

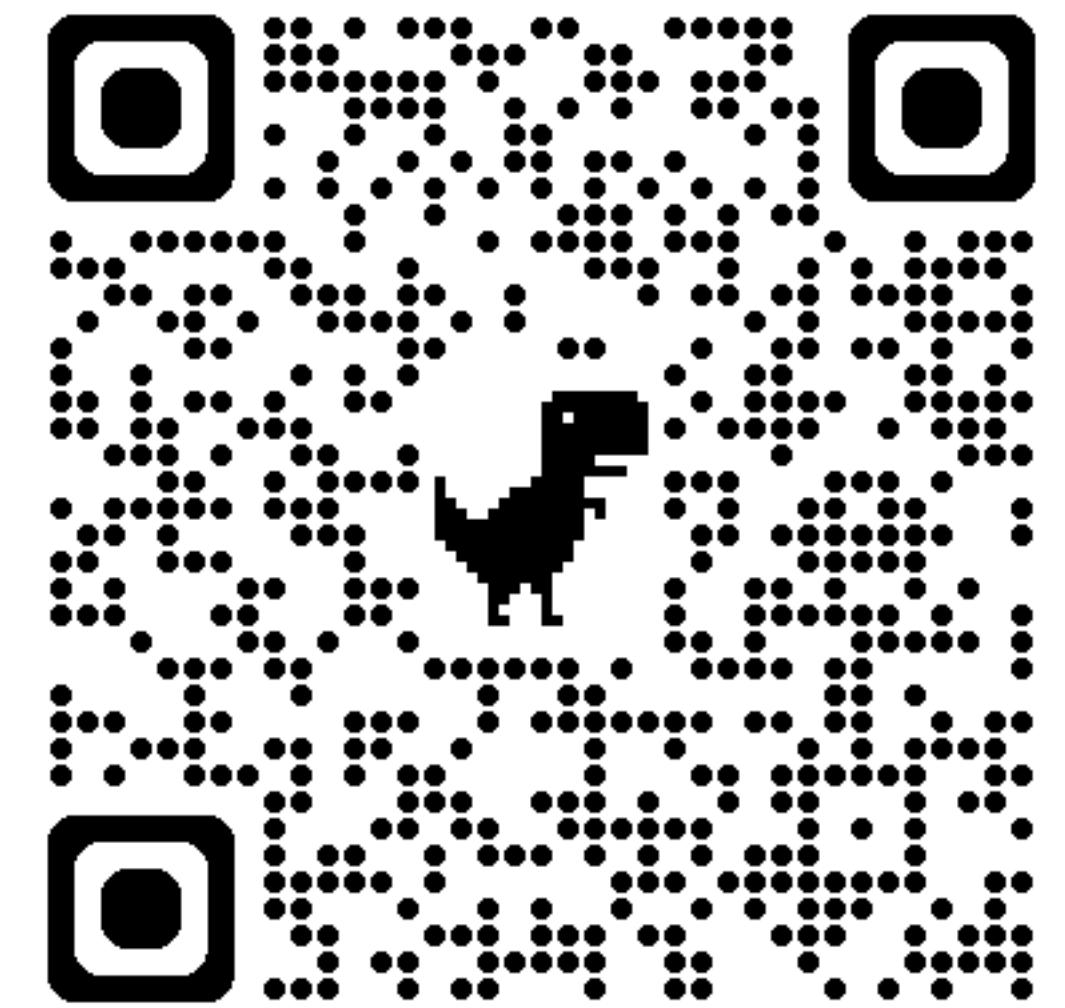
Java Enable Preview: Vector API

Daniel Hinojosa

Using SIMD Processing in the JVM

In this Presentation

- Vector API
- SIMD
- Species
- Lanes & Components
- Masking
- Lane-wise Operations
- Cross-lane Operations



Slides and Material:

https://github.com/dhinojosa/java_enable_vector_api

Vector API



Vector API

- Introduce an API to express vector computations that reliably compile at runtime **to vector instructions** on supported CPU architectures.
- The performance from the Vector API is superior to equivalent scalar computations.
- A part of **Project Panama**

SIMD

- Single Instruction/Multiple Data
- Such machines exploit data level parallelism, but not concurrency
- Not all machines offer the ability to process SIMD processes
- A binary operation applied to two vectors with the same number of lanes would, for each lane, apply the equivalent scalar operation on the corresponding two scalar values from each vector
- This is not new, and have been used with supercomputers in the 60s and 70s



What Modern Machinery has SIMD?

- All x86 processors with AVX Based Instruction Sets
- ARM processor with Neon Instruction Set
- Other processors are available, although it is not a goal to support vector instructions on CPU architectures other than x86 and AArch64

Use Cases

- Fast Computation of Arrays
- Machine Learning and Ai
- Gaming Arithmetic and Physics Engines
- Imaging and Audio
- Cryptography

Parallel Processing

- In Standard Java Today:
 - Distributed Process - Break up the data into small chunks and perform your calculations
 - Carried through with threads
 - Might be performed concurrently and not really in parallel, depending if threads are completely scheduled and have to undergo timesharing
- Using Vector API:
 - Fast Parallel Computation, not Concurrent
 - SIMD **does not use** Threads
 - Uses large amount of operating units, executing the same operation *at the same CPU cycle* at the exact same time
 - Requires the C2 Compiler



Vector API is Waiting

- Vector API is waiting for Project Valhalla to be released
- What does Project Valhalla have that Vector API under Panama need?
 - Inline Classes - The current implementation has to box/unbox vector objects and store them in heap, which would defeat the purpose of having fast parallelism.
 - Primitive Classes - Allow us to declare classes that behave like primitives

Species



Species

- Know the types that we need to use: byte, short, int, long, float, and double
- Know the size of what can be handled, anywhere from 64 bits to 512 bits, *for now*
- Depending on the type that used is called the *species*

Vectors



Vectors

- A vector computation consists of a sequence of operations on vectors
- A vector comprises a (usually) fixed sequence of scalar values, where the scalar values correspond to the number of hardware-defined vector lanes
- A binary operation applied to two vectors with the same number of lanes would, for each lane, apply the equivalent scalar operation on the corresponding two scalar values from each vector.

Vector<E>

- The Vector<E> class has six abstract subclasses for each of the six supporting types: ByteVector, ShortVector, IntVector, LongVector, FloatVector, and DoubleVector.
- Specific implementations are important with SIMD machines, which is why shape-specific subclasses further extend these classes for each type.
- Vector<E> is mapped to a hardware vector register when vector computations are compiled by the HotSpot C2 compiler.
- For example Int128Vector, Int512Vector, etc. These will be abstracted.

2

3

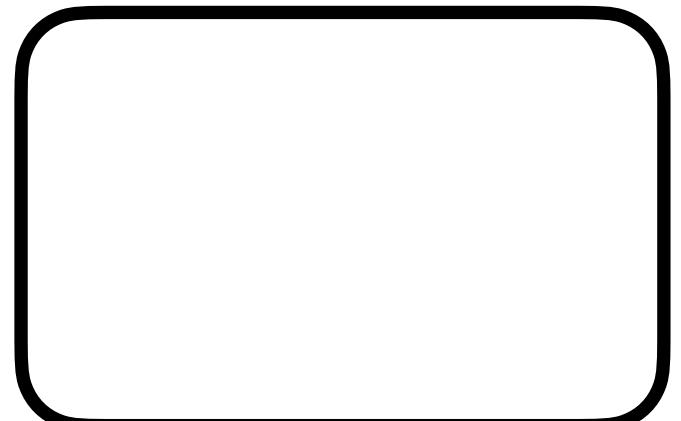
0



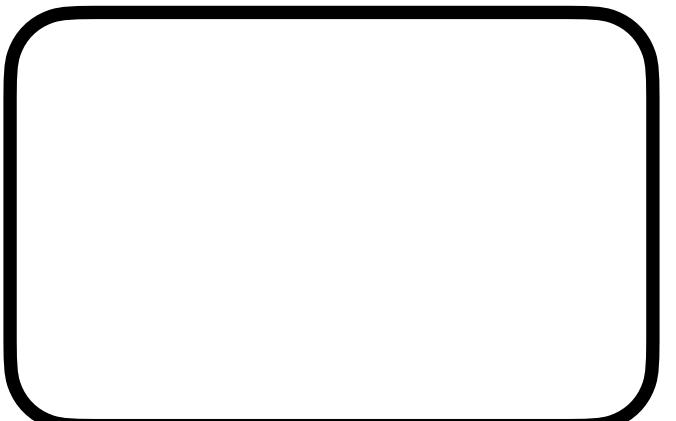
+



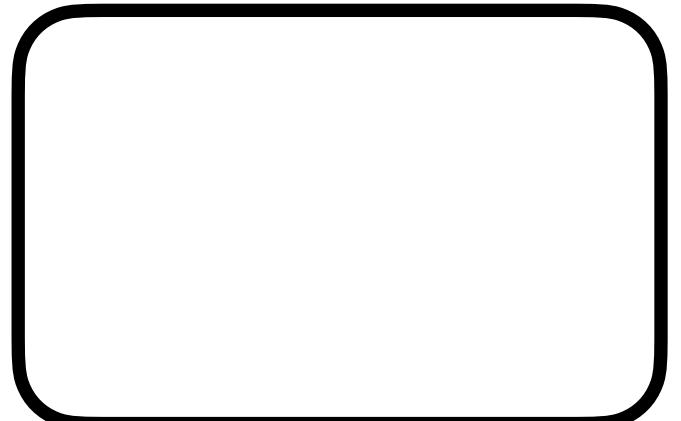
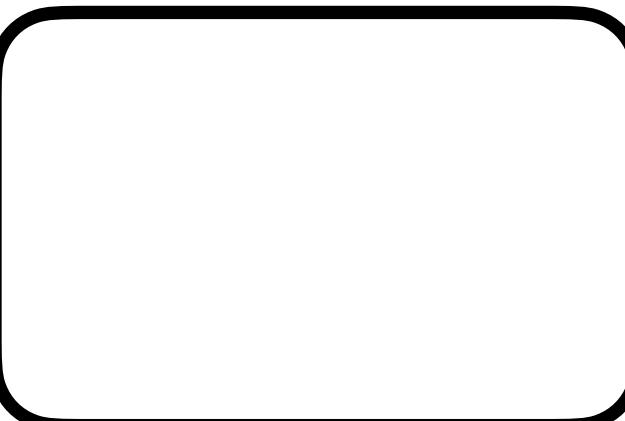
=



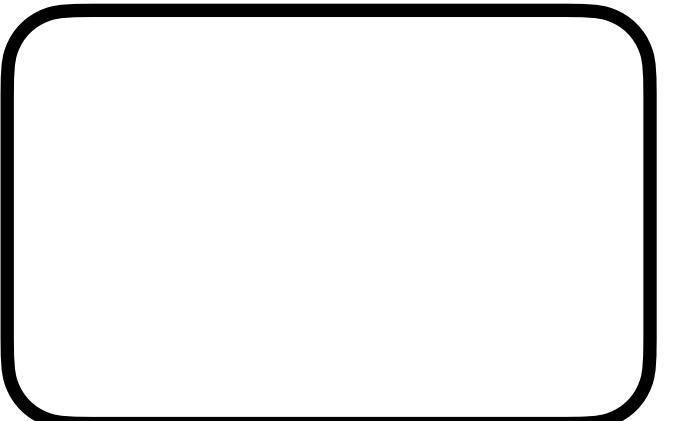
+



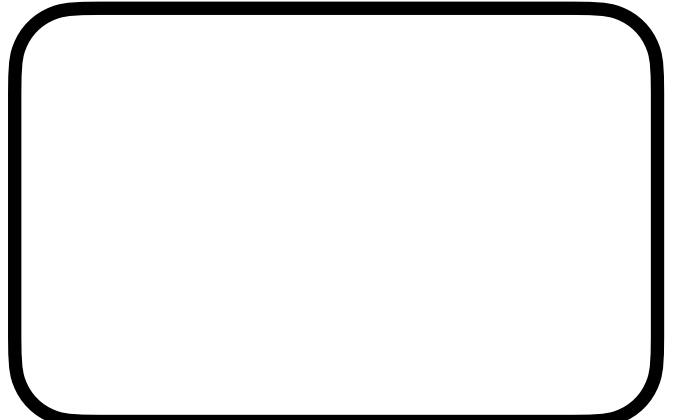
=



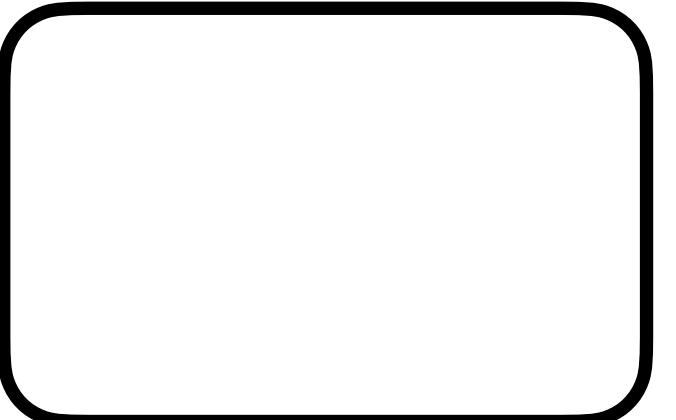
+



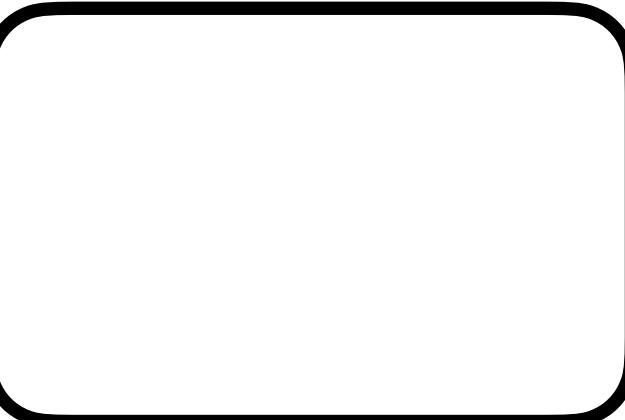
=



+



=



5

9

0

+

=

+

=

+

=

2

+

3

=

0

3

6

0

+

=

+

=

5

+

9

=

0

2

+

3

=

0

1

4

0

+

=

3

+

6

=

0

5

+

9

=

0

2

+

3

=

0

$$\begin{array}{r} \boxed{1} \\ + \\ \boxed{4} \\ = \\ \boxed{0} \end{array}$$
$$\begin{array}{r} \boxed{3} \\ + \\ \boxed{6} \\ = \\ \boxed{0} \end{array}$$
$$\begin{array}{r} \boxed{5} \\ + \\ \boxed{9} \\ = \\ \boxed{0} \end{array}$$
$$\begin{array}{r} \boxed{2} \\ + \\ \boxed{3} \\ = \\ \boxed{0} \end{array}$$

Vector 1

$$\begin{array}{c} \boxed{1} \\ + \\ \boxed{3} \\ + \\ \boxed{5} \\ + \\ \boxed{2} \end{array} \quad \begin{array}{c} \boxed{4} \\ + \\ \boxed{6} \\ + \\ \boxed{9} \\ + \\ \boxed{3} \end{array} = \begin{array}{c} \boxed{0} \\ = \boxed{0} \\ = \boxed{0} \\ = \boxed{0} \end{array}$$

Vector 2

$$\begin{array}{c} \boxed{1} \\ + \\ \boxed{3} \\ + \\ \boxed{5} \\ + \\ \boxed{2} \end{array} \quad \boxed{4} \quad \boxed{6} \quad \boxed{9} \quad \boxed{3} = \begin{array}{c} \boxed{0} \\ = \\ \boxed{0} \\ = \\ \boxed{0} \\ = \\ \boxed{0} \end{array}$$

Vector 3

1

+

4

3

+

6

5

+

9

2

+

3

=

0

=

0

=

0

=

0

$$\begin{array}{r} \boxed{1} \\ + \\ \boxed{4} \\ = \\ \boxed{0} \end{array}$$
$$\begin{array}{r} \boxed{3} \\ + \\ \boxed{6} \\ = \\ \boxed{0} \end{array}$$
$$\begin{array}{r} \boxed{5} \\ + \\ \boxed{9} \\ = \\ \boxed{0} \end{array}$$
$$\begin{array}{r} \boxed{2} \\ + \\ \boxed{3} \\ = \\ \boxed{0} \end{array}$$

Lane 1

$$\begin{array}{ccc} \boxed{1} & + & \boxed{4} \\ \boxed{3} & + & \boxed{6} \\ \boxed{5} & + & \boxed{9} \\ \boxed{2} & + & \boxed{3} \end{array} = \begin{array}{c} \boxed{0} \\ \boxed{0} \\ \boxed{0} \\ \boxed{0} \end{array}$$

Lane 2

$$\begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 4 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 6 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 2 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

Lane 3

$$\begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 4 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 6 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 2 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

Lane 4

$$\begin{array}{ccc} \boxed{1} & + & \boxed{4} \\ \boxed{3} & + & \boxed{6} \\ \boxed{5} & + & \boxed{9} \\ \boxed{2} & + & \boxed{3} \end{array} = \begin{array}{c} \boxed{0} \\ \boxed{0} \\ \boxed{0} \\ \boxed{0} \end{array}$$

Four Total Components

(In this example)

$$\begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 4 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 6 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

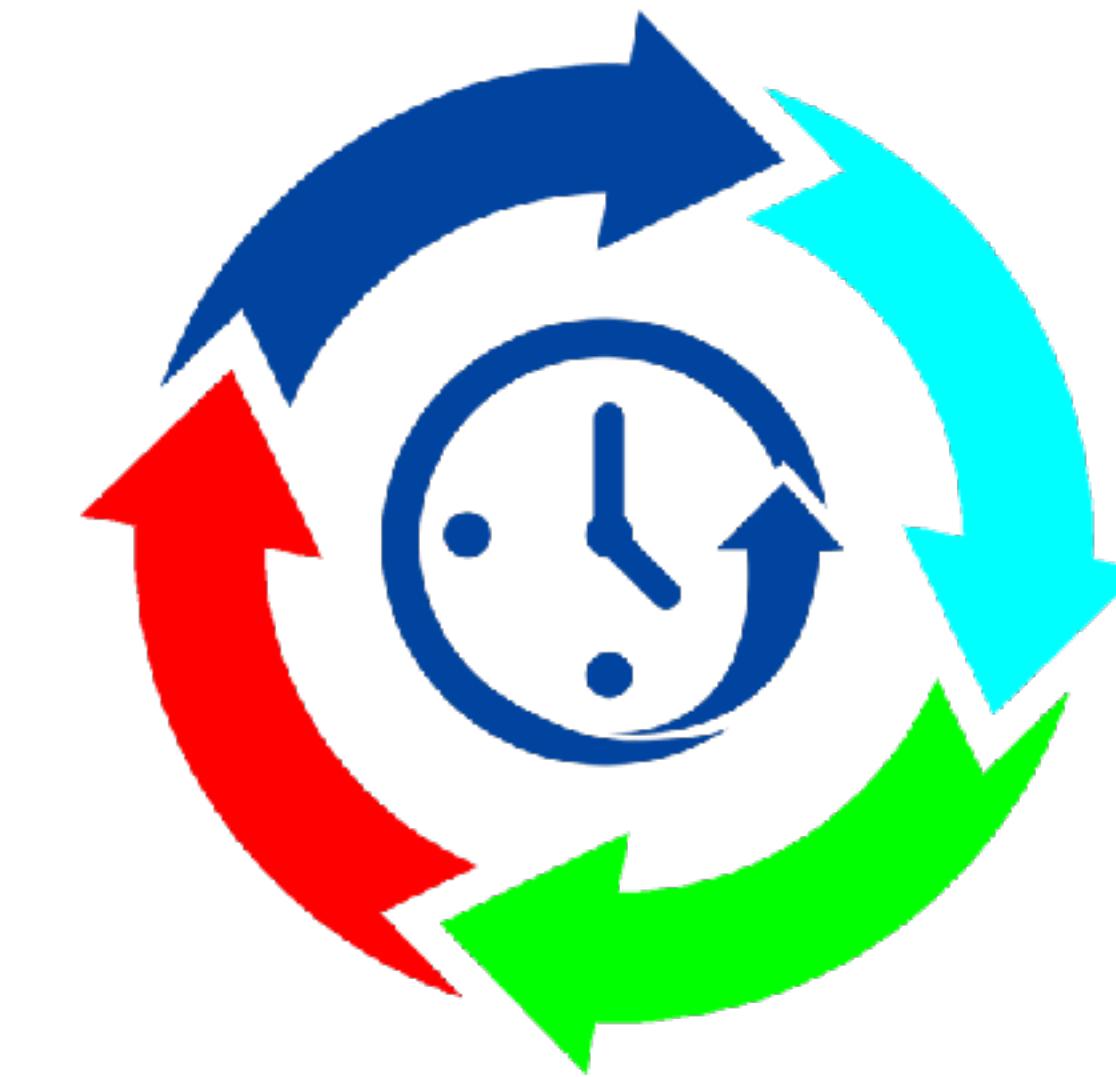
$$\begin{array}{|c|} \hline 2 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$



$$\begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 4 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 6 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 14 \\ \hline \end{array}$$

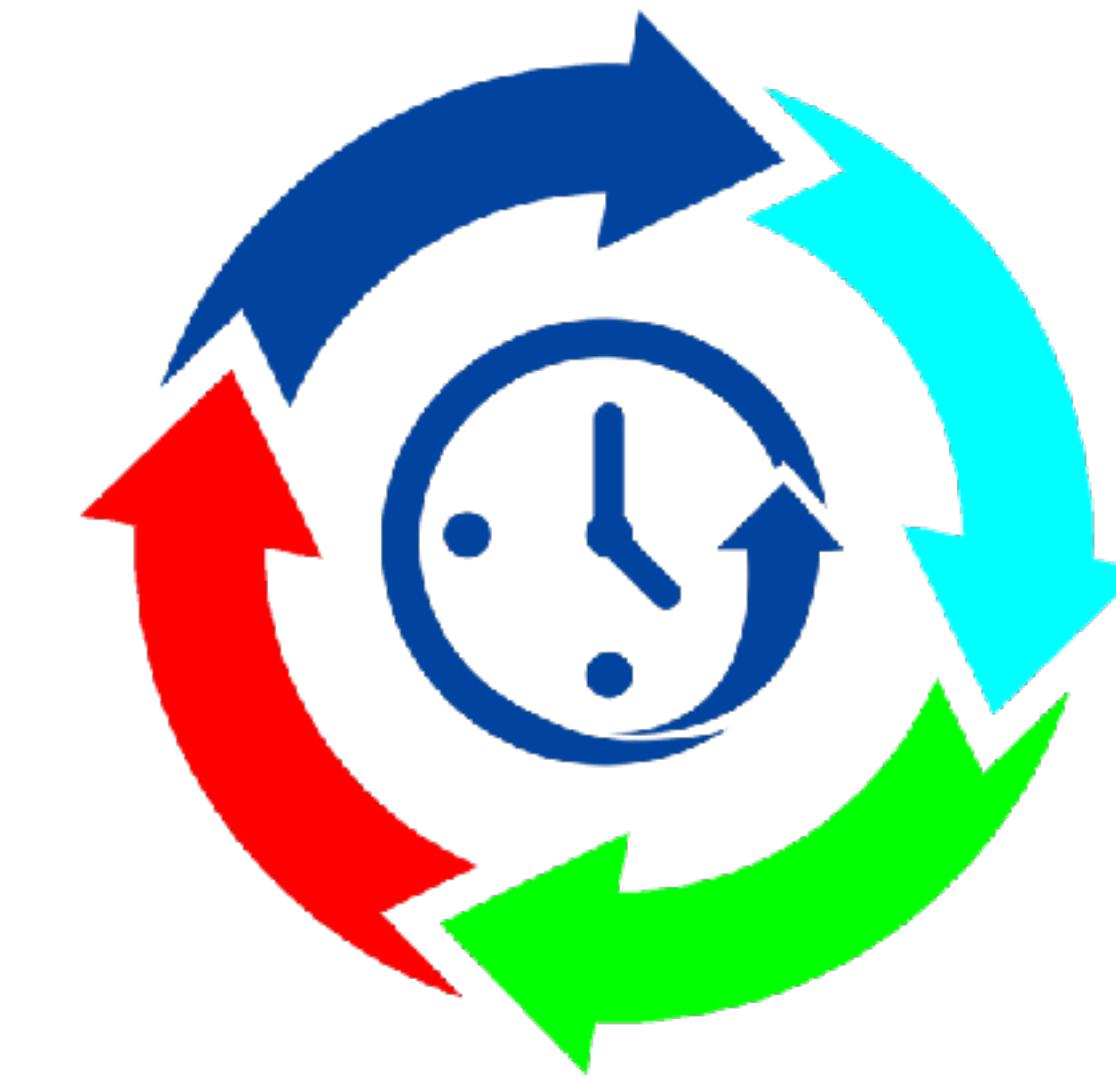
$$\begin{array}{|c|} \hline 2 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$



$$\begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 4 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$
$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 6 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$
$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 14 \\ \hline \end{array}$$
$$\begin{array}{|c|} \hline 2 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$

7

2

0

+

=

+

=

+

=

+

=

5

1

0

+

=

+

=

+

=

7

+

2

=

0

4

3

0

+

=

+

=

5

+

1

=

0

7

+

2

=

0

9

6

0

+

=

4

+

3

=

0

5

+

1

=

0

7

+

2

=

0

$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 6 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 4 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

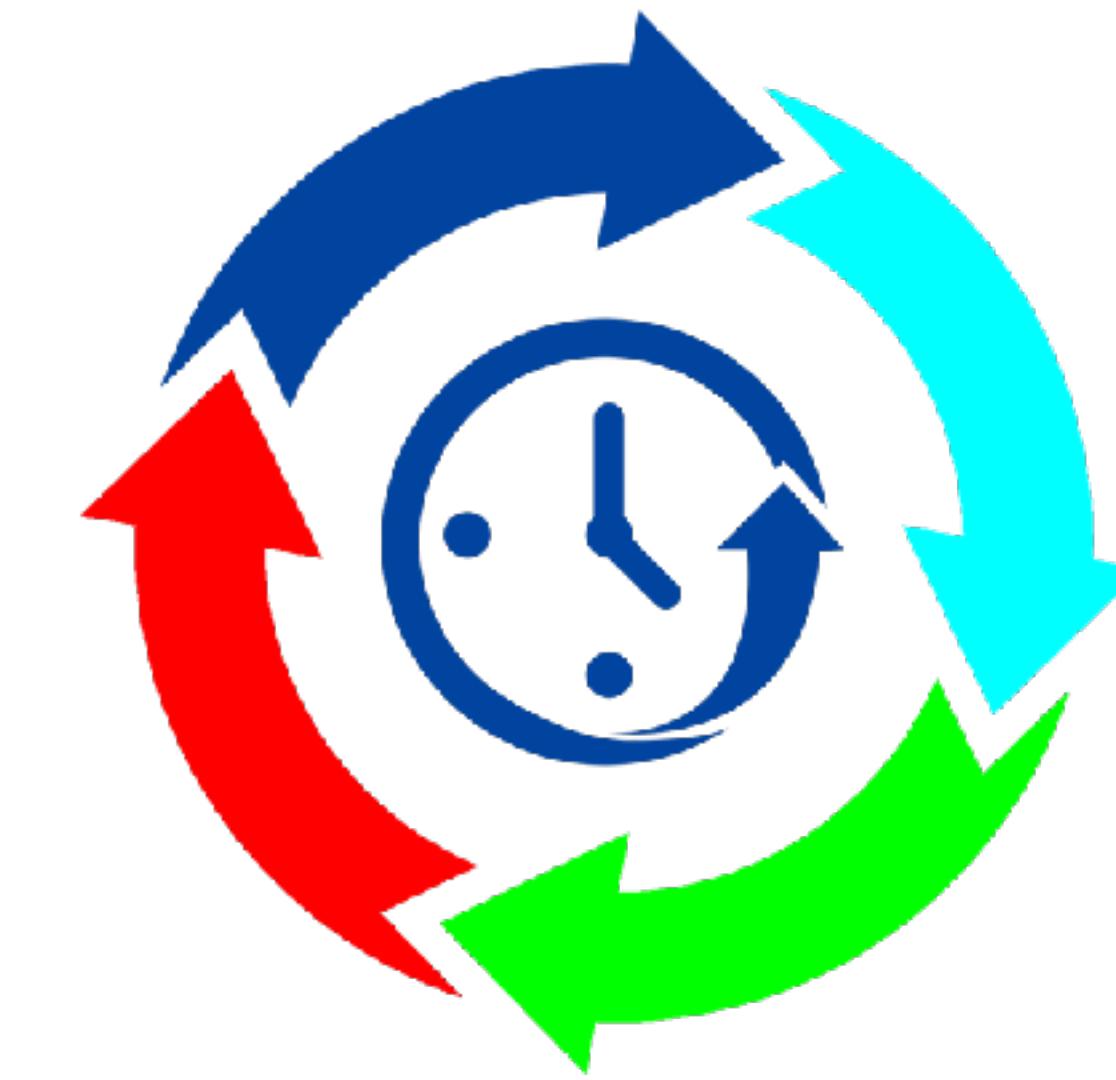
$$\begin{array}{|c|} \hline 7 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 2 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$



$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 6 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 15 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 4 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 7 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 6 \\ \hline \end{array}$$

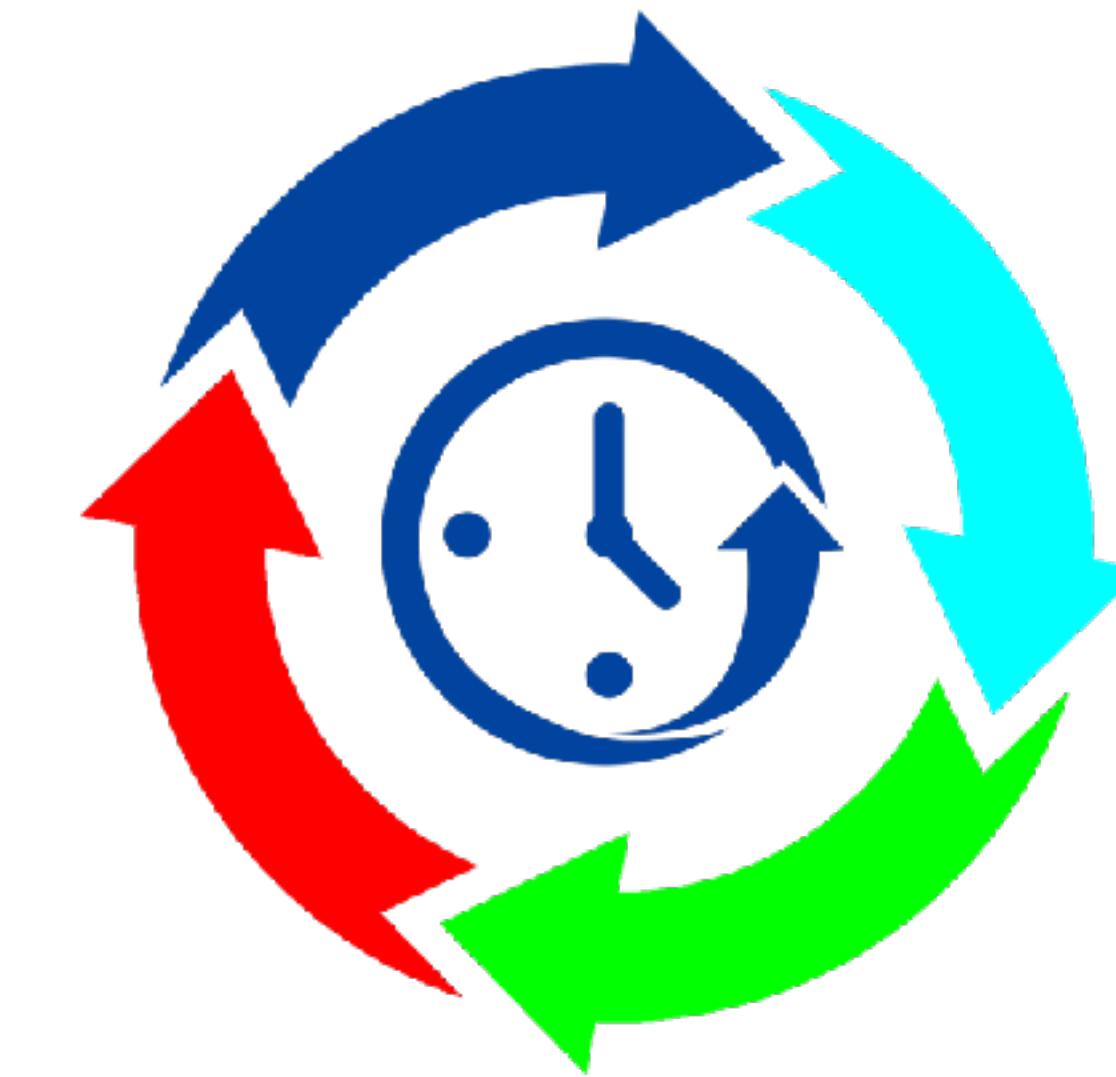
$$\begin{array}{|c|} \hline 7 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 2 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$



SIMD Register Width (w) = 256 bits

Size for Float (s) = 32 bits

Number of Lanes = w/s

SIMD Register Width (w) = 256 bits

Size for Float (s) = 32 bits

Number of Lanes = 256/32

SIMD Register Width (w) = 256 bits

Size for Float (s) = 32 bits

Number of Lanes = 8

Demo: Creating a Vector API Application



- Here we will create a Vector API application in a simple form that will be an exact fit with the number of components
- We will run AddStandardVector and AddStandardVectorLoop

Masking



Masking

- To support control flow, some vector operations optionally accept masks represented by the public abstract class `VectorMask<E>`
- Each element in a mask is a boolean value corresponding to a vector lane.
- A mask selects the lanes to which an operation is applied: It is applied if the mask element for the lane is true, and some alternative action is taken if the mask is false.
- In other words, it's like window where each slice will be processed in parallel



Applying Masks

- Similar to vectors, instances of `VectorMask<E>` are instances of non-public concrete subclasses defined for each element type and length combination.
- The instance of `VectorMask<E>` used in an operation should have the same type and length as the vector instances involved in the operation.
- Vector comparison operations produce masks, which can then be used as input to other operations to selectively operate on certain lanes and thereby emulate flow control.
- Masks can also be created using static factory methods in the `VectorMask<E>` class.



Efficient Application of Mask<T>

- On such platforms an instance of `VectorMask<E>` is mapped to a predicate register, and a mask-accepting operation is compiled to a predicate-register-accepting vector instruction.
- On platforms that don't support predicate registers, a less efficient approach is applied: An instance of `VectorMask<E>` is mapped, where possible, to a compatible vector register, and in general a mask-accepting operation is composed of the equivalent unmasked operation and a blend operation.



12

8

5

22

true

true

true

true

12

8

5

22

12

8

5

22

false

true

false

true

0

8

0

22

12

8

5

22

true

true

false

false

12

8

0

0

12

8

true

true

false

false

12

8

0

0

SPECIES.indexInRange(index, a.length)

Demo: Using Masking



- Let's use some masking so that we can be sure to process all the data in chunks.
- We will run:
 - AddStandardVectorLoopWith Mask,
 - AddStandardVectorLoopWith outMask
 - BroadcastVector

Vector Operations



Vector Operations

- There are two types of vector operations:
 - Lane-wise operations
 - Cross-lane operations

Lane Wise Operations

- Performs a scalar operation, like addition, on a single lane on one or more vectors at a time.
- These operations can combine one lane of a vector with a lane of a second vector, for instance, during an add operation.
- Operations are further classified as unary, binary, ternary, test, or conversion operations

$$\begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 4 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 6 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$

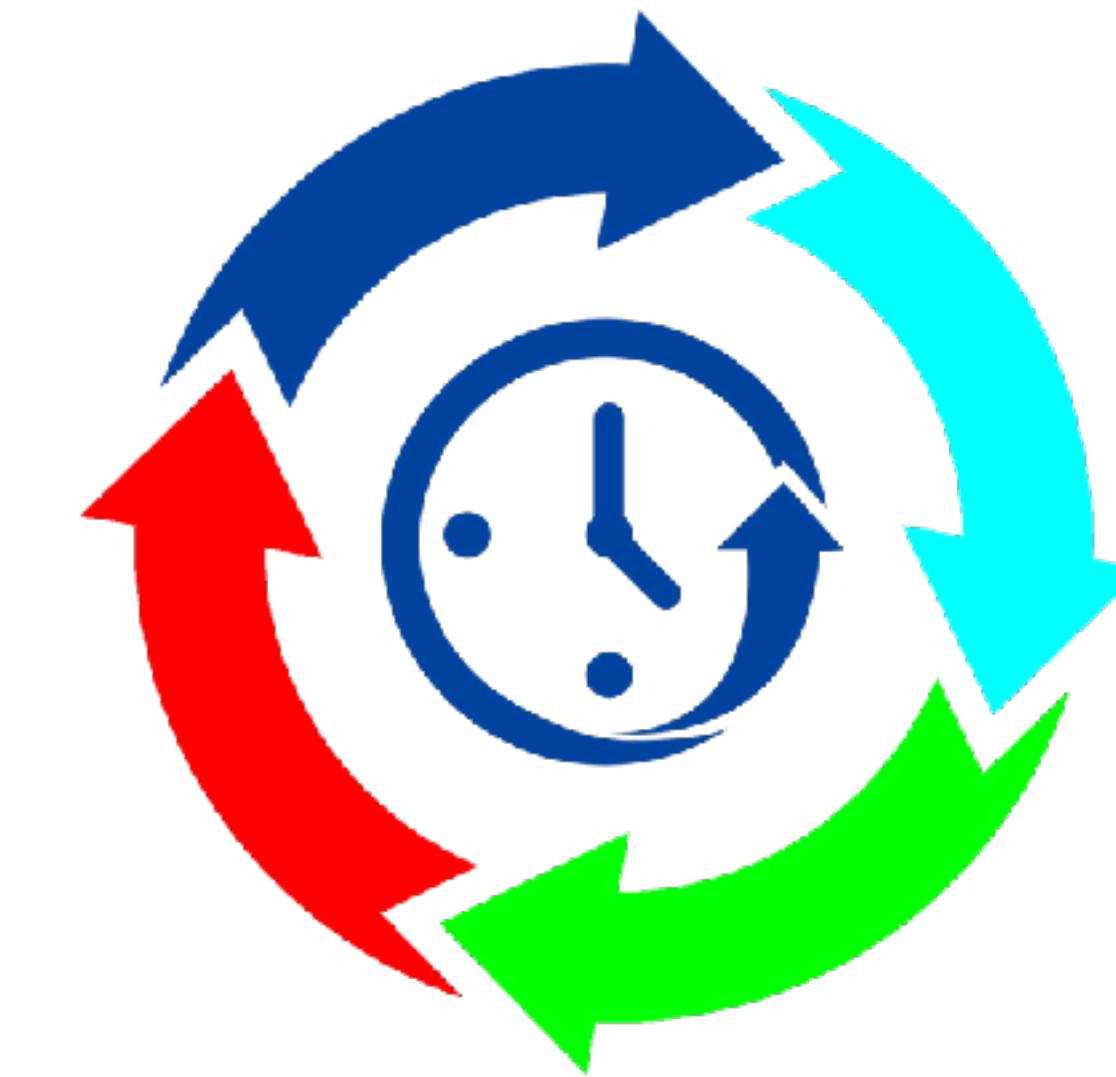
$$\begin{array}{|c|} \hline 2 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 0 \\ \hline \end{array}$$



$$\begin{array}{|c|} \hline 1 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 4 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 6 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 9 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 14 \\ \hline \end{array}$$

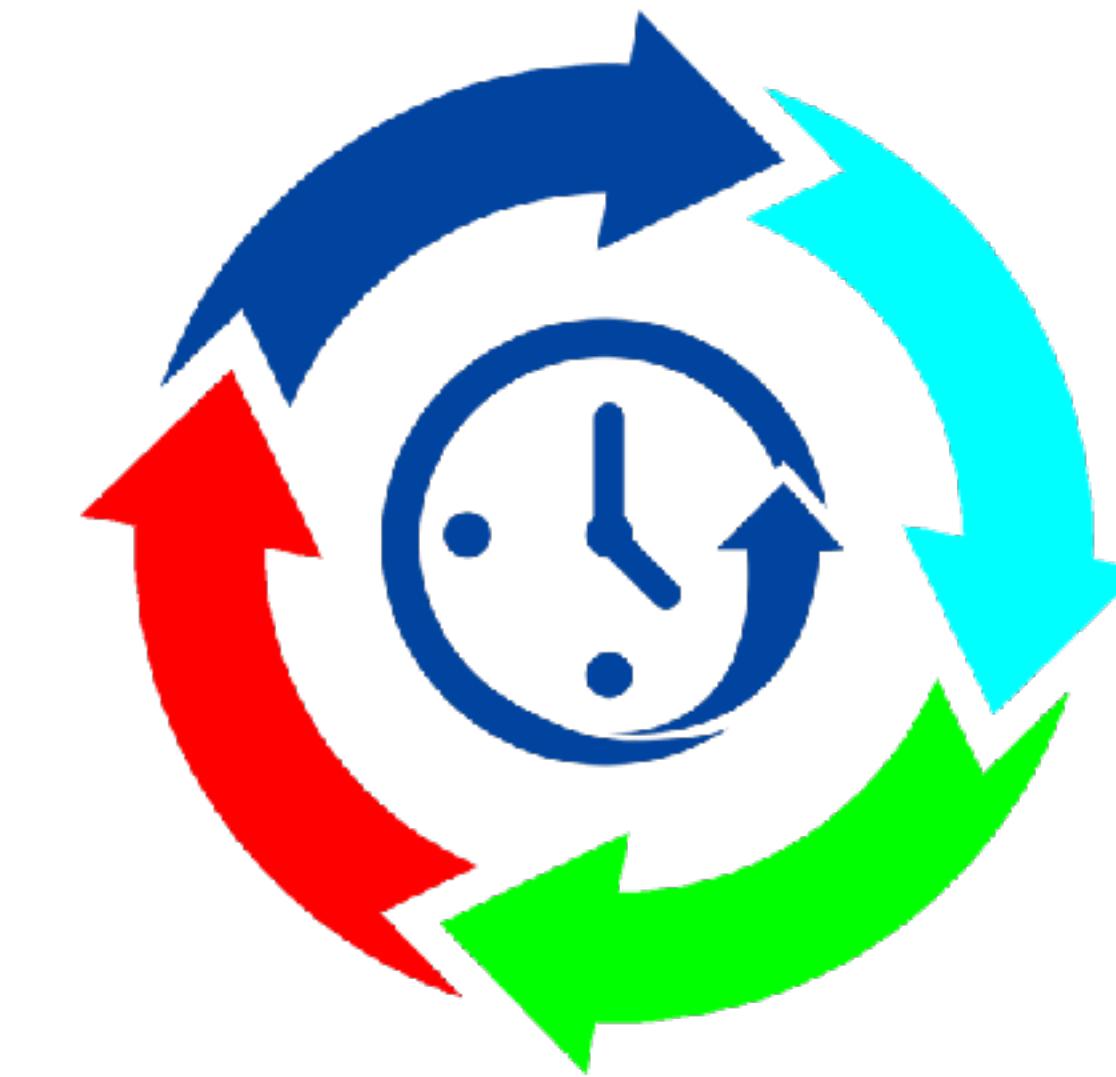
$$\begin{array}{|c|} \hline 2 \\ \hline \end{array}$$

+

$$\begin{array}{|c|} \hline 3 \\ \hline \end{array}$$

=

$$\begin{array}{|c|} \hline 5 \\ \hline \end{array}$$



Cross Lane Operations

- Compute or modify data from different lanes of an entire vector.
- Sorting the components of a vector is an example of a cross-lane operation.
- Cross-lane operations can produce scalars or vectors of different shapes from the source vectors.
- Cross-lane operations can be further classified into permutation and reduction operations

$$\begin{array}{c} 1 \\ = \\ \boxed{1} \end{array}$$

$$\begin{array}{c} 0 \\ = \\ \boxed{0} \end{array}$$

$$\begin{array}{c} 3 \\ = \\ \boxed{3} \end{array}$$

$$\begin{array}{c} 0 \\ = \\ \boxed{0} \end{array}$$

$$\begin{array}{c} 5 \\ = \\ \boxed{5} \end{array}$$

$$\begin{array}{c} 0 \\ = \\ \boxed{0} \end{array}$$

$$\begin{array}{c} 2 \\ = \\ \boxed{2} \end{array}$$

$$\begin{array}{c} 0 \\ = \\ \boxed{0} \end{array}$$



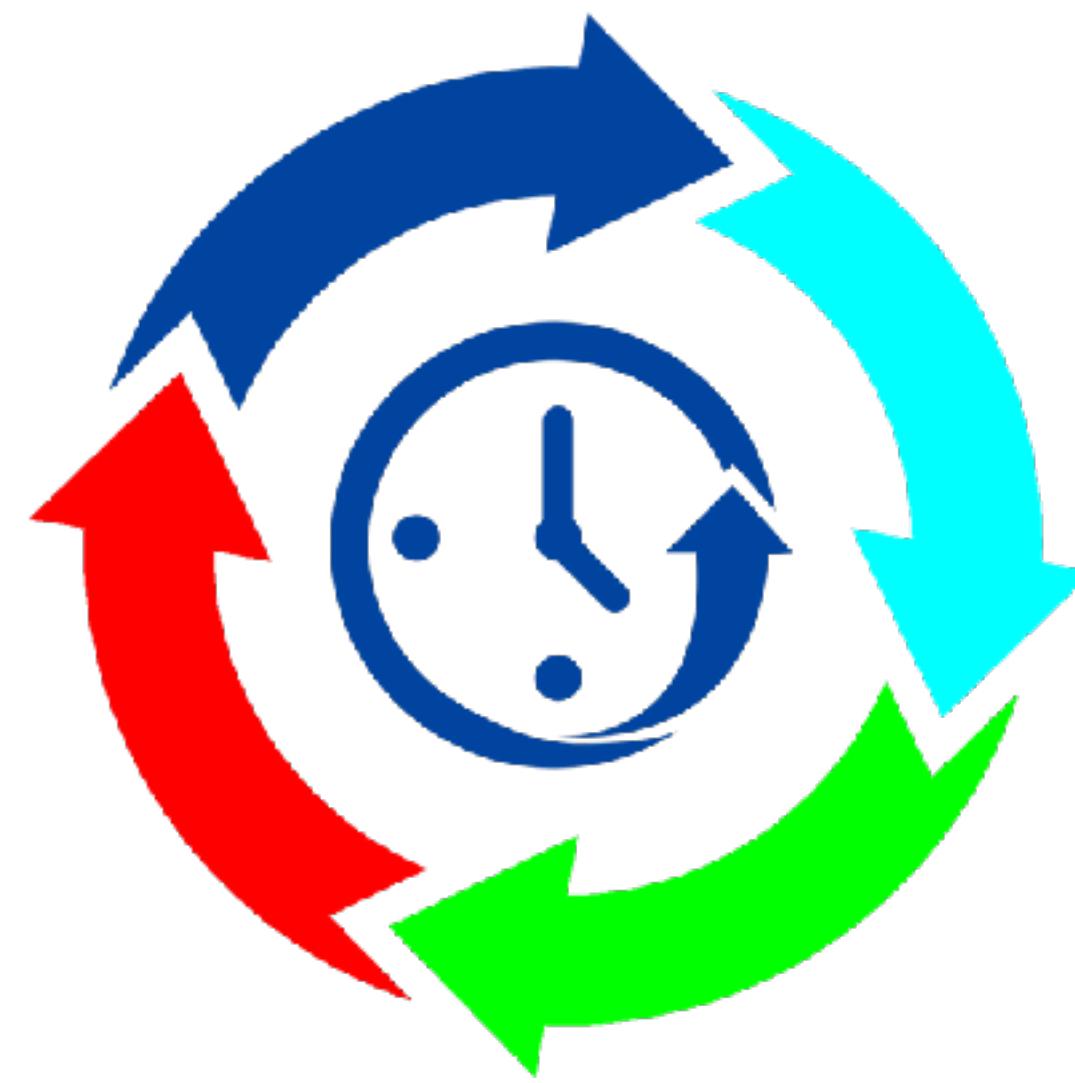
$$\begin{array}{c} 1 \\ = \\ \boxed{1} \end{array}$$

$$\begin{array}{c} 3 \\ = \\ \boxed{3} \end{array}$$

$$\begin{array}{c} 5 \\ = \\ \boxed{5} \end{array}$$

$$\begin{array}{c} 2 \\ = \\ \boxed{2} \end{array}$$

$$\begin{array}{c} 1 \\ = \\ \boxed{1} \\ \\ 3 \\ = \\ \boxed{3} \\ \\ 0 \\ = \\ \boxed{0} \\ \\ 0 \\ = \\ \boxed{0} \end{array}$$



Cross Lane Permutation

- To support cross-lane permutation operations, some vector operations accept shuffles represented by the public abstract class `VectorShuffle<E>`.
- Each element in a shuffle is an `int` value corresponding to a lane index.
- A shuffle is a mapping of lane indexes, describing the movement of lane elements from a given vector to a result vector.
- Similar to vectors and masks, instances of `VectorShuffle<E>` are instances of non-public concrete subclasses defined for each element type and length combination.
- The instance of `VectorShuffle<E>` used in an operation should have the same type and length as the vector instances involved in the operation.

Demo: Vector Shuffling



- Let's perform some vector cross-lane operations
 - CompareVector
 - BroadcastVector
 - ReduceVector
 - ReduceVectorBlend
 - ShuffleVector

Demo: Let's Make a Machine Learning Model!



- Let's take a look at rudimentary Machine Learning Model!

Thank You



- Email: dhinojosa@evolutionnext.com
- Github: <https://www.github.com/dhinojosa>
- Mastodon: <https://mastodon.social/@dhinojosa>
- Linked In: <http://www.linkedin.com/in/dhevolutionnext>