Market Scale Data

A tour of xts, xtime, mmap, indexing and more!

Presented May 11, 2012
R/Finance 2012, Chicago IL, USA

Jeffrey A. Ryan   jeffrey.ryan@lemnica.com
lemnica, corp.
www.lemnica.com
Chicago, IL, USA

About Me

Education in Economics and Finance (UIC)

Live in Chicago (West Loop)

President of lemnica - an R consultancy

Using R since 2003

Active contributor to open source R projects

Co-Organizer of R/Finance Conferences

What is Market Scale

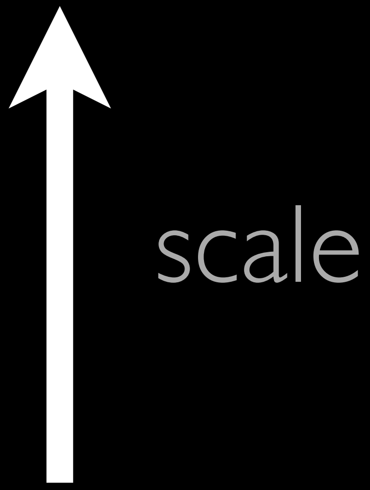
Order Book Messages

Time and Sales

Intraday Bars

EOD Equity Options

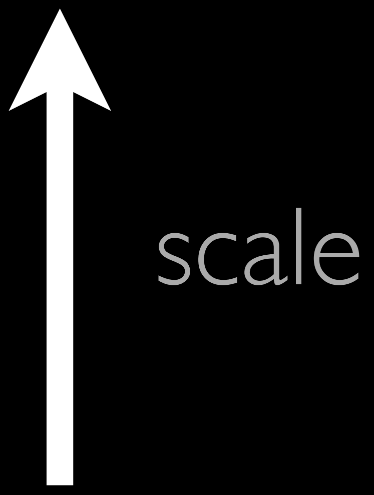
EOD US Equities



scale

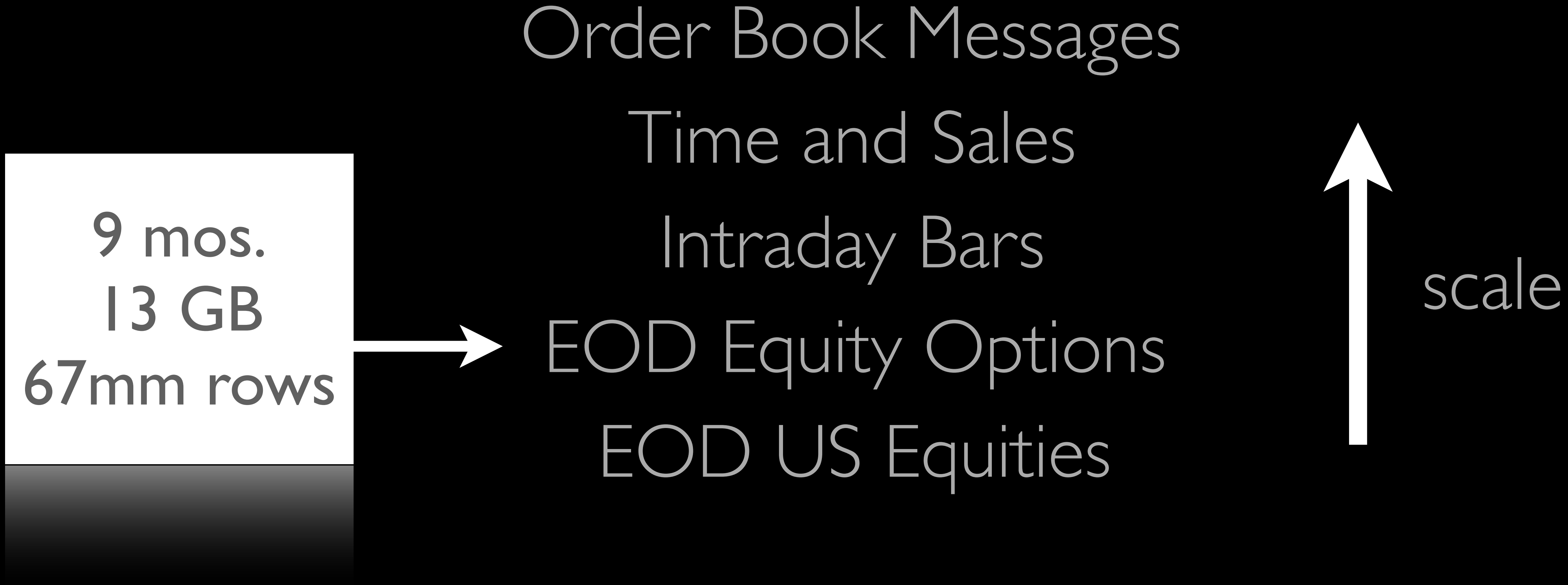
What is Market Scale

3 MM / sec	Order Book Messages
	Time and Sales
	Intraday Bars
	EOD Equity Options
7000/day	EOD US Equities



scale

What is Market Scale



9 mos.
13 GB
67mm rows

Order Book Messages

Time and Sales

Intraday Bars

EOD Equity Options

EOD US Equities

scale

Reinventing the wheel

The Current

R

Query

Disk

The NoDB

Query

R

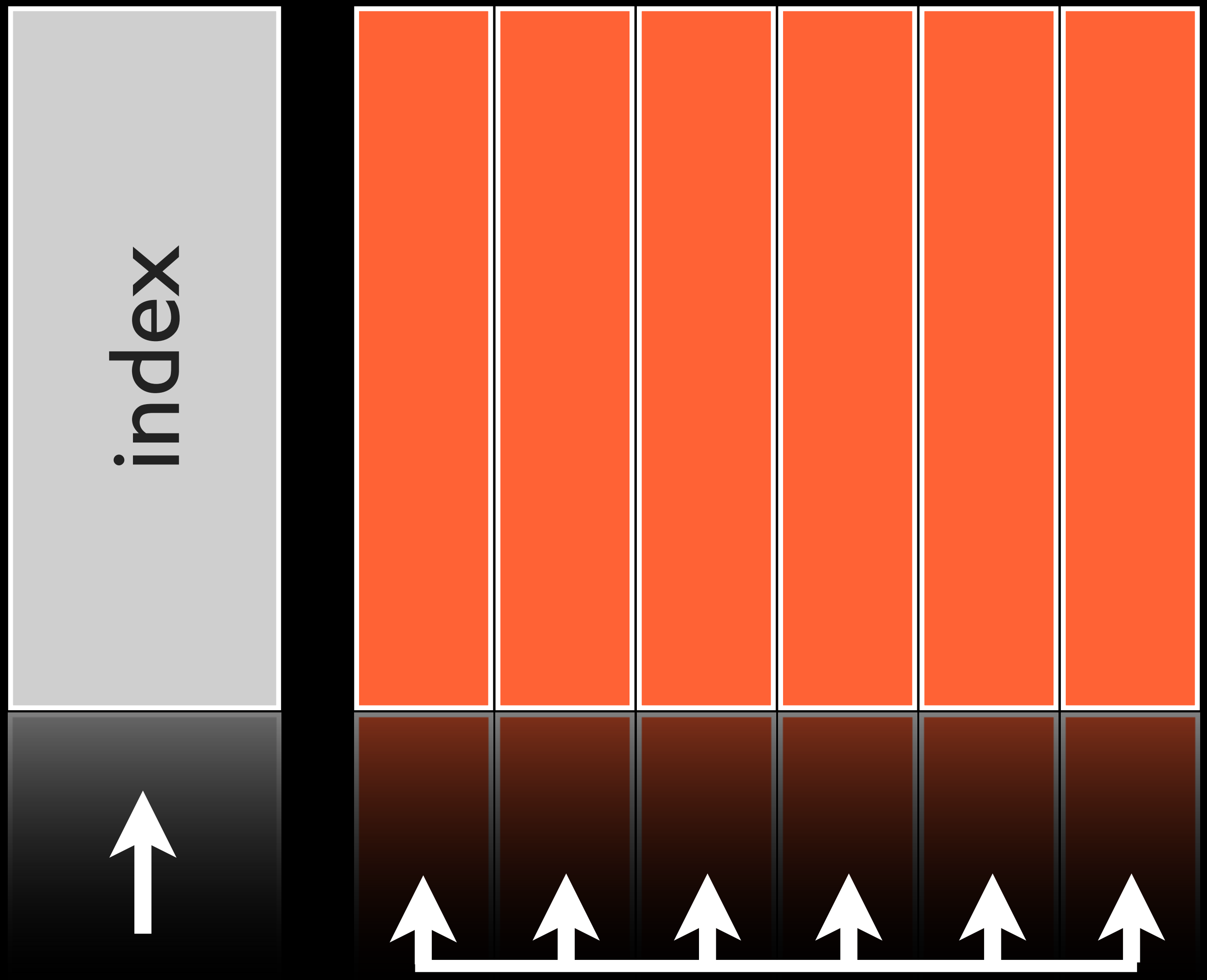
Disk

Is a pure R solution possible ?

R

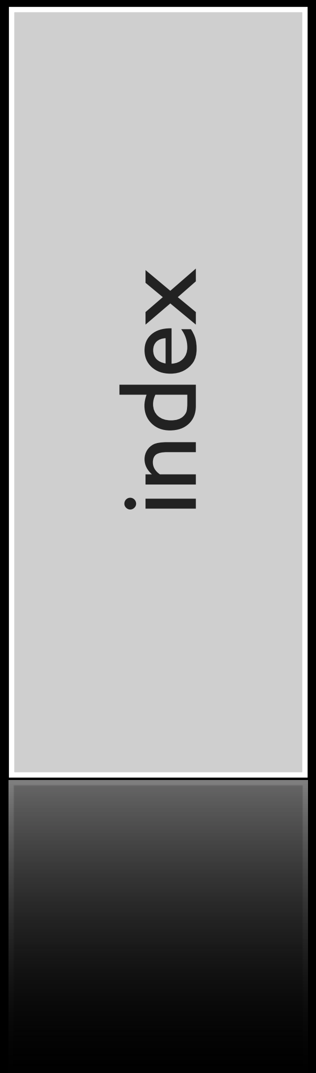
extensible time series

xts design



seconds homogenous

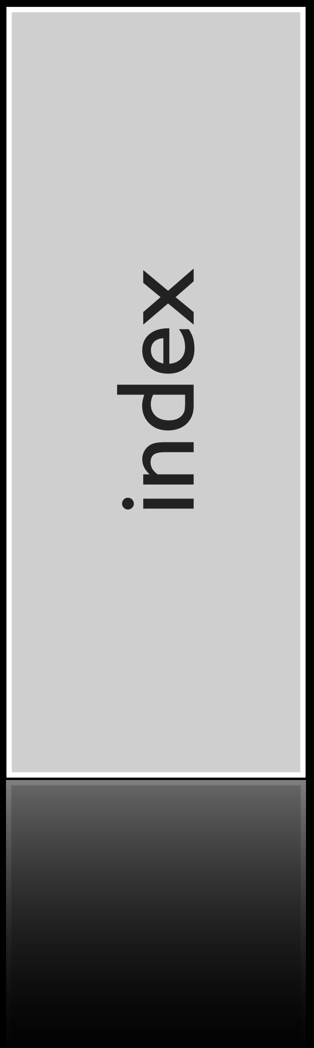
xts index



index( )

.index( )

xts index



index

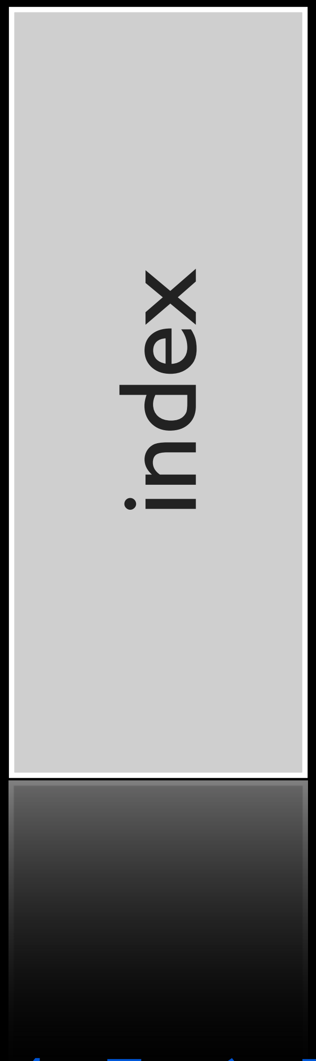
index( )

.index( )

1970-01-01 00:00:00, 1970-01-01 00:00:01,

Some “timeBased” class/representation

xts index



index( )

.index( )

c(1, 2, 3, 4, 5, 6, 7, 8, 9, 10, ..., 100, 101, 102, ...

* raw seconds since the epoch

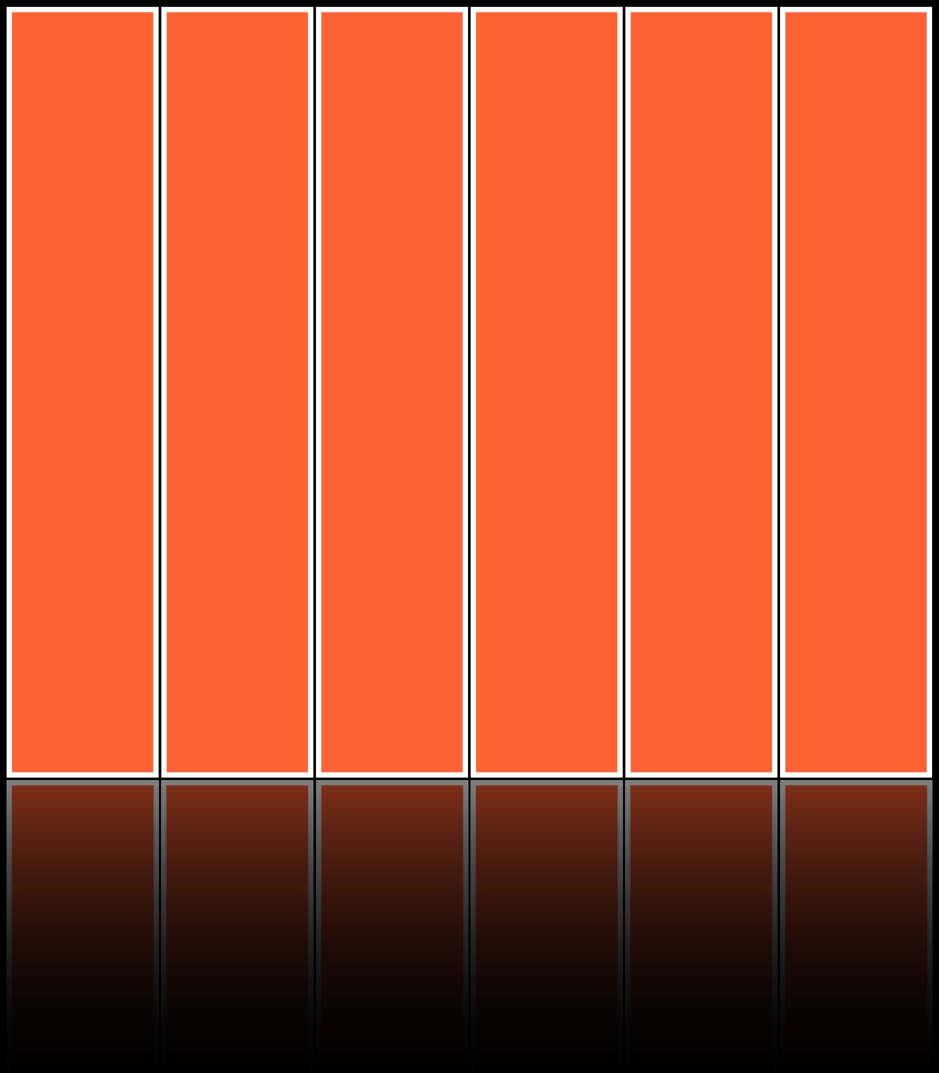
xts data

homogenous

column major

“matrix”

dense



ex.

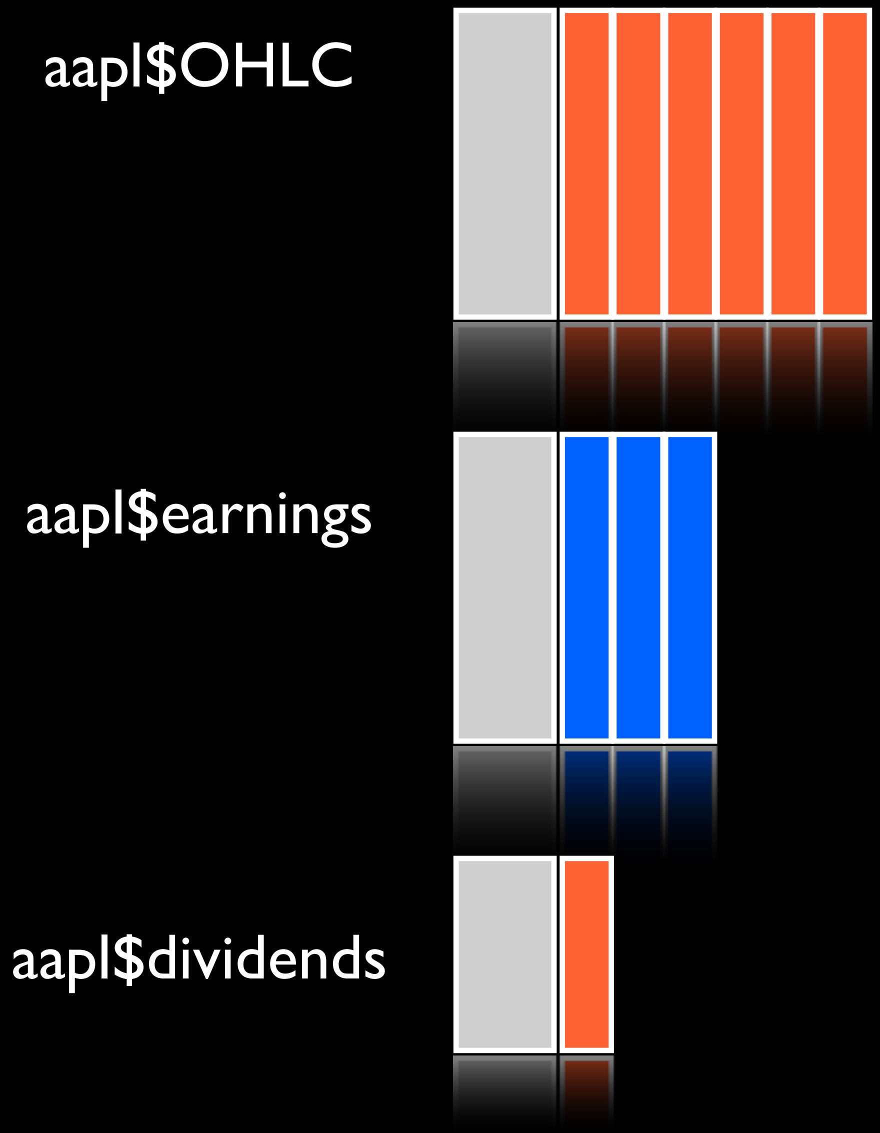
xts performance & use

subsets

merging

aggregation

xts lists



aapl$OHLC

aapl$earnings

aapl$dividends

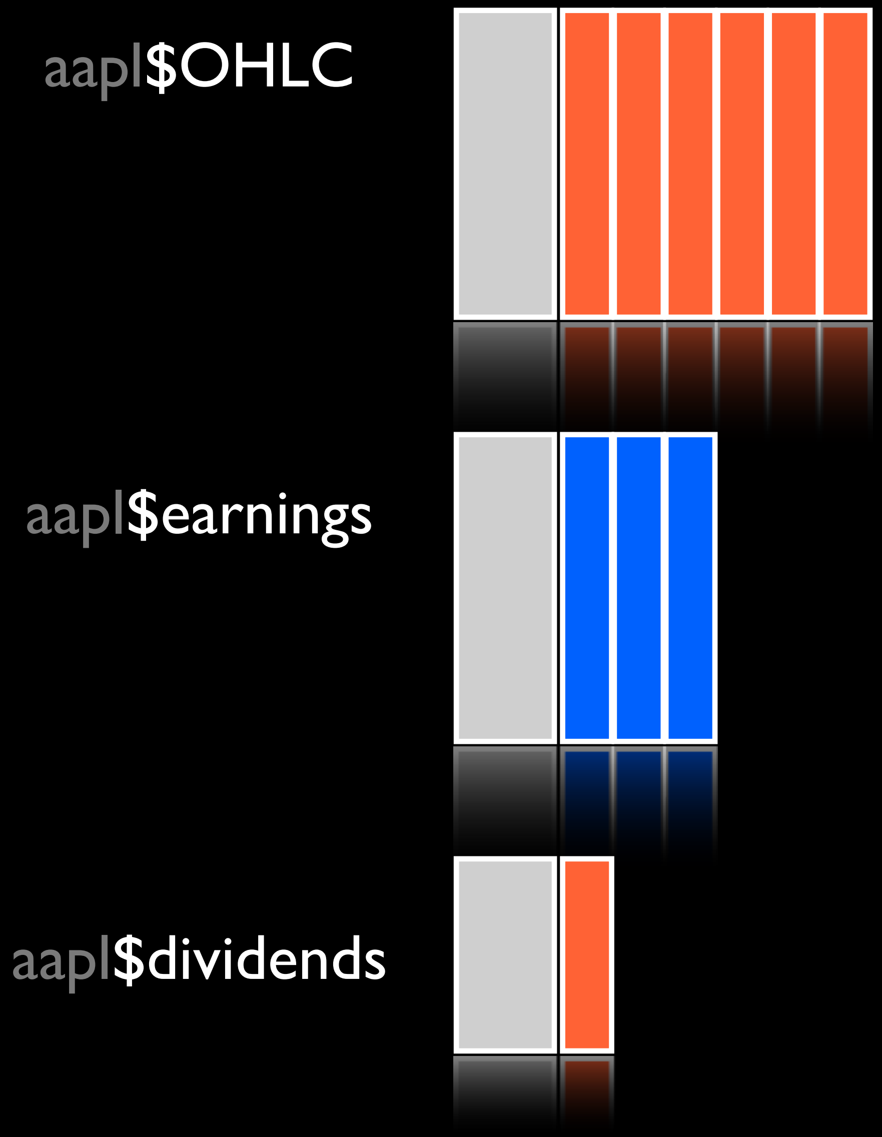
inhomogenous

column major

“matrix”

sparse

xts lists



aapl$OHLC

aapl$earnings

aapl$dividends

```
do.call(merge,
    lapply(aapl,
        function(x) {
            x['2010']
        }))
```

NoDB

(a.k.a. Persistence)

Disk

NoDB

(a.k.a. Persistence)

R Objects On Disk (.rds)

Key/Value Storage (R, BDB, ...)

OSS Column Stores (Cassandra, MonetDB, ...)

OneTick, etc.

NoDB

R Objects On Disk (.rds)

data/aapl.rds

Fastest for R

Built in

Scales well

Language dependent

NoDB

Key/Value Storage (R, BDB, ...)

KV DB

Fast for R

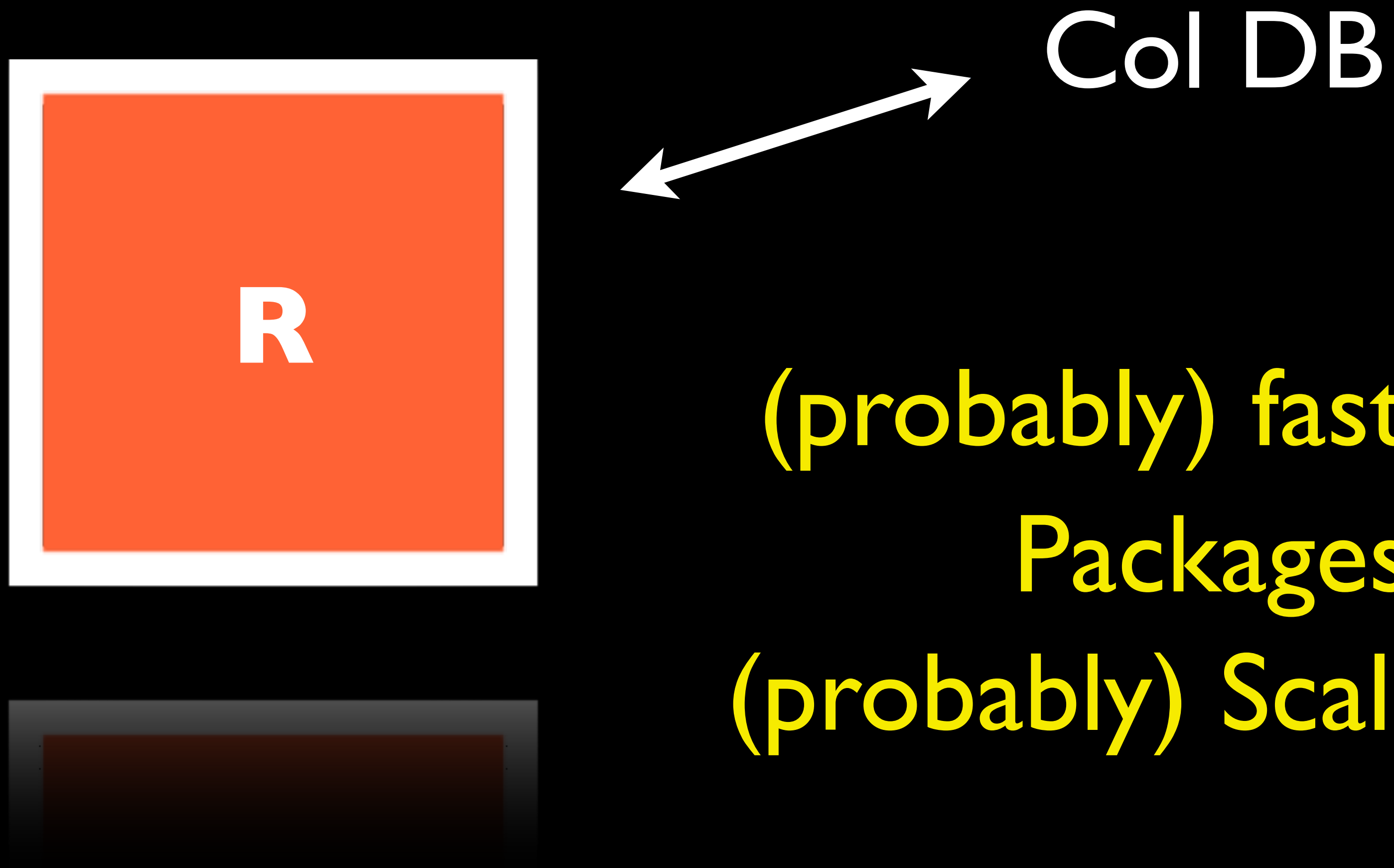
Packages

Scales well

Externally dependent

~~No~~DB

OSS Column Stores (Cassandra, MonetDB, ...)



Col DB

(probably) fast for R

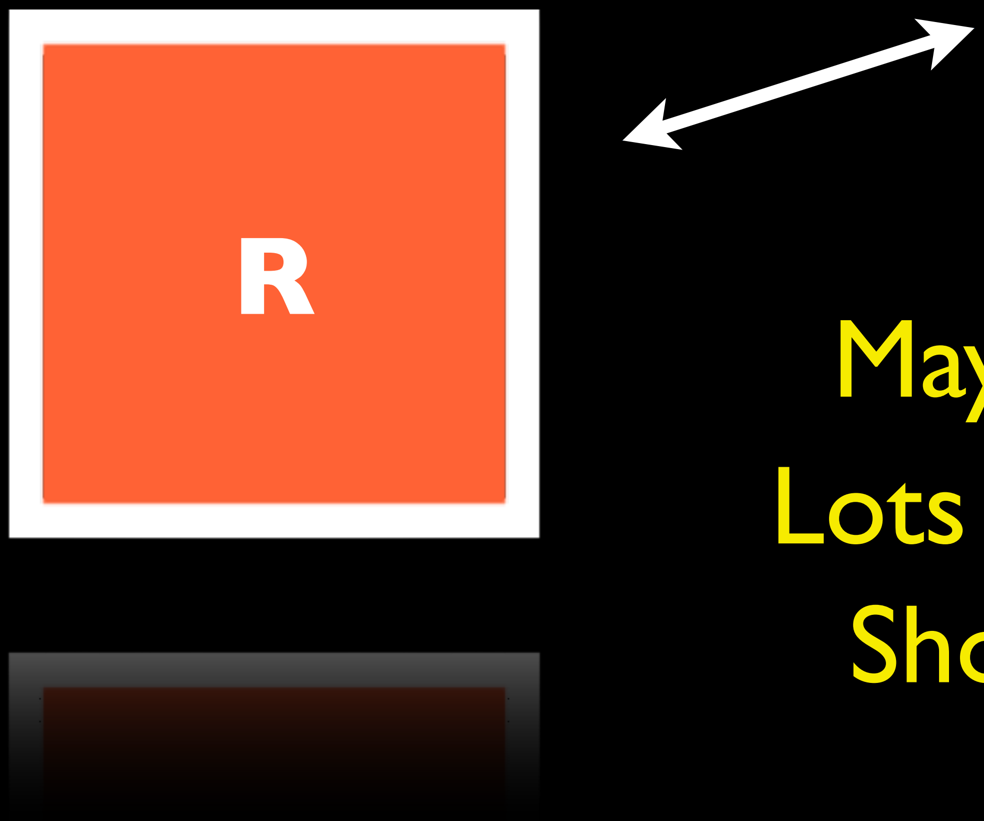
Packages

(probably) Scales well

(Complex) External dependency

~~No~~DB

Commercial Fin DBs: OneTick, etc.



Commercial
Store

May be fast for R

Lots of hand coding

Should scale well

(Complex) External dependency

Expensive for many

ex.

NoDB

(a.k.a. Persistence)

R Objects On Disk (.rds)

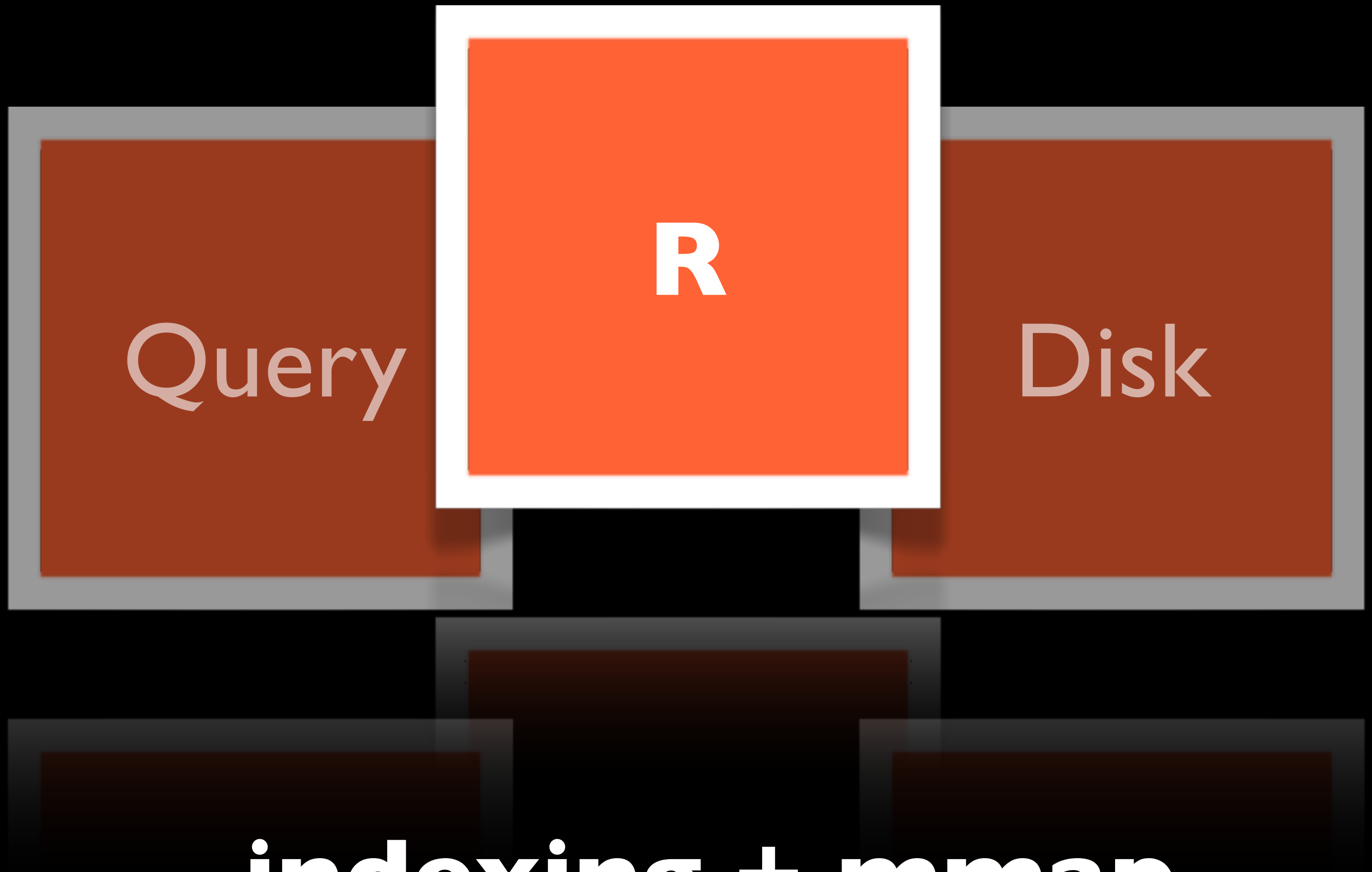
Key/Value Storage (R, BDB, ...)

OSS Column Stores (Cassandra, MonetDB, ...)

OneTick, etc.

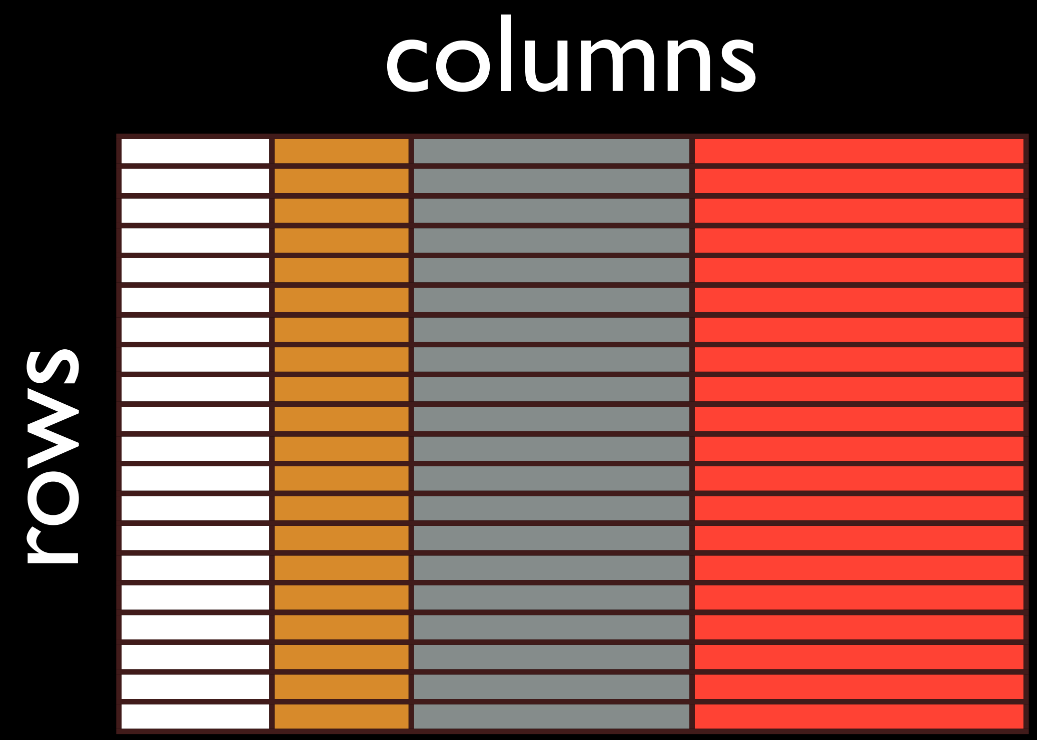
NoDB

(aka reinvent the wheel)



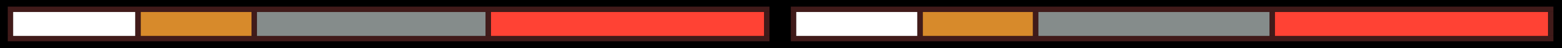
indexing + mmap

Database Design101

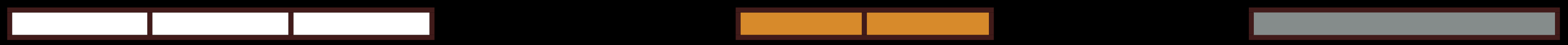


Row or Column Based

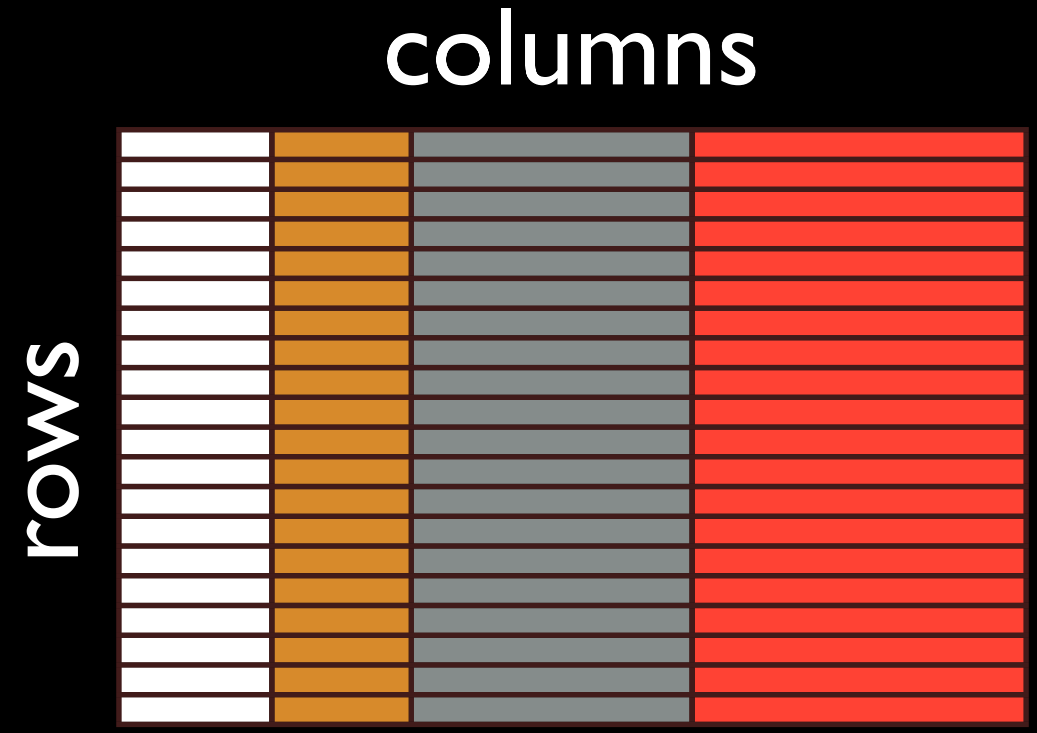
By Row



By Column



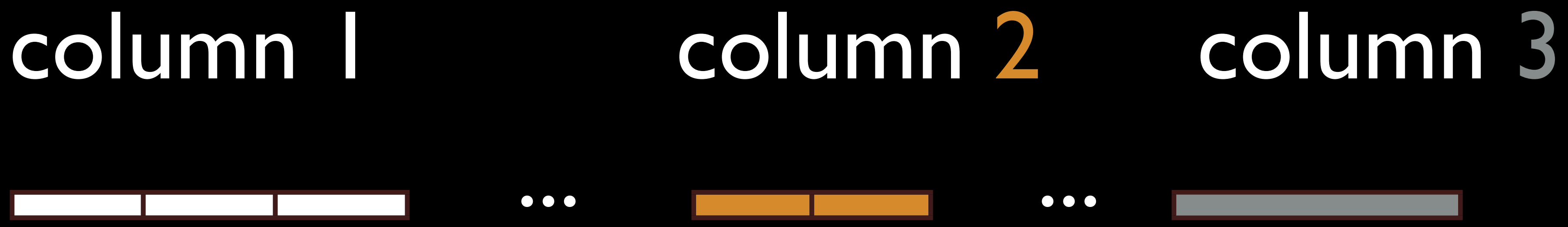
R is Column Oriented



i.e.

data.frames store column values sequentially

a.k.a column-major



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

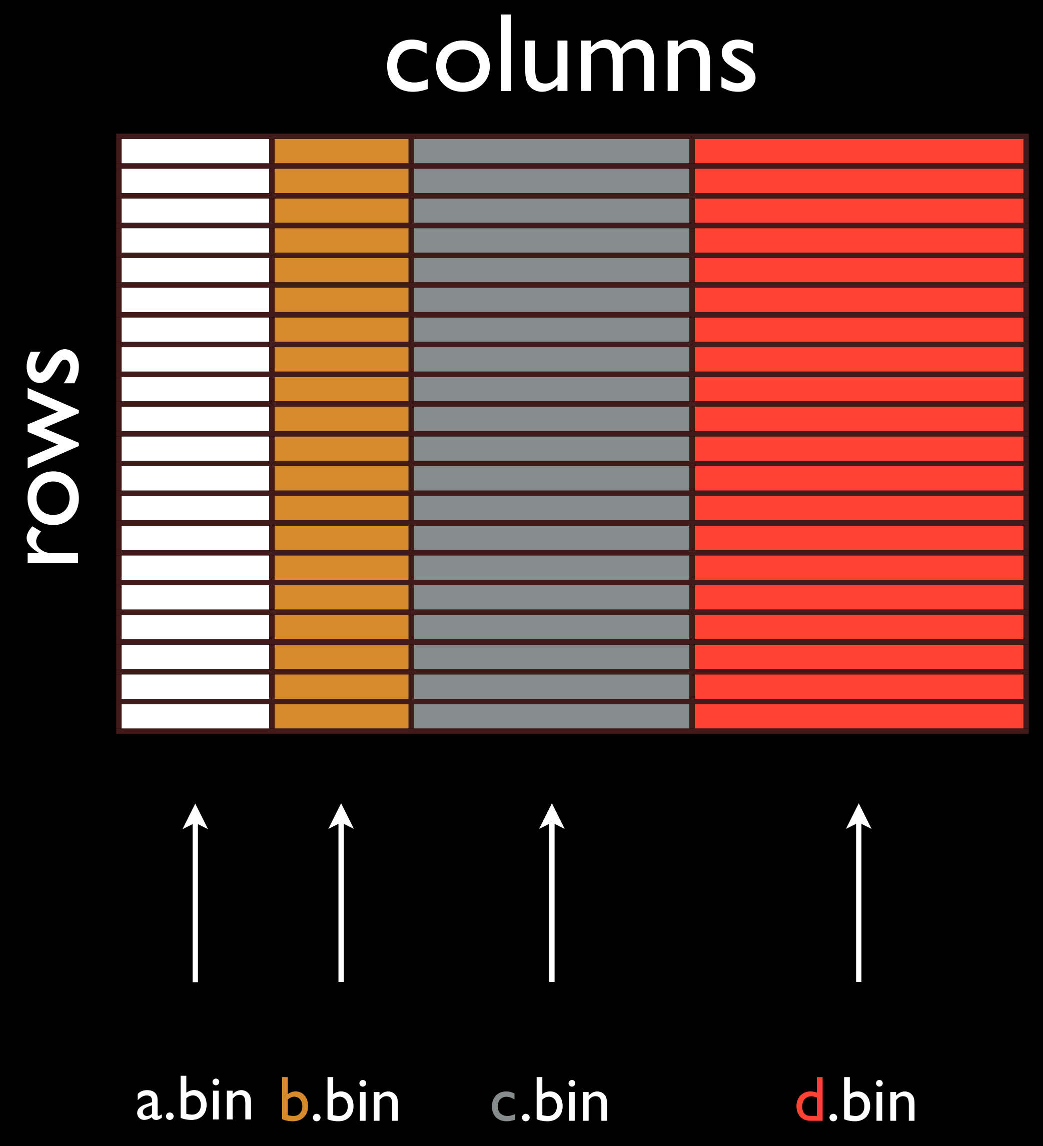
O(log n)

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 + 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

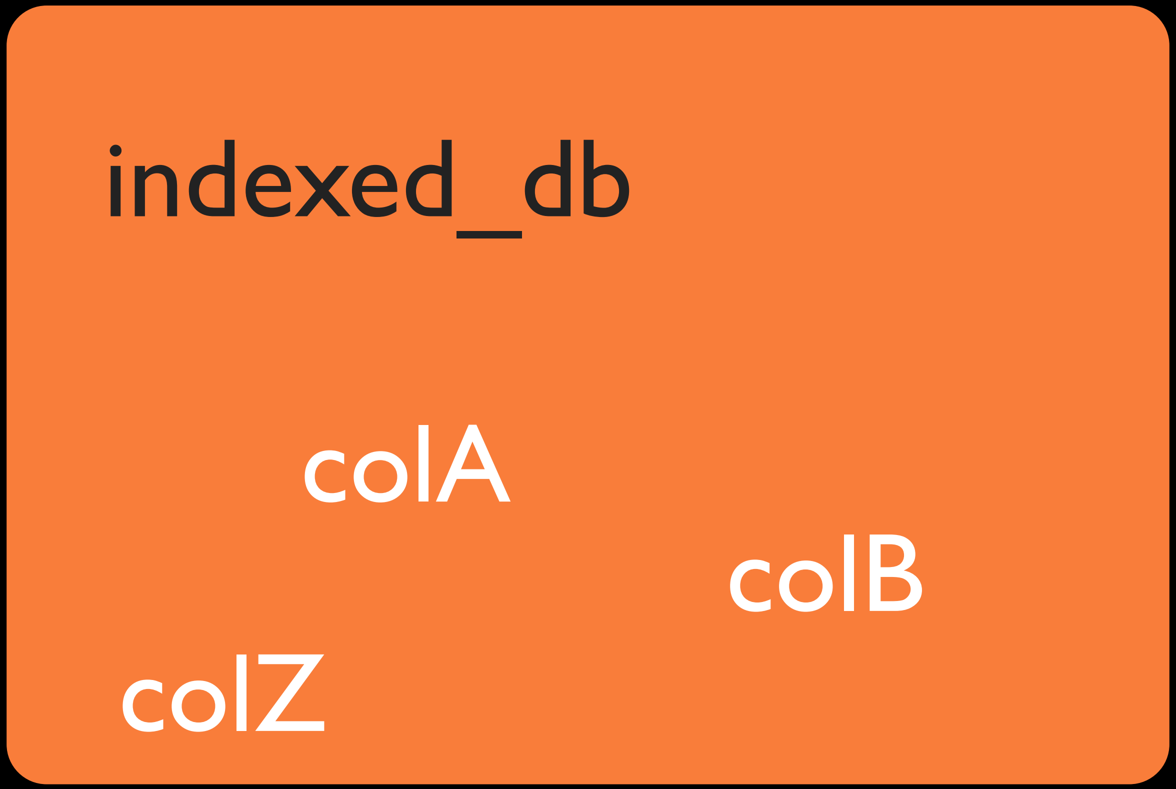
networked

query optimization

caching

mmap + indexing

indexed_db
is an
environment



indexed_db

colA

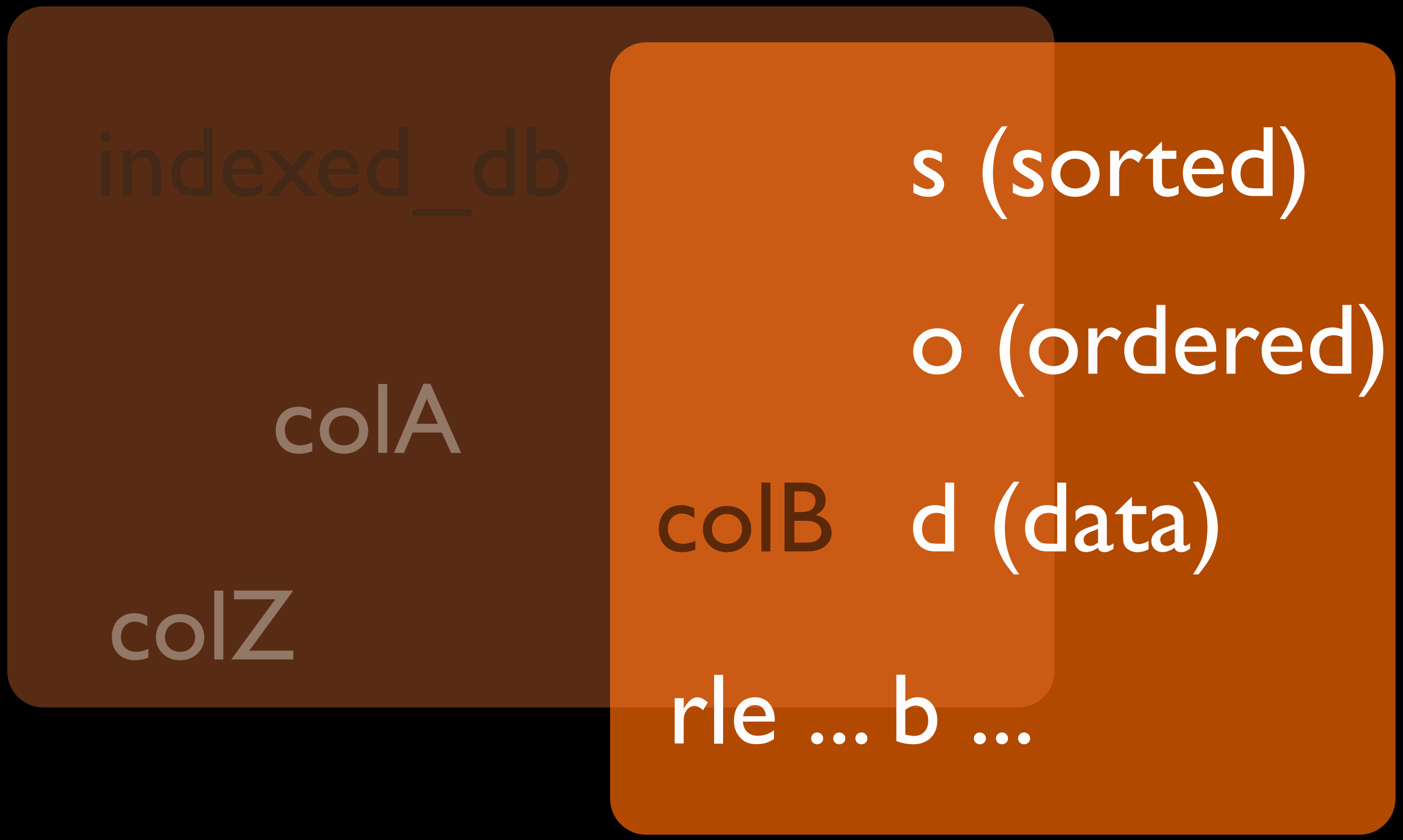
colB

colZ

colA - Z are
“columns”
of your data

“columns” are really objects (lists) in the environment

mmap + indexing



indexed_db

colA

colZ

colB

s (sorted)

o (ordered)

d (data)

rle ... b ...

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

6 queries in 3.3 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%

Conclusion And Caveats

Nothing is free

R centric workflow vs. DB

Understand your domain and requirements

www.lemnica.com