Scientific Workflows: a Bottom-Up Perspective

Bruno Raffin,
DataMove
INRIA Grenoble Rhône-Alpes

ORAP, March 2019

Motivation

« Classical » workflows:

- Code coupling
- Data exchange through files
- Grid support

Workflows from a bottom-up perspective:

- Performance (No intermediate files)
- Code development and maintenance
- Tool coupling for added functionalities
  - Benfit from new programming paradigm shifts

Two Phase I/O

Goal: get I/Os outside of the simulation code for performance purpose

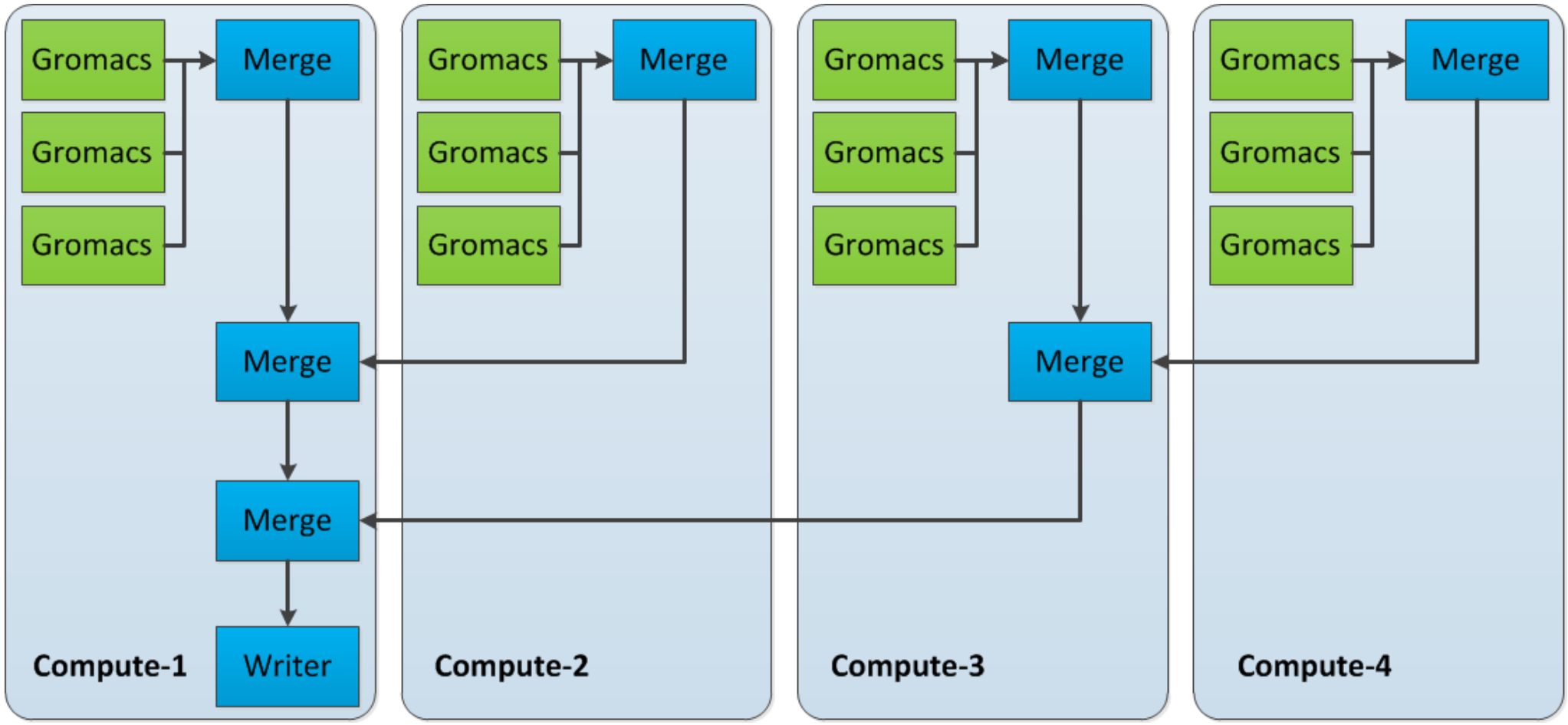
How:

- Overlap I/Os and computation with asynchronous I/Os
- Aggregate data per nodes

But also better code modularity

Examples: Adios, XSIO

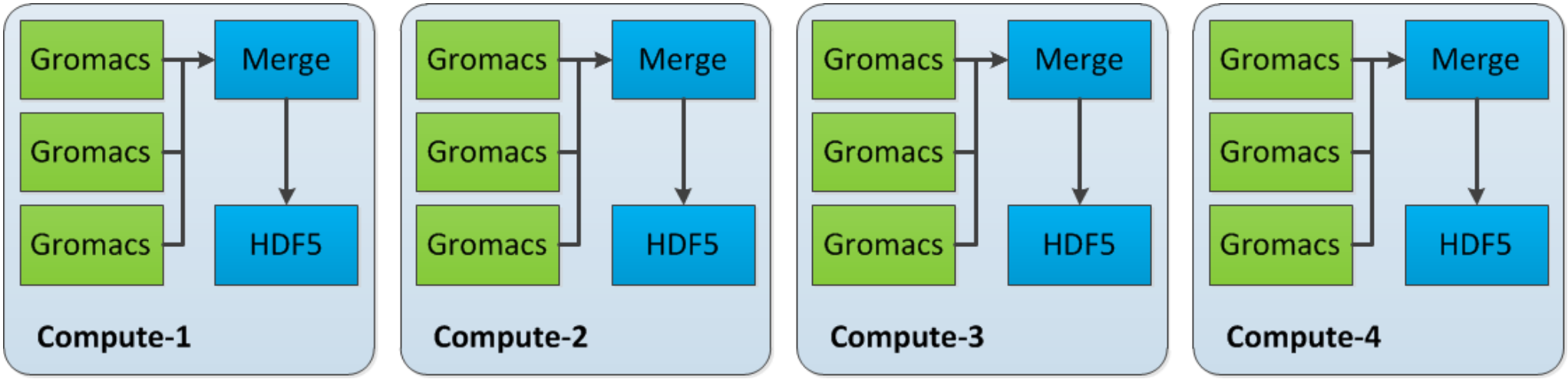
Two Phase I/O Example



Aggregate the results in situ on helper-core (1 per node) to the master node

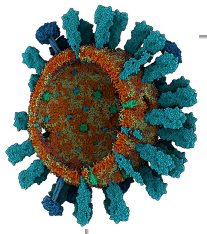
Gromacs: standard molecular dynamics simulation code

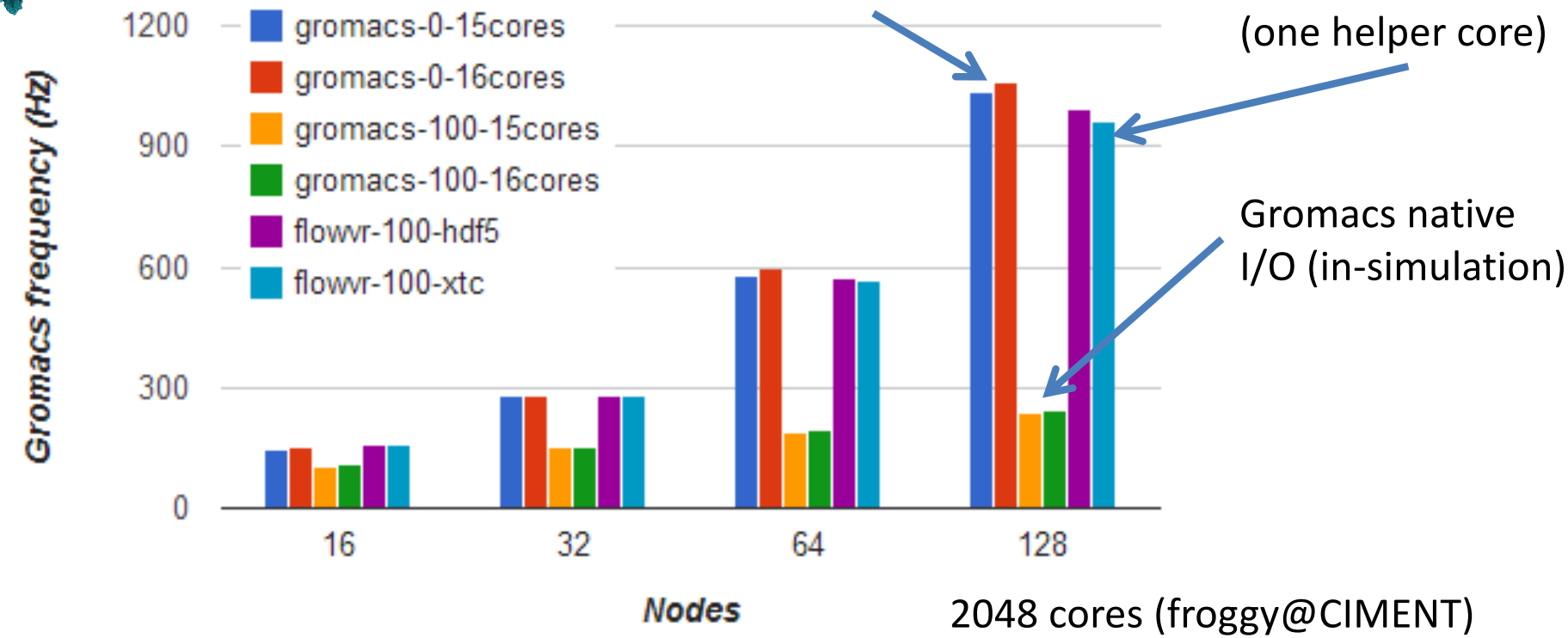
Two Phase I/O Example



Aggregate local results in situ on helper-core (1 per node) and write to disk

Two Phase I/O Performance





Gromacs without I/O: **15 cores/node 3% slower than 16 cores/node**
(- 6% if scalability would have been perfect)

Developed with FlowVR

In Situ Processing

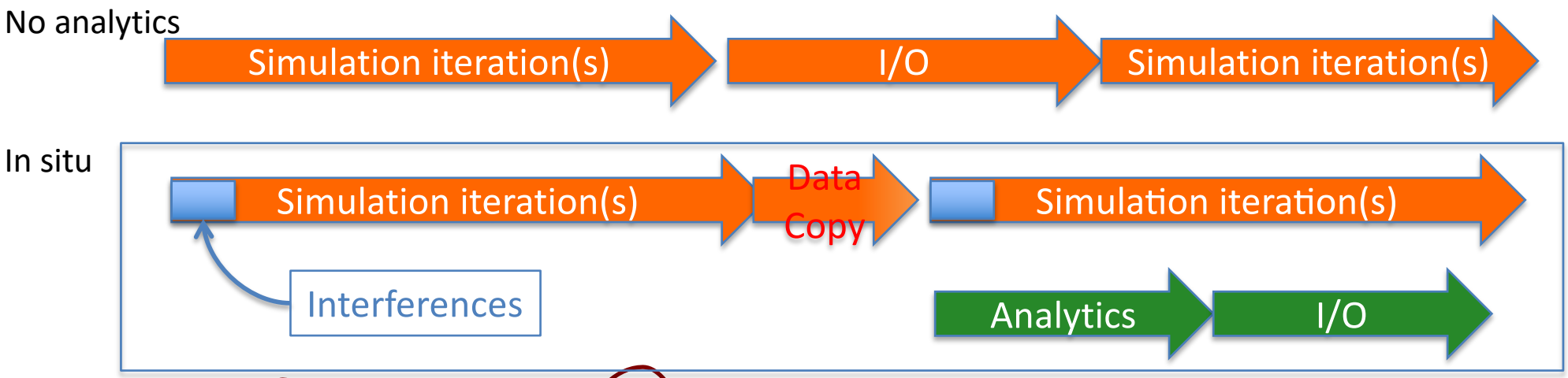
Goal: move from post mortem data processing to on-line/live analytics, i.e suppress intermediate files.

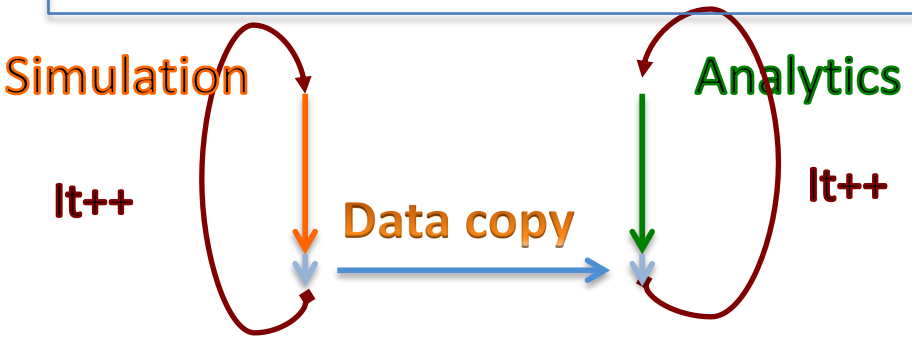
How:

- Overlap I/Os and analytics
- Aggregate data per nodes, process data on helper cores

Examples: Damaris, Decaf, Flexio, FlowVR, Tins

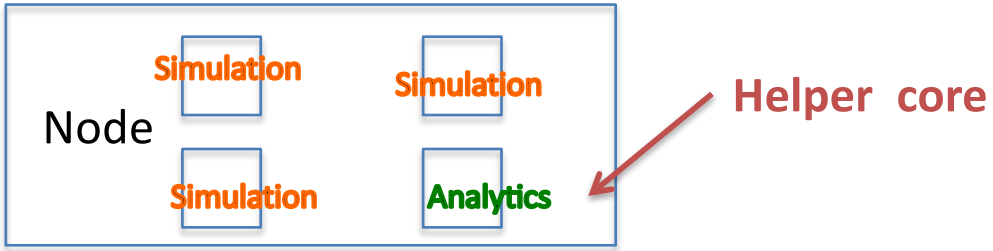
In Situ Processing



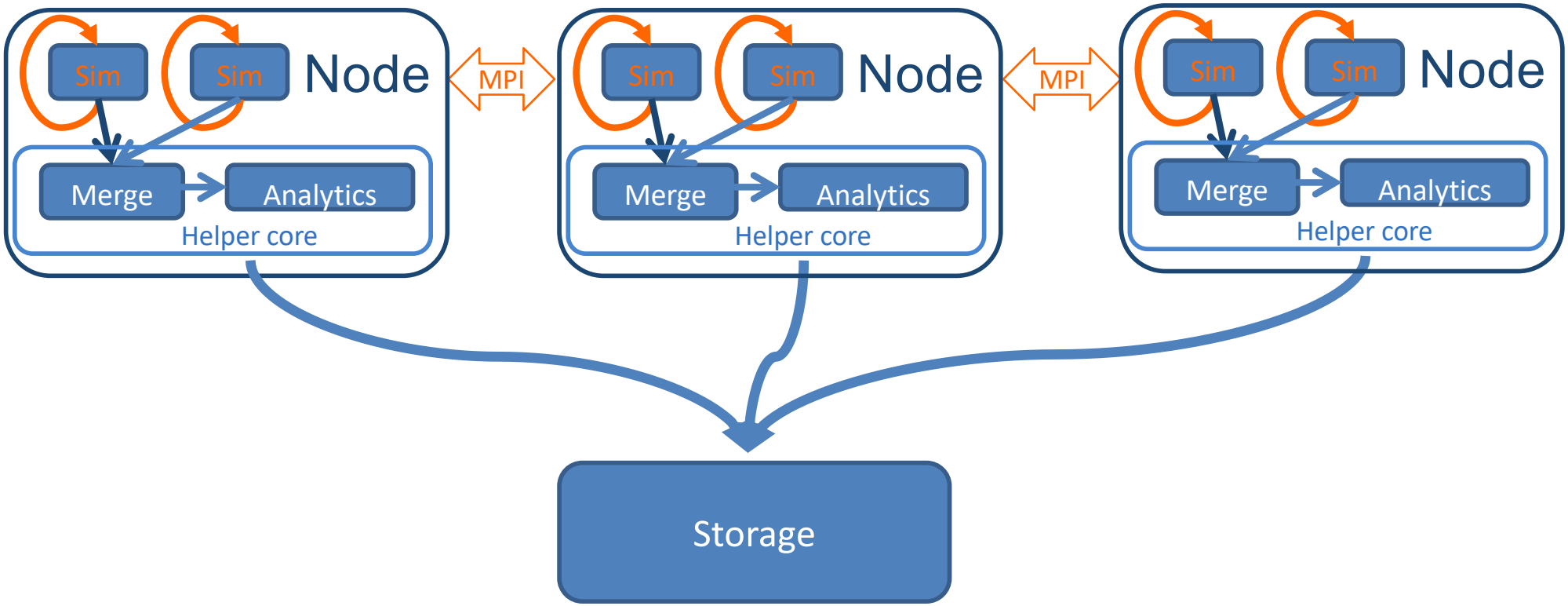


In situ:
simulation and analytics share the same nodes

Resource allocation strategies:
time sharing or space sharing
(dedicated helper core)



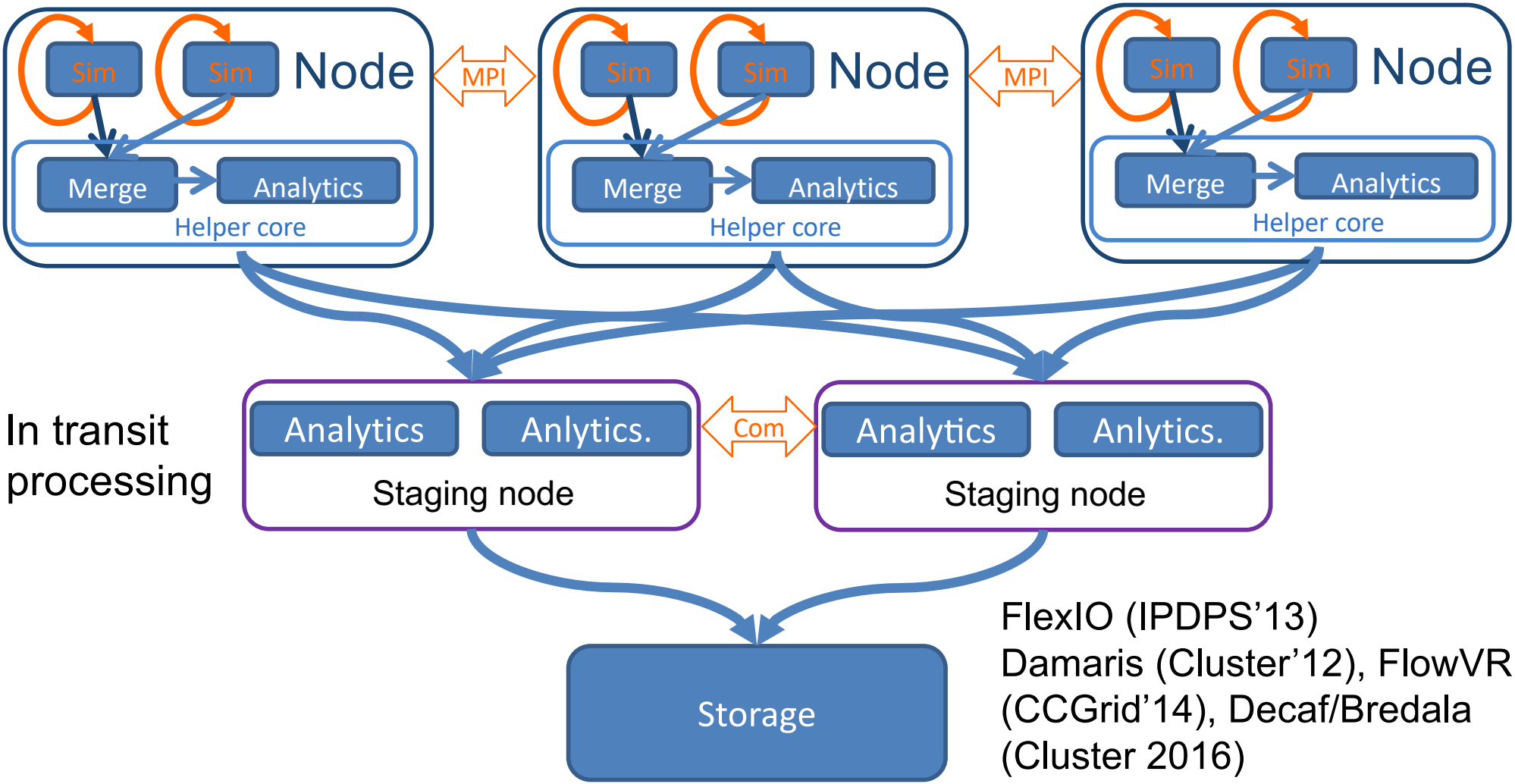
In Situ Processing



Most solutions are process based, but some like Tins [SCA 2017] show higher performance relying on tasks (based on intel TBB)

In Transit Processing

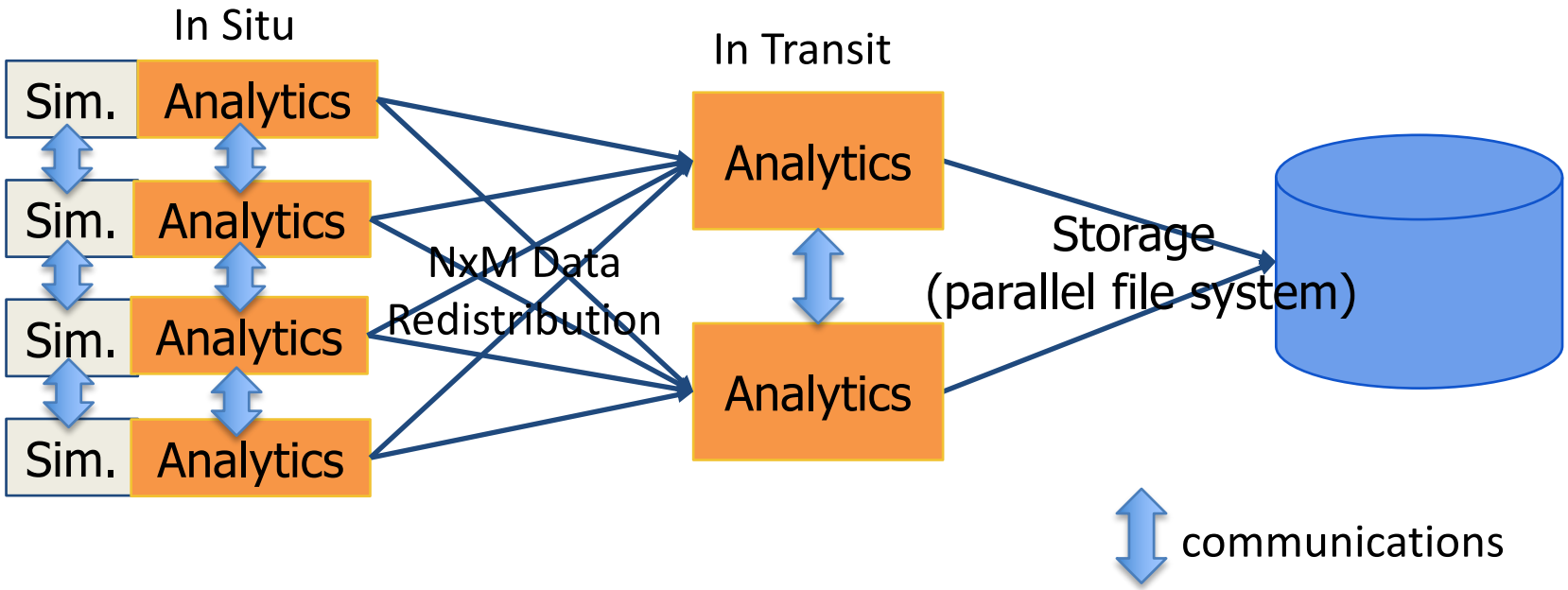
For practical or performance reasons, using nodes dedicated to analytics may be convenient to execute part of the analytics workflow



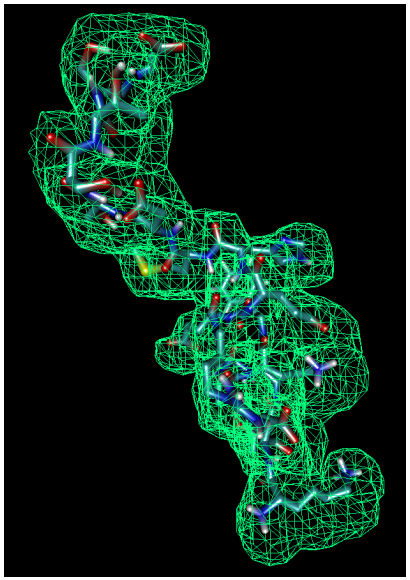
FlexIO (IPDPS'13)
Damaris (Cluster'12), FlowVR (CCGrid'14), Decaf/Bredala (Cluster 2016)

In Situ + In Transit Processing

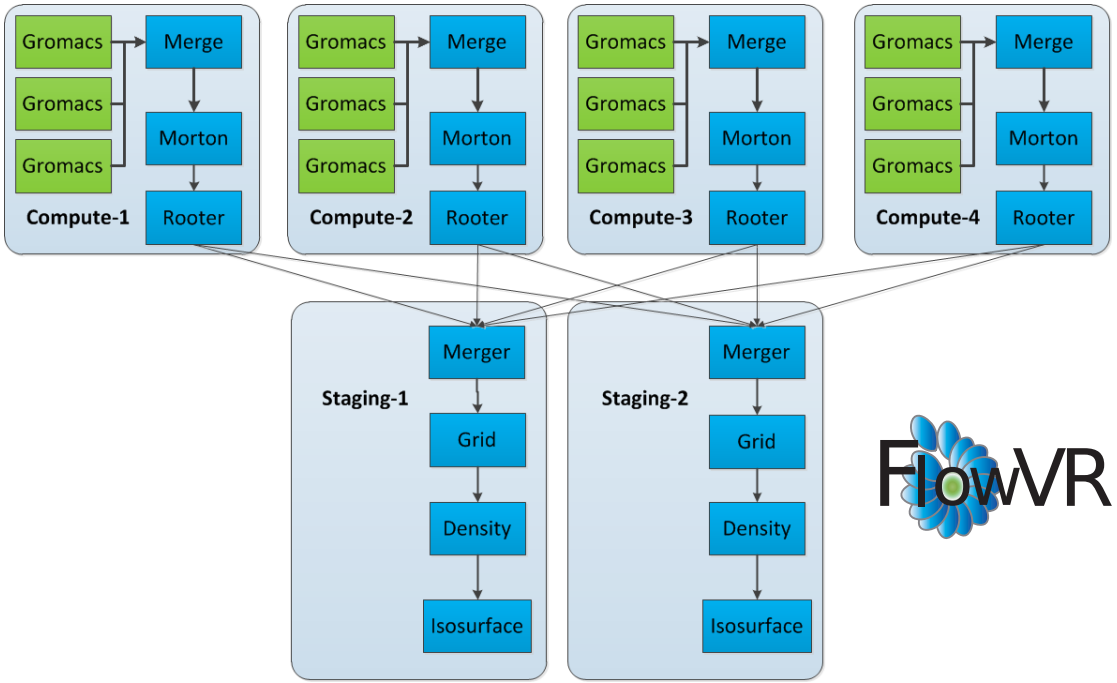
Analytics workflows can be complex calling for solutions that enable performance, modularity and flexibility



Example: Parallel In Situ Isosurface Extraction [Dreher,CCGRID'14]

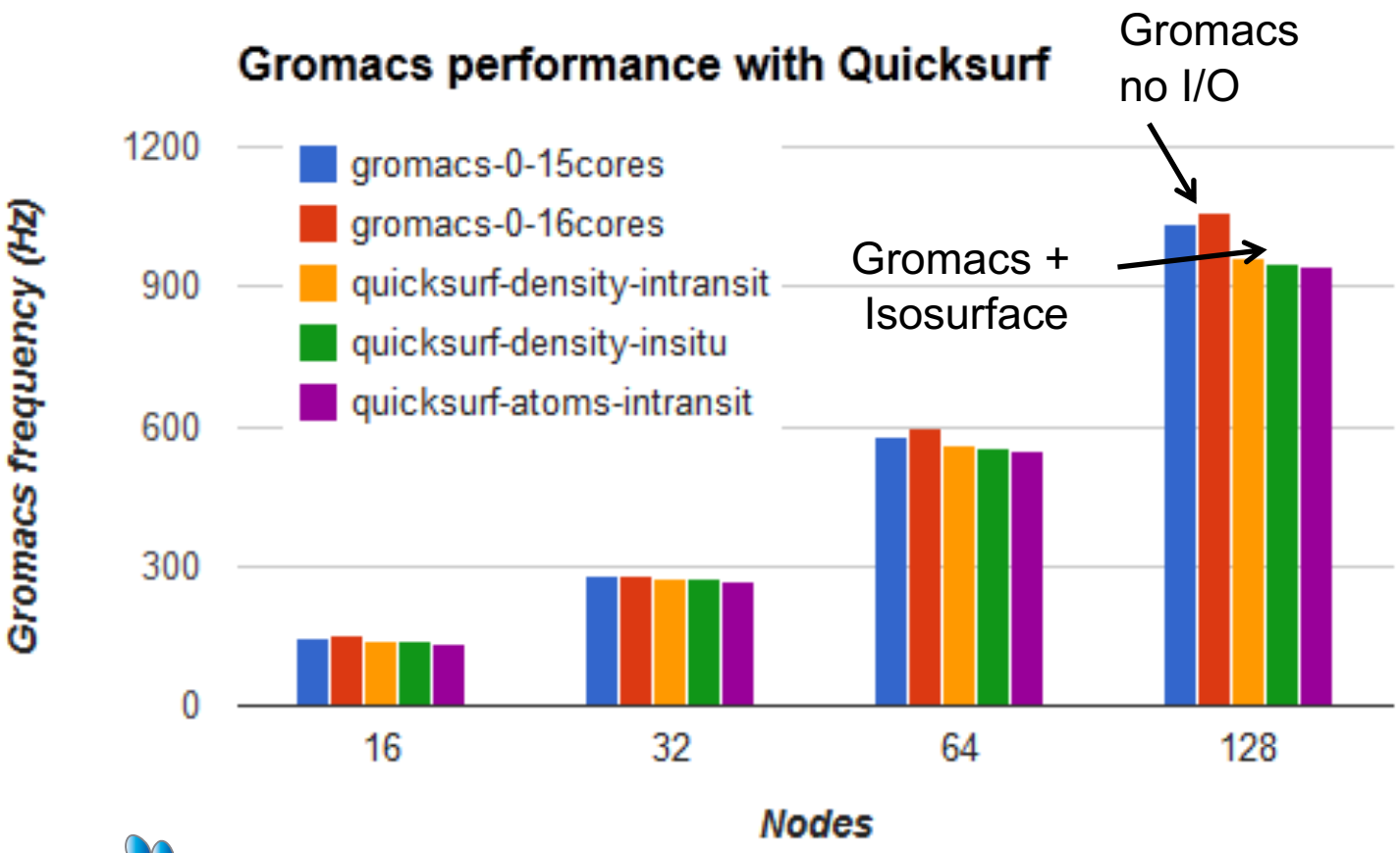


Compute a molecule surface based on atom density



Tested different distributions of processing steps to in situ and in transit nodes.

Performance [Dreher,CCGRID'14]



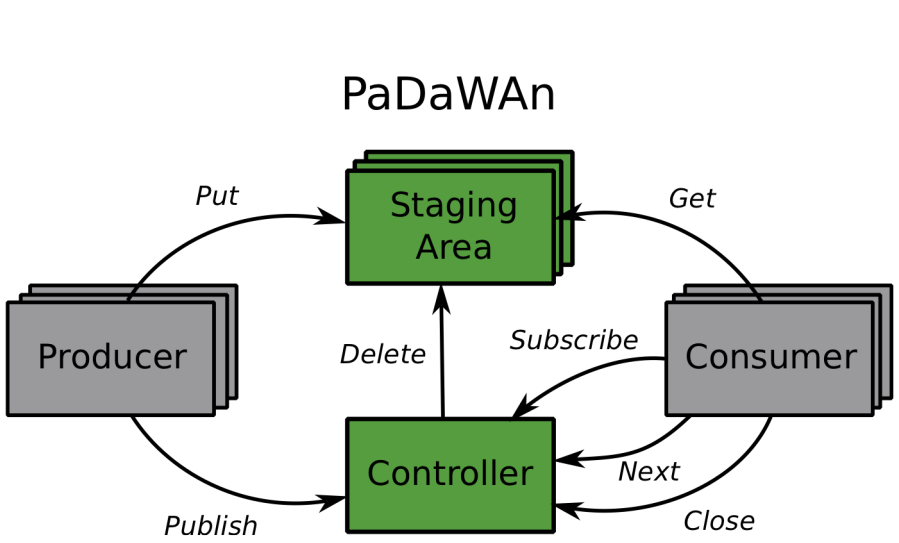
- In transit: 1 staging node every 64 compute nodes
- Density-intransit: costs 7% comp. to gromacs 15 cores
- Density-insitu costs 8% but use 1.5% less nodes than density-intransit
- Atoms-intransit costs 8.6% but enables other in-transit analytics (3x more data to move on stagging nodes than Density-intransit)

FlowVR

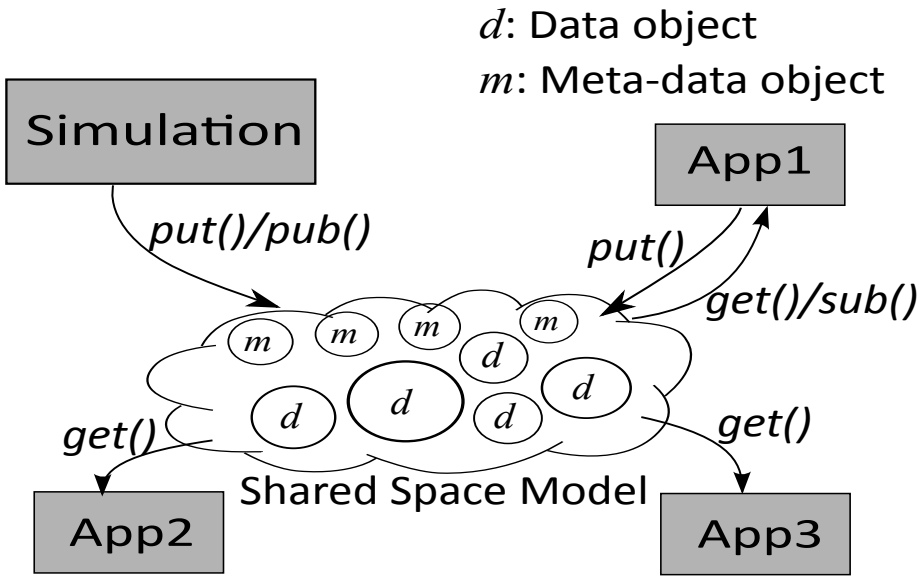
2048 cores (froggy@CIMENT)

Data Staging

Ephemeral in transit store on staging nodes



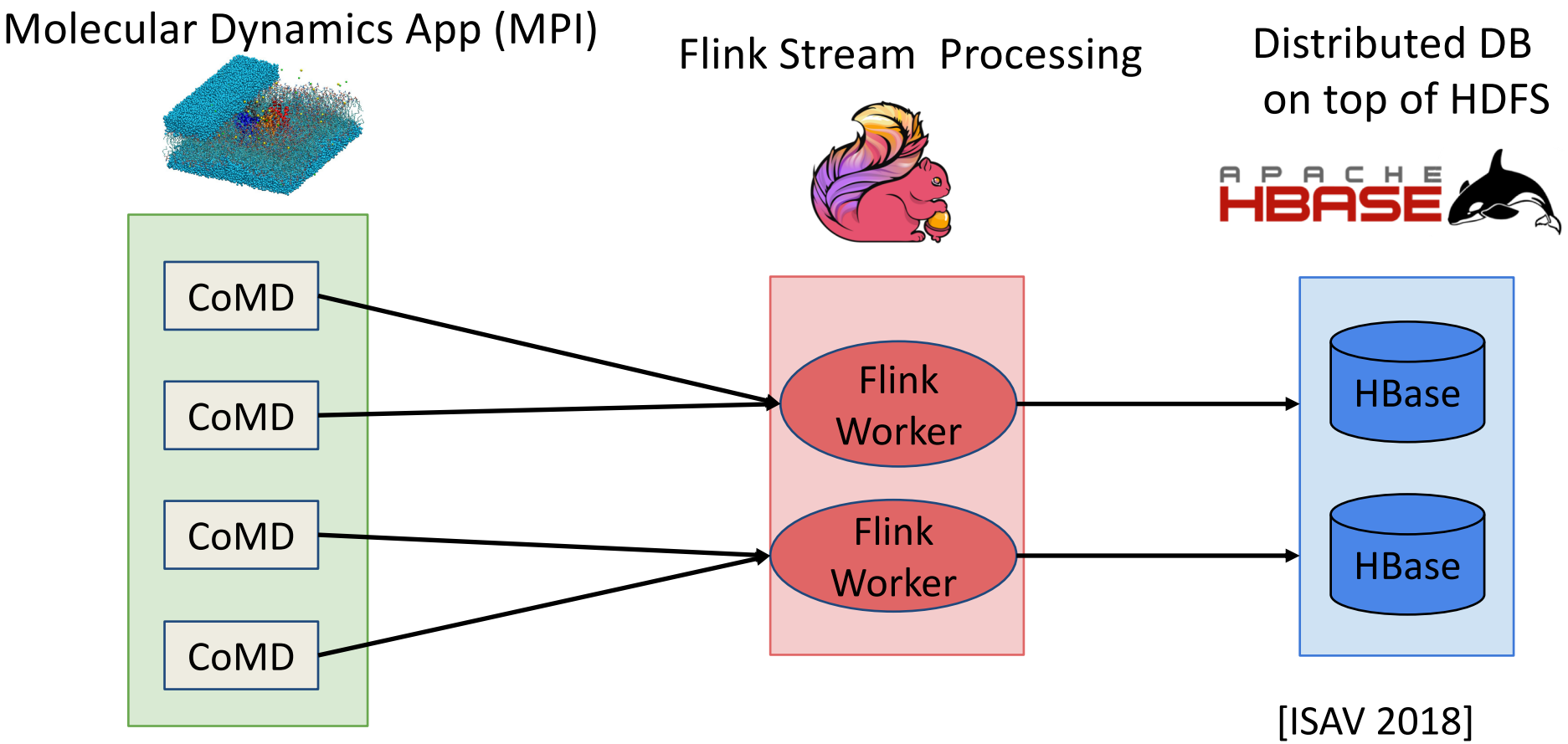
Padawan



DataSpace

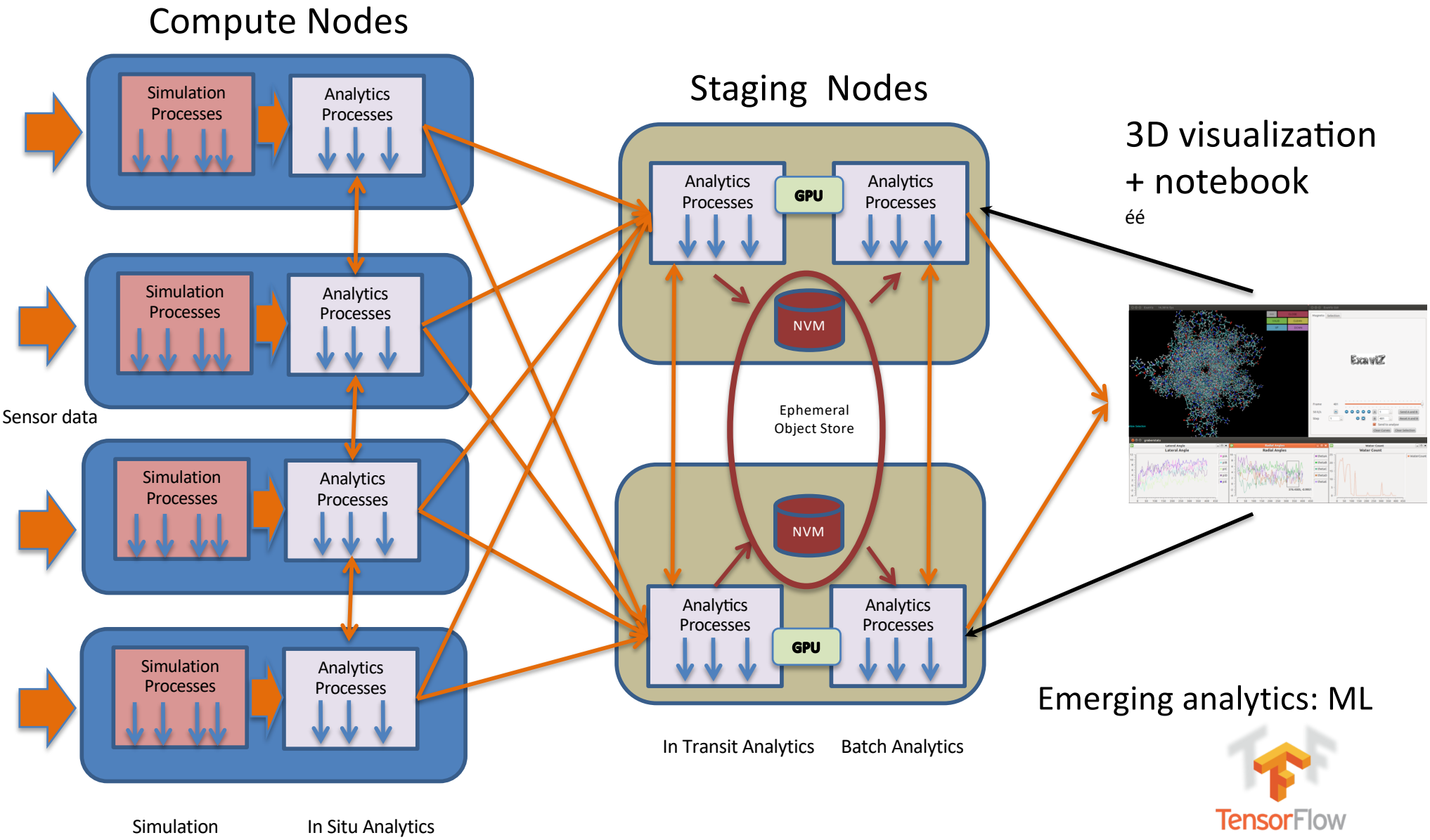
BigData/Stream Processing

Take benefit of “novel” programming frameworks (Map/Reduce)



[ISAV 2018]

High Performance Analytics



In Situ Processing Frameworks

MPI based: Damaris, Decaf, FlexIO

- One mpirun
- One or several executables (SPMD mode)
- Different communicators

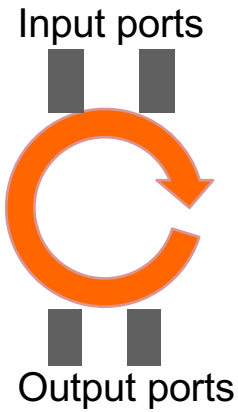
Limited modularity and eleasticity

Broker/daemon/service based: CCA, FlowVR

- Communications handled by a third party
- Enable coupling heterogenenous executables

The FlowVR Programming Model

1. Develop components:

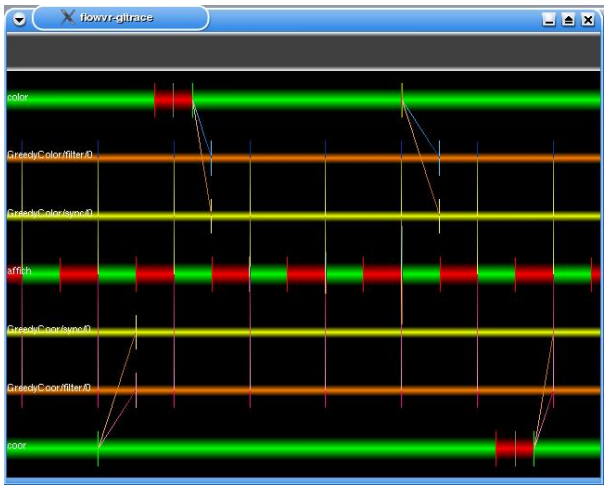


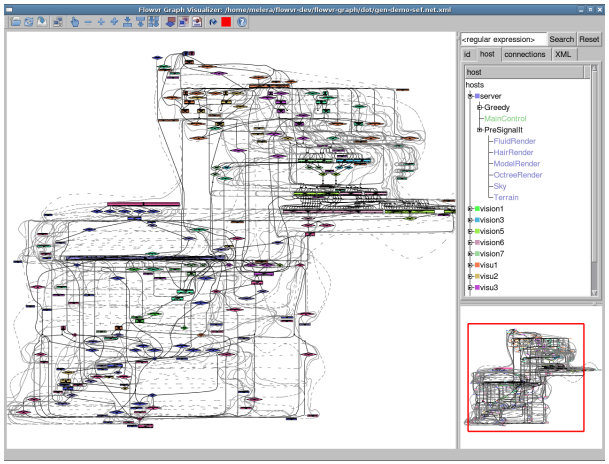

```
While ( wait(inputs) )
    get()
    compute()
    put()
end
```

Simple API (limit code intrusion)

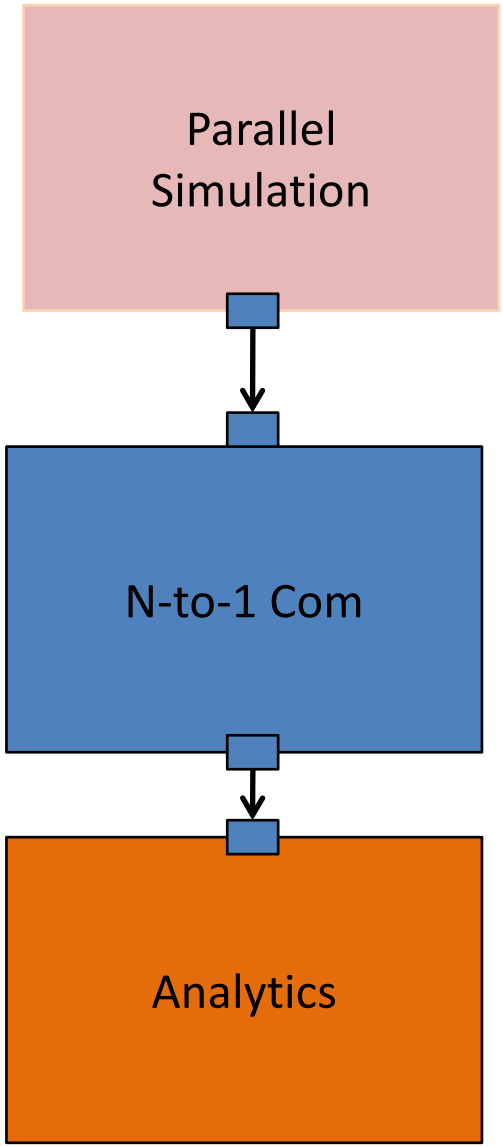
2. Assemble components: Python script
3. Instantiate parameters and execute script

http://flowvr.sf.net

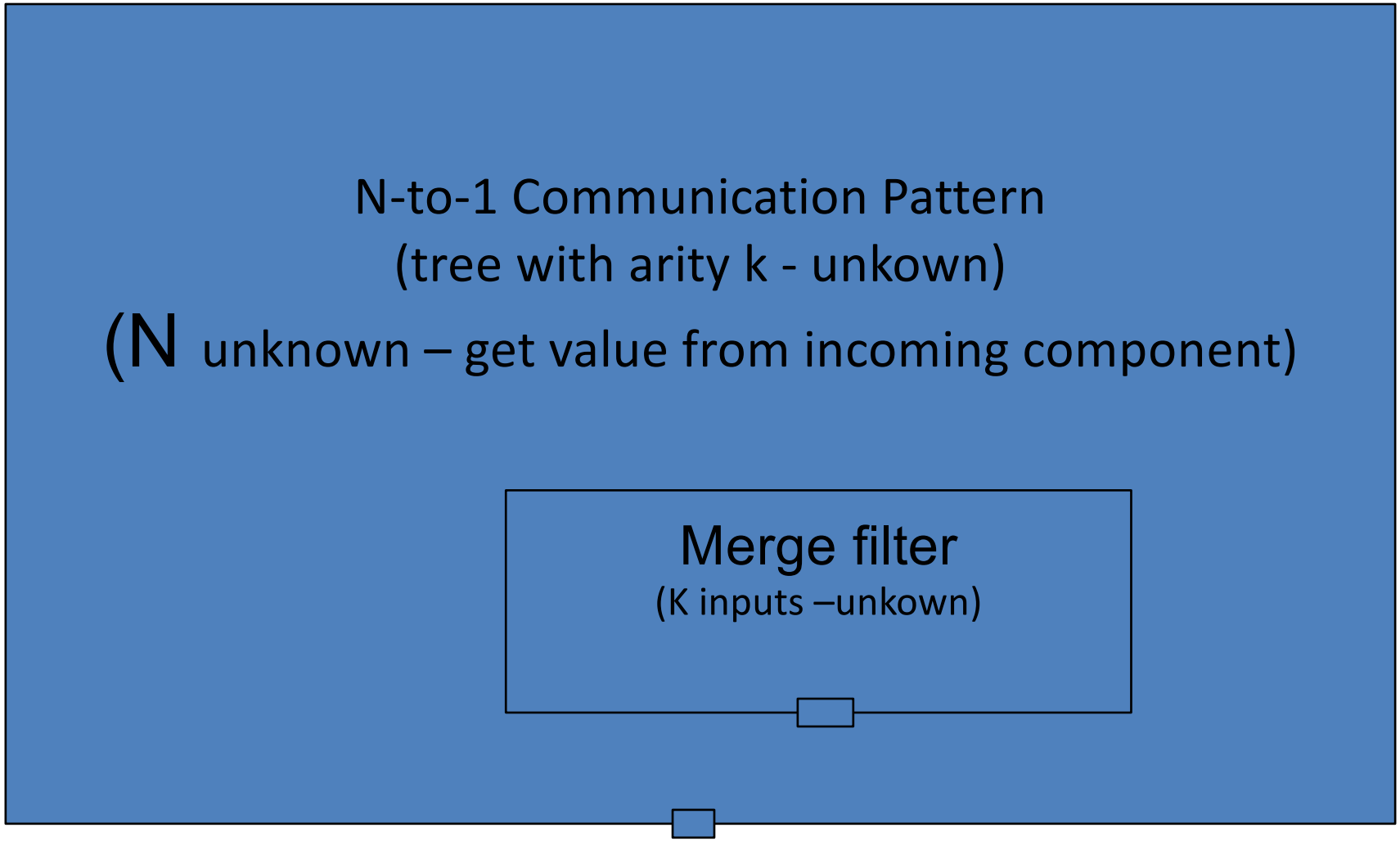




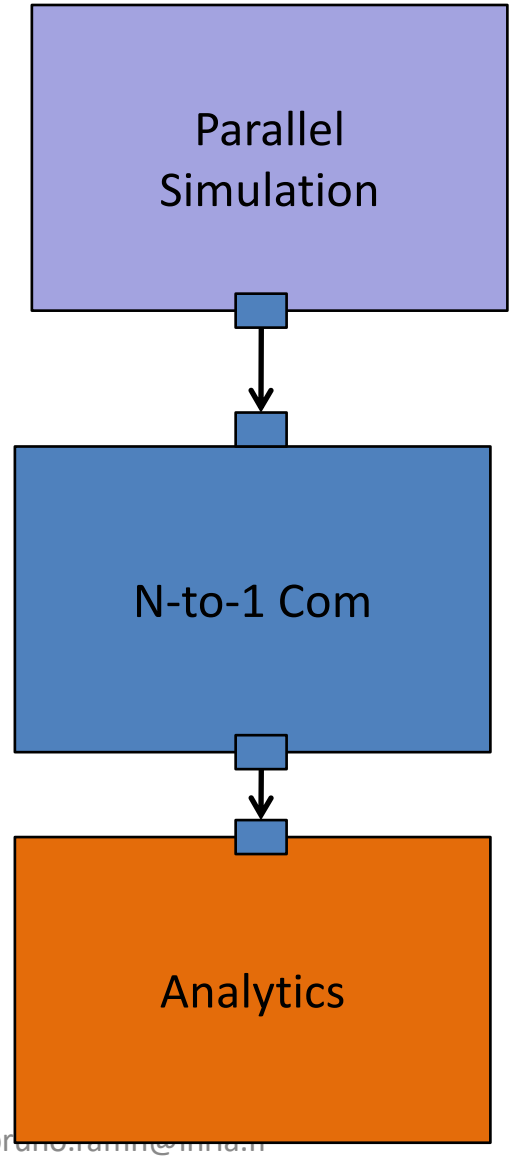
FlowVR: example



FlowVR: example



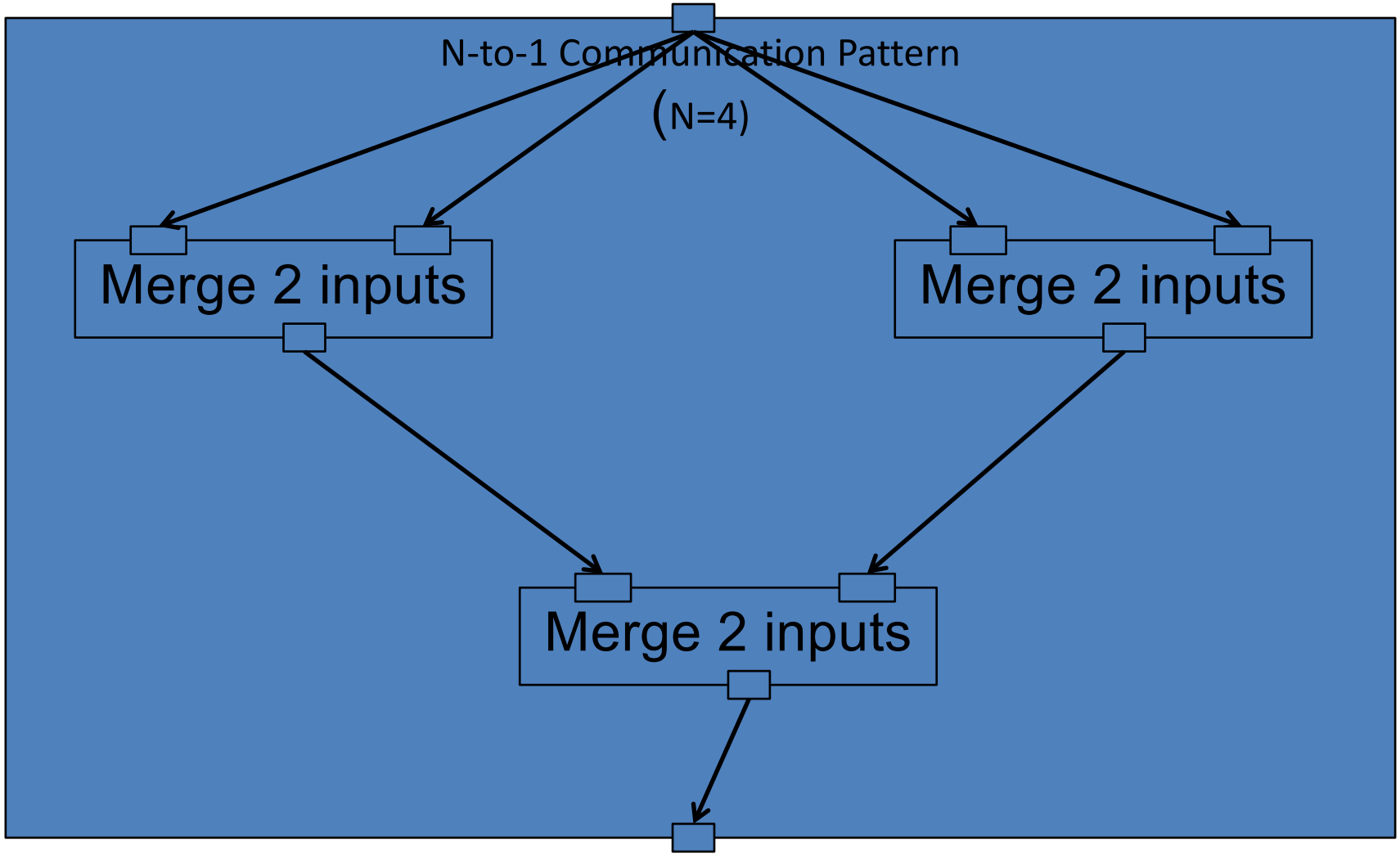
FlowVR: example



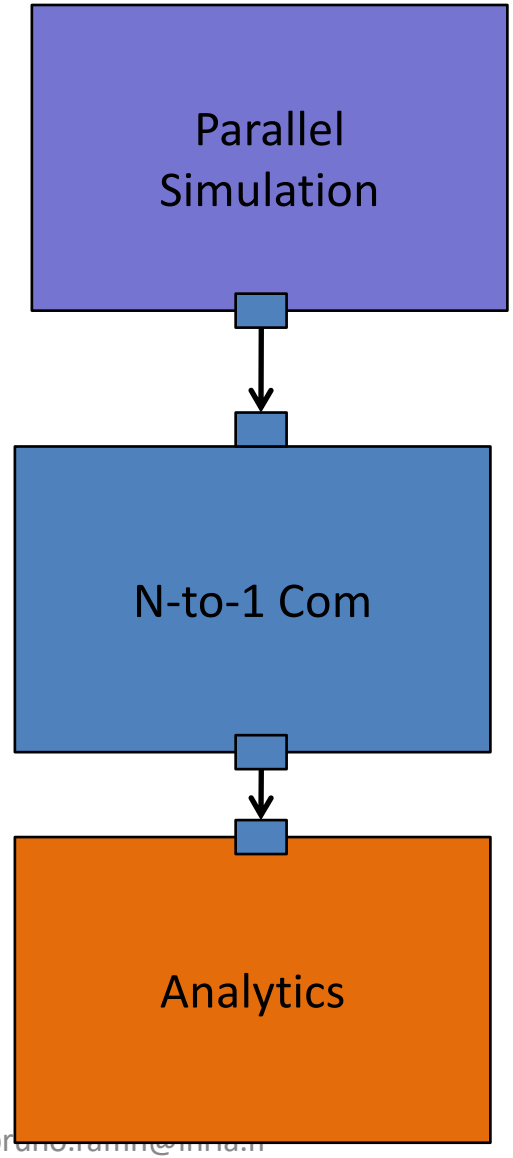
N=4

K=2

FlowVR: example

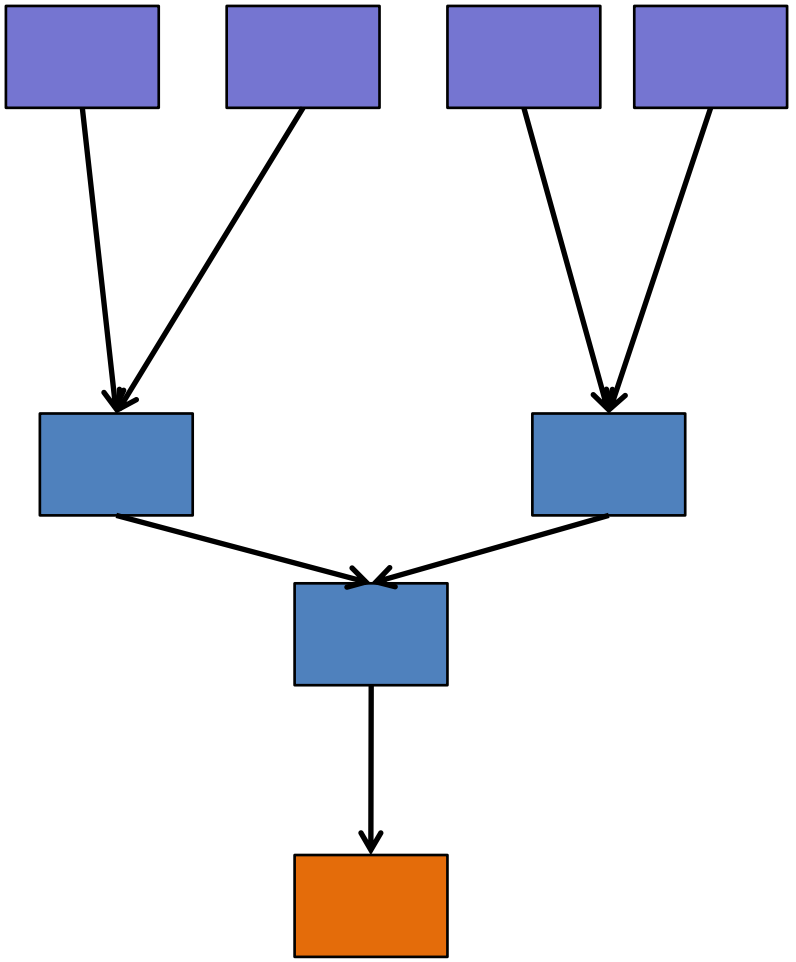


FlowVR: example



N=4

K=2



Let the Simulation Simulate

- Classical push paradigm: the simulation controls everything, in particular what/when data are pushed outside (for I/Os or analytics)

- Pull paradigm:
  - The simulation only exposes internal data structures
  - The consumer retrieves what is necessary when necessary

Data Model

Simulation

```
while ( !computation_finished ) {
/* fill data buffer */
It++;
PDI_Expose (’’newdata’’, ‘’buffer", &buffer,size,"Iteration",it);
}
```

Contract:

Data Model

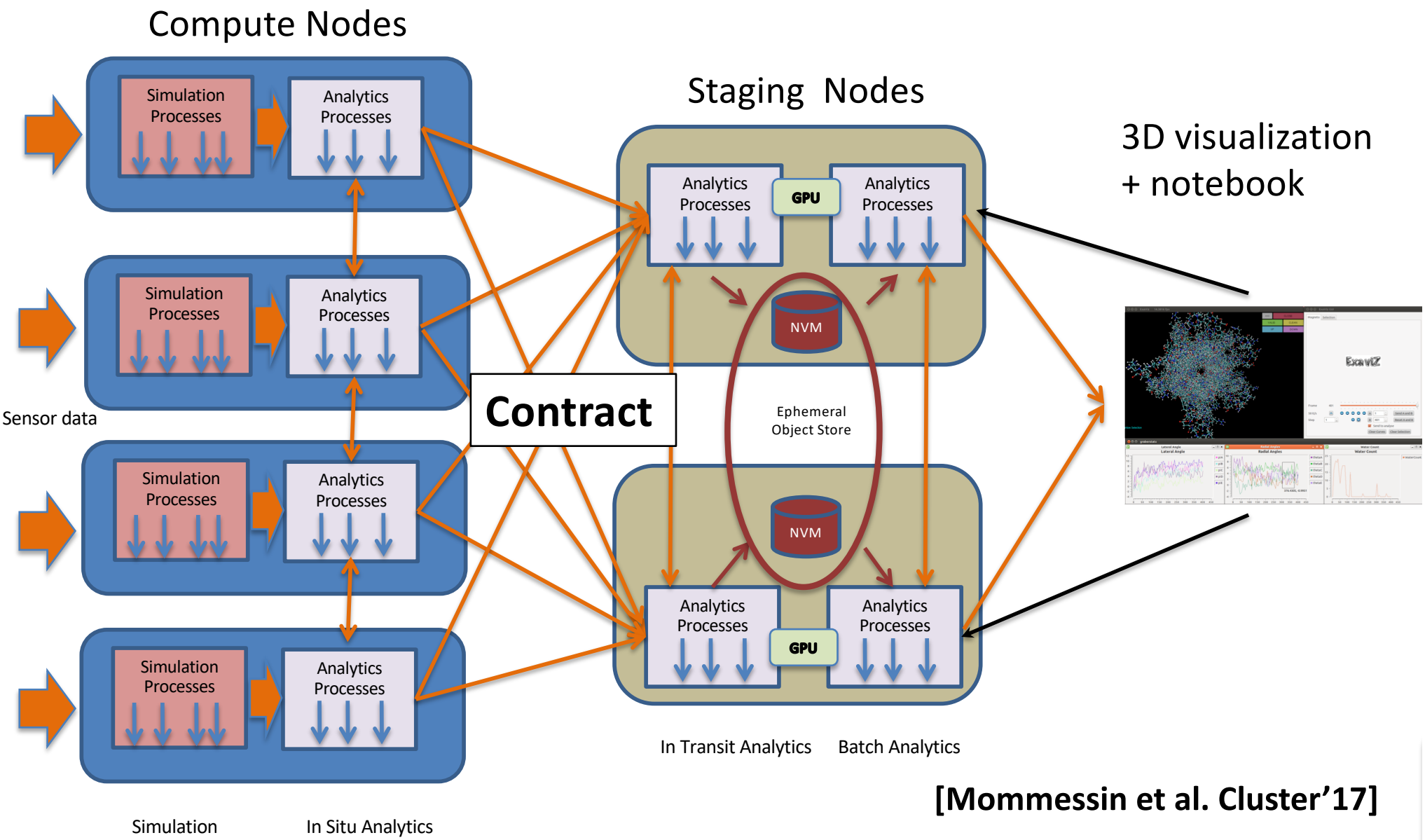
```
Iteration: int
buffer: { type: array, subtype: double, size}
On_event: newdata trigger  foo("buffer","Iteration")
```

Pull code

```
foo(){
        PDI_get("Iteration",&it)
        PDI_get ("buffer", &buffer,&size);
        if (need data at iteration it)  do_something_with(&buffer, size);}
```

Examples: Damaris, Adios, VTK/Ascent, **PDI**

High Performance Analytics



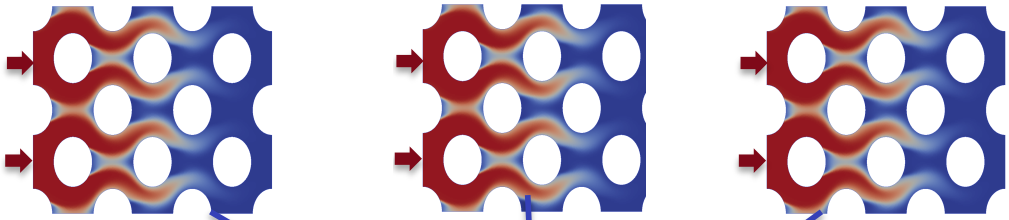
[Mommessin et al. Cluster'17]

Ensemble Runs

- From one to many simulation runs (with different parameters)
  - Statistics
  - Sensibility analysis, uncertainty quantification
- Example with molecular dynamics: Copernicus (SC'11)
  - Single simulation hard to scale (latency bounded)
  - Ensemble runs with adaptive sampling
  - « Killer use case » for exascale computing

Ensemble Runs: Sensitivity Analysis

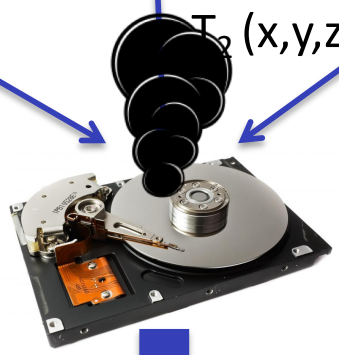
P simulations with different parameter values

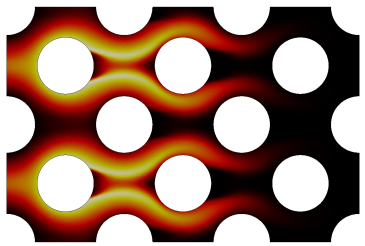


Output: one temperature $T_1(x,y,z,t)$ per mesh cell and timestep

$T_2(x,y,z,t)$

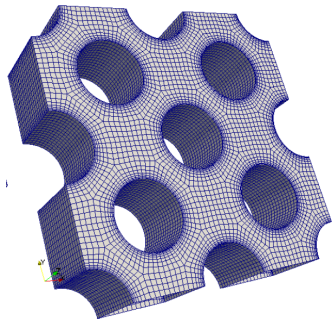
$T_p(x,y,z,t)$





[Terraz et al. SC'17]

Collab. with EDF



I/O bottleneck at scale:

-slower simulations

-slower analytics

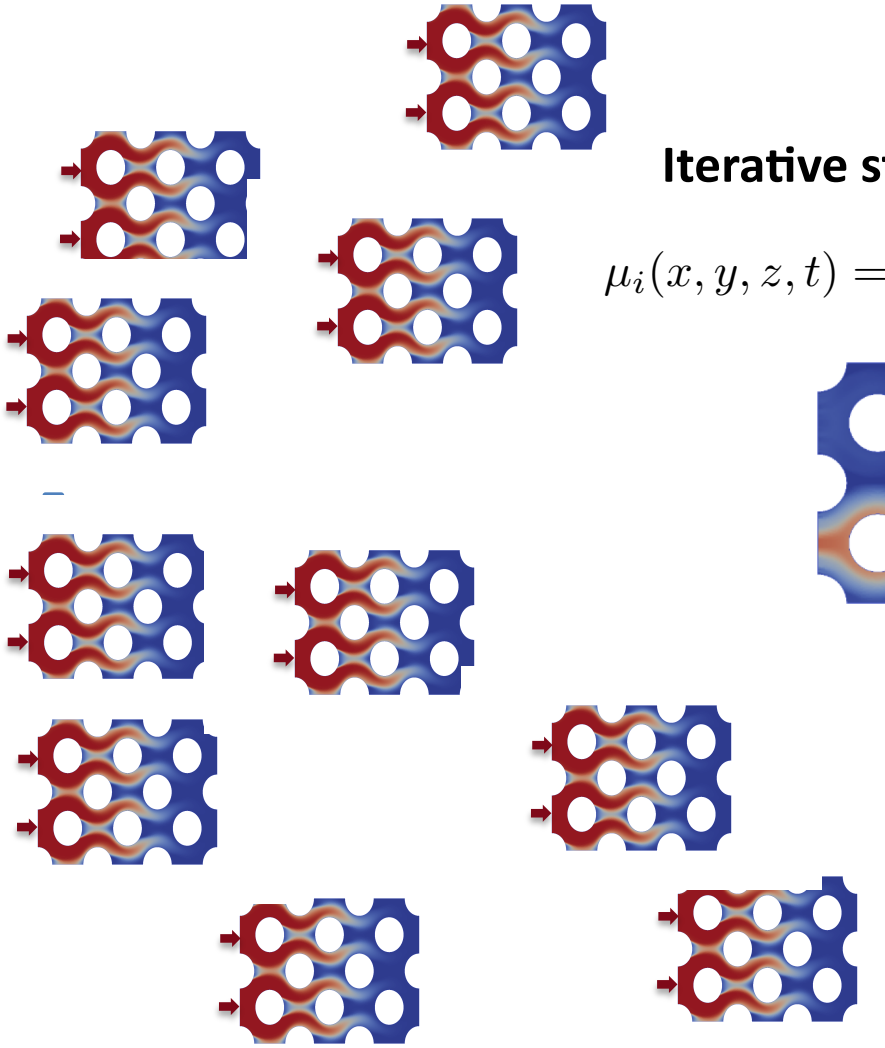
Basic solution: statistics at low resolution

Analytics: spatiol-temporal statistics

E.g. $T_{avg}(x,y,z,t) = \frac{1}{p}\sum_{i=1,p} T_i(x,y,z,t)$

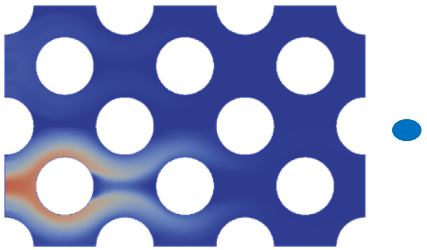
for a selection of probes (x,y,z,t)

In Transit Sensitivity Analysis



Iterative statistics: one-pass algorithms

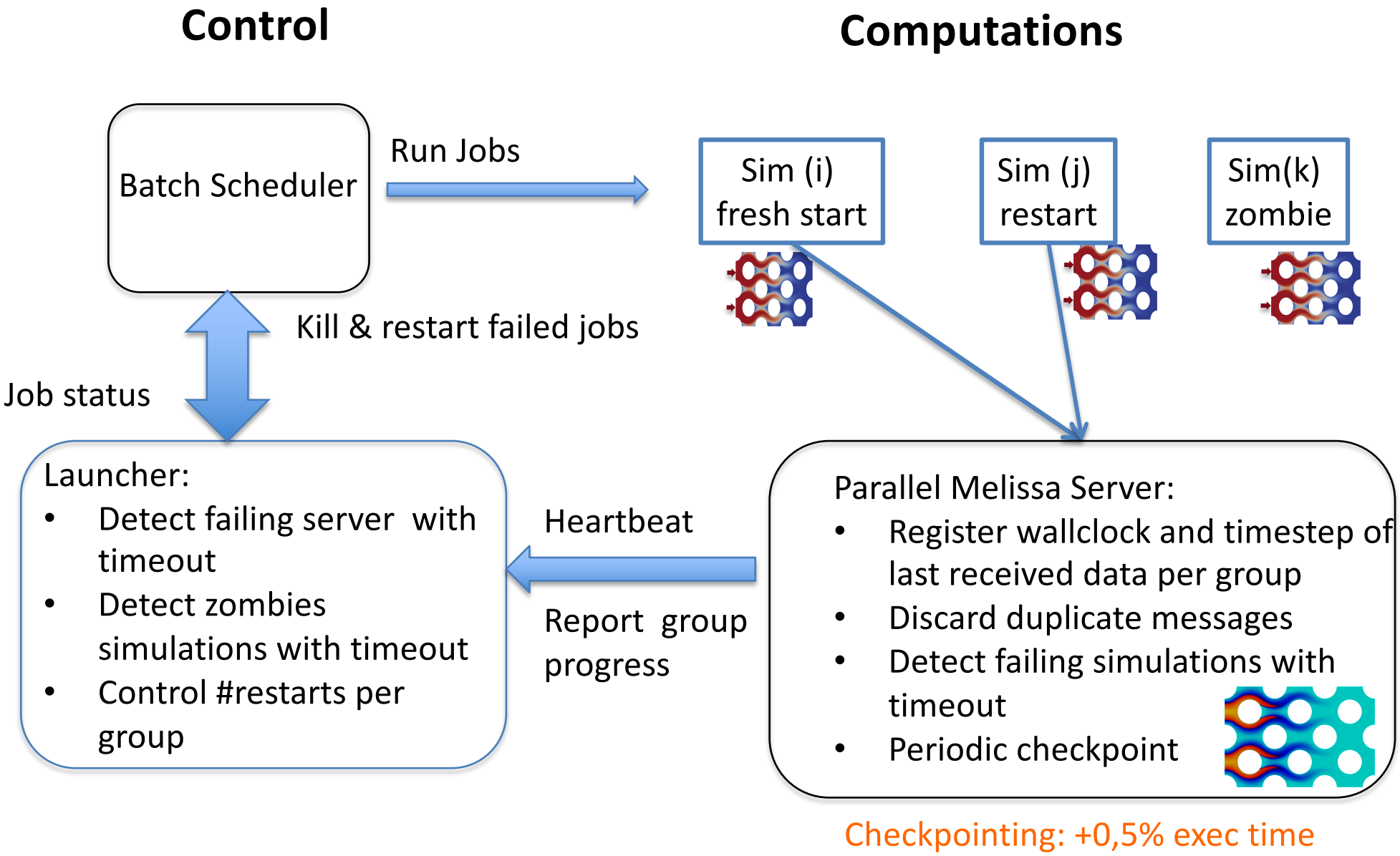
$$\mu_i(x,y,z,t) = \mu_{i-1}(x,y,z,t) + \frac{1}{i}(u_i(x,y,z,t) - \mu_{i-1}(x,y,z,t))$$



MELISSA

Supported stats: average, variances, skewness, kurtosis, min max, threshold exceedance, Sobol' indices, quantiles

Fault Tolerance & Elasticity

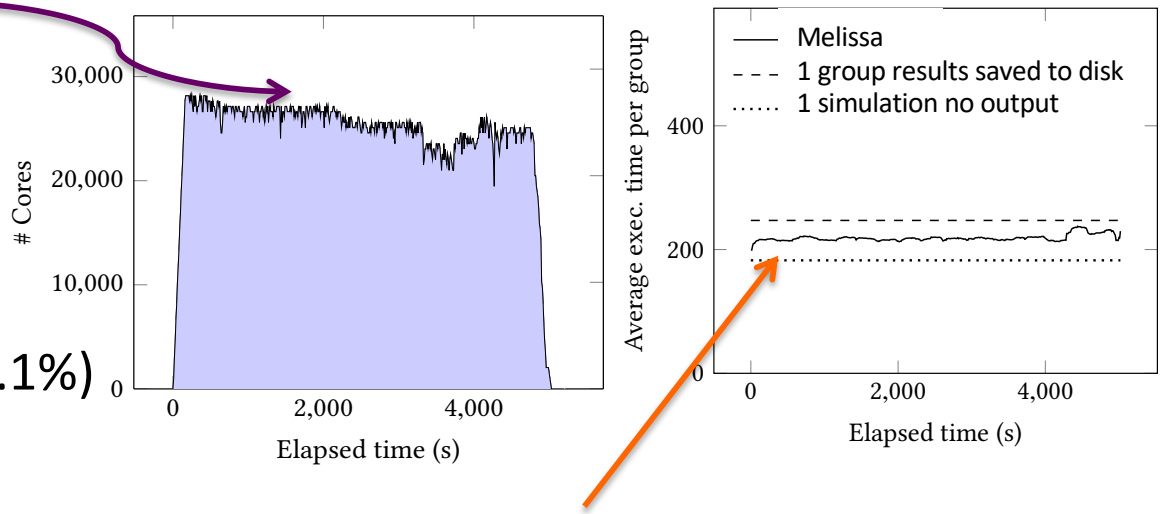


Melissa Experiments

Elastic execution
up to 28K cores

8000 simulation runs
(512 cores each)

32 server nodes: 740 CPU.h (2.1%)
Simulations: 34000 CPU.h



Simulation runs 13% faster on average than when writing to disk

---

Massive run:

- 80 000 simulation runs (24 cores each)
- 271 TB of data processed online (and thus not saved to disk)

Ensemble Runs

Other use cases:

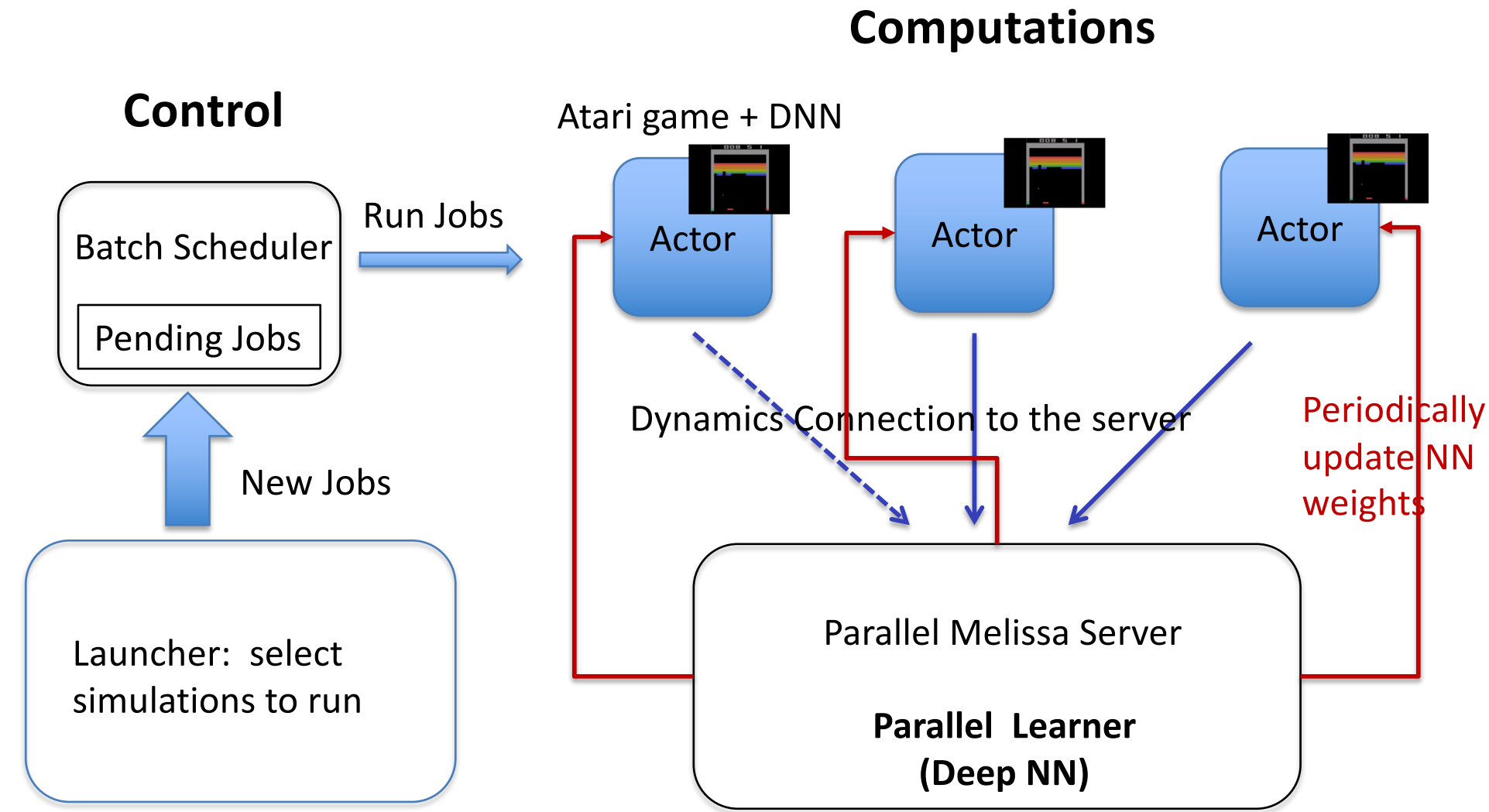
Statistical Data Assimilation

Simulation are driven by data from observations (EnKF)

Deep Reinforcement Learning

Self-learn a score optimizing strategy from simulations (AlphaGoZero)

Melissa for DRL at Scale



Join work between the DataMove And Sequel teams

Goal: to go beyond AlphaGoZero

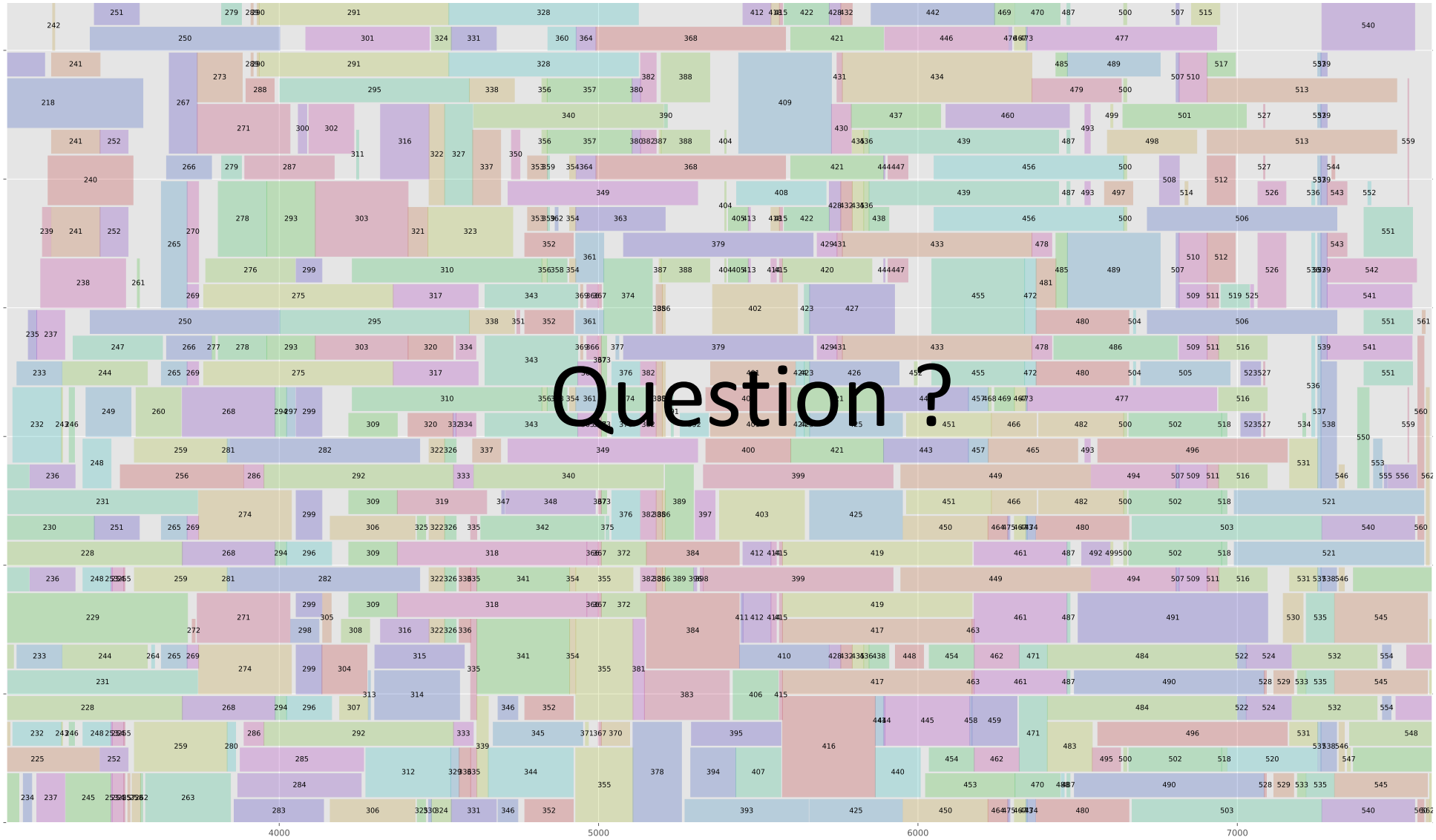
Conclusion

Workflows driven by needs of:

- Performance (the I/O bottleneck)
- Data analytics

Emerging needs:

- Data assimilation
- Ensemble runs
- Deep learning


Question ?
0
5
10
15
20
25
30
4000
5000
6000
7000