Upper and Lower Bounds for Deterministic Approximate Objects

Danny Hendler, Ben-Gurion University
Adnane Khattabi, LaBRI – CNRS
Alessia Milani, LaBRI
Corentin Travers, LaBRI

Context

- **Linearizable shared objects** are fundamental building blocks for distributed algorithms (e.g. Counter, Max Register, Stack etc.).

- The implementations of linearizable shared objects are expensive. (e.g. [Attiya, and Welch, TOCS '94])
- **Relaxing the semantics** of shared objects to improve the complexity of their implementations.
  - k-additive, [Aspnes, Attiya, and Censor-Hillel, J. ACM '12]
  - k-multiplicative, [Aspnes, Censor-Hillel, Attiya, and Hendler, SIAM J. Comput '16]
  - k-out-of-order, stuttering, [Henzinger, Kirsch, Payer, Sezgin, and Sokolova, POPL '13]

How would the relaxation of the sequential specification of different concurrent objects affect their complexity?

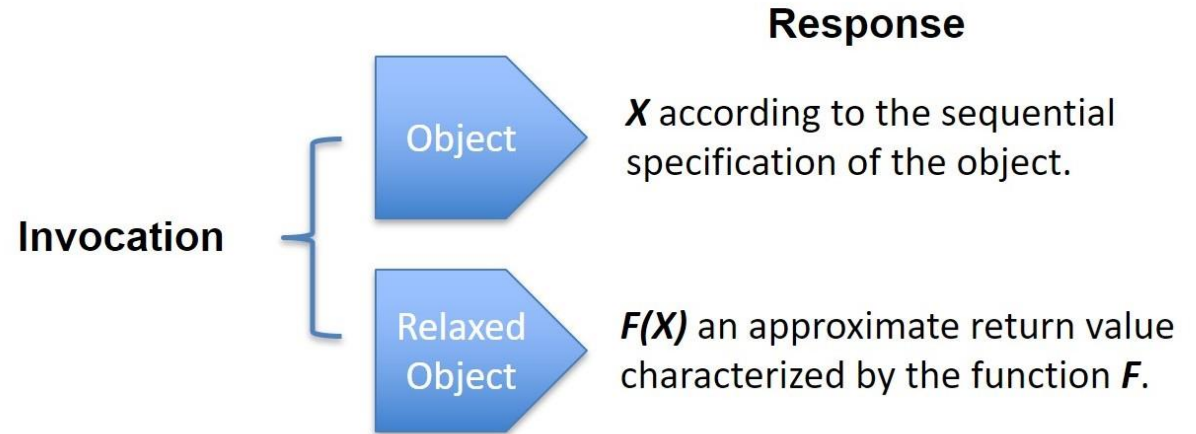
Context

- K-additive, [Aspnes, Attiya, and Censor-Hillel, J. ACM '12]

F(X) = X ± i where (i≤k)

- K-multiplicative, [Aspnes, Censor-Hillel, Attiya, and Hendler, SIAM J. Comput '16]

F(X) = (X × i) OR (X ÷ i) where (i≤k)



Max Register Object (sequential specification)

- *Write(v)*: writes the value *v* to the register, if *v* is larger than all previous values written to the register.
- Read(): returns the maximum value written to the register.
- An **m-bounded** max register can only execute *Write(v)* operations where $v<m$.

- Relaxation of the exact max register object into a **k-multiplicative-accurate** max register.
- *Relaxed Read()* returns a value within a **k** multiplicative factor from the value returned by the exact max register.

Counter Object (sequential specification)

- *CounterIncrement*: increases the value of the counter by 1.
- C*ounterRead*: returns the number of previous calls to *CounterIncrement*.

- Relaxation of the exact counter object into a **k-multiplicative-accurate** counter.
- *Relaxed CounterRead* returns a value within a **k** multiplicative factor from the number of previous calls to *CounterIncrement*.

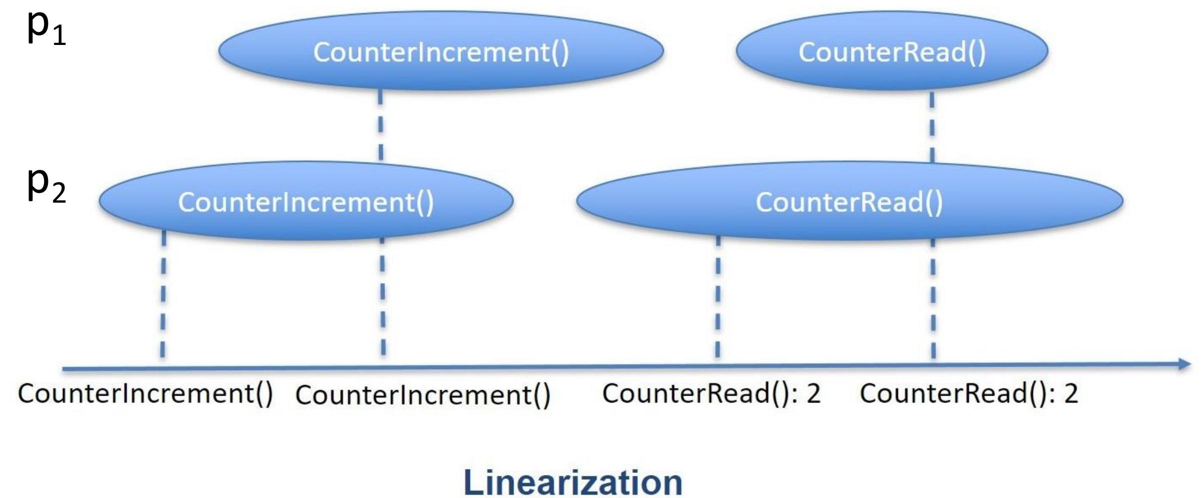
Model & Correctness

- n asynchronous deterministic processes with unique IDs that are prone to crashing (fail-stop).
- Processes communicate using **read**/**write** and **test&set** primitives via shared memory access.

Model & Correctness

- n asynchronous deterministic processes with unique IDs that are prone to crashing (fail-stop).
- Processes communicate using **read/write** and **test&set** primitives via shared memory access.

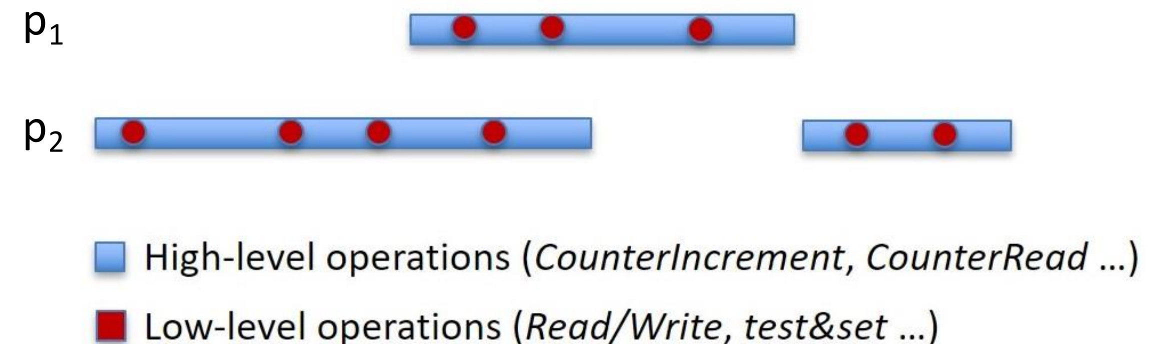
- **Linearizability**: high-level operations appear atomic and the linearization respects real-time order. [Herlihy and Wing, TOPLAS '90]



Model & Correctness

- n asynchronous deterministic processes with unique IDs that are prone to crashing (fail-stop).
- Processes communicate using **read/write** and **test&set** primitives via shared memory access.

- **Wait-Freedom**: all high-level operations finish in a finite number of low-level operations for any interleaving. [Herlihy , TOPLAS '91]



Related Word (Max Register)

	Worst case (step)	
	Lower bound	**Upper bound**
Exact m-bounded Max Register	$\Omega(\log_2 m)$ [Aspnes et al., SIAM J. Comput '16]	$O(\log_2 m)$ [Aspnes et al., SIAM J. Comput '16]

	Amortized (step)
	Upper bound
Exact unbounded Max Register	$O(\log_2 n)$ [Baig et al., DISC '19]

Related Word (Counter)

	Worst case (step)		Amortized (step)
	Lower bound	**Upper bound**	**Upper bound**
Exact Counter	$\Omega(n)$ [Jayanti et al., PODC '96]	$O(n)$ [Inoue et al., IDWA '94] Polylogarithmic for bounded executions. [Aspnes et al., JACM '12]	$O(\log^2 n)$ [Baig et al., DISC '19]

Related Word (Counter)

	Worst case (step)		Amortized (step)
	Lower bound	**Upper bound**	**Upper bound**
Exact Counter	$\Omega(n)$ [Jayanti et al., PODC '96]	$O(n)$ [Inoue et al., IDWA '94] Polylogarithmic for bounded executions. [Aspnes et al., JACM '12]	$O(\log^2 n)$ [Baig et al., DISC '19]

Relaxation of the sequential specification

- **Lower bound on amortized and worst case complexity?**
- **Upper bound on amortized complexity?**

Contributions (Max Register)

- By extension to [Aspnes et al., SIAM J. Comput '16], we prove a lower bound of **$\Omega(\min(n,\log_2\log_k m))$** on the worst case step complexity of any n-process solo terminating implementation of a **k-multiplicative-accurate m-bounded** max register from read/write and conditional primitive operations.

- We present a wait-free implementation of the **k-multiplicative-accurate m-bounded** max register with a worst case step complexity that matches the lower bound.

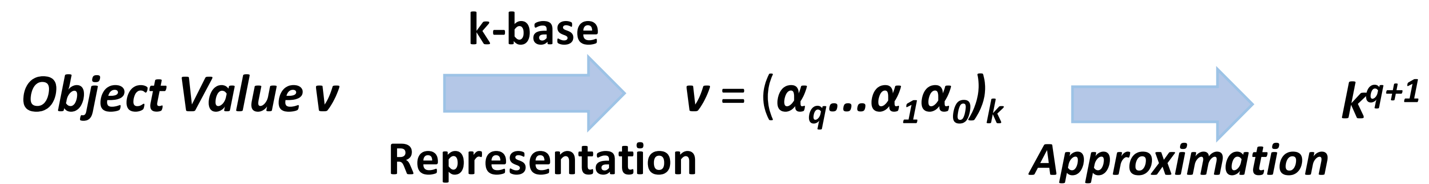
- We plug-in our bounded max register implementation into the construction given by [Baig et al., DISC '19] to obtain an **unbounded k-multiplicative-accurate** max register with **$O(\min(n,\log_2\log_k m))$** amortized complexity.

Contributions (Counter)

- We present the first **deterministic** wait-free read/write and test&set based, **k-multiplicative-accurate** counter implementation with **O(1)** amortized complexity for $k \geq \sqrt{n}$.
- By extension to [Attiya, and Hendler, DISC '05], we prove a lower bound of $\mathbf{\Omega(\log n/k^2)}$ for $k \leq \sqrt{n}/2$ on the amortized step complexity of any n-process solo terminating implementation of a k-multiplicative-accurate counter from read/write and conditional primitive operations.
- We prove that the lower bound of $\Omega(n)$ on the worst case step complexity presented by [Jayanti et al., PODC '96], holds for unbounded k-multiplicative accurate counters.

Algorithms

- **Intuition**:



K-multiplicative-accurate Max Register

- Using a (**$|\log_k(m-1)|$ +1)-**bounded exact max register to implement a k-multiplicative-accurate **m**-bounded max register.
- Exponential improvement in complexity from the implementation of the exact max register in [Aspnes et al., SIAM J. Comput '16].

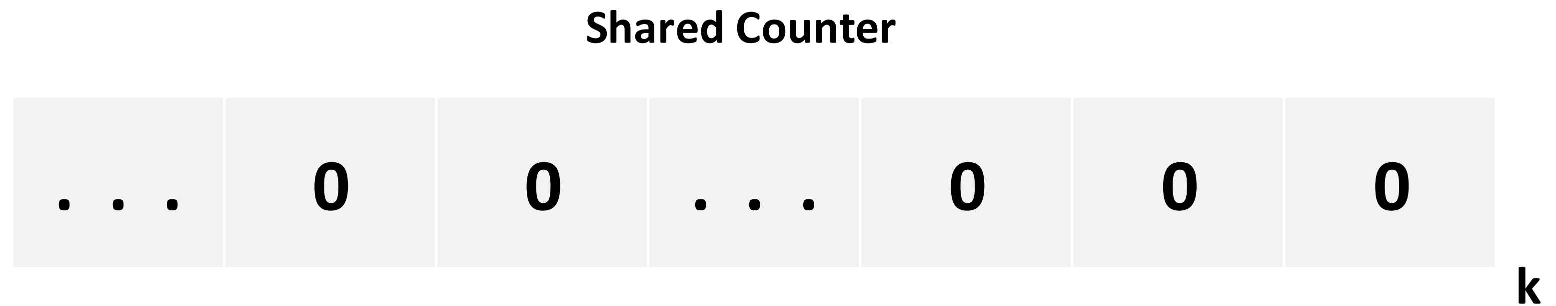
Algorithm 2: A k-multiplicative-accurate m-bounded max register

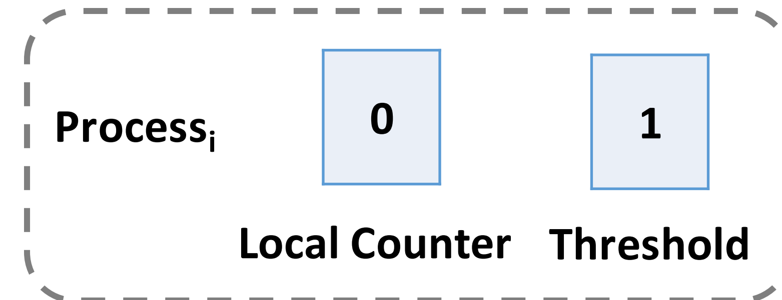
```
1  Shared variables M:
     ((⌊log_k(m − 1)⌋) + 1)-bounded max register
     initially 0
2  Function Read()
3  |   p ⟵ M.read()
4  |   if p=0 then return 0;
5  |   else return k^p;
6  end
7  Function Write(v)
8  |   p ⟵ ⌊log_k v⌋ + 1;
9  |   M.write(p);
10 end
```

K-multiplicative-accurate Counter (CounterIncrement)

- The shared counter is an infinite array of bits.

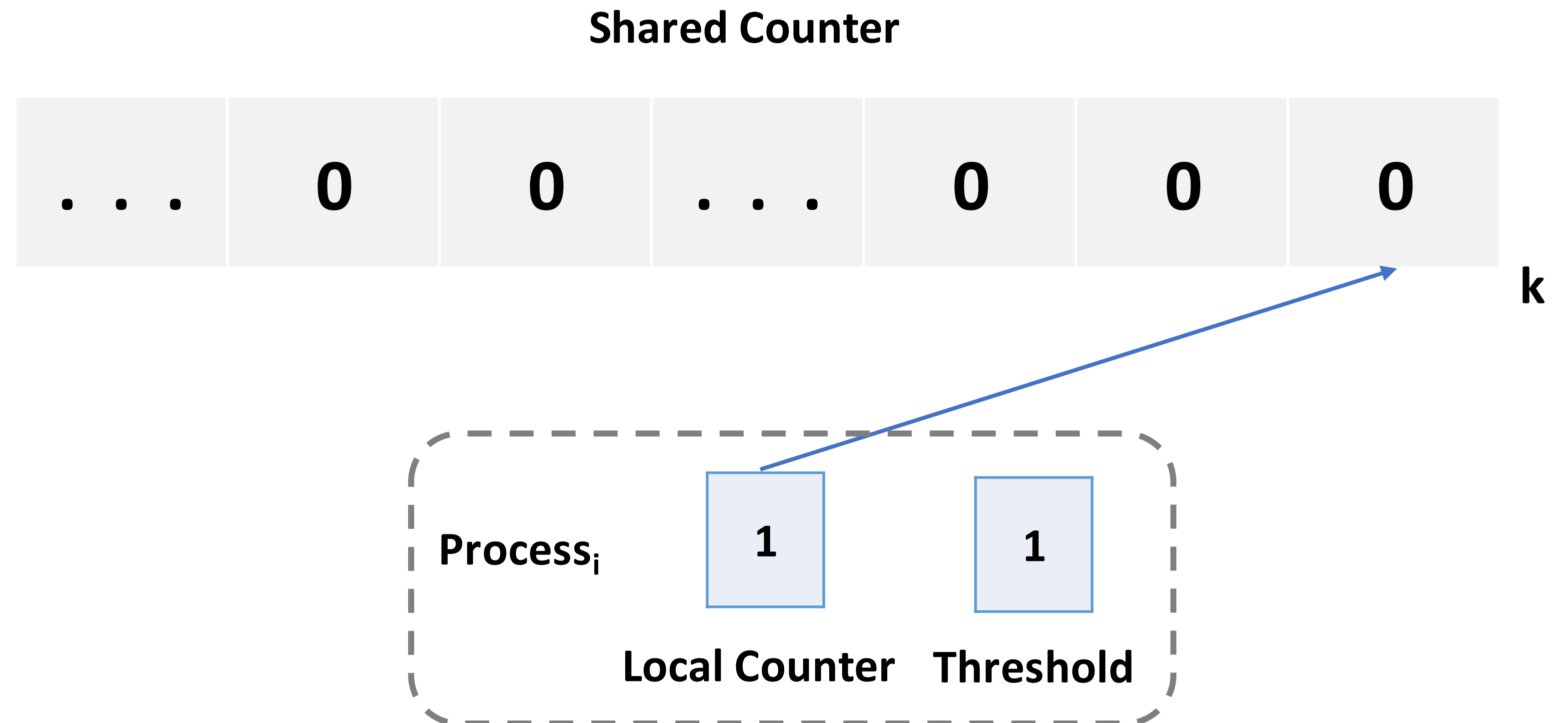
- Initially, the local counter is set to 0 and the threshold to 1.





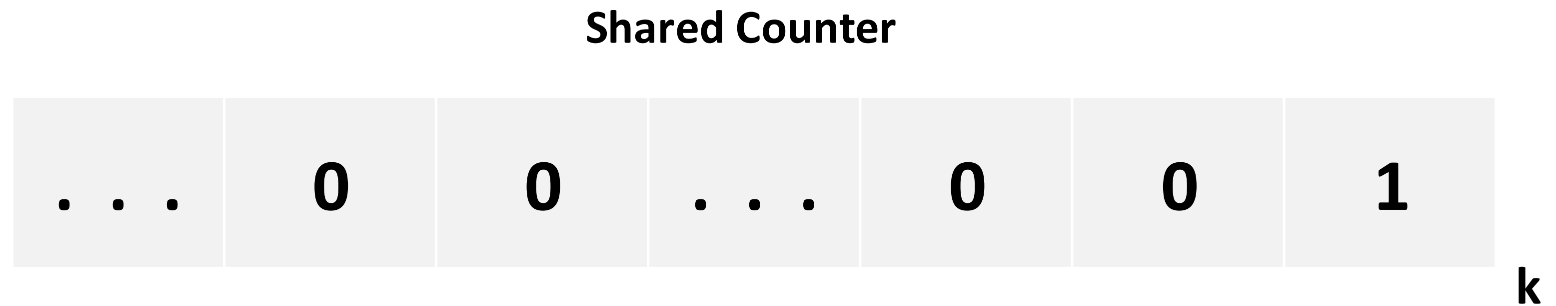
K-multiplicative-accurate Counter (CounterIncrement)

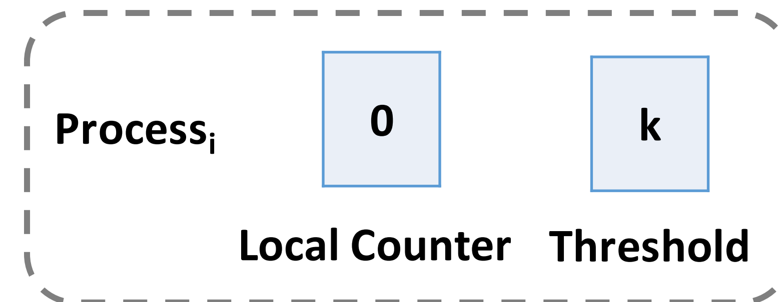
- The shared counter is an infinite array of bits.
- Initially, the local counter is set to 0 and the threshold to 1.
- After the first call to *CounterIncrement()*, the process attemps to set the first bit.



K-multiplicative-accurate Counter (CounterIncrement)

- The shared counter is an infinite array of bits.
- Initially, the local counter is set to 0 and the threshold to 1.
- After the test&set succeeds, the process resets its local counter and updates the threshold.

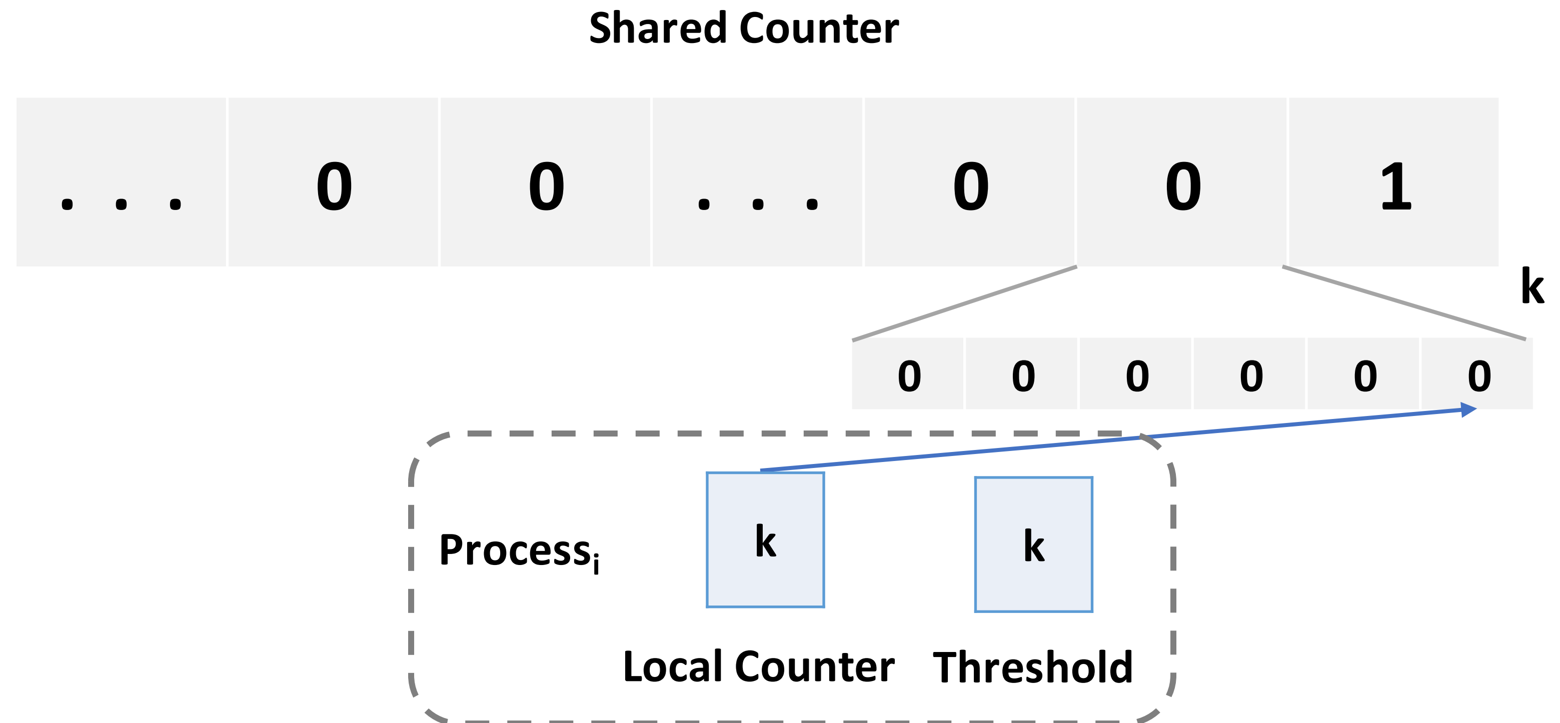




K-multiplicative-accurate Counter (CounterIncrement)

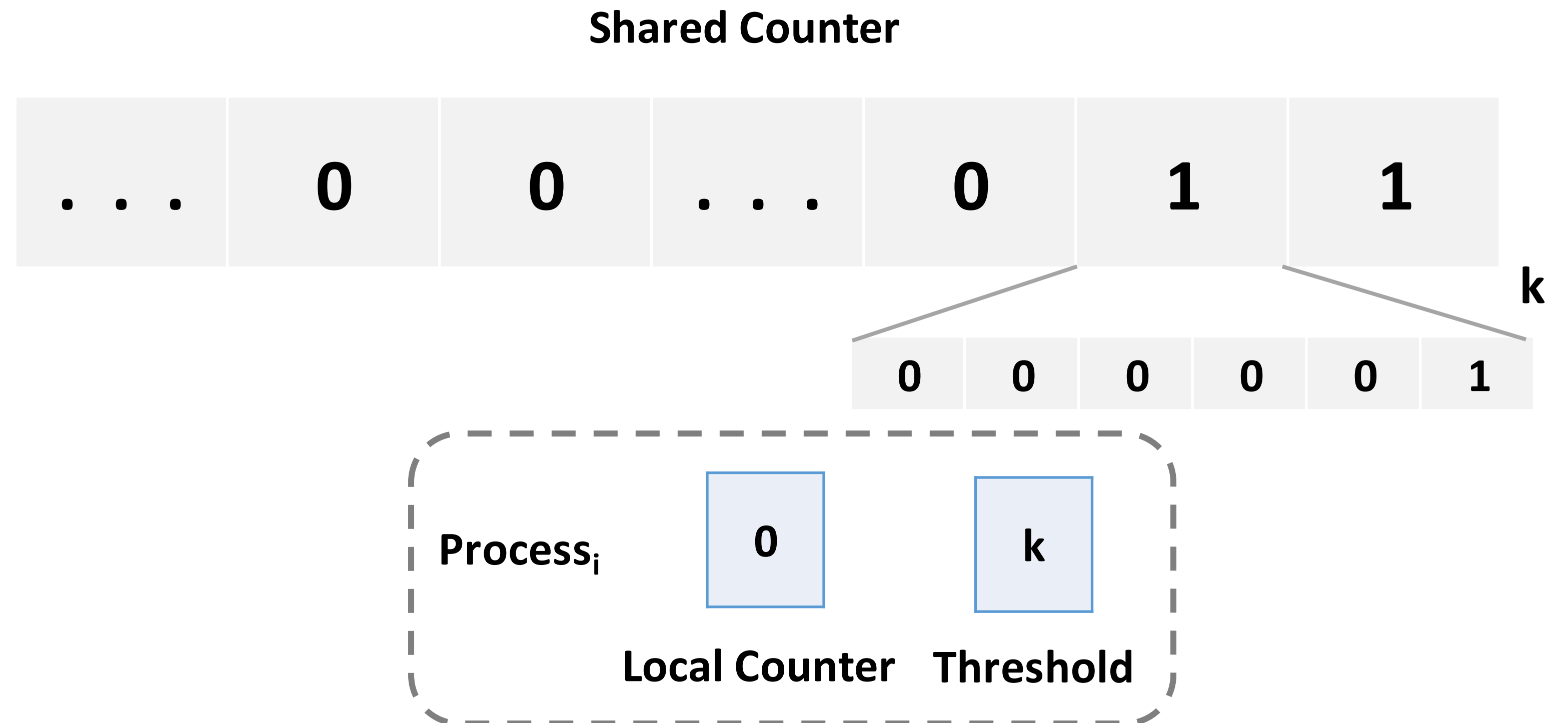
- After the first bit, the shared counter is segmented into **sets of k bits**.

- After the local counter reaches the new threshold of k, the process attempts to set a bit in the first set.



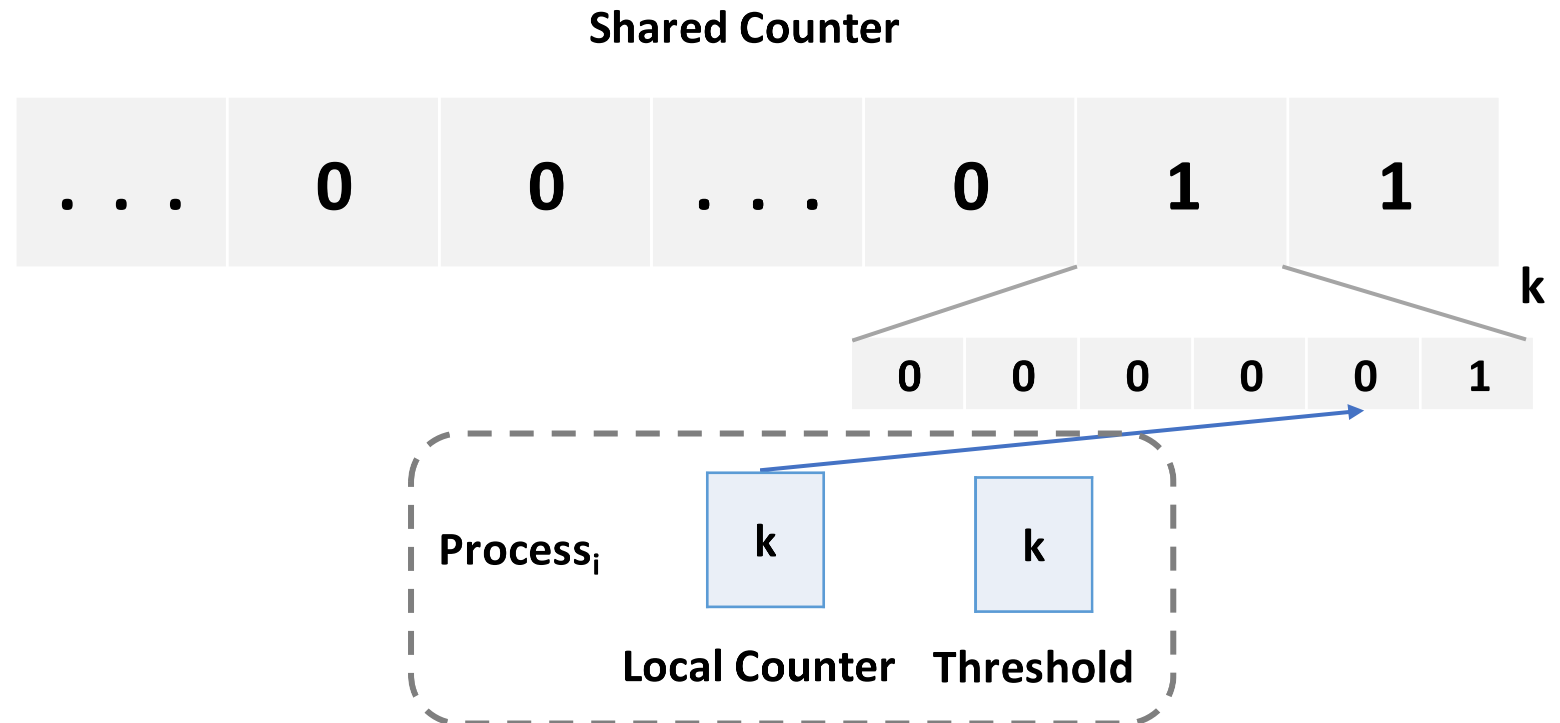
K-multiplicative-accurate Counter (CounterIncrement)

- After the first bit, the shared counter is segmented into **sets of k bits**.
- After the local counter reaches the new threshold of k, the process attempts to set a bit in the first set.
- Until all bits of a set are set, the threshold remains the same.



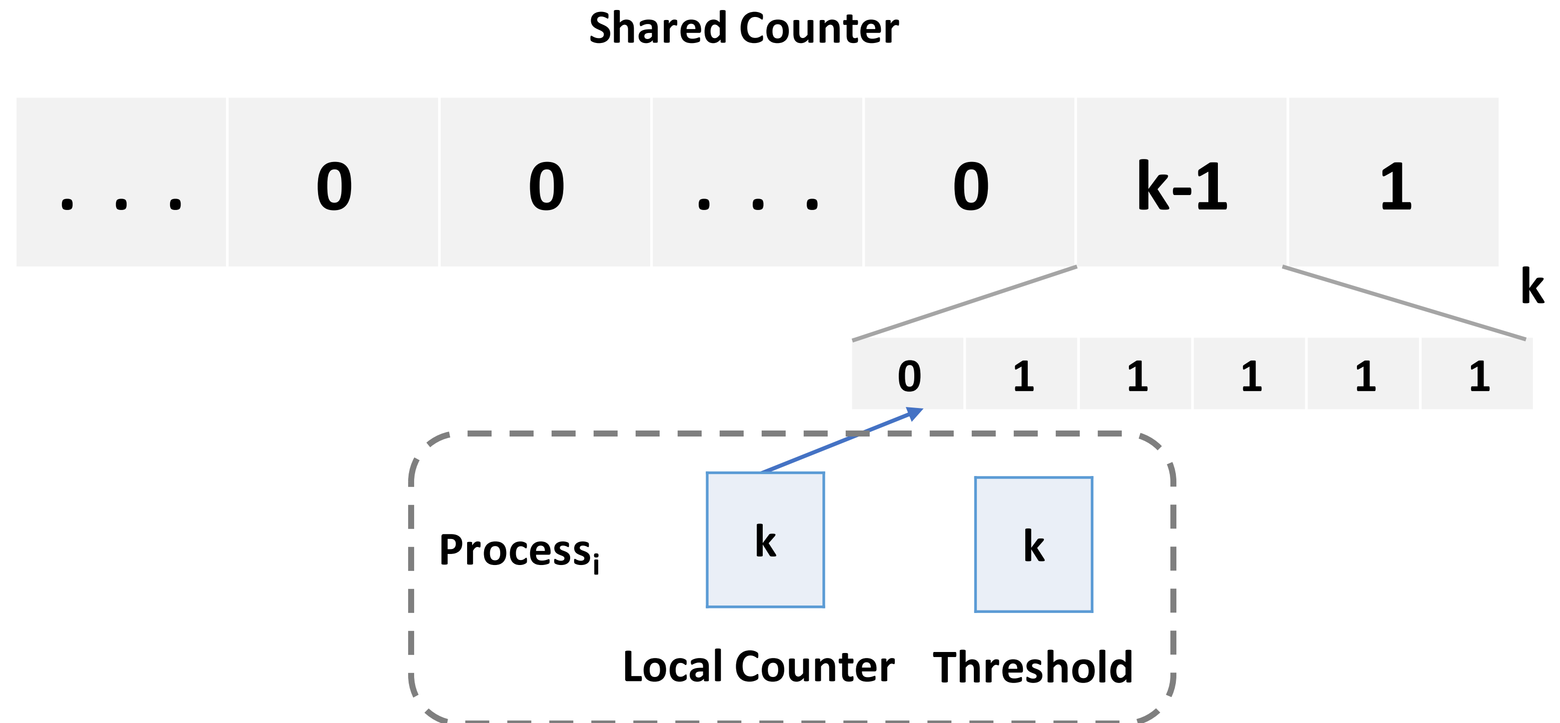
K-multiplicative-accurate Counter (CounterIncrement)

- After the first bit, the shared counter is segmented into **sets of k bits**.
- After the local counter reaches the new threshold of k, the process attempts to set a bit in the first set.
- Until all bits of a set are set, the threshold remains the same.



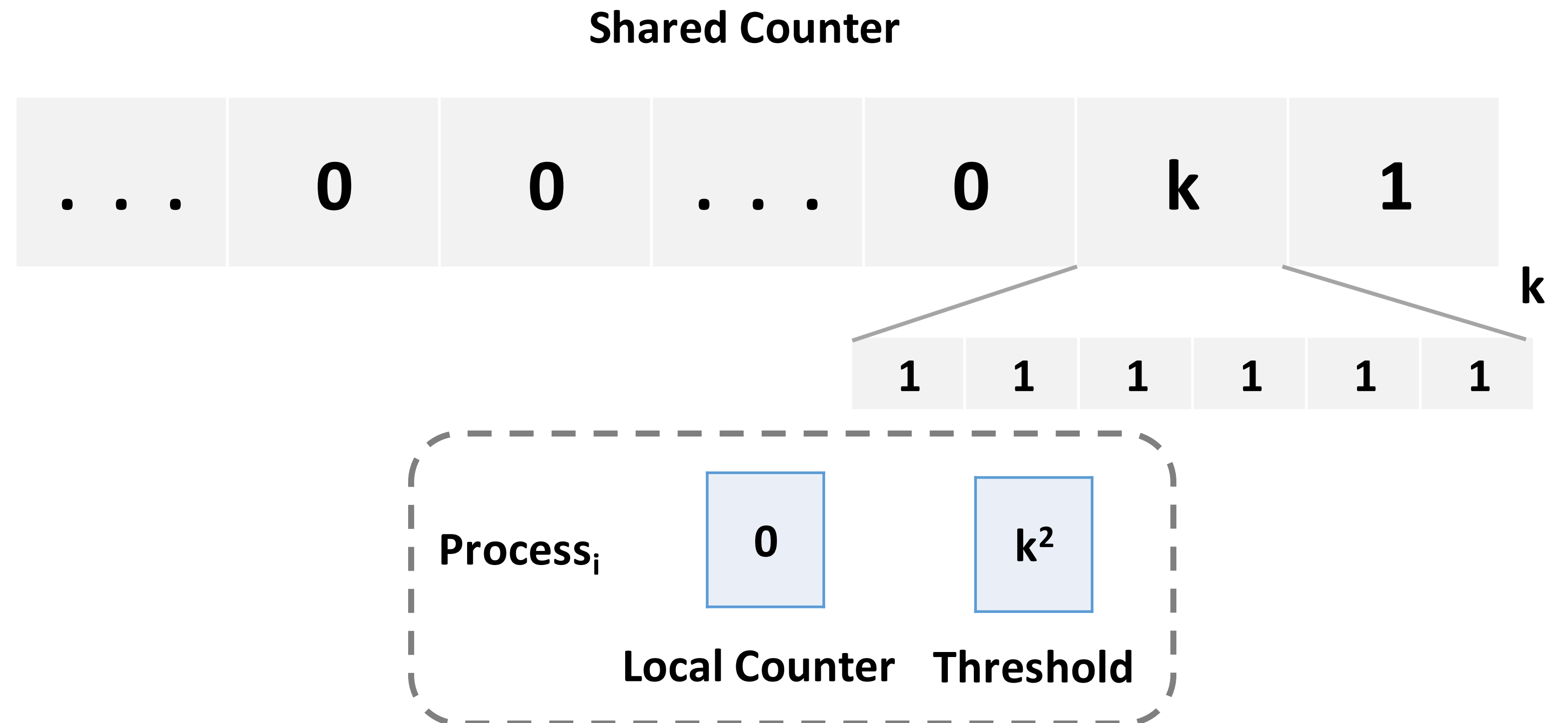
K-multiplicative-accurate Counter (CounterIncrement)

- After the first bit, the shared counter is segmented into **sets of k bits**.
- After the local counter reaches the new threshold of k, the process attempts to set a bit in the first set.
- Until all bits of a set are set, the threshold remains the same.



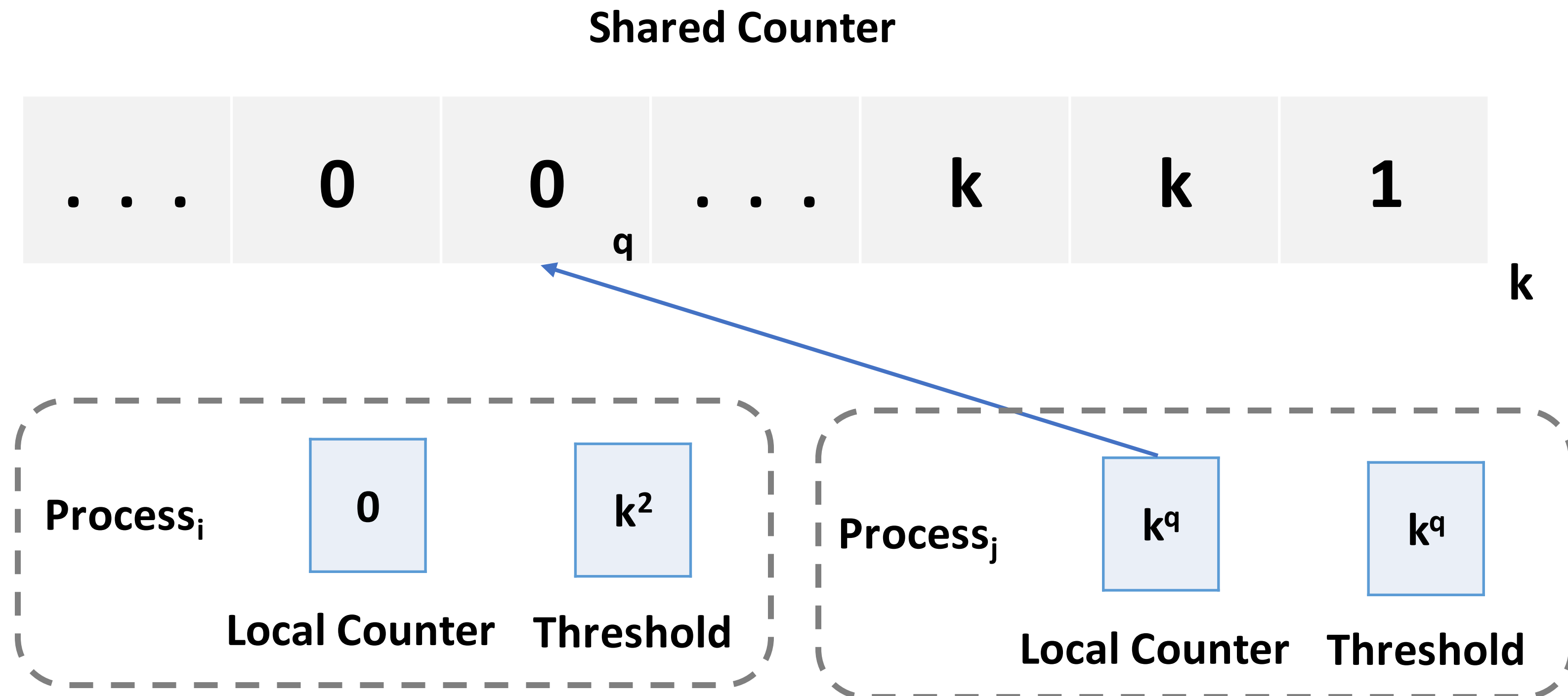
K-multiplicative-accurate Counter (CounterIncrement)

- After the first bit, the shared counter is segmented into **sets of k bits**.
- After the local counter reaches the new threshold of k, the process attempts to set a bit in the first set.
- Until all bits of a set are set, the threshold remains the same.



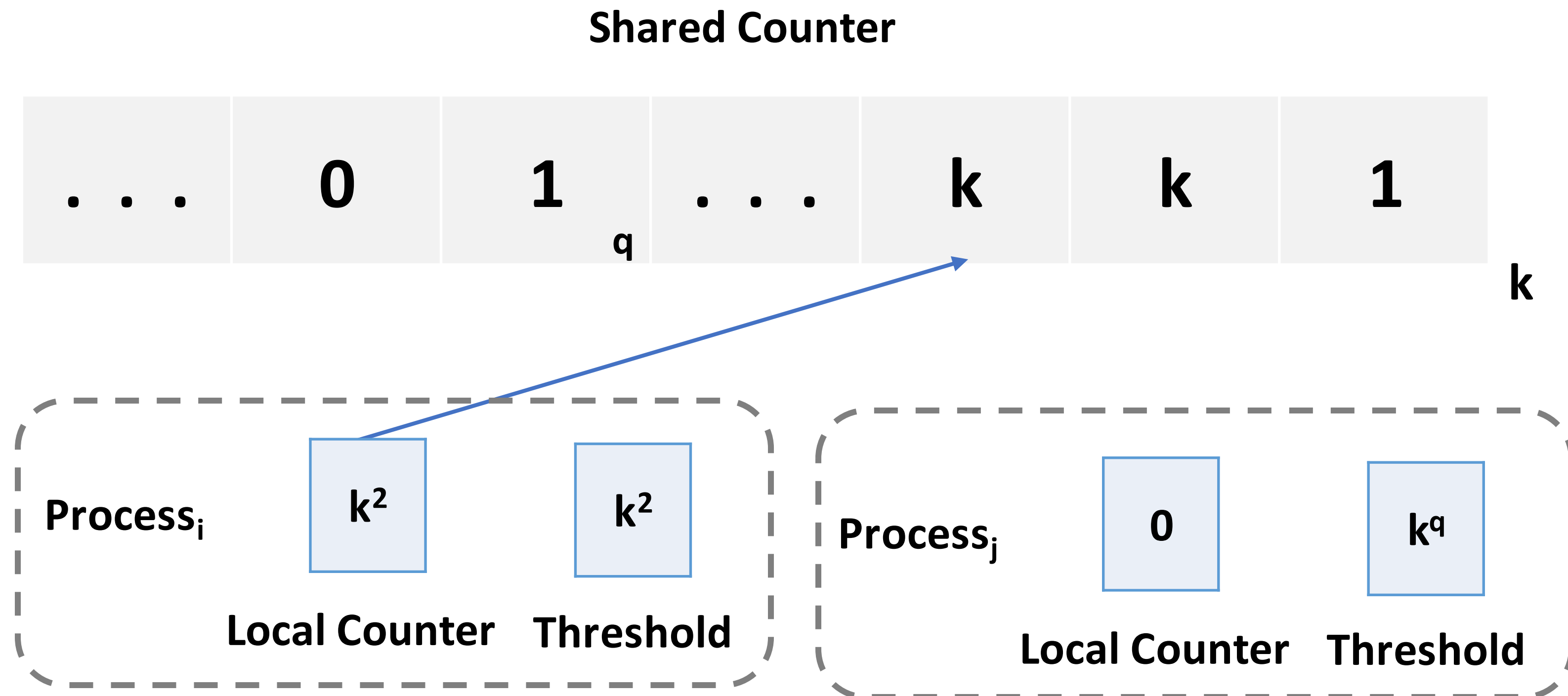
K-multiplicative-accurate Counter (CounterIncrement)

- Threshold value may be outdated due to increment operations by other processes.



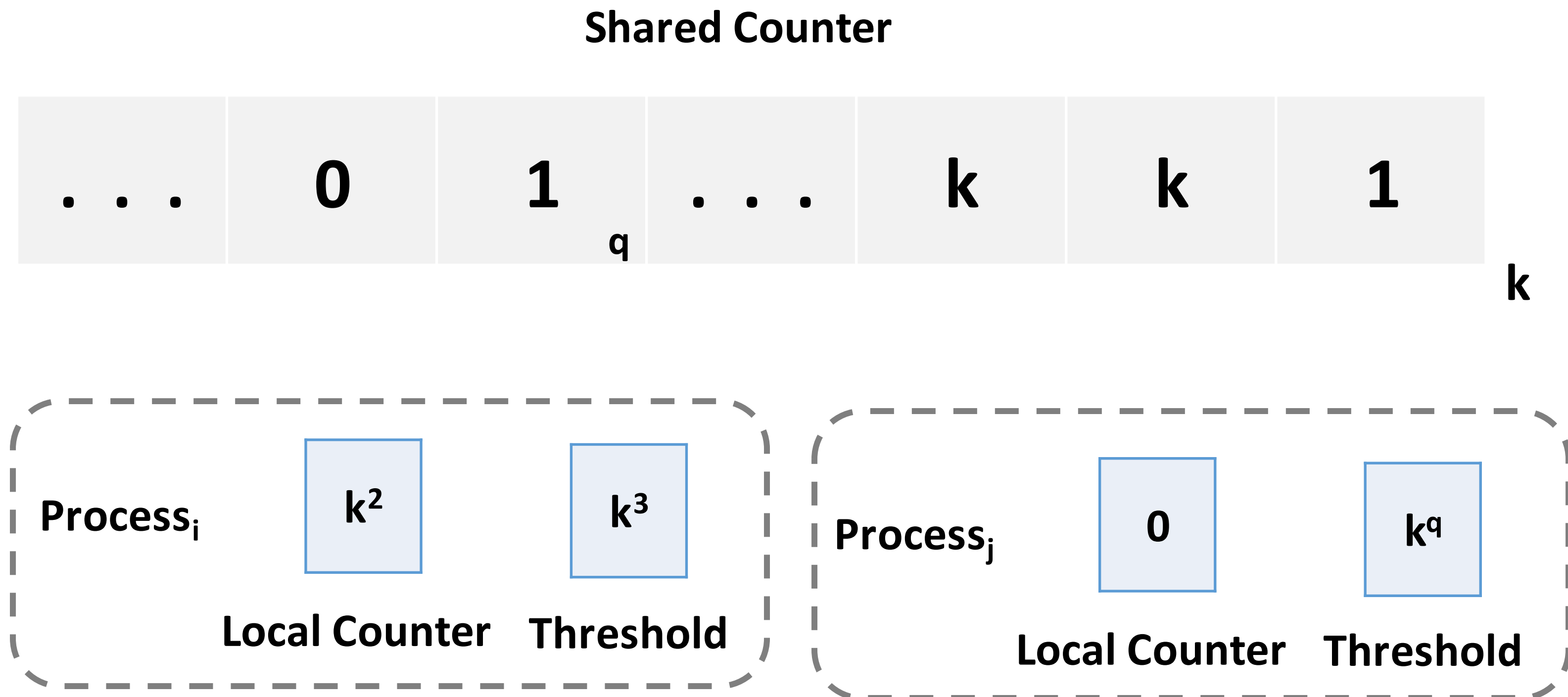
K-multiplicative-accurate Counter (CounterIncrement)

- Threshold value may be outdated due to increment operations by other processes.
- When an attempt at setting a bit fails, the process updates the threshold but keeps the local counter value.



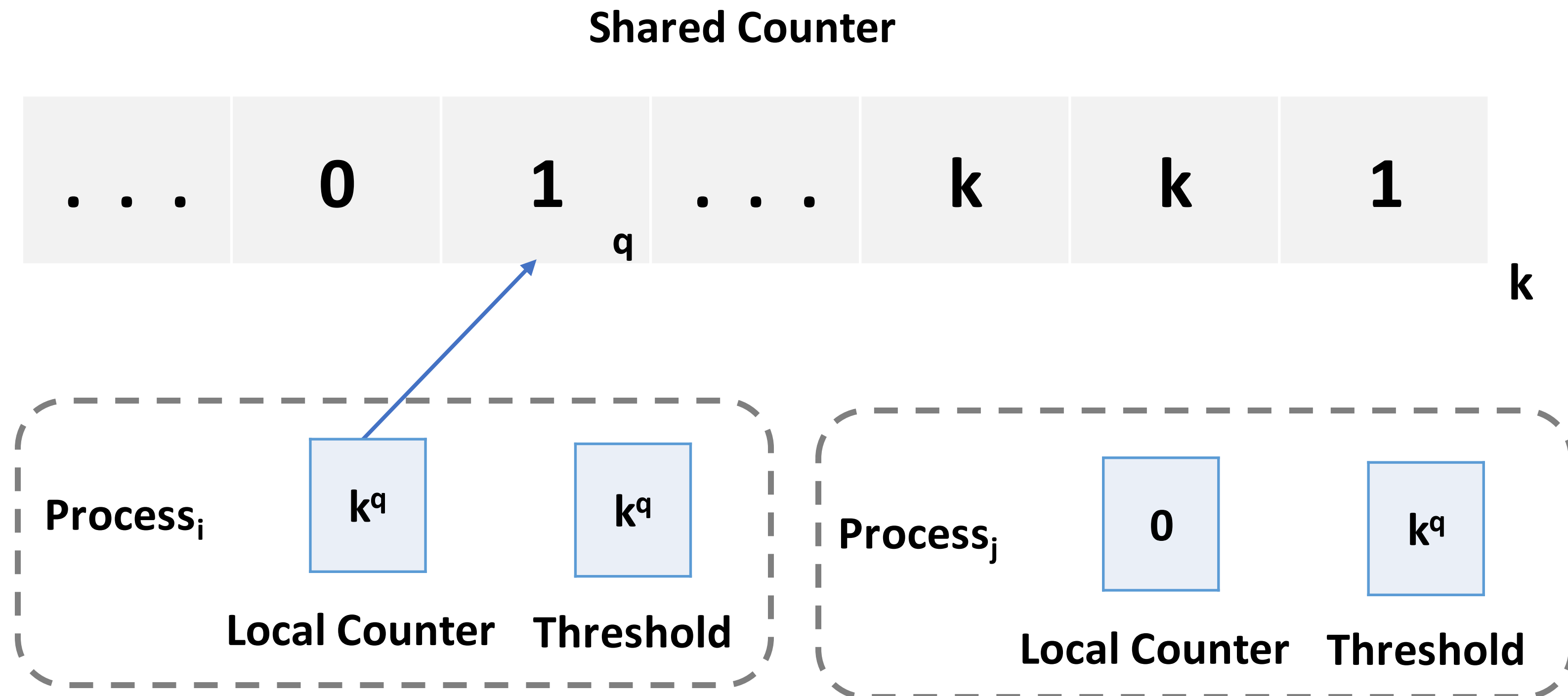
K-multiplicative-accurate Counter (CounterIncrement)

- Threshold value may be outdated due to increment operations by other processes.
- When an attempt at setting a bit fails, the process updates the threshold but keeps the local counter value.



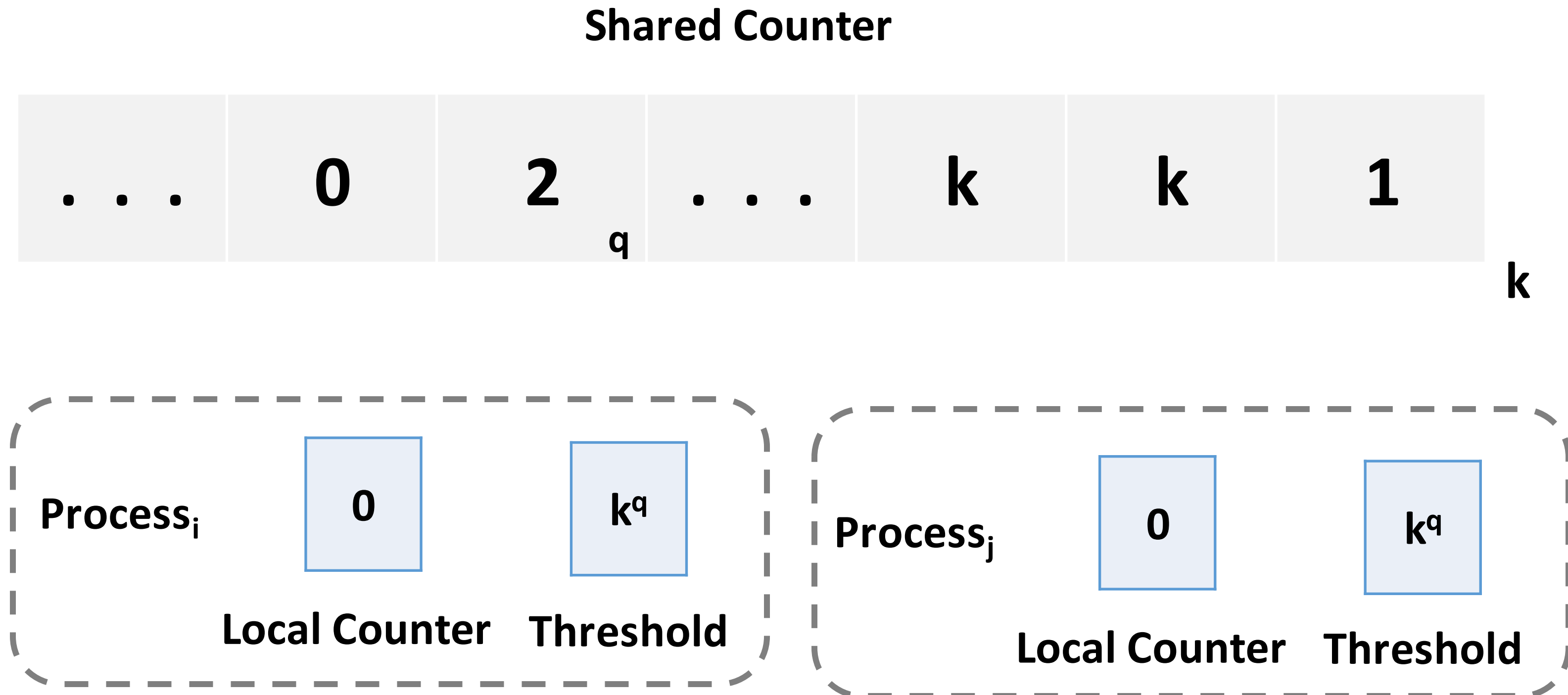
K-multiplicative-accurate Counter (CounterIncrement)

- Threshold value may be outdated due to increment operations by other processes.
- When an attempt at setting a bit fails, the process updates the threshold but keeps the local counter value.
- When the threshold reaches the current state of the shared counter, the process may set a bit.



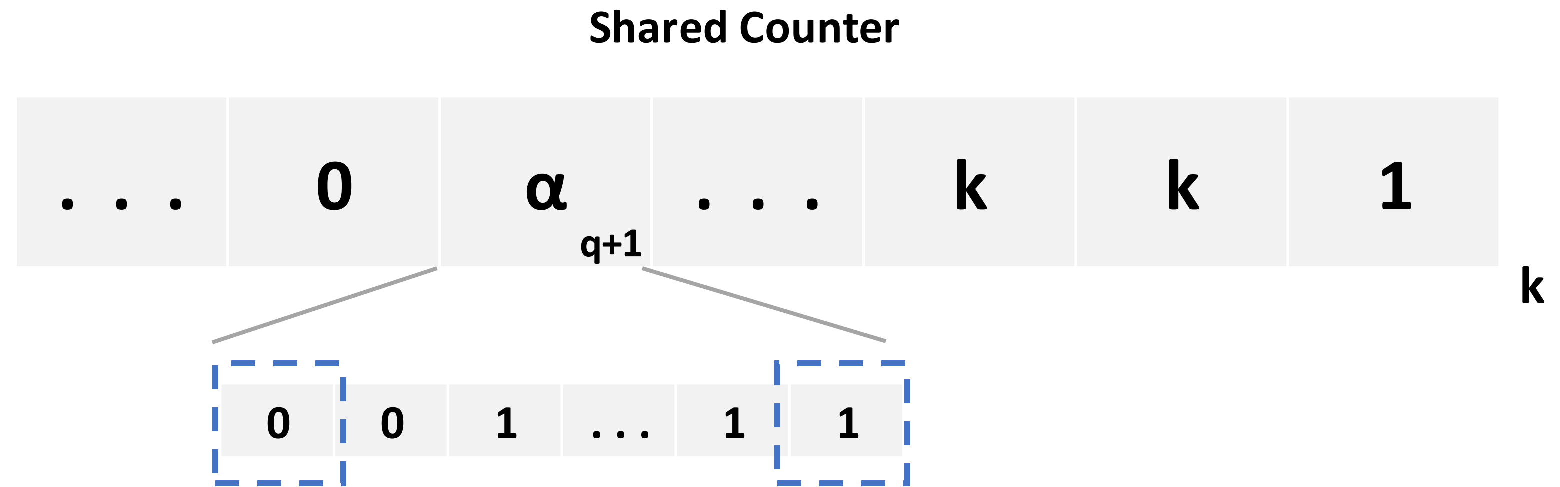
K-multiplicative-accurate Counter (CounterIncrement)

- Threshold value may be outdated due to increment operations by other processes.
- When an attempt at setting a bit fails, the process updates the threshold but keeps the local counter value.
- When the threshold reaches the current state of the shared counter, the process may set a bit.



K-multiplicative-accurate Counter (CounterRead)

- Searching for the first bit that has not already been set by a process.
- The process only checks the first and last bit of each set.
- Based on the index, the process computes an approximation of the counter value within a k-multiplicative range.



Summary

	Worst case (step)		Amortized (step)
	Lower bound	**Upper bound**	**Upper bound**
Exact Counter	$\Omega(n)$ [Jayanti et al., PODC '96]	$O(n)$ [Inoue et al., IDWA '94] Polylogarithmic for bounded executions. [Aspnes et al., JACM '12]	$O(\log^2 n)$ [Baig et al., DISC '19]
k-multiplicative-accurate Counter	$\mathbf{\Omega(n)}$		**$O(1)$** For $k \geq \sqrt{n}$ **$\Omega(\log n/k^2)$** When $k \leq \sqrt{n}/2$

Conclusion & Discussion

	Worst case (step)	
	Lower bound	**Upper bound**
Exact m-bounded Max Register	$\Omega(\log_2 m)$ [Aspnes et al., SIAM J. Comput '16]	$O(\log_2 m)$ [Aspnes et al., SIAM J. Comput '16]
m-bounded k-multiplicative-accurate Max Register	$\Omega(\min(n, \log_2 \log_k m))$	$O(\min(n, \log_2 \log_k m))$

	Amortized (step)
	Upper bound
Exact unbounded Max Register	$O(\log_2 n)$ [Baig et al., DISC '19]
unbounded k-multiplicative-accurate Max Register	$O(\min(n, \log_2 \log_k m))$

Thank you for your attention!

- Presented during ICDCS21
- Ref: https://hal.archives-ouvertes.fr/hal-03202712v2