# Detector Simulation on Modern Processors

## Vectorization of Physics Models

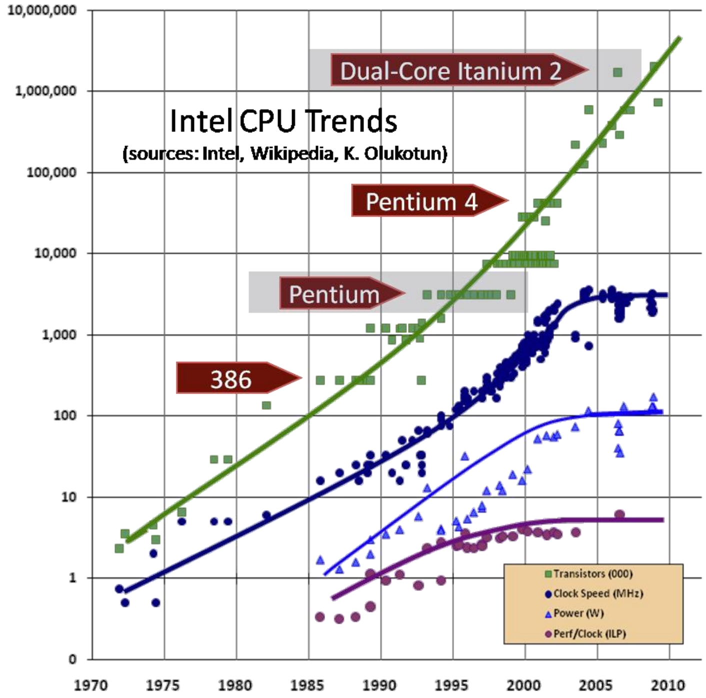
Philippe Canal, Soon Yung Jun (FNAL)
John Apostolakis, Mihaly Novak, Sandro Wenzel (CERN)
*for the GeantV team*

CHEP, Apr 13 - 17, 2015, Okinawa, Japan

# Contents

# Introduction

- Motivations
  - Performance of our code scales with clock cycle
  - HEP code needs to exploit new architectures to improve
  - Data & instruction locality and vectorisation
  - Portability, better physics and optimization will be the targets



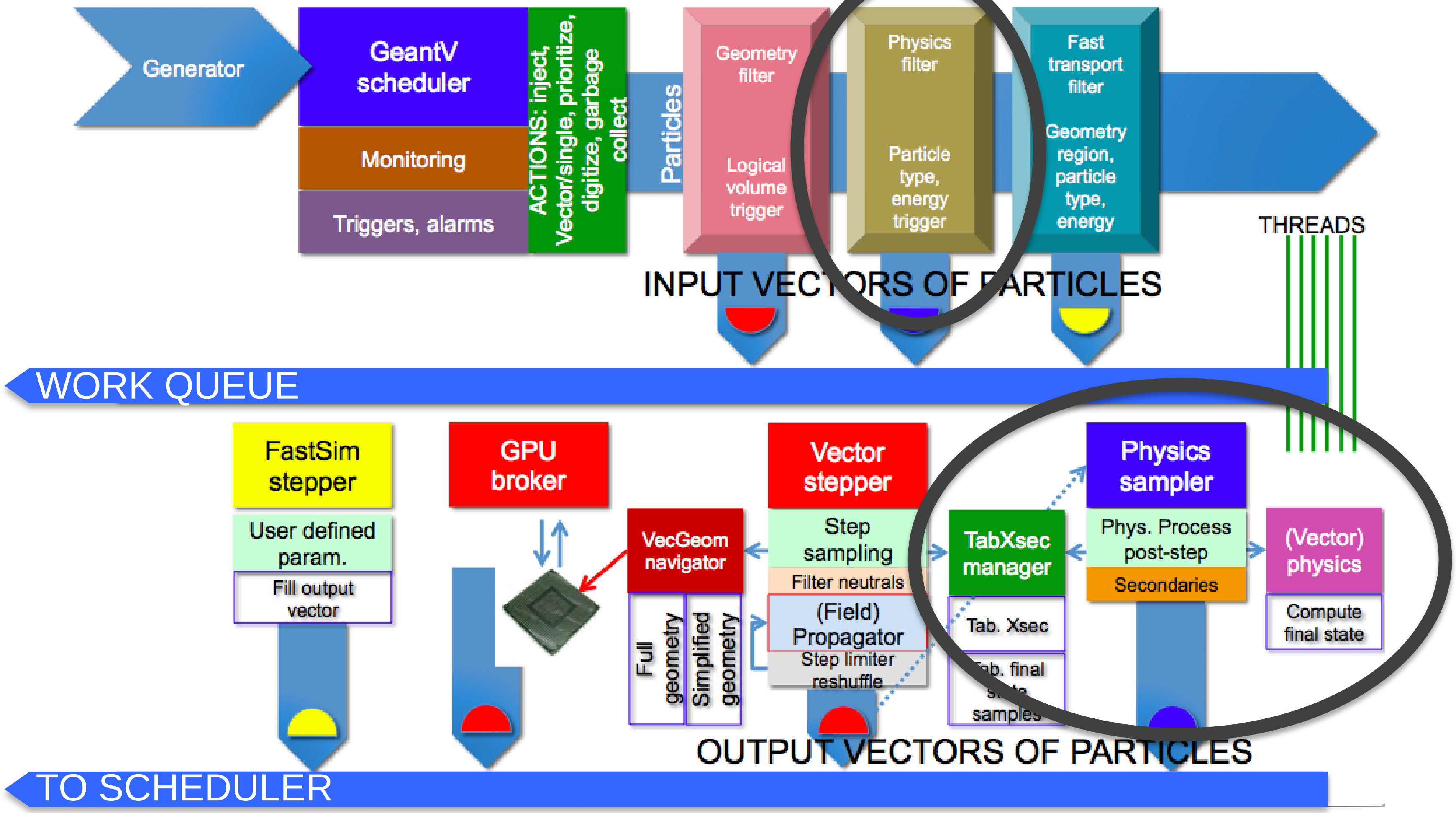
# Introduction

- GeantV Goals
  - Develop an all-particle transport simulation program
    - 2 to 5 times faster than Geant4
    - Continues improvement of physics
    - Full simulation and various options for fast simulation
    - Portable on different architectures, including accelerators (GPUs and Xeon Phi's)
  - Understand the limiting factors for 10x improvement

See The GeantV project: preparing the future of simulation on **14 Apr 2015** at **17:15**

# GeantV: The next generation detector simulation toolkits

- The GeantV framework: scheduling, geometry, ***physics***



Generator

GeantV scheduler

Monitoring

Triggers, alarms

ACTIONS: inject, Vector/single, prioritize, digitize, garbage collect

Particles

Geometry filter

Logical volume trigger

Physics filter

Particle type, energy trigger

Fast transport filter

Geometry region, particle type, energy

THREADS

INPUT VECTORS OF PARTICLES

WORK QUEUE

FastSim stepper

User defined param.

Fill output vector

GPU broker

VecGeom navigator

Full geometry

Simplified geometry

Vector stepper

Step sampling

Filter neutrals

(Field) Propagator

Step limiter reshuffle

TabXsec manager

Tab. Xsec

Tab. final state samples

Physics sampler

Phys. Process post-step

Secondaries

(Vector) physics

Compute final state

OUTPUT VECTORS OF PARTICLES

TO SCHEDULER

# Vector Physics Model

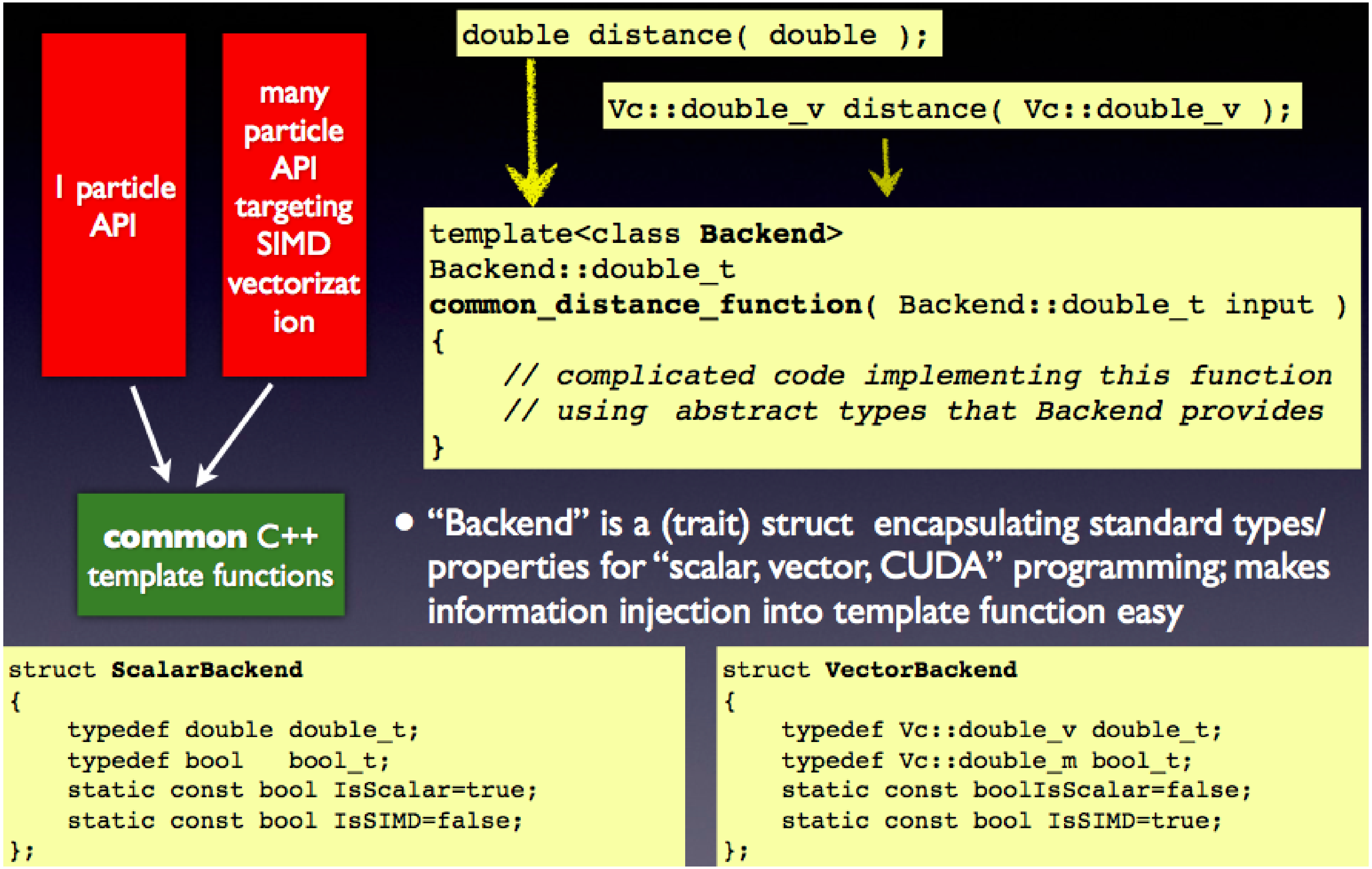
- Assumption: particles are independent during tracking
- Vectorization of the density of collisions, ψ

$$\Psi = \sum_{k=1}^{\mathrm{ntracks}} \Psi_k(\text{sequential iteration}) \rightarrow \prod_{k=1}^{\mathrm{ntasks}} \Psi_k(\text{task}-\text{based execution})$$

$$\Psi_k(\vec{r},\vec{v}) = \int d\vec{r'}[S_k(\vec{r'},\vec{v}) + \int \Psi_k(\vec{r'},\vec{v'}) I_k(\vec{r'},\vec{v'} \rightarrow \vec{v}) d\vec{r'}] T_k(\vec{r'} \rightarrow \vec{r},\vec{v})$$

  - $S_k$ (source): vector scheduler
  - $T_k$ (transport): VectGeom + vector propagator (geometry limited step)
  - $I_k$ (interaction): vectorized physics (step length, secondary production, ...)
- Vector strategies: data locality and instruction throughput
  - decomposition sequential tracking and regroup them by tasks
  - algorithmic vectorization and parallel data patterns
  - targeting both external and internal (SIMD) vectorization

# Portability (Template Approach): Scalar, Vector, CUDA, MIC



I particle API

many particle API targeting SIMD vectorization

common C++ template functions

```
double distance( double );
```

```
Vc::double_v distance( Vc::double_v );
```

```
template<class Backend>
Backend::double_t
common_distance_function( Backend::double_t input )
{
    // complicated code implementing this function
    // using  abstract types that Backend provides
}
```

- “Backend” is a (trait) struct  encapsulating standard types/ properties for “scalar, vector, CUDA” programming; makes information injection into template function easy

```
struct ScalarBackend
{
    typedef double double_t;
    typedef bool   bool_t;
    static const bool IsScalar=true;
    static const bool IsSIMD=false;
};
```

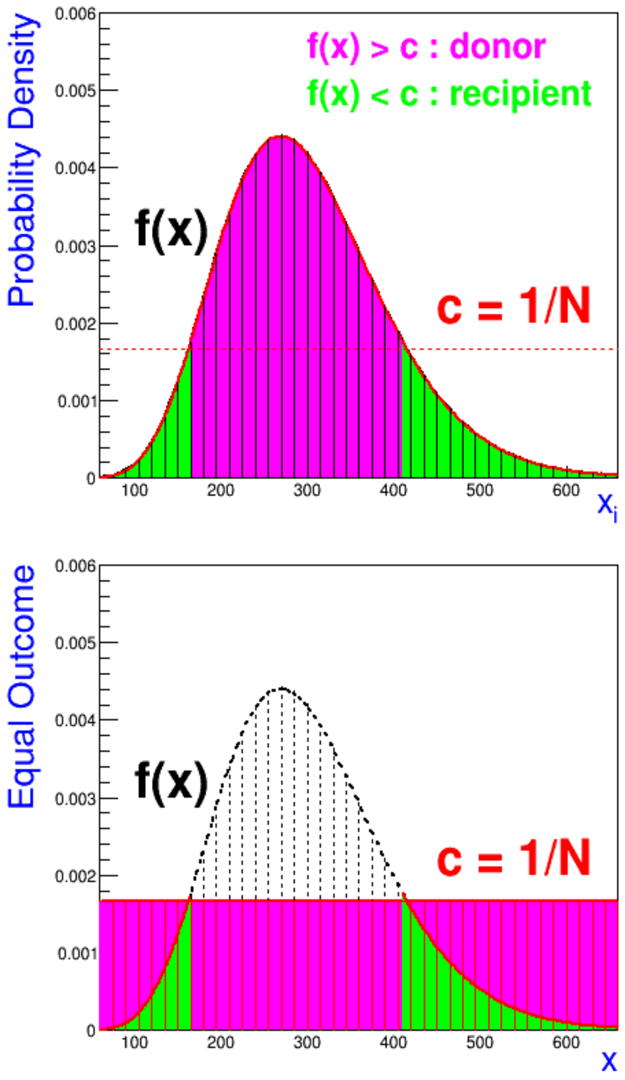
```
struct VectorBackend
{
    typedef Vc::double_v double_t;
    typedef Vc::double_m bool_t;
    static const boolIsScalar=false;
    static const bool IsSIMD=true;
};
```

# Prerequisites to Achieve Efficient Vectorization

- Vectorized pseudo-random number generator
- Data layout: coalesced memory access on vector operands
  - SoA (struct of array) tracks parameters (**x**,**p**,t,E …)
  - ordered and aligned data arrays
- Data locality for the vector of particles
  - particle type, geometry and material, physics process
- Vector operations
  - identical instructions on each components of the vector
  - no conditional branches,  no data dependencies
  - replace non-vectorizable algorithms (ex. composition and rejection methods) by alternatives

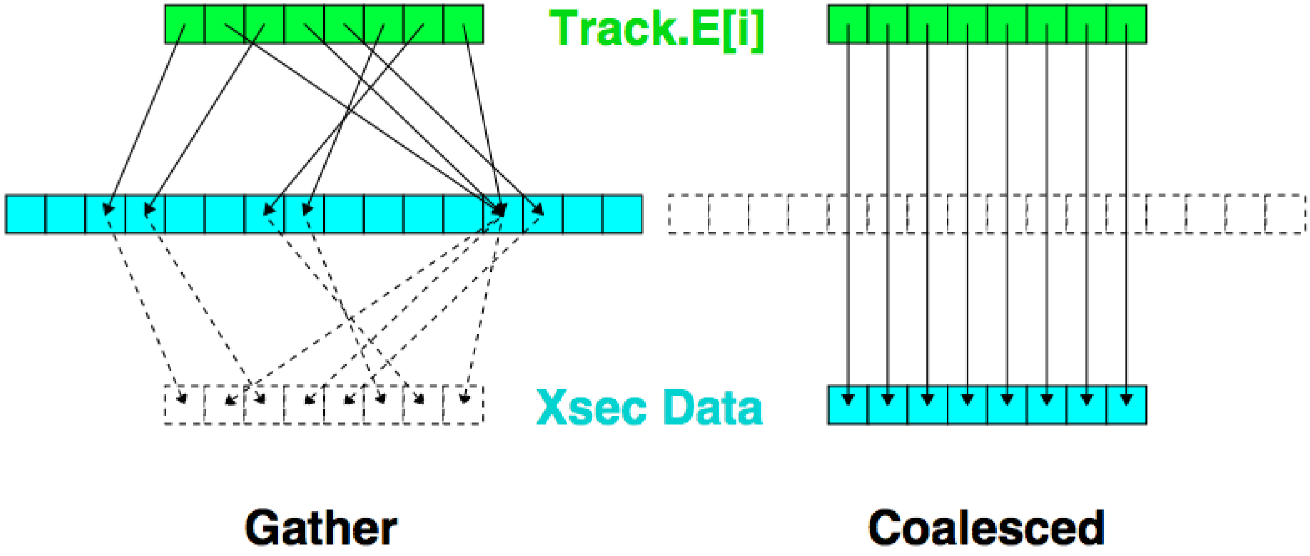
# Sampling Secondary Particles: Alias Method (A.J.Walker)

- Recast a cross section, f(x) to N equal probable events, each with likelihood c = 1/N
- Alias table
  - a[recipient] = donor
  - q[N] = non-alias probability
- Sampling $x_j$ : random $u_1$, $u_2$
  - bin index: $N \times u_1 = i + \alpha$
  - sample $j = (q[i] < u_2)\ ?\ i : a[i]$
  - $x_j = [\alpha\, j + (1-\alpha)\,(j+1)]\delta x$
- Replace composition and rejection methods (conditional branches – not vectorizable)



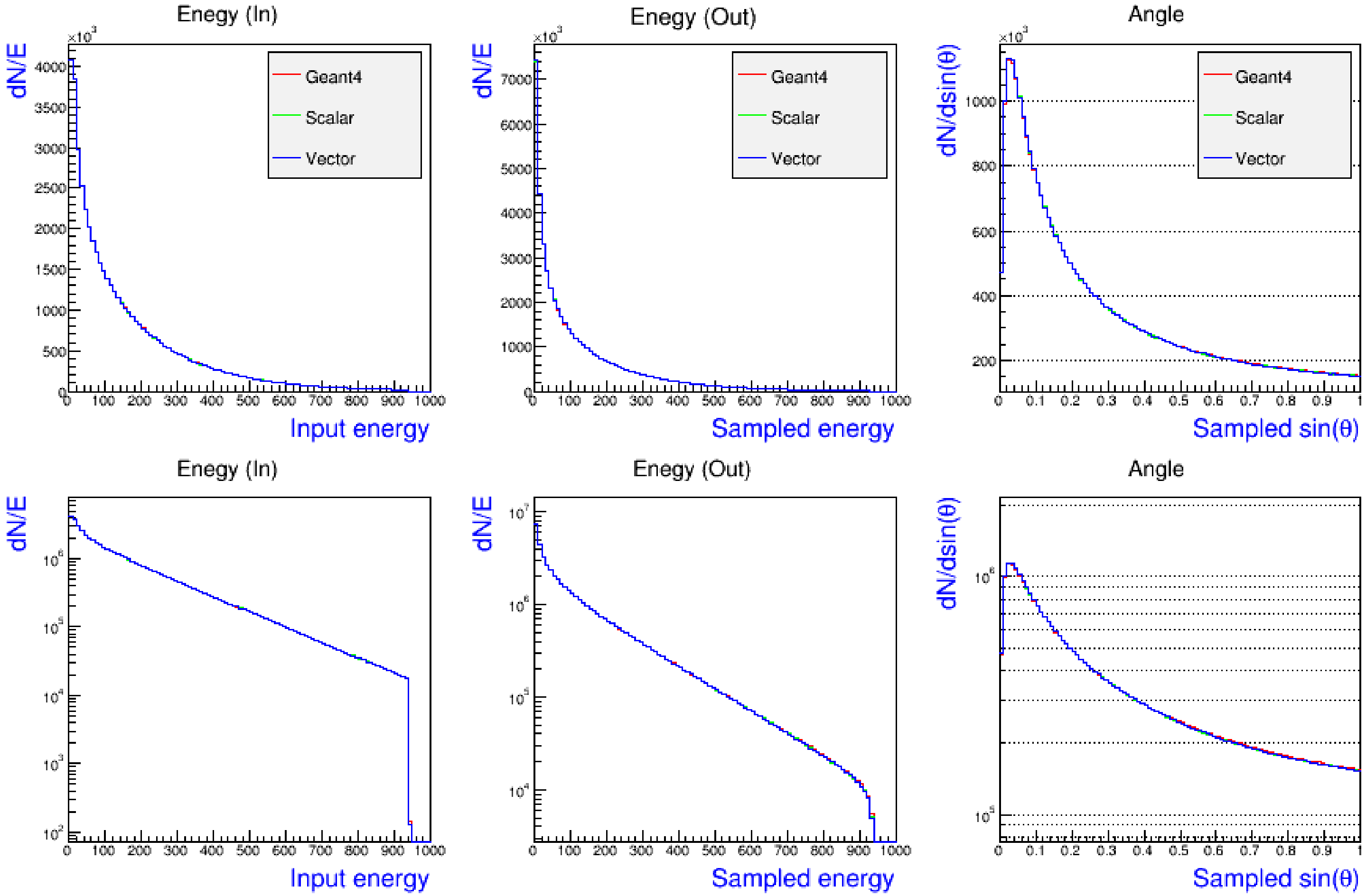
# Coalesced Memory Access

- Sampling the step length and the physics process
  - cross section calculation on-the-fly (fully vectorizable, likely expensive)
  - tabulated physics (table-lookups, bandwidth limited)
- Gather data to enable contiguously ordered accesses
  - loss by overhead < gain by vectorization

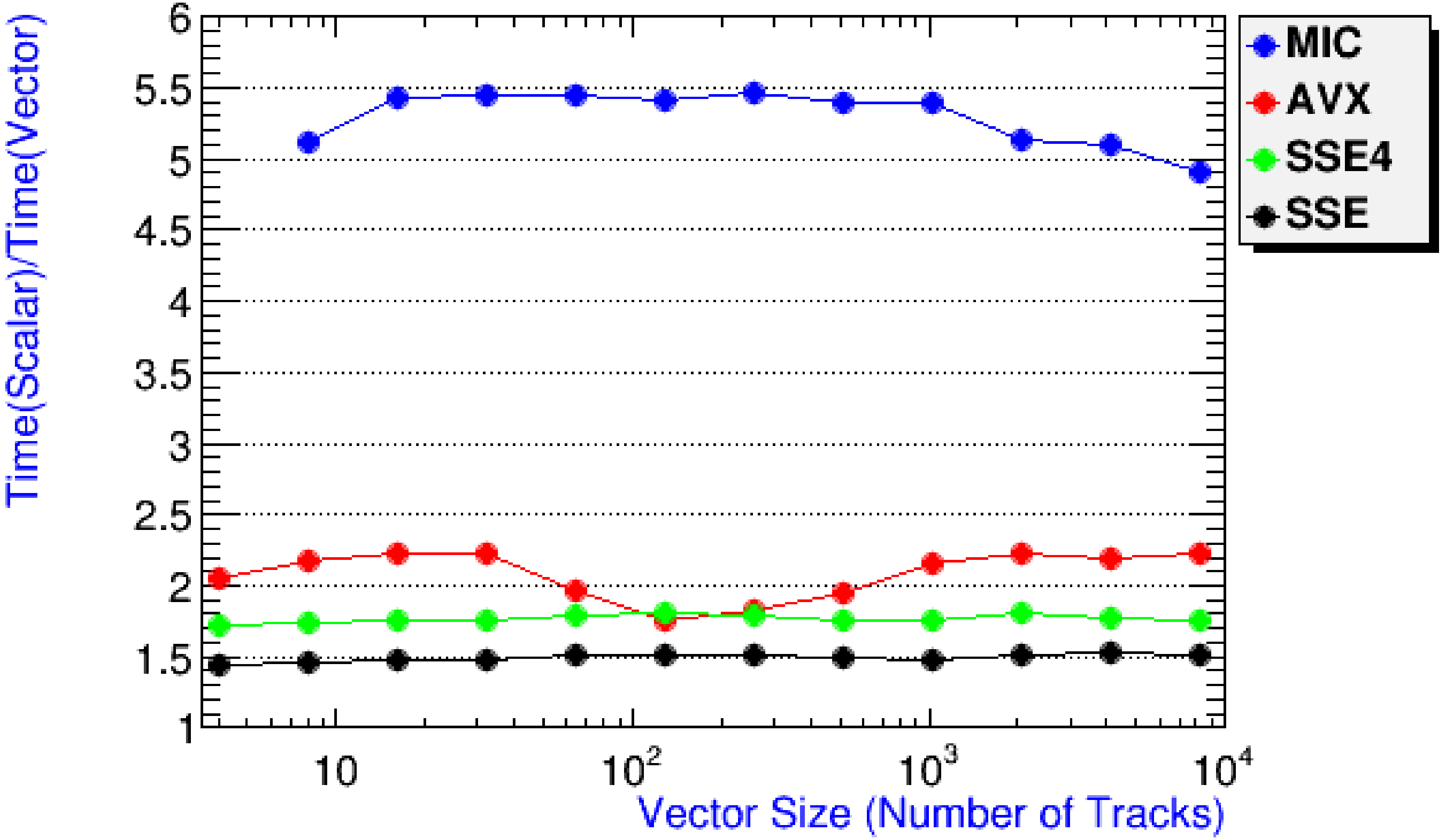


# Validation: Alias vs. Composition and Rejection Method

- Compton (Klein-Nishina model): energy and angle of scattered photons



# Vector Speedup: Factor 2 on Xeon



# Runtime Performance

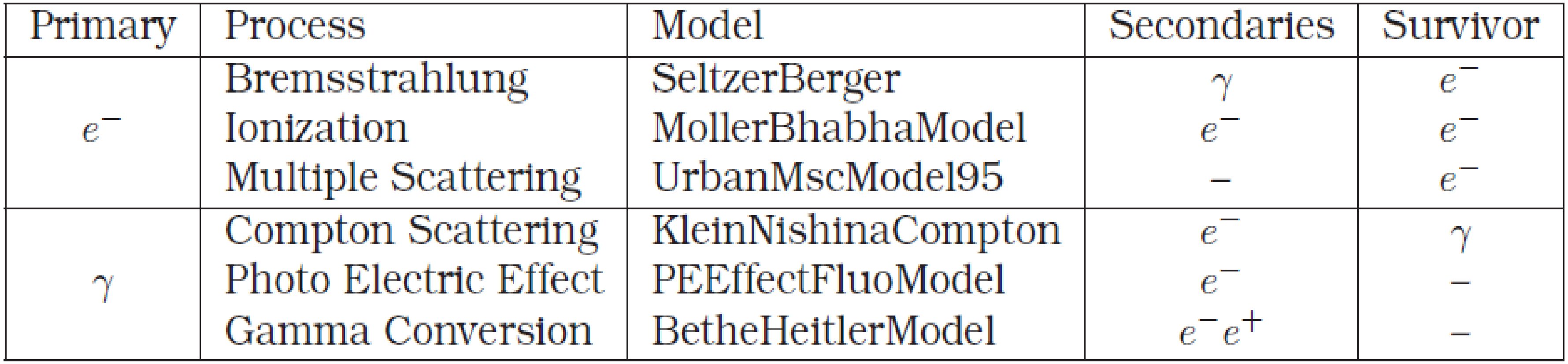
- Relative performance for sampling the secondary electron
  - Composition Method, Scalar, Vector
  - average time for 100 trials for 4992x100 tracks – SSE
  - Table size [input energy bins, sample energy bins]

| Table size | Time with [100,100] | Time with [100,1000] |
|---|---|---|
| Composition Method | 11.609 | 11.347 |
| Alias Method, Scalar | 8.439 | 10.080 |
| Alias Method, Vector | 5.446 | 6.185 |

- Note that Composition Method Klein-Nishina model is one of the most efficient composition and rejection examples (ε~1)

# Status and Plan

- Implement one fully vectorized EM physics model (Klein Nishina Compton) and test with GeantV
  - Backend: Scalar, Vector, CUDA
  - Performance evaluation and validation
- Complete all EM physics by Dec. 2015



| Primary | Process | Model | Secondaries | Survivor |
|---|---|---|---|---|
| $e^-$ | Bremsstrahlung | SeltzerBerger | $\gamma$ | $e^-$ |
|  | Ionization | MollerBhabhaModel | $e^-$ | $e^-$ |
|  | Multiple Scattering | UrbanMscModel95 | – | $e^-$ |
| $\gamma$ | Compton Scattering | KleinNishinaCompton | $e^-$ | $\gamma$ |
|  | Photo Electric Effect | PEEffectFluoModel | $e^-$ | – |
|  | Gamma Conversion | BetheHeitlerModel | $e^-e^+$ | – |

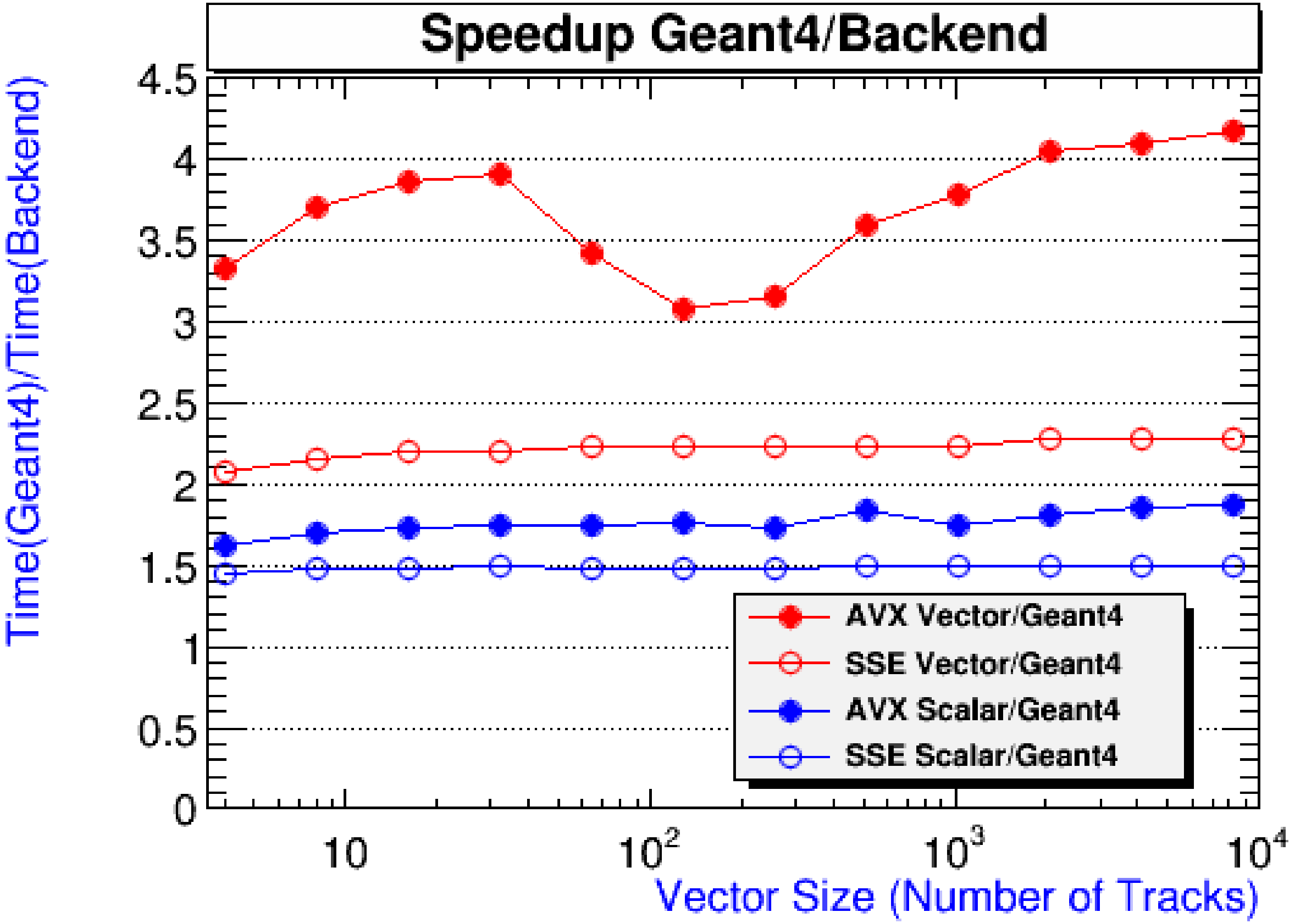
- Extend for hadron physics and explore other algorithms

# Conclusion

- Significant performance improvement achievable in detector simulation physics code using a combination of:
  - Alternative algorithm (reducing branching, etc.)
  - Vectorization
  - Increased use of code and data caches

- Using template techniques, code is portable to different modern computing architectures while still being tuned for each architecture.

# Backup Slides


Speedup Geant4/Backend
Time(Geant4)/Time(Backend)
Vector Size (Number of Tracks)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
10
$10^2$
$10^3$
$10^4$
AVX Vector/Geant4
SSE Vector/Geant4
AVX Scalar/Geant4
SSE Scalar/Geant4