Decoupling data collocation and storage performance using scientifically meaningful object identifiers

HPC I/O in the Data Center Workshop

Olivier Iffrig, James Hawkes, Simon Smart, Tiago Quintino

ECMWF

olivier.iffrig@ecmwf.int

ECMWF

ECMWF's Forecasting Systems

Established in 1975, Intergovernmental Organisation

- 23 Member States | 12 Cooperating States
- 350+ staff

24/7 operational service

- Operational NWP – 4x HRES+ENS forecasts / day
- Supporting NWS (coupled models) and businesses

Research institution

- Experiments to continuously improve our models
- Reforecasts and Climate Reanalysis

Operate 2 EU Copernicus Services

- Climate Change Service (C3S)
- Atmosphere Monitoring Service (CAMS)
- Support Copernicus Emergency Management Service (CEMS)

Reading, GB

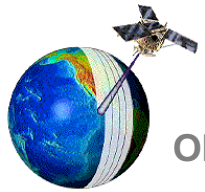
Bonn, DE

Bologna, IT

Starting from weather forecasting…

Handling weather data at ECMWF

The ECMWF operational workflow



Global Observations

200M obs/day

100TiB/day

45TiB/day
83M products

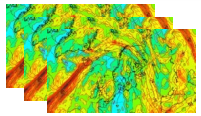
~350 destinations worldwide

Acquisition → IFS Model → Product Generation → Product Dissemination

Member States & Customers

Obs

Fields

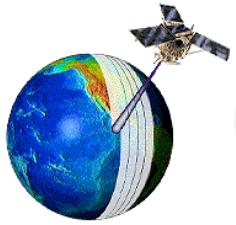


Products

MARS

Perpetual Archive

Storage View of Workflow



Observations

Acquisition

Product Dissemination

Member States & Customers

Acquire

Disseminate

Produce

Earth Sys Model

Storage Parallel FS

Analyse

ML + Analytics

Archive

Modify

Product Generation

Data is Central!

Perpetual Archive

Labelling and accessing data: the MARS language

- Operational data:

```
RETRIEVE,
    TYPE      = FC,
    CLASS     = OD,
    EXPVER    = 0001,
    STREAM    = OPER,
    DATE      = 20160517,
    TIME      = 1200,
    STEP      = 12/24/36,
    PARAM     = Z/T,
    LEVTYPE   = PL,
    LEVELIST  = 1000/500,
    TARGET    = out.grib
```

- Research experiment:

```
ARCHIVE,
    TYPE      = FC,
    CLASS     = RD,
    EXPVER    = ABCD,
    STREAM    = OPER,
    DATE      = 20160517,
    TIME      = 1200,
    STEP      = 12/24/36,
    PARAM     = Z/T,
    LEVTYPE   = PL,
    LEVELIST  = 1000/500,
    SOURCE    = in.grib
```

- Product generation:

```
DISSEMINATE,
    TYPE      = FC,
    CLASS     = OD,
    EXPVER    = 0001,
    STREAM    = OPER,
    DATE      = 20160517,
    TIME      = 1200,
    STEP      = 12/24/36,
    PARAM     = Z/T,
    LEVTYPE   = PL,
    LEVELIST  = 1000/500,
    GRID      = 0.5/0.5,
    AREA      = europe
    TARGET    = XXX:ZZ
```

- Observations:

```
RETRIEVE,
    TYPE      = OB,
    OBSGROUP  = CON,
    OBSTYPE   = SSD,
    DATE      = 20150201,
    TIME      = 00,
    RANGE     = 1439
```

Unique and **semantic** way to describe all ECMWF data

Curating data collections in a meaningful way

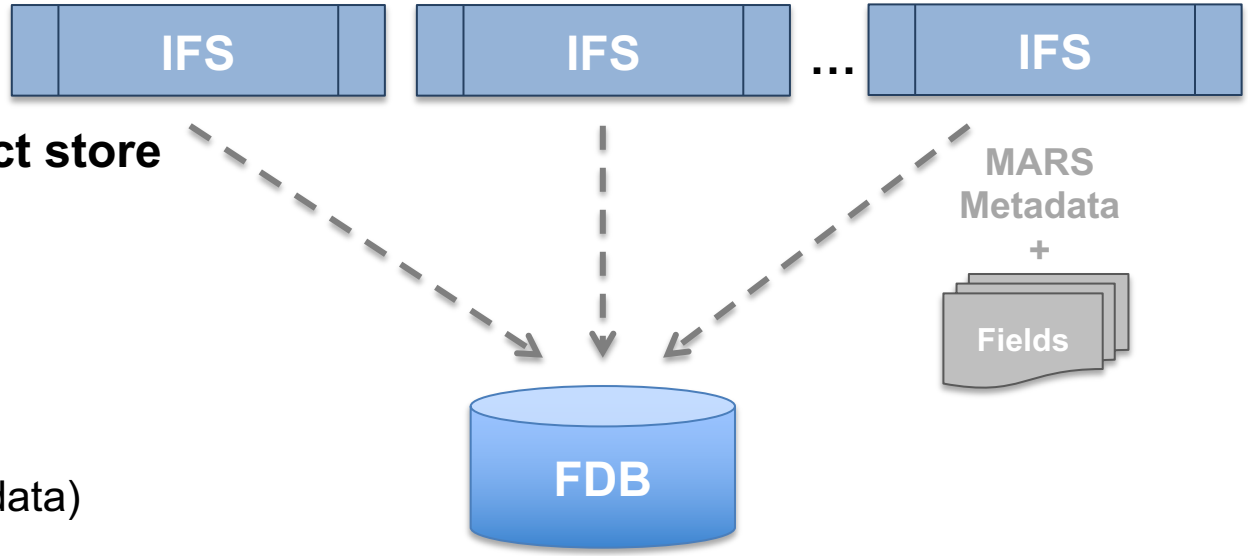
POSIX namespace

```
/scratch/user1/exp42/2020-03-18/output_004
/scratch/user1/exp42/20220415/out39
/scratch/user1/exp42/20220415/out48.txt
/scratch/user2/DATA/2021/FC/1208/STEP13
/shared/project4/exp_abcd/20220502/step_002
/prod/0001/20220329/step053/rank0012
/prod/0001/20220329/step053/rank0013
/prod/0001/20220329/step053/rank0014
/prod/0001/20220329/step053/rank0015
```

Semantic namespace

```
source=user1,experiment=42,date=20200318,step=4
source=user1,experiment=42,date=20220415,step=39
source=user1,experiment=42,date=20220415,step=48
source=user2,experiment=fc,date=20211208,step=13
source=project4,experiment=abcd,date=20220502,step=2
source=prod,experiment=1,date=20220329,step=53,rank=12/to/15
```

FDB (version 5)



- **Domain specific (NWP) Distributed object store**
- Transactional, No synchronisation
- Semantic access to data
- Key-value store
  - Keys are scientific meta-data (MARS Metadata)
  - Values are byte streams (GRIB)
- Support for multiple backends:
  - POSIX file-system (currently on Lustre)
  - Intel DAOS (under development)
  - CEPH (Cloud suited object store)
- Decouples data administration from the usage semantics
  - ***In the semantic/scientific language of the user:***

```
param=temperature/humidity,
levels=all,
steps=0/240/by/3
date=01011999/to/31122015,
```

FDB 5 Semantics

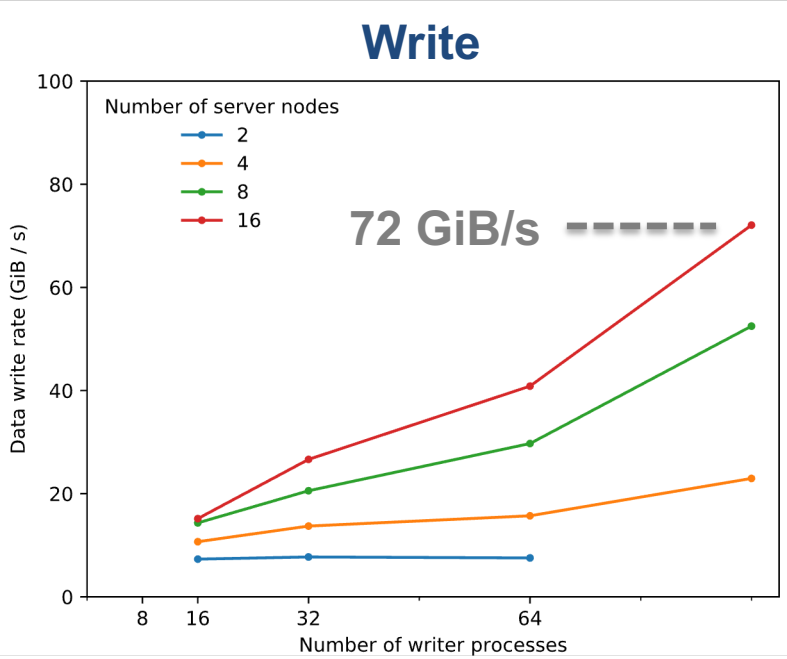
1. ACID – ***Transactional***
2. Write blocks until data handed over – ***Asynchronous***
3. `flush()` blocks until data is visible – ***Consistent***
4. Write-once, don't overwrite - ***Immutable***
5. Data can be masked – ***Versioned***

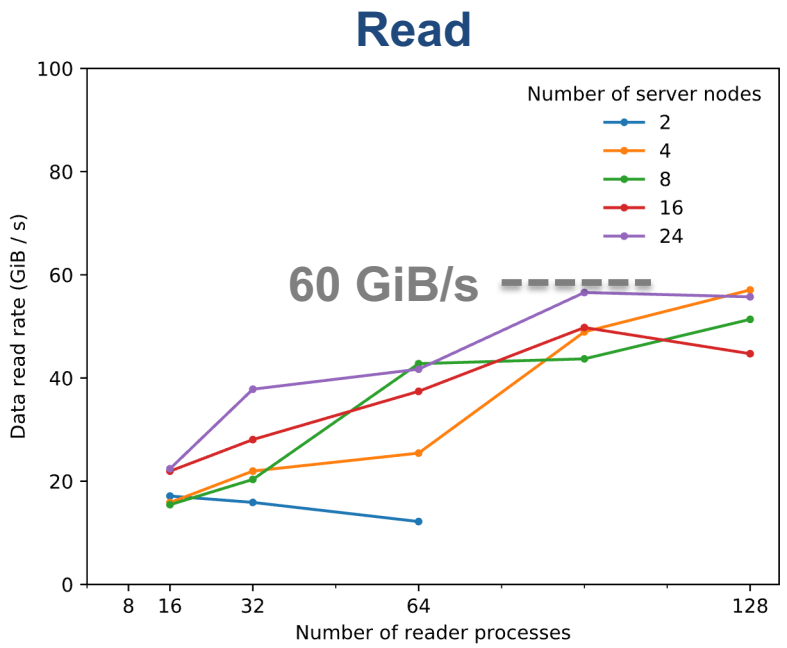
- All I/O operations are asynchronous, so computation can continue
- Distributed to all servers using a ***Rendezvous Hash***, *so no synchronisation needed*

To summarise

- Semantic data access system at ECMWF
  - Based on meteorological keywords
  - Unique description of all datasets
  - Language defined by a schema
  - Decouple data specification from the actions of the system

- FDB: semantic object store software
  - github.com/ecmwf/fdb
  - Robust and performant
  - Implementation tied to meteorology

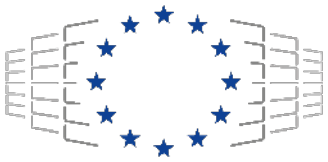




IO-SEA

… Towards a domain-agnostic storage system

The Data Access and Storage Interface



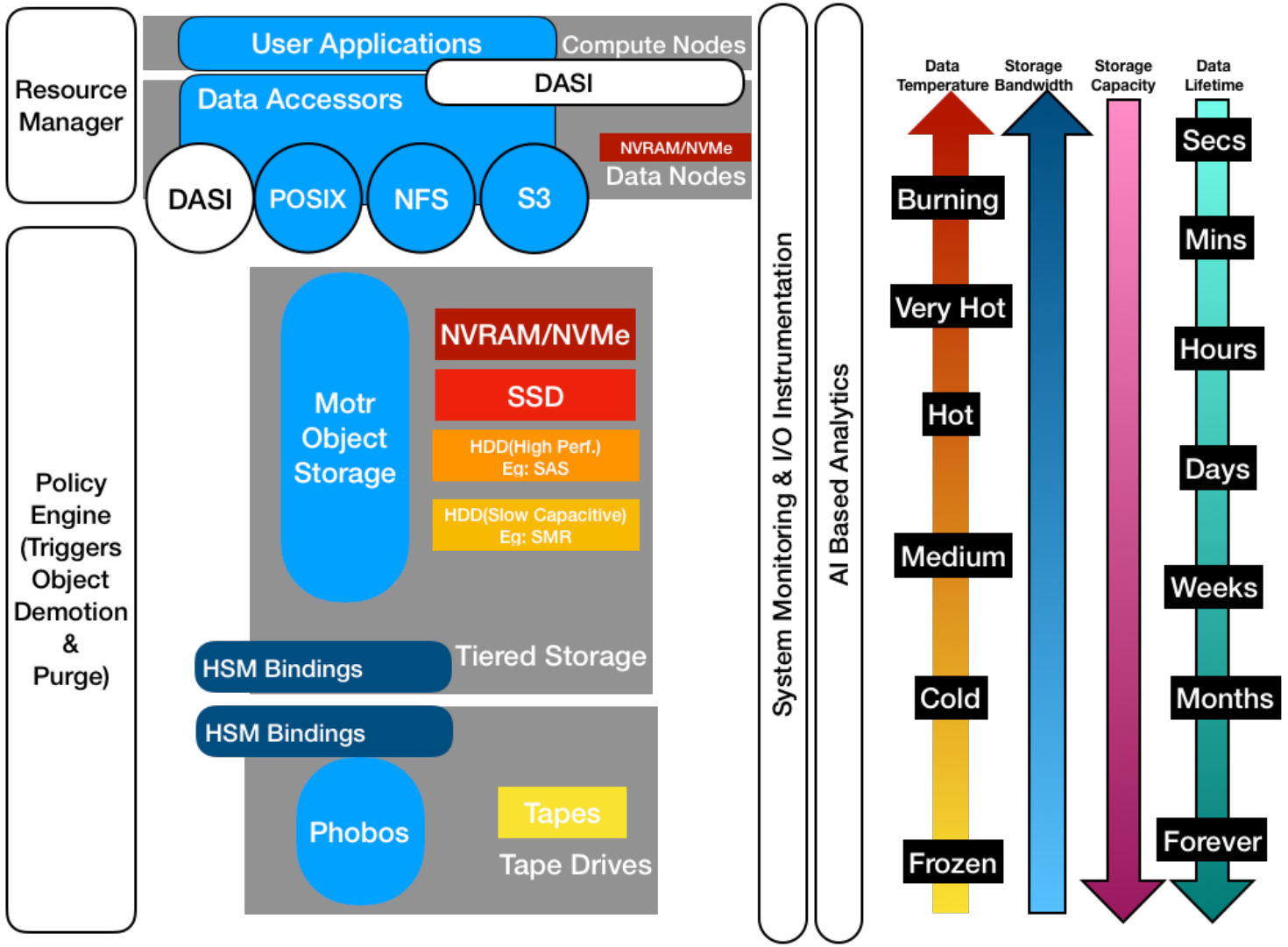
EuroHPC
Joint Undertaking

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955811. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, the Czech Republic, Germany, Ireland, Sweden, and the United Kingdom.

Enter DASI: the Data Access and Storage Interface

- Goal: **Lower the entry barrier to effective semantic data management**
- Pillars:
  - Declaration of domain-specific languages via a schema
  - Efficient data indexing
  - Abstraction of the underlying storage system

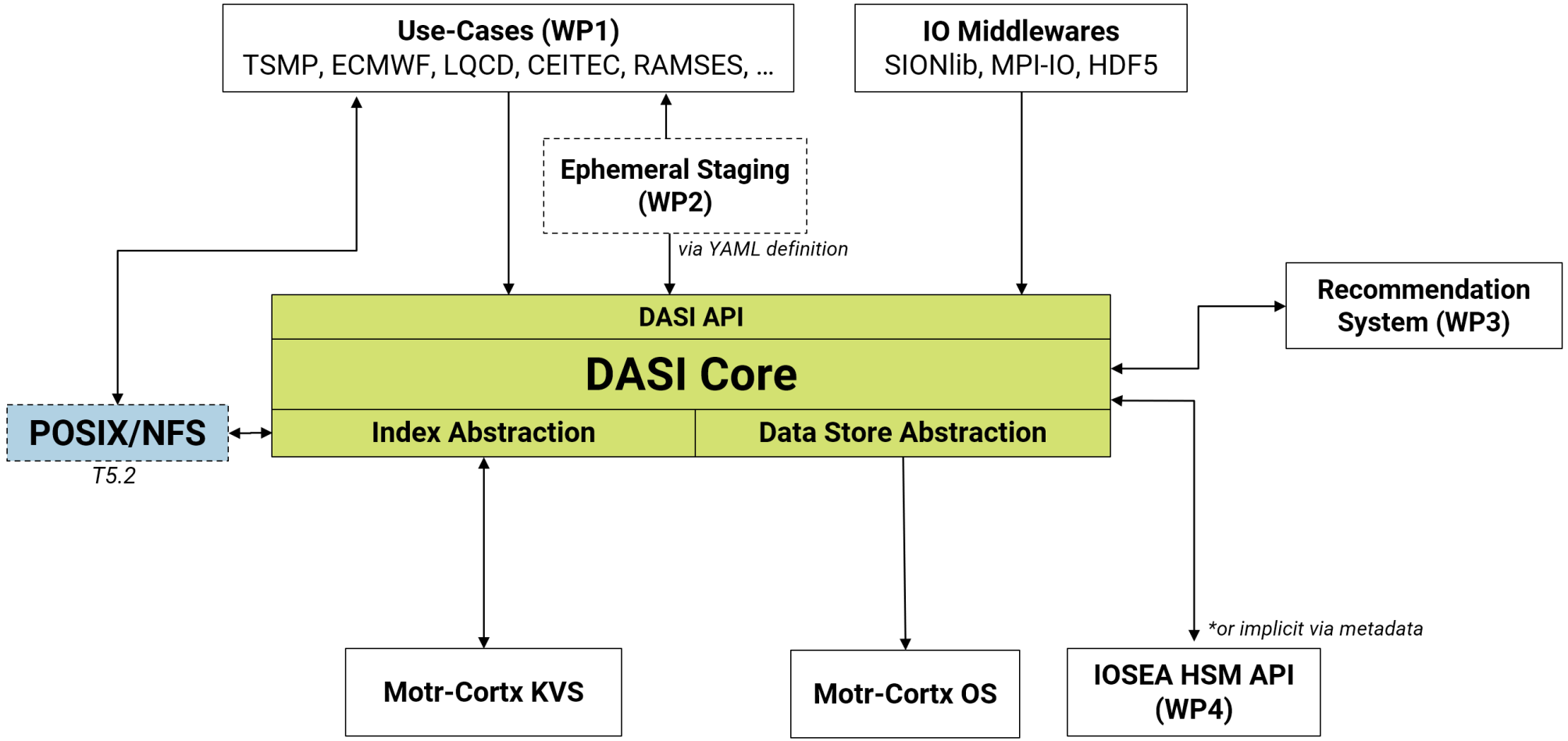
The IO-SEA project



DASI: decoupling data collocation and storage performance using scientifically meaningful object identifiers

IO-SEA

DASI in the context of IO-SEA



Interacting with the DASI

- Identifying data
  - Sets of keyword/value pairs
  - Defined by a schema
  - Key: identifies a single object **uniquely**
  - Query: multiple values allowed: lists, wildcard
- Actions
  - Put(Key, Bytes)
  - Get(Query) → [(Key, Bytes)]
    - Evaluates the cartesian product of all combinations
  - List(Query) → [Key]
    - Supports incomplete queries
  - Delete(Query)

Key

```
model: covid-spread
date: 20210112
experiment: 42
epoch: 123
variable: R0
```

Query

```
model: any
date: any
experiment: 42
epoch: [123,124,125]
variable: any
```

The process in a nutshell

DASI API	Builds keys and queries Interprets the user requests (put, get, list...)
DASI Core	Validates keys and queries according to the schema Converts keys and queries into a set of unique normalised items
Index abstraction	Resolves the object locations from the keys
Store abstraction	Interacts with the storage system to access the data

The DASI schema

- DASI provides an API for scientifically meaningful access to data
- The schema follows a **domain-specific hierarchical taxonomy**
- The schema is split into three levels that can be interpreted by the backend

```
↳ model > experiment
  ↳ date > epoch
    ↳ variable
    ↳ location > variable
  ↳ date > location
    ↳ variable
↳ institute > location
  ↳ date > method
    ↳ variable
```

```
model: covid-spread
date: 20210112
experiment: 42
variable: R0
epoch: 123
```

Building your own schema

- Identify your data collection
  - Which data do you want to store together?
  - What are your base objects?
- Define your domain-specific keywords
  - How do you **uniquely** identify an object?
  - If needed, what are the different sets of keywords that you need?
- Order your taxonomy
  - Build a tree that lists all the possible sets of keywords
  - Choose the relevant order for the keywords
- Decide on where to put the boundaries for the three storage levels

```
model: covid-spread
date: 20210112
experiment: 42
variable: R0
epoch: 123
```

```
↳ model > experiment
  ↳ date > epoch
    ↳ variable
    ↳ location > variable
  ↳ date > location
    ↳ variable
↳ institute > location
  ↳ date > method
    ↳ variable
```

The DASI backends

- Index backend
  - Resolves object locations
    - Resolve(SplitKey) → Location
  - Lists available data
    - List(SplitQuery) → [SplitKey]
  - Creates / Deletes entries
    - Create(SplitKey, Location)
    - Delete(SplitKey)
- Storage backend
  - Implements atomic operations
    - Put(Location, Bytes)
    - Get(Location) → Bytes
    - Delete(Location)
  - Implements three storage levels
    - Allow for performance optimisation
    - Small objects → aggregate at third level
    - Many objects → split between first and second levels

Wrapping up

- Semantic data access provides
  - Description of data in scientific terms
  - Consistency and reproducibility
  - Decoupling between performance and data collocation

- Software
  - FDB: github.com/ecmwf/fdb
  - DASI: github.com/ecmwf-projects/dasi

This project has received funding from the European High-Performance Computing Joint Undertaking (JU) under grant agreement No 955811. The JU receives support from the European Union's Horizon 2020 research and innovation programme and France, the Czech Republic, Germany, Ireland, Sweden, and the United Kingdom.