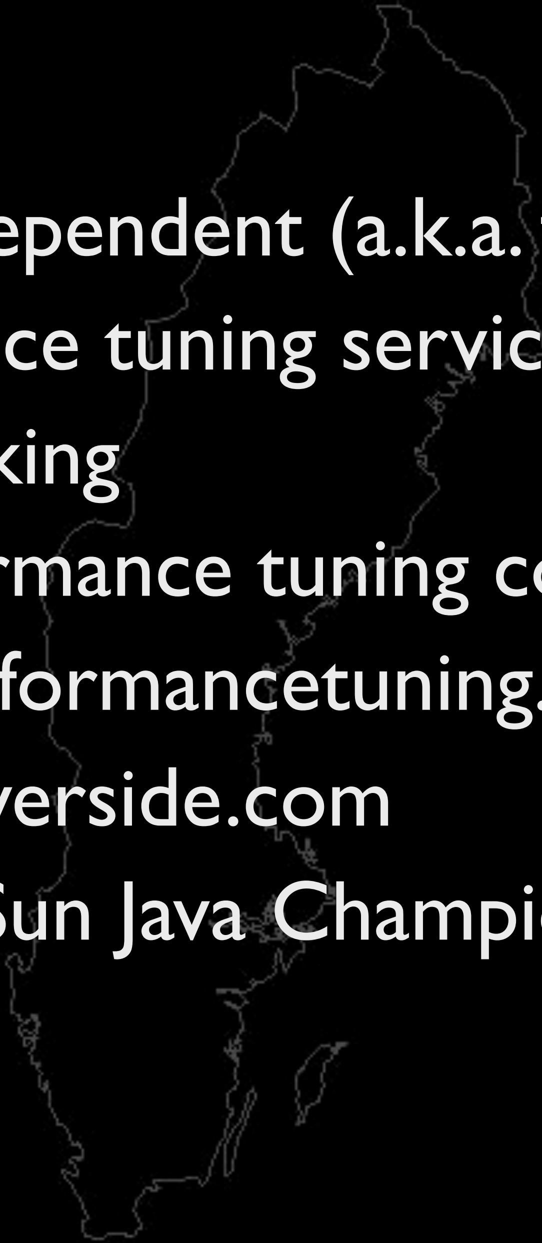
Concurrency and High Performance Reloaded

Me



- Work as independent (a.k.a. freelancer)
  - performance tuning services
  - benchmarking
  - Java performance tuning course
- www.javaperformancetuning.com
- www.theserverside.com
- Nominated Sun Java Champion
- Other stuff

Kodewerk
Java tm Performance Services

single-threaded, single-core

Kodewerk
Java™ Performance Services

how did we get better performance?

Kodewerk
Java™ Performance Services

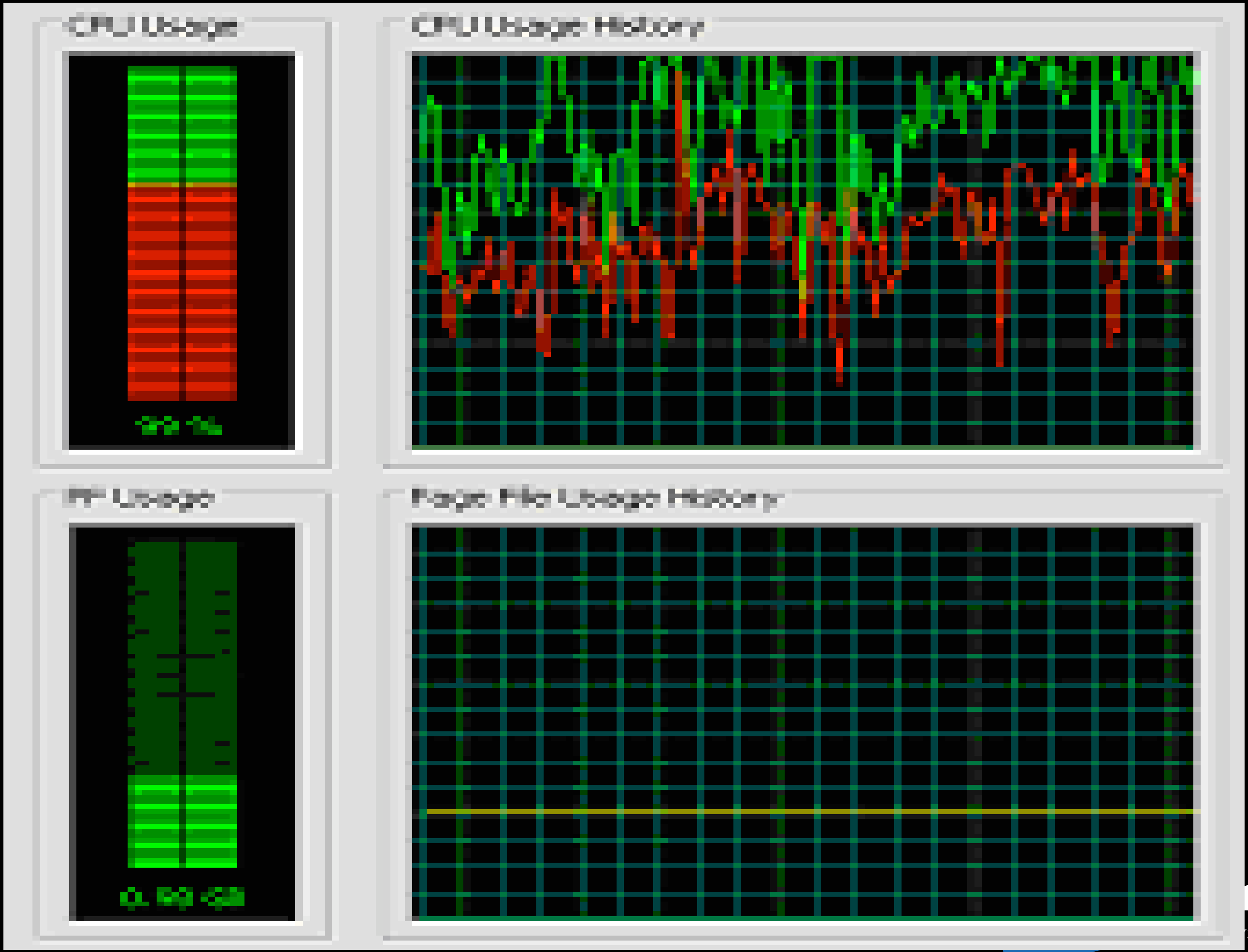
concurrent programming is the norm

Kodewerk
Java™ Performance Services

sharing adds latency

Kodewerk

Java™ Performance Services


CPU Usage
99 %
CPU Usage History
PF Usage
0.99 GB
Page File Usage History

rk
ices

multi-core is a fact of life!

Kodewerk

Java™ Performance Services

we need to "deliver twice as much concurrency every 18 mounts"

Kodewerk
Java™ Performance Services

hardware components are notsharable

Kodewerk
Java™ Performance Services

access to shared data must be serialized

Kodewerk

Java™ Performance Services

databases offer access to shared data

Kodewerk
Java^tm Performance Services

serialization limits scalability

Kodewerk
Java™ Performance Services

- Maths to explain relationship between serialized execution and processor utilization

$$\frac{1}{F + \frac{(1 - F)}{N}}$$

  - F -> 0 number of utilized CPU -> N
  - F -> 1 number of utilized CPU -> 1

Amdahl's Law

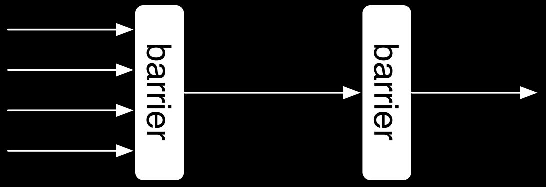
Kodewerk
Java™ Performance Services

serialization limits throughput

Kodewerk

Java™ Performance Services

- Maths explaining the relationship between locking and throughput

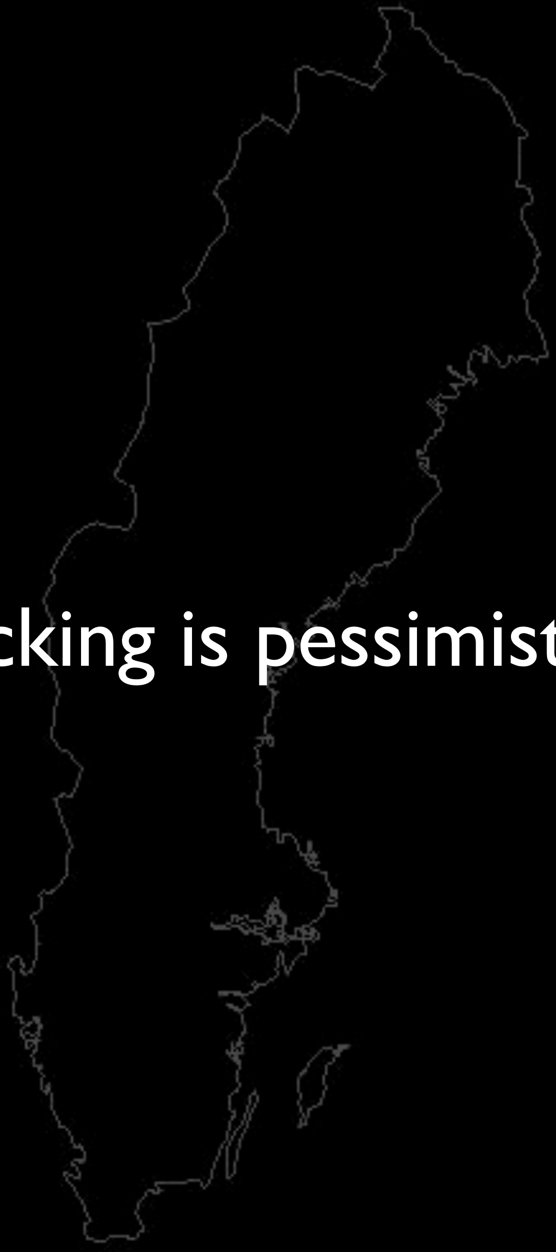


$\lambda = 1 / \mu$

μ = 10ms, λ = 100 tps

μ = 100ms, λ = 10 tps

Little's Law



locking is pessimistic

Kodewerk
Java™ Performance Services

getting better concurrency in the JVM

Kodewerk
Java™ Performance Services

Java and system level locks

Kodewerk
Java^tm Performance Services

```
StringBuffer sb = new StringBuffer();
sb.append( “a”);
sb.append(“b”);
sb.append(“c”);
...
```

Lock Coarsening

Kodewerk
Java™ Performance Services

```
StringBuffer sb = new StringBuffer();
sb.append( “a”);
sb.append(“b”);
sb.append(“c”);
...
```

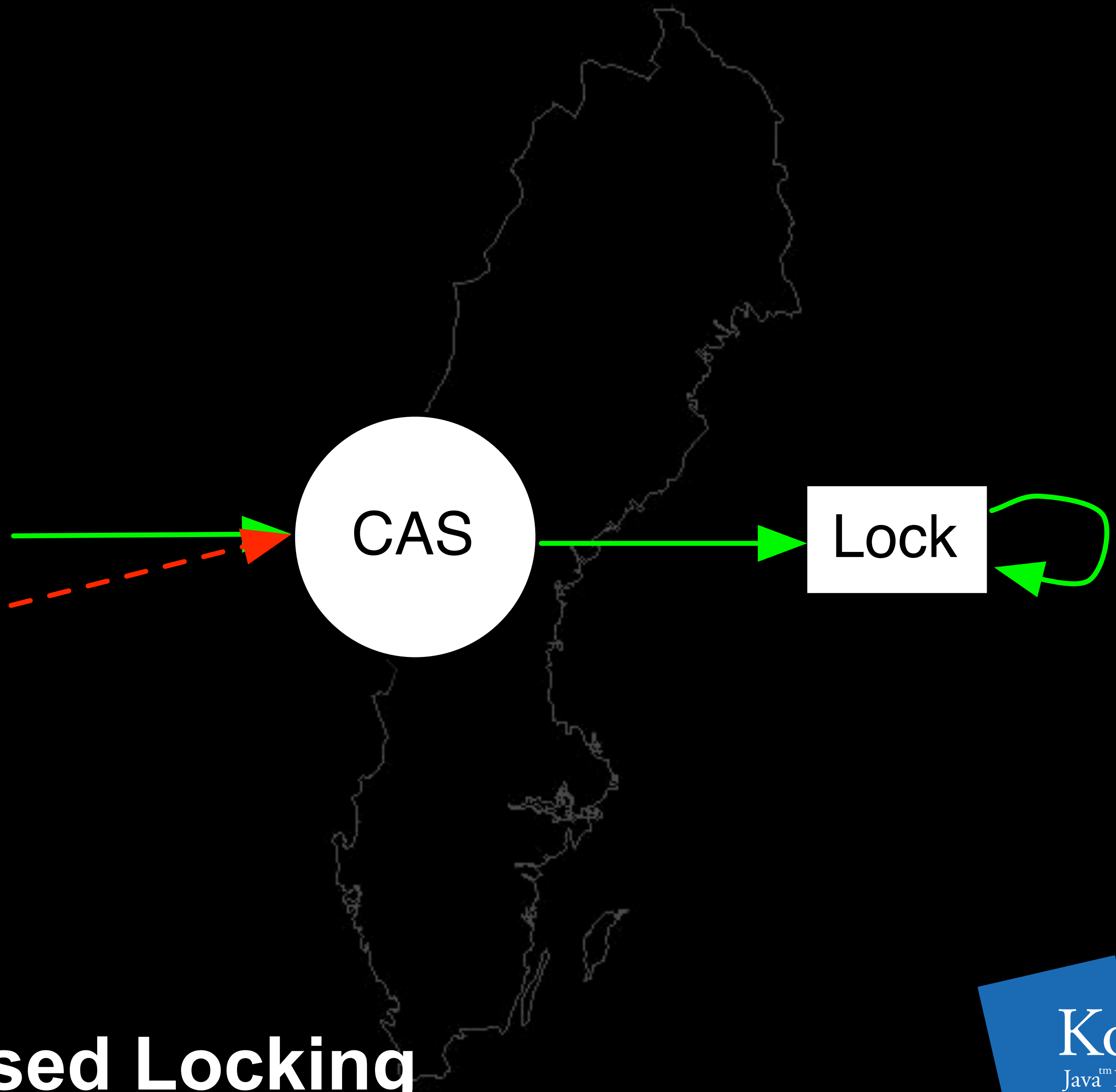
Lock Coarsening

Kodewerk
Java™ Performance Services

```
{
    StringBuffer sb = new StringBuffer();
    sb.append( “a”);
    sb.append(“b”);
    sb.append(“c”);
}
```

Lock Elision

Kodewerk
Java™ Performance Services

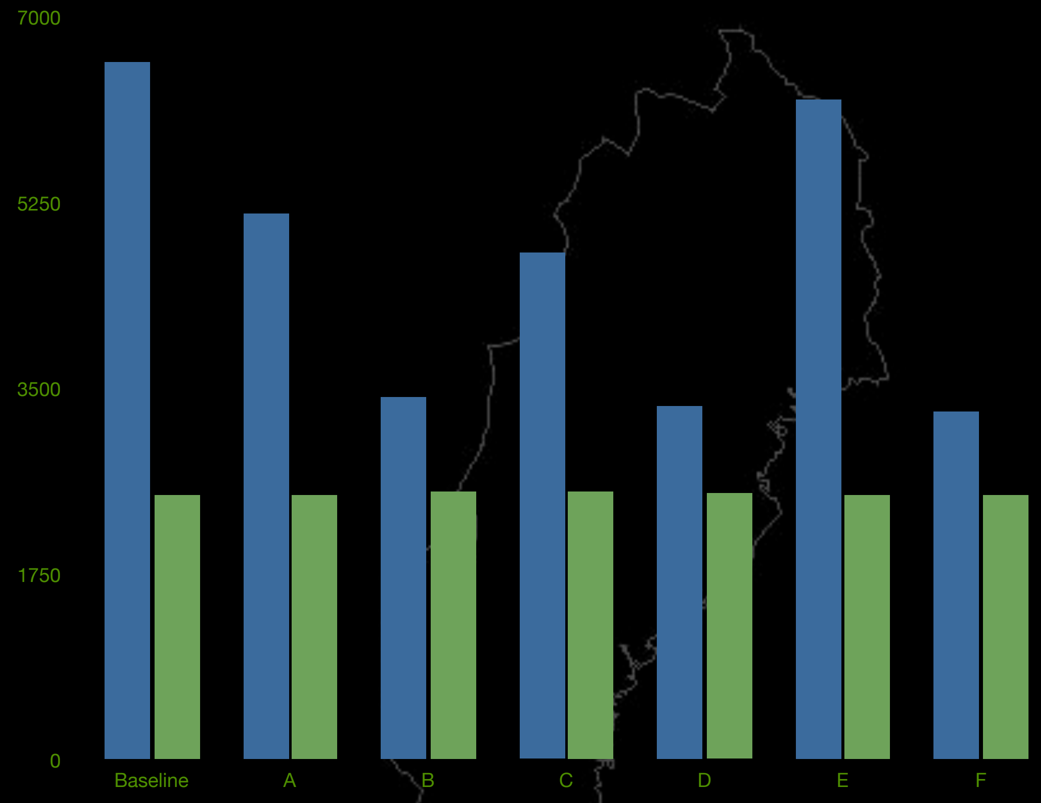


Biased Locking

Kodewerk
Java™ Performance Services

do these optimizations work?

Kodewerk
Java™ Performance Services



A EliminateLocks
B UseBiasedLocking (working)
C UseBiasedLocking (not working)
D EliminateLocks with UseBiasedLocking
E DoEscapeAnalysis
F EliminateLocks with UseBiasedLocking and DoEscapeAnalysis

🟦 StringBuffer 🟩 StringBuilder

Kodewerk
Java™ Performance Services

techniques we can use

Kodewerk
Java^{tm} Performance Services

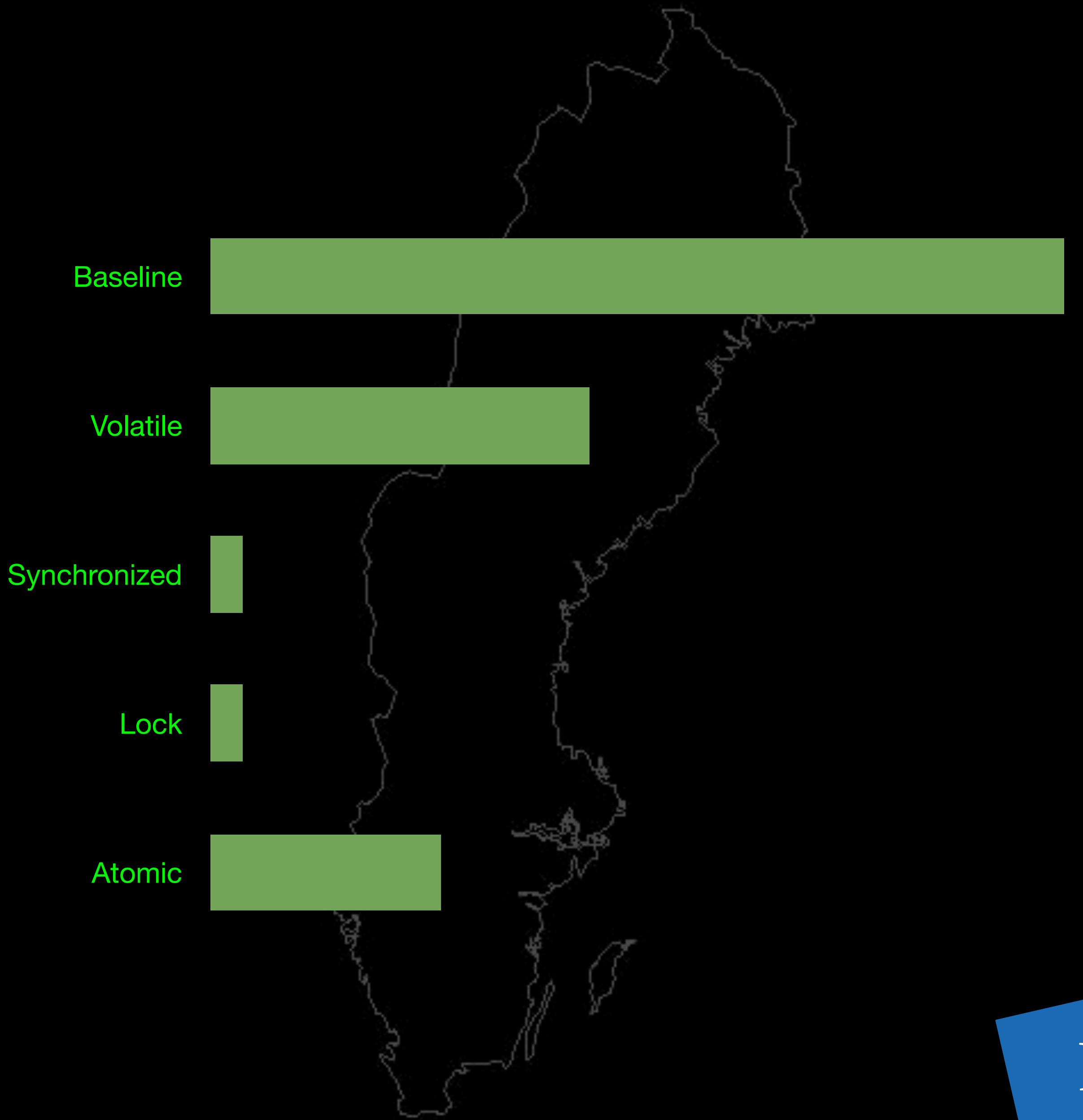
Atomics to reduce lock contention

Kodewerk
Java™ Performance Services

```
private int counter = 0;

Runnable mutator = new Runnable() {
    public void run() {
        long localCount = 0;
        while ( running) {
            counter++;
            counter--;
            localCount++;
        }
        addToTotalCount( localCount);
    }
};
```

Kodewerk
Java™ Performance Services

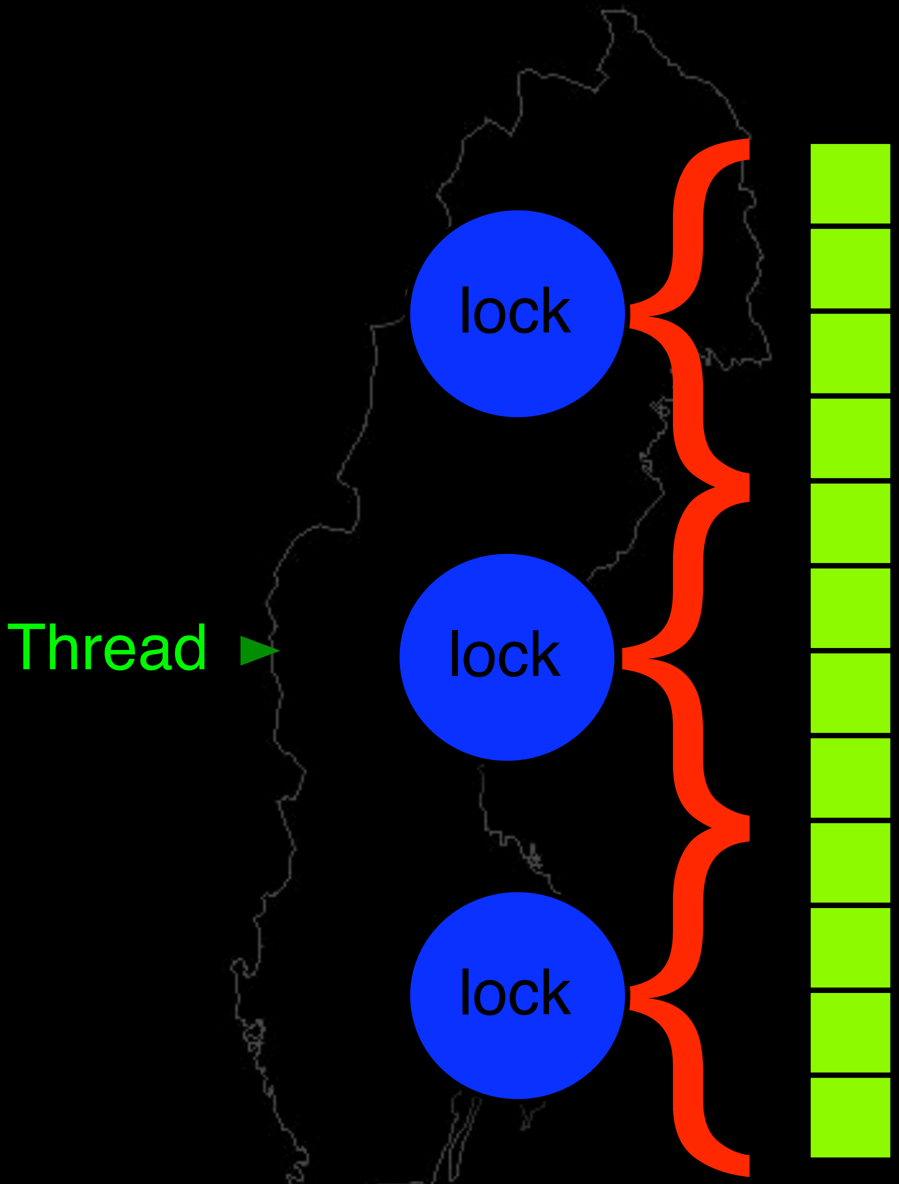


Kodewerk

Java™ Performance Services

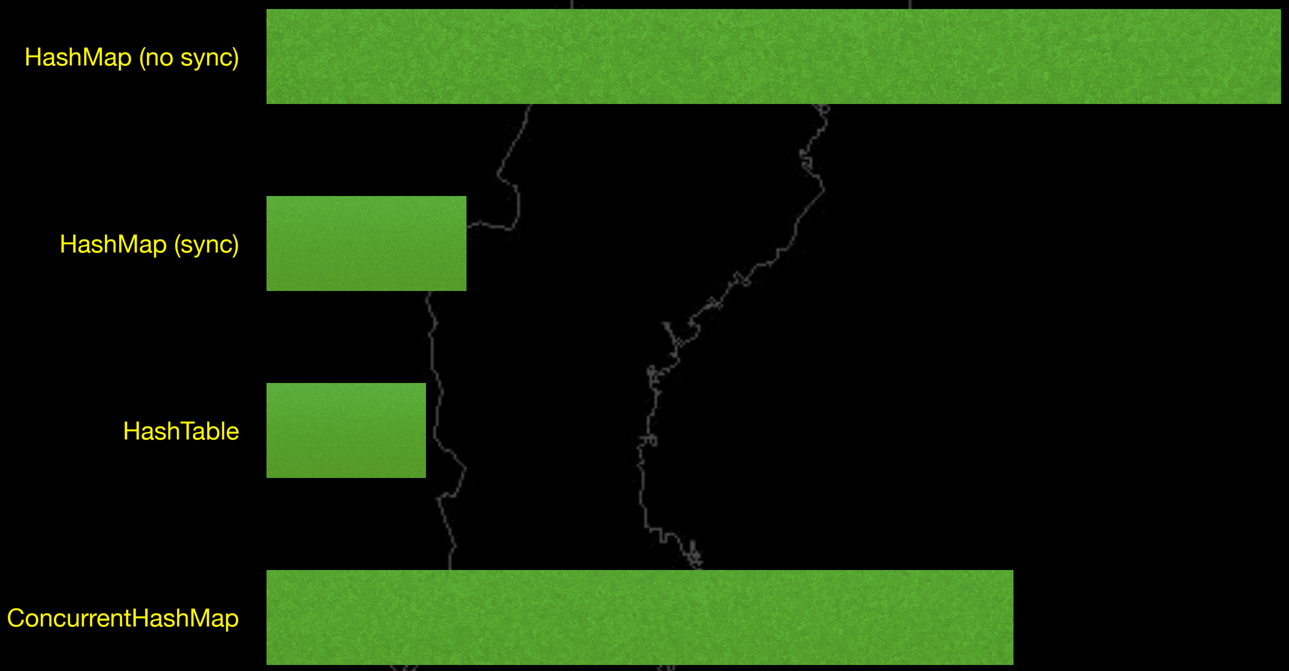
Lock striping

Kodewerk
Java™ Performance Services



ConcurrentHashMap

Kodewerk
Java™ Performance Services



Blackboard

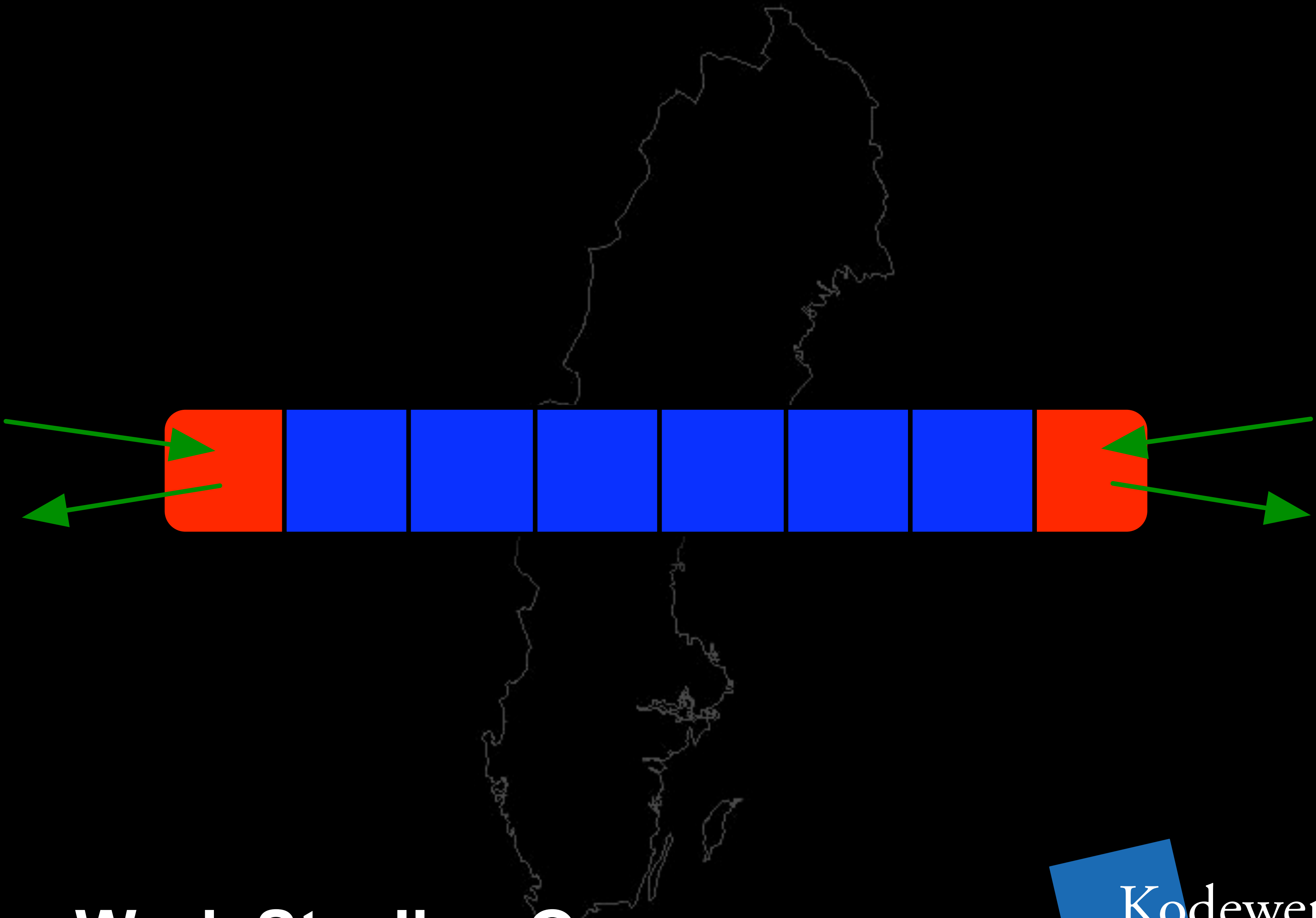
Kodewerk
Java™ Performance Services

teaching threads to steal

Kodewerk
Java™ Performance Services

Fork-Join

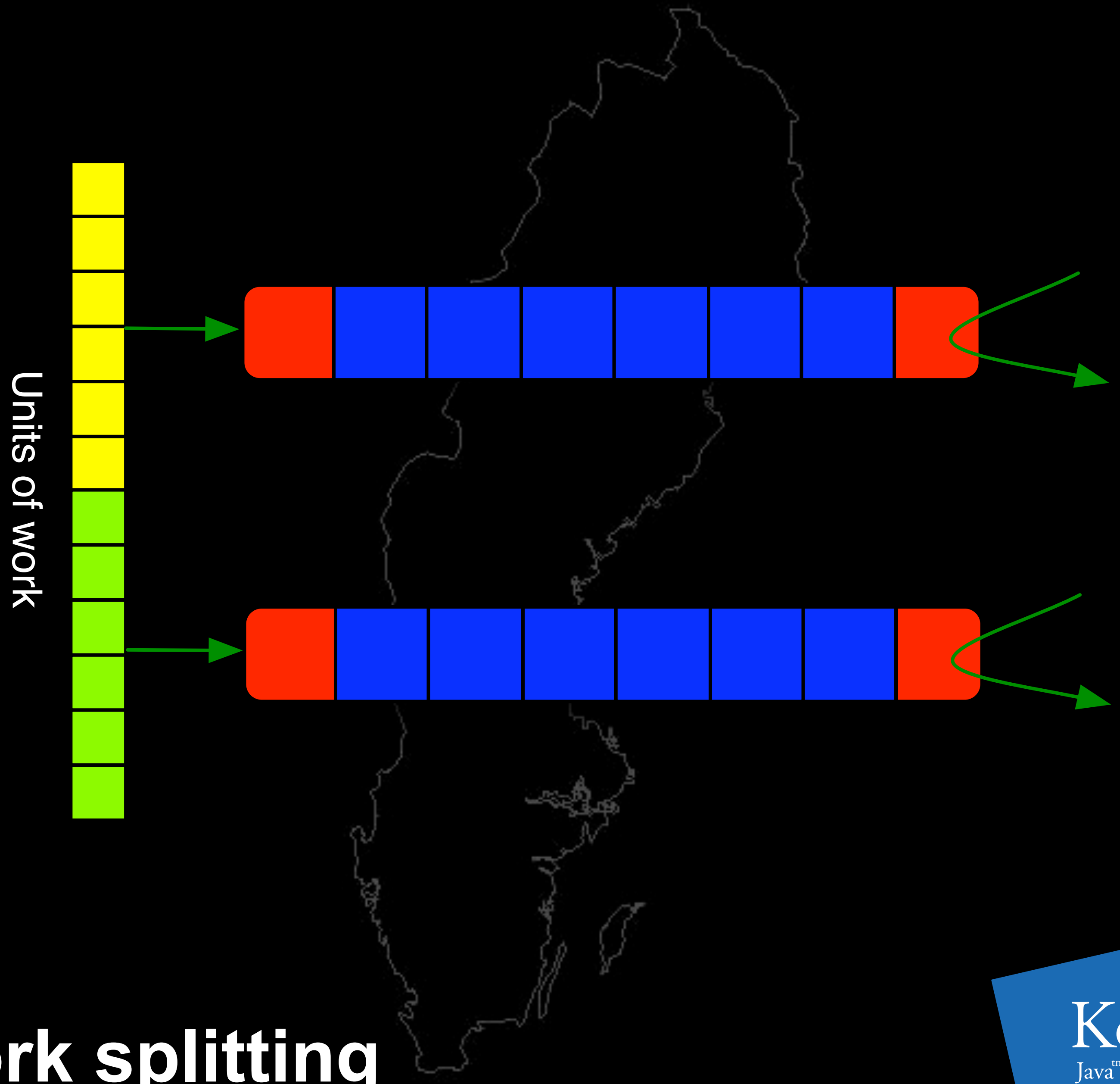
Kodewerk
Java™ Performance Services



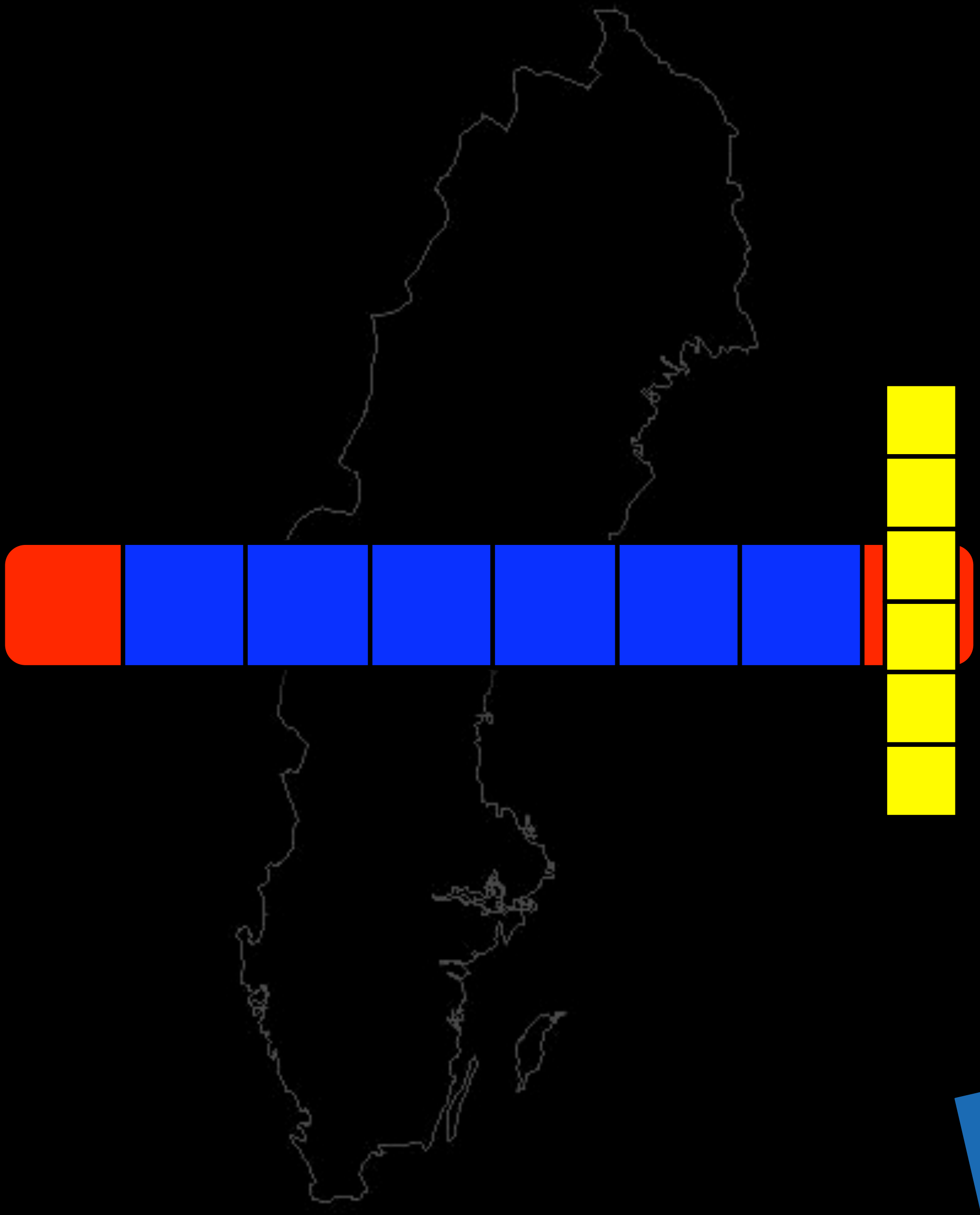
Work Stealing Queue

Kodewerk

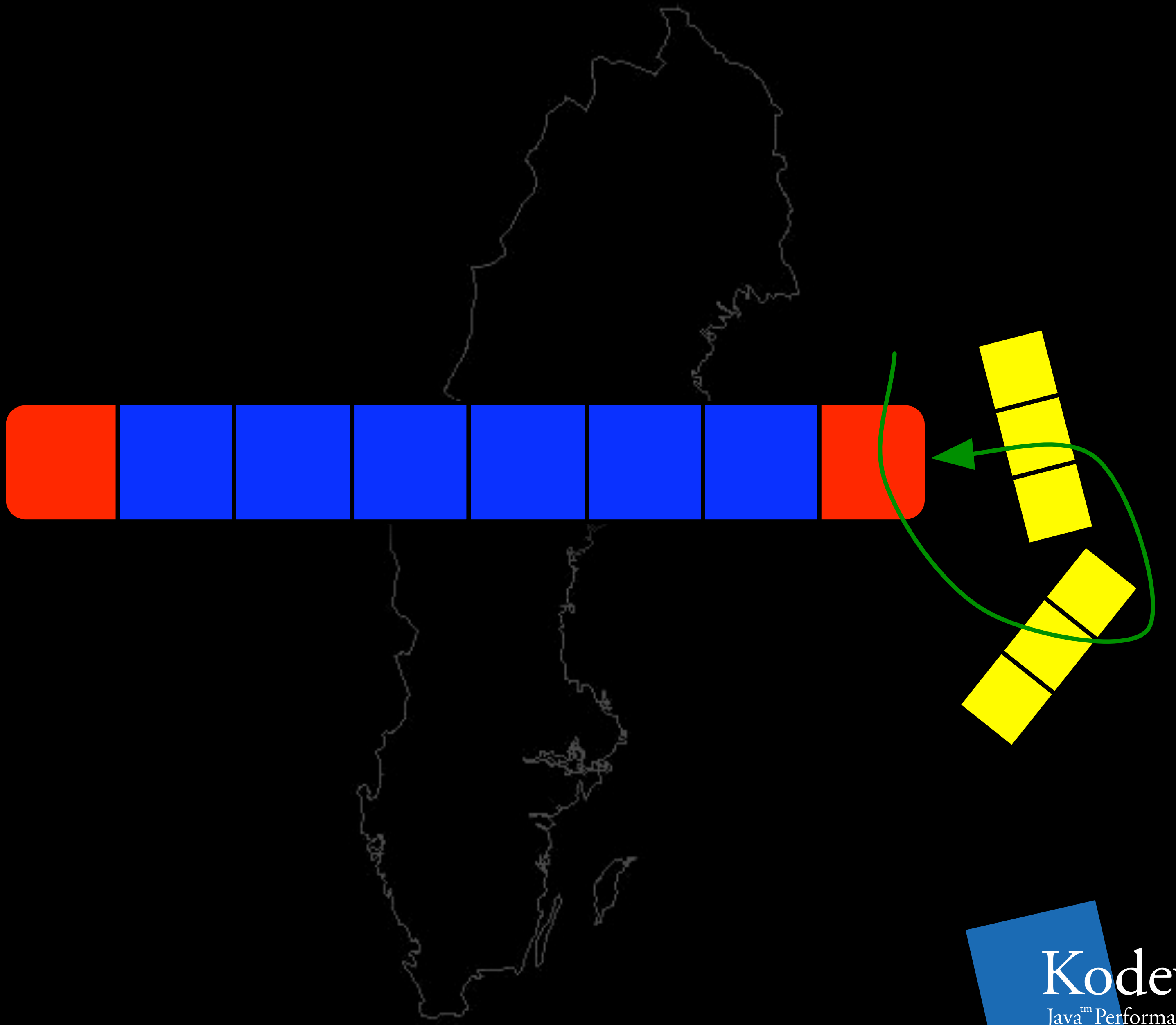
Java tm Performance Services



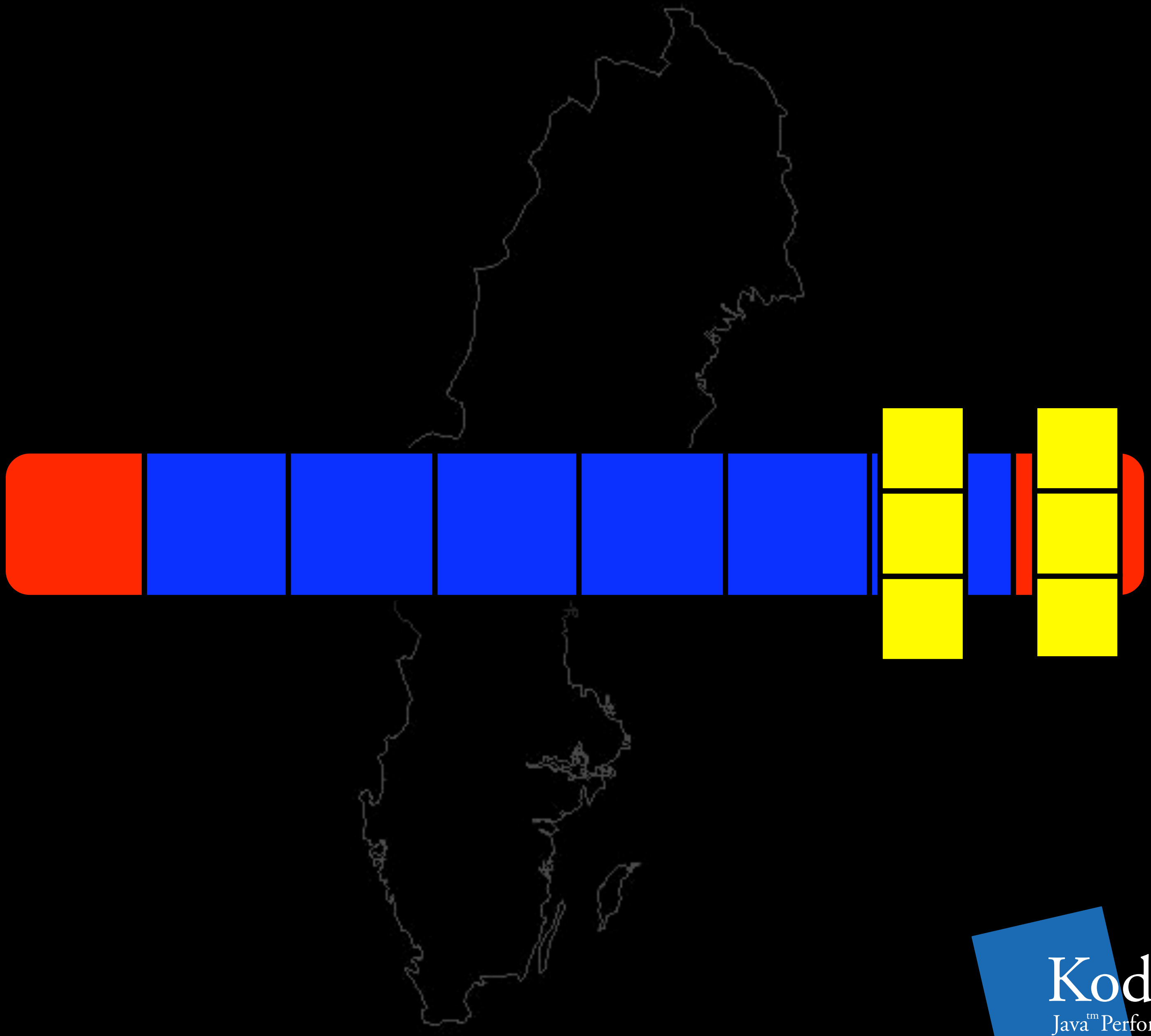
Work splitting



Kodewerk
Java™ Performance Services

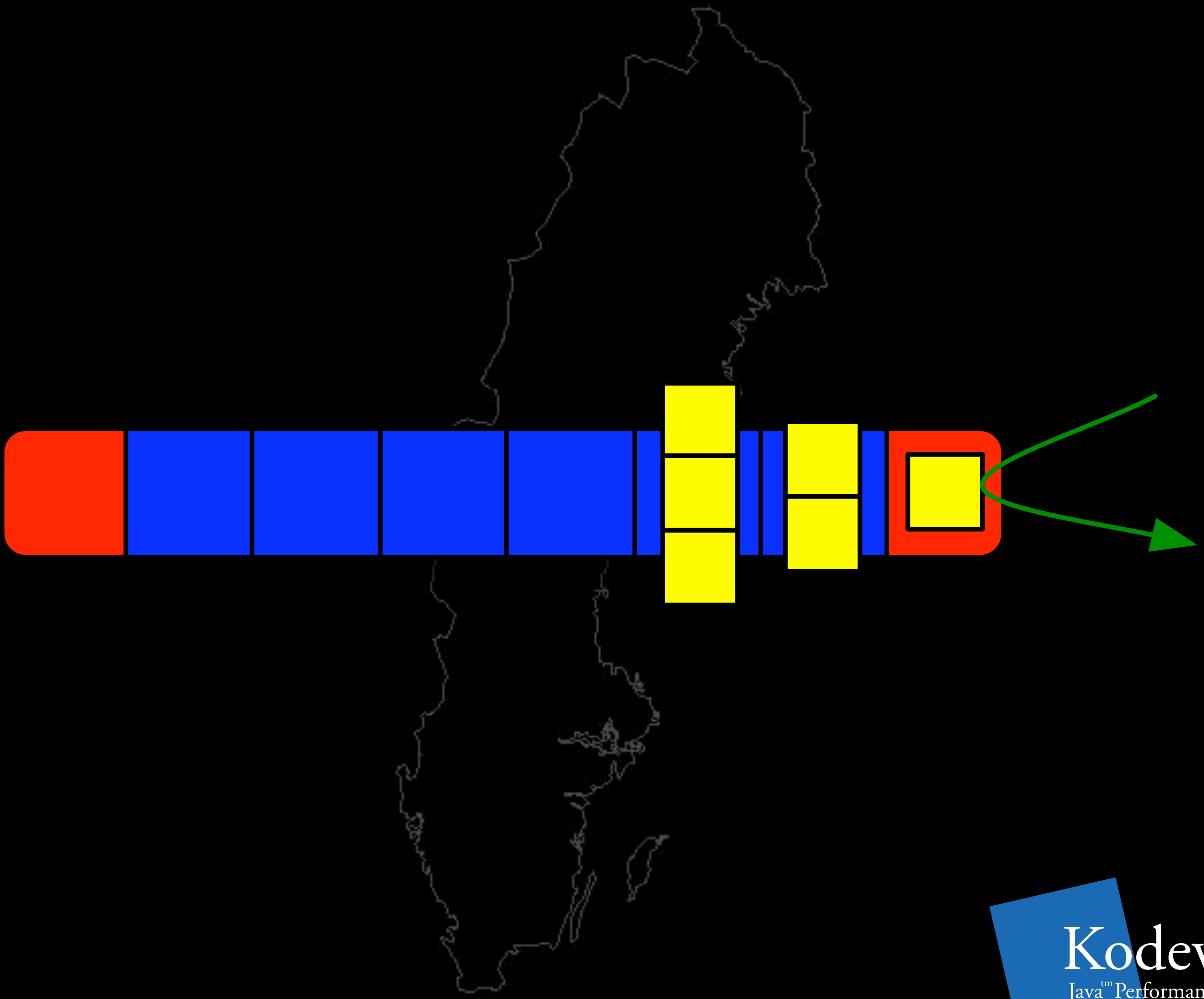


Kodewerk
Java™ Performance Services

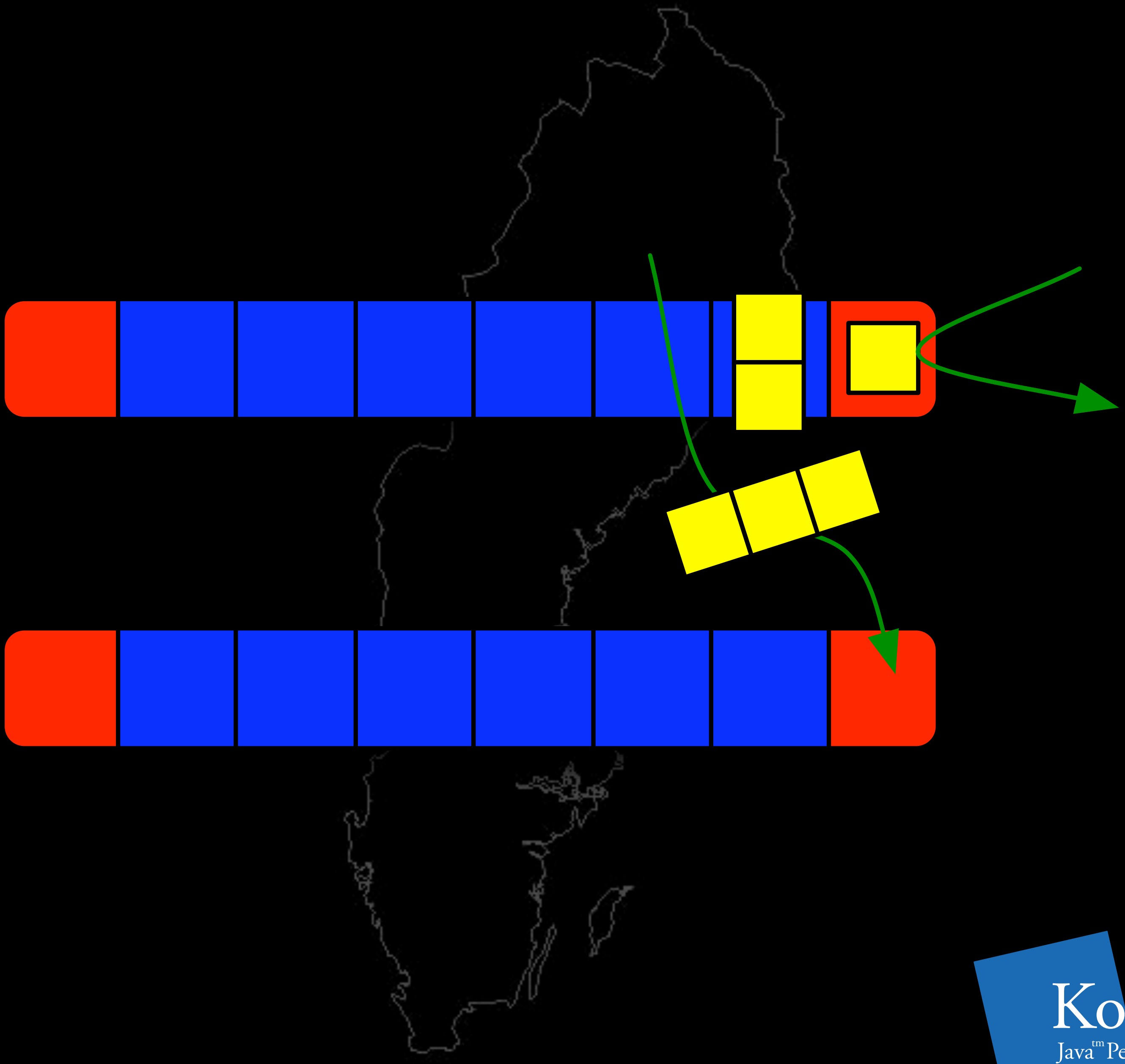


Kodewerk

Java^{tm} Performance Services



Kodewerk
Java tm Performance Services



Kodewerk

Java™ Performance Services

Degrees of Scalability

Kodewerk
Java™ Performance Services

Lock free concurrency

Kodewerk
Java™ Performance Services

Parallel reads, serialized writes

Kodewerk
Java™ Performance Services

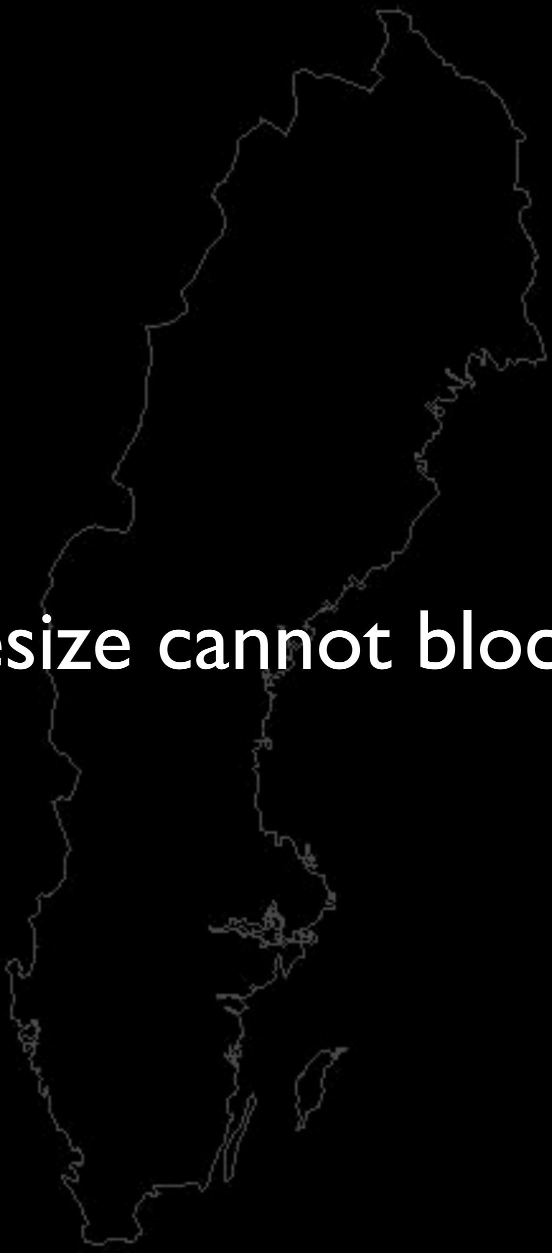
Reader/Writer lock with only readers will not scale beyond 100 cpus

large arrays for concurrent

Kodewerk
Javatm Performance Services

arrays to hold all data

Kodewerk
Java^{tm} Performance Services



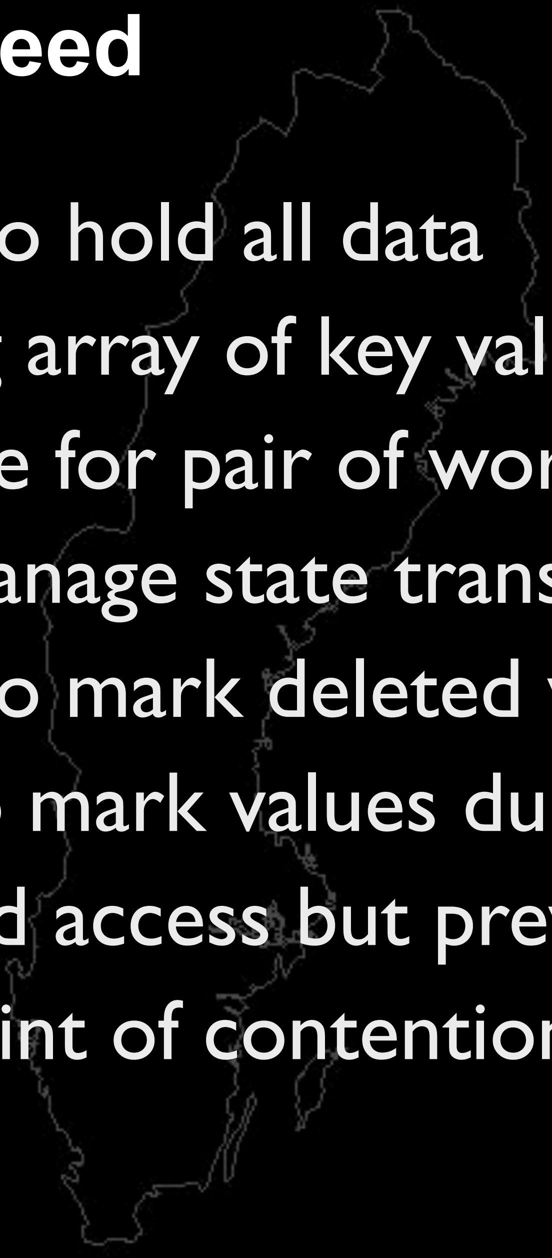
resize cannot block

Kodewerk
Java™ Performance Services

fully concurrent lock-less hashmap

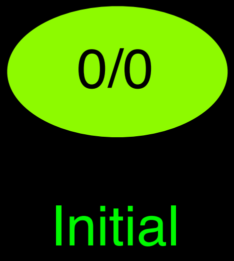
Kodewerk
Java tm Performance Services

Things we need



- Large array to hold all data
  - alternating array of key value pairs
- state machine for pair of words
  - CAS to manage state transistion
- Tombstone to mark deleted words
- Use a box to mark values during resize
  - allows read access but prevents update
- No single point of contention

Kodewerk
Java™ Performance Services



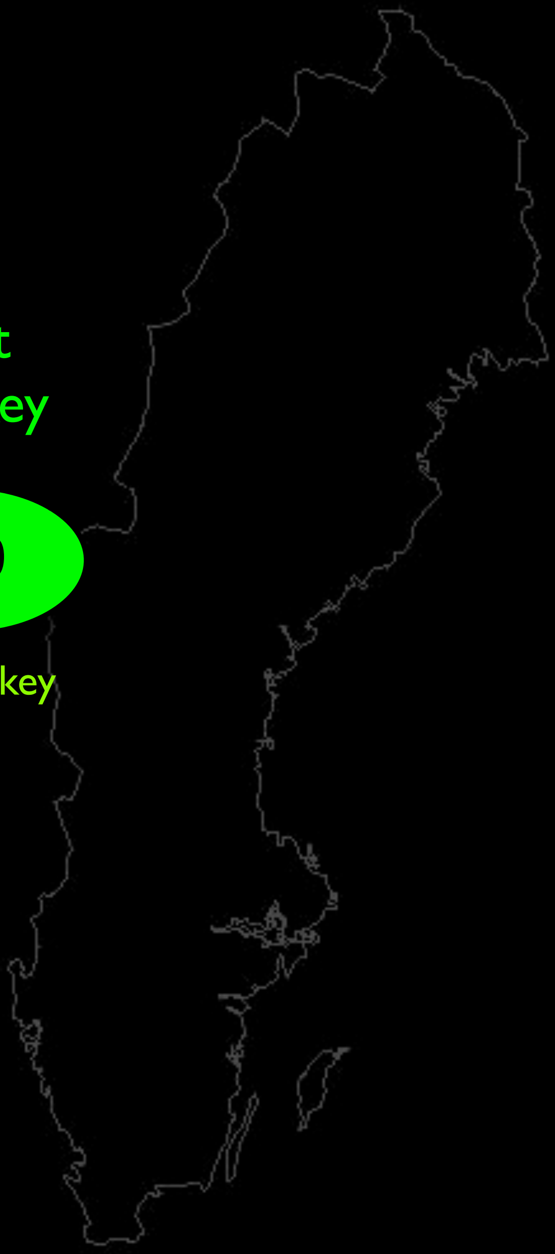
Inserting K/V pair
Already probed table, missed
Found proper empty K/V slot
Ready to claim slot for this Key

Kodewerk
Java™ Performance Services

claim slot
CAS in a key

$$\begin{tikzcd} 0/0 \arrow[r, bend right, "\text{insert key}"'] & K/0 \end{tikzcd}$$

naked key

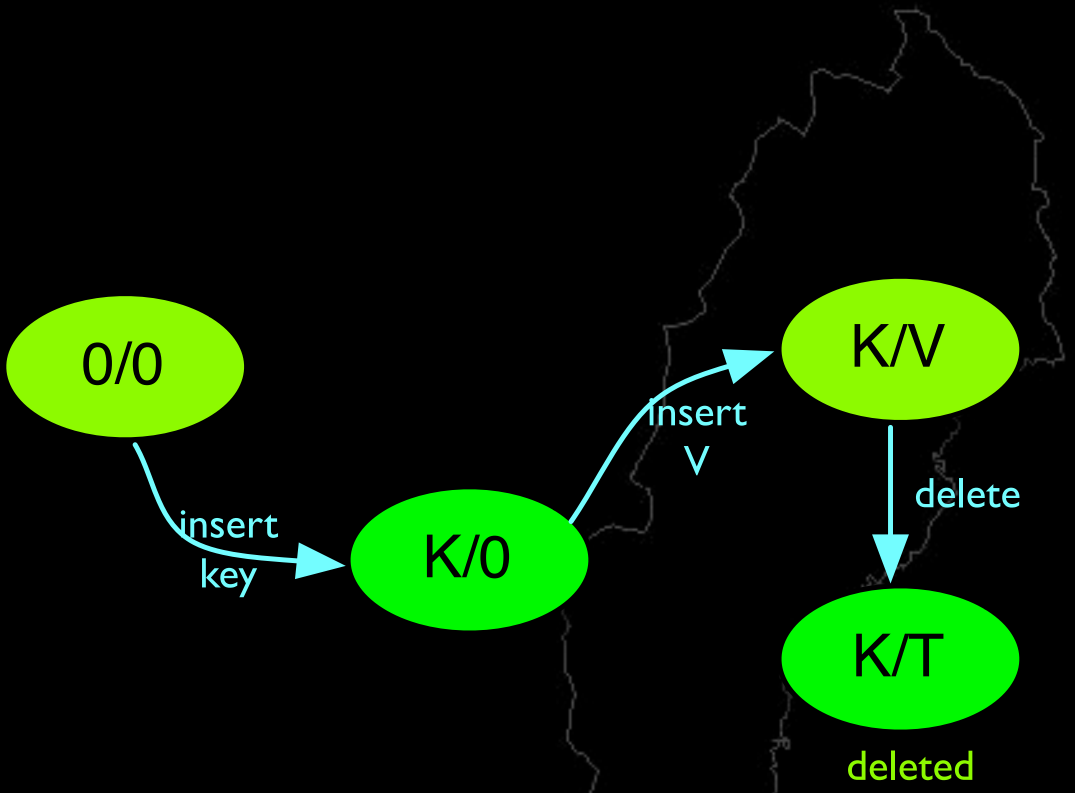


$$\begin{tikzcd}
0/0 \arrow[r, "\text{insert key}"'] & K/0 \arrow[r, "\text{insert V}"'] & K/V
\end{tikzcd}$$

active

initial set of value

Kodewerk
Java™ Performance Services



Tombstone marks delete, key remains

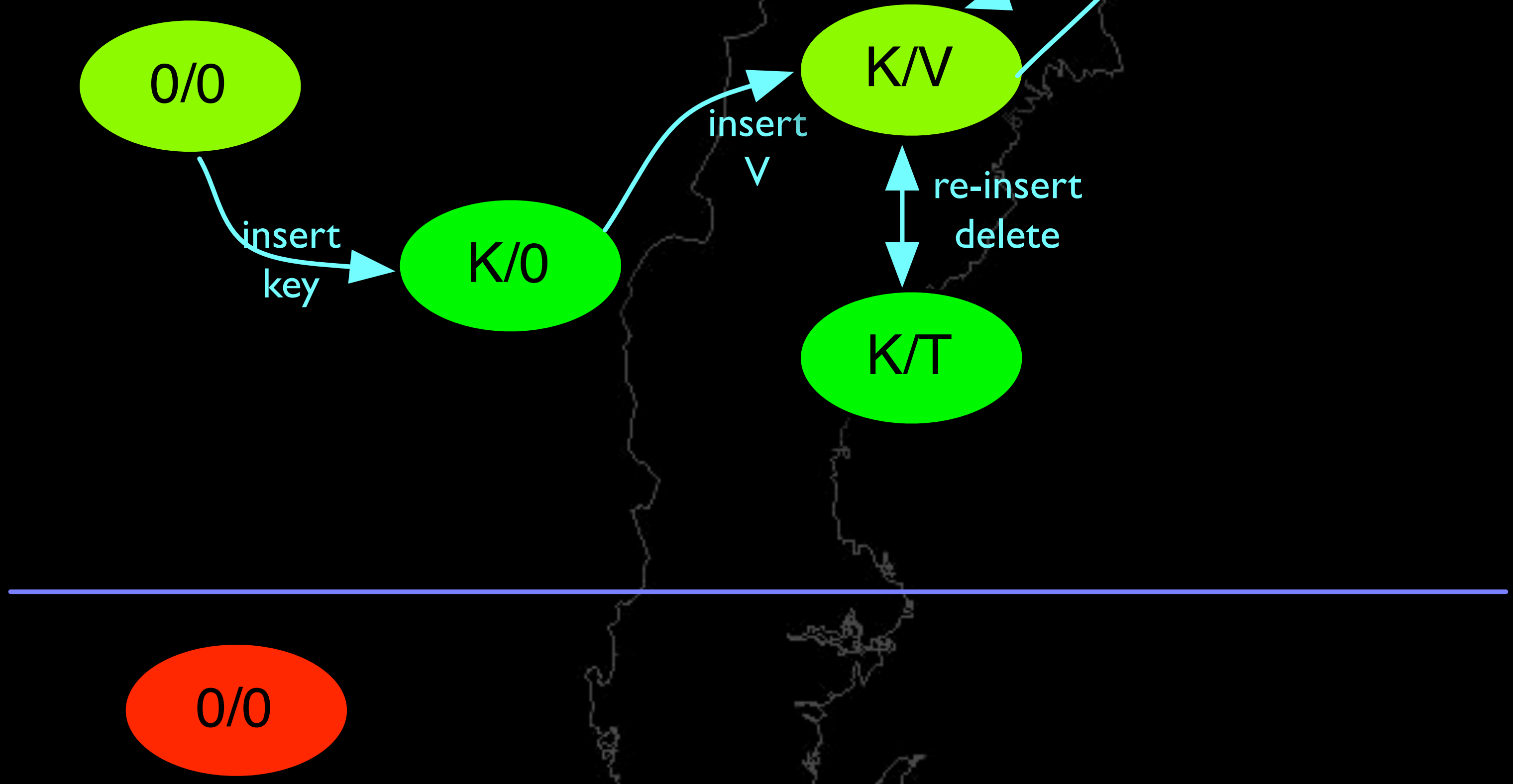
Kodewerk Java™ Performance Services

$$\begin{tikzcd}
0/0 \arrow[r, "\text{insert key}"'] & K/0 \arrow[r, "\text{insert V}"] & K/V \arrow[loop right, "\text{over-write V}"] \arrow[d, shift left, "\text{delete}"] \\
 & & K/T \arrow[u, shift left, "\text{re-insert}"]
\end{tikzcd}$$

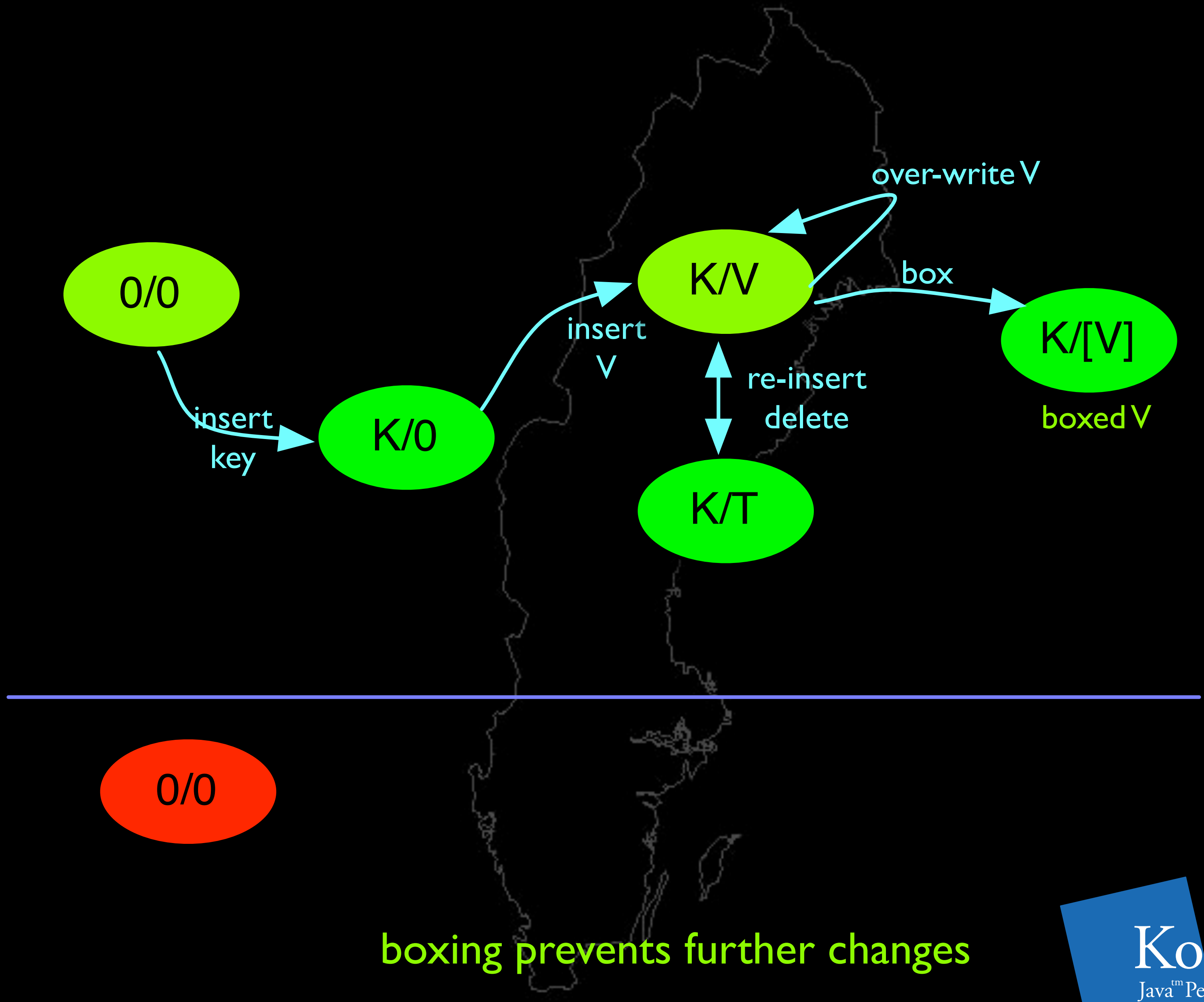
deleted

operations use same key slot
all operations operate normally

Kodewerk
Java^{tm} Performance Services

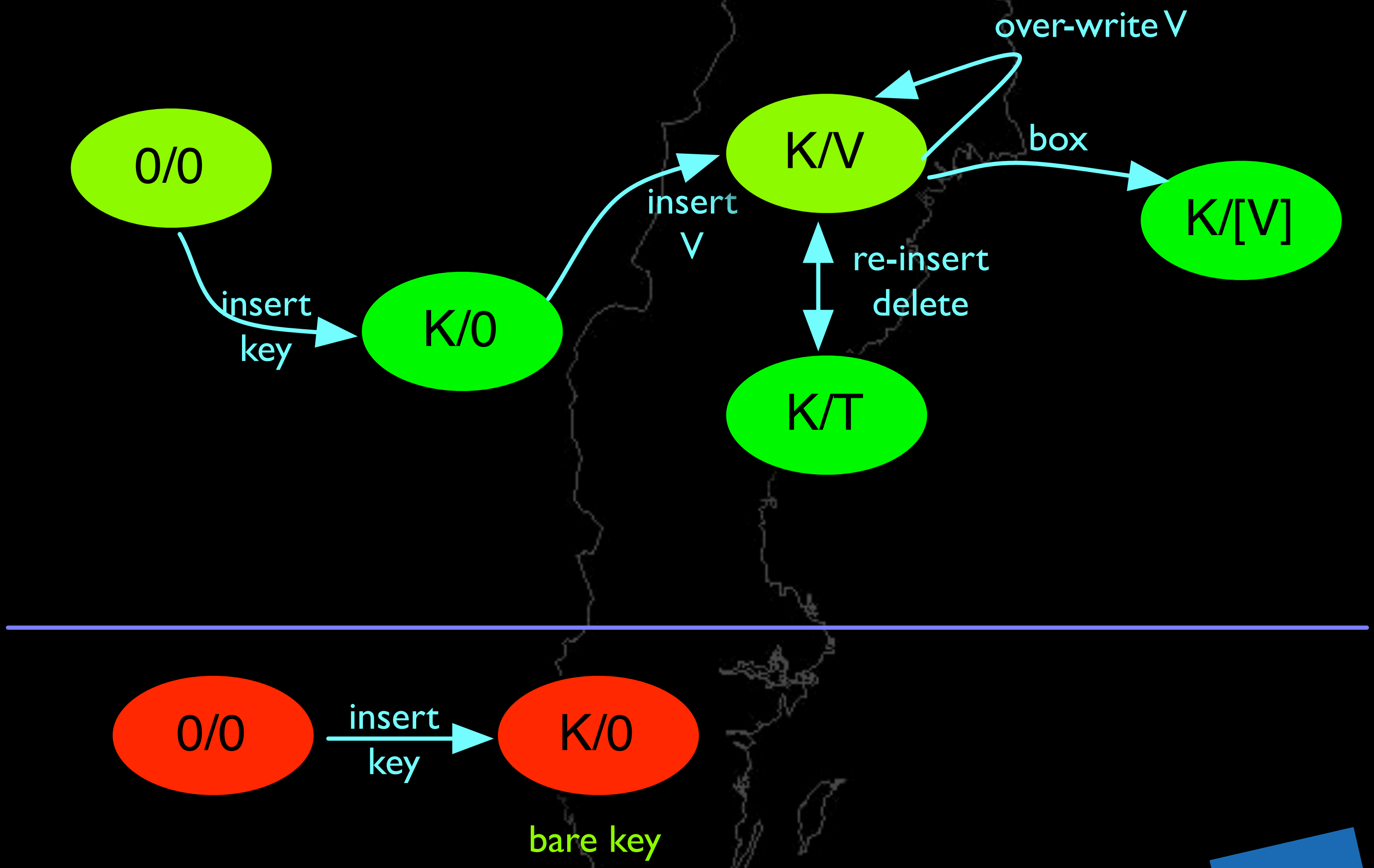


resize triggered
new array created
helper threads CAS a promise counter



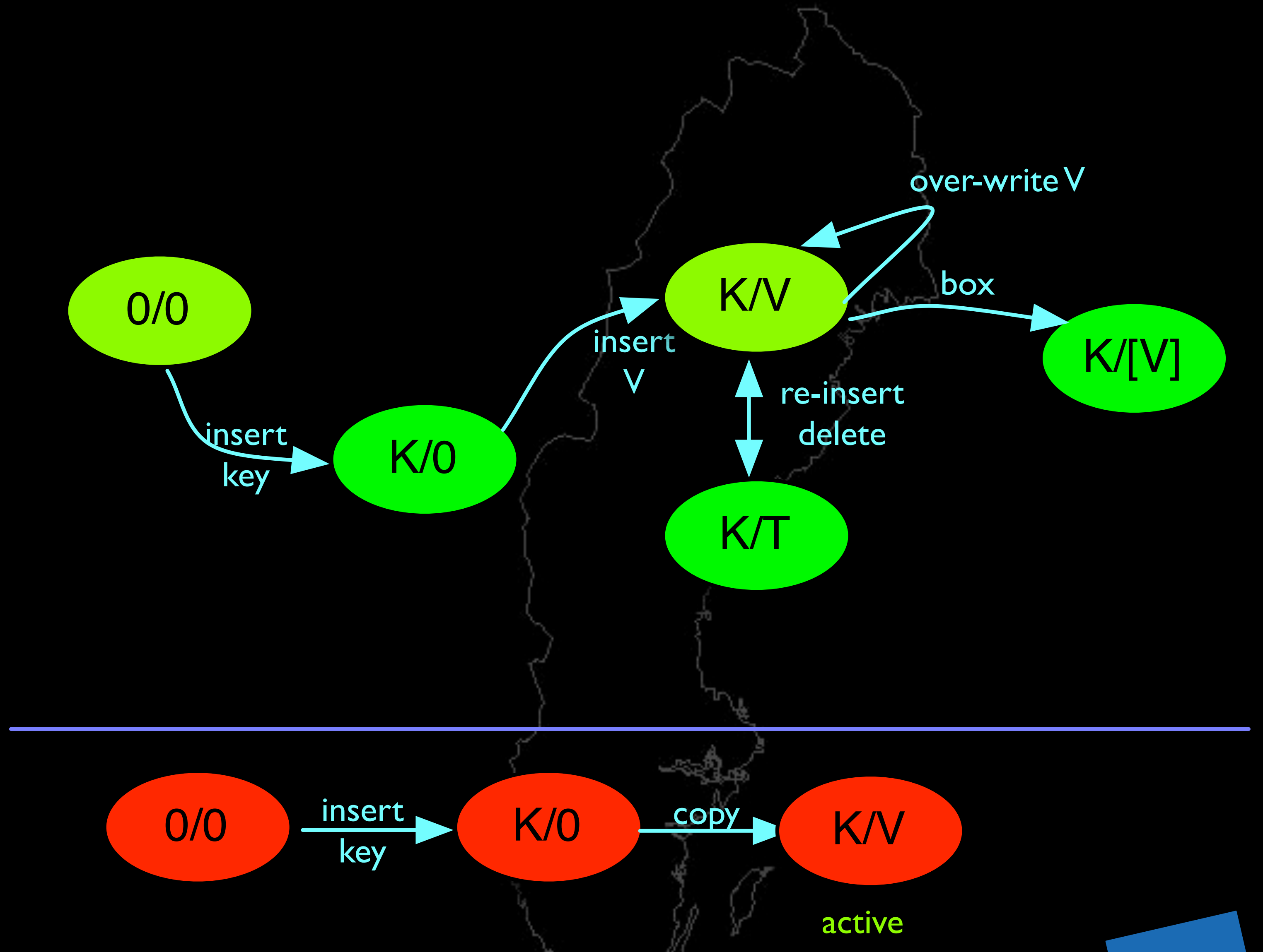
boxing prevents further changes

Kodewerk
Java™ Performance Services



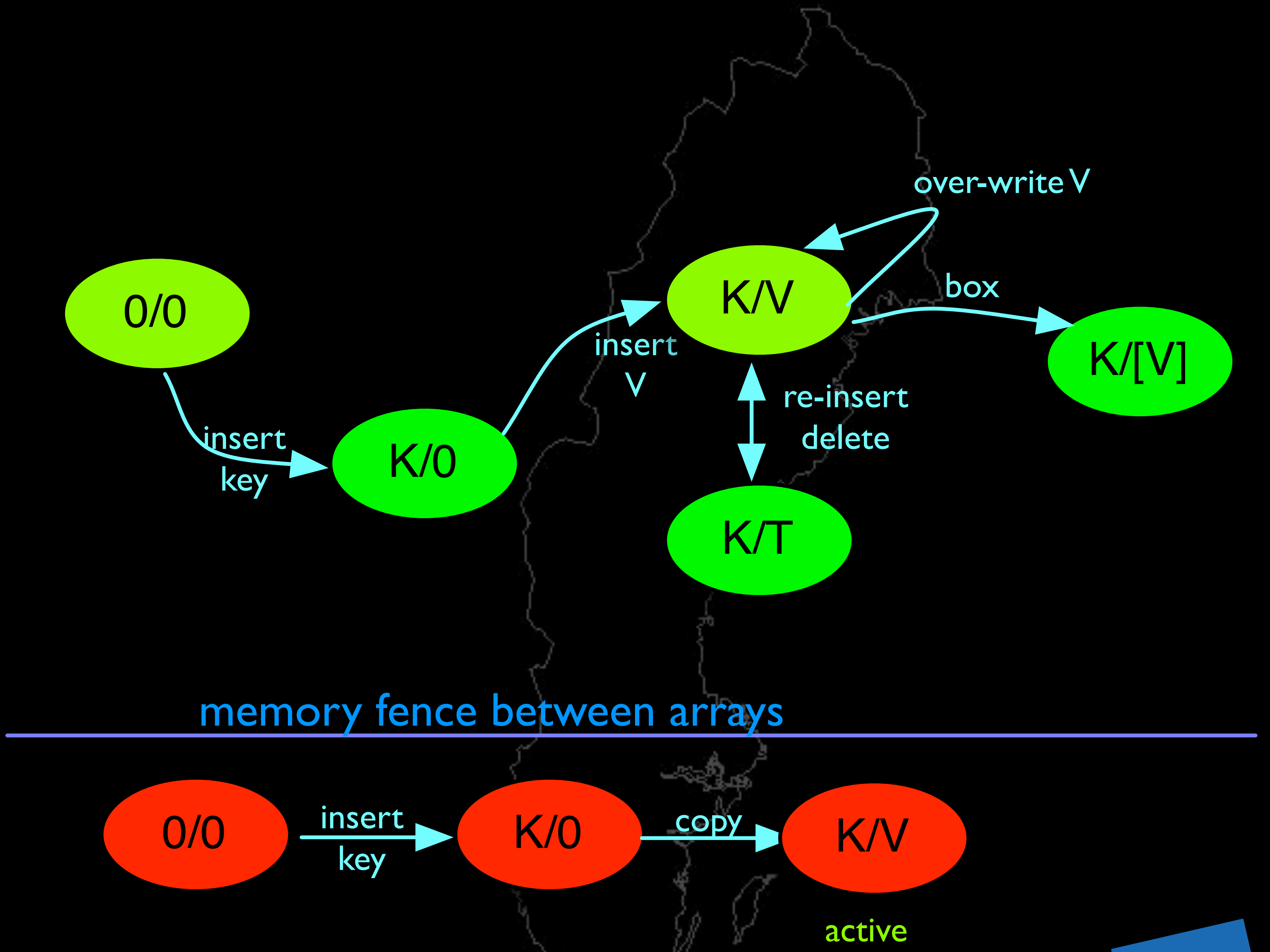
claim key slot in new table

Kodewerk
Java™ Performance Services



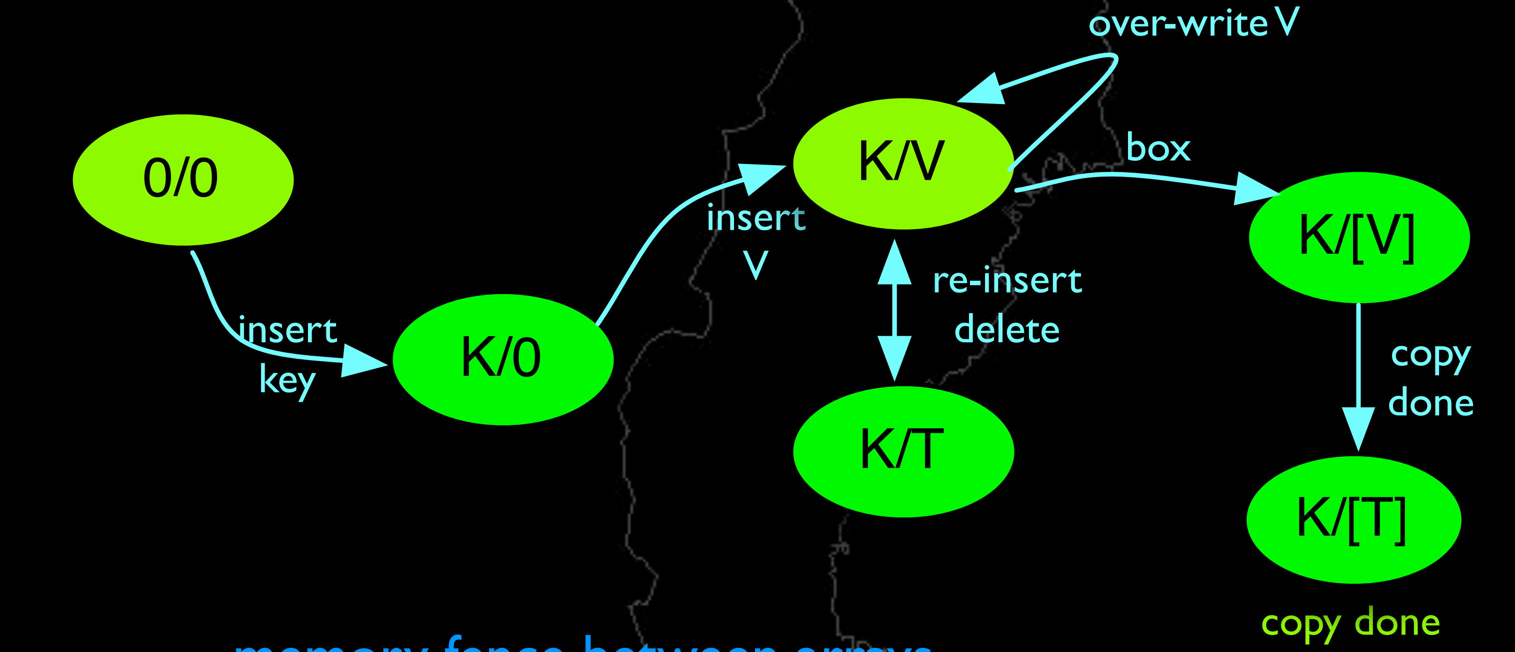
copy V without box

Kodewerk
Java™ Performance Services

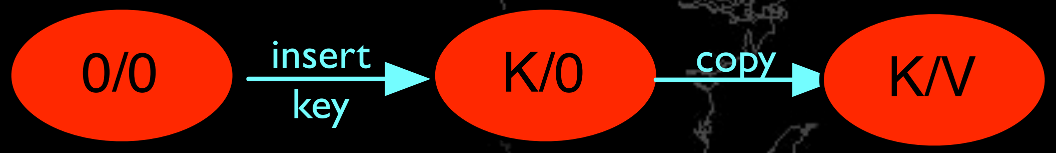


fence after writing new array and before copy done

Kodewerk
Java™ Performance Services

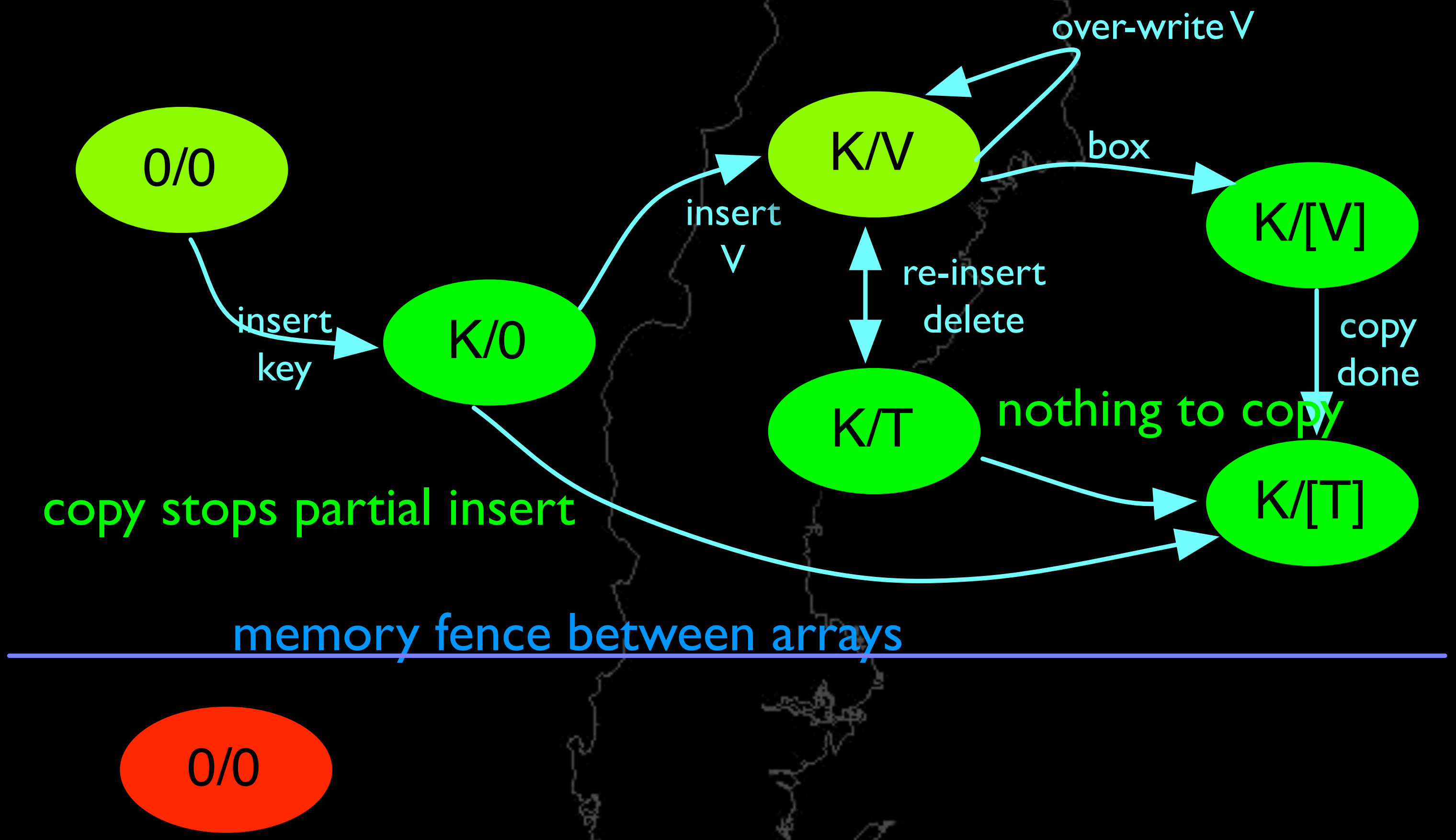


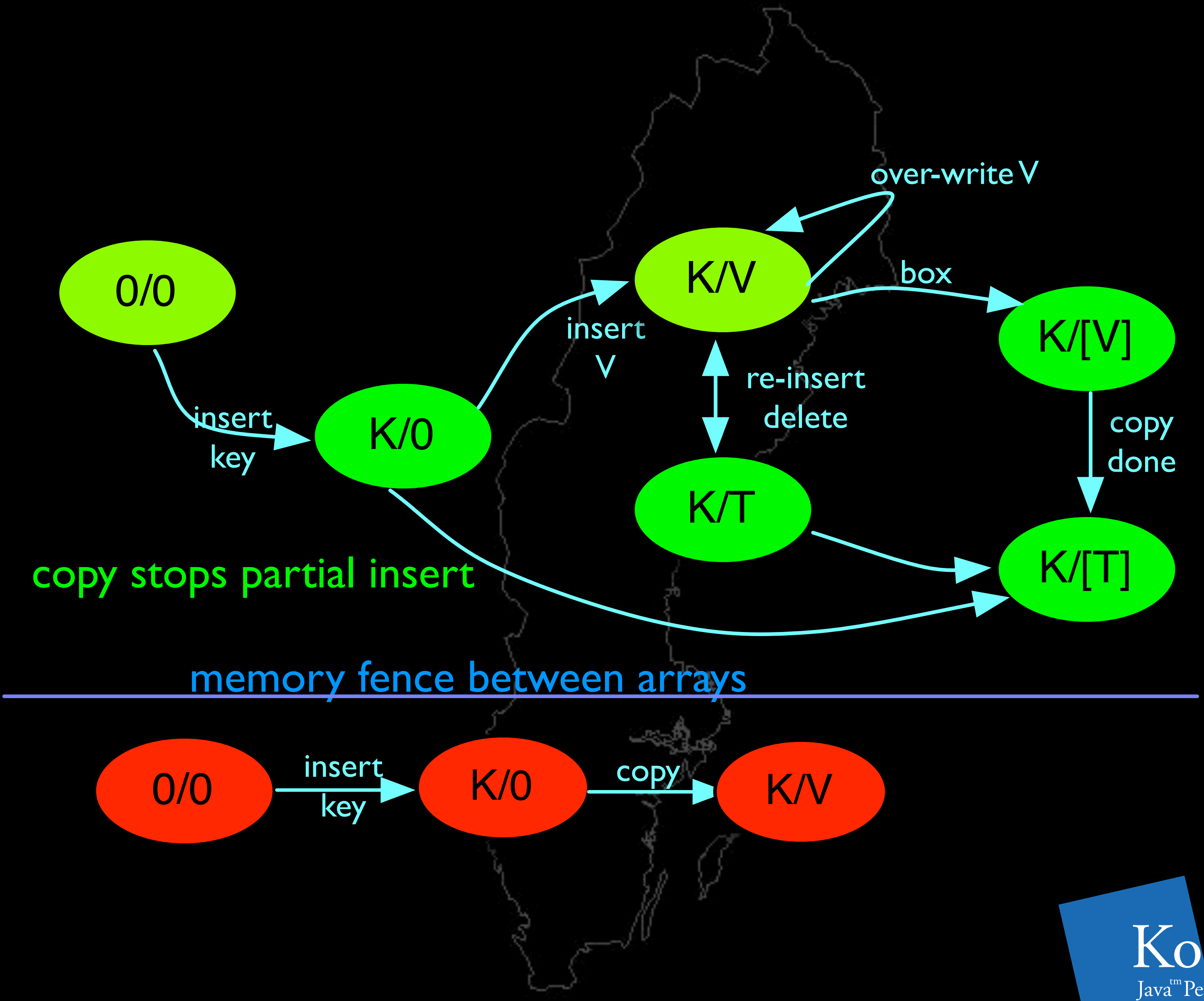
memory fence between arrays



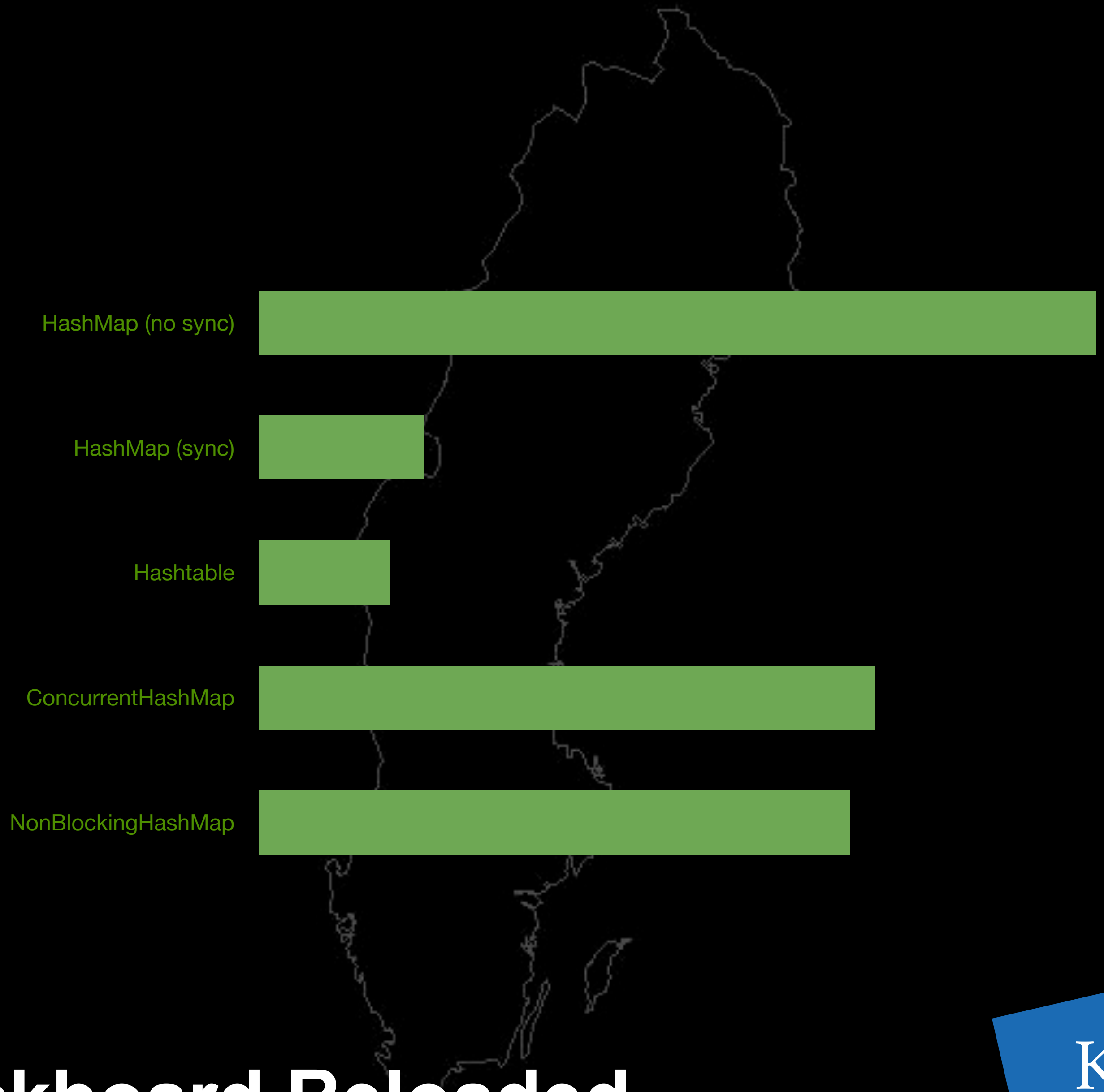
fence after writing new array and before copy done

Kodewerk
Java™ Performance Services


over-write V
0/0
K/V
box
insert V
K/[V]
re-insert delete
insert key
K/0
copy done
K/T
nothing to copy
copy stops partial insert
K/[T]
memory fence between arrays
0/0

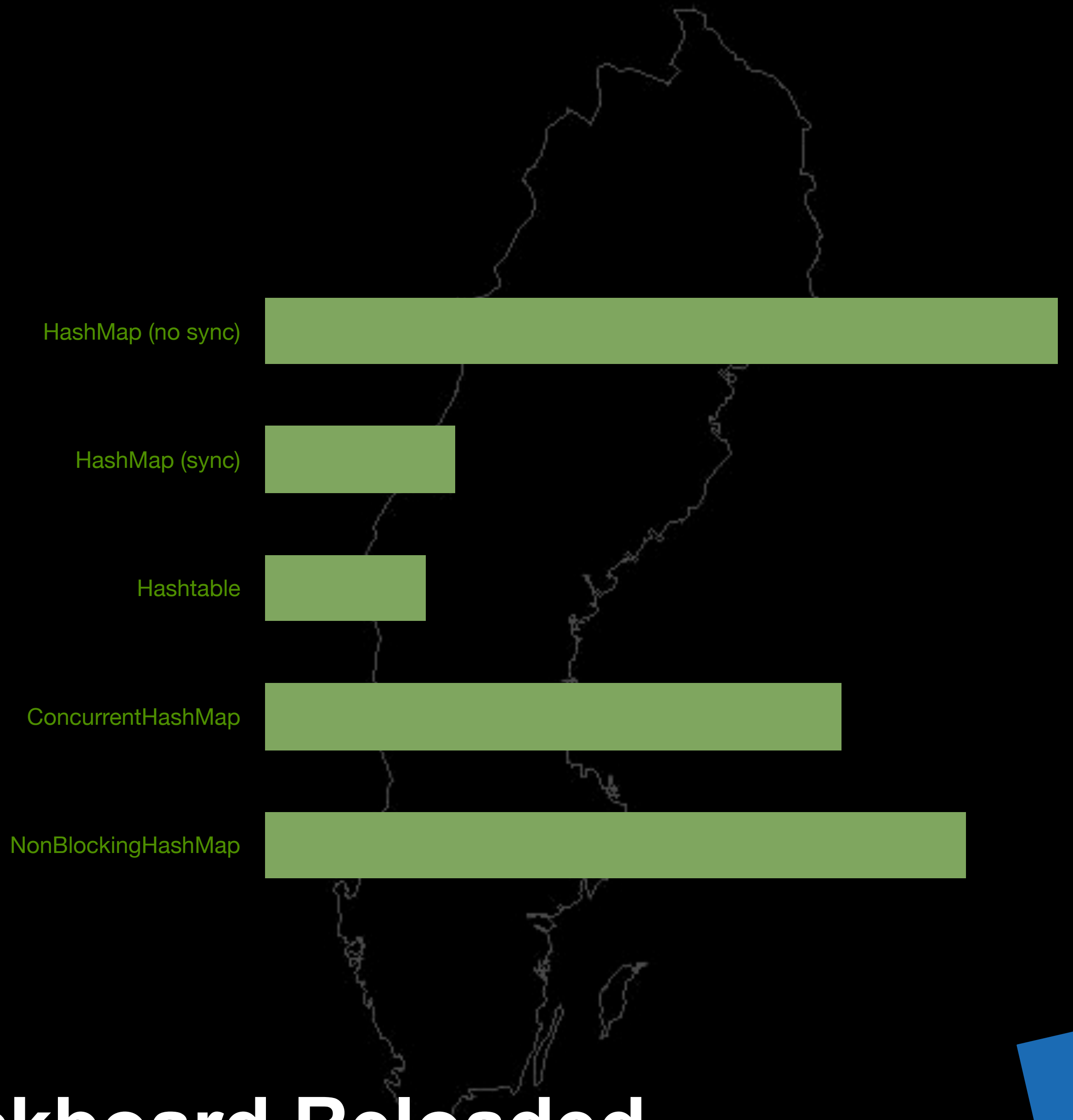


Kodewerk
Java™ Performance Services



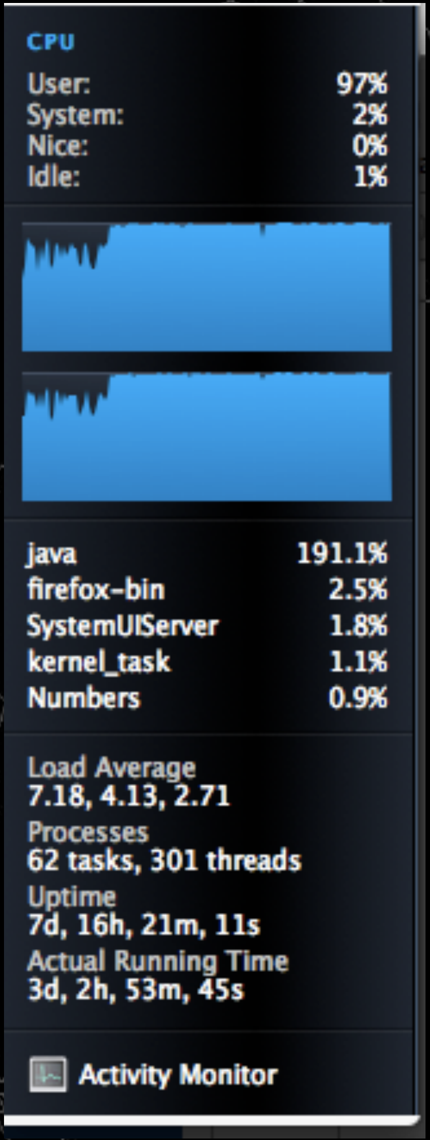
Blackboard Reloaded

Kodewerk
Java™ Performance Services



Blackboard Reloaded

Kodewerk
Java™ Performance Services



```
CPU
User:                97%
System:               2%
Nice:                 0%
Idle:                 1%

java             191.1%
firefox-bin        2.5%
SystemUIServer     1.8%
kernel_task        1.1%
Numbers            0.9%

Load Average
7.18, 4.13, 2.71
Processes
62 tasks, 301 threads
Uptime
7d, 16h, 21m, 11s
Actual Running Time
3d, 2h, 53m, 45s
```

Kodewerk
Java™ Performance Services

scales linerarly up too 1000 CPUs

Kodewerk
Java™ Performance Services

Fully concurrent lock-less FIFO?

Kodewerk
Java™ Performance Services

Stripe on queues and randomly pick one

Kodewerk
Java™ Performance Services

stripe ad-absurdum

Kodewerk
Java™ Performance Services

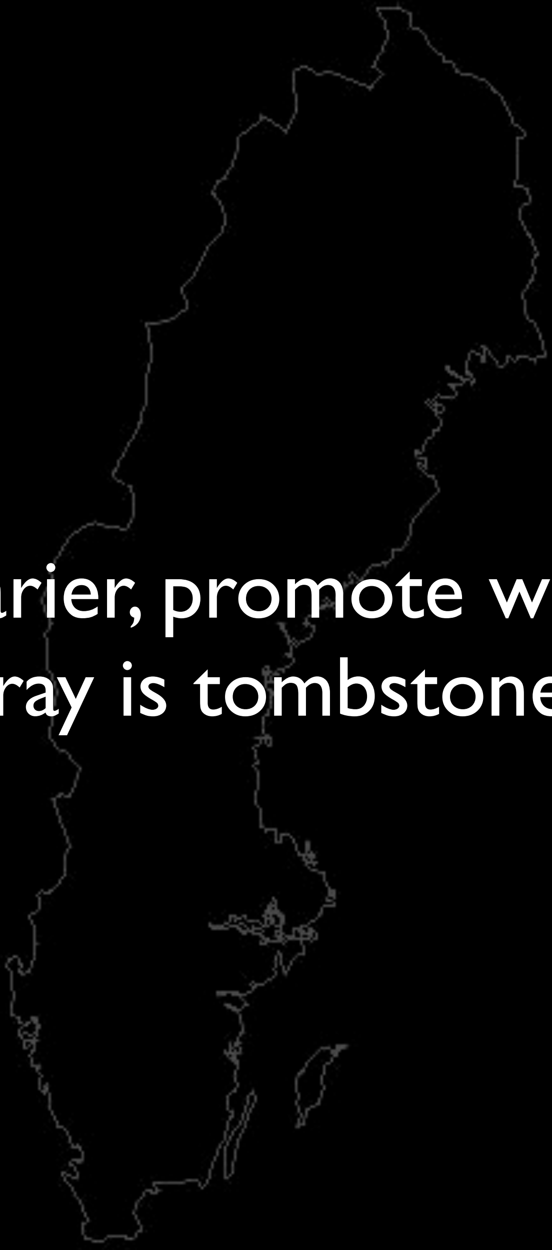
insert searches for null CAS down value

Kodewerk
Java™ Performance Services

read searches for value and CAS down null

too large read spin, too small inserts spin

Kodewerk
Java™ Performance Services



resize is earier, promote when entire array is tombstoned

Kodewerk

Java^{tm} Performance Services

Questions?

Kodewerk

Java™ Performance Services