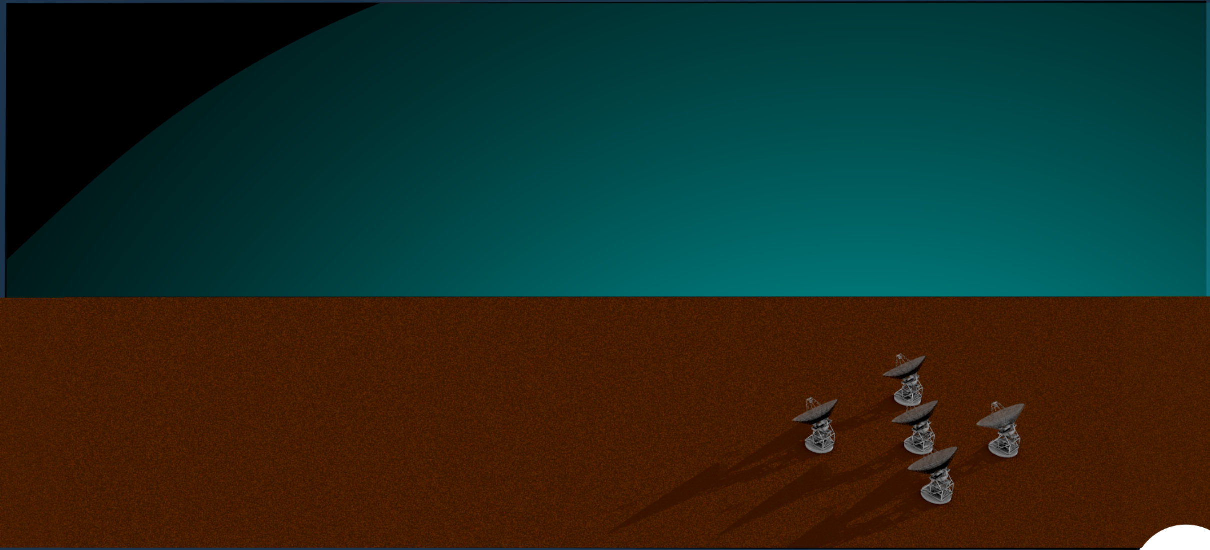
Laboratoire d'Annecy de Physique des Particules

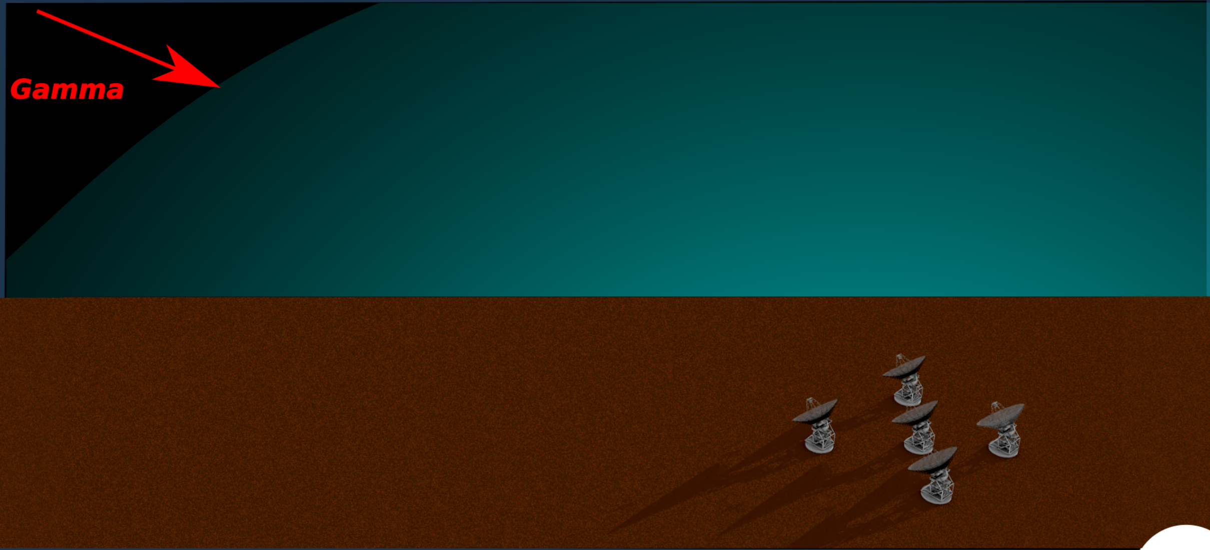
# GPU with C++20 : CTA example

Pierre Aubert

# Event reconstruction

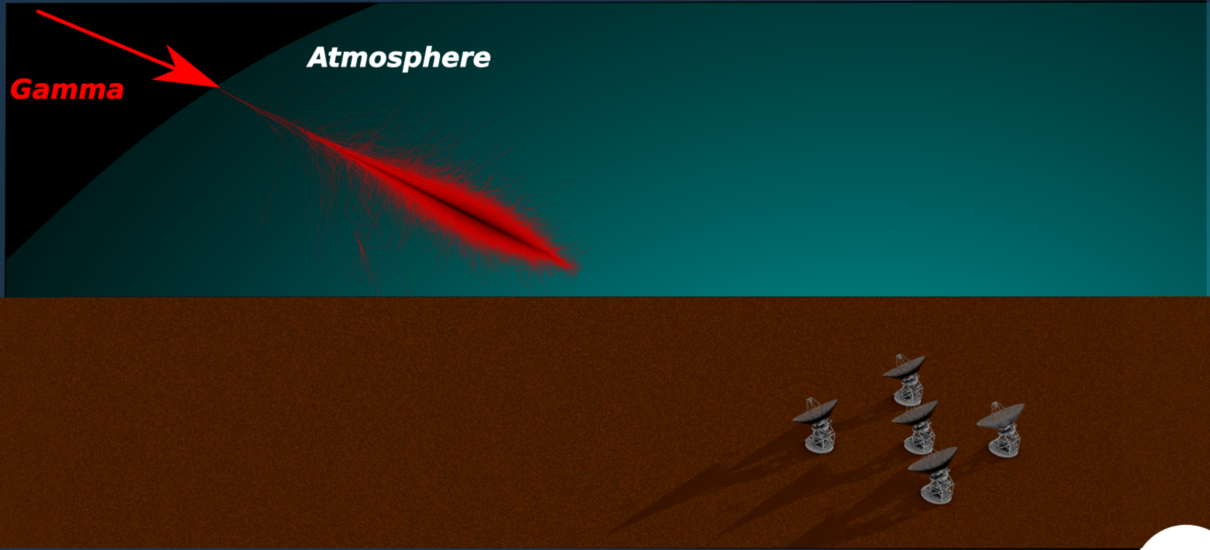


# Event reconstruction

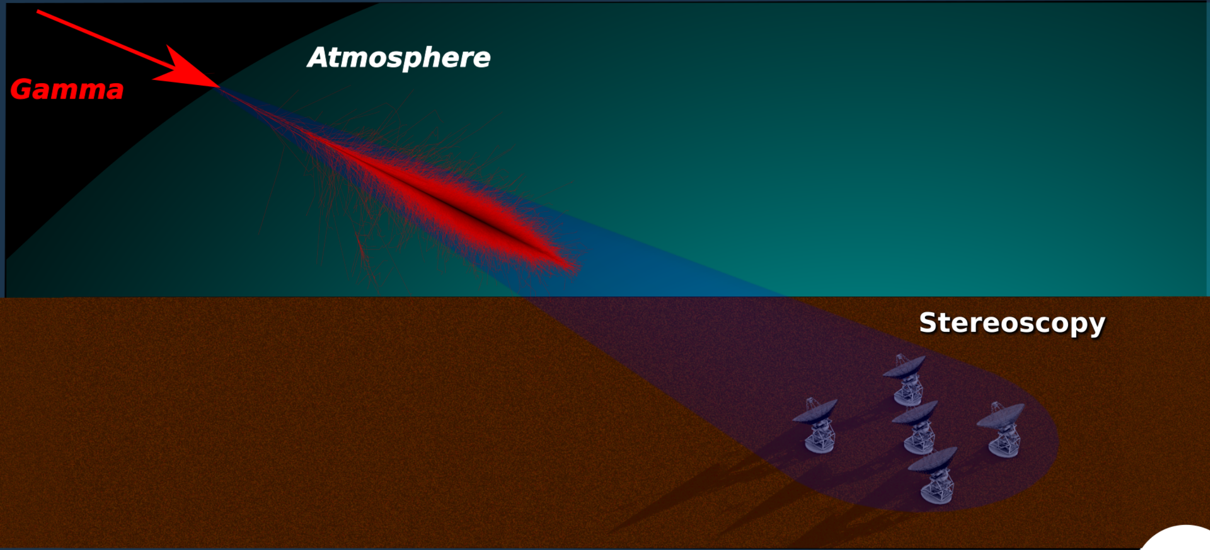


## Event reconstruction

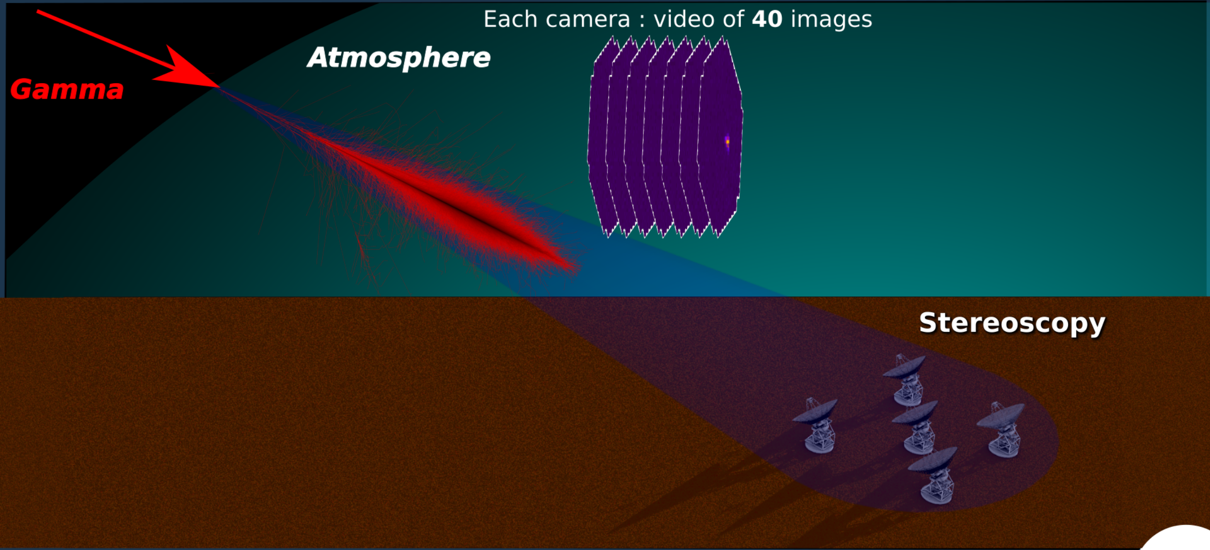
# Event reconstruction



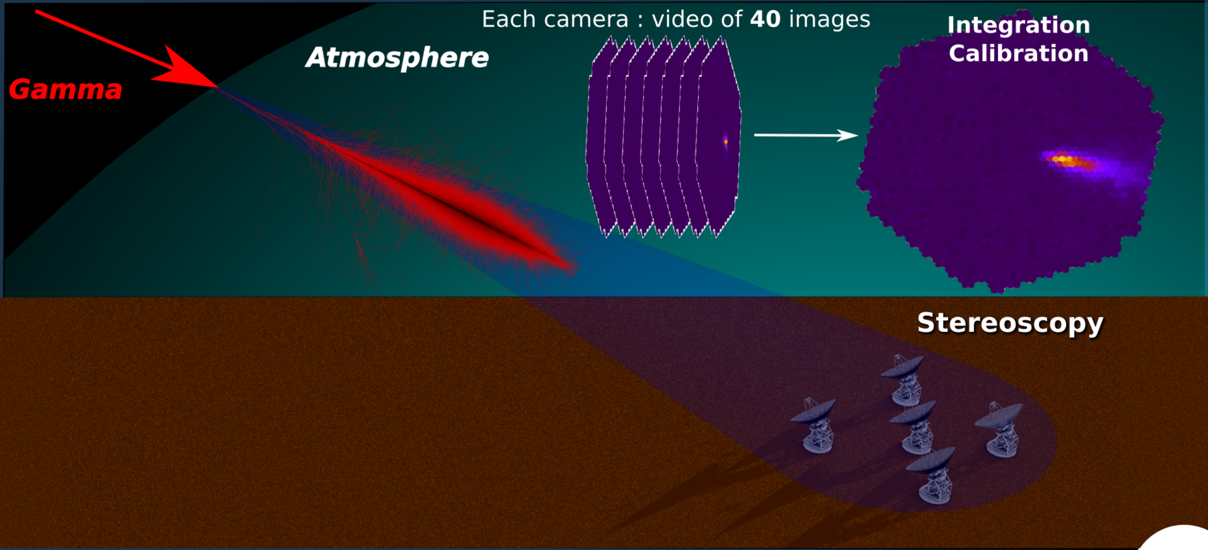
# Event reconstruction



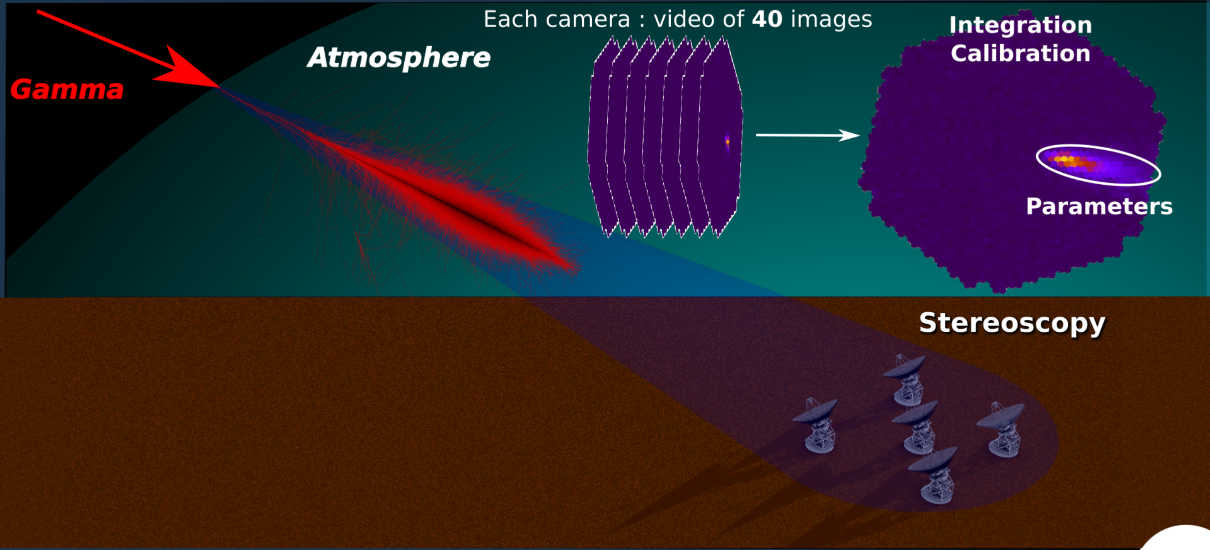
# Event reconstruction



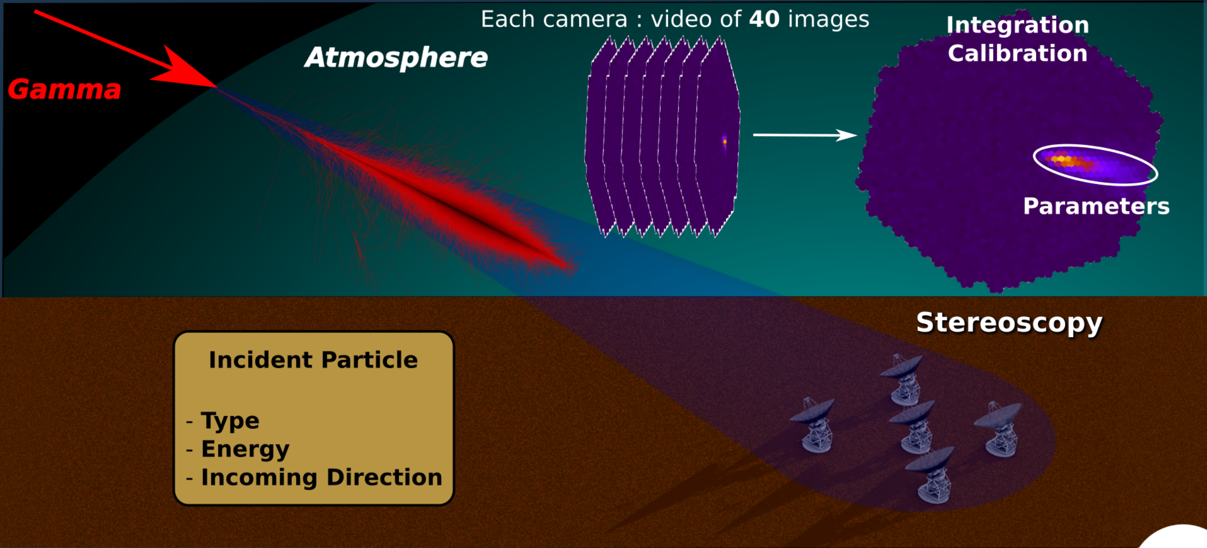
# Event reconstruction



# Event reconstruction



# Event reconstruction



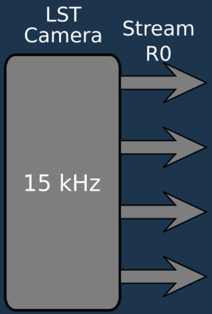

**Incident Particle**

- **Type**
- **Energy**
- **Incoming Direction**

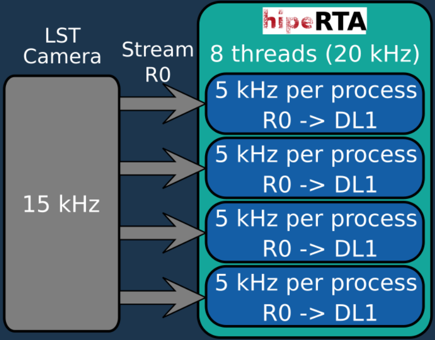
# HiPeRTA : R0 -> DL3

LST Camera

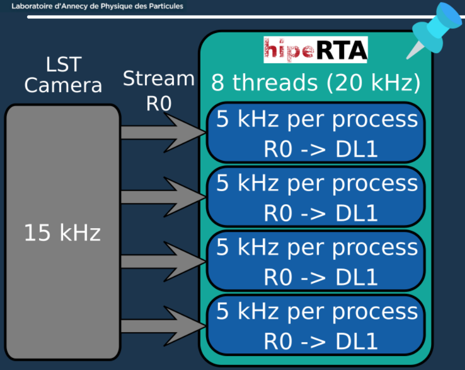
# HiPeRTA : R0 -> DL3



# HiPeRTA : R0 -> DL3

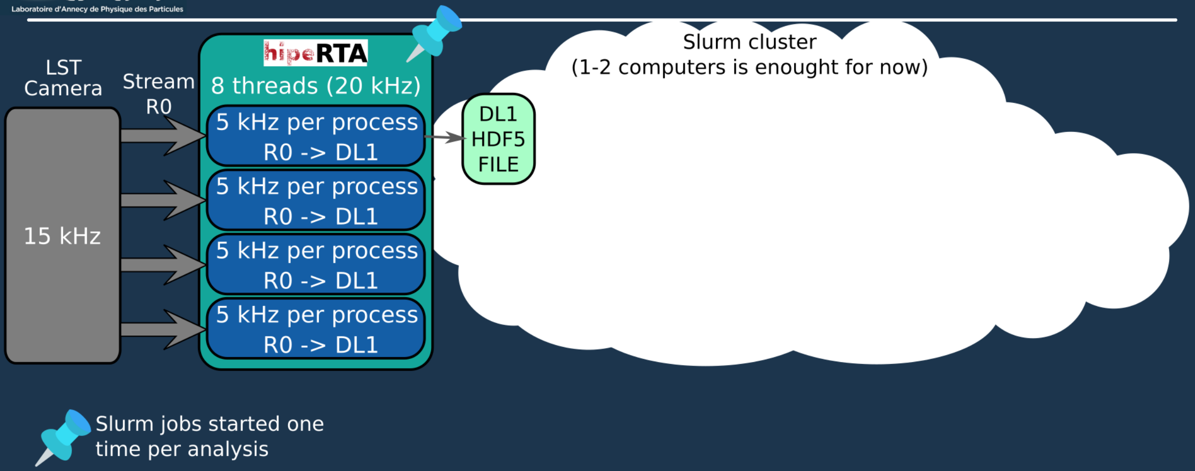


# HiPeRTA : R0 -> DL3

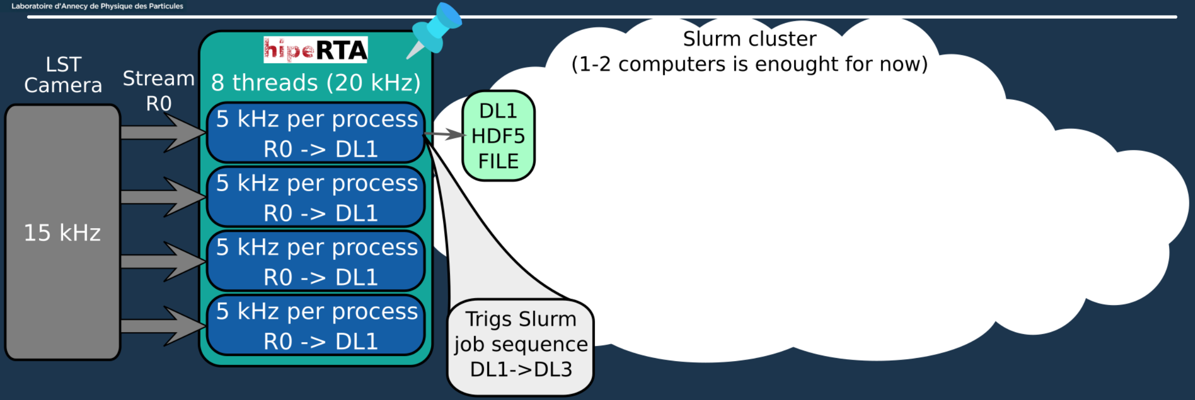



Slurm jobs started one time per analysis

# HiPeRTA : R0 -> DL3

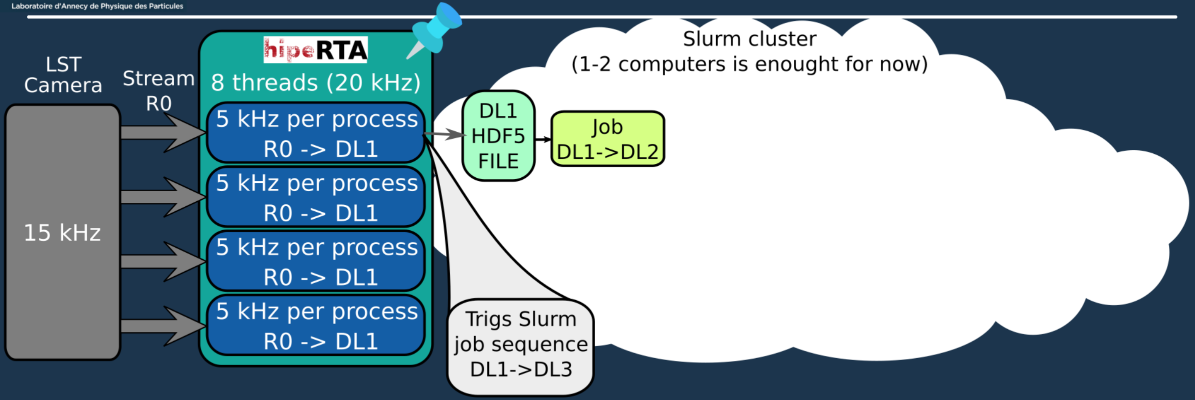


# HiPeRTA : R0 -> DL3



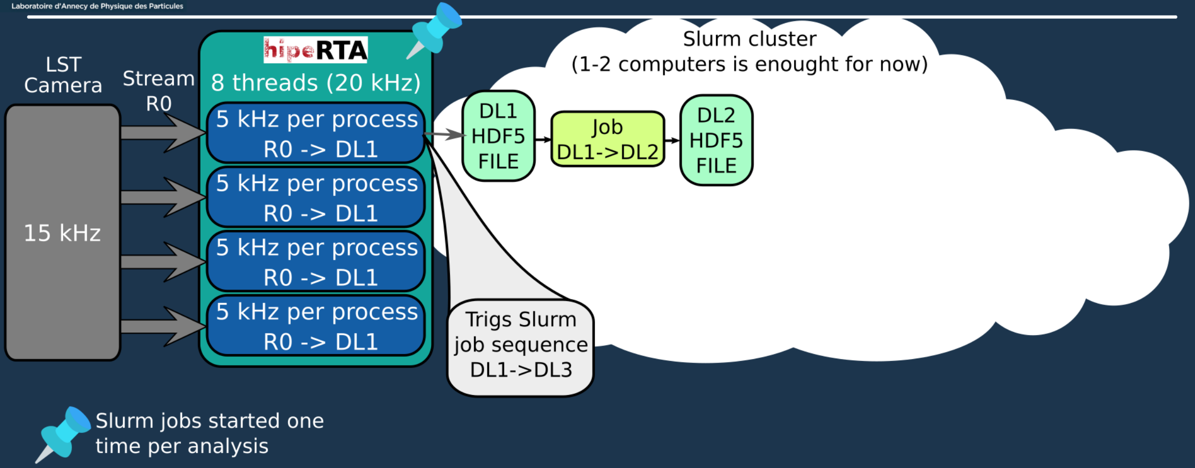

Slurm jobs started one time per analysis

# HiPeRTA : R0 -> DL3

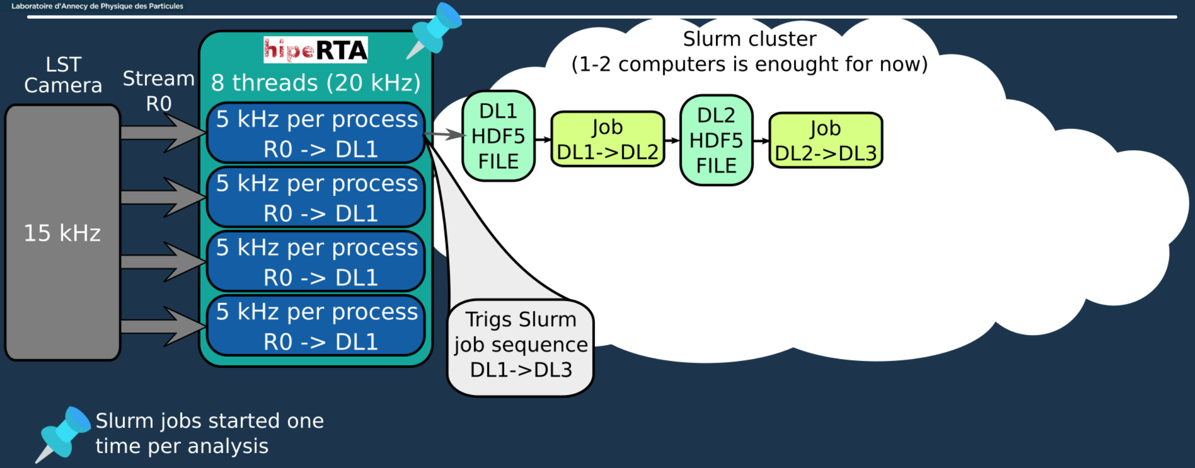



Slurm jobs started one time per analysis

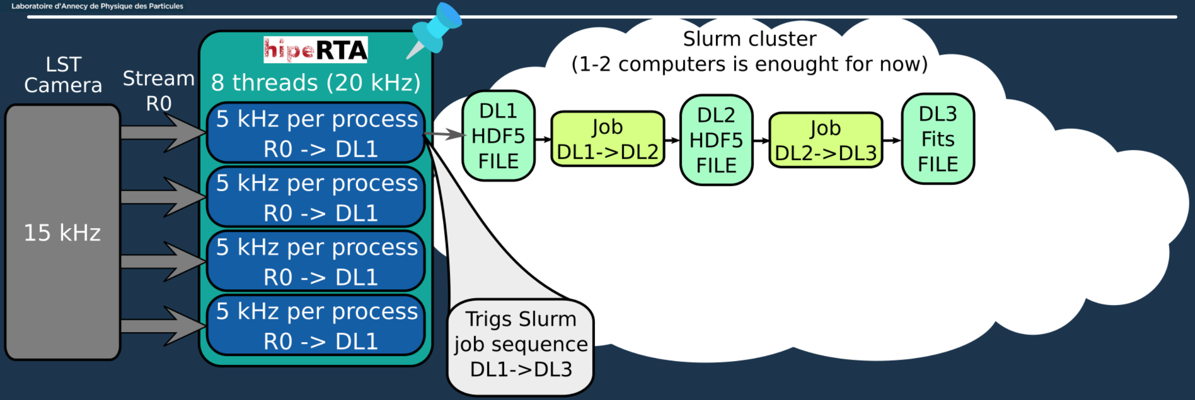
# HiPeRTA : R0 -> DL3



# HiPeRTA : R0 -> DL3

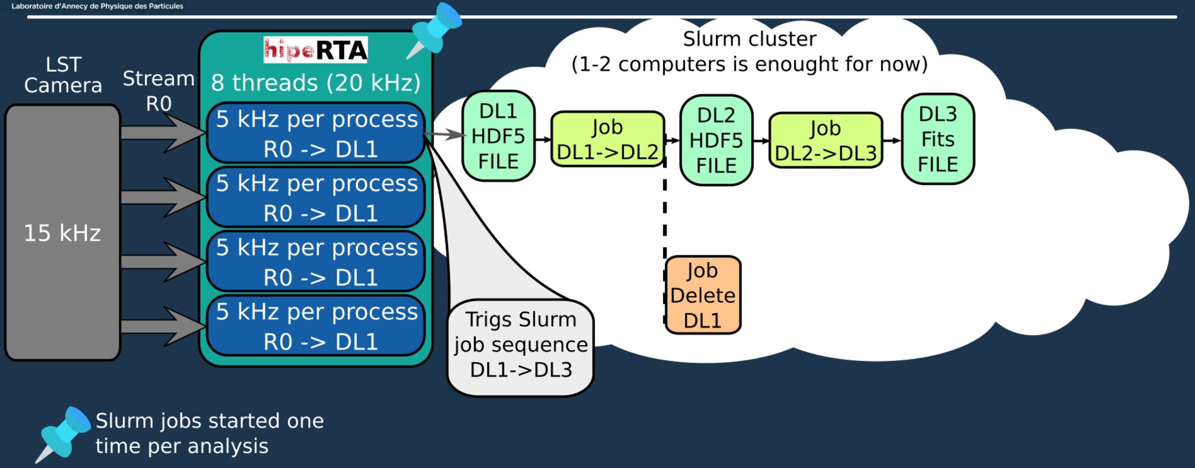


# HiPeRTA : R0 -> DL3

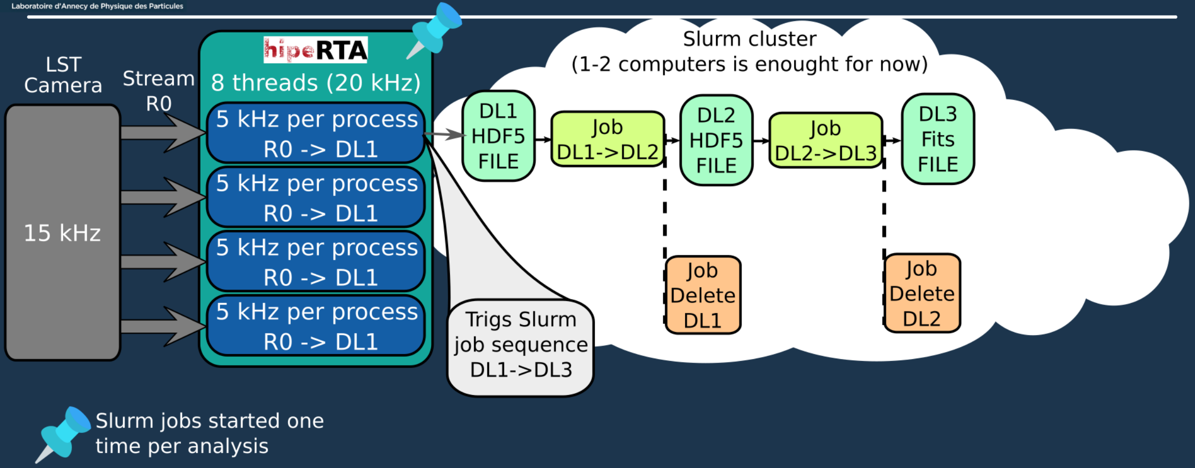



Slurm jobs started one time per analysis

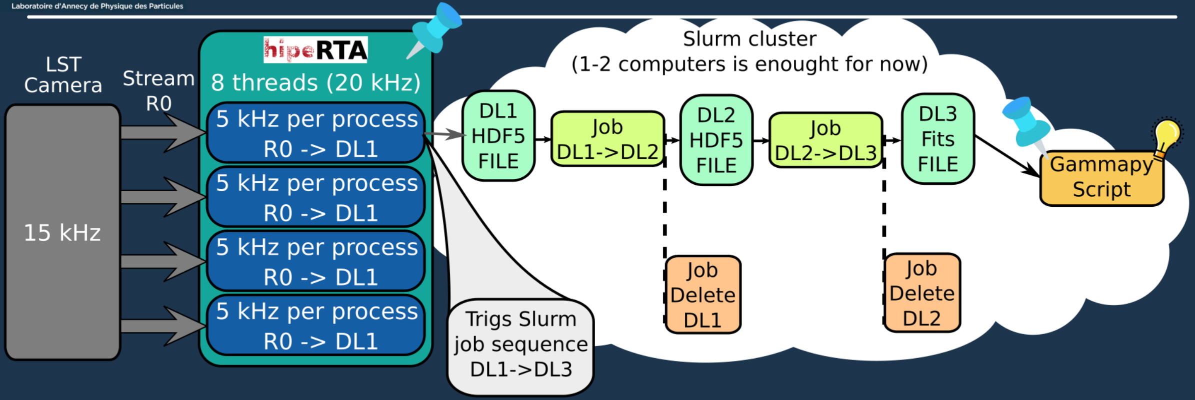
# HiPeRTA : R0 -> DL3



# HiPeRTA : R0 -> DL3



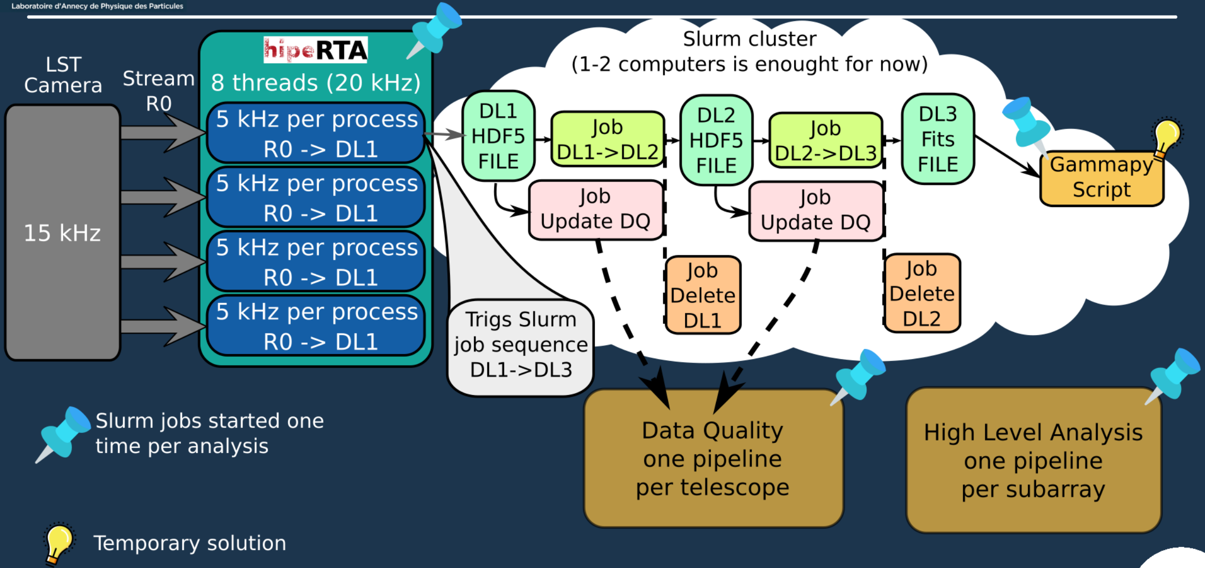
# HiPeRTA : R0 -> DL3



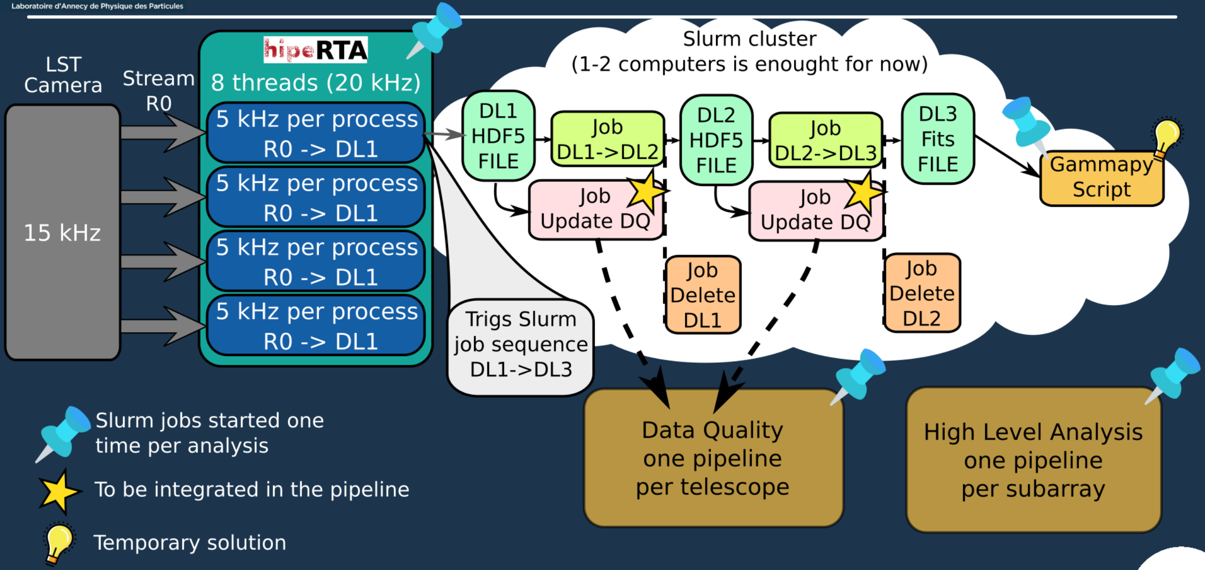

Slurm jobs started one time per analysis

Temporary solution

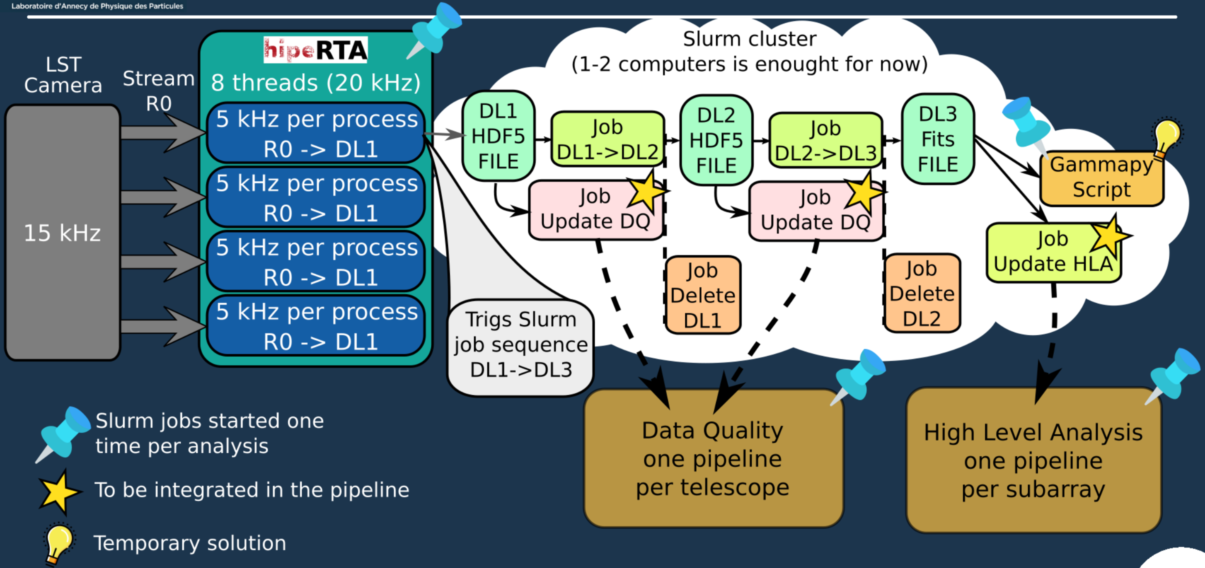
# HiPeRTA : R0 -> DL3



# HiPeRTA : R0 -> DL3



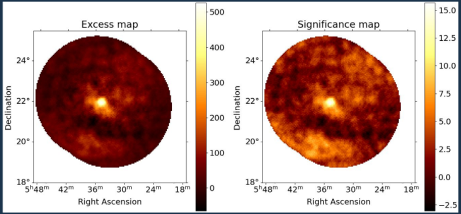
# HiPeRTA : R0 -> DL3



# Detections

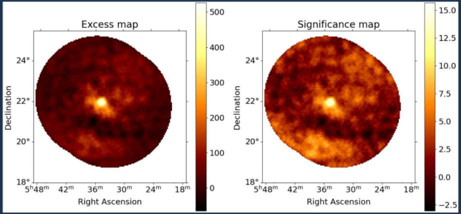
# Detections

## Crab Nebula

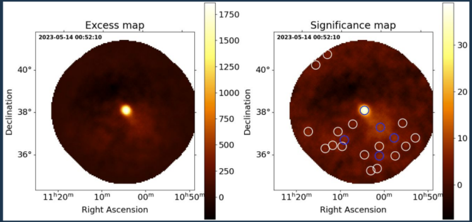


# Detections

## Crab Nebula

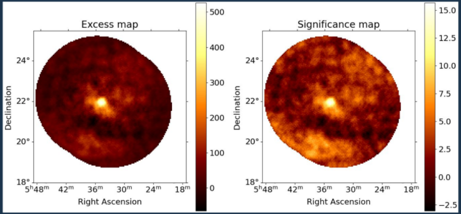



## Mkn 421

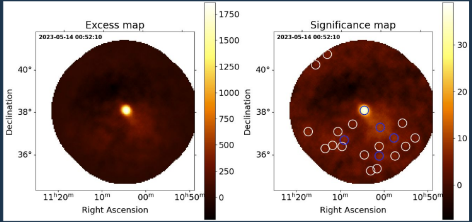


# Detections

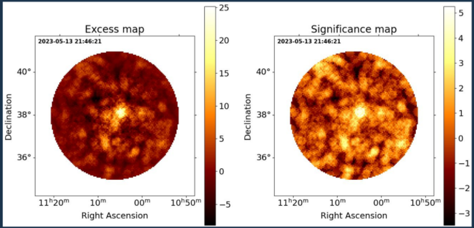
## Crab Nebula




## Mkn 421

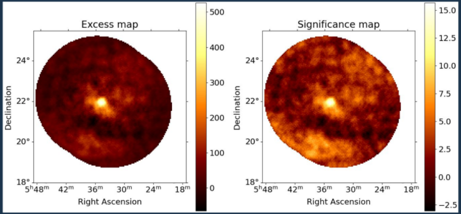



## Mkn 421 - 4 min

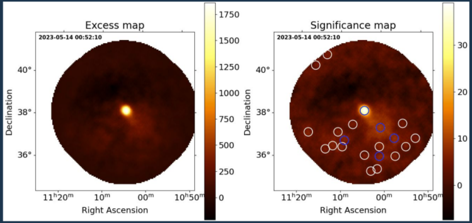


# Detections

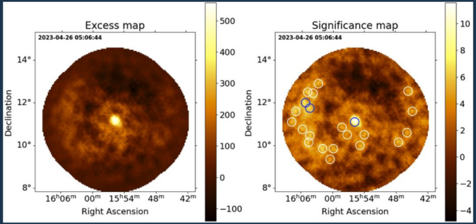
## Crab Nebula



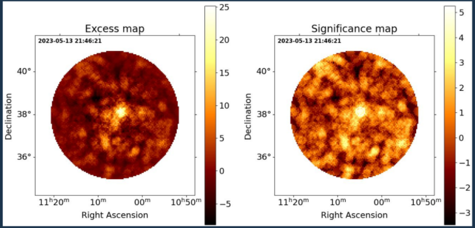

## Mkn 421




## AGN

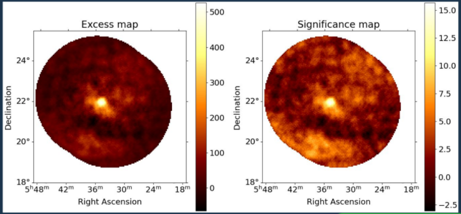



## Mkn 421 - 4 min

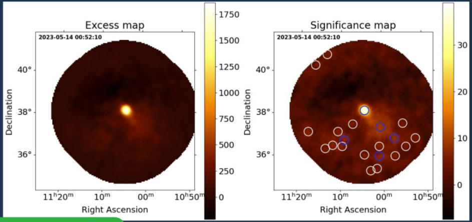


# Detections

## Crab Nebula

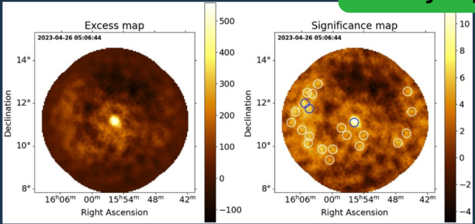



## Mkn 421

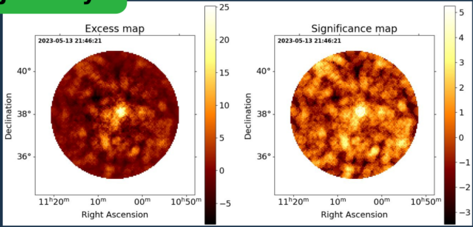



**Started automatically almost every night since january**

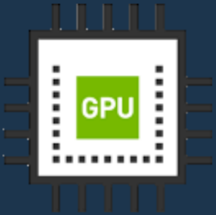
## AGN



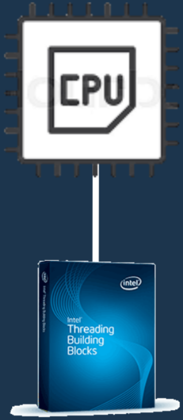

## Mkn 421 - 4 min

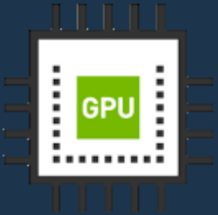


# Standard C++ for CPU and GPU

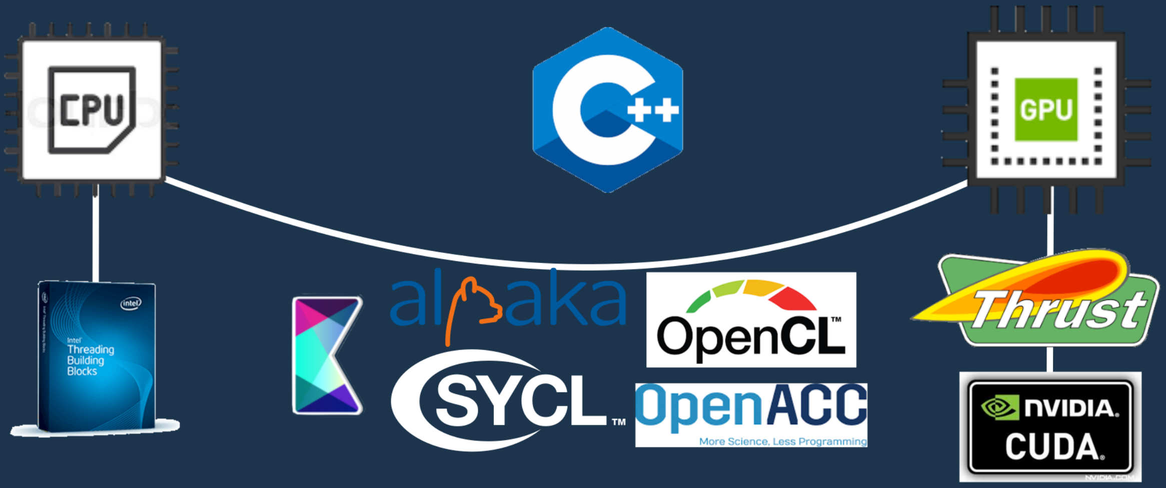
# Standard C++ for CPU and GPU



# Standard C++ for CPU and GPU

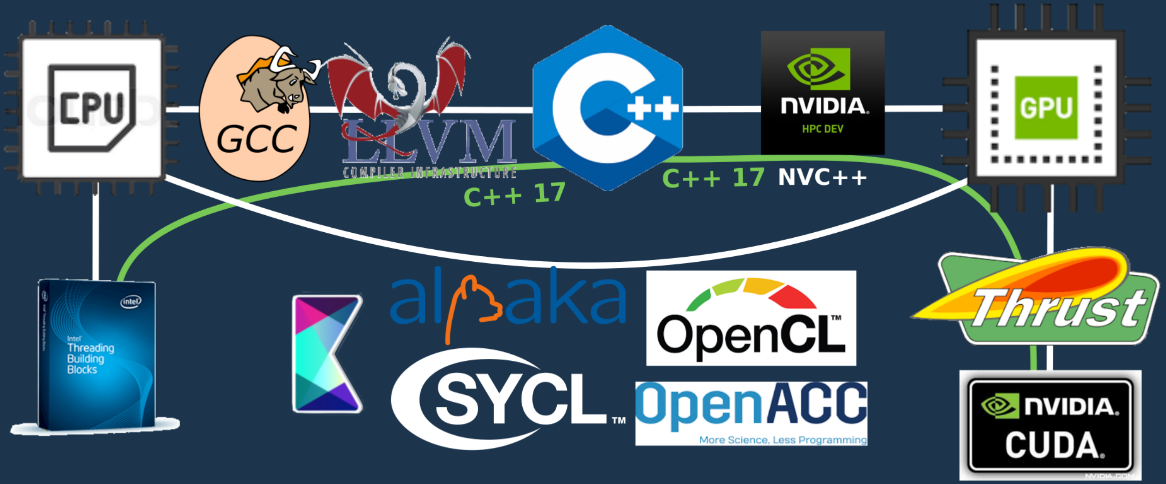
# Standard C++ for CPU and GPU



# Standard C++ for CPU and GPU

# Standard C++ for CPU and GPU

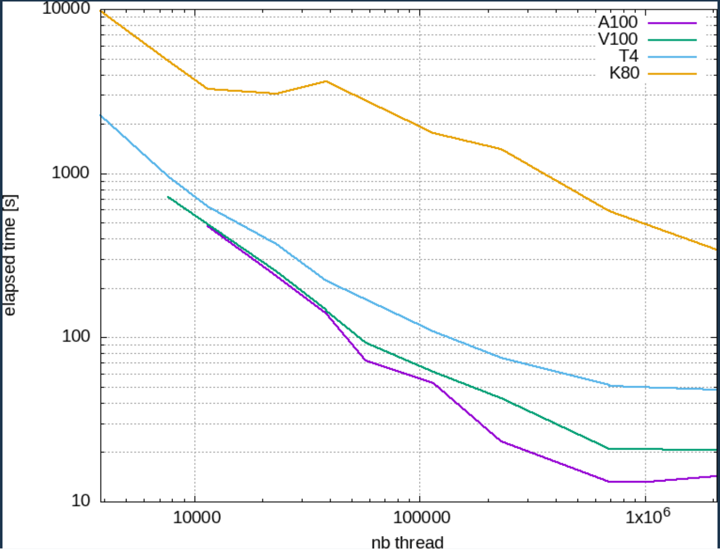
# Standard C++ for CPU and GPU



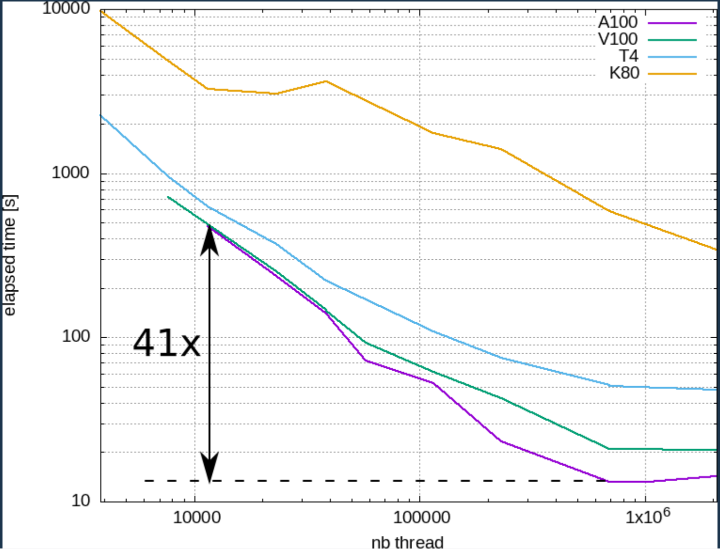
# How many threads ?

Laboratoire d'Annecy de Physique des Particules

# How many threads ?



# How many threads ?



# Express Parallelism

# Express Parallelism

Event

# Express Parallelism

Event

Independent Events => Independent Computing => Parallelism

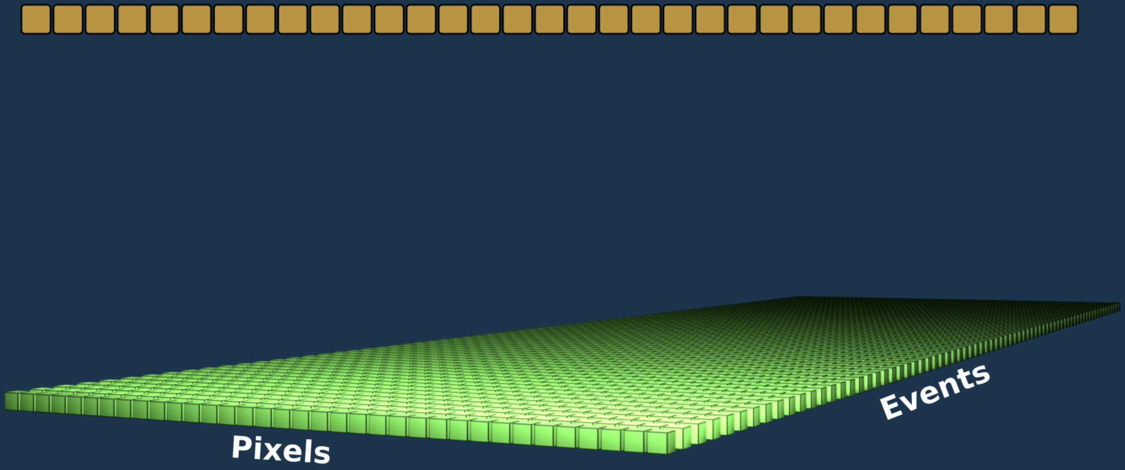
# Express Parallelism

Event

Independent Events => Independent Computing => Parallelism

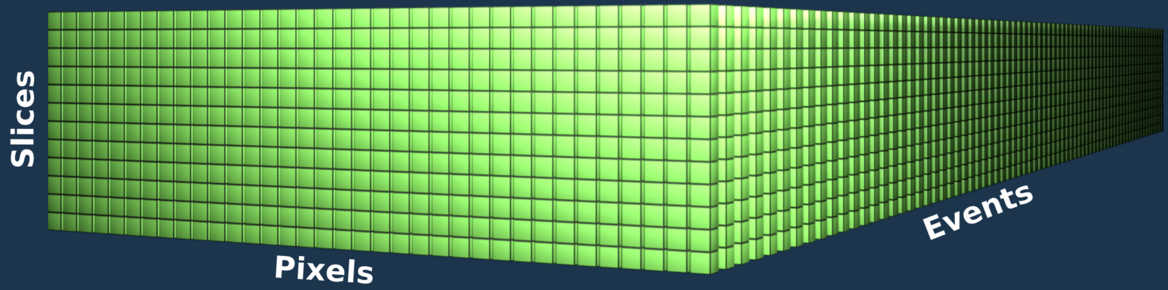
# Express Parallelism

Event Independent Events => Independent Computing => Parallelism



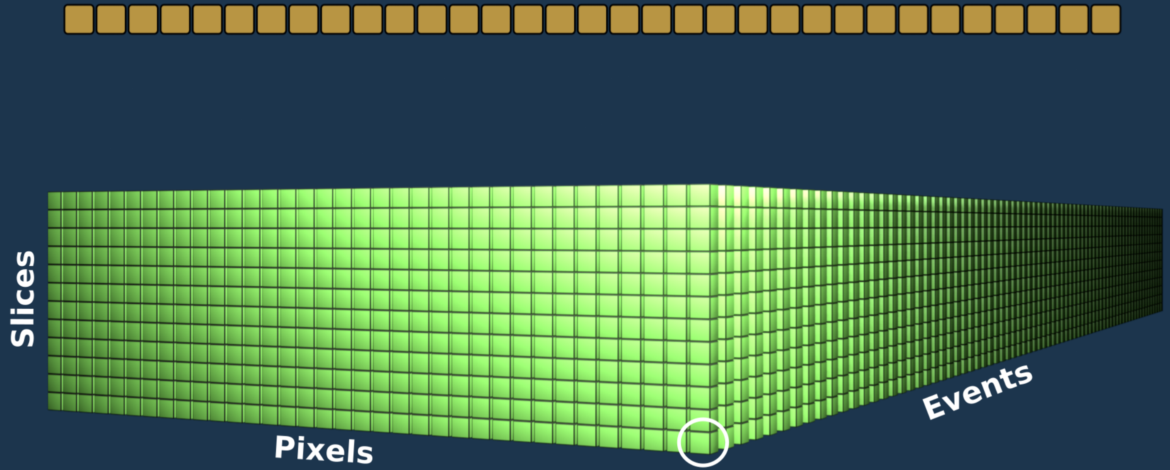
## Express Parallelism

Independent Events => Independent Computing => Parallelism

# Express Parallelism

Event Independent Events => Independent Computing => Parallelism

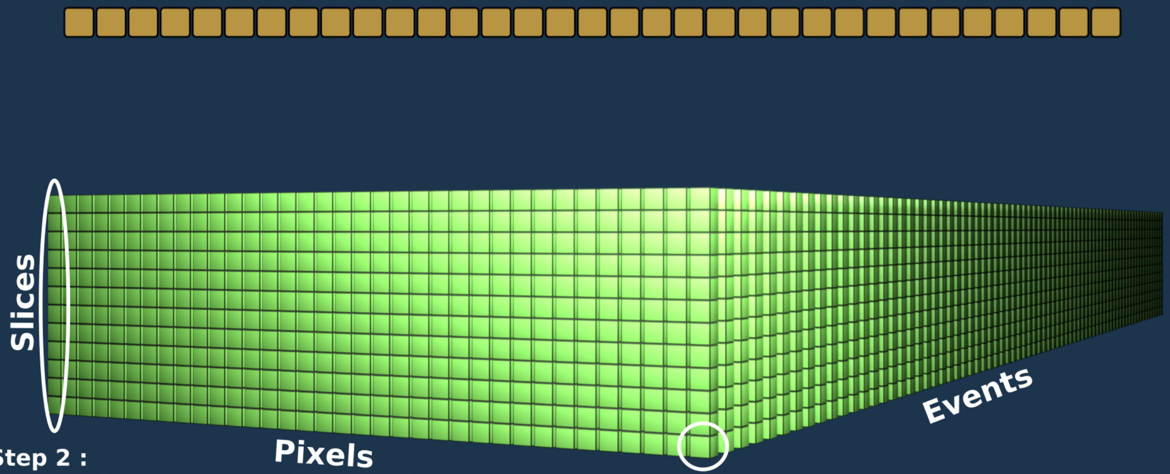



**Step 1 : Independent** Computing in Calibration

# Express Parallelism

Event

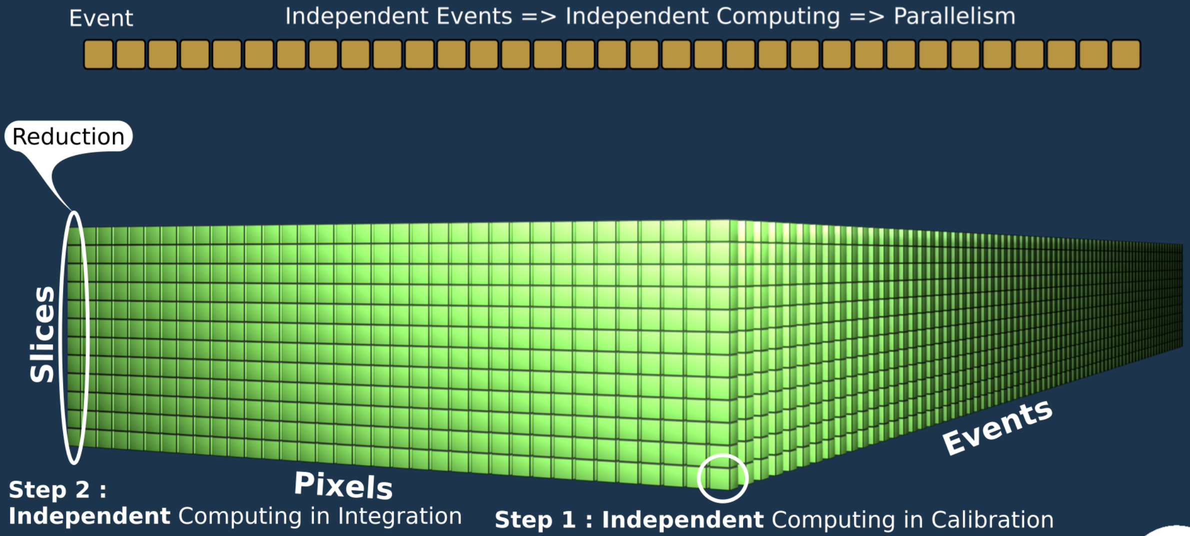
Independent Events => Independent Computing => Parallelism




**Step 2 :**
**Independent** Computing in Integration

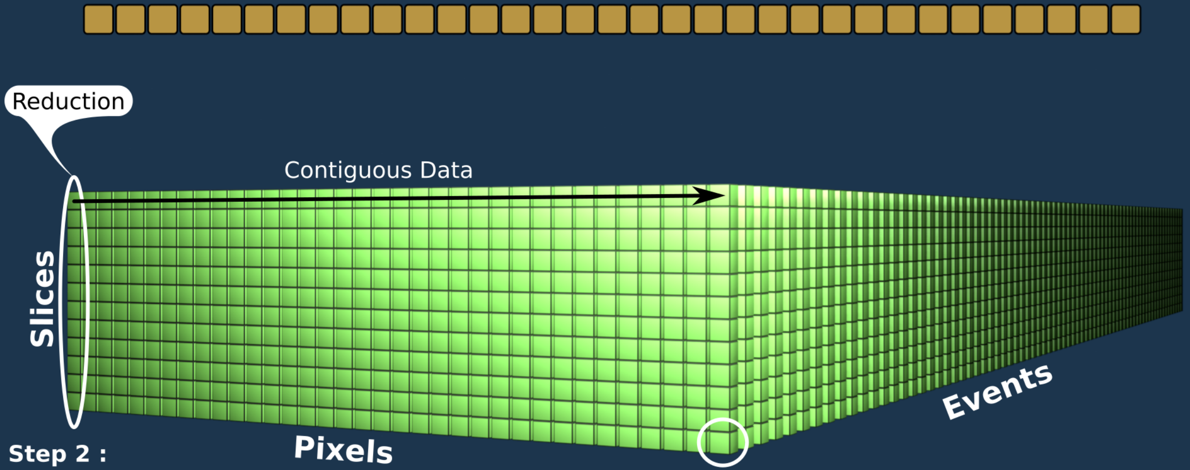
**Step 1 : Independent** Computing in Calibration

# Express Parallelism



# Express Parallelism

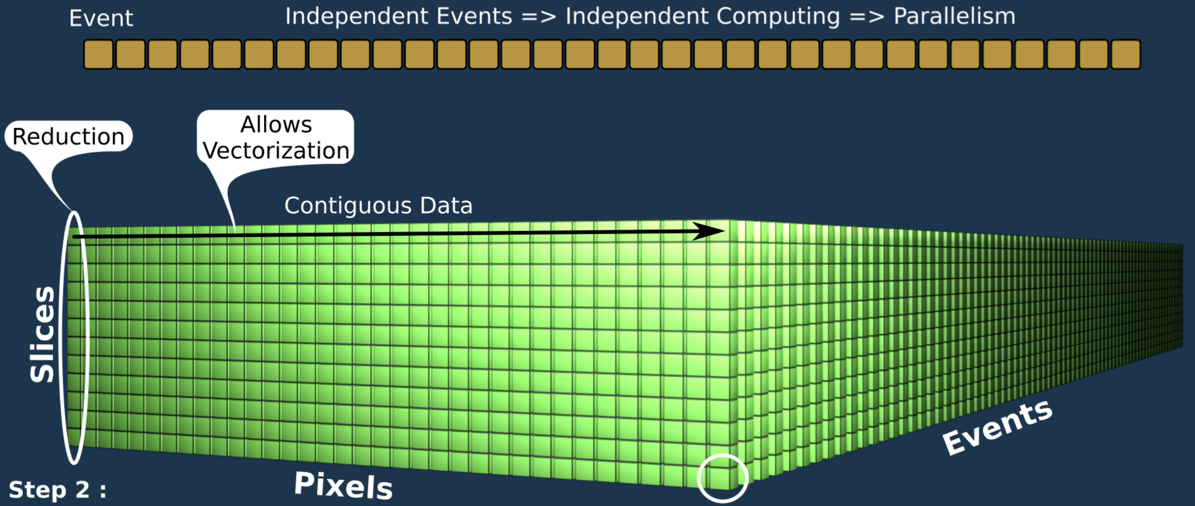
Event Independent Events => Independent Computing => Parallelism




**Step 2 : Independent** Computing in Integration

**Step 1 : Independent** Computing in Calibration

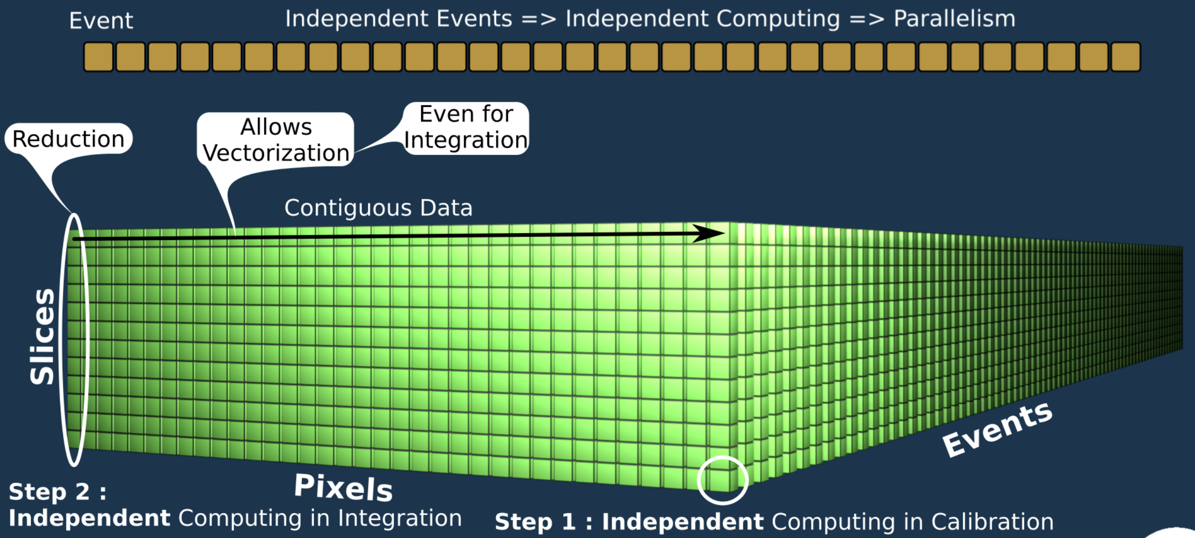
# Express Parallelism




**Step 2 : Independent** Computing in Integration

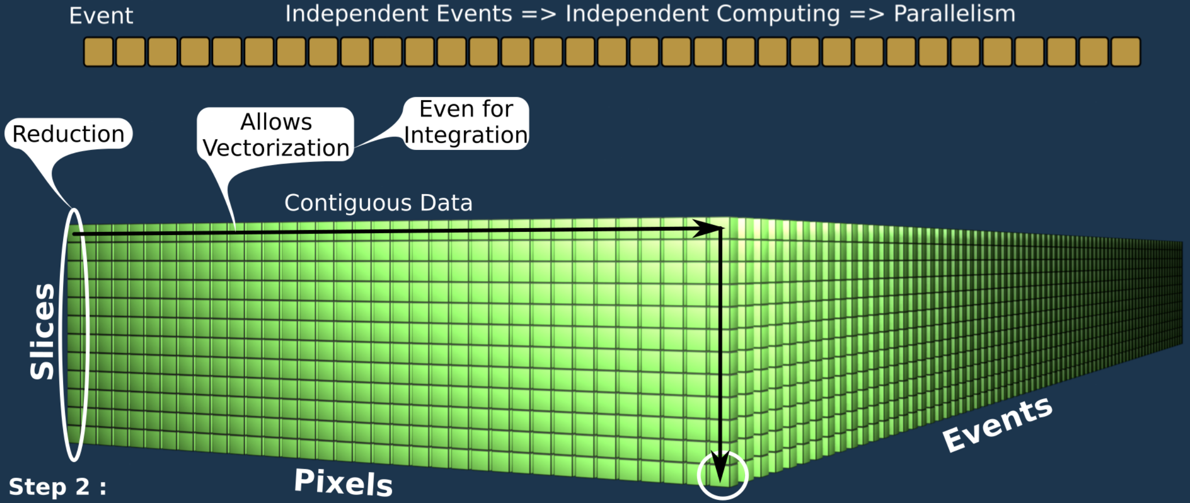
**Step 1 : Independent** Computing in Calibration

# Express Parallelism



# Express Parallelism

Independent Events => Independent Computing => Parallelism

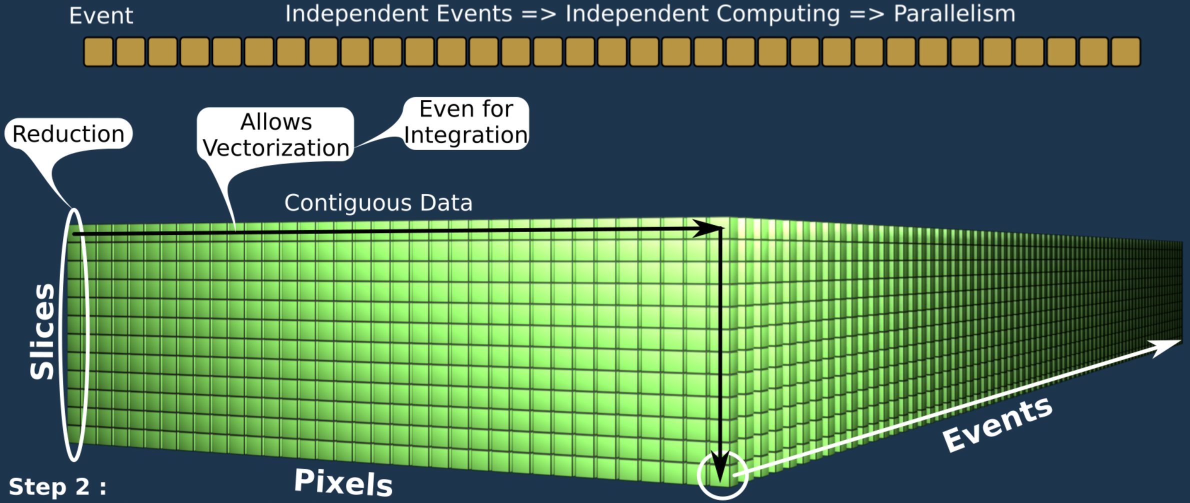



**Step 2 :**
**Independent** Computing in Integration

**Step 1 : Independent** Computing in Calibration

# Express Parallelism

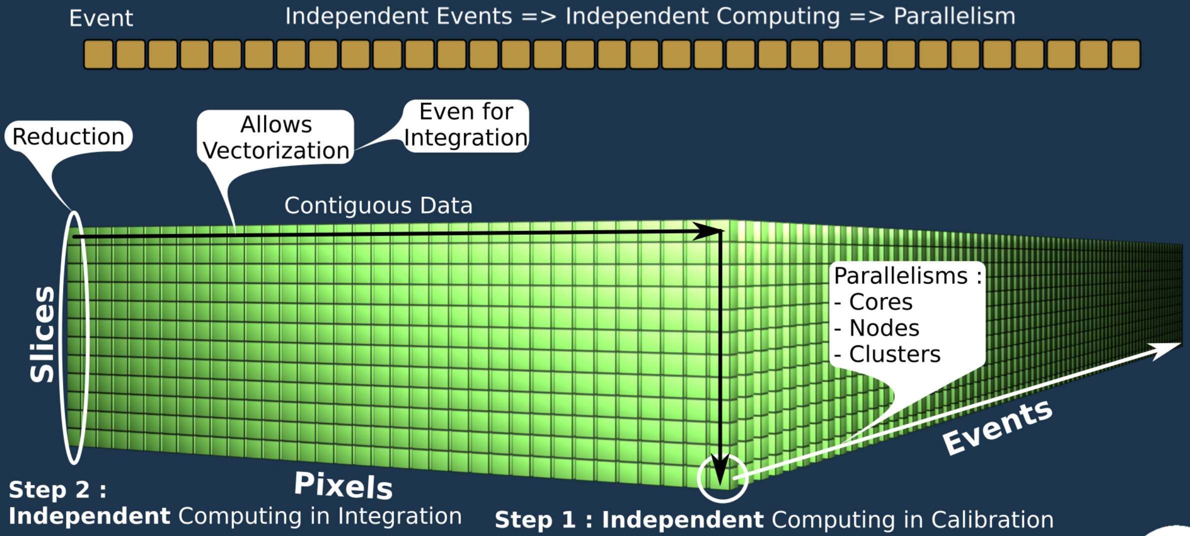
Independent Events => Independent Computing => Parallelism



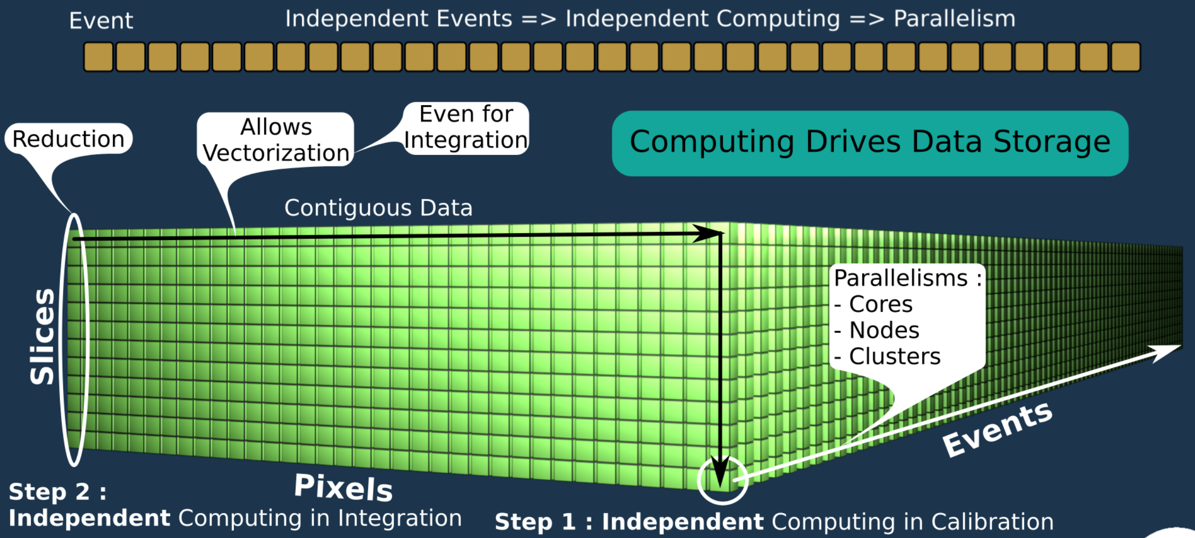

**Step 2 :**
**Independent** Computing in Integration

**Step 1 : Independent** Computing in Calibration

# Express Parallelism



# Express Parallelism



# HiPeRTA on GPU with CUDA

## Express Global Computation -> Linear Algebra

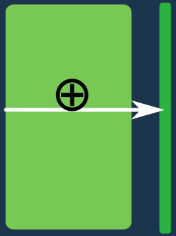
## Express Global Computation -> Linear Algebra

### Calibration -> Broadcast

## Express Global Computation -> Linear Algebra

**Calibration -> Broadcast**

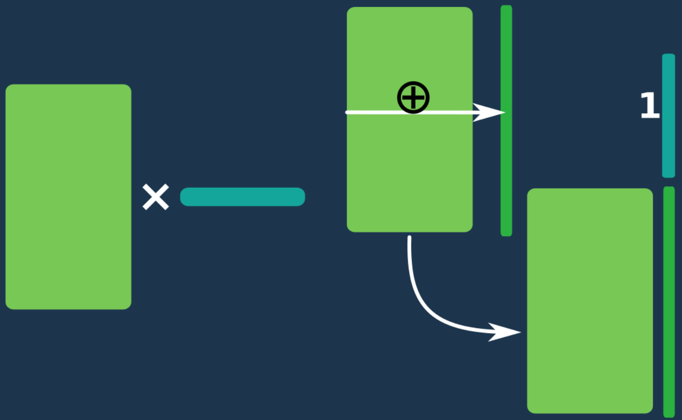
**Signal Reduction -> SGEMV**



## Express Global Computation -> Linear Algebra

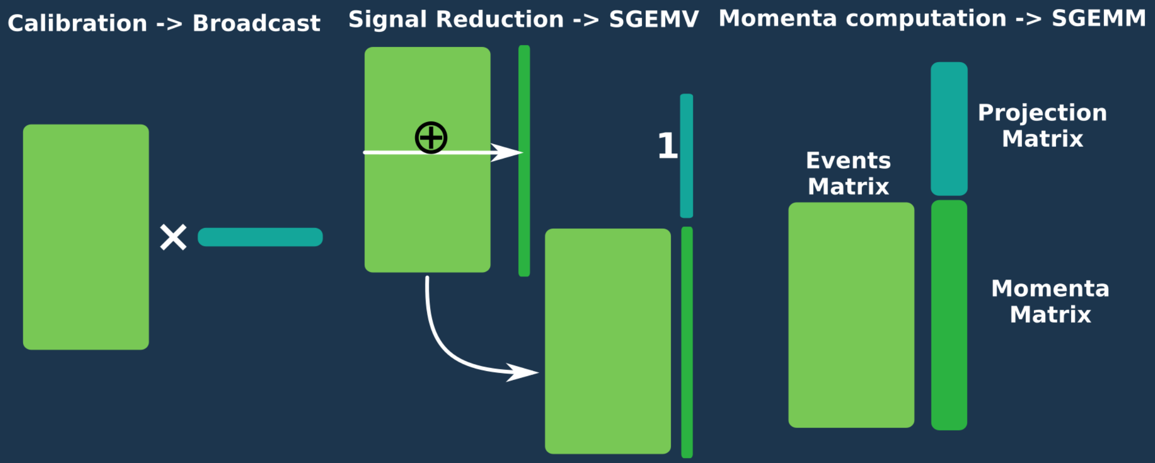
### Calibration -> Broadcast

### Signal Reduction -> SGEMV



# HiPeRTA on GPU with CUDA

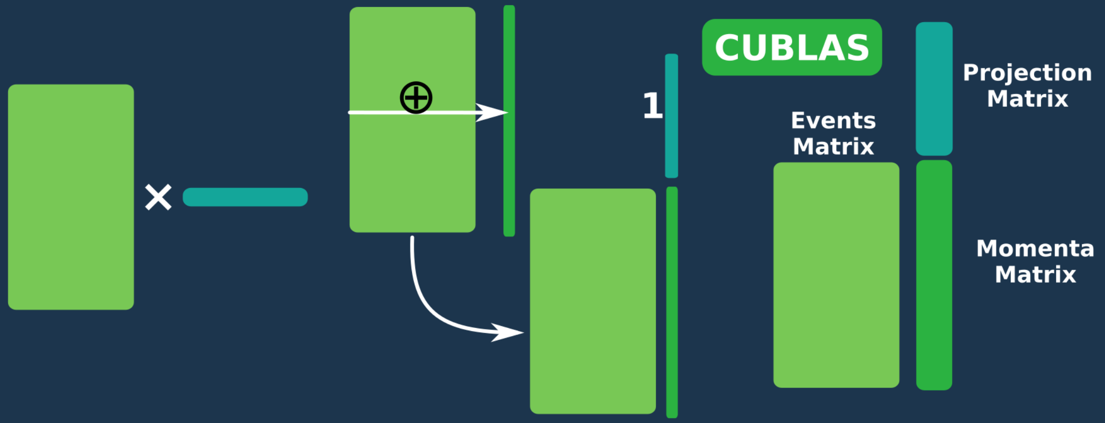
## Express Global Computation -> Linear Algebra



# HiPeRTA on GPU with CUDA

**Express Global Computation -> Linear Algebra**

**Calibration -> Broadcast** | **Signal Reduction -> SGEMV** | **Momenta computation -> SGEMM**



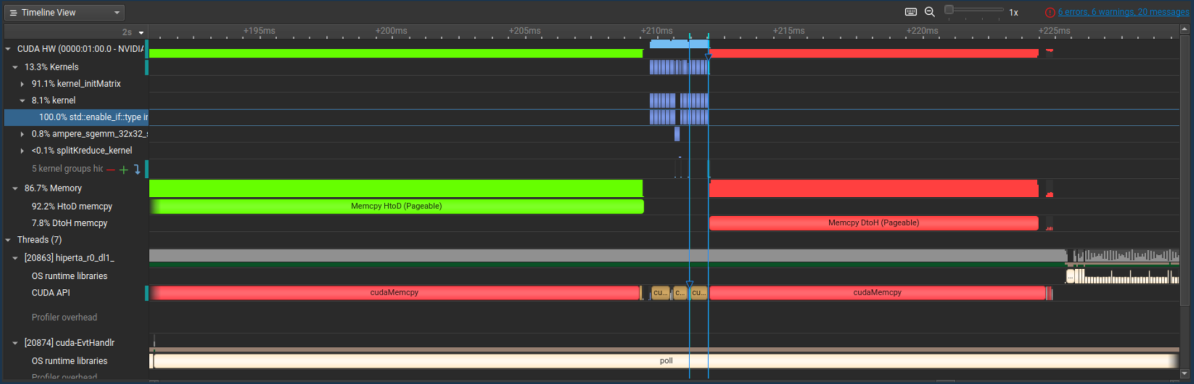
# NSight Profiling : HiPeRTA CUDA

**A3000, 12 GB** DRAM, **2048** cores

**6243 Events** **93 %** of DRAM

**CUDA Kernels** : **13.3 %**

**Memory** : **86.7 %**



# NSight Profiling : HiPeRTA CUDA

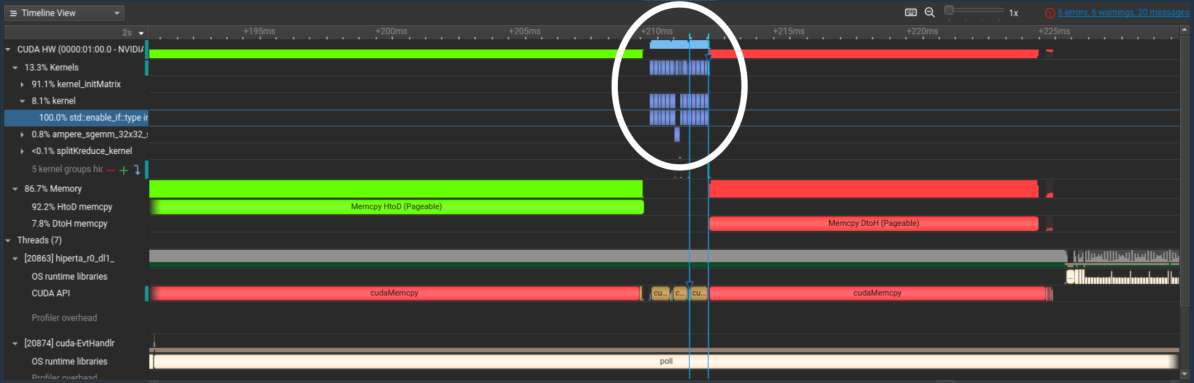
**A3000**, **12 GB** DRAM, **2048** cores

**6243 Events** **93** % of DRAM

**CUDA Kernels** ; **13.3 %**

**Memory** : **86.7 %**

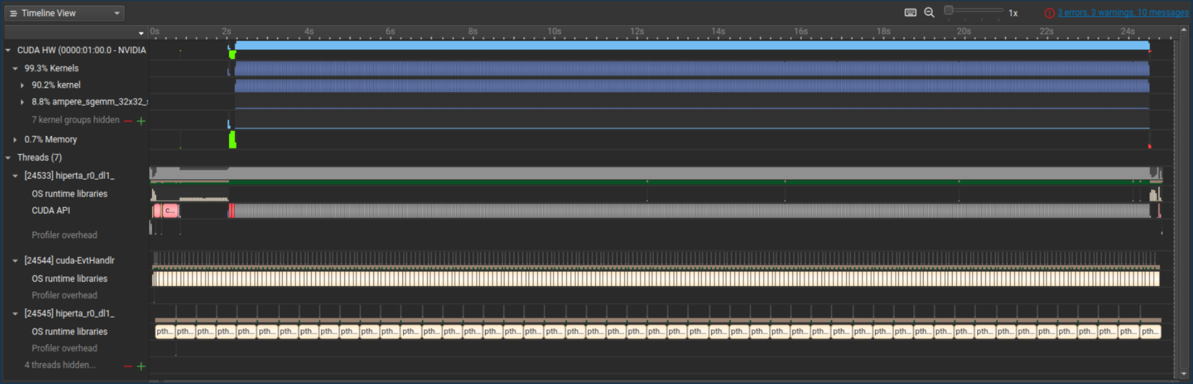
**~3ms**



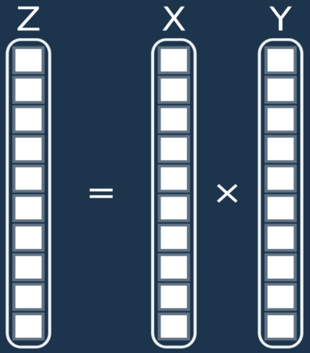
# NSight Profiling : HiPeRTA CUDA

## A3000, 12 GB DRAM, 2048 cores

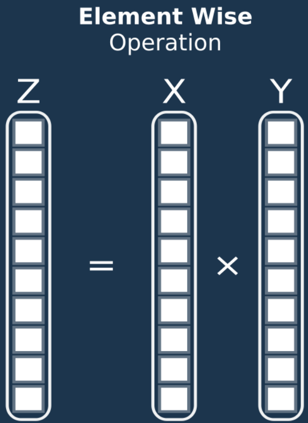
### 6243 Events x10 000



# Example : Hadamard Product



# Example : Hadamard Product

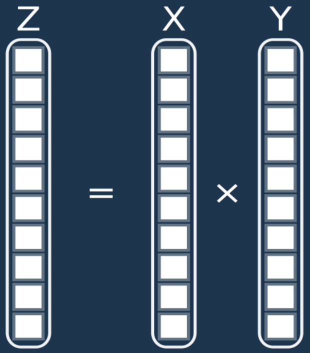


# Example : Hadamard Product

## C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabResult[i] = tabX[i]*tabY[i];
}
```

**Element Wise**
Operation

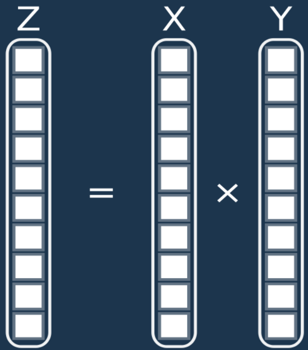


# Example : Hadamard Product

## C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabResult[i] = tabX[i]*tabY[i];
}
```

Explicit order **not** necessary



# Example : Hadamard Product

## C++

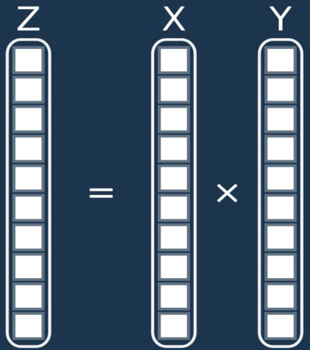
```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabResult[i] = tabX[i]*tabY[i];
}
```

Explicit order **not** necessary

## C++ Algorithm : std::transform

```
std::transform(std::begin(tabX), std::end(tabX),
		std::begin(tabY), std::begin(tabRes),
		[](float xi, float yi){ return xi * yi; });
```

**Element Wise**
Operation



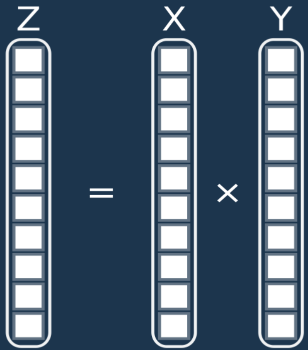
# Example : Hadamard Product

## C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabResult[i] = tabX[i]*tabY[i];
}
```

Explicit order **not** necessary

**Element Wise** Operation




## C++ Algorithm : std::transform

```
std::transform(std::begin(tabX), std::end(tabX),
	std::begin(tabY), std::begin(tabRes),
	[](float xi, float yi){ return xi * yi; });
```

## C++ 17 / C++ 20

```
std::transform(std::execution::par_unseq,
	std::begin(tabX), std::end(tabX),
	std::begin(tabY), std::begin(tabRes),
	[](float xi, float yi){ return xi * yi; });
```

LAPP
Laboratoire d'Annecy de Physique des Particules

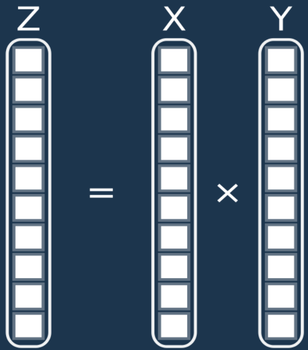
# Example : Hadamard Product

## C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
»       tabResult[i] = tabX[i]*tabY[i];
}
```

Explicit order **not** necessary

**Element Wise** Operation




## C++ Algorithm : std::transform

```
std::transform(std::begin(tabX), std::end(tabX),
»       std::begin(tabY), std::begin(tabRes),
»       [](float xi, float yi){ return xi * yi; });
```

## C++ 17 / C++ 20

**Execution Policy**

```
std::transform(std::execution::par_unseq,
»       std::begin(tabX), std::end(tabX),
»       std::begin(tabY), std::begin(tabRes),
»       [](float xi, float yi){ return xi * yi; });
```

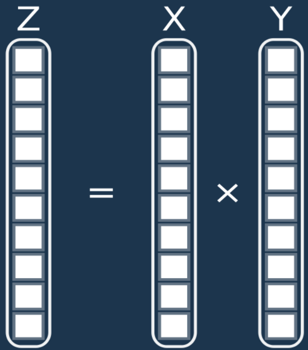
# Example : Hadamard Product

## C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
        tabResult[i] = tabX[i]*tabY[i];
}
```

Explicit order **not** necessary

**Element Wise** Operation




## C++ Algorithm : std::transform

```
std::transform(std::begin(tabX), std::end(tabX),
        std::begin(tabY), std::begin(tabRes),
        [](float xi, float yi){ return xi * yi; });
```

## C++ 17 / C++ 20

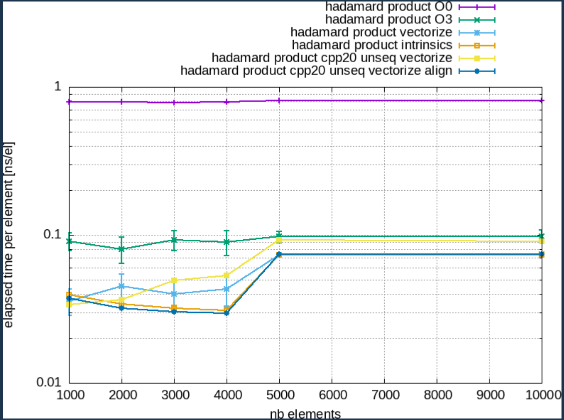
**Execution Policy**

```
std::transform(std::execution::par_unseq,
        std::begin(tabX), std::end(tabX),
        std::begin(tabY), std::begin(tabRes),
        [](float xi, float yi){ return xi * yi; });
```

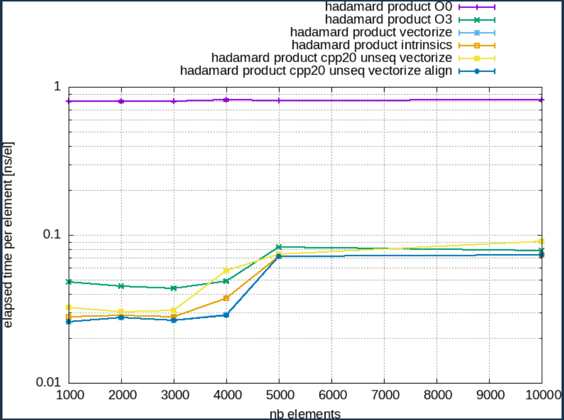
- **seq**
- **unseq**
- **par**
- **par_unseq**

# The Hadamard product : Performances

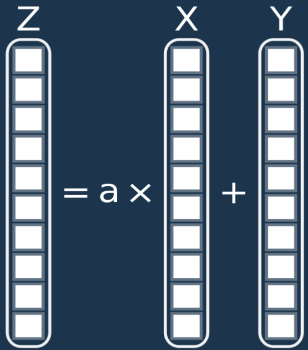
## G++ 11



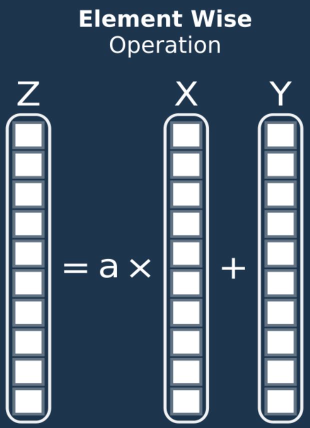

## Clang++ 14



# Example : Saxpy



# Example : Saxpy

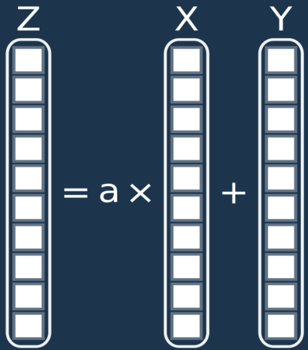


# Example : Saxpy

## C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
»       tabResult[i] = a*tabX[i] + tabY[i];
}
```

**Element Wise**
Operation

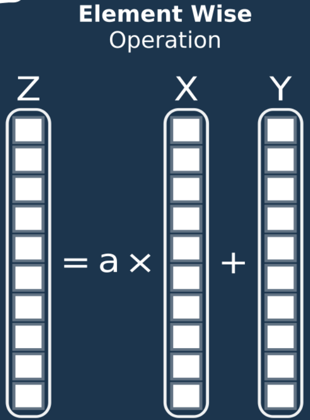


# Example : Saxpy

## C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabResult[i] = a*tabX[i] + tabY[i];
}
```

Explicit order **not** necessary



# Example : Saxpy

## C++

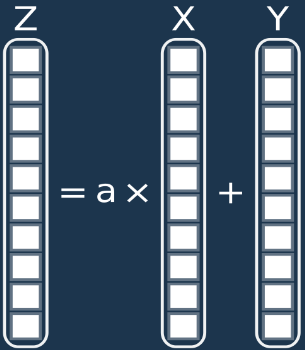
```
for(long unsigned int i(0lu); i < nbElement; ++i){
		tabResult[i] = a*tabX[i] + tabY[i];
}
```

Explicit order **not** necessary

## C++ Algorithm : std::transform

```
std::transform(std::begin(tabX), std::end(tabX),
		std::begin(tabY), std::begin(tabResult),
		[=](float xi, float yi){ return a*xi + yi; });
```

**Element Wise** Operation



# Example : Saxpy

## C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabResult[i] = a*tabX[i] + tabY[i];
}
```

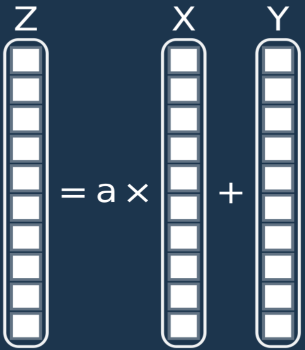
Explicit order **not** necessary

**Element Wise** Operation

## C++ Algorithm : std::transform

Catches Extra variables by copy

```
std::transform(std::begin(tabX), std::end(tabX),
	std::begin(tabY), std::begin(tabResult),
	[=](float xi, float yi){ return a*xi + yi; });
```



# Example : Saxpy

## C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
»       tabResult[i] = a*tabX[i] + tabY[i];
}
```

Explicit order **not** necessary

## C++ Algorithm : std::transform

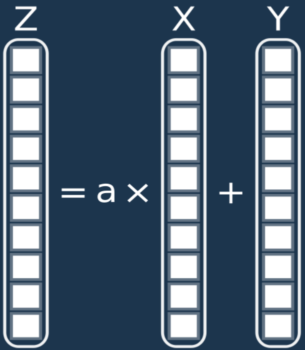
Catches Extra variables by copy

```
std::transform(std::begin(tabX), std::end(tabX),
»       std::begin(tabY), std::begin(tabResult),
»       [=](float xi, float yi){ return a*xi + yi; });
```

## C++ 17 / C++ 20

```
std::transform(std::execution::par_unseq,
»       std::begin(tabX), std::end(tabX),
»       std::begin(tabY), std::begin(tabResult),
»       [=](float xi, float yi){ return a*xi + yi; });
```

**Element Wise**
Operation



# Example : Saxpy

## C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
»       tabResult[i] = a*tabX[i] + tabY[i];
}
```

Explicit order **not** necessary

## C++ Algorithm : std::transform

Catches Extra variables by copy

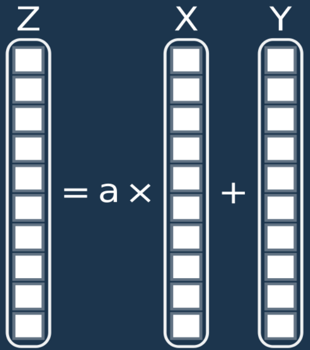
```
std::transform(std::begin(tabX), std::end(tabX),
»       std::begin(tabY), std::begin(tabResult),
»       [=](float xi, float yi){ return a*xi + yi; });
```

## C++ 17 / C++ 20

**Execution Policy**

```
std::transform(std::execution::par_unseq,
»       std::begin(tabX), std::end(tabX),
»       std::begin(tabY), std::begin(tabResult),
»       [=](float xi, float yi){ return a*xi + yi; });
```

**Element Wise** Operation



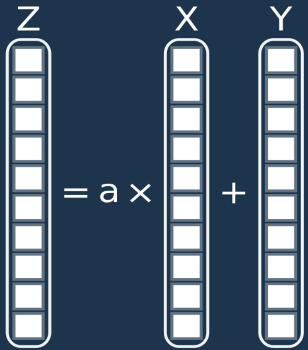
# Example : Saxpy

## C++

```
for(long unsigned int i(0lu); i < nbElement; ++i){
»	tabResult[i] = a*tabX[i] + tabY[i];
}
```

Explicit order **not** necessary

**Element Wise** Operation




## C++ Algorithm : std::transform

Catches Extra variables by copy

```
std::transform(std::begin(tabX), std::end(tabX),
»	std::begin(tabY), std::begin(tabResult),
»	[=](float xi, float yi){ return a*xi + yi; });
```

## C++ 17 / C++ 20

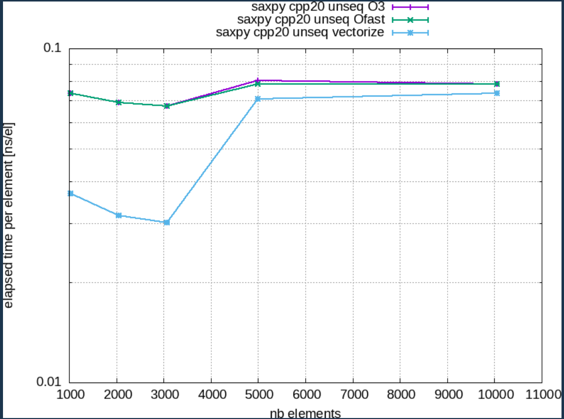
**Execution Policy**

- **seq**
- **unseq**
- **par**
- **par_unseq**

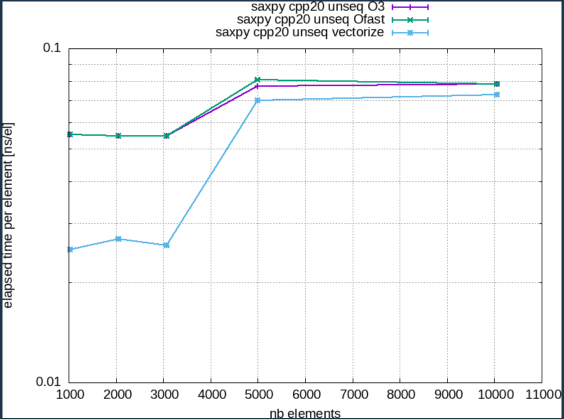
```
std::transform(std::execution::par_unseq,
»	std::begin(tabX), std::end(tabX),
»	std::begin(tabY), std::begin(tabResult),
»	[=](float xi, float yi){ return a*xi + yi; });
```

# Saxpy : Performances

G++ 11




CLang++ 14



# Example : Reduction

# Example : Reduction

## C++

```
float res(0.0f);
for(long unsigned int i(0lu); i < nbElement; ++i){
	res += tabValue[i];
}
return res;
```

# Example : Reduction

**C++**

```
float res(0.0f);
for(long unsigned int i(0lu); i < nbElement; ++i){
»       res += tabValue[i];
}
return res;
```

Explicit order **not** necessary every time

# Example : Reduction

## C++

```
float res(0.0f);
for(long unsigned int i(0lu); i < nbElement; ++i){
»       res += tabValue[i];
}
return res;
```

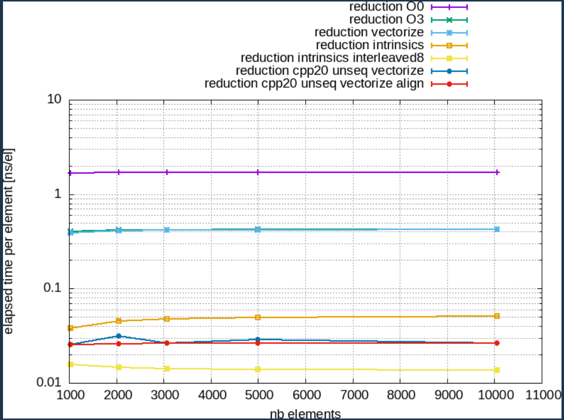
Explicit order **not** necessary every time


## C++ 17 / C++ 20

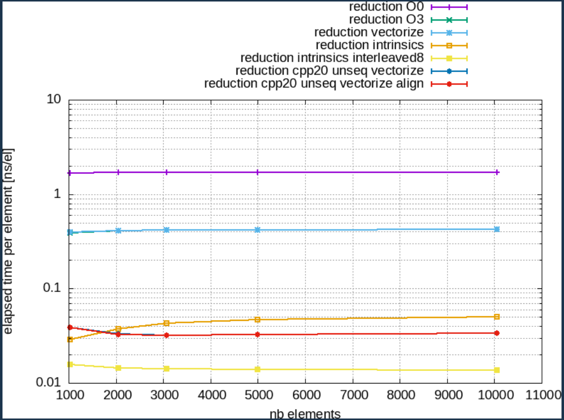
```
return std::reduce(std::execution::par_unseq,
»       std::begin(vecX), std::end(vecX),
»       0.0f, std::plus{});
```

# Reduction : Performances

## G++ 11




## Clang++ 14



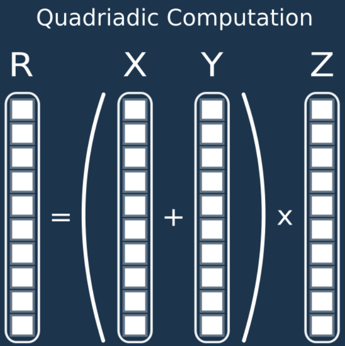
# std::transform : triadic

# std::transform : triadic

Triadic : **z = x + y**

# std::transform : triadic

Triadic : **z = x + y**



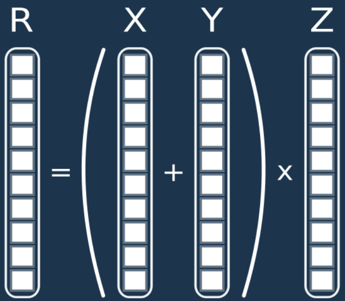
# std::transform : triadic

Triadic : **z = x + y**

**Classic C++**

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```

Quadriadic Computation



# std::transform : triadic

Triadic : **z = x + y**

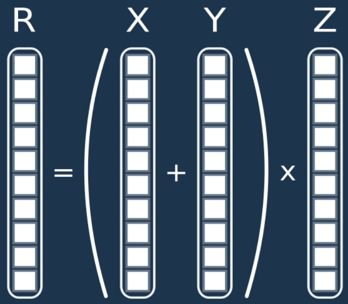
**Classic C++**

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```

**C++ 17 / 20 / 23**

```
std::transform(std::execution::par_unseq,
	std::begin(vecIndex), std::end(vecIndex),
	std::begin(vecX), std::begin(vecRes),
	[=](int i, float x){
		return (x + vecY[i]) * vecZ[i];
	});
```

Quadriadic Computation



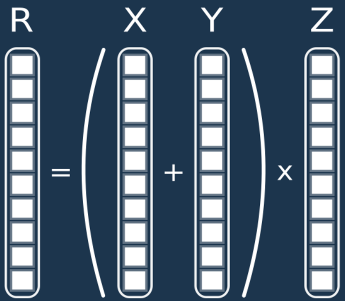
# std::transform : triadic

Triadic : **z = x + y**

**Classic C++**

```
for(long unsigned int i(0lu); i < nbElement; ++i){
		tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```

Quadriadic Computation




**C++ 17 / 20 / 23**

**vecY**, **vecZ** have to be **std::vector**

```
std::transform(std::execution::par_unseq,
		std::begin(vecIndex), std::end(vecIndex),
		std::begin(vecX), std::begin(vecRes),
		[=](int i, float x){
			return (x + vecY[i]) * vecZ[i];
		});
```

# std::transform : triadic

Triadic : **z = x + y**

**Classic C++**

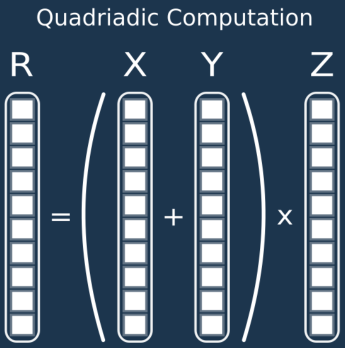
```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```

**C++ 17 / 20 / 23**

**vecY**, **vecZ** have to be **std::vector**

```
std::transform(std::execution::par_unseq,
	std::begin(vecIndex), std::end(vecIndex),
	std::begin(vecX), std::begin(vecRes),
	[=](int i, float x){
		return (x + vecY[i]) * vecZ[i];
	});
```

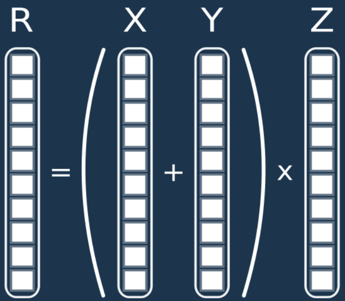
Fully Vectorized



# std::transform : triadic

Triadic : **z = x + y**

Quadriadic Computation




**Classic C++**

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```

**C++ 17 / 20 / 23**

**vecY**, **vecZ** have to be **std::vector**

```
std::transform(std::execution::par_unseq,
	std::begin(vecIndex), std::end(vecIndex),
	std::begin(vecX), std::begin(vecRes),
	[=](int i, float x){
		return (x + vecY[i]) * vecZ[i];
	});
```

Fully Vectorized

Needs extra index table

# std::transform : triadic

Triadic : **z = x + y**

**Classic C++**

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```

**C++ 17 / 20 / 23**

**vecY**, **vecZ** have to be **std::vector**

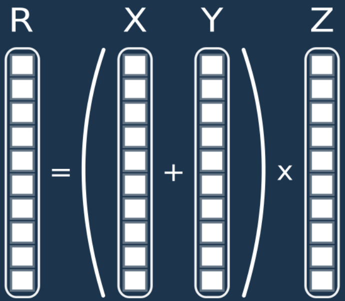
```
std::transform(std::execution::par_unseq,
	std::begin(vecIndex), std::end(vecIndex),
	std::begin(vecX), std::begin(vecRes),
	[=](int i, float x){
		return (x + vecY[i]) * vecZ[i];
	});
```

Fully Vectorized

Needs extra index table

Not vectorized with **std::for_each**

Quadriadic Computation



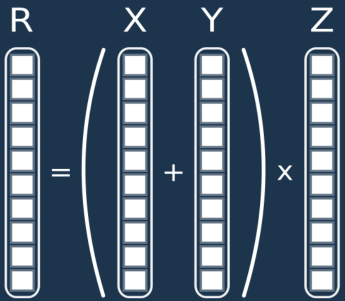
# std::transform : triadic

Triadic : **z = x + y**

**Classic C++**

```
for(long unsigned int i(0lu); i < nbElement; ++i){
	tabRes[i] = (tabX[i] + tabY[i])*tabZ[i];
}
```

Quadriadic Computation




**C++ 17 / 20 / 23**

**vecY**, **vecZ** have to be **std::vector**

```
std::transform(std::execution::par_unseq,
	std::begin(vecIndex), std::end(vecIndex),
	std::begin(vecX), std::begin(vecRes),
	[=](int i, float x){
		return (x + vecY[i]) * vecZ[i];
	});
```

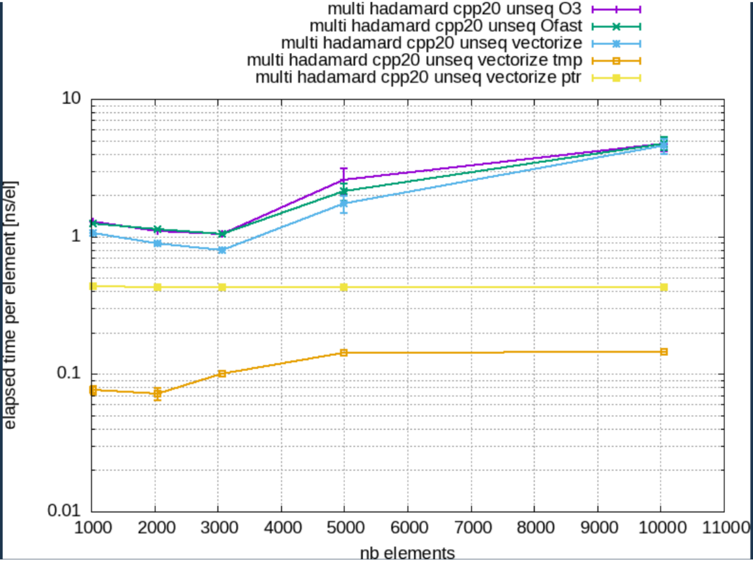
Fully Vectorized

Needs extra index table

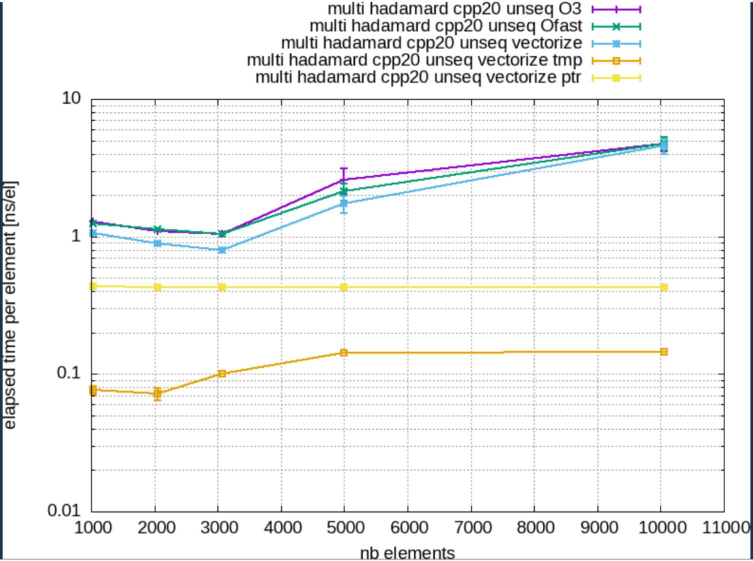
Not vectorized with **std::for_each**

Not vectorized with pointers **vecX**, **vecY**

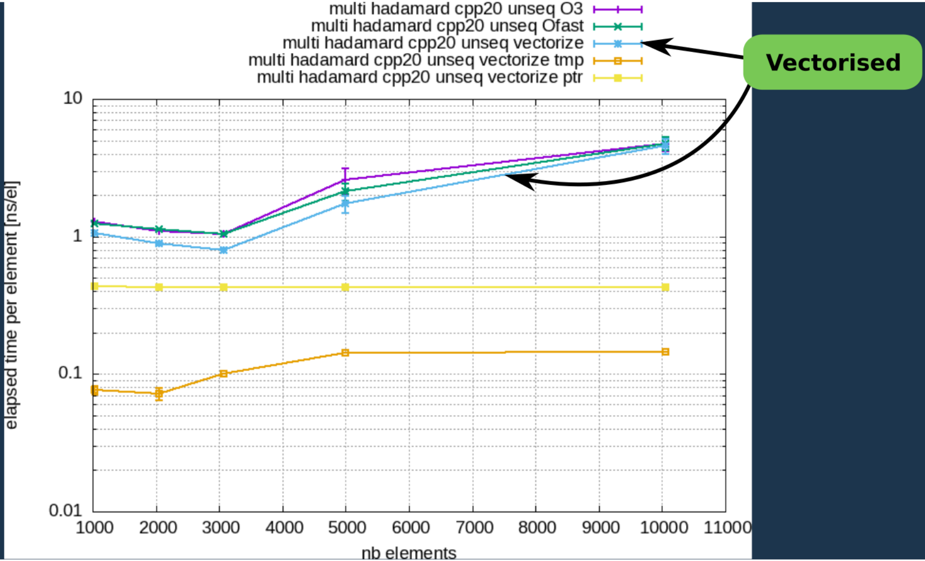
# std::transform : (X + Y) x Z



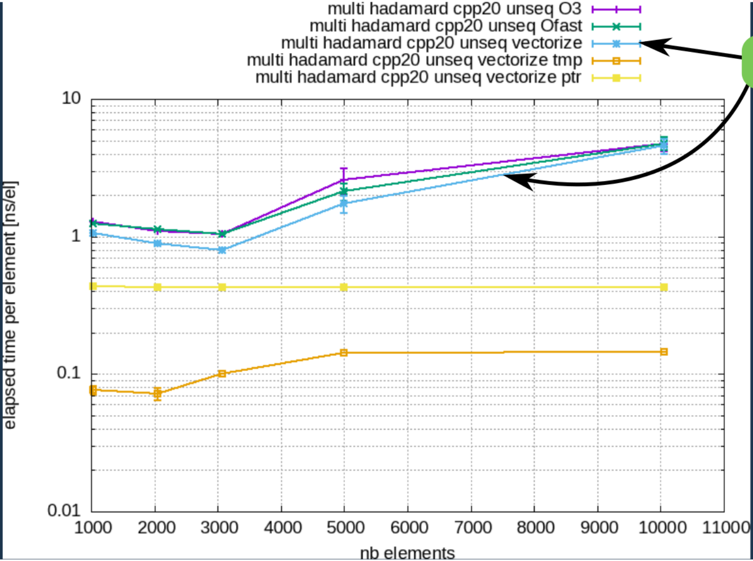
# std::transform : (X + Y) x Z



# std::transform : (X + Y) x Z



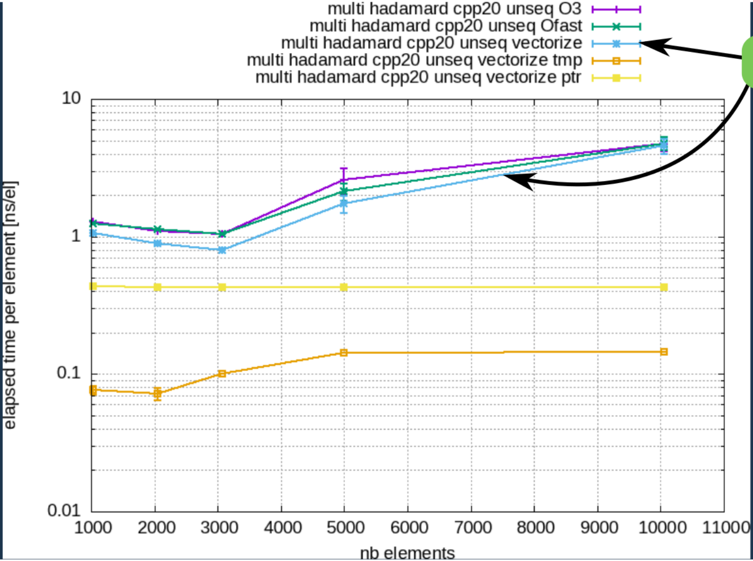
# std::transform : (X + Y) x Z




**Vectorised**

But too much **branching** and **data copy**

# std::transform : (X + Y) x Z

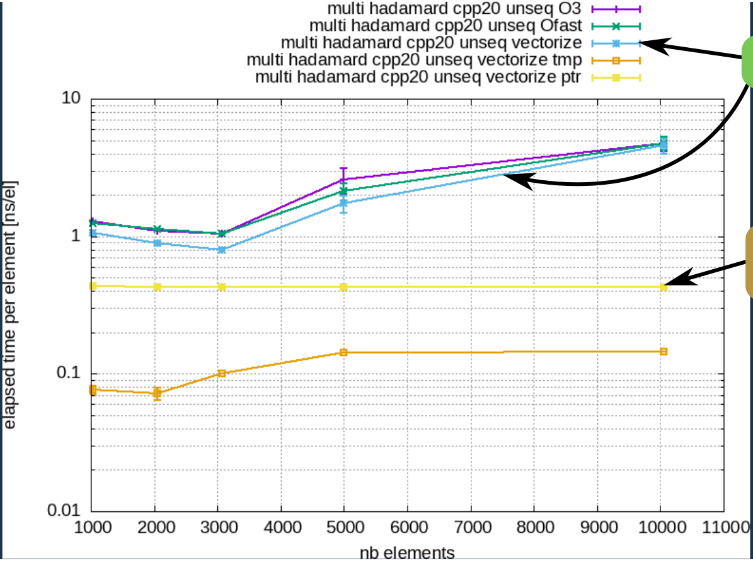



MAQAO

**Vectorised**

But too much **branching** and **data copy**

# std::transform : (X + Y) x Z

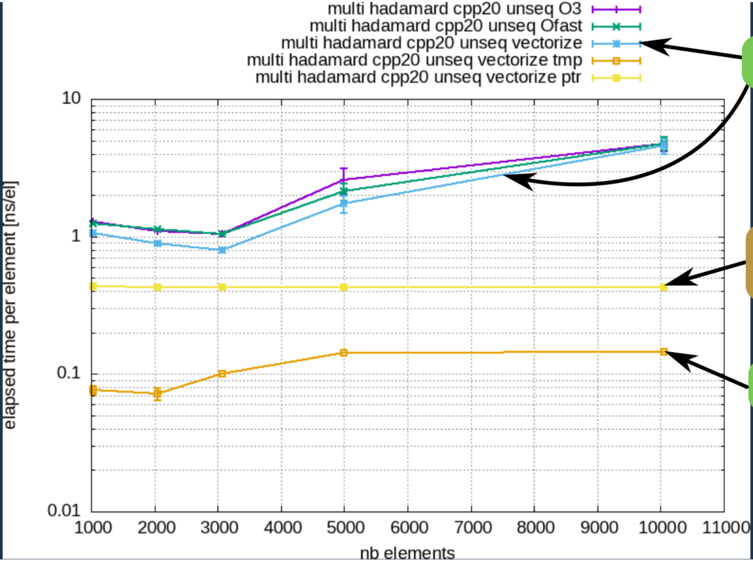



**Vectorised**

But too much **branching** and **data copy**

**Not Vectorized** but no **data copy**

# std::transform : (X + Y) x Z



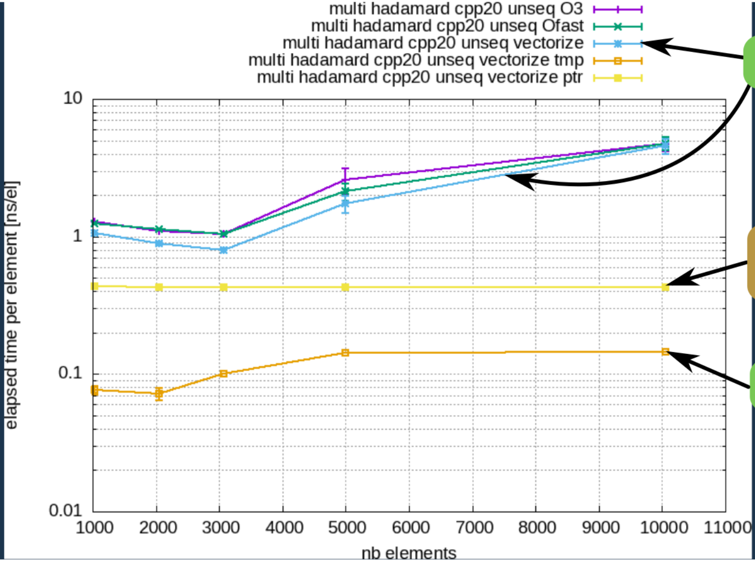

**Vectorised**

But too much **branching** and **data copy**

**Not Vectorized** but no **data copy**

**Vectorised**

# std::transform : (X + Y) x Z




**Vectorised**

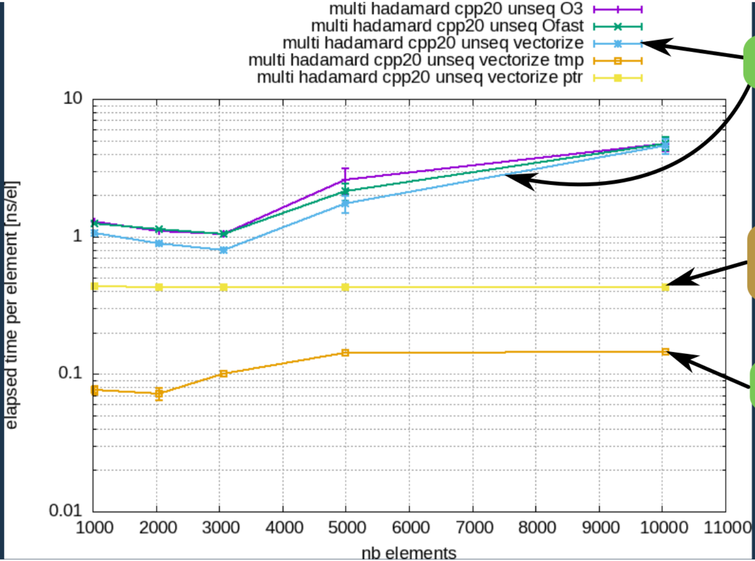
But too much **branching** and **data copy**

**Not Vectorized** but no **data copy**

**Vectorised**

But use **temporary vecXY**

# std::transform : (X + Y) x Z




**Vectorised**

But too much **branching** and **data copy**

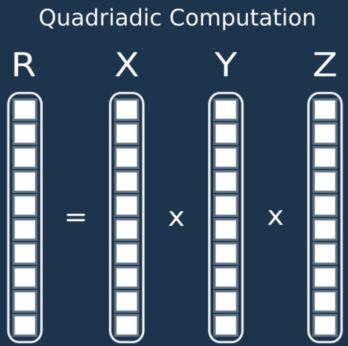
**Not Vectorized** but no **data copy**

**Vectorised**

But use **temporary vecXY**

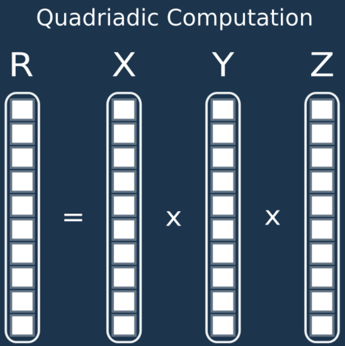
**2 std::transform**

# std::views::transform



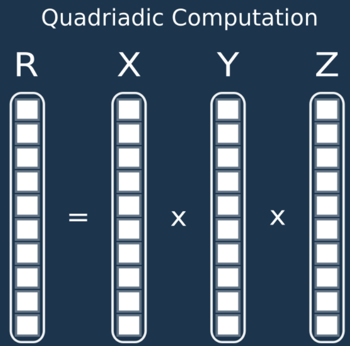
# std::views::transform

```
auto vXY = std::views::transform(std::views::zip(vecX, vecY),
                [](auto tuple){
                        auto & [x, y] = tuple;
                        return x*y;
                }
);
std::transform(std::execution::par_unseq,
        std::begin(vXY), std::end(vXY), std::begin(vecZ),
        std::begin(vecRes),
        [](float xy, float z){
                return xy *z;
        }
);
```



# std::views::transform

```
auto vXY = std::views::transform(std::views::zip(vecX, vecY, vecZ),
»	»	[](auto tuple){
»	»	»	auto & [x, y, z] = tuple;
»	»	»	return x*y*z;
»	»	}
);
std::transform(EXECUTION_POLICY,
»	std::begin(vXY), std::end(vXY),
»	std::begin(vecRes),
»	[](float res){
»	»	return res;
»	}
);
```

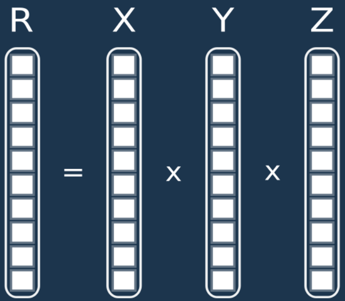


# std::views::transform

```
auto vXY = std::views::transform(std::views::zip(vecX, vecY, vecZ),
                [](auto tuple){
                        auto & [x, y, z] = tuple;
                        return x*y*z;
                }
);
std::transform(EXECUTION_POLICY,
        std::begin(vXY), std::end(vXY),
        std::begin(vecRes),
        [](float res){
                return res;
        }
);
```

No extra table needed

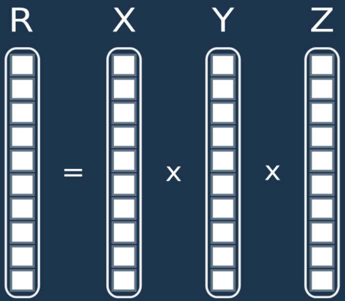
Quadriadic Computation



# std::views::transform

```
auto vXY = std::views::transform(std::views::zip(vecX, vecY, vecZ),
				[](auto tuple){
					auto & [x, y, z] = tuple;
					return x*y*z;
				}
);
std::transform(EXECUTION_POLICY,
	std::begin(vXY), std::end(vXY),
	std::begin(vecRes),
	[](float res){
		return res;
	}
);
```

Quadriadic Computation




No extra table needed

Not vectorized yet because of **std::views::zip**

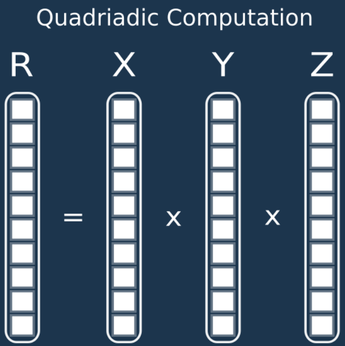
# std::views::transform

```
auto vXY = std::views::transform(std::views::zip(vecX, vecY, vecZ),
			[](auto tuple){
				auto & [x, y, z] = tuple;
				return x*y*z;
			}
);
std::transform(EXECUTION_POLICY,
	std::begin(vXY), std::end(vXY),
	std::begin(vecRes),
	[](float res){
		return res;
	}
);
```

No extra table needed

Use **std::tuple**
Contiguous elements

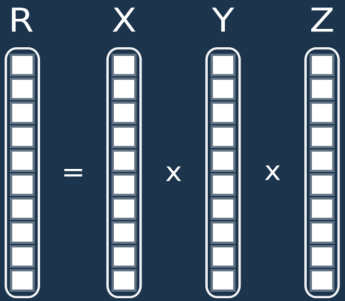
Not vectorized yet
because of **std::views::zip**



# std::views::transform

```
auto vXY = std::views::transform(std::views::zip(vecX, vecY, vecZ),
                [](auto tuple){
                        auto & [x, y, z] = tuple;
                        return x*y*z;
                }
);
std::transform(EXECUTION_POLICY,
        std::begin(vXY), std::end(vXY),
        std::begin(vecRes),
        [](float res){
                return res;
        }
);
```

Quadriadic Computation




Use **std::tuple**
Contiguous elements

Not vectorized yet
because of **std::views::zip**

No extra table needed

**Not Vectorisable**

# std::views::transform

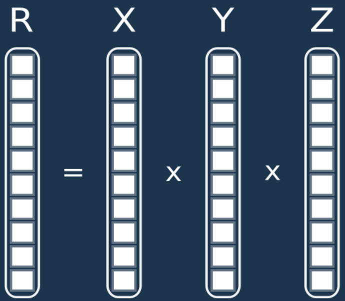
```cpp
auto vXY = std::views::transform(std::views::zip(vecX, vecY, vecZ),
				[](auto tuple){
					auto & [x, y, z] = tuple;
					return x*y*z;
				}
);
std::transform(EXECUTION_POLICY,
		std::begin(vXY), std::end(vXY),
		std::begin(vecRes),
		[](float res){
			return res;
		}
);
```

No extra table needed

Not vectorized yet because of **std::views::zip**

Use **std::tuple**
Contiguous elements

Quadriadic Computation




**Not Vectorisable**

**Vectorisable**

# Next Development

# Next Development

- **Steroscopic Reconstruction**

# Next Development

- **Steroscopic Reconstruction**

**Complete Refactoring**

hipeRTA

# Next Development

- **Steroscopic Reconstruction**

**Complete Refactoring**

**hipeRTA**

First with **Offline Version**

# Next Development

- **Steroscopic Reconstruction**

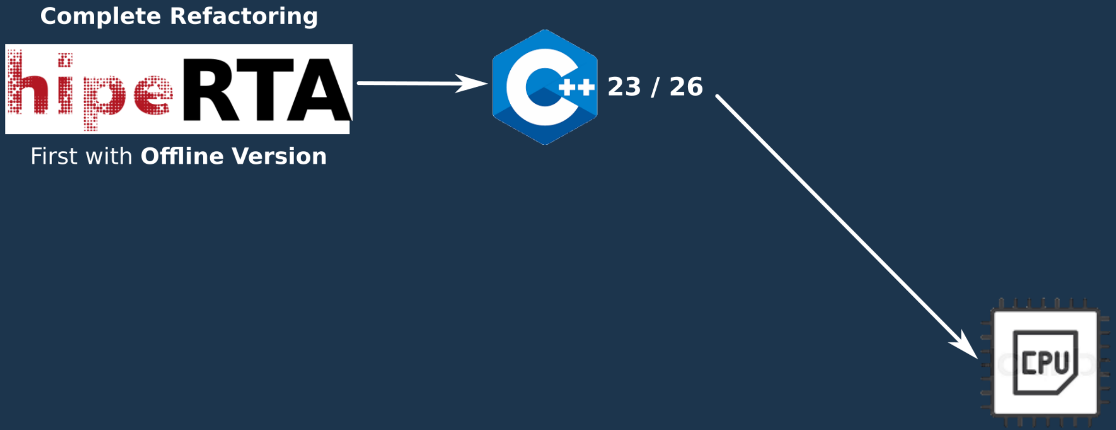
**Complete Refactoring**

hipeRTA → C++ **23 / 26**

First with **Offline Version**

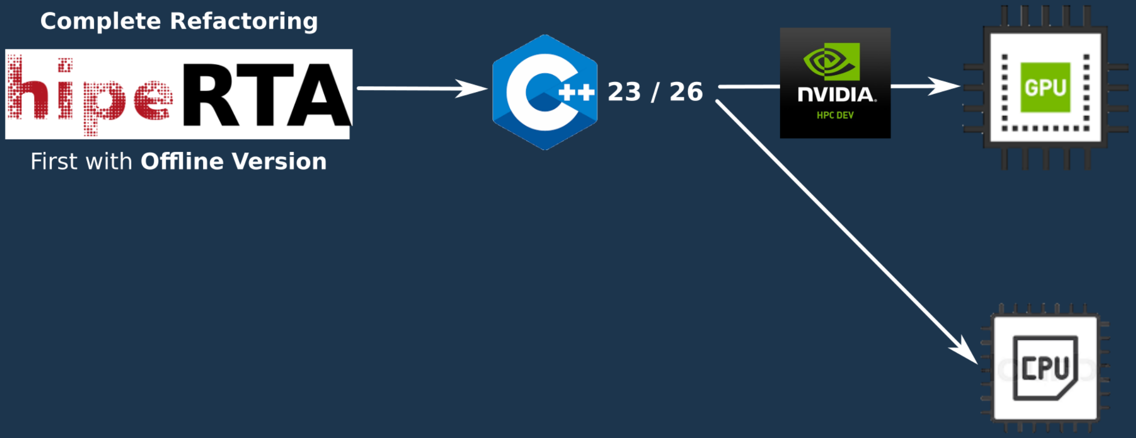
# Next Development

- **Steroscopic Reconstruction**



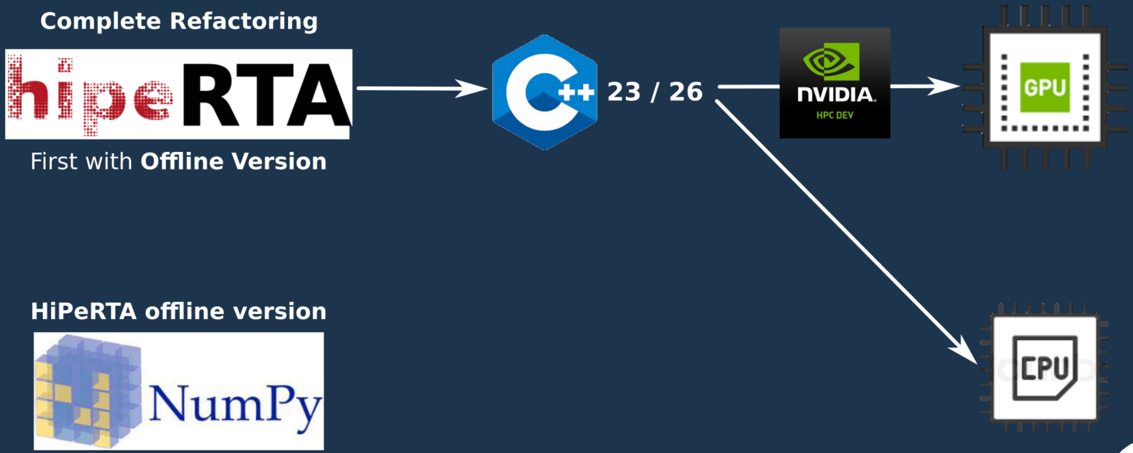
# Next Development

- **Steroscopic Reconstruction**



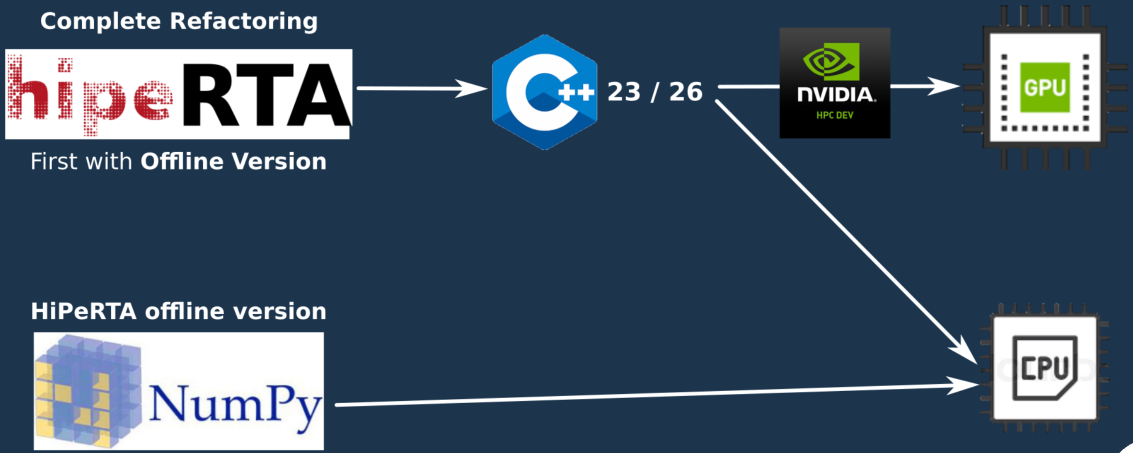
# Next Development

- Steroscopic Reconstruction



# Next Development

- **Steroscopic Reconstruction**



# Next Development

- **Steroscopic Reconstruction**