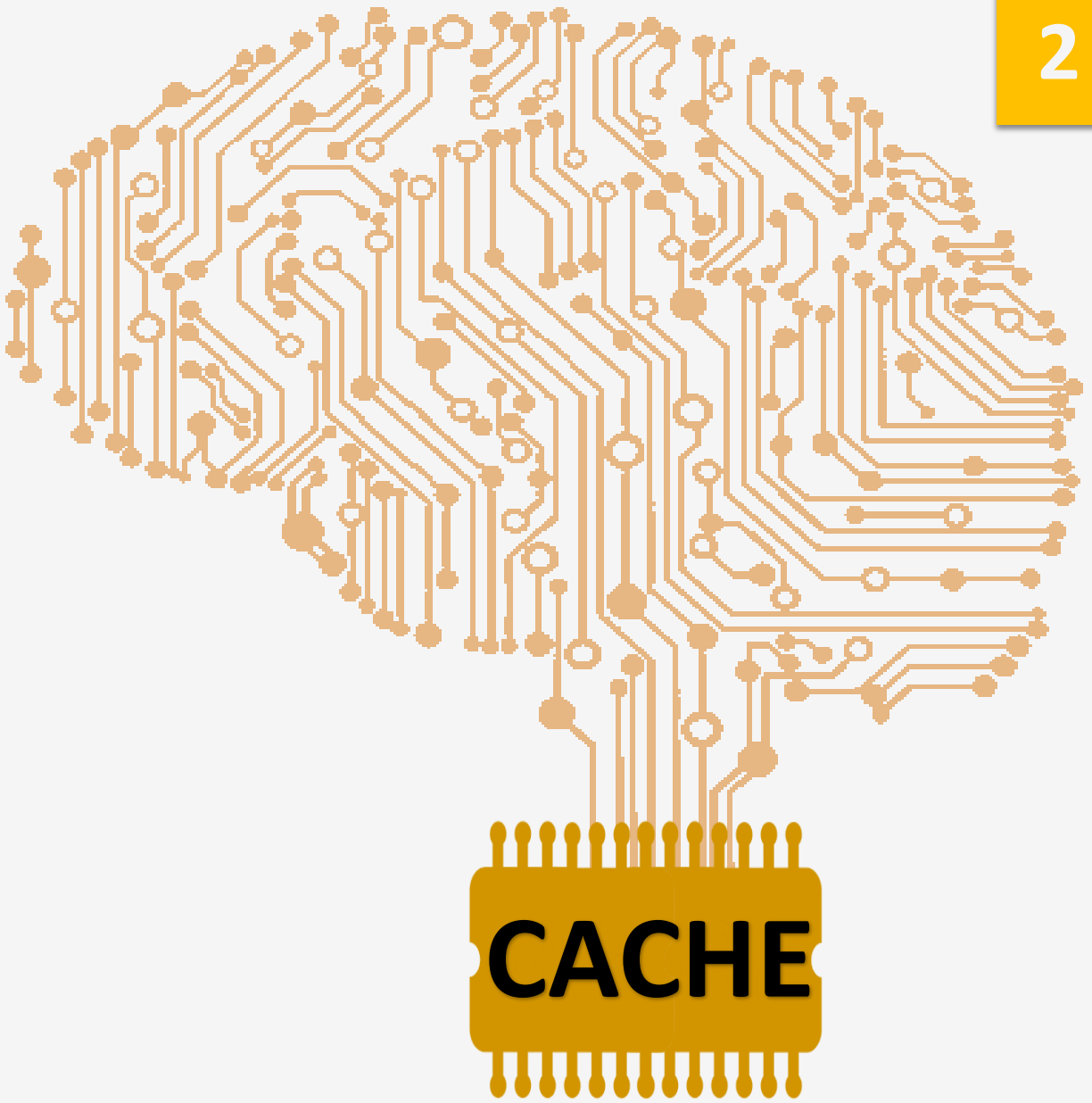
Driving Cache Replacement with ML-based LeCaR

Giuseppe Vietri†, Liana V. Rodriguez†, Wendy A. Martinez†, Steven Lyons†, Jason Liu†, Raju Rangaswami†, Ming Zhao‡, Giri Narasimhan†

† Florida International University

‡Arizona State University

Reinforcement Learning On Cache Replacement Algorithms



Outline

- Previous work
- Algorithm
- Results
- Conclusion

Hit-rate Performance vs. ARC

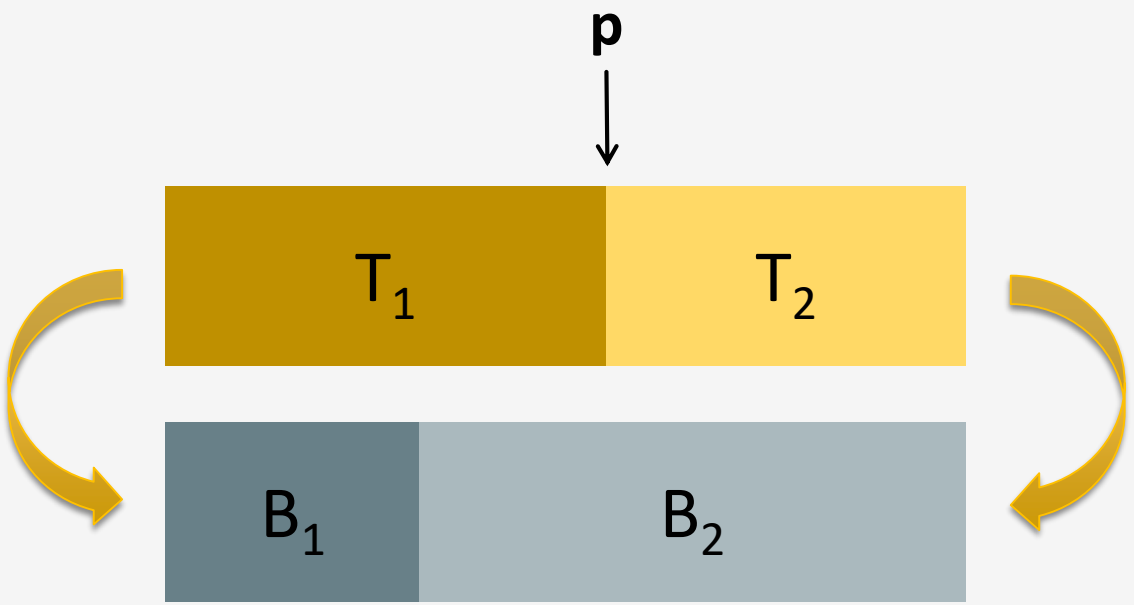
Cache Size	Non-Parameterized					Adaptive	Fixed-Parameter		
	LRU	LFU	FBR	LIRS	MQ	ARC	LRU-2	2Q	LRFU
1000	-6.1	-10.95	-1.97	-4.13	-1.07	0	0.37	1.55	1.59
2000	-3.61	-10.87	-2.1	-3.57	-1.89	0	-0.26	0.45	0.03
5000	-1.6	-10.49	-1.72	-0.86	-0.86	0	-0.47	0.45	1.48
10000	-1.17	-9.72	0.43	-1.52	-0.79	0	0.55	0.71	1.67
15,000	-0.77	-9.18	0.26	-1.41	-0.59	0	-0.18	0.42	1.66

Worse than ARC

Better than ARC

Megiddo, N., & Modha, D. S. (2004). Outperforming LRU with an adaptive replacement cache algorithm. *Computer*, *37*(4), 58-65.

Adaptive Replacement Cache (ARC)



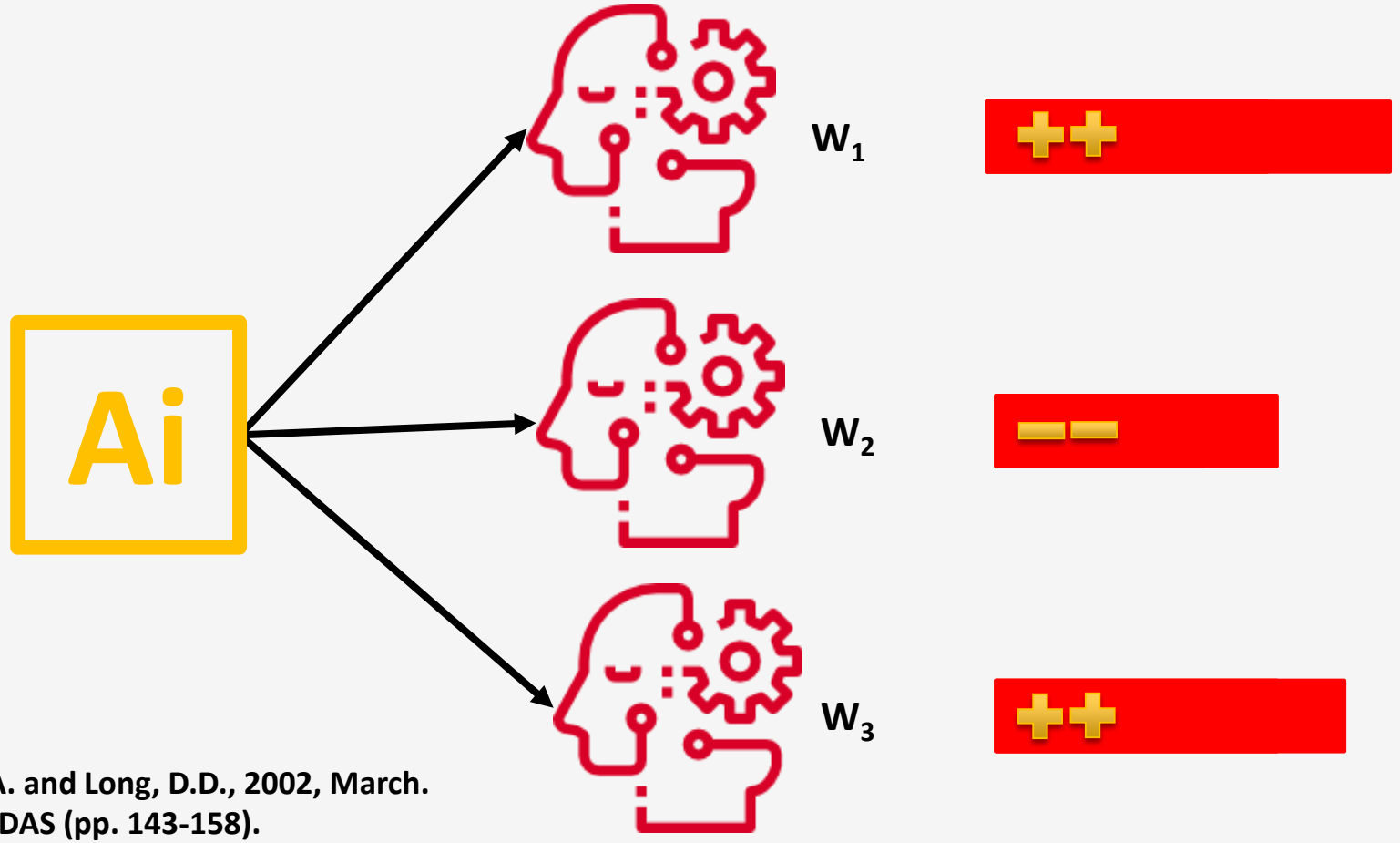
Megiddo, N., & Modha, D. S. (2004). Outperforming LRU with an adaptive replacement cache algorithm. Computer, 37(4), 58-65.

Strengths of ARC

- Manages both **recent** items as well as **frequent** items
- Dynamically **adapts** – Self-tuning
- **Low overhead**
- Competitive Hit-Rate **performance**

Conventional Online Learning Systems

1. Choose Expert
2. Follow Advice
3. Adjust Weights
4. Repeat



Ari, I., Amer, A., Gramacy, R.B., Miller, E.L., Brandt, S.A. and Long, D.D., 2002, March. ACME: Adaptive Caching Using Multiple Experts. In WDAS (pp. 143-158).

Disadvantages of current ML-based methods

Not efficient

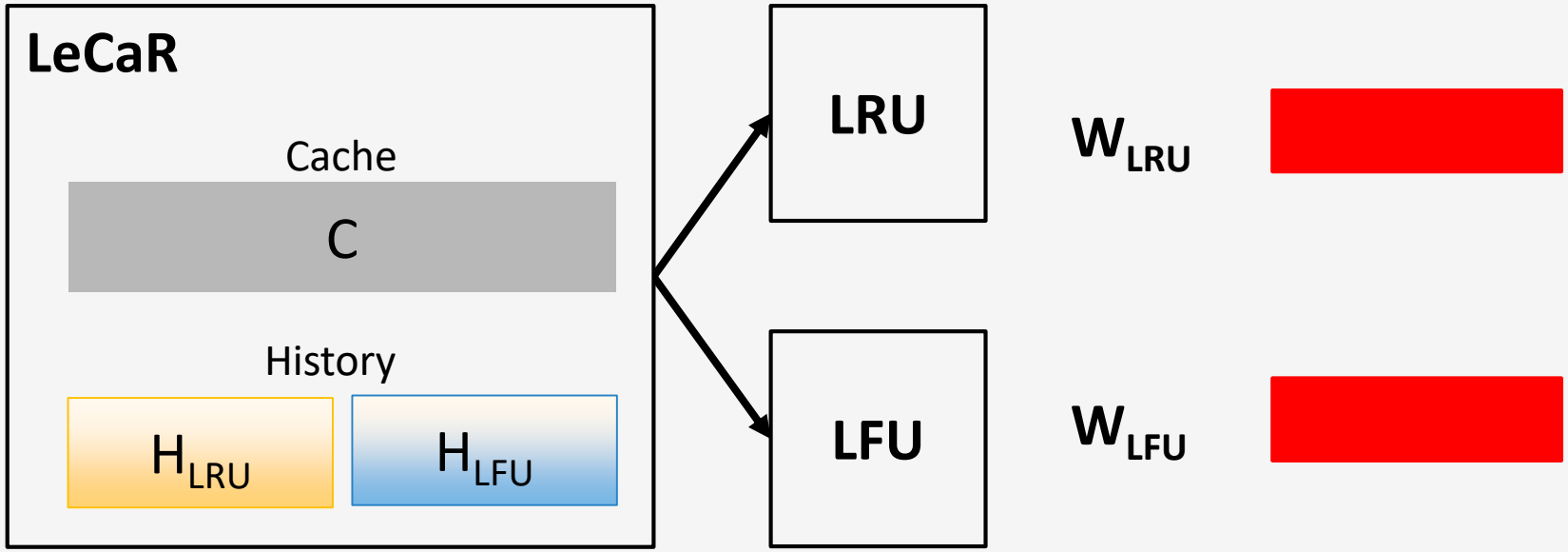
Not competitive

ML-based LeCaR

- **LRU** and **LFU**
- **Efficiency**
- **Learning** the optimal mix.

The LeCaR approach

- **LRU** and **LFU**
- **History**
- **Regret**



Lee, D., Choi, J., Kim, J. H., Noh, S. H., Min, S. L., Cho, Y., & Kim, C. S. (2001). LRFU: A spectrum of policies that subsumes the least recently used and least frequently used policies. IEEE transactions on Computers, (12), 1352-1361.

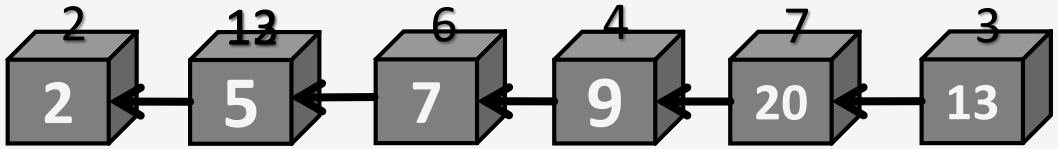
LeCaR (Hit)

```
Input: requested page q
If q in C then
    C.Update(q)
else
    if q is in H_LRU then
        H_LRU.Delete(q)
    if q is in H_LFU then
        H_LFU.Delete(q)
    UpdateWeights(q)
    if C is full then
        action = (LRU, LFU)~(w_LRU,w_LFU)
        if act == LRU then
            if H_LRU is full then
                H_LRU.Delete(LRU(H_LRU))
            H_LRU.Add(LRU(C))
            C.Delete(LRU(C))
        else
            if H_LFU is full then
                H_LFU.Delete(LFU(H_LFU))
            H_LFU.Add(LFU(C))
            C.Delete(LRU(C))
    C.Add(q)
```

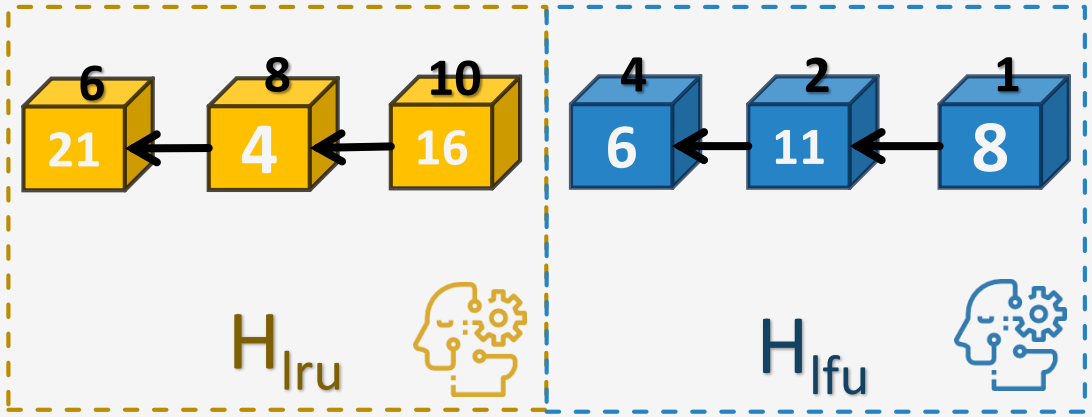
Request

Weights

Cache



History

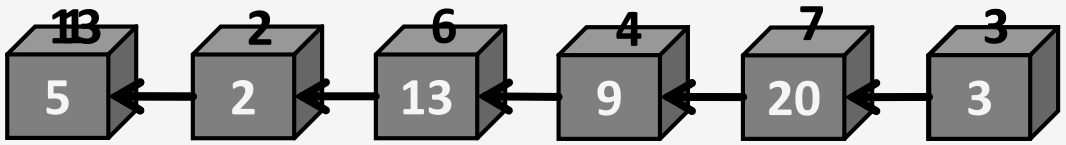


LeCaR (Miss not in History)

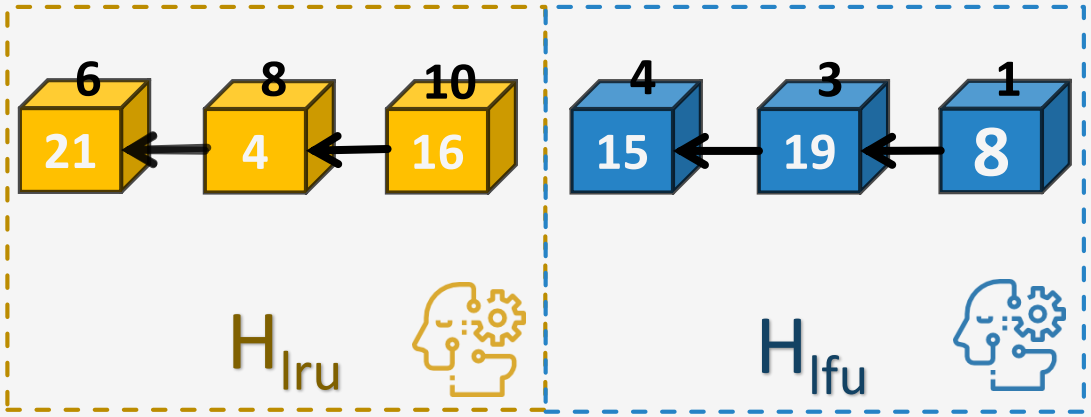
```
Input: requested page q
If q in C then
    C.Update(q)
else
    if q is in H_LRU then
        H_LRU.Delete(q)
    if q is in H_LFU then
        H_LFU.Delete(q)
    UpdateWeights(q)
    if C is full then
        action = (LRU, LFU)~(w_LRU,w_LFU)
        if act == LRU then
            if H_LRU is full then
                H_LRU.Delete(LRU(H_LRU))
            H_LRU.Add(LRU(C))
            C.Delete(LRU(C))
        else
            if H_LFU is full then
                H_LFU.Delete(LFU(H_LFU))
            H_LFU.Add(LFU(C))
            C.Delete(LRU(C))
    C.Add(q)
```

Request

Cache



History



Weight Update

α = Learning rate

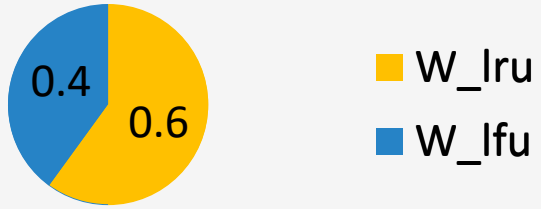
	LRU Regret	LFU Regret
Update	$w_{LRU} := \alpha * w_{LRU}$	$w_{LFU} := \alpha * w_{LFU}$

LeCaR (Miss in History)

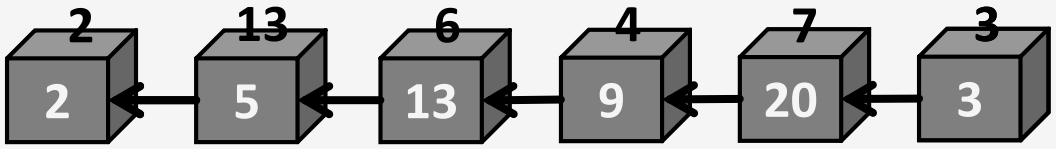
```
Input: requested page q
If q in C then
    C.Update(q)
else
    if q is in H_LRU then
        H_LRU.Delete(q)
    if q is in H_LFU then
        H_LFU.Delete(q)
    UpdateWeights(q)
    if C is full then
        action = (LRU, LFU)~(w_LRU,w_LFU)
        if act == LRU then
            if H_LRU is full then
                H_LRU.Delete(LRU(H_LRU))
            H_LRU.Add(LRU(C))
            C.Delete(LRU(C))
        else
            if H_LFU is full then
                H_LFU.Delete(LFU(H_LFU))
            H_LFU.Add(LFU(C))
            C.Delete(LRU(C))
    C.Add(q)
```

Request

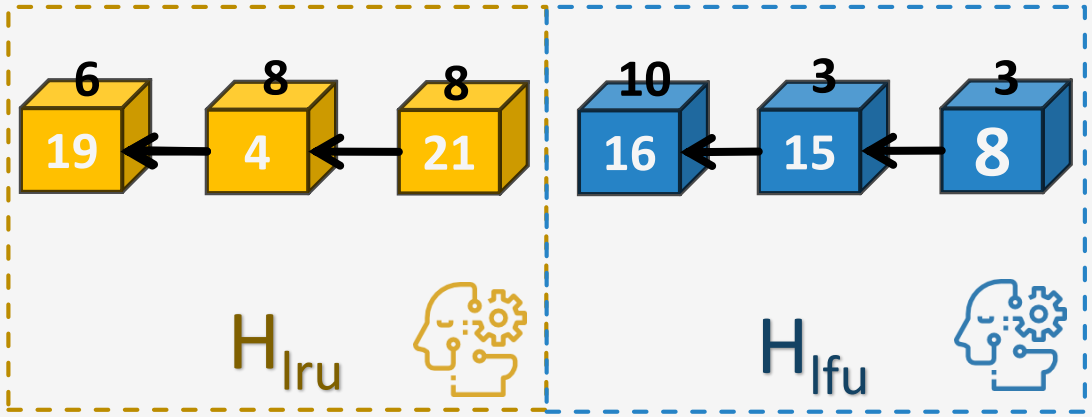
Weights



Cache



History



Experiments

- **8** Workloads
- 3 days of data[FIU datasets]
- Small, Medium and Large **Cache Sizes**
- Fixed **Learning Rate**

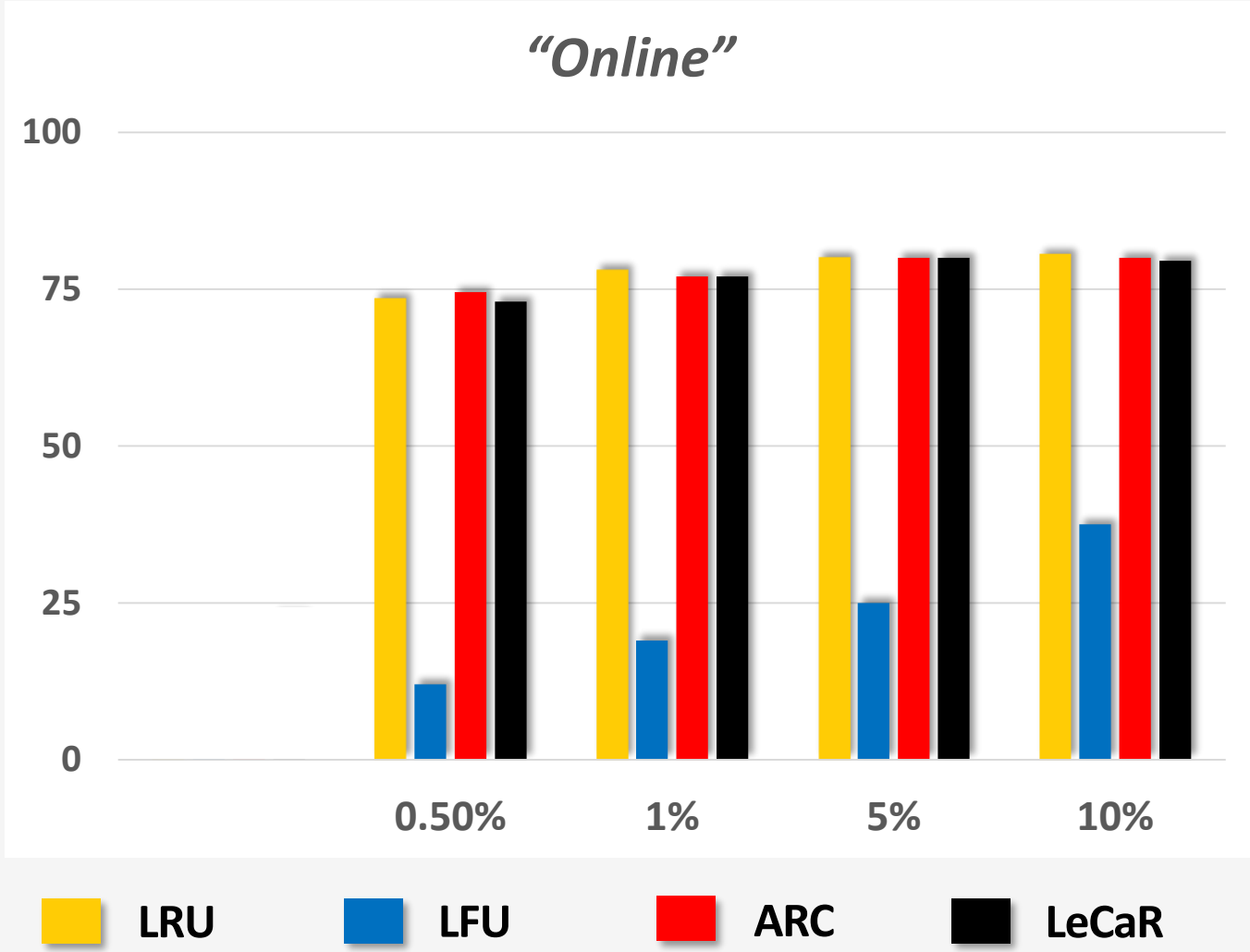
Data used for the Experiments

Casa, Ikki, Madmax and Topgun	Four different end-user/developer home directories
Online	Departments online course management system
Webresearch	Document store for research projects
Webmail	Mail server of the FIU Computer and Engineering department using Postfix
Webuser	Web server hosting faculty, staff, and graduate student web sites

Collected at the School of Computing and Information Sciences at FIU.

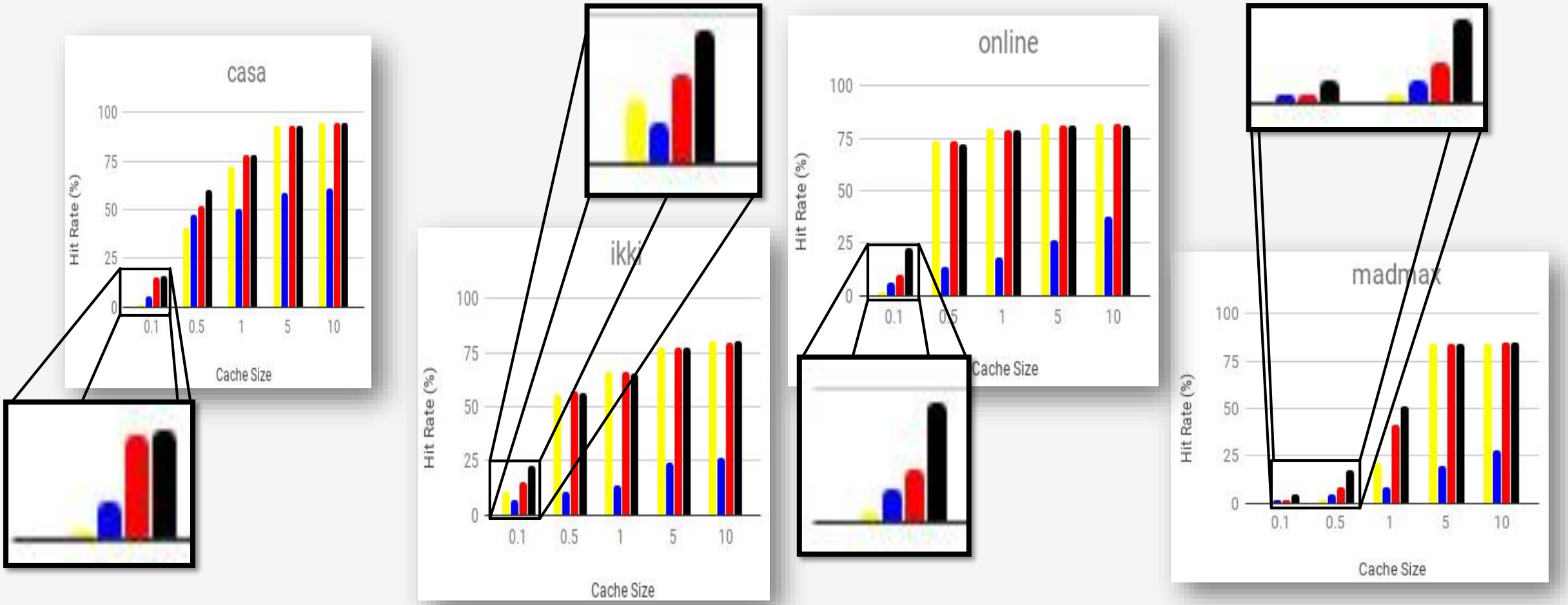
R. Koller , R. Rangaswami. I/O deduplication: Utilizing content similarity to improve i/o performance. In Proc. 8th USENIX Conference on File and Storage Technologies (2009), FAST 10.

Results



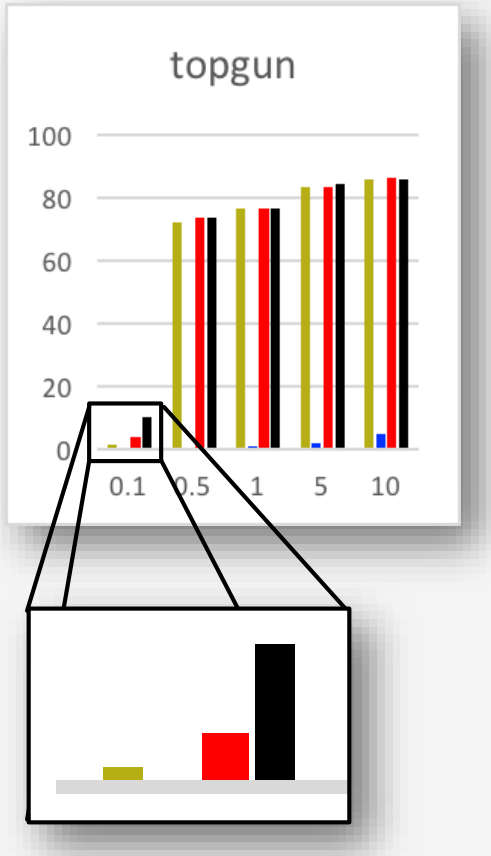
Results

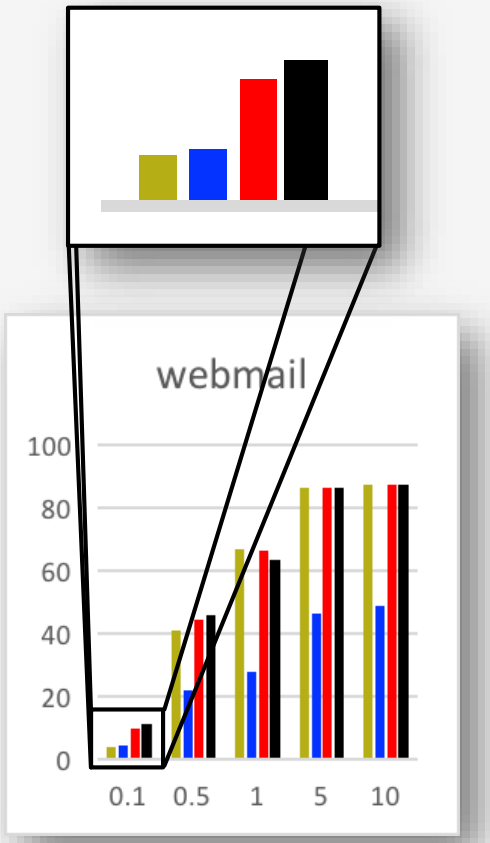
LRU
ARC
LFU
LeCaR

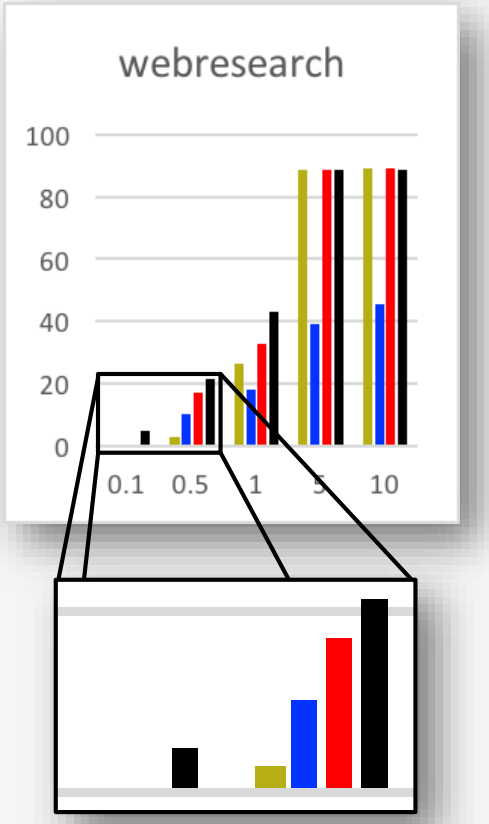


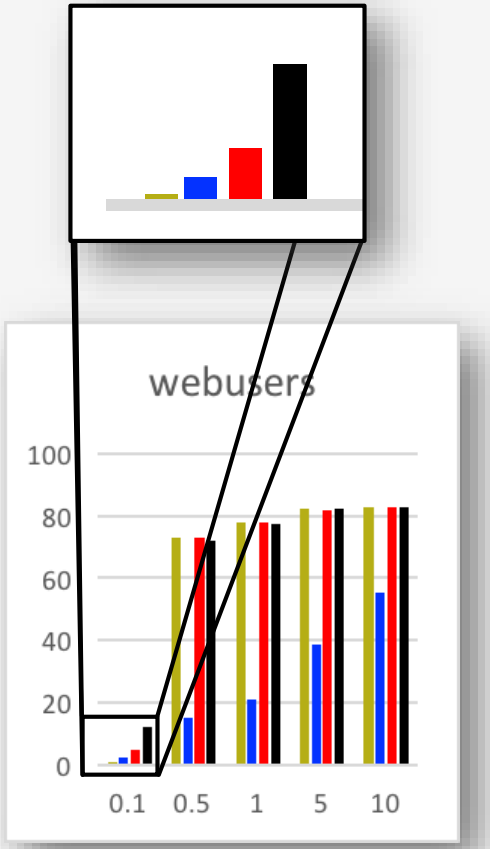
Results

LRU
LFU
ARC
LeCaR

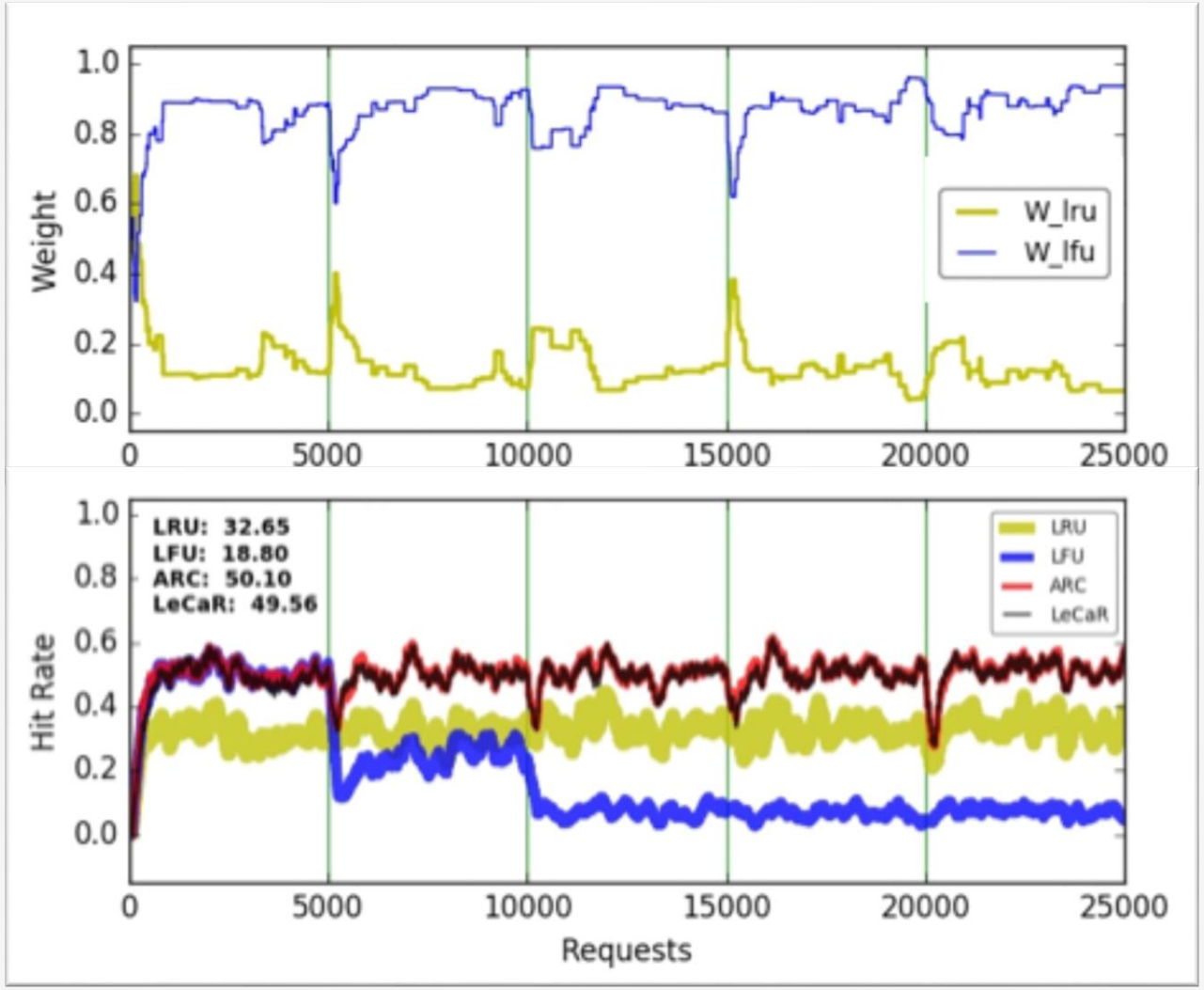




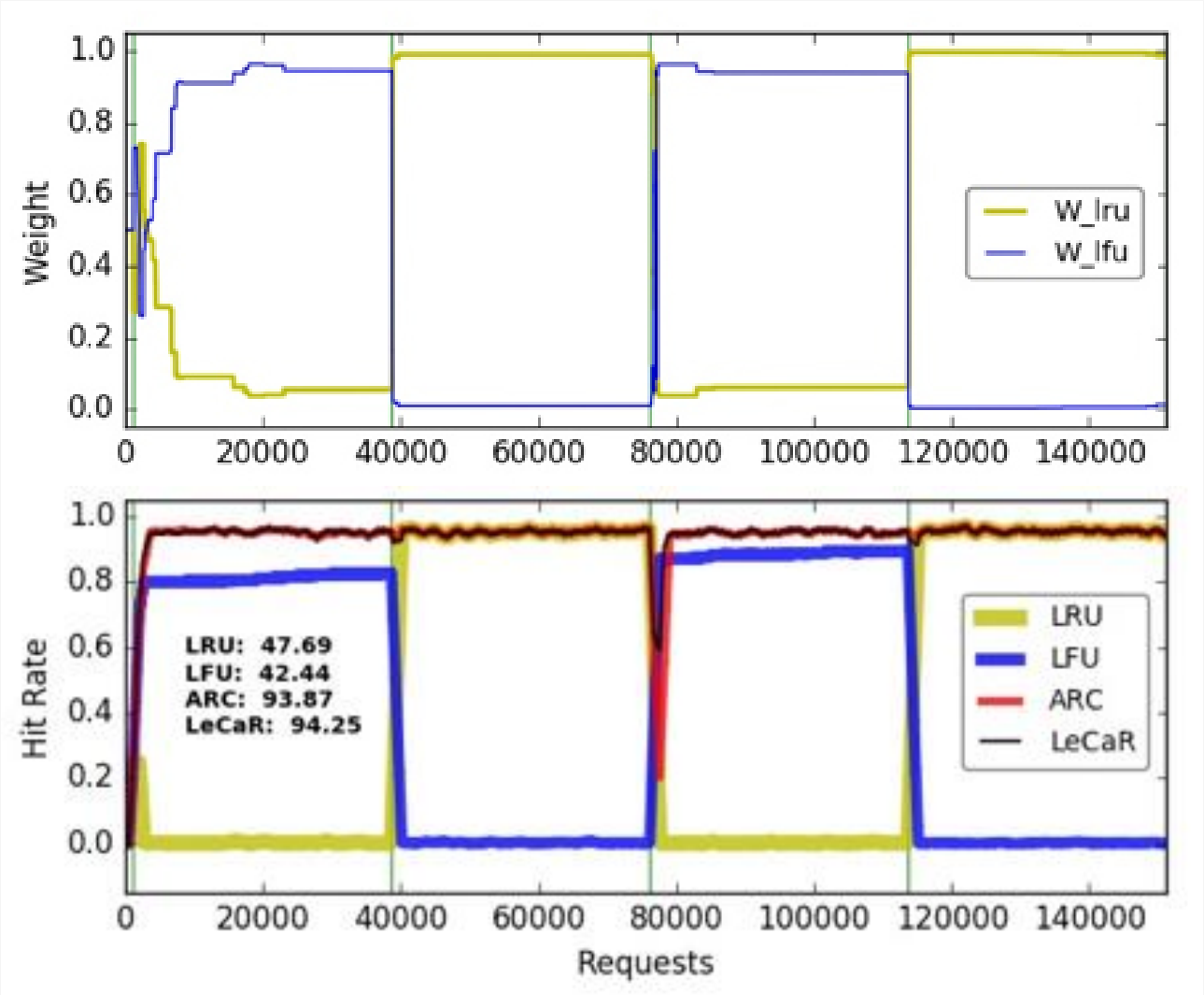




Synthetic Data



Synthetic Data

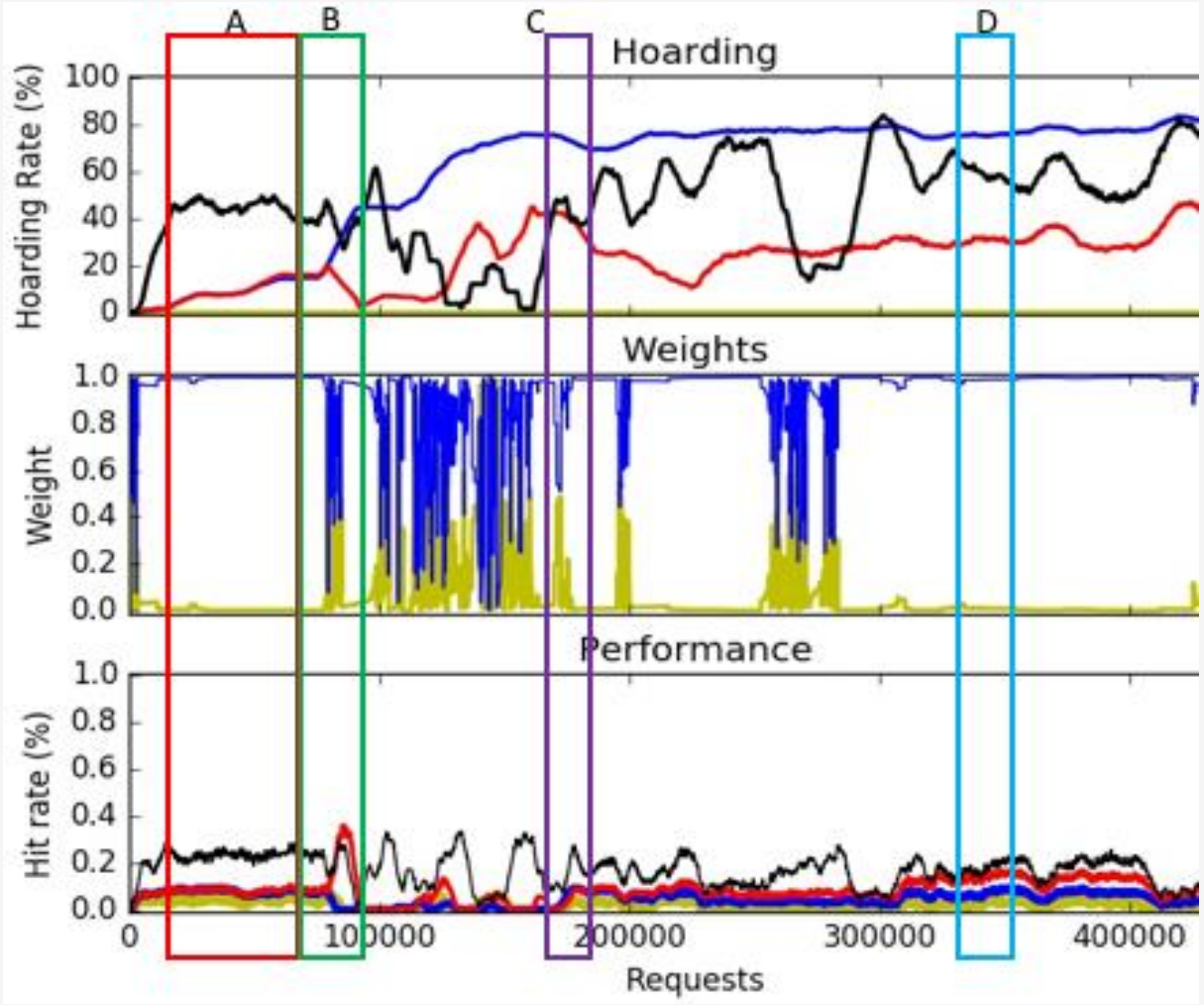


Hoarding Rate

- Definition
- A: Stable Period
- B: LFU gets penalized

Hoarded Page:

- Accessed at least twice
- Not among the last 2N unique pages



Conclusion

- ML-based **LeCaR**
- Small cache size
- **Real + Synthetic** experiments