

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

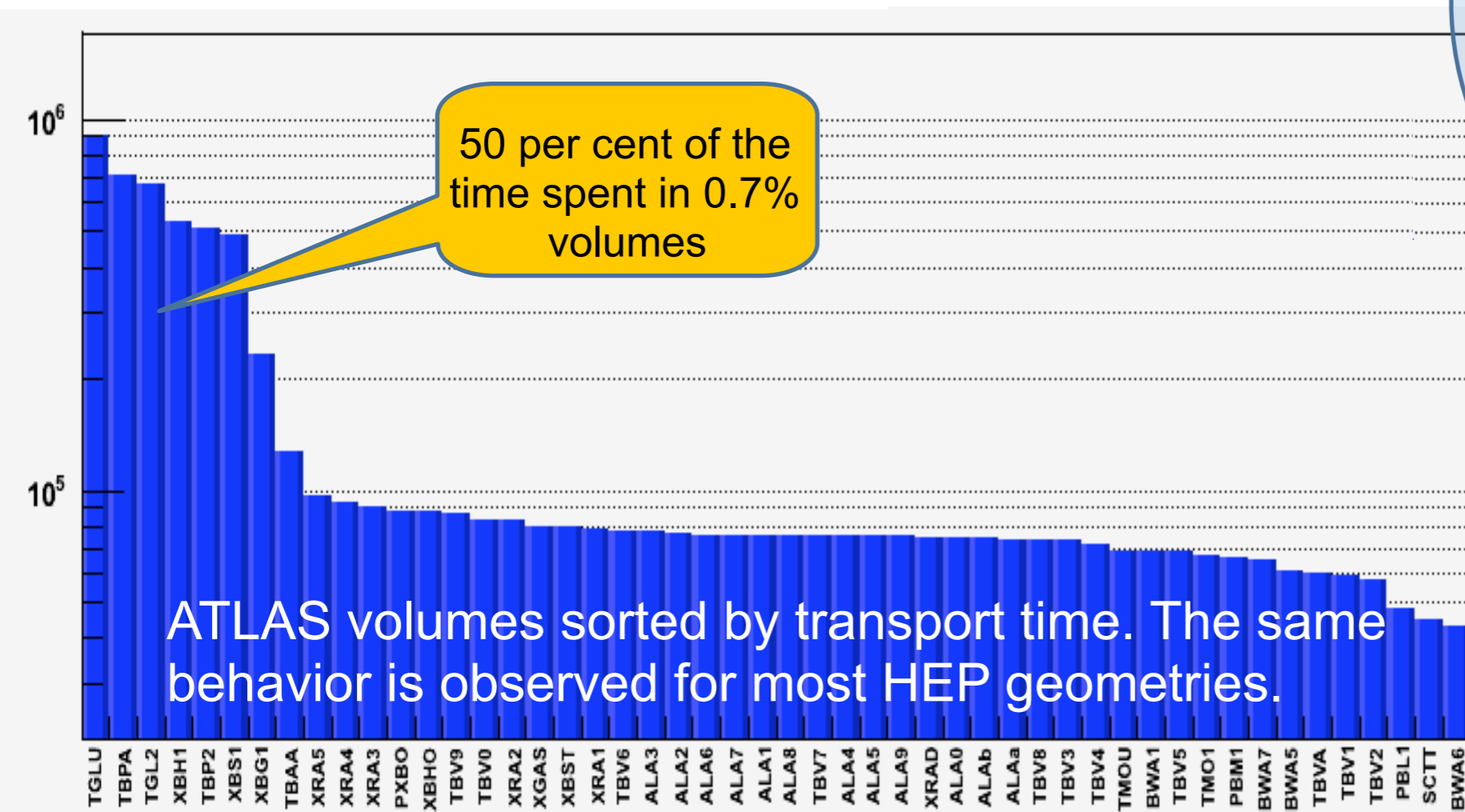
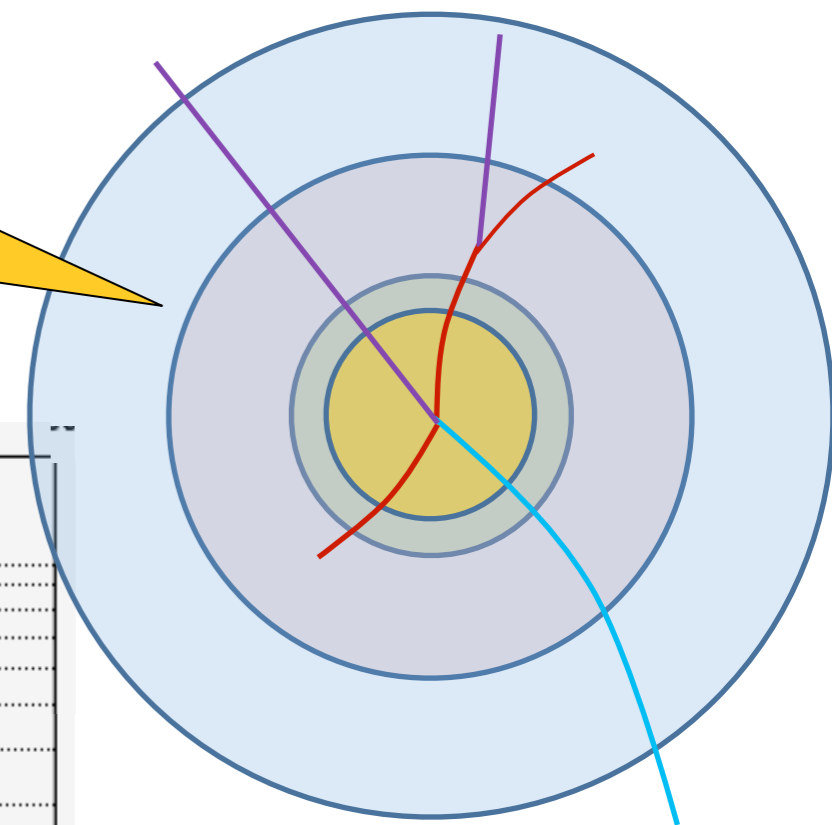
Expected limits on performance scaling (multiplied)			
	SIMD	ILP	HW
THEORY	8	32	43.2
OPTIMISED	6	9.43	11.79
HEP	1	0.8	1

OpenLab@CHEP12

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

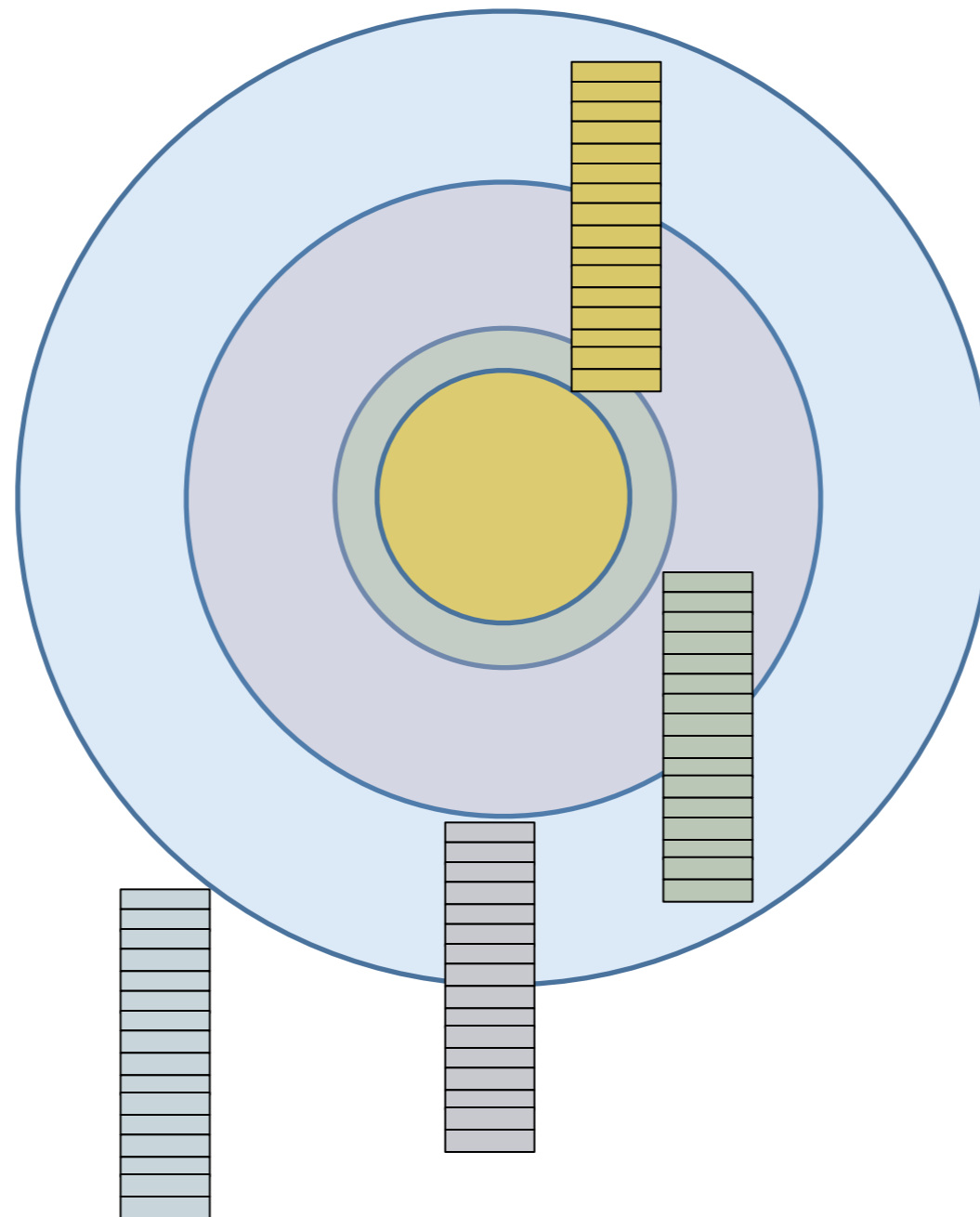
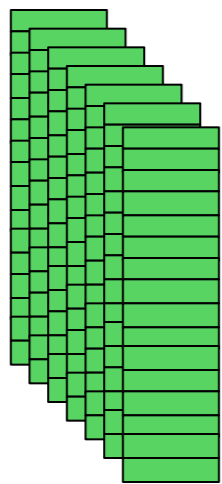


- Navigating very large data structures
- No locality
- OO abused: very deep instruction stack
- Cache misses

Introduced basketized transport

Deal with particles in parallel

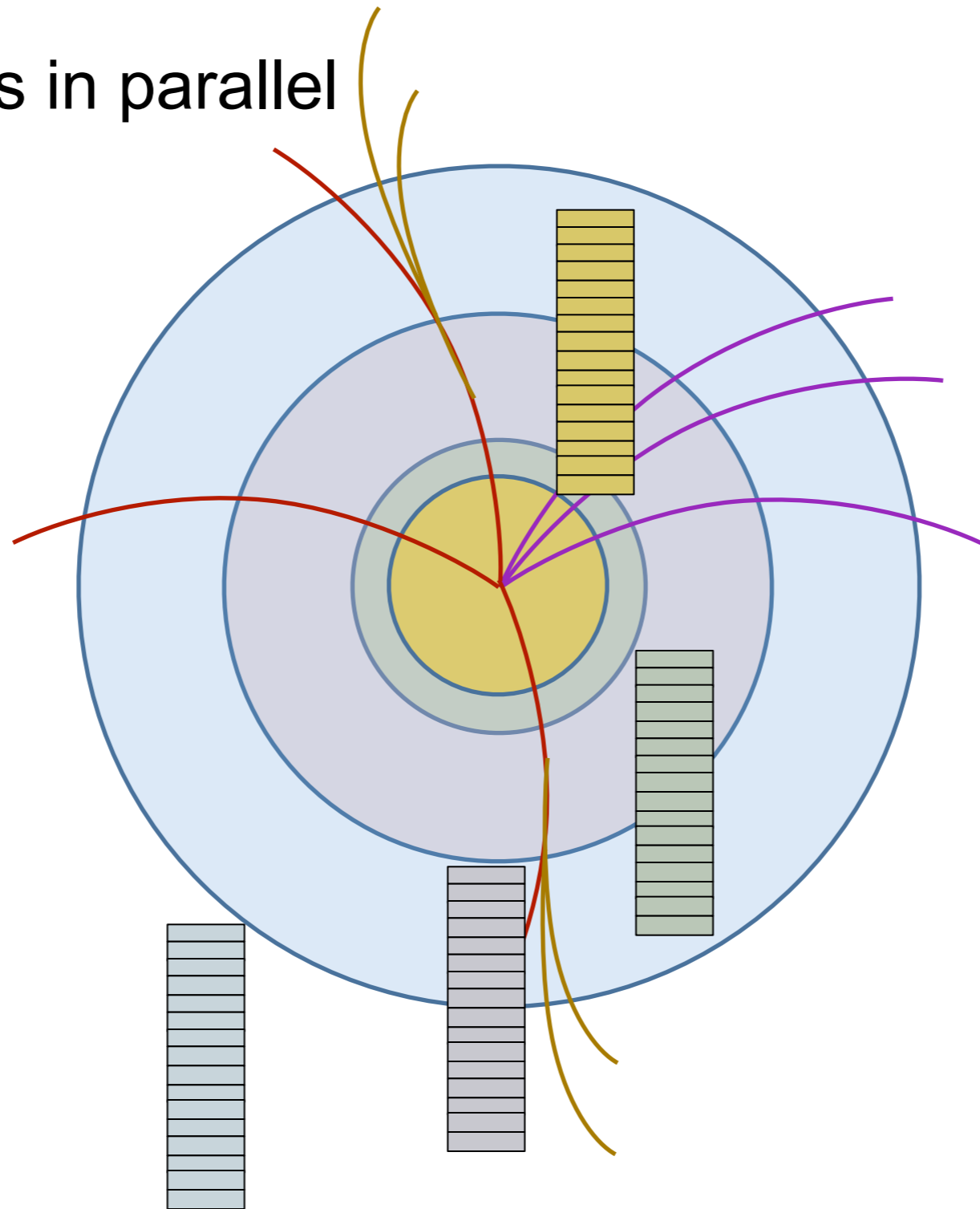
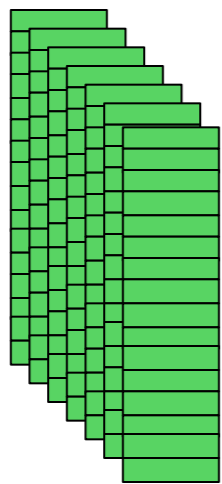
Output buffer(s)



Introduced basketized transport

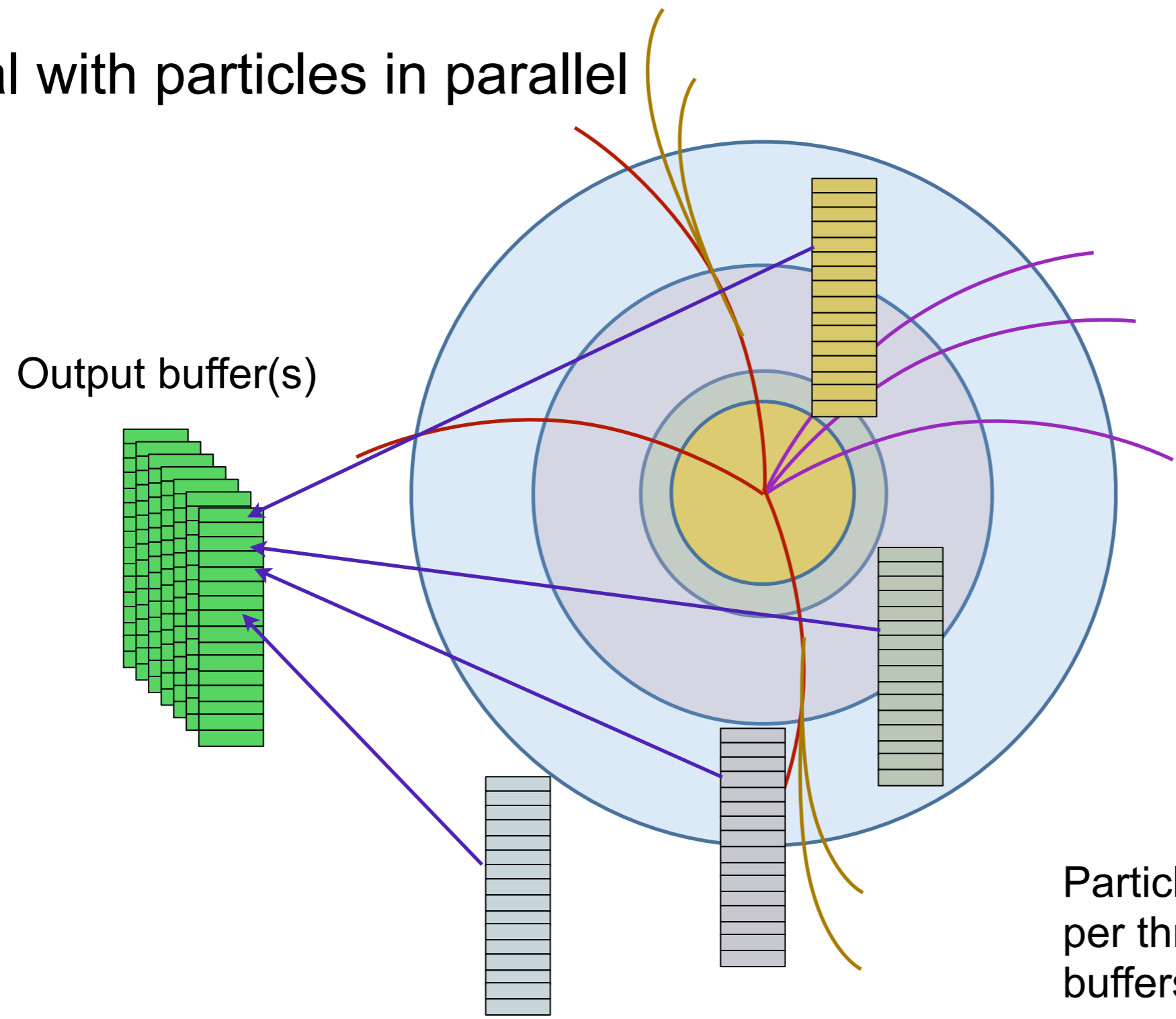
Deal with particles in parallel

Output buffer(s)



Introduced basketized transport

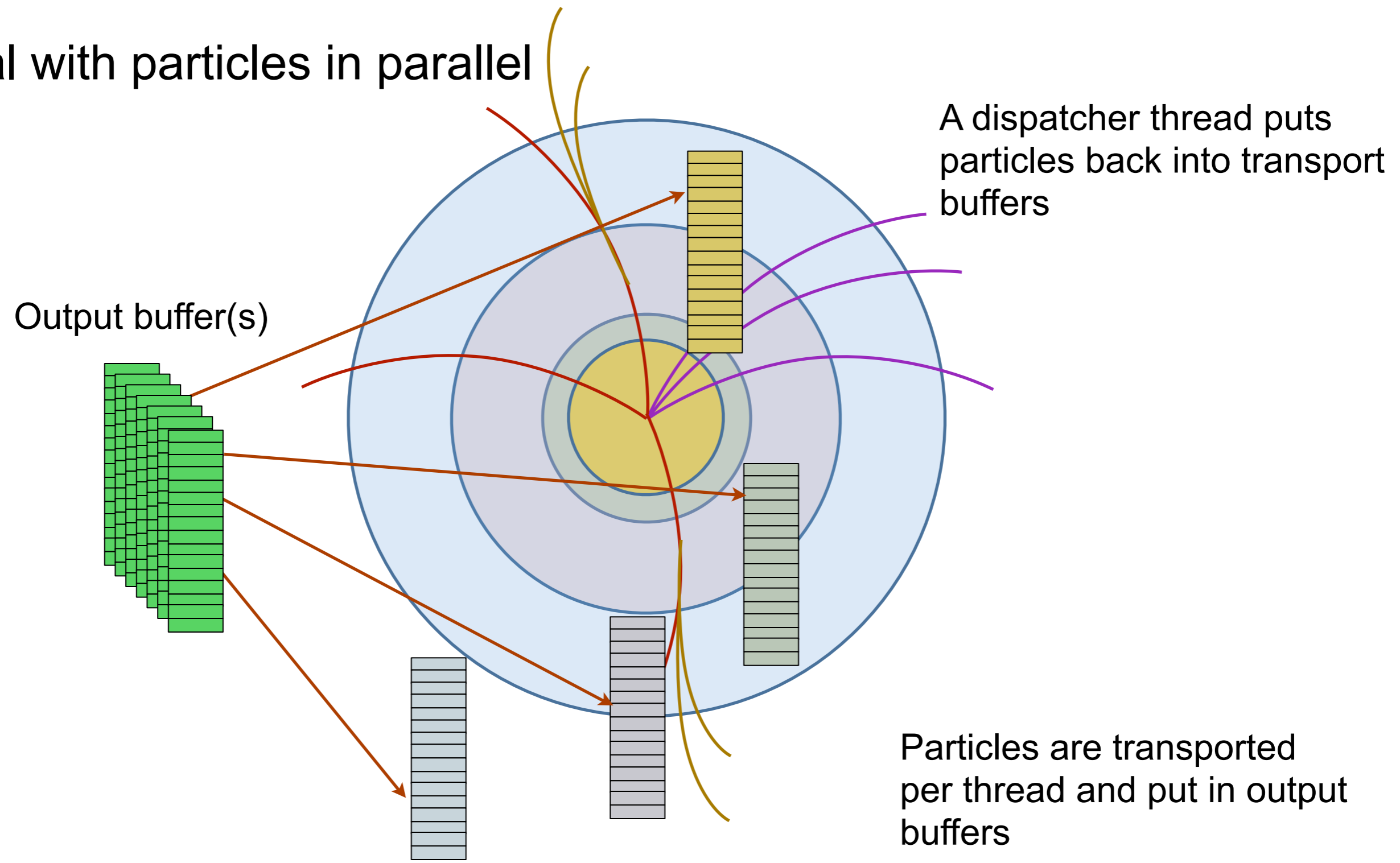
Deal with particles in parallel



Particles are transported per thread and put in output buffers

Introduced basketized transport

Deal with particles in parallel



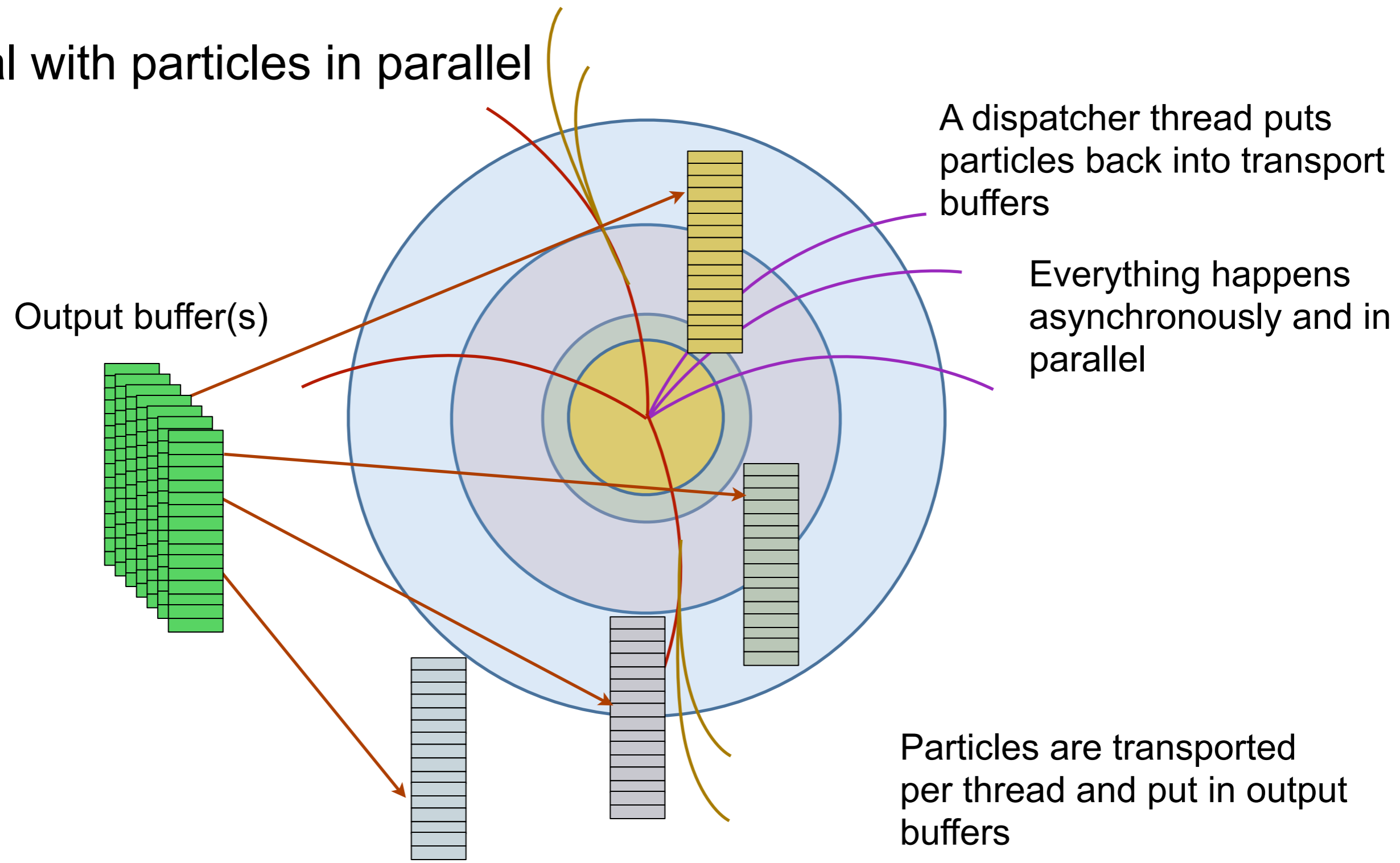
A dispatcher thread puts particles back into transport buffers

Output buffer(s)

Particles are transported per thread and put in output buffers

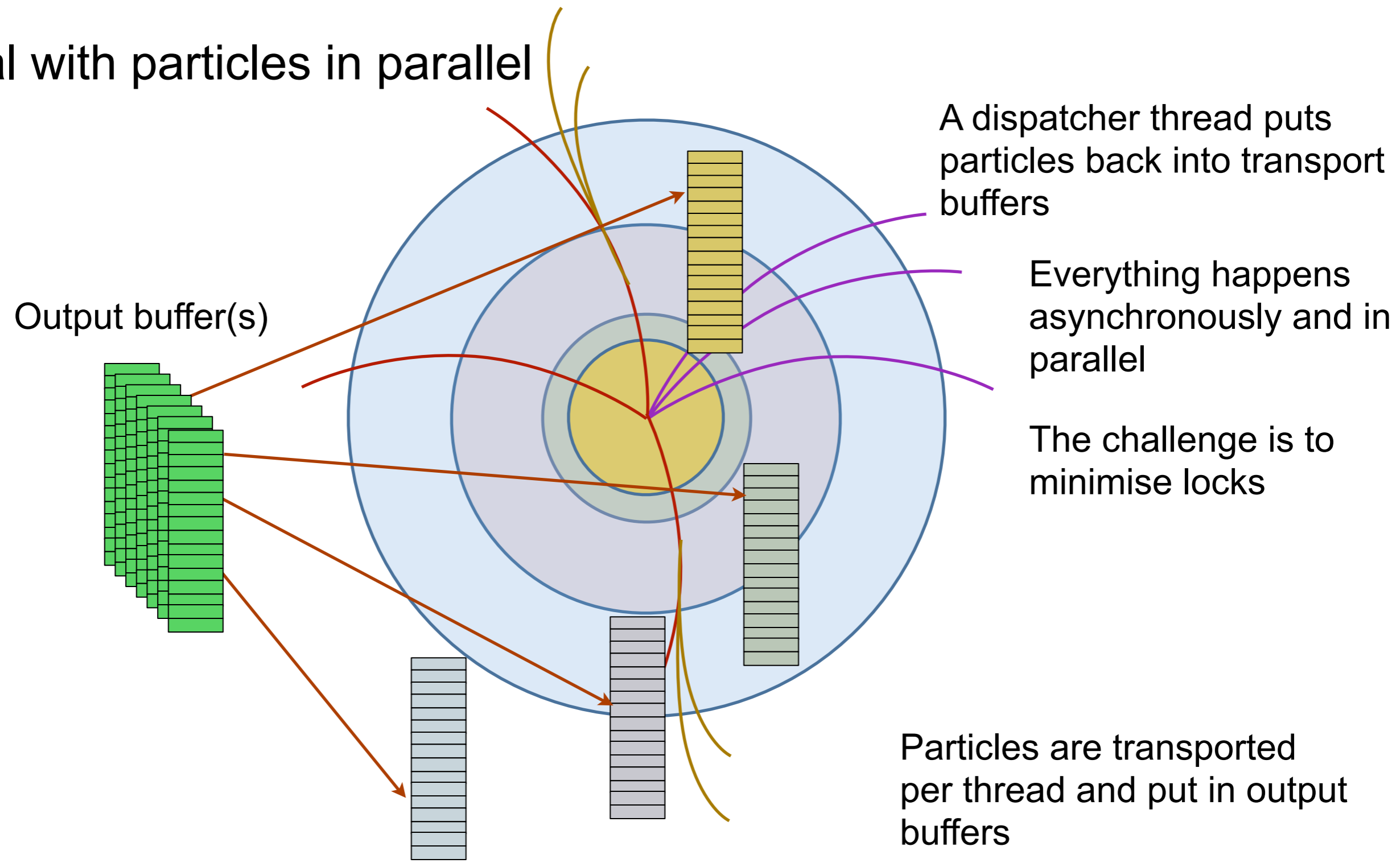
Introduced basketized transport

Deal with particles in parallel



Introduced basketized transport

Deal with particles in parallel



A dispatcher thread puts particles back into transport buffers

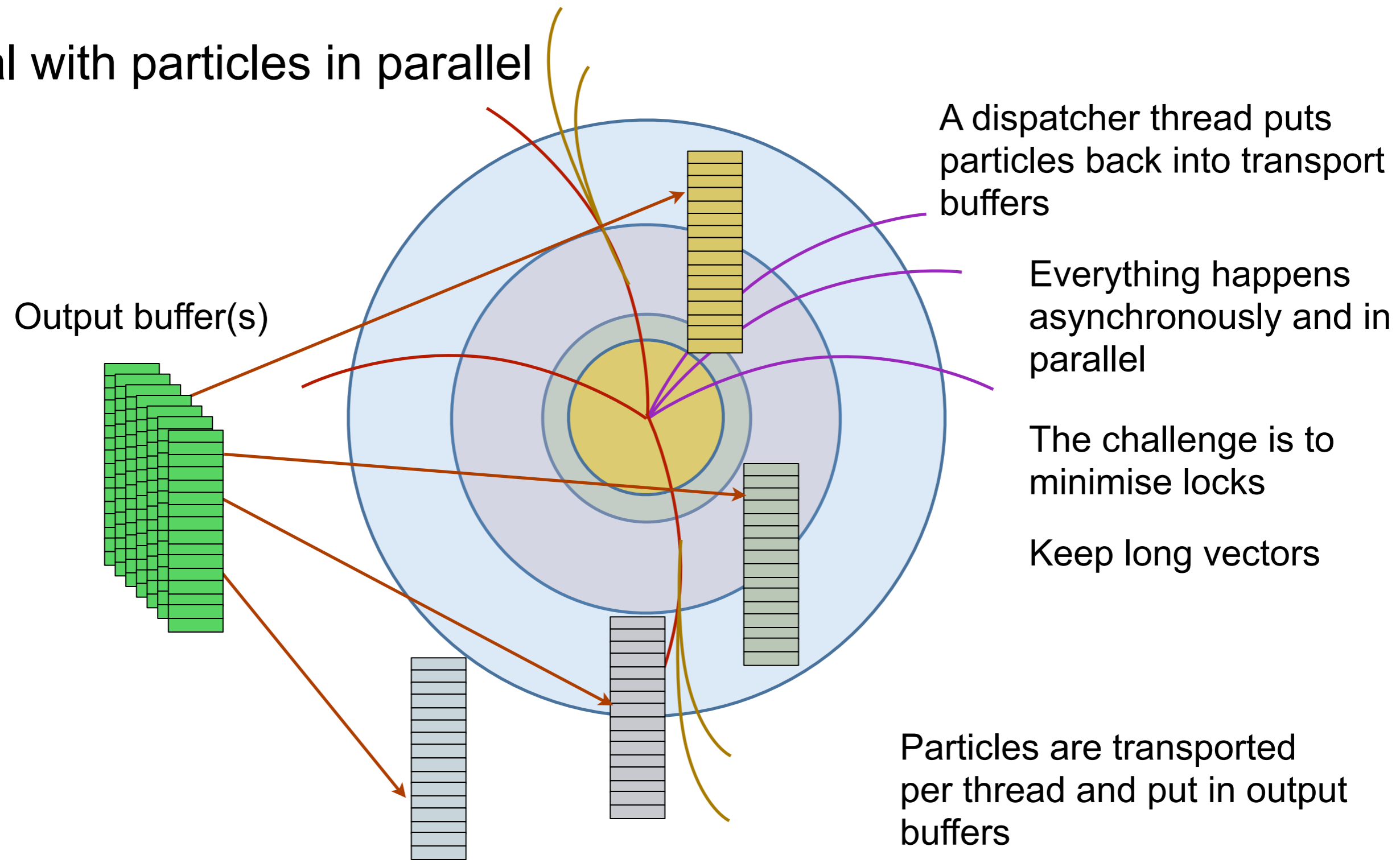
Everything happens asynchronously and in parallel

The challenge is to minimise locks

Particles are transported per thread and put in output buffers

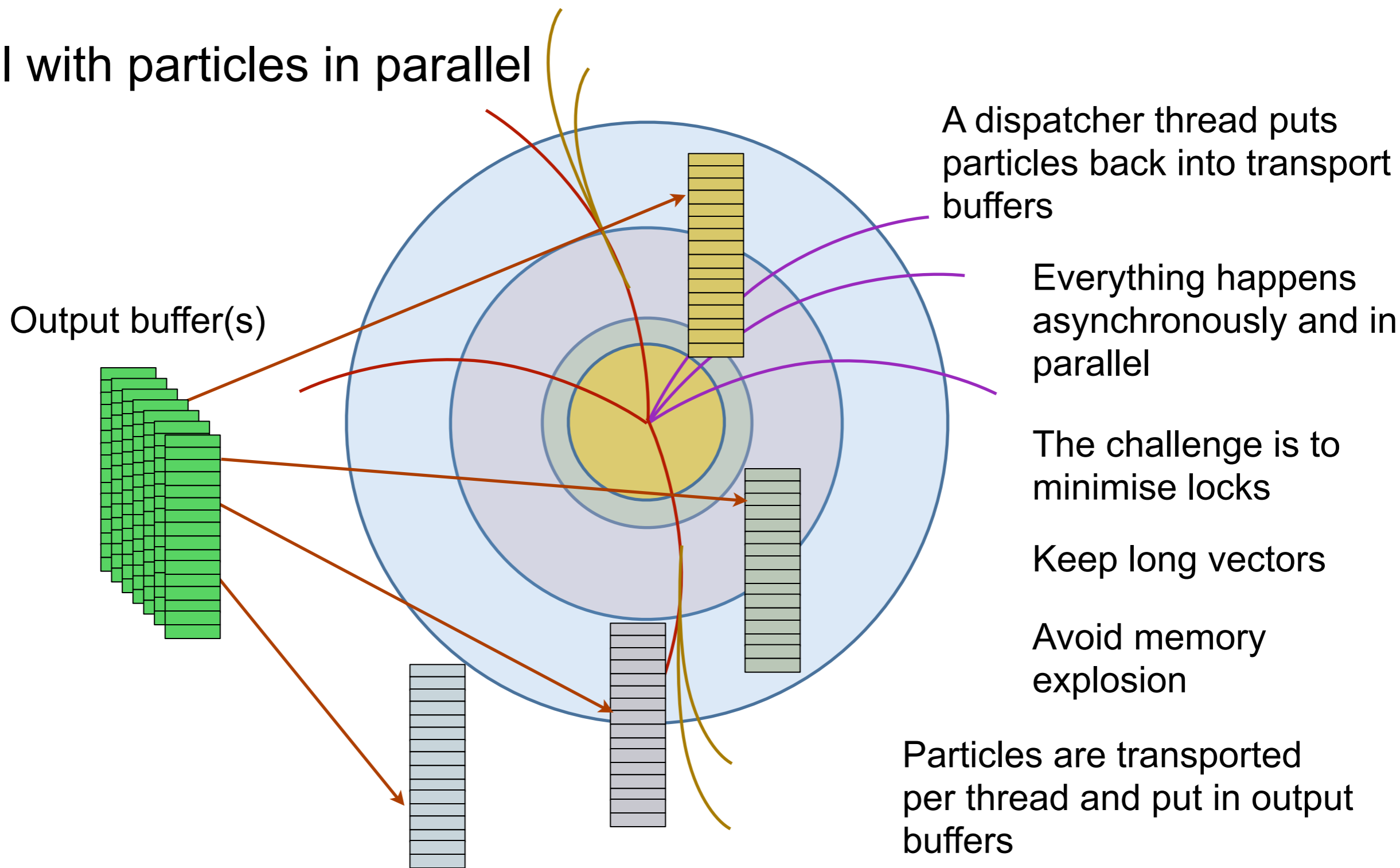
Introduced basketized transport

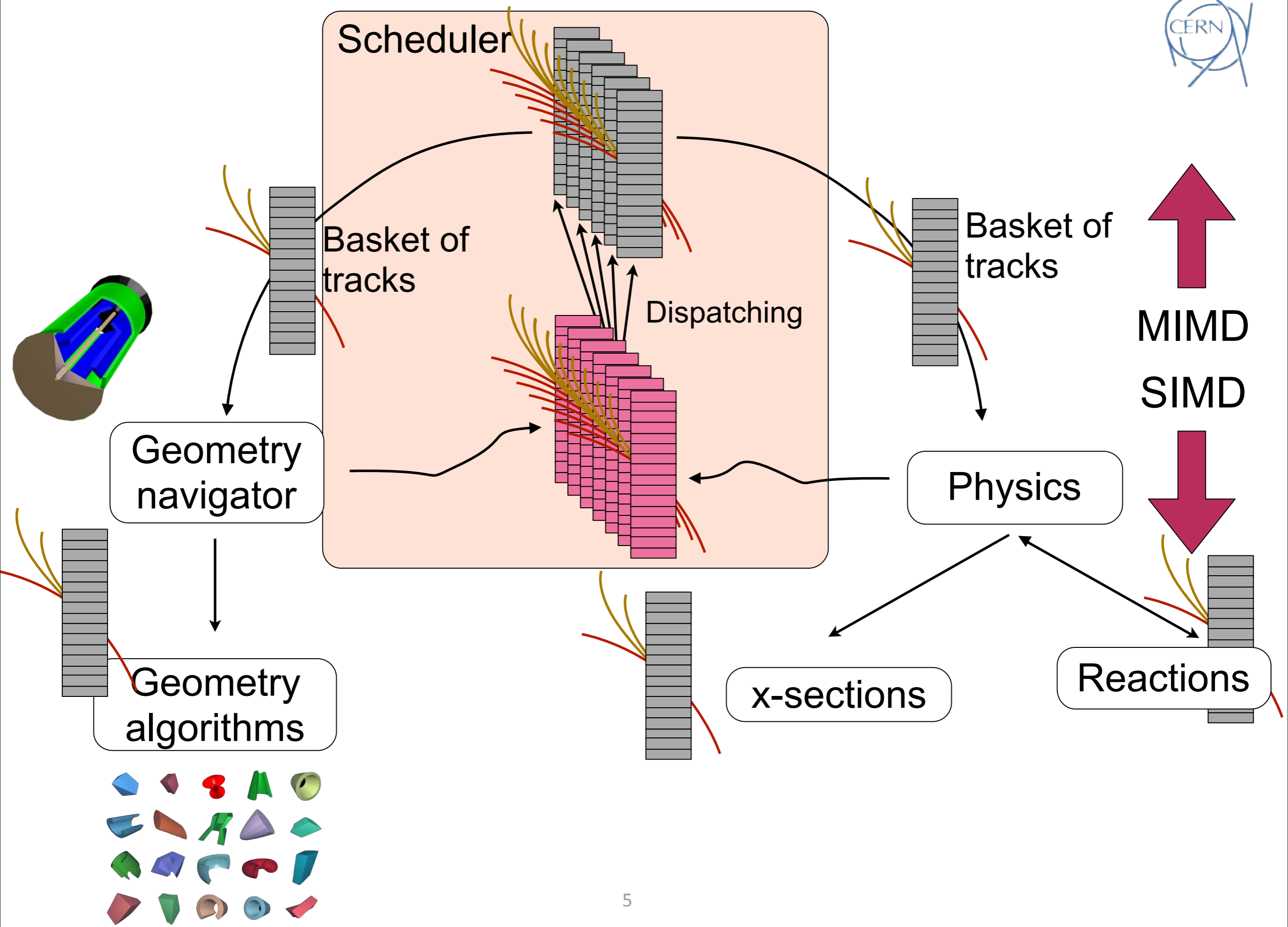
Deal with particles in parallel

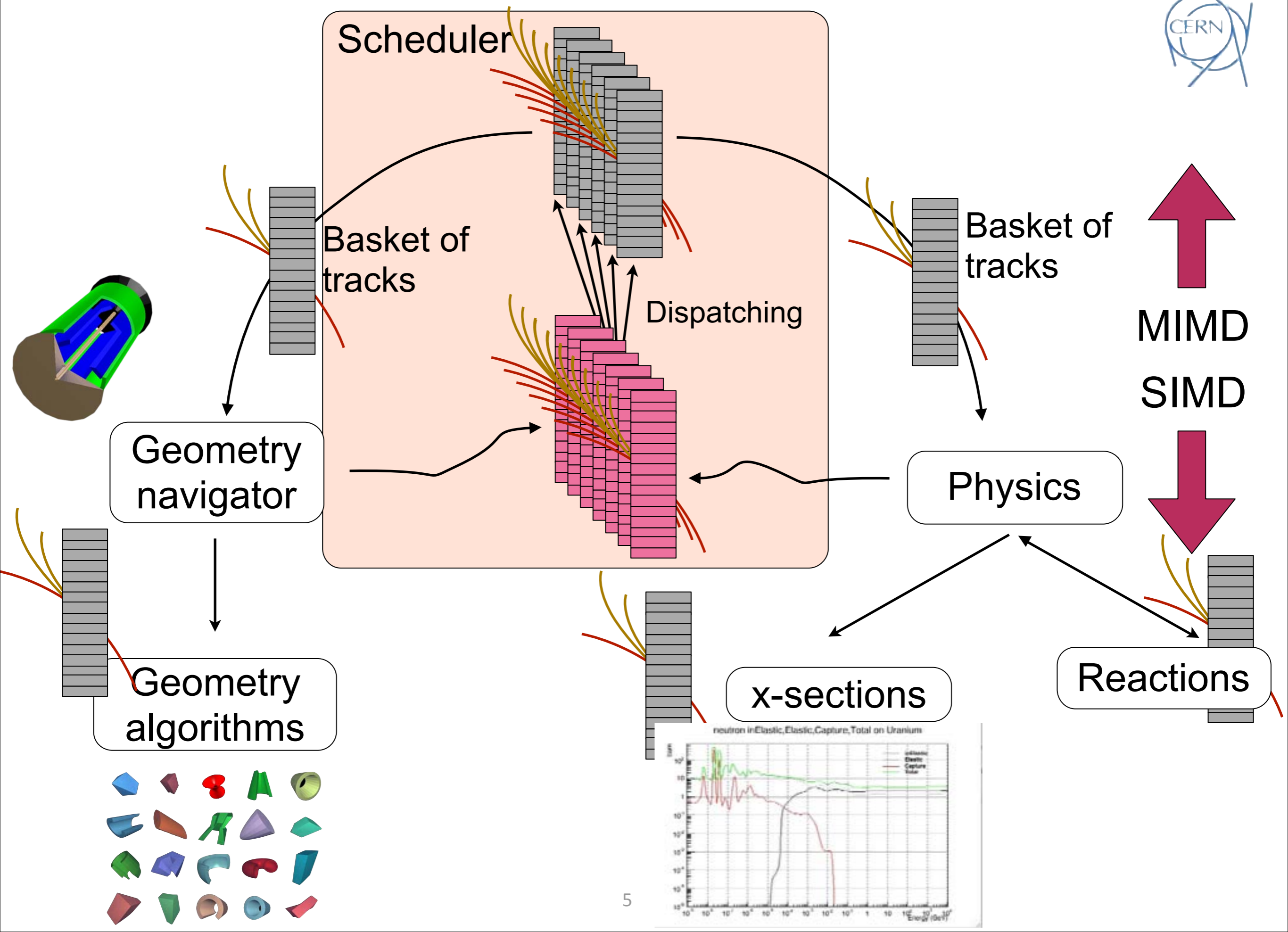


Introduced basketized transport

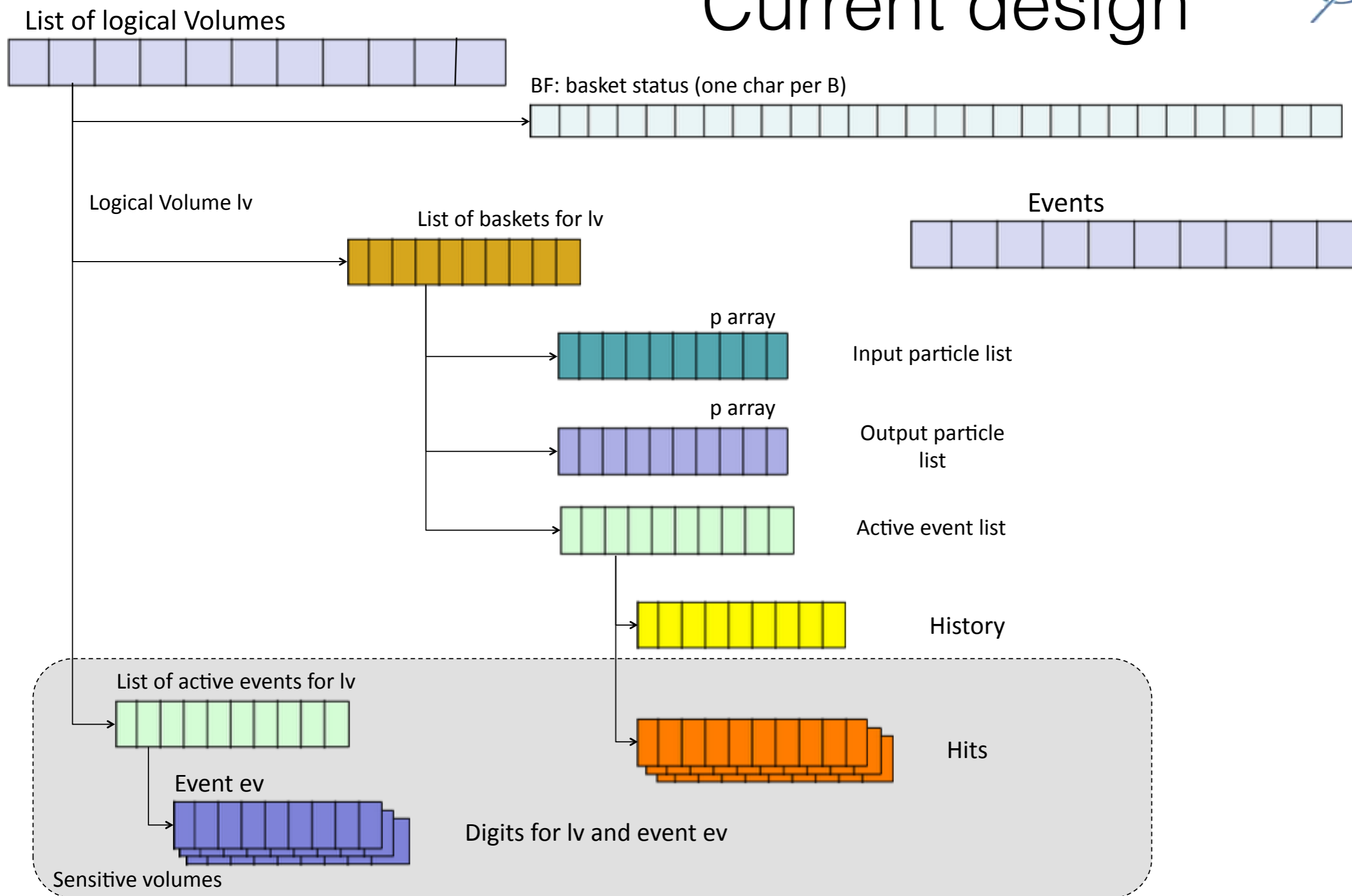
Deal with particles in parallel



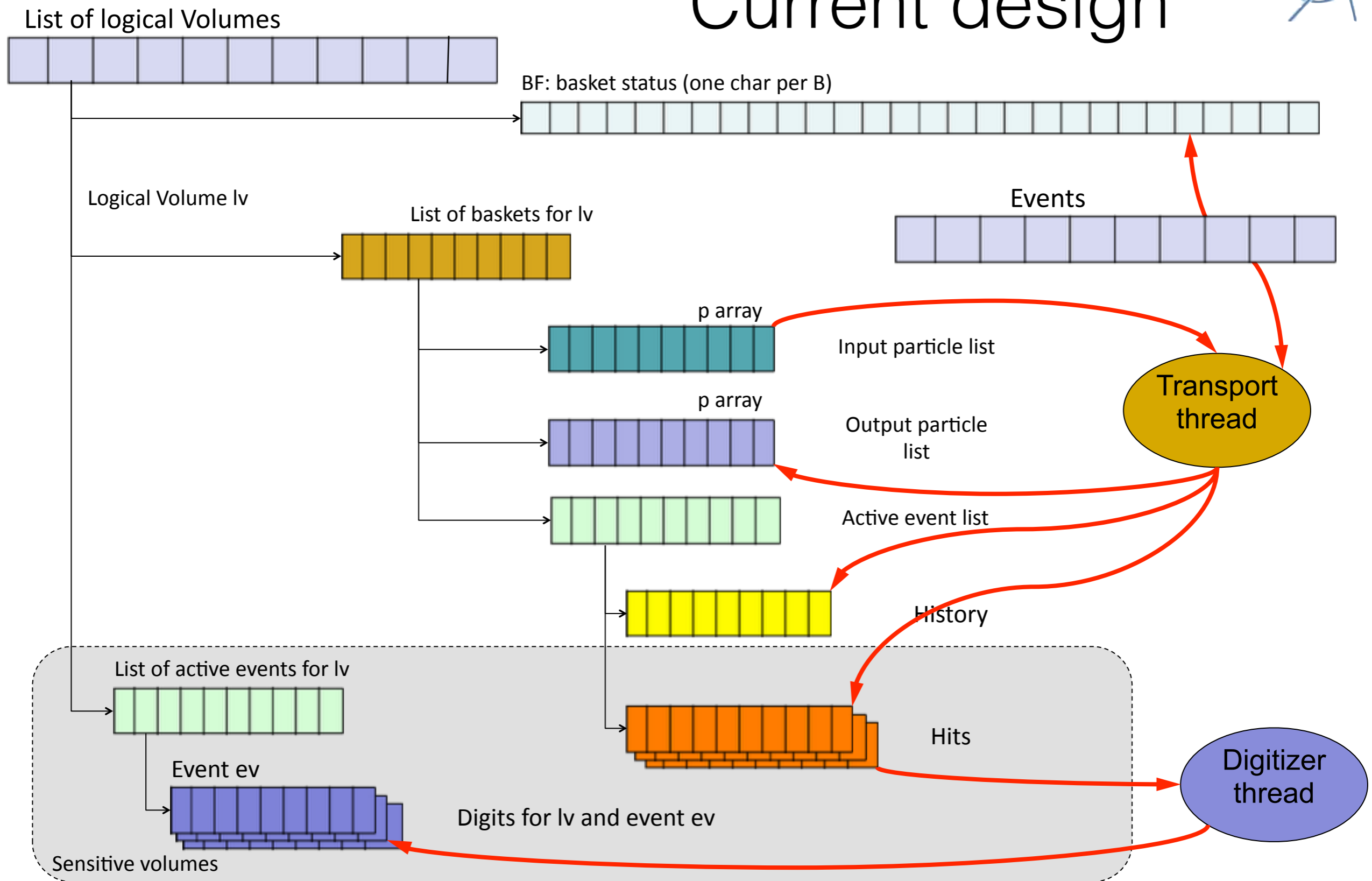




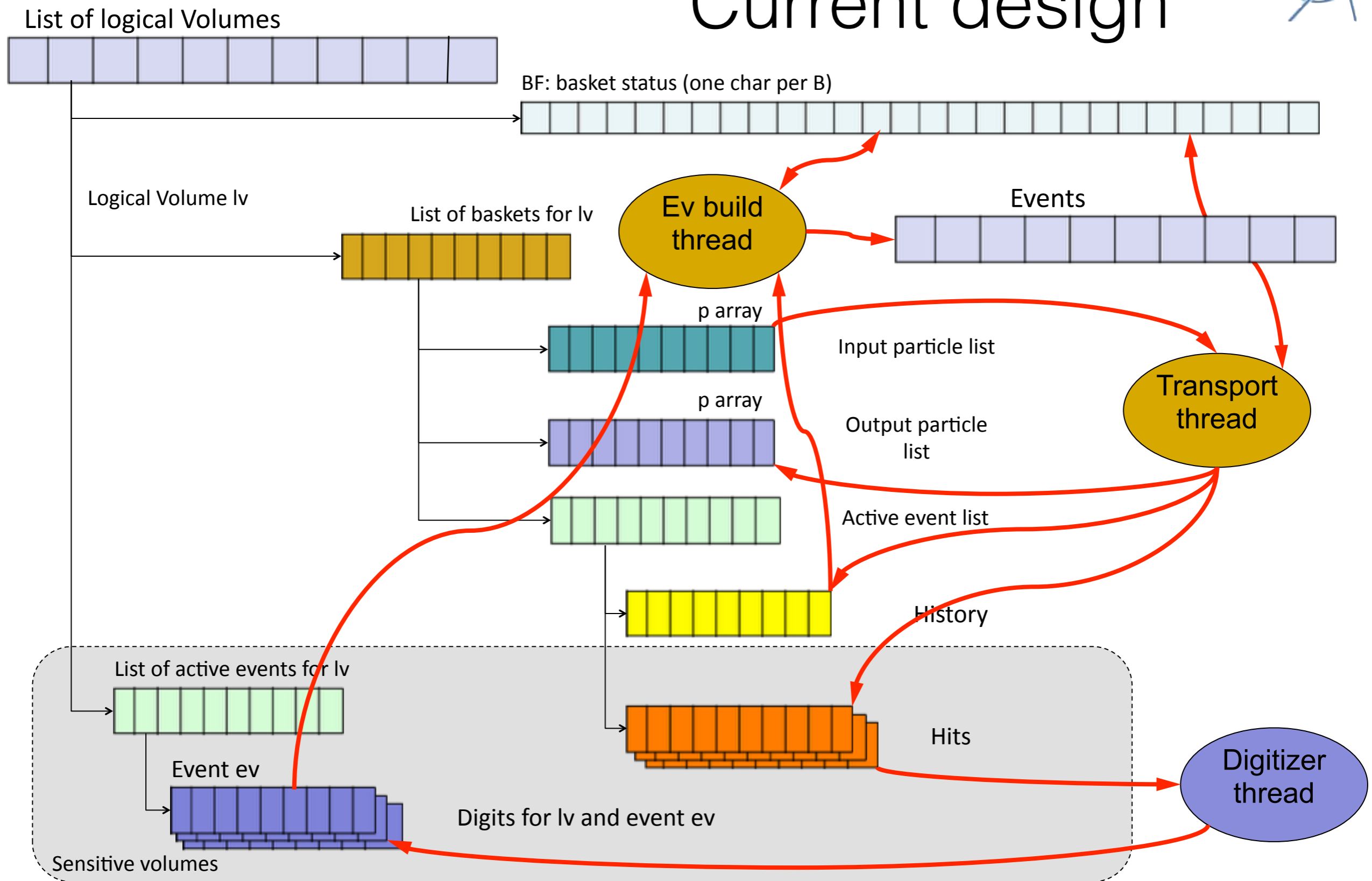
Current design



Current design



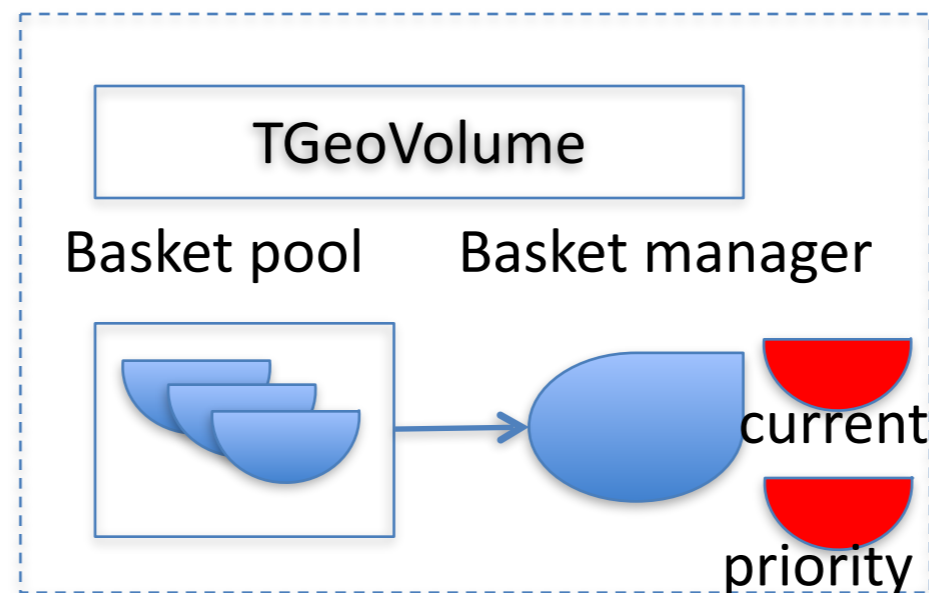
Current design



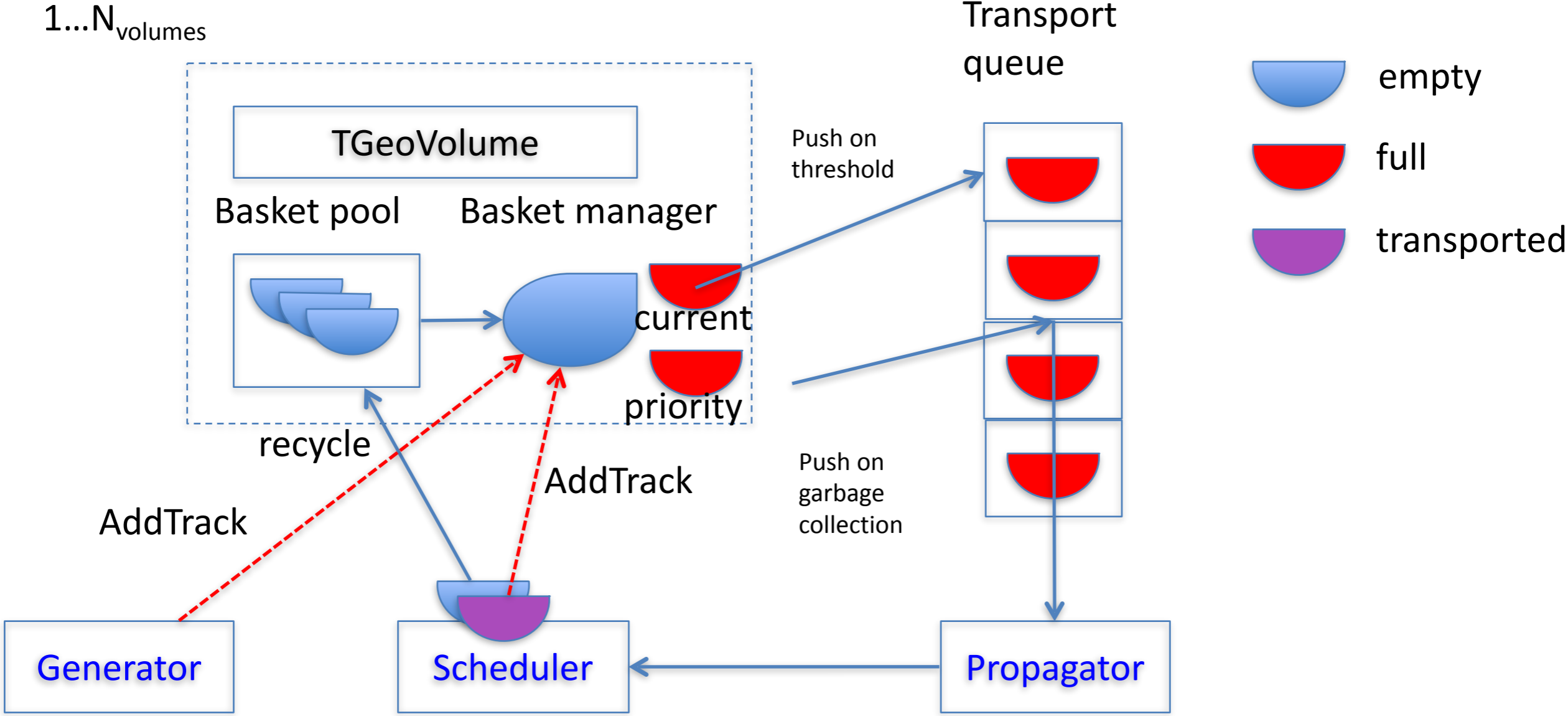
Basket managers

- 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
 - $\text{Threshold}(\text{vol}) = \text{Ntracks_in_flight}(\text{vol}) / 2N_threads$ rounded to %4 (min 4, max 256)

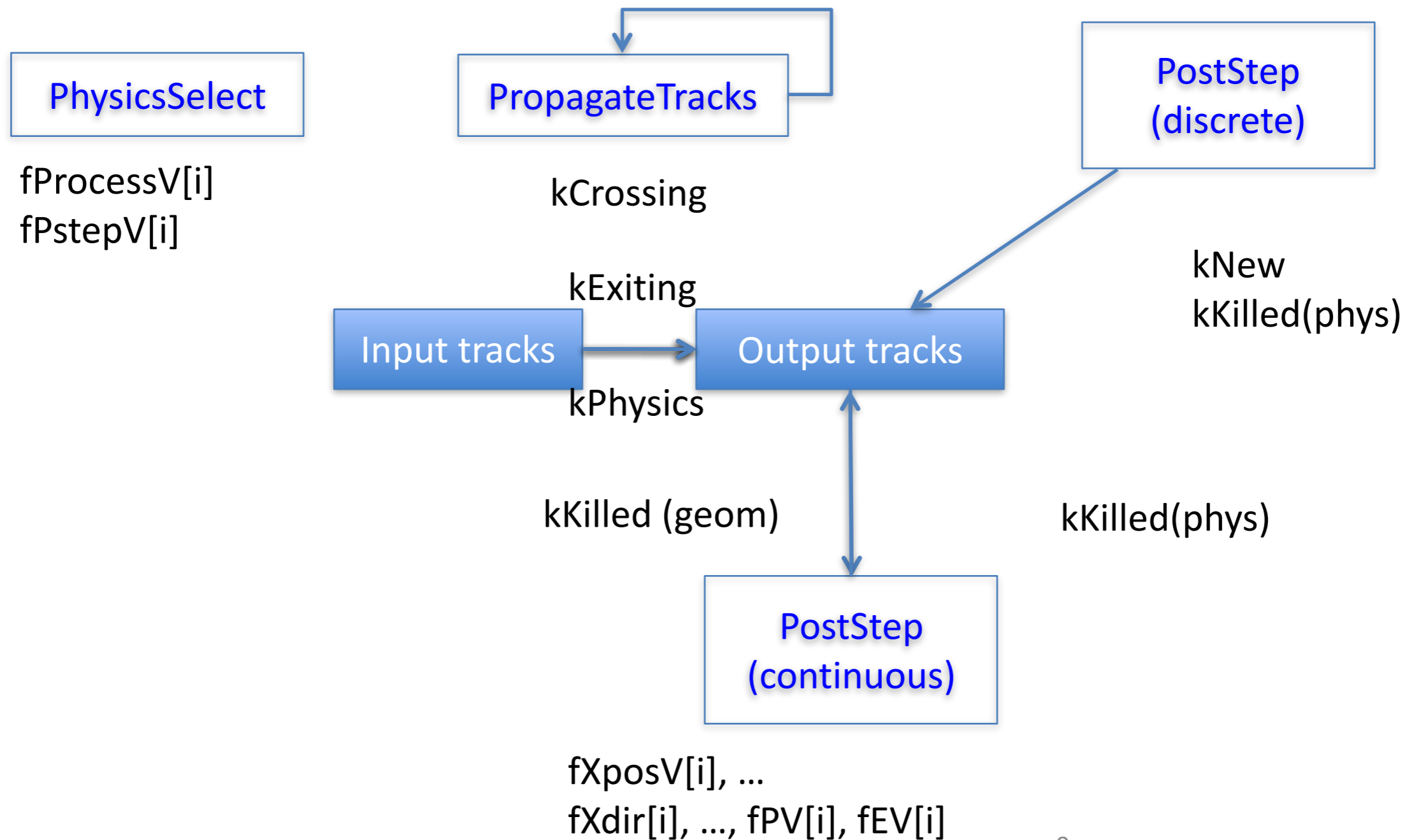
1...N_{volumes}



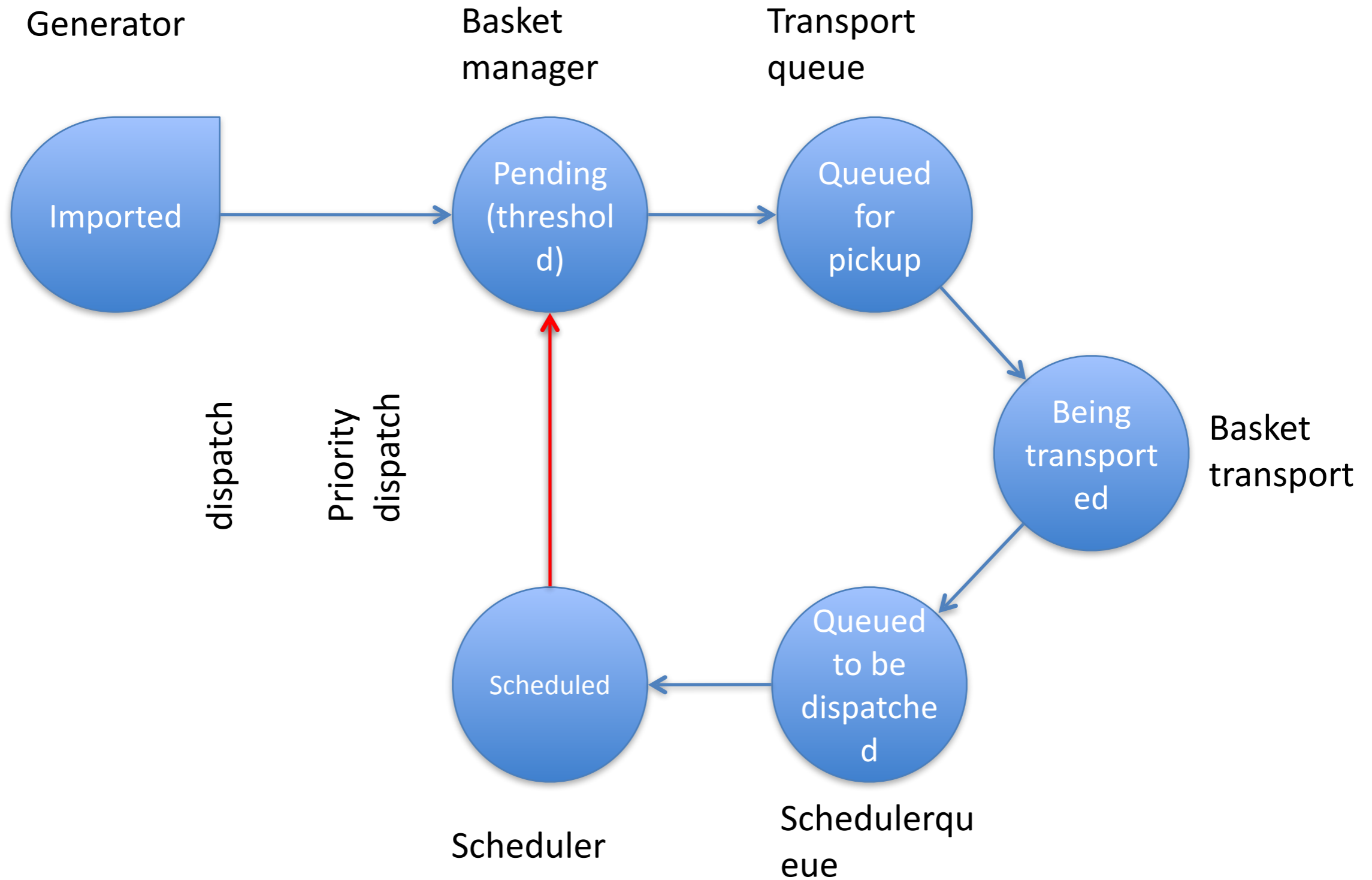
Basket lifecycle



Basket transport



Track stages

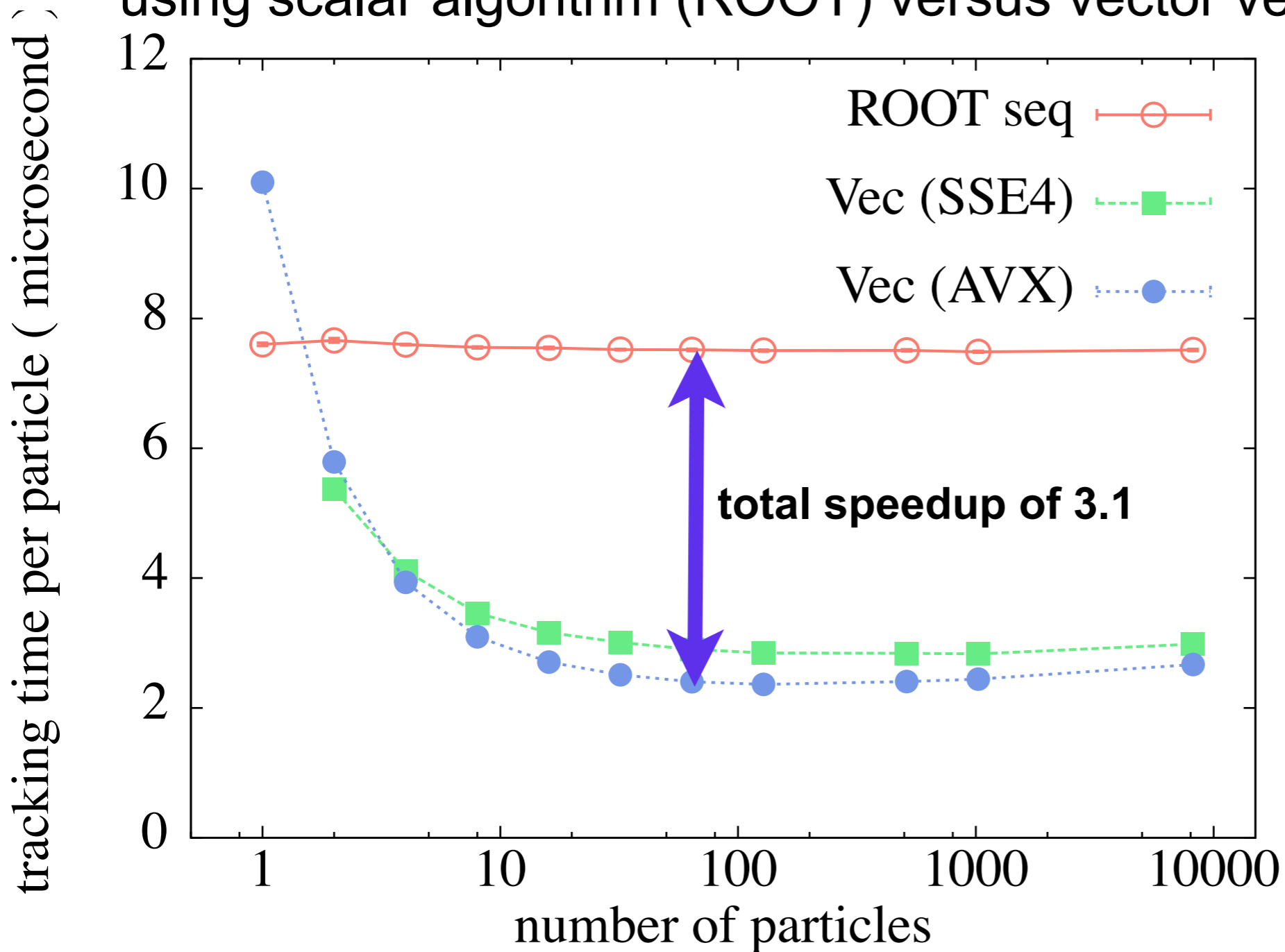


Scheduling policies

- 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 is 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!)

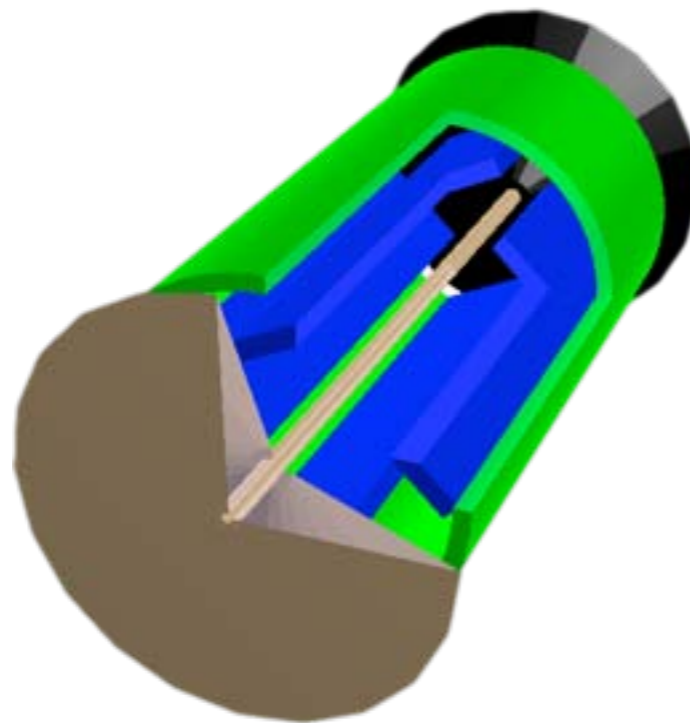
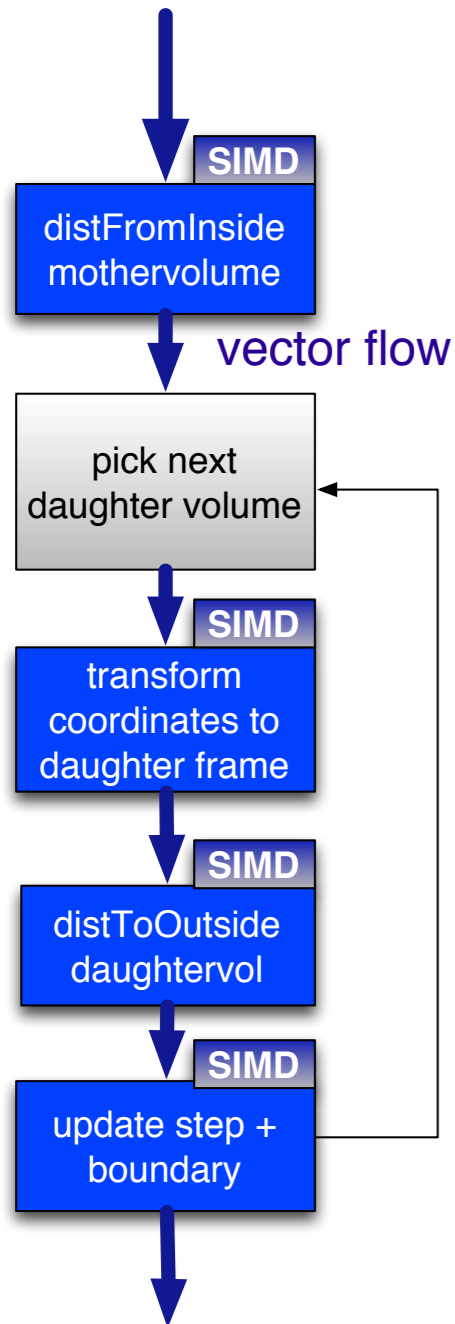
- Time of processing/navigating N particles (P repetitions) using scalar algorithm (ROOT) versus vector version



- 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)

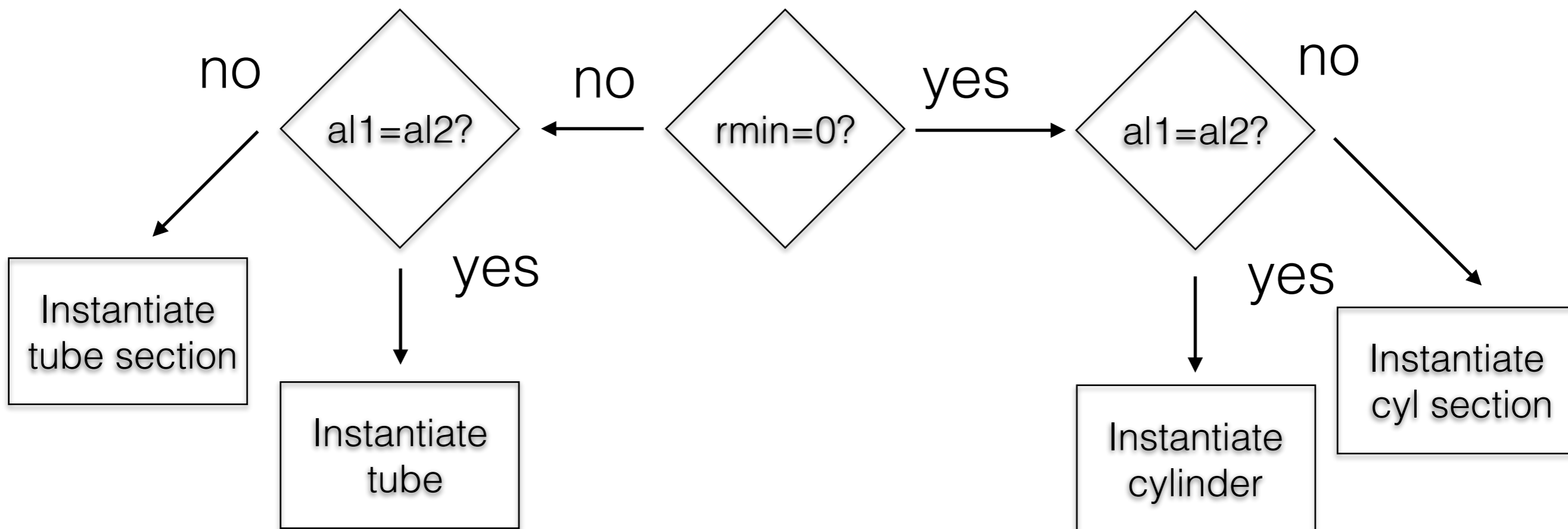
CHEP13 paper: <http://arxiv.org/pdf/1312.0816.pdf>

Portable HPC?

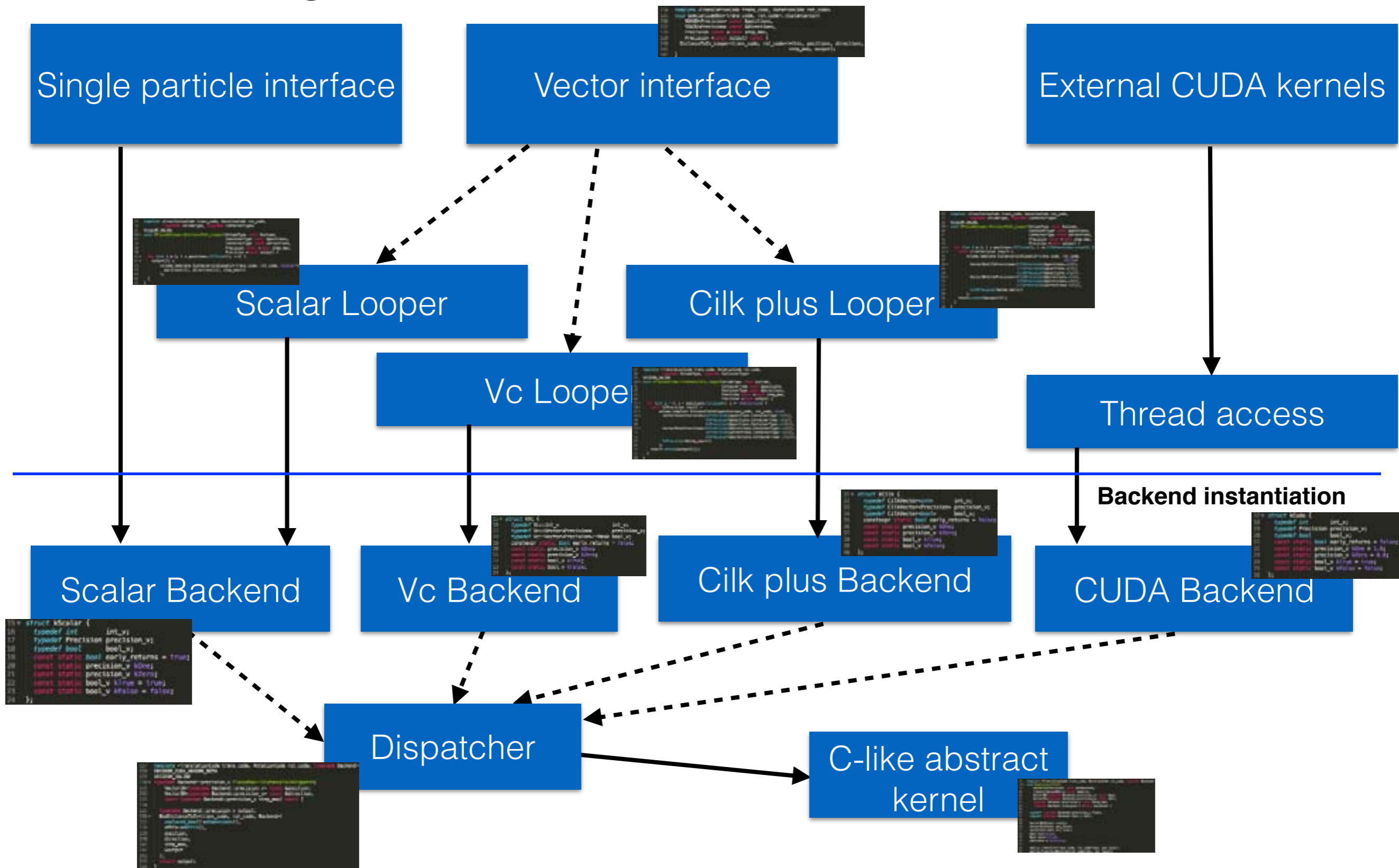
- Straight “vectorisation” of existing code is difficult to impossible
- Resulting code is hard to read and maintain
 - And it is largely compiler-dependency
- Porting to different high end devices is very difficult
- Explored solution is to use template specification for solid placement, specialisation 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  }

```

Single

```

// Scalar Looper code snippet

```

Scalar Looper

```

// Cilk plus Looper code snippet

```

Cilk plus Looper

Vc Looper

```

// Vc Looper code snippet

```

Thread access

Scalar Backend

```

// Scalar Backend code snippet

```

Vc Backend

```

// Vc Backend code snippet

```

Cilk plus Backend

```

// Cilk plus Backend code snippet

```

CUDA Backend

```

// CUDA Backend code snippet

```

Backend instantiation

Dispatcher

```

// Dispatcher code snippet

```

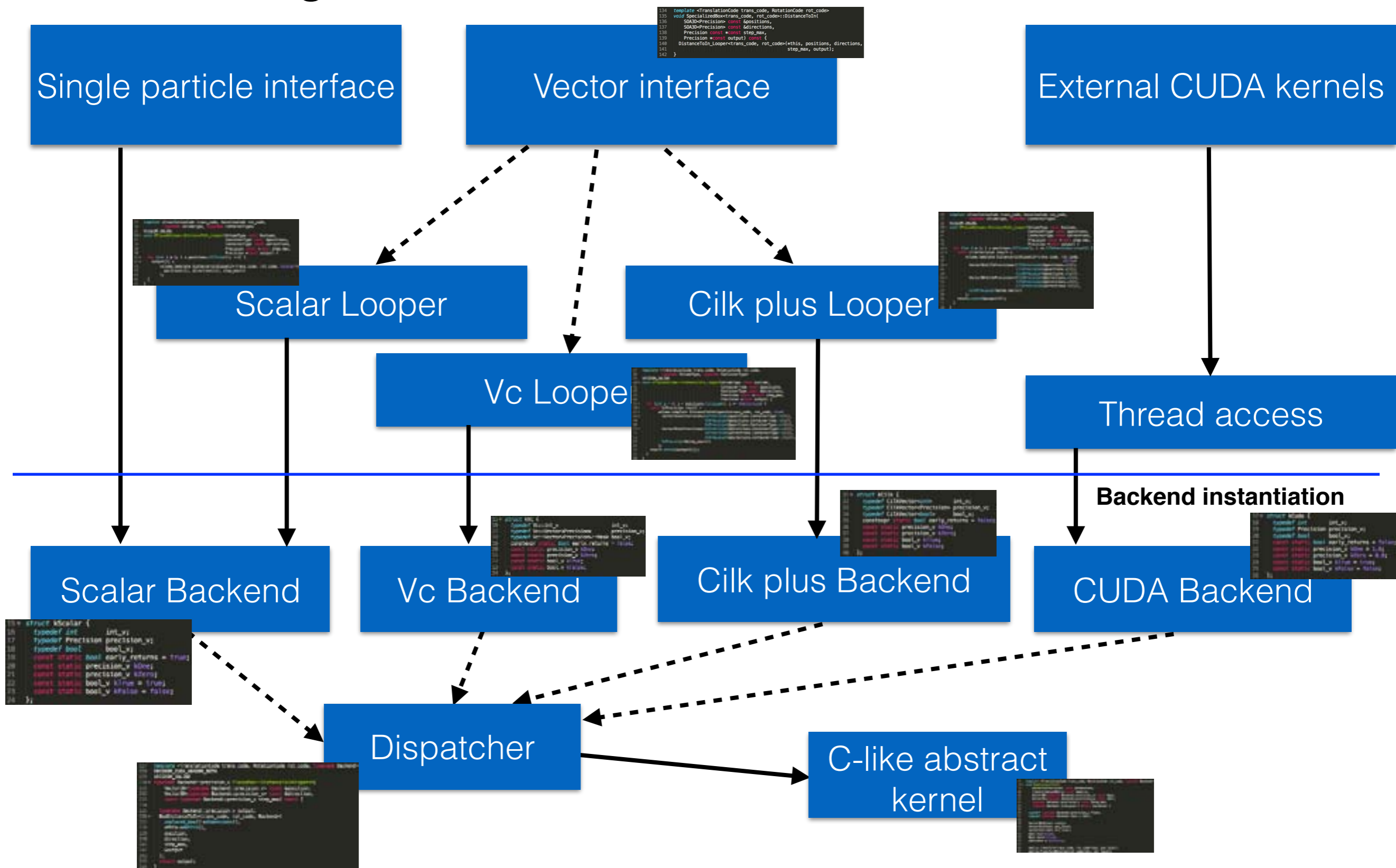
C-like abstract kernel

```

// C-like abstract kernel code snippet

```

Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

```

33  template <TranslationCode trans_code, RotationCode rot_code,
34          typename VolumeType, typename ContainerType>
35  VECGEOM_INLINE
36  void VPlacedVolume::DistanceToIn_Looper(VolumeType const &volume,
37                                         ContainerType const &positions,
38                                         ContainerType const &directions,
39                                         Precision const *const step_max,
40                                         Precision *const output) {
41  for (int i = 0; i < positions.fillsize(); ++i)
42  output[i] =
43  volume.template DistanceToInDispatch<trans_code, rot_code, kScalar>(
44  positions[i], directions[i], step_max[i]);
45  };
46  };
47  }
  
```

Thread access

Backend instantiation

Scalar Backend

Vc Backend

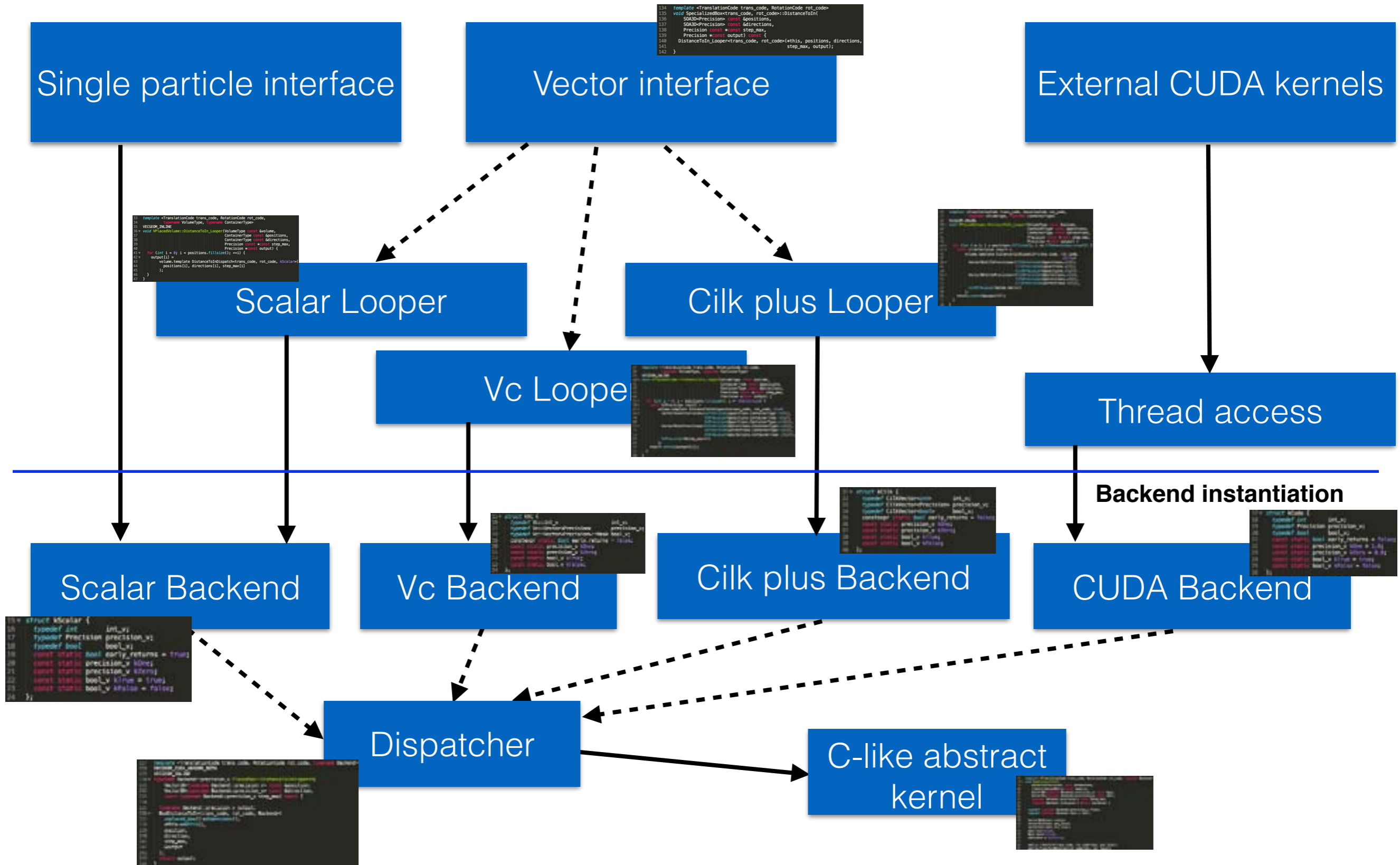
Cilk plus Backend

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

```

33 template <TranslationCode trans_code, R
34         VolumeType, typename
35 VECGEOM_INLINE
36 void VPplacedVolume::DistanceToIn_Looper
37
38
39 for (int i = 0; i < positions.fillsize();
40       output[i] =
41       volume.template DistanceToInDis
42       positions(i), directions(i));
43 }

```

```

154 template <TranslationCode trans_code, RotationCode rot_code>
155 void SpecializedBoxtrans_code_rot_code::DistanceToIn(
156     SQA3D*Precision) const {
157     SQA3D*Precision) const {
158     Precision const *const step_max,
159     Precision *const output) const {
160     DistanceToIn_Looper<trans_code, rot_code>(this, positions, directions,
161     step_max, output);
162 }

```

```

37 template <TranslationCode trans_code, RotationCode rot_code,
38         typename VolumeType, typename ContainerType>
39 VECGEOM_INLINE
40 void VPplacedVolume::DistanceToIn_Looper(VolumeType const &volume,
41     ContainerType const &positions,
42     ContainerType const &directions,
43     Precision const *const step_max,
44     Precision *const output) {
45     for (int i = 0; i < positions.fillsize(); i += kVectorSize) {
46         const VcPrecision result =
47             volume.template DistanceToInDispatch<trans_code, rot_code, kVc>(
48                 Vector3D<VcPrecision>(VcPrecision(&positions.ContainerType::x(i)),
49                                     VcPrecision(&positions.ContainerType::y(i)),
50                                     VcPrecision(&positions.ContainerType::z(i))),
51                 Vector3D<VcPrecision>(VcPrecision(&directions.ContainerType::x(i)),
52                                     VcPrecision(&directions.ContainerType::y(i)),
53                                     VcPrecision(&directions.ContainerType::z(i))),
54                 VcPrecision(step_max[i])
55             );
56         result.store(&output[i]);
57     }
58 }

```

access

instantiation

backend

Scalar Backend

Dispatcher

C-like abstract kernel

```

16 struct Scalar {
17     typedef int int_t;
18     typedef Precision precision_t;
19     typedef bool bool_t;
20     const static bool early_returns = true;
21     const static precision_t kMin;
22     const static precision_t kMax;
23     const static bool_t kTrue;
24     const static bool_t kFalse;

```

```

1000 // ...
1001 // ...
1002 // ...
1003 // ...
1004 // ...
1005 // ...
1006 // ...
1007 // ...
1008 // ...
1009 // ...
1010 // ...

```

```

1000 // ...
1001 // ...
1002 // ...
1003 // ...
1004 // ...
1005 // ...
1006 // ...
1007 // ...
1008 // ...
1009 // ...
1010 // ...

```


Illustrating scalar/SIMD abstraction and kernels

Single particle interface

kernels

```
33 template <TranslationCode trans_code, RotationCode rot_code,
34           typename VolumeType, typename ContainerType>
35 VECGEOM_INLINE
36 void VPlacedVolume::DistanceToIn_Looper(
37     VolumeType const &volume,
38     ContainerType const &positions,
39     ContainerType const &directions,
40     Precision const *const step_max,
41     Precision *const output) {
42     for (int i = 0; i < positions.fillsize(); i++) {
43         output[i] =
44             volume.template DistanceToInDispatch<trans_code, rot_code, kCilk>
45             (positions[i], directions[i], step_max[i]);
46     }
47 }
```

Scalar Loop

```
36 template <TranslationCode trans_code, RotationCode rot_code,
37           typename VolumeType, typename ContainerType>
38 VECGEOM_INLINE
39 void VPlacedVolume::DistanceToIn_Looper(
40     VolumeType const &volume,
41     ContainerType const &positions,
42     ContainerType const &directions,
43     Precision const *const step_max,
44     Precision *const output) {
45     for (int i = 0; i < positions.fillsize(); i += CilkPrecision::size()) {
46         const CilkPrecision result =
47             volume.template DistanceToInDispatch<trans_code, rot_code,
48             kCilk>(
49                 Vector3D<CilkPrecision>(CilkPrecision(&positions.x(i)),
50                                         CilkPrecision(&positions.y(i)),
51                                         CilkPrecision(&positions.z(i))),
52                 Vector3D<CilkPrecision>(CilkPrecision(&directions.x(i)),
53                                         CilkPrecision(&directions.y(i)),
54                                         CilkPrecision(&directions.z(i))),
55                 CilkPrecision(&step_max[i])
56             );
57         output[i];
58     }
59 }
```

Scalar Backend

```
16 struct Scalar {
17     typedef int int_t;
18     typedef Precision precision_t;
19     typedef bool bool_t;
20     const static bool early_returns = true;
21     const static precision_t kMin;
22     const static bool_t kTrue;
23     const static bool_t kFalse;
24 }
```

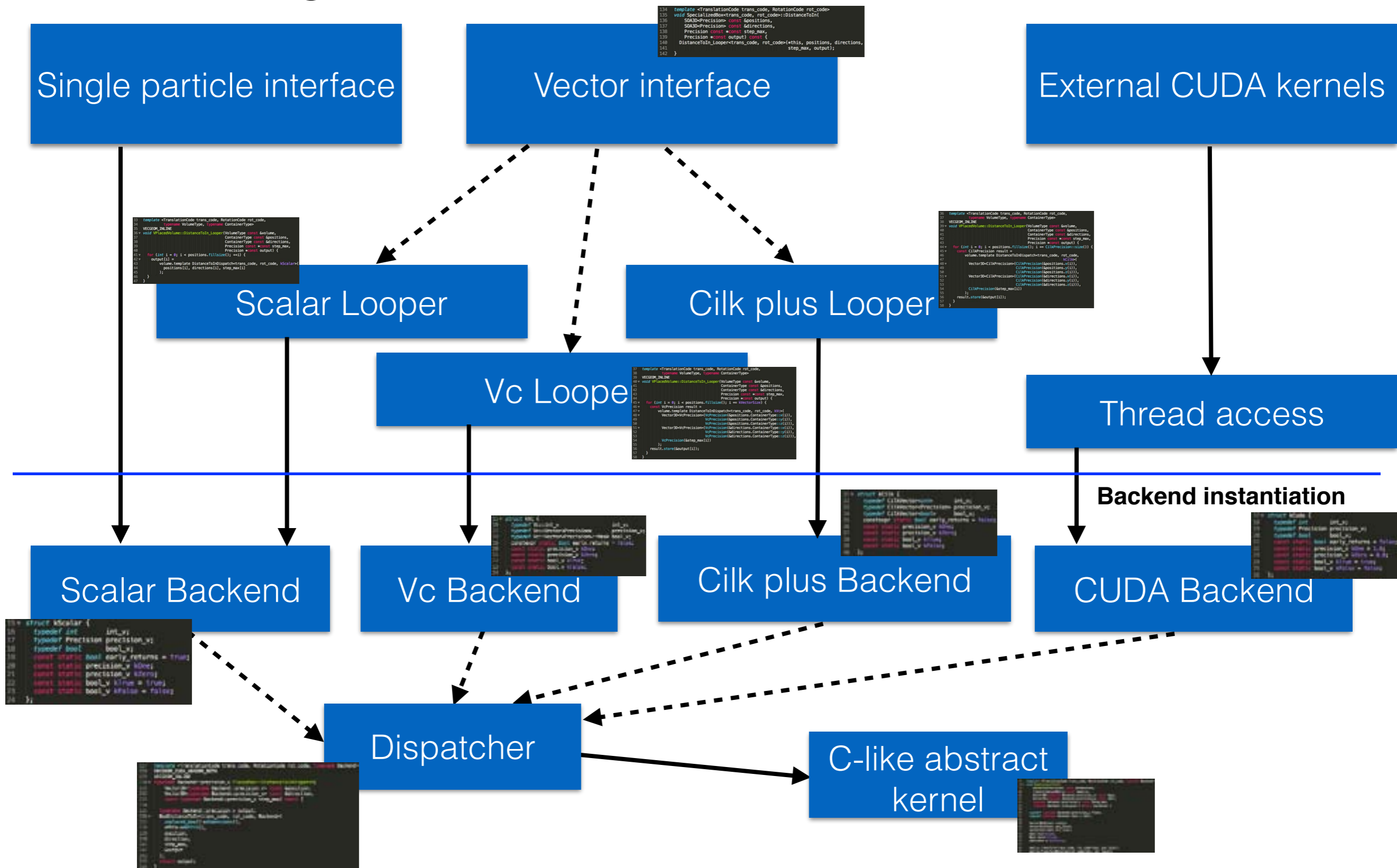
Dispatcher

C-like abstract kernel

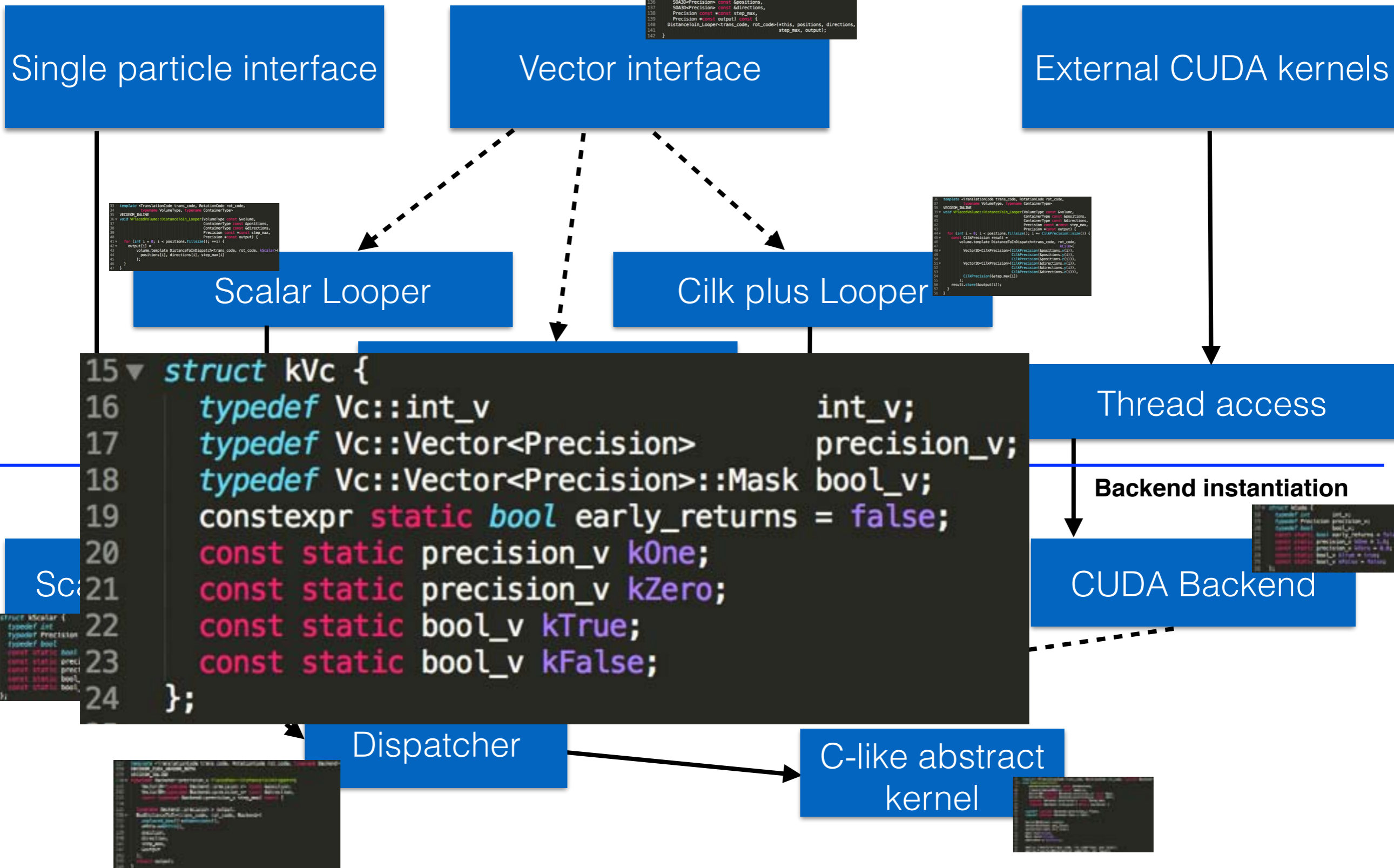
```
1 // ...
2 #include <...>
3 ...
4 ...
5 ...
6 ...
7 ...
8 ...
9 ...
10 ...
11 ...
12 ...
13 ...
14 ...
15 ...
16 ...
17 ...
18 ...
19 ...
20 ...
21 ...
22 ...
23 ...
24 ...
25 ...
26 ...
27 ...
28 ...
29 ...
30 ...
31 ...
32 ...
33 ...
34 ...
35 ...
36 ...
37 ...
38 ...
39 ...
40 ...
41 ...
42 ...
43 ...
44 ...
45 ...
46 ...
47 ...
48 ...
49 ...
50 ...
51 ...
52 ...
53 ...
54 ...
55 ...
56 ...
57 ...
58 ...
59 ...
60 ...
61 ...
62 ...
63 ...
64 ...
65 ...
66 ...
67 ...
68 ...
69 ...
70 ...
71 ...
72 ...
73 ...
74 ...
75 ...
76 ...
77 ...
78 ...
79 ...
80 ...
81 ...
82 ...
83 ...
84 ...
85 ...
86 ...
87 ...
88 ...
89 ...
90 ...
91 ...
92 ...
93 ...
94 ...
95 ...
96 ...
97 ...
98 ...
99 ...
100 ...
```

```
1 // ...
2 #include <...>
3 ...
4 ...
5 ...
6 ...
7 ...
8 ...
9 ...
10 ...
11 ...
12 ...
13 ...
14 ...
15 ...
16 ...
17 ...
18 ...
19 ...
20 ...
21 ...
22 ...
23 ...
24 ...
25 ...
26 ...
27 ...
28 ...
29 ...
30 ...
31 ...
32 ...
33 ...
34 ...
35 ...
36 ...
37 ...
38 ...
39 ...
40 ...
41 ...
42 ...
43 ...
44 ...
45 ...
46 ...
47 ...
48 ...
49 ...
50 ...
51 ...
52 ...
53 ...
54 ...
55 ...
56 ...
57 ...
58 ...
59 ...
60 ...
61 ...
62 ...
63 ...
64 ...
65 ...
66 ...
67 ...
68 ...
69 ...
70 ...
71 ...
72 ...
73 ...
74 ...
75 ...
76 ...
77 ...
78 ...
79 ...
80 ...
81 ...
82 ...
83 ...
84 ...
85 ...
86 ...
87 ...
88 ...
89 ...
90 ...
91 ...
92 ...
93 ...
94 ...
95 ...
96 ...
97 ...
98 ...
99 ...
100 ...
```

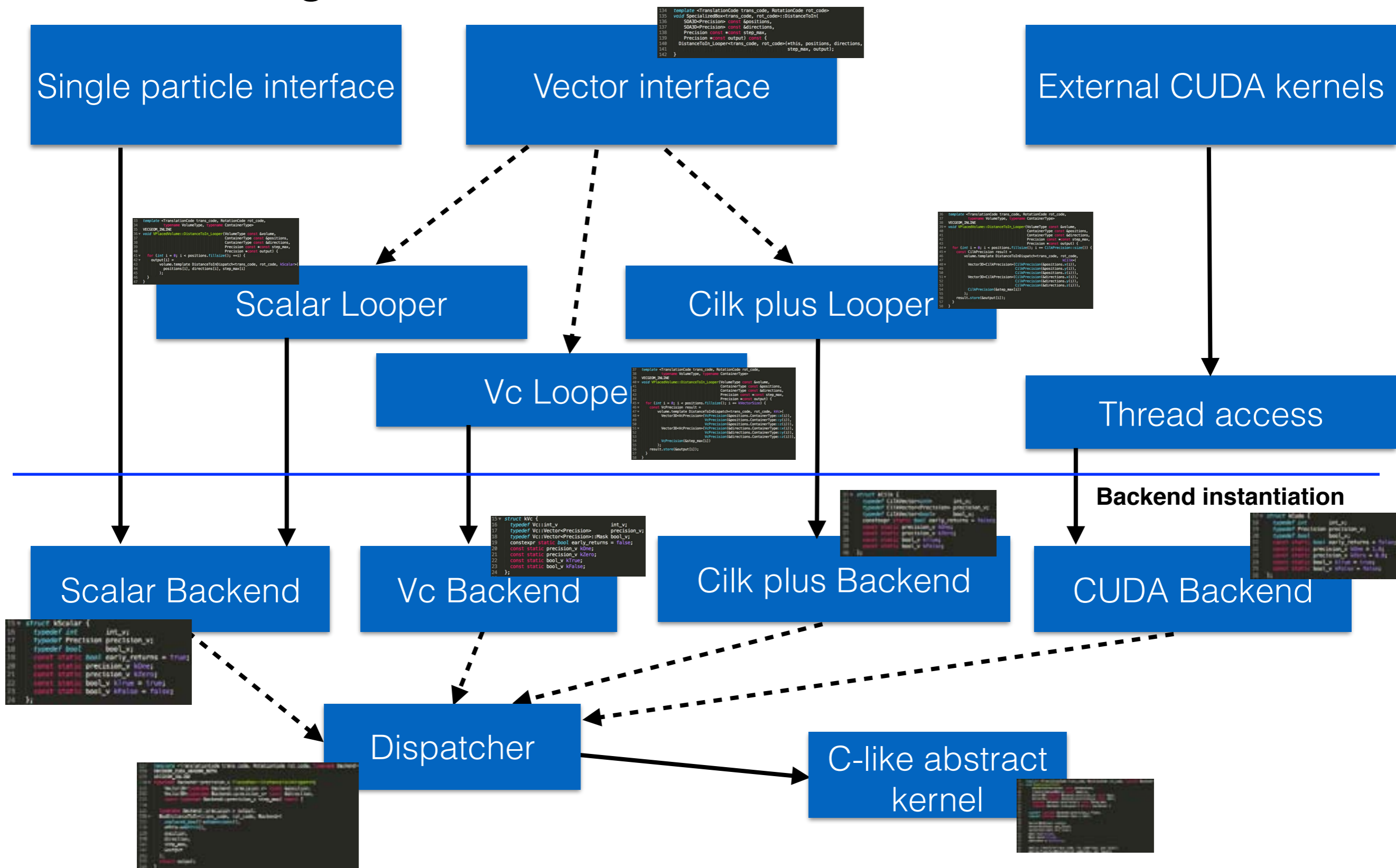
Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

```
13 template <TranslationCode trans_code, RotationCode rot_code,
14           VolumeType, typename ContainerType>
15 VECCOMP_INLINE
16 void WpVolume::DistanceToIn_Loop(VolumeType const &volume,
17                                 ContainerType const &positions,
18                                 ContainerType const &directions,
19                                 Precision const &mask, Precision const &step_max,
20                                 Precision const &output) const {
21     for (int i = 0; i < positions.fillsize(); ++i) {
22         output[i] =
23             volume.template DistanceToDispatch(trans_code, rot_code, kScalar,
24                                               positions[i], directions[i], step_max[i]);
25     }
26 }
27
28 struct kScalar {
29     typedef int int_v;
30     typedef Precision precision_v;
31     typedef bool bool_v;
32     const static bool early_returns = true;
33     const static precision_v kNone;
34     const static precision_v kZero;
35     const static bool_v kTrue = true;
36     const static bool_v kFalse = false;
37 };
38
39 template <TranslationCode trans_code, RotationCode rot_code,
40           VolumeType, typename ContainerType>
41 VECCOMP_INLINE
42 void WpVolume::DistanceToIn_Loop(VolumeType const &volume,
43                                 ContainerType const &positions,
44                                 ContainerType const &directions,
45                                 Precision const &mask, Precision const &step_max,
46                                 Precision const &output) const {
47     for (int i = 0; i < positions.fillsize(); ++i) {
48         const WpPrecision result =
49             volume.template DistanceToDispatch(trans_code, rot_code, kNone,
50                                               positions[i], directions[i], step_max[i]);
51         WpPrecision::store(output[i], result);
52     }
53 }
54
55 struct kNone {
56     typedef int_v int_v;
57     typedef Precision precision_v;
58     typedef bool_v bool_v;
59     const static bool early_returns = false;
60     const static precision_v kNone;
61     const static precision_v kZero;
62     const static bool_v kTrue;
63     const static bool_v kFalse;
64 };
65
66 template <TranslationCode trans_code, RotationCode rot_code,
67           VolumeType, typename ContainerType>
68 VECCOMP_INLINE
69 void WpVolume::DistanceToIn_Loop(VolumeType const &volume,
70                                 ContainerType const &positions,
71                                 ContainerType const &directions,
72                                 Precision const &mask, Precision const &step_max,
73                                 Precision const &output) const {
74     for (int i = 0; i < positions.fillsize(); ++i) {
75         const WpPrecision result =
76             volume.template DistanceToDispatch(trans_code, rot_code, kTrue,
77                                               positions[i], directions[i], step_max[i]);
78         WpPrecision::store(output[i], result);
79     }
80 }
81
82 template <TranslationCode trans_code, RotationCode rot_code,
83           VolumeType, typename ContainerType>
84 VECCOMP_INLINE
85 void WpVolume::DistanceToIn_Loop(VolumeType const &volume,
86                                 ContainerType const &positions,
87                                 ContainerType const &directions,
88                                 Precision const &mask, Precision const &step_max,
89                                 Precision const &output) const {
90     for (int i = 0; i < positions.fillsize(); ++i) {
91         const WpPrecision result =
92             volume.template DistanceToDispatch(trans_code, rot_code, kFalse,
93                                               positions[i], directions[i], step_max[i]);
94         WpPrecision::store(output[i], result);
95     }
96 }
```

Scalar

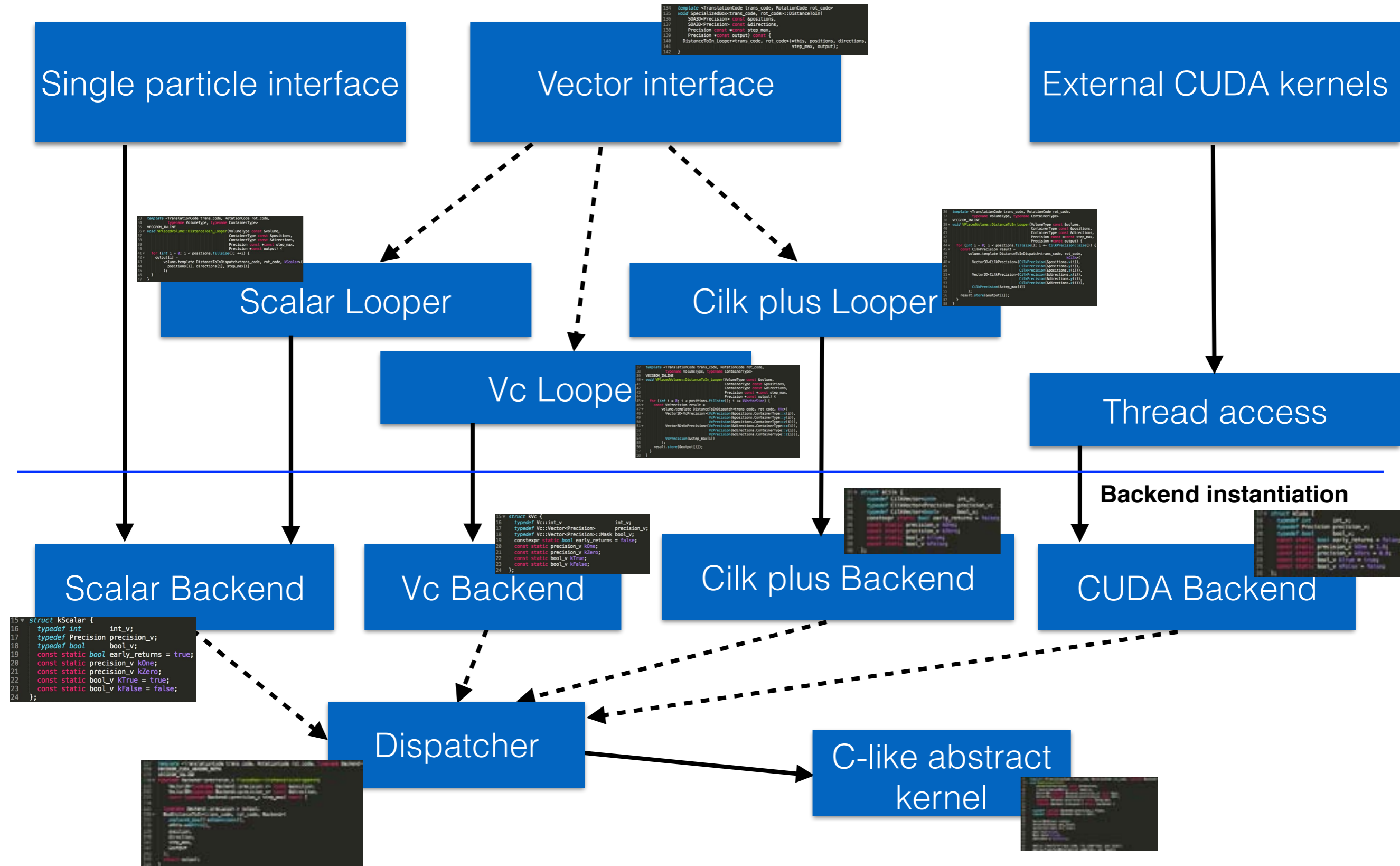
Dispatcher

C-like abstract kernel

```
15 struct kScalar {
16     typedef int int_v;
17     typedef Precision precision_v;
18     typedef bool bool_v;
19     const static bool early_returns = true;
20     const static precision_v kNone;
21     const static precision_v kZero;
22     const static bool_v kTrue = true;
23     const static bool_v kFalse = false;
24 };
```

```
15 struct kNone {
16     typedef int_v int_v;
17     typedef Precision precision_v;
18     typedef bool_v bool_v;
19     const static bool early_returns = false;
20     const static precision_v kNone;
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

```

13 template <TranslationCode trans_code, RotationCode rot_code,
14           VolumeType, typename ContainerType>
15 VECCOMP_INLINE
16 void WPlacedVolume::DistanceToIn_Loop(VolumeType const &volume,
17                                       ContainerType const &positions,
18                                       ContainerType const &directions,
19                                       Precision const &mask, Precision const &step_max,
20                                       Precision const &output) const {
21     for (int i = 0; i < positions.fillsize(); ++i) {
22         output[i] =
23             volume.template DistanceToDispatch(trans_code, rot_code, kScalar,
24                                               positions[i], directions[i], step_max[i]);
25     }
26 }
    
```

```

154 template <TranslationCode trans_code, RotationCode rot_code>
155 void SpecialLineBox::trans_code, rot_code::DistanceToIn
156 SQA3D::Precision> const &positions,
157 SQA3D::Precision> const &directions,
158 Precision const &mask, Precision const &step_max,
159 Precision const &output) const {
160     DistanceToIn_Loop(trans_code, rot_code)(this, positions, directions,
161                                             step_max, output);
162 }
    
```

```

10 template <TranslationCode trans_code, RotationCode rot_code,
11           VolumeType, typename ContainerType>
12 VECCOMP_INLINE
13 void WPlacedVolume::DistanceToIn_Loop(VolumeType const &volume,
14                                       ContainerType const &positions,
15                                       ContainerType const &directions,
16                                       Precision const &mask, Precision const &step_max,
17                                       Precision const &output) const {
18     for (int i = 0; i < positions.fillsize(); ++i) {
19         output[i] =
20             volume.template DistanceToDispatch(trans_code, rot_code,
21                                               CilkPrecision::kScalar,
22                                               positions[i], directions[i], step_max[i]);
23     }
24 }
    
```

```

31 struct kCilk {
32     typedef CilkVector<int> int_v;
33     typedef CilkVec<Precision> precision_v;
34     typedef CilkVec<bool> bool_v;
35     constexpr static bool early_returns = false;
36     const static precision_v kOne;
37     const static precision_v kZero;
38     const static bool_v kTrue;
39     const static bool_v kFalse;
40 };
    
```

Scalar Backend

Backend access

Backend instantiation

Backend

Dispatcher

C-like abstract kernel

```

15 struct kScalar {
16     typedef int int_v;
17     typedef Precision precision_v;
18     typedef bool bool_v;
19     constexpr static bool early_returns = true;
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 };
    
```

```

15 struct kCilk {
16     typedef CilkVector<int> int_v;
17     typedef CilkVec<Precision> precision_v;
18     typedef CilkVec<bool> 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

Single particle interface

Vector interface

External CUDA kernels

```

154 template <TranslationCode trans_code, RotationCode rot_code>
155 void SpecialLineBoxTrans_code_rot_code::DistanceToIn(
156     SQA3D*Precision> const& positions,
157     SQA3D*Precision> const& directions,
158     Precision const& step_max,
159     Precision const& output) const {
160     DistanceToIn_Loopier<trans_code, rot_code>(this, positions, directions,
161     step_max, output);
162 }
    
```

```

33 template <TranslationCode trans_code, RotationCode rot_code,
34           VolumeType, typename ContainerType>
35 VECEM_INLINE
36 void WFaceVolume::DistanceToIn_Loopier(VolumeType const& volume,
37 ContainerType const& positions,
38 ContainerType const& directions,
39 Precision const& step_max,
40 Precision const& output) {
41     for (int i = 0; i < positions.fillsize(); ++i) {
42         output[i] =
43             volume.template DistanceToInDispatch<trans_code, rot_code, kScalar>
44                 (positions[i], directions[i], step_max[i]);
45     }
46 }
    
```

Scalar

```

17 struct kCuda {
18     typedef int int_v;
19     typedef Precision precision;
20     typedef bool bool_v;
21     const static bool early_returns = false;
22     const static precision_v kOne = 1.0;
23     const static precision_v kZero = 0.0;
24     const static bool_v kTrue = true;
25     const static bool_v kFalse = false;
26 };
    
```

```

36 template <TranslationCode trans_code, RotationCode rot_code,
37           VolumeType, typename ContainerType>
38 VECEM_INLINE
39 void WFaceVolume::DistanceToIn_CUDA_Loopier(VolumeType const& volume,
40 ContainerType const& positions,
41 ContainerType const& directions,
42 Precision const& step_max,
43 Precision const& output) {
44     for (int i = 0; i < positions.fillsize(); ++i) {
45         CLIPrecision result =
46             volume.template DistanceToInDispatch<trans_code, rot_code,
47             kCuda>
48                 (positions[i], directions[i], step_max[i]);
49         CLIPrecision result =
50             volume.template DistanceToInDispatch<trans_code, rot_code,
51             kCuda>
52                 (positions[i], directions[i], step_max[i]);
53         CLIPrecision result =
54             volume.template DistanceToInDispatch<trans_code, rot_code,
55             kCuda>
56                 (positions[i], directions[i], step_max[i]);
57         result.store(output[i]);
58     }
    
```

and access

and instantiation

Scalar Backend

Vc Backend

CLIK plus Backend

CUDA Backend

```

15 struct kScalar {
16     typedef int int_v;
17     typedef Precision precision_v;
18     typedef bool bool_v;
19     const static bool early_returns = true;
20     const static precision_v kOne;
21     const static precision_v kZero;
22     const static bool_v kTrue;
23     const static bool_v kFalse = false;
24 };
    
```

Dispatcher

C-like abstract kernel

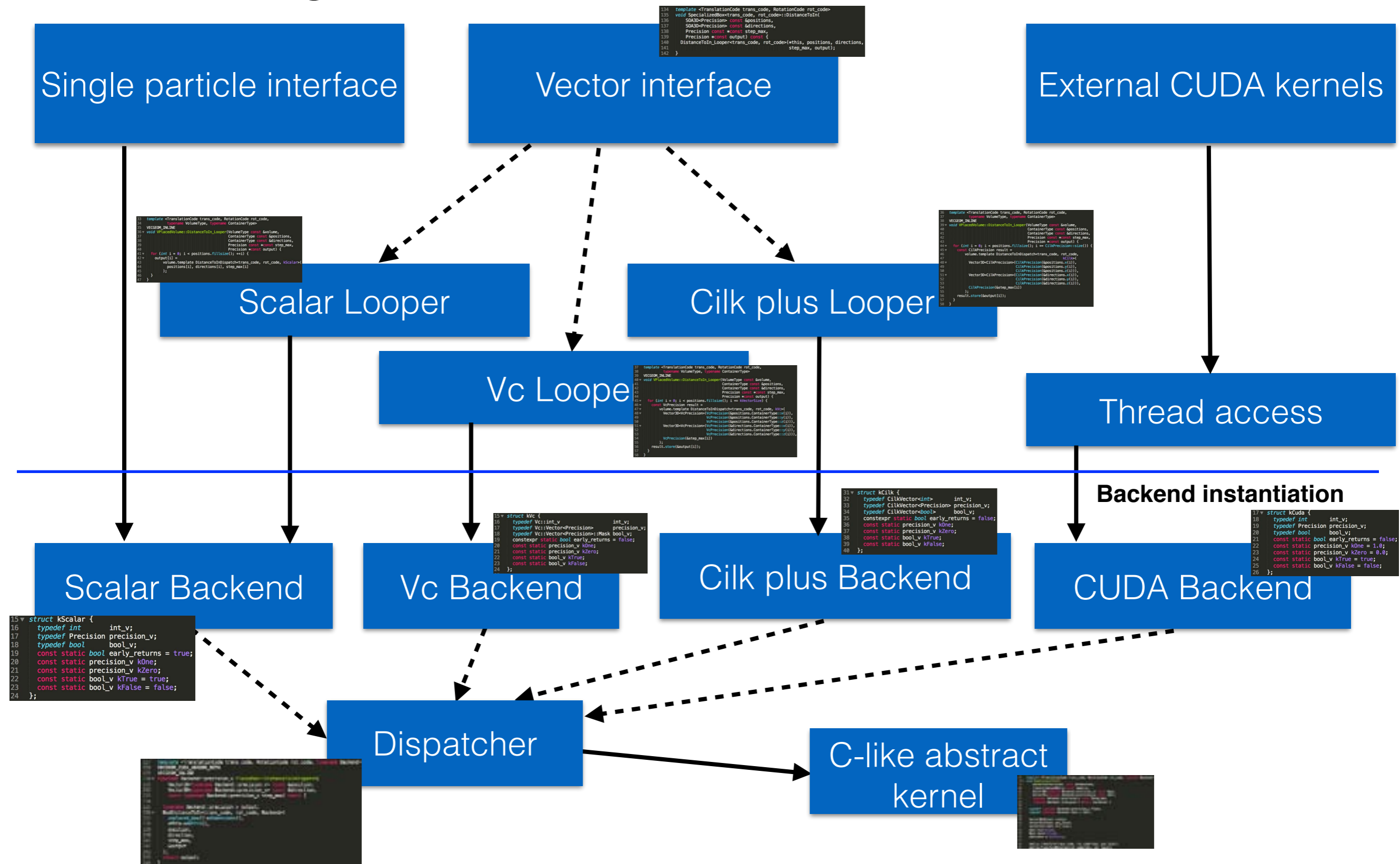
```

...
...
...
...
...
    
```

```

...
...
...
...
...
    
```

Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

```

134 template <TranslationCode trans_code, RotationCode rot_code>
135 void SpecializedBox<trans_code, rot_code>::DistanceToIn(
136     SQA3D*Precision> const &positions,
137     SQA3D*Precision> const &directions,
138     Precision const &const_step_max,
139     Precision &const_output) const {
140     DistanceToIn_Loop<trans_code, rot_code>(this, positions, directions,
141     step_max, output);
142 }
    
```

```

227 template <TranslationCode trans_code, RotationCode rot_code, typename Backend>
228 VECGEOM_CUDA_HEADER_BOTH
229 VECGEOM_INLINE
230 ▼ typename Backend::precision_v PlacedBox::DistanceToInDispatch(
231     Vector3D<typename Backend::precision_v> const &position,
232     Vector3D<typename Backend::precision_v> const &direction,
233     const typename Backend::precision_v step_max) const {
234
235     typename Backend::precision_v output;
236 ▼ BoxDistanceToIn<trans_code, rot_code, Backend>(
237     unplaced_box()->dimensions(),
238     *this->matrix(),
239     position,
240     direction,
241     step_max,
242     &output
243 );
244     return output;
245 }
    
```

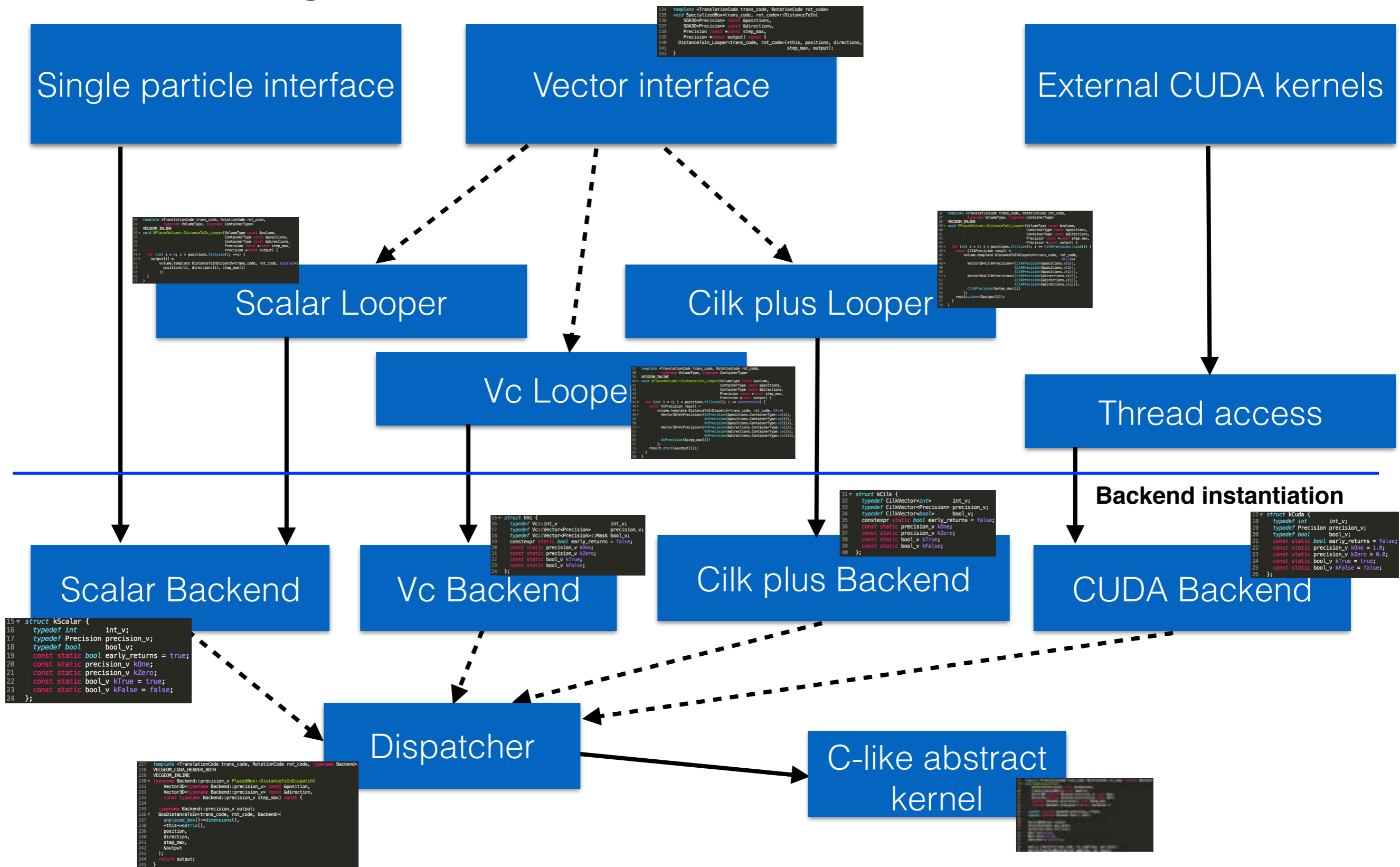
Dispatcher

C-like abstract kernel

```

false;
.0;
0.0;
e;
    
```

Illustrating scalar/SIMD abstraction and kernels



Illustrating scalar/SIMD abstraction and kernels

Single particle interface

Vector interface

External CUDA kernels

```

13 template <TranslationCode trans_code, RotationCode rot_code,
14           typename VolumeType, typename ContainerType>
15 VECGEOM_INLINE
16 void WPlacedBox::DistanceToIn_Loop(VolumeType const &volume,
17 ContainerType const &positions,
18 ContainerType const &directions,
19 Precision const &step_max,
20 Precision const &output) {
21   for (int i = 0; i < positions.fillsize(); ++i) {
22     output[i] =
23       volume.template DistanceToInDispatch(trans_code, rot_code, kScalar);
24   }
25 }

```

Scalar

```

77 template <TranslationCode trans_code, RotationCode rot_code,
78           typename VolumeType, typename ContainerType>
79 void BoxDistanceToIn(
80   Vector3D<Precision> const &dimensions,
81   TransformationMatrix const &matrix,
82   Vector3D<typename Backend::precision_v> const &pos,
83   Vector3D<typename Backend::precision_v> const &dir,
84   typename Backend::precision_v const &step_max,
85   typename Backend::precision_v const &output) {
86   typedef typename Backend::precision_v precision_v;
87   typedef typename Backend::precision_v kPrecision;
88   typedef typename Backend::precision_v kOutput;
89   Vector3D<Float> safety;
90   Vector3D<Float> pos_local;
91   Vector3D<Float> dir_local;
92   Bool hit(false);
93   Bool done(false);
94   *distance = kinfinity;
95   matrix.Transform<trans_code, rot_code>(pos, pos_local);
96   matrix.TransformRotation<rot_code>(dir, dir_local);

```

```

10 template <TranslationCode trans_code, RotationCode rot_code,
11           typename VolumeType, typename ContainerType>
12 VECGEOM_INLINE
13 void WPlacedBox::DistanceToIn_CUDA(VolumeType const &volume,
14 ContainerType const &positions,
15 ContainerType const &directions,
16 Precision const &step_max,
17 Precision const &output) {
18   for (int i = 0; i < positions.fillsize(); ++i) {
19     output[i] =
20       volume.template DistanceToInDispatch(trans_code, rot_code, kCuda);
21   }
22 }

```

Thread access

Backend instantiation

```

17 struct Kcuda {
18   typedef int int_v;
19   typedef Precision precision_v;
20   typedef bool bool_v;
21   const static bool early_returns = false;
22   const static precision_v kOne = 1.0;
23   const static precision_v kZero = 0.0;
24   const static bool_v kTrue = true;
25   const static bool_v kFalse = false;
26 };

```

Scalar Backend

CUDA Backend

```

15 struct kScalar {
16   typedef int int_v;
17   typedef Precision precision_v;
18   typedef bool bool_v;
19   const static bool early_returns = true;
20   const static precision_v kOne;
21   const static precision_v kZero;
22   const static bool_v kTrue = true;
23   const static bool_v kFalse = false;
24 };

```

Dispatcher

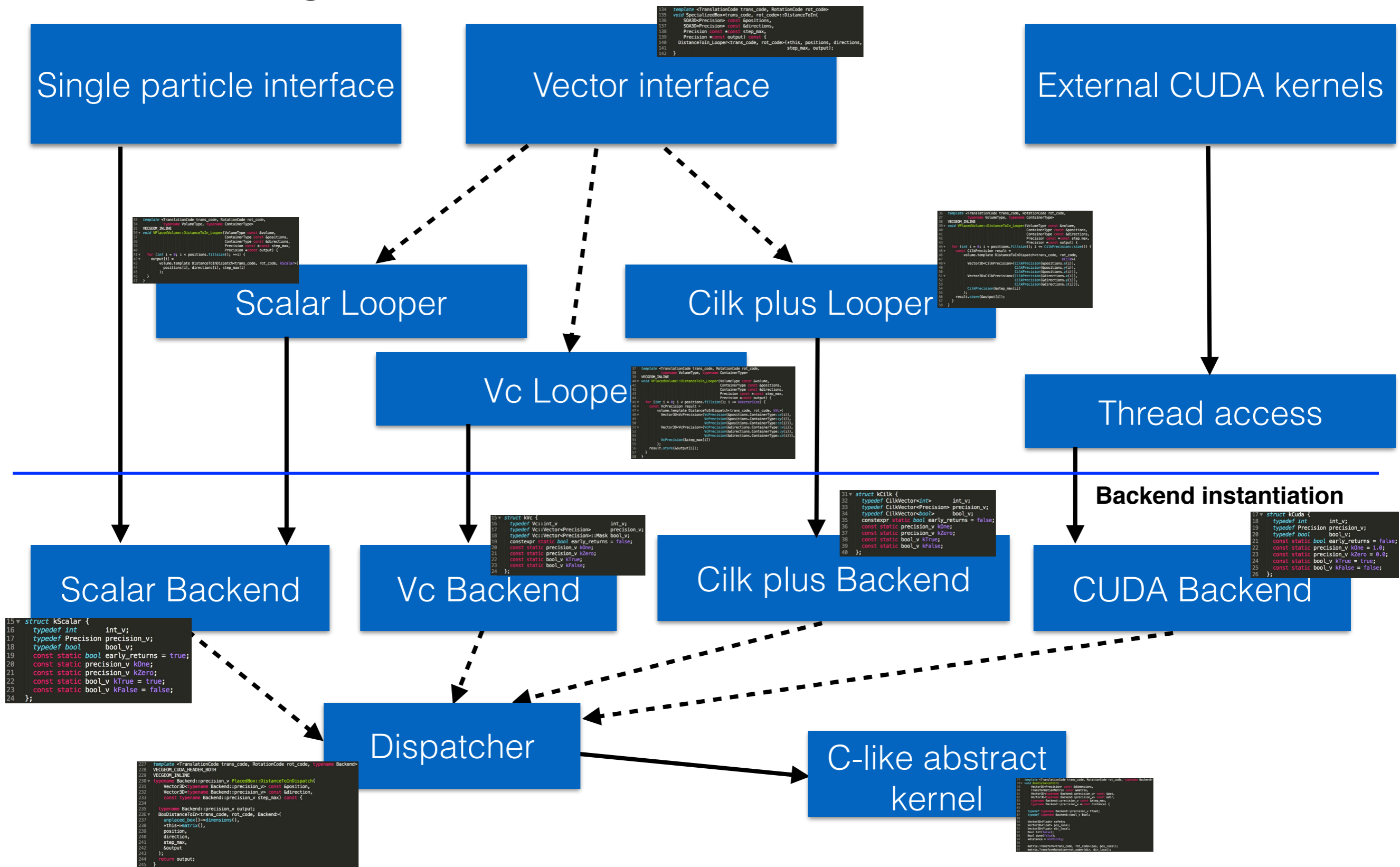
C-like abstract kernel

```

227 template <TranslationCode trans_code, RotationCode rot_code, typename Backend>
228 VECGEOM_CUDA_HEADER_BOTH
229 VECGEOM_INLINE
230 void WPlacedBox::DistanceToInDispatch(
231   Vector3D<typename Backend::precision_v> const &position,
232   Vector3D<typename Backend::precision_v> const &direction,
233   const typename Backend::precision_v step_max) const {
234   typename Backend::precision_v output;
235   BoxDistanceToIn(trans_code, rot_code, Backend)(
236     unplaced_box()-->dimensions(),
237     *this->matrix(),
238     position,
239     direction,
240     step_max,
241     &output);
242   return output;
243 }

```


Illustrating scalar/SIMD abstraction and kernels

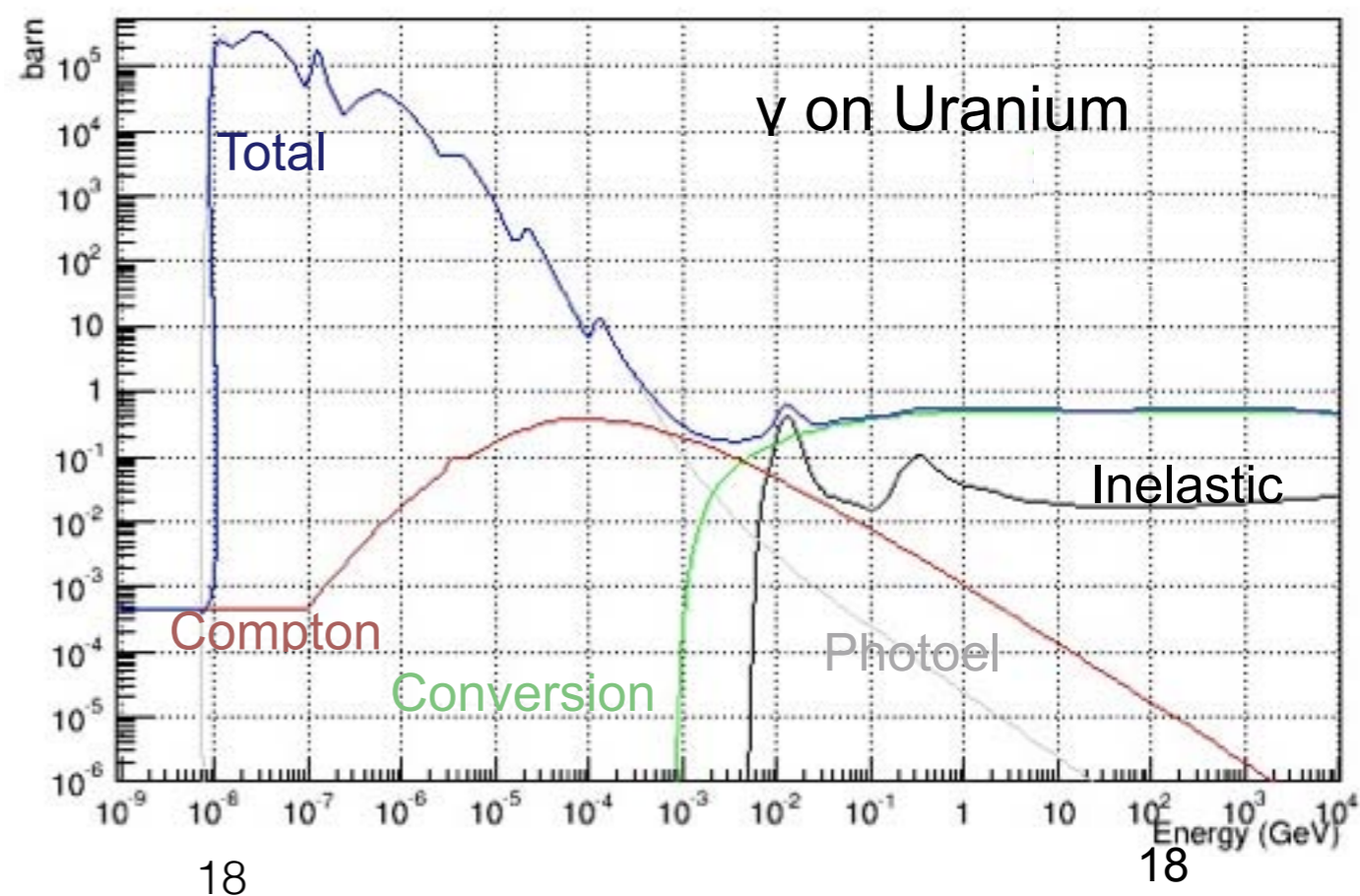
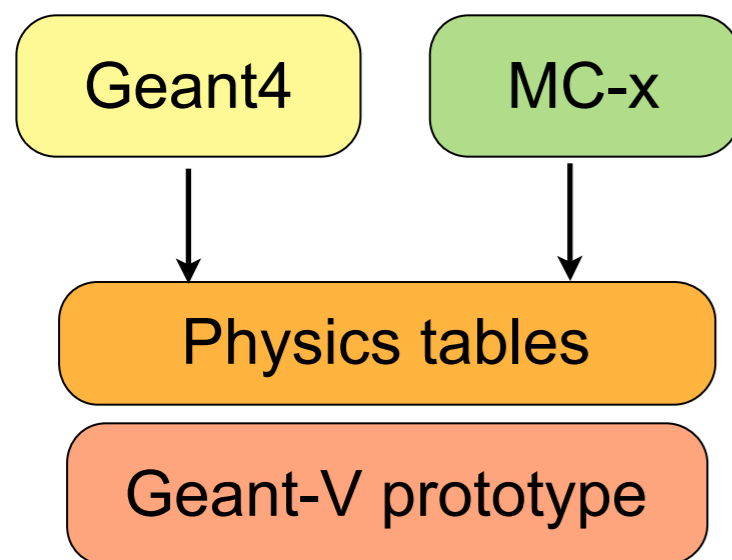


USolid

- 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 \rightarrow 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



Where are we now?

- Scheduler

- The new version, hopefully improved of the scheduler has been committed and we are testing it

- Solids

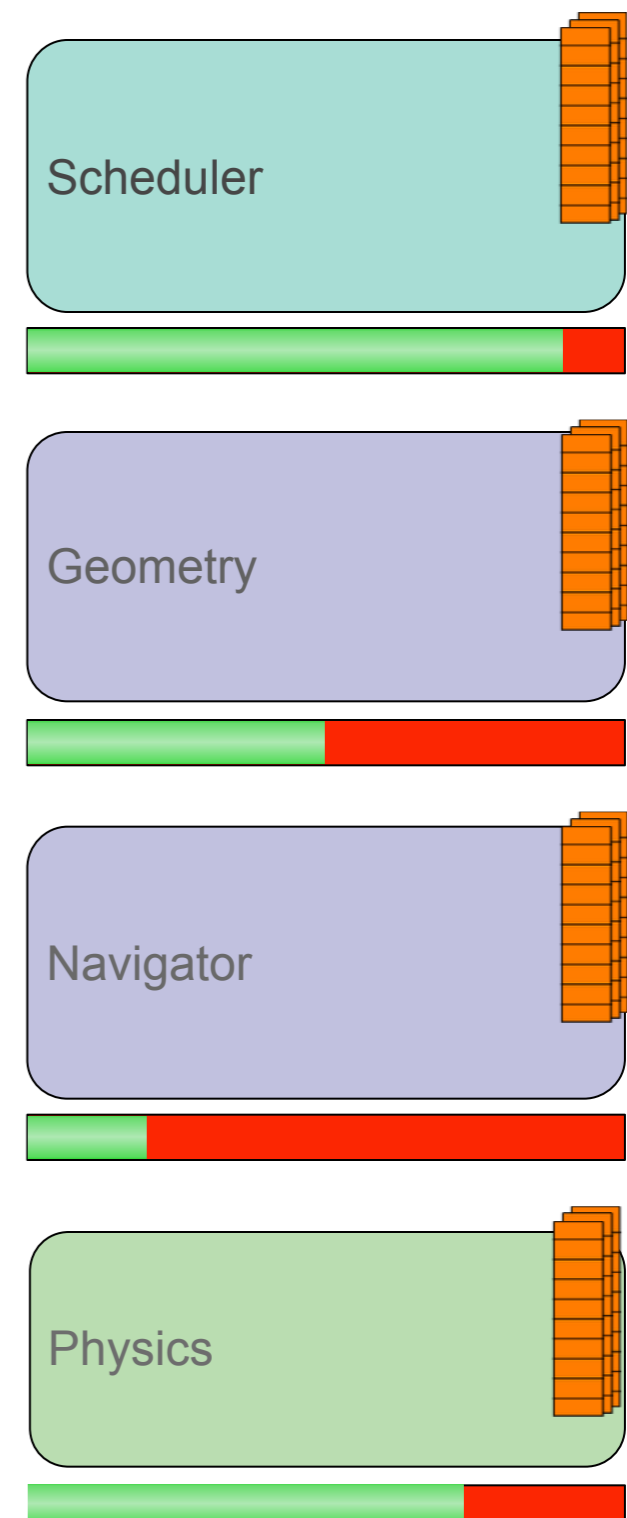
- The proof or principle that we can achieve large speedups (3-5+) is there, however a lot of work lays ahead

- 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

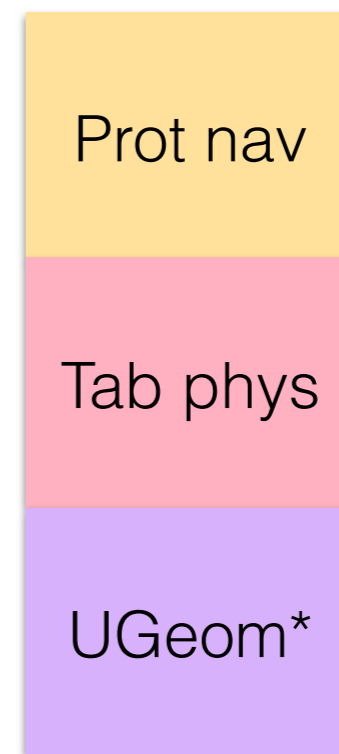
- April 11th
 - Simple (EX03-like) benchmark
 - Connect tabulated physics with Geant4 & prototype
 - Port optimised Bremsstrahlung to prototype & reuse in Geant4
 - USolid and UGeom to run Ex03 in prototype
- July 31st
 - *Move to Geant4 10.0*
 - *Nightly build system*
 - *Magnetic Field transport*
 - 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



Prototype



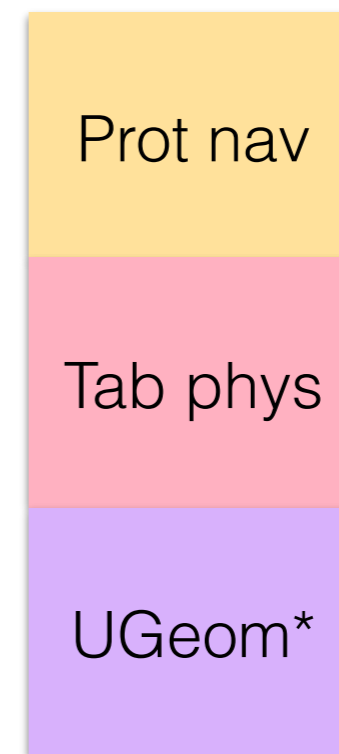
With small, large & venti geometries

Testing - benchmarking

Geant4



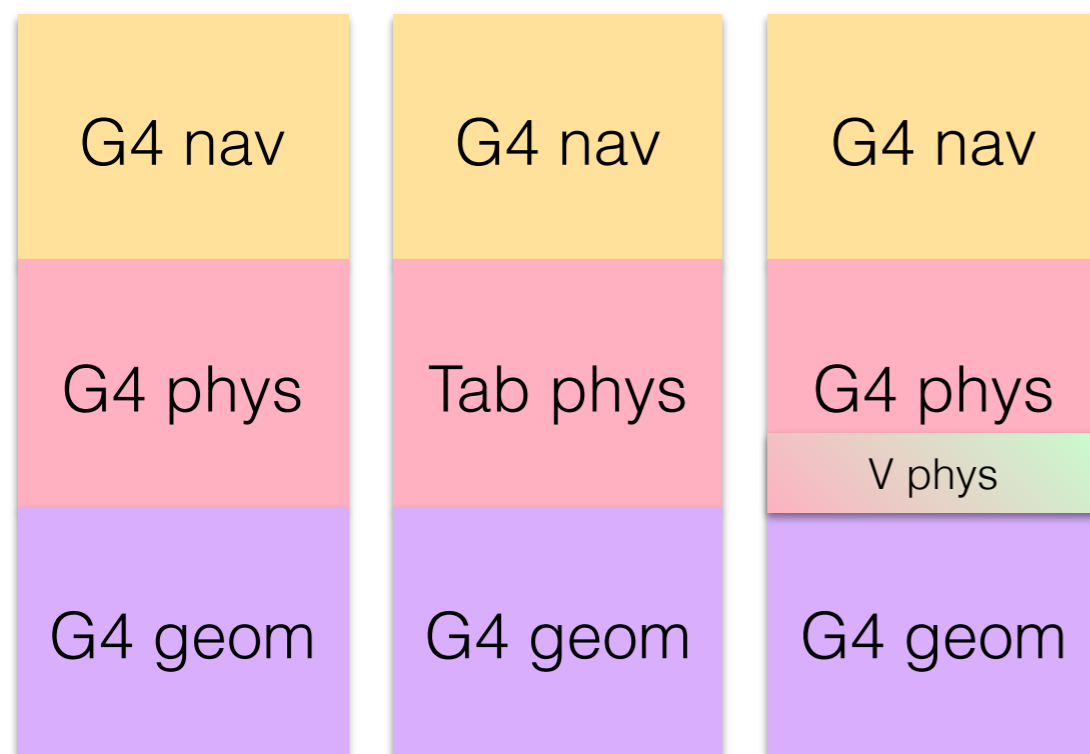
Prototype



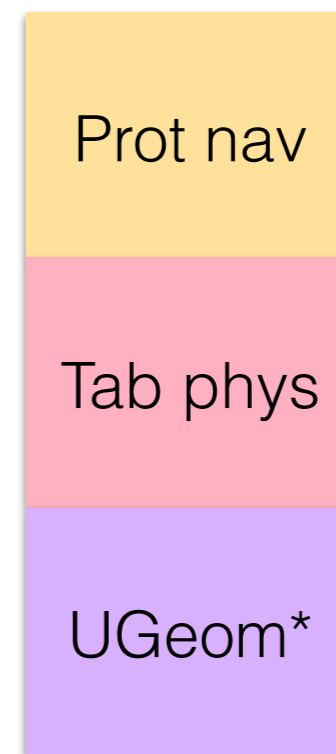
With small, large & venti geometries

Testing - benchmarking

Geant4



Prototype

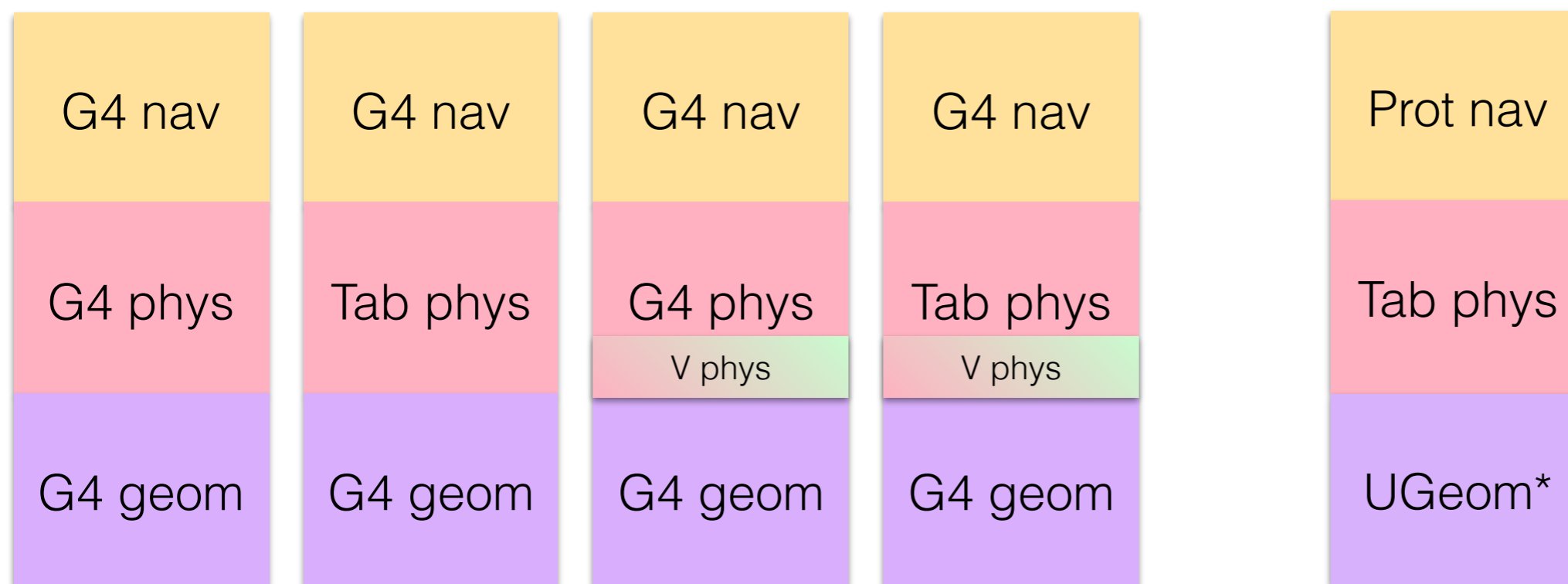


With small, large & venti geometries

Testing - benchmarking

Geant4

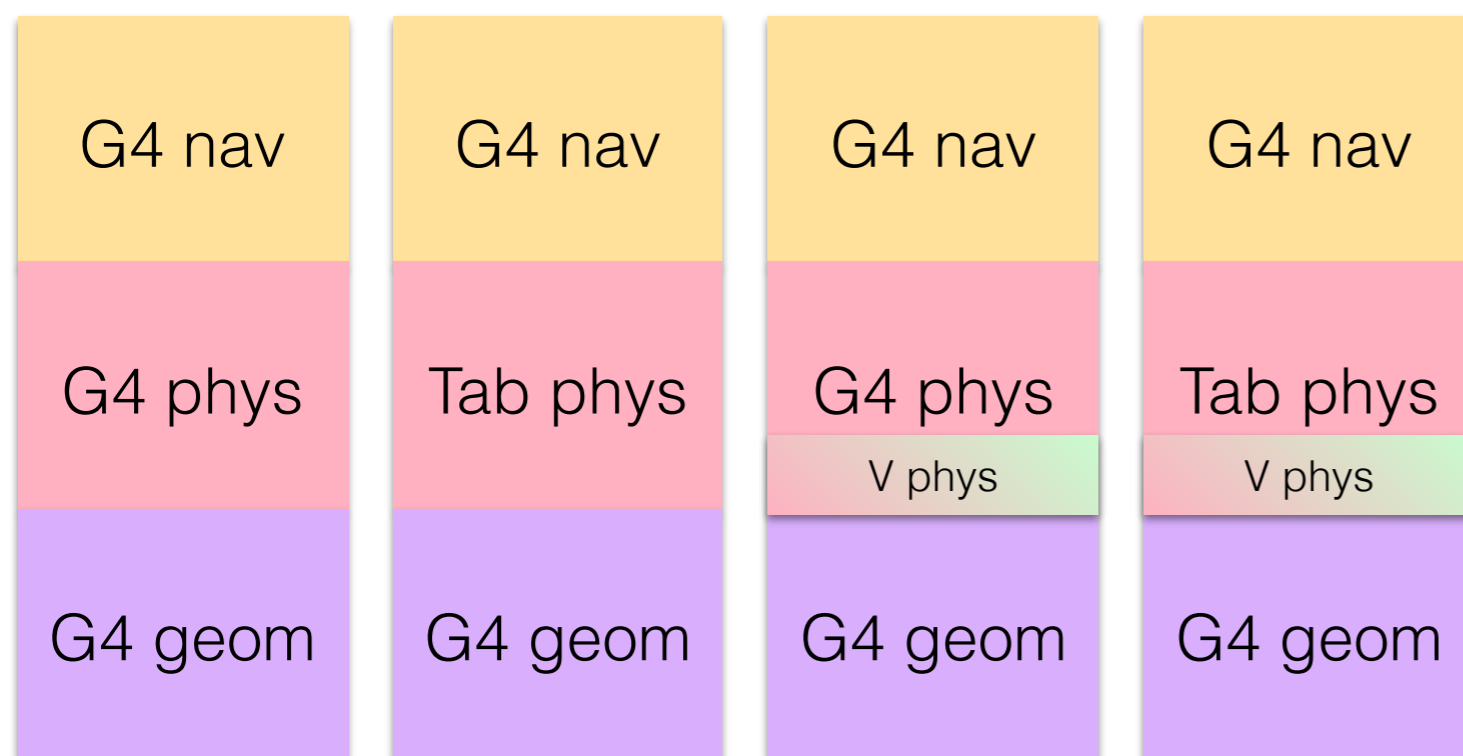
Prototype



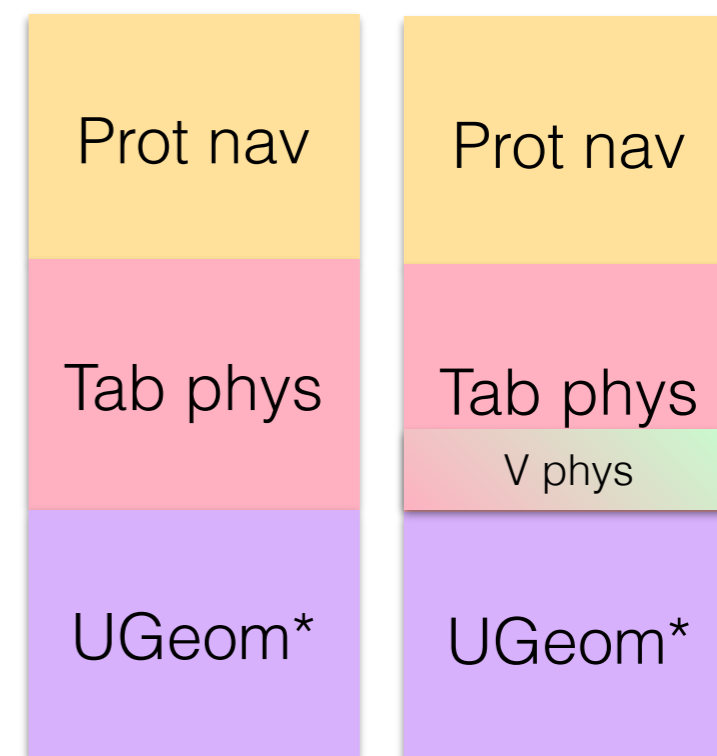
With small, large & venti geometries

Testing - benchmarking

Geant4



Prototype



With small, large & venti geometries

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