

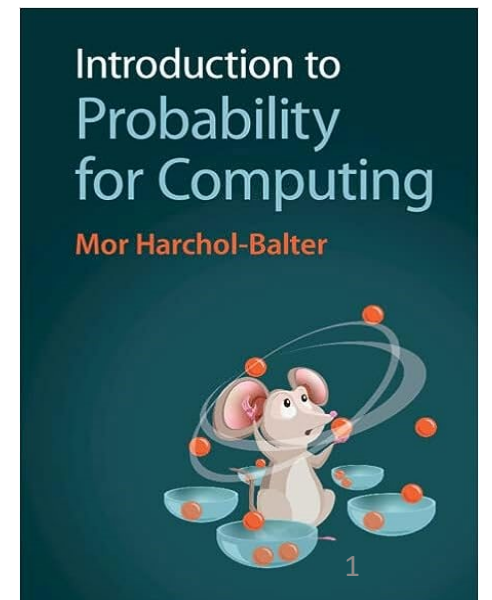
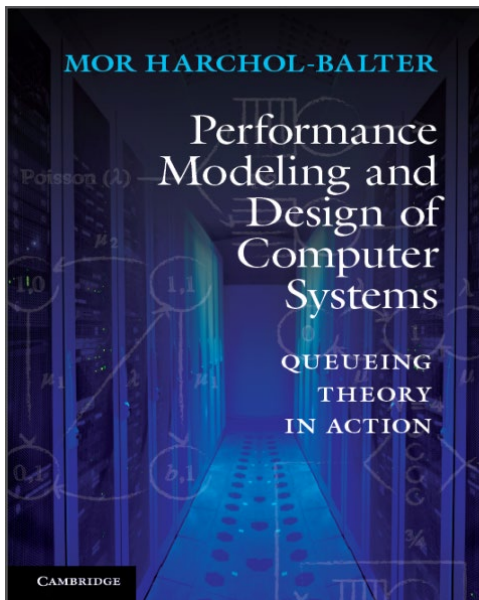
Can Increasing the Hit Ratio **Hurt** Cache Throughput?

Ziyue Qiu

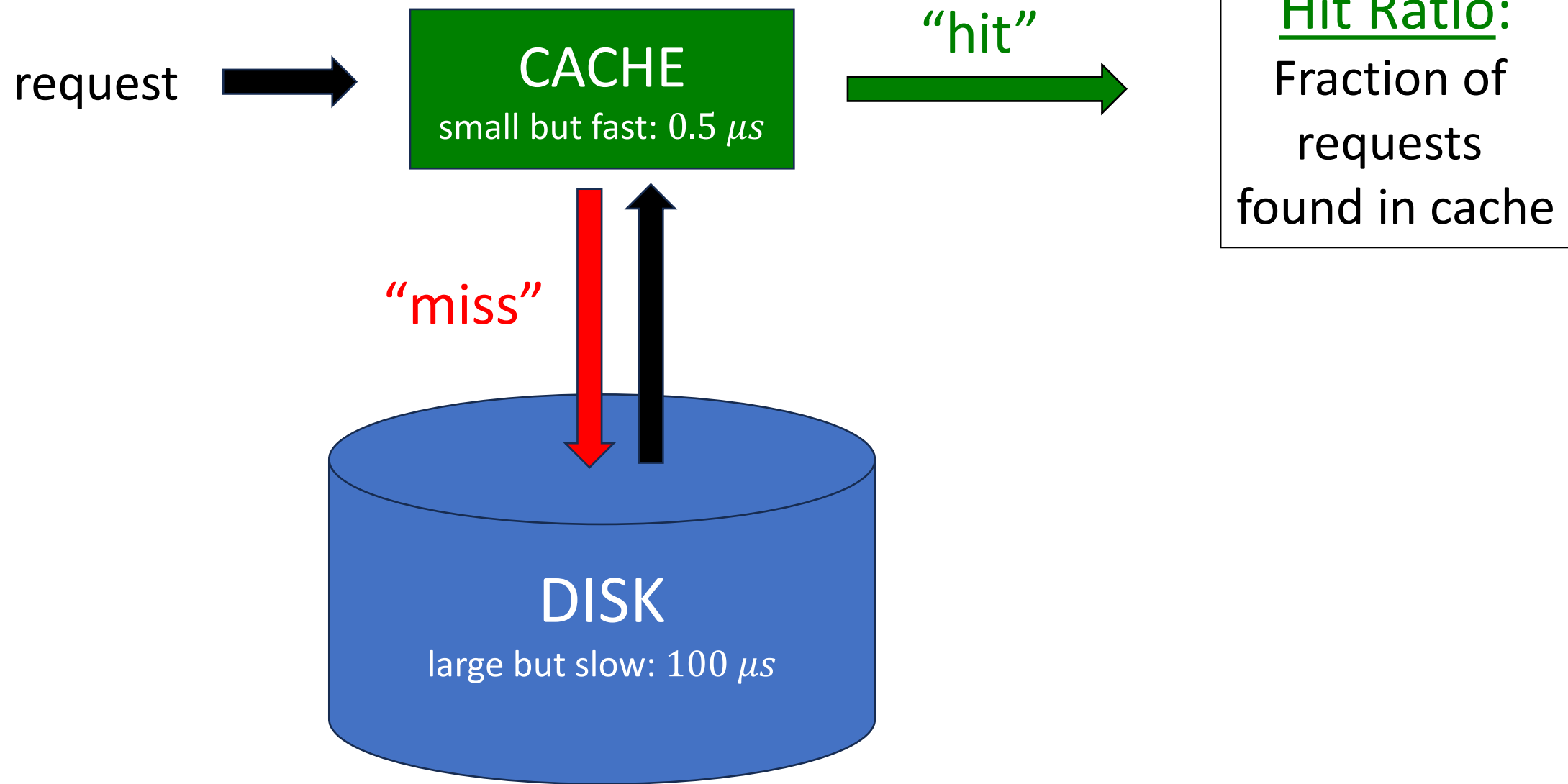
Juncheng Yang

Mor Harchol-Balter

Carnegie Mellon University
Computer Science Dept.



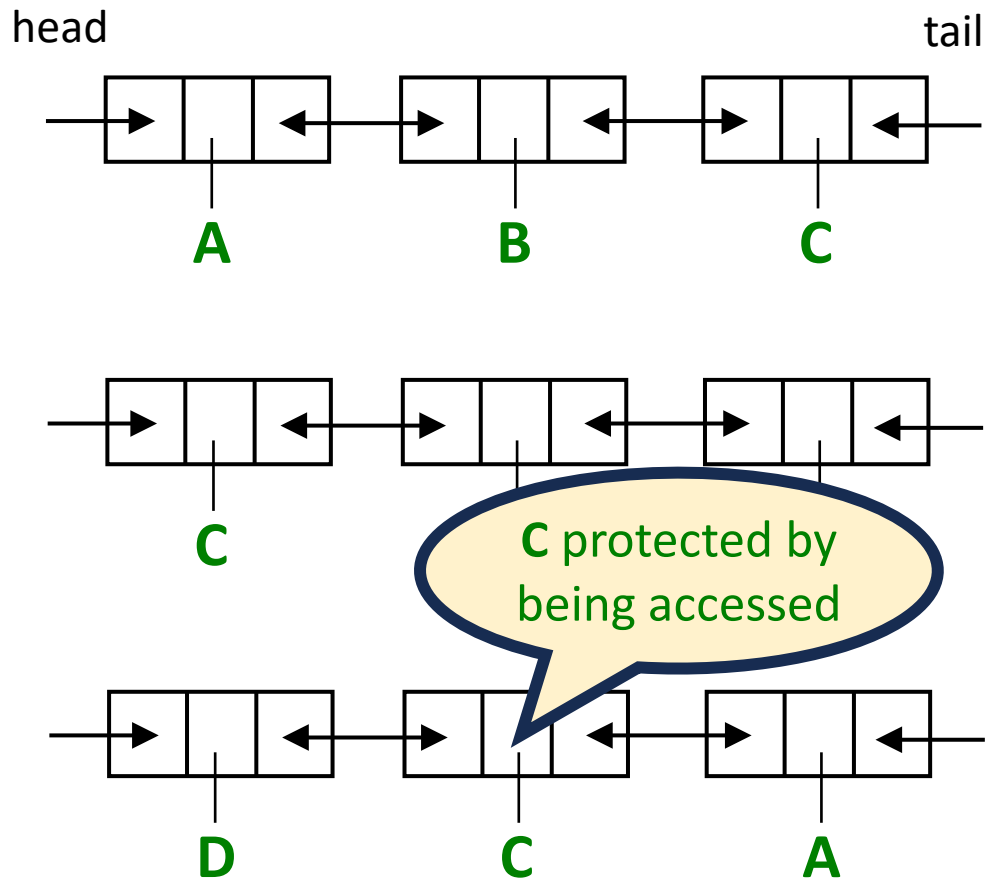
What is a Cache?



Cache Eviction Policies

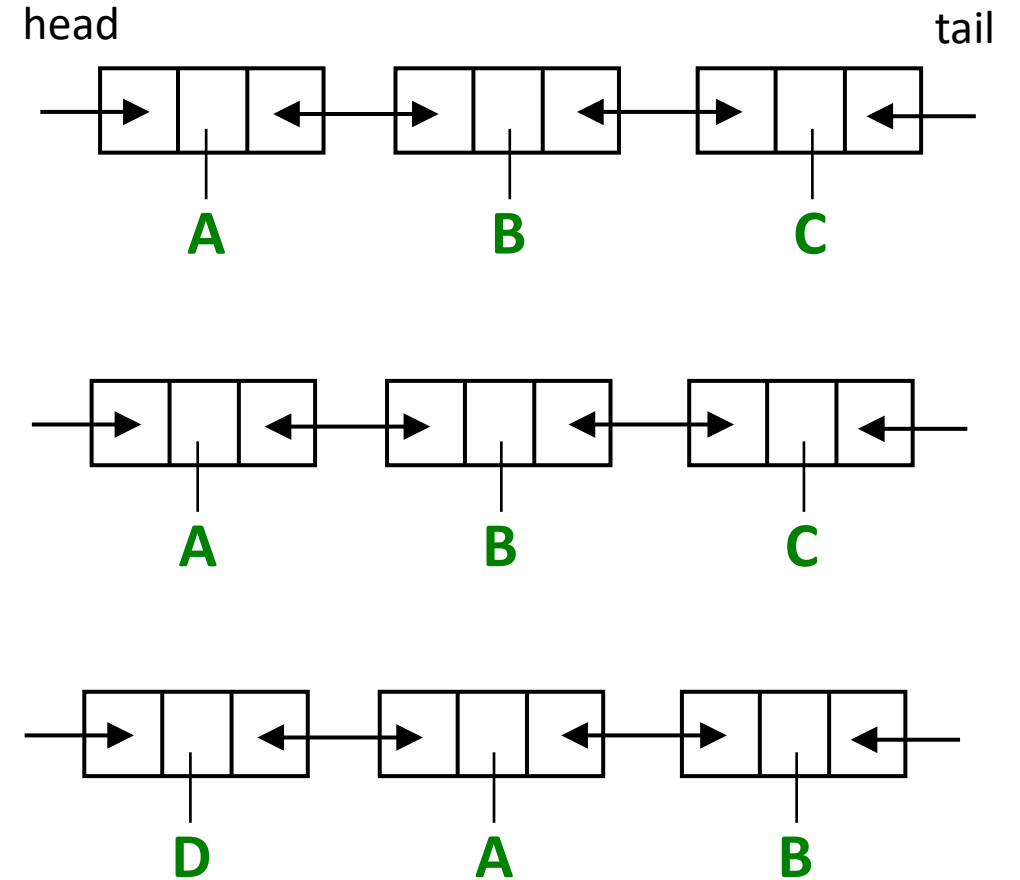
Most Common: LRU

Evict least recently accessed item



Also Common: FIFO

Evict least recently inserted item



access C

access D

Why LRU is most popular

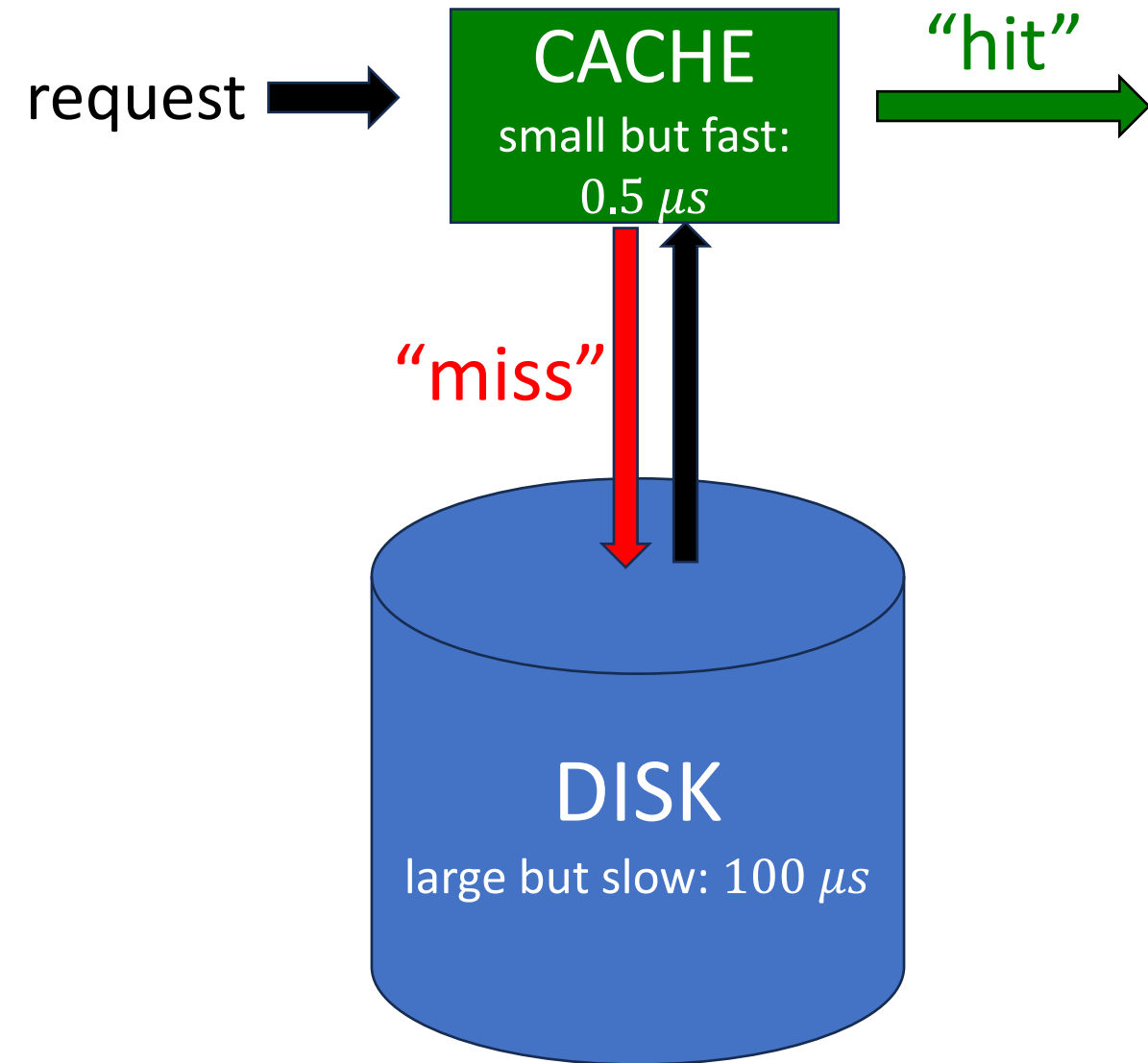


Peter Denning

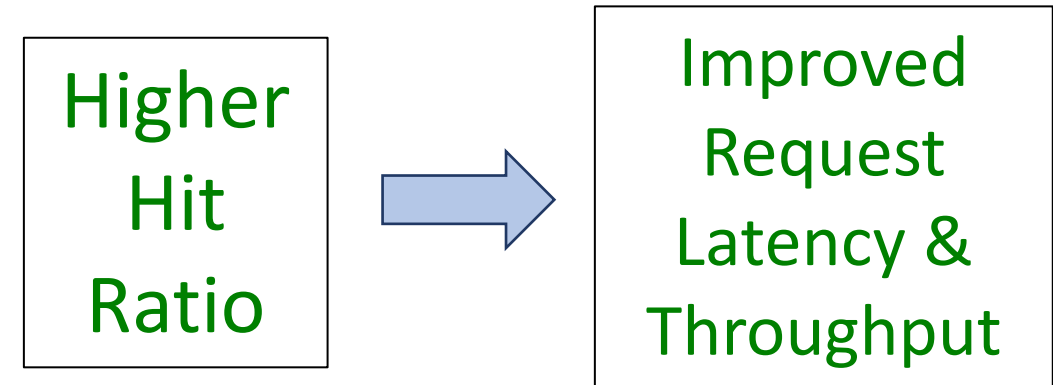
Data Access patterns show
temporal locality.
Recently accessed data is more
likely to be accessed again.



LRU is designed
for this!

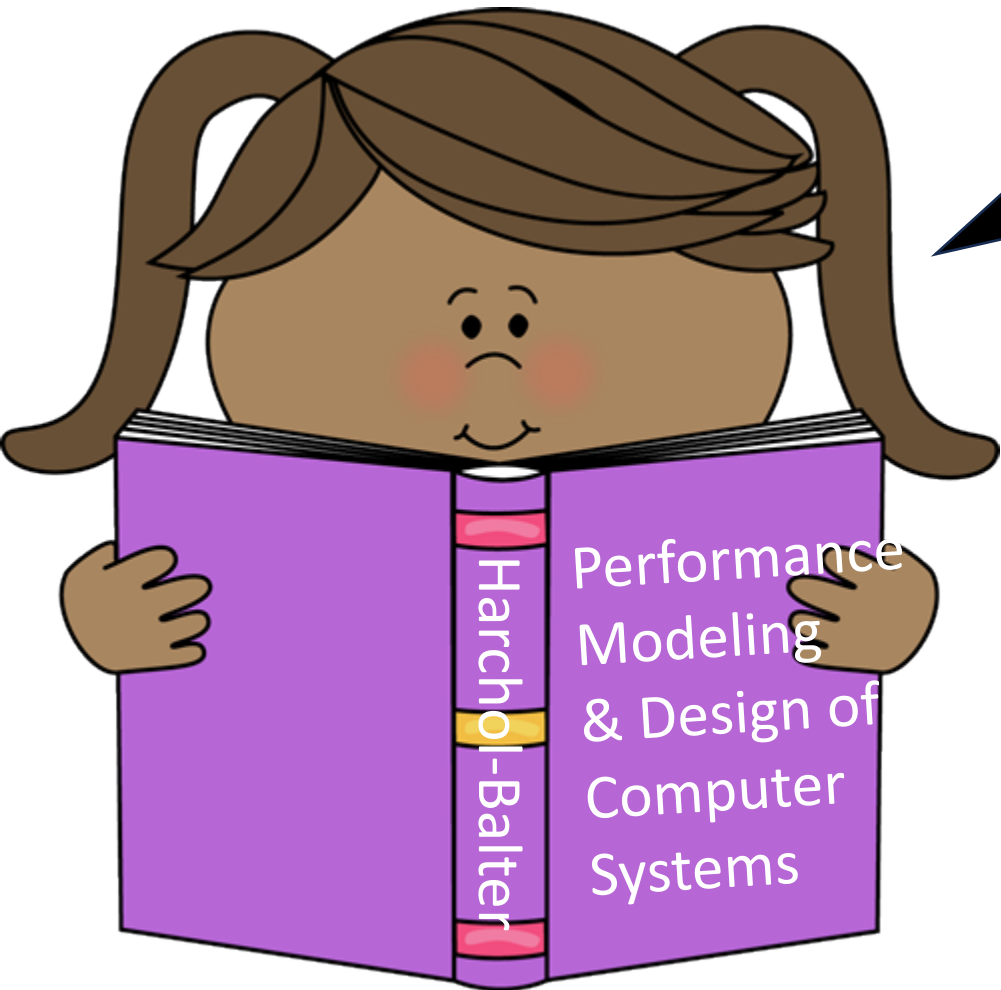


Common Wisdom



Beckman, Berg, Berger, Bunt, Carrig,
Chen, Cheng, Cho, Cidon, Ciucu,
Crooks, Eager, Feng, Gandhi, Ganger,
Grosof, Gunasekar, Harchol-Balter,
Hellerstein, Henningsen, Kozuch, Lakew,
Li, Lu, McAllister, Sabnis, Schmitt,
Sitaraman, Stoica, Sunderrajan, Tran,
Vinayak, Willick, Yang, Yu, Yue, Zhu ...

Ziyue Qiu



But in my caching system:

Higher
Hit
Ratio

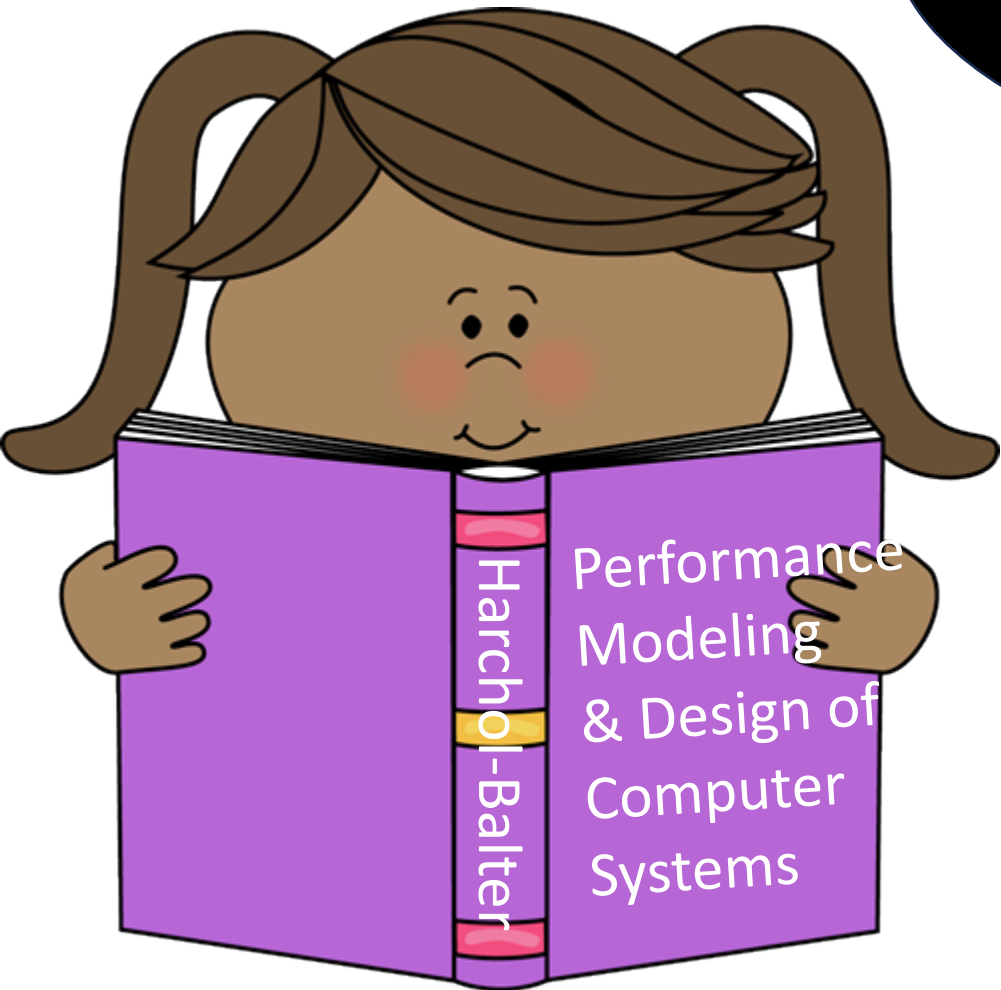


WORSE
Latency &
Throughput

Hmmm...
Super
interesting!



Ziyue Qiu

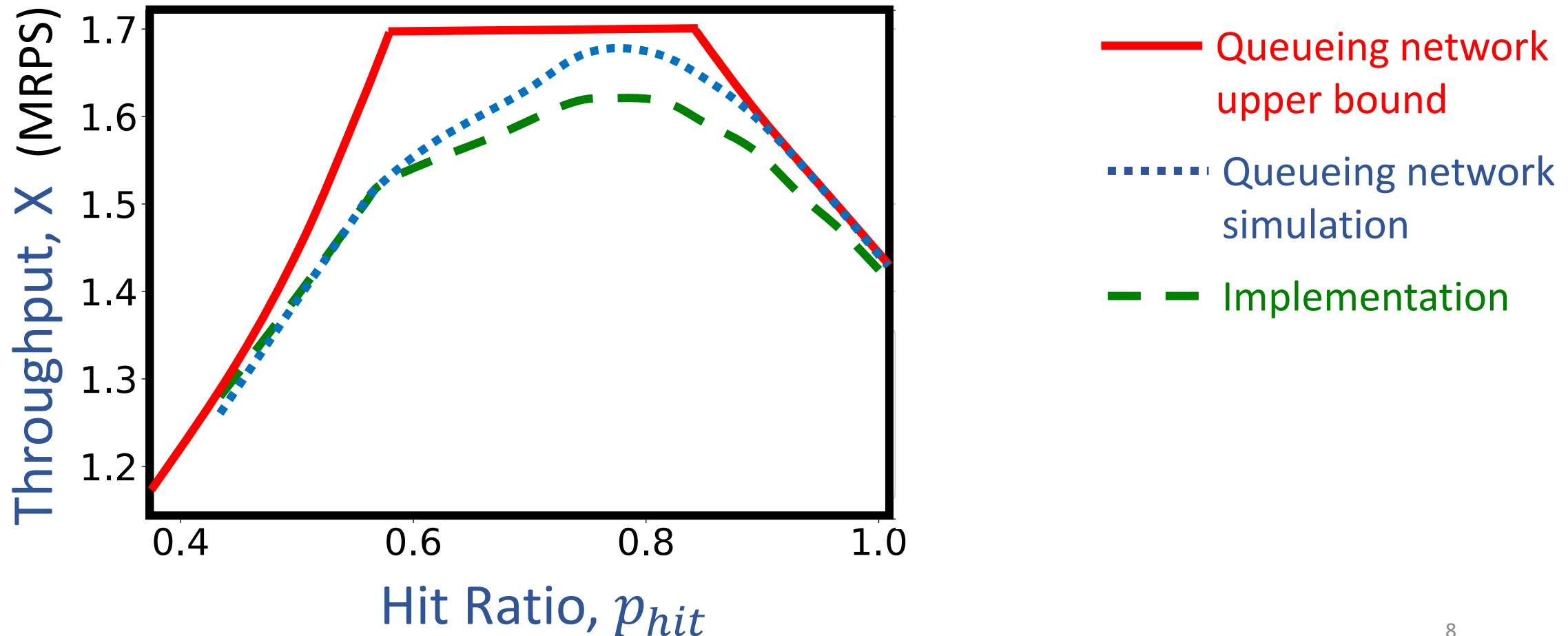


Seems no one has actually studied the **relationship** between hit ratio and throughput/latency...



Thesis of Talk

For today's LRU-based caching systems,



Caching System Implementation

- ❑ Prototype of Meta's HHVM cache
- ❑ Run on CloudLab platform
- ❑ Requests are for 4KB blocks from Zipfian ($\theta = 0.99$) popularity distribution
- ❑ Intel Xeon Platinum CPU for cache with 72 cores.

❑ KEY POINTS:

➤ DRAM-based cache

- Very fast ($0.51 \mu s$) & Highly concurrent (72 cores)

➤ SSD-based disk

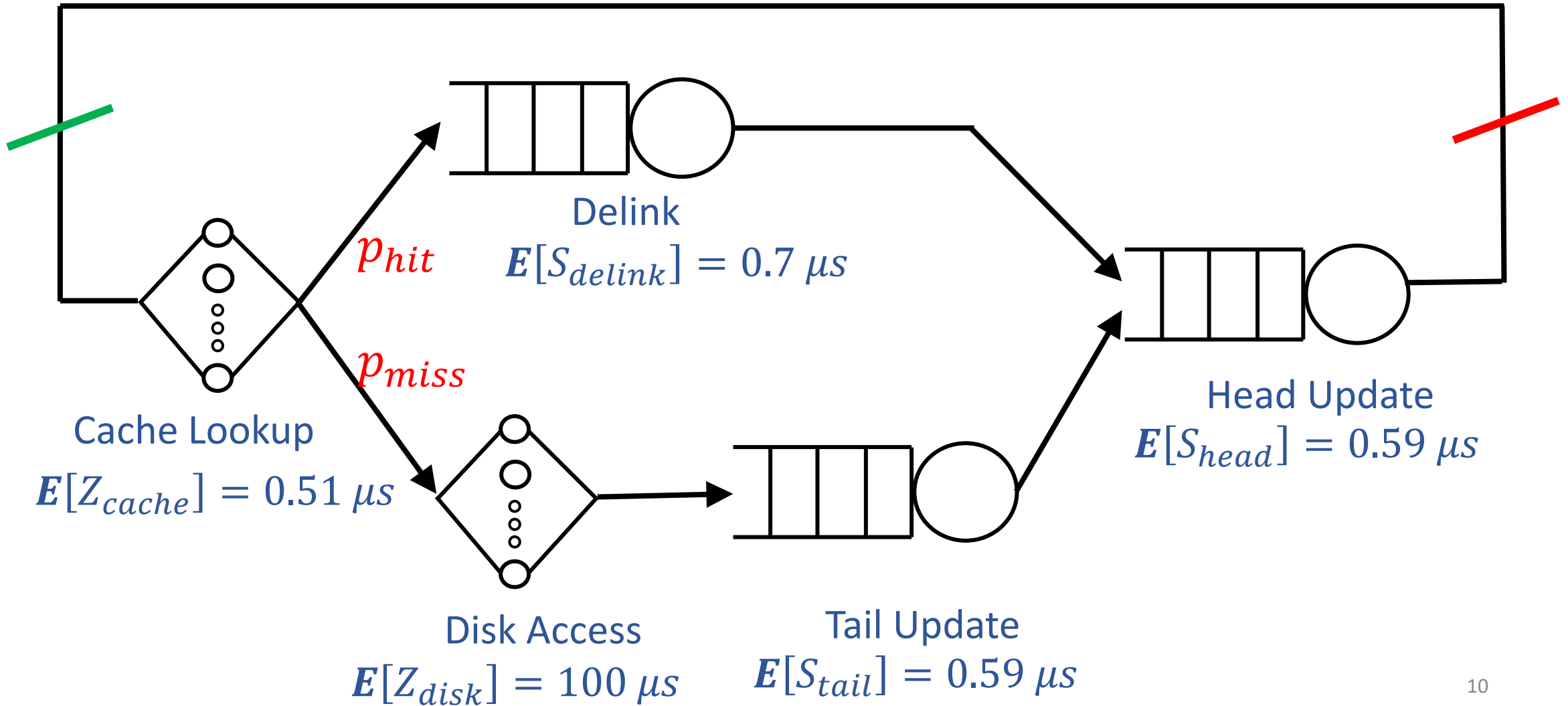
- $100 \mu s$ but we emulate range from $5 \mu s - 500 \mu s$
- Highly concurrent (72 concurrent requests)

➤ Each request is handled by a single core.

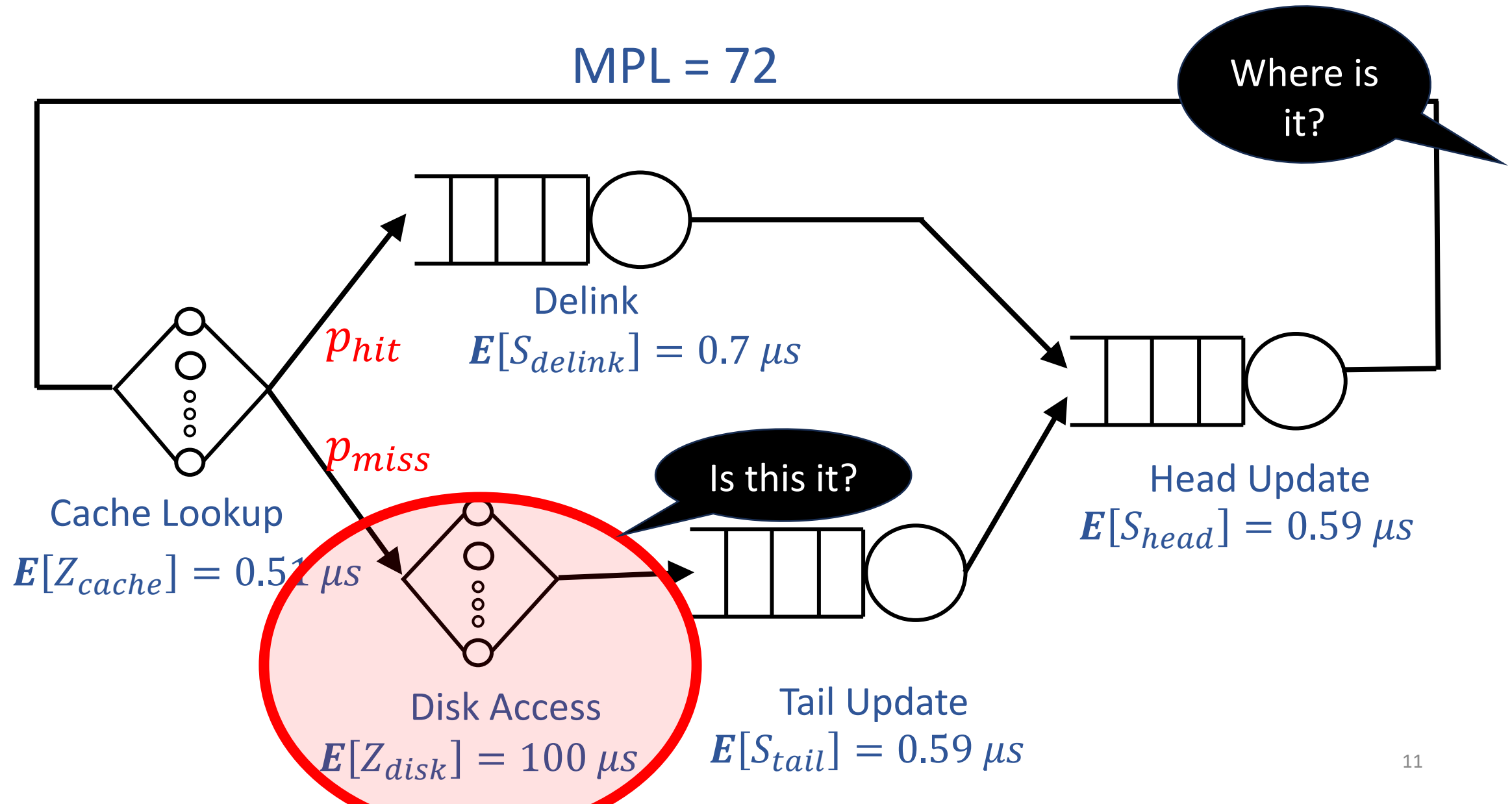
Total # requests in system is limited by #cores → MPL = 72

Queueing model for LRU caching system

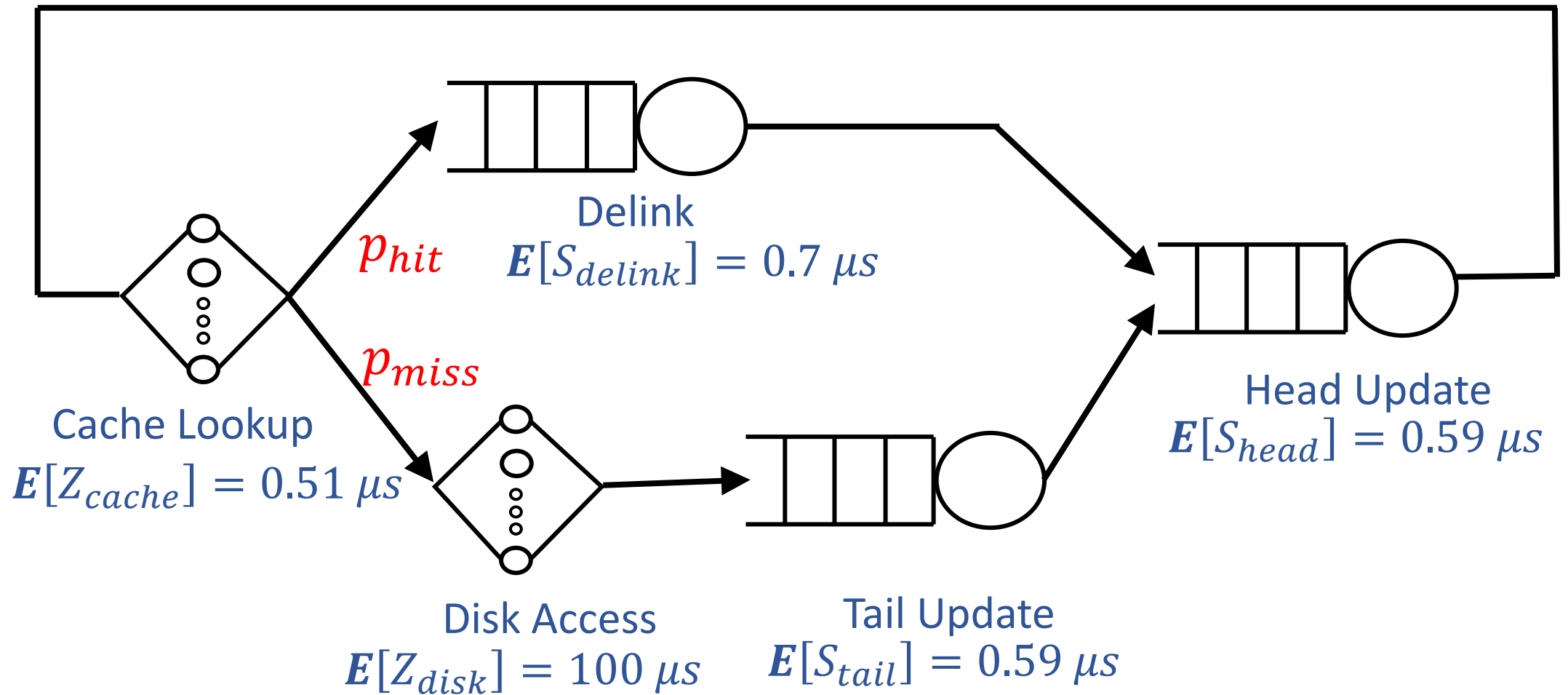
MPL = 72



Q-theory: “Find the bottleneck”



MPL = 72

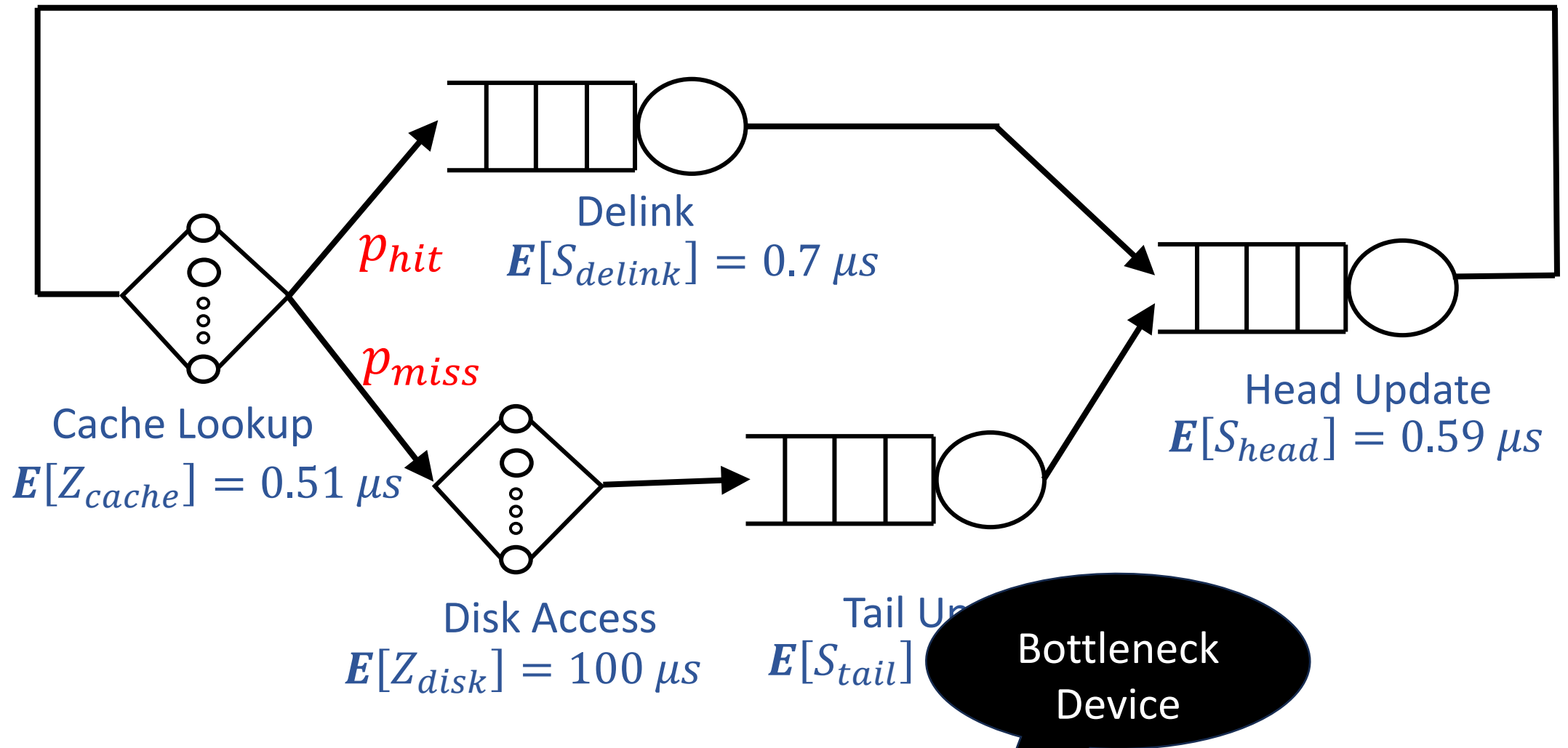


STEP 1:

Think
time

$$E[Z] = E[Z_{cache}] + (1 - p_{hit})E[Z_{disk}]$$

MPL = 72



STEP 2:
Device
demands

$$D_{delink} = 0.7 p_{hit}$$

$$D_{tail} = 0.59 (1 - p_{hit})$$

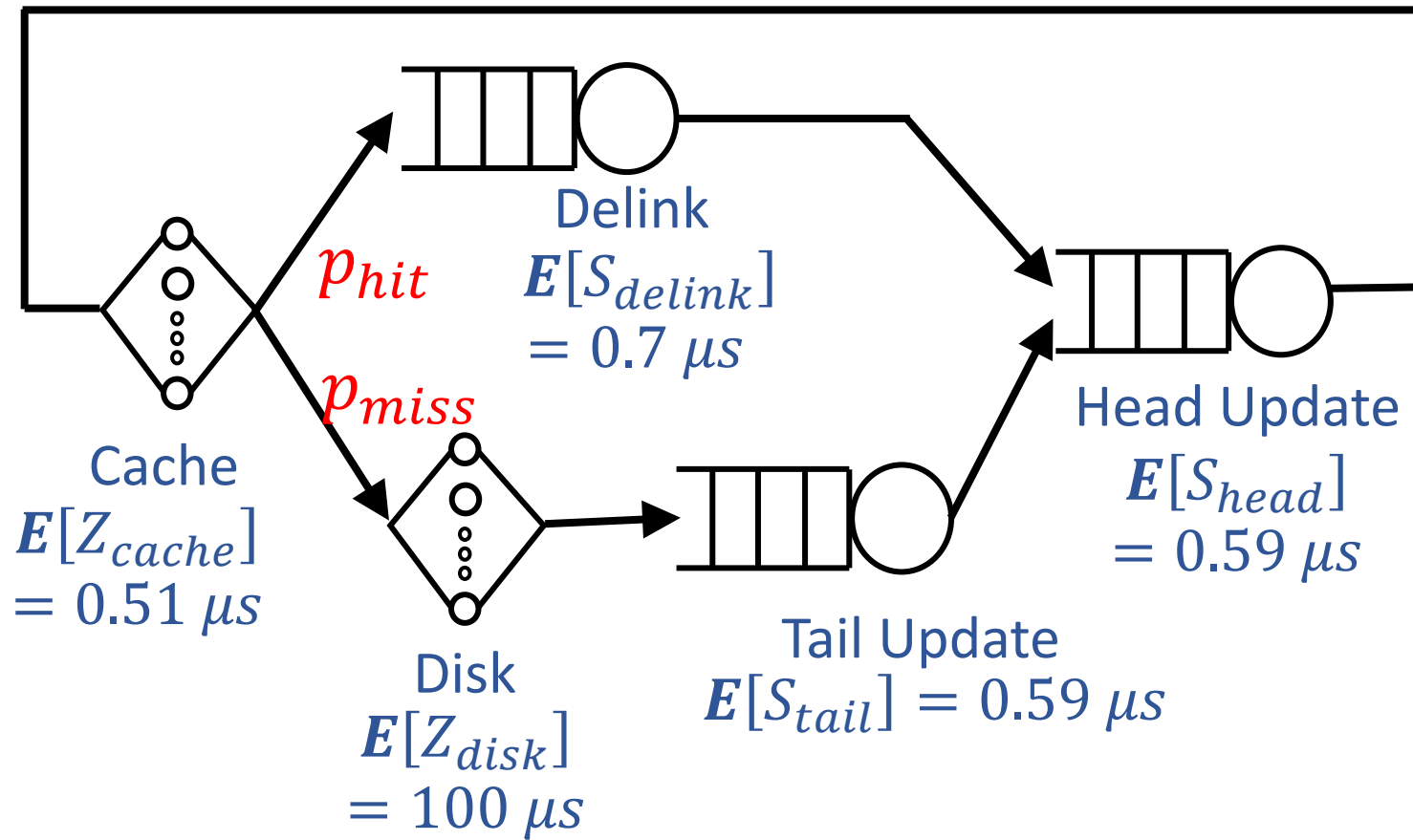
$$D_{head} = 0.59$$



$D_{max} =$

$$\left\{ \begin{array}{l} D_{head} \text{ if } p_{hit} < 0.84 \\ D_{delink} \text{ if } p_{hit} \geq 0.84 \end{array} \right.$$

MPL = 72



$$E[Z] = E[Z_{cache}] + p_{miss}E[Z_{disk}]$$

$$D_{delink} = 0.7 p_{hit}$$

$$D_{tail} = 0.59 (1 - p_{hit})$$

$$D_{head} = 0.59$$

$$D = D_{delink} + D_{tail} + D_{head}$$

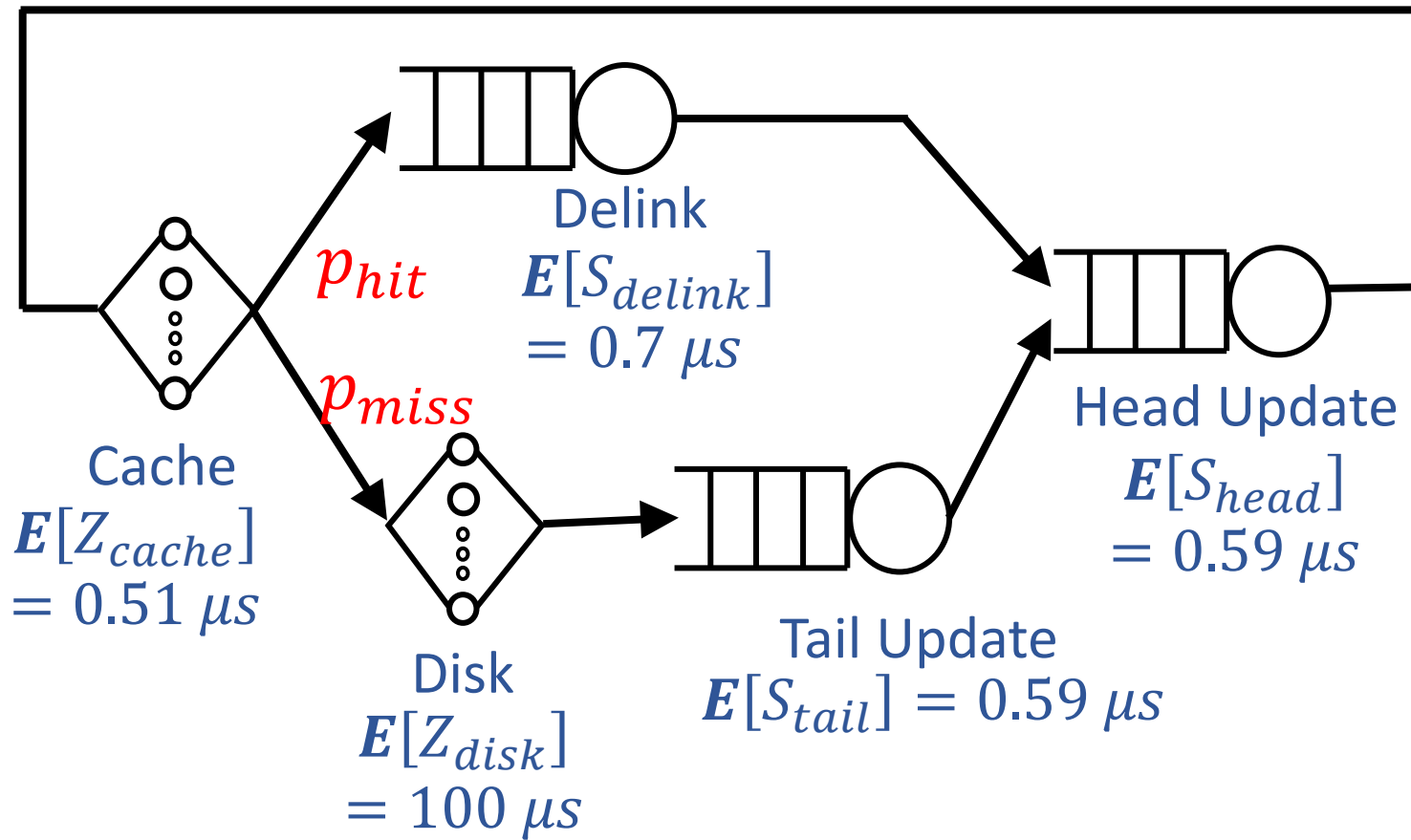
$$\text{Throughput} = X \leq \min \left(\frac{\text{MPL}}{D + E[Z]}, \frac{1}{D_{max}} \right)$$

$$\frac{72}{101.1 - 99.3p_{hit}}$$

$$\frac{1}{\max(0.59, 0.7p_{hit})}$$

$$D_{max} = \begin{cases} D_{head} & \text{if } p_{hit} < 0.84 \\ D_{delink} & \text{if } p_{hit} \geq 0.84 \end{cases}$$

MPL = 72



$$\text{Throughput} = X \leq \min \left(\frac{\text{MPL}}{D + E[Z]}, \frac{1}{D_{\max}} \right)$$

$$\frac{72}{101.1 - 99.3p_{\text{hit}}}$$

$$\frac{1}{\max(0.59, 0.7p_{\text{hit}})}$$

3 Regimes

❖ $p_{\text{hit}} < 0.59$

→ $X = \text{Left term}$

→ X increases with p_{hit}

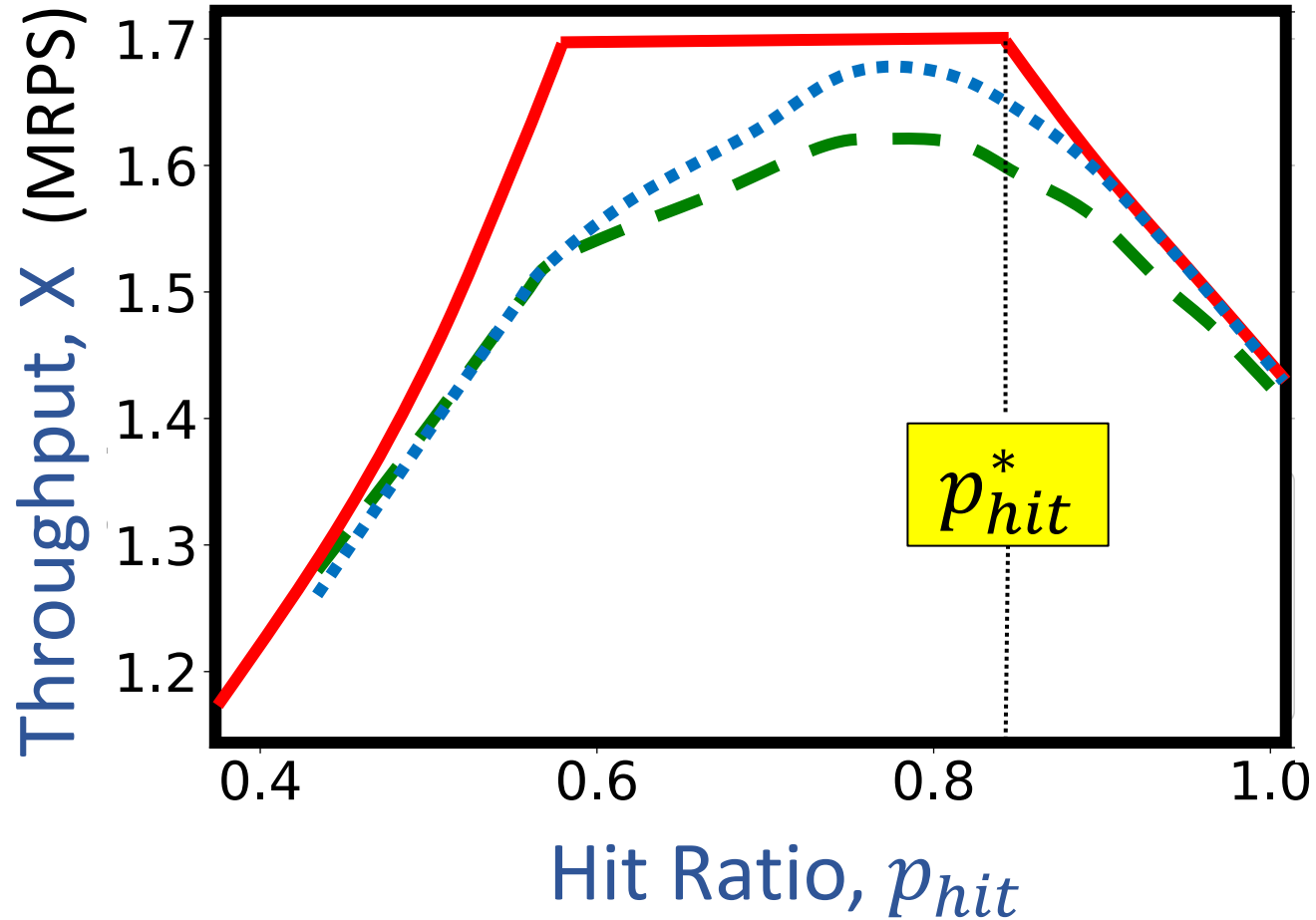
❖ $0.59 < p_{\text{hit}} < 0.84$

→ $X = \frac{1}{D_{\text{head}}} = \frac{1}{0.59}$

❖ $p_{\text{hit}} > 0.84$

→ $X = \frac{1}{D_{\text{delink}}} = \frac{1}{0.7p_{\text{hit}}}$

→ X decreases with p_{hit}



- Queueing network upper bound
- Queueing network simulation
- - Implementation

3 Regimes

❖ $p_{hit} < 0.59$

→ $X = \text{Left term}$

→ X increases with p_{hit}

❖ $0.59 < p_{hit} < 0.84$

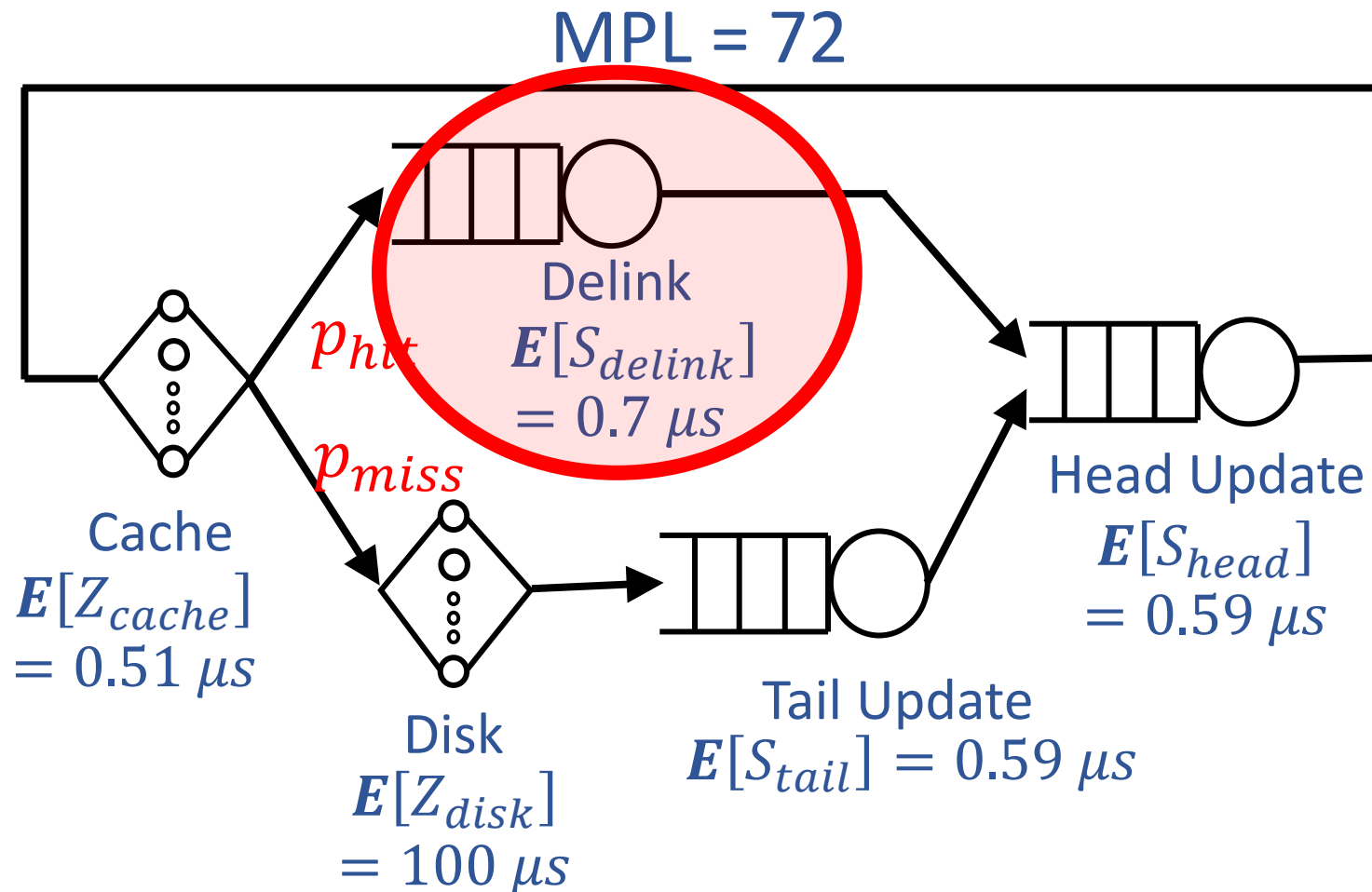
$$\rightarrow X = \frac{1}{D_{head}} = \frac{1}{0.59}$$

❖ $p_{hit} > 0.84$

$$\rightarrow X = \frac{1}{D_{delink}} = \frac{1}{0.7p_{hit}}$$

→ X decreases with p_{hit}

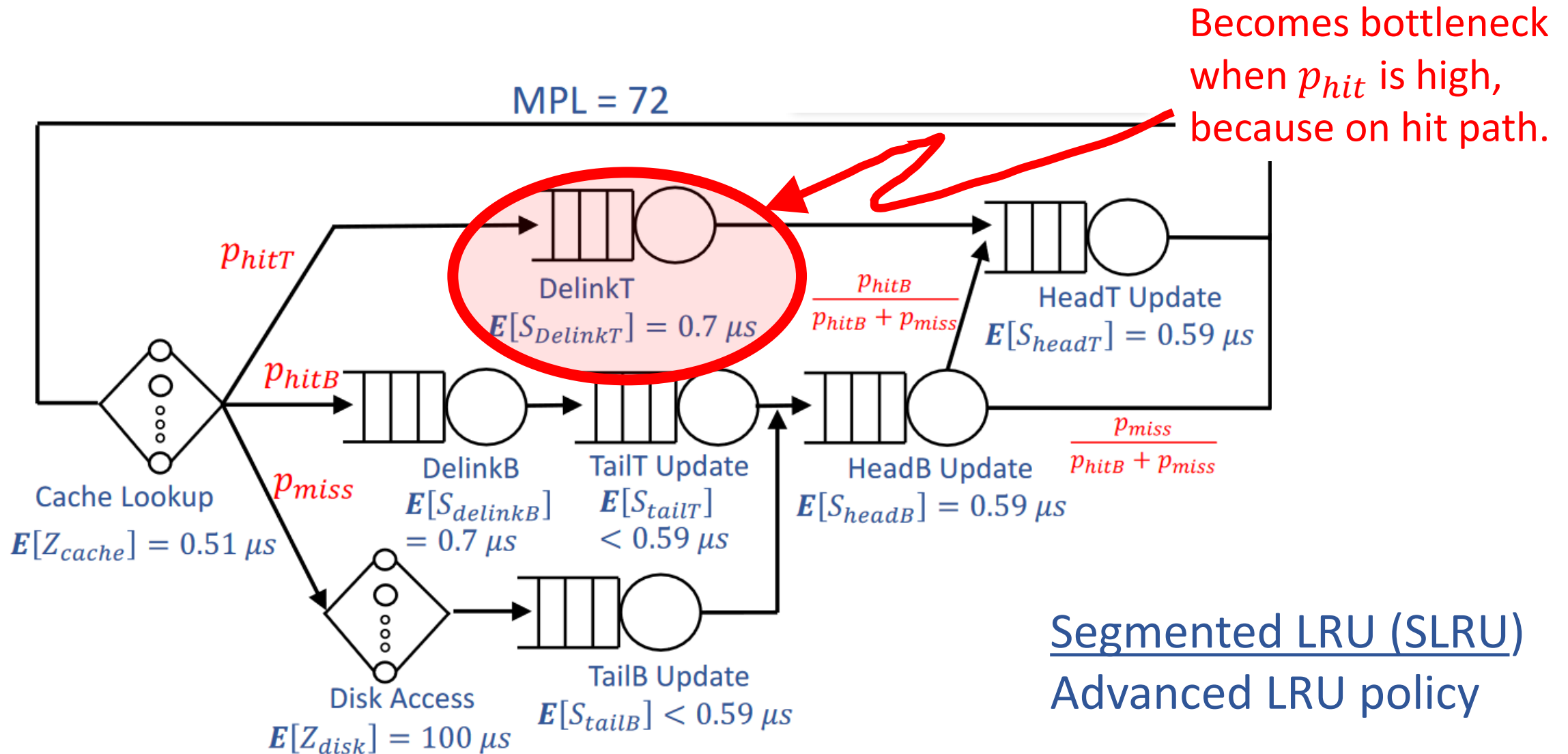
Summary



When p_{hit} is high:

- Delink server becomes bottleneck
- Increasing p_{hit} increases demand on Delink server, making queue even longer
➔ Request latency $\uparrow$
Throughput $\downarrow$

Same story holds for all LRU variants



Future trends

1. Disks will get faster.
2. Concurrency level will increase for both cache and disk.



How do these
affect p_{hit}^* ?

Faster Disk Speed

Higher Concurrency

MPL 16

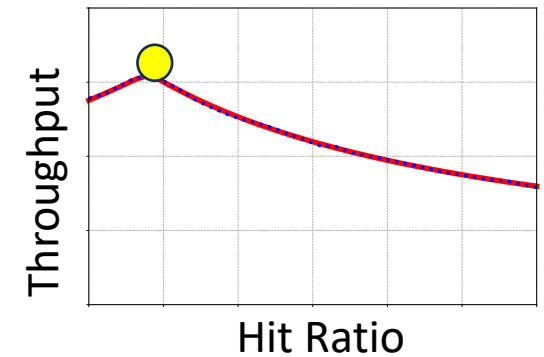
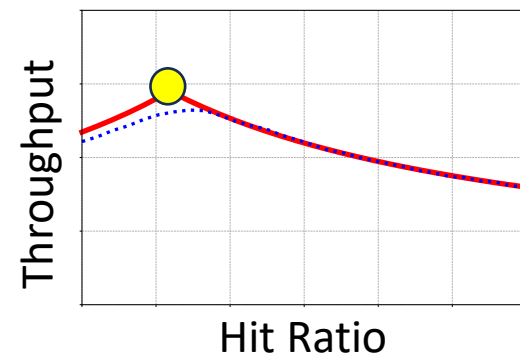
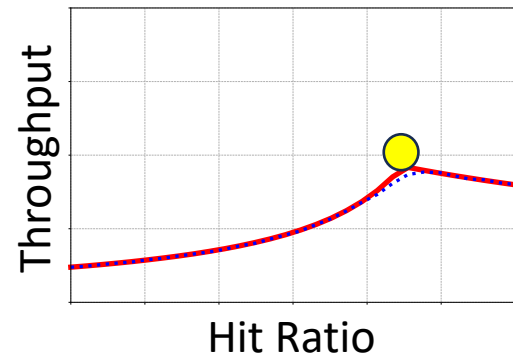
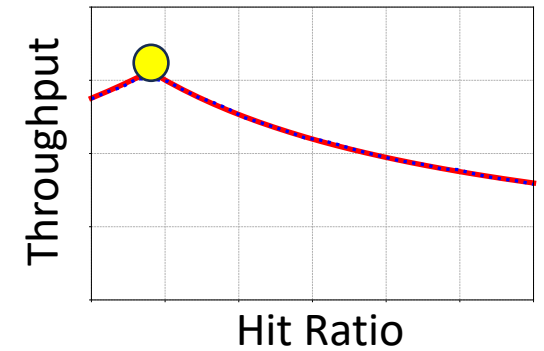
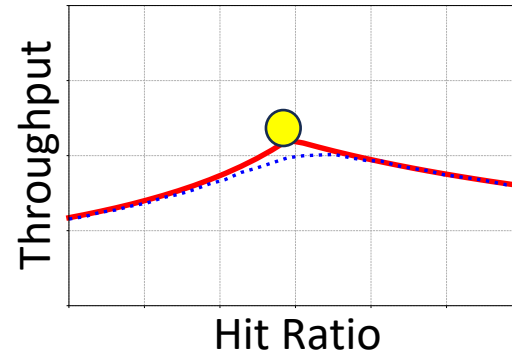
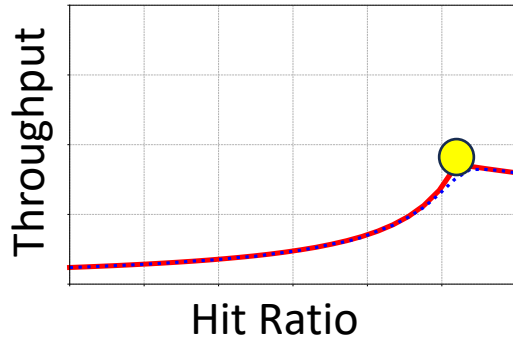
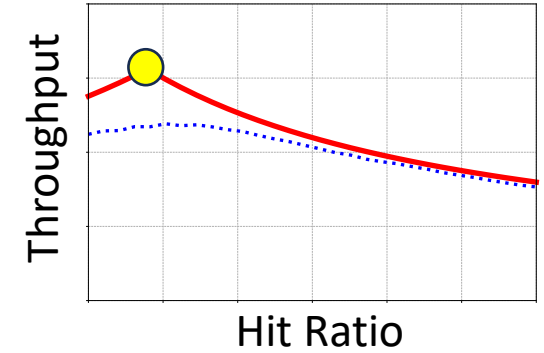
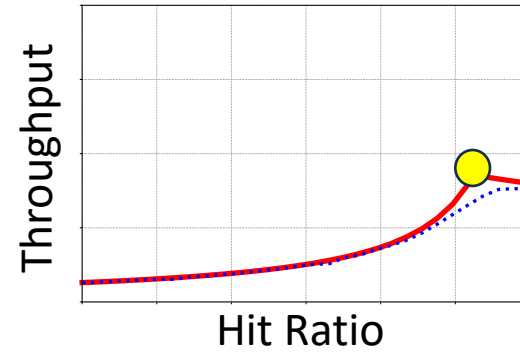
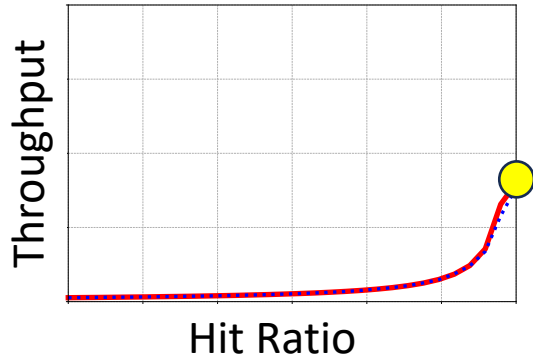
MPL 72

MPL 144

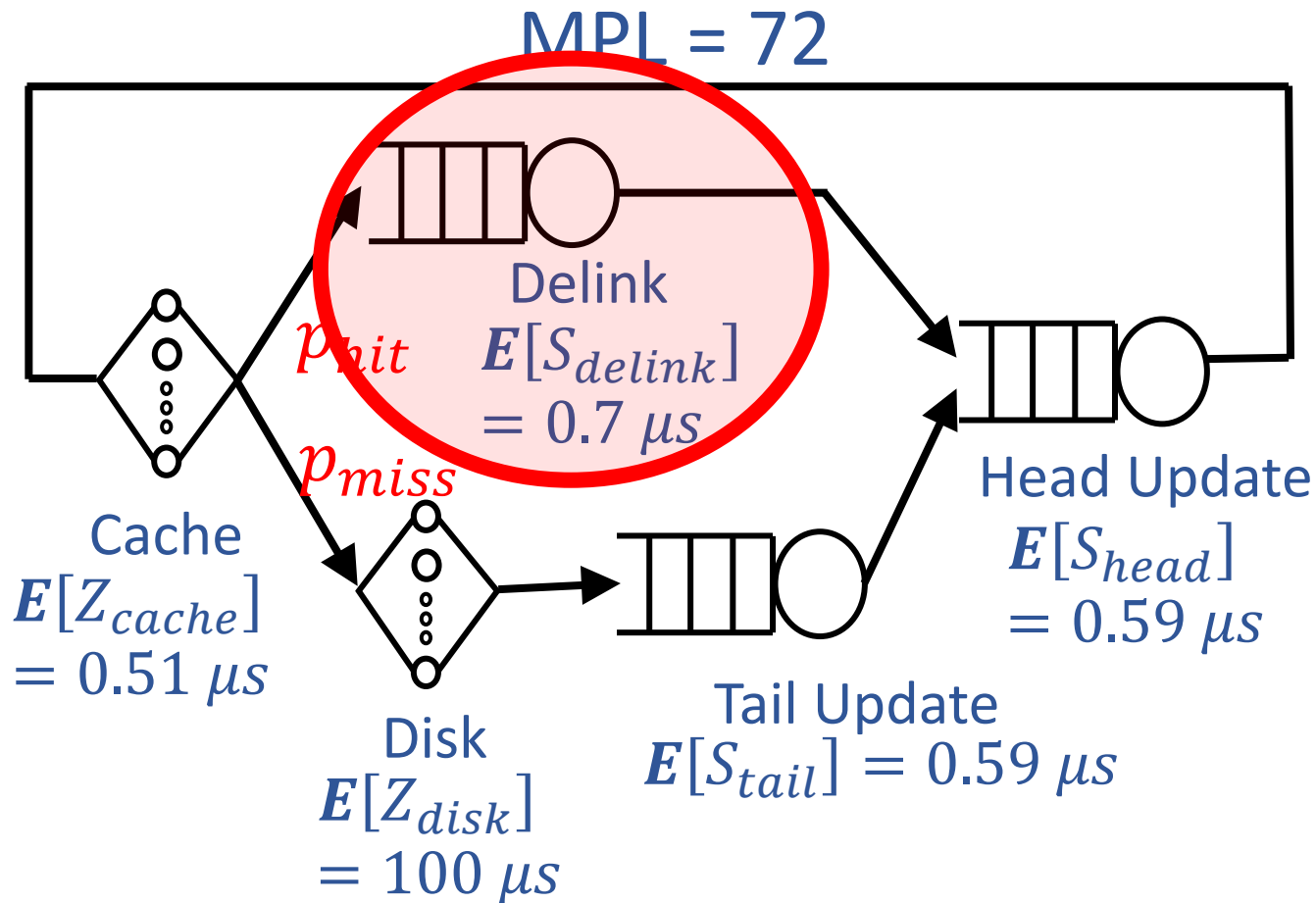
500 μ s

100 μ s

5 μ s



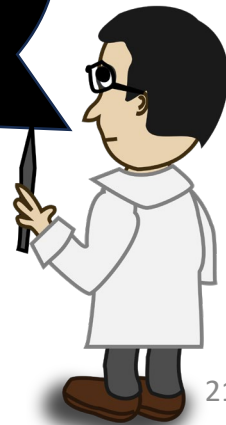
Both trends decrease p_{hit}^*



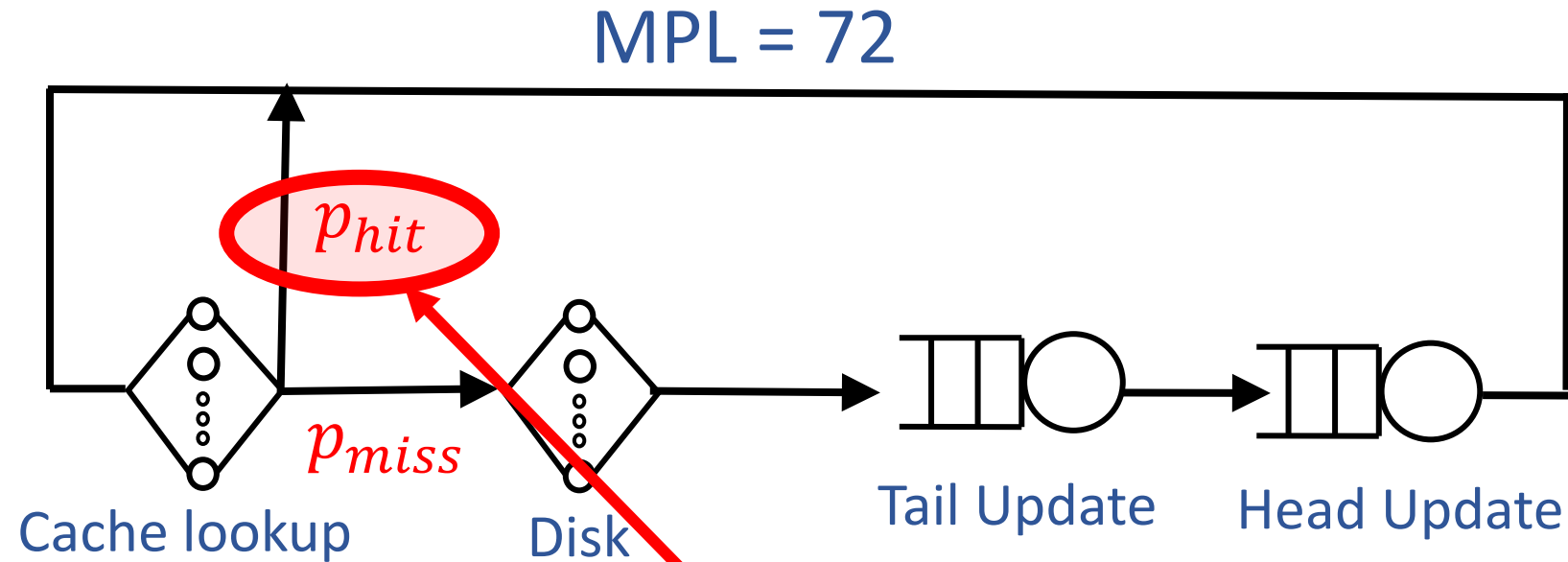
Recall:

$$X \leq \min \left(\frac{MPL}{D + E[Z]} , \frac{1}{D_{max}} \right)$$

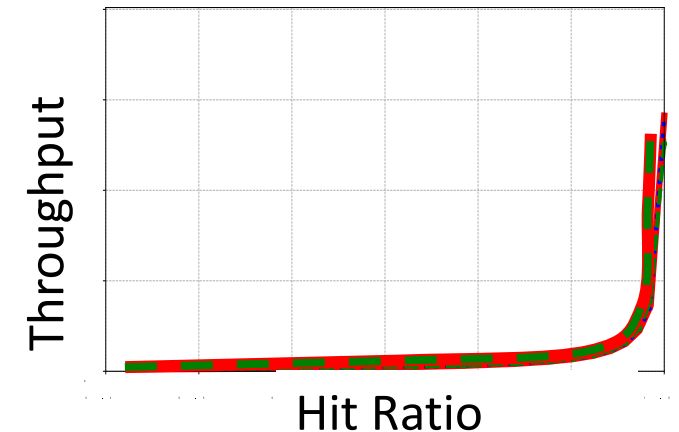
As $MPL \uparrow$ and $E[Z_{disk}] \downarrow$
the D_{max} term
matters sooner.



For FIFO-based caches, X-put only rises



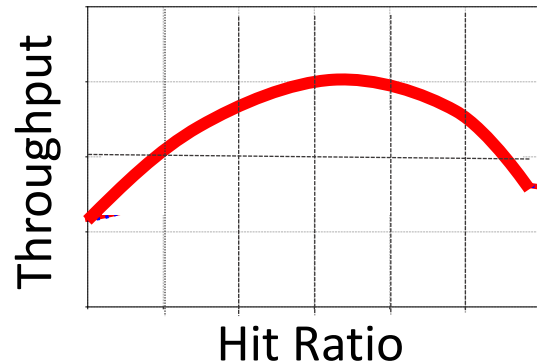
No bottleneck
device on p_{hit} path!



Breakdown of Cache Eviction policies

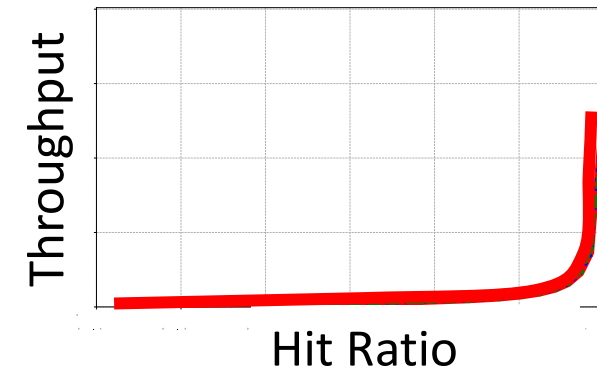
LRU-like behavior

LRU	LeCaR
SLRU	CACHEUS
ARC	
LIRS	
TinyLFU	
LFU	



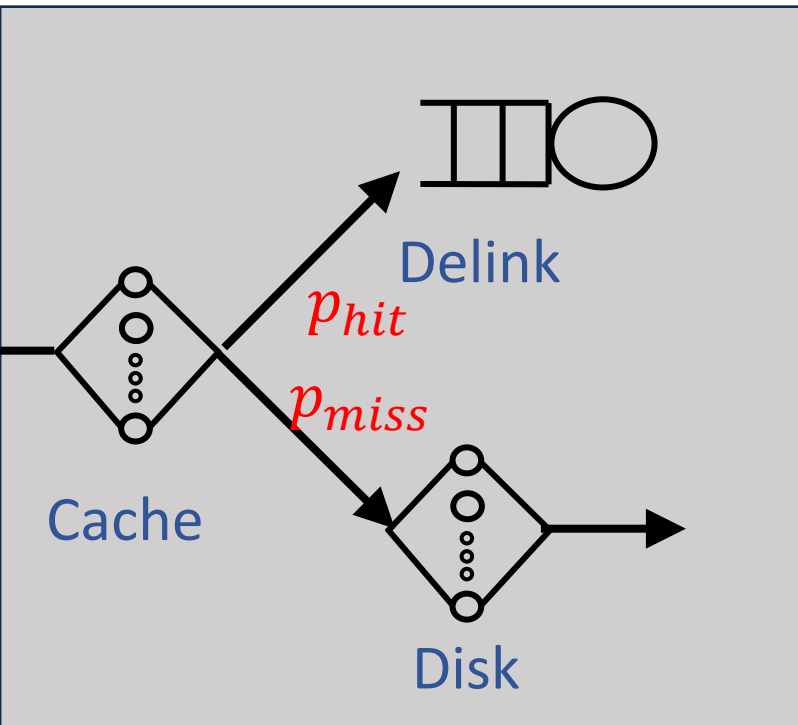
FIFO-like behavior

FIFO	LHD
CLOCK	LRB
S3-FIFO	Random
SIEVE	
QDLP	
Hyperbolic	



Improving future Caching Systems

The problem with LRU:



Q: Why not just forgo LRU altogether & do FIFO?

A: FIFO is less efficient in its use of cache space!

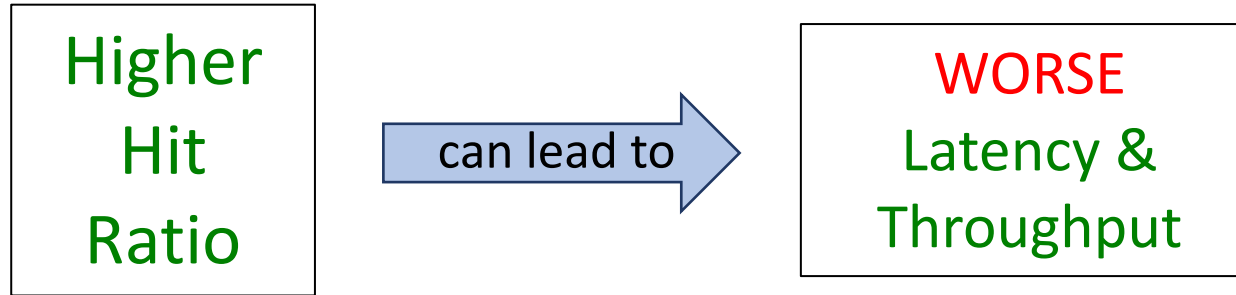
What we really need is some combination of LRU & FIFO!

- Naïve mixture: Probabilistic-LRU
- Better idea:



As p_{hit} gets high, if X-put starts dropping, skip doing Delink step (as in FIFO).

Conclusion



How was this missed?

Faster Disk Speed

Higher Concurrency

MPL 16

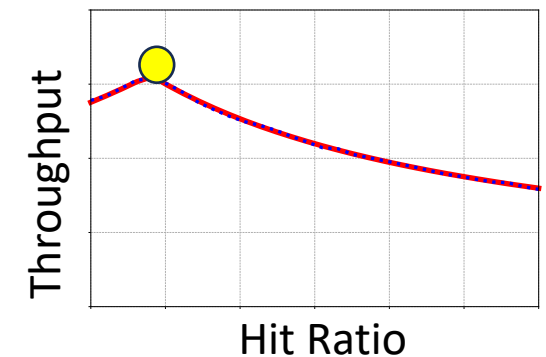
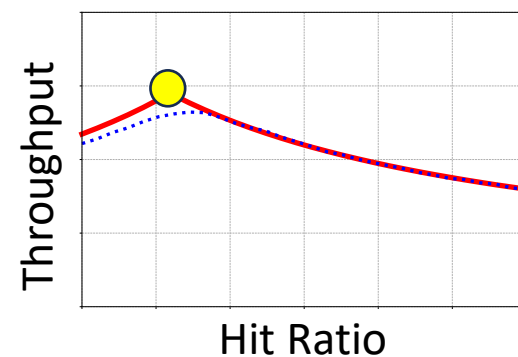
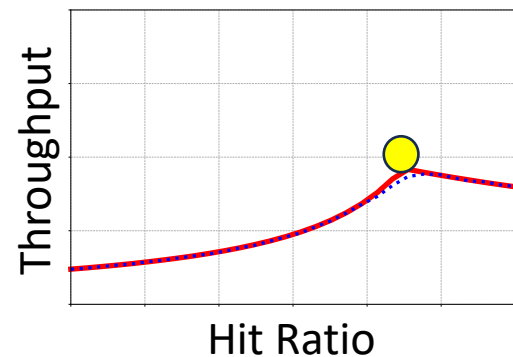
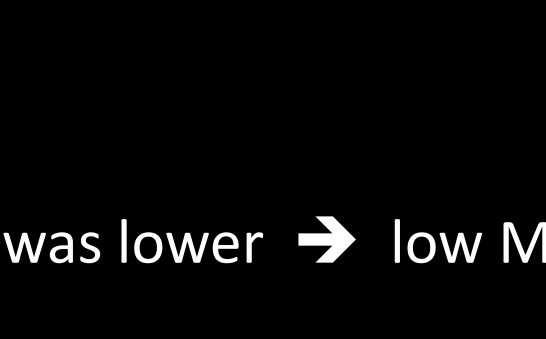
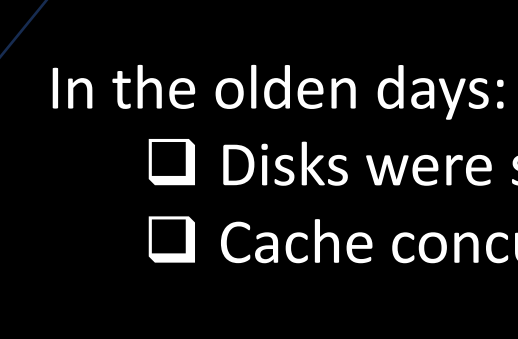
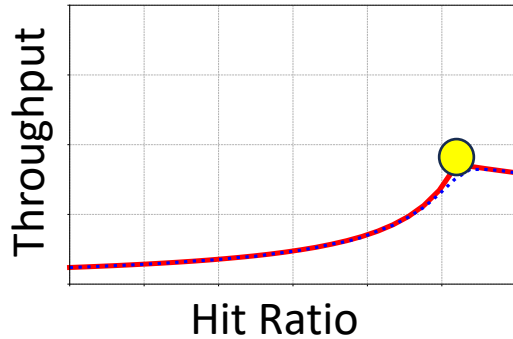
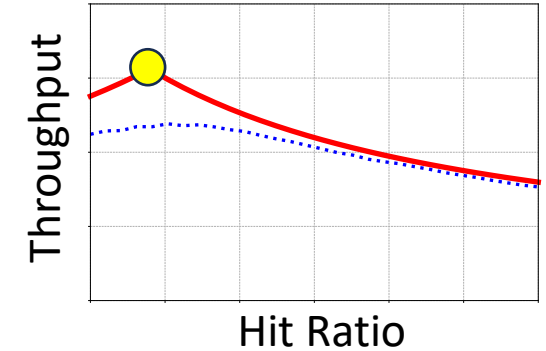
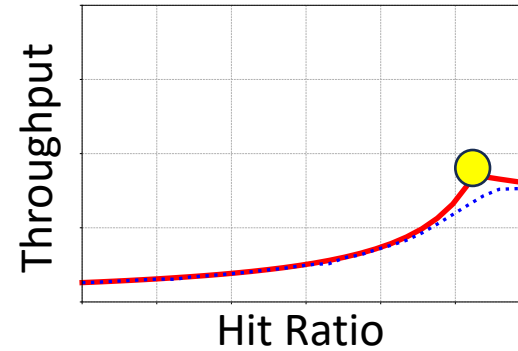
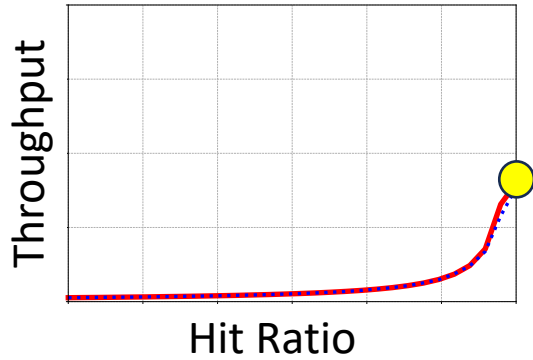
MPL 72

MPL 144

500 μ s

100 μ s

5 μ s

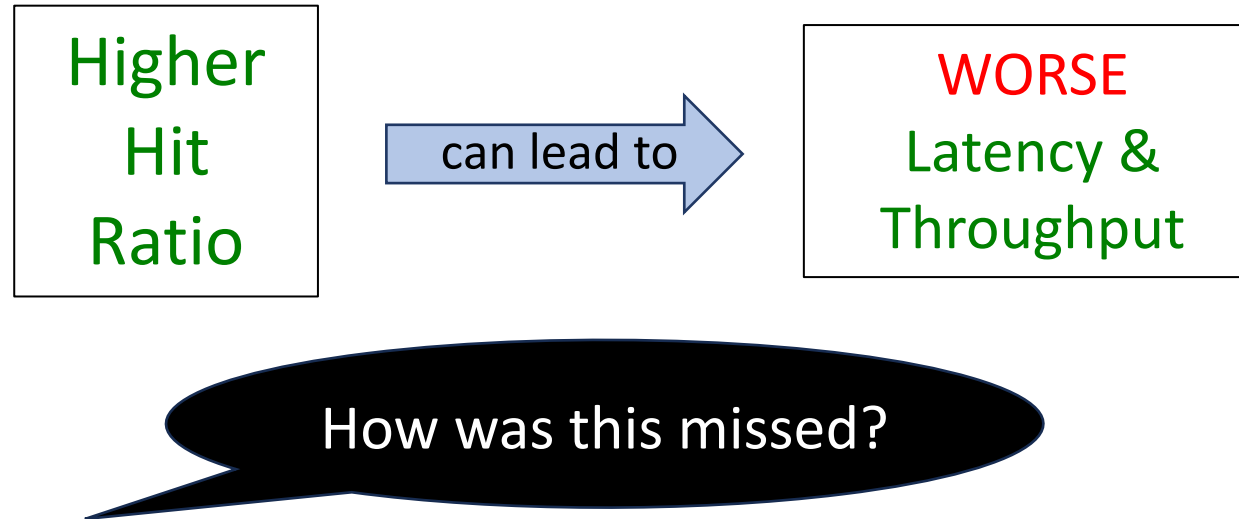


In the olden days:

❑ Disks were slower

❑ Cache concurrency was lower → low MPL

Conclusion



- Olden days: Slower disk + lower MPL → “top left corner”: higher hit ratio helps
- Also in olden days: lower disk concurrency → Queueing at disk → Disk is bottleneck
- **But today** with concurrent disks, bottleneck has shifted to cache operations.
- Operations on the hit path (Delink) become bottleneck when p_{hit} is high.
- When this happens, throughput will drop. One solution: mix LRU & FIFO.