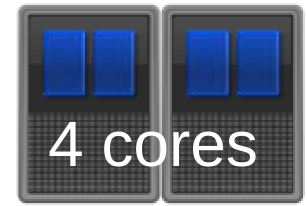
SQL/XML-IMDBg

GPU boosted
In-Memory Database
for
ultra fast data management

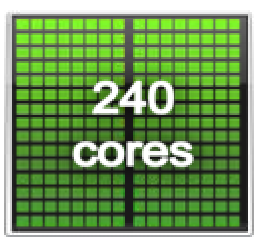
Harald Frick CEO

The parallel revolution…

Future computing systems are parallel, but



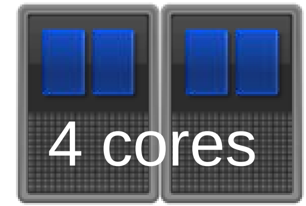
- Programmers have to code the parallelism
- Lake of development systems and experience
- Radical departure from general programming wisdom
- Requires combining CPU and GPU code
- Requires "close to metal" knowledge for max. performance
- The Hardware is evolving and changing rapidly



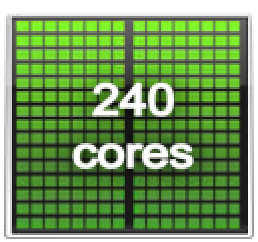
How are we going to program such systems ?

…changes everything

Database systems now are ubiquitous, but



- The core DB technology is from the 70s !
- Database systems now hitting a performance-wall
- Computer memory grows faster than enterprise data
- Hardware is evolving rapidly and gets more heterogeneous
- Parallel computing is a necessity for higher performance



How are we going to develop such systems ?

The Database has been re-invented

SQL/XML-IMDBg

GPGPU

Parallel In-Memory
Database Server

Complete re-engineering of the database architecture is required to make a DB kernel multi-core ready and scalable to 100's of processors

Unlimited scalability

Declarative parallel
data management
and processing

Cost effective
upgrade path

Bridging the technology gap with

GPGPU

SQL/XML-IMDBg

Database Server

Domain Experts can use a standard interface (SQL) for massive parallel computation and data mining

Unlimited scalability

Cost effective

Universal
Data management

Agenda (1)

- Overview about general database architecture
- Short introduction to SQL/XML-IMDB
- Overview of the re-engineering tasks performed to make the IMDB database kernel many-core ready
  - Three levels of re-engineering:
    1. General database architecture
    2. Relational algebra structures and functions
    3. Coding level

Agenda (2)

- **Architecture of the database kernel**
  Explains the overall design principle chosen to make a database kernel ready for massive parallel processing and GPU ready

- **Database Table structure and outline**
  Explains the vertical partitioning layout and why this is one of the most important pre-requisites for successful GPU co-processing in database kernels

- **Memory management**
  The memory management subsystem architecture and how to manage memory on a GPU device with missing dynamic memory management facilities

- **Query Optimizer**
  Gives insights into the Split-Work architecture and explains the Optimizer architecture with a special focus on the dynamic re-optimizing steps performed during query processing

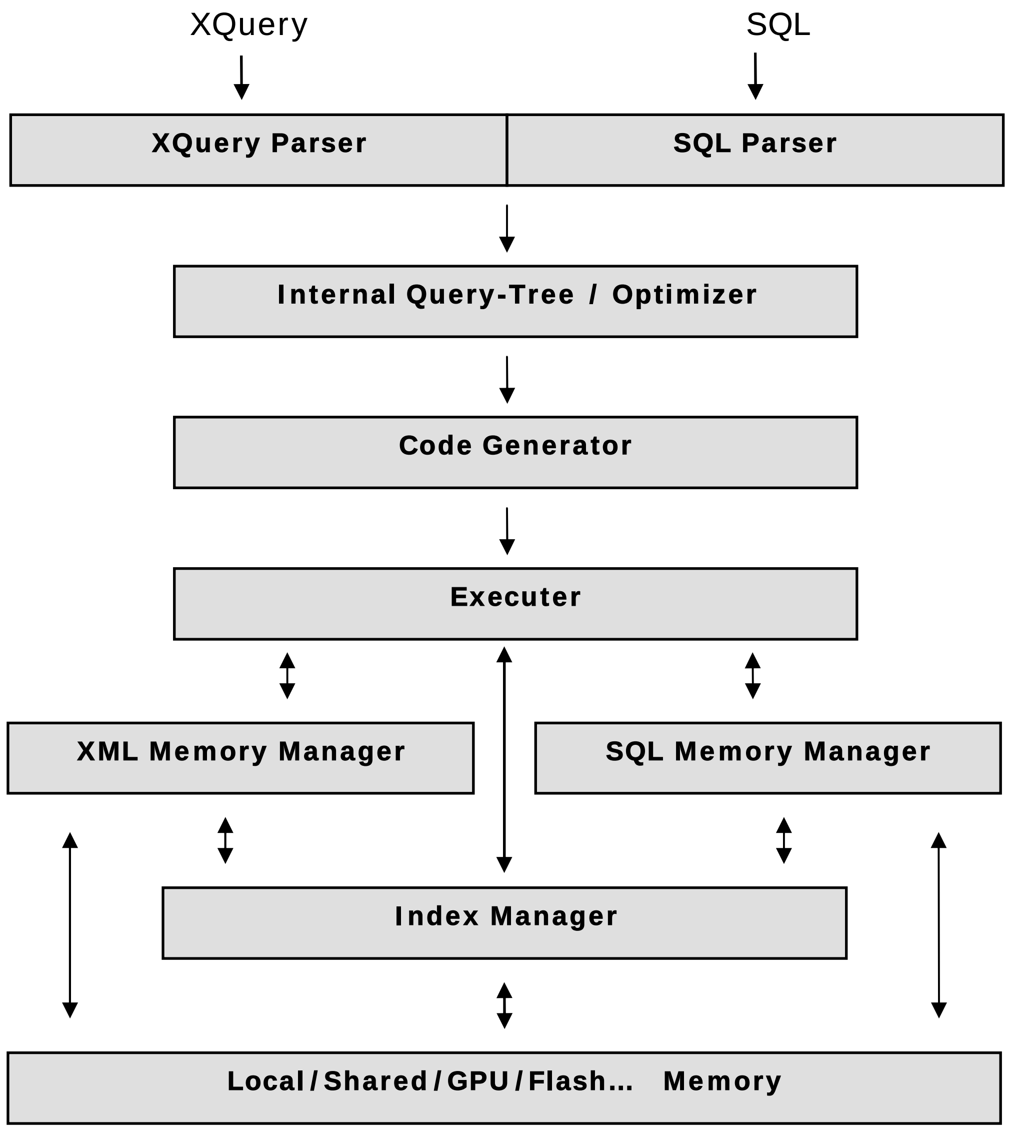
- **Query Executer structure**
  Explains the architecture of the query Executer with a special focus on processing the split-work plan for distributing the query work between CPU and GPU

Agenda (3)

Summary of presentation:

- Lessons learned from re-engineering the database kernel
- Some important conclusions drawn from our experiences regarding parallelizing and re-engineering a complex application for massive parallel platforms like GPU’s
- Outlook on future enhancements planned for the IMDB
- Questions

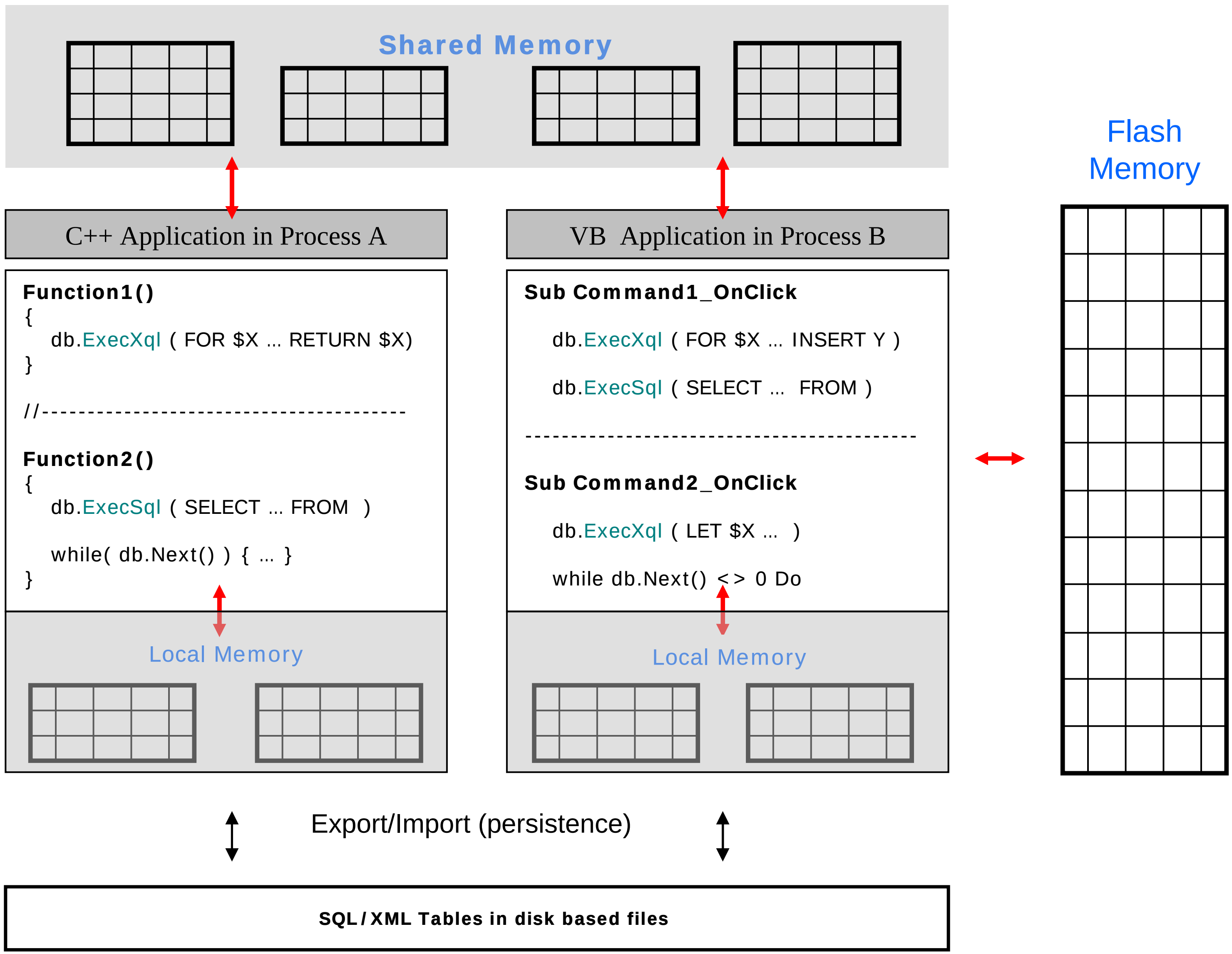
DB Architecture overview



SQL/XML-IMDb overview

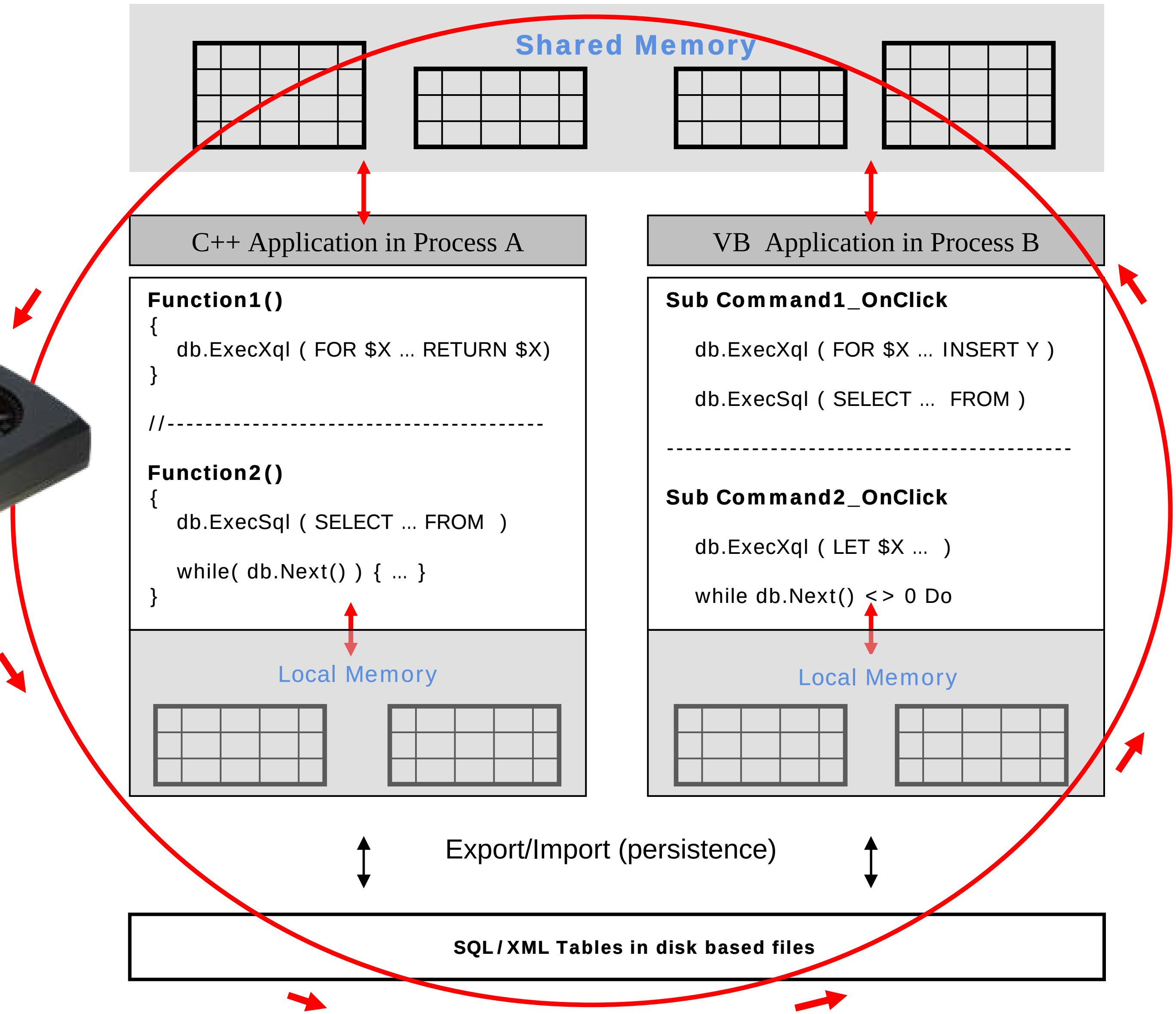
- DB Library (DLL, LIB, .NET Assembly)
- Application embedded SQL/XQuery statements
- Functional API
- Declarative universal data management
- Kind of declarative STL library
- Data exchange between process boundaries
- 8 years of market experience
- WIN32, WIN64, Linux32, Linux64, ...

SQL/XML-IMDB application practice



Vision: Integrating the GPU memory

GPU Co-Processor
+ Memory Tables



Vision: Seamless utilization of GPU power

- Automatically distribute SQL query work and relational data between standard CPU cores and high performance graphics GPU's

- In-Memory: CPU-Local Memory, CPU-Shared Memory, GPU-Memory

SQL/XML-IMDBg

Place tables (and indexes) where you want:

Create Table Local TR (…)
Create Table Shared TS (…)
Create Table GPU TG (…)

Select * From TR, TS, TG WHERE ….
Select TG.a * TG.b / TG.c From TR, TS, TG WHERE ….

Re-inventing the Database by complete re-engineering of the “old” IMDB

Re-Engineering (overall)

- **Vertical partitioned table structure**
- **Arrays instead of Lists, Hash-Tables, Trees …**
- Compression (dictionary based)
- Regular shaped data structures
- Only one Index structure
- Vector processing of Algebra functions
- Fork-Join model (OpenMP)

Re-Engineering (coding)

- Reduced program code complexity
  - Instruction cache friendlier
  - Reduced memory footprint
  - From C++ back to C
- Arrays everywhere
  - Supports memory prefetch
  - Simple access functions
  - Loop unrolling
  - Simpler buffer management
- Dynamic, array like SQL data tables
  - No space waste (no static pre-allocation)
  - Strings represented by ID's whenever possible
  - Simple access/scan functions
- Simplified index structures
  - Array like
  - Better parallelizable
  - Co-Processor friendly

3 essentials for ultra high performance

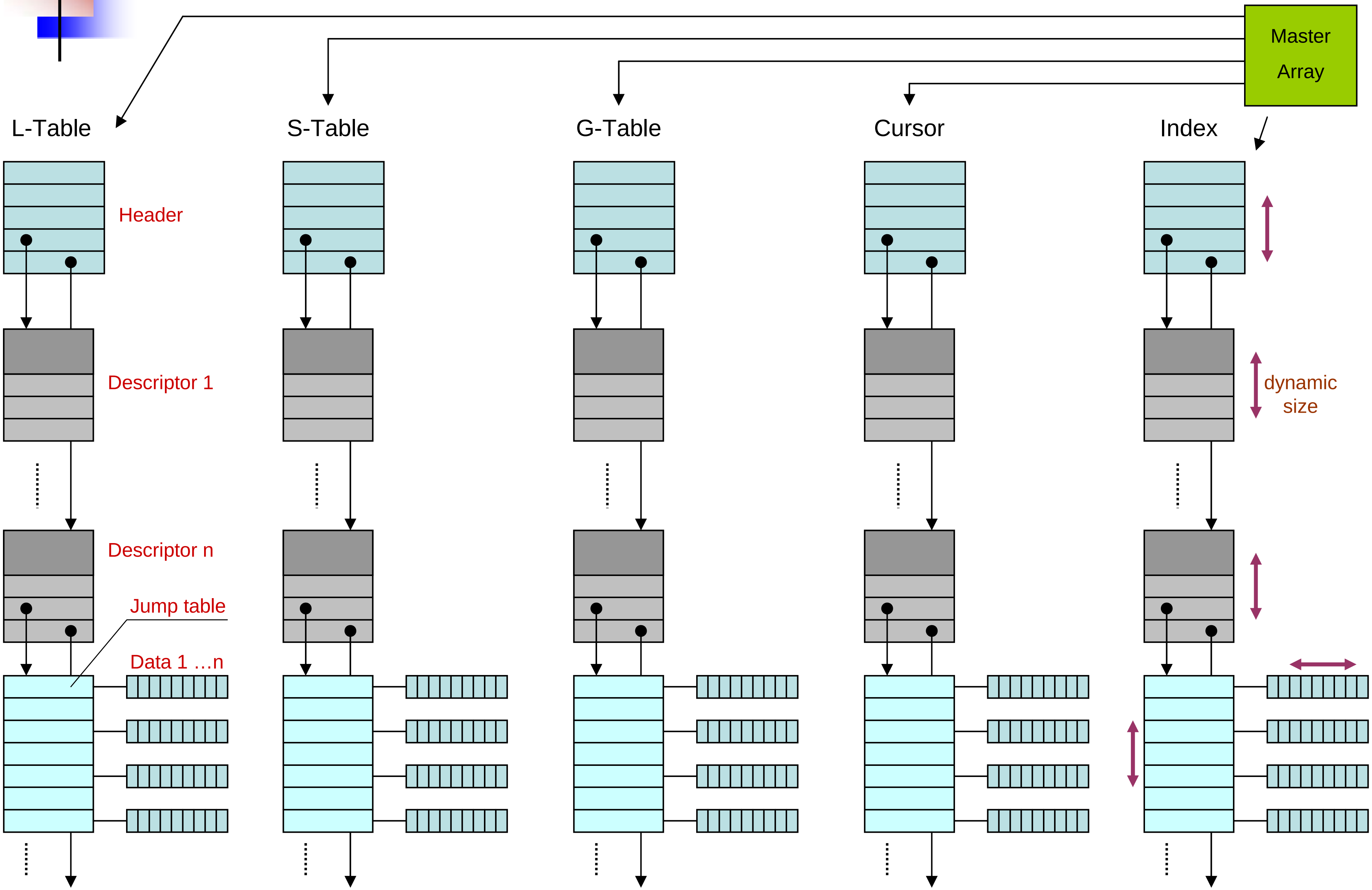
- *RISC* like DB core structure
  - Simple and repeating data structures
  - Dynamic Arrays everywhere
  - Dramatically reduced DB-kernel complexity
  - Favors parallel algorithmic structures !
- Split-Work Optimizer
  - Dynamic mid-query re-optimizing
  - Schedules query execution between CPU and GPU
- Split-Work Query Executer
  - Work duplication and parallel execution
  - Materialized intermediate results (compressed bitmaps)
  - Vector processing of relational algebra

Agenda (2)

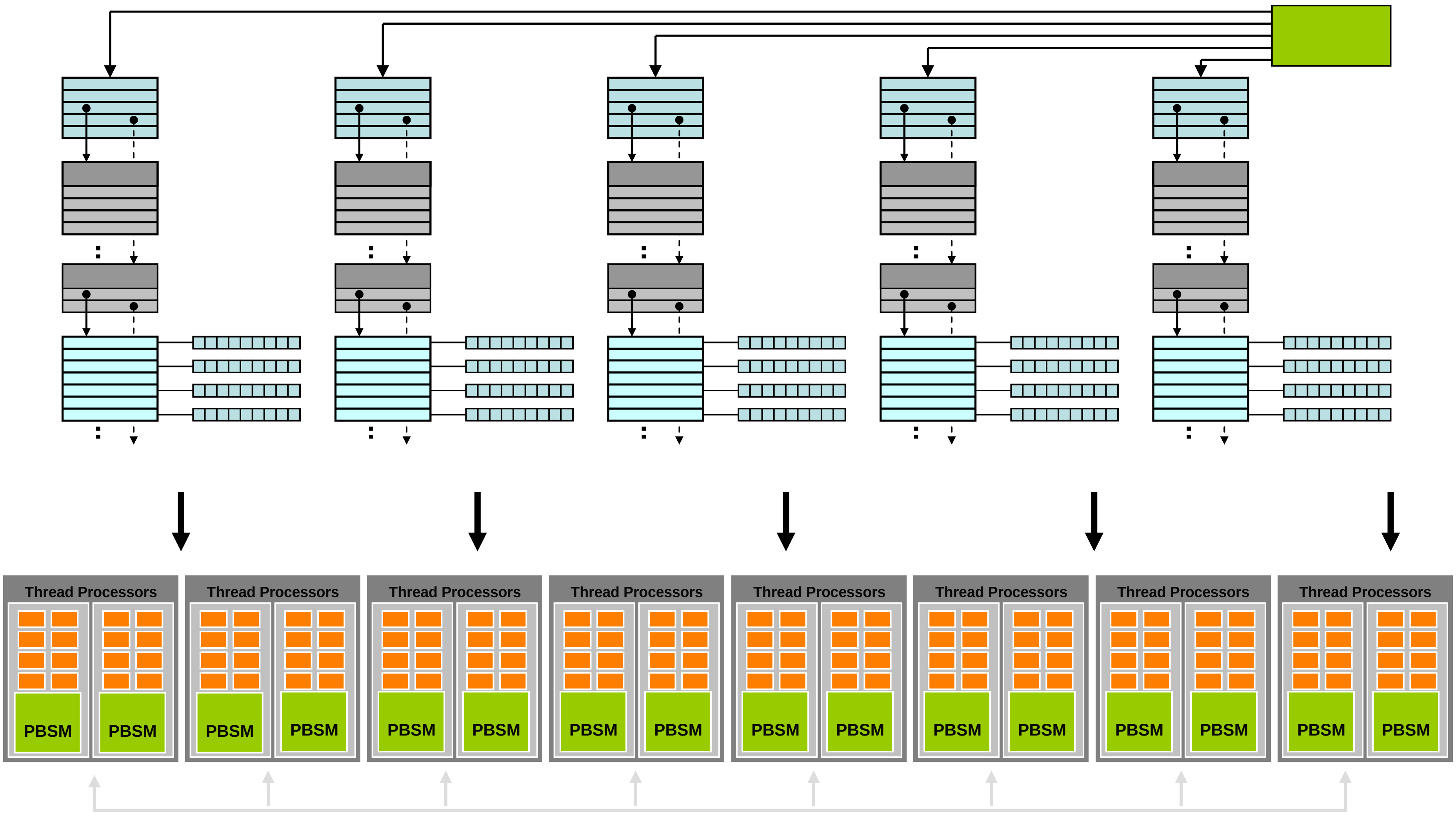
- **General architecture outline of database kernel**
  Explains the overall design principle chosen to make a database kernel ready for massive parallel processing and GPU ready
- **Database Table structure and outline**
  Explains the vertical partitioning layout and why this is one of the most important pre-requisites for successful GPU co-processing in database kernels
- **Memory management**
  The memory management subsystem architecture and how to manage memory on a GPU device with missing dynamic memory management facilities
- **Query Optimizer**
  Gives insights into the Split-Work architecture and explains the Optimizer architecture with a special focus on the dynamic re-optimizing steps performed during query processing
- **Query Executer structure**
  Explains the architecture of the query Executer with a special focus on processing the split-work plan for distributing the query work between CPU and GPU

Regular Shaped Structures
and
Dynamic Arrays
everywhere

DB Architecture (core layout)



Very good DB / GPU structure fit



Agenda (2)

- General architecture outline of database kernel
  Explains the overall design principle chosen to make a database kernel ready for massive parallel processing and GPU ready

- **Database Table structure and outline**
  Explains the vertical partitioning layout and why this is one of the most important pre-requisites for successful GPU co-processing in database kernels

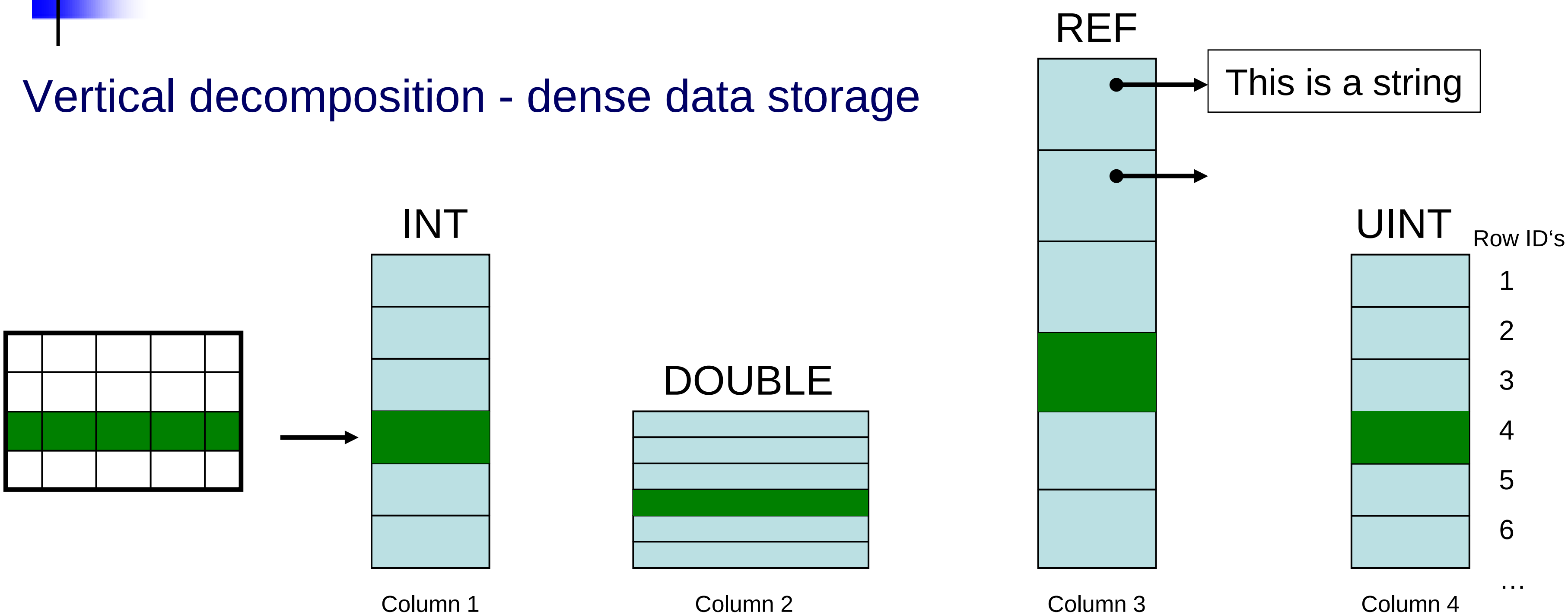
- Memory management
  The memory management subsystem architecture and how to manage memory on a GPU device with missing dynamic memory management facilities

- Query Optimizer
  Gives insights into the Split-Work architecture and explains the Optimizer architecture with a special focus on the dynamic re-optimizing steps performed during query processing

- Query Executer structure
  Explains the architecture of the query Executer with a special focus on processing the split-work plan for distributing the query work between CPU and GPU

Table Layout (1)

Vertical decomposition - dense data storage



12345678 …. ... 32

Column Example:

Bool Data Type

	Row ID's
10100011010010010000111011000010	32
10100011010010010000111011000010	64
10100011010010010000111011000010	96
10100011010010010000111011000010	128
⋮	…

Array like columns allow:

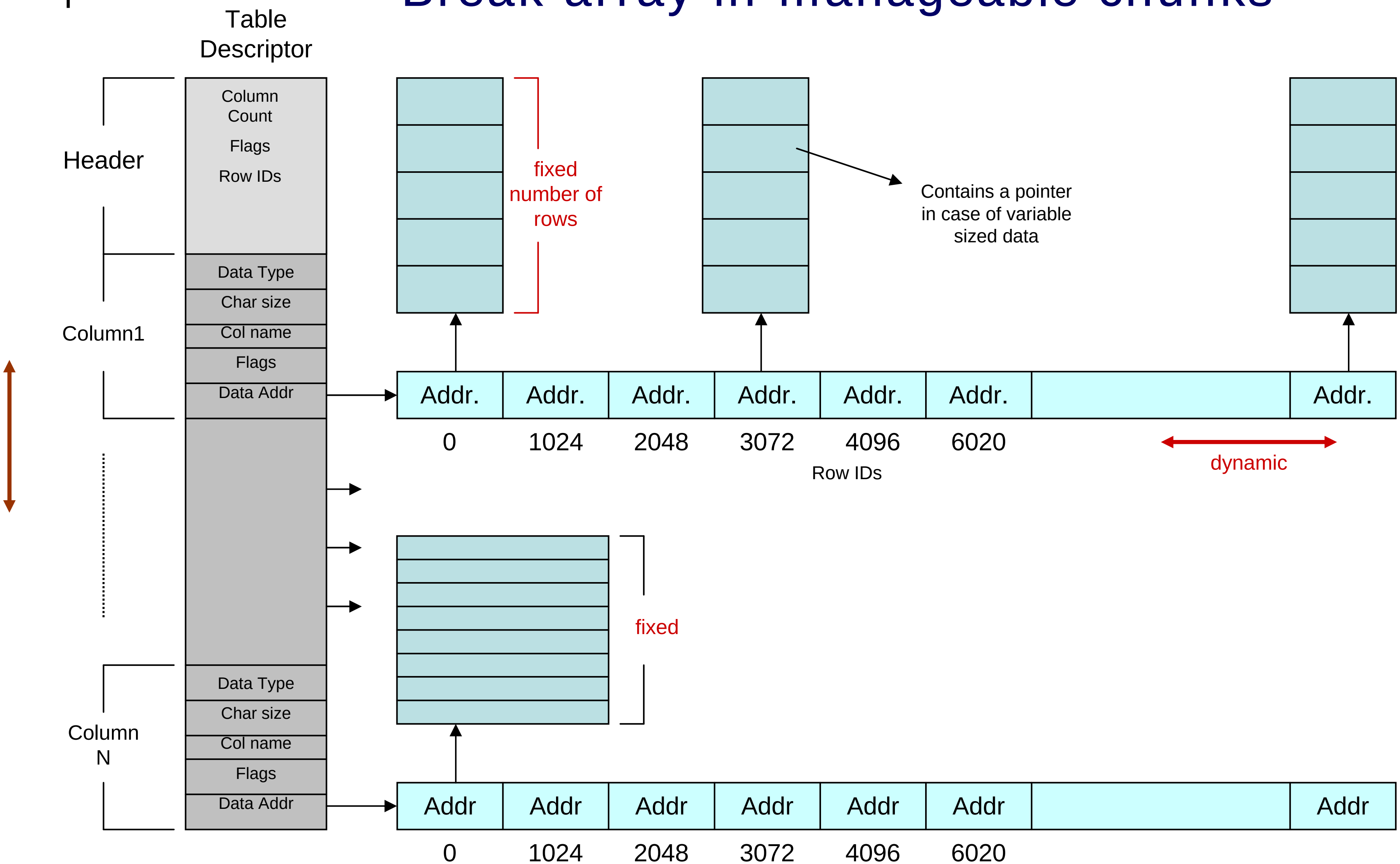
- Coalesced access on GPU
- CPU cache friendly access
- Distributed table layout

Big Problem

How to manage huge sized arrays

Table Layout (2)

Break array in manageable chunks



Agenda (2)

- General architecture outline of database kernel
  Explains the overall design principle chosen to make a database kernel ready for massive parallel processing and GPU ready

- Database Table structure and outline
  Explains the vertical partitioning layout and why this is one of the most important pre-requisites for successful GPU co-processing in database kernels

- Memory management
  The memory management subsystem architecture and how to manage memory on a GPU device with missing dynamic memory management facilities

- Query Optimizer
  Gives insights into the Split-Work architecture and explains the Optimizer architecture with a special focus on the dynamic re-optimizing steps performed during query processing

- Query Executer structure
  Explains the architecture of the query Executer with a special focus on processing the split-work plan for distributing the query work between CPU and GPU

Memory management (CPU)

Local Memory pool

Win-OS

VirtualAlloc

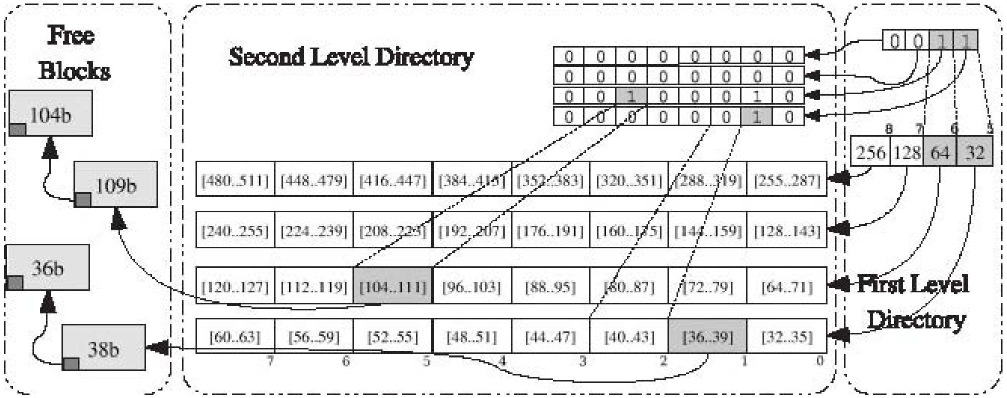
Shared Memory pool

Flash Memory pool

TLFS Allocator (Two-Level Segregate Fit)

POOL 1 …n

- Bitmap based chunk management
- Simple code structure (instruction cache!)
- **Memory pools** based !
- Fastest allocator for shared memory
- Good cache locality
- Small size



Source:
Implementation of a constant-time dynamic storage allocator.
Miguel Masmano, Ismael Ripoll, et al. Software: Practice and Experience. Volume 38 Issue 10, Pages 995 - 1026. 2008.

Memory management (GPU)

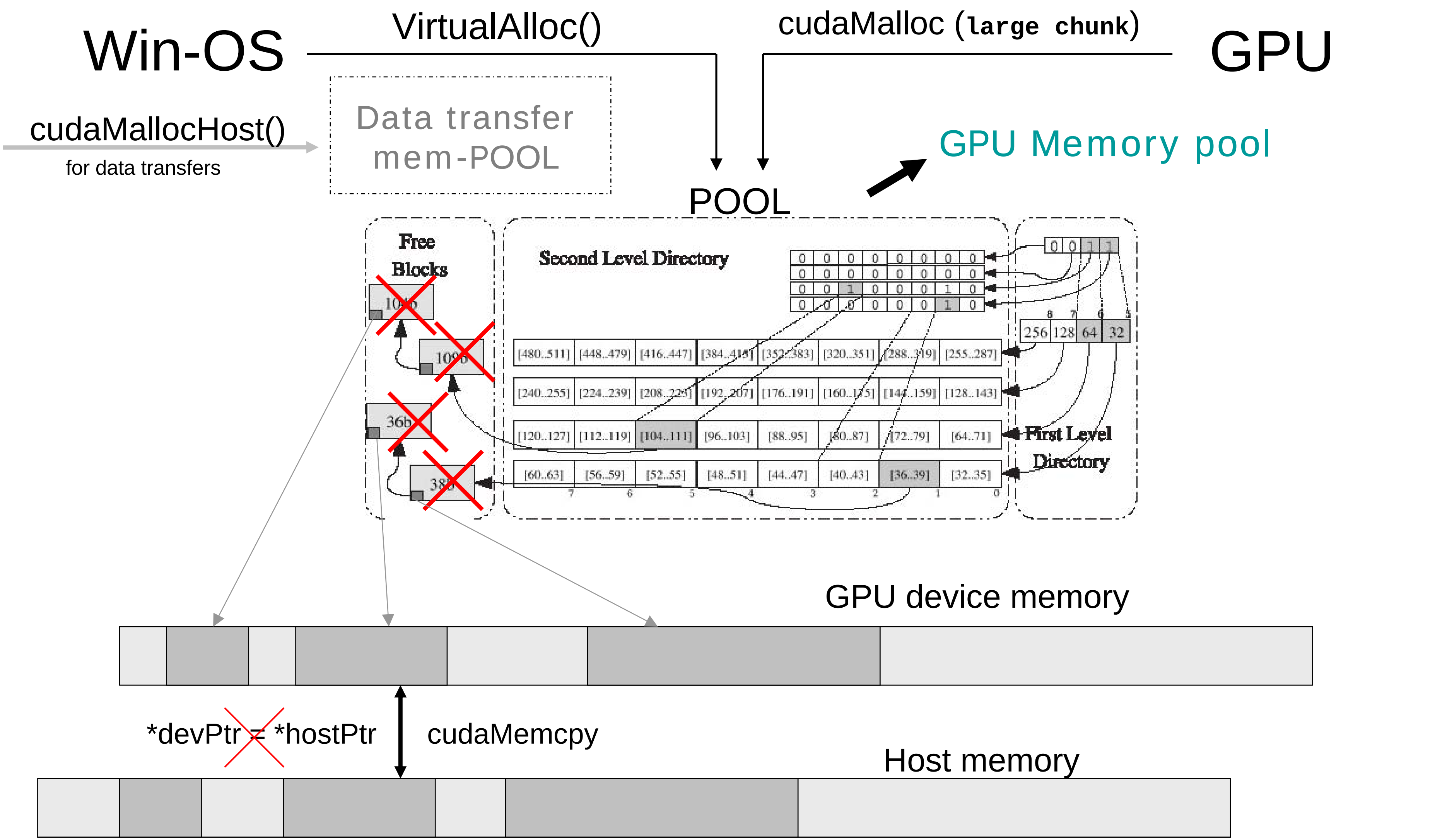
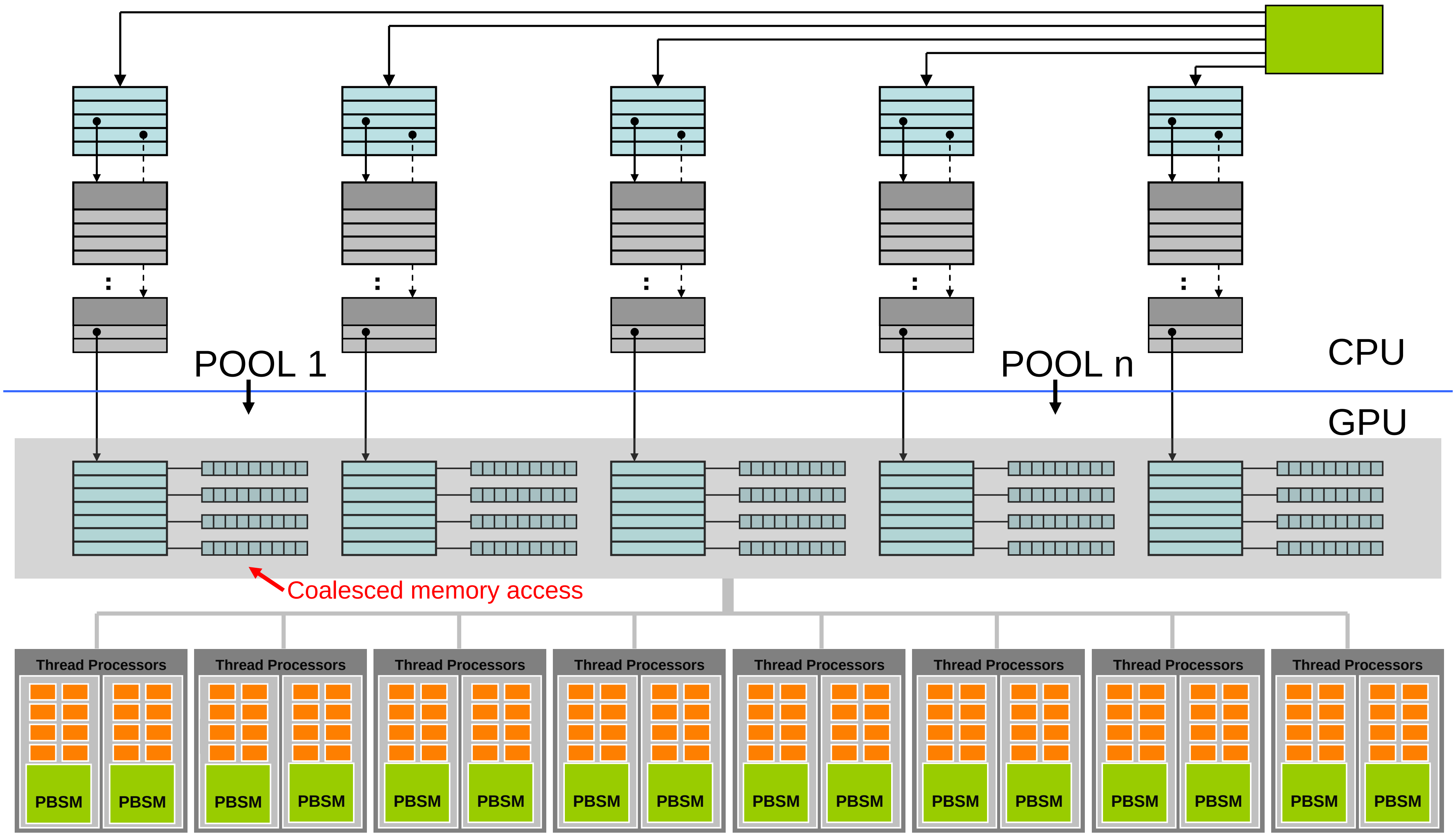
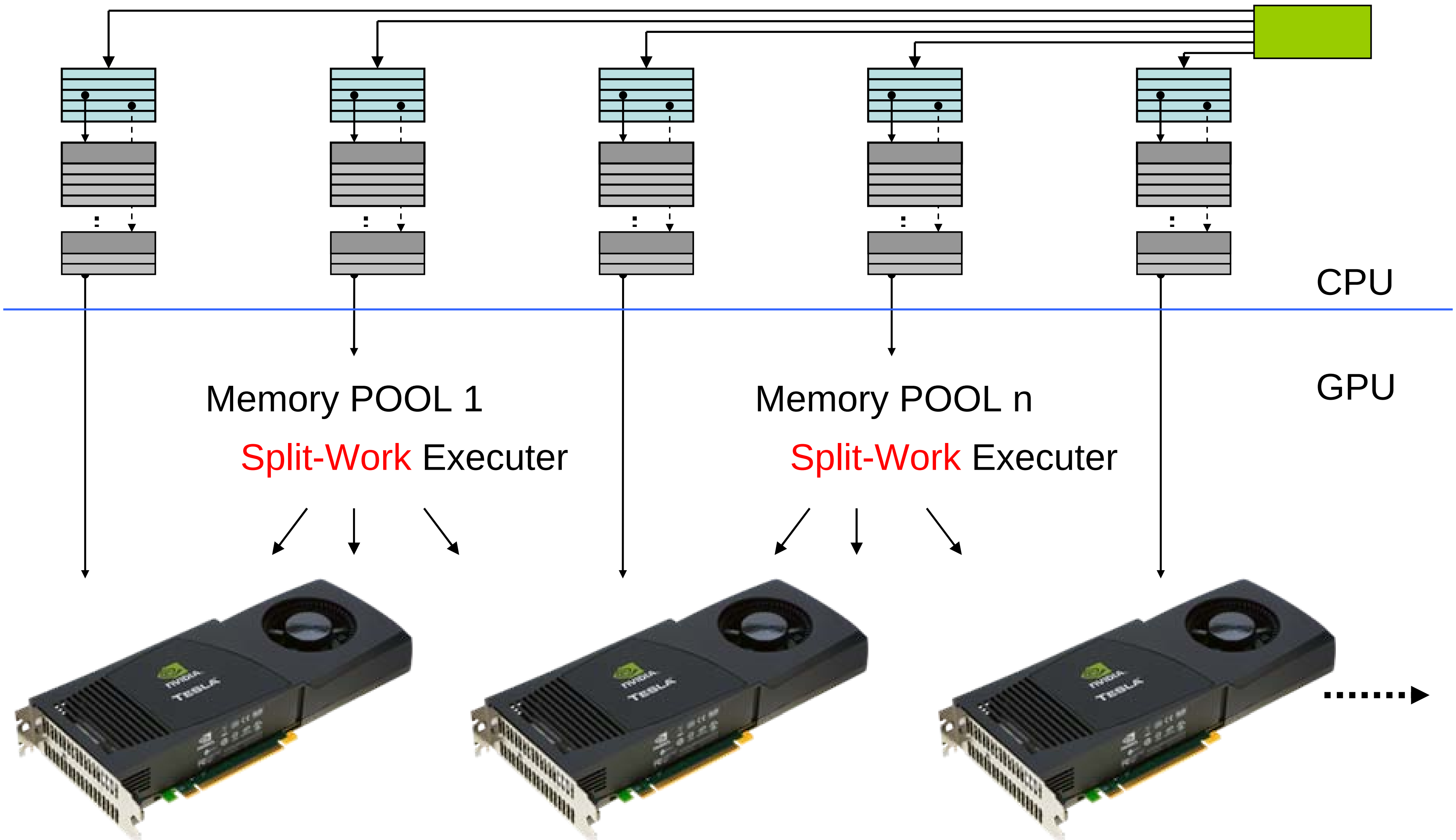


Table management on CPU and GPU

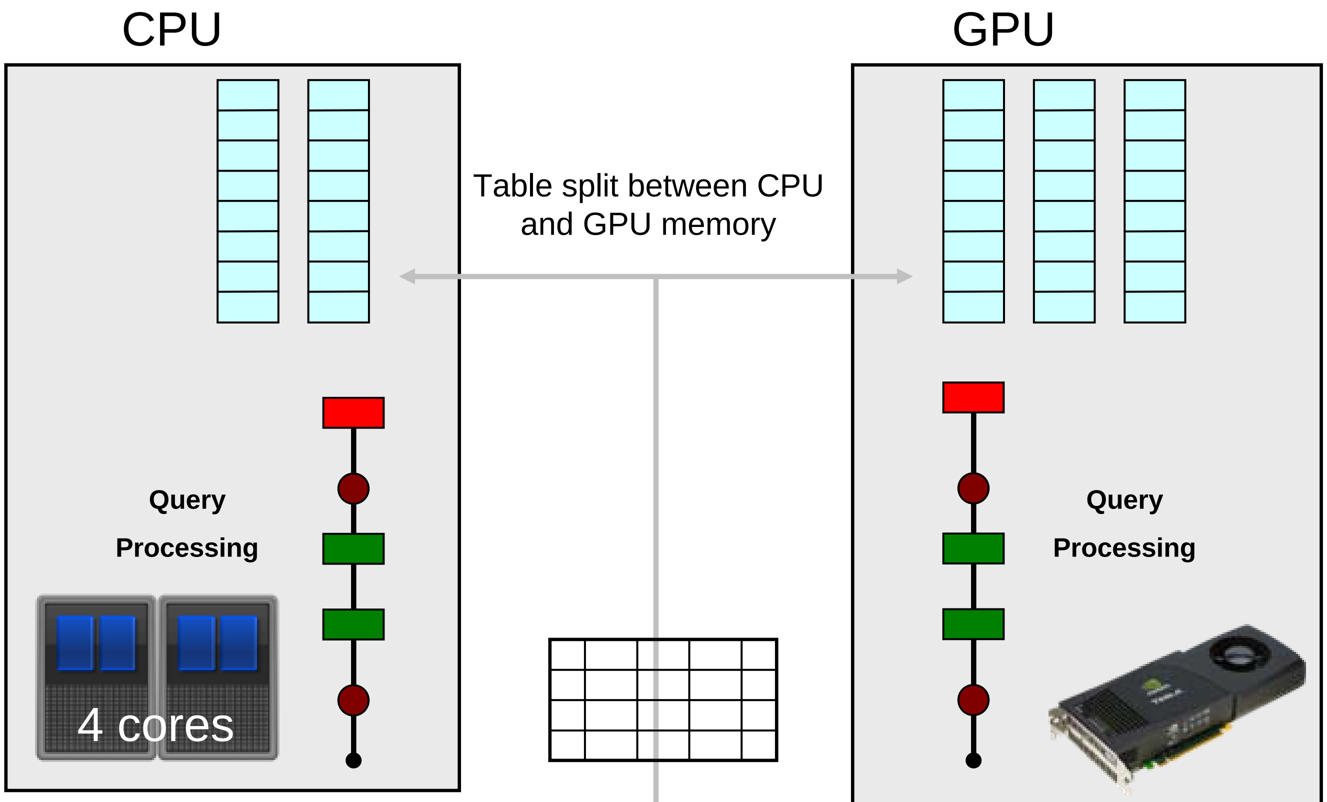


Scales to multi GPU clusters

Some problems do not fit within a single GPU memory



GPU / CPU Table split



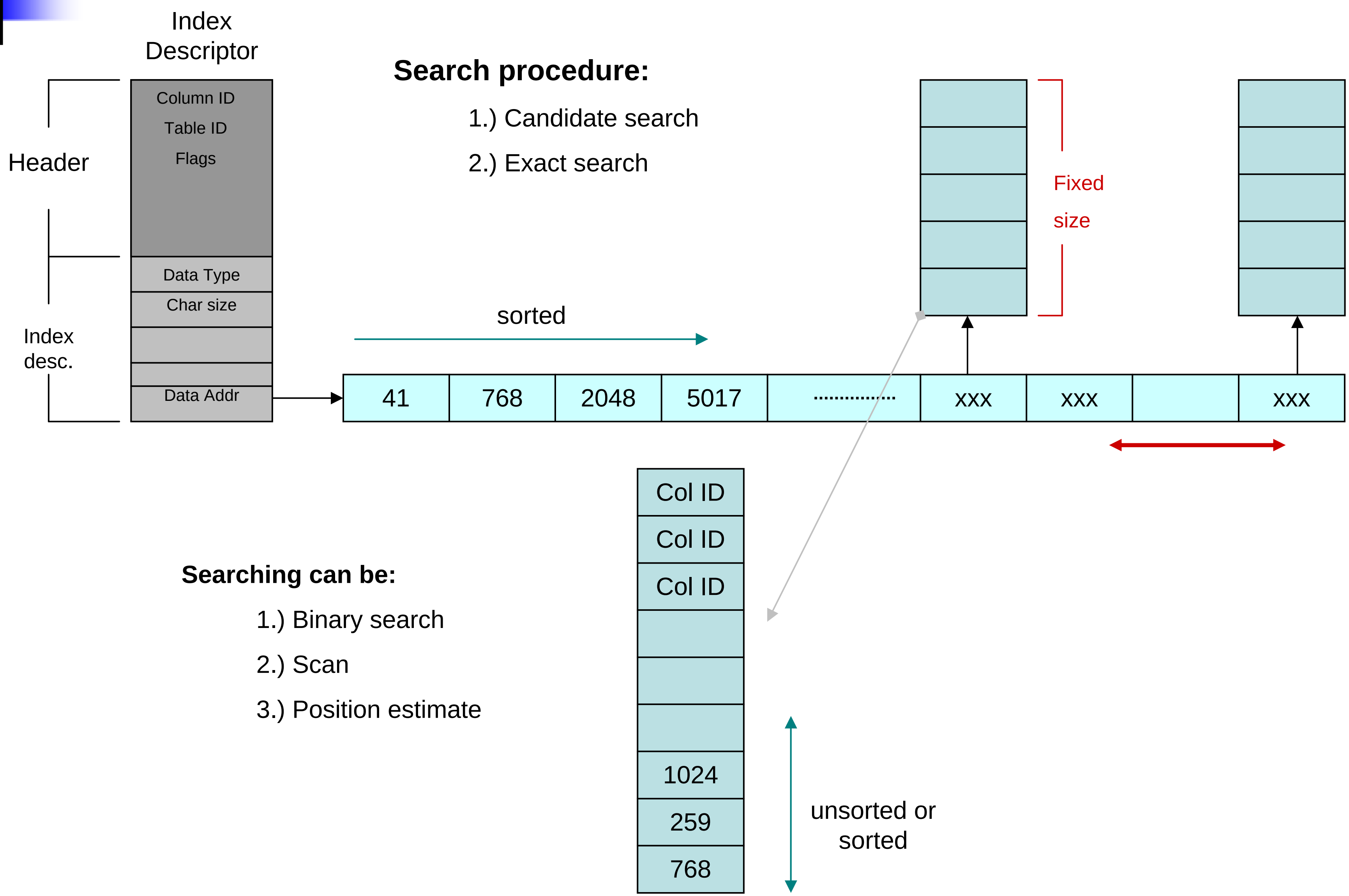
Create Table TR(…)

Create Table Shared TS(…)

Create Table GPU TG(…)

Table split is done automatically depending on column data type: varchar, char, blob, …

Index Layout



Search procedure:

1.) Candidate search

2.) Exact search

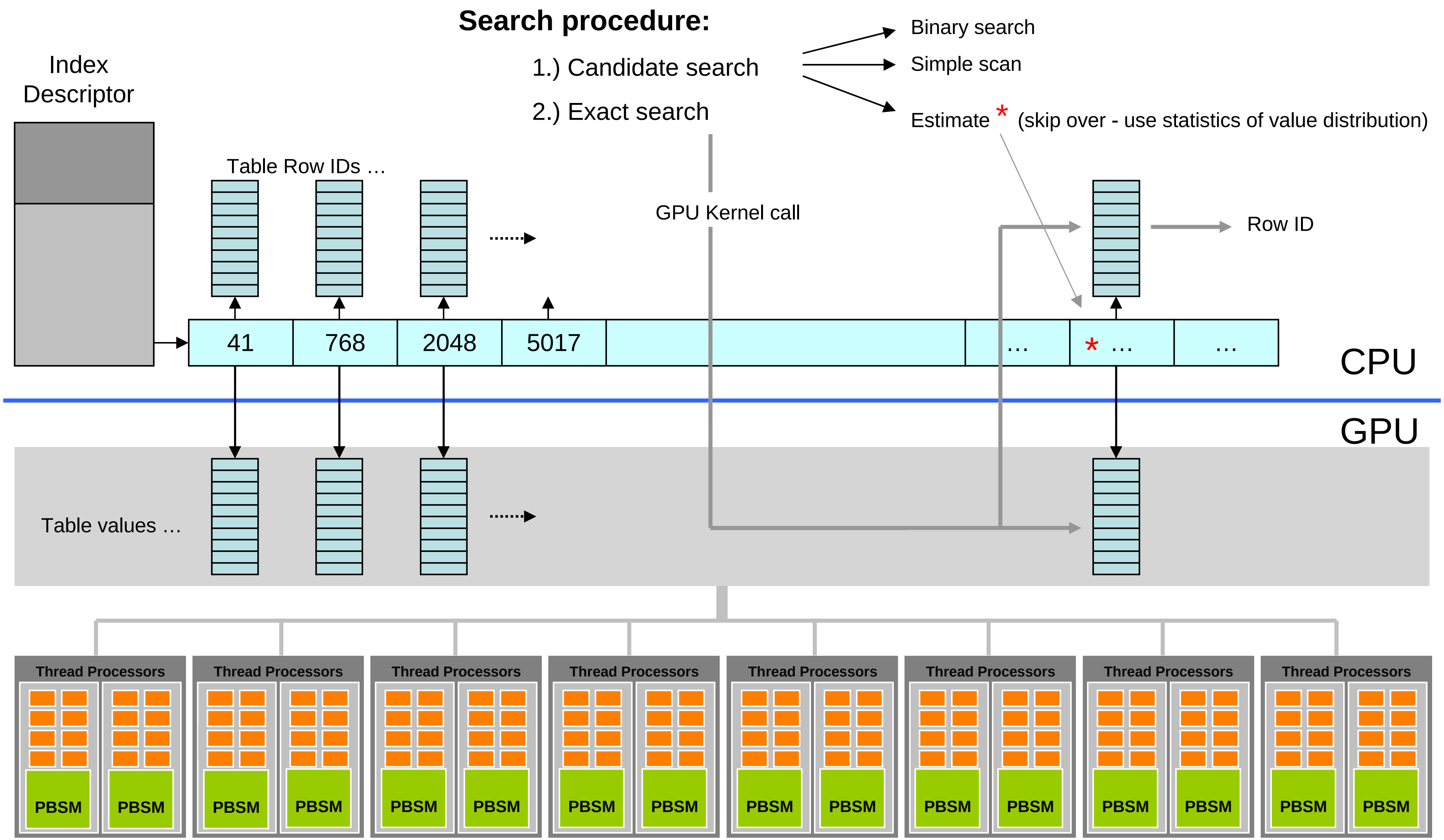
Searching can be:

1.) Binary search

2.) Scan

3.) Position estimate

Index processing on GPU

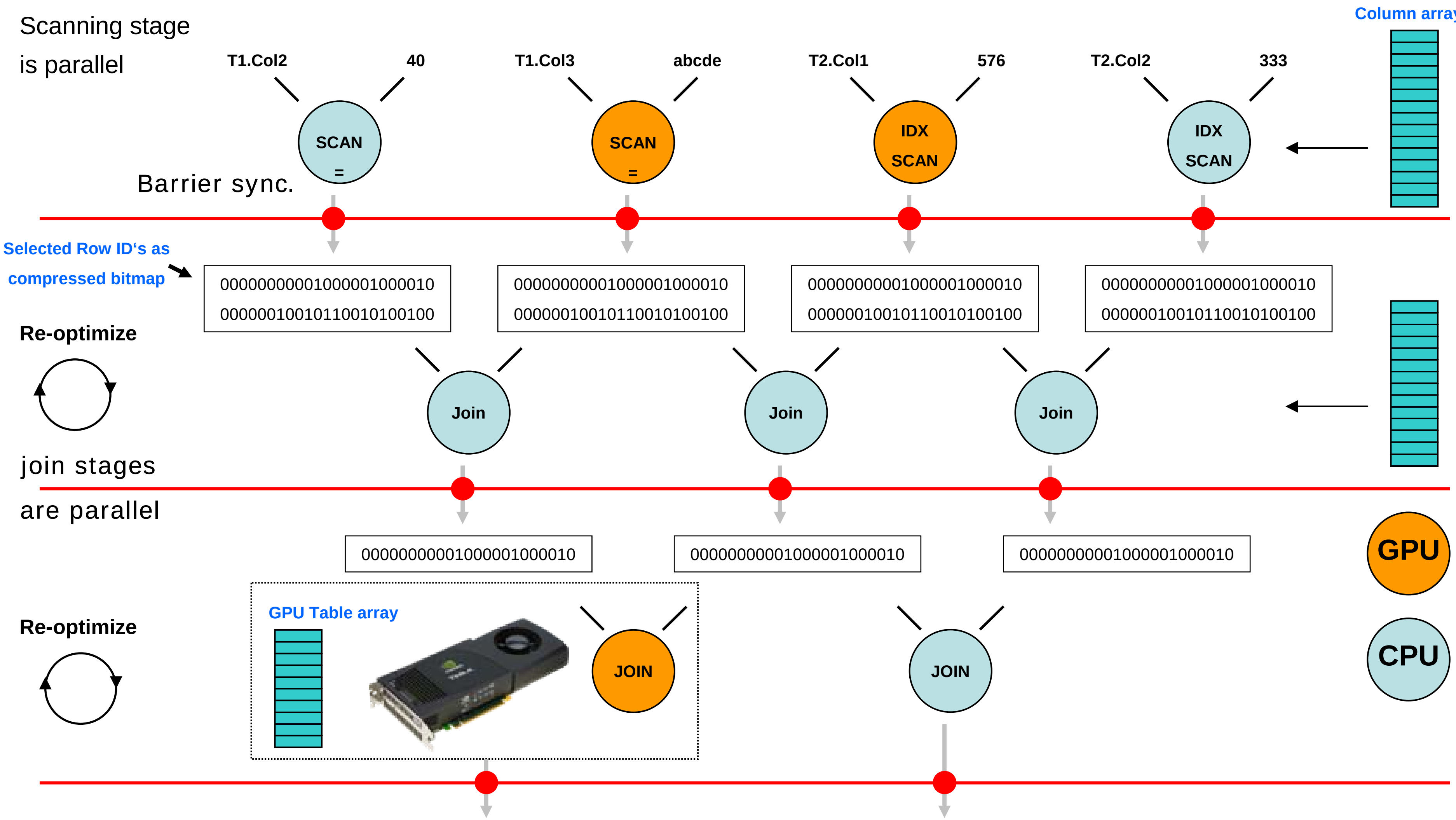


Agenda (2)

- General architecture outline of database kernel
  Explains the overall design principle chosen to make a database kernel ready for massive parallel processing and GPU ready
- Database Table structure and outline
  Explains the vertical partitioning layout and why this is one of the most important pre-requisites for successful GPU co-processing in database kernels
- Memory management
  The memory management subsystem architecture and how to manage memory on a GPU device with missing dynamic memory management facilities
- **Query Optimizer**
  Gives insights into the Split-Work architecture and explains the Optimizer architecture with a special focus on the dynamic re-optimizing steps performed during query processing
- Query Executer structure
  Explains the architecture of the query Executer with a special focus on processing the split-work plan for distributing the query work between CPU and GPU

Query Optimizer and Executer

Re-optimizing steps performed during query execution



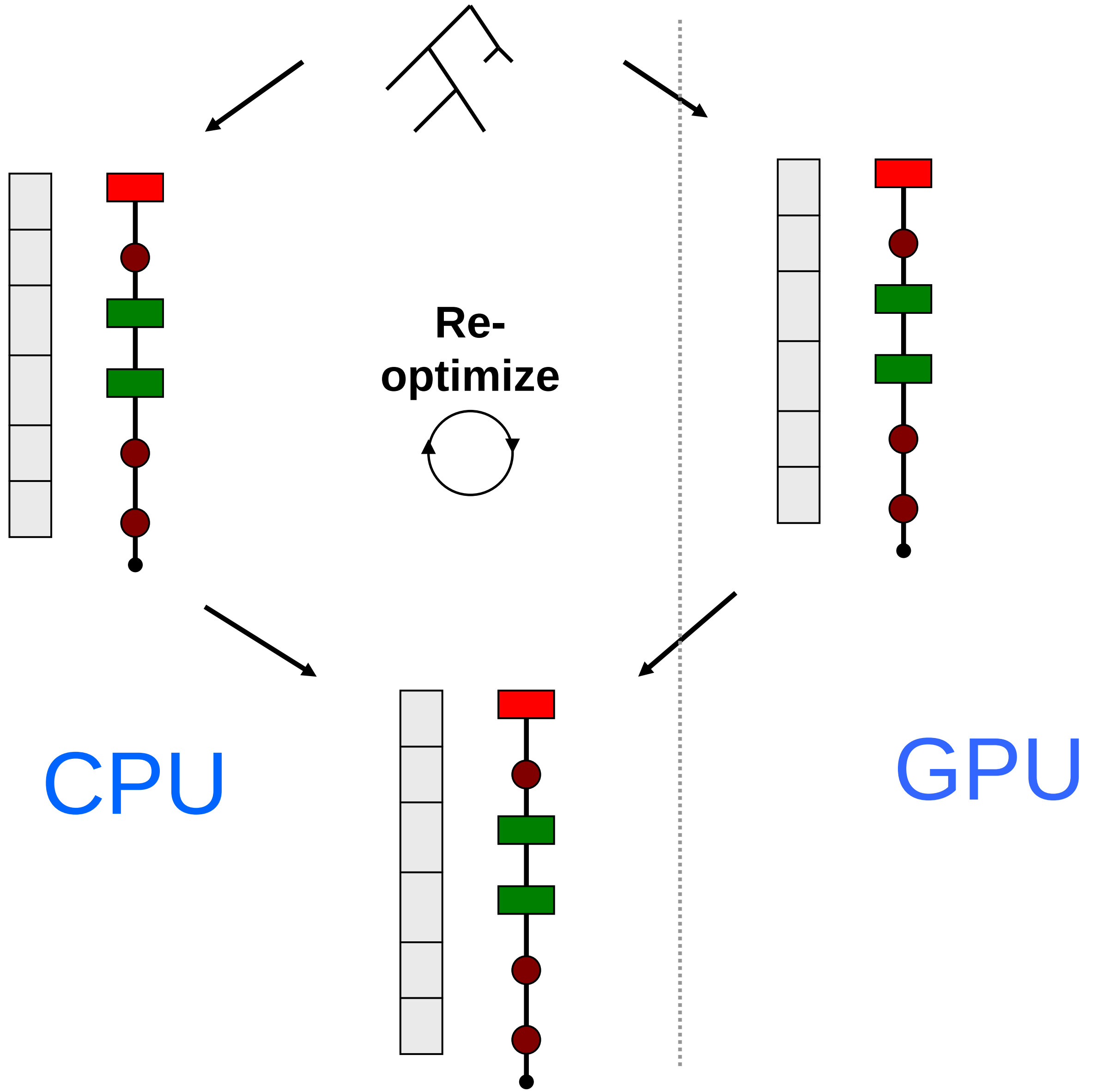
Why is re-optimizing important

- ➢ True size of intermediate result sets !
  - – Split-Work requires re-examination of intermediate results
  - – Schedules the query tasks either to CPU or GPU

- Should sorting be done on GPU or CPU ?
- Leave intermediate results on GPU or move over ?
- Perform join processing on GPU or CPU ?
- Is the table persistent on GPU or on CPU ?
- …

➢ Tight integration of optimizer and executer necessary

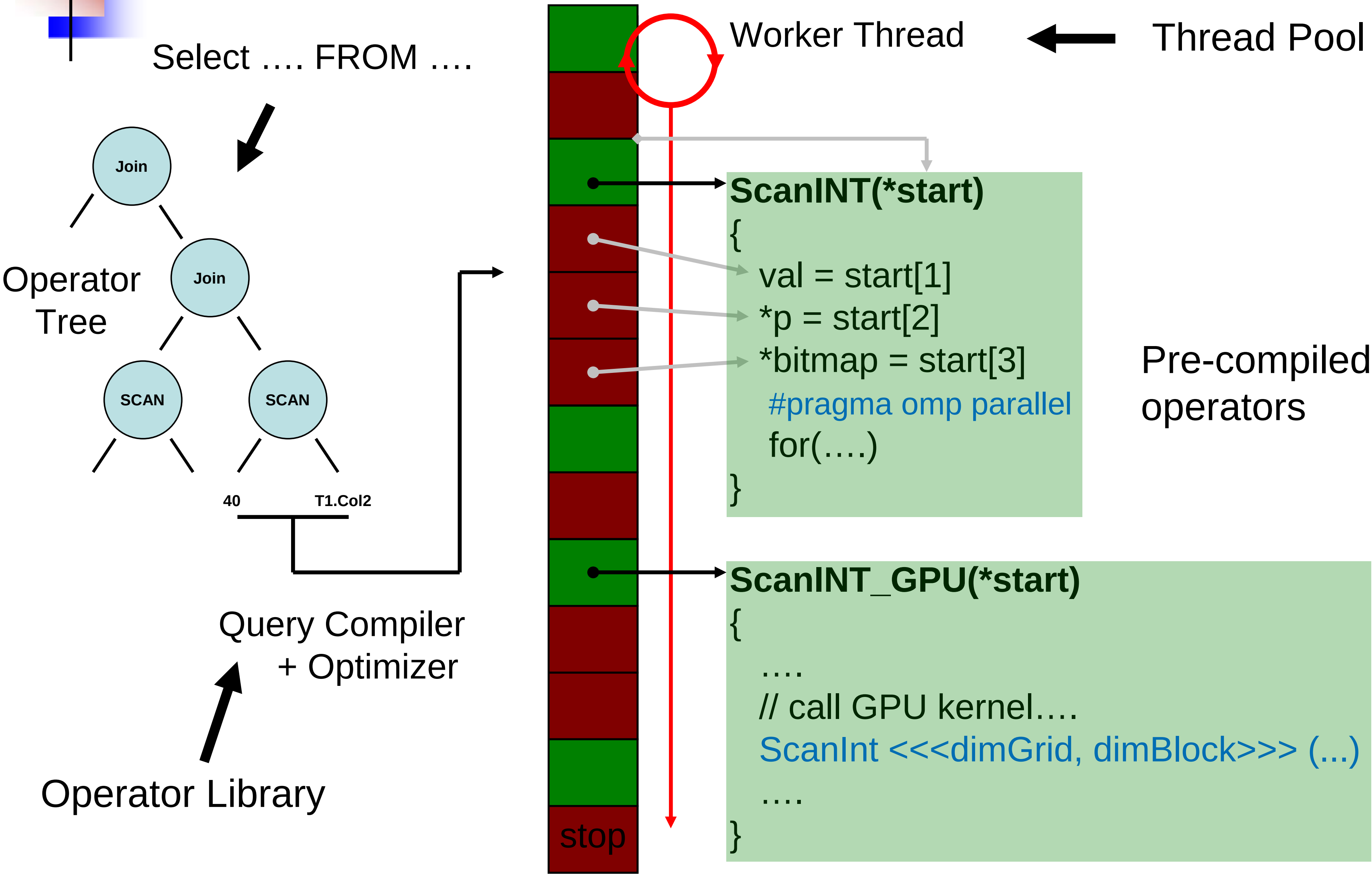
Split-Work



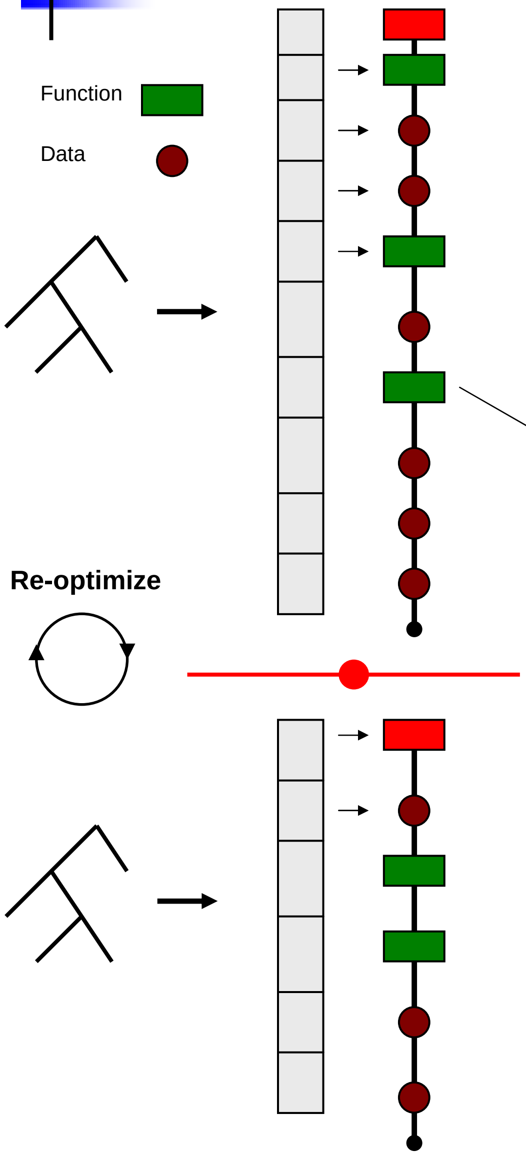
Agenda (2)

- General architecture outline of database kernel
  Explains the overall design principle chosen to make a database kernel ready for massive parallel processing and GPU ready
- Database Table structure and outline
  Explains the vertical partitioning layout and why this is one of the most important pre-requisites for successful GPU co-processing in database kernels
- Memory management
  The memory management subsystem architecture and how to manage memory on a GPU device with missing dynamic memory management facilities
- Query Optimizer
  Gives insights into the Split-Work architecture and explains the Optimizer architecture with a special focus on the dynamic re-optimizing steps performed during query processing
- **Query Executer structure**
  Explains the architecture of the query Executer with a special focus on processing the split-work plan for distributing the query work between CPU and GPU

Query Executer - basic principle



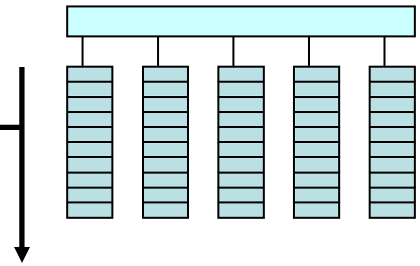
Executer



Operator Library

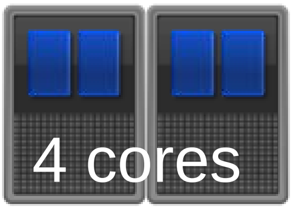
ScanINT(void *start)
ScanUINT(void *start)
ScanFLOAT(void *start)
ScanDBL(void *start)
ScanIntIDX(void *start)
AddINT(void *start)
AddDBL(void *start)
⋮

Function call



Vector processing of algebra functions

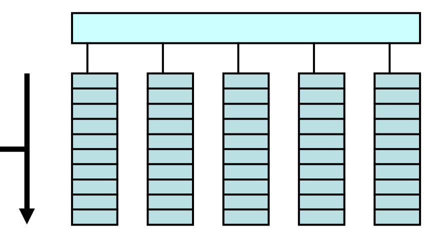
- Compiler optimization
- Filled cache lines
- Memory prefetch support
- No instruction cache misses
- Loop unrolling
- Branch prediction



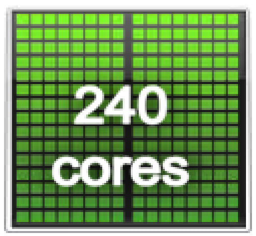
ScanINT_GPU

⋮

GPU Kernel call

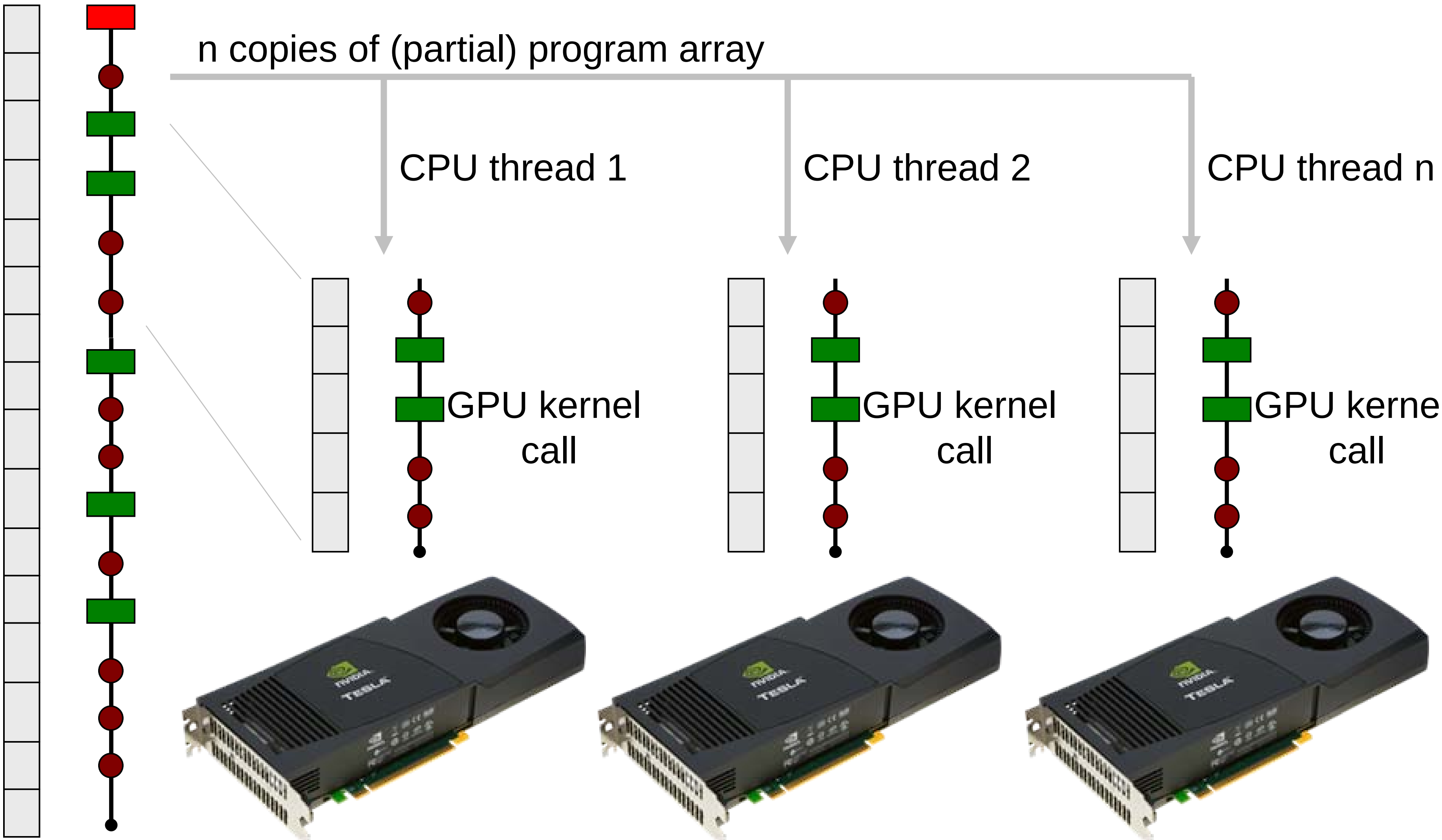


- Coalesced memory access
- High bandwidth
- Massive parallel scanning
- Fast sorting
- Arithmetic processing
- Data local on GPU
- Loop unrolling



Split-Work scheduler scales to multi GPUs

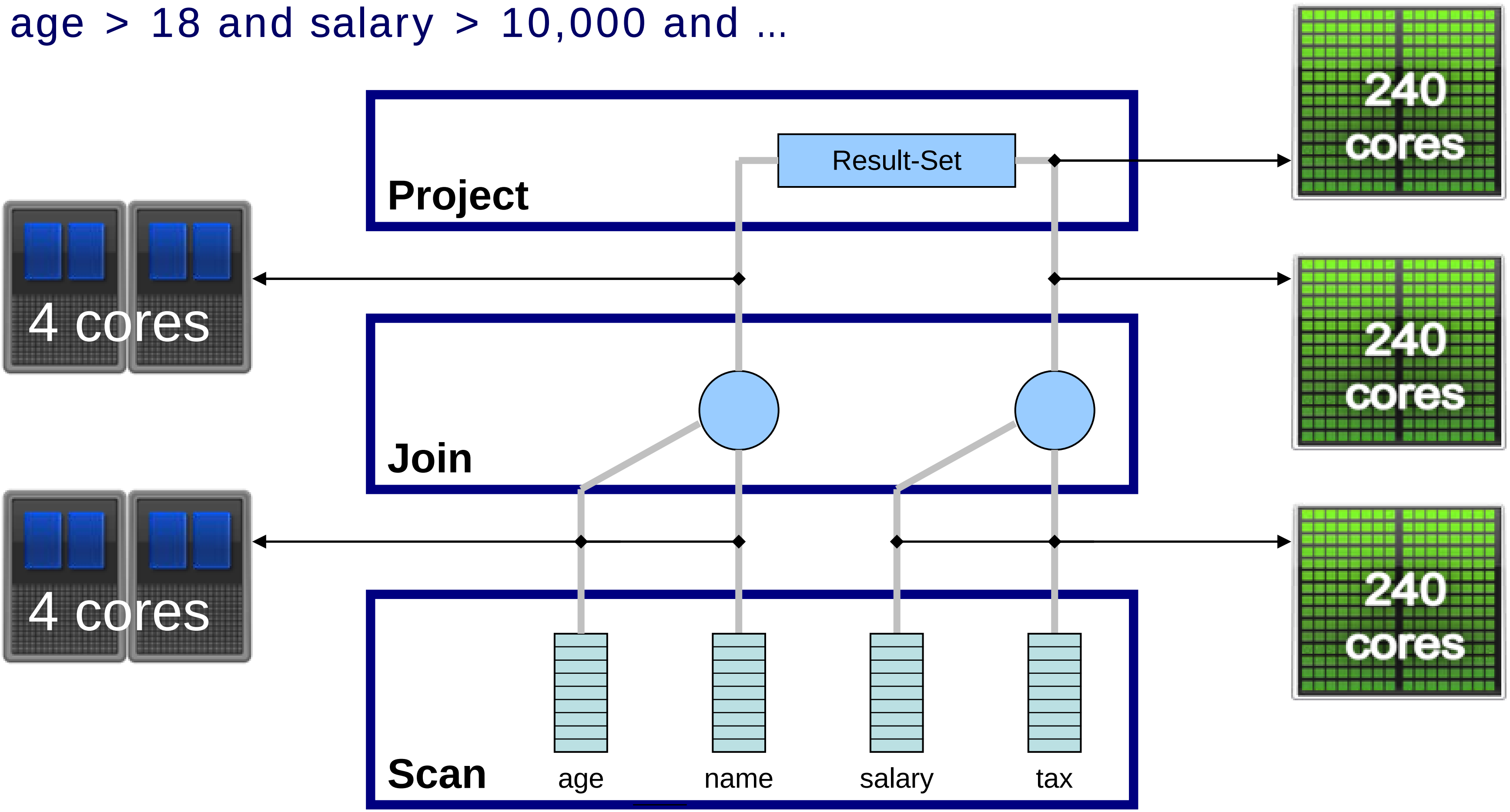
Query work scales linear with GPU count



SQL query processing

Task distribution to maximize performance !

Select age, name, (salary * tax) From T Where age > 18 and salary > 10,000 and …



Lessons learned

➢ Try to keep things simple and regular

- Classic Database technology **to complex** for GPU architecture
- Complete re-engineering required
- Regular structures favor parallelize and scalability
- Index and algorithmic structures needs to be revised
- Use regular shaped data structures for fast iterative access
- Re-optimizing is important for best query work scheduling
- Coalesced memory access required for high performance
- Keep the GPU bussy
- OpenMP is a good starting point
- Object oriented coding style is useless for GPU co-processing

Future plans

- GPU accelerated projection stage of SQL query is „work in progress“
  - SELECT Tg.a * Tg.b + (1.323 * Tg.c) FROM ....
- LINQ integration
  - (Language INtegrated Query for Microsoft .NET-Framework)
- GPU based XML processing
- Self tuning (Experiment mode)
- Embedded devices ?

Conclusions

- The *RISC* like „regular-shaped“ DB technology scales to hundreds of parallel processor cores

- GPU usage as DB co-processor is a valid concept and boosts performance orders of magnitude

- Declarative programming style for application development is easier, faster and …
  - You get the GPU power for free !

- QuiLogic makes GPU power available for domain experts in a simple and declarative way (SQL)

Questions ?

harald.frick@chello.at

office@quilogic.com