The HSF Conditions Database: Intelligent Caching

Candidate: Ernest Sorokun

Mentor: Lino Gerlach

HSF Conditions DB – Overview

Conditions Data handing is reoccurring problem w/ unique challenges

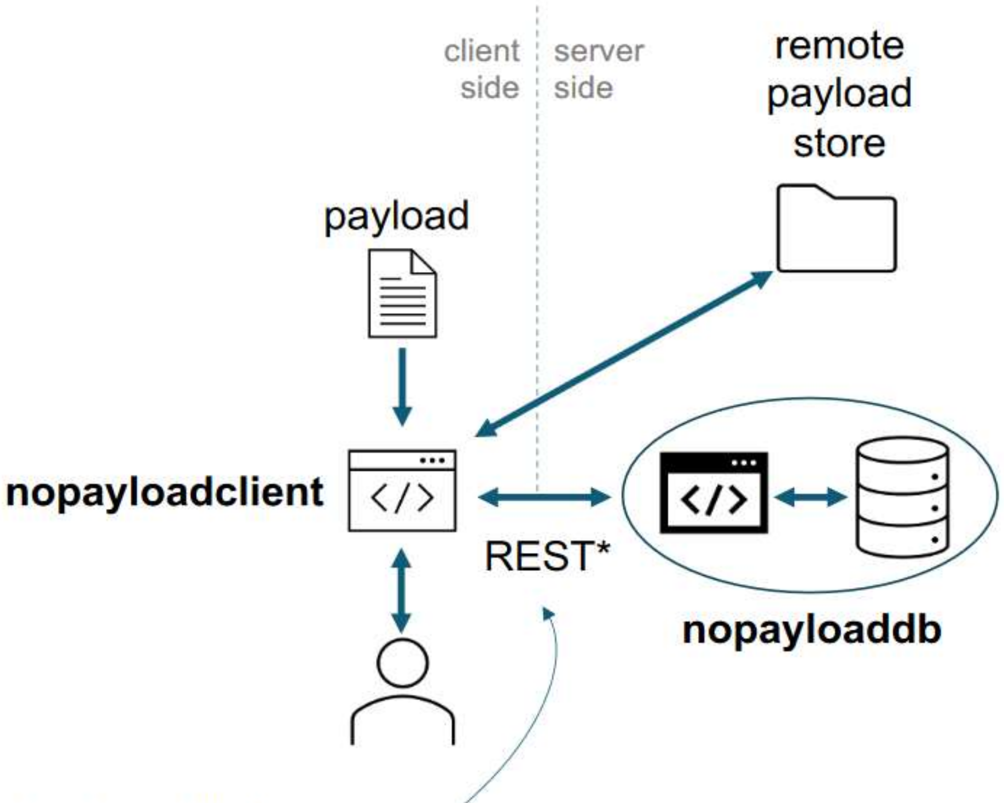
- Heterogenous data structure (a priori unknown)
- High access rates (from distributed computing)

HSF gathered experience from various experiments

- Published set of recommendations *

A reference implementation was developed **

- Separate meta-data & payloads (file catalogue)
- Already in use: sPHENIX @ BNL (~25k jobs)
- No server-side caching yet



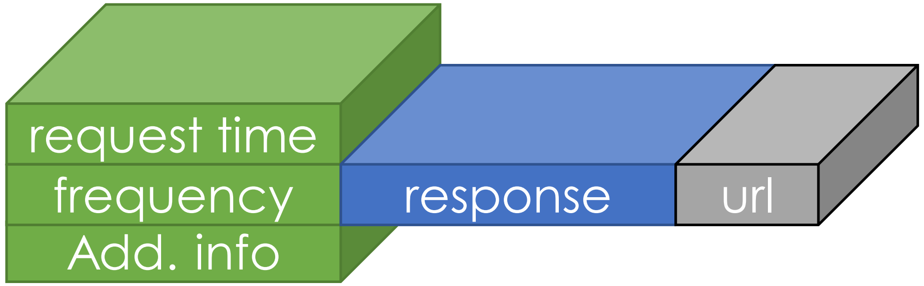
*Example query (simplified)

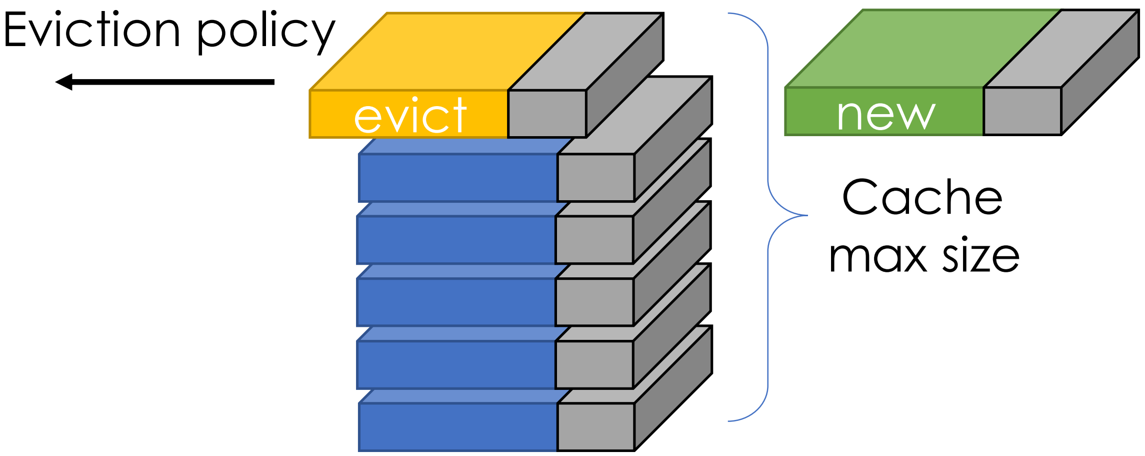
```
curl http://<host>/api/payloadiovs/?gtName=test_gt&iovNum=42
-> {type_1: url_1, type_2: url_2, …}
```

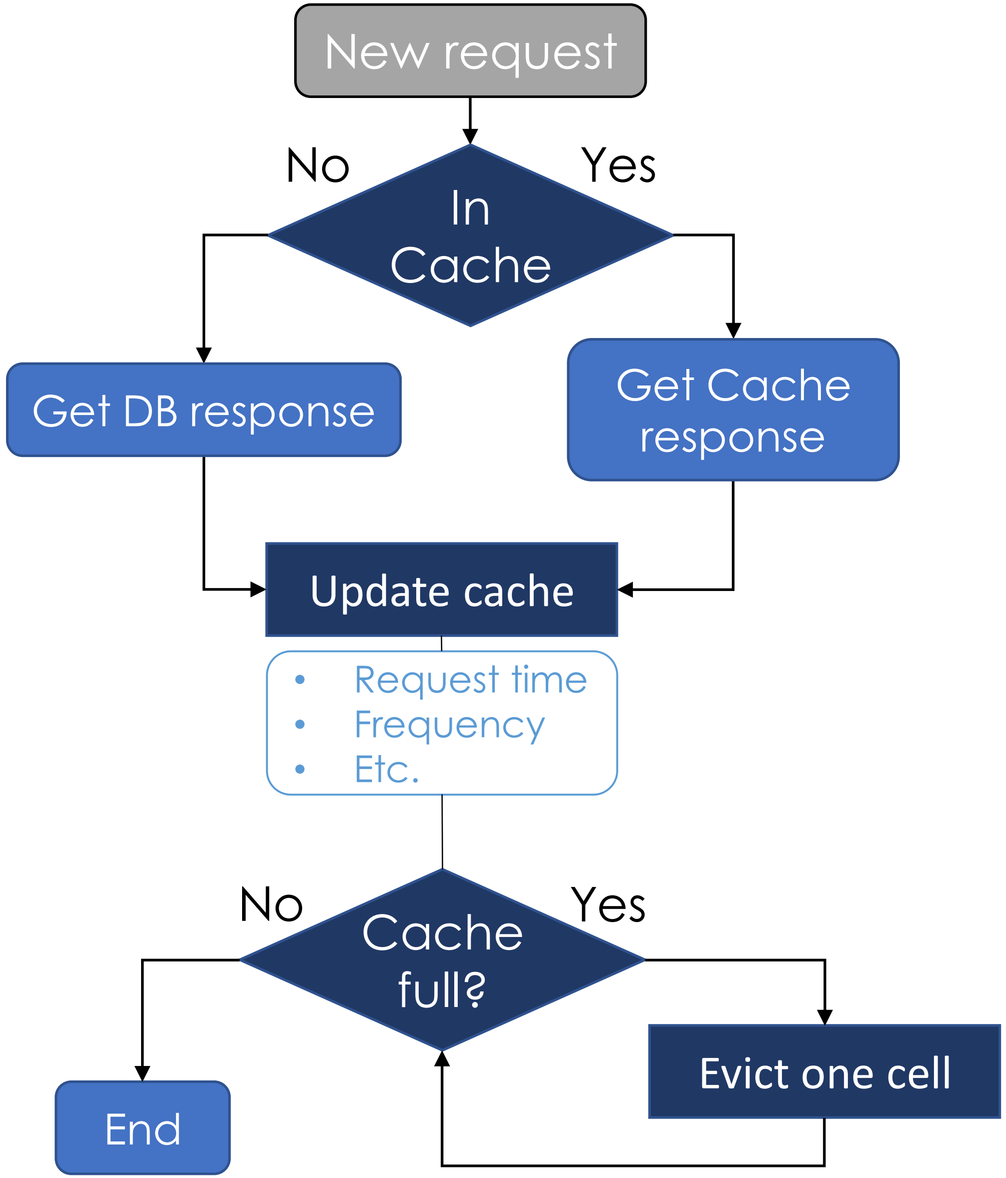
* HSF Conditions Databases activity: https://hepsoftwarefoundation.org/activities/conditionsdb.html
** HSF Conditions Database Reference Implementation: link

Cache workflow

Cache cell



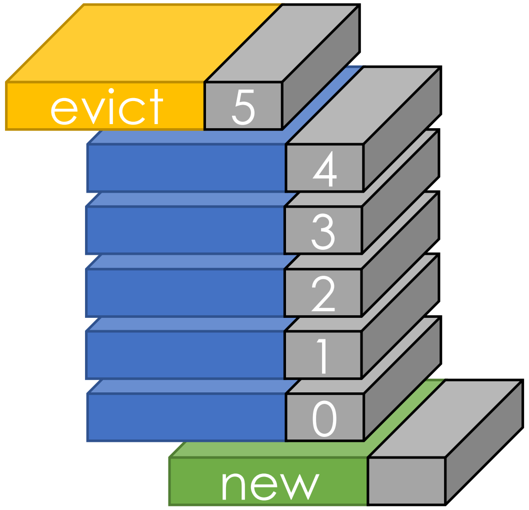




Classical cache strategies

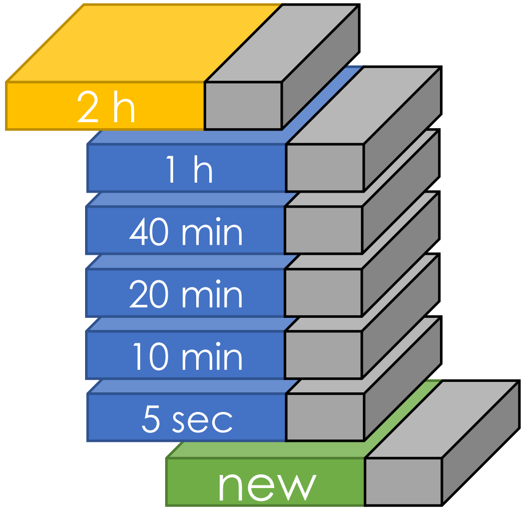
FIFO: First In First Out

- Order for eviction is the same as an entry order
- Insertion ordered dictionary



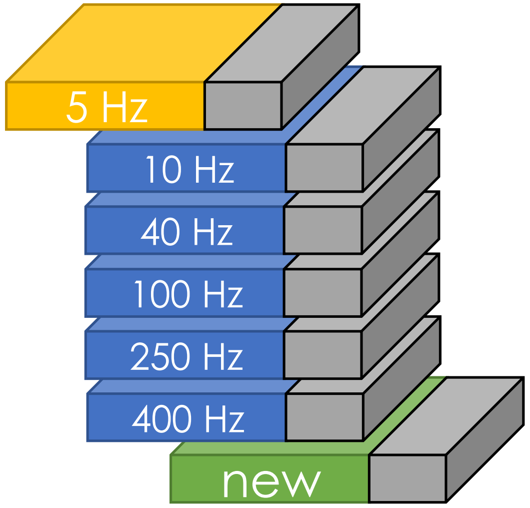
LRU: Least Recently Used

- Evict a cell with the oldest request timestamp
- Request ordered dictionary with timestamps



LFU: Least Frequently Used

- Evict a cell with the lowest request frequency
- Dictionary with request counters



Data example

- Web server access logs from sPHENIX
  - First iteration of calibrations
  - Time independent
  - No need of Intelligent caching
  - Classic caching strategies gives 99.6% serves from cache

- Cryogenic failure prevented next data taking period

- Retrieve log files from other experiments
  - ALICE
  - CMS
  - ATLAS

- We investigate them for our research but none of them are 1:1 comparible to HSF Conditions DB

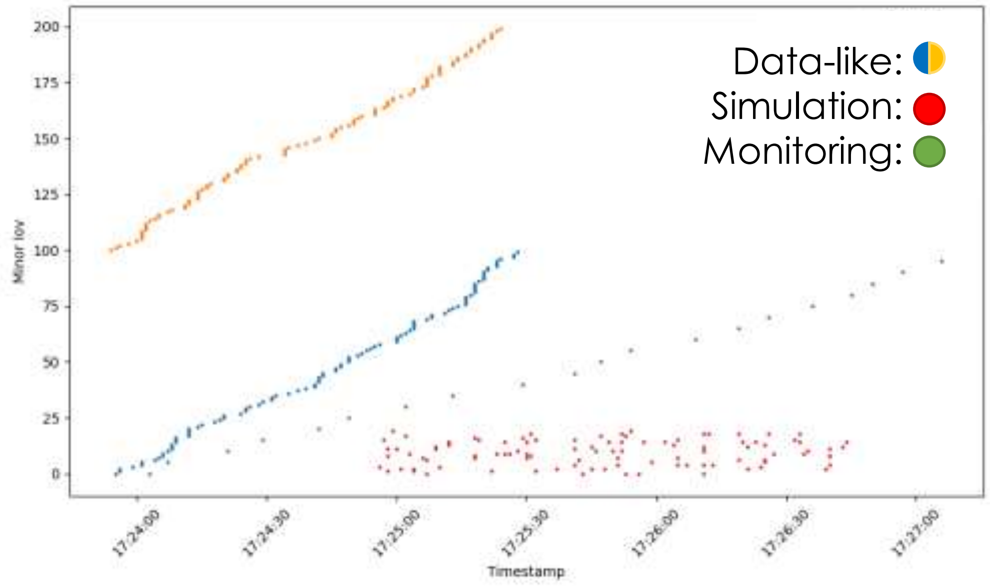
URL's

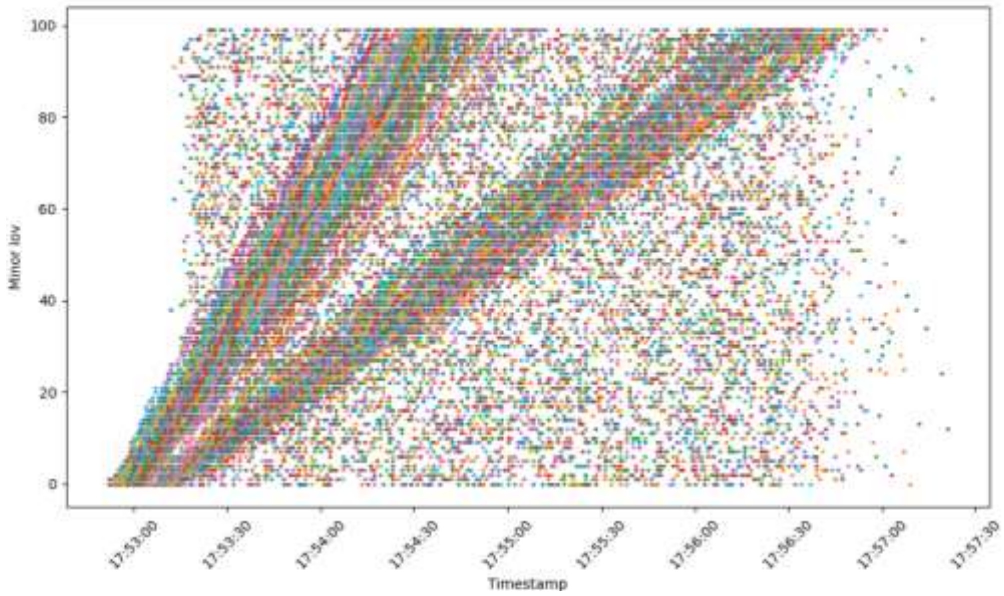
sPHENIX

Special thanks to:
Costin Grigoras (ALICE)
Andrea Formica (ATLAS)
Dave Dykstra (CMS)
Chris Pinkenburg (sPHENIX)

Simulated Data

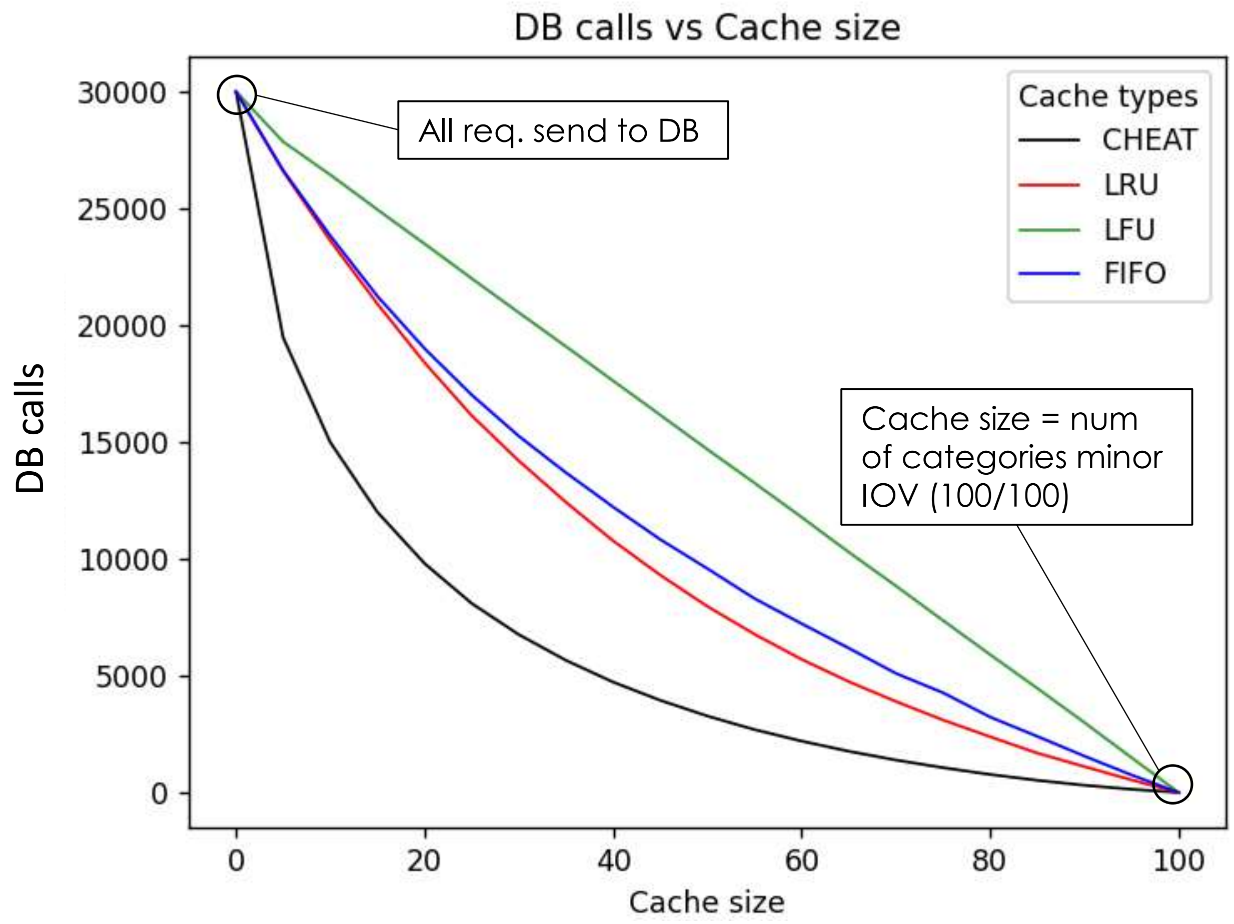
- **We study only the change in the minor IOV sub-parameter**
- **Individual 'requesters' run in parallel** (at least have overlap)
  - A requester never repeats a request (local caching)
  - A requester takes a fixed period (plus random fluct.) before moving to next IOV
- **Three 'campaigns' running in parallel**:
  - Data-like, 100 requesters, 1s and 2s period
  - Simulation-like, 100 requesters, 2s period
- **Each requester makes 100 calls**
  - The requesters start with random delay
    - Simulate non-instant batch submission





Use of classical strategies

- Cheat cache - optimal cache strategy (theory)
    - Knows all requests (even in future)
- LRU – best strategy, but still room for improvement
- Goal: get closer to Cheat Cache's performance using ML



Deep Learn approach

Deep Learn model

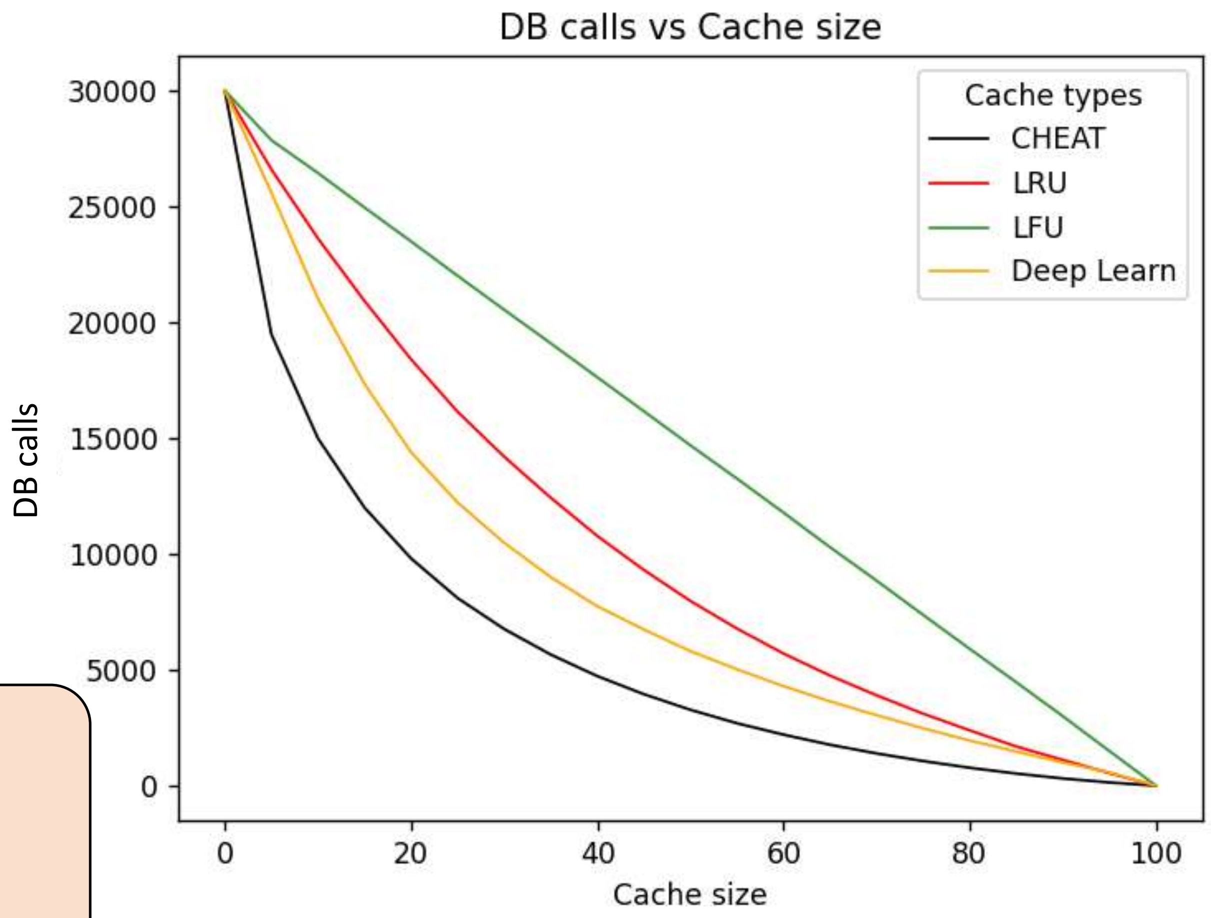
- Input: last 100 requested minor IOV
- Hidden layers:
  - Dense 16
  - Dense 16
  - Dense 16
  - LSTM 16
- Output: probability for each of 100 minor IOVs to be in the next request
- Loss_func – categorical crossentropy

Evict policy

- Removes cell in the cache with the lowest probability to be requested on next step

Problems

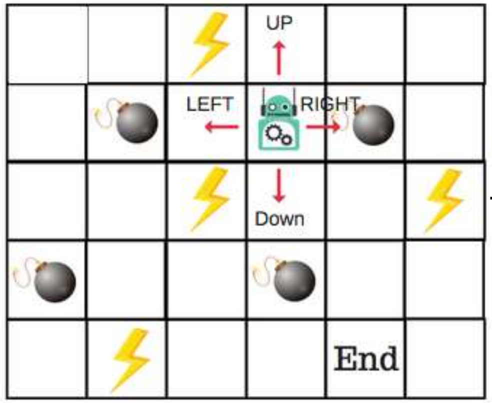
- Processing time - slow
- Needs additional memory for processing



Reinforcement Learning | Q-Learning

Concepts

- **State Space**
- **Action Space**
- **Reward**
- **Q-Table**



Q-table initialised at zero

	UP	DOWN	LEFT	RIGHT
0	0	0	0	0
1	0	0	0	0
2	0	0	0	0
3	0	0	0	0
4	0	0	0	0
5	0	0	0	0
6	0	0	0	0
7	0	0	0	0
8	0	0	0	0

After few episodes

	UP	DOWN	LEFT	RIGHT
0	0	0	0	0
1	0	0	0	0
2	0	2.25	2.25	0
3	0	0	5	0
4	0	0	0	0
5	0	0	0	0
6	0	5	0	0
7	0	0	2.25	0
8	0	0	0	0

Eventually

	UP	DOWN	LEFT	RIGHT
0	0	0	0.45	0
1	0	1.01	0	0
2	0	2.25	2.25	0
3	0	0	5	0
4	0	0	0	0
5	0	0	0	0
6	0	5	0	0
7	0	0	2.25	0
8	0	0	0	0

$$New\ Q(s,a) = Q(s,a) + \alpha[R(s,a) + \gamma\ maxQ'(s',a') - Q(s,a)]$$

- New Q value
- Current Q value
- Learning rate
- Reward
- Discount rate
- Max expected future reward
- Current Q value

Q-Learning for caching

Time Ordered Dict Cache

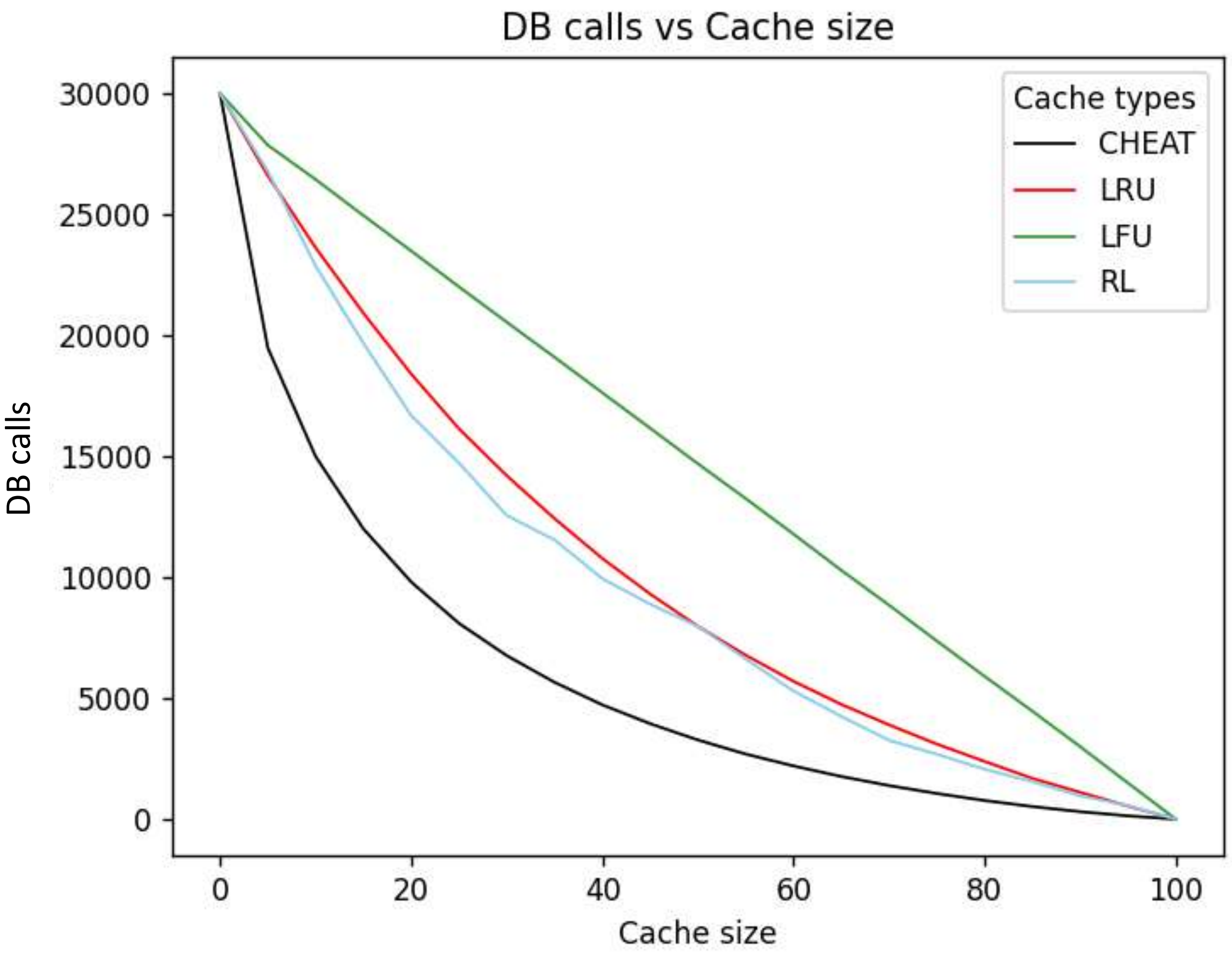
$$[cell_0, cell_1, cell_2, \ldots, cell_{n-2}, cell_{n-1}, cell_n]$$

Oldest request ← **Newest request**

Actions:
Eviction one of the first and last two values of the time–ordered cache

States:
Parametrized states by the combination of features
- IOV, counter, timestamp –> Boolean / discrete

- Difference of timestamps and counters shows actuality of the value
- Difference of IOV shows backward values



Conclusion & Outlook

- Investigated request logs of condition DB of sPHENIX
  - Cryogenic failure before time-dependent calibrations
  - Classic caching strategies would reduce DB-calls by 99.6%
- Resorted to more complex simulated access logs
- Developed two intelligent caching methods:
  - Supervised Deep Learning
  - Reinforcement Learning
- Both result in fewer db-calls than classic strategies
  - still have to optimize run-time and robustness

Thank you for attention

RL Features

Feature 1

$$IOV_0 - IOV_1 < IOV_{n-1} - IOV_n$$

True: 1
False: -1
Else: 0

Feature 2

$$IOV_0 - IOV_{n-1} < IOV_1 - IOV_n$$

True: 1
False: -1
Else: 0

Feature 3

$$tm_{LRU} - tm_{avg} \leq tm_{MRU} - tm_{avg}$$

True: 1
False: 0

Feature 4

$$use_{LRU} - use_{avg} \leq use_{MRU} - use_{avg}$$

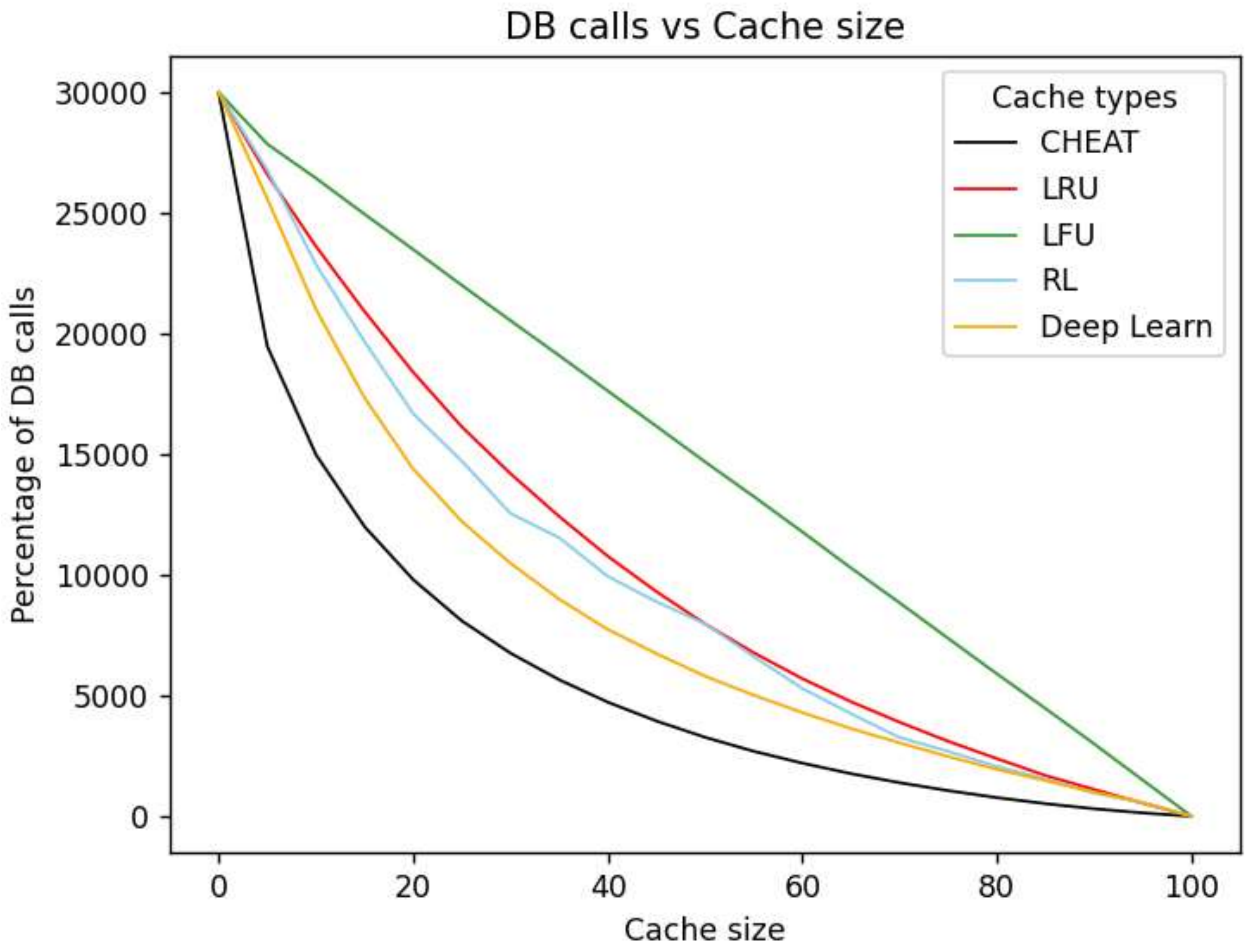
True: 1
False: 0

Feature 5

$$IOV_{LRU} - IOV_{avg} \leq IOV_{MRU} - IOV_{avg}$$

True: 1
False: 0

All strategies



DB calls vs requests