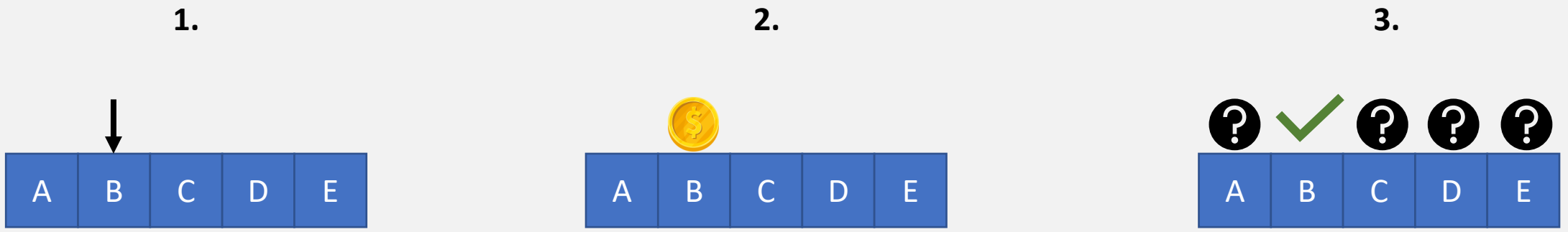
Practical Adversarial Multi-Armed Bandits with Sublinear Runtime

Kasper Overgaard Mortensen · Ama Bembua Bainson · Mathias Ravn Tversted · Kristoffer Strube Græm · Andrea Paudice · Renata Borovica-Gajic · Davide Mottin · Panagiotis Karras

Classical MABs - Simple online learning models

1. **Pick -** choose an arm to pull.
2. **Observe -** see the reward from that arm.
3. **Update -** adjust your estimates to improve future picks.



Problem: Runtime scales with number of arms

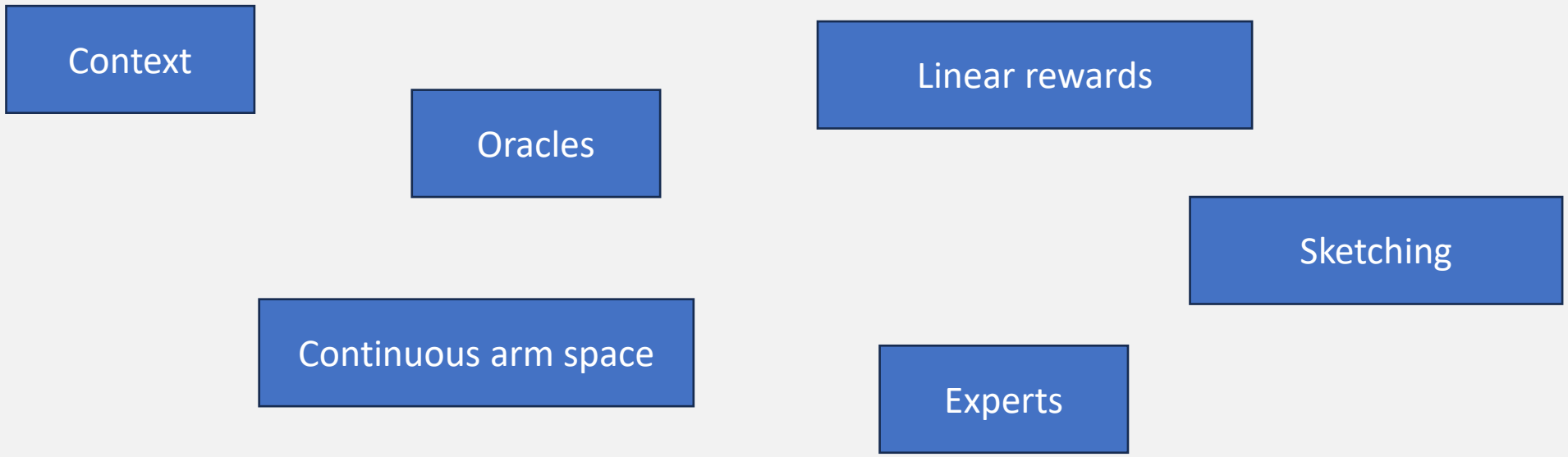
Many arms = Slow iteration runtime

1. Pick
2. Observe
3. Update

Runtime issues:
1) Maintain evolving probability distribution over k arms.
2) Perturb multiple weights every iteration.

The common approach: change setting!

Add new assumptions or utility to avoid the issue.



And many, many more keywords.

Should we care?

Classical bandits rarely work well in real settings anyhow…

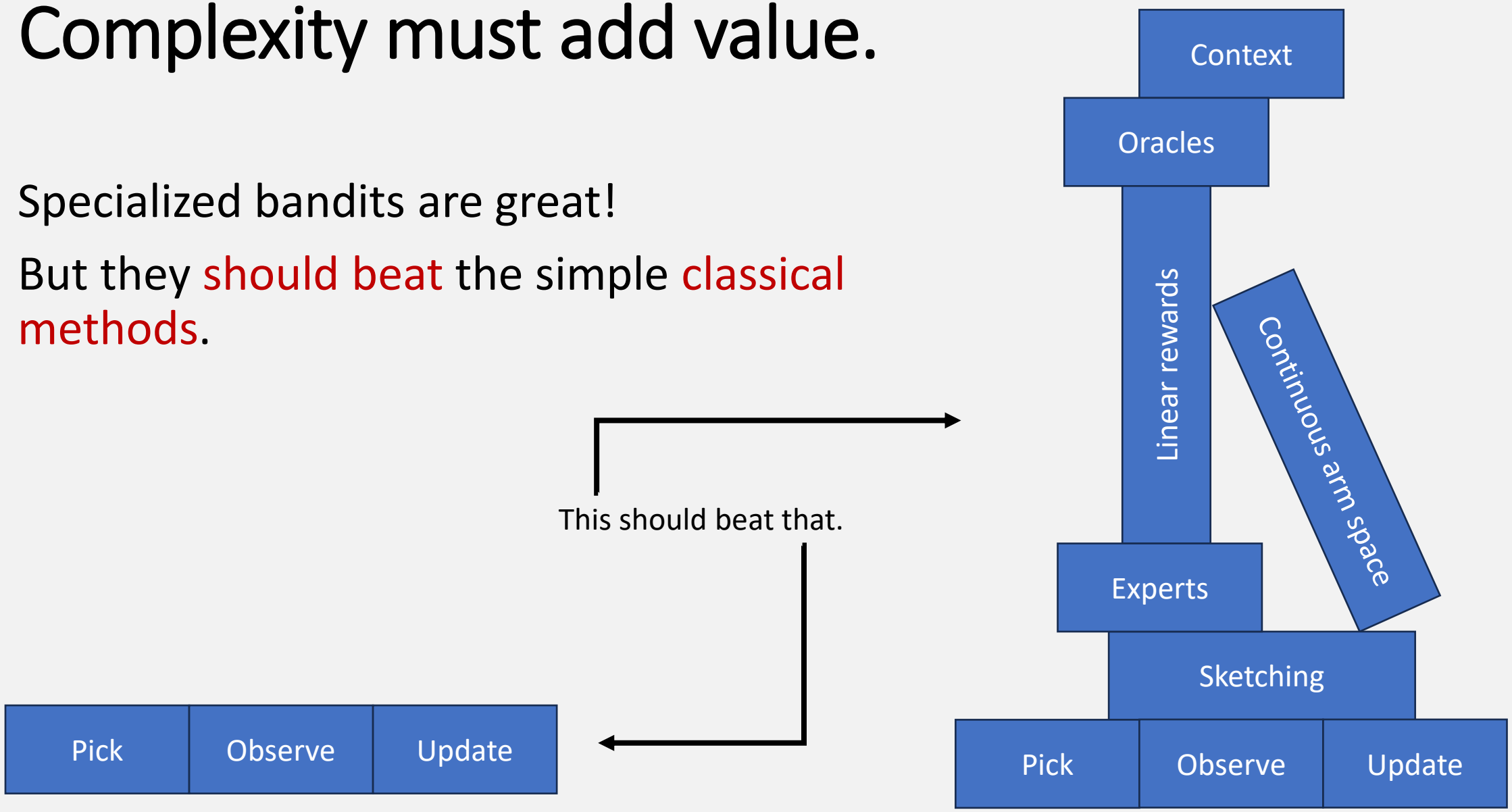
> *I'm having difficulty finding interesting real-world k-Armed Bandit settings […] without altering them to take context into account.*
>
> - John Langford

However, we need them as simple baselines to justify complex alterations.

Complexity must add value.

Specialized bandits are great!

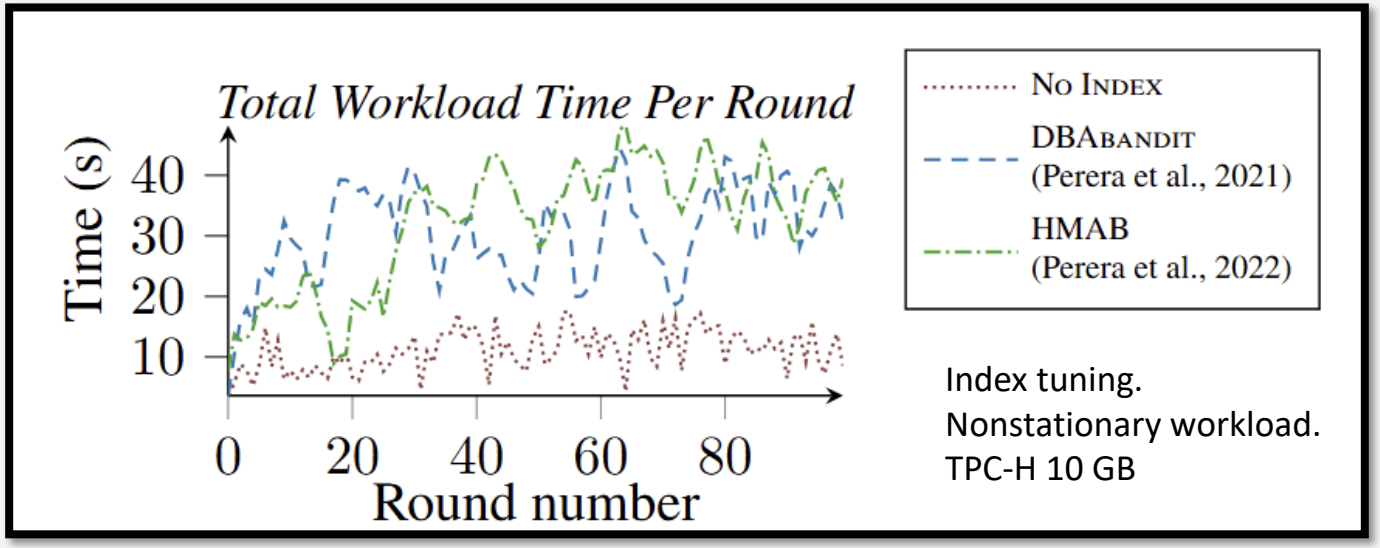
But they should beat the simple classical methods.



System integration - moving beyond intended settings

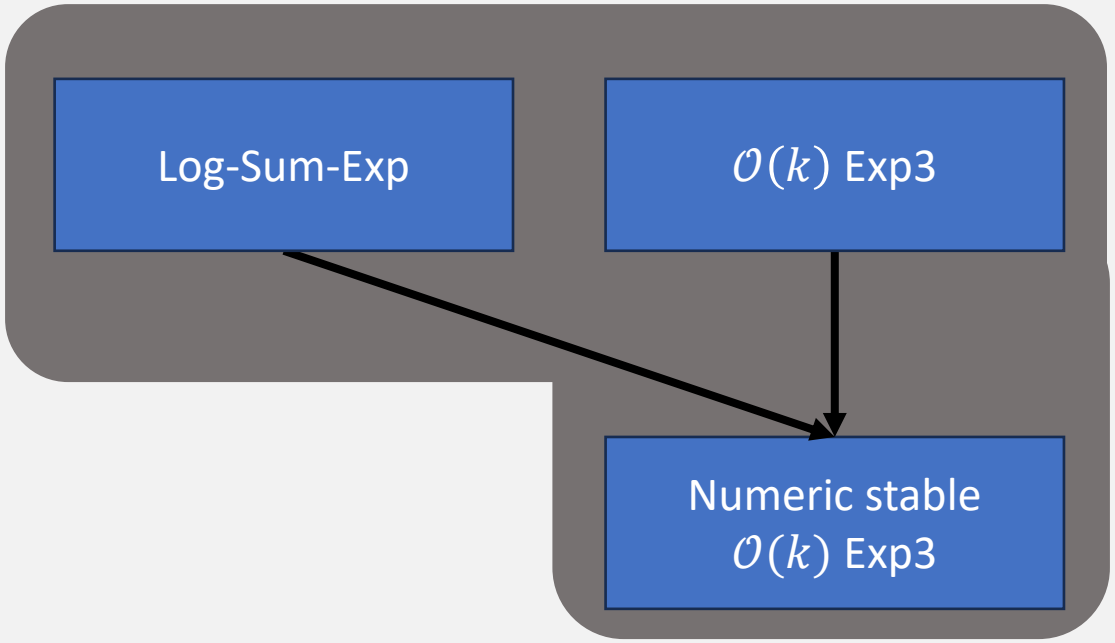
Growing interest for bandits in database systems.

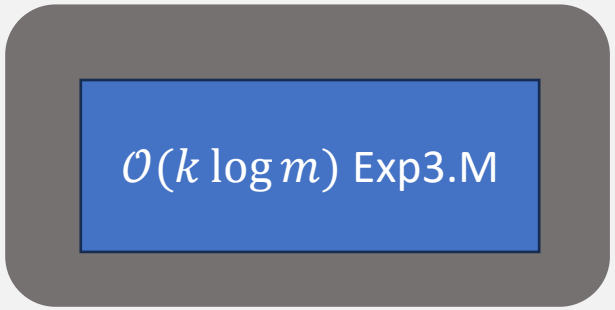
Works well within the targeted assumptions but struggles in extreme cases.



A classic adversarial bandit would be a nice baseline… but they don’t scale.

Exp3 is such a bandit.





Combinatorial setting.
(Select m arms)

But runtime is linear in the number of arms k.

Let’s make a Exp3 scale sublinear to arms!

Solved 10 years ago.

Sum-heap sampling solves the linear overhead

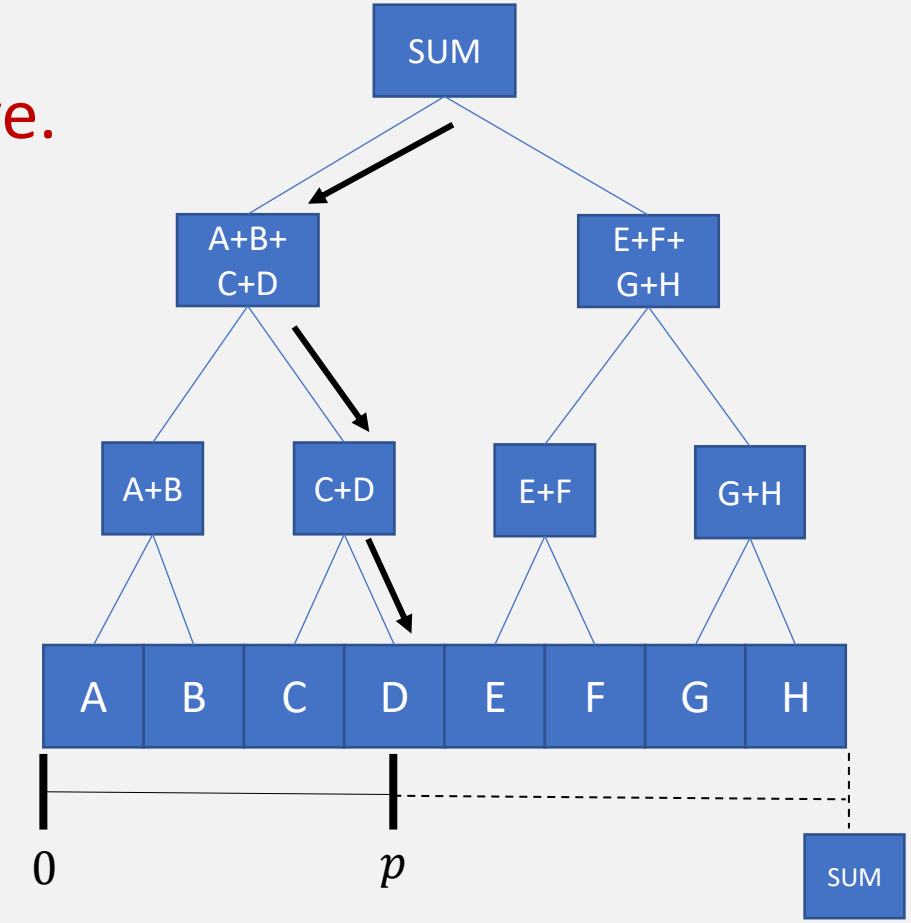
Sample from a distribution that may evolve.

$$p = Uniform(0,1) \cdot \boxed{\text{SUM}}$$

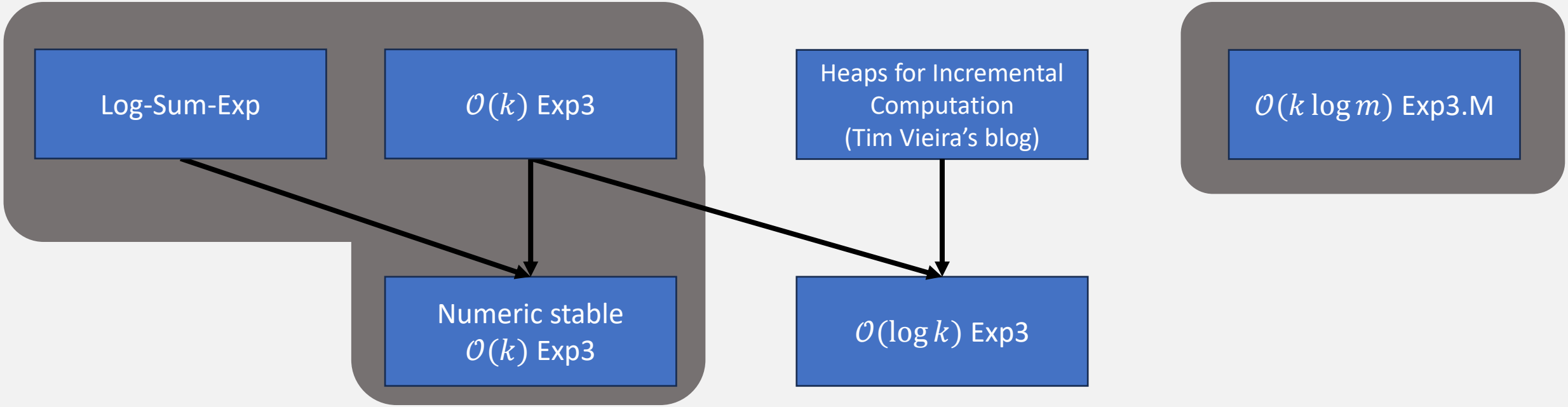
Method	Sample	Update	Init
heap	O(log n)	O(log n)	O(n)

- EXP3 (mutli-armed bandit algorithm) is an excellent example of an algorithm that samples and modifies a single weight in the distribution.

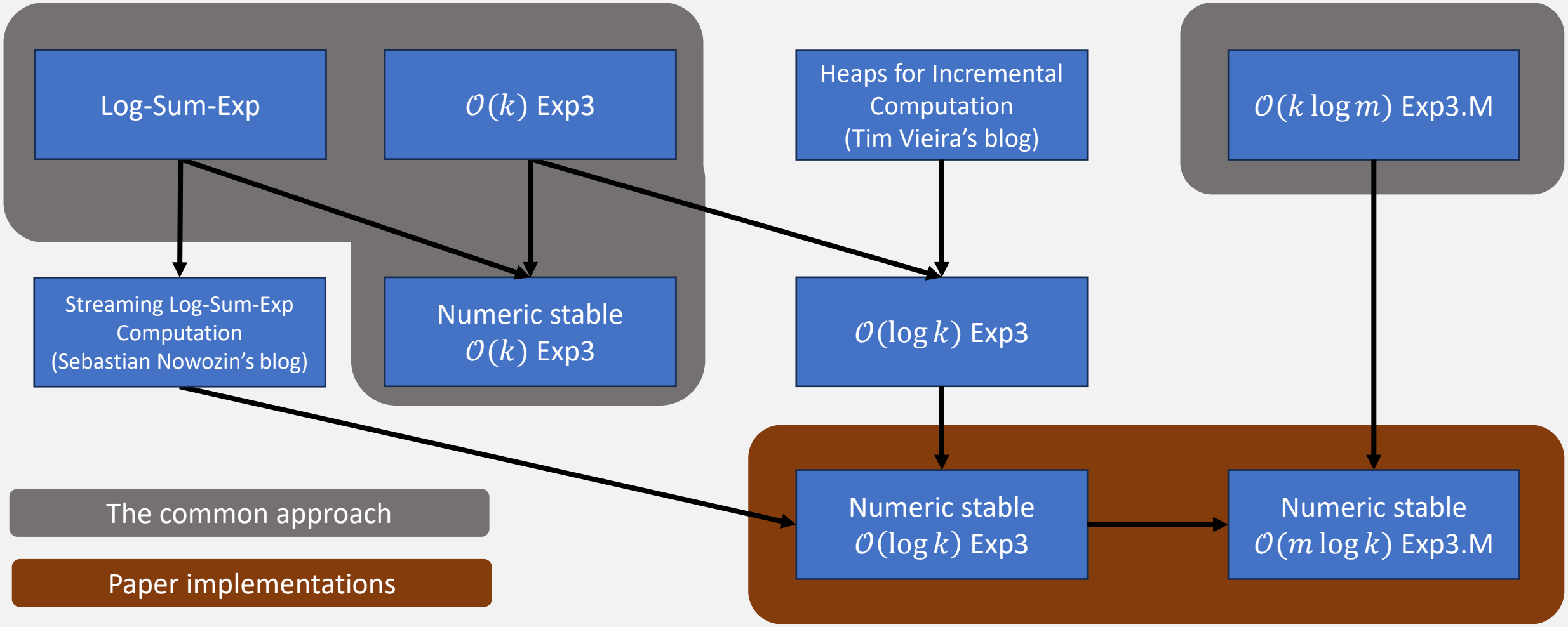
Blog link: https://timvieira.github.io/blog/heaps-for-incremental-computation/



Forgotten folklore?



Covering considerations for large time horizons.



$\mathcal{O}(log\ k)$ and $\mathcal{O}(m\ log\ k)$ is good. *But* can we go faster?

Design for speed.

Design for Speed

Design principle: Best of both worlds.

From Follow-the-leader family: Always pick best perceived option.

From Exp3 family: Update a single weight.

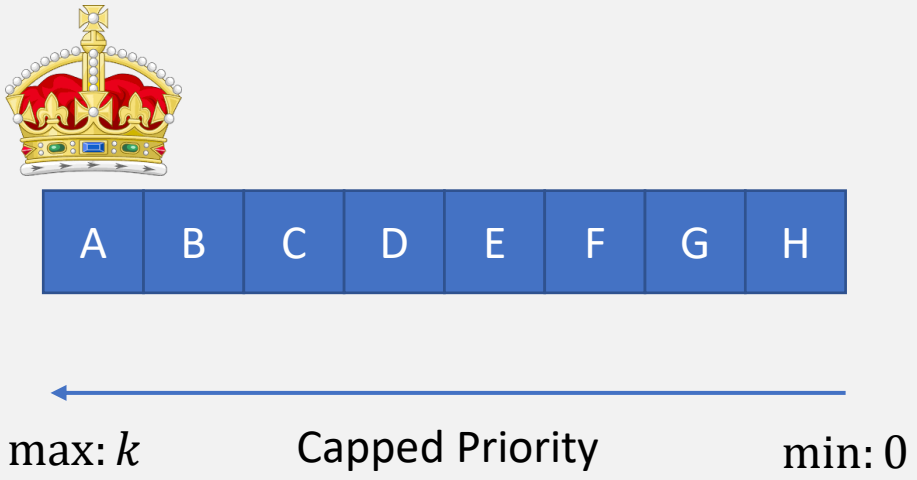
Approach: *Never* look at anything but the best arm.

If deemed good: Keep selecting it.

If deemed bad: Demote it.

Forgo quality guarantees to obtain speed.

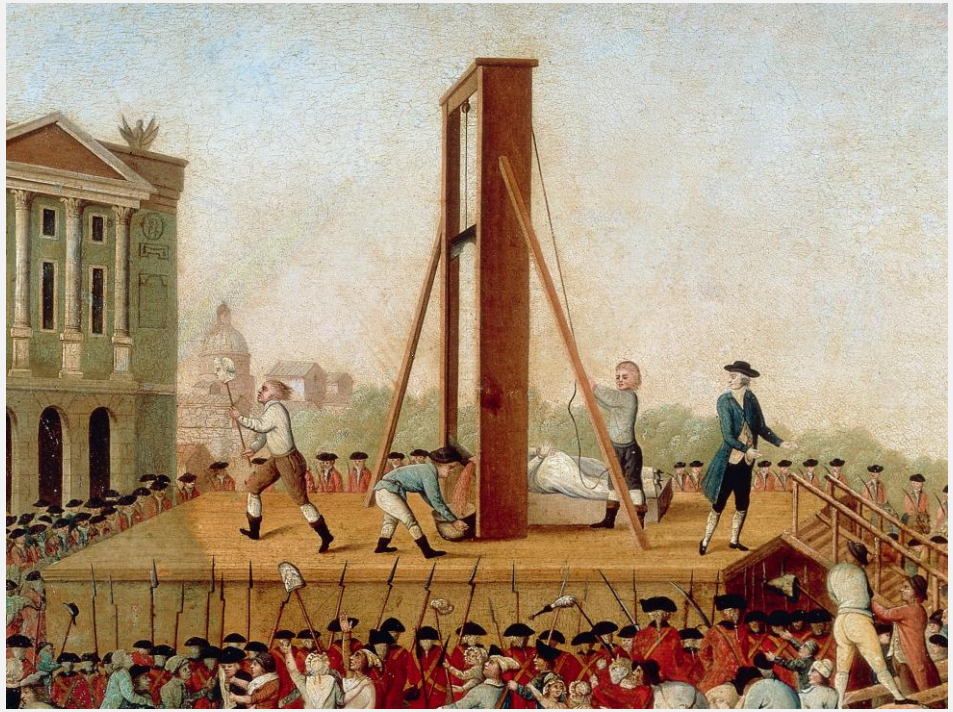
Queue-Behind-the-Leader (QBL)



Demote bad leaders. A two-way pressure.

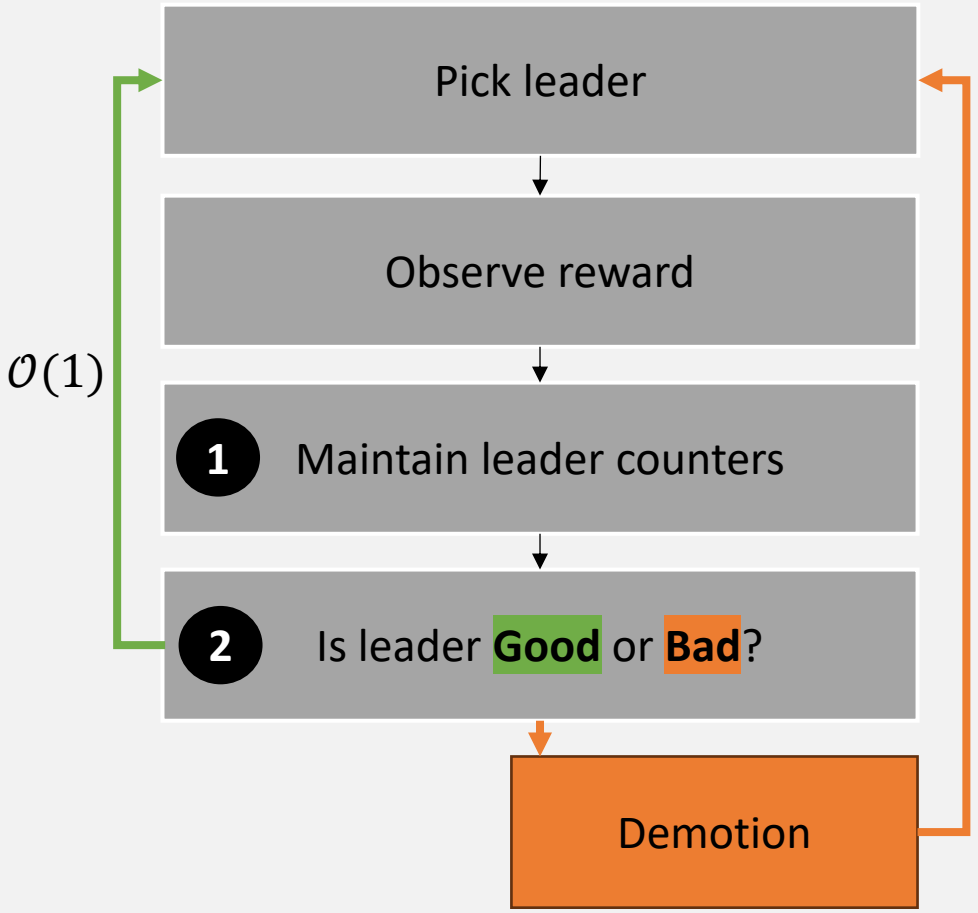
Explicit Demotion

Implicit Promotion



A historical queue of leaders

Queue-Behind-the-Leader (QBL)



1 Constant-time maintenance:

Weighted average performance of all arm

Mean of leader

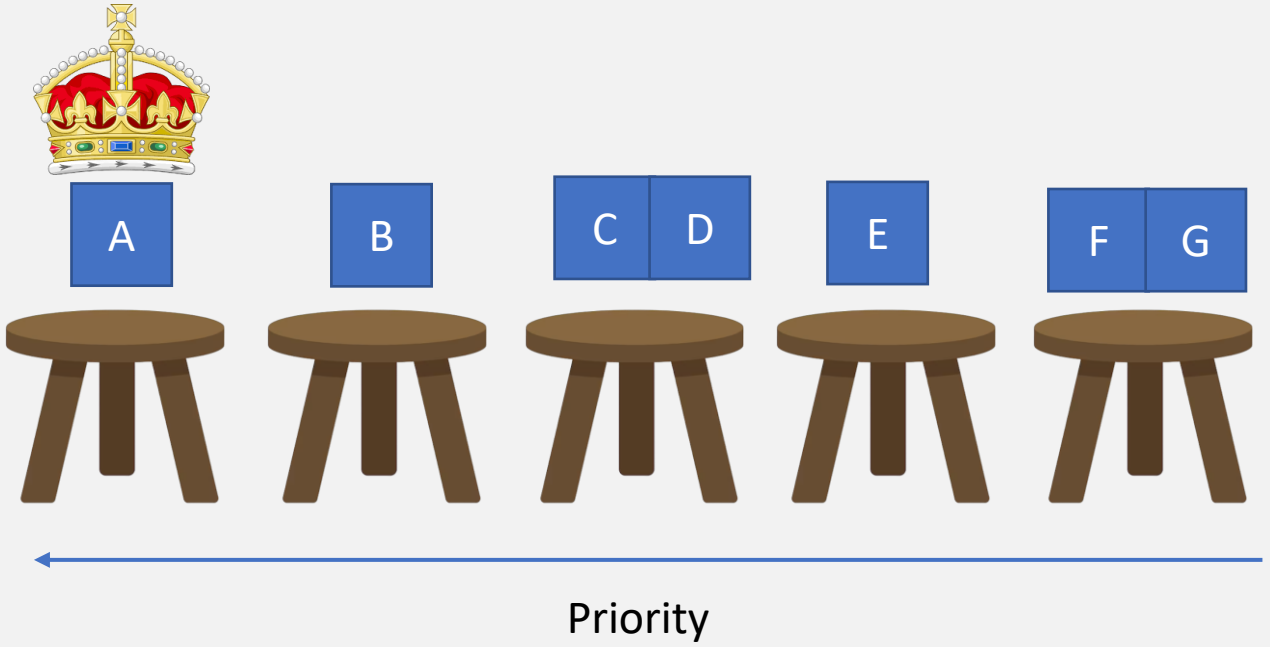
2

Weighted average performance of all arm $>$ Mean of leader → Is leader **Bad**

Weighted average performance of all arm $\approx$ Mean of leader → Leader is suspiciously good

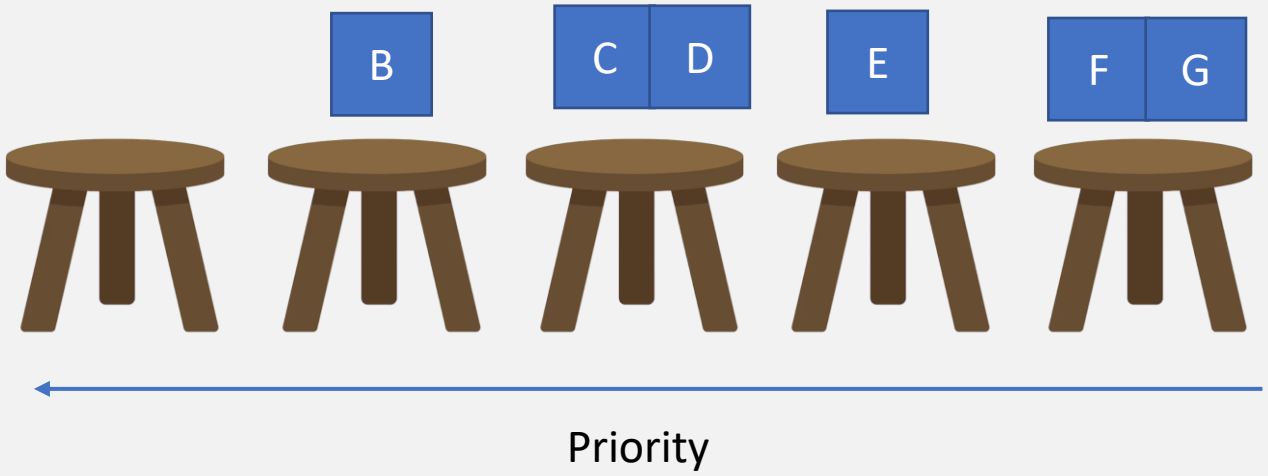
(Randomize decision to break determinism)

QBL – Implicit promotion

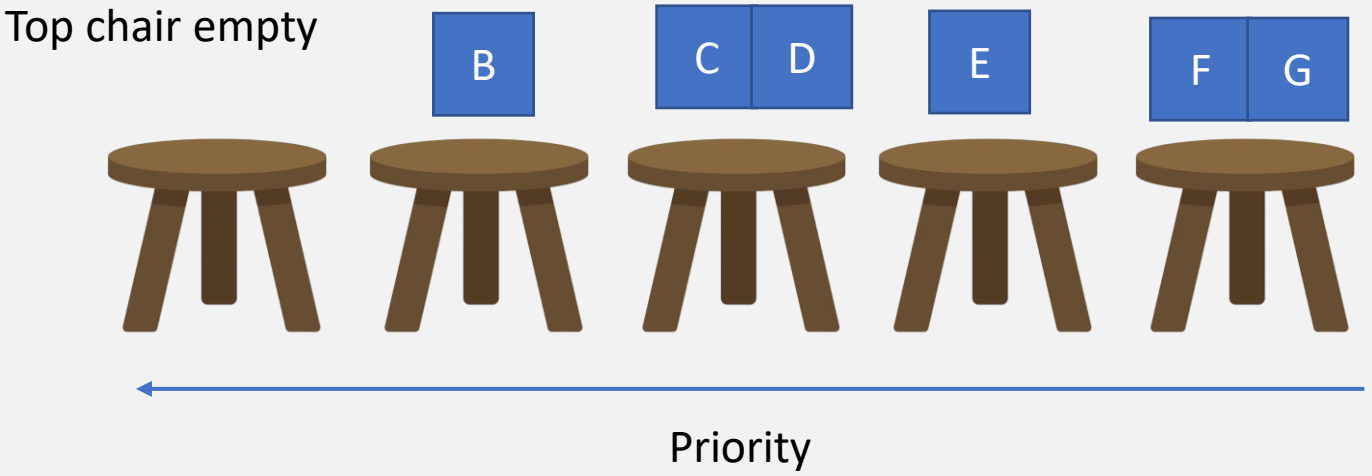


QBL – Implicit promotion

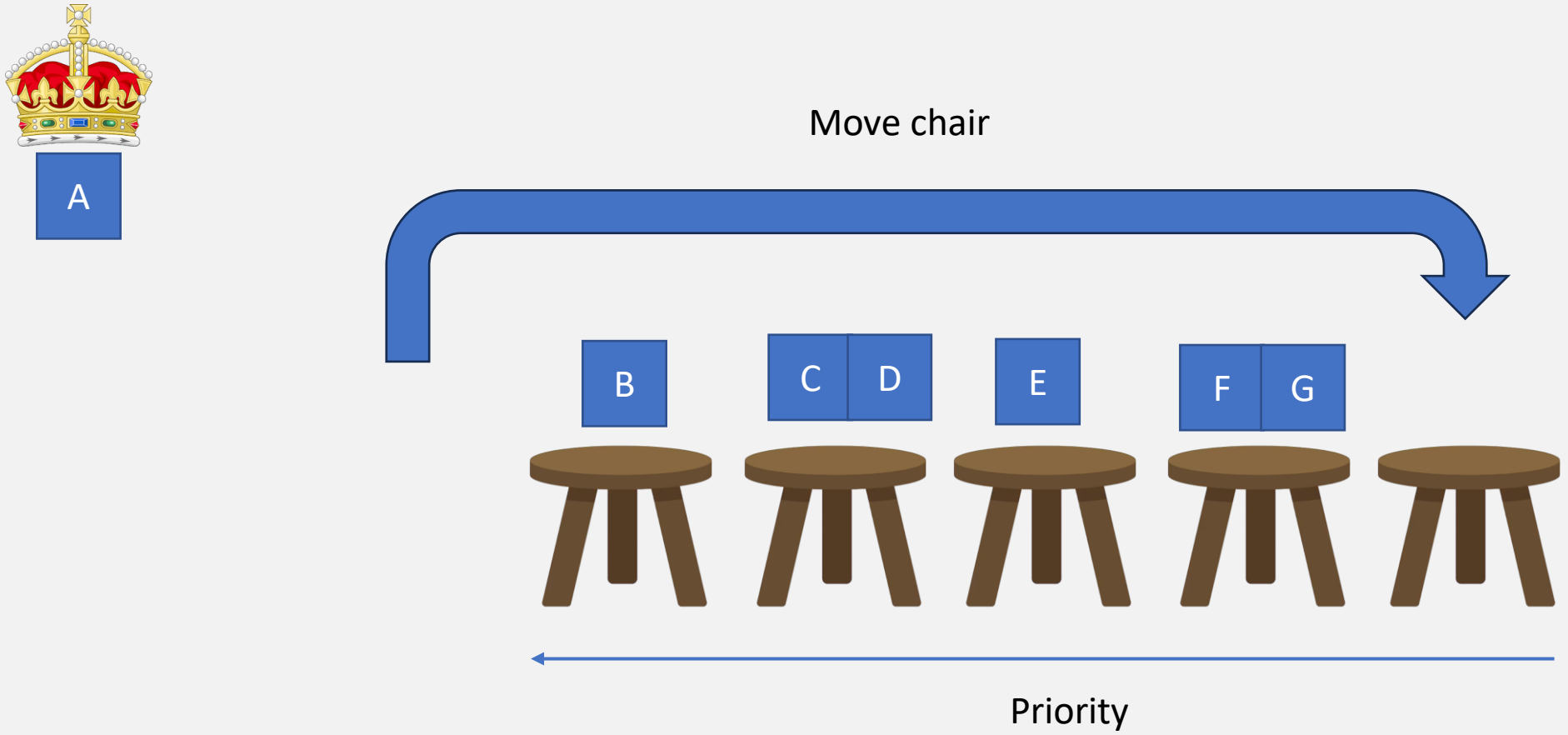
Leader is demoted



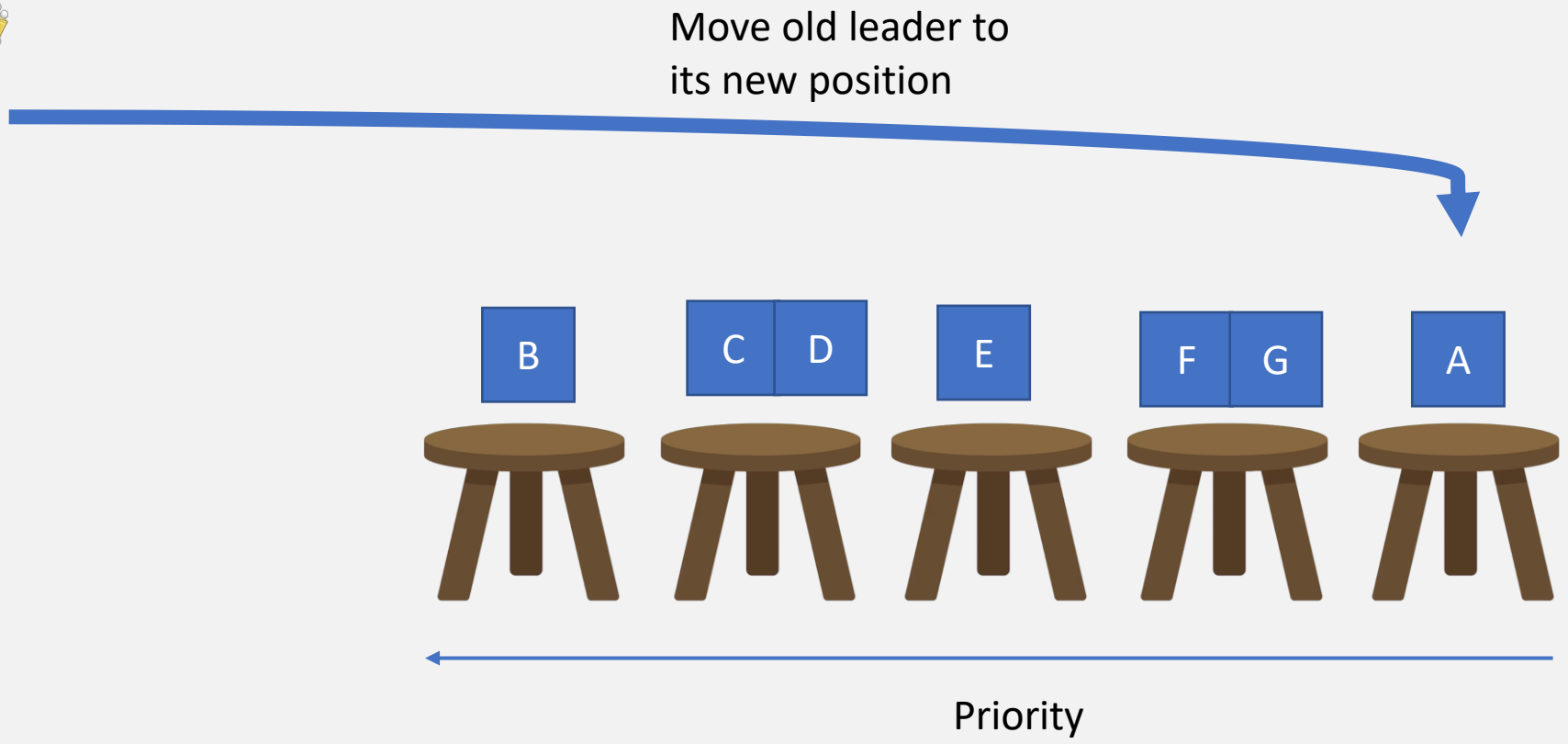
QBL – Implicit promotion



QBL – Implicit promotion

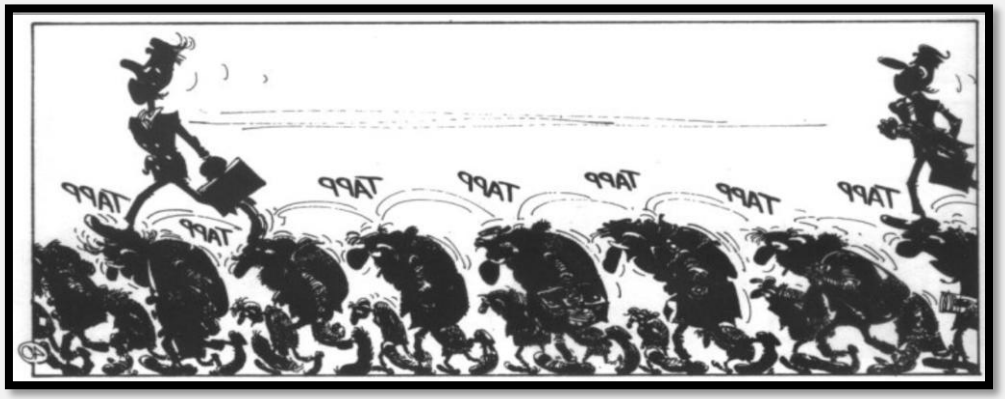


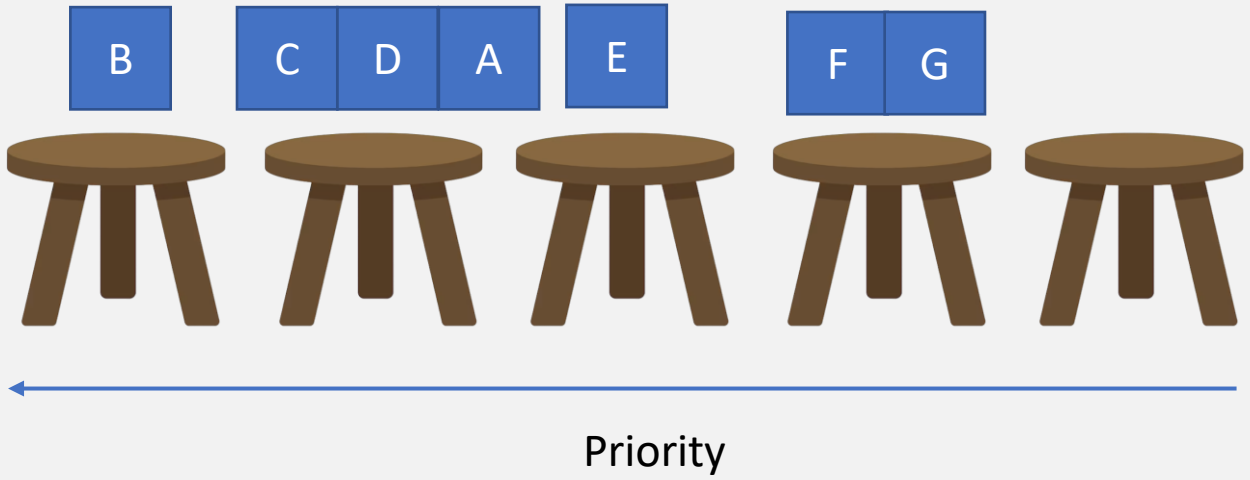
QBL – Implicit promotion



QBL – Implicit promotion

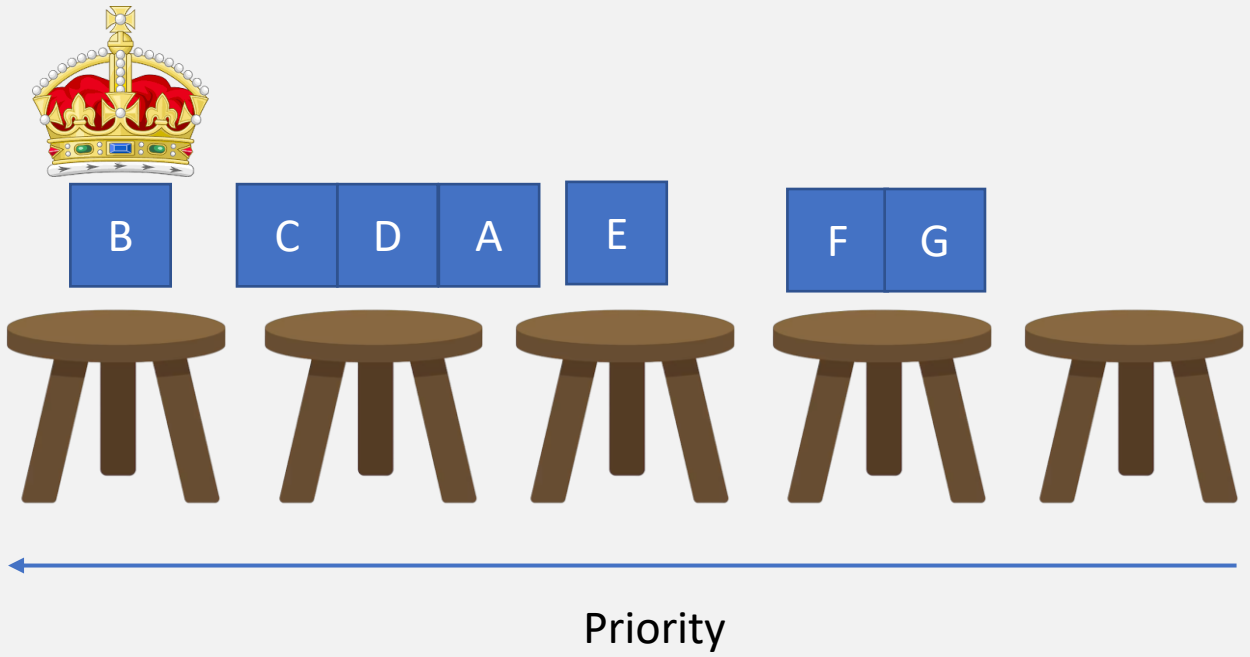
Further up the longer it remained the leader





QBL – Implicit promotion

Elect new leader at
the new top chair



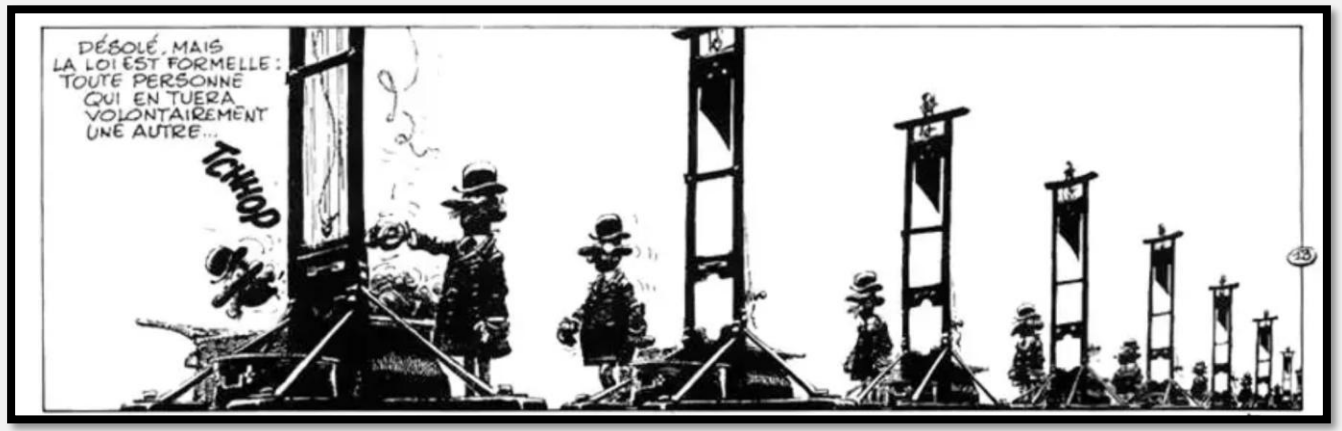
QBL - Overview

Exploration through demotions. Best case skips update phases.

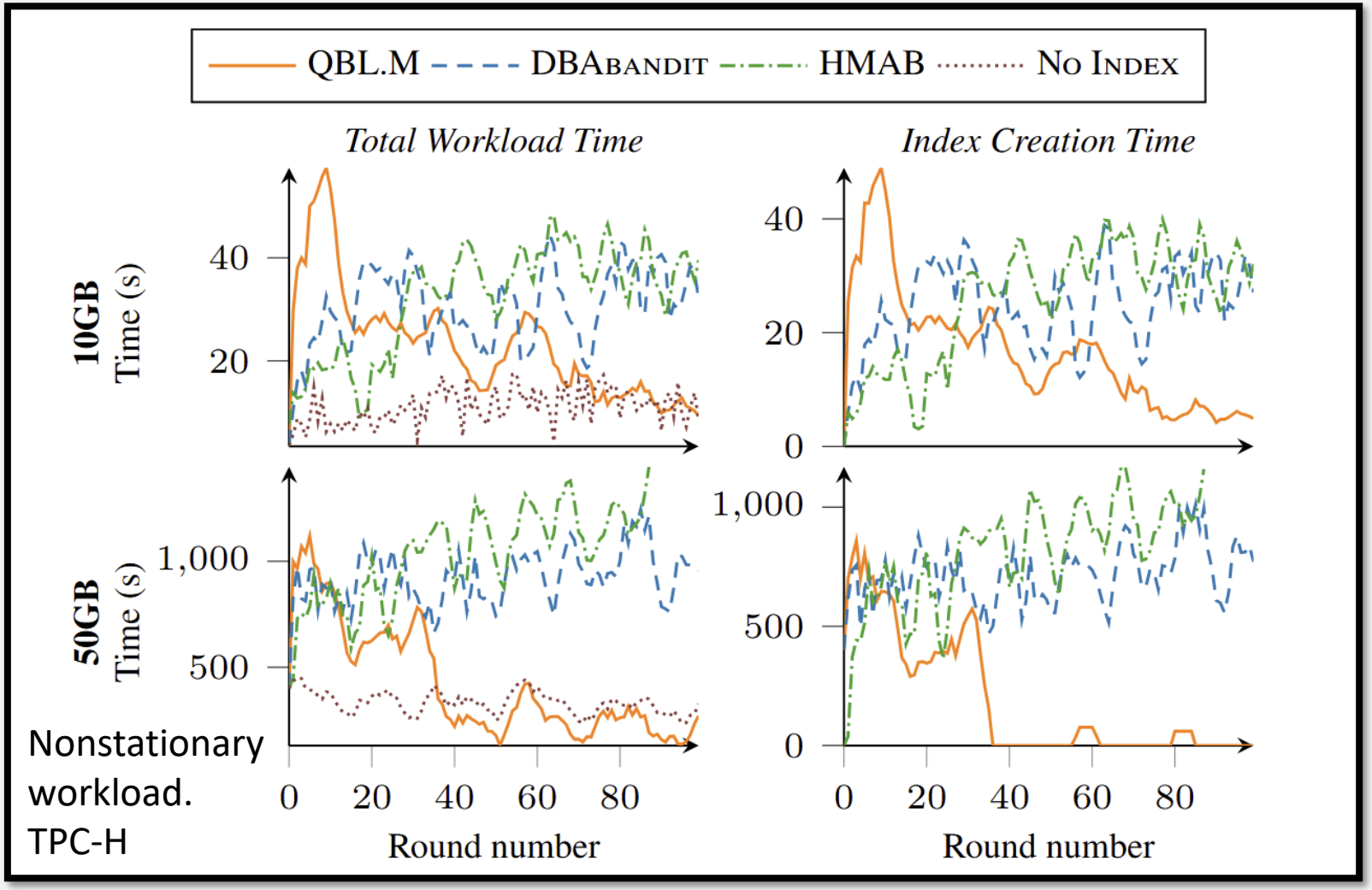
Good leader: $\mathcal{O}(1)$ Demotion: $\mathcal{O}(\log k)$

Combinatorial setting with dynamic m : $\mathcal{O}(m \log k)$

Forgo quality guarantees to achieve fast runtime.

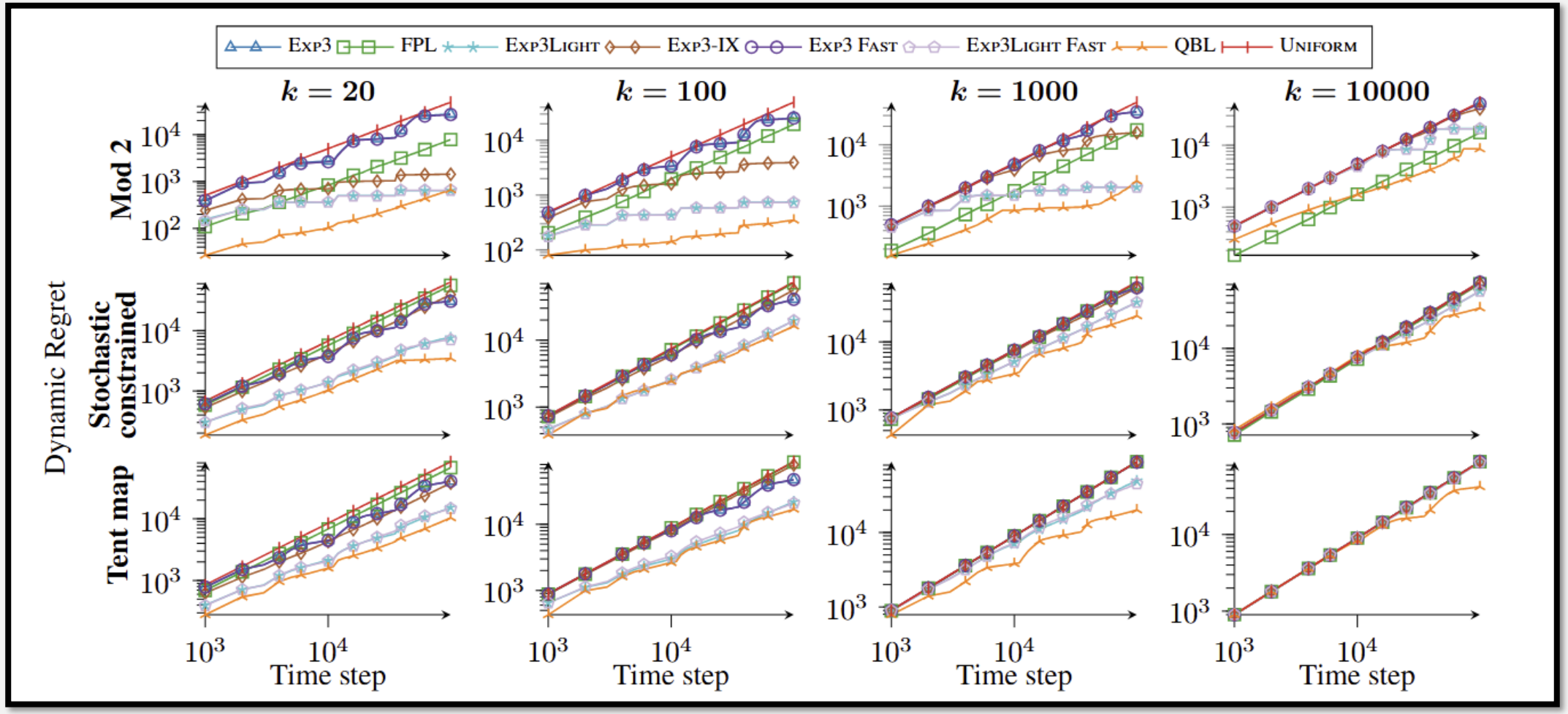


Returning to index tuning in extreme workloads

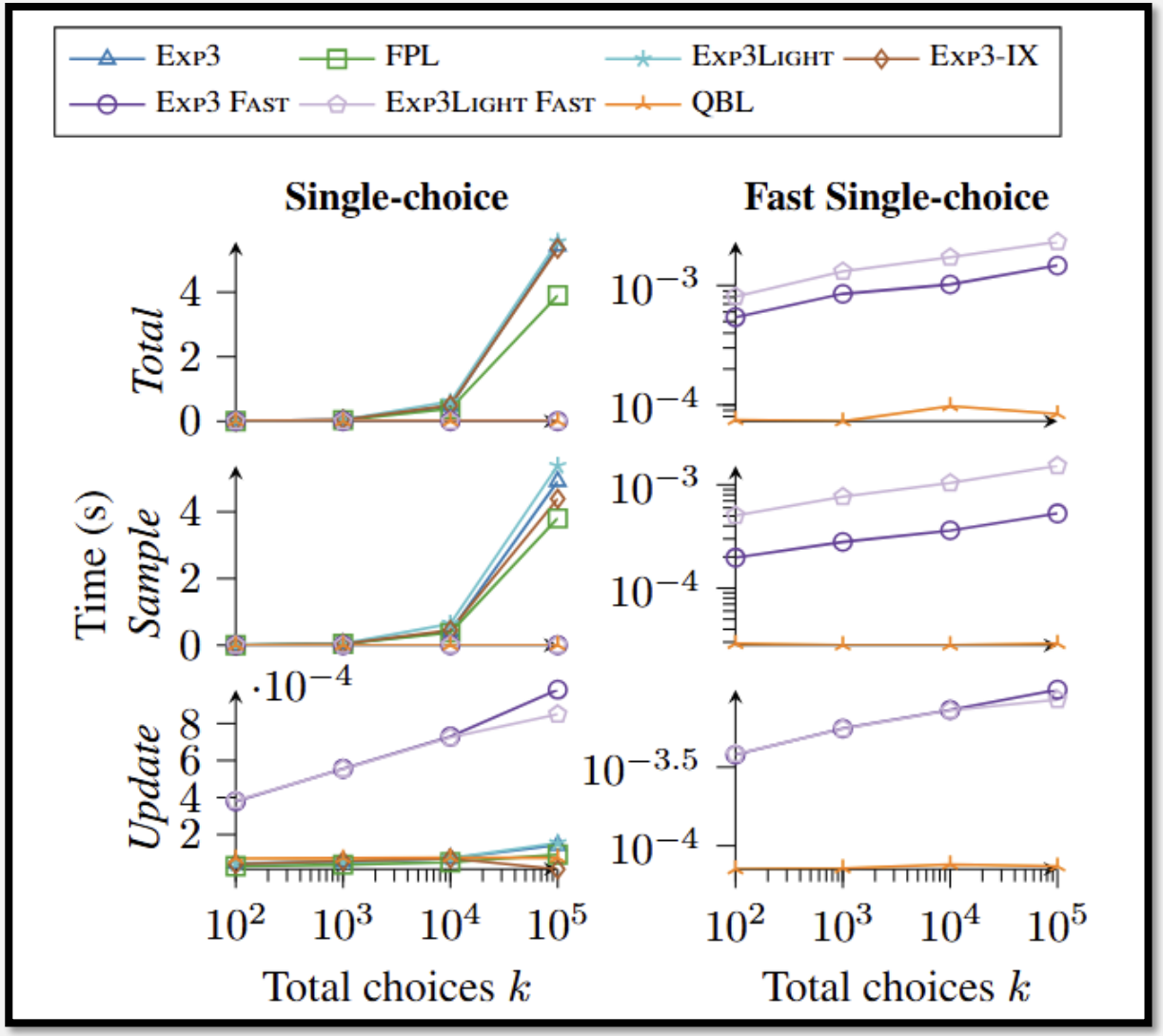


Nonstationary workload. TPC-H

Quality is still promising.



And of course, runtime



Some final thoughts…

Reminder: Scalable and simple baselines are needed.

QBL is designed to be cynical. Worse quality in stable settings.

The hunt for quality guarantees:

Recent work (AdSwitch) utilize similar Good/Bad classification.

Mechanics in QBL can be swapped out.

Thanks!

Checkout Tim Vieira's blog:
https://timvieira.github.io/blog/heaps-for-incremental-computation/

Checkout Sebastian Nowozin's blog:
https://www.nowozin.net/sebastian/blog/streaming-log-sum-exp-computation.html

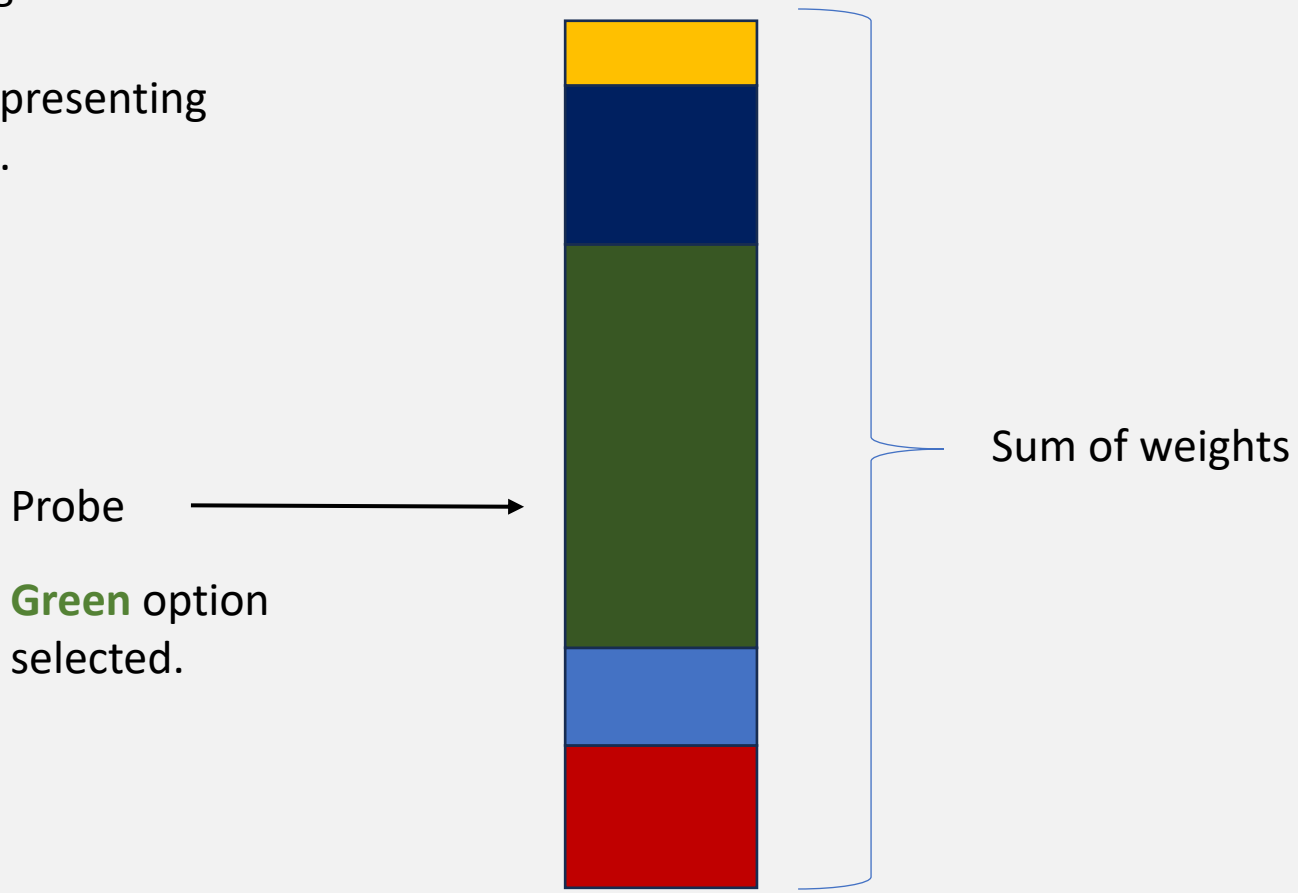
Contact me: km@cs.au.dk

Comics strips from: Idées Noires - André Franquin

Inverse Sampling

1. Uniform probing of sum.

2. Select option representing the probe value.

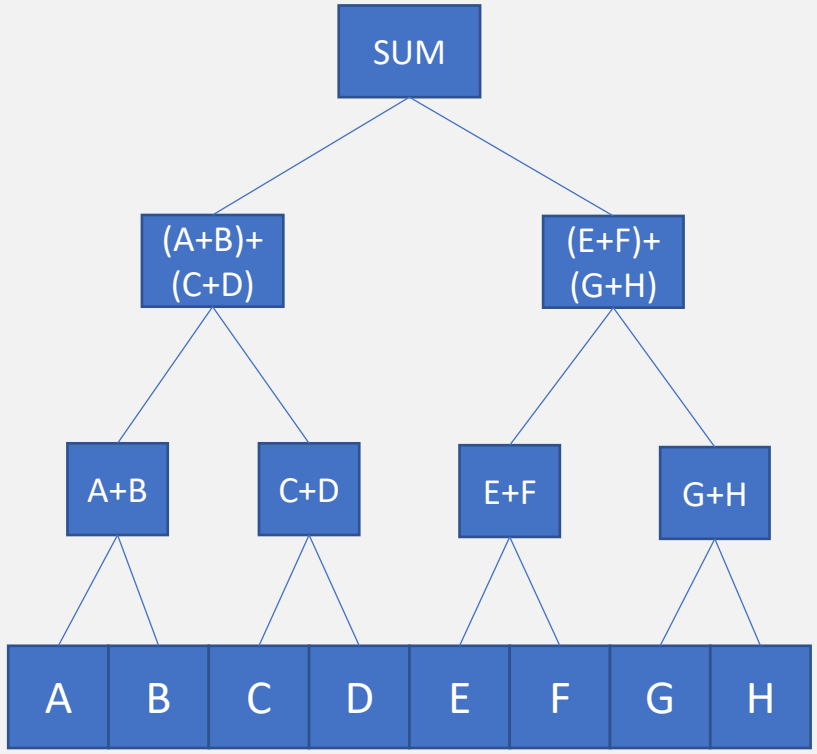


Sum heap – incremental computation

Store weights in a sum heap.

Update weights by fixing intermediate sums. $O(\log k)$.

Costly, so why do it?



Sum heap – Sampling edition

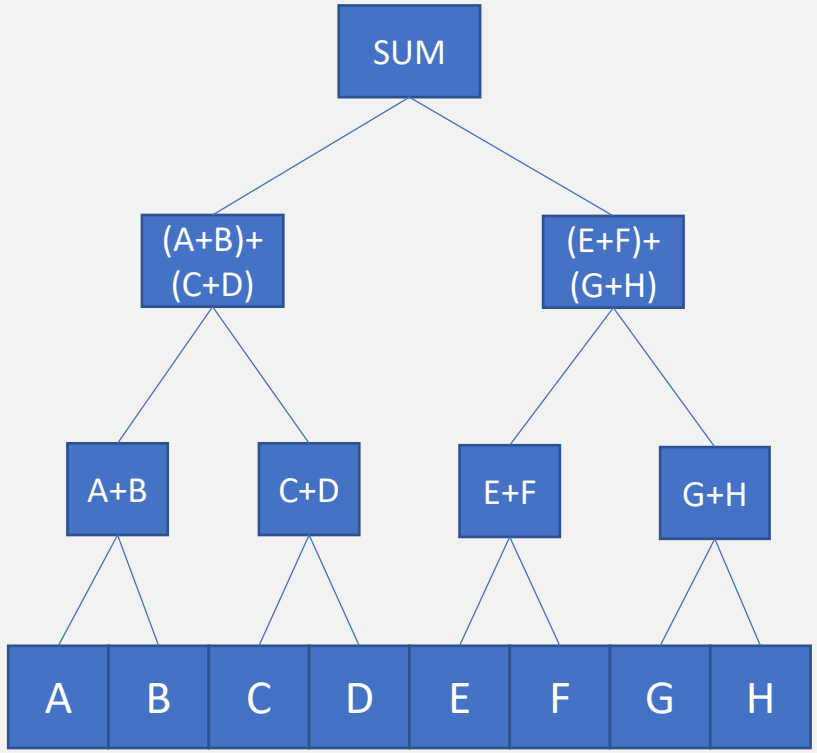
Useable in sampling!

Corresponds to an unnormalized CDF of weights.

Supports that it may evolve.

$$p = Uniform(0,1) \cdot \boxed{\text{SUM}}$$

Find largest CDF value smaller than p. $O(\log k)$.



Integrates nicely with log scale

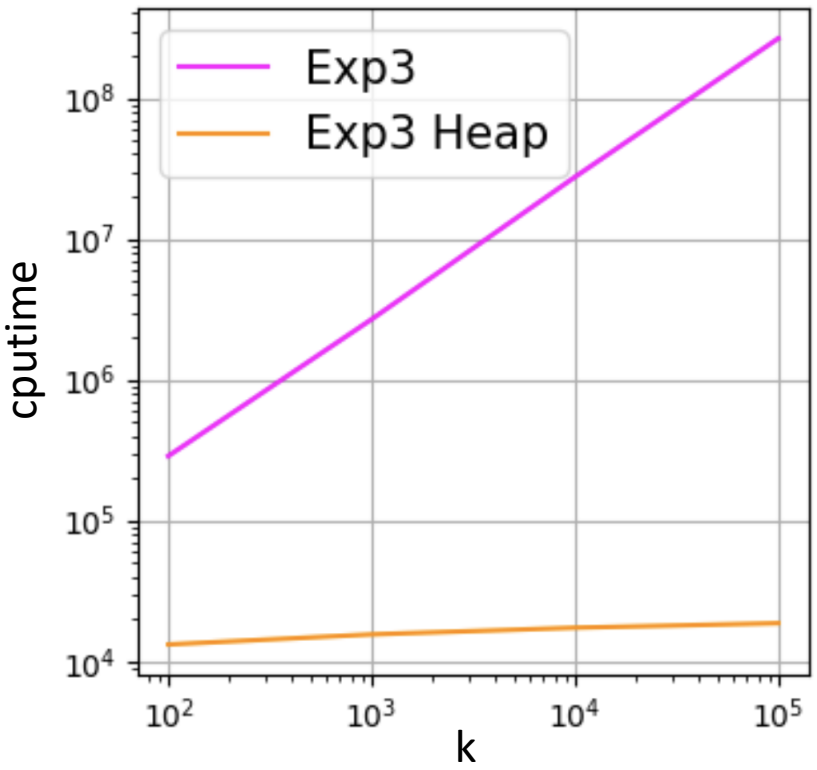
$$LSE(\ln(w_1), \ln(w_2), \ldots, \ln(w_k)) = \ln(w_1 + w_2 + \cdots + w_k)$$

$$\ln(w_x + w_y) = \ln(w_y) + \ln(\exp(\ln(w_x) - \ln(w_y)) + 1)$$

Heap intermediate

Never access *any* weight outside of logspace

Exp3 with Sum heap

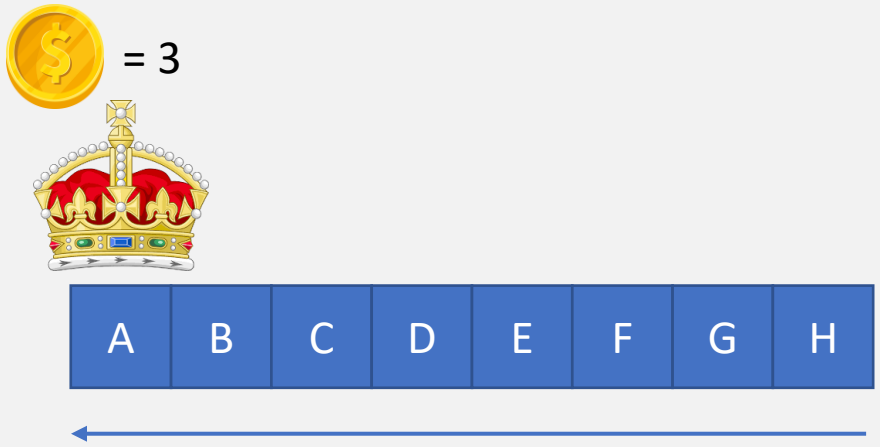


(c) The total runtime of the algorithms, including sampling and updates.

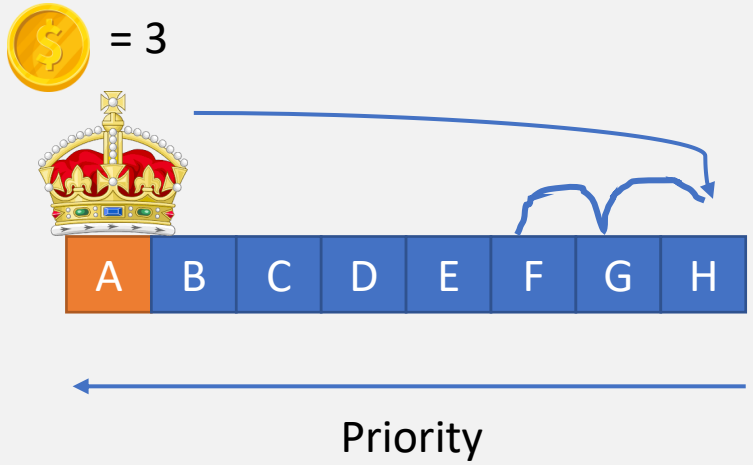
QBL – Explicit Demotion

Explicit Demotion

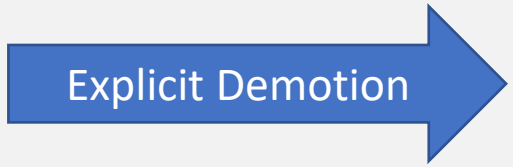
Caps priority to k.
Leaders accumulate wealth. Maximum $k - 1$.



Bad leaders are demoted based on and loose their wealth.



QBL – Explicit Demotion



Problem: Exploration not guaranteed

Solution: Implicit promotion.

QBL – Implicit promotion

Implicit Promotion

Shift the allowed priority interval

Remember: Demote by at least 1

Option:	A	B	C	D	E	F	G	H
Priority of option:	8	6	5	5	2	1	1	0	-1

max: 8 — Priority — min: $8 - k - 1$

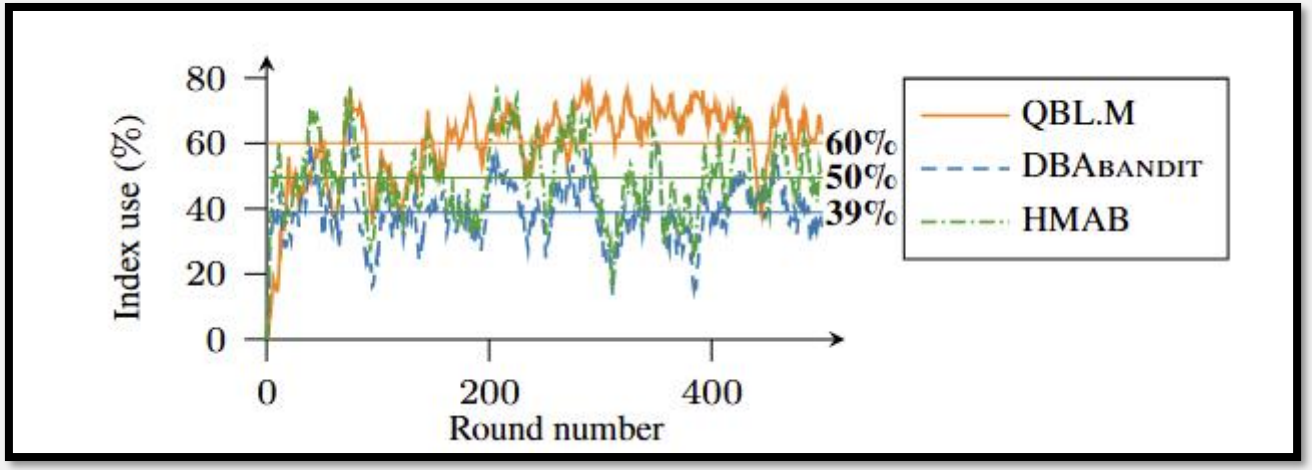
Max is priority of current leader

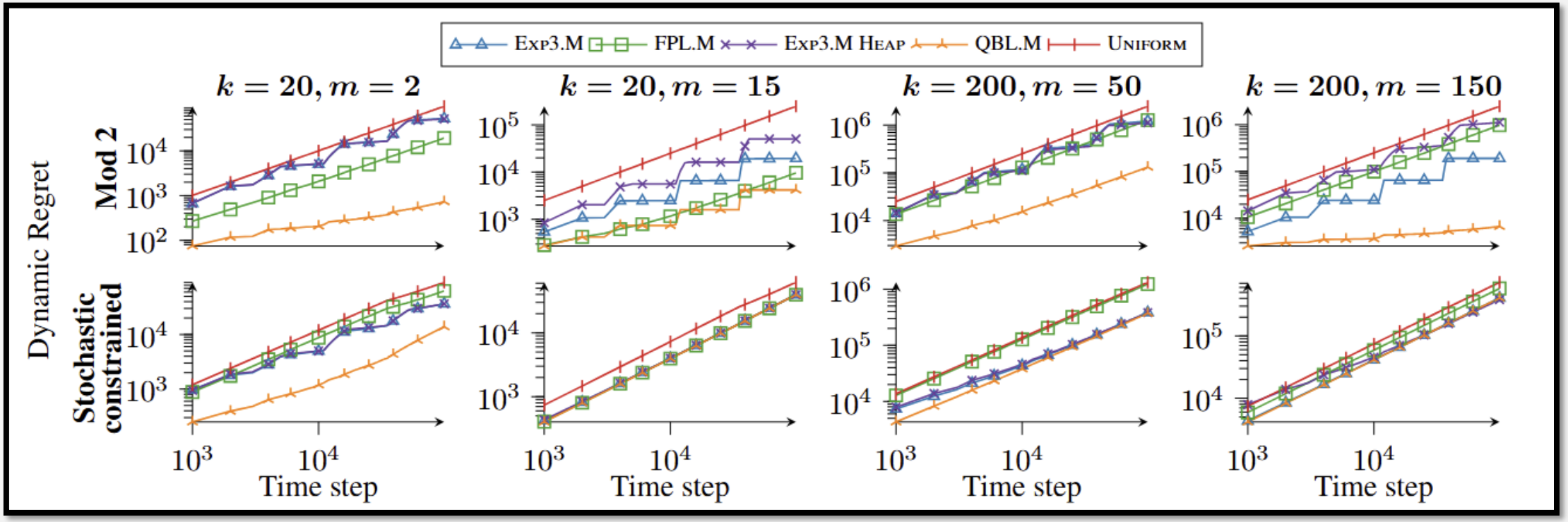
Min is set based on max

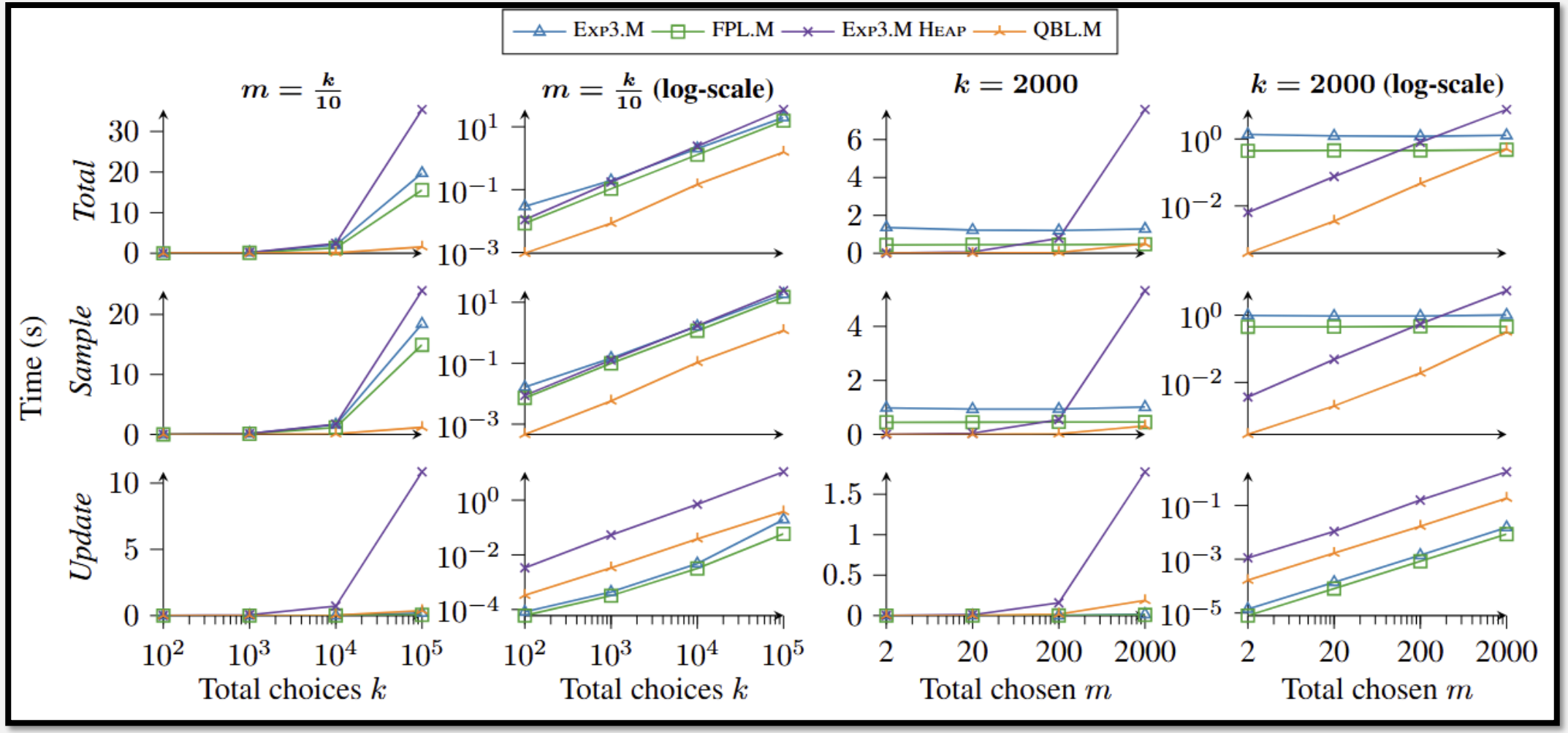
A	B	C	D	E	F	G	H	
8	6	5	5	2	1	1	0	-1

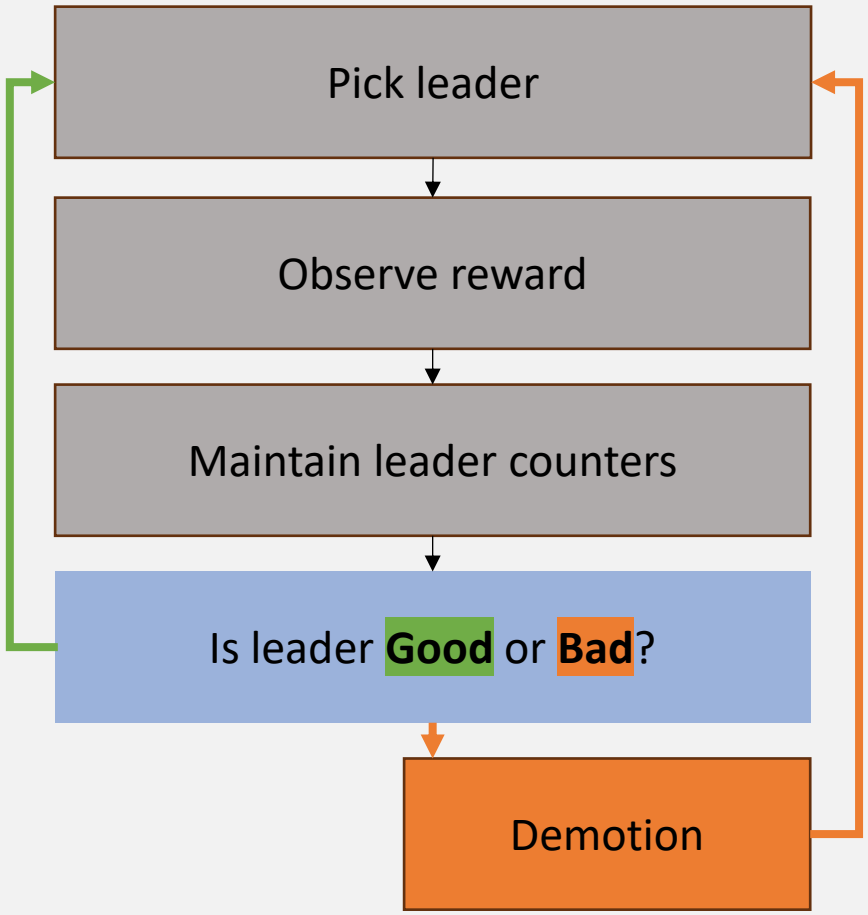
	B	C	D	E	F	G	H	A	
8	6	5	5	2	1	1	0	-1	-2	-3

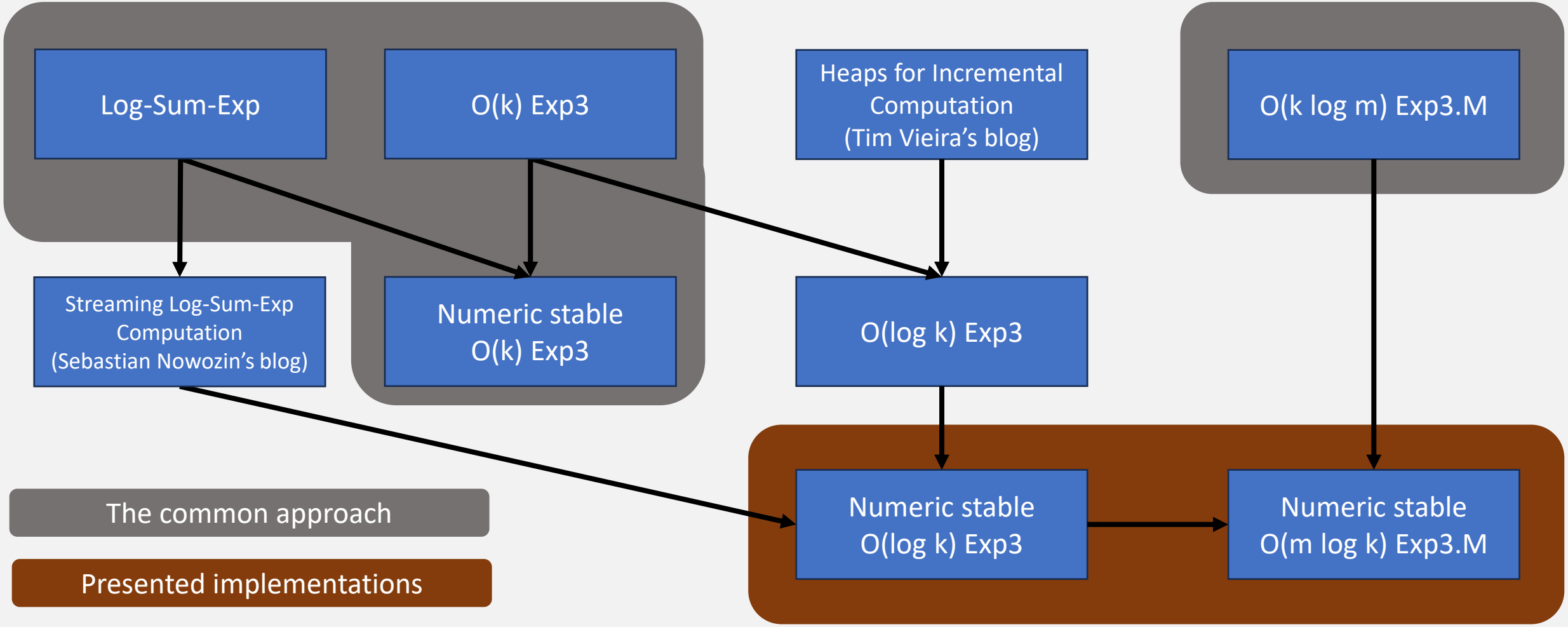
max: 6 — Priority — min: $6 - k - 1$


80
60
40
20
0
Index use (%)
0
200
400
Round number
60%
50%
39%
QBL.M
DBABANDIT
HMAB


Exp3.M
FPL.M
Exp3.M Heap
QBL.M
Uniform
$k = 20, m = 2$
$k = 20, m = 15$
$k = 200, m = 50$
$k = 200, m = 150$
Dynamic Regret
Mod 2
Stochastic constrained
Time step
Time step
Time step
Time step


Exp3.M
FPL.M
Exp3.M Heap
QBL.M
$m = \frac{k}{10}$
$m = \frac{k}{10}$ (log-scale)
$k = 2000$
$k = 2000$ (log-scale)
Time (s)
Total
Sample
Update
Total choices k
Total choices k
Total chosen m
Total chosen m


Pick leader
Observe reward
Maintain leader counters
Is leader Good or Bad?
Demotion


Log-Sum-Exp
O(k) Exp3
Heaps for Incremental Computation (Tim Vieira's blog)
O(k log m) Exp3.M
Streaming Log-Sum-Exp Computation (Sebastian Nowozin's blog)
Numeric stable O(k) Exp3
O(log k) Exp3
Numeric stable O(log k) Exp3
Numeric stable O(m log k) Exp3.M
The common approach
Presented implementations