# The Design of the Borealis Stream Processing Engine

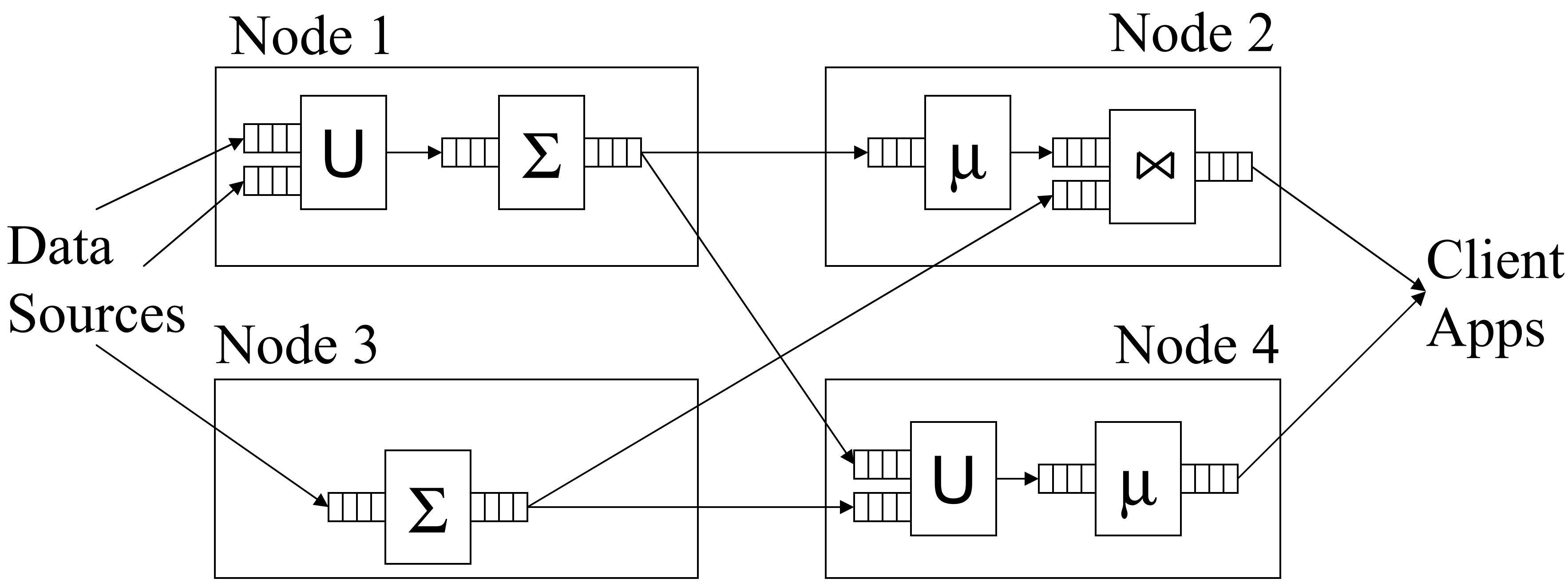

Brandeis University, Brown University, MIT

Magdalena Balazinska
*MIT*

Nesime Tatbul
*Brown*

# Distributed Stream Processing

# Distributed Stream Processing




- Data sources push tuples continuously
- Operators process windows of tuples

# Where are we today?

- Data models, operators, query languages
- Efficient single-site processing
- Single-site resource management
- [STREAM, TelegraphCQ, NiagaraCQ, Gigascope, Aurora]
- Basic distributed systems
  - Servers [TelegraphCQ, Medusa/Aurora]
  - or sensor networks [TinyDB, Cougar]
  - Tolerance to node failures
  - Simple load management

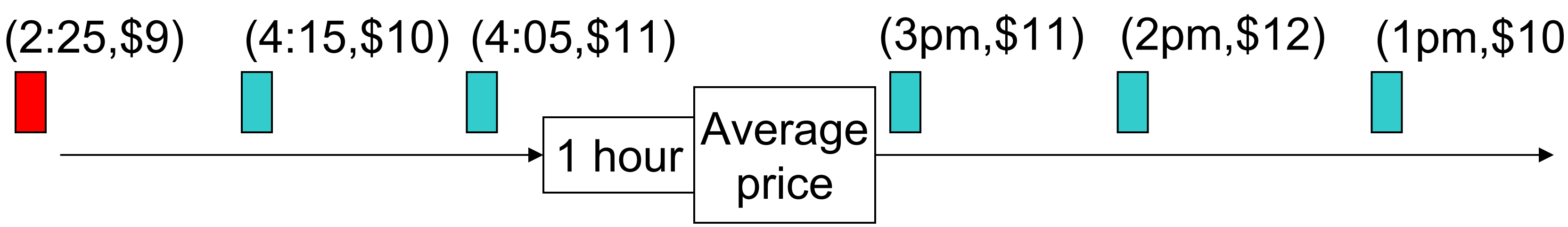
# Challenges

- Tuple revisions
  - Revisions on input streams
  - Fault-tolerance
- Dynamic & distributed query optimization
- ...

# Causes for Tuple Revisions

- Data sources revise input streams:

  "On occasion, data feeds put out a faulty price [...] and send a correction within a few hours" [MarketBrowser]
- Temporary overloads cause tuple drops
- Temporary failures cause tuple drops



# Current Data Model

header data

```
(time,a1,...,an)
```

- `time`: tuple timestamp

# New Data Model for Revisions

header / data

`(time,type,id,a1,...,an)`

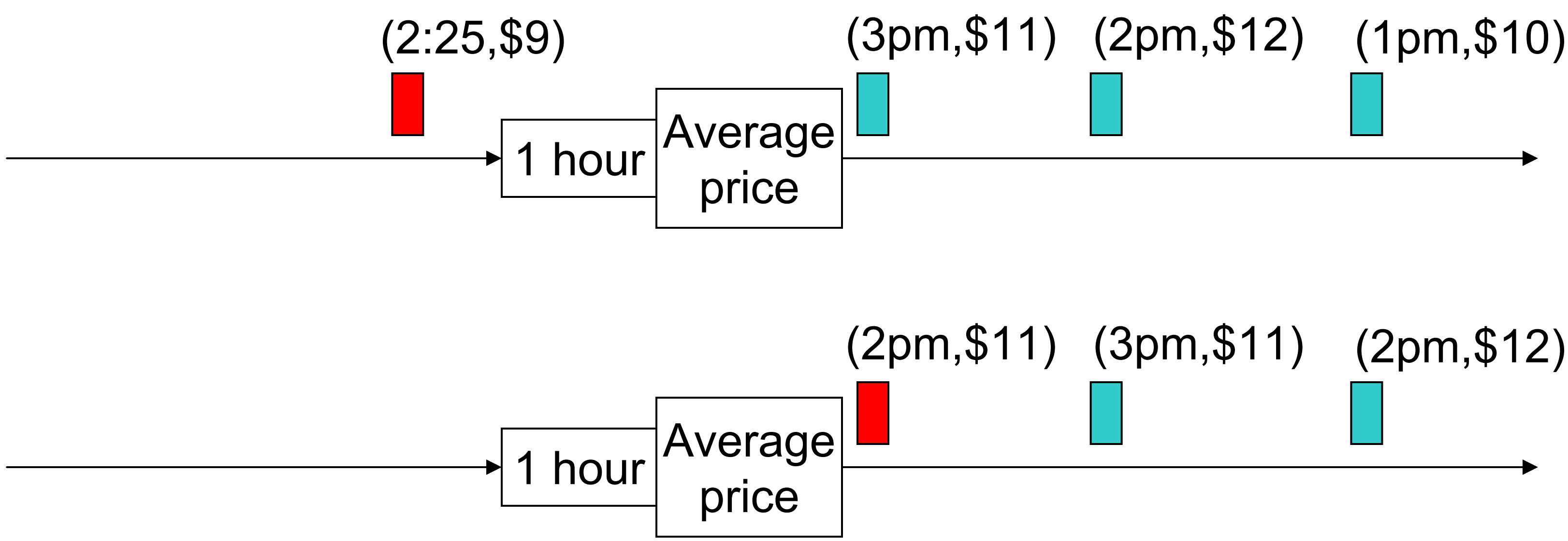
- `time`: tuple timestamp
- **`type`**: tuple type
  - insertion, deletion, replacement
- **`id`**: unique identifier of tuple on stream

# Revisions: Design Options

- Restart query network from a checkpoint
- Let operators deal with revisions
  - Operators must keep their own history
  - (Some) streams can keep history

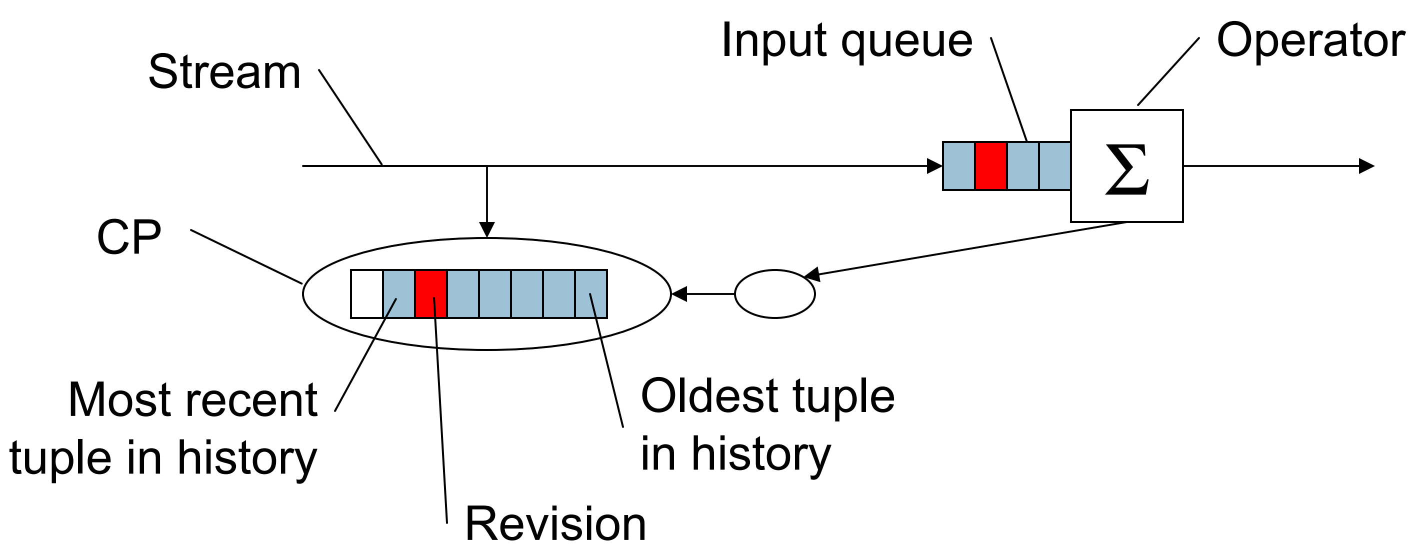
# Revision Processing in Borealis

- Closed model: revisions produce revisions



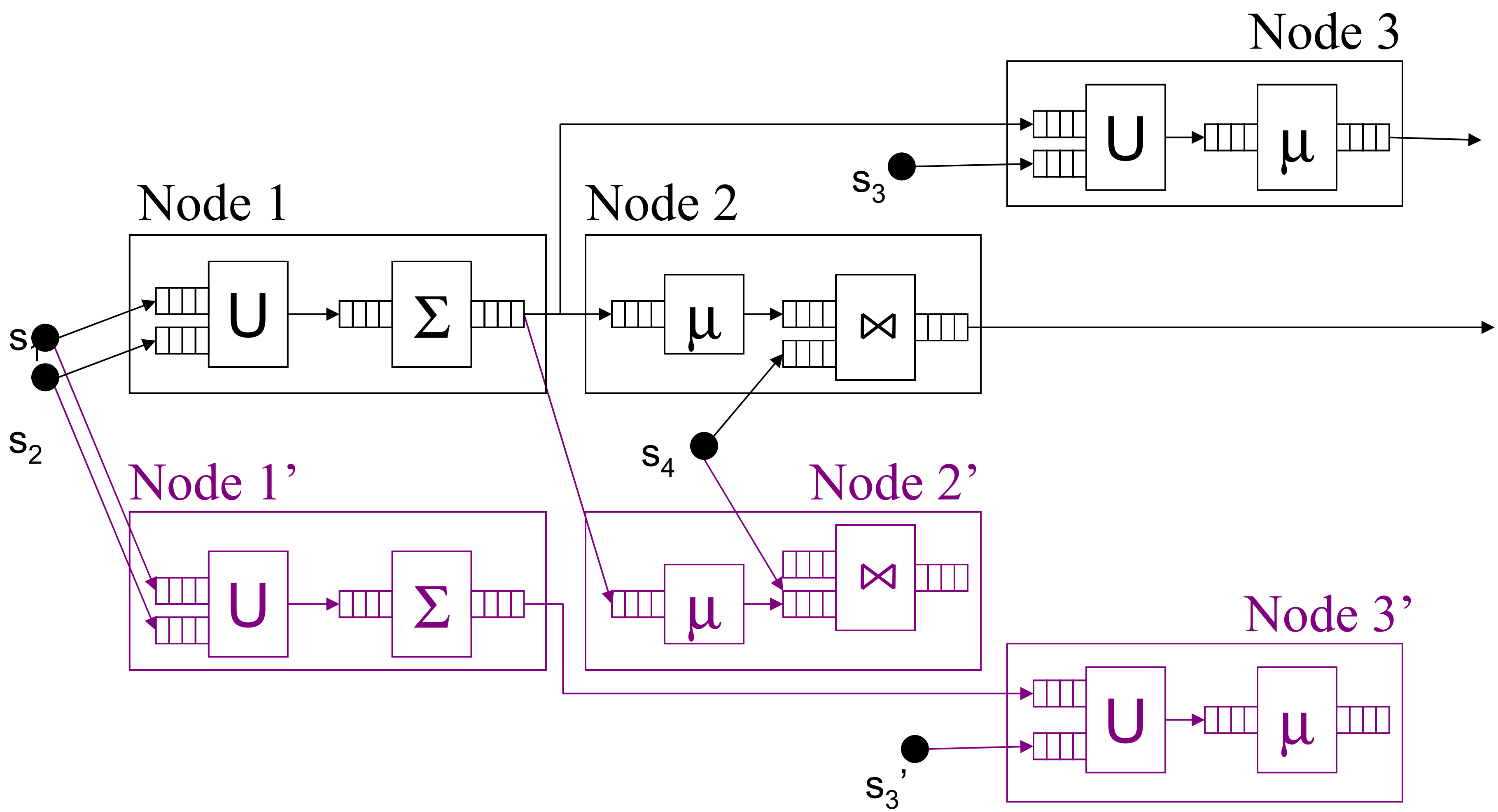
# Revision Processing in Borealis

- Connection points (CPs) store history
- Operators pull the history they need



# Fault-Tolerance through Replication

- Goal: Tolerate node and network failures

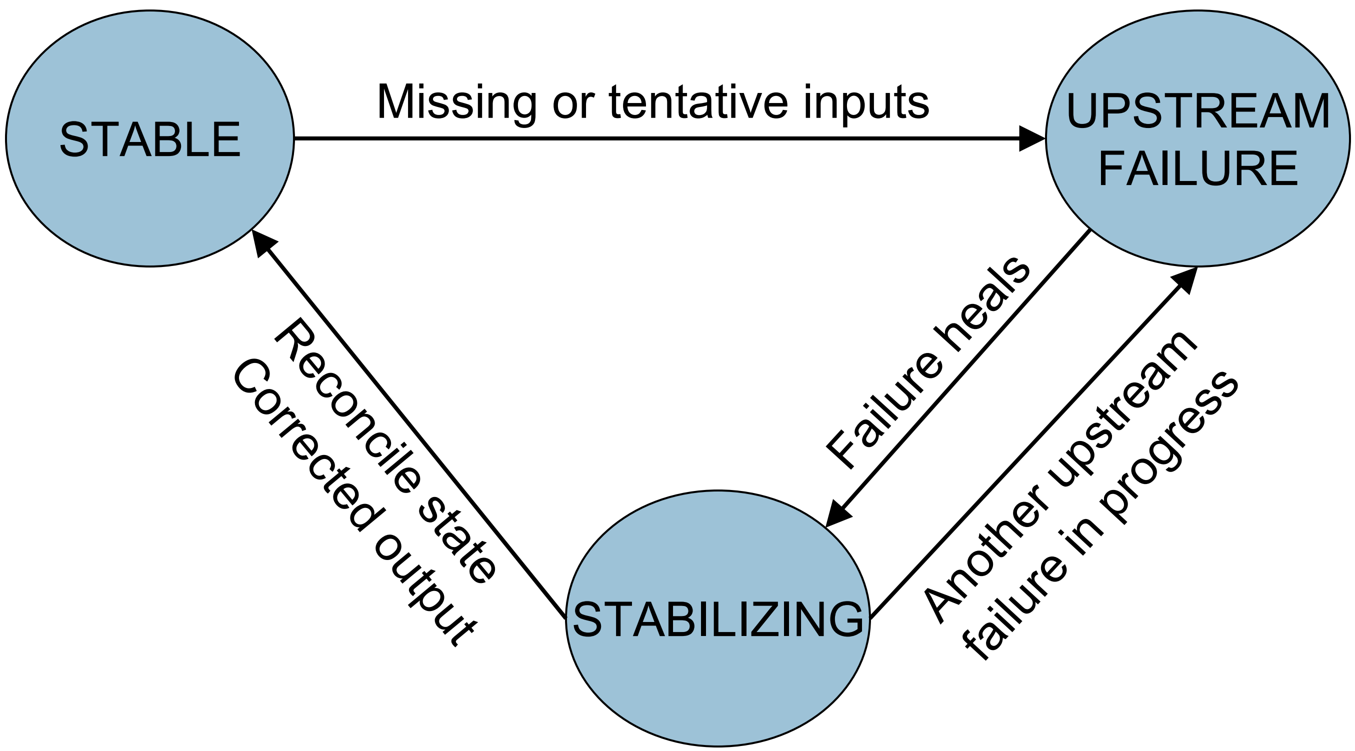


# Reconciliation

- State reconciliation alternatives
  - Propagate tuples as revisions
  - Restart query network from a checkpoint
  - Propagate UNDO tuple
- Output stream revision alternatives
  - Correct individual tuples
  - Stream of deletions followed by insertions
  - Single UNDO tuple followed by insertions

# Fault-Tolerance Approach

- If an input stream fails, find another replica
- No replica available, produce tentative tuples
- Correct tentative results after failures



# Challenges

- Tuple revisions
  - Revisions on input streams
  - Fault-tolerance
- **Dynamic & distributed query optimization**
- ...

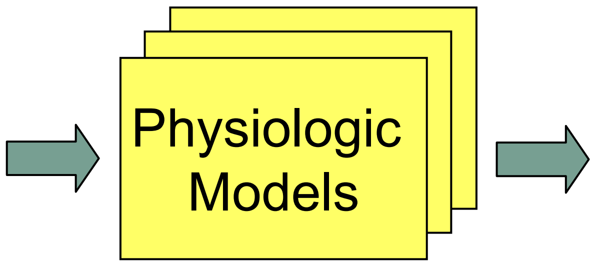
# Optimization in a Distributed SPE

- Goal: Optimized resource allocation
- Challenges:
  - Wide variation in resources
    - High-end servers vs. tiny sensors
  - Multiple resources involved
    - CPU, memory, I/O, bandwidth, power
  - Dynamic environment
    - Changing input load and resource availability
  - Scalability
    - Query network size, number of nodes

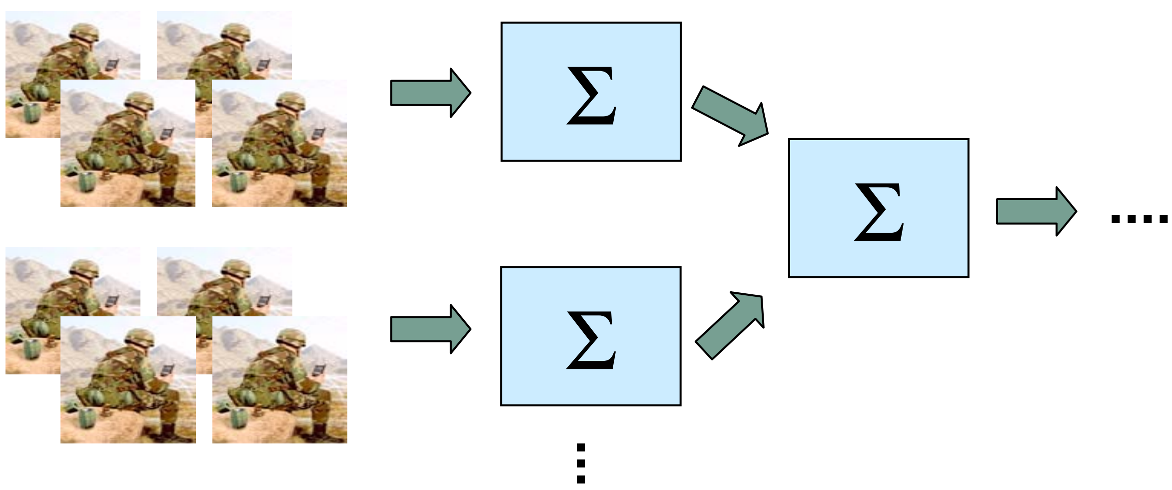
# Quality of Service

- A mechanism to drive resource allocation
- Aurora model
  - QoS functions at query end-points
  - Problem: need to infer QoS at upstream nodes
- An alternative model
  - Vector of Metrics (VM) carried in tuples
  - Operators can change VM
  - A Score Function to rank tuples based on VM
  - Optimizers can keep and use statistics on VM

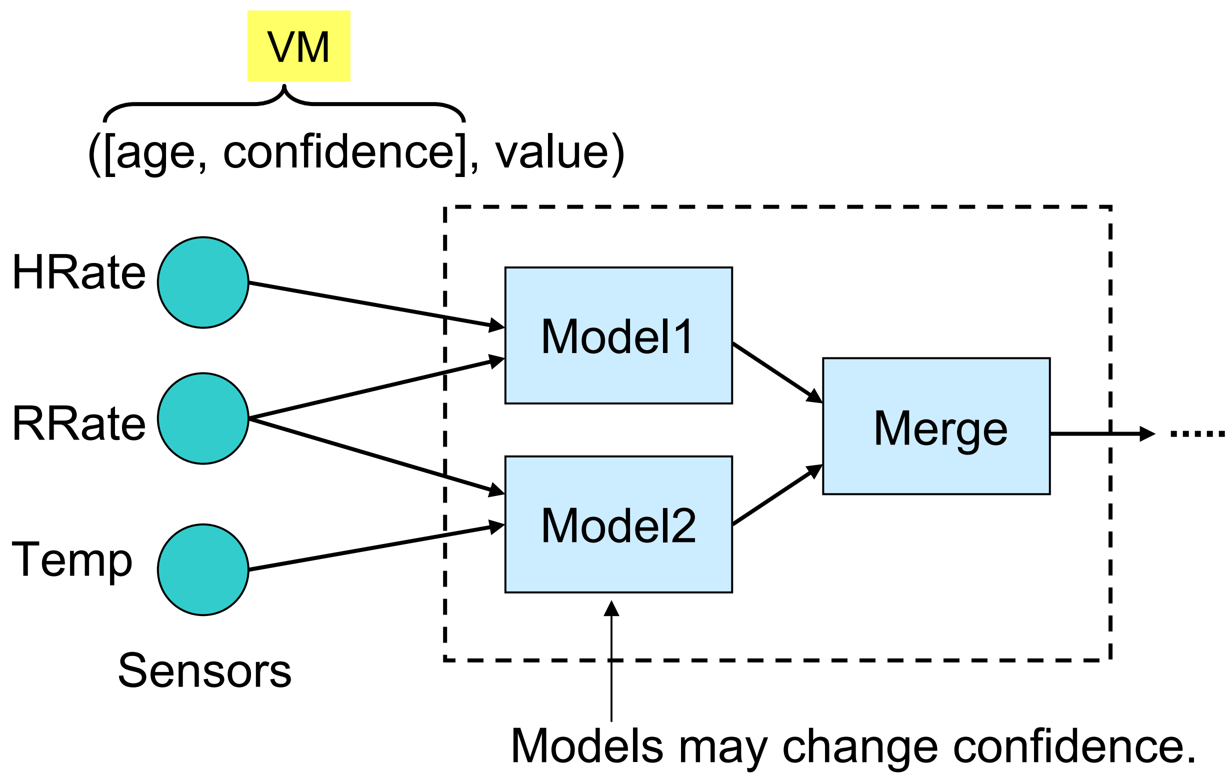
# Example Application:
## Warfighter Physiologic Status Monitoring (WPSM)




| Area | State | Confidence |
|---|---|---|
| Thermal | Red | 90% |
| Hydration | Yellow | 60% |
| Cognitive | Green | 100% |
| Life Signs | Green | 90% |
| Wound Detection | Yellow | 80% |



# Ranking Tuples in WPSM

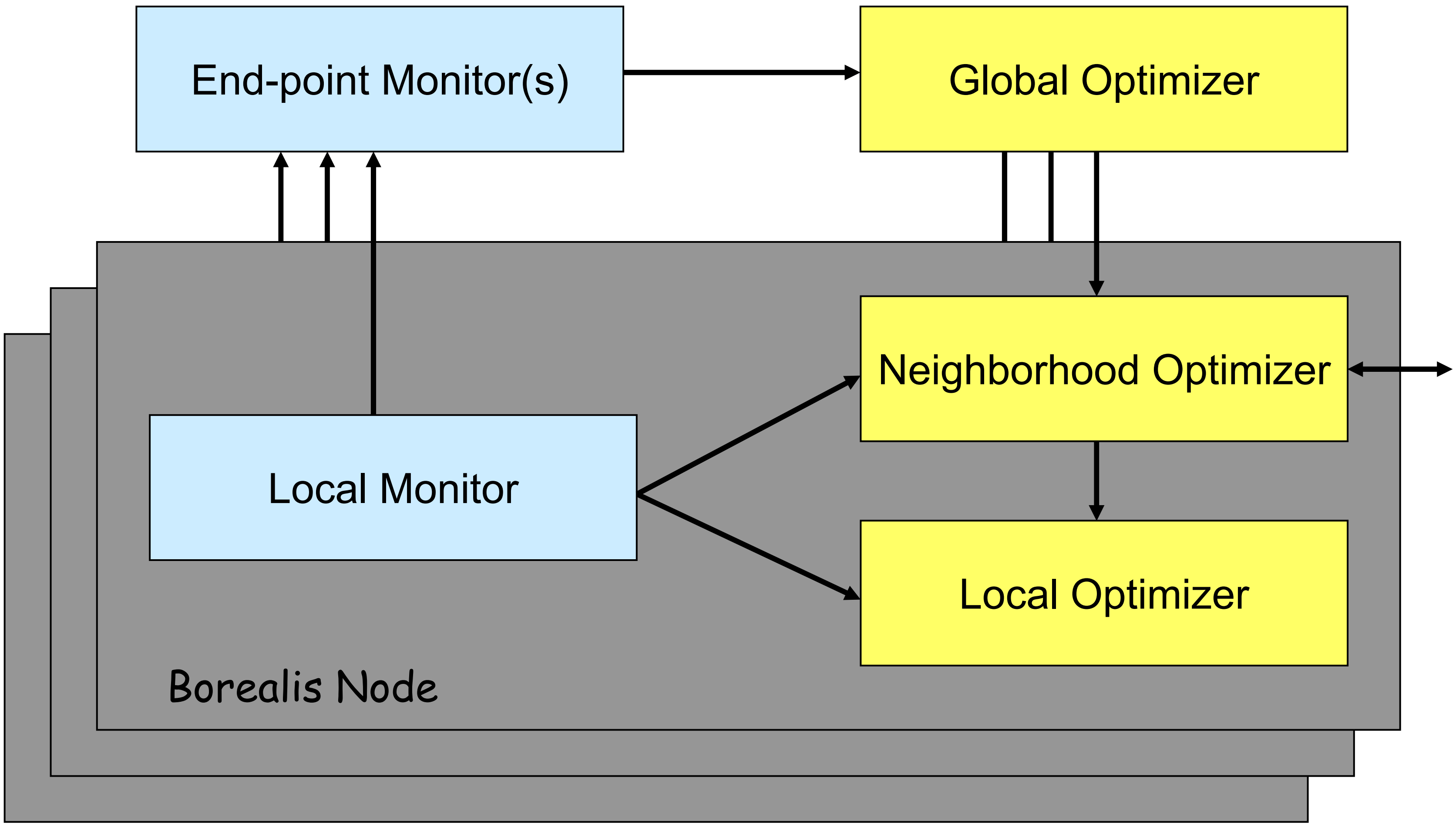



Score Function

SF(VM) = VM.confidence x ADF(VM.age)

↑ age decay function

# Borealis Optimizer Hierarchy

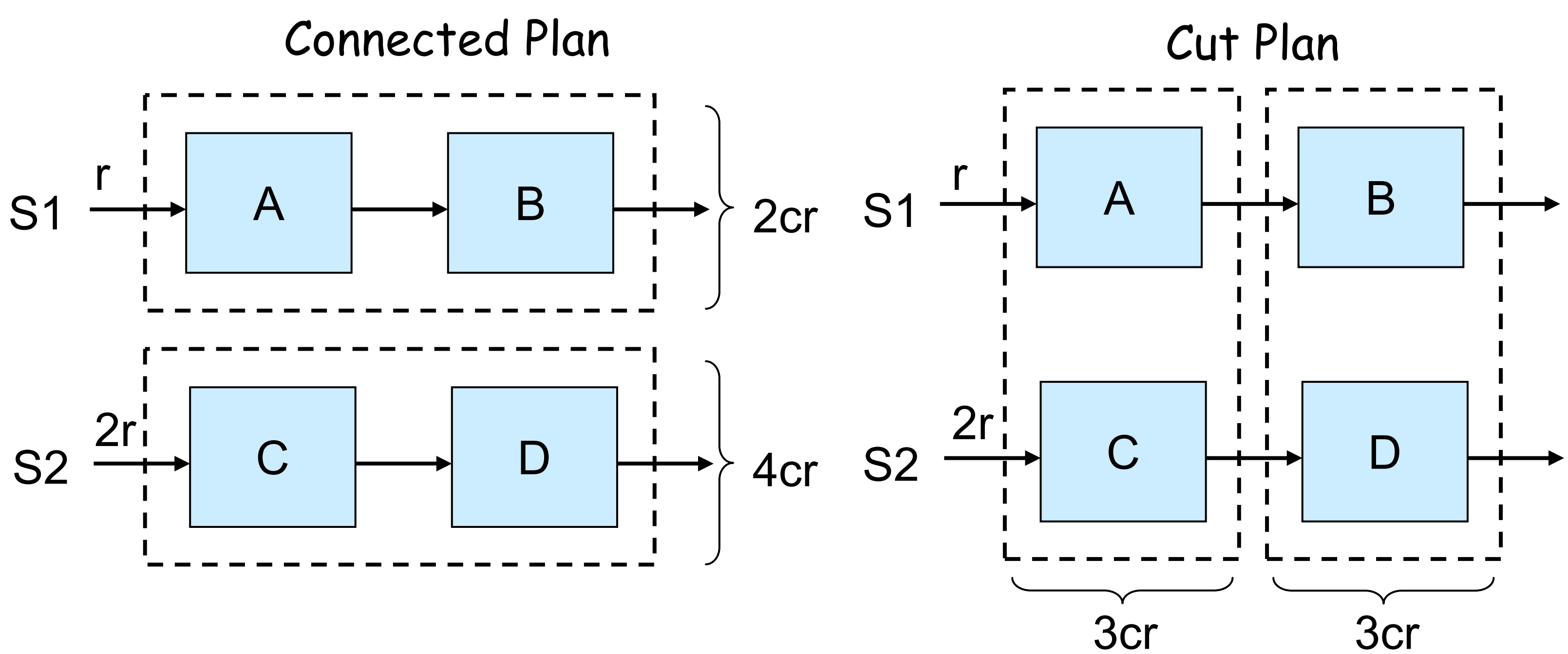


# Optimization Tactics

- Priority scheduling
- Modification of query plans
  - Commuting operators
  - Using alternate operator implementations
- Allocation of query fragments to nodes
- Load shedding

# Correlation-based Load Distribution

- Goal: Minimize end-to-end latency
- Key ideas:
  - Balance load across nodes to avoid overload
  - Group boxes with small load correlation together
  - Maximize load correlation among nodes

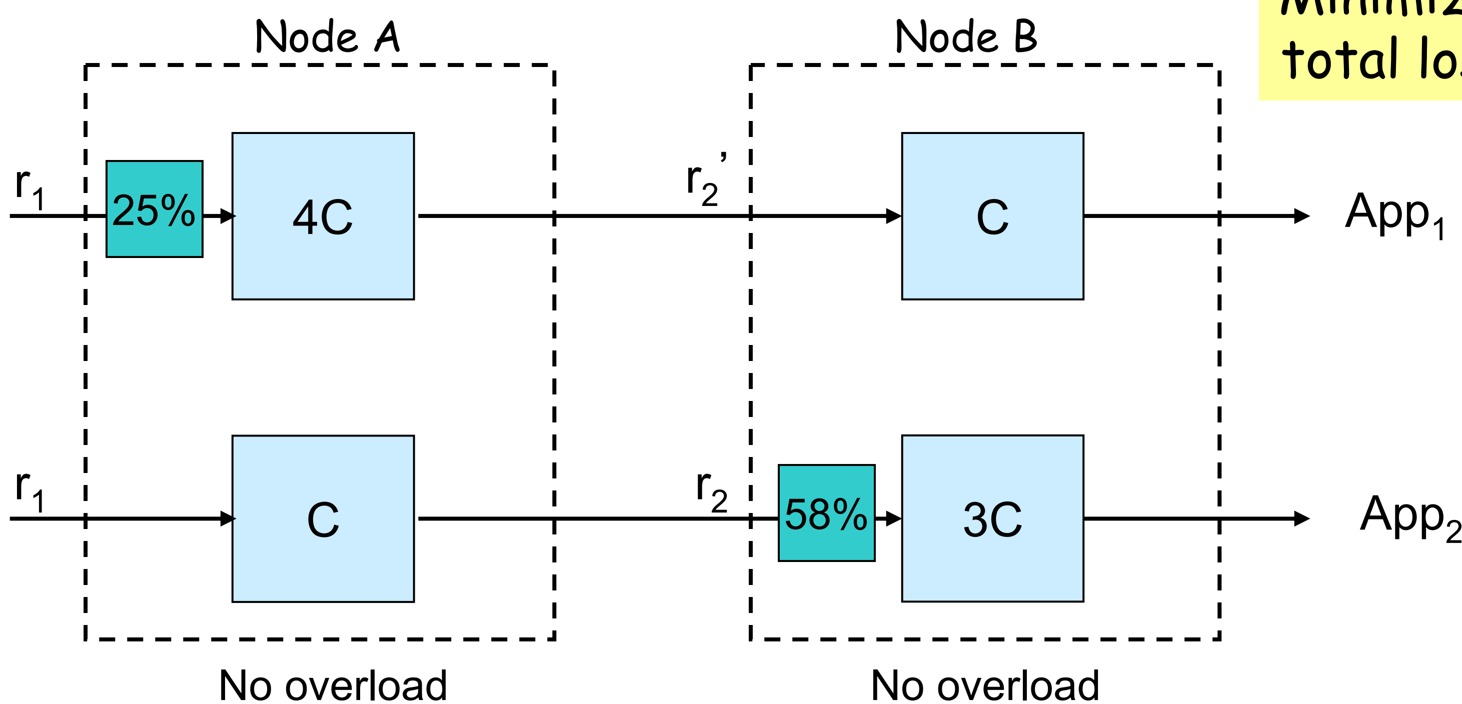


# Load Shedding

- Goal: Remove excess load at all nodes and links
- Shedding at node A relieves its descendants
- Distributed load shedding
  - Neighbors exchange load statistics
  - Parent nodes shed load on behalf of children
  - Uniform treatment of CPU and bandwidth problem
- Load balancing or Load shedding?

# Local Load Shedding

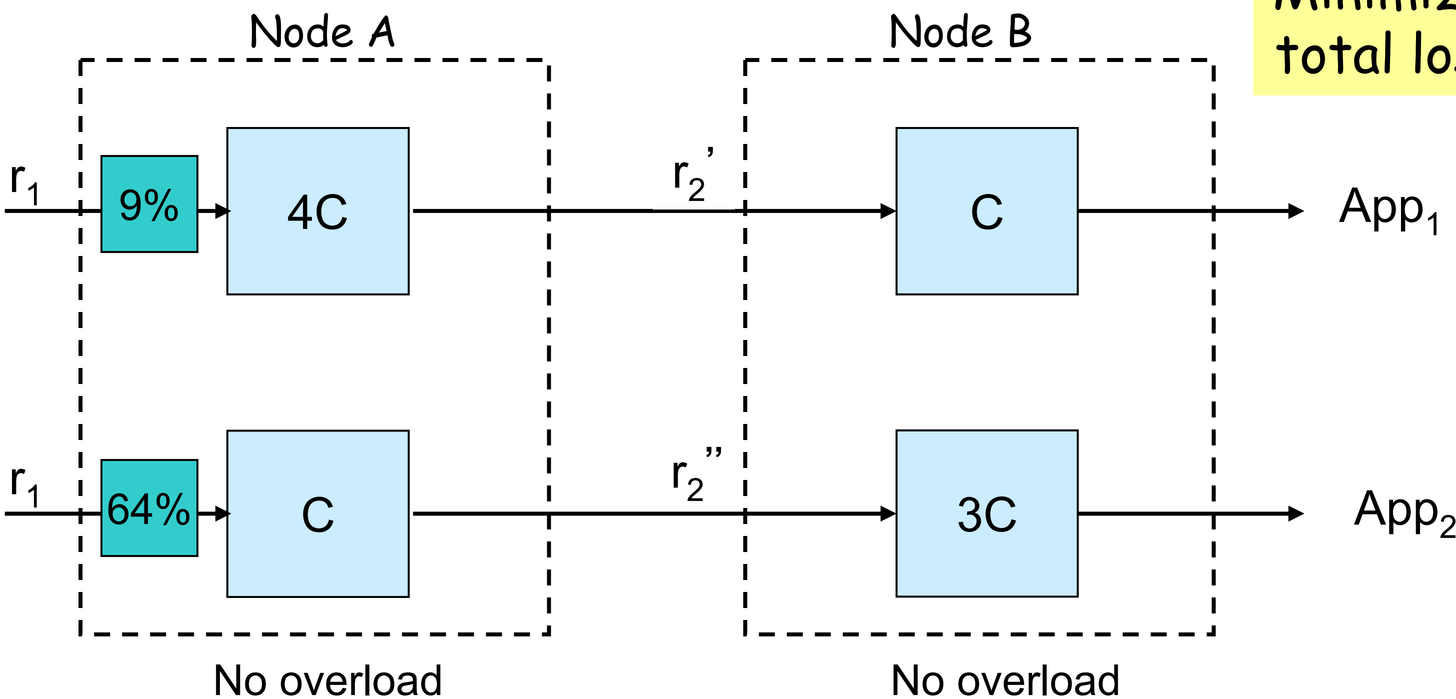
Goal: Minimize total loss




| *Plan* | *Loss* |
|---|---|
| **Local** | $App_1$: 25% $App_2$: 58% |

# Distributed Load Shedding

Goal: Minimize total loss

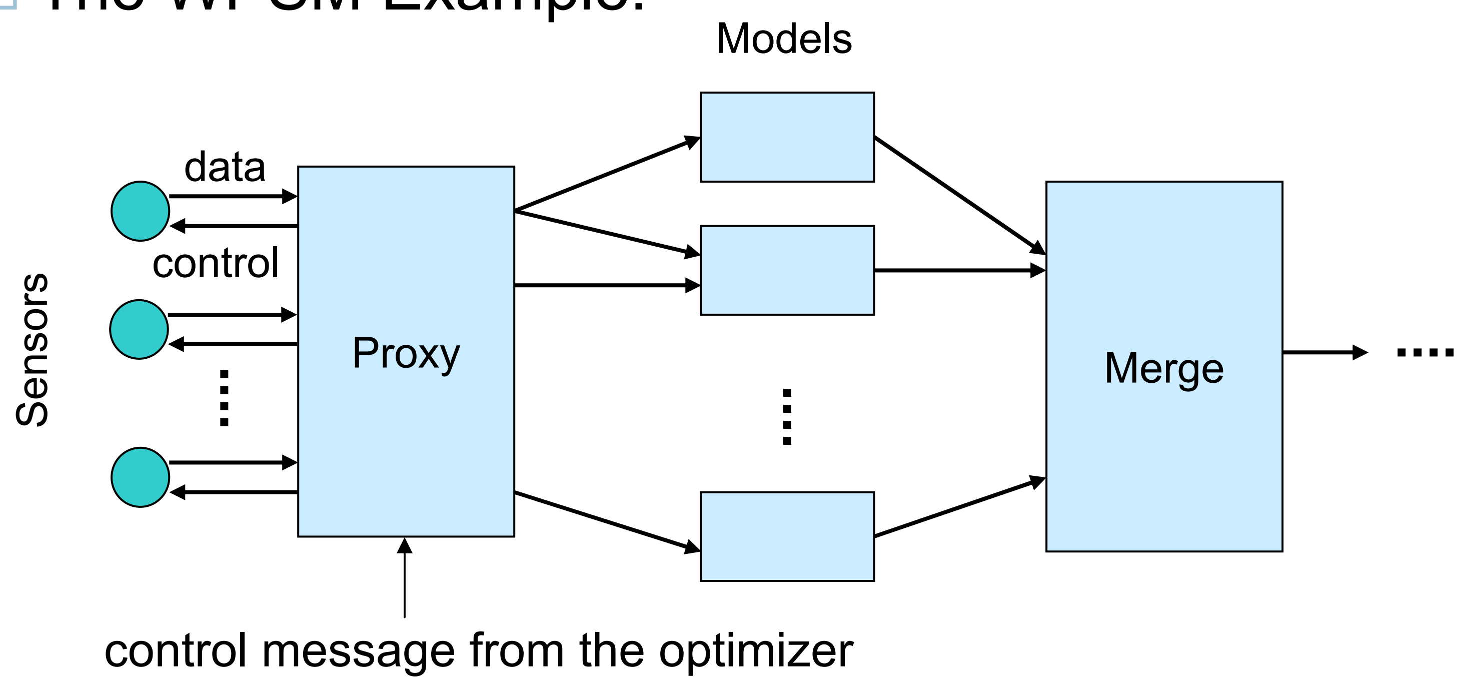



| *Plan* | *Loss* |
|---|---|
| **Local** | $App_1$: 25% $App_2$: 58% |
| **Distributed** | $App_1$: 9% $App_2$: 64% |

← smaller total loss!

# Extending Optimization to Sensor Nets

- Sensor proxy as an interface
- Moving operators in and out of the sensor net
- Adjusting sensor sampling rates
  - The WPSM Example:



# Conclusions

- Next generation streaming applications require a flexible processing model
  - Distributed operation
  - Dynamic result and query modification
  - Dynamic and scalable optimization
  - Server and sensor network integration
  - Tolerance to node and network failures
- Borealis has been iteratively designed, driven by real applications
- First prototype release planned for Spring'05
- http://nms.lcs.mit.edu/projects/borealis