CSE 541: Database Systems I

Main-Memory Database Overview

Slides Credit: Tianzheng Wang

What we have assumed so far

- Data is storage-resident
- I/O happens frequently during transaction processing
- Main memory (DRAM)
  - Much smaller than storage (e.g., disk or flash)
    - Expensive!
  - Too small to store all data or the working set
  - In essence main memory is a cache

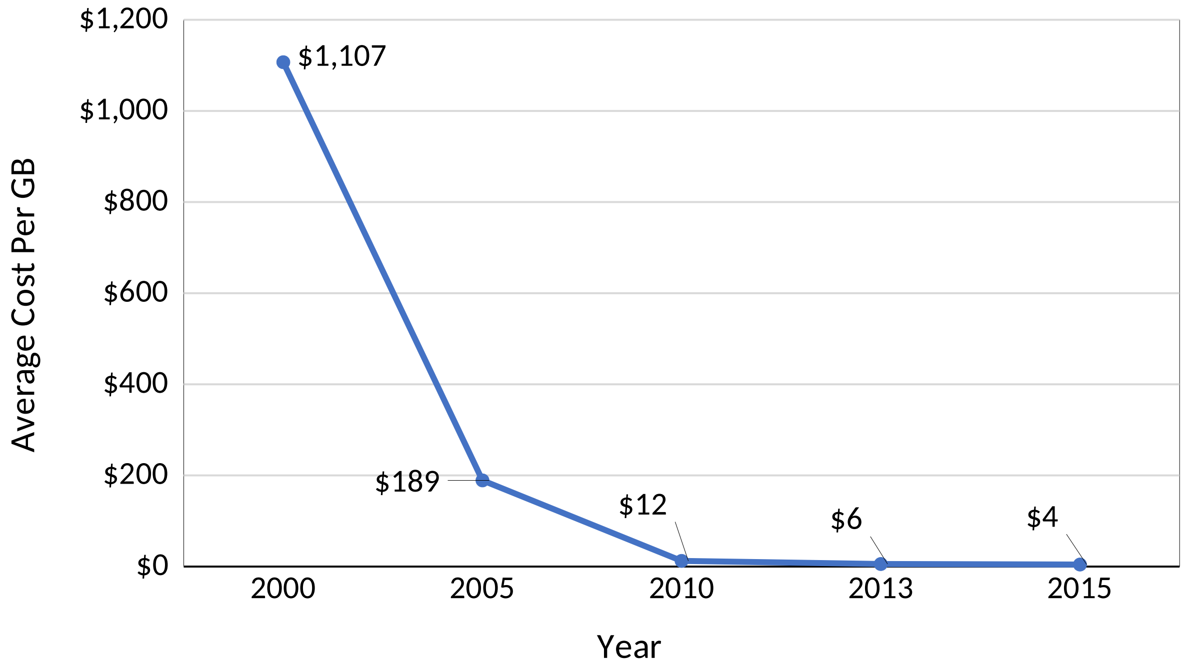
Hardware is quickly changing to (partially) invalidate these assumptions are quickly

- For certain workloads and scenarios only.

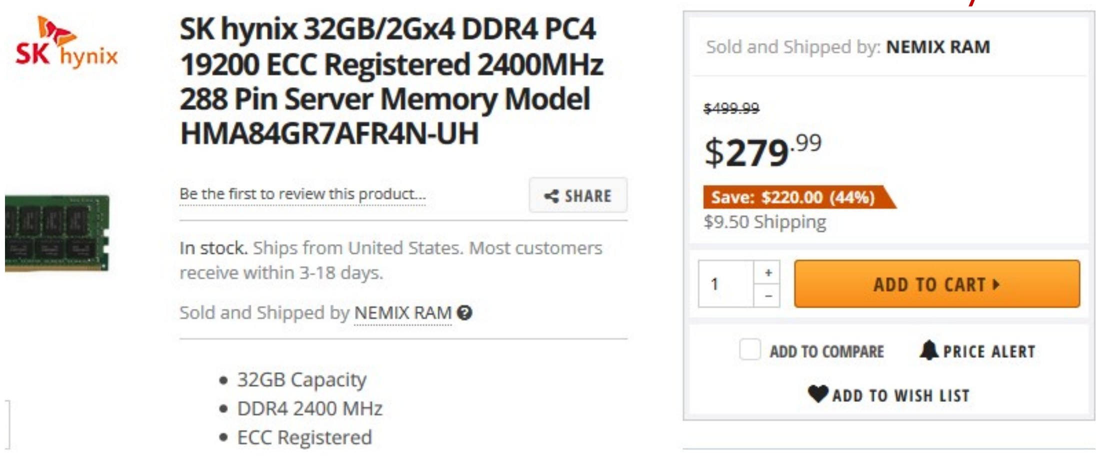
Hardware Trends

Big memory:

- Enabled by low memory price – significant fall since post 2000
- Unit price: ~$4-10per GB
- Possible to fit the entire database or at least working set in memory
  - No storage access needed during transaction processing



As of May 2019:

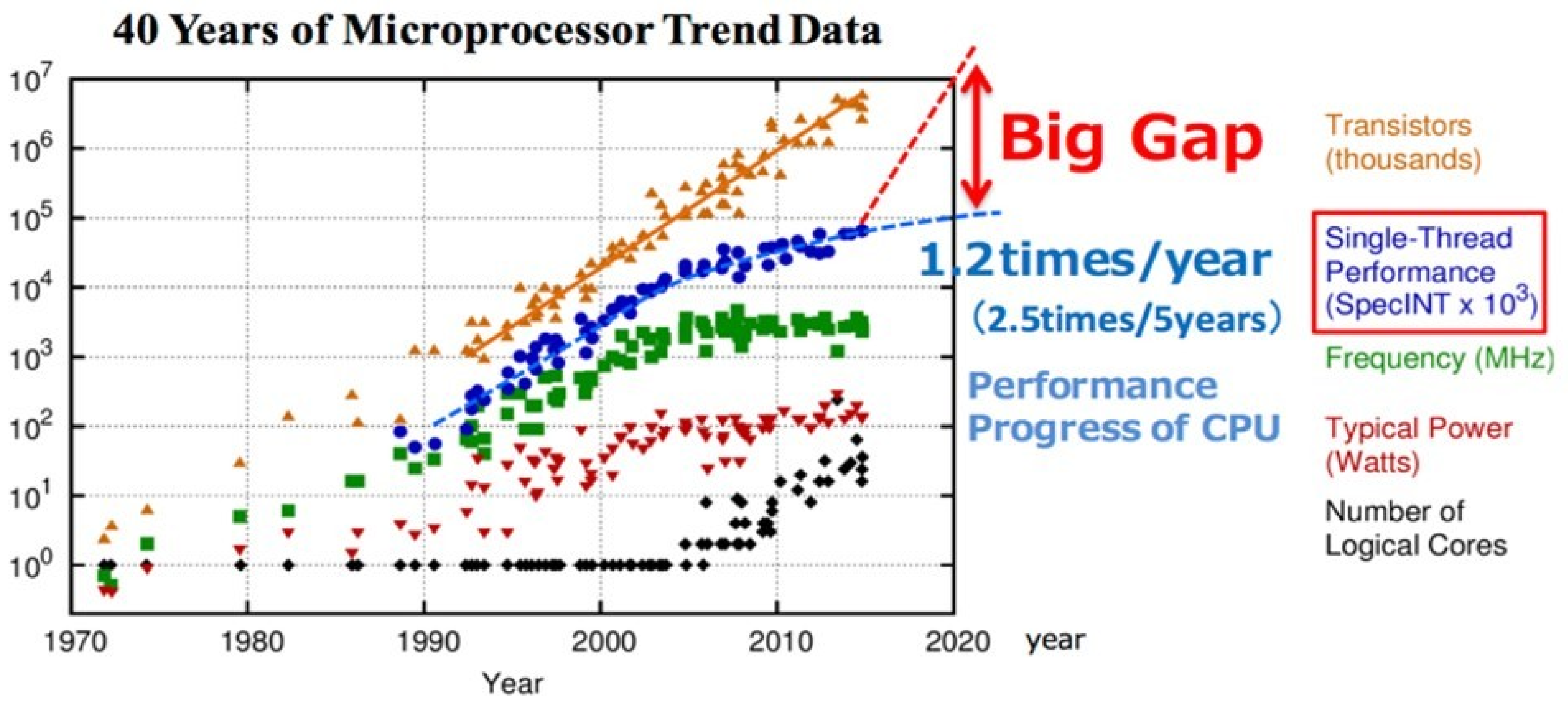


* Modern Main-Memory Database Systems, Paul Larson and Justin Levandoski, VLDB 2016 Tutorial

Hardware Trends

Massively parallelism (CPU):

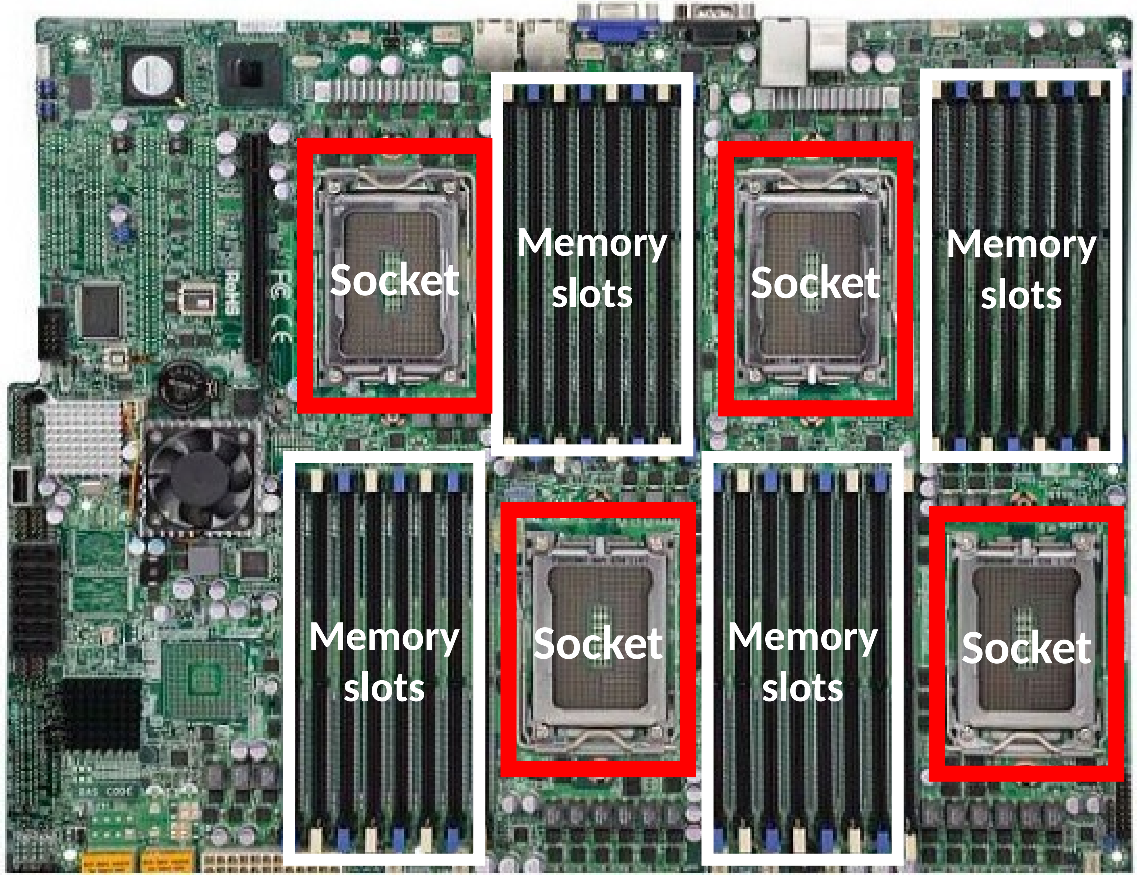
- Multi-core CPUs
  - Clock speed not increasing much, but number of cores is increasing per chip



Hardware Trends

Massively parallelism (CPU):

- Multiple processors in each server



Socket to place a multicore processor

Usually 4-32 cores per processor/socket

Mainstream database servers: 2 – 4 sockets

Very high-end/mission critical: 8-32 sockets

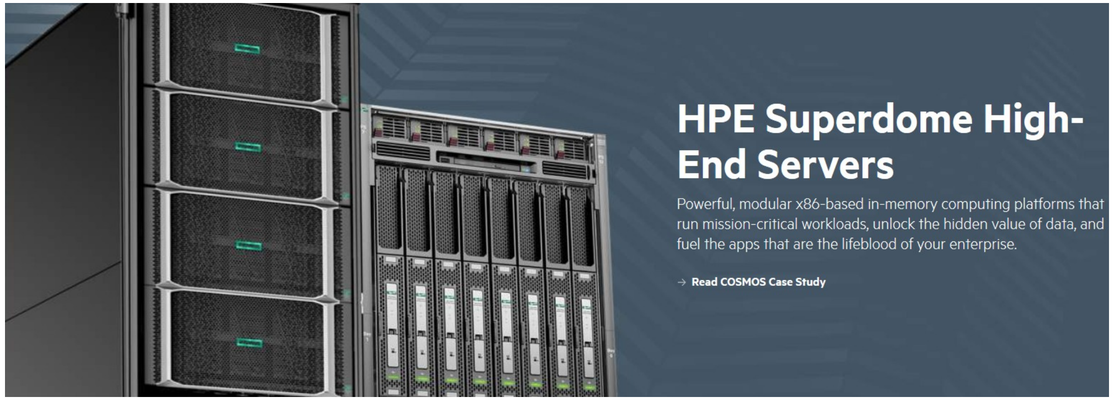
Hardware Trends

Modern (mid-high) servers can have:

- Big memory: 100s of GB or terabytes
- Massive parallelism: 10s or 100s of CPU cores with multi-core CPUs and multi-socket

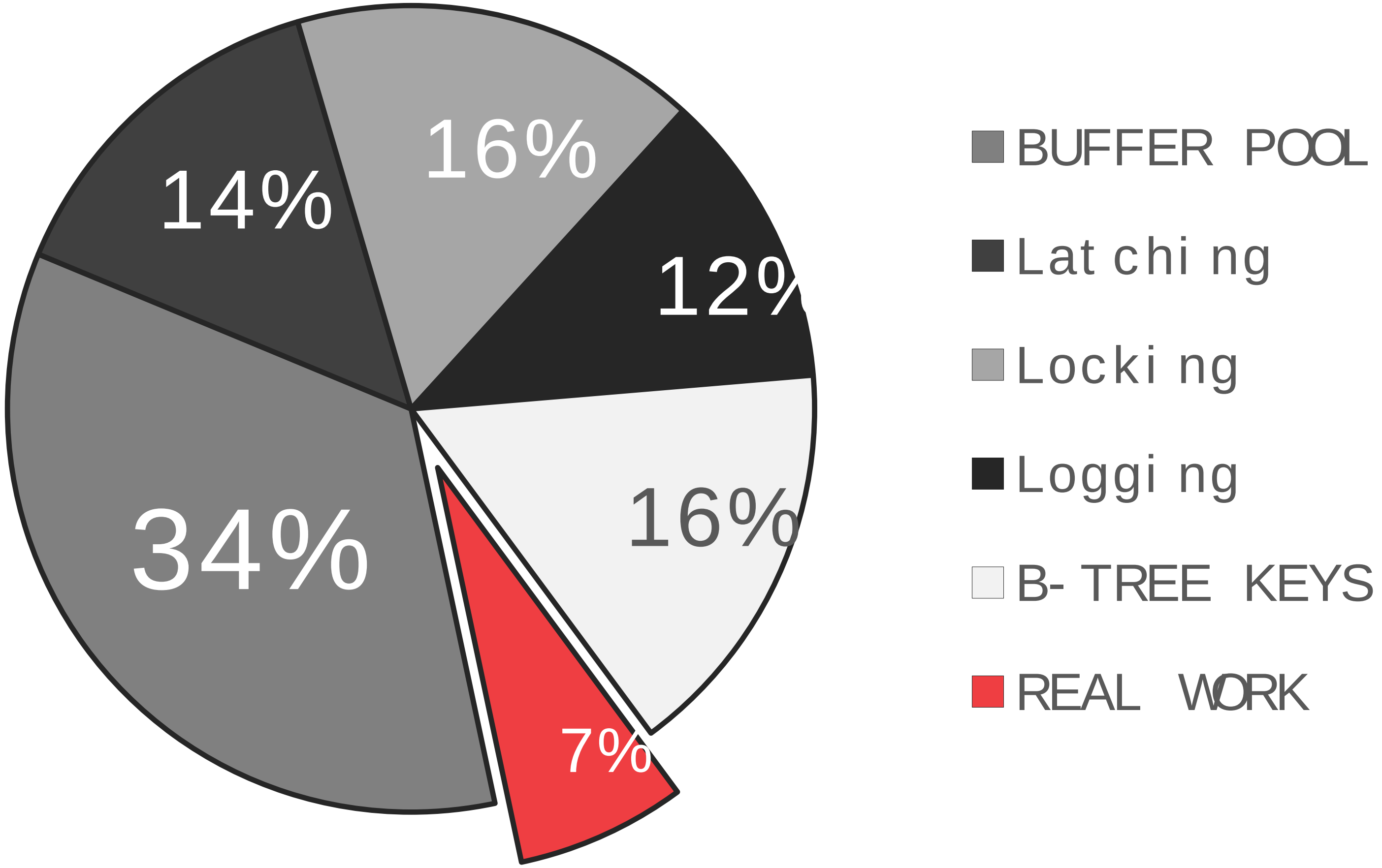
Example: HPE Superdome X

- 288 cores (16 sockets x 18 cores each), 576 hardware threads
- 12 TB memory



Very different hardware assumption ➔ New designs required

Disk-Oriented DBMS Overhead



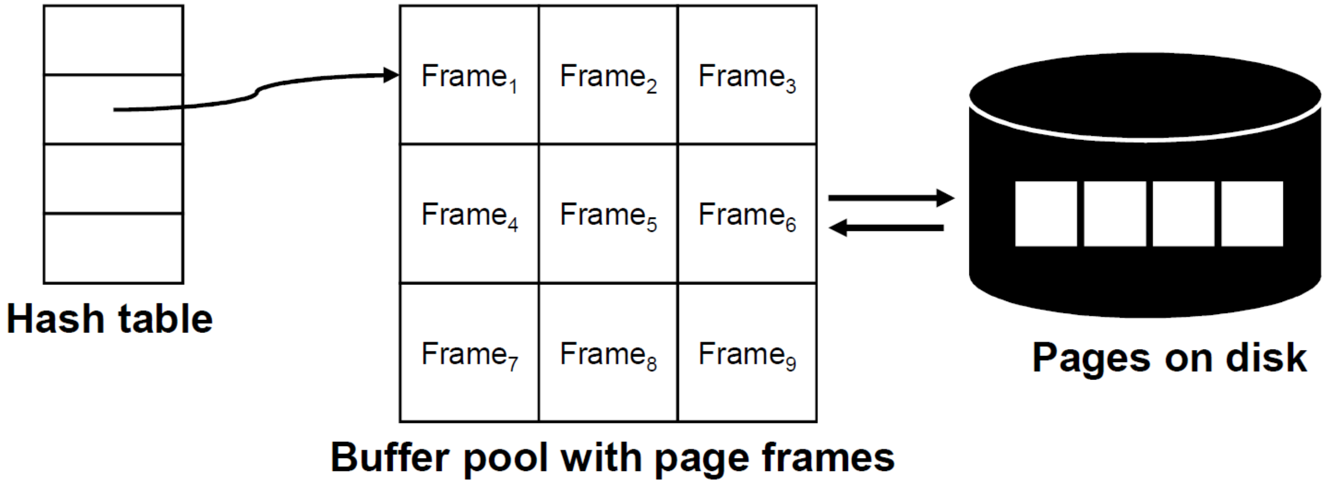
Modern Main-Memory DMBS Design

Fundamentally different from traditional DBMS in terms of:

- Data organization and indexing
- Concurrency Control
- Durability and recovery
- Query processing and compilation

Data Organization in Traditional Systems

Hash table indirection + buffer pool



- Hash table maps page IDs to locations in the buffer pool
- Fixed-size pages, fetch pages from storage on demand
- Too heavyweight for main memory systems
  - **Relative** overhead much larger
  - In traditional systems, I/O is often the largest bottleneck

Data Organization in MMDBMS

Entire database (or at least working set) lives in memory

No longer used:

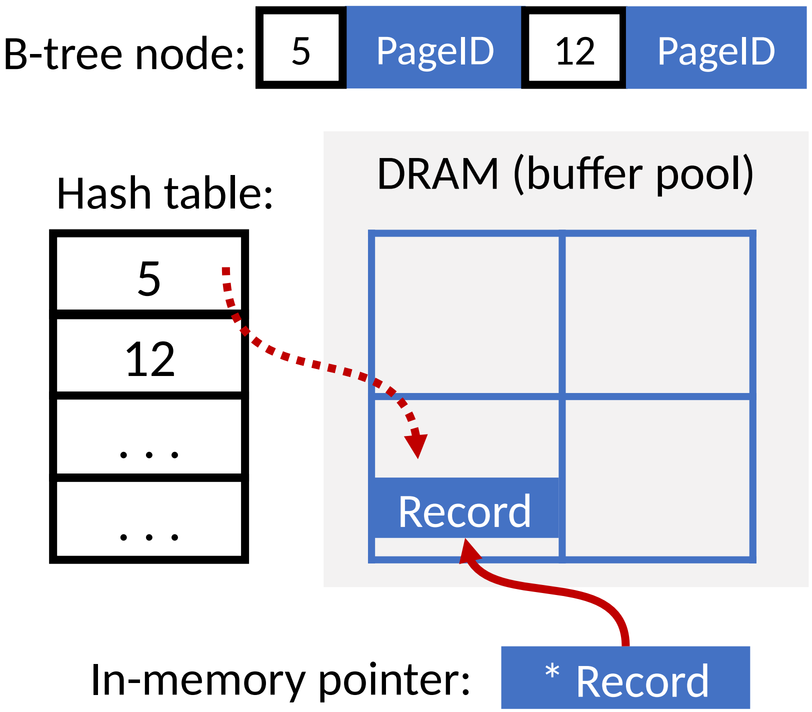
- Paging in/out during transaction processing
- Buffer pool
- Hash table indirection
- Latching
  - Instead, use lock-free techniques
- Physical record IDs <Page ID, slot number>
  - Instead, use direct virtual memory pointers for records
  - May or may not have a “page” concept

➔ Orders of magnitude better performance

Avoid Paging Indirection

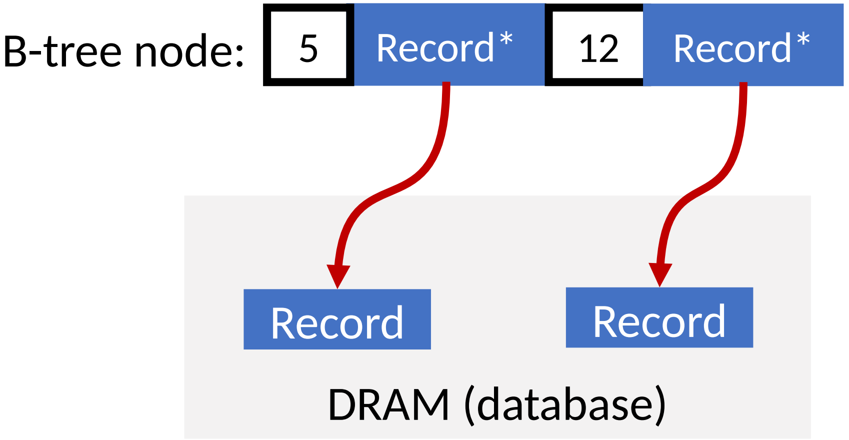
Traditional system:

- Two-level indirection
- First find page frame
- Then calculate pointer to record



Main-memory system:

- No indirection
- Memory pointer stored in leaf nodes



Data Organization Choices

Partitioning:

- Whether to physically partition the database
- Typically at the thread level: each thread only allowed to touch a partition
- Partitioned (shared-nothing)
  - Simplifies concurrency control – no CC even needed within a partition
  - Complex/slow when transactions cross partitions
  - Load balancing issues
- Non-partitioned (shared-everything)
  - Must coordinate and synchronize threads carefully
  - No load balancing issues

Data Organization Choices

Multi-versioning: Readers/writers do not block each other

- Especially important for main-memory systems
- Less context switch overhead (i.e., no sleep on conflict)
- Context switch: a major overhead in main-memory systems
  - **Relative** overhead much larger without storage I/O impact (i.e., it shows up!)

Row/columnar layout:

- Similar to the case for storage-centric systems
- Row format good for OLTP (transactions)
- Columnar format good for OLAP (analytics)

Concurrency Control in Traditional Systems

Two-phase locking + centralized lock manager

- Pessimistic in essence
  - Assuming transactions do conflict with each other
- Locks needed prior to accessing records
- Locks managed centrally in lock manager
  - To handle page in/outs during transaction execution
  - To handle deadlocks
- Lock manager: a centralized component
  - Synchronization needed
  - Each transaction needs roundtrips to the lock manager multiple times
  - Relatively small cost compared to storage I/O

Concurrency Control in MMDBMS

Centralized lock manager avoided

- Embed locking metadata in records
- No need to worry about page in/out
    - Basic assumption is at least the working set fits in memory

Example:

Lock word	Data record
8-byte	Variable-size record

Leveraging partitioning

- Single-threaded, serial execution inside partition
- No concurrency control needed for transactions not crossing partition boundaries ("local transactions")

Concurrency Control in MMDBMS

Apriori knowledge in of footprint

- Full read/write sets available before execution
  - Enables a lot of optimizations
  - Not always possible
  - Many main-memory CC schemes rely on it
- Can be done by static analysis, pre-run, etc.

Optimistic execution and multi-versioning

- Favour multi-versioning for less context switch overhead
- Favour optimistic approaches for low synchronization and coordination overhead

Durability and Recovery

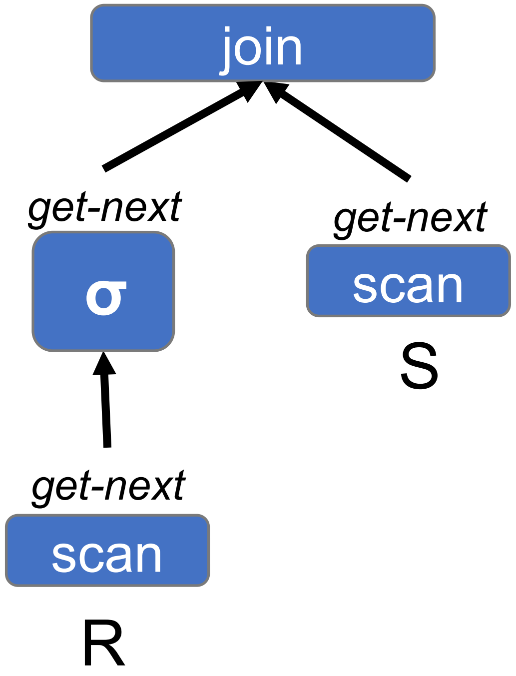
Main-memory database != no durability

- Still need to persist data
  - Mostly in the form of checkpoints, no page in/out
  - Checkpoints must include all rows, but not necessarily index
  - Often larger and more complex than checkpointing in traditional systems
- Read-only logging
  - Significantly simpler and more lightweight
  - Log records for aborted transactions discarded
  - Log only contains records for committed transactions
  - Much smaller log size – no undo images
  - Leverage group commit, commit pipelining to hide I/O’s impact
  - One-pass recovery: redo only
  - Easy to distribute
    - E.g., partition by record ID set – trivial to parallelize recovery

Query Processing and Compilation

- Traditional systems: physical query plans
  - Fixed set of operators
  - Each operator processes tuples using “get-next” interfaces
  - Often virtual functions (or function pointers)

  ➔Extra indirection and may degrade branch prediction performance in modern CPUs



- Main-memory systems: compiled queries
  - Queries compiled as machine code
  - Executed directly
  - E.g., convert to C code then compile
  - Often use compiler frameworks (e.g., LLVM)