ADDRESSING HUMAN BOTTLENECKS IN BIG DATA

JOSEPH M HELLERSTEIN

Berkeley
OUTLINE

1. A Deal with the Devil

2. Distributed Programming
   CALM and <~ bloom

3. Data Wrangling
   Predictive Interaction & TRIFACTA
MOORE’S LAW

2010: 45 years of Moore’s Law

- 1 Billion-fold growth
- Quantitative becomes qualitative

Big Data: the inevitable conclusion
I CAN GIVE YOU POWER

All the Data you desire
All the Storage you desire
All the Compute you desire
AT WHAT COST?

Loss of illusions

- Sequential programs
- Single-copy state
- Reliable components
- Data modeling
HUMAN COST

1. Distributed Programming
2. Data Wrangling

Who has the skills?
And are they being sacrificed?

The signal challenge of our time...
By relieving the brain of all unnecessary work, a good notation sets it free to concentrate on more advanced problems, and, in effect, increases the mental power of the race.

—Alfred North Whitehead
DOMAIN SPECIFIC LANGUAGES

High Level
Little
Bespoke

SQL, VBA, Lex/Yacc, LaTeX, HTML, OpenGL, Bash, Datalog, ...
A user immersed in a domain already knows the domain semantics. All the DSL designer needs to do is provide a notation to express that semantics. More concise, quicker to write, easier to maintain, easier to reason about, accessible to non-programmers. Sounds easy...
2 SIDES OF BIG DATA

THE DEVELOPER

NoSQL

THE DATA ANALYST

Hadoop
OUTLINE

1. A Deal with the Devil

2. Distributed Programming
   CALM and \(<~\) bloom

3. Data Wrangling
   Predictive Interaction & TRIFACTA
Non-trivial software is distributed

Distributed programming is hard
to (software engineering)
× (parallelism + asynchrony + failure)

PROGRAMMING TODAY
ORDERLY COMPUTING

ORDER
- LIST of Instructions
- ARRAY of Memory

STATE
- Mutation in time
ORDERLY COMPUTING

ORDER
- LIST of Instructions
- ARRAY of Memory

STATE
- Mutation in time
ORDERLY COMPUTING

ORDER
- LIST of Instructions
- ARRAY of Memory

STATE
- Mutation in time
ORDERLY COMPUTING

ORDER
- LIST of Instructions
- ARRAY of Memory

STATE
- Mutation in time
DISTRIBUTED PROGRAMMING

REPLICATED for availability
PARTITIONED to scale out
ASYNCHRONOUS for performance

All this ... in Java.
ORDERLY CODE IN A DISORDERLY WORLD
WHAT COULD GO WRONG?
Model: Distributed State (R/W)
- i.e. traditional von Neumann programming
- i.e. NoSQL (NoDSL!)

Desire: Eventual Consistency
- of distributed state (as a proxy for program intent)

Mechanism: Linearization (SSI)
- E.g. Paxos distributed log
<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>1</td>
</tr>
</tbody>
</table>

- Two paper bags
- Two apples in the bags
- A stopwatch showing 60 seconds
<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>🍎</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>🍎</td>
<td>1</td>
</tr>
<tr>
<td>Item</td>
<td>Count</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>🍎</td>
<td>0</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>🍎</td>
<td>0</td>
</tr>
<tr>
<td>Item</td>
<td>Count</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>Apple</td>
<td>0</td>
</tr>
<tr>
<td>Apple</td>
<td>0</td>
</tr>
<tr>
<td>Item</td>
<td>Count</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td><img src="apple.png" alt="Apple" /></td>
<td>1</td>
</tr>
</tbody>
</table>

![Timer](timer.png)
<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="apple1.png" alt="Apple" /></td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td><img src="apple2.png" alt="Apple" /></td>
<td>1</td>
</tr>
<tr>
<td>Item</td>
<td>Count</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>Apple</td>
<td>1</td>
</tr>
<tr>
<td>Orange</td>
<td>1</td>
</tr>
<tr>
<td>Item</td>
<td>Count</td>
</tr>
<tr>
<td>------</td>
<td>-------</td>
</tr>
<tr>
<td>Apple</td>
<td>1</td>
</tr>
<tr>
<td>Orange</td>
<td>1</td>
</tr>
</tbody>
</table>
Questions

- Do multiple agents need to coordinate?
- On which lines of Java code?

Variations

- Replicated & Partitioned.
- Unreliable network, agents

Software testing and maintenance
YESTERDAY, ON TWITTER

**damien mutant Qatz** @damienkatz · 12h
Distributed systems, don’t read the literature. Most of it is outdated and unimaginine. Invent and reinvent. The field is fertile. Really.

**Puppy Agenda** @aphyr · 12h
@damienkatz what could possibly go wrong

Damien Katz, CouchDB creator(@damienkatz)
Kyle Kingsbury, Factual (@aphyr)
Yesterday, on Twitter

damien mutant Qatz @damienkatz · 10h
@aphyr If you narrow your distributed actions to commutative. Two people edit the same wiki page. Show the algorithm for reaching consensus.

Puppy Agenda @aphyr · 10h
@damienkatz You can't do consensus in Couch *because* reads are noncommutative.

damien mutant Qatz @damienkatz · 10h
@aphyr Explain like I'm 5 what you mean by Commutative reads.
YESTERDAY, ON TWITTER

**damien mutant Qatz** @damienkatz · 8h
@aphyr You are pointing out properties. Not problems. Again, what PROBLEM are you trying to solve?

**Puppy Agenda** @aphyr · 8h
@damienkatz You said you had a consensus algorithm. This fails the consensus invariants trivially.

**Puppy Agenda** @aphyr · 8h
@damienkatz I'm starting to suspect that not only do you not *have* a consensus algorithm; you don't even know what the problem *means*.

**damien mutant Qatz** @damienkatz · 8h
@aphyr You are probably right. Explain the PROBLEM you are trying to solve.
YESTERDAY, ON TWITTER

Duchess of V a. a. → a @bitemyapp · 10h
I don’t see why @aphyr is picking on @damienkatz - Damien is taking the pragmatic approach.

Puppy Agenda @aphyr · 10h
@damienkatz To my knowledge Couch a.) punts on conflict resolution and b.) cites the Dynamo paper in its replication docs.

Toby Hede @tobyhede · 10h
@bitemyapp @aphyr @damienkatz We have a quorum of badly designed systems that are laughably naive. Similar to roll-your-own-crypto.

Tom Santero @tsantero · 10h
@bitemyapp because encouraging people to ignore the wealth of information provided by research is irresponsible /cc @aphyr @damienkatz

Chris @leftside · 10h
@bitemyapp @aphyr @damienkatz "Ignore decades of accumulated knowledge and build your own broken shit from scratch" is not pragmatism.
WHAT DO WE MAKE OF ALL THIS

Little developer consensus on consensus...

But some widely-quoted negative results from an academic practitioner
BREWER’S CAP THEOREM

A NEGATIVE RESULT FOR CLASSICAL TREATMENTS

It is impossible in the asynchronous network model to implement a read/write data object that guarantees the following properties:

Consistency

Availability

Partition-tolerance

[Gilbert and Lynch 2002]
PARTITION IS NOT THE PROBLEM

Consistency is possible
But at what cost? Latency.
Waits-For: Global Coordination
“The first principle of successful scalability is to batter the consistency mechanisms down to a minimum, move them off the critical path, hide them in a rarely visited corner of the system, and then make it as hard as possible for application developers to get permission to use them”

— James Hamilton (IBM, MS, Amazon), quoted in [Birman/Chockler 2009]
THE WRONG SIDE OF PROBABILITY
"The large standard deviation in write latencies is caused by a pretty fat tail due to lock conflicts. The even larger standard deviation in read latencies is partially due to the fact that Paxos leaders are spread across two data centers, only one of which has machines with SSDs."


<table>
<thead>
<tr>
<th>operation</th>
<th>latency (ms)</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>mean</td>
<td>std dev</td>
</tr>
<tr>
<td>all reads</td>
<td>8.7</td>
<td>376.4</td>
</tr>
<tr>
<td>single-site commit</td>
<td>72.3</td>
<td>112.8</td>
</tr>
<tr>
<td>multi-site commit</td>
<td>103.0</td>
<td>52.2</td>
</tr>
</tbody>
</table>
What do people do?
Mutable State is an “anti-pattern”
Pattern: Event Log Shipping
<table>
<thead>
<tr>
<th>Item</th>
<th>Count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Apple</td>
<td>1</td>
</tr>
<tr>
<td>Orange</td>
<td>1</td>
</tr>
</tbody>
</table>

- ✔
1. Practitioners do make progress
   - Pay attention to best practices and maxims

2. Theory Questions
   - When is this kind of thing possible (correct)?
   - What to do when impossible?

3. DSL Design and SW Engineering
   - “Disorderly” language design
   - Techniques to enforce/check good patterns

Goal: Maxims $\rightarrow$ Theorems $\rightarrow$ Software
A THEORY QUESTION

Do we need coordination?
When can we avoid it?

NOT a question about reads, writes, and race conditions.
It’s a question of the expressivity of a language construct!
THE CALM THEOREM

Monotonic => Consistent
- Accumulative, disorderly computing.
- Confluence.

¬Monotonic => ¬Consistent
- Inherent non-monotonicity requires sealing
- The reason for coordination

[Conjecture: Hellerstein PODS ’09, “The Declarative Imperative”]
MONOTONICITY

Monotonic Code

Information accumulation
- The more you know, the more you know
- E.g. map, filter, join

Non-Monotonic Code

Belief revision
- New inputs can change your mind; need to “seal” input
- E.g. reduce, aggregation, negation, state update
SEALING, TIME, SPACE

Non-monotonicity: sealing a world

$$\neg \exists x \in S ( p(x) )$$

$$\iff \forall x \in S ( \neg p(x) )$$

Time: a mechanism to seal fate:

Before and after

“Time is what keeps everything from happening at once.”
— Ray Cummings
SEALING, TIME, SPACE

Non-monotonicity: sealing a world

\[ \neg \exists x \in S ( p(x) ) \]

\[ \iff \forall x \in S ( \neg p(x) ) \]

Time: a mechanism to seal fate

Space: multiple perceptions of time

Coordination: sealing in time and space
Non-monotonicity: sealing a world

\[ \neg \exists x \in X (p(x)) \iff \forall x \in X (\neg p(x)) \]

Time: a mechanism to seal fate
Space: multiple perceptions of time
Coordination: sealing in time and space

SEALING, TIME, SPACE
INTUITION: SETS

State change in the land of sets
MUTABLE SETS

- Introduce time into each relation
  - shirt(‘Joe’, ‘black’, 1)

- Persistence is induction
  - shirt(x, y, t+1) <= shirt(x, y, t)

- Mutation via negation
  - shirt(x, y, t+1) <= shirt(x, y, t), \neg del\_shirt(x, y, t)
  - shirt(x, z, t+1) <= new\_shirt(x, z, t)

“Time is what keeps everything from happening at once.”
— Ray Cummings

[Statelog: Ludäscher 95, Dedalus: Alvaro ‘11]
RECALL THE QUESTION

Do we need coordination?
When can we avoid it?
Do we need time?
When can we collapse it?

We need time to seal fate, avoid paradox.
We need time for non-monotonic operations.
Monotonic $\Rightarrow$ Consistent
$\neg$Monotonic $\Rightarrow$ $\neg$Consistent

Guard non-monotonic operations with seals.

[Conjecture: Hellerstein PODS '09, “The Declarative Imperative”]

Also:
- CRON Conjecture
- Coordination Complexity
THEORETICAL RESULTS

CALM Proofs
- Abiteboul, et al.: M => C [PODS ’11]
- Ameloot, et al.: CALM [PODS ’11, JACM ’13]
- Ameloot, et al.: Weaker forms of M => C [PODS ’14 best paper]
- Marczak, et al.: M => C, NM+Coord => C [Datalog 2.0 ’12]

CRON (Proofs & Refutations)
- Ameloot, et al.: [JCSS ’15]

Coordination Complexity: MP Model
- Koutris & Suciu (min-coordination & LB): [PODS ’11]
- Beame et al. (minimizing replication): [PODS ’13]

More! See survey by Ameloot [SIGMOD Record 6/14]
1. Practitioners do make progress
   - Pay attention to best practices and maxims

2. Theory Questions
   - When is this kind of thing possible (correct)?
   - What to do when impossible?

3. DSL Design and SW Engineering
   - “Disorderly” language design
   - Techniques to enforce/check good patterns
REPLICATED for availability
PARTITIONED to scale out
ASYNCHRONOUS for performance

a disorderly data-centric DSL

All this ... in Java.
AN ONGOING AGENDA

10 years of DSL and systems experimentation:

- Declarative Networking: [http://declarativity.net](http://declarativity.net) [Loo et al., CACM ’09]
- BOOM Project: [http://boom.cs.berkeley.edu](http://boom.cs.berkeley.edu)

2011/2013: “Programming the Cloud” undergrad course

- [http://programthecloud.github.com](http://programthecloud.github.com)
A disorderly language of data, space and distributed time

Based on Alvaro’s Dedalus logic

[Hellerstein, CIDR ‘11]
[Conway, SOCC ‘12]
http://bloom-lang.org
OPERATIONAL MODEL

Nodes with local clocks, state
Timestep at each node:

1. local updates
2. system events
3. network

bloom rules
atomic, local

now
next
network
<object>  <merge>  <expression>
<table>
<thead>
<tr>
<th></th>
<th>&lt;object&gt;</th>
<th>&lt;merge&gt;</th>
<th>&lt;expression&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>persistent</td>
<td>table</td>
<td></td>
<td></td>
</tr>
<tr>
<td>transient</td>
<td>scratch</td>
<td></td>
<td></td>
</tr>
<tr>
<td>networked transient</td>
<td>channel</td>
<td></td>
<td></td>
</tr>
<tr>
<td>scheduled transient</td>
<td>periodic</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Persistent</td>
<td>Table</td>
<td>Merge</td>
<td>Expression</td>
</tr>
<tr>
<td>------------------</td>
<td>-------------</td>
<td>---------------------</td>
<td>-----------------------------</td>
</tr>
<tr>
<td>Transient</td>
<td>Scratch</td>
<td>&lt;= now</td>
<td>Map, flat_map</td>
</tr>
<tr>
<td>Networked transient</td>
<td>Channel</td>
<td>&lt;+ next</td>
<td>Reduce, group, argmin/max</td>
</tr>
<tr>
<td>Scheduled transient</td>
<td>Periodic</td>
<td>&lt;~ async</td>
<td>(r * s).pairs</td>
</tr>
<tr>
<td></td>
<td></td>
<td>&lt;= del_next</td>
<td>Empty? include?</td>
</tr>
</tbody>
</table>
### SYNTAX

**<object>**

<table>
<thead>
<tr>
<th>persistent</th>
<th>table</th>
</tr>
</thead>
<tbody>
<tr>
<td>transient</td>
<td>scratch</td>
</tr>
<tr>
<td>networked transient</td>
<td>channel</td>
</tr>
<tr>
<td>scheduled transient</td>
<td>periodic</td>
</tr>
<tr>
<td>persistent</td>
<td>&lt;lattice&gt;</td>
</tr>
</tbody>
</table>

**<merge>**

| <= | now |
| <+ | next |
| <~ | async |
| <= | del_next |

**<expression>**

| map, flat_map |
| reduce, group, argmin/max |
| (r * s).pairs |
| empty? include? |
| monotone fns, morphisms, non-monotone fns |
module ChatServer
  state do
    table :nodelist
    channel :mcast; channel :connect
  end

  bloom do
    nodelist <= connect.payloads
    mcast <= (mcast*nodelist).pairs do |m,n|
      [n.key, m.val]
    end
  end
end
Monotone function: 
set $\rightarrow$ increase-int

Monotone function: 
increase-int $\rightarrow$ boolean

Set 
(Merge = \textit{Union})

Increasing Int 
(Merge = \textit{Max})

Boolean 
(Merge = \textit{Or})
module ChatServer
state do
    table :nodelist
    channel :mcast; channel :connect
end

end

module bloom
do
    nodelist <= connect.payloads
    mcast ~ (mcast*nodelist).pairs do lm,nl
        [n.key, m.val]
    end
end

if quorum_done
    counter <= nodelist.size
    quorum_done <= counter.gt_eq(QUORUM_SIZE)
**VECTOR CLOCKS:**

bloom v. wikipedia

Initially all clocks are zero.

- Each time a process experiences an internal event, it increments its own logical clock in the vector by one.

- Each time a process prepares to send a message, it increments its own logical clock in the vector by one and then sends its entire vector along with the message being sent.

- Each time a process receives a message, it increments its own logical clock in the vector by one and updates each element in its vector by taking the maximum of the value in its own vector clock and the value in the vector in the received message (for every element).
TOOLS AND EXTENSIONS

Blazes: CALM analysis, Coordination Synthesis
[Alvaro et al., ICDE13]

Fault Tolerance Testing
[Alvaro, et al., 14]

Edelweiss: Event Exchange & GC
[Conway, et al., VLDB14]

Highly-Available Xactions
[Bailis, et al., VLDB14]

Invariant Confluence
[Bailis, et al., SIGMOD15]

And more... http://boom.cs.berkeley.edu
OUTLINE

1. A Deal with the Devil

2. Distributed Programming
   CALM and \(<~\) bloom

3. Data Wrangling
   Predictive Interaction & TRIFACTA
WHERE DOES TIME GO?
WHERE DOES TIME GO?

80% of the work in any data project is preparing the data.
Interview study of 35 analysts:

25 companies
Healthcare
Retail, Marketing
Social networking
Media
Finance, Insurance

Various titles
Data analyst
Data scientist
Software engineer
Consultant
Chief technical officer

[Kandel et al., VAST12]
In Data Science, 80% of time spent prepare data, 20% of time spent complain about need for prepare data.
“I spend more than half of my time integrating, cleansing and transforming data without doing any actual analysis. Most of the time I’m lucky if I get to do any ‘analysis’ at all.”

Friction

“Most of the time once you transform the data ... the insights can be scarily obvious.”

Lost potential
“It’s easy to just think you know what you are doing and not look at data at every intermediary step.

An analysis has 30 different steps. It’s tempting to just do this then that and then this. You have no idea in which ways you are wrong and what data is wrong.”

Interactivity and Visualization
THE DATA TRANSFORMATION PROBLEM

DATA SOURCE
Complexity

Business System Data
Machine Generated Data
Log Data
...

DATA TRANSFORMATION

DATA PRODUCT
Simplicity

Fraud Detection
Recommendations
Data Visualization
...

A PROGRAMMING PROBLEM
1990’s DSLs FOR DATA TRANSFORMATION

∀T → HARDER

[Lakshmanan, et al. VLDB96]
[Galhardas, et al. VLDB11]
POTTER’S WHEEL (2001): ENTER THE VISUAL

- Step-by-step DSL
- Immediate visual feedback
- Ongoing discrepancy detection
- Data lineage, redo/undo

Problem: Remaining burden of specification for users.

[Raman & Hellerstein, VLDB11]
VISUAL SPECIFICATION

Data Vis \(\xrightarrow{interaction}\) Visual Results

\(\xleftarrow{visualization}\) 

\(\xrightarrow{compilation}\) 

Data \(\xleftarrow{DSL}\) Results
TRADITIONAL BURDEN OF SPECIFICATION

1. User authors a draft transformation script

2. User tests the script on a small amount of data

3. User inspects output data to assess effects

Data Transformation Code

Visualization and Interaction
HINTS OF INTELLIGENT INTERFACES

Type-ahead uses context and data to predict your search term — and preview results.
WRANGLER (2011): ADD INTELLIGENCE

➔ Automatic inference of transforms
➔ Semantic data types
➔ Interactive history
➔ User Studies

[Kandel, et al. CHI 11]
PREDICTIVE INTERACTION

[Diagram showing the process of data visualization and interaction]

Data (input) -> Datavis (ambiguous interaction) -> Response

Visualization of probable Next Steps -> Visual Results

prediction -> disambiguation

[Heer, et al. CIDR15]
PREDICTIVE INTERACTION™

1. User highlights features of a data visualization
   - Data previews allow user to choose, adjust and confirm

2. ML methods predict distribution over DSL statements
   - Example DSL statements:
     ```python
     extract col: Description after: ' ' before: 'br'
     ```
   - Suggested transforms:
     - `extract col: Description after: ' ' before: 'br'`
     - `extract col: Description on: '# '`
     - `extract col: Description on: '#' after: ''`
     - `split col: Description on: ' ' before: 'br'`

3. Data Transformation Code

Visualization and Interaction
DATA TRANSFORMATION LOGIC

VISUAL PROFILING
Distributions, correlations, outliers

STRUCTURING
Columns, nesting, key/value, pivots, sparse-dense

CLEANING
Standardization, data dictionaries, deduplication

ENRICHMENT
Join, data augmentation

DISTILLATION
Sampling, filtering, aggregation, windows/sessions
A MICROCOSM

### SUGGESTED TRANSFORMS

<table>
<thead>
<tr>
<th>Source</th>
<th>Preview</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc</td>
<td>Screen_Detail</td>
</tr>
<tr>
<td><img src="image.png" alt="Image" /></td>
<td><img src="image.png" alt="Image" /></td>
</tr>
</tbody>
</table>

- **6 Categories**
  - `31 adtam_name=utarget1&adtam_source=dynamic&adtam_size=180x150` | `dynamic` | `Nokia`
  - `32 adtam_name=holidaypromo1&adtam_source=dynamic&adtam_size=300x250` | `dynamic` | `Nokia`
  - `33 adtam_name=utarget1&adtam_source=dynamic&adtam_size=180x150` | `dynamic` | `samsung`
  - `34 adtam_name=holidaypromo2&adtam_source=mobile&adtam_size=240x400` | `mobile`  | `Nokia`
A MICROCOSM
A MICROCOSM

<table>
<thead>
<tr>
<th>Source</th>
<th>Screen_Detail</th>
<th>Preview</th>
</tr>
</thead>
<tbody>
<tr>
<td>abc</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th></th>
<th>abc</th>
<th>Screen_Detail</th>
<th>abc</th>
<th>Device</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6 Categories</td>
<td></td>
<td>2 Categories</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>31</td>
<td>adtam_name=utarget1&amp;adtam_source=dynamic&amp;adtam_size=180x150</td>
<td></td>
<td>dynamic</td>
<td>Nokia</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>adtam_name=holidaypromo1&amp;adtam_source=dynamic&amp;adtam_size=300x250</td>
<td></td>
<td>dynamic</td>
<td>Nokia</td>
<td></td>
</tr>
<tr>
<td>33</td>
<td>adtam_name=utarget1&amp;adtam_source=dynamic&amp;adtam_size=180x150</td>
<td></td>
<td>dynamic</td>
<td>samsung</td>
<td></td>
</tr>
<tr>
<td>34</td>
<td>adtam_name=holidaypromo2&amp;adtam_source=mobile&amp;adtam_size=240x400</td>
<td></td>
<td>mobile</td>
<td>Nokia</td>
<td></td>
</tr>
</tbody>
</table>
TECHNICAL PHILOSOPHY

The subtle trifecta of **People • Data • Computation**

**HCI**
- Affordances to gather *features* (input)
- Feedback on *representation* (DSL) and *domain* (previews)

**DB**
- An interactive DSL for Data Transformation
- Visually interpretable, easily readable and *learnable*

**ML**
- ML = *Representation* + Evaluation + Optimization
- Easily the most important factor is the *features* used
TAKEAWAYS

Methodology
1. Listen to practitioners
2. Best practices and pain-points inspire research
3. Theory to practice

Distributed Programming & Declarative DSLs
1. Go beyond Read/Write analysis
2. CALM can give you intuition on how to avoid coordination
3. Power of declarative DSLs: not just for queries/ML!

Productivity for Data Analysis
1. More than visualization: next gen interaction with data
2. DSL is a key internal representation
3. ML and HCI/Vis in service of DSL authoring