APL Optimization Techniques For Commercial Applications

Eugene Ying

2013

Topics

1. Table Look Up
2. Arithmetic Optimization
3. Scalar Extension

Assumptions

- Dyalog APL Version 13.2.17248.0 64 bit Classic
- Using defaults that come with V13.2
- Running on IBM power7 processor (3.1Ghz)
- In AIX 6.1

Topic #1
Table Look Up

Vector Membership Function

Subject Argument s   Principal Argument P

18 65 87 …	∊	20 37 65 81 98 …

0 1 0 …

Tables in Commercial Data Processing are not always numeric

```
Ticker Symbols
Customer Names
User IDs
Machine Names
Model IDs
Product Codes
...
```

Commercial Applications

Your Customers

Exxon Mobil
Chrysler Group
Apple
RadioShack

Subject Argument

Matrix
$\in$

Fortune 500 List

Wal-Mart Stores
Exxon Mobil
Chevron
Phillips 66
Berkshire Hathaway
Apple
…

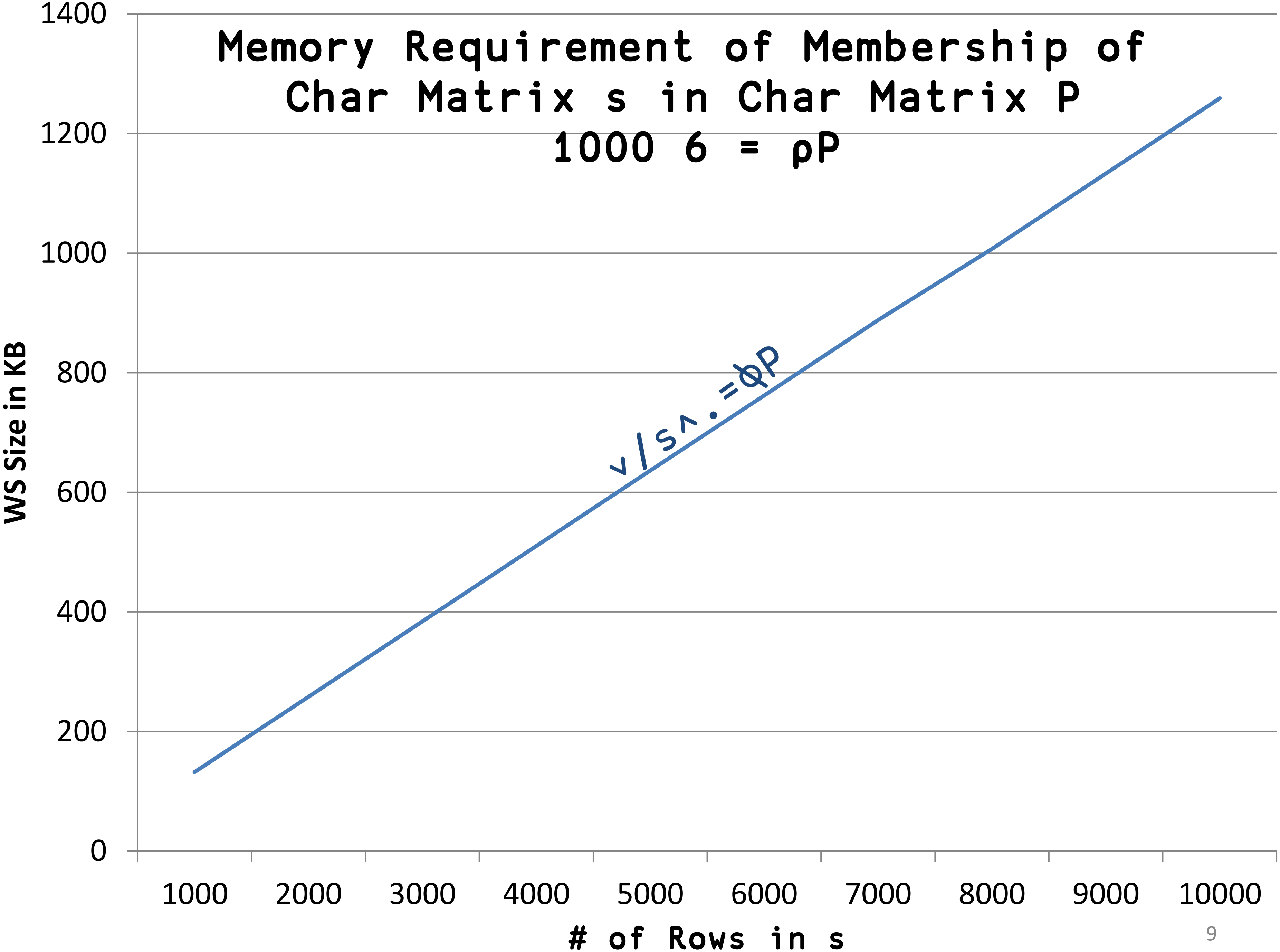
Principal Argument

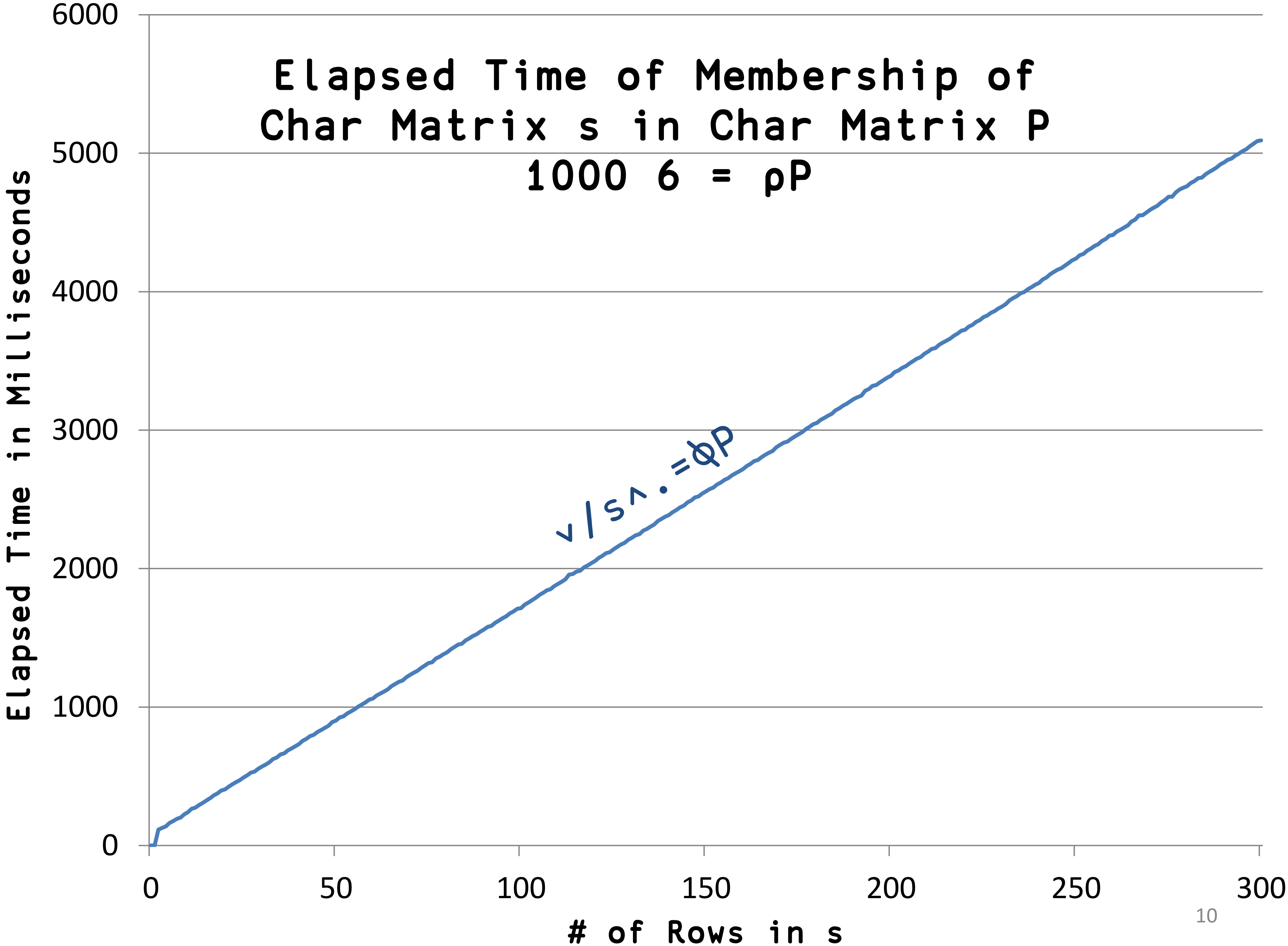
Answer is 1 0 1 0

Matrix Membership Function

Inner Product

```
∨/subject∧.=⍉PRINCIPAL
∨/s∧.=⍉P
```


Memory Requirement of Membership of
Char Matrix s in Char Matrix P
1000 6 = ⍴P
∨/s∧.=⍉P
WS Size in KB
0
200
400
600
800
1000
1200
1400
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
# of Rows in s


Elapsed Time of Membership of
Char Matrix s in Char Matrix P
1000 6 = ρP
∨/s∧.=⍉P
Elapsed Time in Milliseconds
0
1000
2000
3000
4000
5000
6000
0
50
100
150
200
250
300
# of Rows in s

Temporary Memory Requirement

`∨/s∧.=⍉P` requires a matrix transpose that takes up temporary WS and processing time, followed by an inner product that creates a sparse matrix that takes up more temporary WS.

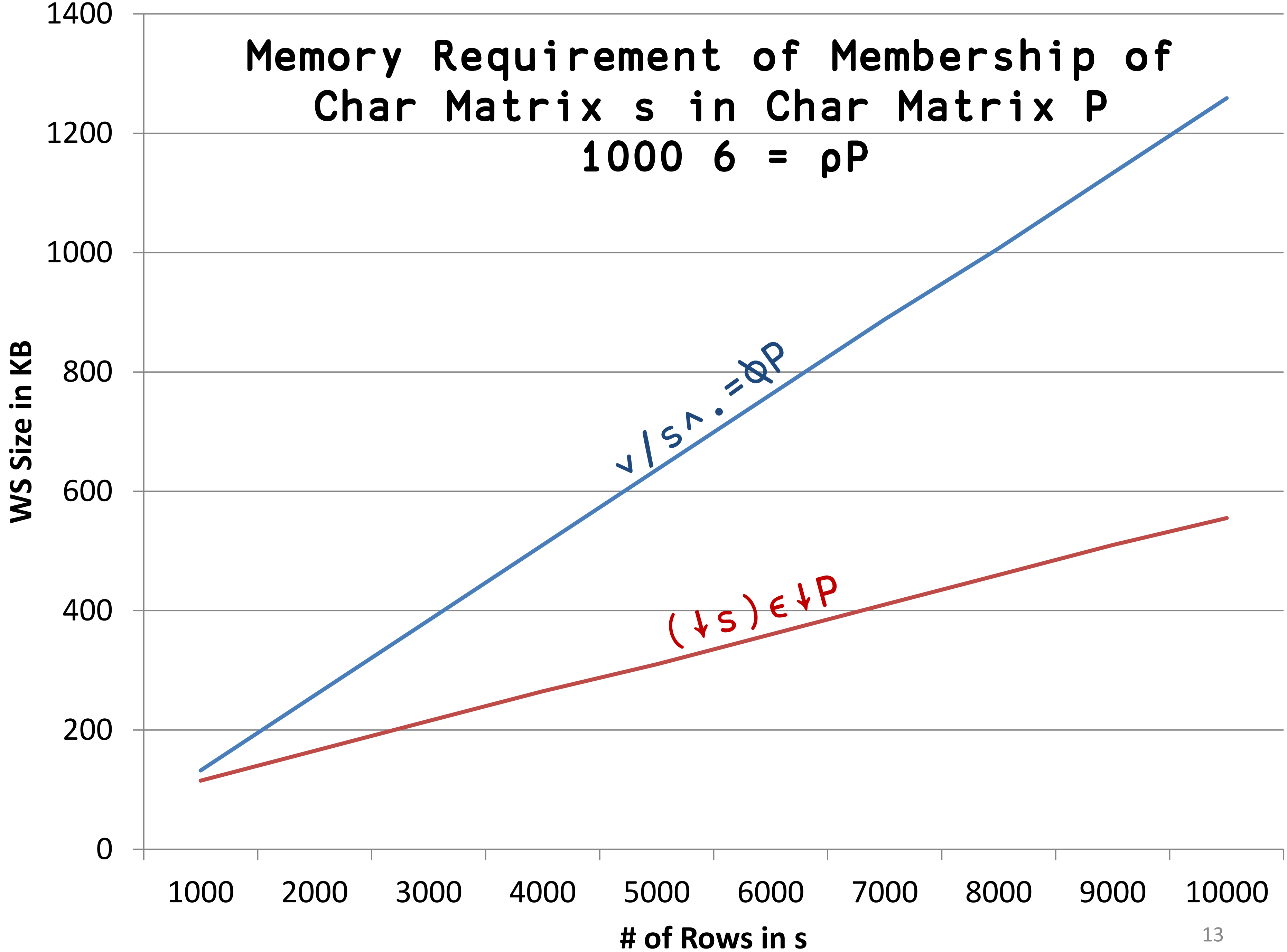
Matrix Membership Function

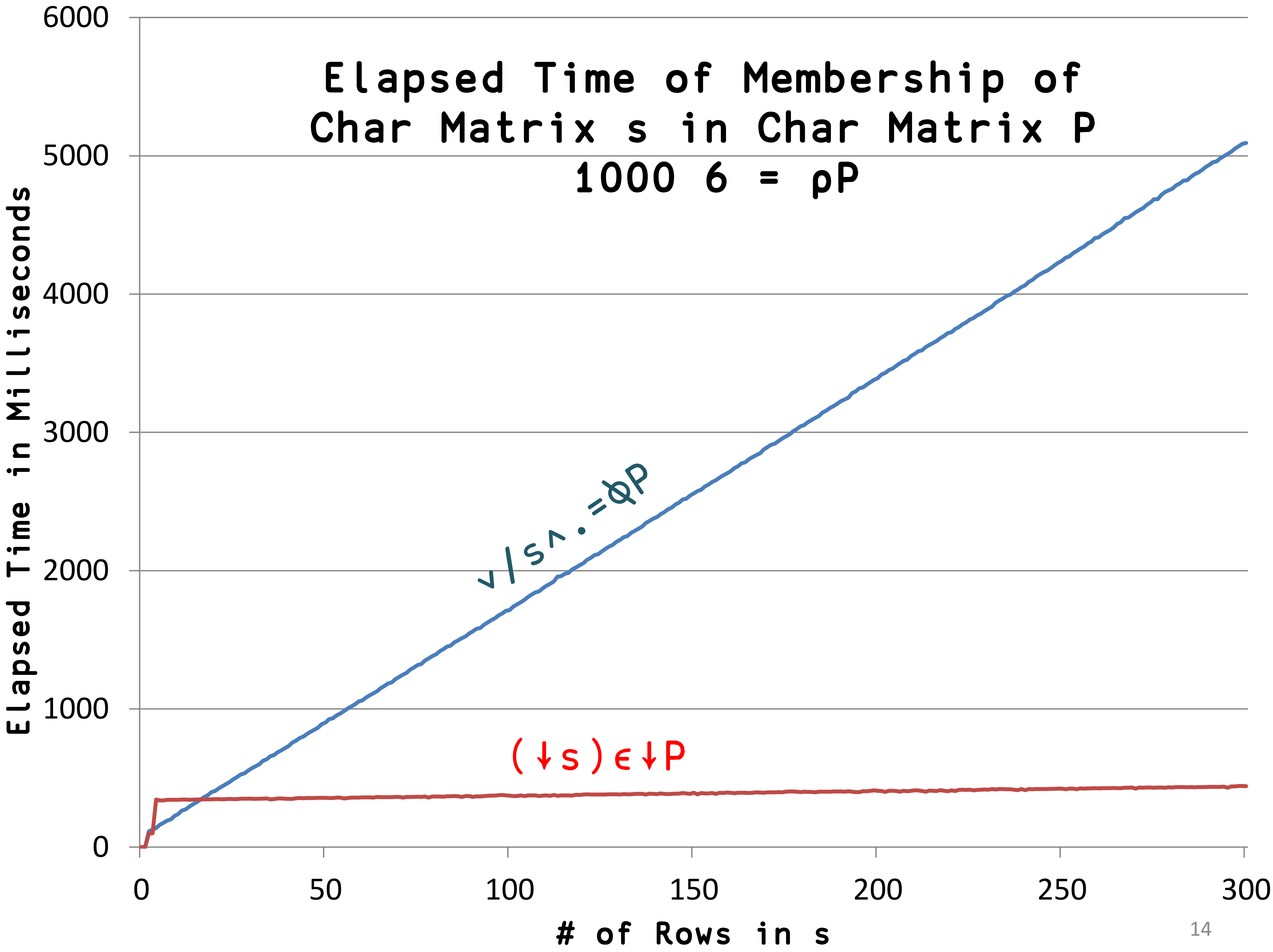
using inner product

```
∨/subject∧.=⍉PRINCIPAL
```

using nested vector

```
(↓subject)∊↓PRINCIPAL
```


Memory Requirement of Membership of
Char Matrix s in Char Matrix P
1000 6 = ⍴P
∨/s∧.=⍉P
(↓s)∊↓P
WS Size in KB
0
200
400
600
800
1000
1200
1400
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
# of Rows in s


Elapsed Time of Membership of
Char Matrix s in Char Matrix P
1000 6 = ⍴P
∨/s∧.=⍉P
(↓s)∊↓P
Elapsed Time in Milliseconds
0
1000
2000
3000
4000
5000
6000
0
50
100
150
200
250
300
# of Rows in s

Relationship Between Membership ∊ and Index Of ⍳

s∊P

can be emulated by

(⍴P)≥P⍳s

Similarly

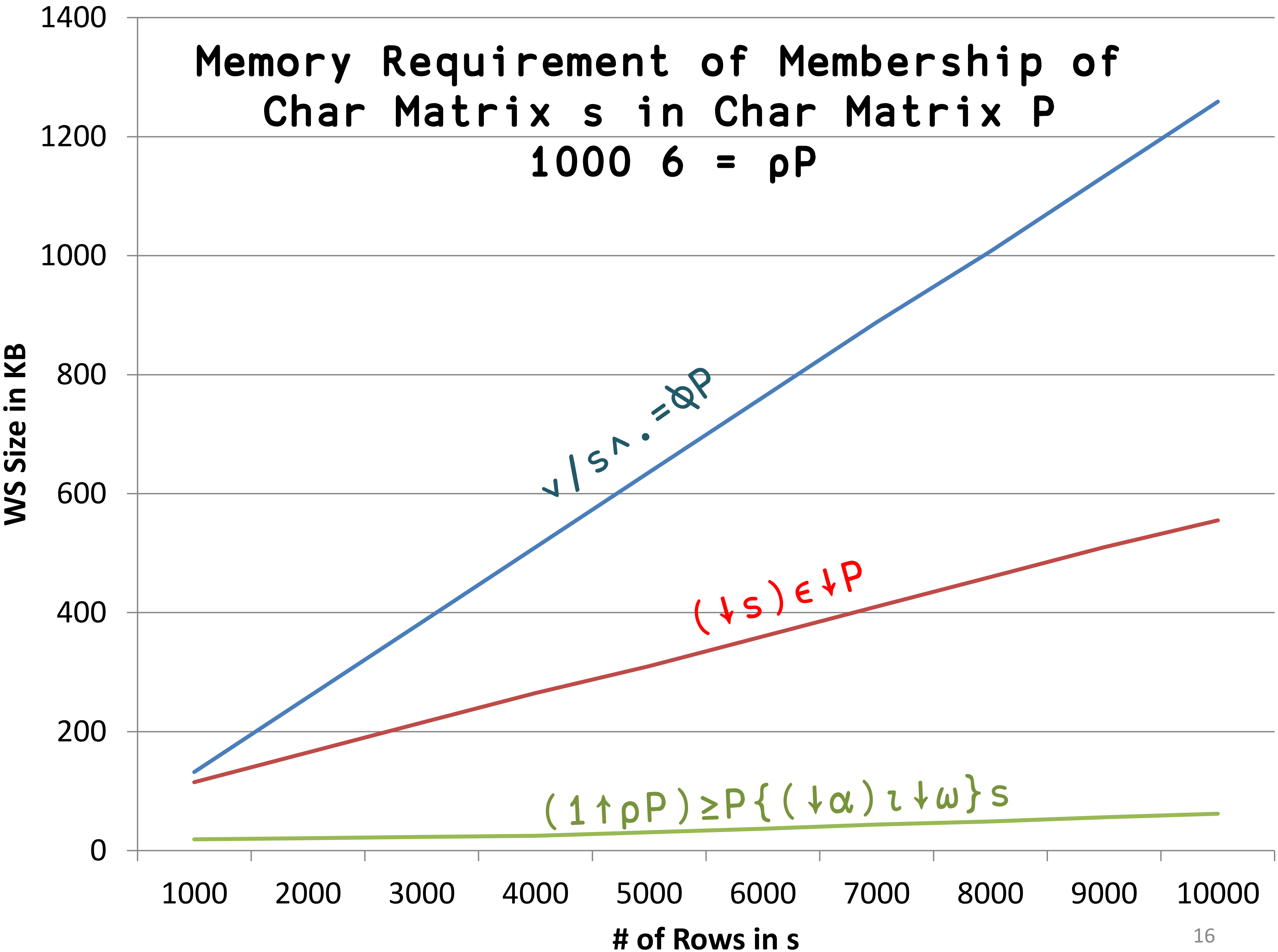
(↓s)∊↓P

can be emulated by

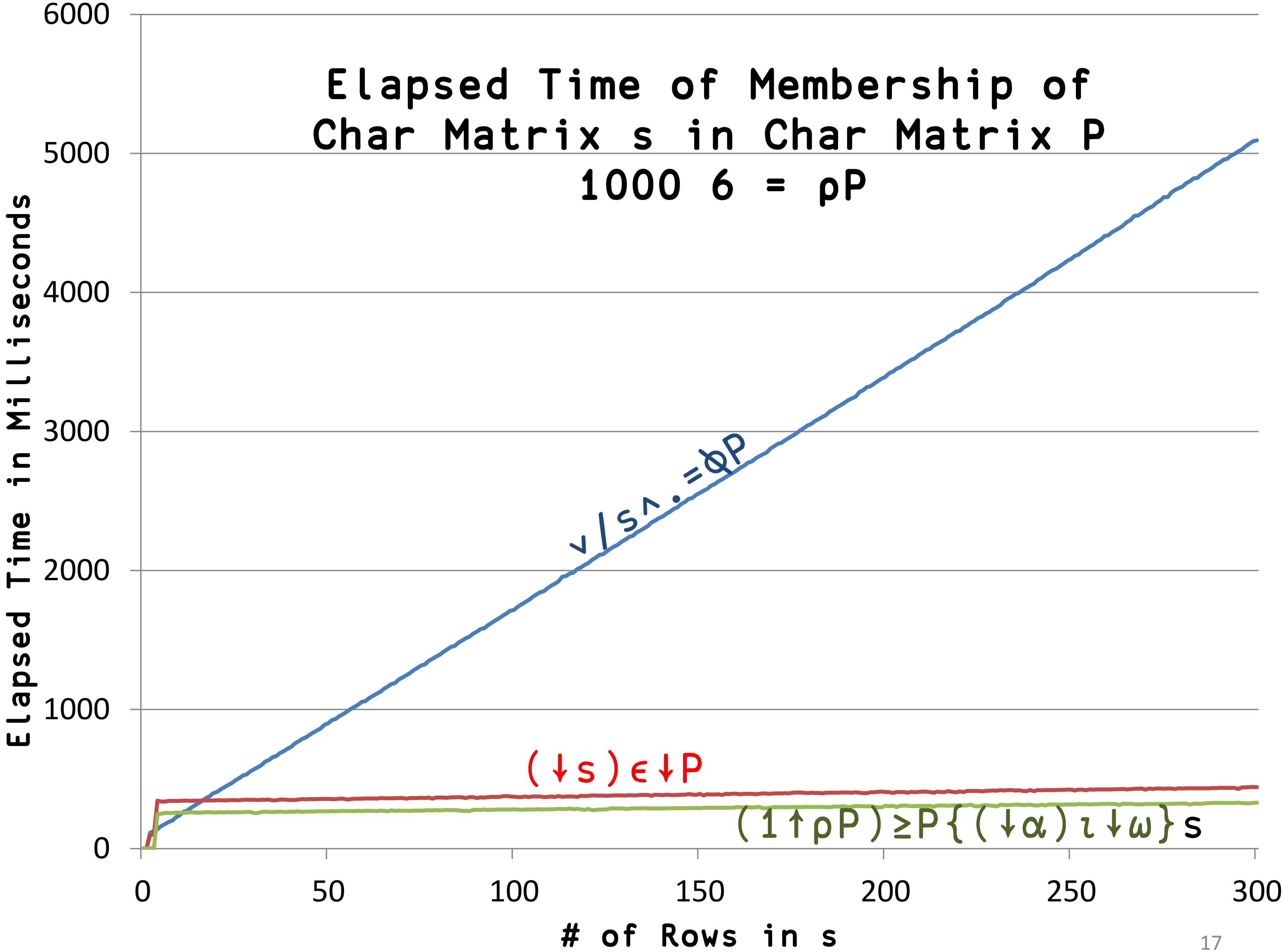
(1↑⍴P)≥(↓P)⍳↓s

or

(1↑⍴P)≥P{(↓⍺)⍳↓⍵}s



Memory Requirement of Membership of
Char Matrix s in Char Matrix P
1000 6 = ⍴P
∨/s∧.=⍉P
(↓s)∊↓P
(1↑⍴P)≥P{(↓⍺)⍳↓⍵}s
WS Size in KB
0
200
400
600
800
1000
1200
1400
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
# of Rows in s



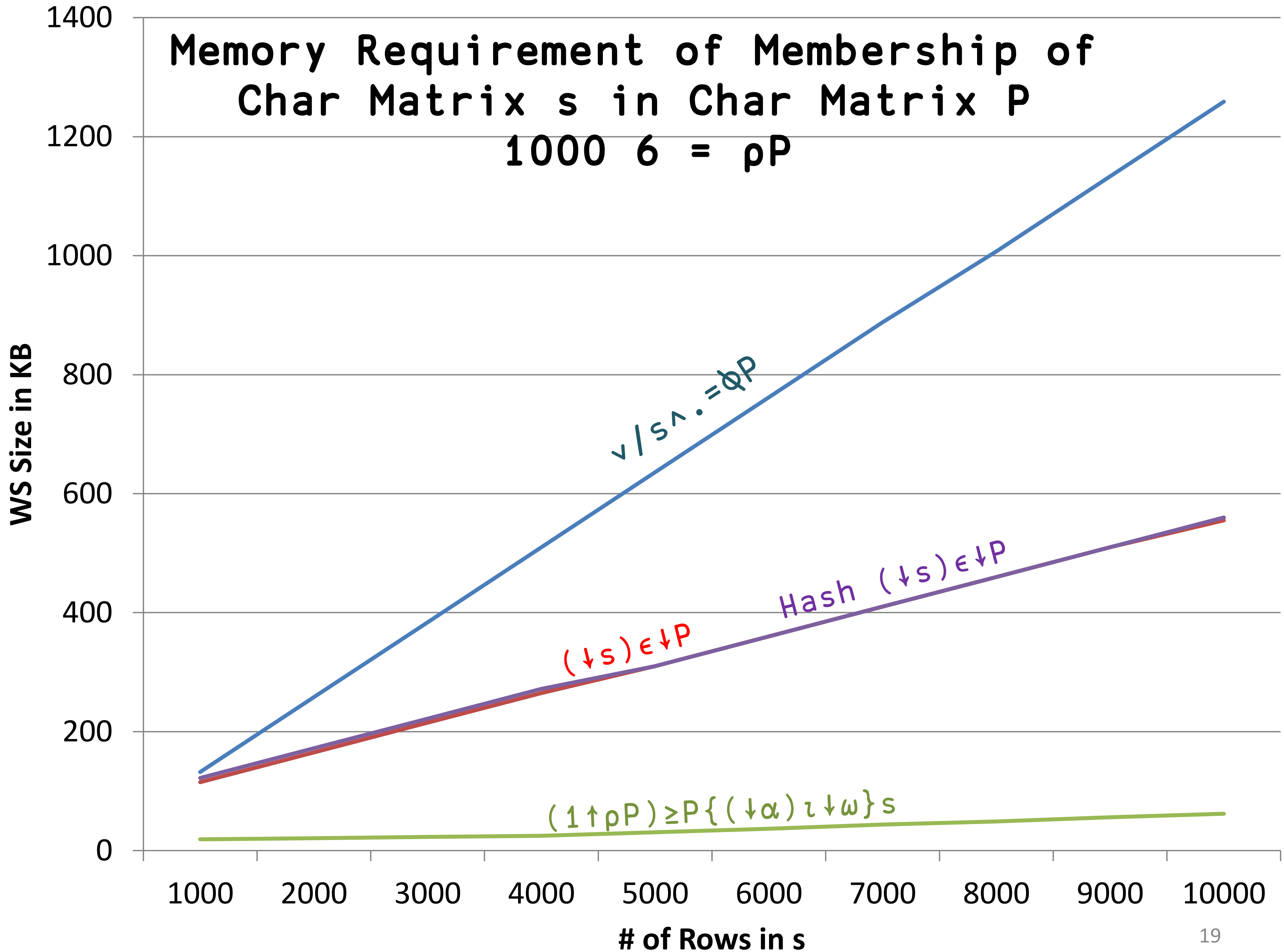
Elapsed Time of Membership of
Char Matrix s in Char Matrix P
1000 6 = ⍴P
∨/s∧.=⍉P
(↓s)∊↓P
(1↑⍴P)≥P{(↓⍺)⍳↓⍵}s
Elapsed Time in Milliseconds
6000
5000
4000
3000
2000
1000
0
0
50
100
150
200
250
300
# of Rows in s

If PRINCIPAL argument P is inside a loop and remains constant,

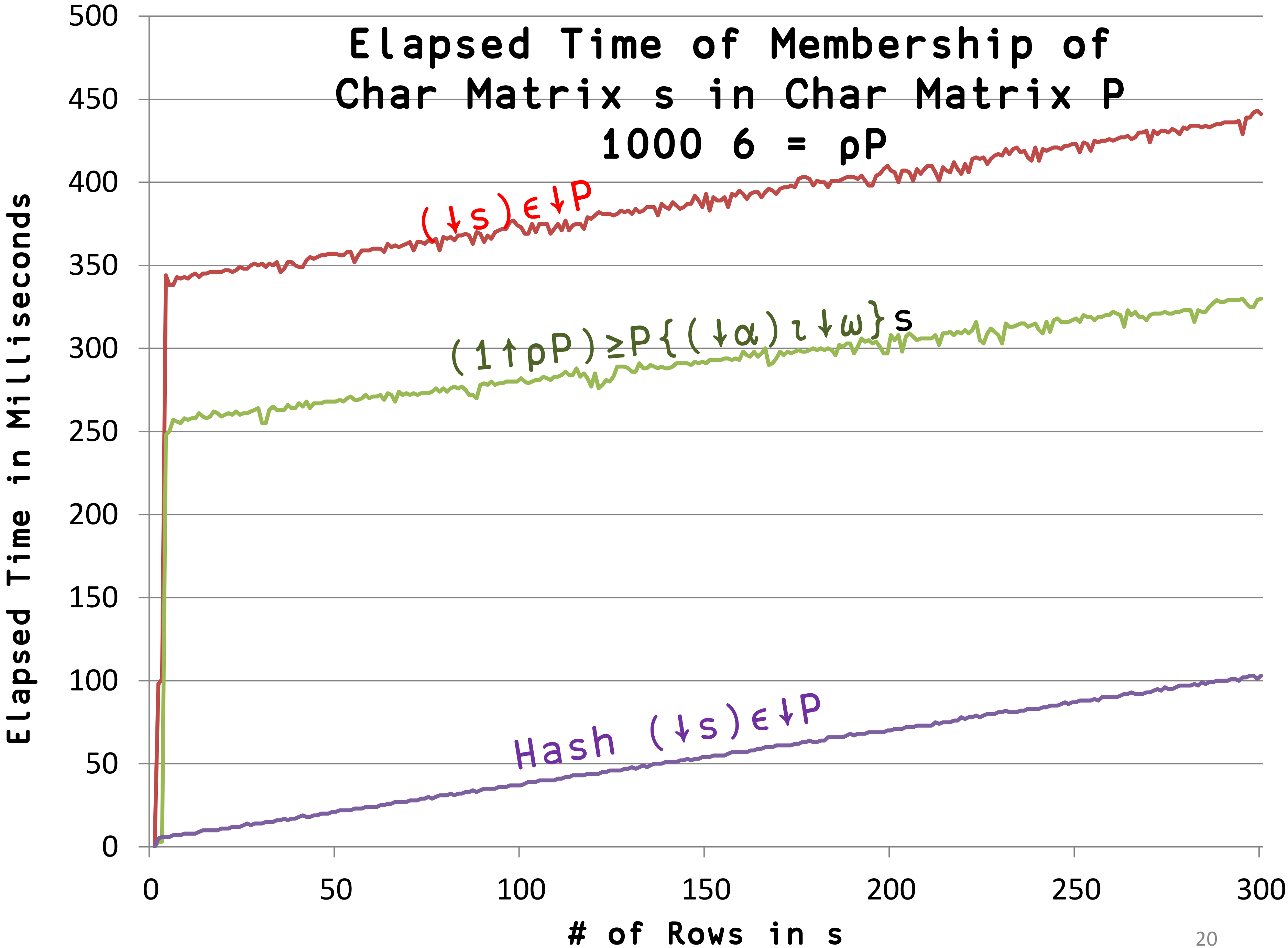
```
:For I :in ⍳LOOP
    B←(↓s)∊↓P
:Endfor
```

then P can be taken outside the loop for hash table.

```
Z←↓P
HASH←∊∘Z
:For I :in ⍳LOOP
    B←HASH ↓s
:Endfor
```



Memory Requirement of Membership of Char Matrix s in Char Matrix P
1000 6 = ⍴P
WS Size in KB
# of Rows in s
1400
1200
1000
800
600
400
200
0
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
∨/s∧.=⍉P
Hash (↓s)∊↓P
(↓s)∊↓P
(1↑⍴P)≥P{(↓⍺)⍳↓⍵}s

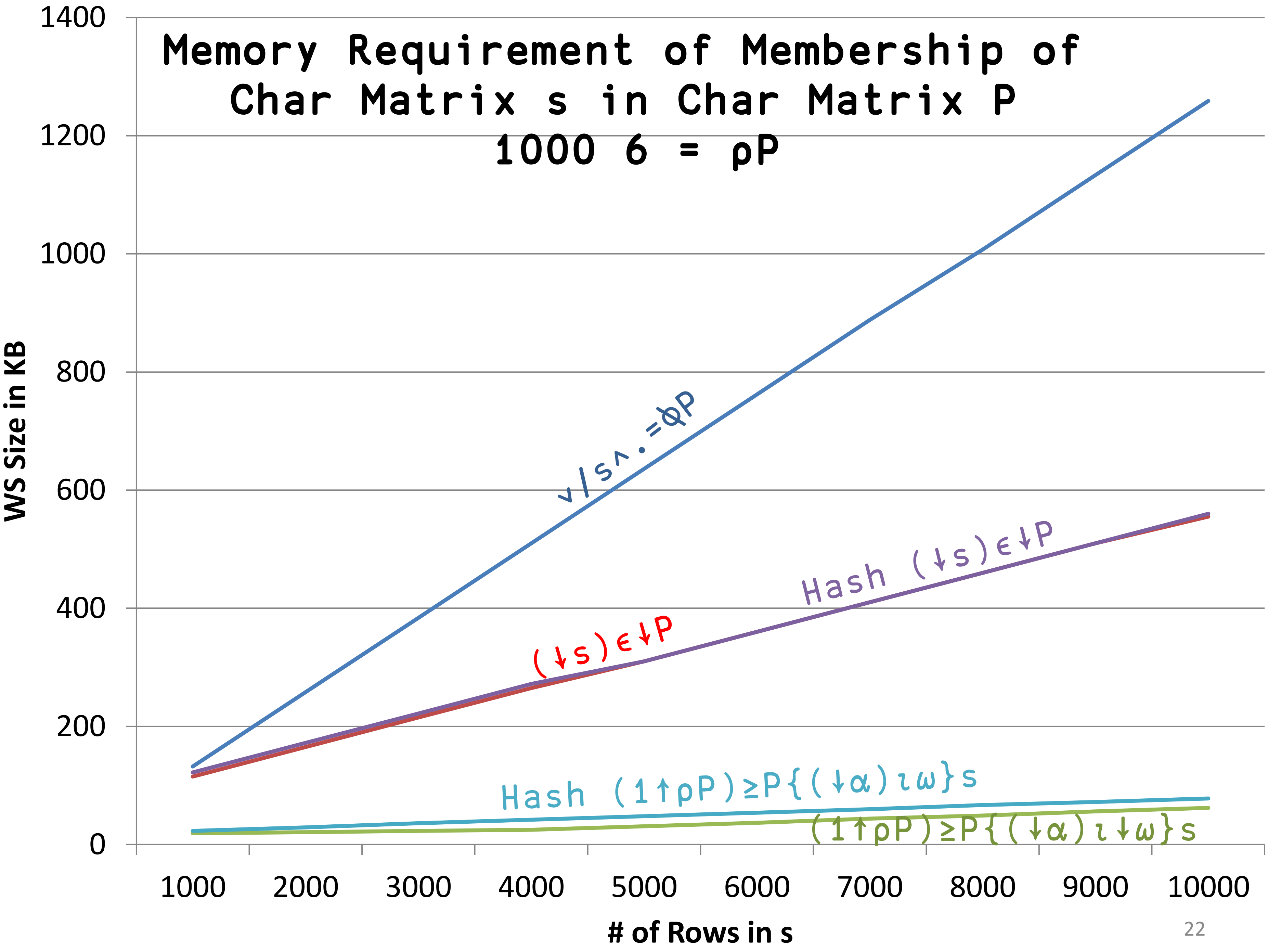

Elapsed Time of Membership of
Char Matrix s in Char Matrix P
1000 6 = ρP
(↓s)∊↓P
(1↑ρP)≥P{(↓α)⍳↓ω}s
Hash (↓s)∊↓P
Elapsed Time in Milliseconds
# of Rows in s
0
50
100
150
200
250
300
350
400
450
500
0
50
100
150
200
250
300

If PRINCIPAL argument P is inside a loop and remains constant,

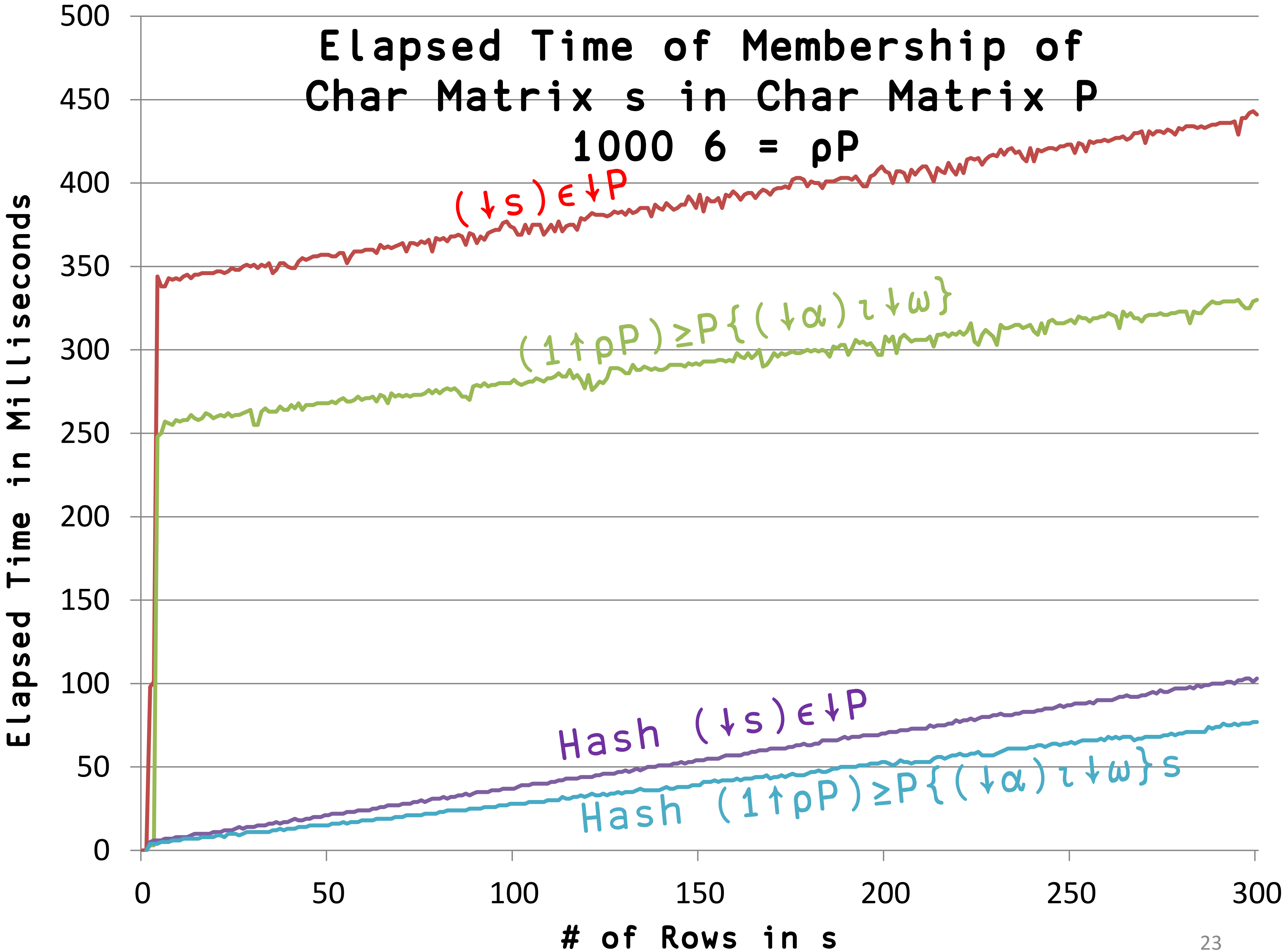
```
:For I :in ⍳LOOP
    B←(1↑⍴P)≥P{(↓⍺)⍳↓⍵}s
:Endfor
```

then P can be taken outside the loop for a hash table.

```
HASH←P∘{(↓⍺)⍳↓⍵}
:For I :in ⍳LOOP
    B←(1↑⍴P)≥HASH s
:Endfor
```



Memory Requirement of Membership of Char Matrix s in Char Matrix P
1000 6 = ρP
∨/s∧.=⍉P
Hash (↓s)∊↓P
(↓s)∊↓P
Hash (1↑ρP)≥P{(↓α)⍳ω}s
(1↑ρP)≥P{(↓α)⍳↓ω}s
WS Size in KB
0
200
400
600
800
1000
1200
1400
1000
2000
3000
4000
5000
6000
7000
8000
9000
10000
# of Rows in s



Elapsed Time of Membership of
Char Matrix s in Char Matrix P
1000 6 = ⍴P
(↓s)∊↓P
(1↑⍴P)≥P{(↓⍺)⍳↓⍵}
Hash (↓s)∊↓P
Hash (1↑⍴P)≥P{(↓⍺)⍳↓⍵}s
Elapsed Time in Milliseconds
# of Rows in s
0
50
100
150
200
250
300
350
400
450
500

Precaution

The P{(↓α)⍳↓ω}s Idiom
is optimized for character matrix and
integer matrix arguments.

This idiom is many times less efficient if
your matrix arguments have floating point
numbers.

In such cases you may wish to explore
algorithms more efficient than this Idiom
to substitute for the expression ∨/s∧.=⍉P

Recommendation

In your APL function libraries, whenever you see

```
∨/s∧.=⍉P
or
(↓s)∊↓P
```

if the character matrices are large, replace them by

```
(1↑⍴P)≥P{(↓⍺)⍳↓⍵}s
```

and you will see much faster execution speed with less workspace requirement.

Recommendation

Furthermore, if the Matrix Epsilon is inside a loop, and if the PRINCIPAL character matrix P is constant, then replace

```
:For I :In ⍳LOOP
    B←(1↑⍴P)≥P{(↓⍺)⍳↓⍵}s
:Endfor
```

by

```
HASH←P∘{(↓⍺)⍳↓⍵}
:For I :In ⍳LOOP
    B←(1↑⍴P)≥HASH s
:Endfor
```

Another Situation

Up to now, we are concerned with `∨/s∧.=⍉P`
but if you see the expression `∨⌿A∧.=⍉B`
you can do this optimization:

```
∨⌿A∧.=⍉B
    ↓
∨/B∧.=⍉A
    ↓
(↓B)∊↓A
    ↓
(1↑⍴A)≥(↓A)⍳↓B
    ↓
(1↑⍴A)≥A{(↓⍺)⍳↓⍵}B
```

And see the previous page in case the hash table approach is applicable.

Topic # 2
Arithmetic Optimization

Divide vs Multiply

```
M←5000 10⍴0.1+⍳50000
```

```
:For I :In ⍳10000
    {}M÷100
:EndFor
```

```
⍝ 24492 ms
```

```
:For I :In ⍳10000
    {}M×0.01
:EndFor
```

```
⍝   901 ms
```

Readability of Constants

```
:For I :In ⍳10000
    {}M÷7
:EndFor
```

```
:For I :In ⍳10000
    {}M×0.1428571429
:EndFor
```

```
R7←÷7
:For I :In ⍳10000
    {}M×R7
:EndFor
```

Divide vs Reciprocal

In APL, the Reciprocal function is usually faster than the Divide function

```
:For I :In ⍳1000000
    {}1÷7
    {}÷7
:EndFor
```

```
⍝ 1234 ms
⍝ 1051 ms
```

```
Since 1÷Array is slower than ÷Array
should we change 0-Array to -Array?
```

```
M←10000 10⍴0.1+⍳100000
:For I :In ⍳1000000
    {}0-M
    {}-M
:EndFor
```

```
⍝   8546 ms
⍝ 110814 ms
```

An APL interpreter can sometimes make use of extra help from an APL programmer to make division and subtraction execute faster.

But should the APL programmer make the function arguments conformable to further help the APL interpreter?

e.g.   Should M×0.01
be changed to M×(⍴M)⍴0.01

```
M←5000 10⍴0.1+⍳50000
```

```
:For I :In ⍳10000
    {}M÷100
:EndFor
```

⍝ 24570 ms

```
:For I :In ⍳10000
    {}M×0.01
:EndFor
```

⍝ 653 ms

```
:For I :In ⍳10000
    {}M×(⍴M)⍴0.01
:EndFor
```

⍝ 1545 ms

Topic #3. Scalar Extension

```
M←5000 10⍴0.1+⍳50000 ⍝ Matrix M
S←30                 ⍝ Scalar S
```

```
:For I :In ⍳10000
    {}S×M×0.01
:EndFor
```
```
⍝ 1566 ms
```

```
:For I :In ⍳10000
    {}M×S×0.01
:EndFor
```
```
⍝  773 ms
```

Order of Execution for Scalars

```
Scalar1×Array×Scalar2
```

can be optimized as

```
Array×Scalar1×Scalar2
```

so that two scalar extensions are reduced to only one scalar extension.

```
Matrix ∊ Scalar
Matrix = Scalar
```

Should we use ’∊’ ?
Or should we use ‘=‘ ?

```
M←5000 10⍴0.1+⍳50000
```

```
:For I :In ⍳10000
    {}M∊1000.1          ⍝ 10207 ms
    {}M=1000.1          ⍝  4385 ms
:EndFor
```

Because of scalar extension,
M=1000.1 is faster than M∊1000.1

Encoding & Decoding Characters

In a ⎕IO=1 environment

```
      26⊥¯1+⎕A⍳'DYALOG'
46619358
```

The ¯1+ is performed 6 times because of scalar extension.

```
      ⎕A[1+(6⍴26)⊤4619358]
DYALOG
```

The 1+ is performed 6 times because of scalar extension.

Avoid Scalar Extension

```
      {⎕IO←0 ⋄ 26⊥⎕Aιω}'DYALOG'
46619358
```

Now there is no need for scalar extension of ¯1+

```
      {⎕IO←0 ⋄ ⎕A[(6⍴26)⊤ω]}46619358
DYALOG
```

Now there is no need for scalar extension of 1+

Advantage of Localizing System Variable in D-fn

```
SYNTAX ERROR
      {⎕IO←0 ◇ 26↑⎕A⍳⍵}=
     ∧
      ⎕IO
1
```

```
When an error occurs,
thanks to D-fn,
⎕IO bounces back to 1
```

Recommendation

Scalar extension has been implemented very efficiently in APL. Whenever there is a chance, let APL do the scalar extension for you. Do not reshape the scalar to help the APL interpreter.

If an expression has a combination of scalars and arrays, let APL work with scalars first, then do scalar extension later.

Scalar Array

When you enclose a numeric array to make it a scalar, this scalar array can take advantage of scalar extension to retrieve items from a nested array efficiently.

This is best illustrated by examples.

A 2 by 3 Nested Matrix Where Each Item Is A 3 by 3 Numeric Matrix

```
      DISPLAY A
.→------------------------------------------------.
↓ .→-------------.  .→-------------.  .→-------------. |
| ↓1111 1112 1113|  ↓1211 1212 1213|  ↓1311 1312 1313| |
| |1121 1122 1123|  |1221 1222 1223|  |1321 1322 1323| |
| |1131 1132 1133|  |1231 1232 1233|  |1331 1332 1333| |
| '~-------------'  '~-------------'  '~-------------' |
| .→-------------.  .→-------------.  .→-------------. |
| ↓2111 2112 2113|  ↓2211 2212 2213|  ↓2311 2312 2313| |
| |2121 2122 2123|  |2221 2222 2223|  |2321 2322 2323| |
| |2131 2132 2133|  |2231 2232 2233|  |2331 2332 2333| |
| '~-------------'  '~-------------'  '~-------------' |
'∊------------------------------------------------'
```

```
          A[((1 1)(2 2)) ((2 3)(2 2))]

.→-------------------------------------------------------.
↓ A[⊂((1 1)(2 2))]                                        |
| .→--------------.  .→--------------.  .→--------------. |
| ↓1111 1112 1113|  ↓1211 1212 1213|  ↓1311 1312 1313| |
| |1121 1122 1123|  |1221 1222 1223|  |1321 1322 1323| |
| |1131 1132 1133|  |1231 1232 1233|  |1331 1332 1333| |
| '~--------------'  '~--------------'  '~--------------' |
|                                                         |
|                                        A[⊂((2 3)(2 2))] |
| .→--------------.  .→--------------.  .→--------------. |
| ↓2111 2112 2113|  ↓2211 2212 2213|  ↓2311 2312 2313| |
| |2121 2122 2123|  |2221 2222 2223|  |2321 2322 2323| |
| |2131 2132 2133|  |2231 2232 2233|  |2331 2332 2333| |
| '~--------------'  '~--------------'  '~--------------' |
'∊--------------------------------------------------------'
```

To select the red items, you can use
A[((1 1)(2 2)) ((2 3)(2 2))]

```
                   (⊂2 2)⌷¨
.→--------------------------------------------------------.
↓        2 2⌷             2 2⌷             2 2⌷            |
|  .→--------------.  .→--------------.  .→--------------.  |
|  ↓1111 1112 1113|  ↓1211 1212 1213|  ↓1311 1312 1313|  |
|  |1121 1122 1123|  |1221 1222 1223|  |1321 1322 1323|  |
|  |1131 1132 1133|  |1231 1232 1233|  |1331 1332 1333|  |
|  '~--------------'  '~--------------'  '~--------------'  |
|                                                           |
|        2 2⌷             2 2⌷             2 2⌷            |
|  .→--------------.  .→--------------.  .→--------------.  |
|  ↓2111 2112 2113|  ↓2211 2212 2213|  ↓2311 2312 2313|  |
|  |2121 2122 2123|  |2221 2222 2223|  |2321 2322 2323|  |
|  |2131 2132 2133|  |2231 2232 2233|  |2331 2332 2333|  |
|  '~--------------'  '~--------------'  '~--------------'  |
'∊---------------------------------------------------------'
```

```
To select the red items, you can use
A[2 3⍴((1 1)(2 2)) ((1 2)(2 2)) ((1 3)(2 2)) ((2 1)(2 2))
((2 2)(2 2)) ((2 3)(2 2))]    ⍝ 8.1 sec 1,000,000 loops

or (⊂2 2)⌷¨A   ⍝ Squad Indexing 6.7 sec 1,000,000 loops
```

```
                          (⊂2 2)↑¨
.→-----------------------------------------------------.
↓         2 2↑              2 2↑              2 2↑     |
|  .→--------------.  .→--------------.  .→--------------.  |
|  ↓1111 1112 1113|  ↓1211 1212 1213|  ↓1311 1312 1313|  |
|  |1121 1122 1123|  |1221 1222 1223|  |1321 1322 1323|  |
|  |1131 1132 1133|  |1231 1232 1233|  |1331 1332 1333|  |
|  '~--------------'  '~--------------'  '~--------------'  |
|                                                       |
|         2 2↑              2 2↑              2 2↑     |
|  .→--------------.  .→--------------.  .→--------------.  |
|  ↓2111 2112 2113|  ↓2211 2212 2213|  ↓2311 2312 2313|  |
|  |2121 2122 2123|  |2221 2222 2223|  |2321 2322 2323|  |
|  |2131 2132 2133|  |2231 2232 2233|  |2331 2332 2333|  |
|  '~--------------'  '~--------------'  '~--------------'  |
'∊------------------------------------------------------'

To select the red items, you can use
2↑[2]¨2↑[1]¨A    ⍝ 13.3 sec (looping 1,000 times)
or
(⊂2 2)↑¨A         ⍝  4.9 sec (looping 1,000 times)
```

```
                        (⊂(2 3)(,2))⌷¨A
.→----------------------------------------------------.
↓   (2 3)(,2)⌷         (2 3)(,2)⌷          (2 3)(,2)⌷    |
| .→--------------. .→--------------. .→--------------. |
| ↓1111 1112 1113| ↓1211 1212 1213| ↓1311 1312 1313| |
| |1121 1122 1123| |1221 1222 1223| |1321 1322 1323| |
| |1131 1132 1133| |1231 1232 1233| |1331 1332 1333| |
| '~--------------' '~--------------' '~--------------' |
|   (2 3)(,2)⌷         (2 3)(,2)⌷         (2 3)(,2)⌷     |
| .→--------------. .→--------------. .→--------------. |
| ↓2111 2112 2113| ↓2211 2212 2213| ↓2311 2312 2313| |
| |2121 2122 2123| |2221 2222 2223| |2321 2322 2323| |
| |2131 2132 2133| |2231 2232 2233| |2331 2332 2333| |
| '~--------------' '~--------------' '~--------------' |
'∊----------------------------------------------------'
```

```
To select the red items, you can use
¯1↑[2]¨(⊂¯2 2)↑¨A     ⍝ 10 sec (looping 1,000 times)
or
(⊂(2 3)(,2))⌷¨A       ⍝  7 sec (looping 1,000 times)
```

```
                                  (⊂⊂1 3)⌷[1]¨
.→----------------------------------------------------------.
↓     (⊂1 3)⌷[1]           (⊂1 3)⌷[1]           (⊂1 3)⌷[1]  |
|  .→---------------.  .→---------------.  .→---------------.  |
|  ↓1111 1112 1113|  ↓1211 1212 1213|  ↓1311 1312 1313|  |
|  |1121 1122 1123|  |1221 1222 1223|  |1321 1322 1323|  |
|  |1131 1132 1133|  |1231 1232 1233|  |1331 1332 1333|  |
|  '~---------------'  '~---------------'  '~---------------'  |
|                                                           |
|     (⊂1 3)⌷[1]           (⊂1 3)⌷[1]           (⊂1 3)⌷[1]  |
|  .→---------------.  .→---------------.  .→---------------.  |
|  ↓2111 2112 2113|  ↓2211 2212 2213|  ↓2311 2312 2313|  |
|  |2121 2122 2123|  |2221 2222 2223|  |2321 2322 2323|  |
|  |2131 2132 2133|  |2231 2232 2233|  |2331 2332 2333|  |
|  '~---------------'  '~---------------'  '~---------------'  |
'∊----------------------------------------------------------'
```

```
To select the red items, you can use
(1↑[1]¨A),[1]¨¯1↑[1]¨A      17.4 sec (looping 1,000 times)
or
(⊂⊂1 3)⌷[1]¨A                7.5 sec (looping 1,000 times)
```

Recommendation

When you need to uniformly index a matrix consisting of nested vectors as items, squad indexing with the left argument enclosed as a scalar can be a good alternative.

The scalar extension could simplify the expression and speed up the execution time.

APL Optimization Techniques For Commercial Applications

Questions?

Eugene Ying
2013