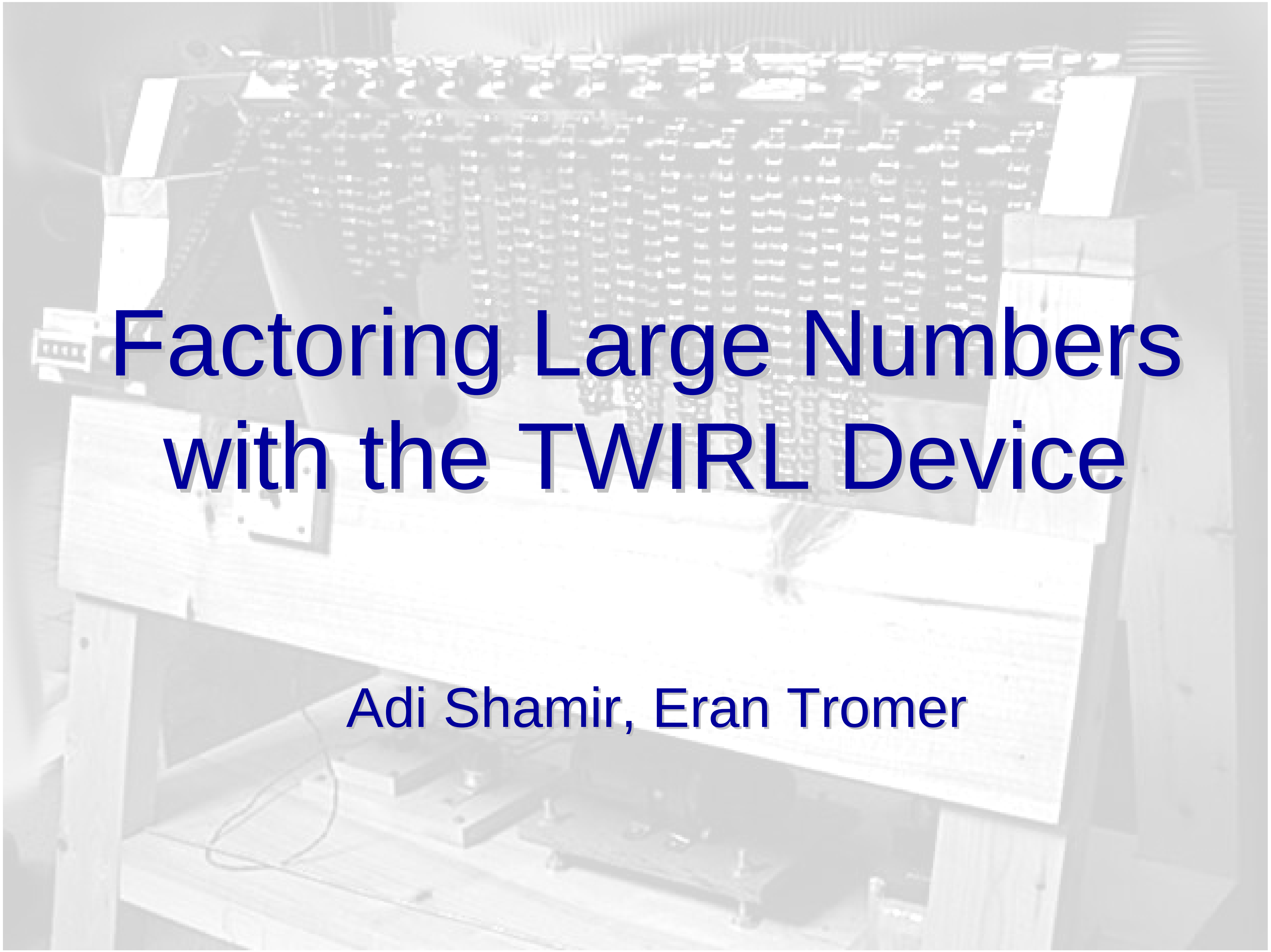
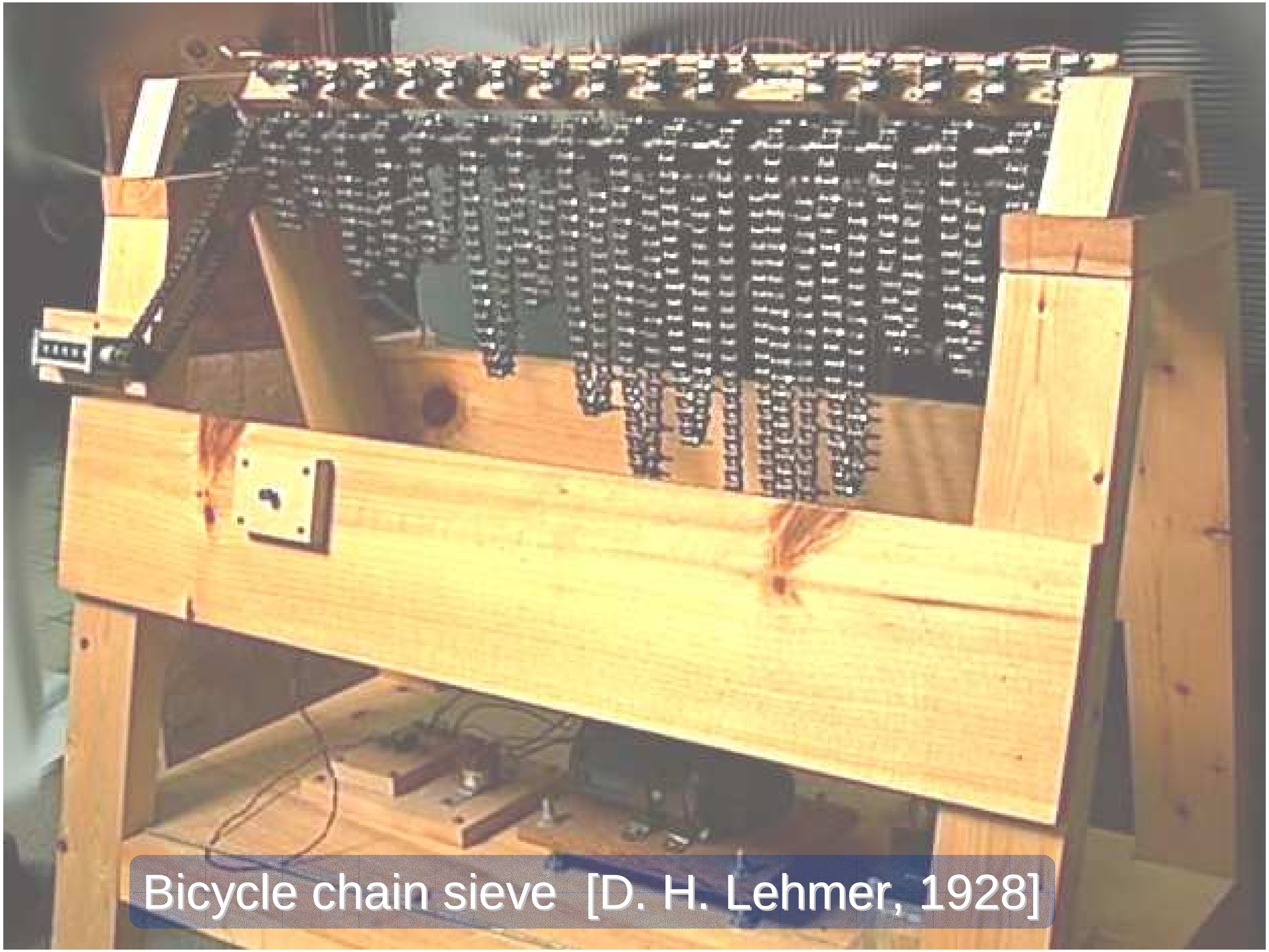


Factoring Large Numbers with the TWIRL Device

Adi Shamir, Eran Tromer


Bicycle chain sieve [D. H. Lehmer, 1928]

The Number Field Sieve Integer Factorization Algorithm

- Best algorithm known for factoring large integers.
- Subexponential time, subexponential space.

- Successfully factored a 512-bit RSA key in 1999 (hundreds of workstations running for many months).
- Record: 530-bit integer factored in 2003.

NFS: Main steps

Relation collection (sieving) step:	**Matrix step:**
Find many numbers satisfying a certain (rare) property.	Find a linear dependency among the numbers found.

NFS: Main steps

Relation collection (sieving) step:	**Matrix step:**
Find many numbers satisfying a certain (rare) property. This work	Find a linear dependency among the numbers found. Cost dramatically reduced by [Bernstein 2001] followed by [LSTT 2002] and [GS 2003].

Cost of sieving for RSA-1024 in 1 year

- Traditional PC-based: [Silverman 2000]
  100M PCs with 170GB RAM each: $\$5\times10^{12}$

- TWINKLE: [Lenstra,Shamir 2000][Silverman 2000]*
  3.5M TWINKLEs and 14M PCs: ~ $\$10^{11}$

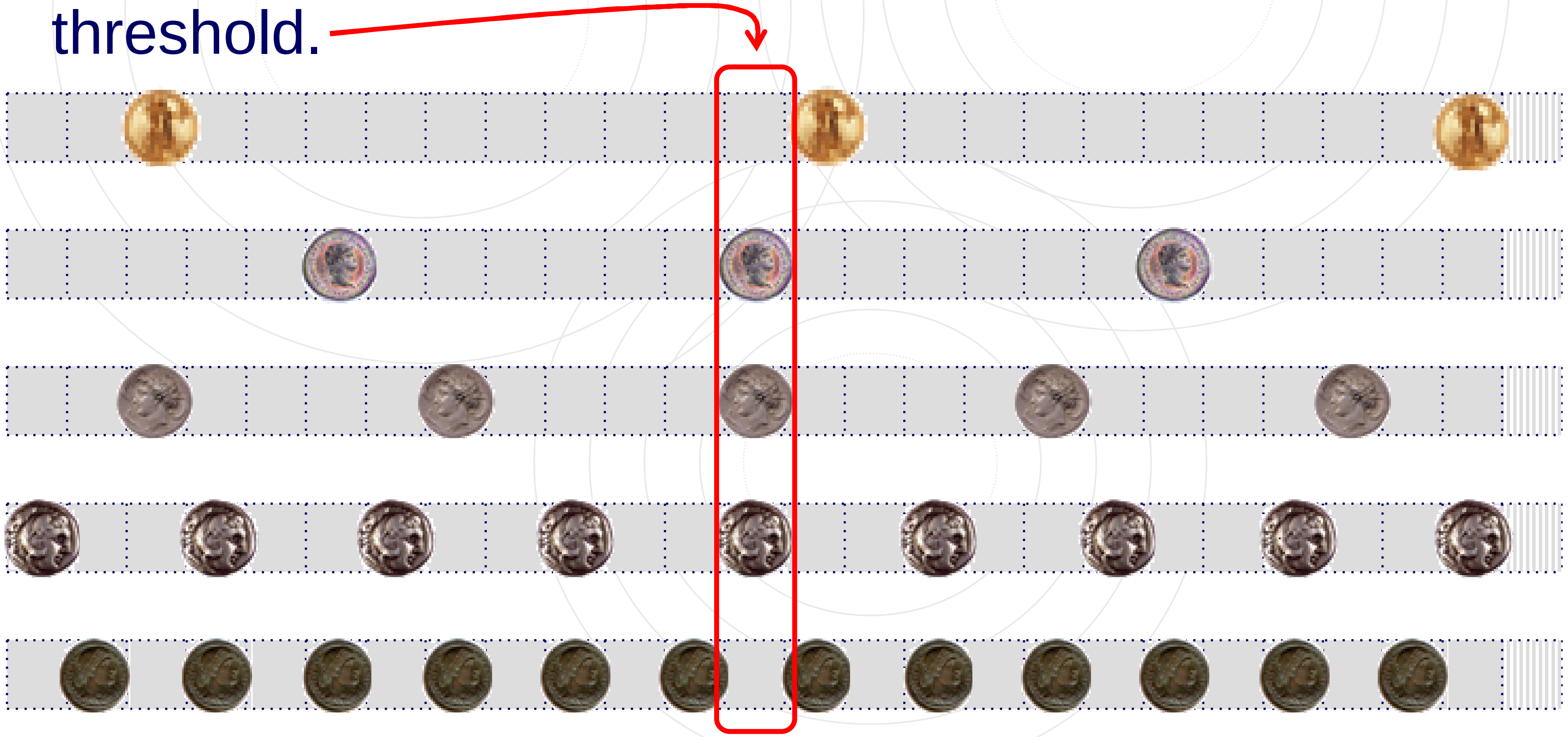
- Mesh-based sieving [Geiselmann,Steinwandt 2002]*
  Millions of devices, $\$10^{11}$ to $\$10^{10}$ (if at all?)
  Multi-wafer design – feasible?

- Our design: $10M using standard silicon technology (0.13um, 1GHz).

The Sieving Problem

Input: a set of arithmetic progressions. Each progression has a prime interval p and value $\log p$.

Output: indices where the sum of values exceeds a threshold.

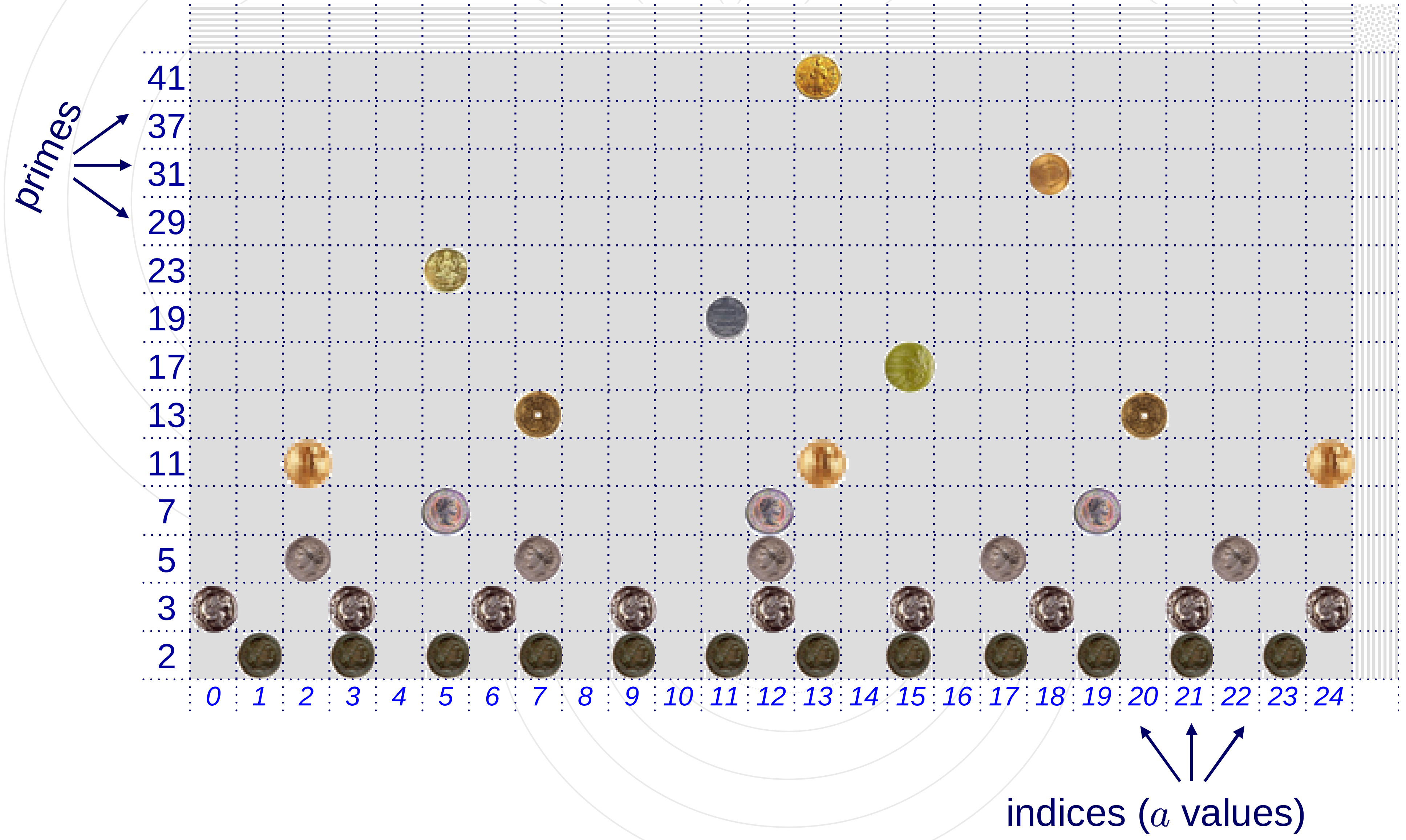


1024-bit NFS sieving parameters

- Total number of indices to test: 3×10^{23}.

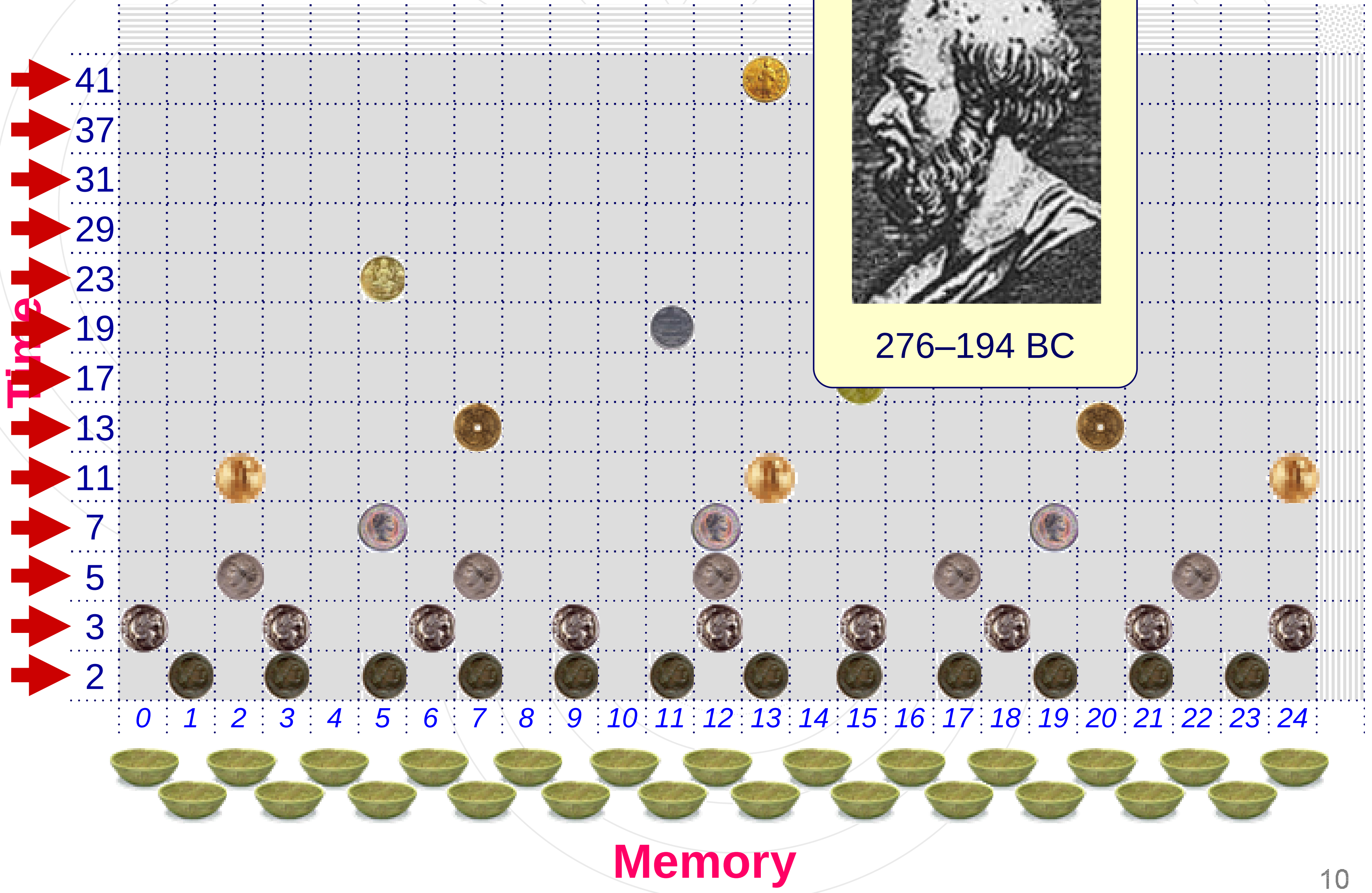
- Each index should be tested against all primes up to 3.5×10^{9}.

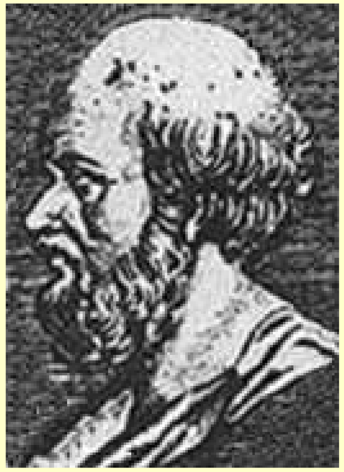
Three ways to sieve your numbers...



PC-based sieving, à la Eratosthenes

One contribution per clock cycle.





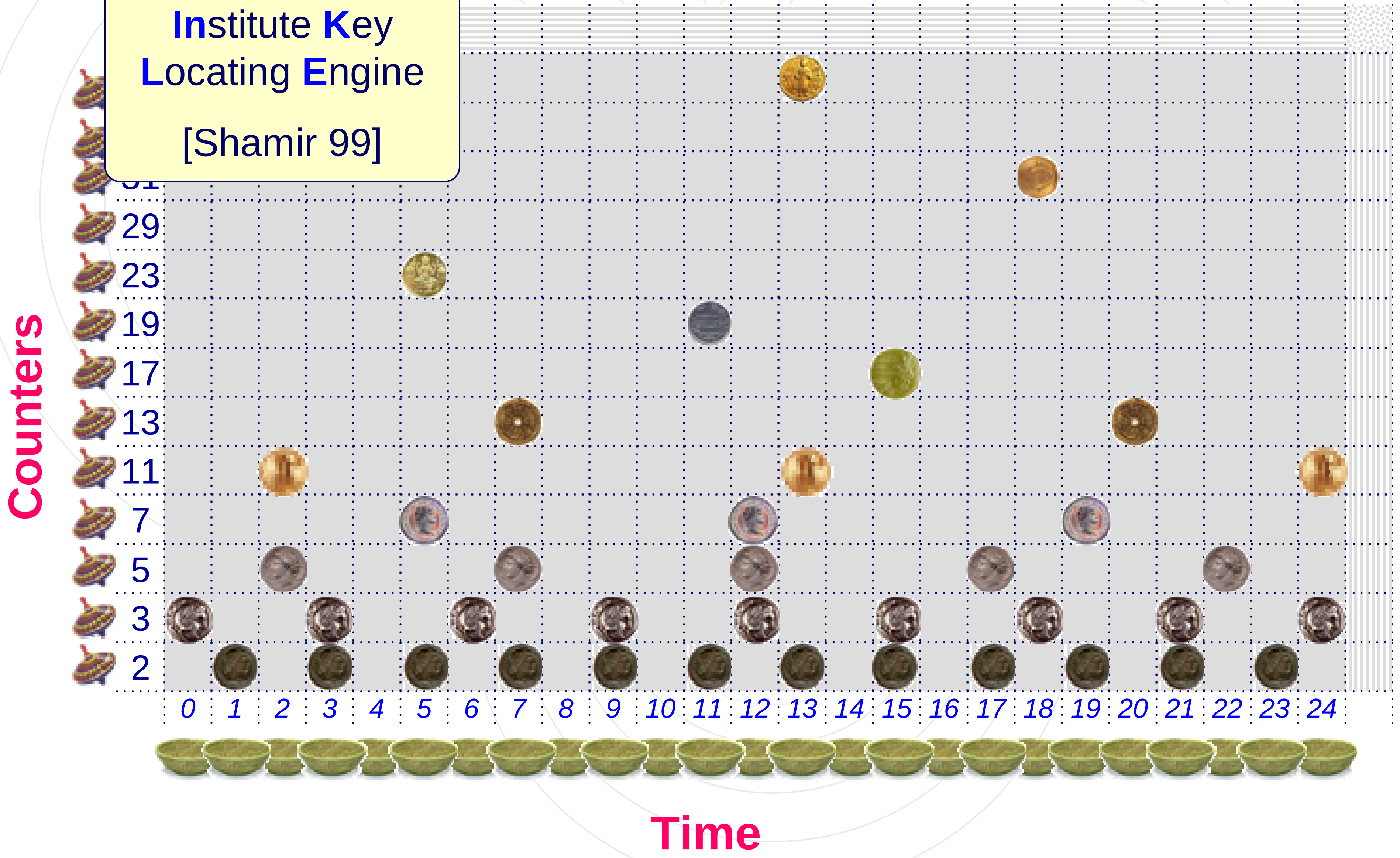
276–194 BC

TWINKLE: time-space reversal

ed at each clock cycle.

The **W**eizmann **I**nstitute **K**ey **L**ocating **E**ngine

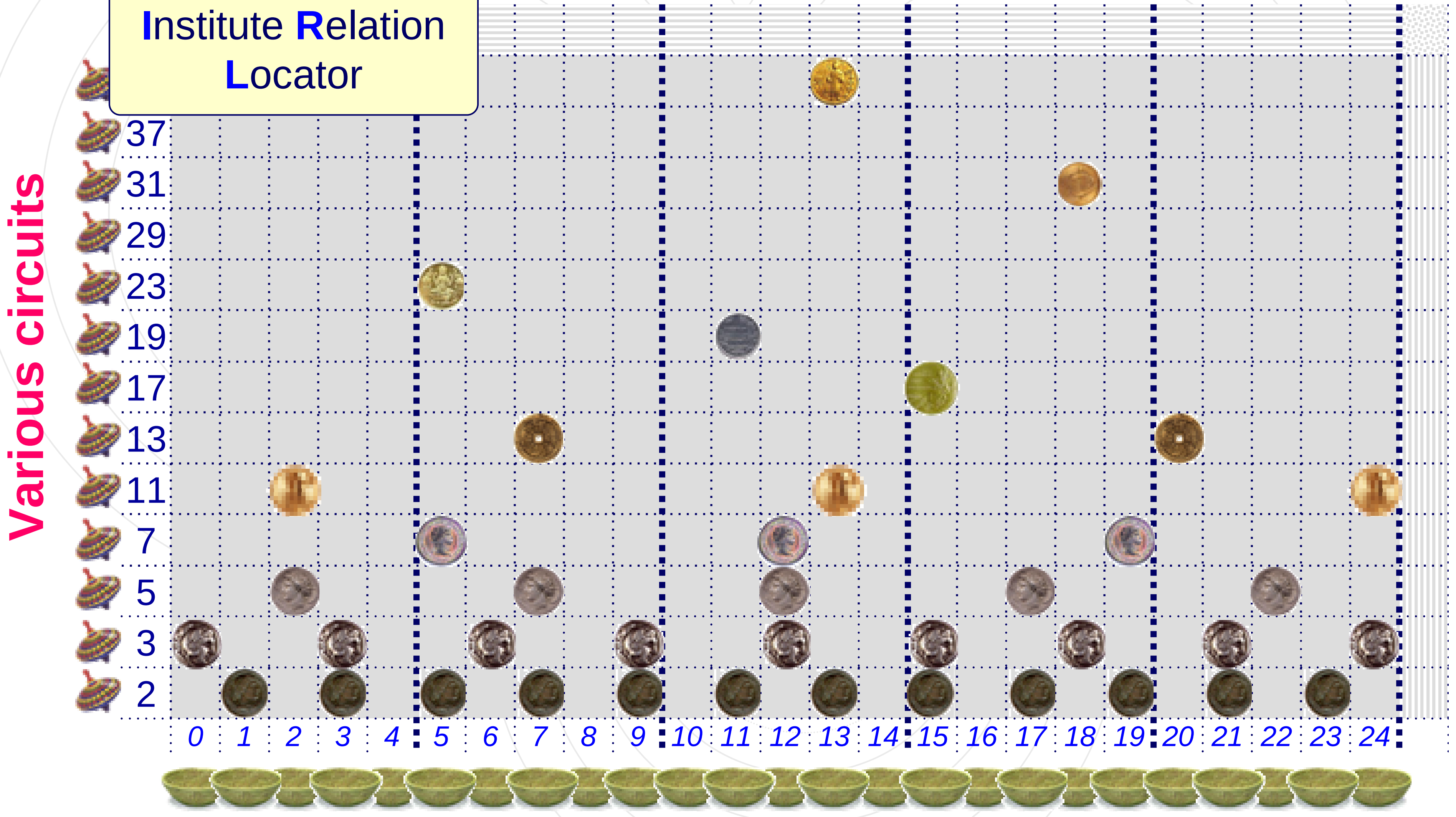
[Shamir 99]



TWIRL: compressed time

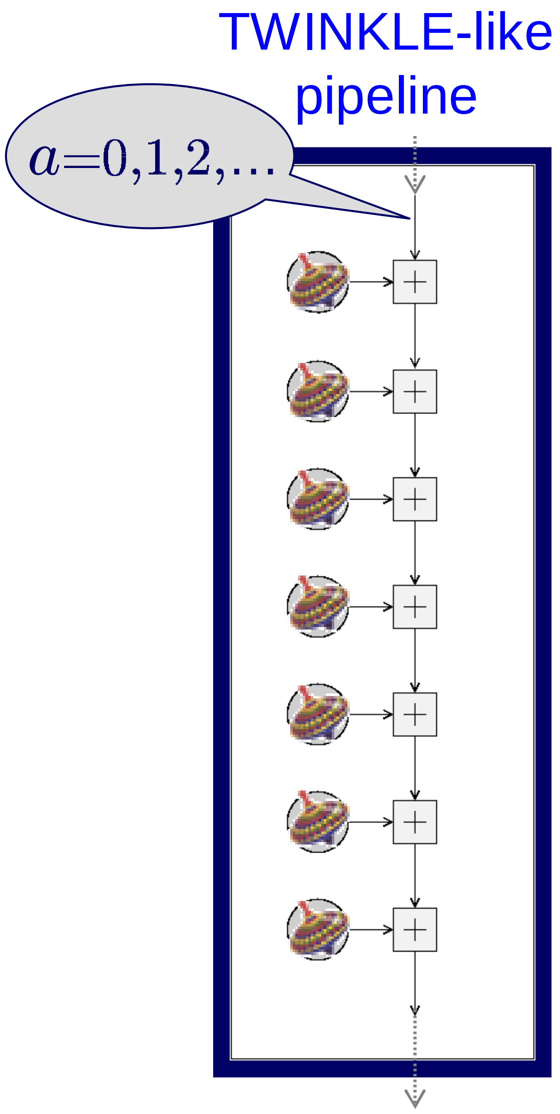
The **W**eizmann **I**nstitute **R**elation **L**ocator

dled at each clock cycle. (real: $s=32768$)

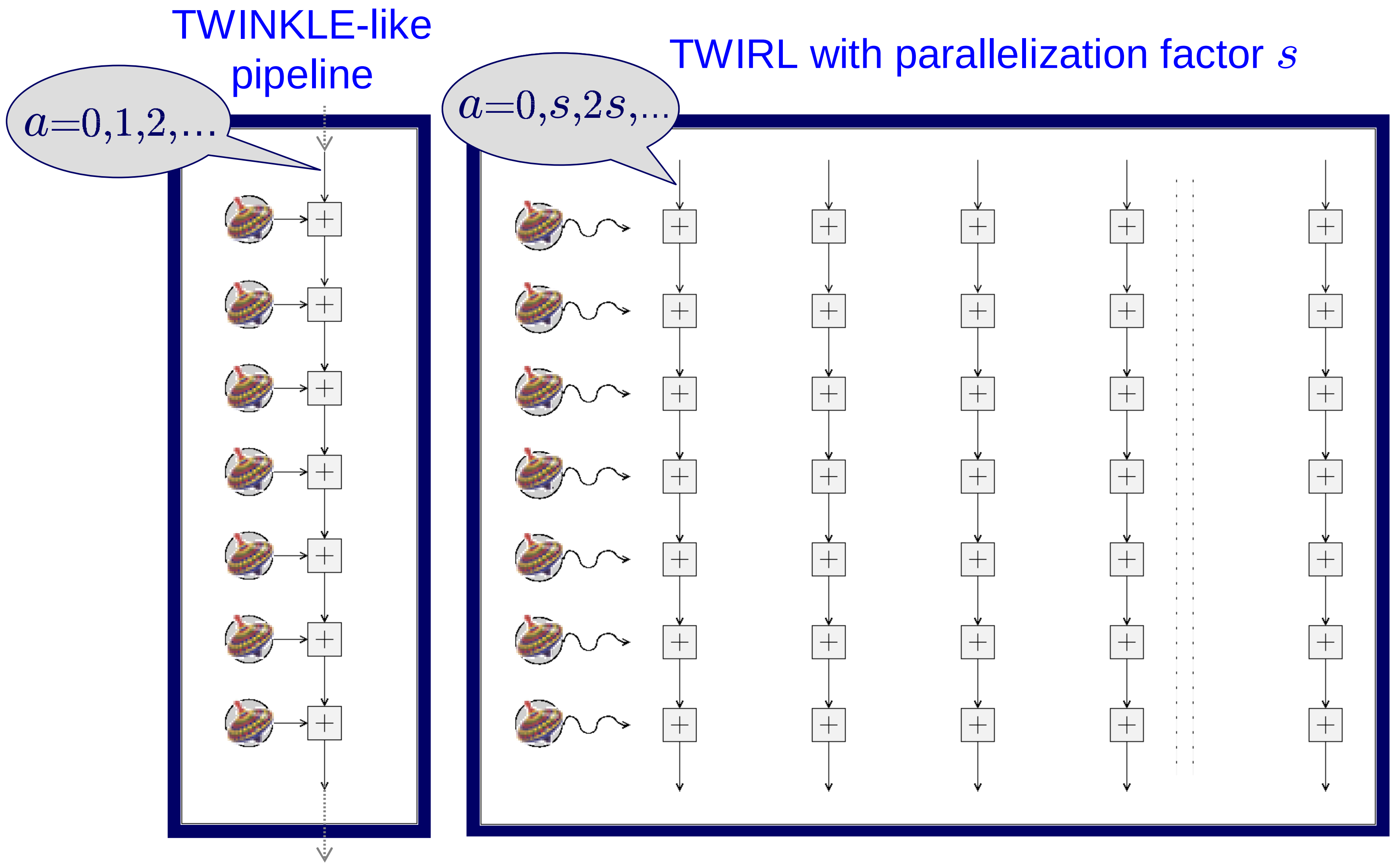


Time

Parallelization in TWIRL

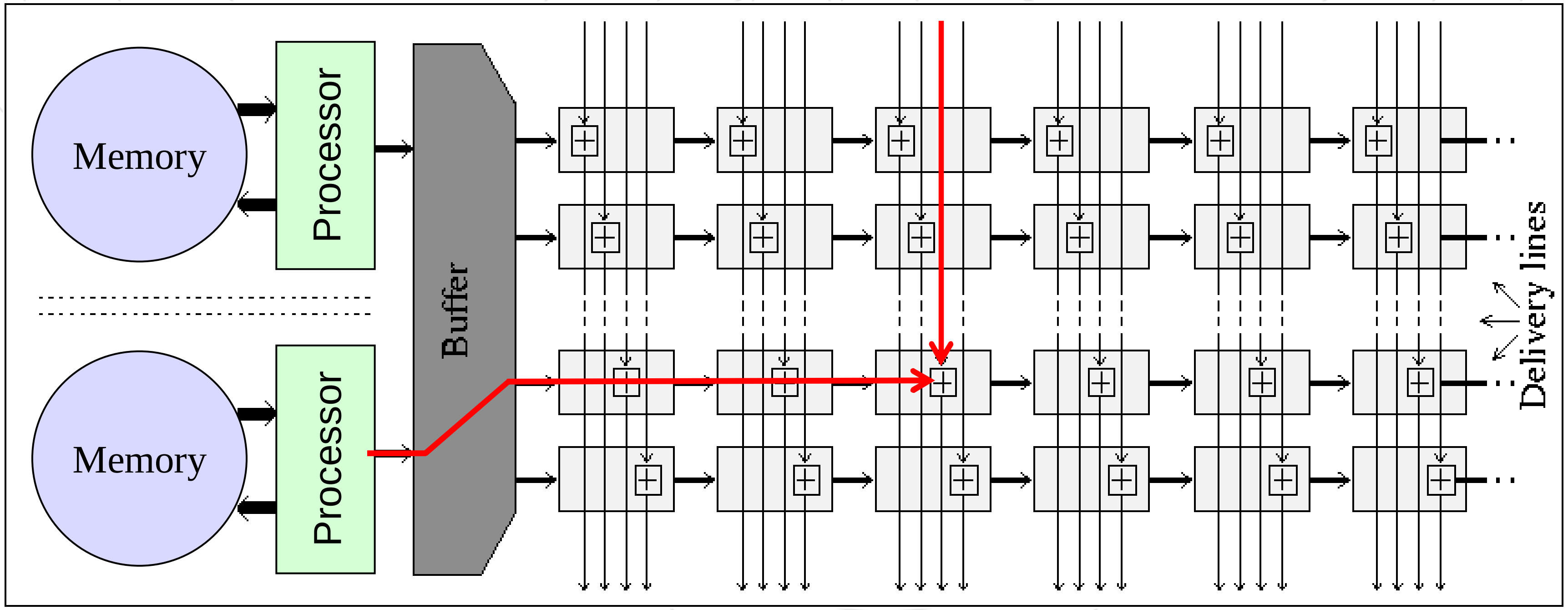


Parallelization in TWIRL

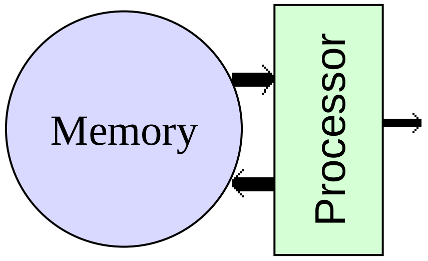


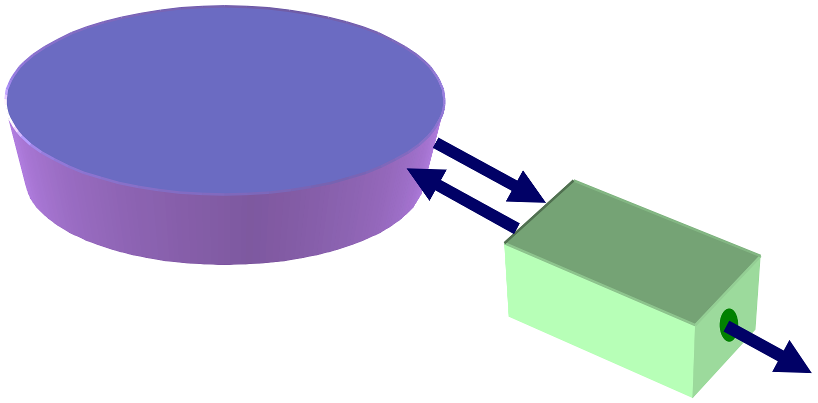
Example (simplified): handling large primes

- Each prime makes a contribution once per 10,000’s of clock cycles (after time compression); inbetween, it’s merely stored compactly in DRAM.
- Each memory+processor unit handles 10,000’s of progressions. It computes and sends contributions across the bus, where they are added at just the right time. Timing is critical.



Handling large primes (cont.)

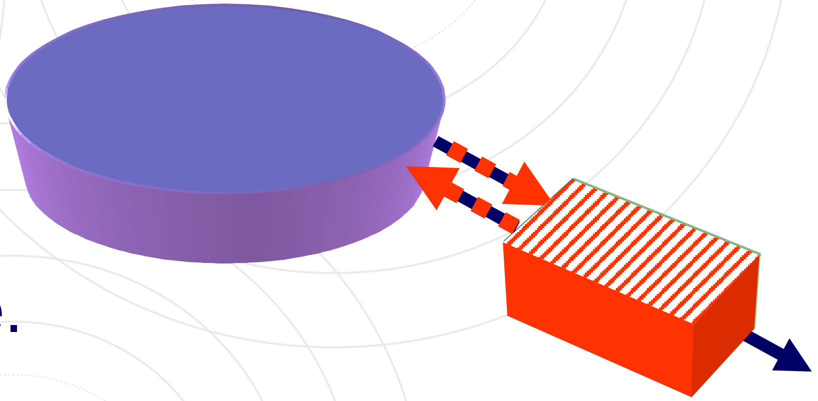




Implementing a priority queue of events

- The memory contains a list of events of the form (p_i, a_i), meaning "*a progression with interval p_i will make a contribution to index a_i*". Goal: implement a priority queue.
- The list is ordered by increasing a_i.
- At each clock cycle:

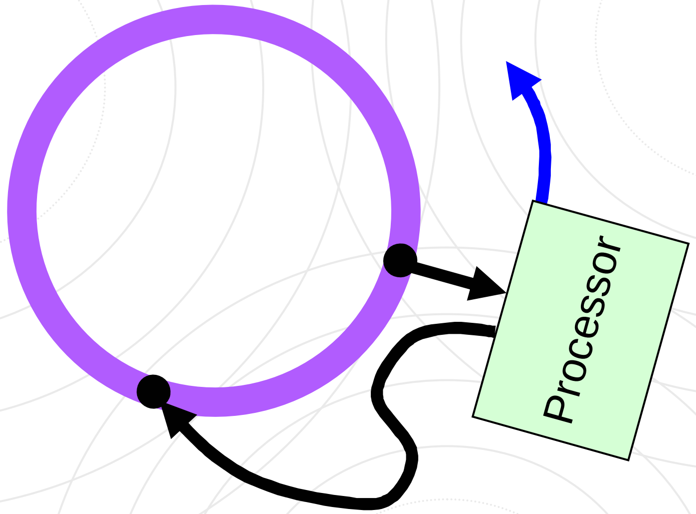
1. Read next event (p_i, a_i).
2. Send a $\log p_i$ contribution to line $a_i \pmod{s}$ of the pipeline.
3. Update $a_i \leftarrow a_i + p_i$
4. Save the new event (p_i, a_i) to the memory location that will be read just before index a_i passes through the pipeline.



- To handle collisions, slacks and logic are added.

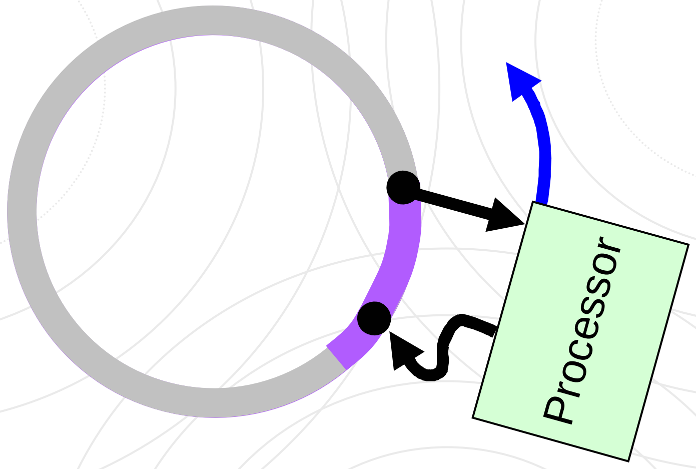
Handling large primes (cont.)

- The memory used by past events can be reused.
- Think of the processor as rotating around the cyclic memory:



Handling large primes (cont.)

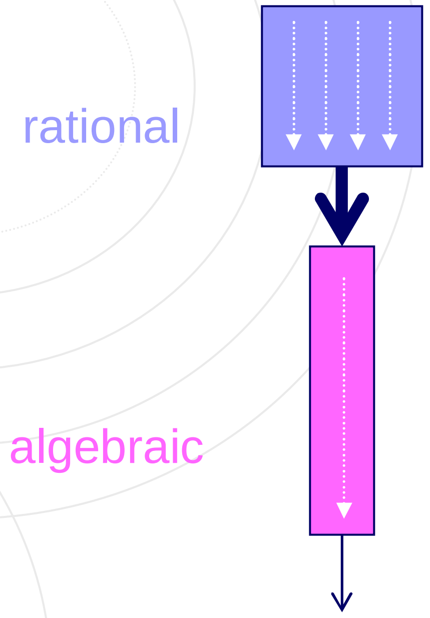
- The memory used by past events can be reused.
- Think of the processor as rotating around the cyclic memory:



- By assigning similarly-sized primes to the same processor (+ appropriate choice of parameters), we guarantee that new events are always written just behind the read head.
- There is a tiny (1:1000) window of activity which is “twirling” around the memory bank. It is handled by an SRAM-based cache. The bulk of storage is handled in compact DRAM.

Rational vs. algebraic sieves

- In fact, we need to perform two sieves: rational (expensive) and algebraic (even more expensive).
- We are interested only in indices which pass both sieves.
- We can use the results of the rational sieve to greatly reduce the cost of the algebraic sieve.



Notes

- TWIRL is a hypothetical and untested design.
- It uses a highly fault-tolerant wafer-scale design.
- The following analysis is based on approximations and simulations.

TWIRL for 512-bit composites

One silicon wafer full of TWIRL devices (total cost ~$15,000) can complete the sieving in under 10 minutes.

This is 1,600 times faster than the best previous design.

TWIRL for 1024-bit composites

- Operates in clusters of 3 almost independent wafers.
- Initial investment (NRE): ~$20M
- To complete the sieving in 1 year
  - Use 194 clusters (~600 wafers).
  - Silicon cost: ~$2.9M
  - Total cost: ~$10M   (compared to ~$1T).