ECMWF's New Product Generation

Lessons learned from development to operations

T. Quintino, B. Raoult, M. Zink, P. Maciel

ECMWF

tiago.quintino@ecmwf.int

ECMWF’s Forecasting Systems

What do we do?

Operations – **Time Critical**

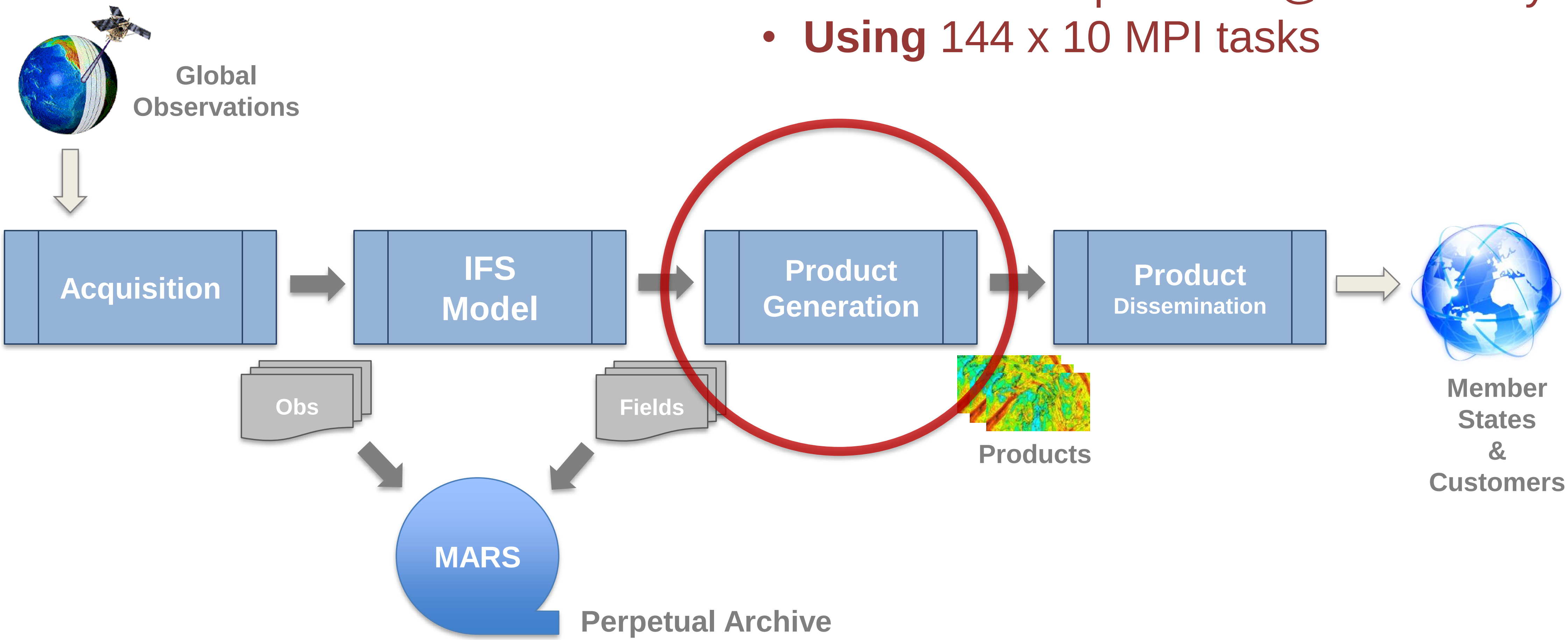
- HRES 0-10 day, 00Z+12Z
  - O1280 (9km) 137 levels
- ENS 0-15 day, 00Z+12Z
  - O640 (18km) 91 levels
- ENS extended 16-46 day, twice weekly
  - O320 (36km) 91 levels
- BC 06Z and 18Z
  - hourly post-processing 0-5 days

Research – **Non Time Critical**

- Experiments to improving our models
- Reforecasts, Climate reanalysis, etc

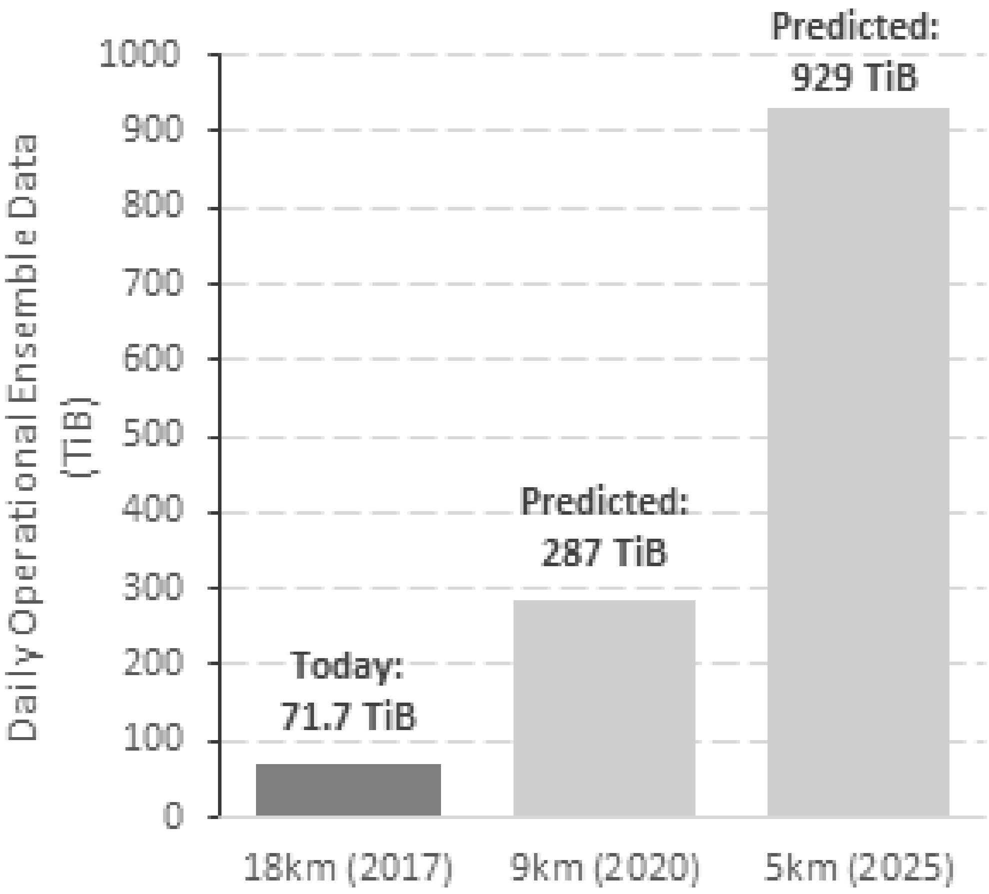
ECMWF's Production Workflow

- **From** 288 million fields @ 90 TiB/day
- **To** 230 million products @ 30 TiB/day
- **Using** 144 x 10 MPI tasks



Data Growth – History and Projections

Historical Growth of Generated Products



Model Output Projected Growth

Venerable **ProdGen**

- **Legacy code**
  - Written in Fortran 77/90
  - Continuously tweaked, No configuration files
  - Performance was acceptable but **not optimal**
  - **Aborted** on first error
  - Grown organically over decades without a “fresh rethink”

- **Limited Reproducibility**
  - Source code versioning was inconsistent
  - Workflow / Suite was maintained manually
  - Production was **not transactional,** could lead to data corruption

New **PGen**

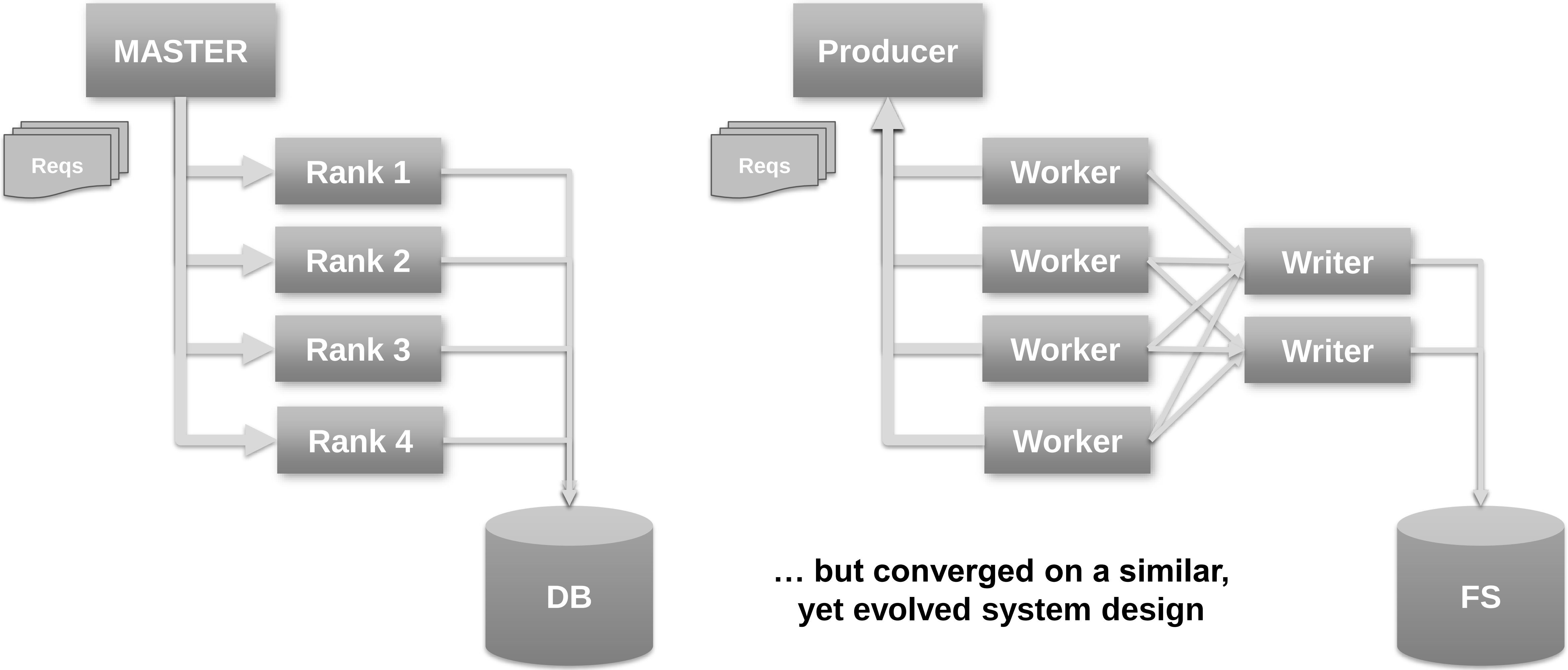
- **New code**
  - Written in C++ 11
  - Optimized for **minimal work**
  - Configuration driven
  - **Resilient** to errors
  - New design, Based on **past experience**
  - **I/O server** for reduced I/O impact

- **Full Reproducibility**
  - Automated system from versioning to testing to production
  - Workflow / Suite was automatically generated from source
  - **Transactional** production

FRESH START

Architecture: ProdGen vs PGen

Explored different designs and architectures…



… but converged on a similar, yet evolved system design

Be bold in new directions,

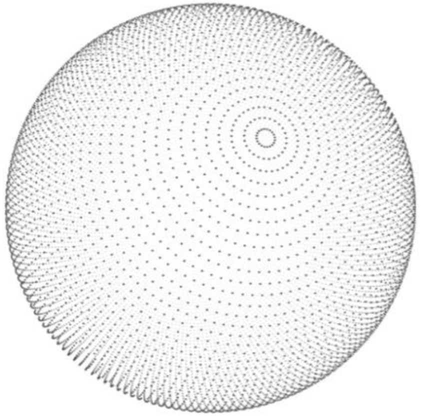
But respect and learn from the past

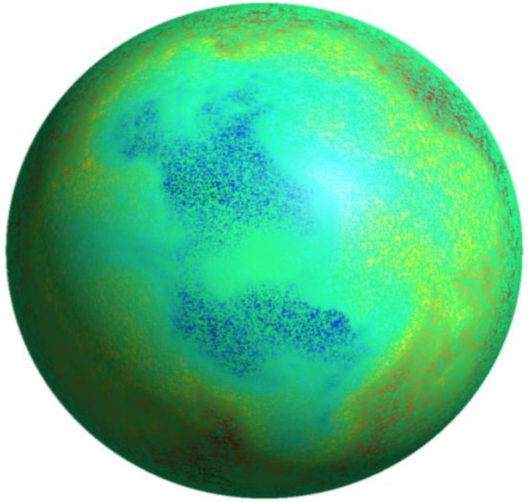
MIR - ECMWF’s New Interpolation

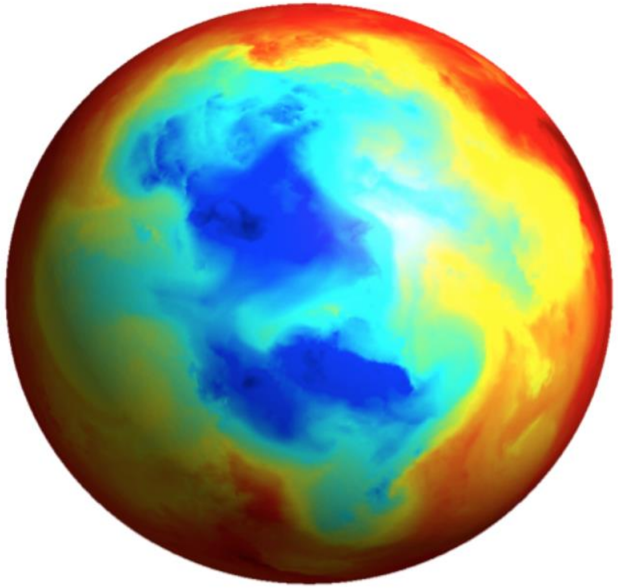
- **Flexible** and maintainable design
  – Rewrite in C++
  – Configuration driven

- Kernel based on linear **Interpolation Operators**

- **Programmable Pipelines**







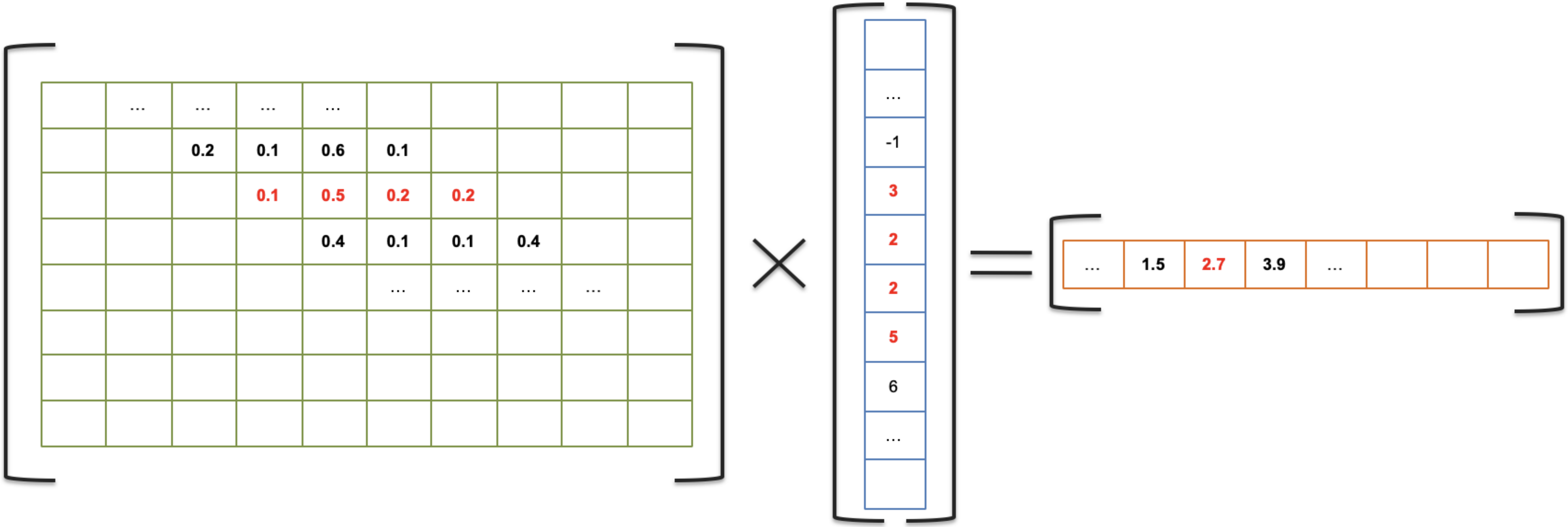
ECMWF

Optimizing Interpolation

- Kernel is a Matrix-Vector multiply
- **Deterministic Interpolation** operators
  - Enabling caching of operators

$$F_i = \Sigma\, w_{ij} G_j$$

- Use of highly optimized **Linear Algebra** libraries (**BLAS & MKL**)



Performance

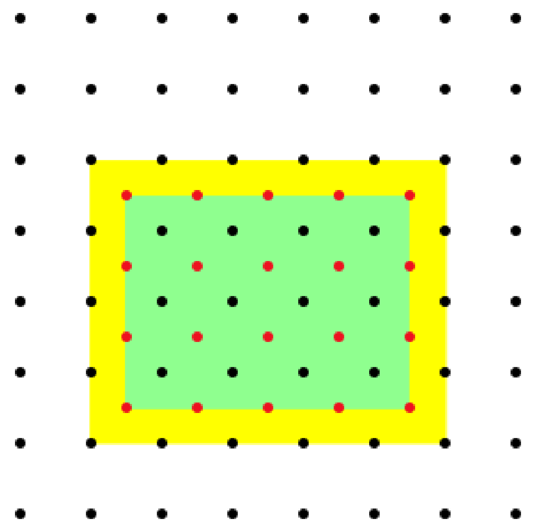
Scalability

Grid	N Points	Memory [GiB]	Wall Time [ms]	Speed [Mp/s]
N160	204 k	1.7	28.4	**7.2**
N256	524 k	1.8	33.0	**15.9**
N512	2097 k	1.8	51.2	**40.9**
LL 0.1/0.1	6483 k	2.6	99.9	**64.9**
N1024	8388 k	2.7	115.4	**72.7**
LL 0.05/0.05	25 927 k	6.1	252.2	**102.8**

Beware of the Floating Point

Users request a specific area of interest…

Lat <= 89.5 & Long 0.5



AREA=89.5/0.5/-89.5/359.5,
GRID=1.0/1.0

However **Floating Point** arithmetic…

- has **finite precision**
- is **non-associative   (a + b) + c != a + (b + c)**
- some optimizations **relax truncation error assumptions**

Beware of the Floating Point (2)

Assume

$$R = A * B + C * D$$

IF **A = C** and **B = - D**

Then R = 0 ...

```
double c = sin(13./7.);
double d = tan(3./17.);
double a = c;
double b = -d;
```

```
[17:02:14
0.9592821
0
```

```
[17:01:32
0.9592821
9.0361908148775841e-18
```

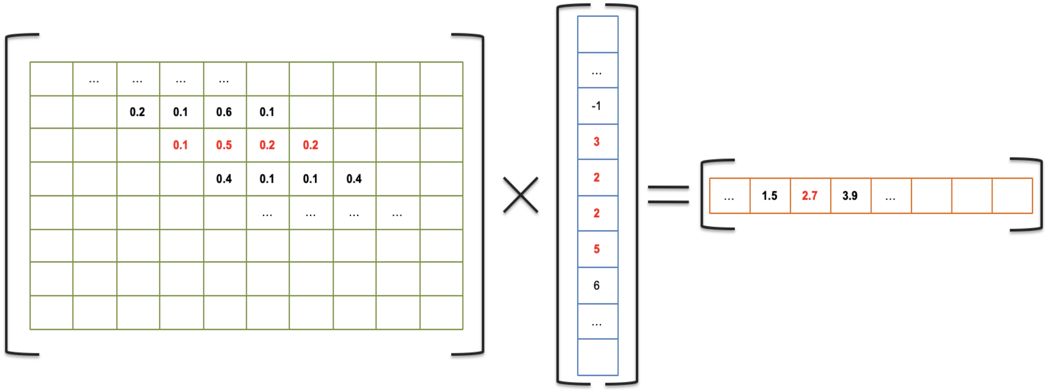
```
...
104 0180 C5FD2884        vmovapd -1920016(%rbp,%rax), %ymm0
105 0189 C5FD28A4        vmovapd -960016(%rbp,%rax), %ymm4
106 0192 C5FD5984        vmulpd  -2400016(%rbp,%rax), %ymm0, %ymm0
107 019b C4E2DDB8        vfmadd231pd     -1440016(%rbp,%rax), %ymm4, %ymm0
108 01a5 C5FD2984        vmovapd %ymm0, -480016(%rbp,%rax)
 ...
```

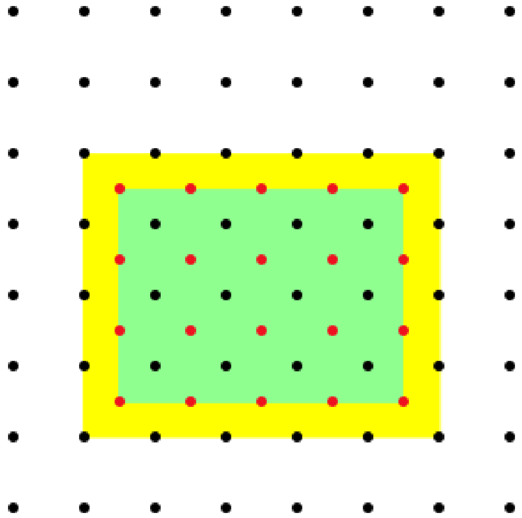
Fused multiply-add

PGen Optimizations

- Fast **Linear Algebra**
  - Use Intel® MKL
  - Activate most FP optimizations

- Correct and **Deterministic**
  - Disable some optimizations
    - Fused multiply-add
    - Defined locations as **Factions or Integers**
      - **A / B**





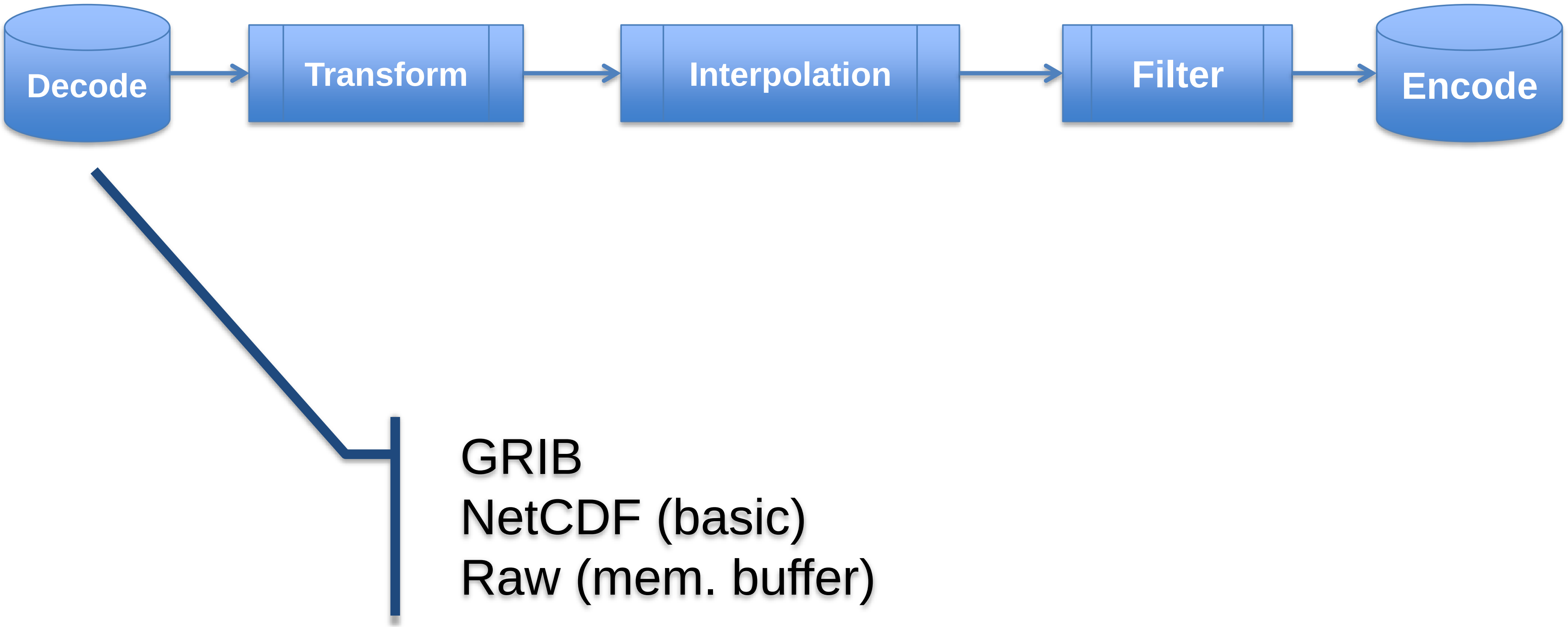
Be mindful of cross-architecture reproducibility:

Be fast where you can,

But don’t sacrifice determinism

Programmable Pipeline Architecture

Construct an Action Plan



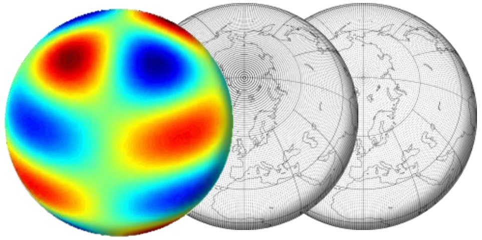
GRIB
NetCDF (basic)
Raw (mem. buffer)

Programmable Pipeline Architecture

Construct an Action Plan

Decode → Transform → Interpolation → Filter → Encode

SH to SH,
VOD-2-UV
SH to Grid



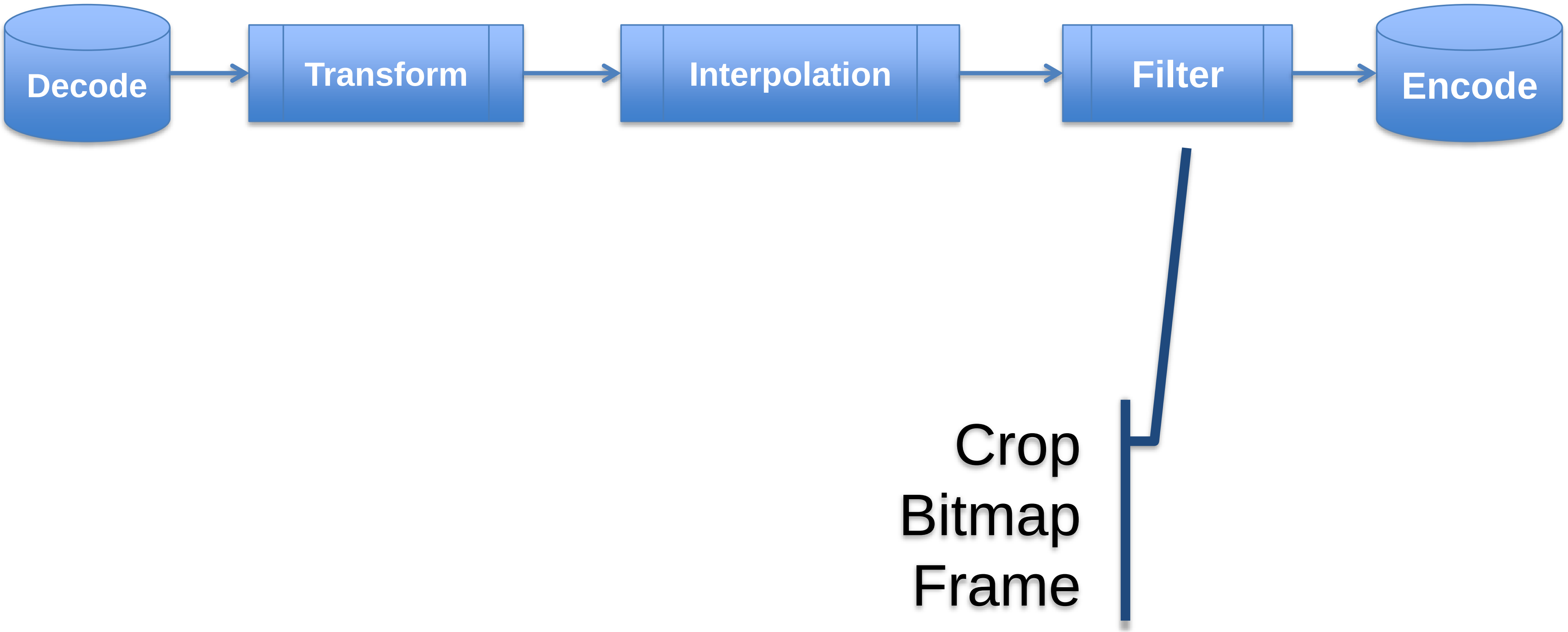
Programmable Pipeline Architecture

Construct an Action Plan

Compute Interpolation Operator
Caching of operators
Linear algebra kernel

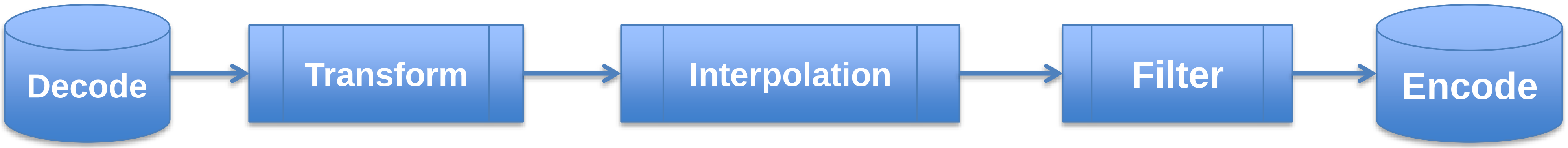
Programmable Pipeline Architecture

Construct an Action Plan



Programmable Pipeline Architecture

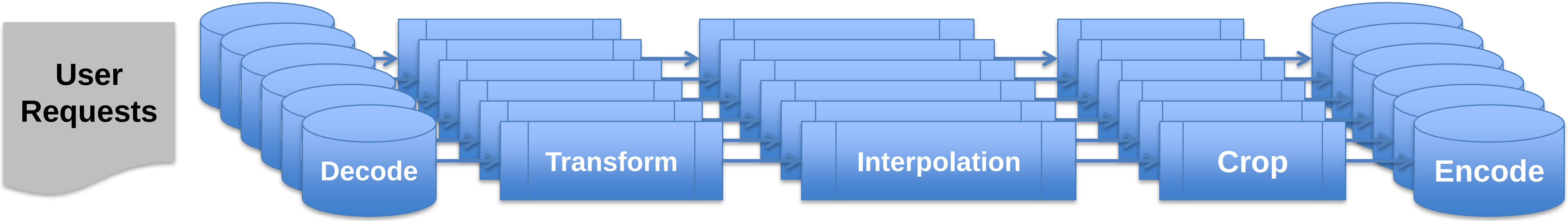
Construct an Action Plan



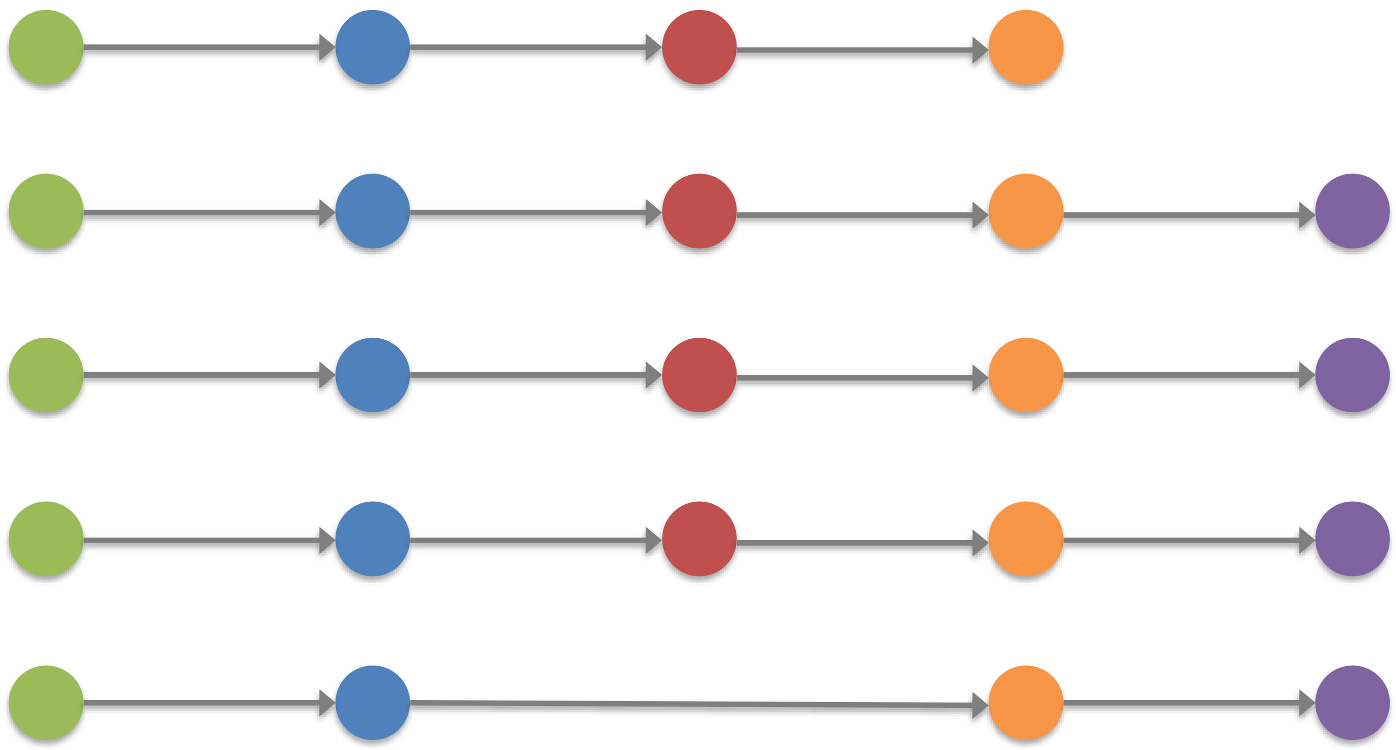
GRIB
Accuracy
Compression

Product Generation – PGen / MIR

- Explicit **Task Graph** analysis
  - ***Users can update requests daily***
  - Factorise common tasks
  - Batch and Reorder execution
  - Compute time-series on-the-fly

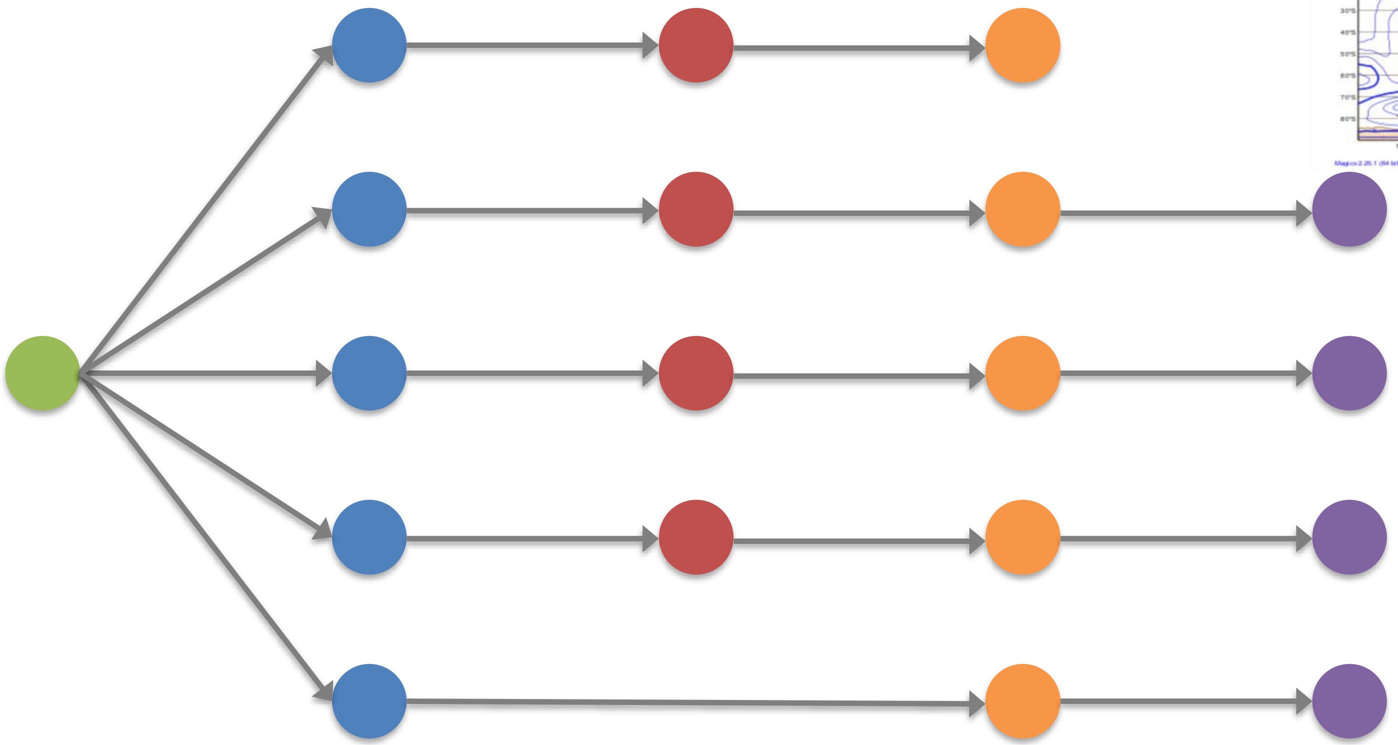


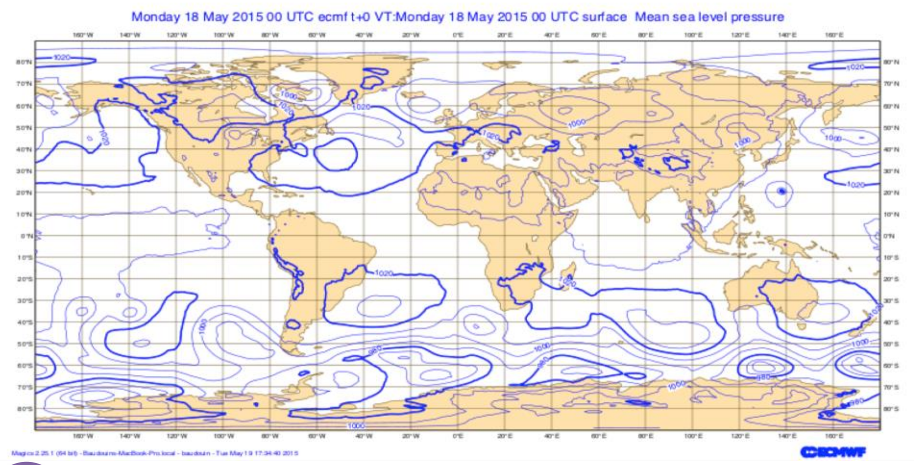
PGen – Task Graph Analysis



PGen – Task Graph Analysis

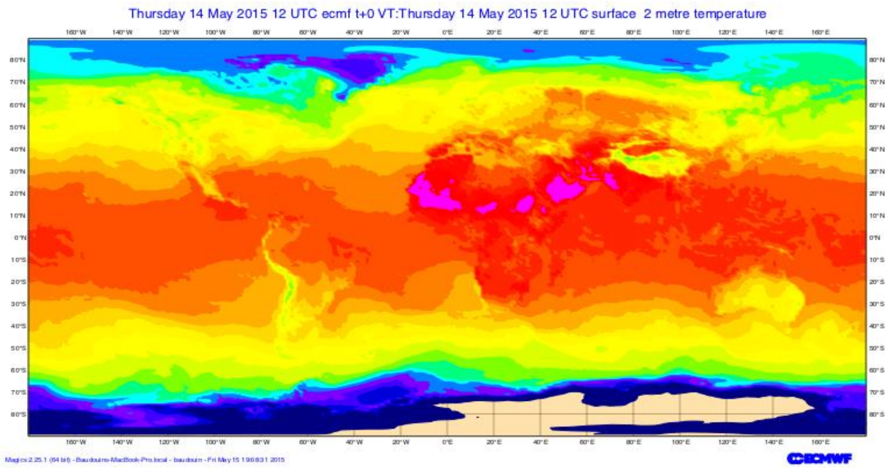
Merge same input

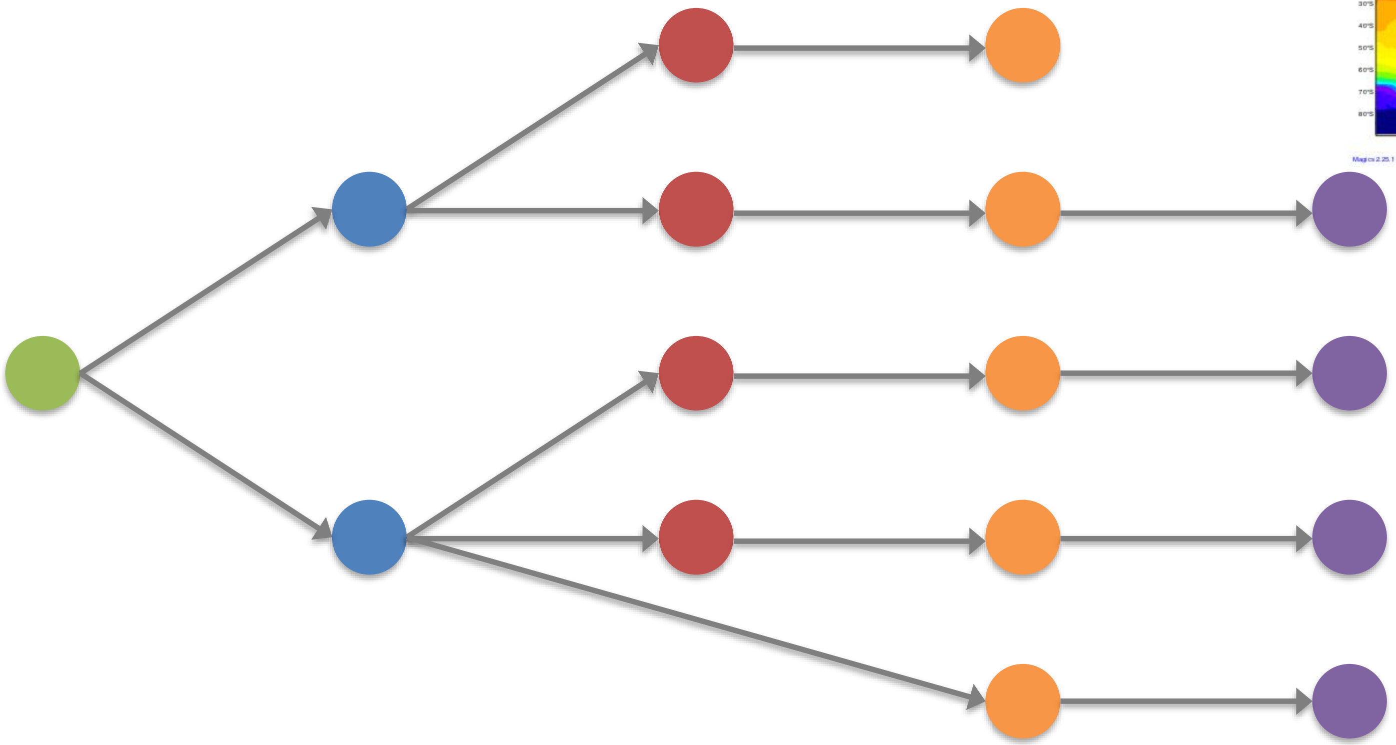




PGen – Task Graph Analysis

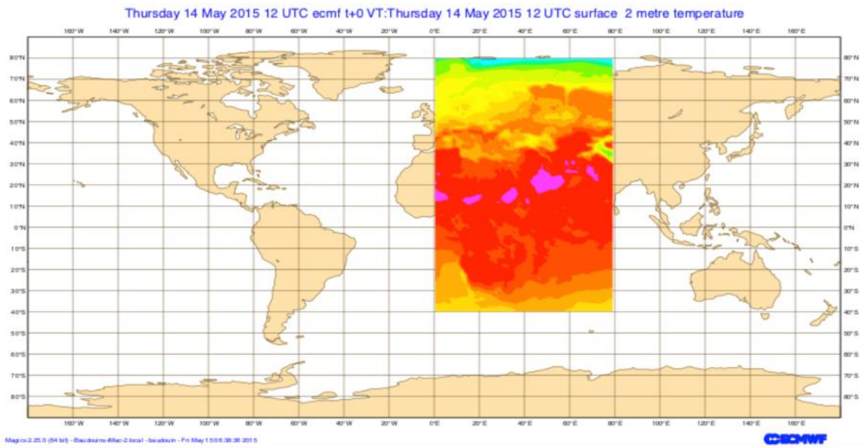
Merge same interpolation target grids

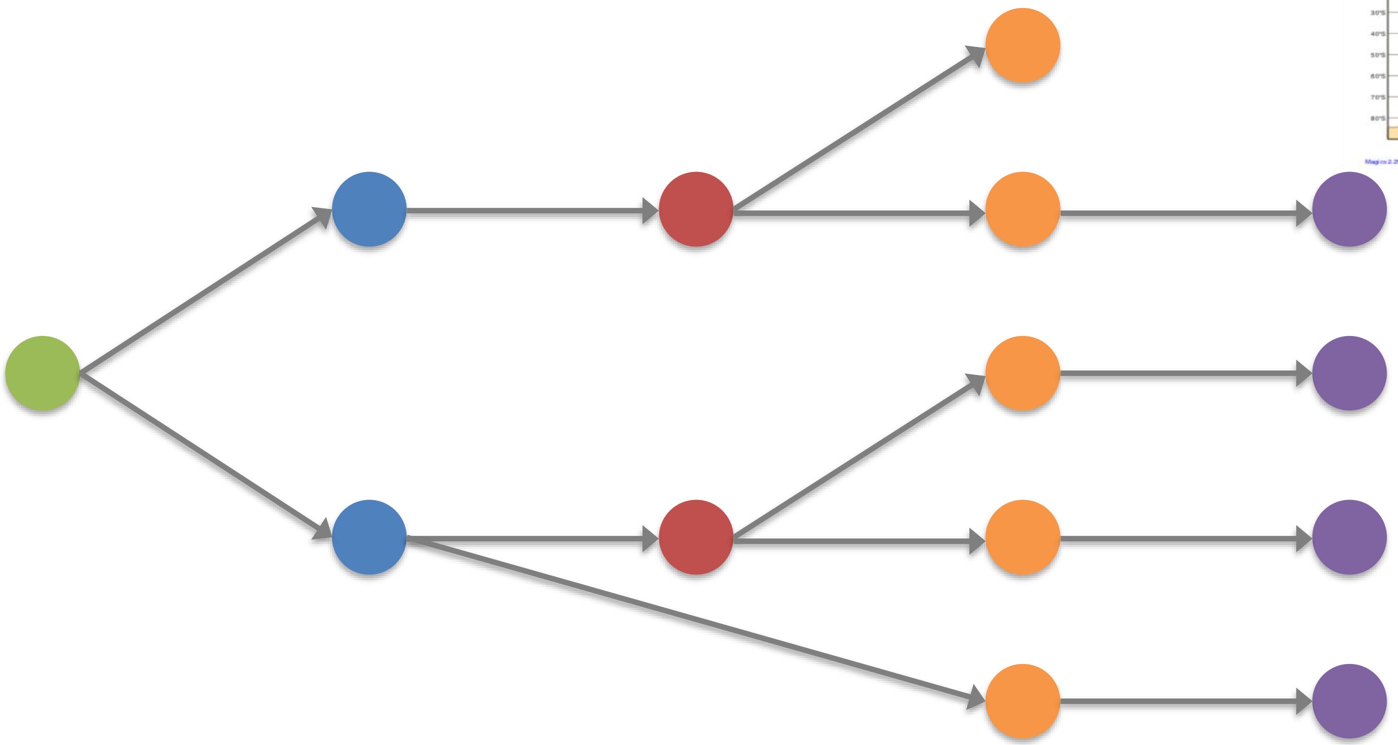




PGen – Task Graph Analysis

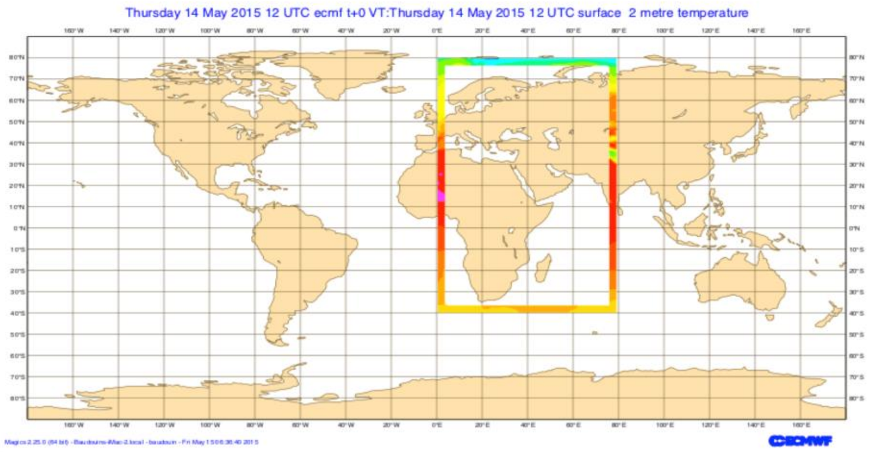
Merge same local area cropping

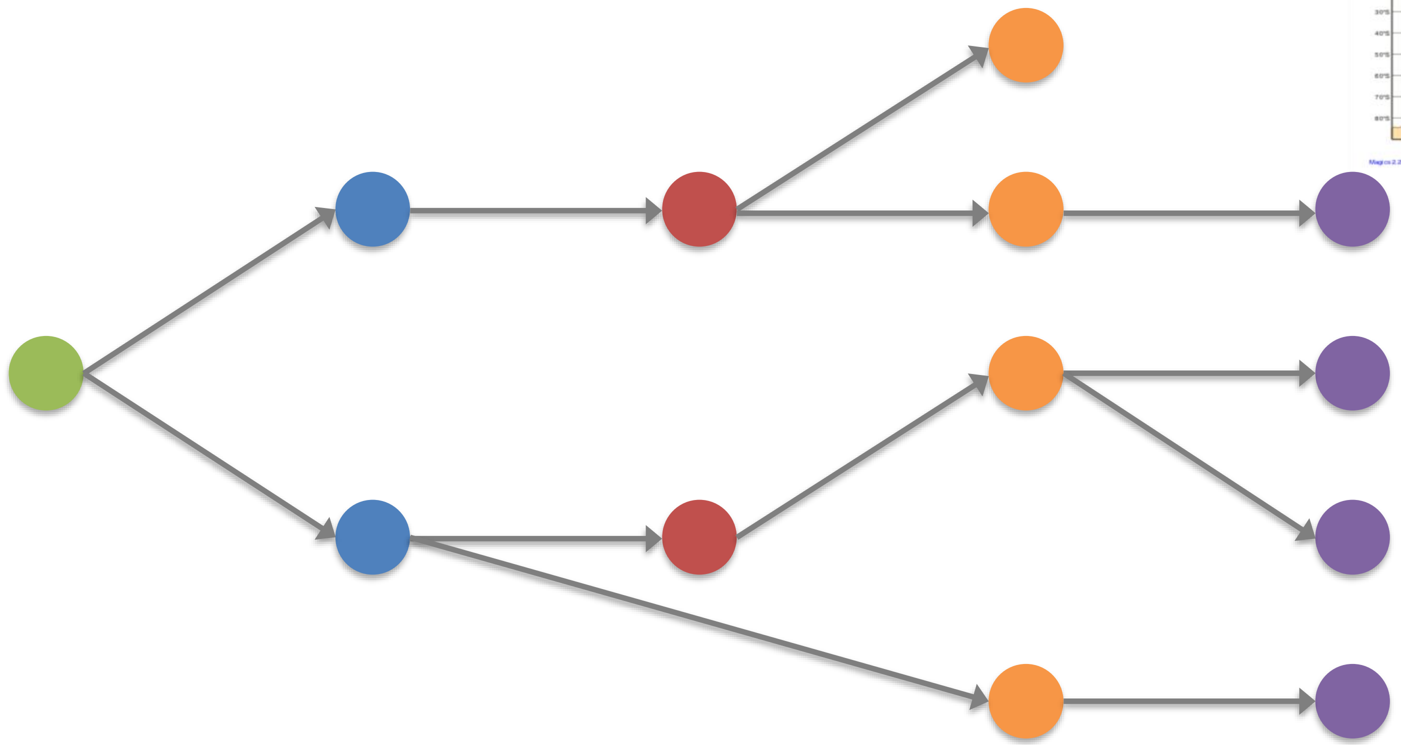




PGen – Task Graph Analysis

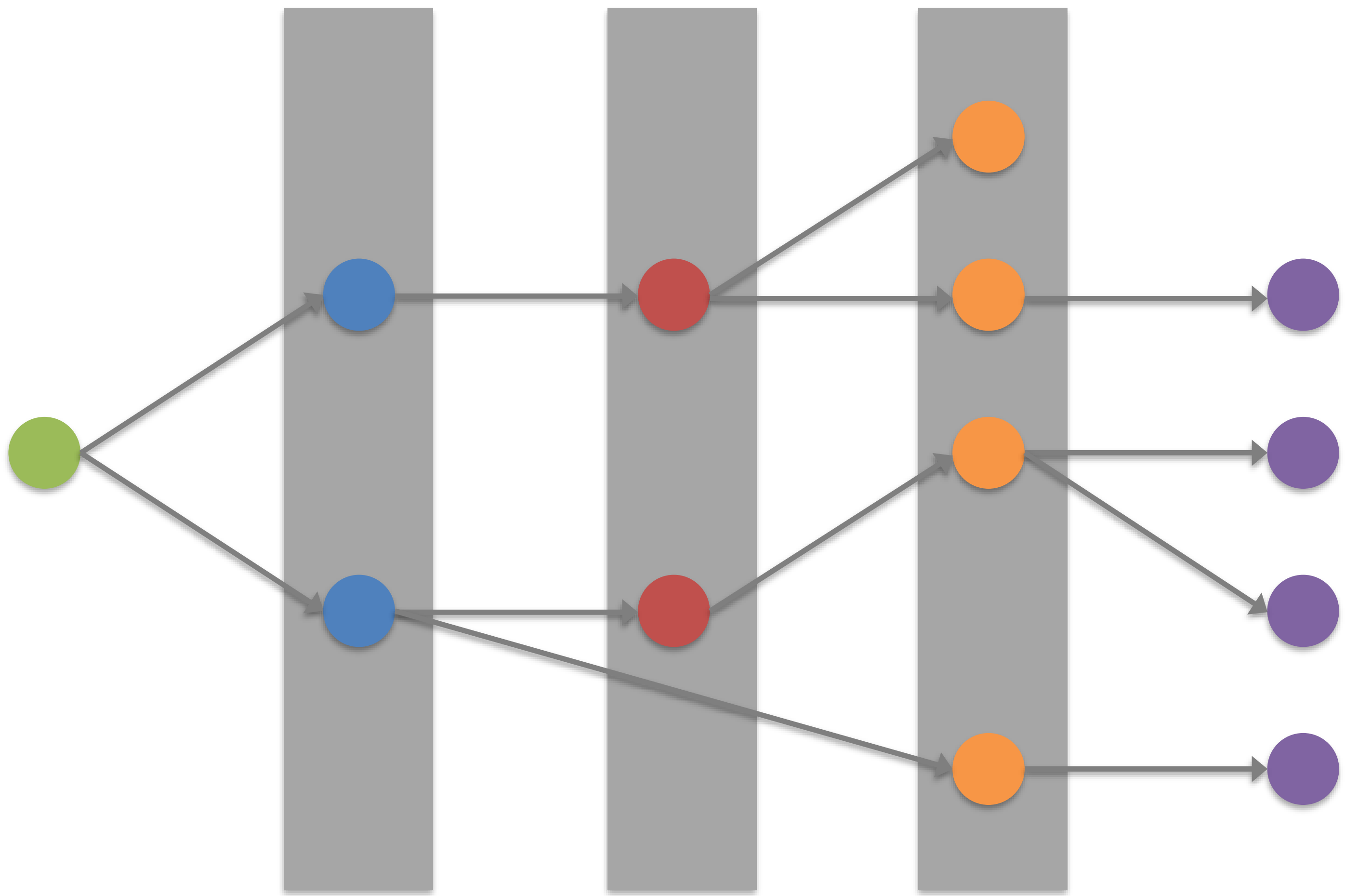
Merge same local area frames



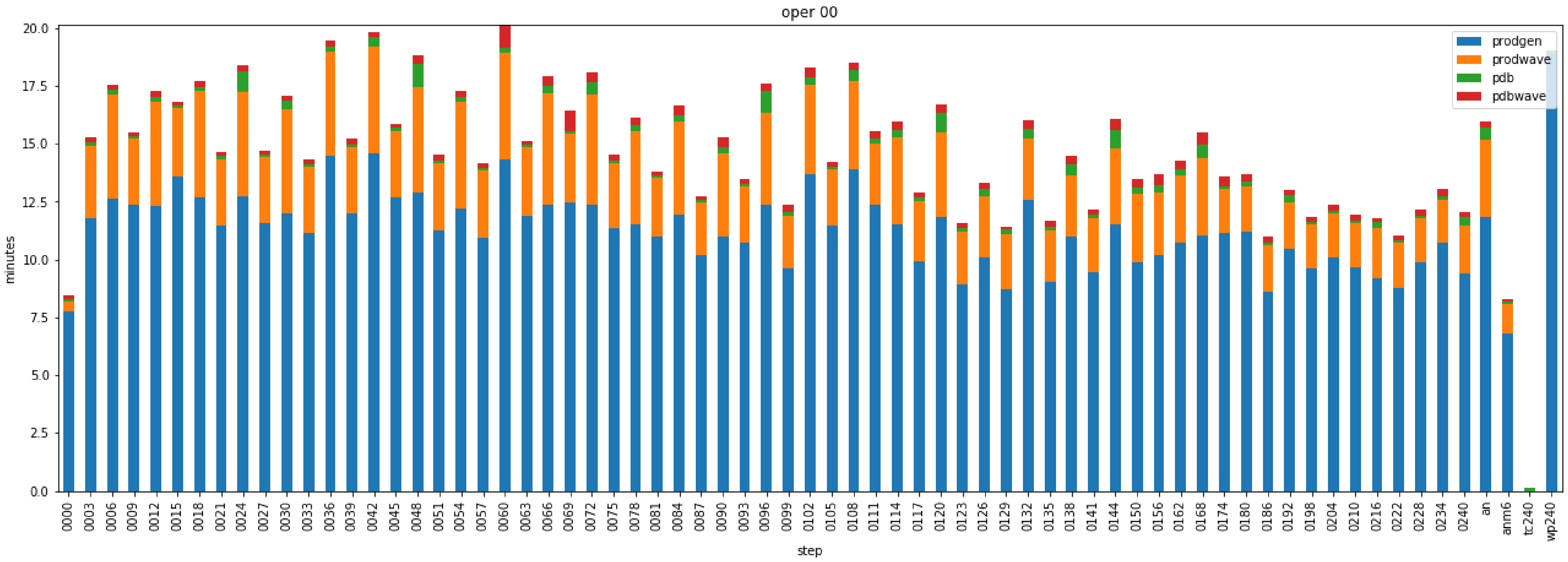


PGen – Task Graph Analysis

Caching of operators

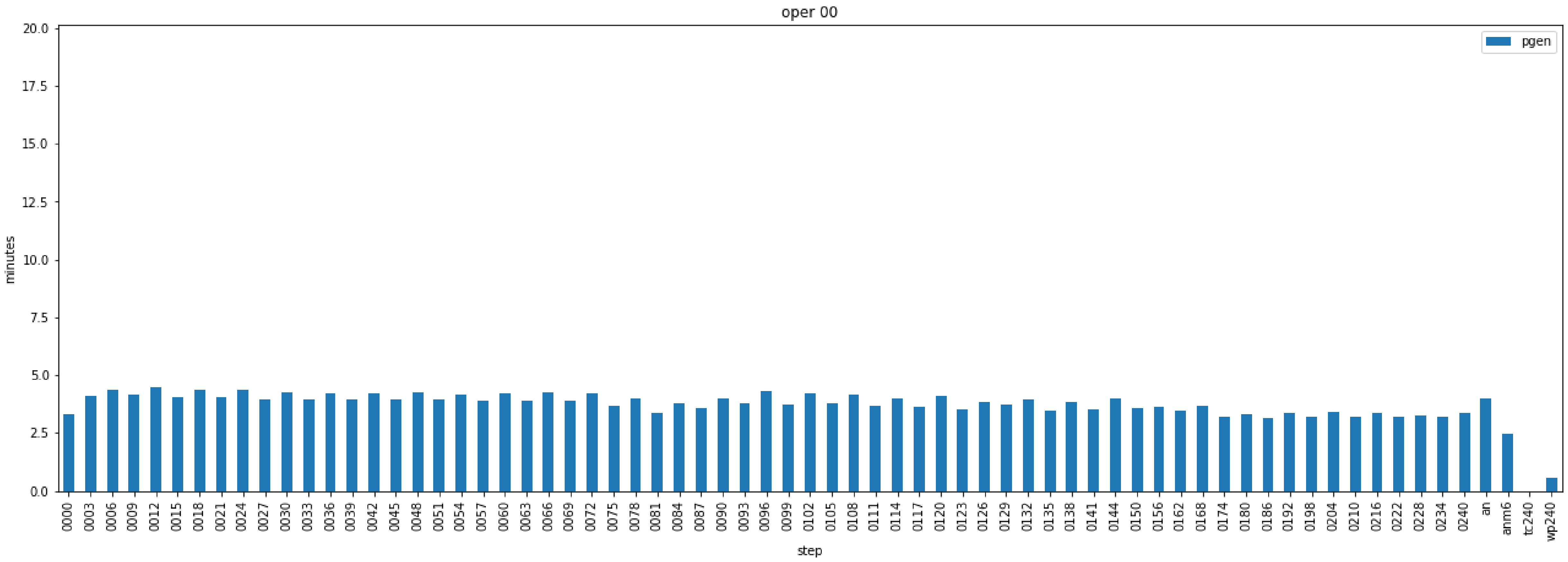


Performance Analysis – Oper Stream 00Z run



e.g. step 24 ~ 18 min

Performance Analysis – Oper Stream 00Z run



e.g. step 24 ~ 4.3 min = 412% faster

Compose your complex workflow of manageable, deterministic building blocks

Thanks for your attention

Questions?