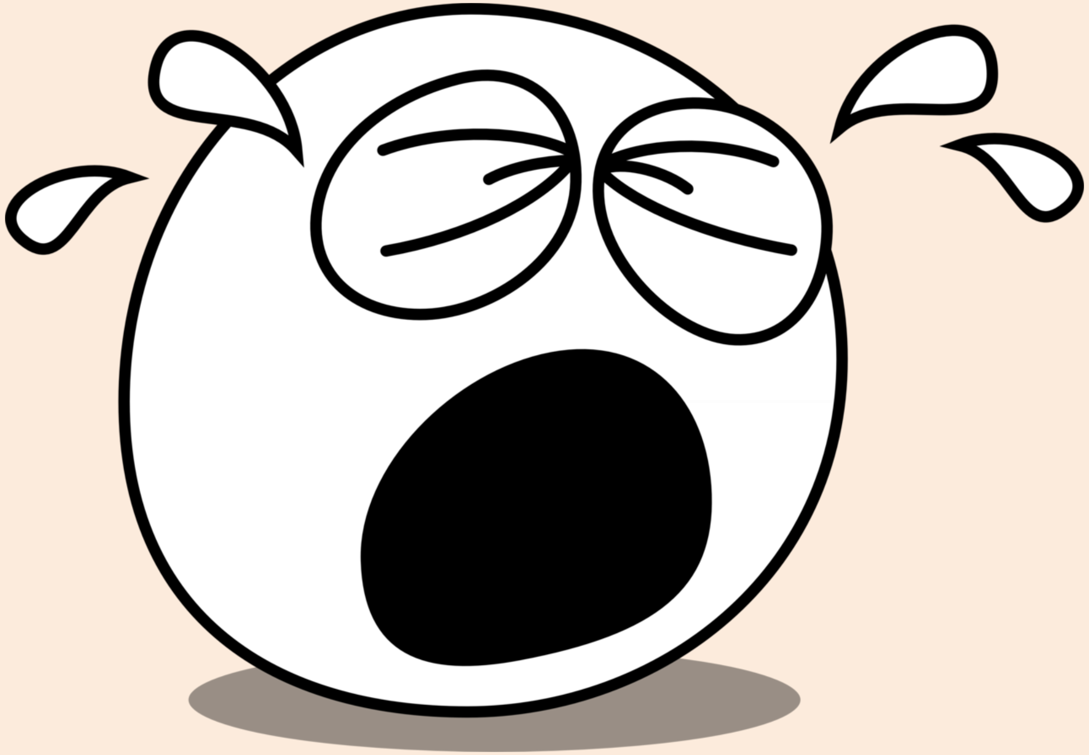
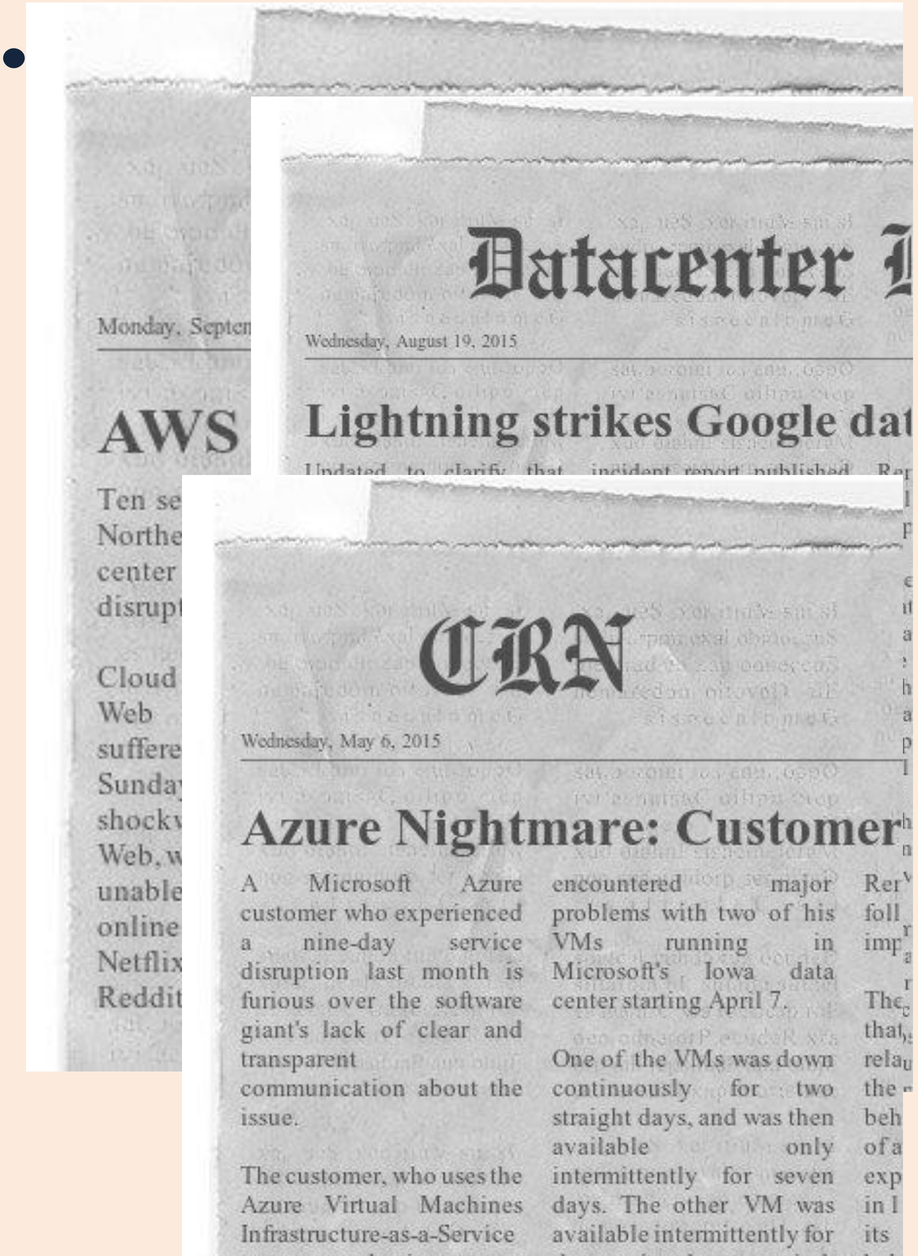
NetPoirot: Taking The Blame Game Out of Data Center Operations

Behnaz Arzani, Selim Ciraci, Boon Thau Loo,

Assaf Schuster, Geoff Outhred

Datacenters can fail ..





Monday, Septem

AWS

Ten se
Northe
center
disrupt

Cloud
Web
suffere
Sunday
shockw
Web, w
unable
online
Netflix
Reddit

Datacenter

Wednesday, August 19, 2015

Lightning strikes Google dat

Updated to clarify that incident report published

CRN

Wednesday, May 6, 2015

Azure Nightmare: Customer

A Microsoft Azure customer who experienced a nine-day service disruption last month is furious over the software giant's lack of clear and transparent communication about the issue.

The customer, who uses the Azure Virtual Machines Infrastructure-as-a-Service encountered major problems with two of his VMs running in Microsoft's Iowa data center starting April 7.

One of the VMs was down continuously for two straight days, and was then available only intermittently for seven days. The other VM was available intermittently for

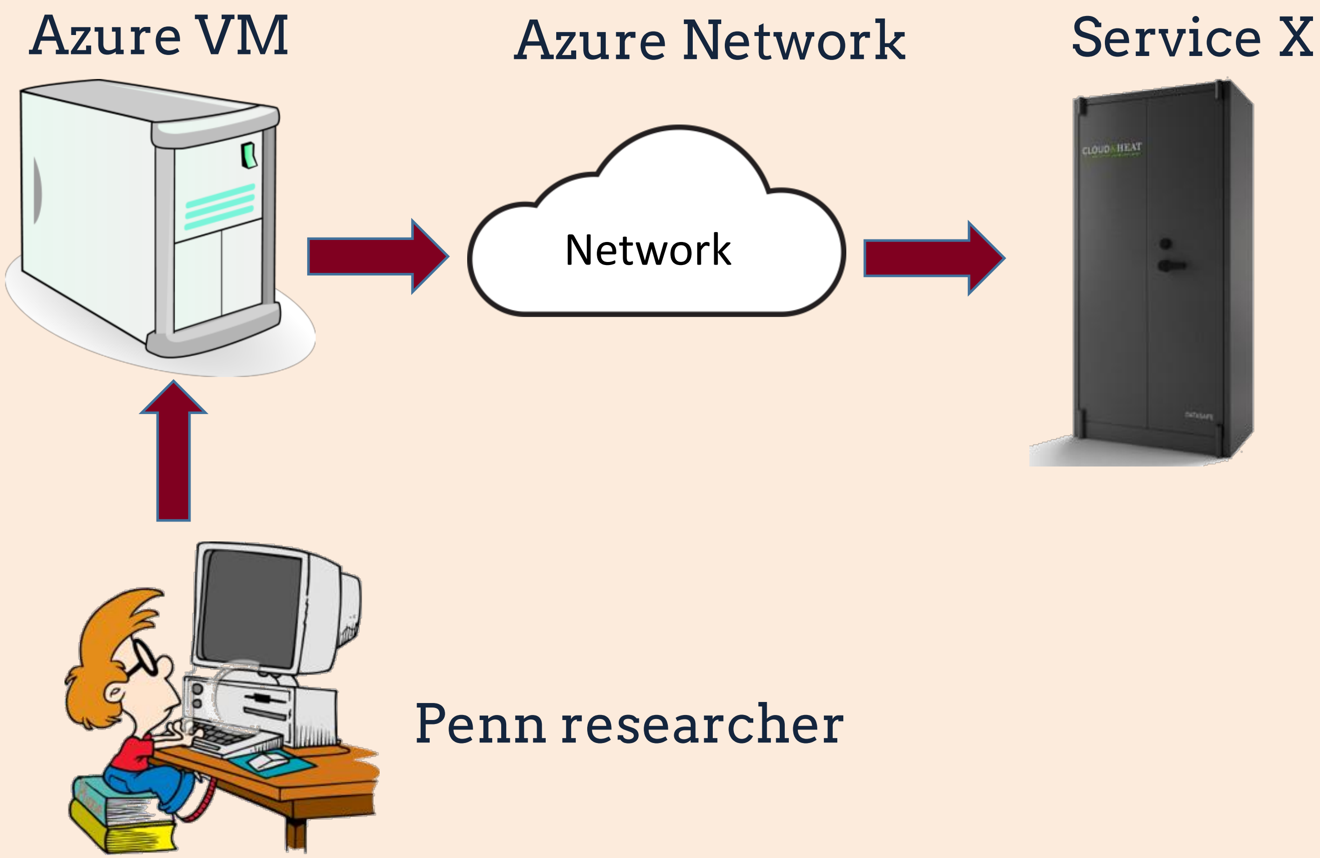
Rer
foll
imp

The
that
rela
the
beh
of a
exp
in l
its

Failures are disruptive

- They can cause significant user downtime
  - Loss of revenue for network providers
  - Lower QoE (Quality of Experience) for the users
- This introduces the need for debugging tools

Why is debugging hard?



Azure VM

Azure Network

Service X

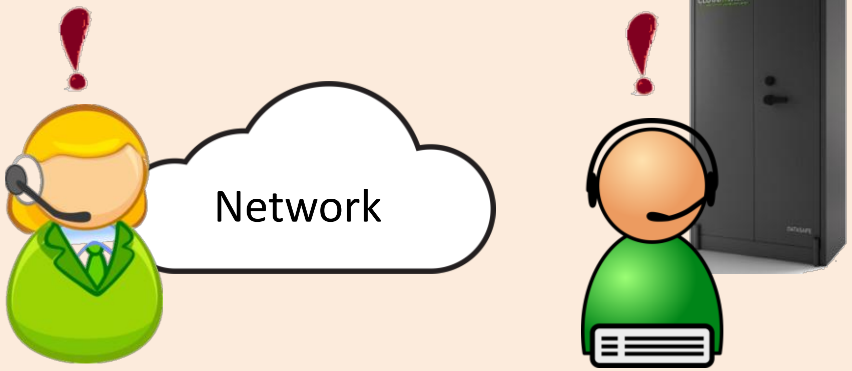
Network

Penn researcher

In the case of a failure…

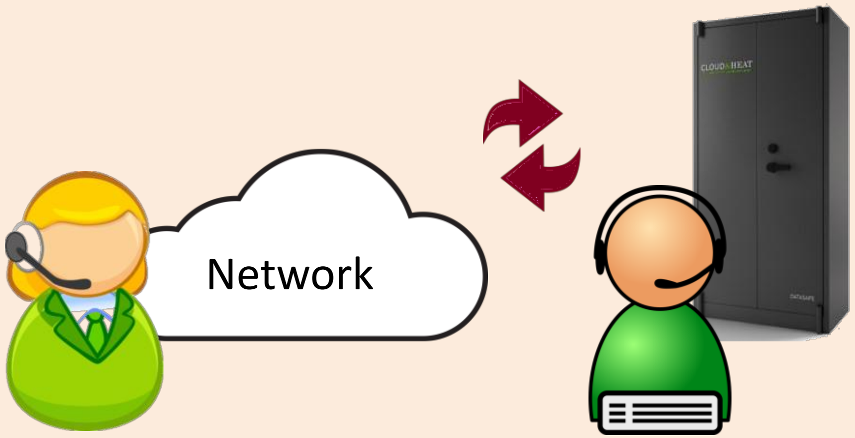
Someone accepts responsibility

✓



Each blames the other

✗



A real example… Event X

- Azure hypervisors connect to a remote service
- If these connections fail, the VM has uncertain state
  - VM has to reboot
- Did the service fail, or was it the network?

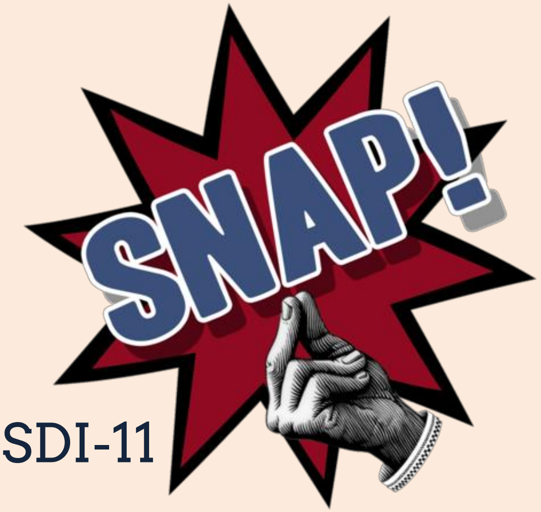
Current tools are insufficient

TRat
SIGCOMM-02

Netprofiler
P2P[illegible]-05

Sherlock
SIGCOMM-07

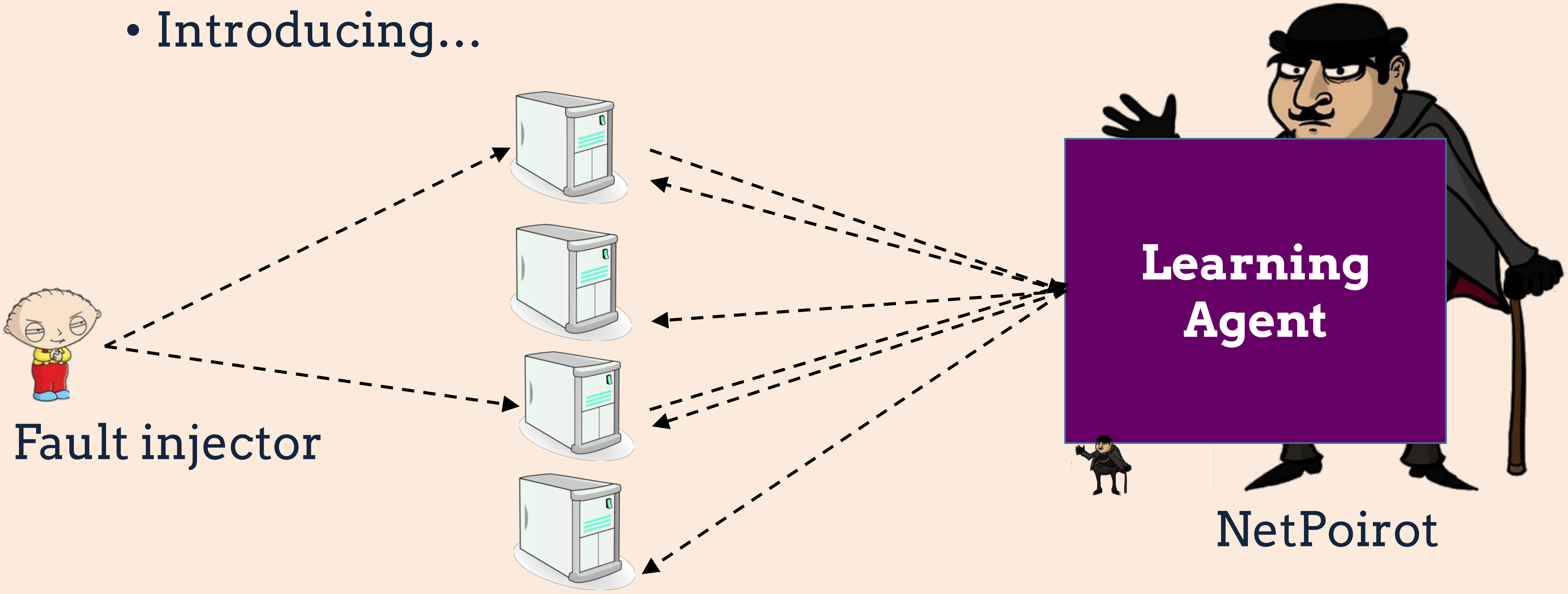
NetMedic
SIGCOMM-09



NSDI-11

Can we do better? (Overview)

- Introducing…



The monitoring agent

- Runs on all *clients* in our data center
- Captures and digests Windows TCP events
- Reports digests every 30 seconds
- Examples of metrics captured
    - Number of duplicate Acks
    - Number of timeouts
    - Time spent in zero window probing

What is the TCP event digest?

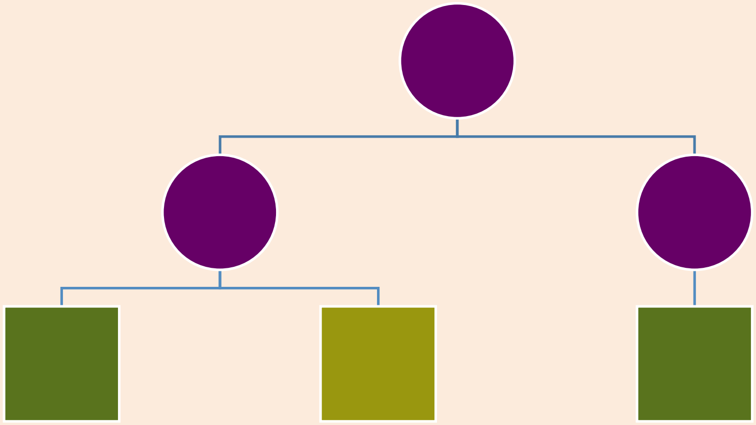
- We aggregate the captured TCP data into epoch digests
- Keep the min, max, 10th,50th,95th percentile, as well as mean and standard deviation across all connections in an epoch for each metric
  - Helps compare performance across the various connections

Why do we think this can work?

- TCP observes the entire communication path
  - It goes through the client, the network, and the server
- It “sees” the failure no matter where it happened
- We know how network failures impact TCP
- How does it react to end point failures?
  - Hard to predict based on protocol design

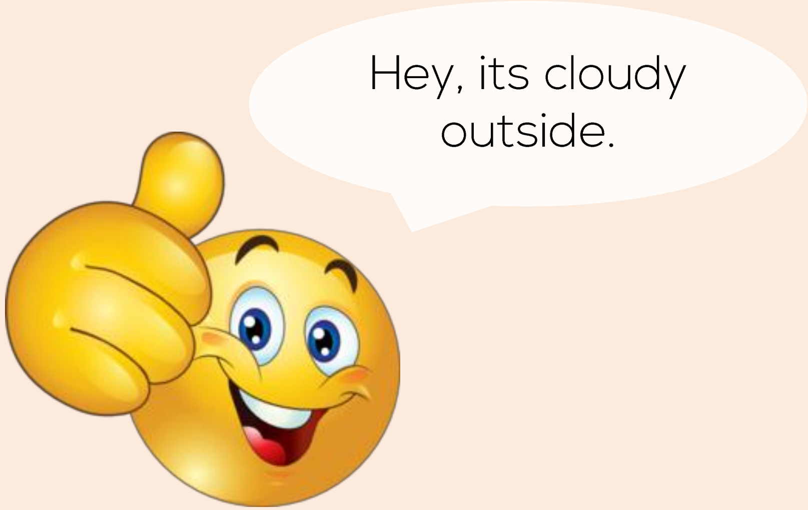
To distinguish failures…

- We use variants of decision trees
- Other algorithms combine/manipulate features
  - Makes it hard to reason about why they arrive at a decision



Decision trees…

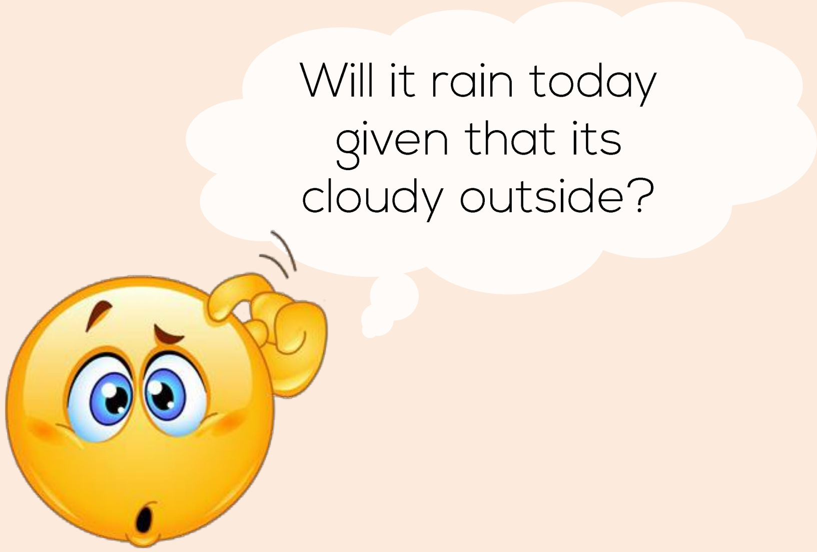
- Greedily pick features that maximize information gain



His uncertainty is X

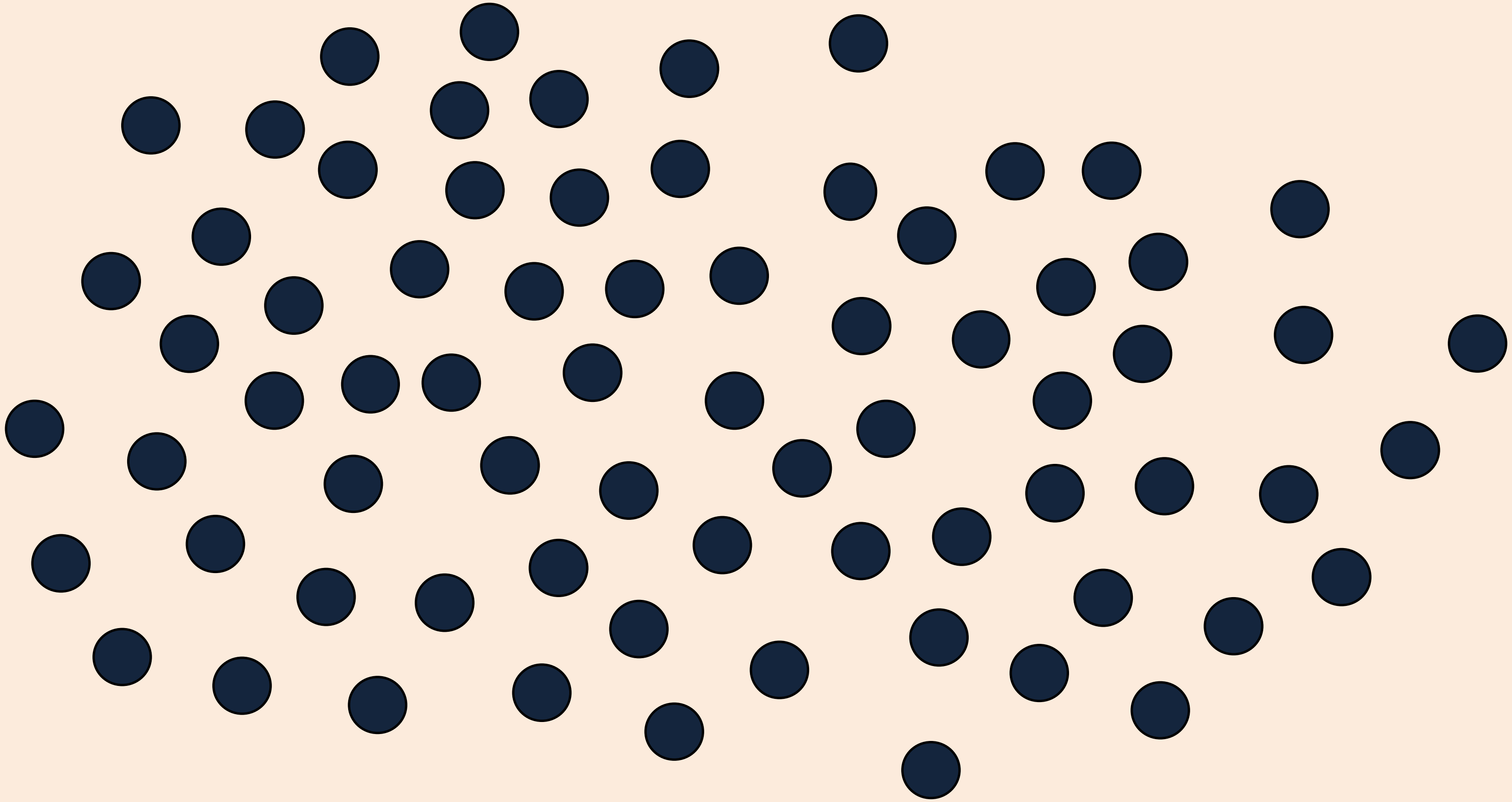
Decision trees...

- Greedily pick features that maximize information gain
  - Pick the most "informative" features in each step

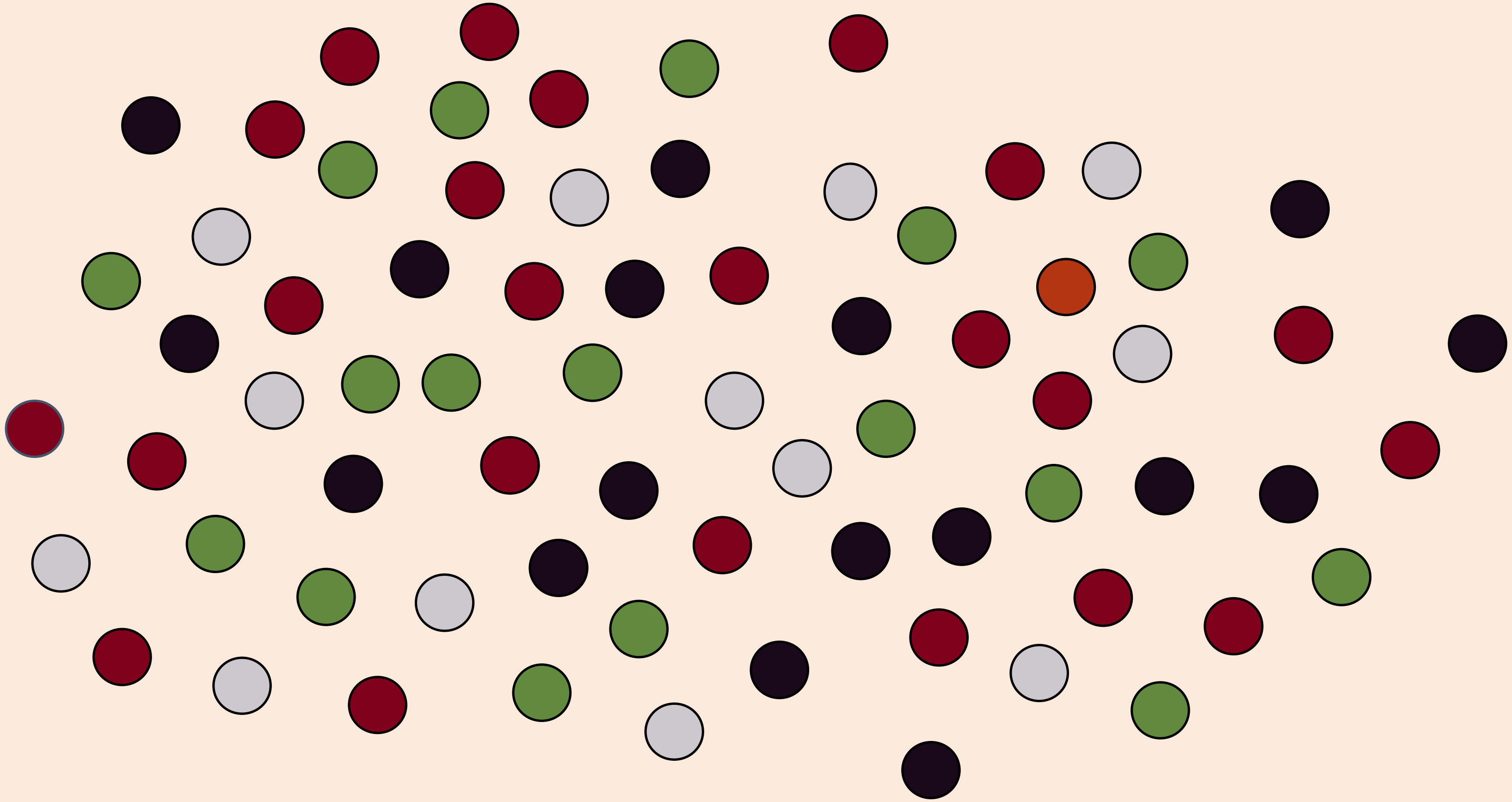


His uncertainty isY X-

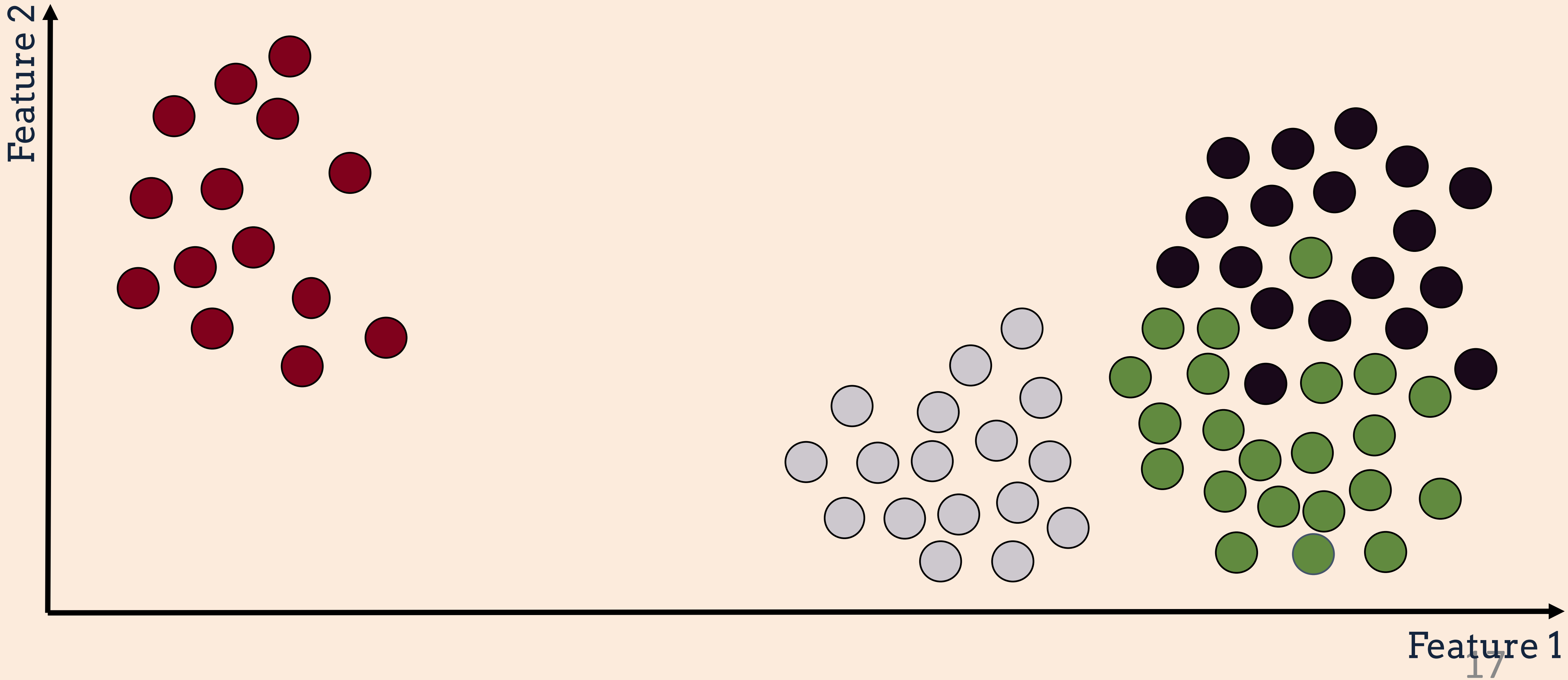
Decision trees alone are not enough



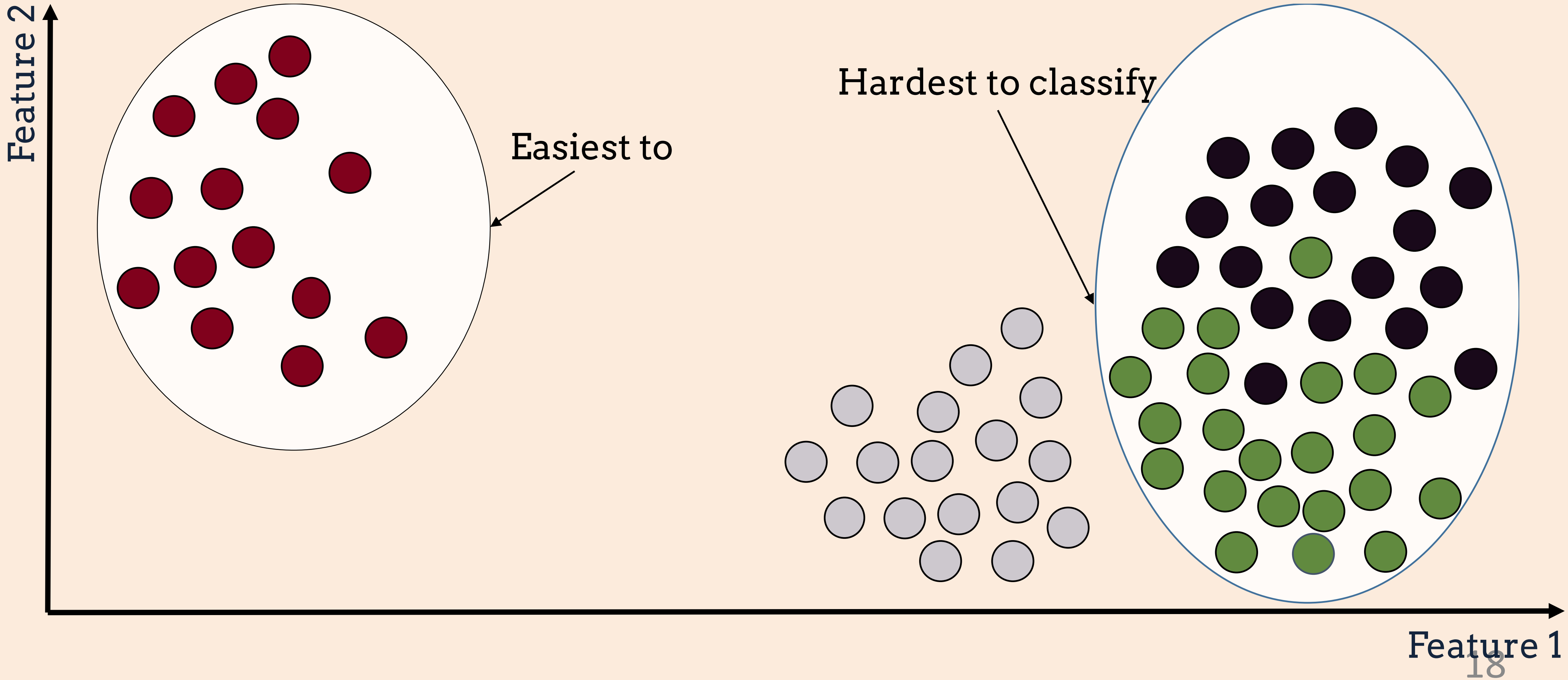
Decision trees alone are not enough



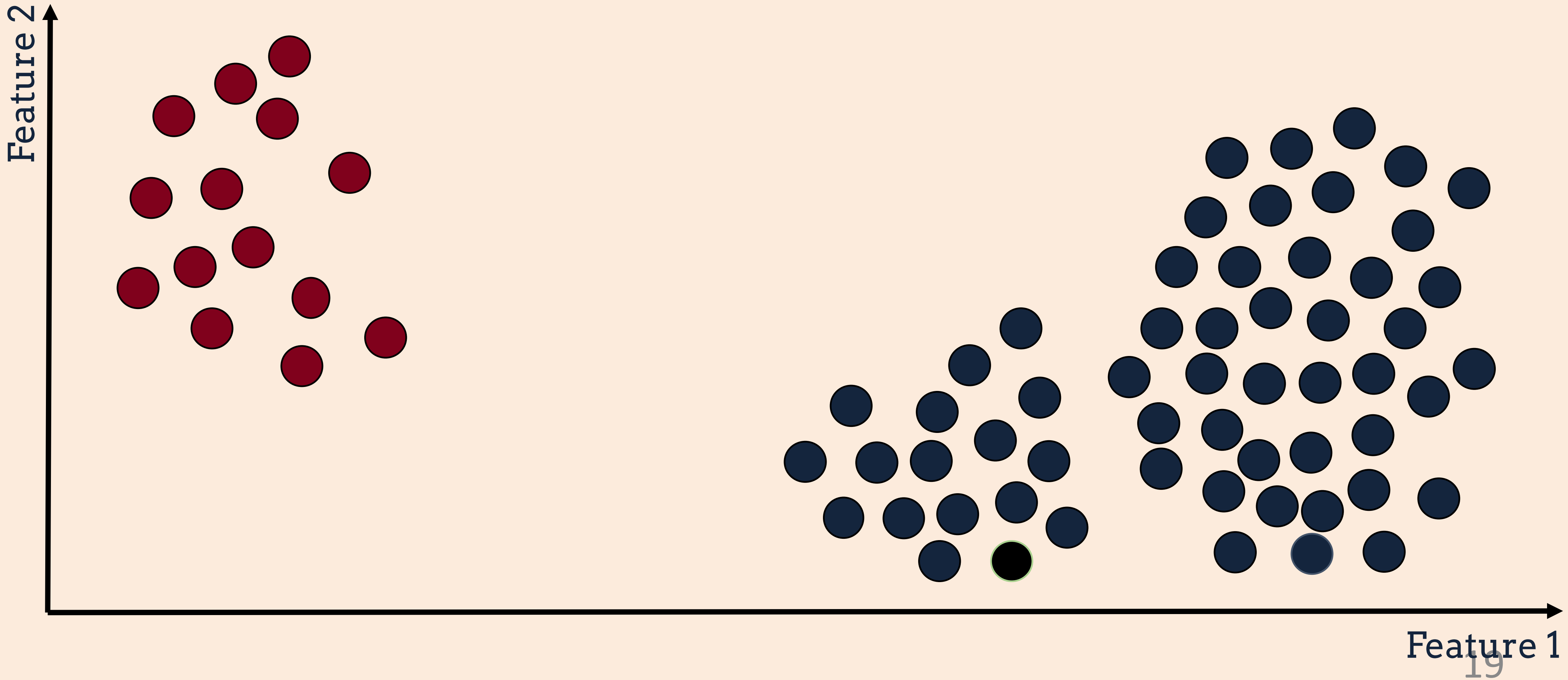
Decision trees alone are not enough



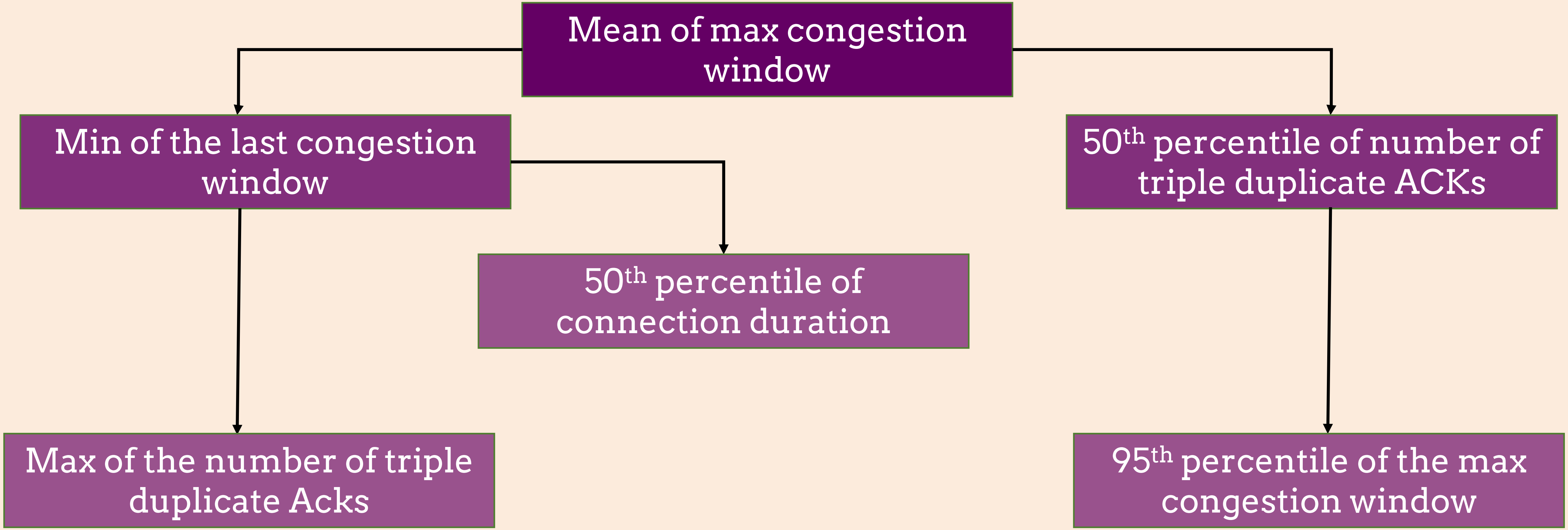
Decision trees alone are not enough



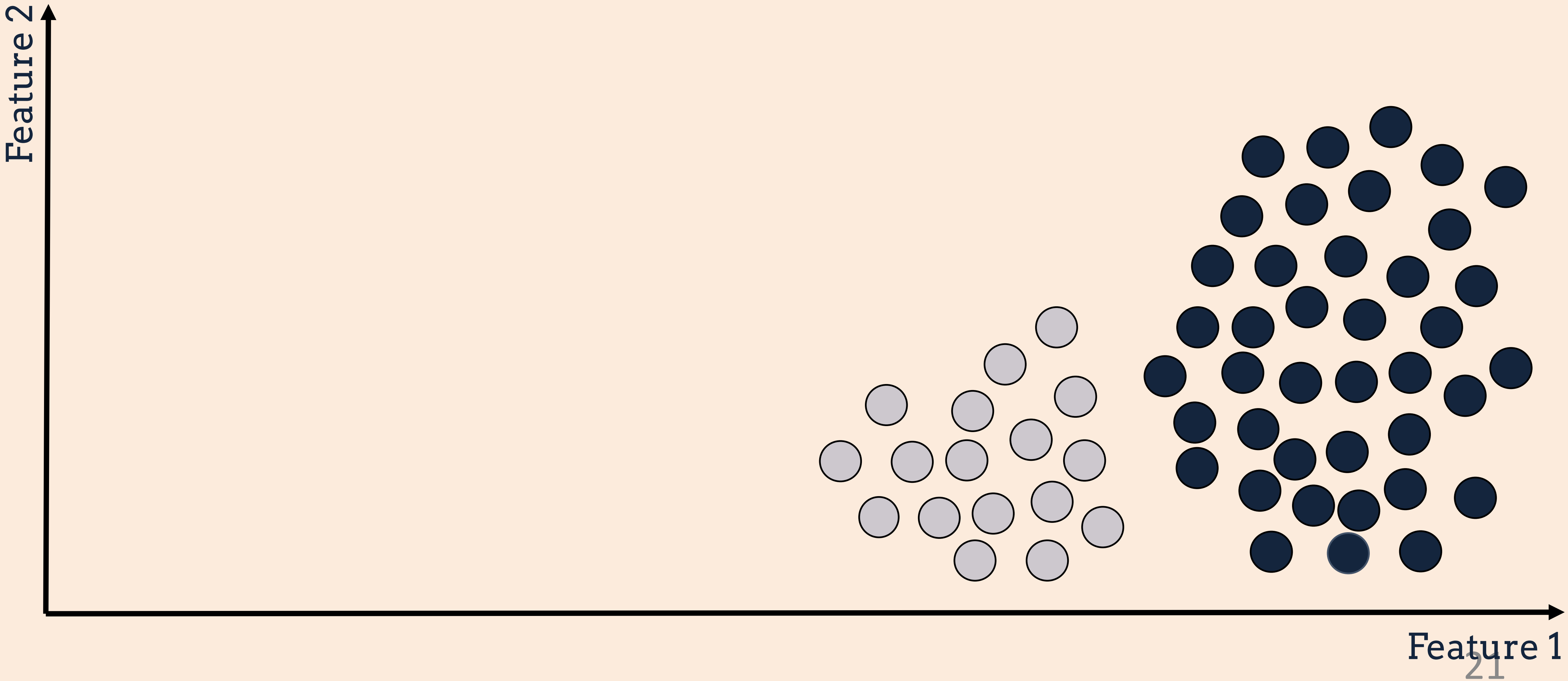
What we do to deal with this



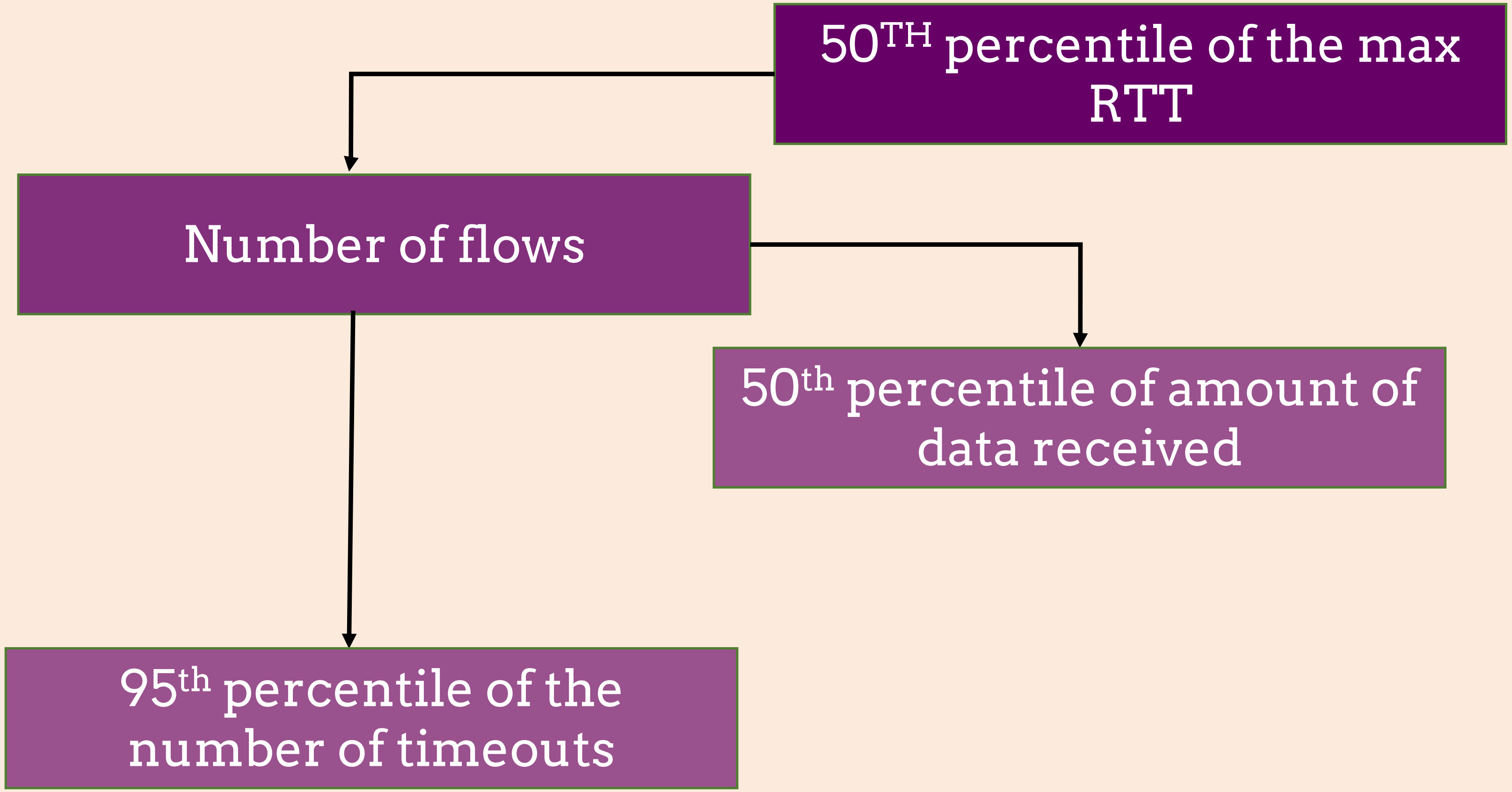
Upper portion of an example tree…


Mean of max congestion window
Min of the last congestion window
50th percentile of number of triple duplicate ACKs
50th percentile of connection duration
Max of the number of triple duplicate Acks
95th percentile of the max congestion window

What we do to deal with this



Upper portion of an example tree…



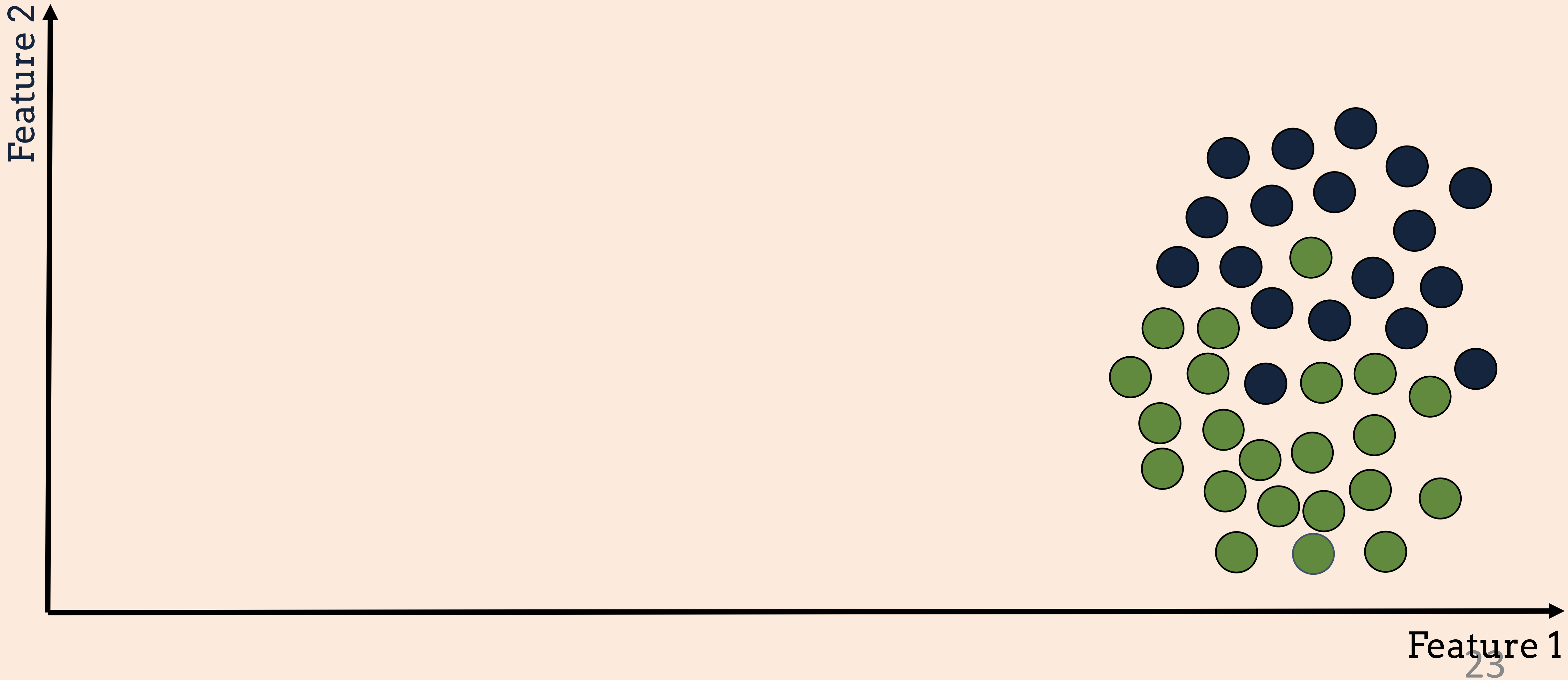
50TH percentile of the max RTT

Number of flows

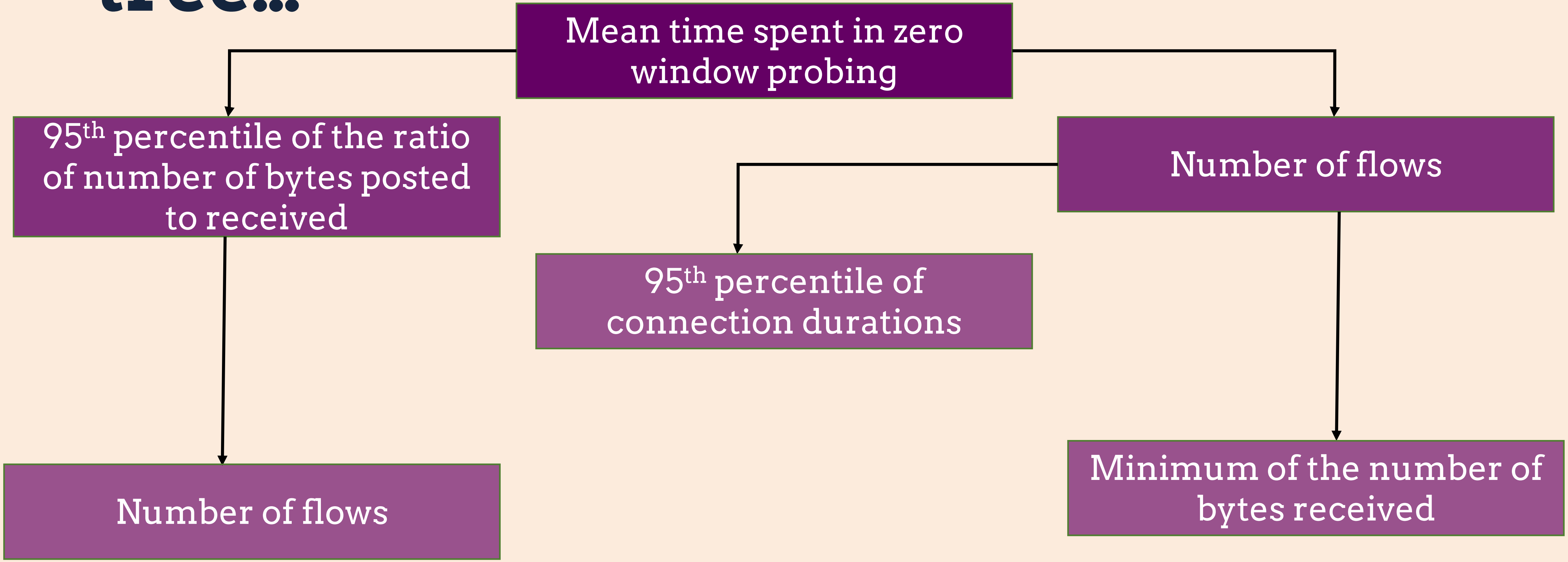
50th percentile of amount of data received

95th percentile of the number of timeouts

Decision trees alone are not enough



The upper portion of an example tree…



Mean time spent in zero window probing

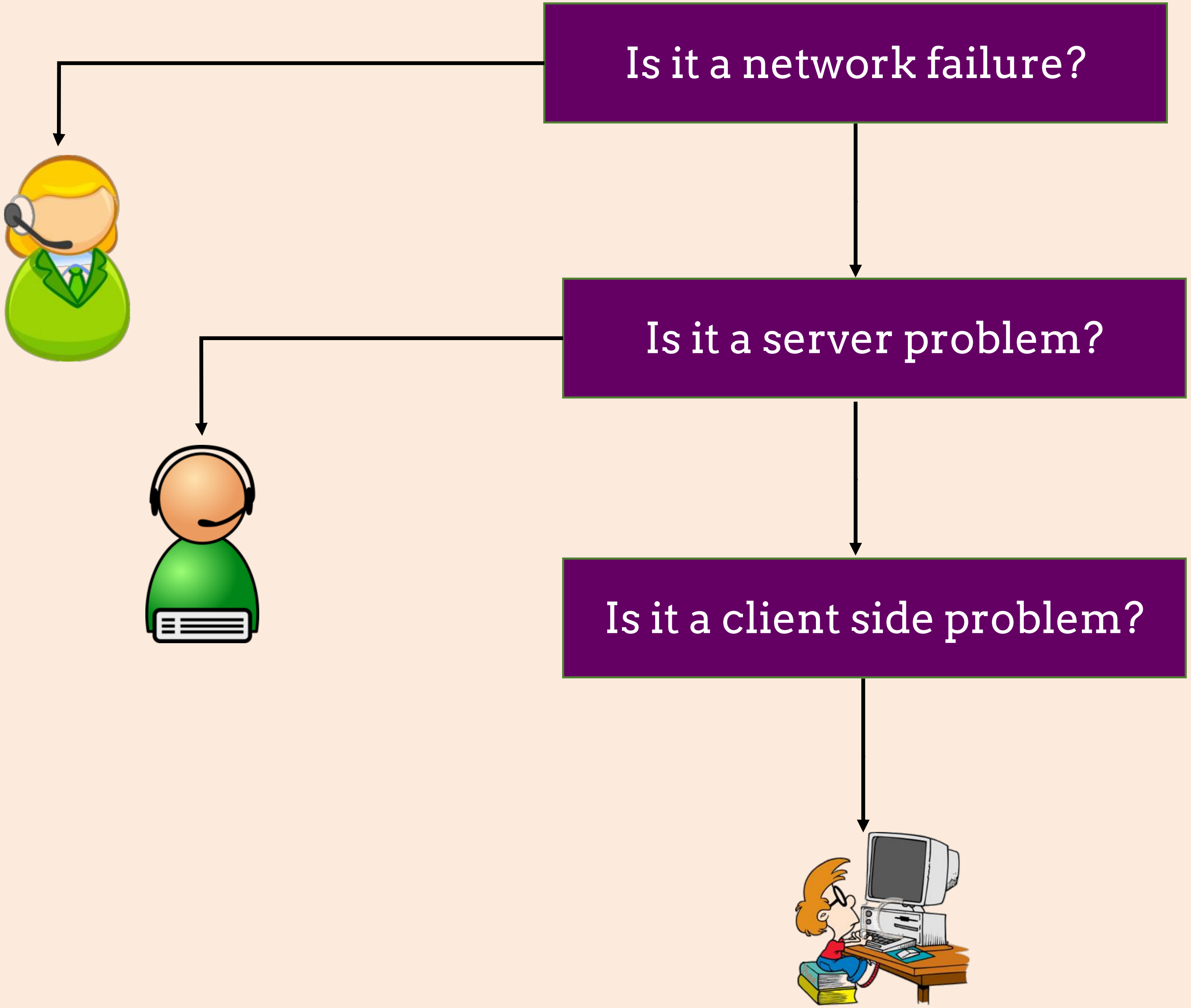
95th percentile of the ratio of number of bytes posted to received

Number of flows

95th percentile of connection durations

Number of flows

Minimum of the number of bytes received



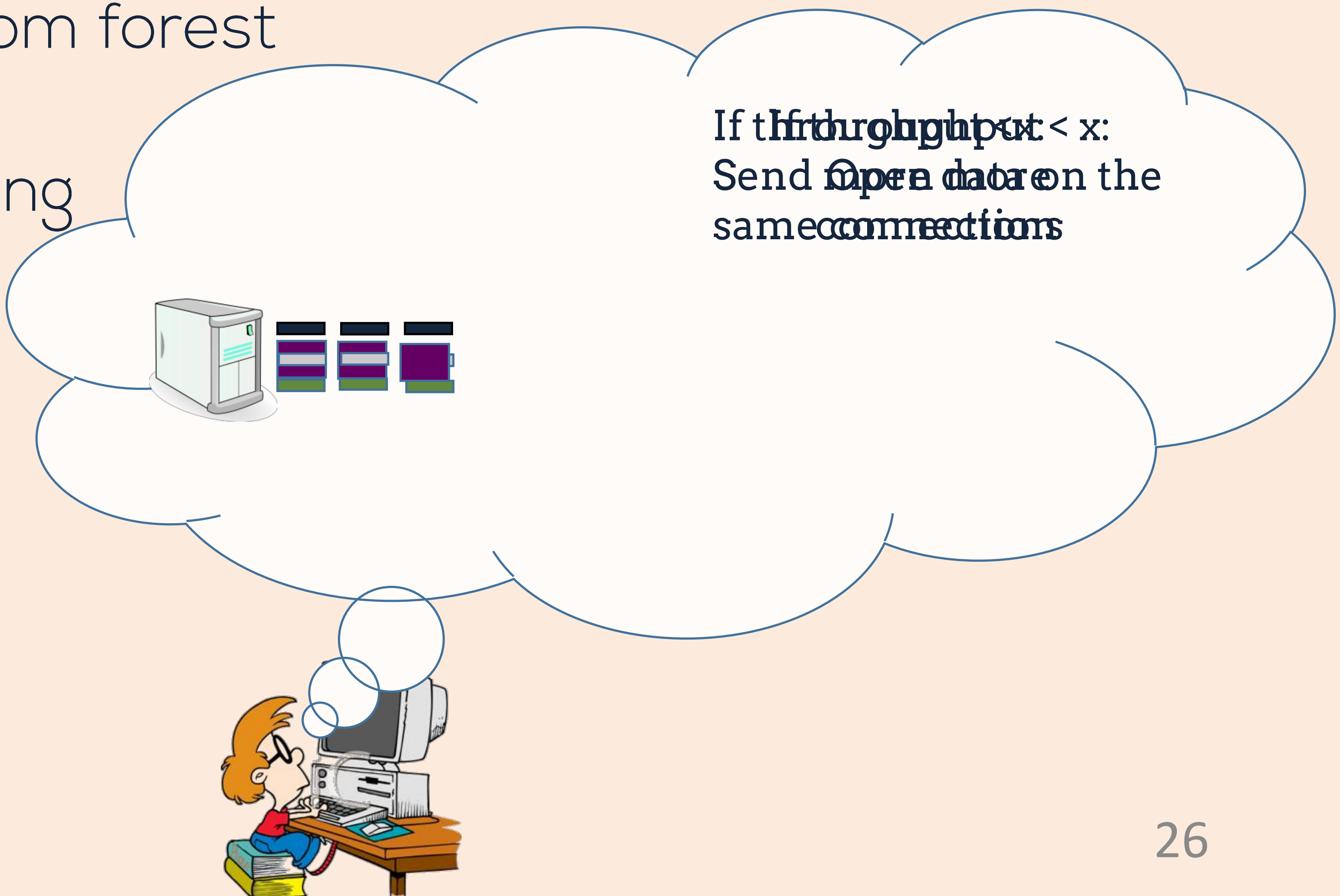
Is it a network failure?

Is it a server problem?

Is it a client side problem?

Other details

- We had to use random forest
  - More stable
- Per application training
- Normalize the data



If throughput < x:
Send more data on the same connection

If throughput < x:
Open more connections

What did we learn from all this?

- TCP sees everything, even at a single end point
  - Allows us to find who was responsible for a failure
- Failures in a group (Client/Server/Network) are similar
  - Makes individual failure classification more challenging
  - Helps NetPoirot be resilient to failures we haven't seen in the past
- The relationship between failures and TCP metrics is non-linear
  - Pearson correlation is low
- Two features suffice to describe each failure as observed by TCP
  - Two largest eigenvalues of the data matrix capture 95% of its variance

Evaluation

- What is the worst case performance?
    - Applications react to failures
    - Their reactions provide useful information
    - But what if this information is not available?

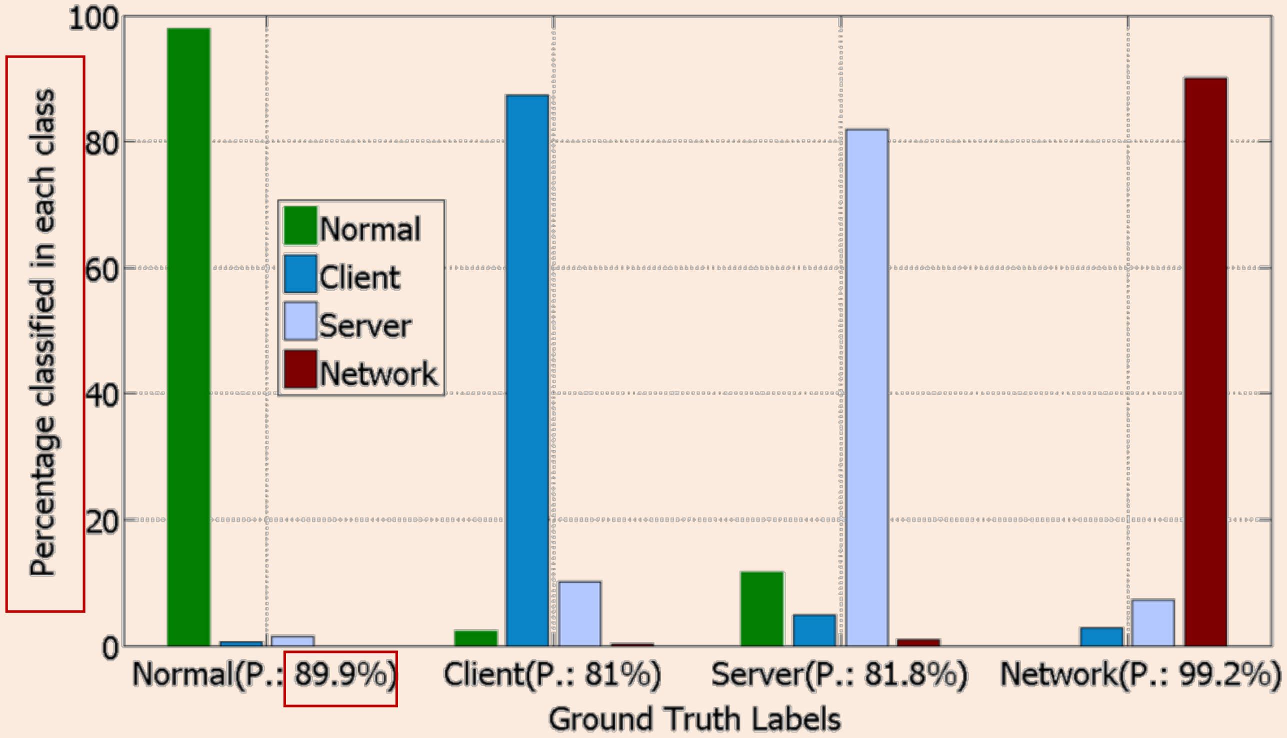
- What if we did not anticipate a failure type?
    - Dormant failures
    - Unknown failures

How did we get labeled data?

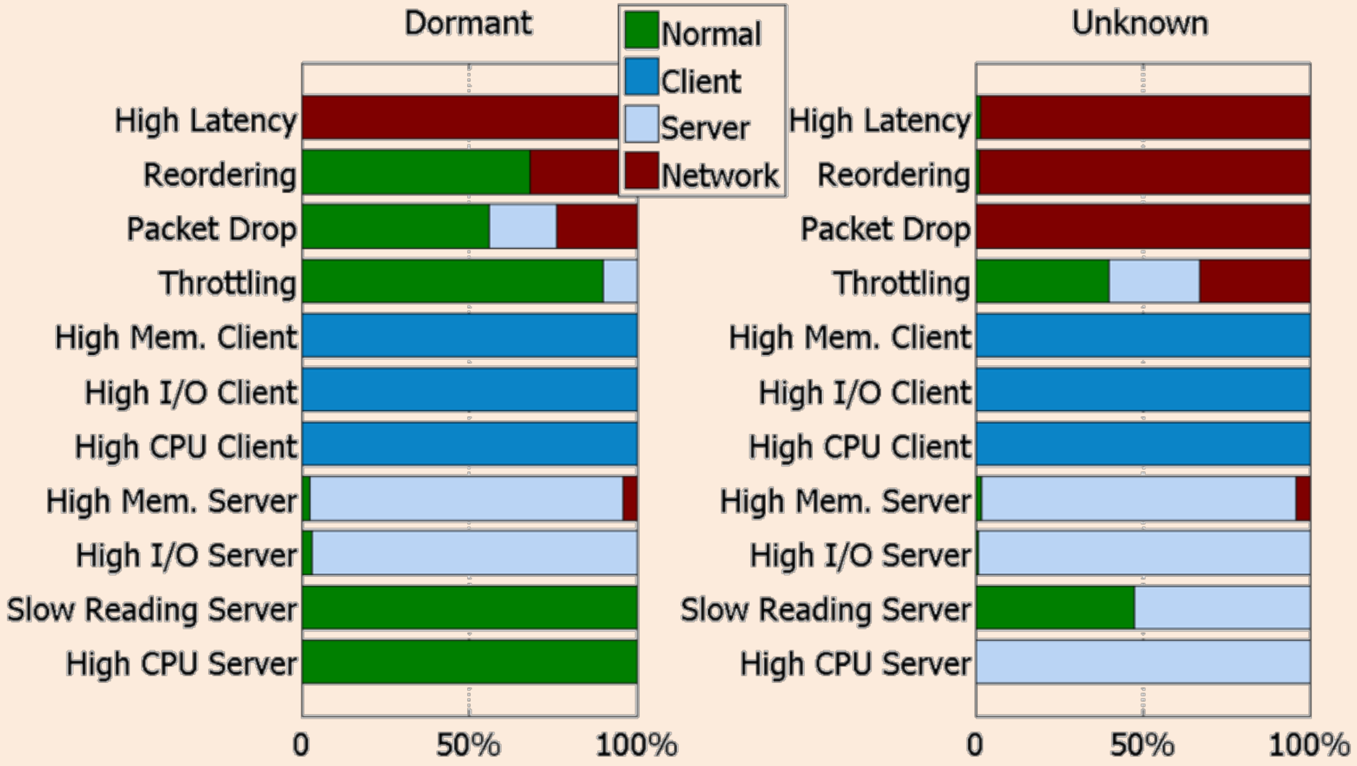
- We inject faults into the communication
- Over 6 months of data
- Examples:
    - High CPU load on the client
    - High I/O load on the server
    - Bandwidth throttling in the network
    - Packet reordering in the network

Worse case application

- Only TCP statistics are used from the client side machine



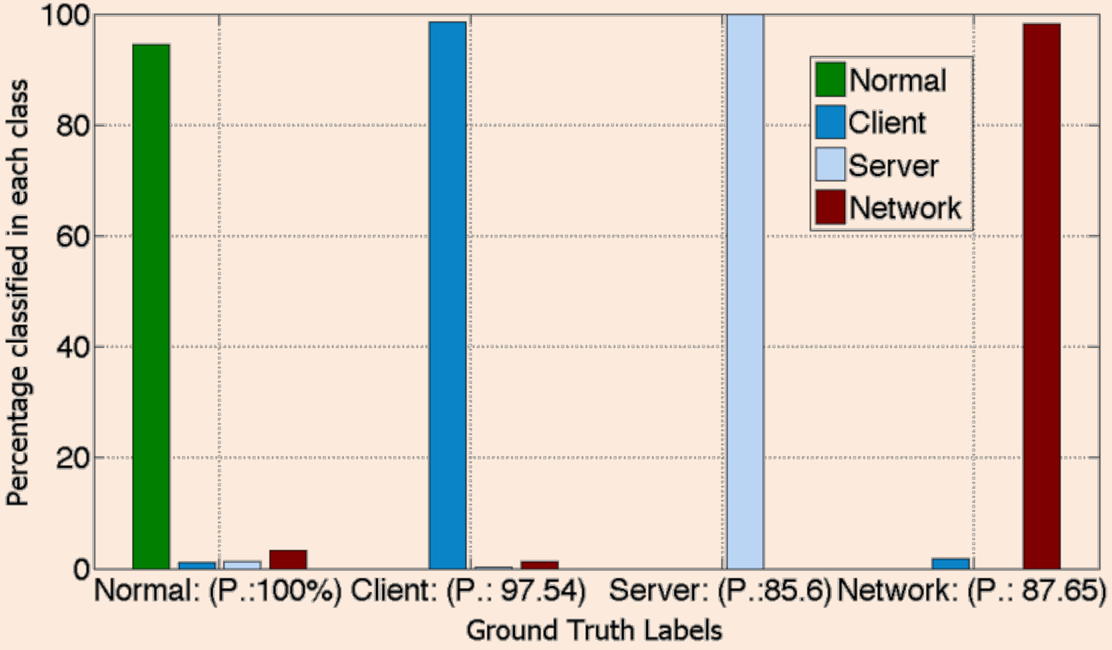
What if we haven’t seen the failure before?



Performance on real applications

General label	Normal	Client	Network
Precision	97.78%	99.7%	100%
Recall	99.68%	98.25%	99.37

YouTube



Event X

Things we did not talk about

- Identifying the actual type of failure
- Sensitivity to machine location
- Aggregation vs per connection classification
- Sensitivity to failure duration
- Modifications to traditional cross validation required

What's next?

- Can we make this application independent?
  - Transfer learning
- Can the end point identify the device causing the failure?
  - Correlate information across clients

Conclusion

- TCP's reactions to network and endpoint failures are significantly different
- We can utilize these differences to find the entity that caused the failure