HPI Hasso Plattner Institut

IT Systems Engineering | Universität Potsdam

In-Memory Data Management

Jens Krueger

Enterprise Platform and Integration Concepts
Hasso Plattner Intitute

OLTP vs. OLAP

On**l**ine **T**ransaction **P**rocessing (OLTP)

Organized in rows

On**l**ine **A**nalytical **P**rocessing (OLAP)

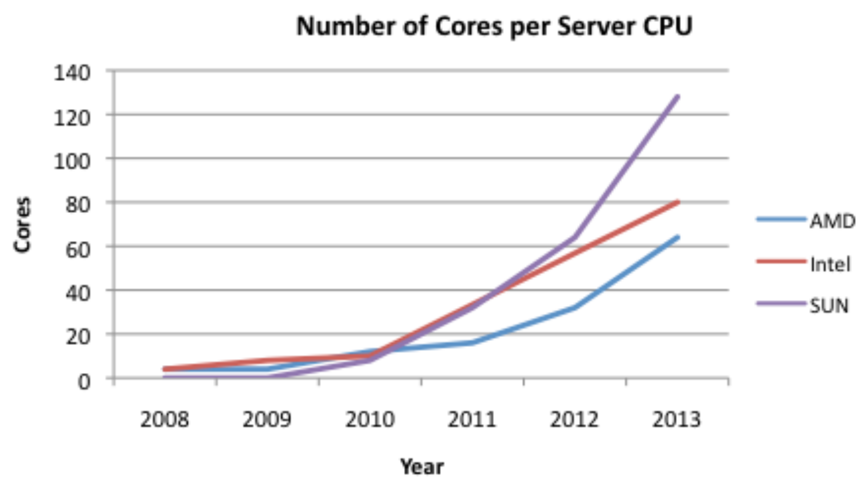
Organized in columns

Modern enterprise resource planning (ERP) systems are challenged by **mixed workloads**, including OLAP-style queries. For example:

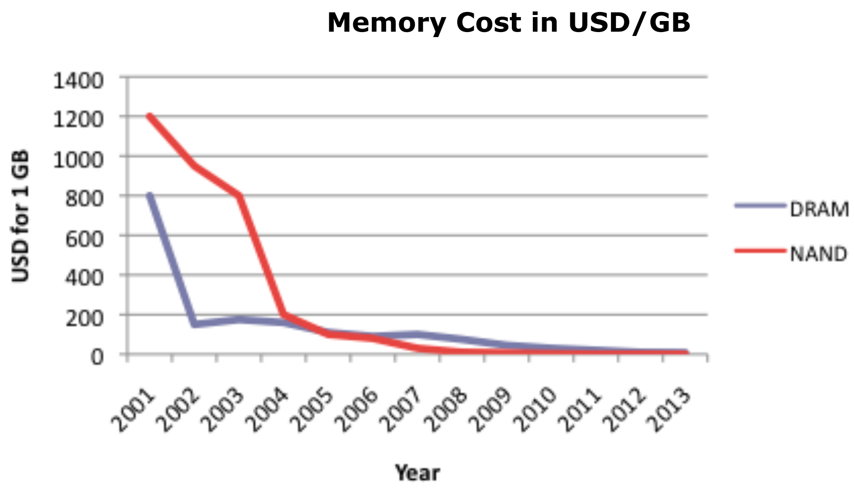
- Dunning runs
- Available-to-promise
- Real-time reporting

Dominant Hardware Trends

- Multicore Technology
  - Moore's Law:
    "... number of transistors ...
    doubling every 18 months"
  - CPU frequency hit limit in 2002, but Moore's Law holds today



- Main Memory Technology
  - Increased size: up to 2 TB of main memory on one main board as of today
  - Constantly dropping costs



Enterprise Application-Specific Data Management

Requirements engineering to:

- Define enterprise application-**specific** requirements
- Leverage the advantages of an **in-memory** system
- Identify **patterns** and data characteristics
- Find potential improvements on **data schema**
- Estimate **compression** in enterprise environments
- Validate our assumptions against **real** data and systems

Enterprise Data is Sparse Data

- Many columns are not used even once
- Many columns have a low cardinality of values
- NULL values/default values are dominant
- Sparse distribution facilitates high compression

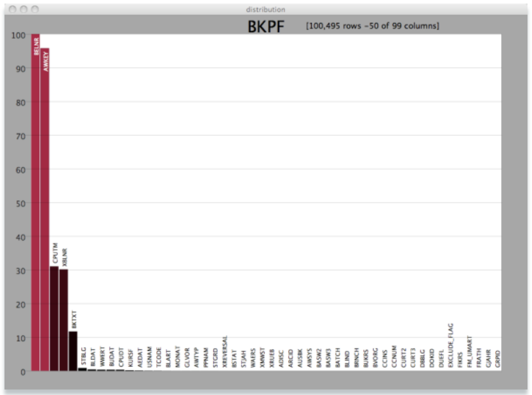
Results:
Distinct Values per Attribute

Results from analyzing financials

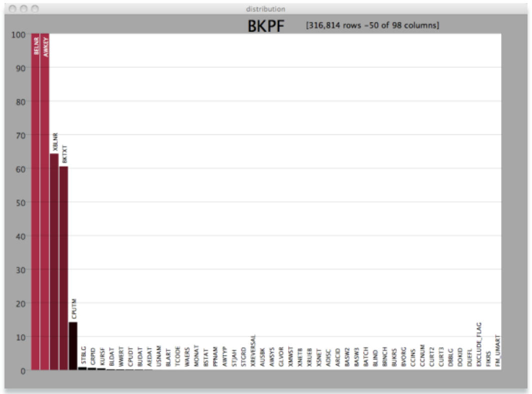
Distinct values in accounting document headers

(99 attributes)

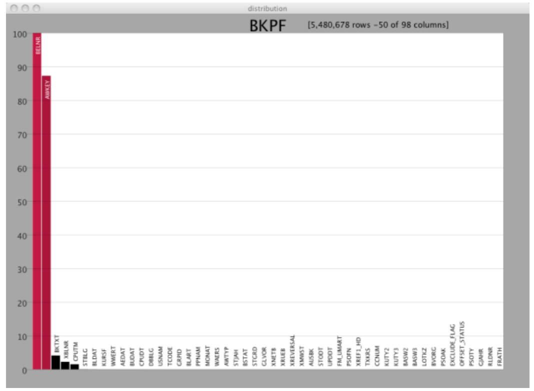
CPG



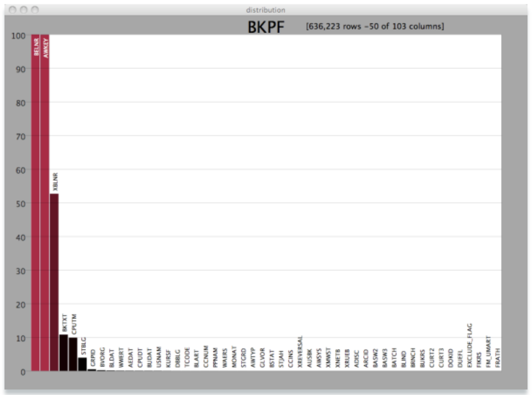
Logistics



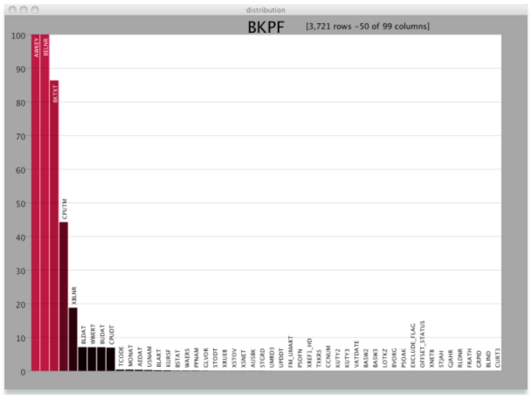
Banking



High tech



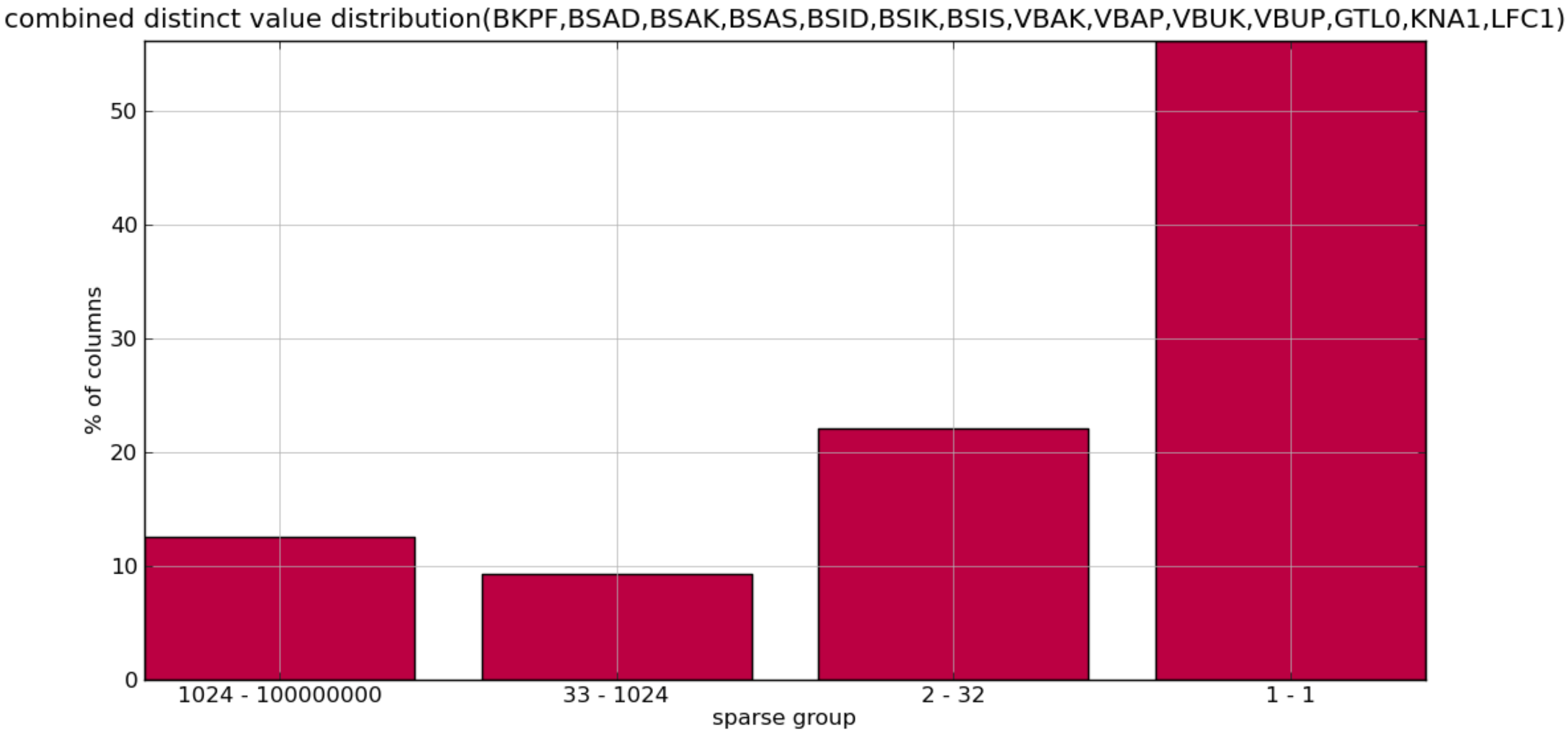
Discrete manufacturing



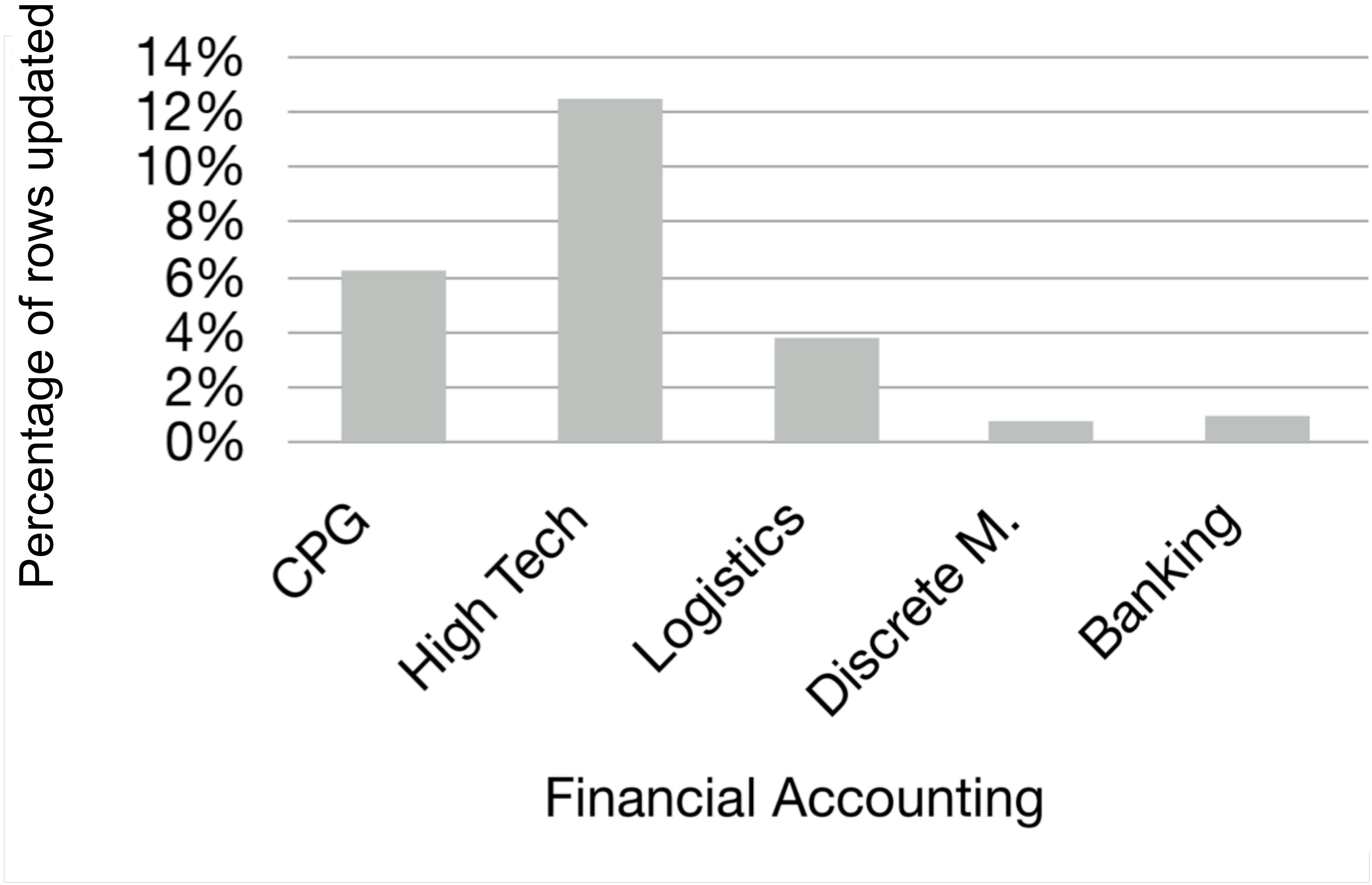
Enterprise Data is Sparse Data

55% unused columns per company in average

40% unused columns across all companies

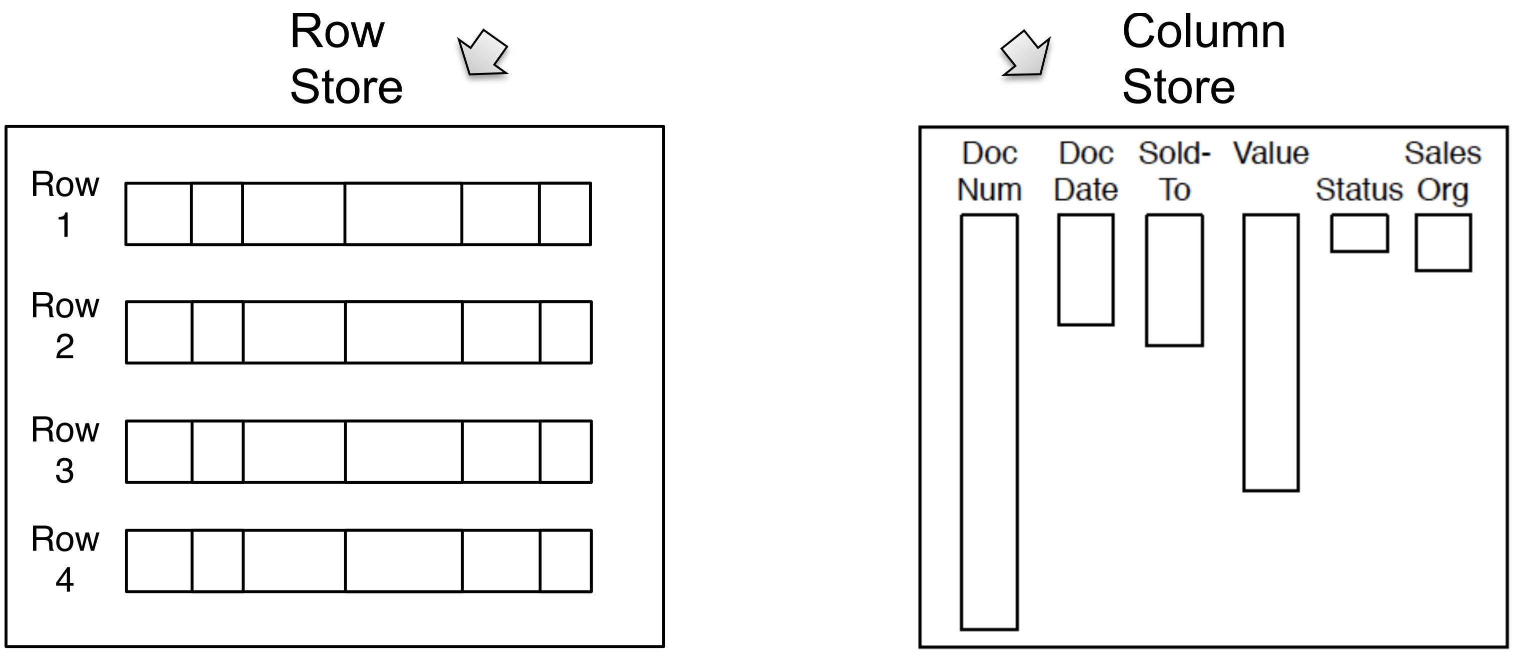


Results:
Accounting Document Updates



Row vs. Column Store

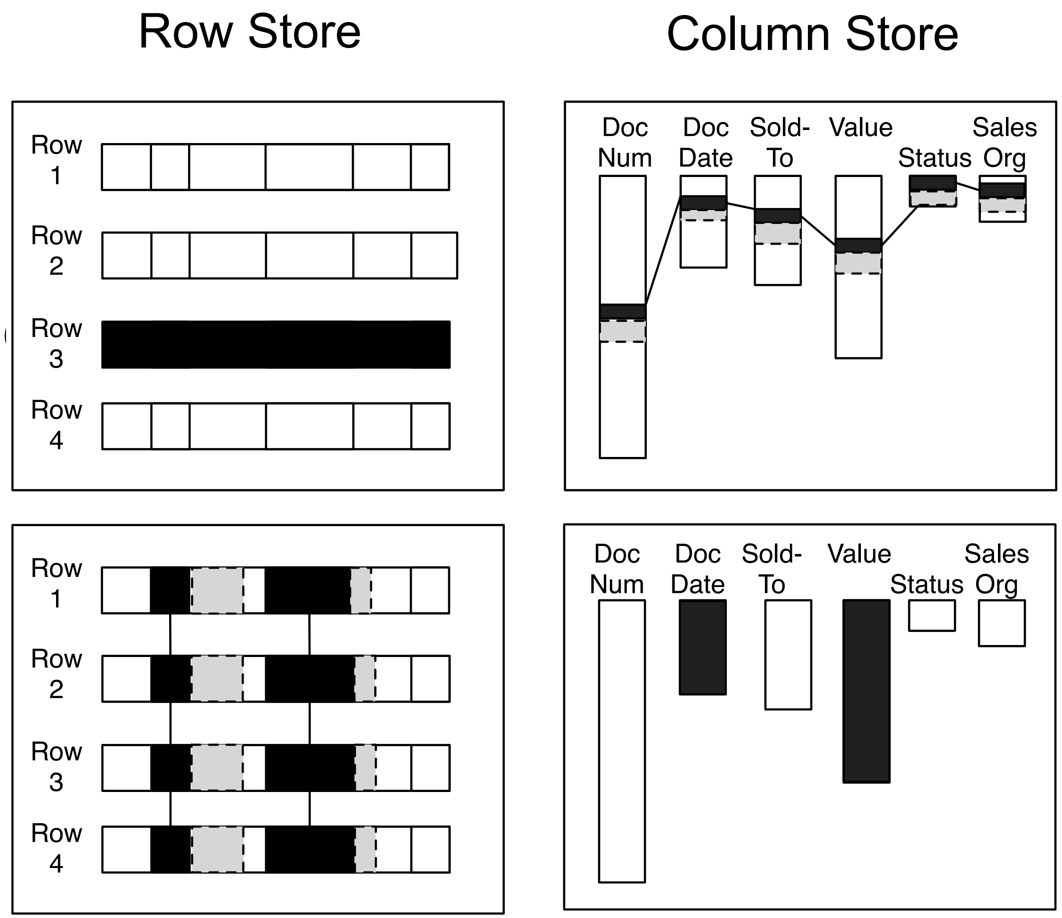
Document Number	Document Date	Sold-To Party	Order Value	Status	Sales Organization	...
95769214	2009-10-01	584	10.24	CLOSED	Germany Frankfurt	...
95769215	2009-10-01	1215	124.35	CLOSED	Germany Berlin	...
95779216	2009-10-21	584	47.11	OPEN	Germany Berlin	...
95779217	2009-10-21	454	21.20	OPEN	Germany Frankfurt	...



OLTP vs. OLAP Queries

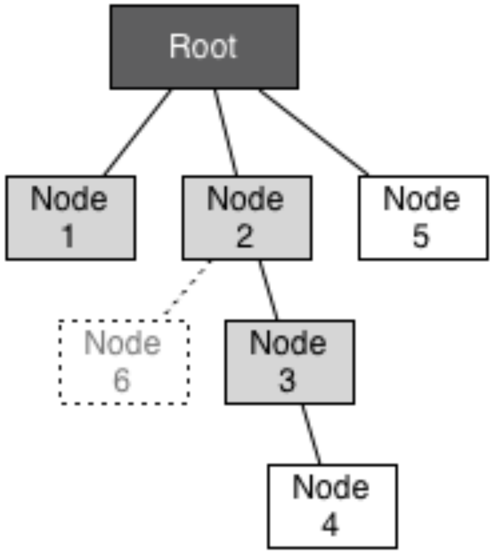
SELECT *
FROM Sales Orders
WHERE Document Number = ‘9577921

SELECT SUM(Order Value)
FROM Sales Orders
WHERE Document Date > 2009-01-20



- Single object instance vs. set processing on attributes of nodes of objects
- Enterprise applications perform **set processing** (items for an order, orders for a customer)
- Bring application logic closer to the storage layer using stored procedures

Object Data Guides

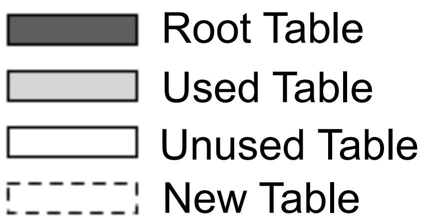


ID	Type	...	...	ODG
1	Order	...	...	(1,1,1,0,0)

1 = table is relevant

0 = table not relevant

- Enterprise systems make heavy use of objects - objects must be mapped to relations
- Often, objects are distributed sparsely over all tables representing nodes
- Relevant tables can now be queried in parallel
- When adding new tables, only add another bit



Compression in Column Stores

Document Number	Document Date	Sold-To Party	Order Value
95769214	2009-10-01	584	10.24
95769215	2009-10-01	1215	124.35
95779216	2009-10-21	584	47.11
95779217	2009-10-21	454	21.20

Dictionaries

Document Number

0	95769214
1	95769215
2	95779216
3	95779217

Order Value

0	10.24
1	21.20
2	47.11
3	124.35

Document Date

0	2009-10-01
1	2009-10-21

Sold-To Party

0	454
1	584
2	1215

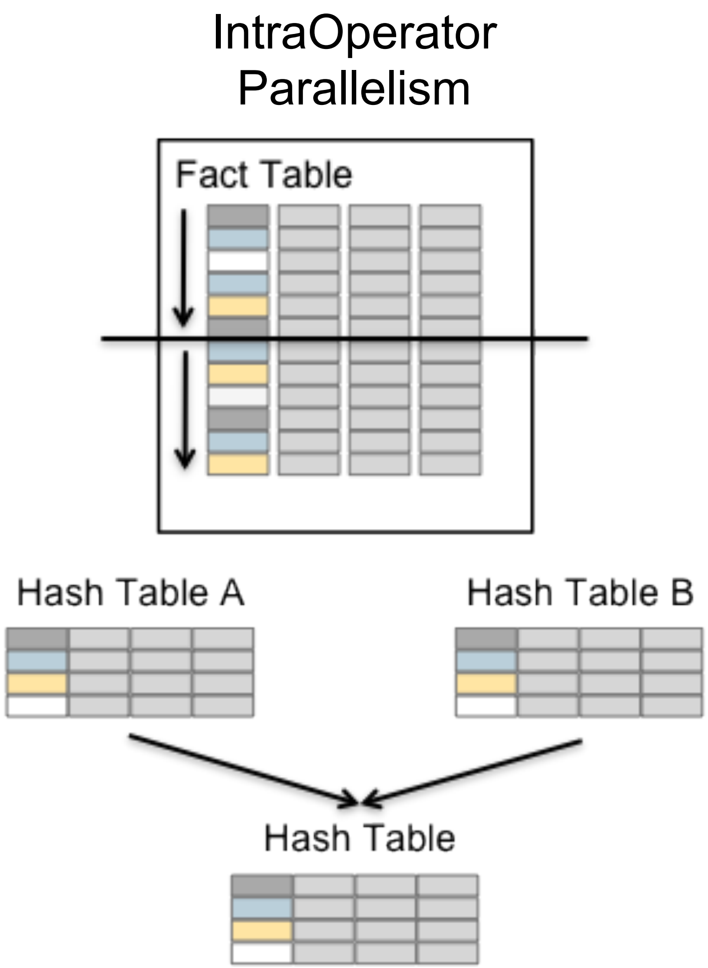
Document Number	Document Date	Sold-To Party	Order Value
0	0	1	0
1	0	2	3
2	1	1	2
3	1	0	1

Multi-Core Usage

- **Set processing** – scan is dominant pattern in enterprise apps
- **Sequential** scans allow best bandwidth utilization between CPU cores and memory
- **Independence** of tuples within columns allows easy partitioning and therefore parallel processing (see Hennessy [1])
- Increased memory bandwidth in current and next generation CPUs allows even **faster memory scans**. Current Nehalem architecture allows multiple memory channels, with an increased combined bandwidth.
- No more materialized views and aggregates: everything is calculated **on-the-fly**

[1] John L. Hennessy, David A. Patterson:
Computer Architecture: A Quantitative Approach

Parallelization in Column Stores



- Columns are optimal for dynamic range partitioning
- One sequential block can be easily split into many (as number of cores) blocks

Stored Procedures

- New enterprise data management requires rethinking of how application logic is written
- Identify common application logic
- Rethink how applications are developed

Insert Only

- Tuple visibility indicated by timestamps (POSTGRES-style time-travel [2])
- Additional storage requirements can be neglected due to low update frequency
- Timestamp columns are not compressed to avoid additional merge costs
- Snapshot isolation
- Application-level locks

[2] Michael Stonebraker:
The Design Of The Postgres Storage System (1987)

Insert Only (Insert)

Document Number	Document Date	Sold-To Party	Order Net Value	Status	BEGIN	END
95769214	2009-10-01	584	10.24	CLOSED	2009-10-01	
95769215	2009-10-01	1215	124.35	CLOSED	2009-10-01	
95779216	2009-10-21	584	47.11	OPEN	2009-10-21	
95779217	2009-10-21	454	21.20	OPEN	2009-10-21	
95779218	2009-10-22	454	0.00	OPEN	2009-10-22	

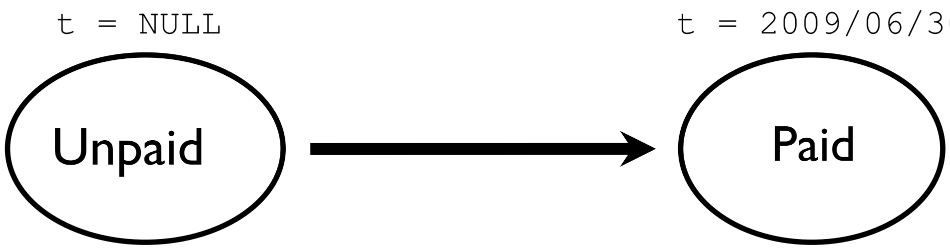
Insert

Insert Only (Update)

Document Number	Document Date	Sold-To Party	Order Net Value	Status	BEGIN	END	
95769214	2009-10-01	584	10.24	CLOSED	2009-10-01		
95769215	2009-10-01	1215	124.35	CLOSED	2009-10-01		
95779216	2009-10-21	584	47.11	OPEN	2009-10-21	2009-10-23	Mark Invalid
95779217	2009-10-21	454	21.20	OPEN	2009-10-21		
95779218	2009-10-22	454	0.00	OPEN	2009-10-22		
95779216	2009-10-21	584	40.00	OPEN	2009-10-23		Insert

Status Updates

- When updates of status fields are changed by replacement, do we need to insert a new version of the tuple?
- Most status fields are binary
- Idea: uncompressed in-place updates with row timestamp



Optimizing Write Performance

- OLTP workload requires many appends
- Instantly applying compression has a severe impact on the performance
- New values are written transactionally safe to a special write optimized storage
- Asynchronous re-compression of all values
- Current binary representation is stored on secondary storage (Flash) for faster recovery

The Delta & Merge

Main-Memory

Secondary Storage

Main Table

Document Number	Document Date	Sold-To Party	Order Net Value	Status	BEGIN	END
95769214	2009-10-01	584	10.24	CLOSED	2009-10-01	
95769215	2009-10-01	1215	124.35	CLOSED	2009-10-01	
95779216	2009-10-21	584	47.11	OPEN	2009-10-21	
95779217	2009-10-21	454	21.20	OPEN	2009-10-21	

Delta Table

Document Number	Document Date	Sold-To Party	Order Net Value	Status	BEGIN	END

Binary Dump

Delta Log (empty)

The Delta & Merge
-Insert -

Main-Memory

Secondary Storage

Main Table

Document Number	Document Date	Sold-To Party	Order Net Value	Status	BEGIN	END
95769214	2009-10-01	584	10.24	CLOSED	2009-10-01	
95769215	2009-10-01	1215	124.35	CLOSED	2009-10-01	
95779216	2009-10-21	584	47.11	OPEN	2009-10-21	
95779217	2009-10-21	454	21.20	OPEN	2009-10-21	

Binary Dump

Delta Table

Document Number	Document Date	Sold-To Party	Order Net Value	Status	BEGIN	END
95779218	2009-10-22	454	0.00	OPEN	2009-10-22	

Delta Log
INSERT INTO Sales Order VALUES ...

The Delta & Merge
- Update -

Main-Memory | Secondary Storage

Main Table

Document Number	Document Date	Sold-To Party	Order Net Value	Status	BEGIN	END
95769214	2009-10-01	584	10.24	CLOSED	2009-10-01	
95769215	2009-10-01	1215	124.35	CLOSED	2009-10-01	
95779216	2009-10-21	584	47.11	OPEN	2009-10-21	
95779217	2009-10-21	454	21.20	OPEN	2009-10-21	

Binary Dump

Delta Table

Document Number	Document Date	Sold-To Party	Order Net Value	Status	BEGIN	END
95779218	2009-10-22	454	0.00	OPEN	2009-10-22	
95779216	2009-10-21	584	47.11	OPEN	2009-10-21	2009-10-23
95779216	2009-10-21	584	40.00	OPEN	2009-10-23	

Delta Log
INSERT INTO VBAK VALUES ...
INSERT INTO VBAK VALUES ...
INSERT INTO VBAK VALUES ...

The Merge Process

- Insert values of delta table into the main table
- Re-compress main table and update dictionary table
- Capture binary image of main table

After the Merge

Main-Memory | Secondary Storage

Main Table

Document Number	Document Date	Sold-To Party	Order Net Value	Status	BEGIN	END
95769214	2009-10-01	584	10.24	CLOSED	2009-10-01	
95769215	2009-10-01	1215	124.35	CLOSED	2009-10-01	
95779216	2009-10-21	584	47.11	OPEN	2009-10-21	2009-10-23
95779217	2009-10-21	454	21.20	OPEN	2009-10-21	
95779218	2009-10-22	454	0.00	OPEN	2009-10-22	
95779216	2009-10-21	584	40.00	OPEN	2009-10-23	

Delta Table

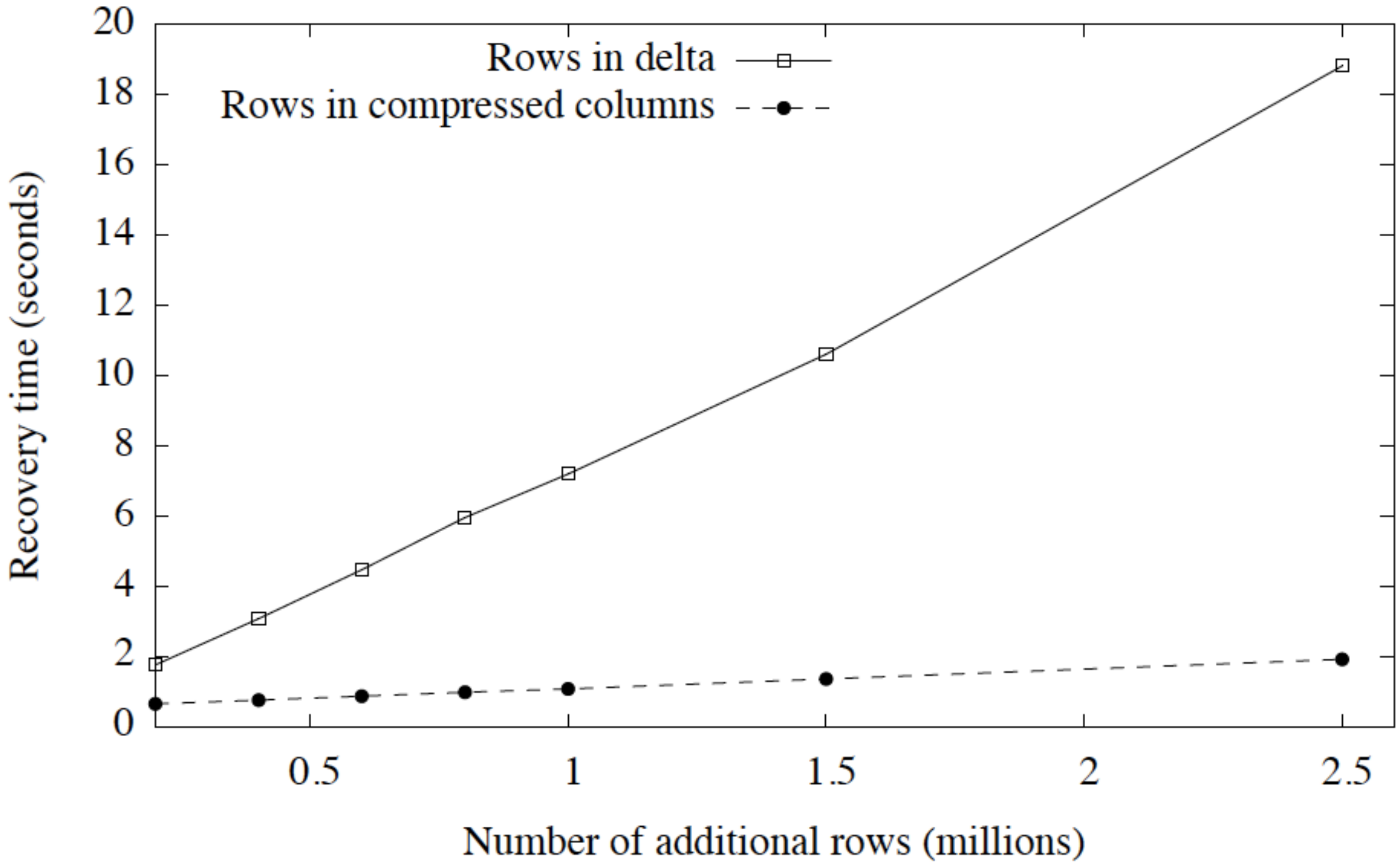
Document Number	Document Date	Sold-To Party	Order Net Value	Status	BEGIN	END

Binary Dump

New Data

Delta Log (empty)

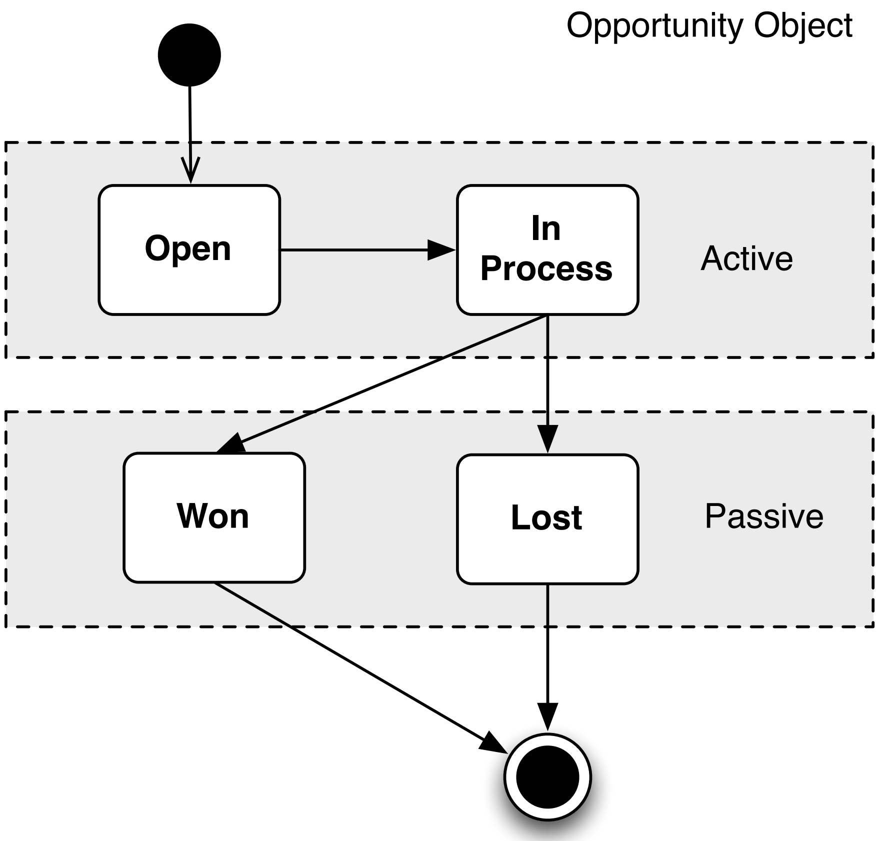
Recovery Time



Memory Consumption

- Experiments show a general factor 10 in compression (using dictionary compression and bit-vector encoding
- Additional storage savings by removing materialized aggregates, save ~2x
- Keep only the active partition of the data in memory (based on fiscal year), save ~5x
- In total 100x possible

Aging = Partitioning



- Each enterprise object has a dedicated lifecycle - modeled using a state-transition diagram
- Events determine the status of an object
- Map states to partitions
- Multiple partitions = parallel queries

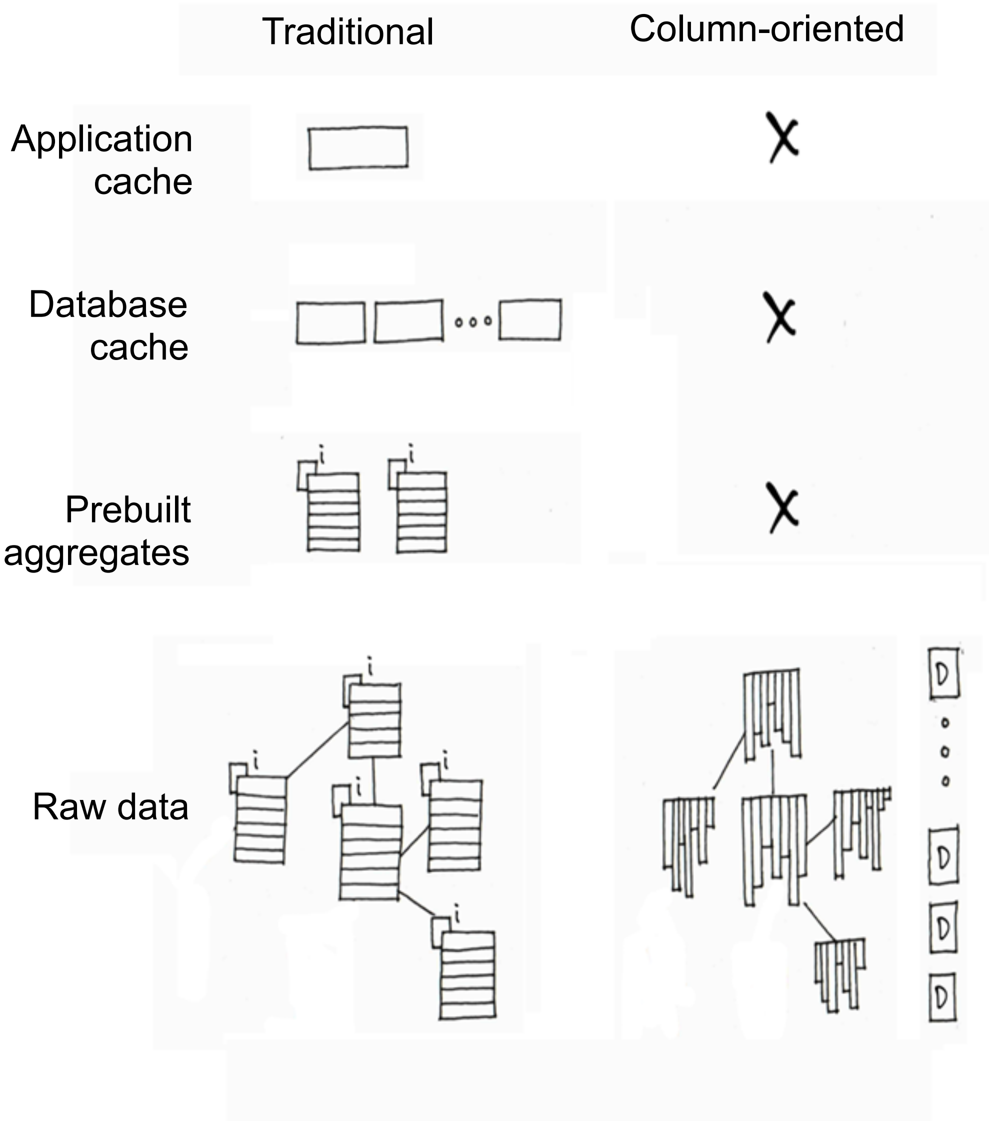
Customer Study: Dunning Run in < 1s?

- Dunning run determines all open and due invoices
- Customer defined queries on 250M records
- Current system: 20 min
- New logic: **1.5 sec**
  - In-memory column store
  - Parallelized stored procedures
  - Simplified Financials

Why?

- Being able to perform the dunning run in such a short time **lowers TCO**
- Add more functionality!
- Run other jobs in the meantime! - in a multi-tenancy cloud setup hardware must be used wisely

Simplified Application Development



- No caches needed
- No redundant objects
- No maintenance of indexes or aggregates
- Data movements are minimized

Advantages

- **Functional**
  - Analytics on current (up-to-the-moment) data
  - No need to predefine reports
  - Transactions enriched with analytics
  - Faster completion of processes
  - More accuracy due to on-the-fly calculation
- **Technical**
  - Column-oriented data organization enables better utilization of modern hardware
  - Redundancy-free schema decreases system complexity
  - Fast full table scan possible on all columns
- **Lower total cost of ownership (TCO)**

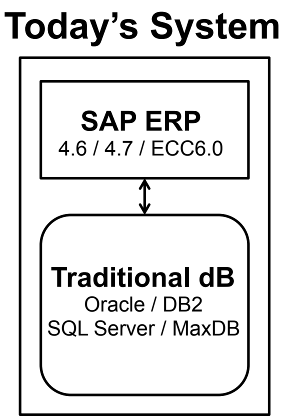
Advantages

- Fewer indices
- Minimal projections
- Active/passive data store
- Any attribute as an index
- Partitioning
- Reduction of layers
- Insert only on delta
- Back-dated analytics
- Multi-core/ parallelization
- Combined column and row store
- Single and multi-tenancy
- Compression
- No materialized aggregates
- SQL interface on columns & rows
- Price/performance of memory

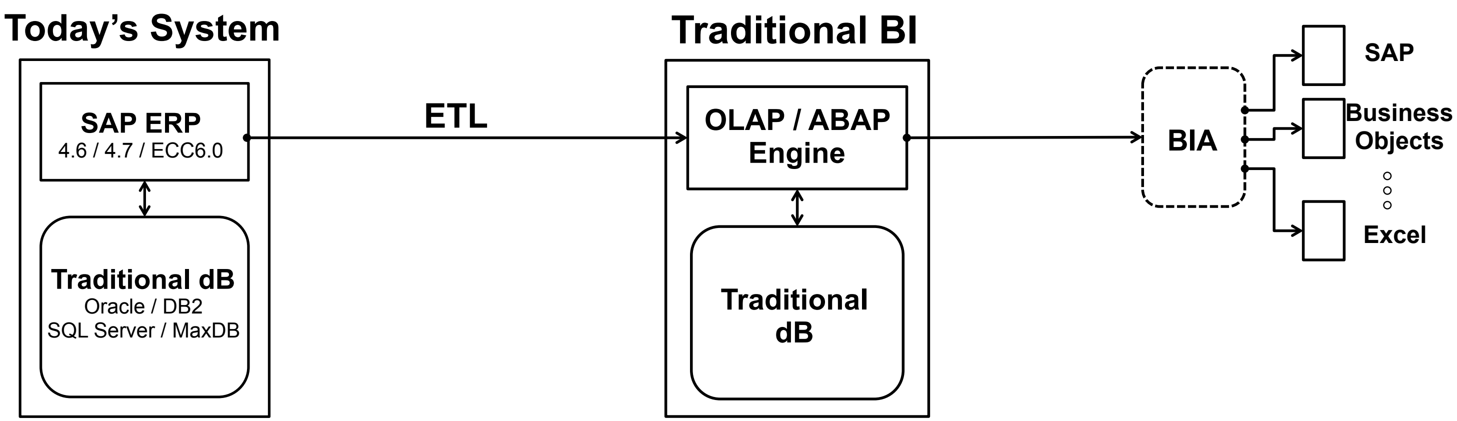
Transition

- Millions of “old” un-optimized lines of code at the customers’ site Transition required
- Row-store replacement
- Part-for-part replacement with bypass
- Transform row-store to column-store on the fly
- Change of application code

Bypass Solution for Transition

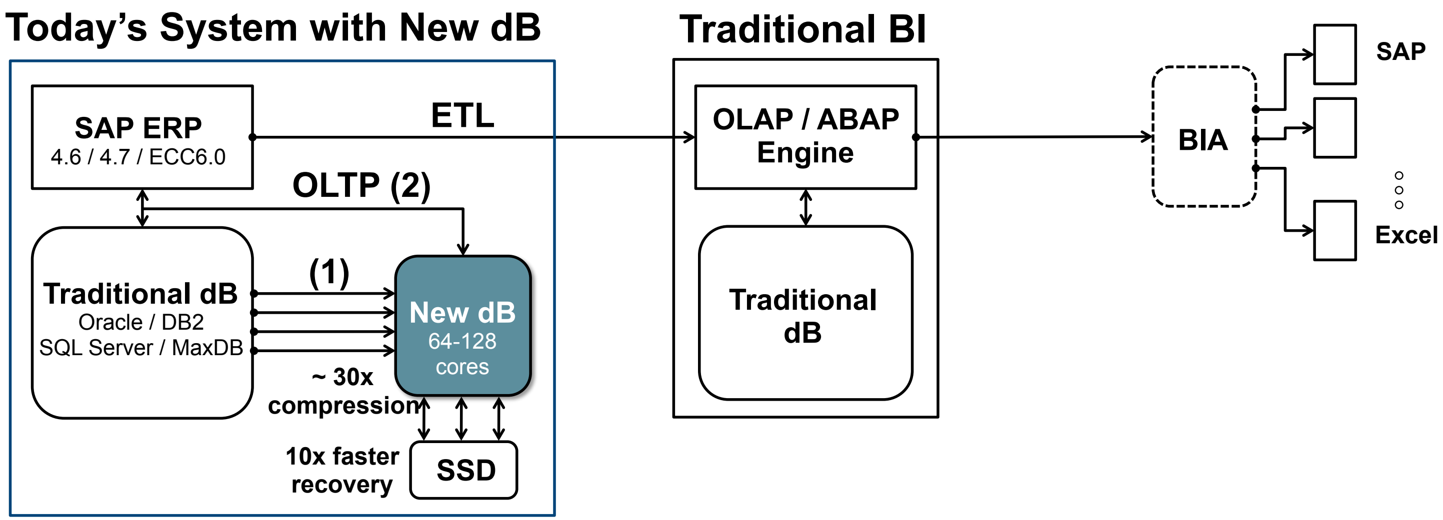


Bypass Solution for Transition



Bypass Solution for Transition

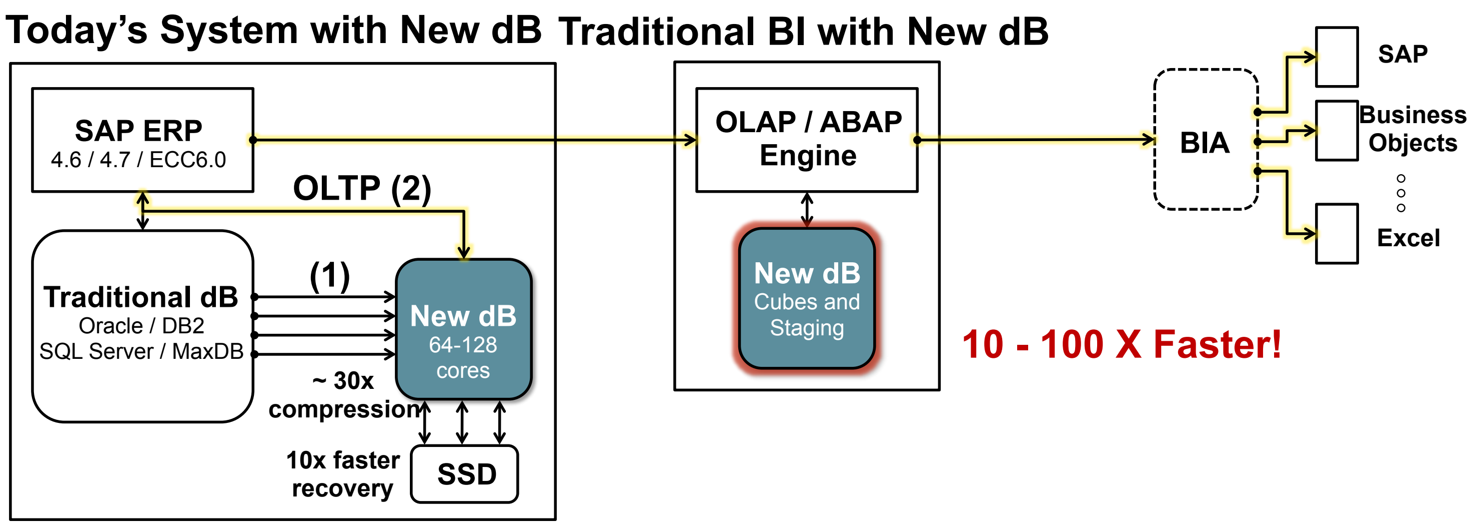
STEP 1: Install and run the in-memory database in parallel



Bypass Solution for Transition

STEP 1: Install and run the in-memory database in parallel

STEP 2: Re-create traditional-style BI in main memory

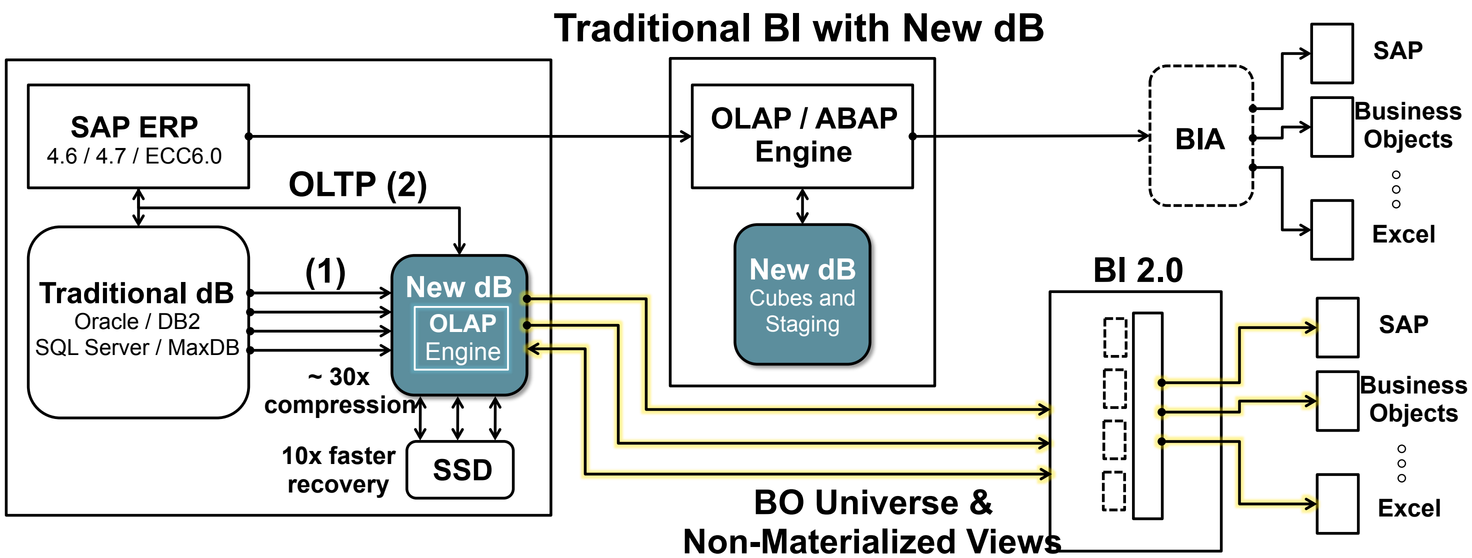


Bypass Solution for Transition

STEP 1: Install and run the in-memory database in parallel

STEP 2: Re-create traditional-style BI in main memory

STEP 3: Introduce next-gen BI running in parallel with no materialized views



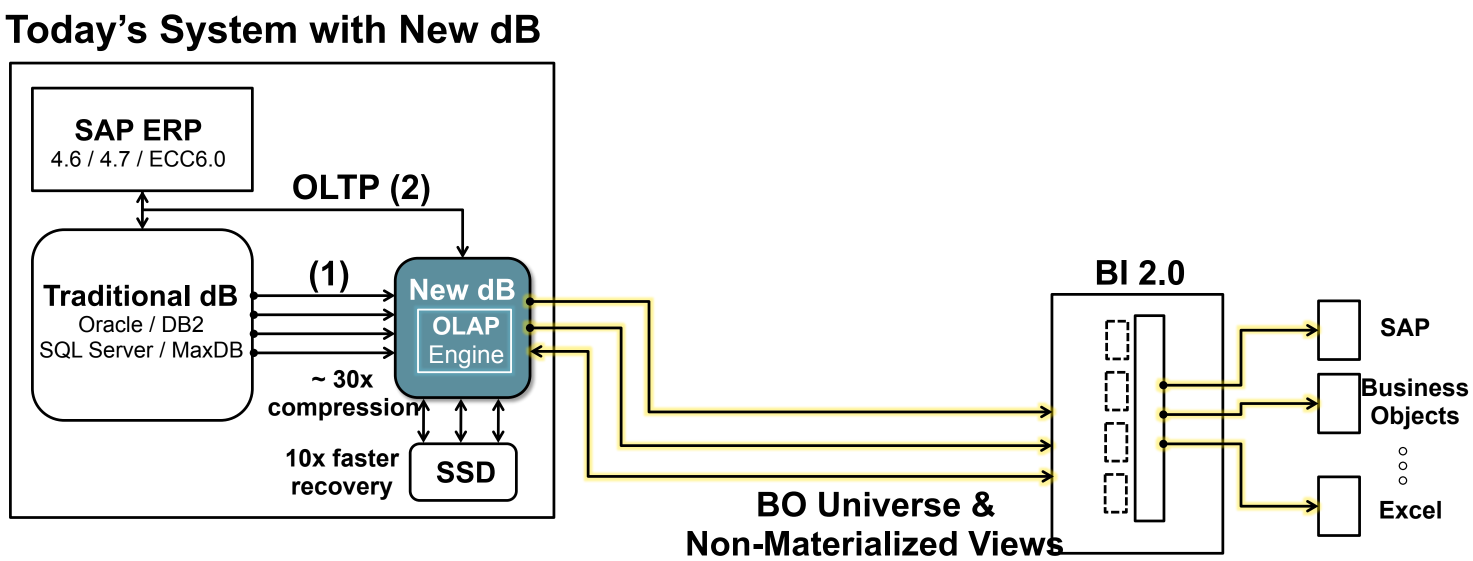
Bypass Solution for Transition

STEP 1: Install and run the in-memory database in parallel

STEP 2: Re-create traditional-style BI in main memory

STEP 3: Introduce next-gen BI running in parallel with no materialized views

STEP 4: Eliminate all the traditional BI, virtualize all in-memory BI, using non-materialized views



Bypass Solution for Transition

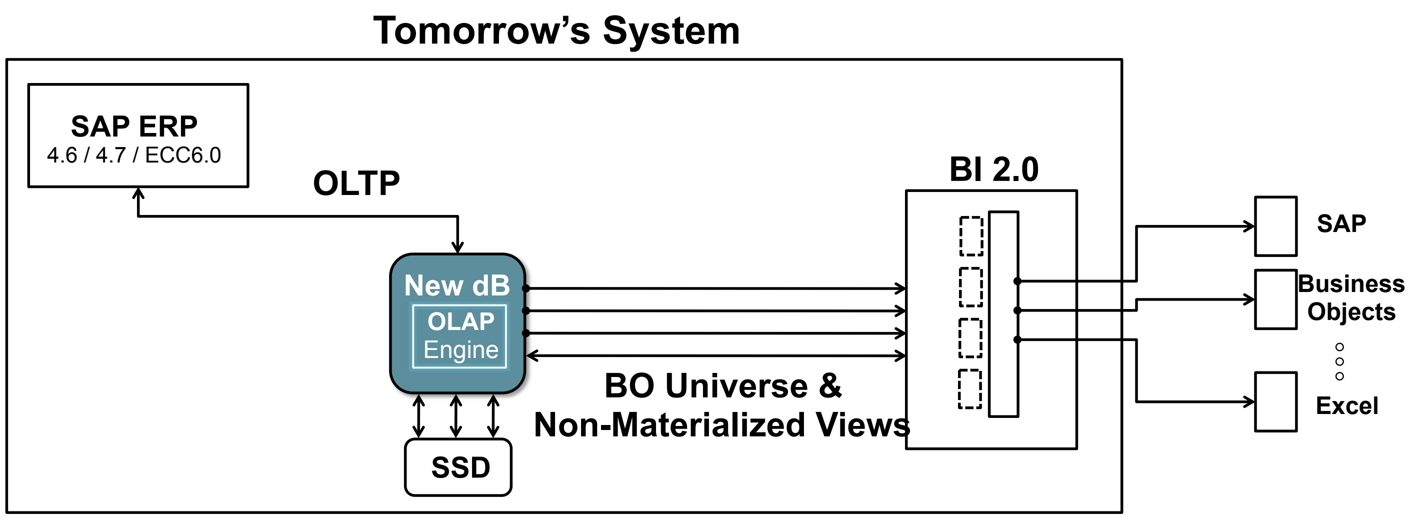
STEP 1: Install and run the in-memory database in parallel

STEP 2: Re-create traditional-style BI in main memory

STEP 3: Introduce next-gen BI running in parallel with no materialized views

STEP 4: Eliminate all the traditional BI, virtualize all in-memory BI, using non-materialized views

STEP 5: Eliminate all disk storage and run directly on the in-memory store



Bypass Solution for Transition

STEP 1: Install and run the in-memory database in parallel

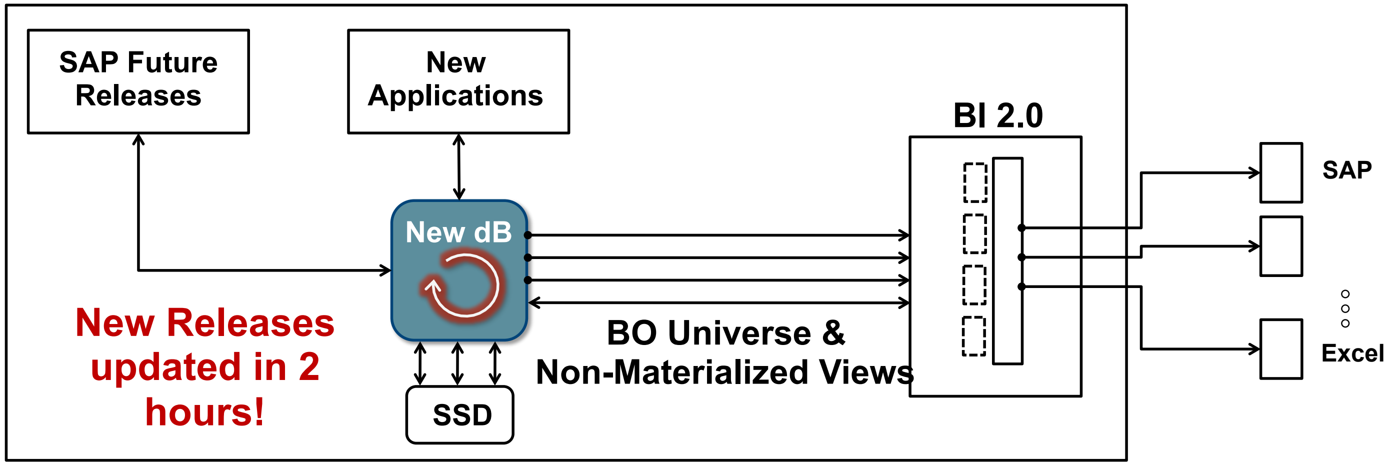
STEP 2: Re-create traditional-style BI in main memory

STEP 3: Introduce next-gen BI running in parallel with no materialized views

STEP 4: Eliminate all the traditional BI, virtualize all in-memory BI, using non-materialized views

STEP 5: Eliminate all disk storage and run directly on the in-memory store

STEP 6: Roll-out new releases (new tables, new attributes) and new applications without disruption



Thank You!