Microsoft

Hash Tables

Learnings covered in this Unit

- What is a Hash Table
- Creating Hash Tables
- Working with Hash Tables
- Techniques and use cases

What is a Hash Table

Hash Table Overview

COLLECTION WITH A CONSISTENT SEARCH TIME

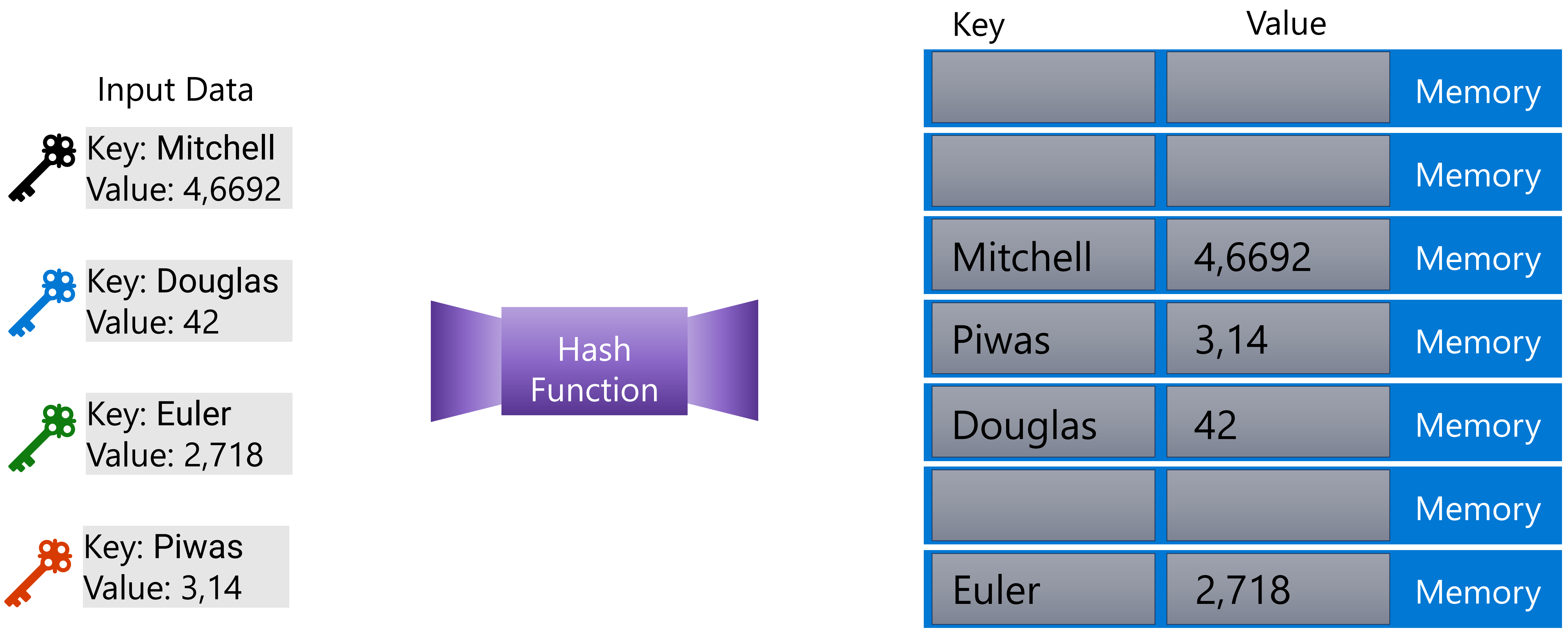
MEMORY LOCATION DETERMINED BY THE HASH ALGORITHM

USES KEY VALUE PAIRS TO STORE DATA

VALUE CAN BE ANY DATATYPE

Hash Table Storage

A collection where a hash function determines the memory location of the data stored



Input Data

Key: Mitchell
Value: 4,6692

Key: Douglas
Value: 42

Key: Euler
Value: 2,718

Key: Piwas
Value: 3,14

Key	Value	
		Memory
		Memory
Mitchell	4,6692	Memory
Piwas	3,14	Memory
Douglas	42	Memory
		Memory
Euler	2,718	Memory

Creating a Hash Table

Empty hash table

```
PS> $hash = @{}
```

Create and populate hash table

```
PS> $Server = @{
        'HV-SRV-1' = '192.168.1.1'
        Memory = 64GB
        Serial = 'THX1138'
    }

PS> $Server

Name                           Value
----                           -----
HV-SRV-1                       192.168.1.1
Serial                         THX1138
Memory                         68719476736
```

Creating a Hash Table from a string variable

```
PS> $string = "
Msg1 = Hello
Msg2 = Enter an email alias
Msg3 = Enter a username
Msg4 = Enter a domain name
"

PS> ConvertFrom-StringData -StringData $string

Name                           Value
----                           -----
Msg4                           Enter a domain name
Msg3                           Enter a username
Msg2                           Enter an email alias
Msg1                           Hello
```

Create a hash table using Group-Object

Group-Object outputs a **Key : value** pair.

Needs **-AsString** parameter to convert **key** to a string instead of an object.

```
PS> $svcshash = Get-Service |
Group-Object Status -AsHashTable -AsString

PS> $svcshash

Name       Value
----       -----
Stopped    {AeLookupSvc, ALG, AppMgmt...}
Running    {AppIDSvc, Appinfo...}

PS> $svcshash.Stopped

Status    Name                  DisplayName
-------   ----                  -----------
Stopped   AeLookupSvc           Look up ser...
```

Demonstration

Creating Hash Tables

Accessing Hash Table Items

Accessing Hash Table Items

Access the item by key

Special characters allowed in key names

“Keys” and “values” properties available

Access Hash Tables Items - Examples

Display all items in hash table

```
PS> $Server


Name             Value
----             -----
HV-SRV-1         192.168.1.1
Serial           THX1138
Memory           68719476736
```

Return value using dot notation

```
PS> $Server.'HV-SRV-1'
192.168.1.1


PS> $Server.Serial
THX1138
```

Return value using “index” notation

```
PS> $Server["Serial"]
THX1138
```

Display All Hash Tables Keys and Values

Display all keys in hash table

```
PS> $Server.Keys
HV-SRV-1
Serial
Memory
```

Display all values in hash table

```
PS> $Server.Values
192.168.1.1
THX1138
68719476736
```

Note: Individual key lookup is fast, individual value lookup is slow on large tables

Demonstration

Accessing Hash Tables

Modifying Hash Table Items

Adding Items To a Hash Table

Add or set key and value using index notation

```
PS> $Server["CPUCores"] = 4
```

Add or set key and value using dot notation

```
PS> $Server.Drives = "C", "D", "E"
```

Add key and value using hash table ADD method

```
PS> $Server.Add("HotFixCount", (Get-HotFix -Computer
$Server["HV-SRV-1"]).count)
```

Note: Adding a key that already exists will cause an error

Removing Items From a Hash Table

Remove key

```
PS> $Server.Remove("HotFixCount")
```

Empty the Complete table

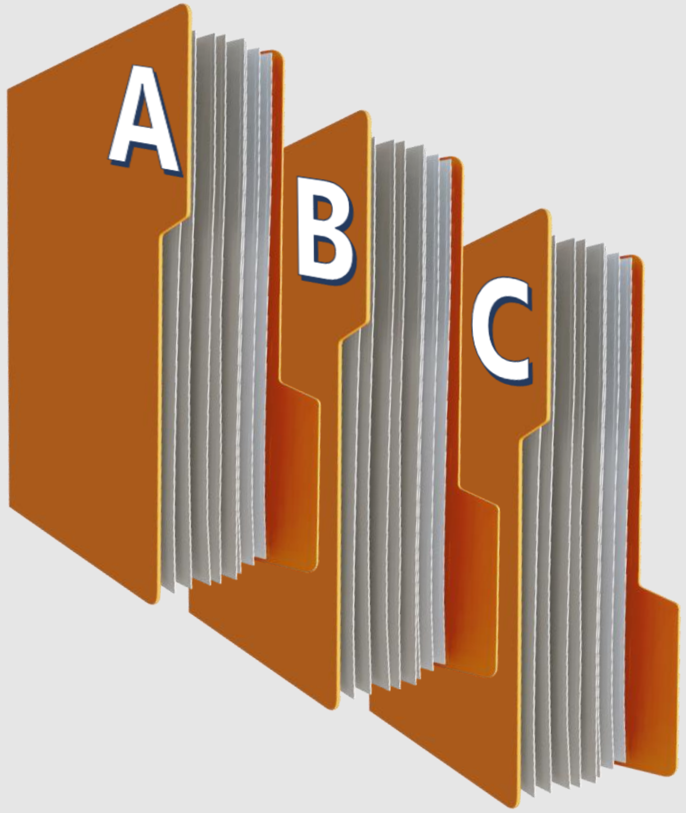
```
PS> $Server.Clear()
```

Demonstration

Modifying Hash Table Items

Sorting and Searching Hash Tables

Sorting Hash Tables



- Hash tables are intrinsically **unordered**
- It is **not** possible to sort a hash table as it's a **single** object
- **GetEnumerator()** reads the table one entry at a time, returning a **list** of objects on **key-value pairs**

Sorting Hash Tables - Example

```
PS> $Server.GetEnumerator() | Sort-Object -Property key

Name         Value
----         -----
CPUCores     4
Drives       {C, D, E}
HV-SRV-1     192.168.1.1
Memory       68719476736
```

Searching Inside Hash Tables

Searching on Key:

- Contains() or Containskey()
- Constant lookup time
- Case insensitive

Searching on Value:

- ContainsValue()
- Variable lookup time
- Case sensitive

Searching Hash Tables - Example

```
PS> $hash = @{"John"=23342;"Linda"=54345;"James"=65467}

PS> $hash.ContainsKey("Linda")    #Fast hashed key search
True

PS> $hash.ContainsValue(19)       #Slow non-hashed search
False

PS> $hash.ContainsValue(65467)
True
```

Demonstration

Searching Hash Table Items

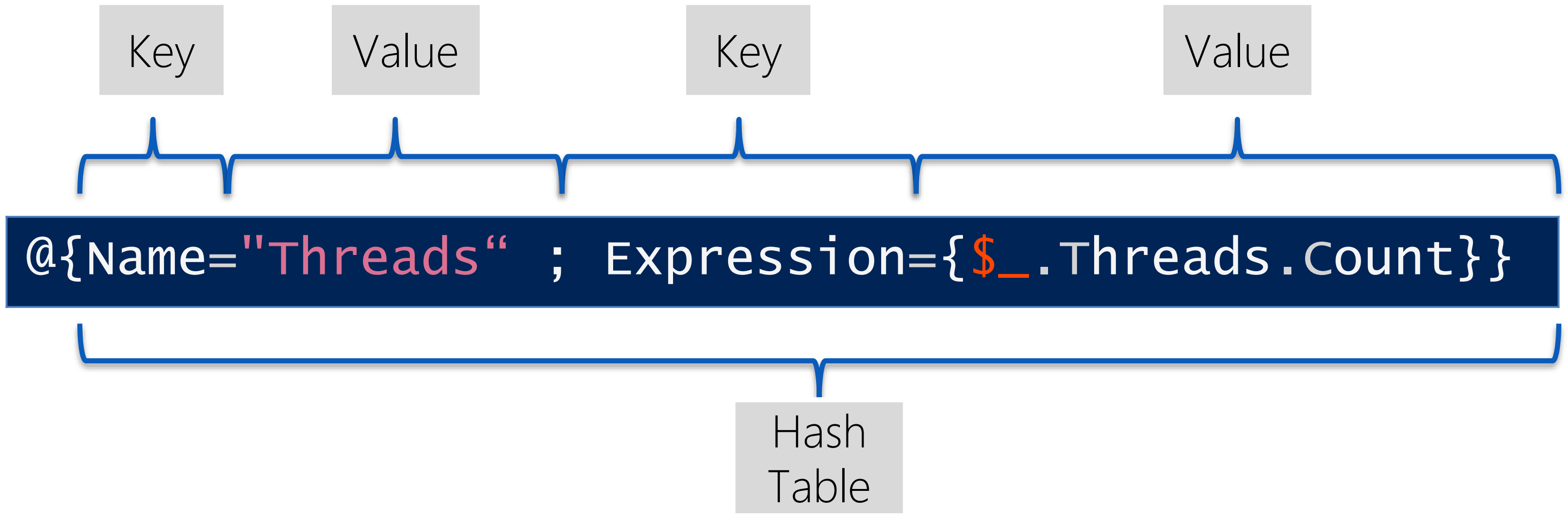
Hash Tables – Practical Use Cases

Calculated Properties – Simple Example

- Most display commands support **calculated** properties
- **Calculated** properties can use key:value pair of a hash table

```
PS> Get-Process | FT Name,@{Name = "Threads"; Expression = {$_.threads.count}}

Name                                                          Threads
----                                                          -------
aesm_service                                                        2
ApplicationFrameHost                                                5
Calculator                                                         28
```



```
@{Name="Threads“ ; Expression={$_.Threads.Count}}
```

Custom Object Creation

Use a hash table to create a PSObject that can be added to an array directly

```
$ping = Test-Connection -computername "dns.google“ -count 4
$pingmeasure = $ping |
Measure-Object -Property "ResponseTime“ -Maximum -Minimum -Average

$properties = @{
    'Name' = $object
    'pingtime' = $pingmeasure.average
    'pingcount' = $pingmeasure.count
    'pingMaxmum' = $pingmeasure.Maximum
    'pingMinimum' = $pingmeasure.Minimum
}

[array]$result += New-Object –TypeName PSObject –Properties
$properties
```

Splatting

A technique for passing arguments to commands

```
Get-ChildItem -Path c:\windows -File | Measure-Object -Average -Sum
-Maximum -Minimum -Property Length
```

Versus

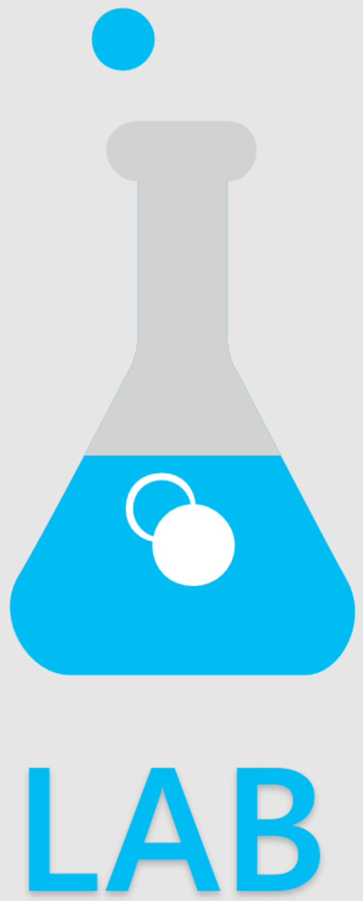
```
$moparams = @{
    Average = $true
    Maximum = $true
    Sum = $true
    Minimum = $true
    Property = 'length'
}
$gciparams = @{
    Path = 'c:\windows'
    File = $true
}
Get-ChildItem @gciparams | Measure-Object @moparams
```

Demonstration

Hash Table Use Cases

Lab 9:
Hash Tables

60 minutes



Microsoft