High-Performance Processing of Large Data Sets via Memory Mapping A Case Study in R and C++

Daniel Adler, Jens Oelschlägel, Oleg Nenadic, Walter Zucchini

Georg-August University Göttingen, Germany - Research Consultant, Munich, Germany

Joint Statistical Meeting - 5th of August 2008 - Denver, USA

Motivation

Working with large data sets in R is restricted by virtual memory and the virtual address space.

Motivation

Working with large data sets in R is restricted by virtual memory and the virtual address space.

R console-mode (standard edition) on Mac OS X 10.5, 4 GB

```
> numeric(1024^3*2.3/8)
Error: cannot allocate vector of size 2.3 Gb
```

Motivation

Working with large data sets in R is restricted by virtual memory and the virtual address space.

R console-mode (standard edition) on Mac OS X 10.5, 4 GB

```
> numeric(1024^3*2.3/8)
Error: cannot allocate vector of size 2.3 Gb
```

R console-mode on Linux 2.6 (Ubuntu), 2 GB RAM (Parallels VM)

```
> numeric(1024^3*2.4/8)
Error: cannot allocate vector of size 2.4 Gb
```

Motivation

Working with large data sets in R is restricted by virtual memory and the virtual address space.

R console-mode (standard edition) on Mac OS X 10.5, 4 GB

```
> numeric(1024^3*2.3/8)
Error: cannot allocate vector of size 2.3 Gb
```

R console-mode on Linux 2.6 (Ubuntu), 2 GB RAM (Parallels VM)

```
> numeric(1024^3*2.4/8)
Error: cannot allocate vector of size 2.4 Gb
```

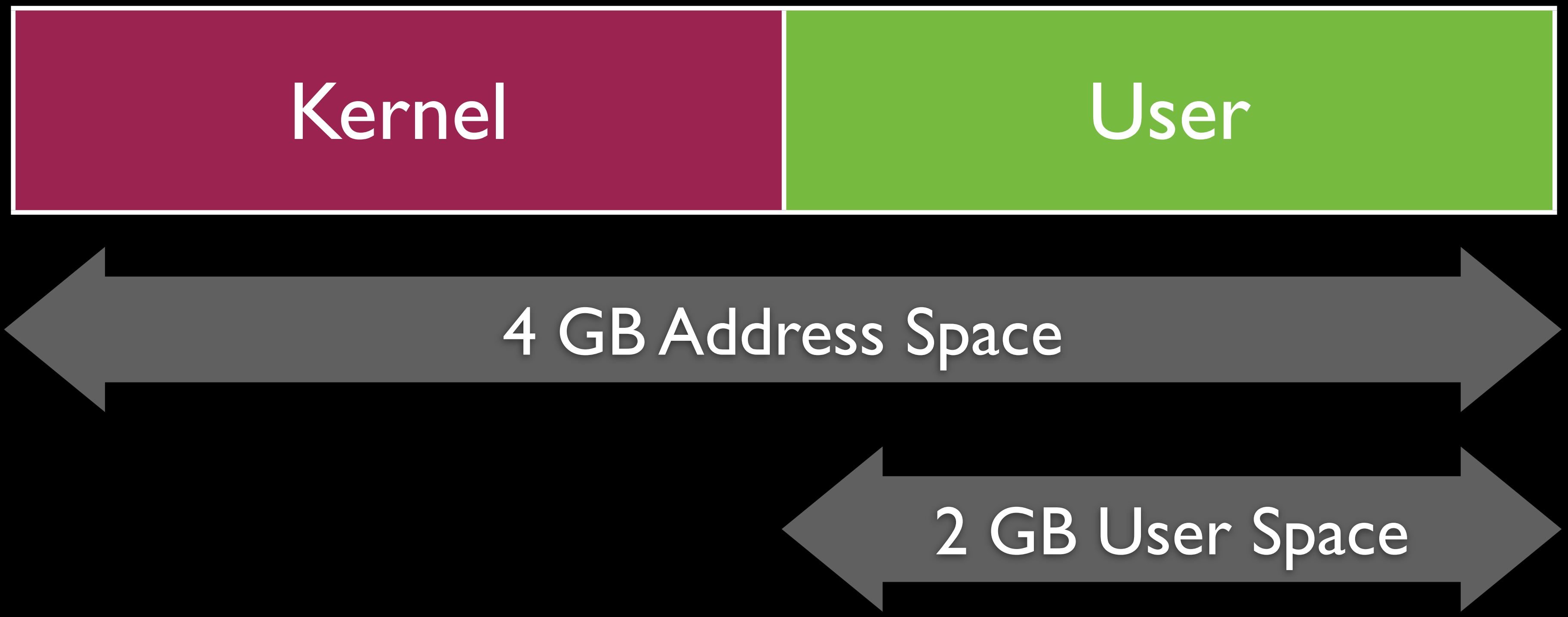
R on Windows XP, 2 GB RAM (Parallels VM)

```
> numeric(1024^3*2/8)
Error: cannot allocate vector of size 2.0 Gb
Reached total allocation of 1535Mb..
> memory.limit()
[1] 1535.36
```

Motivation

Working with large data sets in R is restricted by virtual memory and the virtual address space.

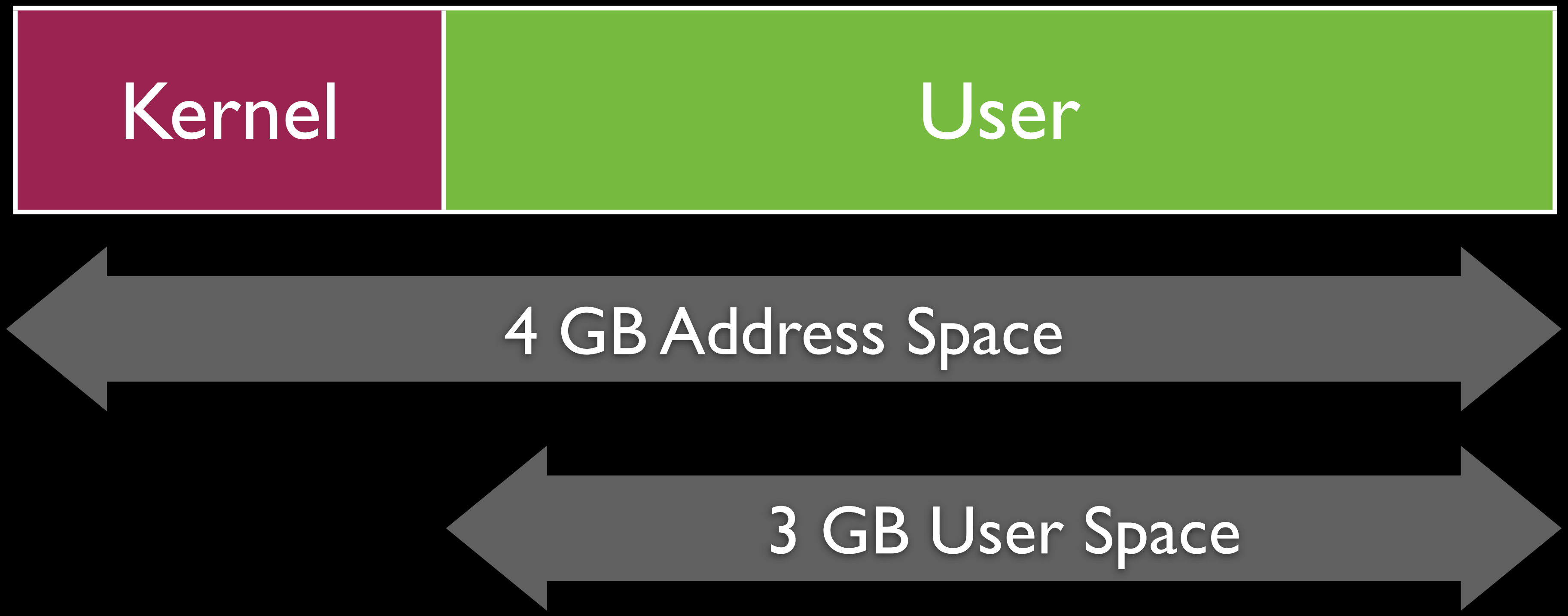
R on 32-Bit Windows



Motivation

Working with large data sets in R is restricted by virtual memory and the virtual address space.

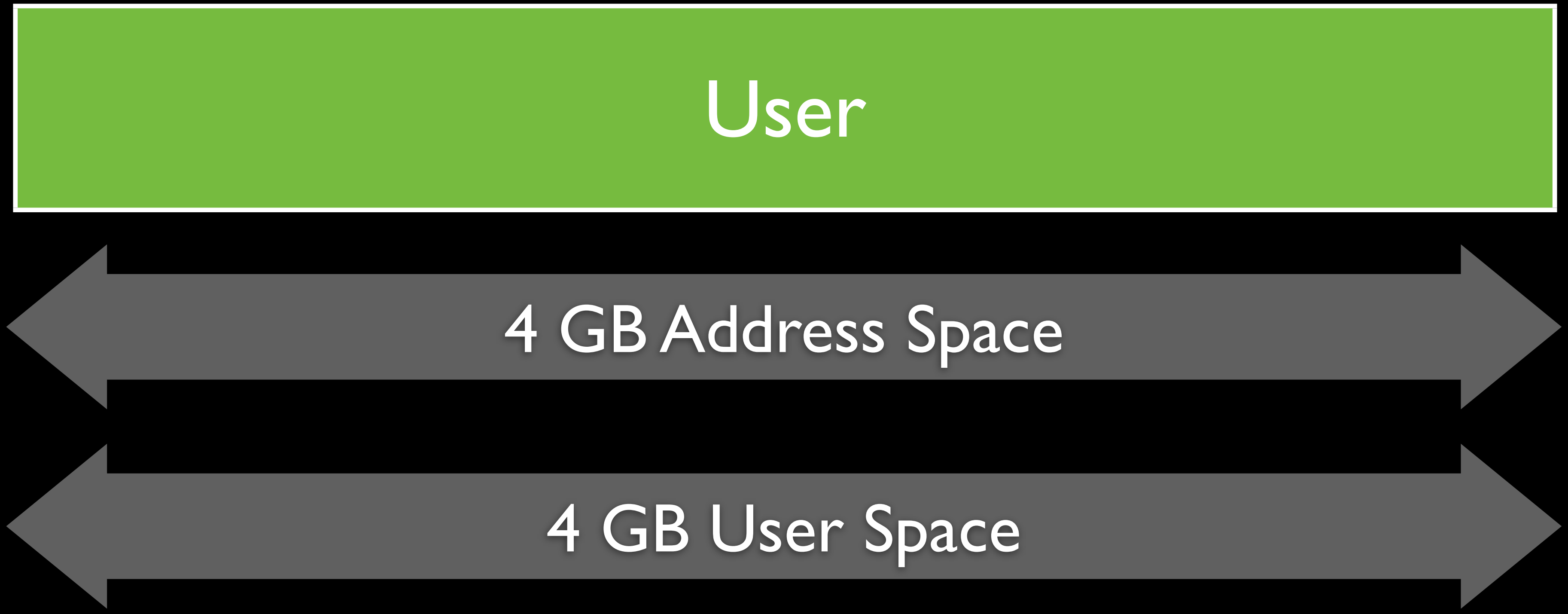
R on 32-Bit Windows
(tweaked with "/3GB")



Motivation

Working with large data sets in R is restricted by virtual memory and the virtual address space.

R on 64-Bit Windows



User

4 GB Address Space

4 GB User Space

Motivation

Working with large data sets in R is restricted by virtual memory and the virtual address space.

Goal

Enabling work with large data sets on desktop PCs.

Motivation

Working with large data sets in R is restricted by virtual memory and the virtual address space.

Goal

Enabling work with large data sets on desktop PCs.

Idea

Data resides on disk storage. Parsimonious use of virtual address space via “paging”.

Motivation

Working with large data sets in R is restricted by virtual memory and the virtual address space.

Goal

Enabling work with large data sets on desktop PCs.

Idea

Data resides on disk storage. Parsimonious use of virtual address space via “paging”.

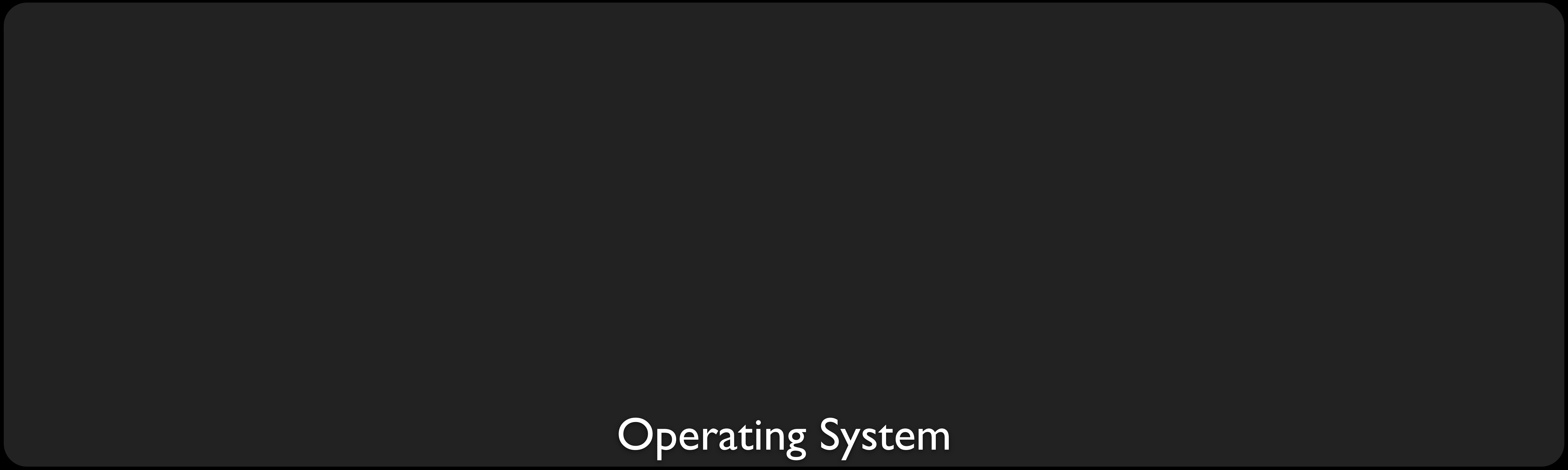
Efficiency problem

Disk I/O is slow (1 million times slower than RAM).

Overview

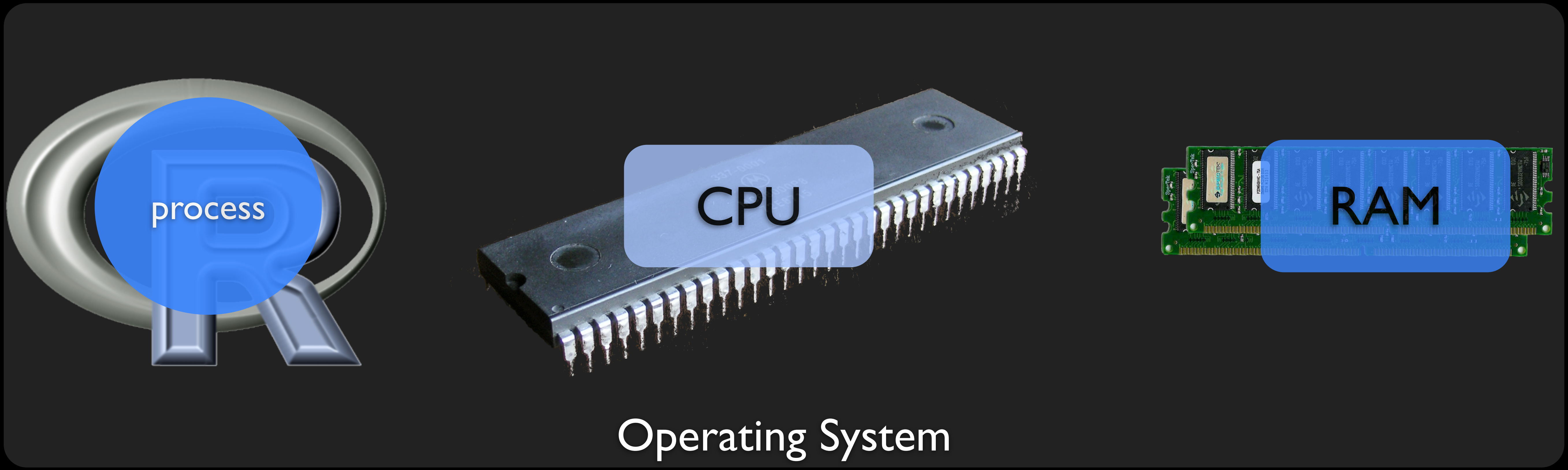
- 'ff' low-level C++ Library (Daniel)
  Virtual Memory & Memory-Mapping,
  Flat Files & Paging
- 'ff' high-level R Package (Jens)
  Virtual Atomic Objects, Batch processing,
  Data types, Hybrid Indexing
- Performance
  Page size & System cache, Enhancements
- Epilog
  Possible Improvements & Conclusion

Virtual Memory and Memory-Mapping



Operating System

Virtual Memory and Memory-Mapping



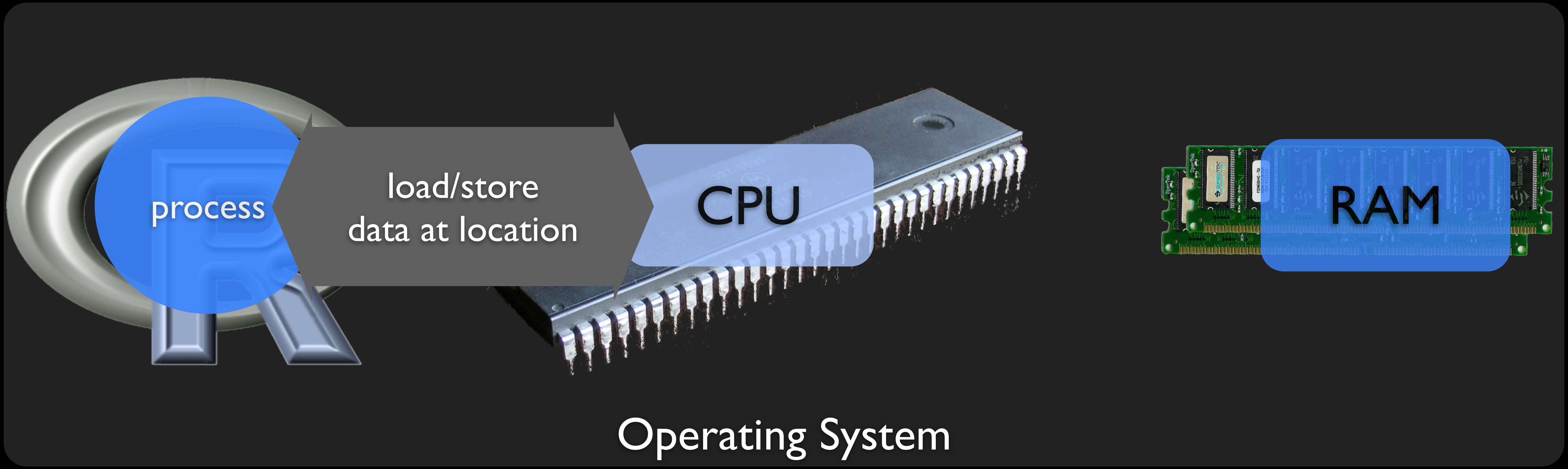
process

CPU

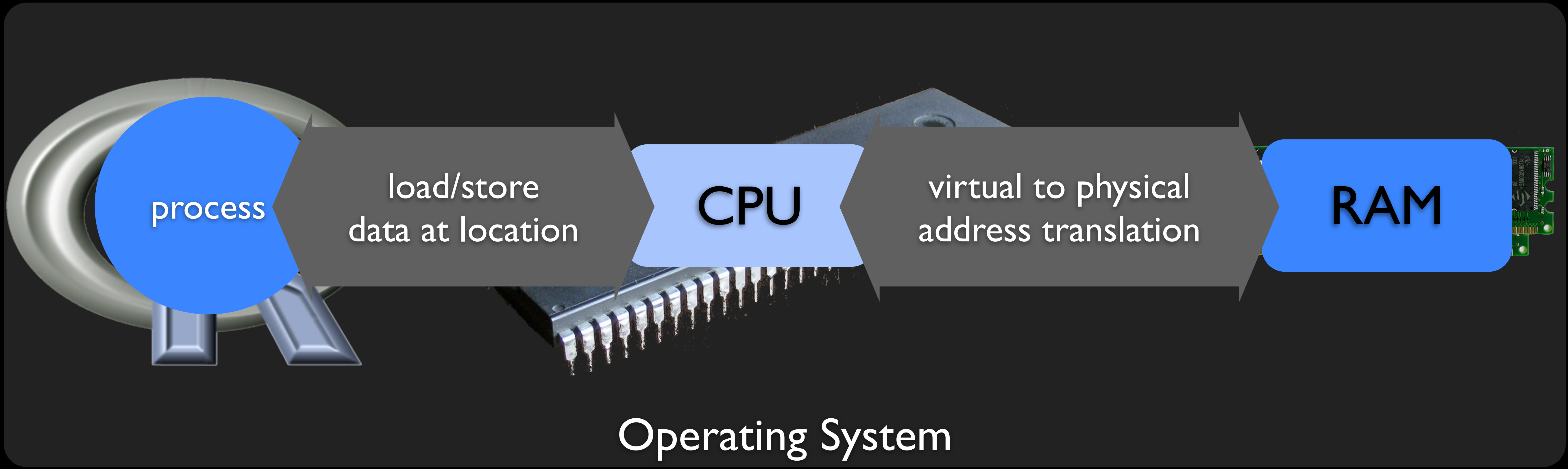
RAM

Operating System

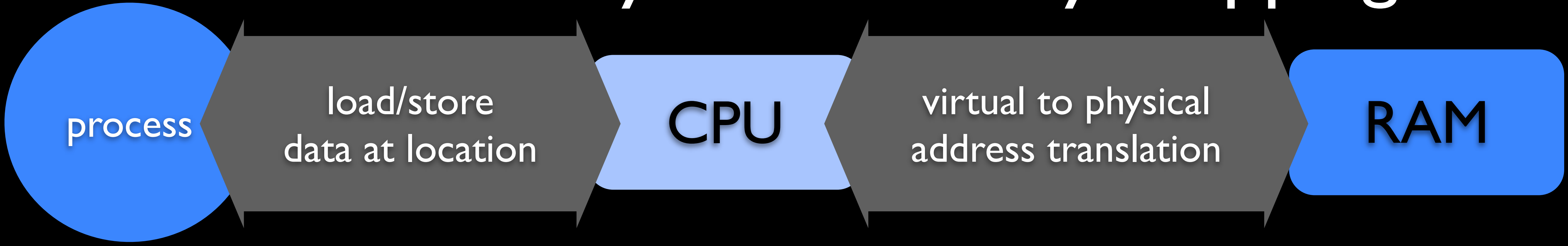
Virtual Memory and Memory-Mapping

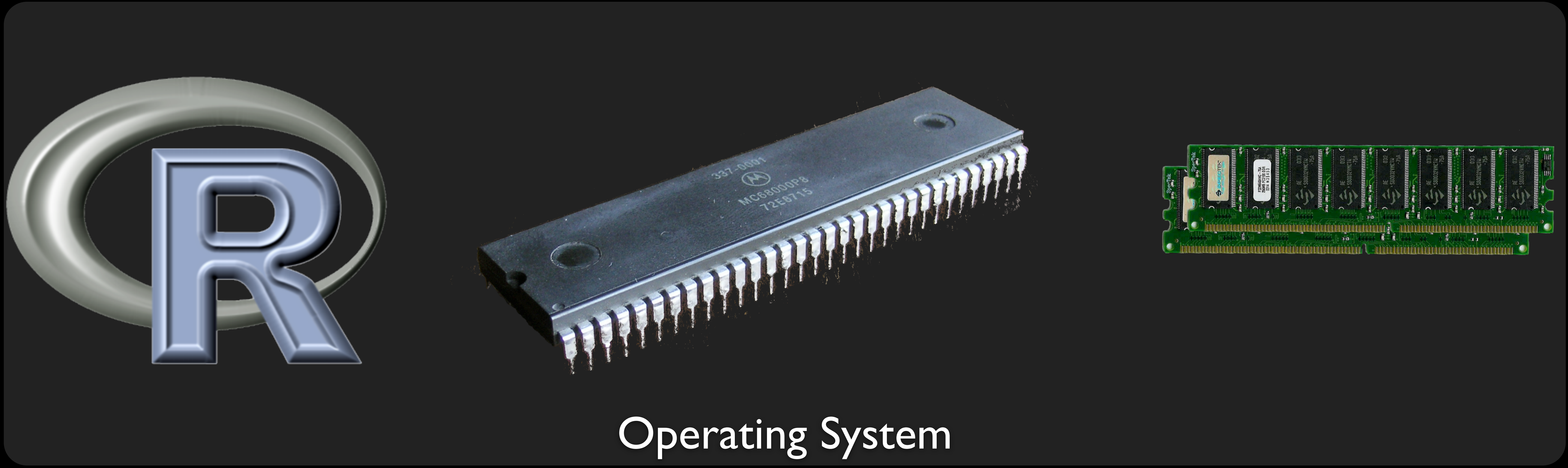


Virtual Memory and Memory-Mapping

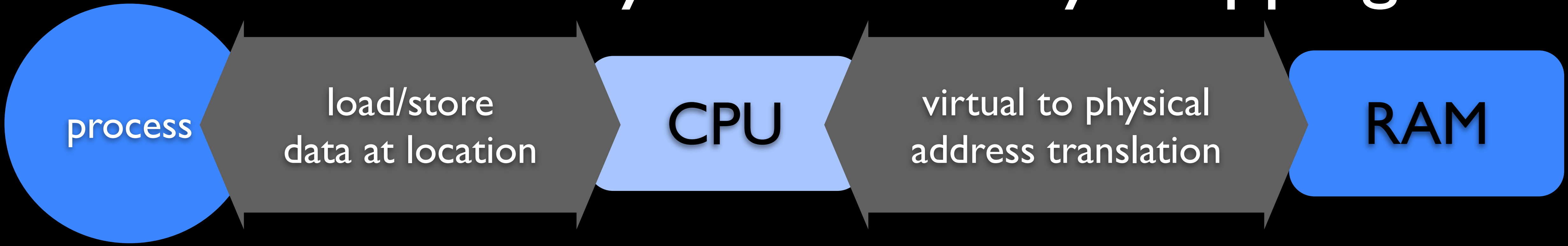


Virtual Memory and Memory-Mapping





Virtual Memory and Memory-Mapping



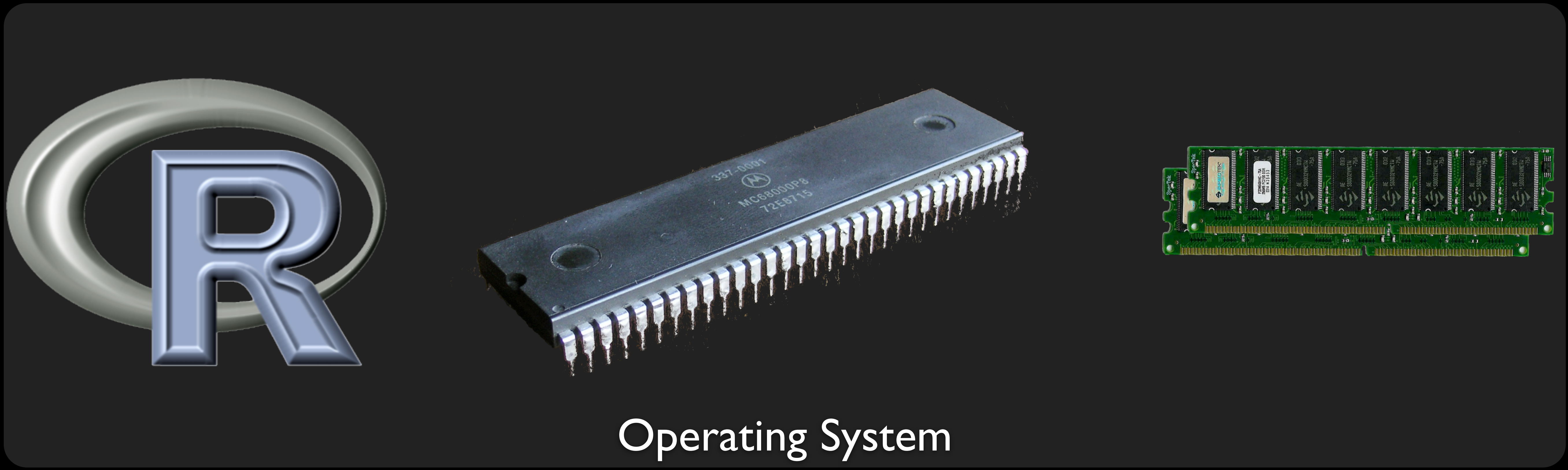
process

load/store data at location

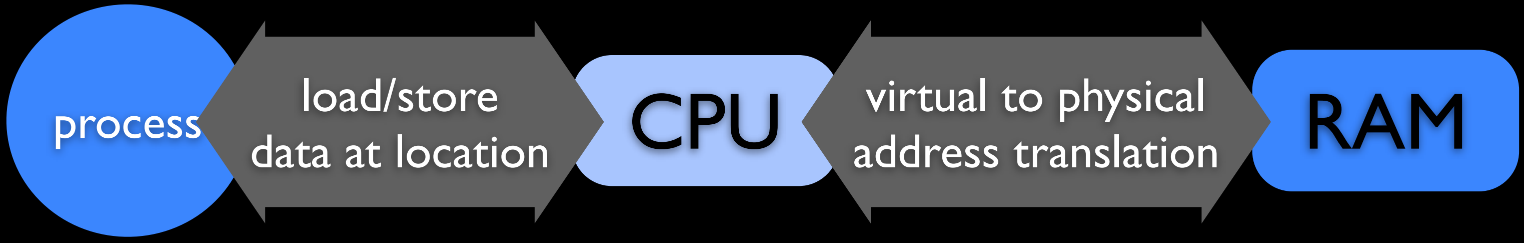
CPU

virtual to physical address translation

RAM



Virtual Memory and Memory-Mapping



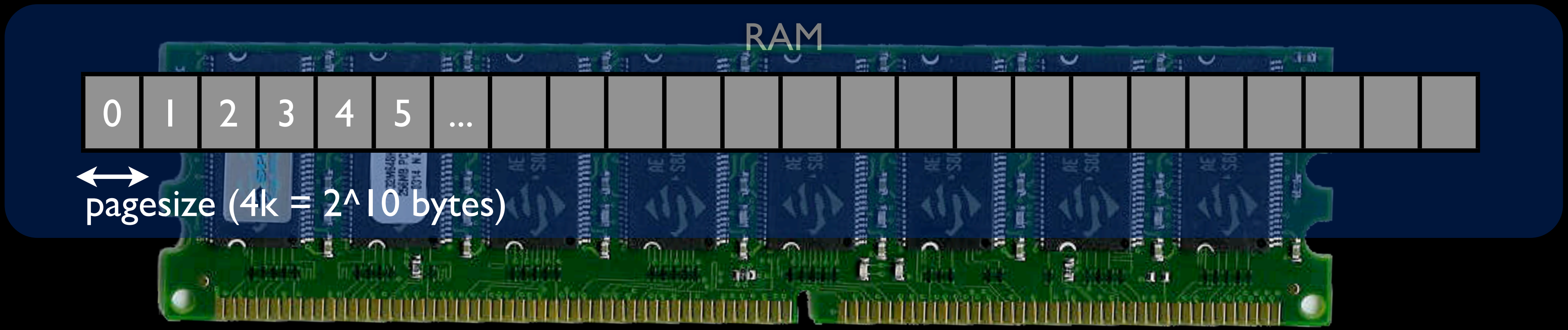
process

load/store
data at location

CPU

virtual to physical
address translation

RAM

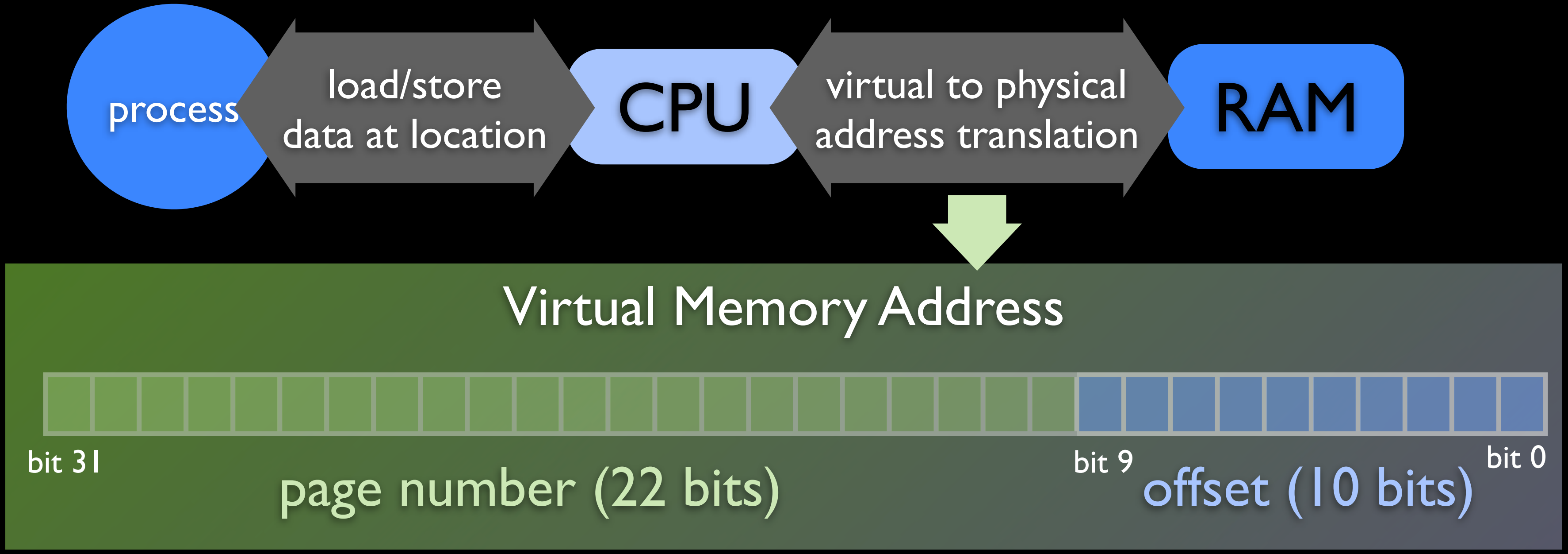


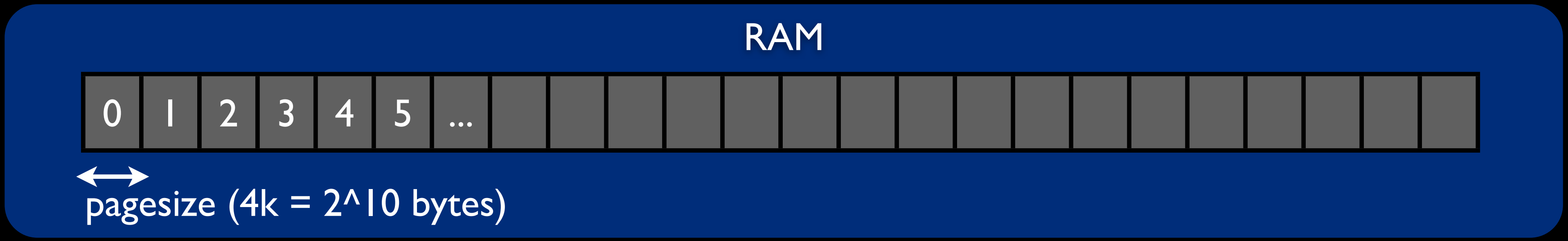
RAM

0 | 1 | 2 | 3 | 4 | 5 | ...

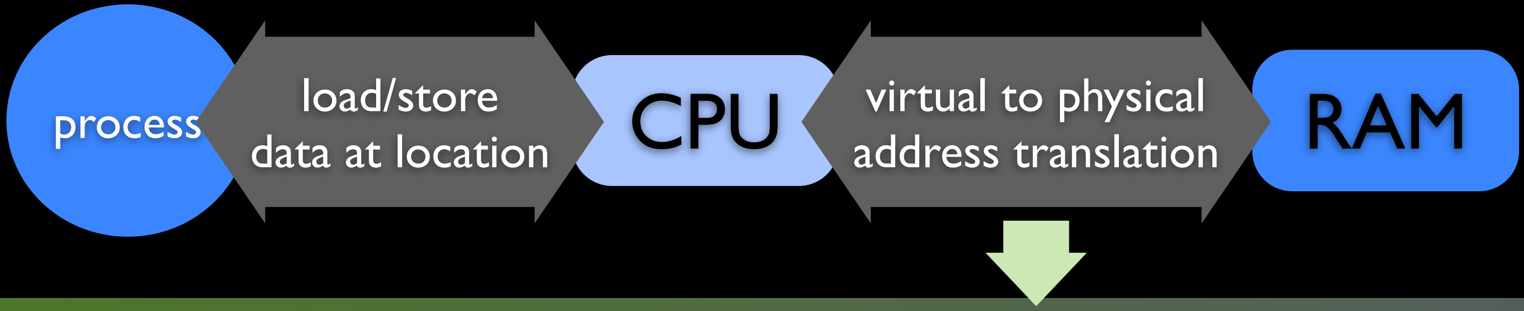
pagesize (4k = 2^10 bytes)

Virtual Memory and Memory-Mapping





Virtual Memory and Memory-Mapping



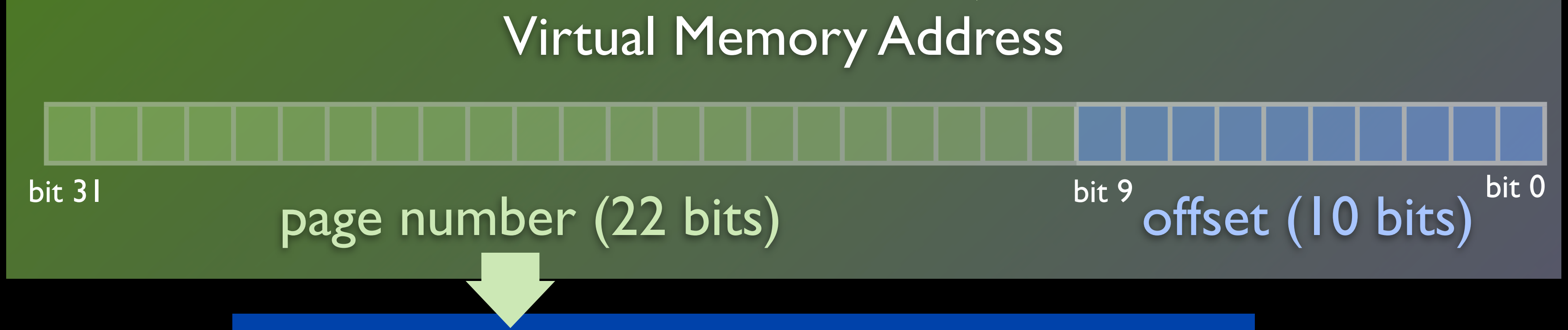
process

load/store data at location

CPU

virtual to physical address translation

RAM



Virtual Memory Address

bit 31

bit 9

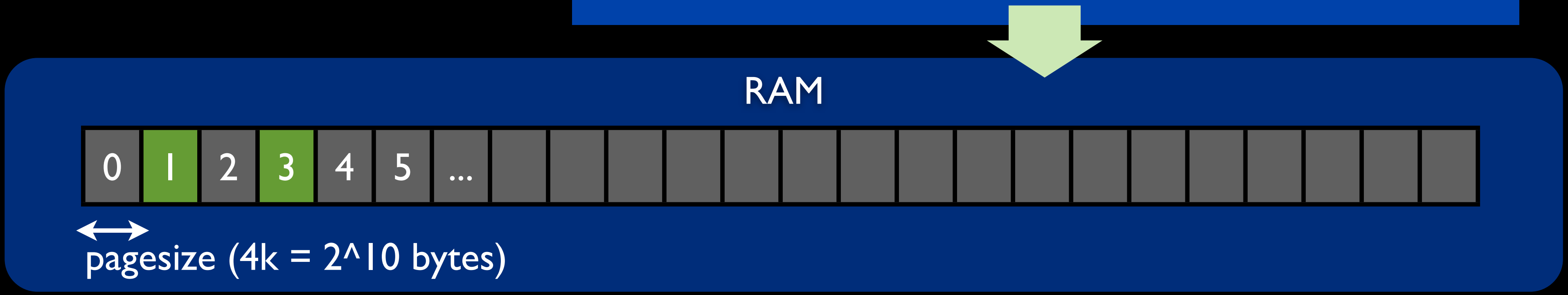
bit 0

page number (22 bits)

offset (10 bits)

Process Page Table, e.g

Page	Physical location
200	RAM page 3
201	RAM page 1
202	N/A

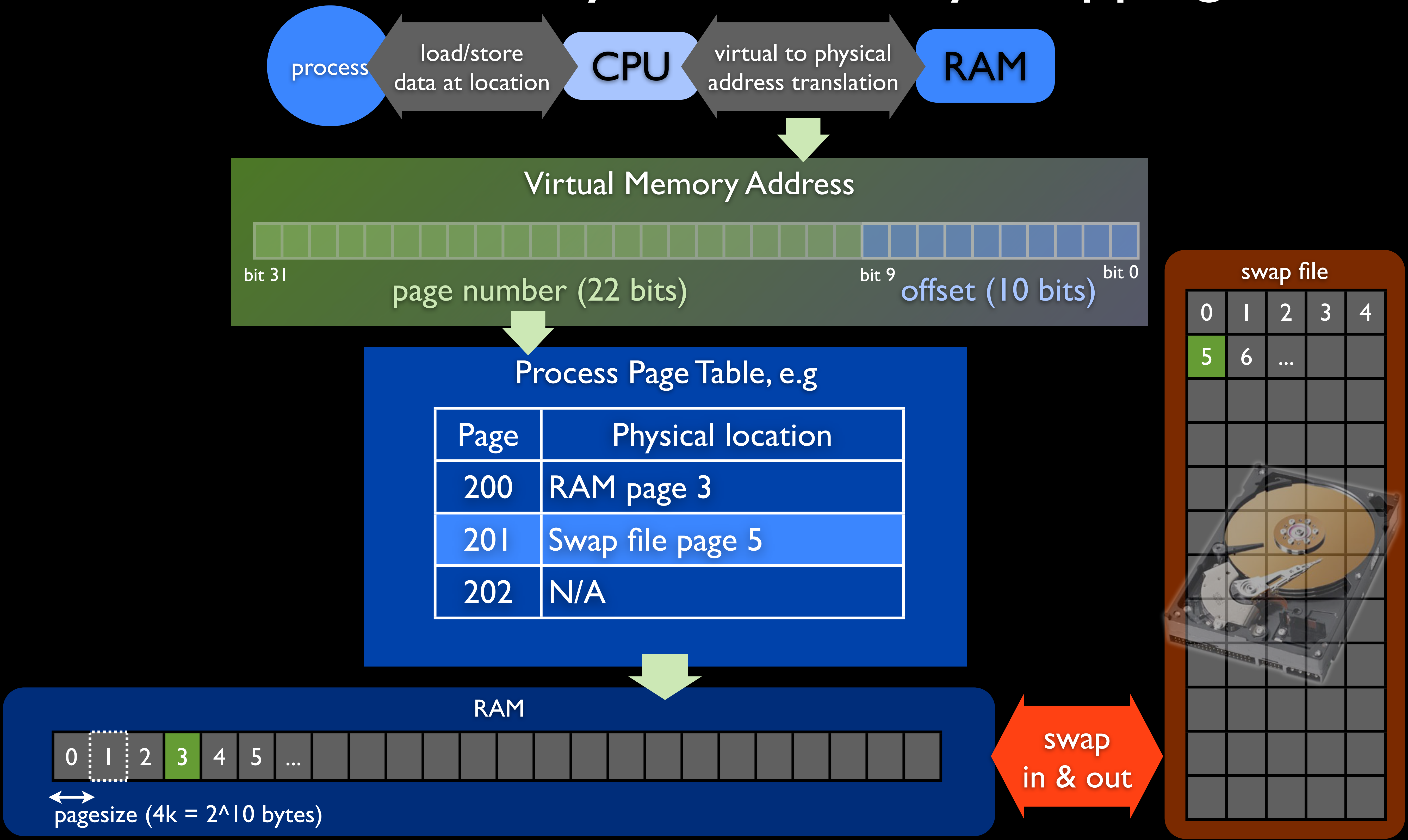


RAM

0 1 2 3 4 5 ...

pagesize (4k = 2^10 bytes)

Virtual Memory and Memory-Mapping



process

load/store data at location

CPU

virtual to physical address translation

RAM

Virtual Memory Address

bit 31

page number (22 bits)

bit 9

offset (10 bits)

bit 0

Process Page Table, e.g

Page	Physical location
200	RAM page 3
201	Swap file page 5
202	N/A

RAM

0 1 2 3 4 5 ...

pagesize (4k = 2^10 bytes)

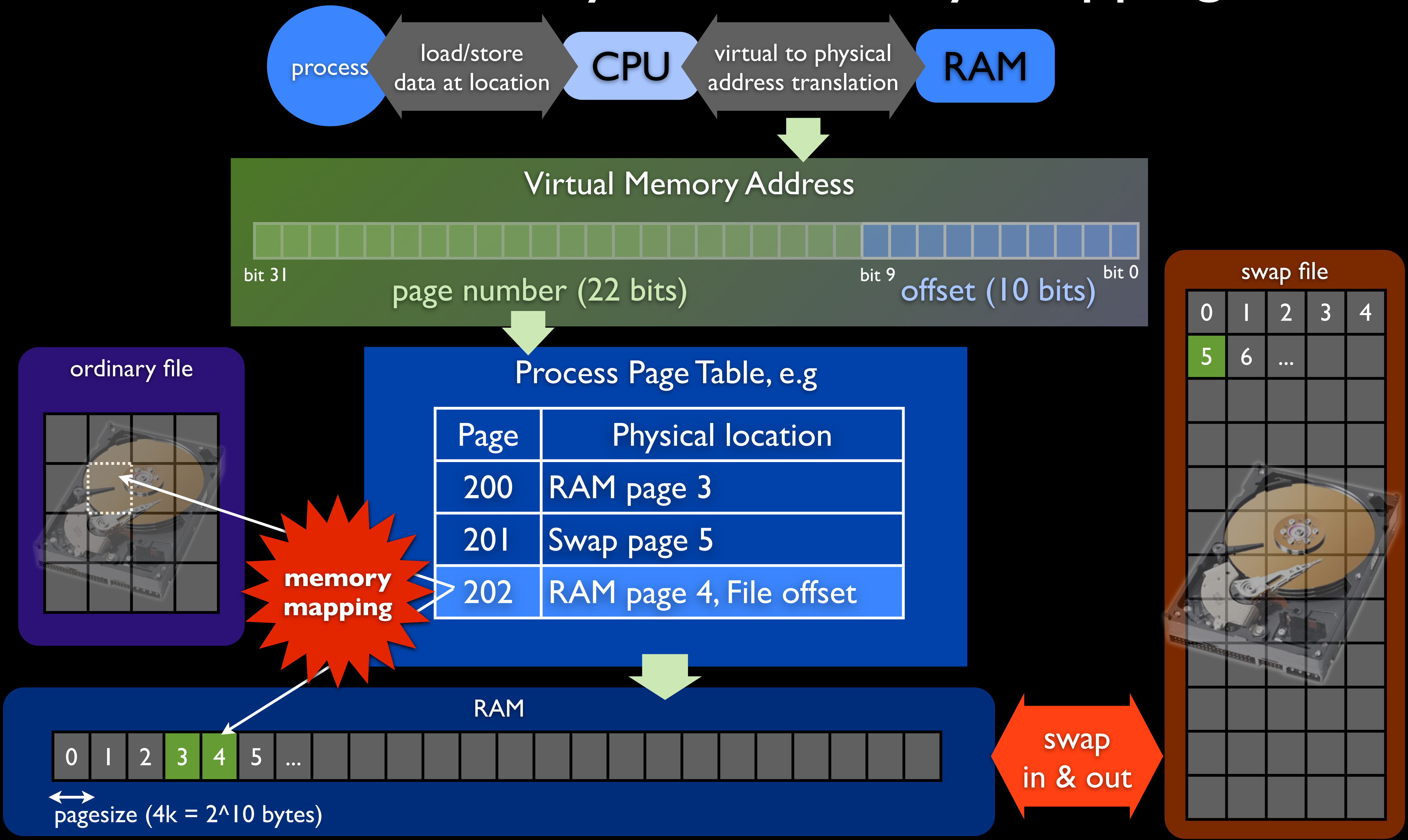
swap in & out

swap file

0 1 2 3 4

5 6 ...

Virtual Memory and Memory-Mapping



Page	Physical location
200	RAM page 3
201	Swap page 5
202	RAM page 4, File offset

Flat Files and Paging

Flat Files and Paging

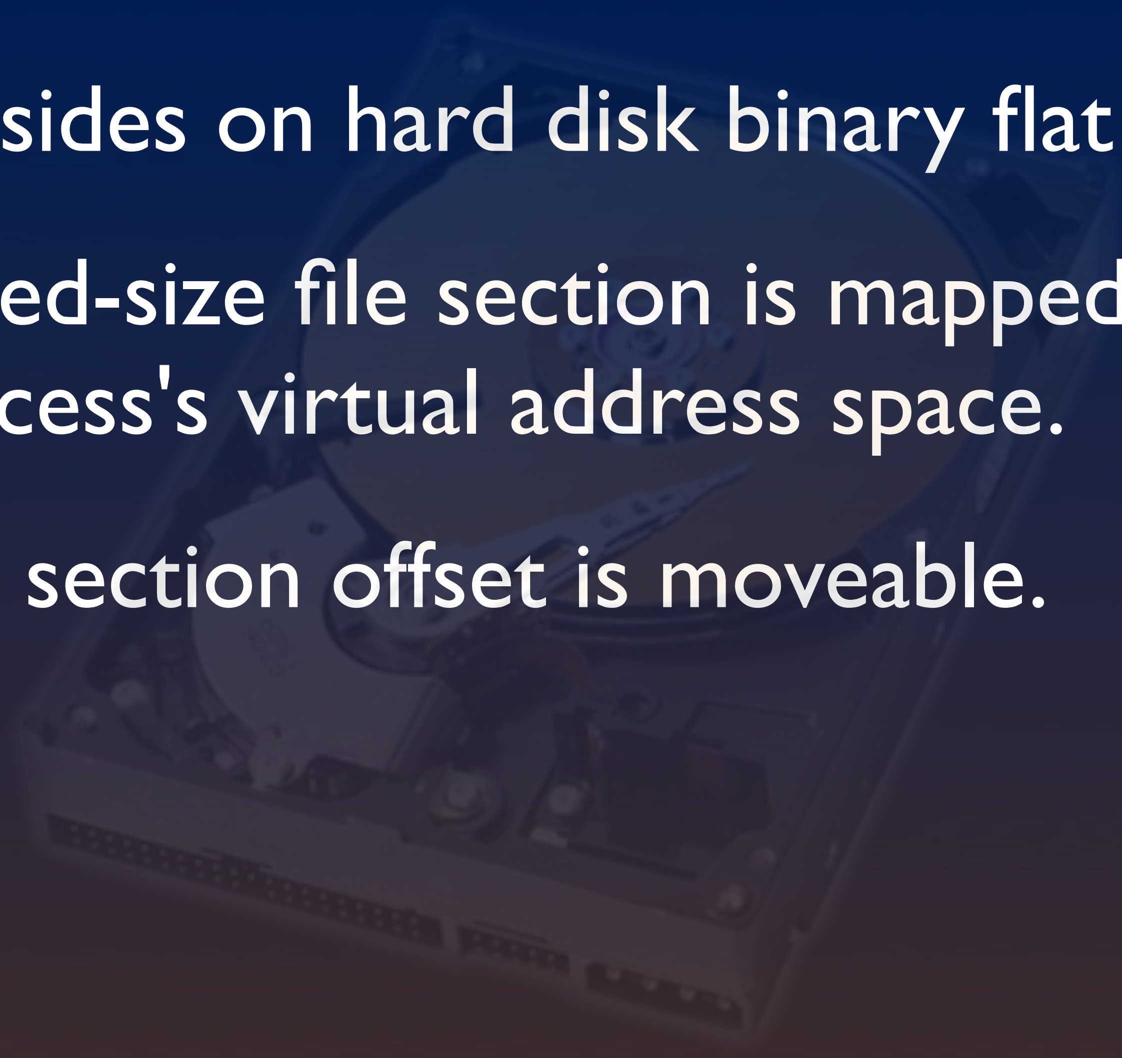
- Data resides on hard disk binary flat files.

Flat Files and Paging

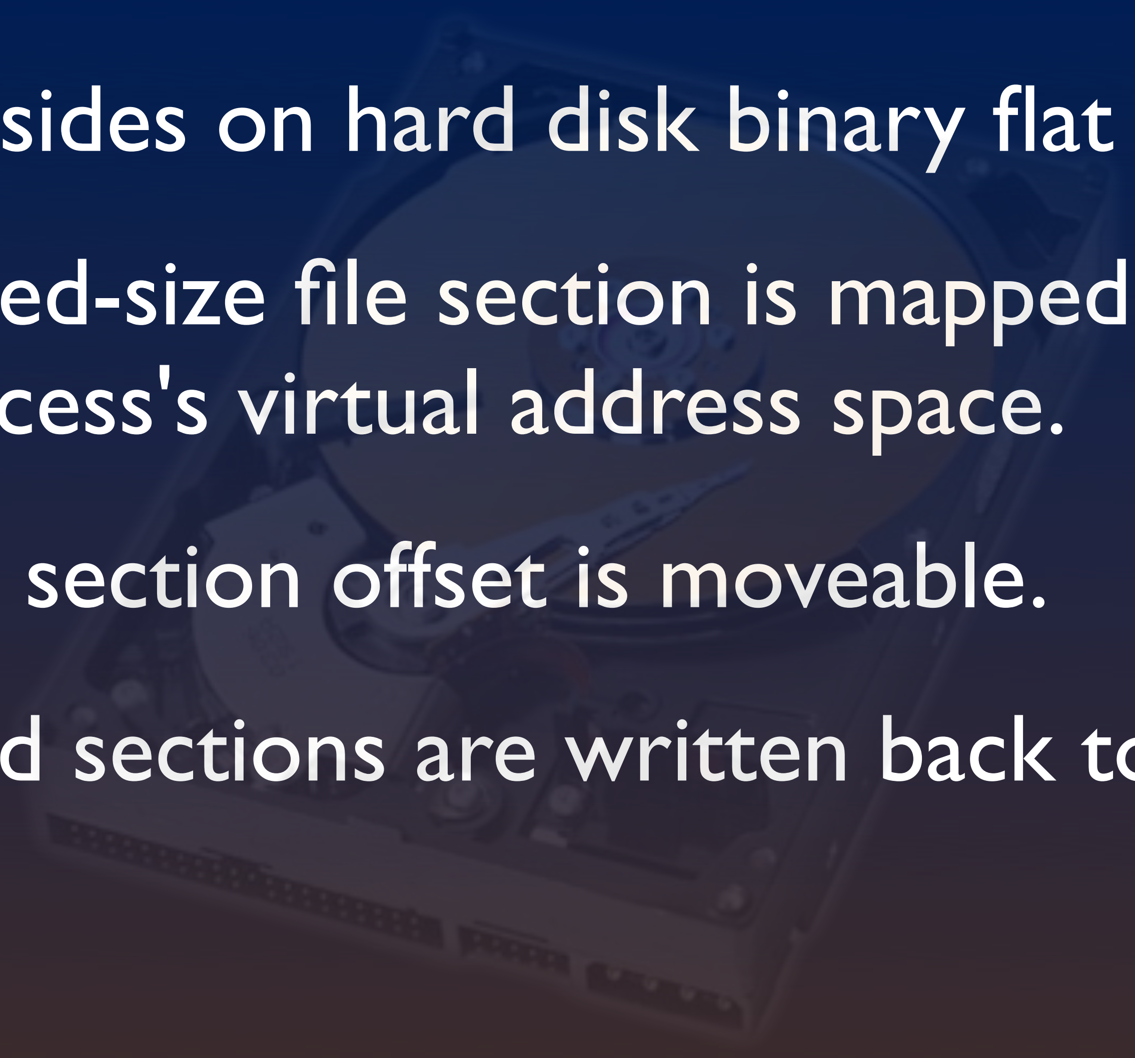
- Data resides on hard disk binary flat files.
- One fixed-size file section is mapped into the process's virtual address space.

Flat Files and Paging

- Data resides on hard disk binary flat files.
- One fixed-size file section is mapped into the process's virtual address space.
- The file section offset is moveable.

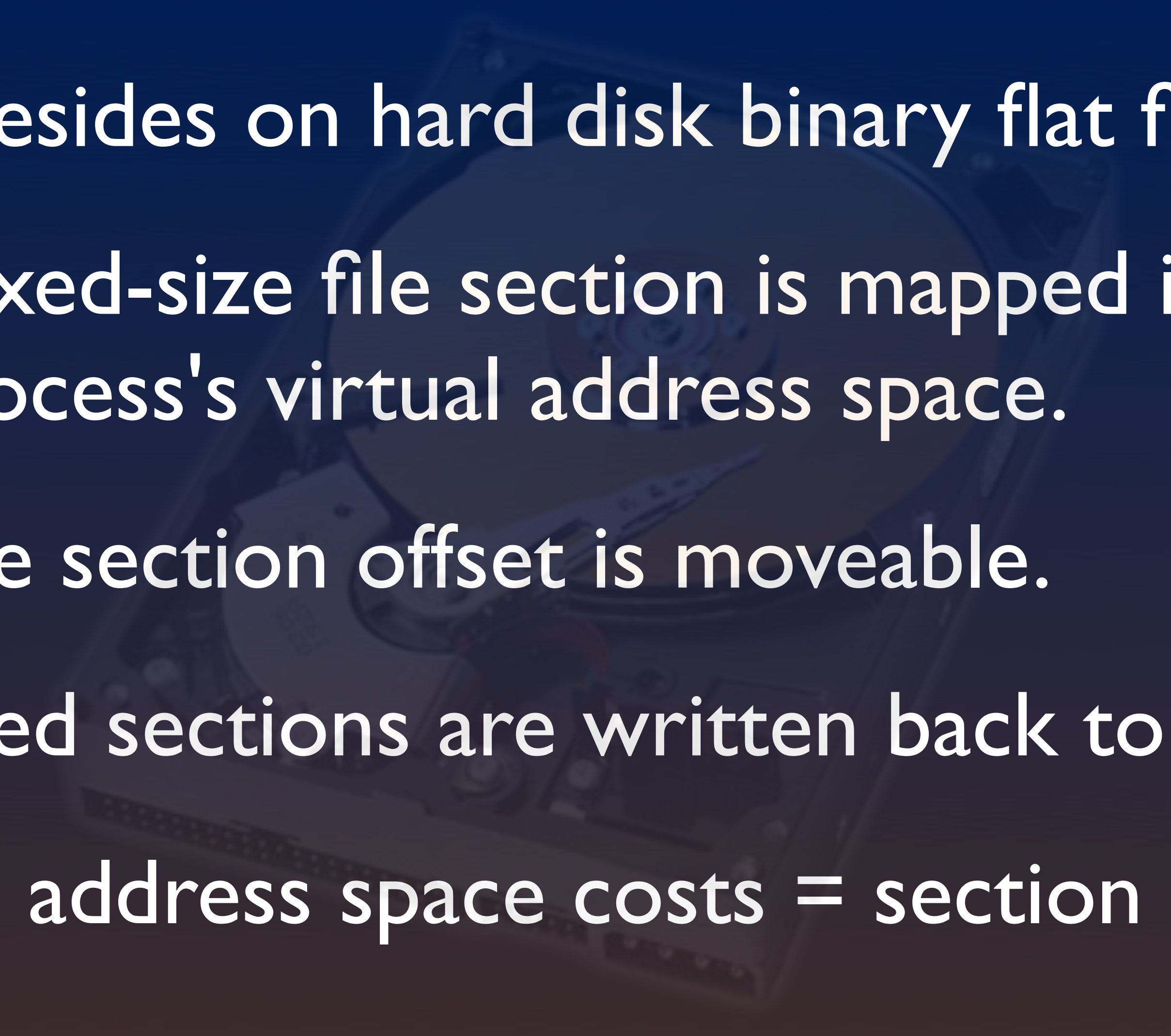


Flat Files and Paging



- Data resides on hard disk binary flat files.
- One fixed-size file section is mapped into the process's virtual address space.
- The file section offset is moveable.
- Modified sections are written back to disk.

Flat Files and Paging



- Data resides on hard disk binary flat files.
- One fixed-size file section is mapped into the process's virtual address space.
- The file section offset is moveable.
- Modified sections are written back to disk.
- Virtual address space costs = section size.

Virtual atomic R objects

Virtual atomic R objects

Creating vectors, matrices, arrays and factors.

```
> vec <- ff(vmode=”double”,length=100000000)
> mat <- ff(vmode=”double”,dim=c(5000,6000))
> arr <- ff(vmode=”integer”,dim=c(10,200,300))
> fac <- ff(vmode="integer",levels=c('A','B'),length=10e6))
```

Virtual atomic R objects

Creating vectors, matrices, arrays and factors.

```
> vec <- ff(vmode=”double”,length=100000000)
> mat <- ff(vmode=”double”,dim=c(5000,6000))
> arr <- ff(vmode=”integer”,dim=c(10,200,300))
> fac <- ff(vmode="integer",levels=c('A','B'),length=10e6))
```

Standard subsetting in R

```
> vec[1:1000] <- rnorm(1000)
> sum(mat[c(1,3,4),])
> arr[5:1,100,150]
> fac[] <- 'B'
```

Virtual atomic R objects

Creating vectors, matrices, arrays and factors.

```
> vec <- ff(vmode=”double”,length=100000000)
> mat <- ff(vmode=”double”,dim=c(5000,6000))
> arr <- ff(vmode=”integer”,dim=c(10,200,300))
> fac <- ff(vmode="integer",levels=c('A','B'),length=10e6))
```

Standard subsetting in R

```
> vec[1:1000] <- rnorm(1000)
> sum(mat[c(1,3,4),])
> arr[5:1,100,150]
> fac[] <- 'B'
```

Batch processing

```
> s <- 0
> ffvecapply( s <<- s + sum(vec[i1:i2]), X=vec )
> mymean <- s/length(vec)
```

data types and packing

data types and packing

vmode	size	R mode	NA handling	range
boolean	1 bit	logical		TRUE,FALSE
logical	2 bit	logical	NA	TRUE,FALSE
quad	2 bit	integer		0:3
nibble	4 bit	integer		0:15
byte	8 bit	integer	NA	-127:+127
ubyte	8 bit	integer		0:255
short	16 bit	integer	NA	-32767:+32767
ushort	16 bit	integer		0:65535
integer	32 bit	integer	NA	-(2^31-1):+(2^31-1)
single	32 bit	double	NA	C float
double	64 bit	double	NA	C double
raw	8 bit	raw		0:255

Hybrid Indexing

Hybrid Indexing

index expressions are packed (if possible) before evaluated. Saves space!

```
> vec[1:(length(vec)/2)] <- 1
```

Hybrid Indexing

index expressions are packed (if possible) before evaluated. Saves space!

```
> vec[1:(length(vec)/2)] <- 1
```

otherwise...

```
> 1:(length(vec)/2)
cannot allocate vector of size 2.0 Gb
```

Hybrid Indexing

index expressions are packed (if possible) before evaluated. Saves space!

```
> vec[1:(length(vec)/2)] <- 1
```

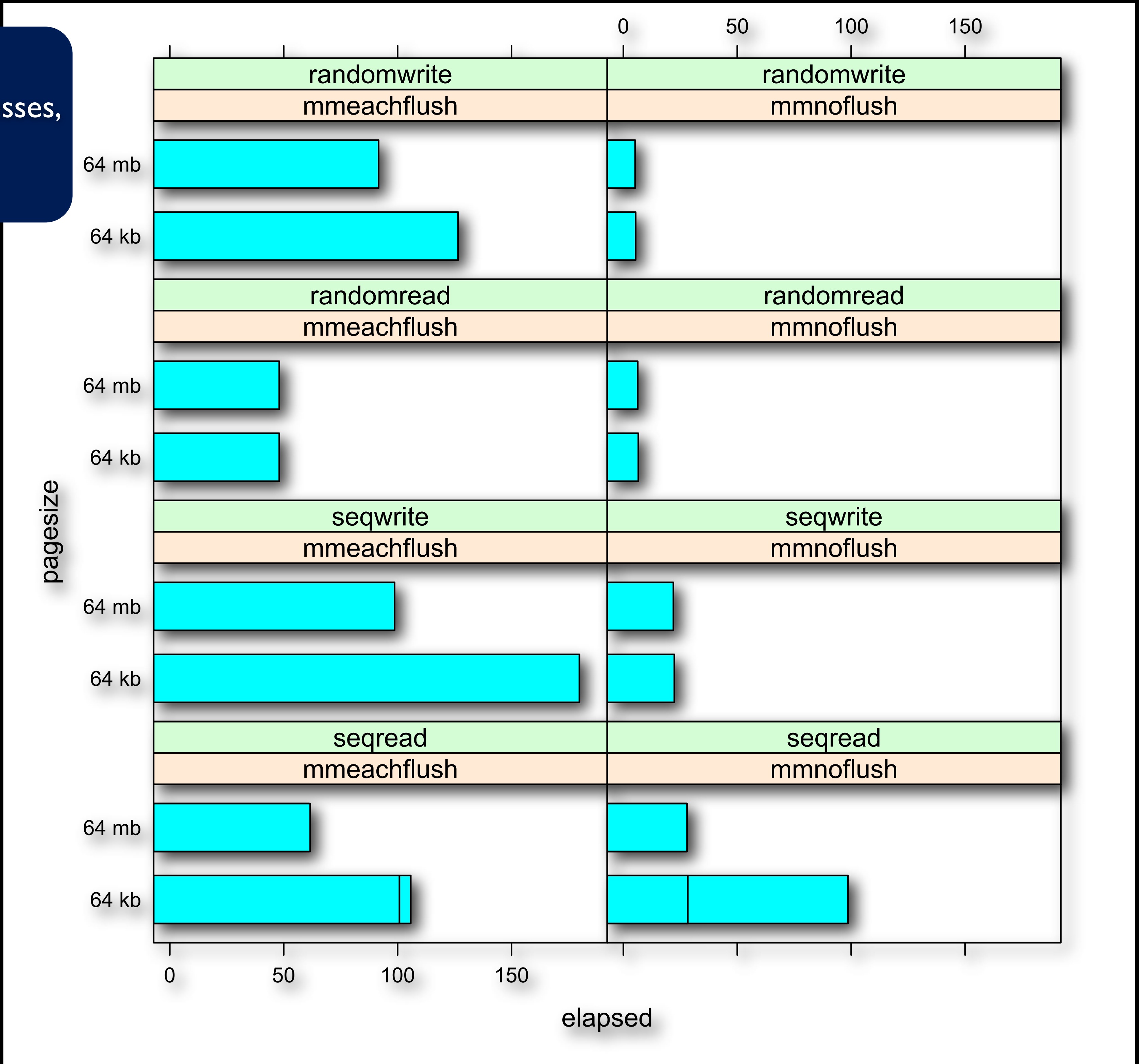
from	to	by
1	length(vec)/2	1

otherwise...

```
> 1:(length(vec)/2)
cannot allocate vector of size 2.0 Gb
```

Page size and system cache

2 GB double ff vector,
10,000,000 random accesses,
Intel Mac OS X 10.5,
4 GB, 2.5 GhZ



Performance Enhancements

- Presorting indices in ascending order to minimize disk head movements.
- Fast creation of flat files.
- Using system cache to prevent Disk I/O.
- Increase page size to reduce pagings.
- Exploit parallelism; Flat files are shareable among multiple R processes.

Possible improvements

- Increase index resolution to 52 bits in R.
- Support for mixed-type data frames.
- On-demand presorting indices.
- Automatic adjustments of system cache usage and page size.
- Paging Garbage Collector?

Conclusion

- Memory-mapping in contrast to stream-based Disk I/O has advantage of exploiting system cache and - at the same time - allow to share pages among multiple processes.
- While system cache enabled will also consume physical memory it still does not consume more virtual address space.

Availability of the 'ff' package

- Version 2.0.0 on CRAN (since Monday)
  GPL-2, C++ Library ISCL (BSD style)
- Web resources:
  http://134.76.173.220/ff *
  http://www.truecluster.com/ff.htm

* contains version 1(64 bit internal indexing), slides, datasets