ISPDC 2015

14th International Symposium on Parallel and Distributed Computing

29/Jun - 01/Jul 2015 | Limassol, Cyprus

dataClay

The integration of persistent data, parallel programming, and true sharing

Toni Cortes

Storage system research group
Barcelona Supercomputing Center

BSC Barcelona Supercomputing Center
Centro Nacional de Supercomputación

EXCELENCIA
SEVERO
OCHOA

Agenda

- The pillars
- The dark side
- The secret potential
- Time to wake up!

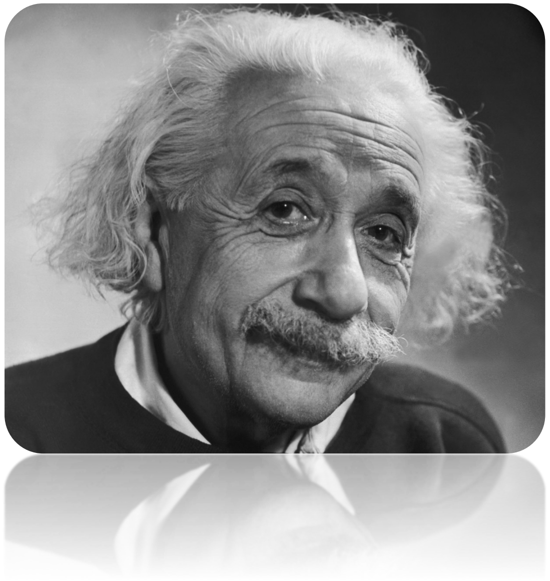
Agenda

- The motivation
  The pillars
- The dark side
- The secret potential
- Time to wake up!

From a different perspective...

- **“We cannot solve our problems with the same thinking we used when we created them”**

  Albert Einstein



- **Some of today’s thinking**
  - Data stored in
    - Files
    - Databases
  - Data is a 2nd-class citizen
    - Accessed with its own primitives
    - Data and code are different

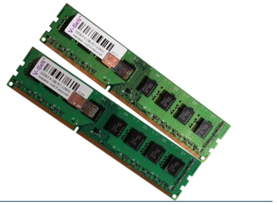
Before everything started
The pillars of dataClay

- **What ignited our research, our "big bang"**
  - Different data models: persistent vs. non persistent
  - New storage devices: byte addressable
  - Coupling data and code
  - Sharing is what really matters

- **And then dataClay came to life …**

  (more details on how all fits together
  in the next minutes)

Two data models!
Why waste time doing it twice?

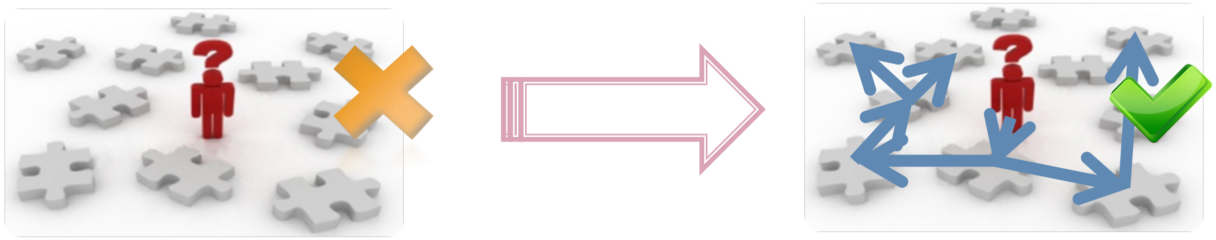


- **Today**
  - We have one data model for volatile data
  - Traditional data structures and/or objects
  - We have a different data model for the persistent data
  - Relational database, NoSQL database, files

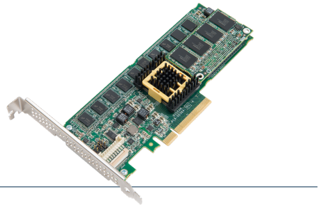
- **Future**
  - Store data in the same way as when volatile
  - Store objects and their relations

Data selection
No more database queries

- **In memory**
  - Data is “never” queried like in a DB
  - Data linked according to needs of program
  - Next data item found by following a link, not a query
- **Persistent data should behave in a similar way**
  - Following links is faster than a queries over whole dataset
  - Programs should not make any differences whether data is
    - In memory or
    - In persistent storage



New storage devices
Better to be prepared on time



- **New storage hardware is coming**
  - Storage class memory
  - Non-volatile RAM
- **Main characteristics**
  - Performance between memory and SSDs
  - Byte addressable
- **File systems or table based DB are not the right abstraction**
  - Both were designed to use block devices
  - Can be used, but would be a pity
    - What a potential loss!!

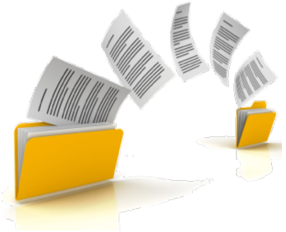
Coupling data and computation

They can live isolated, but ...

- **Computation and data are two different abstractions**
  - They are separated
- **This brings the problem of**
  - Should I move the data to compute it?
    - Does not work for big data sets
  - Should I move computation to the data?
    - Deployment difficult
- **If data and code were the same thing ...**
  - Using data would be much easier
  - (and safer ➔ see more in a few minutes)

Data sharing today
And why it is not enough

- **Download files**
  - Flexible
  - Only for static data
  - Avoid unneeded copies and transfers
  - Data provider loses control over the downloaded data



- **"Data services" an API to access the data**
  - Data provider keeps control
  - Both dynamic and static data
  - No unneeded copies or transfers
  - API restricted to what the provider can do

EXCELENCIA
SEVERO
OCHOA

Agenda

- The motivation
  The pillars
- The technology
  The dark side
- **The secret potential**
- **Time to wake up!**

Barcelona
Supercomputing
Center
Centro Nacional de Supercomputación

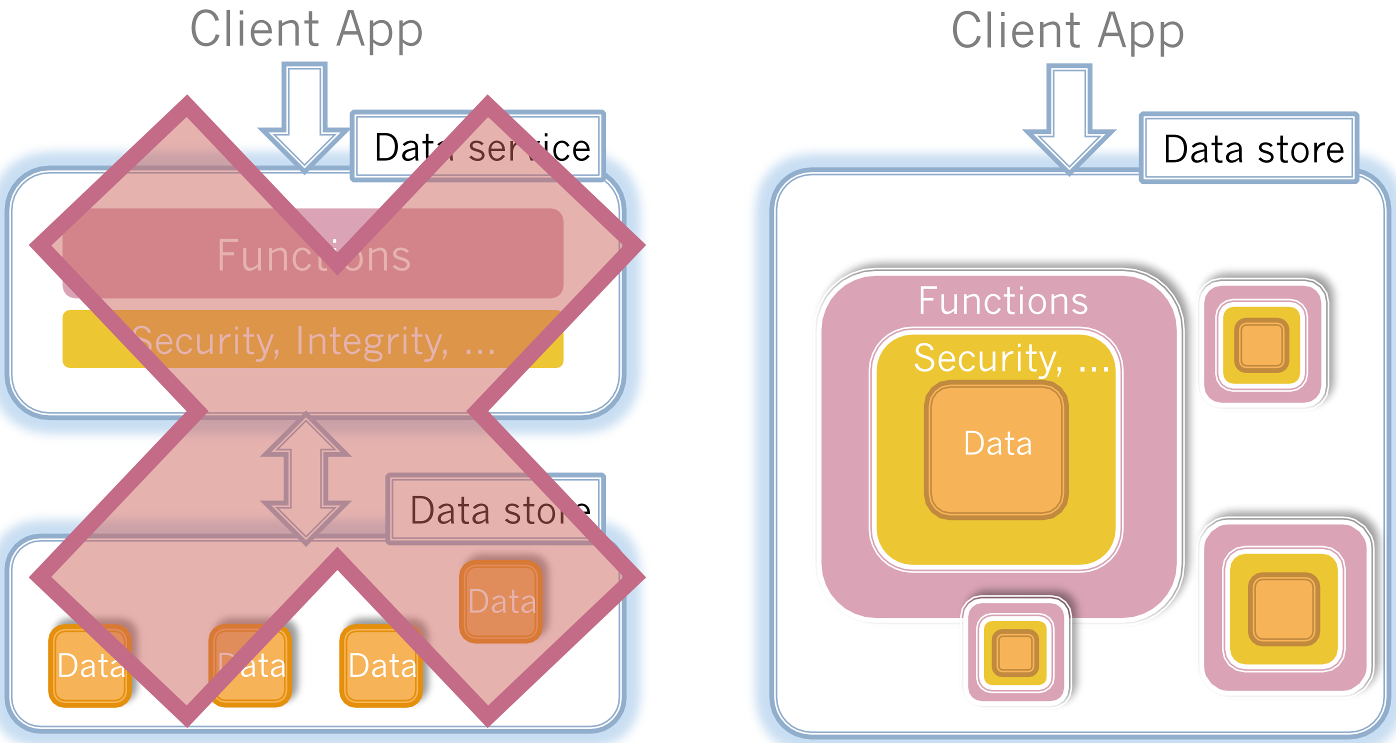
Our vision
What dataClay does

- **dataClay is a platform that enables**
  - Apps to make **objects** and their **relationships persistent**
  - 3rd parties to add mode data or "**change**" the **data model**
  - 3rd parties to **upload computations** to be shared
  - Each user to see different "views" of the data
  - Data owner to maintain control over its data
  - Efficient access to data

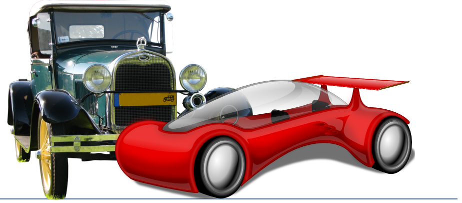
- **Key technologies**
  - Self-contained objects
  - Data enrichment by 3rd parties

Key technology
Self-contained objects

- **Push the idea of data services to the limit**
  - Based on the OO paradigm



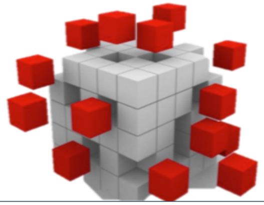
Self-contained objects
But, what is really new?



- **Self-contained and data services**
  - Same concept different implementation?

- **Then…**
  - … we need something else …
  - … something to make it really flexible!

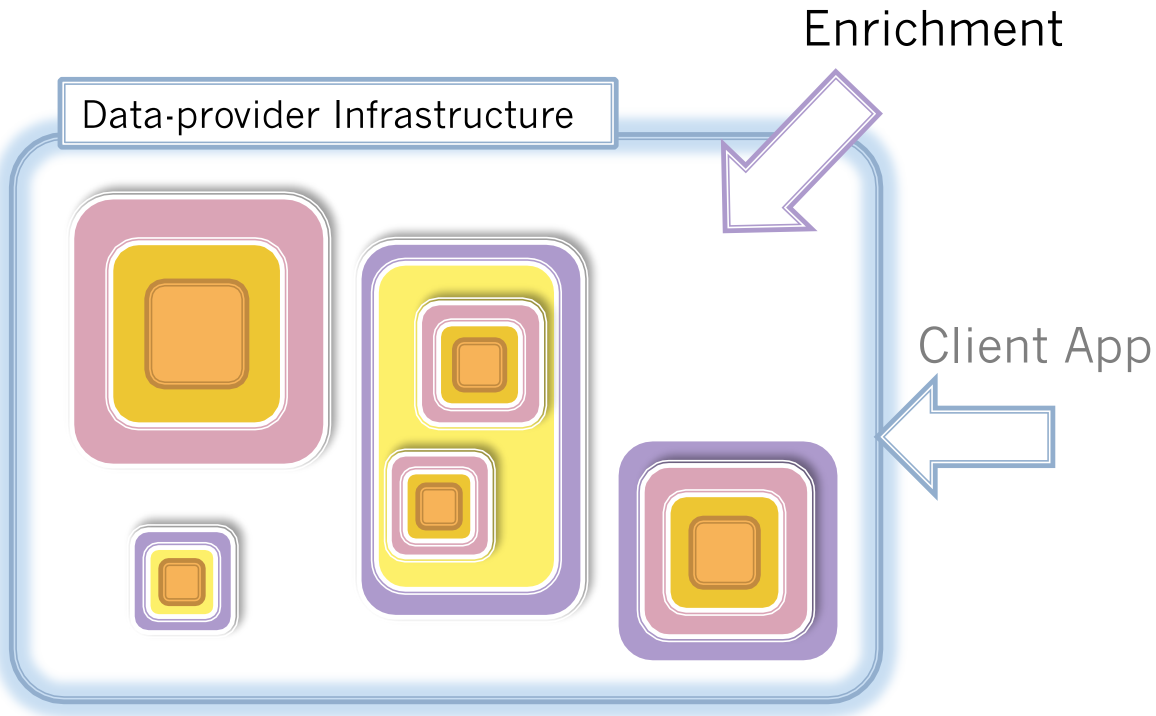
3rd-party enrichment
What is it exactly?



- **By enrichment we understand:**
  - Adding new information (fields or data) to existing datasets
  - Adding new code to existing datasets
    - New methods
    - New implementations
- **This enrichment should**
  - Be possible during the life of data
  - Not be limited to the data owner
  - Enable different views of the data to different users/clients
    - Not everybody should see the same enrichments
    - Several enrichments should be available concurrently
  - Enable the avoidance of queries

3rd-party enrichment
And now animated

- Data can be enriched both with **data and code**
- **Code will be executed in the provider infrastructure**



Using a single infrastructure?
Killing the bottleneck

- **Using a “single” infrastructure may become a bottleneck**
- **Security and privacy policies should be part of the data**
  - Thus, data could be offloaded to other infrastructures
    - Without breaking the data policies
  - Data owner enables 3rd party enrichment and …

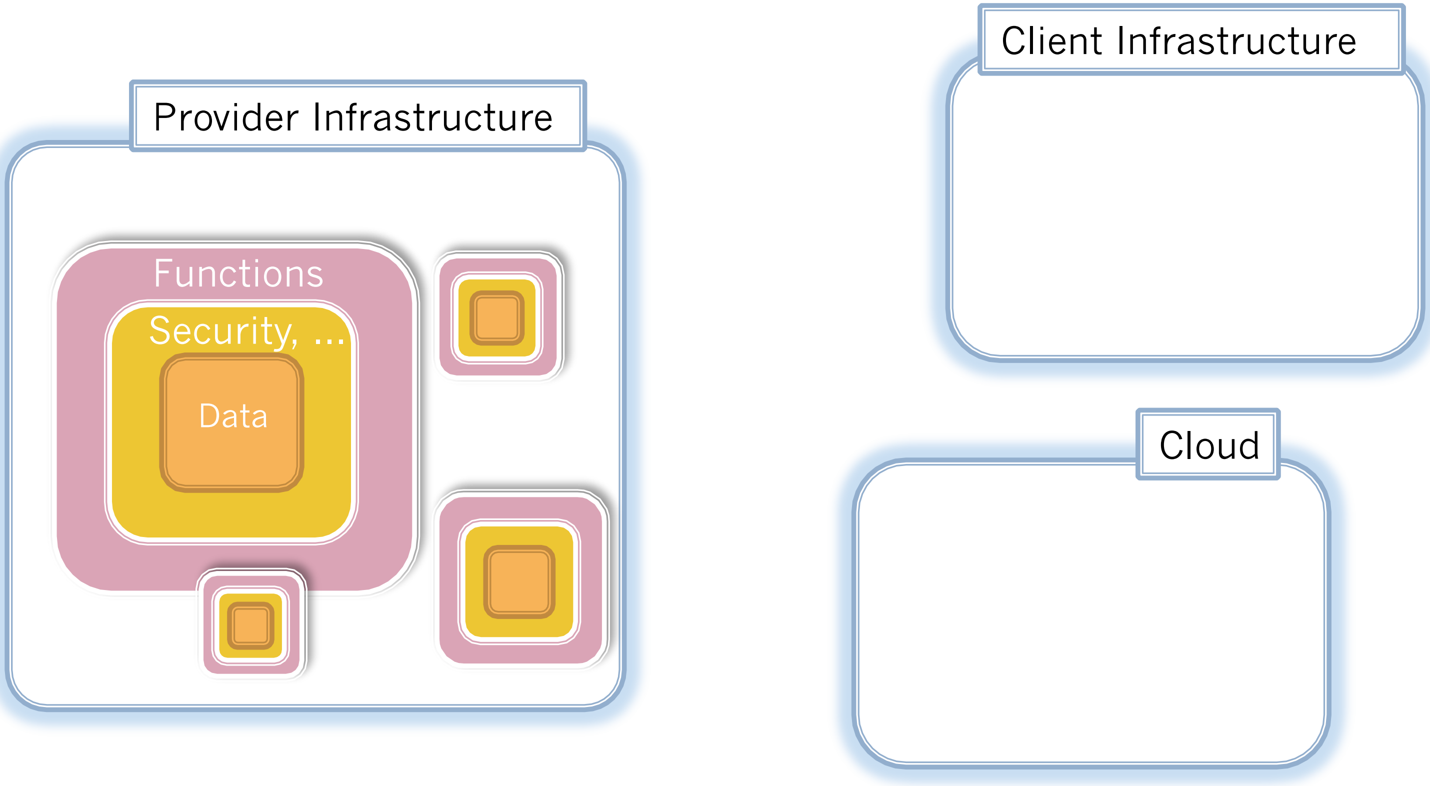
    … does not lose control

- **How it is implemented?**
  - Policies are defined using a declarative language
  - Policies enforced as part of object methods

Distributing objects

Distributing objects

- **Efficient usage of resources**
  - Data and code can be offloaded
    - to resources not accessible by the data provider

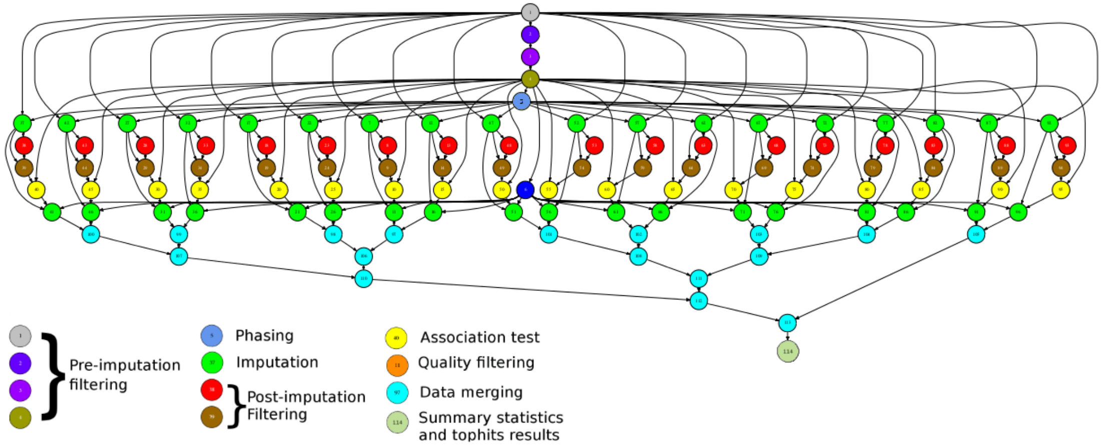


Agenda

- The motivation
  The pillars
- The technology
  The dark side
- The integration into the parallel programming language
  The secret potential
- **Time to wake up!**

Task-based programming

- Task is the unit of work
- Data dependences between tasks
  - Imply partial order
  - Exhibit potential parallelism
  - Imply local synchronization
    - Not global!
- Implicit workflow



COMPSs

- **Sequential programming**
  - General purpose programming language + annotations
    - Currently Java and Python
- **Task based**
  - Builds a task graph at runtime
    - Express potential concurrency
    - Includes dependencies
  - Simple linear address space
- **Unaware of computing platform**
  - Enabled by the runtime for clusters, clouds and grids

Python (PyCOMPSs) syntax
How to write PyCOMPS code

- **Invoke tasks**
  - As functions/methods

Main Program

```
foo = Foo()
myFunction(foo)
foo.myMethod()
…
foo =
   compss_wait_on(foo)
foo.bar()
```

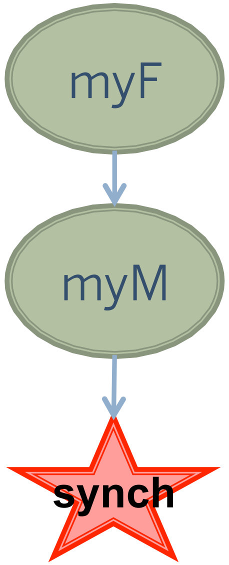
- **API** for data synchronization

- **Task definition** in function declaration
  - decorators

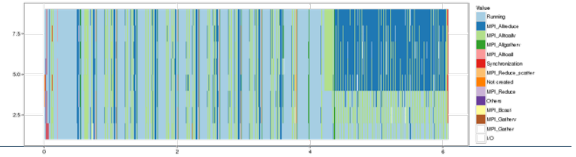
Function definition

```
@task( par = INOUT)
def myFunction(par):
   …
```

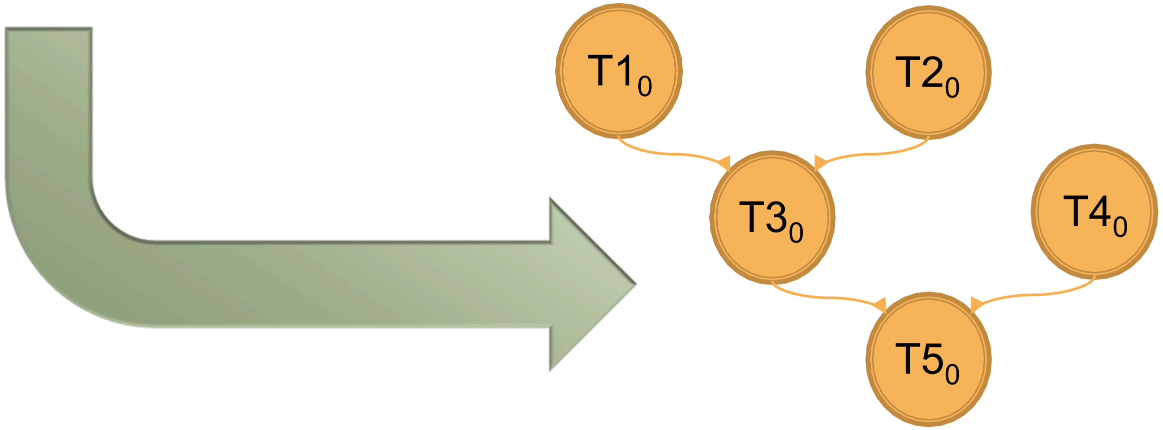
```
class Foo(object):
   @task()
   def
myMethod(self):
      …
```



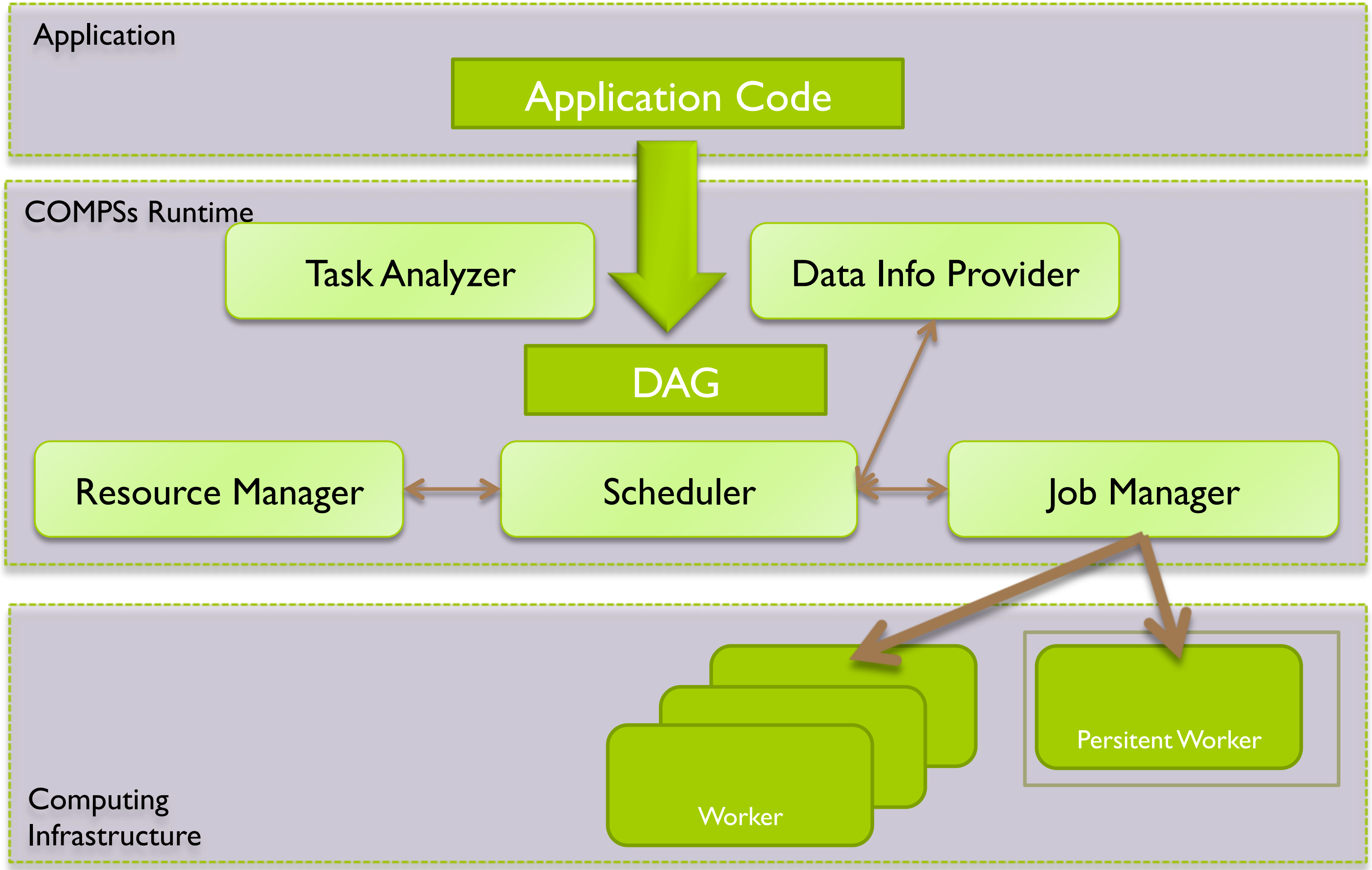
Parallel execution



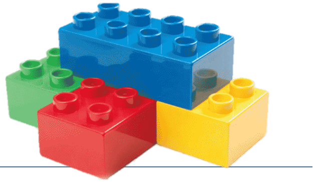
```
...

T1  (data1, out data2);
T2  (data4, out data5);
T3  (data2, data5, out data6);
T4  (data7, out data8);
T5  (data6, data8, out data9);

...
```

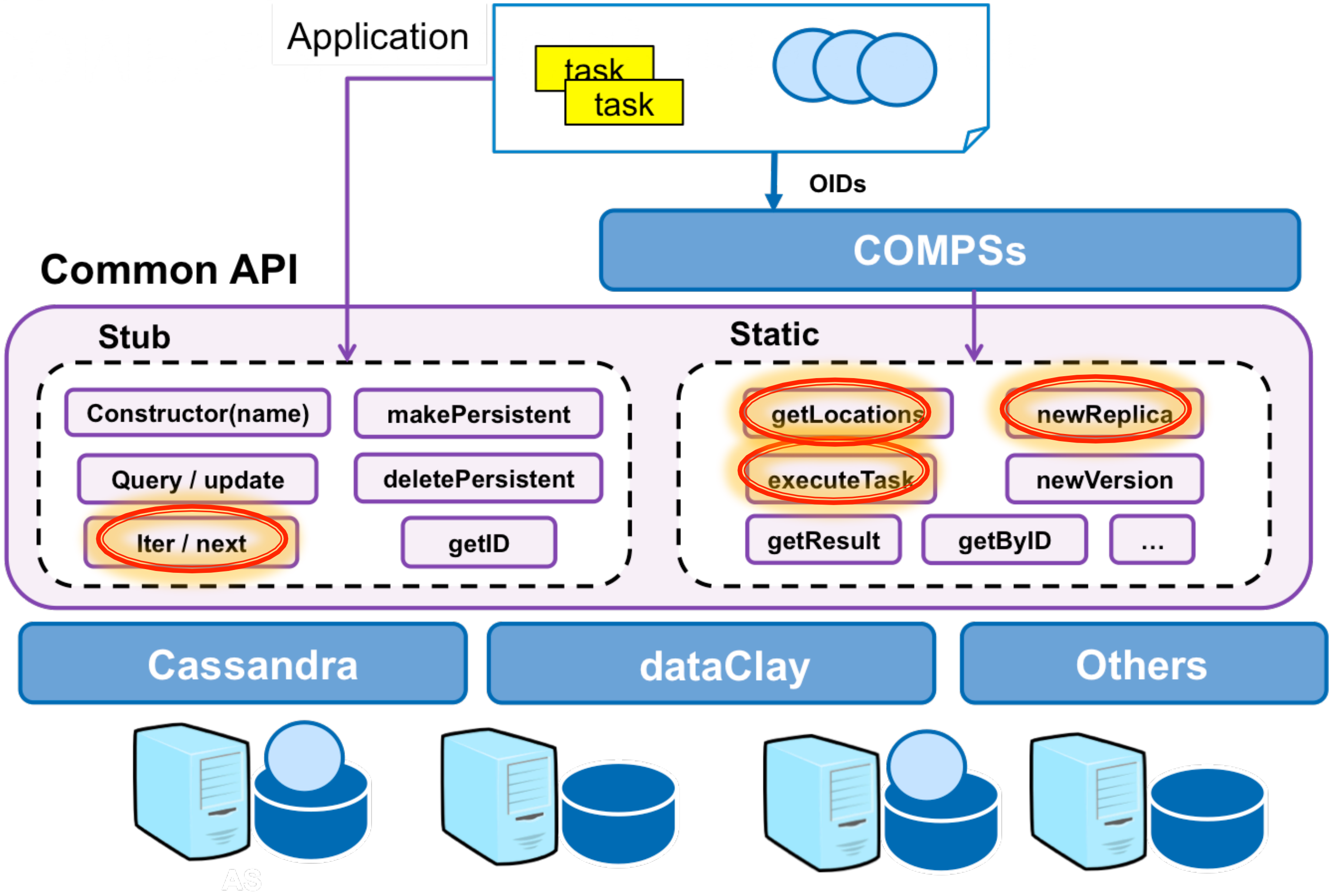


COMPSs framework



COMPSs & dataClay integration
Global view

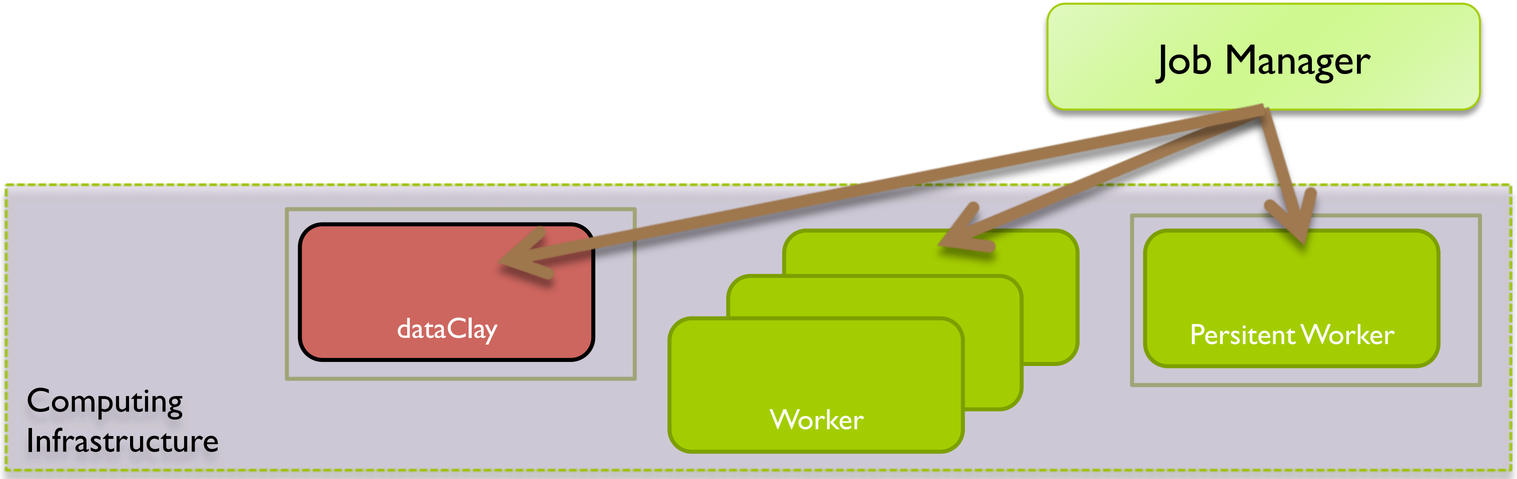




ExecuteTask
dataClay as a COMPSs worker

- **Executes a method (possibly static) in a given backend**
  - Replaces COMPSs worker threads

- **As opposed to direct method execution**
  - You can decide the execution backend `executeTask`
  - Asynchronous
    - Result can be checked by using `getResult`



A trivial example to follow

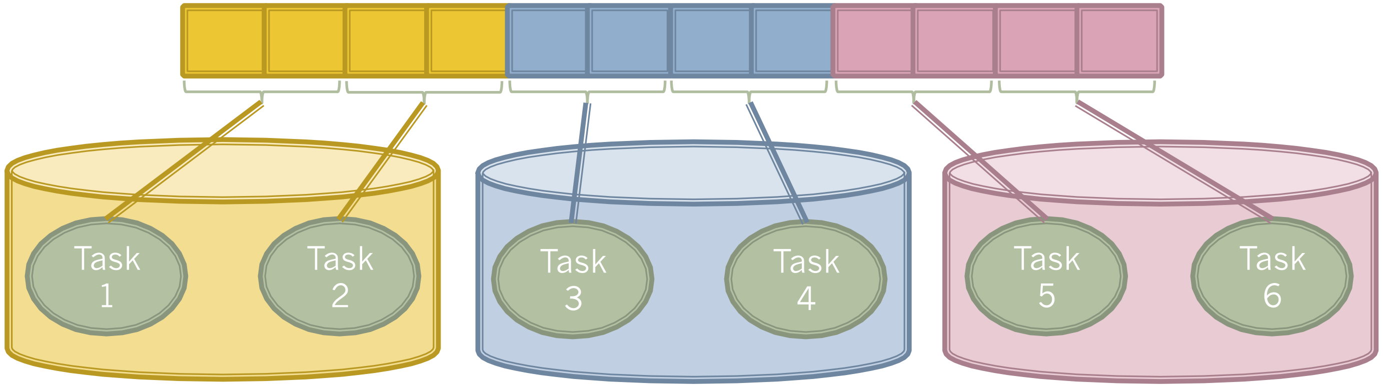
- **Input: collection of persons**
  - Person

```
    ...
    Integer age
    ...
     Boolean isOlder (limit, outCollection) {
                 if (age>limit) add self into outCollection
    }
```

- **Output: collection of persons older than a given age (*limit*)**

"Per object" **parallelism**

- **COMPSs "instantiates" one worker per object**
  - Iterates over a collection using a standard iterator
    - Instantiates the method in the node where the object is
      - Targeted at object methods
      - `getLocations`
    - Blocking may be needed
      - Object-method granularity may be too small
      - It implies grouping objects in the same backend



“Per object” **parallelism**

- **Declare method `isOlder` as a parallel task**
- **Code**

```
For element in the collection
 // For each element
 // This method is executed in parallel
 // in the node where the data is
  element.isOlder(age)
```

- **Parallelizing for each element may be too small**
  - Blocking

“Per object” **parallelism**

- **Create a new method `isOlderBlocking(age,ini,num)`**

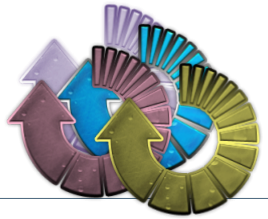
```
For element between ini and ini+num
  element.isOlder(age)
```

- **Code**

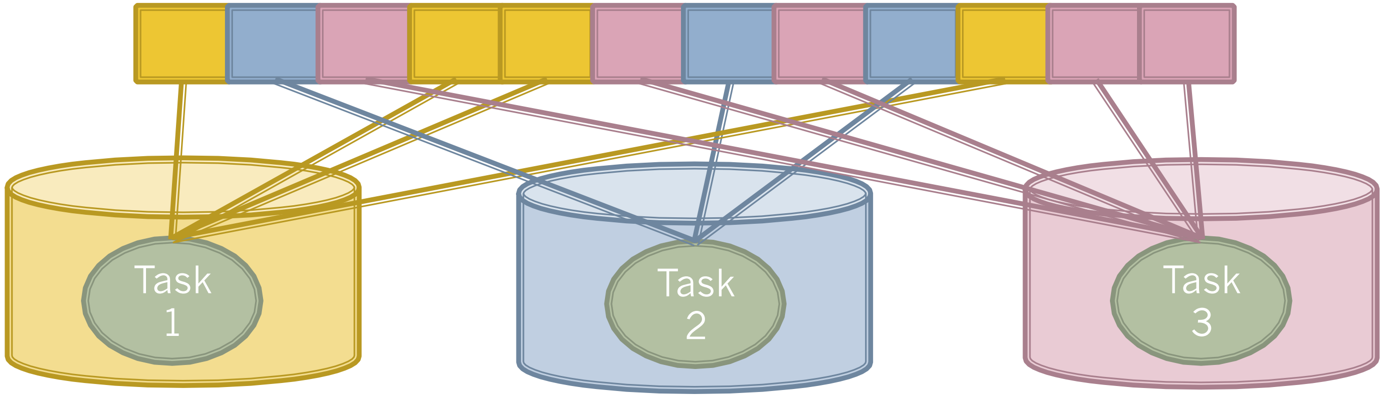
```
For i in (#elements in collection/block)
 // For each element
 // This method is executed in parallel
  element.isOlderBloking(age,i*block, block)
```

- **Now we have the right granularity**
  - The scientist needs to define blocking size
  - And placement if locality is important!!!

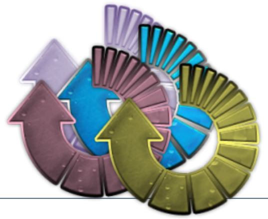
“*Per backend*” **parallelism**



- **COMPSs ‘‘instantiates’’ one worker per backend**
  - Obtains all locations using on the collection
    - `getLocations`
  - Each task executes a collection method
    - Iterates over a “*local*” iterator
      - Will only return objects in the current back end
      - Work stealing may be implemented if needed



“*Per backend*” **parallelism**



- **Create a new collection method `isOlderCollection(age)`**

```
For element in collection using local iterator
 // No parallelism here
 element.isOlder(age)
```

- **Define this method as “parallel”**
- **Code**

```
// Parallelism: executed in all backends with
// elements
isOlderCollection (age)
```

- **Now we have the right granularity**
  - The scientists has not done different code
    - Only encapsulated and used a “local” iterator

"Other" iterators

- **These are just examples, other iterators could be defined**
  - To implement locality as in a close backend
  - To implement work stealing
  - To take into account heterogeneity
- **The iterators are implemented as general in the collection**
  - Scientist only need to understand what they do
    - And use them

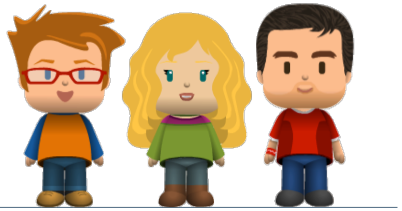
Agenda

- The motivation
  The pillars
- The technology
  The dark side
- The integration into the parallel programming language
  The secret potential
- Conclusions
  Time to wake up!

Conclusions
Ideas to take back home

- **Integrating persistent data into the programming model**
  - Unifies the model for both persistent and volatile data
  - Simplifies the decision of where to compute
    - Code is part of the data
  - Enables the use of data parallelism
    - Iterators can be adapted transparently to the programmer
  - Enables data distribution
    - Behavior policies are embedded

I travel, they do the work
Thanks to …



- **Current team**
  - **Anna Queralt**
  - Jonathan Martí
  - Daniel Gasull
  - Juanjo Costa
  - Alex Barceló

- **Master students**
  - David Gracia
  - Christos Ioannidis

- **Former team members**
  - Ernest Artiaga