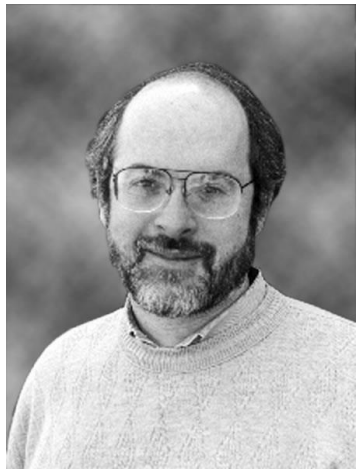
Tools for Scalable Data Mining

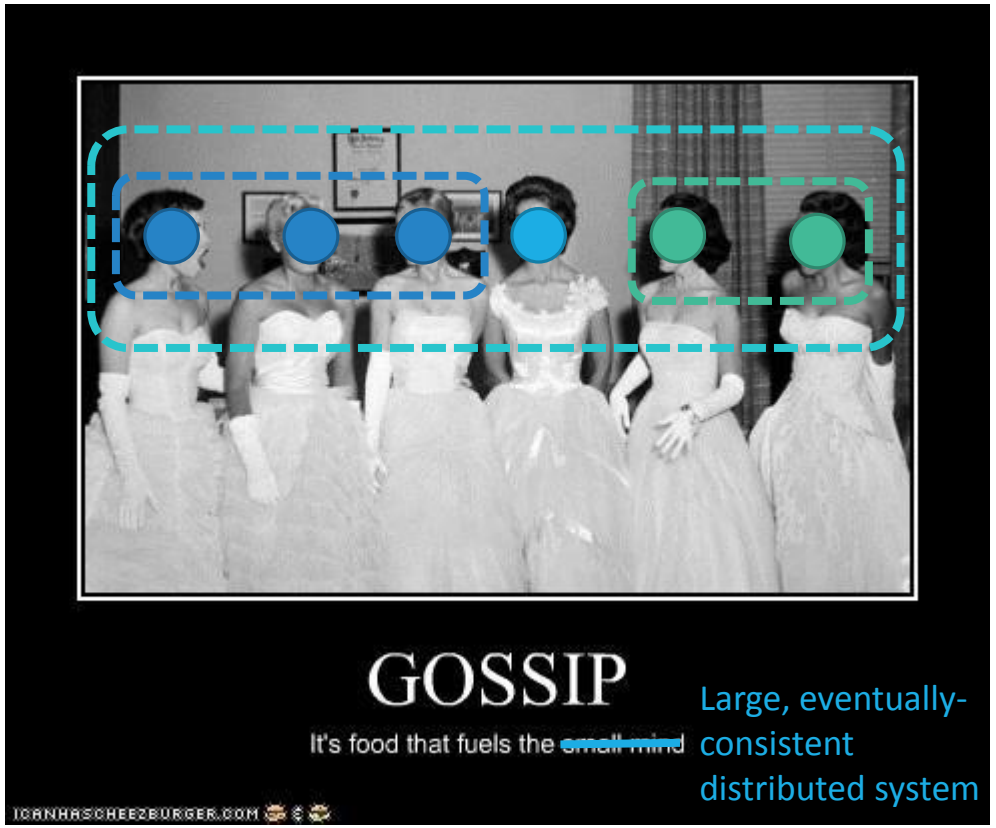
XANDA SCHOFIELD

CS 6410

11/13/2014


1. Astrolabe



ROBERT VAN RENESSE, KEN BIRMAN, WERNER VOGELS

[Source: Wikipedia]

The Problem

How do we quickly find out information about overall distributed system state?

- Classic consensus protocols: very accurate but almost certainly slow
- Pure Gossip: fast and correct in some cases, slow and approximate in others

System management becomes a **data mining problem.**

A solution: Astrolabe

Impose some hierarchy (a **spanning tree** on nodes)

- Replication across layers
- Computation up through the layers

Compute via the tree

- Leaf values report information from one host
- Child nodes report to their parents
- Replication makes this accurate and O(log N)

[Source: Wikipedia]

Named **Astrolabe** based on the instrument for helping sailors find their latitude in rough water

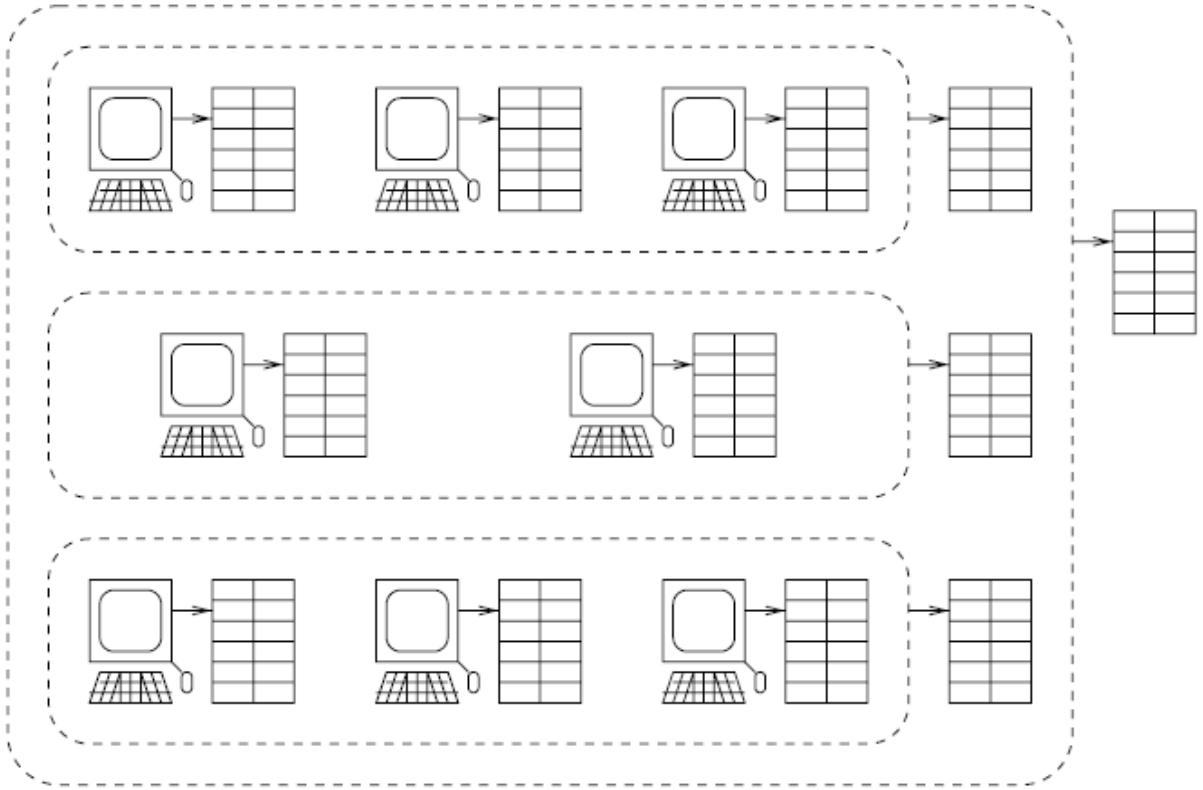
What does Astrolabe offer?

- **Scalability:** efficient aggregation with hierarchical structure
- **Flexibility:** mobile code in SQL query form
- **Robustness:** decentralized random P2P communication
- **Security:** signatures with scalable verification

Zones and MIBs

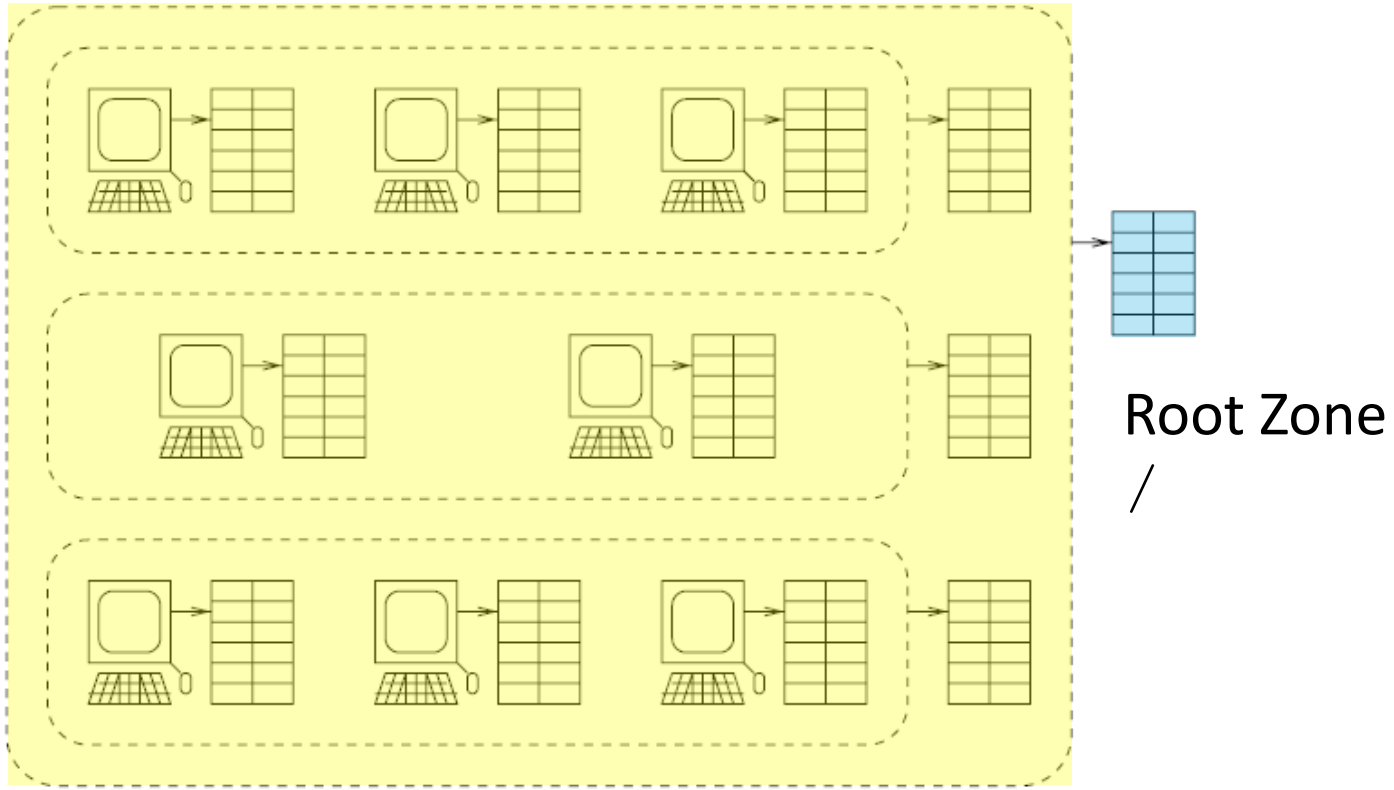
Example
System Map

---- → zones



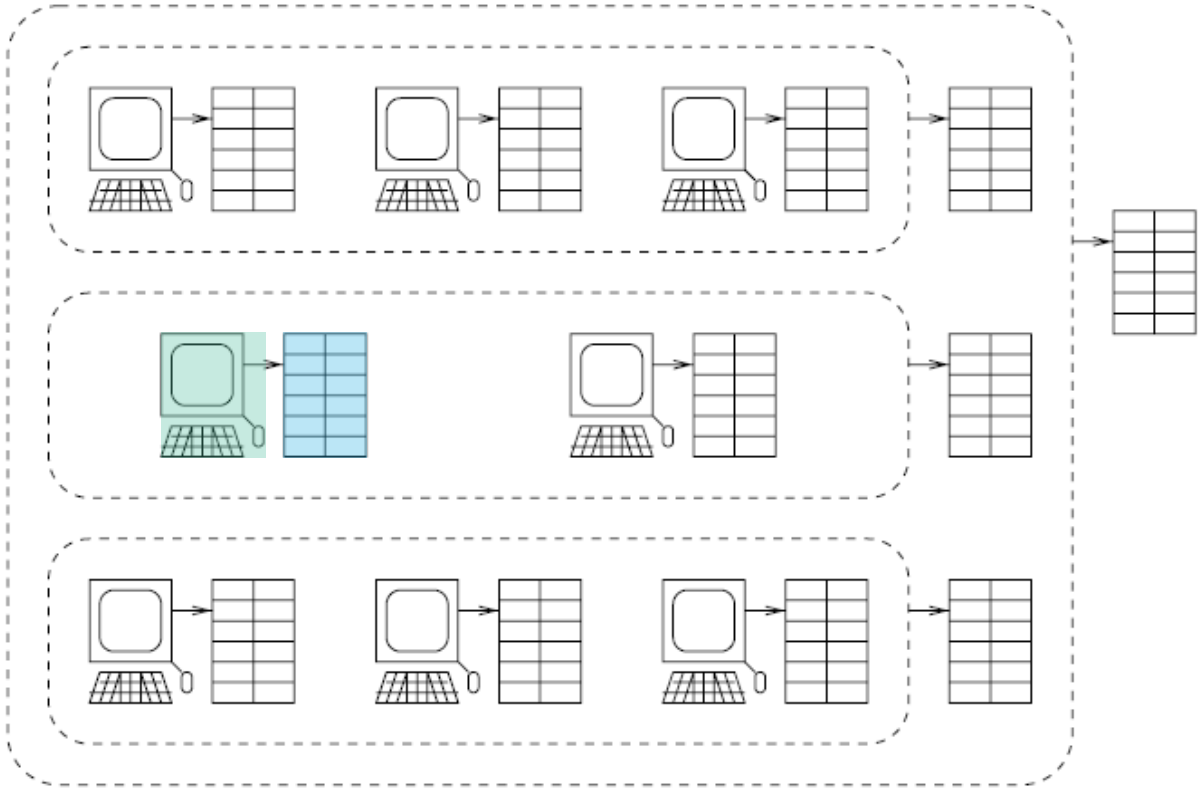
[Source: Astrolabe paper]

Zones and MIBs

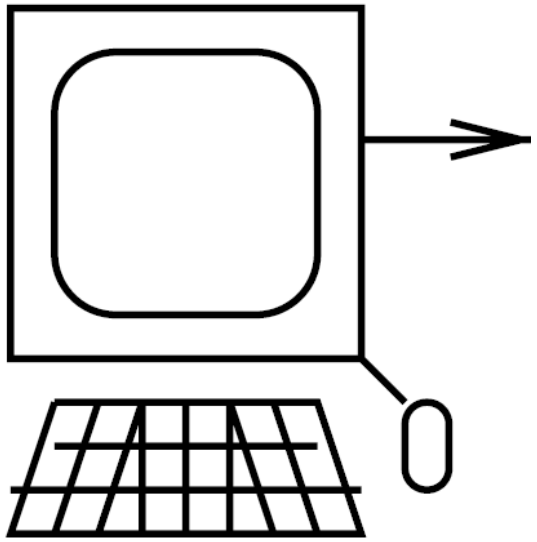


Zones and MIBs

Leaf Zone
`/Cornell/pc3/`



Leaf Nodes



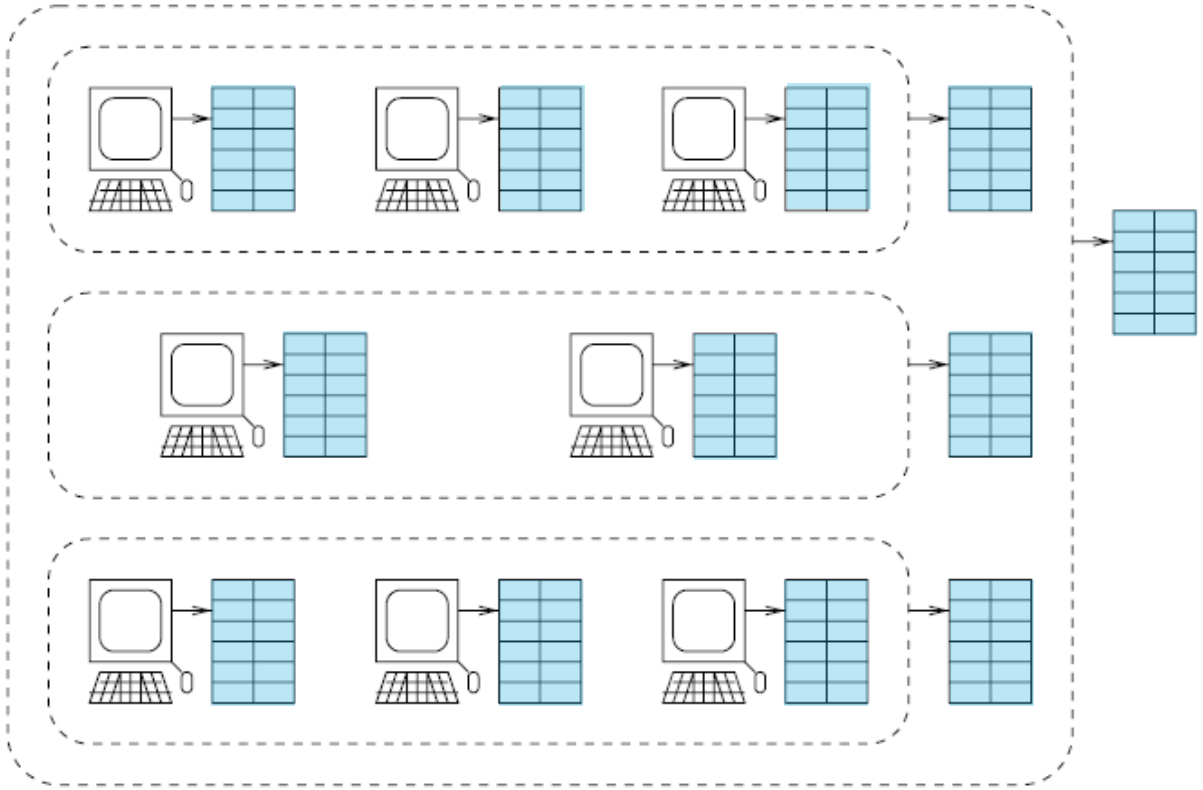
Broken into 1 or more **virtual child zones**

- Initialized with one: “system”
- Others created by the local application
- Locally readable and writeable via the Astrolabe API

Supply the information to aggregate across the system

Zones and MIBs

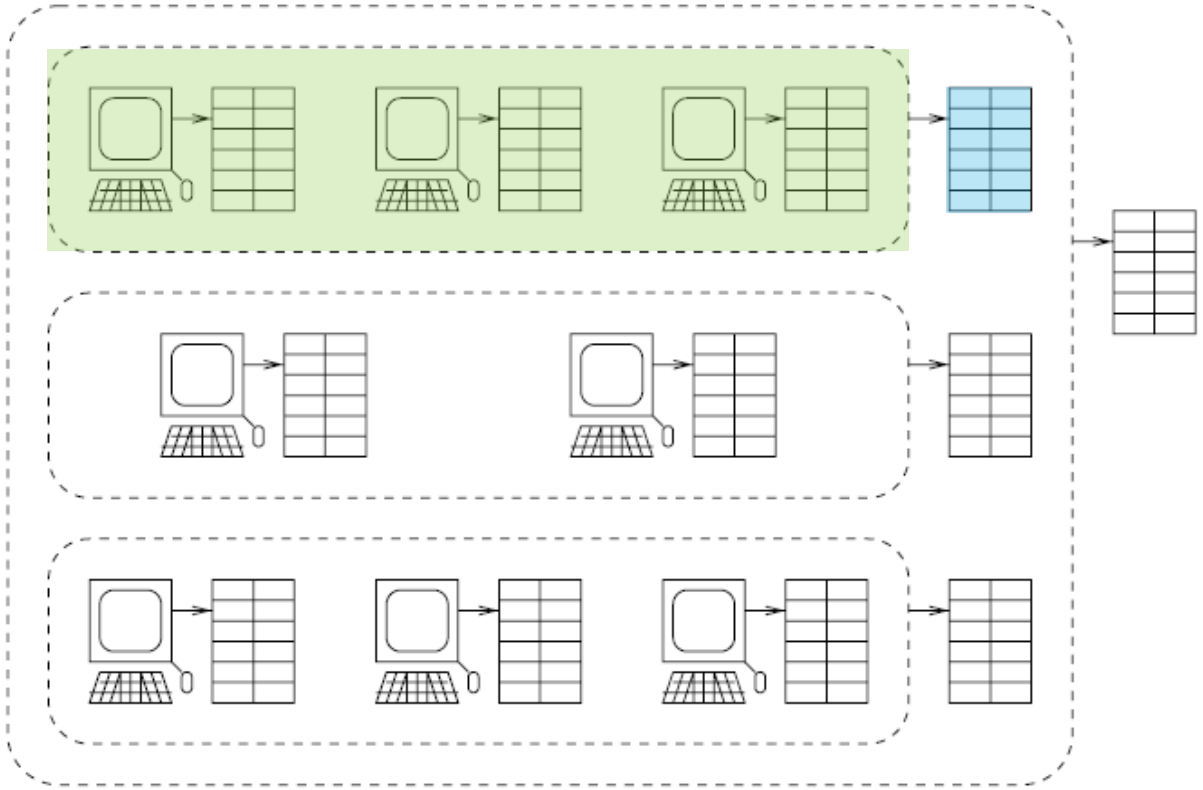
MIBs:
Management
Information
Bases



Zones and MIBs

Child Zone
`/Cornell/`

- Nodes locate each other through broadcast and gossip
- Nodes replicate each other via periodic random merges



Example Merge

1. Pick two nodes to merge information

lion.cs.cornell.edu MIB

Name	Time	Load	SMTP?	Python
lion	1417	1.1	1	V2.6
tiger	1347	1.6	0	V2.7.2
cheetah	1399	4.1	0	V2.4

cheetah.cs.cornell.edu MIB

Name	Time	Load	SMTP?	Python
lion	1325	2.0	1	V2.6
tiger	1398	1.3	0	V2.7.2
cheetah	1421	0.3	1	V2.4

[Example adapted from CS 5412 slides]

Example Merge

1. Pick two nodes to merge information
2. Swap information about all sibling MIBs

lion.cs.cornell.edu MIB

Name	Time	Load	SMTP?	Python
lion	1417	1.1	1	V2.6
tiger	1347	1.6	0	V2.7.2
cheetah	1399	4.1	0	V2.4

cheetah.cs.cornell.edu MIB

Name	Time	Load	SMTP?	Python
lion	1325	2.0	1	V2.6
tiger	1398	1.3	0	V2.7.2
cheetah	1421	0.3	1	V2.4

Example Merge

1. Pick two nodes to merge information
2. Swap information about all sibling MIBs
3. Update based on timestamp

lion.cs.cornell.edu MIB

Name	Time	Load	SMTP?	Python
lion	**1417**	1.1	1	V2.6
tiger	1347	1.6	0	V2.7.2
cheetah	1399	4.1	0	V2.4

cheetah.cs.cornell.edu MIB

Name	Time	Load	SMTP?	Python
lion	1325	2.0	1	V2.6
tiger	**1398**	1.3	0	V2.7.2
cheetah	**1421**	0.3	1	V2.4

Example Merge

1. Pick two nodes to merge information
2. Swap information about all sibling MIBs
3. Update based on timestamp

lion.cs.cornell.edu MIB

Name	Time	Load	SMTP?	Python
lion	1417	1.1	1	V2.6
tiger	1398	1.3	0	V2.7.2
cheetah	1421	0.3	1	V2.4

cheetah.cs.cornell.edu MIB

Name	Time	Load	SMTP?	Python
lion	1417	1.1	1	V2.6
tiger	1398	1.3	0	V2.7.2
cheetah	1421	0.3	1	V2.4

How far off is this from consistent?

The node is still updating its own information

By the next round of gossip, these will likely look different.

lion.cs.cornell.edu MIB

Name	Time	Load	SMTP?	Python
lion	1438	1.6	1	V2.6
tiger	1398	1.3	0	V2.7.2
cheetah	1421	0.3	1	V2.4

cheetah.cs.cornell.edu MIB

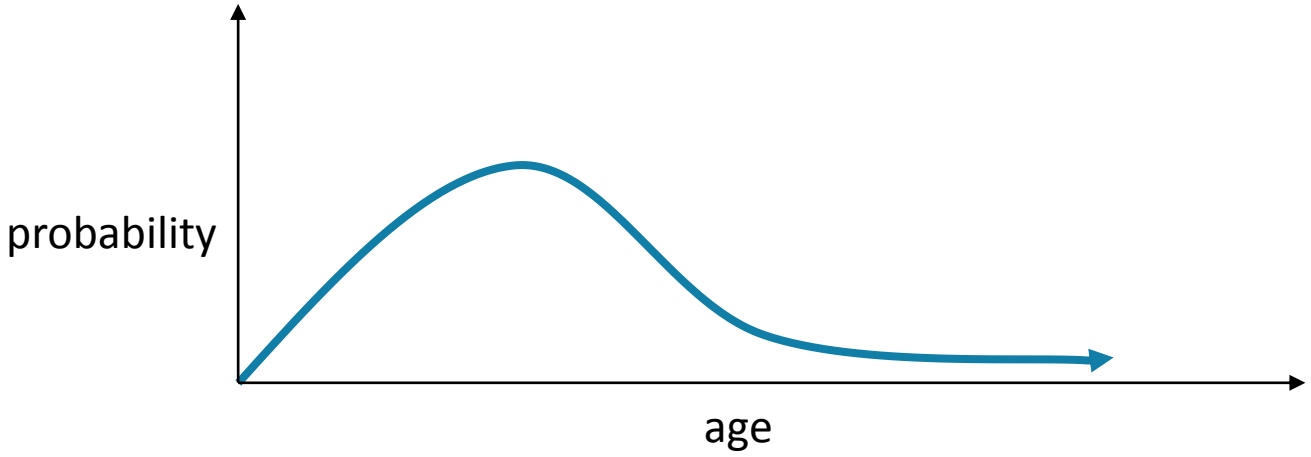
Name	Time	Load	SMTP?	Python
lion	1382	1.1	1	V2.6
tiger	1426	1.4	0	V2.7.2
cheetah	1433	0.5	0	V2.4

Stochastic replication

The collection of MIBs is effectively a database

Instances in a zone replicate that database

For a given non-local row, there is a **probability distribution** for how up-to-date data is



Stochastic replication

Easy or hard with gossip?

- “How many nodes are there?”
- “Tell me the average load across all nodes.”
- “Tell me which nodes don’t have this patch.”
- “If you are the last node in the room, turn off the light when you leave”.

Easy

Easy to approximate

Maybe outdated

Hard

Constructing MIBs

AFCs: **A**ggregation **F**unction **C**ertificates – signed SQL programs for computing attributes from child MIBs

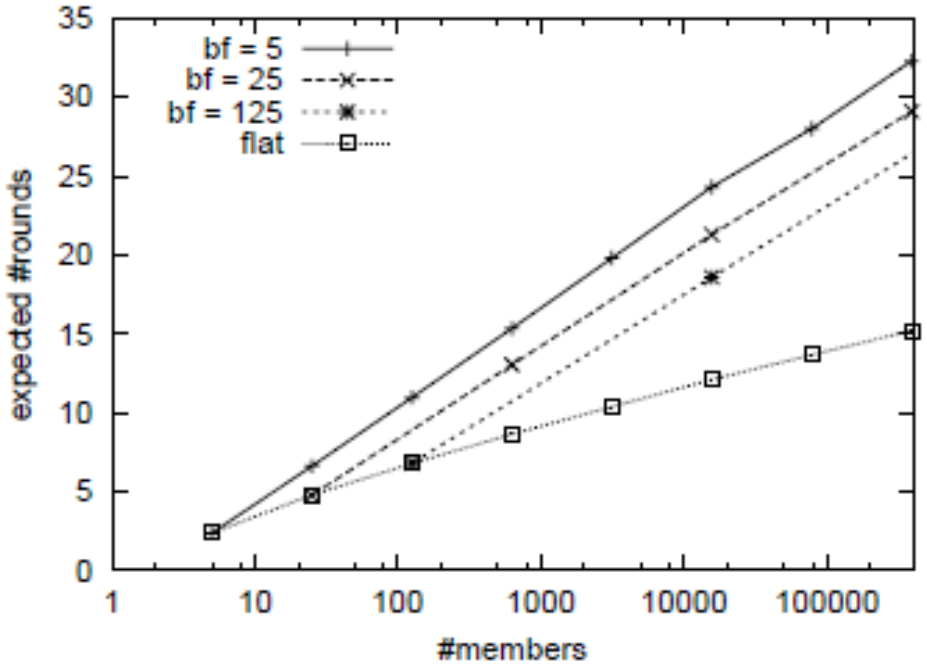
Scalable: AFCs are small and fast and *limited in number in a node*

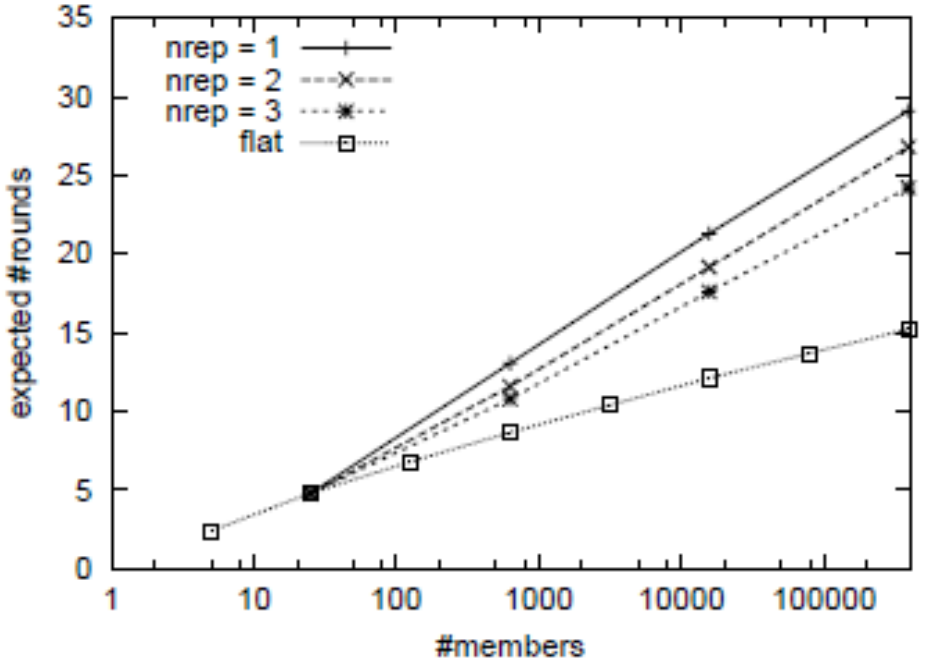
Flexible: SQL syntax can be applied to whatever MIB values are available at the level below *so long as results don't grow at O(n)*

Robust: computed hierarchically efficiently by elected representative nodes for each zone

Secure: certificates are used to verify zone IDs, AFCs, MIBs, and clients based on keys from a trusted CA

How fast is it?





Where does it struggle or fail?

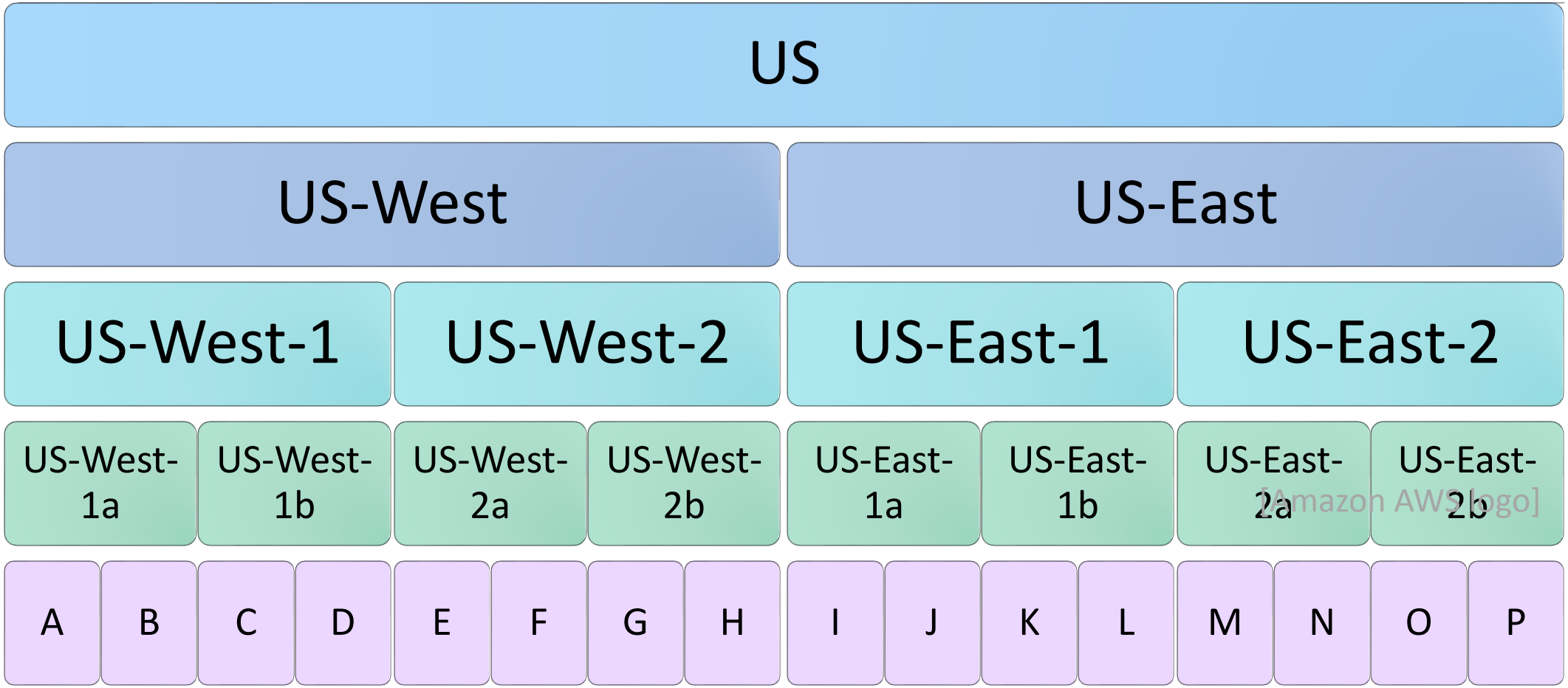
Too many AFCs? **Messages get too big.**

Not enough representatives per zone? **Node fails hurt.**

Too many representatives per zone? **Networks saturate.**

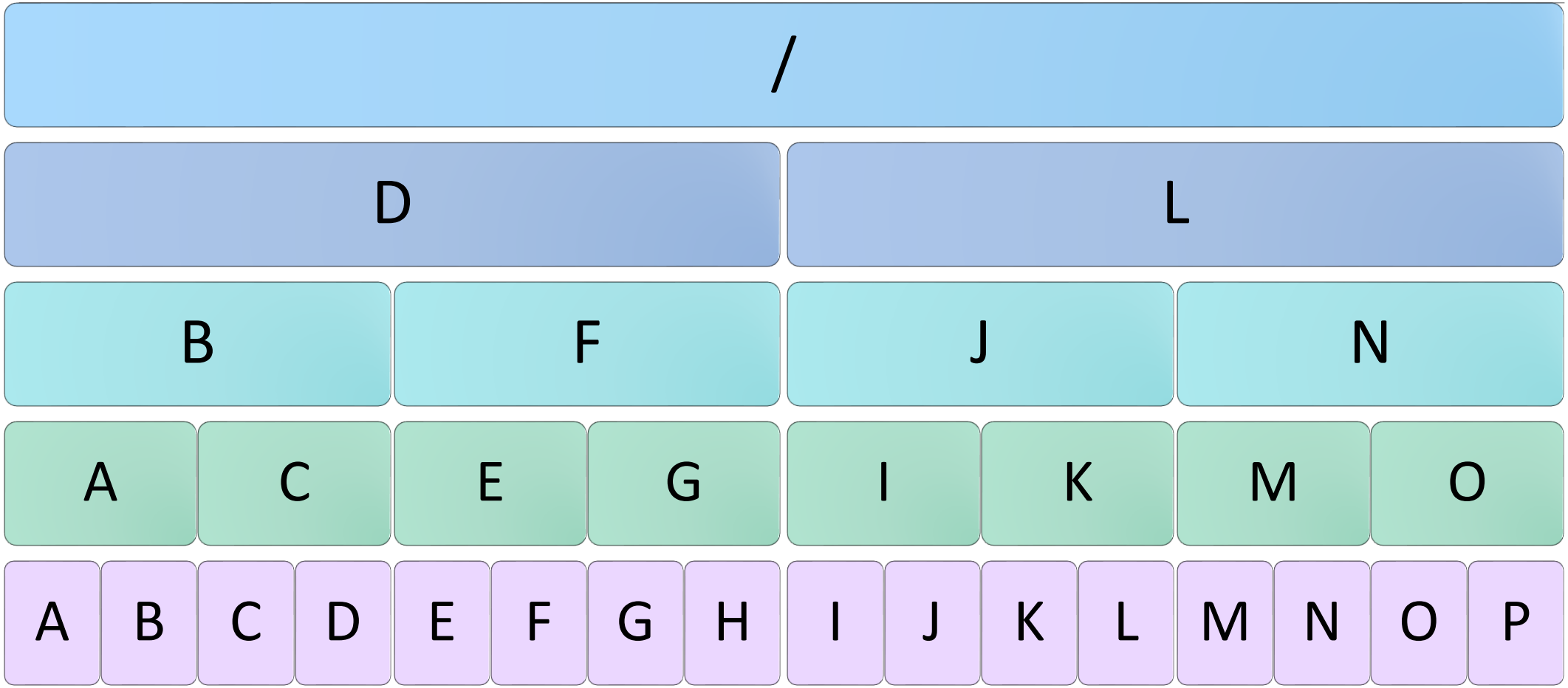
Balancing work too well? **Paths get long.**

The Tree

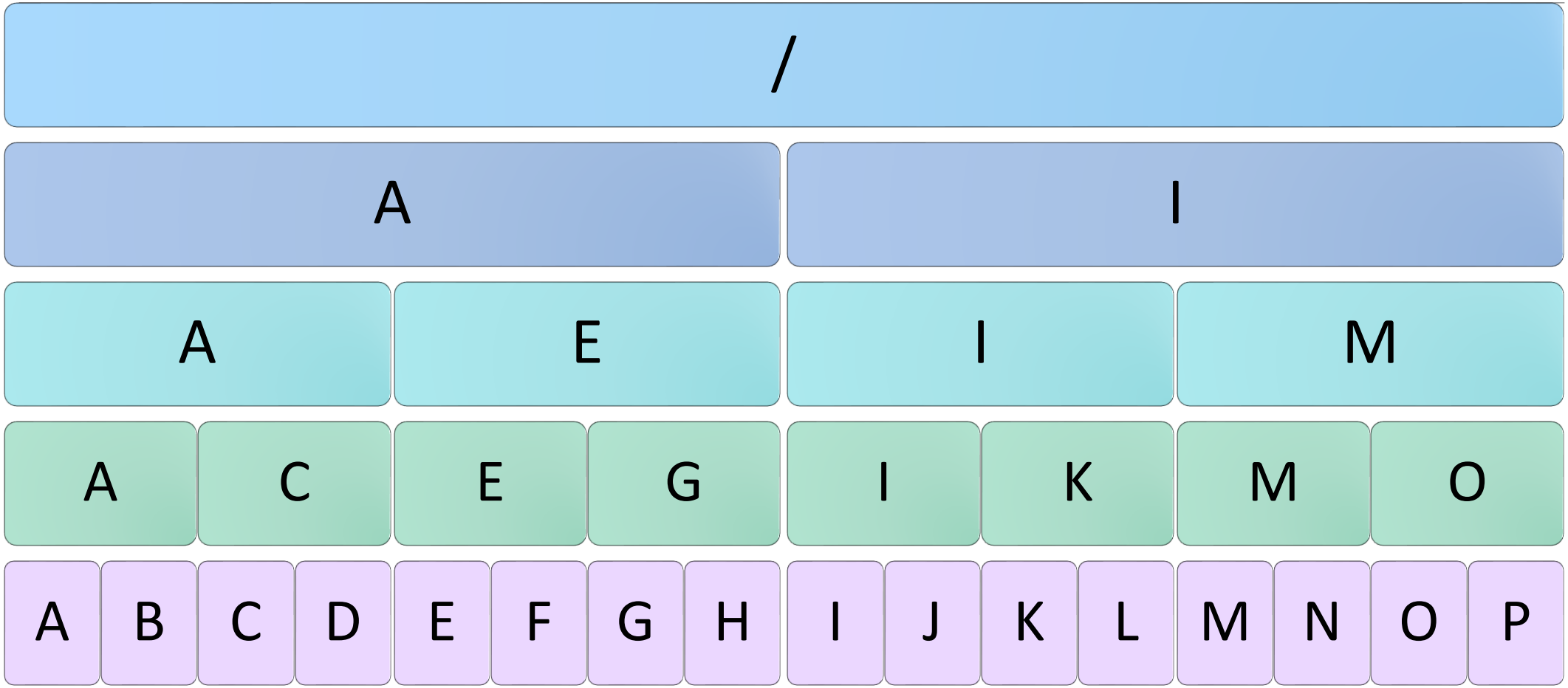


[Amazon AWS logo]

Balanced Work [Example adapted from CS 5412 slides]



Good Representatives



But what about larger less-exact computations?

What if we want a more complicated computation but are okay with an approximate answer?

What if we want to know the probability of a system reaching a certain state?

How does probabilistic analysis scale?





2. Bayesian Inference

GUILLAUME CLARET, SRIRAM RAJAMANI, ADITYA NORI,

ANDREW GORDON, JOHANNES BORGSTRÖM

What is Bayesian inference?

Suppose we have evidence *E* and want to figure out how likely a hypothesis *H* is based on seeing *E*.

Bayesian Inference: a method of figuring out what a *posterior* probability $P(H|E)$ is given

- prior probability $P(H)$
- likelihood function $P(E|H)$ / $P(E)$

Bayes' Rule:

$$P(H|E) = \frac{P(E|H)P(H)}{P(E)}$$

What is probabilistic programming?

```
float skillA, skillB,skillC;
float perfA1,perfB1,perfB2,
      perfC2,perfA3,perfC3;
skillA := Gaussian(100,10);
skillB := Gaussian(100,10);
skillC := Gaussian(100,10);

// first game:A vs B, A won
perfA1 := Gaussian(skillA,15);
perfB1 := Gaussian(skillB,15);
observe(perfA1 > perfB1);
// second game:B vs C, B won
perfB2 := Gaussian(skillB,15);
perfC2 := Gaussian(skillC,15);
observe(perfB2 > perfC2);

// third game:A vs C, A won
perfA3 := Gaussian(skillA,15);
perfC3 := Gaussian(skillC,15);
observe(perfA3> perfC3);
```

Programming, but with primitives for sampling and conditioning probability distributions

E.g. computing Xbox TrueSkill

How can we infer from probabilistic programs?

Few variables: we can use *data flow analysis* to symbolically solve for posterior distributions

- Uses *Algebraic Decision Diagrams* (ADDs): DAGs describing probabilities of outcomes

Lots of variables: the same, but with *batching* (transfers from joint ADDs to marginal ADDs):

$$p(x_1, x_2, \dots, x_n) \rightarrow p_1(x_1)p_2(x_2) \dots p_n(x_n)$$

…but are we talking about a PLs and data mining paper?

Inferring probabilistic outcomes **with a distributed system** can enable more complicated machine learning and data mining algorithms

Inferring probabilistic outcomes **about a distributed system** can be useful for monitoring and load distribution

Examples: a power grid with a chance of failure, driving in New York City, storing files in s3, sharding data in a search engine

Driving

If you're driving in NYC:

- You drive at the speed of traffic (stochastic average)
- You observe the cars ahead of you and react to them
- You expect the cars behind you to observe you and react to you
- You plan for the possibility of more common "bad" behaviors

[Source: picphotos.net; example stolen from Ken]

Amazon S3

Clients can store files, modify metadata, and delete files

We need to find a node with space for new files

Lots of transactions are happening at the same time

[Amazon AWS logo]

How do we distribute storage requests?

- Hash-based: expected to be evenly distributed, but maybe not
- Pick the least full: everyone will flock to the same node at once
- Probabilistically weight nodes based on observed space free? Maybe, but we don't have great strategies to do that yet.

Yelp's structure

Broken up into geographic shards,

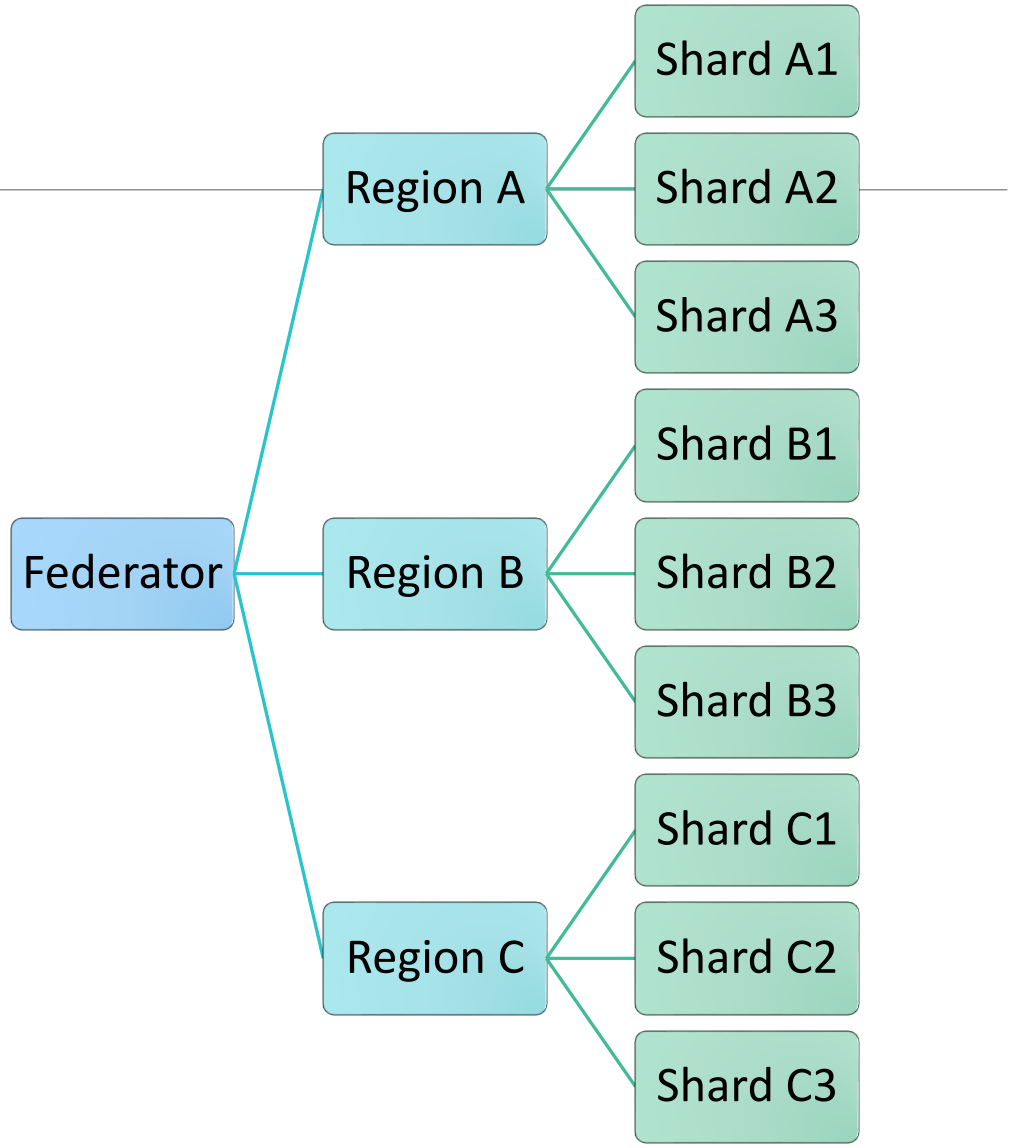
which are then broken into random shards,

which each have several replicas,

which need to be able to handle

- Searches
- New businesses
- Business updates

How do we distribute load?



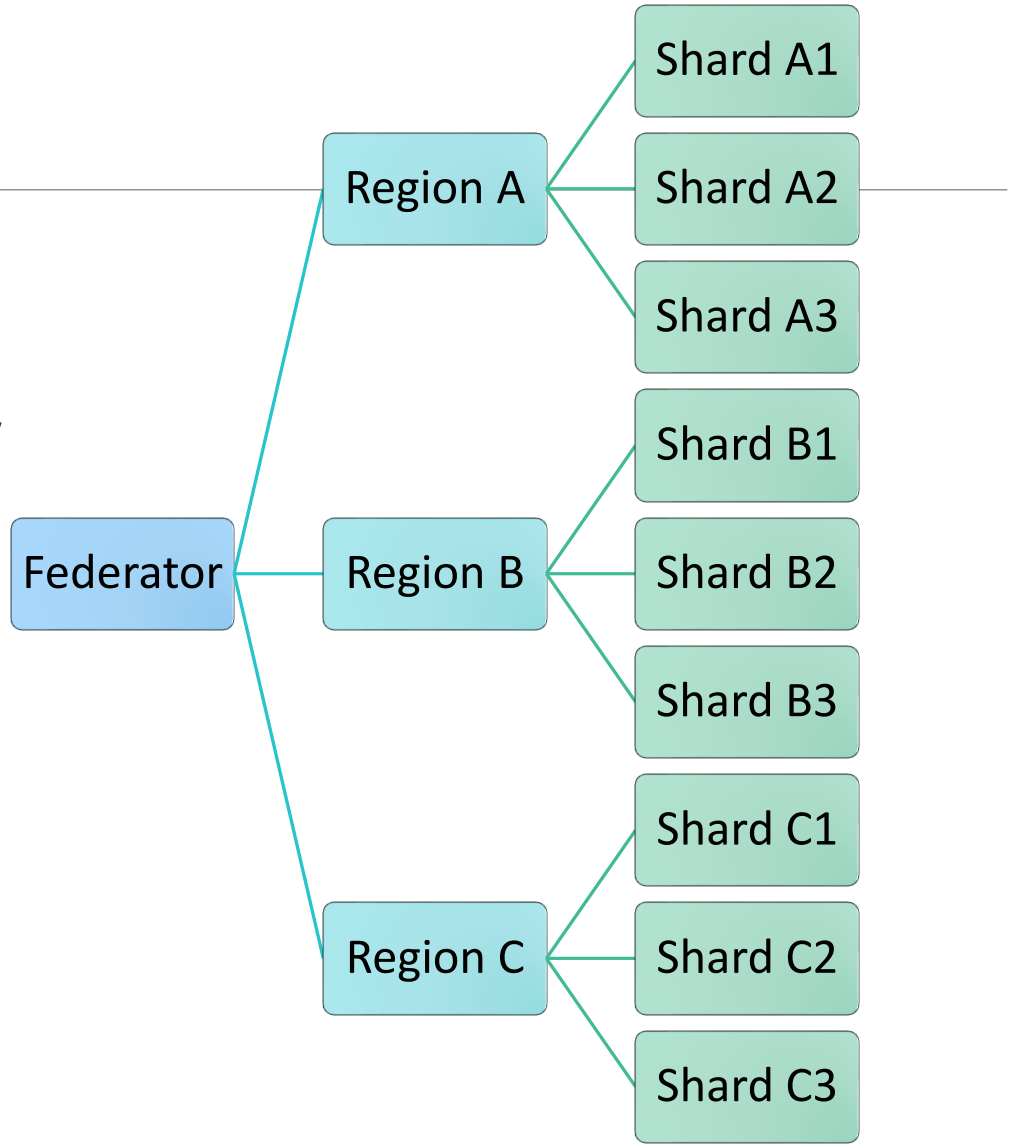
Yelp's structure

We can observe *priors* about request load in different shards

We would then estimate *probability distributions* for different levels of load

We could use that to reason about

- Where to put new businesses
- Where to direct queries
- Whether a different sharding strategy would work better



Questions

How do we best leverage different types of protocols to build good systems?

Is gossip good enough?

What large-scale distributed systems ideas could help data mining researchers?

What data mining ideas could help distributed systems researchers?