BioBITs

Euromicro PDP 2011 - Ayia Napa - Cyprus
10th Feb 2011

On Designing Multicore-Aware Simulators for Biological Systems

Marco Aldinucci
Mario Coppo, Ferruccio Damiani, Maurizio Drocco*, Angelo Troina
Computer Science Dept. - University of Torino - Italy
Massimo Torquati
Computer Science Dept. - University of Pisa - Italy

Outline

Modelling Complex Bio Systems

- A large effort to formally model complex systems is underway. Goal: developing a discipline for engineering
  - synthetic immune responses, virus diffusion, social behaviours, ...
- Two main approaches to study models
  - (Traditional) Ordinary Differential Equations (ODEs) + numerical solvers
  - Stochastic Process calculi + Monte Carlo Simulations (e.g. Gillespie ...)
    - Slower but able to (theoretically) model non-steady-state, non-average dynamic of systems

Computing Models for Systems Bio

- Lambda-calculus [Fontana & Buss, 1996];
- Petri nets [Matsuno et al., 2000];
- Process Calculi:
  - Biological π-calculus [Regev, Shapiro et al., 2001/2002]; BioAmbients [Regev et al. 2004]; Brane Calculi [Cardelli, 2005]; Beta-binders [Priami & Quaglia, 2005]; BioPEPA [Hillston et al., 2006];
- Rewrite Systems:
  - P-Systems [Paun, 1998]; κ-calculus [Danos & Laneve, 2003]; CLS [Barbuti et al. 2005]; Stochastic Bigraphs [Krivine et al., 2007]; **CWC** [Coppo et al. 2010];
- Statecharts [Harel et al., 2003];
- Hybrid Automata [Mishra et al. 2006]; ...

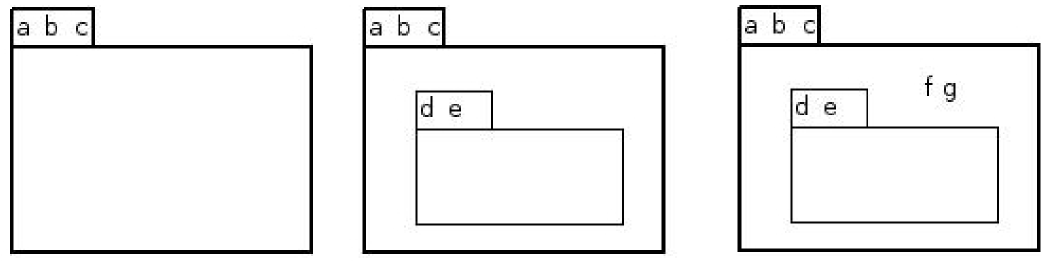
The Calculus of Wrapped Compartments (CWC)

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:

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

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

Examples of SCWC terms



(i) (ii) (iii)

- (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).

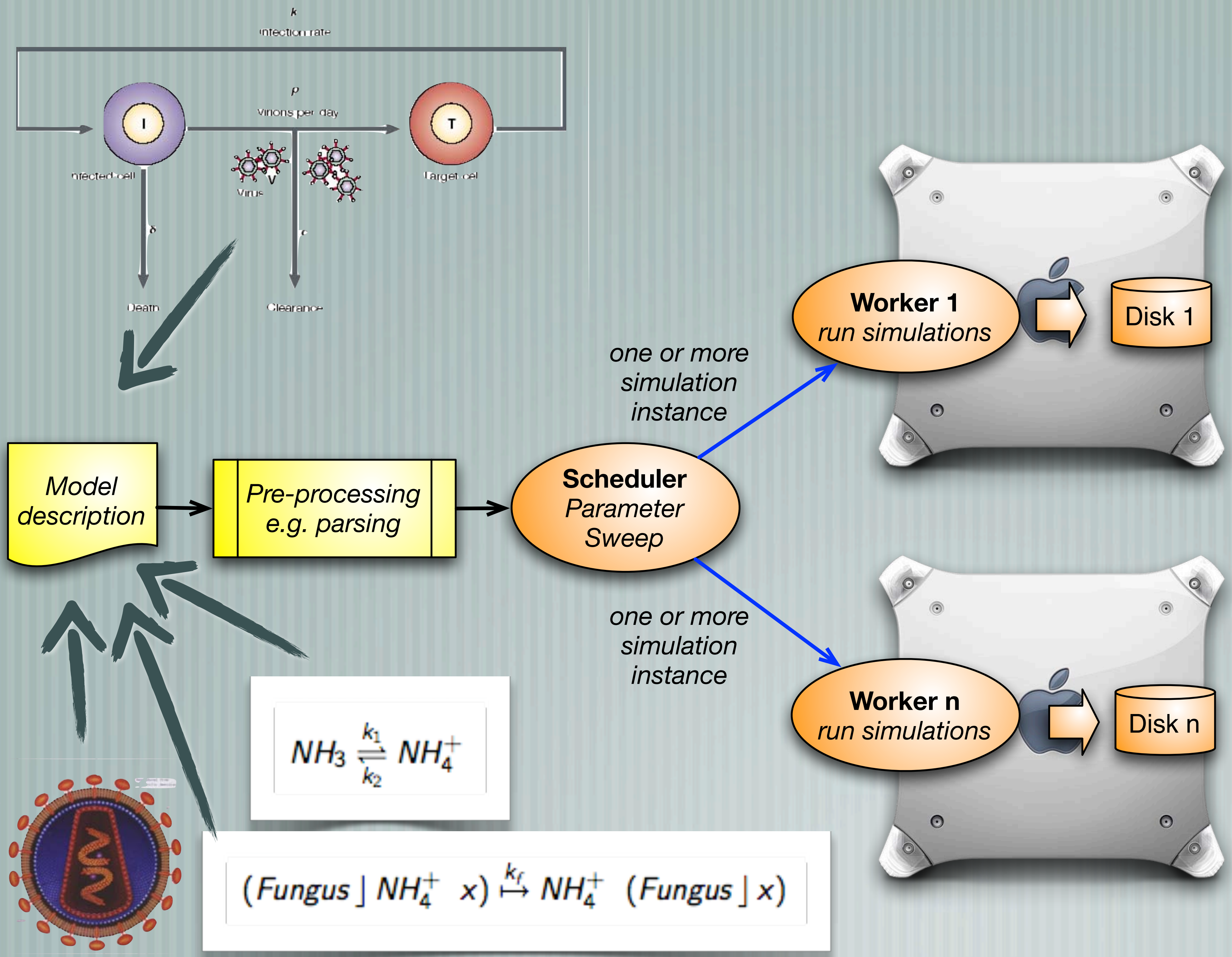
A **Stochastic Rewrite Rule**, R, is denoted by $P \overset{k}{\mapsto} P'$.

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:

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

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.

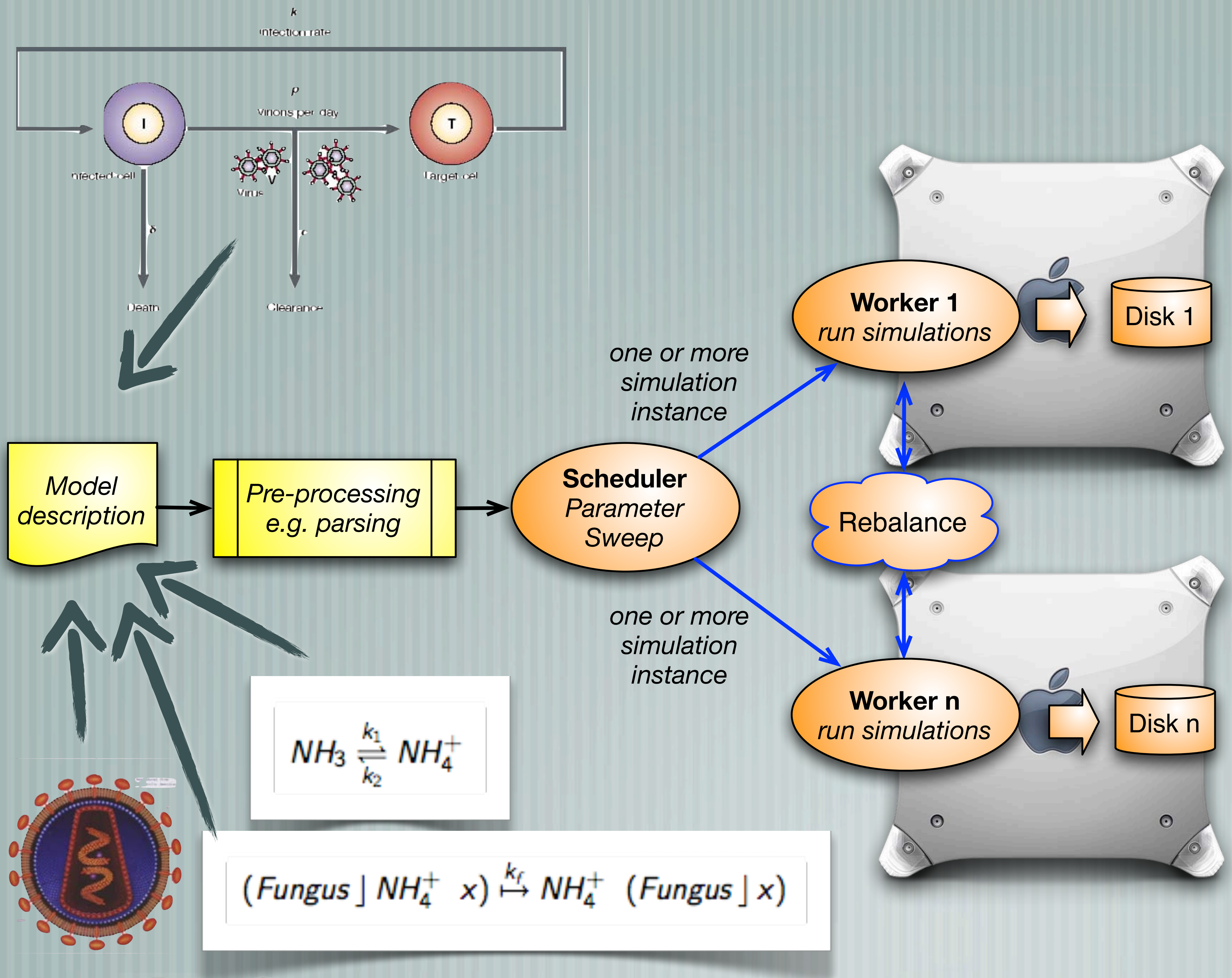
MonteCarlo sim: distributed solution



$$NH_3 \underset{k_2}{\overset{k_1}{\rightleftharpoons}} NH_4^+$$

$$(Fungus \rfloor NH_4^+ \; x) \overset{k_f}{\hookrightarrow} NH_4^+ \; (Fungus \rfloor x)$$

MonteCarlo sim: distributed solution

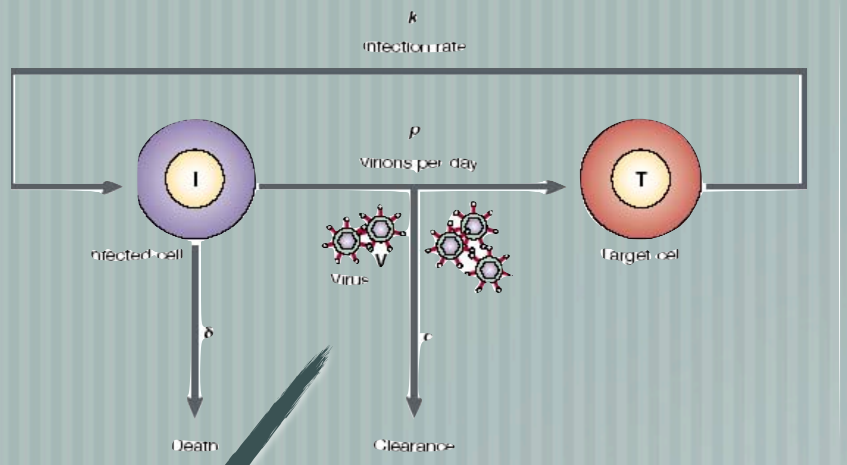


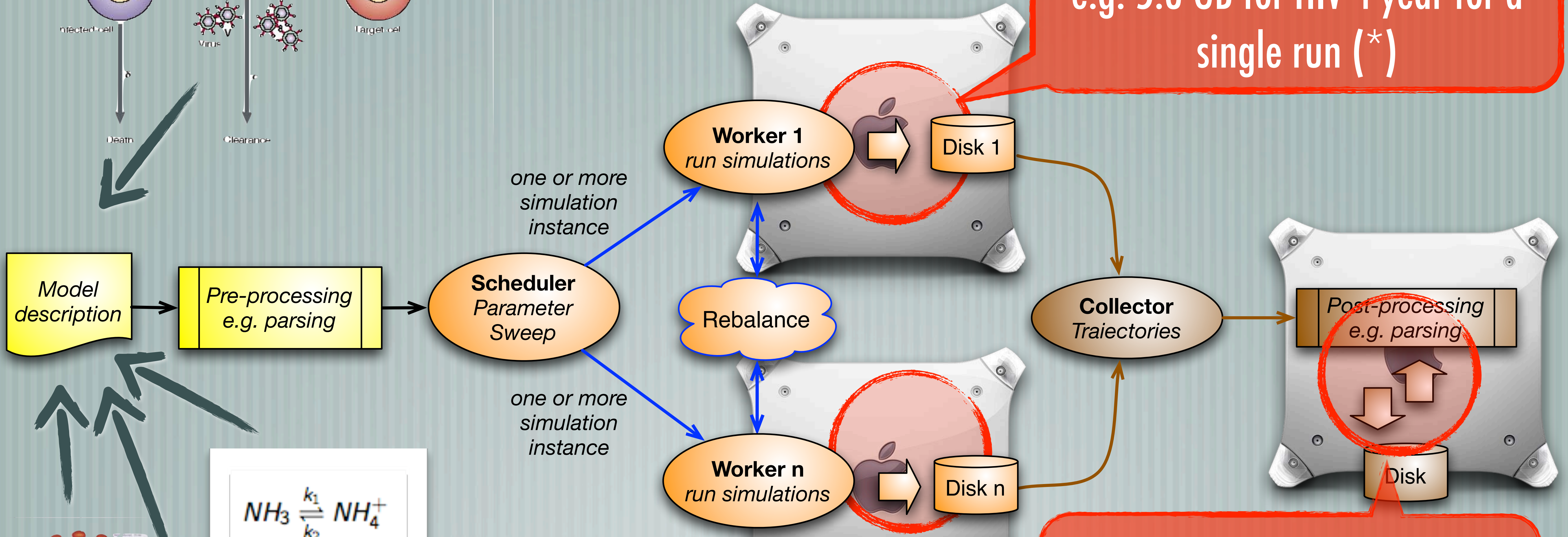
$$NH_3 \underset{k_2}{\overset{k_1}{\rightleftharpoons}} NH_4^+$$

$$(Fungus \rfloor NH_4^+ \;\; x) \overset{k_f}{\hookrightarrow} NH_4^+ \;\; (Fungus \rfloor x)$$

MonteCarlo sim: distributed solution

In BioSims the whole trajectory is needed, easily GB of data, e.g. 5.6 GB for HIV 4-year for a single run (*)




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

$$NH_3 \underset{k_2}{\overset{k_1}{\rightleftharpoons}} NH_4^+$$

$$(Fungus \rfloor NH_4^+ \;\; x) \xrightarrow{k_f} NH_4^+ \;\; (Fungus \rfloor x)$$

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

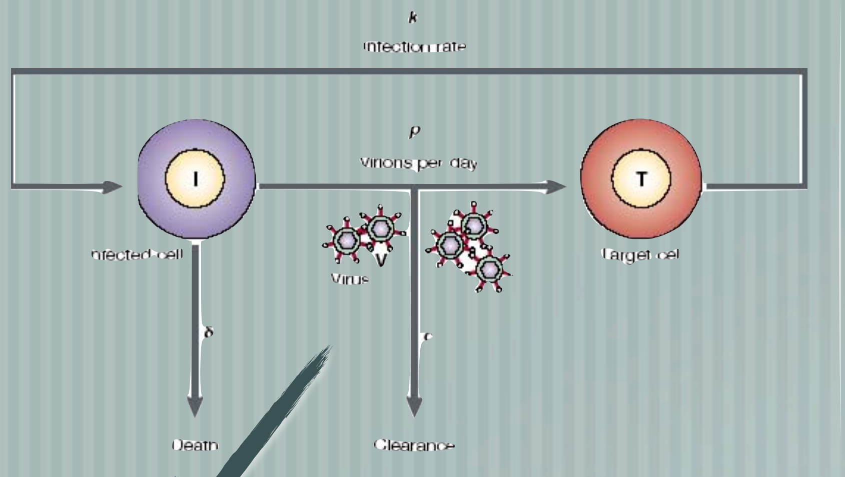
(*) M. Aldinucci, A. Bracciali, P. Liò, A. Sorathiya, and M. Torquati. StochKit-FF: Efficient systems biology on multicore architectures. In Euro-Par Workshops 2010, LNCS, Ischia, Italy, Sept. 2010. Springer.

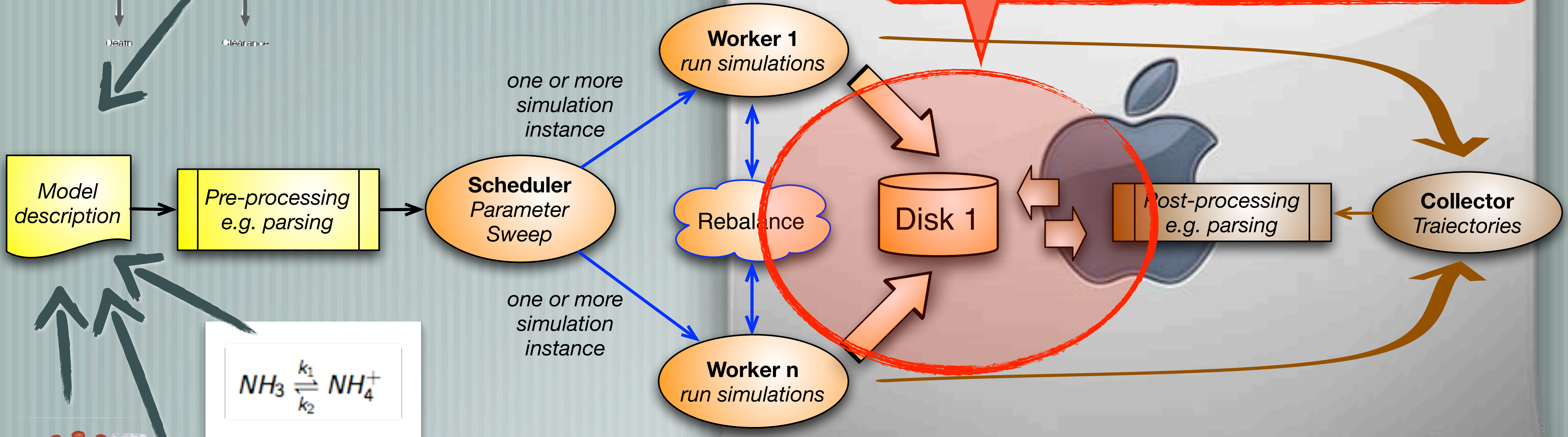
MonteCarlo sim: on multicore?

Now the issue become a real problem

Bottlenecks: disk and memory

Post-processing: longer pipelined





Model description
Pre-processing e.g. parsing
Scheduler Parameter Sweep
one or more simulation instance
one or more simulation instance
Worker 1 run simulations
Rebalance
Worker n run simulations
Disk 1
Post-processing e.g. parsing
Collector Trajectories

$$NH_3 \underset{k_2}{\overset{k_1}{\rightleftharpoons}} NH_4^+$$

$$(Fungus \rfloor NH_4^+ \;\; x) \overset{k_f}{\hookrightarrow} NH_4^+ \;\; (Fungus \rfloor x)$$

From Distributed to Multicore

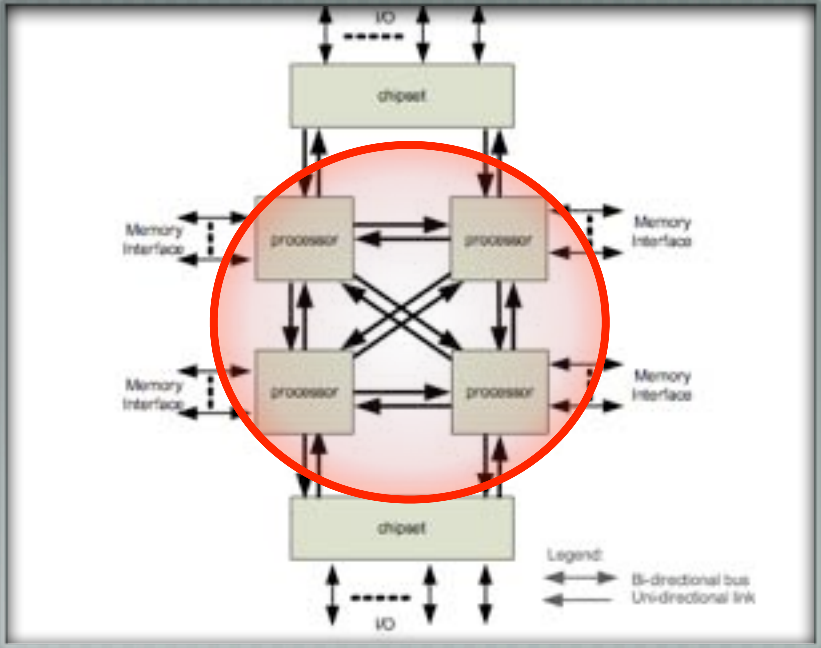
- MultiCarlo sims for Bio are I/O-bound
  - The whole trajectory is needed
  - Sampling reduce I/O traffic but worsen precision and analysis of “strange” dynamics (spikes, diversion from average, etc.), which observation motivates stochastic analysis (ODEs)
- The porting of distributed solution “as is” on multicore is going insist on weak points of multicore architectures
  - Memory wall, I/O, disk
  - SIMD/GPGPUs do not change the analysis substantially

Manage large data set on multicore

- Biological data is typically huge
  - not only simulators but also data from DB/web, analysis instruments, ...
- Rationale
  - Manage data as stream, compute everything online
  - Establish fast data paths across cores
  - Avoid low-level concurrency management
    - Portability, performance, portability of performance, maintenance, porting from sequential

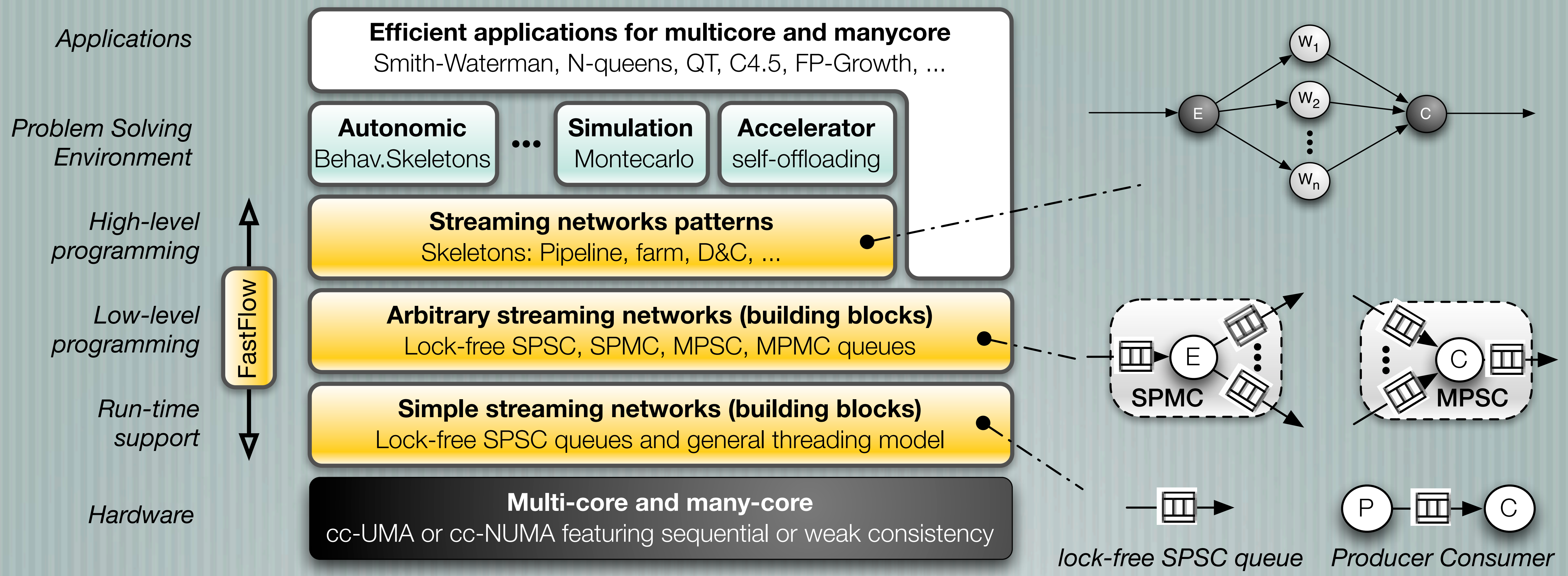
This and next generation Multi-cores

- Are programmed at "concurrent assembler" level
  - Complex, not portable, not efficient
- Exploit cache coherence
  - Lock/Memory-Fences are expensive
  - Will worsen with core count
  - Atomic ops do not solve the problem (still fences)
- Fine-grained parallelism is the problem
  - I/O bound problems, High-throughput, Streaming, Irregular DP problems



[2009] i7 QuickPath
(MESI-F Directory Coherence)

FastFlow: easy streaming in C++


Applications
Problem Solving Environment
High-level programming
Low-level programming
Run-time support
Hardware
FastFlow
Efficient applications for multicore and manycore
Smith-Waterman, N-queens, QT, C4.5, FP-Growth, ...
Autonomic
Behav.Skeletons
...
Simulation
Montecarlo
Accelerator
self-offloading
Streaming networks patterns
Skeletons: Pipeline, farm, D&C, ...
Arbitrary streaming networks (building blocks)
Lock-free SPSC, SPMC, MPSC, MPMC queues
Simple streaming networks (building blocks)
Lock-free SPSC queues and general threading model
Multi-core and many-core
cc-UMA or cc-NUMA featuring sequential or weak consistency
E
W1
W2
Wn
C
E
SPMC
C
MPSC
P
C
lock-free SPSC queue
Producer Consumer

High-level programming

- Implemented on top of lock-free/fence-free non-blocking synchronisations
- C++ STL-like implementation

Lock-free and CAS-free (fence-free)

- Single-Producer-Single-Consumer FIFO queues
  - Lamport et al. 1983 Trans. PLS (Sequential consistency only - passive)
  - Higham and Kawash, 1997 ISPAN (Relaxed consistencies (e.g. TSO) - passive)
  - Giacomoni et al. 2008 PPoPP (Relaxed cons + cache friendly - passive)
- Multiple-Producers-Multiple-Consumers FIFO queues
  - with CAS (at least one) - passive ... a plethora
  - without CAS - passive ➟ Cannot be done
  - without CAS - active ➟ FastFlow

Lamport & FastFlow FIFO queues

```
push_nonbocking(data) {
  if (NEXT(head) == tail) {
    return EWOULDBLOCK;
  }
  buffer[head] = data;
  head = NEXT(head);
  return 0;
}

pop_nonblocking(data) {

  if (head == tail) {
    return EWOULDBLOCK;
  }
  data = buffer[tail];
  tail = NEXT(tail);
  return 0;
}
```

Lamport FIFO
1983

```
push_nonbocking(data) {
  if (NULL != buffer[head]) {
    return EWOULDBLOCK;
  }
  buffer[head] = data;
  head = NEXT(head);
  return 0;
}

pop_nonblocking(data) {
  data = buffer[tail];
  if (NULL == data) {
    return EWOULDBLOCK;
  }
  buffer[tail] = NULL;
  tail = NEXT(tail);
  return 0;
}
```

FastFlow FIFO
(derived from P1C1 - Higham 1997)

Lamport & FastFlow FIFO queues

```
push_nonbocking(data) {
  if (NEXT(head) == tail) {
    return EWOULDBLOCK;
  }
  buffer[head] = data;
  head = NEXT(head);
  return 0;
}

pop_nonblocking(data) {

  if (head == tail) {
    return EWOULDBLOCK;
  }
  data = buffer[tail];
  tail = NEXT(tail);
  return 0;
}
```

head and tail are mutually invalidated by producer and consumer
1 cache miss every push and pop (at least)

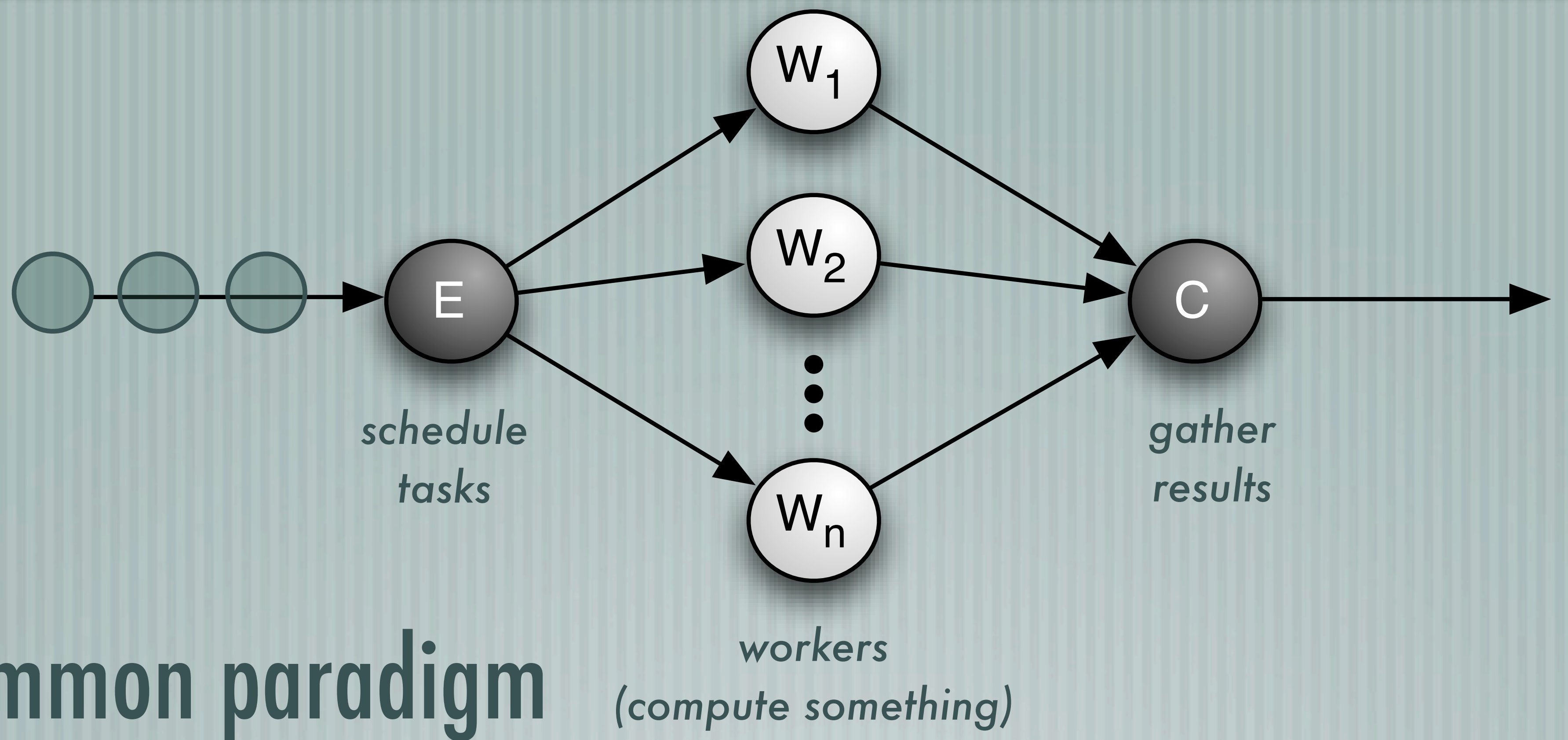
Lamport FIFO
1983

```
push_nonbocking(data) {
  if (NULL != buffer[head]) {
    return EWOULDBLOCK;
  }
  buffer[head] = data;
  head = NEXT(head);
  return 0;
}

pop_nonblocking(data) {
  data = buffer[tail];
  if (NULL == data) {
    return EWOULDBLOCK;
  }
  buffer[tail] = NULL;
  tail = NEXT(tail);
  return 0;
}
```

producer read/write head
consumer read/write tail
no misses

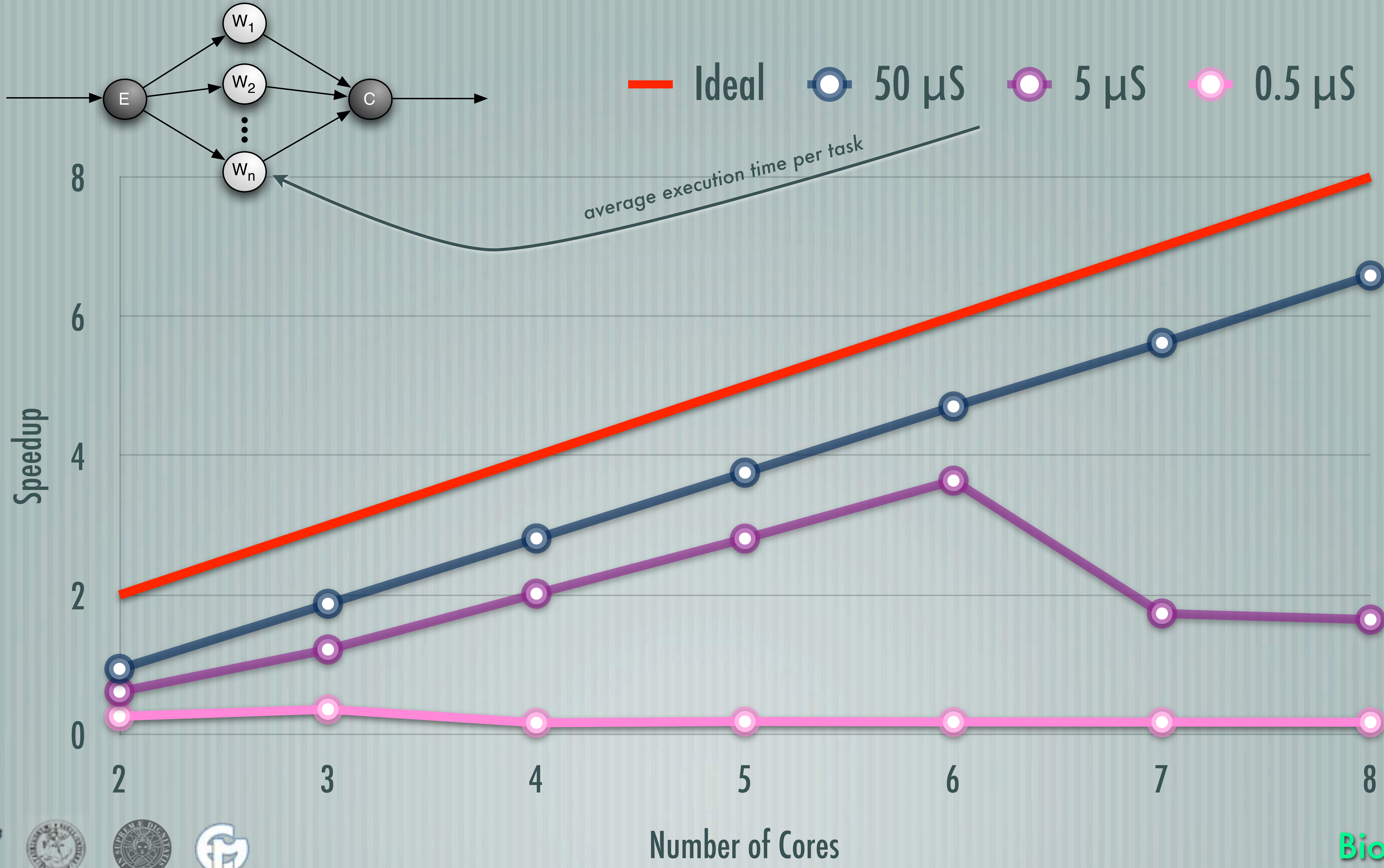
FastFlow FIFO
(derived from P1C1 - Higham 1997)

E.g. farm (a.k.a. master-workers)

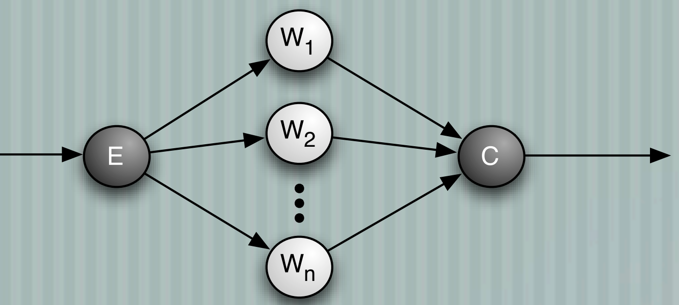


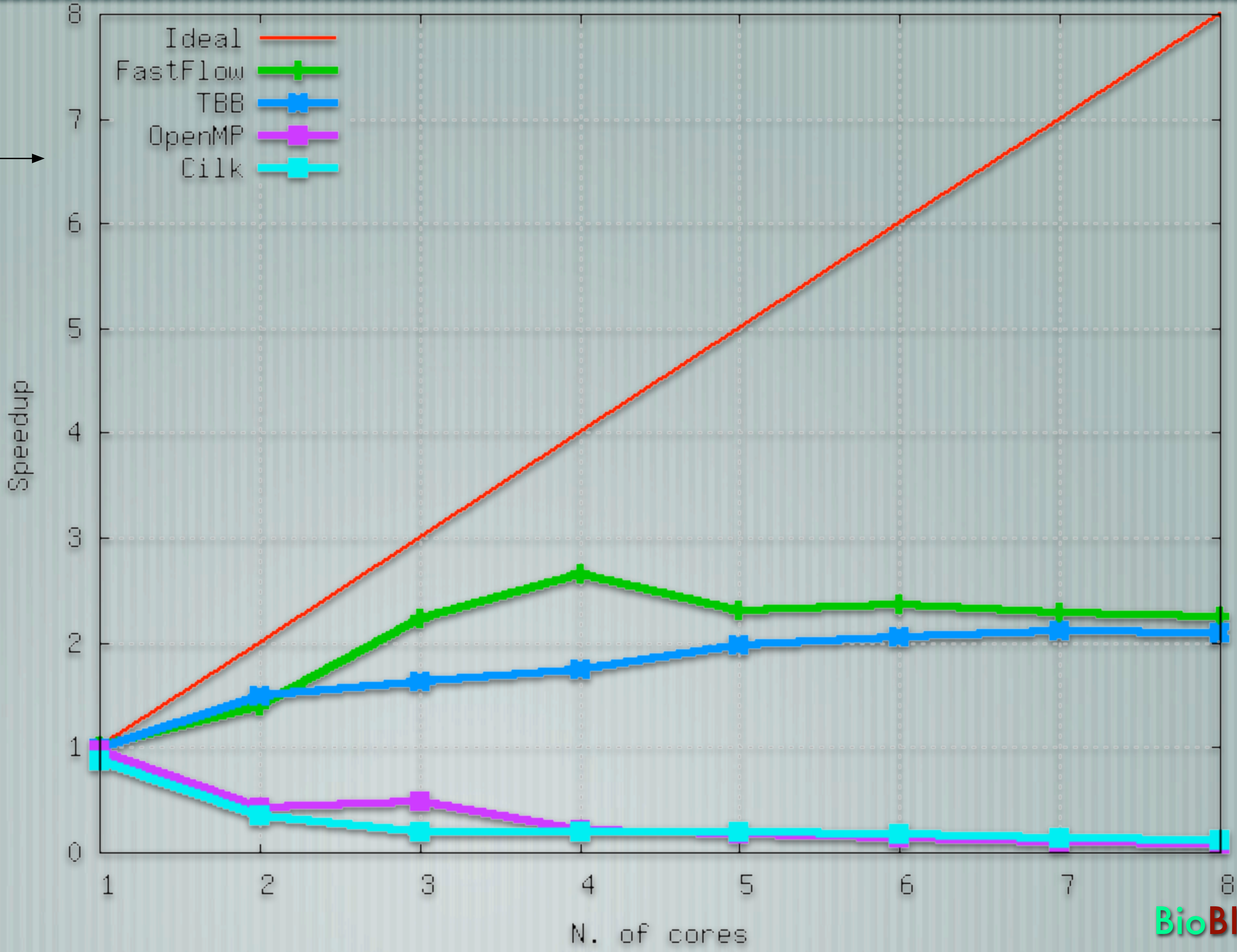
- Common paradigm
  - Model foreach loop and Divide&Conquer
  - Exploit it as a high-order language construct
    - Why should we re-code it from scratch each application?
  - A C++ template factory exploiting highly optimised implementation

E.g. farm with POSIX lock/unlock



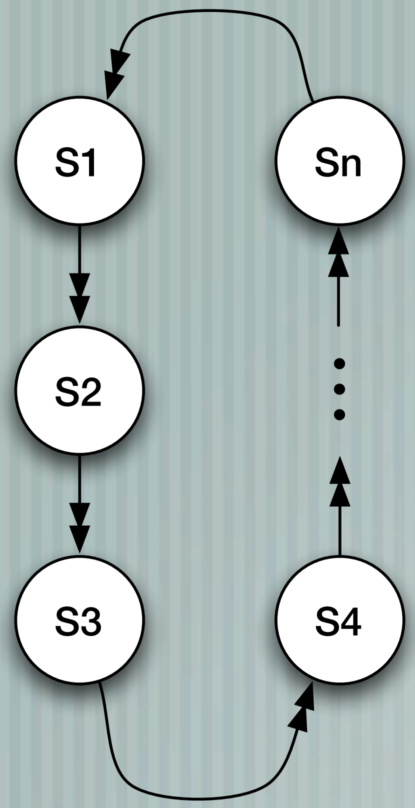
E.g. farm with TBB, OpenMP, Cilk

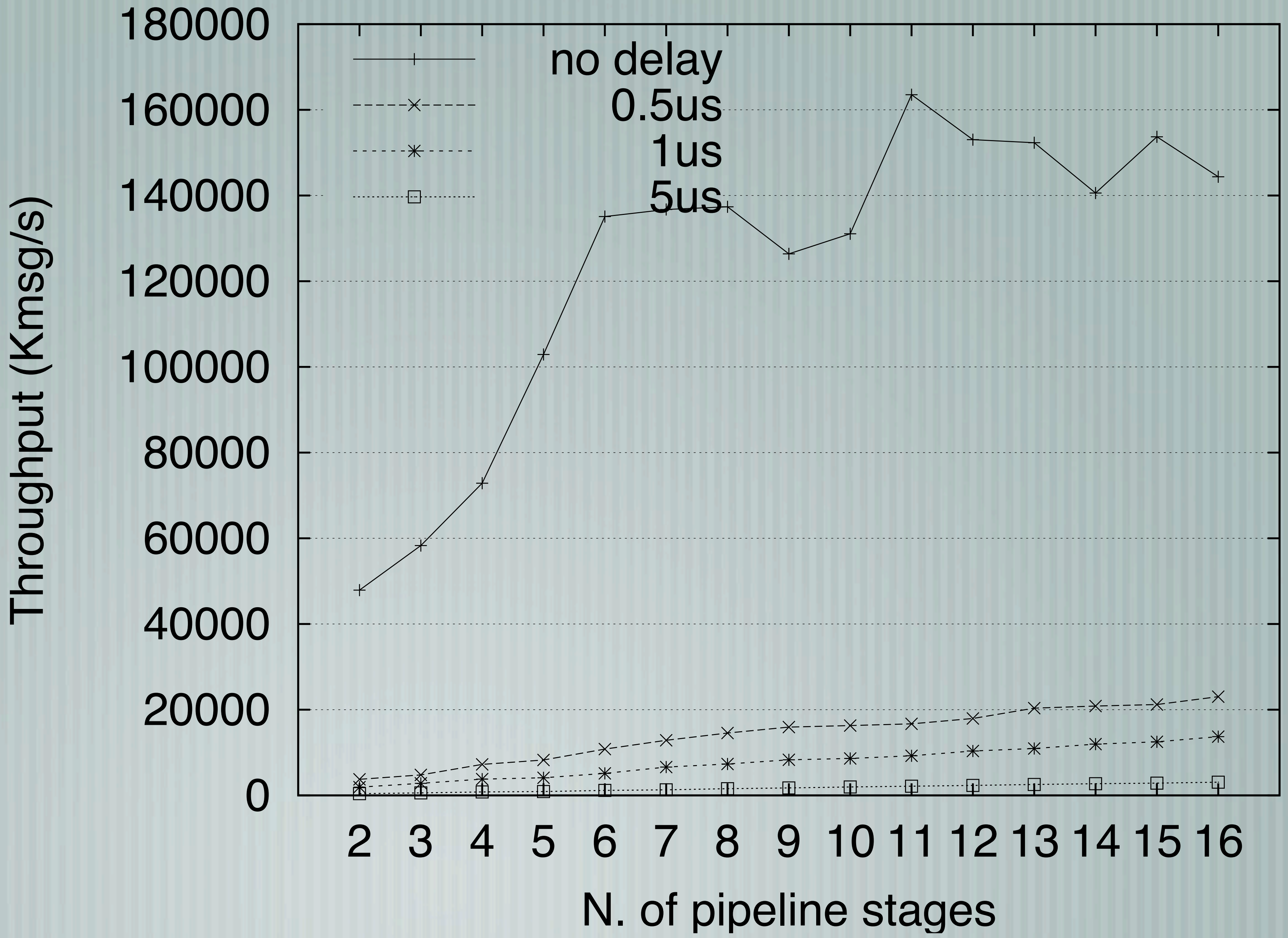




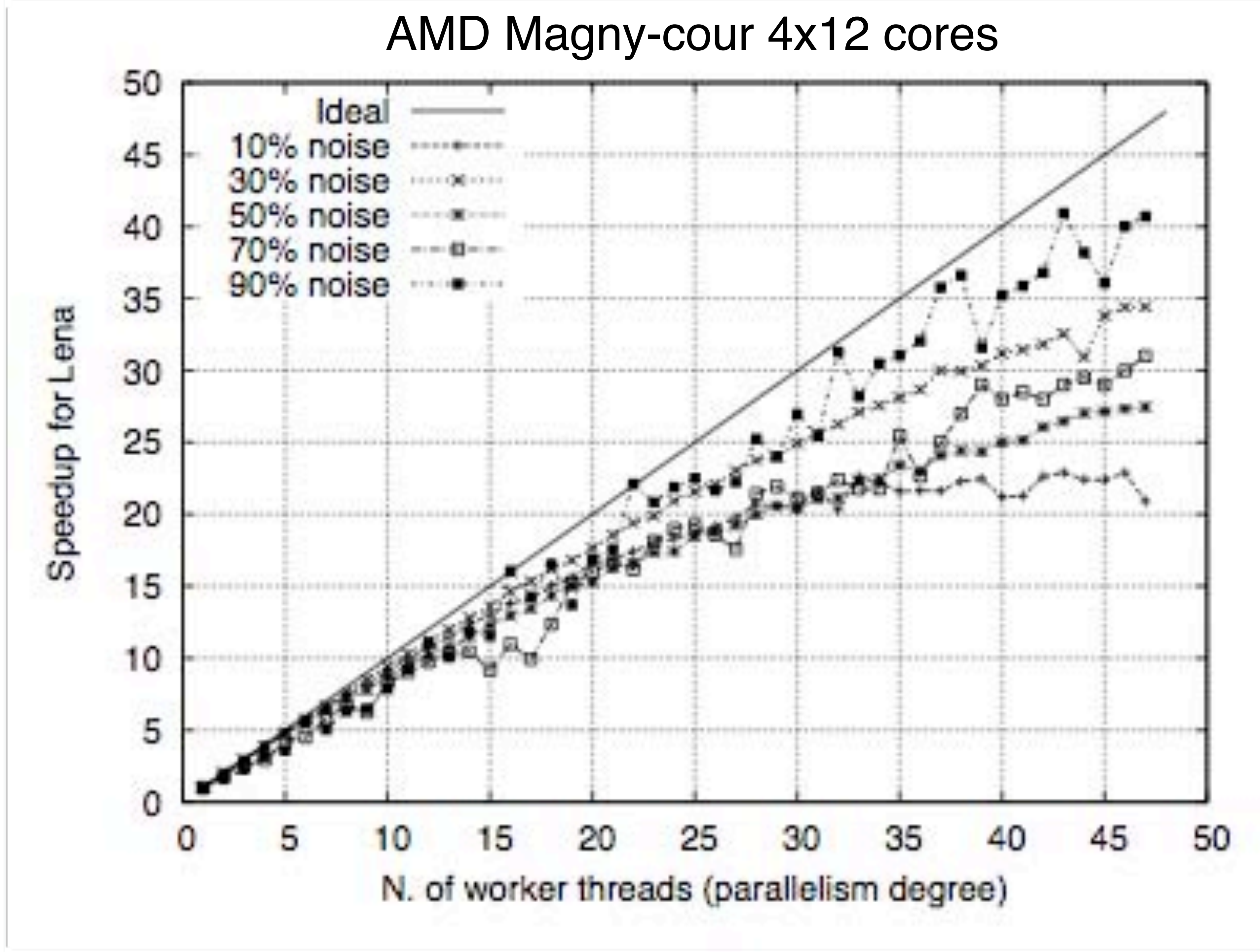
BioBITs

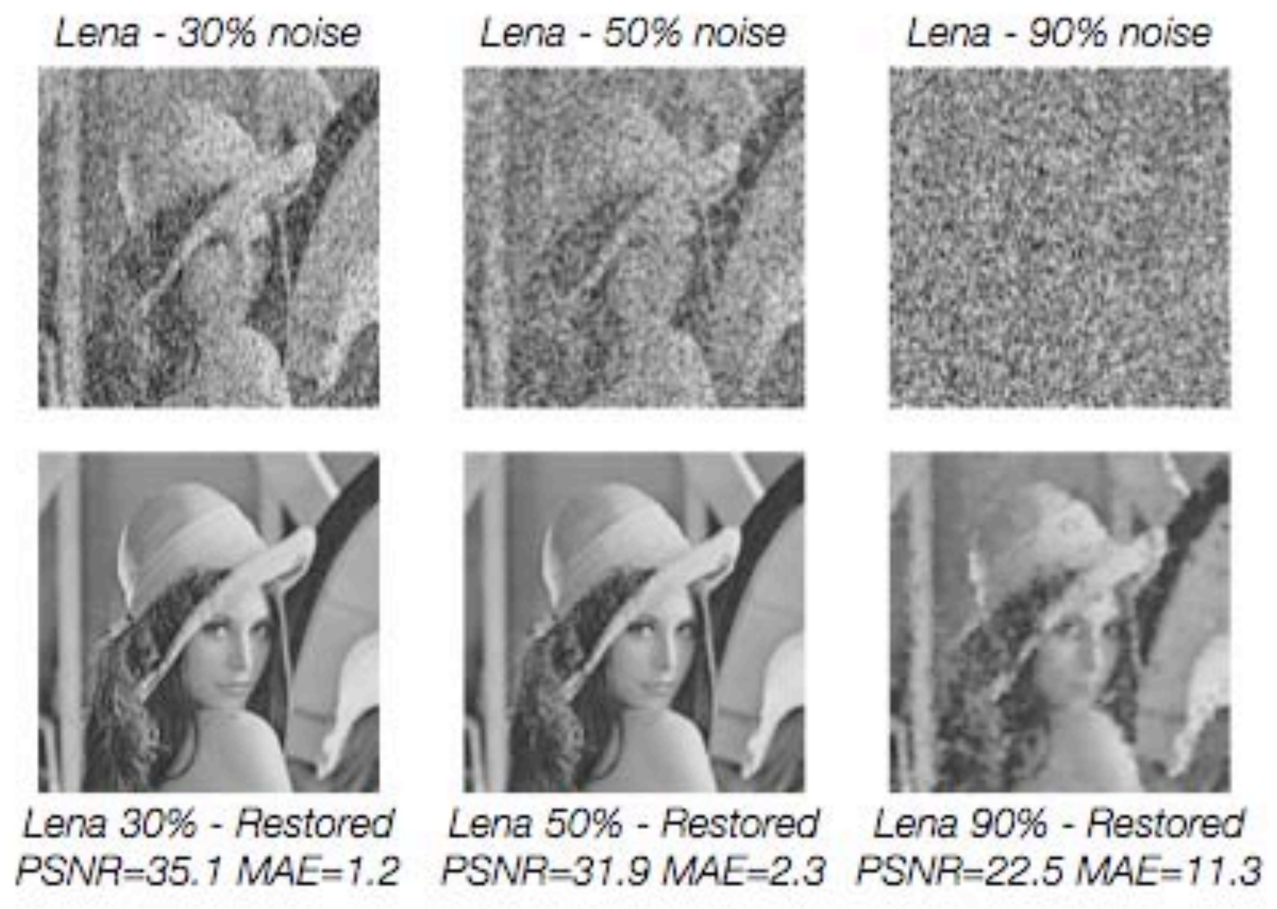
Circular Pipeline (with unbound FIFO)





vs CUDA: Edge-Preserving denoiser



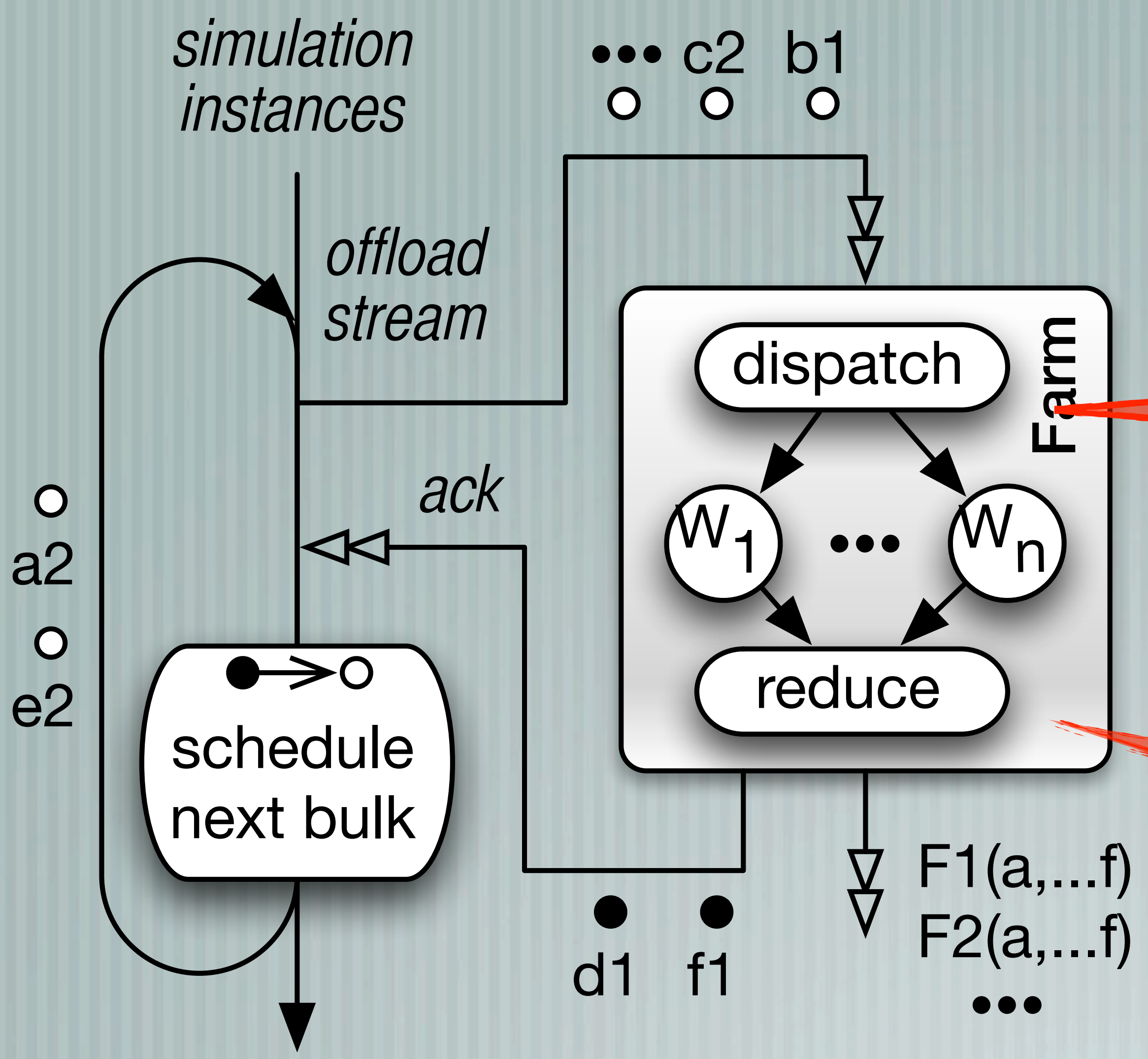


Lena* with 90% of noise is restored in 4 seconds
Next best result in literature for is about 180 seconds

Why efficiency at fine-grain is important?

- It is enabling feature for the scalability of irregular, recursive apps; it reduces the programming and tuning effort
  - Gillespie's sim need about 50 instructions per iteration
- Other issues for Monte Carlo simulators
  - Data stream as a first-class concept (cannot store the whole dataset)
  - Assisted porting methodology (easy migration existing codes)
  - Cache-friendly synchronisation for data streams (performance)
  - Load balancing of irregular workloads (performance, portability)

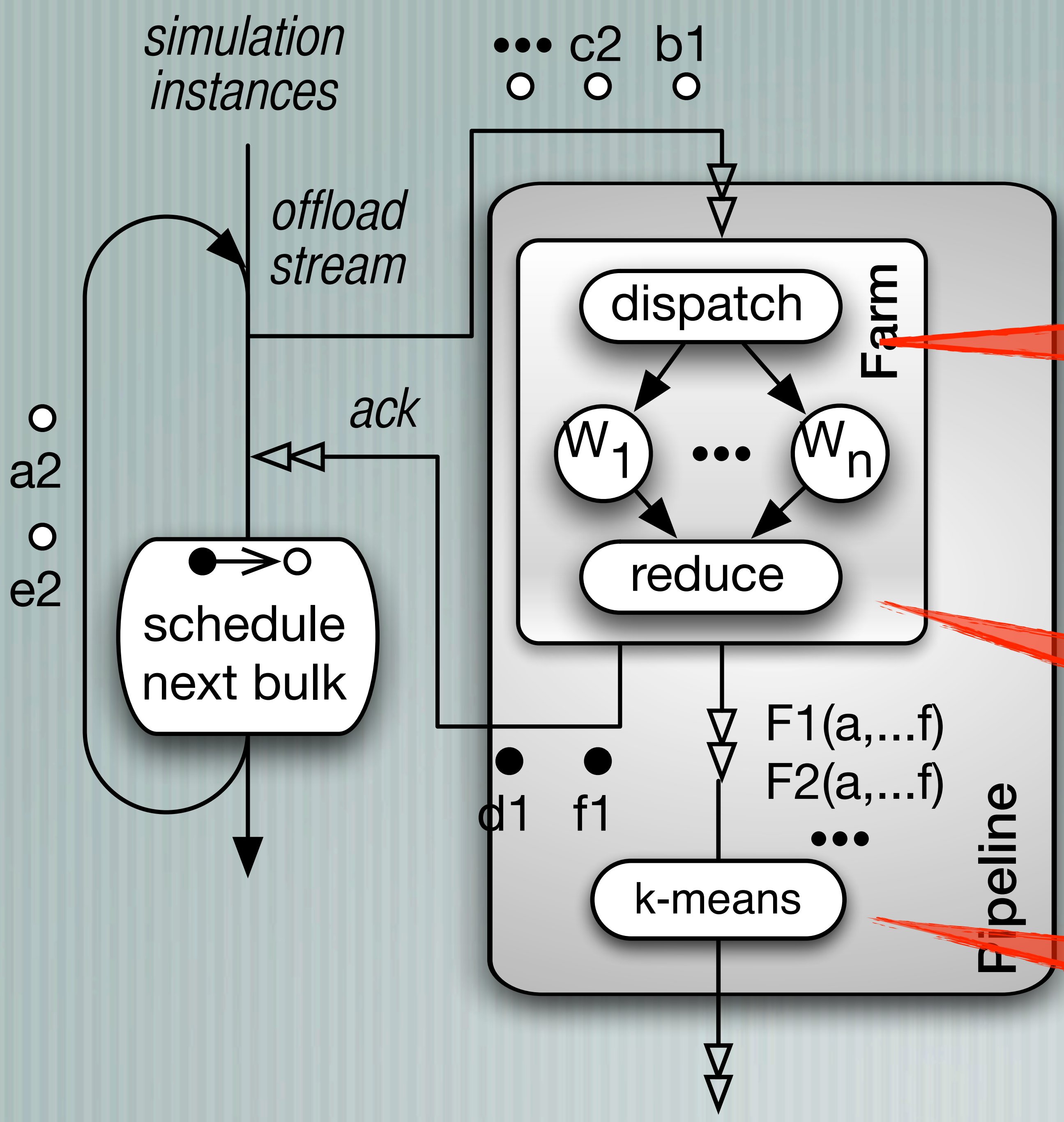
Simulator architecture



The code of the sequential simulator has been copy pasted in the subclass worker
The library manage thread creation, synchronizations, etc.
Tokens are pointers, they enable the access to shared memory

Statistical estimators or any other associative function

Simulator architecture

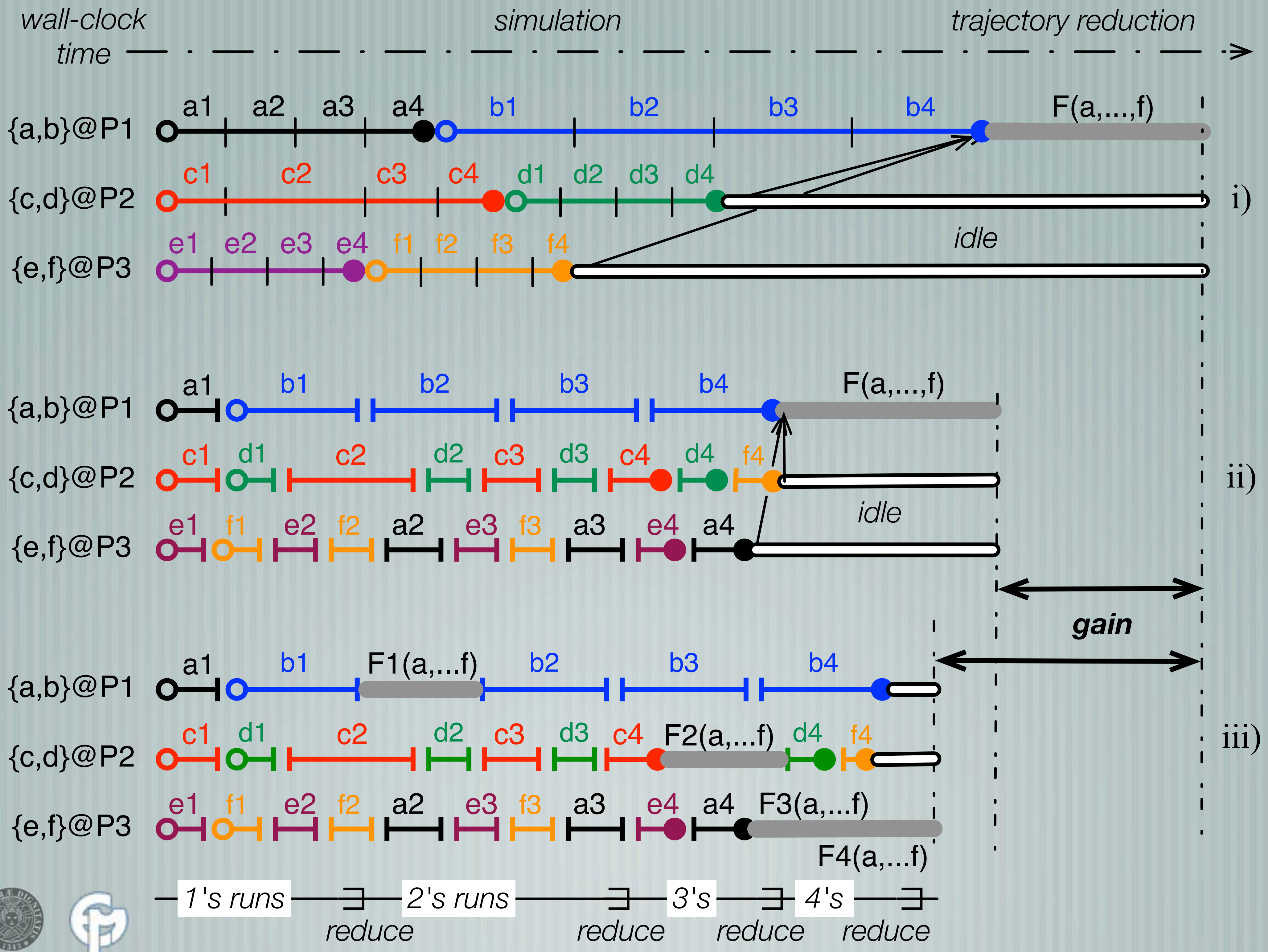


The code of the sequential simulator has been copy pasted in the subclass worker
The library manage thread creation, synchronizations, etc.
Tokens are pointers, they enable the access to shared memory

Statistical estimators or any other associative function

Patterns can be composed to extend simulator features

Three scheduling policies: i) Round Robin ii) Auto-balancing iii) Auto-balancing with pipelined reduction



FastFlow accelerator

```
// Original code
#define N 1024
long A[N][N],B[N][N],C[N][N];        ①
int main() {
  // < init A,B,C>

  for(int i=0;i<N;++i) {             ②
    for(int j=0;j<N;++j) {

      int _C=0;
      for(int k=0;k<N;++k)           ❸
        _C += A[i][k]*B[k][j ];
      C[i][ j]=_C;

    }                                ④
  }
}                                    ⑤
```

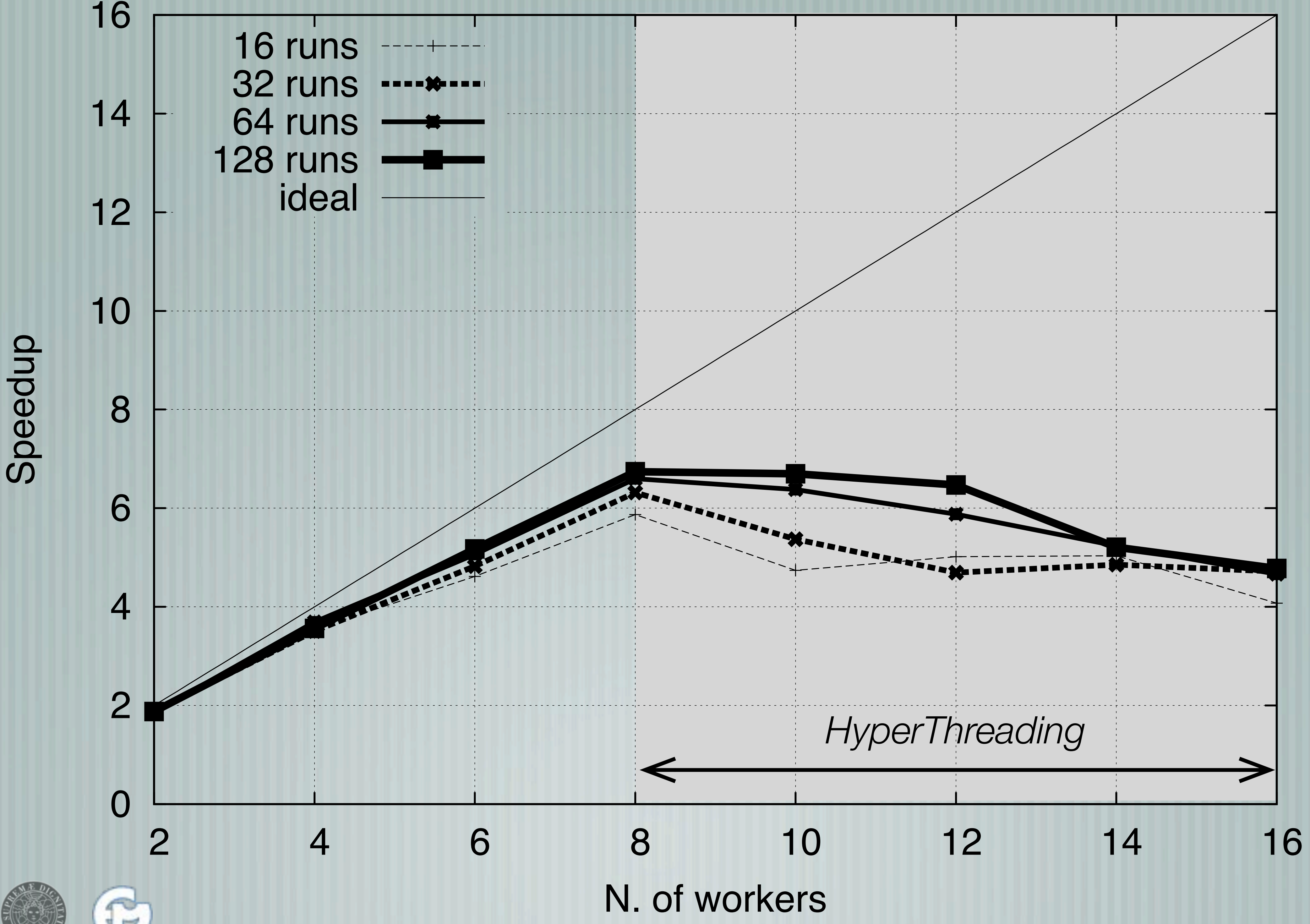
```
    // FastFlow accelerated code
    #define N 1024
①   long A[N][N],B[N][N],C[N][N];
    int main() {
      // < init A,B,C>

      ff :: ff_farm<> farm(true /* accel */);
      std :: vector<ff :: ff_node *> w;
      for(int i=0;i<PAR_DEGREE;++i)
        w.push_back(new Worker);
      farm.add_workers(w);
      farm.run_then_freeze();

      for (int i=0;i<N;i++) {
②       for(int j=0;j<N;++j) {
          task_t * task = new task_t(i,j);
          farm.offload(task);               ➔
        }
④     }
      farm.offload((void *)ff :: FF_EOS);
      farm.wait();  // Here join
⑤   }

    // Includes
    struct task_t {
      task_t(int i,int j):i(i),j(j) {}
      int i; int j;};

    class Worker: public ff :: ff_node {
    public: // Offload target service
➔     void * svc(void *task) {
        task_t * t = (task_t *)task;
        int _C=0;
❸       for(int k=0;k<N;++k)
          _C += A[t−>i][k]*B[k][t−>j];
        C[t−>i][t−>j] = _C;
        delete t;
        return GO_ON;
        }
    };
```

Demo: A, B, C (grass, sheep, wolves)

```
%model "wolf vs sheep vs grass - all with
strange alimentary habits"
%alphabet
A B C iA iB iC

%rules
{T} C ({IN} ~x | ~X) ~Y >>>[0.01]>>> ({IN}
~x|iC ~X) ~Y %%
{T} ({IN} ~x|iA ~X) ~Y >>>[0.01]>>> A ({IN}
~x |~X) ~Y %%
{T} ({IN} ~x|iB ~X) ~Y >>>[0.01]>>> B ({IN}
~x |~X) ~Y %%

{T} A ~X >>>[1.0]>>> A A ~X %%
{T} B ~X >>>[1.0]>>> B B ~X %%
{T} C ~X >>>[1.0]>>> C C ~X %%
{T} A A ~X >>>[0.0015]>>> ~X %%
{T} B B ~X >>>[0.0015]>>> ~X %%
{T} C C ~X >>>[0.0015]>>> ~X %%
{T} A B ~X >>>[0.002]>>> ~X %%
{T} A C ~X >>>[0.0015]>>> ~X %%
{T} B C ~X >>>[0.002]>>> ~X %%
{IN} iA ~X >>>[1.0]>>> iA iA ~X %%
{IN} iB ~X >>>[1.0]>>> iB iB ~X %%
{IN} iC ~X >>>[1.0]>>> iC iC ~X %%
{IN} iA iA ~X >>>[0.0015]>>> ~X %%
{IN} iB iB ~X >>>[0.0015]>>> ~X %%
{IN} iC iC ~X >>>[0.0015]>>> ~X %%
{IN} iA iB ~X >>>[0.002]>>> ~X %%
{IN} iA iC ~X >>>[0.0015]>>> ~X %%
{IN} iB iC ~X >>>[0.002]>>> ~X %%


%term
2*C ({IN} |2*iA 2*iB)

%monitors
"external A": {T} A%%
"external B": {T} B%%
"external C": {T} C%%
"internal A": {IN} iA%%
"internal B": {IN} iB%%
"internal C": {IN} iC%%
```

BioBITs

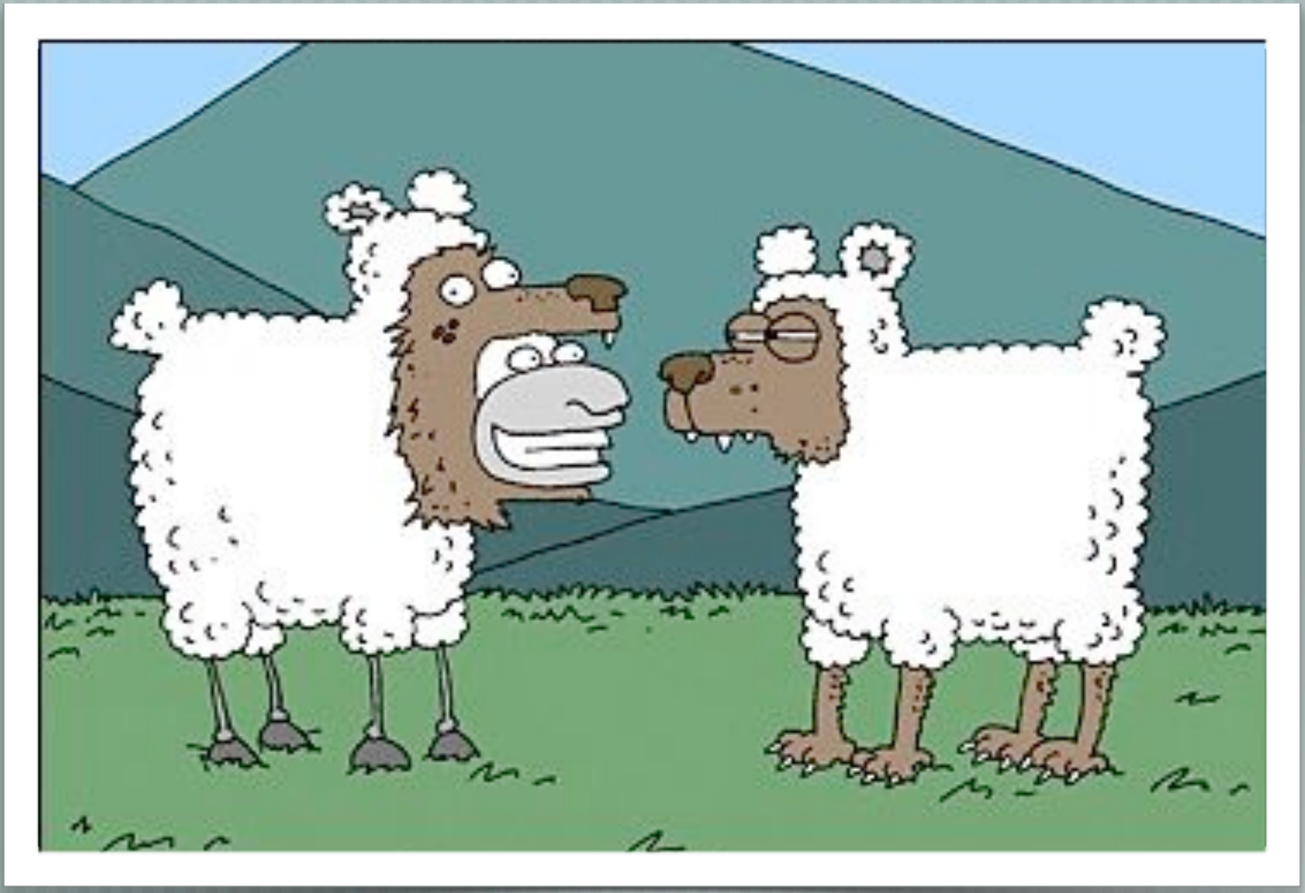
Speedup (Intel core 2x4-core)



Conclusion

- Streaming is effective on multicore is properly programmed
  - efficient: lock-free, fence-free
  - high-level, easy to exploit: pattern-based programming (skeleton-based)
- Can be used to build efficient simulators
  - Not only for Bio
  - More effective than other solutions GPGPU, SIMD (see the paper)
- Several development ongoing
  - Online automatic classification, online singular trajectory mining, ...

Beware of false prophets who come to you in sheep's clothing but inwardly are savage wolves (Matthew 7:15)



Things got weird when the wolf in the sheep's clothing happened across the sheep in wolf-in-sheep's-clothing clothing

http://mc-fastflow.sourceforge.net

http://sourceforge.net/projects/cwcsimulator/

Thank you