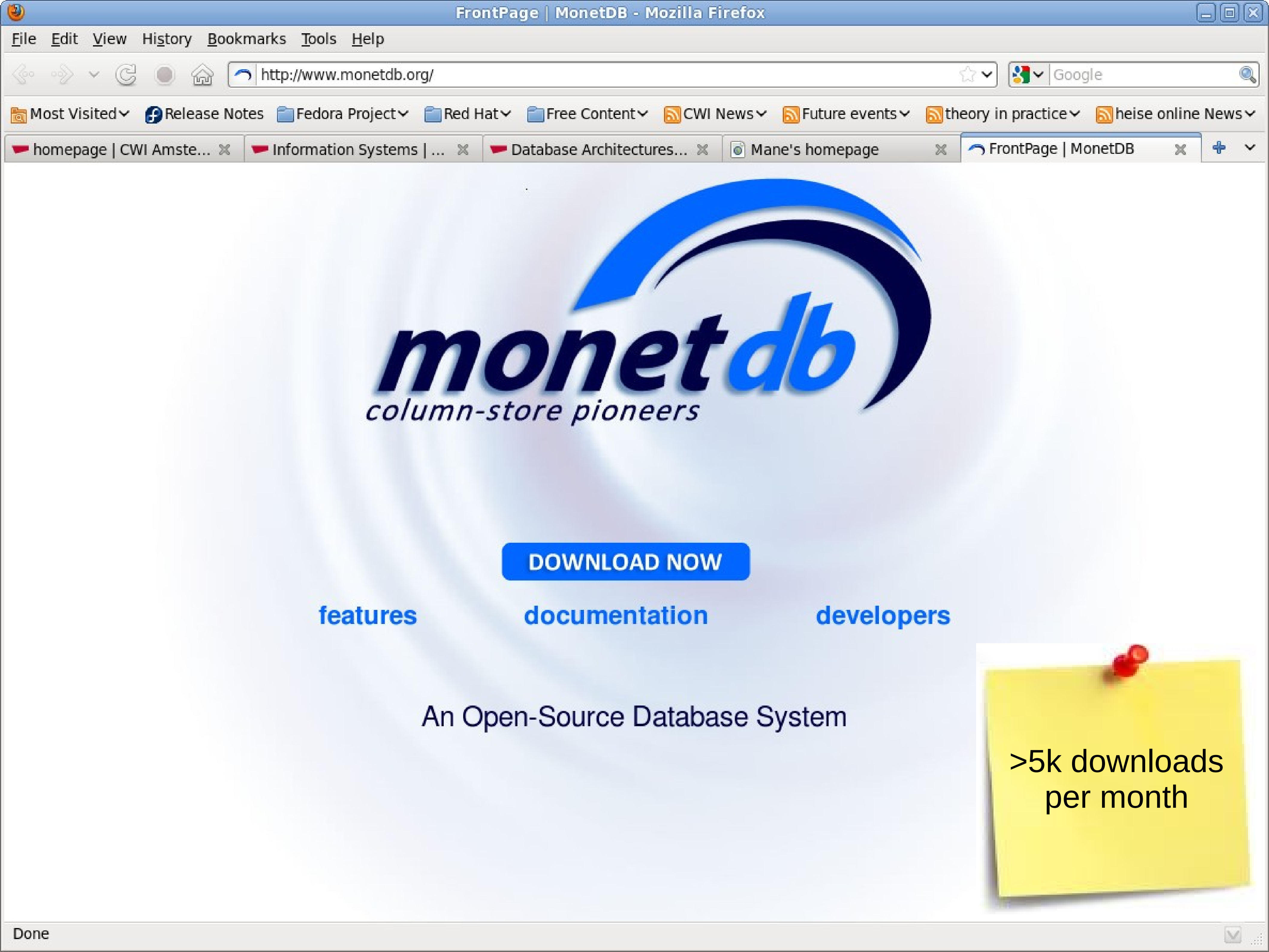
MonetDB:
Open-source Columnar Database Technology Beyond Textbooks

http://www.monetdb.org/

Stefan Manegold
Stefan.Manegold@cwi.nl
http://homepages.cwi.nl/~manegold/



>5k downloads per month

Why?

Why?

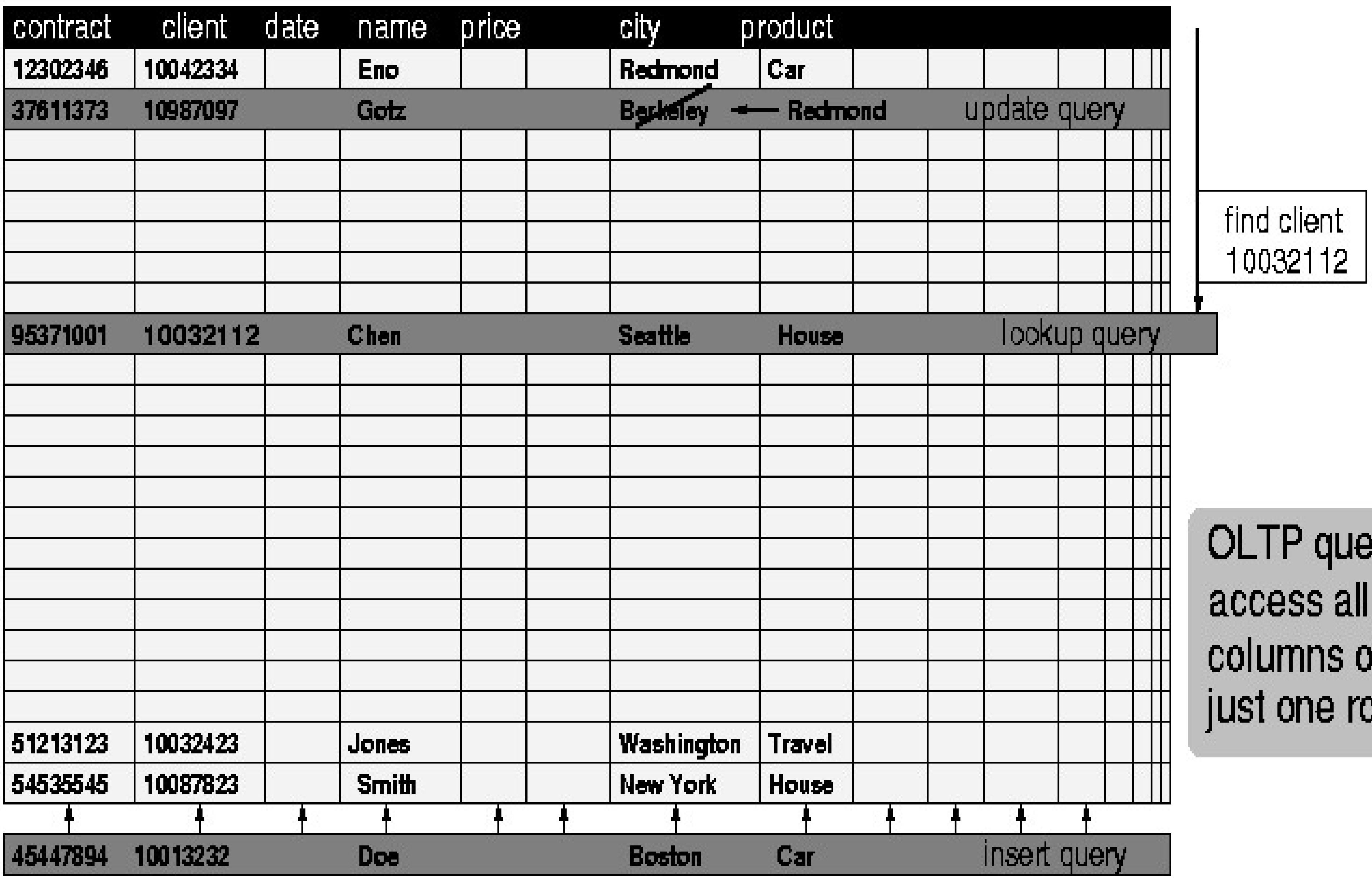
Motivation (early 1990s)

- Relational DBMSs dominate since the late 1970's / early 1980's
  - IBM DB2, MS SQL Server, Oracle, Ingres, ...
  - Transactional workloads (OLTP, row-wise access)
  - I/O based processing

- ***But:***
  - Workloads change (early 1990s)
  - Hardware changes (late 1990s)
  - Data “explodes” (early 2000s)

Why?

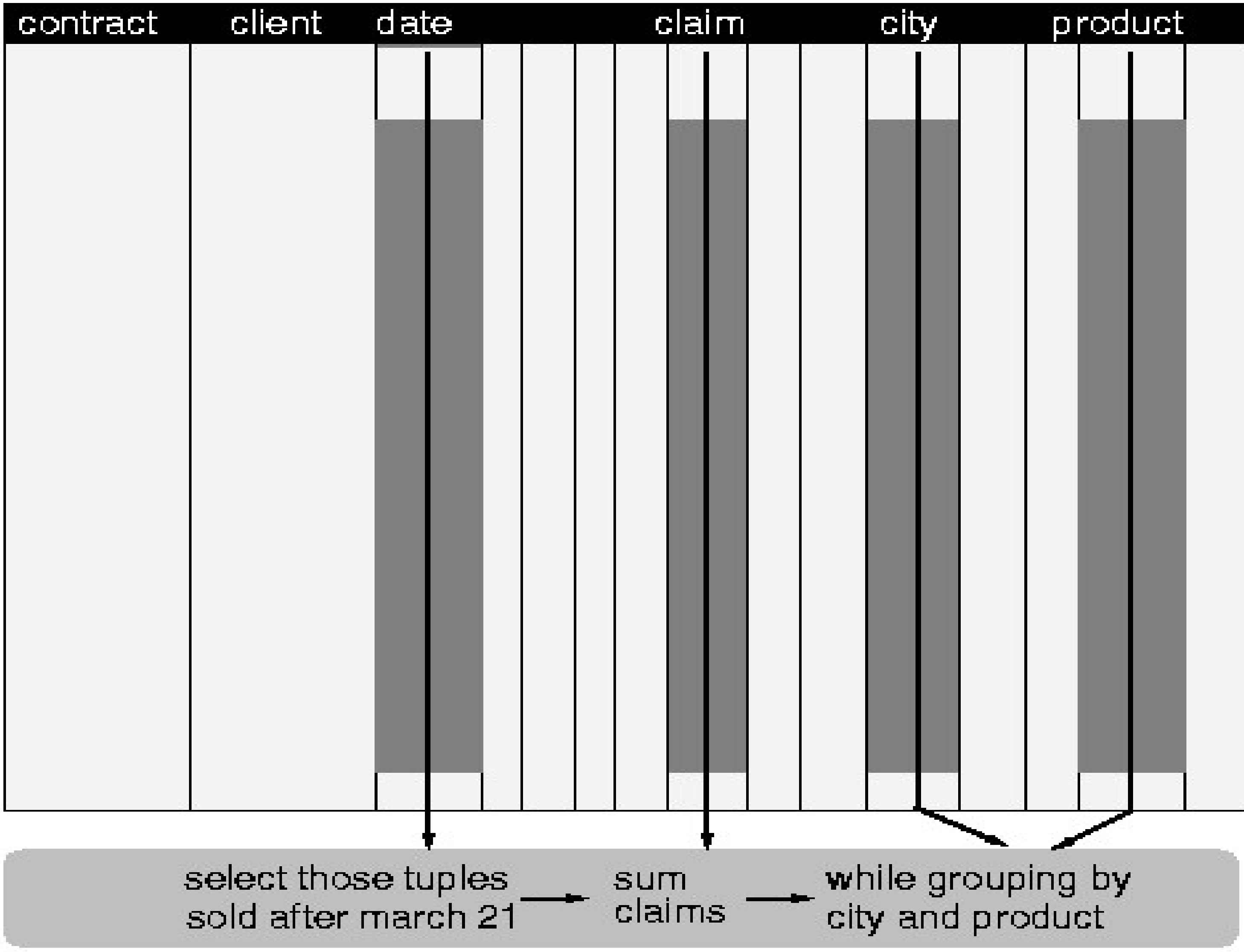
Workload changes: Transactions (OLTP) vs ...



OLTP queries: access all columns of just one row.

Why?

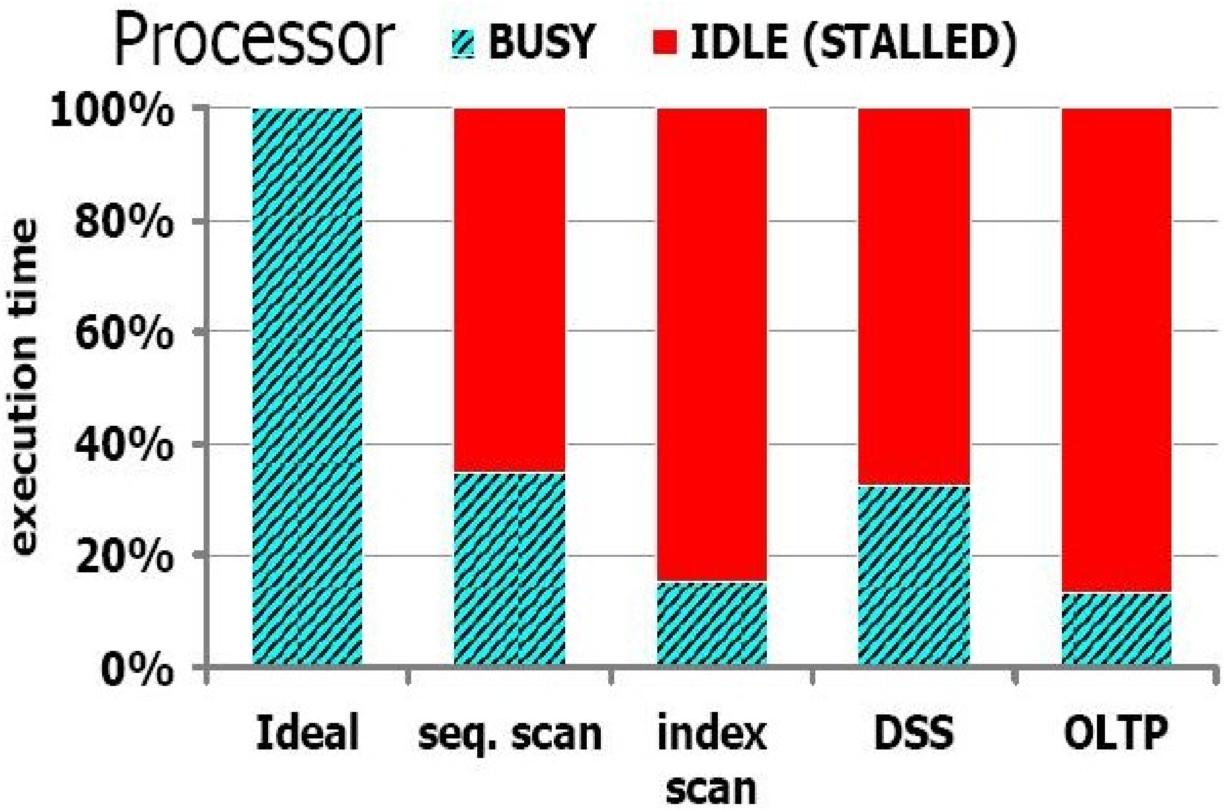
Workload changes: ... vs OLAP, BI, Data Mining, ...



OLAP query: accesses only a few columns of almost all rows.

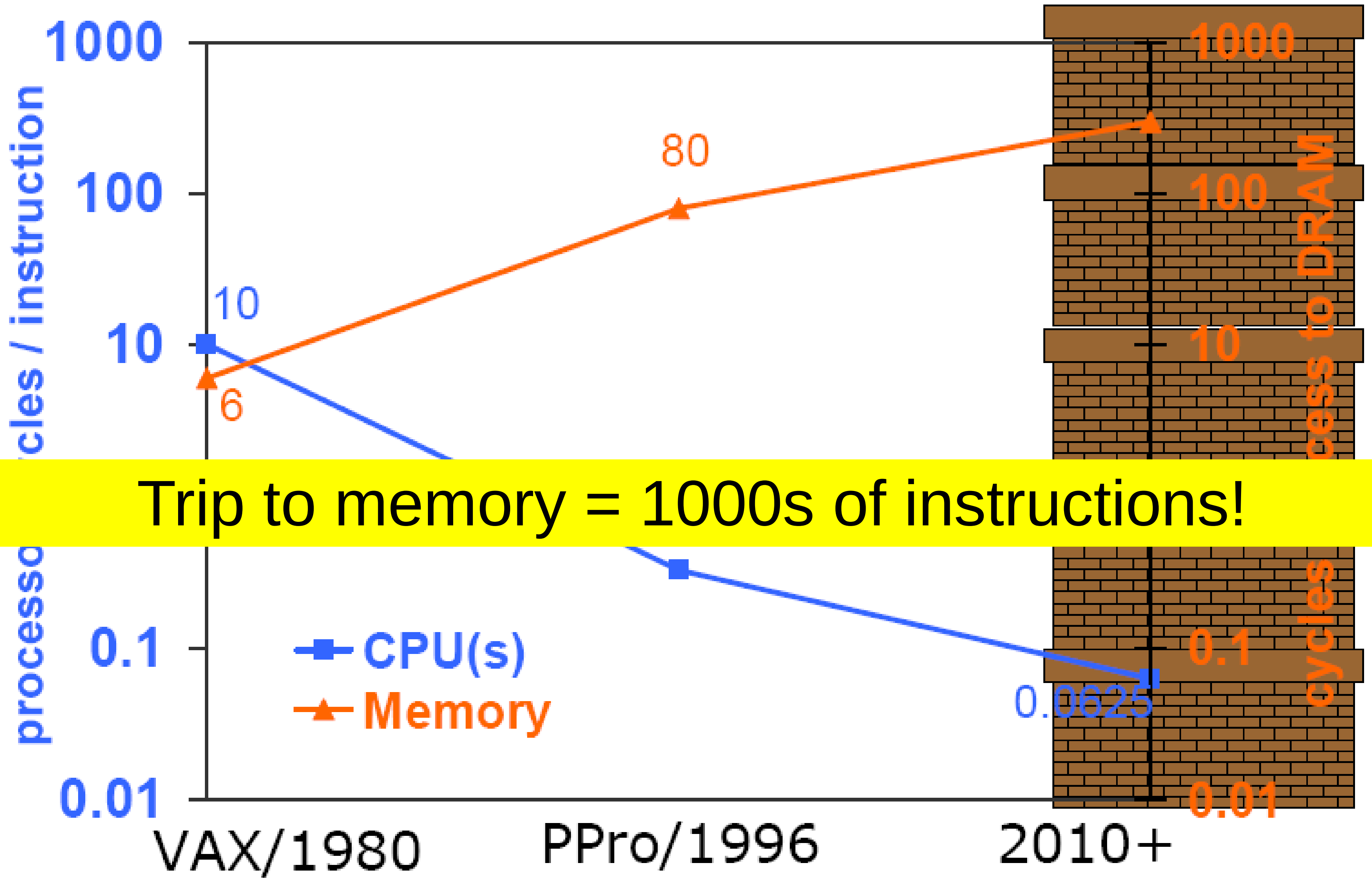
Databases hit The Memory Wall

- Detailed and exhaustive analysis for different workloads using 4 RDBMSs by Ailamaki, DeWitt, Hill, Wood in VLDB 1999: "*DBMSs On A Modern Processor: Where Does Time Go?*"
- CPU is 60%-90% idle, waiting for memory:
  - L1 data stalls
  - L1 instruction stalls
  - L2 data stalls
  - TLB stalls
  - Branch mispredictions
  - Resource stalls



Why?

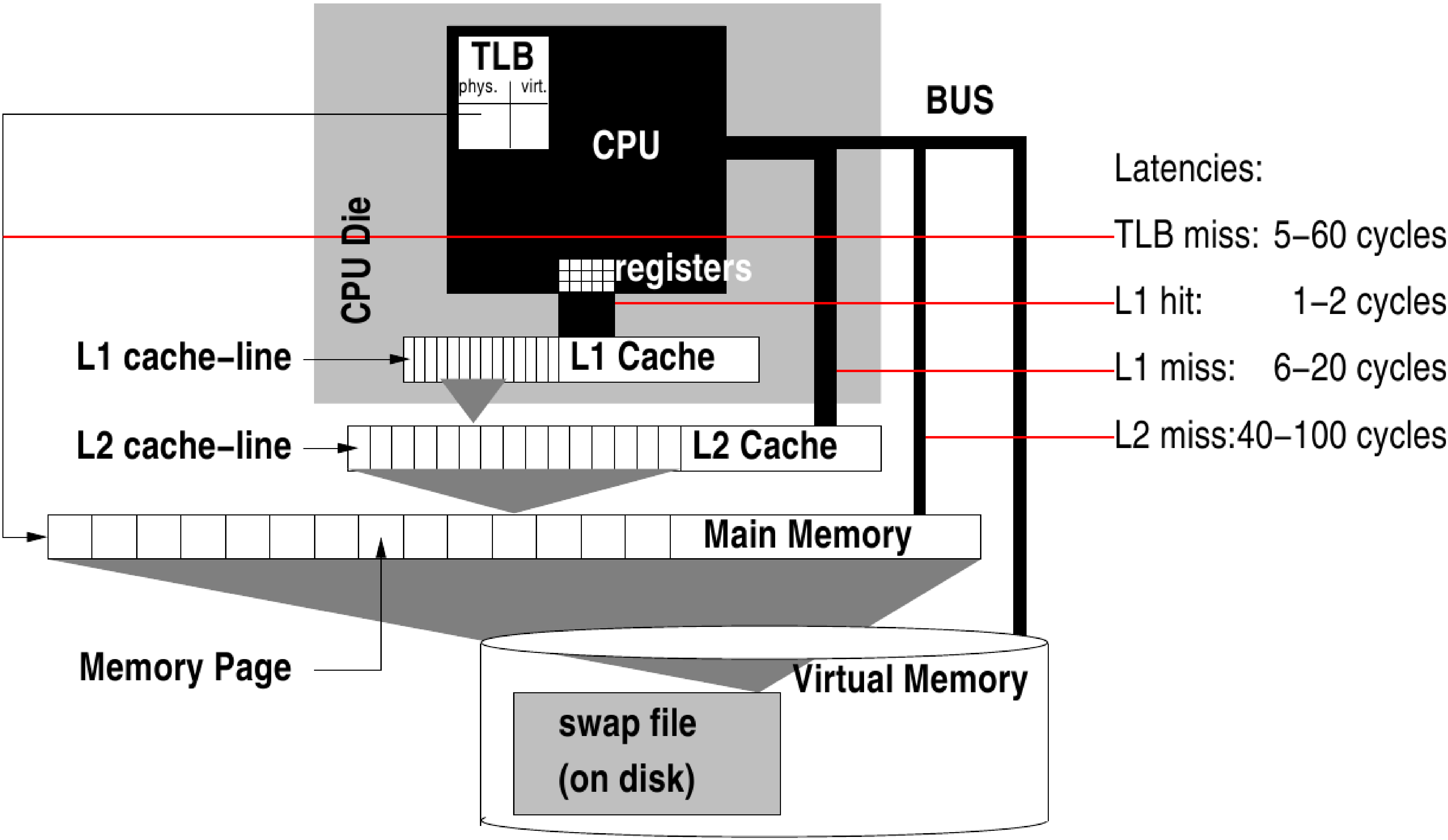
Hardware Changes: The Memory Wall



Trip to memory = 1000s of instructions!

Why?

Hardware Changes: Memory Hierarchies



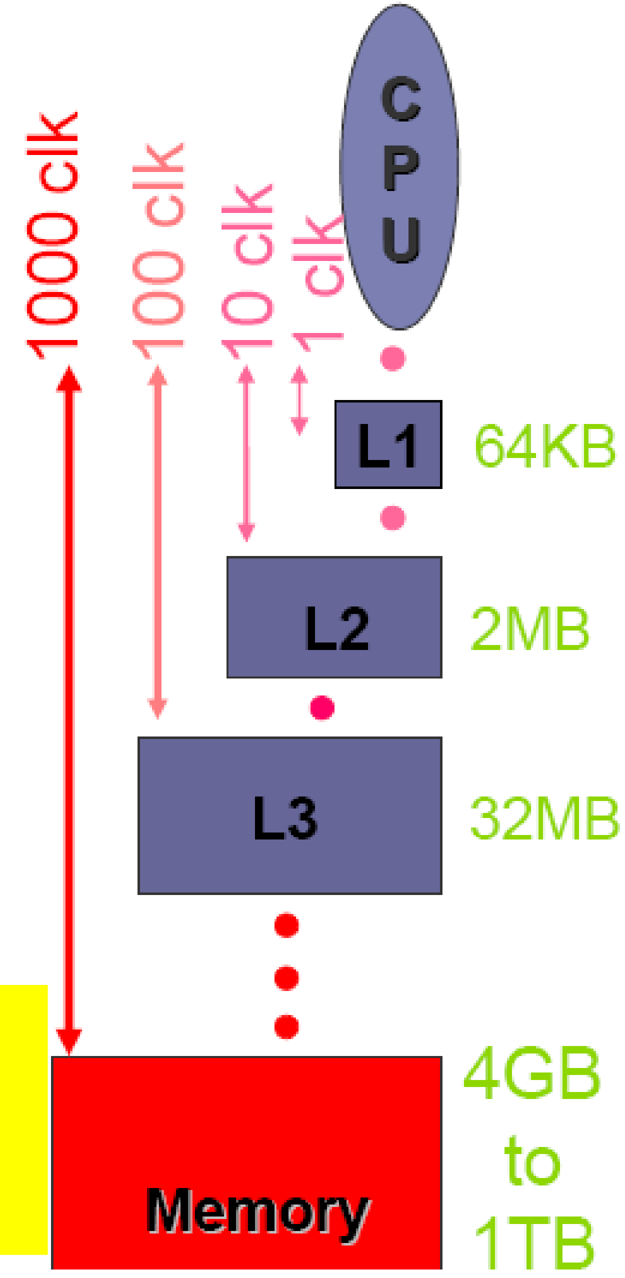
Hardware Changes: Memory Hierarchies

- Caches trade off capacity for speed
- Exploit instruction/data locality
- Demand fetch/wait for data

[ADH99]:

- Running top 4 database systems
- **At most 50% CPU utilization**

+Transition Lookaside Buffer (TLB)
Cache for VM address translation ➔
only 64 entries!



What?

What?

MonetDB

- Database kernel developed at CWI since 1993
  - *Research prototype turned into open-source product*
- Pioneering columnar database architecture
  - *Complete Relational/SQL (& XML/XQuery) DBMS*
- Focusing on in-memory processing
  - *Data is kept persistent on disk and can exceed memory limits*
- Aiming at OLAP, BI, data mining & scientific workloads (“read-dominated”)
  - *Supporting ACID transactions (WAL, optimistic CC)*
- Platform for database architecture research
  - *Used in academia (research & teaching) & commercial environments*
- Back-end for various DB research projects:

  Multi-Media DB & IR (“Tijah”), XML/XQuery (“Pathfinder”),

  Data Mining (“Proximity”), Digital Forensics (“XIRAF”), GIS (“OSM”), ...

Column-Store

formerly know as

Decomposed Storage Model

- 1985: ***DSM*** (Copeland et al.; SIGMOD 1985)
- 1992: First ideas and kernel for *MonetDB* (Kersten)
- 1993: *MonetDB* is born
- 1993: KDB (first commercial DSM system (?))
- 1995: Sybase IQ
- 2002: *MonetDB* goes open-source
- 2004[?]: Stonebraker et al. start "C-Store" project and coin DSM as "***Column-Store***"
- 2006[?]: Stonebraker et al. found "Vertica"; end of "C-Store" as research project
- 2008: Zukowski, Boncz, et al. (CWI) found VectorWise (based on *MonetDB*/X100)
- 2010: INGRES (now called Actian) acquires VectorWise
- 2011: HP acquires Vertica
- 201?: SAP HANA, IBM BLINK -> ISAO -> BLU, Oracle Database In-Memory Microsoft SQL Server Column-store indexes ("Apollo"), ...

How?

How?

SIGMOD 1985

A DECOMPOSITION STORAGE MODEL

George P Copeland
Setrag N Khoshafian
Microelectronics And Technolo...
...search...
Texa...

2 1 Support Of Multivalued Attributes

A more comprehensive data model than normalized relations might allow multivalued

2 2 Support Of Entities

A more comprehensive data model than the original relational model might support the notion

2 3 Support Of Multiple Parent Relations

A data model with more generality than relations might allow multiple parent relations, where a single record can have more than one parent

2 4 Support Of Heterogeneous Records

A data model with more generality than relations might allow heterogeneous records, where records of a single relation can have different

2 5 Support Of Directed Graphs

A data model with more generality than relations might allow a directed graph structure,

The DSM offers simplicity Simple systems have several major advantages over complex systems One advantage is that a set of fewer and simpler functions, given fixed development resources, can be either further tuned in software or pushed further into hardware to improve performance This is similar to the RISC (Patterson and Ditzel 1980) approach in general purpose architectures A second advantage is that many alternative cases with different processing strategies can less often be exploited, since the cases are not always recognized

studies have compared the performance of transposed storage models with the NSM (Hoffer 1976, Batory 1979, March and Severance 1977, March and Scudder 1984) In this report, we describe the advantages of a fully decomposed storage model (DSM), which is a transposed storage model with surrogates included The DSM pairs each attribute value with the surrogate of its conceptual schema record in a binary relation For example, the above relation would be stored as

a1	sur	val
	s1	v11
	s2	v12
	s3	v13

a2	sur	val
	s1	v21
	s2	v22
	s3	v23

a3	sur	val
	s1	v31
	s2	v32
	s3	v33

Storing Relations in MonetDB

DSM => Column-store

Front-End

contract	client	date	name	price	city	product
12302346	10042334		Eno		Redmond	Car
37611373	10987097		Gotz		Redmond	House
51213123	10032423		Jones		Washington	Travel
54535545	10087823		Smith		New York	House
45447894	10013232		Doe		Boston	Car
95371001	10032112		Chen		Seattle	House

logical data model

mapping rules

oid	contract
1000	12302346
1001	37611373
1002	51213123
1003	54535545
1004	45447894
1005	95371001

oid	client
1000	10042334
1001	10987097
1002	10032423
1003	10087823
1004	10013232
1005	10032112

oid	name
1000	Eno
1001	Gotz
1002	Jones
1003	Smith
1004	Doe
1005	Chen

oid	city
1000	Redmond
1001	Redmond
1002	Washington
1003	New York
1004	Boston
1005	Seattle

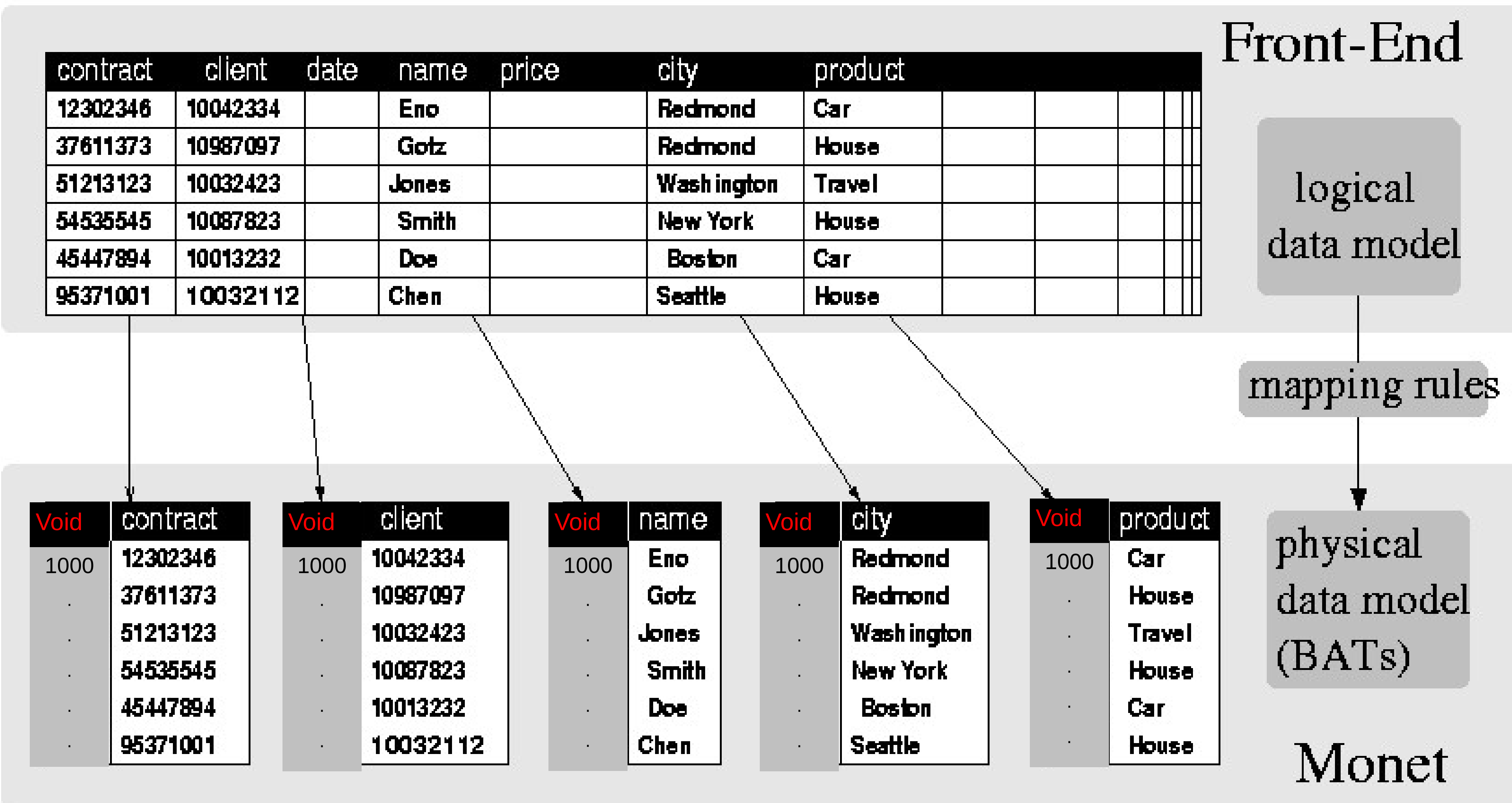
oid	product
1000	Car
1001	House
1002	Travel
1003	House
1004	Car
1005	House

physical data model (BATs)

Monet

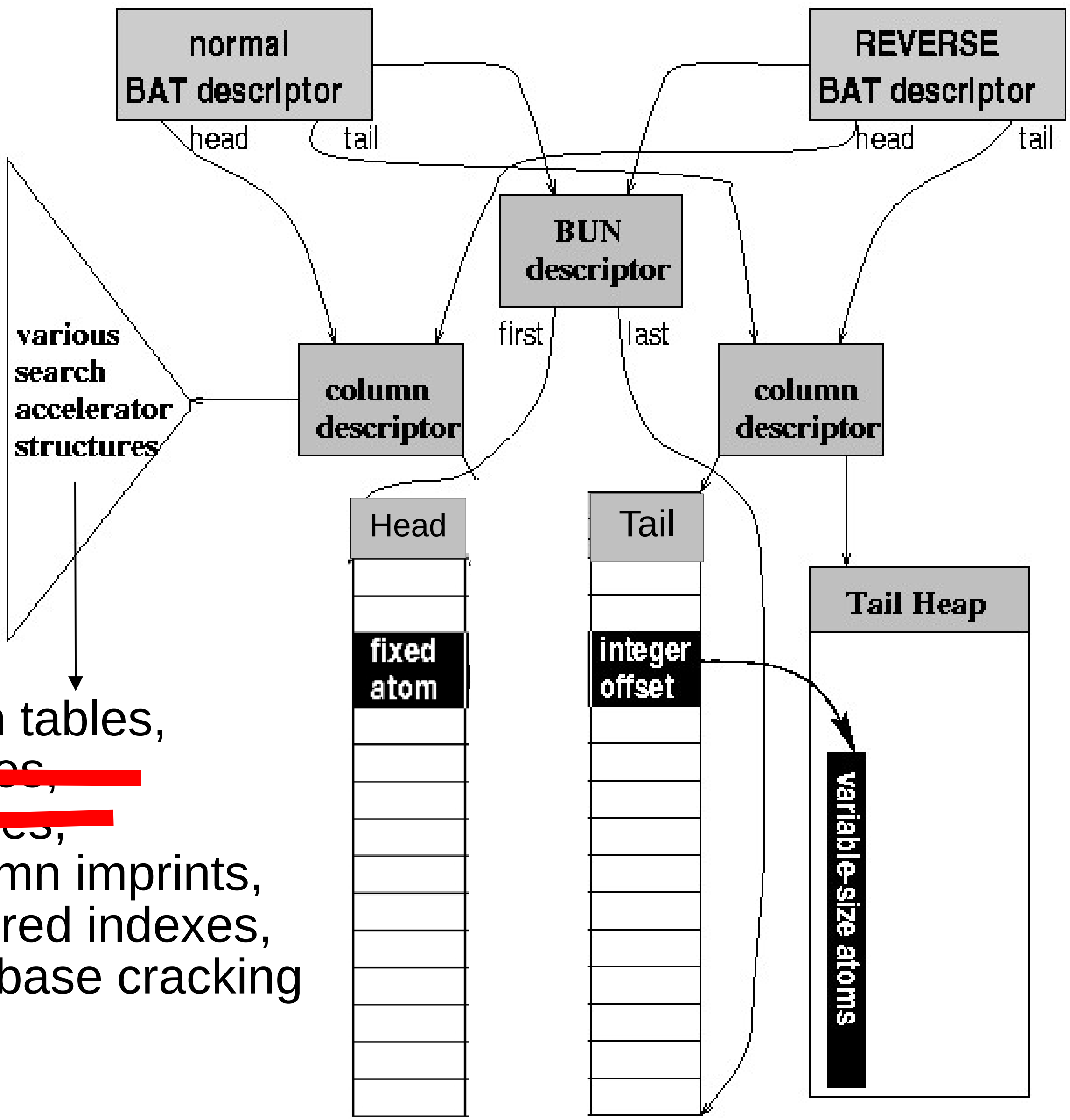
Storing Relations in MonetDB

DSM => Column-store



Virtual OID: seqbase=**1000** (increment=1)

BAT Data Structure (old)



Hash tables,
~~T-trees,~~
~~R-trees,~~
Column imprints,
Ordered indexes,
Database cracking

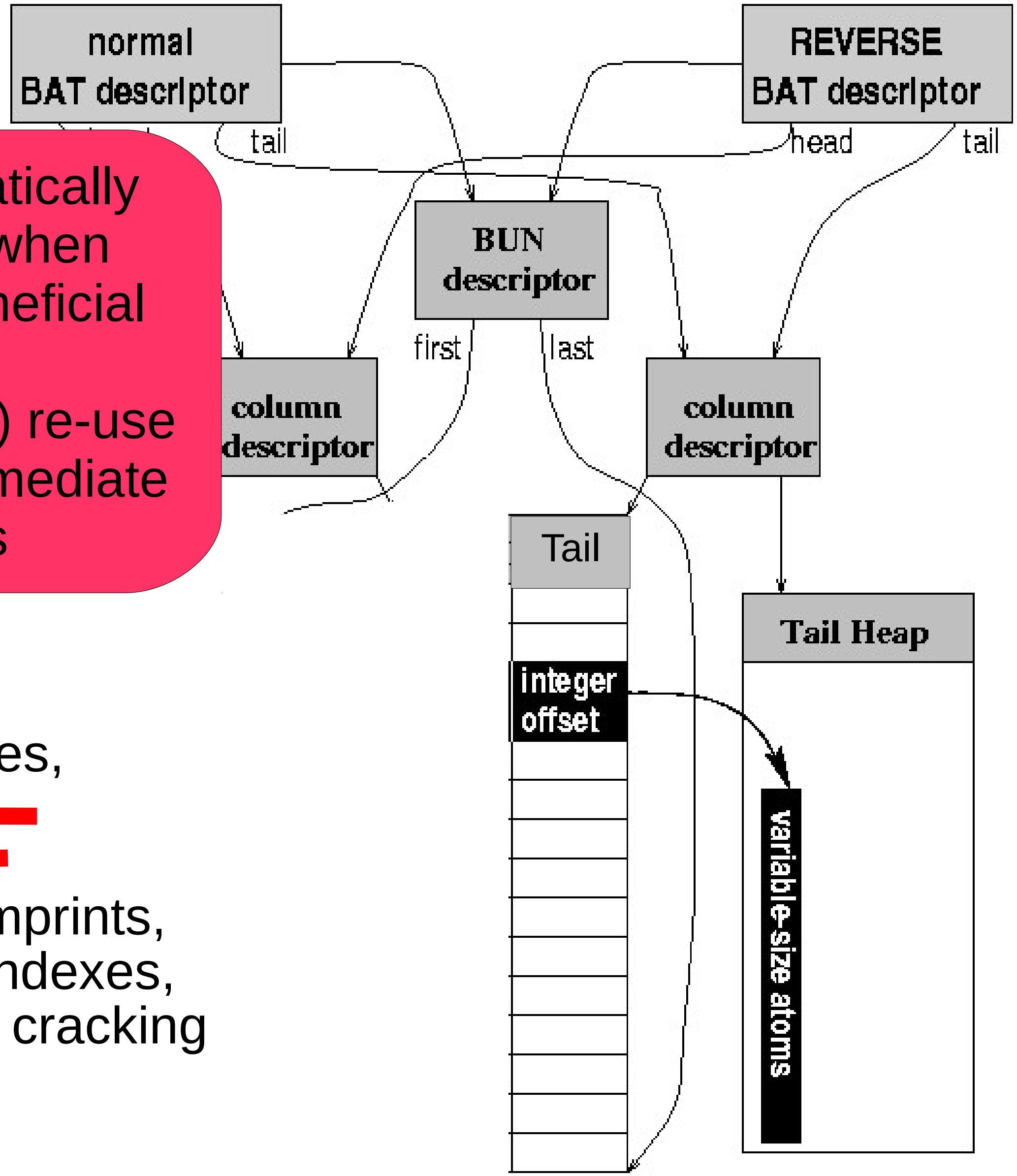
BAT:
binary association table

BUN:
binary unit

Head & Tail:
- consecutive memory blocks (arrays)
- memory-mapped files

Tail Heap:
- best-effort duplicate elimination for strings (~ dictionary encoding)
- minimal offset width; adapts automatically (1, 2, 4, 8 byte wide)

BAT Data Structure (new)



built automatically on-the-fly when required/beneficial

kept for (later) re-use also for intermediate results

Hash tables,
~~T-trees,~~
~~R-trees,~~
Column imprints,
Ordered indexes,
Database cracking

Tail:
- consecutive memory block (array)
- memory-mapped files

Tail Heap:
- best-effort duplicate elimination for strings (~ dictionary encoding)
- minimal offset width; adapts automatically (1, 2, 4, 8 byte wide)

RISC Relational Algebra

```
SELECT   id, name, (age-30)*50 as bonus
FROM     people
WHERE    age > 30
```

people_id (void)	people_id (int)
0	101
1	102
2	104
3	105
4	108
5	109
6	112
7	113
8	114
9	115

people_name (void)	people_name (str)
0	Alice
1	Ivan
2	Peggy
3	Victor
4	Eve
5	Walter
6	Trudy
7	Bob
8	Zoe
9	Charlie

people_age (void)	people_age (int)
0	22
1	37
2	45
3	25
4	19
5	31
6	27
7	29
8	42
9	35

RISC Relational Algebra

```
SELECT   id, name, (age-30)*50 as bonus
FROM     people
WHERE    age > 30
```

people_id

(void)	(int)
0	101
1	102
2	104
3	105
4	108
5	109
6	112
7	113
8	114
9	115

people_name

(void)	(str)
0	Alice
1	Ivan
2	Peggy
3	Victor
4	Eve
5	Walter
6	Trudy
7	Bob
8	Zoe
9	Charlie

people_age

(void)	(int)
0	22
1	37
2	45
3	25
4	19
5	31
6	27
7	29
8	42
9	35

select(30,nil)

sel_age

(oid)	(int)
1	37
2	45
5	31
8	42
9	35

Zero cost

markH()

(void)	(oid)
0	1
1	2
2	5
3	8
4	9

markT()

(void)	(int)
0	37
1	45
2	31
3	42
4	35

RISC Relational Algebra

```
SELECT   id, name, (age-30)*50 as bonus
FROM     people
WHERE    age > 30
```

people_id

(void)	(int)
0	101
1	102
2	104
3	105
4	108
5	109
6	112
7	113
8	114
9	115

people_name

(void)	(str)
0	Alice
1	Ivan
2	Peggy
3	Victor
4	Eve
5	Walter
6	Trudy
7	Bob
8	Zoe
9	Charlie

people_age

(void)	(int)
0	22
1	37
2	45
3	25
4	19
5	31
6	27
7	29
8	42
9	35

select(30,nil)

sel_age

(oid)	(int)
1	37
2	45
5	31
8	42
9	35

markH()

(void)	(oid)
0	1
1	2
2	5
3	8
4	9

markT()

(void)	(int)
0	37
1	45
2	31
3	42
4	35

VIEWS

(not materialized)

RISC Relational Algebra

```
SELECT   id, name, (age-30)*50 as bonus
FROM     people
WHERE    age > 30
```

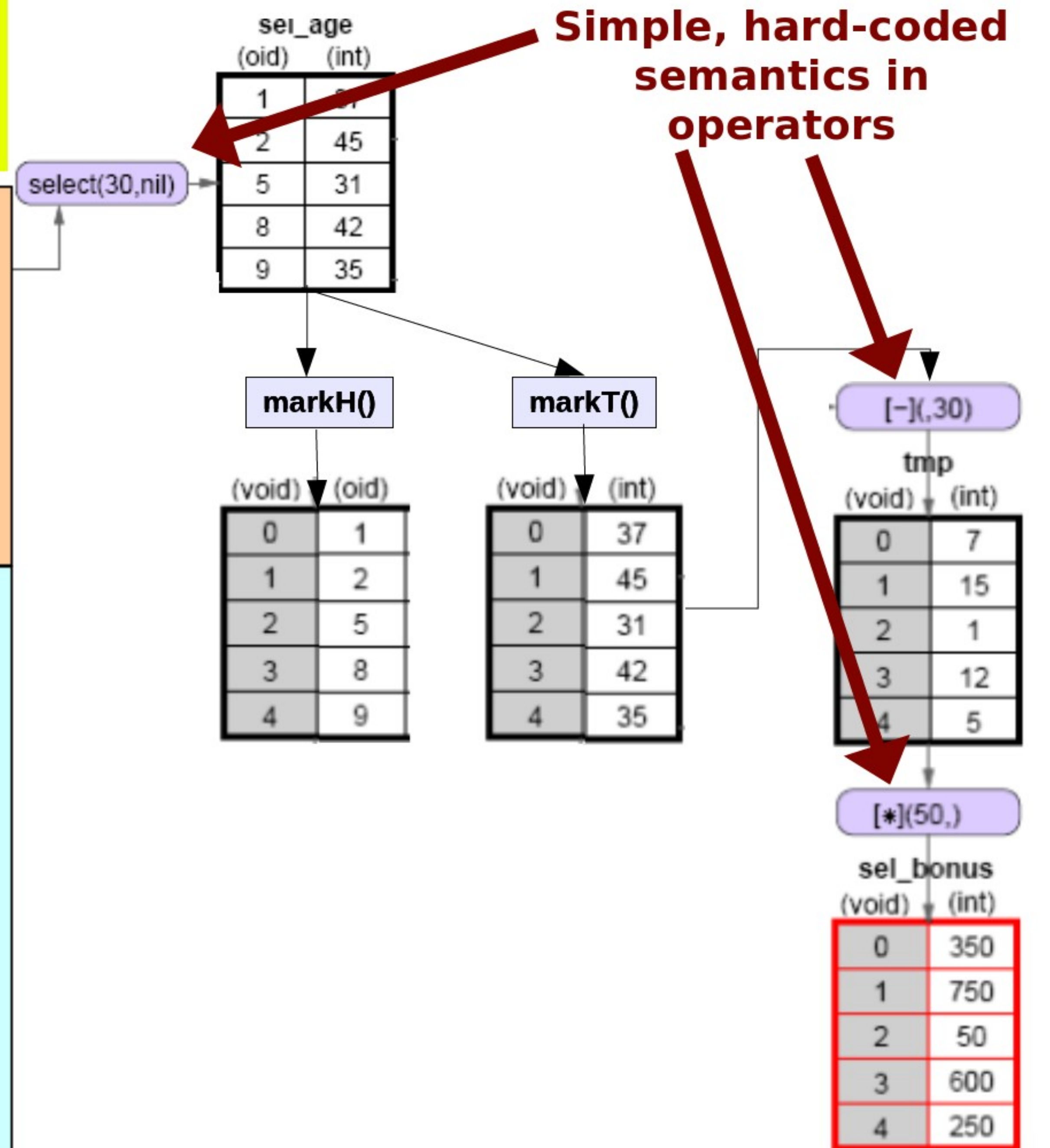
```
batcalc_minus_int(int* res,
                  int* col,
                  int  val,
                  int n)
{
    for(i=0; i<n; i++)
        res[i] = col[i] - val;
}
```

CPU ☺? Give it "nice" code !

- few dependencies (control,data)
- CPU gets out-of-order execution
- compiler can e.g. generate SIMD

One loop for an entire column

- no per-tuple interpretation
- arrays: no record navigation
- better instruction cache locality



RISC Relational Algebra

```
SELECT   id, name, (age-30)*50 as bonus
FROM     people
WHERE    age > 30
```

people_id

(void)	(int)
0	101
1	102
2	104
3	105
4	108
5	109
6	112
7	113
8	114
9	115

people_name

(void)	(str)
0	Alice
1	Ivan
2	Peggy
3	Victor
4	Eve
5	Walter
6	Trudy
7	Bob
8	Zoe
9	Charlie

people_age

(void)	(int)
0	22
1	37
2	45
3	25
4	19
5	31
6	27
7	29
8	42
9	35

select(30,nil)

sel_age

(oid)	(int)
1	37
2	45
5	31
8	42
9	35

MATERIALIZED intermediate results

markH()

(void)	(oid)
0	1
1	2
2	5
3	8
4	9

markT()

(void)	(int)
0	37
1	45
2	31
3	42
4	35

[−](,30)

tmp

(void)	(int)
0	7
1	15
2	1
3	12
4	5

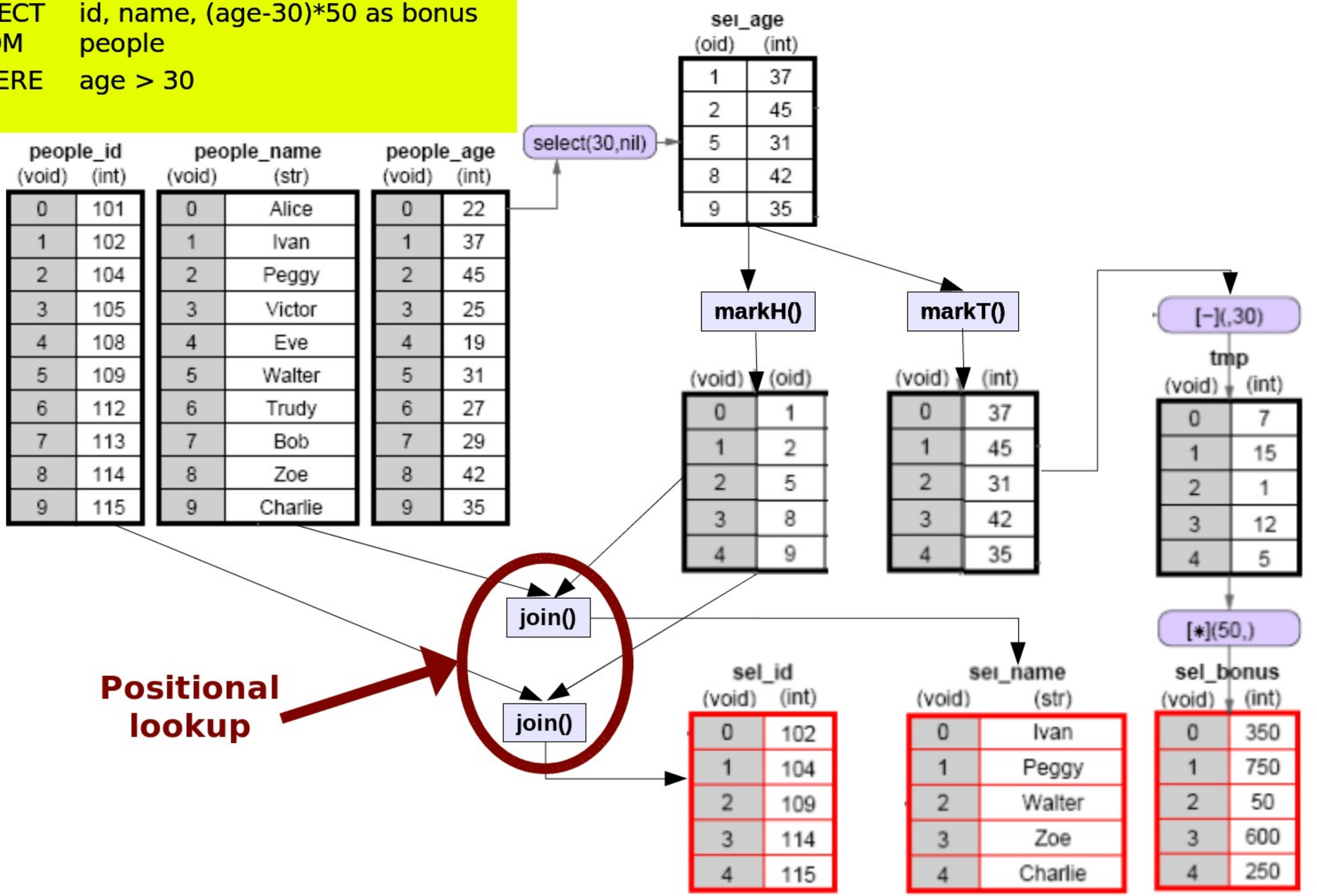
[∗](50,)

sel_bonus

(void)	(int)
0	350
1	750
2	50
3	600
4	250

RISC Relational Algebra

```
SELECT   id, name, (age-30)*50 as bonus
FROM     people
WHERE    age > 30
```



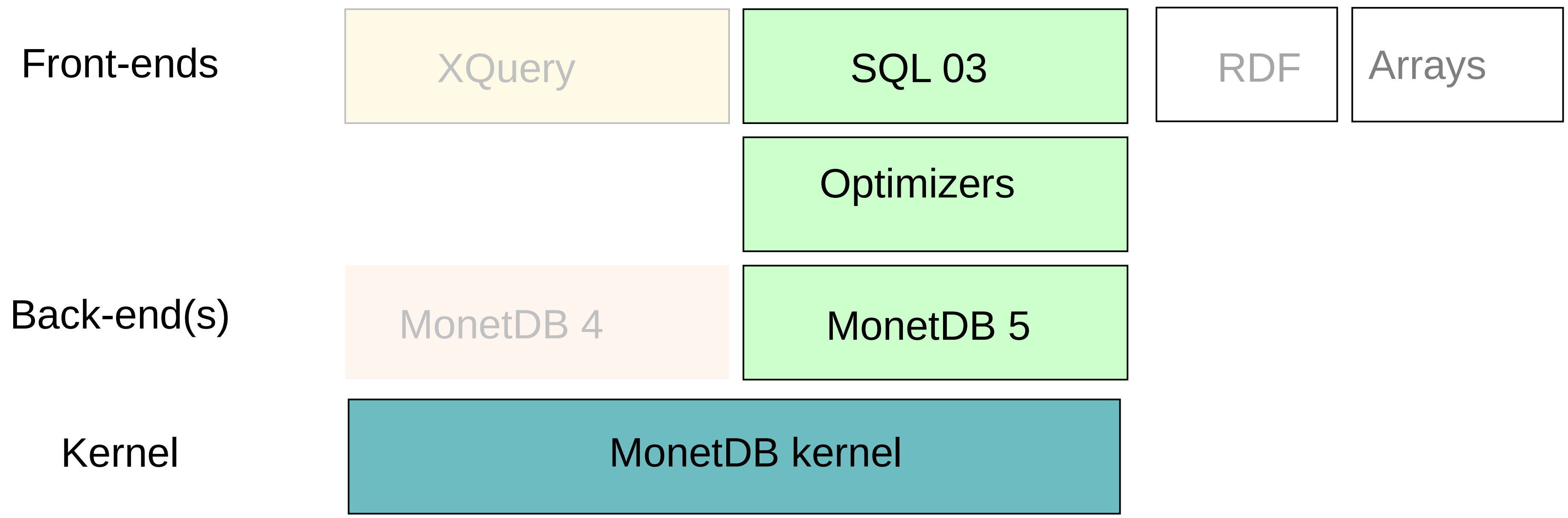
How is MonetDB Different

- full vertical fragmentation: always!
  - everything in binary (2-column) tables (**B**inary **A**ssociation **T**able)
  - saves you from table scan hell in OLAP and Data Mining
- RISC approach to databases
  - simple back-end data model (BATs)
  - simple back-end query language (binary/columnar relational algebra: *MAL*)
  - no need (to pay) for a buffer manager => manage virtual memory
  - admission control in scheduler to regulate memory consumption
  - explicit transaction management => DIY approach to ACID
- Multiple user data models & query languages
  - SQL, XML/XQuery, (RDF/SPARQL)
  - front-ends map data models to BATs and query languages to MAL

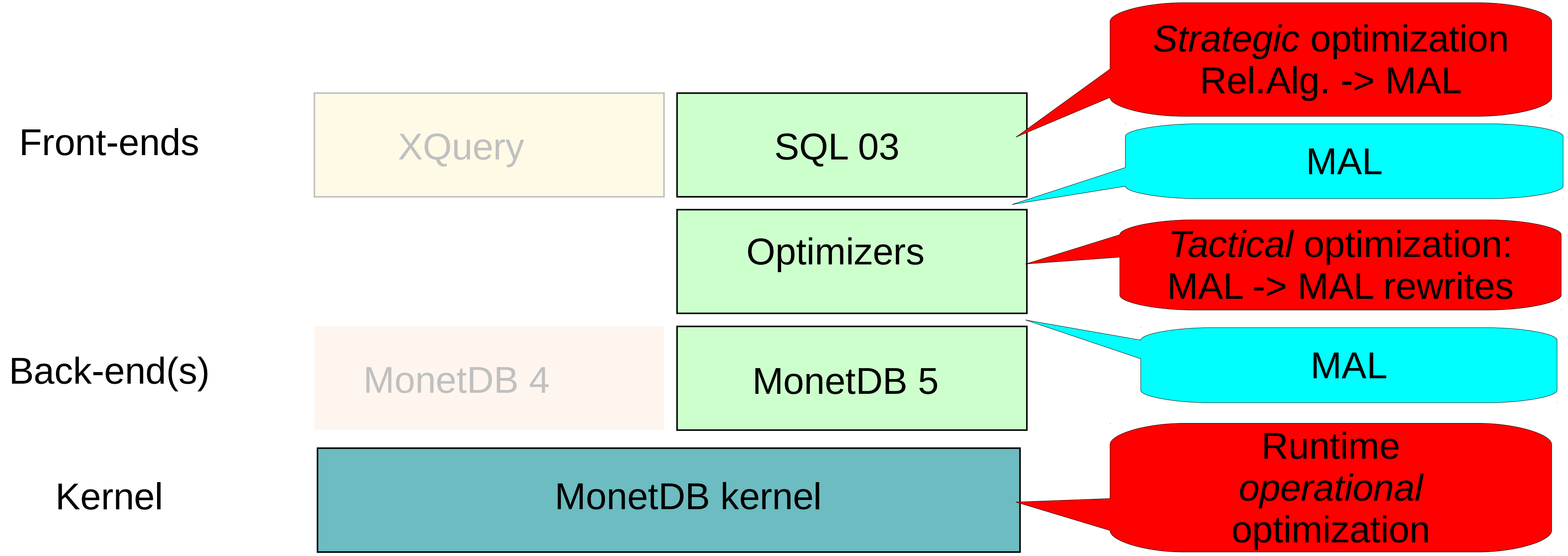
How is MonetDB Different

- operator-at-a-time bulk processing
  - avoids tuple-at-a-time management overhead
- CPU and memory cache optimized
  - Techniques adopted from scientific programming
  - Data structures:
    - Arrays
  - Code:
    - Compiler-friendly, branch-free, loop unrolling, instruction cache friendly
  - Algorithms:
    - Exploit spacial & temporal access locality

The MonetDB Software Stack



The MonetDB Software Stack



RISC Relational Algebra

```
SELECT   id, name, (age-30)*50 as bonus
FROM     people
WHERE    age > 30
```

people_id

(void)	(int)
0	101
1	102
2	104
3	105
4	108
5	109
6	112
7	113
8	114
9	115

people_name

(void)	(str)
0	Alice
1	Ivan
2	Peggy
3	Victor
4	Eve
5	Walter
6	Trudy
7	Bob
8	Zoe
9	Charlie

people_age

(void)	(int)
0	22
1	37
2	45
3	25
4	19
5	31
6	27
7	29
8	42
9	35

select(30,nil)

sel_age

(oid)	(int)
1	37
2	45
5	31
8	42
9	35

markH()

(void)	(oid)
0	1
1	2
2	5
3	8
4	9

markT()

(void)	(int)
0	37
1	45
2	31
3	42
4	35

[−](,30)

tmp

(void)	(int)
0	7
1	15
2	1
3	12
4	5

[∗](50,)

sel_id

(void)	(int)
0	102
1	104
2	109
3	114
4	115

sel_name

(void)	(str)
0	Ivan
1	Peggy
2	Walter
3	Zoe
4	Charlie

sel_bonus

(void)	(int)
0	350
1	750
2	50
3	600
4	250

```
_01 := sql.bind("sys","people","age",0);
_02 := algebra.select(_01,30,nil);
_03 := algebra.markH(_02,0);
_04 := algebra.markT(_02,0);
_05 := batcalc.-(_04,30);
_06 := batcalc.*(_05,50);
_07 := sql.bind("sys","people","name",0);
_08 := algebra.join(_03,_07);
_09 := sql.bind("sys","people","id",0);
_10 := algebra.join(_03,_09);
```

MonetDB Front-end: SQL

```
PLAN SELECT a FROM t WHERE c < 10;

project (
  select (
    table(sys.t) [ t.a, t.c, t.%TID% NOT NULL ]
  ) [ t.c < convert(10) ]
) [ t.a ]
```

MonetDB Front-end: SQL

```
EXPLAIN SELECT a FROM t WHERE c < 10;

function user.s1_1():void;
barrier _55 := language.dataflow();
    _02:bat[:void,:int] := sql.bind("sys","t","c",0);
    _07:bat[:oid, :int] := algebra.thetauselect(_02,10,"<");
    _10:bat[:oid,:void] := algebra.markT(_07,0@0);
    _11:bat[:void,:oid] := bat.reverse(_10);
    _12:bat[:oid, :int] := sql.bind("sys","t","a",0);
    _14:bat[:void,:int] := algebra.leftjoin(_11,_12);
exit _55;
    _15 := sql.resultSet(1,1,_14);
    sql.rsColumn(_15,"sys.t","a","int",32,0,_14);
    _21 := io.stdout();
    sql.exportResult(_21,_15);
end s1_1;
```

MonetDB Front-end: SQL

```
PLAN SELECT a, z FROM t, s WHERE t.c = s.x;

project (
  join (
    table(sys.t) [ t.a, t.c, t.%TID% NOT NULL ],
    table(sys.s) [ s.x, s.z, s.%TID% NOT NULL ]
  ) [ t.c = s.x ]
) [ t.a, s.z ]
```

MonetDB Front-end: SQL

```
EXPLAIN SELECT a, z FROM t, s WHERE t.c = s.x;

function user.s2_1():void;
 barrier _73 := language.dataflow();
  _02:bat[:void,:int] := sql.bind("sys","t","c",0);
  _07:bat[:void,:int] := sql.bind("sys","s","x",0);
  _10:bat[:int,:void] := bat.reverse(_07);
  _11:bat[:oid, :oid] := algebra.join(_02,_10);
  _13:bat[:oid,:void] := algebra.markT(_11,0@0);
  _14:bat[:void,:oid] := bat.reverse(_13);
  _15:bat[:void,:int] := sql.bind("sys","t","a",0);
  _17:bat[:void,:int] := algebra.leftjoin(_14,_15);
  _18:bat[:oid, :oid] := bat.reverse(_11);
  _19:bat[:oid,:void] := algebra.markT(_18,0@0);
  _20:bat[:void,:oid] := bat.reverse(_19);
  _21:bat[:void,:int] := sql.bind("sys","s","z",0);
  _23:bat[:void,:int] := algebra.leftjoin(_20,_21);
 exit _73;
  _24 := sql.resultSet(2,1,_17);
  sql.rsColumn(_24,"sys.t","a","int",32,0,_17);
  sql.rsColumn(_24,"sys.s","z","int",32,0,_23);
  _33 := io.stdout();
  sql.exportResult(_33,_24);
end s2_1;
```

Multi-core Parallelism: *Mitosis*

- Horizontally slice largest table
  - As many slices as CPU cores
  - As many slices such that #cores slices fit in memory
- Replicate query plan per slice
  - As far as possible
- Evaluate replicated plan fragments concurrently
- Concatenate partial intermediate result
- Evaluate remaining query plan

Multi-core Parallelism: *Mitosis*

```
EXPLAIN SELECT a FROM t WHERE c < 10;

function user.s1_1():void;
barrier _55 := language.dataflow();
    _02a:bat[:void,:int] := sql.bind("sys","t","c",0,0,2);
    _07a:bat[:oid, :int] := algebra.thetauselect(_02a,10,"<");
    _10a:bat[:oid,:void] := algebra.markT(_07a,0@0);
    _11a:bat[:void,:oid] := bat.reverse(_10a);
    _12a:bat[:oid, :int] := sql.bind("sys","t","a",0,0,2);
    _14a:bat[:void,:int] := algebra.leftjoin(_11a,_12a);
    _02b:bat[:void,:int] := sql.bind("sys","t","c",0,1,2);
    _07b:bat[:oid, :int] := algebra.thetauselect(_02b,10,"<");
    _10b:bat[:oid,:void] := algebra.markT(_07b,0@0);
    _11b:bat[:void,:oid] := bat.reverse(_10b);
    _12b:bat[:oid, :int] := sql.bind("sys","t","a",0,1,2);
    _14b:bat[:void,:int] := algebra.leftjoin(_11b,_12b);
exit _55;
    _14 := mat.pack(_14a,_14b);
    _15 := sql.resultSet(1,1,_14);
    sql.rsColumn(_15,"sys.t","a","int",32,0,_14);
    _21 := io.stdout();
    sql.exportResult(_21,_15);
end s1_1;
```

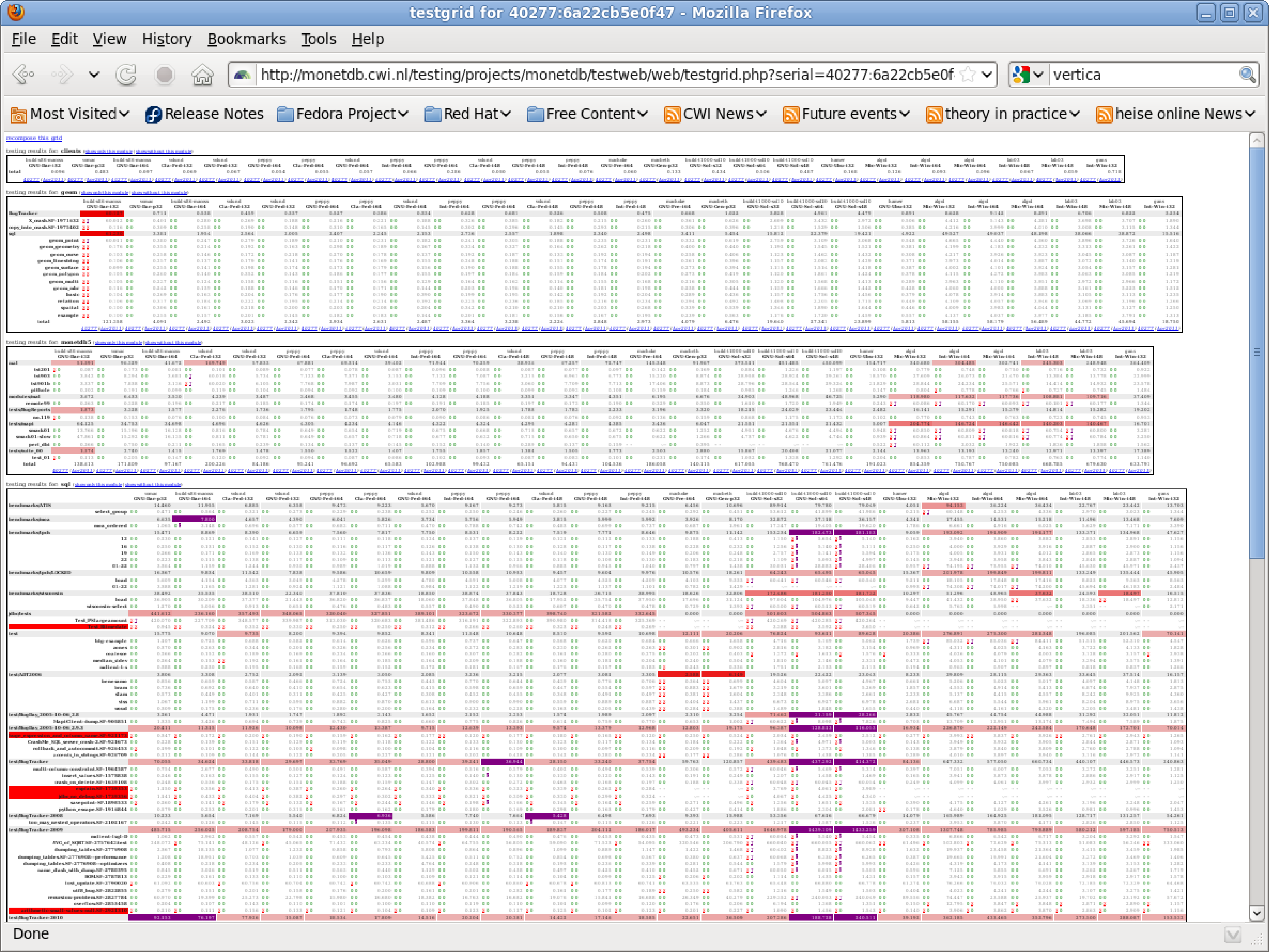
Multi-core Parallelism: *Mitosis*

```
EXPLAIN SELECT a FROM t WHERE c < 10;

function user.s1_1():void;
barrier _55 := language.dataflow();
    _02a:bat[:void,:int] := sql.bind("sys","t","c",0,0,2);
    _02b:bat[:void,:int] := sql.bind("sys","t","c",0,1,2);
    _07a:bat[:oid, :int] := algebra.thetauselect(_02a,10,"<");
    _07b:bat[:oid, :int] := algebra.thetauselect(_02b,10,"<");
    _10a:bat[:oid,:void] := algebra.markT(_07a,0@0);
    _10b:bat[:oid,:void] := algebra.markT(_07b,0@0);
    _11a:bat[:void,:oid] := bat.reverse(_10a);
    _11b:bat[:void,:oid] := bat.reverse(_10b);
    _12a:bat[:oid, :int] := sql.bind("sys","t","a",0,0,2);
    _12b:bat[:oid, :int] := sql.bind("sys","t","a",0,1,2);
    _14a:bat[:void,:int] := algebra.leftjoin(_11a,_12a);
    _14b:bat[:void,:int] := algebra.leftjoin(_11b,_12b);
exit _55;
    _14 := mat.pack(_14a,_14b);
    _15 := sql.resultSet(1,1,_14);
    sql.rsColumn(_15,"sys.t","a","int",32,0,_14);
    _21 := io.stdout();
    sql.exportResult(_21,_15);
end s1_1;
```

Open-Source Development

- Feature releases: 3-4 per year
  - Research results
  - User requests
- Bug-fix releases: monthly
- QA
  - Automated nightly testing on >20 platforms
  - Ensure correctness & stability
  - Ensure portability
  - Bug reports become test cases
  - Semi-automatic performance monitoring
  - Passed static code verification by Coverity with only minor problems


testgrid for 40277:6a22cb5e0f47 - Mozilla Firefox
File Edit View History Bookmarks Tools Help
http://monetdb.cwi.nl/testing/projects/monetdb/testweb/web/testgrid.php?serial=40277:6a22cb5e0f
vertica
Most Visited
Release Notes
Fedora Project
Red Hat
Free Content
CWI News
Future events
theory in practice
heise online News
Done

MonetDB vs Traditional DBMS Architecture

- **Architecture-Conscious Query Processing**
  - **vs Magnetic disk I/O conscious processing**
  - *Data layout, algorithms, cost models*
- **RISC Relational Algebra (operator-at-a-time)**
  - **vs Tuple-at-a-time Iterator Model**
  - *Faster through simplicity: no tuple expression interpreter*
- **Multi-Model: ODMG, SQL, XML/XQuery, ..., RDF/SPARQL**
  - **vs Relational with Bolt-on Subsystems**
  - *Columns as the building block for complex data structures*
- **Decoupling of Transactions from Execution/Buffering**
  - **vs ARIES integrated into Execution/Buffering/Indexing**
  - *ACID, but not ARIES.. Pay as you need transaction overhead.*
- **Run-Time Indexing and Query Optimization**
  - **vs Static DBA/Workload-driven Optimization & Indexing**
  - *Extensible Optimizer Framework;*
  - *cracking, recycling, sampling-based runtime optimization*