

# Understanding Fabric Capacities

Benni De Jagere



Slides



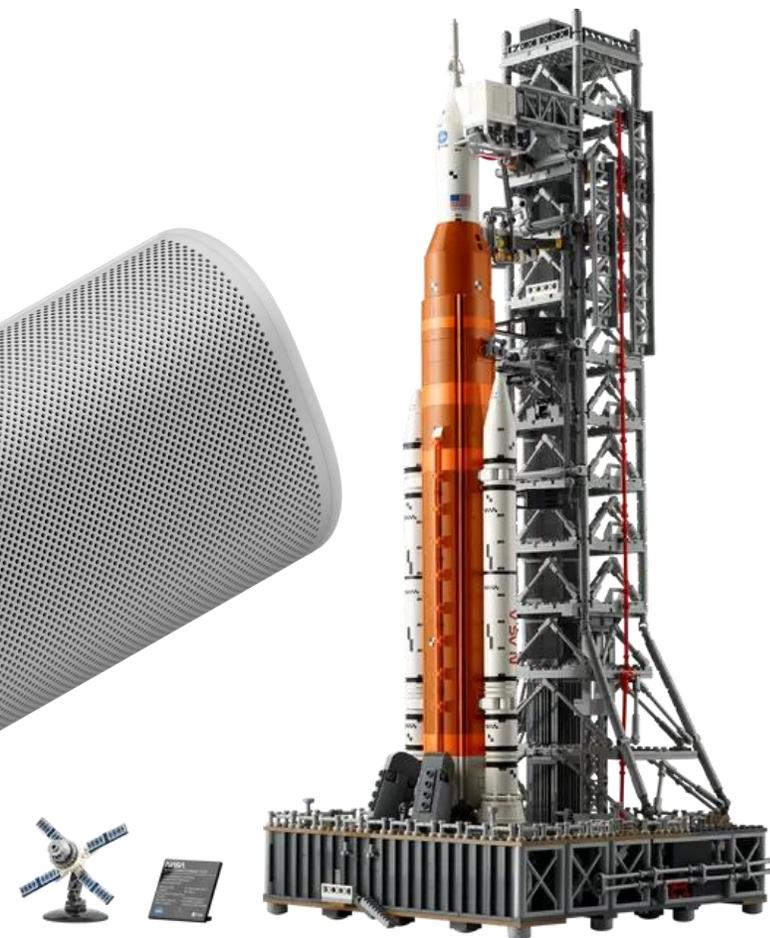
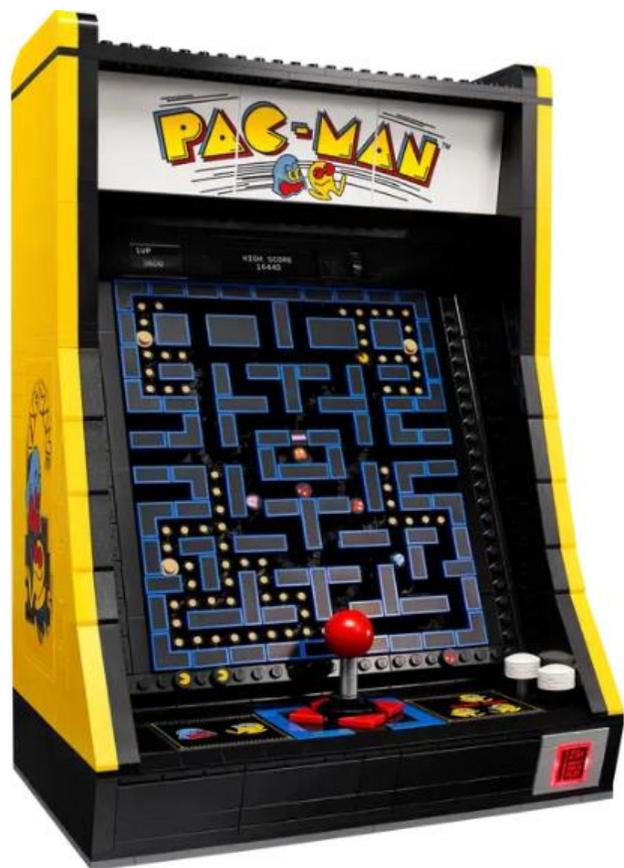
## Premium sponsors



## Standard sponsors



# Raffle Prizes





# Benni De Jagere

Senior Program Manager | Fabric Customer Advisory Team ( FabricCAT )



Fabric CAT



.be Member



@BenniDeJagere



/bennidejagere

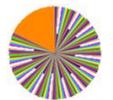


sessionize

/bennidejagere



/bennidejagere



#SayNoToPieCharts





# Fabric Capacities Introduction



# Microsoft Fabric



Data  
Factory



Synapse Data  
Engineering



Synapse Data  
Science



Synapse Data  
Warehouse



Synapse Real  
Time Analytics



Power BI



Data  
Activator

AI Assisted

Shared Workspaces

Universal Compute Capacities

One Security

OneLake

Intelligent data foundation

## Single...

Onboarding and trials

Sign-on

Navigation model

UX model

Workspace organization

Collaboration experience

Data Lake

Storage format

Data copy for all engines

Security model

CI/CD

Monitoring hub

Governance & Capacity Metrics

Data Hub

# Capacities are to Fabric what CPUs are to PCs

## Personal Computing

When you purchase a PC you choose the number of CPU cores. The more CPU cores the more load the PC can handle.



The CPU cores are dynamically shared across all applications with no need to pre-allocate by app.



The total consumption of the CPU across all the apps cannot exceed the number of cores. CPU overload causes a slowdown.

Name	Status	48% CPU	82% Memory	1% Disk
<b>Apps (10)</b>				
> Task Manager		2.6%	65.0 MB	0 MB/s
> Snipping Tool (2)		0%	28.1 MB	0 MB/s
> Settings		0%	45.3 MB	0 MB/s
> Microsoft Word (2)		0.5%	131.3 MB	0 MB/s
> Microsoft Teams (9)		4.5%	800.3 MB	0.1 MB/s
> Microsoft PowerPoint (2)		1.5%	300.5 MB	0.1 MB/s
> Microsoft Outlook (13)		0.9%	414.0 MB	0.1 MB/s
> Microsoft OneNote		0%	85.2 MB	0 MB/s
> Microsoft Excel (2)		0.8%	93.5 MB	0 MB/s
> Microsoft Edge (19)		0%	1,628.3 MB	0.1 MB/s

## Fabric Capacities

In Fabric, you provision a Capacity with a number of "capacity units". The more capacity units provisioned, the more compute load handled.

Unlike the PC, capacity units can be scaled up or down as needed.

The capacity units are dynamically shared across all the Fabric workloads, with no pre-allocation necessary.

A single capacity can simultaneously drive BI, DW, Spark, ML and every other compute engine in Fabric

The total consumption of the capacity across all the workloads cannot exceed the capacity units provisioned.

Overloading the capacity will throttle it (slow down).

Auto scale can dynamically increase the available compute units avoiding the slowdown.

# Capacities are a shared resource

## Shared across workloads

A single capacity is providing the compute power for all Fabric workloads.

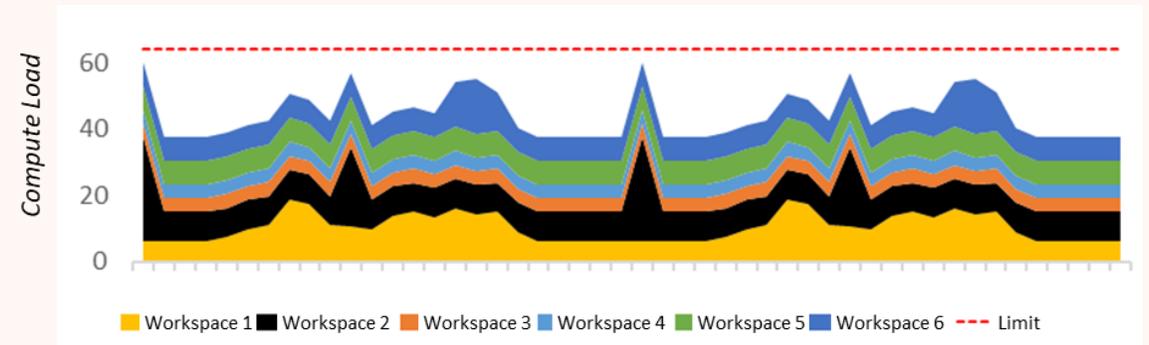
There is no need to allocate compute for each workload separately.



## Shared Across Projects

A single capacity typically supports dozens of separate projects simultaneously, each managed in its own workspace.

It is rare to have a capacity dedicated to a single project



## Shared across users

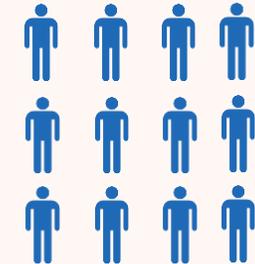
For each project, many developers will share a workspace where collaborative development and consumption at scale is managed.

Each creator can provision any artifact and run any job without the need for any pre-approval or planning

Developers/Creators



Consumers

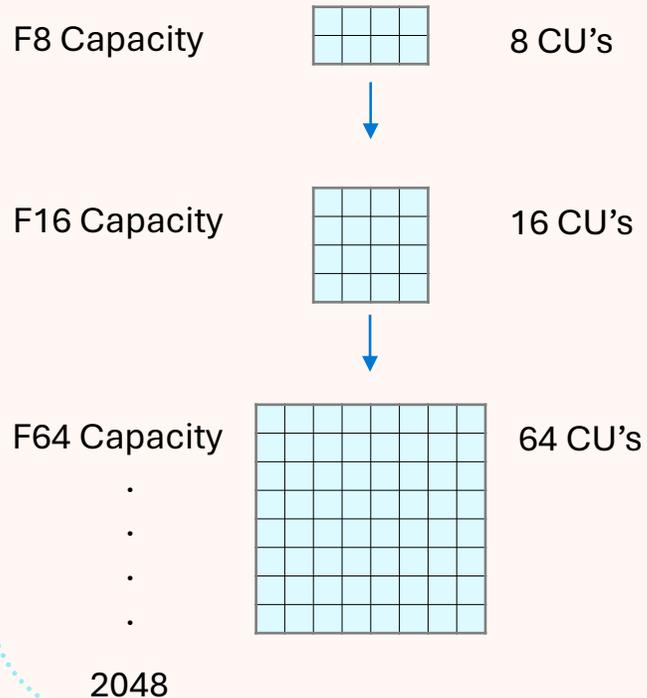


# Capacities are flexible building blocks for growth

Capacities can be configured in endless ways to meet scale, usage and governance requirements while tuning to minimize TCO and performance goals

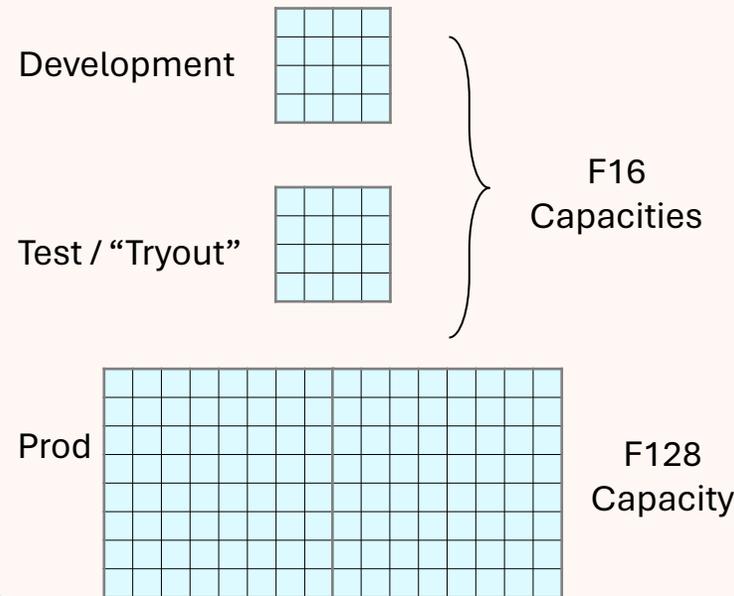
## Scale Vertically

Increased capacity size provides more throughput



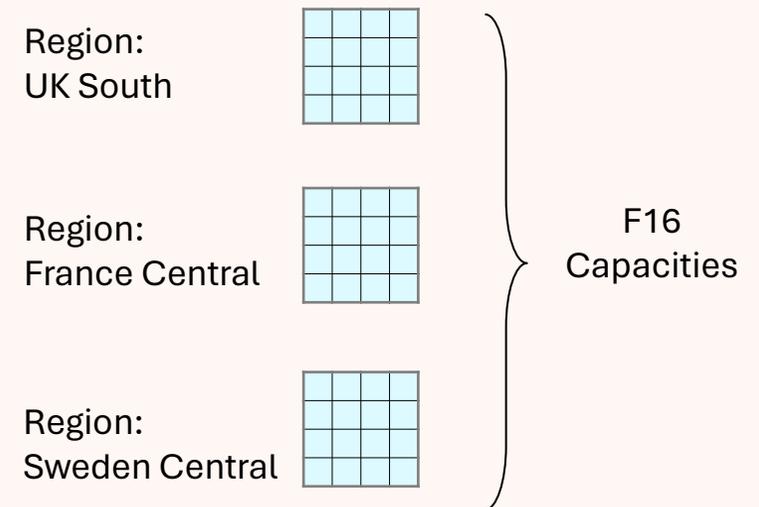
## Scale Horizontally

Scale horizontally using the benefits of modular design for hardened isolation and governance



## Regional Availability

Use different capacities for different regions to support GDPR / Data residency requirements



# Provisioning and Deploying Capacities

## Purchased in Azure

- **Purchased** either as a PAYG or RI resource
- **Provisioned with a certain amount of compute** units, analogous to CPU cores.
- The **more capacity units are provisioned, the more load** the capacity can support
  - Multiply SKU size by 30s to match platform evaluation in metrics app
- Capacities are **priced at a fixed hourly rate**, based on capacity units provisioned
- The RI commitment (1-year reserved instance) enjoys a **41% discount**

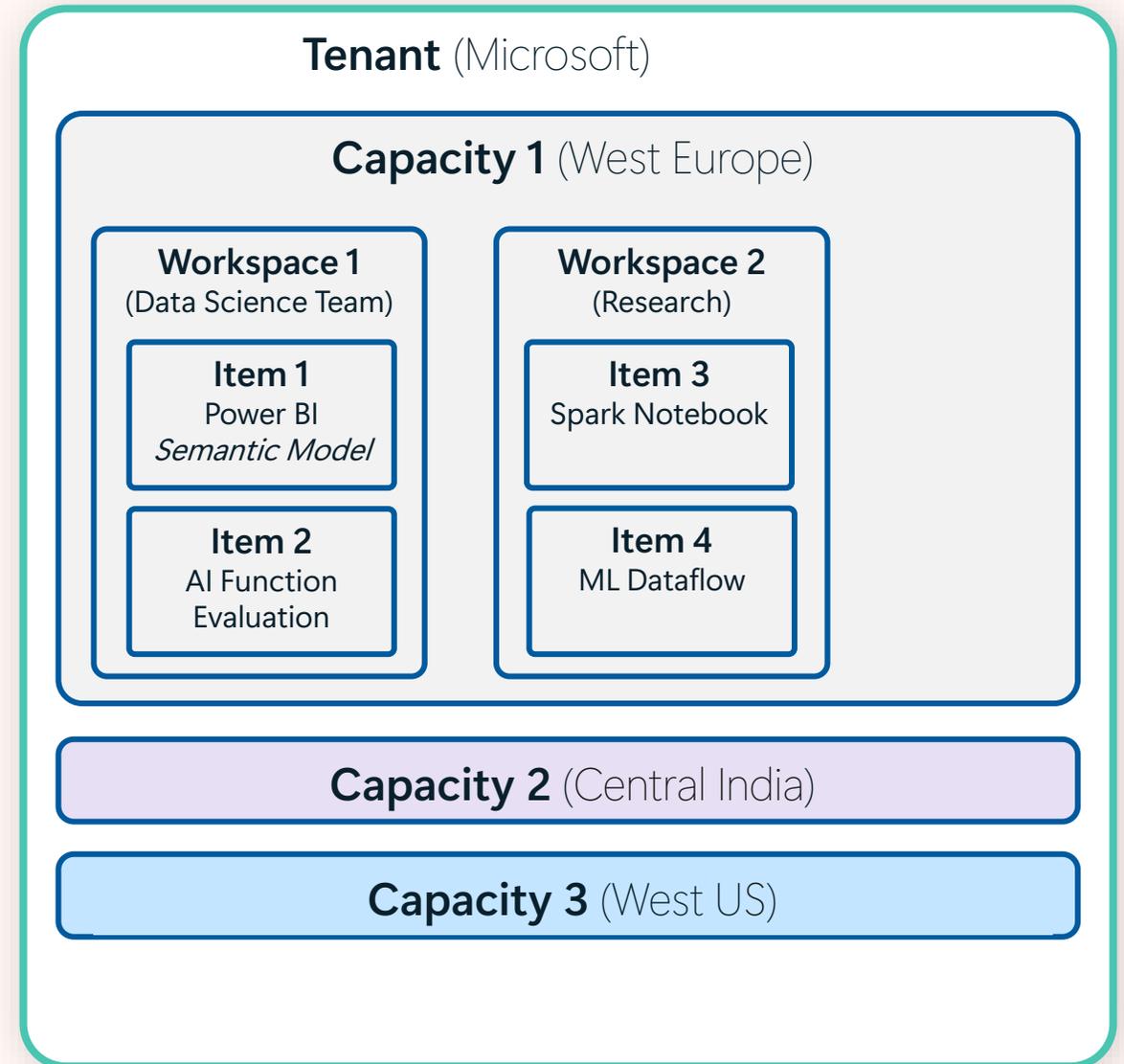
## Universal Compute Capacities SKU Sizing

SKU	Capacity Units (CU)	CU's (per 30s)	Power BI SKU	Power BI V-cores
F2	2	60	-	0.25
F4	4	120	-	0.5
F8	8	240	A1	1
F16	16	480	A2	2
F32	32	960	A3	4
F64	64	1920	P1	8
F128	128	3840	P2	16
F256	256	7680	P3	32
F512	512	15360	P4	64
F1024	1024	30720	P5	128
F2048	2048	61440	-	256

# Provisioning and Deploying Capacities

## Deployed to Regions

- Each capacity **resides in a specific region of the buyers' choice** where both the data & compute reside
- **Workspaces are assigned to a capacity** that provides the compute and storage for all the workspace artifacts
- Multiple capacities can be purchased, deployed and managed by **different owners** residing in a single tenant allowing each business unit to pay for their own consumption





# Bursting and Smoothing

# Smoothing intro and benefits

## Load stabilization

Smoothing helps capacities self-stabilize by flattening large spikey loads into a smooth load profile, eliminating temporal spikes

## Eliminates Scheduling contention

Large/scheduled Jobs usage (not execution) are smoothed over 24 hours, eliminating the need to decide the timing and order of job execution

## Bad actor protection

Interactive operations smoothed over several minutes, preventing a single user with a very demanding query from hogging the entire capacity



# What is Bursting?

## Job acceleration

Bursting provides extra compute resources to jobs and queries to accelerate their completion

## Go beyond

The extra resources of bursting allow jobs to **utilize far more resources than "face value"**

Instead of running a job on 64 CU and completing in 60 seconds, bursting could use 256 CUs to complete the job in 15 seconds.

Same amount of work, just completed faster

## No hassle, No overload

**Bursting is automatic** when the system reasons it can accelerate the job by applying extra resources. No settings are required.

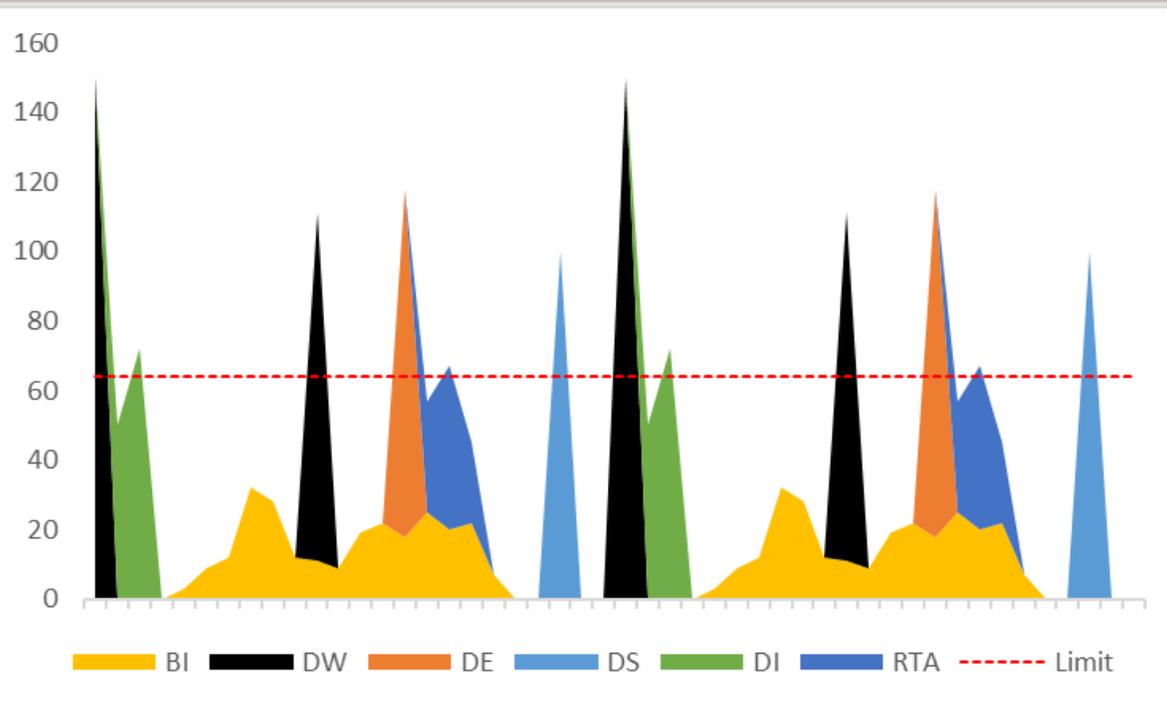
**Bursting prevents an overload** as the *smoothing* mechanism will always flatten the resource burst

# Bursting and smoothing | before and after

Looking at an example of a 64 CU capacity, running multiple workloads over a couple of days...

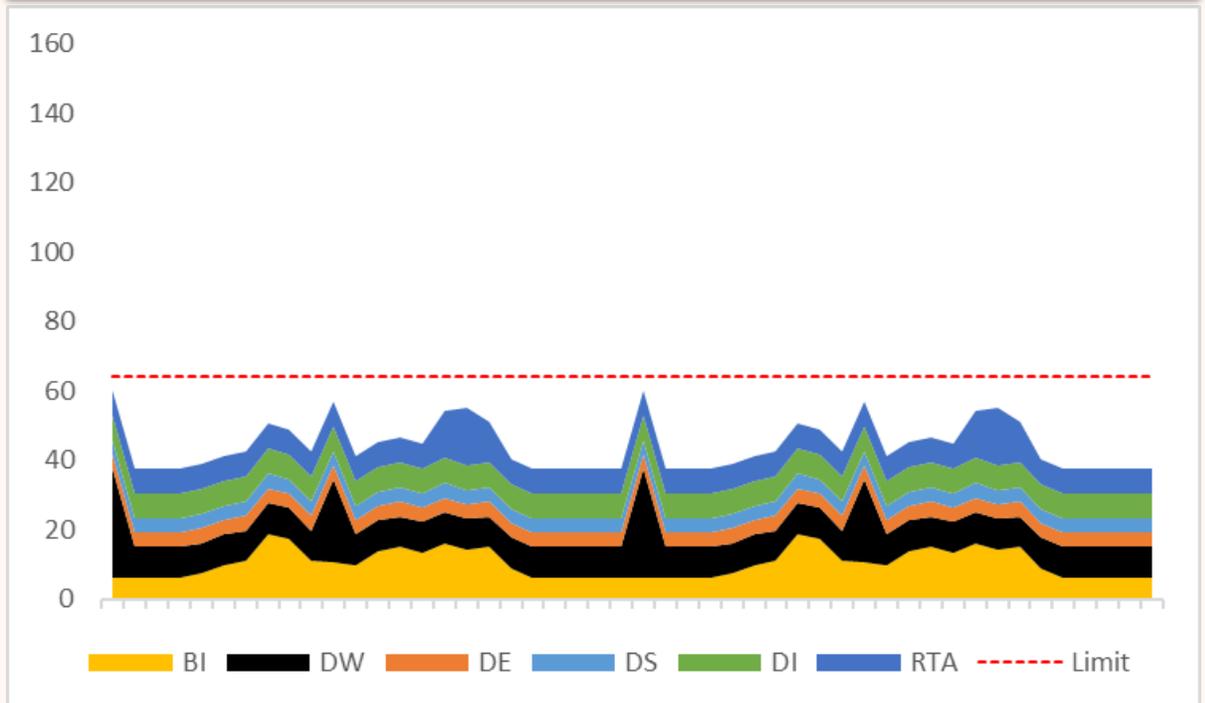
## Before Smoothing

- Actual load as executed on the capacity before smoothing
- *Bursting* accelerates jobs execution by resource boosting
- The capacity could be overloaded 25% of the time
- Some of the overloads are more than 2x the limit
- There are periods of no/low usage

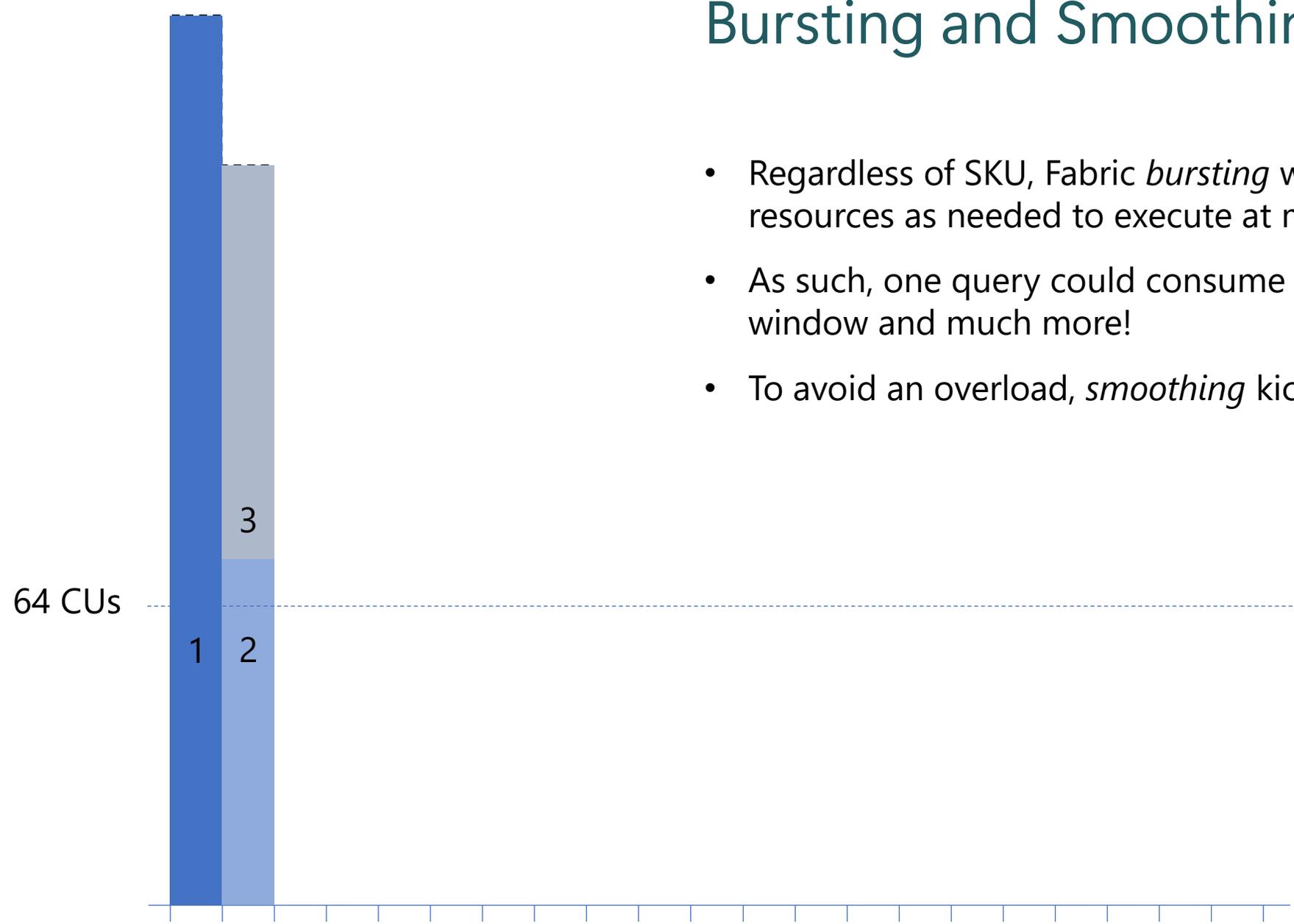


## After Smoothing

- Shows the reported load (not runtime execution) against the capacity limits
- There is NO overload, and consumption is more stable
- The smoothing of usage fills in gaps



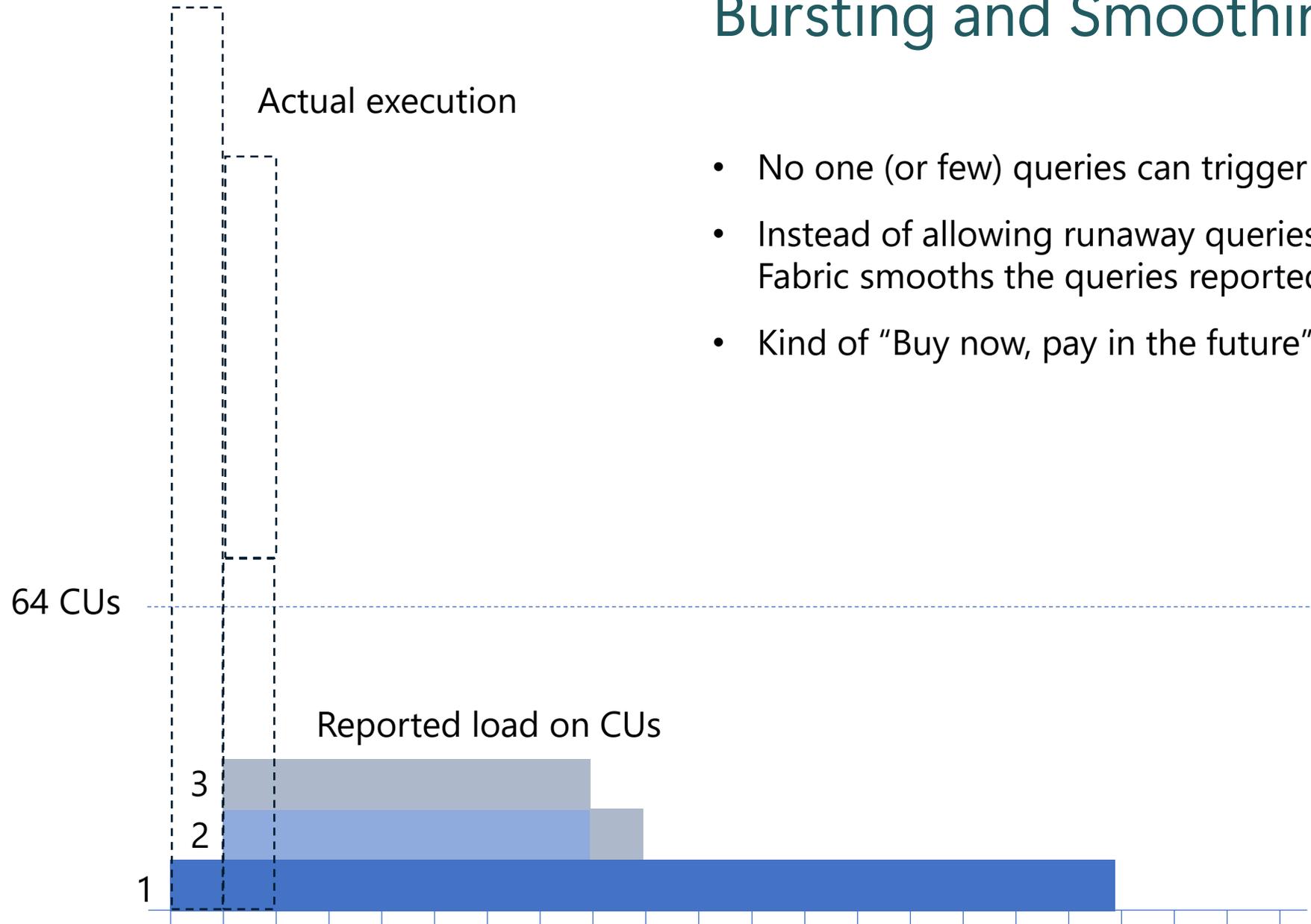
Jobs Executed



# Bursting and Smoothing

- Regardless of SKU, Fabric *bursting* will automatically allocate resources as needed to execute at maximum performance
- As such, one query could consume all the quota of a single time window and much more!
- To avoid an overload, *smoothing* kicks in

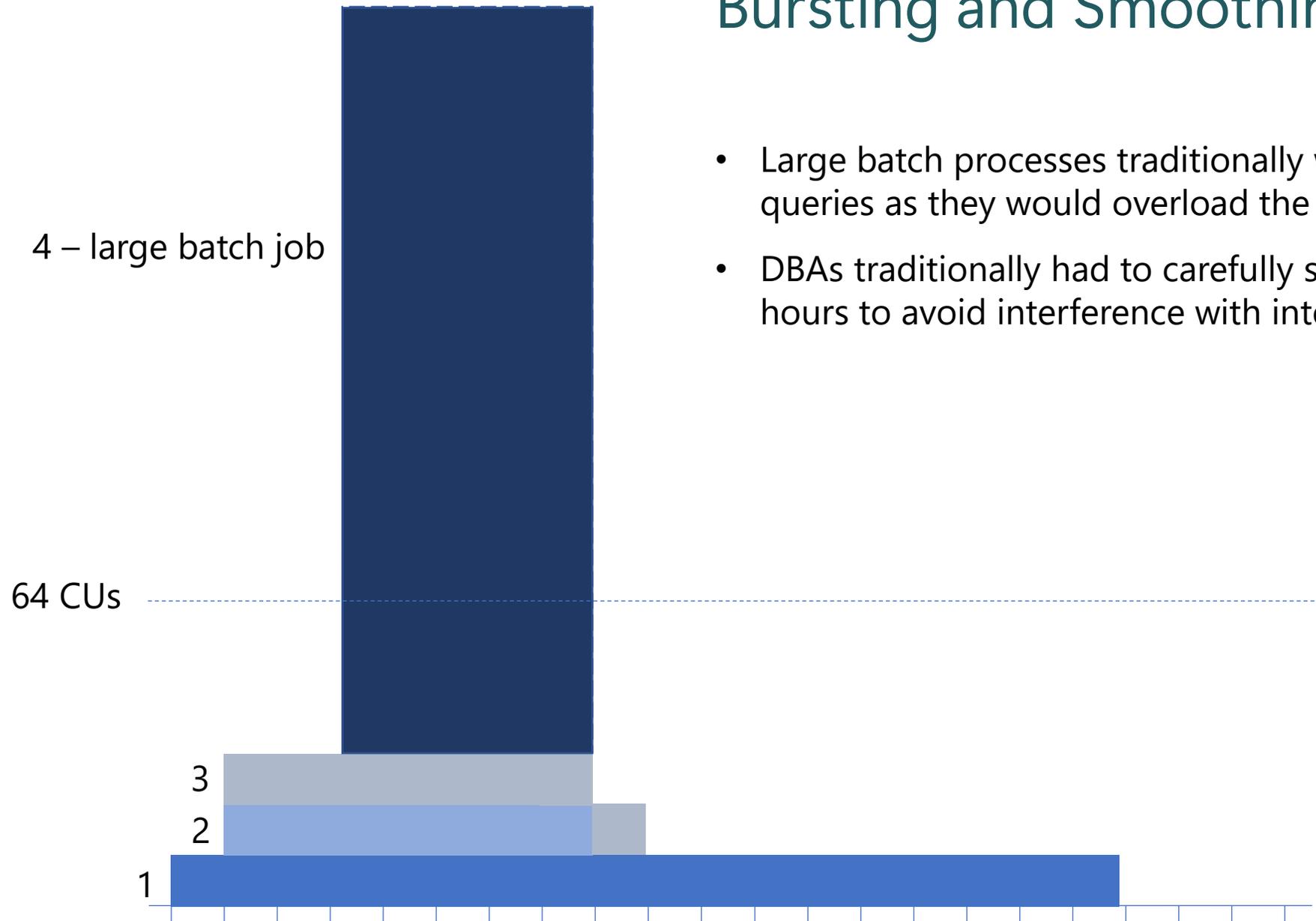
# Bursting and Smoothing



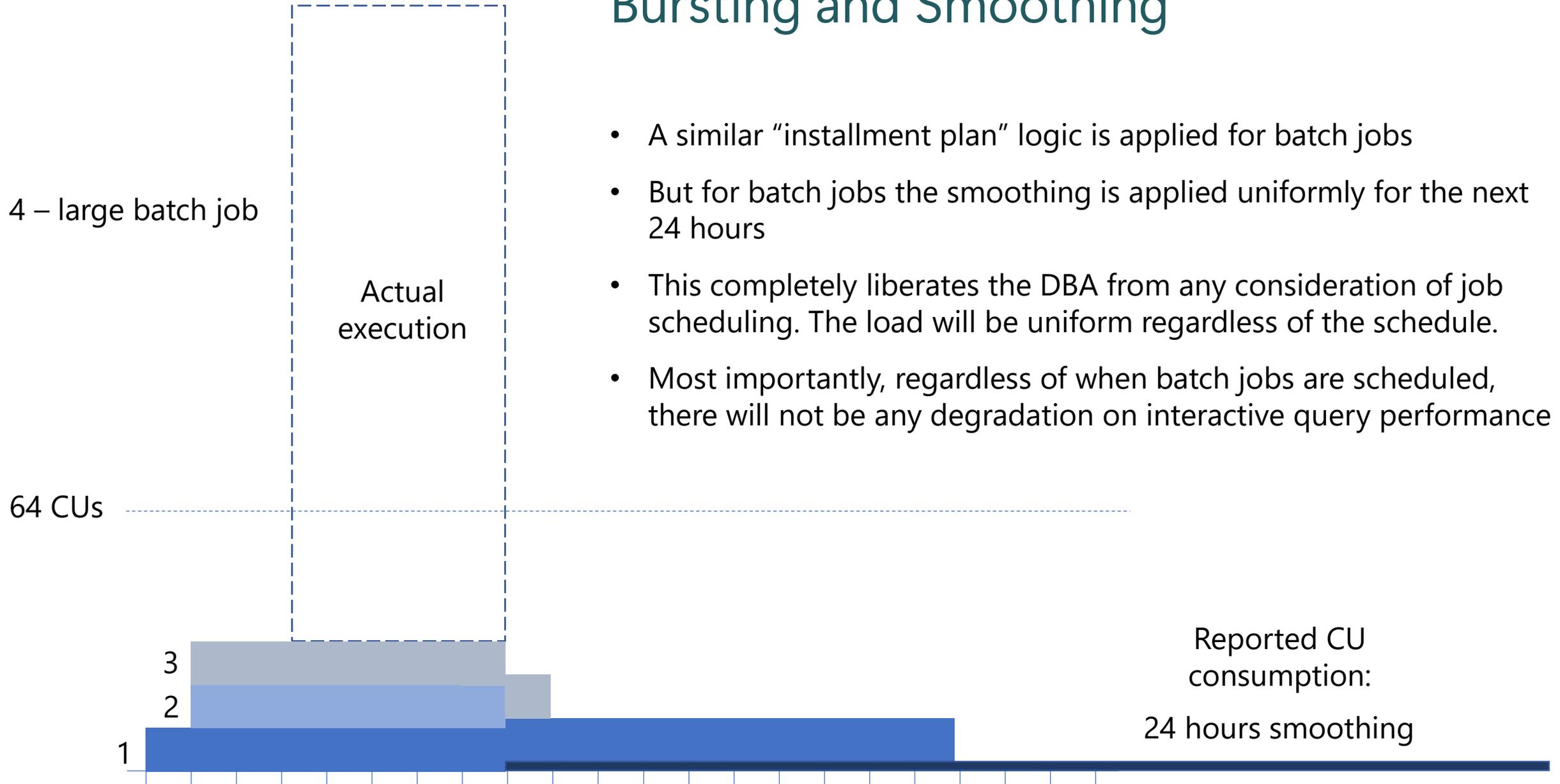
- No one (or few) queries can trigger an overload
- Instead of allowing runaway queries to create a local overload, Fabric smooths the queries reported usage to future time windows
- Kind of "Buy now, pay in the future" installment plan

# Bursting and Smoothing

- Large batch processes traditionally were a threat to interactive queries as they would overload the compute resource
- DBAs traditionally had to carefully schedule these jobs to off-hours to avoid interference with interactive user experiences



# Bursting and Smoothing



- A similar “installment plan” logic is applied for batch jobs
- But for batch jobs the smoothing is applied uniformly for the next 24 hours
- This completely liberates the DBA from any consideration of job scheduling. The load will be uniform regardless of the schedule.
- Most importantly, regardless of when batch jobs are scheduled, there will not be any degradation on interactive query performance

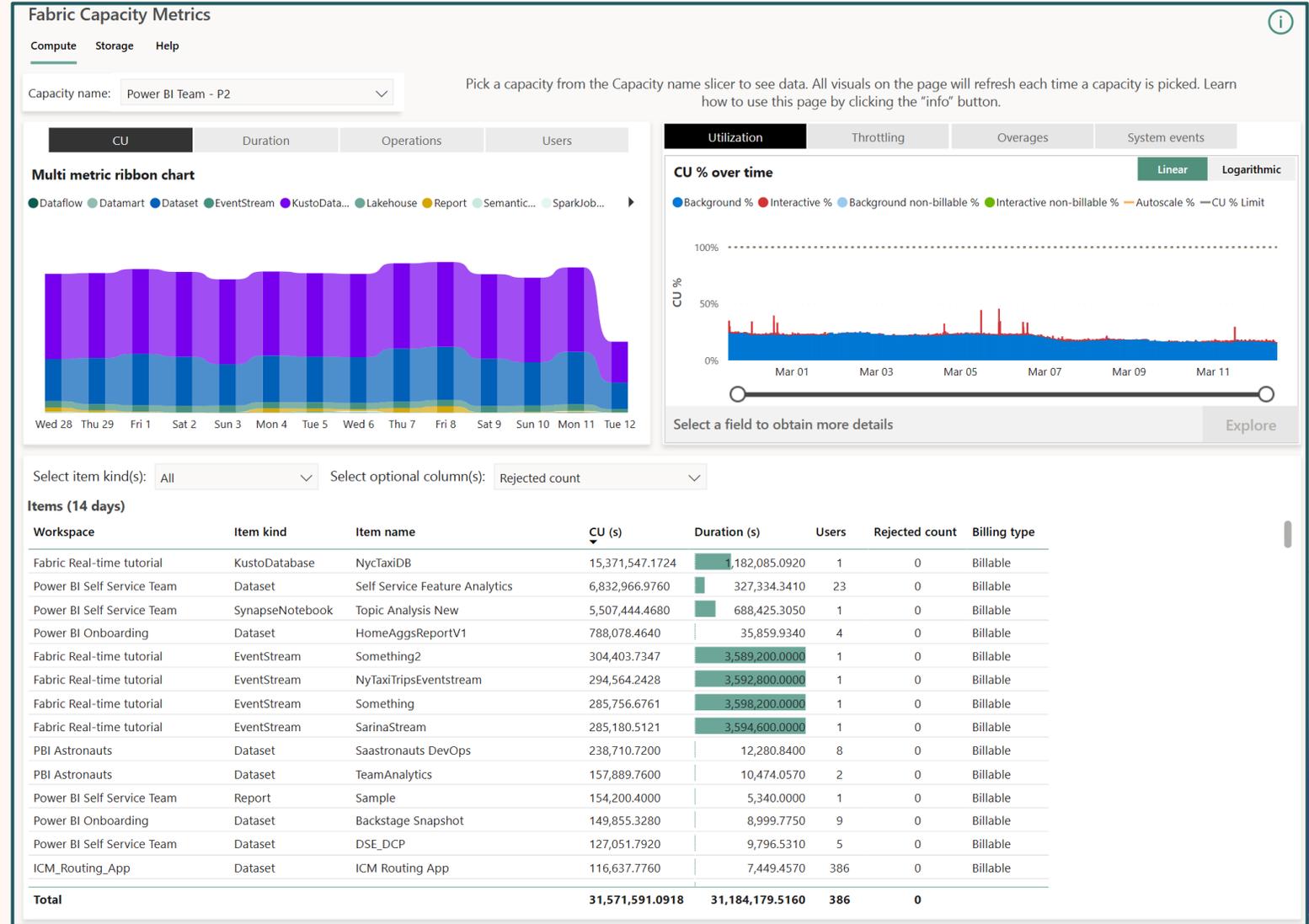


# Monitoring with Capacity Metrics

# Capacity Metrics

## Monitor Capacities and Plan capacity scale-up with confidence

- Tenant wide visibility into capacity usage for all Fabric experiences
- Identify resource usage trends and their impact to autoscale & throttling
- View preview workload usage alongside production workloads to make data-driven capacity sizing decisions

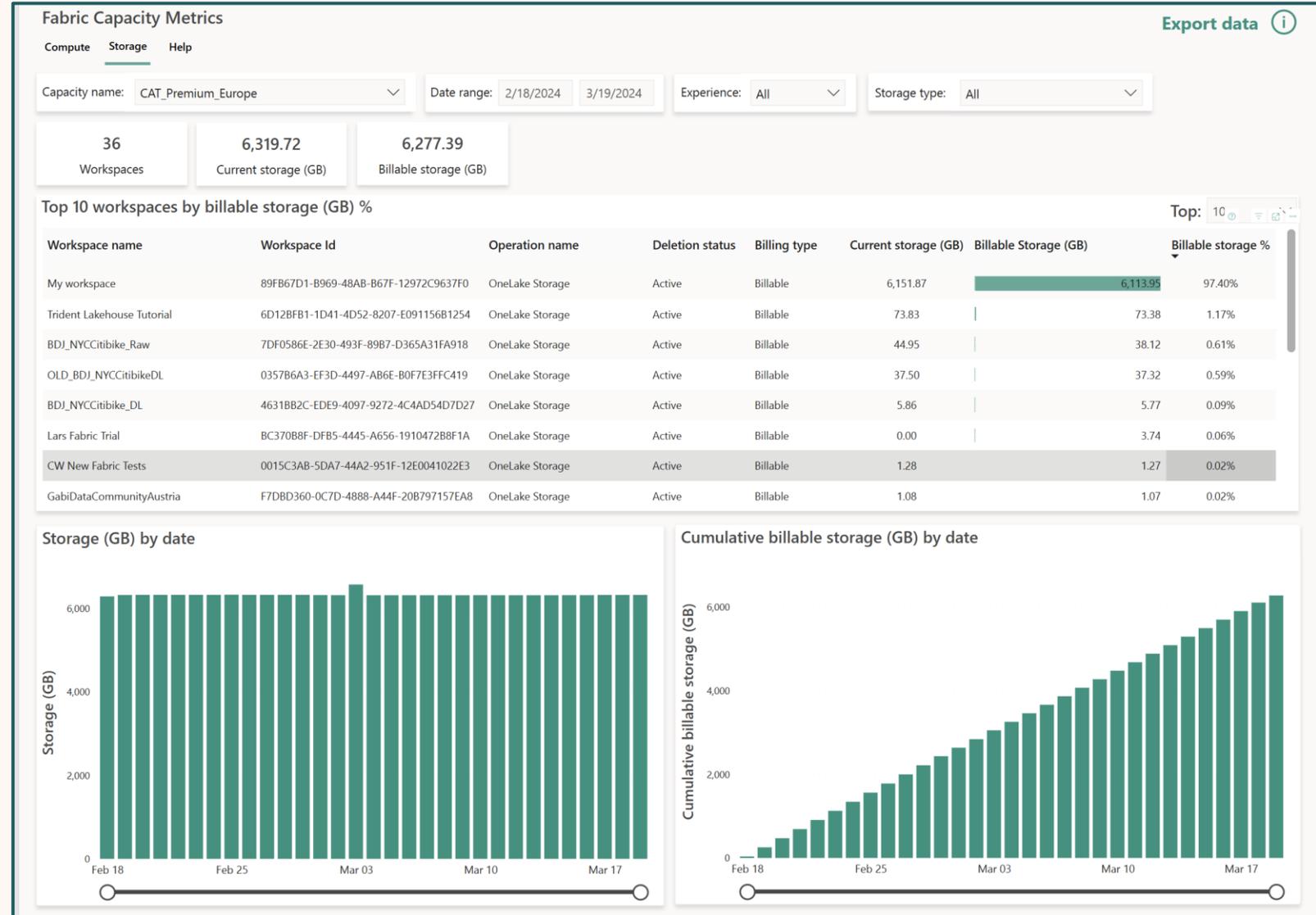


# Capacity Metrics

## Monitor OneLake consumption

Measure the trends of workspace storage consumption against capacity limits, by day or hour

Reconcile costs with internal chargeback processes



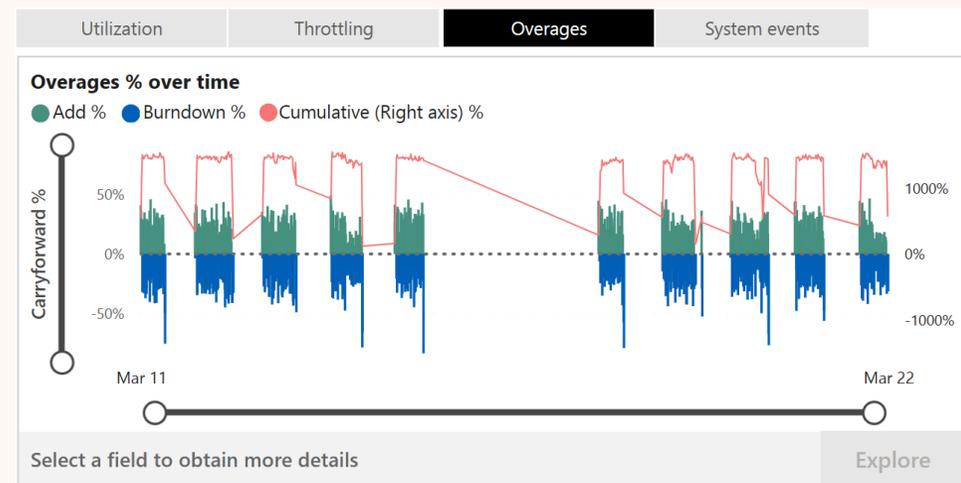
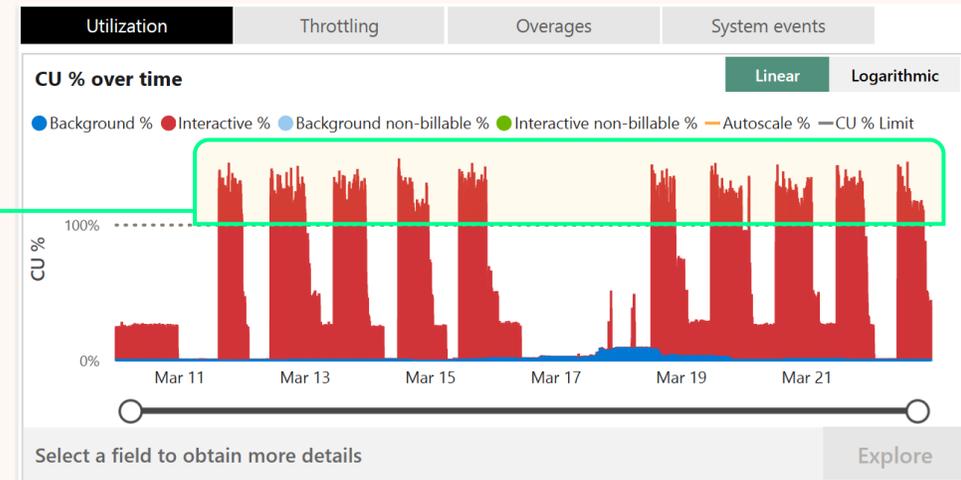


# Capacity Throttling Policies

# Throttling intro

- Throttling is the platform policy for managing consumption that exceeds throughput is provided by SKU choice
- When workloads exceed the throughput of a capacity a cumulative debt is tracked to be burned down
- Cumulative debt is used to determine throttling policies and is burned down when resources are free

Overage Operation	Description
 Overages - Added	<ul style="list-style-type: none"> <li>• Timepoint when job requests exceed the throughput of a capacity, overages are added to the cumulative buffer to burn down.</li> <li>• This graph simplifies identification of the optimal timepoint to load timepoint drill to analyze the user operations that contributed to an overage.</li> </ul>
 Overages - Burndown	<ul style="list-style-type: none"> <li>• Overages being reconciled when future capacity is free to burn down</li> </ul>
 Overages - Cumulative	<ul style="list-style-type: none"> <li>• The total amount of queued work on the capacity to be burned down in the future when the capacity is not fully utilized</li> </ul>



# Capacity throttling evolution for Fabric

- For Fabric, throttling policies were refined to deliver multiple benefits
  - **Reduced throttling** for capacities that only experience occasional spikes
  - **Added overage protection** – rejection policies prevent overloaded capacities from irrecoverable overload
  - **Optimizations for long-running jobs:** We're optimizing the platform for long-running jobs, so if a job exceeds capacity limits, it will run to completion and the overage will be burned down against future capacity



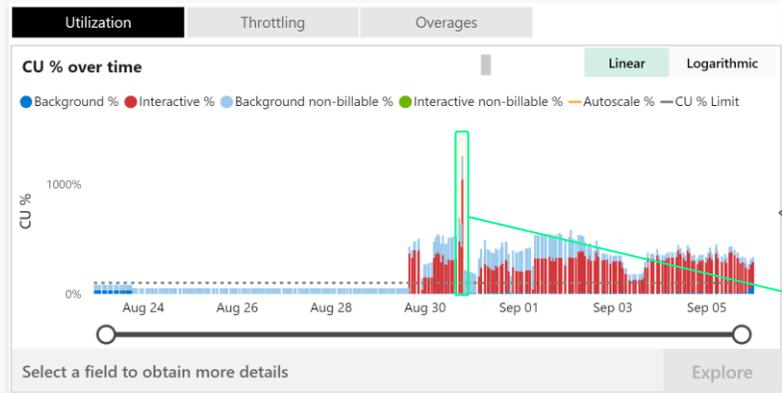
Smoothed Capacity - Future Use	Platform Policy	Customer Impact
$\leq 10m$	Overage Protection	Jobs can consume 10 minutes of future capacity use without throttling
$> 10m \rightarrow \leq 60m$	Interactive Delay	User requested interactive type jobs will be throttled
$> 60m \rightarrow \leq 24h$	Interactive Rejection	User requested interactive type jobs will be rejected
$> 24h$	Background Rejection	User Scheduled background jobs will be rejected from execution



# Capacity Planning with Capacity Metrics

# Capacity planning case study - measurement

Start with a test or trial capacity to evaluate the load of specific Fabric Experiences i.e., Power BI Datasets, Spark Notebooks or a Datawarehouse



If usage is above the current capacity limits, choose the desired utilization rate to accommodate via capacity scale up

Fabric Capacity Metrics

Overview Help

8/30/2023 7:26:00 PM Timepoint

CU % Limit 1,249.04%

CU % Limit 100.00%

Autoscale % 100.00%

CU(s) 749.42

Limit in CU(s) 60

287 Interactive operations

100K Background operations

F2 SKU

60 CU (s)

Item	Operation	Start	End	Duration (s)	Total CU (s)	Timepoint CU (s)	Throttling (s)	% of Base Capacity	Billing type
[...]	Query	8/30/2023 7:11:25...	8/30/2023 7:11:25...	10	900	30.00	0	50.00%	Billable
[...]	Query	8/30/2023 7:23:45...	8/30/2023 7:23:45...	14	1,110	30.00	0	50.00%	Billable
[...]	Query	8/30/2023 7:19:30...	8/30/2023 7:20:03...	32	509	29.98	0	49.97%	Billable
[...]	Query	8/30/2023 7:18:05...	8/30/2023 7:18:16...	10	958	29.95	0	49.91%	Billable
[...]	Query	8/30/2023 7:12:46...	8/30/2023 7:12:58...	12	986	29.90	0	49.84%	Billable
[...]	Query	8/30/2023 7:20:06...	8/30/2023 7:20:16...	10	980	29.71	0	49.52%	Billable
[...]	Query	8/30/2023 7:17:28...	8/30/2023 7:18:02...	33	531	29.54	0	49.24%	Billable
[...]	Query	8/30/2023 7:10:32...	8/30/2023 7:10:50...	17	944	29.50	0	49.17%	Billable
[...]	Query	8/30/2023 7:16:41...	8/30/2023 7:17:14...	33	530	29.49	0	49.14%	Billable
[...]	Query	8/30/2023 7:17:15...	8/30/2023 7:17:26...	11	1,000	29.43	0	49.04%	Billable
<b>Tot...</b>				<b>13,812</b>	<b>13,894</b>	<b>619.94</b>	<b>0</b>	<b>1033.23%</b>	

Load Capacity Metrics timepoint drill to analyze :

- Total CU's consumed : 749 CU(s)
- Capacity Size : (F2)
- CU(s) available on your capacity : 60 CU(s)

# Capacity planning case study – SKU selection

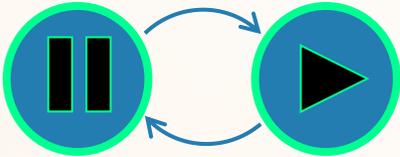
## Universal Compute Capacities SKU Sizing

SKU	Capacity Units (CU)	CU's (per 30s)	Power BI SKU	Power BI V-cores
F2	2	60	-	0.25
F4	4	120	-	0.5
F8	8	240	A1	1
F16	16	480	A2	2
F32	32	960	A3	4
F64	64	1920	P1	8
F128	128	3840	P2	16
F256	256	7680	P3	32
F512	512	15360	P4	64
F1024	1024	30720	P5	128
F2048	2048	61440	-	256

To accommodate a **749 CU(s) load** the admin can purchase an F32 capacity providing 960 CU(s) of throughput



# Pausing and Resuming Capacities



# Introduction to Pausing and Resuming Capacities

## Overview and Benefits

**Pause and Resume** lets you manage compute costs on F SKU capacities by suspending the execution of all workloads running on the capacity

- When a capacity administrator pauses a capacity:

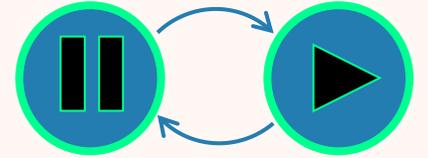
Workloads stop  
execution

New requests are  
not run

Smoothed usage  
will be reconciled

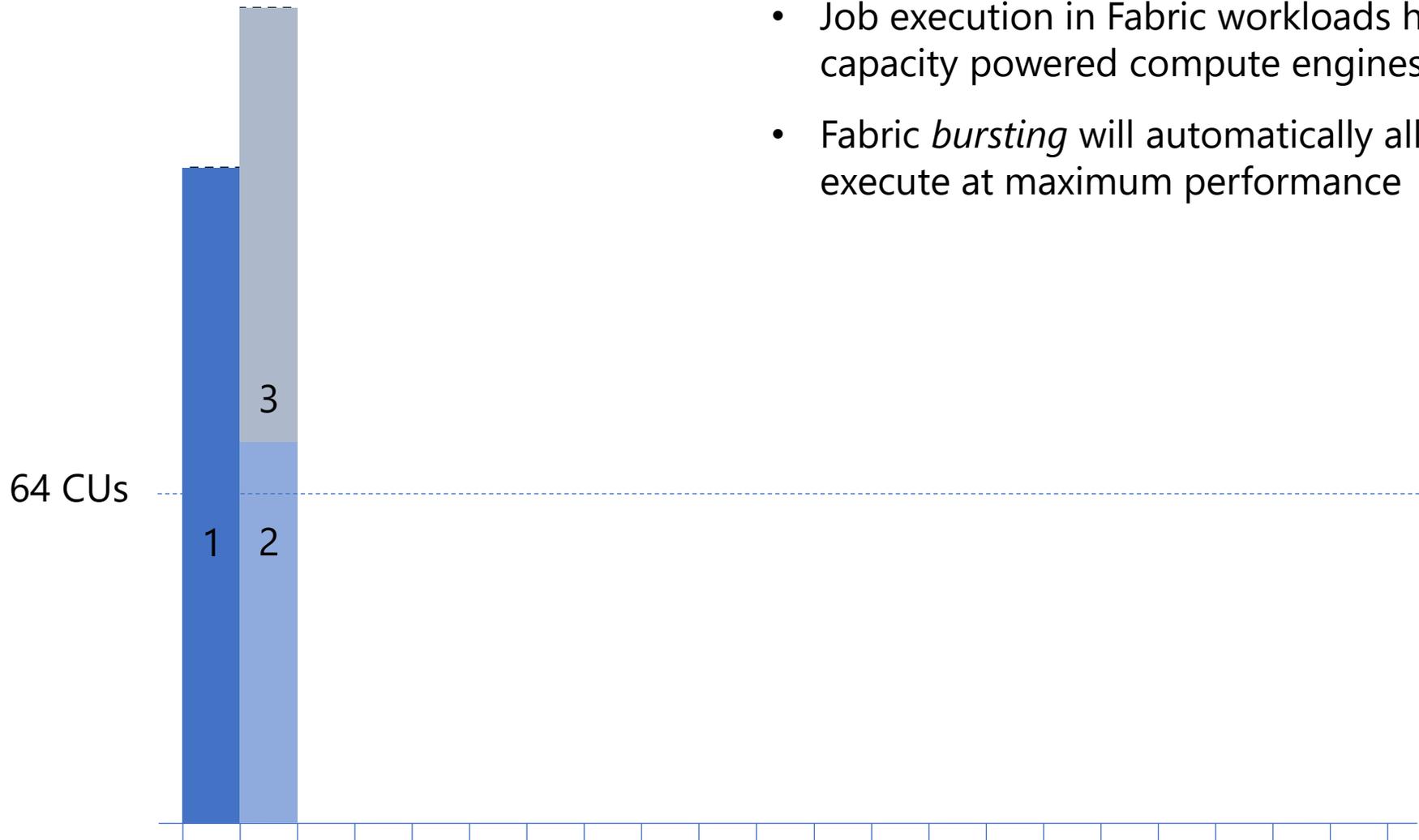
(details in the next demo)

**Note:** OneLake storage will remain active and billable while a capacity is paused



# Bursting and Smoothing

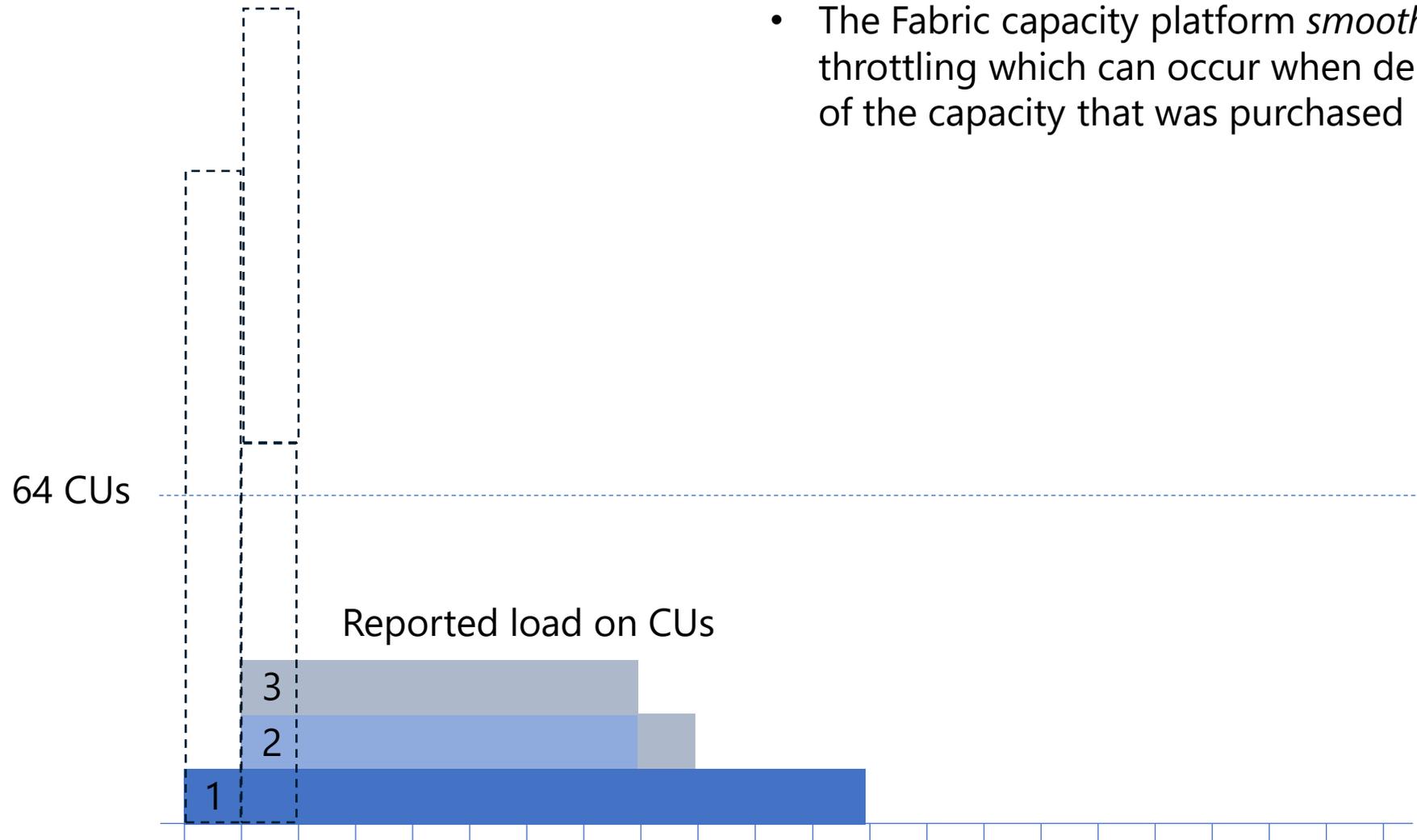
Jobs Executed



- Job execution in Fabric workloads happens on-demand via capacity powered compute engines
- Fabric *bursting* will automatically allocate resources as needed to execute at maximum performance

# Bursting and Smoothing

Actual execution



- The Fabric capacity platform *smooths* usage out to reduce throttling which can occur when demand exceeds the throughput of the capacity that was purchased

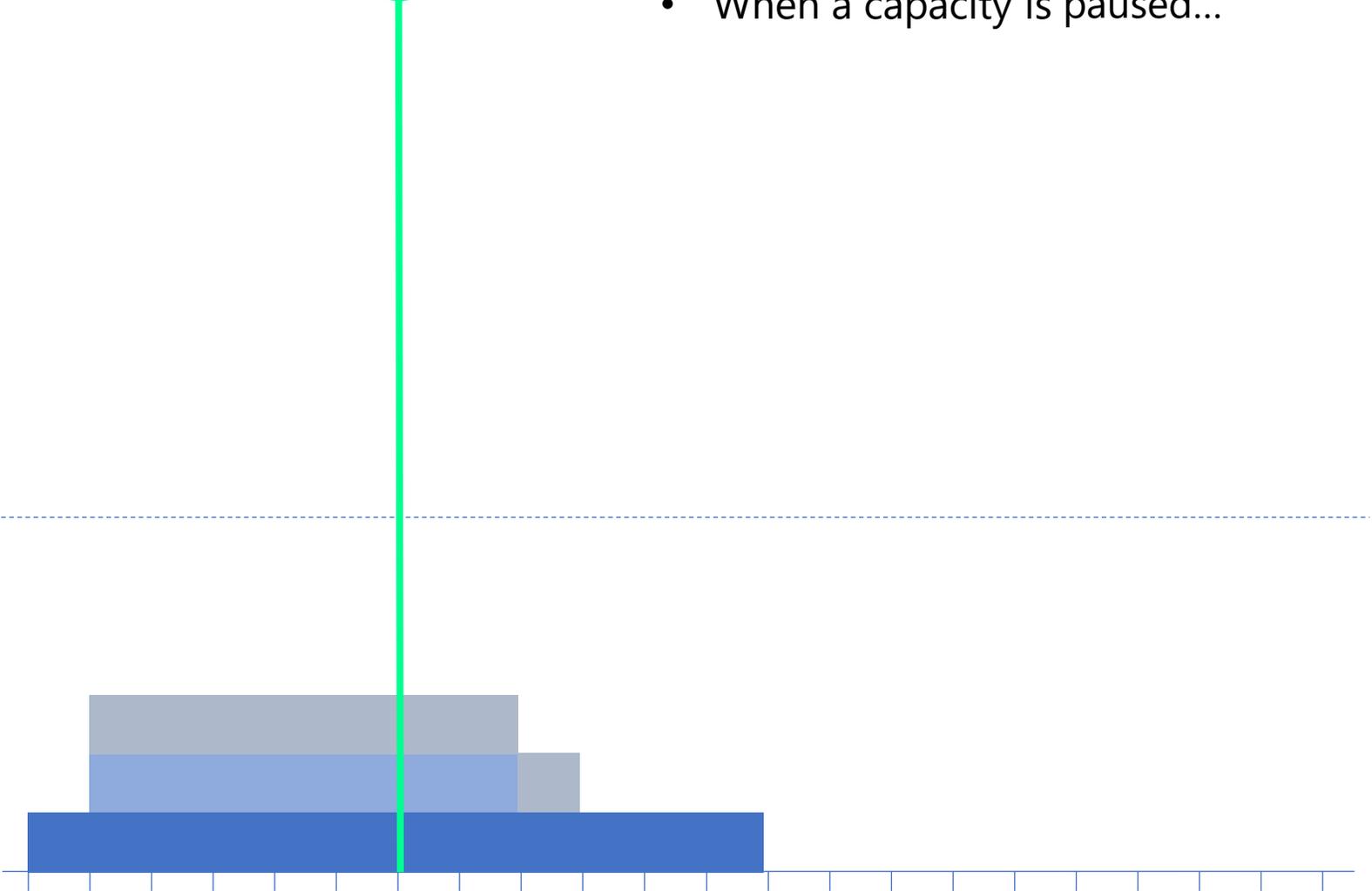
# Smoothing and Paused Capacities

Pause event on Capacity



- When a capacity is paused...

64 CUs

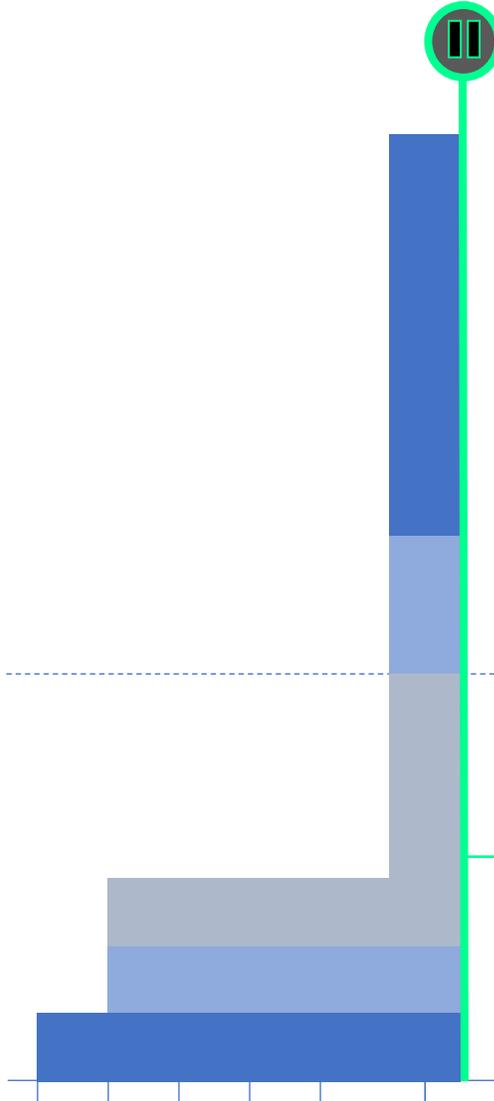


# Smoothing and Paused Capacities

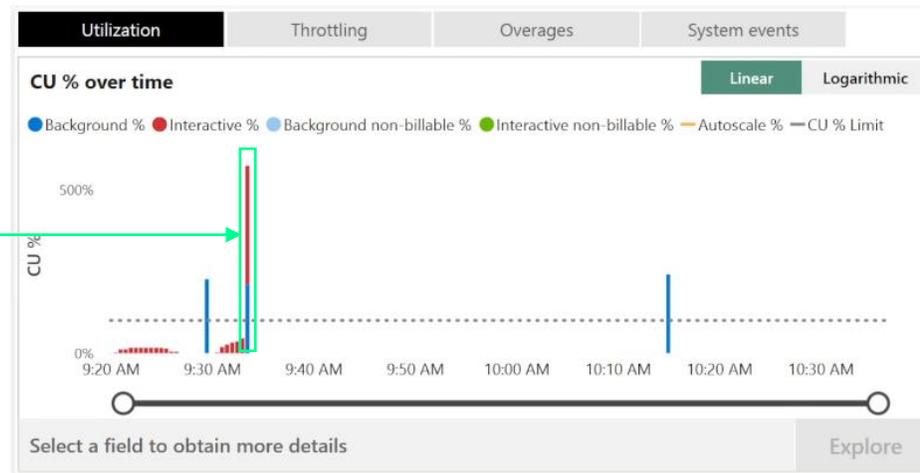
Pause event on Capacity



64 CUs



- When a capacity is paused...
- Usage that was smoothed into the future will be “reconciled” and charged against the capacity at the timestamp the capacity was paused
- Reconciled usage will show up as a spike in capacity metrics

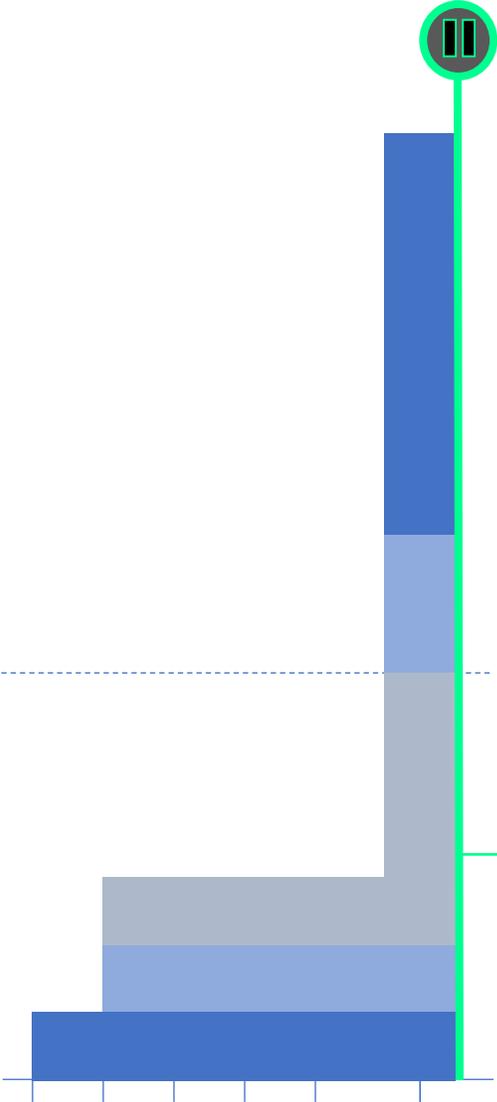


# Smoothing and Paused Capacities

Pause event on Capacity



64 CUs



- When a capacity is paused...
- Usage that was smoothed into the future will be “reconciled” and charged against the capacity at the timestamp the capacity was paused
- Pause events can be viewed in the new System events tab

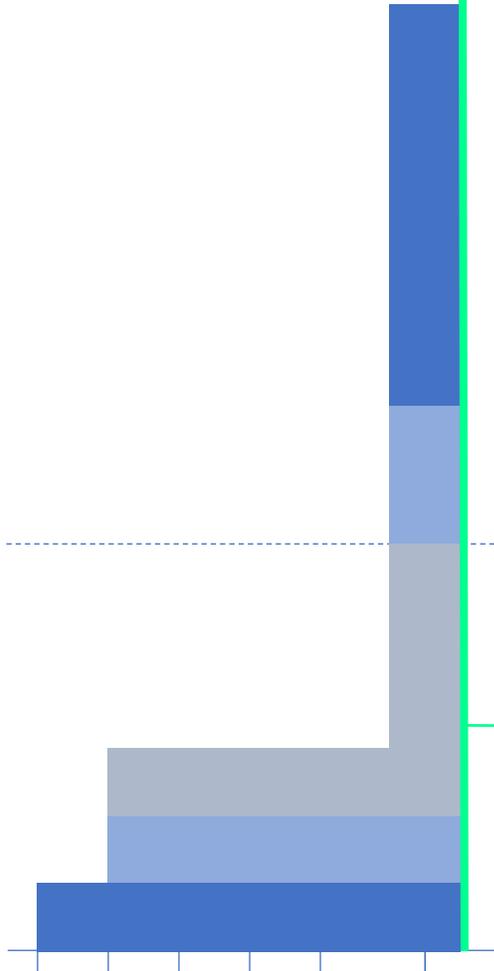
Utilization	Throttling	Overages	System events
<b>System events</b>			
State transition time	Capacity state	Capacity state change reason	
12/13/2023 9:12:14 AM	Active	Created	
12/13/2023 9:29:12 AM	Suspended	ManuallyPaused	
12/13/2023 9:30:15 AM	Active	ManuallyResumed	
12/13/2023 9:33:29 AM	Suspended	ManuallyPaused	
12/13/2023 9:34:58 AM	Active	ManuallyResumed	
12/13/2023 9:35:53 AM	Suspended	ManuallyPaused	

# Smoothing and Paused Capacities

Pause event on Capacity



64 CUs



- When a capacity is paused...
- Usage that was smoothed into the future will be “reconciled” and charged against the capacity at the timestamp the capacity was paused
- Pause events timestamp is shown in the smoothing end field in timepoint drill views

Fabric Capacity Metrics

Start : 12/13/2023 9:33:30 AM  
End : 12/13/2023 9:34:00 AM

CU %

70 Interactive operations  
5 Background operations

SKU

CU (s)

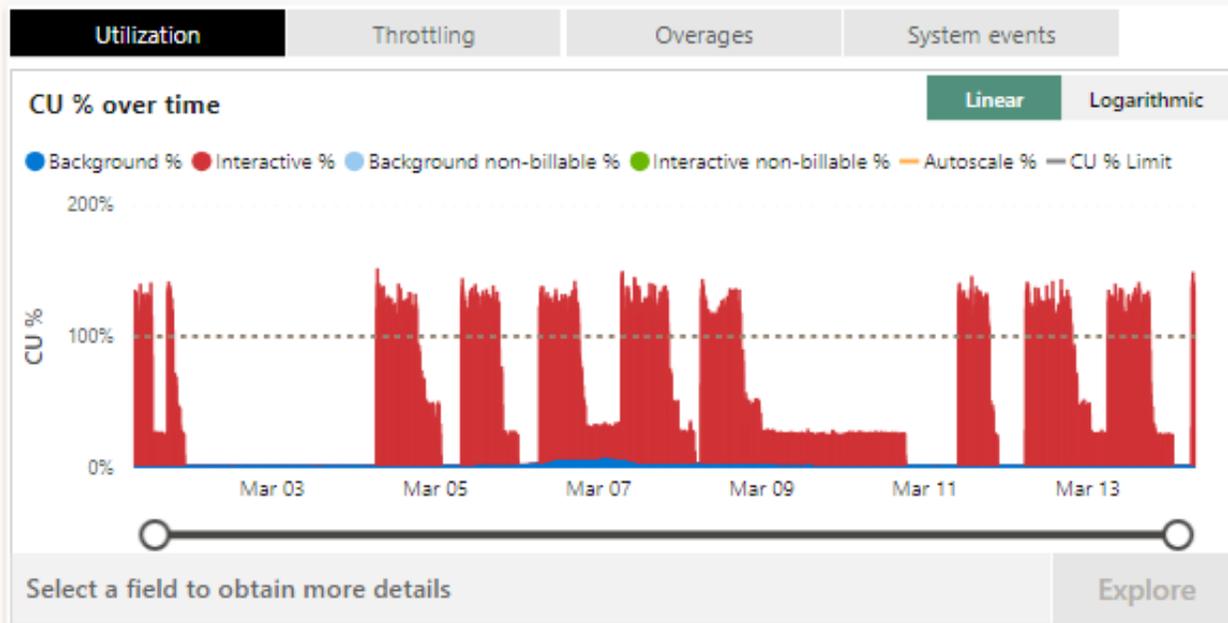
Interactive operations for timerange

End	Status	User	Duration (s)	Total CU (s)	Timepoint CU (s)	Throttling (s)	% of Base capacity	Billing type	Operation Id	Smoothing start	Smoothing end
12/13/2023 9:33:02 AM	Success	Admin@FabricMSIT.on...	22	32.4960	32.4960	0	54.16%	Billable		12/13/2023 9:33:00 AM	12/13/2023 9:33:00 AM
12/13/2023 9:31:33 AM	Success	Admin@FabricMSIT.on...	23	28.2560	19.7792	0	32.97%	Billable		12/13/2023 9:31:30 AM	12/13/2023 9:33:00 AM
12/13/2023 9:30:49 AM	Success	Admin@FabricMSIT.on...	7	31.5040	15.7520	0	26.25%	Billable		12/13/2023 9:30:30 AM	12/13/2023 9:33:00 AM
12/13/2023 9:32:51 AM	Success	Admin@FabricMSIT.on...	11	13.7440	12.3696	0	20.62%	Billable		12/13/2023 9:32:30 AM	12/13/2023 9:33:00 AM
12/13/2023 9:31:23 AM	Success	Admin@FabricMSIT.on...	13	18.0000	10.8000	0	18.00%	Billable		12/13/2023 9:31:00 AM	12/13/2023 9:33:00 AM
12/13/2023 9:32:08 AM	Success	Admin@FabricMSIT.on...	14	13.2480	10.5984	0	17.66%	Billable		12/13/2023 9:32:00 AM	12/13/2023 9:33:00 AM
12/13/2023 9:32:28 AM	Success	Admin@FabricMSIT.on...	4	11.7440	10.5696	0	17.62%	Billable		12/13/2023 9:32:30 AM	12/13/2023 9:33:00 AM
12/13/2023 9:33:06 AM	Success	Admin@FabricMSIT.on...	4	10.2560	10.2560	0	17.09%	Billable		12/13/2023 9:33:00 AM	12/13/2023 9:33:00 AM
12/13/2023 9:30:51 AM	Success	Admin@FabricMSIT.on...	9	19.2480	9.6240	0	16.04%	Billable		12/13/2023 9:30:30 AM	12/13/2023 9:33:00 AM
			<b>169</b>	<b>314.0160</b>	<b>218.0848</b>	<b>0</b>	<b>363.47%</b>				

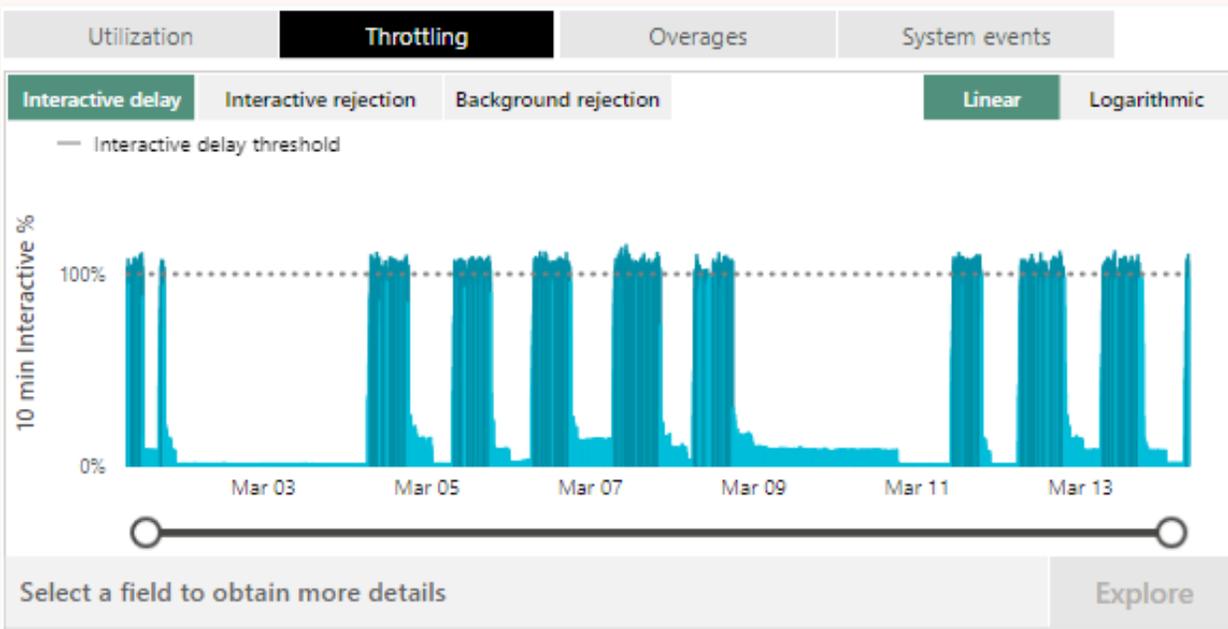


Bonus: Tips and Tricks  
for capacity  
management and  
monitoring

# My capacity is being throttled! What can I do?



Over 100% utilization doesn't always result in throttling

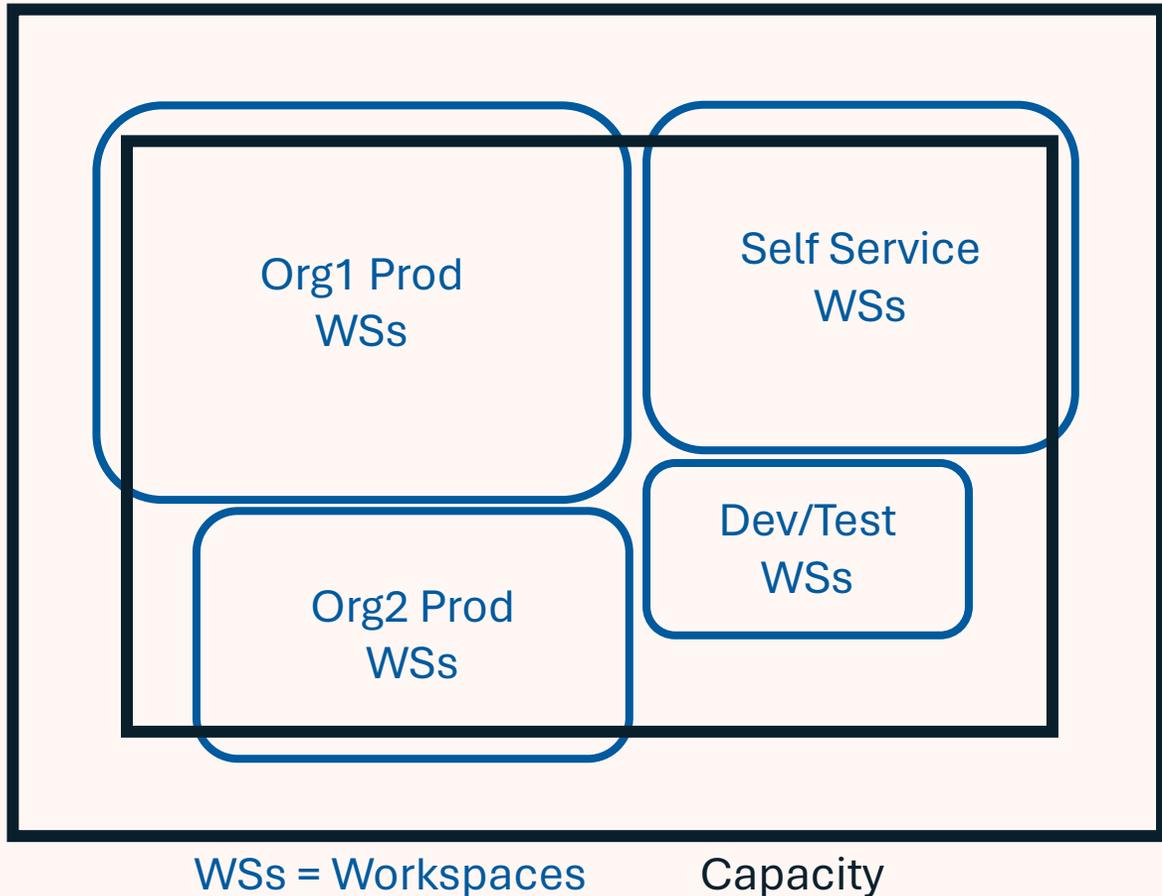


No penalty until you hit 100% on one of the throttling tabs

Note: For F SKU, if throttled, you can pause/resume to pay now and clear the carry forward, but that is not a long-term solution

# When Capacity Units Run Out

## Option 1 – Scale Up



### Options to add compute

- Move to a bigger P SKU or RI F SKU
- Turn on autoscale (P SKU)
- Manual/Dynamic change size (F SKU)

### Pros

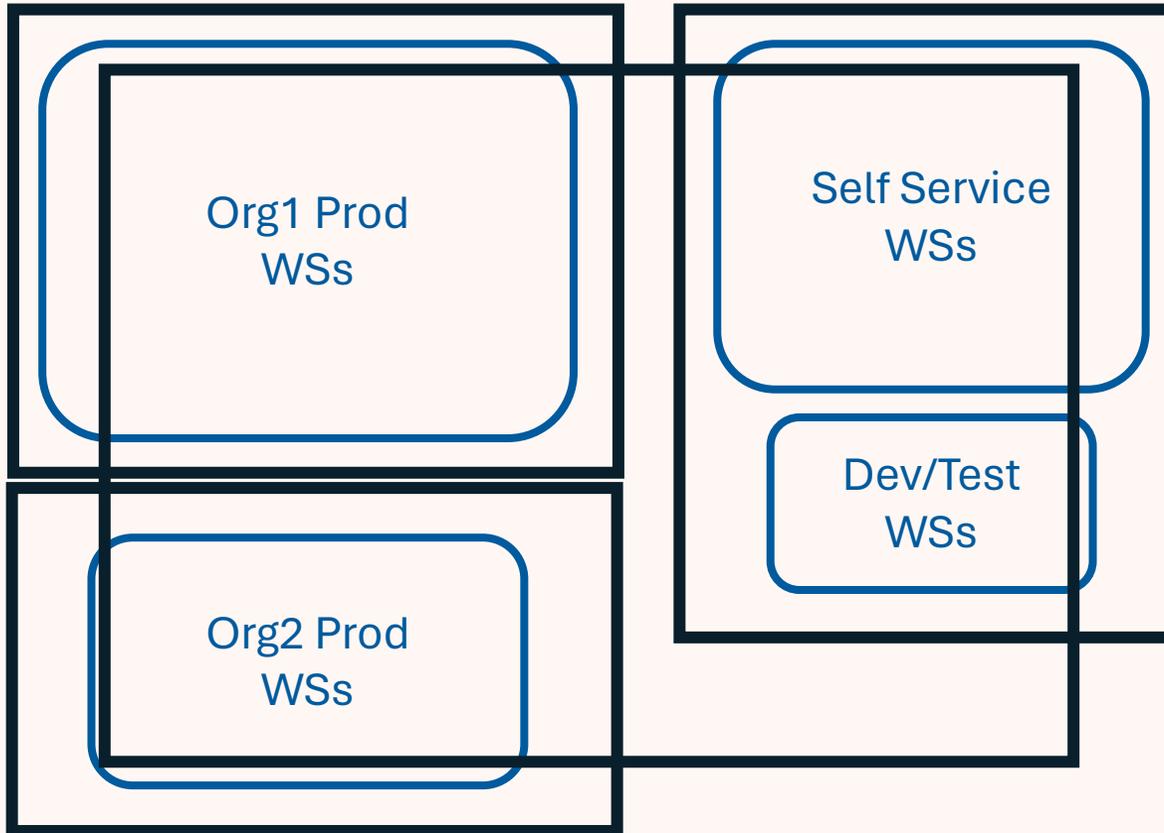
- Add CUs for all items
- Easy

### Cons

- Cost
- Bad actors (items with unintentionally high CU burn) can still be a problem

# When Capacity Units Run Out

## Option 2 – Scale Out



WSs = Workspaces

Capacity

### Options

- Create multiple smaller P or F SKUs based on organization, type of work, etc.

### Pros

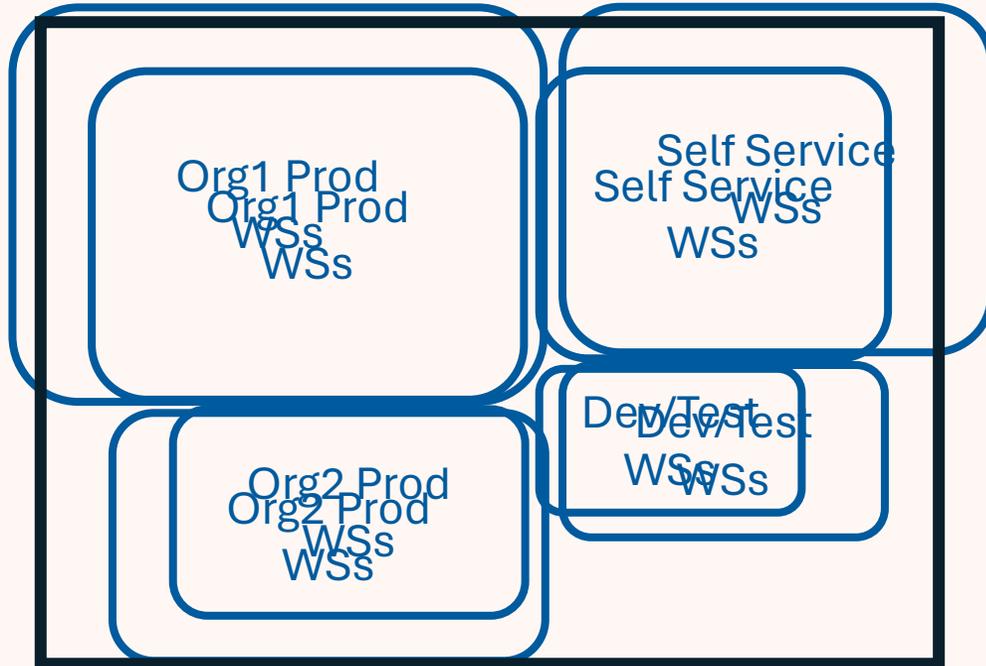
- Easy
- Provides some isolation from bad actors (items with unintentionally high CU burn)
- Flexibility in capacity settings/governance

### Cons

- Cost
- High CU items have increased chance of throttling

# When Capacity Units Run Out

## Option 3 – Optimize



WSs = Workspaces

Capacity

### Approach

- Work with content creators to follow best practices and reduce CU consumption

### Pros

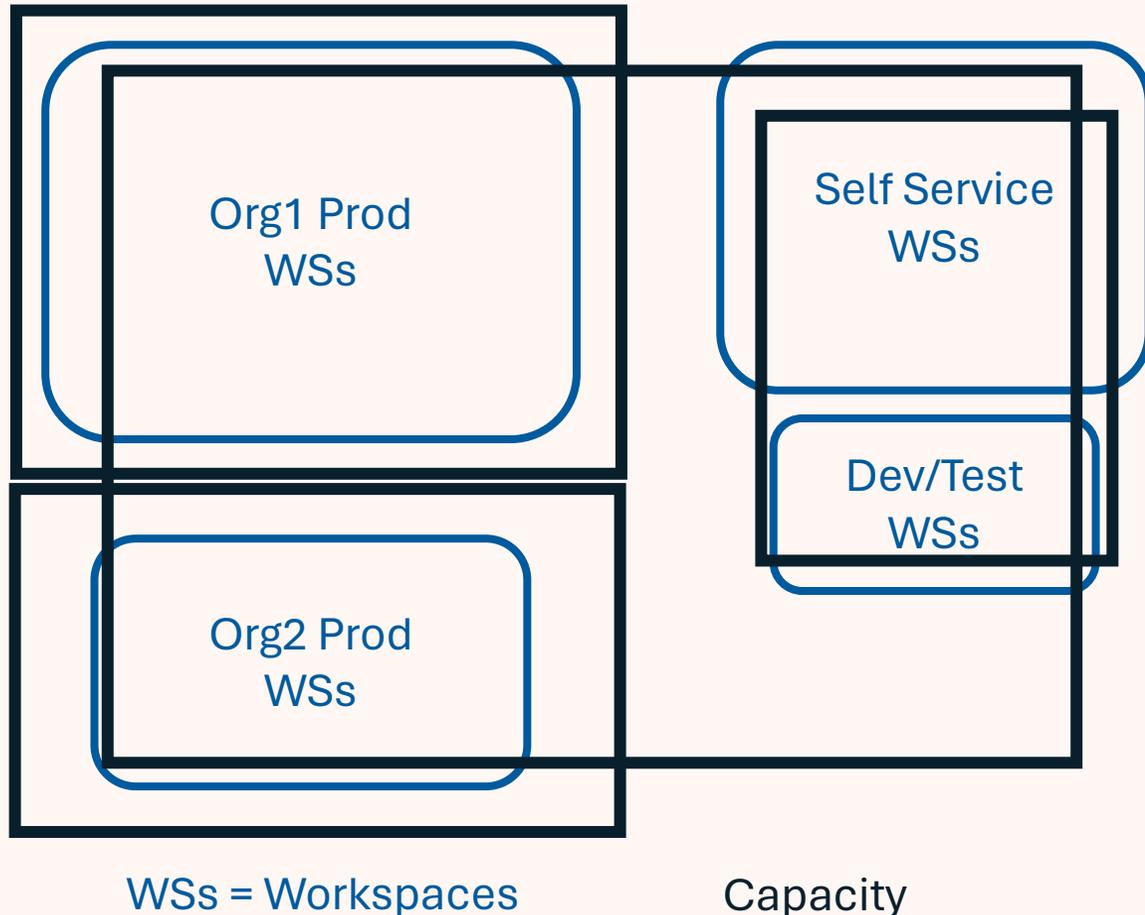
- Avoids increased cost
- Learning carries over to future content

### Cons

- Can be difficult/time consuming

# When Capacity Units Run Out

## Option 4 – Isolate



### Approach

- Provide isolated capacity for key items built by experienced developers

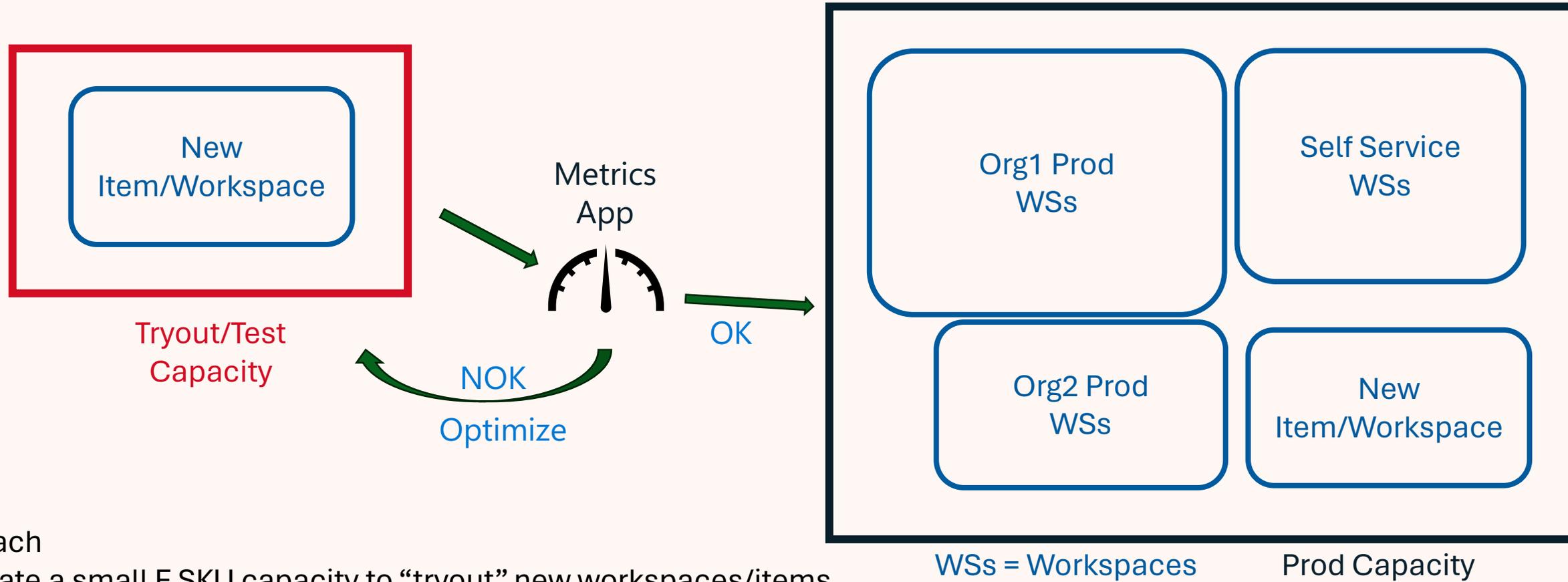
### Pros

- Easy
- Provides isolation from items built by inexperienced developers and/or rapid unplanned usage growth
- Flexibility in capacity settings/governance

### Cons

- Cost
- May lead to frustration of lower priority content developers/consumers

# Isolation Strategy #4a – Tryout Capacity



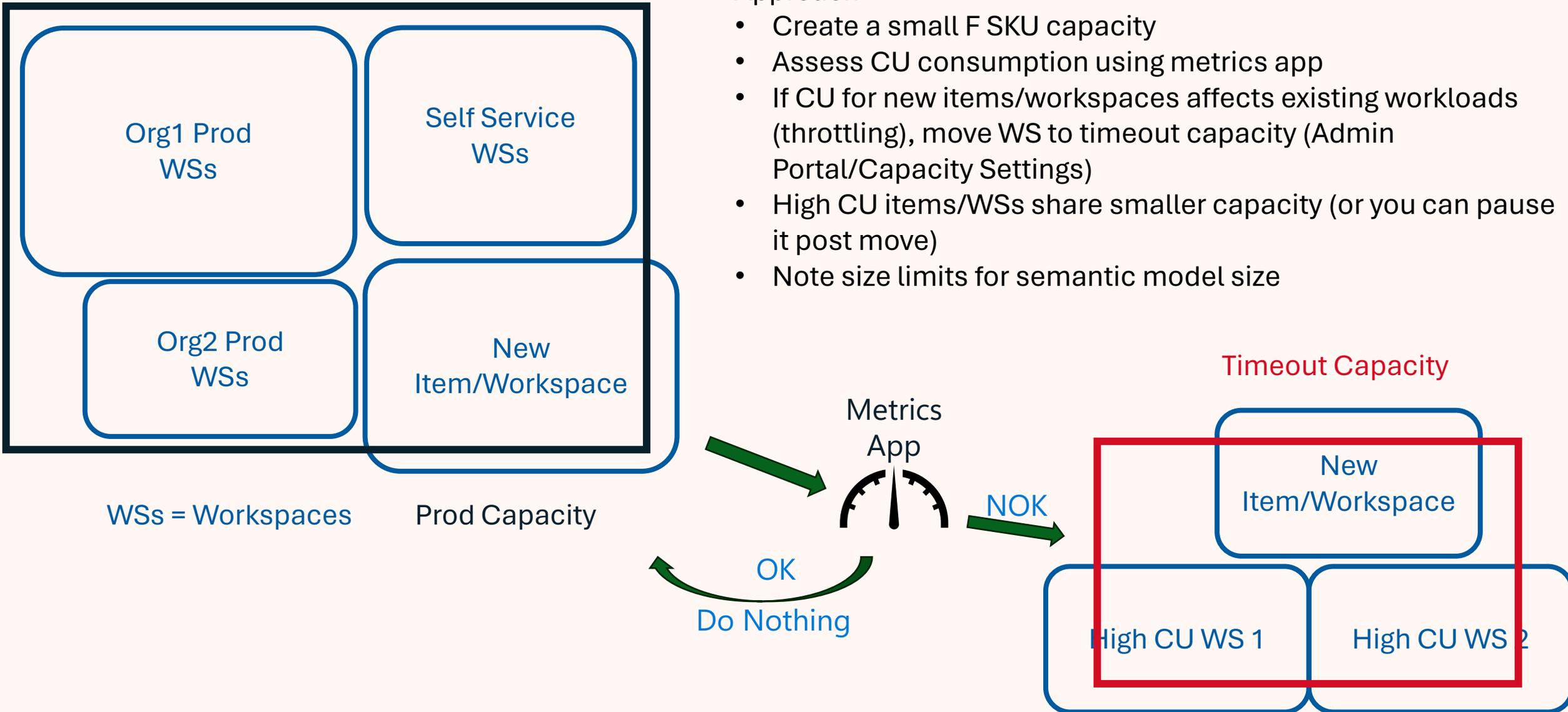
## Approach

- Create a small F SKU capacity to “tryout” new workspaces/items
- Assess CU consumption using metrics app
- If acceptable, move to prod capacity
- If not, optimize
- Pause tryout capacity when not in use, if possible
- Note size limits for semantic model size

# Isolation Strategy #4b – Timeout Capacity

## Approach

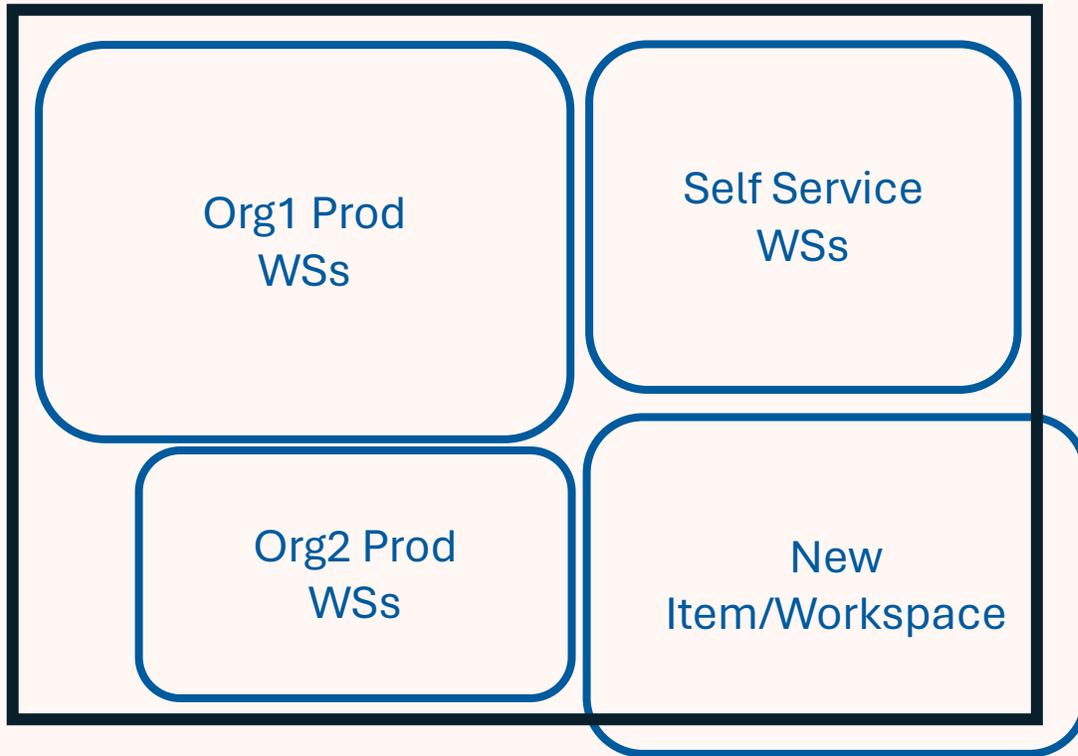
- Create a small F SKU capacity
- Assess CU consumption using metrics app
- If CU for new items/workspaces affects existing workloads (throttling), move WS to timeout capacity (Admin Portal/Capacity Settings)
- High CU items/WSs share smaller capacity (or you can pause it post move)
- Note size limits for semantic model size



# Isolation Strategy #4c – Rescue Capacity

## Approach

- Create an F SKU capacity, keep it paused
- Assess CU consumption using metrics app
- If CU for new items/workspaces affects priority workloads (throttling), resume the new capacity and move priority WS to it (Admin Portal/Capacity Settings)
- Address issues with new content, then bring it back to original capacity, and pause the new one
- Note size limits for semantic model size



WSs = Workspaces

Prod Capacity



Rescue Capacity



# Recommendations for Cost/CU Savings

- Invest in education, knowledge/best practice sharing, COE, etc. for creators and consumers (proactive optimization)
- Avoid data/report sprawl (leverage certified/promoted models, OneLake shortcuts, etc.)
- Leverage a multi-capacity strategy (isolate, tryout, timeout, etc.)
- Right size your capacities and leverage F SKUs for pause/resume/resize, or reserved instances for discounts
  - Consider a combo of RI and PAYGO (for predictable surge activity)
- Choose the right tool for the job and stay up to date on Fabric feature releases
  - High concurrency mode for notebooks

# Leverage the capacity settings in the UI

- Notifications on CU overuse
- Power BI workloads settings (e.g., query limits, page refresh)

Capacity settings

- Refresh summary
- Embed Codes
- Organizational visuals
- Azure connections
- Workspaces
- Custom branding
- Protection metrics
- Featured content
- Help + support

Your P1 SKU gives you access to 64 capacity units.

[Change size](#)

- Disaster Recovery
- Capacity usage report
- Notifications
- Contributor permissions  
*Enabled for a subset of the organization*
- Admin permissions
- Power BI workloads
- Preferred capacity for My workspace
- Data Engineering/Science Settings
- Workspaces assigned to this capacity

Notifications

Get notified when you're close to exceeding your available capacity (which includes base and Autoscale v-cores).

Send notifications when

- You're using  % of your available capacity
- You've exceeded your available capacity and might experience slowdowns
- An Autoscale v-core has been added
- You've reached your Autoscale maximum

Send notifications to

- Capacity admins
- These contacts:

Power BI workloads

**SEMANTIC MODELS**

Observe XMLA-based workspace settings (which may override capacity settings)

On

Query Memory Limit (%)

Query Timeout (seconds)

Max Intermediate Row Count

Max Result Row Count

Max Offline Dataset Size (GB)

Automatic page refresh  On

Minimum refresh interval

Change detection measure  On

Minimum execution interval

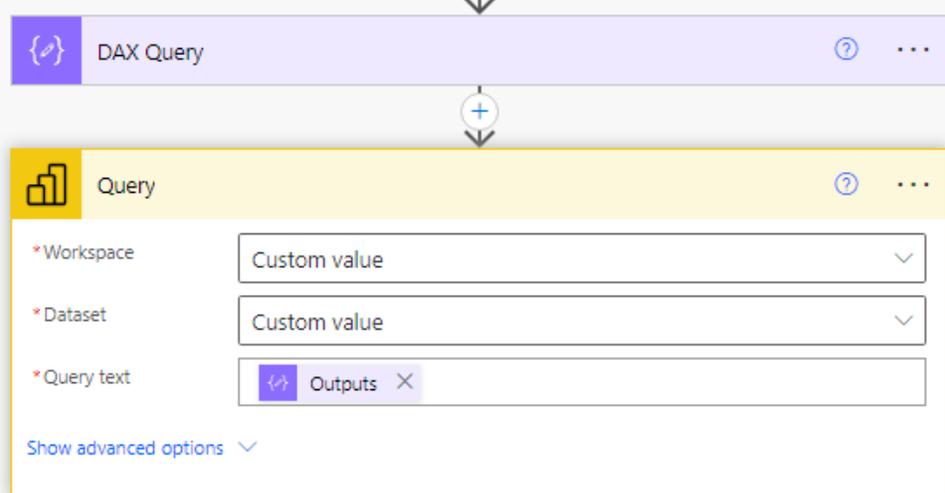
XMLA Endpoint

# Custom Solutions

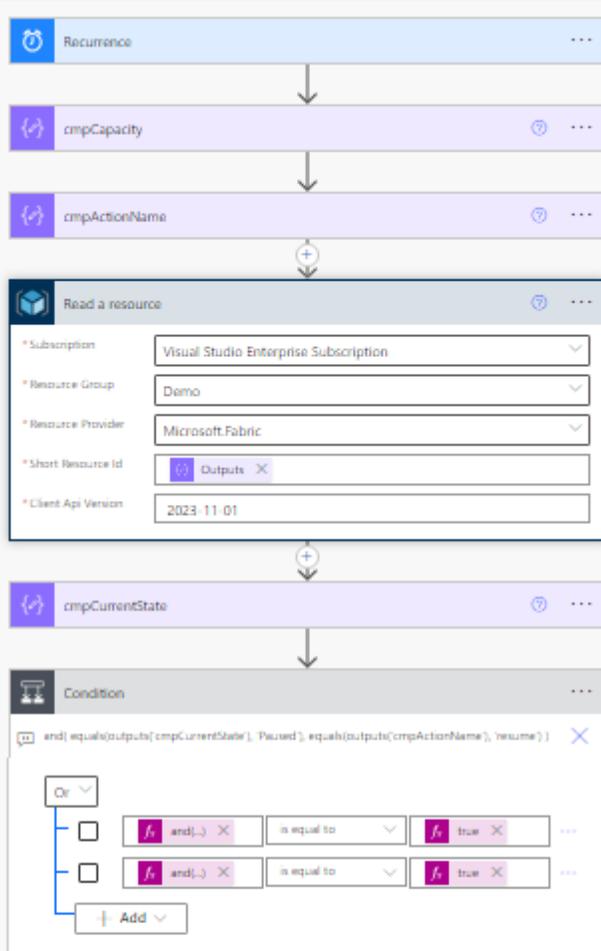
- Modify the Metrics App to meet your needs
- Build a custom report off the semantic model
- Send DAX queries to the metrics app semantic model in your own solution
  - Power Automate, Notebook (SemPy), PowerShell, etc.
  - Get throttling % values (Interactive Delay, Interactive Rejection, and/or Background Rejection)
  - Latest values and/or trends over time
  - Best for summarized data only (e.g., hour, day)

Collect data from multiple capacities and store it long term

Incorporate Metrics App queries into custom solutions



```
8 # Get max date from current delta table (to avoid loading duplicate days)
9 try:
10     df_max = spark.sql(f'''
11         SELECT MAX(Date) as MaxDate
12         FROM throttling;
13     ''')
14     maxdate = df_max.first()['MaxDate']
15 except:
16     maxdate = datetime.today() + timedelta(days=-6)
17 maxdateforDAX = maxdate.strftime('%Y,%m,%d')
18
19 if maxdate.date() < (datetime.today() + timedelta(days=-1)).date():
20
21     # Get data for each capacity, write daily csv and append delta
22     for capacity in lst_capacities:
23         querytext = '''\
24             DEFINE
25             MPARAMETER 'CapacityID' = "{capID}"
26             VAR yesterday =
27                 FILTER(ALL('Dates'[Date]), 'Dates'[Date] < TODAY() && 'Dates'[Date] > DATE({MD}))
28
29             EVALUATE
30             SUMMARIZECOLUMNS(
31                 'Dates'[Date],
32                 'TimePoints'[Start of Hour],
33                 yesterday,
34                 "IntDelay", ROUND('All Measures'[Dynamic InteractiveDelay %] * 100, 2),
35                 "IntReject", ROUND('All Measures'[Dynamic InteractiveRejection %] * 100, 2),
36                 "BackReject", ROUND('All Measures'[Dynamic BackgroundRejection %] * 100, 2)
37             )
38         '''
39         df_throttling = fabric.evaluate_dax(workspace=MetricsWS, dataset=MetricsModel, dax_string=querytext)
40         if len(df_throttling) >= 1:
41             df_throttling.columns = df_throttling.columns.str.replace(r'(\.|\.)|(\.|\.)', '', regex=True)
42             df_throttling.columns = df_throttling.columns.str.replace(' ', '_')
43             df_throttling['capacityId'] = capacity
44             filename = capacity + '_throttling_' + (datetime.today()).strftime('%Y,%m,%d') + '.csv'
45             df_throttling.to_csv("/lakehouse/default/Files/ThrottlingData/" + filename)
46             spk_throttle = spark.createDataFrame(df_throttling)
47             spk_throttle.write.mode("append").format("delta").option("overwriteSchema", "true").saveAsTable('Throttling')
```



if yes

Invoke resource operation

\*Subscription: Visual Studio Enterprise Subscription

\*Resource Group: Demo

\*Resource Provider: Microsoft.Fabric

\*Short Resource Id: Outputs

\*Client Api Version: 2023-11-01

\*Action name: Outputs

body: Action request body

if no

Add an action

Pause/Resume on a Schedule

# Automate With F SKUs

- Pause/resume on a schedule
  - Automate with Power Automate, Logic Apps, or a Notebook
- Resize at peak/slow times
  - Mix with Reserved Instance (PAYGO when at increased size)
  - Query the metrics app and respond to actual demand (DIY autoscale)

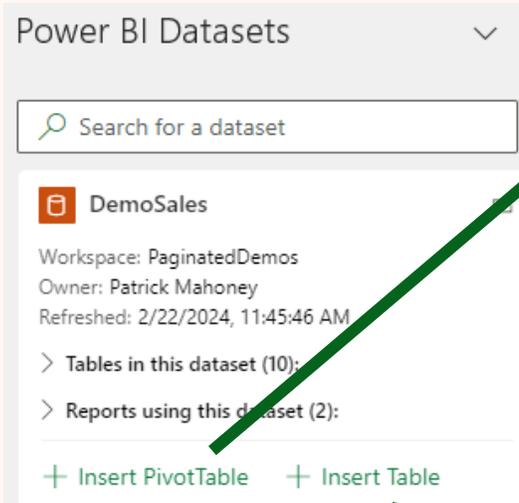


# Most Common Capacity Issues (Power BI)

Bad Practice	Recommendations/Typical Resolution
Model issues (M:M, bi-di, snowflake, etc.) and/or inefficient DAX	Follow best practices (e.g., BPA), star schema
Too many visuals	Multi card, small multiples, Deneb, PowerPoint background, etc.
Big single visual (i.e., matrix with lots of rows, columns, and/or measures)	Improve report design (e.g., drillthrough, apply all Slicers, report page tooltip), field parameters, calc group guardrails, etc.
Complex RLS	Remodel to enable simple filter like <code>Table[Email] = USERPRINCIPALNAME()</code>
Very high concurrency	Optimize reports, DAX, etc. (big multiplier) Consider QSO, data subsets
Direct Query	Switch to import or Direct Lake, if possible. Aggregations, hybrid tables, etc.
Analyze in Excel	Automate downstream analytics with a Power BI report instead, subscriptions, DAX connected table, slicers/measures first, etc.
Excessive refresh	Don't "break the fold", incremental refresh, reduce frequency, optimize M code

# Save Those CUs – Getting Data Into Excel

Analyze in Excel



Connected Table

✗ Rows, Measures, Filter

StartTime	Type	Duration	User	Database	Query
11:49:30	MDX	2,328ms	Power BI...	DemoSales	SELECT {[Measures].[To
11:49:26	MDX	0ms	Power BI...	DemoSales	SELECT {AddCalculated
11:49:23	MDX	0ms	Power BI...	DemoSales	SELECT {AddCalculated
11:49:17	MDX	1,875ms	Power BI...	DemoSales	SELECT {[Measures].[To
11:49:03	MDX	4,469ms	Power BI...	DemoSales	SELECT {[Measures].[To
11:48:54	MDX	3,938ms	Power BI...	DemoSales	SELECT {[Measures].ITo

✓ Filter, measures, rows

StartTime	Type	Duration	User	Database	Query
10:06:13	MDX	1,625ms	Power BI...	DemoSales	SELECT {[Measu
10:06:03	MDX	781ms	Power BI...	DemoSales	SELECT {[Measu
10:05:49	MDX	109ms	Power BI...	DemoSales	SELECT {[Measu
10:05:46	MDX	312ms	Power BI...	DemoSales	SELECT {[Measu
10:05:43	MDX	234ms	Power BI...	DemoSales	SELECT FROM [M
10:05:14	MDX	0ms	Power BI...	DemoSales	SELECT {AddCal

Refresh (same for both)

StartTime	Type	Duration	User	Database	Query
11:50:30	MDX	2,234ms	Power BI...	DemoSales	SELECT {[Measures].[To

✗ Rows, Measure, Filter

StartTime	Type	Duration	User	Database	Query
01:28:50	DAX	31ms	Power BI...	DemoSales	DEFINE VAR __C
01:28:41	DAX	1,516ms	Power BI...	DemoSales	DEFINE VAR __C
01:28:40	DAX	16ms	Power BI...	DemoSales	DEFINE VAR __C
01:28:34	DAX	156ms	Power BI...	DemoSales	DEFINE VAR __C
01:28:33	DAX	16ms	Power BI...	DemoSales	DEFINE VAR __C
01:28:31	DAX	0ms	Power BI...	DemoSales	DEFINE VAR __C
01:28:30	DAX	141ms	Power BI...	DemoSales	DEFINE VAR __C
01:28:15	DAX	2,047ms	Power BI...	DemoSales	DEFINE VAR __C
01:28:11	DAX	1,797ms	Power BI...	DemoSales	DEFINE VAR __C
01:28:08	DAX	594ms	Power BI...	DemoSales	DEFINE VAR __C
01:27:56	DAX	281ms	Power BI...	DemoSales	DEFINE VAR __C
01:27:50	DAX	16ms	Power BI...	DemoSales	DEFINE VAR __C

✓ Filter, measures, rows

StartTime	Type	Duration	User	Database	Query
09:14:20	DAX	16ms	Power BI...	DemoSales	DEFINE VAR __DSOFilte
09:14:07	DAX	1,000ms	Power BI...	DemoSales	DEFINE VAR __DSOFilte
09:14:02	DAX	1,188ms	Power BI...	DemoSales	DEFINE VAR __DSOFilte
09:13:59	DAX	594ms	Power BI...	DemoSales	DEFINE VAR __DSOFilte
09:13:51	DAX	531ms	Power BI...	DemoSales	DEFINE VAR __DSOFilte
09:13:50	DAX	0ms	Power BI...	DemoSales	DEFINE VAR __DSOCor

Refresh (same for both)

StartTime	Type	Duration	User	Database	Query
11:54:49	DAX	1,969ms	Power BI...	DemoSales	DEFINE VAR __DSOFilterTable = TREATA

## Key Takeaways

- How you build it matters
  - Filters & measures first!
- This shows durations but it's CU that matters (test your use cases/models)
- Opt for DAX Connected Tables
  - Create pivot table from that, if needed



# Session Feedback



# Slides

[https://github.com/BenniDeJagere/Presentations/{Year}/{YYYYMMDD}\\_{Event}](https://github.com/BenniDeJagere/Presentations/{Year}/{YYYYMMDD}_{Event})





# Thank you

