# Monitoring & Management

# (Distributed Query Processing)

PLANETLAB

# Internet-Scale Querying

- Large scale
  - thousands to millions of nodes
- Imprecise results
  - non-serializable
  - latency
  - packet loss
  - node / link failure
- No single schema
  - system evolves in decentralized manner
  - multiple sources in multiple domains



PLANETLAB

# Internet Queries (cont)

- Highly dynamic queries
  - non-static, change over time
  - large number of users
  - spatial parallelism
- Query optimization highly data-dependent
  - can be optimized at query-time

# Applications

- *Large system management*
- Network measurement
- Enhanced file sharing services
- Planetary-scale sensor networks
- Mobility services
- Etc.

# Systems Management

- Data Sources / Sensors
- ***Distributed Query Processing***
- Distributed Logging
- Data Visualization
- Machine Learning
- Remote Actuation

# Models

- Hierarchical: IrisNet, Astrolabe
  - establish single hierarchy in advance (XML)
- Inference-based: Sophia
  - completely freeform (Prolog)
- Relational: PIER
  - schema (sort of), late-bound query plan (SQL)
- Others?

# Managing PlanetLab

## Today

- Observe
  - central pull of data every 5 minutes
- Analysis
  - post-processing (lots of pearl scripts)
  - human in the loop
- React
  - email & rsh

# Managing PlanetLab

## Sophia

- Observe
  - sensors produce facts
  - facts move to where needed
- Analysis
  - Prolog rules
- React
  - actuators

# Sensors

- Organized into Sensor Servers
  - kernel stats
  - network probes
  - software configurations
  - service-specific
- Access locally
  - via HTTP
- Two types
  - snapshot
  - streaming

# Sophia Expressions

- Evaluate an expression at some point in time/space

  eval(when, where, exp)

  - when: future, past, now, last, event
  - where: specific node
- Unification
  - find a fact that makes the expression true

    eval(time(now), node(42), load(L))

  - evaluation results asserted in *fact database*
    - time/place-stamped

# Simple Examples

eval(time(now), node(42), (load(L), L<0.7))
true(time(1049246673), node(42), (load(0.5), 0.5<0.7)

eval(time(now), node(42), (load(L), L<0.4))
false(time(1049246673), node(42), (load(L), L<0.4)

eval(time(now), node(42), (load(L), L<0.7))
maybe(time(1049246673), node(42), (load(L), L<0.7)

eval(time(now), node(42), (load(L), L>10), react(…))

# Examples (cont)

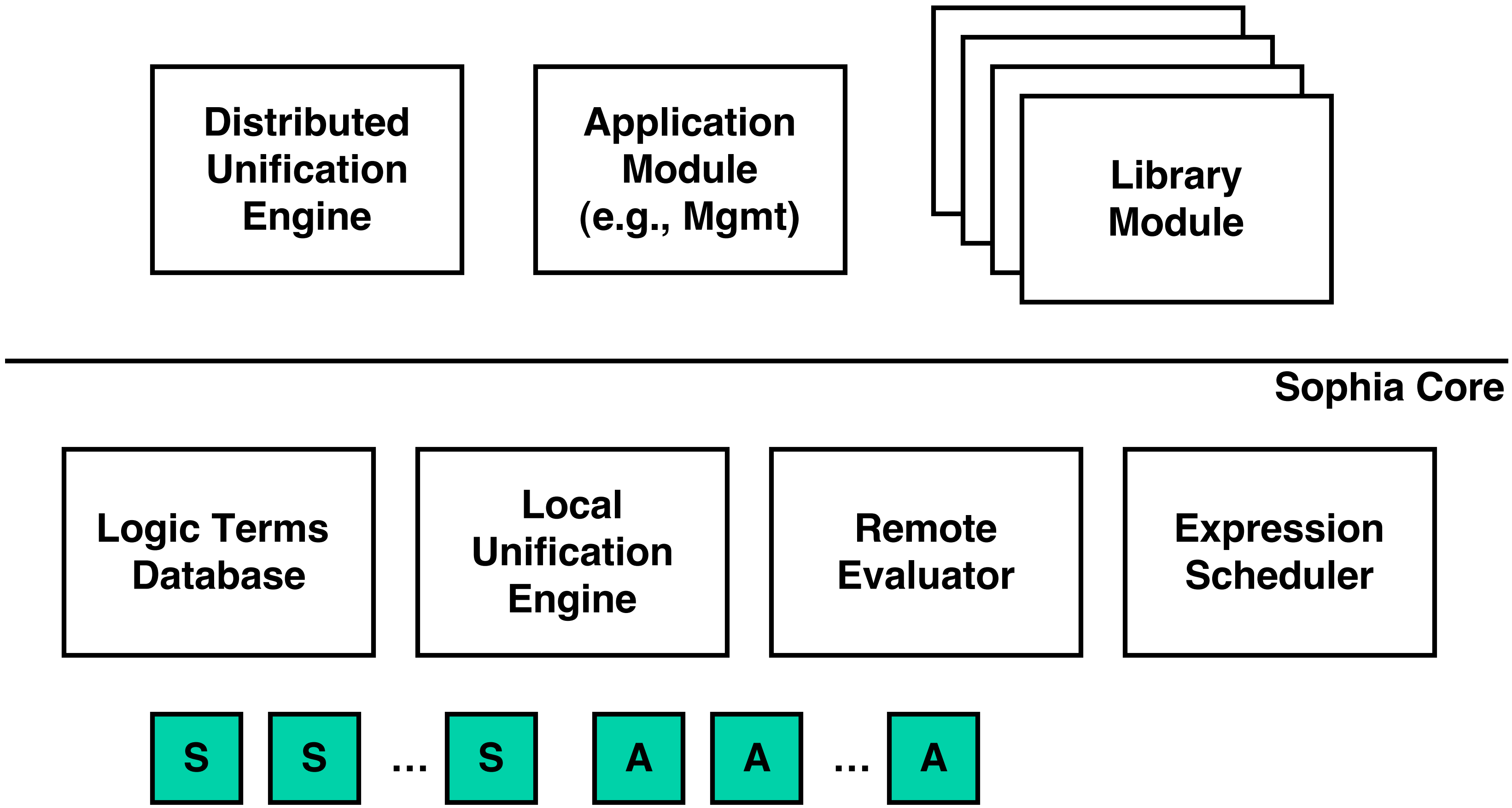
```
eval(time(now), bagof([L, N],
  (node(N),
   eval(time(now), node(N), (load(L), L<0.7))),
   Vs)
).

true(time(1049246673),…
[[37, 0.5], [42, 0.4], [55, 0.6]]))).
```



PLANETLAB

# Components



Distributed Unification Engine
Application Module (e.g., Mgmt)
Library Module
Sophia Core
Logic Terms Database
Local Unification Engine
Remote Evaluator
Expression Scheduler
S
S
...
S
A
A
...
A

# Design Issues

- Performance
  - expression parallelization
  - caching (logging)
  - Scheduling (pre-fetching)
  - query planning
    - query rewriting
    - introspection (rewrite on the fly)
- Failures
  - accommodate *holes*

# Issues (cont)

- Extensibility
  - privilege → capabilities

    cap3527519(Val) :- read_bw_sensor.

    Bandwidth(BwVal) :- cap3527519(BwVal).
  - module composition
    - protect private modules with capabilities
    - publish public module names

# Advantages

- Declarative language
  - natural way to express desired properties/behavior
  - permits efficient implementation
  - decouples *what* from *how*
- Easy to extend at runtime
  - supports evolving management tools
  - promise of introspection
- Explicitly exposes…
  - where → transparently distribute expressions
  - when → both past (logging) and future (events)



PLANETLAB

# PIER

- Relational model / query language
  - actually, a *query plan* right now
- Use a DHT substrate
  - rehashing
  - rendezvous
  - multicast
  - aggregation

# Relational Queries

- Data is tuples in named tables
  - tables exist on nodes
- Relational operators:
  - selection
  - projection
  - join:
    - correlate, intersect, match
  - aggregation:
    - summarize, compress

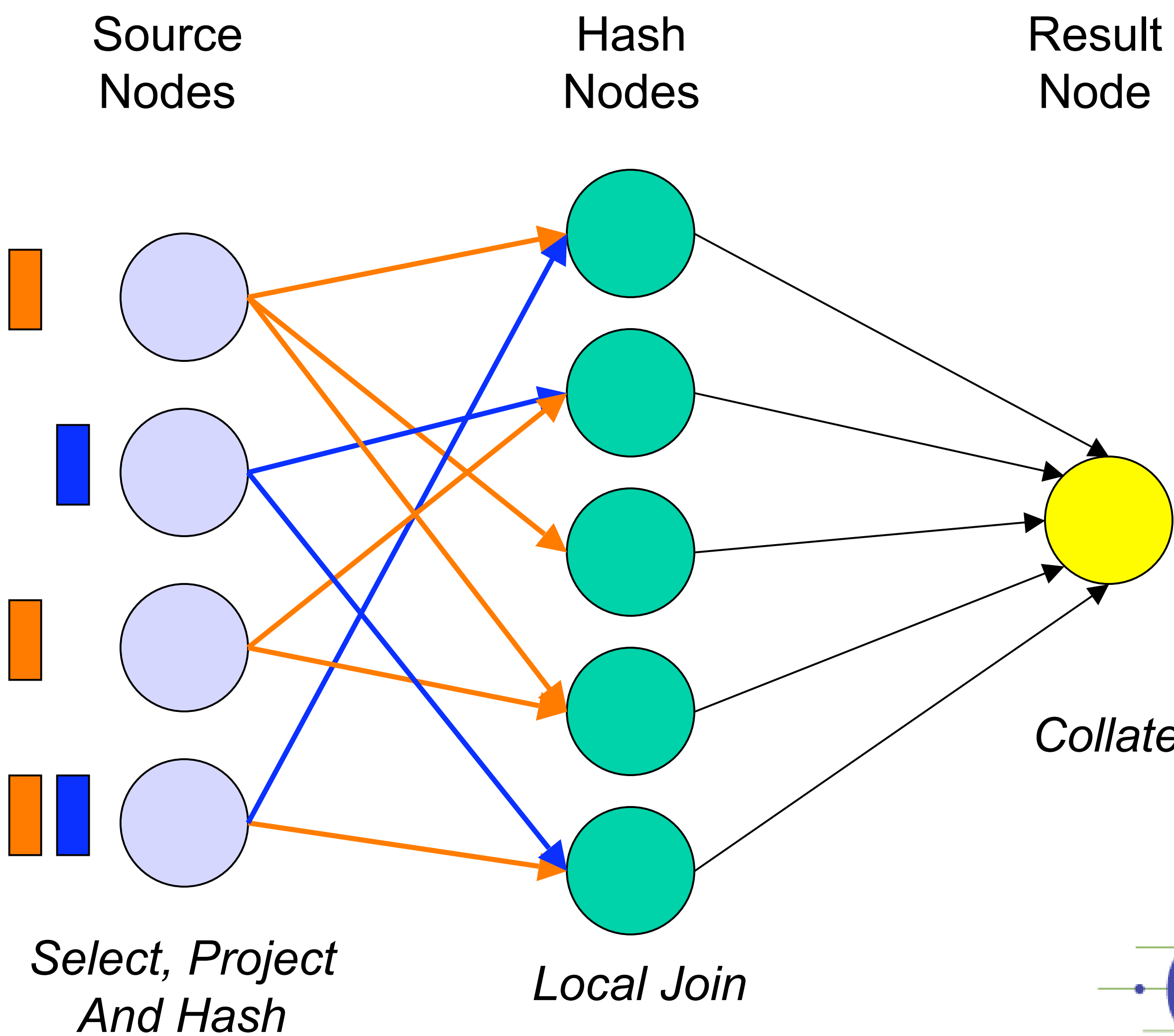
# Symmetric Hash Join

- Goal: get tuples with same join key to same node
- Each node:
  - scans data for join candidates
  - stores tuple in DHT with hash(join key)
- Hash nodes send matches data to query origin



PLANETLAB

# Symmetric Hash Join






PLANETLAB