Contemporary misconceptions that limit distributed system design and implementation

Database Seminar
July 20th 2011

Andrew Kane <arkane@cs.uwaterloo.ca>

Abstract:

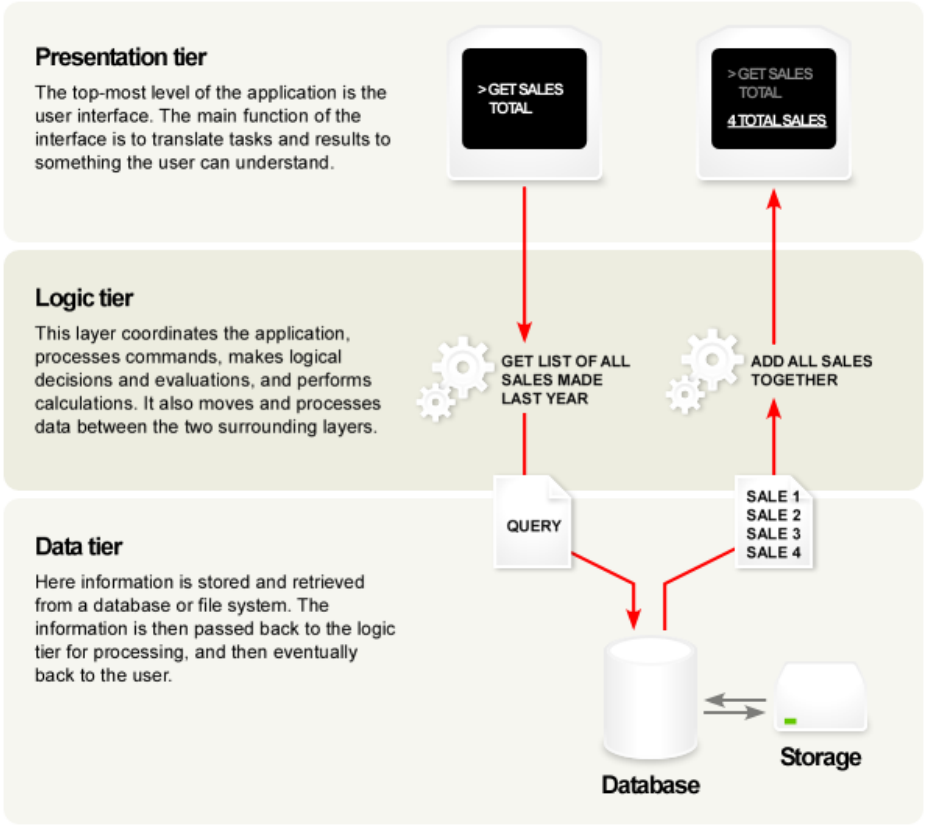
The current practice for distributed systems builds on unwritten assumptions about how to design and implement such systems. I believe that some of these assumptions are limiting potential optimizations, for example, the belief that static compilation is faster than just-in-time compilation leads to query plans that don't change after a query starts, and query executions that don't adapt to data distribution changes across the span of a query. In order to make these ideas more concrete, I will show potential benefits using a three tiered distributed database system as an example. The misconceptions presented are related to machine design, data level distribution, and tiered solutions.

outline

- Three tiered distributed database
- Assumption areas:
  - A) Fundamental optimization approaches
  - B) Machine design
  - C) Architecture of DB level distribution
  - D) Architecture of full software solution
- Possible system design
- Conclusions

three tiered distributed database

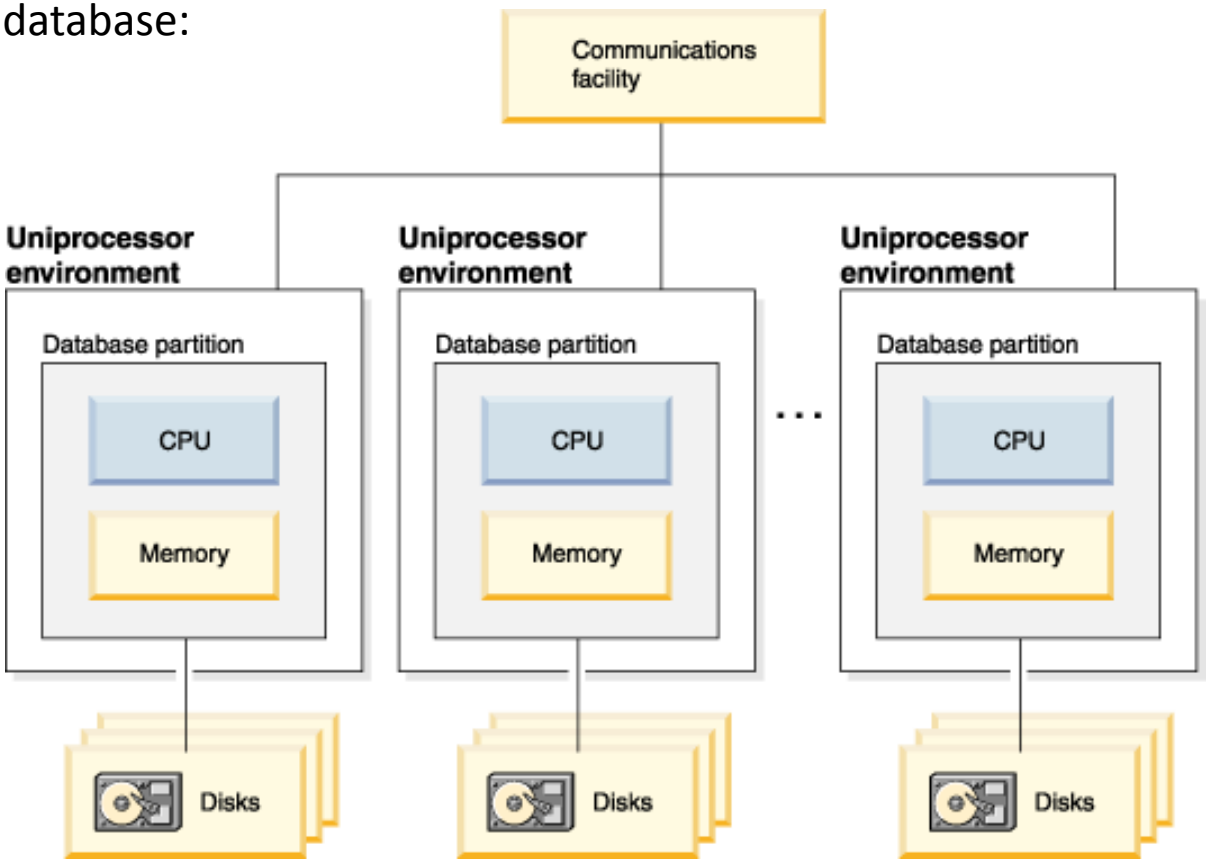
Three tiered system:



From: http://dineshtheblackmamba.wordpress.com/category/uncategorized/

three tiered distributed database

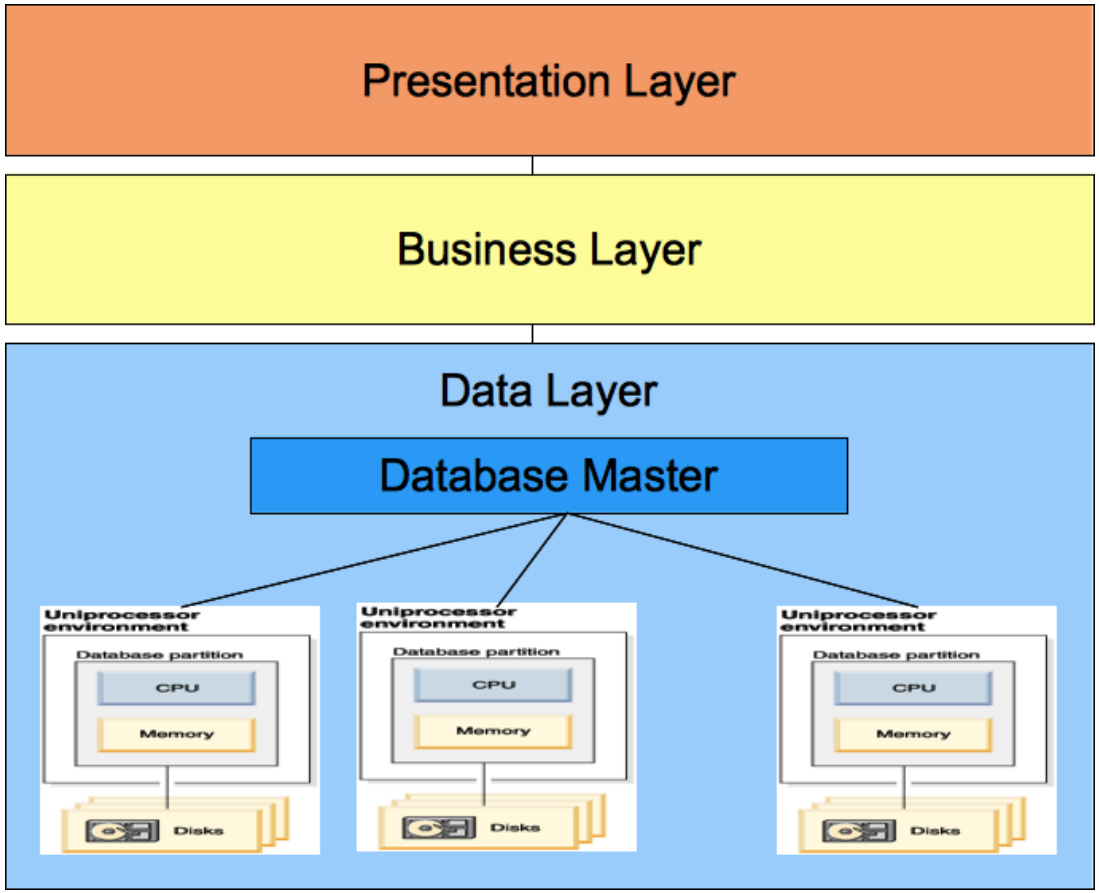
Distributed database:



From: http://publib.boulder.ibm.com/infocenter/db2luw/v9/index.jsp?topic=/com.ibm.db2.udb.admin.doc/doc/c0004569.htm

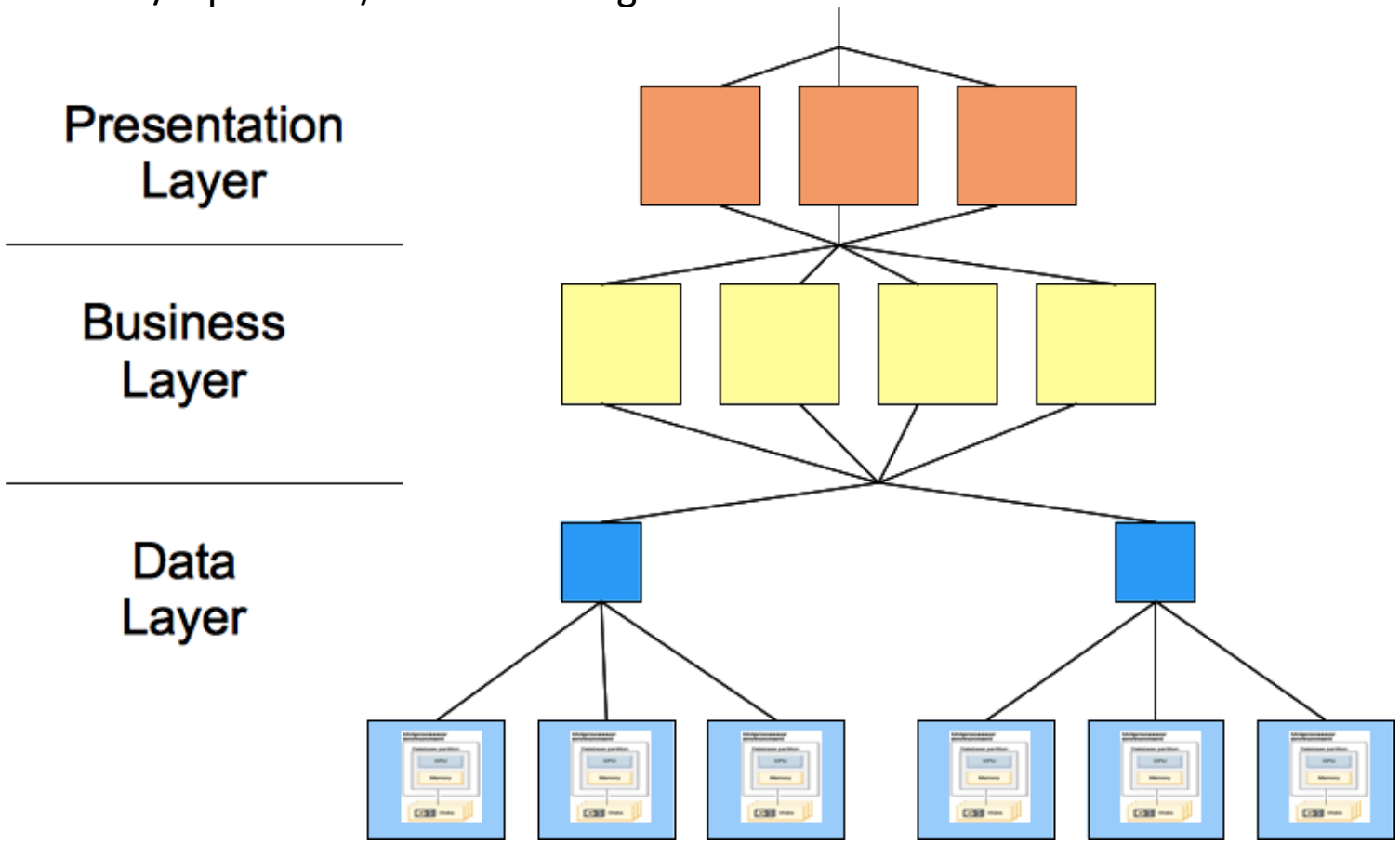
three tiered distributed database

Three tiered distributed database:



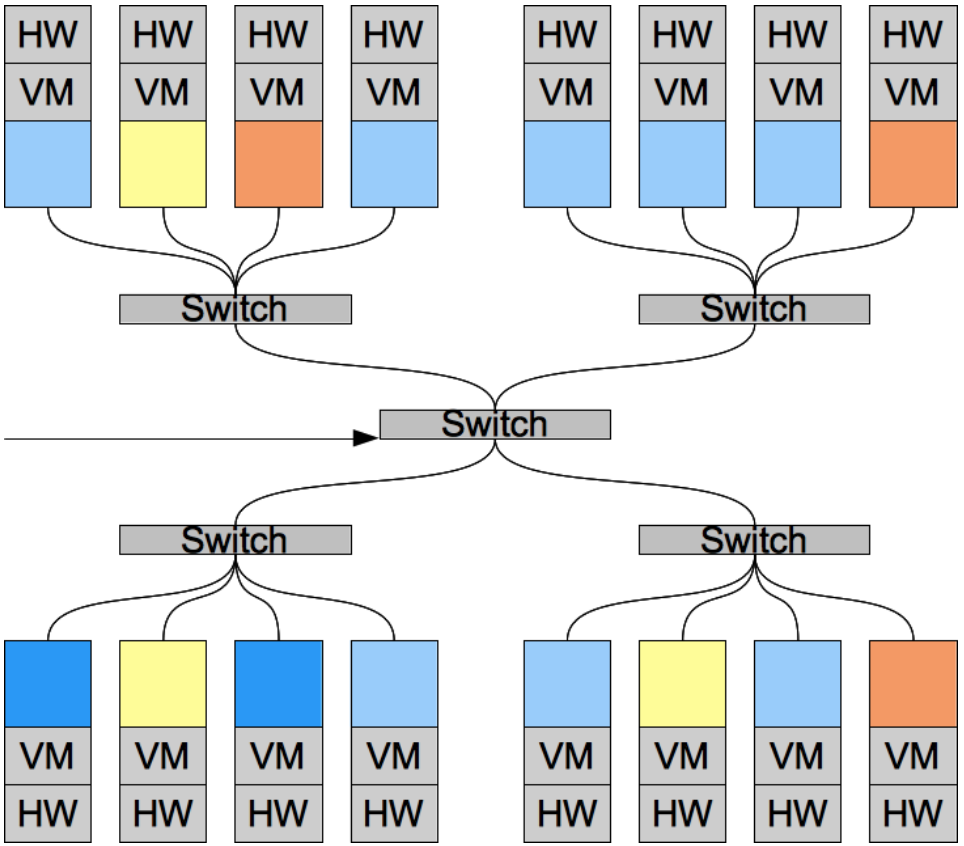
three tiered distributed database

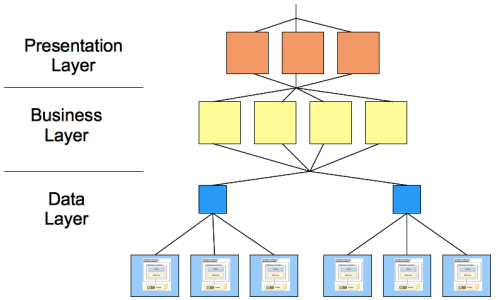
Add distribution/replication/load balancing:



three tiered distributed database

Deploy with VM over cluster:





three tiered distributed database

- **Who decides on process/VM location in cluster?**
- **What happens when a process/VM moves?**
- **How do you optimize such a system?**

outline

- Three tiered distributed database
- Assumption areas:
  - A) Fundamental optimization approaches
  - B) Machine design
  - C) Architecture of DB level distribution
  - D) Architecture of full software solution
- Possible system design
- Conclusions

A) fundamental optimization approaches

- belief: static optimization is better than dynamic optimization (e.g. C++ > Java)

- GoogleTechTalks: "A JVM Does That?" by Cliff Click

- Just-In-Time (JIT) compiler can do optimizations static compilers can't:
  - Reorder operations dependent on usage/data
    - e.g. if statement reordering
  - Remove dynamic linking
    - e.g. virtual calls if only one object type exists, dynamic library inlining
  - Hardware specific optimizations
    - e.g. take advantage of extra CPU commands, or GPU
  - Resource usage based optimizations
    - e.g. add compression if data movement is bottleneck

- Are there any query engines that use JIT for execution? … No.

A) fundamental optimization approaches

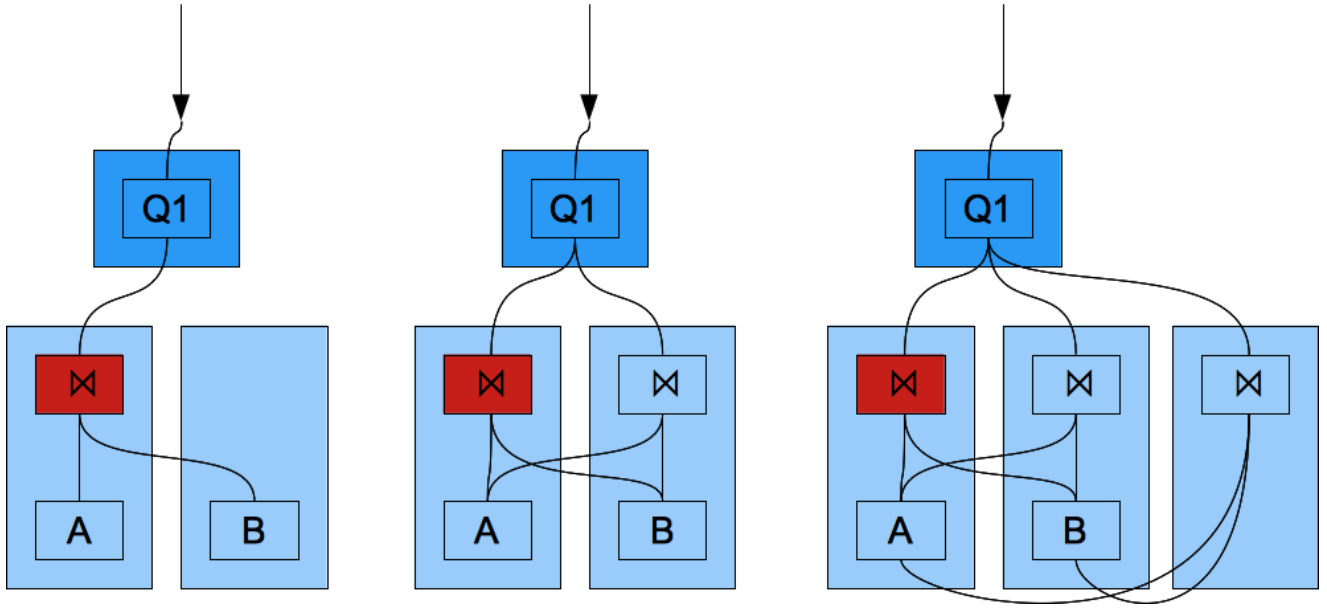
- **Databases do some dynamic optimization:**
  - **Optimizer decides on query plan using data information (histograms, etc).**
  - **Autonomic/self-managing databases (see Ashraf's CS848 course)**
    - **Add/remove machines to adapt to load**
    - **Index/physical layout recommendations**
    - **Materialized view recommendations**
    - **Adapting memory usage**
  - **Query mix optimizations (admission control type) (see Xuhui)**

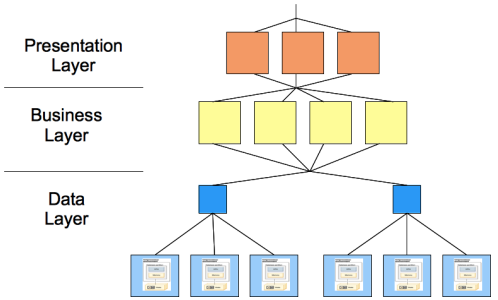
A) fundamental optimization approaches

- DB's rarely change query plan or execution after starting query
- If after starting distributed query, CPU of join portion becomes a bottleneck:
  - Move join onto another machine…
  - Run join on more machines and/or with more threads.
- Try the start of two query plans to see if your performance predictions hold, then pick one to finish and abort the other, or re-optimize with new predictions.

A) fundamental optimization approaches

Dynamically move join if it becomes a CPU bottleneck:





A) fundamental optimization approaches

- **DB's don't use dynamic optimization at lower level.**
  - **e.g. stored procedure**

- **This is where JIT type compilation could be immediately valuable.**

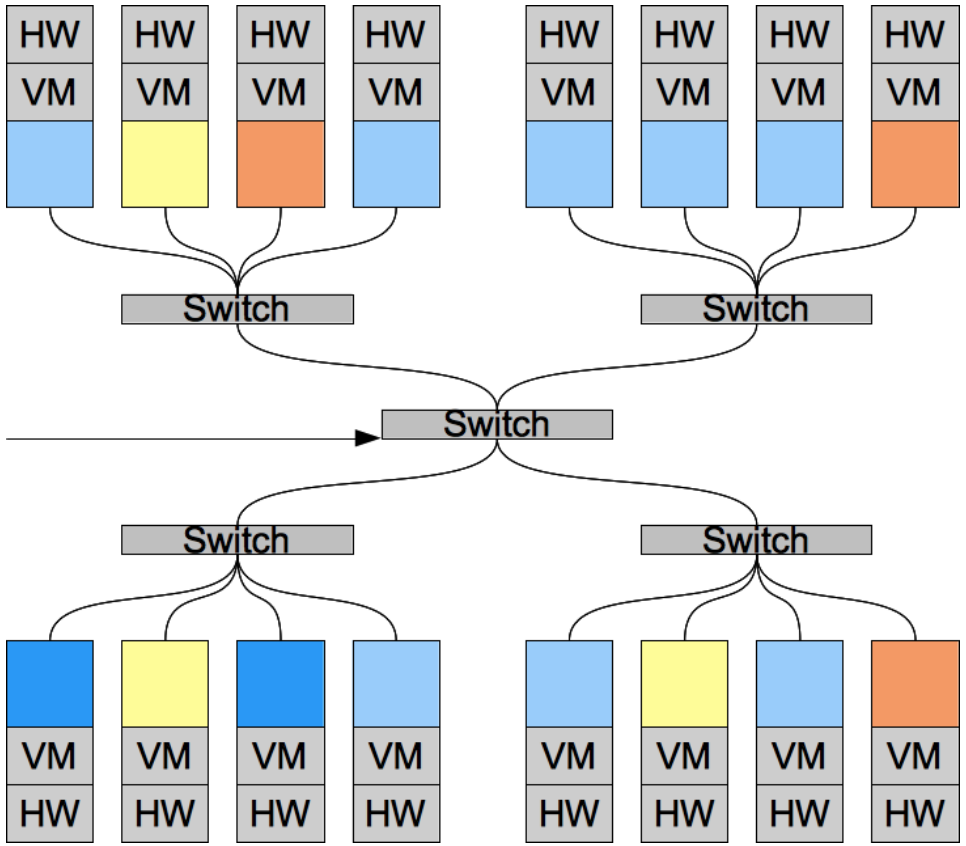
A) fundamental optimization approaches

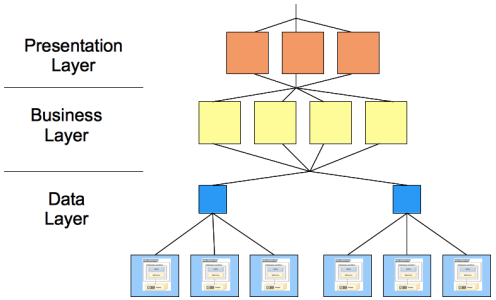
- **belief: you must include hardware optimizations in the entire system**

- **Current locations of hardware knowledge:**
  - **Database buffer pool means you don't trust the underlying system to do memory management and cache optimization.**
  - **Optimizer knows or assumes a lot about hardware.**
  - **Threading assumptions imply knowledge of CPU hardware AND other queries or applications running in the system.**
  - **The number of processes at each tier implies some knowledge of hardware.**

A) fundamental optimization approaches

But, processes can move and hardware can change.





A) fundamental optimization approaches

- **= separate hardware from logic.**
  - **This is difficult, especially because of data layout.**

- **= expose enough information/options so you can dynamically optimize.**

B) machine design

- belief: Cache != memory

- Use cache as memory & cache isolation issues:
  - Cache moving with context switch:
    - "Memory Contexts: Supporting Selectable Cache and TLB Contexts", Tim Brecht, NDCA –2, June, 2011.
  - Cache works well until your working space can't fit, then it falls over unpredictably.
    - Allocate cache to a process and grow when needed.
    - Really need predictability for Quality-of-Service using shared VM environment.

- Use memory as cache:
  - Database Buffer Pool
    - Interact with other caches (e.g. OS cache) (see Xuhui)
  - OS file system cache

B) machine design

- **belief: memory != stable storage**

- **Memory used as stable storage:**
  - **SAP IN-MEMORY COMPUTING: Real-time Analytics**
  - **Memory replication on another machine is 'stable'**

- **Stable storage that looks like memory:**
  - **Flash memory, e.g. SSD**
    - **Read is fast, write is in blocks and slower**
  - **Phase-change memory (PCM):**
    - **IBM Press Release - developed PCM 100x faster than flash (Jun 30th 2011)**
      - "the scientists achieved a worst-case write latency of about 10 microseconds, which represents a 100x performance increase over even the most advanced Flash memory on the market today"
    - **"Better I/O Through Byte-Addressable, Persistent Memory", Condit et al., SOSP 2009**
      - "byte-addressable persistent memory technologies (BPRAM) … such as phase change memory … can be placed side-by-side with DRAM on the memory bus, available to ordinary loads and stores by a CPU"

B) machine design

- belief: new hardware systems will look like and work like old hardware systems.

- Flash + ALU + very little cache = processing node.
  - (Partha Ranganathan HPLabs)
- PCM on memory bus (Byte-Addressable…).
- Isolating cache to a specific process (reduce interaction).
- Cell processors (= vector processors with normal CPU on one chip).
- Processing in the GPU.

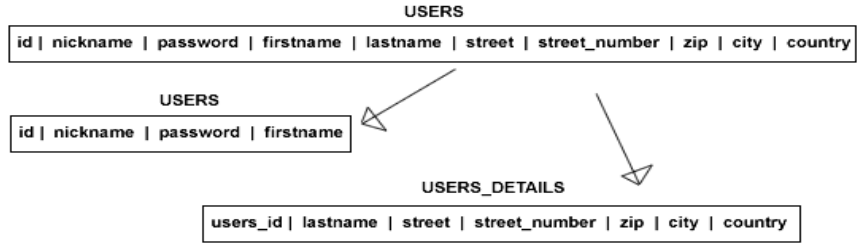
B) machine design

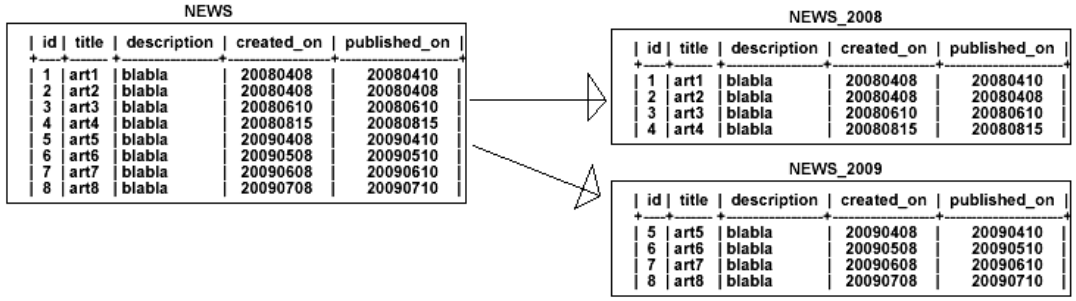
- = logic should act on data and specify changes in the data's access methods, persistence, redundancy, and consistency when needed.
  - Transactional Address Spaces (TAS) - Andrew Kane (course project).
  - Options for data layout should be available to the optimizer, with conversion routines.
    - Note, this could support upgrade and migration while online.

C) architecture of DB level distribution

- **belief: data is owned by a machine/partition in chunks**

Vertical:





:Horizontal

Images from: http://blog.nickbelhomme.com/php/scalable-web-applications_158

C) architecture of DB level distribution

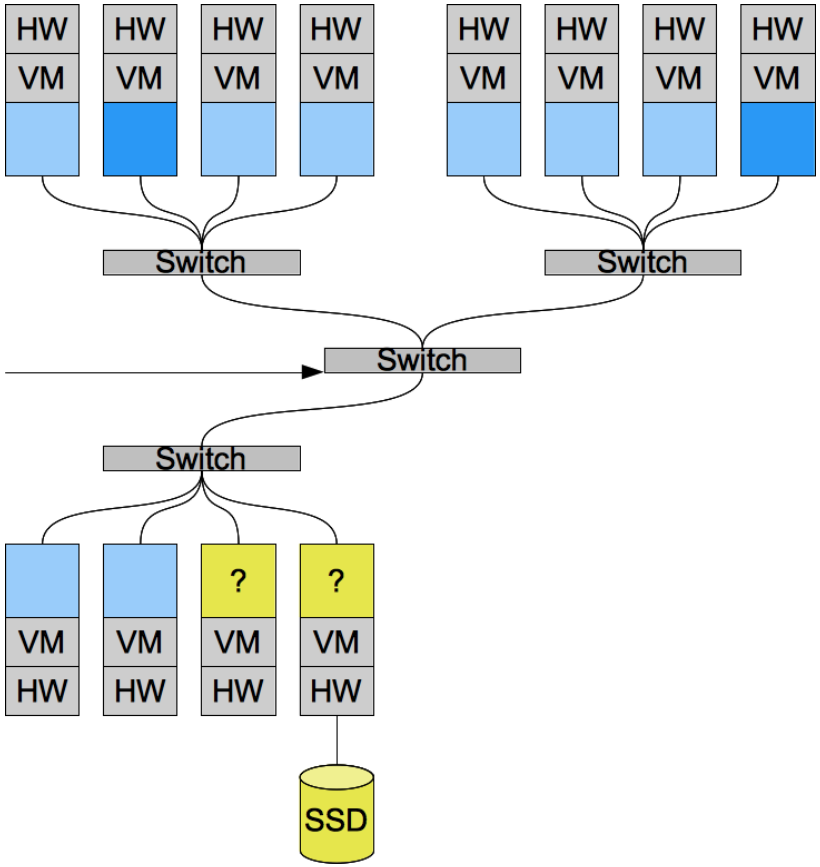
- Indexes and materialized views may break the rule of data ownership by a node.

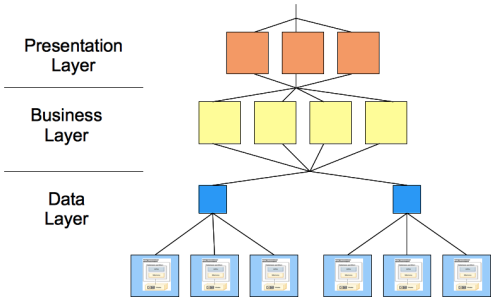
- Rarely see duplication or temporary migration of data.
  - Could maintain both column store and row store, or parts of each…
  - Could move hot or recently updated data to another (faster/ better/less loaded) machine.

- E.g. how to take advantage of a new machine? What if it has new/different hardware?

C) architecture of DB level distribution

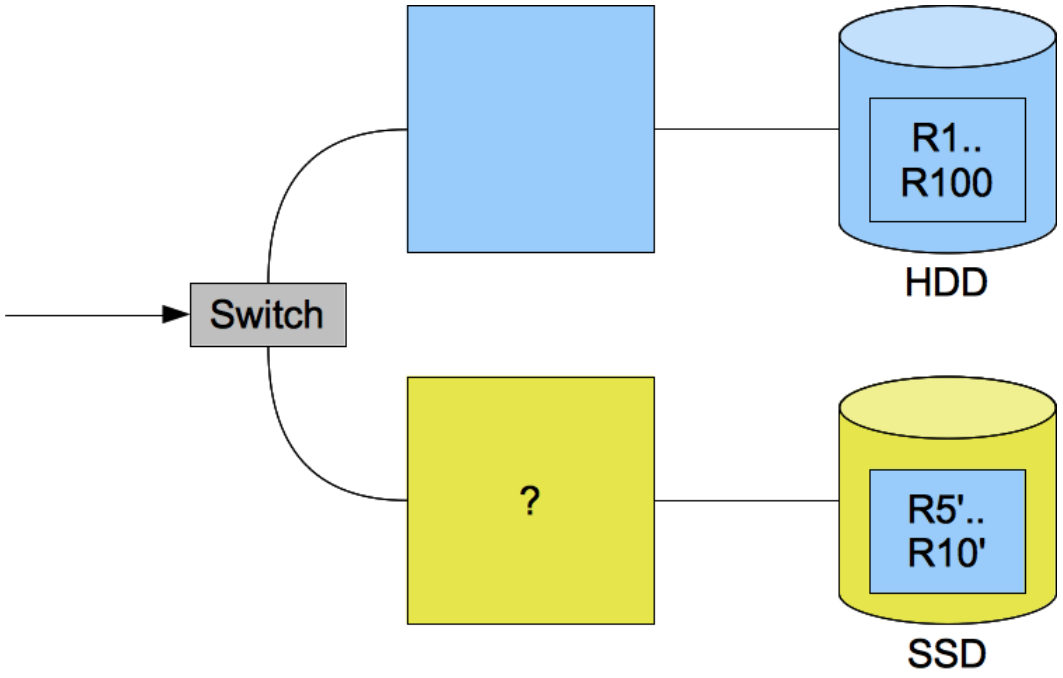
Add one machine with SSD, but can't take advantage of it because of nodes owning data



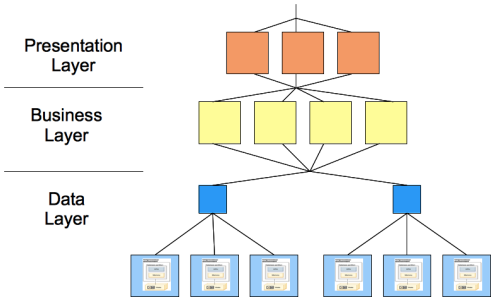


C) architecture of DB level distribution

Really want to put some modified data on SSD and refer to primary location.



May migrate back to original location at later time, or before access by query…



C) architecture of DB level distribution

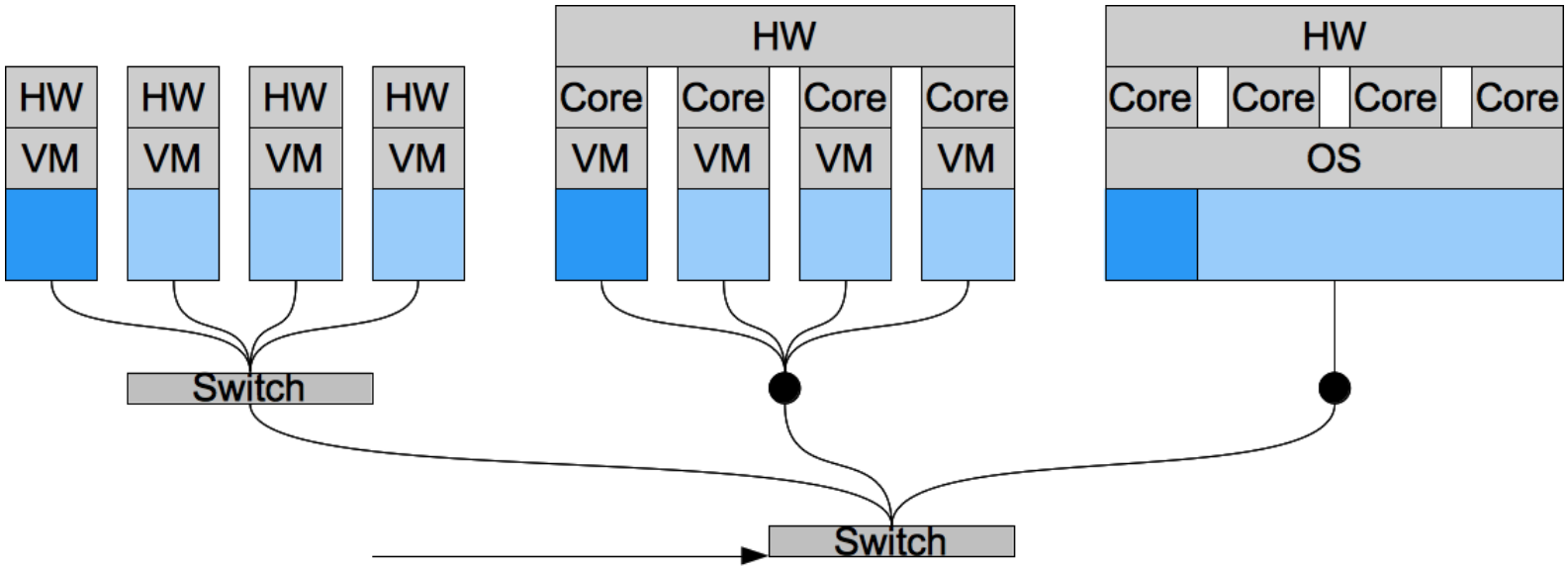
- **belief: software approaches within a chip/machine are separated from distributed approaches (i.e. nodes + hierarchy connecting them).**

- **Movement of data within a chip or between cores is not normally taken into account.**

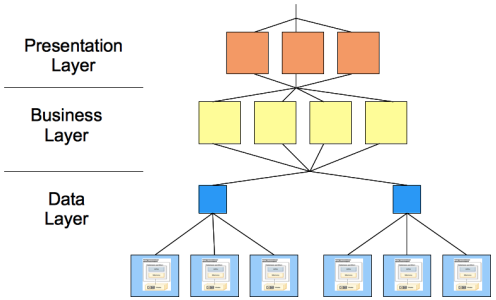
- **Indirectly taking advantage of limiting data movement:**
  - **Split processing of query into processing type chunks executed using a thread pool, results in faster system (MS SQLServer) - Pedro Celis W09.**

C) architecture of DB level distribution

Why approach the optimization differently if you have 4 machines or a 4 core machine?



Cost of accessing shared memory/cache on the single machine should be taken into account (shared L3 cache…), just like communication between processes/machines.



C) architecture of DB level distribution

- **= allow more dynamic treatment of data ownership, including duplication.**

- **= optimize across entire distributed database system in a consistent way.**

D) architecture of full software solution

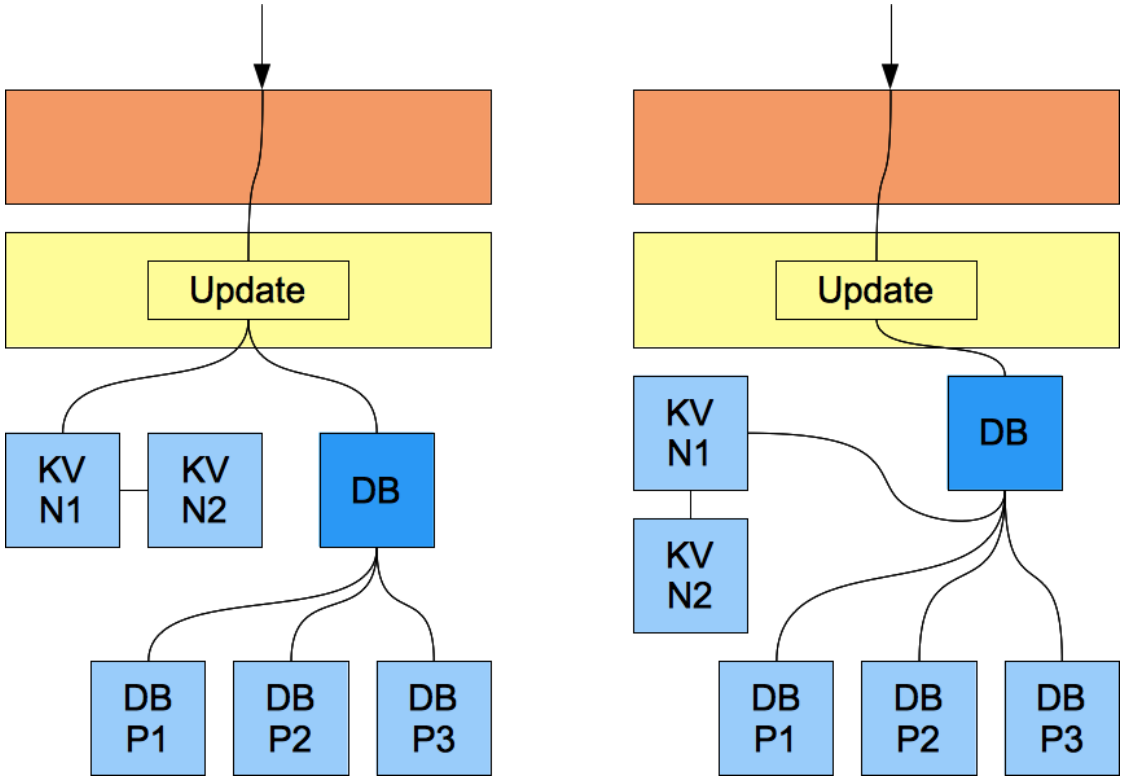
- belief: you can't use multiple distributed systems together
  - (i.e. interaction only at the top level if at all and no shared transactions)

- E.g. combine DBMS with Key-Value store.
  - Need to maintain data consistency…
  - Expose transactions to layer above, consistent for multiple data systems.

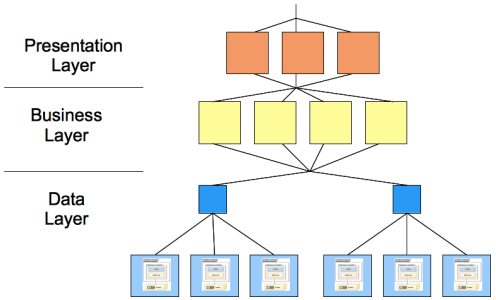
- Could do this in the same way as maintaining an index or materialized view.
  - Allow optimizer to use multiple data systems.

D) architecture of full software solution

Maintain updates to Key-Value store and database.



Explicit (user) **vs.** **Implicit (optimizer)**



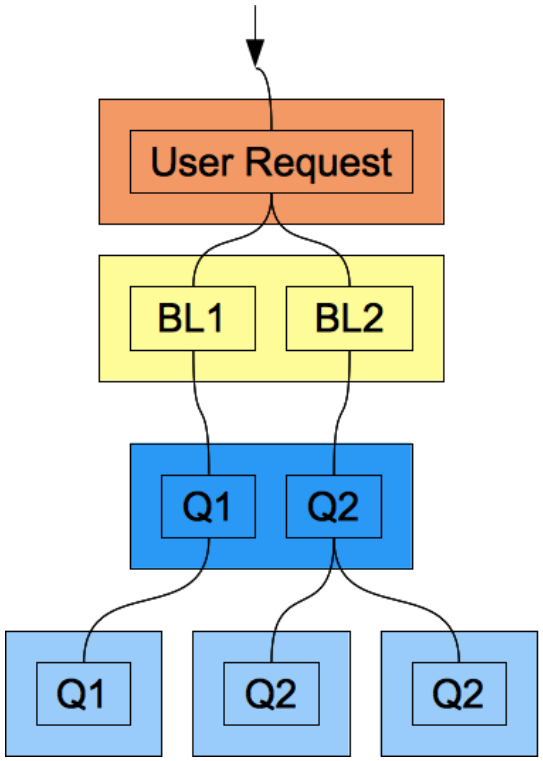
D) architecture of full software solution

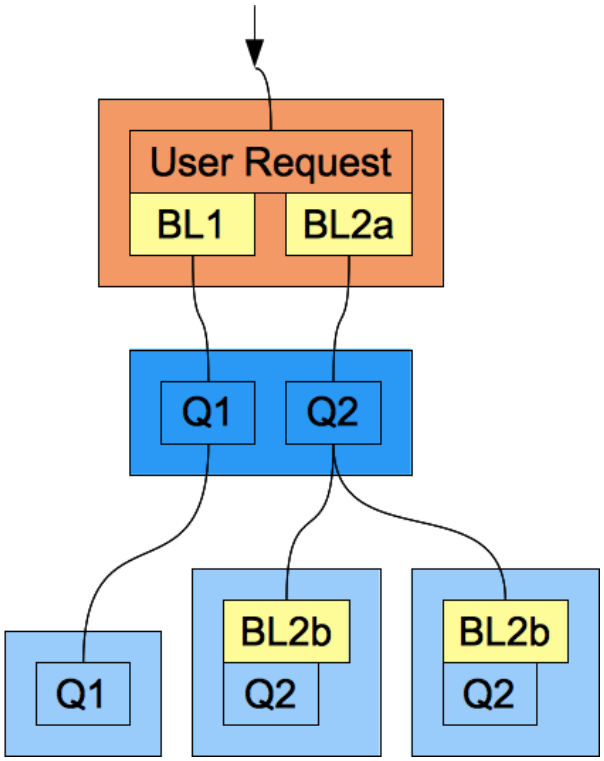
- belief: tiered systems are the way to go, and can't do optimization between tiers.

- "Dynamic Provisioning of Multi-tier Internet Applications", Urgaonkar et al., ICAC 2005.
  - Decide in which tier to add a new machine.
  - Can't share resources well.

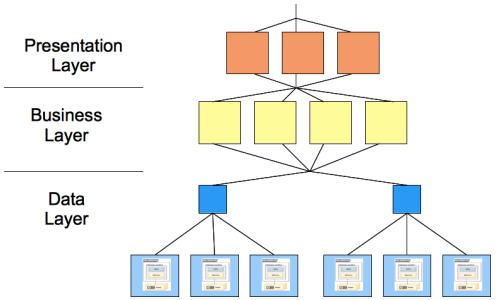
- Combine the tiers and dynamically decide where to execute code pieces.
  - Better load balancing, failover, resource usage, etc.
  - Design everything as code snippets:
    - Allow snippet execution to move around.
    - Including DB query elements.
    - Multiple implementations with selection at runtime by optimizer.

D) architecture of full software solution





Fixed tiers **vs.** **Mixing tiers**



D) architecture of full software solution

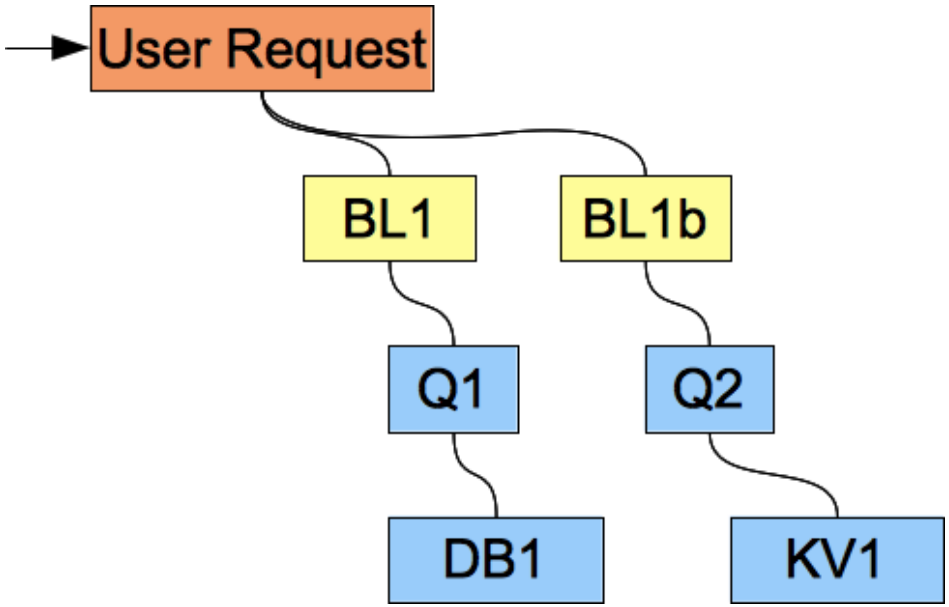
- **= optimize across entire system including all levels, and multiple data engines**

outline

- Three tiered distributed database
- Assumption areas:
  - A) Fundamental optimization approaches
  - B) Machine design
  - C) Architecture of DB level distribution
  - D) Architecture of full software solution
- Possible system design
- Conclusions

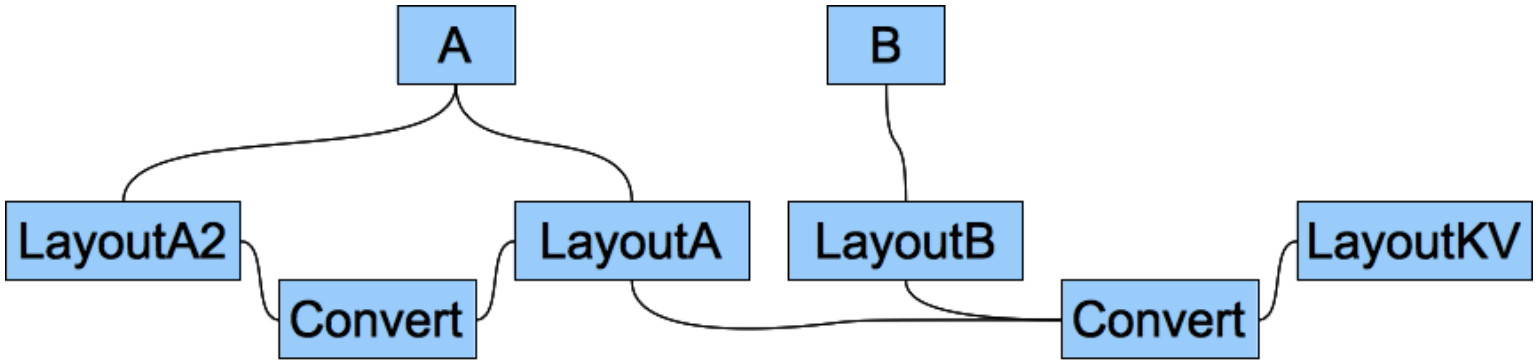
possible system design

- Specify tree of code chunks to execute all code levels. Multiple implementations of the chunk interfaces allow execution plans and optimization. Must specify enough information to do such optimization (estimates of processing, data input/output size, etc).



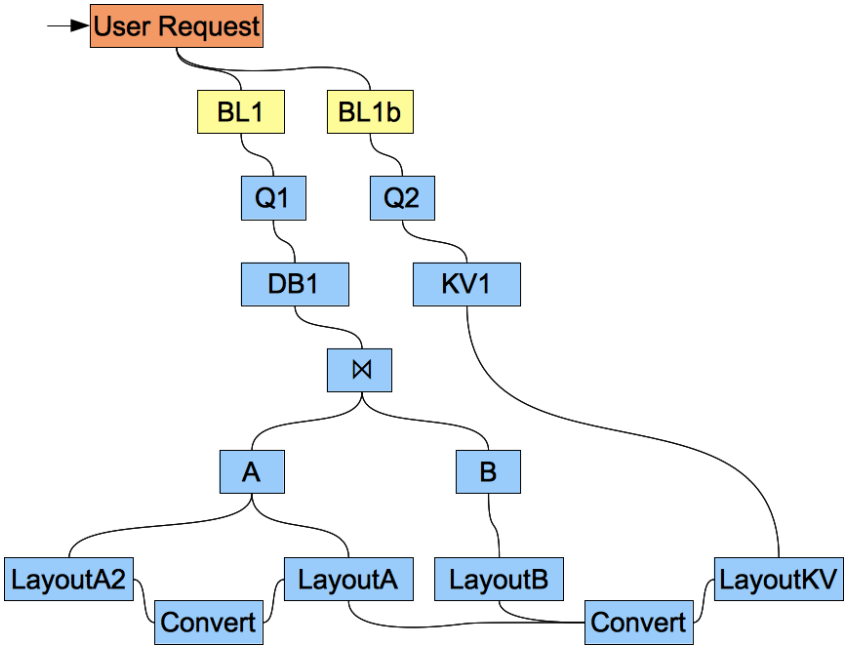
possible system design

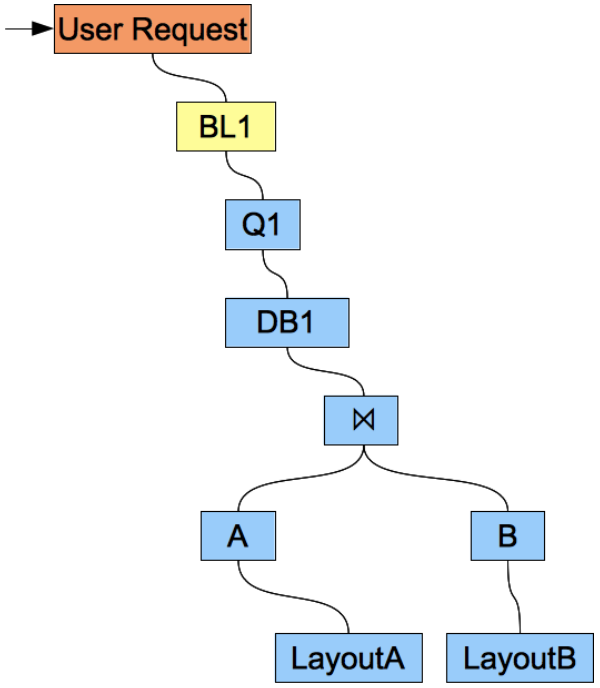
- Specify routines to convert data between layout formats to allow dynamic layout and duplication of data in different formats.



possible system design

- Optimizer decides which implementations to use and where to run them.



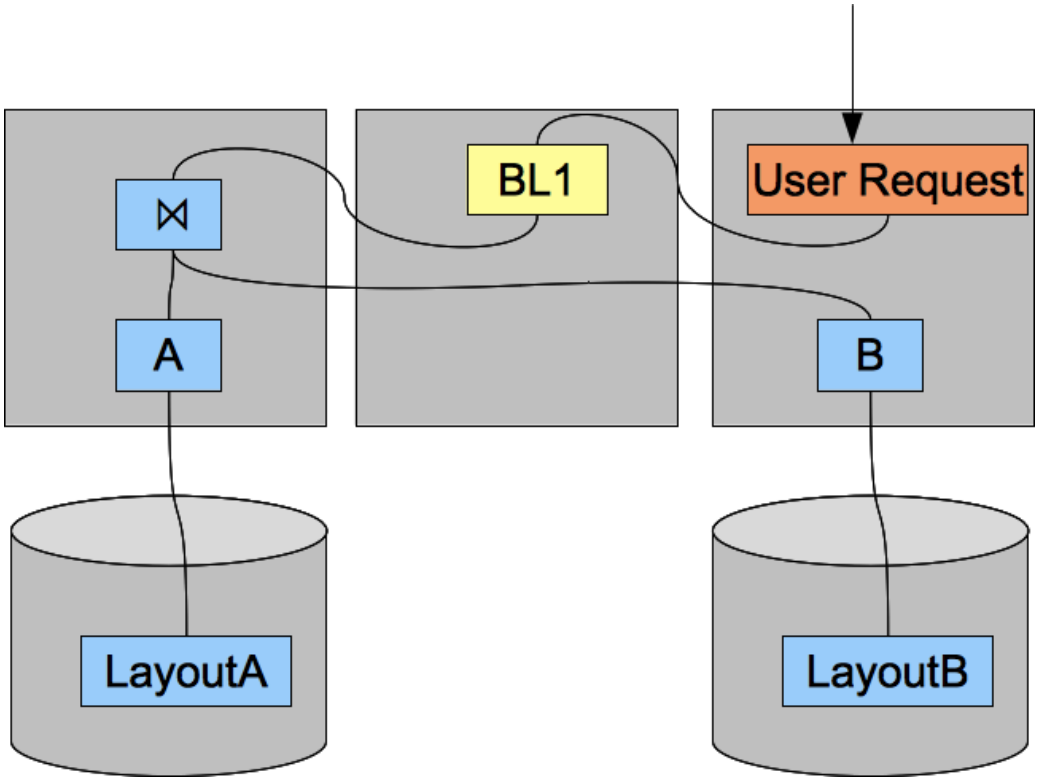


- Possible choices

Plan selection

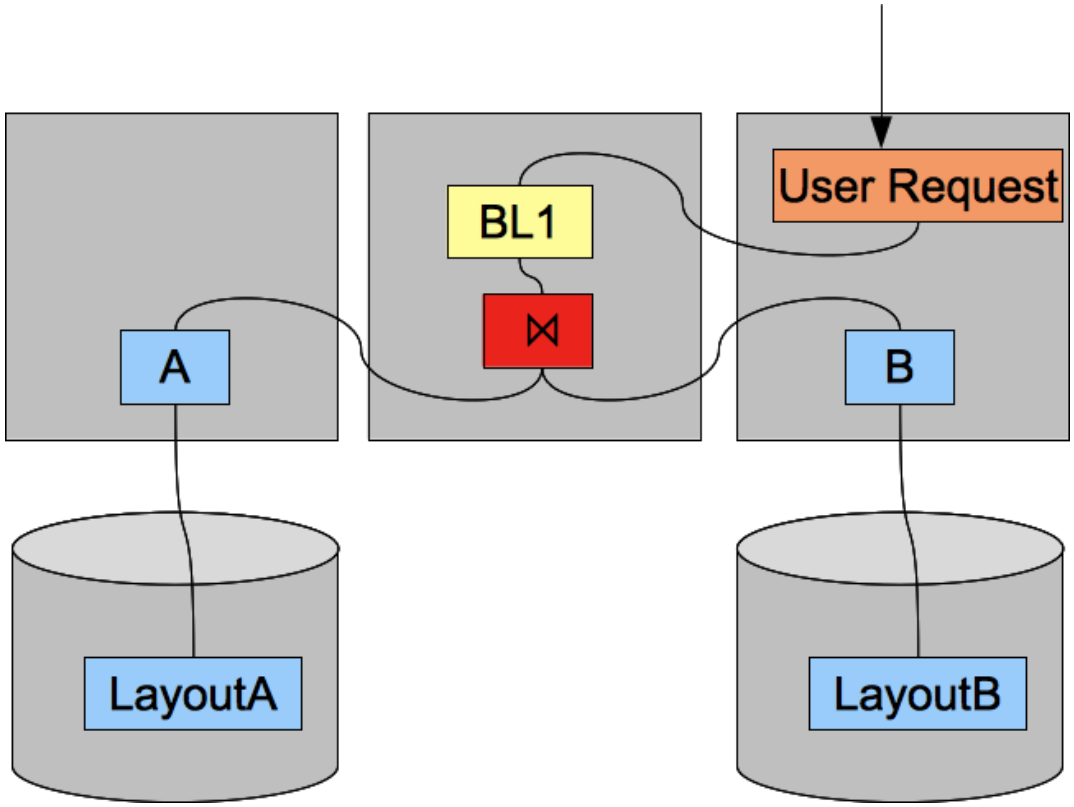
possible system design

- Map onto machines, JIT low-level as needed, monitor for higher level changes.



possible system design

- Execute higher level changes if needed, JIT/un-JIT as needed.



possible system design

- Tools to help:
  - Transactional memory with attributes specifying replication/consistency/etc.
    - Transactional Address Spaces (TAS)?
  - Consistent intermediate form for code would allow JIT type optimizations at all points in the system.
    - (Perhaps Microsoft has the right idea here with .NET intermediate form.)

outline

- Three tiered distributed database
- Assumption areas:
  - A) Fundamental optimization approaches
  - B) Machine design
  - C) Architecture of DB level distribution
  - D) Architecture of full software solution
- Possible system design
- Conclusions

conclusions

- Talk to the compiler people.
  - Try using Just-In-Time compilation for query execution.
- Separate the code logic from the optimizer.
- Look at the entire system as a unit.
  - Include multiple data management systems.
  - Try to have one optimizer for entire system.

- Questions / Comments ?