Big Data Analytics on Modern Hardware Architectures

Volker Markl
Michael Saecker

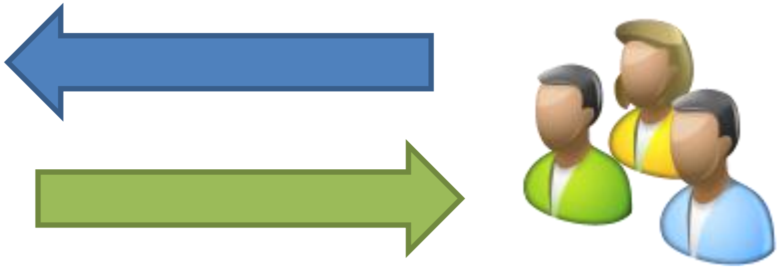
With material from:
S. Ewen, M. Heimel, F. Hüske, C. Kim, N. Leischner, K. Sattler

Database Systems and Information Management Group
Technische Universität Berlin

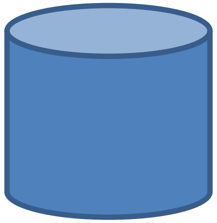
http://www.dima.tu-berlin.de/

Motivation

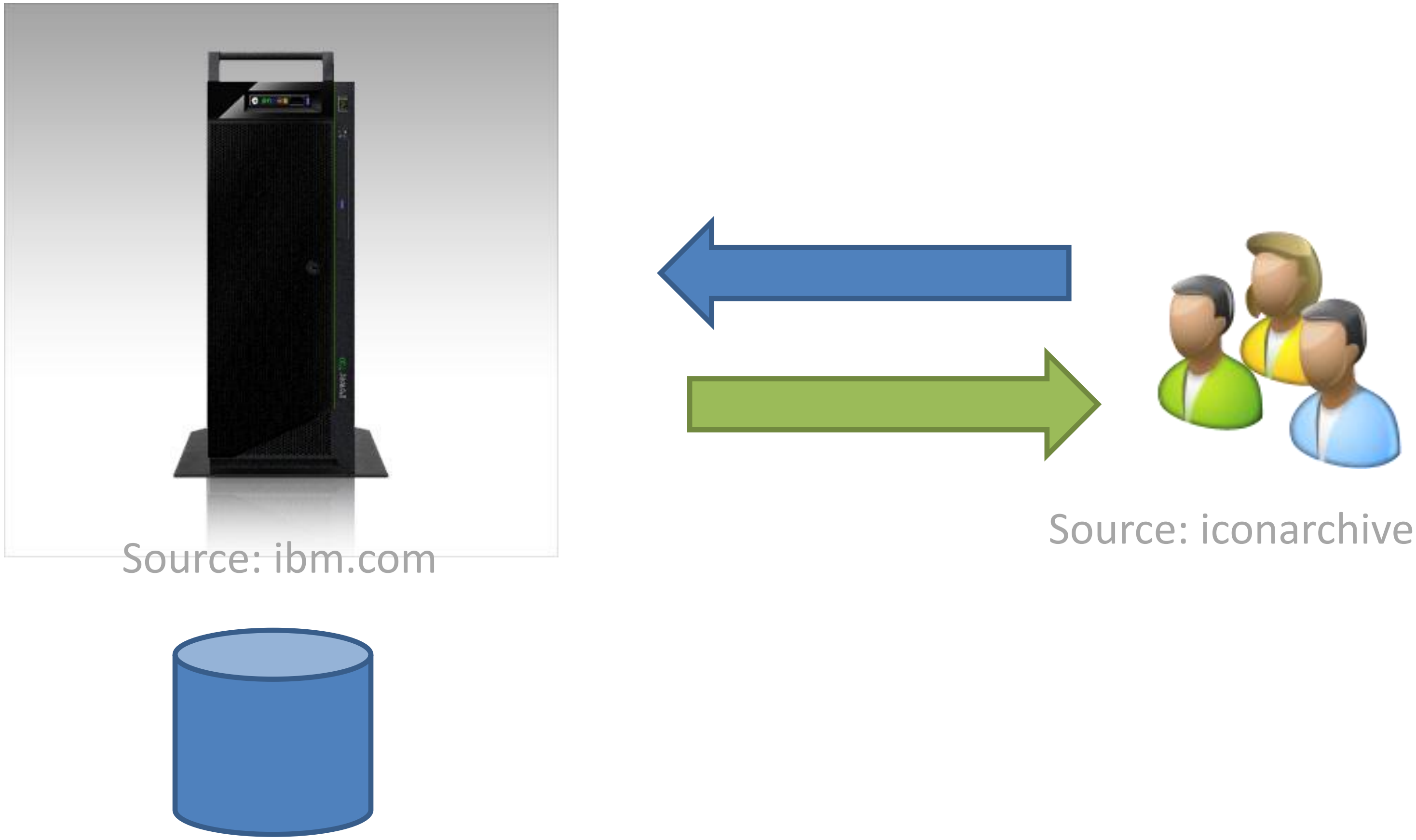
Source: old-computers.com



Source: iconarchive.com



Motivation

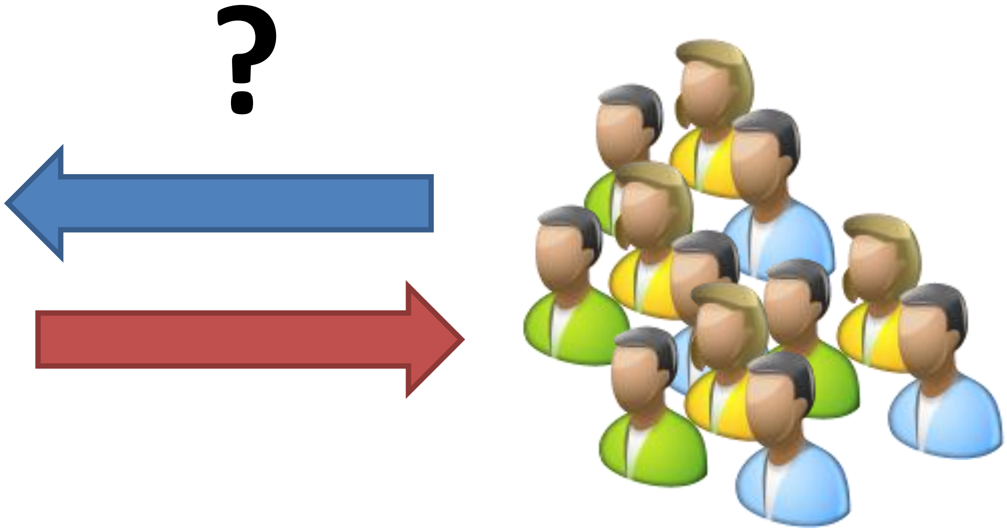


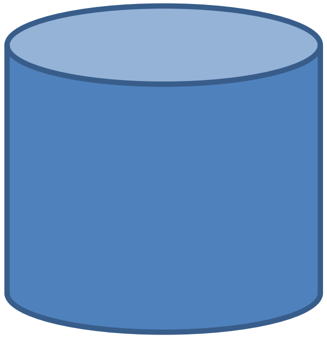
Source: ibm.com

Source: iconarchive.com

Motivation

Source: ibm.com





- Amount of data increases at a high speed
- Response time grows
- Number of requests / users increase

Motivation – Data Sources

- Scientific applications
  - □ Large Hadron Collider (15 PB / year)
  - □ DNA sequencing

Source: cern.ch

Source: readwriteweb.com

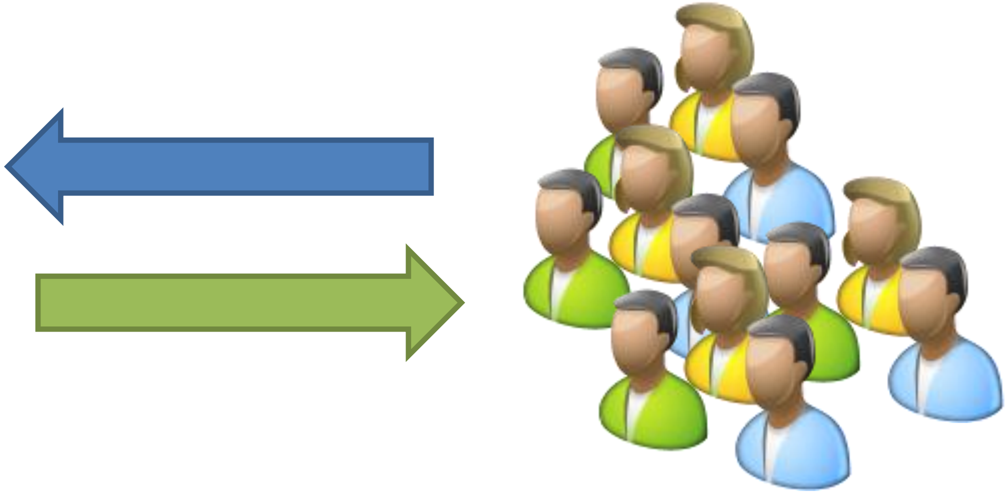
- Sensor networks
  - □ Smart homes
  - □ Smart grids

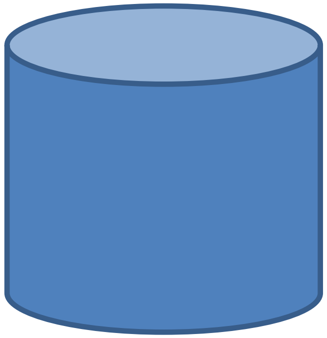
- Multimedia applications
  - □ Audio & video analysis
  - □ User generated content

Source: uio.no

Motivation – Scale up

Source: ibm.com



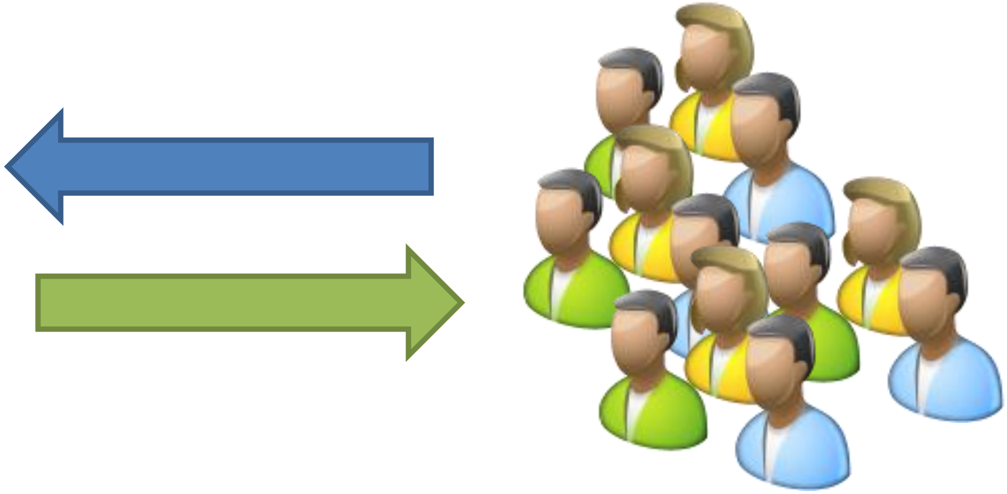


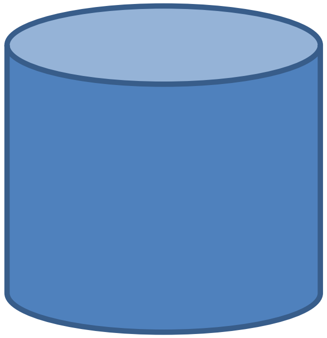
Solution 1

- Powerful server

Motivation – Scale out

Source: 330t.com





Solution 2

- Many (commodity-) server

Outline

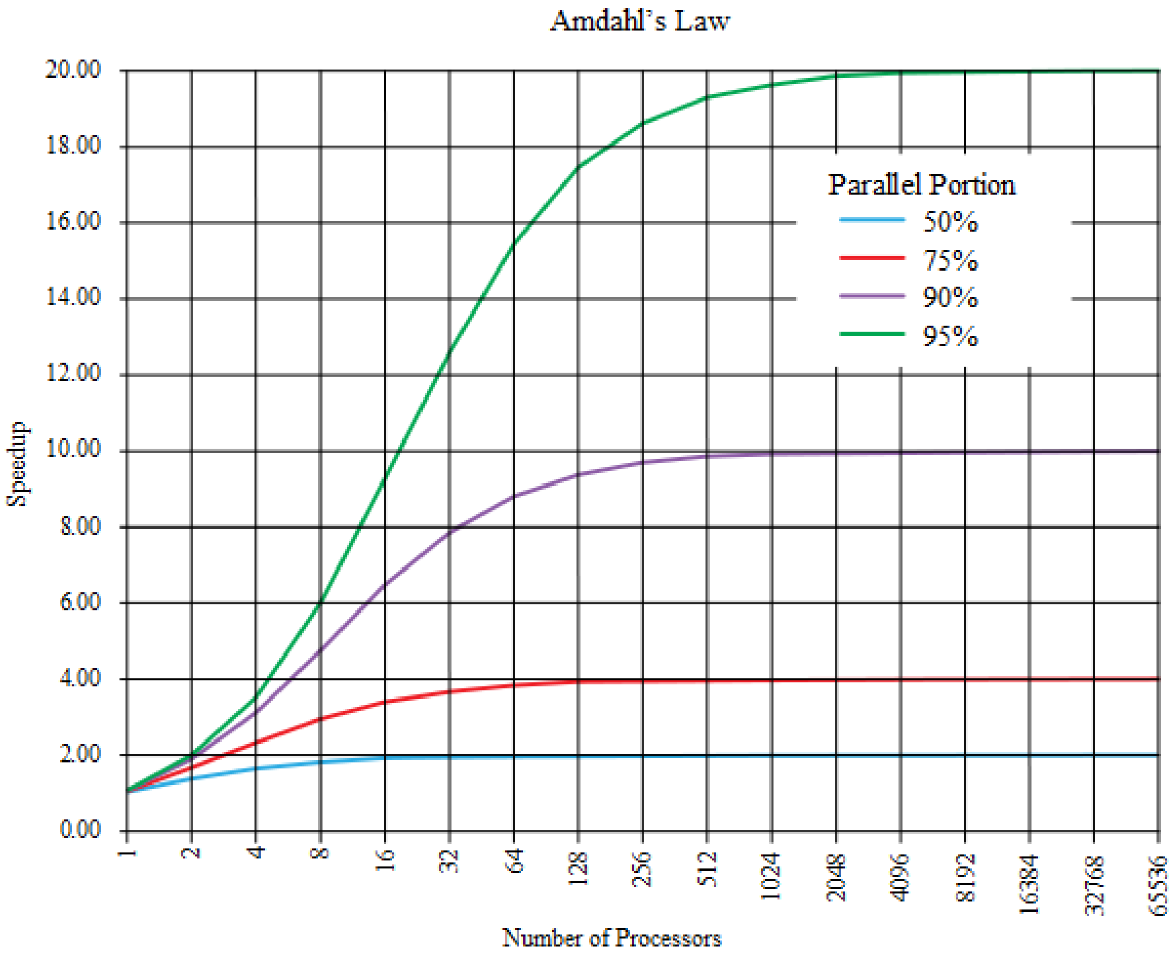
- **Background**
  - Parallel Speedup
  - Levels of Parallelism
  - CPU Architecture
- Scale out
  - MapReduce
  - Stratosphere
- Scale up
  - Overview of Hardware Architectures
  - Parallel Programming Model
  - Relational Processing
  - Further Operations
  - Research Challenges of Hybrid Architectures

Parallel Speedup

- The speedup is defined as: $S_p = {}^{T_1}/_{T_p}$
  - □ T_1: runtime of sequential program
  - □ T_p: runtime of the parallel program on p processors

- Amdahl‘s Law: „The maximal speedup is determined by the non-parallelizable part of a program.“
  - □ $S_{max} = \frac{1}{(1-f) + f/p}$ f: fraction of the program that can be parallelized
  - □ Ideal speedup: S=p for f=1.0 (linear speedup)
  - □ However – since usually f < 1.0, S is bound by a constant
  - □ Fixed problems can be parallelized only to a certain degree

Parallel Speedup



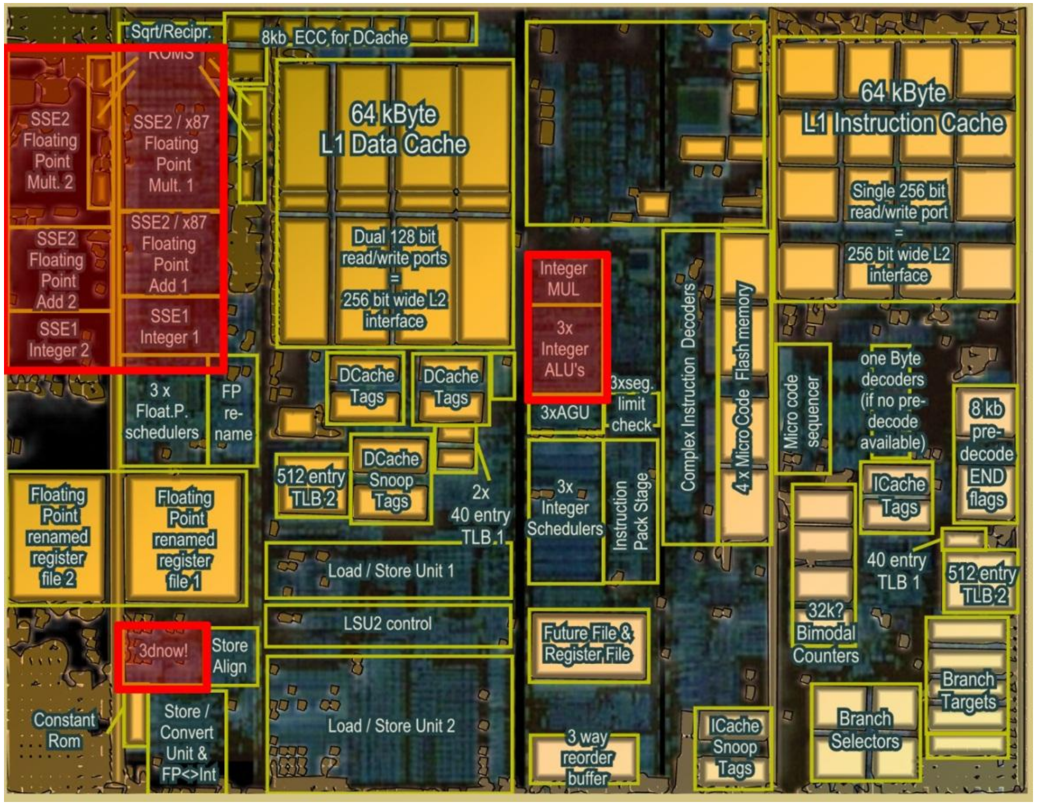
Levels of Parallelism on Hardware

- Instruction-level Parallelism
  - □ Single instructions are automatically processed in parallel
  - □ **Example:** Modern CPUs with multiple pipelines and instruction units.

- Data Parallelism
  - □ Different Data can be processed independently
  - □ Each processor executes the same operations on it‘s share of the input data.
  - □ **Example:** Distributing loop iterations over multiple processors, or CPU’s vectors

- Task Parallelism
  - □ Tasks are distributed among the processors/nodes
  - □ Each processor executes a different thread/process.
  - □ **Example:** Threaded programs.

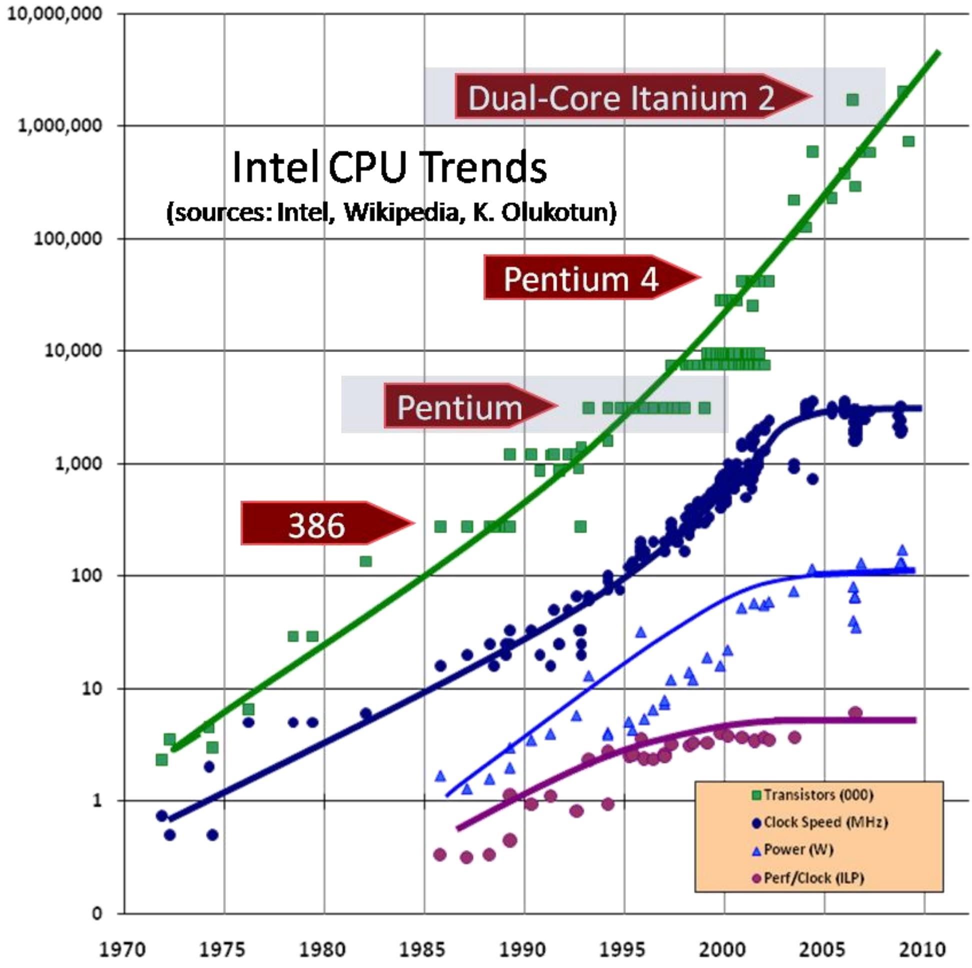
CPU Architecture


Sqrt/Recipr. ROMS
8kb ECC for DCache
SSE2 Floating Point Mult. 2
SSE2 / x87 Floating Point Mult. 1
SSE2 Floating Point Add 2
SSE2 / x87 Floating Point Add 1
SSE1 Integer 2
SSE1 Integer 1
64 kByte L1 Data Cache
Dual 128 bit read/write ports = 256 bit wide L2 interface
64 kByte L1 Instruction Cache
Single 256 bit read/write port = 256 bit wide L2 interface
Integer MUL
3x Integer ALU's
3xAGU
3xseg. limit check
Complex Instruction Decoders
4 x Micro Code Flash memory
Micro code sequencer
one Byte decoders (if no pre-decode available)
8 kb pre-decode END flags
3 x Float.P. schedulers
FP re-name
DCache Tags
DCache Tags
DCache Snoop Tags
512 entry TLB 2
2x 40 entry TLB 1
3x Integer Schedulers
Instruction Pack Stage
ICache Tags
40 entry TLB 1
512 entry TLB 2
Floating Point renamed register file 2
Floating Point renamed register file 1
Load / Store Unit 1
LSU2 control
Future File & Register File
32k? Bimodal Counters
Branch Targets
3dnow!
Store Align
Constant Rom
Store / Convert Unit & FP<>Int
Load / Store Unit 2
3 way reorder buffer
ICache Snoop Tags
Branch Selectors

AMD K8L www.chip-architect.com

- Most die space devoted to control logic & caches
- Maximize performance for *arbitrary*, *sequential* programs

Trends in processor architecture



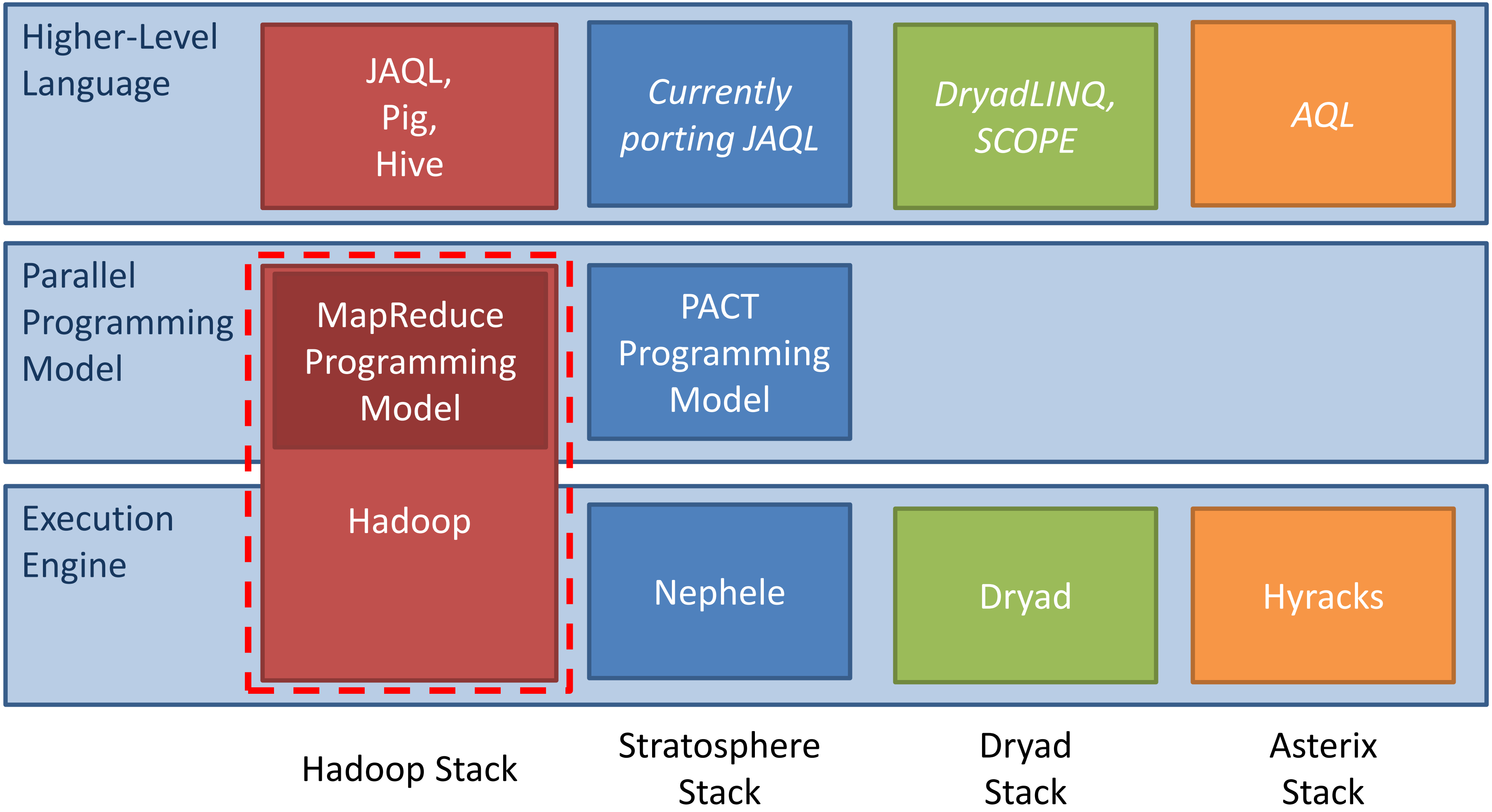
Free lunch is over:

- Power wall
  - Heat dissipation
- Memory wall
  - Memory gained little performance
- ILP wall
  - Extracting more ILP scales poorly

Outline

- Background
  - Parallel Speedup
  - Levels of Parallelism
  - CPU Architecture
- Scale out
  - **MapReduce**
  - Stratosphere
- Scale up
  - Overview of Hardware Architectures
  - Parallel Programming Model
  - Relational Processing
  - Further Operations
  - Research Challenges of Hybrid Architectures

Comparing Architectural Stacks



	Hadoop Stack	Stratosphere Stack	Dryad Stack	Asterix Stack
Higher-Level Language	JAQL, Pig, Hive	*Currently porting JAQL*	*DryadLINQ, SCOPE*	*AQL*
Parallel Programming Model	MapReduce Programming Model	PACT Programming Model		
Execution Engine	Hadoop	Nephele	Dryad	Hyracks

Where traditional Databases are unsuitable

- Analysis over raw (unstructured) data
  - □ Text processing
  - □ In general: If relational schema does not suit the problem well
    - – XML, RDF

- Where cost-effective scalability is required
  - □ Use commodity hardware
  - □ Adaptive cluster size (horizontal scaling)
  - □ Incrementally growing, add computers without requirement for expensive reorganization that halts the system

- In unreliable infrastructures
  - □ Must be able to deal with failures – hardware, software, network
    - – Failure is expected rather than exceptional
  - □ Transparent to applications
    - – very expensive to build reliability into each application

Example Use Case: Web Index Creation

- A Search Engine scenario:
  - Have crawled the internet and stored the relevant documents
  - Documents contain words (Doc-URL, [list of words])
  - Documents contain links (Doc-URL, [Target-URLs])

- Need to build a search index
  - Invert the files (word, [list of URLs])
  - Compute a ranking (e.g. page rank), which requires an inverted graph: (Doc-URL, [URLs-pointing-to-it])

- Obvious reasons against relational databases here
  - Relational schema and algebra do not suit the problem well
  - Importing the documents, converting them to the storage format is expensive

- A mismatch between what Databases were designed for and what is really needed:
  - Databases come originally from transactional processing. They give hard guarantees about absolute consistencies in the case of concurrent updates.
  - Analytics are added on top of that
  - Here: The documents are never updated, they are read only. It is only about analytics here!

An Ongoing Re-Design…

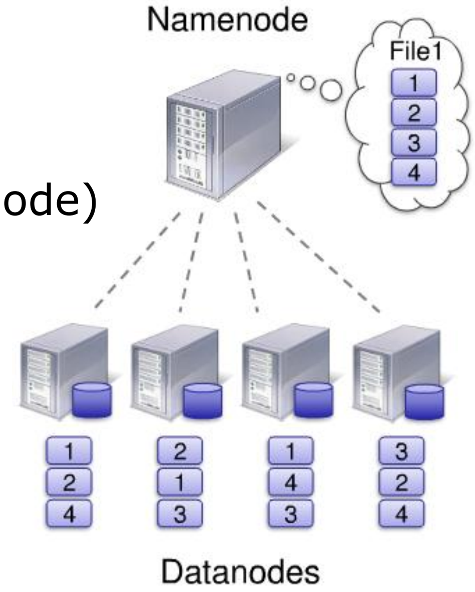
- Driven by companies like Google, Facebook, Yahoo
- Use heavily distributed system
  - □ Google used 450,000 low-cost commodity servers in 2006 in cluster of 1000 – 5000 nodes
- Redesign infrastructure and architectures completely with the key goal to be
  - □ Highly scalable
  - □ Tolerant of failures
- Stay generic and schema free in the data model
- Start with: Data Storage
- Next Step: Distributed Analysis

Storage Requirements

- Extremely large files
  - In the order of Terabytes to Petabytes
- High Availability
  - Data must be kept replicated
- High Throughput
  - Read/Write Operations must not go through other servers
  - A write operation must not be halted until the write is completed on the replicas.
    → Even if it may require to make files unmodifyable
- No single point of failure
  - A Master must be kept redundantly
- Many different distributed file systems exist. They have very different goals, like transparency, updateability, archiving, etc…
- A widely used reference architecture for high-throughput and high-availability DFS is the Google Filesystem (GFS)

The Storage Model – Distributed File System

- The file system
  - □ is distributed across many nodes (DataNodes)
  - □ provides a single namespace for the entire cluster
  - □ metadata is managed on a dedicated node (NameNode)
  - □ realizes a write-once-read-many access model



- Files are split into blocks
  - □ typically 128 MB block size
  - □ each block is replicated on multiple data nodes

- The client
  - □ can determine the location of blocks
  - □ can access data directly from the DataNode over the network

- Important: No file modifications (except appends),
  - □ Spares the problem of locking and inconsistent or conflicting updates

Retrieving and Analyzing Data

- Data is stored as custom records in files
  - Most generic data model that is possible
- Records are read and written with data model specific (de)serializers
- Analysis or transformation tasks must be written directly as a program
  - Not possible to generate it from a higher level statement
  - Like a query-plan is automatically generated from SQL
- Programs must be parallel, highly scalable, fault tolerant
  - Extremely hard to program
  - Need a programming model and framework that takes care of that
  - The **MapReduce** model has been suggested and successfully adapted on a broad scale

What is MapReduce?

- Programming model
  - borrows concepts from functional programming
  - suited for parallel execution – automatic parallelization & distribution of data and computational logic
  - clean abstraction for programmers

- Functional programming influences
  - treats computation as the evaluation of mathematical functions and avoids state and mutable data
  - no changes of states (no side effects)
  - output value of a function depends only on its arguments

- `Map` and `Reduce` are higher-order functions
  - take user-defined functions as argument
  - return a function as result
  - to define a MapReduce job, the user implements the two functions

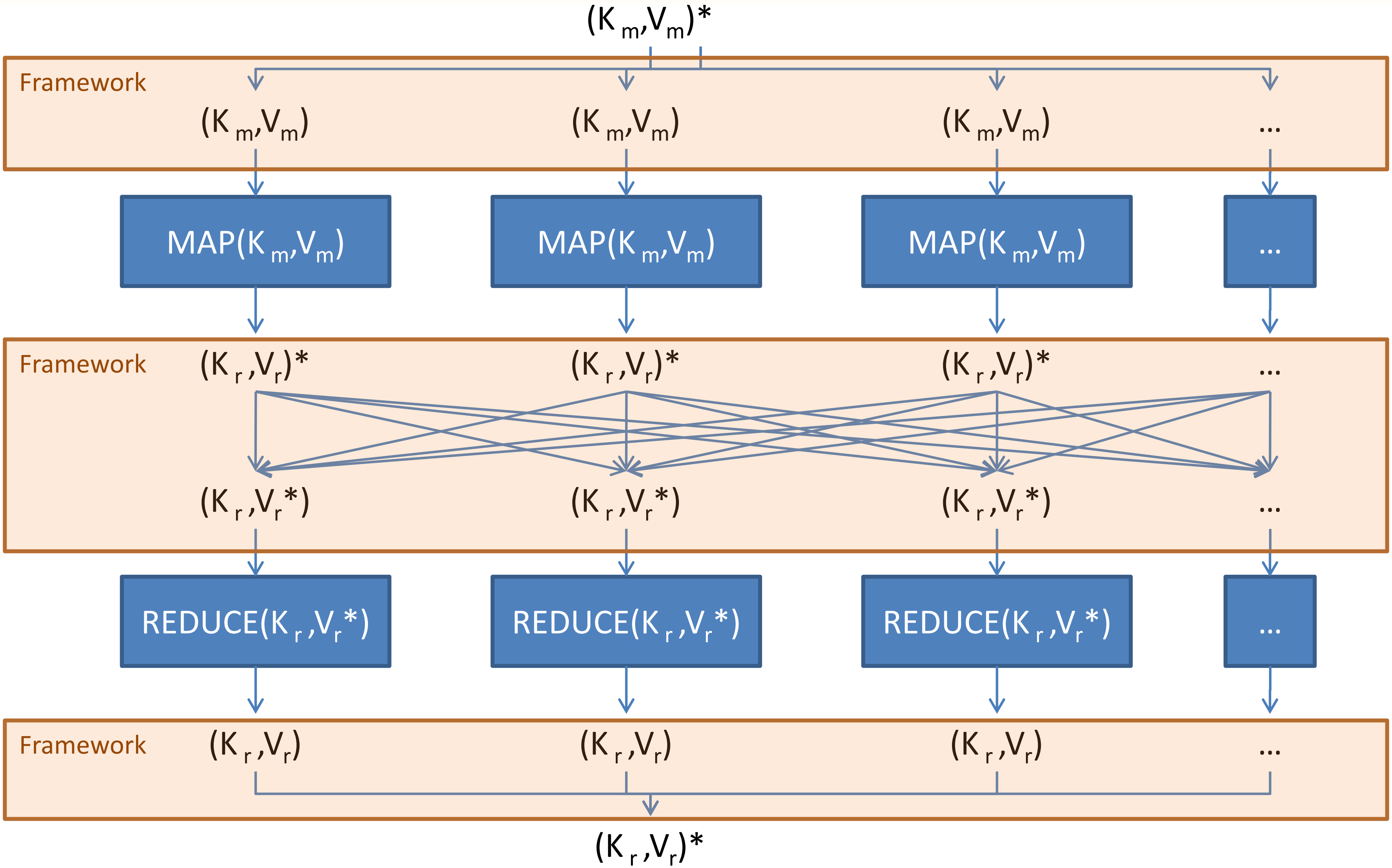
User Defined Functions

- The data model
  - key/value pairs $(K \times V)$
  - e.g. (int, string)

- The user defines two functions
  - map: $\mathcal{M} : (K_m \times V_m) \mapsto (K_r \times V_r)^*$
    - input key-value pairs: $(k, v)\ k \in K_m,\ v \in V_m$
    - output key-value pairs: $(g, w)\ g \in K_r,\ w \in V_r$
  - reduce: $\mathcal{R} : (K_r, V_r^*) \mapsto (K_r, V_r)$
    - input key $\in K_r$ and a list of values $\in V_r^*$
    - output key $\in K_r$ and a single value $\in V_r$

- The framework
  - accepts a list $(K_m \times V_m)^*$
  - outputs result pairs $(K_r, V_r)^*$

Data Flow in MapReduce



MapReduce Illustrated (1)

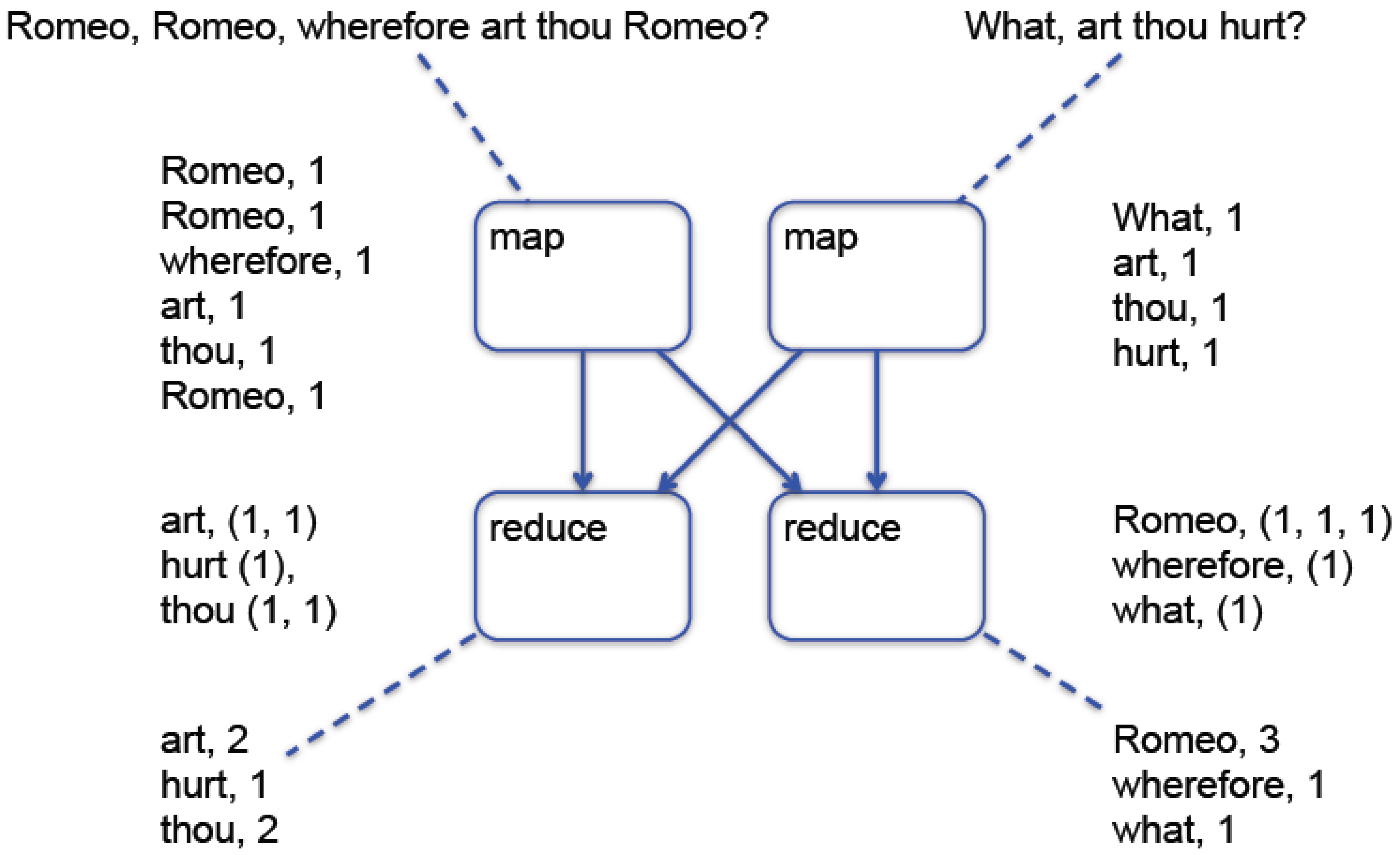
- *Problem*: Counting words in a parallel fashion
  - □ How many times different words appear in a set of files
  - □ **juliet.txt**: Romeo, Romeo, wherefore art thou Romeo?
  - □ **benvolio.txt:** What, art thou hurt?
  - □ Expected output: Romeo (3), art (2), thou (2), art (2), hurt (1), wherefore (1), what (1)

- *Solution*: MapReduce Job

```
map(filename, line) {
  foreach (word in line)
     emit(word, 1);
 }

reduce(word, numbers) {
  int sum = 0;
  foreach (value in numbers) {
    sum += value;
  }
  emit(word, sum);
}
```

MapReduce Illustrated (2)



Romeo, Romeo, wherefore art thou Romeo?

What, art thou hurt?

Romeo, 1
Romeo, 1
wherefore, 1
art, 1
thou, 1
Romeo, 1

What, 1
art, 1
thou, 1
hurt, 1

art, (1, 1)
hurt (1),
thou (1, 1)

Romeo, (1, 1, 1)
wherefore, (1)
what, (1)

art, 2
hurt, 1
thou, 2

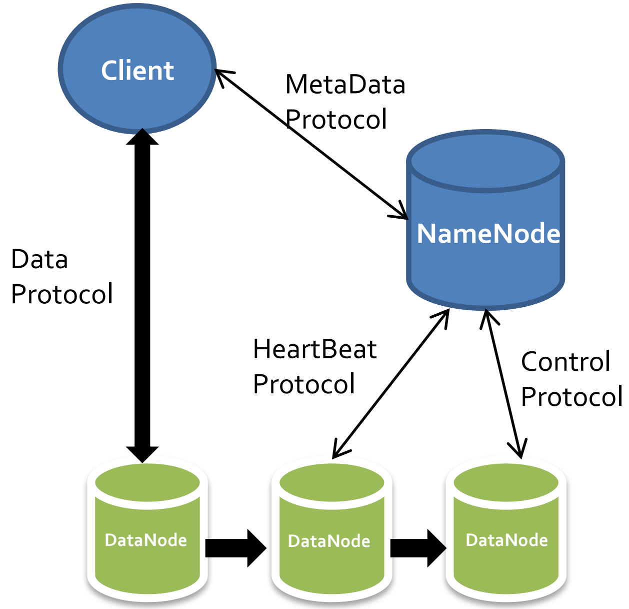
Romeo, 3
wherefore, 1
what, 1

Hadoop – A MapReduce Framework

- Hadoop: Apache Top Level Project
  - □ open Source
  - □ written in Java

- Hadoop provides a stack of
  - □ distributed file system (HDFS) – modeled after the Google File System
  - □ MapReduce engine
  - □ data processing languages (Pig Latin, Hive SQL)
- Runs on
  - □ Linux, Mac OS/X, Windows, Solaris
  - □ Commodity hardware

Hadoop Distributed File System (HDFS)



- Master-Slave Architecture

- HDFS Master "NameNode"
  - manages all filesystem metadata
  - controls read/write access to files
  - manages block replication

- HDFS Slave "DataNode"
  - communicates with the NameNode periodically via heartbeats
  - serves read/write requests from clients
  - performs replication tasks upon instruction by NameNode

Hadoop MapReduce Engine

- Master / Slave architecture
- MapReduce Master: JobTracker
  - accepts jobs submitted by clients
  - assigns map and reduce tasks to TaskTrackers
  - monitors execution status, re-executes tasks upon failure
- MapReduce Slave: TaskTracker
  - runs map / reduce tasks upon instruction from the task tracker
  - manage storage, sorting and transmission of intermediate output

Hadoop MapReduce Engine

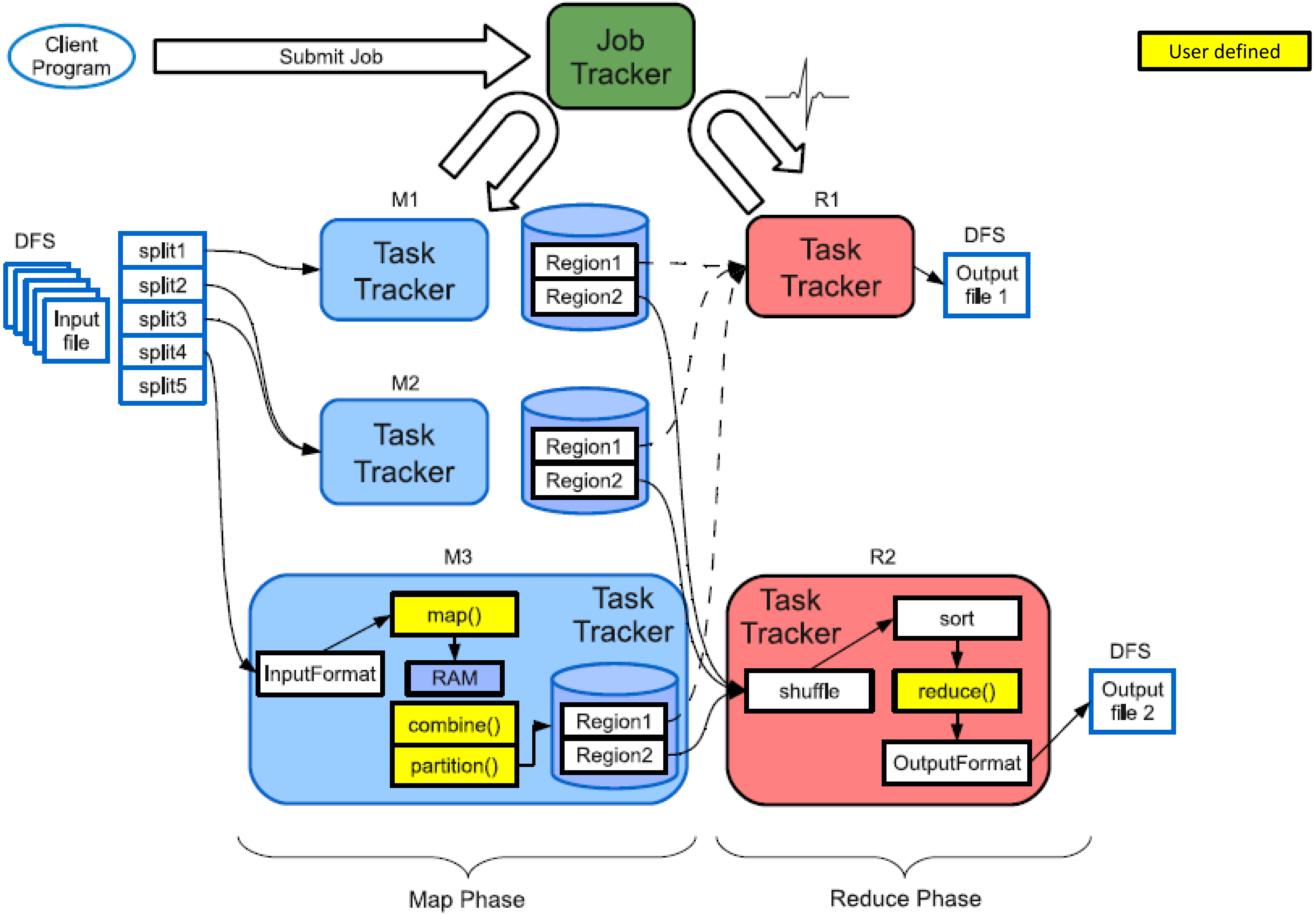
- Jobs are executed like a Unix pipeline:

```
□ cat * | grep | sort | uniq -c | cat    > output
□ Input | Map  | Shuffle & Sort | Reduce | Output
```

- Workflow
  - □ *input phase:* generates a number of FileSplits from input files (one per Map task)
  - □ *map phase:* executes a user function to transform input kv-pairs into a new set of kv-pairs
  - □ *sort & shuffle:* sort and distribute the kv-pairs to output nodes
  - □ *reduce phase:* combines all kv-pairs with the same key into new kv-pairs
  - □ *output phase* writes the resulting pairs to files

- All phases are distributed with many tasks doing the work
  - □ Framework handles scheduling of tasks on cluster
  - □ Framework handles recovery when a node fails

Hadoop MapReduce Engine



Hadoop Fault Tolerance

- Inputs are stored in a fault tolerant way by the DFS

- Mapper crashed
  - □ Detected when no report is given for a certain time
  - □ Restarted at a different node, reads a different copy of the same input split

- Reducer crashed
  - □ Detected when no report is given for a certain time
  - □ Restarted at a different node also. Pulls the results for its partition from each Mapper again.

- The key points are:
  - □ The input is redundantly available
  - □ Each intermediate result (output of the mapper) is materialized on disk
  - → Very expensive, but makes recovery of lost processes very simple and cheap

Higher-Level Languages

Goals

- Hide parallelization from programmer
- Offer a familiar way to formulate queries
- Provides optimization potential

Pig Latin

- SQL-inspired language
- Nested data model
- Operators resemble relational algebra
- Applies DB optimizations
- Compiled into MapReduce jobs

Pig Latin Example

- Schema

urls: (url, category, pagerank)

- SQL

```
SELECT category, AVG(pagerank) FROM urls
WHERE pagerank > 0.2
GROUP BY category
HAVING COUNT(*) > 10^6
```

- Pig Latin

```
good_urls    = FILTER urls BY pagerank > 0.2;
groups       = GROUP good_urls BY category;
big_groups   = FILTER groups BY COUNT(good_urls)>10^6 ;
output       = FOREACH big_groups GENERATE
                   category, AVG(good_urls.pagerank);
```

Distributed DBMS vs. MapReduce

	Distributed DBMS	MapReduce
Schema Support	☑	☒
Indexing	☑	☒
Programming Model	Stating what you want (declarative: SQL)	Presenting an algorithm (procedural: C/C++, Java, …)
Optimization	☑	☒
Scaling	1 – 500	10 - 5000
Fault Tolerance	Limited	Good
Execution	Pipelines results between operators	Materializes results between phases

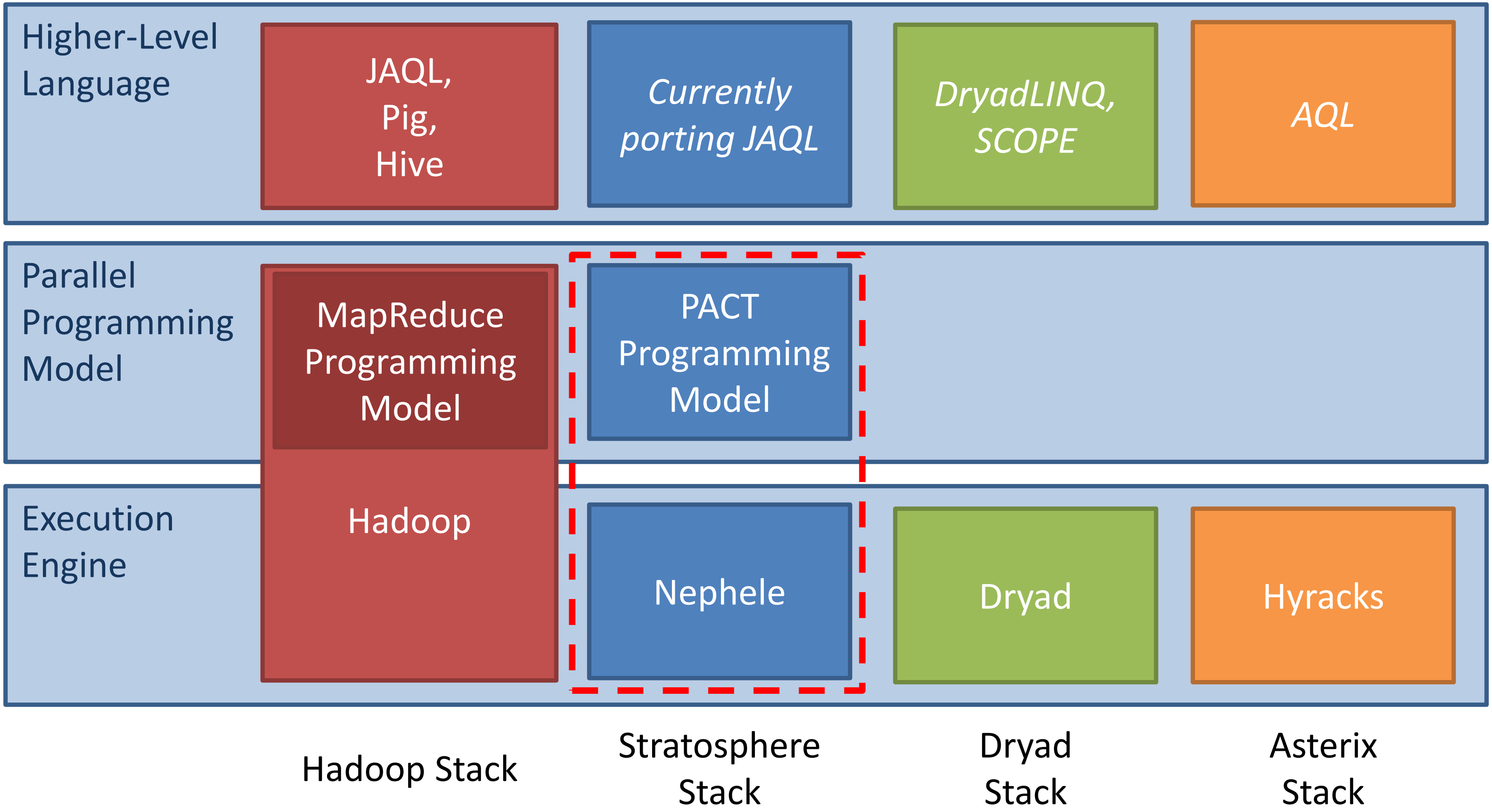
References & Further Reading

- Sanjay Ghemawat, Howard Gobioff, Shun-Tak Leung: The Google file system. SOSP 2003: 29-43
- J. Dean and S. Ghemawat. MapReduce: Simplified Data Processing on Large Clusters. In OSDI, 2004.
- Hadoop. URL: http://hadoop.apache.org.
- DeWitt, S. Madden, and M. Stonebraker. A Comparison of Approaches to Large-Scale Data Analysis. SIGMOD Conference 2009.
- C. Olston, B. Reed, U. Srivastava, R. Kumar, A. Tomkins: Pig Latin: A Not-So-Foreign Language for Data Processing. *In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08)*

Outline

- Background
  - Parallel Speedup
  - Levels of Parallelism
  - CPU Architecture
- Scale out
  - MapReduce
  - **Stratosphere**
- Scale up
  - Overview of Hardware Architectures
  - Parallel Programming Model
  - Relational Processing
  - Further Operations
  - Research Challenges of Hybrid Architectures

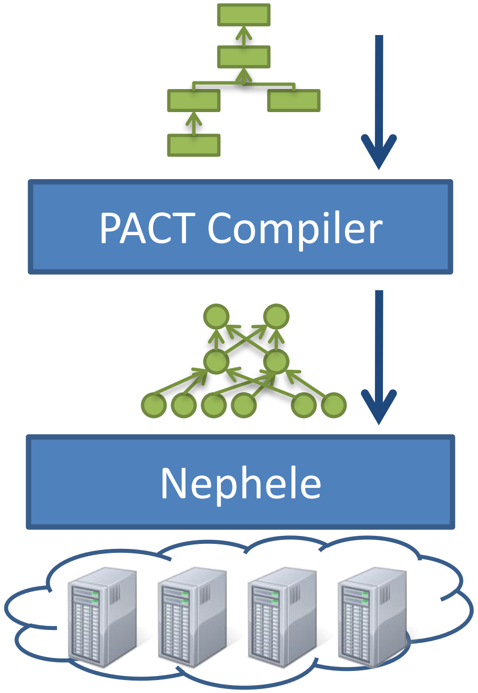
Comparing Architectural Stacks



	Hadoop Stack	Stratosphere Stack	Dryad Stack	Asterix Stack
Higher-Level Language	JAQL, Pig, Hive	*Currently porting JAQL*	*DryadLINQ, SCOPE*	*AQL*
Parallel Programming Model	MapReduce Programming Model	PACT Programming Model		
Execution Engine	Hadoop	Nephele	Dryad	Hyracks

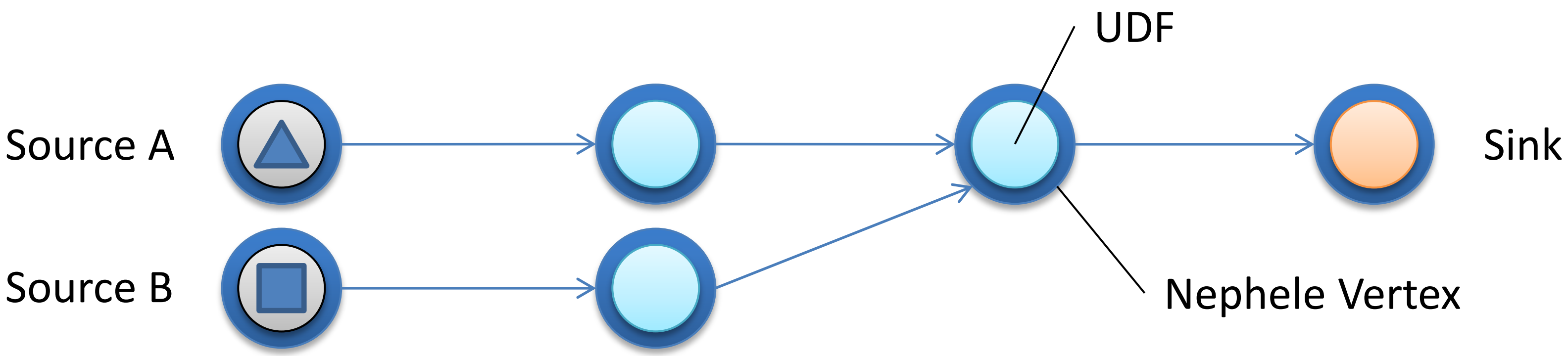
The Stratosphere System

- PACT Programming Model
  - Parallelization Contract (PACT)
  - Declarative definition of data parallelism
  - Centered around second-order functions
  - Generalization of MapReduce
- Nephele
  - Dryad-style execution engine
  - Evaluates dataflow graphs in parallel
  - Data is read from distributed file system
  - Flexible engine for complex jobs
  - Designed for IaaS environments
- Stratosphere = Nephele + PACT
  - Compiles PACT programs to Nephele dataflow graphs
  - Combines parallelization abstraction and flexible execution
  - Choice of execution strategies gives optimization potential
- Stratosphere is Open Source!

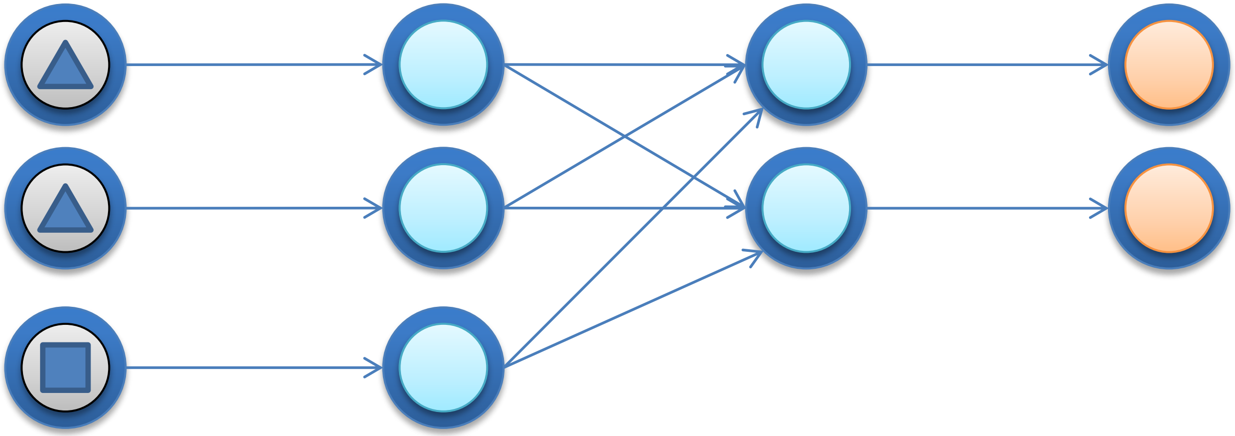


Nephele Data Flow Example

Define Dataflow



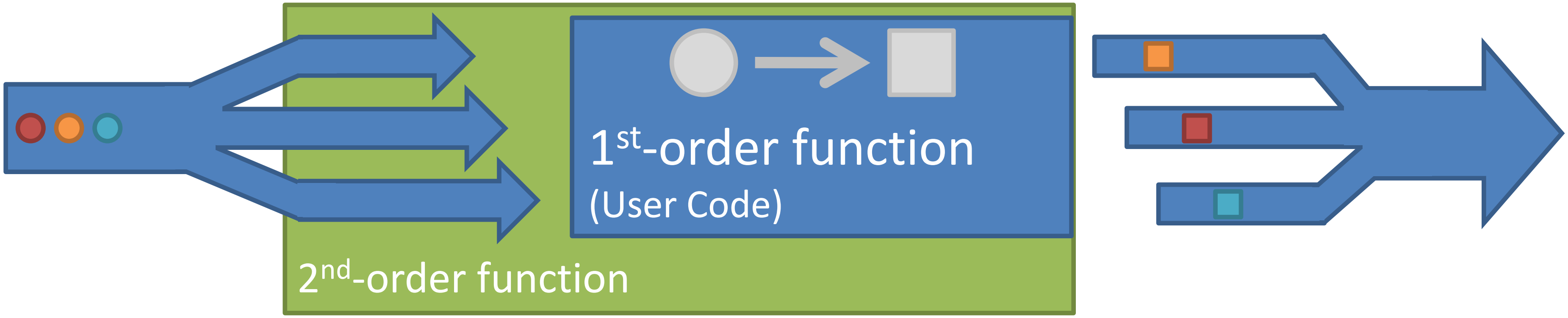
Parallelized Dataflow



Common Concepts of MapReduce and PACT

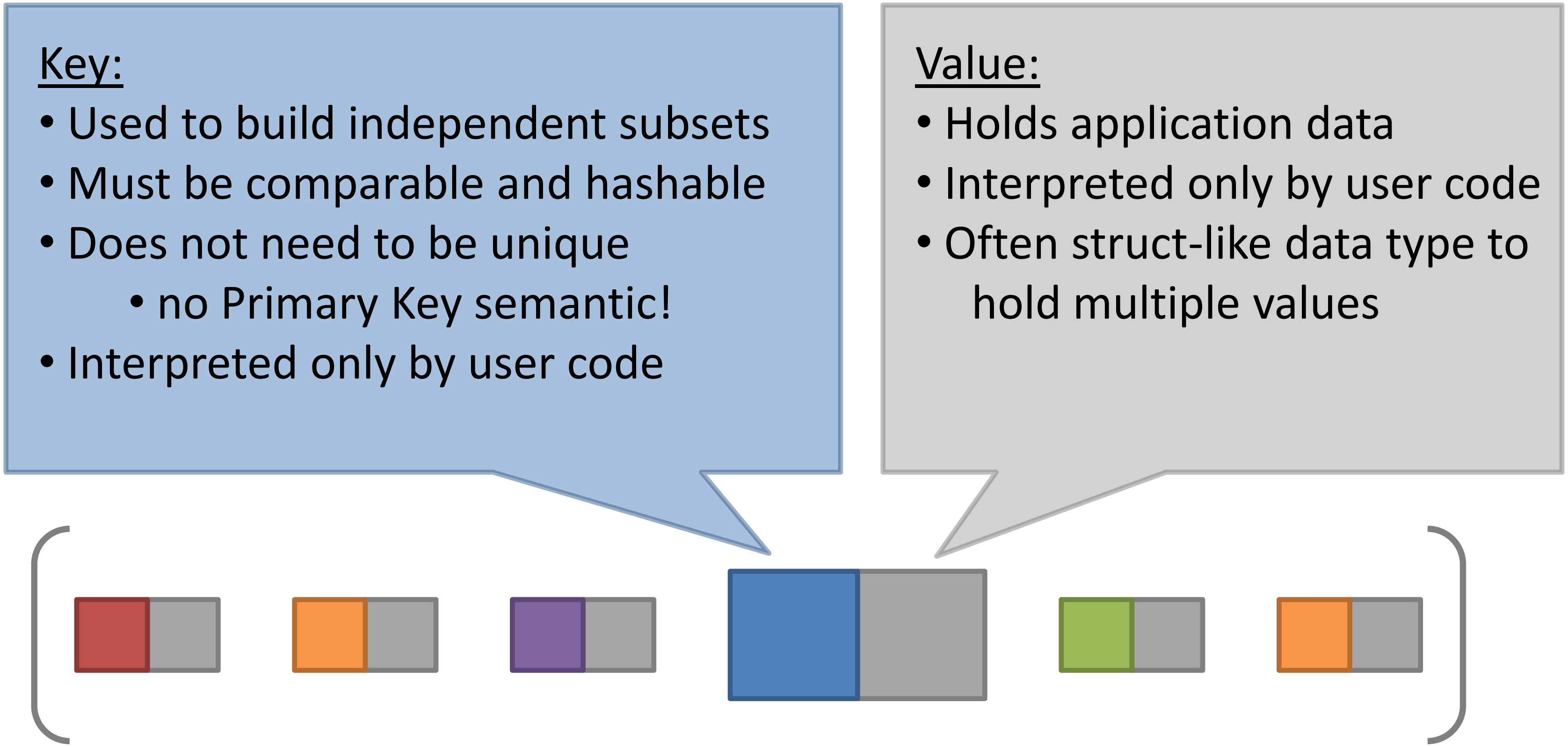
- PACT is a generalization and extension of MapReduce
  - PACT inherits many concepts of MapReduce

- Both are inspired by functional programming
  - Fundamental concept of programming model are 2nd-order functions
  - User writes 1st-order functions (user functions)
  - User code can be arbitrarily complex
  - 2nd-order function calls 1st-order function with independent data subsets
  - No common state should be held between calls of user function



Common Concepts of MapReduce and PACT

- Both use a common data format
  - Data is processed as pairs of keys and values
  - Keys and Values can be arbitrary data structures



Key:
- Used to build independent subsets
- Must be comparable and hashable
- Does not need to be unique
  - no Primary Key semantic!
- Interpreted only by user code

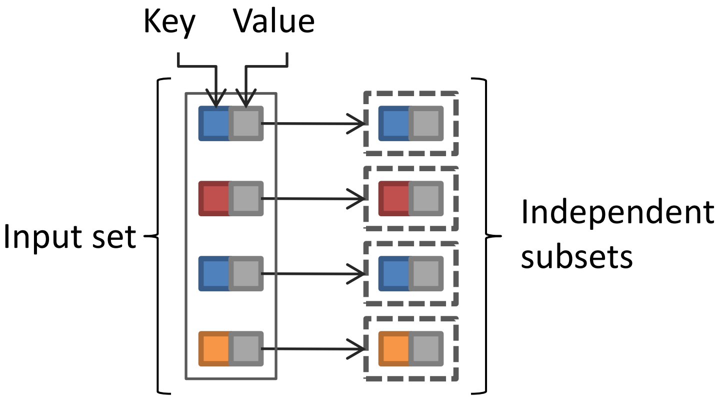
Value:
- Holds application data
- Interpreted only by user code
- Often struct-like data type to hold multiple values

MapReduce Programming Model

- MapReduce provides two 2nd-order functions

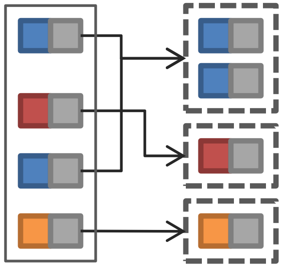
Map:

- All pairs are independently processed

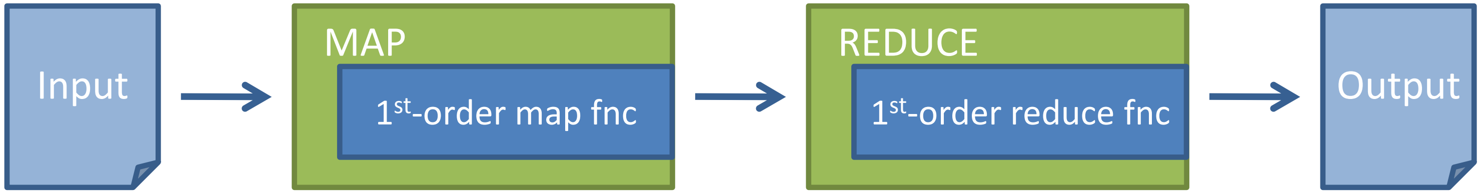


Reduce:

- Pairs with identical keys are grouped
- Groups are independently processed



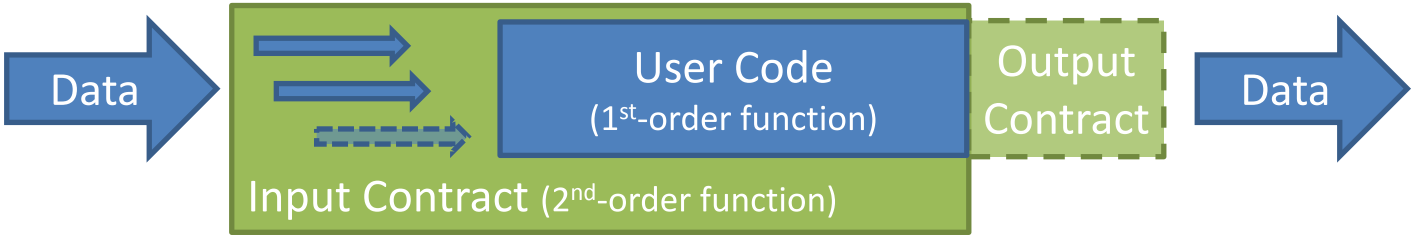
- MapReduce programs have a fixed structure:



Input → MAP 1st-order map fnc → REDUCE 1st-order reduce fnc → Output

PACT Programming Model

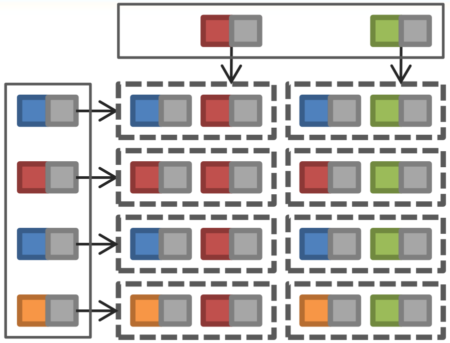
- Generalization and Extension of MapReduce Programming Model
- Based on Parallelization Contracts (PACTs)



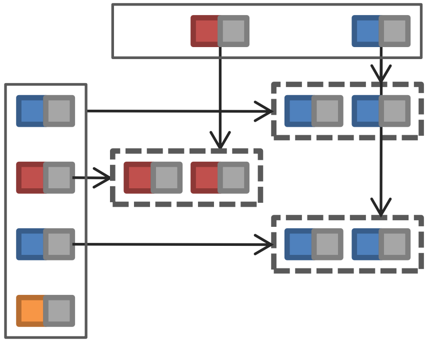
- Input Contract
  - □ 2nd-order function; generalization of Map and Reduce
  - □ Generates independently processable subsets of data
- User Code
  - □ 1st-order function
  - □ For each subset independently called
- Output Contract
  - □ Describes properties of the output of the 1st-order function
  - □ Optional but enables certain optimizations

Input Contracts beyond Map and Reduce

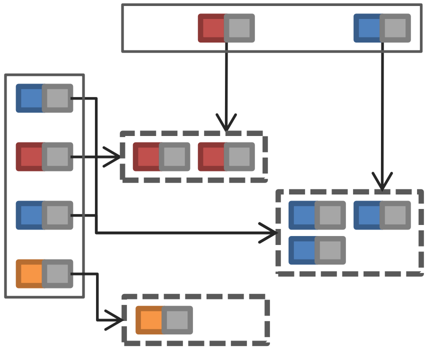
- **Cross**
  - Builds a Cartesian Product
  - Elements of CP are independently processed



- **Match**
  - Performs an equi-join on the key
  - Join candidates are independently processed

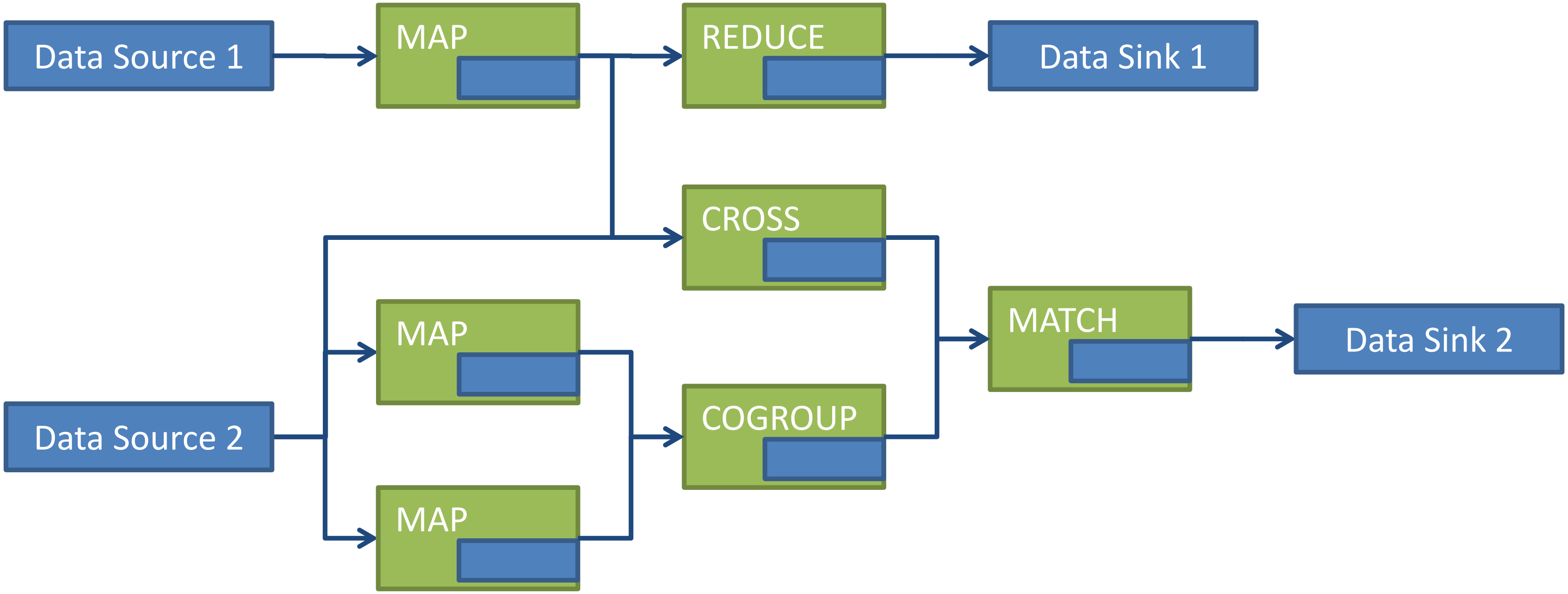


- **CoGroup**
  - Groups each input on key
  - Groups with identical keys are processed together



PACT Programming Model

- PACT Programs are data flow graphs
  - Data comes from sources and flows to sinks
  - PACTs process data in-between sources and sinks
  - Multiple sources and sinks allowed
  - Arbitrary complex directed acyclic data flows can be composed



Optimization of PACT Programs

- Optimization Opportunities
  - Declarative definition of data parallelism (Input Contracts)
  - Annotations reveal user code behavior (Output Contracts)
  - Compiler hints improve intermediate size estimates
  - Flexible execution engine

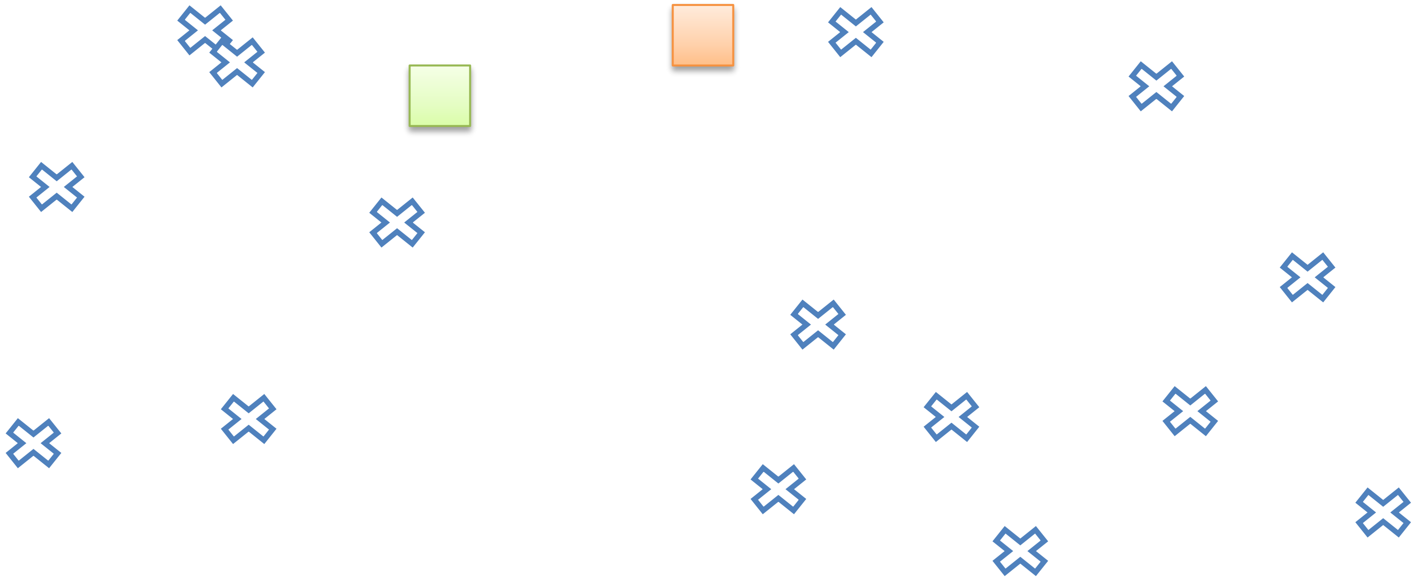
- PACT Optimizer
  - Compiles PACT programs to Nephele DAGs
  - Physical optimization as known from relational database optimizers
  - Avoids unnecessary expensive operations (partition, sort)
  - Chooses between local strategies (hash- vs. sort-based)
  - Chooses between ship strategies (partition, broadcast, local forward)
  - Sensitive to data input sizes and degree of parallelism

Example – K-Means Clustering

- Partition n points into x clusters:
  - Measure distance between points and clusters
  - Assign each point to a cluster
  - Move cluster to the center of associated points
  - Repeat until it converges

✖ Points

🟧 Center

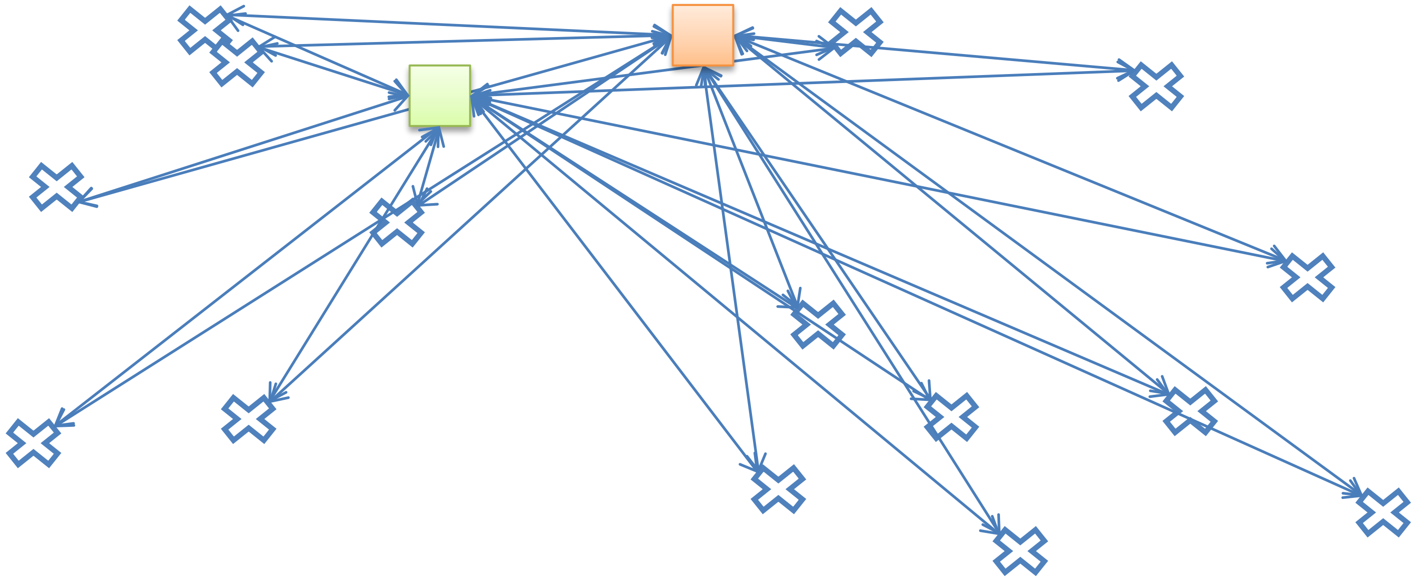


Example – K-Means Clustering

- Partition n points into x clusters:
  - **Measure distance between points and clusters**
  - Assign each point to a cluster
  - Move cluster to the center of associated points
  - Repeat until it converges

✖ Points

🟧 Center

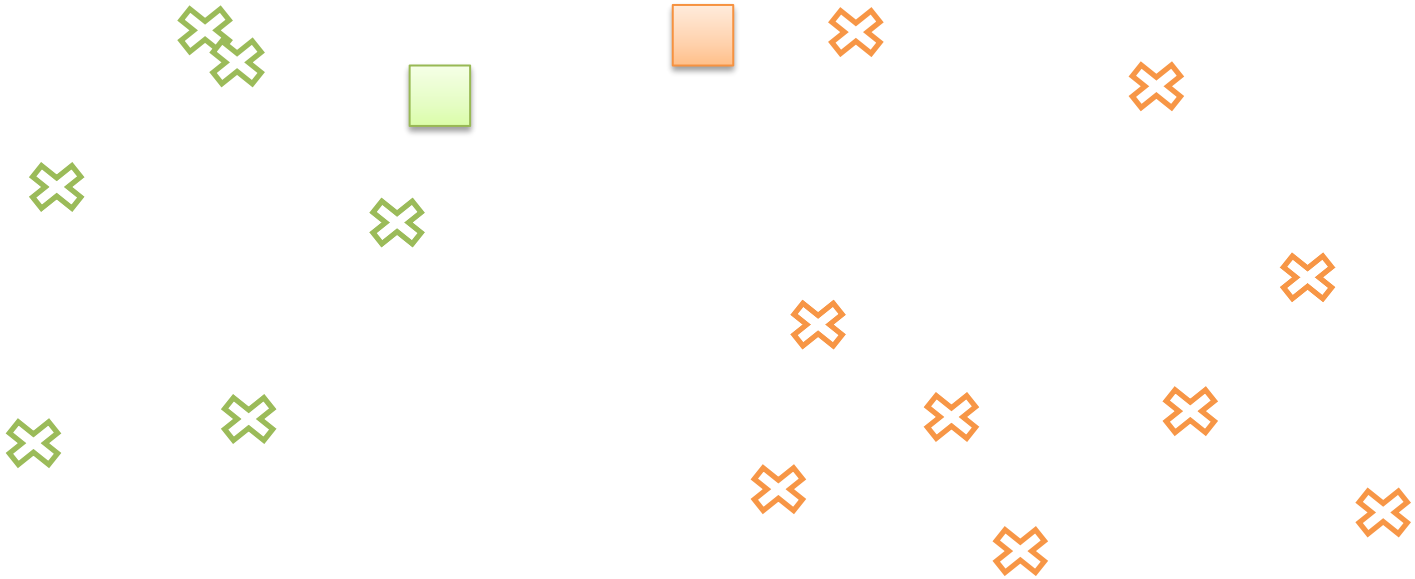


Example – K-Means Clustering

- Partition n points into x clusters:
  - □ Measure distance between points and clusters
  - □ **Assign each point to a cluster**
  - □ Move cluster to the center of associated points
  - □ Repeat until it converges

✖ Points

🟧 Center

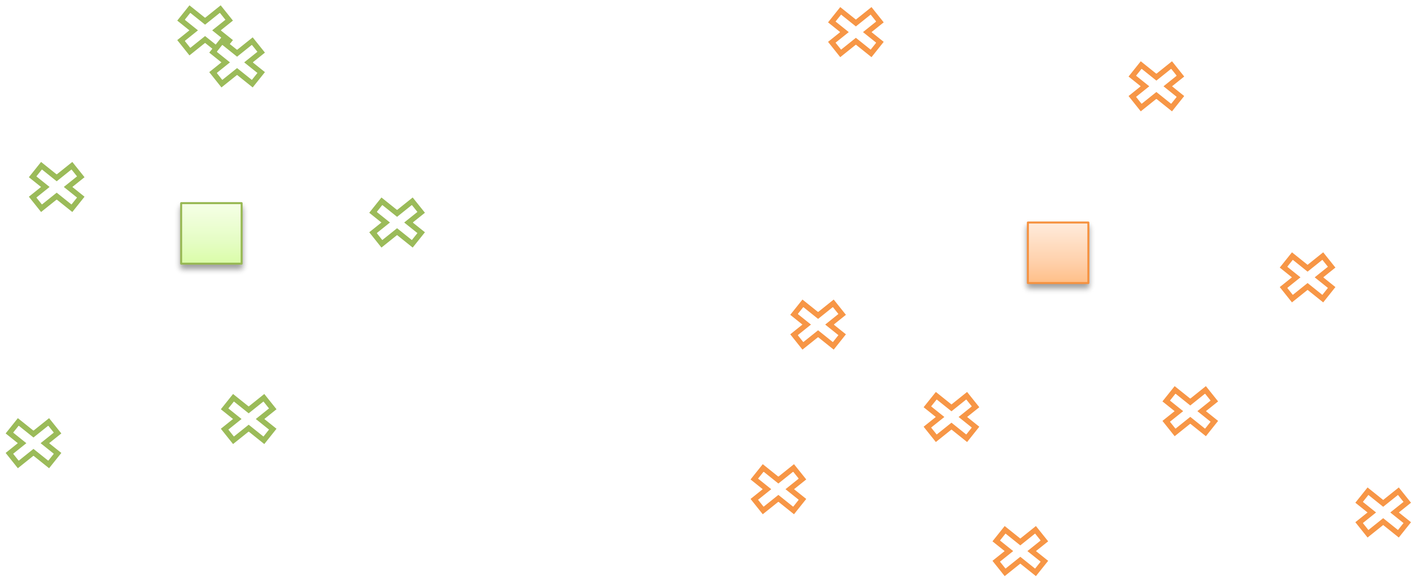


Example – K-Means Clustering

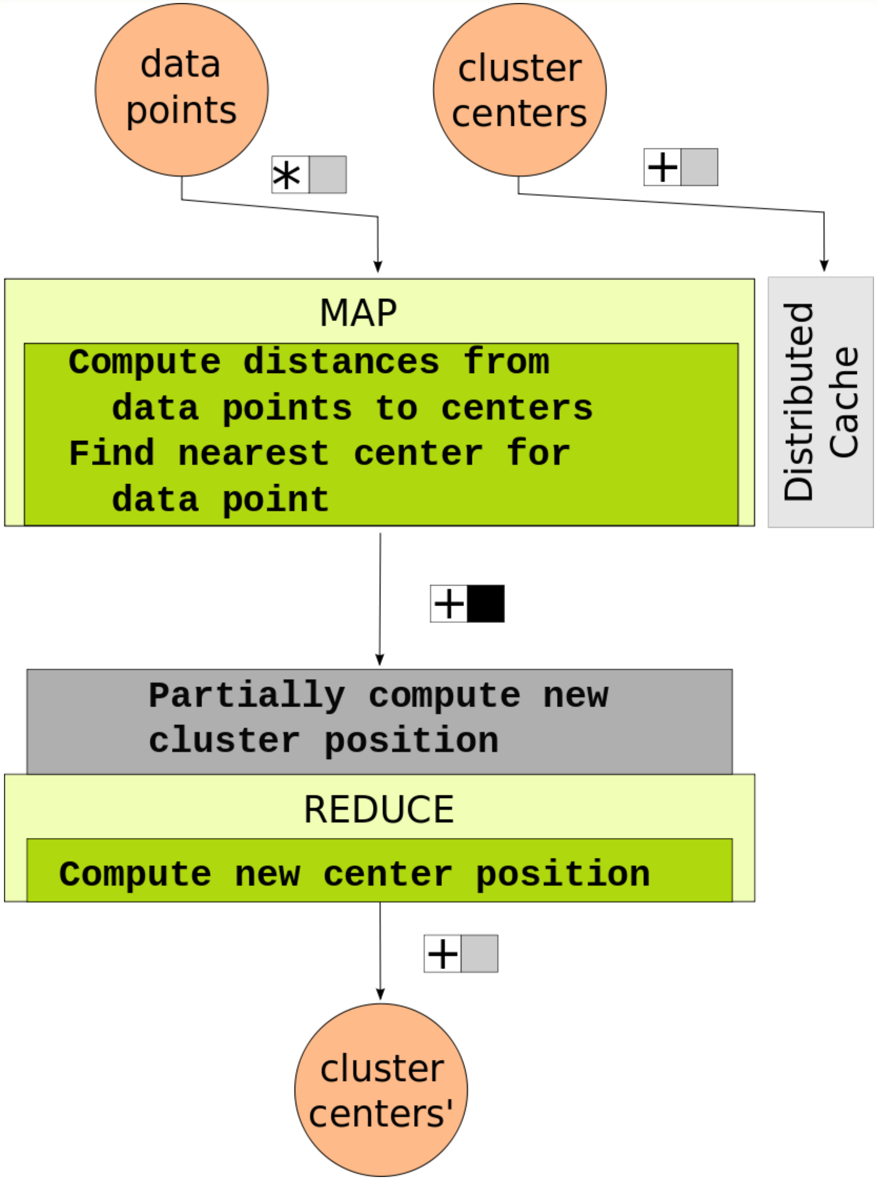
- Partition n points into x clusters:
  - □ Measure distance between points and clusters
  - □ Assign each point to a cluster
  - □ **Move cluster to the center of associated points**
  - □ Repeat until it converges

✖ Points

🟧 Center

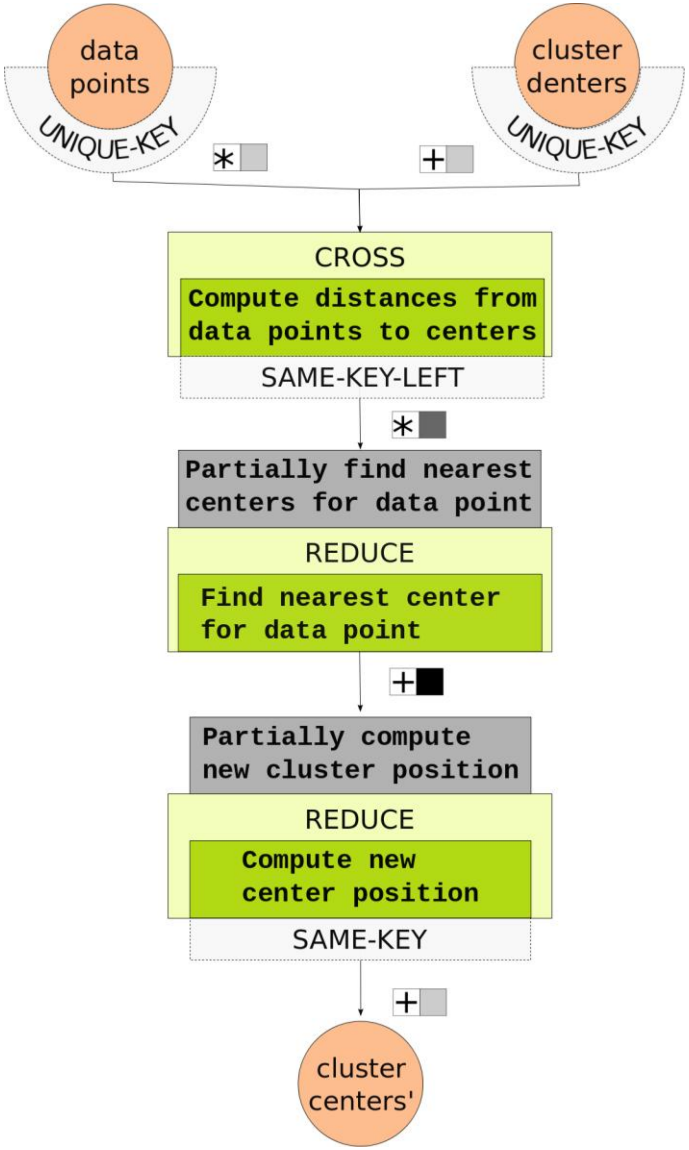


K-Means with MapReduce



data points
cluster centers
*
+
MAP
Compute distances from data points to centers
Find nearest center for data point
Distributed Cache
+
Partially compute new cluster position
REDUCE
Compute new center position
+
cluster centers'

K-Means with Stratosphere


data points
UNIQUE-KEY
cluster denters
UNIQUE-KEY
*
+
CROSS
Compute distances from data points to centers
SAME-KEY-LEFT
*
Partially find nearest centers for data point
REDUCE
Find nearest center for data point
+
Partially compute new cluster position
REDUCE
Compute new center position
SAME-KEY
+
cluster centers'

Comparing MapReduce and PACT

- Comparison from developer perspective

- PACT has same expressive power as MapReduce
  - □ At PACT’s current state

- In practice, many tricks must be played with MapReduce
  - □ Depends on the execution engine, such as Hadoop
  - □ Concatenation of two or more MapReduce programs
  - □ Auxiliary structures (e.g., Distributed Cache, Counters)
  - □ Multiple inputs (Union of inputs and flagging the source)
  - □ Overriding other parametric functions (sort, partition, split, …)

- Tricks have drawbacks
  - □ Programming becomes much more difficult
  - □ Knowledge of execution model must be exploited
  - □ Parallelization strategies are hardcoded

Features of Upcoming Release

- Record Data Model
  - Generalization of Key-Value Pair Data Model
  - Records with arbitrary many Key and Value fields
  - Improved optimization potential

- Significant Runtime Improvements
  - Extensive usage of mutable objects
  - Reduced instantiation and garbage collection overhead

- Chained Mappers and Combiners
  - Map und Combine Tasks are directly chained
  - Less setup costs for data flow
  - No serialization overhead

Stratosphere published as Open Source

- Published Components
  - PACT Programming Model
  - PACT Optimizer
  - Nephele Execution Engine

- Extensive documentation
  - Local and Distributed Setup Instructions
  - PACT Programming How-to and Best Practices
  - PACT Example Programs
  - Mailing Lists and more…

http://www.stratosphere.eu

References & Further Reading

- Alexander Alexandrov, Stephan Ewen, Max Heimel, Fabian Hueske, Odej Kao, Volker Markl, Erik Nijkamp, Daniel Warneke: MapReduce and PACT - Comparing Data Parallel Programming Models. *Proceedings of the 14th Conference on Database Systems for Business, Technology, and Web (BTW '11)*

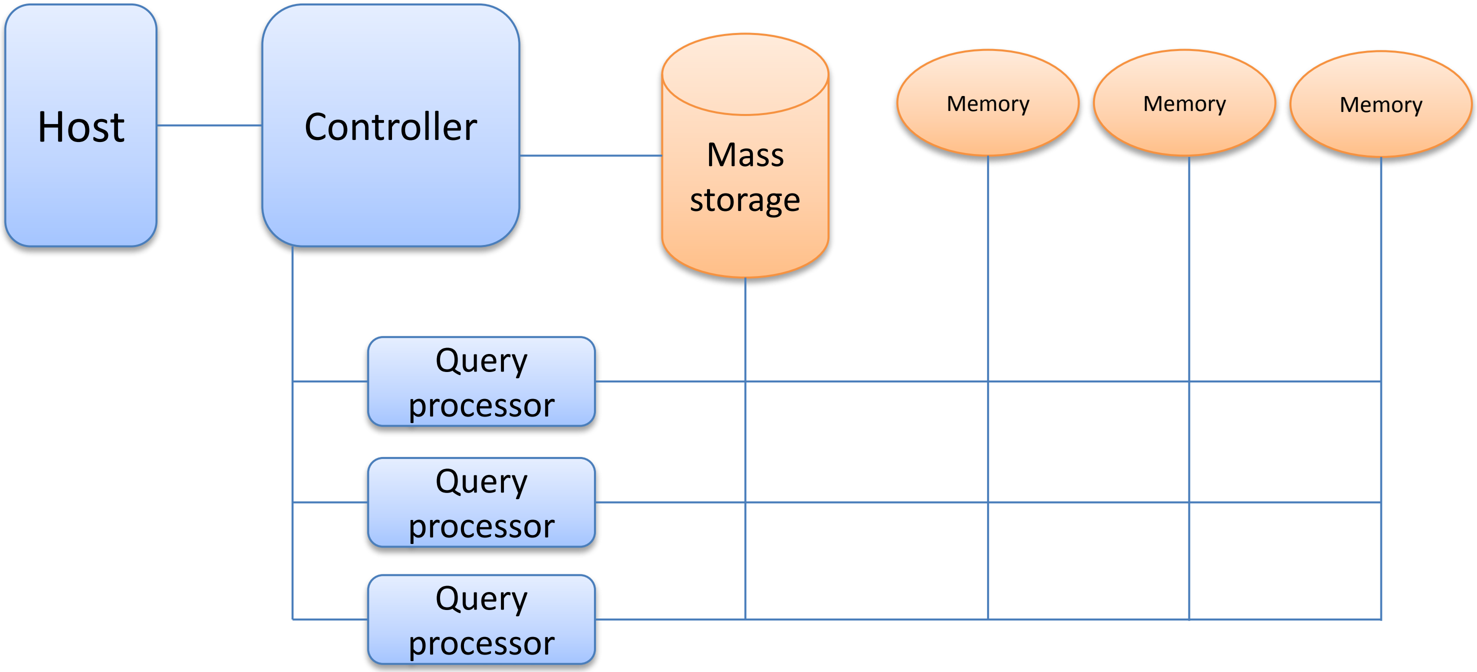
- Dominic Battré, Stephan Ewen, Fabian Hueske, Odej Kao, Volker Markl, and Daniel Warneke: Nephele/PACTs: A Programming Model and Execution Framework for Web-Scale Analytical Processing. *In Proceedings of the ACM Symposium on Cloud Computing (SoCC) 2010 ACM, pp. 119–130*

Outline

- Background
  - Parallel Speedup
  - Levels of Parallelism
  - CPU Architecture
- Scale out
  - MapReduce
  - Stratosphere
- Scale up
  - **Overview of Hardware Architectures**
  - Parallel Programming Model
  - Relational Processing
  - Further Operations
  - Research Challenges of Hybrid Architectures

A blast from the past: DIRECT

- Specialized database processors already in 1978
- Back then could not keep up with rise of commodity CPUs
- Today is different: memory wall, ILP wall, power wall..

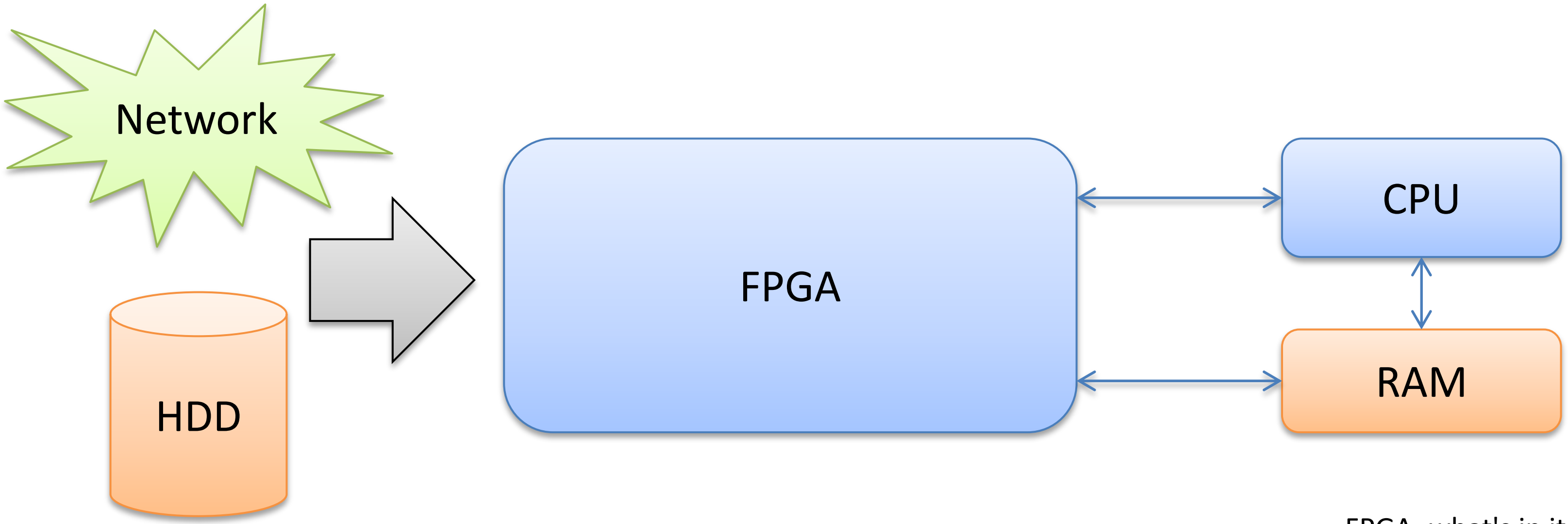


DIRECT - a multiprocessor organization for supporting relational data base management systems

David J. DeWitt

FPGAs

- Massively parallel & reconfigurable processors
- Lower clock speeds, but configuration for specific tasks makes up for this
- Very difficult to program
- Used in real world appliances: Kickfire/Teradata, Netezza/IBM..

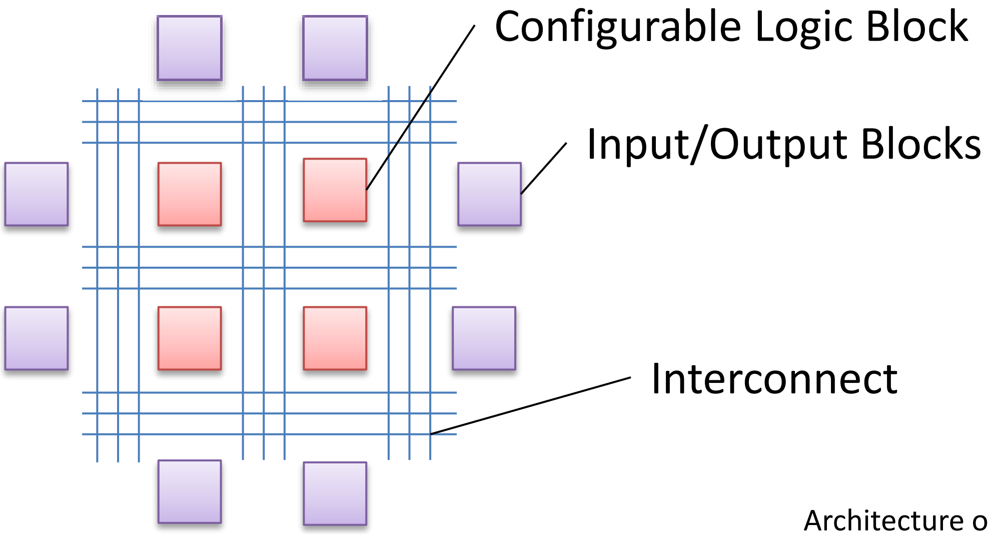


FPGA: what's in it for a database?

Jens Teubner and Rene Mueller

FPGAs

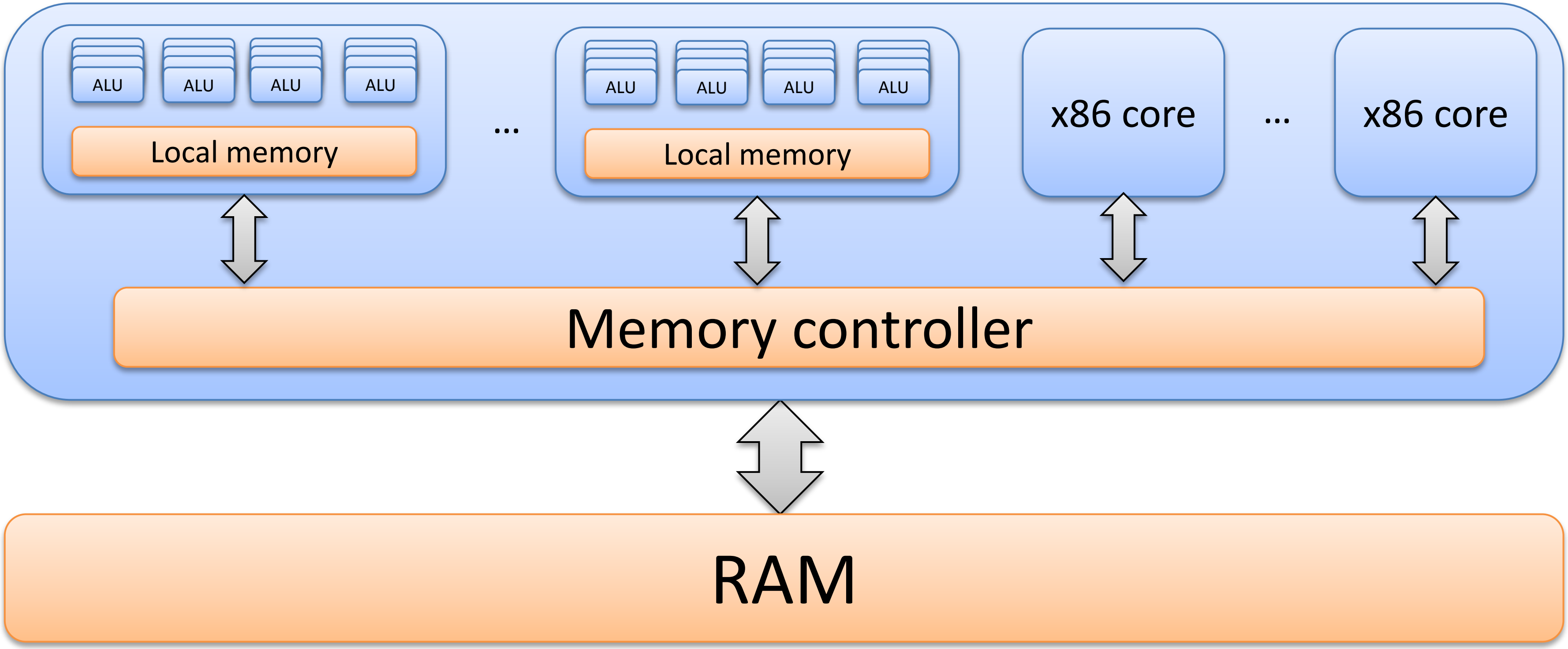
- Input/Output blocks: interface to outside resources
- Configurable Logic Block (CLB) consists of
  - □ Logic Gate (e.g., n-Lookup Table)
  - □ Carry logic
  - □ Storage elements (e.g., D flip flop)
- May contain hard intellectual property (IP) cores
  - □ Discrete chips for common functionality: Block RAM, PowerPC cores



Architecture of FPGAs and CPLDs: A Tutorial
Stephen Brown and Jonathan Rose

CPU/GPU hybrid (AMD Fusion)

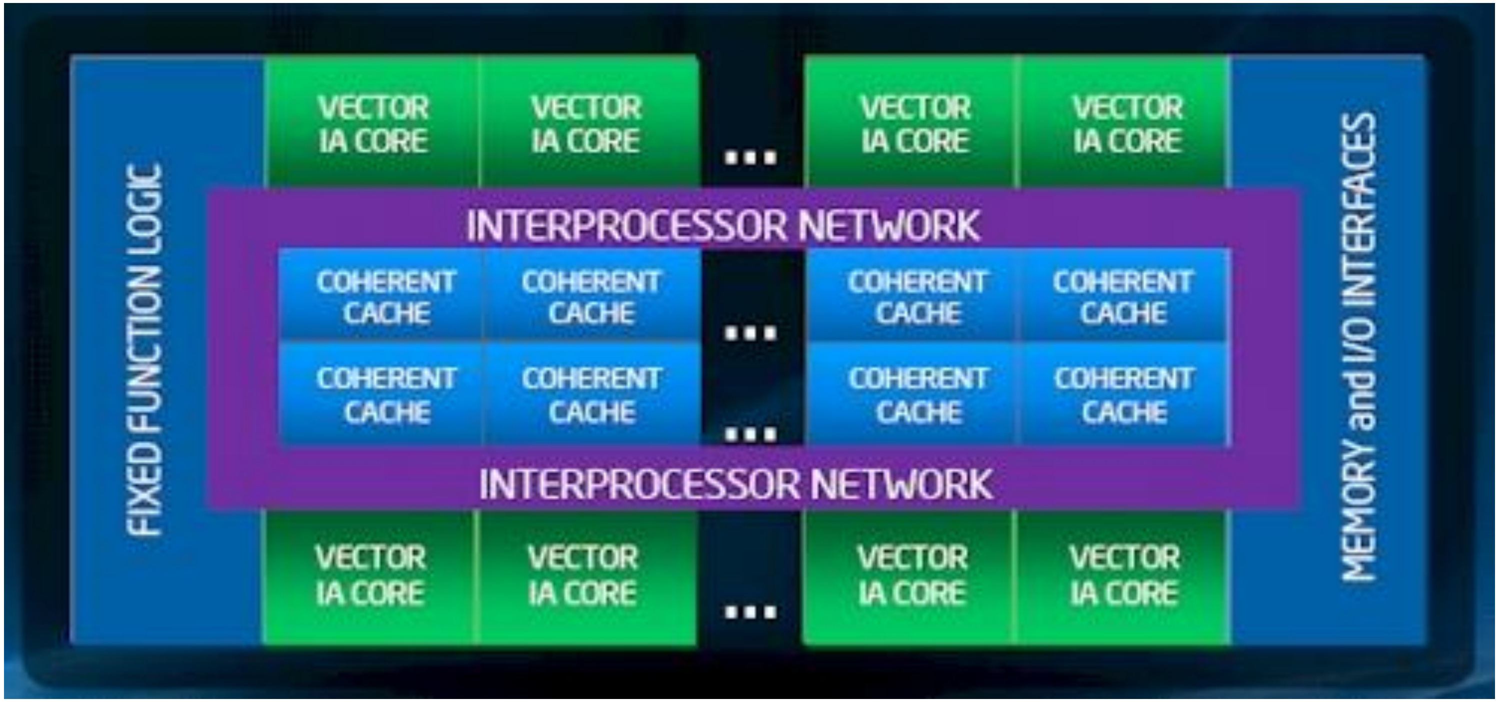
- CPU & GPU integrated on one die, shared memory
- Uses GPU programming model (OpenCL)
- No PCIe-bottleneck but only memory bandwidth like CPU



AMD Whitepaper: AMD Fusion Family of APUs

Intel MIC

- First product: Knights Corner
- Focus: High Performance Computing (HPC)
- > 50 cores on a single chip
- Same tools, compilers, libraries as Intel® Xeon processors
  - Portability



Source: theregister.co.uk

There is more...

- Network processors
  - Many-core architectures
  - Associative memory (constant time search)
- CELL
  - Similar to CPU/GPU hybrid: CPU core + several wide SIMD cores w/ local scratchpad memories (similar to L1 cache)

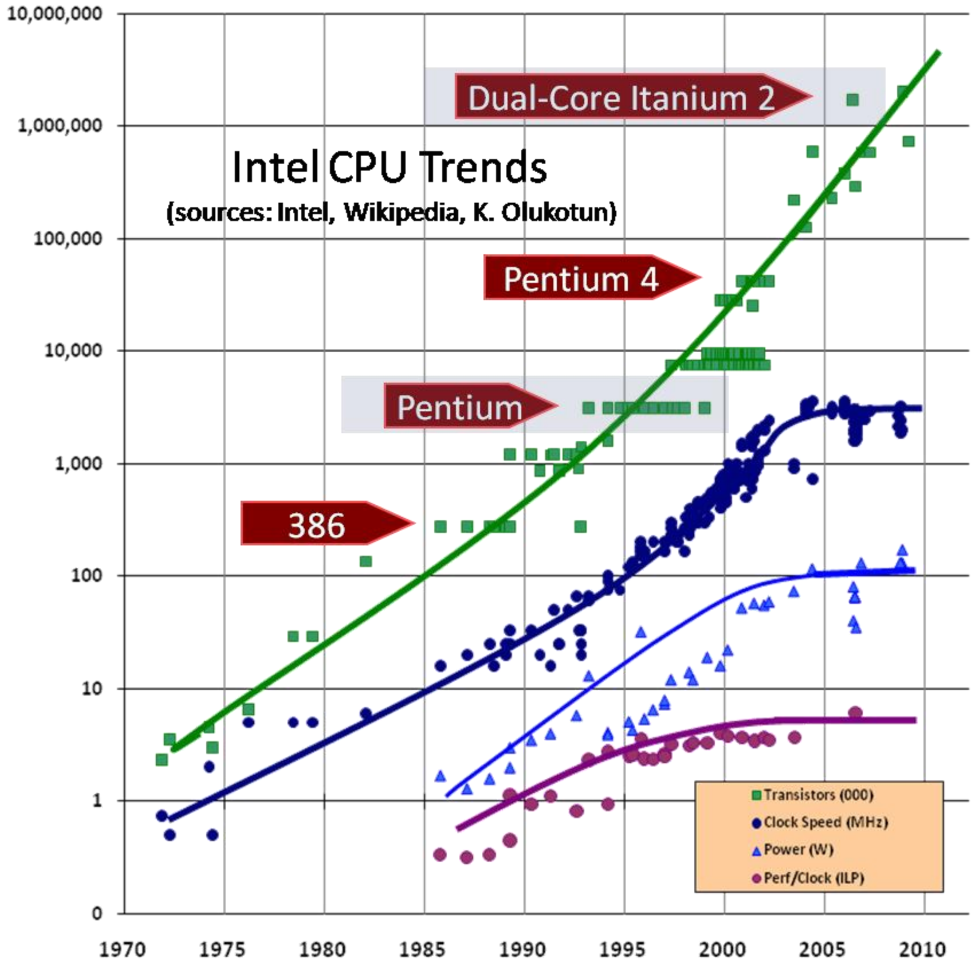
Common goals, common problems

- Similar approaches
  - Many-core
  - Distributed on-chip memory
  - Hope for better perf/$ and perf/watt than traditional CPUs
- Similar difficulties
  - Memory bandwidth always a bottleneck
  - Hard to program
    - parallel
    - low-level & architecture-specific

References & Further Reading

- D. J. DeWitt: DIRECT - a multiprocessor organization for supporting relational data base management systems. *In Proceedings of the 5th annual symposium on Computer architecture (ISCA '78). ACM, New York, NY, USA, 182-189.*
- R. Mueller and J. Teubner: FPGA: what's in it for a database?. *In Proceedings of the 35th SIGMOD international conference on Management of data (SIGMOD '09), Carsten Binnig and Benoit Dageville (Eds.). ACM, New York, NY, USA, 999-1004.*
- S. Brown, J. Rose: FPGA and CPLD Architectures: A Tutorial. *In IEEE Design and Test of Computers, Volume 13, Number 2, pp. 42--57, 1996*
- AMD White paper: AMD Fusion family of APUs, http://sites.amd.com/us/Documents/48423B_fusion_whitepaper_WEB.pdf (Visited May 2011)
- N. Bandi, A. Metwally, D. Agrawal, and A. El Abbadi: Fast data stream algorithms using associative memories. *In Proceedings of the 2007 ACM SIGMOD international conference on Management of data (SIGMOD '07). ACM, New York, NY, USA, 247-256.*
- B. Gedik, P. S. Yu, and R. R. Bordawekar: Executing stream joins on the cell processor. *In Proceedings of the 33rd international conference on Very large data bases (VLDB '07). VLDB Endowment 363-374.*
- Intel: Many Integrated Core (MIC) Architecture – Advanced. http://www.intel.com/content/www/us/en/architecture-and-technology/many-integrated-core/intel-many-integrated-core-architecture.html (Visited June 2012)

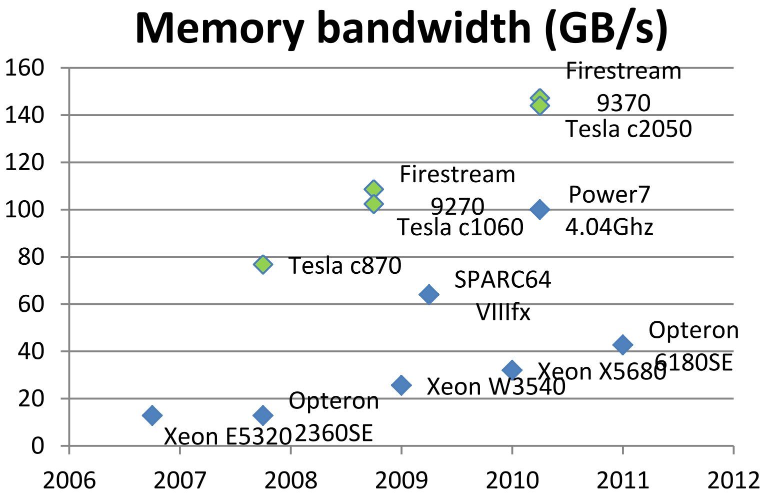
A case for GPU architectures

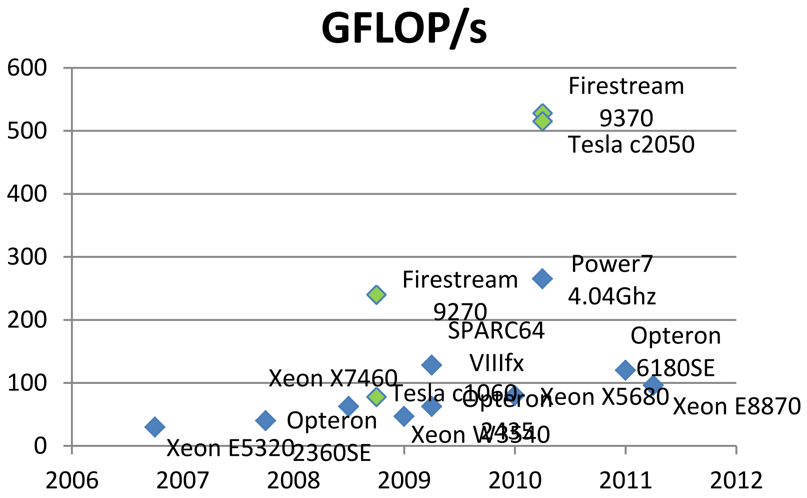


- Trends in processor architecture
  - The free lunch is over: power wall, memory wall, ILP wall…
- Motivates a look at alternatives to traditional CPU chips
  - GPUs: cheap, massively parallel, available today
  - „Fruit fly“ architecture for developing parallel algorithms and systems

GFLOPS & GB/s

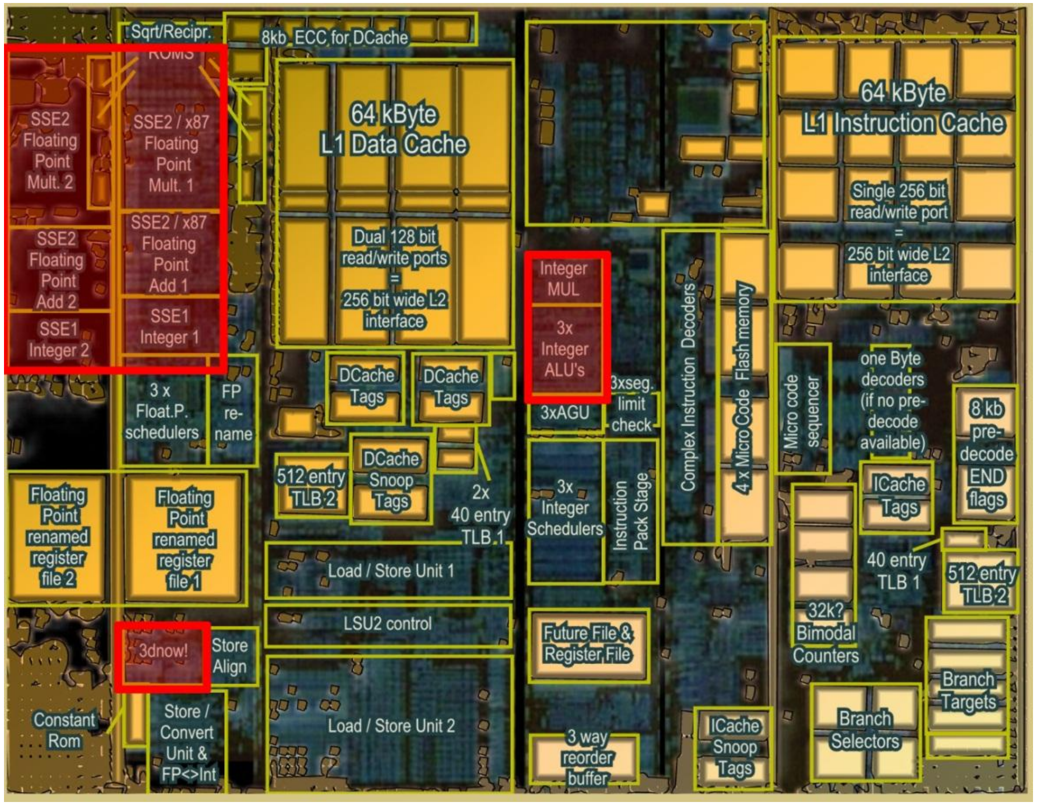
- GFLOPS & GB/s for different CPU & GPU chips..
- Beware: theoretical peak numbers





Wikipedia, Intel, AMD, Nvidia, IBM

What is the difference? Look at a modern CPU..

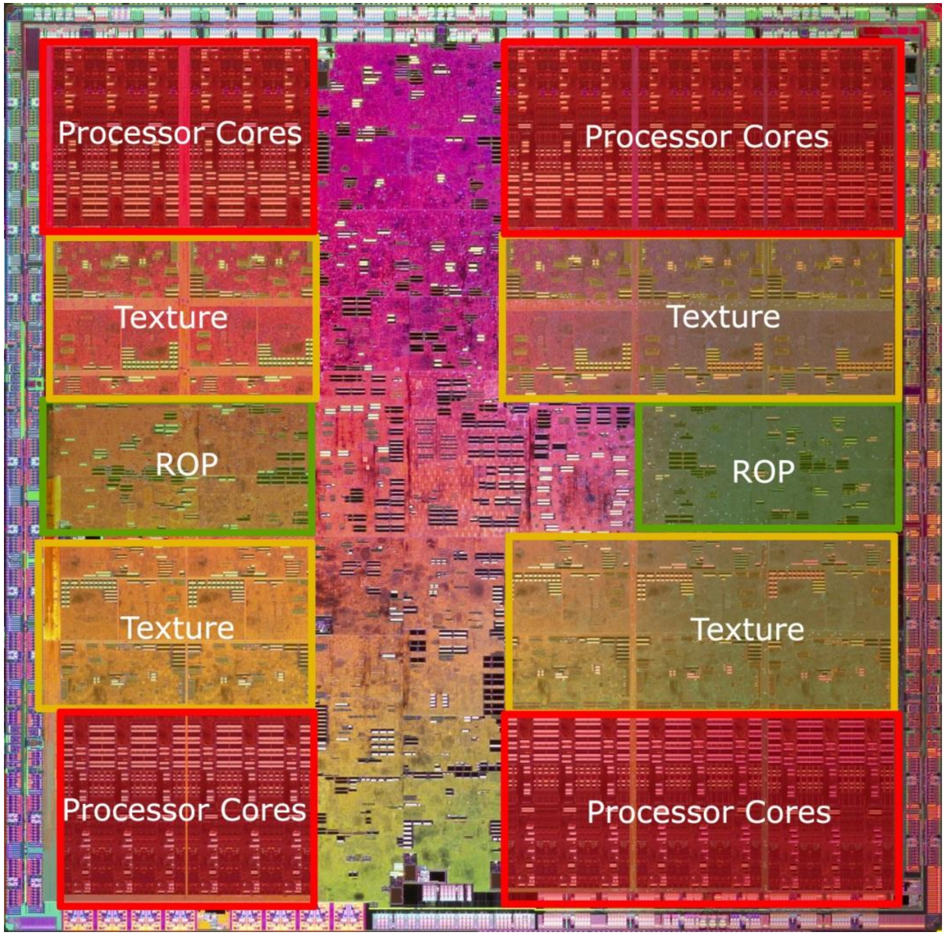

Sqrt/Recipr. ROMS
8kb ECC for DCache
64 kByte L1 Data Cache
64 kByte L1 Instruction Cache
SSE2 Floating Point Mult. 2
SSE2 / x87 Floating Point Mult. 1
SSE2 / x87 Floating Point Add 1
SSE2 Floating Point Add 2
SSE1 Integer 1
SSE1 Integer 2
Dual 128 bit read/write ports = 256 bit wide L2 interface
Single 256 bit read/write port = 256 bit wide L2 interface
Integer MUL
3x Integer ALU's
3xAGU
3xseg. limit check
Complex Instruction Decoders
4 x Micro Code Flash memory
Micro code sequencer
one Byte decoders (if no pre-decode available)
8 kb pre-decode END flags
3 x Float.P. schedulers
FP re-name
DCache Tags
DCache Tags
DCache Snoop Tags
512 entry TLB 2
2x 40 entry TLB 1
3x Integer Schedulers
Instruction Pack Stage
ICache Tags
40 entry TLB 1
512 entry TLB 2
Floating Point renamed register file 2
Floating Point renamed register file 1
Load / Store Unit 1
LSU2 control
32k? Bimodal Counters
3dnow!
Store Align
Future File &amp; Register File
Branch Targets
Constant Rom
Store / Convert Unit &amp; FP&lt;&gt;Int
Load / Store Unit 2
3 way reorder buffer
ICache Snoop Tags
Branch Selectors

AMD K8L

www.chip-architect.com

- Most die space devoted to control logic & caches
- Maximize performance for *arbitrary*, *sequential* programs

And at a GPU..

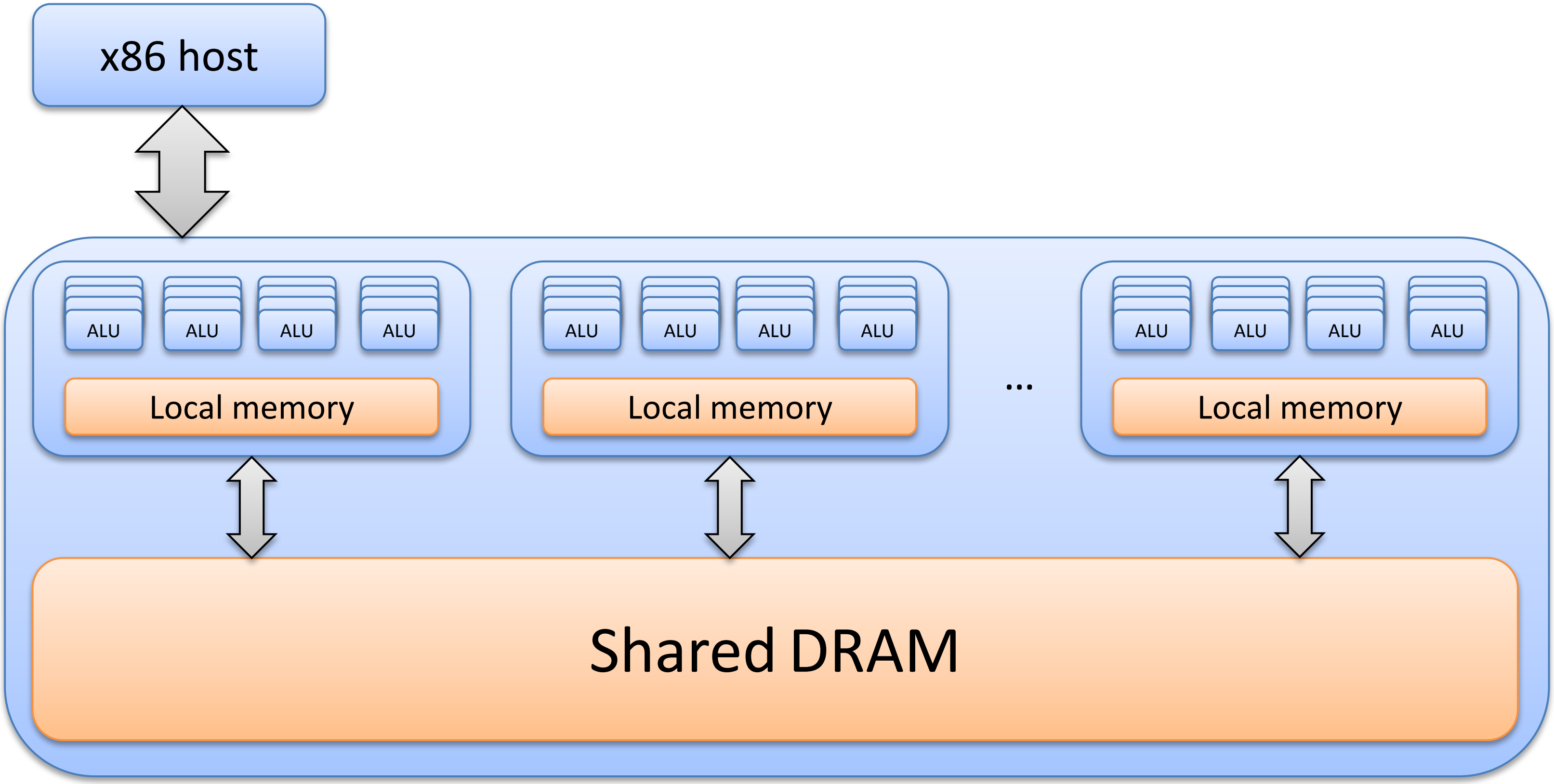


Nvidia G200 www.beyond3d.com

- Little control logic, a lot of execution logic
- Maximize parallel computational throughput

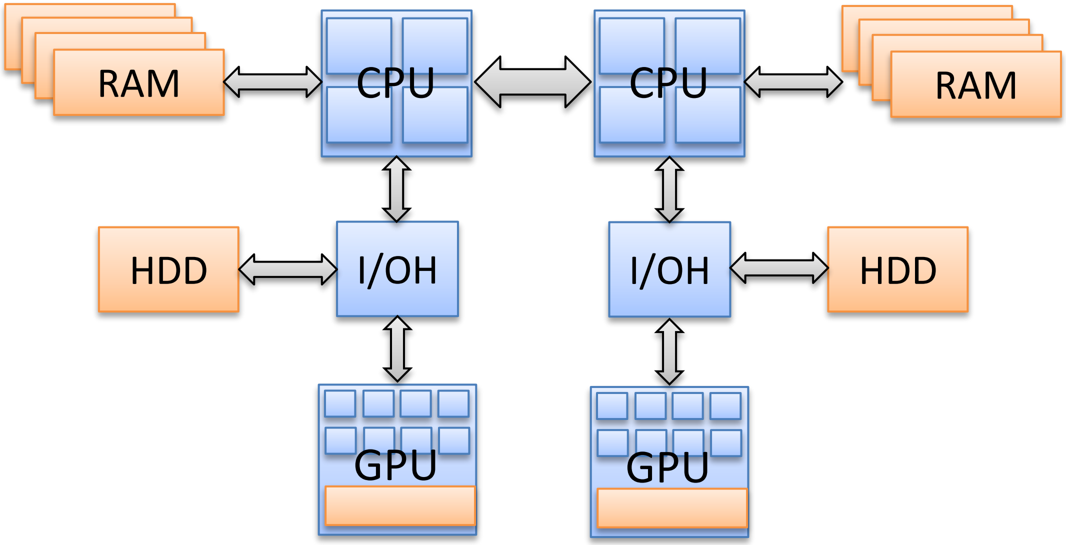
GPU architecture

- SIMD cores with small local memories (16-48KiB)
- Shared high bandwidth + high latency DRAM (1-6GiB)
- Multi-threading hides memory latency
- Bulk-synchronous execution model



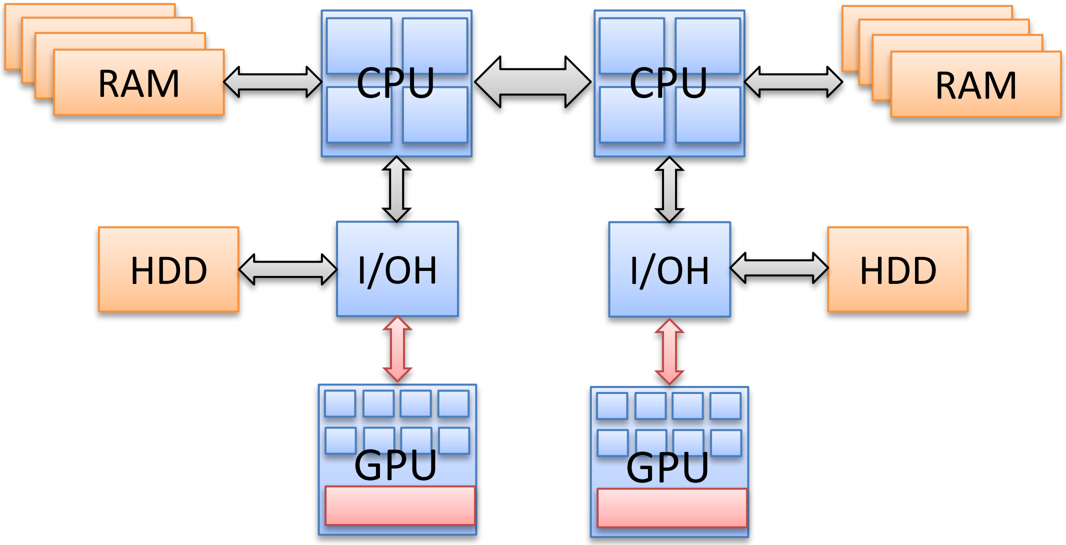
Hybrid CPU+GPU system

- 1 or several (interconnected) multicore CPUs
- 1 or several GPUs



Bottlenecks

- PCIe bandwidth & latency
- GPU memory size
- GPU has no direct access to storage & network



GPUs & databases: opportunities

- Most database operations well-suited for massively parallel processing
- Efficient algorithms for many database primitives exist
  - Scatter, reduce, prefix sum, sort..
- Find ways to utilize higher bandwidth & arithmetic throughput

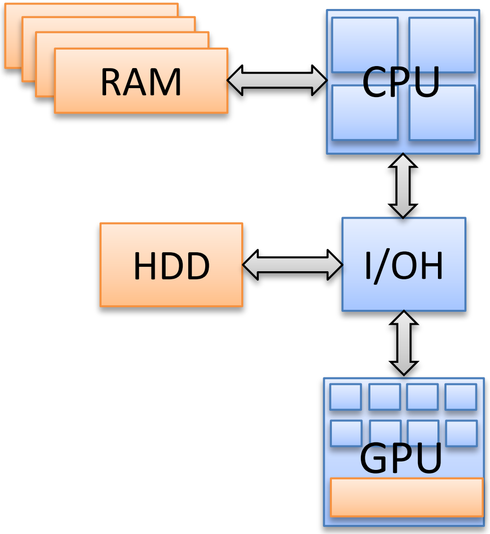
www.nvidia.com

GPUs & databases: challenges & constraints

- Find a way to live with architectural constraints
  - GPU to CPU bandwidth (PCIe) smaller than CPU to RAM bandwidth
  - Fast GPU memory smaller than CPU RAM
- Technical hurdles
  - GPU is a co-processor: needs CPU to orchestrate work, get data from storage devices etc..
  - GPU programming models (CUDA, OpenCL) are low level, need architecture-specific tuning
  - Limited support for multi-tasking

GDB: hybrid CPU+GPU query processing

- Fully-fledged CPU/GPU query processor
  - GPU hash indices, B+ trees, hash join, sort-merge join
  - Handles data sizes larger than GPU memory, supports non-numeric data types
- Executes queries either on CPU, GPU, or on both
  - Analytical models & calibration via micro-benchmarks used to estimate cost for execution on CPU vs. GPU
- System with 1 CPU + 1 GPU, only 1 query at a time

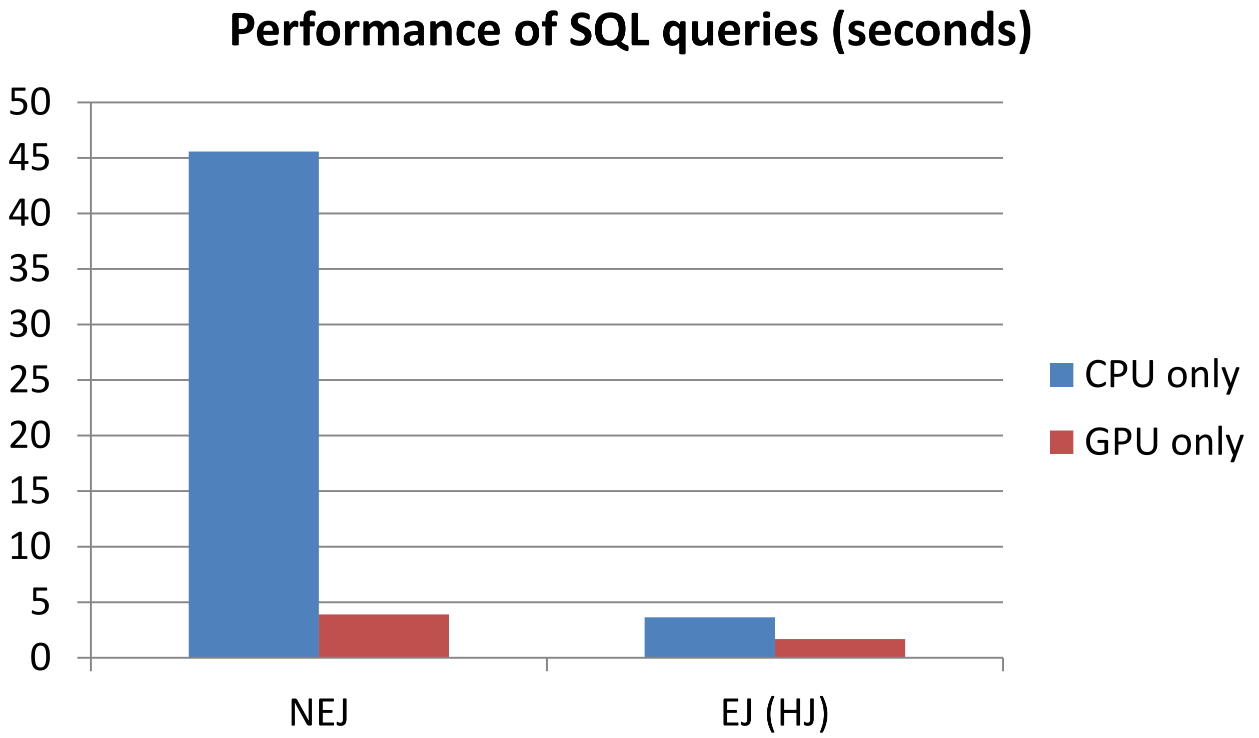


Relational query coprocessing on graphics processors

Bingsheng He and Mian Lu and Ke Yang and Rui Fang and Naga K. Govindaraju and Qiong Luo and Pedro V. Sander

GDB: results

- Intel Core 2 Duo, Nvidia Geforce 8800 GTX
- 2x-7x speedup for compute-intense operators *if data fits inside GPU memory*
- Mixed mode operators provide no speedup
- No speedup if data does not fit into GPU memory



References & Further Reading

- J. Owens: GPU architecture overview, *In ACM SIGGRAPH 2007 courses (SIGGRAPH '07). ACM, New York, NY, USA, , Article 2*

- H. Sutter: The Free Lunch is Over: A Fundamental Turn Toward Concurrency in Software, *Dr. Dobb's Journal, 30(3), March 2005,* http://www.gotw.ca/publications/concurrency-ddj.htm (Visited May 2011)

- V. W. Lee, C. Kim, J. Chhugani, M. Deisher, D. Kim, A. D. Nguyen, N. Satish, M. Smelyanskiy, S. Chennupaty, P. Hammarlund, R. Singhal, and P. Dubey: Debunking the 100X GPU vs. CPU myth: an evaluation of throughput computing on CPU and GPU. *SIGARCH Comput. Archit. News 38, 3 (June 2010), 451-460.*

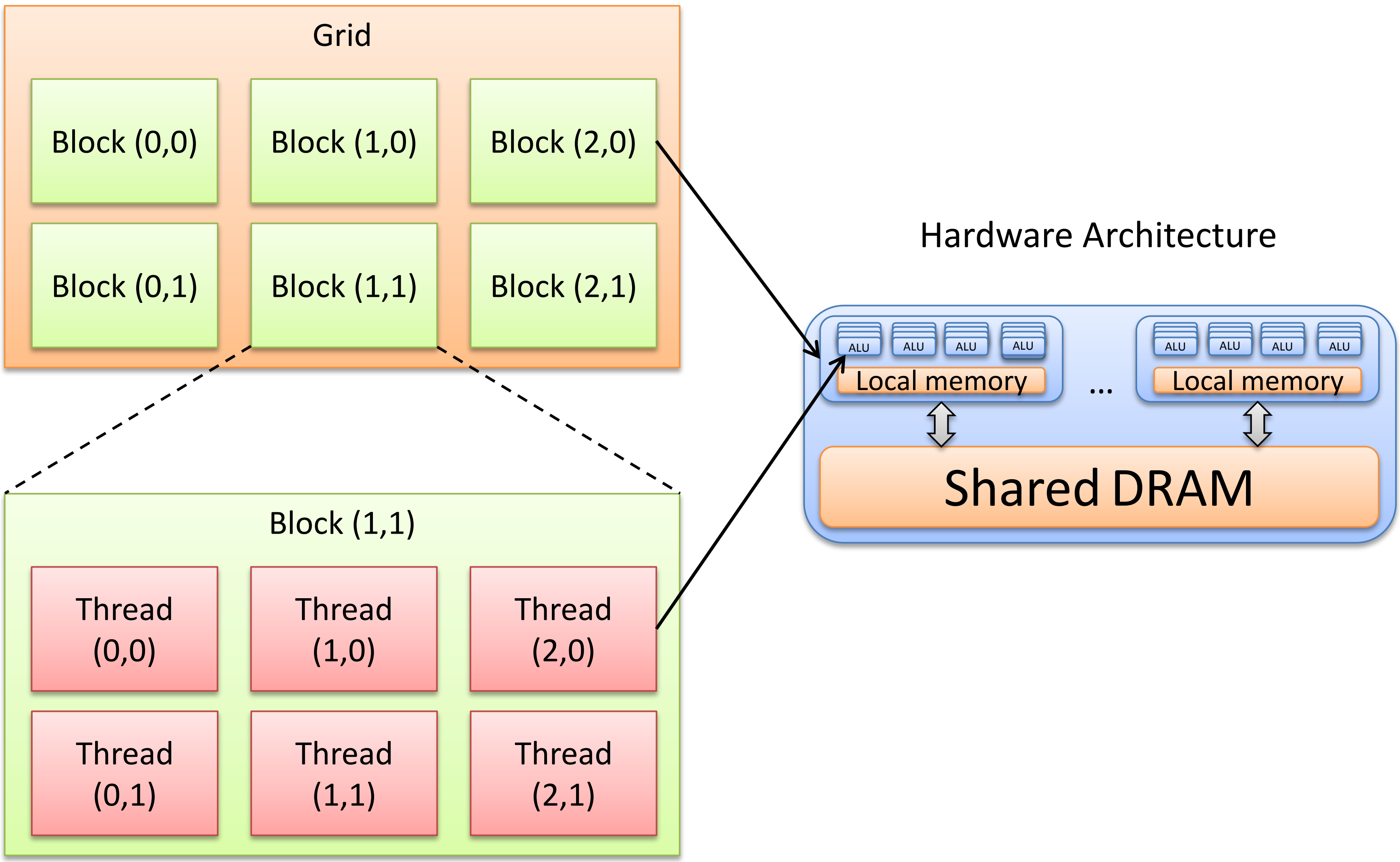
- B. He and J. Xu Yu: High-throughput transaction executions on graphics processors. *Proc. VLDB Endow. 4, 5 (February 2011), 314-325.*

- B. He, M. Lu, K. Yang, R. Fang, N. K. Govindaraju, Q. Luo, and P. V. Sander: Relational query coprocessing on graphics processors. *ACM Trans. Database Syst. 34, 4, Article 21 (December 2009), 39 pages.*

Outline

- Background
  - Parallel Speedup
  - Levels of Parallelism
  - CPU Architecture
- Scale out
  - MapReduce
  - Stratosphere
- Scale up
  - Overview of Hardware Architectures
  - **Parallel Programming Model**
  - Relational Processing
  - Further Operations
  - Research Challenges of Hybrid Architectures

Parallel Programming Model



Programming Model

Grid

Block (0,0) Block (1,0) Block (2,0)

Block (0,1) Block (1,1) Block (2,1)

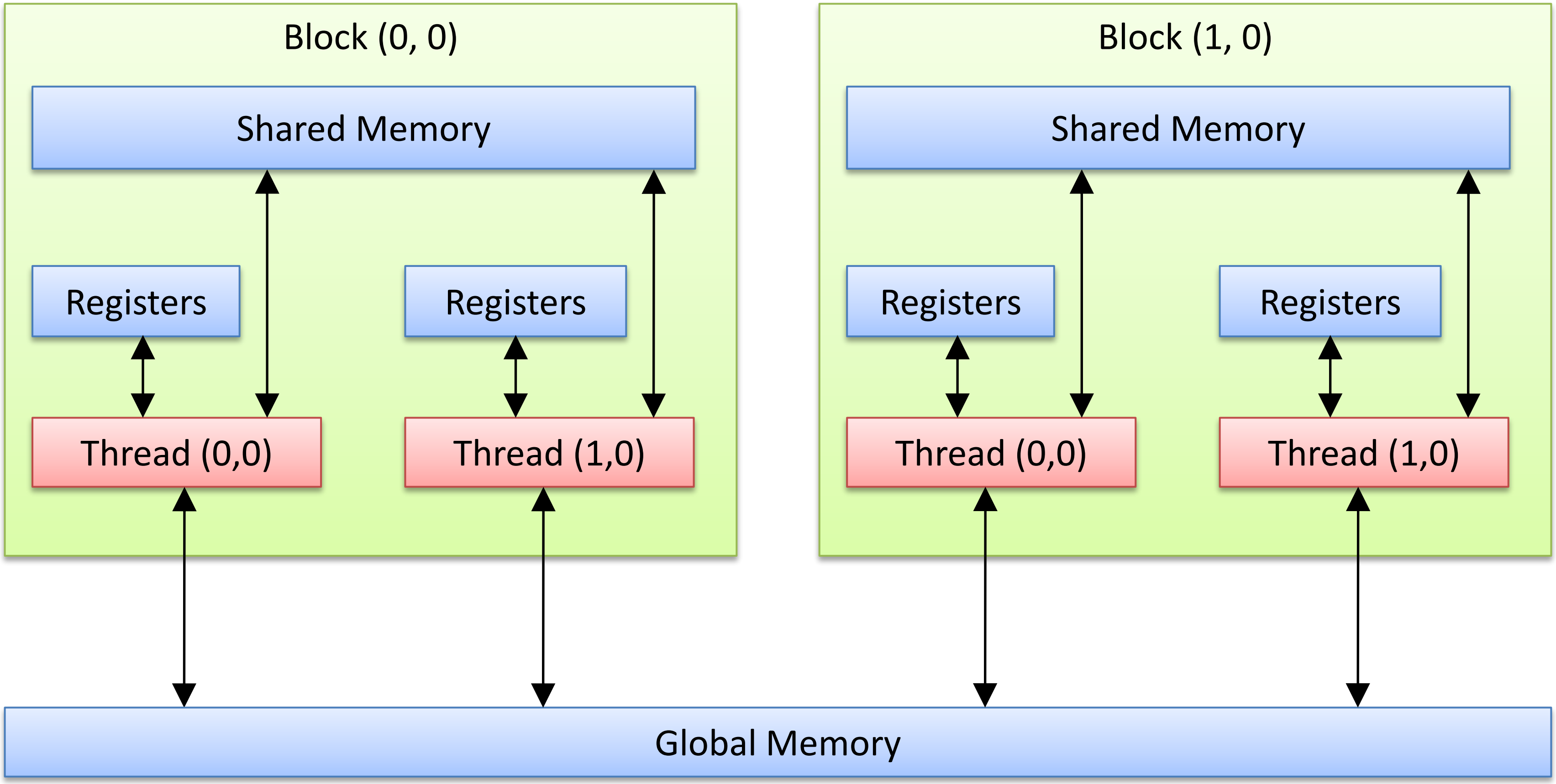
Block (1,1)

Thread (0,0) Thread (1,0) Thread (2,0)

Thread (0,1) Thread (1,1) Thread (2,1)

Hardware Architecture

Memory Hierarchy



Challenges

- No dynamic memory allocation
- Divergence
  - □ Threads taking different code paths
  - → Serialization
- Limited synchronization possibilities
  - □ Only on block level
  - □ Limited block size

Sample CUDA Kernel

- Kernel

```
__global__ void increment(int* values, int elements) {
      // store thread identifier in register
      unsigned int tid = threadIdx.x;
      if (tid < elements) {
            // increment stored value by 1
            int value = values[tid];
            values[tid] = value + 1;
      }
}
```

- Host Code

```
increment <<< nrBlocks, blockSize >>>(values, 200000);
```

References & Further Reading

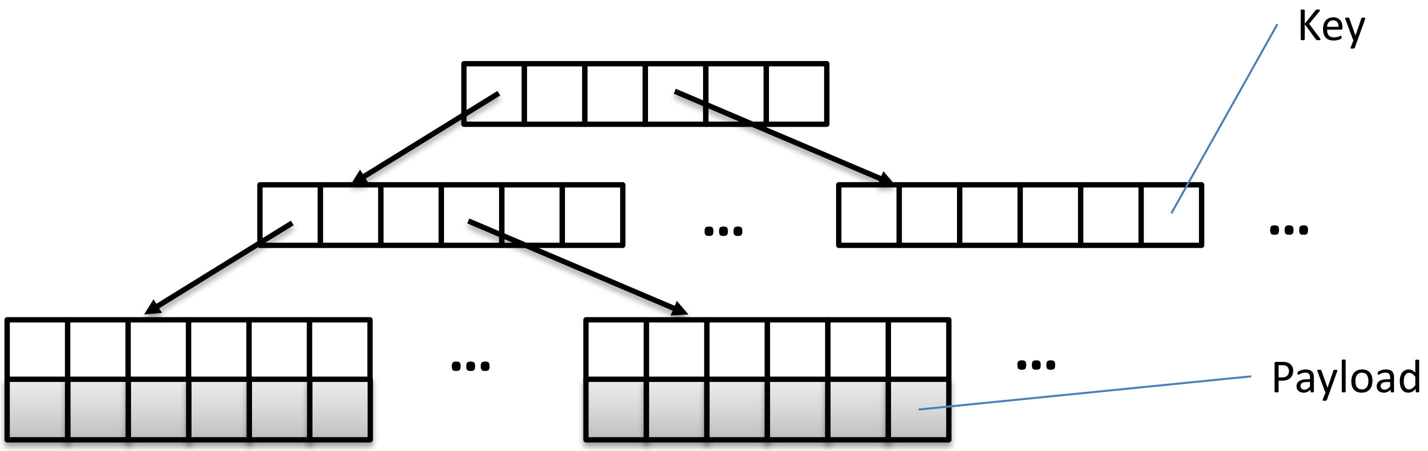
- NVIDIA CUDA Programming Guide Version 3.1.1, http://developer.download.nvidia.com/compute/cuda/3_1/toolkit/docs/NVIDIA_CUDA_C_ProgrammingGuide_3.1.pdf (Visited June 2011)

Outline

- Background
  - Parallel Speedup
  - Levels of Parallelism
  - CPU Architecture
- Scale out
  - MapReduce
  - Stratosphere
- Scale up
  - Overview of Hardware Architectures
  - Parallel Programming Model
  - **Relational Processing**
  - Further Operations
  - Research Challenges of Hybrid Architectures

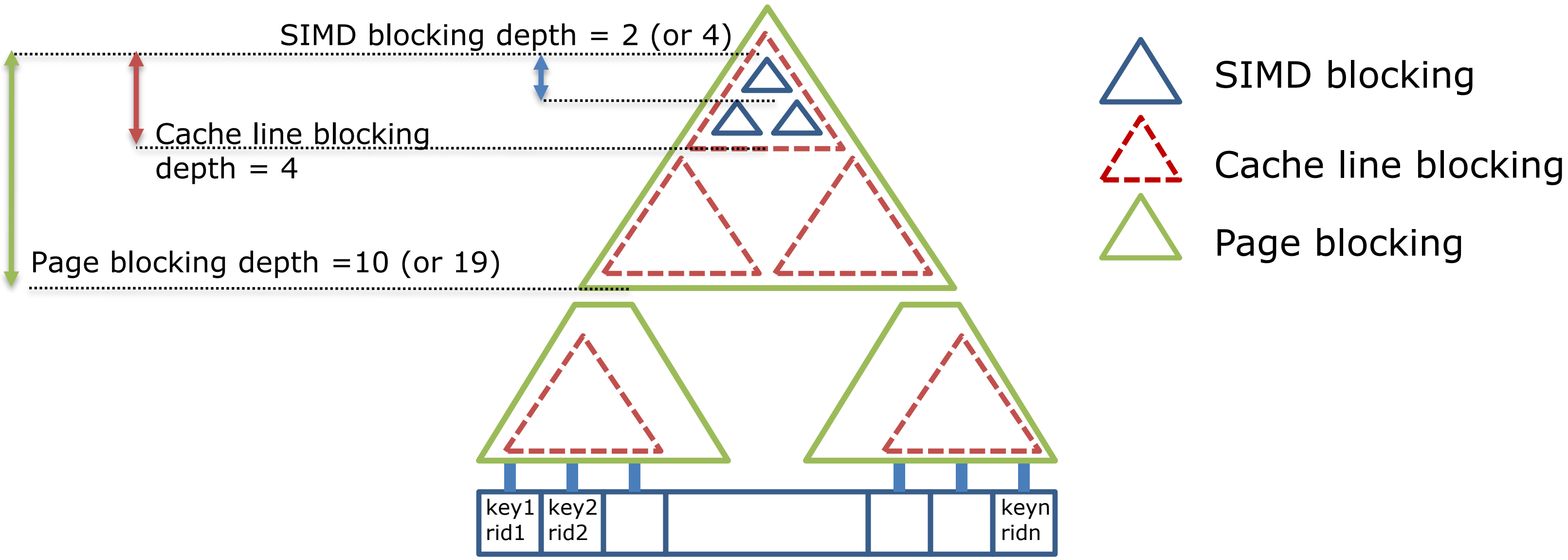
Indexing

- Classic: B+ Tree



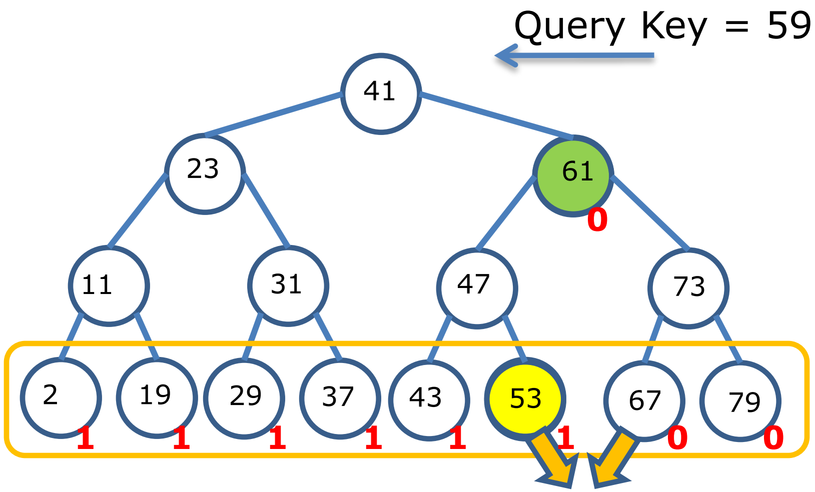
- Accelerate access by organizing data in a sorted and hierarchical way
- Compression to map variable-length data to fixed size

Hierarchical Blocking


SIMD blocking depth = 2 (or 4)
Cache line blocking depth = 4
Page blocking depth =10 (or 19)
key1 rid1
key2 rid2
keyn ridn
SIMD blocking
Cache line blocking
Page blocking

- Rearrange the binary tree node and block in a hierarchical fashion
  - Page level, cache line level, SIMD level
  - Minimize # of pages, cache lines brought from memory
  - Increased computation overhead when crossing a blocking boundary

GPU Search



1. Load 16 tree nodes within a SIMD blocked sub-tree

2. Compare with a query key (query > tree nodes) and store results (“*res*”) into shared memory buffer

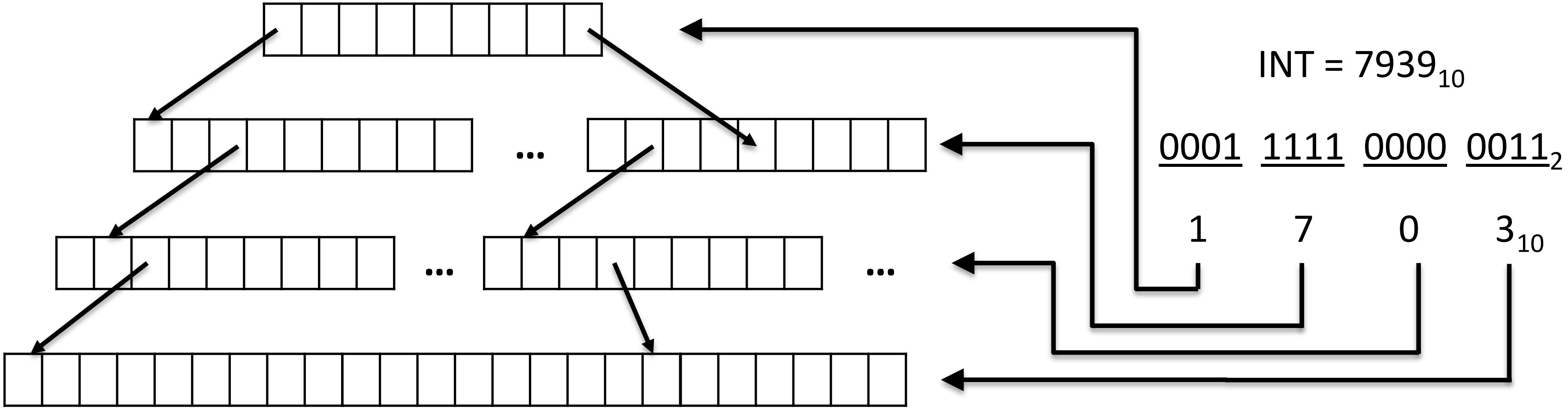
3. Among 8 leaf nodes, find a largest index (j) such that “res_j =1” and “res_{j+1} =0

4. Find a common ancestor node and check the result in the shared memory buffer

5. Compute the address of the next sub-tree

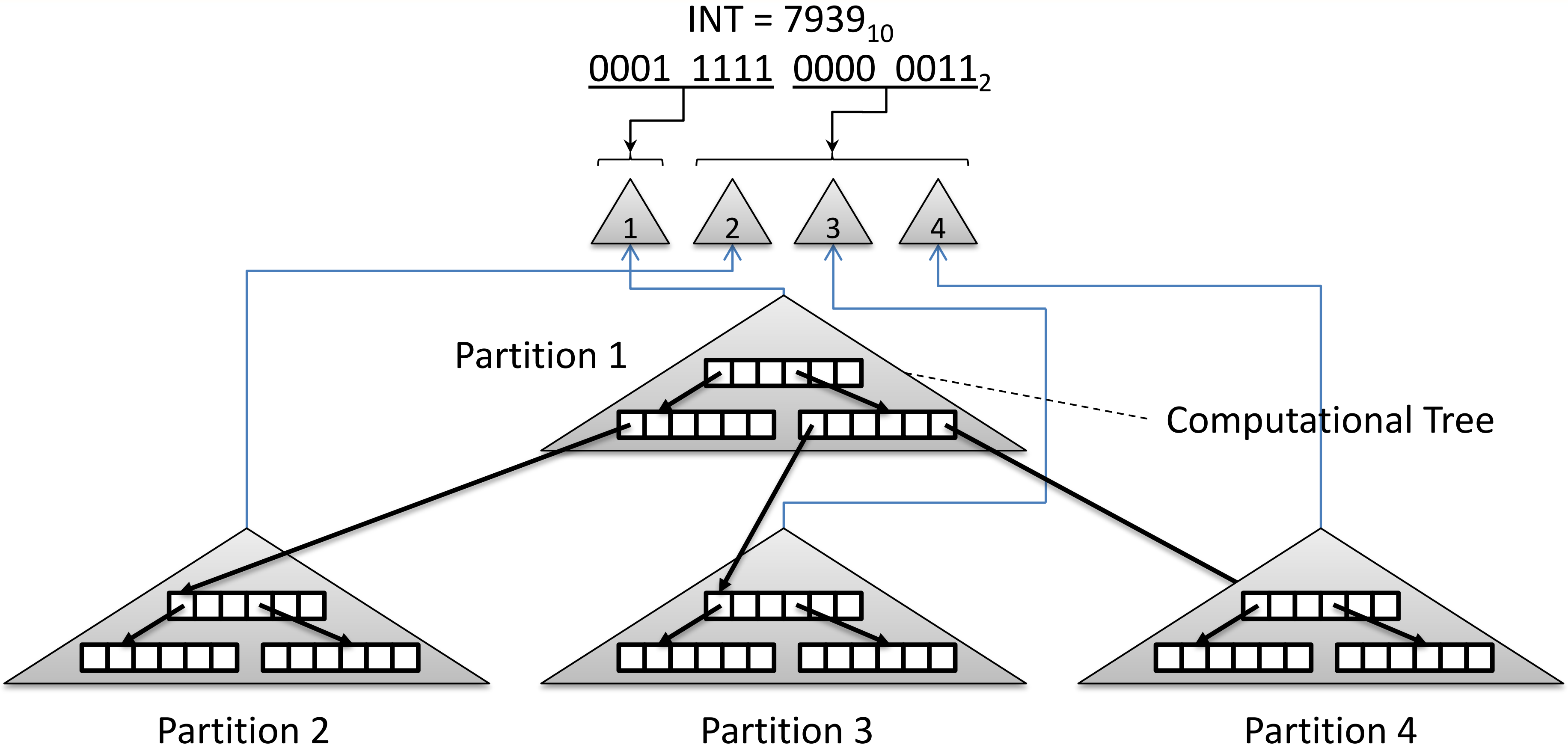
=> more instruction overhead due to the lack of inter-SIMD lane operations

Prefix Tree


INT = 7939_{10}
$\underline{0001}\ \underline{1111}\ \underline{0000}\ \underline{0011}_2$
1 7 0 3_{10}
…
…
…

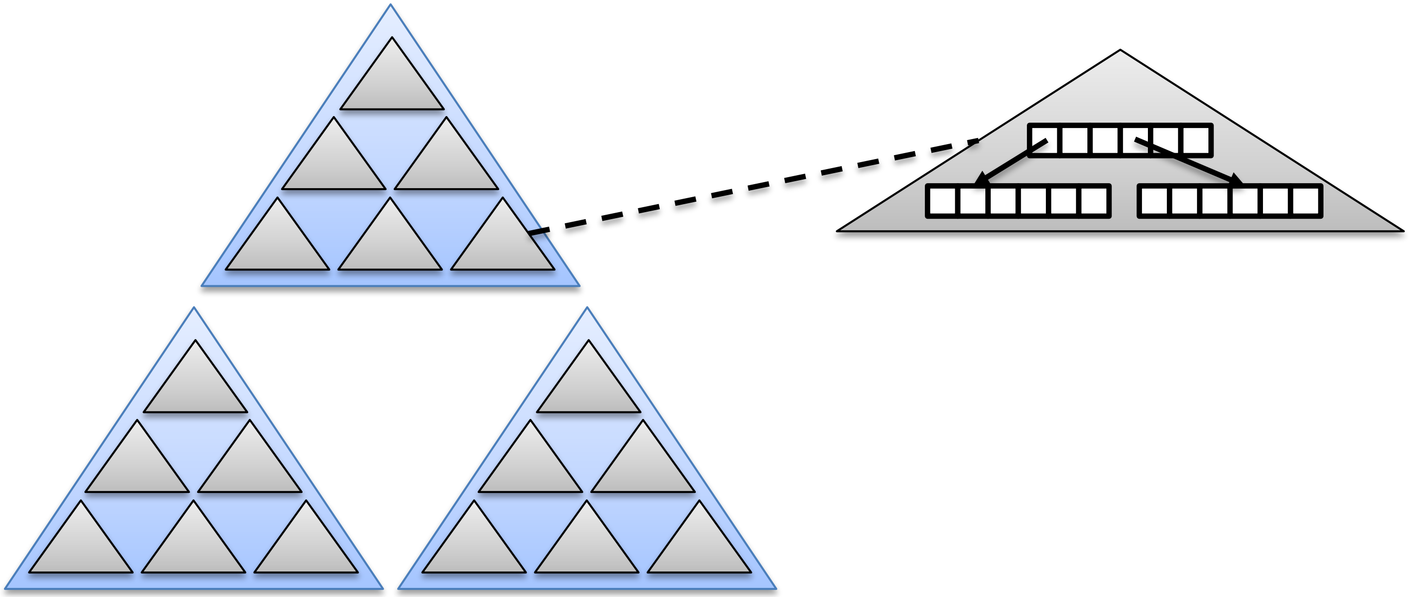
- Path of a key within the tree defined by absolute value
- Split key into equally sized sub-keys
- Tree depth depends on key length

Speculative Partition-Based Traversal



- Each computational tree is assigned to a GPU thread
- All computational trees are processed in parallel
- Intermediate results are stored in global list to retrieve final result later

Speculative Hierarchical Traversal



- Idea: Group processing to exploit faster local memory
- Group computational trees into blocks
- Tree block is executed by a GPU thread block
  - □ Allows usage of shared memory for intermediate results
- Result for global retrieved by traversing internal result list

References & Further Reading

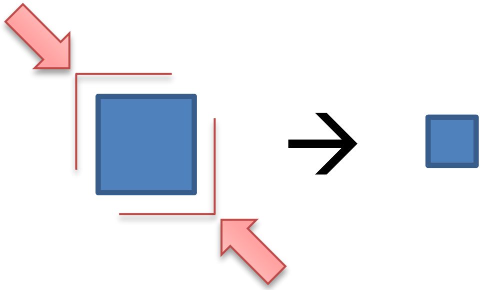
- J. Rao, K. A. Ross: Cache Conscious Indexing for Decision-Support in Main Memory, *In Proceedings of the 25th International Conference on Very Large Data Bases (VLDB '99), Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie (Eds.). Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 78-89*

- C. Kim, J. Chhugani, N. Satish, E. Sedlar, A. D. Nguyen, T. Kaldewey, V. W. Lee, S. Brandt, P. Dubey: FAST: fast architecture sensitive tree search on modern CPUs and GPUs, *ACM SIGMOD '10, 339-350, 2010*

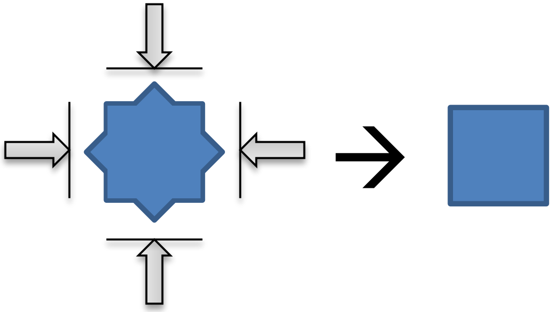
- P. B. Volk, D. Habich, W. Lehner: GPU-Based Speculative Query Processing for Database Operations, *First International Workshop on Accelerating Data Management Systems Using Modern Processor and Storage Architectures In conjunction with VLDB 2010*

Database Compression

- Disk-based databases
  - Computation-intensive compression algorithms
  - Trade computation for compression ratio
  - Often full decompression required for evaluations



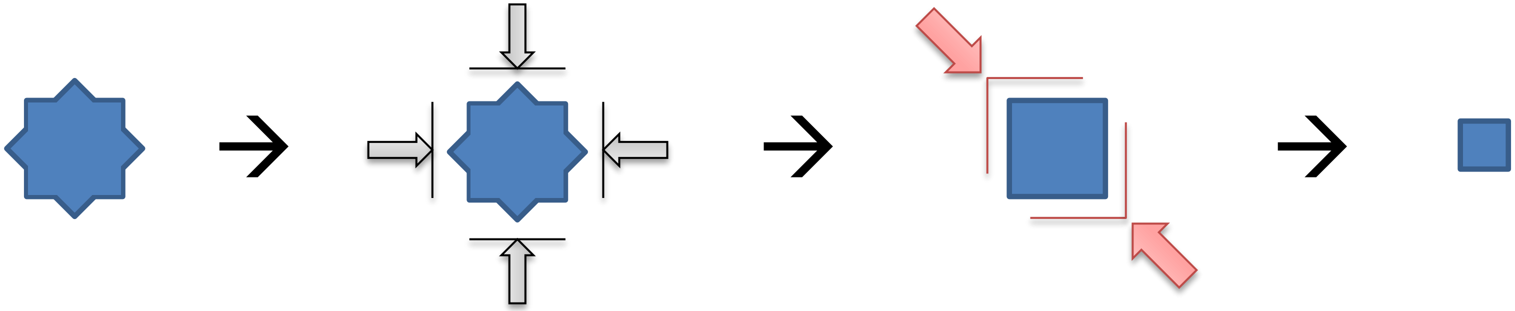
- In-memory databases
  - Lightweight compression schemes
  - Balance compression and computational overhead
  - No decompression for most evaluations
  - Combination of multiple schemes undesirable
    → Computational overhead



Database Compression

- GPUs
  - Device has very limited memory capacity
  - Offer high computation capabilities and bandwidth
  - Compression reduces overhead of data transfer between device and main memory
  - Compression allows GPUs to address bigger problems

→ Combine compression schemes for higher compression ratios



- Two Groups of compression schemes
  - □ Main Schemes 🟥
    - Reduce data size
  - □ Auxiliary Schemes ⬜
    - Transform data into formats suitable for main schemes
- Cascaded Compression
  - □ Huge search space
  - □ Dependent on data properties
  - □ Different criteria
    - Compression ratio
    - (De)Compression performance
    - Combination of multiple factors

Cascaded Compression Example

- Null Suppression with Variable Length (NSV)
  - □ Main Scheme
  - □ Eliminate leading zeros → variable length encoding
  - □ Store encoded value and length of value

- Delta
  - □ Auxiliary Scheme
  - □ Encodes difference from the preceding value
  - □ Small differences for sorted columns

Column (Int)	Σ Size (Bits)	Delta	NSV Length	NSV Value	Σ Size (Bits)
1000	32	0	0x01	0x0	3
1001	64	1	0x01	0x1	6
1002	96	1	0x01	0x1	9
1004	128	2	0x10	0x10	13
1005	160	1	0x01	0x1	16
1009	192	4	0x11	0x100	21

References & Further Reading

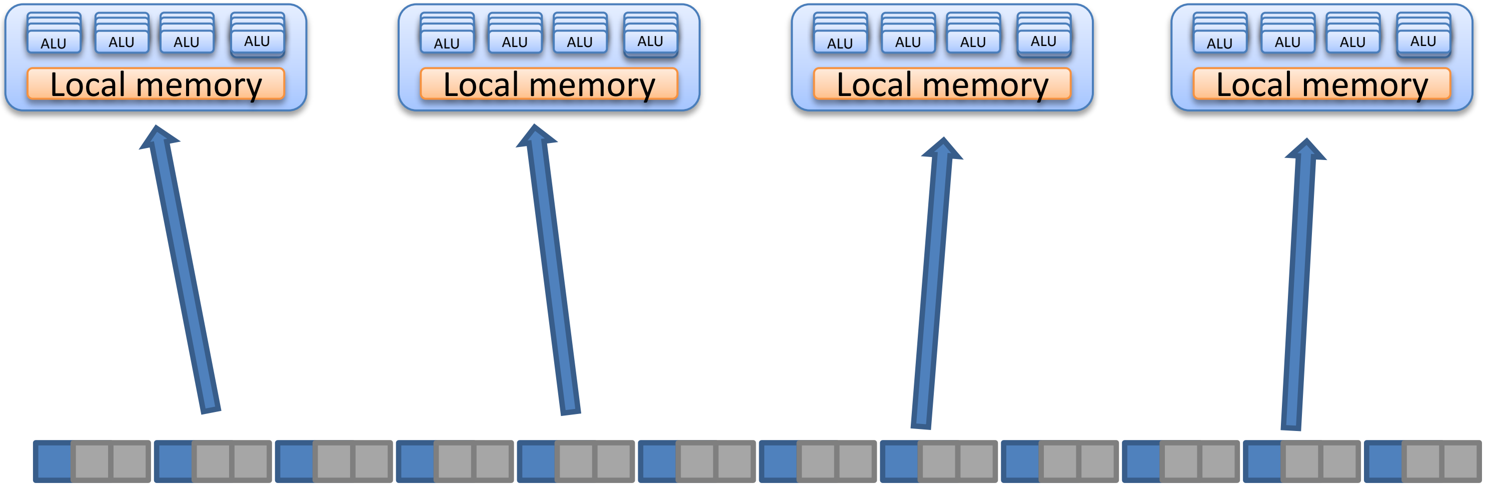
- W. Fang, B. He, Q. Luo: Database Compression on Graphics Processors, *Proc. VLDB Endow. 3, 1-2 (September 2010), 670-680*

Sorting with GPUs

- Numerous GPU algorithms published
  - Bitonic sort, bucket sort, merge sort, quicksort, radix sort, sample sort..
- Existing work: focus on in-memory sorting with 1 GPU
- State of the art: merge sort, radix sort
- GPUs outperform multicore CPUs when data transfer over PCIe is not considered

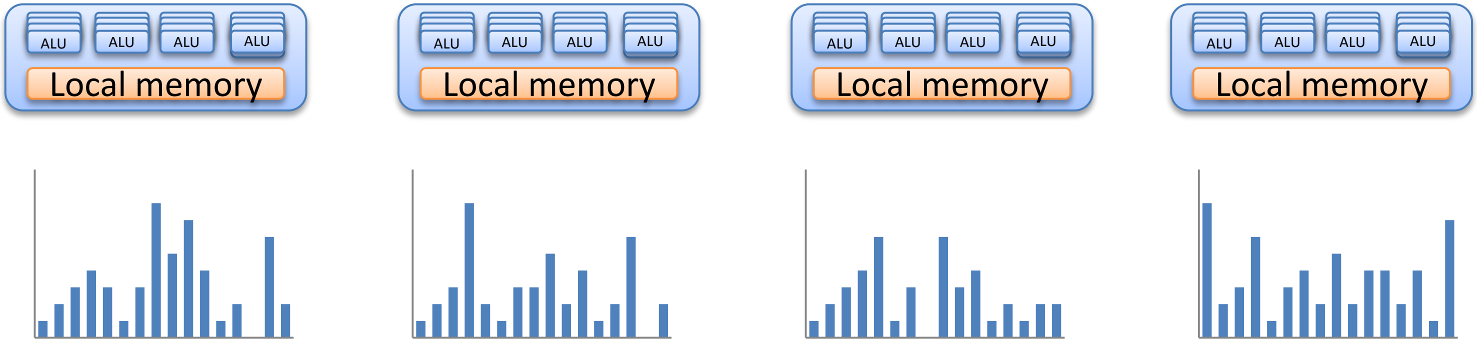
Radix Sort

- Divide data between multiprocessors



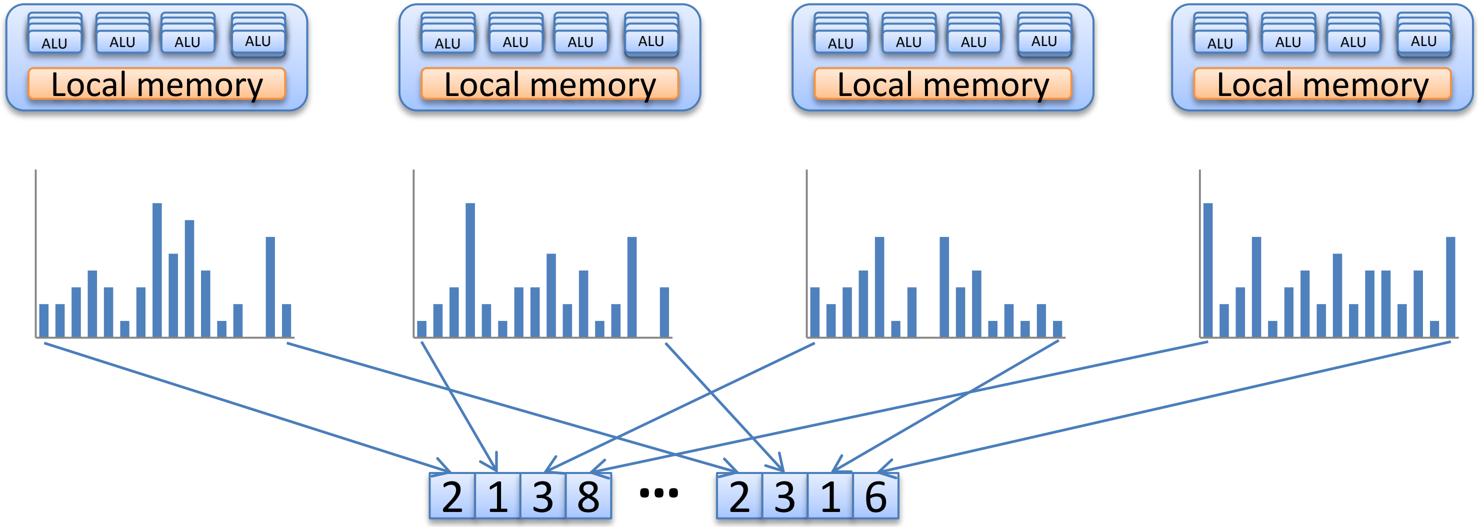
Radix Sort

- Divide data between multiprocessors
- Create a local histogram



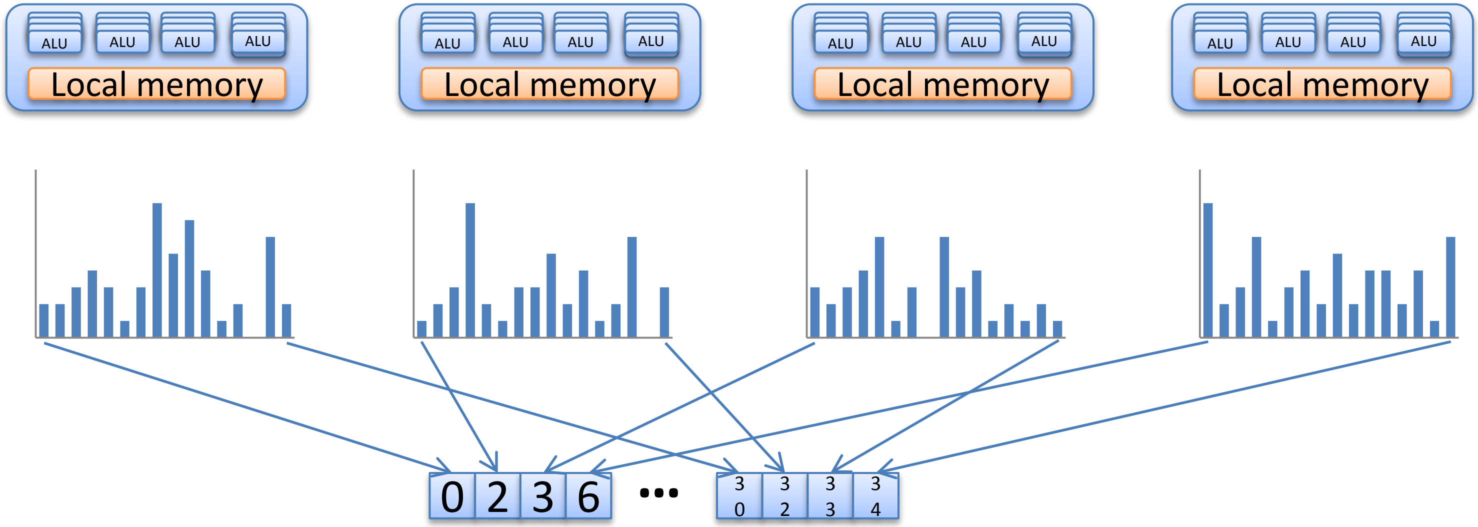
Radix Sort

- Divide data between multiprocessors
- Create a local histogram
- Store histrograms in array sorted by radix values



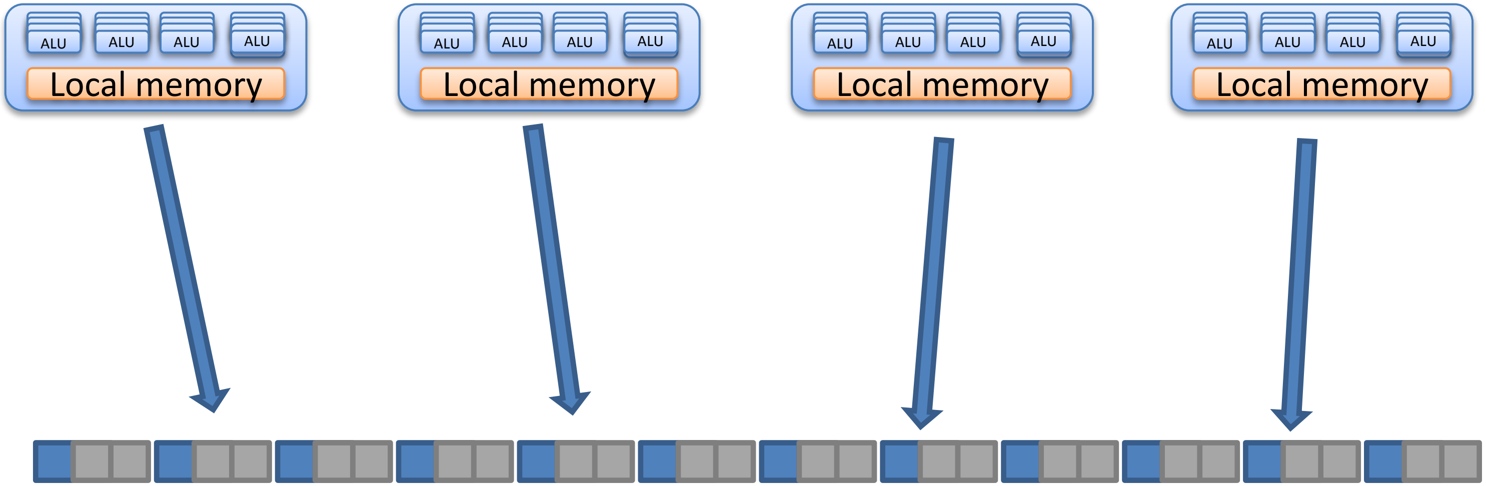
Radix Sort

- Divide data between multiprocessors
- Create a local histogram
- Store histrograms in array sorted by radix values
- Compute prefix sum -> global write position



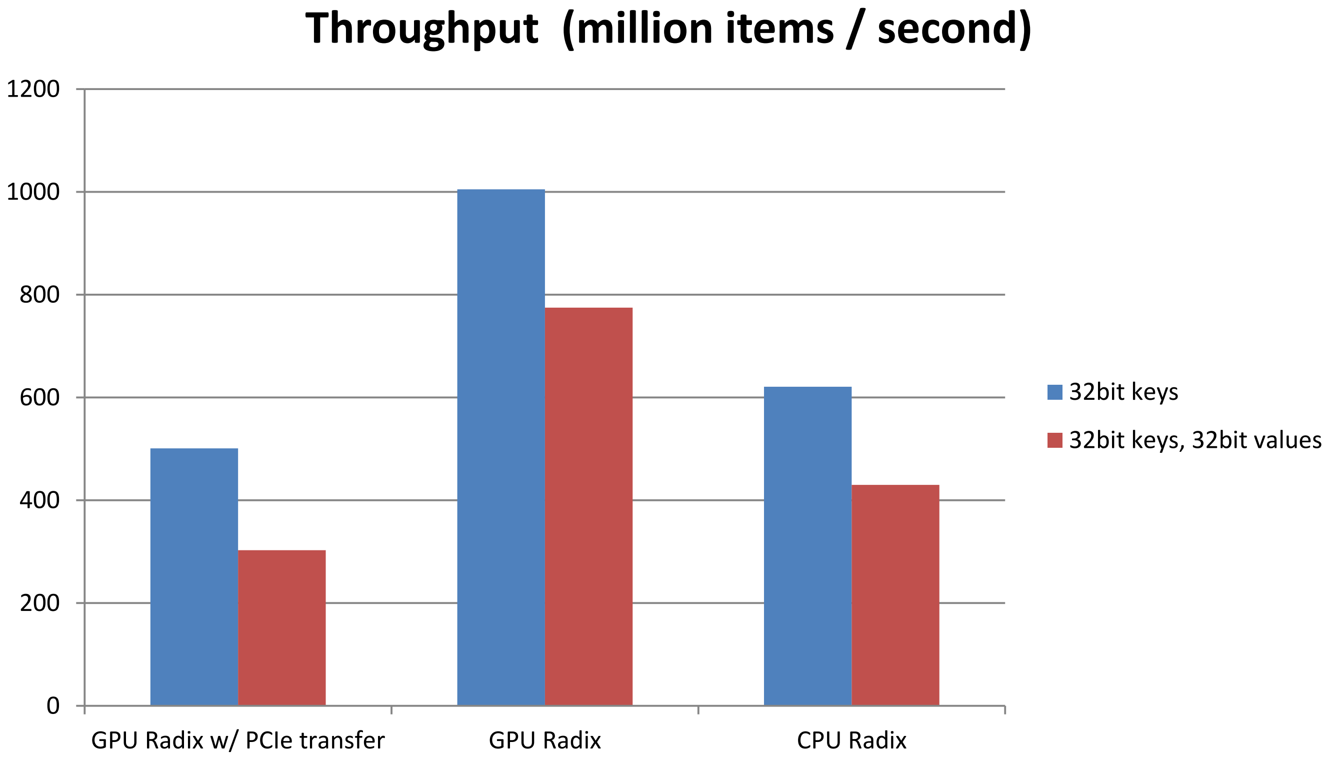
Radix Sort

- Divide data between multiprocessors
- Create a local histogram
- Store histrograms in array sorted by radix values
- Compute prefix sum -> global write position
- Scatter data to new positions



GPU sorting performance

- CPU: 2x Intel W5580, GPU: NVIDIA Geforce GTX480
- For GPU radix sort the PCIe transfer dominates running time



Revisiting Sorting for GPGPU Stream Architectures
Duane Merrill, Andrew Grimshaw

Faster Radix Sort via Virtual Memory and Write-Combining
Jan Wasenberg, Peter Sanders

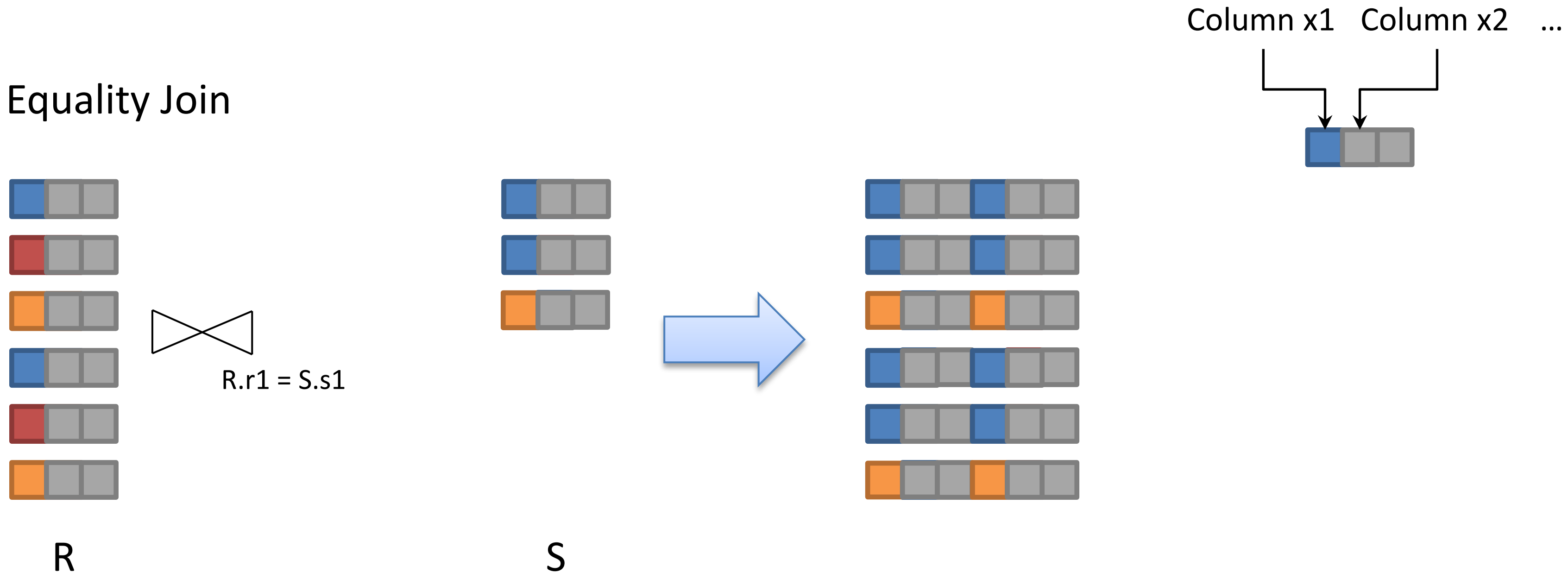
References & Further Reading

- D. G. Merrill and A. S. Grimshaw: Revisiting sorting for GPGPU stream architectures. *In Proceedings of the 19th international conference on Parallel architectures and compilation techniques (PACT '10). ACM, New York, NY, USA, 545-546*

- J. Wassenberg, P. Sanders: Faster Radix Sort via Virtual Memory and Write-Combining, CoRR 2010, http://arxiv.org/abs/1008.2849 (Visited May 2011)

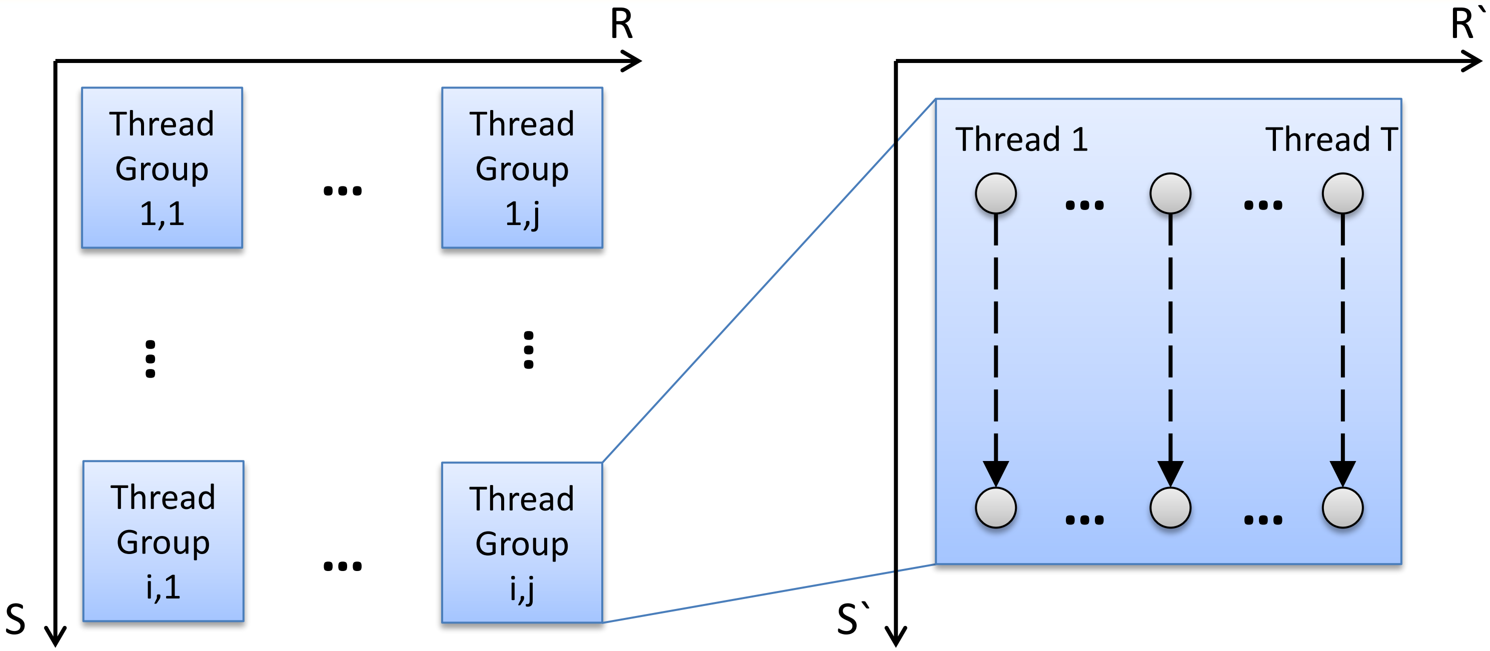
- N. Satish, C. Kim, J. Chhugani, A. D. Nguyen, V. W. Lee, D. Kim, and P. Dubey: Fast sort on CPUs and GPUs: a case for bandwidth oblivious SIMD sort. *In Proceedings of the 2010 international conference on Management of data (SIGMOD '10). ACM, New York, NY, USA, 351-362.*

Joins



- Cross product between two sets R & S
- May apply a filter predicate to each pair
- Result set size hard to estimate

Non-indexed Nested-Loop Join

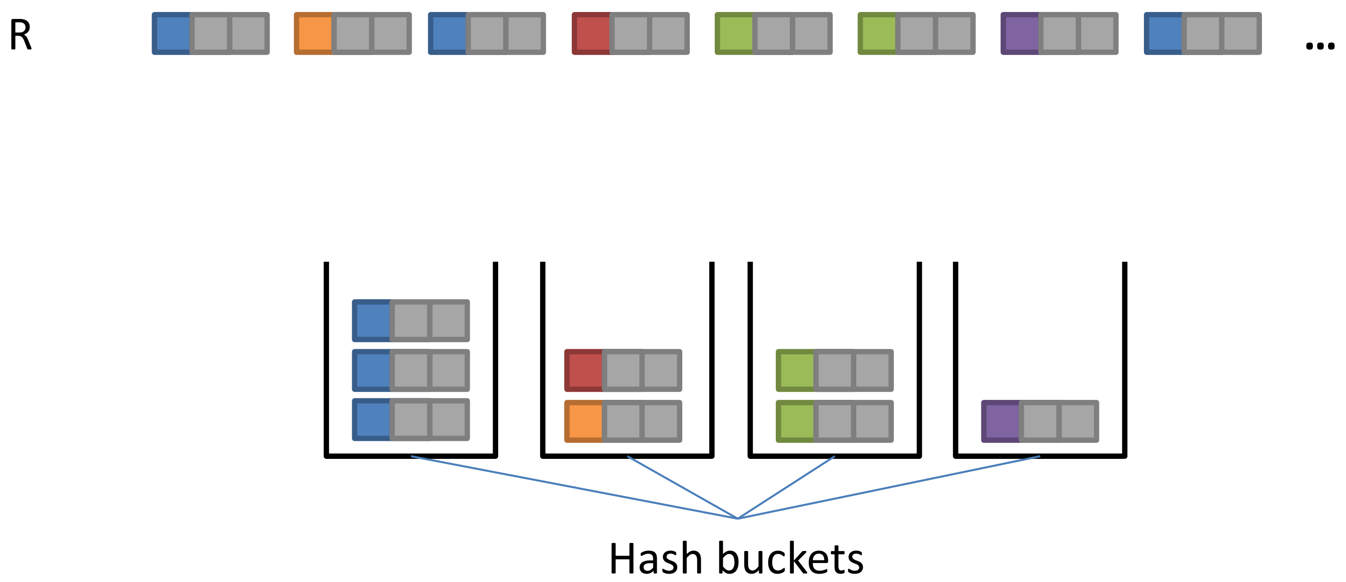


1. Split relations into blocks

2. Join smaller blocks in parallel

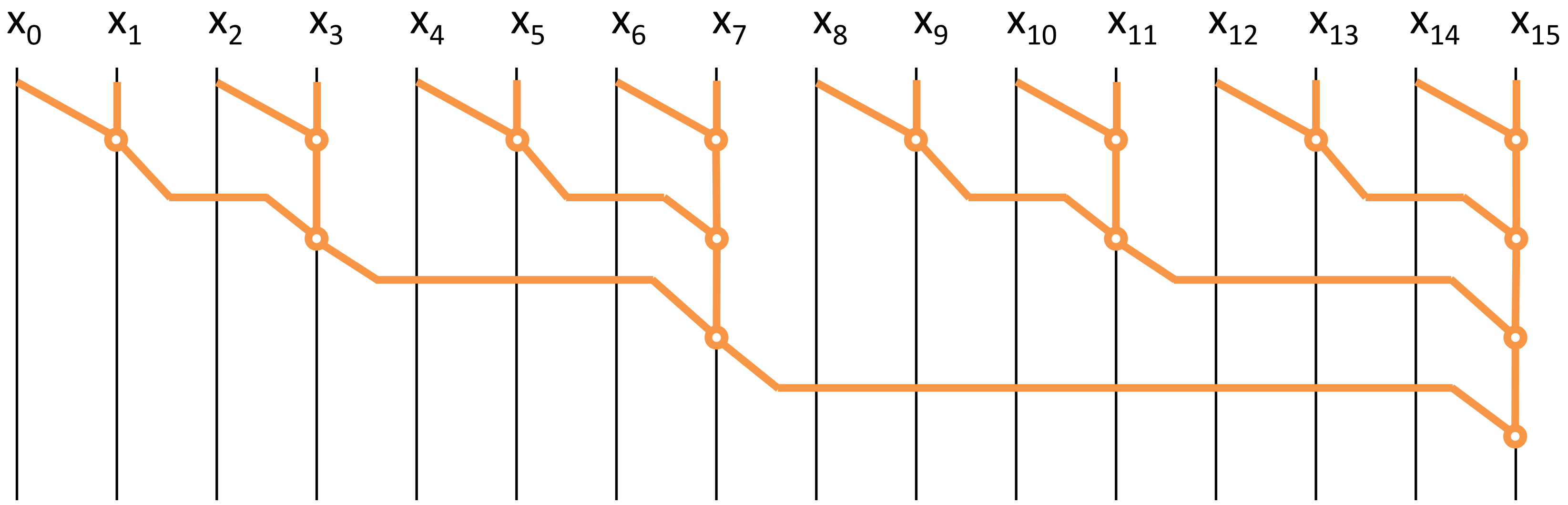
Problem: Lack of dynamic memory allocation!
To allocate memory for the result, the join has to be performed twice.

Hash Join



1. Partition R & S using a hash function (radix partitioning)
2. Combine only buckets of the same hash value
   - Bucket pair handled by a thread group (like NINLJ)

Aggregation



- Brent-Kung circuit strategy
- Only upsweep phase necessary because only final result is needed
- Permutation of elements to minimize memory bank conflicts
- Separate thread group to combine results of blocks

References & Further Reading

- B. He, K. Yang, R. Fang, M. Lu, N. Govindaraju, Q. Luo, P. Sander: Relational joins on graphics processors. *In Proceedings of the 2008 ACM SIGMOD international conference on Management of data (SIGMOD '08). ACM, New York, NY, USA, 511-524.*

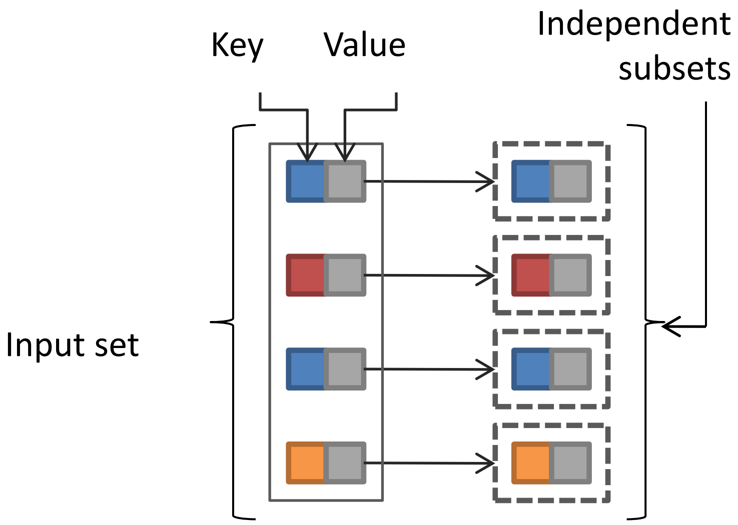
- D. Merrill, A. Grimshaw: Parallel Scan for Stream Architectures. *Technical Report CS2009-14, Department of Computer Science, University of Virginia. December 2009.*

Outline

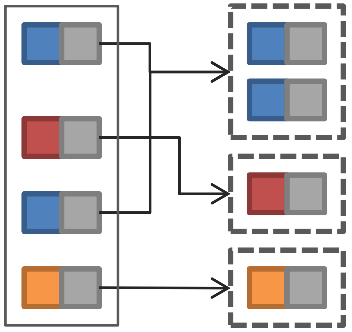
- Background
  - Parallel Speedup
  - Levels of Parallelism
  - CPU Architecture
- Scale out
  - MapReduce
  - Stratosphere
- Scale up
  - Overview of Hardware Architectures
  - Parallel Programming Model
  - Relational Processing
  - **Further Operations**
  - Research Challenges of Hybrid Architectures

Programming Model

- Map and reduce are second-order functions
  - Call first-order functions (user code)
  - Provide first-order functions with subsets of the input data

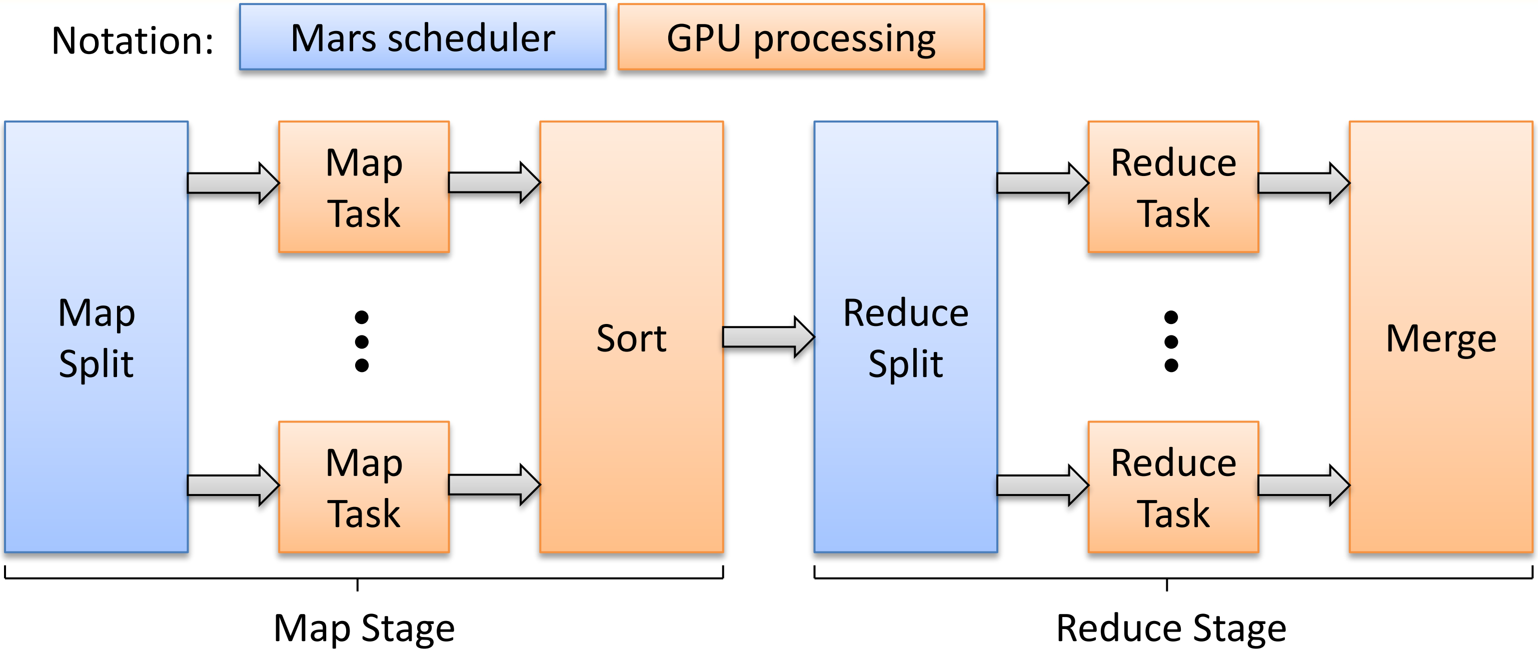


- Map
  - All records are independently processable



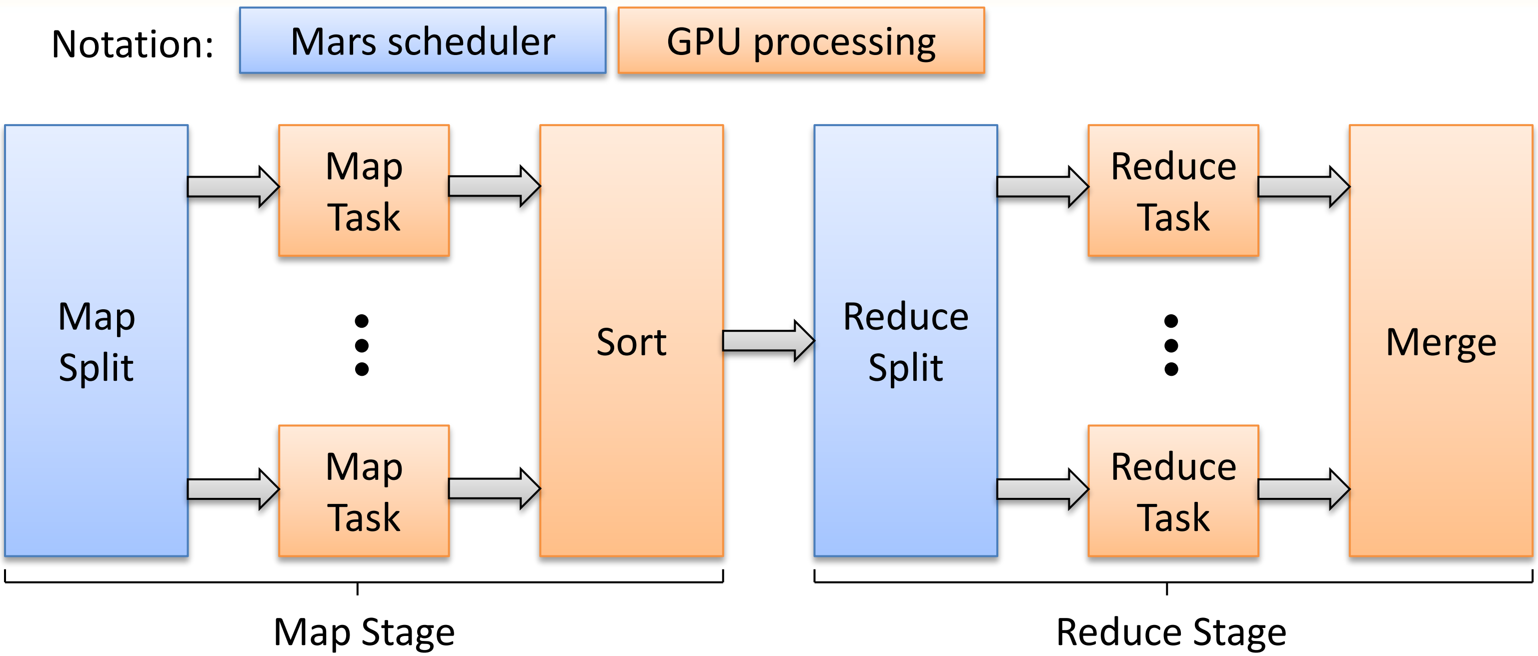
- Reduce
  - Records with identical key must be processed together

Mars: A MapReduce Framework on Graphics Processors



- Scheduler
  - □ Prepares data input
  - □ Invokes map & reduce stages on the GPU
  - □ Returns results to the user
- 2-step output scheme for GPU processing
  - □ Process to retrieve result size
  - □ Process and output results

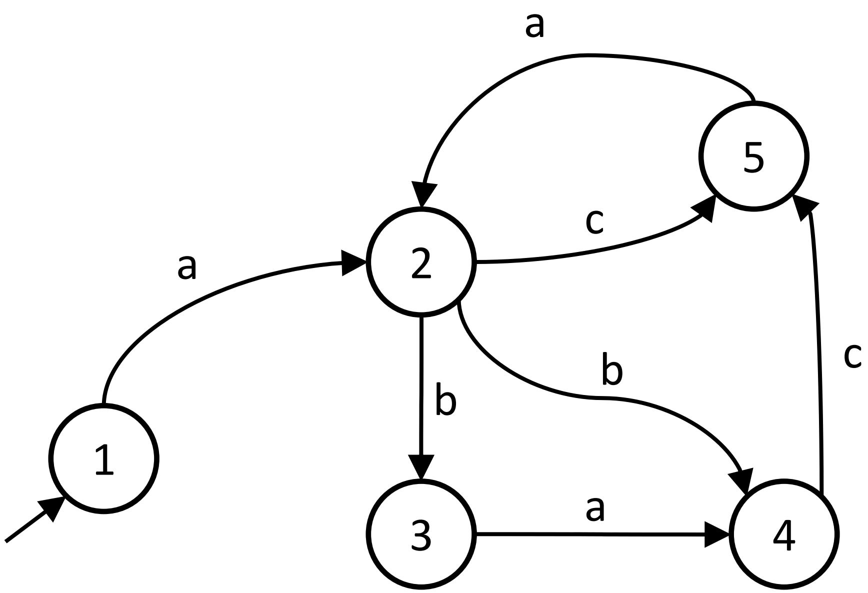
Mars: A MapReduce Framework on Graphics Processors



Conclusion

- Abstracts from GPU architecture
- Doubles computation of map/reduce in the worst case
- Lock and write conflict-free parallel execution
- Combination of scale out and scale up

Regular Expression Matching



- Non-deterministic finite state automatons
- Exploit parallelism
  - □ Analyze multiple packets (one thread group per packet)
  - □ Each thread analyzes a transition

There is more…

- K-Means
- Apriori
- Exact String matching
- Support Vector Machines
- …

References & Further Reading

- N. Cascarano, P. Rolando, F. Risso, R. Sisto: iNFAnt: NFA Pattern Matching on GPGPU Devices. *SIGCOMM Comput. Commun. Rev. 40, 5 20-26.*

- M.C. Schatz, C. Trapnell: Fast Exact String Matching on the GPU, *Technical Report*.

- W. Fang, K. K. Lau, M. Lu, X. Xiao, C. K. Lam, P. Y. Yang, B. He, Q. Luo, P. V. Sander, K. Yang: Parallel Data Mining on Graphics Processors, *Technical Report HKUST-CS08-07, Oct 2008.*

- S. Herrero-Lopez, J. R. Williams, A. Sanchez: Parallel multiclass classification using SVMs on GPUs. *Proceedings of the 3rd Workshop on General-Purpose Computation on Graphics Processing Units (GPGPU '10), Pages 2-11, ACM New York, 2010*

- B. C. Catanzaro, N. Sundaram, K. Keutzer: Fast Support Vector Machine Training and Classification on Graphics Processors. *Proceedings of the 25th international conference on Machine learning (ICML '08), ACM New York, 2008*

- B. He, W. Fang, Q. Luo, N. K. Govindaraju, and T. Wang. 2008. Mars: a MapReduce framework on graphics processors. In *Proceedings of the 17th international conference on Parallel architectures and compilation techniques* (PACT '08). ACM, New York, NY, USA, 260-269.

Outline

- Background
  - Parallel Speedup
  - Levels of Parallelism
  - CPU Architecture
- Scale out
  - MapReduce
  - Stratosphere
- Scale up
  - Overview of Hardware Architectures
  - Parallel Programming Model
  - Relational Processing
  - Further Operations
  - **Research Challenges of Hybrid Architectures**

Architectural constraints

- PCIe bottleneck
  - Direct access to storage devices, network etc.?
  - Caching strategies for device memory
- GPU memory size
  - Deeper memory hierarchy (e.g. a few GB of fast GDDR + large „slow" DRAM)?

Performance portability

- Want forward scalability: performance should scale with next generations of GPUs
  - Existing work often optimized for exactly 1 type of GPU chip
- Need higher level programming models
  - Hide hardware details (processor count, SIMD width, local memory size..)

Database-Specific Challenges

- Database operations are a combination of
  - single threaded compute-intensive operations
  - massively parallel data-intensive operations
- Big data volume and limited memory
- Execution Plan consists of multiple operators
  - Latency!
- Where to execute each operator?
  - Trade off transfer time between CPU and GPU and computational advantage
  - Cost Models for GPGPU, CPU and hybrid
  - Amdahl’s law
- Shared Nothing CPU/GPGPU clusters for scale out

References & Further Reading

- S. Hong and H. Kim: An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness*. SIGARCH Comput. Archit. News 37, 3 (June 2009), 152-163.*

- L. Bic and R. L. Hartmann: AGM: a dataflow database machine. *ACM Trans. Database Syst. 14, 1 (March 1989), 114-146.*

- H. Chafi, A. K. Sujeeth, K. J. Brown, H. Lee, A. R. Atreya, and K. Olukotun: A domain-specific approach to heterogeneous parallelism. *In Proceedings of the 16th ACM symposium on Principles and practice of parallel programming (PPoPP '11). ACM, New York, NY, USA, 35-46.*

Conclusion

- Scale out
  - distribute data & processing among many machines
  - requires a fault-tolerant system
- Scale up
  - use big machines to handle the workload
  - co-processors may accelerate execution
- Ideally: combination of scale up and scale out
  - break problem into computable chunks
  - accelerate processing of chunks
- Added complexity should be hidden by programmers
  - Abstract programming model
  - Optimized execution plans
- Future processor architectures will require a parallel programming approach

धन्यवाद
Hindi

多謝
Traditional Chinese

ขอบคุณ
Thai

Спасибо
Russian

Gracias
Spanish

شكرا
Arabic

Thank You
English

Obrigado
Brazilian Portuguese

Danke
German

Grazie
Italian

多谢
Simplified Chinese

Merci
French

நன்றி
Tamil

ありがとうございました
Japanese

감사합니다
Korean