Understanding Fabric Capacities

Benni De Jagere

Slides

CAT

Thank you to our sponsors

DATA SATURDAY OSLO
2024

Fraktal

CEGAL

Ingraphic

WEBSTEP

Høyskolen Kristiania

Cloudberries

RIKSTOTO

TIMEXTENDER

Tabular Editor

Microsoft

Benni De Jagere

Senior Program Manager | Fabric Customer Advisory Team (FabricCAT)

Fabric CAT

.be Member

@BenniDeJagere

/bennidejagere

/bennidejagere

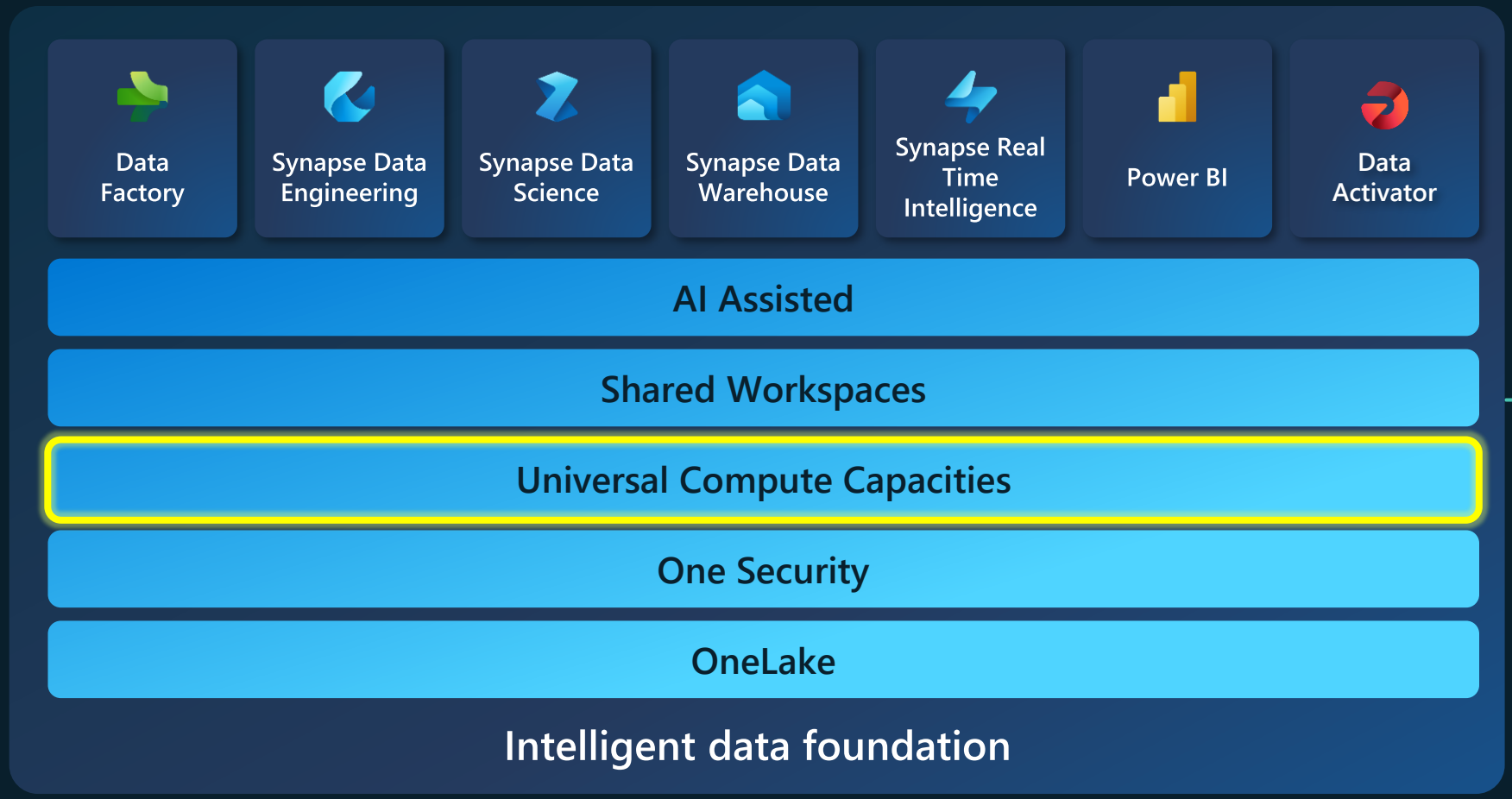
/bennidejagere

#SayNoToPieCharts

Microsoft Fabric
COMMUNITY CONFERENCE

Fabric Capacities Introduction

Microsoft Fabric



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

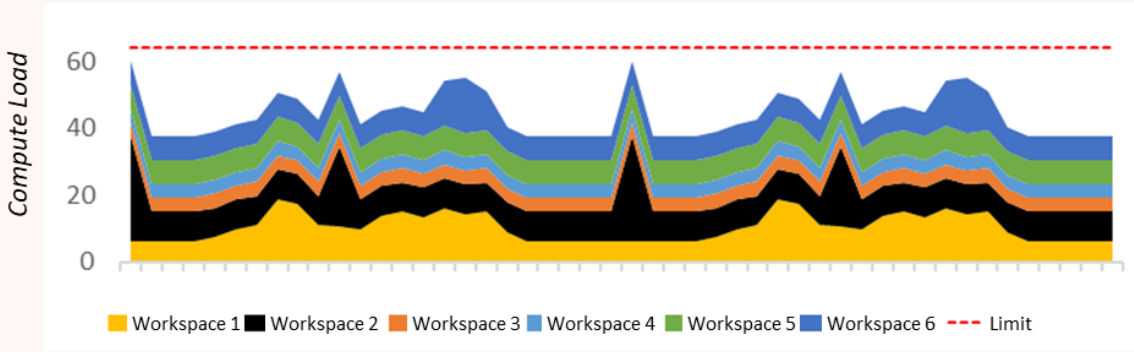
Capacities are a shared resource

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.

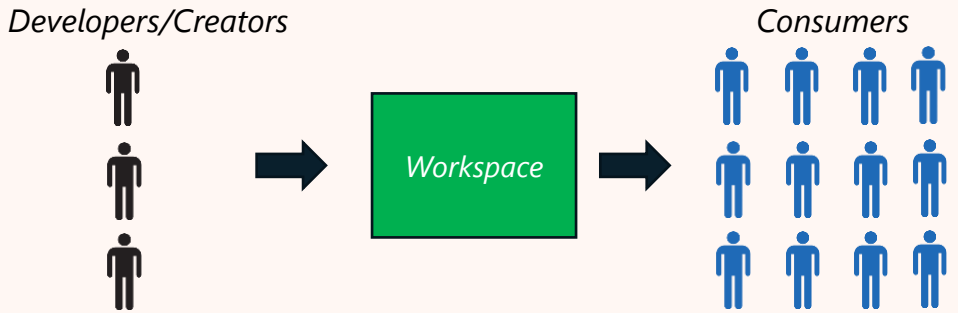
Shared Across Projects
A single capacity typically supports dozens of separate projects simultaneously, each managed in its own workspace.

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

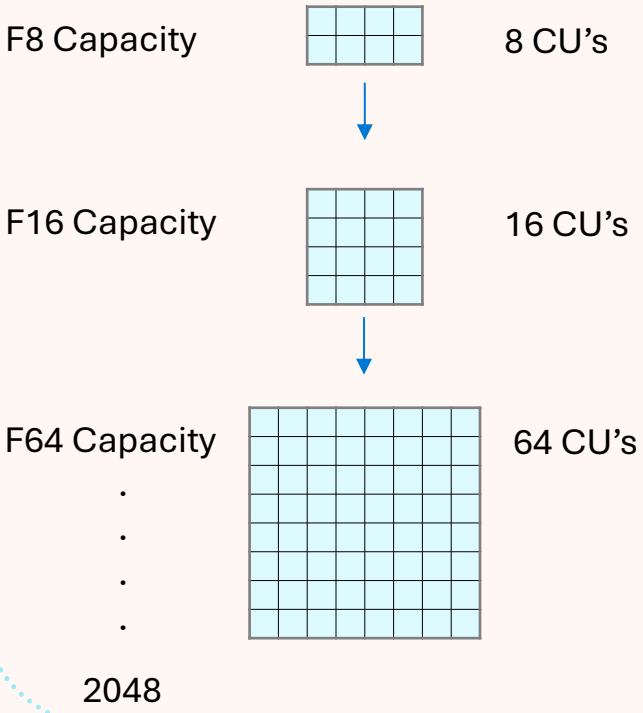


Capacities are flexible building blocks for growth

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

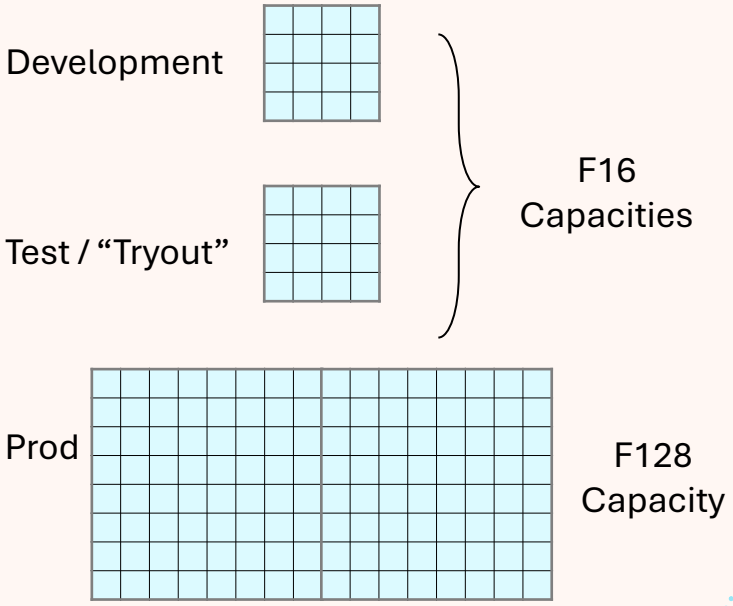
Scale Vertically

Increased capacity size provides more throughput



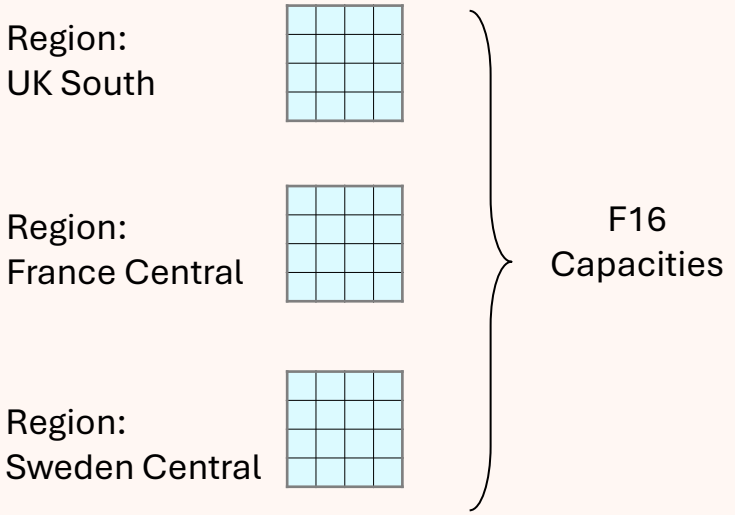
Scale Horizontally

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**

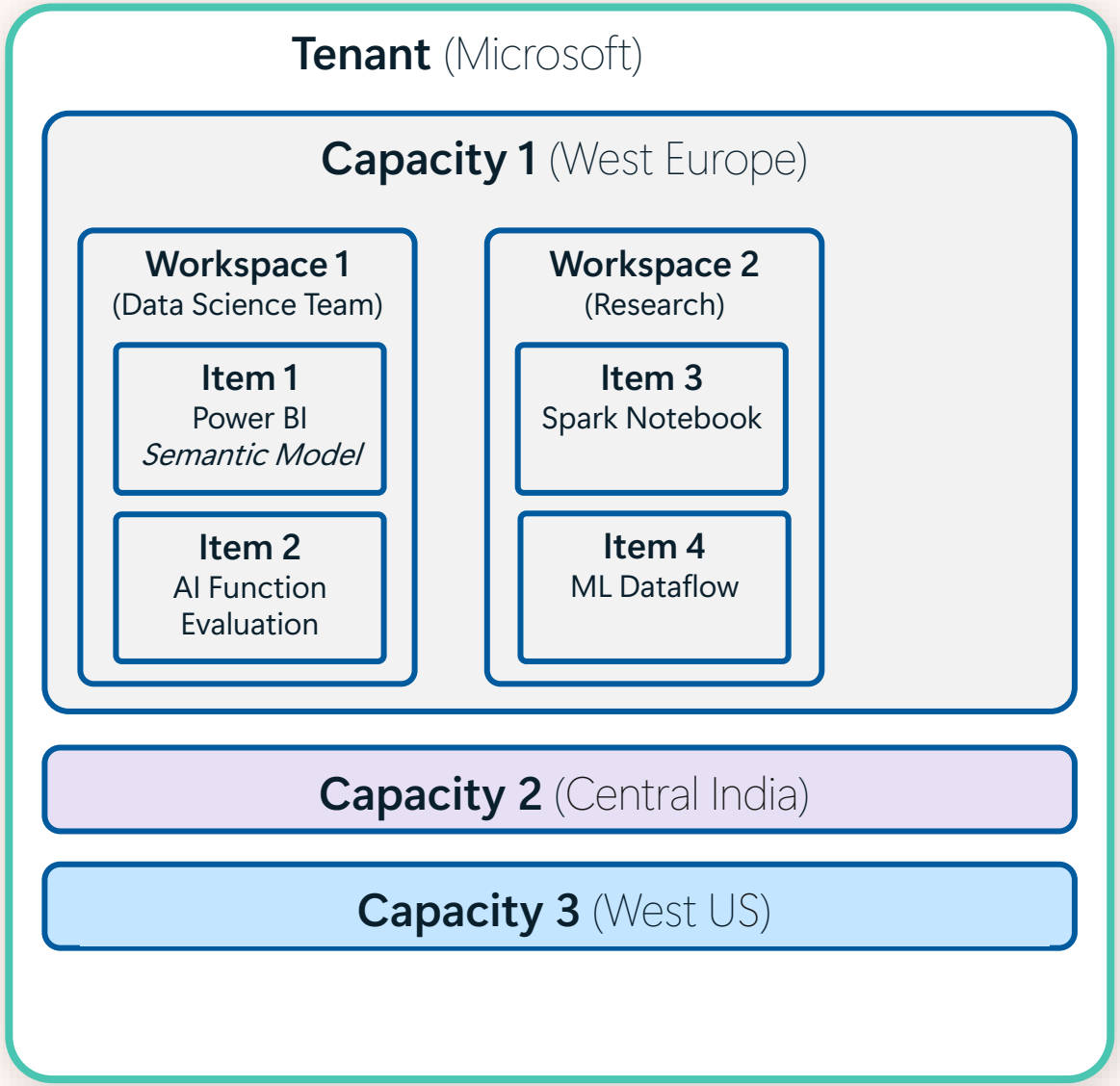
Universal Compute Capacities SKU Sizing

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



Microsoft Fabric
COMMUNITY CONFERENCE

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.

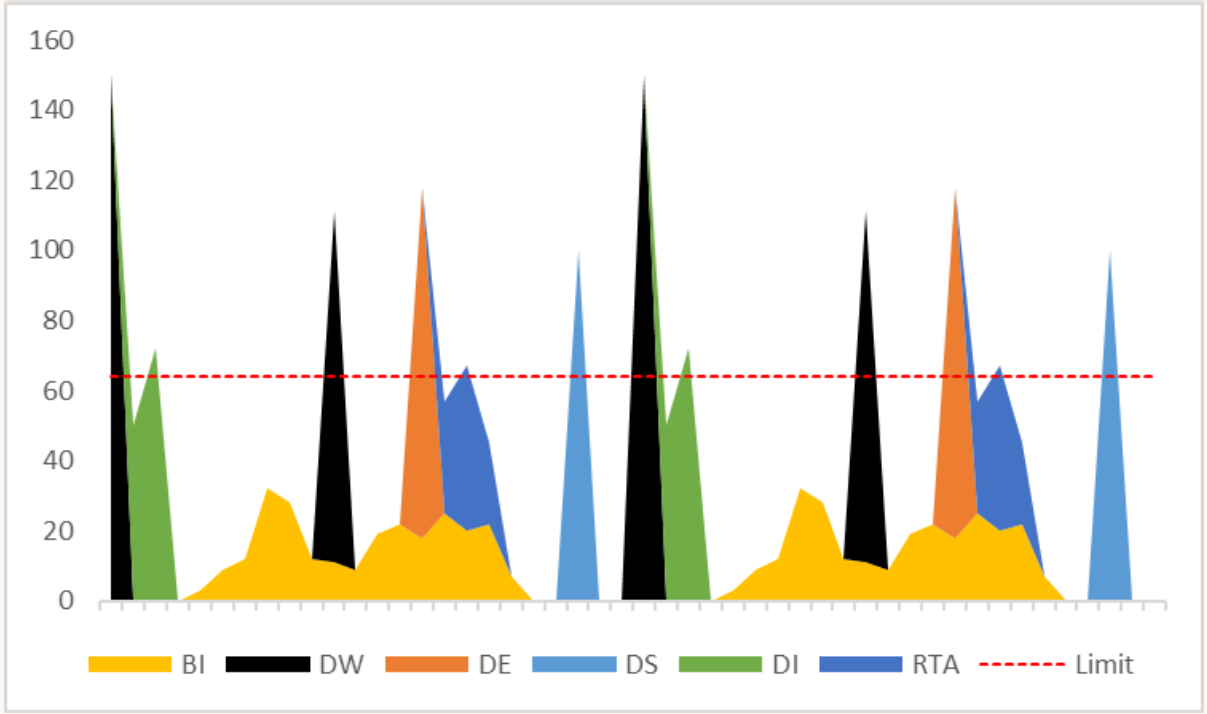
Bursting prevents an overload as the *smoothing* mechanism will always flatten the resource burst

Bursting and smoothing | before and after

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

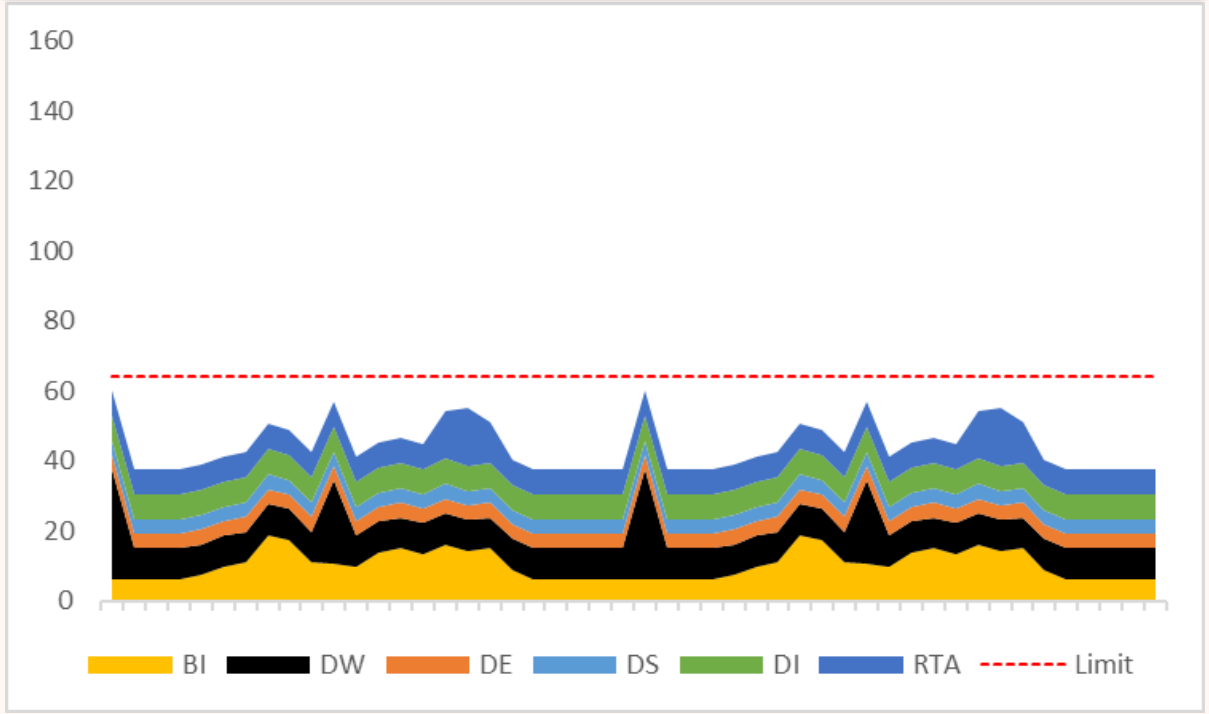
Before Smoothing

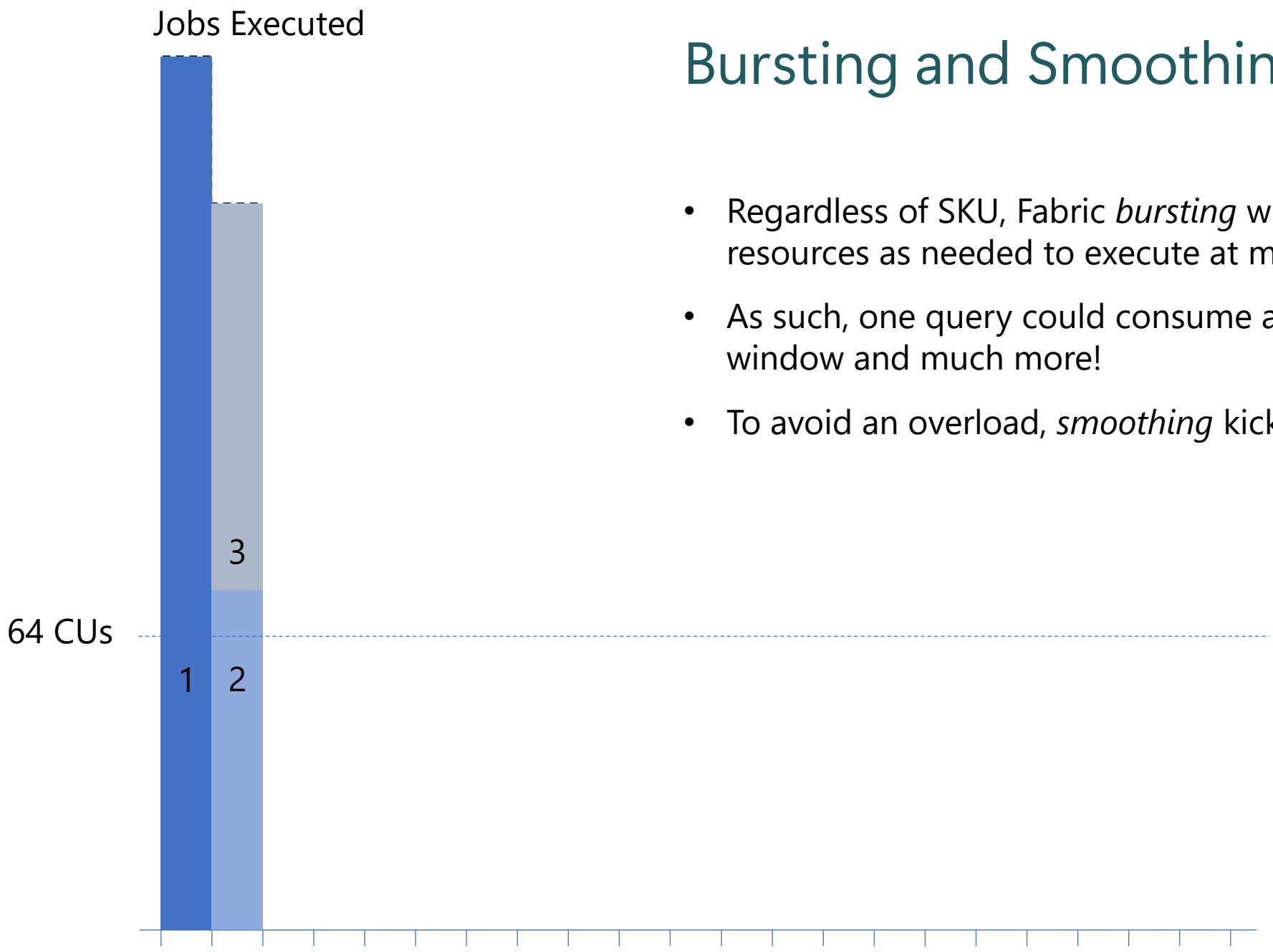
- Actual load as executed on the capacity before smoothing
- *Bursting* accelerates jobs execution by resource boosting
- The capacity could be overloaded 25% of the time
- Some of the overloads are more than 2x the limit
- There are periods of no/low usage



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

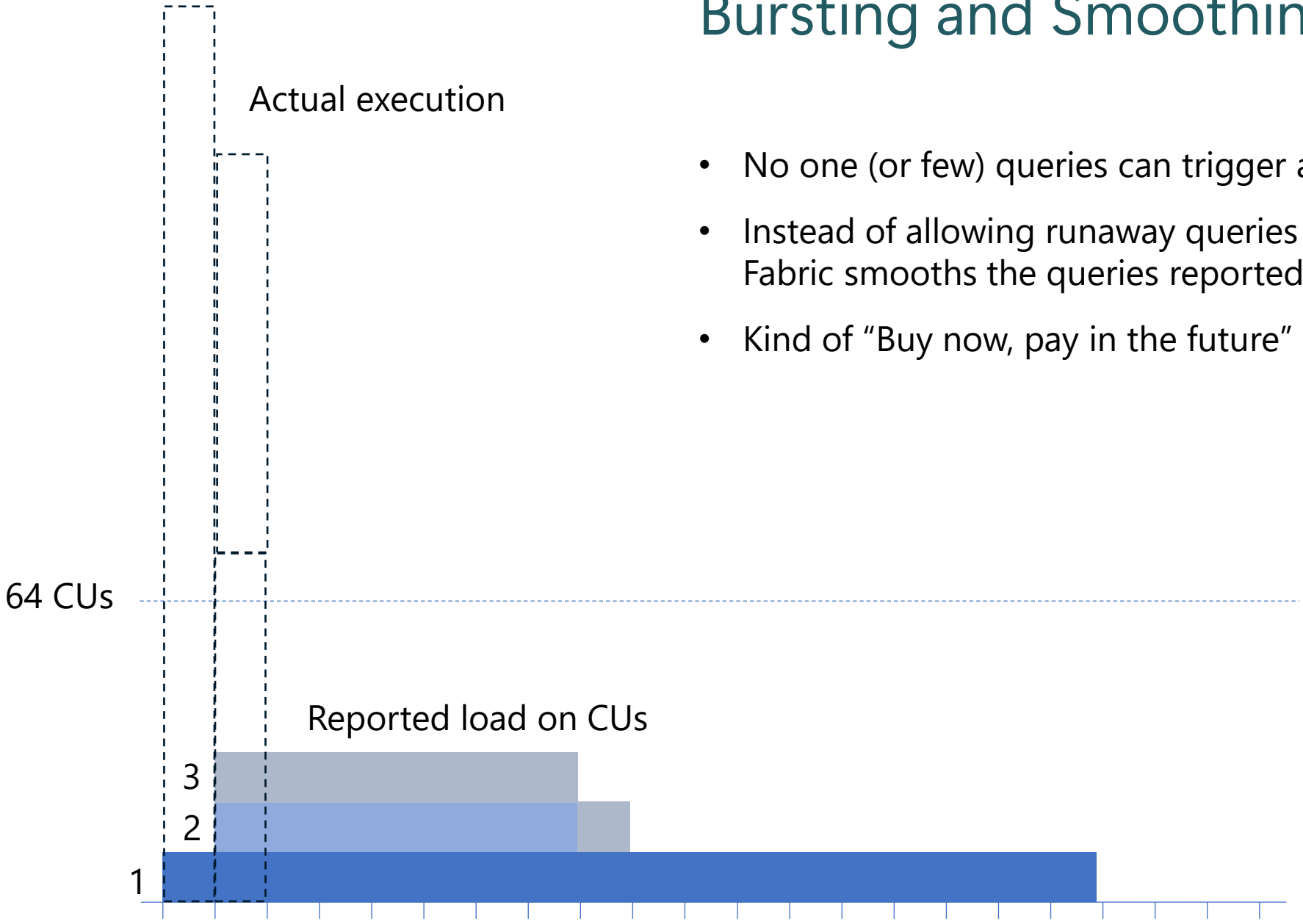




Bursting and Smoothing

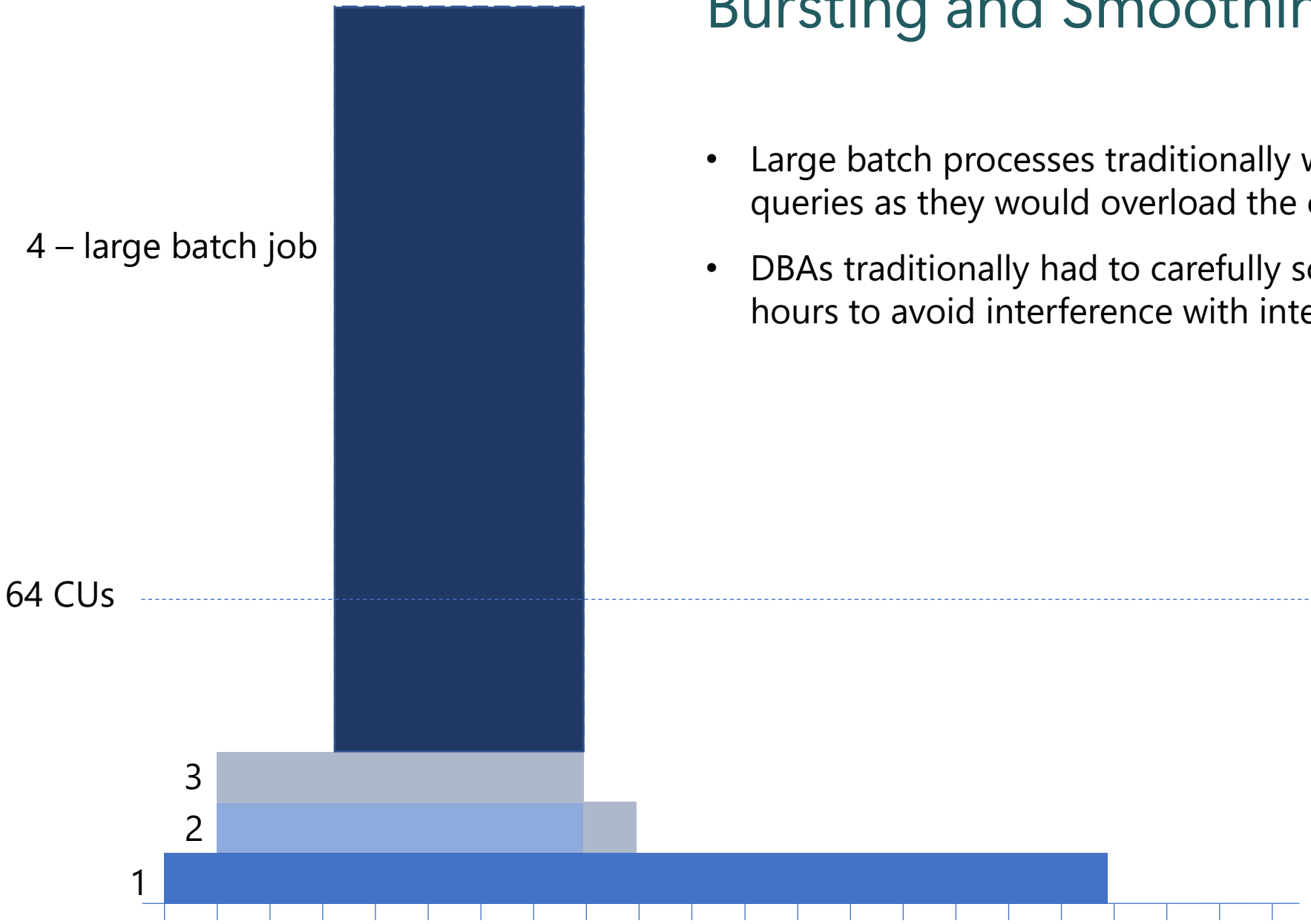
- Regardless of SKU, Fabric *bursting* will automatically allocate resources as needed to execute at maximum performance
- As such, one query could consume all the quota of a single time window and much more!
- To avoid an overload, *smoothing* kicks in

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

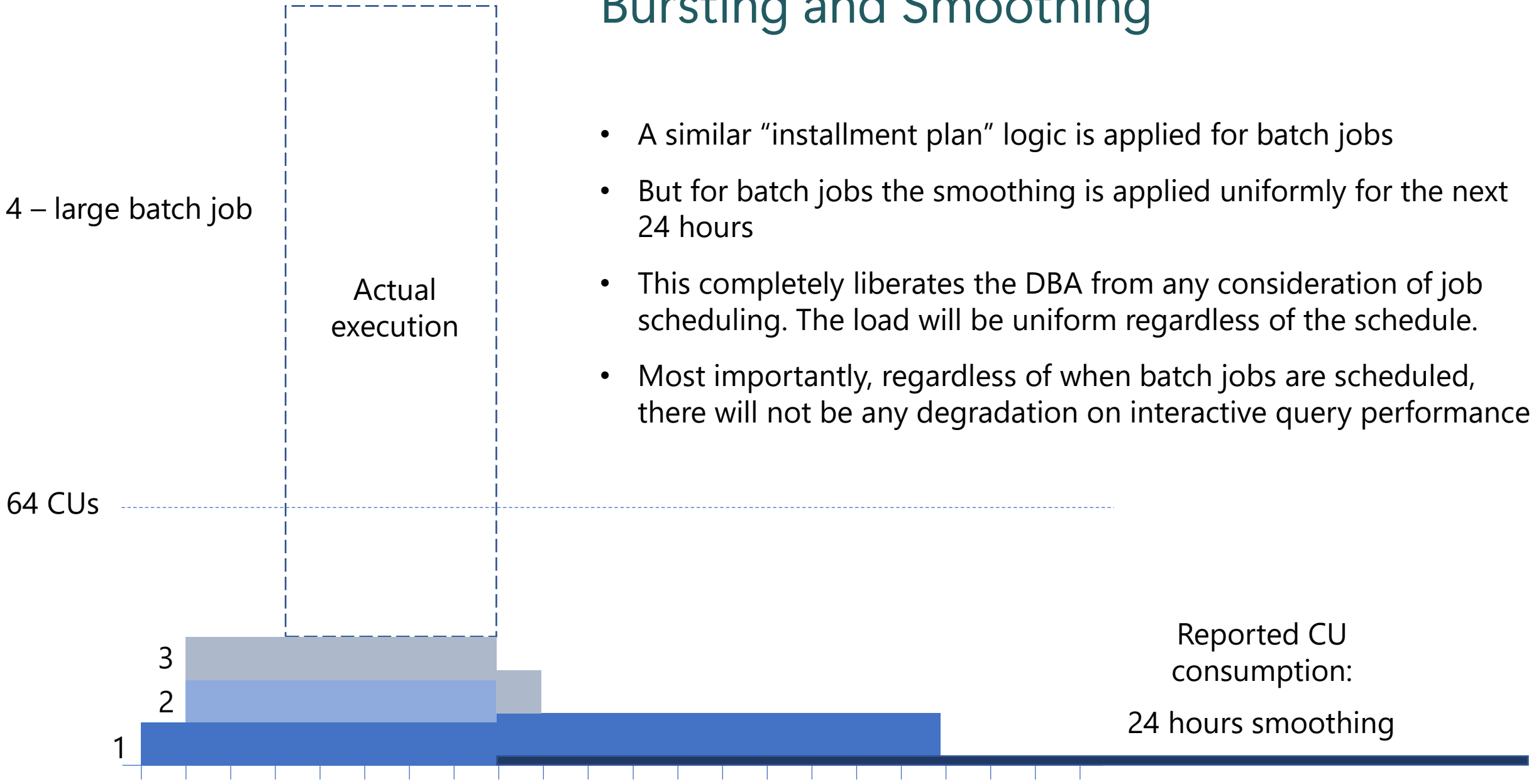
Bursting and Smoothing



- 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



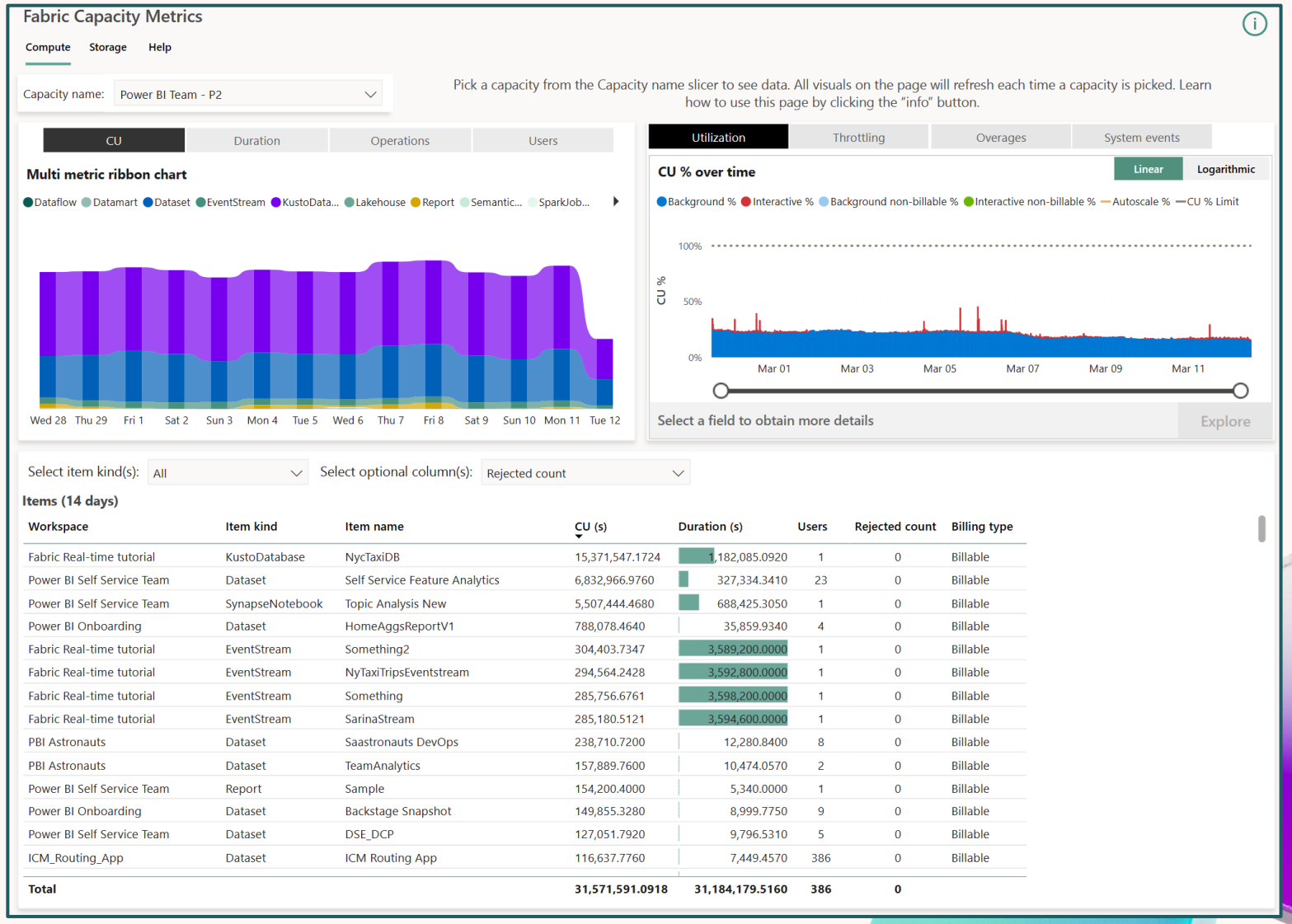
Microsoft Fabric
COMMUNITY CONFERENCE

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

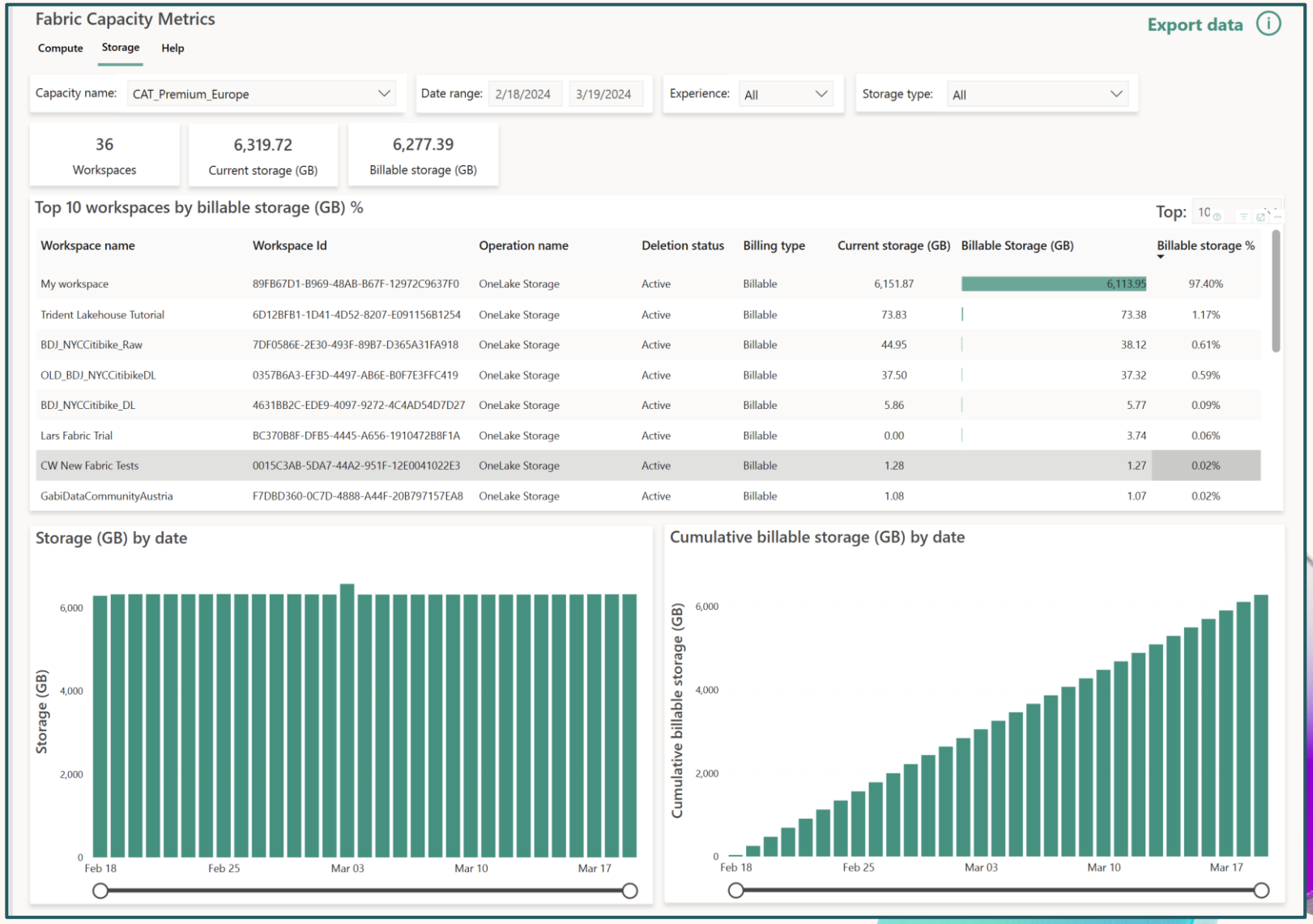


Capacity Metrics

Monitor OneLake consumption

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

Reconcile costs with internal chargeback processes



Fabric Capacity Metrics

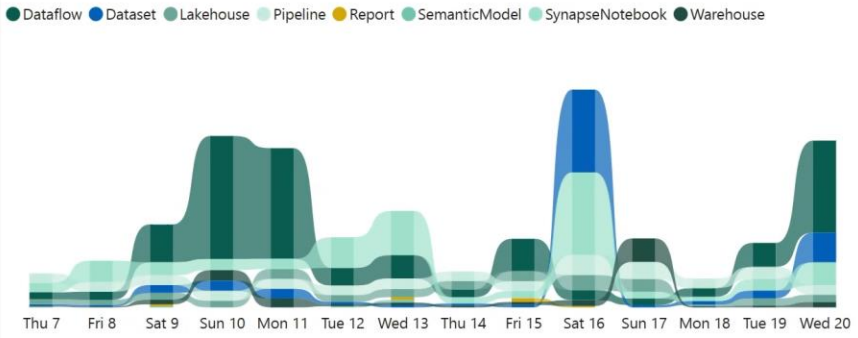
Compute Storage Help

Capacity name: CAT_Premium_US

Pick a capacity from the Capacity name slicer to see data. All visuals on the page will refresh each time a capacity is picked. Learn how to use this page by clicking the "info" button.

CU | Duration | Operations | Users

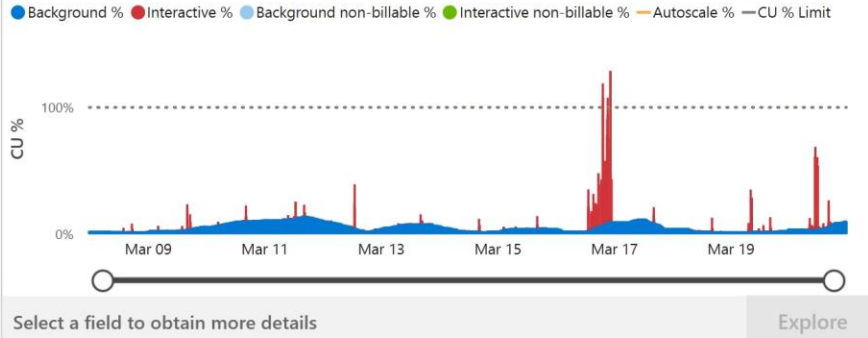
Multi metric ribbon chart



Utilization | Throttling | Overages | System events

CU % over time

Linear | Logarithmic



Select a field to obtain more details | Explore

Select item kind(s): All

Select optional column(s): Multiple selections

Items (14 days)

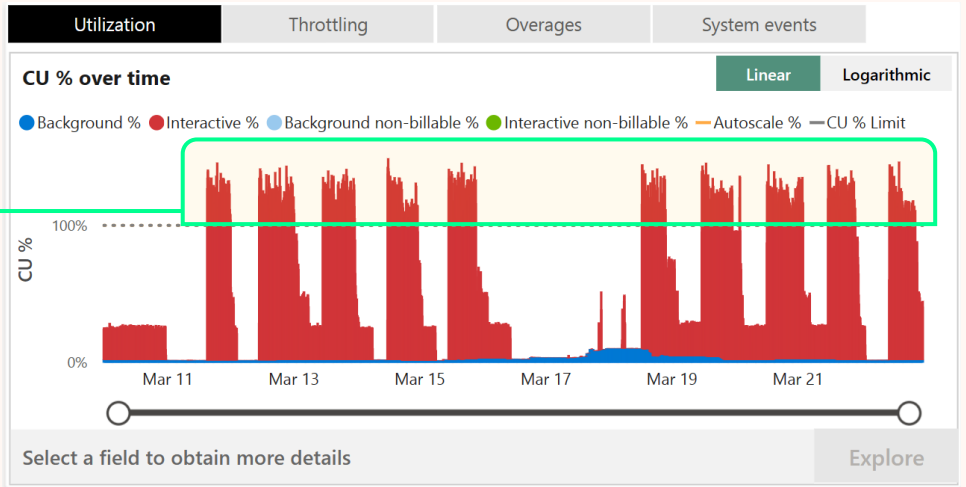
Workspace	Item kind	Item name	CU (s)	Duration (s)	Users	Rejected count	Overloaded minutes	Successful count	Performance delta	Billing type
Fabric for Power BI users (alpowe...	Dataflow	Dataflow 1Nashville	902,712.8000	14,585.3890	1	0	0.0000	97		Billable
Data Factory in an Hour (old)	Pipeline	dfPipeline	475,200.0000	19,858.7140	1	0	0.0000	1247		Billable
Analytics at the speed of Direct L...	Dataset	Metrics	359,352.5440	11,100.0940	3	0	1.3333	1095		Billable
Day After DIAD (alpowers)	Dataflow	OnlineSalesDataflow	355,036.3520	14,154.7430	1	0	0.0000	12		Billable
DADIAD Sample	Dataflow	DataflowTest	343,605.4880	15,609.4410	1	0	0.0000	12		Billable
Survey_test	SynapseNotebook	Survey_Notebook	214,899.6685	24,328.8260	2	0	0.0000	42		Billable
Mahoney_MetricsSnapshots	SynapseNotebook	MetricsAppSnapshots	184,471.7595	20,047.6250	1	0	0.0000	0		Billable
Analytics at the speed of Direct L...	SynapseNotebook	Notebook 1	165,965.6310	19,895.5380	2	0	0.0000	0		Billable
BDJ_NYCCitibike_Base	Warehouse	NYCCitibike_BASE	87,065.7860	50,126.5260	3	0	0.0000	3164		Billable
Lars - ready for demo	SynapseNotebook	Framing Demo	86,681.3000	8,158.5510	1	0	0.0000	0		Billable
CW Direct Lake Paging	Dataflow	CollectDMVData	83,097.0720	7,427.1870	1	0	0.0000	52		Billable
Fabric_Demo	Dataflow	lh_Sample	82,095.6800	4,454.1890	2	0	0.0000	73		Billable
Fabric for Power BI users (alpowe...	Dataset	WeirdDemo	77,008.5280	1,806.6590	2	0	0.0000	540		Billable
Mahoney_MetricsSnapshots	SynapseNotebook	Notebook 1	63,871.2960	7,408.3180	1	0	0.0000	0		Billable
Total			**4,907,150.8628**	**461,929.2860**	**4**	**0**	**2.0000**	**1662399**		

Microsoft Fabric
COMMUNITY CONFERENCE

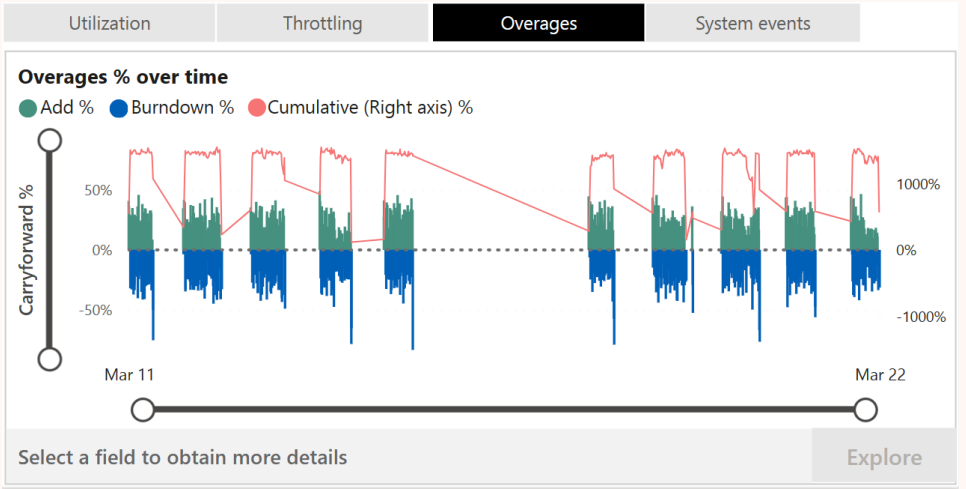
Capacity Throttling Policies

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

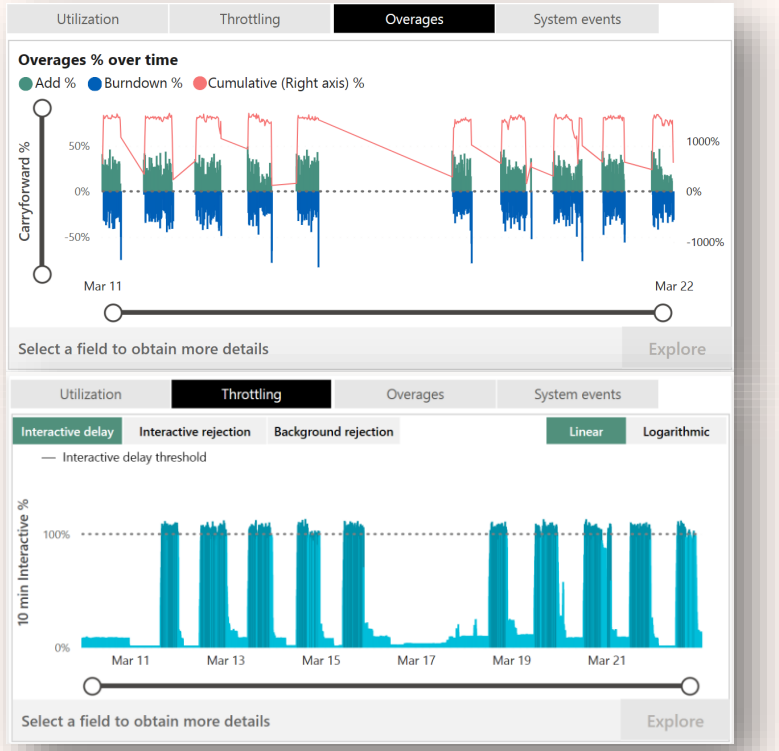


Overage Operation	Description
Overages - Added	• Timepoint when job requests exceed the throughput of a capacity, overages are was added to the cumulative buffer to burn down. • This graph simplifies identification of the optimal timepoint to load timepoint drill to analyze the user operations that contributed to an overage.
Overages - Burndown	• Overages being reconciled when future capacity is free to burn down
Overages - Cumulative	• The total amount of queued work on the capacity to be burned down in the future when the capacity is not fully utilized



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



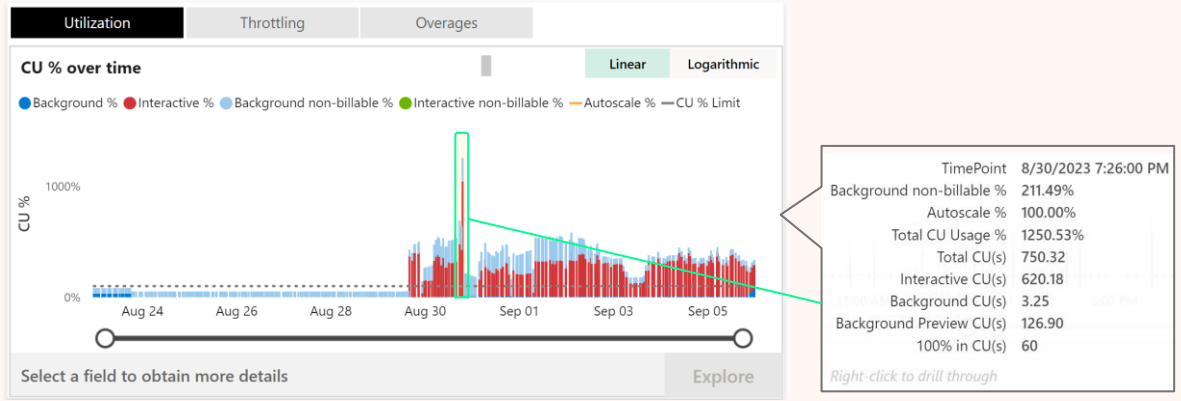
Smoothed Capacity - Future Use	Platform Policy	Customer Impact
<= 10m	Overage Protection	Jobs can consume 10 minutes of future capacity use without throttling
> 10m → <= 60m	Interactive Delay	User requested interactive type jobs will be throttled
> 60m → <= 24h	Interactive Rejection	User requested interactive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

Microsoft Fabric
COMMUNITY CONFERENCE

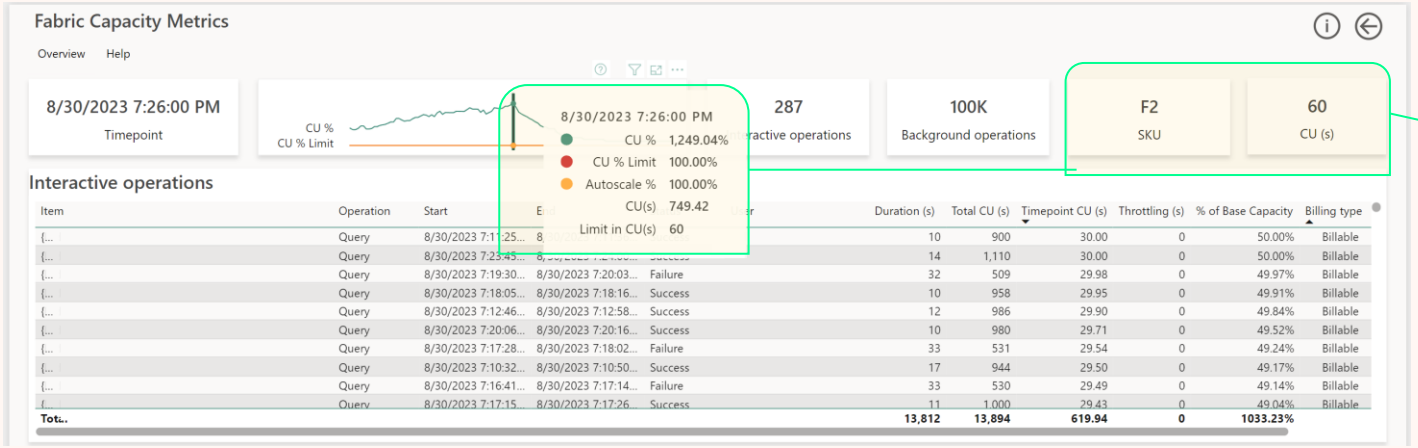
Capacity Planning with Capacity Metrics

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



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

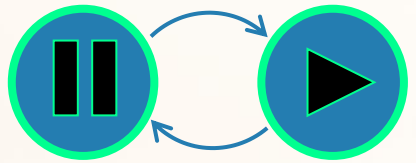
To accommodate a 749 CU(s) load the admin can purchase an F32 capacity providing 960 CU(s) of throughput

Universal Compute Capacities SKU Sizing

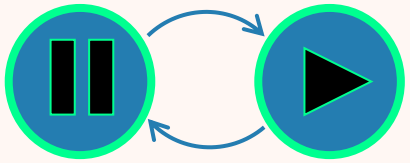
SKU	Capacity Units (CU)	CU's (per 30s)	Power BI SKU	Power BI V-cores
F2	2	60	-	0.25
F4	4	120	-	0.5
F8	8	240	A1	1
F16	16	480	A2	2
F32	32	960	A3	4
F64	64	1920	P1	8
F128	128	3840	P2	16
F256	256	7680	P3	32
F512	512	15360	P4	64
F1024	1024	30720	P5	128
F2048	2048	61440	-	256

Microsoft Fabric
COMMUNITY CONFERENCE

Pausing and Resuming Capacities



Introduction to Pausing and Resuming Capacities



Overview and Benefits

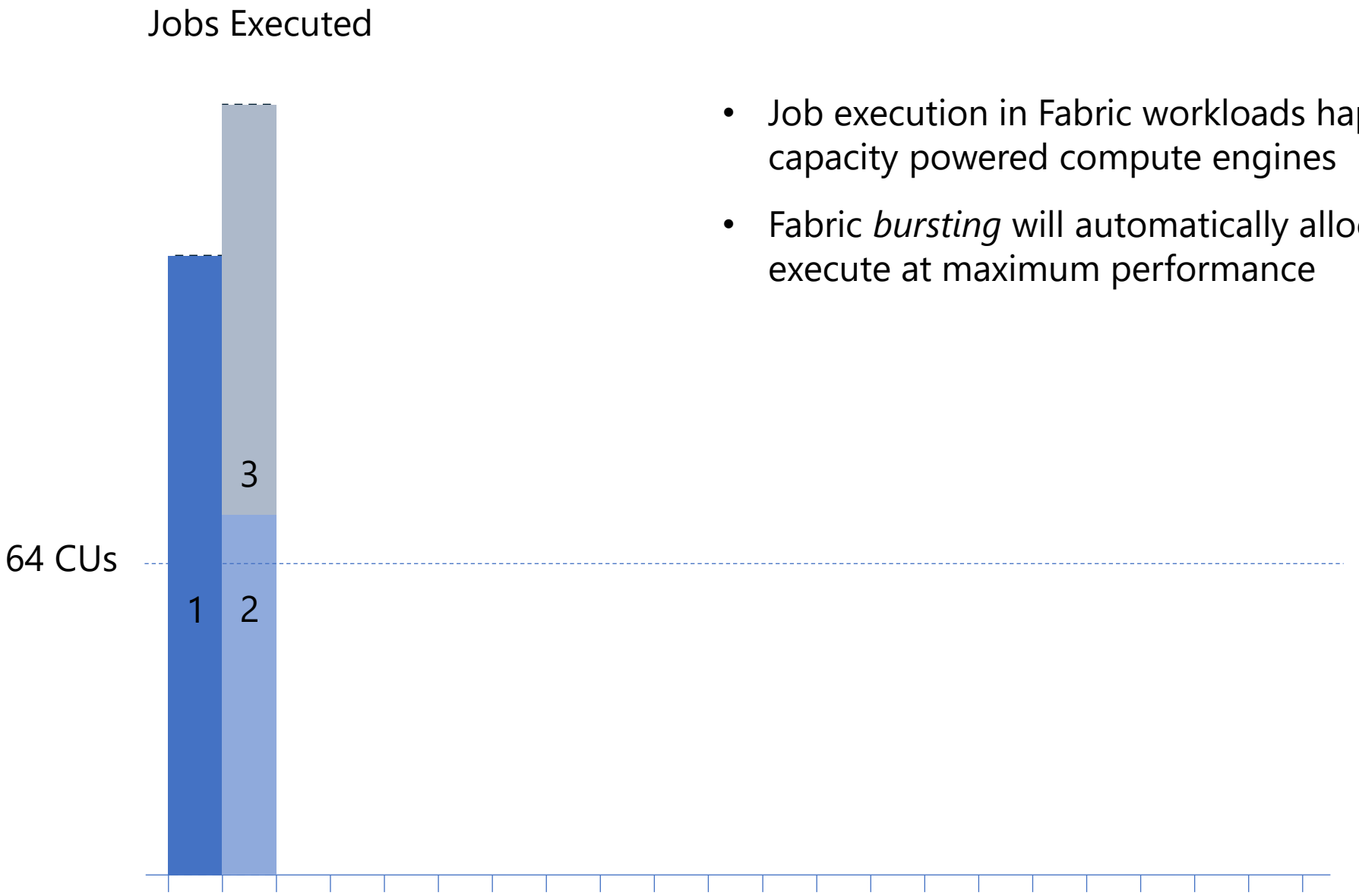
Pause and Resume lets you manage compute costs on F SKU capacities by suspending the execution of all workloads running on the capacity

- When a capacity administrator pauses a capacity:

Workloads stop execution	New requests are not run	Smoothed usage will be reconciled

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

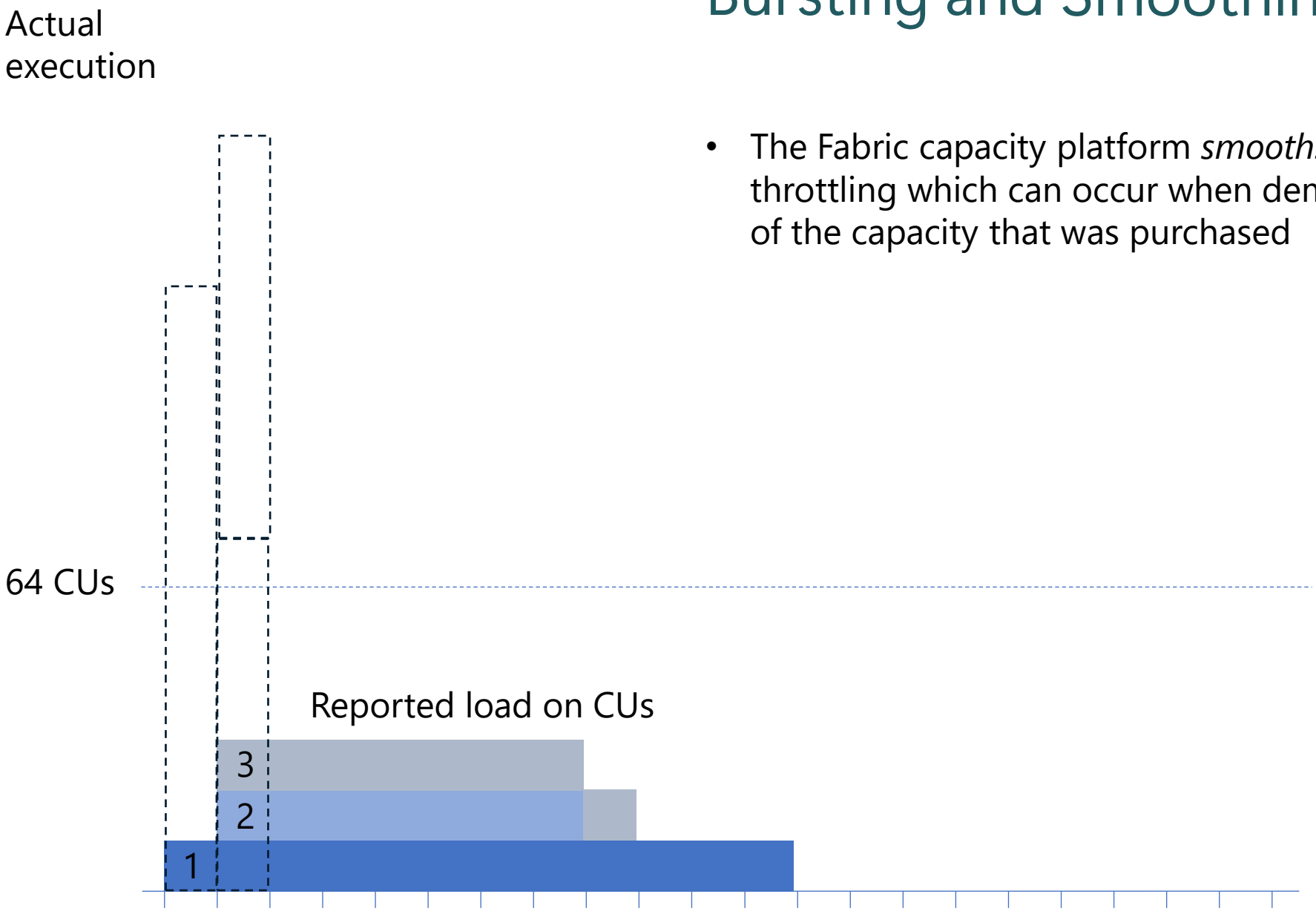
Bursting and Smoothing



- 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

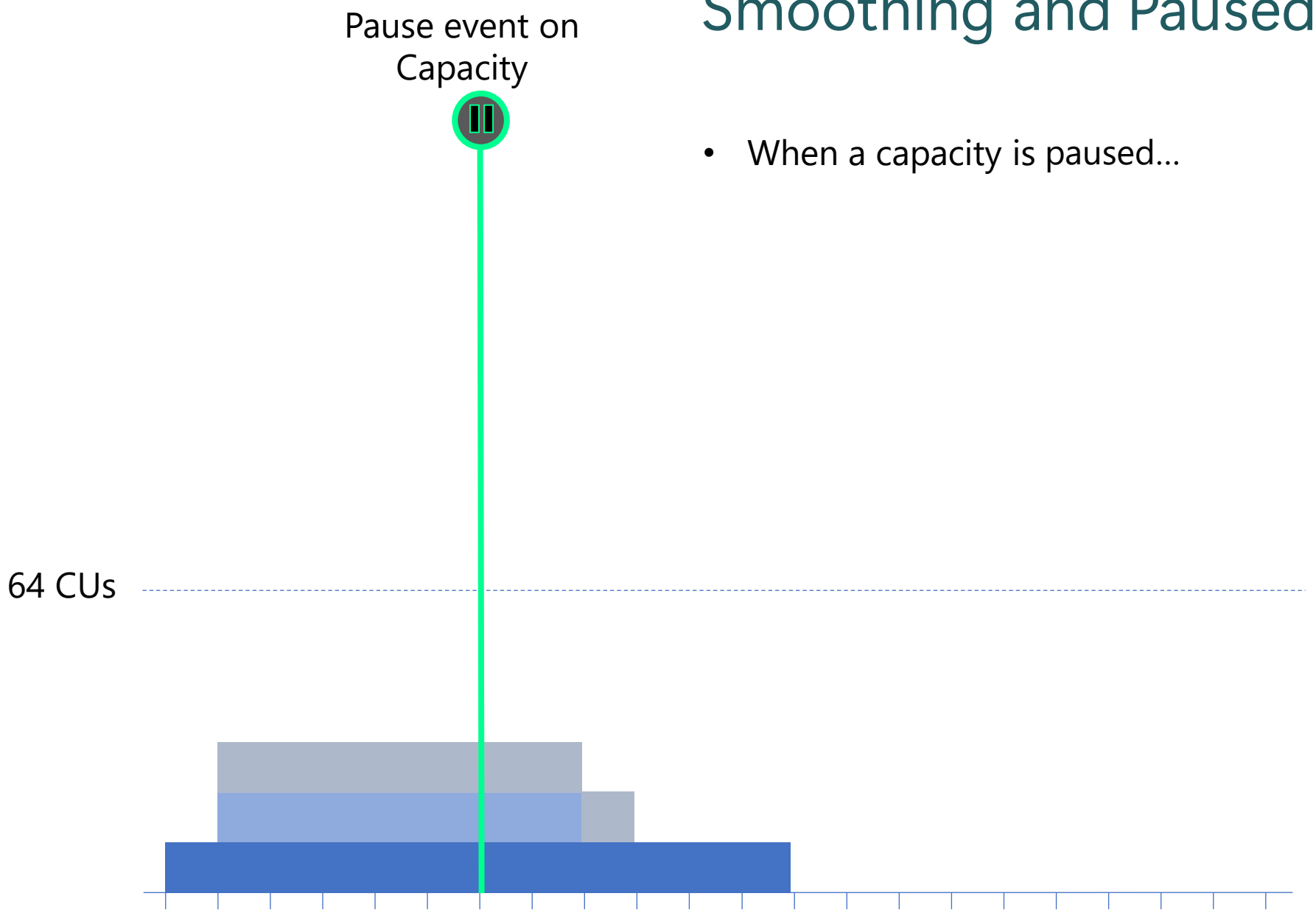
Bursting and Smoothing

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

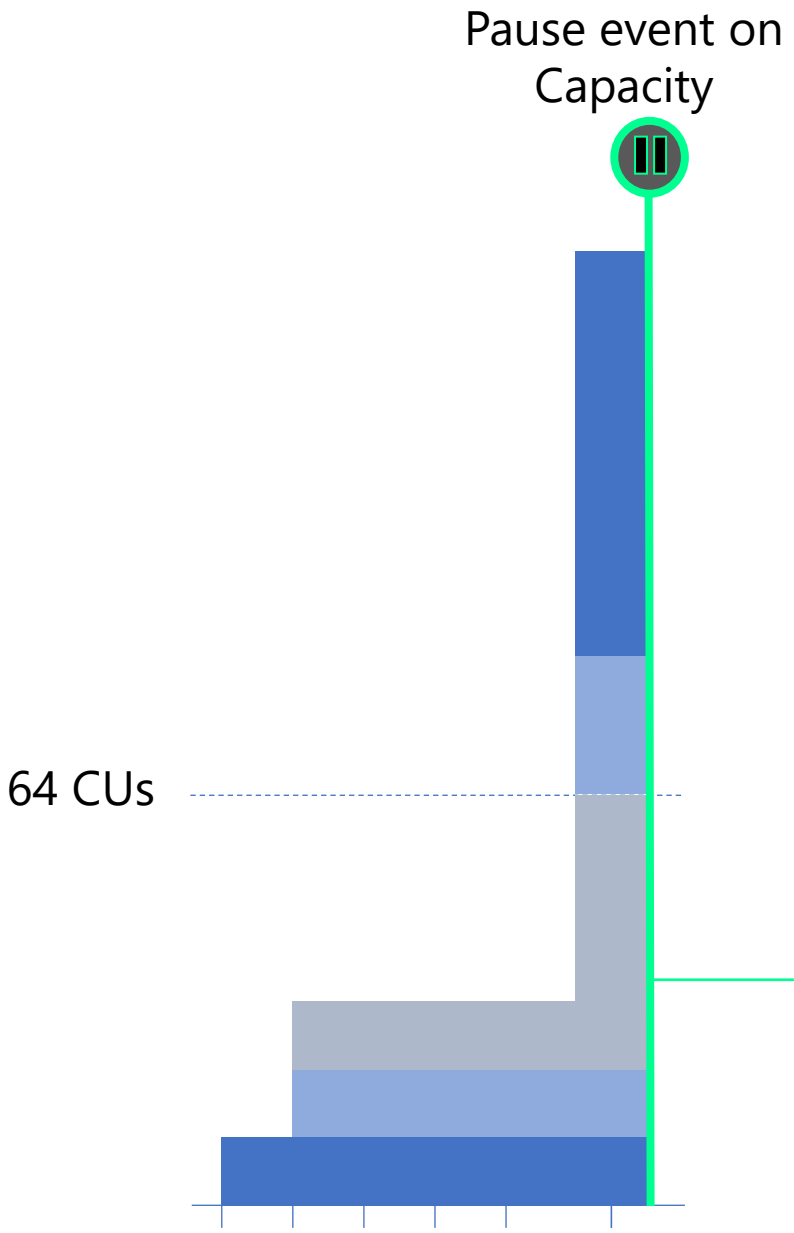


Smoothing and Paused Capacities

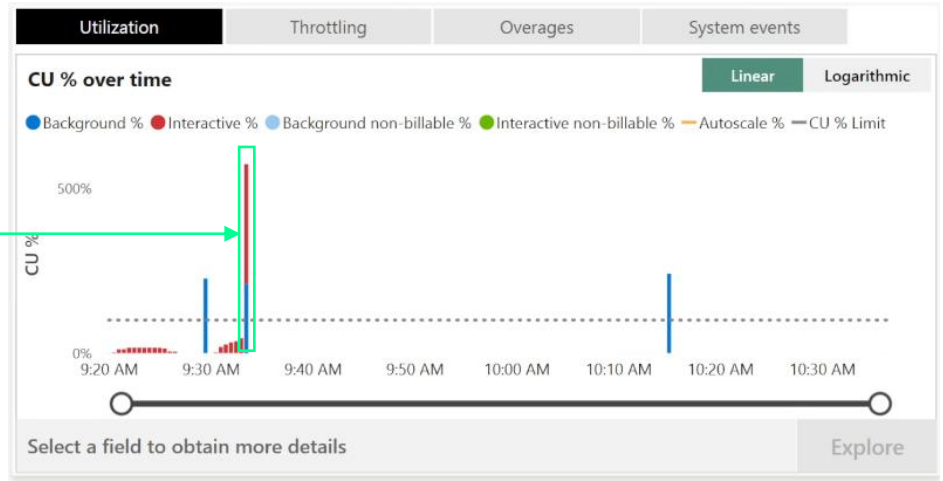
- When a capacity is paused…



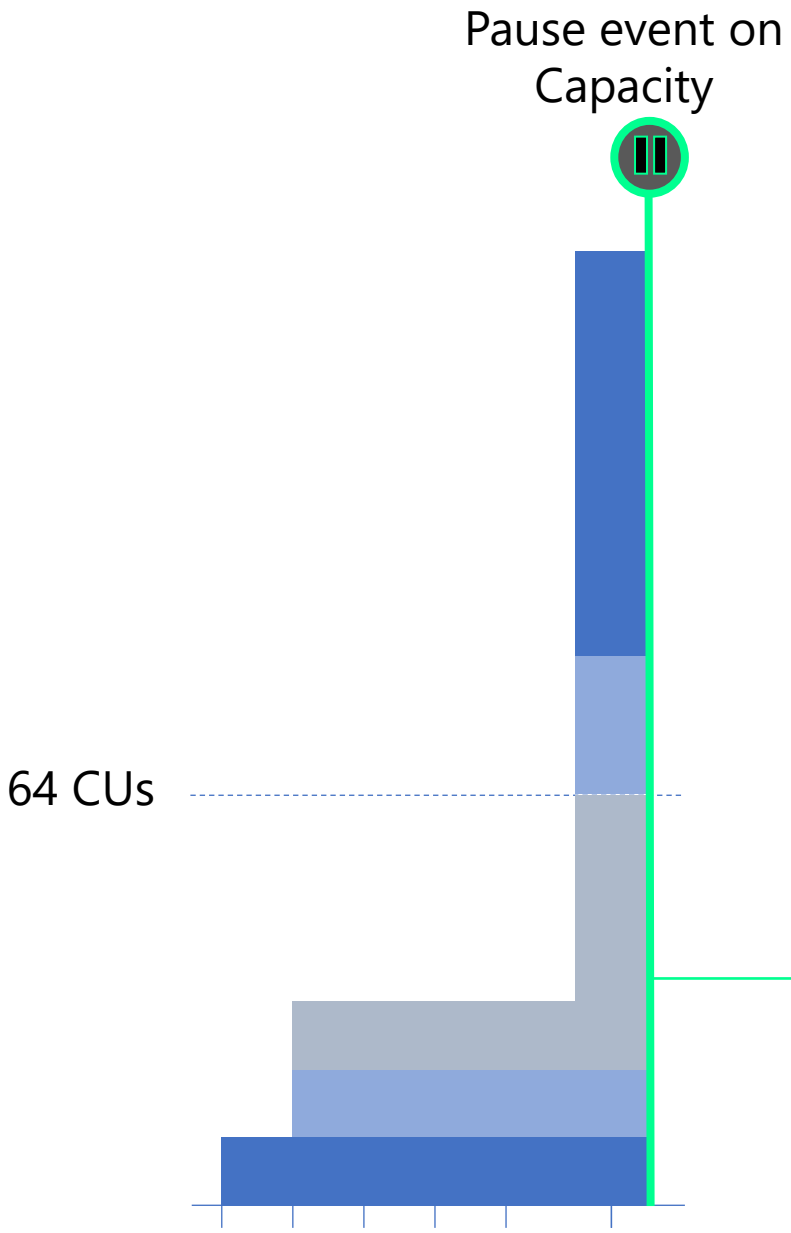
Smoothing and Paused Capacities



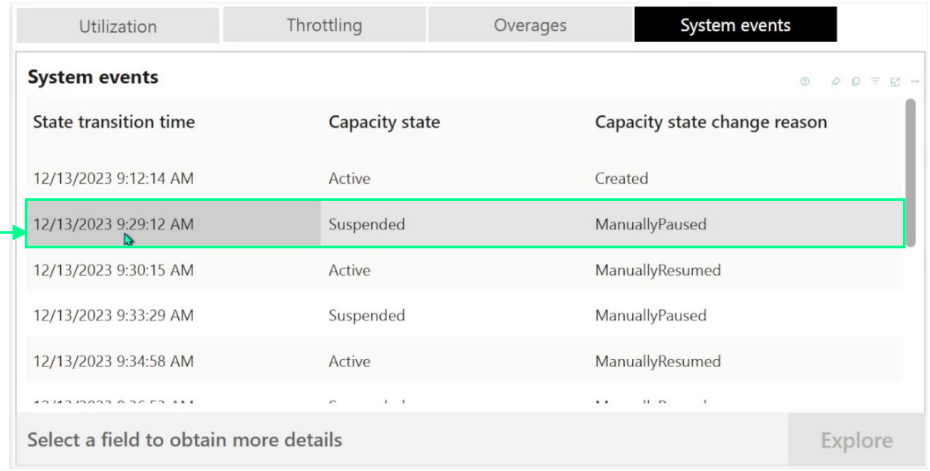
- When a capacity is paused…
- Usage that was smoothed into the future will be “reconciled” and charged against the capacity at the timestamp the capacity was paused
- Reconciled usage will show up as a spike in capacity metrics



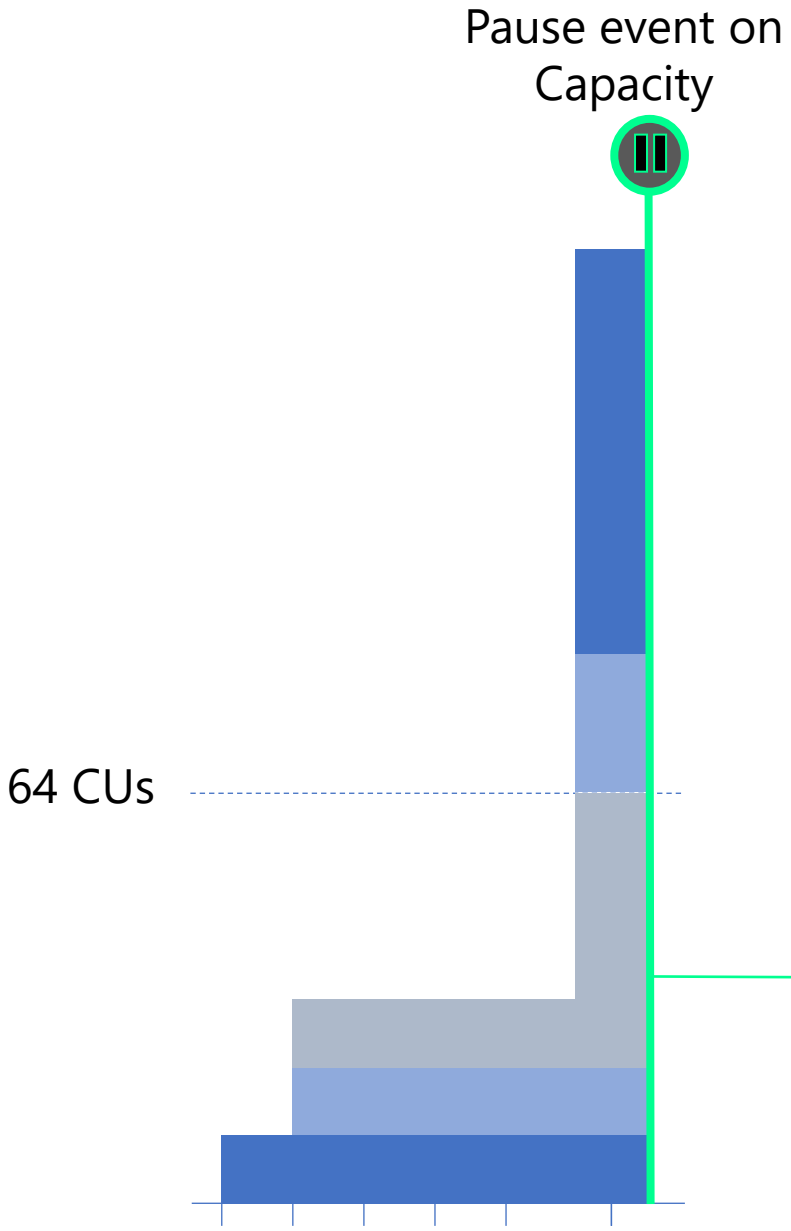
Smoothing and Paused Capacities



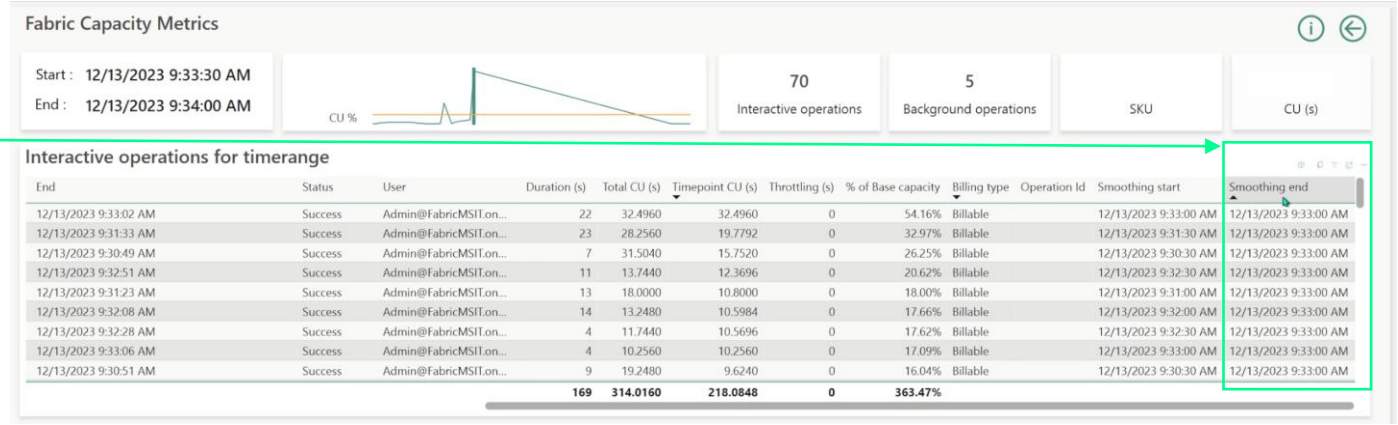
- When a capacity is paused…
- Usage that was smoothed into the future will be “reconciled” and charged against the capacity at the timestamp the capacity was paused
- Pause events can be viewed in the new System events tab



Smoothing and Paused Capacities



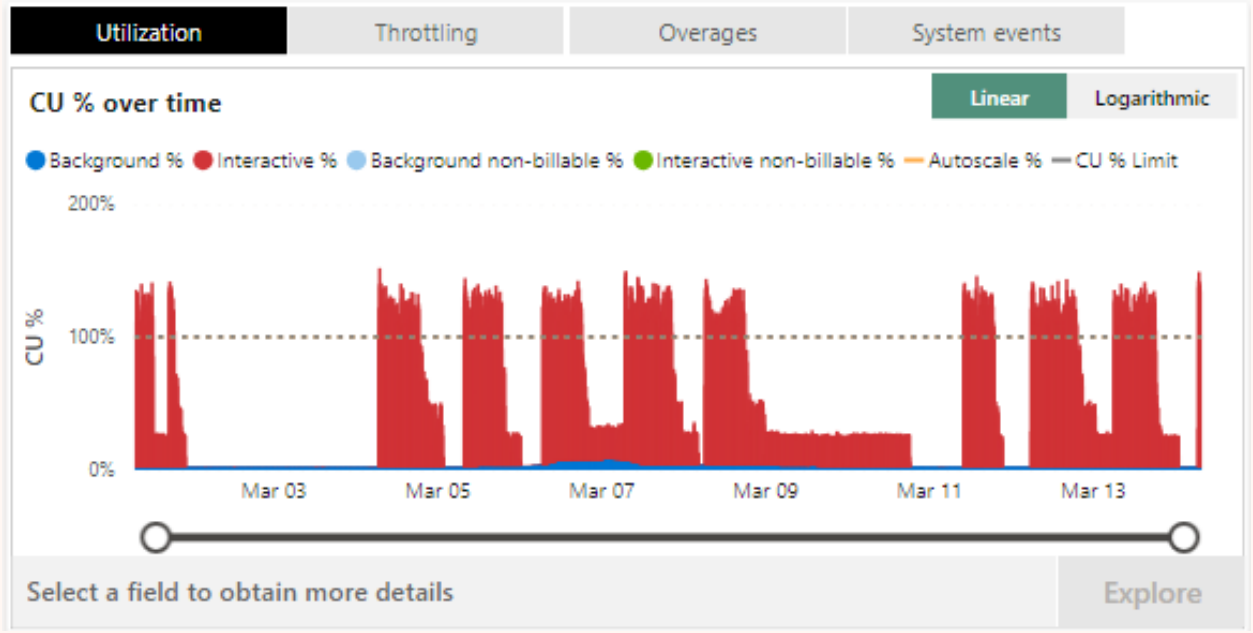
- When a capacity is paused…
- Usage that was smoothed into the future will be “reconciled” and charged against the capacity at the timestamp the capacity was paused
- Pause events timestamp is shown in the smoothing end field in timepoint drill views



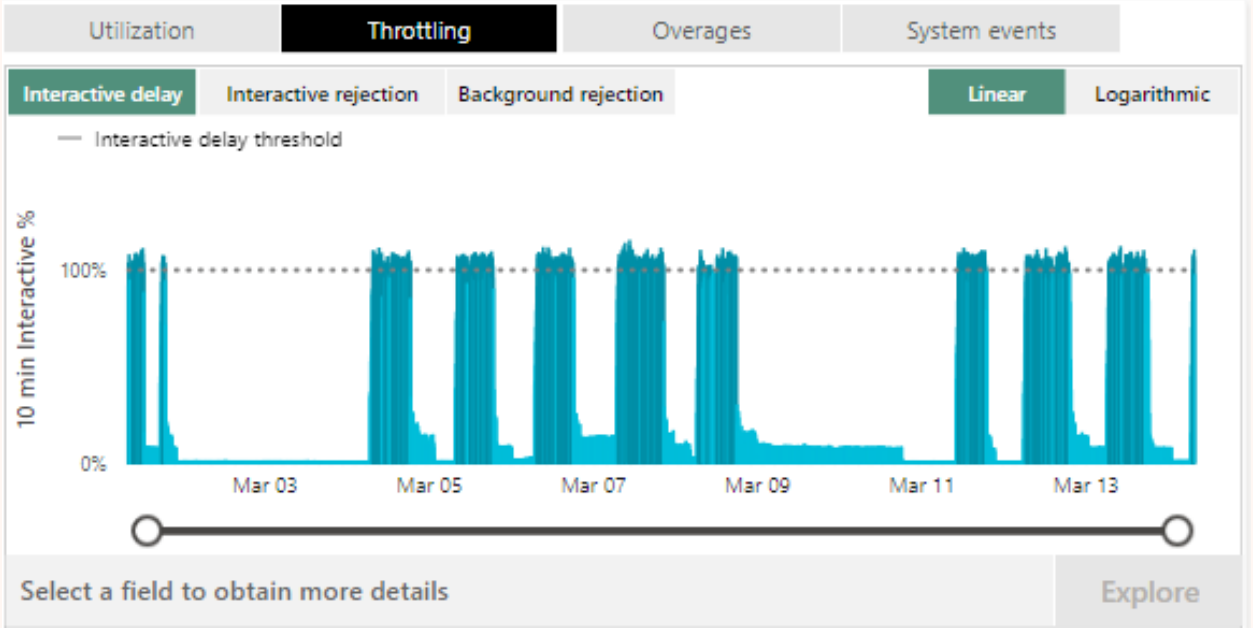
Microsoft Fabric
COMMUNITY CONFERENCE

Bonus: Tips and Tricks for capacity management and monitoring

My capacity is being throttled! What can I do?



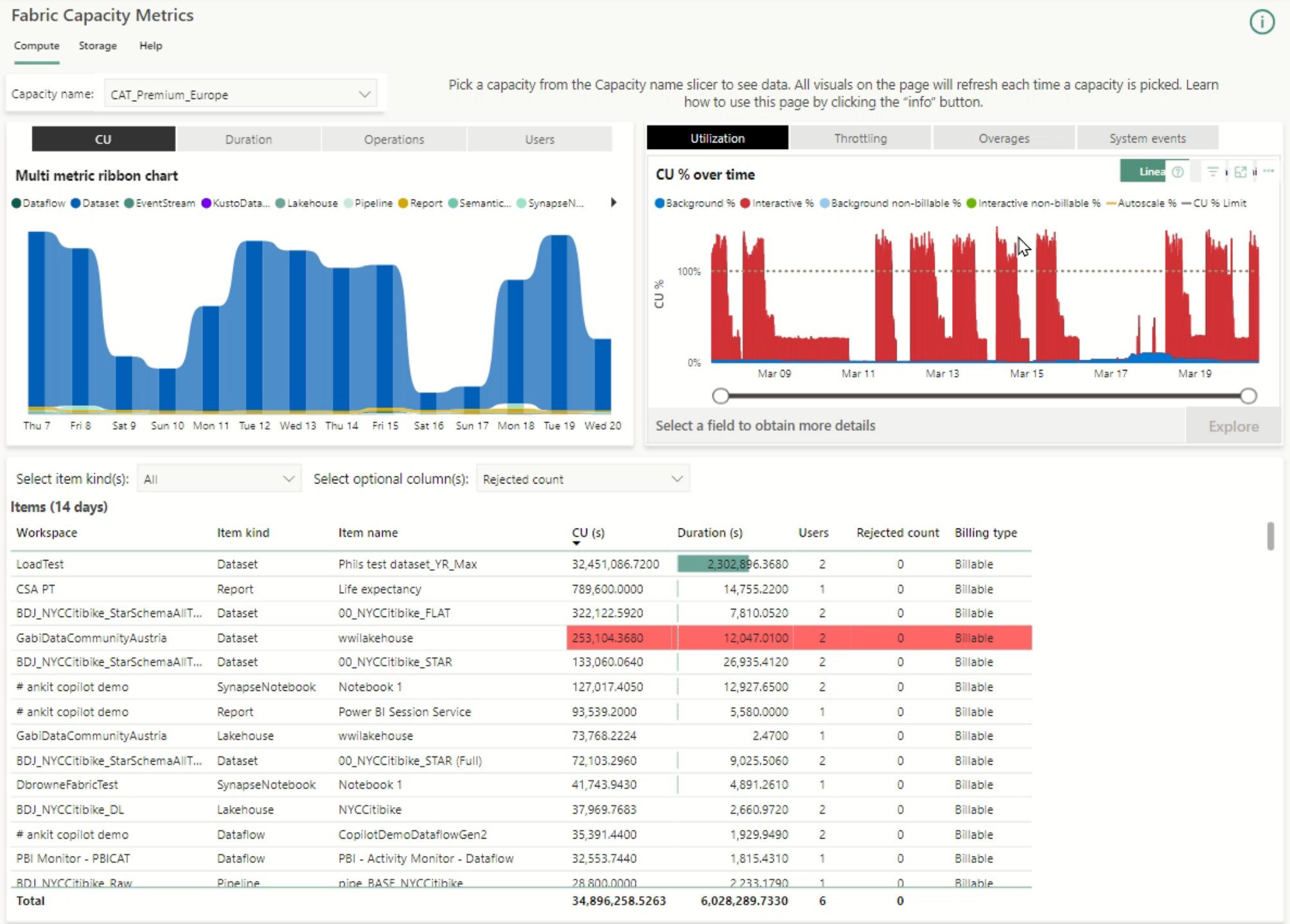
Over 100% utilization doesn't always result in throttling



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

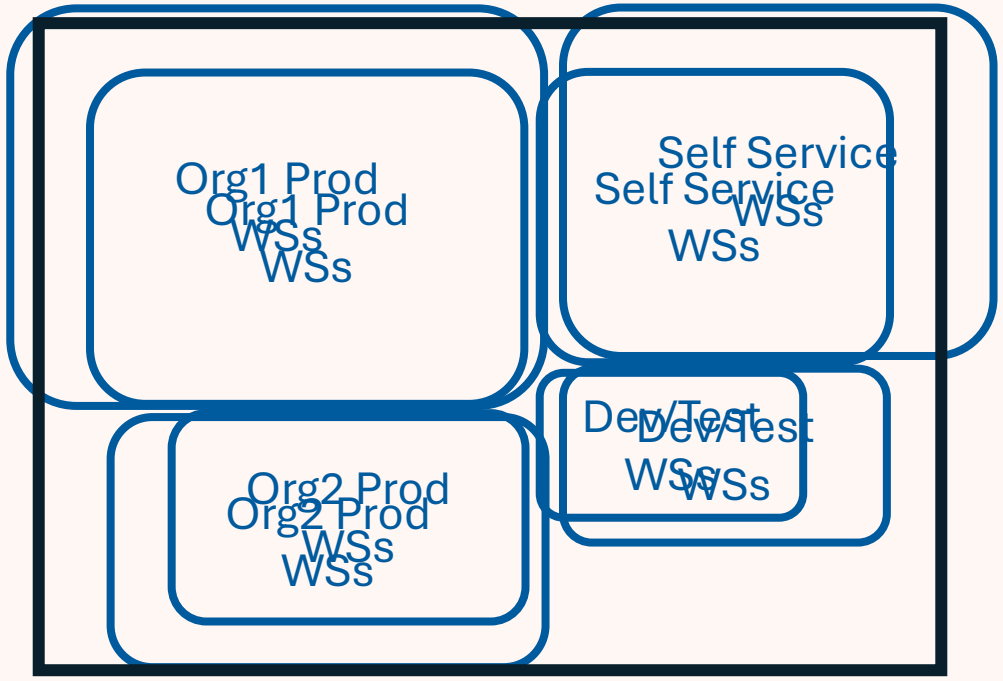
Metrics App Tips



Workspace	Item kind	Item name	CU (s)	Duration (s)	Users	Rejected count	Billing type
LoadTest	Dataset	Phils test dataset_YR_Max	32,451,086.7200	2,302,896.3680	2	0	Billable
CSA PT	Report	Life expectancy	789,600.0000	14,755.2200	1	0	Billable
BDJ_NYCCitibike_StarSchemaAllT...	Dataset	00_NYCCitibike_FLAT	322,122.5920	7,810.0520	2	0	Billable
GabiDataCommunityAustria	Dataset	wwilakehouse	253,104.3680	12,047.0100	2	0	Billable
BDJ_NYCCitibike_StarSchemaAllT...	Dataset	00_NYCCitibike_STAR	133,060.0640	26,935.4120	2	0	Billable
# ankit copilot demo	SynapseNotebook	Notebook 1	127,017.4050	12,927.6500	2	0	Billable
# ankit copilot demo	Report	Power BI Session Service	93,539.2000	5,580.0000	1	0	Billable
GabiDataCommunityAustria	Lakehouse	wwilakehouse	73,768.2224	2.4700	1	0	Billable
BDJ_NYCCitibike_StarSchemaAllT...	Dataset	00_NYCCitibike_STAR (Full)	72,103.2960	9,025.5060	2	0	Billable
DbrowneFabricTest	SynapseNotebook	Notebook 1	41,743.9430	4,891.2610	1	0	Billable
BDJ_NYCCitibike_DL	Lakehouse	NYCCitibike	37,969.7683	2,660.9720	2	0	Billable
# ankit copilot demo	Dataflow	CopilotDemoDataflowGen2	35,391.4400	1,929.9490	2	0	Billable
PBI Monitor - PBICAT	Dataflow	PBI - Activity Monitor - Dataflow	32,553.7440	1,815.4310	1	0	Billable
BDJ_NYCCitibike_Raw	Pipeline	pipe_BASE_NYCCitibike	28,800.0000	2,233.1790	1	0	Billable
Total			**34,896,258.5263**	**6,028,289.7330**	**6**	**0**	

Note: Didn't even go to Timepoint Detail page. Useful for usernames and individual operation time/CU

When Capacity Units Run Out
Option 1 – Optimize



WSs = Workspaces

Capacity

Approach

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

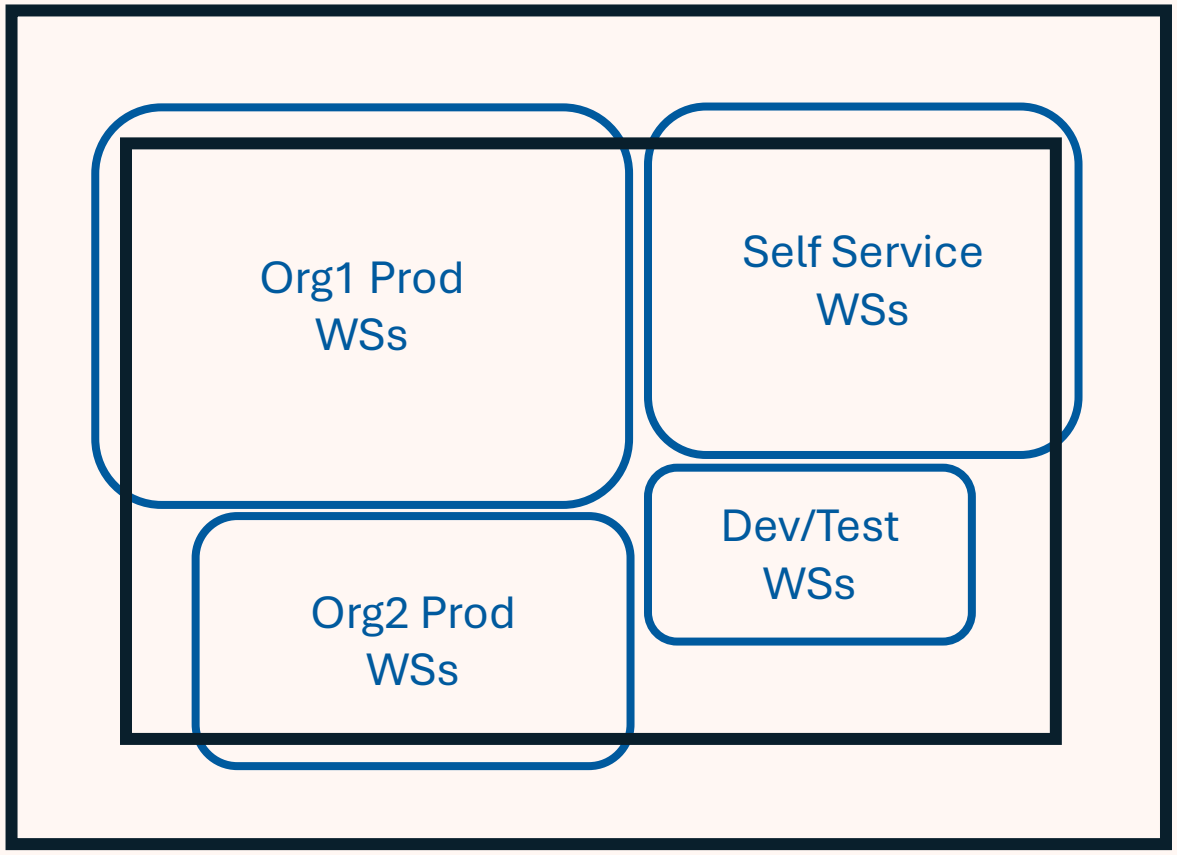
Pros

- Avoids increased cost
- Learning carries over to future content

Cons

- Can be difficult/time consuming

When Capacity Units Run Out
Option 2 – 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)

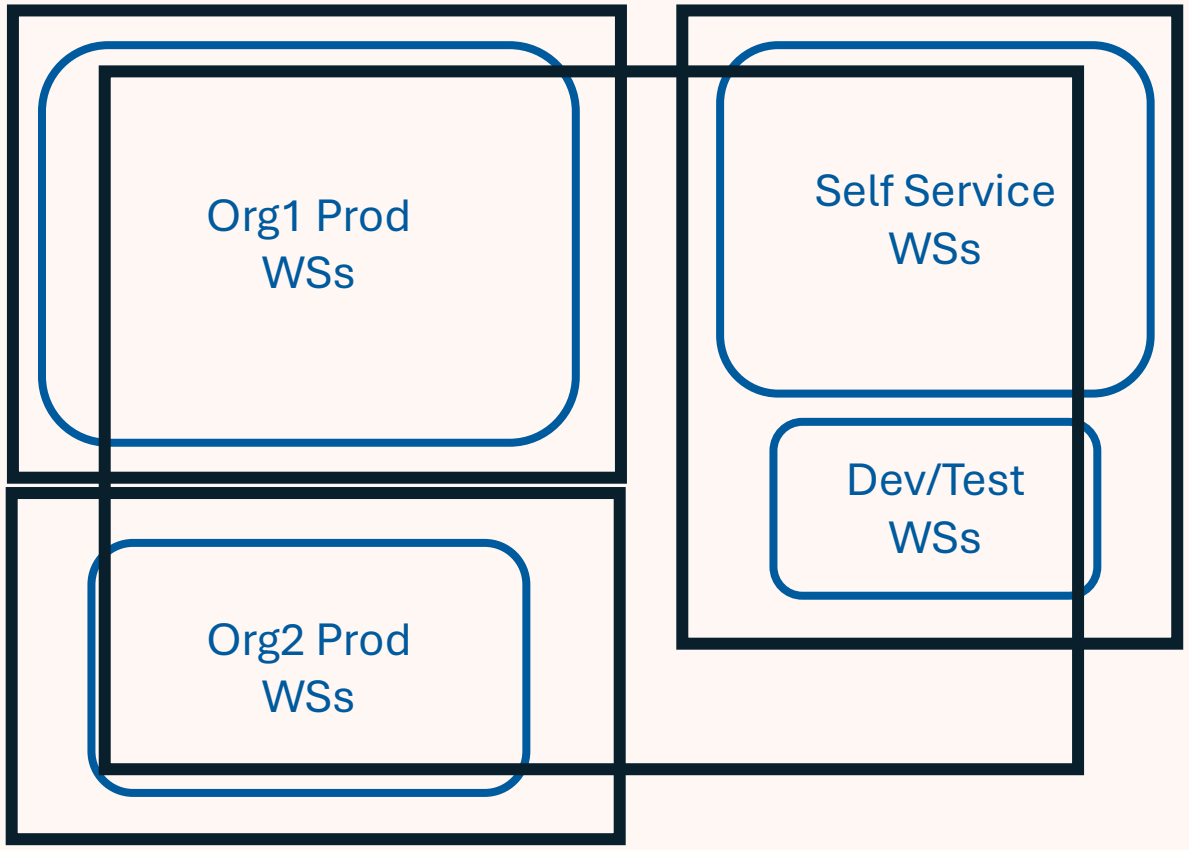
Pros

- 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 3 – Scale Out

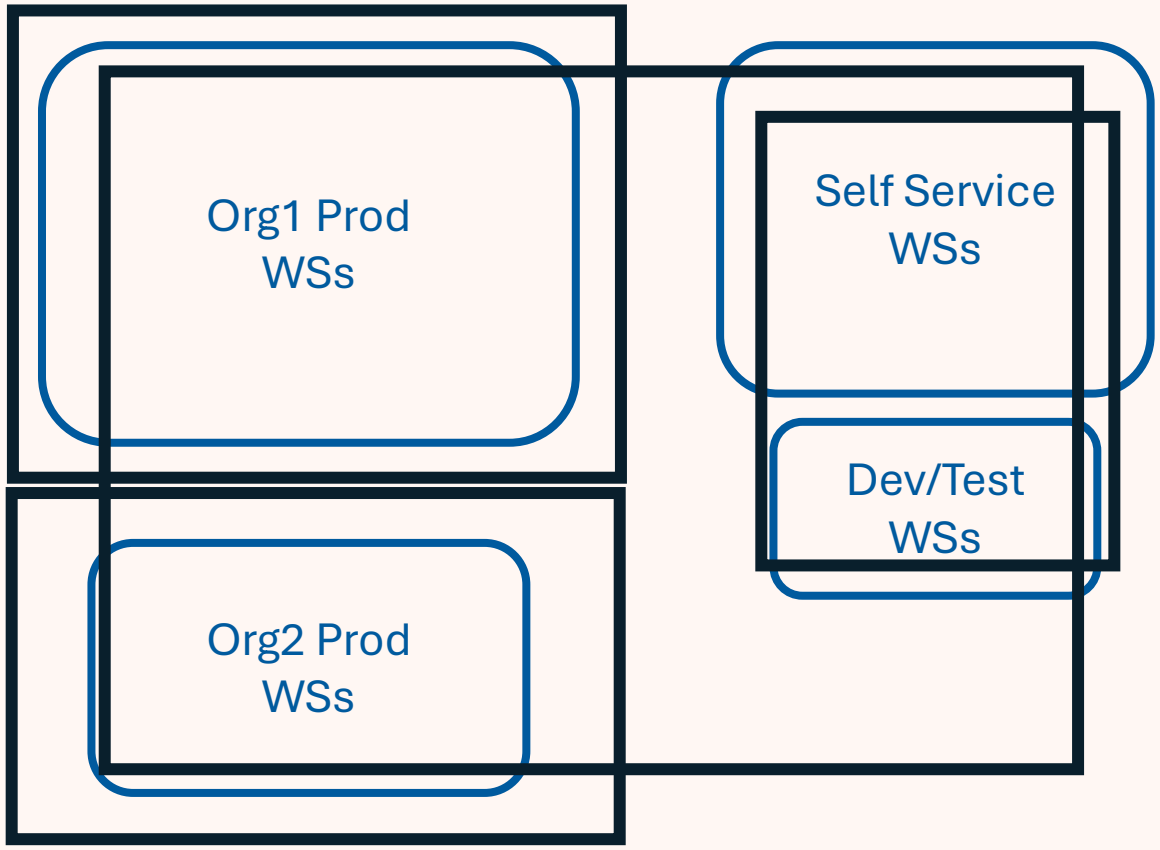


Options
- Create multiple smaller P or F SKUs based on organization, type of work, etc.

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 4 – Isolate

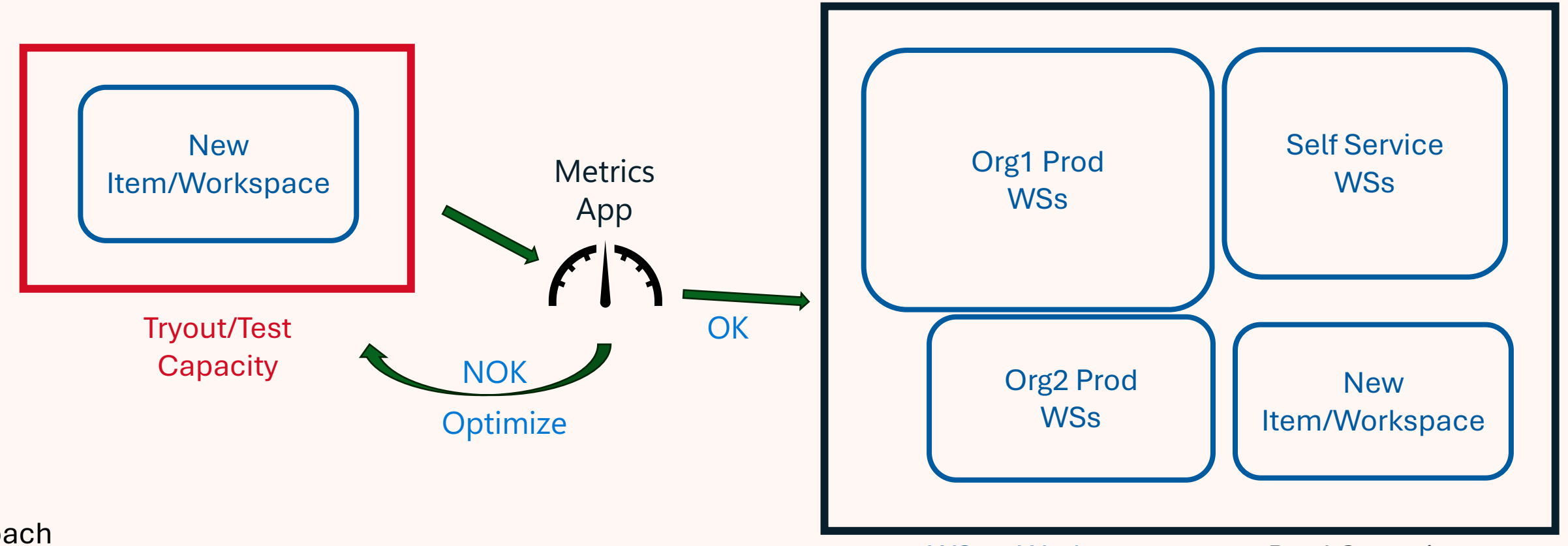


Approach
- Provide isolated capacity for key items built by experienced developers

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

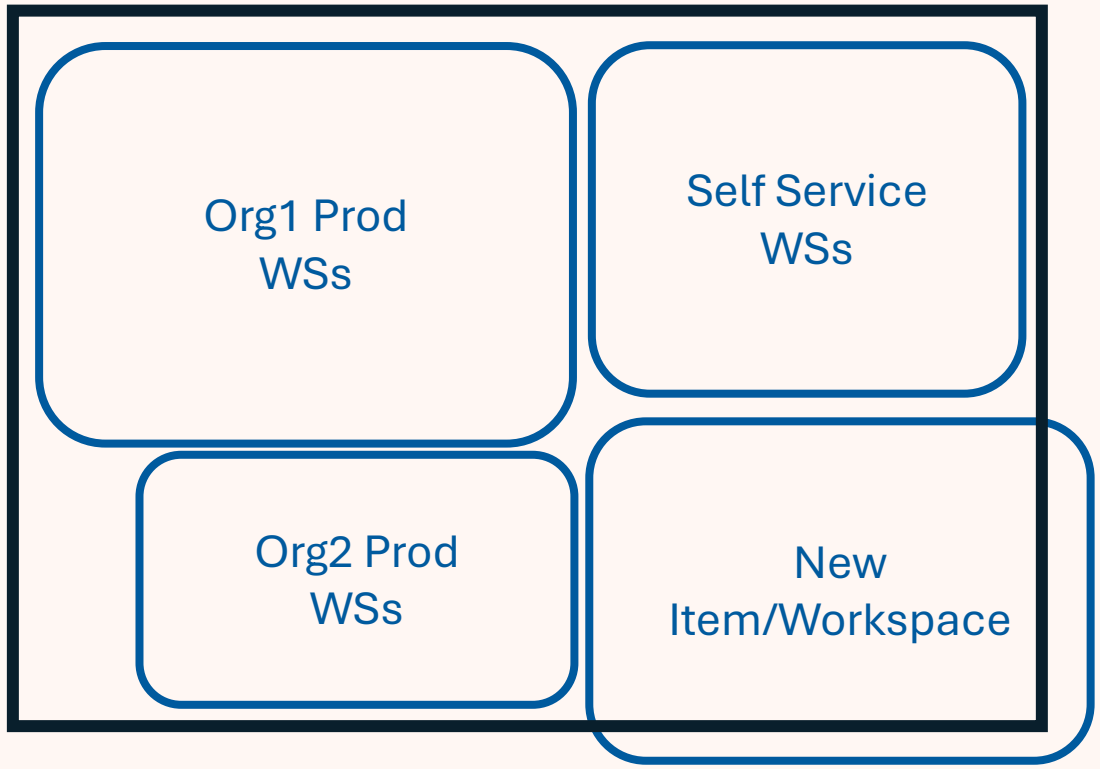
Isolation Strategy #4a – Tryout Capacity



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

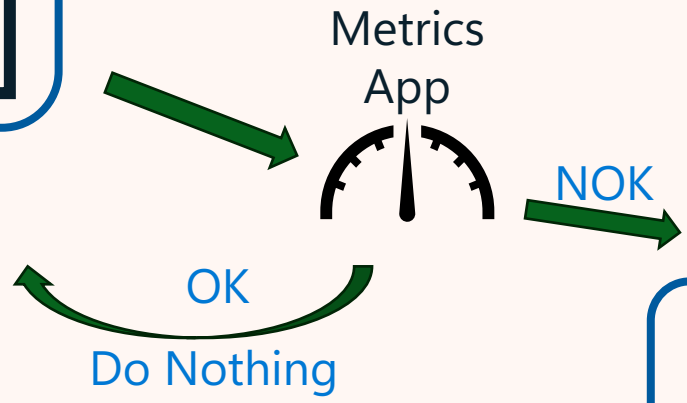


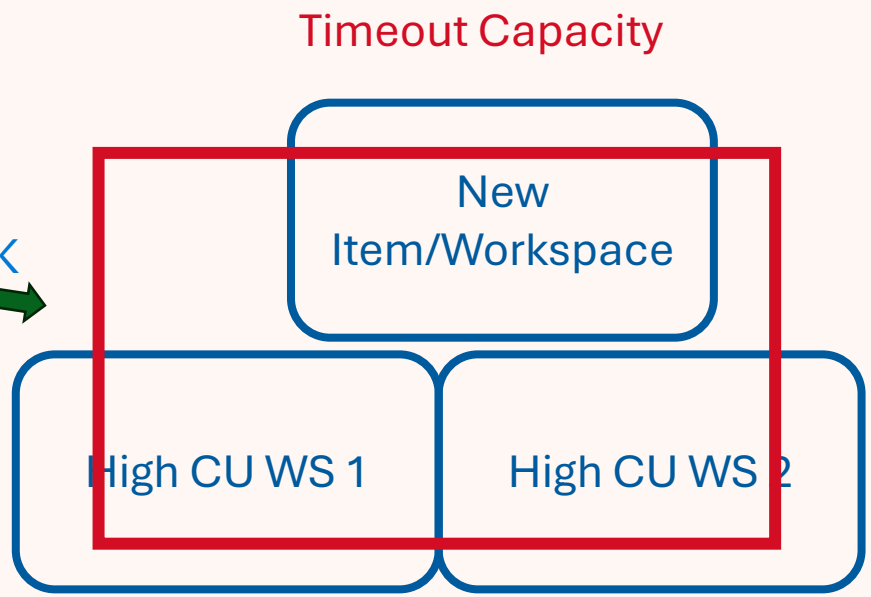
WSs = Workspaces

Prod Capacity

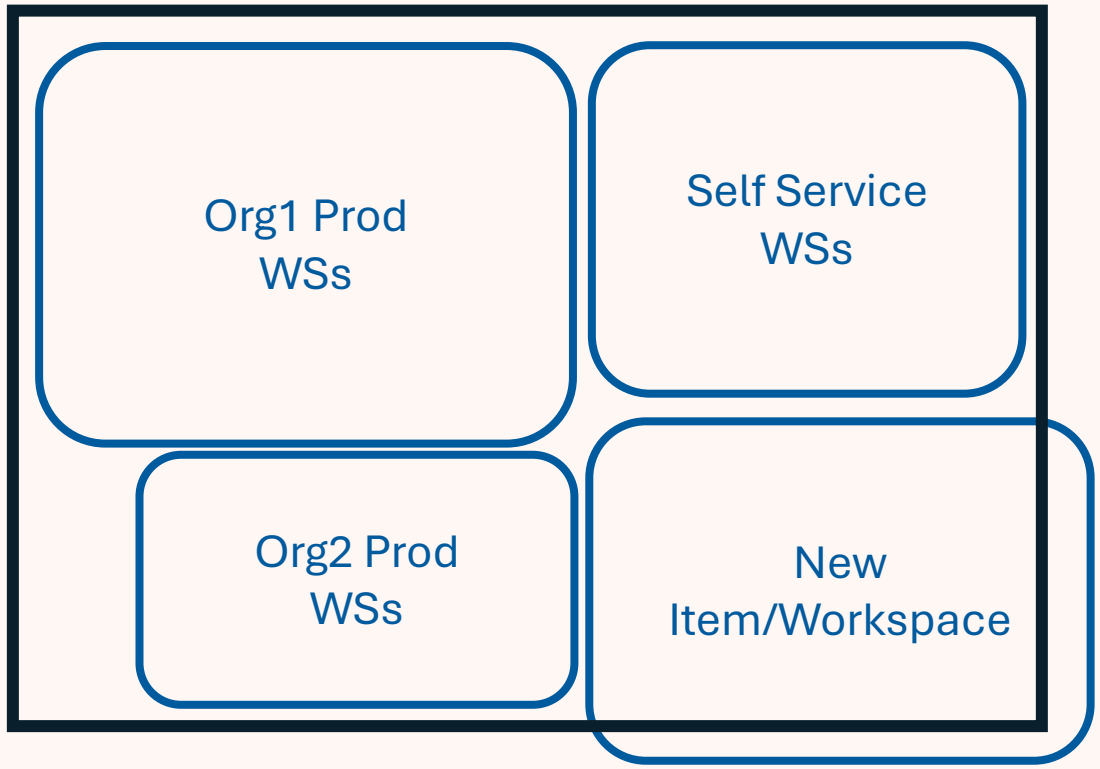
Approach

- 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

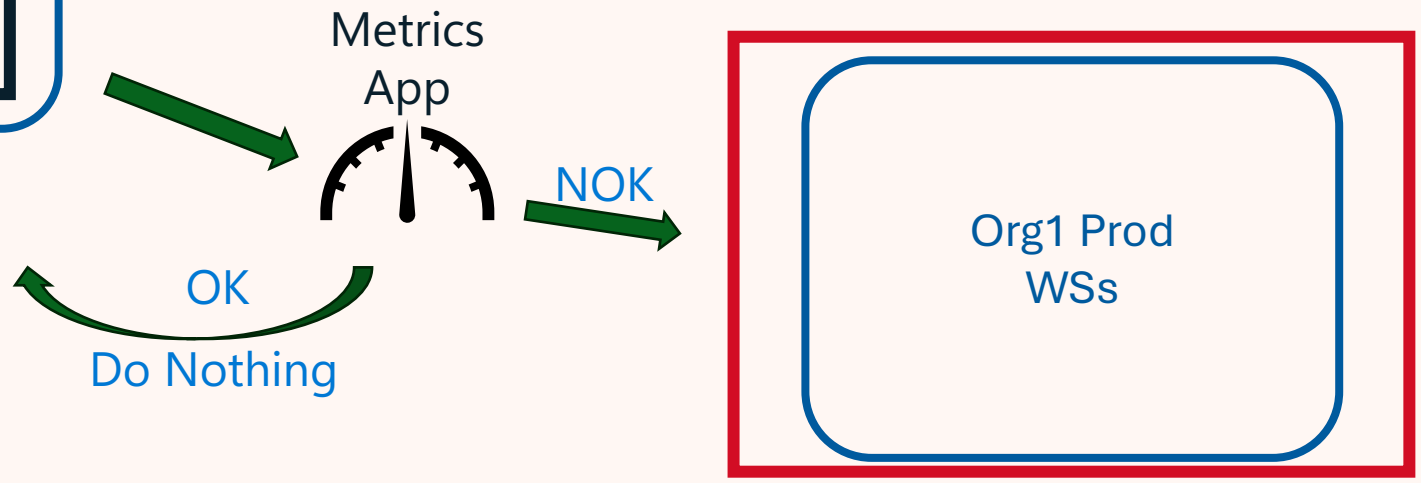


WSs = Workspaces

Prod 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

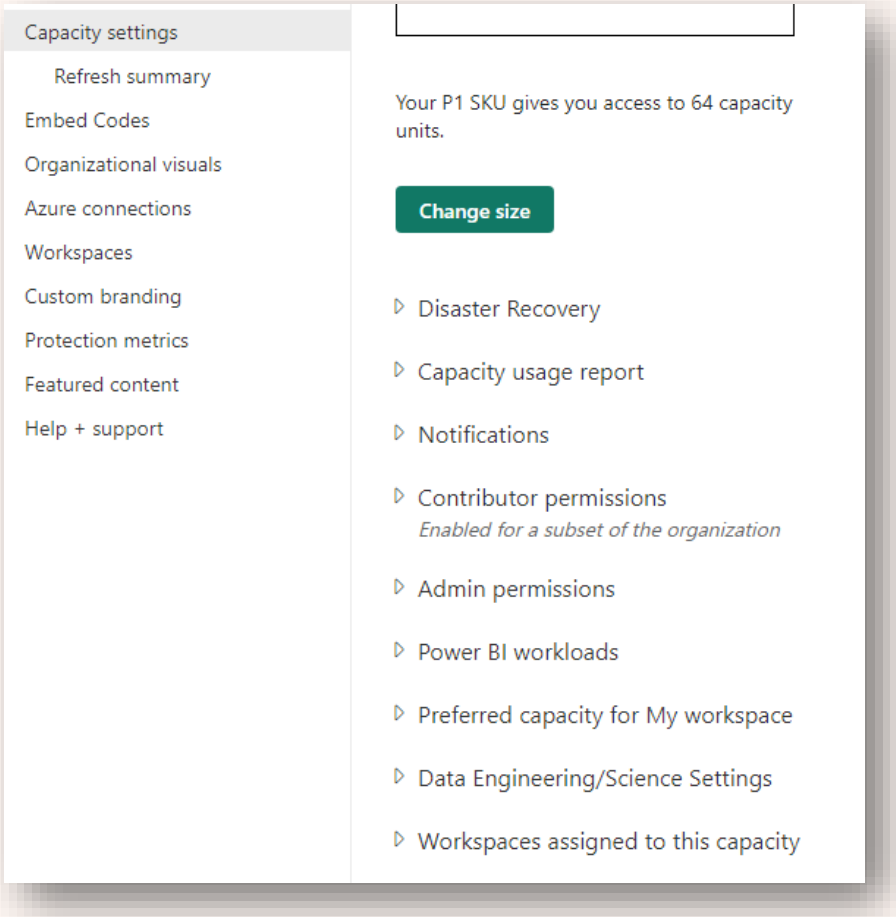


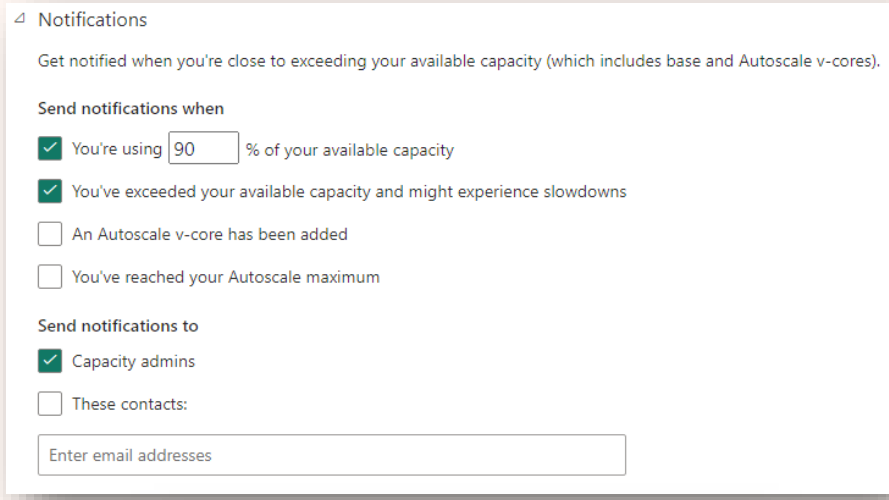
Recommendations for Cost/CU Savings

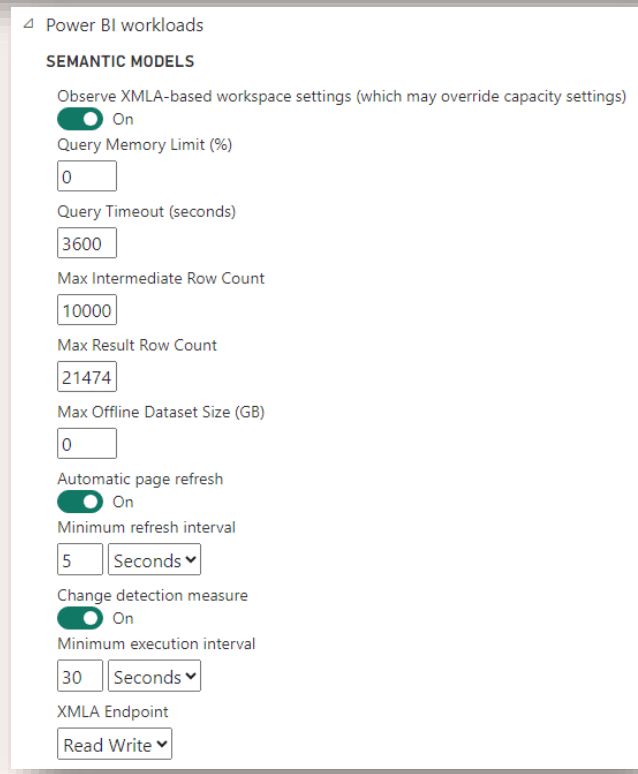
- Invest in education, knowledge/best practice sharing, COE, etc. for creators and consumers (proactive optimization)
- Avoid data/report sprawl (leverage certified/promoted models, OneLake shortcuts, etc.)
- Leverage a multi-capacity strategy (isolate, tryout, timeout, etc.)
- Right size your capacities 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)



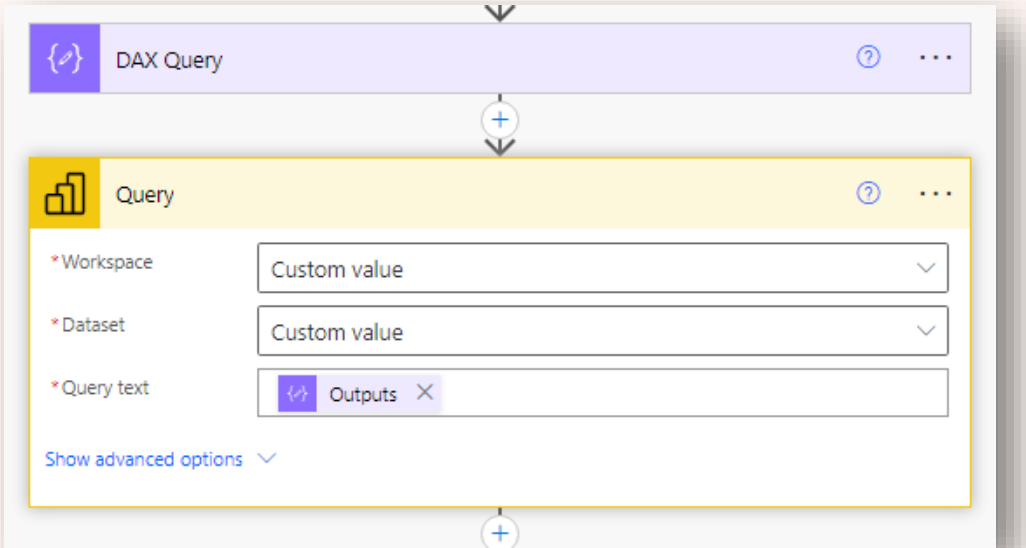




Custom Solutions

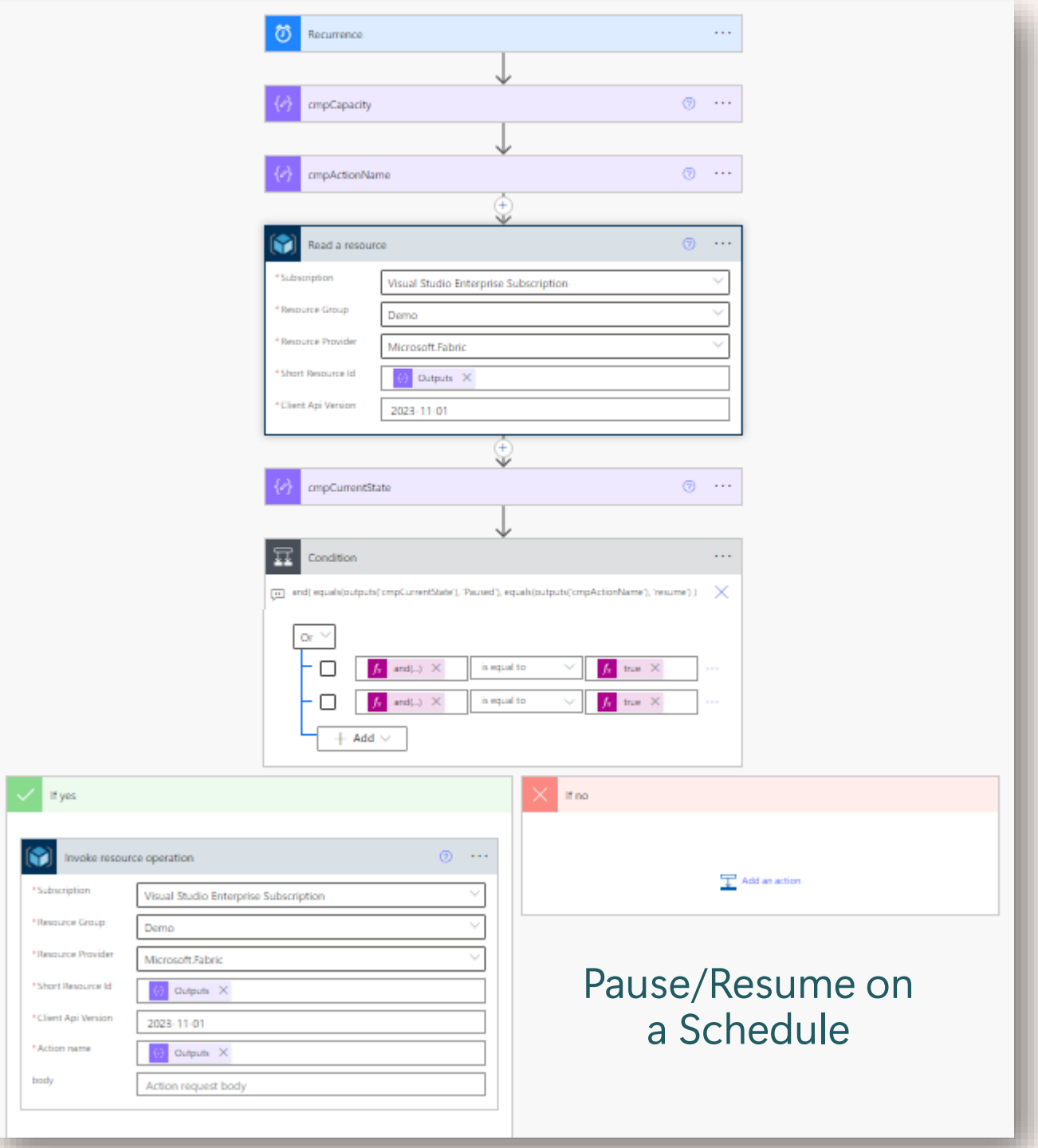
- Modify the Metrics App to meet your needs
- Build a custom report off the semantic model
- Send DAX queries to the metrics app semantic model in your own solution
  - Power Automate, Notebook (SemPy), PowerShell, etc.
  - Get throttling % values (Interactive Delay, Interactive Rejection, and/or Background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

Incorporate Metrics App queries into custom solutions



Collect data from multiple capacities and store it long term

```python
# Get max date from current delta table (to avoid loading duplicate days)
try:
    df_max = spark.sql(f'''
    SELECT MAX(Date) as MaxDate
    FROM throttling;
    ''')
    maxdate = df_max.first()['MaxDate']
except:
    maxdate = datetime.today() + timedelta(days=-6)
maxdateforDAX = maxdate.strftime('%Y,%m,%d')

if maxdate.date() < (datetime.today() + timedelta(days=-1)).date():

    # Get data for each capacity, write daily csv and append delta
    for capacity in lst_capacities:
        querytext = '''\
                    DEFINE
                    MPARAMETER 'CapacityID' = "{capID}"
                    VAR yesterday =
                        FILTER(ALL('Dates'[Date] ), 'Dates'[Date] < TODAY() && 'Dates'[Date] > DATE({MD}) )

                    EVALUATE
                    SUMMARIZECOLUMNS(
                        'Dates'[Date],
                        'TimePoints'[Start of Hour],
                        yesterday,
                        "IntDelay", ROUND( 'All Measures'[Dynamic InteractiveDelay %] * 100, 2 ),
                        "IntReject", ROUND( 'All Measures'[Dynamic InteractiveRejection %] * 100, 2 ),
                        "BackReject", ROUND( 'All Measures'[Dynamic BackgroundRejection %] * 100, 2 )
                    )
                    '''.format(capID=capacity, MD=maxdateforDAX)
        df_throttling = fabric.evaluate_dax(workspace=MetricsWS, dataset=MetricsModel, dax_string=querytext)
        if len(df_throttling) >= 1:
            df_throttling.columns = df_throttling.columns.str.replace(r'(.*\[)|(\].*)', '', regex=True)
            df_throttling.columns = df_throttling.columns.str.replace(' ', '_')
            df_throttling['capacityId'] = capacity
            filename = capacity + '_throttling_' + (datetime.today()).strftime('%Y%m%d') + '.csv'
            df_throttling.to_csv("/lakehouse/default/Files/ThrottlingData/" + filename)
            spk_throttle = spark.createDataFrame(df_throttling)
            spk_throttle.write.mode("append").format("delta").option("overwriteSchema", "true").saveAsTable('Throttling')
```



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)

DIY Autoscale – Fabric Notebook

(Bret Myers)

Set SKU Ranges and Values

```
# Parameters to be passed in from pipeline.
minSku = 'F2' # min sku size we can scale down to
maxSku = 'F128' # max sku size we can scale up to
utilizationTolerance = 90 # Percentage of CU used to st
capacityName = 'fabricbamdemo' #capacity name to be mon
subscriptionId = '[redacted]'
metricsAppWorkspaceName = 'WS_FabricCapacityMetrics' #
metricsAppModelName = 'Fabric Capacity Metrics' # name
alertEmail = '' # email address to send alert that we s
```

Get credentials

```
tenantId = mssparkutils.credentials.getSecret('keyVaultEndpoint', 'secretName_tenantId')
clientId = mssparkutils.credentials.getSecret('keyVaultEndpoint', 'secretName_clientId')
secret = mssparkutils.credentials.getSecret('keyVaultEndpoint', 'secretName_clientSecret')

api_pbi = 'https://analysis.windows.net/powerbi/api/.default'
api_azuremgmt = 'https://management.core.windows.net/.default'
```

Not all code shown

FabricTools/CapacityAutoScale at main · bretamyers/FabricTools · GitHub

Query metrics app model

```
from azure.identity import ClientSecretCredential
import requests, json, math
from pyspark.sql.functions import explode

auth = ClientSecretCredential(tenant_id=tenantId, client_id=clientId, client_secret=secret)
access_token = auth.get_token(api_pbi)

header = {'Authorization': f'Bearer {access_token.token}', 'Content-type': 'application/json'}

body = {
  "queries": [
    {
      "query": f"""
        DEFINE
          MPARAMETER 'CapacityID' = "{capacityId}"

          VAR __DS0FilterTable =
            FILTER(
              KEEPFILTERS(VALUES('TimePoints'[TimePoint])),
              'TimePoints'[TimePoint] >= NOW() - 1
            )

          VAR __DS0FilterTable2 = TREATAS({{"{capacityId}"}}, 'Capacities'[capacityId])

          VAR __DS0Core =
            SELECTCOLUMNS(
              KEEPFILTERS(
                FILTER(
                  KEEPFILTERS(
                    SUMMARIZECOLUMNS(
                      'Capacities'[capacityId],
                      'Items'[Billable type]
```

Change SKU Size

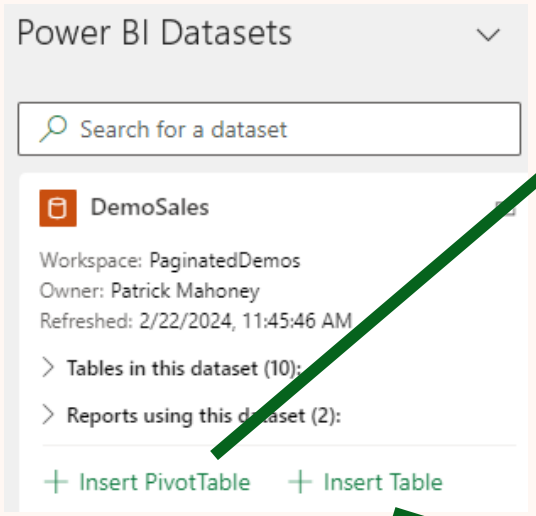
```
    url = f'https://management.azure.com/subscriptions/{subscriptionId}/resourceGroups/{res

    body = {
                "sku": {
                    "name": f"{scaleSku}",
                    "tier": "Fabric"
                }
            }

    response = requests.patch(url, headers=header, data=json.dumps(body))
```

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 Table[Email] = USERPRINCIPALNAME()
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].[To

✅ 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 [
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 __D
01:28:41	DAX	1,516ms	Power BI...	DemoSales	DEFINE VAR __D
01:28:40	DAX	16ms	Power BI...	DemoSales	DEFINE VAR __D
01:28:34	DAX	156ms	Power BI...	DemoSales	DEFINE VAR __D
01:28:33	DAX	16ms	Power BI...	DemoSales	DEFINE VAR __D
01:28:31	DAX	0ms	Power BI...	DemoSales	DEFINE VAR __D
01:28:30	DAX	141ms	Power BI...	DemoSales	DEFINE VAR __D
01:28:15	DAX	2,047ms	Power BI...	DemoSales	DEFINE VAR __D
01:28:11	DAX	1,797ms	Power BI...	DemoSales	DEFINE VAR __D
01:28:08	DAX	594ms	Power BI...	DemoSales	DEFINE VAR __D
01:27:56	DAX	281ms	Power BI...	DemoSales	DEFINE VAR __D
01:27:50	DAX	16ms	Power BI...	DemoSales	DEFINE VAR __D

✅ Filter, measures, rows

StartTime	Type	Duration	User	Database	Query
09:14:20	DAX	16ms	Power BI...	DemoSales	DEFINE VAR __DS0Filte
09:14:07	DAX	1,000ms	Power BI...	DemoSales	DEFINE VAR __DS0Filte
09:14:02	DAX	1,188ms	Power BI...	DemoSales	DEFINE VAR __DS0Filte
09:13:59	DAX	594ms	Power BI...	DemoSales	DEFINE VAR __DS0Filte
09:13:51	DAX	531ms	Power BI...	DemoSales	DEFINE VAR __DS0Filte
09:13:50	DAX	0ms	Power BI...	DemoSales	DEFINE VAR DS0Cor

Refresh (same for both)

StartTime	Type	Duration	User	Database	Query
11:54:49	DAX	1,969ms	Power BI...	DemoSales	DEFINE VAR __DS0FilterTable = 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