Efficient Scripting of XML Processes

David A. Lee
Senior Principal Software Engineer
Epocrates, Inc
dlee@epocrates.com

Norman Walsh
Principal Technologist
Mark Logic Inc.
norman.walsh@marklogic.com

Agenda

- Scripting
  - Why use scripting ?
  - Common scripting languages
- The “Brick Wall” of scripting
- Goal of this research
- Test Cases
  - Input data
  - Test Descriptions
  - Scripting Languages
  - Operating Systems
  - Hardware
- Results
- Conclusions and suggestions

Why use ‘Scripting’ ?

- Steps not easily performed by a single language or program
  - Split into “Manageable Tasks”
  - “Glued” together by a scripting language
- XML and Non-XML processing intermixed
- Easier then writing in a normal programming language (like Java, C++ etc).
- Choice to use different tools for different steps
- Easier to develop and debug in small pieces
- Rarely the focus of performance optimizations
  - Often the “Black Sheep” of real world data processing.

Common Scripting Languages

- Mature / Legacy languages
  - DOS (CMD.EXE)
  - Unix Shell (sh / bash / ksh)
  - perl
  - …

- Newer XML oriented scripting languages
  - xproc
  - xmlsh

The Brick Wall

- Scripting works great for small tasks
  - Tens of Files
  - Tens of commands
  - Seconds to Minutes of runtime

- Then the “Brick Wall” is hit
  - Hundreds of files
  - Thousands of commands
  - Hours to Days of runtime

- Often give up on Scripting due to the “Brick Wall”

Goal of this research

Goals of research project

- Identify bottlenecks and causes of performance problems
  - “The Brick Wall”
    - > is it real ?
    - > What causes it ?
    - > Can it be knocked down ?
- Compare scripting languages
  - Validate experience with legacy languages
  - Validate goals of newer XML scripting languages
- Compare Operating systems and hardware
  - Does tossing CPU power and $ at a problem solve it ?
- Use “Real World” tests case
- Reintroduce scripting as a viable technique

Test Cases

- Taken from “real world” processing at Epocrates
  - Simplified somewhat to target specific areas
    - focus on scripting overhead
    - Many small operations over many files

- Aimed at problems where scripting is often used
  - Difficult to solve in a single processing language
  - Easier to develop and debug as ‘manageable pieces’
  - Mix of XML and non-xml processes

Input Data

- 660 XML files
  - 70 MB total
  - Avg 100k each
  - Real world data
    - **Clinical “Monographs” describe disease, causes, treatments**
    - **Used in Epocrates online and handheld products**
  - 107 distinct element tags

- 3383 Image files
  - Contents not used in tests

Test Cases

Test Descriptions

- Baselines
  - baseline1 – launch scripting command interpreter only
  - baseline2 – Run a trivial xquery (<empty/>)
- Test1
  - Produce a table of contents (xquery across all files)
- Test2
  - Produce a list of images depending on the existence of actual image files in the file system.
- Test3
  - Content generation (xquery & xslt)
  - Complex formatting and conditional logic
  - Which processes to run are data dependant

Test Cases

Test Process Matrix

	Input Files	Input Size	Output Files	Output Size	xquery	xslt
baseline1	0		0		0	0
baseline2	0		0		1	0
test1	660	69,536,759	1	89,437	660	1
test2	660	69,536,759	1	236,743	2,194	660
test3	660	69,536,759	5,229	19,914,764	23,086	5,269

Scripting Languages Tested

- DOS
  - CMD.EXE
  - “Classic” scripting language for windows users
  - Only runs on Windows systems
- bash
  - Modern version of the unix shell
  - Runs on Windows (cygwin) Linux and Mac
- Calabash (XProc implementation)
  - Java based runtime
  - Runs on Windows, Linux, Mac
- xmlsh
  - Java based runtime
  - Runs on Windows, Linux, Mac

XML Processor Runtime

- XML processing limited to
  - XQuery (and xpath)
  - XSLT
- Implemented with Saxon B (9.1.0.6)
- All languages using the same exact JVM and Saxon jars.
  - Same JVM runtime
  - JVM startup parameters identical
  - Same Saxon library
  - Same XML parser
  - Same OS environment

Hardware tested

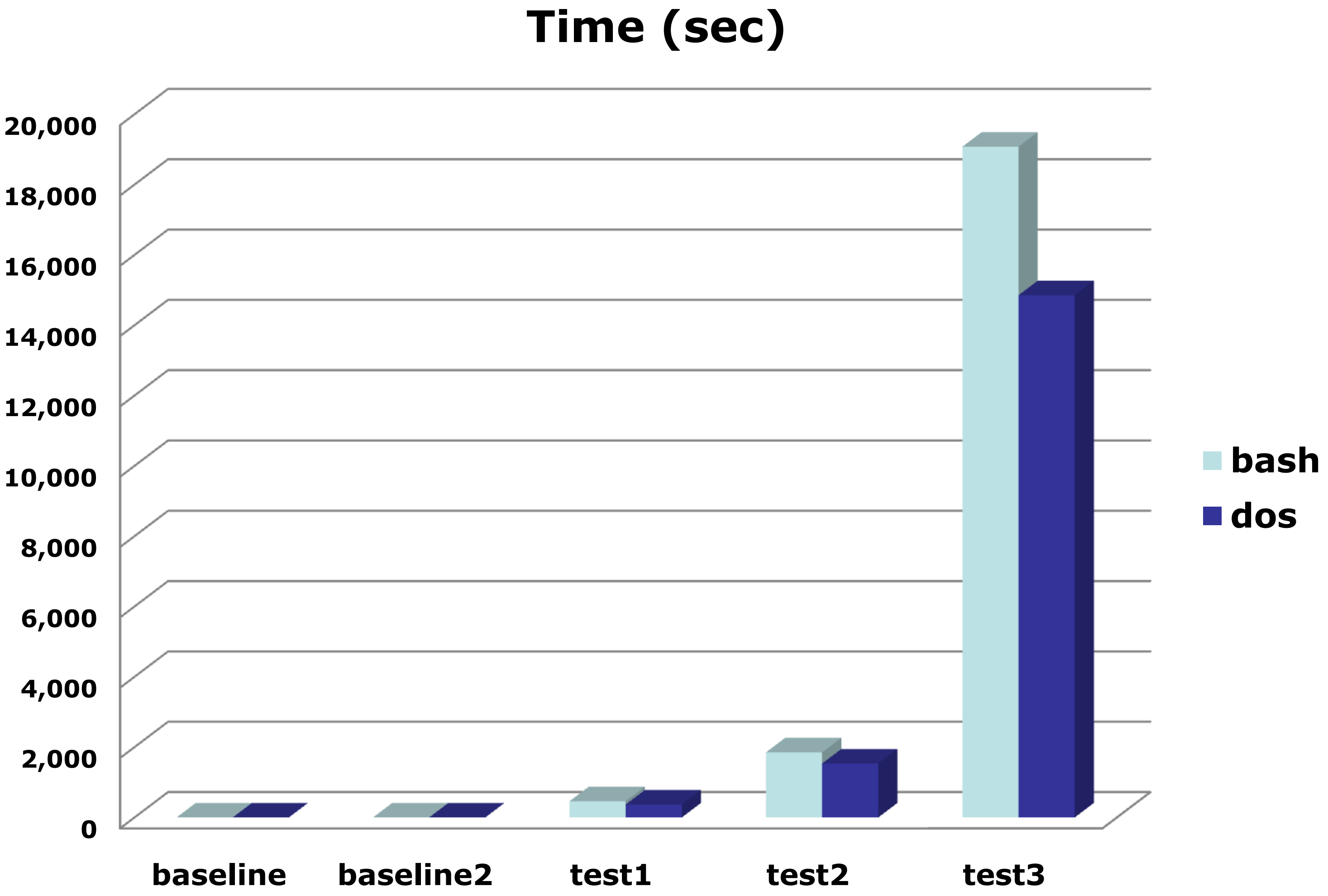
- Variety of available hardware tested
- Limited to what we could get our hands on
  - Some desktop grade machines
  - Some server grade

- Goal was **not** to exhaustively test every hardware or language
  - Goal was to look for trends or anomalies

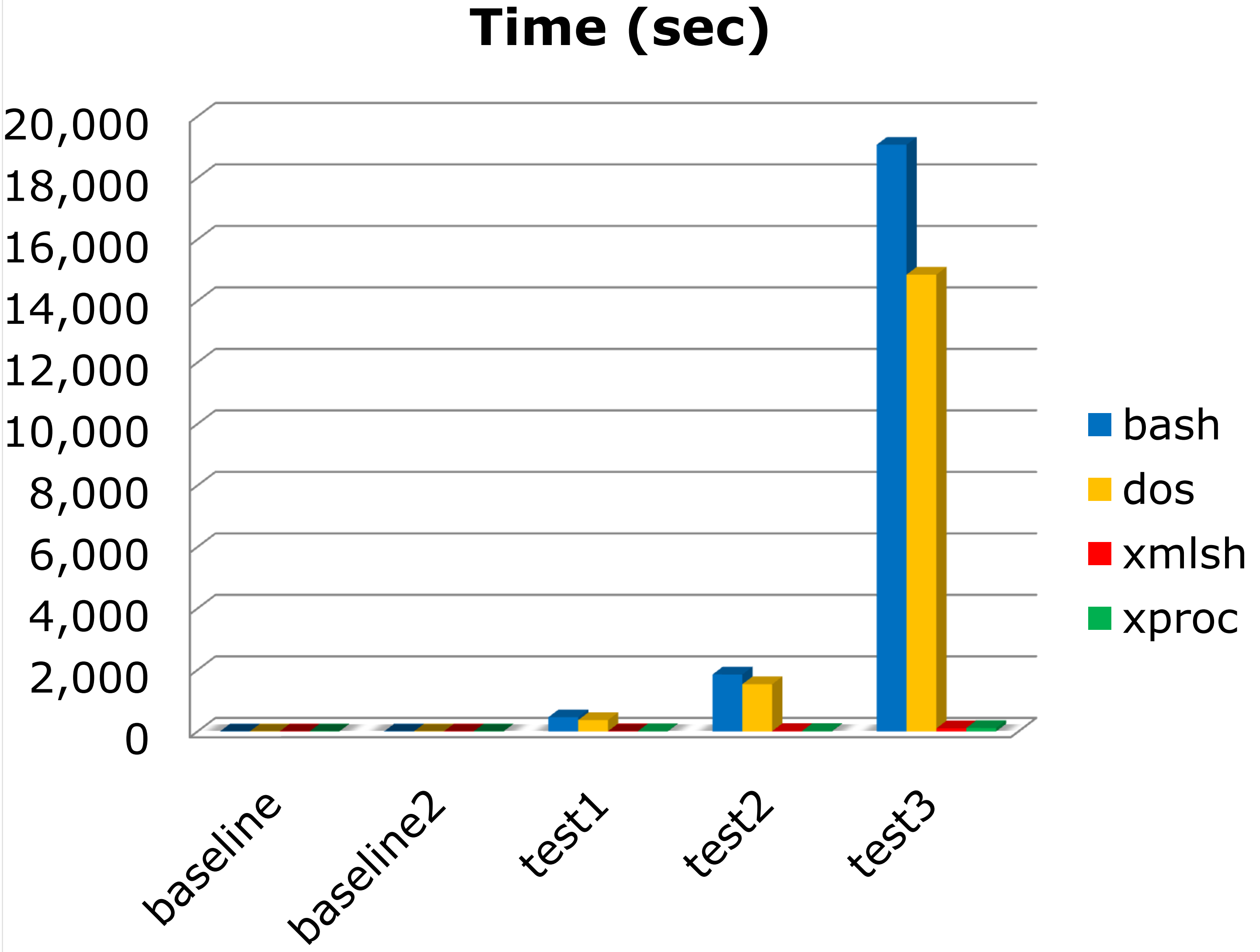
Operating Systems Tested

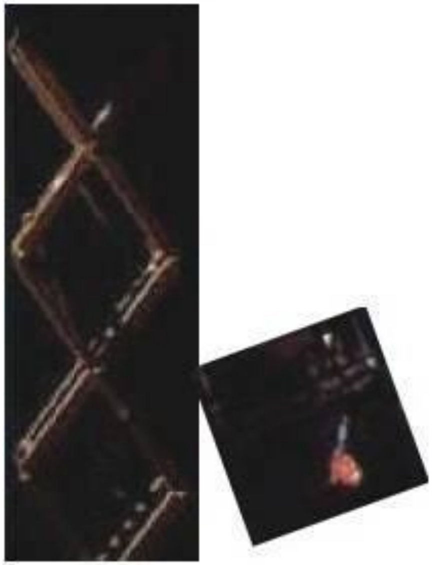
- Windows XP Professional
- Linux Fedora FC9
- Mac/OS (10.5)
- Solaris

Results – “The Brick Wall”



Results – "Tear Down That Wall !!!"



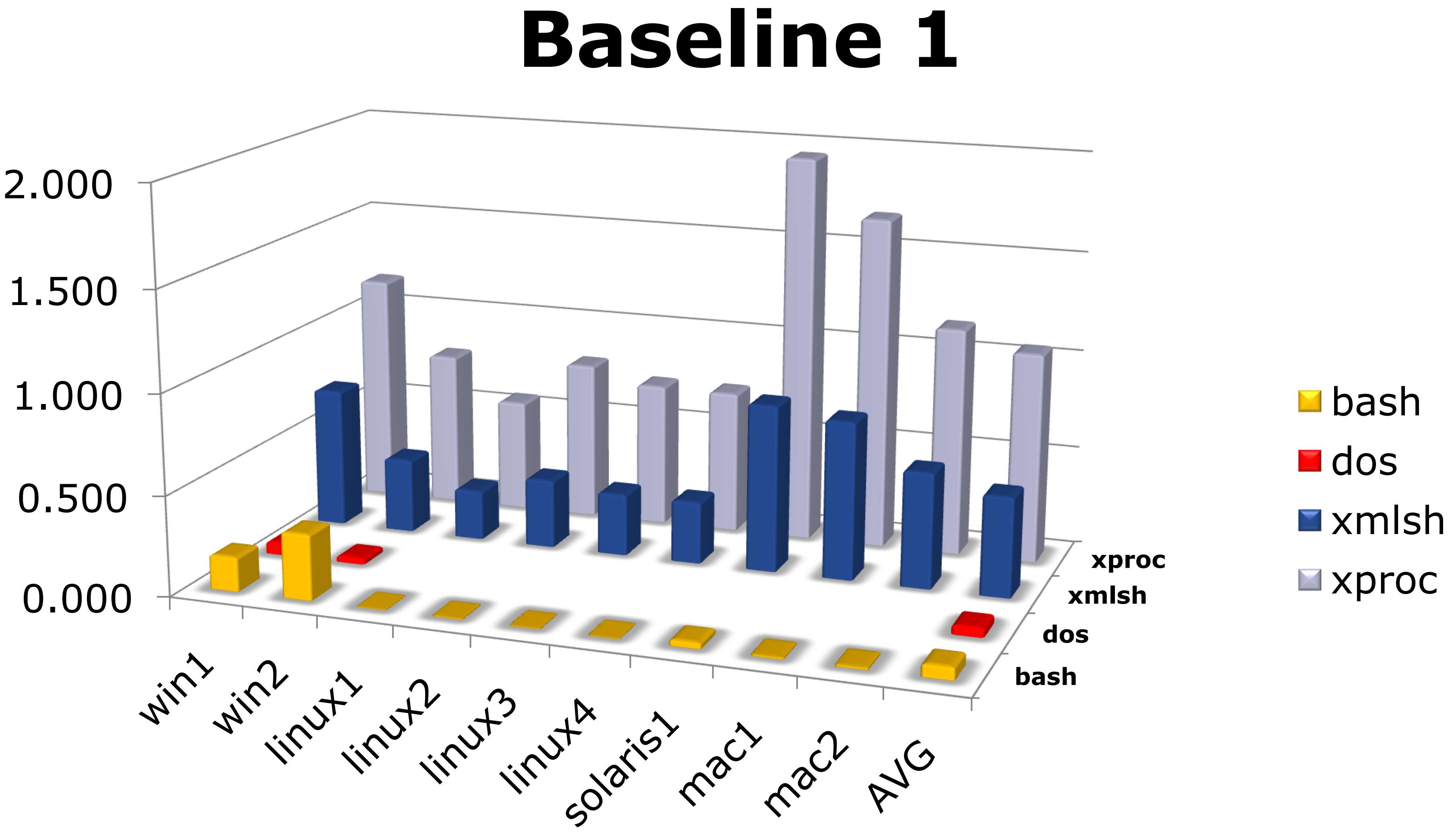


Results

Baseline 1

bash	echo '<empty/>'
cmd	@ECHO ^<empty/^>
xmlsh	echo "<empty/>"
xproc	<?xml version="1.0"?> <p:declare-step xmlns:p="http://www.w3.org/ns/xproc"> <p:input port="source"> <p:inline> <empty/> </p:inline> </p:input> <p:output port="result"/> <p:identity/> </p:declare-step>

Results

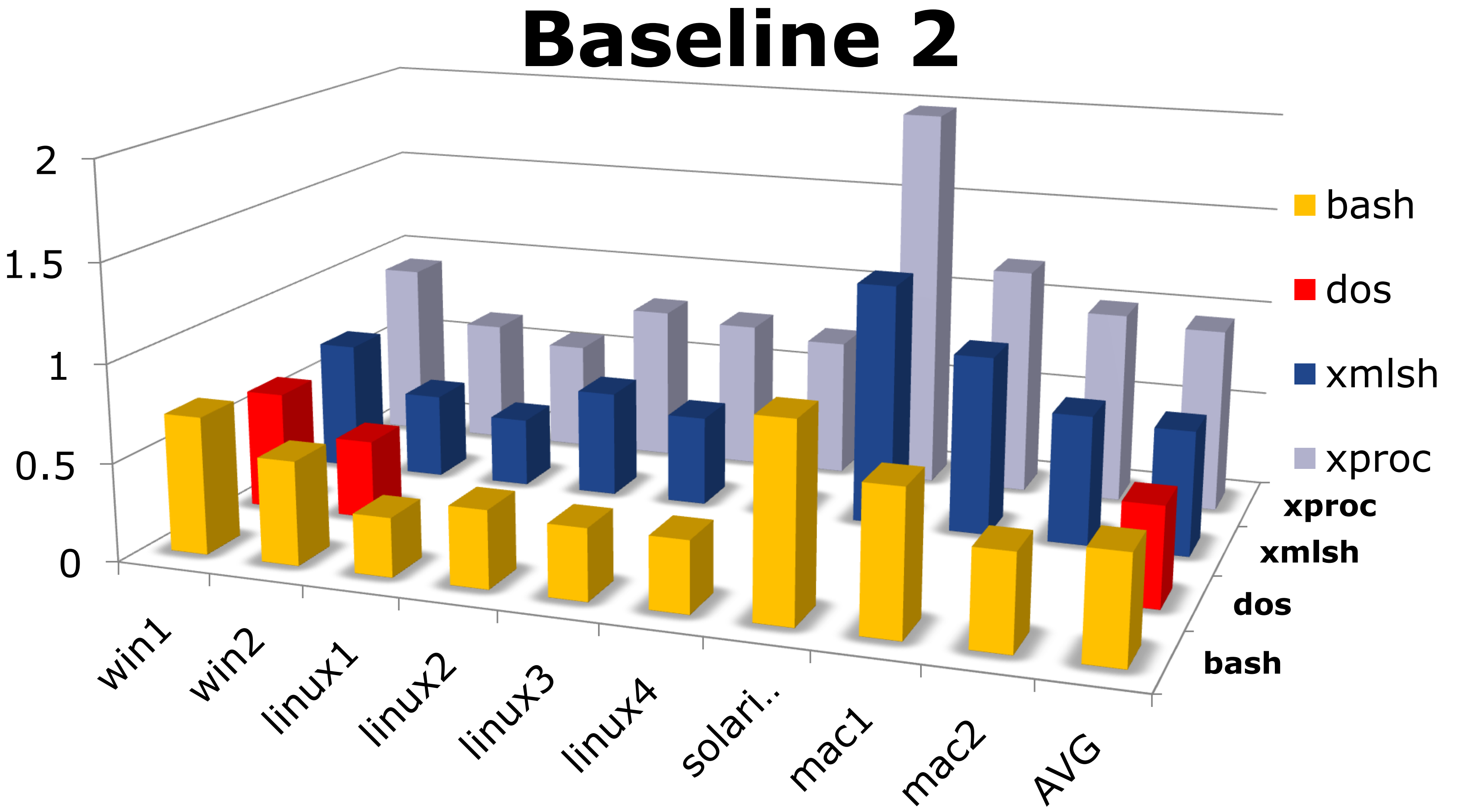

Baseline 1
2.000
1.500
1.000
0.500
0.000
win1
win2
linux1
linux2
linux3
linux4
solaris1
mac1
mac2
AVG
xproc
xmlsh
dos
bash
🟨 bash
🟥 dos
🟦 xmlsh
⬜ xproc

Results

Baseline 2

bash	xquery.sh -qs:'<empty/>'
cmd	xquery -qs:"<empty/>"
xmlsh	xquery -q "<empty/>" -n
xproc	<?xml version="1.0"?> <p:pipeline xmlns:p="http://www.w3.org/ns/xproc" xmlns:c="http://www.w3.org/ns/xproc-step"> <p:xquery> <p:input port="query"> <p:inline> <c:query>element {"empty"}{} </c:query> </p:inline> </p:input> </p:xquery> </p:pipeline>

Results


Baseline 2
2
1.5
1
0.5
0
win1
win2
linux1
linux2
linux3
linux4
solari..
mac1
mac2
AVG
xproc
xmlsh
dos
bash
🟨 bash
🟥 dos
🟦 xmlsh
⬜ xproc

Test 1

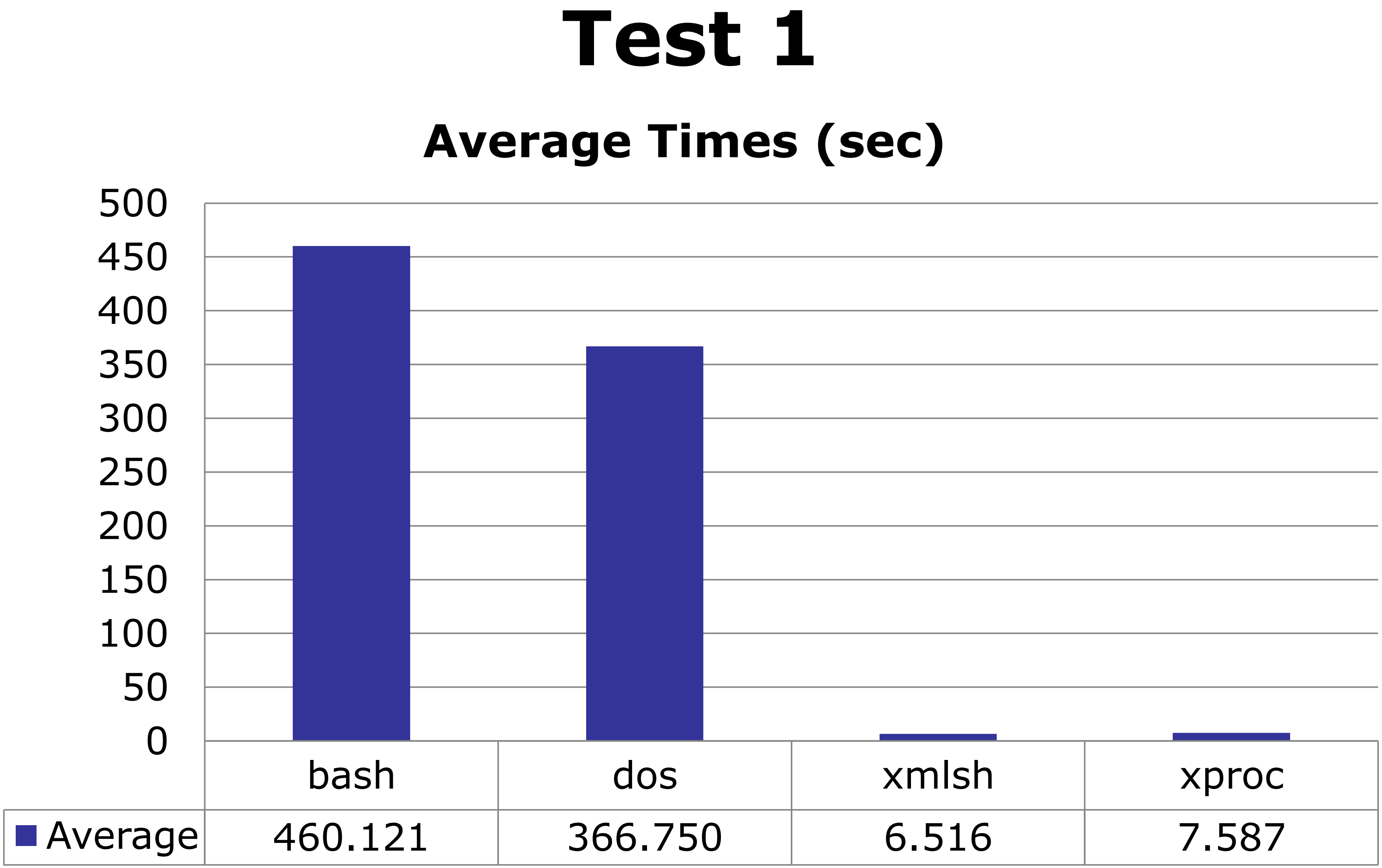
Create a table of contents

```
For each input file
        xquery to extract title

xslt result to html
```

Input Files	Input Size	Output Files	Output Size	xquery	xslt
660	69,536,759	1	89,437	660	1

Results



Test 1

Average Times (sec)

	bash	dos	xmlsh	xproc
■ Average	460.121	366.750	6.516	7.587

Test 2

List all images that actually exist as an image catalog

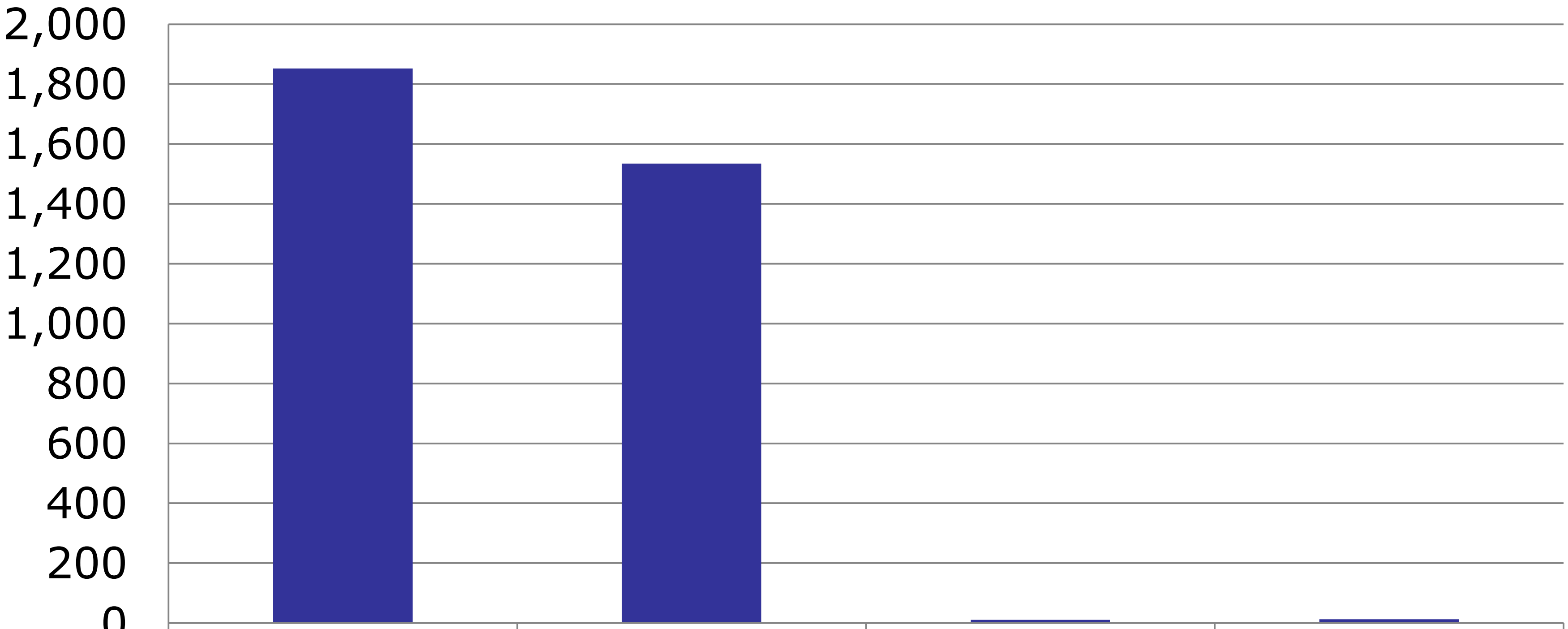
```
For each file
        xquery list of image filenames
        for each image name
                if image file exists
                        output image xml element
        xslt result to HTML
```

Input Files	Input Size	Output Files	Output Size	xquery	xslt
660	69,536,759	1	236,743	2,194	660

Results

Test 2

Average Time (sec)



	bash	dos	xmlsh	xproc
■ Average	1,851.800	1,534.383	10.657	12.699

Test 3

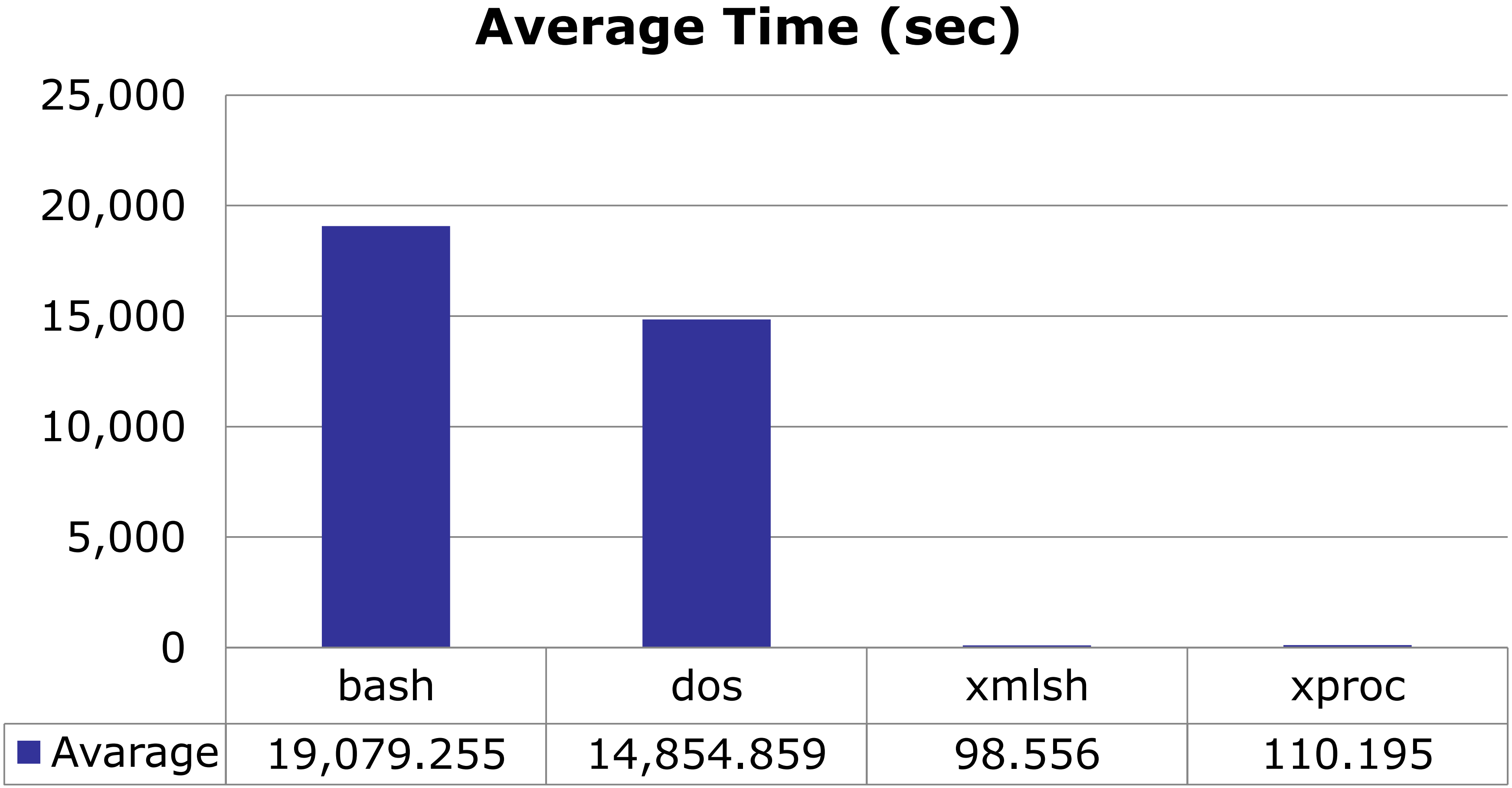
Format complex content

```
for every page type (from xml pages description)
    for every input file (topic)
        if test (xpath) page applies to this file
            xquery file to intermediate form
            xslt to HTML
            append html filename to topic index
        xslt topic list to html
```

Input Files	Input Size	Output Files	Output Size	xquery	xslt
660	69,536,759	5,229	19,914,764	23,086	5,269

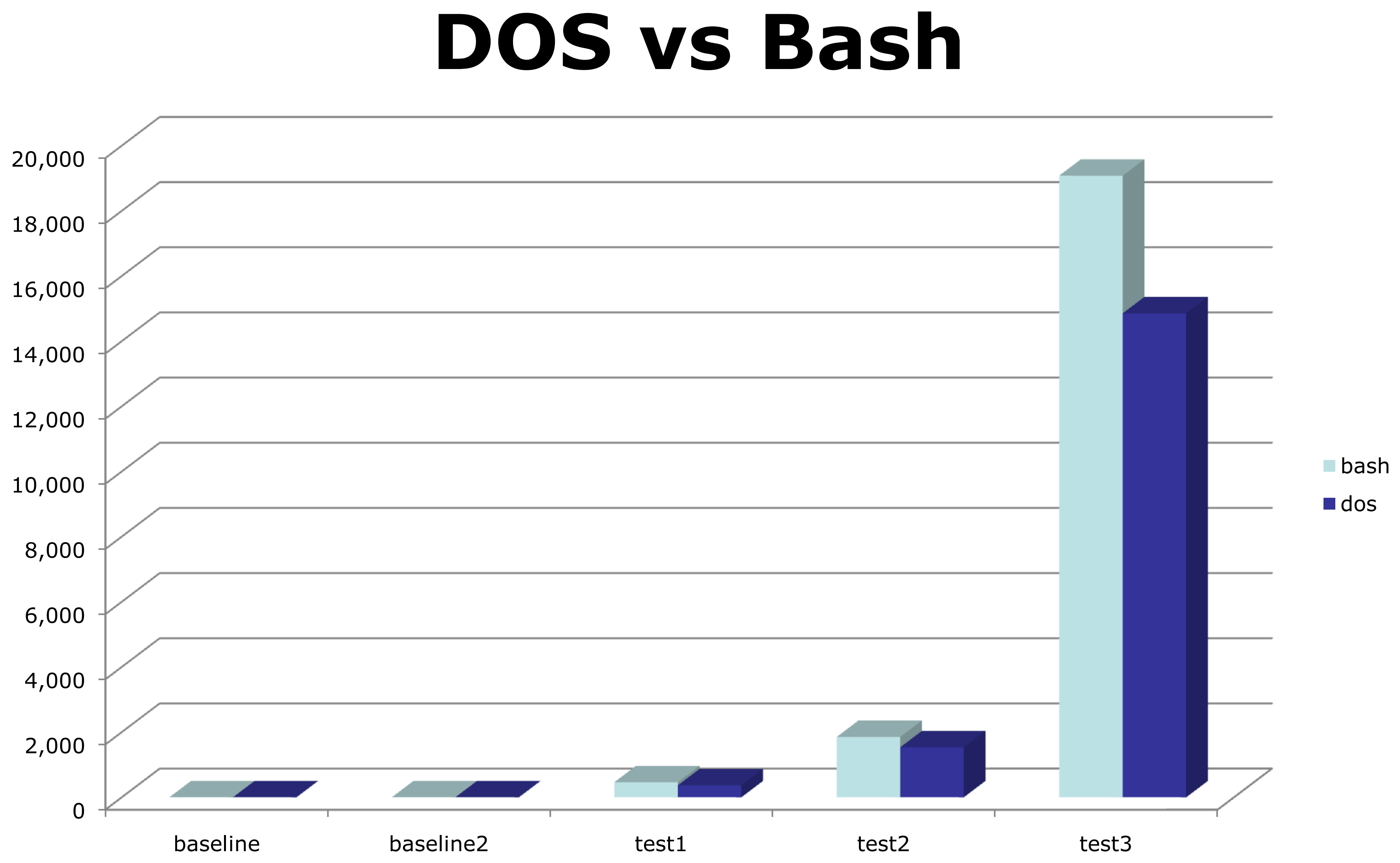
Results

Test 3

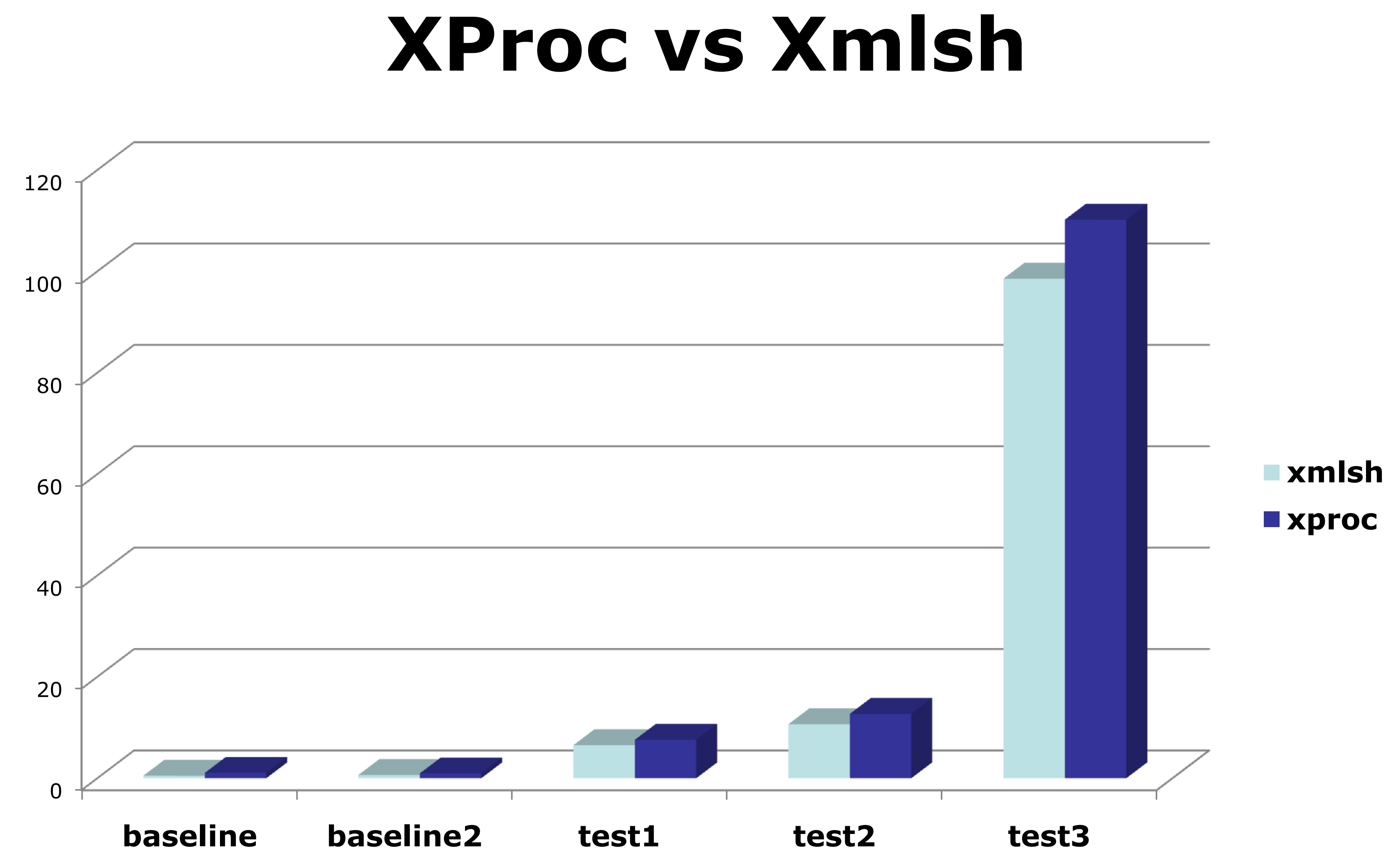


	bash	dos	xmlsh	xproc
■ Avarage	19,079.255	14,854.859	98.556	110.195

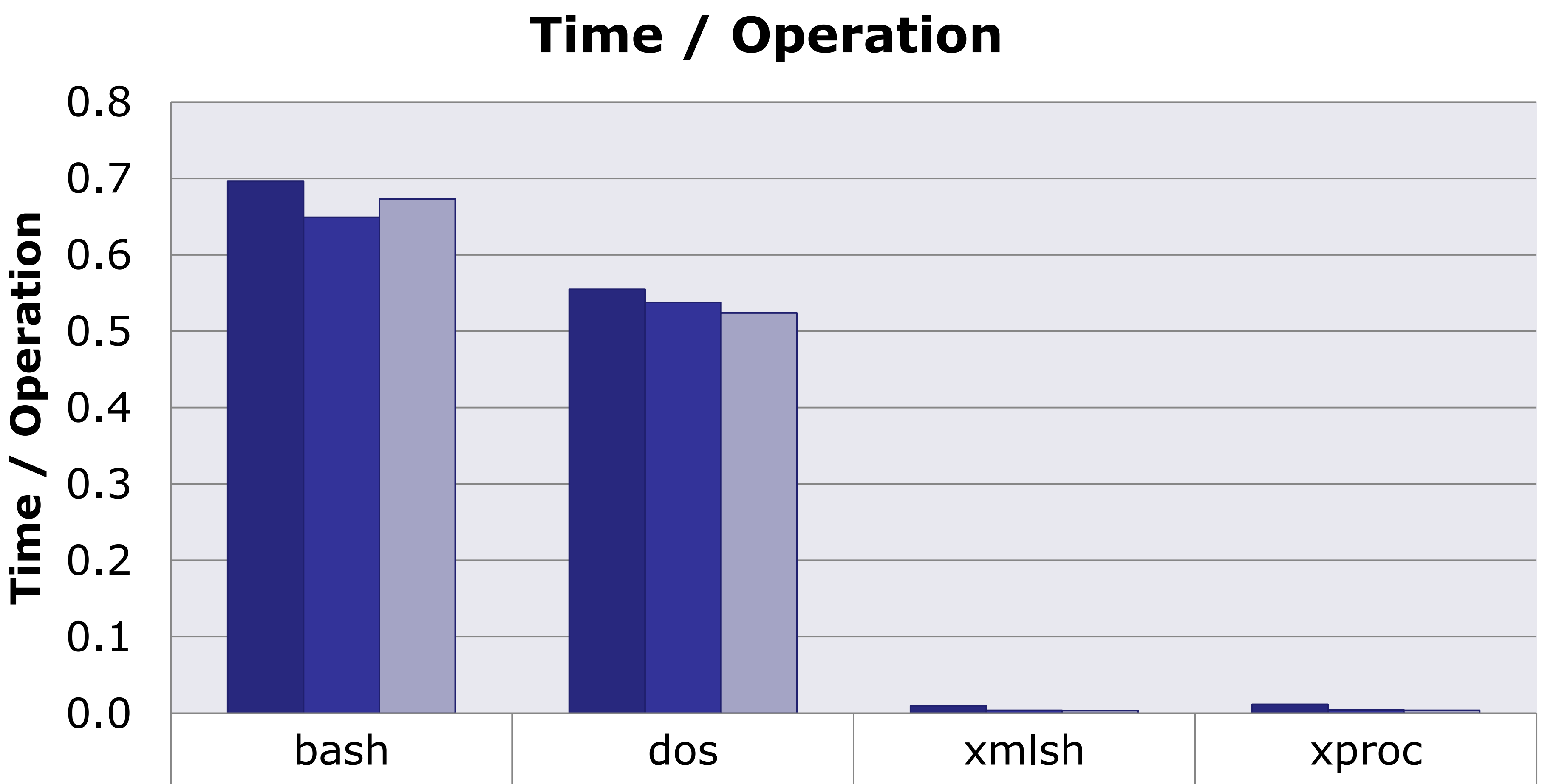
Results


DOS vs Bash
20,000
18,000
16,000
14,000
12,000
10,000
8,000
6,000
4,000
2,000
0
baseline
baseline2
test1
test2
test3
bash
dos

Results


XProc vs Xmlsh
120
100
80
60
40
20
0
baseline
baseline2
test1
test2
test3
xmlsh
xproc

Time linear to number of operations



	bash	dos	xmlsh	xproc
■ test1	0.696	0.555	0.010	0.011
■ test2	0.649	0.538	0.004	0.004
□ test3	0.673	0.524	0.003	0.004

Conclusions

- Scripting of few operations
  - not a problem
- Scripting of many operations
  - Performance linear to number of operations

- "Traditional Languages"
  - Brick Wall
  - Time/Operation high
    - Primarily startup time of Java/VM and sub process creation

- "XML Scripting Languages"
  - Overcome the "Brick Wall"
  - Choice of 'traditional' or 'XML based' language styles
  - Approx 200x faster
  - Runs within the same JVM/Process as XML operations / objects
    - xquery / xpath / xslt / parsing /seralizing / dom

Recommendations

- Traditional Solutions
  - Consolidate operations (“One Big Program”)
  - Run within a single language
    - Avoids startup overheads

- Leads to
  - Monolithic applications
  - Harder to debug
  - Forced to solve all problems in one tool/language

Recommendations

- XML Scripting Solution
  - Choose an “XML Scripting Language”
  - Minimum penalty for splitting up tasks to smaller pieces
    - runs within the same process/JVM
  - Not forced to use a single tool or language
    - Mix & match technologies (xslt, xquery , xpath , file/os utils etc)
- Leads to
  - Modular applications
  - Easier to debug
  - Choice of language for different steps
  - Quick prototyping
    - May work well enough for production
- No more fear of the “Brick Wall”
- Encourage scripting language authors to add native xml support

Thank You !

David A. Lee
Senior Principal Software Engineer
Epocrates, Inc
dlee@epocrates.com

xmlsh
http://www.xmlsh.org

Norman Walsh
Principal Technologist
Mark Logic Inc.
norman.walsh@marklogic.com

Calabash (xproc)
http://xmlcalabash.com