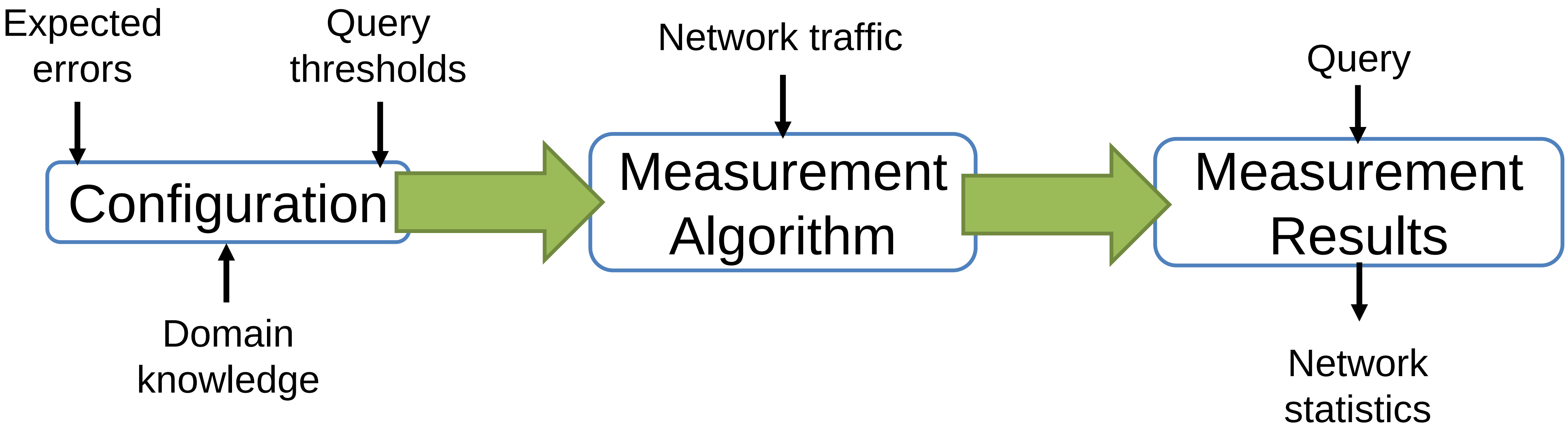
SketchLearn: Relieving User Burdens in Approximate Measurement with Automated Statistical Inference

<u>Qun Huang</u>, Patrick P. C. Lee, Yungang Bao

中科院计算所
INSTITUTE OF COMPUTING TECHNOLOGY, CAS

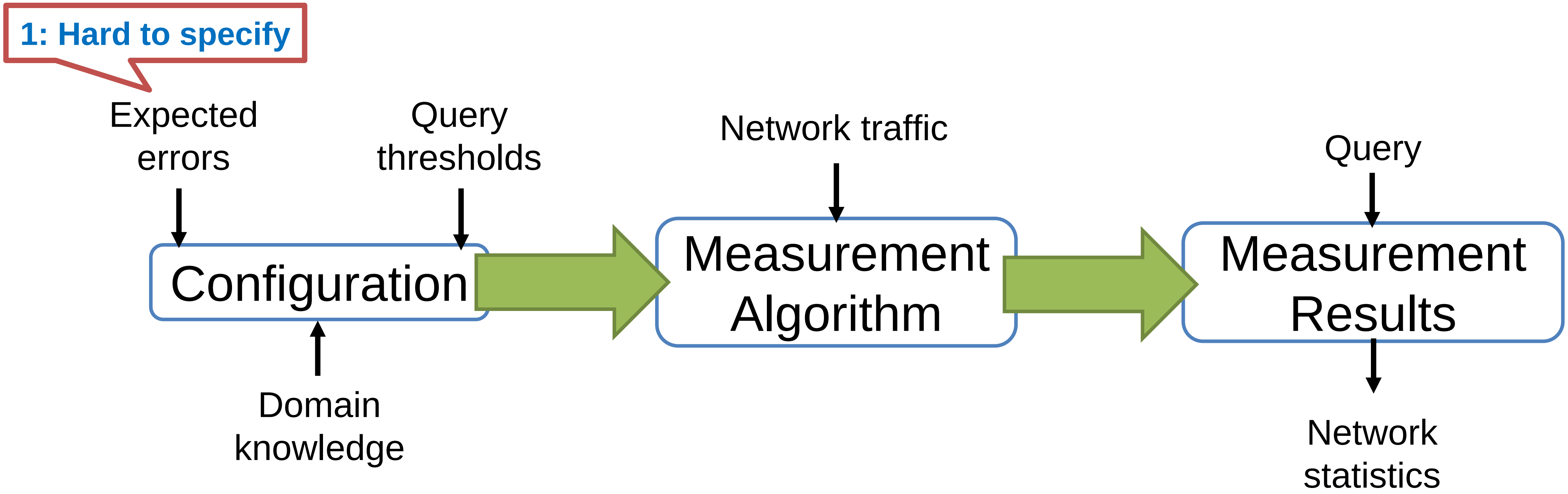
香港中文大學
The Chinese University of Hong Kong

Typical Approximate Measurement



Our goal: identify and relieve user burdens for approximate measurement

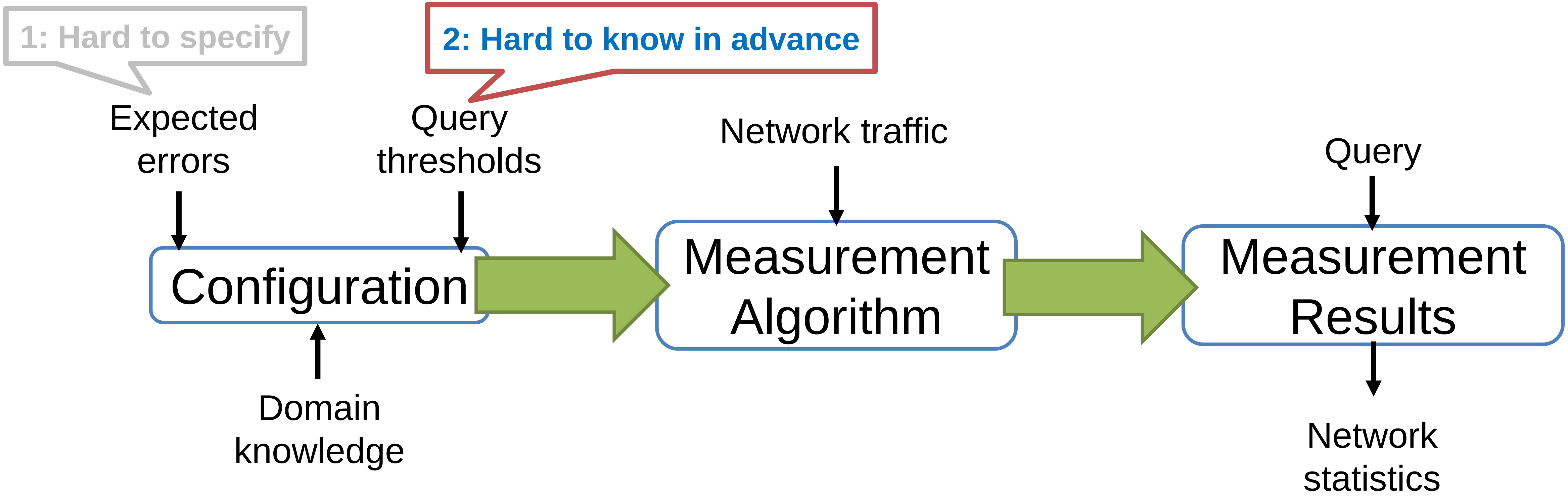
User Burden 1



Errors need to be specified:

- bias: an answer deviates true answer by ε
- failure probability: fail to produce small-error answer with a probability δ

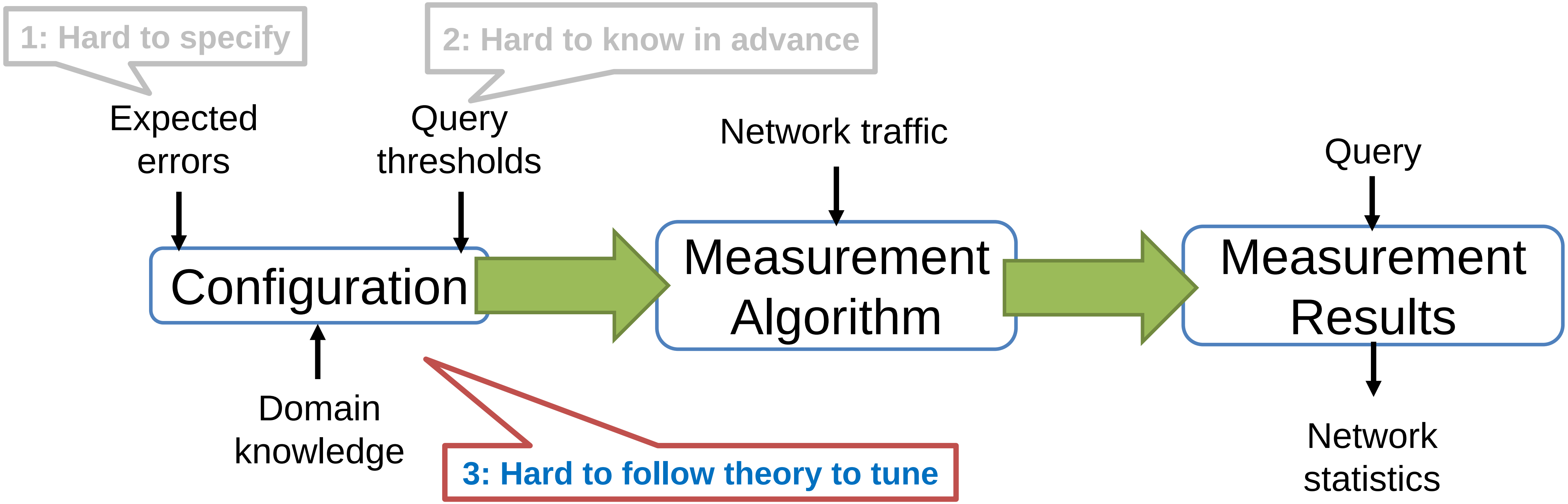
User Burden 2



Large-threshold configurations fail to work for small-threshold queries
No guideline for sufficiently small threshold

- Vary across management operations and traffic characteristics

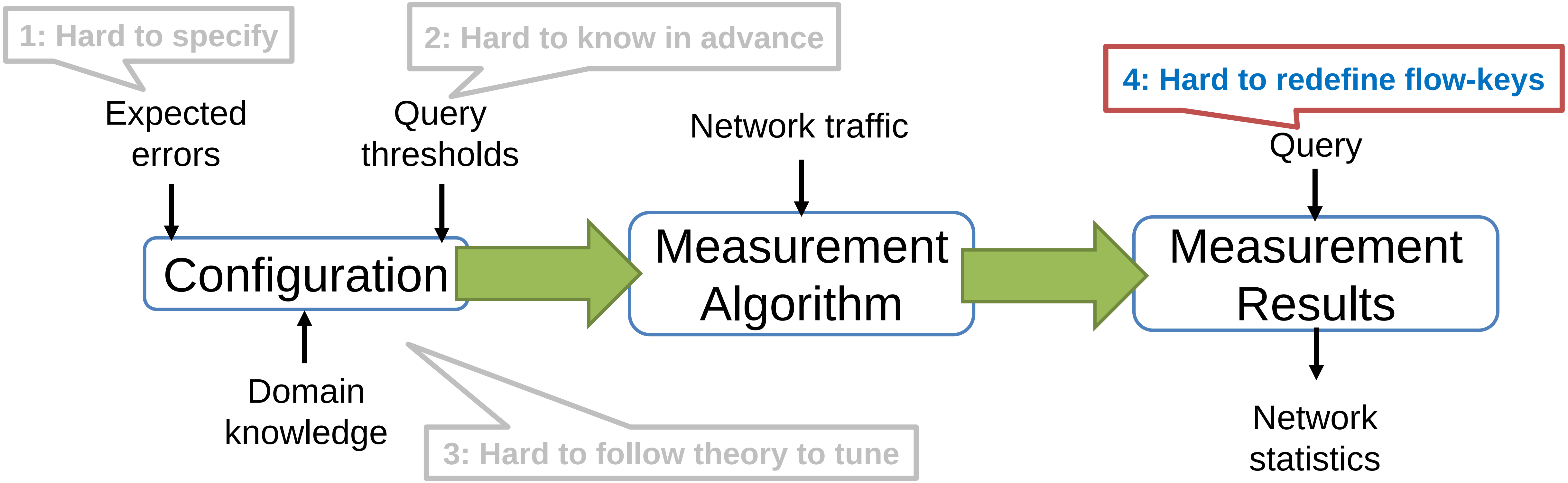
User Burden 3



Domain knowledge is not always available

- Theories usually show worst-case results
- Configuration for worst case not practically efficient

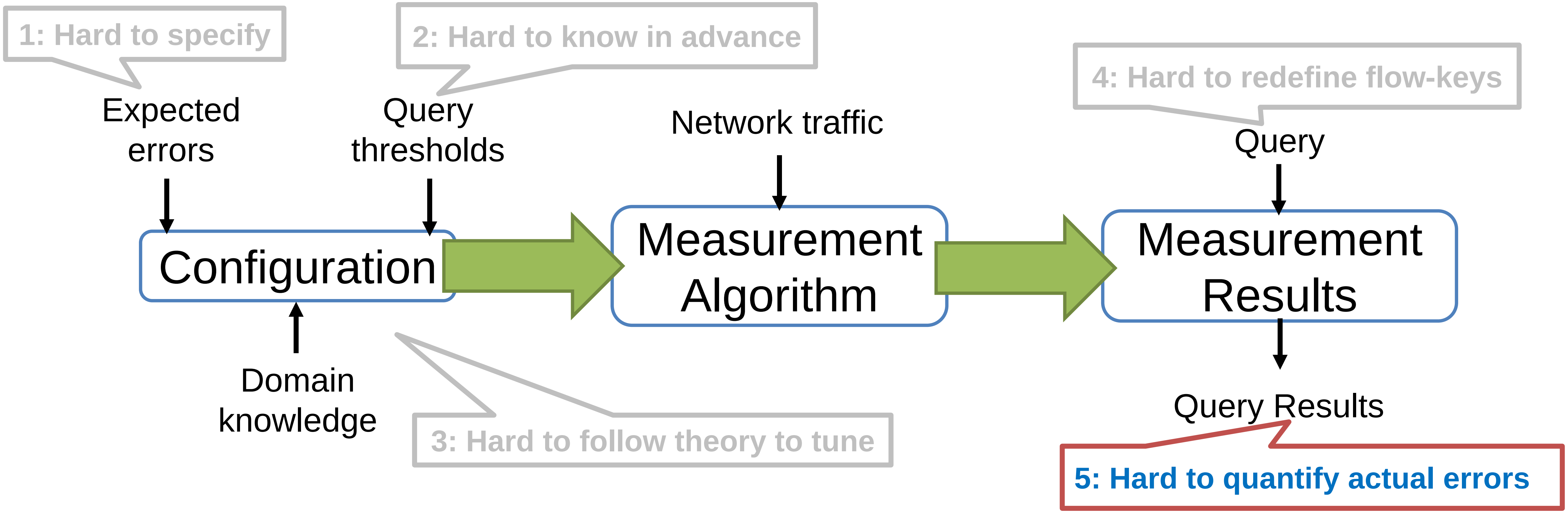
User Burden 4



Algorithm works on a particular flow-key

- Flow-key definition must be fixed in deployment

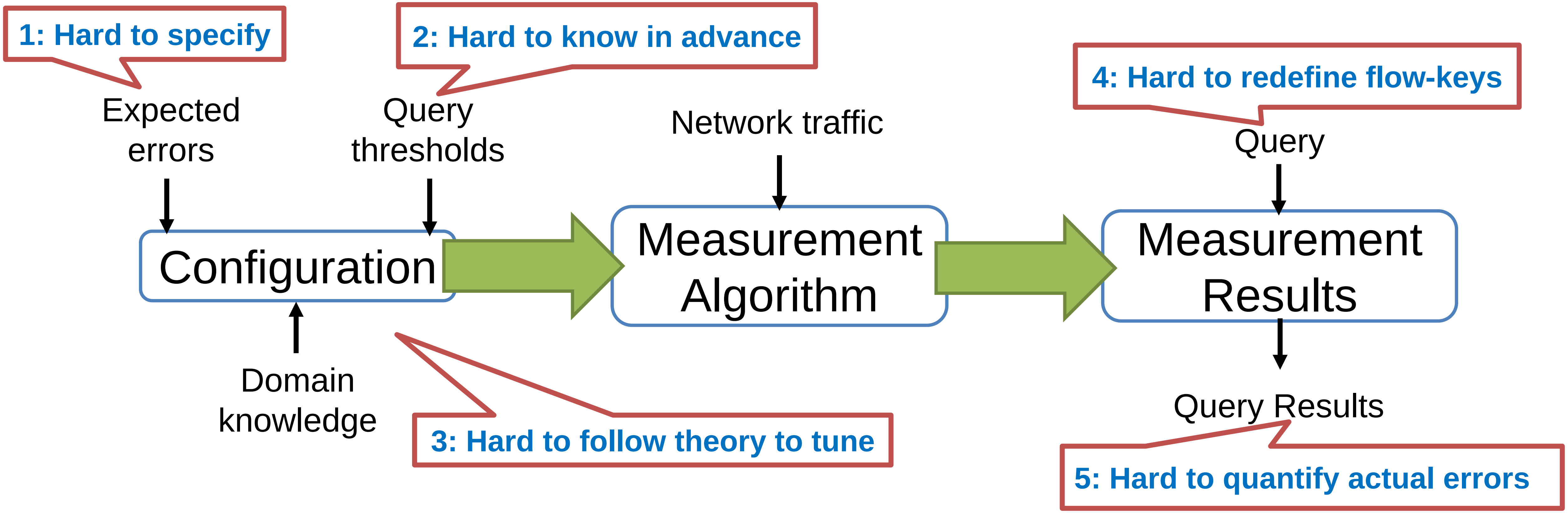
User Burden 5



Infeasible to track errors for a particular flow

- Configuration only tells worst-case and overall errors

All User Burdens



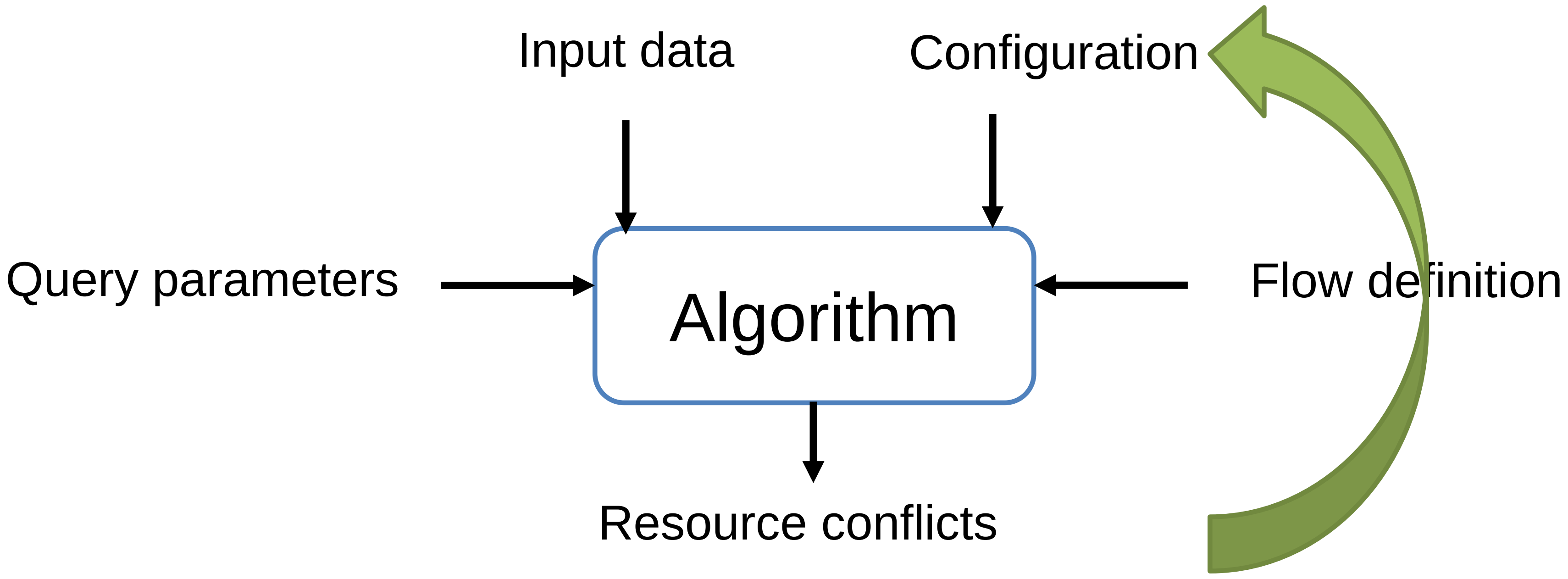
Our Work

SketchLearn: Sketch-based measurement system with limited user burdens

- Relieve five user burdens
- Performance
  - Catch up with underlying packet forwarding speed
- Memory efficiency
  - Consume only limited memory
- Accuracy
  - Preserve high accuracy of sketches
- Generality
  - One design and one configuration for multiple tasks
  - Deployable in both software and hardware

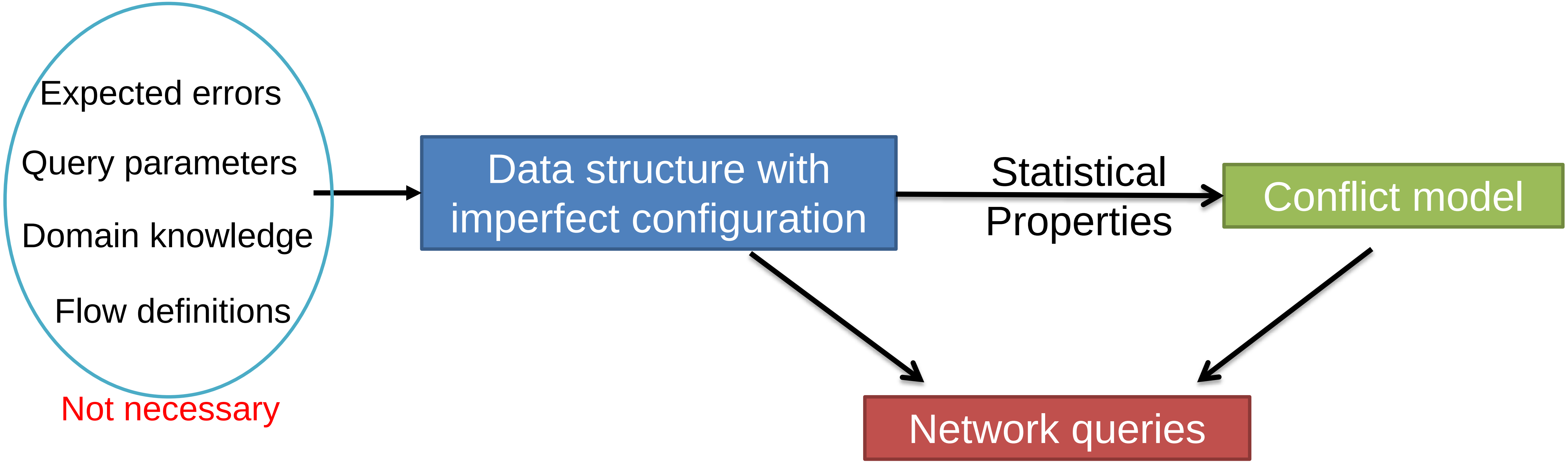
Root Cause: Resource Conflicts

- Previous work: perfect configuration to eliminate conflicts
  - Determined by many factors

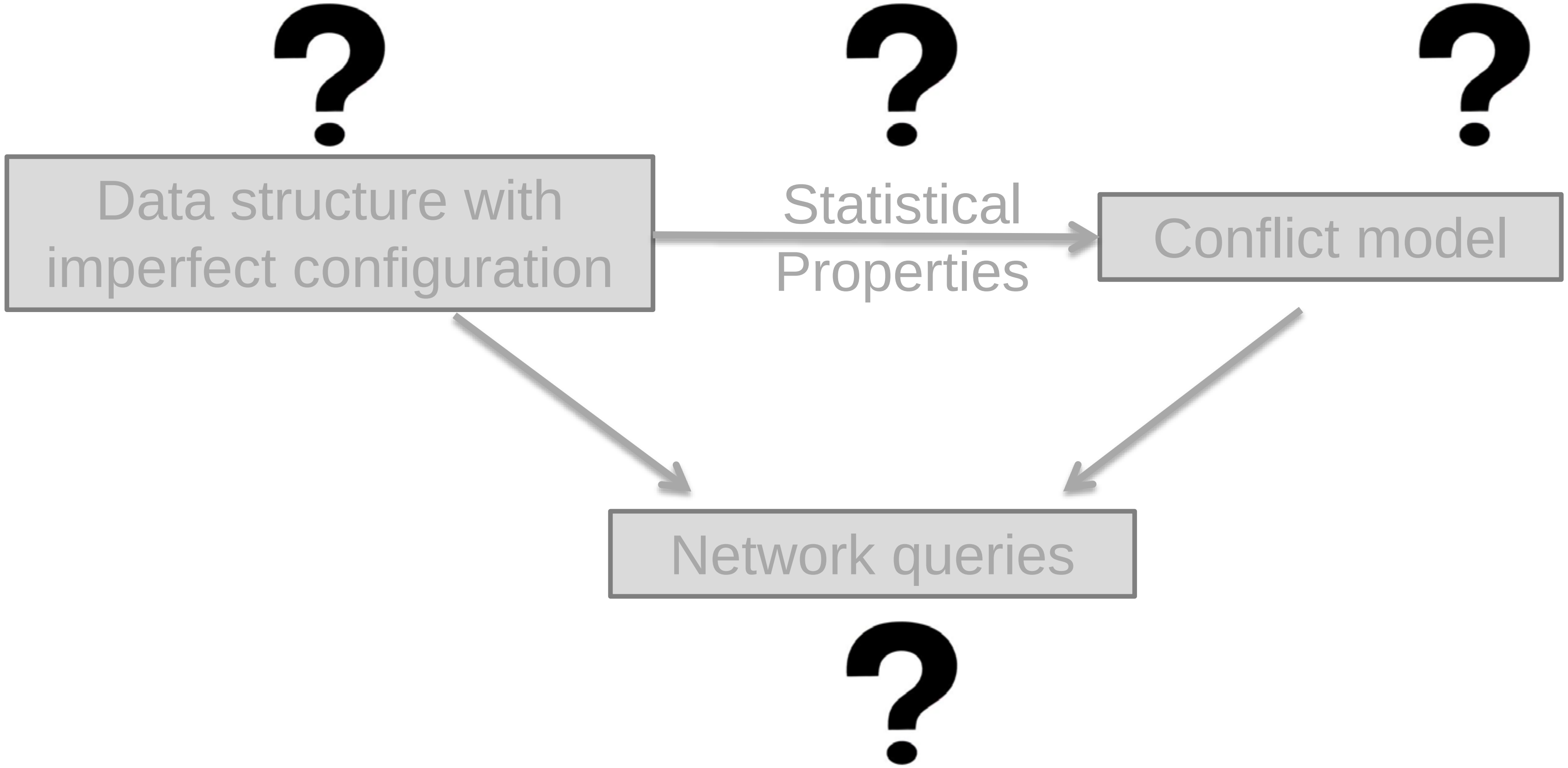


High-Level Idea

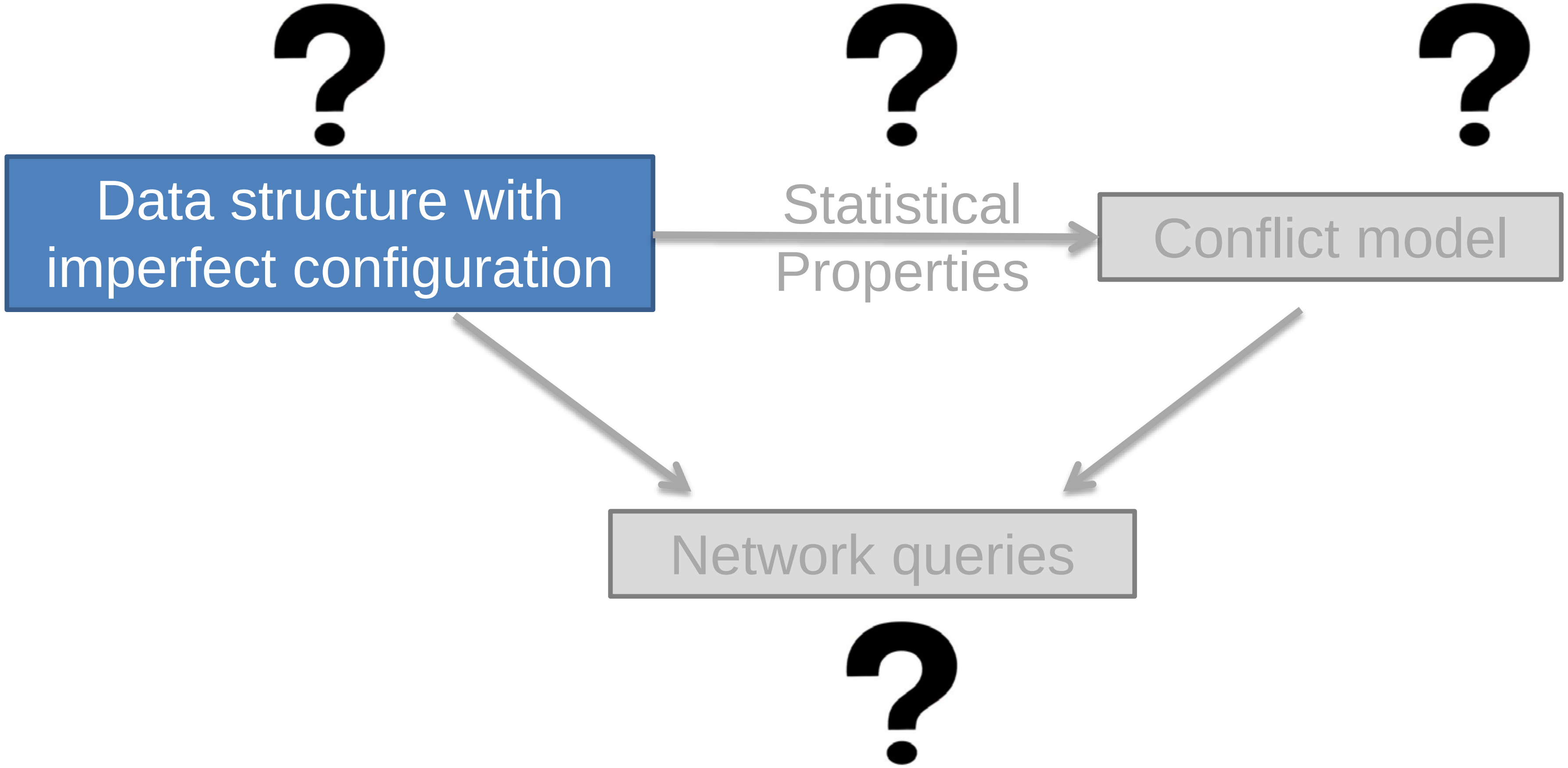
- ➢ Not pursue perfect configurations to mitigate resource conflicts
  - Hard to identify right trade-offs
- ➢ Characterize resource conflicts in an “imperfect” configuration



How to Realize?

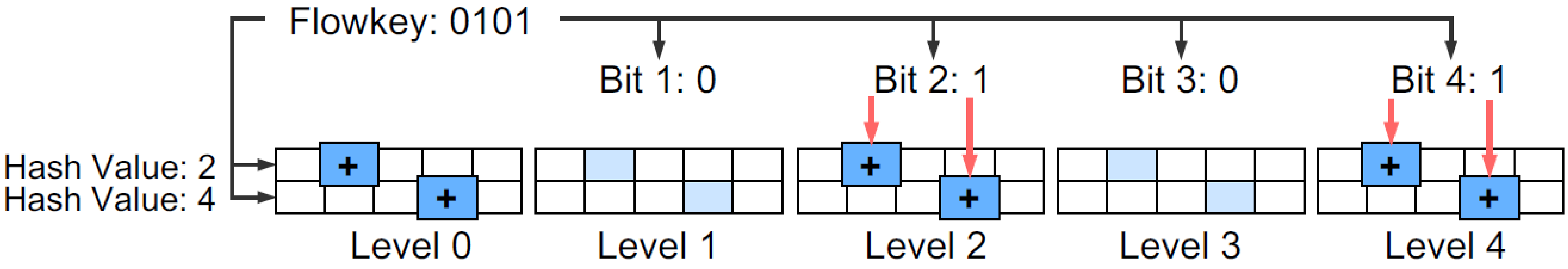


Design Data Structure

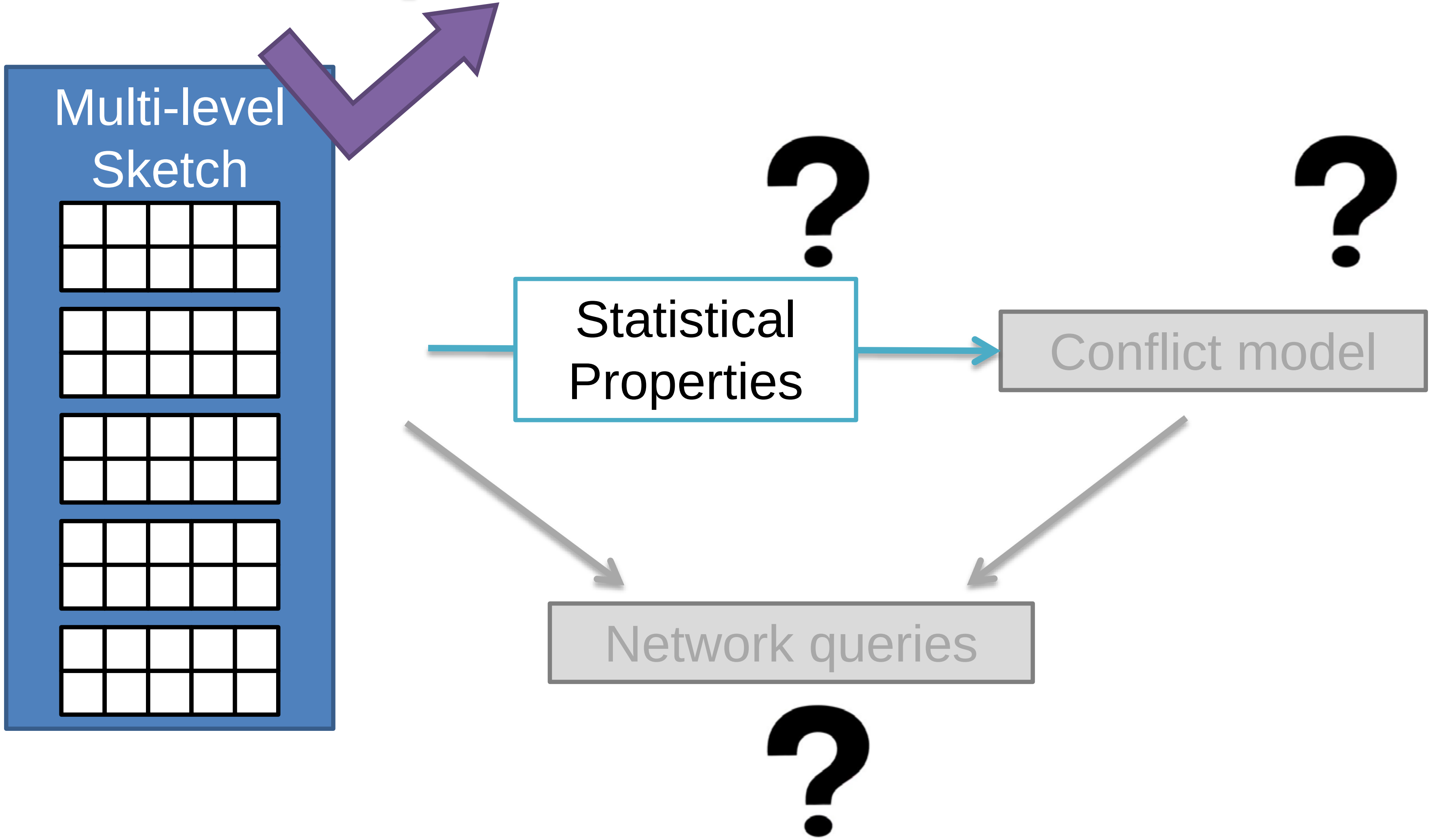


Multi-level Sketch [Cormode, ToN 05]

- L: # of bits considered
- Data structure: L+1 levels (from 0 to L), each of which is a counter matrix
- Level 0 is always updated
- Level k is updated iff k-th bit in a flowkey is 1
- All levels share same hash functions



Statistical Properties of Resource Conflicts

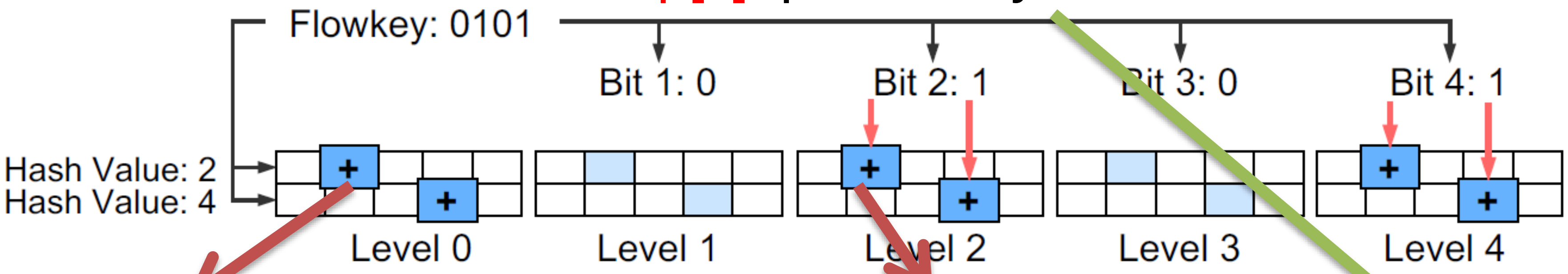


Properties

- Does a flow update level k?
  - Depends on inherent distribution of flow keys
- Does a flow update bucket (i, j)?
  - Depends on hash functions of sketch
- A theory should characterize the above two factors

Main Theorem

p[k]: probability that k-th bit is 1



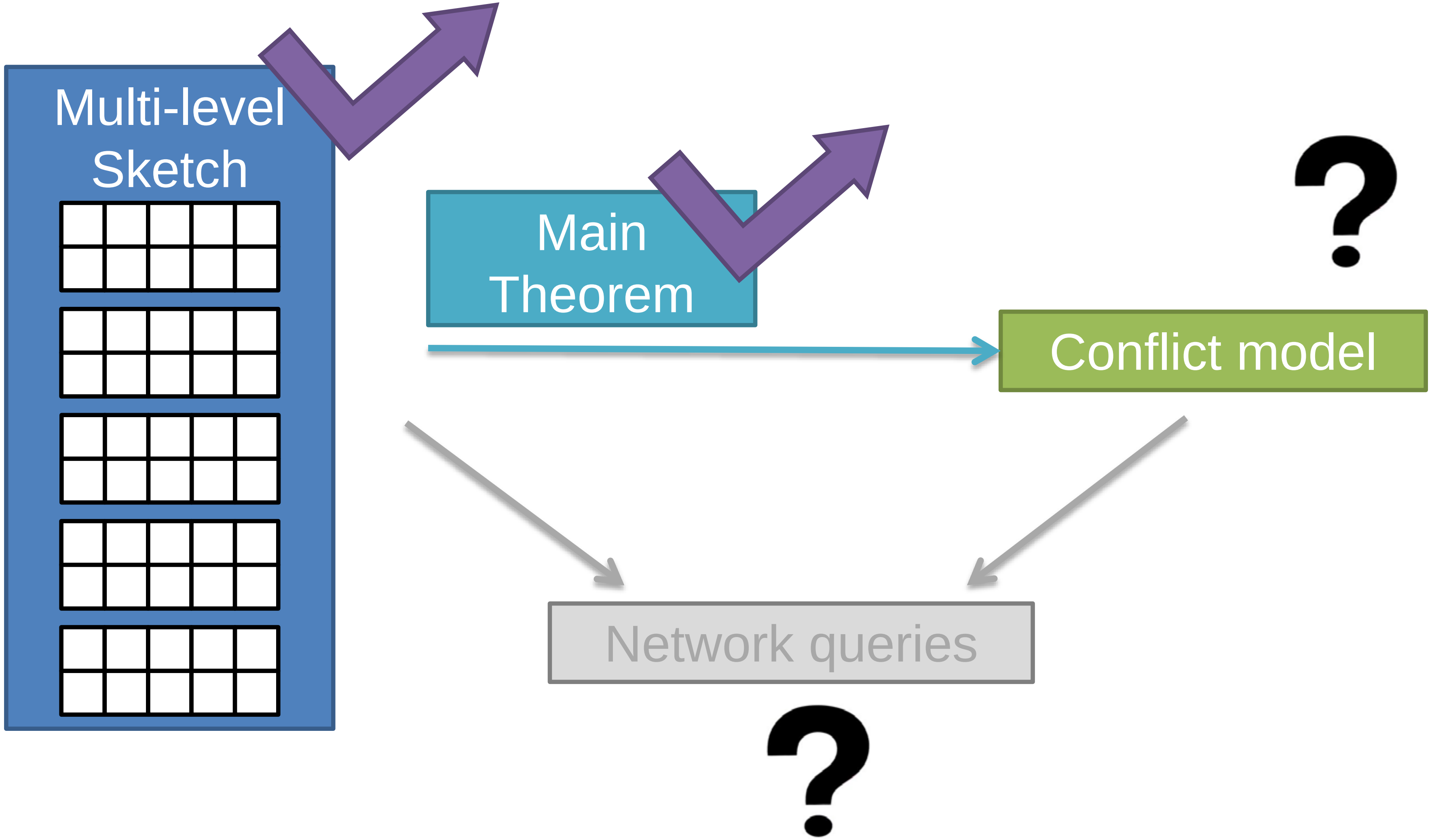
$V_{i,j}[0]$: counter of bucket (i,j)

$V_{i,j}[k]$: counter of bucket (i,j)

random variable $R_{i,j}[k] = \frac{V_{i,j}[k]}{V_{i,j}[0]}$

Main Theorem: if no large flows, $R_{i,j}[k]$ follows Gaussian distribution with mean p[k]

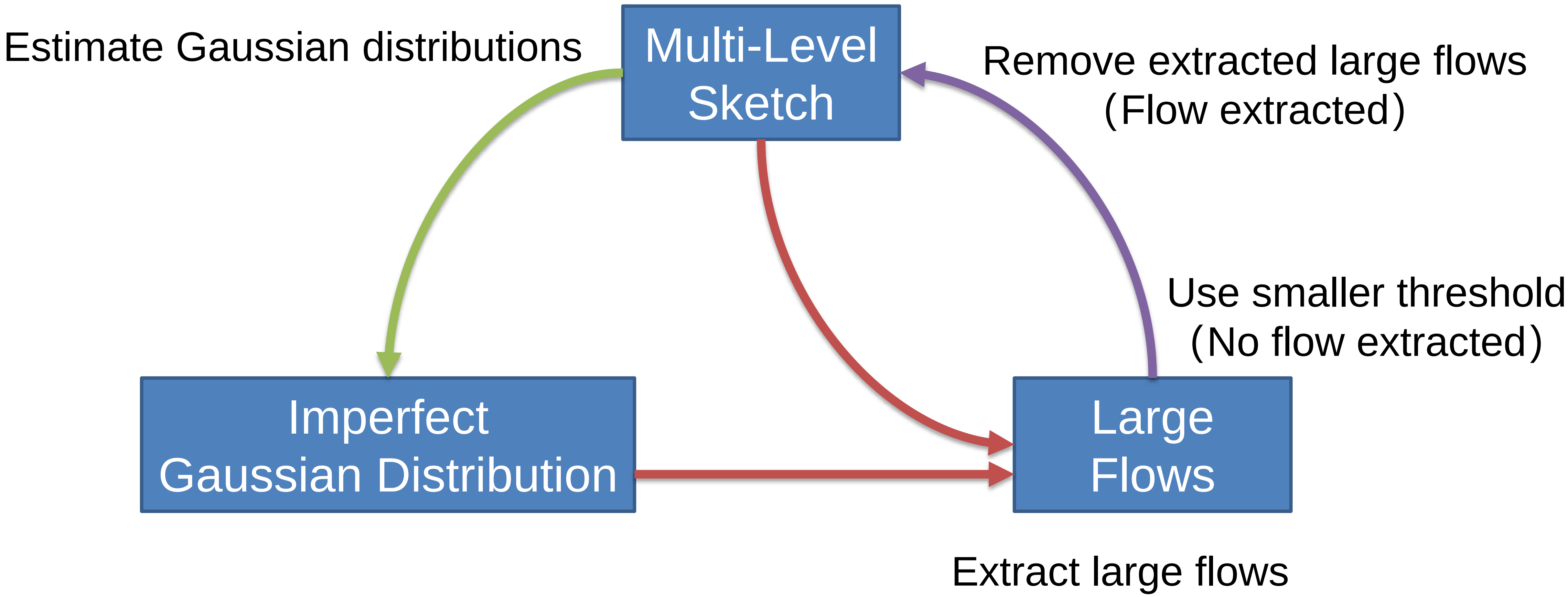
Build Conflict Model



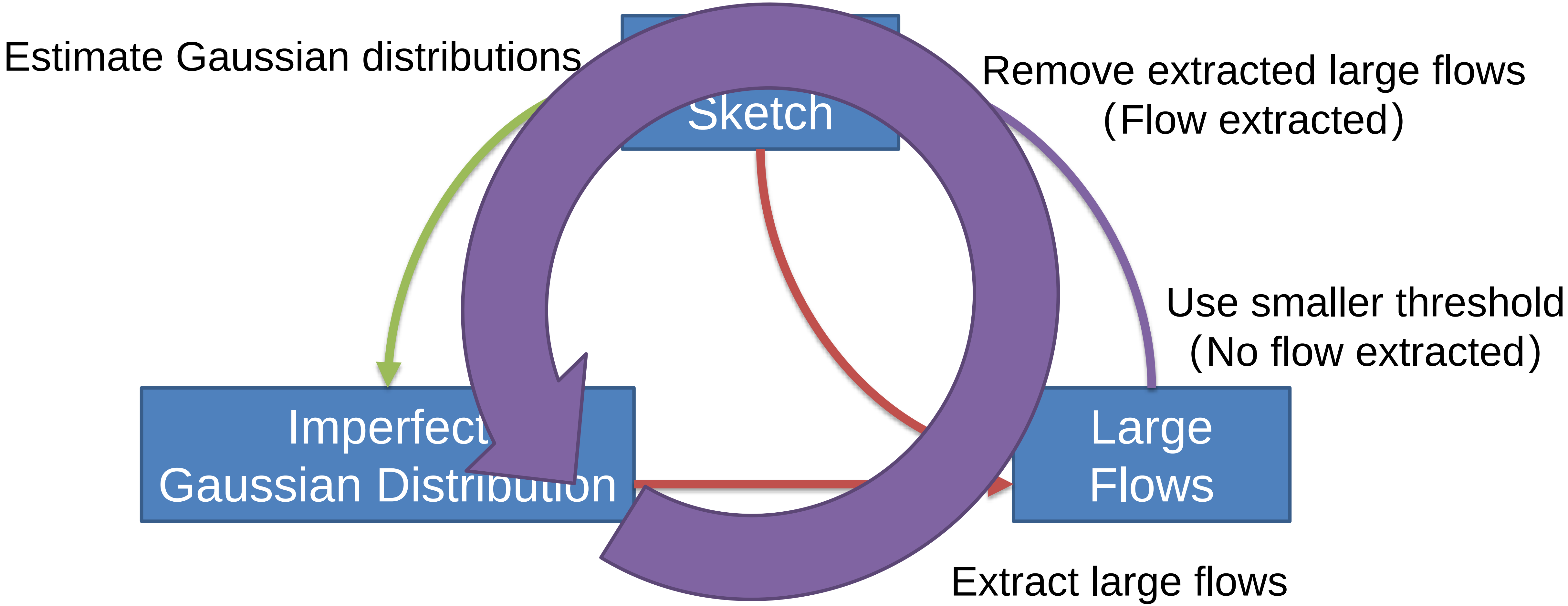
Statistical Model Inference

- Goals
  - Extract all large flows
  - Guarantee remaining flows in sketches are small
  - Estimate Gaussian distributions for each level
- Challenges
  - No guidelines to distinguish large and small flows

Self-Adaptive Inference Algorithm



Self-Adaptive Inference Algorithm



Large Flow Extraction

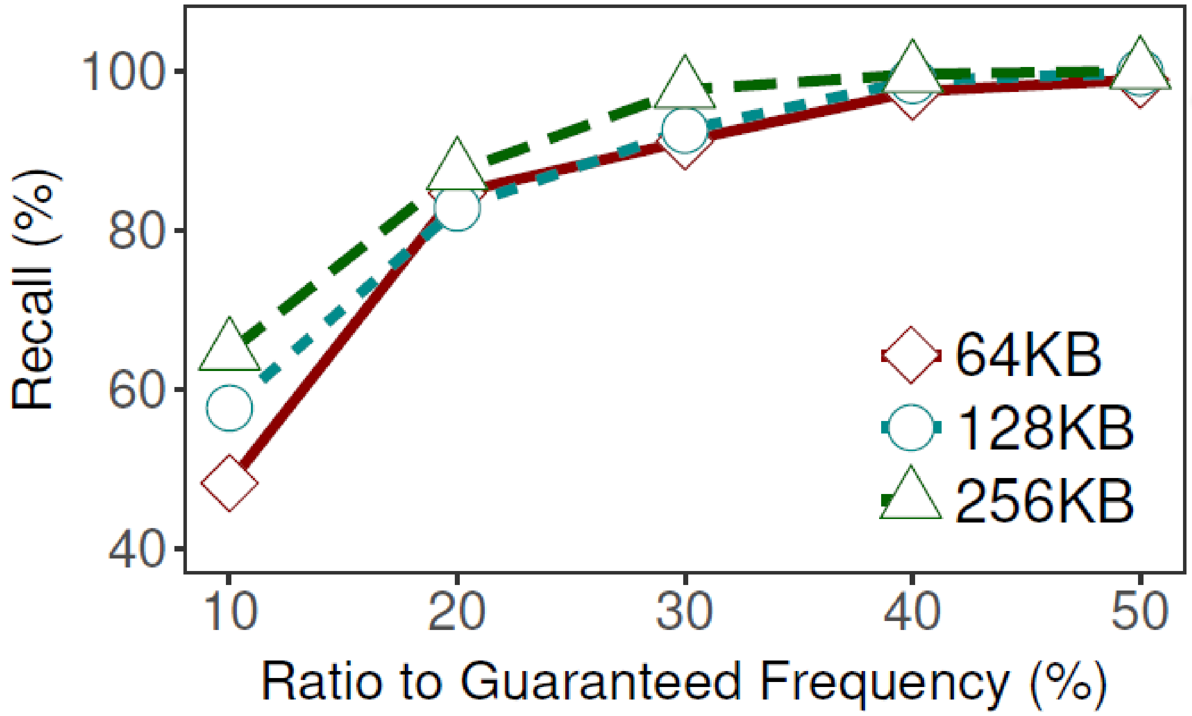
- Intuition: a large flow
  - (i) results in extremely (large or small) $R_{i,j}[k]$, or
  - (ii) at least deviates $R_{i,j}[k]$ from its expectation p[k] significantly
- Key idea: examine $R_{i,j}[k]$ and its difference from p[k], then
  - Determine k-th bit of a large flow (assuming it exists)
  - Estimate frequency
  - Associate flow confidence

More details in paper!

Guarantee

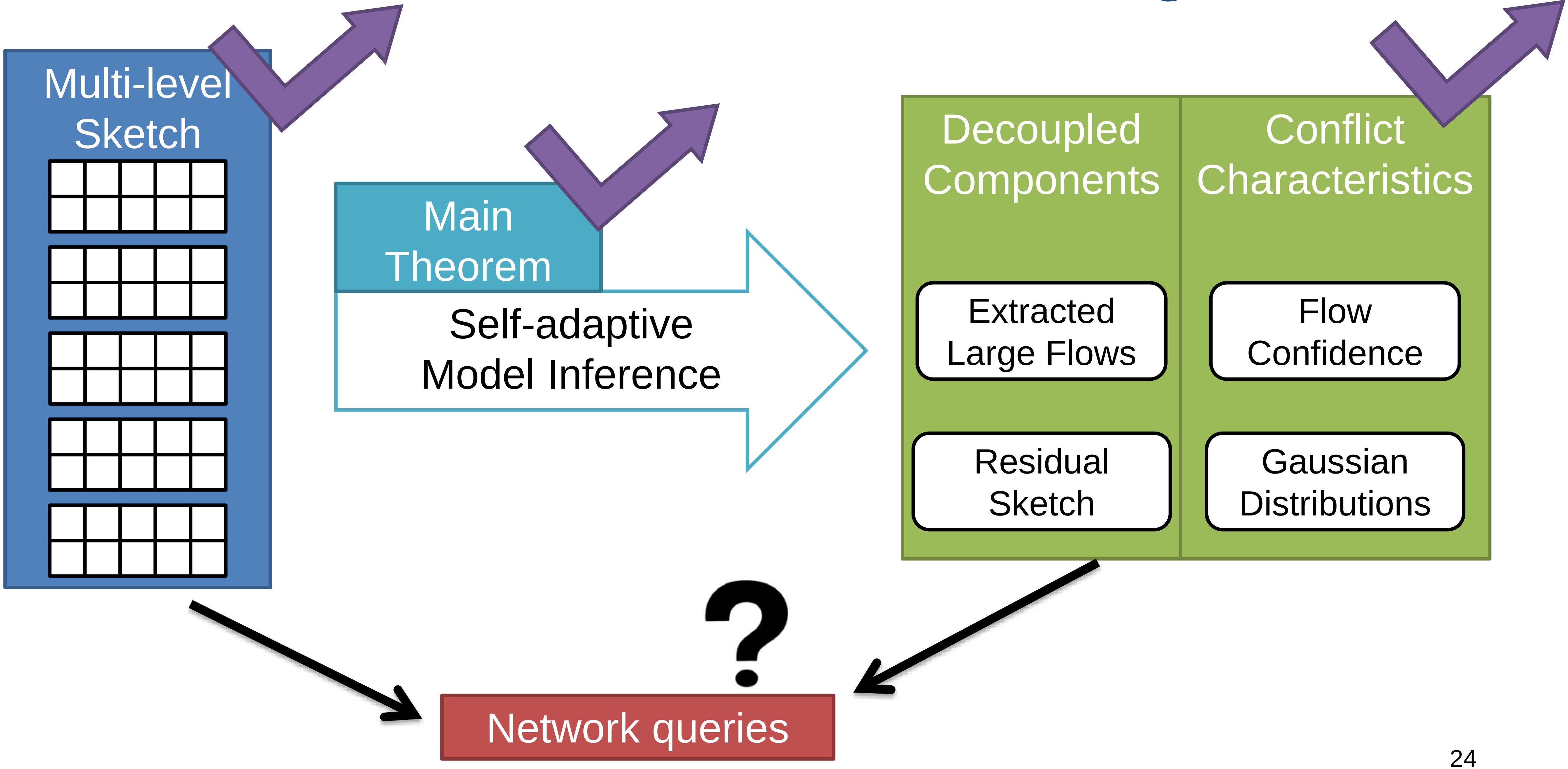
- **Theorem**: w is sketch width, flows exceeding 1/w of total must be extracted
  - Empirical results: even flows that are smaller than 1/w can also be extracted!

64KB: flows above 0.6% of total traffic can be extracted (by theorem)



Practice: >99% flows exceeding 0.3% are also extracted with 64KB

How to Perform Network-wide Queries?



Supported Network Queries

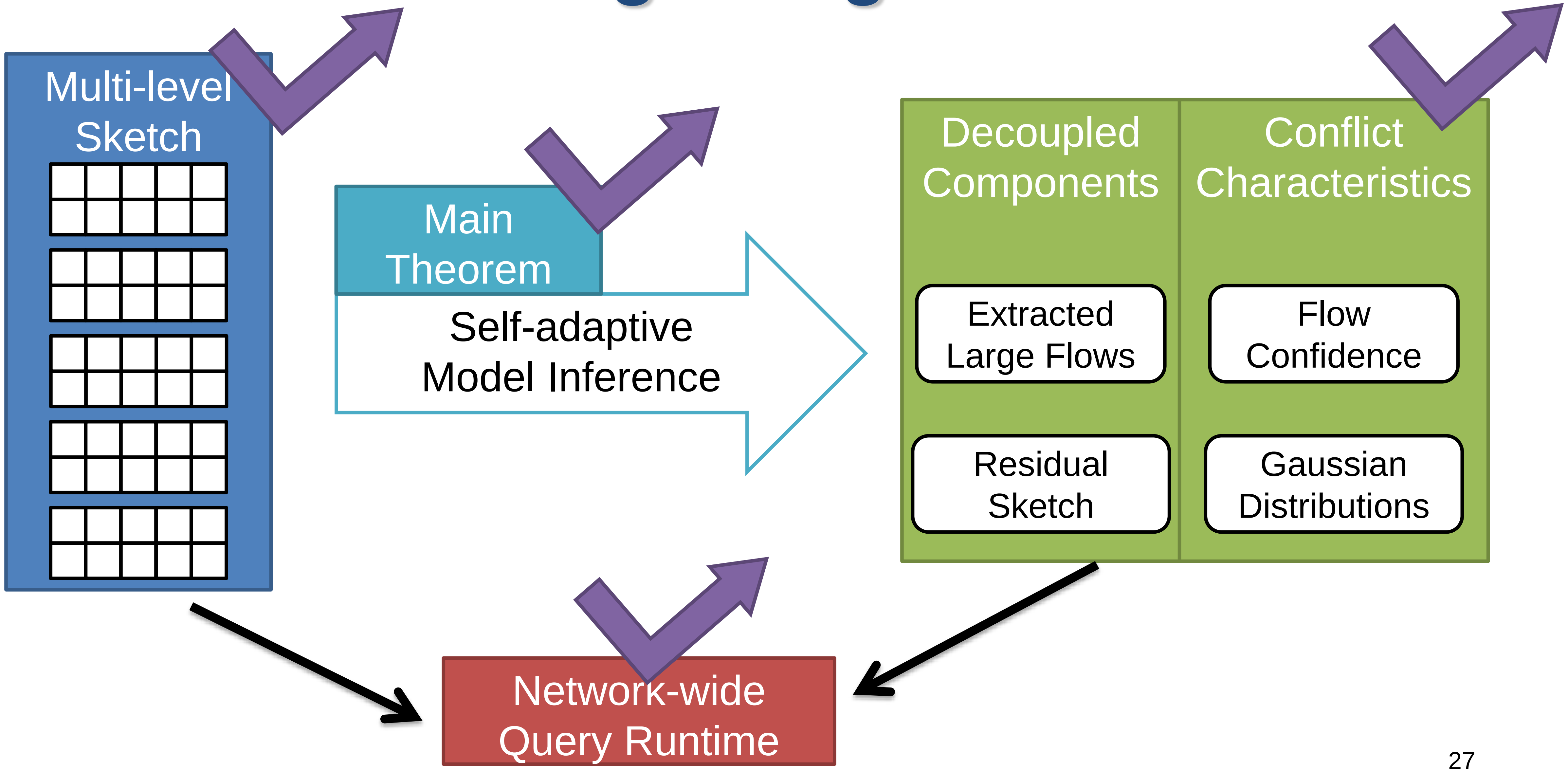
- Per-flow byte count
- Heavy hitter detection
- Heavy changer detection
- Cardinality estimation
- Flow size distribution estimation
- Entropy estimation

Extended Query Model

- Allow query for arbitrary flowkeys
  - Only use corresponding levels

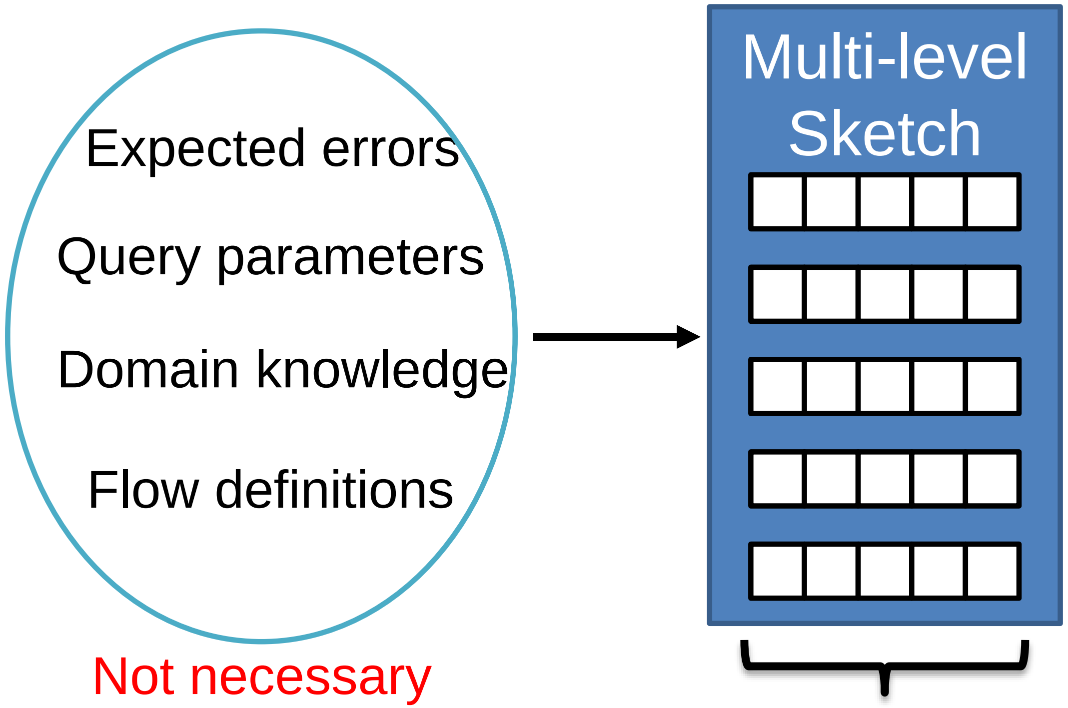
- Estimate actual query errors
  - Use attached errors for each large flow
  - Use Gaussian distributions

Putting It Together



(Slight) User Burdens

- Users now just need to configure the multi-level sketch



One row suffices

Not necessary

Columns: based on minimum flow or memory budget

Example: 400 KB memory, Require flows exceeding 0.1% } 1000 columns

Implementation

- Challenge: updating L+1 levels is time consuming
- Solution: parallel updating
  - L+1 levels are independent
- Software
  - Based on OpenVSwitch + DPDK
  - Parallelism with SIMD
- Hardware
  - Based on P4 programmable switches
  - Parallelism with P4 pipeline stages

Evaluation

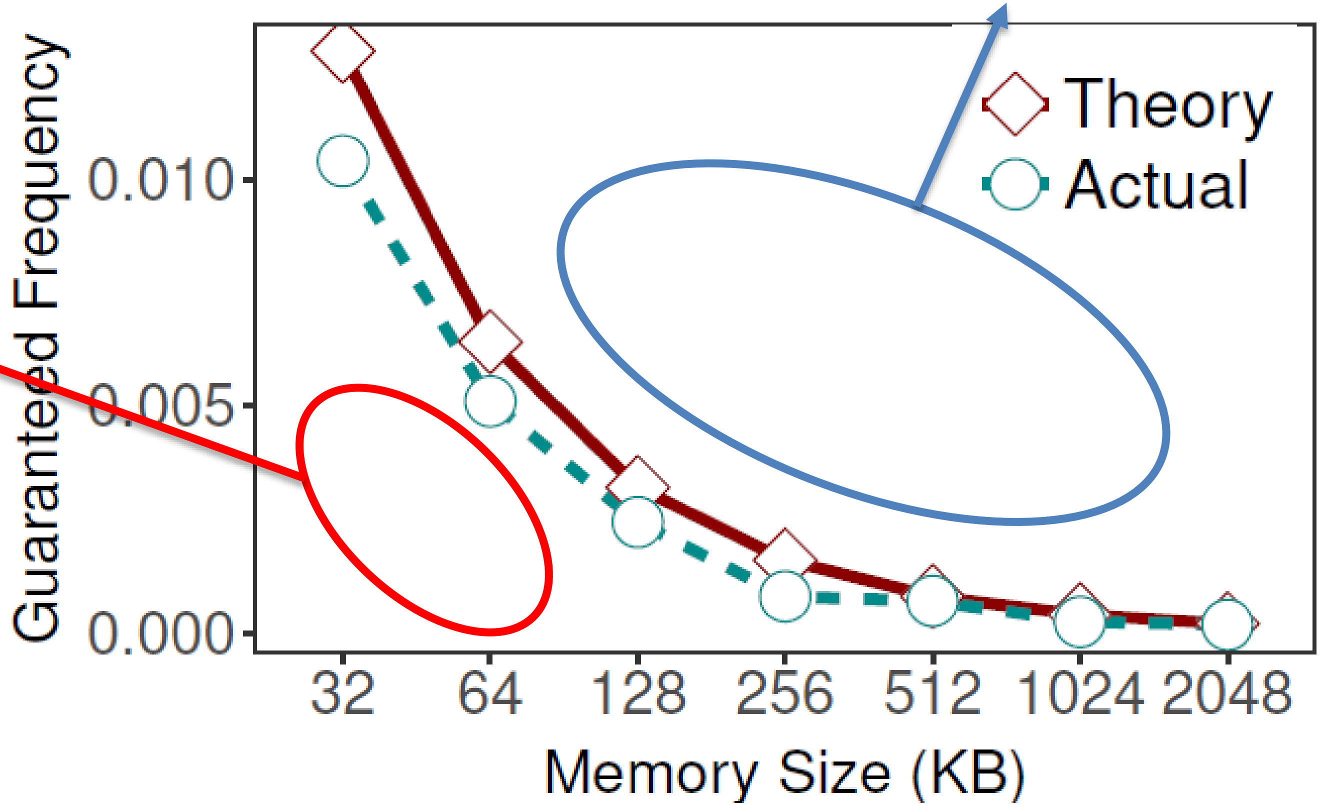
- Platforms
  - Software: OpenVSwitch + DPDK
  - Hardware: Tofino swtich
  - Large-scale simulation
- Traces
  - Caida 2017
  - Data center traffic (IMC 2010)

Fitting Theorem

➢ Guaranteed boundary of flow extraction

Flows that are guaranteed to be extracted

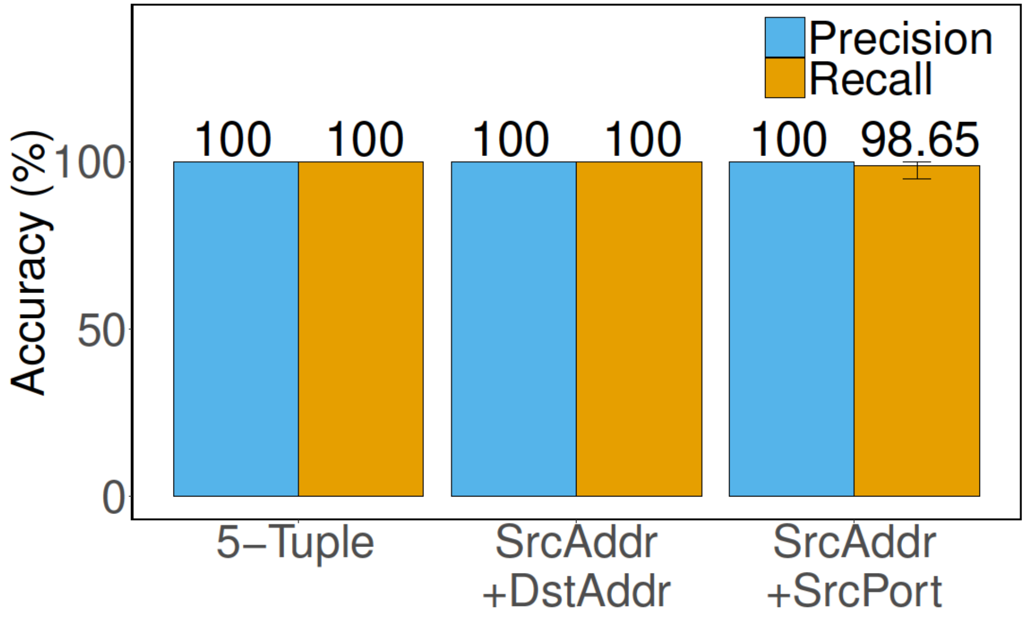
Flows that are not guaranteed (but are likely) to be extracted



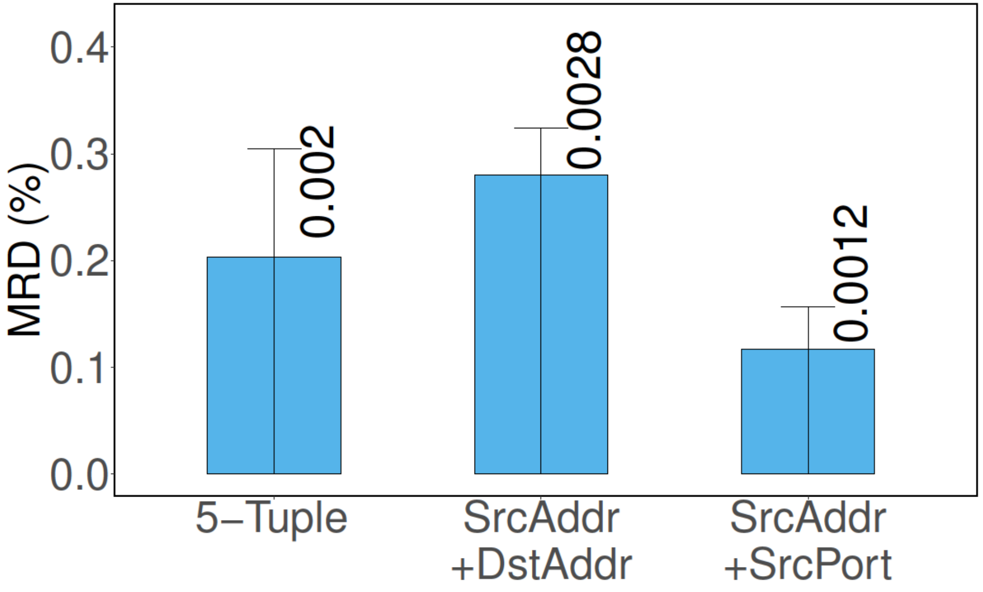
Arbitrary Flow Keys

- Query for three flow keys

Heavy hitter detection



Traffic size distribution



More Experiments

- Resource consumption
- Generality for various measurement tasks
- Efficiency of attached query errors
- Network-wide measurement

Conclusion

- Analyze 5 user burdens in existing approximate measurement
- SketchLearn framework
  - Multi-level data structure design
  - Theory: counters follow Gaussian distributions when no large flows
  - Self-adaptive model inference algorithm
  - Extended query models
- Prototype and evaluations
- Source code available at: https://github.com/huangqundl/SketchLearn

Limitations and Future Work

- Less pipeline consumptions
- Quantify Gaussian distributions and convergence rate
- More applications