Network Monitoring as a Streaming Analytics Problem

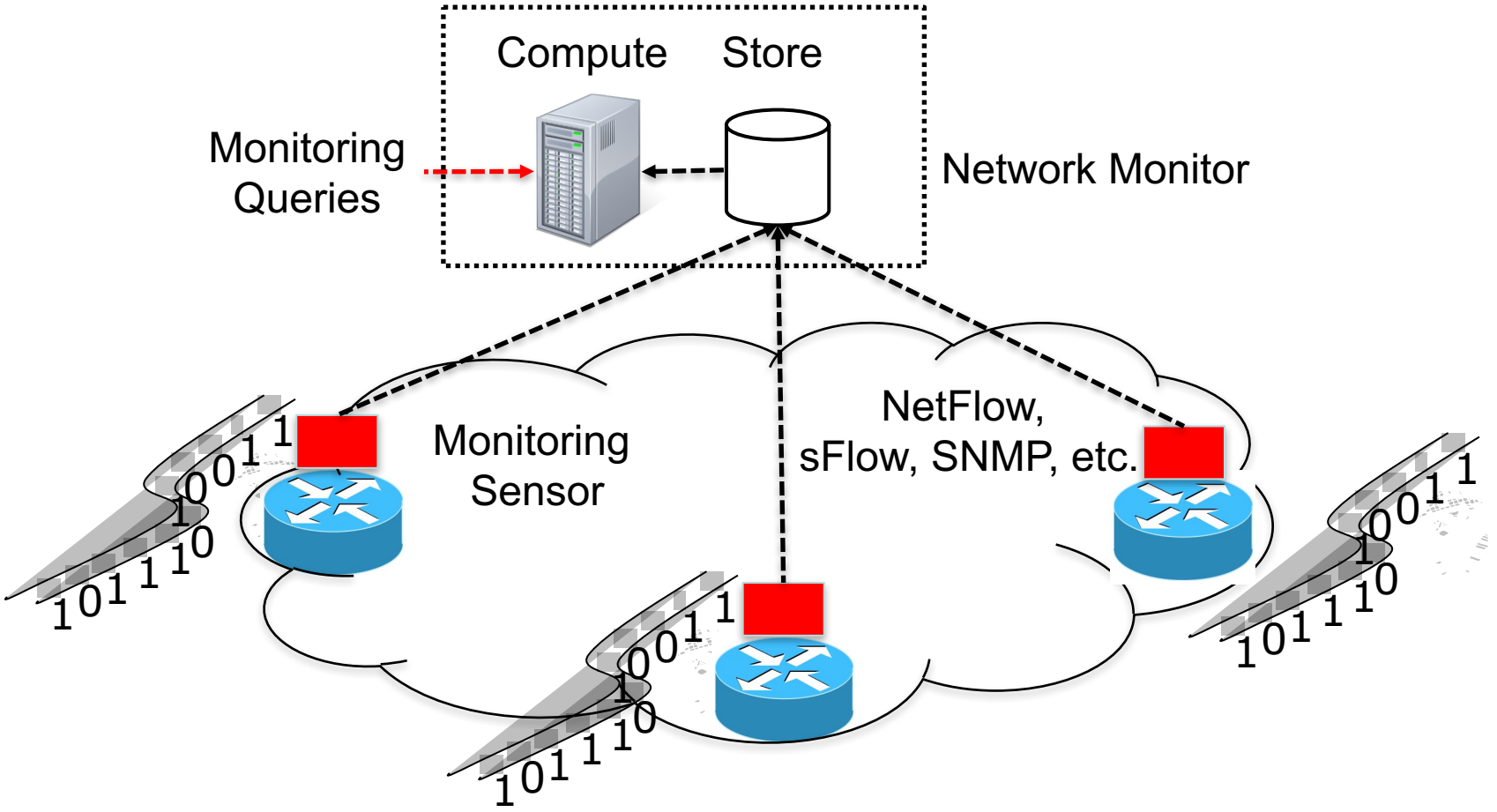
Arpit Gupta

Princeton University

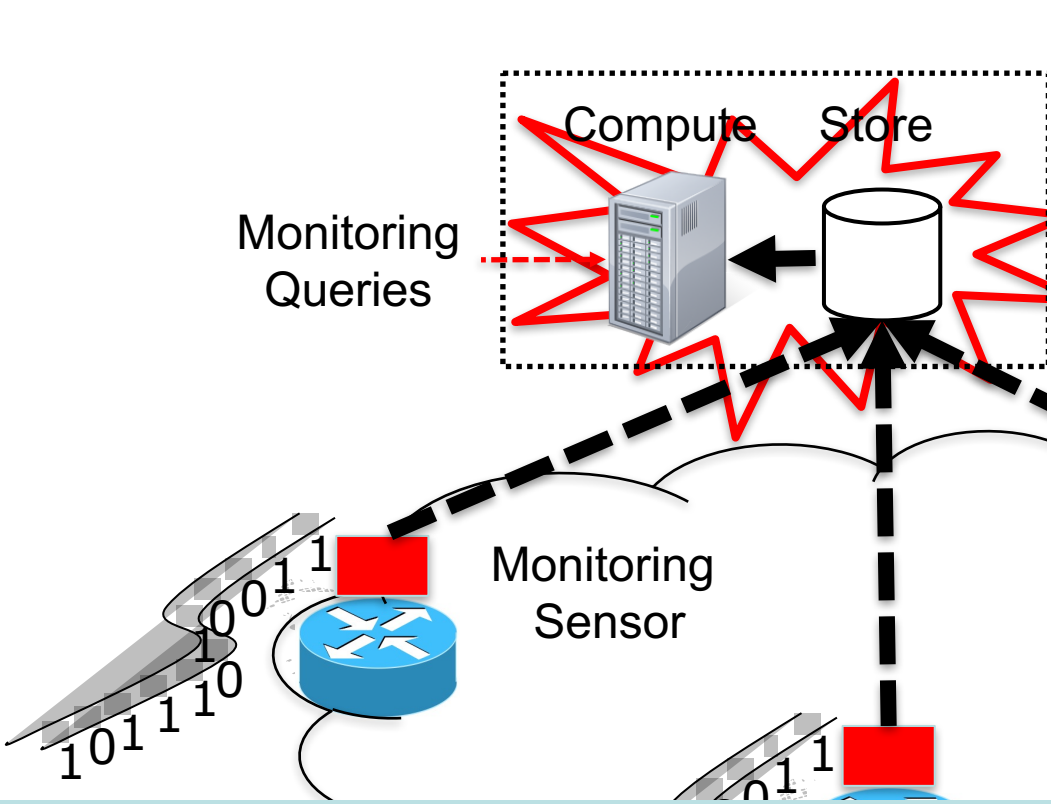
Rüdiger Birkner, Marco Canini, Nick Feamster,

Chris Mac-Stoker, Walter Willinger

Conventional Network Monitoring

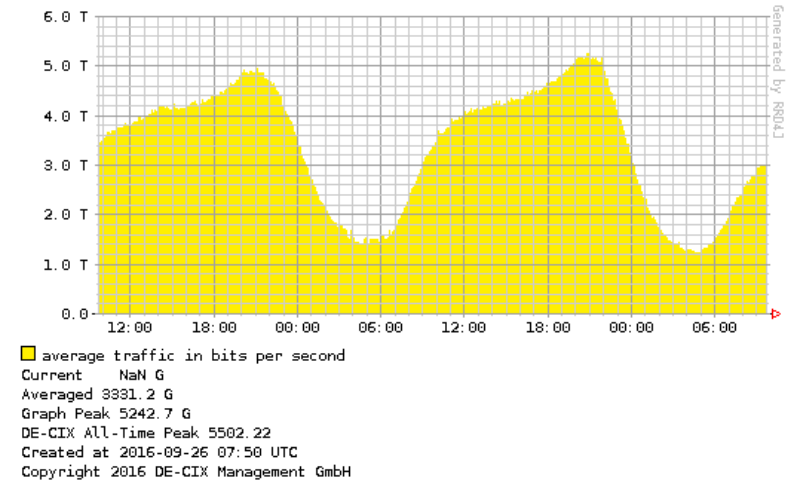


Big (“Internet”) Data



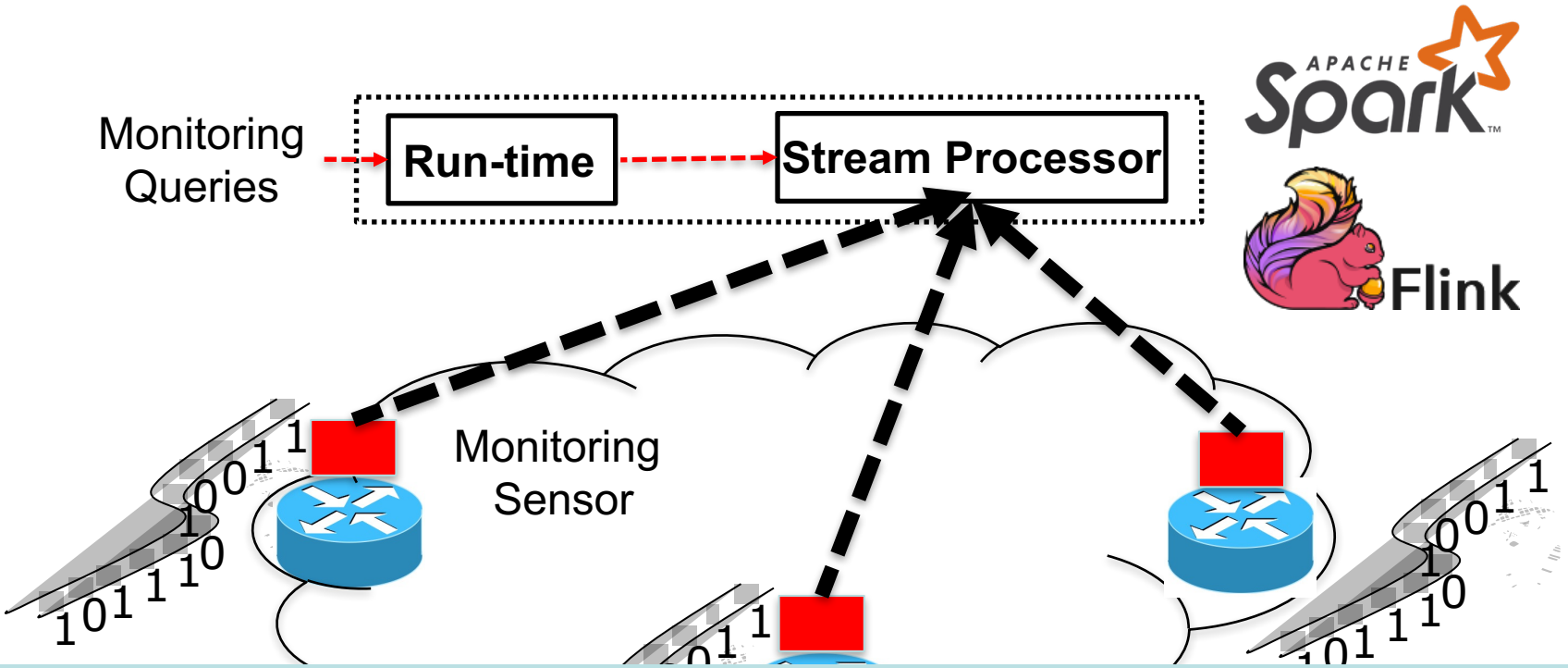
2-day graph

Source: DE-CIX statistics



Not suited for large networks & real-time monitoring applications

Network Monitoring as Streaming Analytics Problem



Is using state-of-art stream processor good enough solution?

Is it Good Enough?

- Use Case:
  - Reflection Attack Monitoring Query

    Detect hosts for which # of unique source IPs sending UDP response messages exceeds threshold

Is it Good Enough?

- Use Case:
  - Reflection Attack Monitoring Query

    Detect hosts for which # of unique source IPs sending UDP response messages exceeds threshold

```
victimIPs =
  pktStream.window(W).transform { wndPkts =>
    wndPkts.filter(p => p.proto == 17)
    .map(p => (p.dIP, p.sIP)).distinct
    .map((dIP, sIP) => (dIP, 1))
    .reduceByKey(sum)
    .filter((dIP, count) => count > T)
    .map((dIP, count) => dIP)
  }
```

Is it Good Enough?

- Use Case:
  - Reflection Attack Monitoring Query

    Detect hosts for which # of unique source IPs sending UDP response messages exceeds threshold
  - Use two hour IPFIX data trace from a large IXP

- Prohibitively Costly:
  - Packet Processing Cost: requires processing **220 M** packets per second

How can we bring down these costs?

Idea 1: Iterative Query Refinement

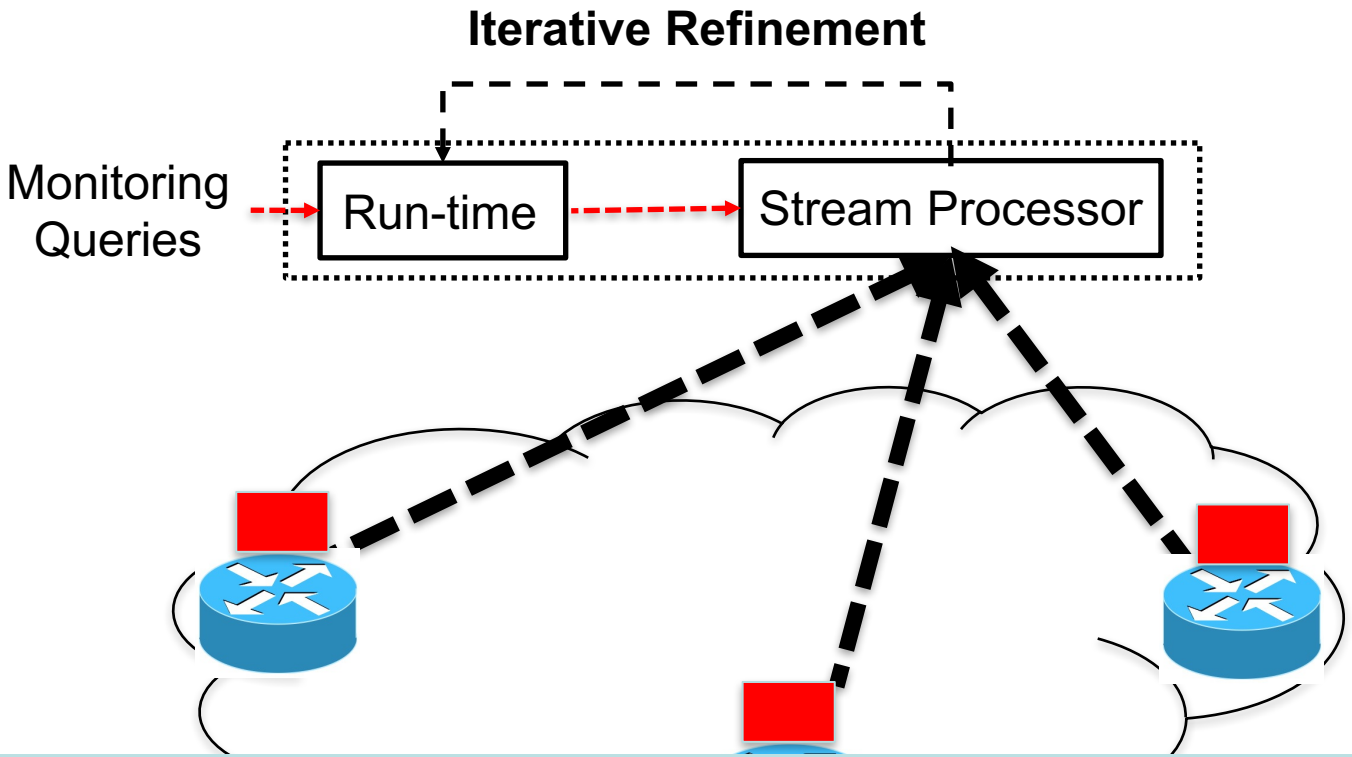
- *Observation*:

  Small fraction of traffic satisfies monitoring queries, e.g. only 1 % of the traffic satisfies reflection attack query

- *How it works*:
  - Augment operator's query to observe at coarser level
  - Iteratively zoom-in to filter out uninteresting traffic

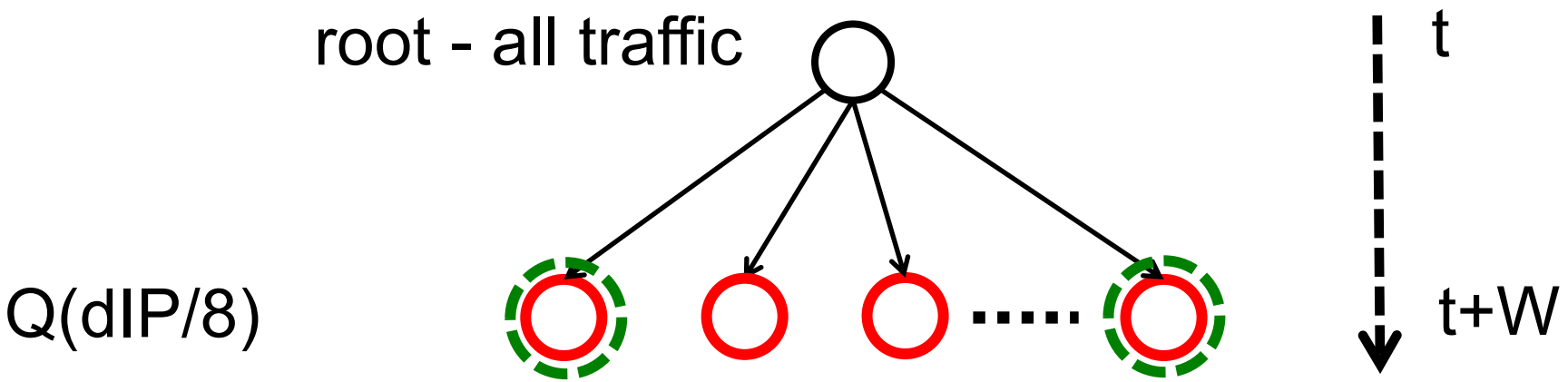
- *Trade-off*:
  - Reduces count bucket cost
  - Introduces additional detection delay cost

Iterative Query Refinement

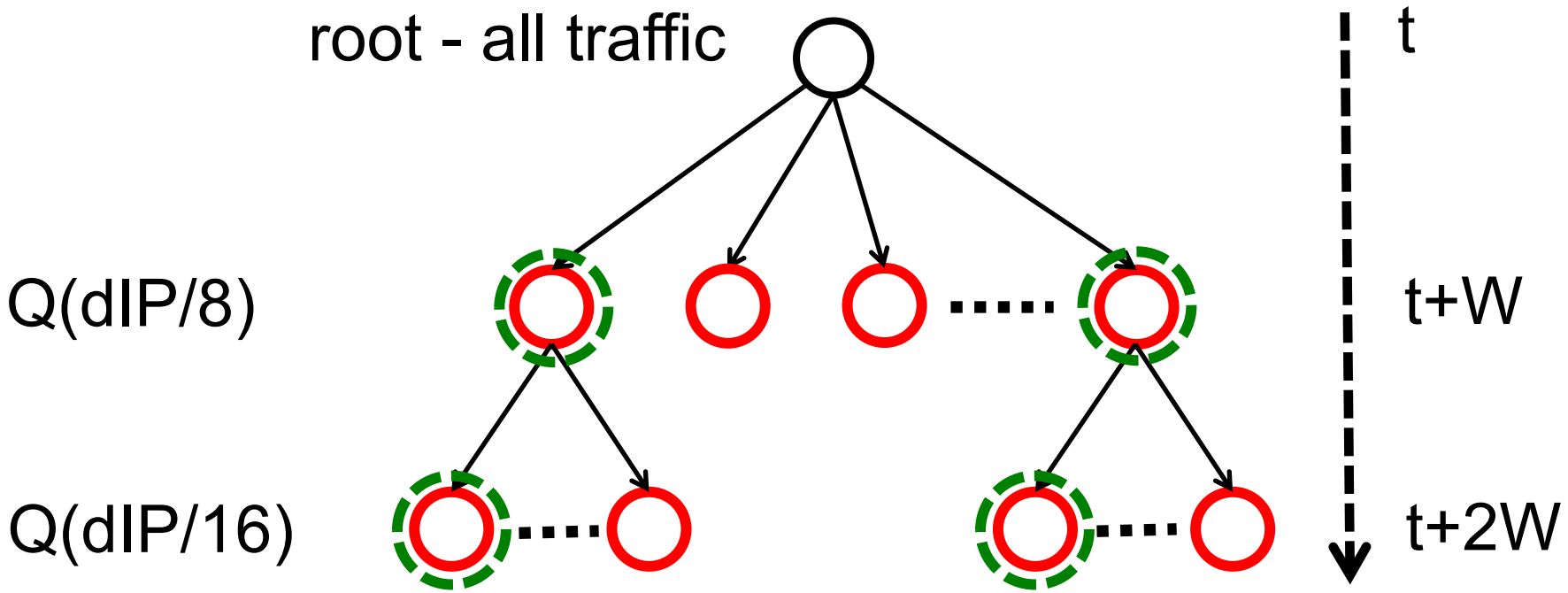


Stream Processor's output used by Run-time to refine queries

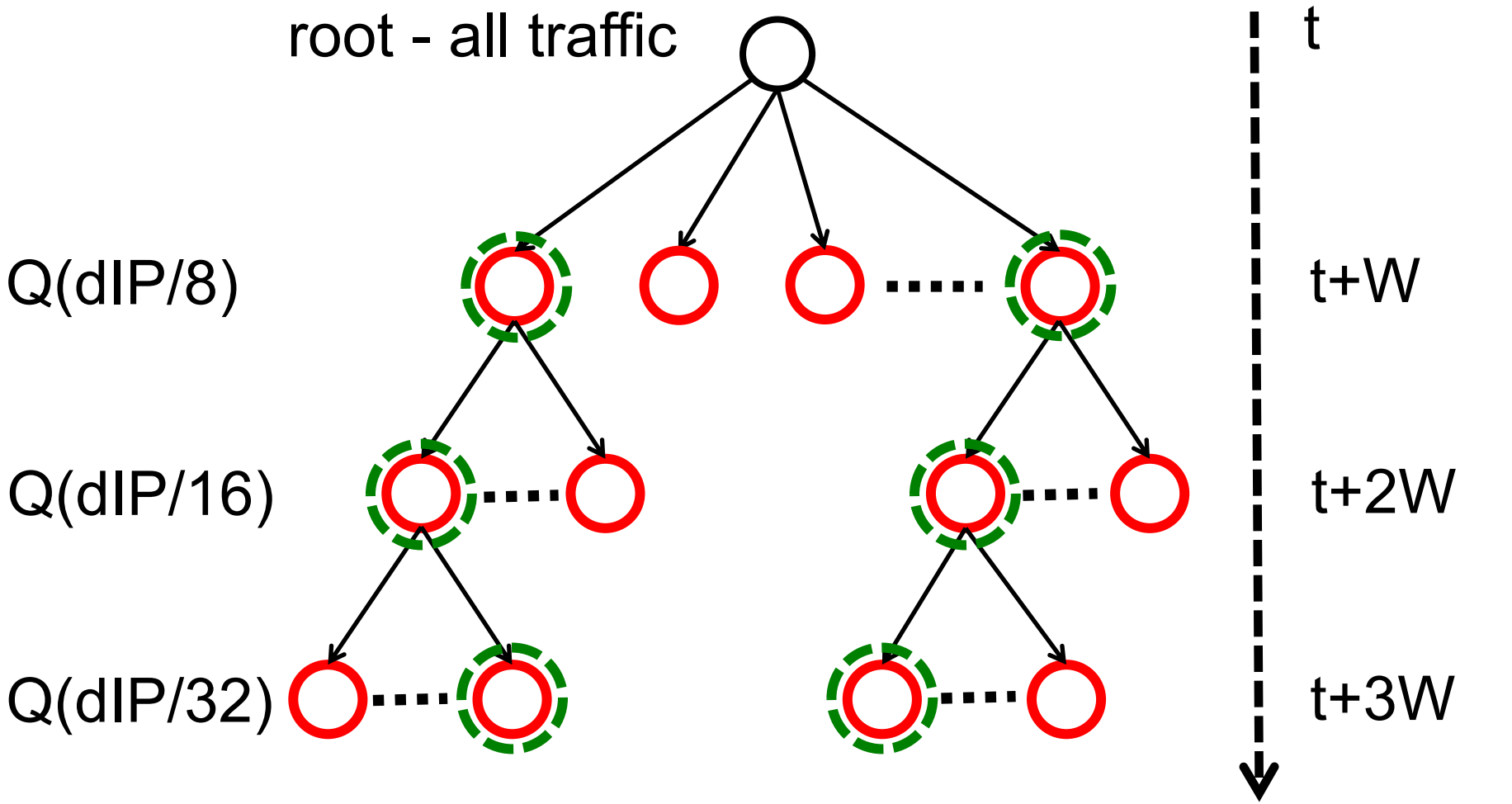
Iterative Refinement in Action



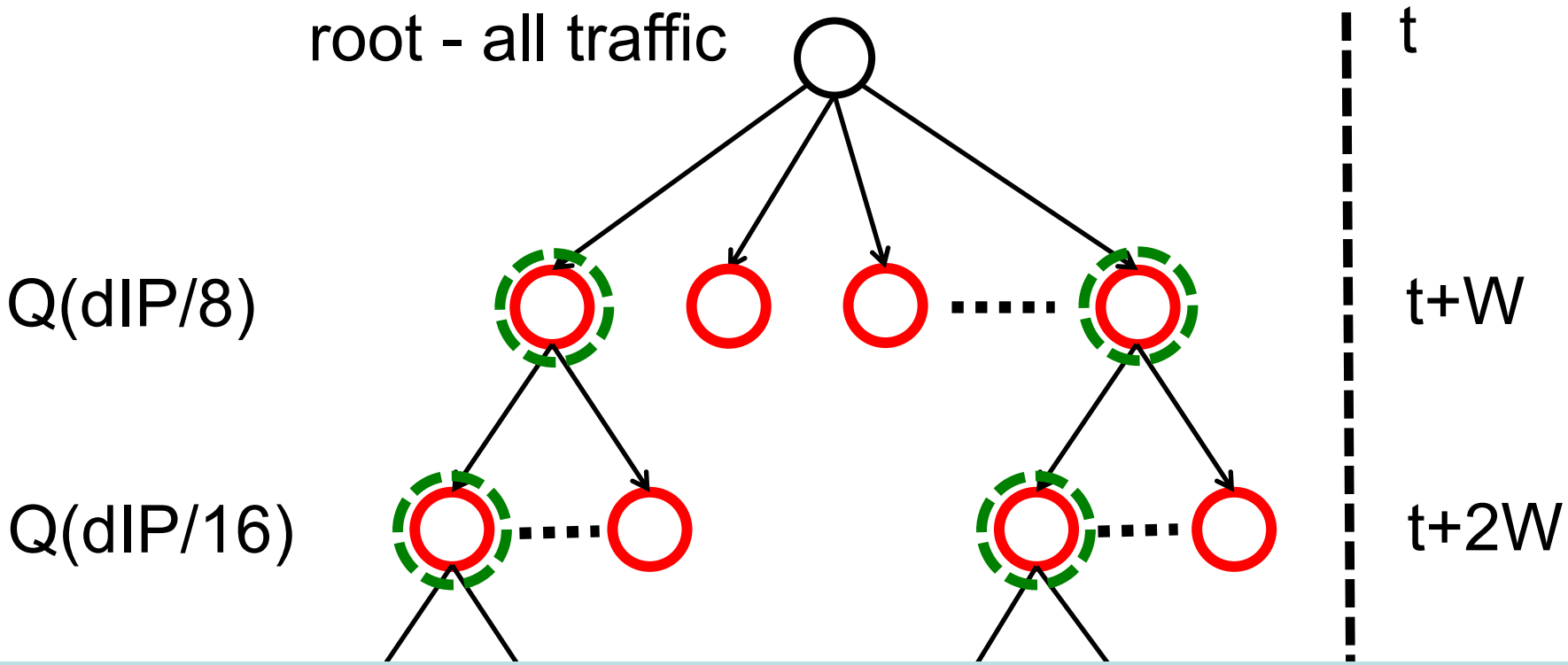
Iterative Refinement in Action



Iterative Refinement in Action



Iterative Refinement in Action



Detects hosts that satisfy the query in 3 window intervals

Idea 2: Query Partitioning

- *Observation:*

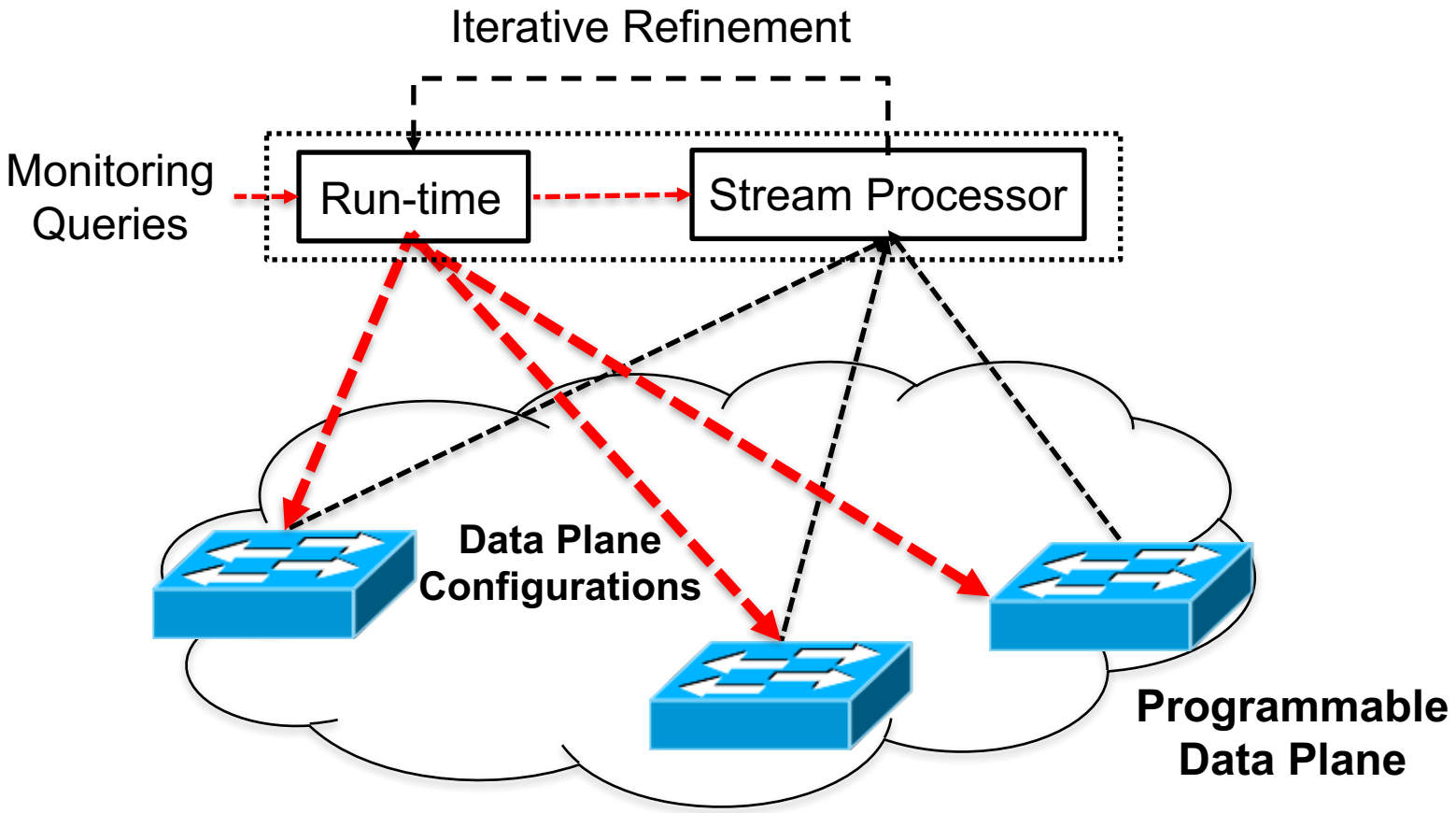
  Data Plane can process packets at line rate

- *How it works:*

  Delegate query processing operations that can be executed in the data plane, e.g. *filtering*, *sampling*

- *Trade-off*:
  - Reduces both the pkt processing & count bucket cost
  - Introduces additional state in the data plane

Query Partitioning



Runtime Partitions Monitoring Queries

Performance Improvements

Reflection Attack Monitoring (dIP/16 → dIP/32)		
	Rate (pps)	**# Buckets**
Stream Processor Only	220 M	1.16 B
Iterative Refinement Only	220 M	12 K
Iterative Refinement + Query Partitioning	**5.4K**	**12 K**

Trades pkt processing & count bucket cost for additional detection delay

Network Monitoring Applications

- Reflection Attack Monitoring (Security)

  Detect hosts for which # of unique sIPs sending UDP response messages exceeds thresh

- Distributed Port Scan Detection (Security)
  - Detect hosts for which # of unique dIP exceeds thresh
  - Detect hosts for which # of unique dPorts exceeds thresh

- Distributed Jitter Monitoring (QoE)

  Detect user groups for which RTT exceeds thresh

Future Directions

- **Query Language**

  How to dynamically map high-level abstractions to packet tuples?

- **Iterative Refinement**

  How to automate generation of optimal refinement plans for a query?

- **Query Partitioning**

  How to execute more complex streaming operations like map, reduce, join etc. in the data plane?

Summary

- Big (“Internet”) Data motivates modulating **network monitoring** as a **streaming analytics** problem
  - Using state-of-art stream processors is not enough
- Stream processors + programmable data planes raise new opportunities
- Iterative Query Refinement and Partitioning can reduce pkt processing and count buckets by **4 and 5 orders** of magnitude, respectively

Backup Slides

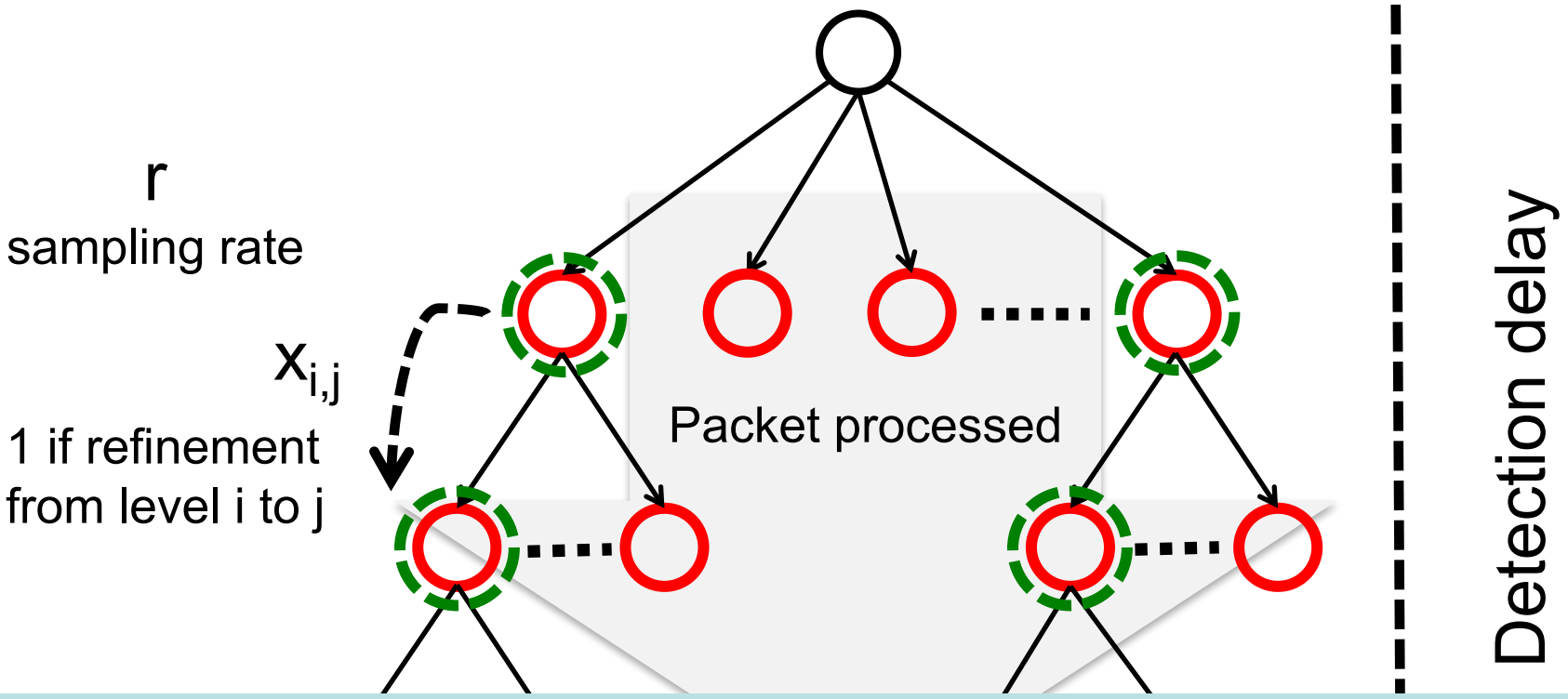
Feast or Famine Dichotomy

- Feast:
  - Capture all traffic, e.g. pcap
  - Detect all interesting network events
  - Higher cost and slower detection

- Famine:
  - Capture subset of traffic, e.g. Netflow, SNMP etc.
  - Not useful for many monitoring applications

Current Trend:
"Capture all the packets, all the time"

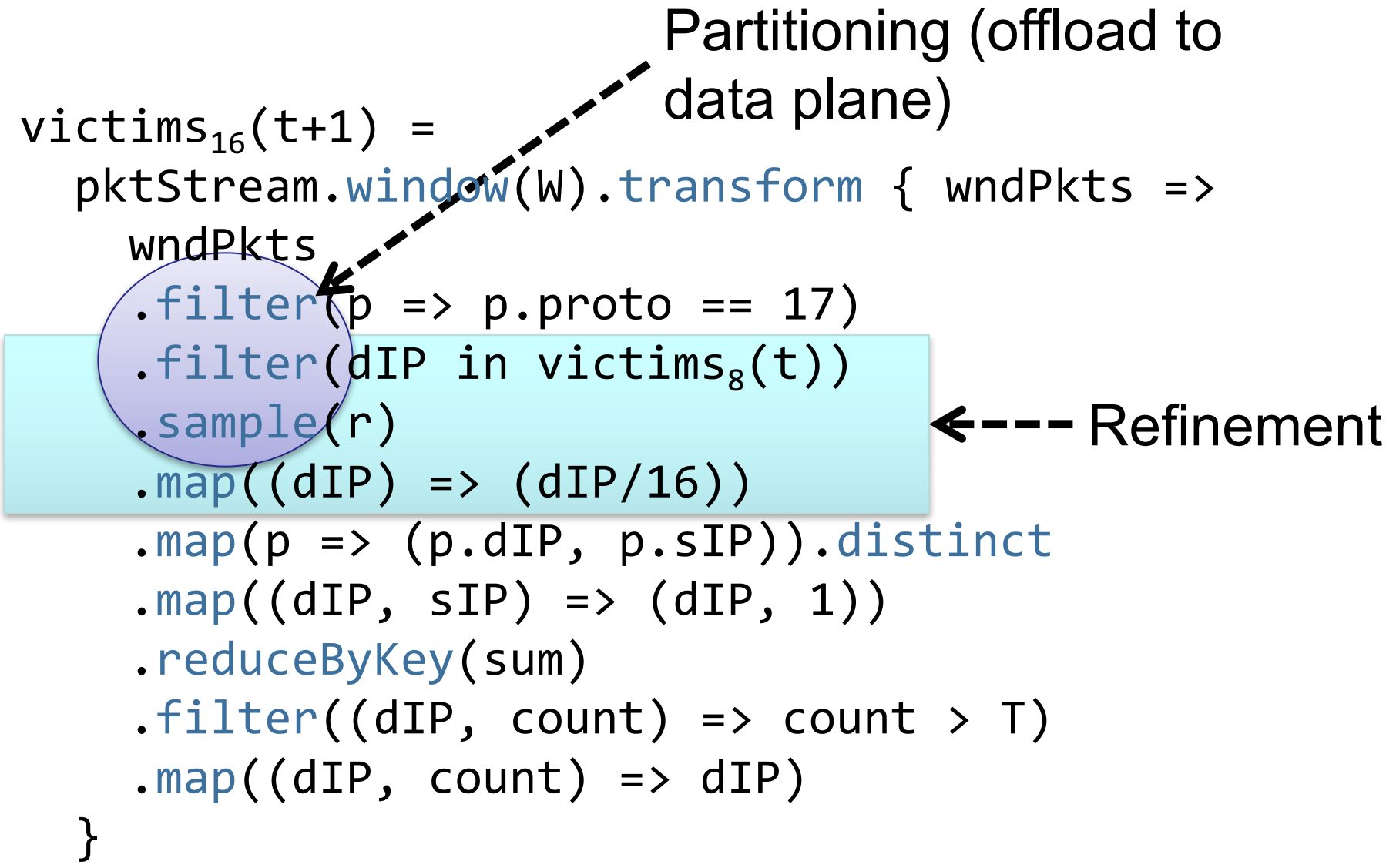
Refinement Plan Search



Learn {x} and r that minimize a linear combination of cost metrics

Count buckets

Run-time


Partitioning (offload to data plane)
victims16(t+1) =
pktStream.window(W).transform { wndPkts =>
wndPkts
.filter(p => p.proto == 17)
.filter(dIP in victims8(t))
.sample(r)
.map((dIP) => (dIP/16))
.map(p => (p.dIP, p.sIP)).distinct
.map((dIP, sIP) => (dIP, 1))
.reduceByKey(sum)
.filter((dIP, count) => count > T)
.map((dIP, count) => dIP)
}
Refinement

SONATA Architecture