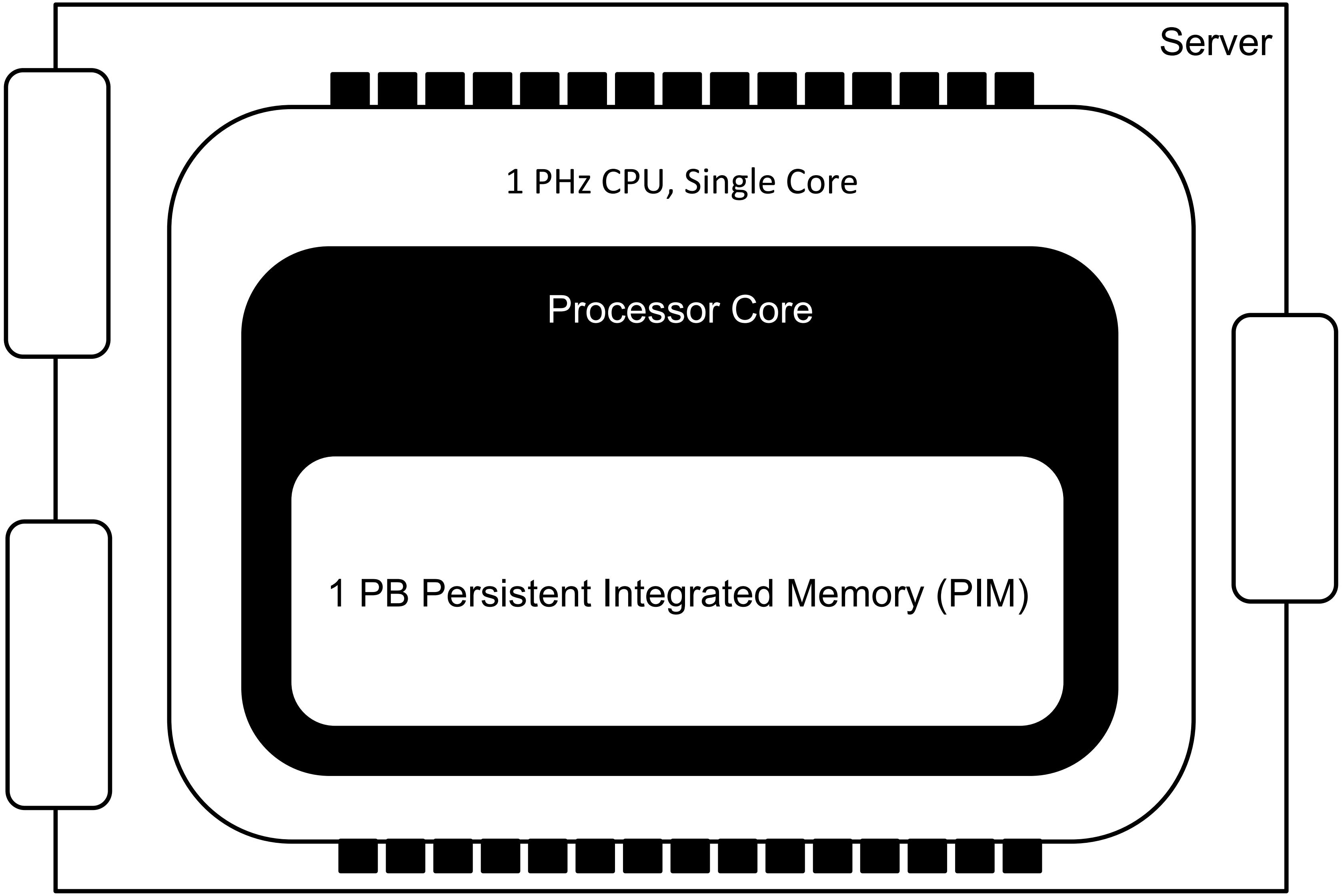
Parallelization

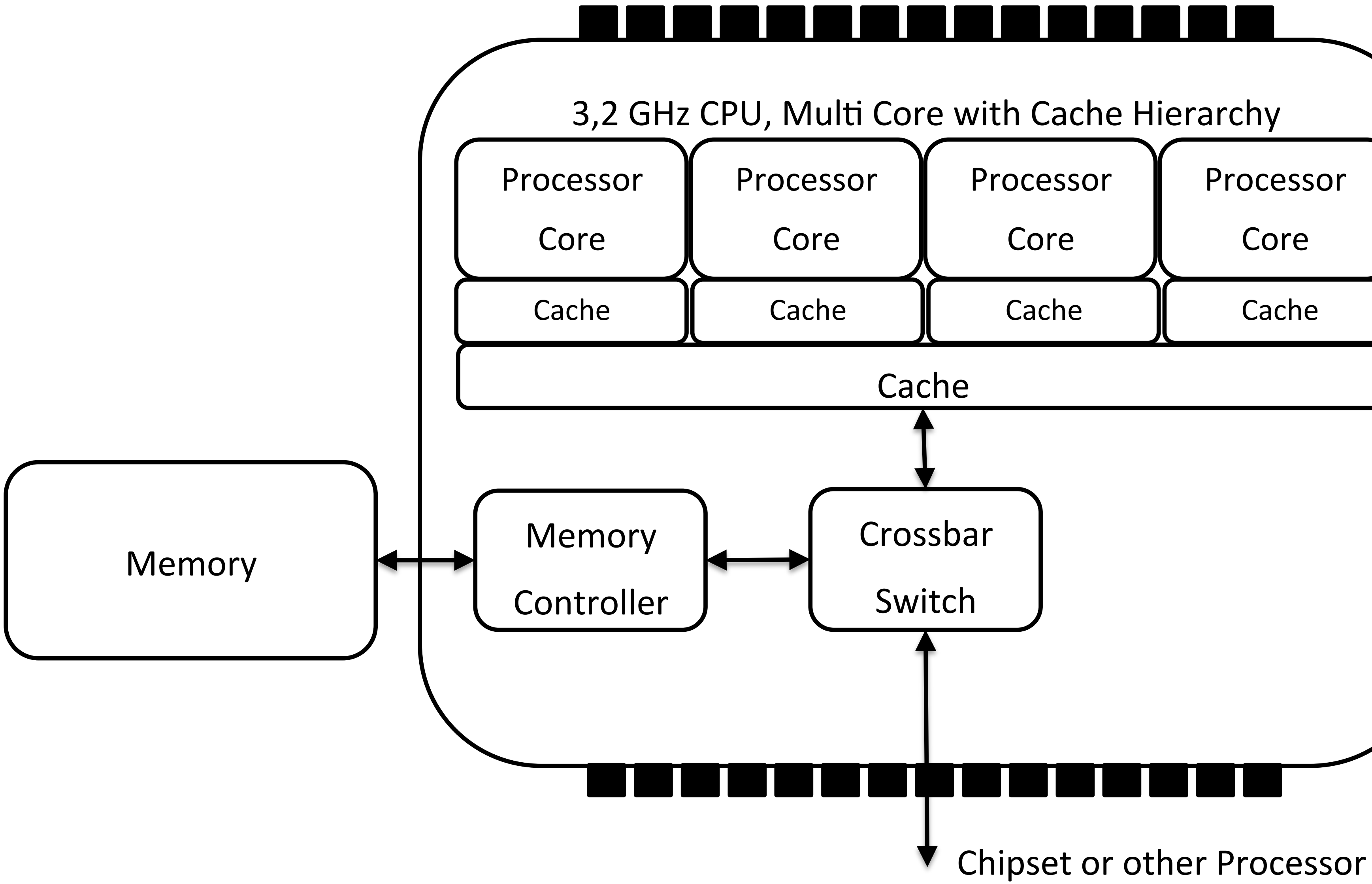
Motivation: Why Parallelization?

Parallel Hardware

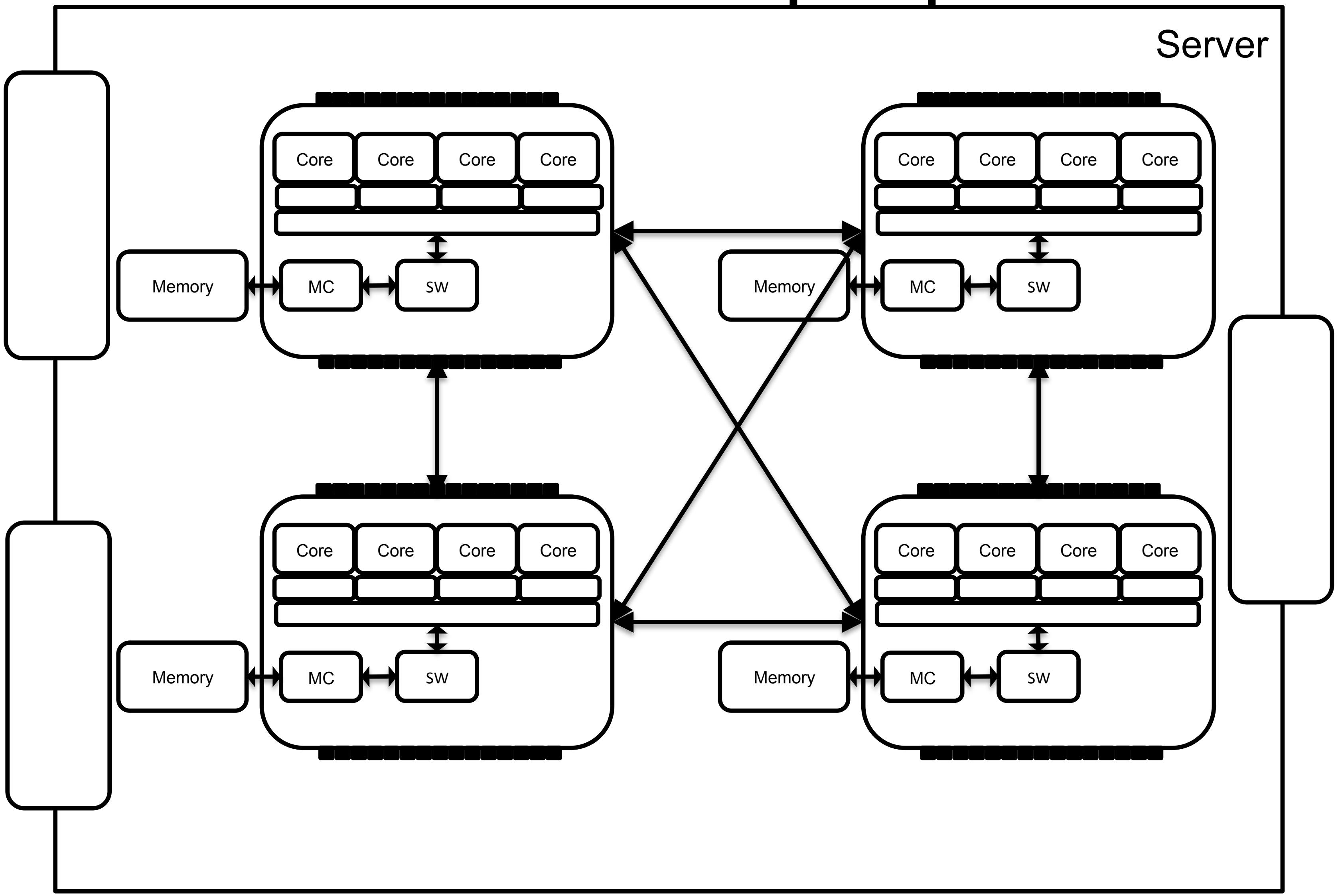
The ideal hardware?



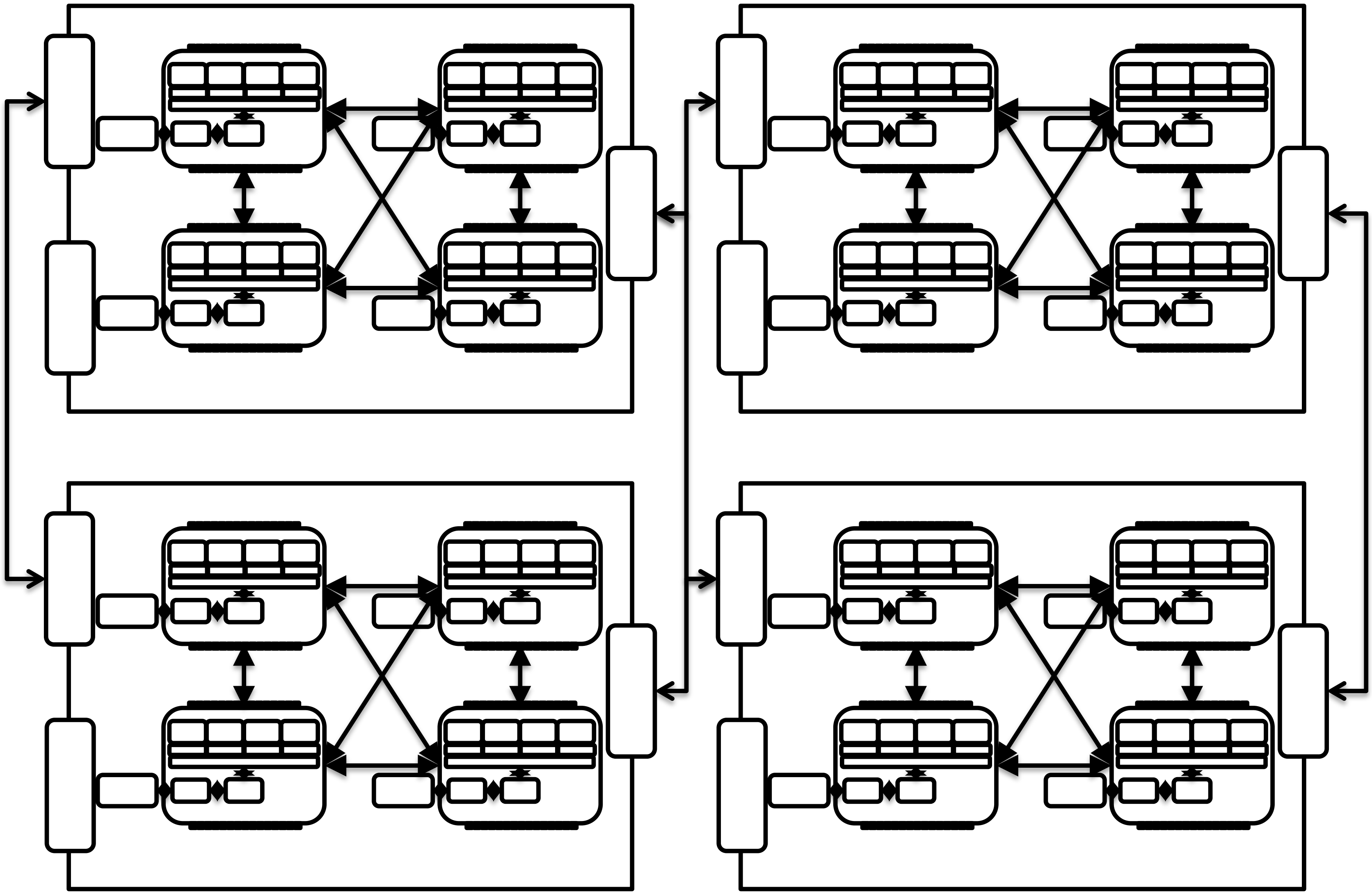
The reality: A multi-core processor



A server with multiple processors



A system with multiple servers



How to program parallel hardware?

Parallel Programming Models

Techniques

- Shared Memory/Threads
- Message Passing
- Data Parallelism
- Combinations (hybrid)

Shared Memory

- Concurrent tasks share common (logical) address space
- (Does not imply physical shared memory)
- No explicit communication required
- To coordinate access, locks/semaphores required
- Problem: Data locality hard to manage (which data belongs to which process)

Threads

- An OS process can start multiple threads
- Each thread has local data, but shares the resources of the parent OS process
- Threads communicate through global memory
- Threads are often associated with shared memory progamming
- Require syncronization, e.g., locks/semaphores
- Example: POSIX Threads

Message Passing

- Each concurrent task has own local memory
- Tasks can reside on one or on different machines
- Tasks communicate and transfer data by sending/receiving messages
- Example: OpenMPI

Data Parallel

- Data resides in shared data structure (array, table, cube, ...)
- Concurrent tasks work independently on data partitions
- Tasks perform same operation
- Sometimes, merge required to create final result
- Task scheduling done by execution framework
- Example: OpenMP ParallelFor, Map/Reduce

Map Reduce

An example for data parallel programming

Parallelization with MapReduce

- MapReduce is a programming paradigm for shared-nothing cluster
  - Developed by Google/Yahoo to analyze large (petabyte) data-sets
  - Allows for automatic parallelization and linear scaling of MapReduce programs

- In a narrow sense, MapReduce describes a programming model based on a ***map*** and a ***reduce*** function (both well known in functional programming):

  map(*key1, value1*) → list of <key2*, value2*>

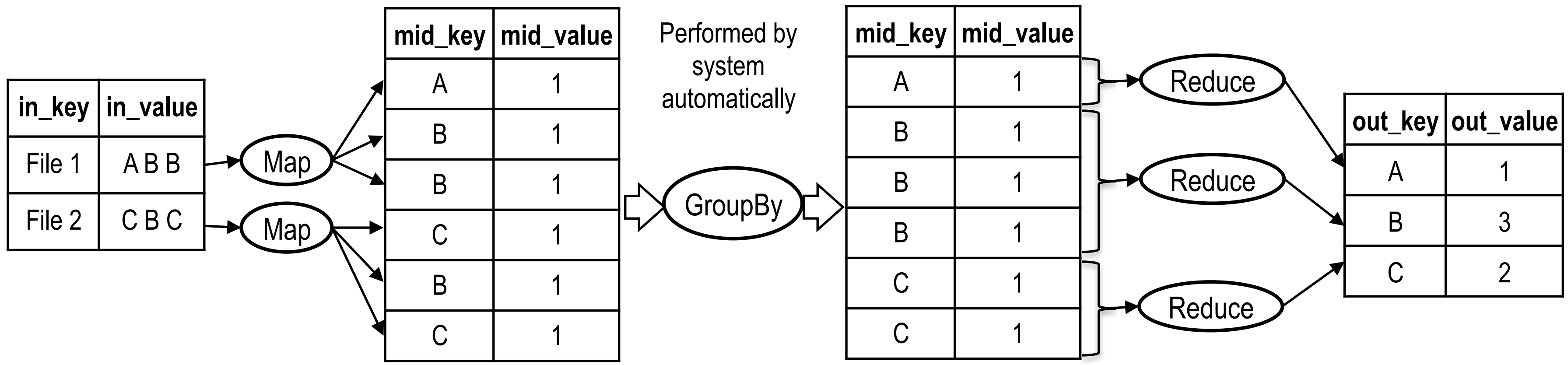
  reduce(*key2*, list of <value2>) → a list of <*key3, value3*>

  map and reduce functions process <*key, value*> pairs independently and thus can be parallelized easily.

- In a wider sense, MapReduce describes an execution framework for distributing map and reduce tasks in a shared-nothing cluster.

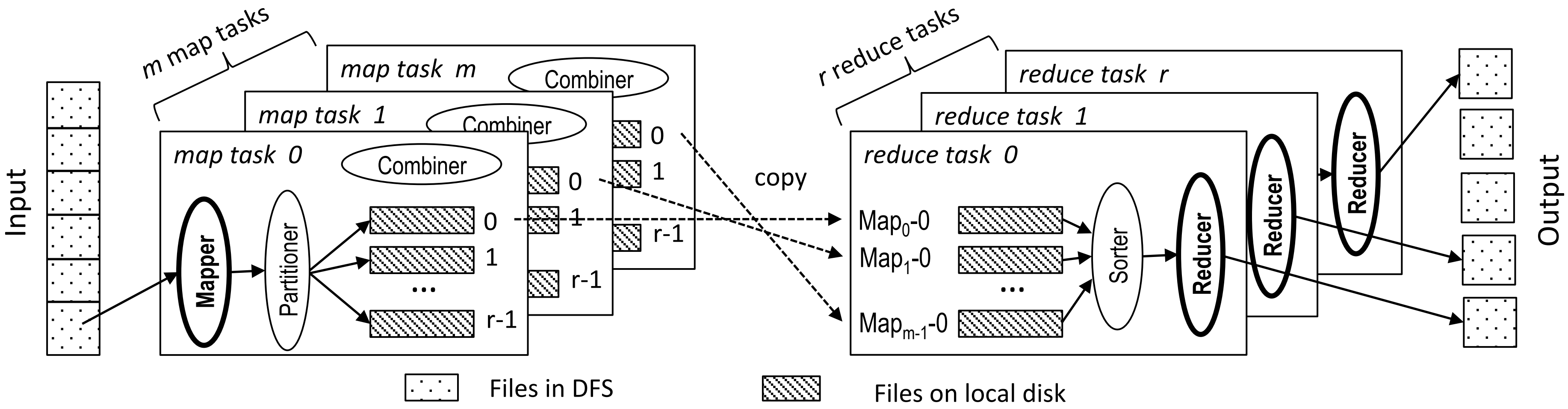
MapReduce Programming Model

Example: Word Count



- Input is a list of key value pairs (E.g. keys="*file names*", values="*file content*")
- A map function produces zero or more <*mid_key, mid_value*> pairs
- The system performs a group-by operation on *mid_key*
- A reduce function processes the list of *mid_values* belonging to a *mid_key* and produces a list of <*out_key, out_value*> pairs, here aggregate over values

MapReduce Execution Engine



- Engine schedules map and reduce tasks, i.e. the execution of the map or reduce function on a <*key, value*> pair

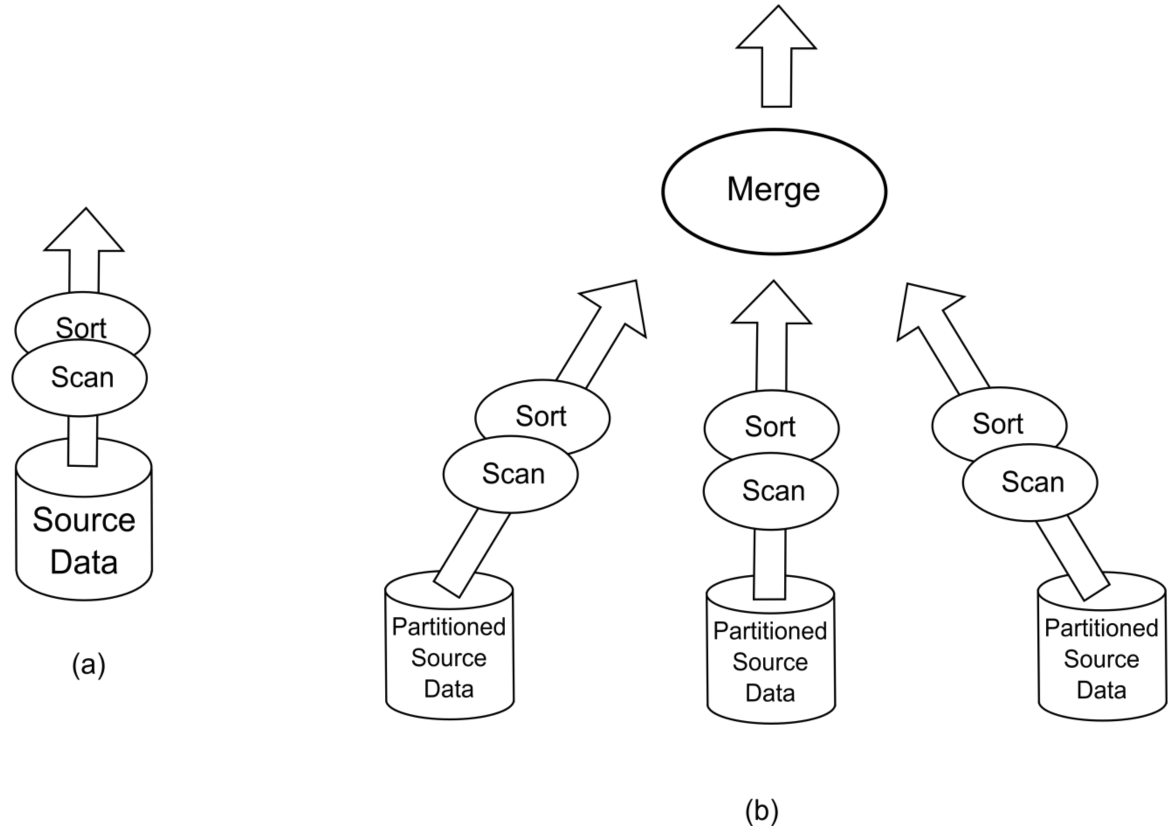
- Programmer can focus on algorithm and has to implement only a map and reduce function. The execution engine takes care of:
  - Task distribution: takes co-location of data into account
  - Shipping data between nodes: communication happens via disk!

Drawbacks and Problems

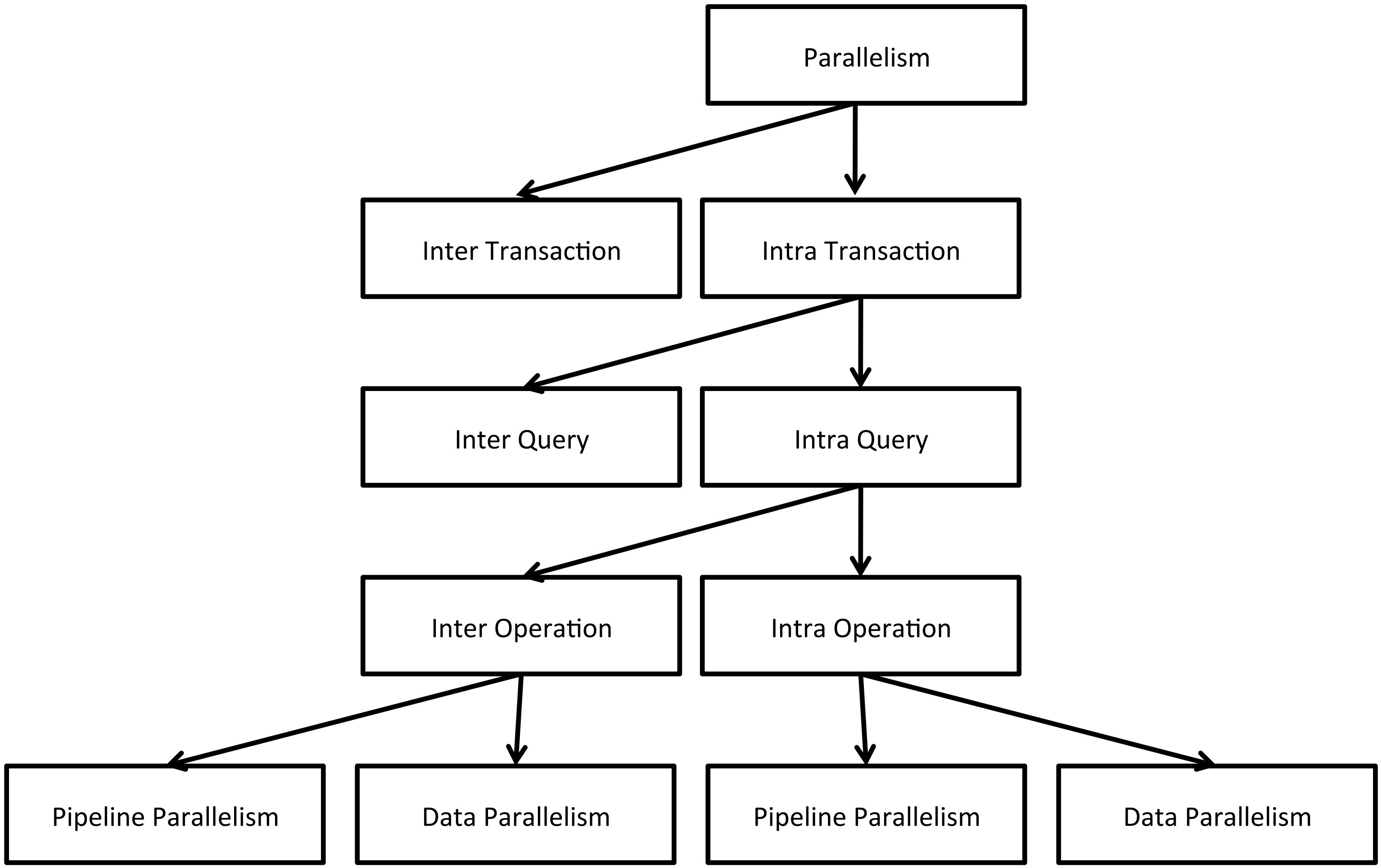
- Not all problems can be implemented efficiently with map and reduce functions! E.g. iterative algorithms
- Batch-oriented execution model, thus not suited for interactive data analysis
- Communication via files is great for fault-tolerance, but inefficient for small datasets
- Good performance is only achieved if additional components e.g. *combiner*, *partitioner*, and *sorter* are also tuned by the programmer. -> Programmer cannot focus on algorithm only!

Parallelism in NewDB

(a) Pipeline and (b) Data Parallelism



Approaches to parallelism



Example (Single Blade)

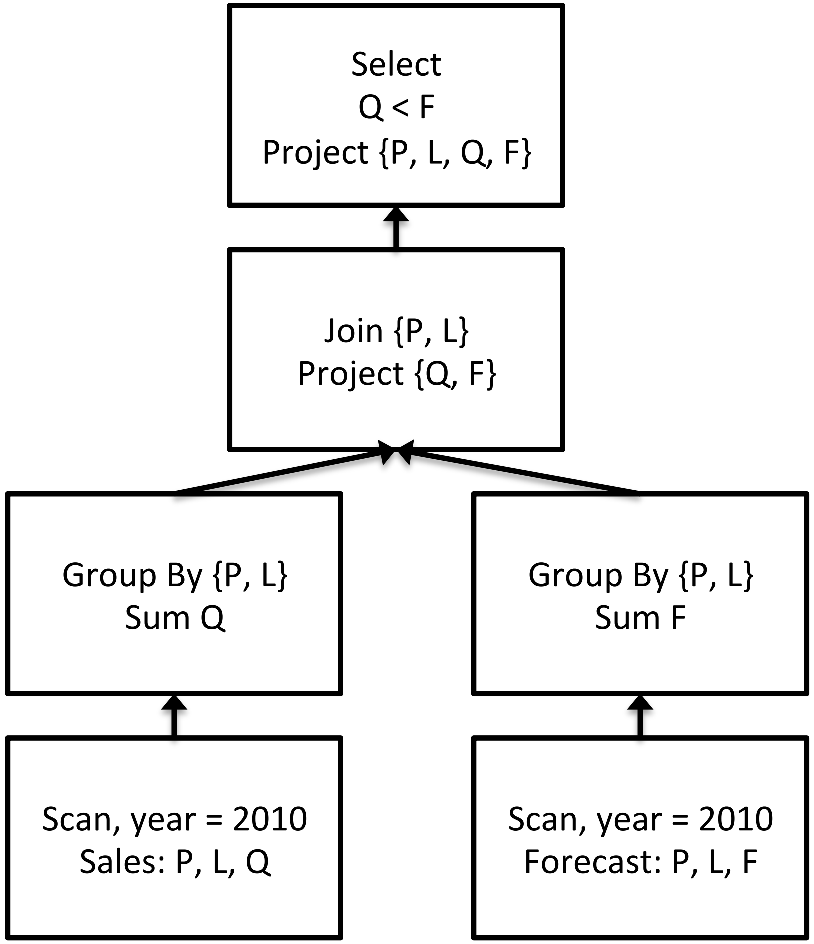
- Table "Sales"
  - Columns: Product P, Location L, Quantity Q, Year Y

- Table "Forecast"
  - Columns: Product P, Location L, Forecast F, Year Y

- Query:
  - "Which product at which location had lower quantity forecasted overall sales in 2010?"

```
SELECT Sales.P, Sales.L, SUM(Sales.Q) as QTY, SUM(Forecast.F) as FCST
FROM Sales, Forecast
WHERE Sales.P  = Forecast.P and Sales.L = Forecast.L and Sales.Y = 2010
GROUP BY Sales.P, Sales.L
HAVING QTY < FCST
```

Inter-Operator Parallelism (One Blade)



Example (Four Blades)

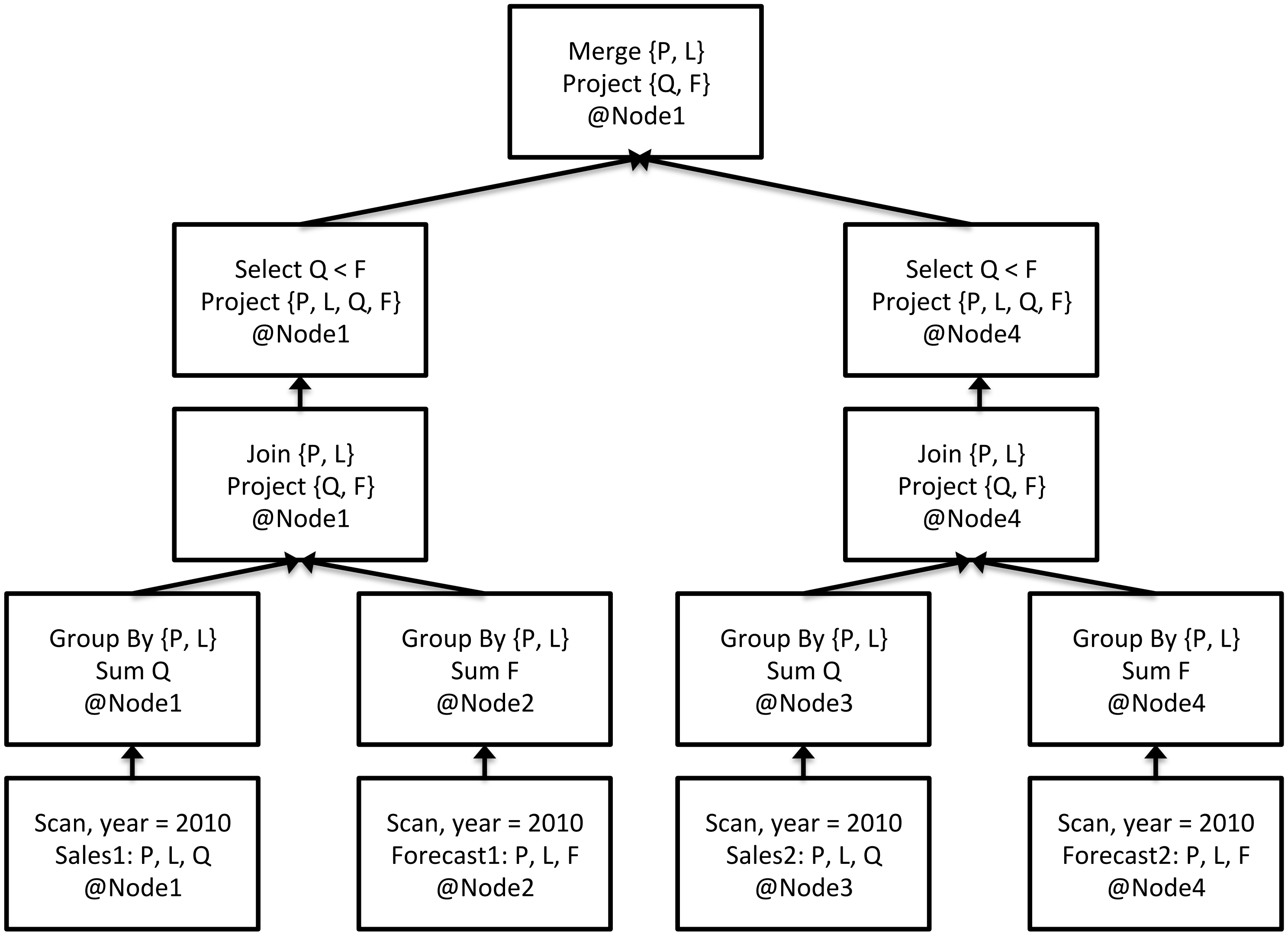
- Table “Sales”
  - Columns: Product P, Location L, Quantity Q, Year Y
  - Split into two parts: “Sales1” and “Sales 2” based on P and L
  - “Sales 1” stored at “node1”
  - “Sales 2” stored at “node3”

- Table “Forecast”
  - Columns: Product P, Location L, Forecast F, Year Y
  - Split into two parts: “Forecast1” and “Forecast2” based on P and L
  - “Forecast1” stored at “node2”
  - “Forecast2” stored at “node4”

- Query (remains the same; distribution transparent to the user):
  - “Which product at which location had lower than forecasted overall sales in 2010?”

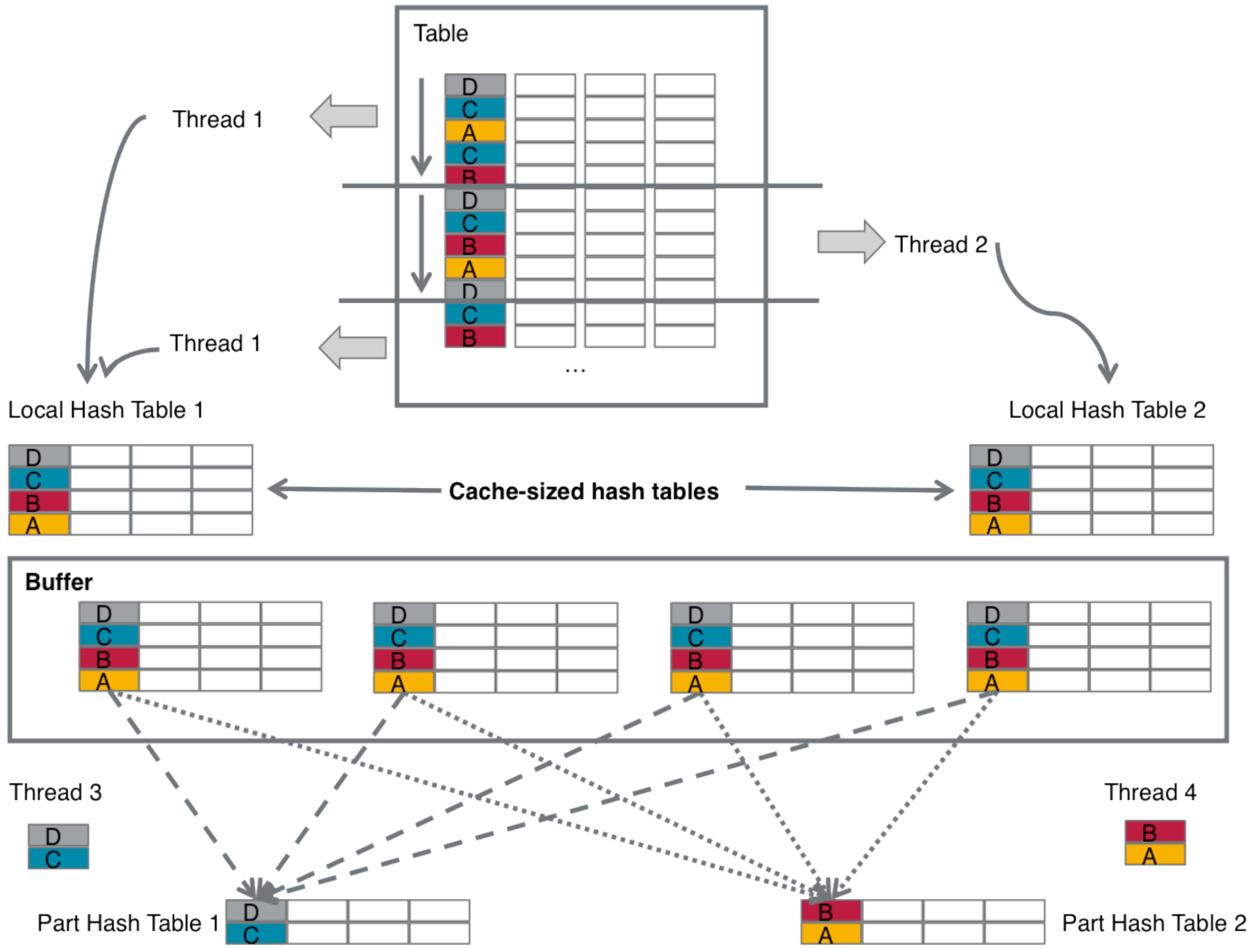
```
SELECT Sales.P, Sales.L, SUM(Sales.Q) as QTY, SUM(Forecast.F) as FCST
FROM Sales, Forecast
WHERE Sales.P  = Forecast.P and Sales.L = Forecast.L and Sales.Y = 2010
GROUP BY Sales.P, Sales.L
HAVING QTY < FCST
```

Inter-Operator Parallelism (Four Blades)



Parallel Filter / Aggregation

- **1.) *n* Aggregation Threads**
  - 1) each thread fetches a small part of the input relation
  - 2) aggregate part and write results into a small hash-table
  - **If the entries in a hash-table exceed a threshold, the hash-table is moved into a shared buffer**
- **2.) *m* Merger Threads**
  - 3) each merge thread operates on a partition of the hash function values and writes its result into a private part hash-table
  - 4) the final result is obtained by concatenating the part hash-tables



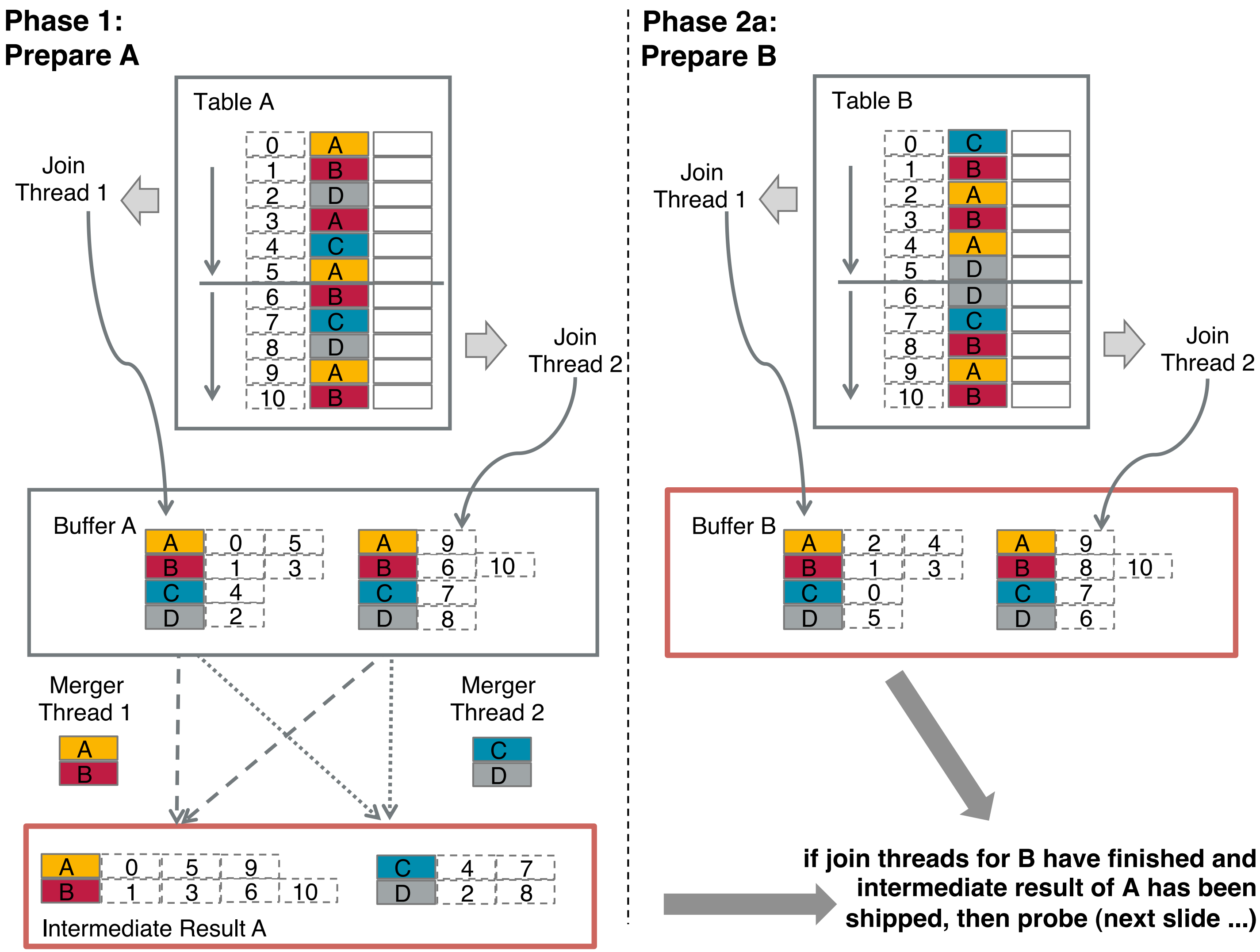
Key Observations

- Algorithm
  - 2-stage pipeline (pipeline parallelism)
  - Programming Model: shared memory/threads, data parallel
- Synchronization: Every thread writes into its private datastructure
  → No synchronization required for that
- Data skew handled by small input work package size (compared to fact table size)
- Cache-awareness due to fixed size of local hash tables
- Main-memory consumption bounded by buffer size
- Number of threads can be adjusted
- Similar algorithm for parallel join computation (see next slide)

Parallel Filter/Join

Like aggregation, joins can be computed using hash-tables

- **1.) Prepare Phase:** parallel computation of part hash-tables on the smaller input relation

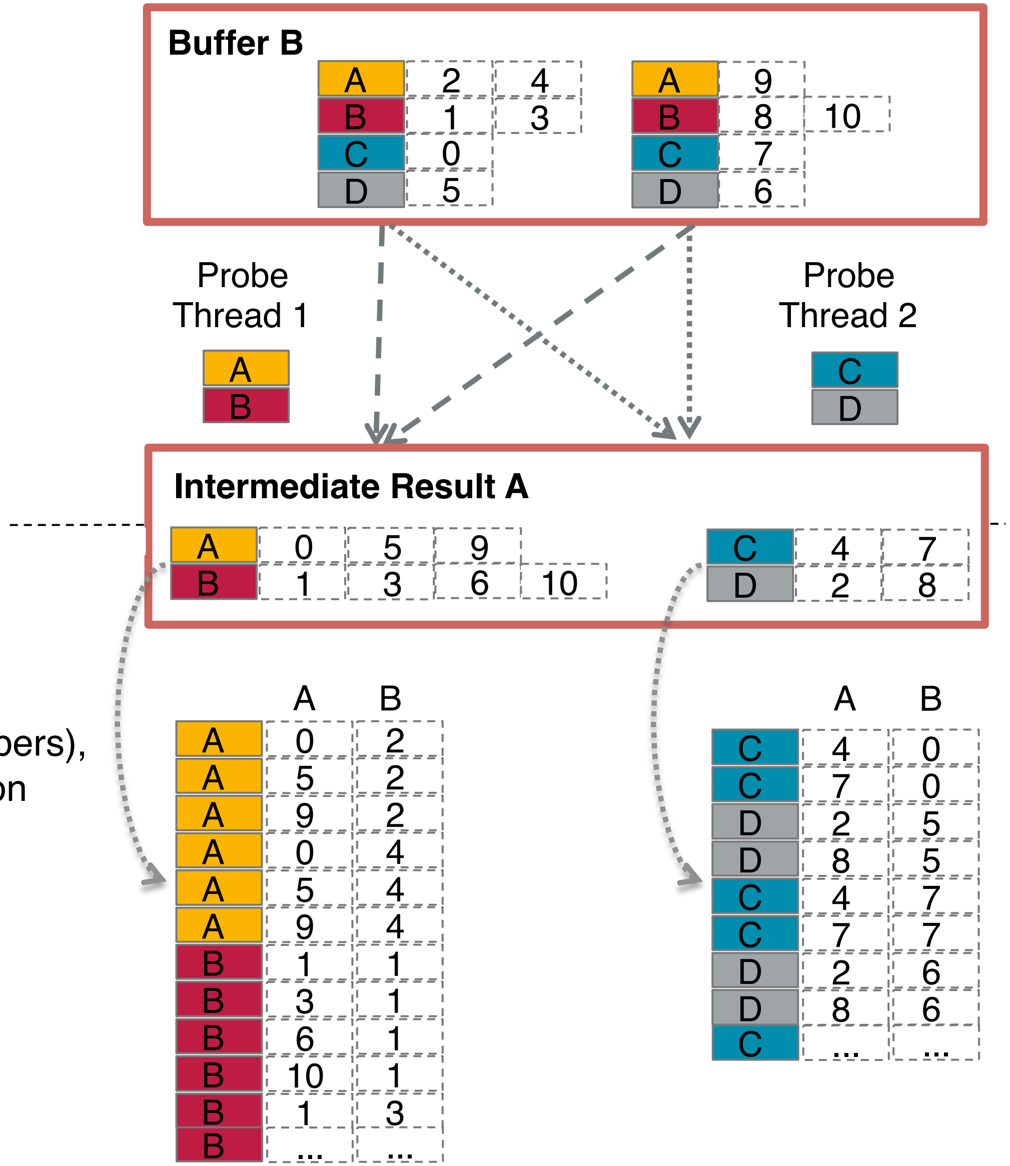


Parallel Filter/Join

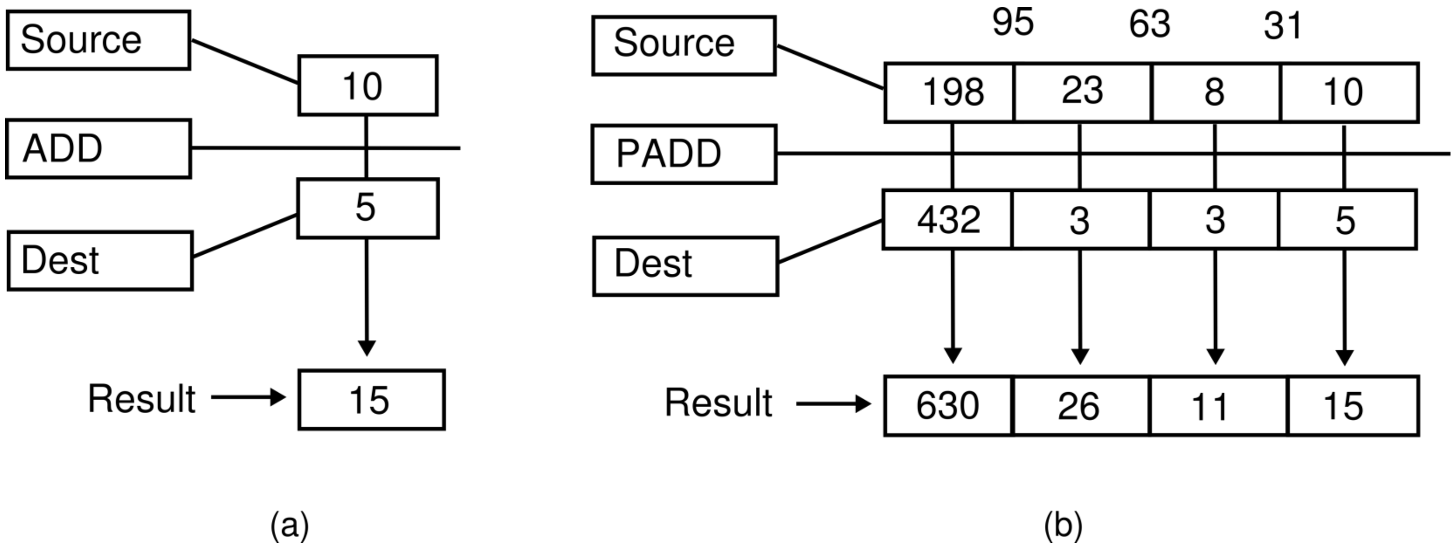
- **2.) Probe Phase:** probing of the larger input relation against the part hash tables:
  - A buffer of hash maps is created in parallel
  - Local hash maps are compared with part hash-maps
- **3.) Concatenate and Materialize** (not shown)
  - List of all matching rows in tables A and B is created
  - Additional fields for these rows are retrieved from tables A and B

Phase 2b: Probe

Virtual Tables (only row numbers), one per partition



(a) Scalar Add and (b) Parallel Add (Low-Level Parallel Column Access)



Implemented in NewDB with Intel Streaming SIMD Extensions (SSE)

Further/Advanced Parallelization Topics in NewDB

- NewDB map/reduce implementation
- Dynamic CalcEngine Split
- Parallelization of further NewDB algorithms: sort, search, distinct values

Map/Reduce in NewDB

- NewDB lends the concepts of *map* and *reduce*
  - But implementation is different from Google MR and has different scope
- Idea: User-defined functions *map* and *reduce* to parameterize aggregation algorithm
- Recap: Aggregation algorithm does grouping and aggregation
- Grouping: *map* as user-defined group calculation (group by)
  - Implemented by row-level function
- Aggregation: *reduce* as user-defined aggregation calculation (aggregate)
- Status
  - *Map* can be specified as L function and passed to parallel aggregation
  - *Reduce* not implemented yet; standard aggregation functions can be applied (sum, min, max, etc.)

Dynamic CalcEngine Split (Example)

- Table "Sales"
  - Columns: Product P, Location L, Quantity Q, Year Y

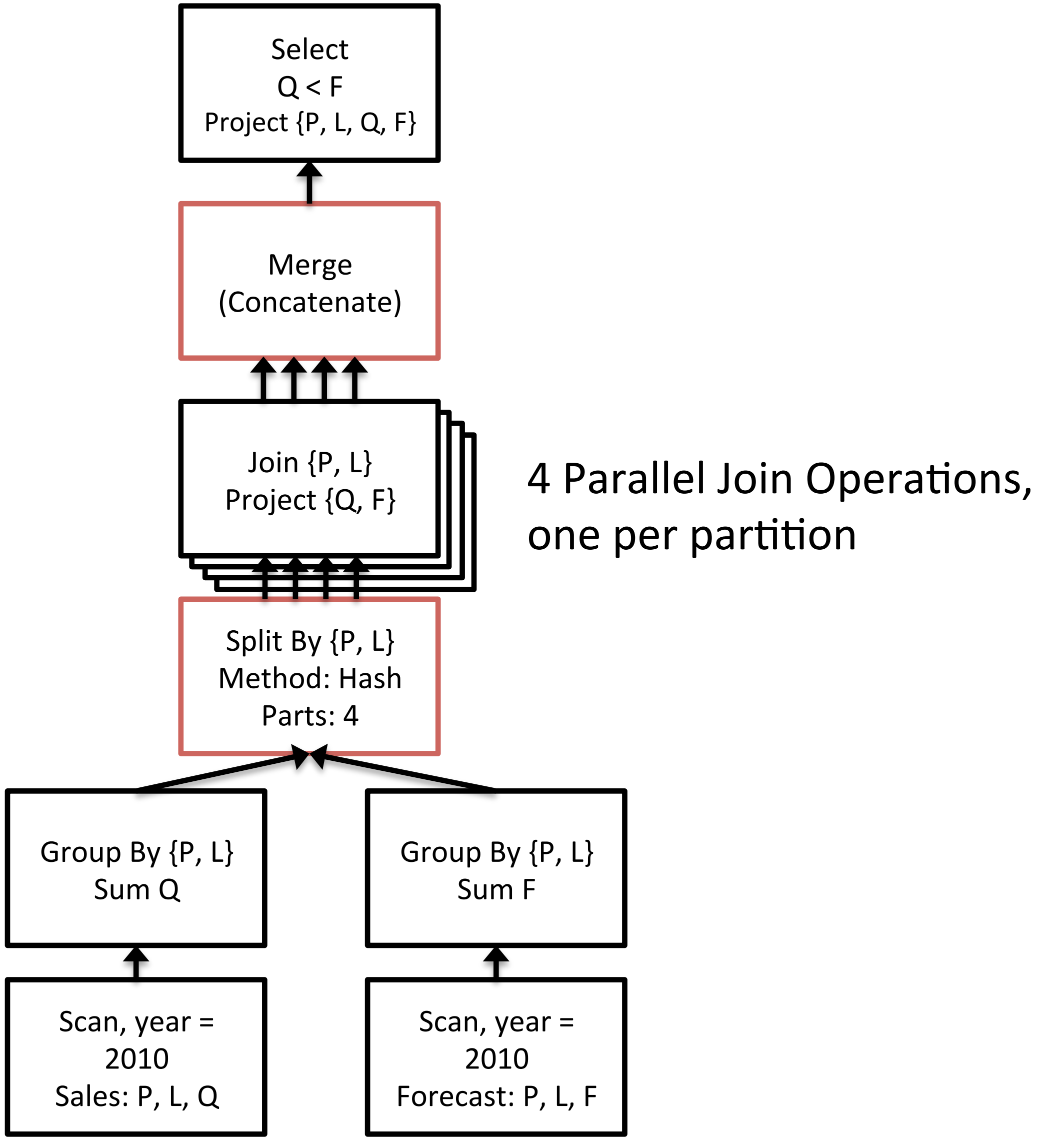
- Table "Forecast"
  - Columns: Product P, Location L, Forecast F, Year Y

- Query:
  - "Which product at which location had lower quantity forecasted overall sales in 2010?"

```
SELECT Sales.P, Sales.L, SUM(Sales.Q) as QTY, SUM(Forecast.F) as FCST
FROM Sales, Forecast
WHERE Sales.P  = Forecast.P and Sales.L = Forecast.L and Sales.Y = 2010
GROUP BY Sales.P, Sales.L
HAVING QTY < FCST
```

- Goal: Introduce dynamic split to parallelize the join computation

Dynamic CalcEngine Split (Example)



Sources

- Parallel Hardware: Future SOC talk by C. Mathis
- Parallel Programming Models: https://computing.llnl.gov/tutorials/parallel_comp/
- Map/Reduce: "Map-Reduce Meets Wider Varieties of Applications" Shimin Chen, Steven W. Schlosser, Intel Research 2008. Technical Report IRP-TR-08-05
- Parallelism in NewDB: The BOOK and Hassos BTW Paper
- NewDB Map/Reduce: Martin Richtarsky
- Dynamic CalcEngine Split: Daniel Bäumges and the Calcies