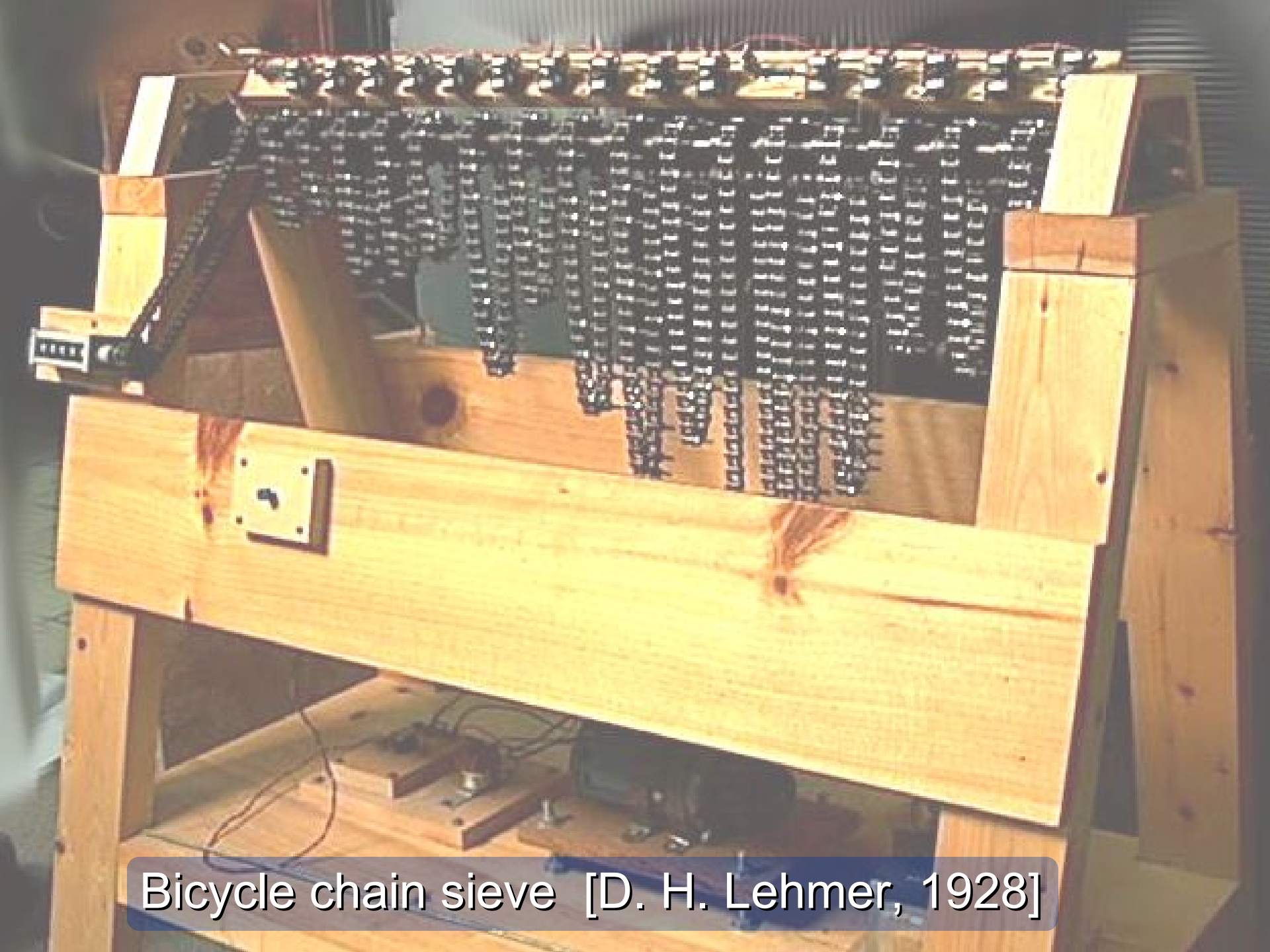
Special Purpose Hardware for Factoring: the NFS Sieving Step

Adi Shamir Eran Tromer

Weizmann Institute of Science


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

NFS: Main computational steps

Relation collection (sieving) step:	**Matrix step:**
Find many relations.	Find a linear relation between the corresponding exponent vectors.
Presently dominates cost for 1024-bit composites.	Cost dramatically reduced by mesh-based circuits.
Subject of this survey.	Surveyed in Adi Shamir's talk.

Outline

- The relation collection problem
- Traditional sieving
- TWINKLE
- TWIRL
- Mesh-based sieving

The Relation Collection Step

The task:
Given a polynomial f (and f'), find many integers a for which $f(a)$ is B-smooth (and $f'(a)$ is B'-smooth).

For 1024-bit composites:

- We need to test 3×10^{23} sieve locations (per sieve).
- The values $f(a)$ are on the order of 10^{100}.
- Each $f(a)$ should be tested against all primes up to $B=3.5\times10^{9}$ (rational sieve) and $B'=2.6\times10^{10}$ (algebraic sieve).

(TWIRL settings)

Sieveless Relation Collection

- We can just factor each $f(a)$ using our favorite factoring algorithm for medium-sized composites, and see if all factors are smaller than B.
- By itself, highly inefficient.
  (But useful for cofactor factorization or Coppersmith's NFS variants.)

Relation Collection via Sieving

- The task:
  Given a polynomial f (and f'), find many integers a for which $f(a)$ is B-smooth (and $f'(a)$ is B'-smooth).
- We look for a such that $p|f(a)$ for many large p:

$$\sum_{\substack{p|f(a) \\ p<B}} \log p > T \approx \log f(a)$$

- Each prime p "hits" at arithmetic progressions:

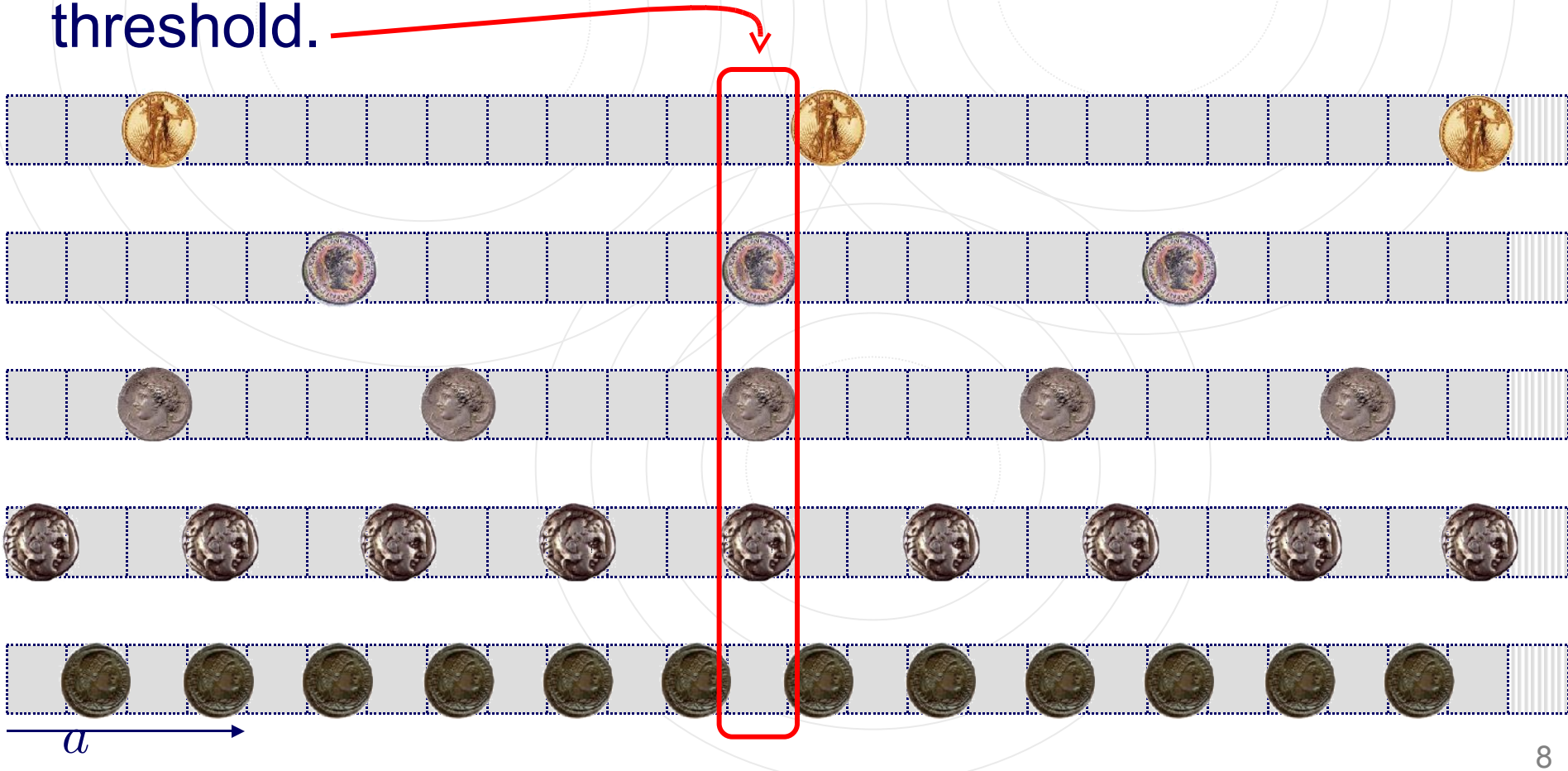
$$\begin{aligned}\{a : p|f(a)\} &= \{a : f(a) \equiv 0 \pmod{p}\} \\ &= \bigcup_i \{r_i + kp : k \in \mathbb{Z}\}\end{aligned}$$

  where r_i are the roots modulo p of f.
  (there are at most $\deg(f)$ such roots, ~1 on average).

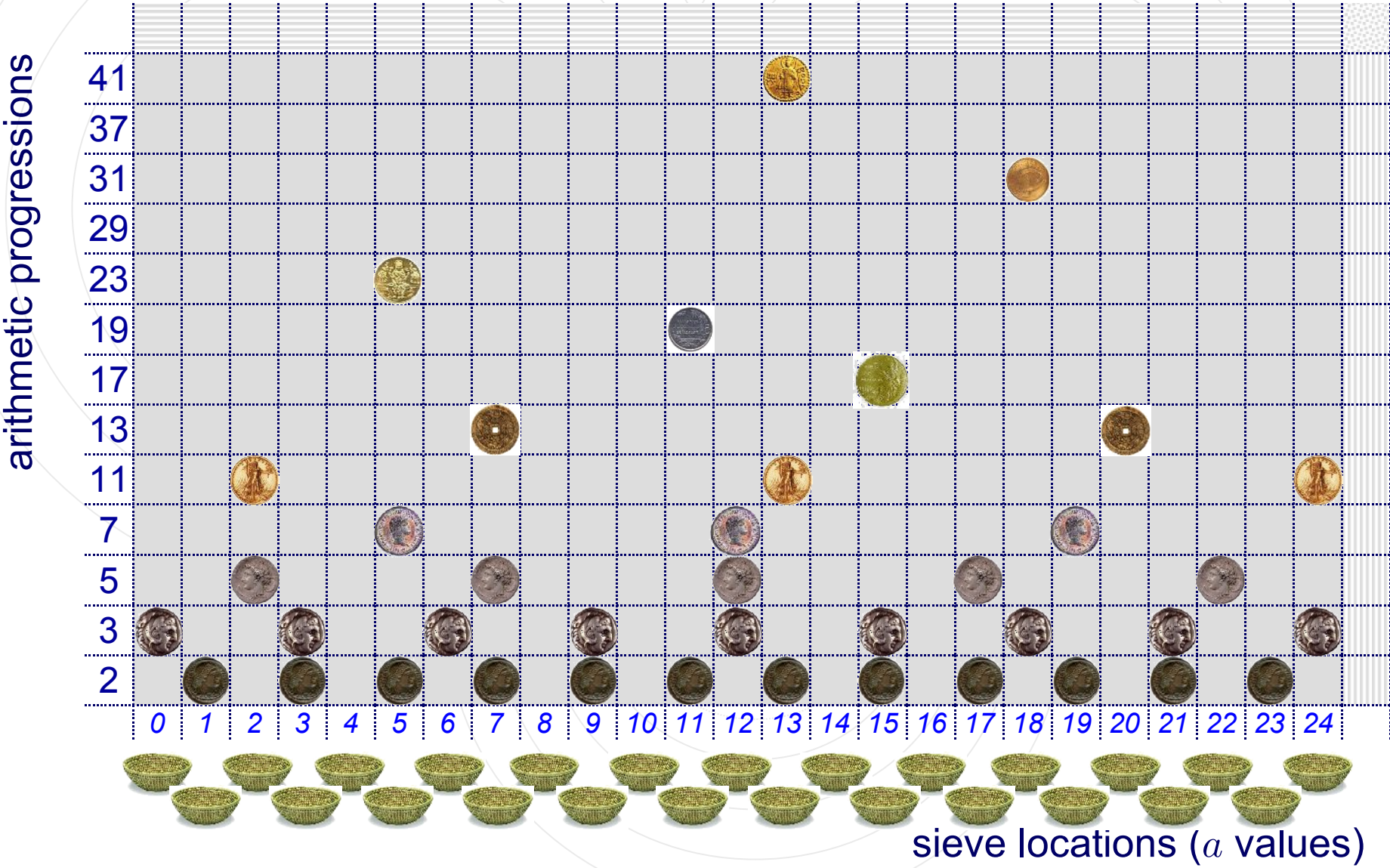
The Sieving Problem

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.



The Game Board



Traditional PC-based sieving

[Eratosthenes of Cyrene]

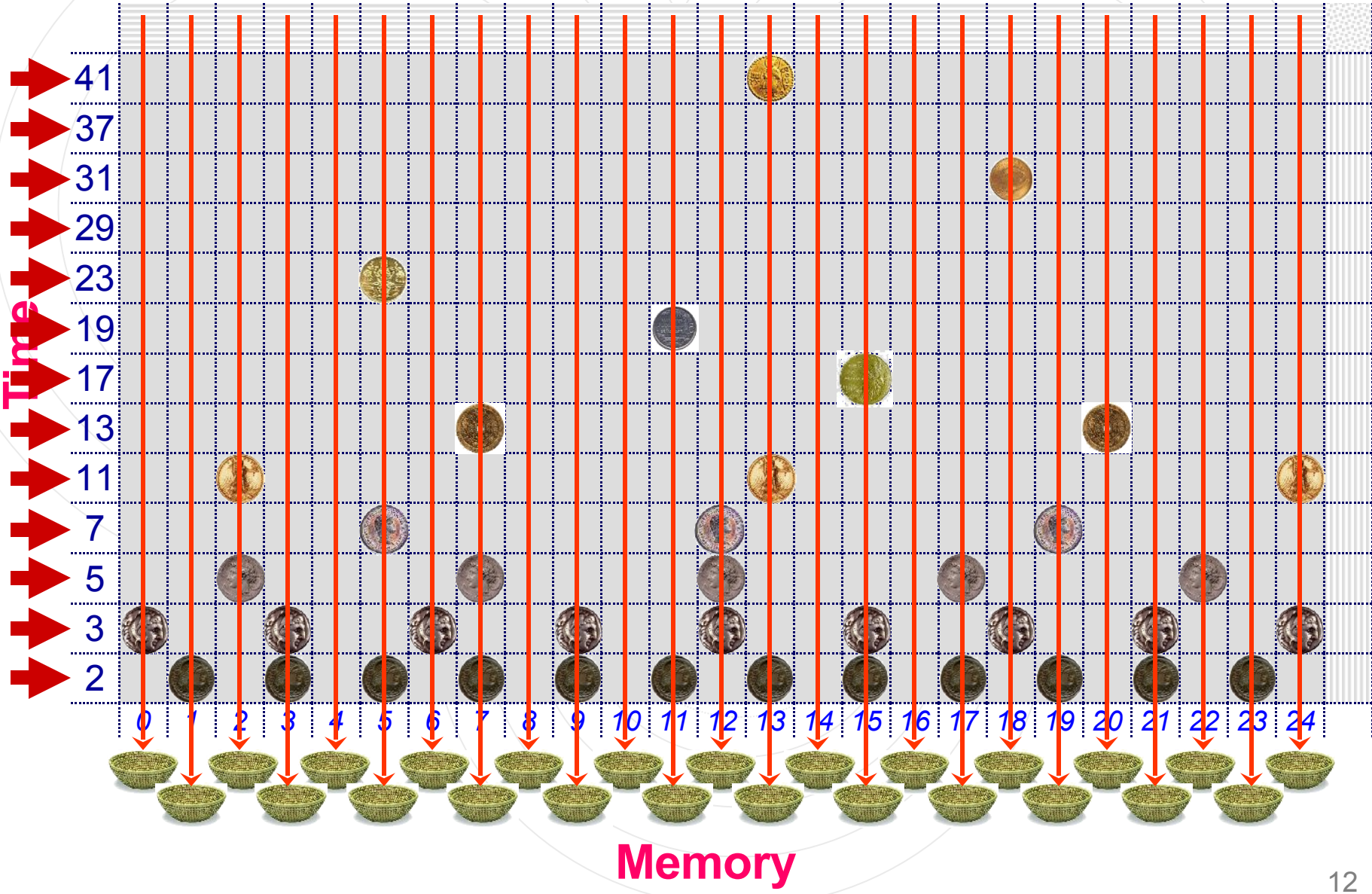
[Carl Pomerance,
Richard Schroeppel]

276–194 BC

PC-based sieving

2. Assign one memory location to each candidate number in the interval.
3. For each arithmetic progression:
   - Go over the members of the arithmetic progression in the interval, and for each:
     - Adding the $\log p$ value to the appropriate memory locations.
4. Scan the array for values passing the threshold.

Traditional sieving, à la Eratosthenes



Properties of traditional PC-based sieving:

- Handles (at most) one contribution per clock cycle.
- Requires PC's with enormously large RAM's.
- For large p, almost any memory access is a cache miss.

Estimated recurring costs with current technology (US$×year)

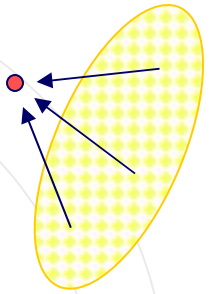
	768-bit	1024-bit
Traditional PC-based	1.3×10^{7}	10^{12}

TWINKLE

(The Weizmann INstitute Key Locating Engine)

[Shamir 1999]
[Lenstra, Shamir 2000]

TWINKLE: An electro-optical sieving device



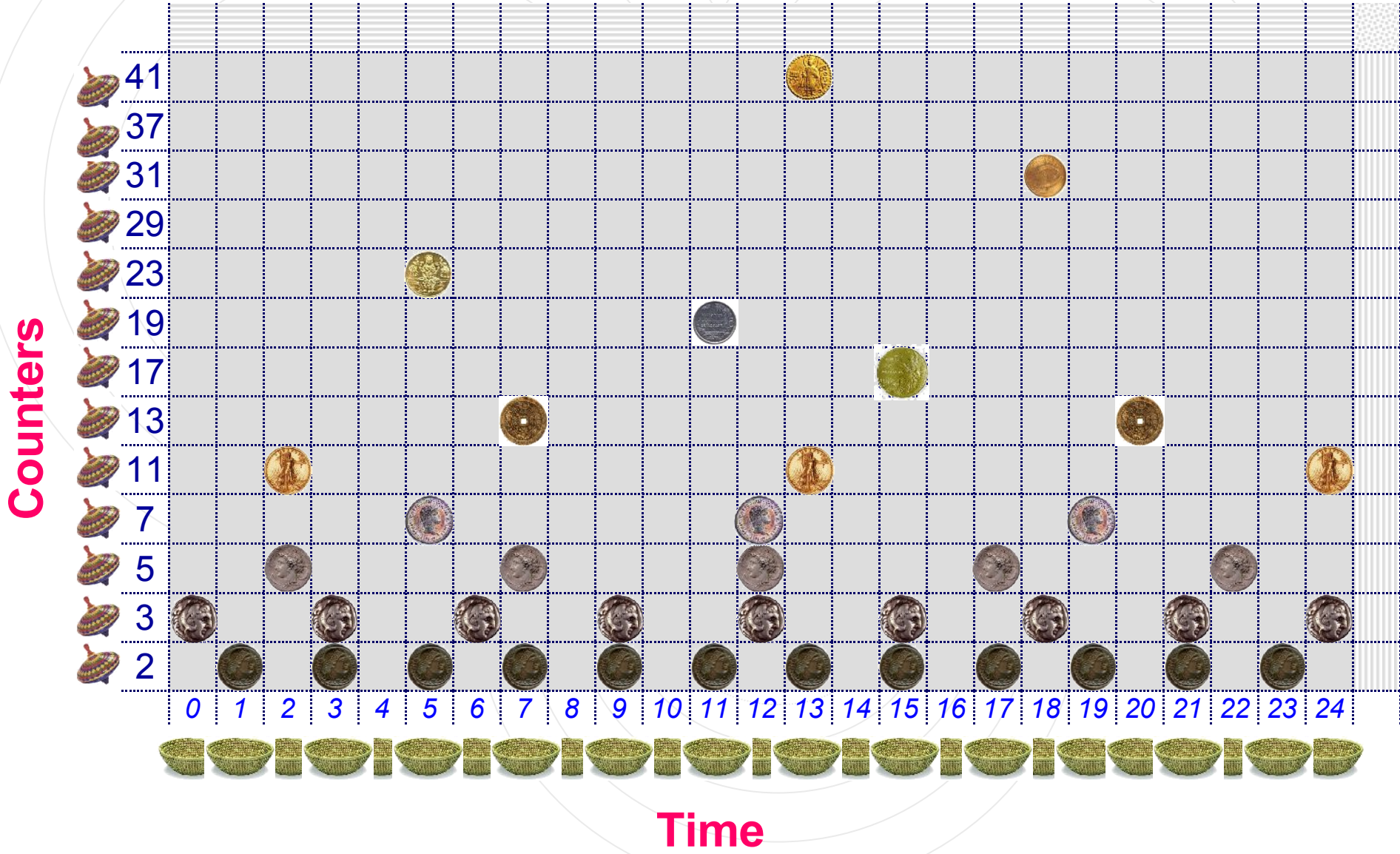
- Reverses the roles of time and space: assigns each arithmetic progression to a small “cell” on a GaAs wafer, and considers the sieved locations one at a time.
- A cell handling a prime p flashes a LED once every p clock cycles.
- The strength of the observed flash is determined by a variable density optical filter placed over the wafer.
- Millions of potential contributions are optically summed and then compared to the desired threshold by a fast photodetector facing the wafer.

Breaking News

Exclusive photos of a working TWINKLE device in this very city!


Photo-emitting cells
(every round hour)
Concave
mirror
Optical
sensor

TWINKLE: time-space reversal



Estimated recurring costs with current technology (US$×year)

	768-bit	1024-bit
Traditional PC-based	1.3×10^{7}	10^{12}
TWINKLE	8×10^{6}	

But: NRE…

Properties of TWINKLE:

- Takes a single clock cycle per sieve location, regardless of the number of contributions.
- Requires complicated and expensive GaAs wafer-scale technology.
- Dissipates a lot of heat since each (continuously operating) cell is associated with a single arithmetic progression.
- Limited number of cells per wafer.
- Requires auxiliary support PCs, which turn out to dominate cost.

TWIRL

(The Weizmann Institute Relation Locator)

[Shamir, Tromer 2003]

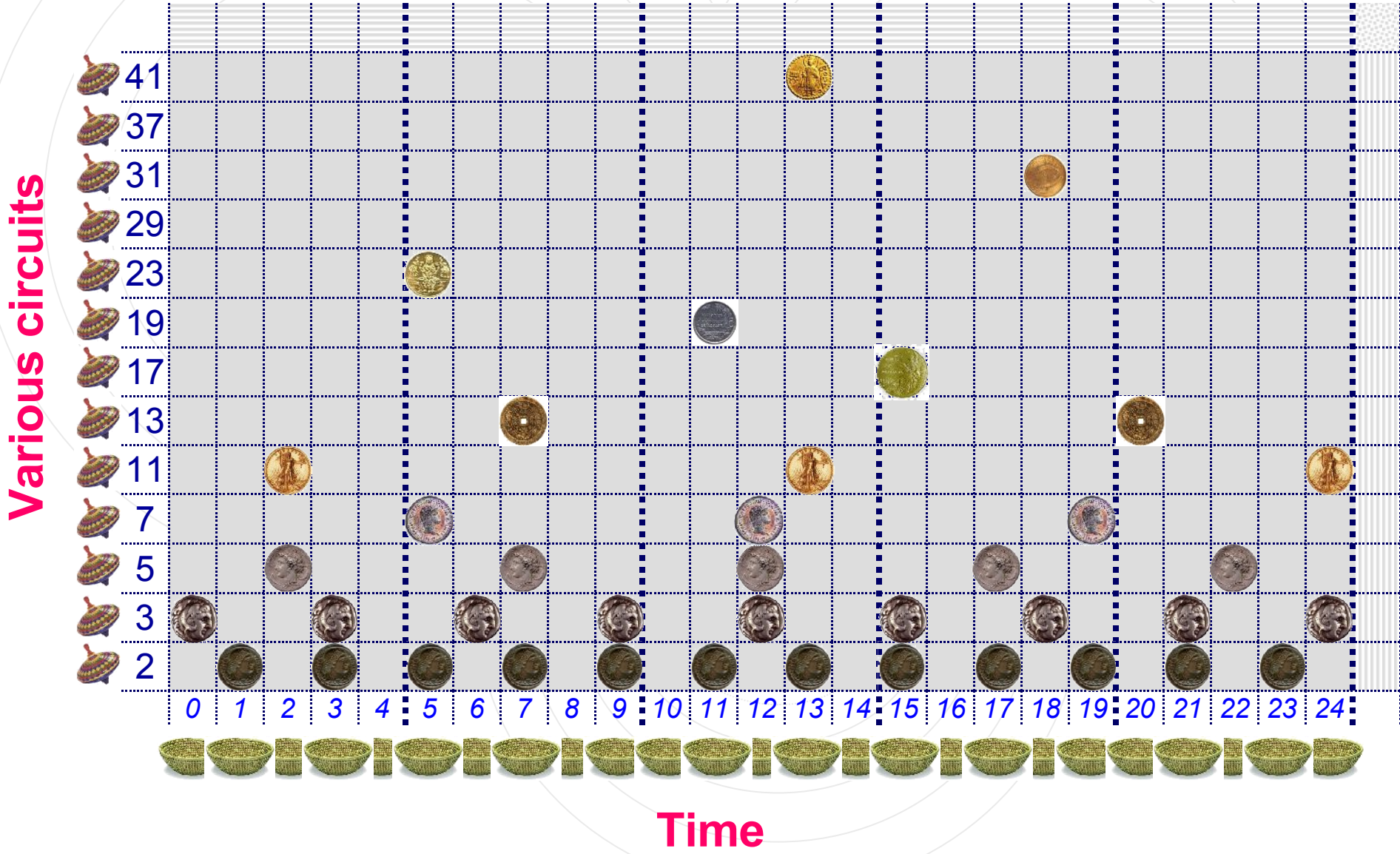
[Lenstra, Tromer, Shamir, Kortsmit, Dodson, Hughes, Leyland 2004]

TWIRL: TWINKLE with compressed time

- Uses the same time-space reversal as TWINKLE.
- Uses a pipeline (skewed local processing) instead of electro-optical phenomena (instantaneous global processing).
- Uses compact representations of the progressions (but requires more complicated logic to “decode” these representations).
- Runs 3-4 orders of magnitude faster than TWINKLE by parallelizing the handling of sieve locations: “compressed time”.

TWIRL: compressed time

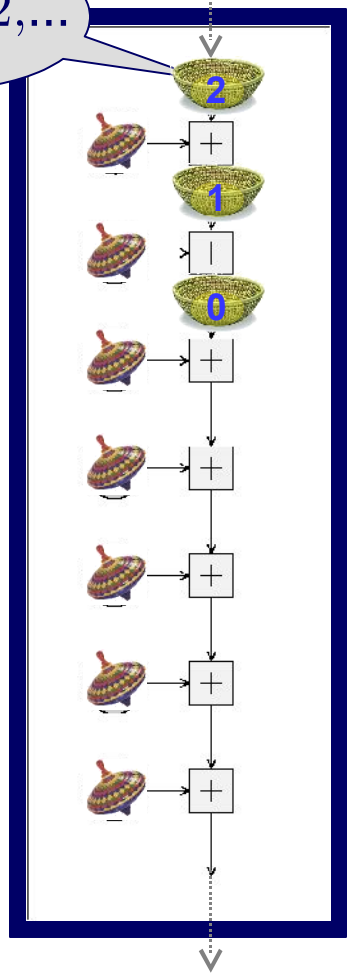
$s=5$ indices handled at each clock cycle. (real: $s=32768$)



Parallelization in TWIRL

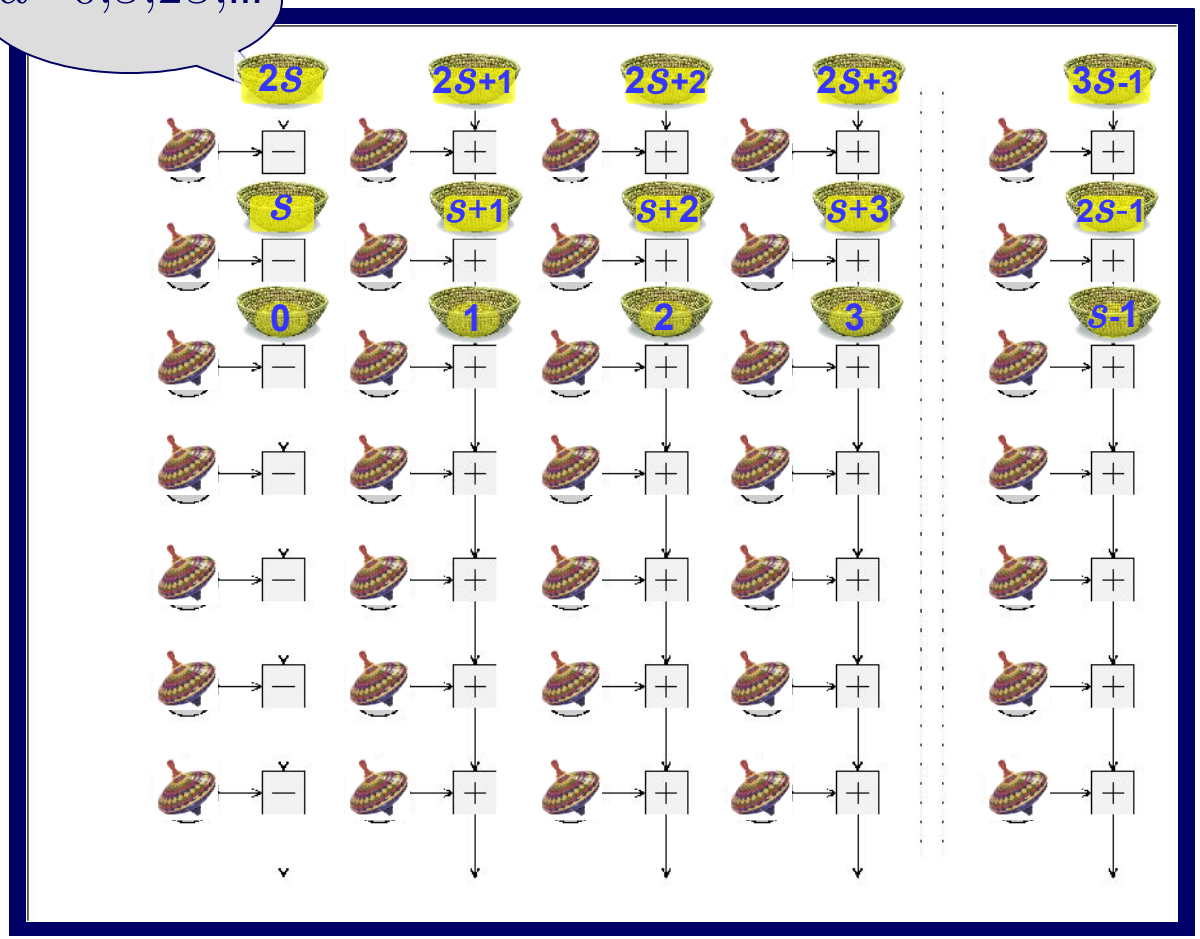
TWINKLE-like pipeline

$a=0,1,2,...$

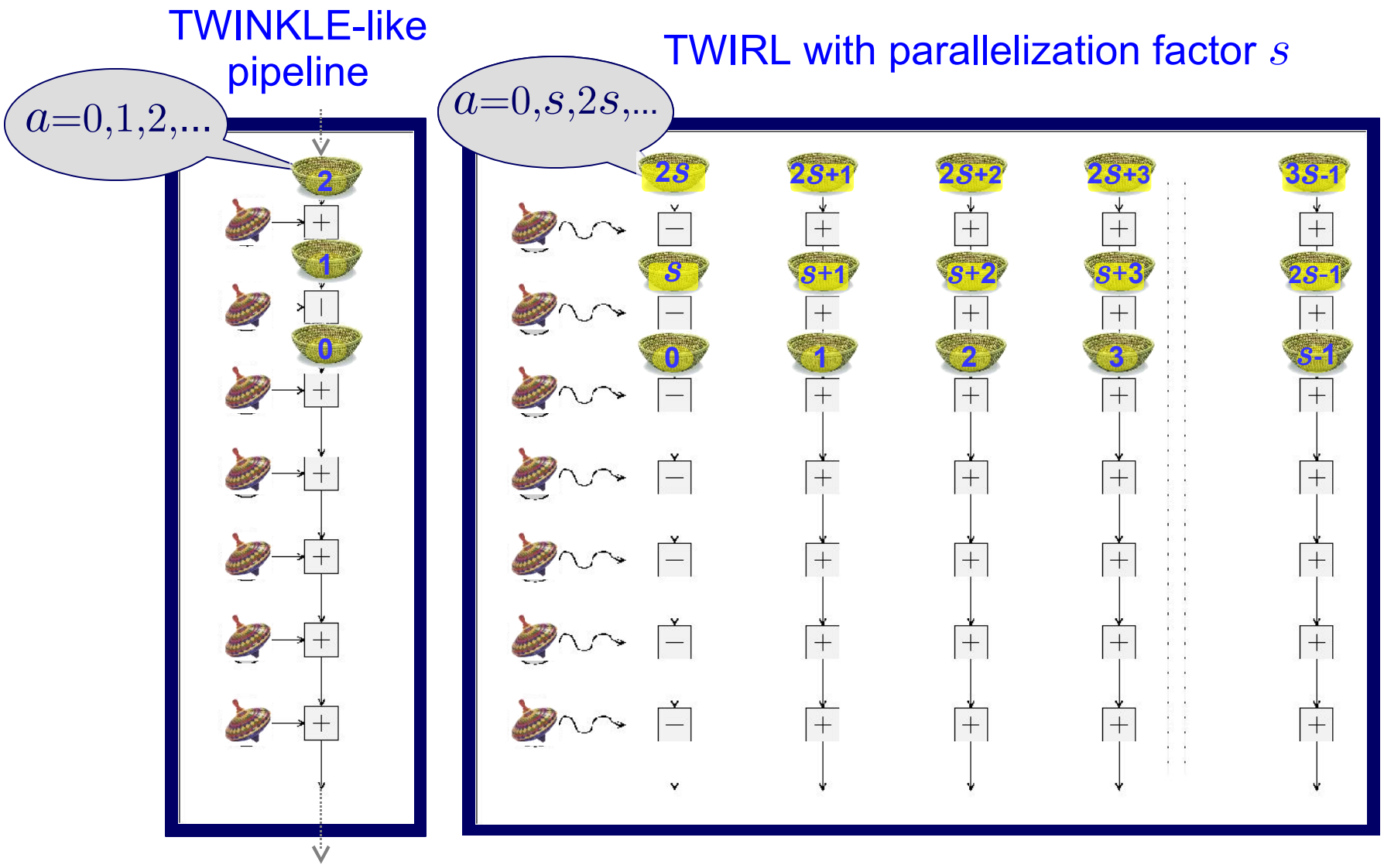


Simple parallelization with factor s

$a=0,s,2s,...$

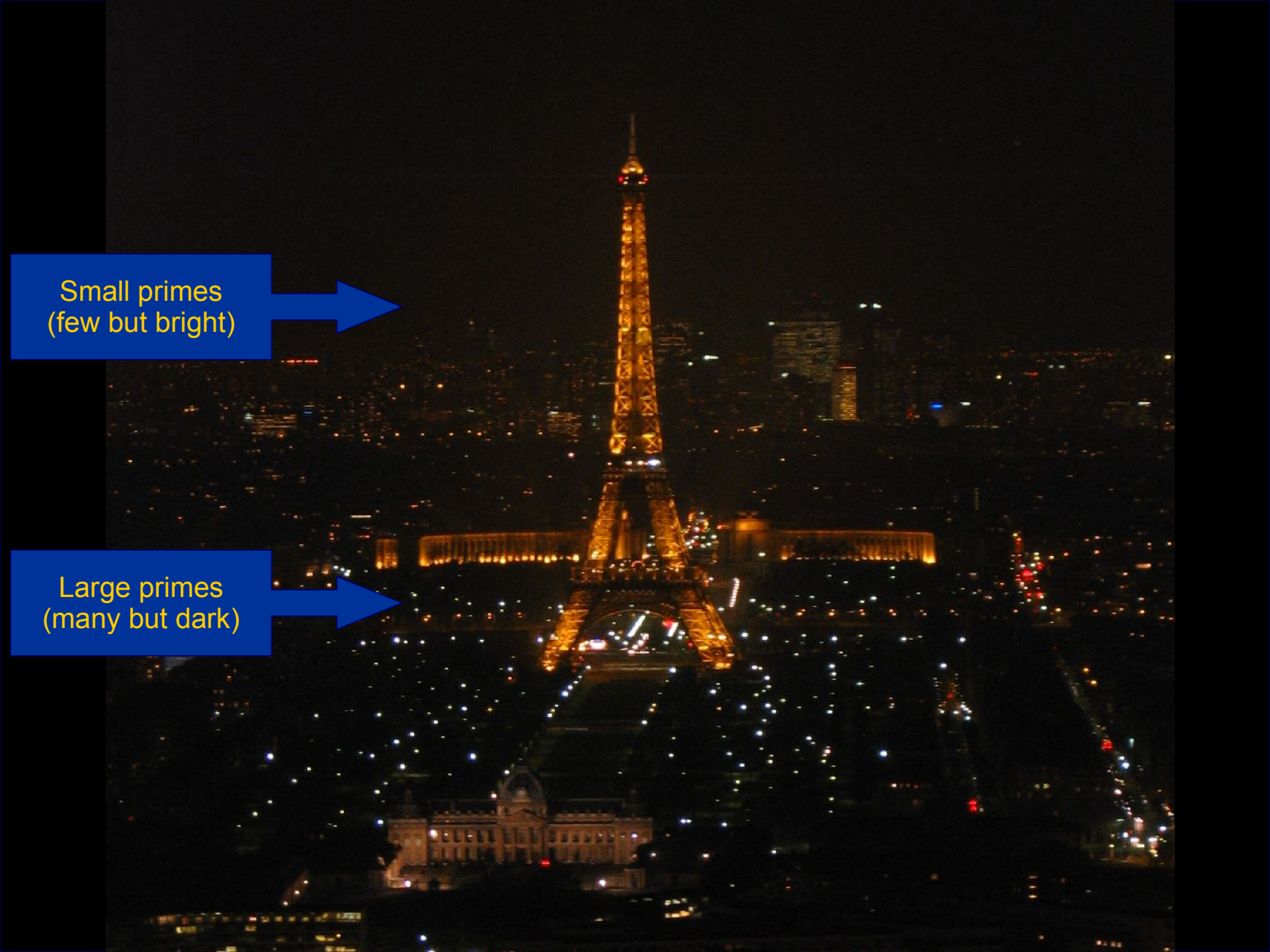


Parallelization in TWIRL



Heterogeneous design

- A progression of interval p makes a contribution every p/s clock cycles.
- There are a lot of large primes, but each contributes very seldom.
- There are few small primes, but their contributions are frequent.

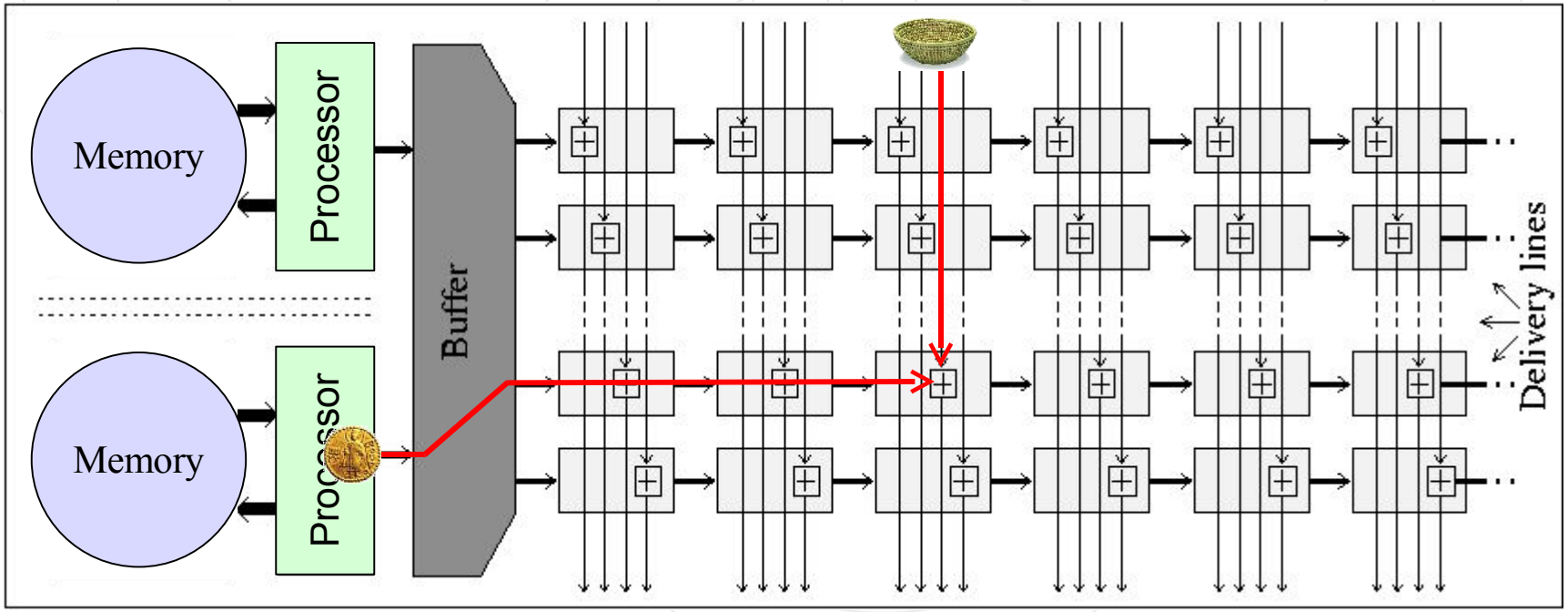

Small primes
(few but bright)
Large primes
(many but dark)

Heterogeneous design

We place several thousand “stations” along the pipeline. Each station handles progressions whose prime interval are in a certain range. Station design varies with the magnitude of the prime.

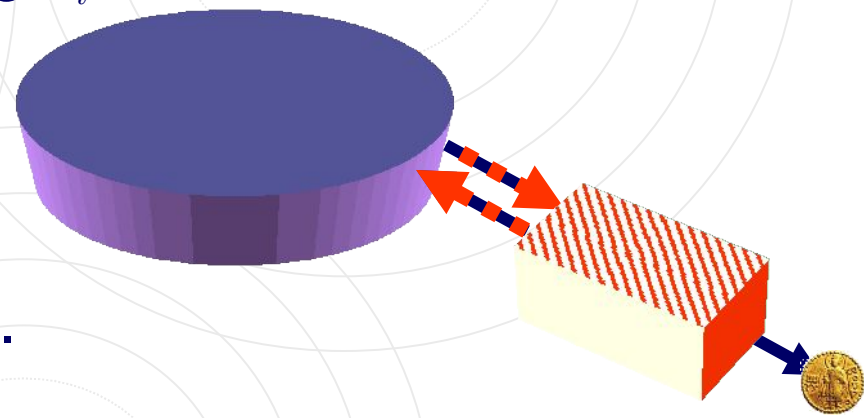
Example: handling large primes

- 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 many progressions. It computes and sends contributions across the bus, where they are added at just the right time. Timing is critical.



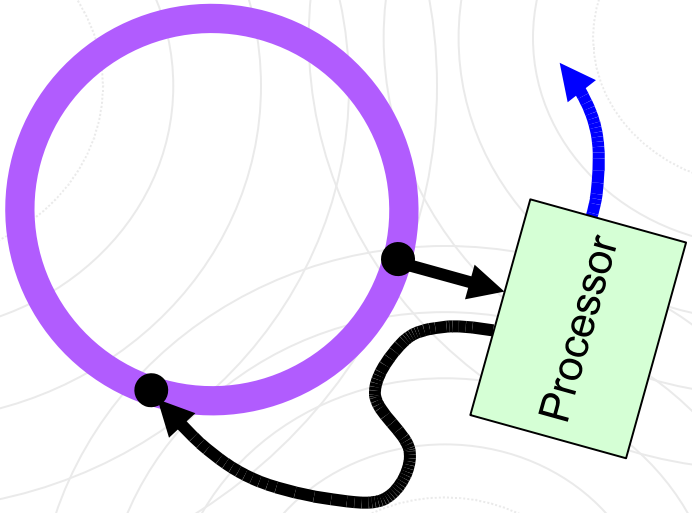
Implementing a priority queue of events

- 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.



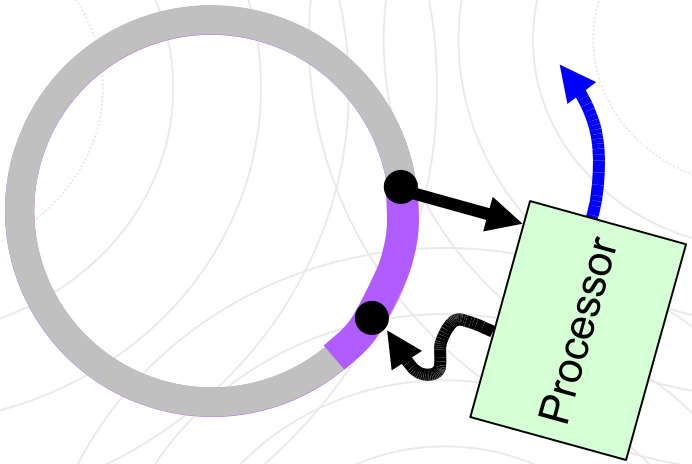
Handling large primes (cont.)

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



Handling large primes (cont.)

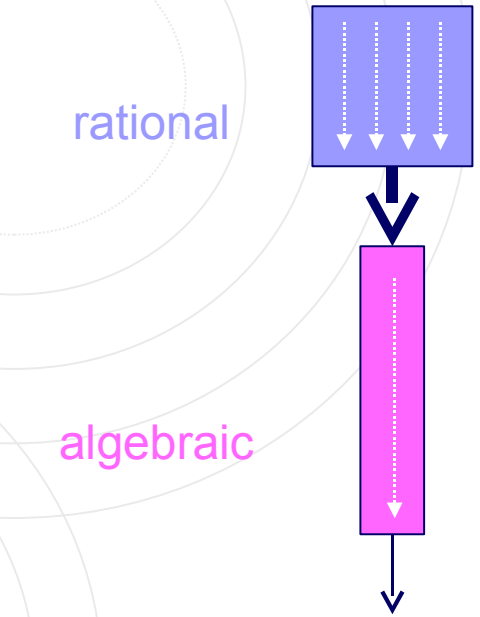
- The memory used by past events can be reused.
- Think of the processor as rotating around the cyclic memory:



- 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.



The wafer-scale TWIRL design has algorithmic-level fault tolerance:

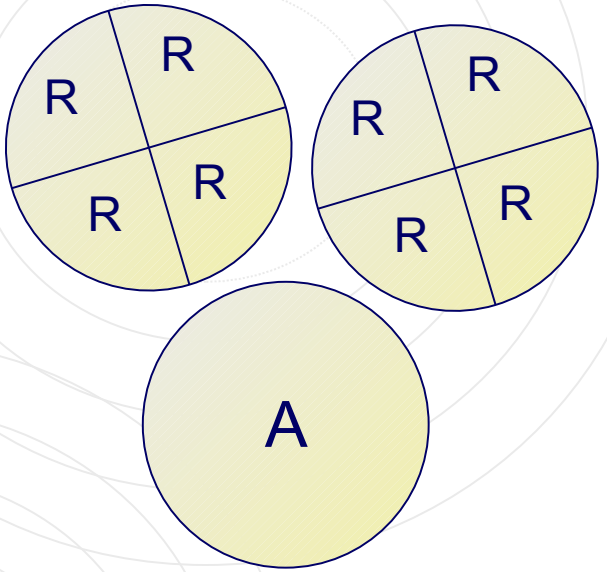
- Can tolerate false positives by rechecking on a host PC the smoothness of the reported candidates.
- Can tolerate false negatives by testing a slightly larger number of candidates.
- Can tolerate faulty processors and memory banks by assigning their primes to other processors of identical design.
- Can tolerate faulty adders and pipeline components by selectively bypassing them.

TWIRL for 1024-bit composites

(for 0.13μm process)

- A cluster of 9 TWIRLs

  on three 30cm wafers can process a sieve line (10^{15} sieve locations) in 34 seconds.



- 12-bit buses between R and A component.
- Total cost to complete the sieving in 1 year, use 194 clusters (<600 wafers): ~$10M (+ NRE).
- With 90nm process: ~1.1M.

Estimated recurring costs with current technology (US$×year)

	768-bit	1024-bit
Traditional PC-based	1.3×10^7	10^{12}
TWINKLE	8×10^6	
TWIRL	5×10^3	10^7 (10^6)

But: NRE, chip size…

Properties of TWINKLE

- Dissipates considerably less heat than TWINKLE, since each active logic element serves thousands of arithmetic progressions.
- 3-4 orders of magnitude faster than TWINKLE.
- Storage of large primes (sequential-access DRAM) is close to optimal.
- Can handle much larger B $\rightarrow$ factor larger composites.
- Enormous data flow banddwidth $\rightarrow$ inherently single-wafer (bad news), wafer-limited (mixed news).

Mesh-based sieving

[Bernstein 2001]

[Geiselmann, Steinwandt 2003]

[Geiselmann, Steinwandt 2004]

Mesh-based sieving

Processes sieve locations in large chunks.

Based on a systolic 2D mesh of identical nodes.

Each node performs three functions:

- Forms part of a generic mesh packet routing network
- In charge of a portion of the progressions.
- In charge of certain sieve locations in each interval of sieve locations.

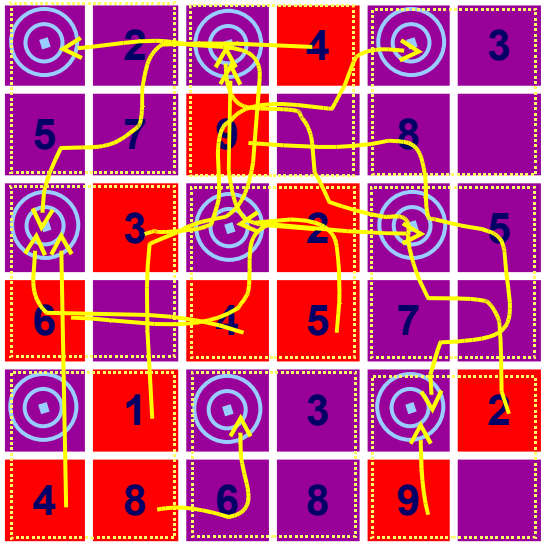
Mesh-based sieving: basic operation

For each sieving interval:

2. Each processor inspects the progressions stored within and emits all relevant contributions as packets: $(a, \log p)$
3. Each packet $(a, \log p)$ is routed, via mesh routing, to the mesh cell in charge of of sieve location a.
4. When a cell in charge of sieve location a receives a packet $(a, \log p)$, it consumes it and add $\log p$ to an accumulator corresponding to a (initially 0).
5. Once all packets arrived, the accumulators are compared to the threshold.

Mesh sieving (cont.)

- In mesh-based sieving, we route and sum progression contributions to sieve locations.
- In mesh-based linear algebra, we route and sum matrix entries multiplied by old vector entries to new vector entries.
- In both cases: balance the cost of memory and logic.



Mesh sieving – enhancements

- Progressions with large intervals represented using compact DRAM storage, as in TWIRL (+compression).
- Efficient handling of small primes by duplication.
- Clockwise transposition routing.
- Torus topology, or parallel tori.
- Packet injection.

Estimated recurring costs with current technology (US$×year)

	768-bit	1024-bit
Traditional PC-based	1.3×10^7	10^{12}
TWINKLE	8×10^6	
TWIRL	5×10^3	10^7 (10^6)
Mesh-based	3×10^4	

But: NRE, chip size…

Properties of mesh-based sieving

- Uniform systolic design
- Fault-tolerant at the algorithm level (route around defaults).
- Similarity to TWIRL: 2D layout, same asymptotic cost, heterogeneous bandwidth-limited.
- Subtle differences: storage compression vs. higher parallelism, chip uniformity.

Estimated recurring costs with current technology (US$×year)

	768-bit	1024-bit
Traditional PC-based	1.3×10^{7}	10^{12}
TWINKLE	8×10^{6}	
TWIRL	5×10^{3}	10^{7} (10^{6})
Mesh-based	3×10^{4}	
SHARK		2×10^{8}

But: NRE, chip size, chip transport networks…

Conclusions

- Special-Purpose Hardware provides several benefits:
  - Reduced overhead
  - Immense parallelism in computation and transport
  - Concrete technology-driven algorithmic optimization
- Dramatic implications for 1024-bit composites.
- But: larger composites necessitate algorithmic advances.