



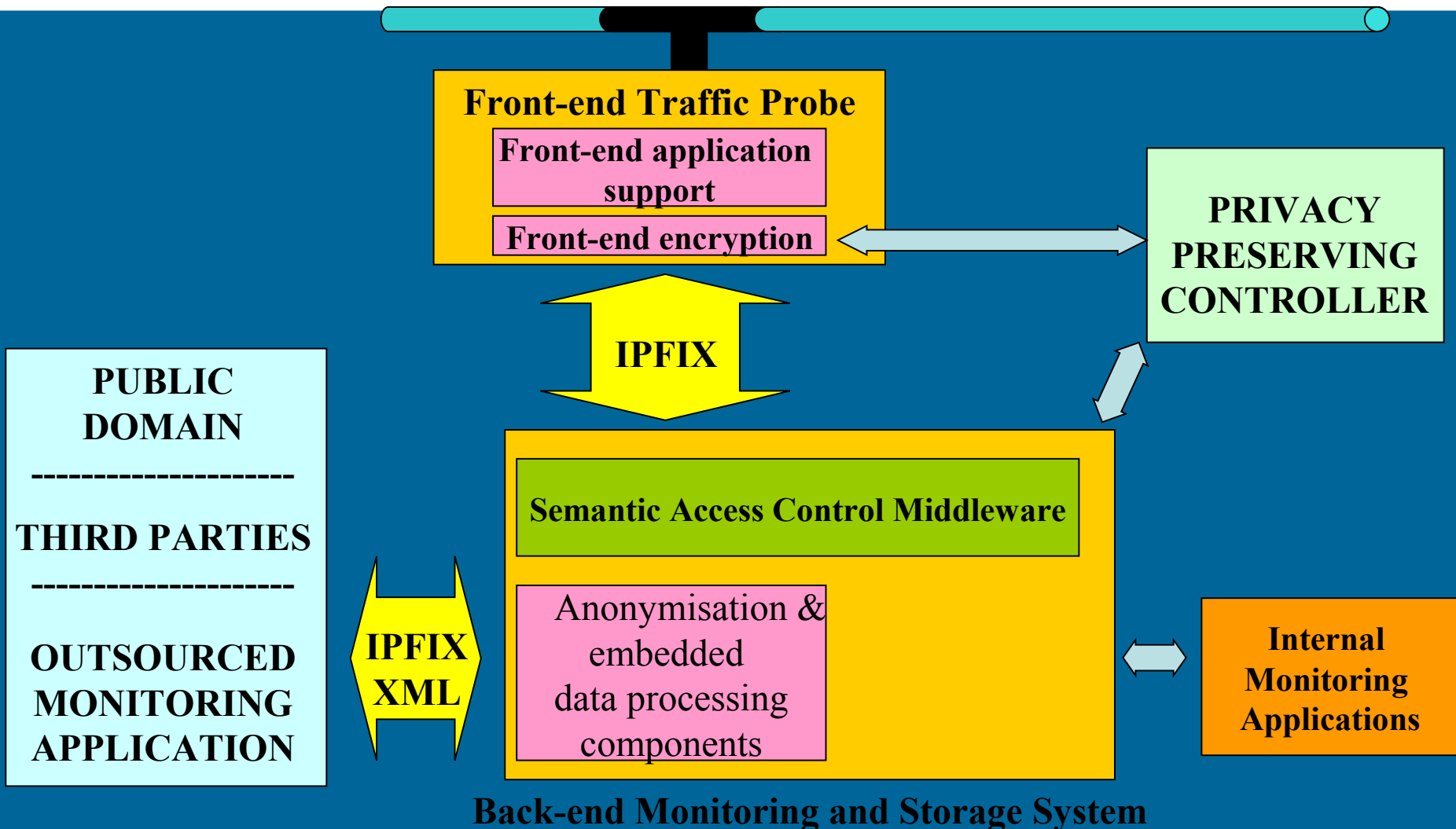
# Fast, Scalable and privacy-preserving data processing for “on-link” network monitoring: The PRISM

Giuseppe Bianchi **approach**  
CNIT / Roma Tor Vergata Unit - Italy  
[giuseppe.bianchi@uniroma2.it](mailto:giuseppe.bianchi@uniroma2.it)

# Why Privacy in Network Monitoring

- Massive amount of data delivered
  - Data mining
  - Statistical analysis
  - Behavioural profiling
  - Data misuse
- Nothing in society poses as grave a threat to privacy as the Internet Service Provider (ISP). ISPs carry their users' conversations, secrets, relationships, acts, and omissions. Until the very recent past, they had left most of these alone because they had lacked the tools to spy invasively, **but with recent advances in eavesdropping technology, they can now spy on people in unprecedented ways.** Meanwhile, advertisers and copyright owners have been tempting them to put their users' secrets up for sale, and judging from a recent flurry of reports, ISPs are giving in to the temptation and experimenting with new forms of spying. **This is only the leading edge of a coming storm of unprecedented and invasive ISP surveillance.**
  - *Source: Paul Ohm, The Rise and Fall of Invasive ISP Surveillance, University of Illinois Law Review, 2009*
- Another privacy concern repeatedly mentioned to the European Commission these days is **behavioural advertisement**: systems that monitor internet users' web browsing to better target them with advertisements. Now, **European Privacy rules are crystal clear: a person's information can only be used with their prior consent. And we cannot give up this basic principle, and have all our exchanges monitored, surveyed and stored, in exchange for a promise of "more relevant" advertisement!** The Commission is closely monitoring the use of behavioural advertising to ensure respect for our privacy rights. I will not shy away from taking action where an EU country falls short of this duty.
  - *Source: Viviane Reding's weekly videomessage theme: protecting privacy in the digital age, April 14, 2009*

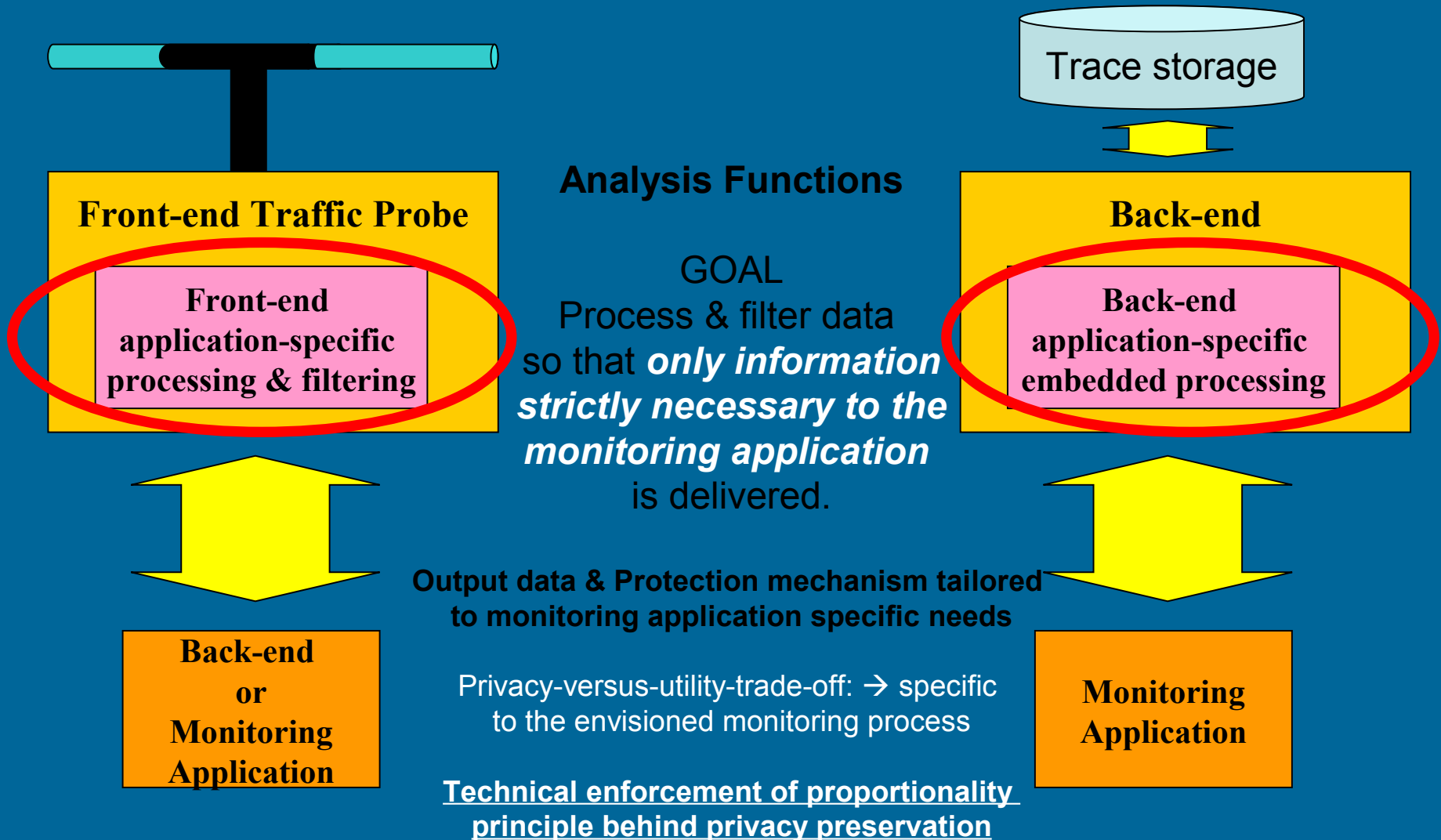
# Two-tiered Approach



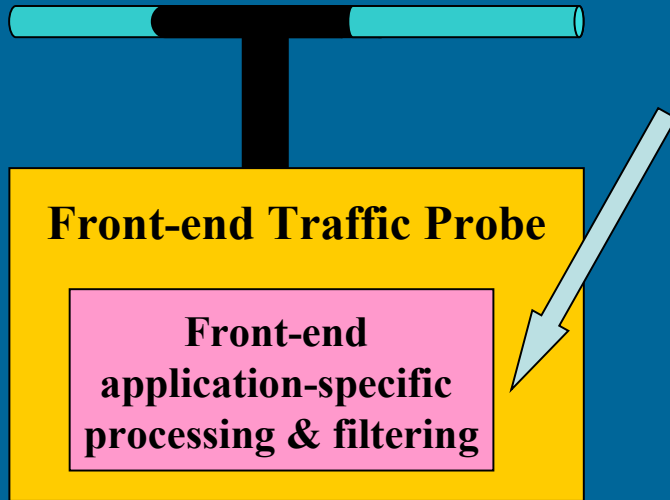
# The 4 PRISM research directions

- As much traffic analysis (and output filtering) as possible directly on the probe
  - Obvious scalability & privacy gain
- per flow encryption + escrow mechanisms
  - Decrypt only “relevant” flows by expressing decrypt conditions in terms of monitoring events
- Privacy Preserving Access Control
  - Access only data types “permitted” for a specific monitoring task
- Anonymization

# “matched monitoring” concept



# PRISM achievements in FE processing

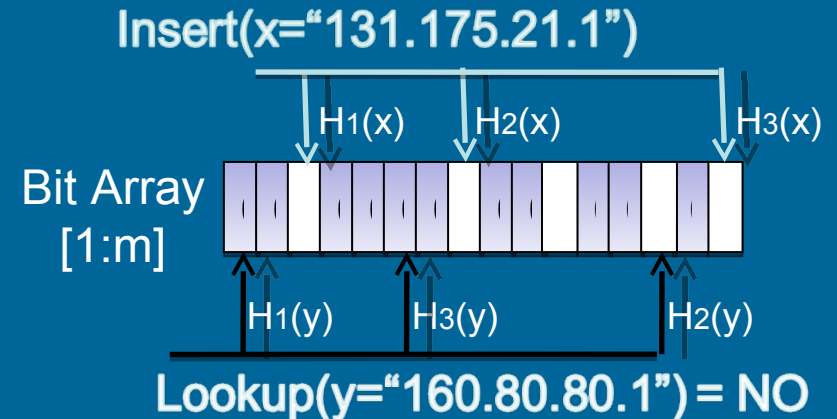


- Question
  - Which traffic analysis primitives, largely exploited as basic blocks in monitoring apps, can effectively be supported on front-end filters?
  - Constraint: stateless, on the fly, per packet processing

- PRISM contribution:
  - Bloom-filter based approaches
    - trade performance and scalability with (bounded) approximation
  - Front End adaptation for variety of apps
    - Including hardware Front-End pre-filter for SNORT

# Background: Bloom filters

- Bloom Filters
  - Probabilistic data structure for storing and checking set membership
  - $O(1)$  insert/lookup
  - Trade memory for certainty
    - memory  $\ll$  list
      - $n \times \text{sizeof}(\text{element})$
    - May provide WRONG response
      - False positive probability
- Counting Bloom Filters (CBFs)
  - Allow element deletion
- CBFs with conservative update
  - Permit **extremely** efficient counting and accounting (Estan, Varghese 2002)

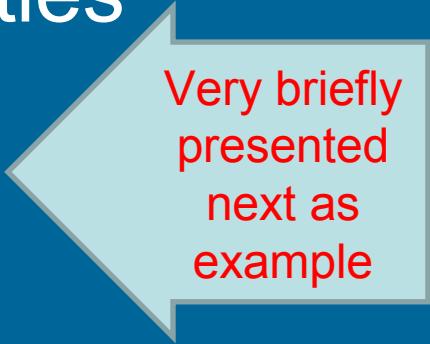


Bit array  $\rightarrow$  Counters array

Increment only smaller bin

# PRISM FE monitoring advances

- New, more efficient, data structures
  - Multi-layer compressed counting Bloom filters
  - Blooming trees & optimised blooming trees
  - Improving deterministic finite automata
- New FE monitoring functionalities
  - Rate Metering
  - Scan detection

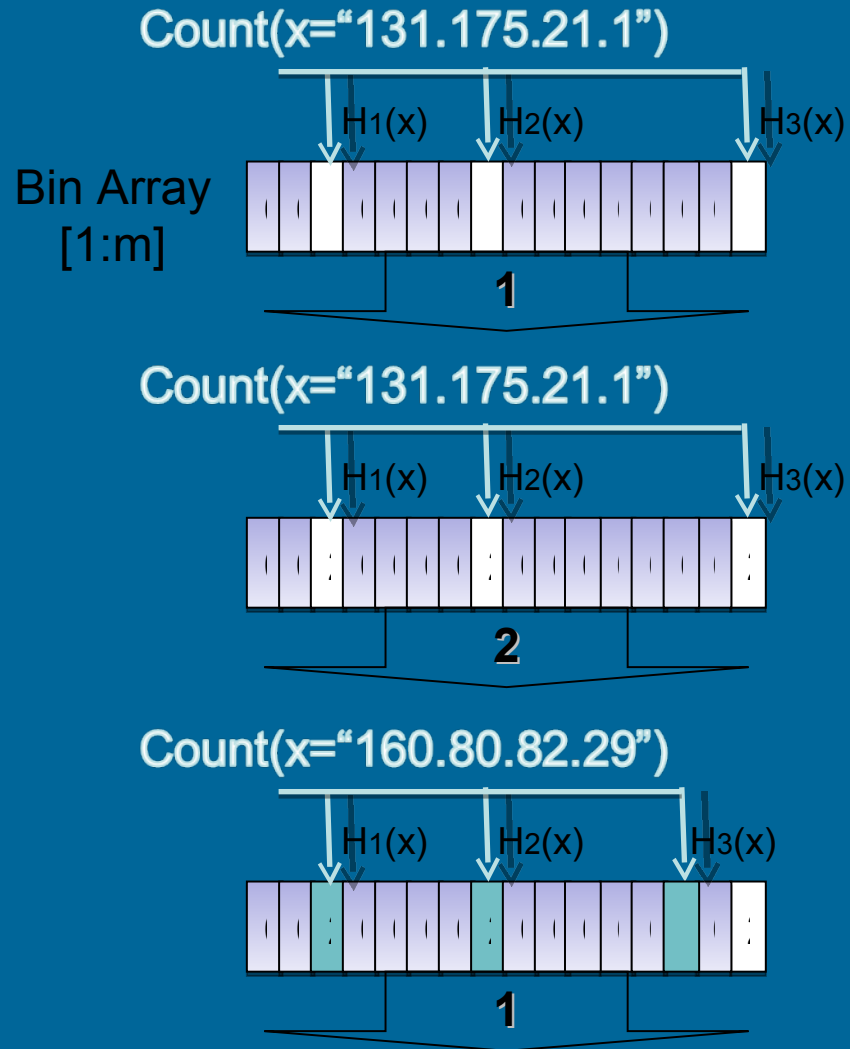


Very briefly  
presented  
next as  
example



# From counting to rate monitoring

- Counting Bloom filters with conservative update
  - Very efficient (only min bin ++)
- But...
  - Require a time window
    - too short → flows span over multiple windows
    - too long → memory consumption
- Our achievement:
  - From count to rate measurement!
  - Surprise: simpler than expected!

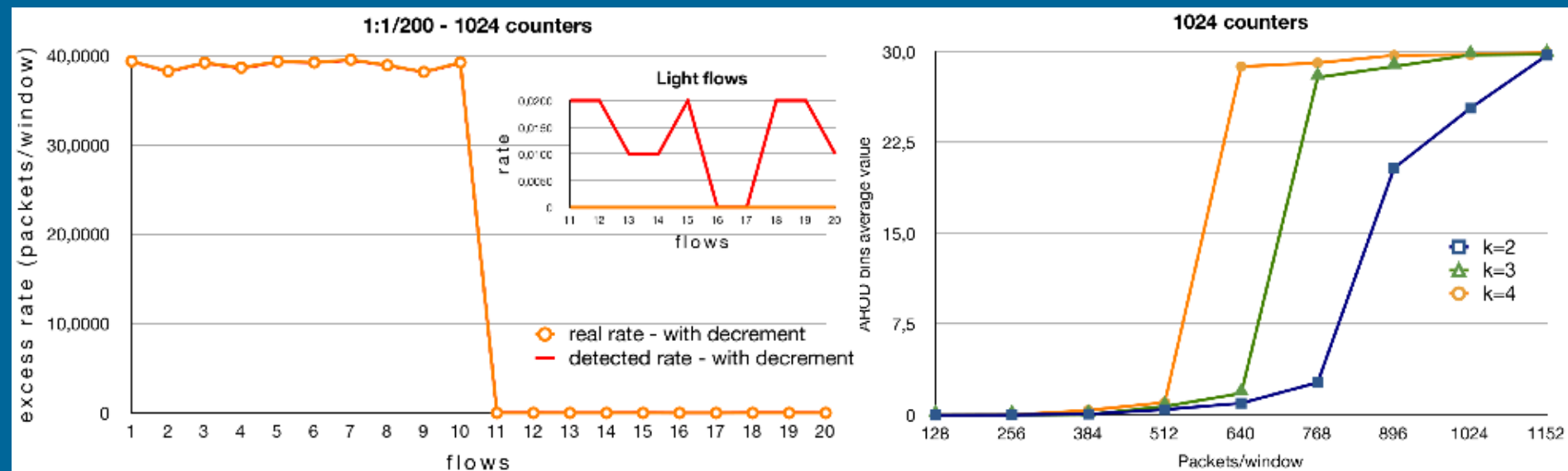


# Approximate token bucket

- Idea: decrement all bins periodically
  - Proof: exact operation!
- Result:
  - Bins decrement rate
    - Threshold on average rate overflow
      - Example: decrement window = 0,2 seconds → rate overflow when long term rate is greater than 5p/second offered
  - Bins size
    - Threshold on peak rate
      - Example: bins = 4 bits → overflow detected when 16 packets in a window, 17 in two consecutive windows, 18 in three, etc. (detected traffic envelope)
    - Equivalent to token bucket with same rate and peak parameters!
      - But approximate operation
      - False positive → counting errors (always in excess, though)

# Performance and efficiency

- With marginal memory deployed:
  - Excellent measuring accuracy
  - Guaranteed stable operation when  $\# \text{ packets/window} < \text{array size} / k$ 
    - But results show more than double sustained load
- Example
  - 1 Gbps link, fully loaded with 500 byte packets at wire speed
  - Target rate overflow detection = 2 Mbps  $\rightarrow$  1024 counters only!



# Detecting scans

- ARP scans
- Host/Port scans
- ICMP scans
- ...
- Patterns frequently appearing in DDoS and as preparation of attacks, ...
- Problem largely different from that solved with BF and CBF
  - Or not?
  - Can we design a front-end on the fly scan detector?

... MACsrc=F1, .... targetIP=X1,...

... MACsrc=F2, .... targetIP=Y,...

... MACsrc=F1, .... targetIP=X2,...

... MACsrc=F2, .... targetIP=Y,...

... MACsrc=F1, .... targetIP=X3,...

... MACsrc=F3, .... targetIP=Z,...

... MACsrc=F3, .... targetIP=Z,...

... MACsrc=F1, .... targetIP=X4,...

... MACsrc=F2, .... targetIP=Y,...

... MACsrc=F2, .... targetIP=Y,...

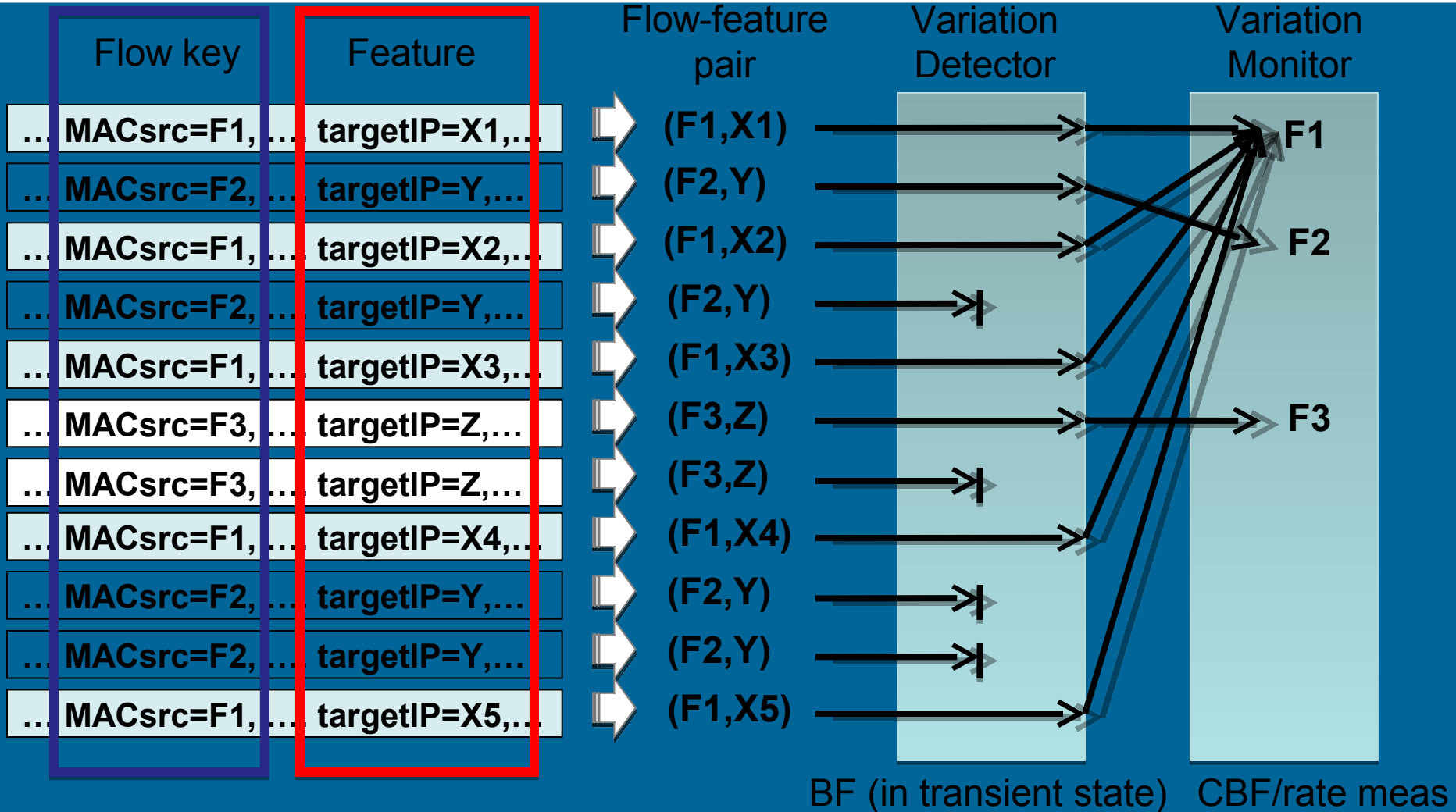
... MACsrc=F1, .... targetIP=X5,...

**F1 = 5 “variations” → suspected scan**

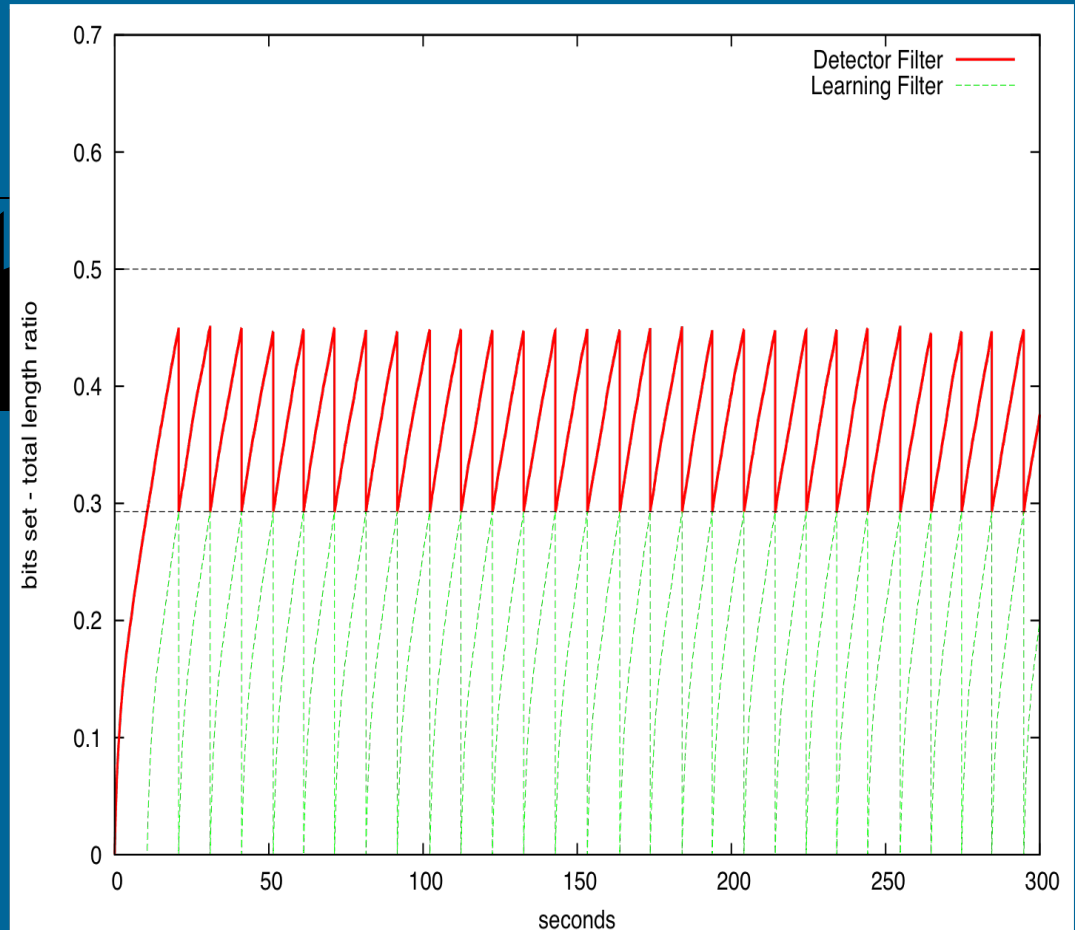
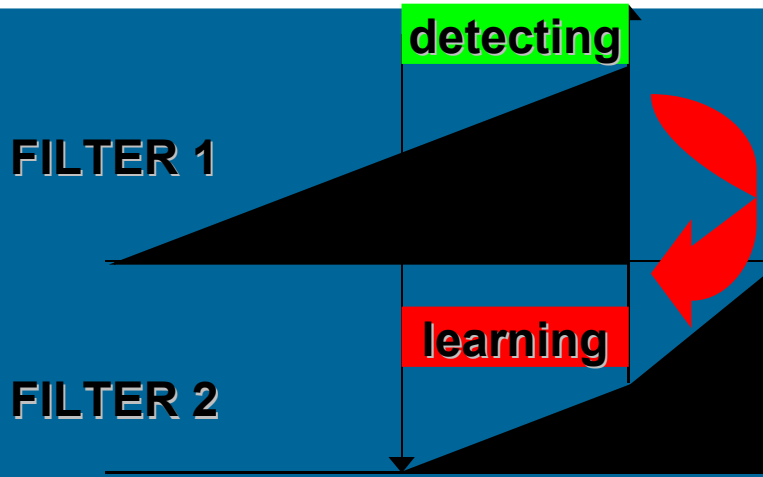
**F2 = 1 “variation”**

**F3 = 1 “variation”**

# Scan detect = count variations



# Variation detection: Coupled BFs



Filter switching rule (static):  
When # learning zeros drops to:

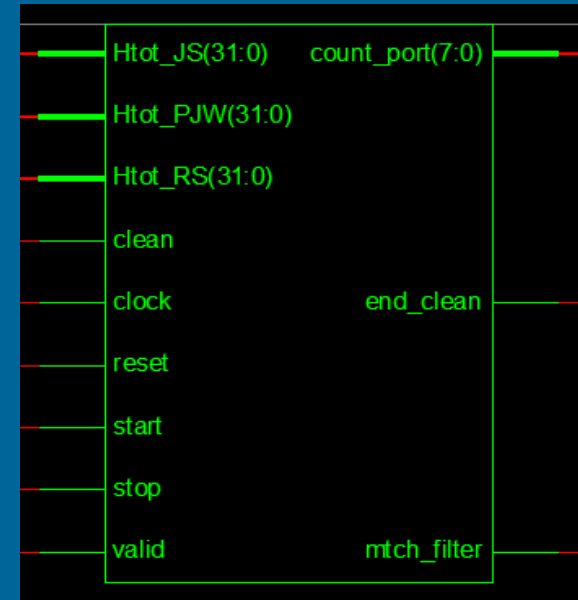
$$B_0(n/2) = \frac{m}{\sqrt{2}}$$

Adaptive improvement:

$$T(t+1) = T(t) \sqrt{\frac{m/2}{D(t)}}$$

# Variation Detector: HW impl.

- BF Implemented as a RAM
  - Delay for filter reset (1 ram access every CLK period)
- Three BFs
  - “cleaning” filter needed
    - (BF reset is slow → RAM)
- Three simple hash function used (General hash function)
  - RS hash (Robert Sedgwicks)
  - JS hash (Justin Sobel)
  - PJW hash (Peter J. Weinberger)
- CBF implementation ongoing



# HW Front-End for SNORT

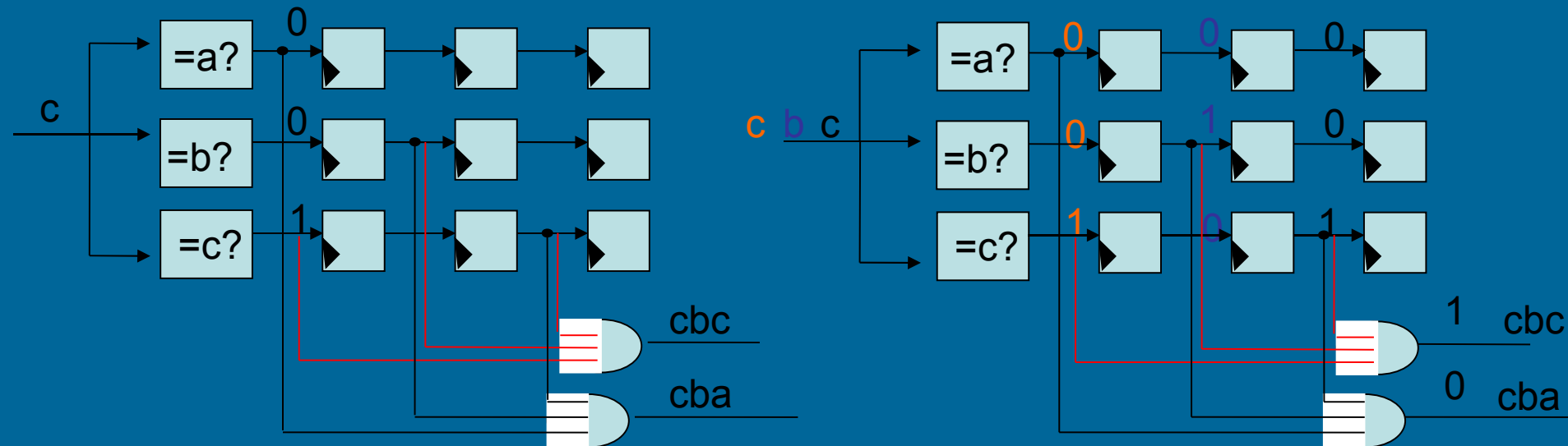
- Content + Uricontent match
  - String matching: Bloom filter or exact approach
- PCRE → DFA, NFA
  - Fact: most rules use PCRE only after content/uricontent match
  - Or only for very specific traffic (e.g., RPC) which can be detected with other rules
- TCP/HTTP reassembly
  - Apparently THE major problem (HW would not scale)
    - But fact 2: full reassembly NOT necessary!
      - ad hoc solutions being considered (ACK or 3-way handshake reconstructions)
    - And partial signature matching viable





# HW approaches

- Bloom filter based:
  - Memory efficient, but not really HW-amenable: BF in RAM
- Exact string matching
  - An AND port for every content to match
  - A flip-flop chain to keep trace of old byte in the packet
  - not as inefficient as initially thought!
  - ++: Support for more complex snort rules
  - ++: Implementation easily extended to regular expression matching



# Content match with modifiers

- Proposed approach (including registers and counters) support rules with modifiers
  - offset, depth, distance, within*

content:"HTTP/1.1 200 OK";

→ HW implem. (VHDL)

**Rule #1:**  
depth:15;

content:"Index of /"; within:200;

```
if (count_payload<=15 and decodifica(14)(72)='1' and decodifica(13)(84)='1' and
decodifica(12)(84)='1' and decodifica(11)(80)='1' and (codifica(10)=x"2F") and
decodifica(9)(49)='1' and (codifica(8)=x"2E") and decodifica(7)(49)='1' and
(codifica(6)=x"20") and decodifica(5)(50)='1' and decodifica(4)(48)='1' and
decodifica(3)(48)='1' and (codifica(2)=x"20") and decodifica(1)(79)='1' and
decodifica(0)(75)='1') then
stringa_trovata(99)(1)<='1';
register_99_1<=count_payload;
end if;
```

**Rule #2:**

```
if (count_payload-register_99_1<=200 and stringa_trovata(99)(1)='1' and decodifica(9)
(73)='1' and decodifica(8)(110)='1' and decodifica(7)(100)='1' and decodifica(6)
(101)='1' and decodifica(5)(120)='1' and (codifica(4)=x"20") and decodifica(3)(111)='1'
and decodifica(2)(102)='1' and (codifica(1)=x"20") and (codifica(0)=x"2F")) then
trovata<='1';
end if;
```

# Current Implementation

- NetFPGA (xilinx Virtex X2V50)
  - 1 gbps link speed
- content match without modifiers
  - All content in the snort rule set (!)
  - String truncated to 40 bytes
- content match with modifiers
  - 400 SNORT rules
  - (we estimate 2000 rules capacity)

Logic utilization	used	available	utiliz.
Number of slices	10059	23616	42%
Number of slice Flip Flops	5369	47232	11%
Number of 4 input LUTs	16713	47232	35%

Logic utilization	used	available	utiliz.
Number of slices	3787	23616	16%
Number of slice Flip Flops	3375	47232	7%
Number of 4 input LUTs	5131	47232	10%

# With more performing COTS HW

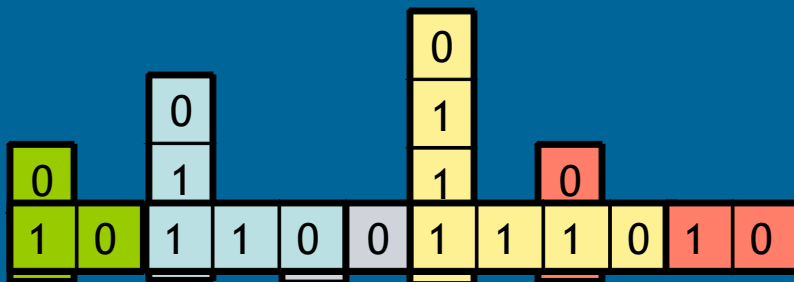
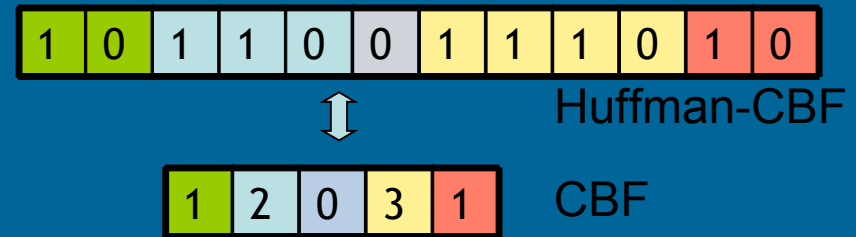
- With up to date FPGA (e.g. Virtex V TXT):
  - provides 30X logic resources
  - achieve 600 MHz (4,8X) data processing
  - Support for higher data rate (10 Gbps → 100)
- Parallelized version of our preliminary prototype should process 2000 rules at
  - 1Gbps X 4.8 (speed factor) X 30 (paral. factor)= scaling to 144Gbps seems easy

# Conclusions

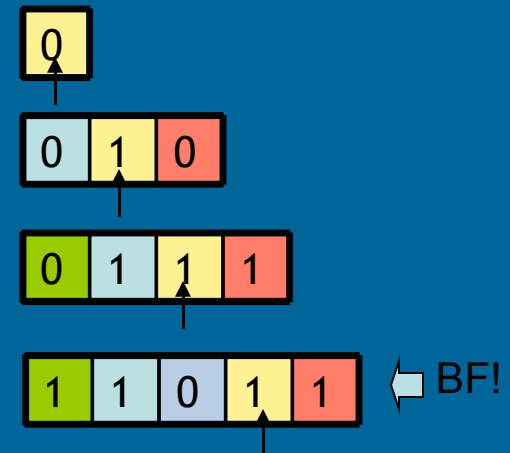
- **Front-End analysis capabilities:**
  - Functionalities well beyond our initial expectation
- **Application-matched FE processing**
  - Strong data reduction → scalability
  - Matched data reduction → technical enforcement of privacy's necessity principle
- **Privacy preservation**
  - Specific privacy preserving solutions addressed
    - e.g. crypto
  - But the aftermath is: privacy comes as “side effect”!

# Multi-layer compressed CBF

- Idea 1: Huffman-encode CBF
  - $0 \rightarrow 0$ ;  $1 \rightarrow 10$ ;  $2 \rightarrow 110$ ;  $3 \rightarrow 1110$ ; ...
  - Apparently not convenient, but...
- Fact 2: Efficient “popcount” operation native in NP or CPU
- Fact3: Lookup much more frequent than insertions/deletions
- **Idea: Multilayer structure**



Lookup example:  
 $h(x) = 3$   
 $\text{popcount}(110) = 2$   
 $\text{popcount}(10) = 1$   
 $\text{popcount}(0) = 0$   
 $\text{counter} = 1110 = 3$



# ML-CCBF savings

