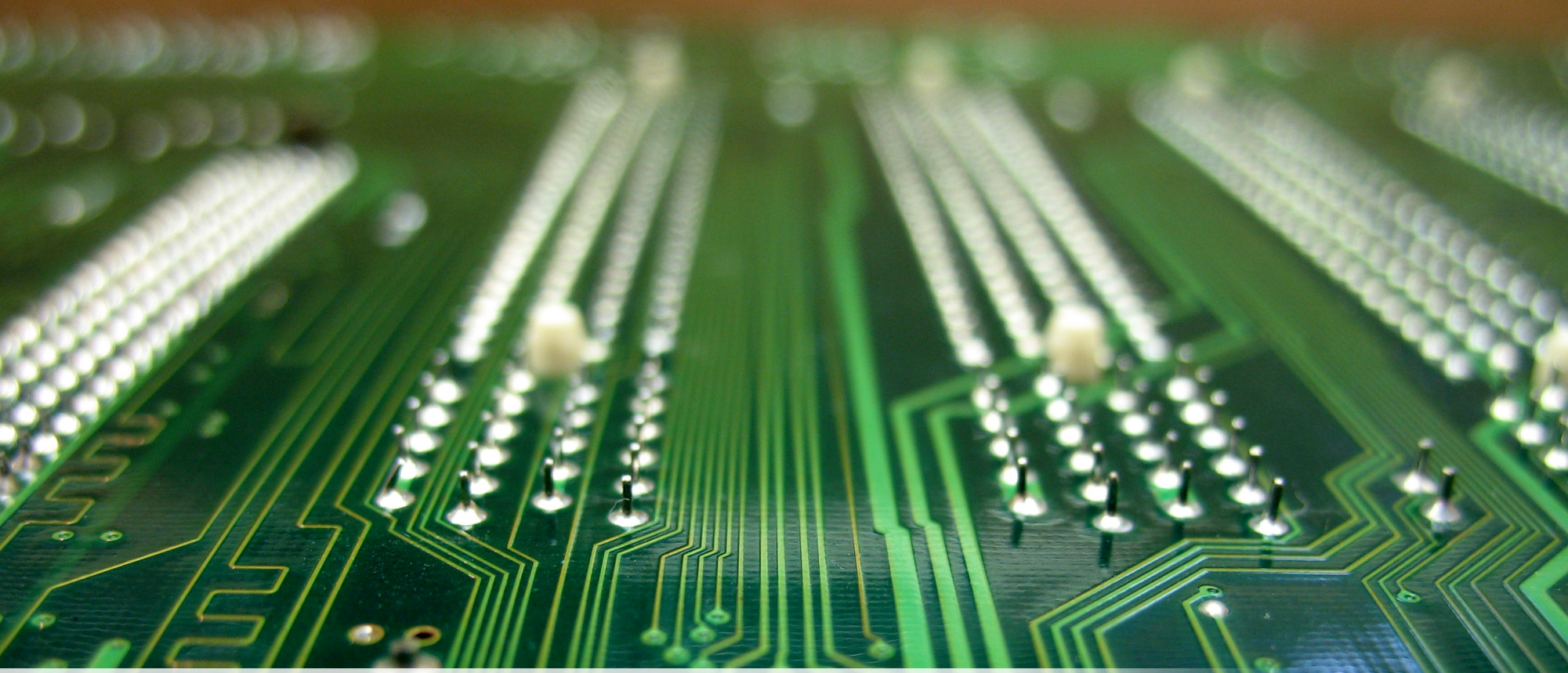


FPGA Acceleration for the Frequent Item Problem

Jens Teubner, René Müller, Gustavo Alonso
ETH Zurich, Systems Group

not (only) about FPGAs

not about a new solution

to the frequent item problem

Frequent Item Problem:

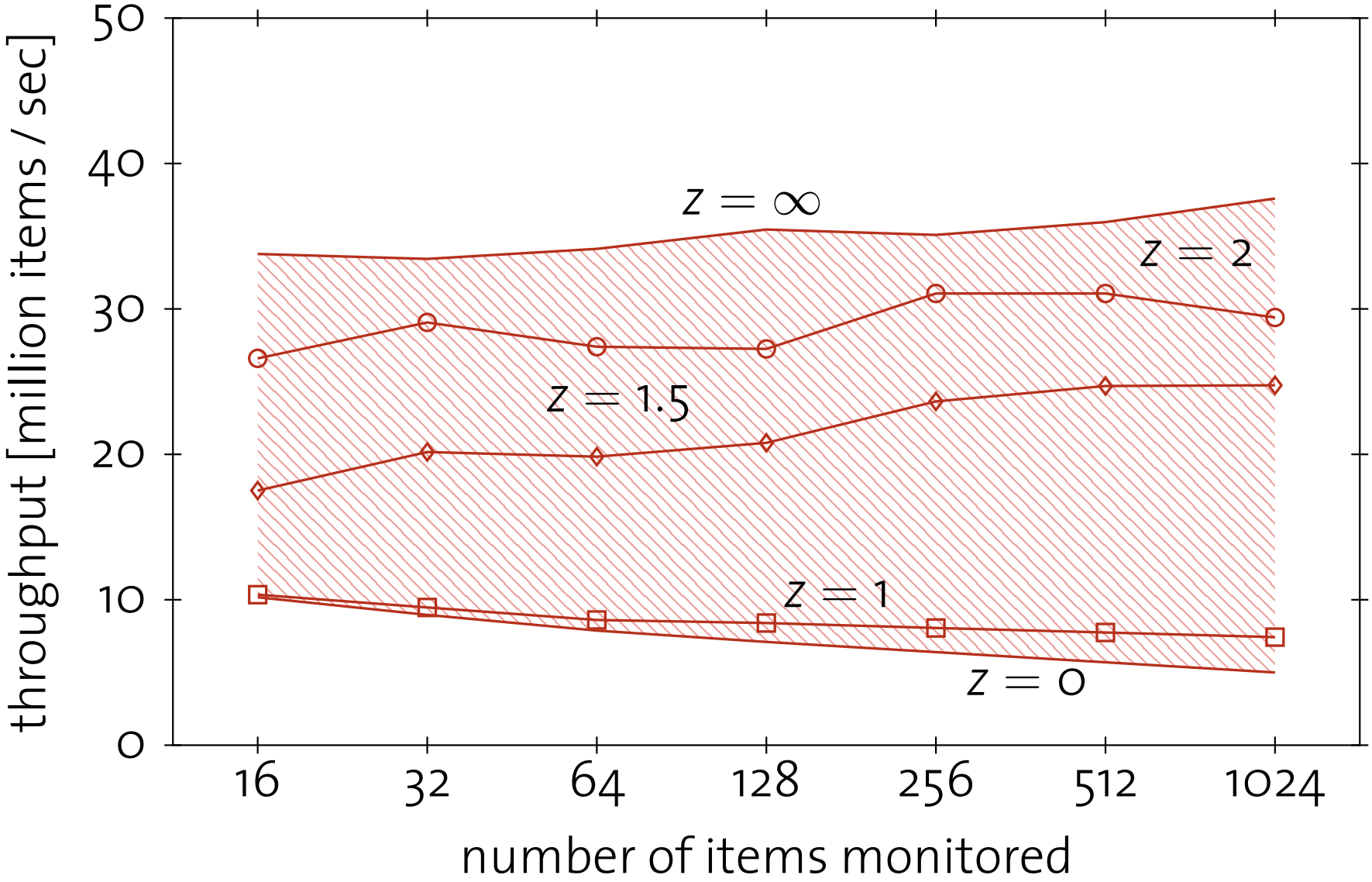
Given a stream S of items x_i,
which items occur most often in S?

Solution: [Metwally *et al.* 2006]

lookup by *item*

lookup by *count*

```
foreach stream item x ∈ S do
    find bin b_x with b_x.item = x ;
    if such a bin was found then
        b_x.count ← b_x.count + 1 ;
    else
        b_min ← bin with minimum count value ;
        b_min.count ← b_min.count + 1 ;
        b_min.item ← x ;
```



(Intel T9550 @ 2.66 MHz; code by Cormode and Hadjieleftheriou, VLDB 2008)

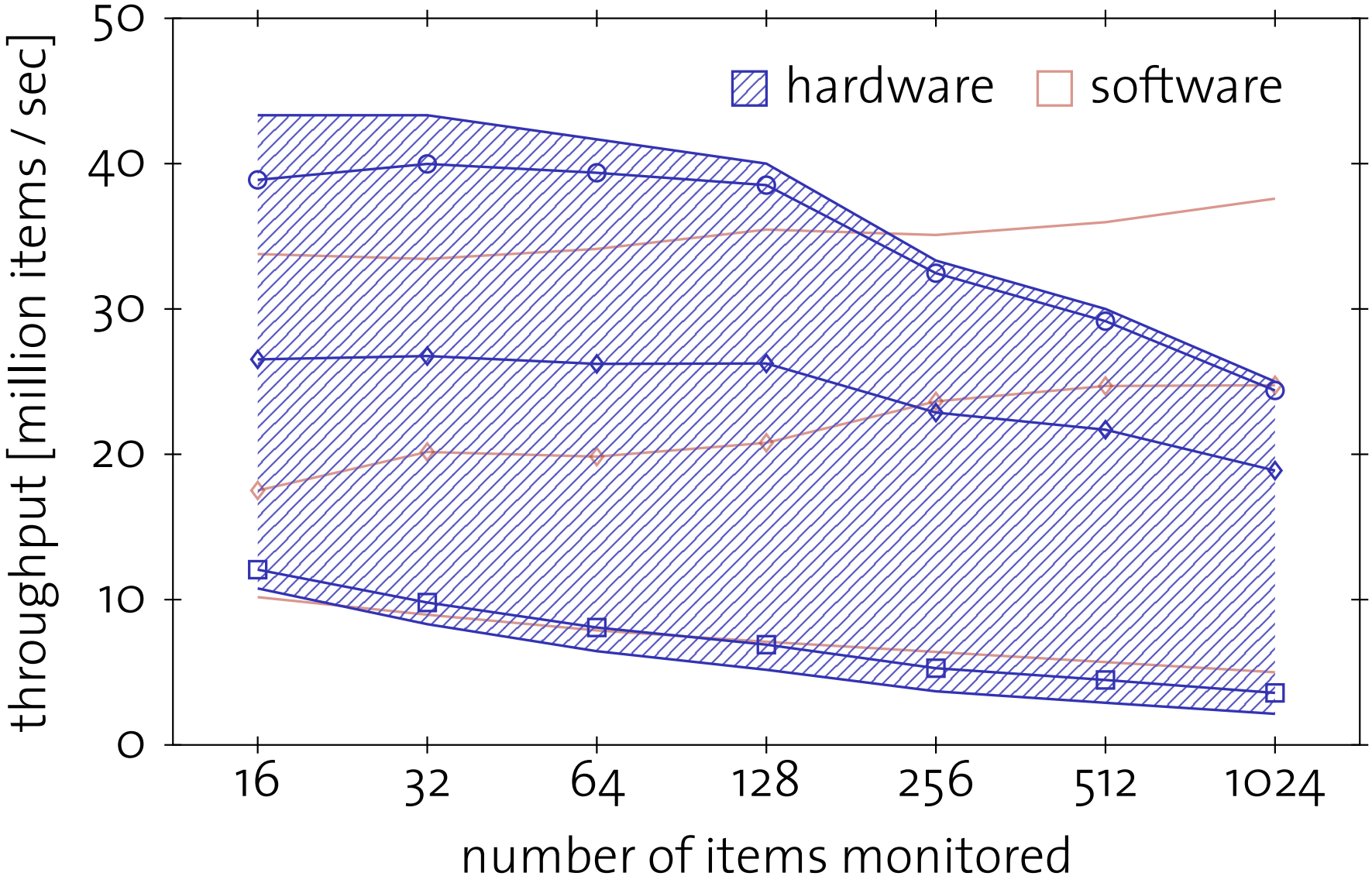
Tricks on FPGAs:

content-addressable memory

"hash table on steroids"

dual-ported memory

min-heap maintenance speed-up

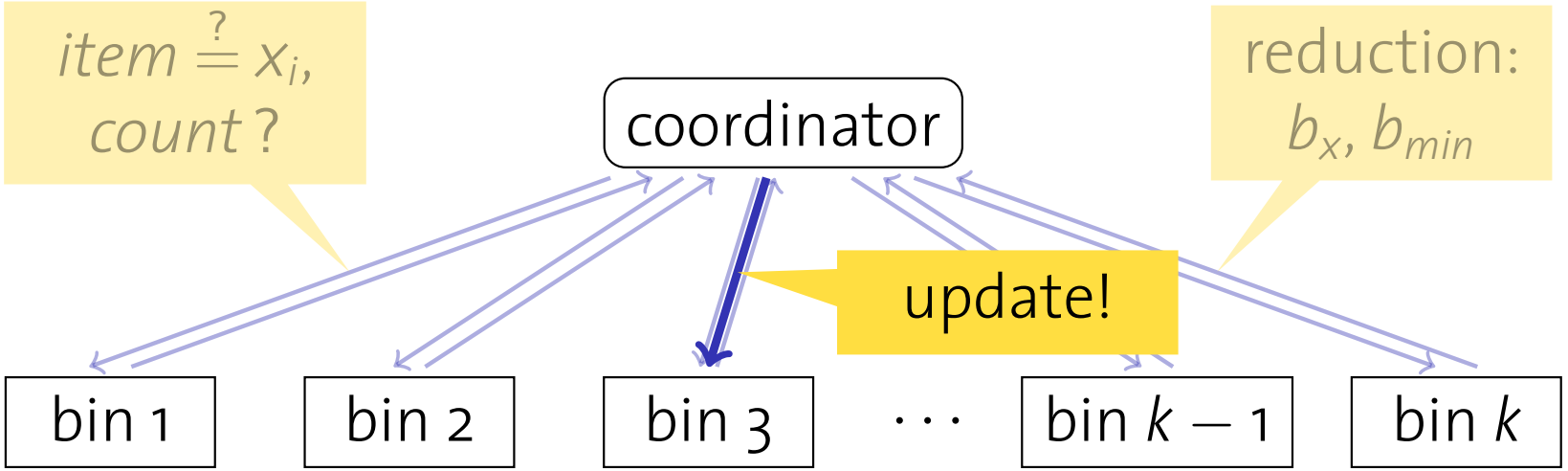


→ **data dependent, not scalable** ☹

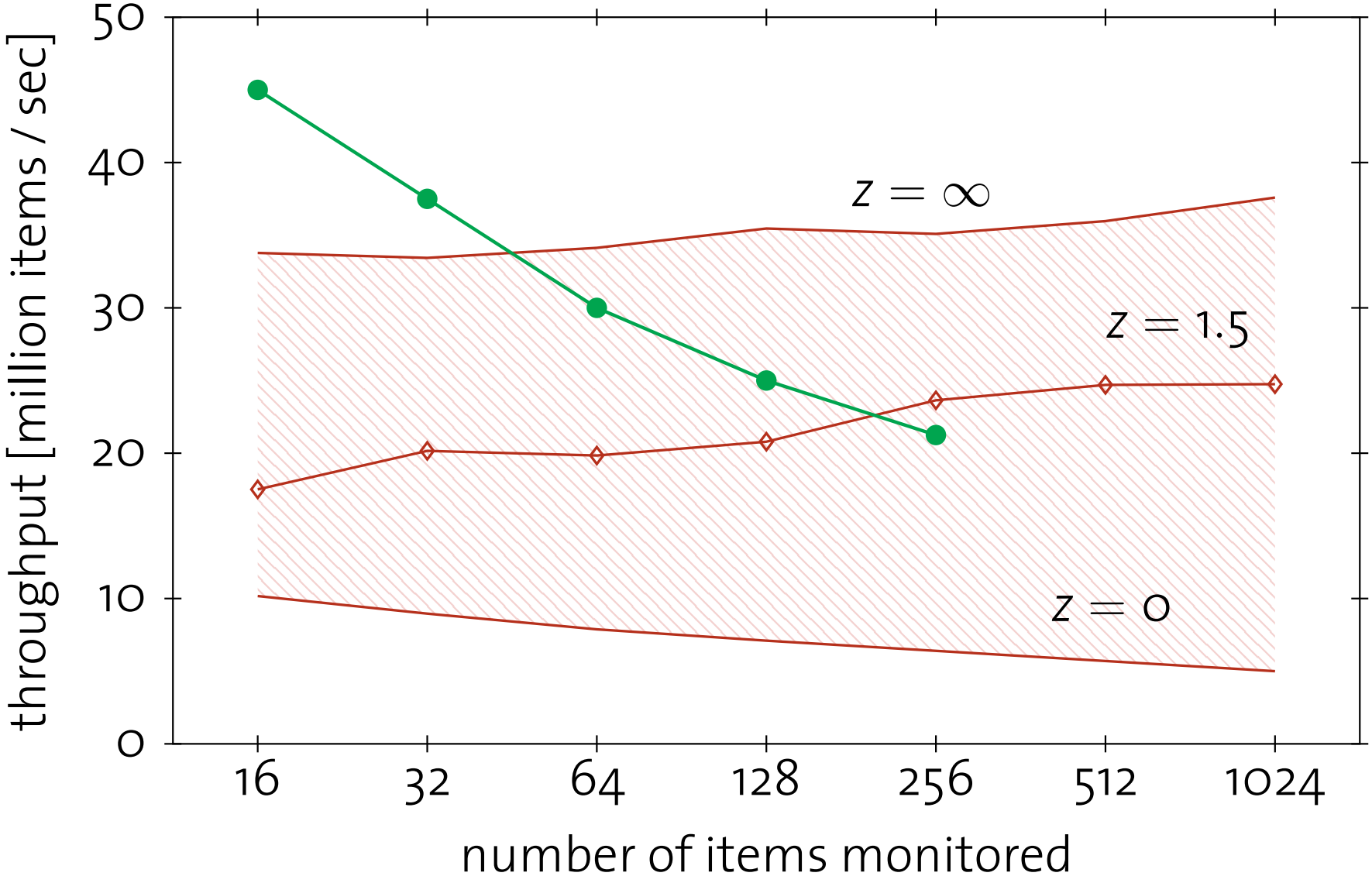
Lesson 1:

FPGAs are not a silver bullet.

Idea: Parallelize

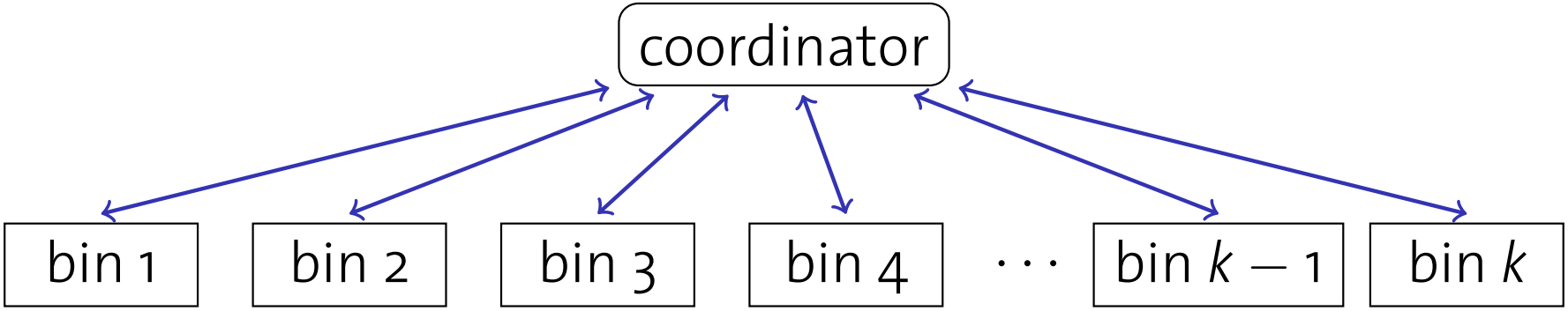


1. **Broadcast** input item x_i to all bins.
2. **Reduce** to determine b_x and b_{min}.
3. **Update** b_x/b_{min}.



→ **still not scalable** ☹

What went wrong?



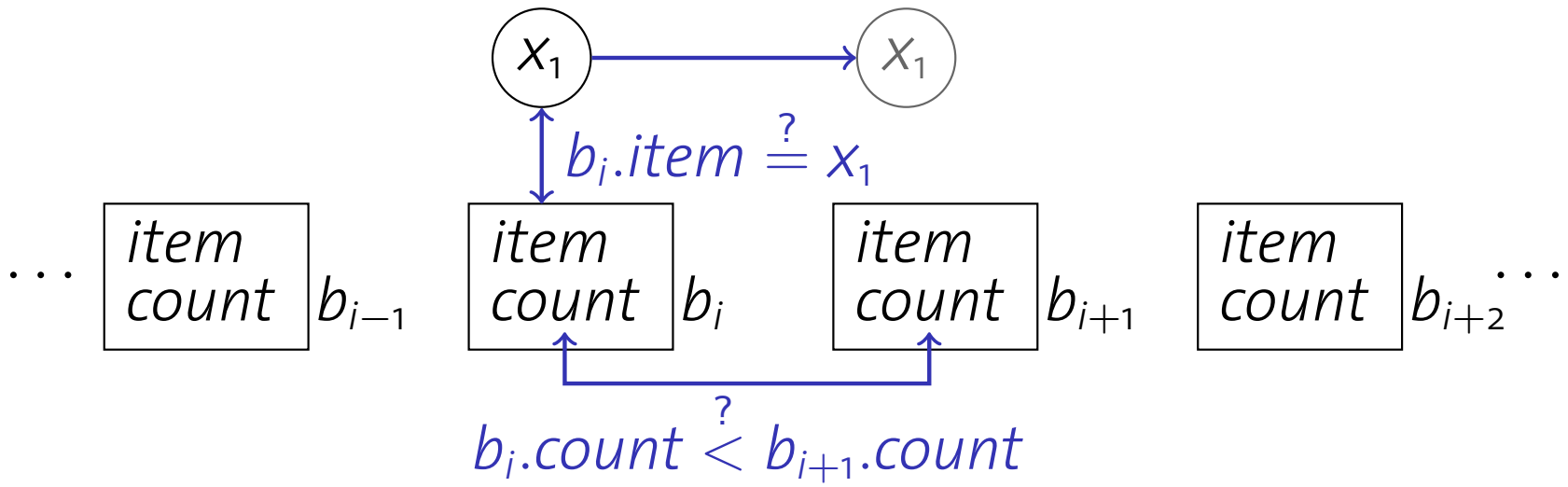
Lesson 2:

Avoid long-distance communication.

Can we **keep processing local**?

(avoid long-distance communication)

Pipeline-Style Processing:



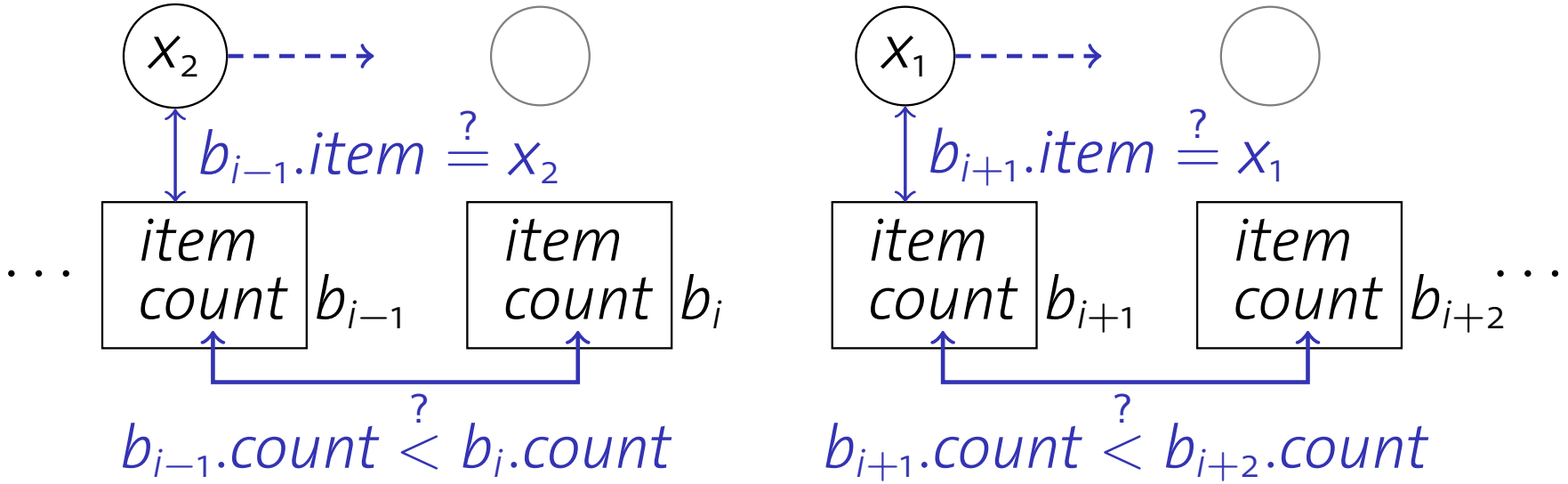
1. **Compare** input item x_1 to content of bin b_i (and **increment** *count* value if a match was found).
2. **Order** bins b_i and b_{i+1} according to *count* values.
3. **Move** x_1 **forward** in the array and **repeat**.

→ Drop x_1 into **last bin** if no match can be found.

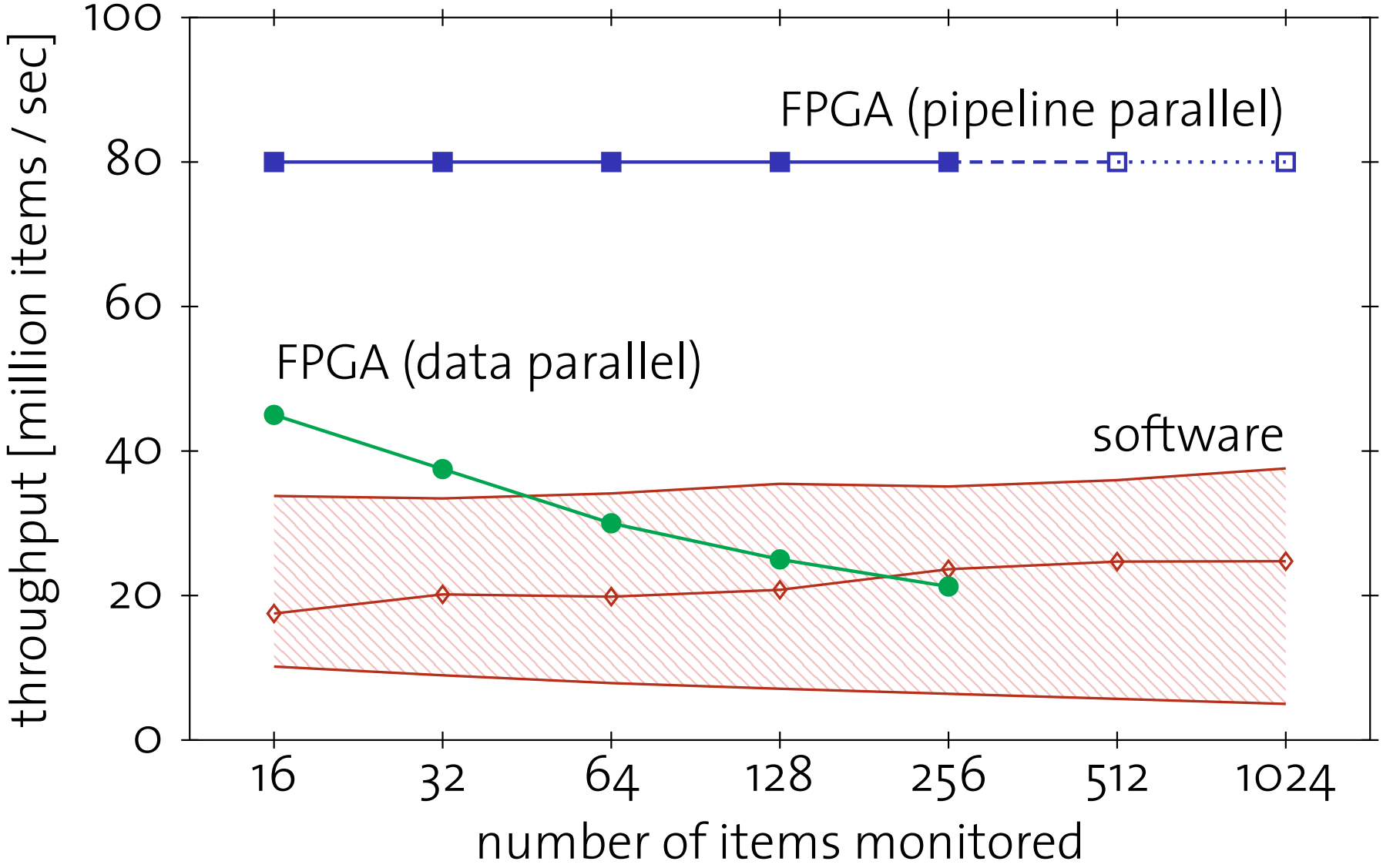
$$\mathcal{O}(1) \rightarrow \mathcal{O}(\#\text{bins})\ ?$$

But: Can be **parallelized** well.

Pipeline Parallelism:



$$\mathcal{O}(\#\text{bins}) \rightarrow \frac{1}{\#\text{bins}} \cdot \mathcal{O}(\#\text{bins})$$


throughput [million items / sec]
100
80
60
40
20
0
FPGA (pipeline parallel)
FPGA (data parallel)
software
16
32
64
128
256
512
1024
number of items monitored

Lesson 3:

Pipelining $\rightarrow$ scalability, performance.

Lessons learned:

1. **FPGAs are not a silver bullet.**

   Straightforward s/w $\rightarrow$ h/w mapping will **not** do the job.

2. **Avoid long-distance communication.**

   Signal propagation delays will limit scalability.

3. **Pipelining $\rightarrow$ scalability, performance.**

   Keep communication and synchronization cheap.

Frequent item solution:

- **three times faster** than software, **data independent.**

This work was supported by the **Swiss National Science Foundation**.