Efficient streaming applications on multi-core with FastFlow: the biosequence alignment test-bed

Marco Aldinucci
Computer Science Dept. - University of Torino - Italy

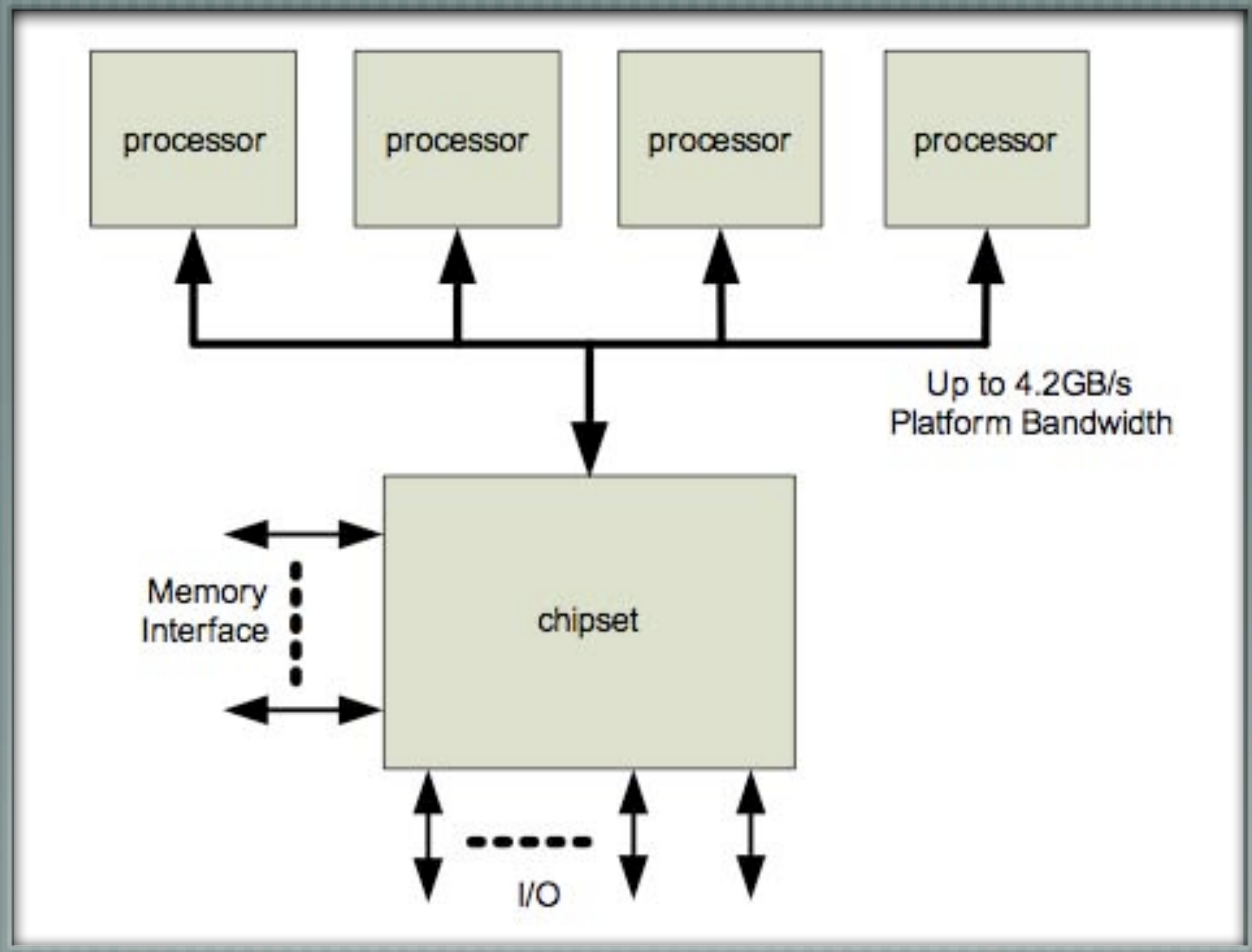
Marco Danelutto, Massimiliano Meneghin, Massimo Torquti
Computer Science Dept. - University of Pisa - Italy

Peter Kilpatrick
Computer Science Dept. - Queen's University Belfast - U.K.

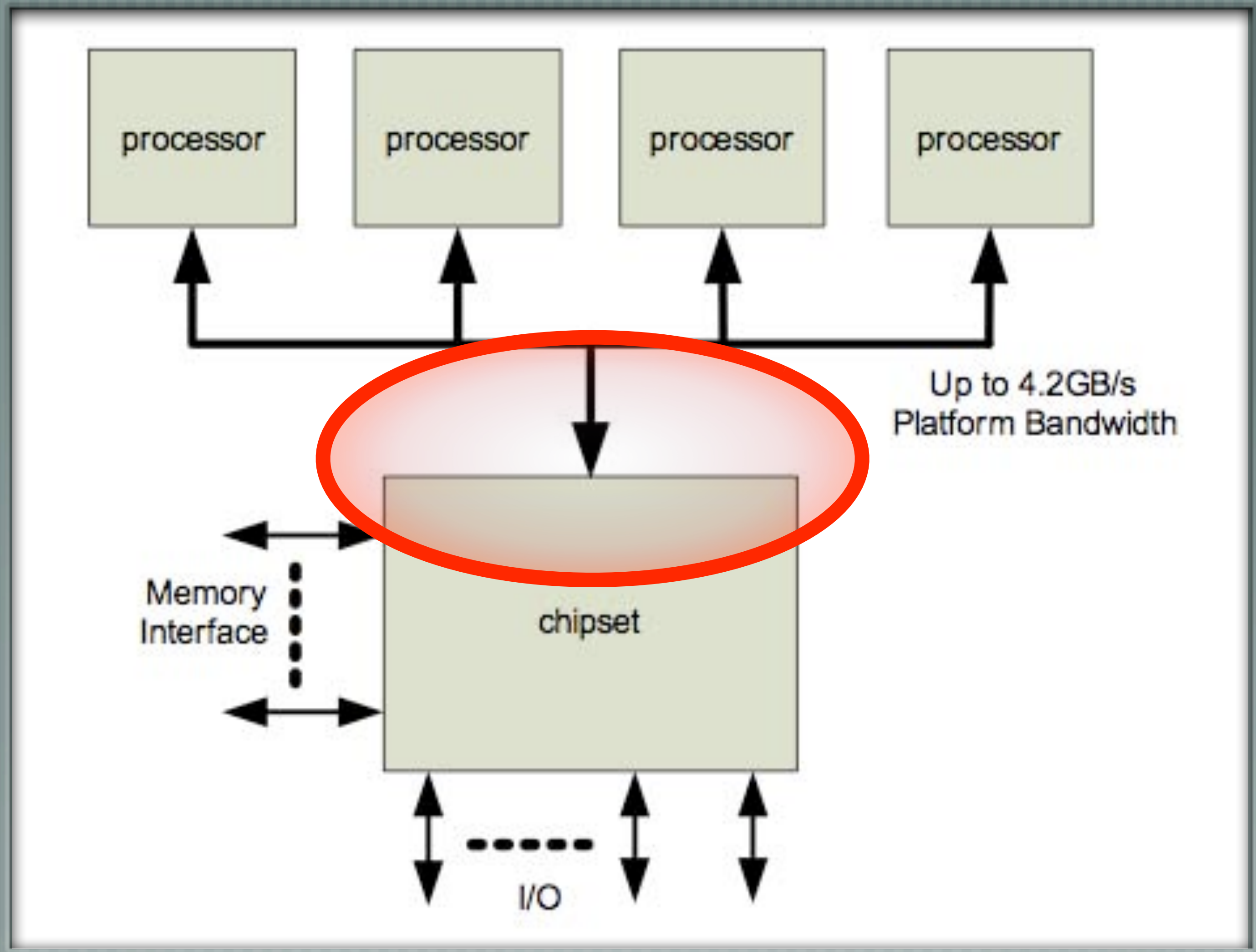
ParCo 2009 - Sep. 1st - Lyon - France

Outline

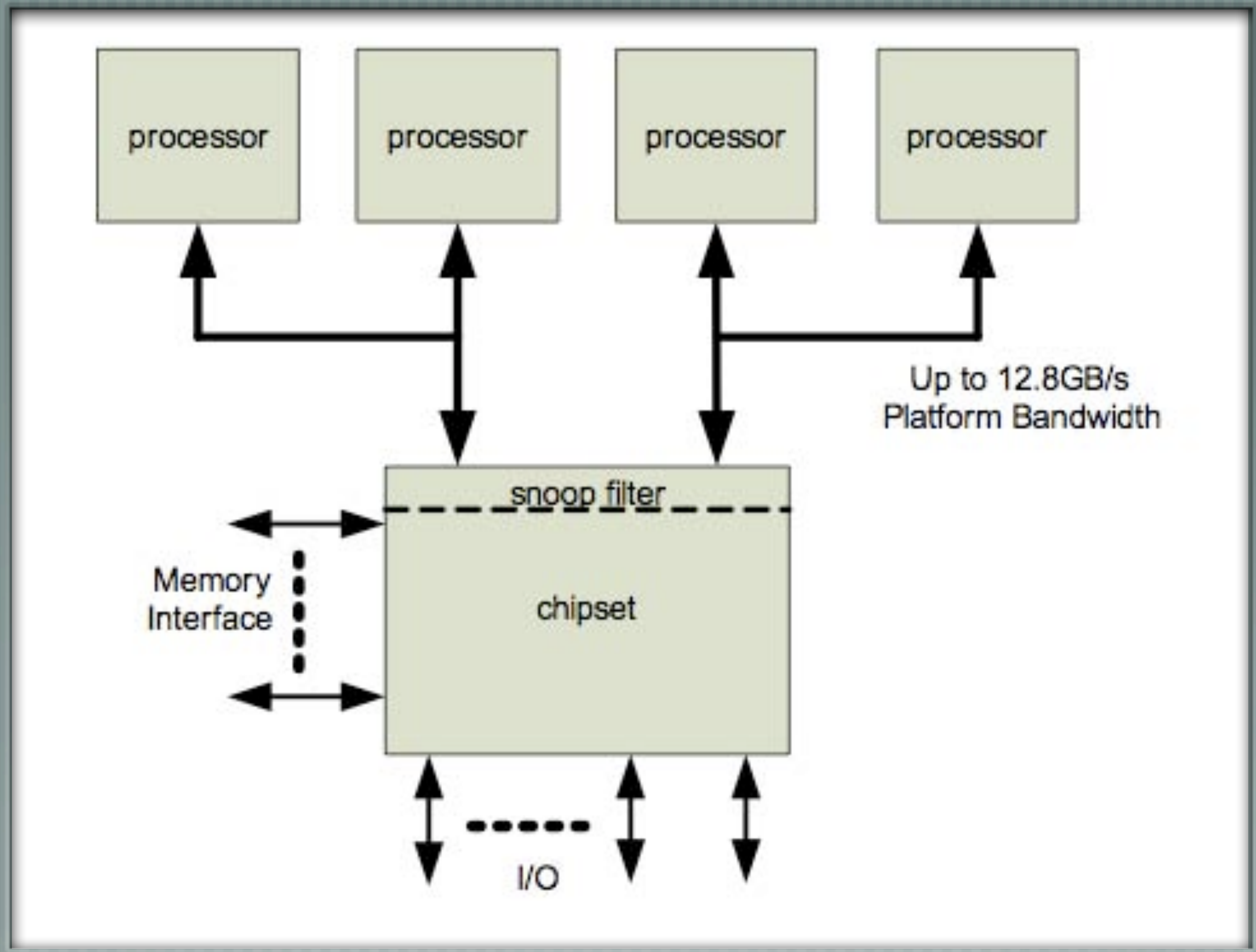
- Motivation
  - Commodity architecture evolution
  - Efficiency for fine-grained computation
  - POSIX thread evaluation
- FastFlow
  - Architecture
  - Implementation
- Experimental results
  - Micro-benchmarks
  - Real-world App: the Smith-Waterman sequence alignment application
- Conclusion, future works, and surprise dessert (before lunch)



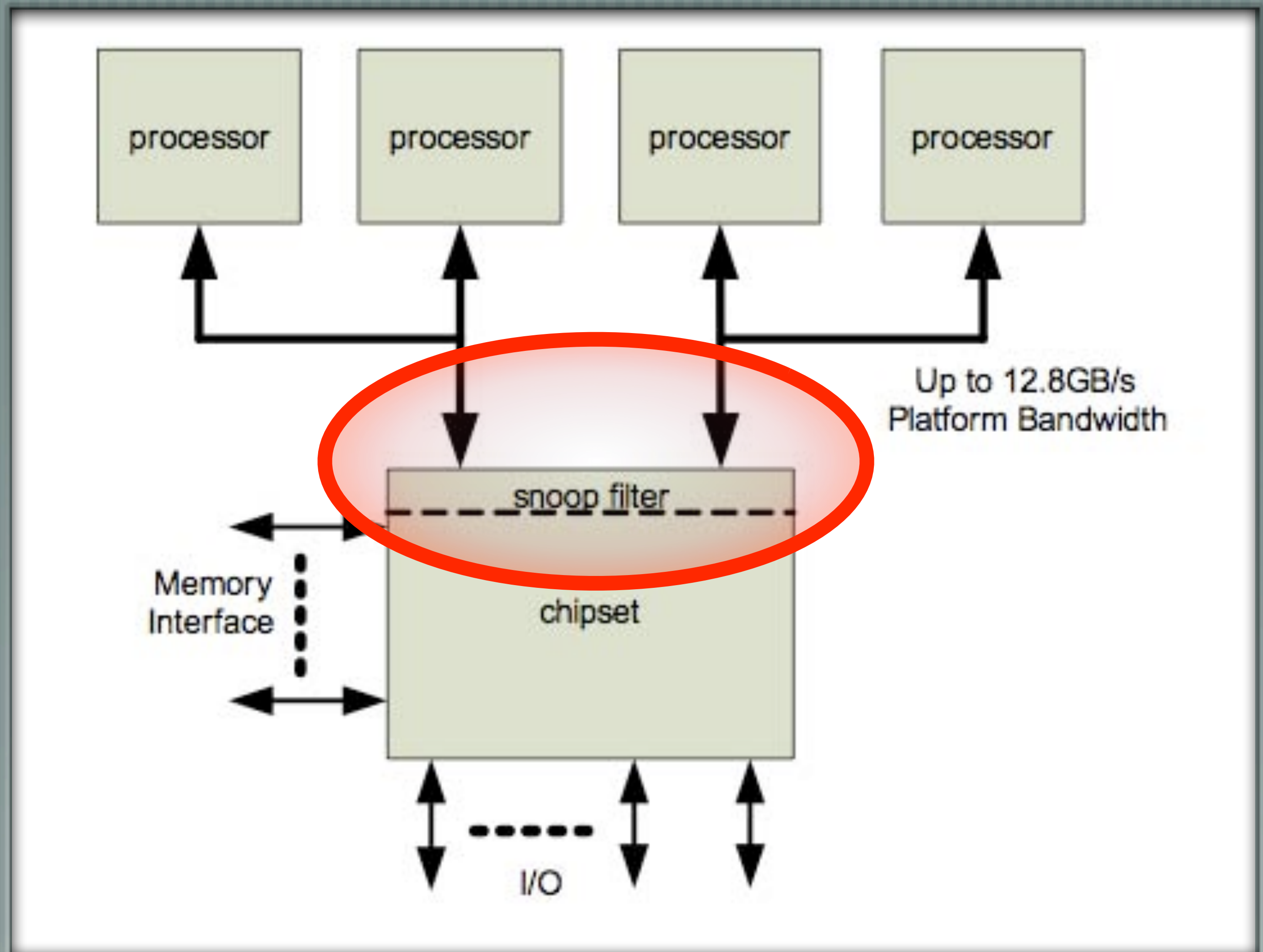
[< 2004] Shared Font-Side Bus (Centralized Snooping)



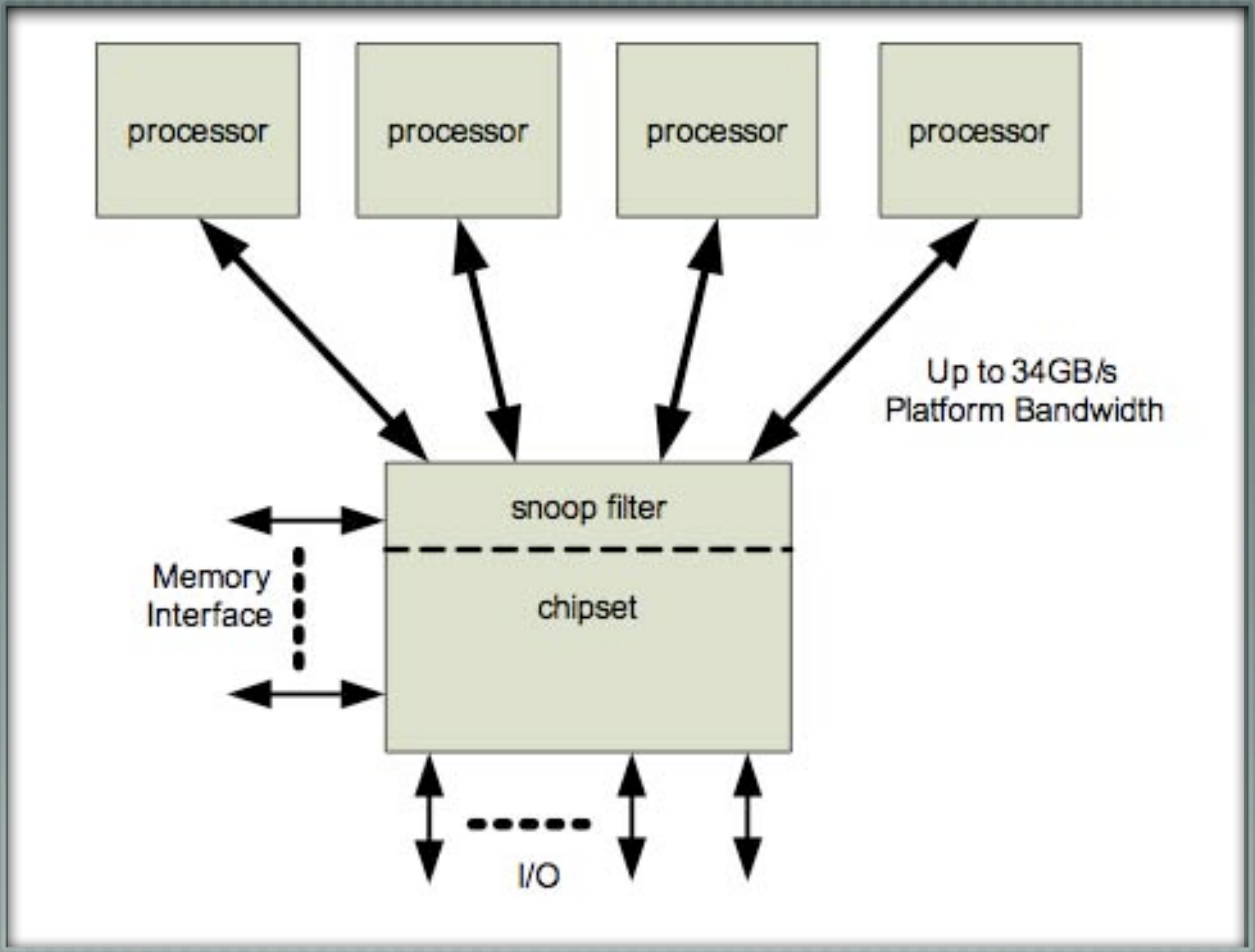
[< 2004] Shared Font-Side Bus (Centralized Snooping)



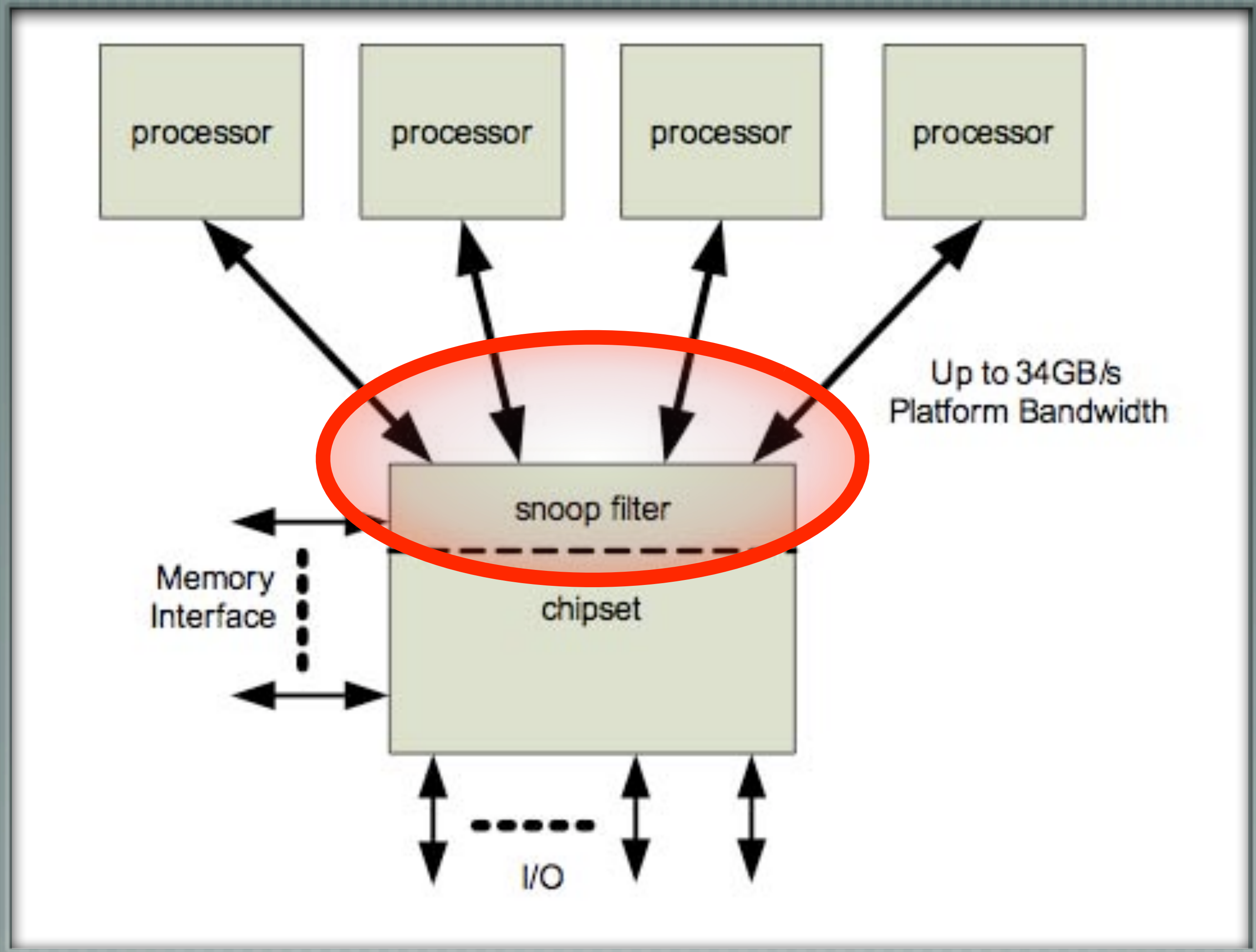
[2005] Dual Independent Buses (Centralized Snooping)



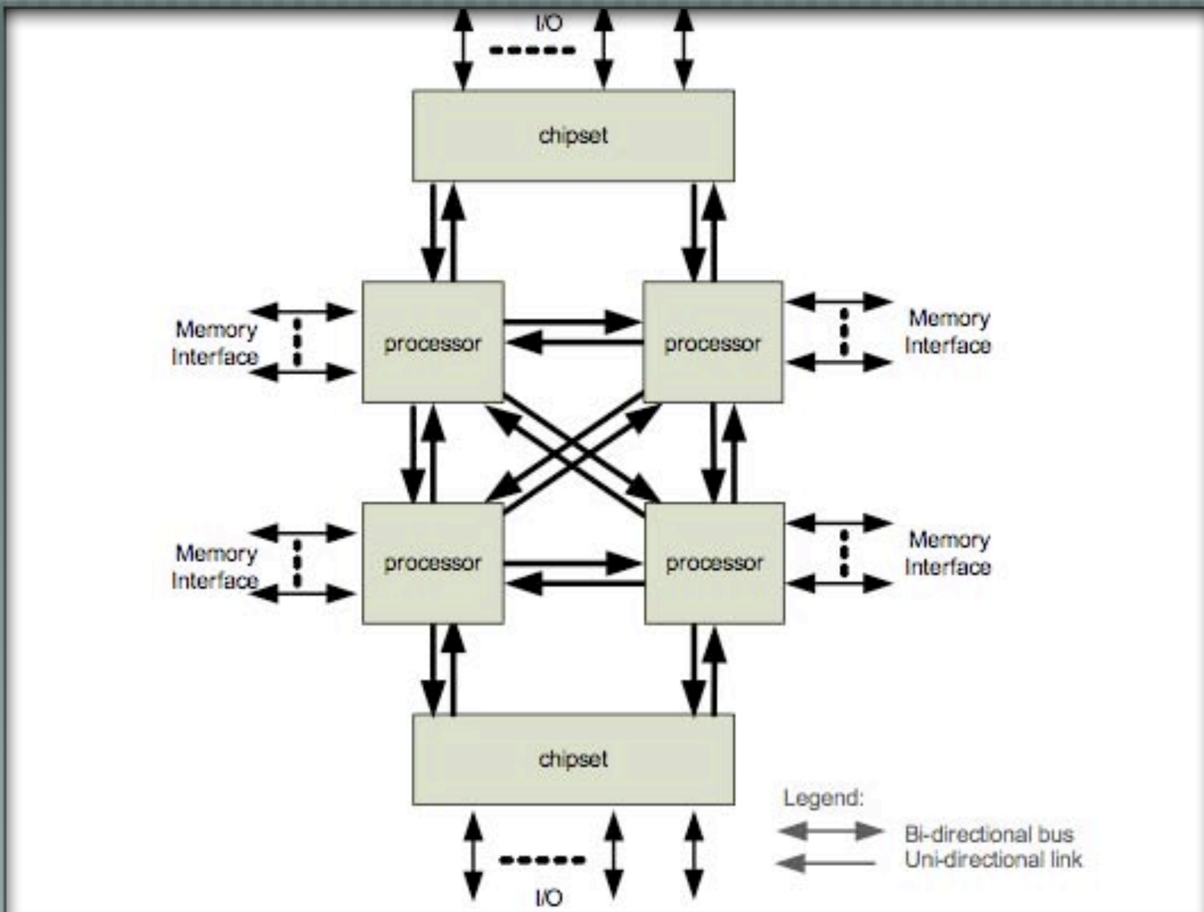
[2005] Dual Independent Buses (Centralized Snooping)

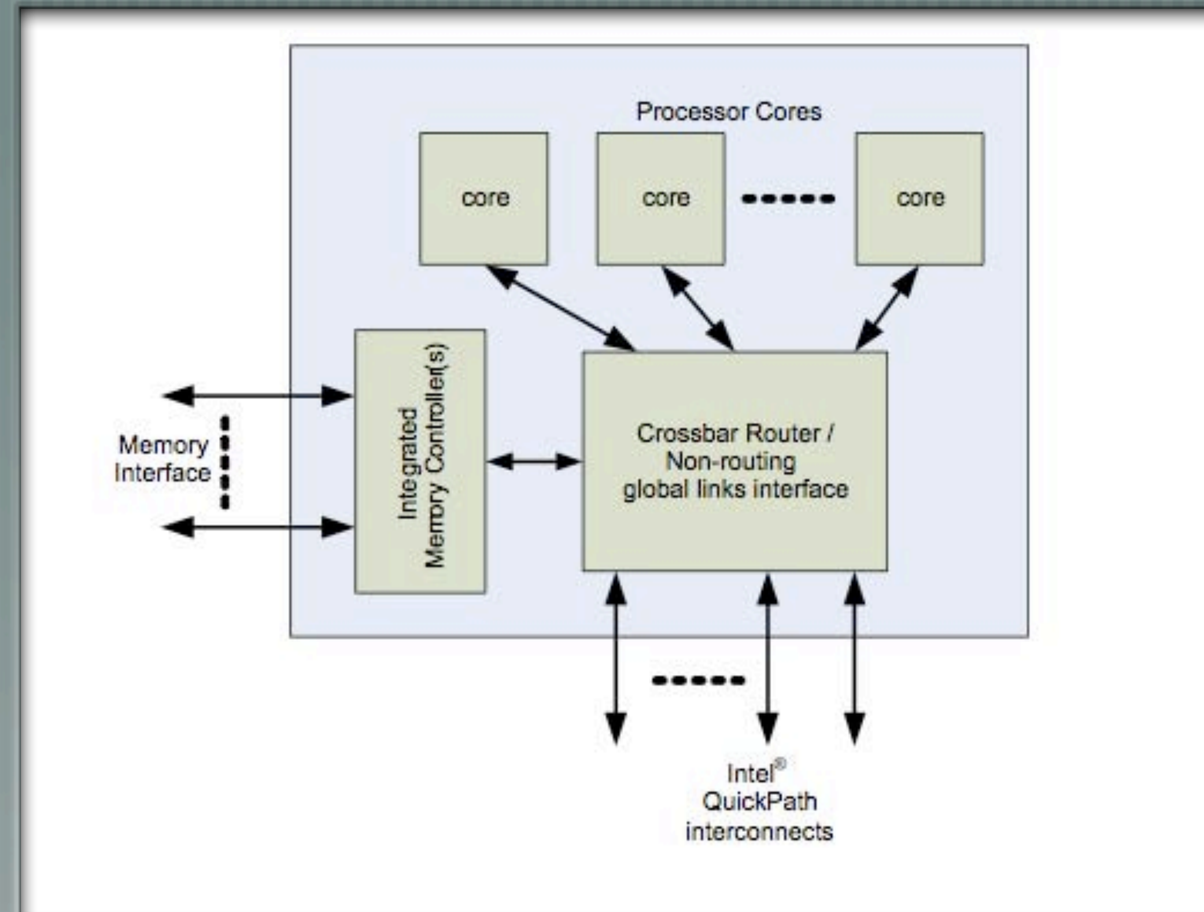


[2007] Dedicated High-Speed Interconnects (Centralized Snooping)

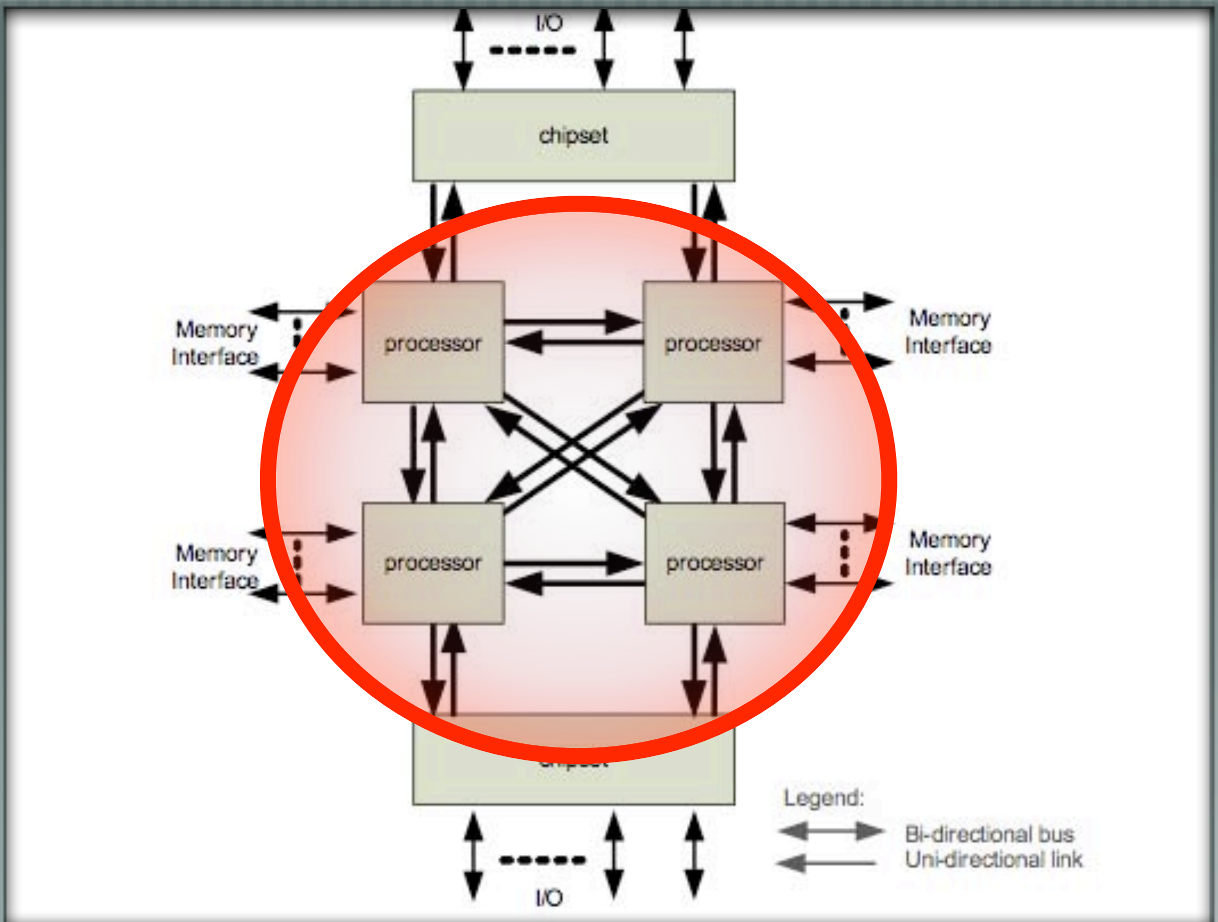


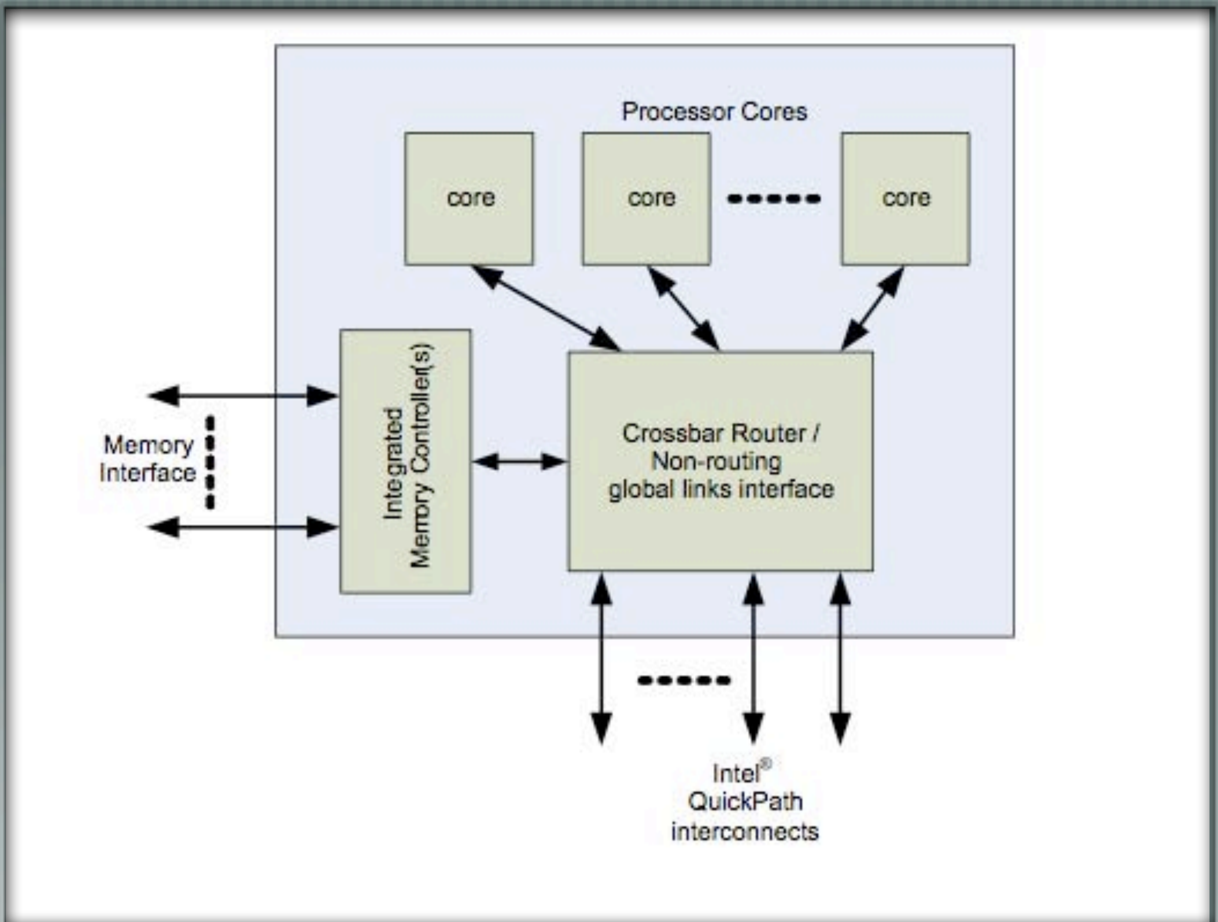
[2007] Dedicated High-Speed Interconnects (Centralized Snooping)





[2009] QuickPath (MESI-F Directory Coherence)





[2009] QuickPath (MESI-F Directory Coherence)

This and next generation SCM

- Exploit cache coherence
  - and it is likely to happens also in the next future
- Memory fences are expensive
  - Increasing core count will make it worse
  - Atomic operations does not solve the problem (still fences)
- Fine-grained parallelism is off-limits
  - I/O bound problems, High-throughput, Streaming, Irregular DP problems
  - Automatic and assisted parallelization

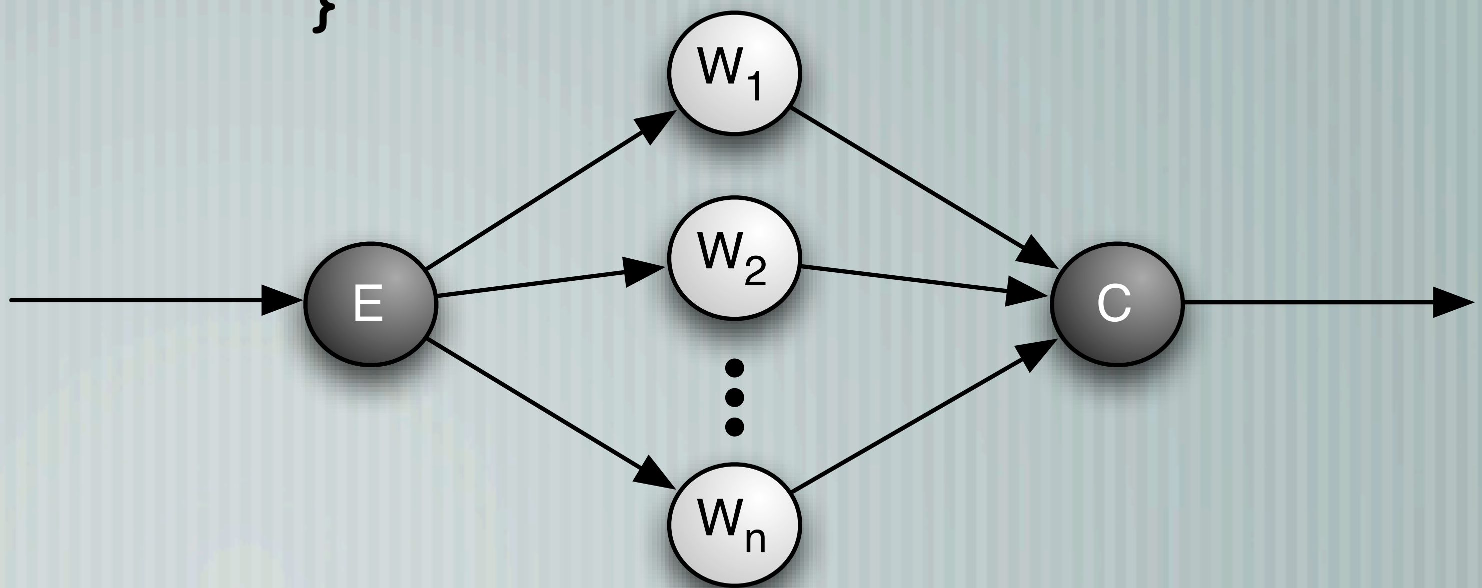
Micro-benchmarks: farm of tasks

Used to implement: parameter sweeping, master-worker, etc.

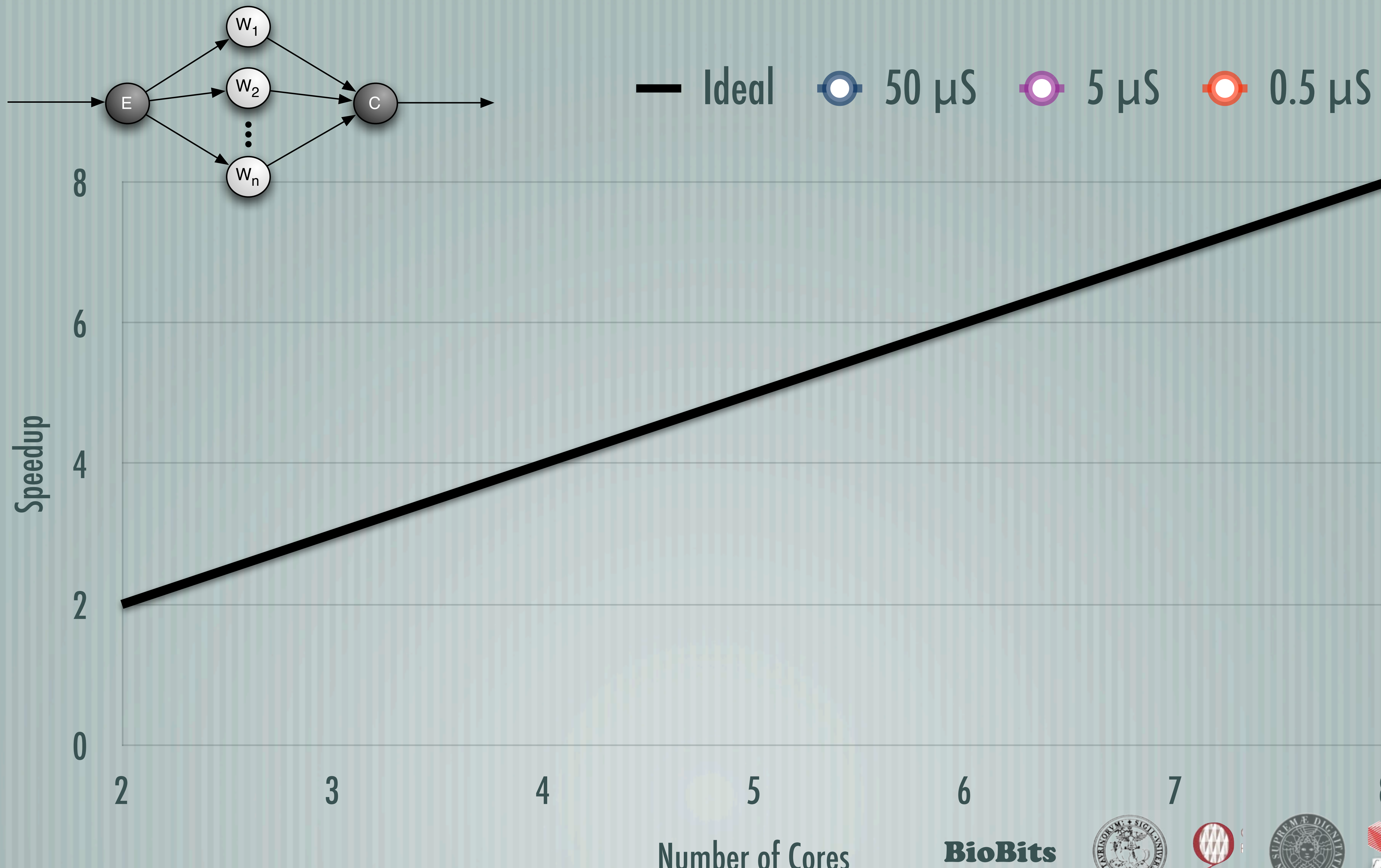
```
void Emitter () {
  for ( i =0; i <streamLen;++i){
    task = create_task ();
    queue=SELECT_WORKER_QUEUE();
    queue ->PUSH(task);
  }
}
```

```
int main () {
  spawn_thread( Emitter ) ;
  for ( i =0; i <nworkers;++i){
    spawn_thread(Worker);
  }
  wait_end () ;
}
```

```
void Worker() {
  while (!end_of_stream){
  myqueue ->POP(&task);
  do_work(task) ;
  }
}
```



Using POSIX lock/unlock queues

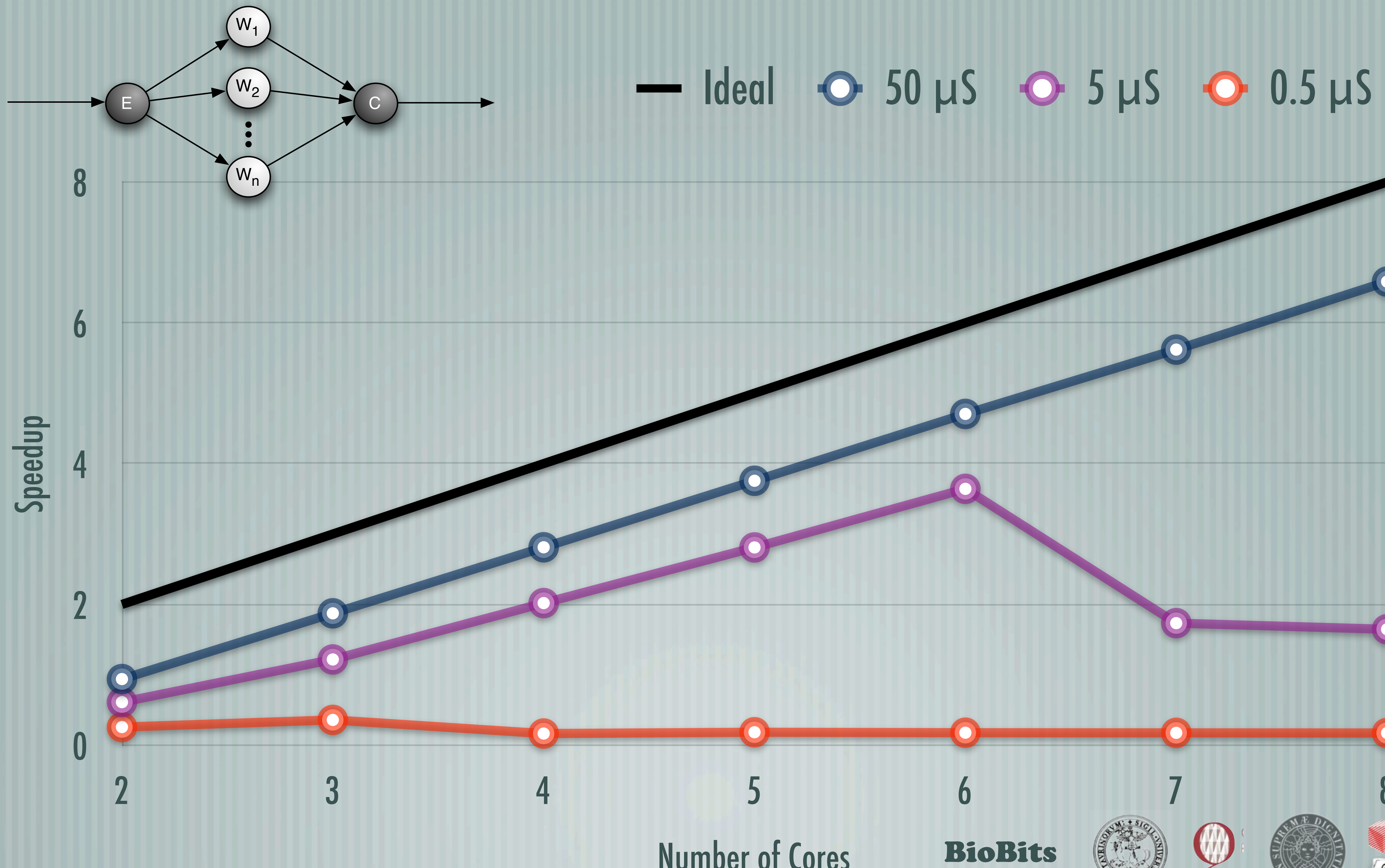


BioBits

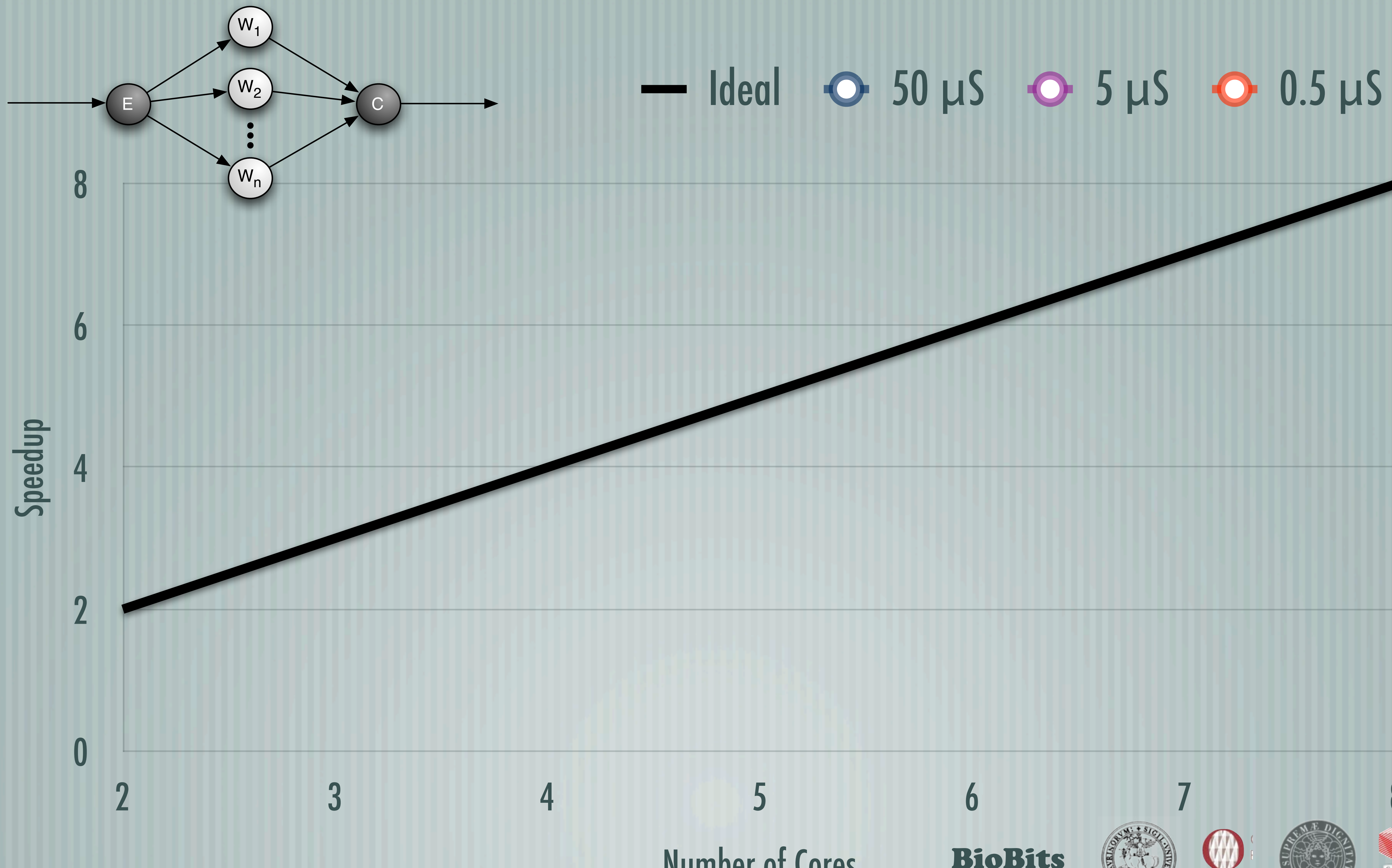
ERCIM

CoreGRID

Using POSIX lock/unlock queues



Using CompareAndSwap queues

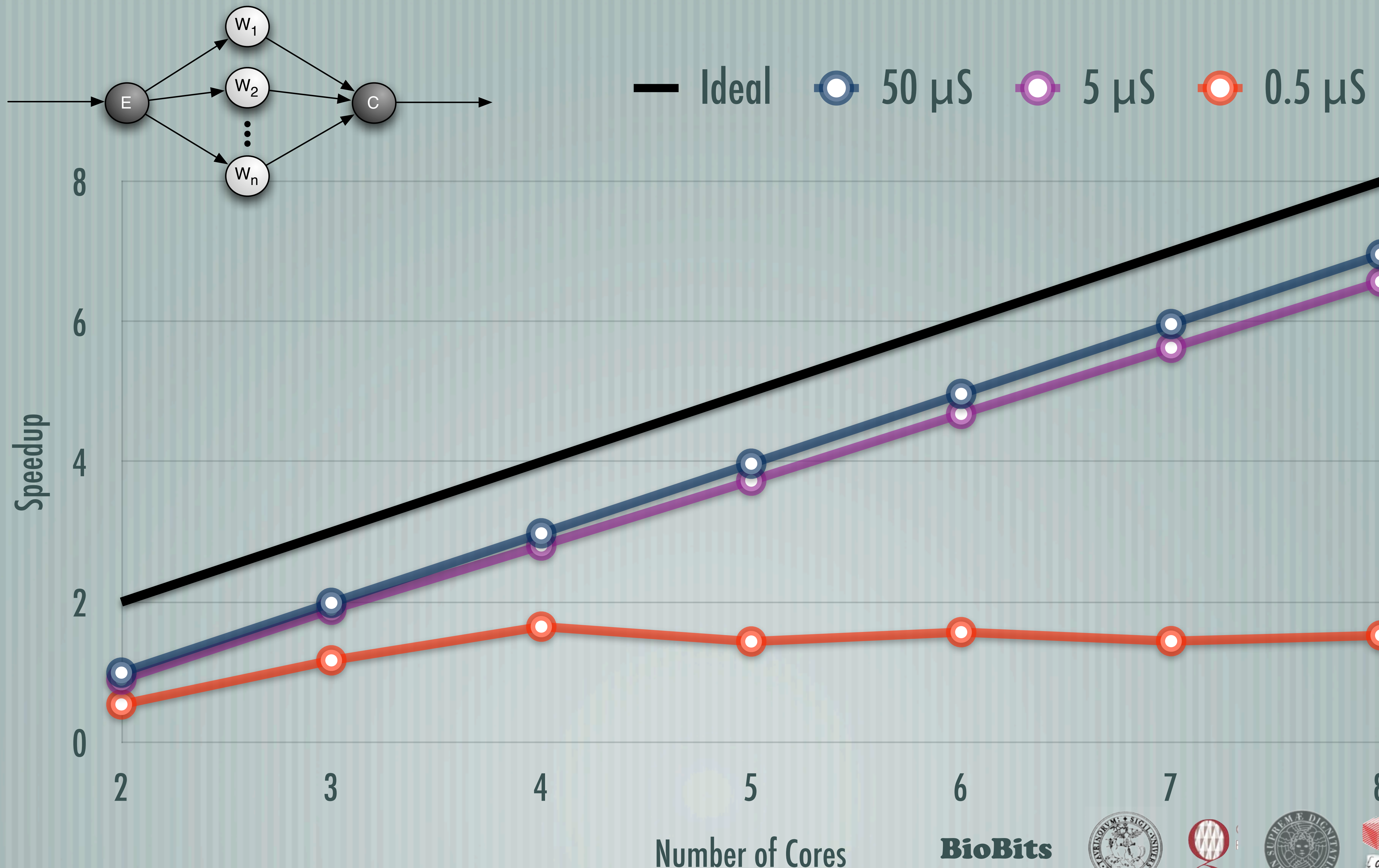


BioBits

ERCIM

CoreGRID

Using CompareAndSwap queues



Evaluation

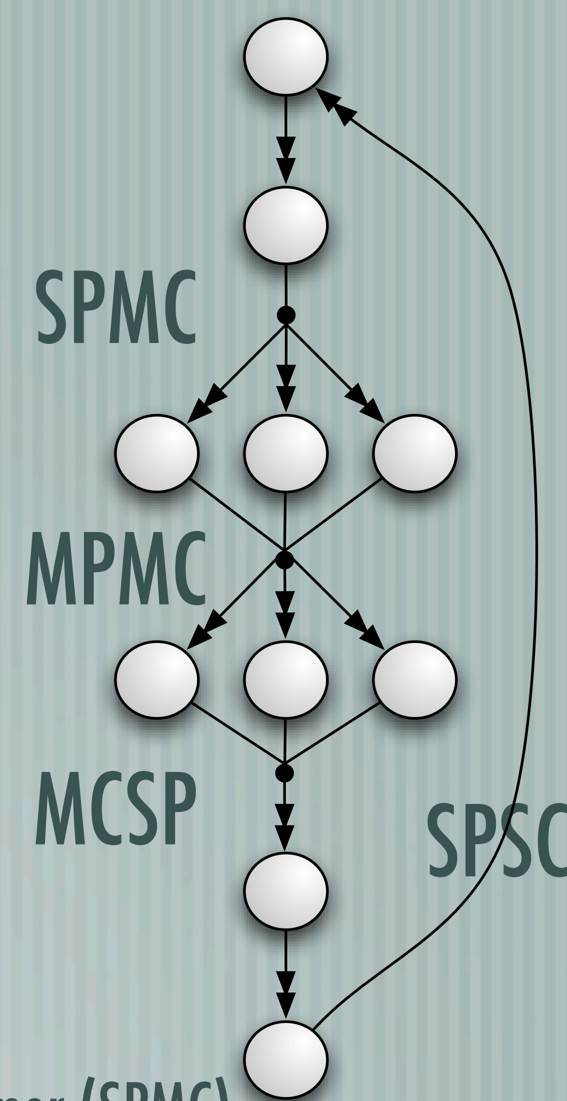
- Poor performance for fine-grained computations
- Memory fences seriously affect the performance

What about avoiding fences in SCM?

- Highly-level semantics matters!
  - DP paradigms entail data bidirectional data exchange among cores
    - Cache reconciliation can be made faster but not avoided
  - Task Parallel, Streaming, Systolic usually result in a one-way data flow
    - Is cache coherency really strictly needed?
    - Well described by a data flowing graphs (streaming networks)

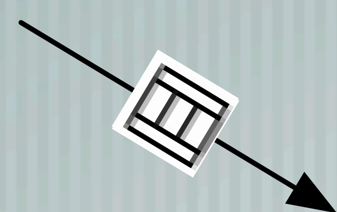
Streaming Networks

- A Streaming Network can be easily build
  - POSIX (or other) threads
  - Asynchronous channels
  - But exploiting a global address space
    - Threads can still share the memory using locks
- Asynchronous channels
  - Thread lifecycle control + FIFO Queue
    - Queue: Single Producer Single Consumer (SPSC), Single Producer Multiple Consumer (SPMC), Multiple Producer Single Consumer (MPSC), Multiple Producer Multiple Consumer (MPMC)
    - Lifecycle: ready - active waiting (yield + over-provisioning)



Queues: state of the art

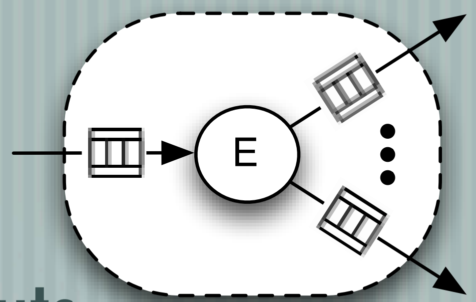
- MPMC
  - Dozen of “lock-free” (and wait-free) proposal
  - The quality is usually measured with number of atomic operations (CAS)
    - CAS ≥ 1
- SPSC



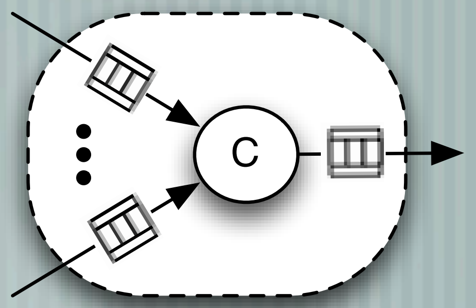
  - lock-free, fence-free
    - J. Giacomoni, T. Moseley, and M. Vachharajani. Fastforward for efficient pipeline parallelism: a cache-optimized concurrent lock-free queue. PPoPP 2008. ACM.
    - Supports Total Store Order OOO architectures (e.g. Intel Core)
    - Active waiting. Use OS as less as possible.
- Native SPMC and MPSC
  - see MPMC

SPMC and MCSP via SPSC + control

- SPMC(x) fence-free queue wit x consumers
  - One SPSC “input” queue and x SPSC “output” queues
  - One flow of control (thread) dispatch items from input to outputs



- MPSC(y) fence-free queue with y producers
  - One SPSC “output” queue and y SPSC “input” queues
  - One flow of control (thread) gather items from inputs to output

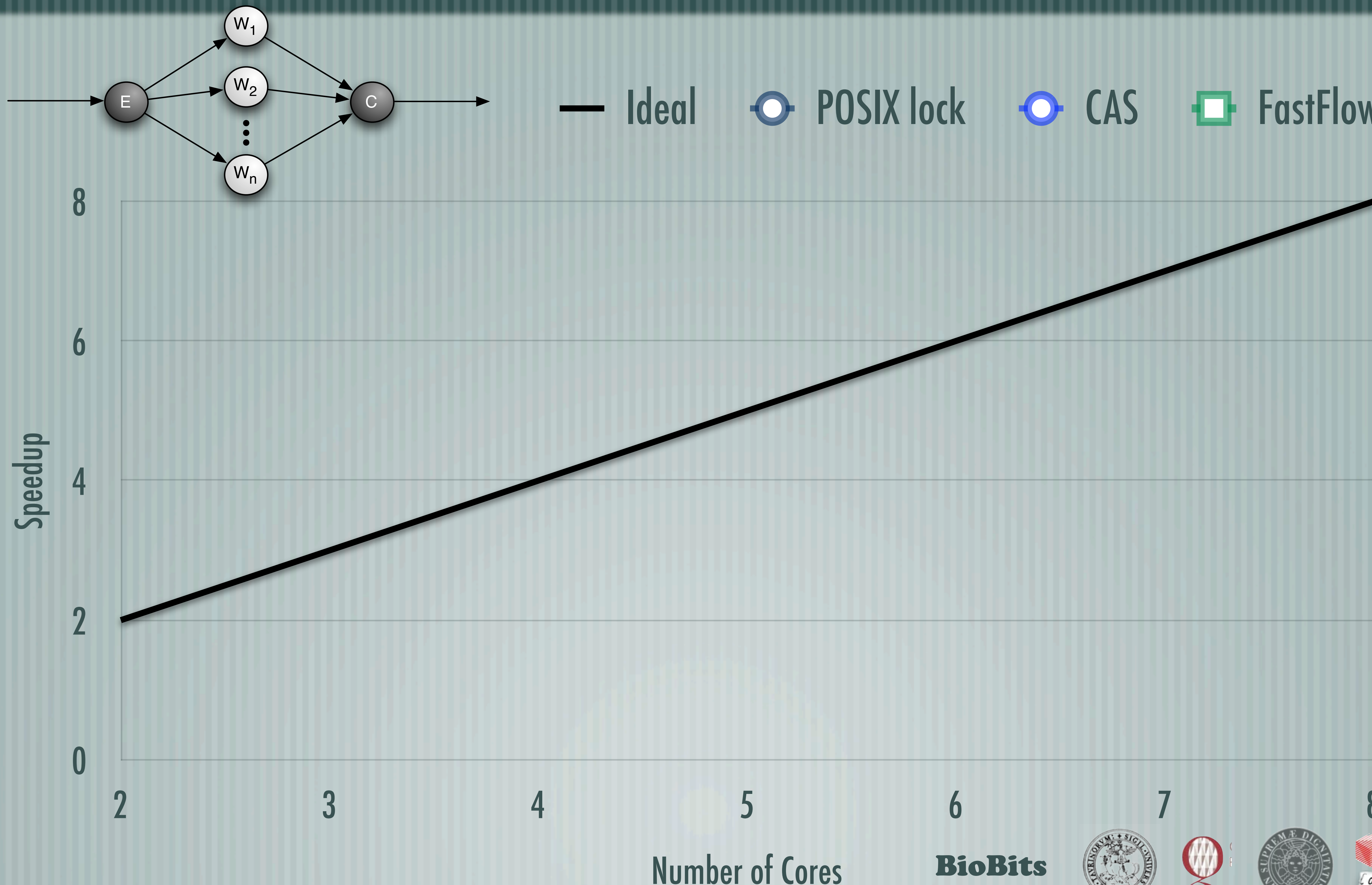


- x and y can be dynamically changed
- MPMC = MCSP + SPMC
  - Just juxtapose the two parametric networks

FastFlow: A step forward

- Implements lock-free SPSC, SPMC, MPSC, MPMC queues
  - Exploiting streaming networks
  - Features can be composed as parametric streaming networks (graphs)
    - E.g. an optimized memory allocator can be added by fusing the allocator graphs with the application graphs
      - Not described here
    - Features are represented as skeletons, actually which compilation target are streaming networks
- C++ STL-like implementation
  - Can be used as a low-level library
  - Can be used to generatively compile skeletons into streaming networks
- Blazing fast on fine-grained computations

Very fine grain (0.5 μS)

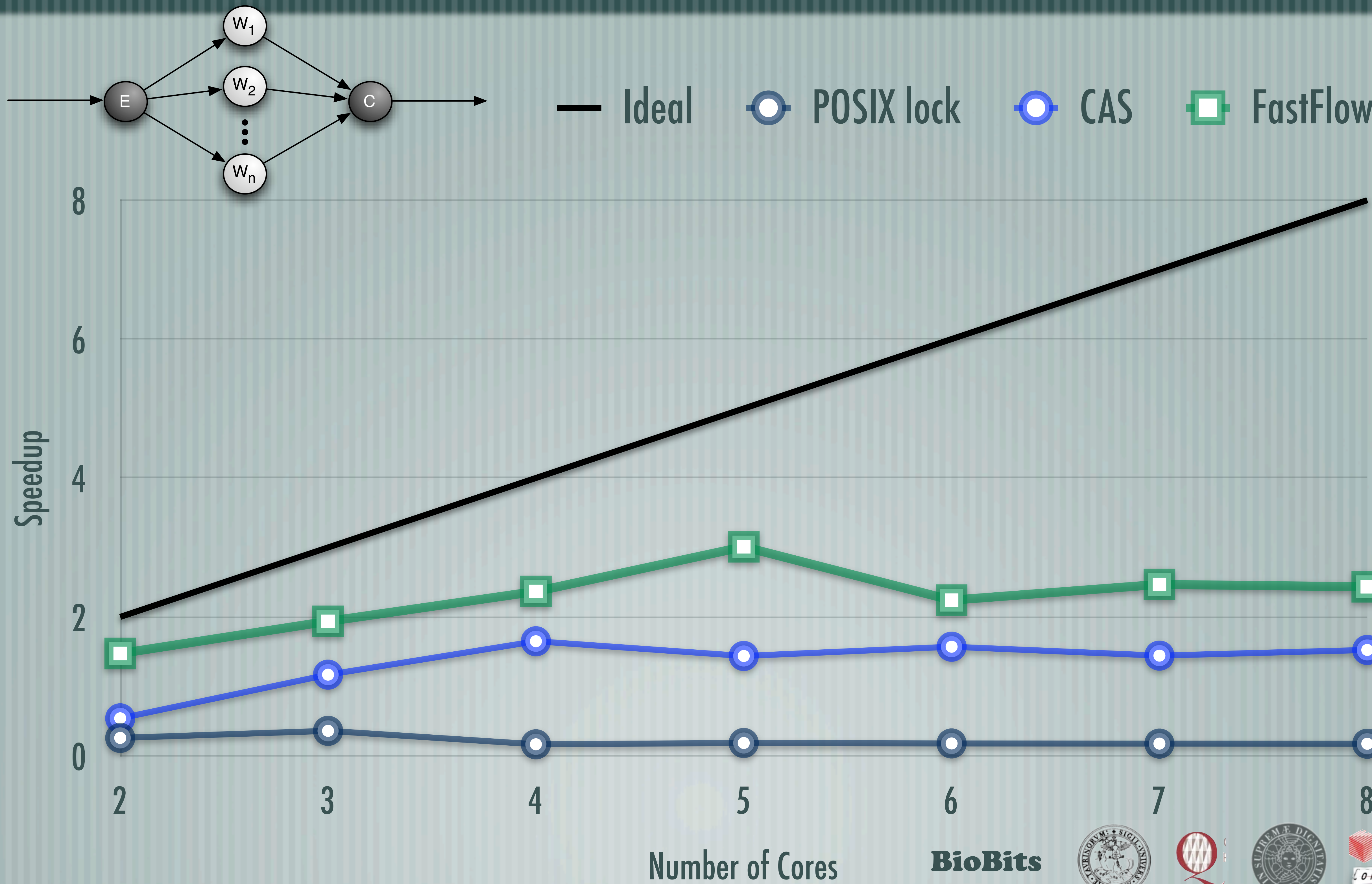


BioBits

ERCIM

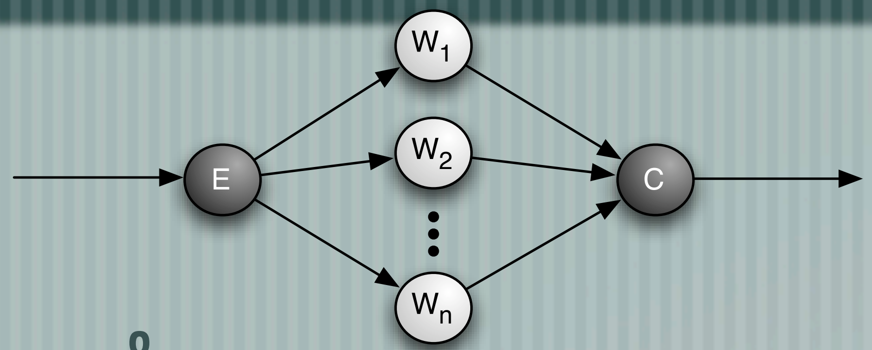
CoreGRID

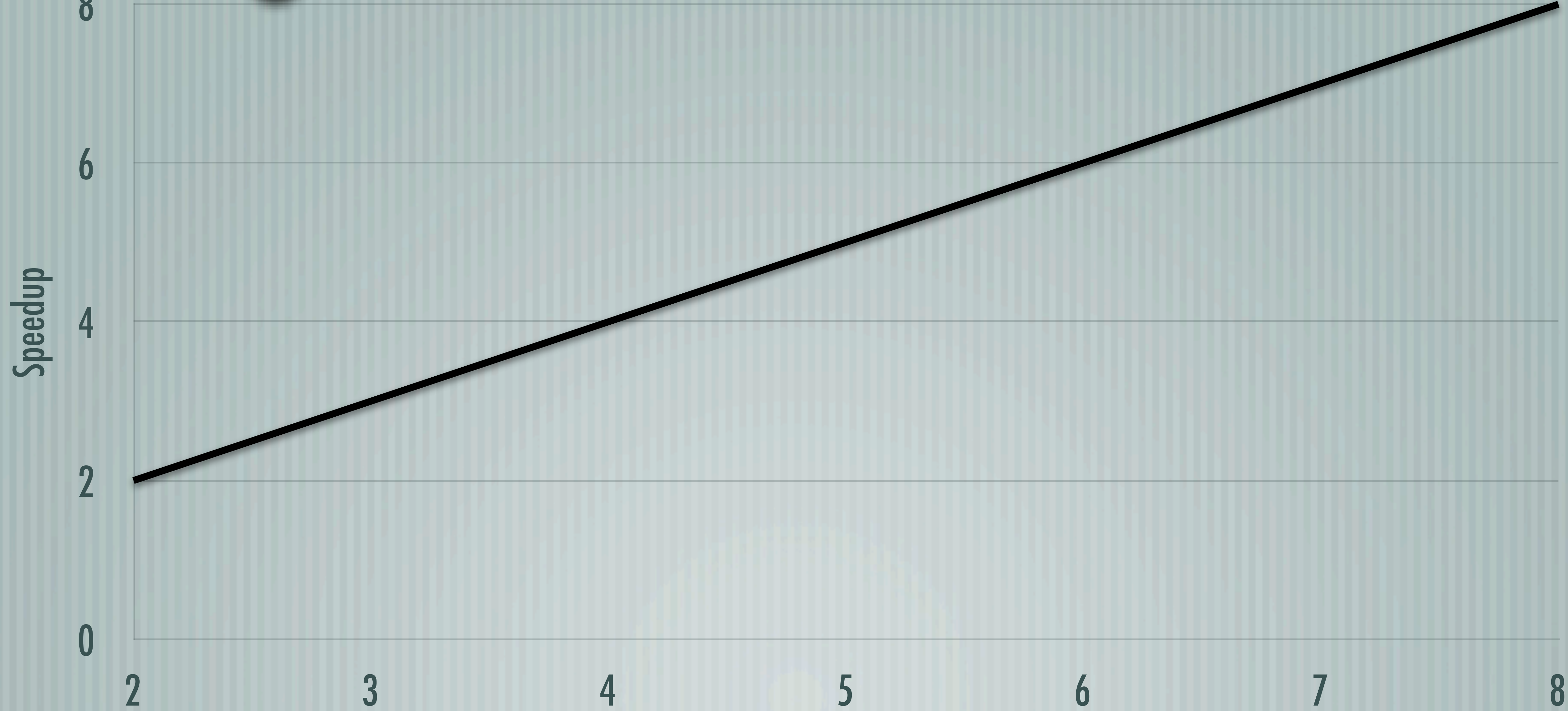
Very fine grain (0.5 μS)



BioBits ERCIM CoreGRID

Fine grain (5 μS)



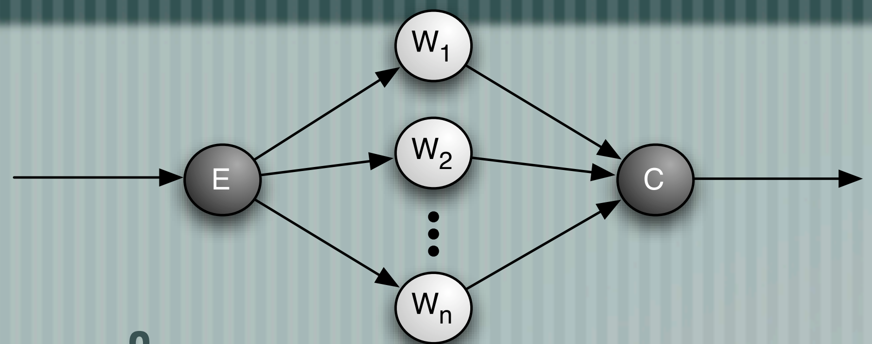


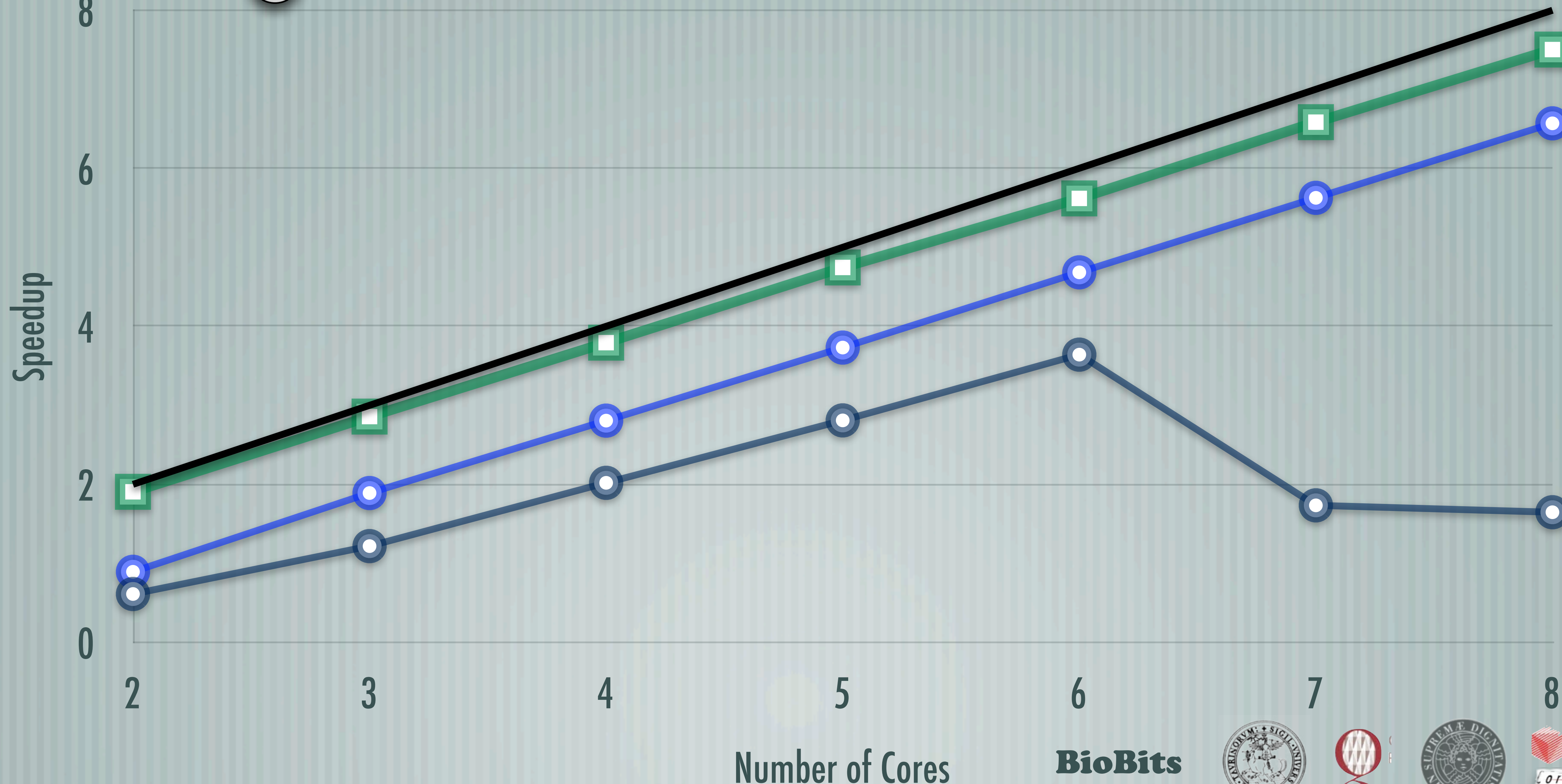
BioBits

ERCIM

CoreGRID

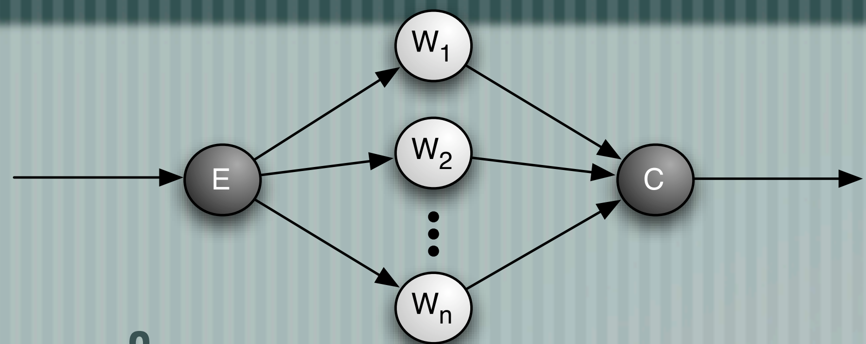
Fine grain (5 μS)

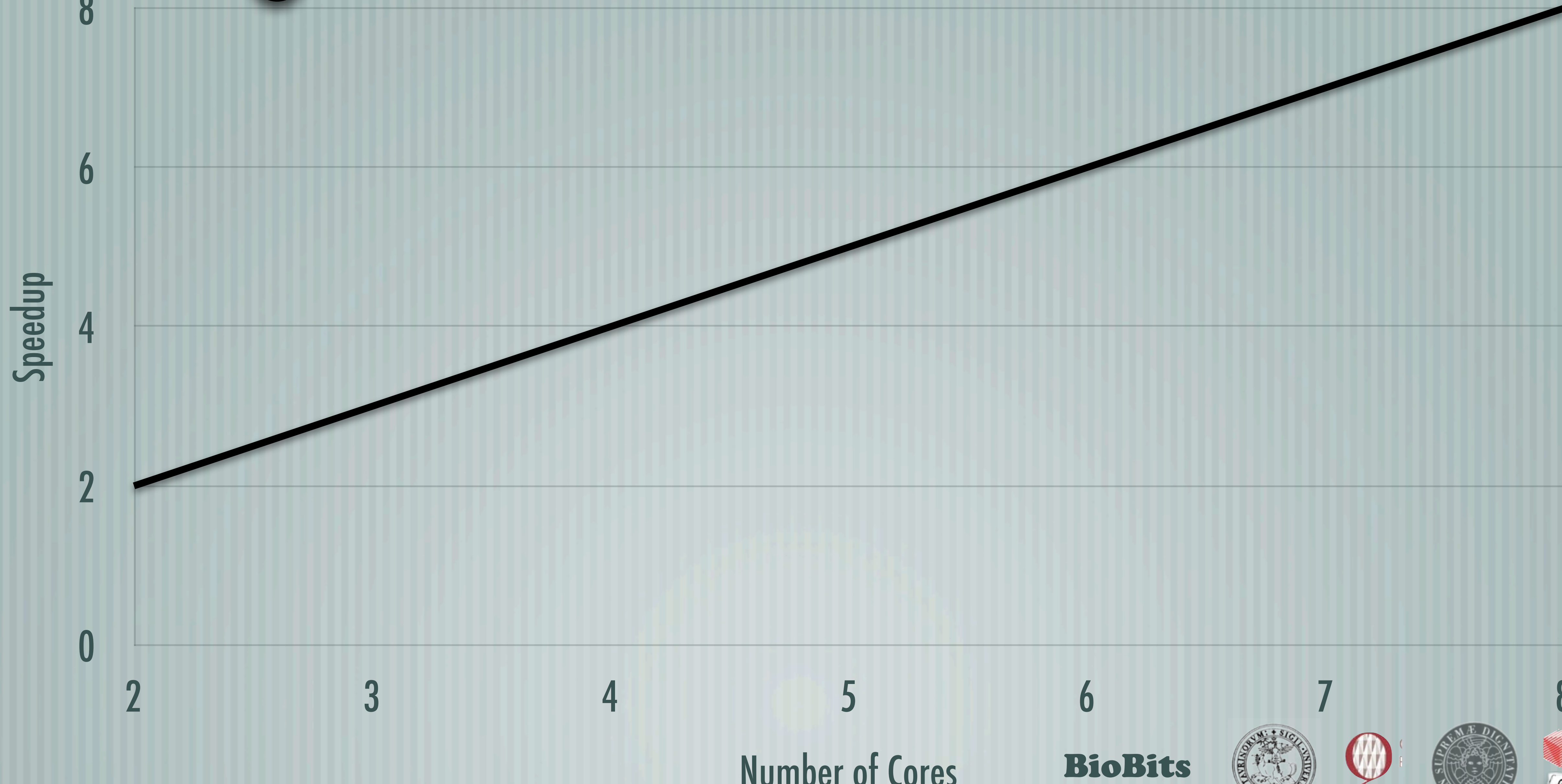




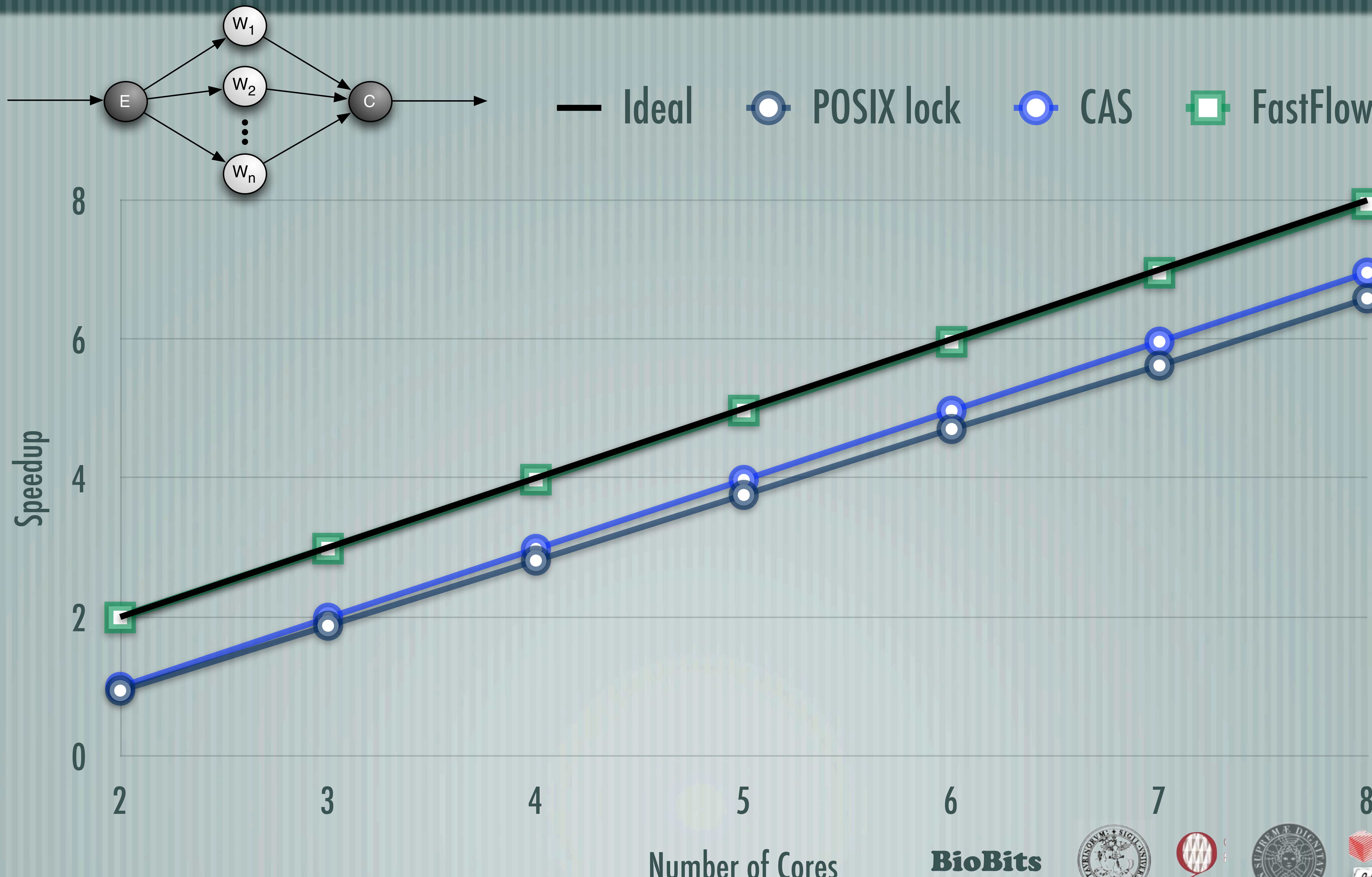
BioBits ERCIM CoreGRID

Medium grain (50 μS)



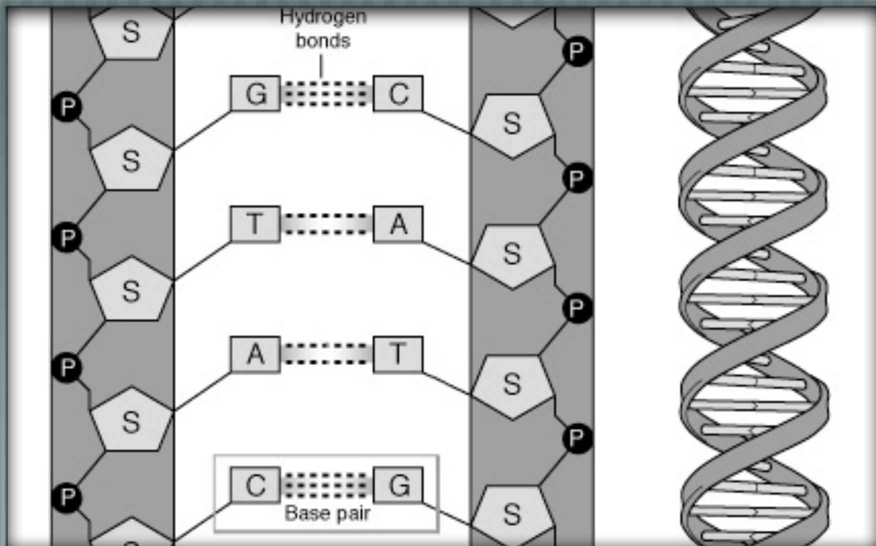


Medium grain (50 μS)



Biosequence alignment

- Smith-Waterman algorithm
  - Local alignment
  - Time and space demanding O(mn), often replaced by approximated BLAST
  - Dynamic programming
  - Real-world application
    - It has been accelerated by using FPGA, GCPU (CUDA), SSE2/x86, IBM Cell
- Best software implementation
  - SWPS3: evolution of Farrar's implementation
    - SSE2 + POSIX IPC



		G	A	A	T	T	C	A	G	T	T	A
	0	0	0	0	0	0	0	0	0	0	0	0
G	0	5	1	0	0	0	0	0	5	1	0	0
G	0	5	2	0	0	0	0	0	5	2	0	0
A	0	1	10	7	3	0	0	5	1	2	0	5
T	0	0	6	7	12	8	4	1	2	6	8	4
C	0	0	2	3	8	9	13	9	5	2	4	5
G	0	5	1	0	4	5	9	10	14	10	6	2
A	0	1	10	6	2	1	5	14	10	11	7	11

```
GAATTCAG     GAATTCAG
| | || |     | || | |
GGA-TC-G     GCAT-C-G

GAATTC-A     GAATTC-A
| | || |     | || | |
GGA-TCGA     GCAT-CGA
```

Smith-Waterman algorithm
Local alignment - dynamic programming - O(nm)

A matrix H is built as follows:

$$H(i,0) = 0,\ 0 \le i \le m$$
$$H(0,j) = 0,\ 0 \le j \le n$$
$$H(i,j) = \max \begin{Bmatrix} 0 & \\ H(i-1,j-1) + w(a_i,b_j) & \text{Match/Mismatch} \\ H(i-1,j) + w(a_i,-) & \text{Deletion} \\ H(i,j-1) + w(-,b_j) & \text{Insertion} \end{Bmatrix},\ 1 \le i \le m, 1 \le j \le n$$

Where:

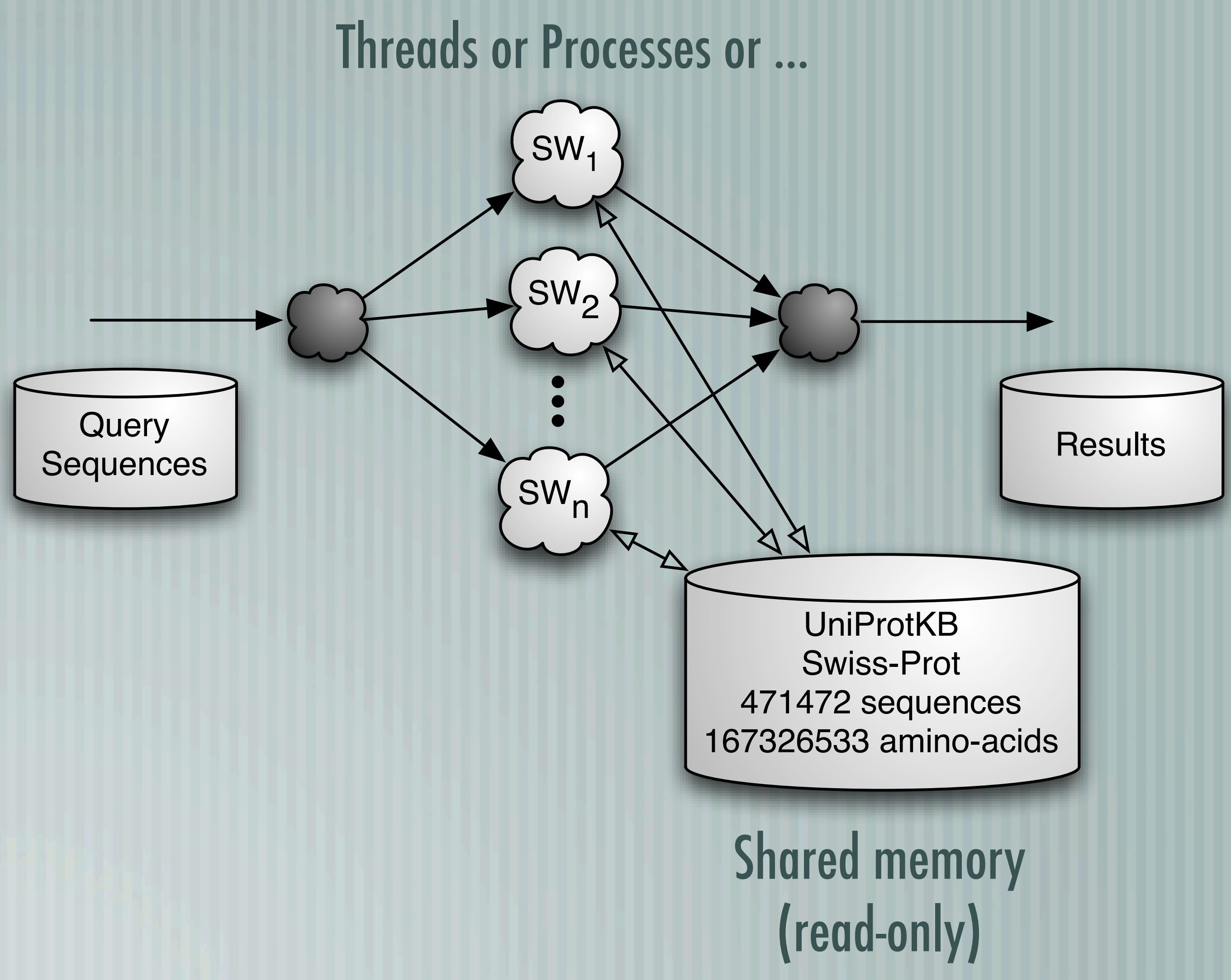
- a, b = Strings over the Alphabet Σ
- m = length(a)
- n = length(b)
- $H(i,j)$ - is the maximum Similarity-Score between the substring of a of length i, and the substring of b of length j
- $w(c,d),\ c,d \in \Sigma \cup \{'-'\}$, '-' is the gap-scoring scheme

- Substitution Matrix: describes the rate at which one character in a sequence changes to other character states over time
- Gap Penalty: describes the costs of gaps, possibly as function of gap length

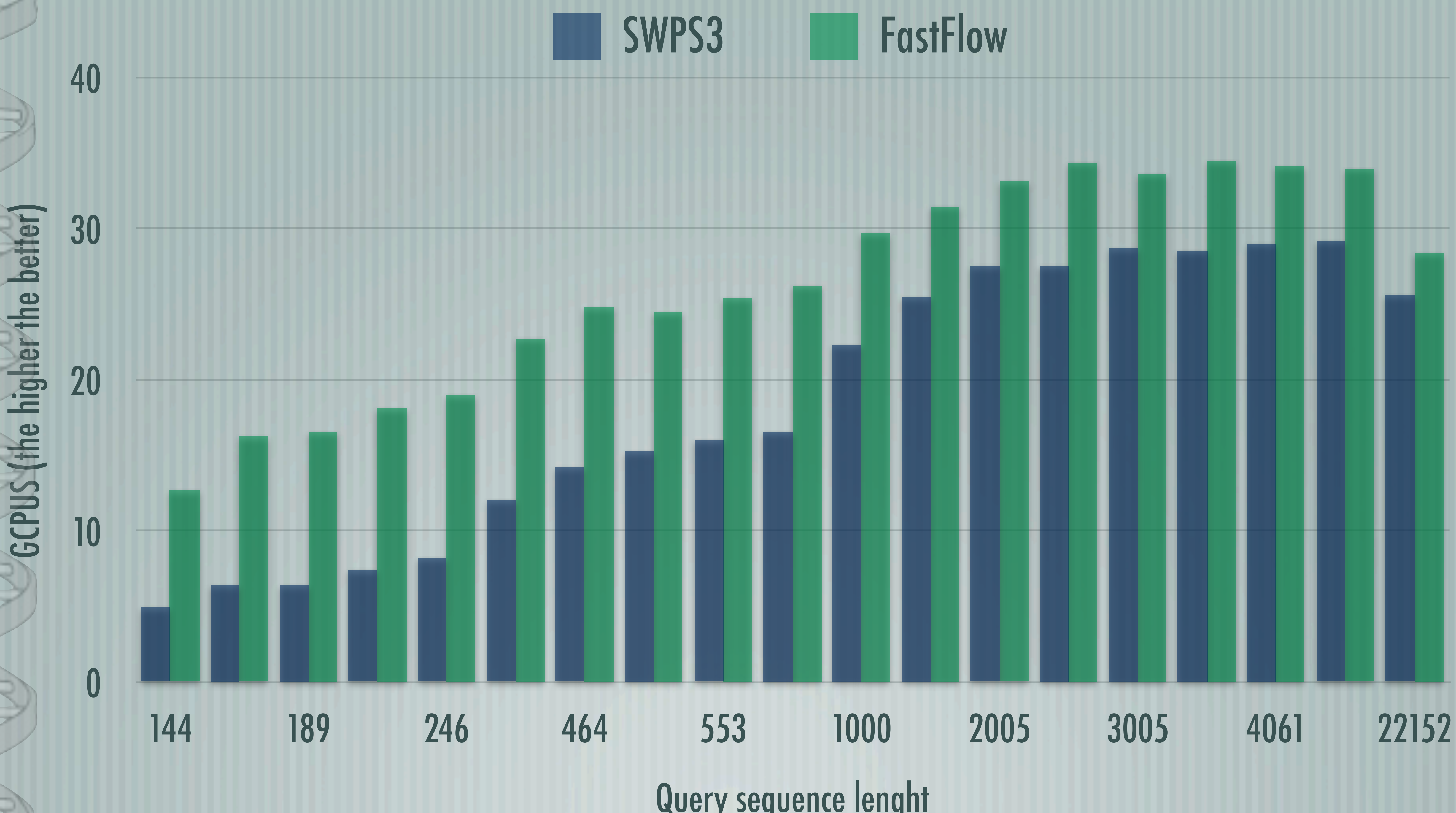
Experiment parameters
Affine Gap Penalty: 10-2k, 5-2k, ...
Substitution Matrix: BLOSUM50

Biosequence testbed

- Each query sequence (protein) is aligned against the whole protein DB
  - E.g. Compare unknown sequence against a DB of known sequences
- SWPS3 implementation exploits POSIX processes and pipes
  - Faster than POSIX threads + locks

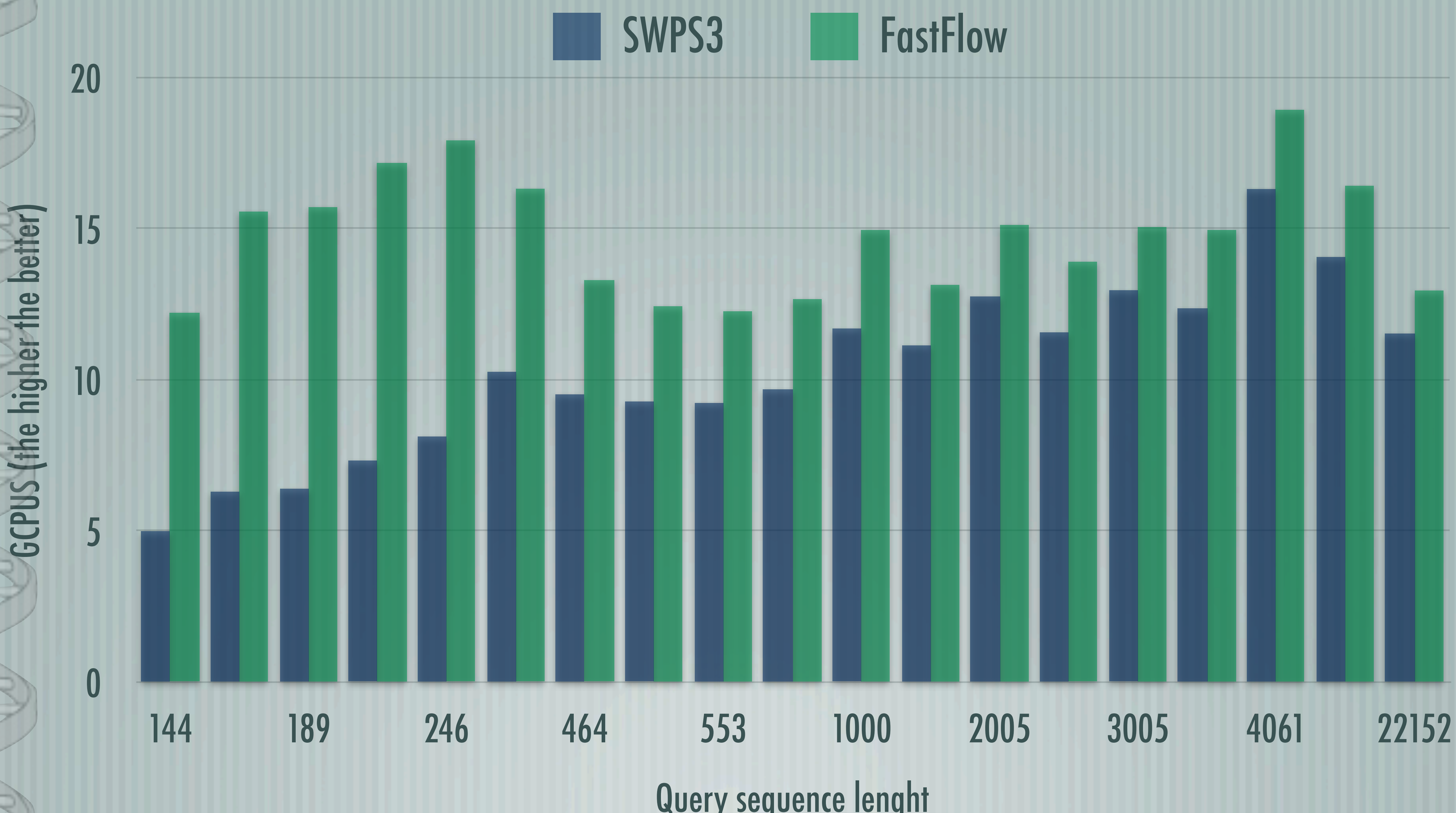


Smith Waterman (10-2k gap penalty)



BioBits ERCIM CoreGRID

Smith Waterman (5-2k gap penalty)



BioBits ERCIM CoreGRID

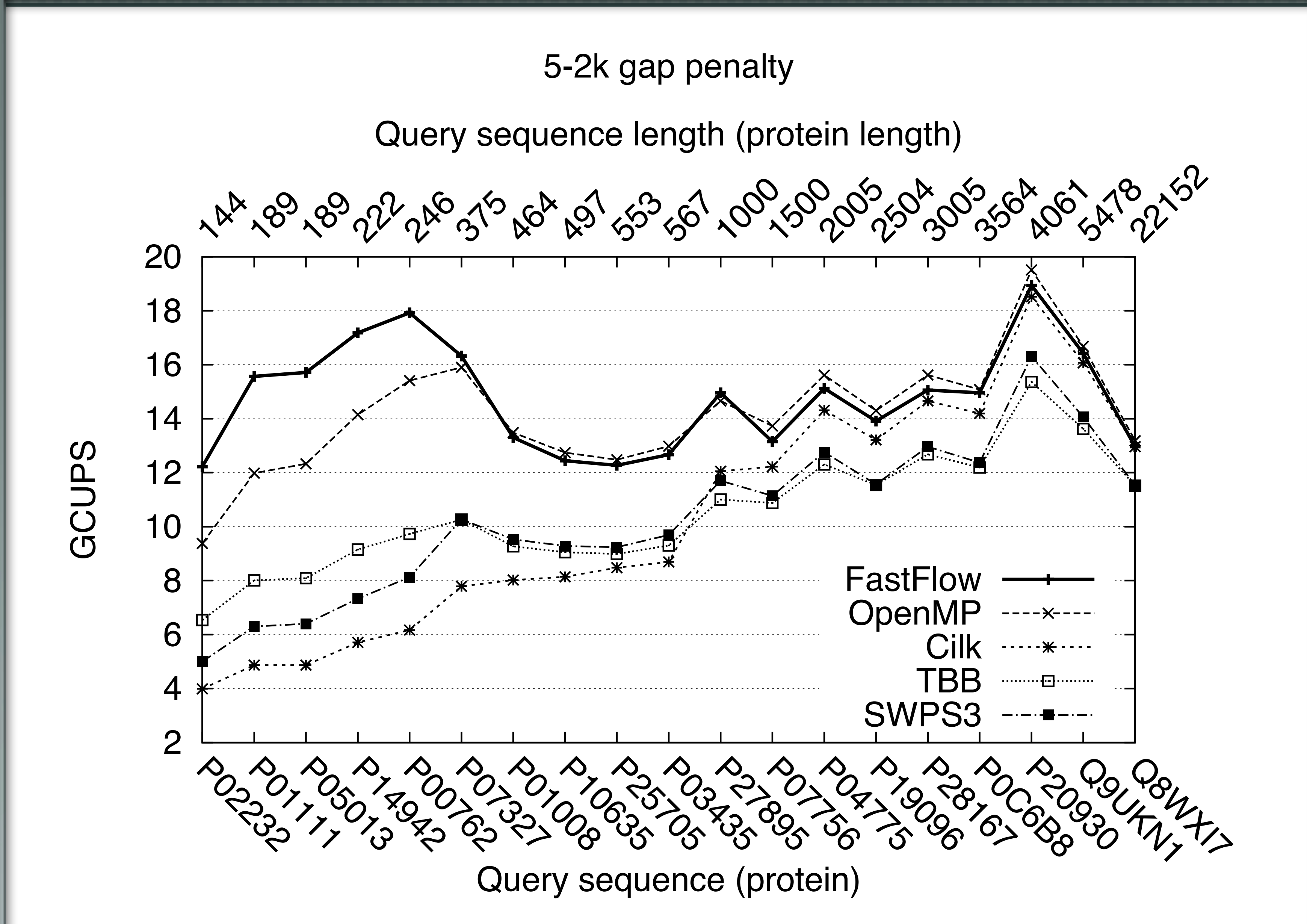
Conclusions

- FastFlow support efficiently streaming applications on commodity SCM (e.g. Intel core architecture)
  - More efficiently than POSIX threads (standard or CAS lock)
- Smith Waterman algorithm with FastFlow
  - Obtained from SWPS3 by syntactically substituting read and write on POSIX pipes with fastflow push and FastFlow pop an push
    - In turn, POSIX pipes are faster than POSIX threads + locks in this case
  - Scores twice the speed of best known parallel implementation (SWPS3) on the same hardware (Intel 2 x Quad-core 2.5 GHz)

Future Work

FastFlow

- Is open source (STL-like C++ library will be released soon) [✔]
  - Contact me if you interested
- Include a specialized (very fast) parallel memory allocator [✔]
- Can be used to automatically parallelize a wide class of problems [ ]
  - Since it efficiently supports fine grain computations
- Can be used as compilation target for skeletons [ ]
  - Support parametric parallelism schemas and support compositionality (can be formalized as graph rewriting)
- Can be extended for CC-NUMA architectures [ ]
- Can be used to extend Intel TBB and OpenMP [✔]
  - Increasing the performances of those tools



FastFlow is also faster than Open MP, Intel TBB and Cilk (at least for streaming on Intel 2 x quad-core)

THANK YOU!
QUESTIONS?

... and one question for you

Are those chips really build for parallel computing?