Analyzing Big Data *in* R

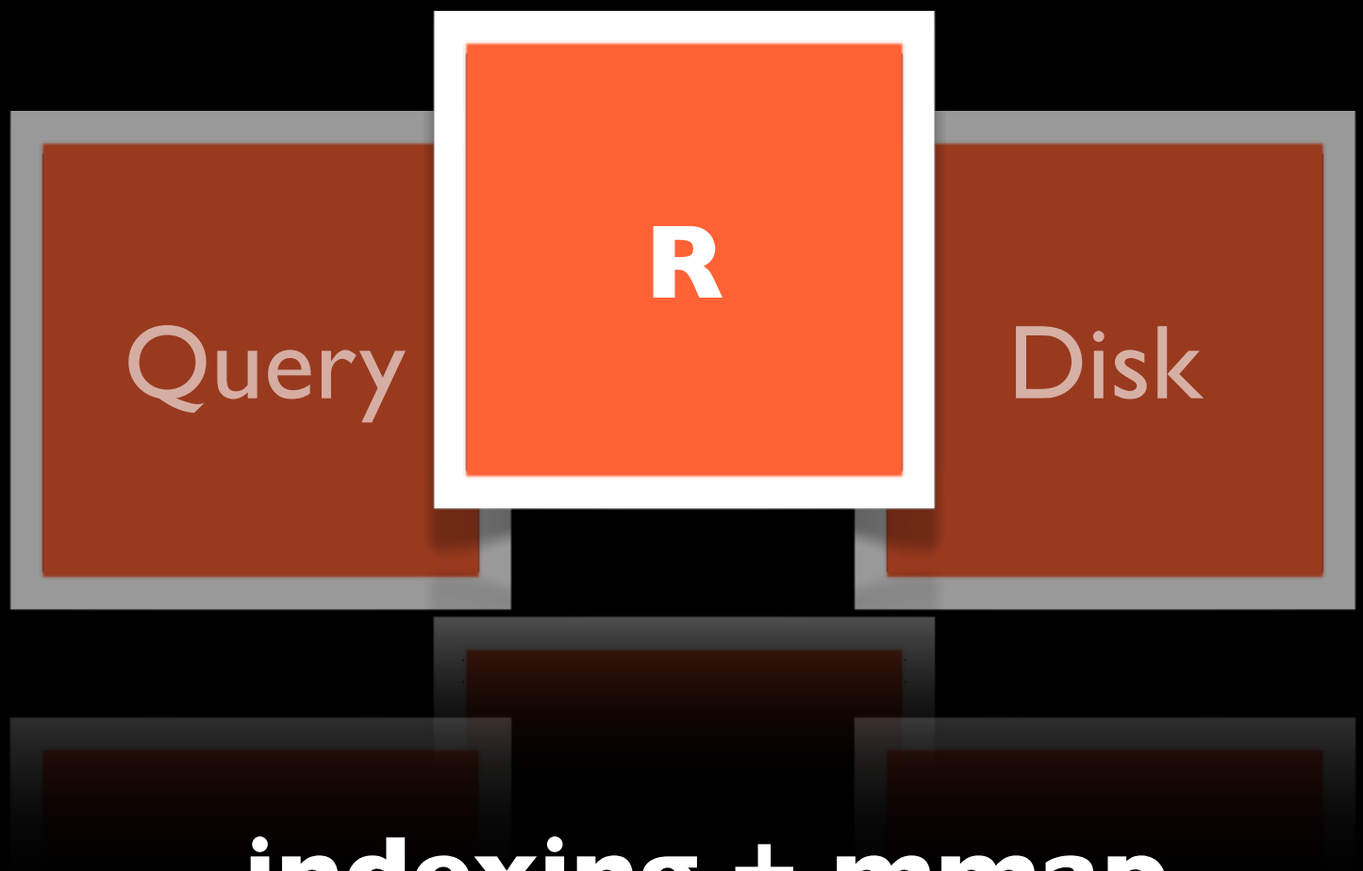
indexing + mmap

StrangeLoop 2012: Unsession R

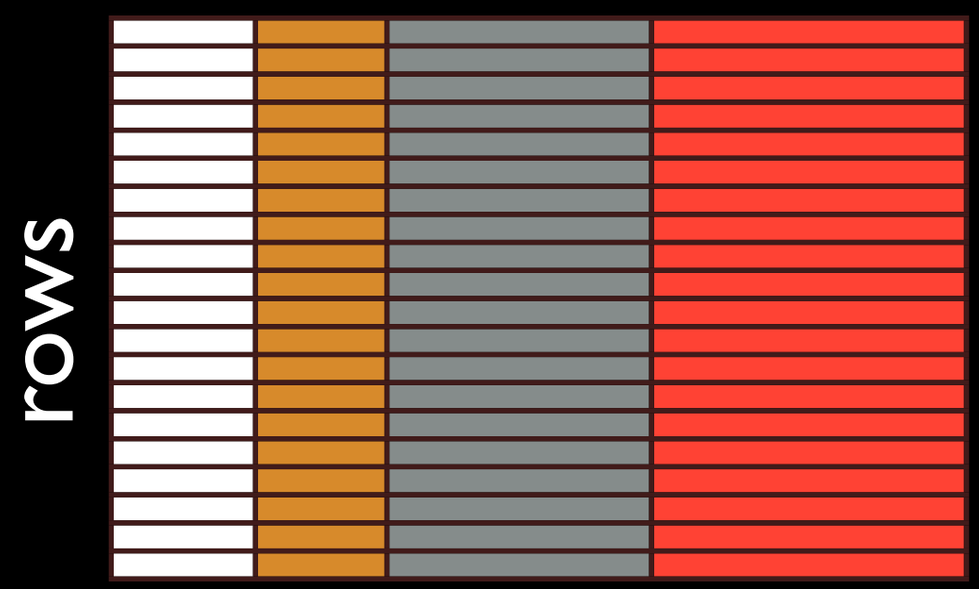
jeffrey.ryan @ lemnica.com

NoDB is Best

(aka *reinvent* the wheel)



Database Design101

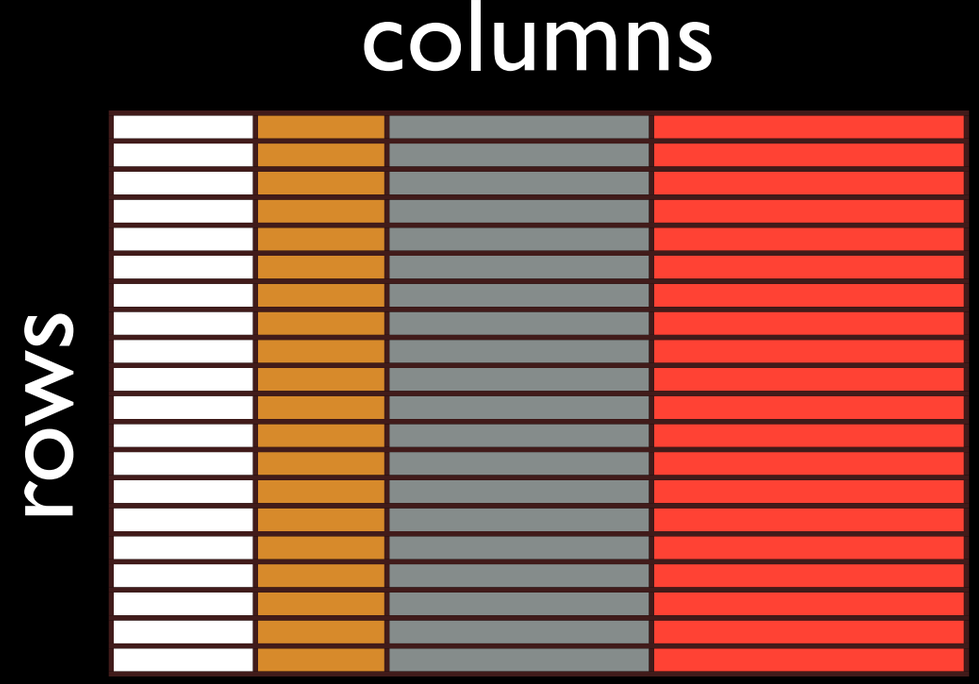


Row or Column Based

By Row

By Column

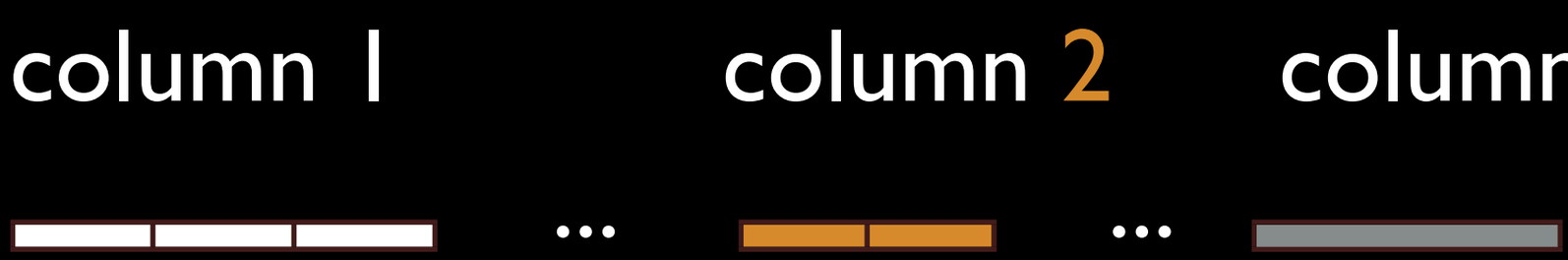
R is Column Oriented



i.e.

data.frames store column values sequentially

a.k.a column-major
a.k.a Fortran-order



The Good

Column-based is inherently read optimized

Columns of homogenous types compress well

Analytics are typically about reading, not writing

R is built for data analytics already!

The Bad

R is memory limited

Need memory many times data size

Searches are always linear scans

Uses extra memory and time

Could use a "real" database ...

... or we could make R the database!

mmap + indexing

The data.frame *supercharged!*

Unlimited Data memory mapped files

Fast Search

O(log n)

Pure R Semantics

db[a > 0.33]

mmap + indexing

The data.frame *supercharged!*

Unlimited Data memory mapped files

Fast Search

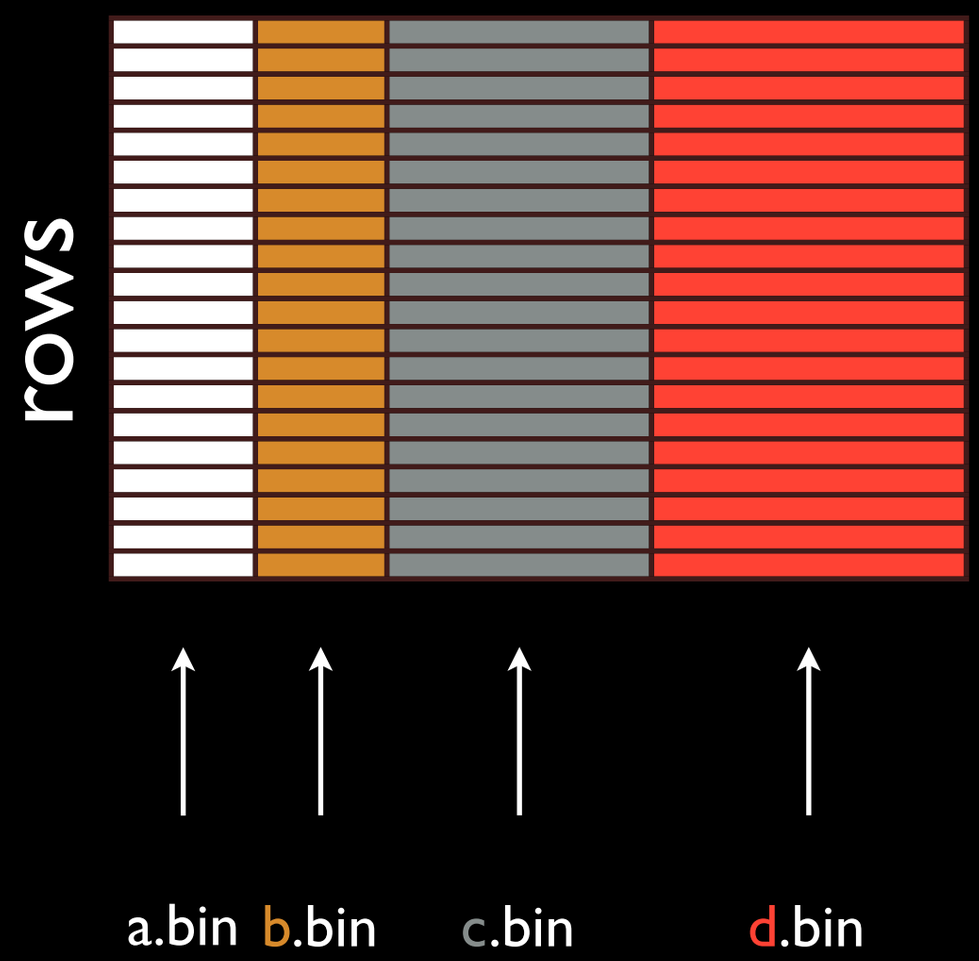
O(log n)

Pure R Semantics

db[a > 0.33]

Unlimited Data

memory mapped files



Keep column orientation

Use disk instead of memory

One file per column

Demand-based paging

mmap

OS system call

very low level API - you see what the C call sees

virtually map files into memory on demand

mmap similar (but different) to the R packages *ff* and *bigmemory*

mmap

mmap	R	C	bytes
raw()	raw	unsigned char	1
bits()	integer	int	1/32
char()	raw	char	1
uchar()	raw	unsigned char	1
int8()	integer	signed char	1
uint8()	integer	unsigned char	1
int16()	integer	signed short	2
uint16()	integer	unsigned short	2
int24()	integer	three byte int	3
uint24()	integer	unsigned three byte int	3
int32()	integer	int	4
integer()	integer	int	4
real32()	double	single precision float	4
real64()	double	double precision float	8
double()	double	double precision float	8
cplx()	complex	complex	16
complex()	complex	complex	16
char(n)	character	fixed-width ascii	n+1
char(n,nul=F)	character	non-nul terminated	n
character(n)	character	fixed-width ascii	n+1
struct(...)	list	struct of above types	variable

mmap

```
> # 2-byte (int16)
> # 4-byte (int32 or integer)
> # 8-byte float (real64 or double)

> record.type <- struct(short=int16(),
                        int=int32(),
                        double=real64())
> record.type
struct: (short) integer(0)
        (int) integer(0)
        (double) double(0)
> nbytes(record.type) # 14 bytes in total
[1] 14

> m <- mmap(tmp, record.type)
> m[1]
$short
[1] 1

$int
[1] 366214

$double
[1] -1.382365
```

mmap

```
> as.data.frame(head(sdohlcv))
  symbol       date  open  high   low close   volume
1      A 1288569600 34.94 35.12 34.38 34.75  2060528
2     AA 1288569600 13.19 13.29 12.95 13.05 18783447
3   AACC 1288569600  5.81  5.81  5.44  5.44    14512
4    AAI 1288569600  7.40  7.46  7.35  7.41 10313167
5    AAN 1288569600 18.90 18.96 18.58 18.65   494759
6    AAP 1288569600 65.29 65.34 64.46 64.76   554125
```

Row Oriented

```
> as.data.frame(lapply(quotes,head)))
  symbol       date  open  high   low close   volume
1      A 1288569600 34.94 35.12 34.38 34.75  2060528
2     AA 1288569600 13.19 13.29 12.95 13.05 18783447
3   AACC 1288569600  5.81  5.81  5.44  5.44    14512
4    AAI 1288569600  7.40  7.46  7.35  7.41 10313167
5    AAN 1288569600 18.90 18.96 18.58 18.65   494759
6    AAP 1288569600 65.29 65.34 64.46 64.76   554125
```

Column Oriented

mmap

```
> as.data.frame(head(sdohlcv))
  symbol       date  open  high   low close   volume
1      A 1288569600 34.94 35.12 34.38 34.75  2060528
2     AA 1288569600 13.19 13.29 12.95 13.05 18783447
3   AACC 1288569600  5.81  5.81  5.44  5.44    14512
4    AAI 1288569600  7.40  7.46  7.35  7.41 10313167
5    AAN 1288569600 18.90 18.96 18.58 18.65   494759
6    AAP 1288569600 65.29 65.34 64.46 64.76   554125
```

Row Oriented

DB — lemnica — vim — 80×35

```
library(mmap)
sdohlcv <- mmap("quotes/sdohlcv.bin",
                struct(symbol=char(6,F),
                       date=double(),
                       open=double(),
                       high=double(),
                       low=double(),
                       close=double(),
                       volume=integer()
                      )
               )

```

mmap

```
> as.data.frame(lapply(quotes,head)))
  symbol       date  open  high   low close   volume
1      A 1288569600 34.94 35.12 34.380 34.75  2060528
2     AA 1288569600 13.19 13.29 12.950 13.05 18783447
3   AACC 1288569600  5.81  5.81  5.440  5.44    14512
4    AAI 1288569600  7.40  7.46  7.350  7.41 10313167
5    AAN 1288569600 18.90 18.96 18.585 18.65   494759
6    AAP 1288569600 65.29 65.34 64.460 64.76   554125
```

Column Oriented

DB — lemnica — vim — 80×35

```
 1 library(mmap)
 2 quotes <- list()
 3 quotes$symbol <- mmap("quotes/symbol.bin",char(6,F))
 4 quotes$date   <- mmap("quotes/date.bin",double())
 5 quotes$open   <- mmap("quotes/open.bin",double())
 6 quotes$high   <- mmap("quotes/high.bin",double())
 7 quotes$low    <- mmap("quotes/low.bin",double())
 8 quotes$close  <- mmap("quotes/close.bin",real64())  # same as double()
 9 quotes$volume <- mmap("quotes/volume.bin",int32())  # same as integer()
10
11
12
13
```

mmap + indexing

The data.frame *supercharged!*

Unlimited Data

memory mapped files

Fast Search

O(log n)

Pure R Semantics

db[a > 0.33]

Fast Search

O(log n)

indexing

provide database style indexing and search tools
for R based data objects

column store + binary search + bitmap indexing + mmap

indexing

extend data.frame to use indexes (fast searching)

build in support for disk-based access (unlimited data)

R interface (painfully simple)

indexing

the interface

create_index load_index

[

vertical partitions

LZO compression

indexing

WAH bitmap compression

binary search

language agnostic storage

the technology

horizontal partitions

bitmap indexing

RLE encoding

column store

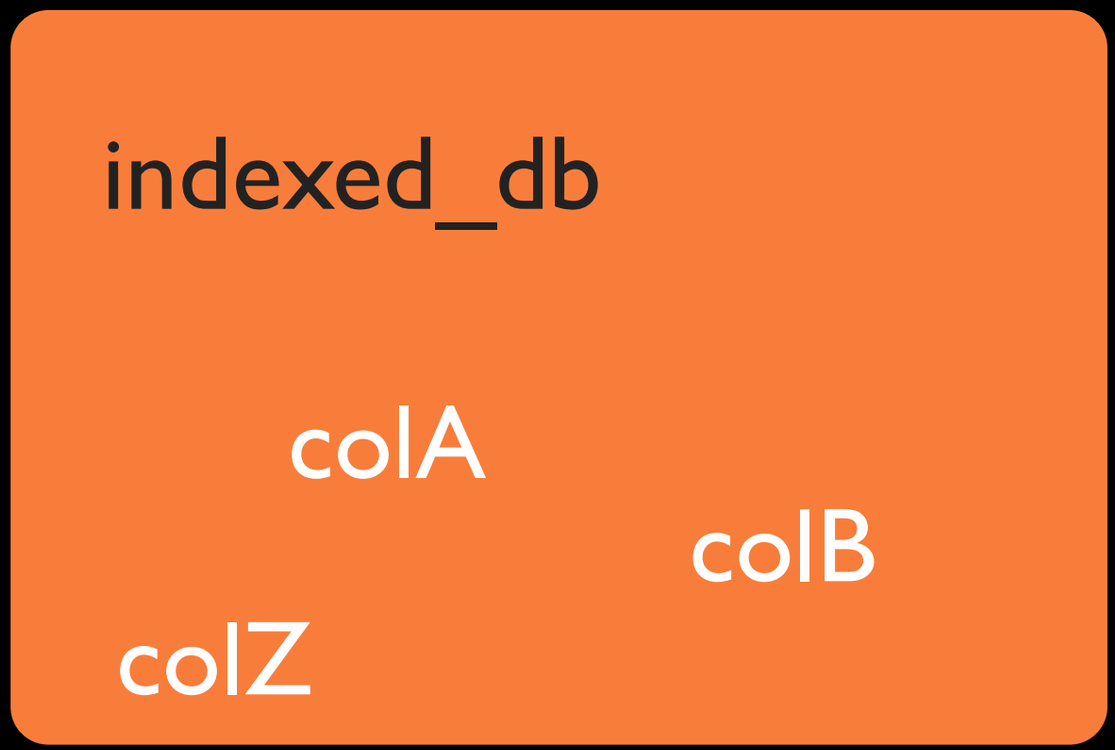
networked

query optimization

caching

mmap + indexing

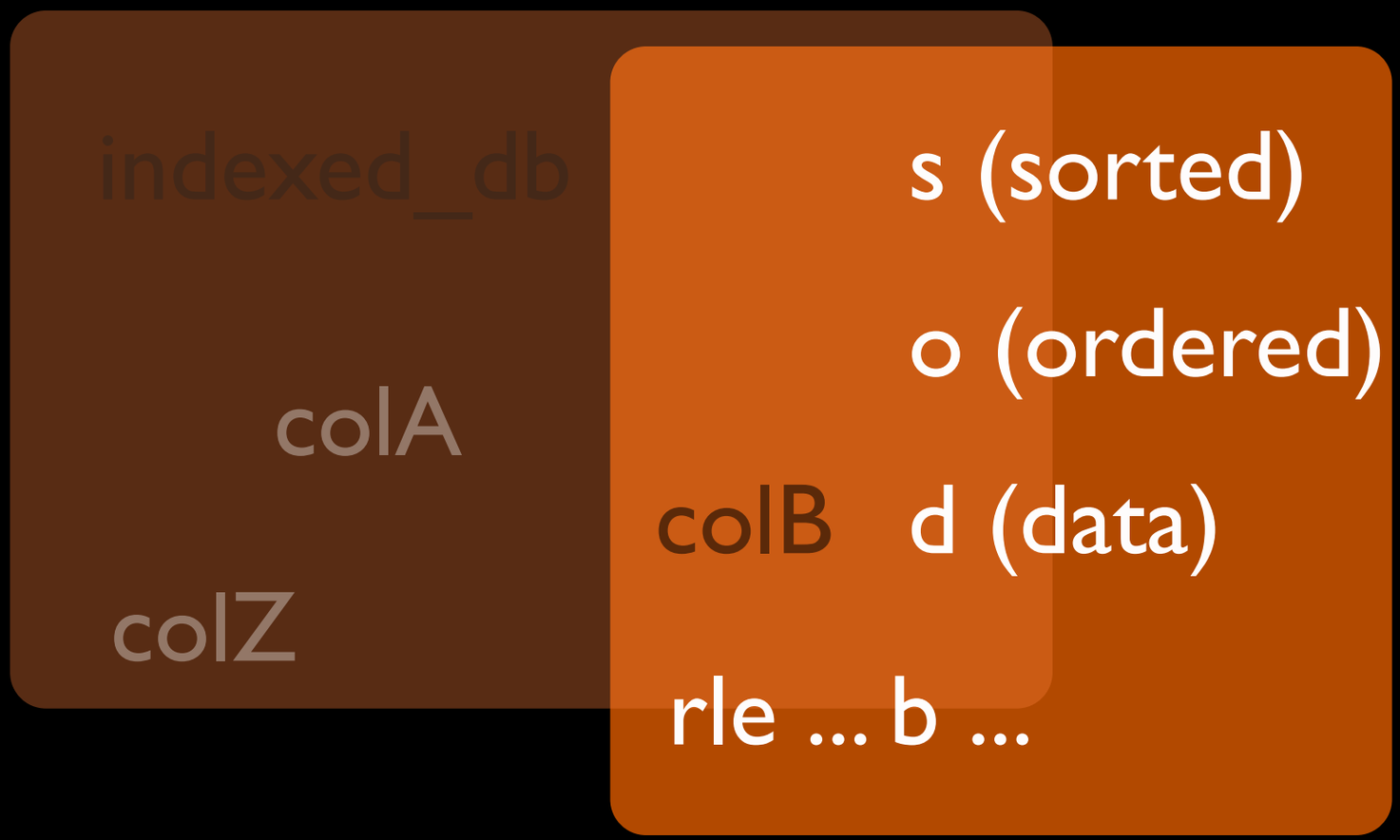
indexed_db is an environment



colA - Z are "columns" of your data

"columns" are really objects (lists) in the environment

mmap + indexing



lists contain the mmap objects to data on disk(s)

mmap + indexing

2 steps

create_index

any column or vector of data

returns the “indexed” environment

e.g.

```
Z <- rnorm(1e6)
db <- create_index(Z)
rm(Z)
```

[

use subsetting to magically extract data from disk using index (fast and friendly)

fancy *j* evaluation included

e.g.

```
db[Z < 0]
db[Z > 1 & Z < -3, Z]
db[Z < -3, mean(Z)]
```

mmap + indexing

Real World Example

67,836,671 equity option contracts
13 columns, 12GB on disk

```
> system.time( db[symbols=="AAPL"] )
   user  system elapsed
  0.012   0.000   0.012

> db[symbols=="AAPL"]
91428 hits
```

mmap + indexing

Real World Example

67,836,671 equity option contracts
13 columns, 12GB on disk

```
> system.time( db[symbols=="AAPL"] )
   user  system elapsed
  0.012   0.000   0.012

> db[symbols=="AAPL"]
91428 hits
```

** yes, not really “big”, but big for my MBA

mmap + indexing

Real World Example

7 queries. 7 graphs. 5 seconds

get a single contract as an xts time-series given OSI key

last 3 days of all AAPL April calls that have a delta at some point between .5 and .8, showing bid,ask,iv, and volume as an xts time-series

number of records on April 13

osi, bid and ask of AAPL puts (delta<0) on April 13, expiring on the April 17

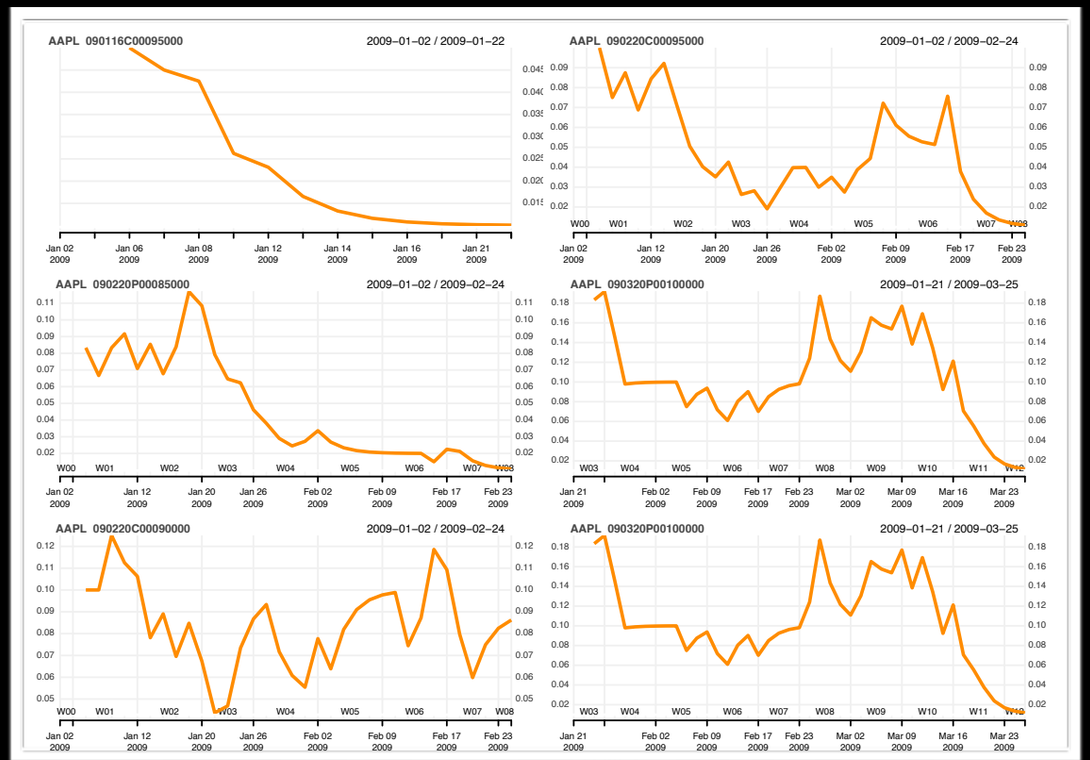
same, sorted by decreasing iv, excluding no-bid contracts, limit to 15

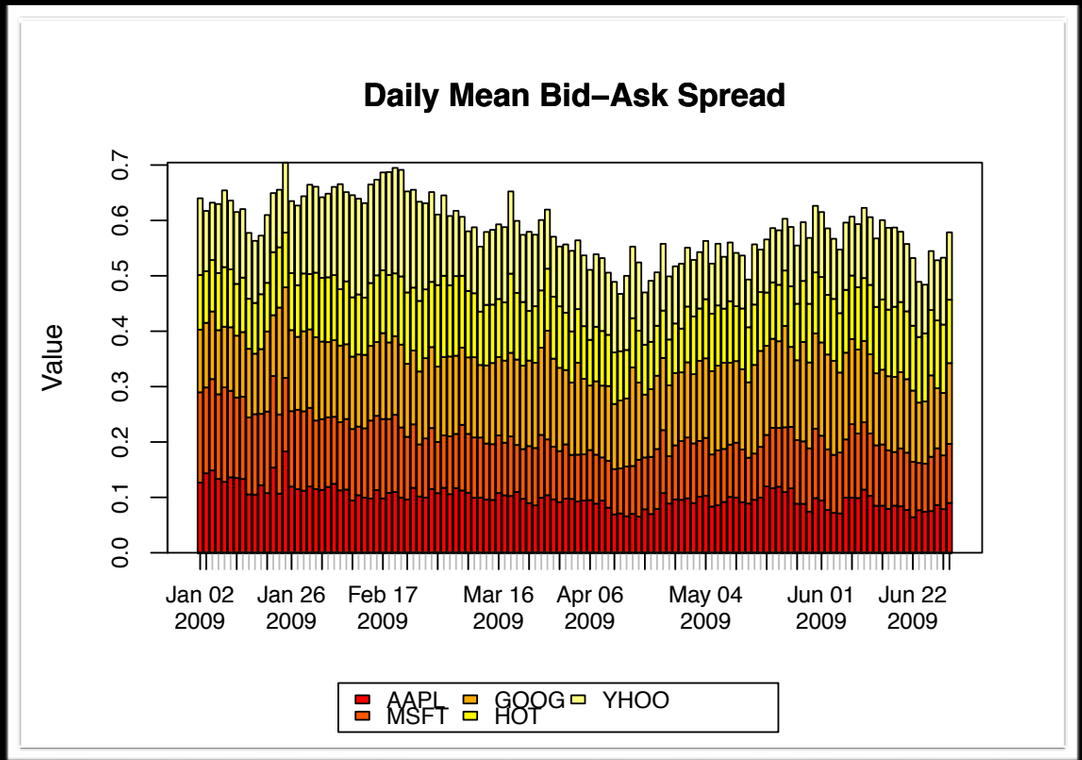
plot 3 day EMA of bid-ask spread of AAPL options with IV between 20% and 30%

plot 6 month mean daily bid ask spread for AAPL, MSFT, YHOO, GOOG, HOT

mmap + indexing

Real World Example





Conclusion And Caveats

Nothing is free

R centric workflow vs. DB

Understand your domain and requirements

www.lemnica.com