The problem with bad framing

Leads to bad assumptions about use, inappropriate features, poor understanding of substitutability and the impacts it will have.
The data lake
The data lake after a little while
Data Exhaust
Data is the new oil
Reality: data is a choice
Technical Debt: the gist of this talk

tek-ni-kuh l det: the cost that accrues due to decisions made in software design and coding.

Look at the choices and mistakes in development:

<table>
<thead>
<tr>
<th>Purposeful choices to optimize schedule, budget, satisfaction</th>
<th>Missed requirements, poor code quality, poor design</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intentional</td>
<td>Unintentional</td>
</tr>
</tbody>
</table>
The cost of some choices can be dealt with in the short term (e.g. the next sprint) and some only in the long term (redesign, start over)

<table>
<thead>
<tr>
<th>Short term</th>
<th>Long term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mostly about the application code</td>
<td>Mostly about architecture, design, and infrastructure</td>
</tr>
</tbody>
</table>

Copyright Third Nature, Inc.
If you enter into decisions knowing the true nature of your coding alternatives, you will be better off.

Green: these are deliberate, the tradeoffs known
Yellow: these are minor defects
Red: these are the things that kill a system

<table>
<thead>
<tr>
<th>Short term</th>
<th>Code choices</th>
<th>Code flaws (i.e. bugs)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intentional</td>
<td>Unintentional</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Long term</th>
<th>Design choices</th>
<th>Design flaws</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Intentional</td>
<td>Unintentional</td>
</tr>
</tbody>
</table>
Technical Debt can’t be avoided  
(but it can be managed)

Sometimes you think it’s intentional: incremental design  
Long term debts can only be dealt with through planning

<table>
<thead>
<tr>
<th>Short term</th>
<th>Agile methods</th>
<th>Development methods</th>
</tr>
</thead>
<tbody>
<tr>
<td>Long term</td>
<td>Redesign</td>
<td>Experience, education</td>
</tr>
</tbody>
</table>

Intentional | Unintentional
Technical Debt can’t be avoided

What you believe about the technology underlying your system has a big influence on design choices, so the focus of this talk is on architecture and design with the hope it will help reduce or avoid long term debt.

<table>
<thead>
<tr>
<th>Short term</th>
<th>Long term</th>
</tr>
</thead>
<tbody>
<tr>
<td>Intentional</td>
<td>Unintentional</td>
</tr>
</tbody>
</table>

Let’s move this to the left
“There is nothing new under the sun but there are lots of old things we don't know.”

Ambrose Bierce
Things I hear

Why is the system slow? *The database*

Why doesn’t the system scale? *The database*

Why is the system so expensive? *The database*
It is a poor carpenter who blames his tools*

*but sometimes it is the tools
Why do we need an entirely new model for the data management layer?

Because scalability aka “big”
Because “unstructured”
Because flexibility
What is best in life?
What is best in life?

Crush the vendors. See them driven before you. To hear the lamentation of their salespeople.
What is best in life?

Wrong!
Scalability? Just add hardware

"The most amazing achievement of the computer software industry is its continuing cancellation of the steady and staggering gains made by the computer hardware industry." — Henry Peteroski

After all, your database is web scale, isn’t it?
Parallelism solves everything?

![Graph showing speedup with number of processors and overhead.]

- **Speedup** (Amdahl's Law)
- **Number of processors**
- **10% overhead**
- **Linear**
How vendors demonstrate “Linear Scalability”

This is the part of the chart most vendors show.

If you’re lucky they leave the bottom axis on so you know where their system flatlines.
Parallelism

You need to coordinate transactions in a distributed environment. Coordination is the enemy of scale. Here’s math.

What needs coordination? Updates, inserts and deletes. Hence ACID compliance.

But there are other ways.

Amdahl's Law

\[ T_p = \left( \frac{\%S}{N} + \frac{1 - \%S}{N} \right) * T_s \]

Speedup = \[ \frac{T_s}{T_p} \]

<table>
<thead>
<tr>
<th>Tp</th>
<th>Parallel runtime</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ts</td>
<td>Serial runtime</td>
</tr>
<tr>
<td>%S</td>
<td>Percentage of time spent in serial code</td>
</tr>
<tr>
<td>N</td>
<td>Number of processors</td>
</tr>
</tbody>
</table>
The CAP theorem and ACID vs BASE

Visual Guide to NoSQL Systems

- **Availability**: Each client can always read and write.
- **Consistency**: All clients always have the same view of the data.
- **Partition Tolerance**: The system works well despite physical network partitions.

**Data Models**
- Relational (comparison)
  - Key-Value
  - Column-Oriented/Tabular
  - Document-Oriented

**Examples**
- **CA**: RDBMSs (MySQL, Postgres, etc), Aster Data, Greenplum, Vertica
- **CP**: BigTable, Hypertable, Hbase, MongoDB, Terrastore, Scalaris, Berkeley DB, MemcacheDB, Redis
- **AP**: Dynamo, Voldemort, Tokyo Cabinet, KAI, Cassandra, SimpleDB, CouchDB, Riak

**Additional Resources**
http://blog.nahurst.com/visual-guide-to-nosql-systems
Simplifying ACID vs BASE

Trade with confidence on the world's largest Bitcoin exchange!

Mt.Gox is the world's most established Bitcoin exchange. You can quickly and securely trade bitcoins with other people around the world with your local currency!

“transaction malleability” is a nice way of saying “broken”

Remember: it’s a poor carpenter who blames his tools.
90% of EVERYTHING is crap
Eventual consistency: a little bird told me...

Let’s analyze some data with long-running queries

Remember: it’s a poor carpenter who blames his tools.
Why doesn’t your database scale?
Just add hardware?

No amount of hardware will make incorrectly coded software run in parallel.

Declarative languages make this easier by turning the problem over to the computer to resolve.

Guess which runs in parallel:

Open cursor
Loop
  Fetch row
  Do-things
Join table 2 row
Insert result
End loop

INSERT INTO table (SELECT do-things(cols)
  FROM table2, table
  WHERE x=y)
90% of EVERYTHING is crap
I can’t get MySQL to scale

therefore

Relational databases don’t scale

therefore

We must use NoSQL* for everything

*including Hadoop and related
Sharding, Making Mess in One Easy Step

Sharding is basically partitioning applied across multiple database servers, faking a distributed DB. Each node holds an independent and (hopefully) self-consistent portion of the database.

Good as long as queried data lives on a single node.

One large database is carved into several smaller databases.
Sharding, Databases and Queries

What happens when you need to scan a full table or join tables across nodes? Multiple queries and stitching at the application level.

Sharding works well for fixed access paths, uniform query plans, and data sets that can be isolated. Mainly this describes an OLTP-style workload.
What application is this hardware and database topology designed for?

- Load balancer
- Web server
- Web server
- UI cache
- UI cache
- service
- service
- service
- Distributed multi-node DB
“In pioneer days they used oxen for heavy pulling, and when one ox couldn't budge a log, they didn't try to grow a larger ox. We shouldn't be trying for bigger computers, but for more systems of computers.”

Grace Hopper
What is the nature of the workloads?

Two workloads, two not dissimilar architectures:
- Load-balanced front ends
- Distributed caching layers
- Scalable distributed parallel databases

The nature of the OLTP and BI workloads is very different above the hardware and below the application. This is where the moving parts are. Forcing them into one platform is almost impossible at scale*
A key point worth remembering:

Performance over size <> performance over complexity

OLTP performance is mostly related to transaction coordination challenges under high concurrency.
BI performance is mostly related to data volume and query complexity.
Analytics performance is about the intersection of these with computational complexity.
<table>
<thead>
<tr>
<th>Workloads</th>
<th>OLTP</th>
<th>BI</th>
<th>Analytics</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Access</strong></td>
<td>Read-Write</td>
<td>Read-only</td>
<td>Read-mostly</td>
</tr>
<tr>
<td><strong>Predictability</strong></td>
<td>Fixed path</td>
<td>Unpredictable</td>
<td>All data</td>
</tr>
<tr>
<td><strong>Selectivity</strong></td>
<td>High</td>
<td>Low</td>
<td>Low</td>
</tr>
<tr>
<td><strong>Retrieval</strong></td>
<td>Low</td>
<td>Low</td>
<td>High</td>
</tr>
<tr>
<td><strong>Latency</strong></td>
<td>Milliseconds</td>
<td>&lt;seconds</td>
<td>msecs to days</td>
</tr>
<tr>
<td><strong>Concurrency</strong></td>
<td>Huge</td>
<td>Moderate</td>
<td>1 to huge</td>
</tr>
<tr>
<td><strong>Model</strong></td>
<td>3NF, nested object</td>
<td>Dim, denorm</td>
<td>BWT</td>
</tr>
<tr>
<td><strong>Task size</strong></td>
<td>Small</td>
<td>Large</td>
<td>Small to huge</td>
</tr>
</tbody>
</table>
The big change in the IT market isn’t technology, it’s architecture.
We are in a transitional phase in IT architecture

<table>
<thead>
<tr>
<th>Architecture</th>
<th>Then</th>
<th>State of Practice</th>
<th>Now, forward</th>
</tr>
</thead>
<tbody>
<tr>
<td>Timeshare</td>
<td>Core TXs</td>
<td>All TXs, some</td>
<td>All data</td>
</tr>
<tr>
<td>Client/server</td>
<td>Core TXs</td>
<td>All TXs, some</td>
<td>All data</td>
</tr>
<tr>
<td>Cloud</td>
<td>Core TXs</td>
<td>All TXs, some</td>
<td>All data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Rate of change</th>
<th>Slow</th>
<th>Rapid</th>
<th>Continuous</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core TXs</td>
<td>Core TXs</td>
<td>Core TXs</td>
<td>Core TXs</td>
</tr>
<tr>
<td>All TXs, some</td>
<td>All TXs, some</td>
<td>All TXs, some</td>
<td>All TXs, some</td>
</tr>
<tr>
<td>events</td>
<td>events</td>
<td>events</td>
<td>events</td>
</tr>
<tr>
<td>All data</td>
<td>All data</td>
<td>All data</td>
<td>All data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Uses</th>
<th>Few</th>
<th>Many</th>
<th>Everything</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core TXs</td>
<td>Core TXs</td>
<td>Core TXs</td>
<td>Core TXs</td>
</tr>
<tr>
<td>All TXs, some</td>
<td>All TXs, some</td>
<td>All TXs, some</td>
<td>All TXs, some</td>
</tr>
<tr>
<td>events</td>
<td>events</td>
<td>events</td>
<td>events</td>
</tr>
<tr>
<td>All data</td>
<td>All data</td>
<td>All data</td>
<td>All data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Latency</th>
<th>Daily+++</th>
<th>&lt; daily to minutes</th>
<th>Immediate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core TXs</td>
<td>Core TXs</td>
<td>Core TXs</td>
<td>Core TXs</td>
</tr>
<tr>
<td>All TXs, some</td>
<td>All TXs, some</td>
<td>All TXs, some</td>
<td>All TXs, some</td>
</tr>
<tr>
<td>events</td>
<td>events</td>
<td>events</td>
<td>events</td>
</tr>
<tr>
<td>All data</td>
<td>All data</td>
<td>All data</td>
<td>All data</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data platform</th>
<th>Uniprocessor</th>
<th>SMP, cluster</th>
<th>Shared nothing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Core TXs</td>
<td>Core TXs</td>
<td>Core TXs</td>
<td>Core TXs</td>
</tr>
<tr>
<td>All TXs, some</td>
<td>All TXs, some</td>
<td>All TXs, some</td>
<td>All TXs, some</td>
</tr>
<tr>
<td>events</td>
<td>events</td>
<td>events</td>
<td>events</td>
</tr>
<tr>
<td>All data</td>
<td>All data</td>
<td>All data</td>
<td>All data</td>
</tr>
</tbody>
</table>
How did we get here?

There’s a difference between having no past and actively rejecting it.
In the beginning: RMSs and pre-relational DBs

At first, common code libraries so there was reusability for file ops.

Problems:

- Portability across languages, OSs
- Queries of more than one file
- Concurrency
- No metadata, what’s in there? Who wrote it?

The databases brought things like recoverability, durability, ACID transactions. But they were rigid, prone to breakage.
Someone else always wants to use your data

Order Entry

Customer Service

Distribution

Accounts Receivable

Analysts & users

Order Database

Inventory Database

Receivables Database

Data Warehouse

Interface Program

Interface Program
"In an infinite universe, the one thing sentient life cannot afford to have is a sense of proportion.” – Douglas Adams
Context (multiple company supply chain)
A value chain diagram, showing the data supply chain for cheese. The side effects of a single bug can be massive.
Centralizing ERP & vendors didn’t solve the context problem; only increased cohesion. Now data lakes?

UPserts, imbecile.

You forget to Ltrim blanks again!
The miracle of pre-relational DB: schema

**Loose coupling** – the physical model of data structures and physical placement are no longer a program’s responsibility; data portability ensues.

**Reusability** – More than one program can access the same data, and no more custom coding for each application or OS

**Scalability** – Constraints of schema and typing reduce resource usage, have finer granularity for concurrent access, multiple online users.
Party like it’s 1985

Fastest TPC-B benchmark in 1985 was IMS running on an IBM 370, 100 TPS, 400 iops, 30 iops/disk

The best relational vendors could muster was 10 TPS

25 years later, SQLServer on an Intel box ran the TPC-B at 25,000 TPS, 100,000 iops, 300 iops/disk
SQL JOINS

1986: Wait, there's more than one?

SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key

SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key

SELECT <select_list>
FROM TableA A
INNER JOIN TableB B
ON A.Key = B.Key

SELECT <select_list>
FROM TableA A
LEFT JOIN TableB B
ON A.Key = B.Key
WHERE B.Key IS NULL

SELECT <select_list>
FROM TableA A
RIGHT JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL

SELECT <select_list>
FROM TableA A
FULL OUTER JOIN TableB B
ON A.Key = B.Key
WHERE A.Key IS NULL OR B.Key IS NULL

© C.L. Moffatt, 2008
What the optimizer does

It turns a SQL query into an optimal* execution plan for a parallel pipelined dataflow engine

1. Enumerate logically equivalent plans by applying equivalence rules
2. For each logically equivalent plan, enumerate all alternative physical query plans
3. Estimate the cost of each of the alternative physical query plans
4. Run the plan with lowest estimated overall cost

Diagram: David J. DeWitt
A simple 3 table join

SELECT C.name, O.num
FROM Orders O, Lines L, Customers C
WHERE C.City = "Copenhagen" AND L.status = "X"
    AND O.num = L.num AND C.cid = O.cid

Number of logical plans: 9
Ways to join (hash, merge, nested): 3
For each plan, there are multiple physical plans: 36
That makes a total of 324 physical plans, the efficiency of which changes based on cardinality.
NoSQL tradeoffs?

“Query optimization is not rocket science. When you flunk out of query optimization, we make you go build rockets.”
In NoSQL Land, Optimizer is You!

You did review each plan in your MapReduce job, right?.
Tradeoffs: In NoSQL the DBMS is in your code

SQL database

Services provided
- Standard API/query layer*
- Transaction / consistency
- Query optimization
- Data navigation, joins
- Data access
- Storage management

NoSQL database

Application

Database

Application

Database

Anything not done by the DB becomes a developer’s task.
The relational database is the franchise technology for storing and retrieving data, but...

1. Global, static schema model
2. No rich typing system
3. No management of natural ordering in data
4. Pretends to separate logical and physical schema, but it’s partial
5. Limited API in atomic SQL statement syntax & simple result set return
6. Poor developer support (in languages, in IDEs, in processes)
Relational: a good conceptual model, but a prematurely standardized implementation

What I did not list:
Scalability and performance

The relational database is the franchise technology for storing and retrieving data, but...

1. Global, static schema model
2. No rich typing system
3. No management of natural ordering in data
4. Many are not a good fit for network parallel computing, aka cloud
5. Limited API in atomic SQL statement syntax & simple result set return
6. Poor developer support
Parallel Efficiency and Platform Costs

**Structured**

- **SQL**
  - Production Data Warehousing
  - Large Concurrent User-base
  - Data Warehouse
  - Enterprise-class System
  - 6+PB

**Semi-Structured**

- **SQL++**
  - Contextual-Complex Analytics
  - Deep, Seasonal, Consumable Data Sets
  - Data Warehouse + Behavioral
  - Low End Enterprise-class System
  - 40+PB

**Unstructured**

- **Java/C**
  - Structure the Unstructured
  - Detect Patterns
  - Commodity Hardware System
  - Hadoop
  - 20+PB

**Concurrent Users**

- 500+
- 150+
- 5-10
Platform Metrics for Table Scan and Sum, Hadoop vs Teradata

**System Unit Cost (norm)**
- EDW: 20
- Singularity: 10
- Hadoop: 5

**Units Consumed**
- EDW: 2
- Singularity: 0.5
- Hadoop: 2

**Job/Query Cost (norm)**
- EDW: 0.2
- Singularity: 0.1
- Hadoop: 0.2

**Latency**
- EDW: 10
- Singularity: 50
- Hadoop: 200

**Parallel Efficiency**
- EDW: 80
- Singularity: 60
- Hadoop: 80
Relational schema inflexibility

Change my-crappy-code-with-select-\* everywhere in it:
ALTER TABLE really_big

    ADD COLUMN omg_wtf

I meant...

create new table, back up old table, wait, load new table, wait 4ever, drop new table, recreate new table with new column default, reload new table, wait 4ever

On each shard...
The Developer View of DBAs
The DBA view of developers
Flexibility – an experience in query

The problem with many of these databases is tight coupling between a program and data structures.

The physical model leaks into the logical with potentially career-ending effects if the DB is used for the wrong thing.

It’s a poor carpenter who blames his tools. Or the users.
Schema on write vs schema on read

Match the shape to the hole

or

Match the hole to the shape

Predicate schemas for write flexibility (agility) and speed
Key schema flexibility tradeoff for data management

Global validation vs contextual validation = Strict rules vs lenient rules = Write rules vs read rules
When to use implicit schema?

Use **implicit** when:
- You can hide the persistence of your data behind a service
- Nobody will ever want access to that data except you
- When data dies with the code
- You need to write data at a very high rate
- Your data sources change or are variable

Use **explicit** when:
- you need to send data to another application
- when more than one application (or person) needs to use data
- when data lives longer than your code
- When the data is regular
- When the sources and structure do not change
- When querying is more important than writing
Unstructured data isn’t really unstructured: language has structure. So do images, audio, video. They can contain traditional structured data elements. The problem is that the content is unmodeled.

**Conclusion:** a database must cope with more complex data structures, storage and processing.
Technologies are not perfect replacements for one another. Often not better, only different. When replacing the old with the new (or ignoring the new over the old) you make *tradeoffs*, and usually you won’t see them for a long time. There is no silver bullet.
Unintended consequences
Away from “one throat to choke”, back to best of breed

Tight coupling leads to slow change. The market is not in the tight coupling phase.

In a rapidly evolving market, componentized architectures, modularity and loose coupling are favorable over monolithic stacks, single-vendor architectures and tight coupling.
Think like an architect, not like a consumer

The technology providers are selling you what *they* have, not necessarily what *you* need.

Follow the goals of the business.

Translate the goals into capabilities needed and match those to the architecture required.
How we develop best practices: survival bias

We don’t need best practices today, we need worst failures.
The big data revolution, more of an evolution
References (things worth reading on the way home)


A query language for multidimensional arrays: design, implementation and optimization techniques, SIGMOD, 1996


“Amorphous Data-parallelism in Irregular Algorithms”, Keshav Pingali et al


About Third Nature

Third Nature is a research and consulting firm focused on new and emerging technology and practices in business intelligence, analytics and performance management. If your question is related to BI, analytics, information strategy and data then you’re at the right place.

Our goal is to help companies take advantage of information-driven management practices and applications. We offer education, consulting and research services to support business and IT organizations as well as technology vendors.

We fill the gap between what the industry analyst firms cover and what IT needs. We specialize in product and technology analysis, so we look at emerging technologies and markets, evaluating technology and how it is applied rather than vendor market positions.
CC Image Attributions

Thanks to the people who supplied the creative commons licensed images used in this presentation:

round hole square peg - https://www.flickr.com/photos.epublicist/3546059144