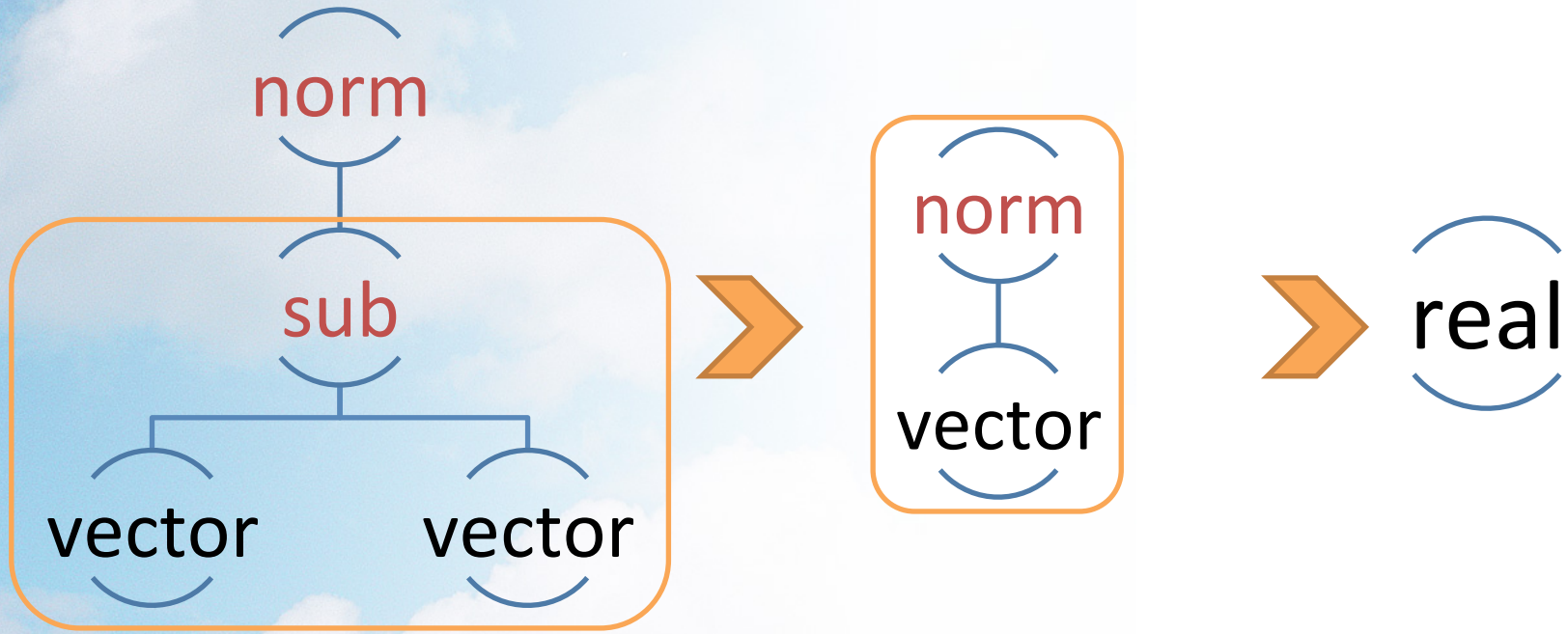
```
write<FDB> ( average ( map ( interpolate, list ( field1,
                                                 field2,
                                                 …,
                                                 field6 ) ) ) )
```

automatic parallelisation



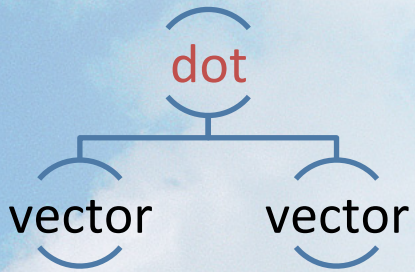
Linear Algebra Expressions

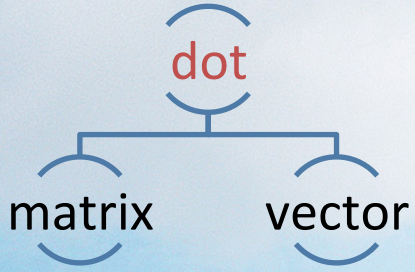
```
norm ( sub ( vector, vector ) )
```

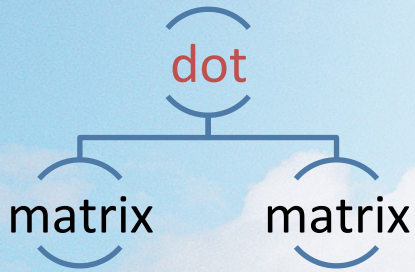


norm of difference of two vectors

Dispatch to Compute Engines







Expression signatures

```
Dot(v,v)
Dot(m,v)
Dot(m,m)
```

dispatch

GPU engine

```
cublasDdot
cublasDgemv
cublasDgemm
```

MKL engine

```
cblas_ddot
cblas_dgemv
cblas_dgemm
```

Generic engine

```
ddot
dgemv
dgemm
```

BLAS/ LAPACK

Expressions: client code

```
const size_t ncols = 1024;
const size_t nrows = 1024;

auto init = [](size_t, size_t j=0) {
  return (real_t)std::rand() / RAND_MAX;
};

XprClient c;

// L2 norm of difference of 2 vectors
c.send( norm( sub( vector(ncols, init), vector(ncols, init) ) ) );

// Dot product of vectors
c.send( dot( vector(ncols, init), vector(ncols, init) ) );

// Dot product of matrix and vector
c.send( dot( matrix(nrows, ncols, init), vector(ncols, init) ) );

// Dot product of matrix and matrix
c.send( dot( matrix(nrows, ncols, init), matrix(ncols, nrows, init) ) );
```

Generic Engine implementation

```
ExpPtr compute_dot_generic(Scope&, const args_t &p)
{
    Vector::value_t v1 = Vector::extract(p[0]);
    Vector::value_t v2 = Vector::extract(p[1]);
    ASSERT( v1.size() == v2.size() );
    Vector::elemt_t r;
    for( size_t i = 0; i < v1.size(); ++i )
        r += v1[i] * v2[i];
    return real(r);
}

ExpPtr compute_gemv_generic(Scope&, const args_t &p);
ExpPtr compute_gemm_generic(Scope&, const args_t &p);

// Register generic functions in dispatch table
struct DotRegister {
    DotRegister() {
        Function::dispatcher()["xpr::Dot(v,v)"] = &compute_dot_generic;
        Function::dispatcher()["xpr::Dot(m,v)"] = &compute_gemv_generic;
        Function::dispatcher()["xpr::Dot(m,m)"] = &compute_gemm_generic;
    }
};
Static DotRegister dotRegister;
```

MKL Engine implementation

```
ExpPtr compute_dot_MKL(Scope&, const args_t &p)
{
  const Vector::value_t& v1 = Vector::extract(p[0]);
  const Vector::value_t& v2 = Vector::extract(p[1]);
  ASSERT( v1.size() == v2.size() );
  Vector::elemt_t r;
  return real( cblas_ddot( v1.size(), v1.data(), 1, v2.data(), 1 ) );
}

ExpPtr compute_gemv_MKL(Scope&, const args_t &p)
ExpPtr compute_gemm_MKL(Scope&, const args_t &p)

// Register MKL functions in dispatch table
void registerMKL() {
  Function::dispatcher()["xpr::Dot(v,v)"] = &compute_dot_MKL;
  Function::dispatcher()["xpr::Dot(m,v)"] = &compute_gemv_MKL;
  Function::dispatcher()["xpr::Dot(m,m)"] = &compute_gemm_MKL;
}
```

GPU Engine implementation

```
ExpPtr compute_dot_CUDA(Scope&, const args_t &p)
{
        const Vector::value_t& x = Vector::extract(p[0]);
        const Vector::value_t& y = Vector::extract(p[1]);
        ASSERT( x.size() == y.size() );
        const size_t size = x.size()*sizeof(real_t);
        Vector::elemt_t r;

        real_t* d_x; ///< device memory vector x
        real_t* d_y; ///< device memory vector y
        cublasHandle_t handle;

        CALL_CUDA( cudaMalloc((void**) &d_x, size) );
        CALL_CUDA( cudaMalloc((void**) &d_y, size) );
        CALL_CUDA( cudaMemcpy(d_x, x.data(), size, cudaMemcpyHostToDevice) );
        CALL_CUDA( cudaMemcpy(d_y, y.data(), size, cudaMemcpyHostToDevice) );

        CALL_CUBLAS( cublasCreate(&handle) );
        CALL_CUBLAS( cublasDdot(handle, x.size(), d_x, 1, d_y, 1, &r) );
        CALL_CUBLAS( cublasDestroy(handle) );

        CALL_CUDA( cudaFree(d_x) );
        CALL_CUDA( cudaFree(d_y) );

        return real( r );
}
```

Live Demo

Questions?