Ideas for a NXCALS API based on Pandas-on-Spark

Vito Baggiolini, with work from Michi Hostettler
and input from Guido , Riccardo, Foteini, Alex,
and the NXCALS team

What is this API?

- Purpose:
  - Make it easier for people to use NXCALS
  - Enable the use of Pandas on the NXCALS cluster
  - Achieve better performance thanks to this
- What is Pandas-on-Spark?
  - An implementation of the Pandas API on top of (Py)Spark
  - A tool to process data on the NXCALS cluster using the Pandas API
- And this API?
  - Methods to load different kinds of data into Pandas-on-Spark DataFrames
  - Built on a library that Michi Hostettler has developed, using PySpark
  - Used by ~15 people from OP and ABT in the Beam Performance Tracking (BPT) project
  - Adapted to use Pandas-on-Spark for this Proof-of-concept prototype
  - Extended to also provide “beam_info” to do cross-machine correlation

What is this API not?

- It’s not (yet) a product:
  - The API (style, method names, arguments, etc) has to be reviewed with you as users
  - The implementation has to be reviewed by the team and adapted
  - It is not (yet) optimized for performance

- It should not end up as YANAPI (“yet another NXCALS API”)
  - → Integrate it into PyTimber as a Pandas-on-Spark Flavour?

Typical steps in using the API

- Create a Spark session and use it to create an NxcalsExtractor:

```
spark = _get_spark(app_name="demo-acc-bpt-nx-cli", yarn_flavor=Flavor.YARN_MEDIUM)
nxcals = NxcalsExtractor(spark)
```

- Initialize a query object that defines the time frame

```
query = nxcals.query_for_time("2024-06-01 00:00:00.000", "2024-06-02 00:00:00.000")
```

- Load variables into a Spark-on-Pandas DataFrame

```
psb_ints = query.get_vars(["BR.BCT.INJ.S:INTENSITY", "BR.BCT.S:INTENSITY"], 'CMW')
```

- Use Pandas API to work with the data

```
psb_ints['xmission'] = psb_ints["BR.BCT.S:INTENSITY"] / psb_ints["BR.BCT.INJ.S:INTENSITY"]
```

How does the Data look like?

- All data is contained in Pandas-on-Spark DataFrames
- DataFrame columns represent named values (e.g. Variables)
- Rows represent time (e.g. one row per cycle)
- NXCALS arrays and matrices are converted to arrays in the cells

	timestamp	SPSQC:INJECTION_INTENSITY	SPSQC:EXTRACTED_INTENSITY
datetime			
2024-06-01 00:00:09.735	1717192809735000000	[11040754394531.25, 11042933349609.375]	2.018947e+13
2024-06-01 00:00:24.135	1717192824135000000	[11116307373046.875, 11013232421875.0]	2.055226e+13
2024-06-01 00:00:38.535	1717192838535000000	[11071873779296.875, 11003344726562.5]	2.054909e+13
2024-06-01 00:00:52.935	1717192852935000000	[11005950927734.375, 11011954345703.125]	2.009181e+13
2024-06-01 00:01:07.335	1717192867335000000	[11062303466796.875, 10936619873046.875]	2.007880e+13

Code examples: API + Pandas-on-Spark

1. Overview of the new API calls
2. Get PSB intensities sent to ISO
3. Aggregate PS intensity statistics per USER
4. “Approximative” merging of cyclestamp with acqStamp
5. Cross-accelerator correlation: intensities for MTE Beam instances

Disclaimer: I’m not a Pandas expert at all (but ChatGPT is)

1. Overview of the API calls

- Get values for Variables
- Get values for Device/Properties
- Get Fundamentals
- Get LSA Trims
- Get LHC Fills and Modes

Get values for NXCALS variables in a p-o-s DataFrame

```
var_names = ["SPSQC:DESTINATION",  # string
             "SPSQC:INJECTION_INTENSITY", # array
             "SPSQC:EXTRACTED_INTENSITY"] # float
spsqc = query.get_vars(var_names, 'CMW')
```

timestamp	SPSQC:DESTINATION	SPSQC:INJECTION_INTENSITY	SPSQC:EXTRACTED_INTENSITY
1717192809735000000	FTARGET	[110407543945 31.25, 110429333496 09.375]	2.018947e+13
1717192824135000000	FTARGET	[111163073730 46.875, 110132324218 75.0]	2.055226e+13
1717192838535000000	FTARGET	[110718733792 96.875, 110033447265 62.5]	2.054909e+13
1717192852935000000	FTARGET	[110059509277 34.375, 110119543457 03.125]	2.009181e+13
1717192867335000000	FTARGET	[110623034667 96.875, 109366198730 46.875]	2.007880e+13
1717192881735000000	FTARGET	[110825805664 06.25, 109657531738 28.125]	2.035792e+13
1717192896135000000	FTARGET	[110441333007 81.25, 110328271484 37.5]	2.048957e+13

```
import numpy as np
spsqc['inj_total'] = spsqc['SPSQC:INJECTION_INTENSITY'].apply(np.sum)
spsqc['inj_avg'] = spsqc['SPSQC:INJECTION_INTENSITY'].apply(np.mean)
```

SPSQC:INJECTION_INTENSITY	inj_avg	inj_total
[110407543945 31.25, 110429333496 09.375]	1.104184e+13	2.208369e+13
[111163073730 46.875, 110132324218 75.0]	1.106477e+13	2.212954e+13
[110718737792 96.875, 110033447265 62.5]	1.103761e+13	2.207522e+13
[110059509277 34.375, 110119543457 03.125]	1.100895e+13	2.201791e+13
[110623034667 96.875, 109366198730 46.875]	1.099946e+13	2.199892e+13

Get values for device/properties in a p-o-s DataFrame

```
query.get_dev_props('PR.BCT/HotspotIntensity', system='CMW')
```

acqStamp	cyclestamp	dcAftEje1	dcAftEje2	dcAftInj1	dcAftInj2	dcAftTra
1719280239023827	5	192800700000000	80.170853	78.922707	411.624664	0.0	405.822815
1719280479023						
1719280623323						
1719280743323						
1719280874023						

dcBefEje1	dcBefEje2	dcBefInj1	dcBefInj2	dcBefTra	selector
406.738922	79.166245	-0.001234	0.0	407.960327	CPS.USER.EAST3
383.836945	28.783575	0.011719	0.0	387.501251	CPS.USER.EAST2
844.929871	0.000000	0.029605	0.0	852.869202	CPS.USER.TOF
847.678040	0.000000	-0.001850	0.0	854.090637	CPS.USER.TOF
1093.492432	0.000000	0.012952	0.0	1098.683716	CPS.USER.MD1
1172.885986	0.000000	0.028989	0.0	1175.023438	CPS.USER.SFTPRO1

Get Fundamentals in a p-o-s DataFrame

```
funds_psb_all = query.get_fundamentals('PSB')
```

```
['BASIC_PERIOD_NB', 'BEAM_ID', 'BEAM_LEVEL_NORMAL', 'BEAM_TO_LHC',
 'BP_DURATION_MS', 'CONSEQ_DISO_SEQUENCE', 'CONSEQ_HRS_SEQUENCE',
 'CONSEQ_ISO_SEQUENCE', 'CONSEQ_PS_SEQUENCE', 'CYCLE_DURATION_MS',
 'CYCLE_NB', 'CYCLE_TAG', 'DYN_DEST', 'DYN_LINAC_DEST', 'END_BEAM',
 'LTB_CONTROL', 'PARTICLE', 'PAYLOAD', 'PROG_DEST_GLOB',
 'PROG_DEST_R1', 'PROG_DEST_R2', 'PROG_DEST_R3', 'PROG_DEST_R4',
 'PROG_LINAC_DEST', 'PROG_PSB_DEST', 'PSB_NO_RING_MASK',
```

CYCLE_NB	CYCLE_TAG	DYN_DEST	DYN_LINAC_DEST	END_BEAM	LTB_CONTROL	PARTICLE	PAYLOAD	PROG_DEST_GLOB
21	4629	ISOGPS	PSB	True	True	PROTON	NaN	ISO_GPS
34	6178	BDUMP	PSB	True	True	PROTON	NaN	BT_DUMP
3	515	PS	PSB	False	True	PROTON	NaN	BYRINGS
24	4632	ISOGPS	PSB	True	True	PROTON	NaN	ISO_GPS
23	535	PS	PSB	False	True	PROTON	NaN	BYRINGS
4	4612	ISOGPS	PSB	True	True	PROTON	NaN	ISO_GPS
34	6178	BDUMP	PSB	True	True	PROTON	NaN	BT_DUMP

Get LSA Trims in a p-o-s DataFrame

```
query.get_lsa_trims('PHYSICS-6.8TeV-1.2m-2024_V1@135%',
                    'LHCBEAM1/IP1_SEPSCAN_X_MM', part='correction')
```

timestamp	LHCBEAM1/IP1_SEPSCAN_X_MM
1717192822207000000	0.012843
1717192864255000000	0.012581
1717192952132000000	0.012319
1717192994206000000	0.012057
1717193036313000000	0.012319

Get Fills in a p-o-s DataFrame

```
nxcals.get_lhc_fills([10000,10001], timestamp=True, datetime=True)
```

fill_number	start_timestamp	end_timestamp	start_time	end_time
10000	1723551725608363525	1723573109511988525	2024-08-13 14:22:05.608363525+02:00	2024-08-13 20:18:29.511988525+02:00
10001	1723573109511988525	1723626820035988525	2024-08-13 20:18:29.511988525+02:00	2024-08-14 11:13:40.035988525+02:00

Get Modes in a p-o-s DataFrame

```
modes = nxcals.get_lhc_modes(10000)
```

beam_mode	start_time	end_time
INJPROB	2024-08-13 15:00:25.010863525+02:00	2024-08-13 15:08:09.680488525+02:00
INJPHYS	2024-08-13 15:08:09.680488525+02:00	2024-08-13 15:41:00.892863525+02:00
PRERAMP	2024-08-13 15:41:00.892863525+02:00	2024-08-13 15:43:13.859238525+02:00
RAMP	2024-08-13 15:43:13.859238525+02:00	2024-08-13 16:04:33.058988525+02:00
FLATTOP	2024-08-13 16:04:33.058988525+02:00	2024-08-13 16:04:46.141738525+02:00
SQUEEZE	2024-08-13 16:04:46.141738525+02:00	2024-08-13 16:13:54.304238525+02:00
ADJUST	2024-08-13 16:13:54.304238525+02:00	2024-08-13 16:19:37.988488525+02:00
STABLE	2024-08-13 16:19:37.988488525+02:00	2024-08-13 20:17:32.570363525+02:00
BEAMDUMP	2024-08-13 20:17:32.570363525+02:00	2024-08-13 20:17:34.944738525+02:00
RAMPDOWN	2024-08-13 20:17:34.944738525+02:00	2024-08-13 20:18:29.511988525+02:00

2. Get PSB intensities sent to ISO

- Get PSB intensities
- Get Fundamentals and filter for ISO
- Merge fundamentals with PSB intensities to keep only ISO values

```
var_names = ["BR.BCT.INJ.S:INTENSITY", "BR.BCT.S:INTENSITY"]
psb_ints = query.get_vars(var_names, 'CMW')
```

timestamp	BR.BCT.INJ.S:INTENSITY	BR.BCT.S:INTENSITY
1717192800065000000	415.859589	415.302032
1717192801265000000	3158.793945	3154.184082
1717192802465000000	390.783051	390.817047
1717192803665000000	3159.321045	3154.684082
1717192804865000000	874.842590	869.105835
1717192806065000000	873.934875	868.538574
1717192807265000000	1186.856812	1178.742065

```
funds_psb_all = query.get_fundamentals('PSB')
```

Already seen

```
['BASIC_PERIOD_NB', 'BEAM_ID', 'BEAM_LEVEL_NORMAL', 'BEAM_TO_LHC',
 'BP_DURATION_MS', 'CONSEQ_DISO_SEQUENCE', 'CONSEQ_HRS_SEQUENCE',
 'CONSEQ_ISO_SEQUENCE', 'CONSEQ_PS_SEQUENCE', 'CYCLE_DURATION_MS',
 'CYCLE_NB', 'CYCLE_TAG', 'DYN_DEST', 'DYN_LINAC_DEST', 'END_BEAM',
 'LTB_CONTROL', 'PARTICLE', 'PAYLOAD', 'PROG_DEST_GLOB',
 'PROG_DEST_R1', 'PROG_DEST_R2', 'PROG_DEST_R3', 'PROG_DEST_R4',
```

CYCLE_NB	CYCLE_TAG	DYN_DEST	DYN_LINAC_DEST	END_BEAM	LTB_CONTROL	PARTICLE	PAYLOAD	PROG_DEST_GLOB
21	4629	ISOGPS	PSB	True	True	PROTON	NaN	ISO_GPS
34	6178	BDUMP	PSB	True	True	PROTON	NaN	BT_DUMP
3	515	PS	PSB	False	True	PROTON	NaN	BYRINGS
24	4632	ISOGPS	PSB	True	True	PROTON	NaN	ISO_GPS
23	535	PS	PSB	False	True	PROTON	NaN	BYRINGS
4	4612	ISOGPS	PSB	True	True	PROTON	NaN	ISO_GPS
34	6178	BDUMP	PSB	True	True	PROTON	NaN	BT_DUMP

```
funds_psb = funds_psb_all[['timestamp', 'USER', 'DYN_DEST']]
```

```
fund_filter = '(DYN_DEST == "ISOGPS") and (USER == "NORMGPS")'
funds_iso = funds_psb.query(fund_filter)
```

	timestamp	USER	DYN_DEST
1717192800065000000		EAST3	PS
1717192801265000000		NORMGPS	ISOGPS
1717192802465000000		EAST2	PS
1717192803665000000		NORMGPS	ISOGPS
1717192804865000000		TOF	PS

	timestamp	USER	DYN_DEST
1717192801265000000		NORMGPS	ISOGPS
1717192803665000000		NORMGPS	ISOGPS
1717192812065000000		NORMGPS	ISOGPS
1717192814465000000		NORMGPS	ISOGPS
1717192818065000000		NORMGPS	ISOGPS

```
fund_queried = query.get_fundamentals("PSB", filter = fund_filter)
```

```
funds_iso.merge(psb_ints, on='timestamp')
```

funds_iso

timestamp	USER	DYN_DEST
1717192801265000000	NORMGPS	ISOGPS
1717192803665000000	NORMGPS	ISOGPS
1717192812065000000	NORMGPS	ISOGPS
1717192814465000000	NORMGPS	ISOGPS
1717192818065000000	NORMGPS	ISOGPS

psb_ints

timestamp	BR.BCT.INJ.S:INTENSITY
1717192800065000000	415.859589
1717192801265000000	3158.793945
1717192802465000000	390.783051
1717192803665000000	3159.321045
1717192804865000000	874.842590
1717192806065000000	873.934875
1717192807265000000	1186.856812

```
funds_iso.merge(psb_ints, on='timestamp')
```

timestamp	USER	DYN_DEST	BR.BCT.INJ.S:INTENSITY	BR.BCT.S:INTENSITY
1717192801265000000	NORMGPS	ISOGPS	3158.793945	3154.184082
1717192803665000000	NORMGPS	ISOGPS	3159.321045	3154.684082
1717192812065000000	NORMGPS	ISOGPS	3165.836670	3161.947754
1717192814465000000	NORMGPS	ISOGPS	3166.471680	3160.242920
1717192818065000000	NORMGPS	ISOGPS	3163.974609	3157.448730
1717192819265000000	NORMGPS	ISOGPS	3169.454346	3166.790039
1717192821665000000	NORMGPS	ISOGPS	3179.278320	3172.782959
1717192825265000000	NORMGPS	ISOGPS	3182.861816	3178.884277

3. PS intensity Statistics per USER

- Get Intensities values and Selectors from PR.BCT/HotspotIntensity
- Filter out CPS.USER.ZERO
- Calculate transmission
- Group by selectors and aggregate

Already seen

```
query.get_dev_props('PR.BCT/HotspotIntensity', system='CMW')
```

acqStamp	cyclestamp	dcAftEje1	dcAftEje2	dcAftInj1	dcAftInj2	dcAftTra
7171928023902382 75	192800700000000	80.170853	78.922707	411.624664	0.0	405.822815
7171928047902382 75	192803100000000	29.011892	-21.000000	387.501251	0.0	385.669098
7171928062332382 75	192805500000000	-0.026521	0.000000	860.197815	0.0	847.067383
7171928074332382 75	192806700000000	-0.130140	0.000000	861.113892	0.0	847.372742
7171928087402382 75	192807900000000	-0.061678	0.000000	1102.347900	0.0	1098.378174

```
# select only a few interesting columns:
pr_bct = pr_bct_all[['cyclestamp', 'dcAftInj1', 'dcBefEje1','selector']]
# exclude ZERO users:
pr_bct = pr_bct[pr_bct['selector'] != 'CPS.USER.ZERO']
```

```
pr_bct['xmission'] = pr_bct['dcBefEje1'] / pr_bct['dcAftInj1']
```

cyclestamp	dcAftInj1	dcBefEje1	selector	xmission
1717192800700000000	411.624664	406.738922	CPS.USER.EAST3	0.988131
1717192803100000000	387.501251	383.836945	CPS.USER.EAST2	0.990544
1717192805500000000	860.197815	844.929871	CPS.USER.TOF	0.982251
1717192806700000000	861.113892	847.678040	CPS.USER.TOF	0.984397
1717192807900000000	1102.347900	1093.492432	CPS.USER.MD1	0.991967
1717192809100000000	1179.603882	1172.885986	CPS.USER.SFTPRO1	0.994305

```
# calculate the average value for all columns:
pr_bct.groupby('selector').mean()
```

selector	cyclestamp	dcAftInj1	dcBefEje1	xmission
CPS.USER.SFTPRO1	1.717193e+18	1178.916855	1174.527267	0.996276
CPS.USER.MD1	1.717193e+18	1101.126465	1093.492432	0.993068
CPS.USER.TOF	1.717193e+18	861.152069	847.372711	0.984000
CPS.USER.EAST3	1.717193e+18	414.189679	409.303925	0.988205
CPS.USER.EAST1	1.717193e+18	390.554855	387.653931	0.992569
CPS.USER.EAST2	1.717193e+18	389.944122	386.483398	0.991120
CPS.USER.MD6	1.717193e+18	322.765045	646.446167	2.002838

```
# make different aggregations
agg = { 'cyclestamp': 'count',
        'dcAftIr
        'dcBefEj
        'xmissic
      }
pr_bct.groupby(
```

selector	play_count	total_inj1	total_eje1	avg_xmission
CPS.USER.TOF	2509546	2.172965e+09	2.133058e+09	0.981699
CPS.USER.SFTPRO1	1304606	1.107485e+09	1.101986e+09	0.995021
CPS.USER.EAST3	1276195	4.836009e+08	4.789938e+08	0.990565
CPS.USER.EAST1	898898	4.354437e+08	4.286183e+08	0.980136
CPS.USER.EAST2	857201	4.030046e+08	3.974711e+08	0.982881
CPS.USER.SFTPRO3	300075	4.070527e+08	4.049302e+08	0.994779
CPS.USER.AD	189675	3.520536e+08	3.496040e+08	0.992287
CPS.USER.ION1	95151	8.629164e+05	7.807343e+05	0.903679
CPS.USER.EAST4	93330	1.604503e+06	1.521263e+06	0.935230

Aggregated over one year

4. Approximative merging of cyclestamp with acqStamp

- Artificially create two DataFrames (one with cyclesamp, one with acqStamp) by selecting column from PR.BCT/HotspotIntensity
- Use merge_asof() to correlate the two DataFrames by cyclestamp with acqStamp

Already seen

```
query.get_dev_props('PR.BCT/HotspotIntensity', system='CMW')
```

acqStamp	cyclestamp	dcAftEje1	dcAftEje2	dcAftInj1	dcAftInj2	dcAftTra
7171928023902382 75	192800700000000	80.170853	78.922707	411.624664	0.0	405.822815
7171928047902382 75	192803100000000	29.011892	-21.000000	387.501251	0.0	385.669098
7171928062332382 75	192805500000000	-0.026521	0.000000	860.197815	0.0	847.067383
7171928074332382 75	192806700000000	-0.130140	0.000000	861.113892	0.0	847.372742
7171928087402382 75	192807900000000	-0.061678	0.000000	1102.347900	0.0	1098.378174

```python
import pyspark.pandas as ps
```

```python
ps.merge_asof(pr_bct_cyc, pr_bct_acq,
              left_on='cyclestamp', right_on='acqStamp',
              direction="forward")
```

pr_bct_cyc

cyclestamp	dcAftEje1	dt_cyclestamp
1717192800700000000	80.170853	2024-05-31 22:00:00.700
1717192803100000000	29.011892	2024-05-31 22:00:03.100
1717192805500000000	-0.026521	2024-05-31 22:00:05.500
1717192806700000000	-0.130140	2024-05-31 22:00:06.700
1717192807900000000	-0.061678	2024-05-31 22:00:07.900

pr_bct_acq

acqStamp	dcAftEje1	dt_acqStamp
1717192802390238275	80.170853	2024-05-31 22:00:02.390238
1717192804790238275	29.011892	2024-05-31 22:00:04.790238
1717192806233238275	-0.026521	2024-05-31 22:00:06.233238
1717192807433238275	-0.130140	2024-05-31 22:00:07.433238
1717192808740238275	-0.061678	2024-05-31 22:00:08.740238

```python
import pyspark.pandas as ps
```

```python
ps.merge_asof(pr_bct_cyc, pr_bct_acq,
              left_on='cyclestamp', right_on='acqStamp',
              direction="forward")
```

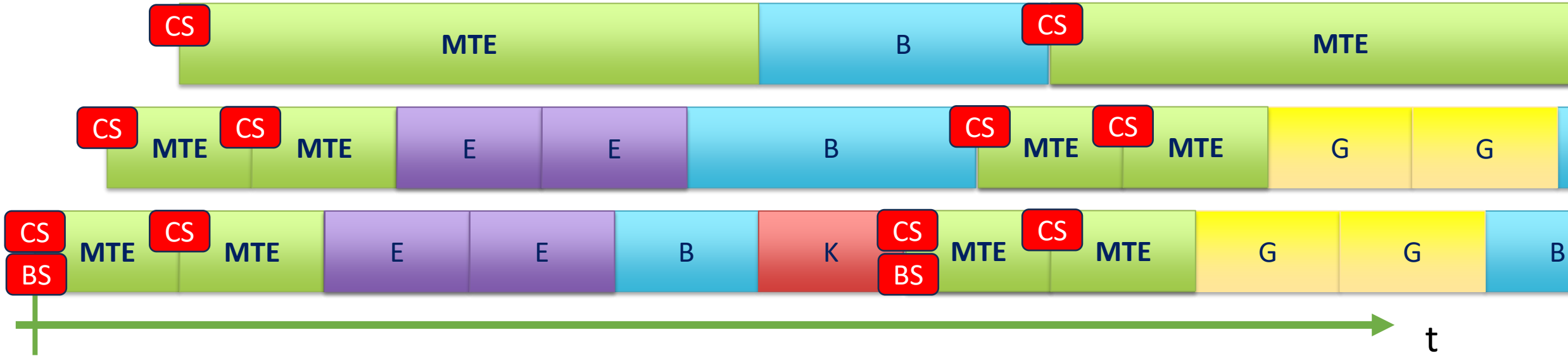
dt_cyclestamp	dt_acqStamp	dcAftEje1_x	dcAftEje1_y
2024-05-31 22:00:00.700	2024-05-31 22:00:02.390238	80.170853	80.170853
2024-05-31 22:00:03.100	2024-05-31 22:00:04.790238	29.011892	29.011892
2024-05-31 22:00:05.500	2024-05-31 22:00:06.233238	-0.026521	-0.026521
2024-05-31 22:00:06.700	2024-05-31 22:00:07.433238	-0.130140	-0.130140
2024-05-31 22:00:07.900	2024-05-31 22:00:08.740238	-0.061678	-0.061678
2024-05-31 22:00:09.100	2024-05-31 22:00:09.940238	-0.101768	-0.101768

5. Cross-accelerator correlation: intensities for MTE Beam instances

- Get “beam_info” data describing the BCD and filter out MTE beams
- Get Intensities for PSB, PS and SPS
- Use “beam_info” data to filter out intensities for MTE beam instances

Cross-accelerator analysis

- Example: BCD with two MTE beam instances
- Need to extract data belonging to each beam instance in all machines
- Obtain cyclestamps CS belonging to the MTE beam in each machine
- Each beam instance has a unique BeamStamp BS



```
beam_instances = query.get_beam_instances(machines=["PSB", "CPS", "SPS"], sort=True)
```

BS	beamStamp	bcdBeamOffsetBp	beamId	CS	cyclestamp	beamName
	1717192837265000000	0	6085		1717192837265000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
	1717192837265000000	0	6085		1717192838465000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
	1717192837265000000	0	6085		1717192837900000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
	1717192837265000000	0	6085		1717192838201000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
	1717192839665000000	2	58522		1717192839665000000	[PSB]:EAST_N_2024+[CPS]:EAST_N_24
	1717192839665000000	2	58522		1717192840300000000	[PSB]:EAST_N_2024+[CPS]:EAST_N_24
	1717192840865000000	3	47298		1717192840865000000	[PSB]:ISOGPS_2024
	1717192842065000000	4	47298		1717192842065000000	[PSB]:ISOGPS_2024
	1717192842700000000	4	57090		1717192842700000000	[CPS]:~~zero~~

```
mte_beams = beam_instances.query('beamId == 6085')
```

```
mte_beams = beam_instances.query('beamId == 6085')
```

BS beamStamp	bcdBeamOffsetBp	beamId	bcdBeamNormal	CS cyclestamp	beamName
1717192837265000000	0	6085	True	1717192837265000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192837265000000	0	6085	True	1717192838465000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192837265000000	0	6085	True	1717192837900000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192837265000000	0	6085	True	1717192839100000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192837265000000	0	6085	True	1717192838535000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192851665000000	12	6085	True	1717192851665000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192851665000000	12	6085	True	1717192852865000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192851665000000	12	6085	True	1717192852300000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192851665000000	12	6085	True	1717192853500000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192851665000000	12	6085	True	1717192852935000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192866065000000	24	6085	True	1717192866065000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192866065000000	24	6085	True	1717192867265000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192866065000000	24	6085	True	1717192866700000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192866065000000	24	6085	True	1717192867900000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...
1717192866065000000	24	6085	True	1717192867335000000	[PSB]:MTE_2024{2}+[CPS]:MTE_BB_24{2}+[SPS]:SFT...

```
mte_beams.merge(intens_values, left_on='cyclestamp', right_on='timestamp')
```

BS beamStamp	CS cyclestamp	BR.BCT.S:INTENSITY	PR.DCBEFEJE_1:INTENSITY	SPSQC:EXTRACTED_INTENSITY
1717192837265000000	1717192837265000000	1179.630615	NaN	NaN
1717192837265000000	1717192837900000000	NaN	1174.107422	NaN
1717192837265000000	1717192838465000000	1182.064453	NaN	NaN
1717192837265000000	1717192838535000000	NaN	NaN	2.054909e+13
1717192837265000000	1717192839100000000	NaN	1175.939575	NaN
1717192851665000000	1717192851665000000	1176.130005	NaN	NaN
1717192851665000000	1717192852865000000	1177.181763	NaN	NaN
1717192851665000000	1717192852935000000	NaN	NaN	2.009181e+13
1717192851665000000	1717192853500000000	NaN	1170.748413	NaN
1717192866065000000	1717192866065000000	1178.257202	NaN	NaN
1717192866065000000	1717192866700000000	NaN	1169.527100	NaN
1717192866065000000	1717192867265000000	1176.900879	NaN	NaN
1717192866065000000	1717192867335000000	NaN	NaN	2.007880e+13
1717192866065000000	1717192867900000000	NaN	1170.137695	NaN

```
mte_intensities = mte_beam_intensities.groupby('beamStamp').sum()
```

```
mte_intensities = mte_beam_intensities.groupby('beamStamp').sum()
```

BS beamStamp	BR.BCT.S:INTENSITY	PR.DCBEFEJE_1:INTENSITY	SPSQC:EXTRACTED_INTENSITY
1717192851665000000	2353.311768	2340.886108	2.009181e+13
1717192837265000000	2361.695068	2350.046997	2.054909e+13
1717192866065000000	2355.158081	2339.664795	2.007880e+13

[End of code examples]

Timestamps

- Problem: NXCALS works with UTC, while users think in local time
  - Both do so for a good reason
- API accepts different formats for timestamps:
  - NXCALS String “2024-06-01 10:53:16.300”
  - ISO 8601 String “2024-06-01T10:53:16.300“
  - ISO 8601 with UTC TimeZone “2024-06-01T08:53:16.300+00”
  - NXCALS nanoseconds 1717231996300000
  - Python milliseconds 1717231996.3
  - Python datetime object
- API accepts timestamps without a TimeZone as local time
- API internally uses timestamps with TimeZone information
  - API immediately adds TimeZone information
  - Results have TimeZone information

Caveats, open questions and next steps

- Caveats
  - Pandas API implementation on PySpark is not yet complete but growing (it is always possible to use PySpark for unsupported methods)
  - Performance benefit from keeping data on the cluster, but can't do wonders
- Open questions:
  - Review of API methods and implementation
  - How to avoid that this API becomes yet another NXCALS API?
  - → Somehow integrate into PyTimber?
- Next steps
  - We need to discuss the API and the open points with you
  - Eat our own dogfood by using this API in the BPT project

Other requirements we are aware of

- Additional data partitionings to increase loading speed
  - Additional partitioning by individual VARIABLES (class/property/field)?
  - LHC LuminosityFollowUp with data partitioned by Fills&Modes
- Additional functionality
  - Store data in NXCALS cluster (PyTimber page-store)
  - Ingest data
  - Handle versioned data more easily
  - On-demand logging
  - Loading data from local storage into a Spark session
- All this has to be seen in a broader context than just NXCALS
- It is being followed-up in the Data Processing Platform Project c.f. Marcin's JAP24 Presentation

Summary

- API makes data analysis on the NXCALS cluster easier and faster
  - All data is presented as Pandas-on-Spark DataFrames
  - VECTOR data is already converted to arrays in the cells
  - Data analysis is done using Pandas API
  - Data is processed on the NXCALS cluster, only result is extracted
- Have a look yourself: acc-bpt-nx-api · GitLab

- There are still several open points we need to discuss
- Other requirements are followed up in the DPP project