Possible issues to optimize stochastic simulation time with parallel sequences and unrolling techniques

David R.C. Hill and Romain Reuillon
romain.reuillon@isima.fr
LIMOS – UMR CNRS 6158
Université Blaise Pascal
ISIMA, Campus des Cézeaux BP 10125
63177 Aubière Cedex
FRANCE

Parallelism in Monte Carlo Applications

- Computationally intensive but naturally parallel
- Appropriate for independent *bag-of-work* paradigm
- Fits the distributed computing paradigm
- Requirements :

  Independence of underlying random number streams

  SPRNG

    (Scalable Parallel Random Number Generation) library

« Random number generators, particularly for parallel computers, should not be trusted. »

Paul Coddington

- It is necessary that RNG can be generated in parallel (ie each process may have an autonomous access to a sub-sequence issued from a common global sequence (for correlation problems))

- If such an autonomy is not guaranteed, the potential parallelism of the application is affected (if for instance processes access to a central RNG, even if this generator is also run in parallel)

- The main problem is to find partitioning techniques which preserve the good properties required to guarantee not only the efficiency of the simulation but mainly the credibility of the results (L'Ecuyer proposed an interesting approach in 2001)

Requirements

- *prop. 1 :* numbers are uniformly generated
- *prop. 2 :* the sequence is uncorrelated
- *prop. 3 :* the sequence is reproducible
- *prop. 4 :* the generator is portable on any computer
- *prop. 5 :* the sequence can be changed by adjusting a seed or a status
- *prop. 6 :* the period is as large as possible
- *prop. 7 :* the generator satisfy any randomness test
- *prop. 8 :* a quick generation is obtained
- *prop. 9 :* the generator uses a limited computer memory

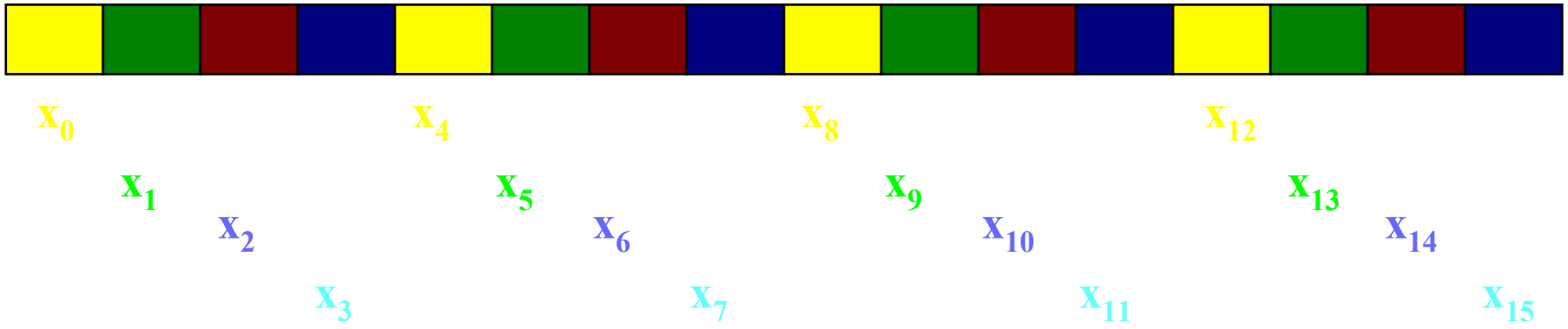
Requirements for Parallel RNGs

- *prop. 10 :* it is easy to split the numbers into many independent sub-sequences that are allocated to different LPs, without the need of communication or synchronization; each sub-sequence is a good sequential RNG
- *prop. 11 :* there is no correlation between the sub-sequences on different LPs (crossed-correlation)

Parallelizing RNGs : partitioning methods

- The LF (Leap Frog) technique:
  partitioning a sequence $\{x_i, i=0, 1, \ldots\}$ into 'n' sub-sequences, the j^{th} sub-sequence is $\{x_{kn+j-1}, k=0, 1, \ldots\}$
- The SS (Sequence Splitting or Cycle Splitting) technique:
  - Partitioning a sequence $\{x_i , i=0, 1, \ldots,\}$ into 'n' sub-sequences
    The j^{th} sub-sequence is $\{x_{k+(j-1)m} , k=0, \ldots, m-1\}$
    where m is the length of each sub-sequence
  - The user deterministically chooses widely separated seeds in same generator.
- The IS (Independent Sequences or Cycle Splitting) technique:
  - Using randomly generated seeds
  - Using different parameters for the same kind of generator (example Matsumoto and Nishimura MT) => need theoretical studies of mathematical properties at first.

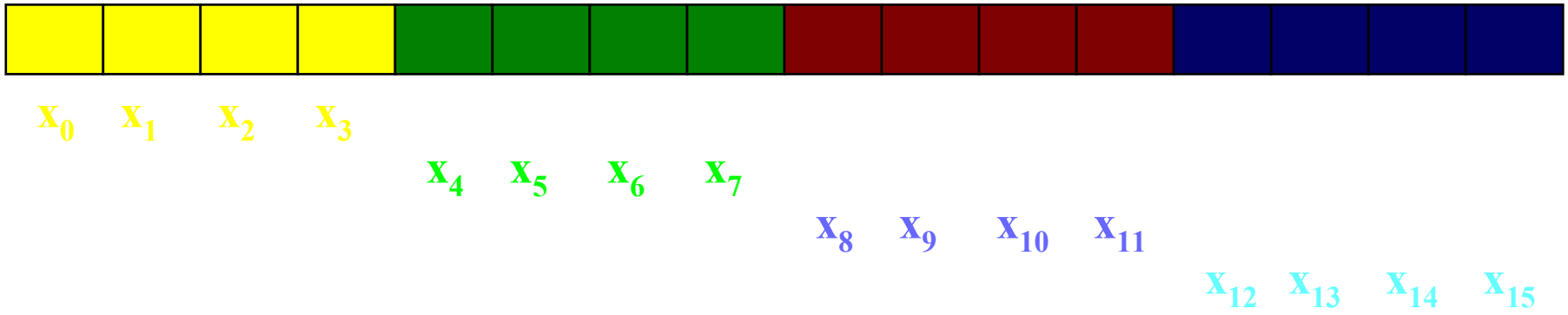
Leap Frog Technique



"Long-Range"correlation problem:

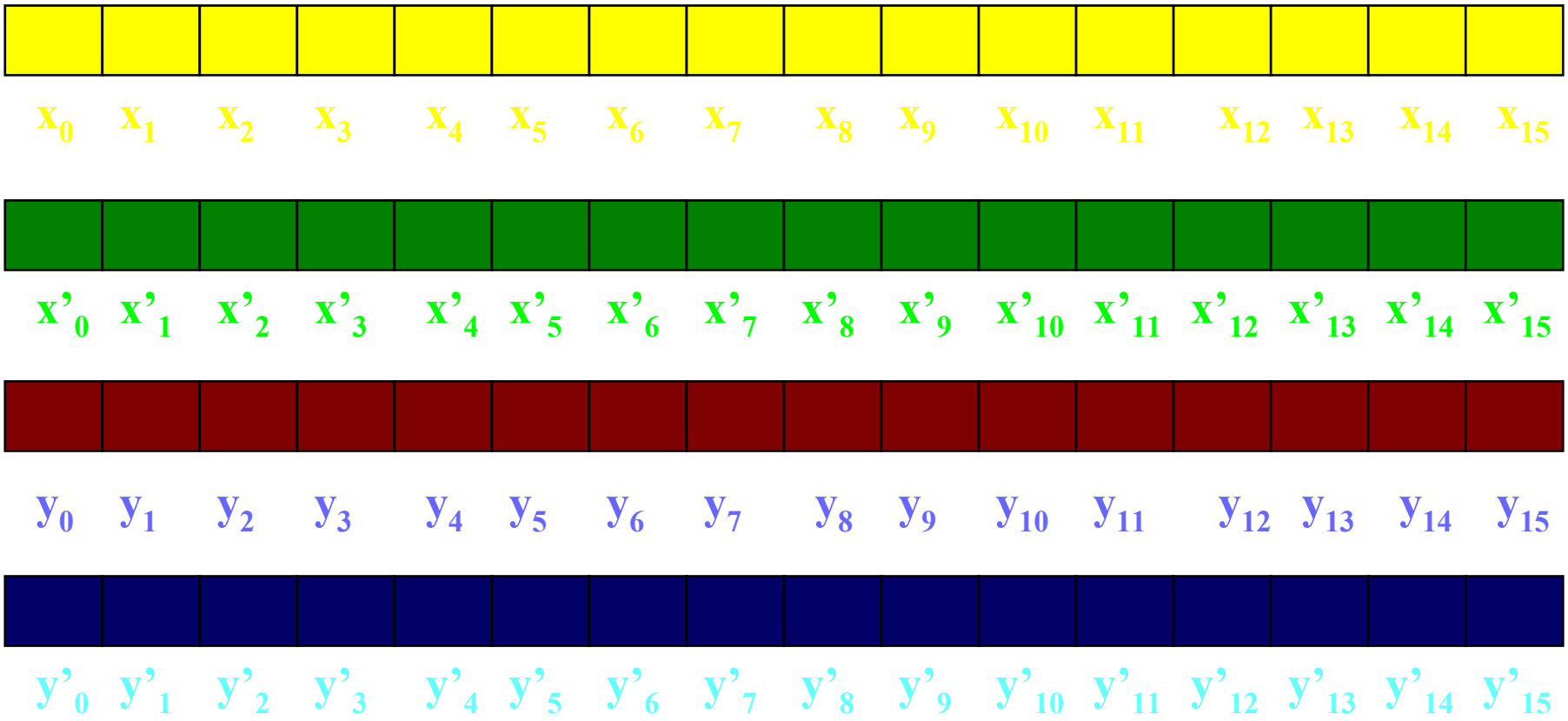
auto-correlation

Sequence splitting technique

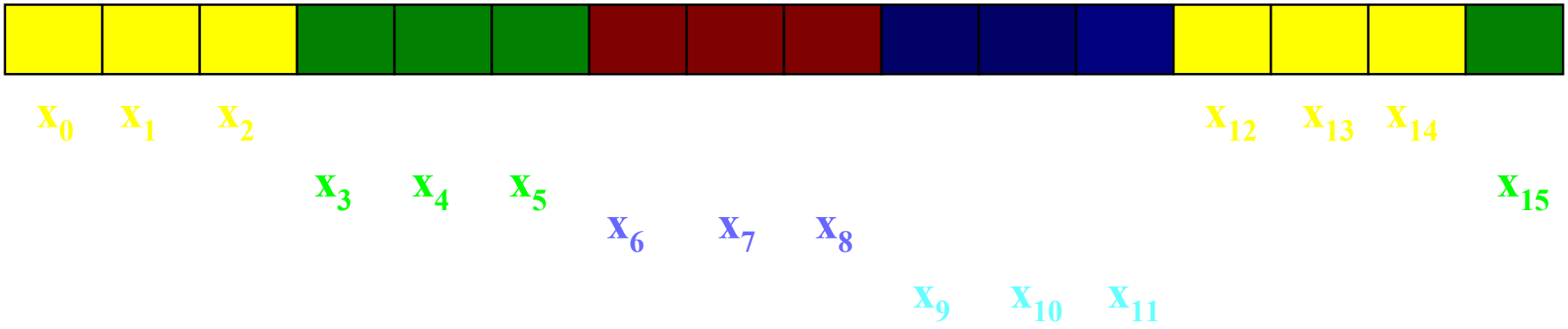


"Long-Range" correlation problem :
Crossed-correlation

Independent sequences technique



Hybrid technique



Hybridizing LF and SS:
partitioning a sequence into ‘n’ sub-sequences,
the jth sub-sequence is {x(kn+j-1)m+s, k=0, … and s=0, …, m-1}
where m is the length of contiguous sub-blocks.

Reducing "Long-Range" correlations

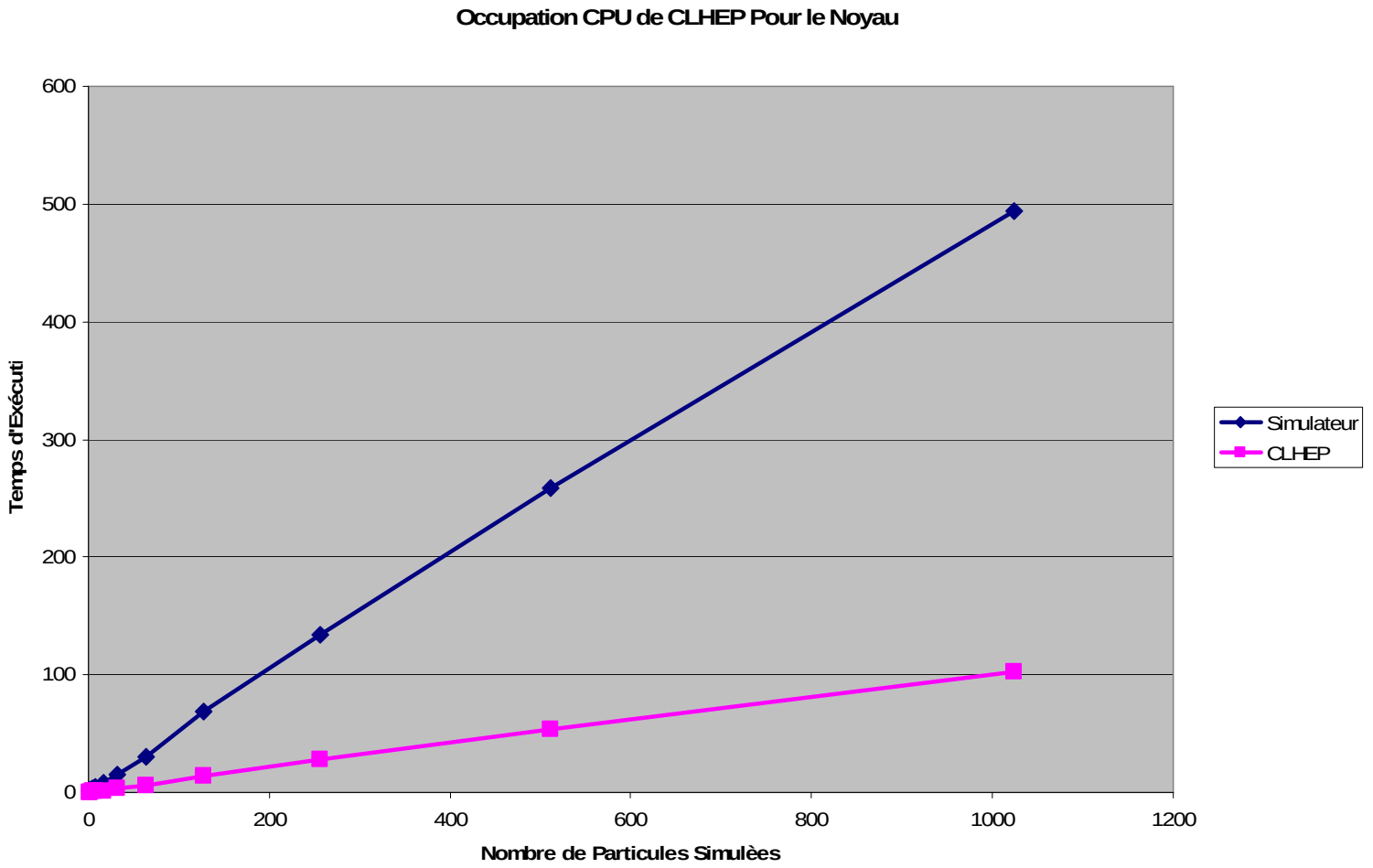
Parallelizing PRNGs : tests

- Theoretical tests:
  the random sequences should have the properties of a random sample drawn from the uniform distribution.
  (Series of Knuth, DieHard tests, L'Ecuyer TestU01)

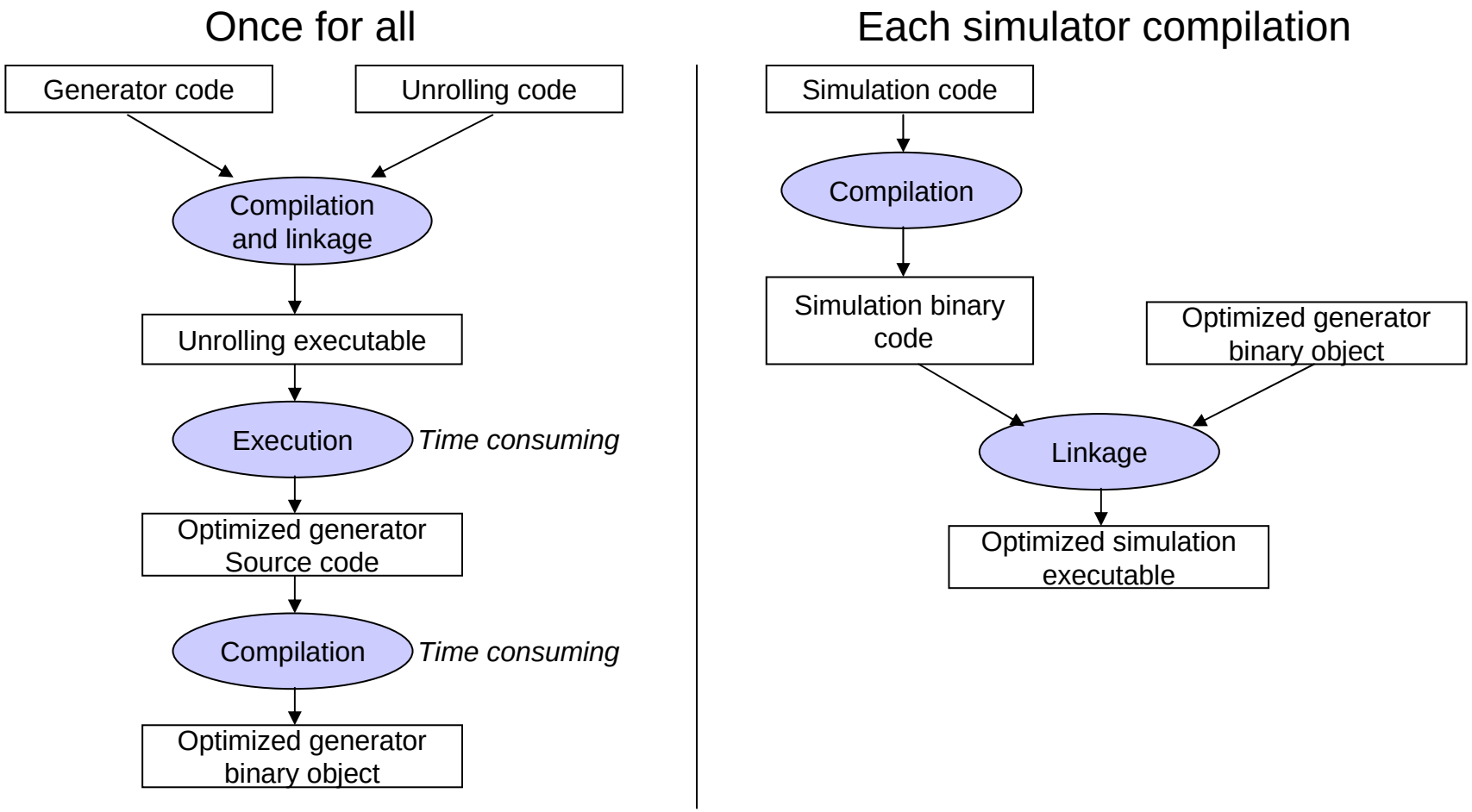
- Empirical tests:
  As stated by Paul Coddington in 1996 : "*It is strongly recommended that all simulations be done with two or more different generators, and the result compared to check whether the random number generator is introducing a bias* "

The classical approach to test PRNG is to test separately each sub-sequence, and then to test the entire sequence generated

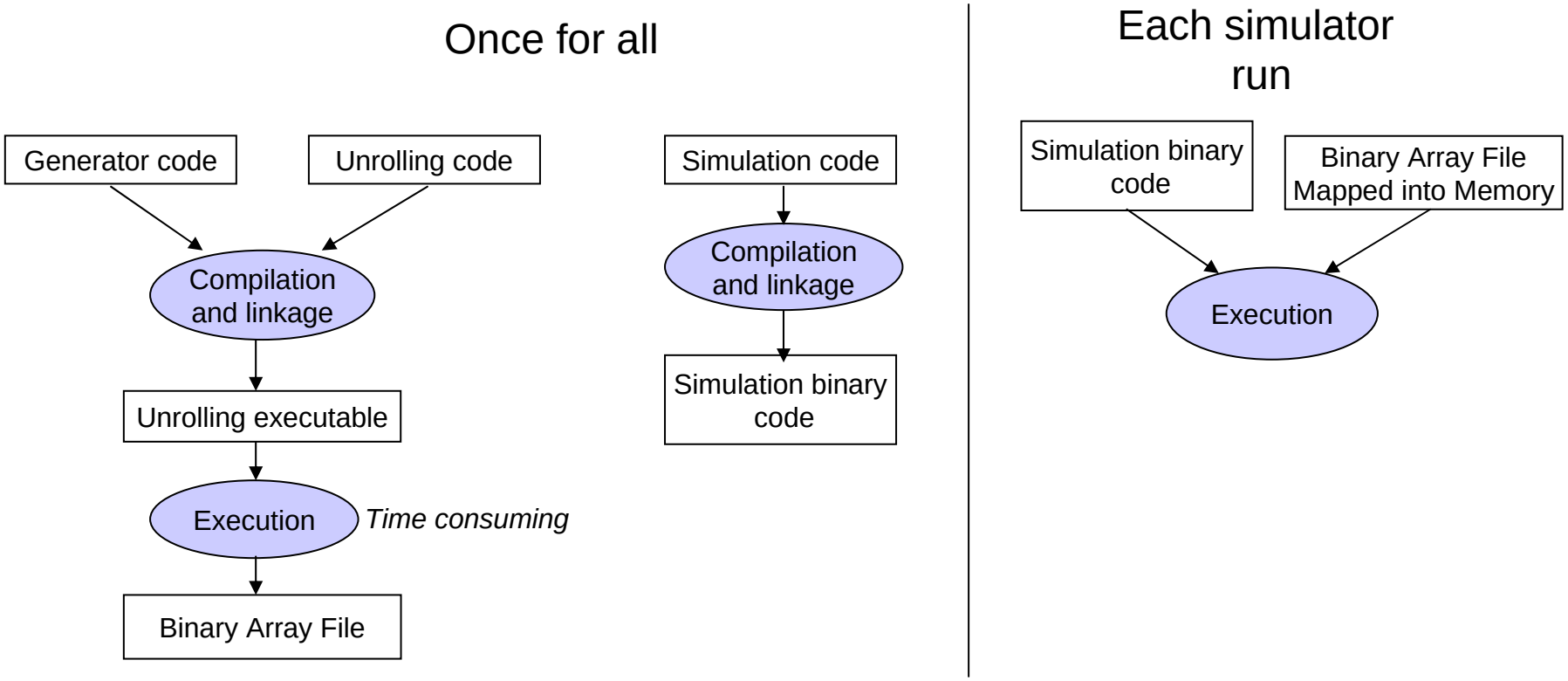
Random number generation in Geant4 : up to 20% of running time



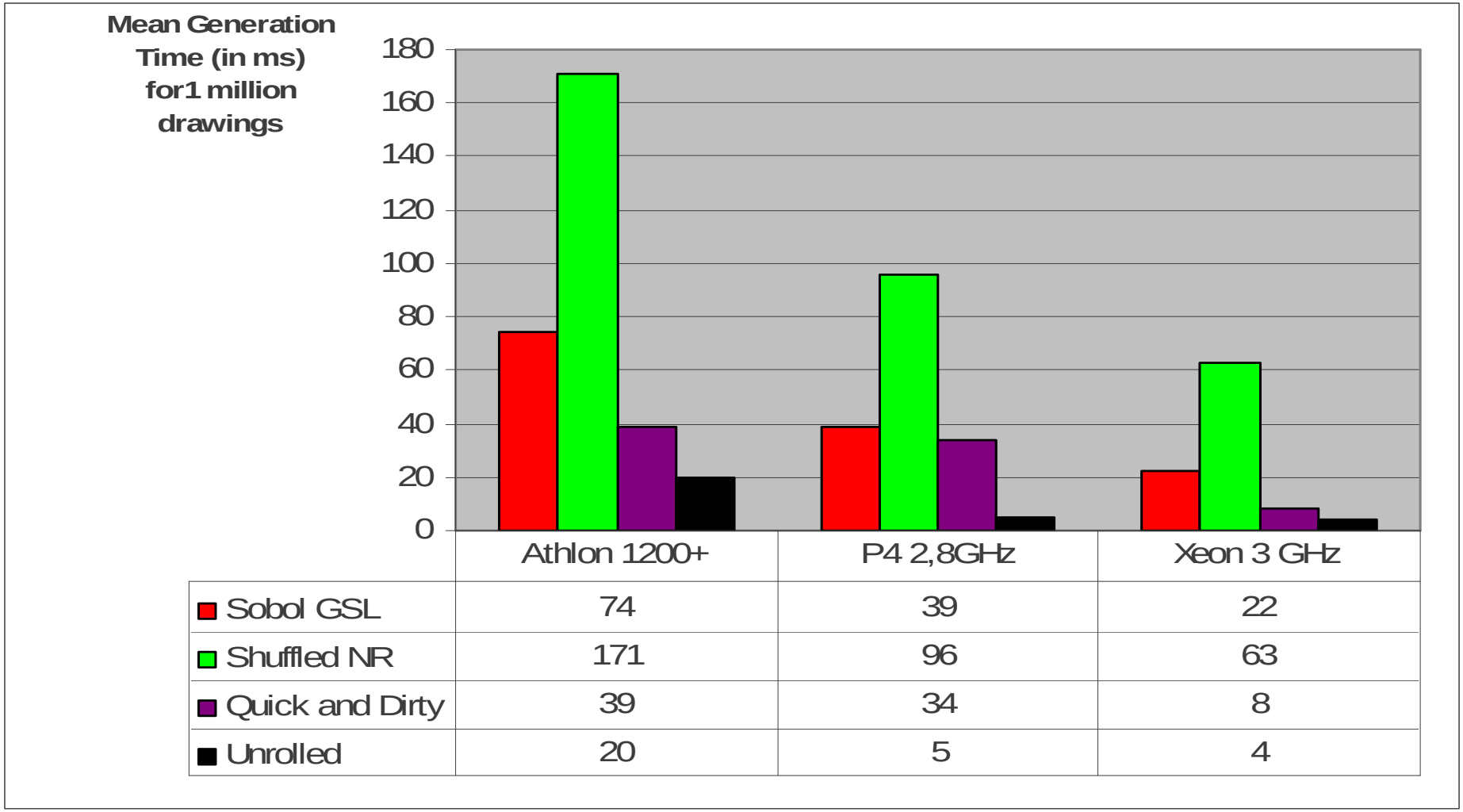
A speedup technique faster than getting RN from hardware cards : metaprogramming by « unrolling » : URNG



Metaprogramming useless for files larger than RAM size => Unrolling with « Memory Mapping »



Comparative tests under Linux on 3 common generators and the unrolled version



	Athlon 1200+	P4 2,8GHz	Xeon 3 GHz
Sobol GSL	74	39	22
Shuffled NR	171	96	63
Quick and Dirty	39	34	8
Unrolled	20	5	4

Conclusion

- Acceleration of Monte Carlo methods
  - Variance Reduction
  - Quasi Monte Carlo
  - Optimising number generation
  - Parallel Computing
- The best option to reduce Geant4 computing time is parallel computing

  Be careful with random number generation !!!!