Status and Plans for Geant V prototype

Federico Carminati for the SFT Simulation Project

Annual Concurrency Forum

April 1-2, 2014

The Eight dimensions

- The “dimensions of performance”
  - Vectors
  - Instruction Pipelining
  - Instruction Level Parallelism (ILP)
  - Hardware threading
  - Clock frequency
  - Multi-core
  - Multi-socket
  - Multi-node

Micro-parallelism: gain in throughput and in time-to-solution

Very little gain to be expected and no action to be taken

Gain in memory footprint and time-to-solution but not in throughput

Possibly running different jobs as we do now is the best solution

The Eight dimensions

- The “dimensions of performance”
  - Vectors
  - Instruction Pipelining
  - Instruction Level Parallelism (ILP)
  - Hardware threading
  - Clock frequency
  - Multi-core
  - Multi-socket
  - Multi-node

Micro-parallelism: gain in throughput and in time-to-solution

Very little gain to be expected and no action to be taken

Gain in memory footprint and time-to-solution but not in throughput

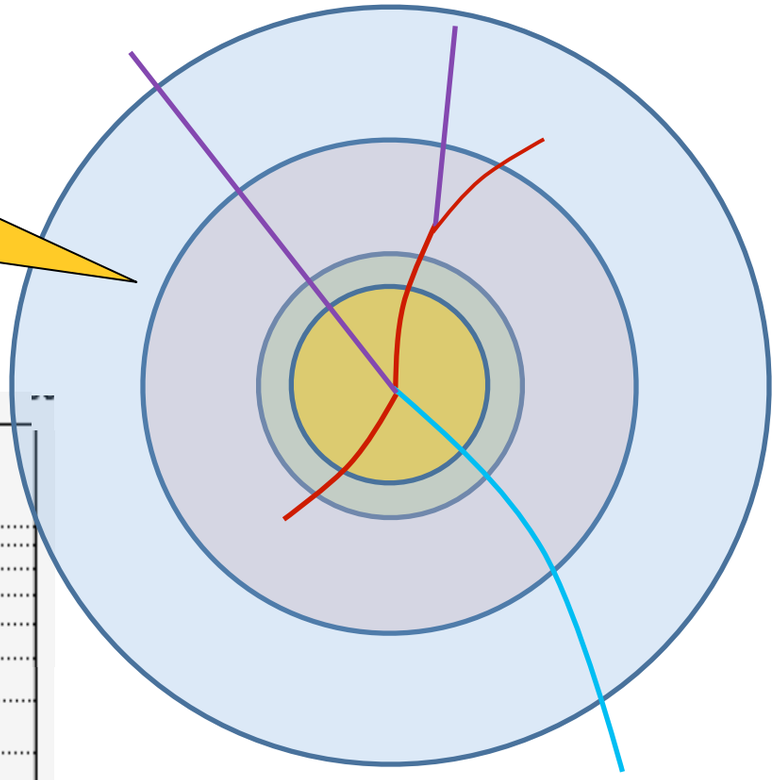
Possibly running different jobs as we do now is the best solution

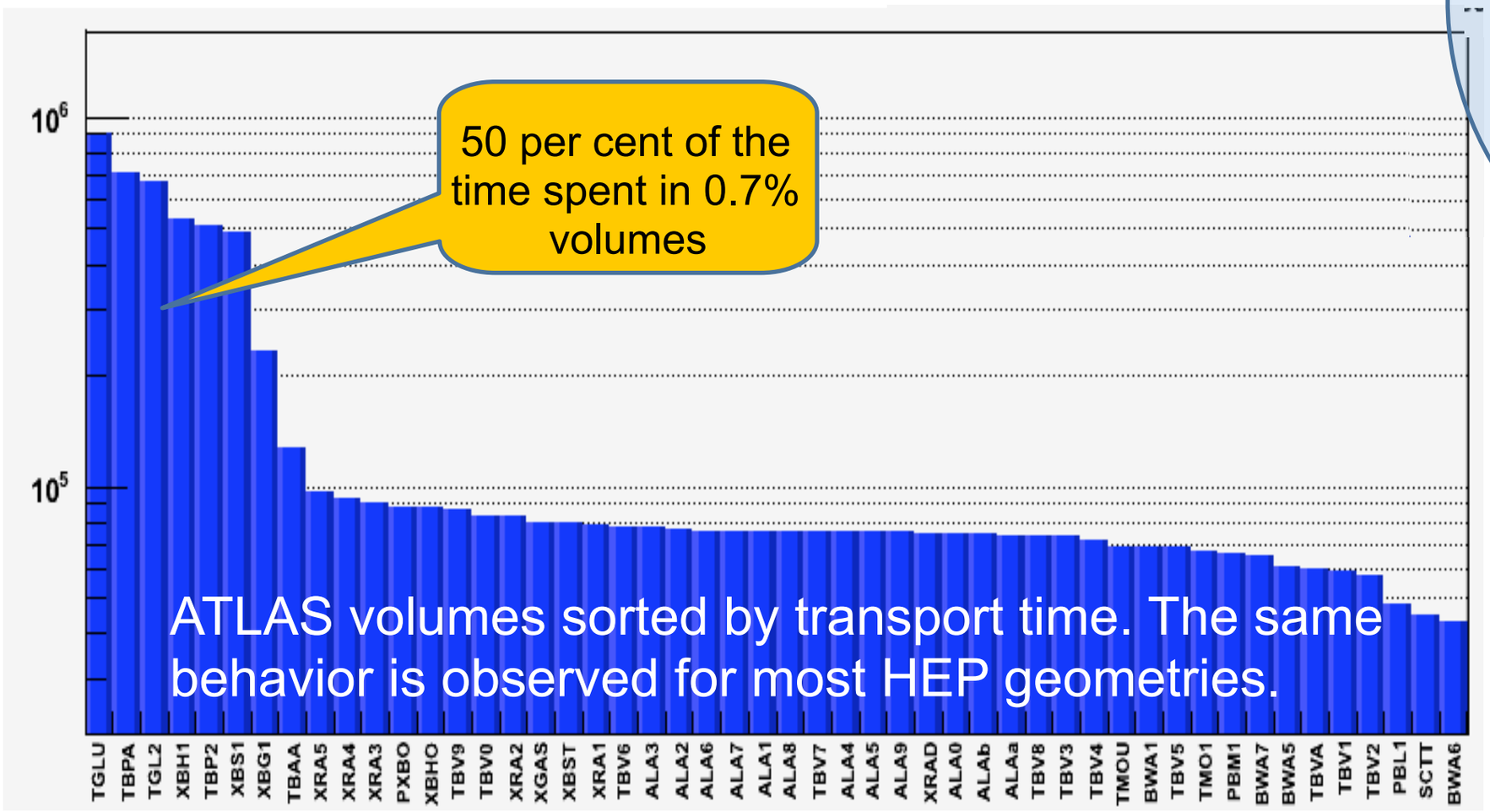
Expected limits on performance scaling			
	SIMD	ILP	HW
THEORY	8	4	1.35
OPTIMISED	6	1.57	1.25
HEP	1	0.8	1.25
Expected limits on performance scaling (multiplied)			
	SIMD	ILP	HW
THEORY	8	32	43.2
OPTIMISED	6	9.43	11.79
HEP	1	0.8	1

Classical HEP transport is mostly local

- Event- or event track-level parallelism will better use resources but won't improve these points

•Geometry navigation (local)
- Material – X-section tables
- Particle type - physics processes



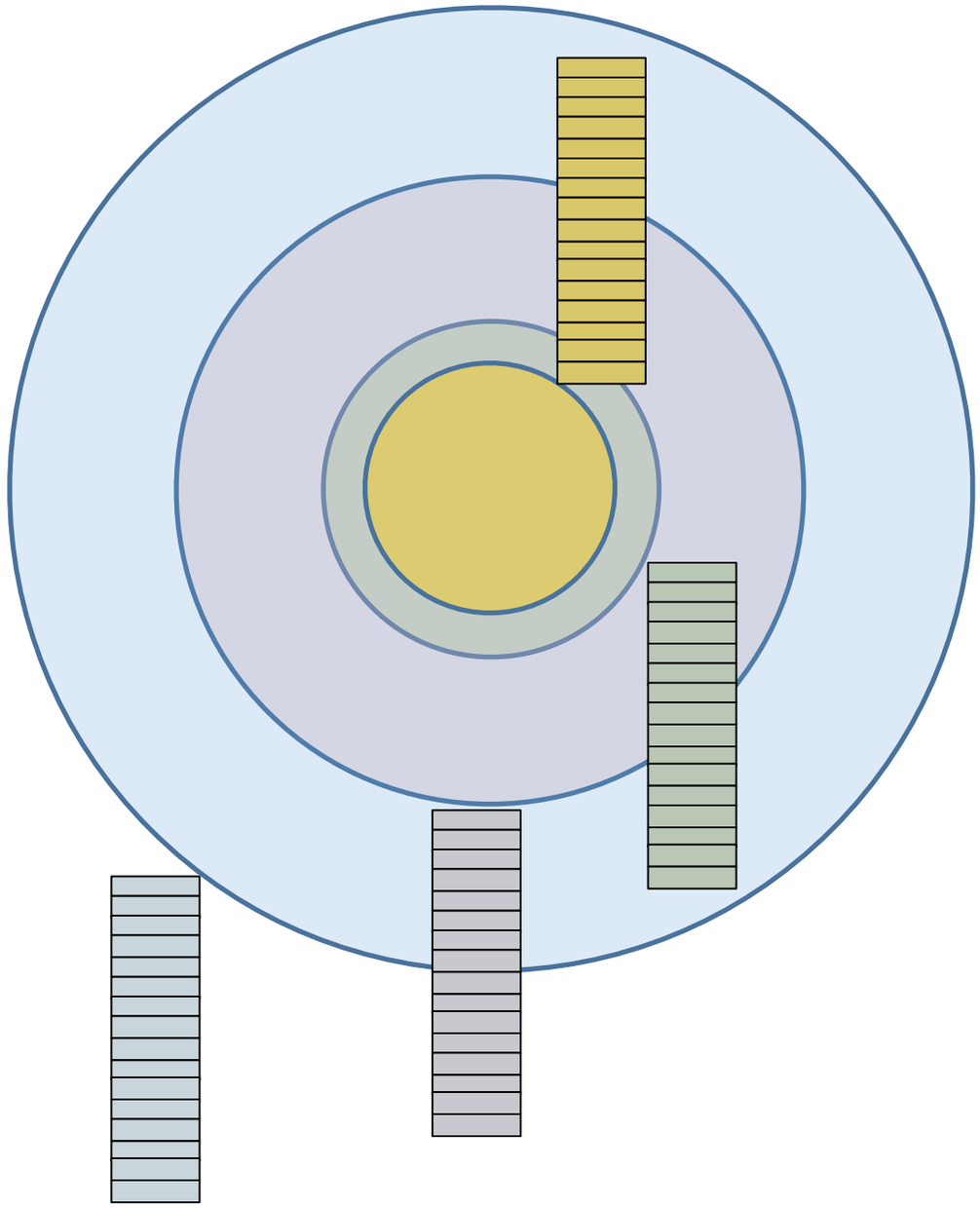


- Navigating very large data structures
- No locality
- OO abused: very deep instruction stack
- Cache misses

Introduced basketized transport

Deal with particles in parallel

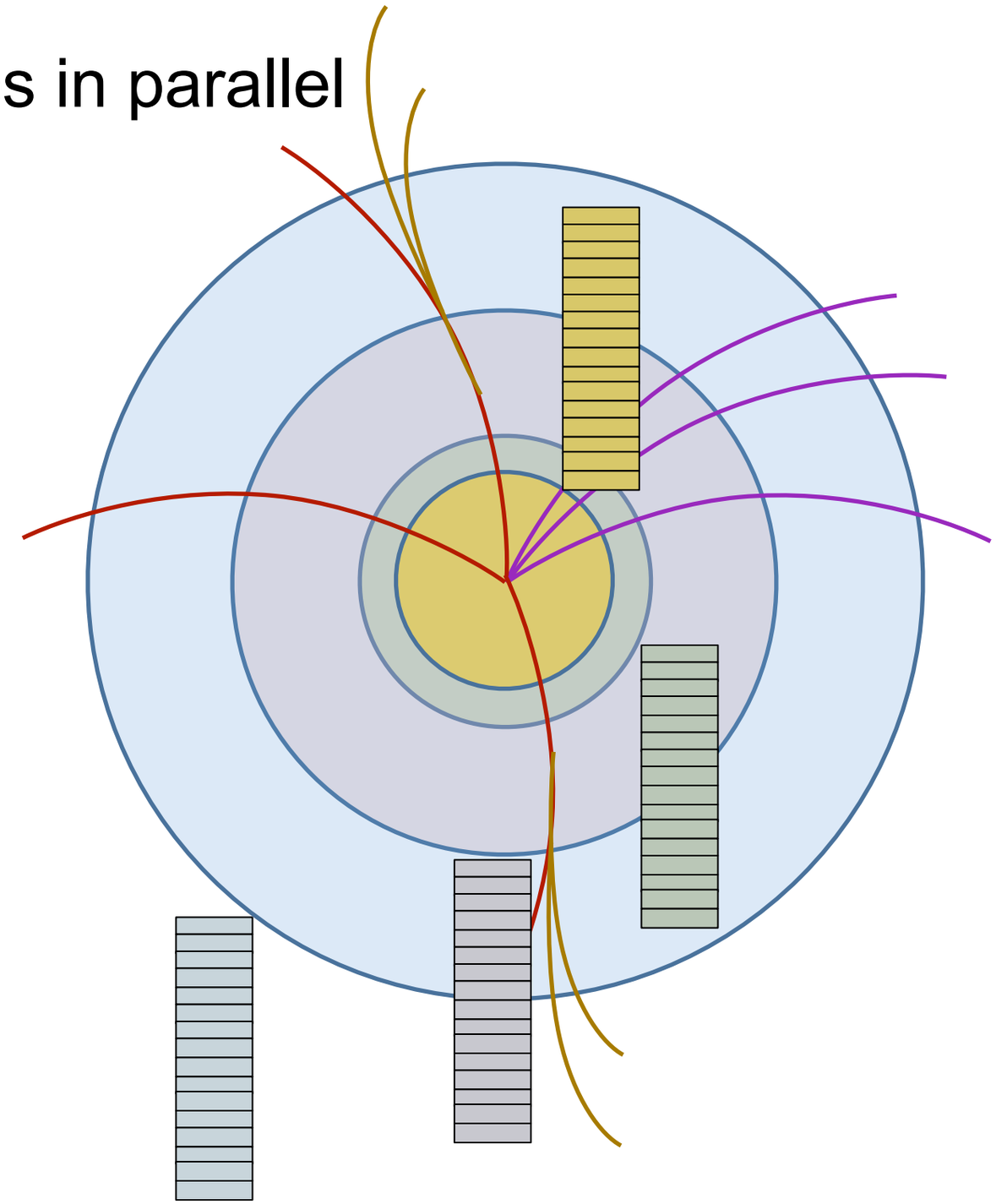
Output buffer(s)



Introduced basketized transport

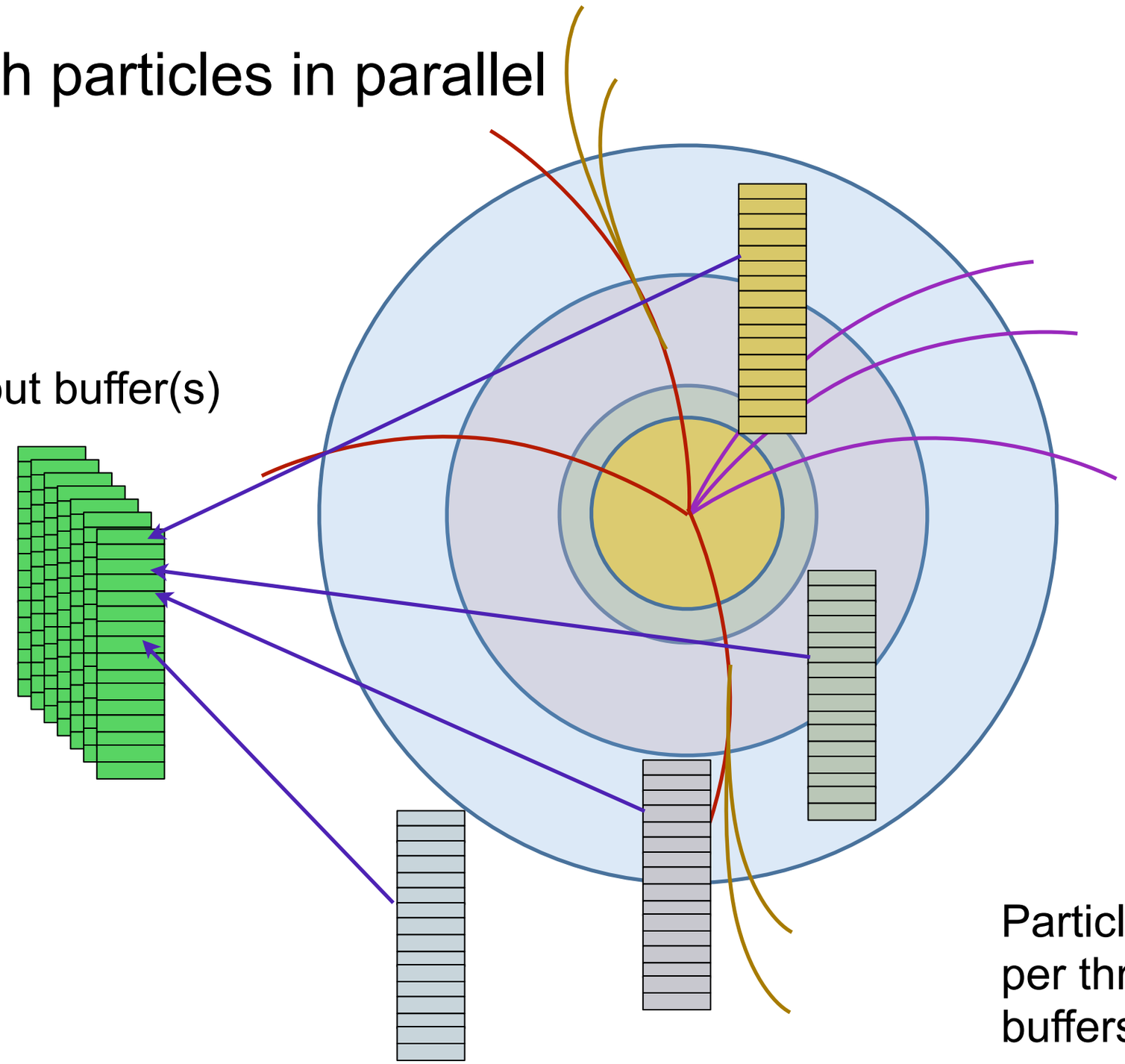
Deal with particles in parallel

Output buffer(s)



Introduced basketized transport

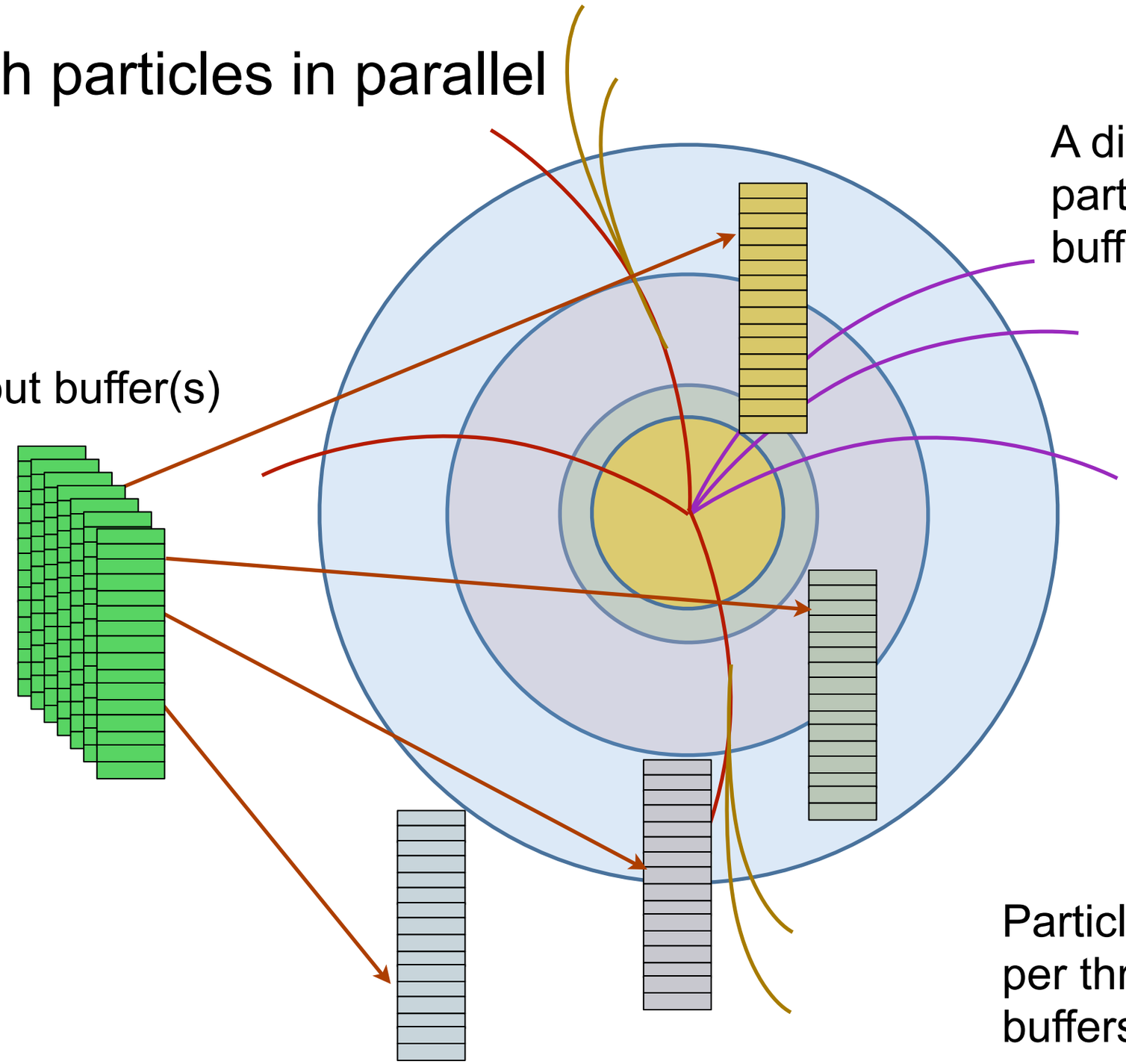
Deal with particles in parallel



Particles are transported per thread and put in output buffers

Introduced basketized transport

Deal with particles in parallel

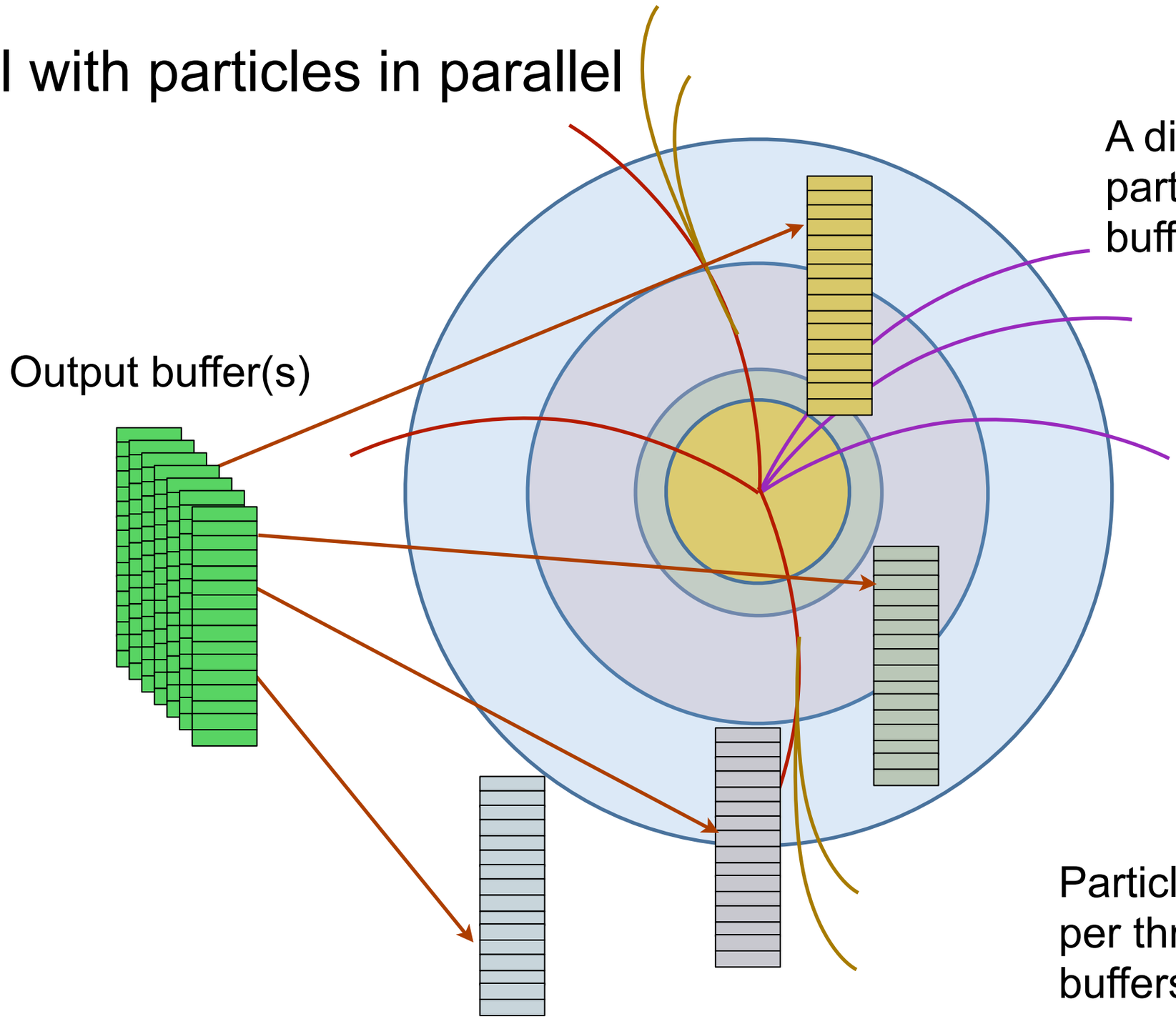


A dispatcher thread puts particles back into transport buffers

Particles are transported per thread and put in output buffers

Introduced basketized transport

Deal with particles in parallel



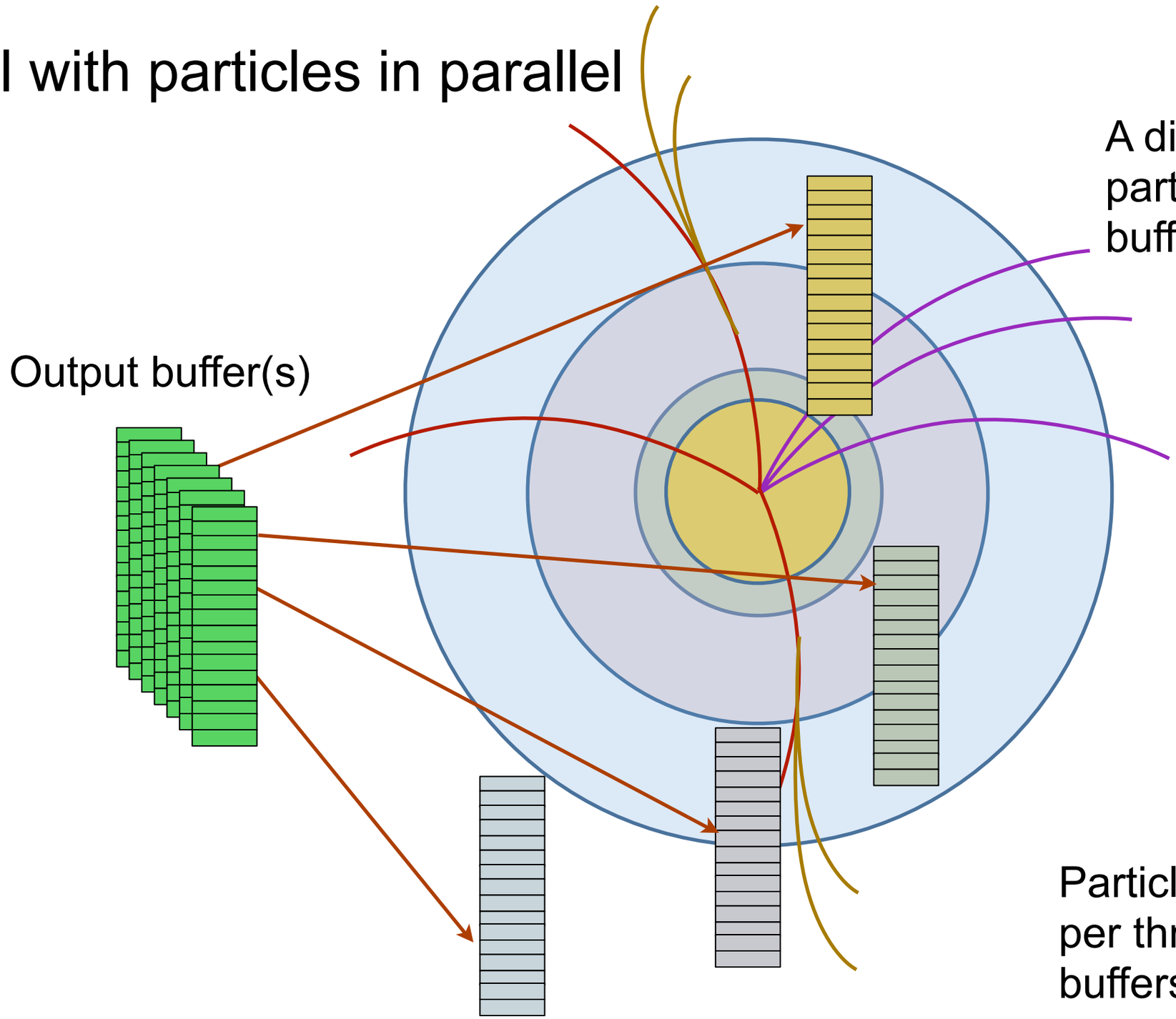
A dispatcher thread puts particles back into transport buffers

Everything happens asynchronously and in parallel

Particles are transported per thread and put in output buffers

Introduced basketized transport

Deal with particles in parallel



A dispatcher thread puts particles back into transport buffers

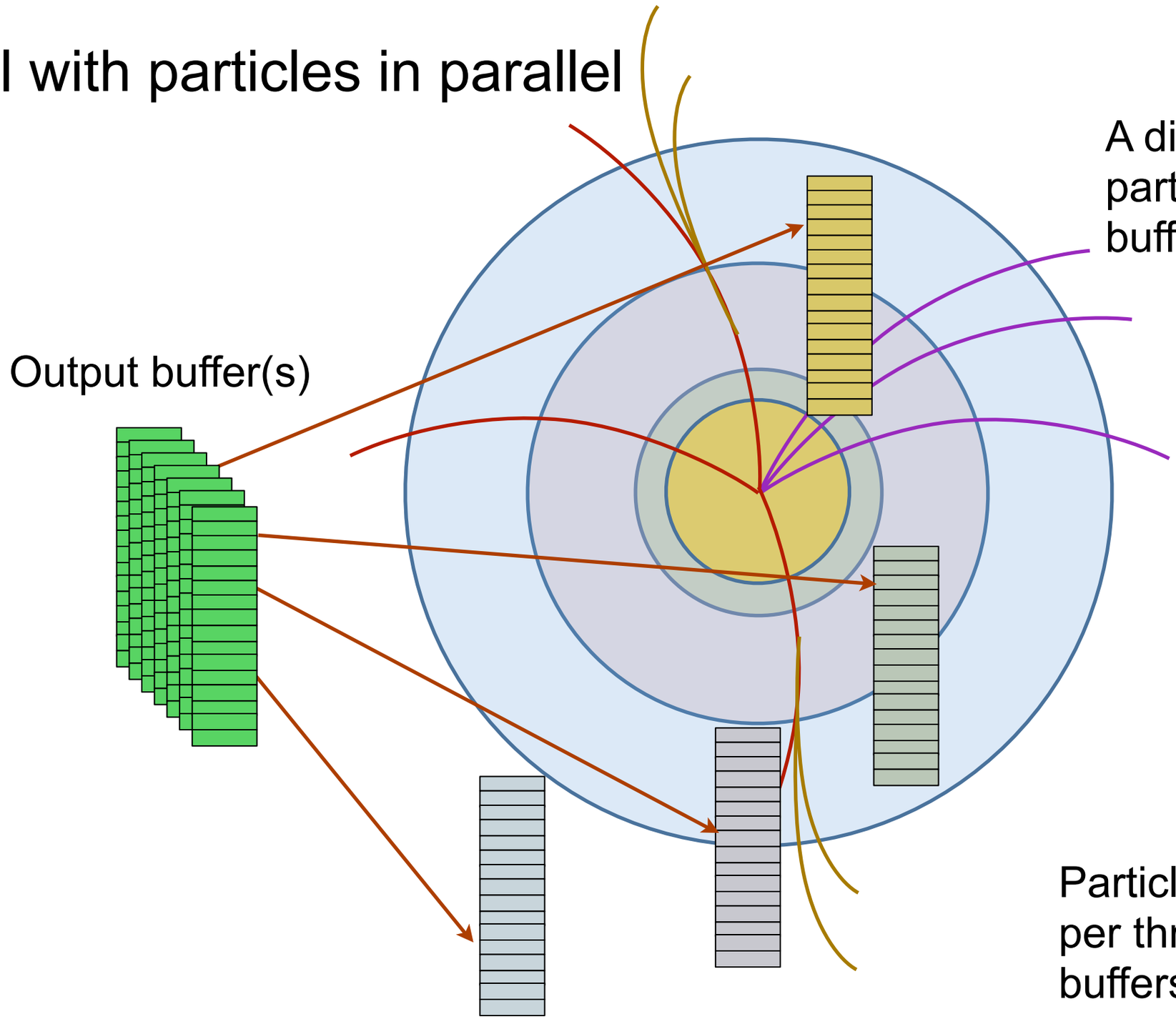
Everything happens asynchronously and in parallel

The challenge is to minimise locks

Particles are transported per thread and put in output buffers

Introduced basketized transport

Deal with particles in parallel



A dispatcher thread puts particles back into transport buffers

Everything happens asynchronously and in parallel

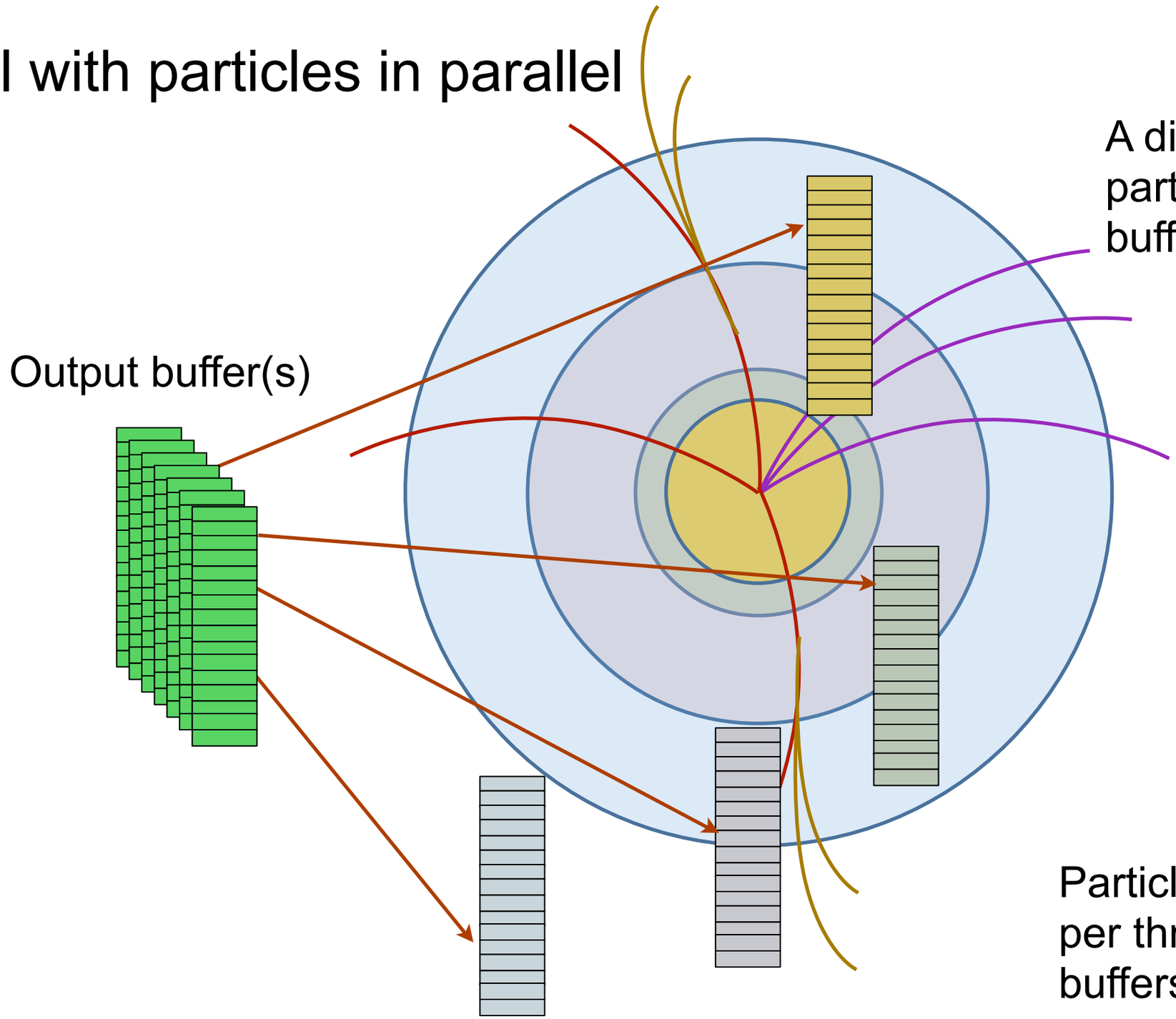
The challenge is to minimise locks

Keep long vectors

Particles are transported per thread and put in output buffers

Introduced basketized transport

Deal with particles in parallel



A dispatcher thread puts particles back into transport buffers

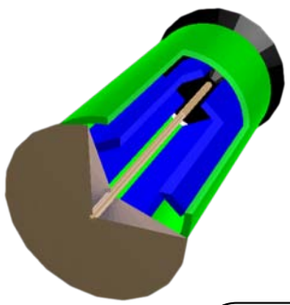
Everything happens asynchronously and in parallel

The challenge is to minimise locks

Keep long vectors

Avoid memory explosion

Particles are transported per thread and put in output buffers



Scheduler

Basket of tracks

Dispatching

Basket of tracks

MIMD

SIMD

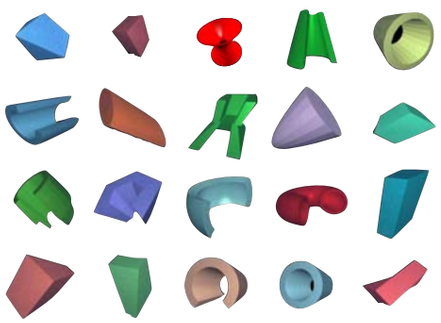
Geometry navigator

Physics

Geometry algorithms

x-sections

Reactions



Scheduler

Basket of tracks

Dispatching

Basket of tracks

MIMD

SIMD

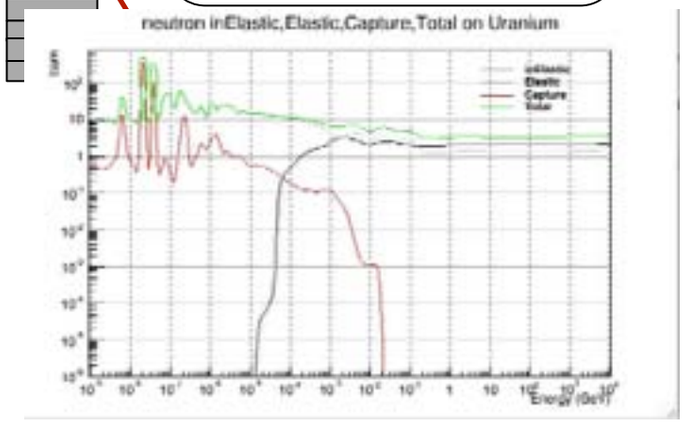
Geometry navigator

Physics

Geometry algorithms

x-sections

Reactions



Current design

List of logical Volumes

BF: basket status (one char per B)

Logical Volume lv

List of baskets for lv

Events

p array

Input particle list

p array

Output particle list

Active event list

History

List of active events for lv

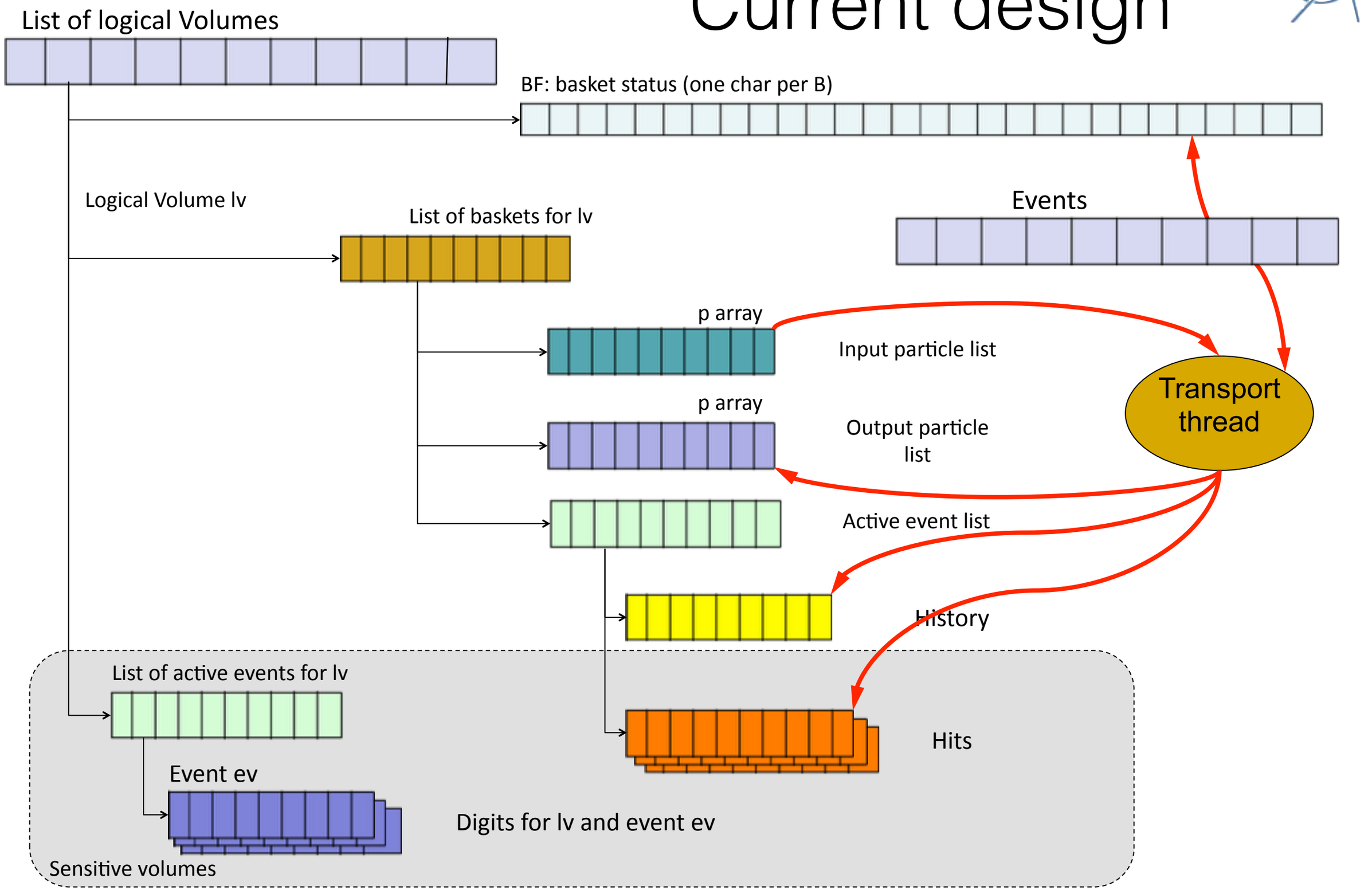
Hits

Event ev

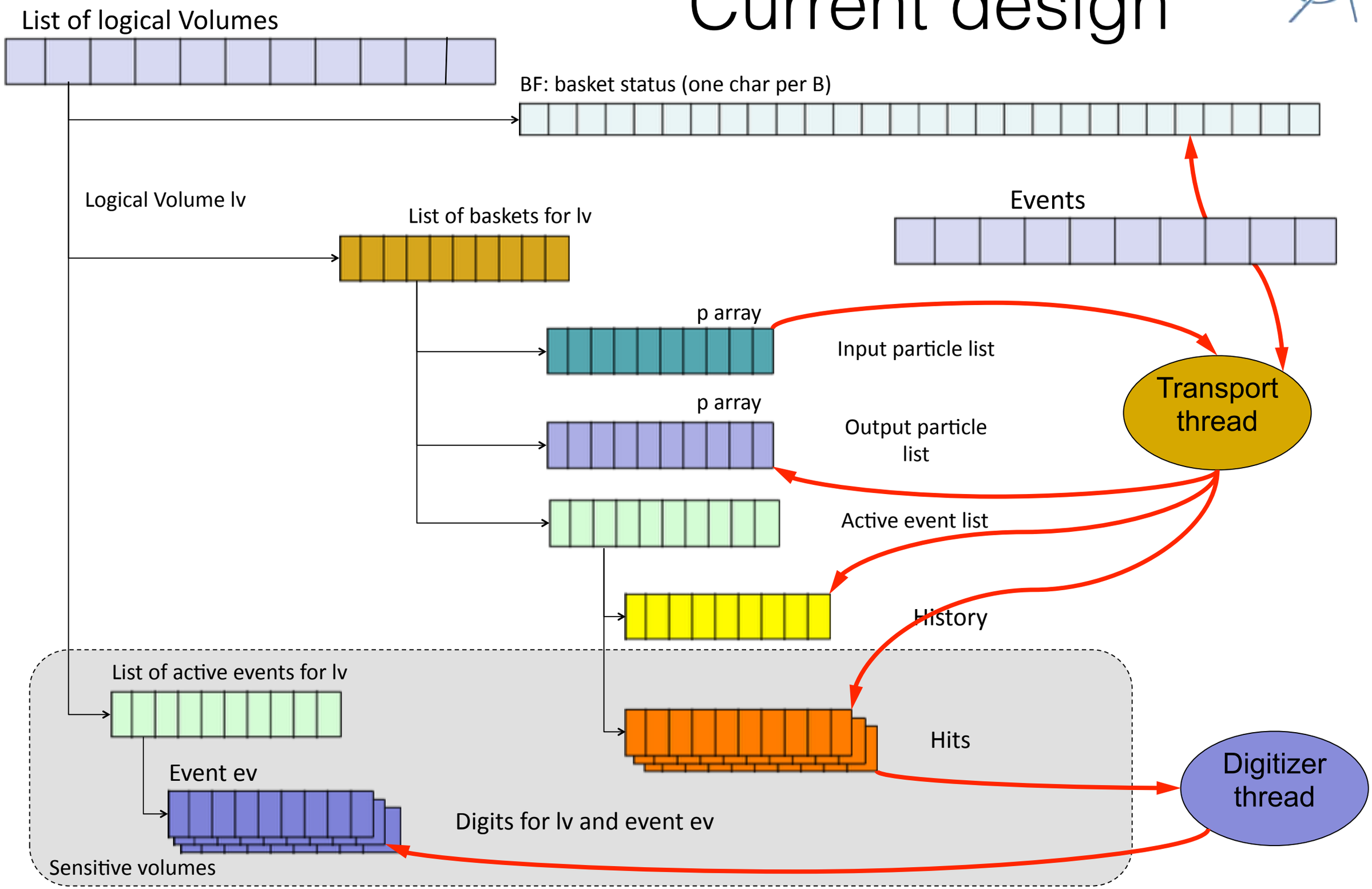
Digits for lv and event ev

Sensitive volumes

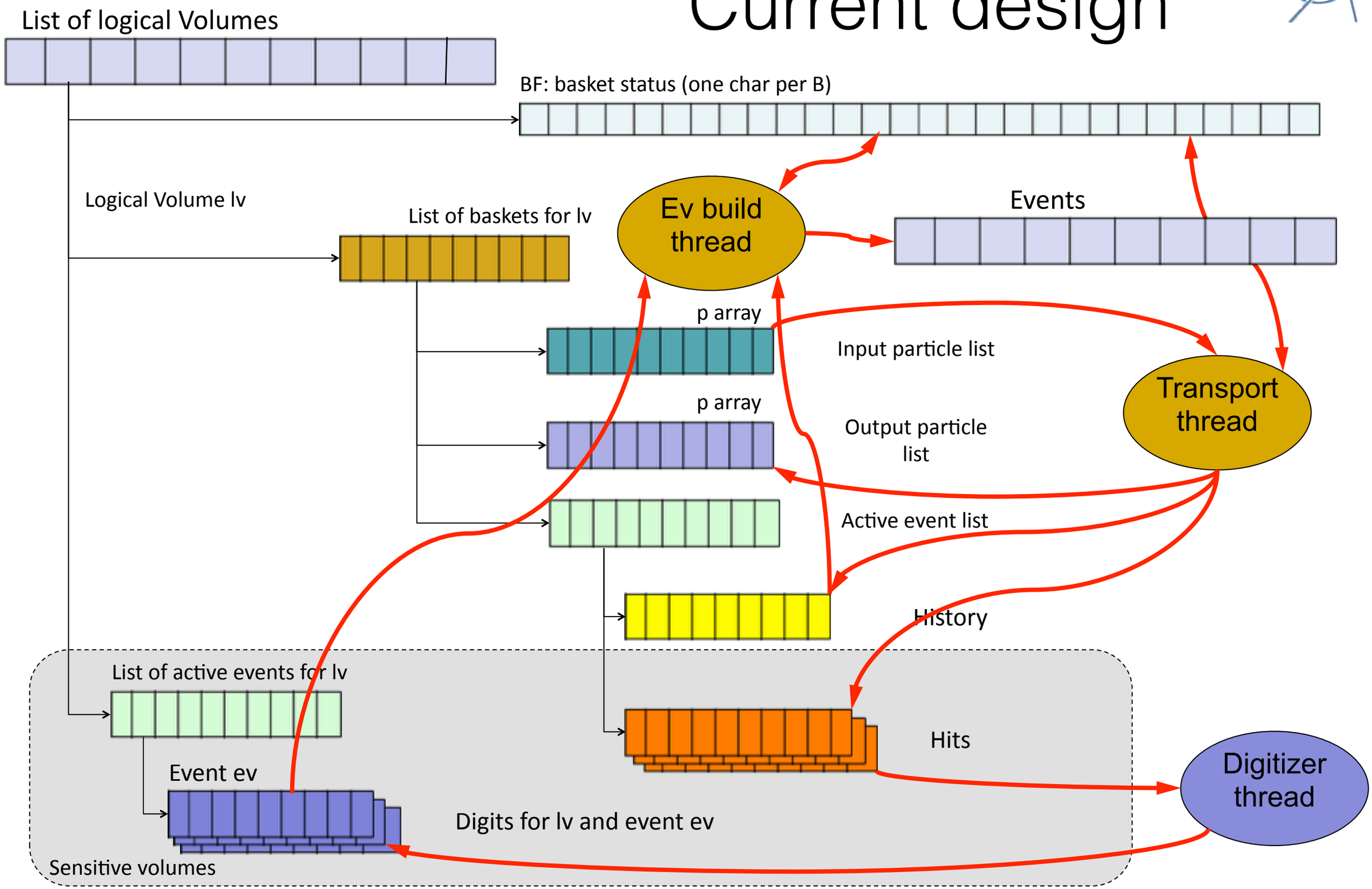
Current design



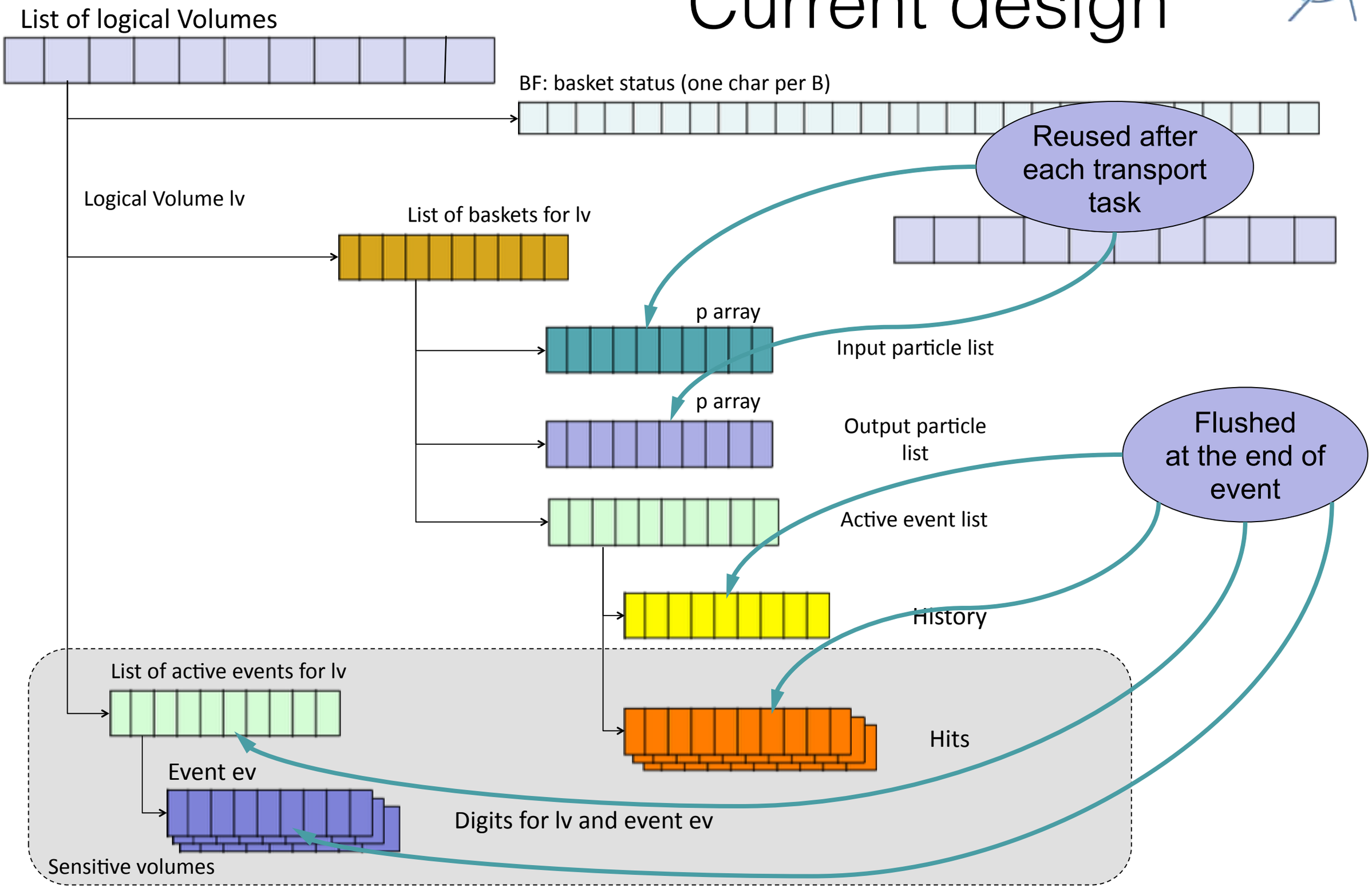
Current design



Current design

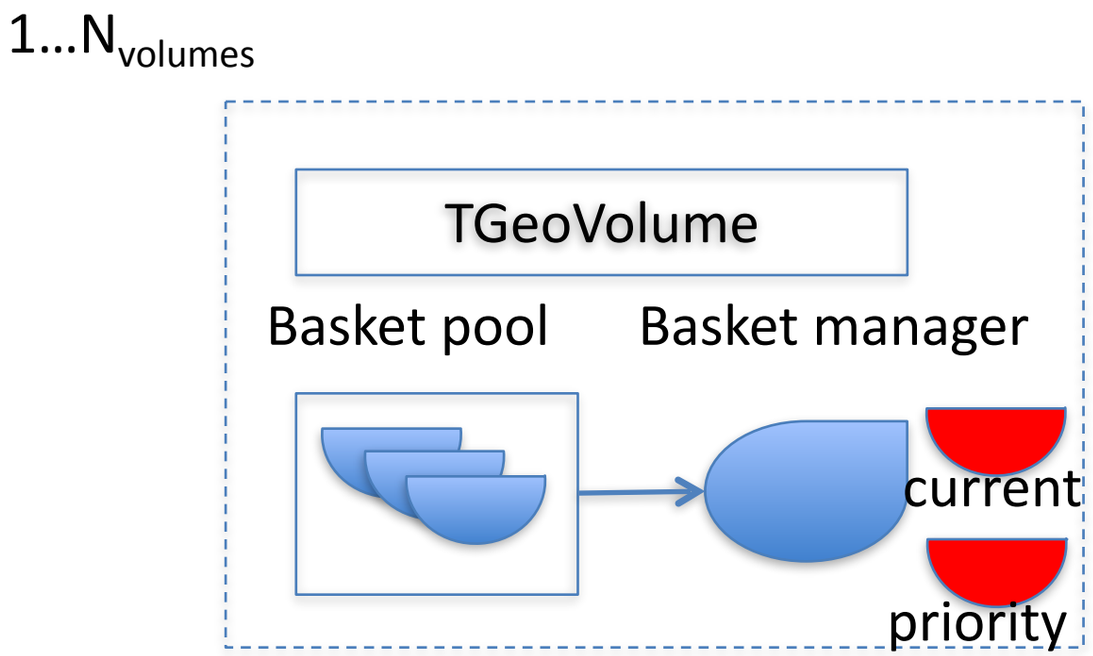


Current design

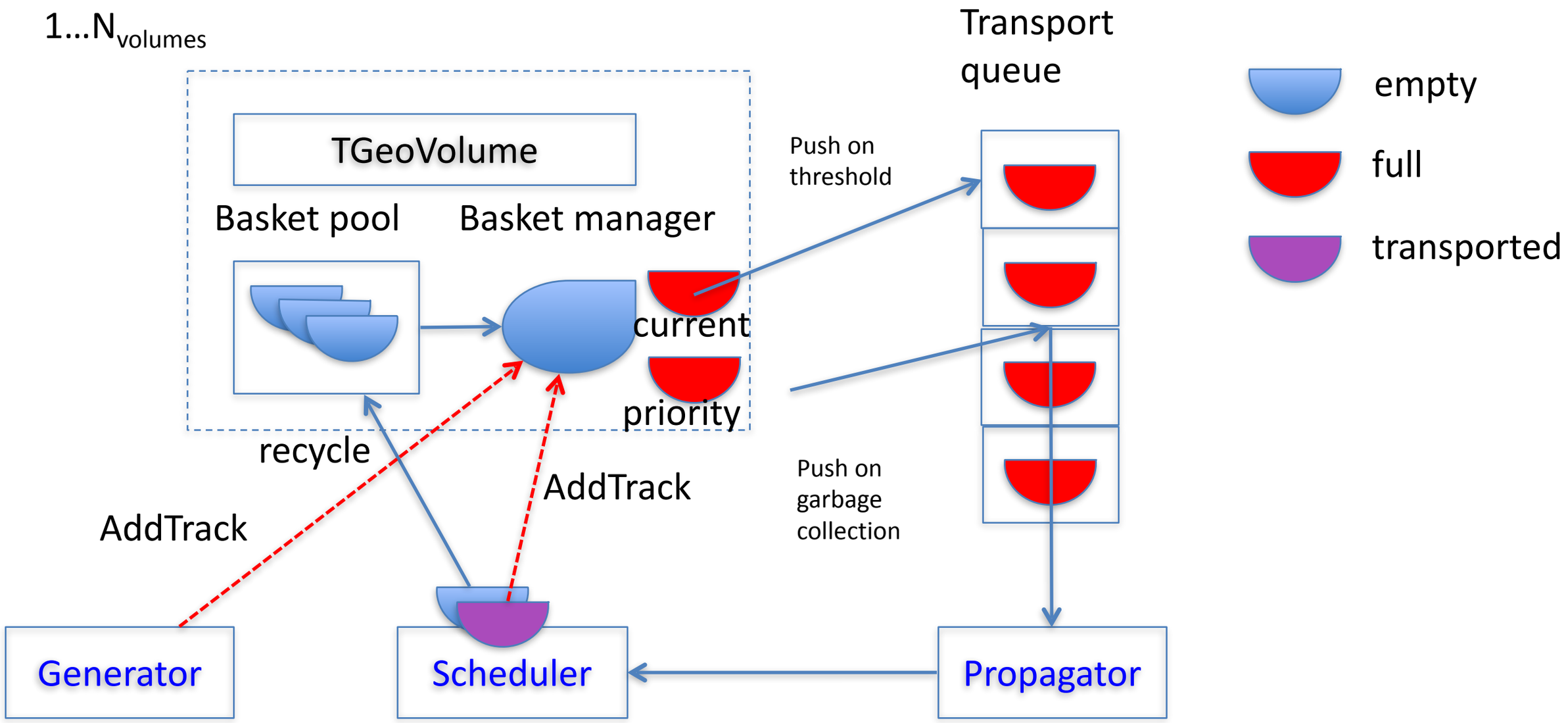


Basket managers

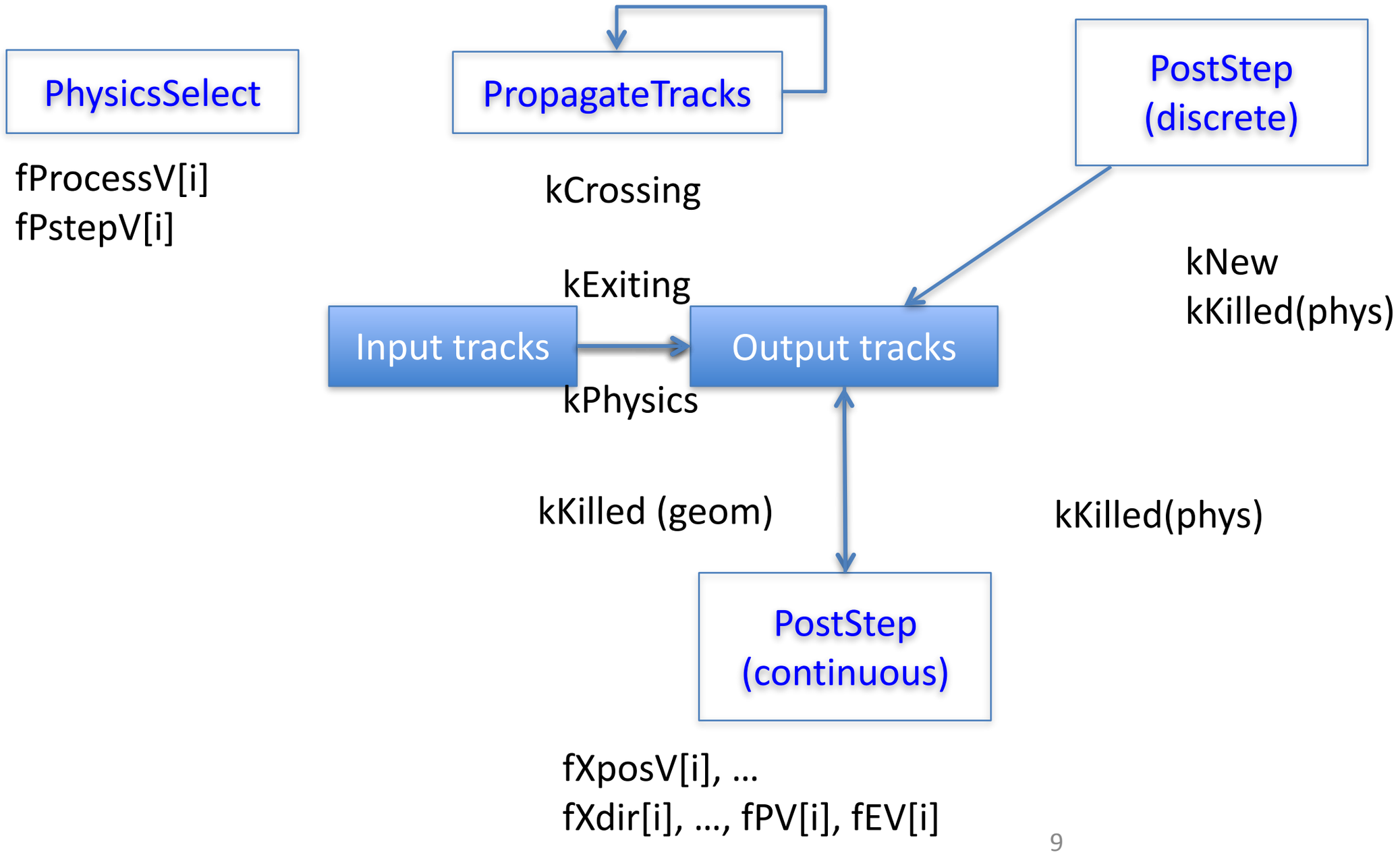
- One basket manager per volume
  - Receiving tracks entering the volume from generator or scheduler
  - Accessed by scheduler only
- Pool of empty baskets, one current basket + one basket for prioritized tracks
- Lock-free access for unique scheduler (only one thread can add tracks)
- Transportability threshold per manager
  - If threshold reached when adding tracks, the current basket is pushed in the work queue and replaced from the pool. Tracks added with the priority flag go to the priority basket which gets pushed to the priority side of the queue
  - Threshold(vol) = Ntracks_in_flight(vol)/2N_threads rounded to %4 (min 4, max 256)



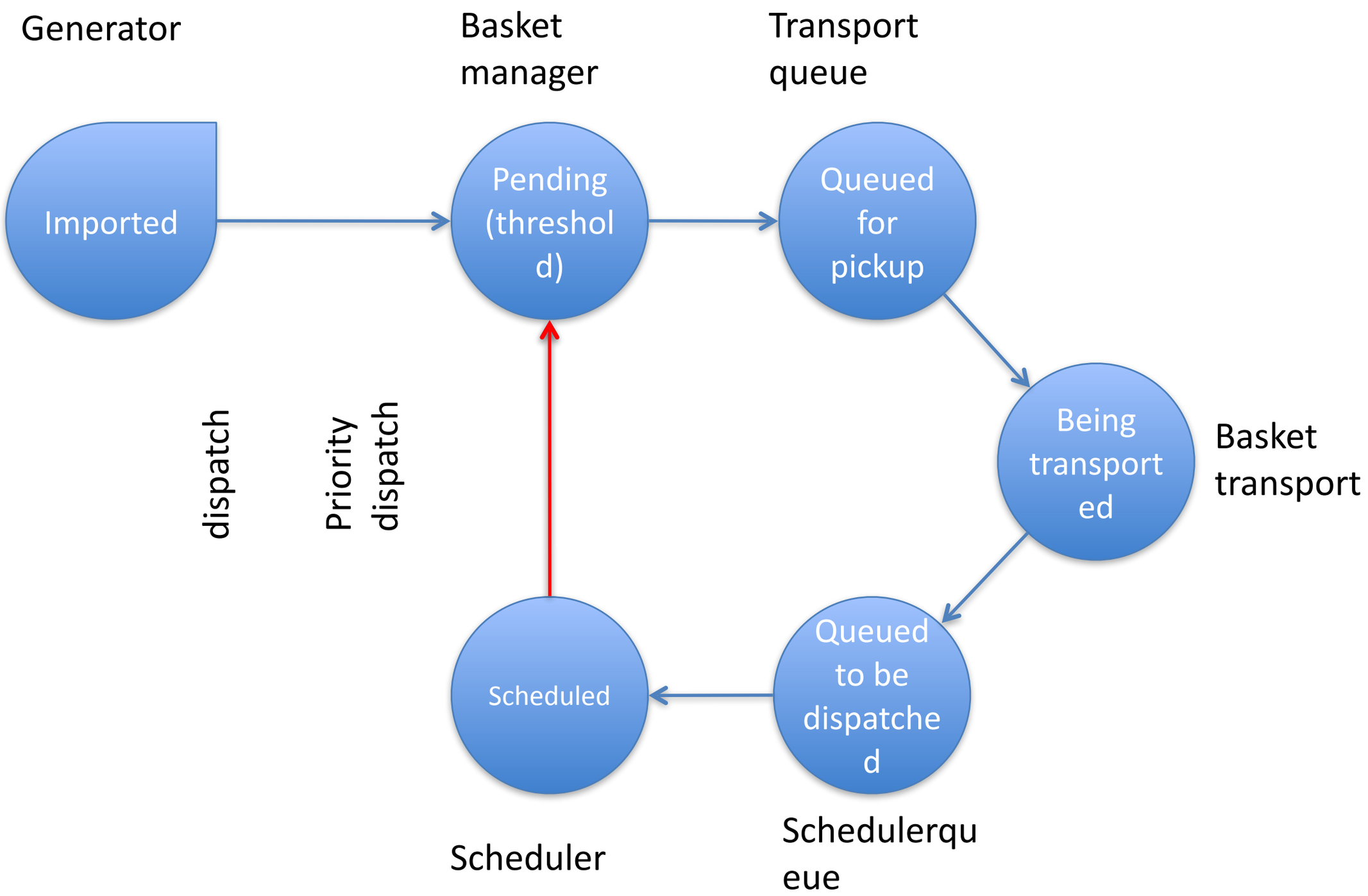
Basket lifecycle



Basket transport



Track stages

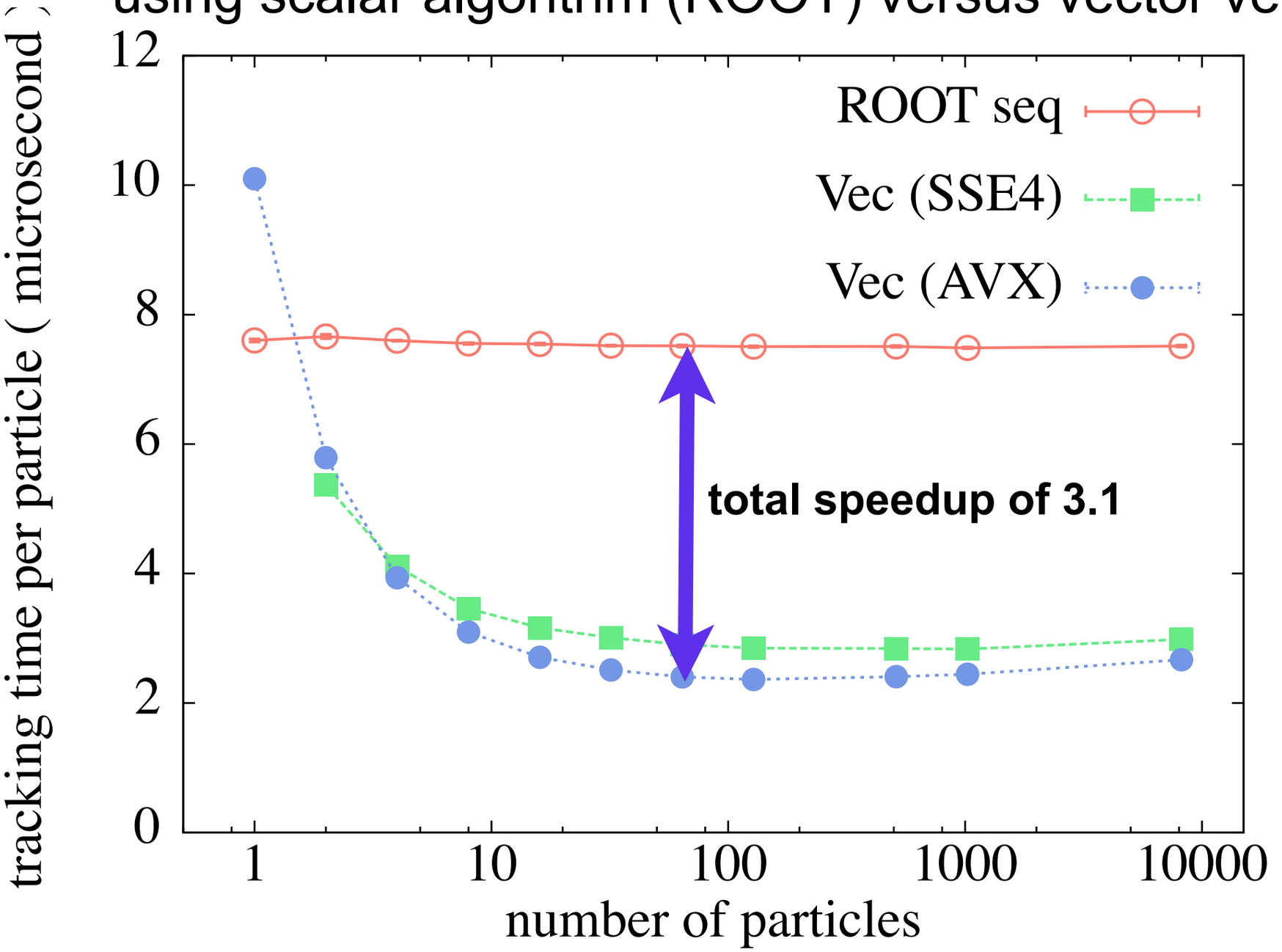


Scheduling policies

- Workload balancing
  - Divide the work evenly to scale with number of workers
  - Queue control: garbage collection on work queue depletion
  - Improvement: schedule physics as separate task (process selection and discrete processes post-step)
- Memory management
  - Not active currently, the idea it to trigger hit/digits collection and memory cleanup on thresholds
- Keep large vectors
  - Raise transportability thresholds per volume
  - Postpone sparse tracks when not in garbage collection mode
- Trigger single track mode when vectorization gives just overhead

Gains from micro parallelism and SIMD (old!)

- Time of processing/navigating N particles (P repetitions) using scalar algorithm (ROOT) versus vector version

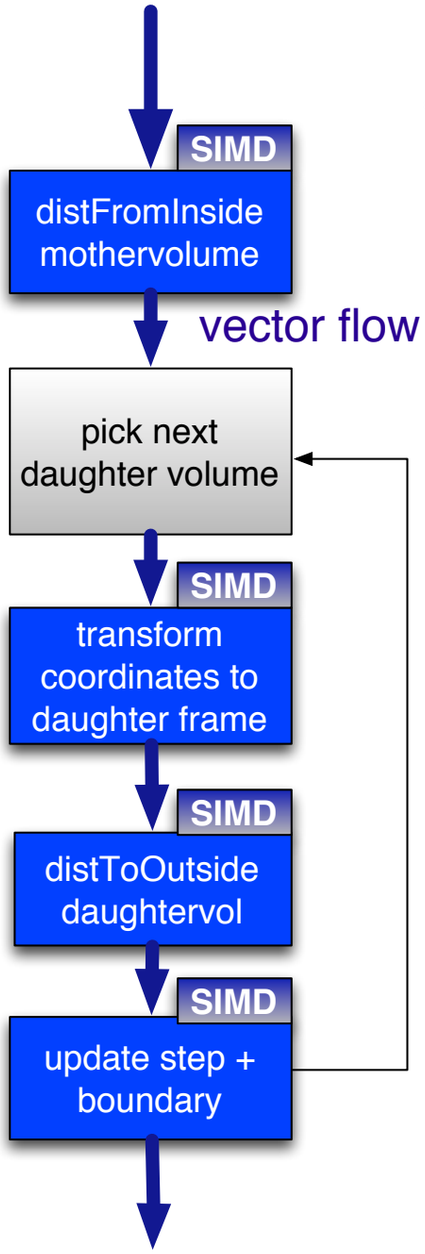


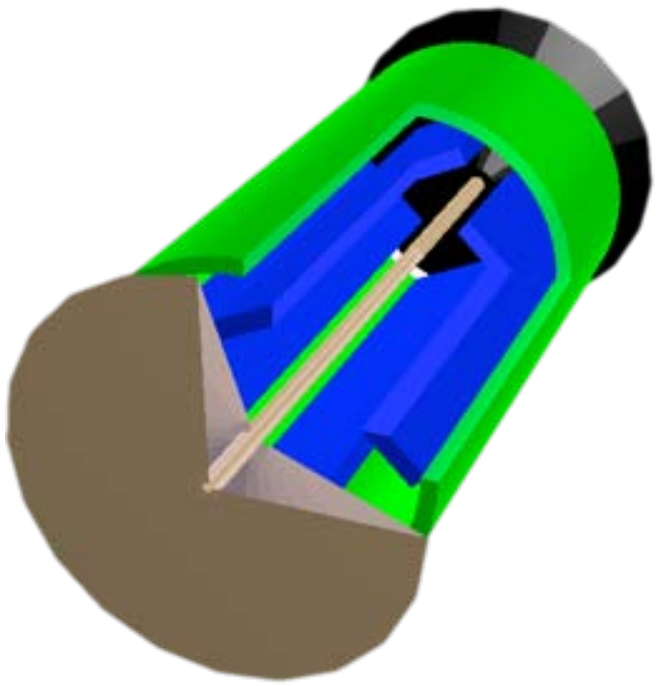
- excellent speedup for SSE4 version
- some further gain with AVX
- already gain considerably for small N
- there is an optimal point of operation (performance degradation for large N)

Recap of performance status

- provided new optimized vector interfaces for some elementary solids and geometric base classes (implemented important functions for particle navigation)

- overall performance gain in a standard navigation benchmark (in toy detector with 4 boxes, 3 tubes, 2 cones) - comparison to ROOT/ 5.34.17





	16 particles	1024 particles
Intel IvyBridge (AVX)	~2.8x	~4.0x
Intel Haswell (AVX2)	~3.0x	~5.0x
Intel Xeon-Phi (AVX512)	~4.1x	~4.8x

Xeon-Phi and Haswell benchmarks by CERN Openlab (Georgios Bitzes)

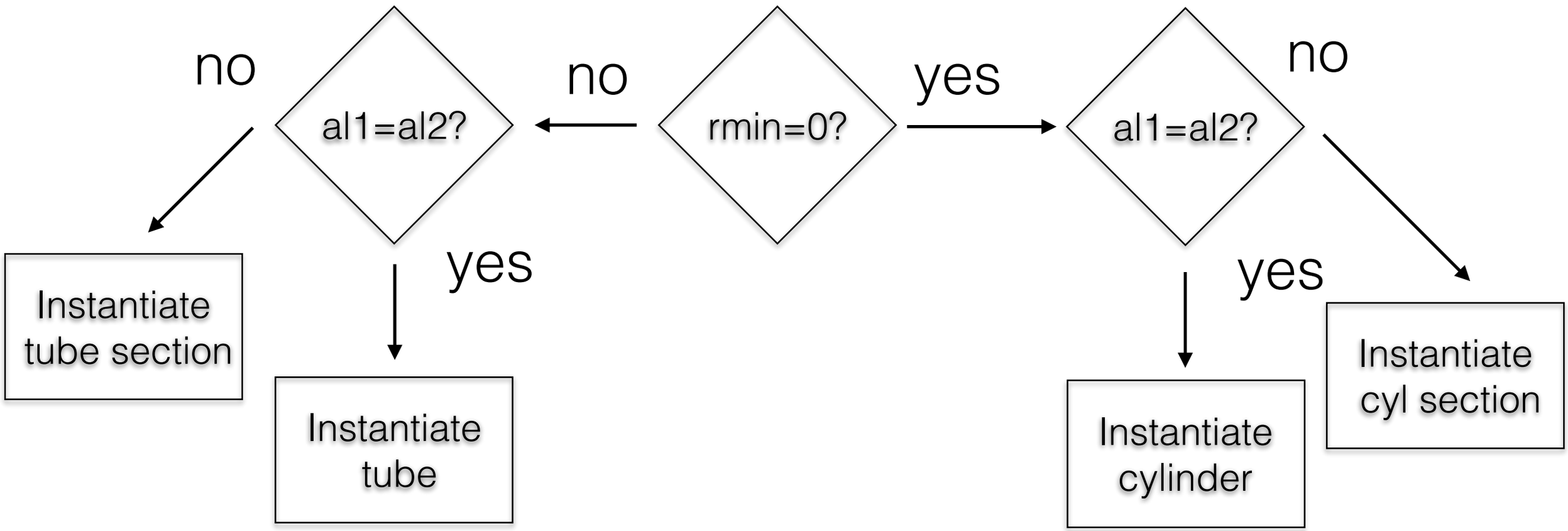
CHEP13 paper: http://arxiv.org/pdf/1312.0816.pdf

Portable HPC?

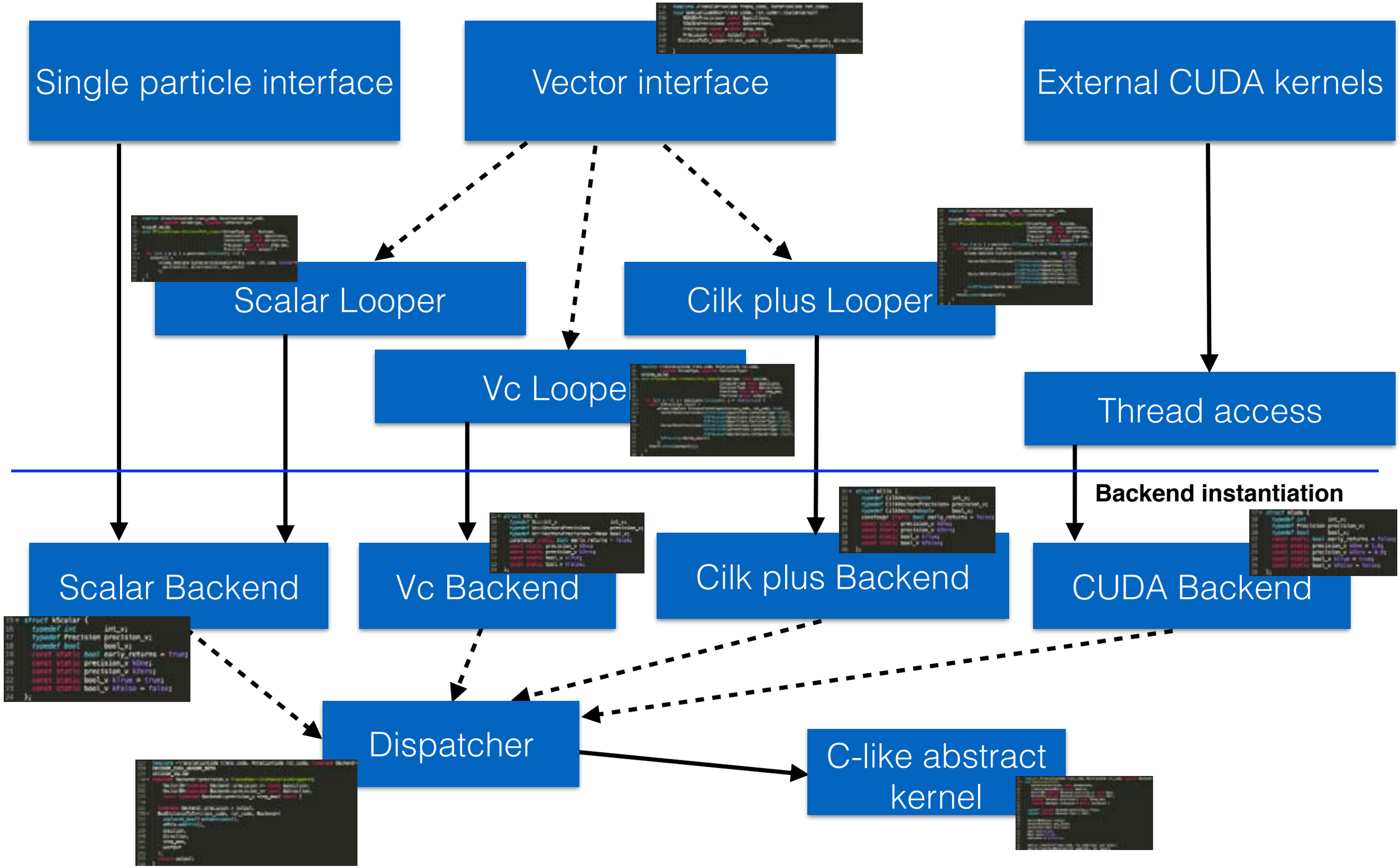
- Straight “vectorisation” of existing code is difficult to impossible
- Resulting code is hard to read and maintain
  - And it is largely compiler-dependency
- Porting to different high end devices is very difficult
- Explored solution is to use template specification for solid placement, specialisation and code generation
- Highly optimised modular “codelets” à la STL are used to construct algorithms

Solid specialisation

CreateTube(rmin, rmax, al1, al2)



Illustrating scalar/SIMD abstraction and kernels



```
134  template <TranslationCode trans_code, RotationCode rot_code>
135  void SpecializedBox<trans_code, rot_code>::DistanceToIn(
136      SOA3D<Precision> const &positions,
137      SOA3D<Precision> const &directions,
138      Precision const *const step_max,
139      Precision *const output) const {
140    DistanceToIn_Looper<trans_code, rot_code>(*this, positions, directions,
141                                              step_max, output);
142  }
```

Illu

Single

Scalar Looper

Cilk plus Looper

Vc Loope

Thread access

Backend instantiation

Scalar Backend

Vc Backend

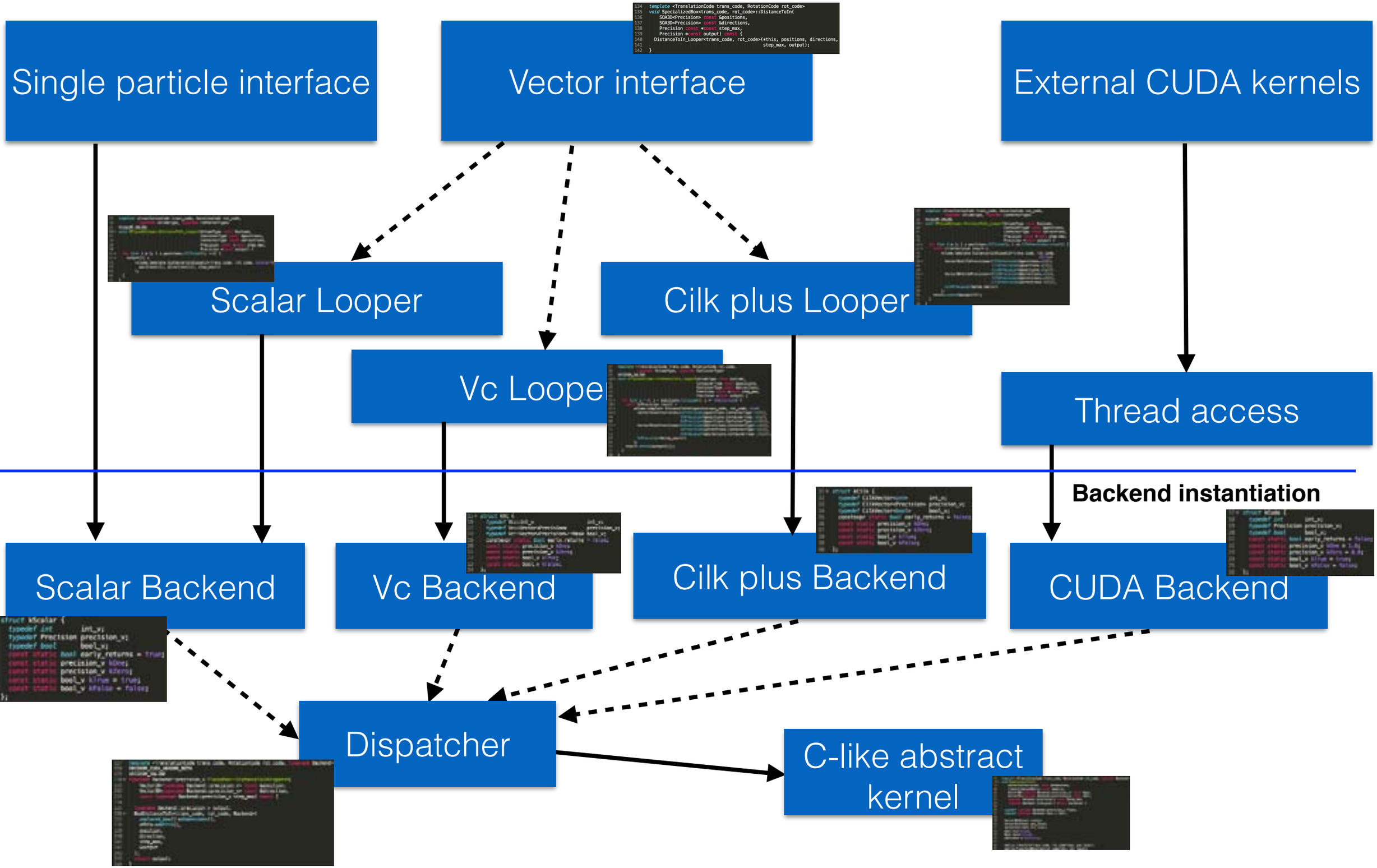
Cilk plus Backend

CUDA Backend

Dispatcher

C-like abstract kernel

Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

```
template <TranslationCode trans_code, RotationCode rot_code,
          typename VolumeType, typename ContainerType>
VECGEOM_INLINE
void VPlacedVolume::DistanceToIn_Looper(VolumeType const &volume,
                                        ContainerType const &positions,
                                        ContainerType const &directions,
                                        Precision const *const step_max,
                                        Precision *const output) {
  for (int i = 0; i < positions.fillsize(); ++
    output[i] =
        volume.template DistanceToInDispatch<
          positions[i], directions[i], step_max[i]
        );
  }
}
```

Thread access

Backend instantiation

Scalar Backend

Vc Backend

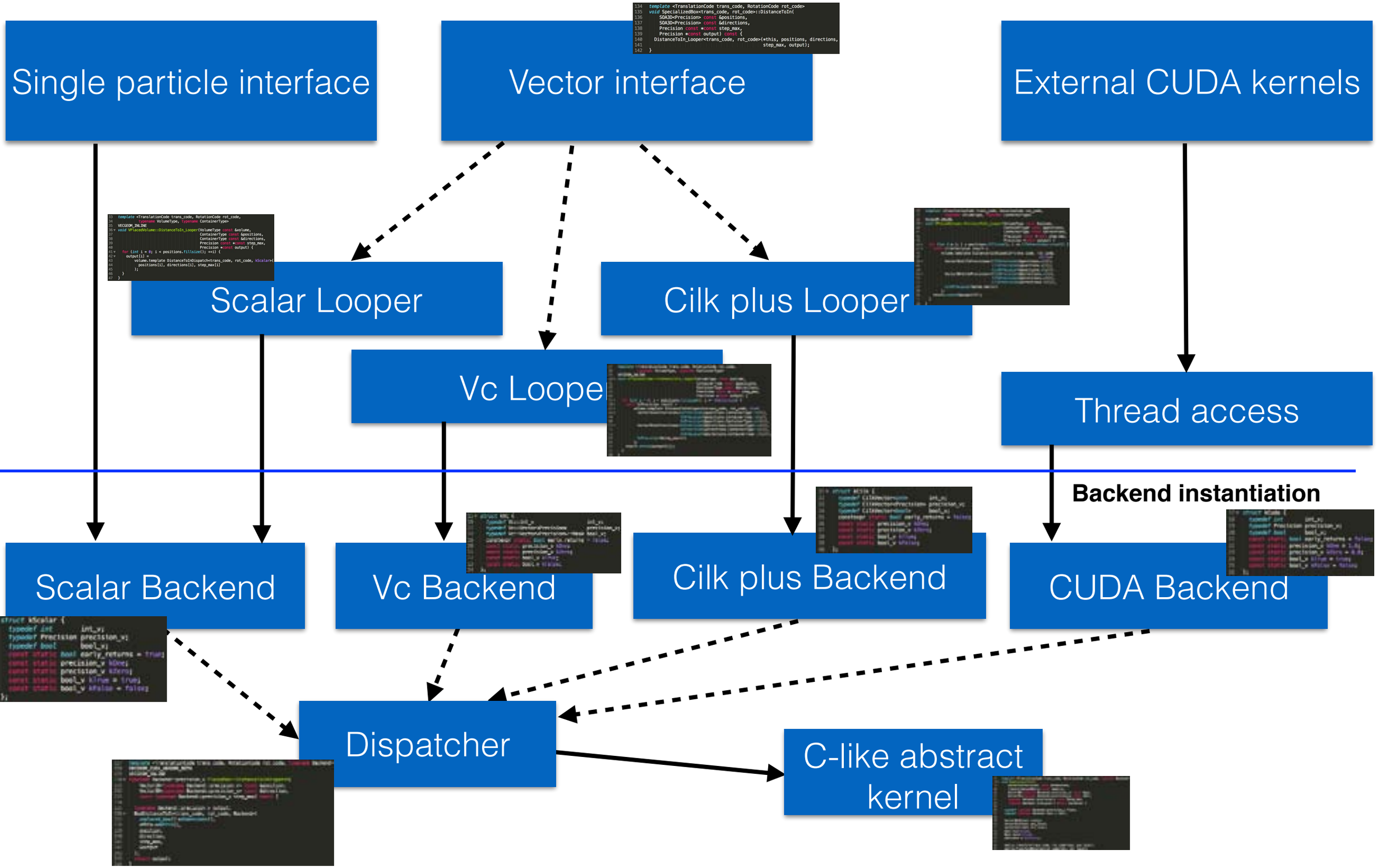
Cilk plus Backend

CUDA Backend

Dispatcher

C-like abstract kernel

Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

```
template <TranslationCode trans_code, RotationCode rot_code,
          typename VolumeType, typename ContainerType>
VECGEOM_INLINE
void VPlacedVolume::DistanceToIn_Looper(VolumeType const &volume,
                                        ContainerType const &positions,
                                        ContainerType const &directions,
                                        Precision const *const step_max,
                                        Precision *const output) {
  for (int i = 0; i < positions.fillsize(); i += kVectorSize) {
    const VcPrecision result =
        volume.template DistanceToInDispatch<trans_code, rot_code, kVc>(
          Vector3D<VcPrecision>(VcPrecision(&positions.ContainerType::x(i)),
                                VcPrecision(&positions.ContainerType::y(i)),
                                VcPrecision(&positions.ContainerType::z(i))),
          Vector3D<VcPrecision>(VcPrecision(&directions.ContainerType::x(i)),
                                VcPrecision(&directions.ContainerType::y(i)),
                                VcPrecision(&directions.ContainerType::z(i))),
          VcPrecision(&step_max[i])
        );
    result.store(&output[i]);
  }
}
```

access

instantiation

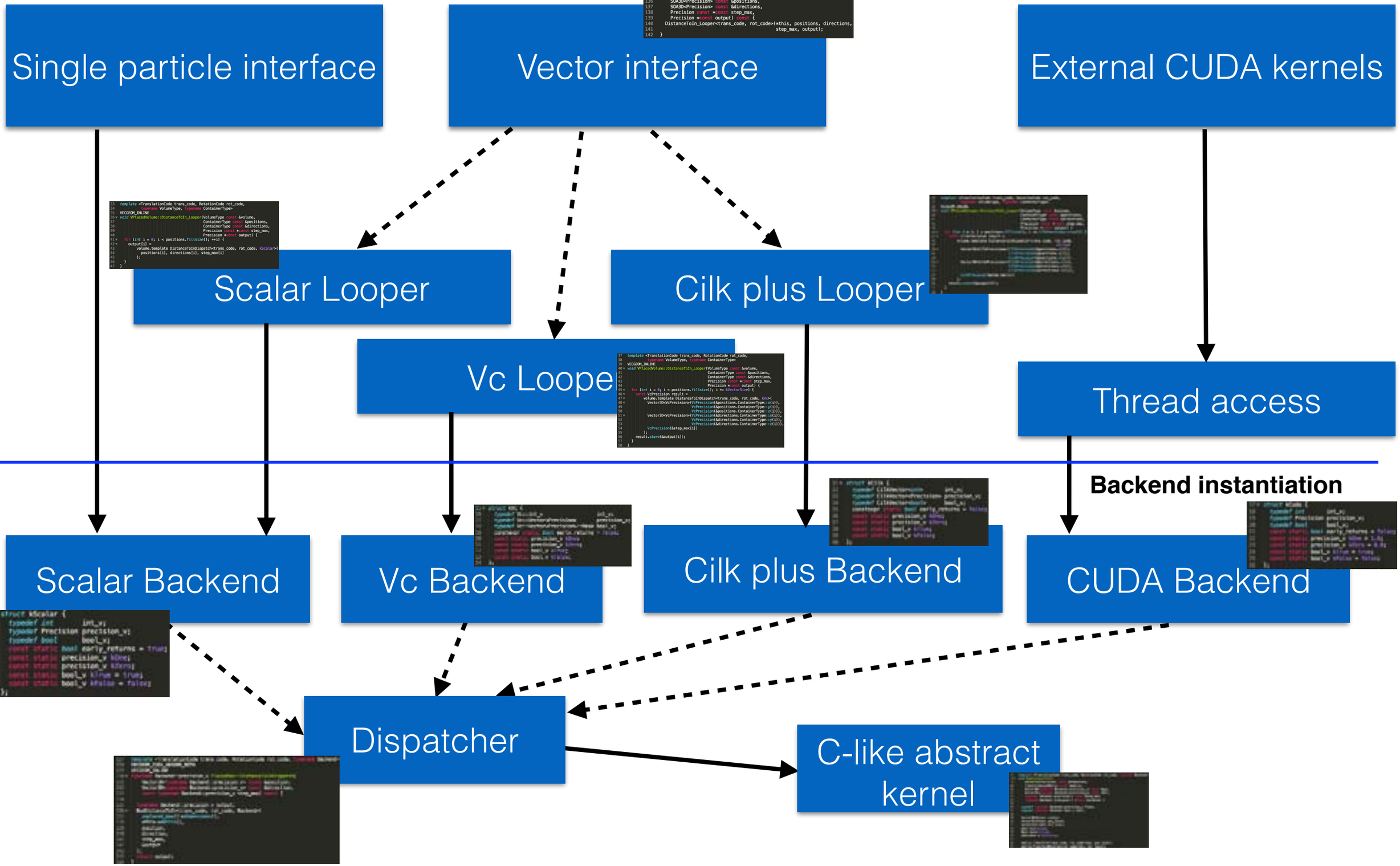
Scalar Bac

ackend

Dispatcher

C-like abstract kernel

Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels

Single particle interfa

kernels

Scalar L

ess

iation

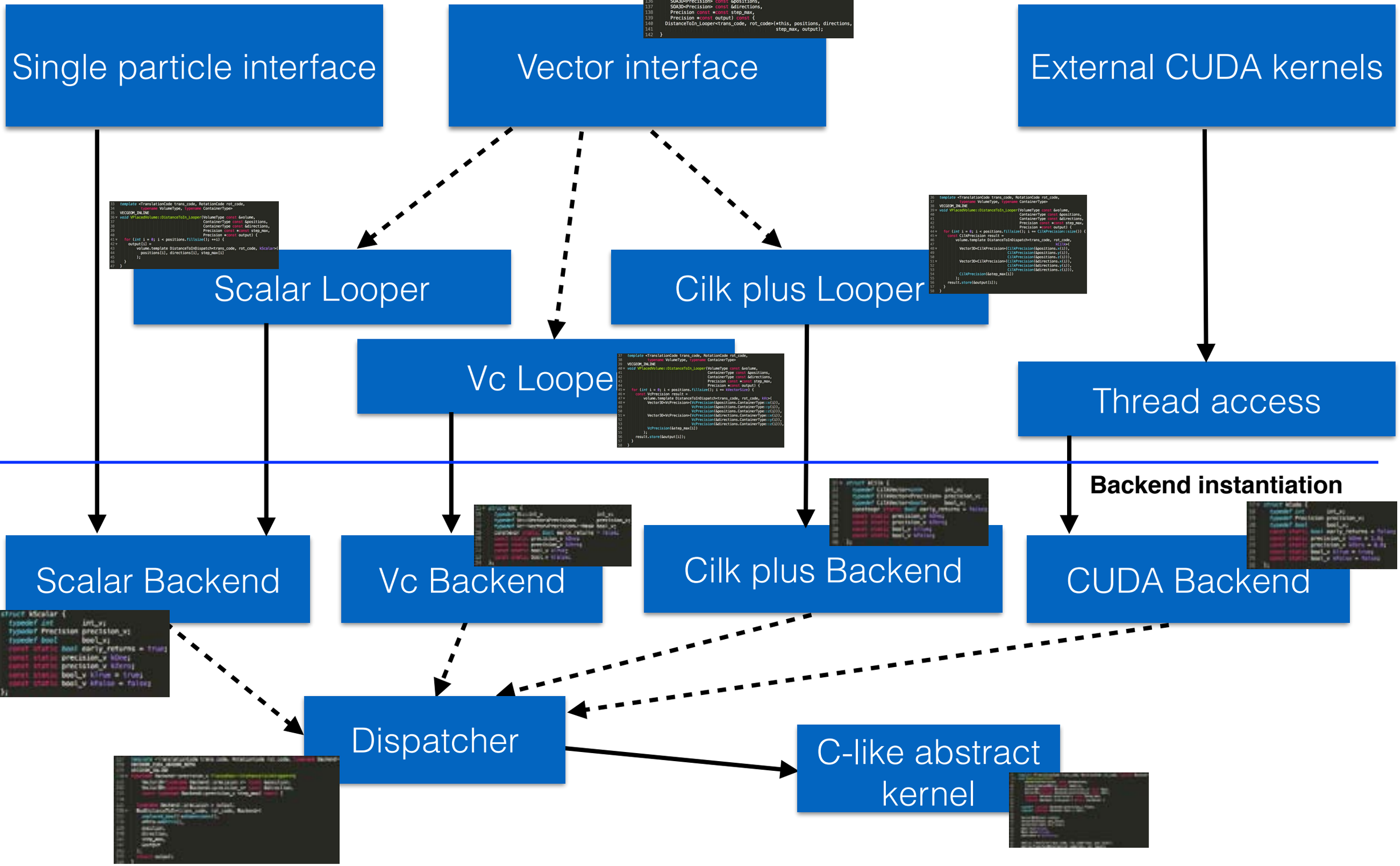
Scalar Backend

nd

Dispatcher

C-like abstract kernel

Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

Scalar Looper

Cilk plus Looper

Thread access

Backend instantiation

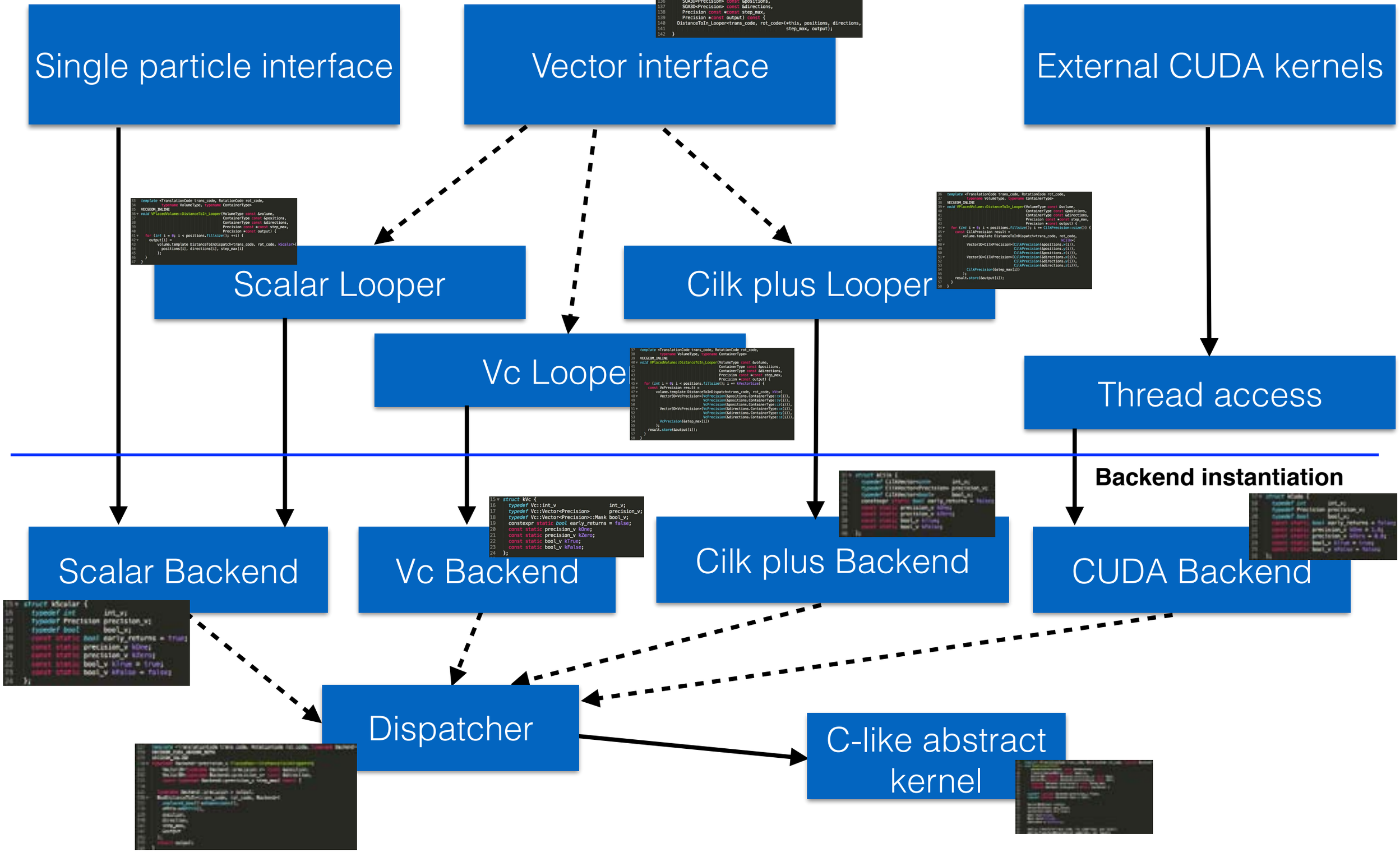
CUDA Backend

Dispatcher

C-like abstract kernel

```
15 ▼ struct kVc {
16       typedef Vc::int_v                          int_v;
17       typedef Vc::Vector<Precision>              precision_v;
18       typedef Vc::Vector<Precision>::Mask bool_v;
19       constexpr static bool early_returns = false;
20       const static precision_v kOne;
21       const static precision_v kZero;
22       const static bool_v kTrue;
23       const static bool_v kFalse;
24    };
```

Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

Scalar I

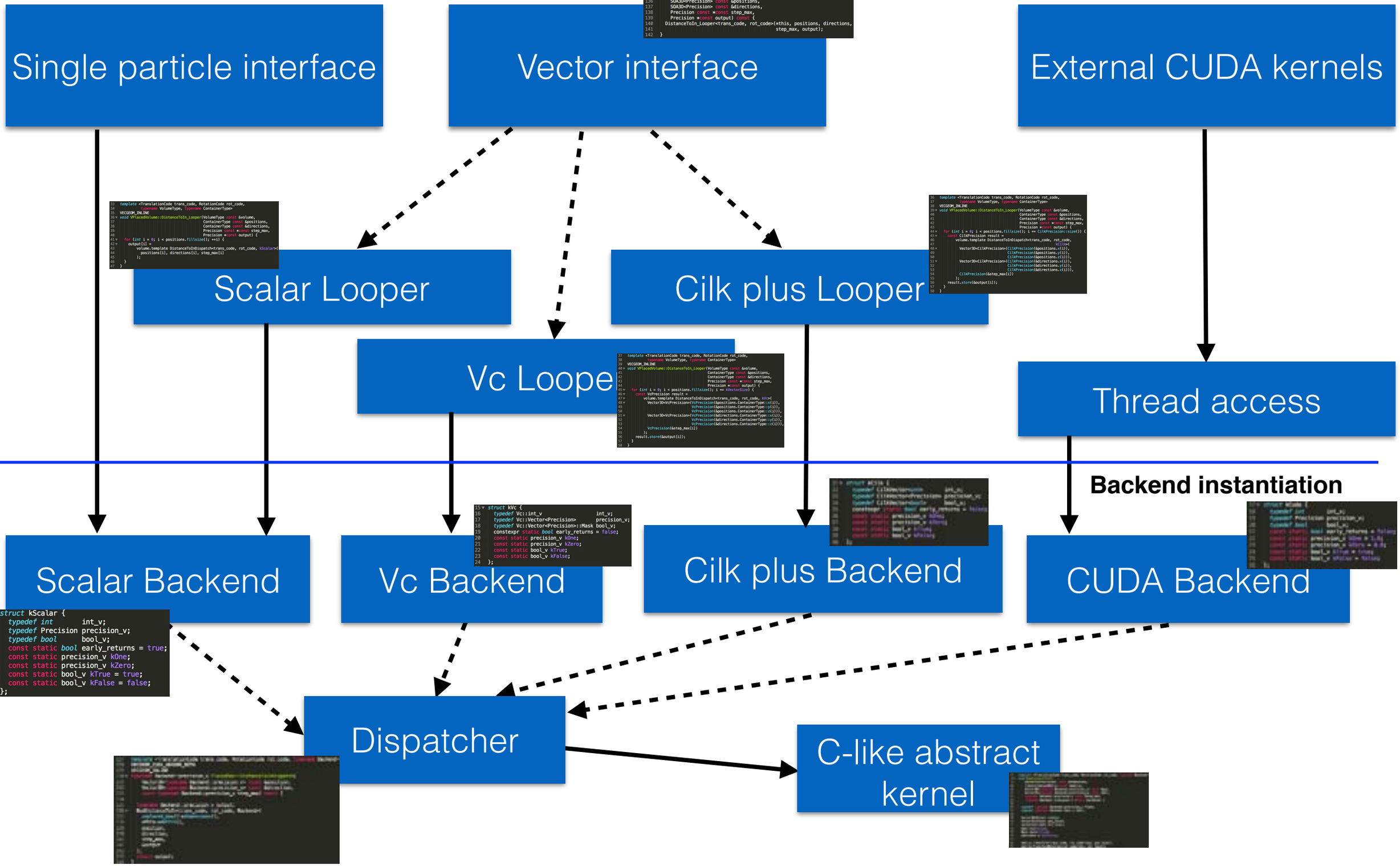
ation

nd

Dispatcher

C-like abstract kernel

Illustrating scalar/SIMD abstraction and kernels

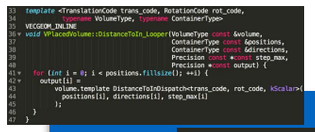


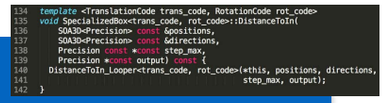
Illustrating scalar/SIMD abstraction and kernels

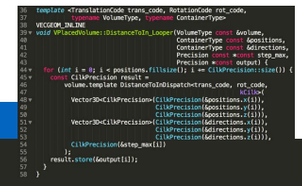
Single particle interface

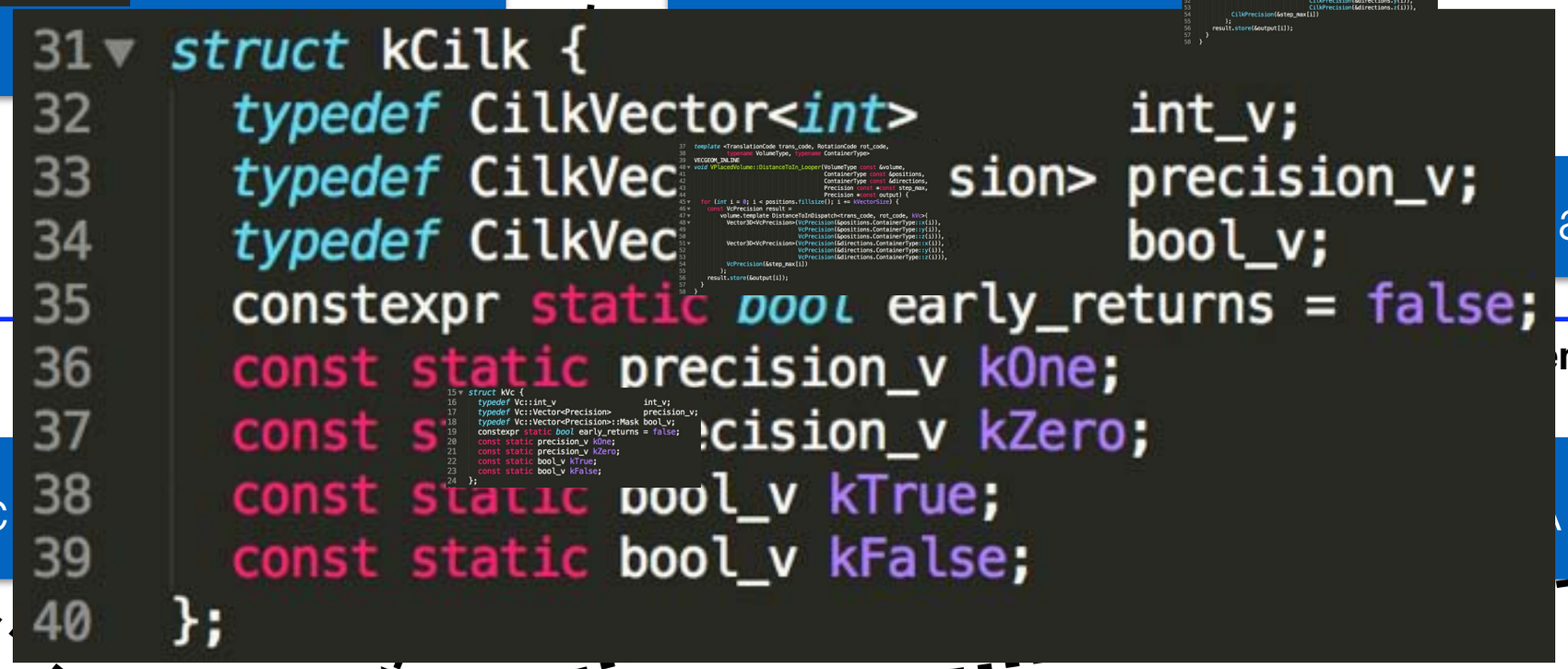
Vector interface

External CUDA kernels







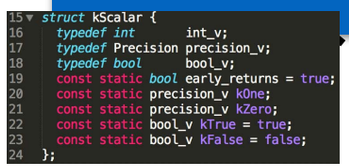


ad access

nd instantiation

Scalar Bac

A Backend



Dispatcher

C-like abstract kernel

Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

Scalar Looper

Cilk plus Looper

Vc Loope

Thread access

Backend instantiation

Scalar Backend

Vc Backend

Cilk plus Backend

CUDA Backend

Dispatcher

C-like abstract kernel

Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

Scal

d access

d instantiation

Scalar Backend

Vc Backend

Clik plus Backend

CUDA Backend

Dispatcher

C-like abstract kernel

Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

Scalar Looper

Cilk plus Looper

Vc Loope

Thread access

Backend instantiation

Scalar Backend

Vc Backend

Cilk plus Backend

CUDA Backend

Dispatcher

C-like abstract kernel

Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

```
template <TranslationCode trans_code, RotationCode rot_code>
VECGEOM_CUDA_HEADER_BOTH
VECGEOM_INLINE
typename Backend::precision_v PlacedBox::DistanceToInDispatch(
    Vector3D<typename Backend::precision_v> const &position,
    Vector3D<typename Backend::precision_v> const &direction,
    const typename Backend::precision_v step_max) const {

  typename Backend::precision_v output;
  BoxDistanceToIn<trans_code, rot_code, Backend>(
    unplaced_box()->dimensions(),
    *this->matrix(),
    position,
    direction,
    step_max,
    &output
  );
  return output;
}
```

Dispatcher

C-like abstract kernel

Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

Scalar Looper

Cilk plus Looper

Vc Loope

Thread access

Backend instantiation

Scalar Backend

Vc Backend

Cilk plus Backend

CUDA Backend

Dispatcher

C-like abstract kernel

Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

Scalar Backend

hread access

ckend instantiation

DA Backend

Dispatcher

C-like abstract kernel

Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

Scalar Looper

Cilk plus Looper

Vc Loope

Thread access

Backend instantiation

Scalar Backend

Vc Backend

Cilk plus Backend

CUDA Backend

Dispatcher

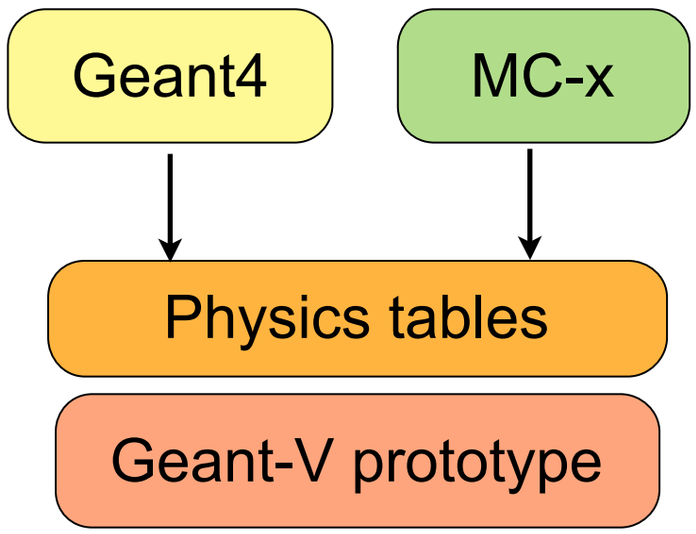
C-like abstract kernel

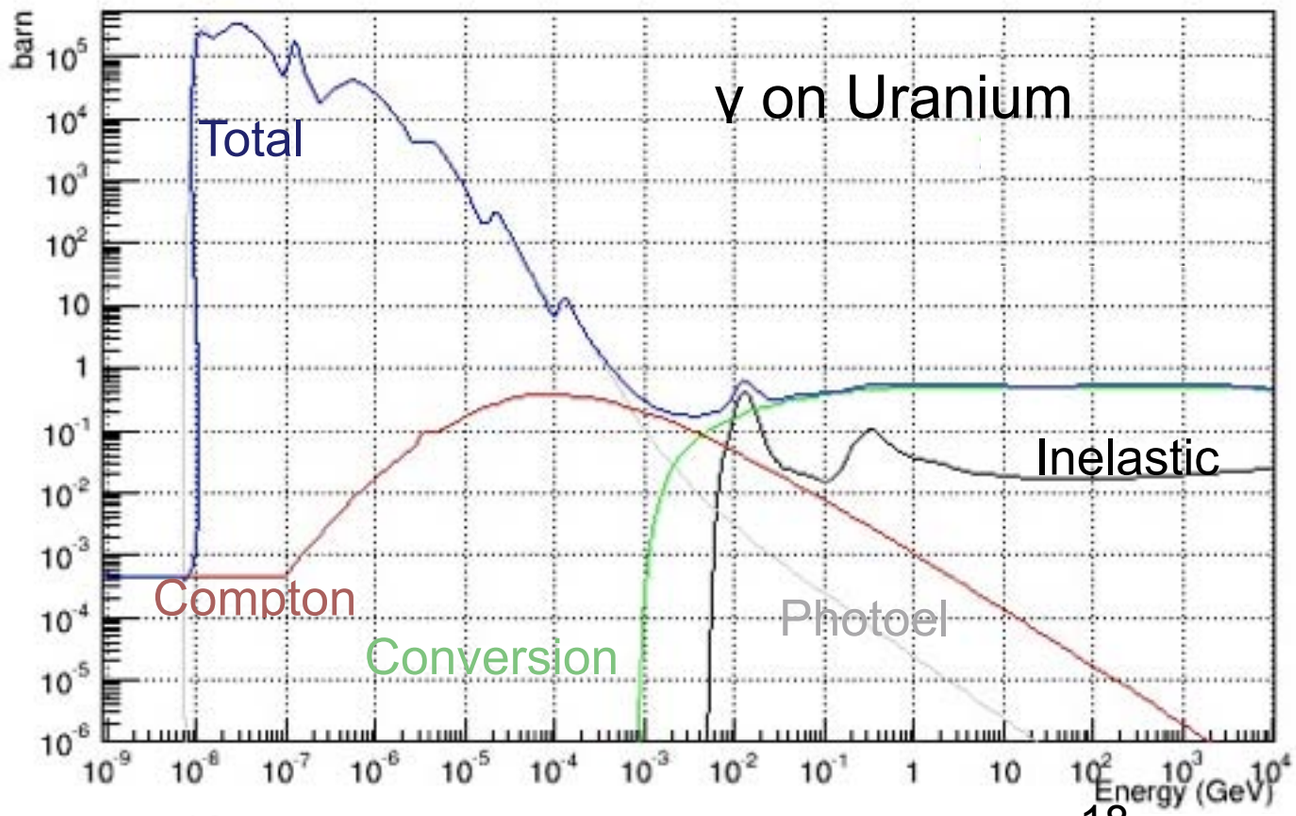
USolid

- Independent “application agnostic” library for geometry
  - In collaboration with AIDA SFT activity
- Already offered as an option for Geant4
- New high performance version will be progressively introduced
- Used both by Geant4 and GeantV
- Thinking about UGeom and UMaterial

Physics

- A lightweight physics for realistic shower development
- Select the major mechanisms
  - Bremsstrahlung, e+ annihilation, Compton, Decay, Delta ray, Elastic hadron, Inelastic hadron, Pair production, Photoelectric, Capture + dE/dx & MS
- Tabulate all x-secs (100 bins -> 90MB)
- Generate (10-50) final states (300kB per final state & element)
- Not good as Geant4, but it could be the seed of a fast simulation option
- Independent from the MonteCarlo that actually generates the tables

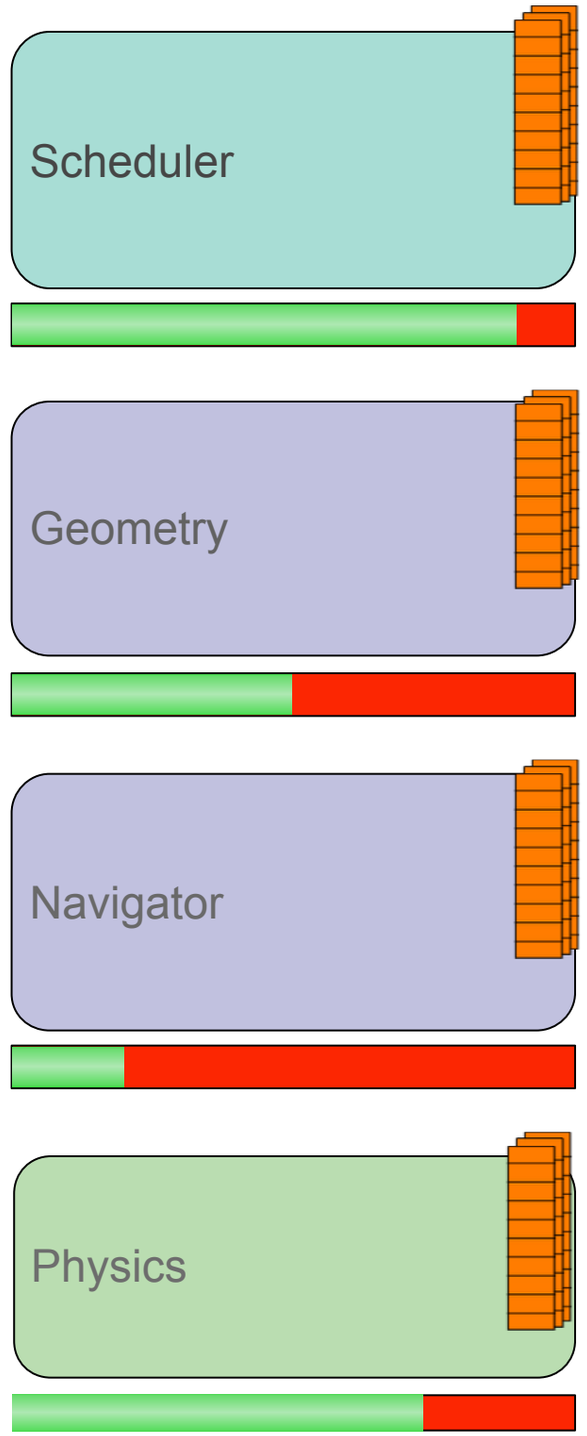




18

Where are we now?

- Scheduler
  - The new version, hopefully improved of the scheduler has been committed and we are testing it
- Solids
  - The proof or principle that we can achieve large speedups (3-5+) is there, however a lot of work lays ahead
- Navigator
  - “Percolating” vectors through the navigator is a difficult business. We have a simplified navigator that achieves that, but more work is needed here
- Physics
  - Can generate x-secs and final states and sample them, but there are still many points to be clarified with Geant4 experts



Physics Specific issues

- Many issues opened in Jira about physics interactions
- “SFT-private” version of G4 created
- Verification of tabulated x-sections against x-sections as sampled and data
- Make common tools for accessing x-sections & process sampling (SampDisOne) a standard facility in Geant4

Planning

- Common program of work FNAL - GeantV (- Geant4)
  - P.Canal, S.Jun, G.Lima
- Development of common CPU / GPU enabled prototype
- Development of MIC version in parallel (Intel)

Milestones

- April 11th
  - Simple (EX03-like) benchmark
  - Connect tabulated physics with Geant4 & prototype
  - Port optimised Bremsstrahlung to prototype & reuse in Geant4
  - USolid and UGeom to run Ex03 in prototype
- July 31st
  - *Move to Geant4 10.0*
  - *Nightly build system*
  - *Magnetic Field transport*
  - Setups intermediate (3-5 v-solids) and full (CMS, top 5 v-solids)
  - Vector Compton & abstraction
  - Test all combos (GPU, CPU and MIC) of the prototype
- November
  - Complete EM Physics (for Std EM Physics) & one bi-model process
  - Full set of Primitive Shapes and voxelisation

Testing - benchmarking

Geant4

G4 nav

G4 phys

G4 geom

Prototype

Prot nav

Tab phys

UGeom*

With small, large & venti geometries

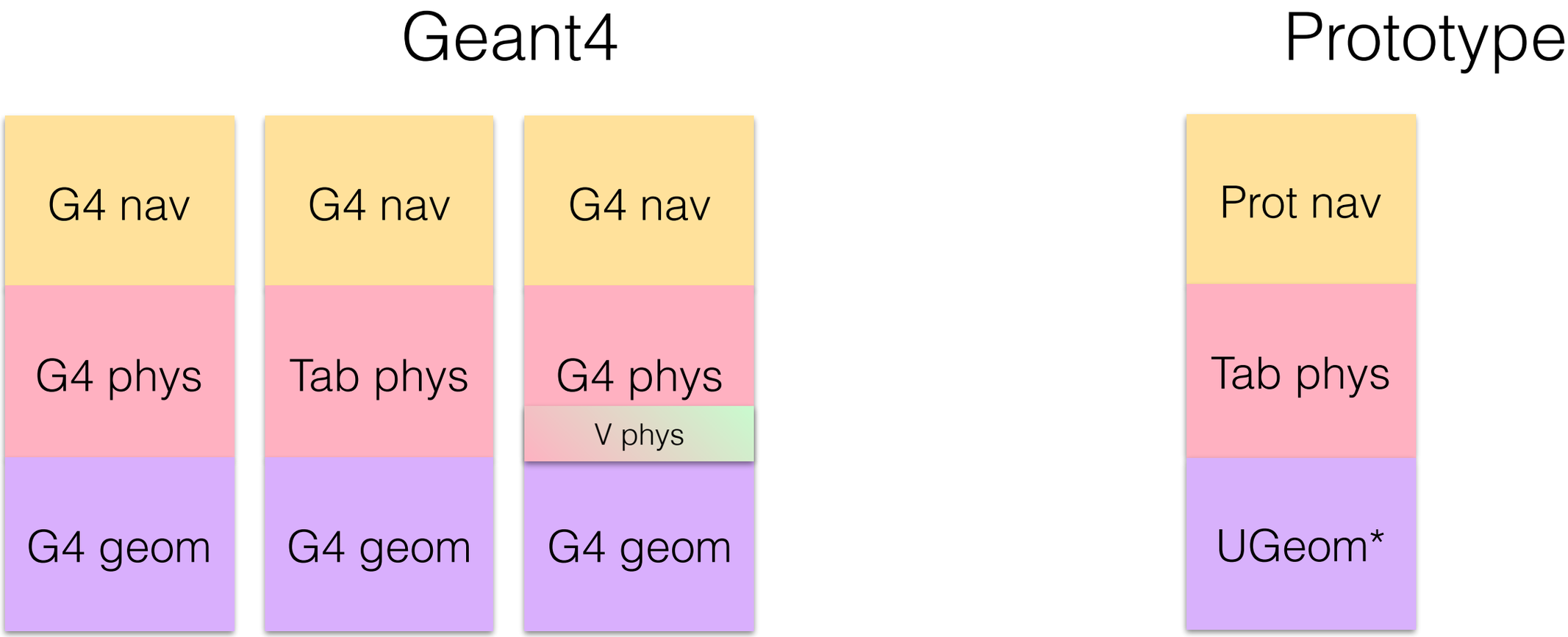
*UGeom == USolid + navigation

Testing - benchmarking

With small, large & venti geometries

*UGeom == USolid + navigation

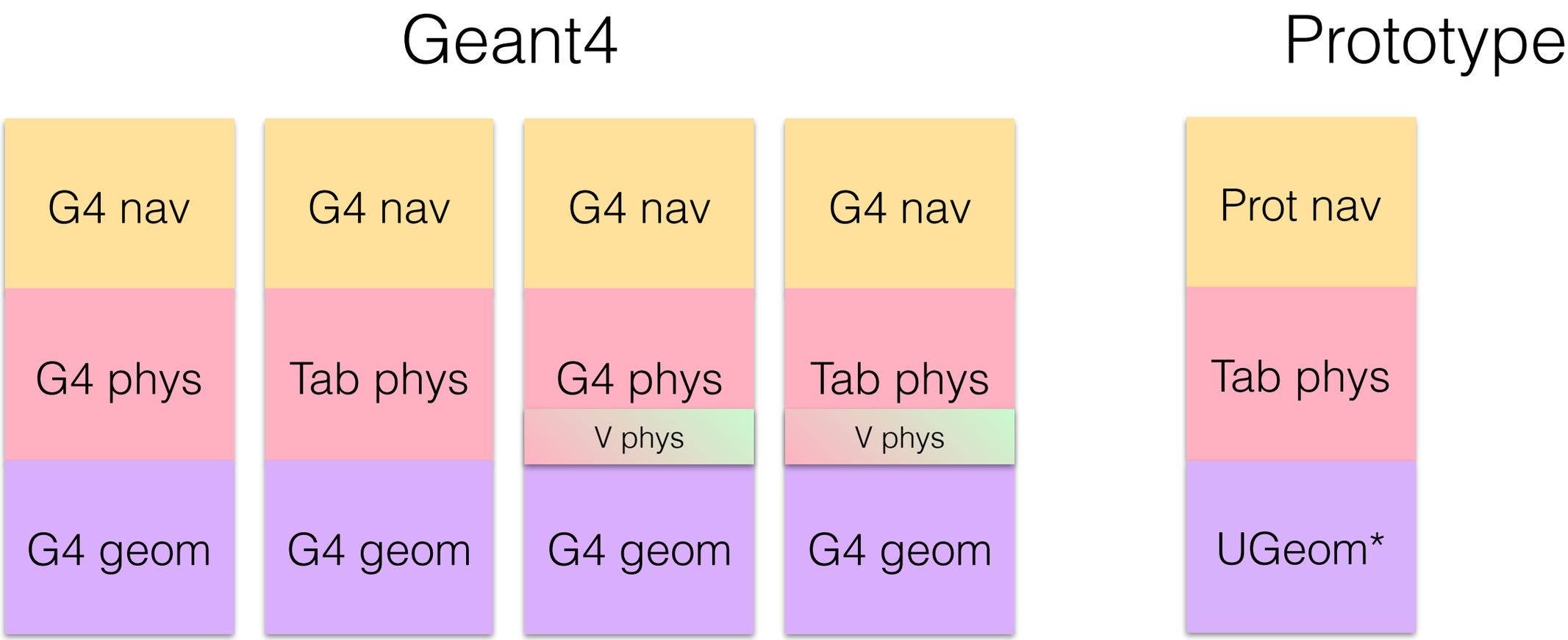
Testing - benchmarking



With small, large & venti geometries

*UGeom == USolid + navigation

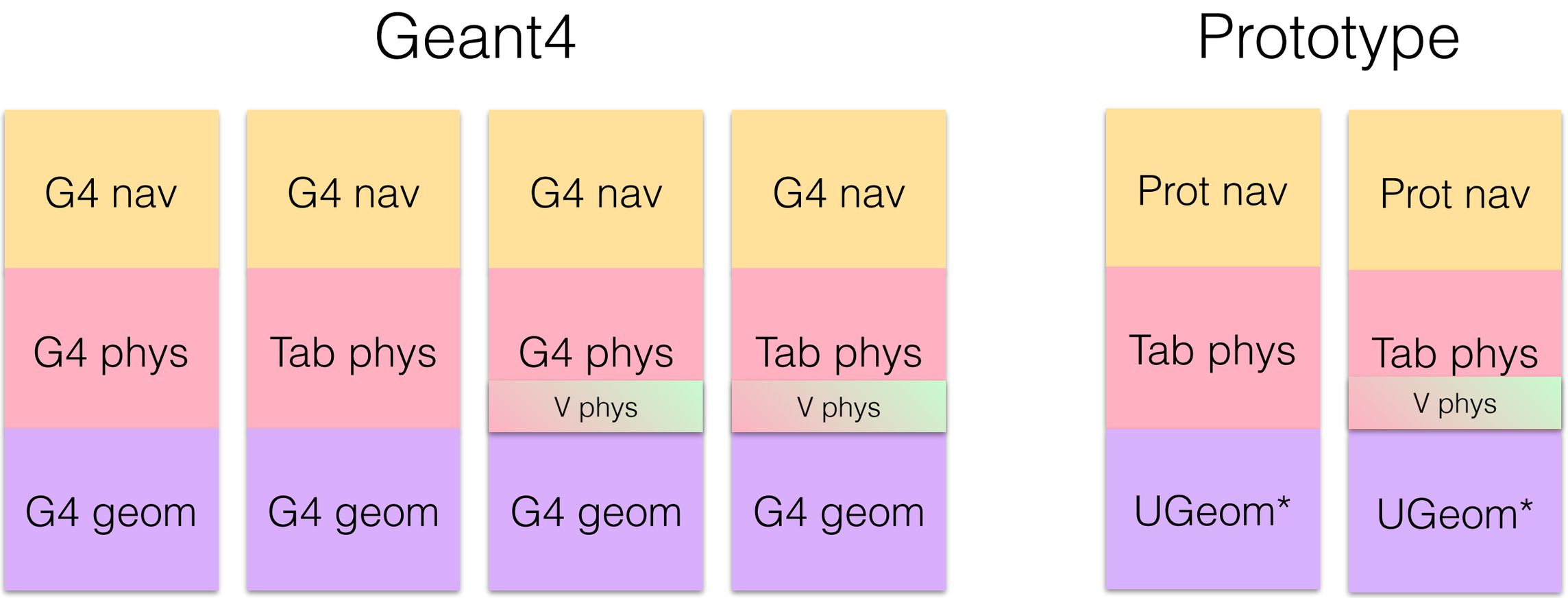
Testing - benchmarking



With small, large & venti geometries

*UGeom == USolid + navigation

Testing - benchmarking



With small, large & venti geometries

*UGeom == USolid + navigation

Outlook

- Parallel development for this year
- Vector physics code may find its way into Geant4 if there is a clear performance benefit
- The vector prototype has made good progress and the collaboration with FNAL is very promising
- We should start a reflection on the design of a new software suite taking advantage from the major upgrade / rewrite that will be necessary to optimise our code