BDMC 2012

Workshop on Big Data Management in Clouds

Turning Big data into knowledge

Techniques and Tools for Parallel Computing on Online Data Streams in Systems Biology and Epidemiology

Marco Aldinucci
University of Torino, Italy

HiPEAC COMPILATION ARCHITECTURE

PARAPHRASE

IMPACT

BioBITs

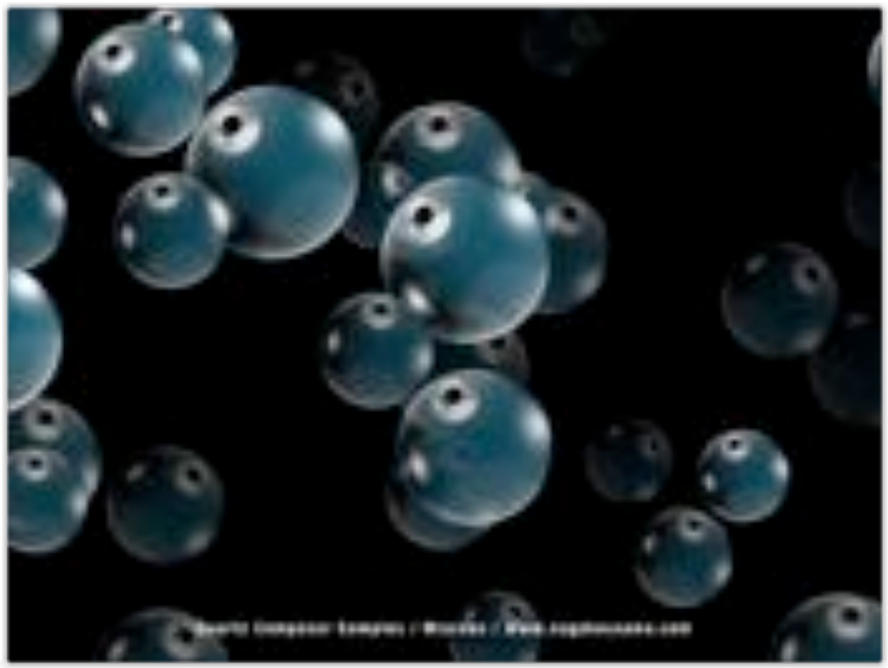
ETICA Tecnologie e servizi

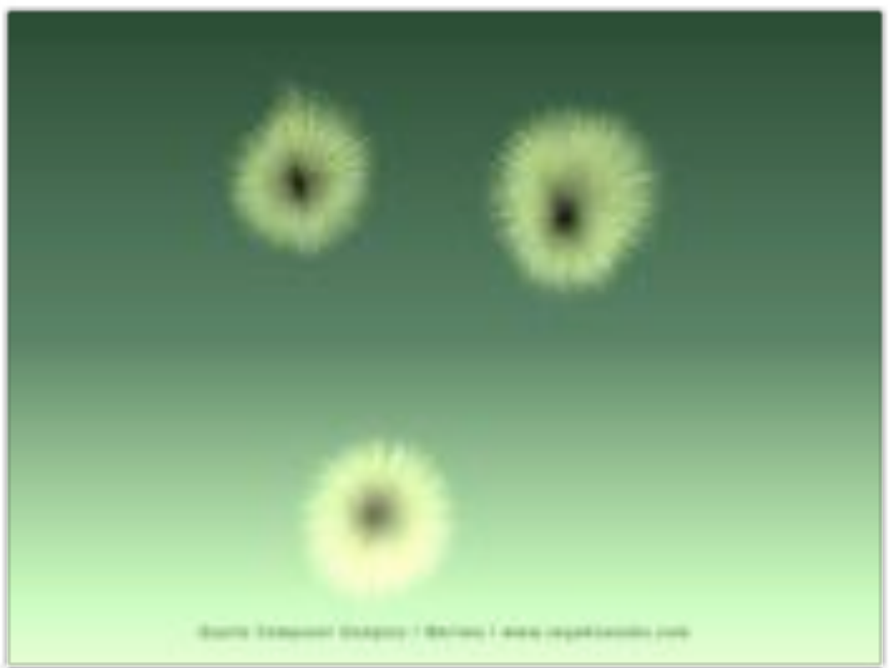
UNIVERSITÀ DEGLI STUDI DI TORINO
ALMA UNIVERSITAS TAURINENSIS

Outline

- Stochastic Formal System Biology
- From Distributed to Multicore and back
- On programming models
  - FastFlow
- The CWC parallel simulator for sys bio
- Some preliminary results

Formal synthetic system biology





?

System Biology & Gillespie's algorithm

- Traditionally studied with continuous ordinary differential equations (ODE)
  - bulk reactions, i.e. average behaviour
- Gillespie algorithm: discrete and stochastic simulation of a system via explicit simulation of each reaction
- Gillespie realization represents a random walk that exactly represents the distribution of the master equation (i.e. ODEs)
  - under some hypothesis

Gillespie's algorithm [77]

1. **Initialization**: Initialize the number of molecules in the system, reactions constants, and random number generators.
2. **Monte Carlo step**: Generate random numbers to determine the next reaction to occur as well as the time interval. The probability of a given reaction to be chosen is proportional to the number of substrate molecules.
3. **Update**: Increase the time step by the randomly generated time in Step 2. Update the molecule count based on the reaction that occurred.
4. **Iterate**: goto Step 2 unless the number of reactants is zero or the simulation time has been exceeded.

Increasingly popular approach

- Sometime more informative than (ODE)
  - multi-stability, divergent or rare behaviours, peaks, ...
  - multi-scale systems
    - e.g. deriving macro behaviour from micro
- More computational demanding
  - much more, especially in motivating cases

Increasingly popular approach

- Bio-PEPA [Hillston, Ciocchetta]
- SPiM [Cardelli, Phillips]
- Stochastic Pi [Priami]
- Stochkit [Petzold]
- Spatial Pi [Uhrmacher]
- Calculus of Wrapped Components [our own]
  - kinetics: mass-action, Michaelis–Menten, Hill ...
- ...

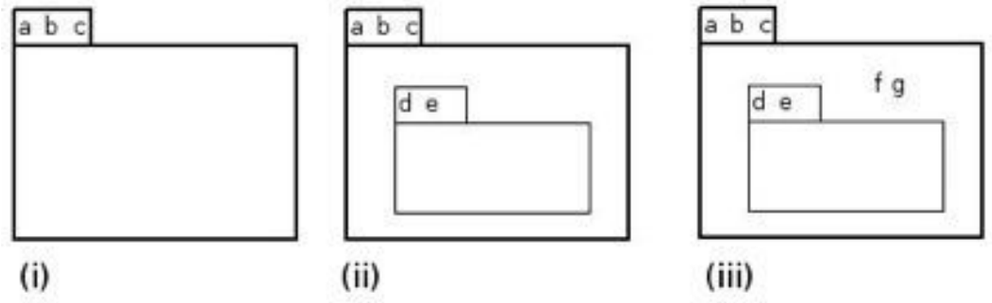
The Calculus of Wrapped Compartments (CWC)

A **term** is intended to represent a biological system. A *term* is built by means of the **compartment** constructor, $(- \rfloor -)$, from a set $\mathcal{E}$ of *atomic elements*, ranged over by a, b, c, d. A **simple term** is defined as:

$$t \quad ::= \quad a \;\Big|\; (\overline{a} \rfloor \overline{t})$$

We write $\overline{t}$ to denote a (possibly empty) multiset of simple terms $t_1 \ldots t_n$. Similarly, with $\overline{a}$ we denote a (possibly empty) multiset of atoms.

Examples of SCWC terms



- (i) represents $(a\ b\ c \rfloor \bullet)$;
- (ii) represents $(a\ b\ c \rfloor (d\ e \rfloor \bullet))$;
- (iii) represents $(a\ b\ c \rfloor (d\ e \rfloor \bullet)\ f\ g)$.

Dynamics of SCWC

Rewrite rules are defined as pairs of terms, in which the left term characterizes the portion of the system in which the event modelled by the rule can occur, and the right one describes how that portion of the system is changed by the event.

Biomolecular Event	**Examples of CWC Rewrite Rules**
State change	$a \mapsto b$
Complexation	$a\ b \mapsto c$
Catalyzed membrane crossing	$a\ (b\ x \rfloor y) \mapsto (b\ x \rfloor a\ y)$ $(b\ x \rfloor a\ y) \mapsto a\ (b\ x \rfloor y)$

Stochastic Rules

Rules are decorated with a **rate** (speed of the reaction).

A **Stochastic Rewrite Rule**, R, is denoted by $P \overset{k}{\mapsto} P'$.

The stochastic semantics is given by transitions between terms labeled with the rule applied, R, and a transition rate depending on the rate of rule R:

$$\overline{t} \overset{R, k \times p}{\longrightarrow} \overline{t'}$$

where R is $P \overset{k}{\mapsto} P'$, and p is the number of different ways in which the pattern P may match $\overline{t}$ ($\overline{t} = C[P\sigma]$) and such that $\overline{t'} = C[P'\sigma]$ for some context C and variable instantiation σ.

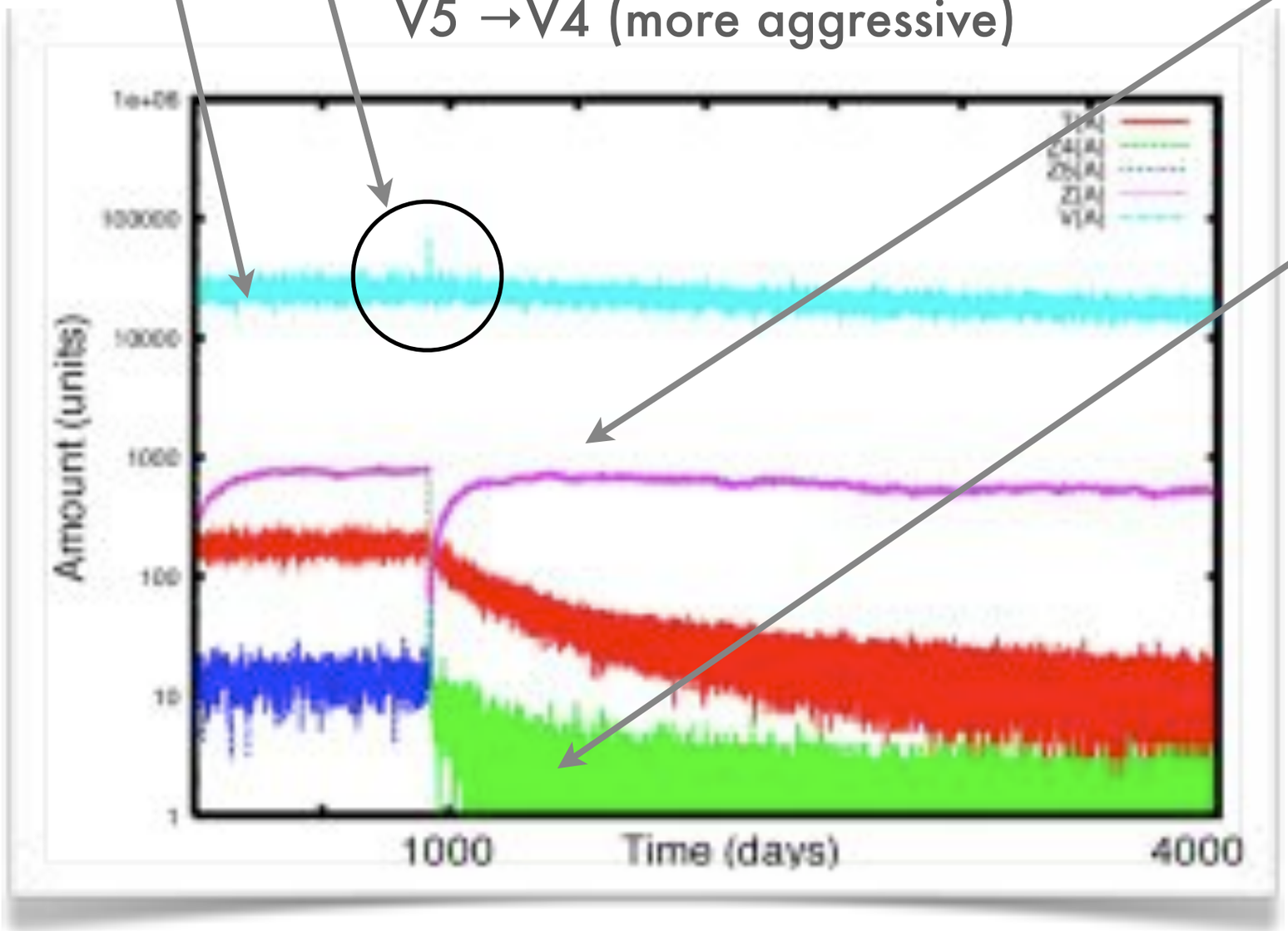
Ex: HIV and immune response (progression to AIDS)

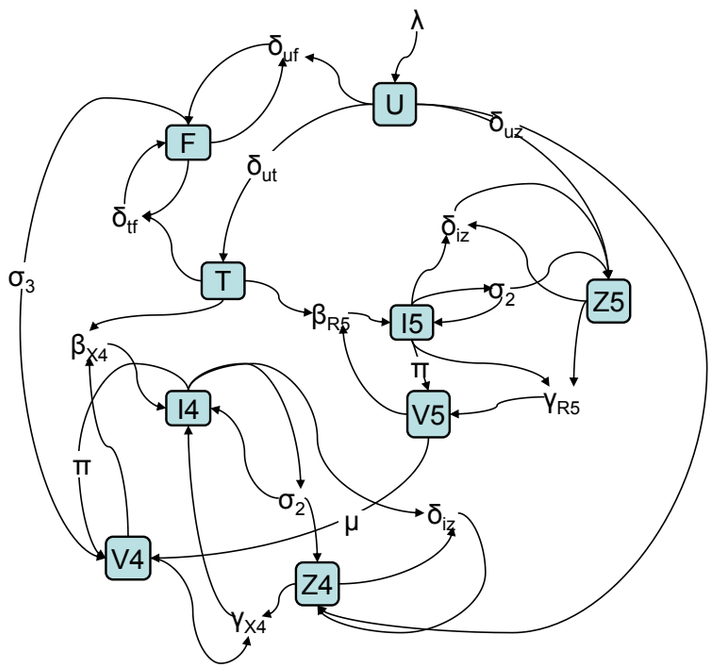
V virus, all phenotypes (V4,V5)

the spike suggests the mutation
V5 →V4 (more aggressive)

immune response Z=Z4+Z5
remain stable (but for the peak).

now Z4 decrease and
Z5 increase
i.e. HIV is turning into
AIDS more rapidly



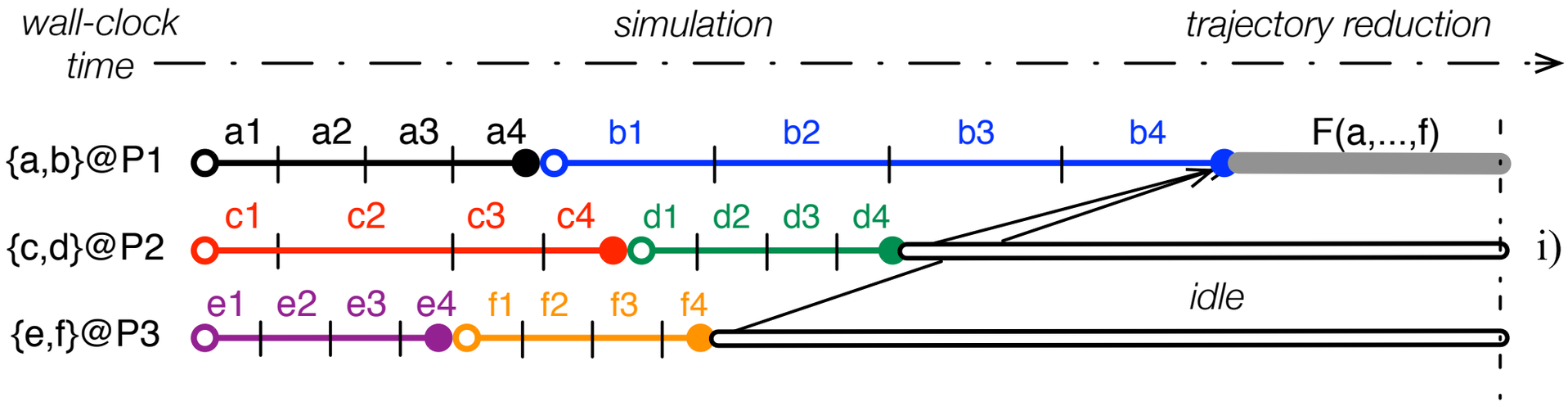


M. Aldinucci, A. Bracciali, P. Liò, A. Sorathiya, and M. Torquati. StochKit-FF: Efficient systems biology on multicore architectures. In High Performance Bioinformatics and Biomedicine (HiBB), vol. 6586 of LNCS, 2011. Springer.

- Peaks are informative events,
  - virus mutation triggers AIDS progression
  - hardly detected with ODEs
- high resolution required to detect spikes,
  - each trajectory can be over 6G Bytes of data
- and thousands of trajectories are needed
  - compute everything, save everything, move and join all data, analyse all data, then get first results
  - often to discover parameters are wrong ...

- It is Monte Carlo,
  - well understood
  - easy to parallelise on different trajectories
- it is Monte Carlo w Markov Chains models (CTMC)
  - single trajectory: no parallelisation without relaxation
  - compute time ≠ simulation time
  - compute time for different trajectories heavily unbalanced
    - fast reactions and slow reactions, some not interesting (e.g. water-steam-water)

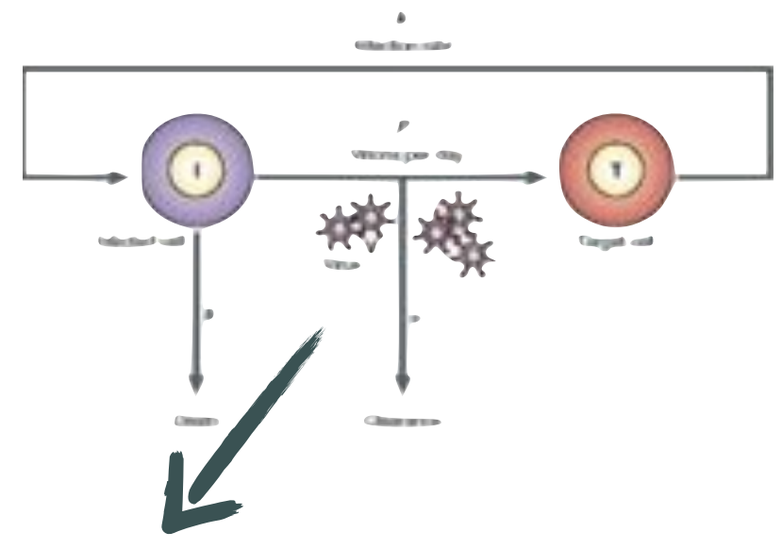
Unbalancing + filtering

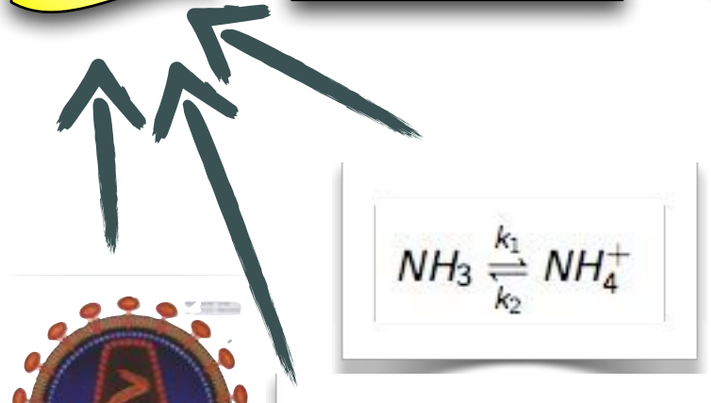


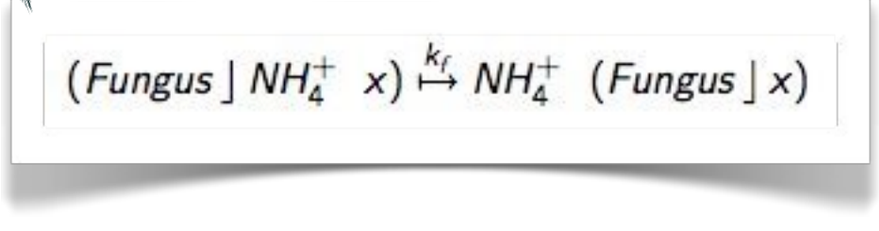
- few trajectories (e.g. the interesting ones) can significantly delay the completion of others
  - over-provisioning don't help that much
  - simulated time moves at different pace w.r.t. wall-clock time
  - data joining from different trajectories should be aligned at the same simulation time

- It is Monte Carlo,
  - well understood
  - easy to parallelise on different trajectories
- it is Monte Carlo w Markov Chains models (CTMC)
  - single trajectory: no parallelisation without relaxation
  - compute time ≠ simulation time
  - compute time for different trajectories heavily unbalanced
    - fast reactions and slow reactions, some not interesting (e.g. water-steam-water)
- it is Monte Carlo AND data analysis
  - data is big, analysis can be very expensive and it typically starts after the simulation
  - the whole workflow is perceived too “slow” by bio-scientists to be really useful

From Distributed to Multicore and back







Model description

Pre-processing e.g. parsing

Scheduler *Parameter Sweep*

one or more simulation instance

one or more simulation instance

Worker 1 *run simulations*

Disk 1

Rebalance

Worker n *run simulations*

Disk n

Collector *Traiectories*

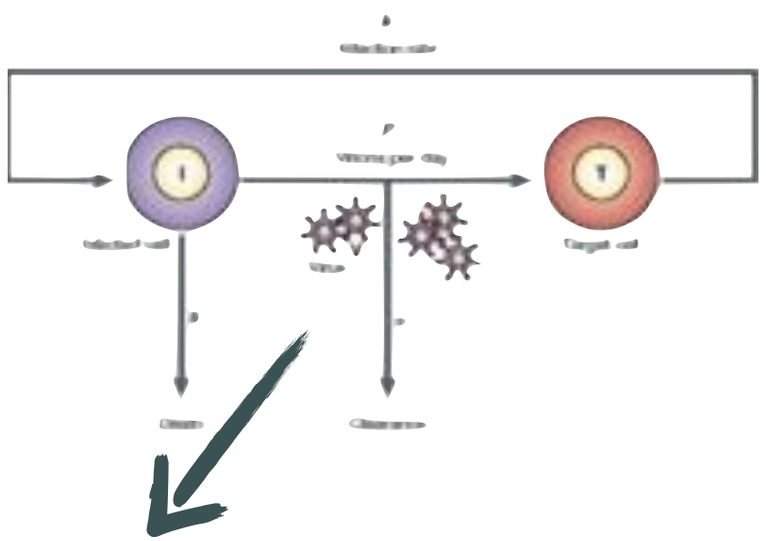
Post-processing e.g. parsing

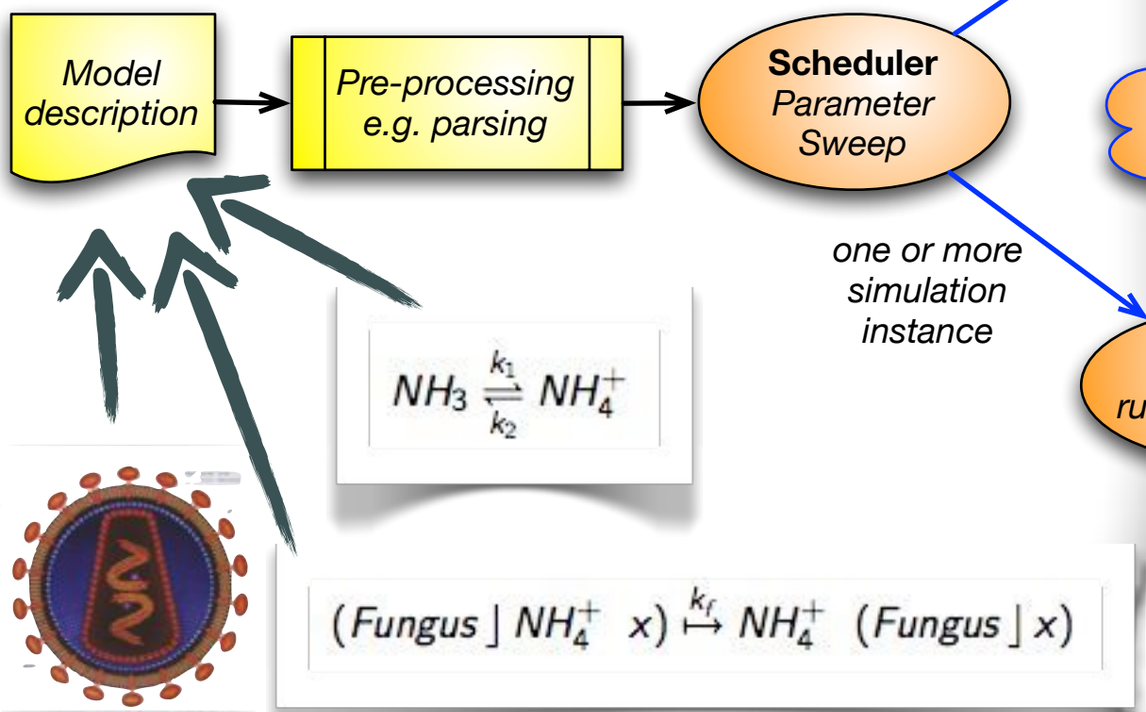
Disk

In BioSims the whole trajectory is needed, easily GB of data

The whole dataset, TB of data, should be re-read to extract statistical estimators

From Distributed to Multicore and back





Now the issue become a real problem

Bottlenecks: disk and memory

Post-processing: not pipelined and often not parallel at all

From Distributed to Multicore and back

- Multi Carlo sims for Bio are I/O-bound
  - Sampling reduce I/O traffic but worsen precision and analysis of "strange" dynamics (spikes, diversion from average, etc.), which observation motivates stochastic analysis (ODEs)
- Data analysis is also I/O-bound
  - if approached is a "post-processing" fashion, data should be retrieved from the disks
- The porting of distributed solution "as is" on multicore is going insist on weakness of multicore architectures
  - Memory wall, I/O, disk
  - SIMD/GPGPUs do not change the analysis substantially

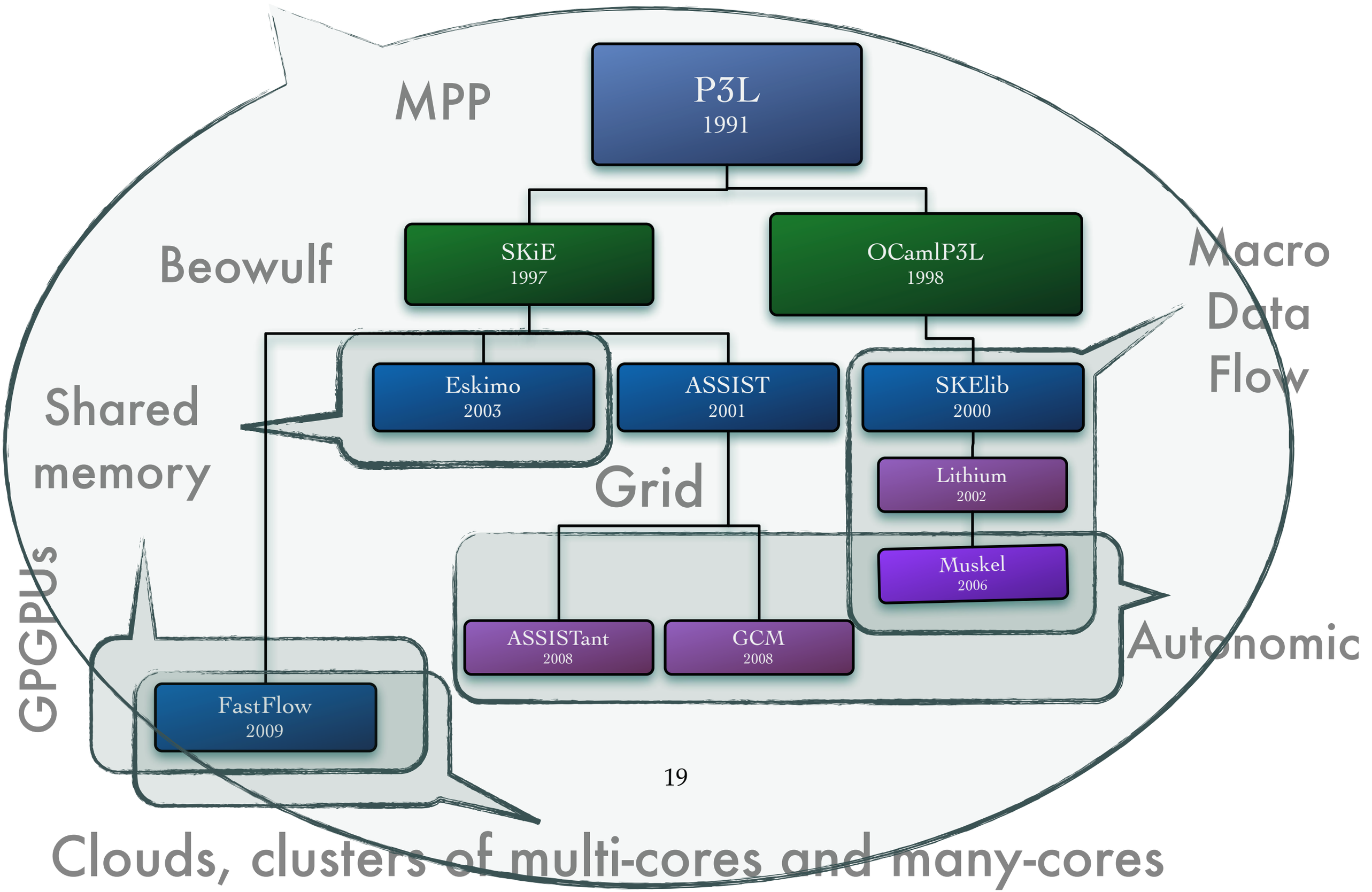
From Distributed to Multicore and back

- The same arguments holds on distributed, grids, and clouds as soon as the workflow is considered as a whole
  - simulation, data collection and merging, analysis
- Rationale
  - **Manage data as stream, compute online**
    - May require more computing and less bandwidth
    - Computation should be designed to be pipelined
  - **Establish fast data paths across cores/hosts**
  - **Avoid low-level concurrency management**
    - Portability, performance, portability of performance, maintenance, porting from sequential

Quotes from P. Beckman EuroPar 11 "exascale" keynote

- *coarse grain concurrency is nearly exhausted*
- *it is not about Flops, it is about data movement*
- *programming systems should be designed to support fast data movement and enforce locality*
- *shared-memory & inter-socket messaging*
- *we need a programming model*
  - *a computer language is not a computing model*
    a library is not a computing model
- *we need a efficient and compositional run-time*

Patterns/skeletons & streams



Clouds, clusters of multi-cores and many-cores

it is not about Flops, it is about data movement, we need compositional run-time

- Streams
  - focus on data movements at the prog model level
  - clear semantics
  - support compositionally and also locality
    - the latter is a bit more counter-intuitive
- High-level programming
  - e.g. patterns
- Patterns + streams
  - can be implemented efficiently on both multi-core, distributed, and both

FastFlow

http://mc-fastflow.sourceforge.net/

FastFlow (multicore)

Applications on multicore, many-core
Efficient and portable - designed with high-level patterns

FastFlow

Streaming network patterns
Skeletons: pipeline, map farm, reduce, D&C, ...

Arbitrary streaming networks
Lock-free SPSC/MPMC queues + FF nodes

Simple streaming networks
Lock-free SPSC queues + threading model

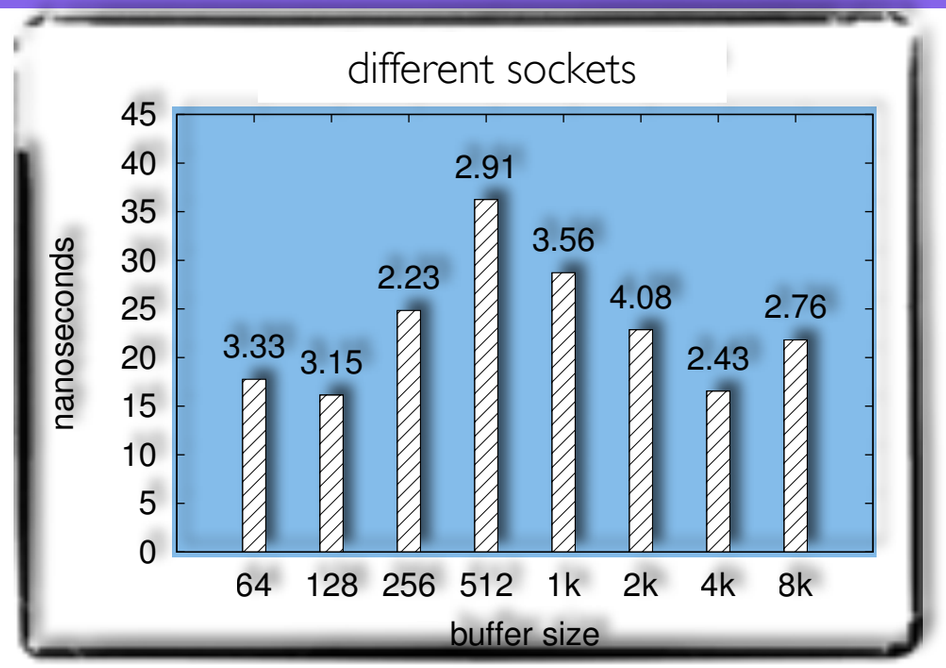
Multicore and manycore
SMP: cc-UMA & cc-NUMA

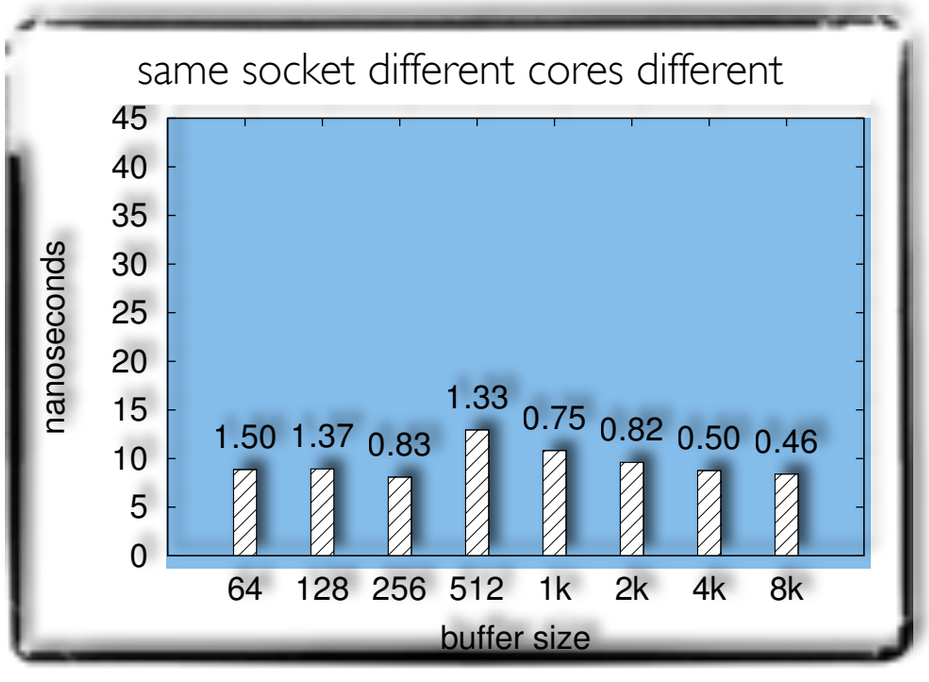
Layer 1: Simple streaming networks

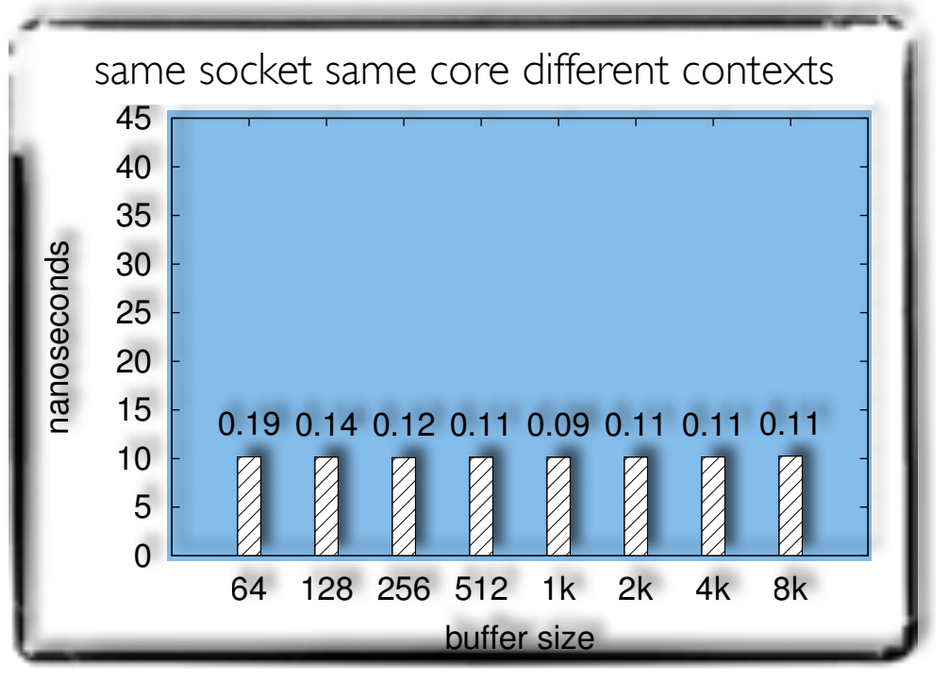
4 sockets x 8 core x 2 contexts

Xeon E7-4820 @2.0GHz Sandy Bridge
18MB L3 shared cache, 256K L2

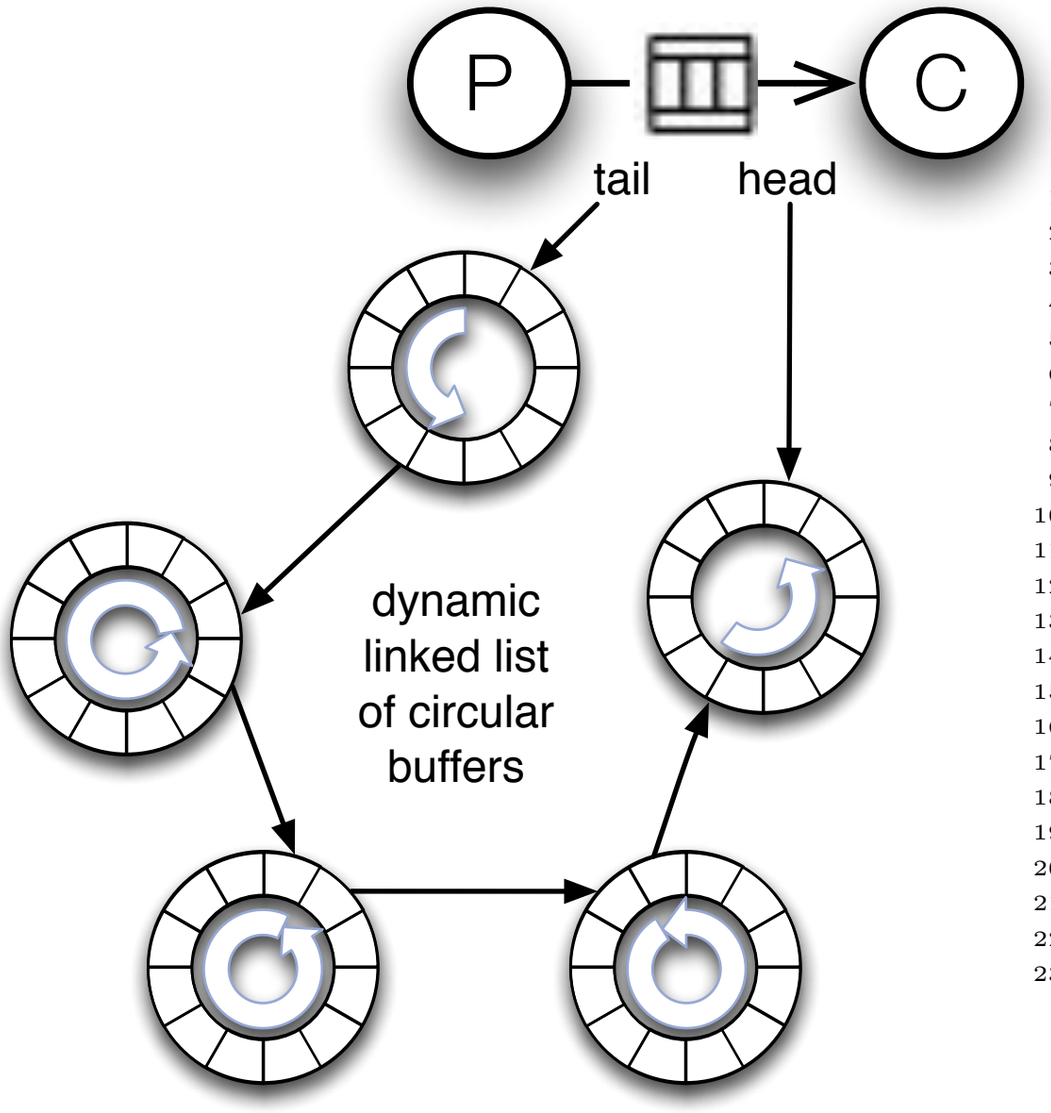
MPI is ~190 ns at best
(D.K. Panda)







Layer 1: Simple streaming networks




```
1  int  size  = N; //SPSC size
2  bool push(void* data) {
3    if (buf_w->full()) {
4      SPSC* t = pool.next_w();
5      if (!t) return false;
6      buf_w = t;
7    }
8    buf_w->push(data);
9    return true;
10 }
11 bool pop(void** data) {
12   if (buf_r->empty()) {
13     if (buf_r == buf_w) return false;
14     if (buf_r->empty()) {
15       SPSC* tmp = pool.next_r();
16       if (tmp) {
17         pool.release(buf_r);
18         buf_r = tmp;
19       }
20     }
21   }
22   return buf_r->pop(data);
23 }
```

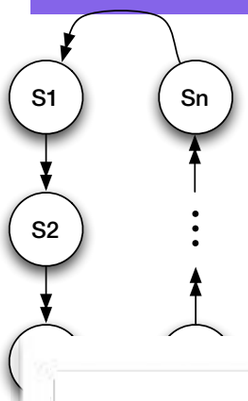
```
25 struct Pool {
26   dSPSC inuse;
27   SPSC cache;

29   SPSC* next_w() {
30     SPSC* buf;
31     if (!cache.pop(&buf))
32       buf = allocateSPSC(size);
33     inuse.push(buf);
34     return buf;
35   }
36   SPSC* next_r() {
37     SPSC* buf;
38     return (inuse.pop(&buf)? buf : NULL);
39   }
40   void release(SPSC* buf) {
41     buf->reset(); // reset pread and pwrite
42     if (!cache.push(buf))
43       deallocateSPSC(buf);
44   }
45 }
```

Fig. 3: Unbounded wait-free uSPSC queue implementation.

M. Aldinucci, S. Campa, M. Danelutto, M. Torquati. An Efficient Synchronisation Mechanism for Multi-Core Systems.
EuroPar 2012.
Wed 29 Aug - B3 multicore 14.30-16.00

Layer 1: Simple streaming networks

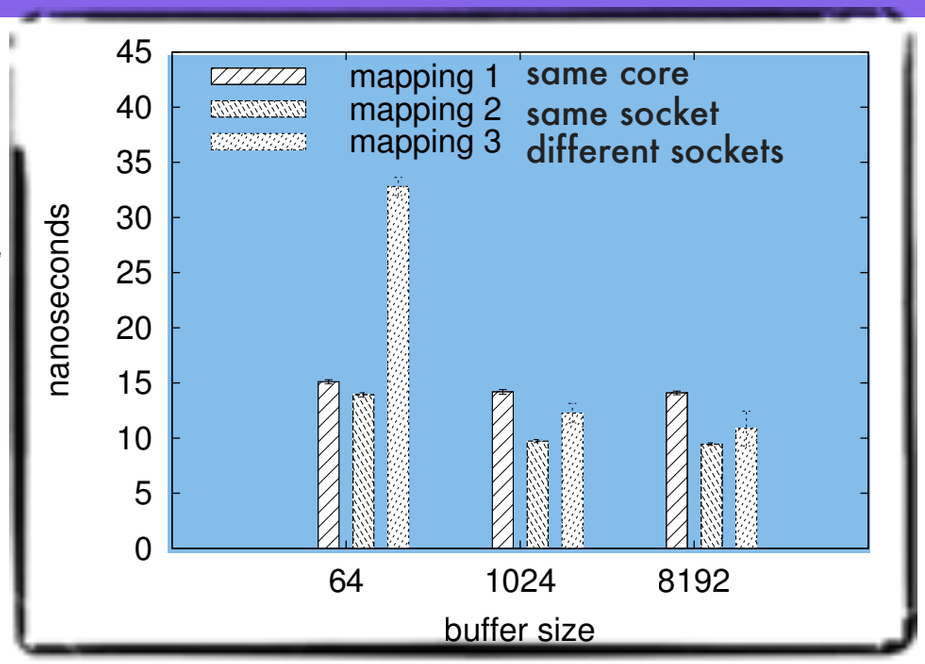
http://www.1024cores.net/home/technologies/fastflow



4 sockets x 8 core x 2 contexts

Xeon E7-4820 @2.0GHz Sandy Bridge
18MB L3 shared cache, 256K L2

<35 ns irrespectively of the mapping



Linked list + circular buffer
FastFlow queue
(our result)

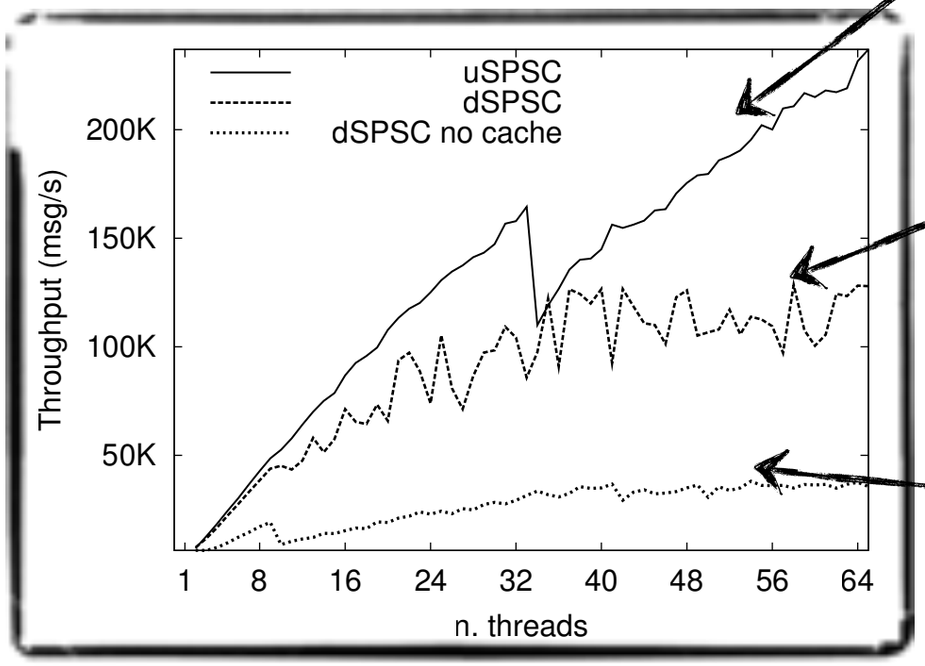
12x faster

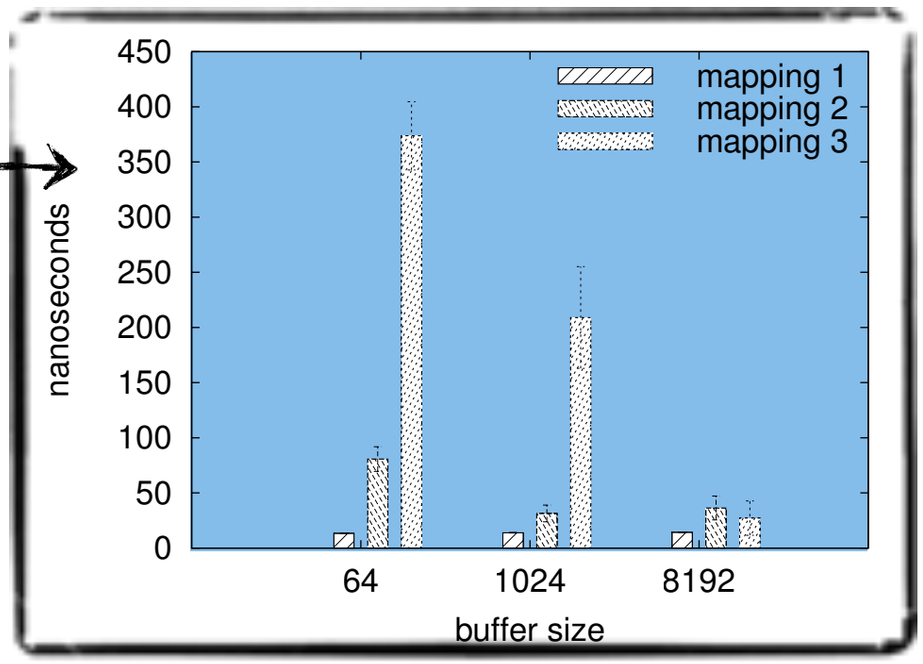
Linked list with pooling
Opt Michael-Scott queue
(our result)

20x faster

Linked list w/ dyn alloc
Michael-Scott queue
well-known ~ 400 citations

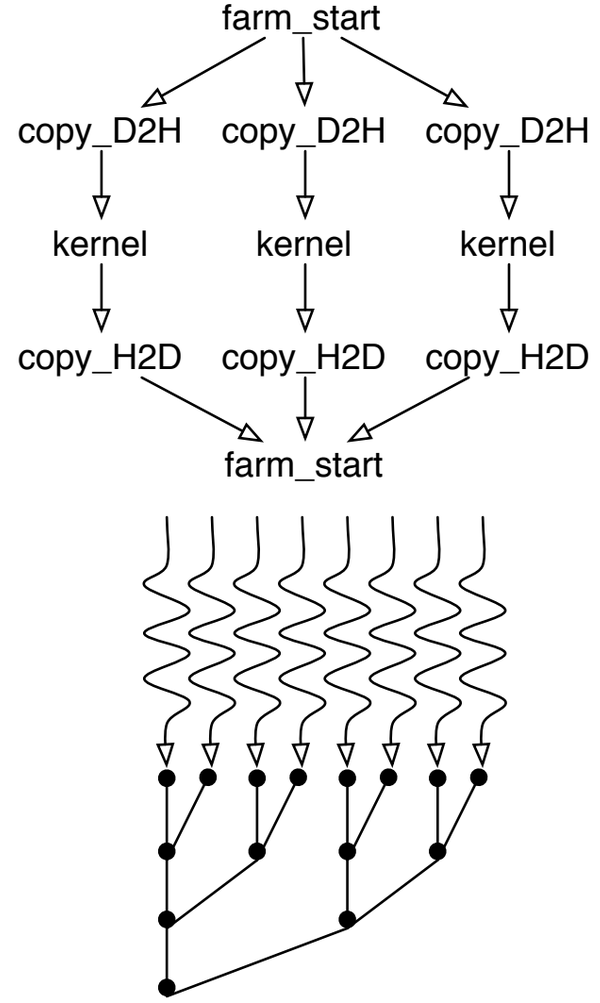
Speedup



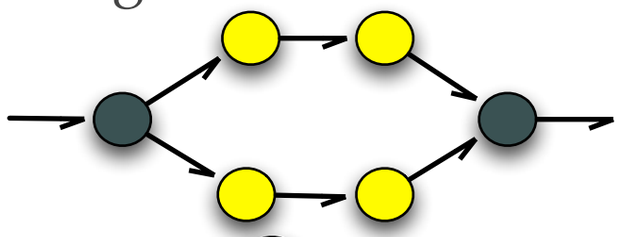


Layer 3: streaming networks patterns

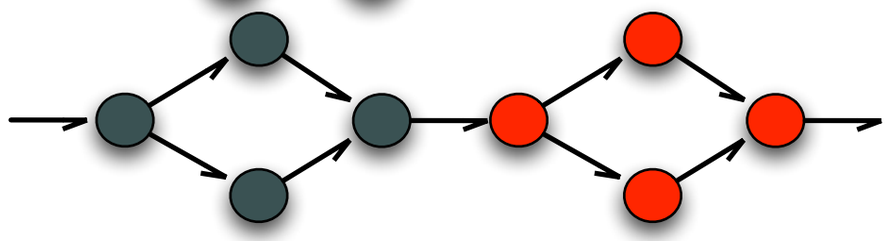
- Composition via C++ template meta-programming
  - CPU: Graph composition
  - GPU: CUDA streams
  - CPU+GPU: offloading



- farm{ pipe }



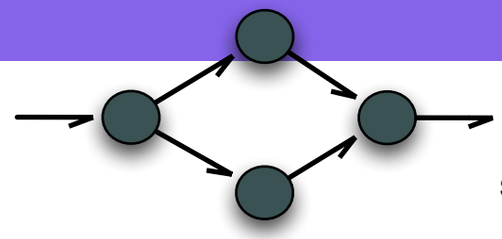
- pipe(farm, farm)

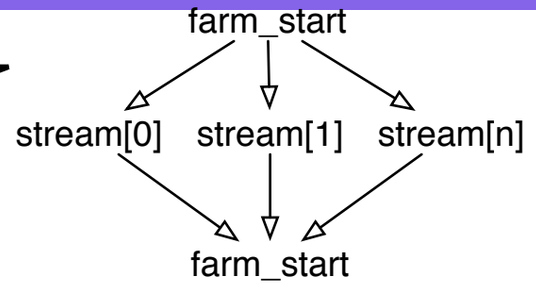


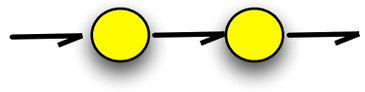
- pipe(map, reduce)
- ....

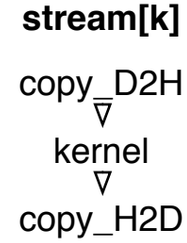
Layer 3: streaming networks patterns

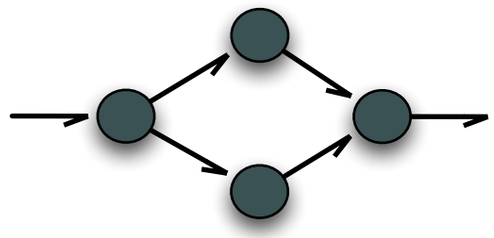
- farm
  - on CPU - master-worker - parallelism exploitation
  - on GPU - CUDA streams - automatic exploitation of asynch comm
- pipeline
  - on CPU - pipeline
  - on GPU - sequence of kernel calls or global mem synch
- map
  - on CPU - master-worker - parallelism exploitation
  - on GPU - CUDA SIMT - parallelism exploitation
- reduce
  - on CPU - master-worker - parallelism exploitation
  - on GPU - CUDA SIMT (reduction tree) - parallelism exploitation
- D&C
  - on CPU - master-worker with feedback - // exploitation
  - on GPU - working on it, maybe loop+farm

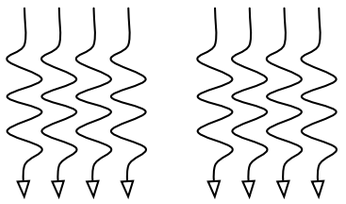


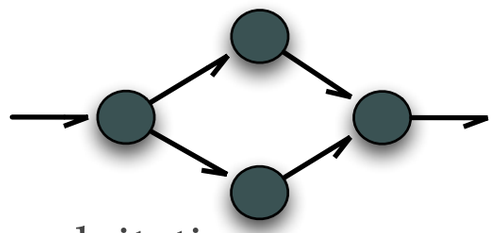


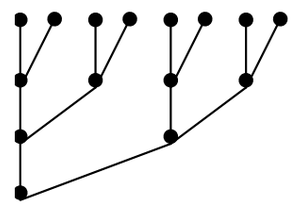


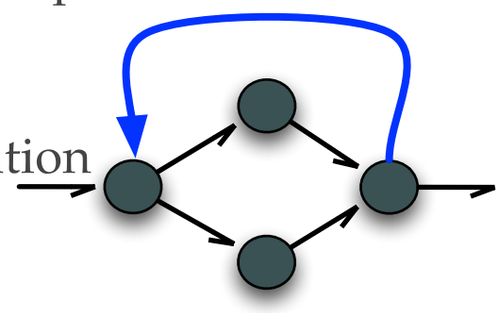






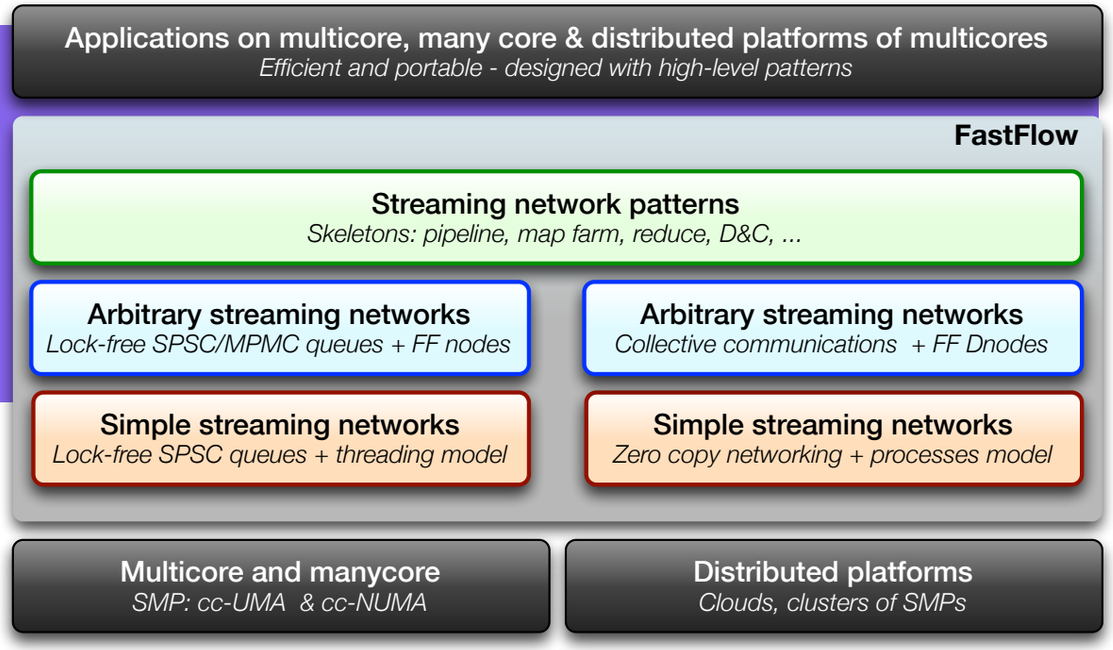






Layer 3: streaming networks patterns (easy to port)

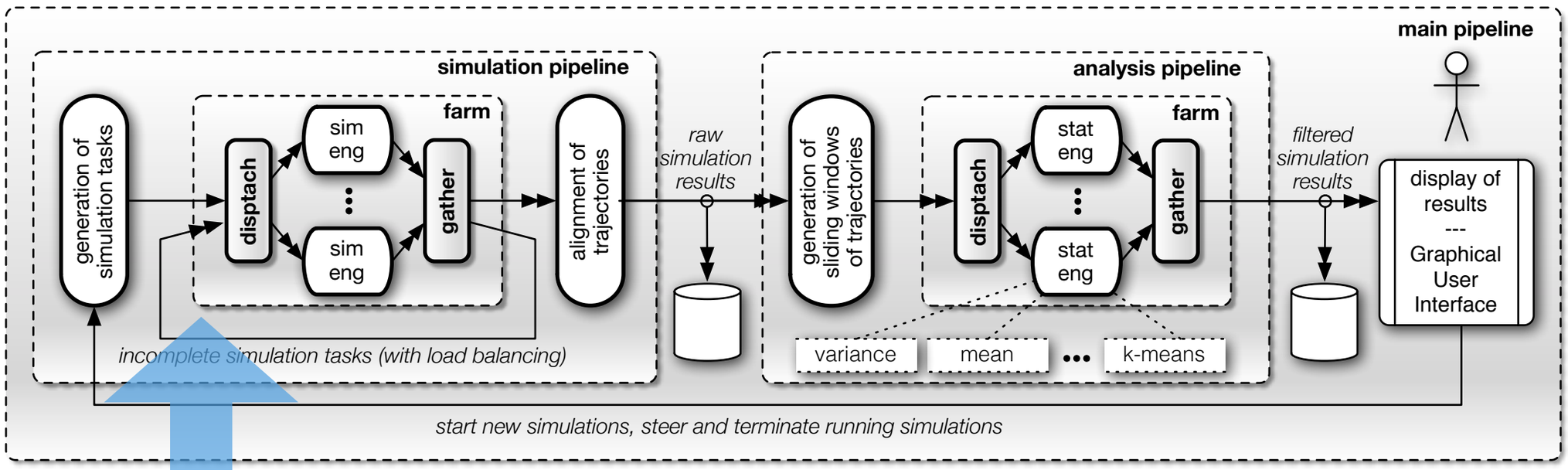
+ distributed

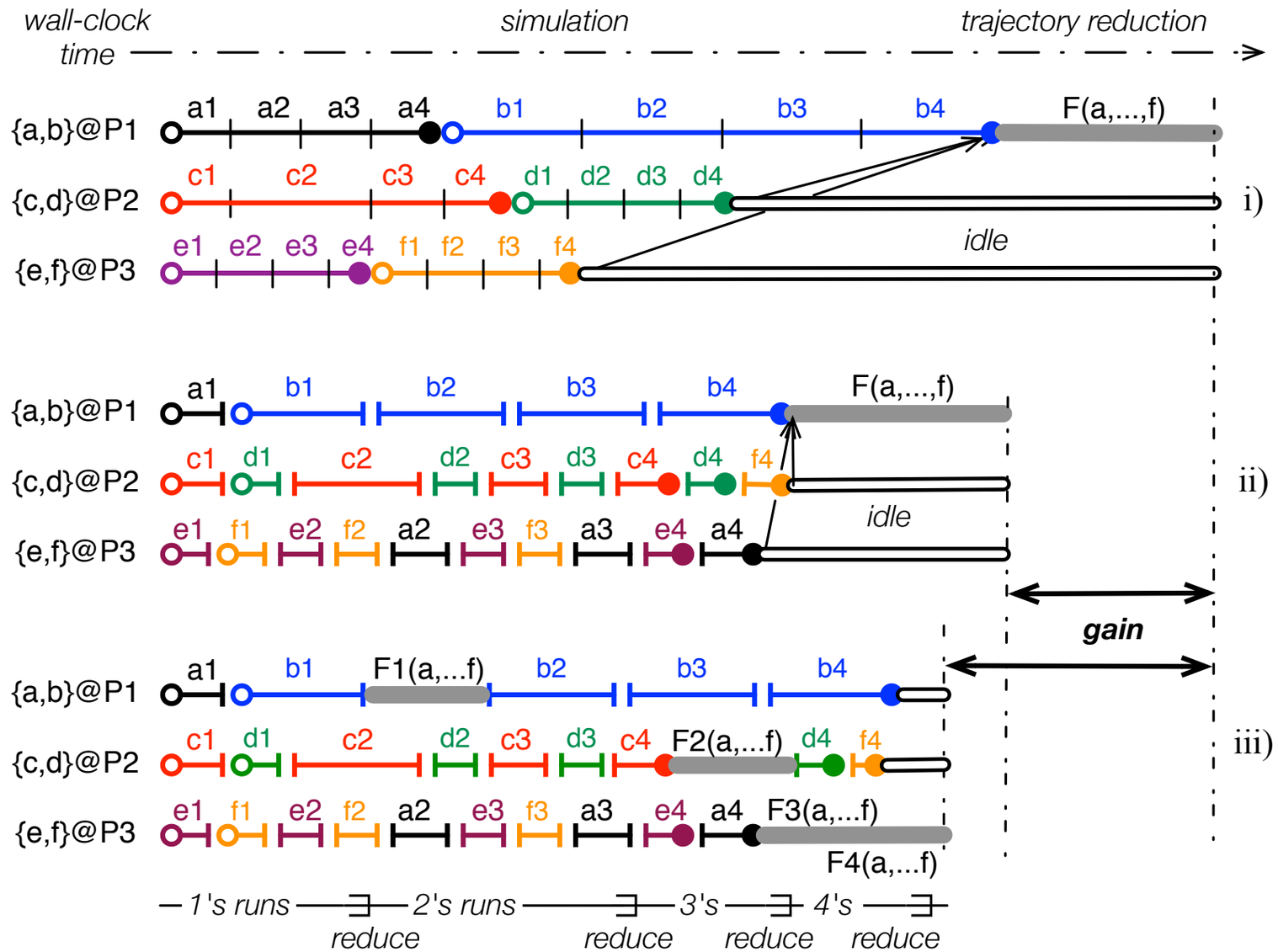


- Generic `ff_node` is subclassed to `ff_dnode`
- `ff_dnode` can support network channels
  - P2P or collective
  - used as frontier node of streaming graph
  - can be used to merge graphs across distributed platforms
- No changes to programming model
  - at least require to "add" stub `ff_dnode`
  - when passing pointers data is serialised
    - serialisation hand-managed (zero-copy, think to Java!)

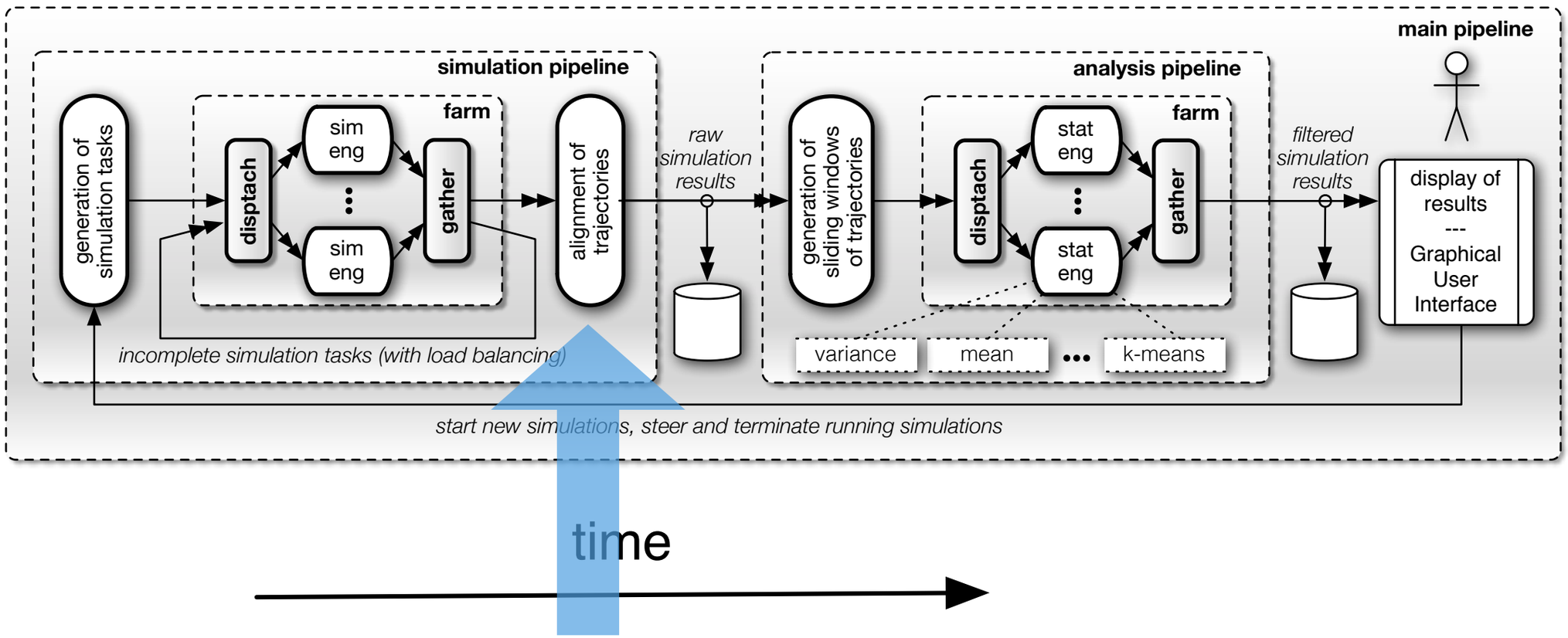
M. Aldinucci, S. Campa, M. Danelutto, M. Torquati, P. Kilpatrick. Targeting distributed systems in FastFlow. Should be next talk!

CWC simulator example

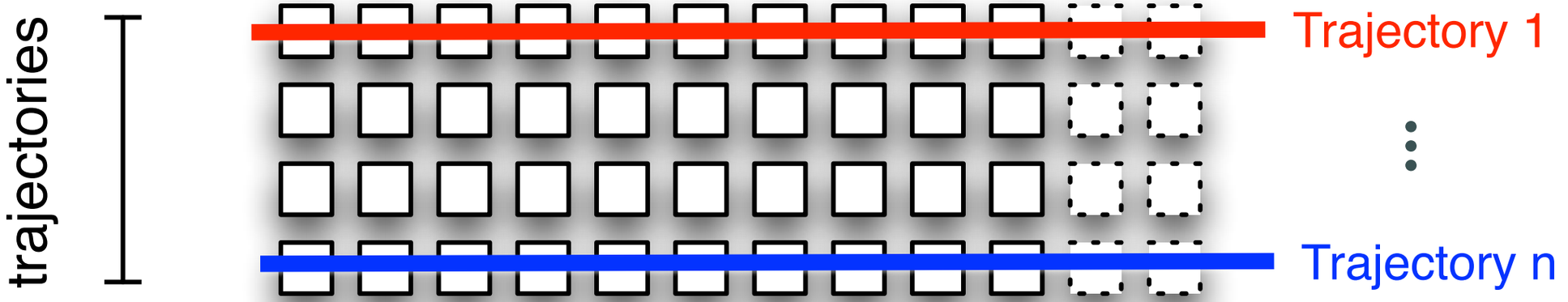


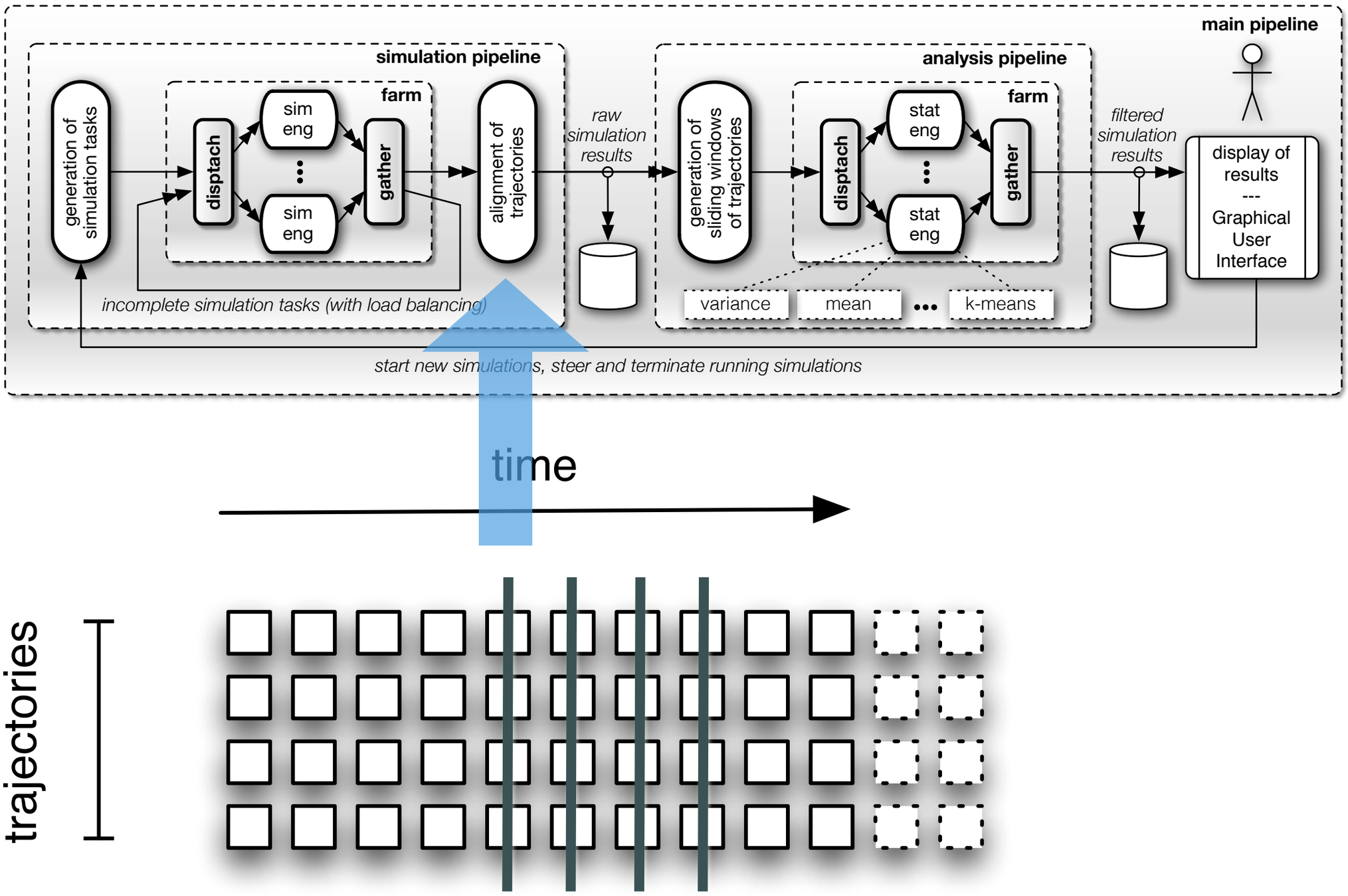


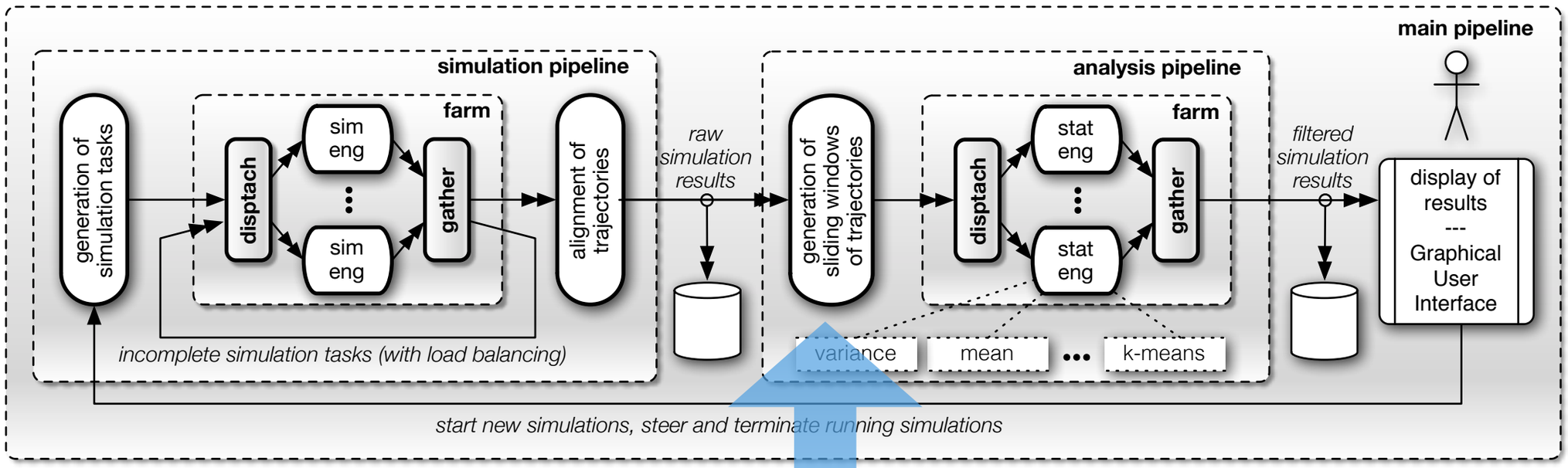
M. Aldinucci, M. Coppo, F. Damiani, M. Drocco, M. Torquati, A. Troina. On designing multicore-aware simulators for biological systems. PDP 2011. 2011. IEEE.

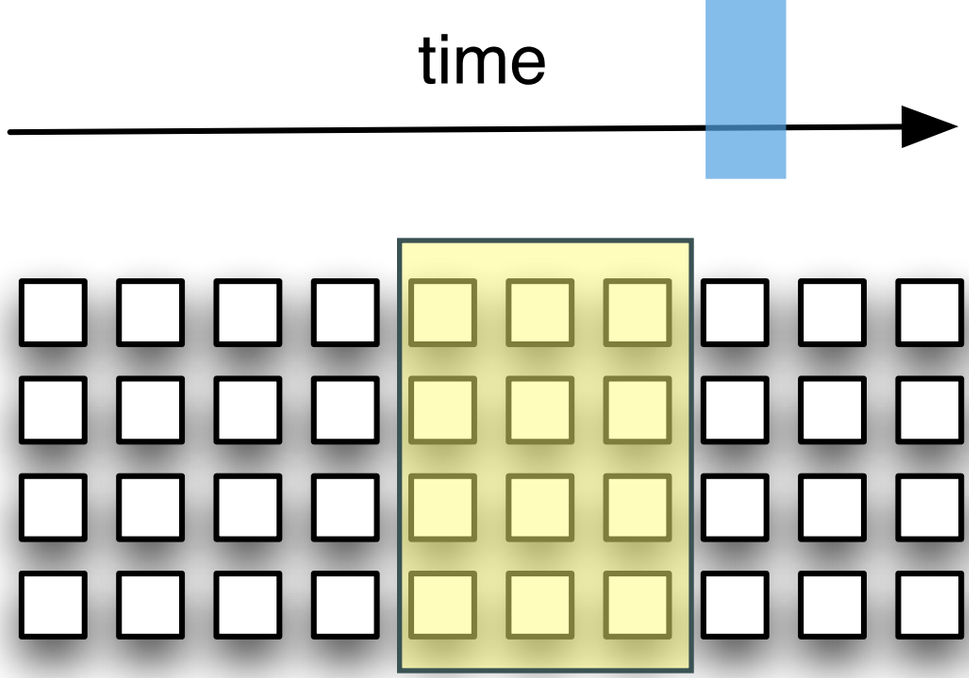

main pipeline
simulation pipeline
analysis pipeline
farm
farm
generation of simulation tasks
disptach
sim eng
sim eng
gather
alignment of trajectories
raw simulation results
generation of sliding windows of trajectories
disptach
stat eng
stat eng
gather
filtered simulation results
display of results

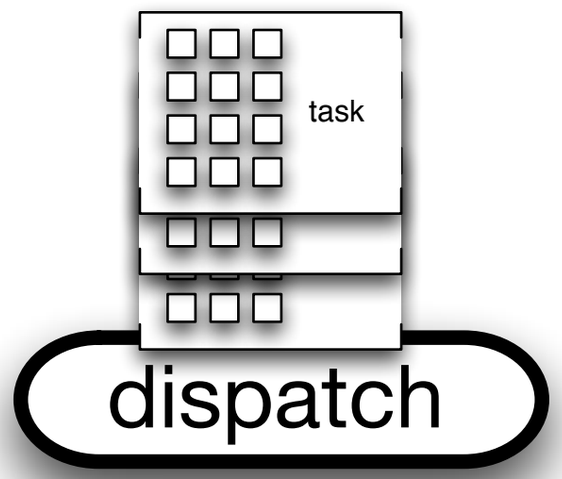
Graphical User Interface
incomplete simulation tasks (with load balancing)
variance
mean
k-means
start new simulations, steer and terminate running simulations
time

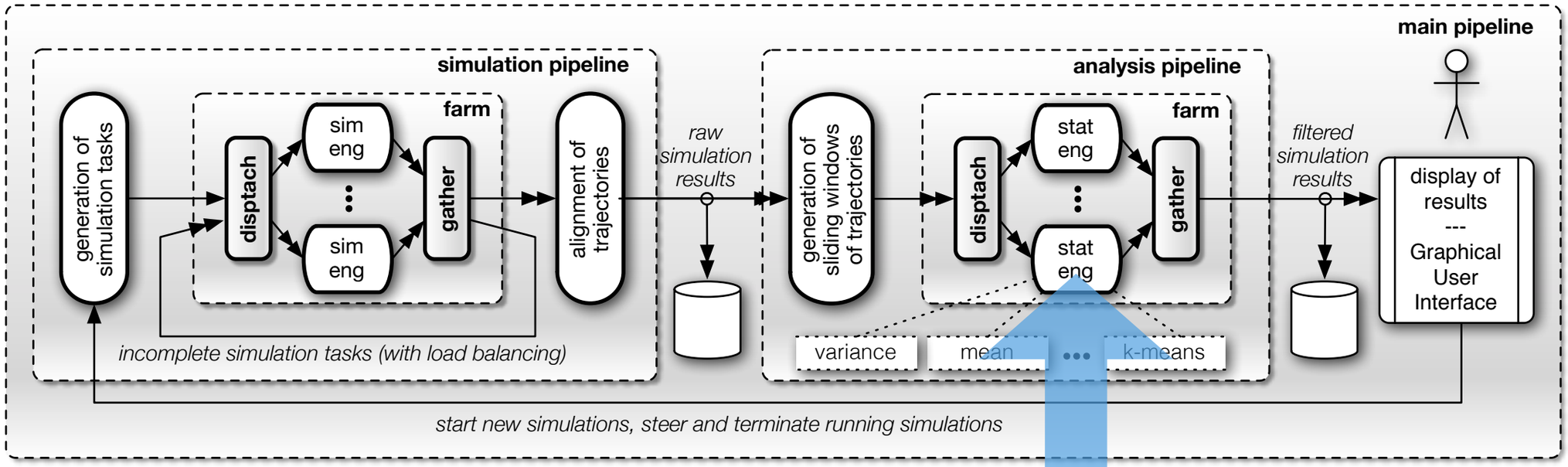

trajectories
Trajectory 1
Trajectory n


main pipeline
simulation pipeline
analysis pipeline
farm
farm
generation of simulation tasks
disptach
sim eng
sim eng
gather
alignment of trajectories
raw simulation results
generation of sliding windows of trajectories
disptach
stat eng
stat eng
gather
filtered simulation results
display of results --- Graphical User Interface
incomplete simulation tasks (with load balancing)
variance
mean
k-means
start new simulations, steer and terminate running simulations
time
trajectories
simulation-time aligned trajectories


main pipeline
simulation pipeline
analysis pipeline
farm
farm
generation of simulation tasks
disptach
sim eng
sim eng
gather
alignment of trajectories
raw simulation results
generation of sliding windows of trajectories
disptach
stat eng
stat eng
gather
filtered simulation results
display of results --- Graphical User Interface
incomplete simulation tasks (with load balancing)
variance
mean
k-means
start new simulations, steer and terminate running simulations


time
trajectories


task
dispatch

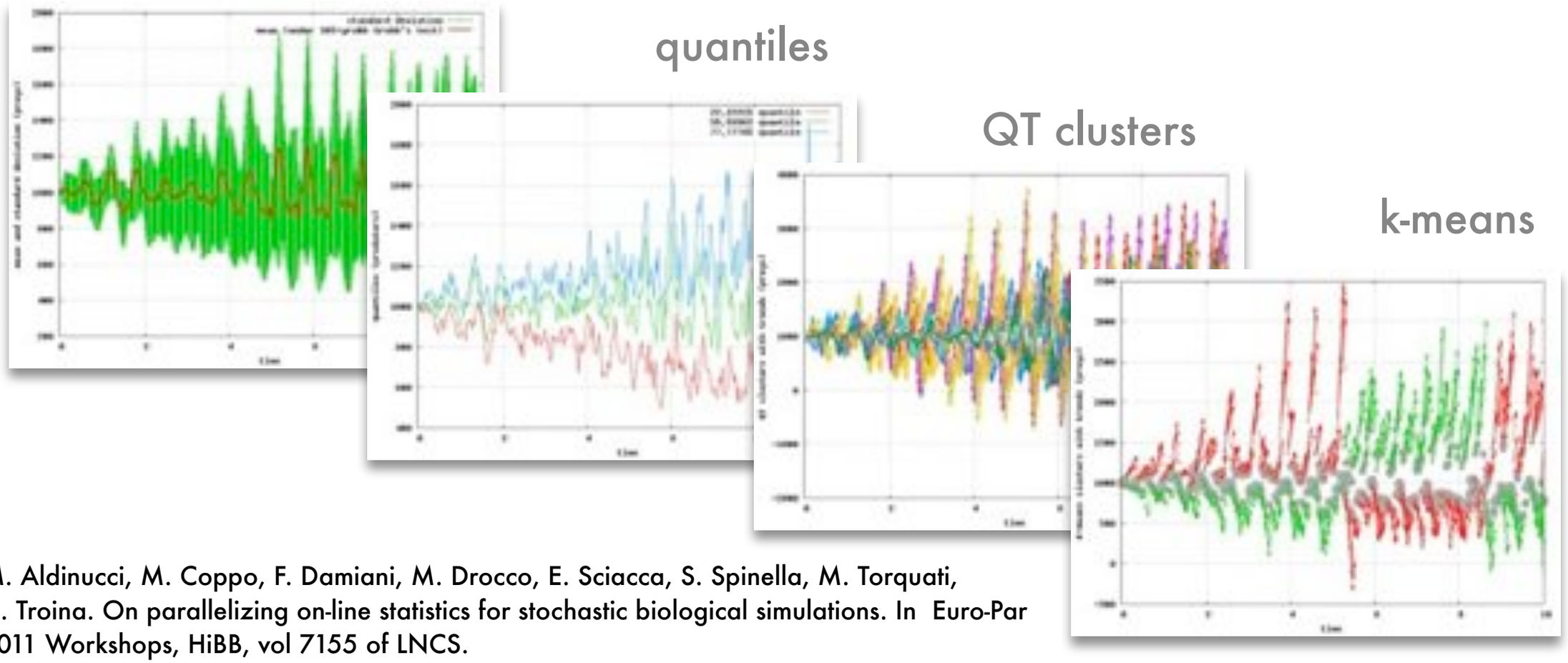


mean variance

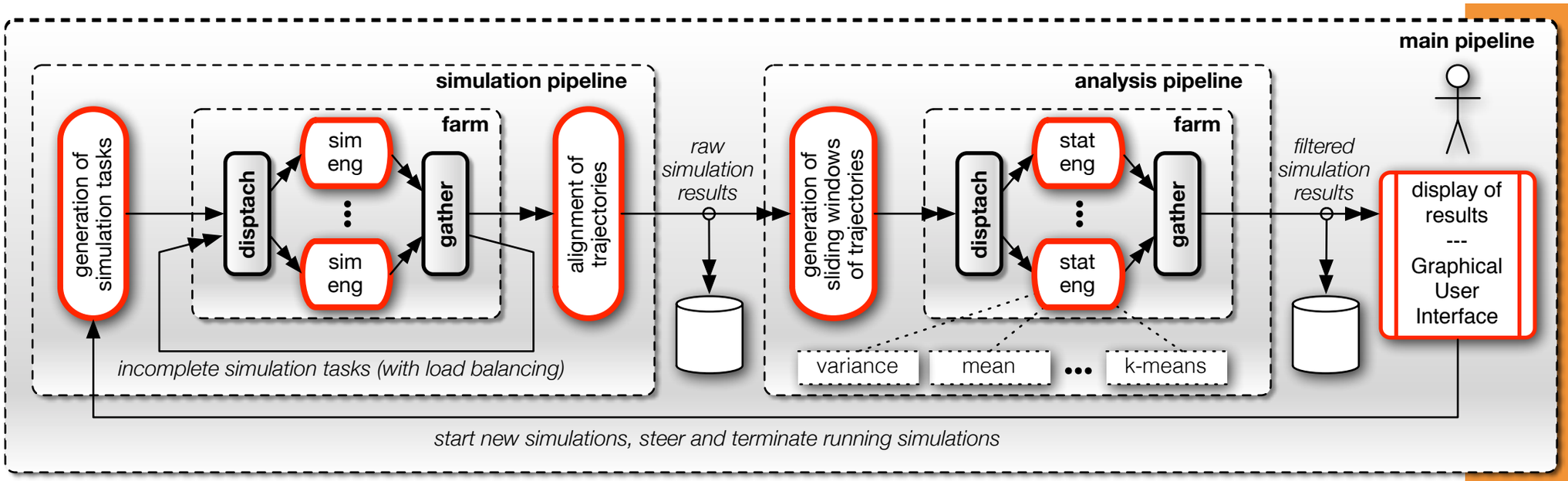
quantiles

QT clusters

k-means

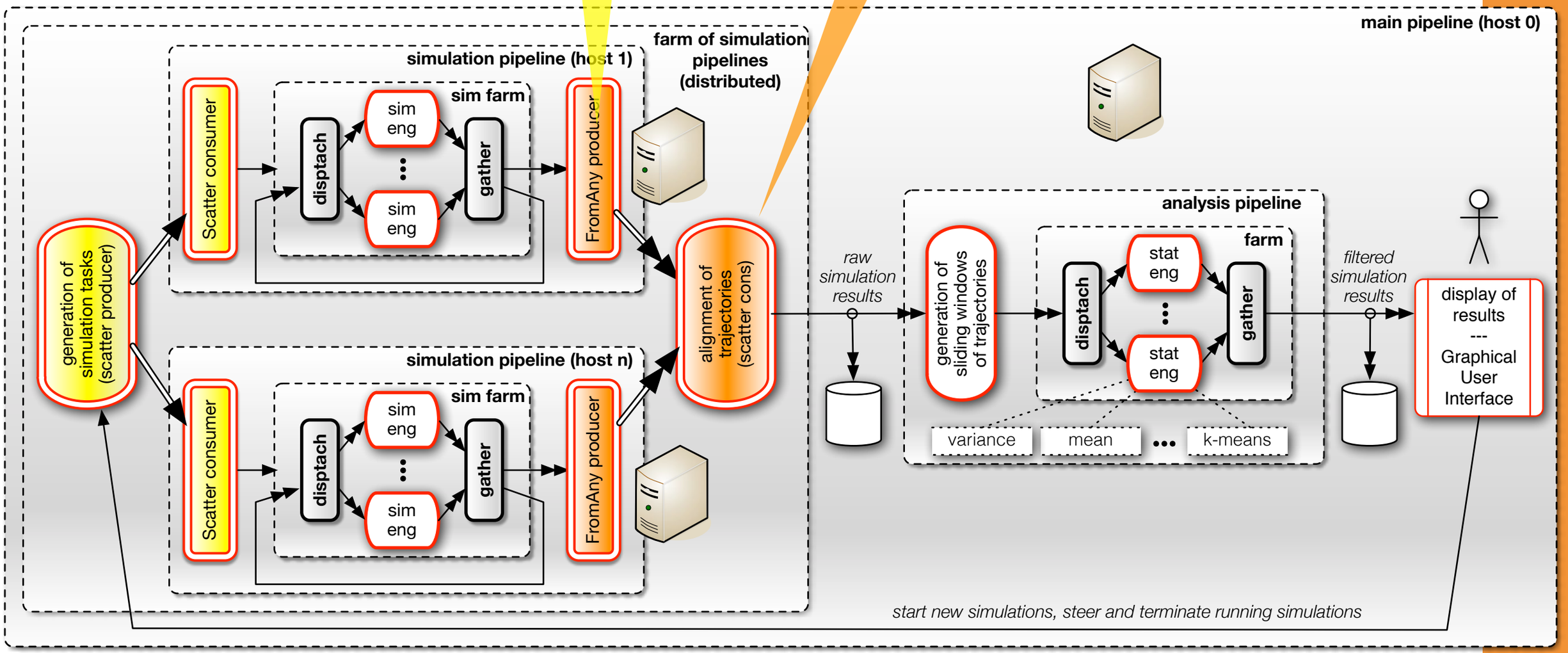


M. Aldinucci, M. Coppo, F. Damiani, M. Drocco, E. Sciacca, S. Spinella, M. Torquati, A. Troina. On parallelizing on-line statistics for stochastic biological simulations. In Euro-Par 2011 Workshops, HiBB, vol 7155 of LNCS.

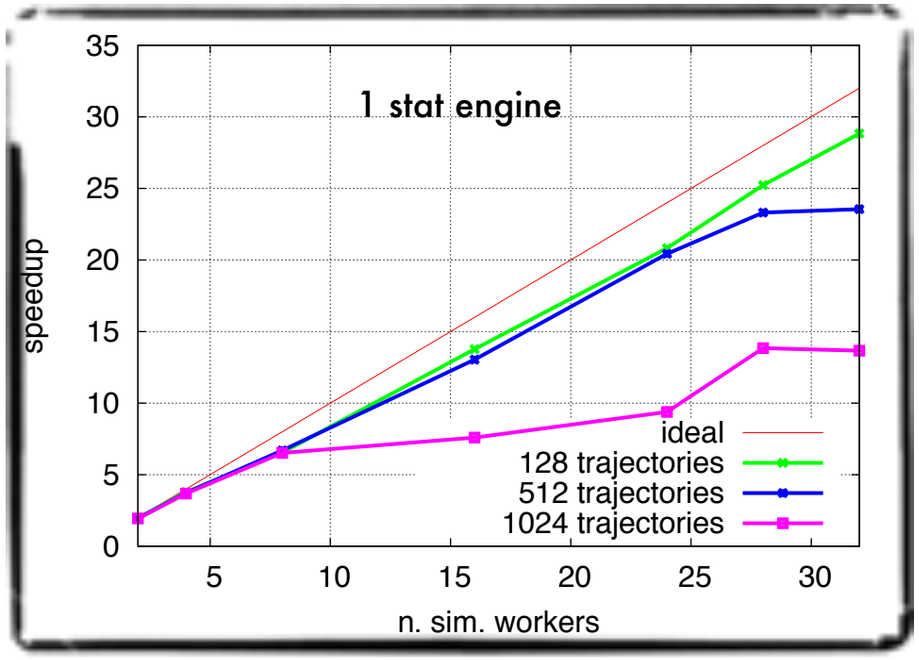


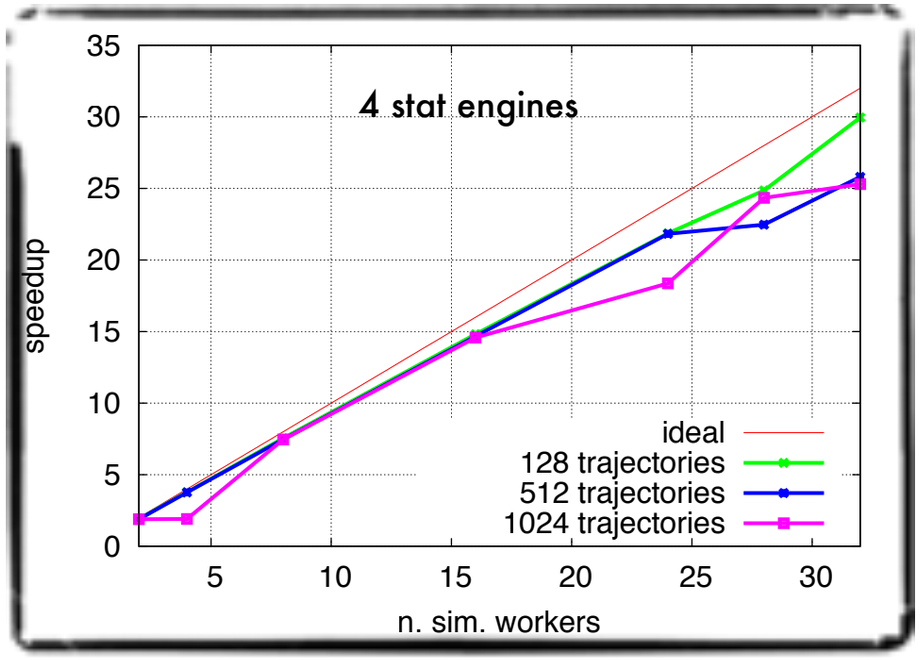
serialisation, coalescing

allocation (zero-copy 0MQ), de-serialisation

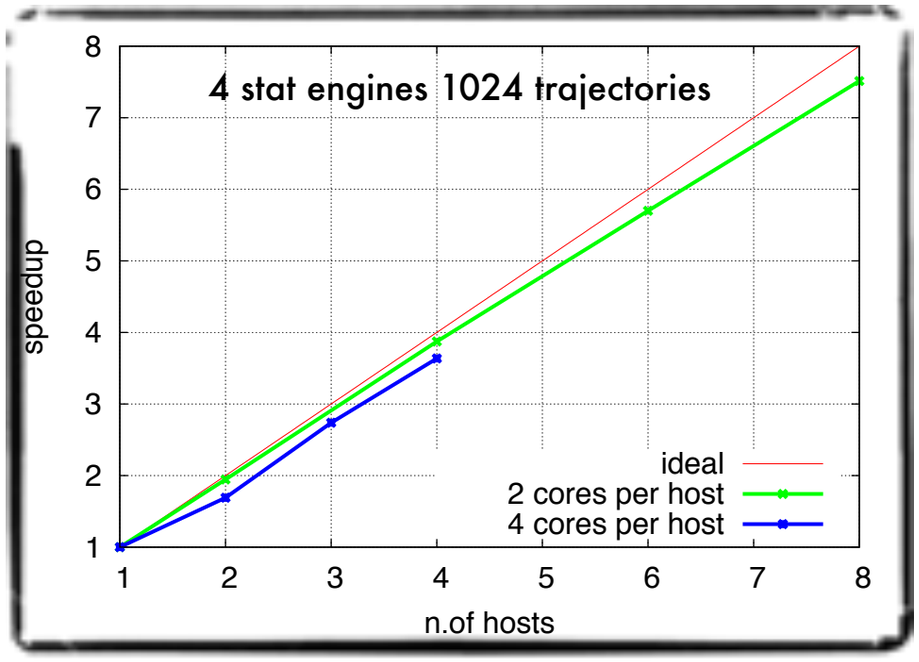


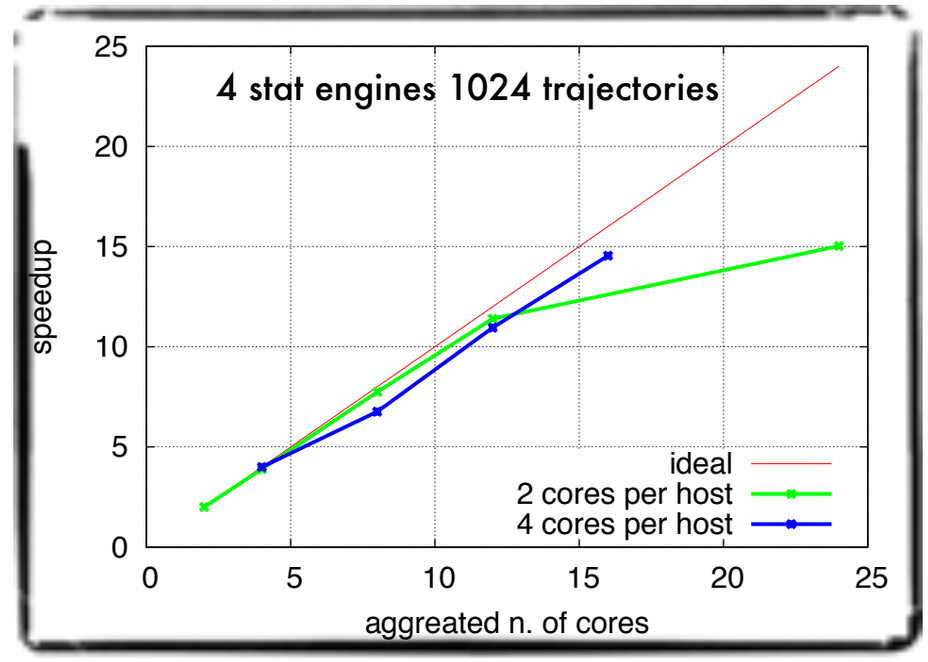
Performance (preliminary)





Intel
Nehalem
32-core





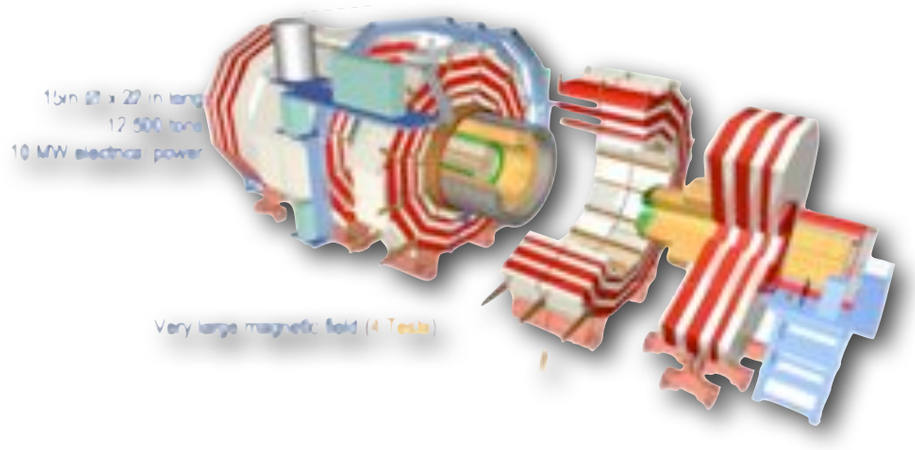
Cluster 8x
Intel Xeon 6-core
Infinband (IPoIB)

courtesy of
Mellanox

Involved data

- Simple examples (neurospora, ...)
  - `2-4 double * n. of variables * n. of samples x n. of trajectories * cases in sensitivity analysis`
  - `e.g.4*8*4*1M*1k*8 ~ 1 TBytes`
- HIV 6GB x 1024 trajectories ~ 6TB
- The more observed variables, precision, cases for sensitivity analysis the more data

Innovative Methods for Particle Colliders at the Terascale (2012-2015)



- CMS: Compact Muon Solenoid at CERN
  - 3500 scientists, 180 Universities and Research Labs (40 countries)
  - CMS is like a ~75 MegaPixels Digital Camera. 40M “photos”/s Selection of 300 ‘photos’/s ~450 MB/s from the detector are ~PBs of data/year
  - CERN has (of course) its well established data flow and infrastructure, however ...

Data flow of the CMS experiment

Enabling Grids for E-sciencE

450MB/s

TIER-0
CERN

CAF

WLCG Computing
Grid Infrastructure

30-300MB/s (ag. 600MB/s)

TIER-1 TIER-1 TIER-1 TIER-1 TIER-1 TIER-1

~50k jobs/day

10-20MB/s

50-500MB/s

TIER-2

~150k jobs/day

Tier-0

(the accelerator centre)

Data acquisition & initial processing
Long-term mass data storage
CMS CERN Analysis Facility
(latency critical data processing, high priority analysis)
Distribution of data → Tier-1 centres

7 Tier-1s

("online" to the DAQ)

High availability centers
Custodial mass storage of share of data
Data reconstruction and reprocessing
Data skimming
Distribute analysis data → Tier-2s

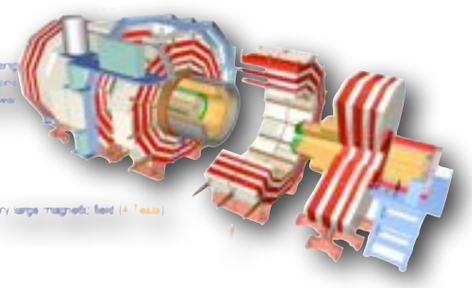
~50 Tier-2s

in ~26 countries

End-user physics analyses
Detector Studies
MonteCarlo Simulation → Tier-1

IMPACT

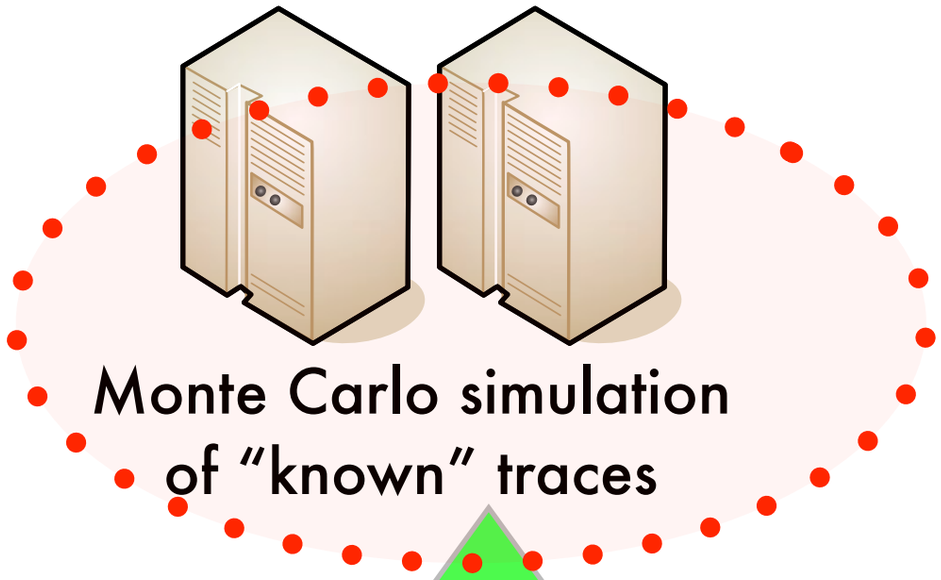
Innovative Methods for Particle Colliders at the Terascale (2012-2015, oversimplified vision)



huge data

Diff
(somehow)

huge data



Monte Carlo simulation
of "known" traces

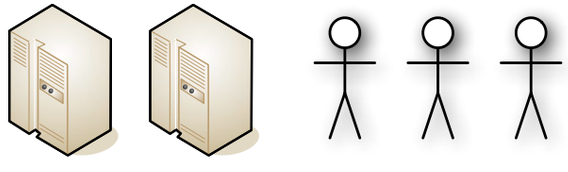
Filtered data
to be analysed
(much smaller)

Monte Carlo simulations
smaller but frequent
requires great "reactivity"
of the workflow to tune models

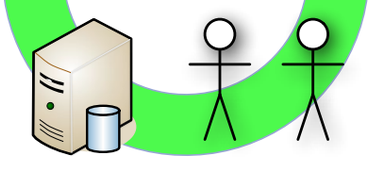
Preliminary results on channel H→ZZ→4l shown at Higg's boson claim Jul 2012 (Amapane at al.)

new speculations

new knowledge

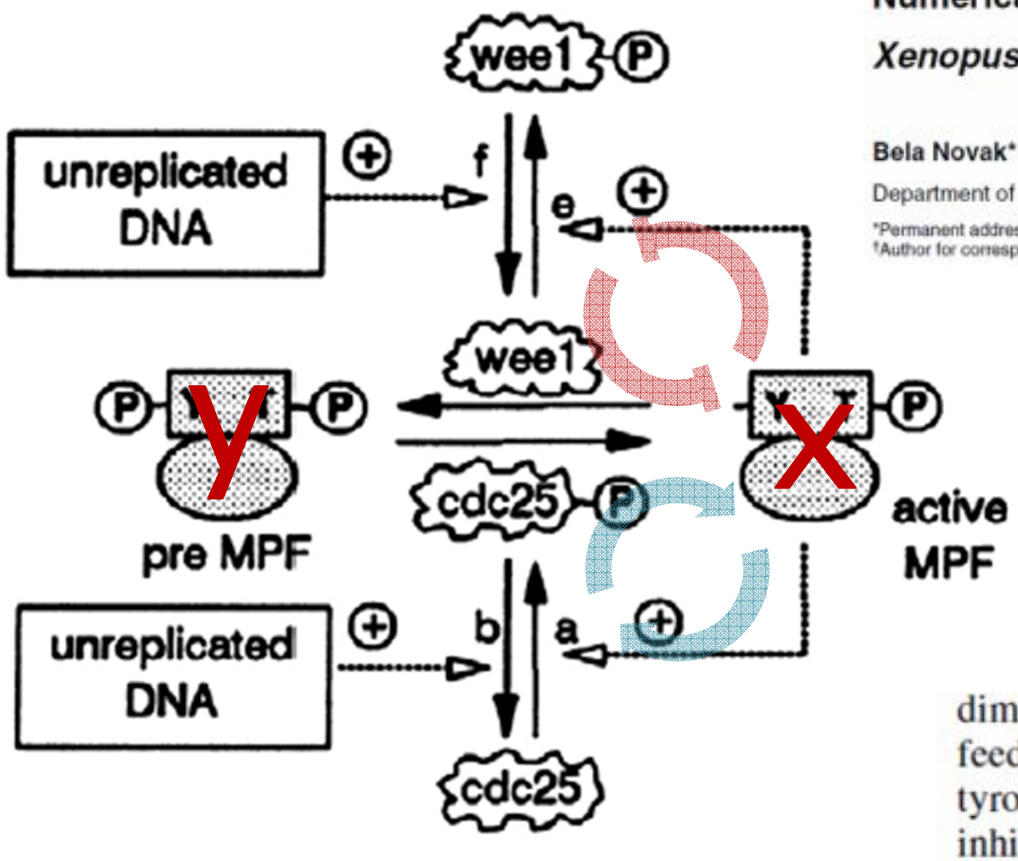


Particle physicists



Theoretic physicists

Formalising the cell cycle switch



Journal of Cell Science 106, 1153-1168 (1993)
Printed in Great Britain © The Company of Biologists Limited 1993

Numerical analysis of a comprehensive model of M-phase control in *Xenopus* oocyte extracts and intact embryos

Bela Novak* and John J. Tyson†

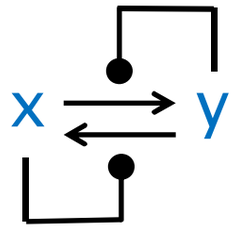
Department of Biology, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24060-0406, USA

*Permanent address: Department of Agricultural Chemical Technology, Technical University of Budapest, 1521 Budapest Gellert Ter 4, Hungary
†Author for correspondence

dimers is left off the diagram to keep it simple.) (B) Positive feedback loops. Active MPF stimulates its own production from tyrosine-phosphorylated dimers by activating Cdc25 and inhibiting Wee1. We suspect that these signals are indirect, but intermediary enzymes are unknown and we ignore them in this paper. The signals from active MPF to Wee1 and Cdc25 generate an autocatalytic instability in the control system. We indicate also an 'external' signal from unreplicated DNA to Wee1 and Cdc25, which can be used to control the efficacy of the positive feedback loops. The letters a, b, e and f are used to label the rate constants for these reactions in Fig. 2. (C) Negative feedback loop. Active

Direct competition: unstable switch

- x catalyzes the transformation of y into x
- y catalyzes the transformation of x into y



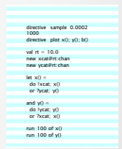
$$x + y \rightarrow x + x$$
$$y + x \rightarrow y + y$$

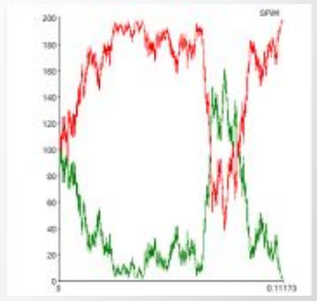
Courtesy of Luca Cardelli

On Switches and Oscillators Program Equivalence in Biology?

http://lucacardelli.name

- This system is bistable, but
  - Convergence to a stable state is slow (a random walk).
  - *Any* perturbation of a stable state can initiate a random walk to the other stable state.
  - With 100 molecules of x and y, convergence is quick, but with 10000 molecules, even at the same concentration (adjusting the rate) you will wait for a long time.

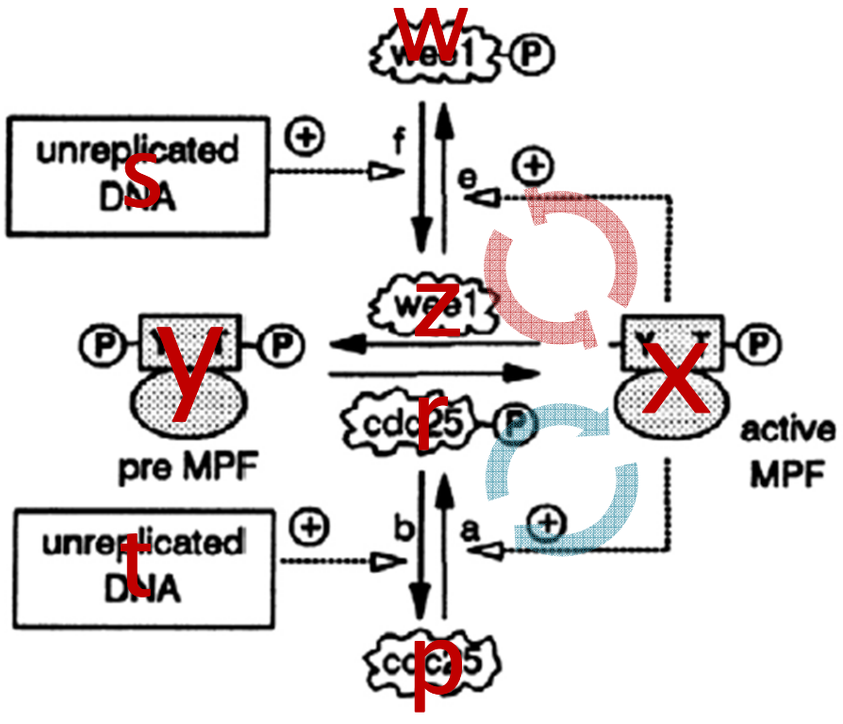


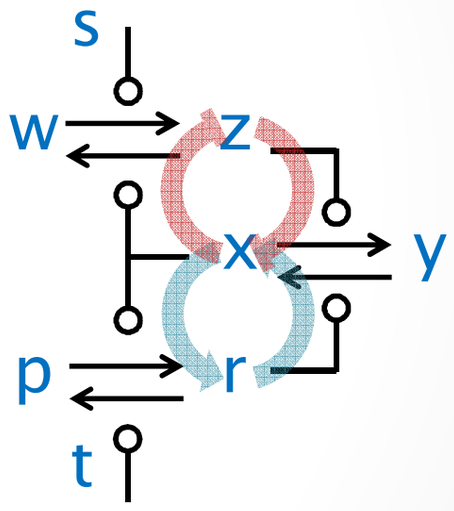


CWC syntax

$$\top : a\ c \xmapsto{10} a\ a \qquad \top : c\ a \xmapsto{10} c\ c$$

… after a number of transformations: a stable switch faithfully modelling cell switch





Courtesy of Luca Cardelli

On Switches and Oscillators Program Equivalence in Biology?

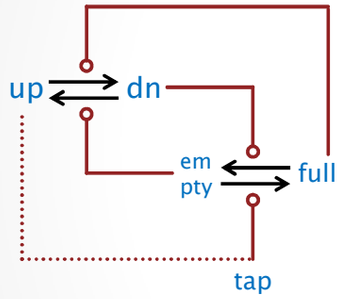
http://lucacardelli.name

CWC syntax

$$\top : a\ c \xrightarrow{10} c\ b \qquad \top : c\ a \xrightarrow{10} a\ b \qquad \top : b\ a \xrightarrow{10} a\ a \qquad \top : b\ c \xrightarrow{10} c\ c$$

The Shishi Odoshi

- A Japanese scarecrow (scare-deer)
  - Used by Bela Novak to illustrate the cell cycle switch.



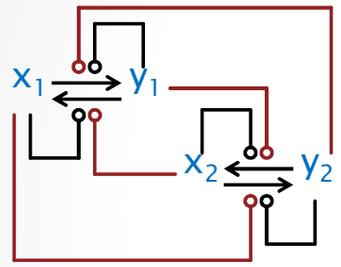
empty + tap → tap + full
up + full → full + dn
full + dn → dn + empty
dn + empty → empty + up

http://www.youtube.com/watch?v=VbvecTIftcE&NR=1&feature=fvwp

To make it into a full trammel (dotted line), we could make the up position mechanically open the tap (i.e. take up = tap)

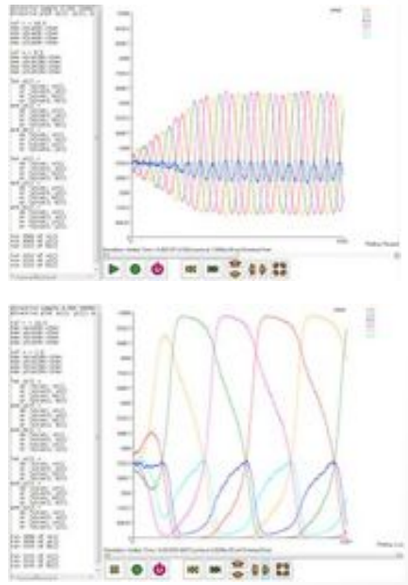
The 2AM Limit-Cycle Oscillator

- Two AM switches in a Trammel pattern



The red reactions need to be slower (even slightly) than the black reactions, but otherwise the oscillation is robust. Oscillation stops at 10 vs. 10 and 1 vs. 10. Here the rates are 8 vs 10.0 top, and 2 vs 10, bottom.

(Simple limit-cycle oscillators in the literature have very critical rate ranges.)

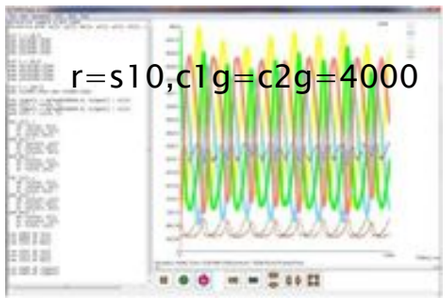


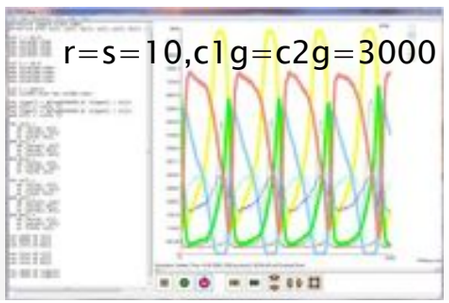
```
directive sample 0.001 10000
directive plot x1(); y1(); b1(); x2(); y2(); b2(); c1();
c2()

val r = 10.0
new x1cat@r:chan
new y1cat@r:chan
new x2cat@r:chan
new y2cat@r:chan

val s = 8.0
new x1cat2@s:chan
new y1cat2@s:chan
new x2cat1@s:chan
new y2cat1@s:chan

let x1() =
   do !x1cat; x1()
   or !x1cat2; x1()
   or ?y1cat; b1()
   or ?y2cat1; b1()
and y1() =
   do !y1cat; y1()
   or !y1cat2; y1()
   or ?x1cat; b1()
   or ?x2cat1; b1()
and b1() =
   do ?x1cat; x1()
   or ?x2cat1; x1()
   or ?y1cat; y1()
   or ?y2cat1; y1()

let x2() =
   do !x2cat; x2()
   or !x2cat1; x2()
   or ?y2cat; b2()
   or ?x1cat2; b2()
and y2() =
   do !y2cat; y2()
   or !y2cat1; y2()
   or ?x2cat; b2()
   or ?y1cat2; b2()
and b2() =
   do ?x2cat; x2()
   or ?y1cat2; x2()
   or ?y2cat; y2()
   or ?x1cat2; y2()

run 3666 of x1()
run 3000 of y1()
run 3333 of b1()

run 3333 of x2()
run 3333 of y2()
run 3333 of b2()
```

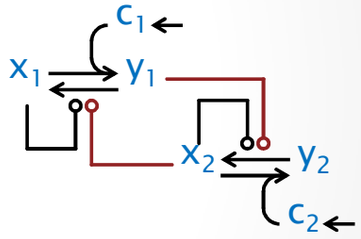

Influx Oscillators

- Similar but:
  - The two-input switches are replaced by one-input switches which are reset by constant influxes.

```
directive sample 0.002 1000
directive plot x1(); y1(); b1(); x2(); y2(); b2(); c1();
c2()

val r = 10.0
new x1cat@r:chan
new y1cat@r:chan
new x2cat@r:chan
new y2cat@r:chan

val s = 10.0
new x1cat2@s:chan
new y1cat2@s:chan
new x2cat1@s:chan
new y2cat1@s:chan

val t = 100.0
new c1ch@t:chan new c2ch@t:chan

let c1gen() = delay@100000.0; (c1gen() | c1())
and c1() = !c1ch; ()
let c2gen() = delay@100000.0; (c2gen() | c2())
and c2() = !c2ch; ()

let x1() =
   do !x1cat; x1()
   or !x1cat2; x1()
   or ?c1ch; b1()
and y1() =
   do !y1cat2; y1()
   or ?x1cat; b1()
   or ?x2cat1; b1()
and b1() =
   do ?x1cat; x1()
   or ?x2cat1; x1()
   or ?c1ch; y1()

let x2() =
   do !x2cat; x2()
   or !x2cat1; x2()
   or ?c2ch; b2()
and y2() =
   do !y2cat1; y2()
   or ?x2cat; b2()
   or ?y1cat2; b2()
and b2() =
   do ?x2cat; x2()
   or ?y1cat2; x2()
   or ?c2ch; y2()

run 1000 of x1()
run 6666 of y1()
run 2333 of b1()

run 3333 of x2()
run 3333 of y2()
run 3333 of b2()

run 4000 of c1gen()
run 4000 of c2gen()
```





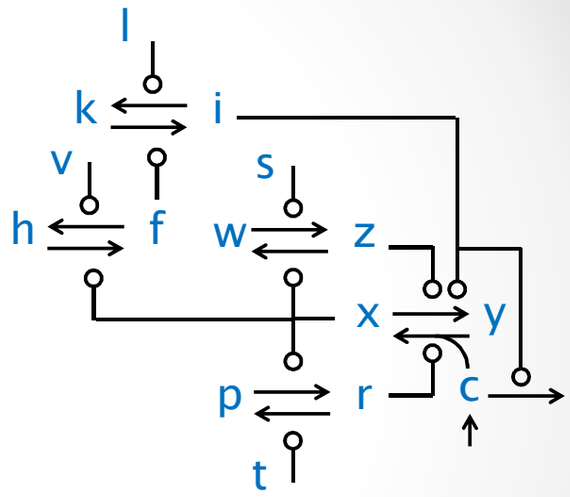


Works best with s=r.

Needs constant influx of c1,c2

Novak-Tyson Oscillator

- First switch
  - Is the 'transformed' AM switch in one-input configuration (driven by constant influx of cyclin).
- Second switch
  - Is a simple two-stage switch working as a delay (the first switch is so good in terms of hysteresis that the second switch is not very critical for oscillation).
  - It can be replaced by a one-stage switch (Ferrell's cell cycle osciallor) but oscillation is a bit harder to obtain.
- Connection
  - Single links, as in the influx oscillator.

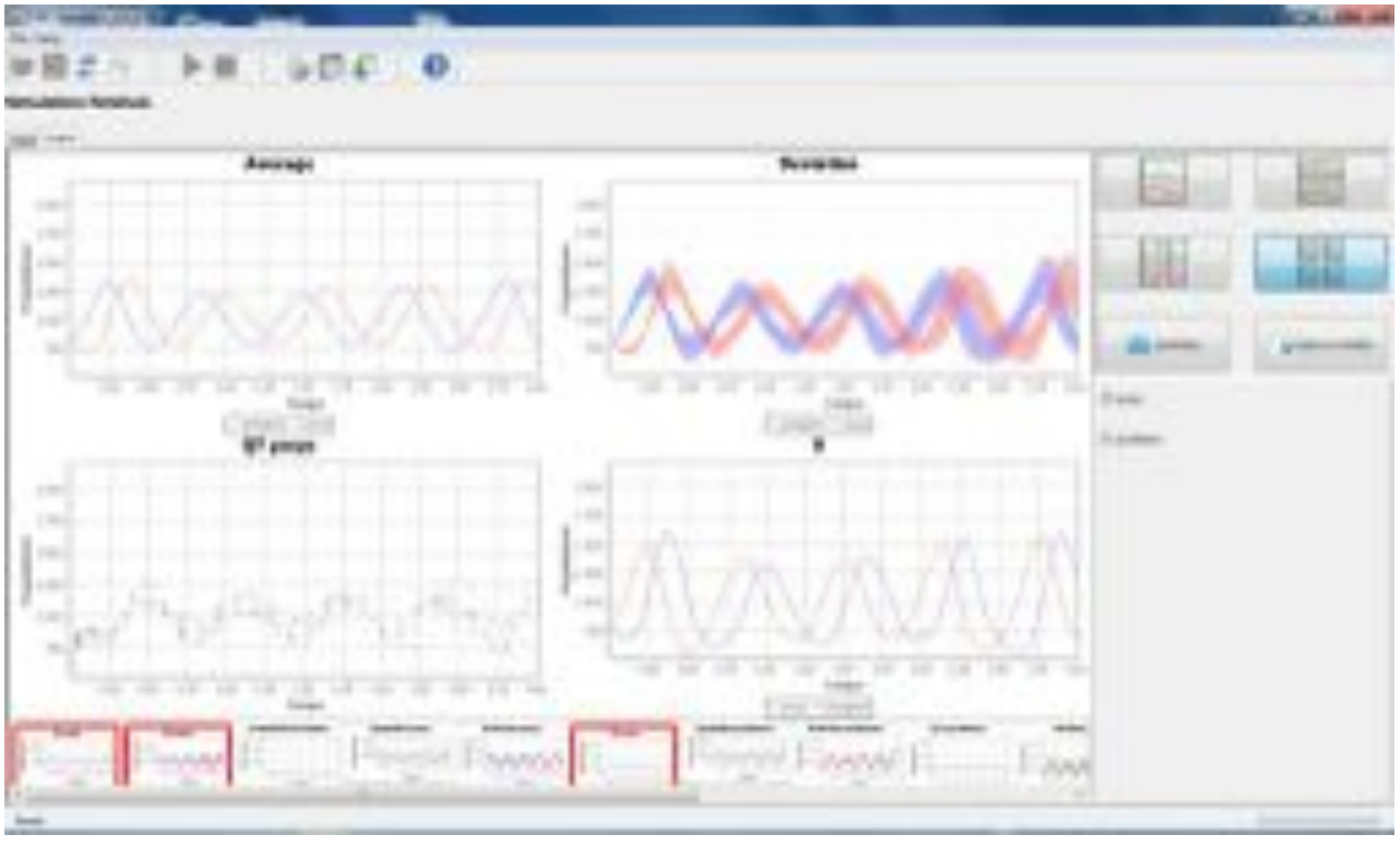


Journal of Cell Science 106, 1153-1168 (1993)
Printed in Great Britain © The Company of Biologists Limited 1993

Numerical analysis of a comprehensive model of M-phase control in *Xenopus* oocyte extracts and intact embryos

Bela Novak* and John J. Tyson†
Department of Biology, Virginia Polytechnic Institute and State University, Blacksburg, Virginia 24060-0406, USA

- Demo: unstable switch

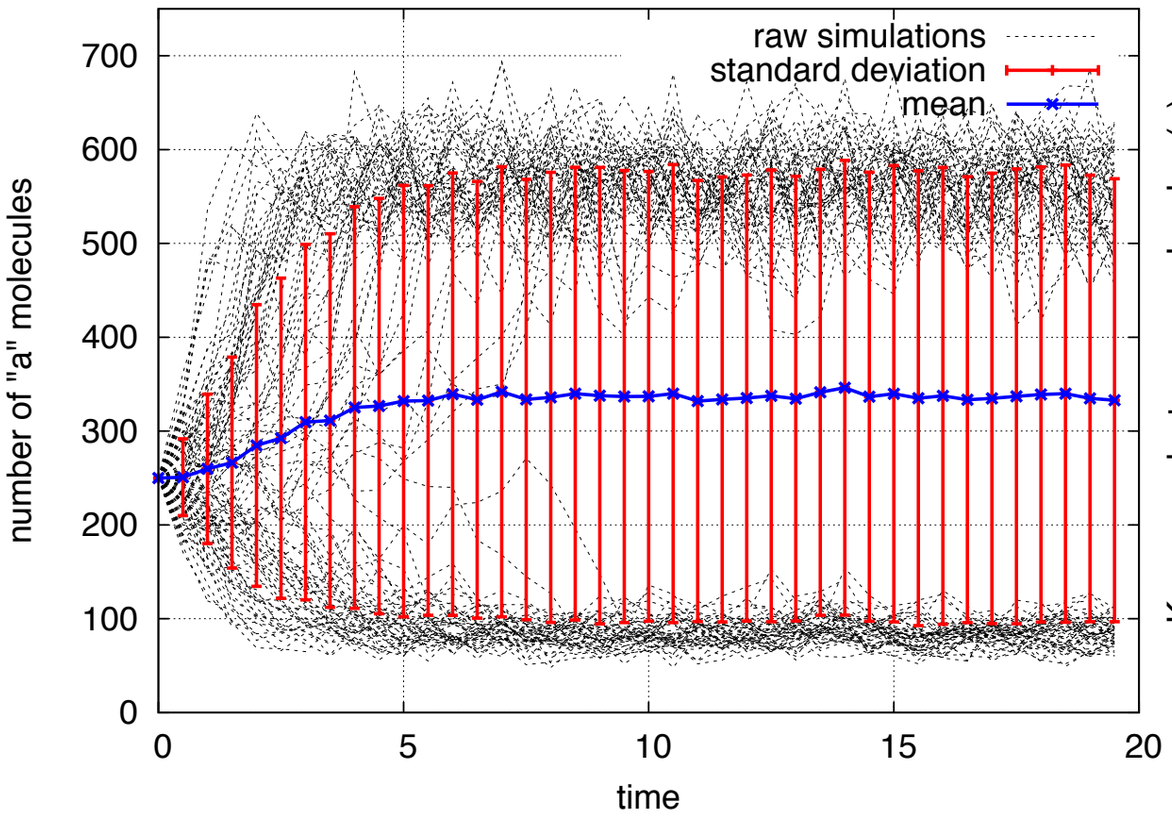


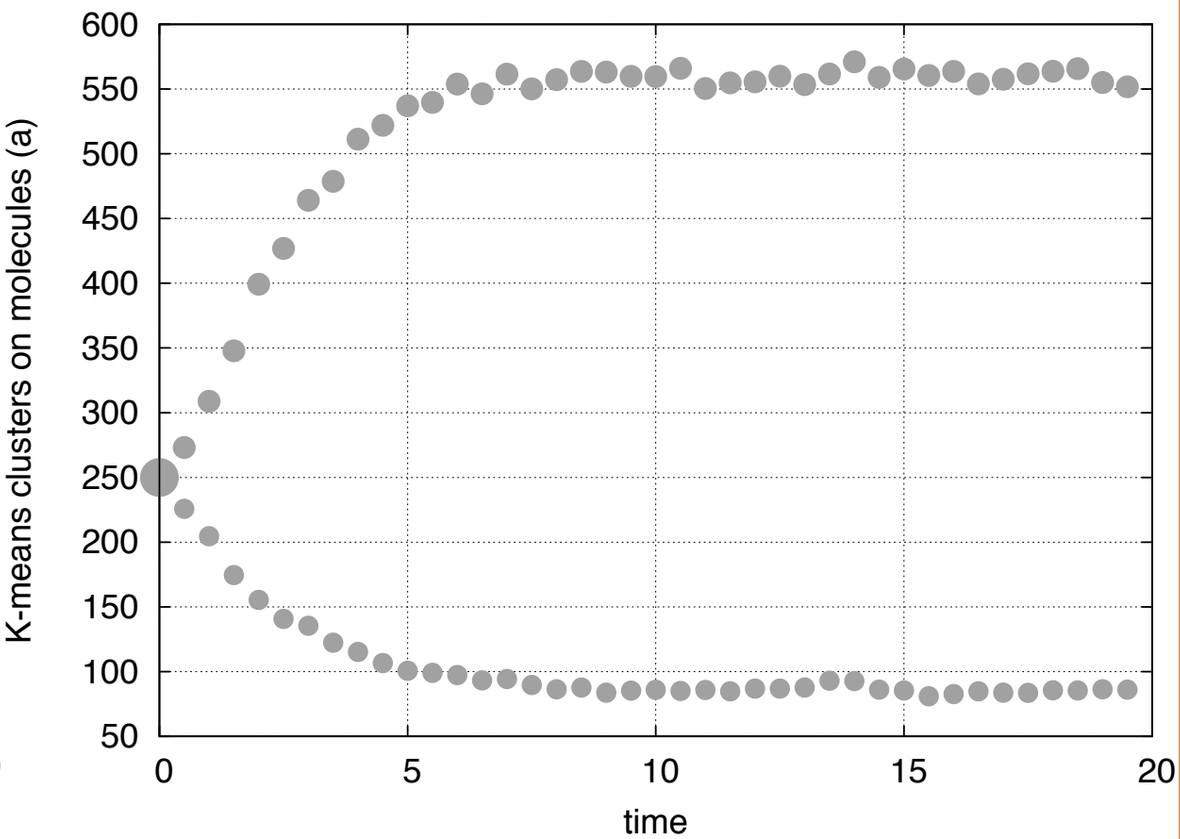
Schlögl model
autocatalytic, trimolecular reaction scheme (bistable)

$$\mathrm{A} + 2\mathrm{X} \underset{k_2}{\overset{k_1}{\rightleftharpoons}} 3\mathrm{X},$$

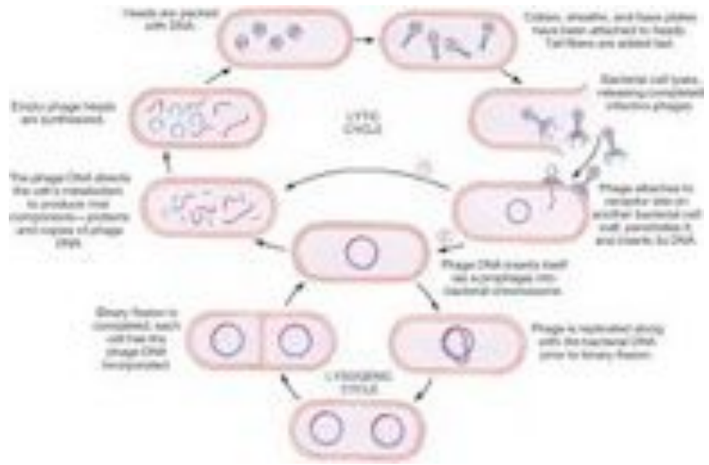
$$\mathrm{B} \underset{k_4}{\overset{k_3}{\rightleftharpoons}} \mathrm{X}.$$

$$\top : a\ a \xrightarrow{0.03} a\ a\ a \qquad \top : a\ a\ a \xrightarrow{0.0001} a\ a \qquad \top : b \xrightarrow{200} b\ a \qquad \top : a \xrightarrow{3.5} \bullet$$





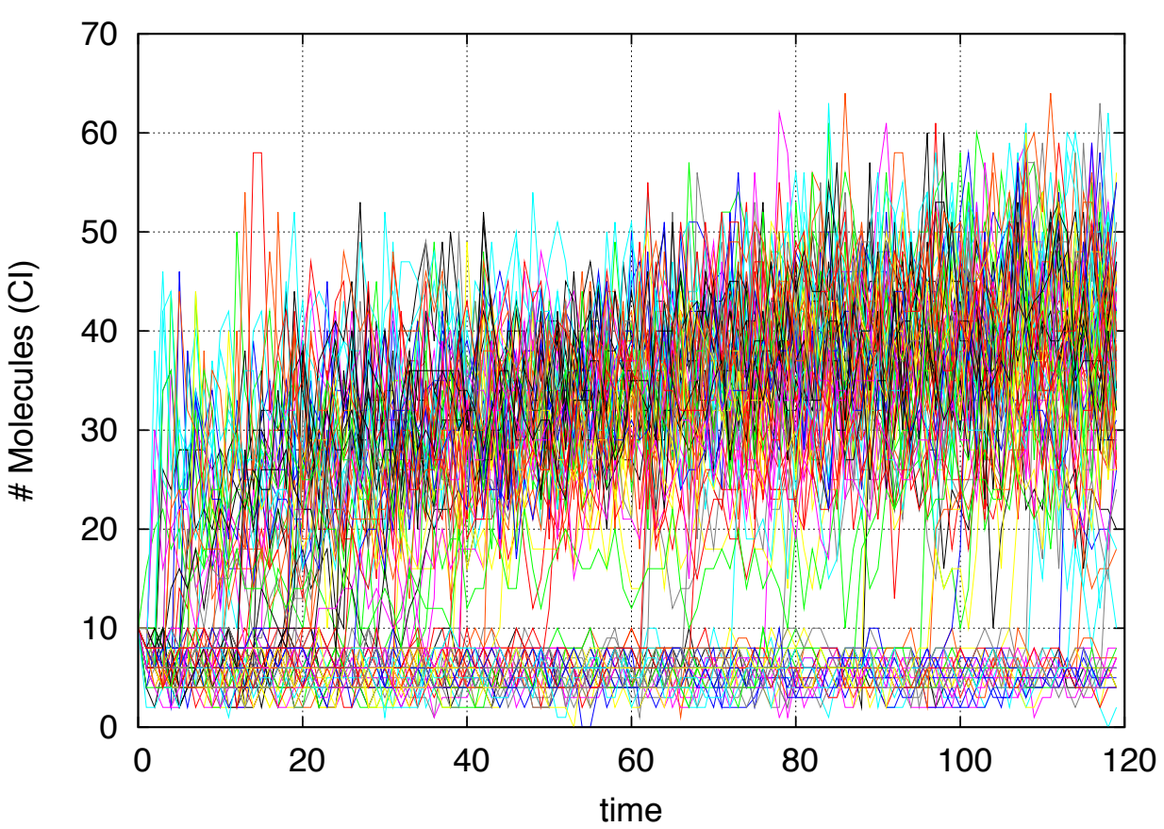
Bacteriophage λ life cycle integration of a strand of DNA in the molecule of E. coli DNA (multi-stable)

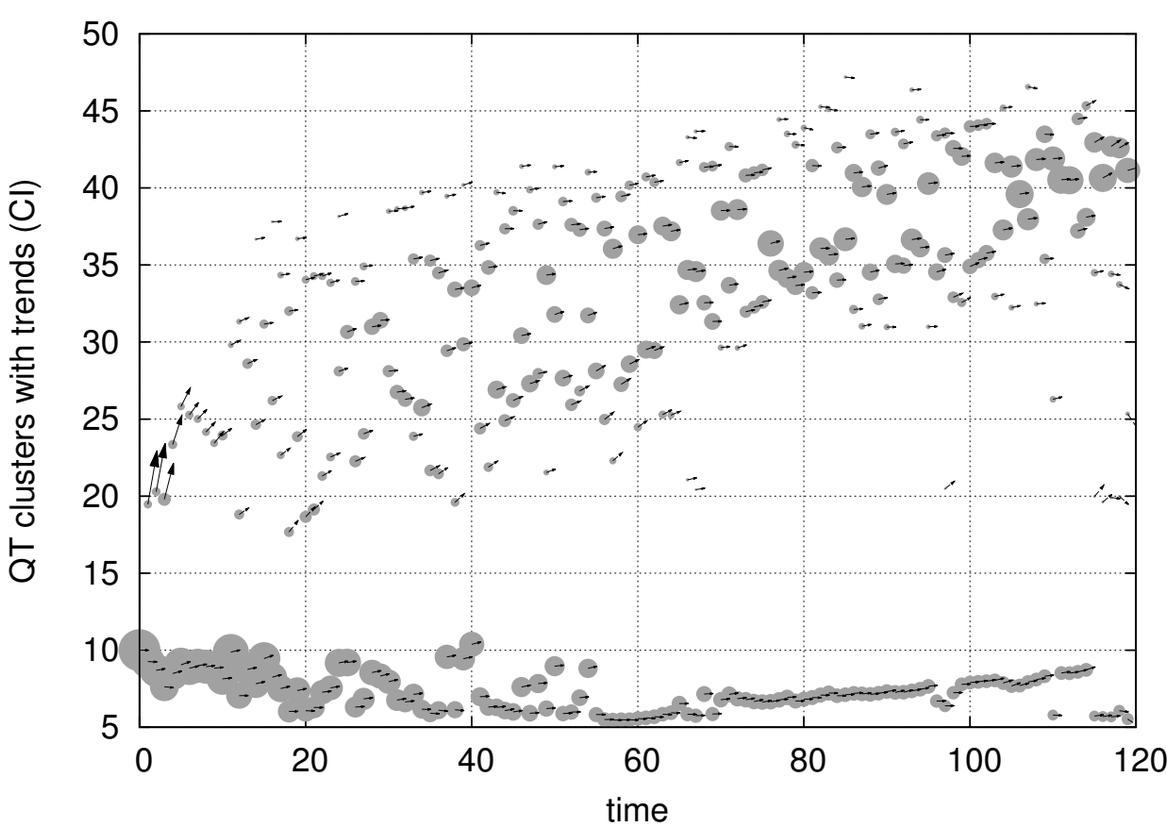


$$\top : CI\ CI \xmapsto{K_1} CI2 \qquad \top : CI2 \xmapsto{K_1} CI\ CI \qquad \top : CI2\ D \xmapsto{K_2} DCI2 \qquad \top : DCI2 \xmapsto{K_2} CI2\ D$$

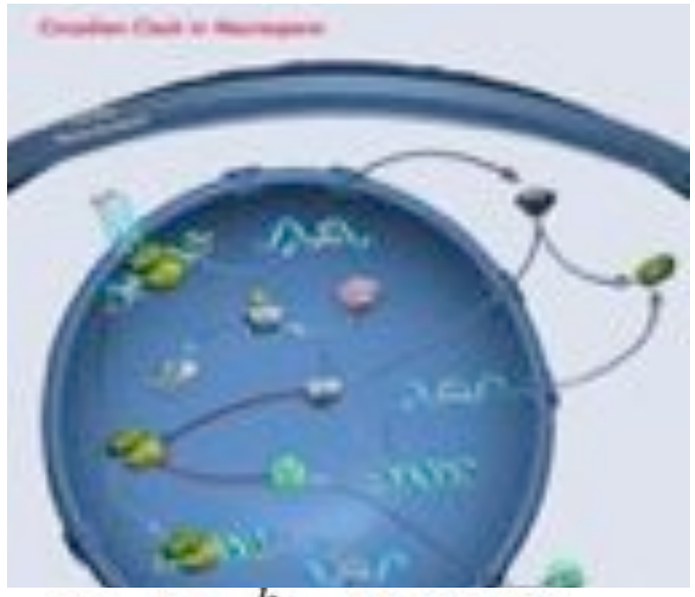
$$\top : CI2\ D \xmapsto{K_3} DCI2^* \qquad \top : DCI2^* \xmapsto{K_3} CI2\ D \qquad \top : DCI2\ CI2 \xmapsto{K_4} DCI2CI2$$

$$\top : DCI2CI2 \xmapsto{K_4} DCI2\ CI2 \qquad \top : DCI2\ P \xmapsto{k_t} DCI2\ P\ CI2\ CI2 \qquad CI \xmapsto{k_d} \bullet$$





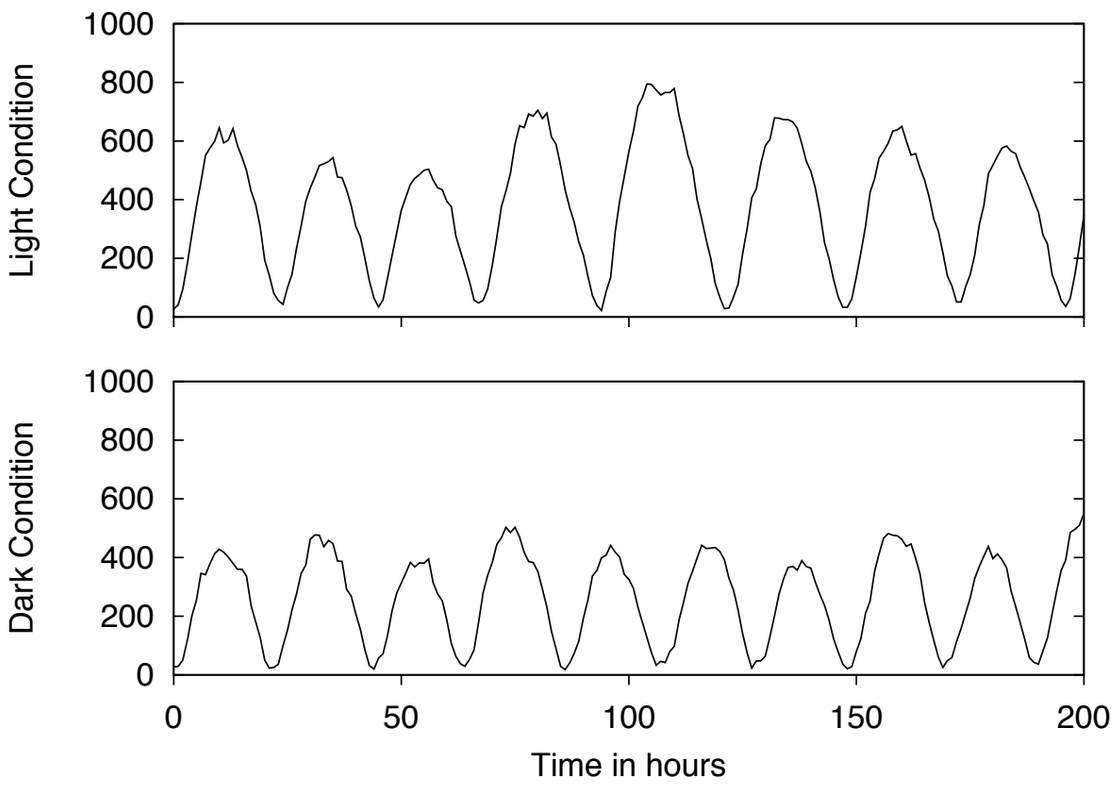
Transcriptional regulation in Neurospora (circadian clock period detection)

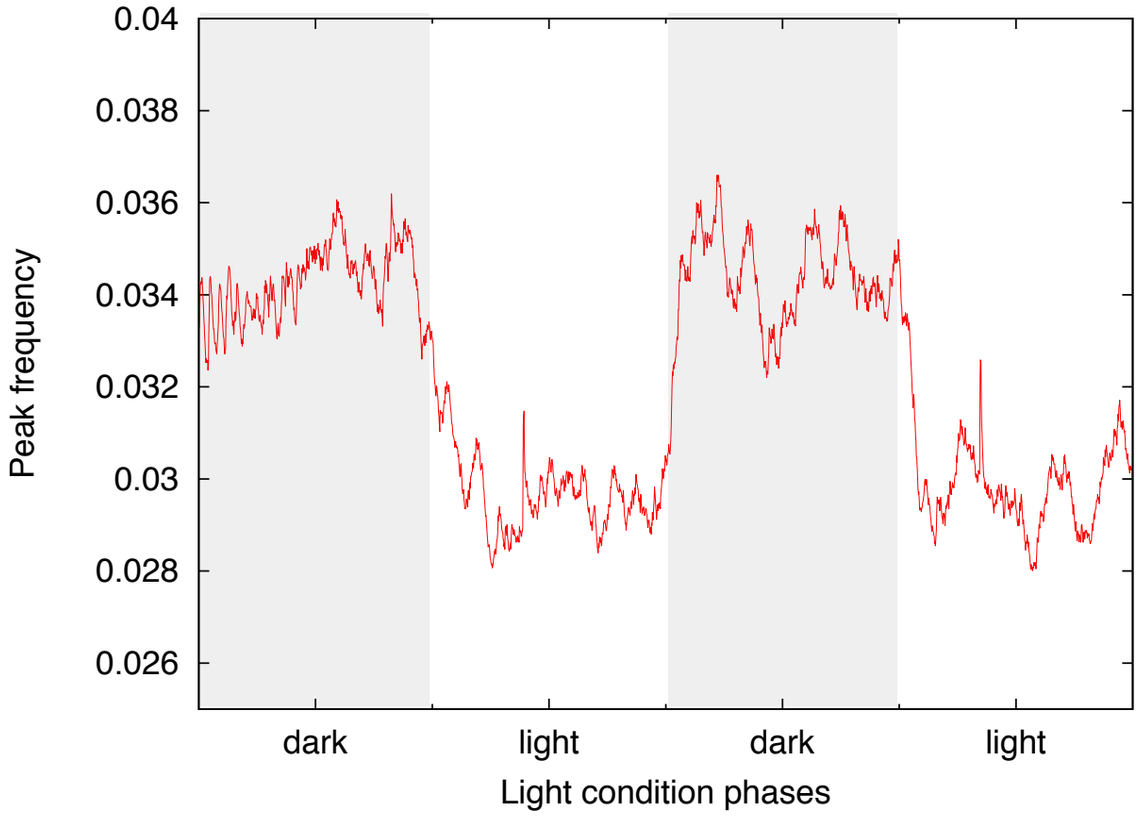


$$\top : (x \rfloor FRQ\ X)^{\eta} \overset{f_{FRQ}(t)}{\longmapsto} (x \rfloor FRQ\ X)^{\eta}\ M \qquad \top : M \overset{k_s}{\longmapsto} M\ FRQ$$

$$\top : M \overset{f_M}{\longmapsto} \bullet \qquad \top : FRQ \overset{f_d}{\longmapsto} \bullet$$

$$\top : FRQ(x \rfloor X)^{\eta} \overset{k_1}{\longmapsto} (x \rfloor FRQ\ X)^{\eta} \qquad \top : (x \rfloor FRQ\ X)^{\eta} \overset{k_2}{\longmapsto} FRQ(x \rfloor X)^{\eta}$$





Conclusions

- Talk focused on programming model
  - Many important“in a cloud” ignored in the talk: middleware, faults, ...
  - Data movement & high-level are key features
    - helps the mapping of features to platforms and performance portability
    - ease the design
    - MapReduce is an instance, should be not the only one
- Formal biology at embryonal stage
  - similar data & computation problems in analysis “in vitro” experiments
  - increasing interest from industrial and “core” bio scientist for parallel computing

Acknowledgements

- FastFlow
  - Massimo Torquati (Pisa), Marco Danelutto (Pisa), Peter Kilpatrick (Belfast), Massimiliano Meneghin (IBM Research Dublin)
- Case studies, biological models
  - Luca Cardelli (Microsoft), Pietro Liò (Cambridge), Andrea Bracciali (Stirling), Cristina Calcagno (Torino)
- CWC simulation implementation
  - Maurizio Drocco (Torino), Fabio Tordini (Torino)
- CWC language design
  - Maurizio Drocco, Eva Sciacca, Salvatore Spinella, Mario Coppo, Angelo Troina, Ferruccio Damiani (Torino)
- Graphical User Interface
  - ETICA company
- Infiniband cluster and computing facilities
  - Mellanox, HPC Advisory Council