10. Parallel Database Systems

Chapter 14

Parallel Database Systems

Fact and Database Problem

- A large volume of data uses disk and large main memory
  - Speed(disk) << speed(RAM) << speed(microprocessor)

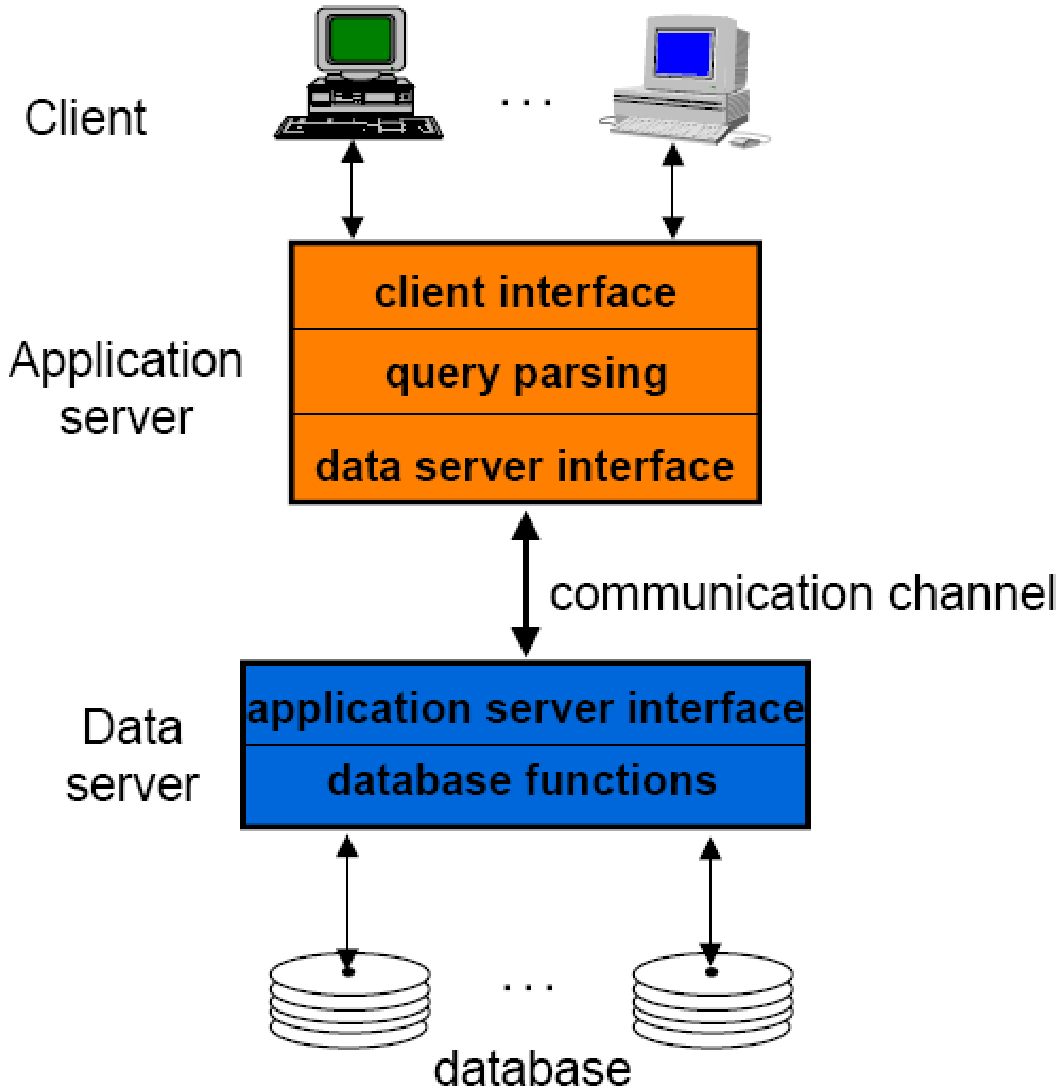
The Solution

- Increase the I/O bandwidth
  - Data partitioning
  - Parallel data access
- Origins (1980's): database machines
  - Hardware-oriented: bad cost-performance failure
  - Notable exception: ICL's CAFS Intelligent Search Processor
- 1990's: same solution but using standard hardware components integrated in a multiprocessor
  - Software-oriented
  - Exploiting continuing technology improvements

Multiprocessor Objectives

- High-performance with better cost-performance than mainframe or vector super computer
- Use many nodes, each with good cost-performance, communicating through network
  - Good cost via high-volume components
  - Good performance via bandwidth
- Trends
  - Microprocessor and memory (DRAM): off-the-shelf
  - Network (multiprocessor edge): custom
- The real challenge is to parallelize applications to run with good load balancing

Data Server Architecture



Objectives of Data Servers

- Avoid the shortcomings of the traditional DBMS approach
  - Centralization of data and application management
  - General-purpose OS (not DB-oriented)
- By separating the functions between
  - Application server (or host computer)
  - Data server (or database computer or back-end computer)

Data Server Approach: Assessment

- Advantages
  - Integrated data control by the server (black box)
  - Increased performance by dedicated system
  - Can better exploit parallelism
  - Fits well in distributed environments
- Potential problems
  - Communication overhead between application and data server
    - High-level interface
  - High cost with mainframe servers

Parallel Data Processing

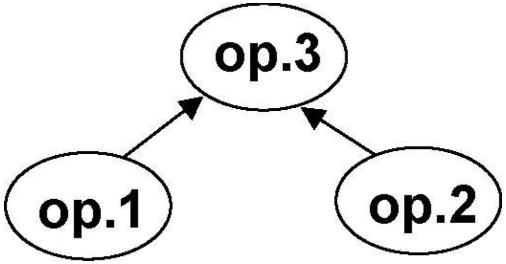
- Three ways of exploiting high-performance multiprocessor systems:
  - Automatically detect parallelism in sequential programs (e.g., Fortran, OPS5)
  - Augment an existing language with parallel constructs (e.g., C*, Fortran90)
  - Offer a new language in which parallelism can be expressed or automatically inferred

Parallel Data Processing (*cont*.)

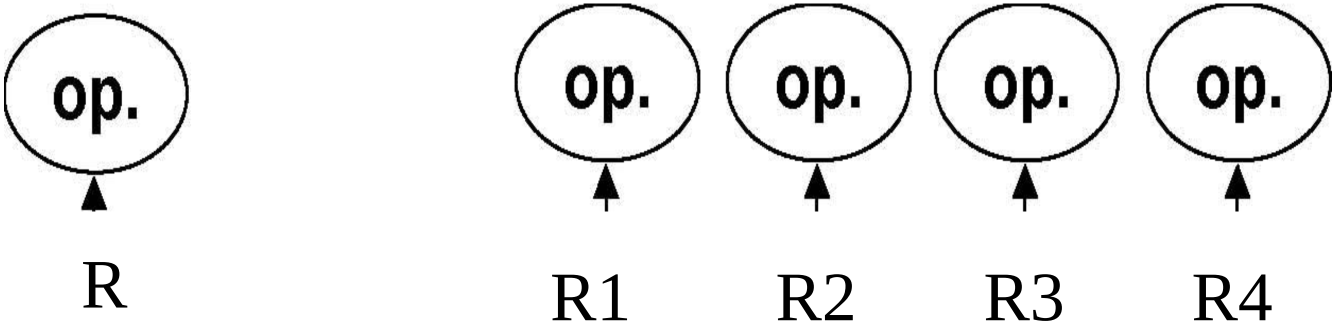
- Critique
  - Hard to develop parallelizing compilers, limited resulting speed-up
  - Enables the programmer to express parallel computations but too low-level

Data-based Parallelism

- Inter-operation
  - operations of the same query in parallel



- Intra-operation
  - the same operation in parallel on different data partitions



Parallel DBMS

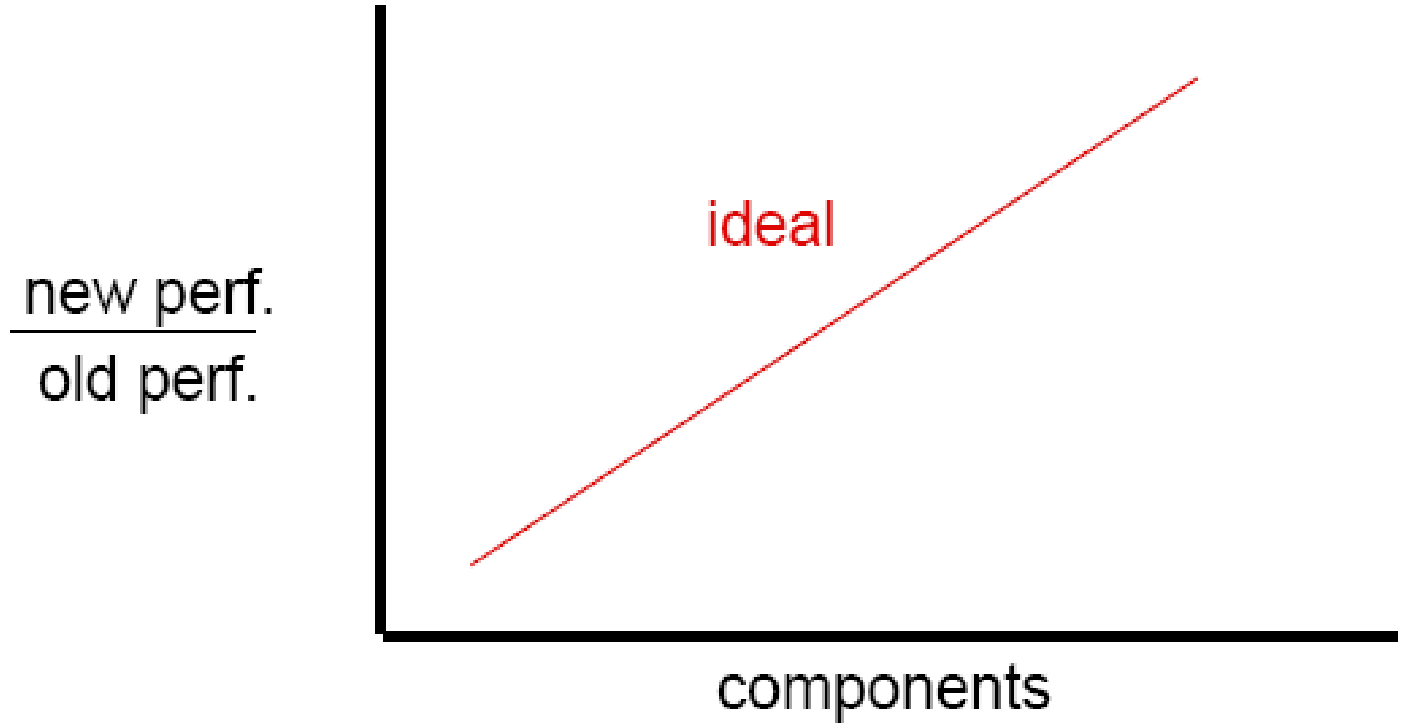
- Loose definition: a DBMS implemented on a tightly coupled multiprocessor
- Naturally extends to distributed databases with one server per site

Parallel DBMS -Objectives

- Much better cost / performance than mainframe solution
- High-performance through parallelism
  - High throughput with inter-query parallelism
  - Low response time with intra-operation parallelism
- High availability and reliability by exploiting data replication
- Extensibility with the ideal goals
  - Linear speed-up
  - Linear scale-up

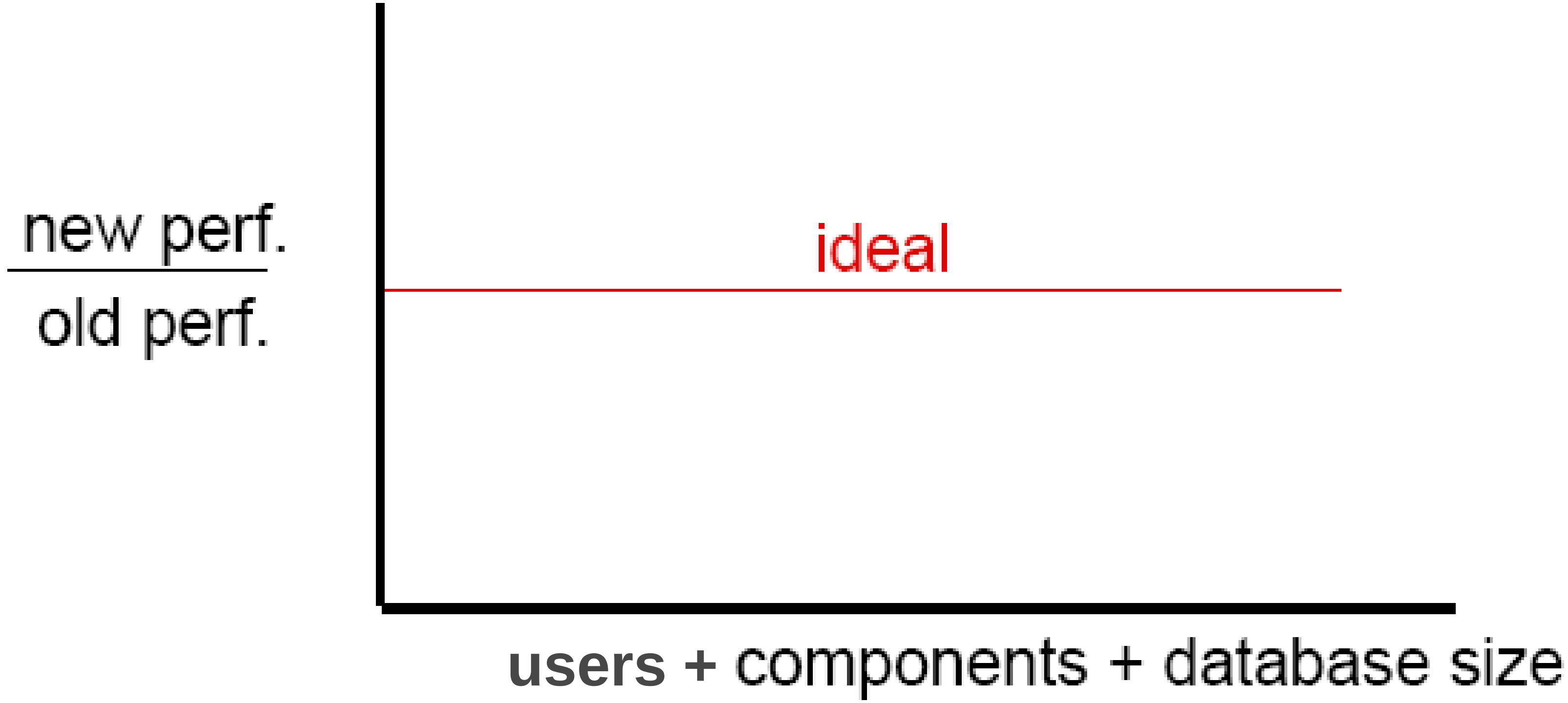
Linear Speed-up

- Linear increase in performance for a constant DB size and proportional increase of the system components (processor, memory, disk)



Linear Scale-up

- Sustained performance for a linear increase of database size and proportional increase of the system components.

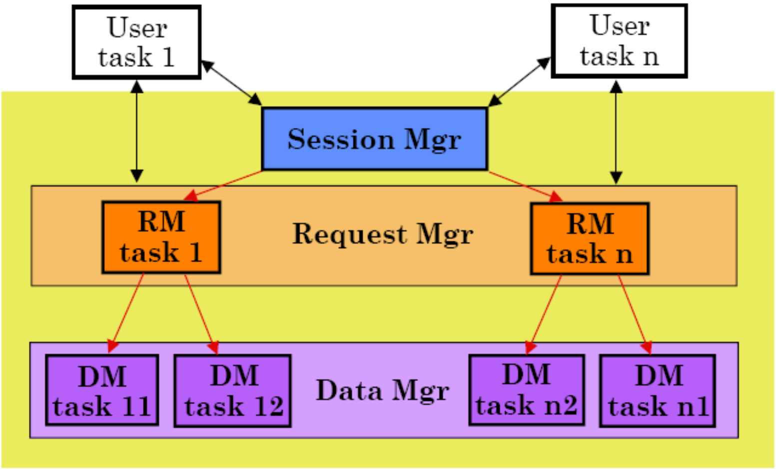


Barriers to Parallelism

- Startup
  - The time needed to start a parallel operation may dominate the actual computation time
- Interference
  - When accessing shared resources, each new process slows down the others (hot spot problem)
- Skew
  - The response time of a set of parallel processes is the time of the slowest one
- Parallel data management techniques intend to overcome these barriers

Parallel DBMS Functions

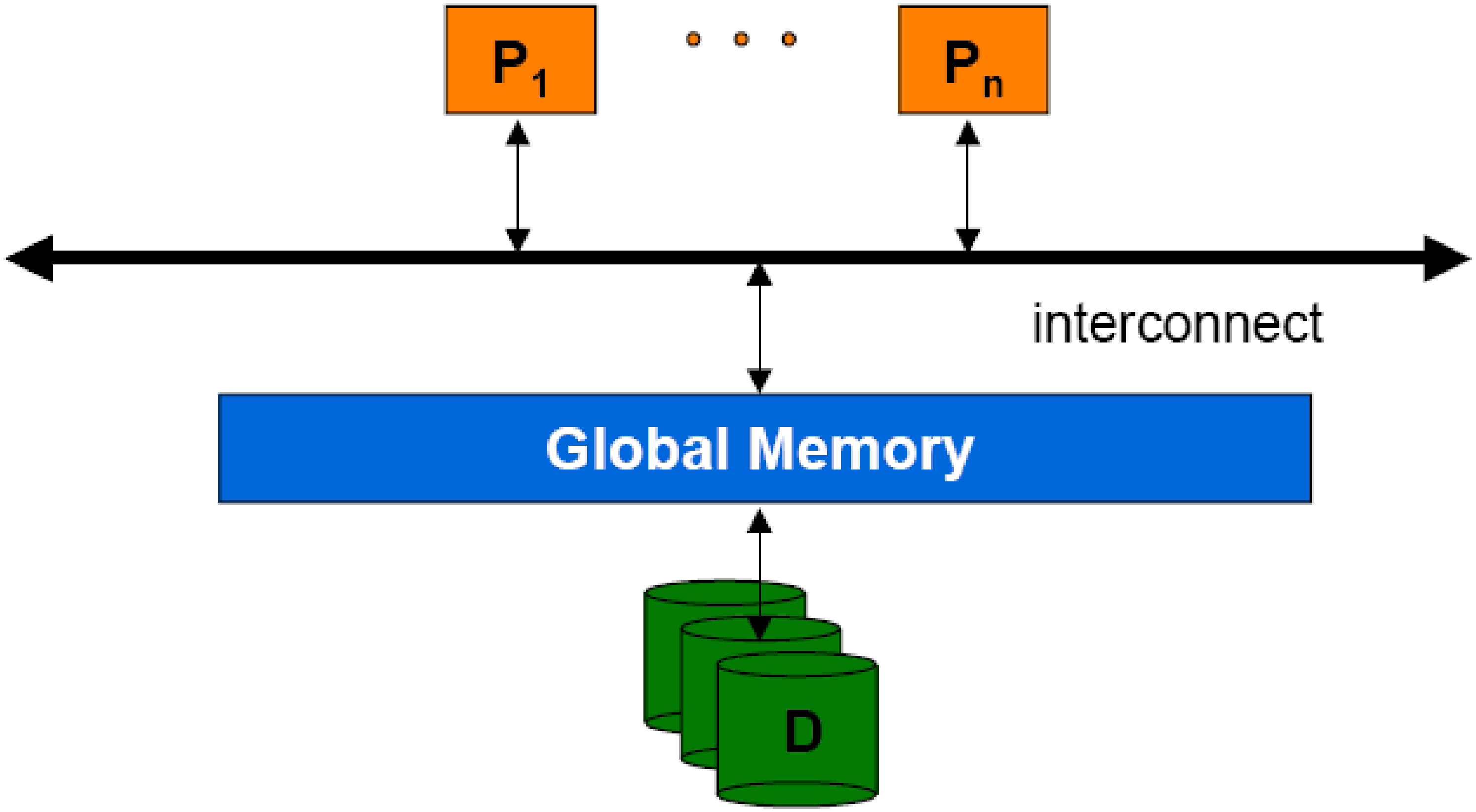
- Session manager
  - Host interface
  - Transaction monitoring
- Request manager
  - Compilation and optimization
  - Data directory management
  - Semantic data control
  - Execution control
- Data manager
  - Execution of DB operations
  - Transaction management support
  - Data management



Parallel System Architectures

- Multiprocessor architecture alternatives
  - Shared memory (shared everything)
  - Shared disk
  - Shared nothing (message-passing)
- Hybrid architectures
  - Hierarchical (cluster)
  - Non-Uniform Memory Architecture (NUMA)

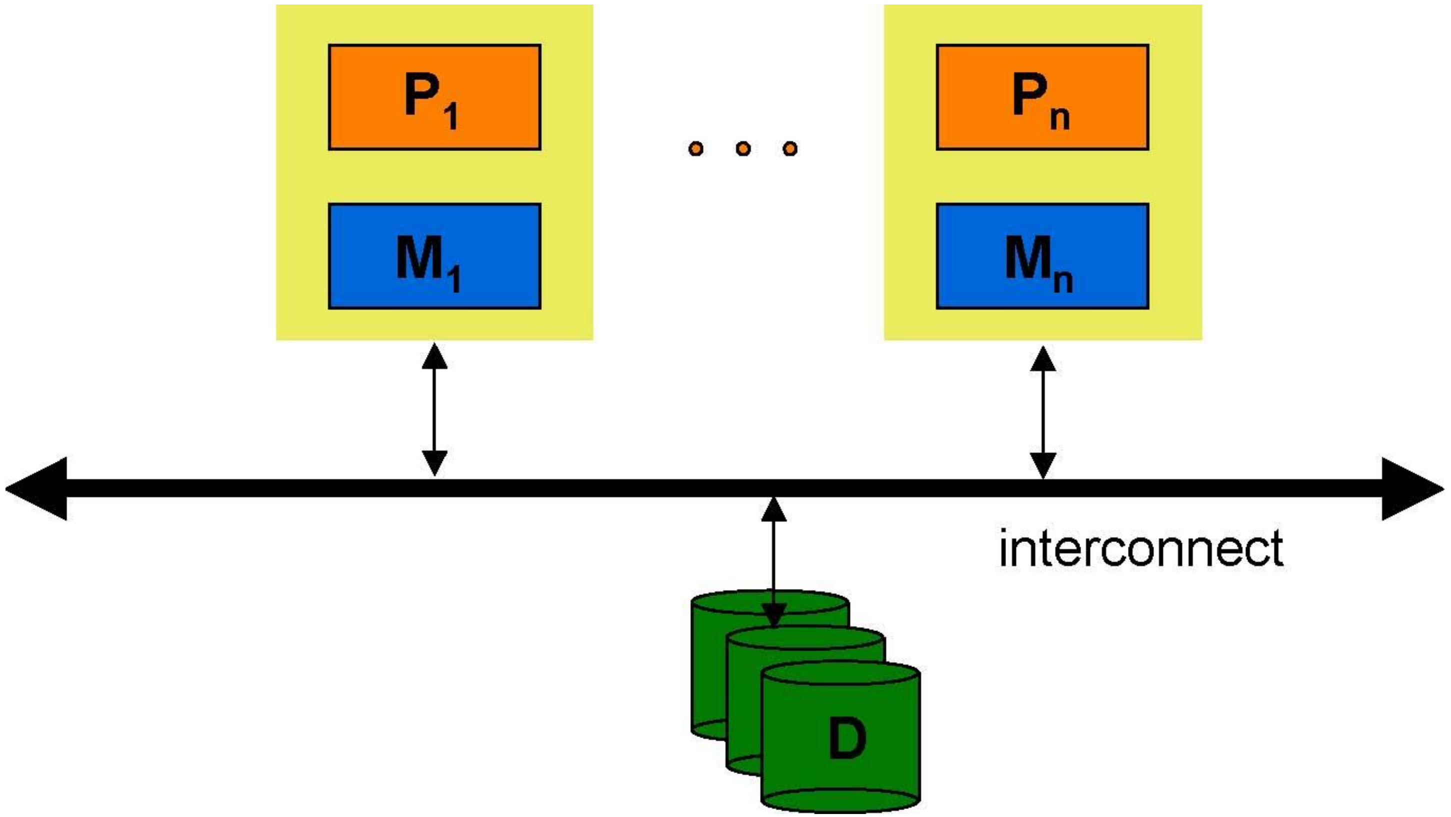
Shared Memory Architectures



Examples: DBMS on symmetric multiprocessors (Sequent, Encore, Sun, etc.)

- Simplicity, load balancing, fast communication
- Network cost, low extensibility

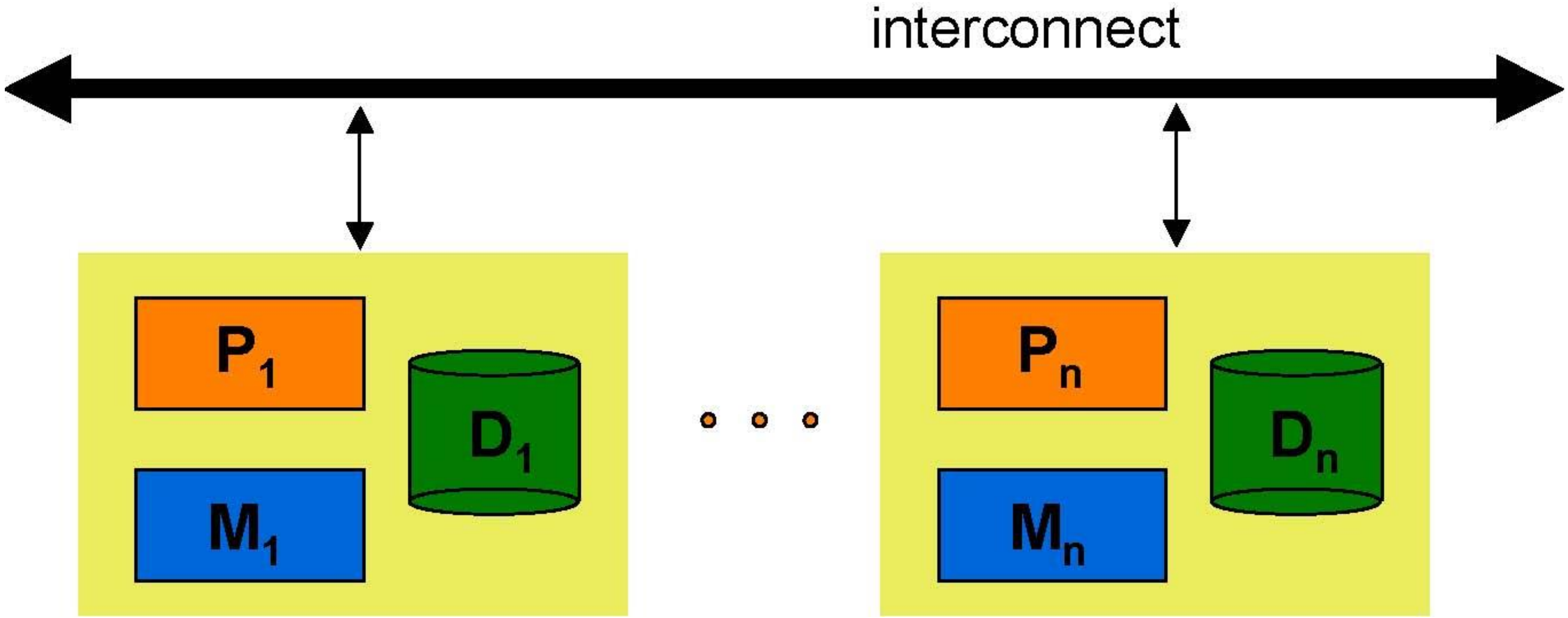
Shared-Disk Architecture



Examples: DEC's VAXcluster, IBM's IMS/VS Data Sharing

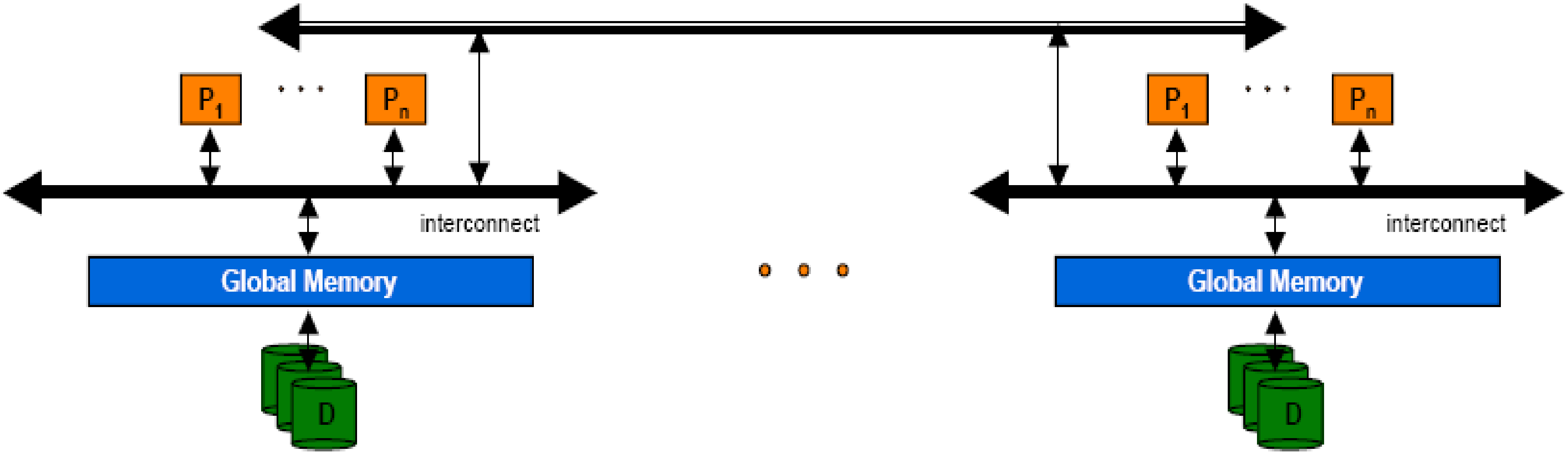
- network cost, extensibility, migration from uniprocessor
- complexity, potential performance problem for copy coherency

Shared-Nothing Architecture



Examples: Teradata (NCR), NonStopSQL (Tandem-Compaq),
Gamma (U. of Wisconsin), Bubba (MCC)
- Extensibility, availability
- Complexity, difficult load balancing

Hierarchical Architecture



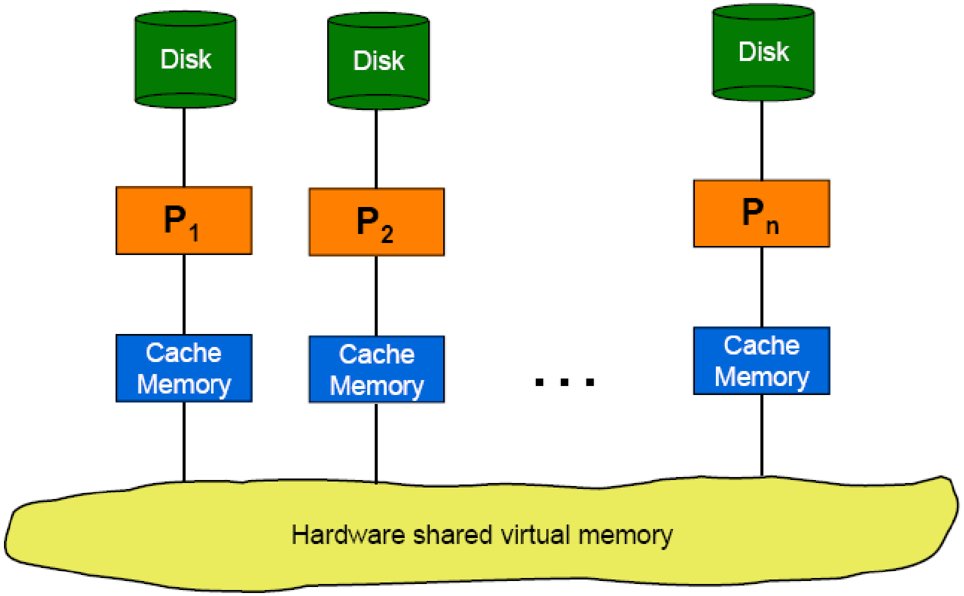
❖ Combines good load balancing of SM with extensibility of SN

Shared-Memory vs. Distributed Memory

- Mix two different aspects : addressing and memory
  - Addressing
    - Single address space : Sequent, Encore, KSR
    - Multiple address spaces : Intel, Ncube
  - Physical memory
    - Central : Sequent, Encore
    - Distributed : Intel, Ncube, KSR
- NUMA (Non-Uniform Memory Architecture): single address space on distributed physical memory eases application portability
  - Extensibility

Non-Uniform Memory Architecture (NUMA) Architectures

- Cache Coherent NUMA (CC-NUMA)
  - statically divide the main memory among the nodes
- Cache-Only Memory Architecture (COMA)
  - convert the per-node memory into a large cache of the shared address space

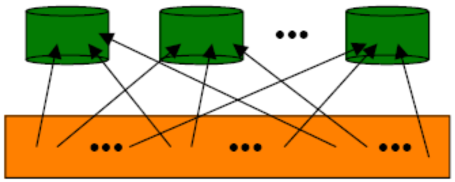


Parallel DBMS Techniques

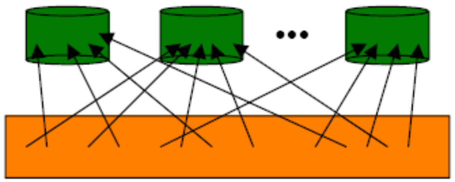
- Data placement
  - Physical placement of the DB onto multiple nodes
  - Static vs. Dynamic
- Parallel data processing
  - Select is easy
  - Join (and all other non-select operations) is more difficult
- Parallel query optimization
  - Choice of the best parallel execution plans
  - Automatic parallelization of the queries and load balancing
- Transaction management
  - Similar to distributed transaction management

Data Partitioning

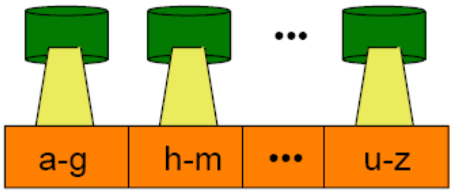
- Each relation is divided in *n* partitions (sub-relations), where *n* is a function of relation size and access frequency
- Implementation



Round-Robin



Hashing



Interval

  - Round-robin
    - – Map the i-th element to node (i mod n)
    - – Simple but only exact-match queries
  - B-tree index
    - – Support range queries but large index size
  - Hash function
    - – Only exact-match queries but small index size

Replicated Data Partitioning

- High-availability requires data replication
  - interleaved or chained partitioning to achieve load balancing

Node	1	2	3	4
Primary copy	R1	R2	R3	R4
Backup copy →		r 1.1	r 1.2	r 1.3
	r 2.3		r 2.1	r 2.2
	r 3.2	r 3.3		r 3.1

Interleaved Partitioning

Node	1	2	3	4
Primary copy	R1	R2	R3	R4
Backup copy	r4	r1	r2	r3

Chained Partitioning

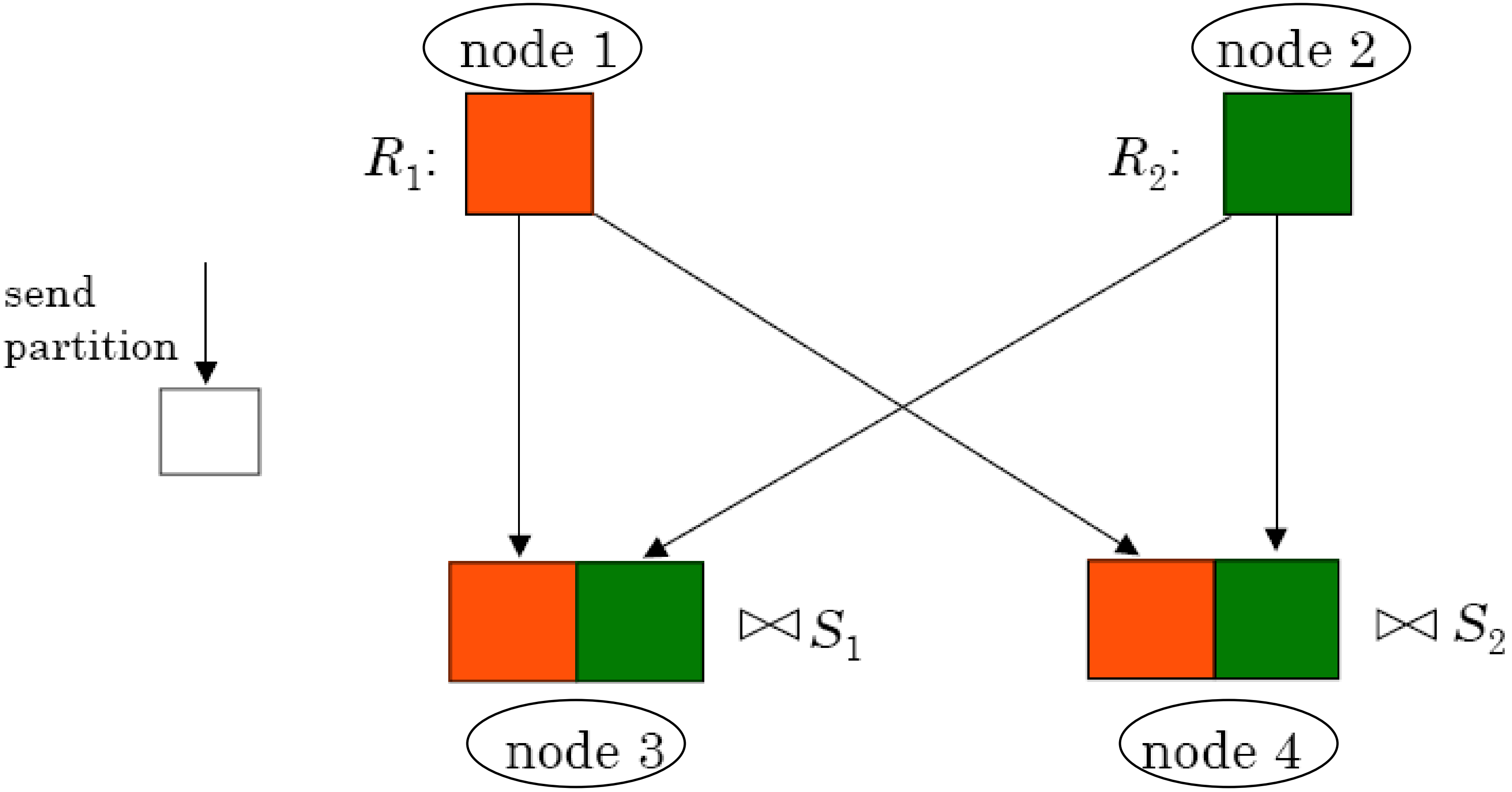
Placement Directory

- Perform two functions
  - *F1* (relname, placement attval) = log node-id
  - *F2* (log node-id) = phy node-id
- In either case, the data structure for *F1* and *F2* should be available when needed at each node

Join Processing

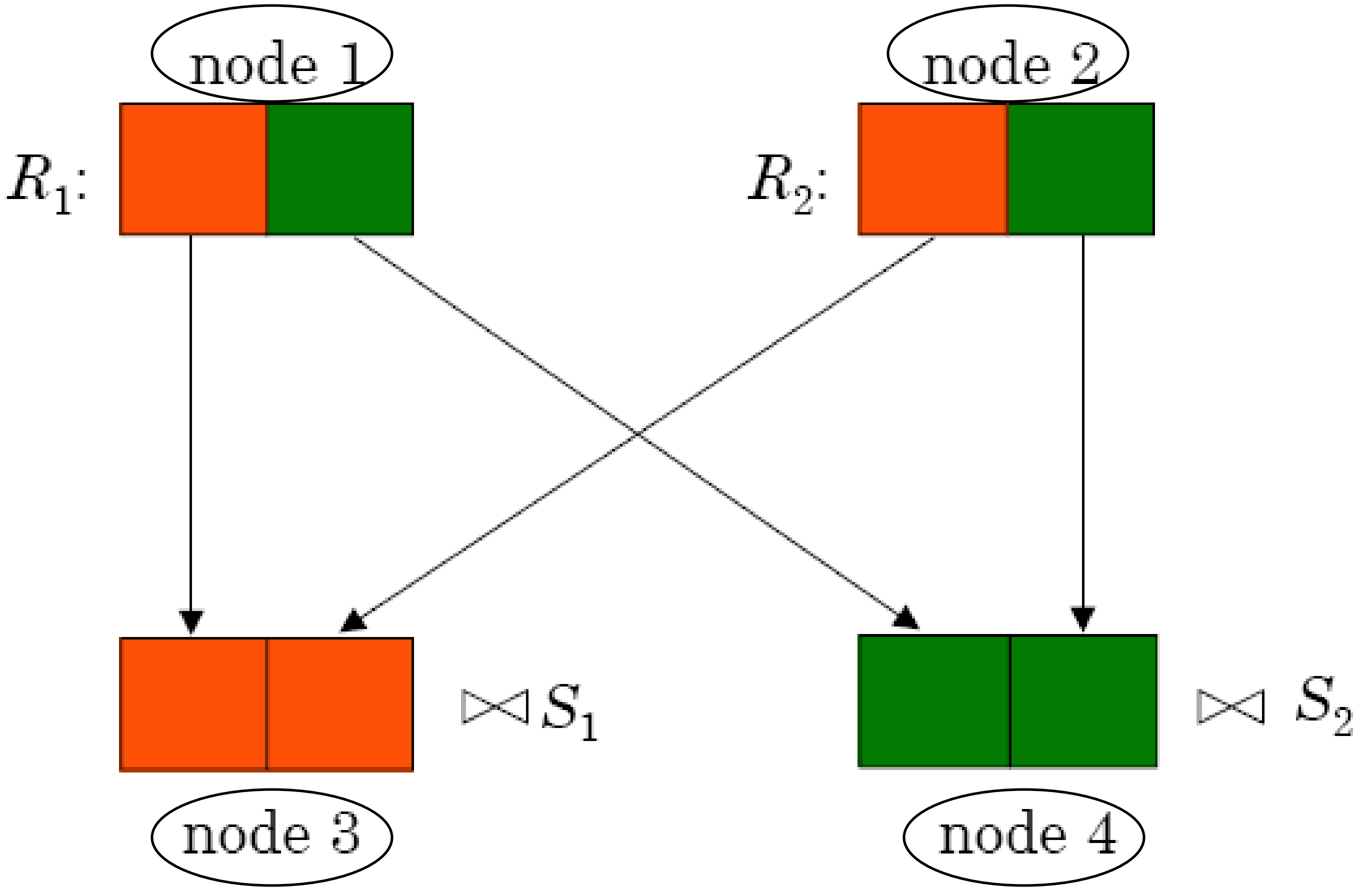
- Three basic algorithms for intra-operator parallelism
  - Parallel nested loop join: no special assumption
  - Parallel associative join: one relation is declustered on join attribute and equi-join
  - Parallel hash join: equi-join
- They also apply to other complex operators such as duplicate elimination, union, intersection, etc. with minor adaptation

Parallel Nested Loop Join



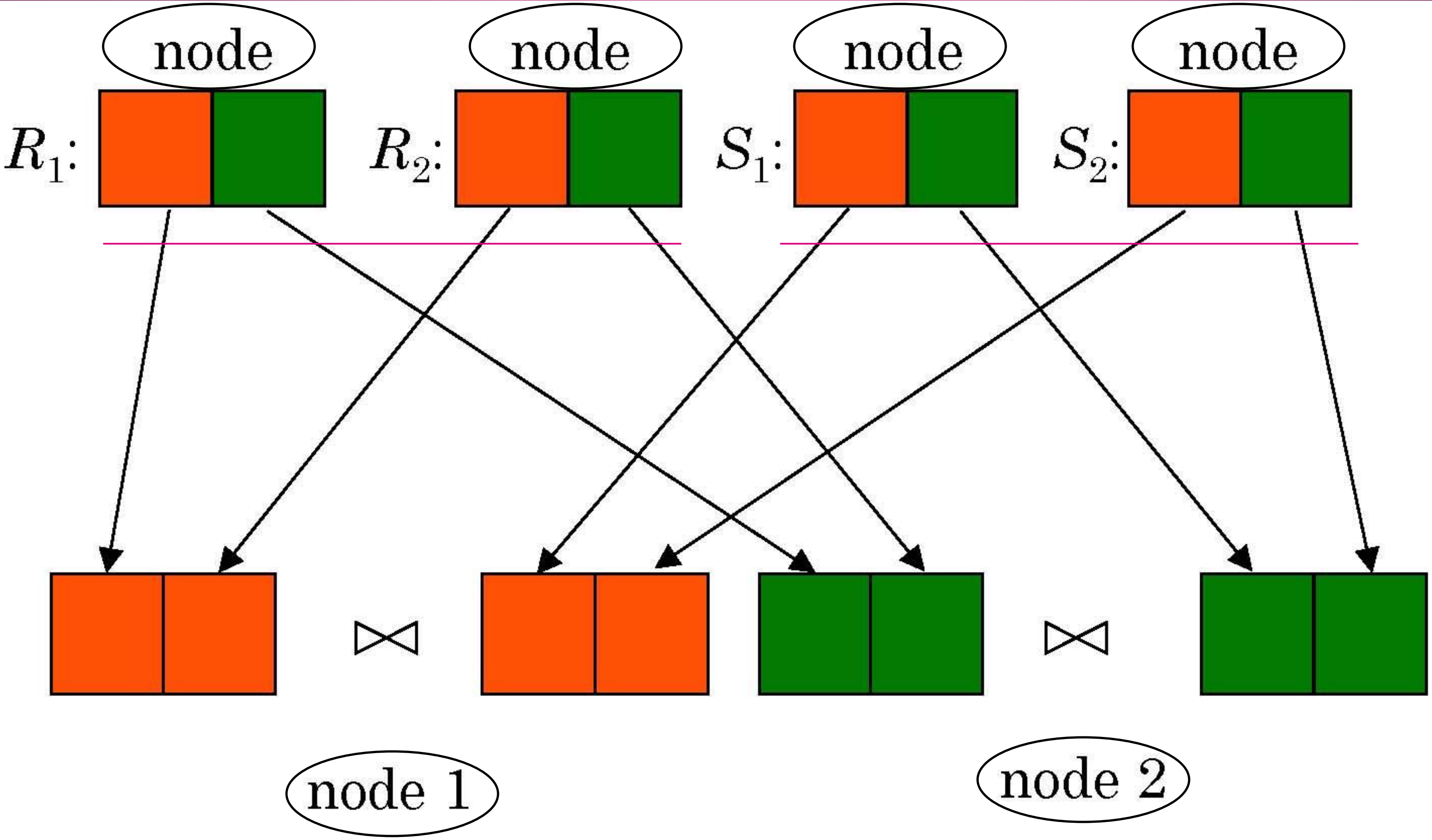
$$R \bowtie S \rightarrow \cup_{i=1,n}(R \bowtie S_i)$$

Parallel Associative Join



$$R \bowtie S \rightarrow \cup_{i=1,n}(R_i \bowtie S_i)$$

Parallel Hash Join



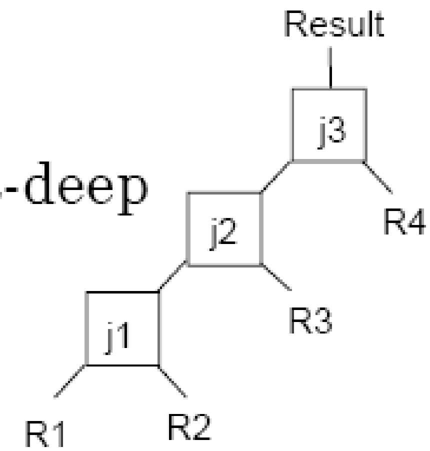
$$R \bowtie S \rightarrow \cup_{i=1,P}(R_i \bowtie S_i)$$

Parallel Query Optimization

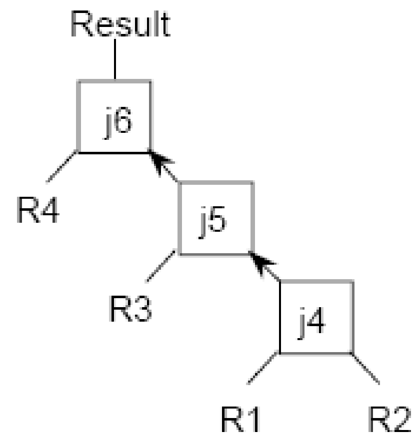
- The objective is to select the "best" parallel execution plan for a query using the following components
- Search space
  - Model alternative execution plans as operator trees
  - Left-deep vs. Right-deep vs. Bushy trees
- Search strategy
  - Dynamic programming for small search space
  - Randomized for large search space
- Cost model (abstraction of execution system)
  - Physical schema info. (partitioning, indexes, etc.)
  - Statistics and cost functions

Execution Plans as Operators Trees

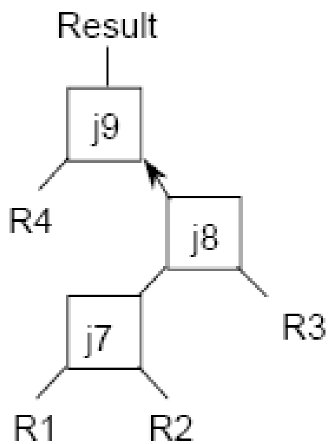
Left-deep



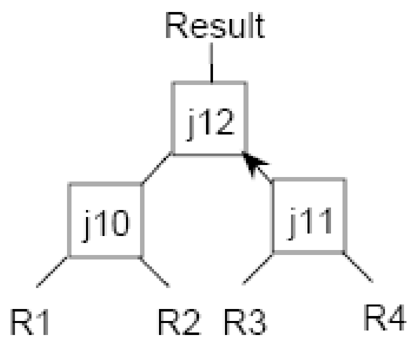
Right-deep



Zig-zag



Bushy



Load Balancing

- Problems arise for intra-operator parallelism with skewed data distributions
  - attribute data skew (AVS)
  - tuple placement skew (TPS)
  - selectivity skew (SS)
  - redistribution skew (RS)
  - join product skew (JPS)
- Solutions
  - sophisticated parallel algorithms that deal with skew
  - dynamic processor allocation (at execution time)

Some Parallel DBMSs

- Prototypes
  - EDS and DBS3 (ESPRIT)
  - Gamma (U. of Wisconsin)
  - Bubba (MCC, Austin, Texas)
  - XPRS (U. of Berkeley)
  - GRACE (U. of Tokyo)
- Products
  - Teradata (NCR)
  - NonStopSQL (Tandem-Compac)
  - DB2 (IBM), Oracle, Informix, Ingres, Navigator (Sybase) ...

Research Problems

- Hybrid architectures
- OS support: using micro-kernels
- Benchmarks to stress speedup and scale up under mixed workloads
- Data placement to deal with skewed data distribution and data replication
- Parallel data languages to specify independent and pipelined parallelism
- Parallel query optimization to deal with mix of precompiled queries and complex ad-hoc queries
- Support of higher functionality such as rules and objects

11. Streaming Data Management

Chapter 18

Streaming Data

Finding a Database Problem

- Pick a simple but fundamental assumption underlying traditional database systems
  - Drop it

- Reconsider all aspects of data management and query processing
  - Many Ph.D. theses
  - Prototype from scratch

Facts

- Dropped assumptions
  - Data has a fixed schema declared in advance
  - All data is accurate, consistent, and complete
  - First load data, then index it, then run queries
    - Continuous data streams
    - Continuous queries

Streaming Data

- Continuous, unbounded, rapid, time-varying streams of data elements
- Occurring in a variety of modern applications
  - Network monitoring and traffic engineering
  - Sensor networks, RFID tags
  - Telecom call records
  - Financial applications
  - Web logs and click-streams
  - Manufacturing processes
- DSMS = Data Stream Management System

DBMS versus DSMS

- Persistent relations
- One-time queries
- Random access
- Access plan determined by query processor and physical DB design

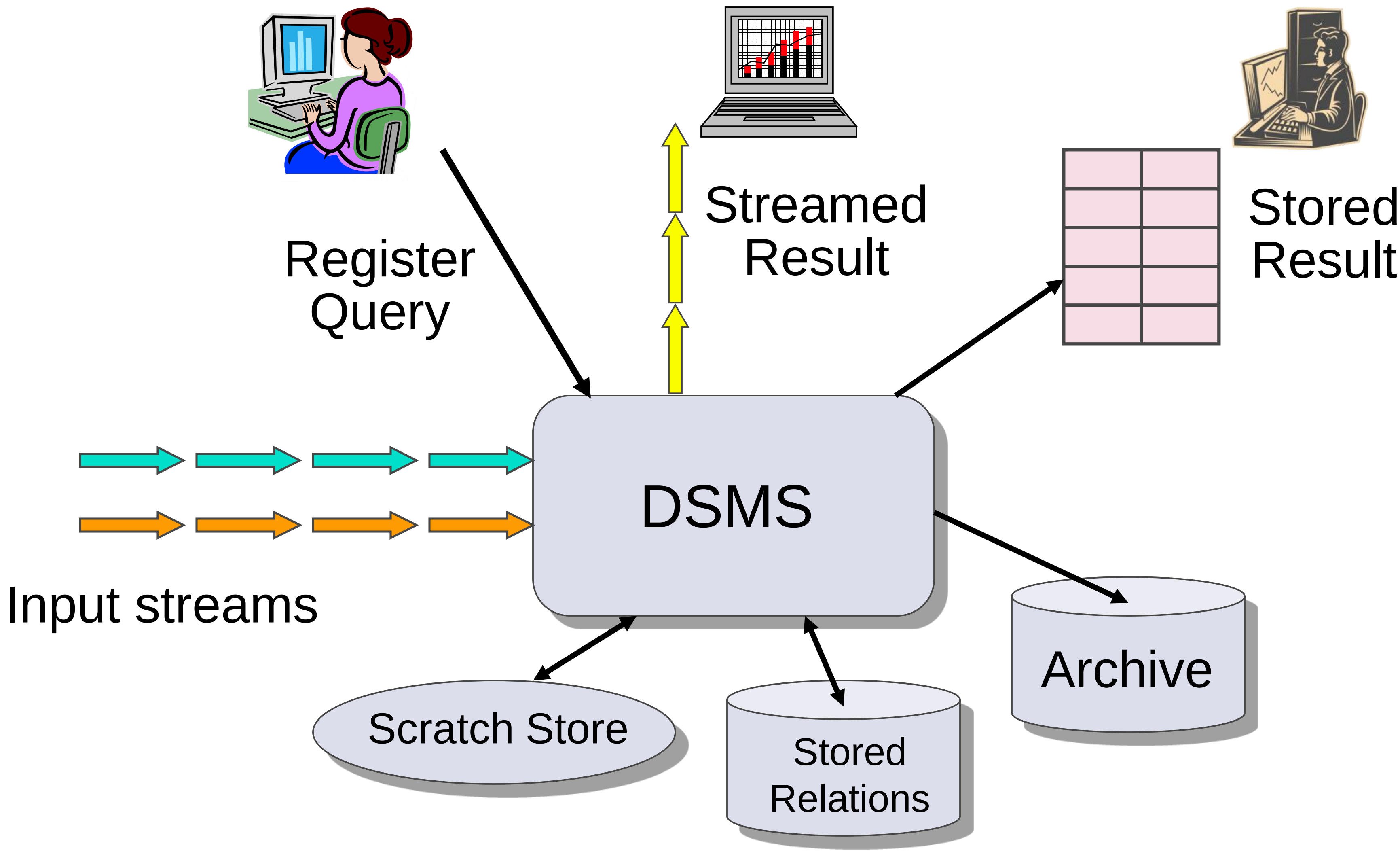
- Transient streams (and persistent relations)
- Continuous queries
- Sequential access
- Unpredictable data characteristics and arrival patterns

Continuous Queries

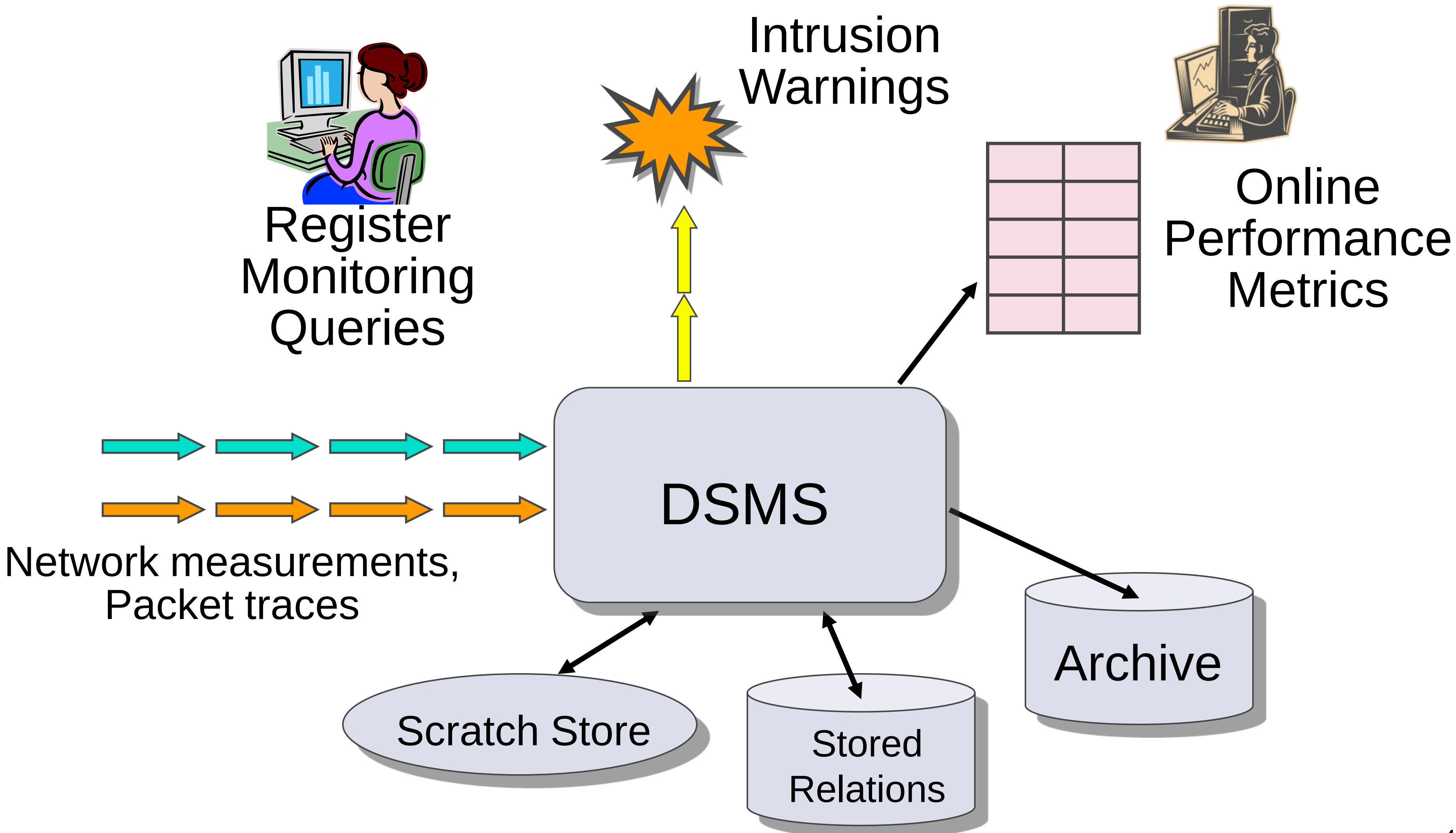
- One time queries – run once to completion over the current data set.

- Continuous queries – issued once and continuously evaluated over the data, e.g.,
  - Notify me when the temperature drops below X
  - Tell me when prices of stock Y > 300

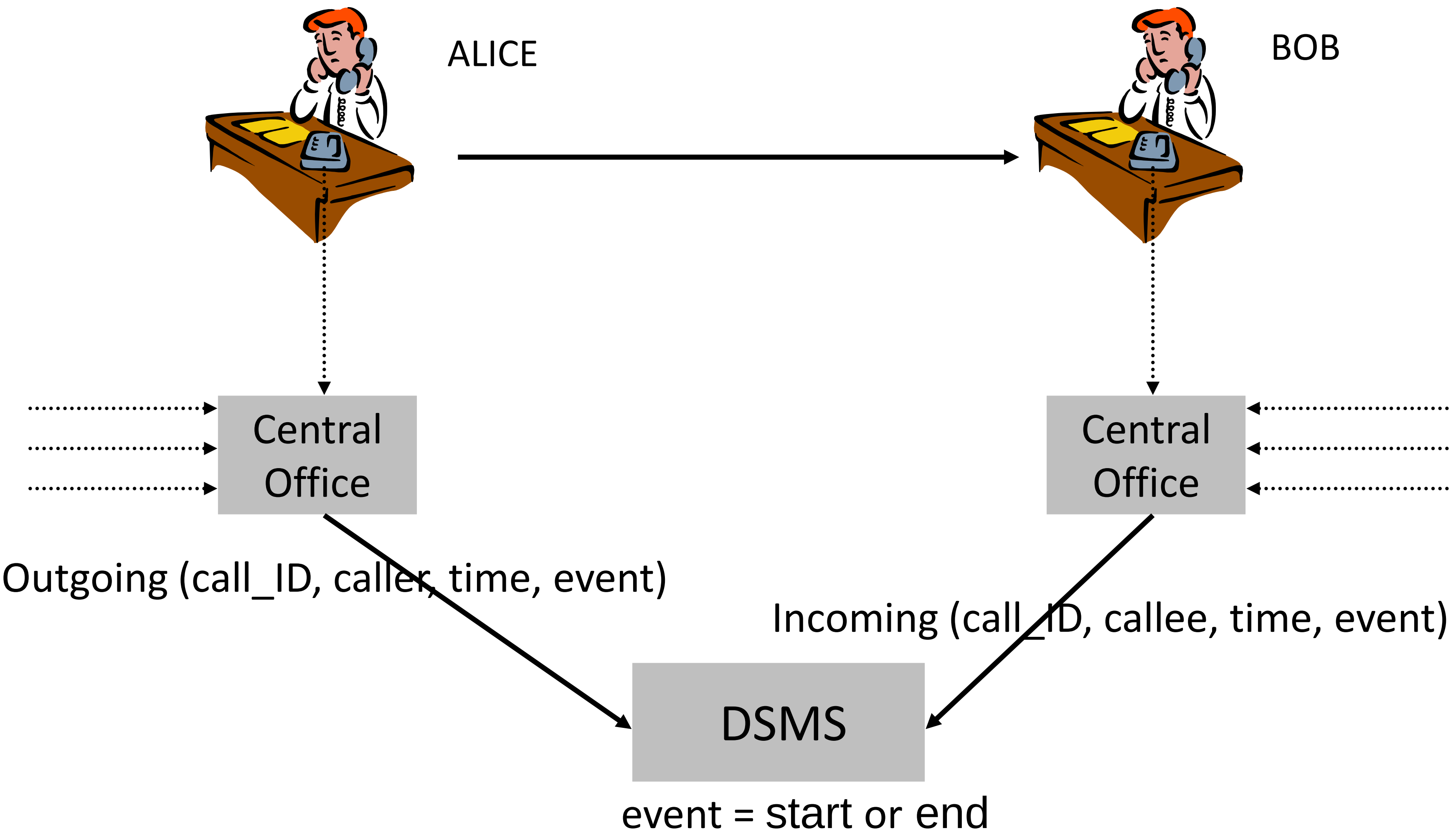
The (Simplified) Big Picture



(Simplified) Network Monitoring



Making Things Concrete



Query 1 (SELF-JOIN)

- Find all outgoing calls longer than 2 minutes

```
SELECT   O1.call_ID, O1.caller
FROM     Outgoing O1, Outgoing O2
WHERE    (O2.time – O1.time > 2)
              AND   O1.call_ID = O2.call_ID
              AND   O1.event = start
               AND   O2.event = end )
```

- Result requires unbounded storage
- Can provide result as data stream
- Can output after 2 min, without seeing end

Query 2 (JOIN)

- Pair up callers and callees

```
SELECT   O.caller, I.callee
FROM     Outgoing O, Incoming I
WHERE    O.call_ID = I.call_ID
```

- Can still provide result as data stream
- Requires unbounded temporary storage …

Query 3 (group-by aggregation)

- Total connection time for each caller

```
SELECT      O1.caller, sum(O2.time – O1.time)
FROM        Outgoing O1, Outgoing O2
WHERE       (O1.call_ID = O2.call_ID
             AND O1.event = start
             AND  O2.event = end )
GROUP BY    O1.caller
```

- Cannot provide result in (append-only) stream
  - Output updates?
  - Provide current value on demand?
  - Memory?

DSMS – Architecture & Issues

- Data streams and stored relations – architectural differences
- Declarative language for registering continuous queries
- Flexible query plans and execution strategies
- Centralized ? Distributed ?

DSMS – Options

- Relation: Tuple Set or Sequence?
- Update: Modification or Append?
- Query Answer: Exact or Approximate?
- Query Evaluation: One or Multiple Pass?
- Query Plan: Fixed or Adaptive?

Architectural Comparison

DSMS

- Resource (memory, per-tuple computation) limited
- Reasonably complex, near real time, query processing
- Useful to identify what data to populate in database
- Query evaluation: one pass
- Query plan: adaptive

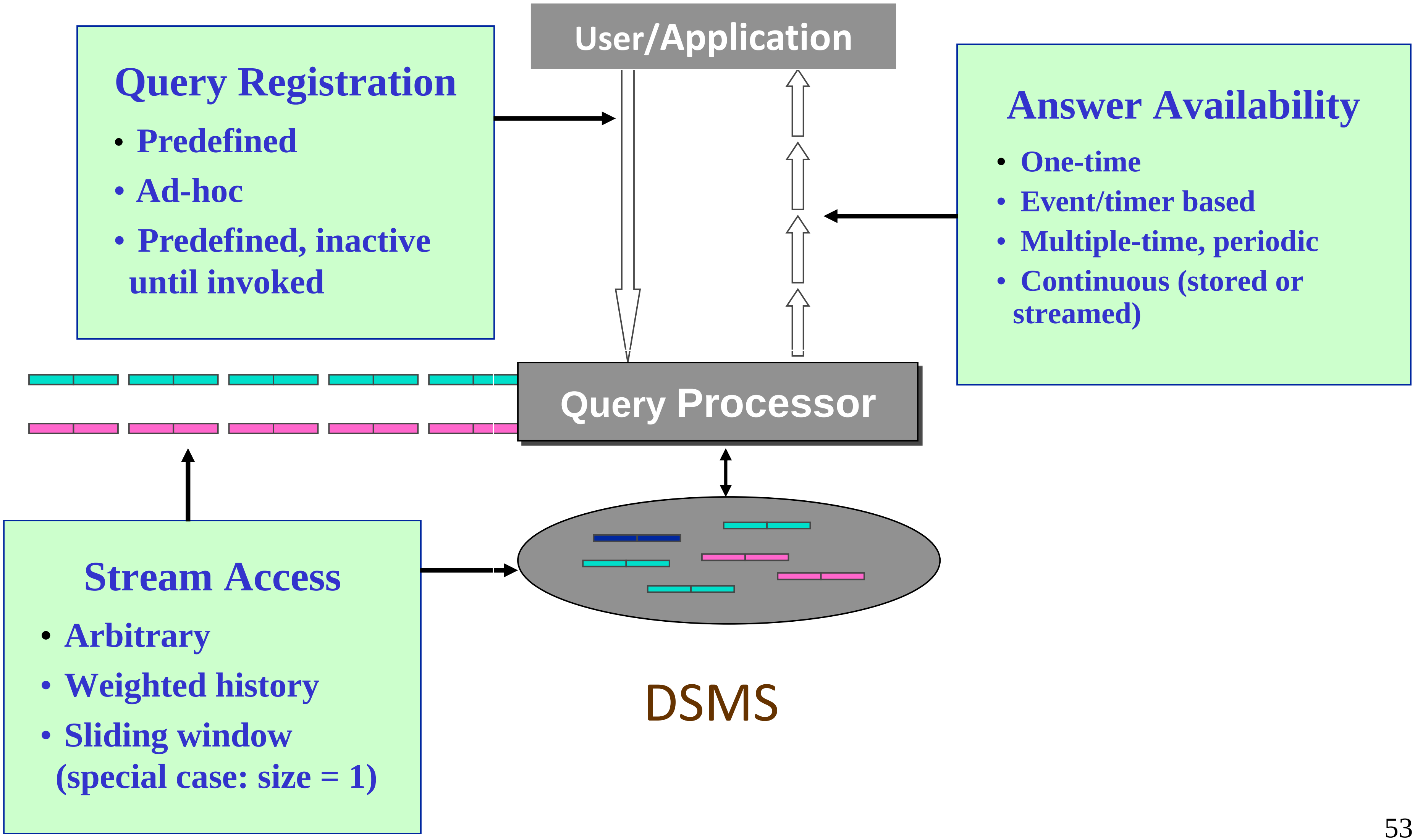
DBMS

- Resource (memory, disk, per-tuple computation) rich
- Extremely sophisticated query processing, analysis
- Useful to audit query results of database systems
- Query evaluation: arbitrary
- Query plan: fixed

DSMS Challenges

- Must cope with:
  - **Stream rates** that may be high, variable, bursty
  - **Stream data** that may be unpredictable, variable
  - **Continuous query loads** that may be high, variable
- Overload – need to use resources very carefully
- Changing conditions – adaptive strategy

Query Model



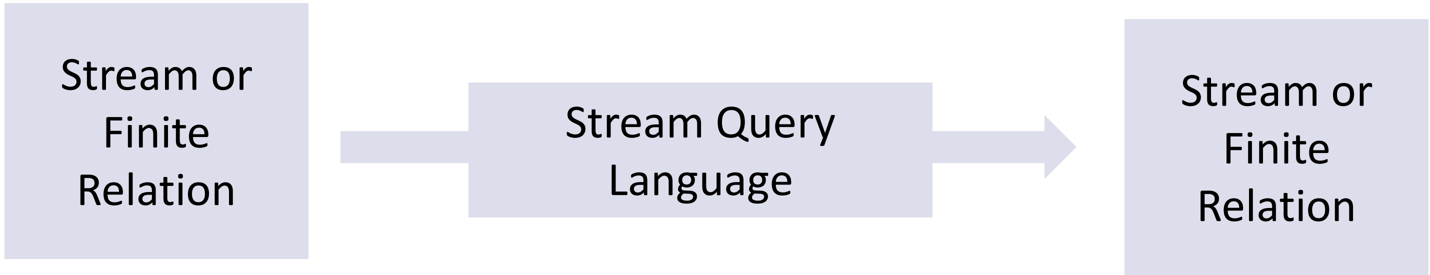
User/Application
Query Registration
• Predefined
• Ad-hoc
• Predefined, inactive until invoked
Answer Availability
• One-time
• Event/timer based
• Multiple-time, periodic
• Continuous (stored or streamed)
Query Processor
Stream Access
• Arbitrary
• Weighted history
• Sliding window (special case: size = 1)
DSMS

Query Processing

- Query Language
- Operators
- Optimization
- Multi-Query Optimization

Stream Query Language

- SQL extension
- Queries reference/produce relations or streams
- Examples: GSQL, CQL



Continuous Query Language – CQL

Start with SQL

Then add…

- Streams as new data type
- Continuous instead of one-time semantics
- Windows on streams (derived from SQL-99)
- Sampling on streams (basic)

Impact of Limited Memory

- Continuous streams grow unboundedly
- Queries may require unbounded memory
- One solution: Approximate query evaluation

Approximate Query Evaluation

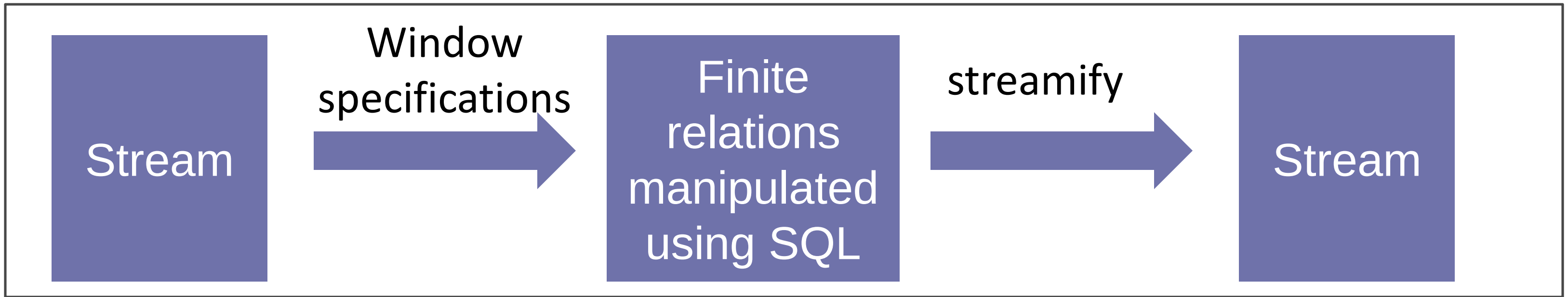
- Why?
  - Handling load – streams coming too fast
  - Avoid unbounded storage and computation
  - Ad hoc queries need approximate history
- How?
  - Sliding windows, synopsis, samples, load-shedding

Approximate Query Evaluation (*cont.*)

- Major Issues
  - Metric for set-valued queries
  - Composition of approximate operators
  - How is it understood/controlled by user?
  - Integrate into query language
  - Query planning and interaction with resource allocation
  - **Accuracy-efficiency-storage tradeoff and global metric**

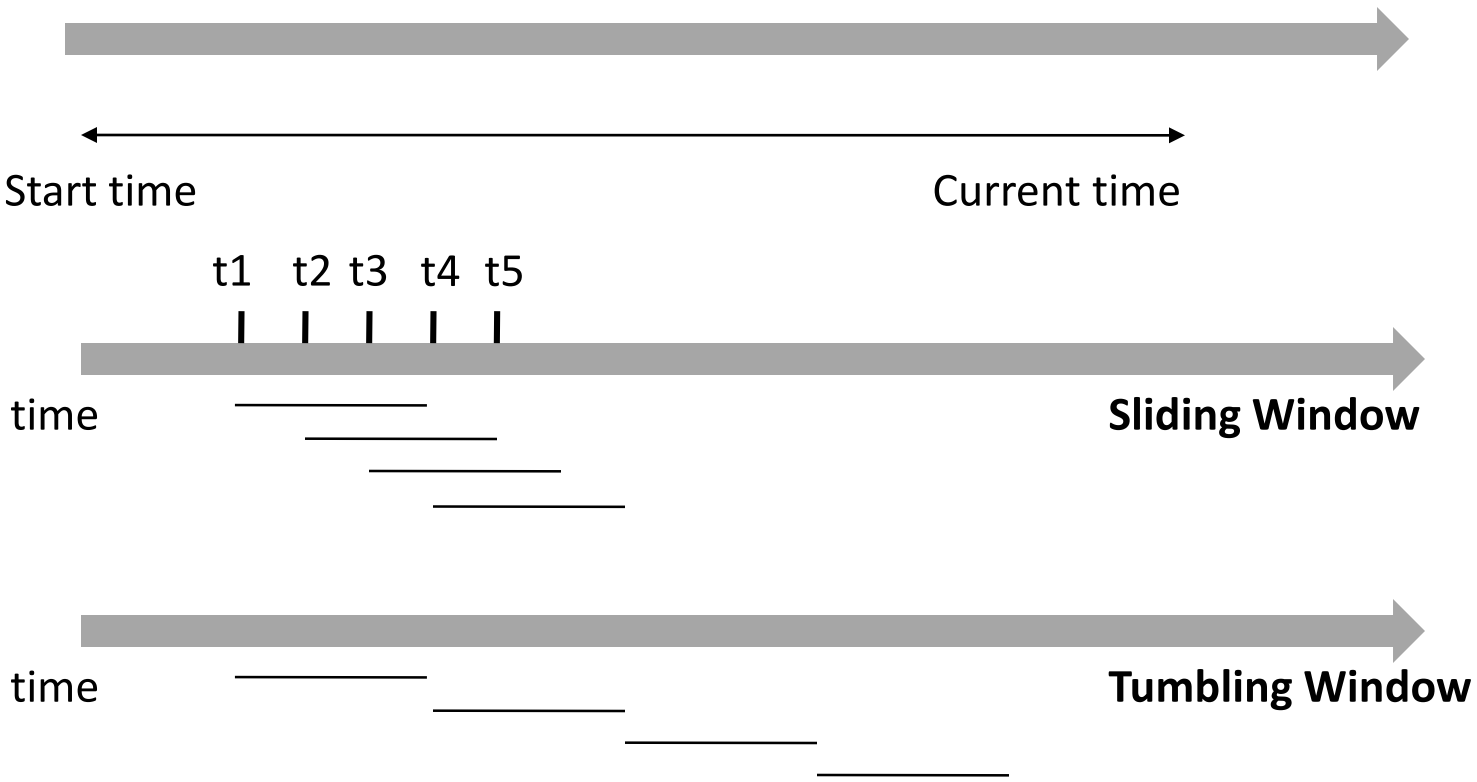
Windows

- Mechanism for extracting a finite relation from an infinite stream
- Various window proposals for restricting operator scope
  - Windows based on ordering attribute (e.g., time)
  - Windows based on tuple counts
  - Windows based on explicit markers (e.g. punctuations)
  - Variants (e.g., partitioning tuples in a window)



Windows (*cont.*)

- Terminology



Query Operators

- Selection - Where clause
- Projection - Select clause
- Join - From clause
- Group-by (Aggregation) – Group-by clause

Query Operators (*cont*.)

- Selection and projection on streams - straightforward
  - Local per-element operators
- Project may need to include ordering attribute
- Join – Problematic
  - May need to join tuples that are arbitrarily far apart
  - Equijoin on stream ordering attributes may be tractable
- Majority of the work focuses on join using windows.

Blocking Operators

- Blocking
  - No output until entire input seen
  - Streams – input never ends
- Simple Aggregate – output “update” stream
- Set Output (sort, group-by)
  - Root – could maintain output data structure
  - Intermediate nodes – try non-blocking analogs
- Join
  - Apply sliding-window restrictions

Optimization in DSMS

- Traditionally table-based cardinalities used in query optimizer.
  - Goal of query optimizer: Minimize the size of intermediate results.
- Problematic in a streaming environment – All streams are unbounded = infinite size!
- Need novel optimization objectives that are relevant when the input sources are streams.

Query Optimization in DSMS

- Novel notions of optimization:
  - Stream rate based [e.g. NiagaraCQ]
  - QoS based [e.g. Aurora]
- Continuous adaptive optimization
- Possibilities that objectives cannot be met:
  - Resource constraints
  - Bursty arrivals under limited processing capabilities

Typical Stream Projects

- Amazon/Cougar (Cornell) – sensors
- Aurora (Brown/MIT) – sensor monitoring, dataflow
- Hancock (AT&T) – telecom streams
- Niagara (OGI/Wisconsin) – Internet XML databases
- OpenCQ (Georgia) – triggers, incr. view maintenance
- Stream (Stanford) – general-purpose DSMS
- Tapestry (Xerox) – pub/sub content-based filtering
- Telegraph (Berkeley) – adaptive engine for sensors
- Tribeca (Bellcore) – network monitoring
- ……

Conclusion

- Conventional DMS technology is inadequate.
- We need to reconsider all aspects of data management in presence of streaming data.

Question & Answer