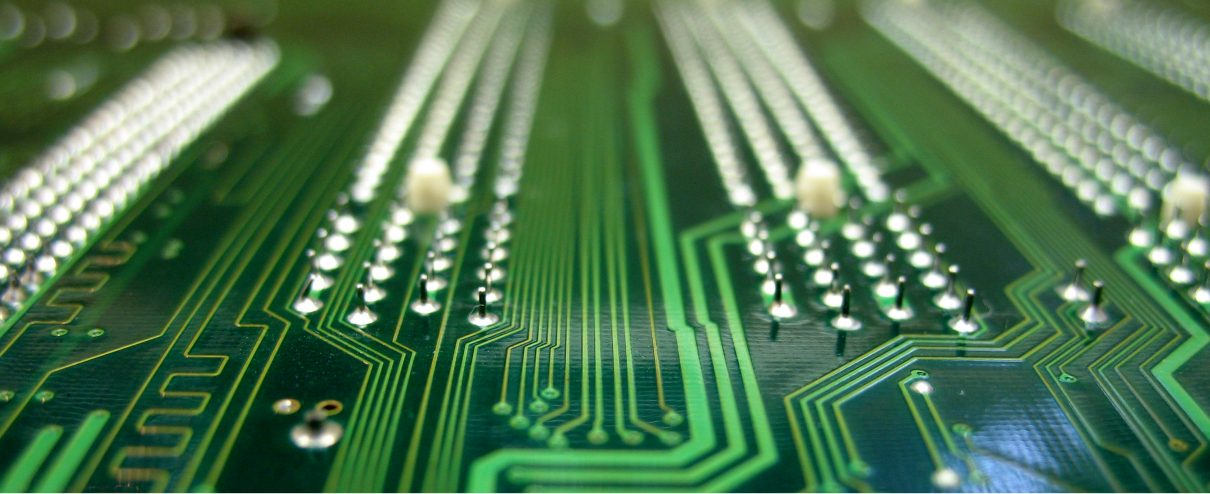
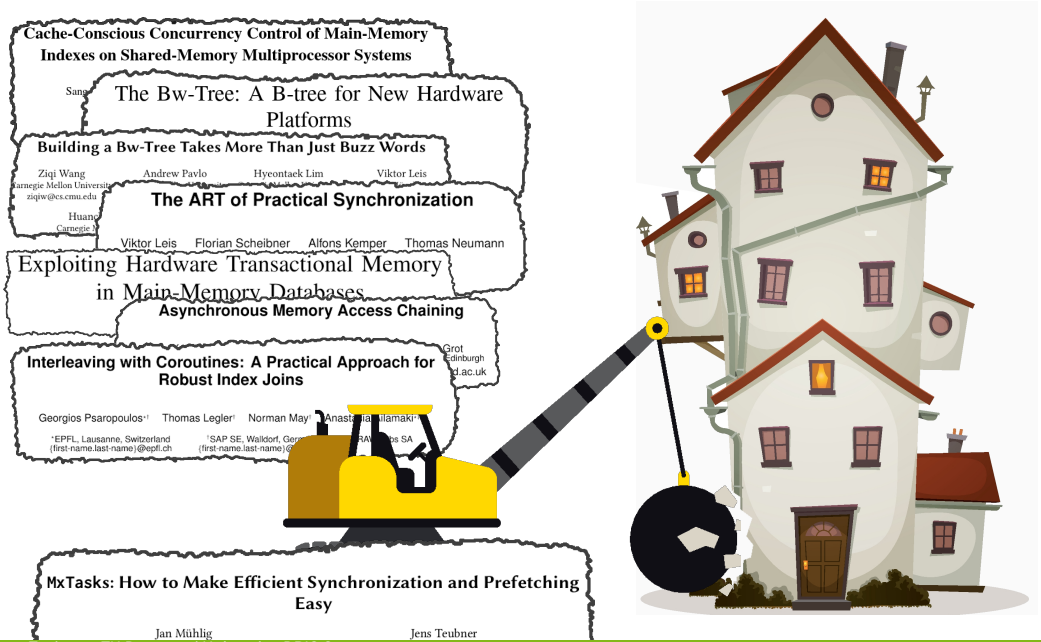


MxTasks: How to Make Efficient Synchronization and Prefetching Easy

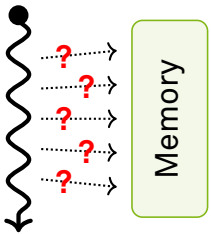
Jan Mühlig, Jens Teubner · TU Dortmund University

technische universität dortmund


Cache-Conscious Concurrency Control of Main-Memory Indexes on Shared-Memory Multiprocessor Systems
The Bw-Tree: A B-tree for New Hardware Platforms
Building a Bw-Tree Takes More Than Just Buzz Words
Ziqi Wang
Andrew Pavlo
Hyeontaek Lim
Viktor Leis
Carnegie Mellon University
ziqiw@cs.cmu.edu
The ART of Practical Synchronization
Viktor Leis
Florian Scheibner
Alfons Kemper
Thomas Neumann
Exploiting Hardware Transactional Memory in Main-Memory Databases
Asynchronous Memory Access Chaining
Interleaving with Coroutines: A Practical Approach for Robust Index Joins
Georgios Psaropoulos
Thomas Legler
Norman May
*EPFL, Lausanne, Switzerland
{first-name.last-name}@epfl.ch
MxTasks: How to Make Efficient Synchronization and Prefetching Easy
Jan Mühlig
Jens Teubner

Are threads the right abstraction here at all?

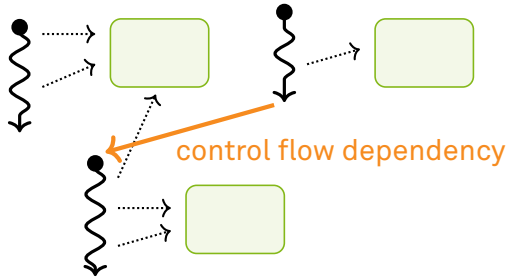
- Behavior of tasks **opaque to OS scheduler**.
- Workarounds:
  - Thread pinning, NUMA placement, lock tuning, …
- What if the database is **not alone** on the system?



MxKernel: tasks (“MxTasks”) as the central abstraction

- Small **units of work**
- Annotated with **meta data**
  - Accessed memory regions, dependencies, runtime characteristics, ...

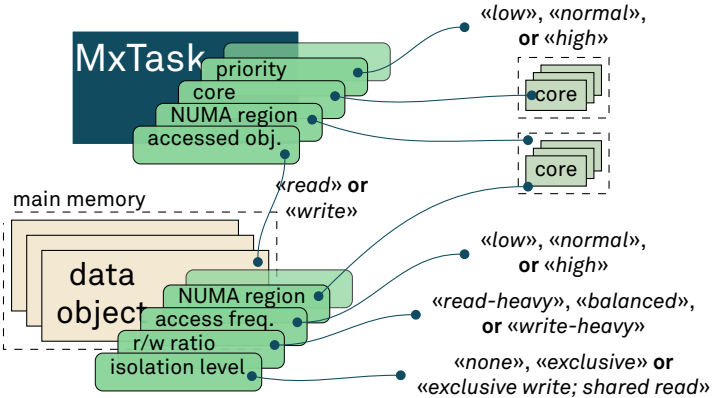
→ Meta data **accessible to scheduler**



Lots of opportunities:

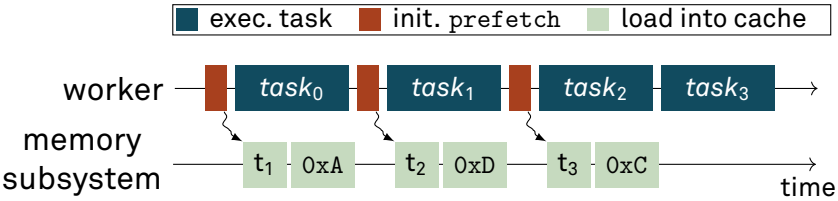
- Locality awareness, heterogeneity, informed scheduling decisions, ...
- **Today: prefetching, annotation-based synchronization**

Annotations to tasks and data objects



Annotation-Based Prefetching

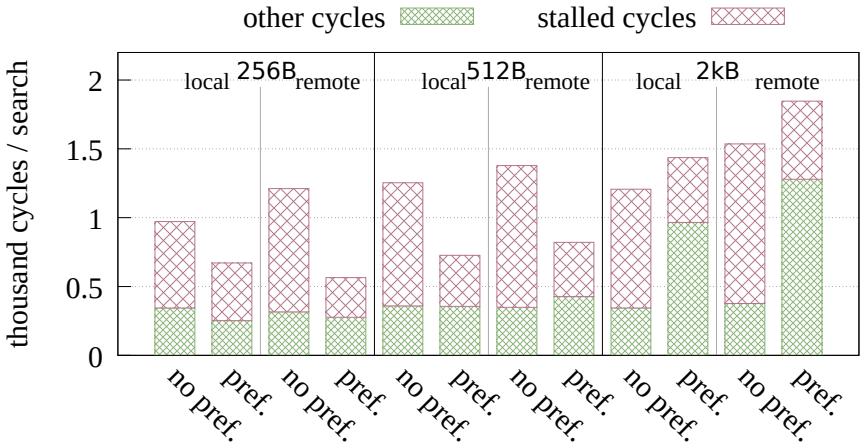
- Scheduler **prefetches** data on application's behalf.



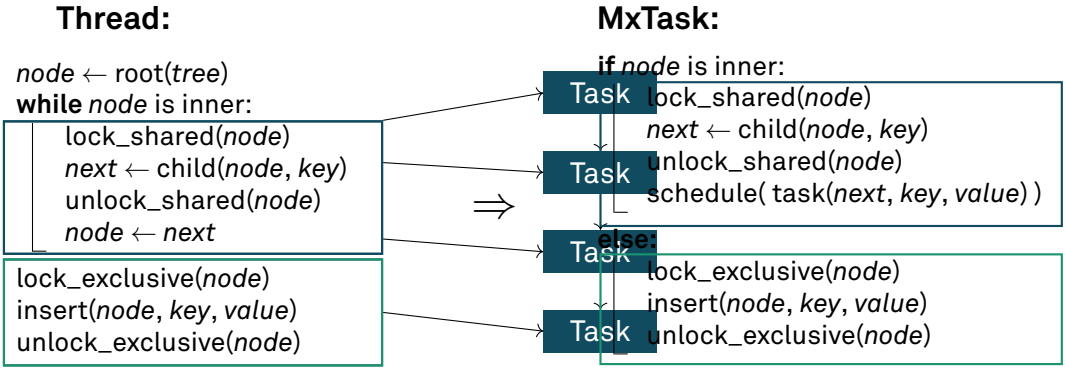
→ Hardware aspects (only) in scheduler/kernel (where they belong!)

→ Prefetch **across** applications (⇝ cloud, multi-tenant)

Microbenchmark: Binary Search

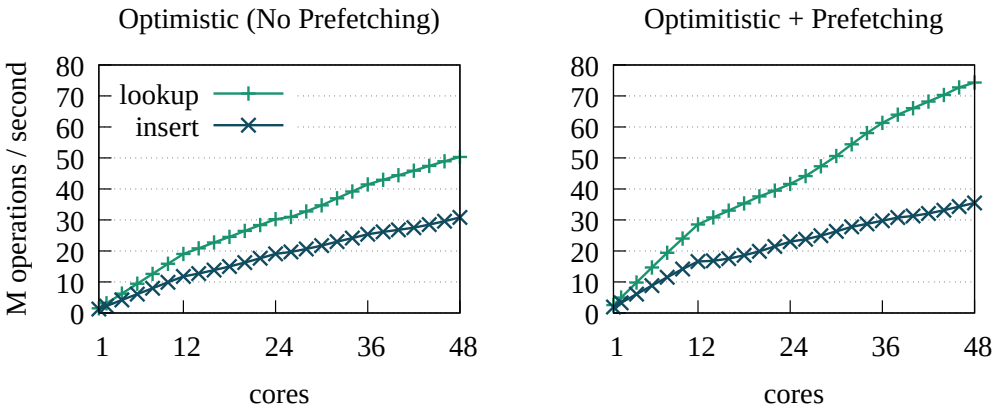


Task-Based Example: Tree Insertion



→ Transforming thread-based code into tasks is often straightforward.

Task-Based Prefetching: B^{link}-tree



- This is without any tweaking or explicit prefetching in the user code!

Actually…

MxTask:

if *node* is inner:

```
lock_shared(node)
next ← child(node, key)
unlock_shared(node)
schedule( task(next, key, value) )
```

else:

```
lock_exclusive(node)
insert(node, key, value)
unlock_exclusive(node)
```

Annotation-Based Synchronization:

- Tasks do **not** explicitly (un)lock data.
- **Instead:** Request isolation through **annotations**.
- Scheduler will fulfill those requests.

→ **Easy** for programmer

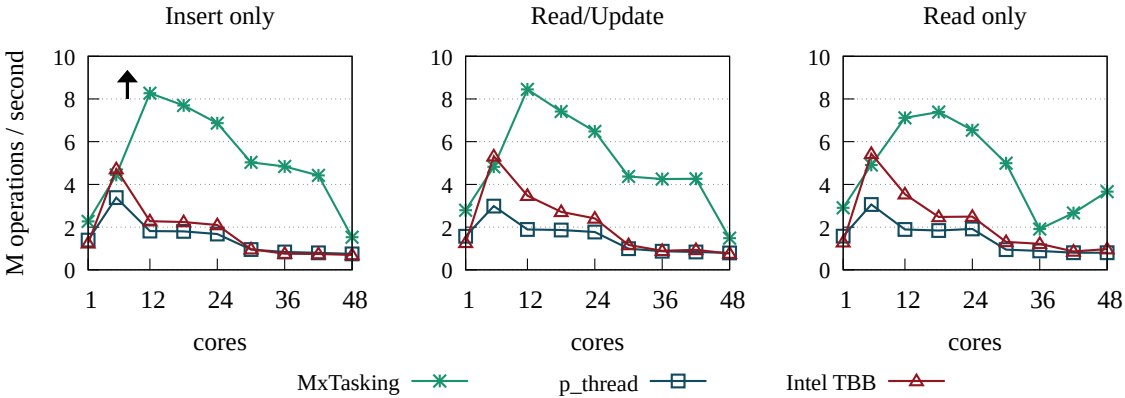
→ Scheduler will flexibly select **appropriate synchronization mechanism**.

Annotation-Based Synchronization

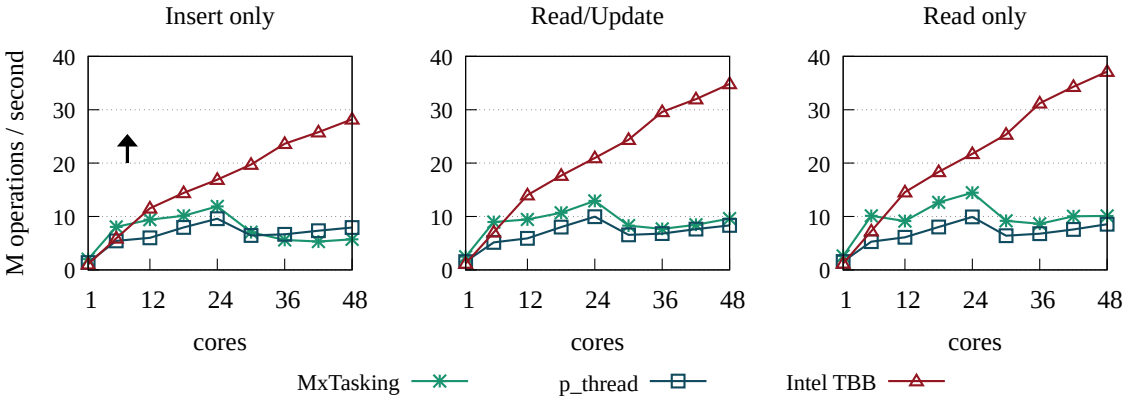
Scheduler will select suitable **synchronization mechanisms**, such as

1. **Lock-Based Synchronization**
   MxKernel offers simple spinlocks or reader/writer locks.
2. **Optimistic Versioning**
   Maintain version counter to detect conflicts. Restart upon a conflict.
3. **Synchronization through Scheduling**
   Scheduler will make sure that no conflicting tasks get scheduled in parallel. Specifically: run tasks without preemption, schedule accesses to same object on same CPU core.

B^{link}-tree: Synchronization through Scheduling vs. Spinlocks

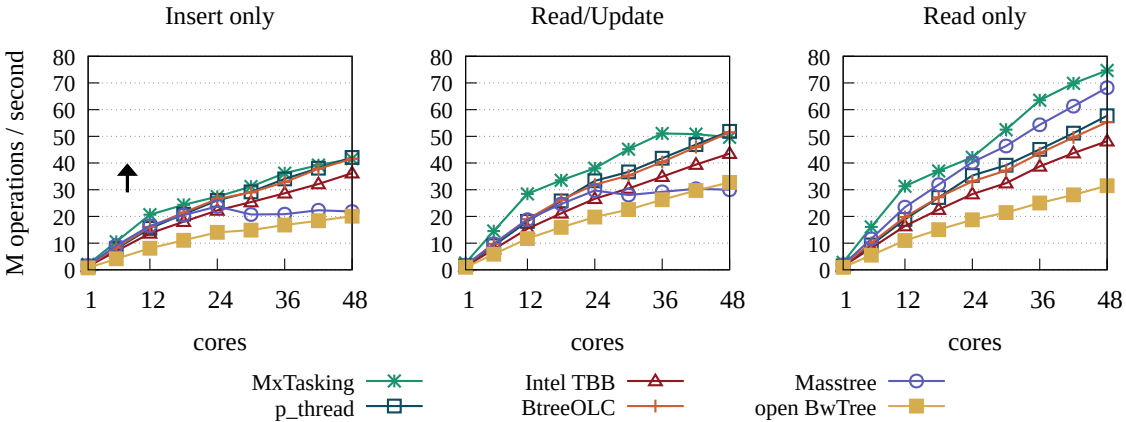


B^{link}-tree: Lock-Based Synchronization (r/w locks)



- MxTasking not (yet) NUMA aware.
- Advantage over pthreads is due to **prefetching**.

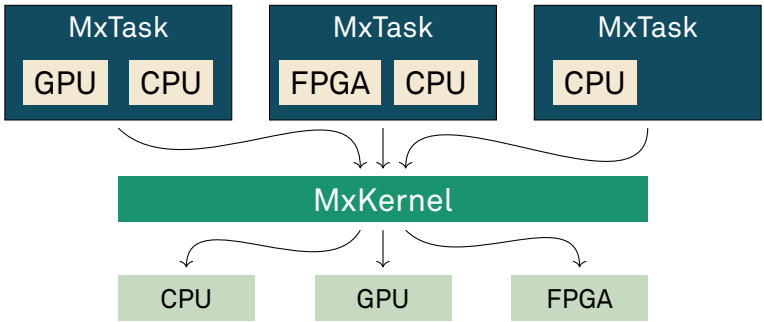
B^{link}-tree: Optimistic Versioning



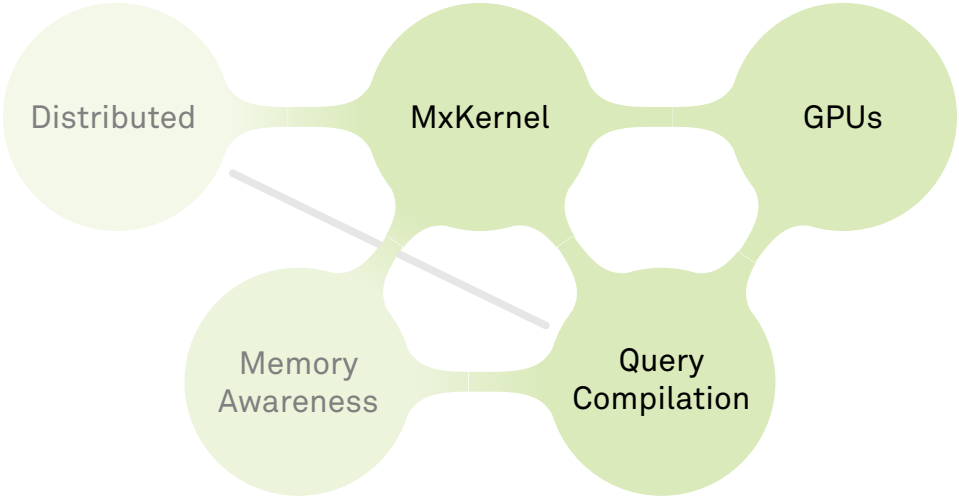
Looking Ahead

User-land **MxTasking** library for Linux; “bare-metal” system **MxKernel**.

The task abstraction may help to overcome the challenge of leveraging **heterogeneous hardware**:

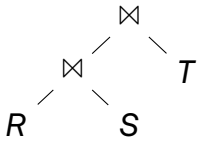


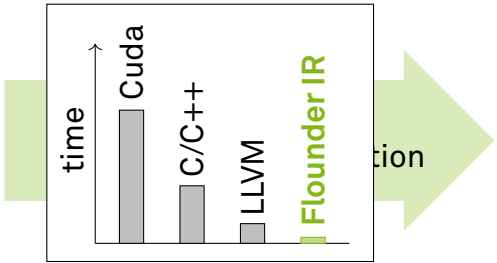
Research Efforts Connected to MxKernel



SQL → Machine Code Compilation

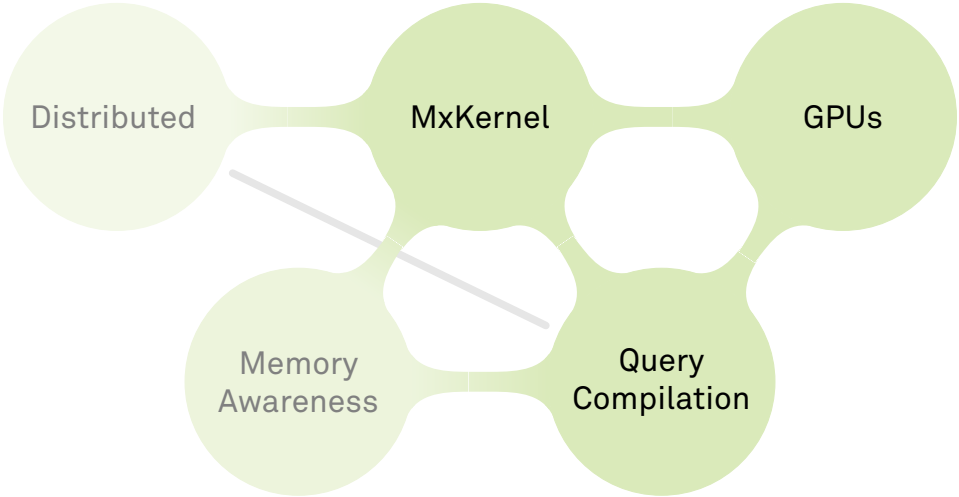
```
SELECT *
  FROM R, S, T
 WHERE r_key = s_rkey
   AND s_key = t_skey
```






```
48 89 e5
53
48 83 ec 08
80 3d 80 0b 20
75 4b
bb 30 0e 60 00
4c 89 fa
4c 89 f6
44 89 ef
41 ff 14 dc
41 5f
c3
```

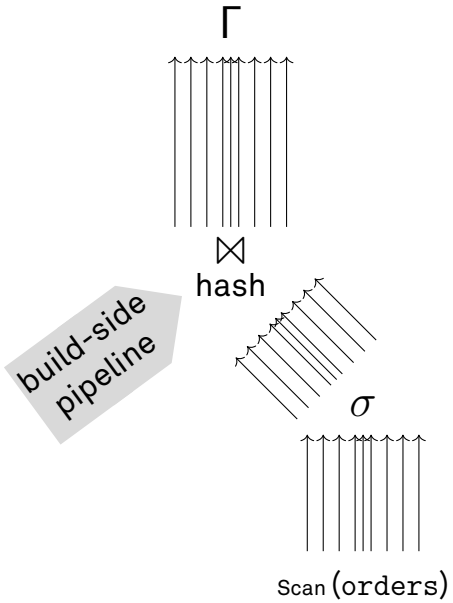
Research Efforts Connected to MxKernel



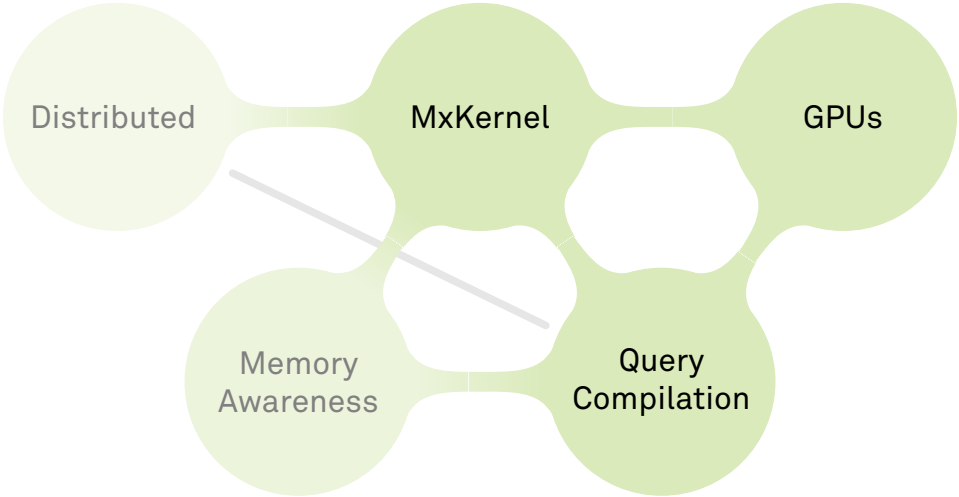
Query Compilation and GPUs

- **Data-parallel** execution model
- Threads within one **warp** execute **same** instruction
  ⇝ 32 “lanes” (↑↑↑↑)

Tuple execution paths may vary.



Research Efforts Connected to MxKernel



Summary

- ~~Threads~~ $\rightarrow$ **Tasks** ($\rightsquigarrow$ “MxTasks”)
- **Annotations** to describe task characteristics
- Today:
  - Annotation-based **Prefetching**
  - Annotation-based **Synchronization**
- Less **programmer effort**, better **hardware awareness**

Credits to **Jan Mühlig** (and to the rest of my team).

This work is supported by Deutsche Forschungsgemeinschaft (DFG).