Multi-GPU Short-Read Alignment

Radhakrishna “RK” Bettadapura

Strand Life Sciences

Next Generation Sequencing (NGS)

Parallel sequencing

Read 1: AGGT Read 2: GAAATTG … Read n: GCGGTTCCG

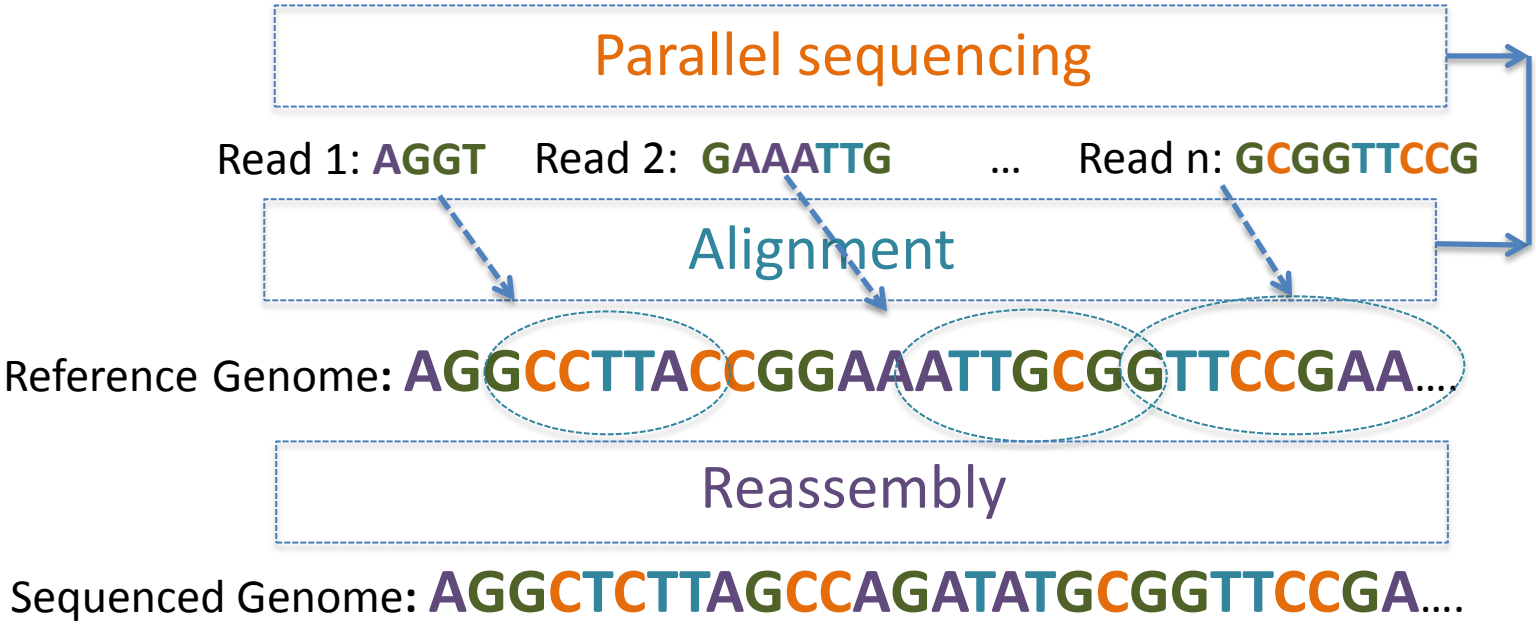
Alignment

Reference Genome: AGGCCTTACCGGAAATTGCGGTTCCGAA….

Reassembly

Sequenced Genome: AGGCTCTTAGCCAGATATGCGGTTCCGA….

Next Generation Sequencing (NGS)



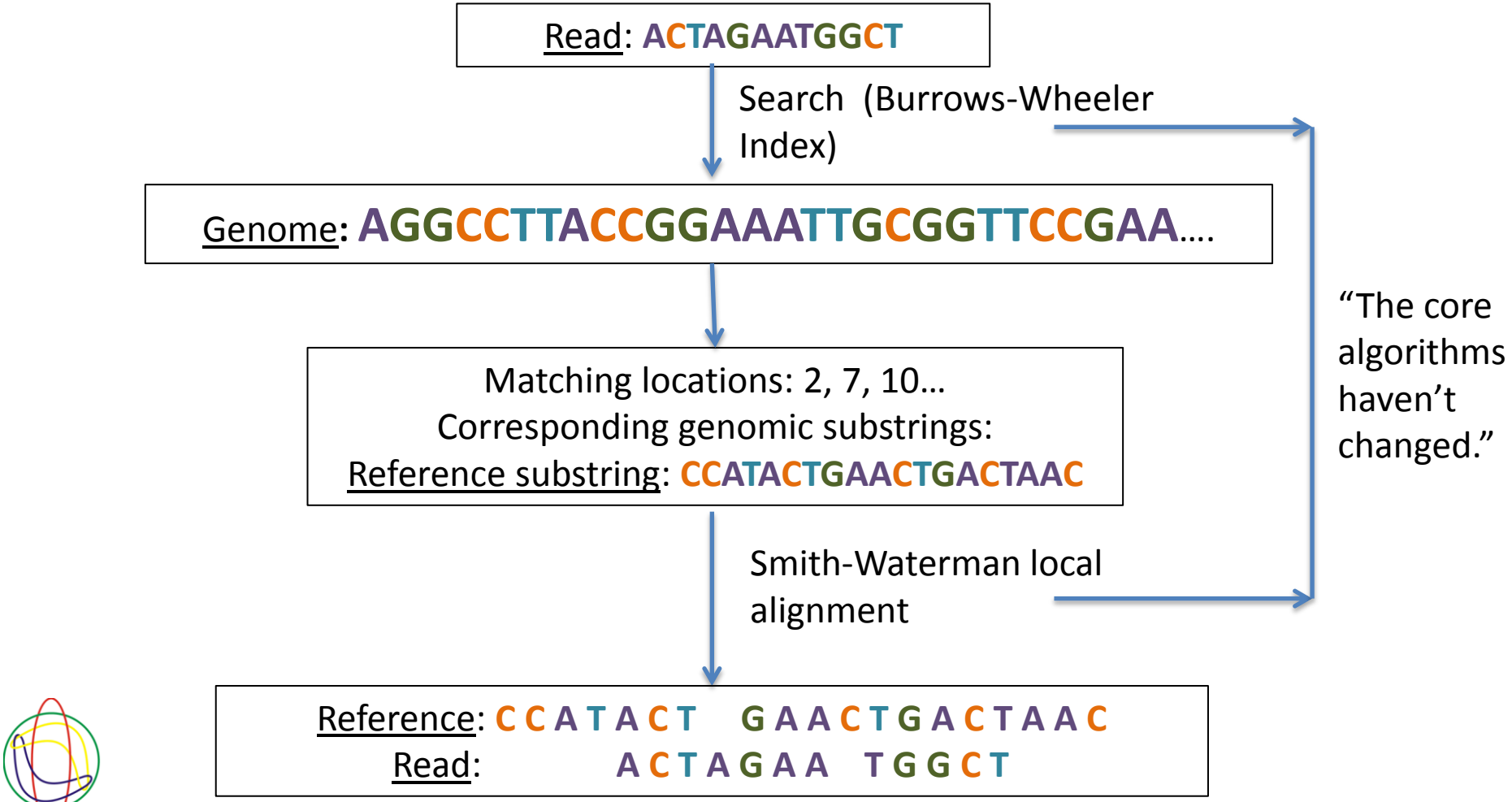
BIG Data

10^9 reads

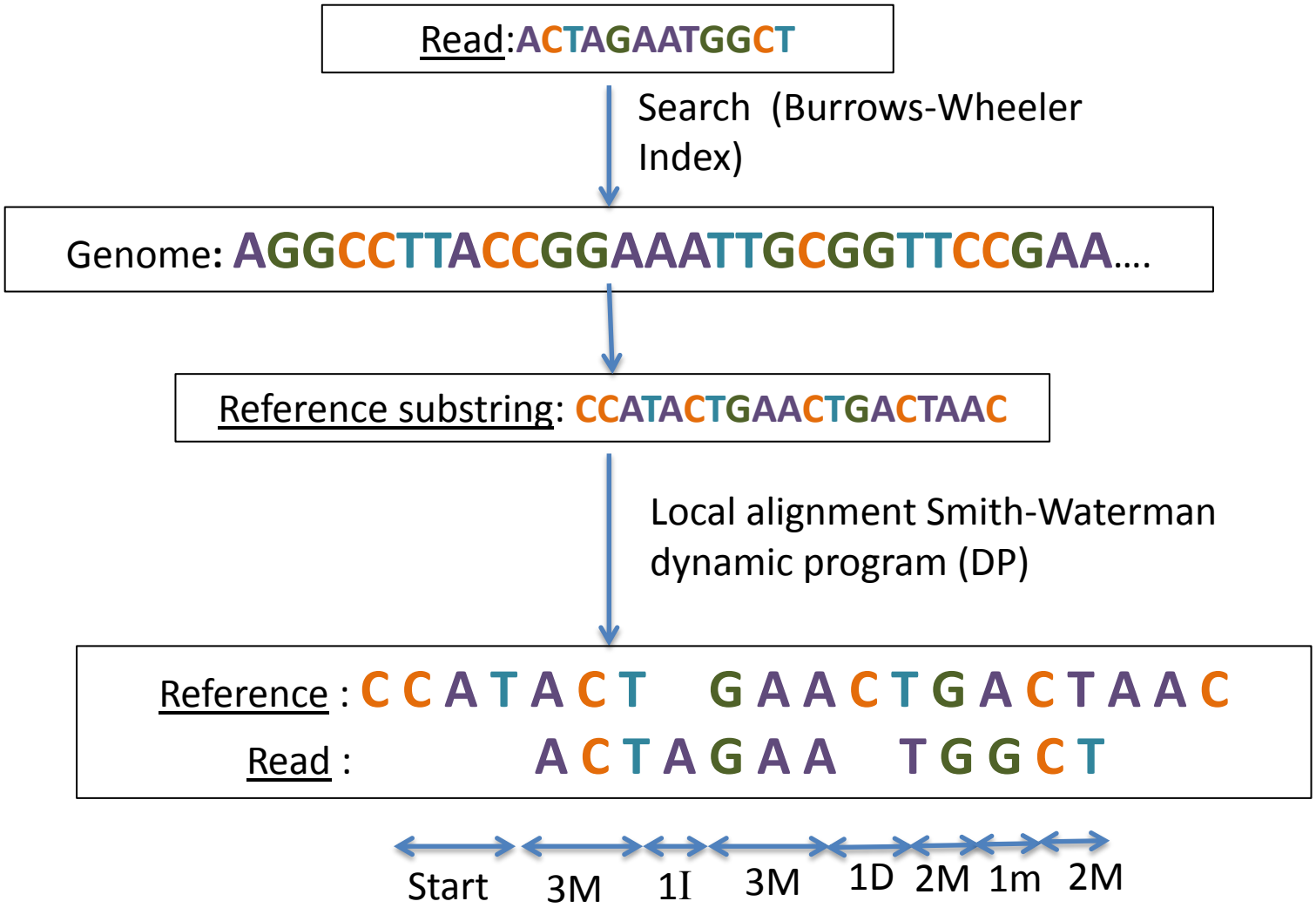
1 Read ~= 100-1000bp

NA 12878: 1 GB reference, 100GB reads

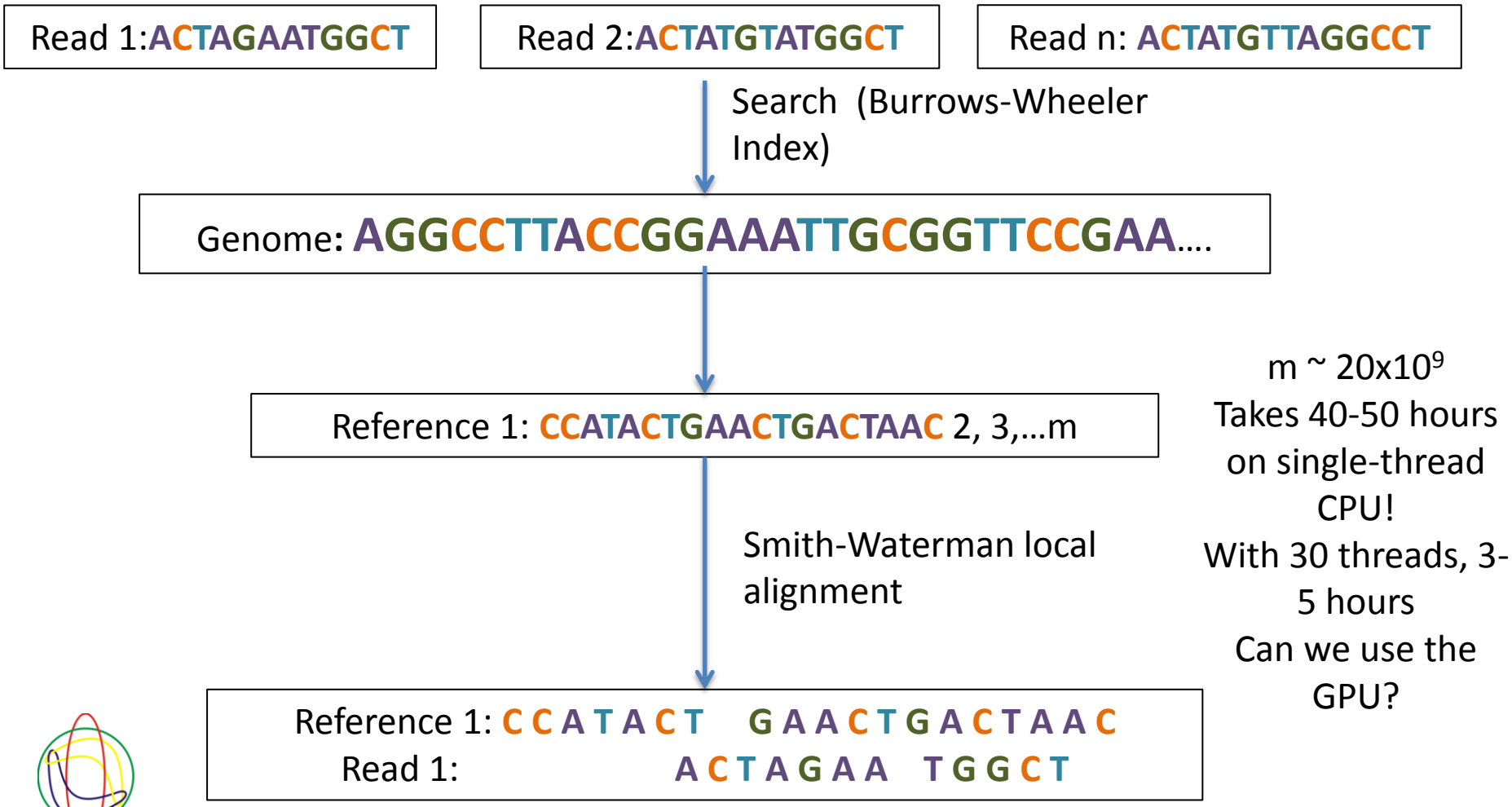
Short-read alignment



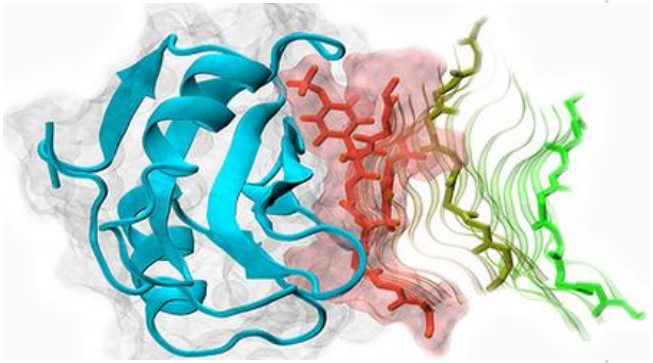
Short-read alignment

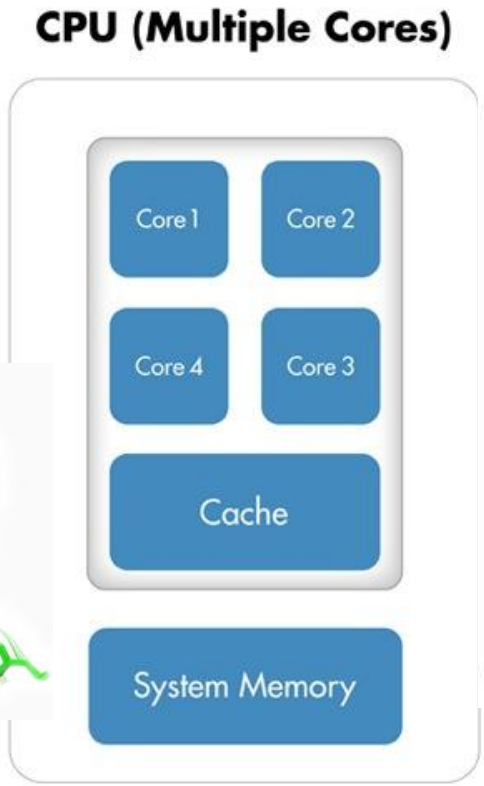


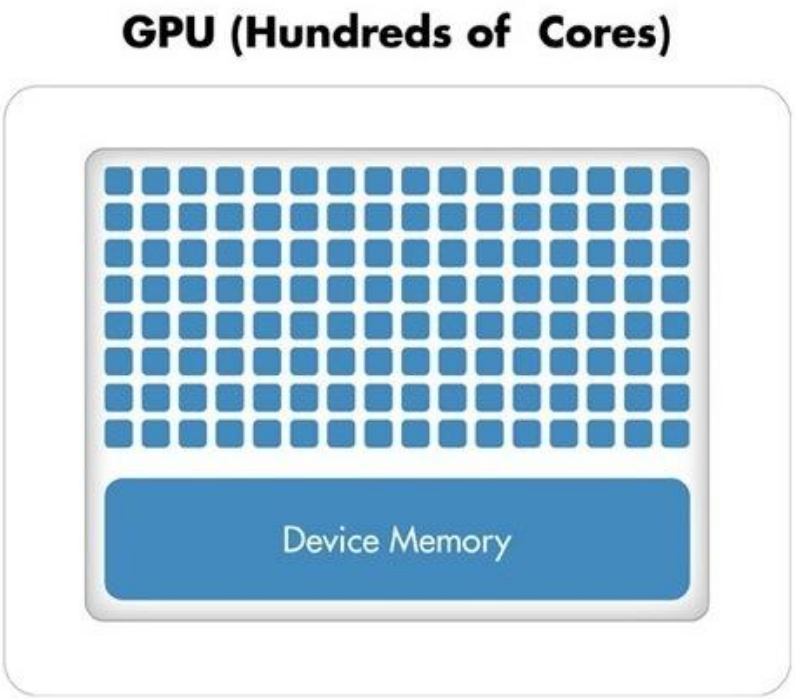
Short-read alignment



Graphics Processing Unit (GPU)







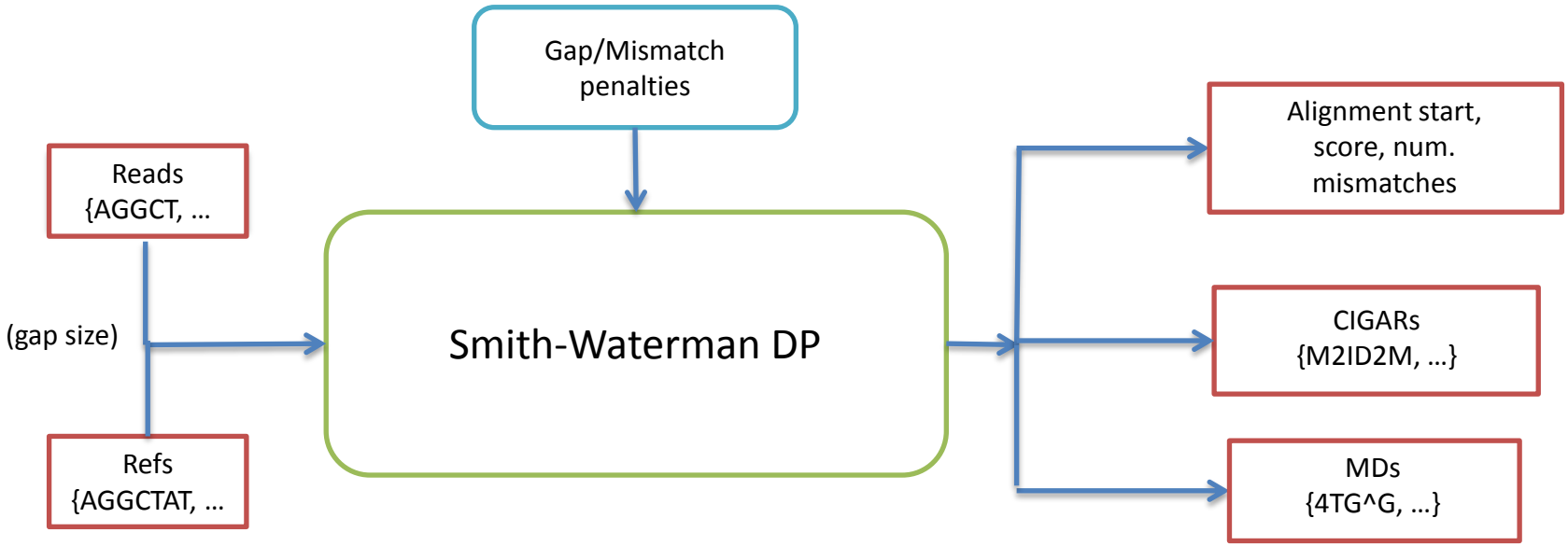
- Scalable, massively parallel computing
- Initially used to speed up graphics
- General purpose took off with CUDA
- Speedups of upto 500x, depending on the problem

K10 Tesla:
2.8GB global mem.
1536 cores
745MhZ clock

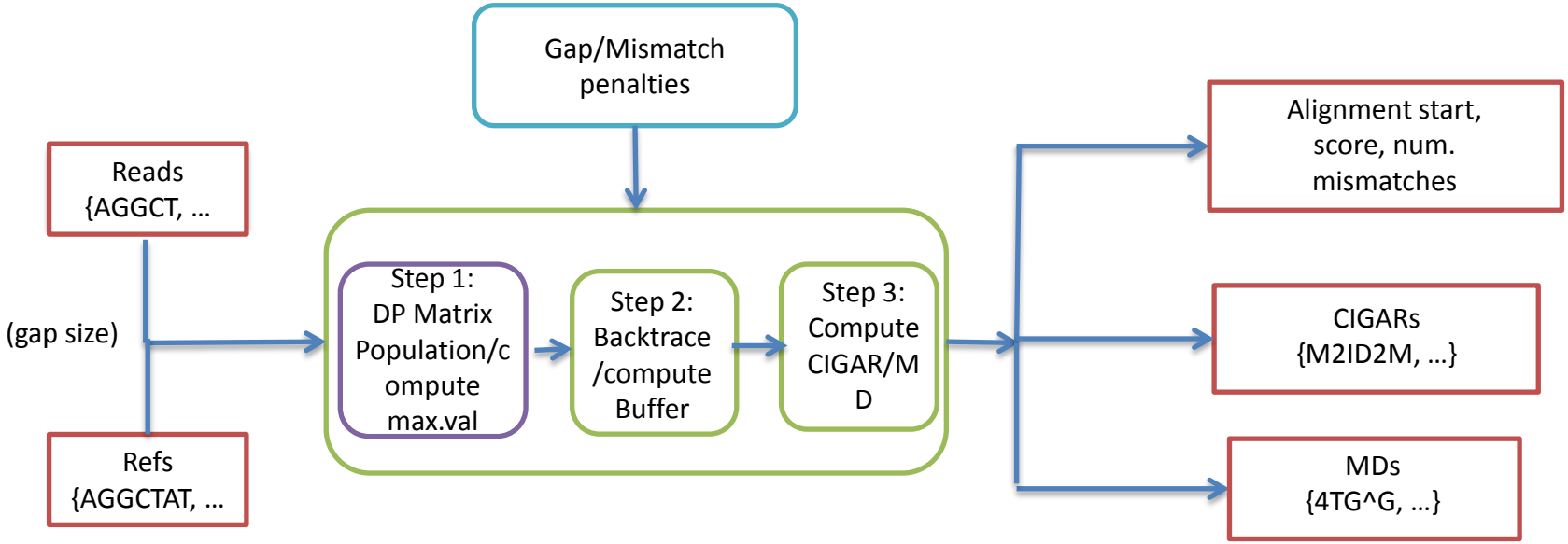
Goal: Implement Smith Waterman on multiple GPUs

Secondary Goal: Prove that it's (much) faster!

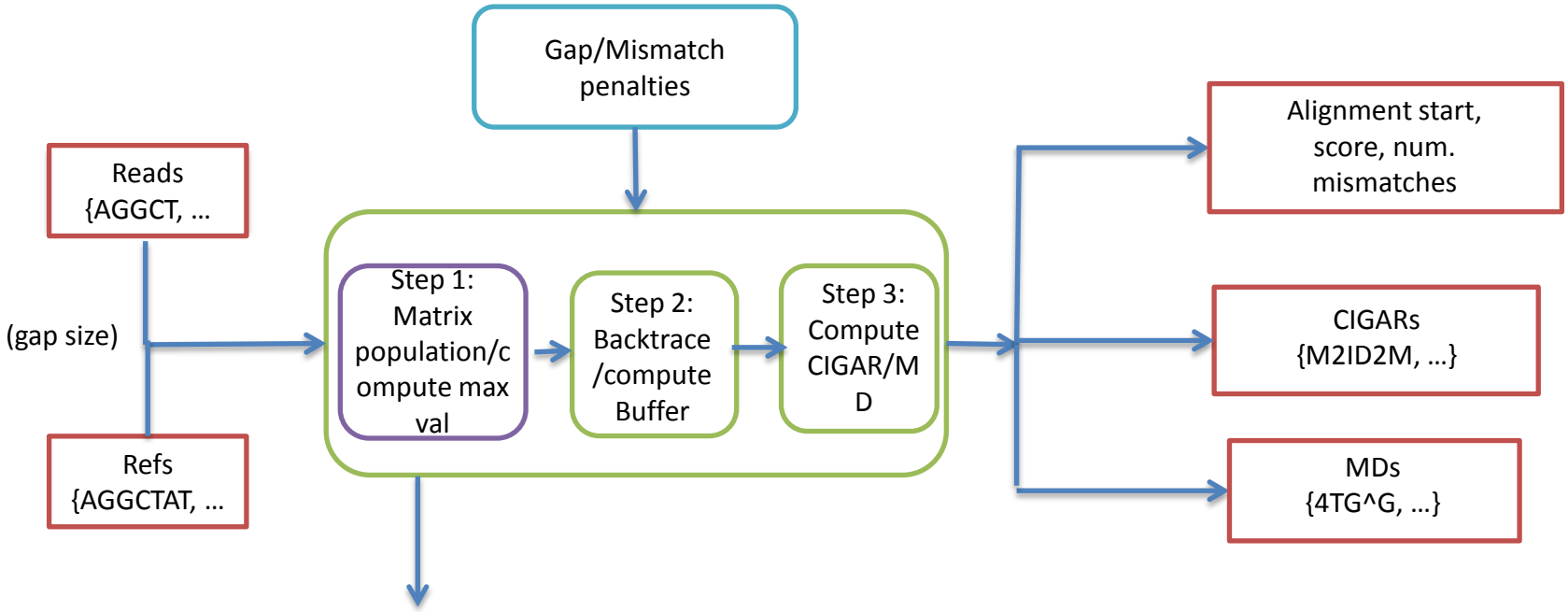
Algorithm


Gap/Mismatch penalties
Reads {AGGCT, …
(gap size)
Refs {AGGCTAT, …
Smith-Waterman DP
Alignment start, score, num. mismatches
CIGARs {M2ID2M, …}
MDs {4TG^G, …}

Algorithm


Gap/Mismatch penalties
Reads {AGGCT, …
(gap size)
Refs {AGGCTAT, …
Step 1: DP Matrix Population/compute max.val
Step 2: Backtrace /compute Buffer
Step 3: Compute CIGAR/MD
Alignment start, score, num. mismatches
CIGARs {M2ID2M, …}
MDs {4TG^G, …}

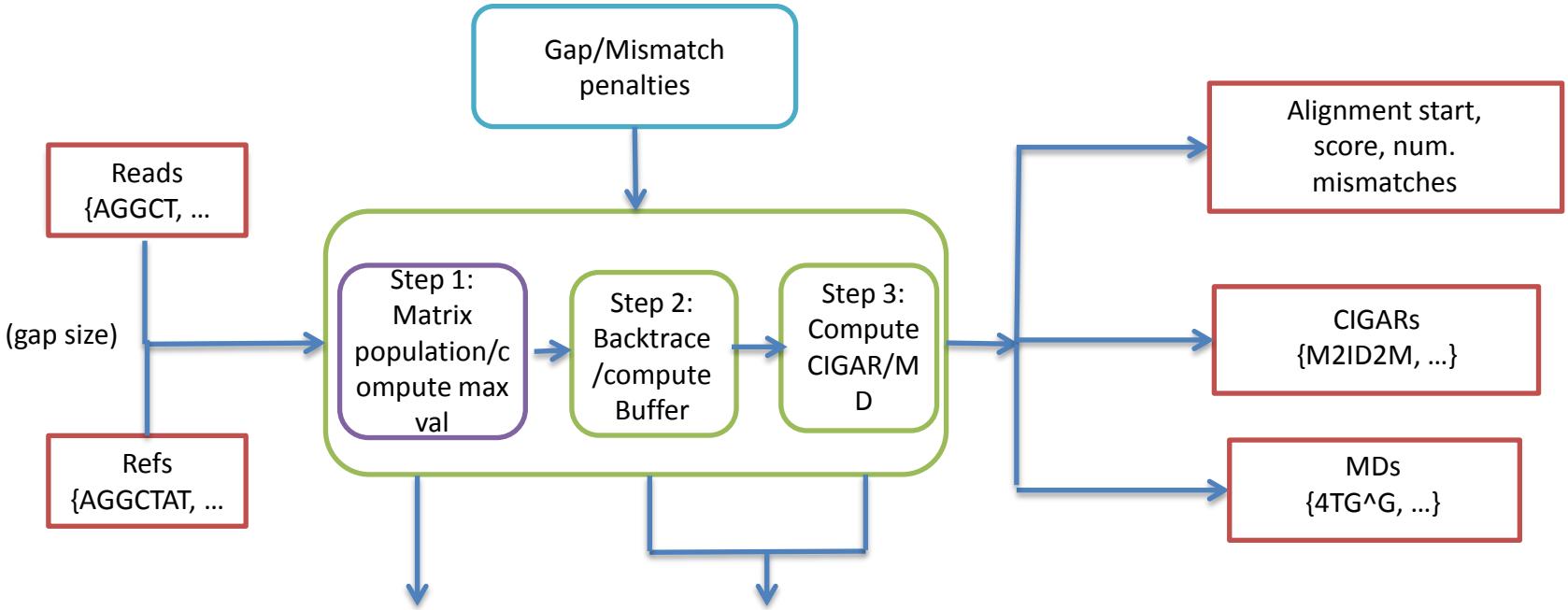
Algorithm



- Prior work (protein alignment)
- Quadratic computation but linear storage

Y. Liu, A. Wirawan, B. Schmidt, CUDASW++ 3.0: accelerating Smith-Waterman protein database search by coupling CPU and GPU SIMD instructions, BMC Bioinformatics, 2013

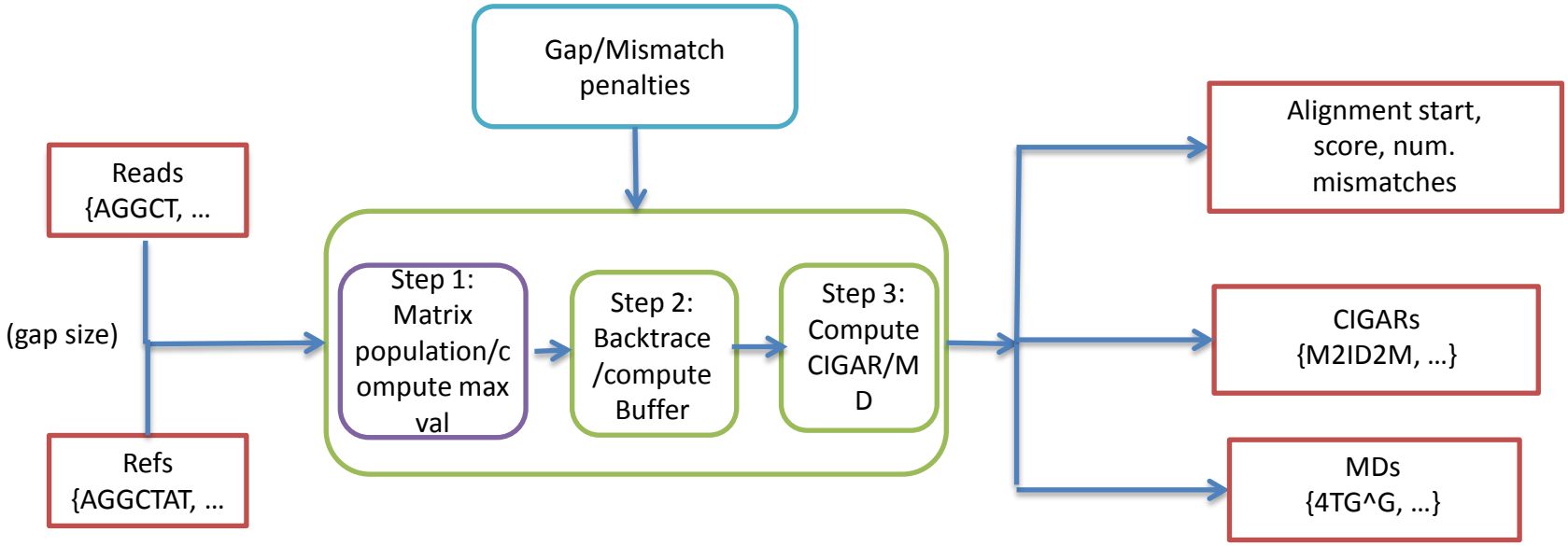
Algorithm



- Prior work (protein alignment)
- Quadratic computation but linear storage

- Quadratic storage
- Cannot be done in parallel with step 1 (though see final slide)
- Very high mem/math ratio (no floating point math + loads/stores + string manipulation)

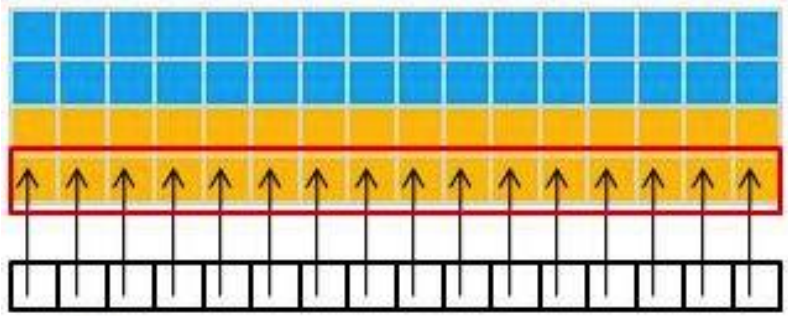
Algorithm



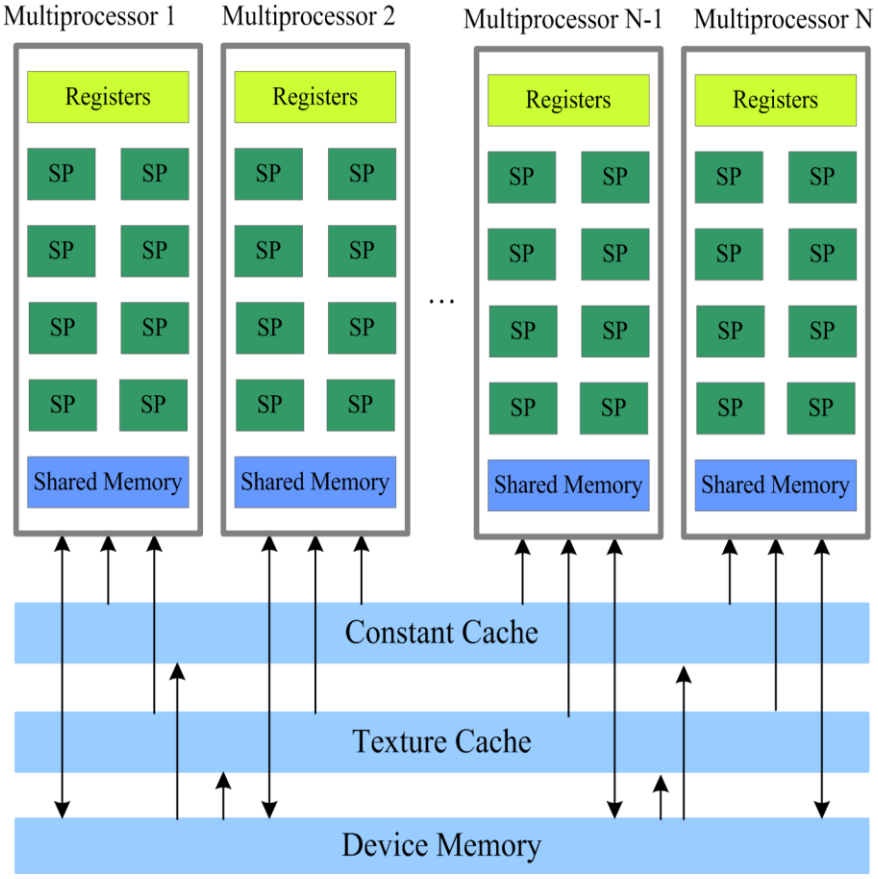
Parallelisation scheme: 1 thread per DP
(Store DP matrix/CIGAR/MD in thread local storage)

Implementation

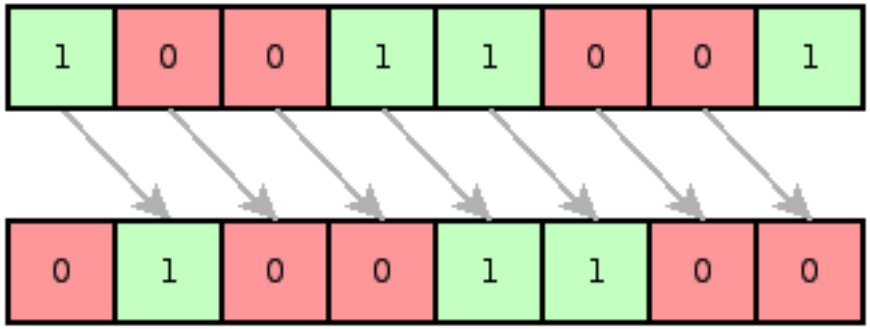
Coalesced memory access:



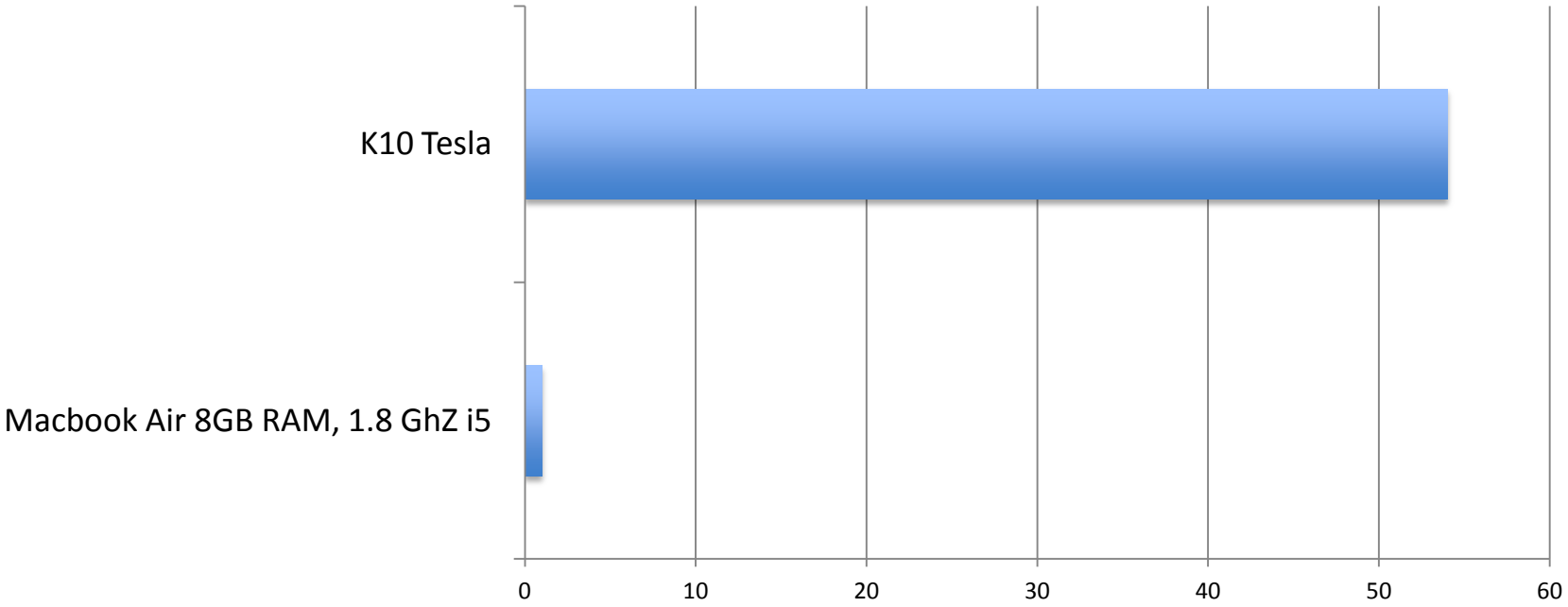
Judicious use of constant/register mem.



Judicious use of bitwise operations (prove them optimal first!)



Implementation



Bottom line: SW GPU is 54x CPU.

Implementation

Performance metrics

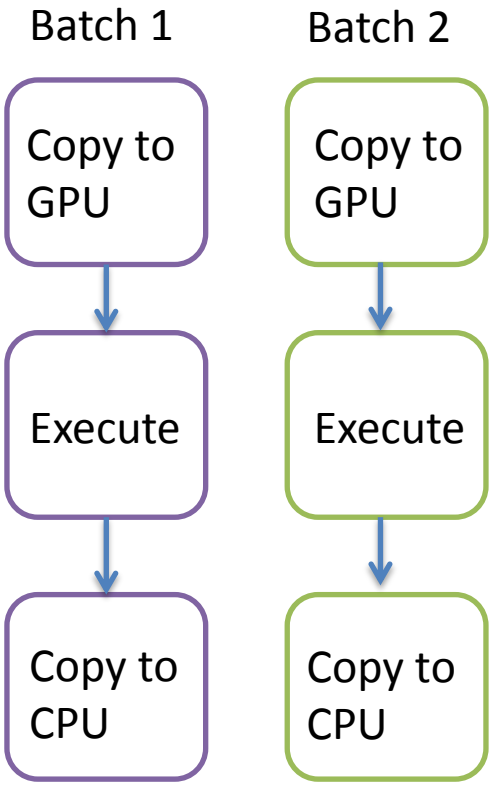
- 7.5 billion DPs/hr for read length = 100 and gap size = 5
- Some more metrics:
  - L1 cache miss ratio = 2%
  - L2 cache miss ratio = 0%
  - No register spills
  - No. of instructions executed/total number of instructions issued= 95% (indicates low warp serialisation)

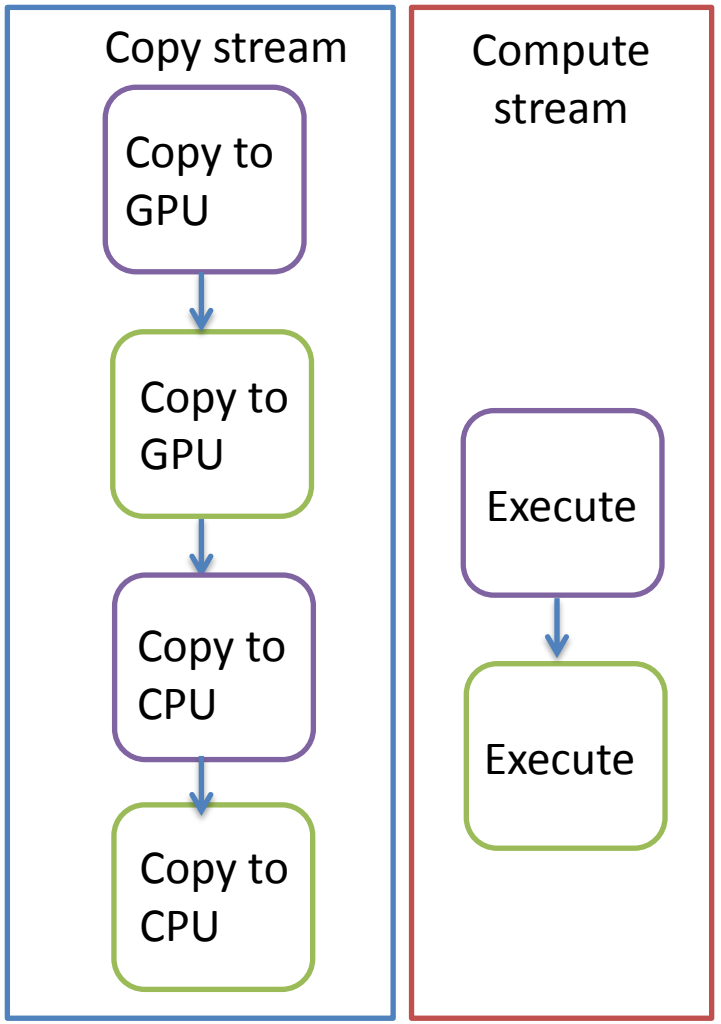
Bottom line: Novel implementation of SW (1, 2, 3) on the GPU results in extremely fast execution times

"That's all right in practice, but does it work in theory?"

Multiple GPUs

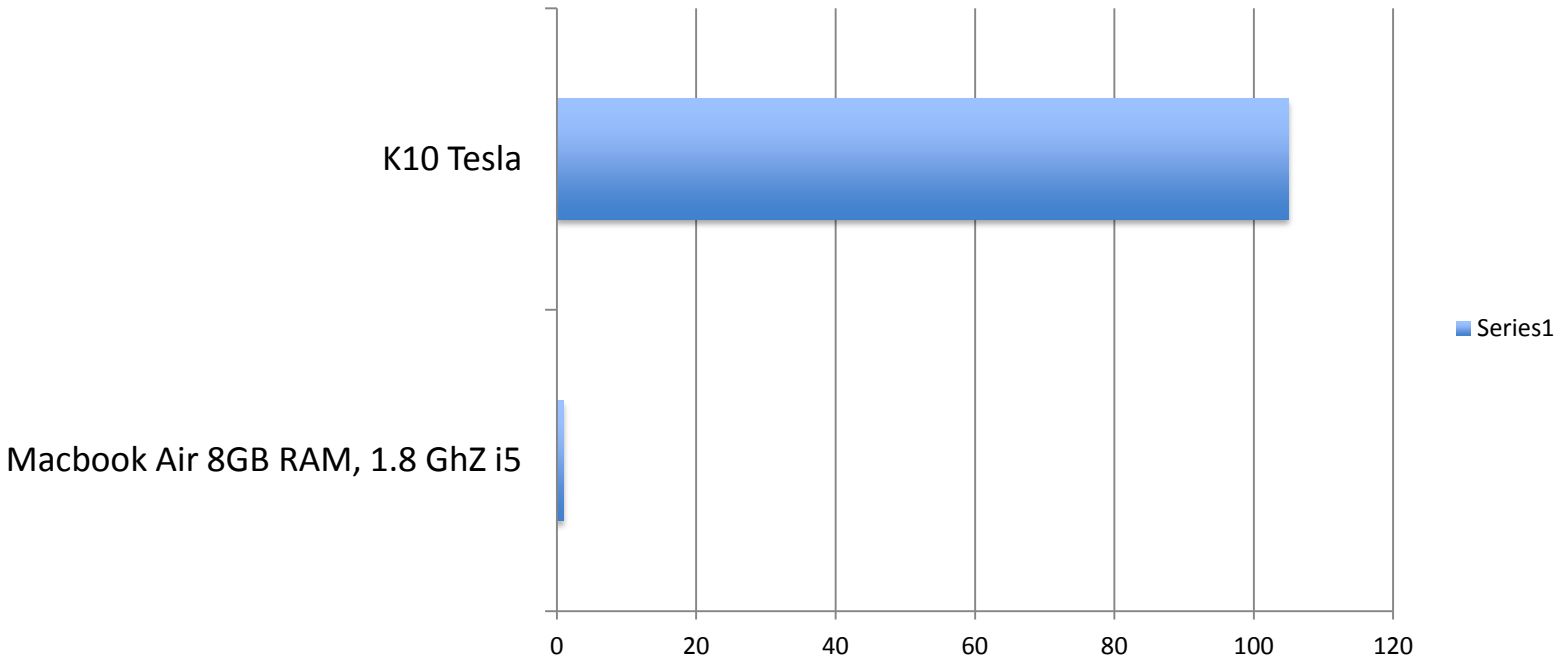
Use streams to schedule work and reduce copy overhead!





On multiple GPUs, streams mitigate copy overhead via GPU-wise asynchronicity

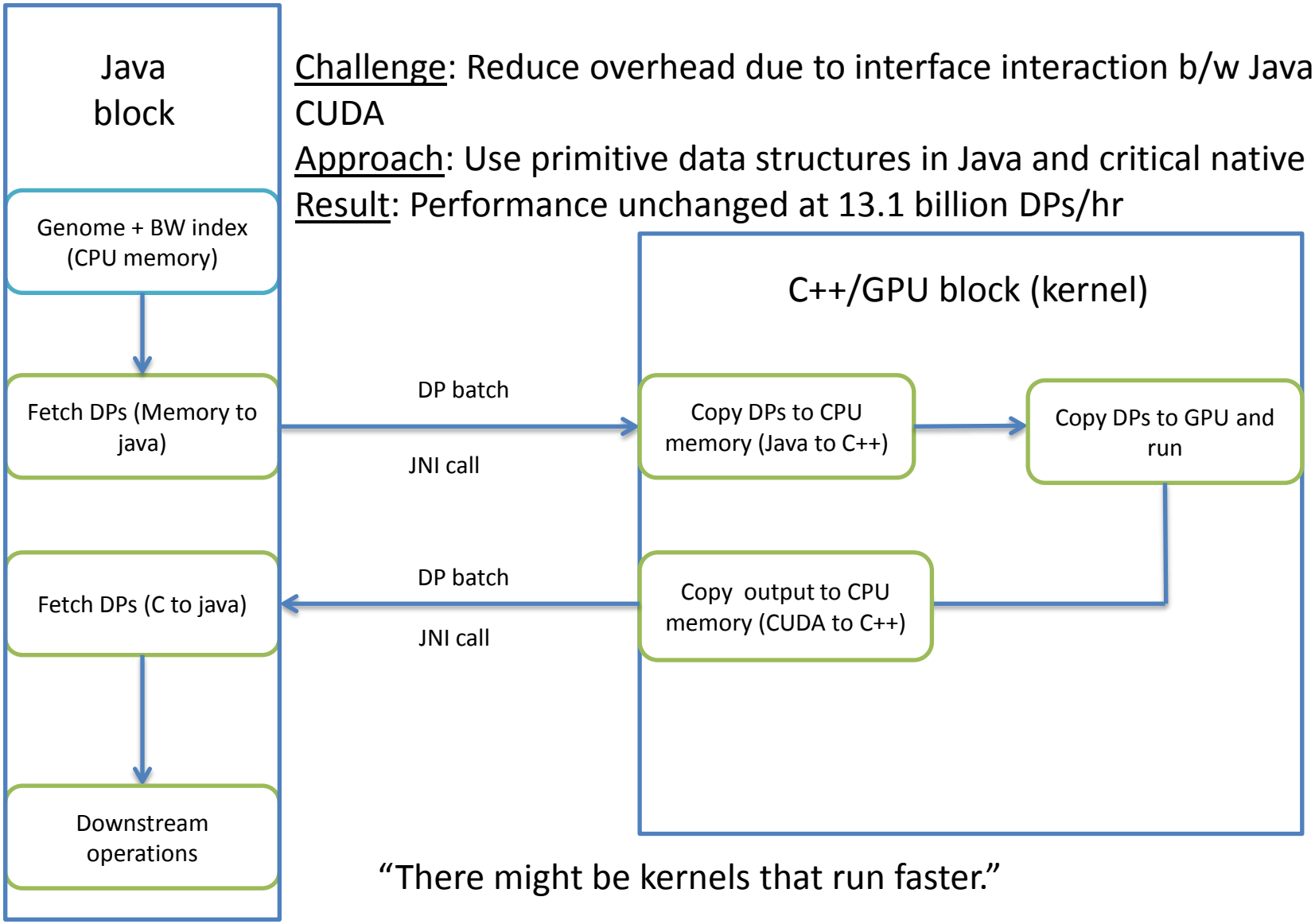
Implementation



Peak rate: 13.1 billion DPs/hr for read length = 100 and gap size = 5

Bottom line: SW 2xGPU is 105x CPU.

Implementation



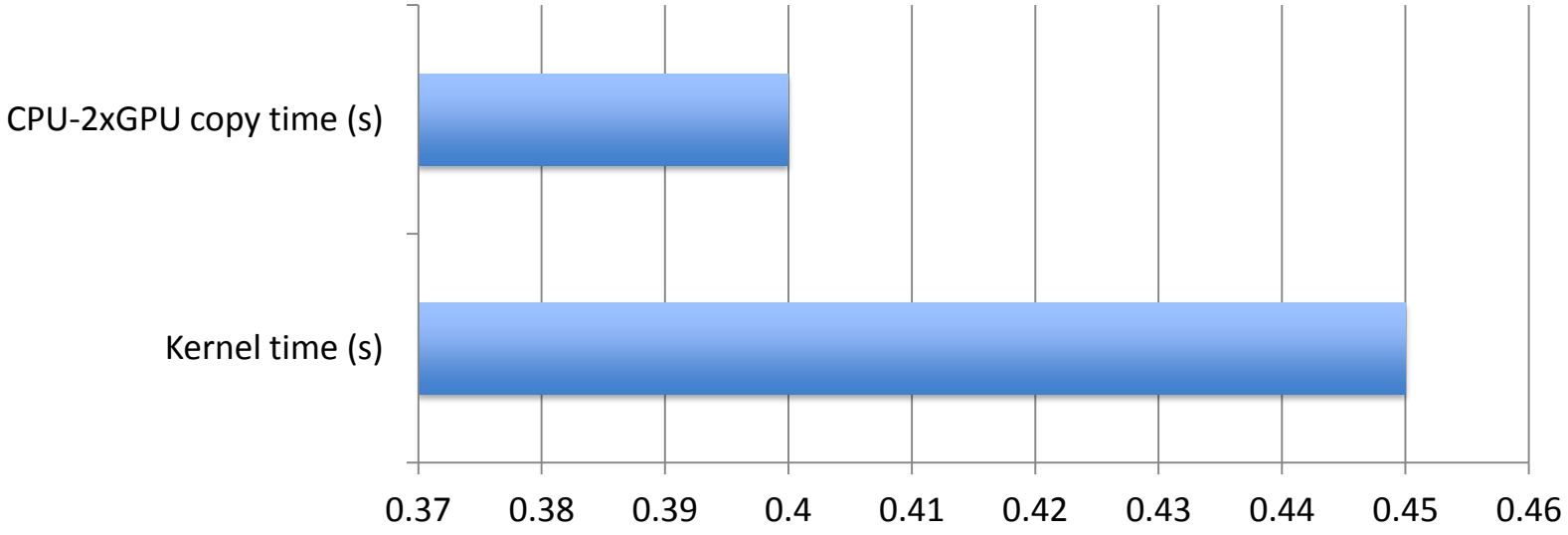
Challenge: Reduce overhead due to interface interaction b/w Java and CUDA

Approach: Use primitive data structures in Java and critical native calls

Result: Performance unchanged at 13.1 billion DPs/hr

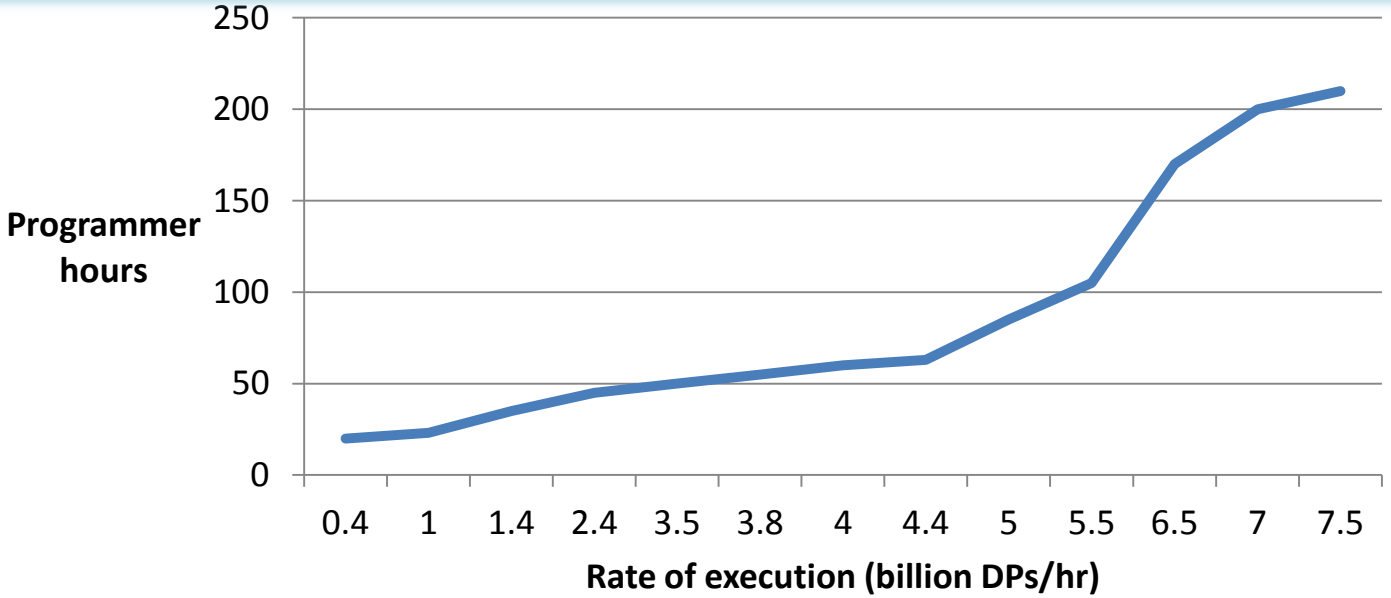
"There might be kernels that run faster."

Lessons and Provisos



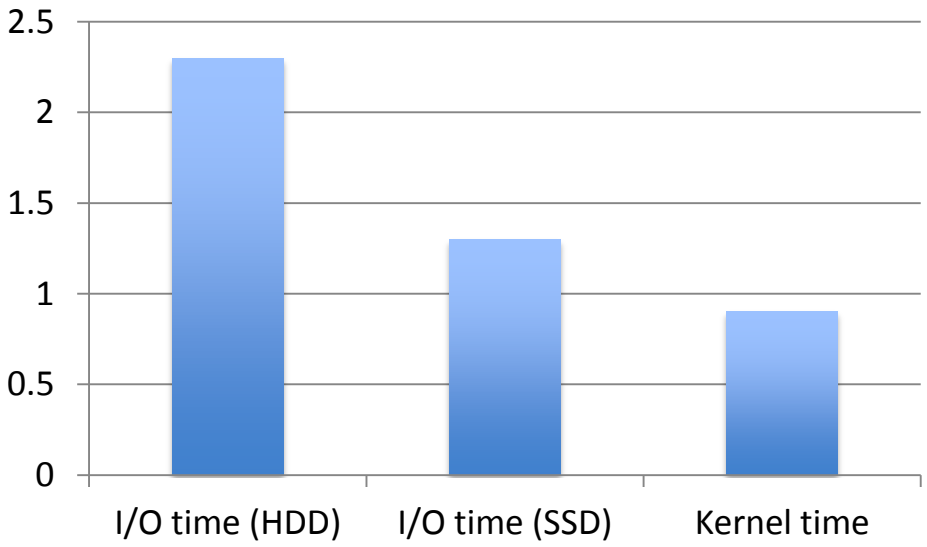
- Can’t get much better (Limited by GPU-CPU bandwidth)

Lessons and Provisos



- GPU isn't cure-all
  - Since core algorithms haven't changed, tweaking implementation on GPU requires many "clever" changes)

Lessons and Provisos



- I/O limitations are dire

"Que cela manque souvent d'architecture."

Thanks!