BEPUG

PowerShell Performance Tuning

"A few moments later"

Bruno Buyck
06/03/2025

BIG THANKS TO :

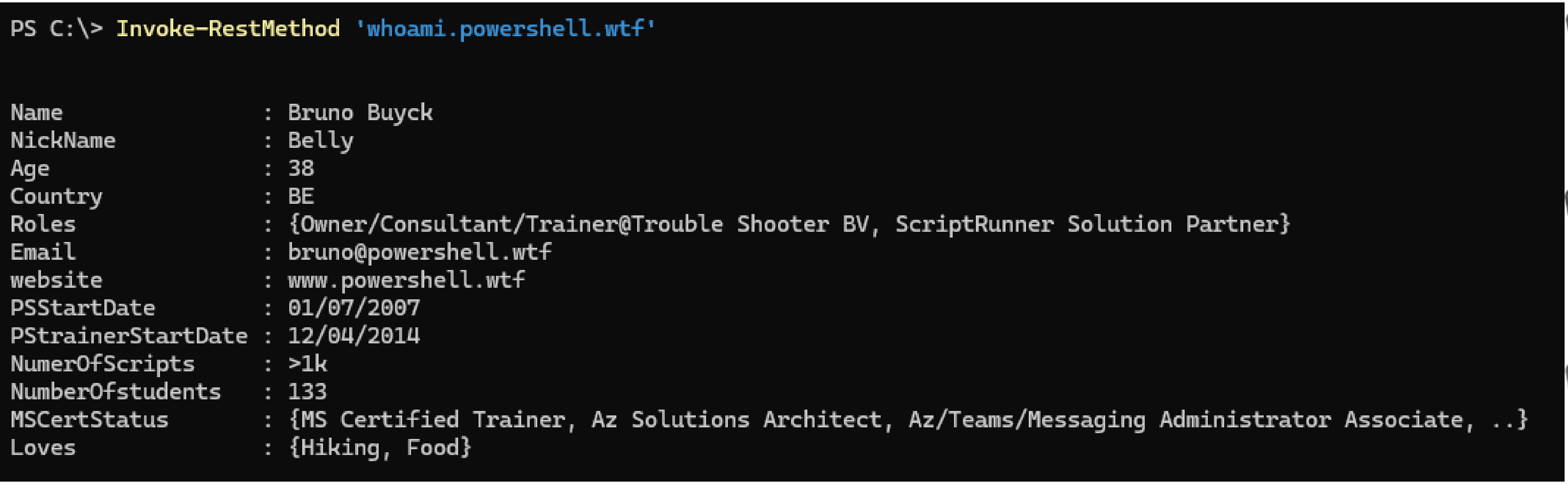
AXXES_

BEPUG

AGENDA

- ABOUT
  - WHOAMI
  - UPCOMING TRAININGS
- PERFORMANCE TUNING
  - WTF
  - Measuring
  - PS Tuning principles
- BUSINESS CASE
- Q & A

WHOAMI



```
PS C:\> Invoke-RestMethod 'whoami.powershell.wtf'


Name                : Bruno Buyck
NickName            : Belly
Age                 : 38
Country             : BE
Roles               : {Owner/Consultant/Trainer@Trouble Shooter BV, ScriptRunner Solution Partner}
Email               : bruno@powershell.wtf
website             : www.powershell.wtf
PSStartDate         : 01/07/2007
PStrainerStartDate  : 12/04/2014
NumerOfScripts      : >1k
NumberOfstudents    : 133
MSCertStatus        : {MS Certified Trainer, Az Solutions Architect, Az/Teams/Messaging Administrator Associate, ..}
Loves               : {Hiking, Food}
```

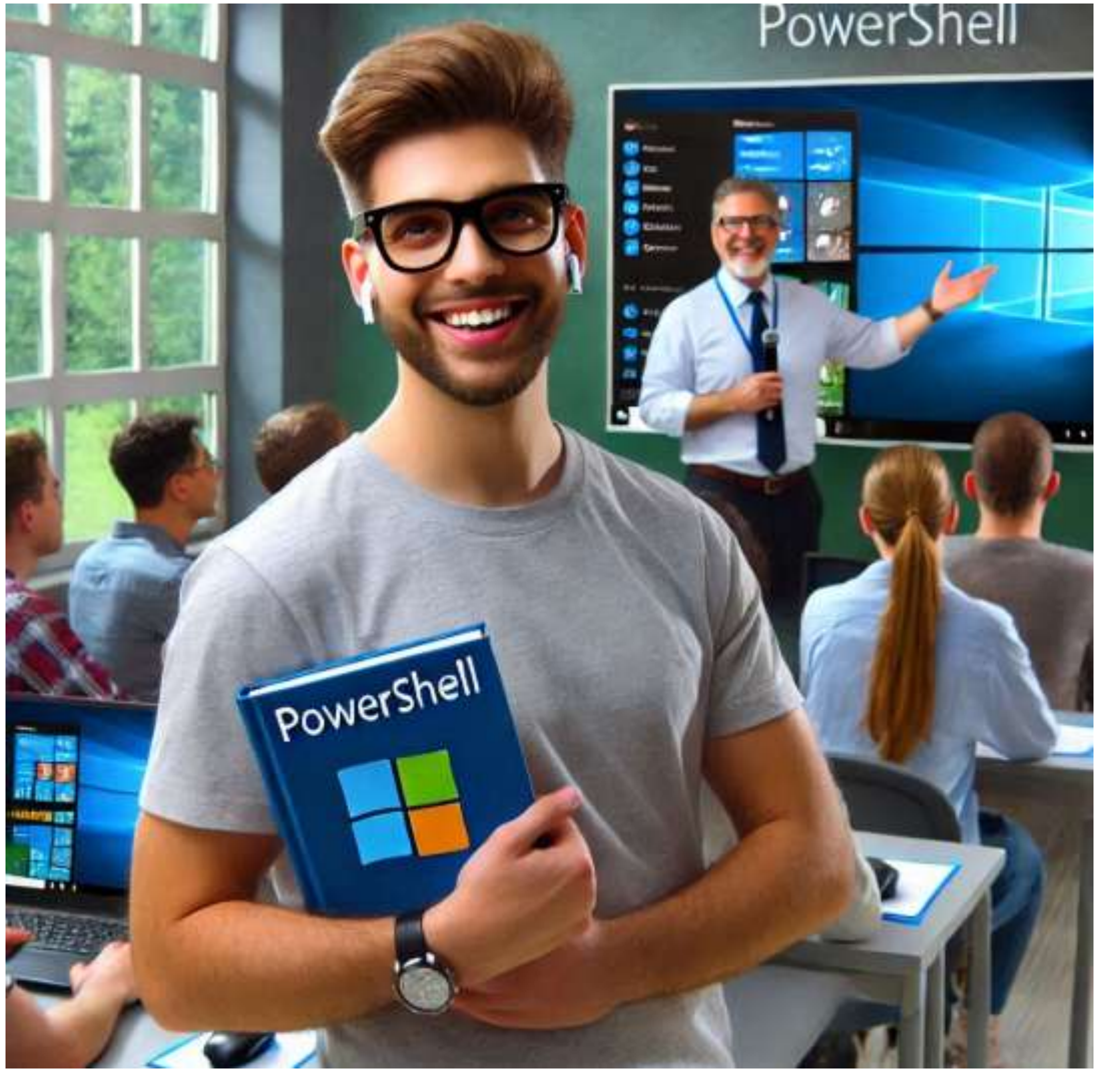
Microsoft CERTIFIED AZURE FUNDAMENTALS
Microsoft CERTIFIED AZURE ADMINISTRATOR ASSOCIATE
Microsoft CERTIFIED AZURE SOLUTIONS ARCHITECT EXPERT
Microsoft 365 CERTIFIED TEAMS ADMINISTRATOR ASSOCIATE
Microsoft 365 CERTIFIED MESSAGING ADMINISTRATOR ASSOCIATE

UPCOMING TRAININGS

- Become a PS Hero in 3 days :

- 02/06/2025 - 05/06/2025
- 24/11/2025 - 26/11/2025
  - Dutch
  - Limited to 12 people
  - Region Antwerp

- sales@powershell.wtf (Lisa)



Performance Tuning WTF !

BEPUG

“Success is the sum of small efforts, repeated day-in and day-out.”
Robert Collier

Performance tuning involves optimizing your scripts to make them run more efficiently, ensuring they use fewer system resources and run faster.

WARNING !

- Performance tuning is subject to enviroment circumstances

- Memory & CPU sizing
- Operating system
- PowerShell Version
- Script content
- …………

PERFORMANCE TUNING METHODOLOGY

1. **Baseline Measurement & Identify Bottlenecks**

2. **Optimize:**

   Pipelining

   Variables

   Data

   Loops

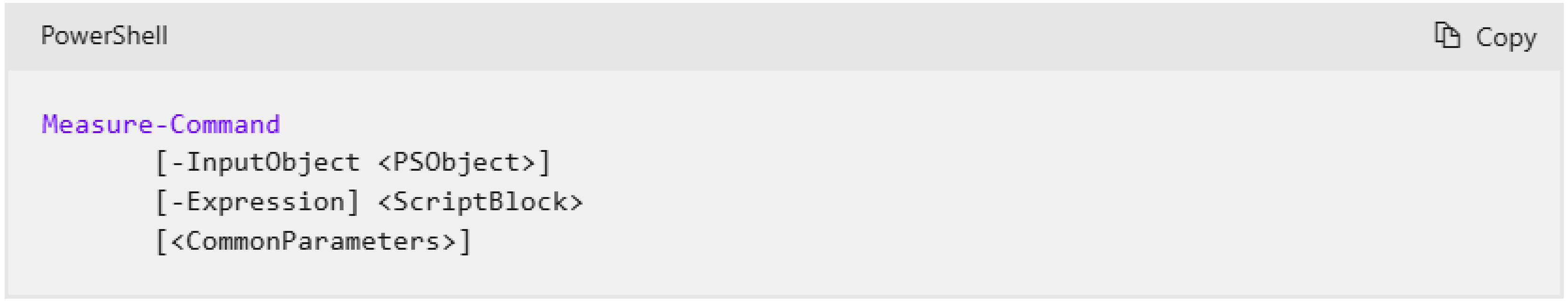
Baseline
Bottlenecks
Measurement

BASELINE MEASUREMENT

- Measure-Command
- [SYSTEM.DIAGNOSTICS.STOPWATCH]
- Profiler + Speedscope.app

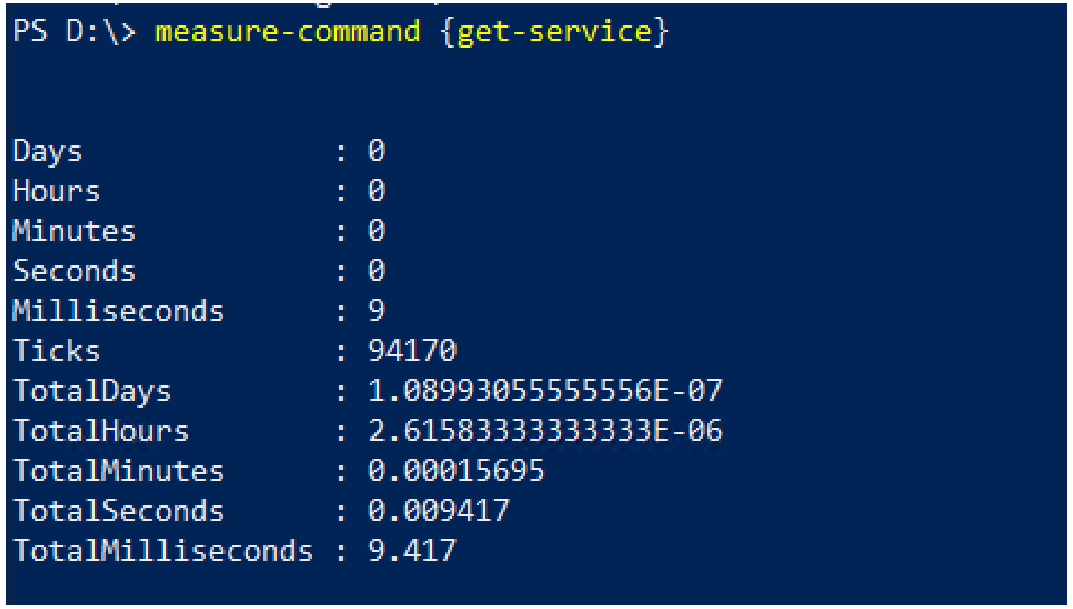
MEASURE-COMMAND

Syntax



PowerShell Copy

```
Measure-Command
       [-InputObject <PSObject>]
       [-Expression] <ScriptBlock>
       [<CommonParameters>]
```



```
PS D:\> measure-command {get-service}


Days              : 0
Hours             : 0
Minutes           : 0
Seconds           : 0
Milliseconds      : 9
Ticks             : 94170
TotalDays         : 1.08993055555556E-07
TotalHours        : 2.61583333333333E-06
TotalMinutes      : 0.00015695
TotalSeconds      : 0.009417
TotalMilliseconds : 9.417
```

Get-help measure-command –Online

SYSTEM.DIAGNOSTICS.STOPWATCH

- .NET class

```
$stopwatch =  [system.diagnostics.stopwatch]::StartNew()
get-service | out-null
$stopwatch.Stop()
$stopwatch
$stopwatch.Elapsed
```

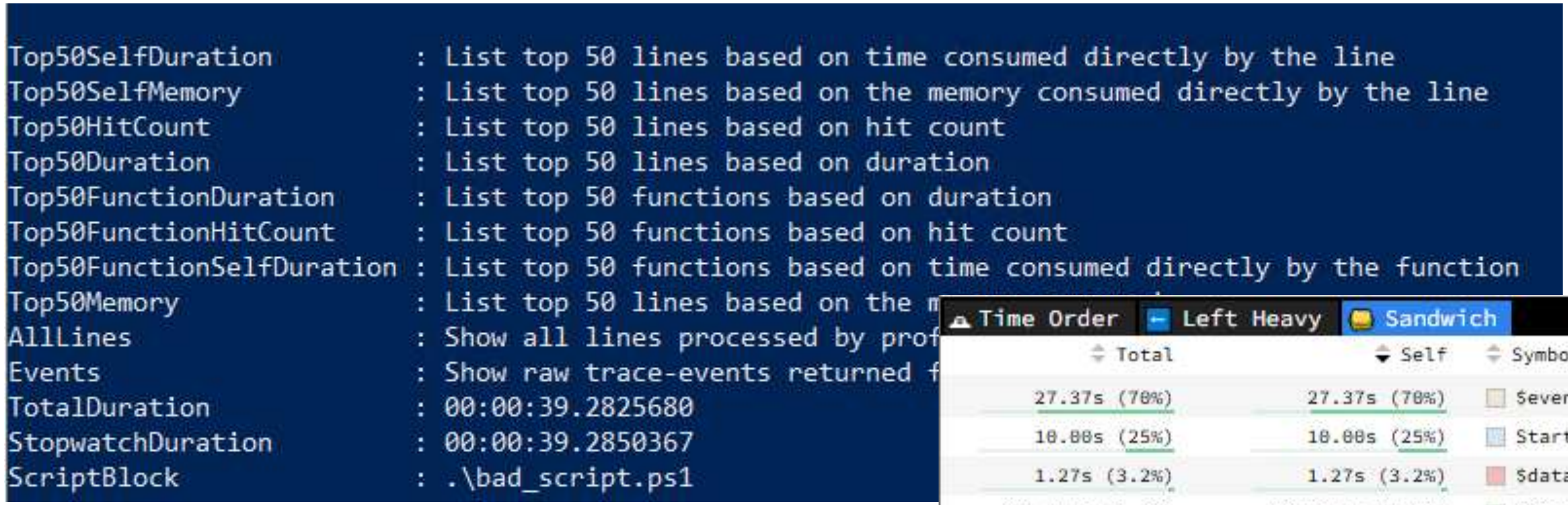
https://learn.microsoft.com/en-us/dotnet/api/system.diagnostics.stopwatch?view=net-9.0

PROFILER MODULE

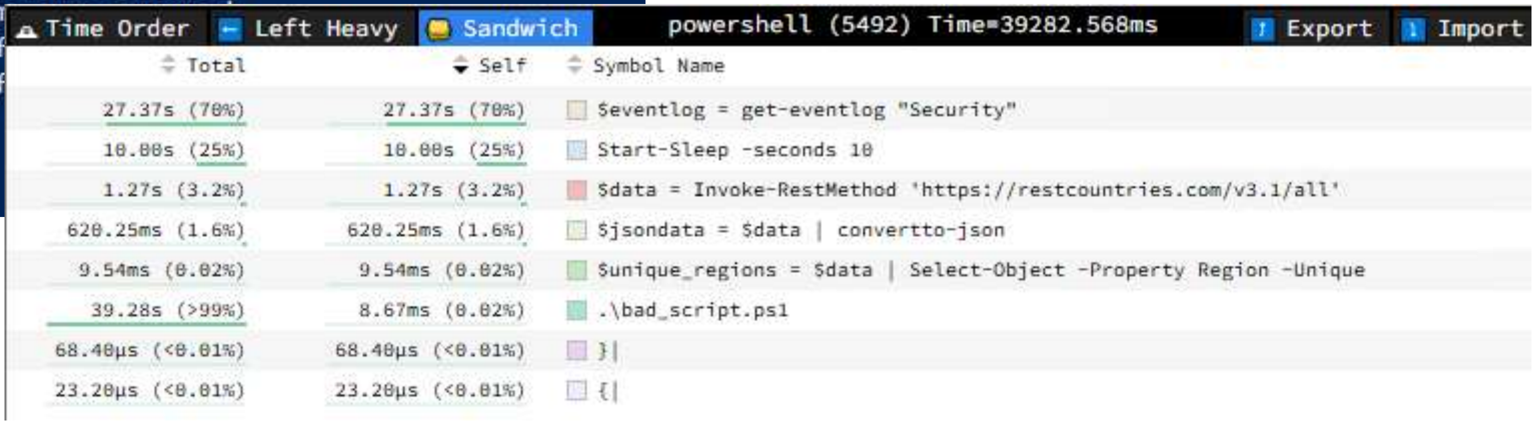
```
Install-Module Profiler

$TraceData = Trace-Script -ScriptBlock {.\bad_script.ps1} -ExportPath 'D:\'

$TraceData.Top50Duration
```



```
Top50SelfDuration         : List top 50 lines based on time consumed directly by the line
Top50SelfMemory           : List top 50 lines based on the memory consumed directly by the line
Top50HitCount             : List top 50 lines based on hit count
Top50Duration             : List top 50 lines based on duration
Top50FunctionDuration     : List top 50 functions based on duration
Top50FunctionHitCount     : List top 50 functions based on hit count
Top50FunctionSelfDuration : List top 50 functions based on time consumed directly by the function
Top50Memory               : List top 50 lines based on the m
AllLines                  : Show all lines processed by prof
Events                    : Show raw trace-events returned f
TotalDuration             : 00:00:39.2825680
StopwatchDuration         : 00:00:39.2850367
ScriptBlock               : .\bad_script.ps1
```



powershell (5492) Time=39282.568ms

Total	Self	Symbol Name
27.37s (70%)	27.37s (70%)	$eventlog = get-eventlog "Security"
10.00s (25%)	10.00s (25%)	Start-Sleep -seconds 10
1.27s (3.2%)	1.27s (3.2%)	$data = Invoke-RestMethod 'https://restcountries.com/v3.1/all'
620.25ms (1.6%)	620.25ms (1.6%)	$jsondata = $data | convertto-json
9.54ms (0.02%)	9.54ms (0.02%)	$unique_regions = $data | Select-Object -Property Region -Unique
39.28s (>99%)	8.67ms (0.02%)	.\bad_script.ps1
68.40µs (<0.01%)	68.40µs (<0.01%)	}|
23.20µs (<0.01%)	23.20µs (<0.01%)	{|

https://www.github.com/nohwnd/Profiler
https://www.speedscope.app/

Optimize
Tuning principles

> Pipelining

- Avoid using pipelines
- Look at properties & methods of objects (get-member)

> Variables

- Avoid creation of new objects/variables
- Use `System.Collections.ArrayList & PSObjects`
- Cast your variables

> Data

- Use hashtables for lookup
- Limit data size
- Look for alternatives

> Loops

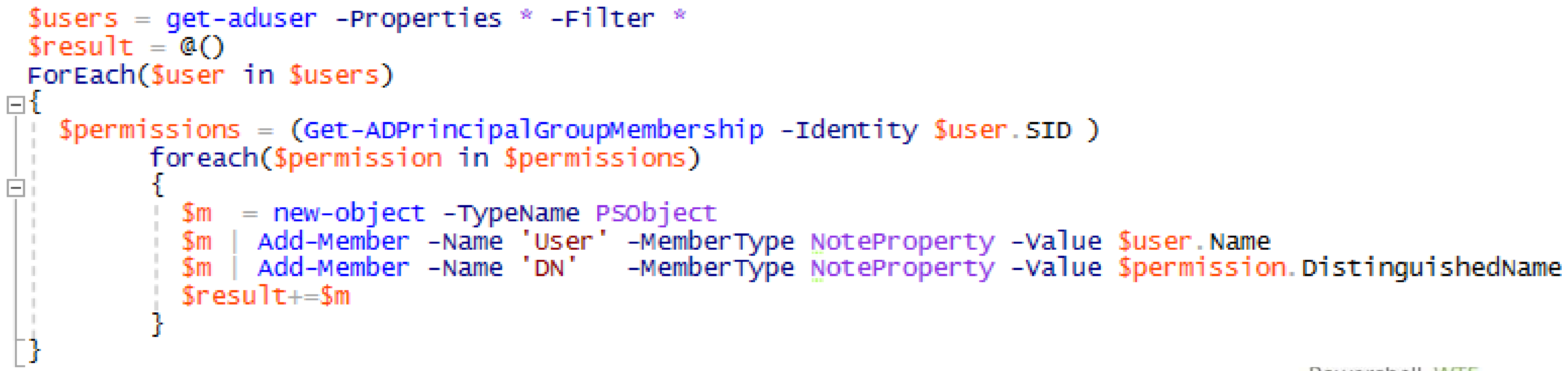
- ForEach-Object = Cool
- -Parallel (!)

BUSINESS CASE

BEPUG

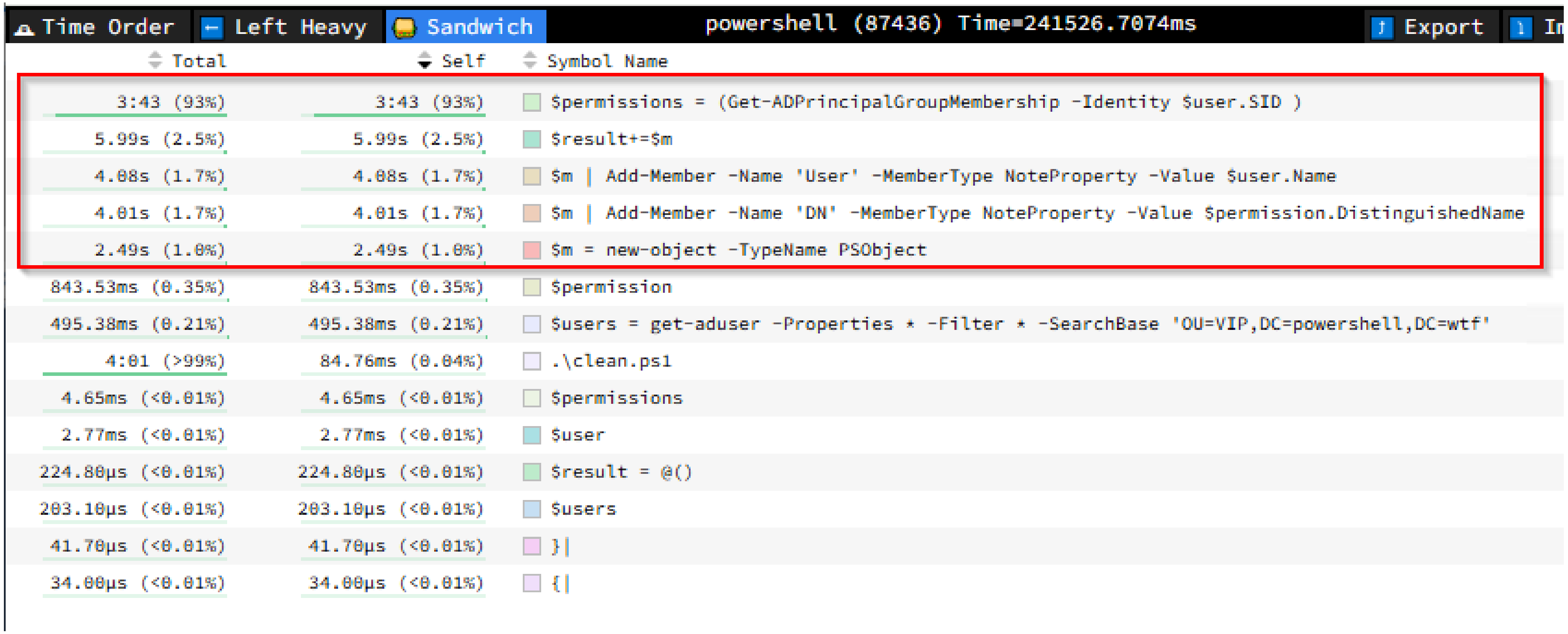
CUSTOMER EXAMPLE

- Slow : Execution time ***<u>hours</u>*** for 10k+ users
- PSVersion 5.1 (hard requirement)
- Lists table of Active Directory group memberships “UserName”,”GroupDN”
- Use of ADSI not allowed

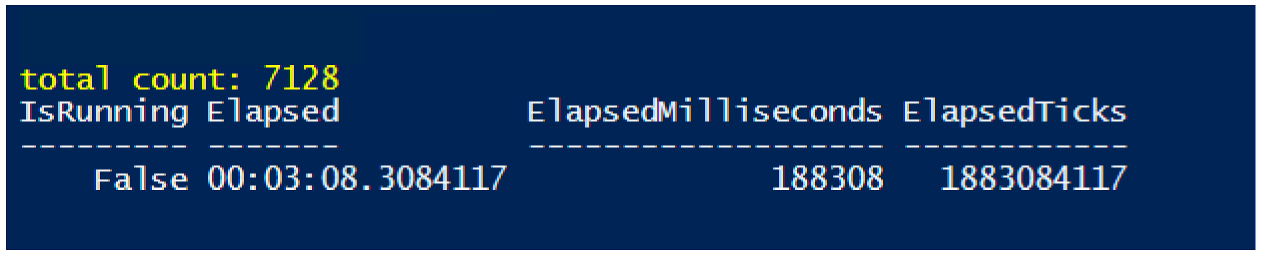


```
$users = get-aduser -Properties * -Filter *
$result = @()
ForEach($user in $users)
{
  $permissions = (Get-ADPrincipalGroupMembership -Identity $user.SID )
        foreach($permission in $permissions)
        {
          $m  = new-object -TypeName PSObject
          $m | Add-Member -Name 'User' -MemberType NoteProperty -Value $user.Name
          $m | Add-Member -Name 'DN'   -MemberType NoteProperty -Value $permission.DistinguishedName
          $result+=$m
        }
}
```

TESTING & PROFILING

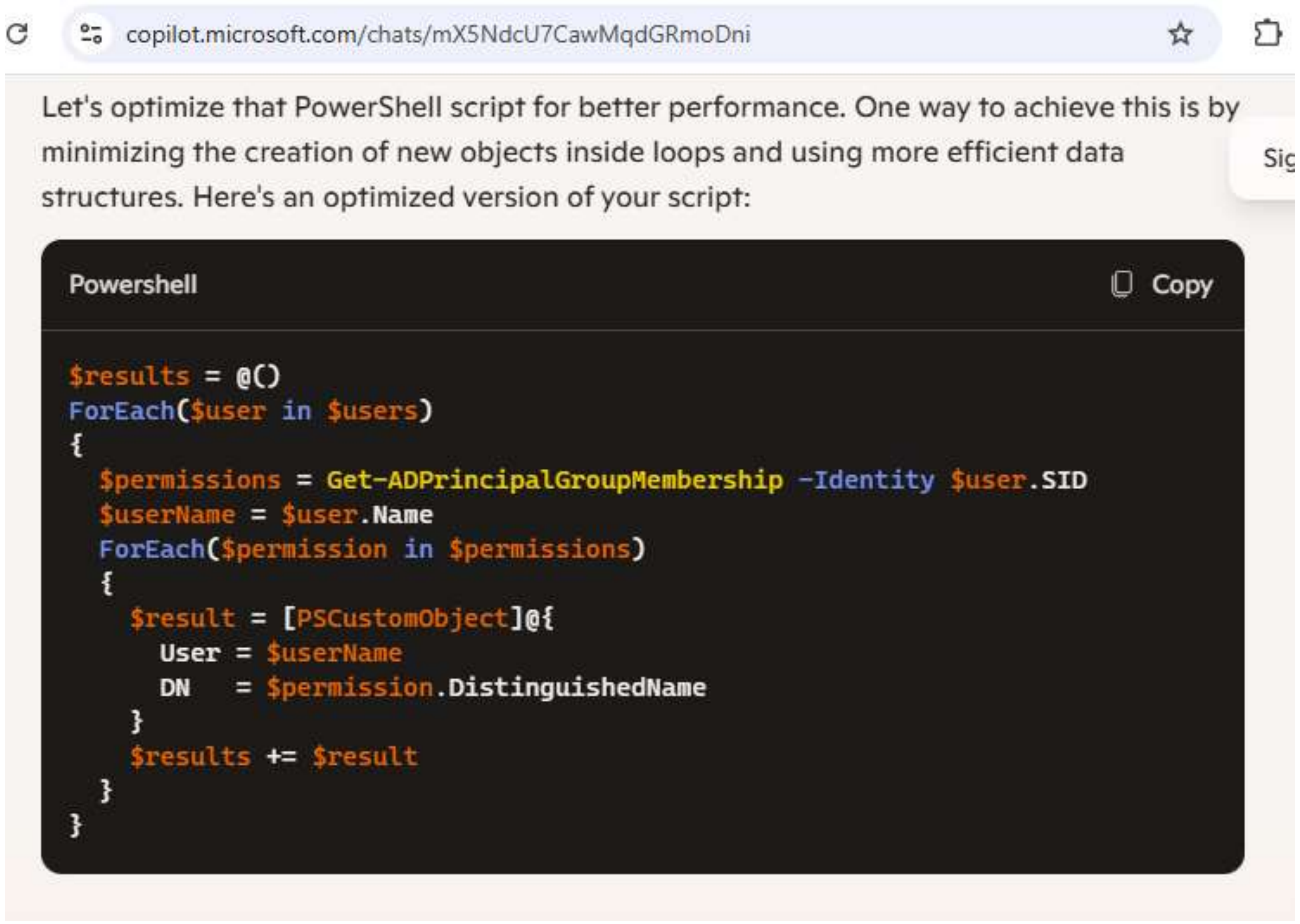


Total	Self	Symbol Name
3:43 (93%)	3:43 (93%)	$permissions = (Get-ADPrincipalGroupMembership -Identity $user.SID)
5.99s (2.5%)	5.99s (2.5%)	$result+=$m
4.08s (1.7%)	4.08s (1.7%)	$m | Add-Member -Name 'User' -MemberType NoteProperty -Value $user.Name
4.01s (1.7%)	4.01s (1.7%)	$m | Add-Member -Name 'DN' -MemberType NoteProperty -Value $permission.DistinguishedName
2.49s (1.0%)	2.49s (1.0%)	$m = new-object -TypeName PSObject
843.53ms (0.35%)	843.53ms (0.35%)	$permission
495.38ms (0.21%)	495.38ms (0.21%)	$users = get-aduser -Properties * -Filter * -SearchBase 'OU=VIP,DC=powershell,DC=wtf'
4:01 (>99%)	84.76ms (0.04%)	.\clean.ps1
4.65ms (<0.01%)	4.65ms (<0.01%)	$permissions
2.77ms (<0.01%)	2.77ms (<0.01%)	$user
224.80µs (<0.01%)	224.80µs (<0.01%)	$result = @()
203.10µs (<0.01%)	203.10µs (<0.01%)	$users
41.70µs (<0.01%)	41.70µs (<0.01%)	}
34.00µs (<0.01%)	34.00µs (<0.01%)	{



```
total count: 7128
IsRunning Elapsed          ElapsedMilliseconds ElapsedTicks
--------- -------          ------------------- ------------
    False 00:03:08.3084117              188308   1883084117
```

Step 0b : ASK AI



Let's optimize that PowerShell script for better performance. One way to achieve this is by minimizing the creation of new objects inside loops and using more efficient data structures. Here's an optimized version of your script:

```powershell
$results = @()
ForEach($user in $users)
{
  $permissions = Get-ADPrincipalGroupMembership -Identity $user.SID
  $userName = $user.Name
  ForEach($permission in $permissions)
  {
    $result = [PSCustomObject]@{
      User = $userName
      DN   = $permission.DistinguishedName
    }
    $results += $result
  }
}
```

TUNING

- Step 1 :
  - Replace `Get-ADPrincipalGroupMembership` by `get-aduser`
- Step 2 :
  - Fixing the missing PrimaryGroup
- Step 3 :
  - Use `System.Collections.ArrayList`
- Step 4 :
  - Consolidate data and variables
- Step 5 :
  - Reduce loops

SUMMARY

- Profile your scripts
- Reduce object size aka “filter early”
- Avoid (new) Object creation
- Avoid loops and pipelines
- Use object methods & properties

Q & A

THANK
YOU
FOR
YOUR
ATTENTION

Bruno@powershell.wtf