XKBlas: under the hood

https://gitlab.inria.fr/xkblas/versions

thierry.gautier@inrialpes.fr
CR INRIA, EPI AVALON, LIP
@Lyon, France

AVALON research team

- INRIA, CNRS, ENS de Lyon, UCBL
  - energy
  - component model
  - distributed algorithms
  - parallel algorithms
  - runtime
    - OpenMP, XKaapi

  ▸ XKBlas

- Target platforms
  - clouds, edge
  - **HPC**

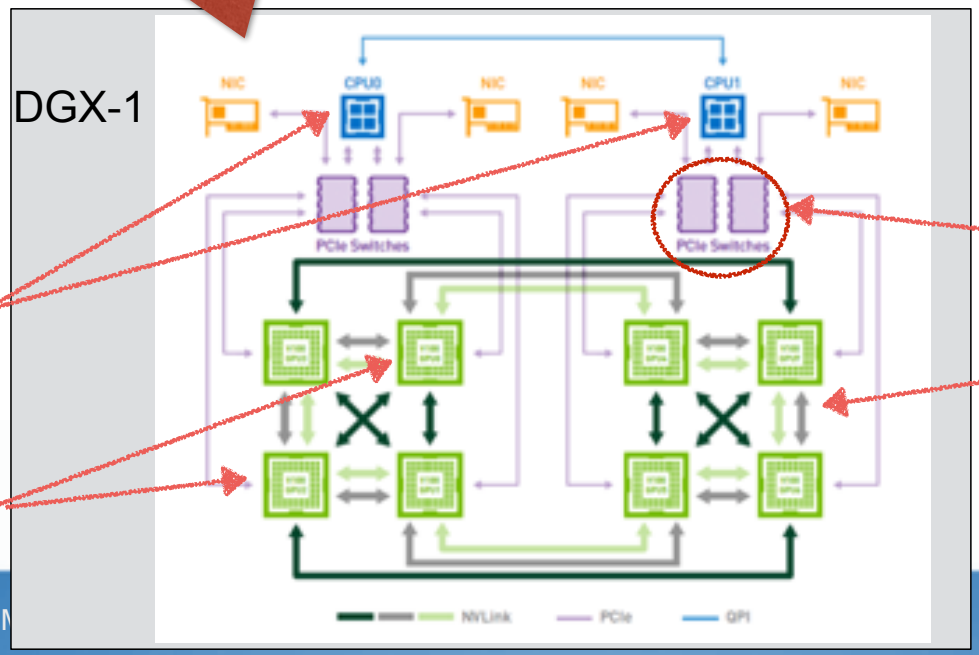
What is XKBlas?

```
dgemm( CblasNoTrans, CblasNoTrans, n, n, n,
   &alpha, A, n,
           B, n,
   &beta,  C, n);
```

Application code

Keypoints:

- multi-GPUs
- overlapping capabilities
- topology



2 CPUs

8 V100 GPUs

PCIe 3.0 switch (bottleneck)

- diff. #NVLink per pair of GPUs
- not complete network

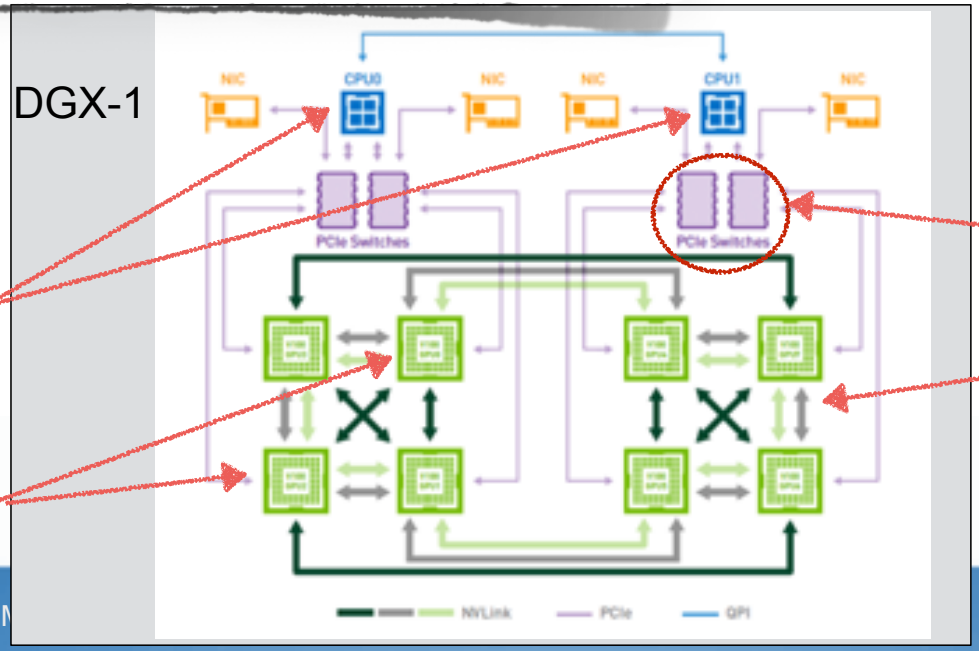
What is XKBlas?

```
dgemm( CblasNoTrans, CblasNoTrans, n, n, n,
   &alpha, A, n,
           B, n,
   &beta,  C, n);
```

Application code

```
…
for (m = 0; m < Cmt; m++)
  for (n = 0; n < Cnt; n++)
    for (k = 0; k < Ant; k++)
      INSERT_TASK_zgemm(
         transA, transB, tempmm, tempnn, tempkn,
         *alpha, A(m, k), ldam,  /* lda * Z */
         B(k, n), ldbk,   /* ldb * Y */
         zbeta, C(m, n), ldcm);  /* ldc * Y */
```

tiled dgemm



2 CPUs

8 V100 GPUs

PCIe 3.0 switch (bottleneck)

- diff. #NVLink per pair of GPUs
- not complete network

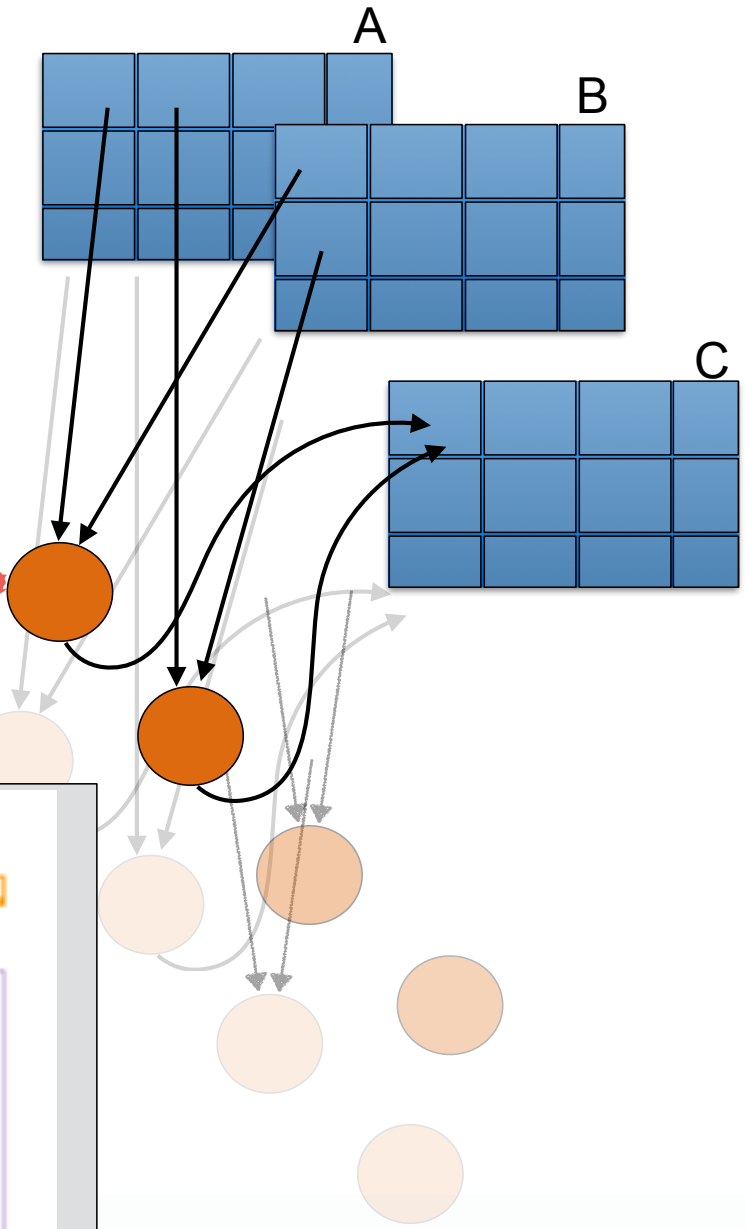
What is XKBlas?

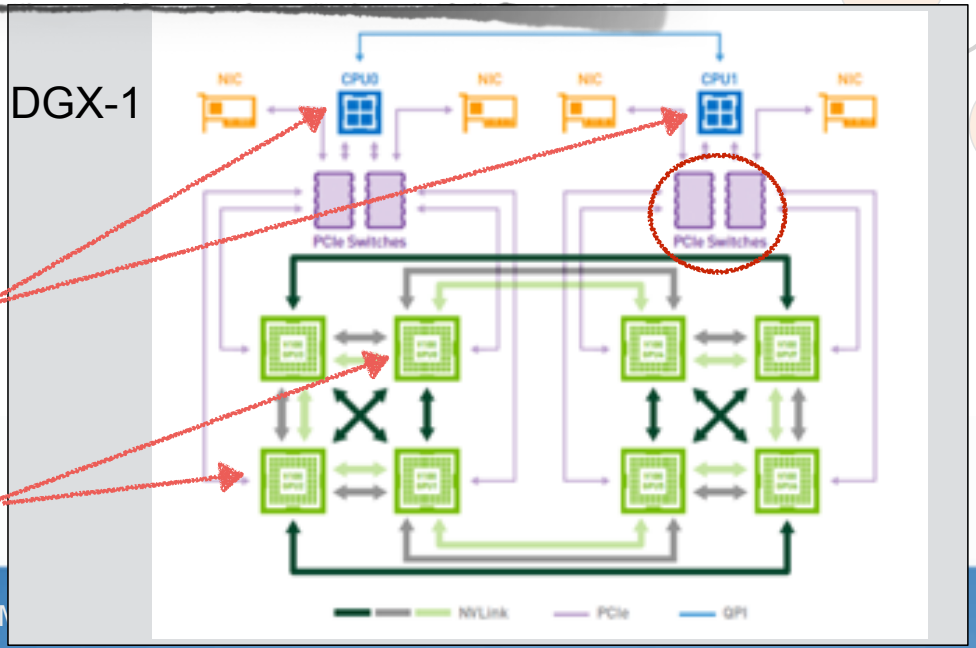
```
dgemm( CblasNoTrans, CblasNoTrans, n, n, n,
   &alpha, A, n,
           B, n,
   &beta,  C, n);
```

Application code

tiled dgemm

```
…
for (m = 0; m < Cmt; m++)
  for (n = 0; n < Cnt; n++)
    for (k = 0; k < Ant; k++)
      INSERT_TASK_zgemm(
         transA, transB, tempmm, tempnn, tempkn,
         *alpha, A(m, k), ldam,  /* lda * Z */
         B(k, n), ldbk,   /* ldb * Y */
         zbeta, C(m, n), ldcm);  /* ldc * Y */
```





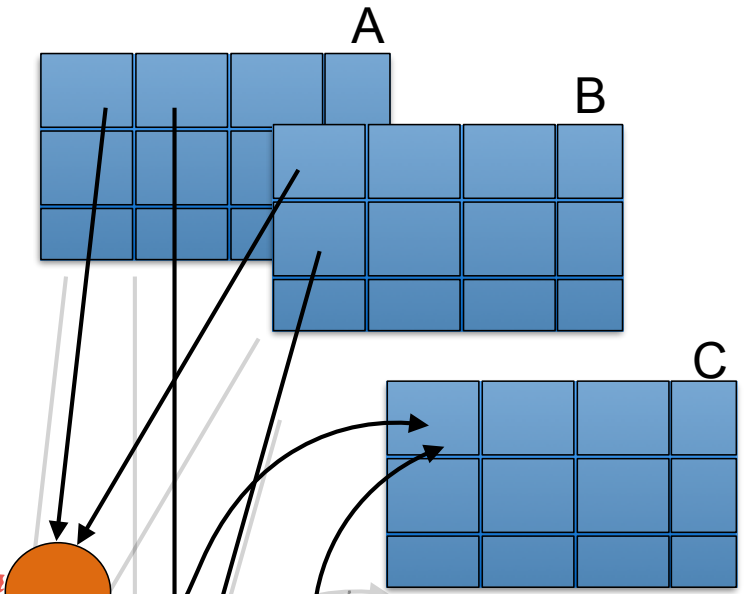
What is XKBlas?

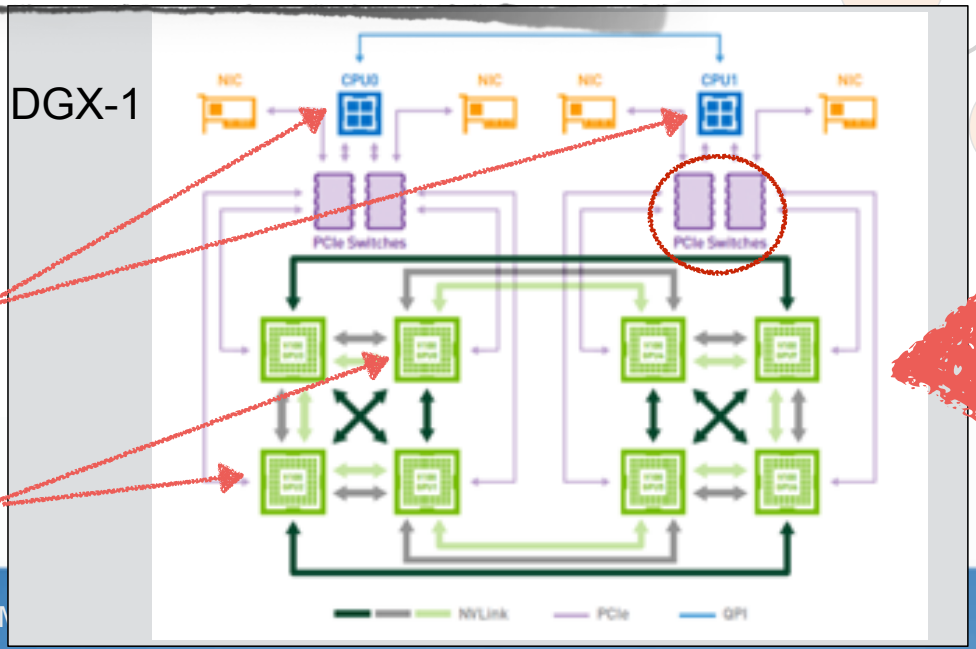
```
dgemm( CblasNoTrans, CblasNoTrans, n, n, n,
   &alpha, A, n,
           B, n,
   &beta,  C, n);
```

Application code

```
…
for (m = 0; m < Cmt; m++)
  for (n = 0; n < Cnt; n++)
    for (k = 0; k < Ant; k++)
      INSERT_TASK_zgemm(
         transA, transB, tempmm, tempnn, tempkn,
         *alpha, A(m, k), ldam,  /* lda * Z */
         B(k, n), ldbk,   /* ldb * Y */
         zbeta, C(m, n), ldcm);  /* ldc * Y */
```

tiled dgemm





2 CPUs

8 V100 GPUs

XKaapi

XKBlas overview

- Joint work with J. V. Lima UFSM, Brasil
- https://gitlab.inria.fr/xkblas/versions, CeCILL-C

- XKBlas = Blas (L3) for hybrid architecture on top of XKaapi
  - multi-CPUs + multi-GPUs
  - Linear Algebra Subroutines within XKBlas
    - Level 3 kernels only, Z, C, D, S precisions
    - GEMM,TRSM, TRMM, SYMM, SYRK, SYR2K, HEMM, HERK, HER2K
      (mostly imported from Chameleon library)
    - + GEMMT (see MKL GEMMT interface)

- XKaapi [2010-, Grenoble] = task based runtime for multi-CPUs multi-GPUs
  - ~ OpenMP dependent task model
  - automatic schedule tasks among the resources
    - take into account hybrid resources (CPU or GPU)
    - paid attention to data locality & topology

XKBlas keypoints

- **LAPACK memory layout**
  - column major (cuBLAS-XT)

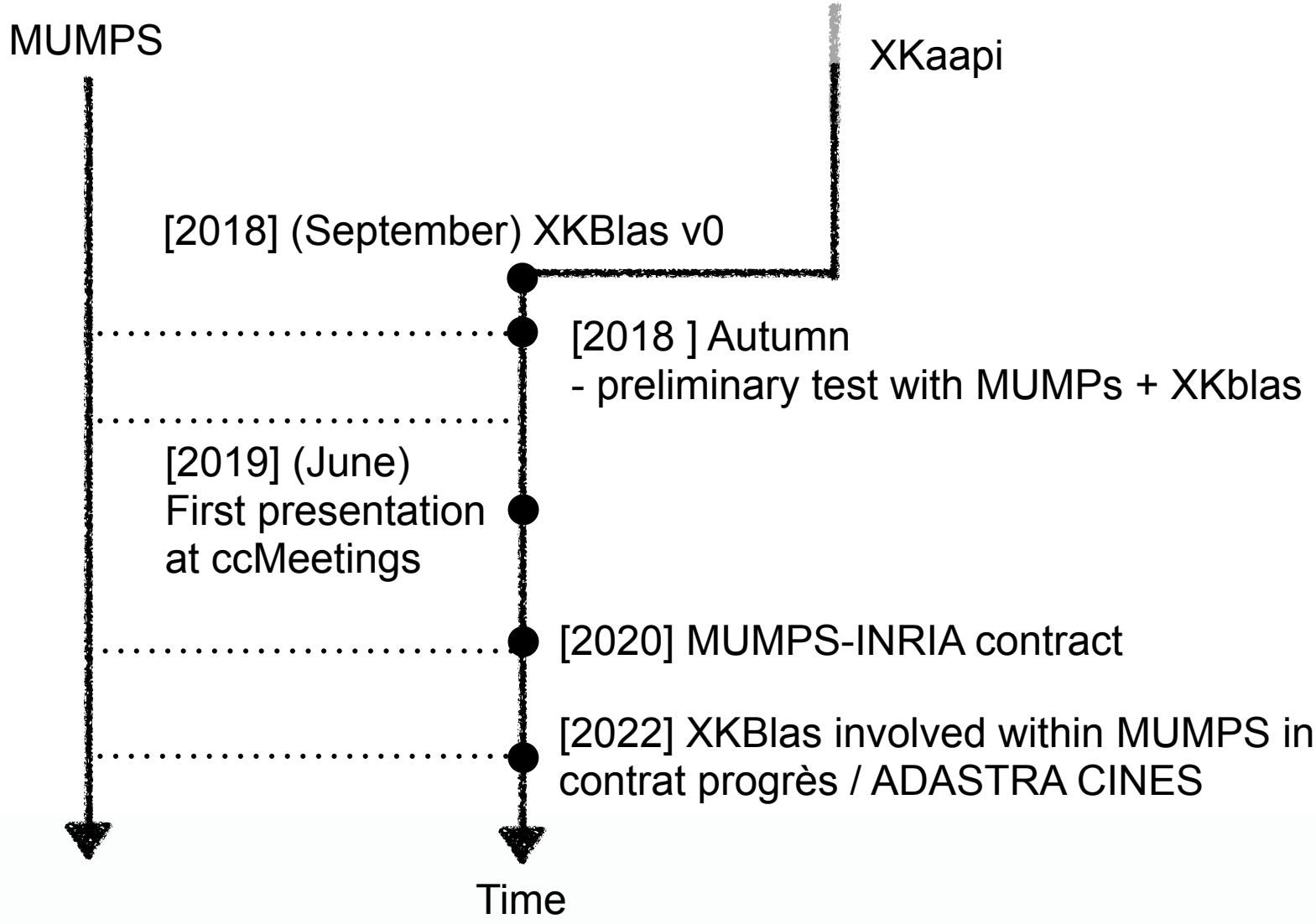
- **Asynchronous APIs**
  - any calls to XKBlas' kernels are non blocking
  - **`xkblas_register_memory_async()`**: "asynchronous `cudaHostRegister`"
  - **`xkblas_sync()`**: ensures completion of all previous kernels

- **Lazy memory coherency**
  - when a task is offloaded, output data remains on device(s)
  - kernel completion ≠ host memory coherency with device memories
  - **`xkblas_memory_coherency_async()`**: makes coherent host & devices memories

- Almost all XKBlas subroutines create XKaapi tasks

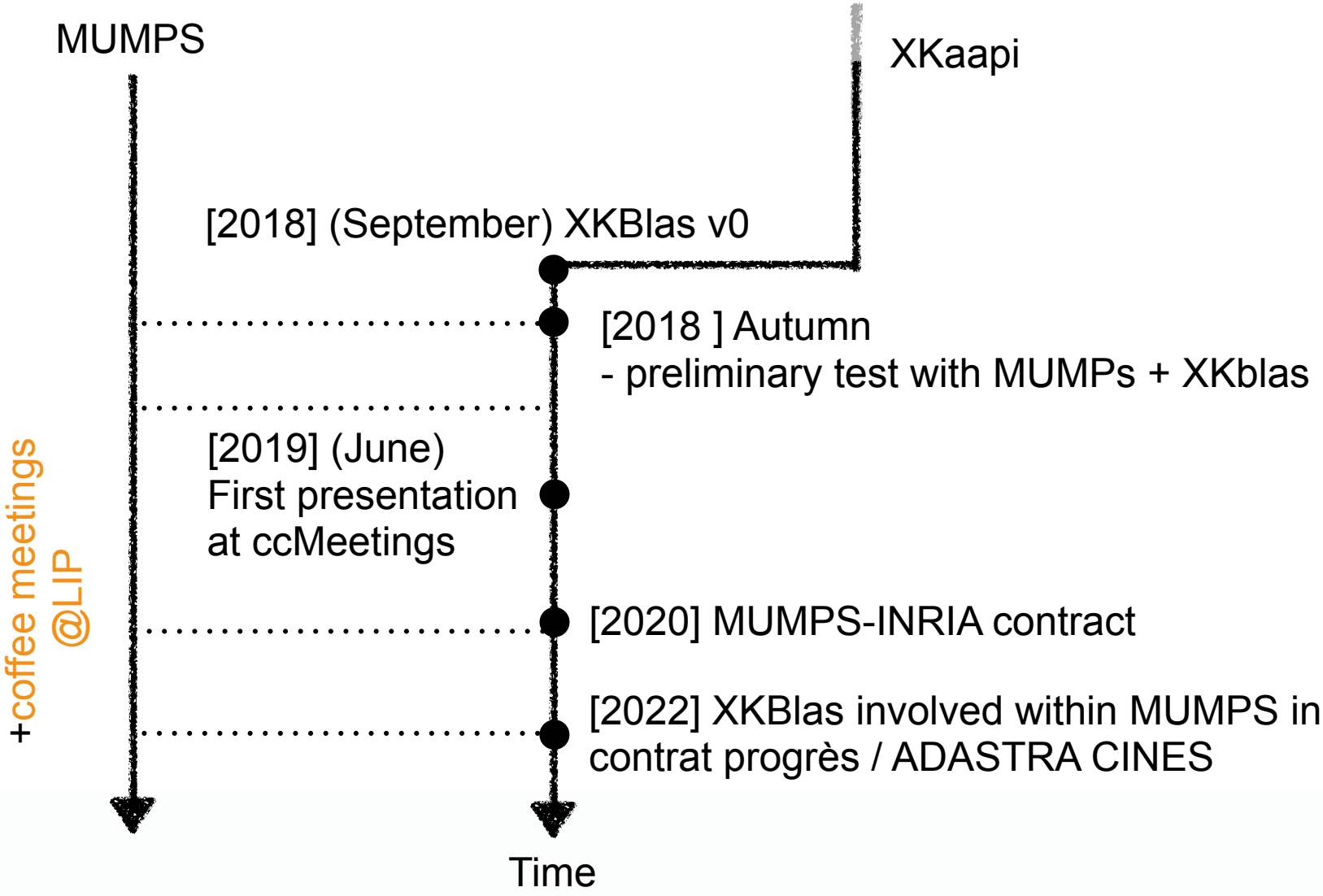
Mumps & XKBlas

- One of the target library for multi-GPUs



Mumps & XKBlas

- One of the target library for multi-GPUs

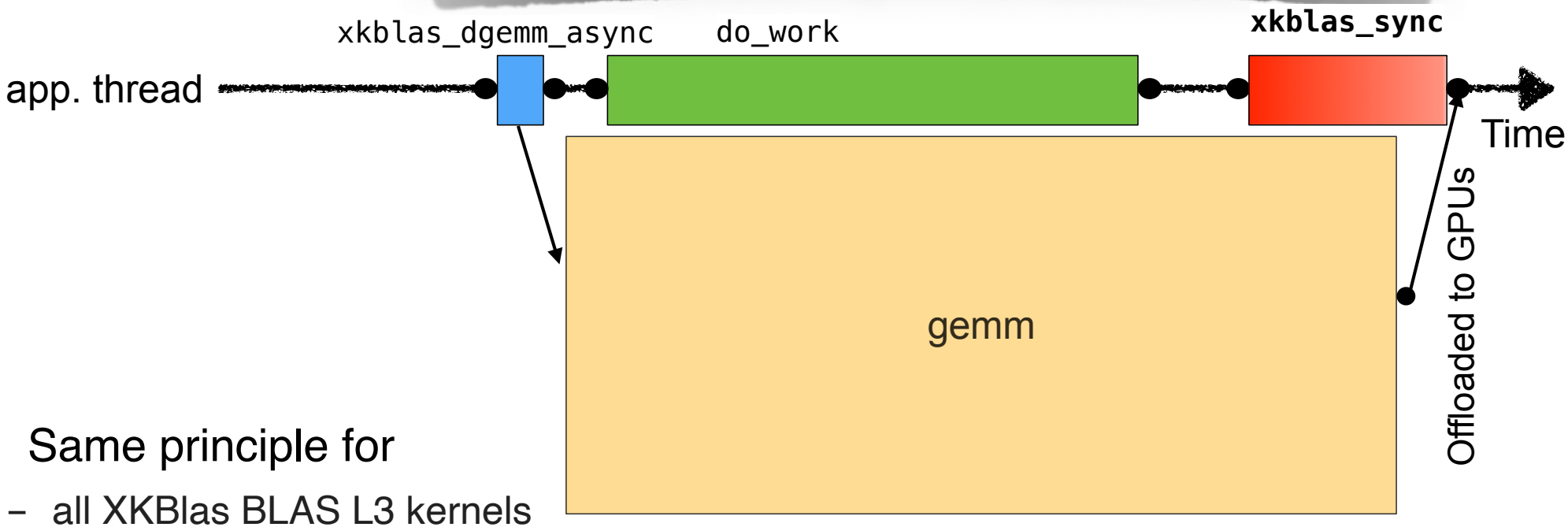


Integration with MUMPS

- Classical BLAS: GEMM, TRSM
  - tiled algorithms
- Extension to GEMMT [introduced in the MKL]
- Use of asynchronous XKBlas functions for overlaping with MUMPS code
  - xkblas_register_memory_async
  - xkblas_memory_coherent_async

Illustration

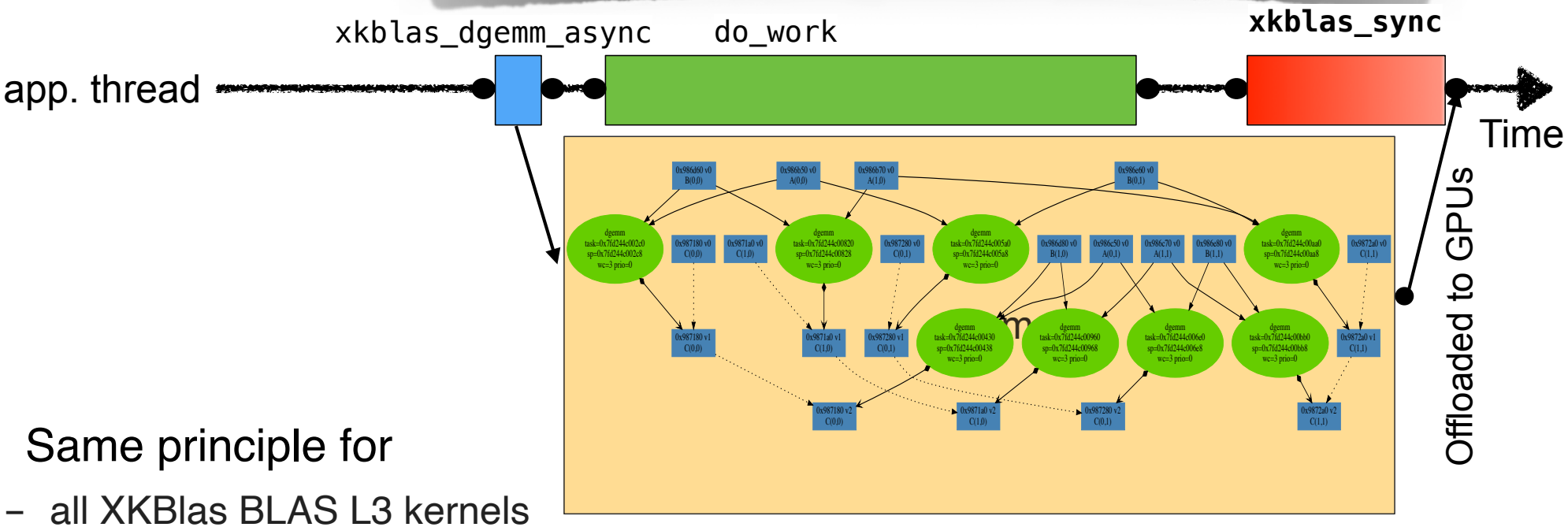
```
/* do non blocking gemm: */
xkblas_dgemm_async( CblasNoTrans, CblasNoTrans, n, n, n,
  &alpha, A, n,
          B, n,
  &beta,  C, n);

/* here local work */
do_work();

/* wait for dgemm computation ends */
xkblas_sync();
```



- Same principle for
  - all XKBlas BLAS L3 kernels
  - Memory coherency function to get results back
  - *Host Memory Registration*

Illustration

```
/* do non blocking gemm: */
xkblas_dgemm_async( CblasNoTrans, CblasNoTrans, n, n, n,
  &alpha, A, n,
          B, n,
  &beta,  C, n);

/* here local work */
do_work();

/* wait for dgemm computation ends */
xkblas_sync();
```

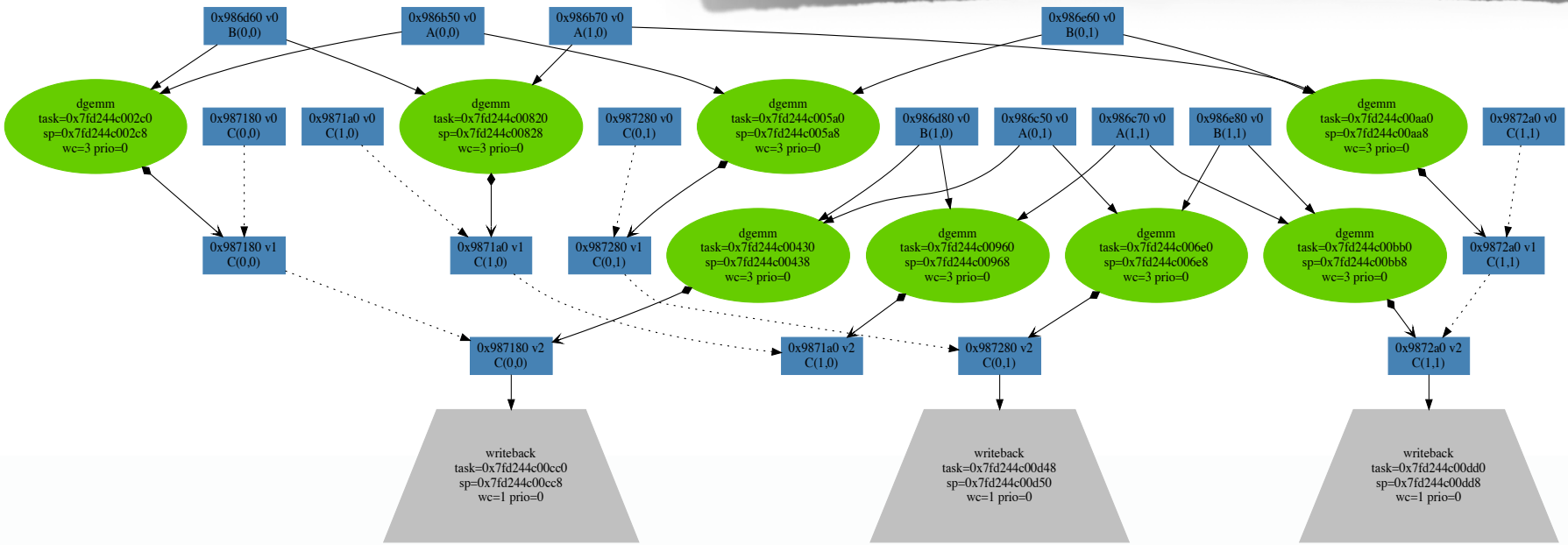


- Same principle for
  - all XKBlas BLAS L3 kernels
  - Memory coherency function to get results back
  - *Host Memory Registration*

Memory coherency

- The way to get back results on host memory

```
xkblas_dgemm_async( CblasNoTrans, CblasNoTrans, n, n, n,
   &alpha, A, n,
           B, n,
   &beta,  C, n);

/* memory coherency: get only the Lower part of C */
xkblas_memory_coherency_async(
  CblasUpper, /* could be CblasLower or 0 (all) */
  0,
  n, n,
  C, n, sizeof(double), /* matrix, ld, sizeof */
);

/* wait all previous tasks spawned */
xkblas_sync();
```

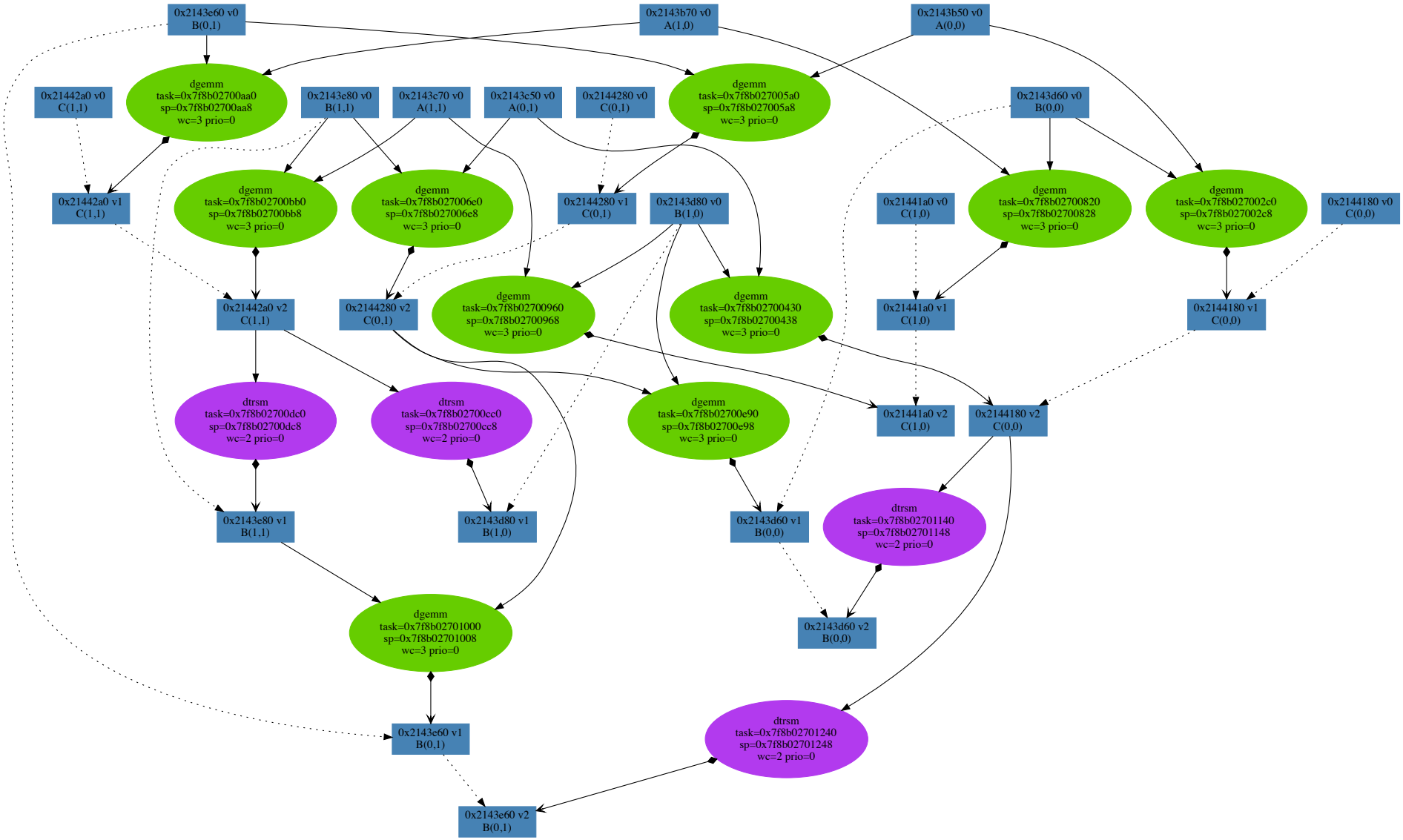


Composition

- Gemm and Trsm

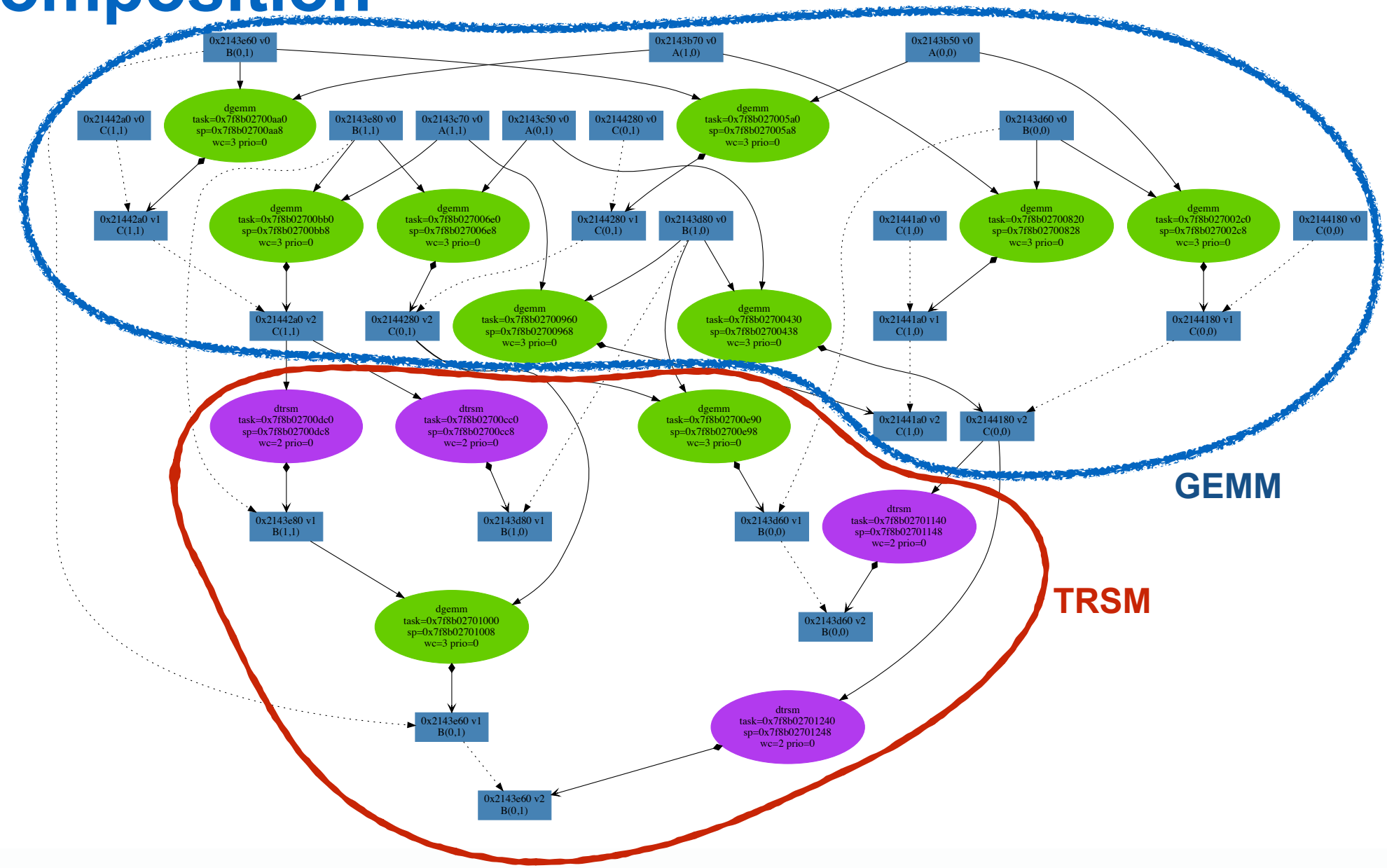
```
xkblas_dgemm_async( CblasNoTrans, CblasNoTrans, n, n, n,
    &alpha, A, n,
            B, n,
    &beta,  C, n);

xkblas_dtrsm_async( CblasLeft, CblasUpper, CblasNoTrans, CblasUnit,
    n, n,
    &alpha, C, n, B, n);

xkblas_blas_sync();
```

Composition



➡ **less idle time => better resource usage = better performance!**

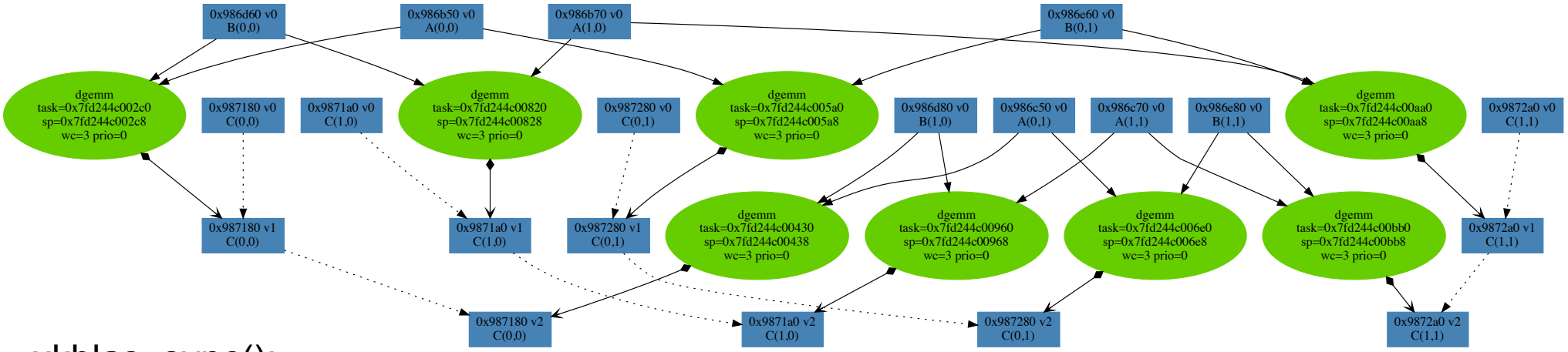
Composition



➡ **less idle time => better resource usage = better performance!**

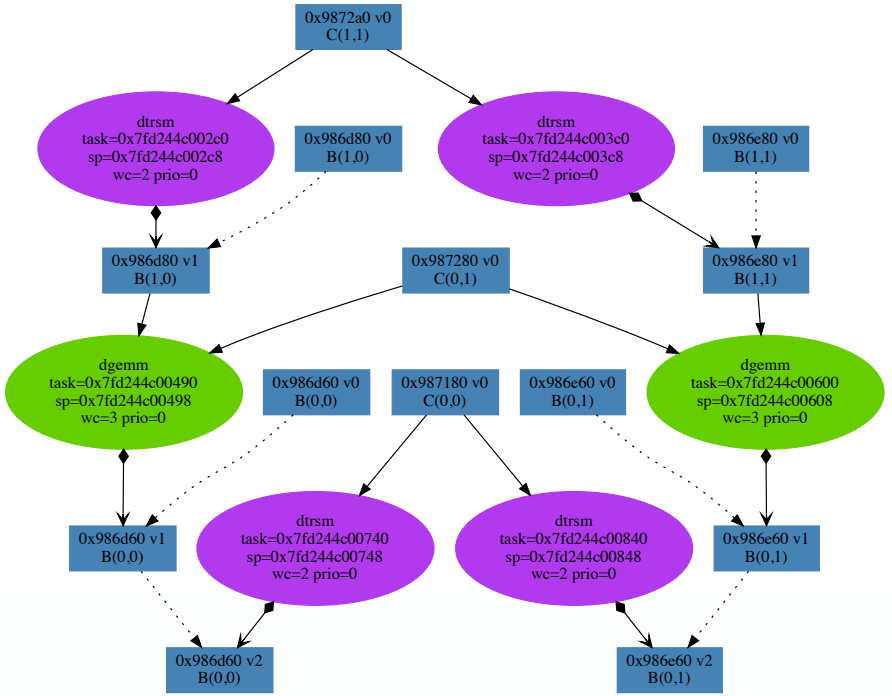
~~Composition~~

~GEMM



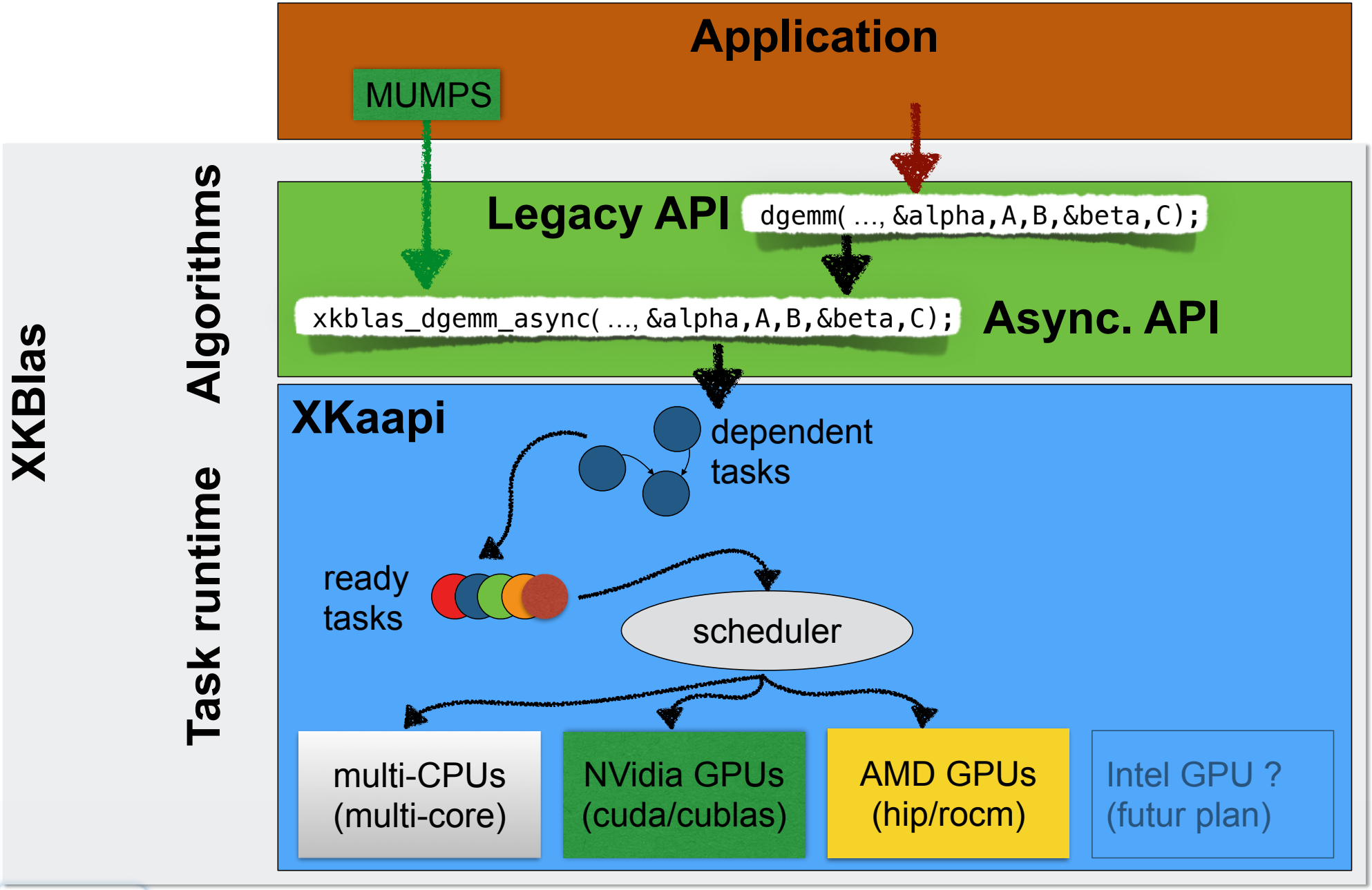
xkblas_sync():

~TRSM



xkblas_sync():

Resume


Application
MUMPS
XKBlas
Algorithms
Legacy API
dgemm(..., &alpha,A,B,&beta,C);
xkblas_dgemm_async(..., &alpha,A,B,&beta,C);
Async. API
Task runtime
XKaapi
dependent tasks
ready tasks
scheduler
multi-CPUs (multi-core)
NVidia GPUs (cuda/cublas)
AMD GPUs (hip/rocm)
Intel GPU ? (futur plan)

XKaapi challenging issues

- Objective
  - write once, run anywhere… with performance guarantee
    - classical task scheduling problem => heuristics
- Goals
  - Exploitation of all the available **hardware parallelism**
    - CPU cores
    - GPU cores
    - Communications between CPU-GPU, GPU-GPU
  - Reduction of the communication cost
    - take care of data locality
    - hide communication latency by computation

- Parallel algorithms
- Parallel slackness
- Multi-versioning task

- Workstealing scheduler
- Data locality heuristic
- Topology aware data transfer

XKaapi guidelines

- Macro data flow model defines in the 90th (Athapascan, [PACT'98])
  - global memory for communication between tasks
  - **task** = computing unit + **description of the data accesses** to the global memory
    - data access = read / write / readwrite / cumulative write
    - dependencies between tasks automatically deduced from theirs data accesses
  - Non preemptive scheduling with theoretical guarantees

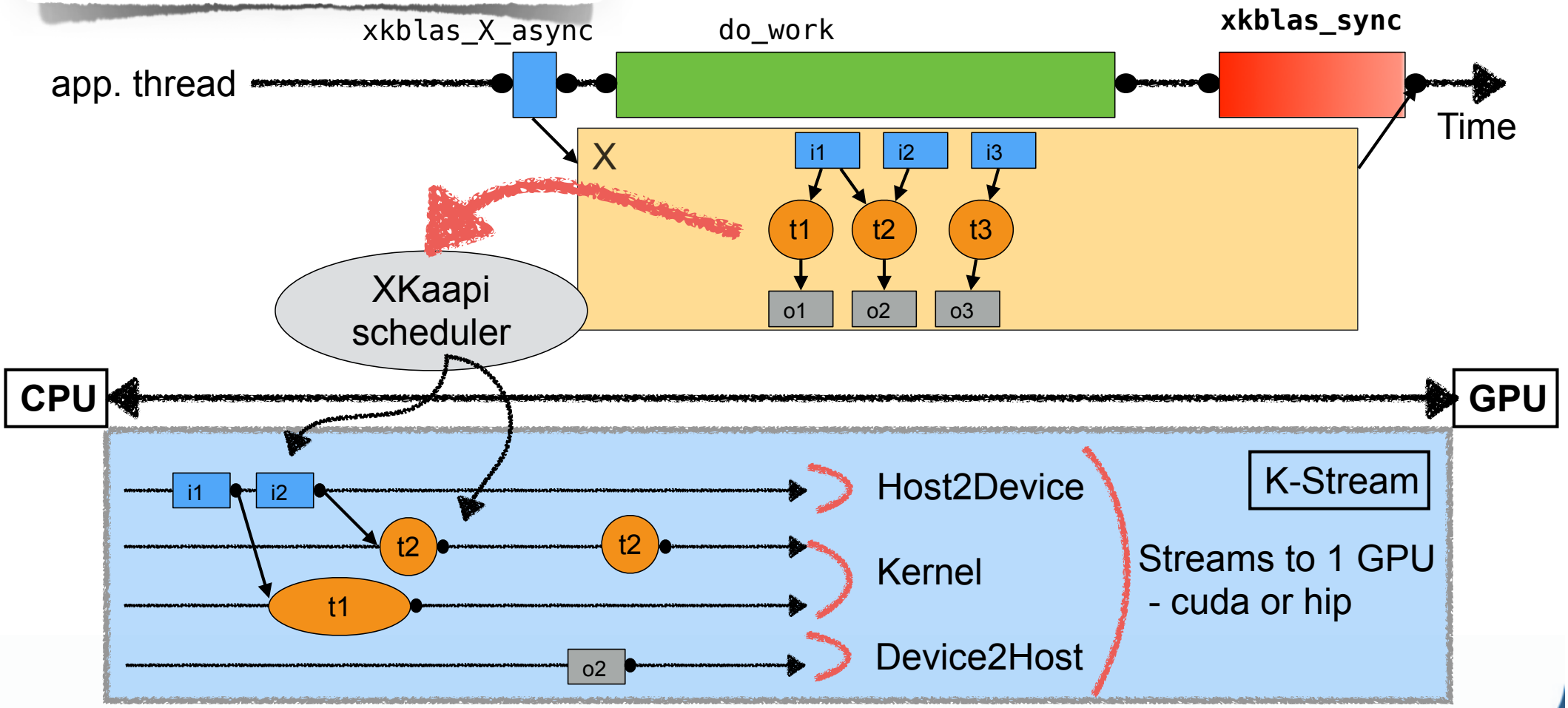
- **Sequential semantic**
  - the value returned by the read statements is the last written value according to the lexicographic order defined by the program (statements are lexicographically ordered by ';')

- **Ultra light task implementation**
  - 16 Bytes for a task descriptor versus ~ 448Bytes for LLVM OpenMP

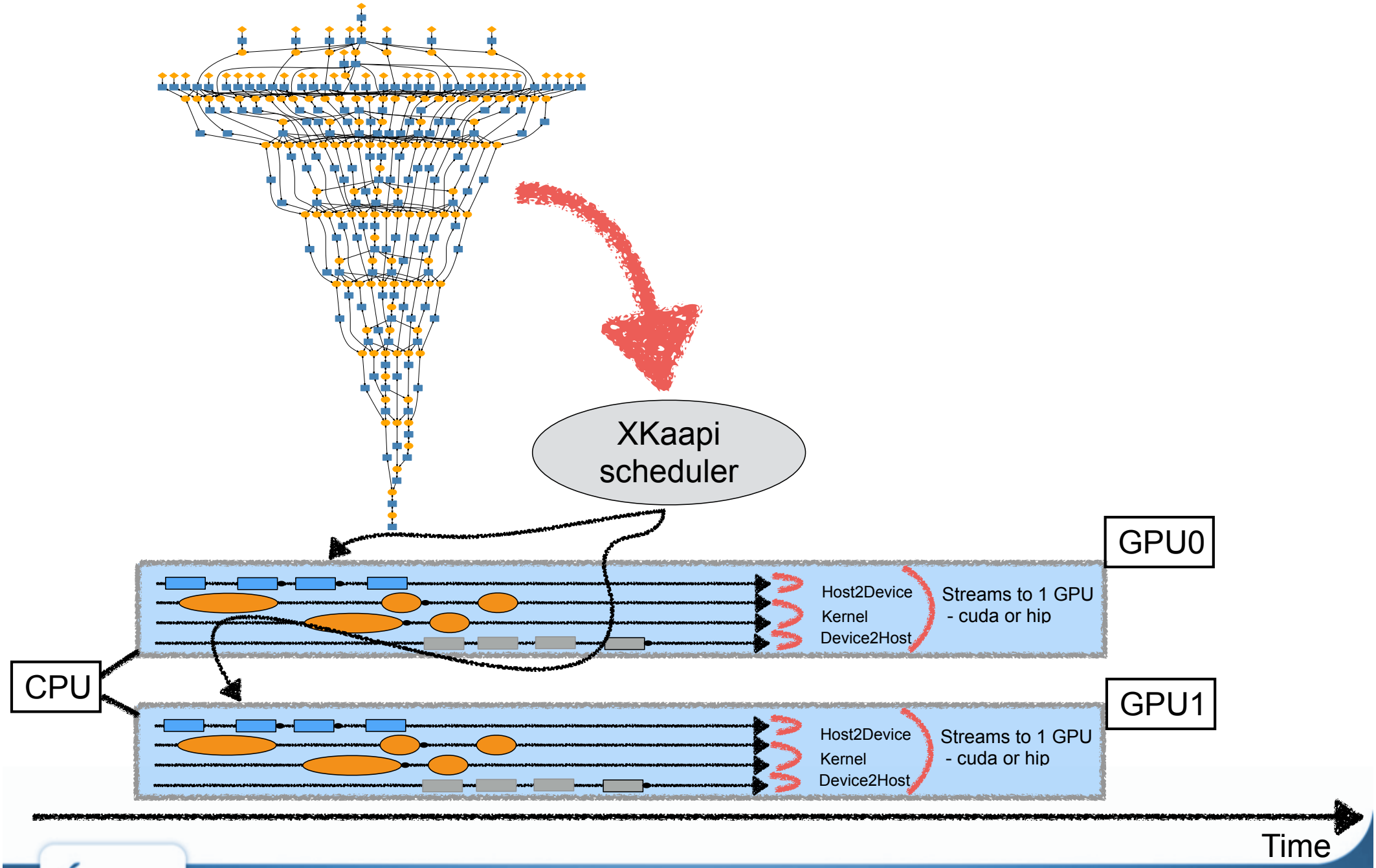
- Various APIs
  - C: the lowest interface, the most verbose  used by XKBlas
  - C++, Fortran, #pragma for kaapi construct

Offloading to 1 GPU: principle

```
/* do xkblas kernel 'X' */
xkblas_X_async( i, o );

/* here local work */
do_work();

/* wait for dgemm computation ends */
xkblas_sync();
```



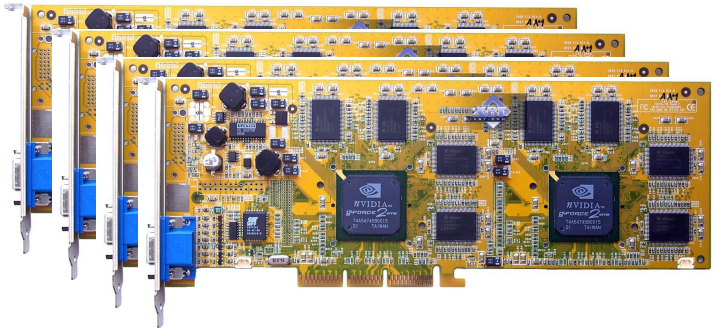
Multi-GPUs = 1 K-stream / GPU

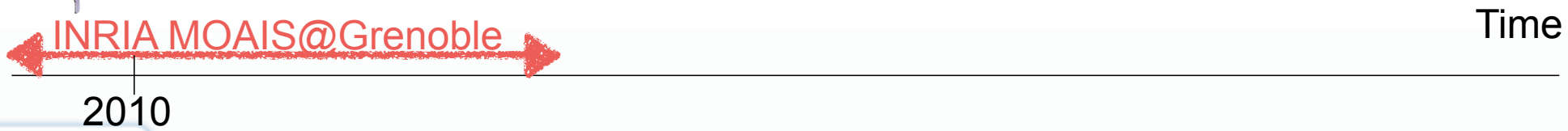


XKaapi & multi-GPUs

- Linux based system

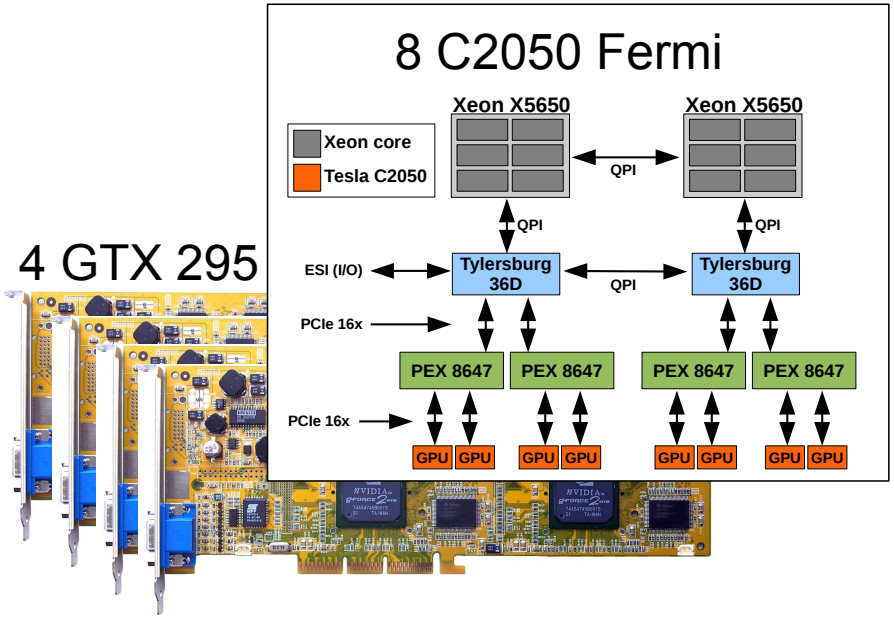
4 GTX 295 = 8 GPUs GT200B

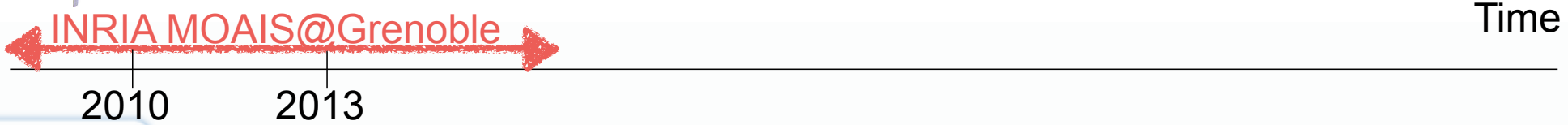




XKaapi & multi-GPUs

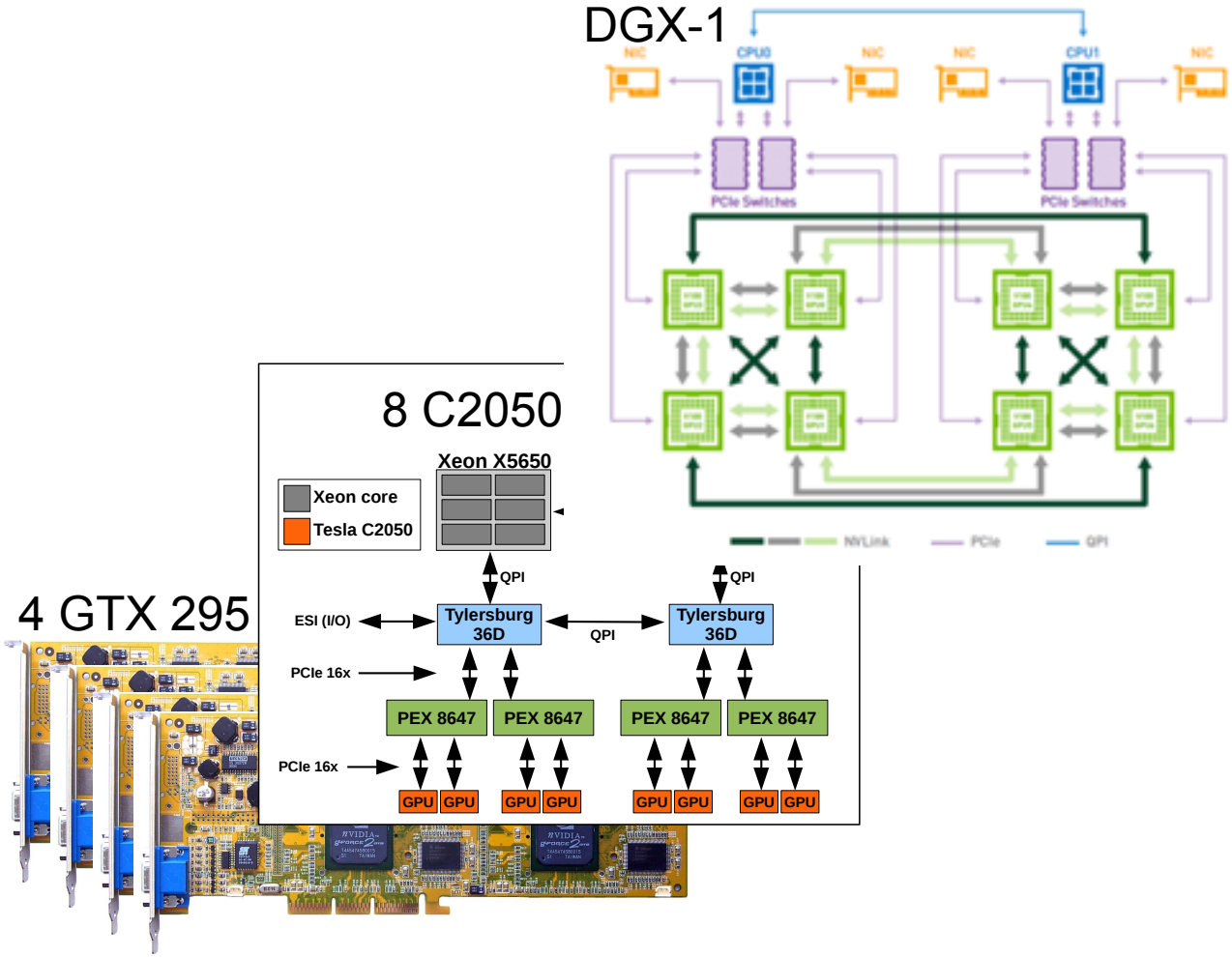
- Linux based system

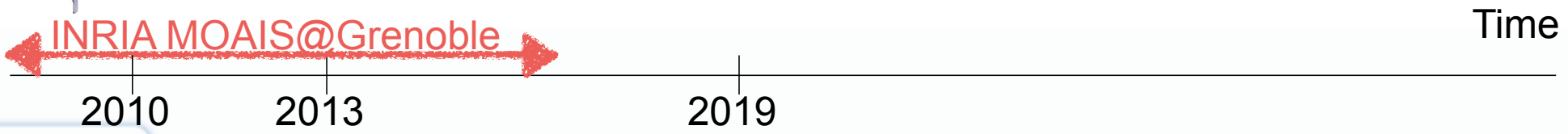




XKaapi & multi-GPUs

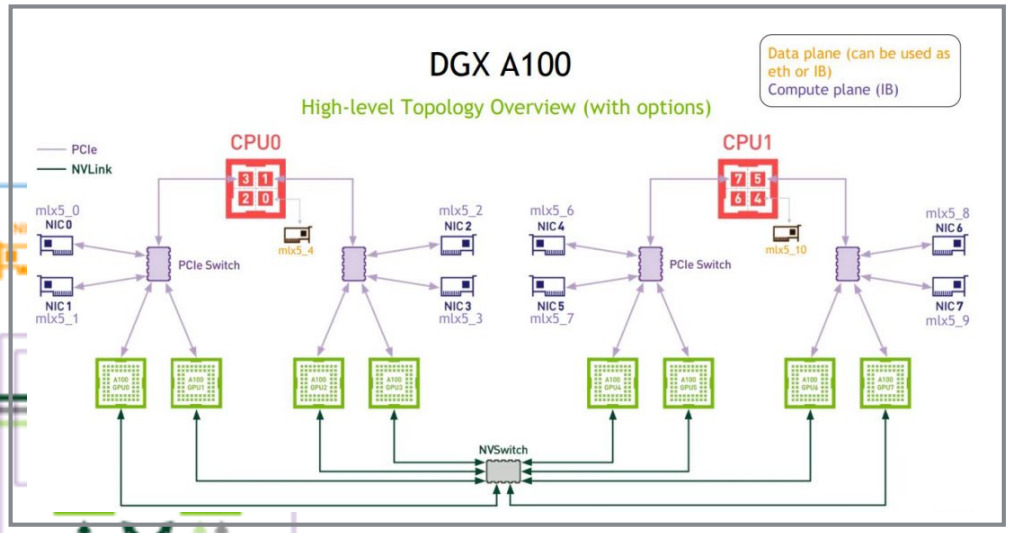
- Linux based system



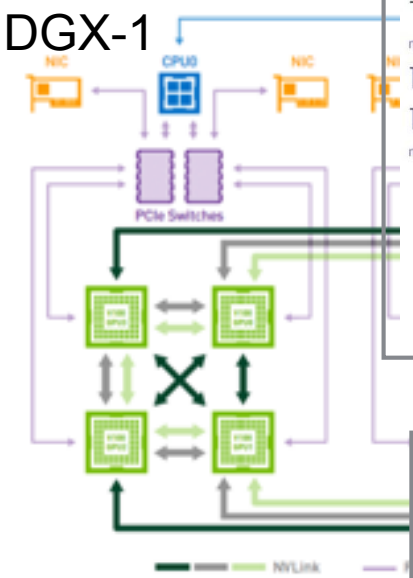


XKaapi & multi-GPUs

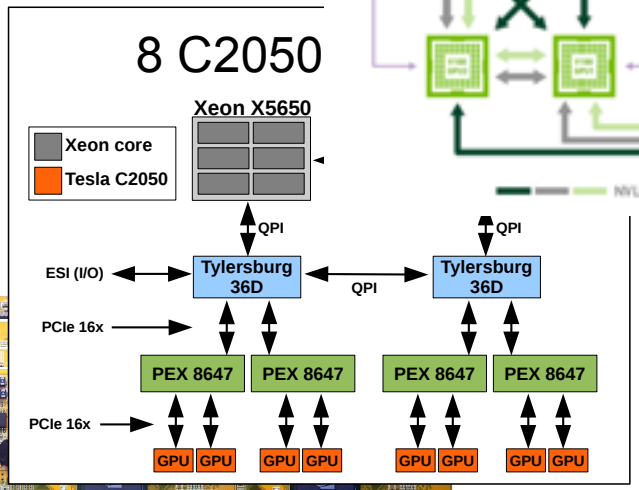
- Linux based system



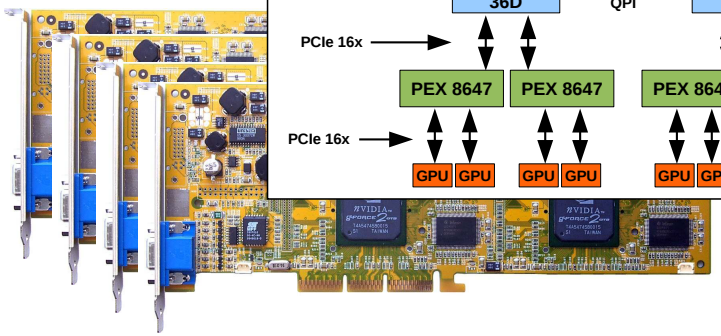
DGX-1



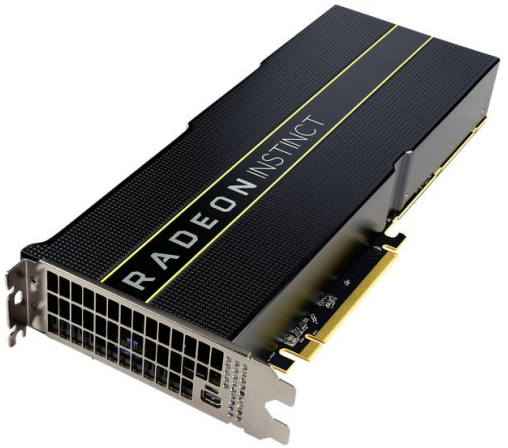
8 C2050



4 GTX 295



8 x AMD Radeon Instinct MI50



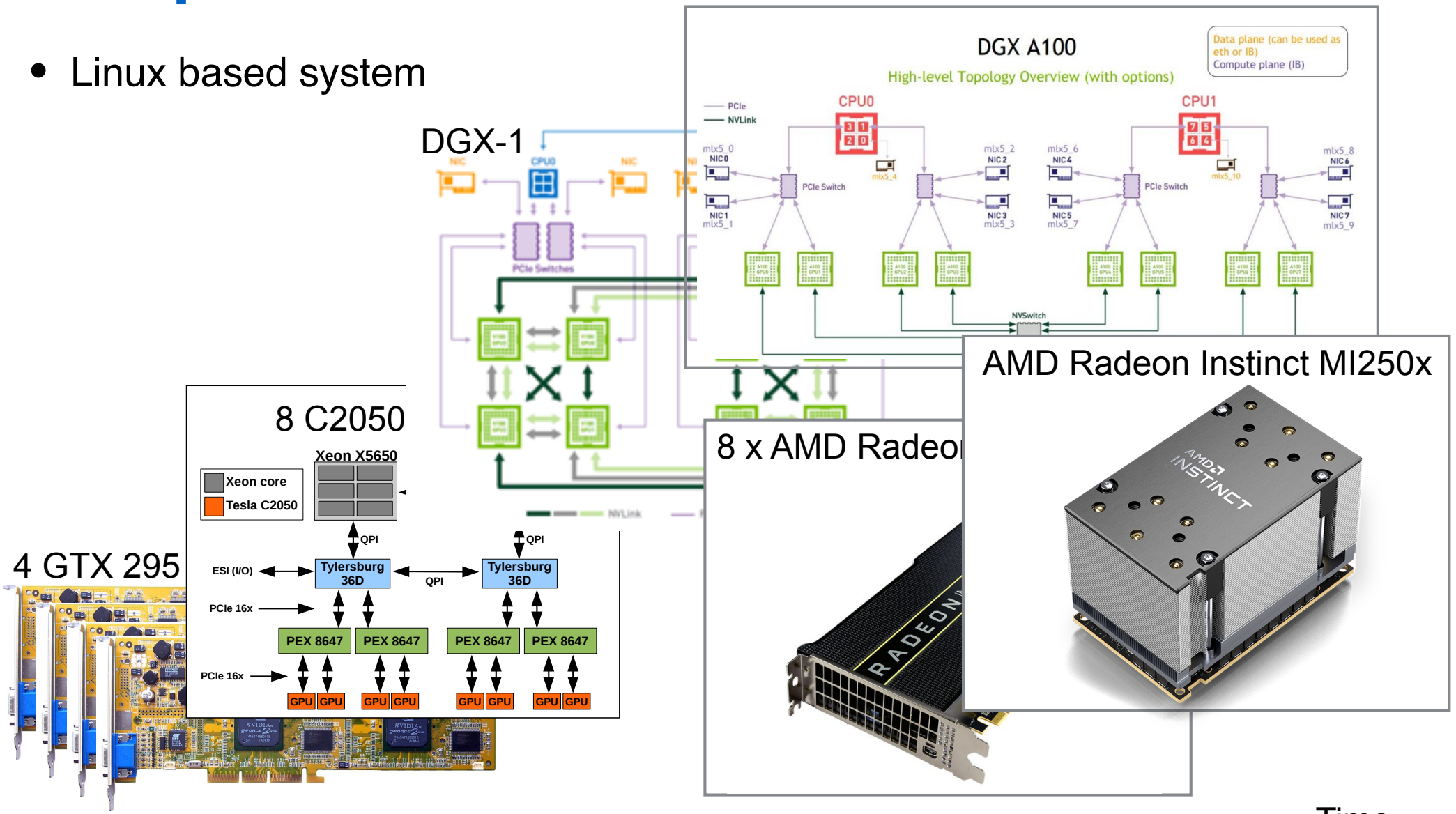
INRIA MOAIS@Grenoble

Time

2010 2013 2019 2021

XKaapi & multi-GPUs

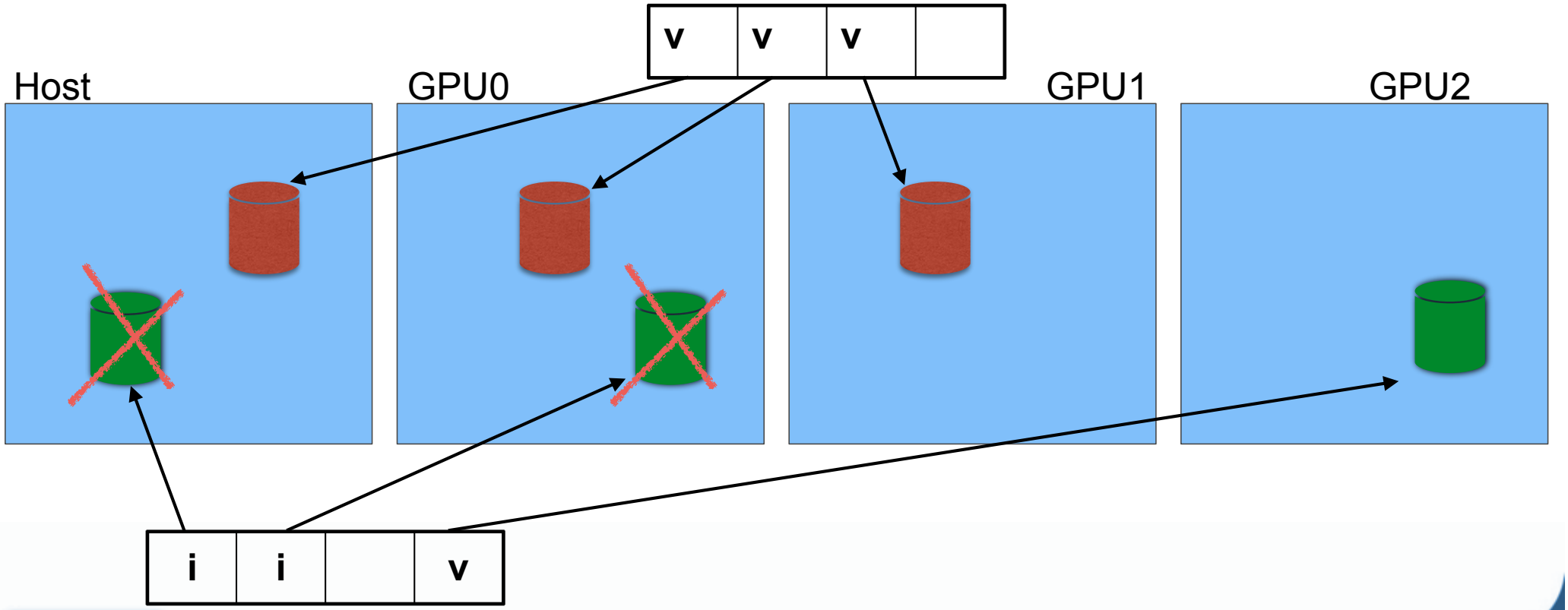
- Linux based system



Managing data across GPUs

- Memory GPU viewed as a «cache» of the host memory
  - distributed shared cache coherency protocols (CPU)

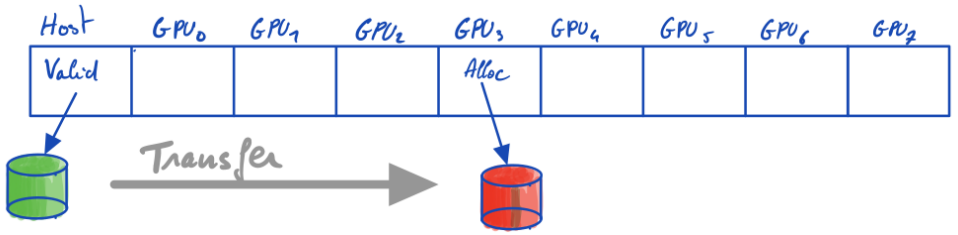
- % of memory reserved for XKBlas defined by XKBLAS_CACHE_LIMIT
  - default value is 95%



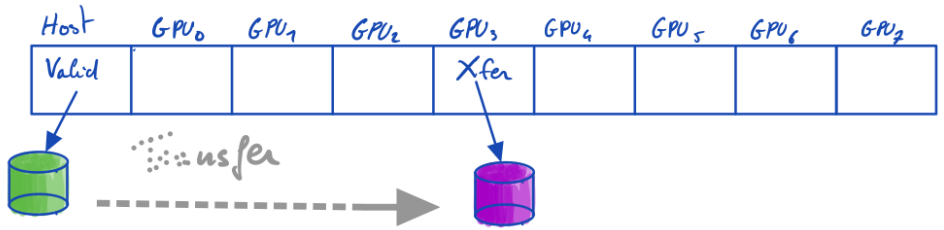
How XKaapi manages copies

- Assume GPU_3 needs a data on the host (that stores a valid copy)

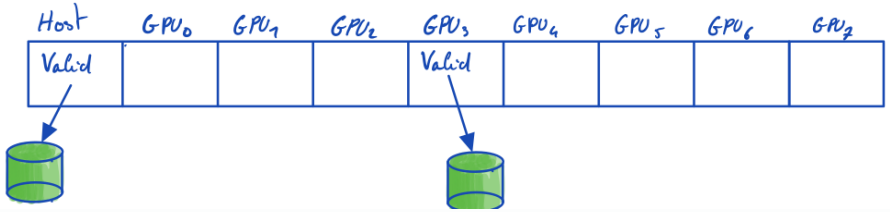
(1) Sender: the Host
Receiver: GPU_3



(2) copy on GPU_3 is in-transfer

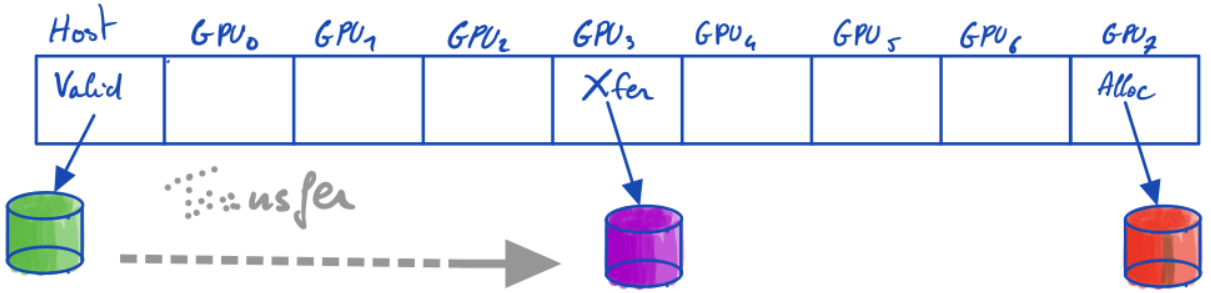


(3) in-transfer is done, the copy is marked as "valid"

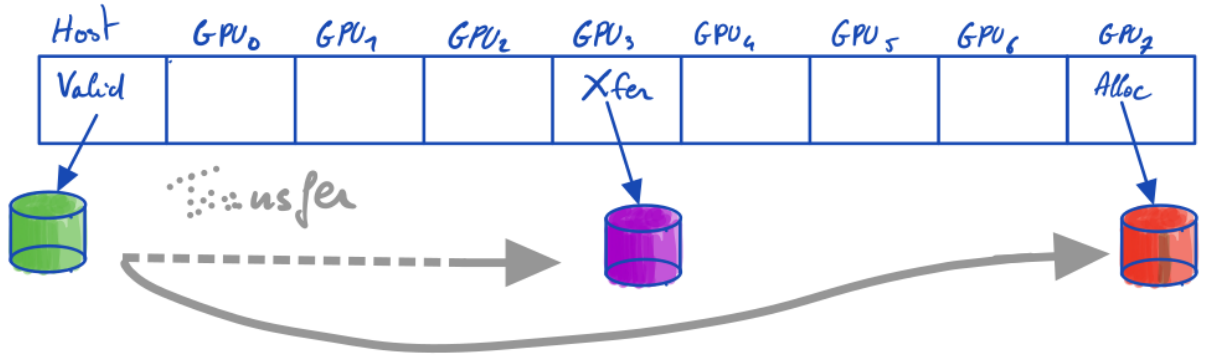


Optimistic heuristic for D-2-D transfer

- Now assume GPU_7 requires the data



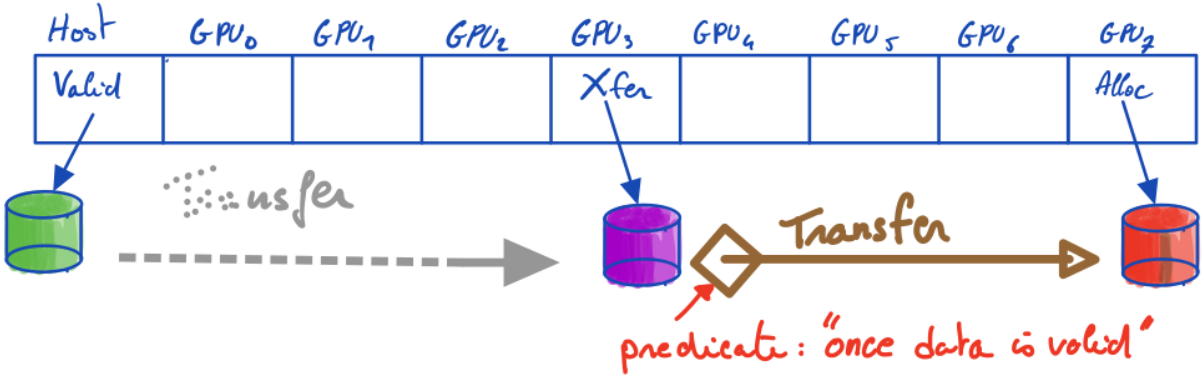
- Without our heuristic, data is send by the Host holding the only valid copy



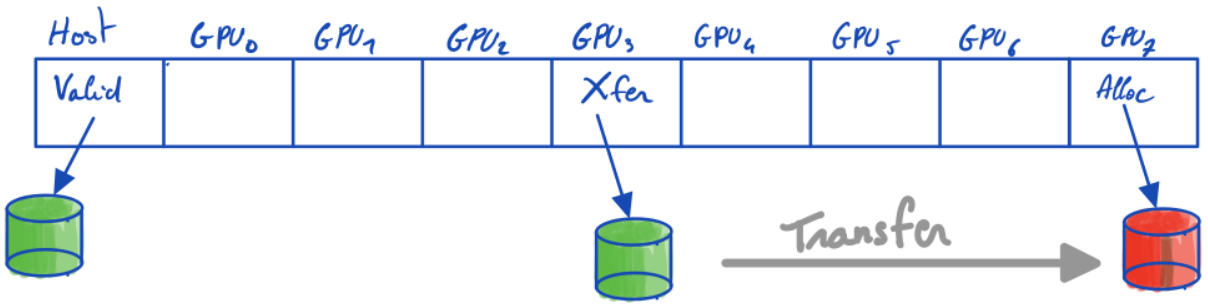
2x host-to-device transfers (PCIe 3.0 x 16~ 15GB/s) but device-to-device transfer can use high speed network (NVLink DGX1 ~ 96GB/s bi-directional)

Optimistic heuristic for D-2-D transfer

- Now assume GPU_7 requires the data

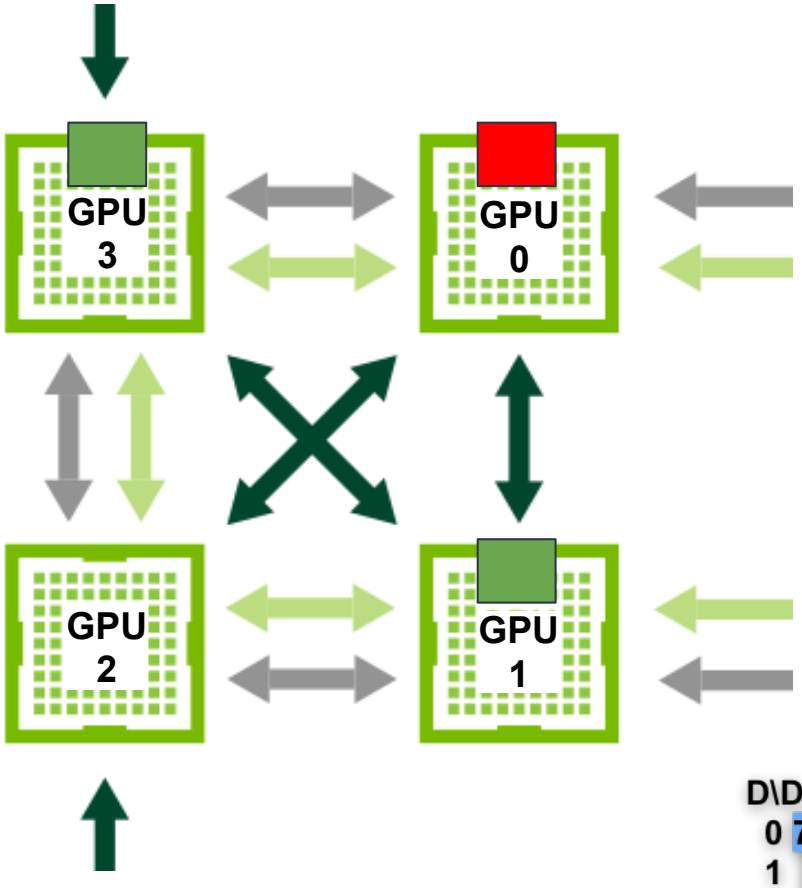


- Data is send by the GPU_3 as soon as it got a valid copy



Transfer will start once the data becomes valid on GPU_3

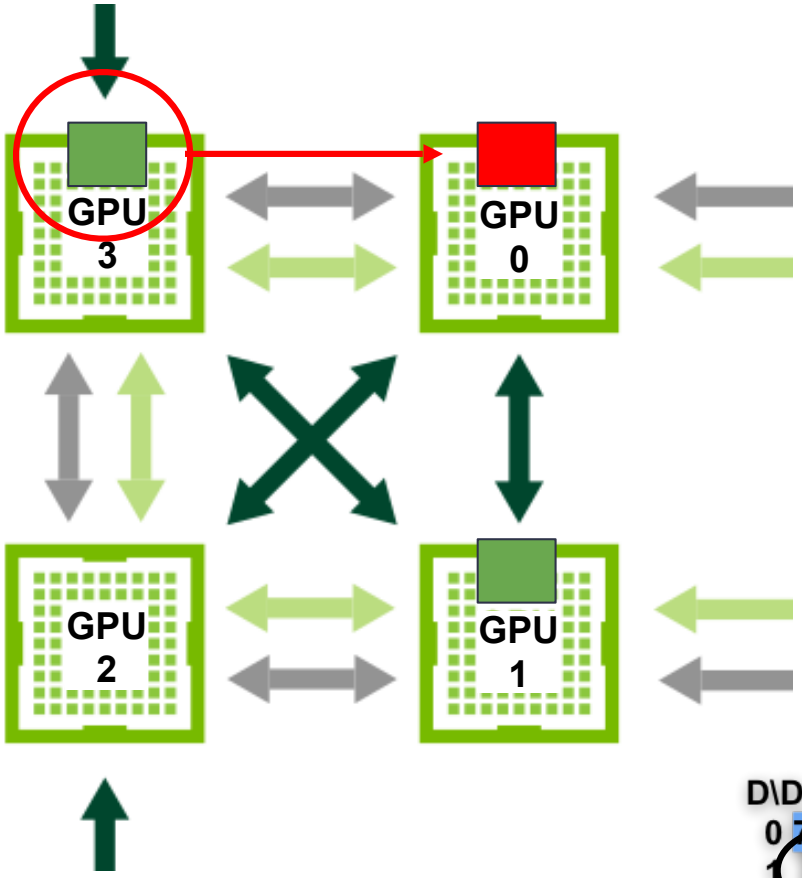
Topology-aware D-to-D transfer



D\D	0	1	2	3	4	5	6	7
0	744.05	48.37	48.39	96.49	96.45	17.11	17.74	17.97
1	48.38	750.48	96.50	48.38	16.98	96.44	17.32	16.97
2	48.34	96.28	750.48	96.47	17.62	16.93	48.39	17.75
3	96.26	48.34	96.28	750.48	17.58	17.22	17.60	48.39
4	96.46	16.98	17.65	17.53	746.89	48.39	48.40	96.49
5	16.94	96.42	16.88	17.21	48.39	745.47	96.51	48.40
6	17.65	16.90	48.40	17.51	48.34	96.47	750.48	96.47
7	17.80	16.91	17.77	48.39	96.28	48.38	96.28	747.61

xx : local GPU
xx : 2 NVLinks
xx : 1 NVLink
xx : PCIe

Topology-aware D-to-D transfer



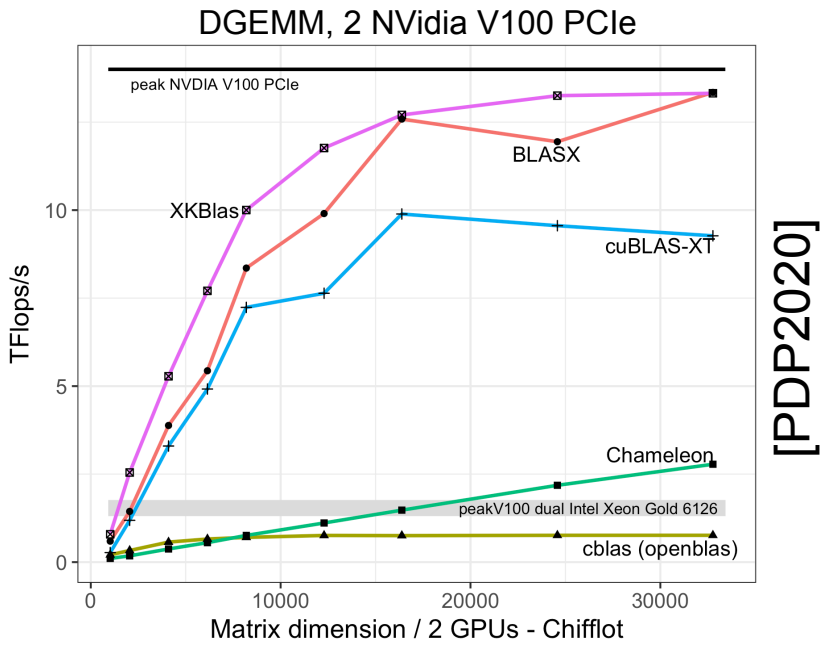
- if several valid copies exist, then always select the link with the fastest performance

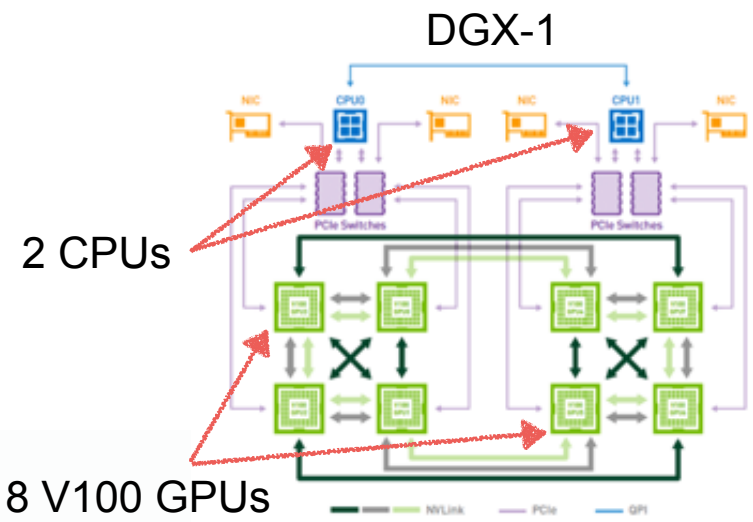
D\D	0	1	2	3	4	5	6	7
0	744.85	48.37	48.39	96.49	96.45	17.11	17.74	17.97
1	48.38	750.48	96.50	48.38	16.98	96.44	17.32	16.97
2	48.34	96.28	750.48	96.47	17.62	16.93	48.39	17.75
3	96.26	48.34	96.28	750.48	17.58	17.22	17.60	48.39
4	96.46	16.98	17.65	17.53	746.89	48.39	48.40	96.49
5	16.94	96.42	16.88	17.21	48.39	745.47	96.51	48.40
6	17.65	16.90	48.40	17.51	48.34	96.47	750.48	96.47
7	17.80	16.91	17.77	48.39	96.28	48.38	96.28	747.61

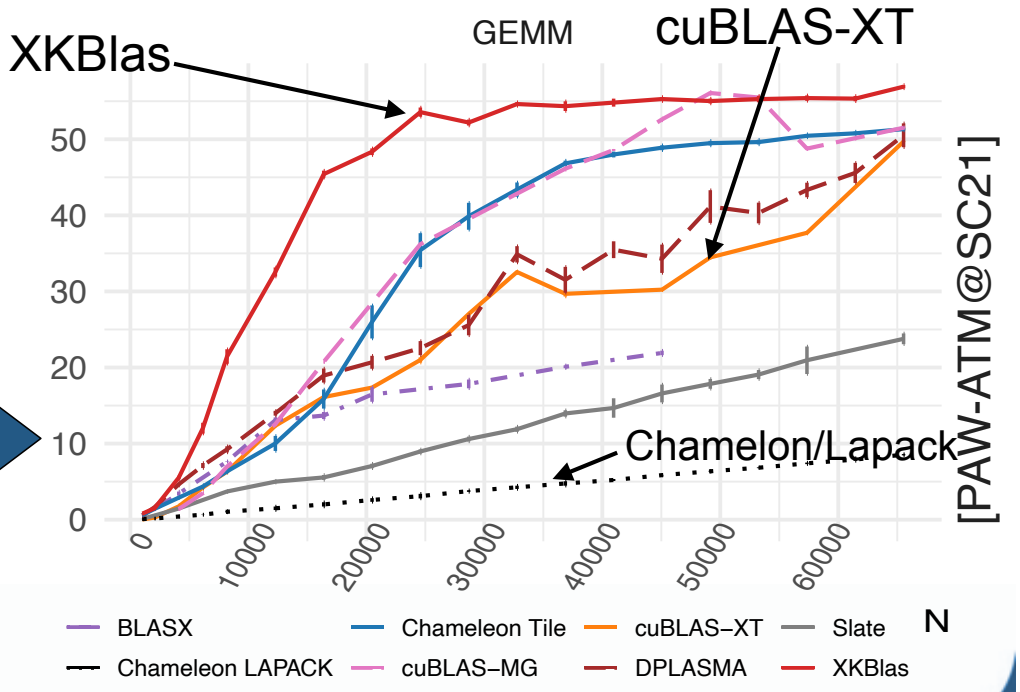
xx : local GPU
xx : 2 NVLinks
xx : 1 NVLink
xx : PCIe

Performances

- XKBlas [2018-]
  - [PDP 2020] first presentation & comparizon
    - non blocking API for better HW utilization
    - LAPACK memory layout
    - Separation of concern : computation & transfer
  - [PAW-ATM@SC21]
    - optimistic heuristic to filling software cache
    - topology aware data transfer







Porting to AMD GPU

- Simple process
  - replace the historical XKaapi CUDA driver impl. by a CUDA runtime impl.
  - convert CUDA to HIP thanks to the AMD hipify tool
- Test with multi-GPUs AMD MI50 [Patrick talk yesterday]
- **Ongoing work: testing on MI250X** before releasing the official AMD GPU compliant version of XKBlas (< 2024)
  - on ADASTRA thanks to Cines

Conclusion & perspectives

- XKBlas = a BLAS library on top of XKaapi
  - asynchronous API
  - tile algorithms from Chameleon
  - multi-GPUs efficiency thanks to XKaapi

- Ongoing work (<2024)
  - performance evaluation of AMD GPUs
    - MI250x , infinity fabric network

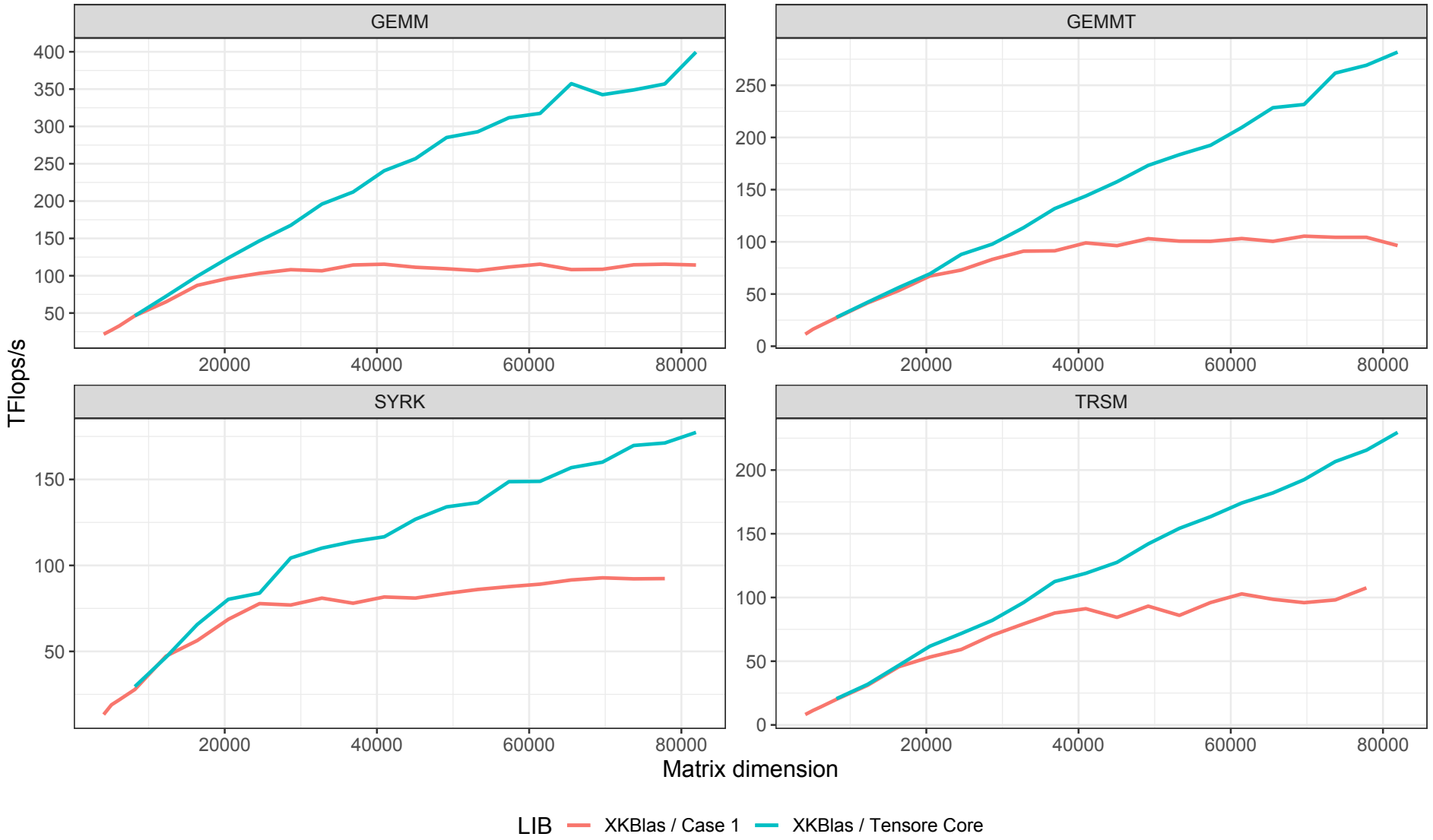
- Optimization
  1. tile size per kernel: performance = f(dimension, #GPU)
  2. better management of dependencies
  3. new GPU MUMPS kernels (CopyScale) to reduce the ‘critical communication path’
  4. partitioning GPUs if L0 thread is enabled ?

- 1 engineer 12 months

Thanks you!

Overview GEMM, GEMMT, SYRK, TRSM with TC

- DGX-1, 8 GPUs, cuda 10.1, driver 418.67, data on host
  - Kernel in FP32 with GEMM accelerated with TC (Compute FP16, Accumulation FP32)
  - Case 1 - Raw performance (red) versus Case 1 - Raw performance with TensorCore (blue)



XKBLAS with Tensor Core

- Accelerating computation of mix-precision algorithms
- Pragmatic approach: Accelerating tile SGEMM
  - FP32 Tile operands, accumulation in FP32
  - 2x more data to communicate: FP32 Tiles
- API mimic cuBLAS set_modemath

```
/* 1/ change modemath for all next kernels */
xkblas_set_modemath(XKBLAS_TENSOR_OP_MATH);

/* 2/ do non blocking dgemm: */
xkblas_dgemm_async( CblasNoTrans, CblasNoTrans, M, N, K,
  &alpha, A, lda,
          B, ldb,
  &beta,  C, ldc);

/* 3/ do non blocking coherent update of C on the host */
xkblas_memory_coherent_async(uplo, memflag, M, N, C, ldc, sizeof(double));

/* 4/ wait completion of previous asynchronous operations */
xkblas_sync();
```