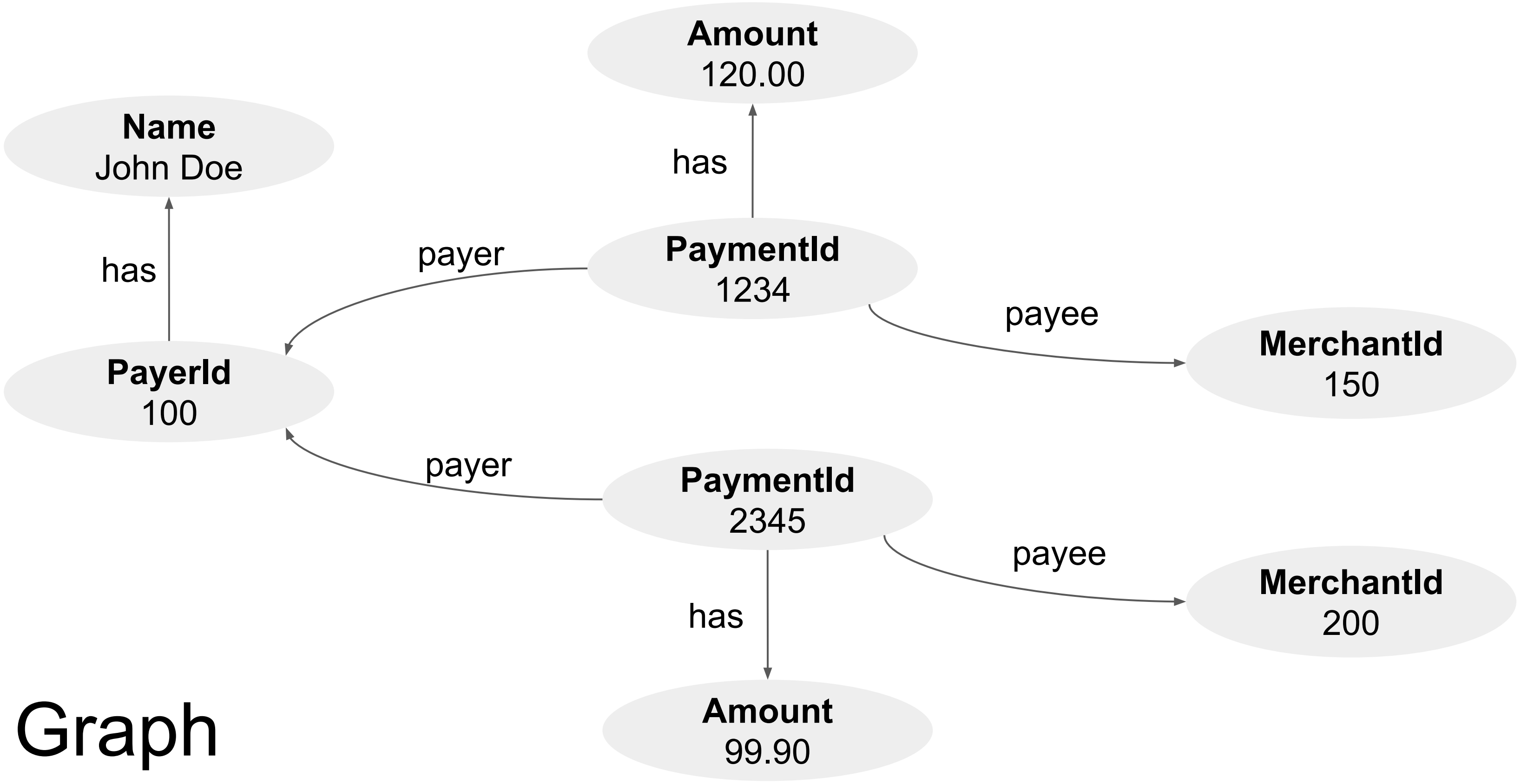
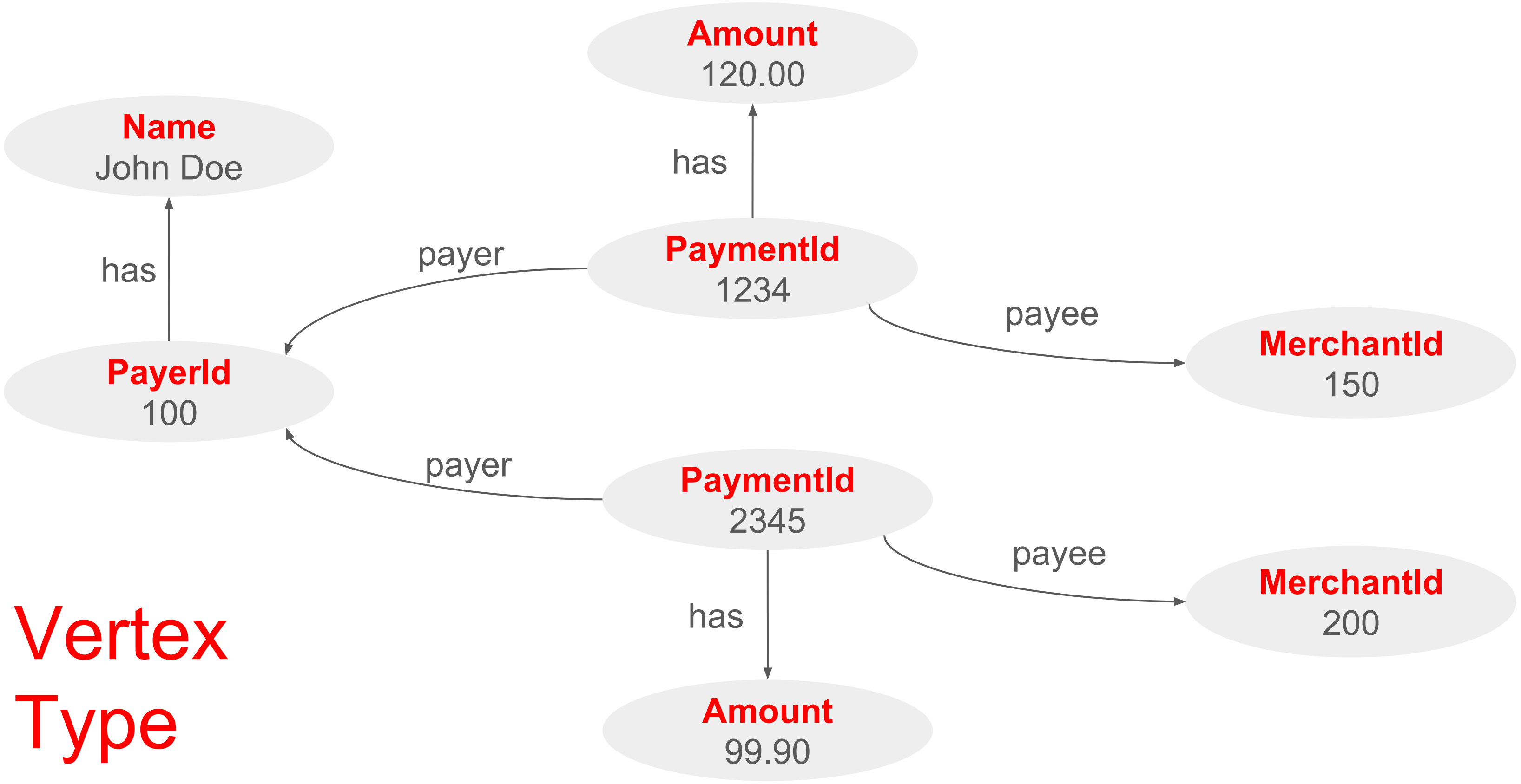
Cadenza

Cadenza

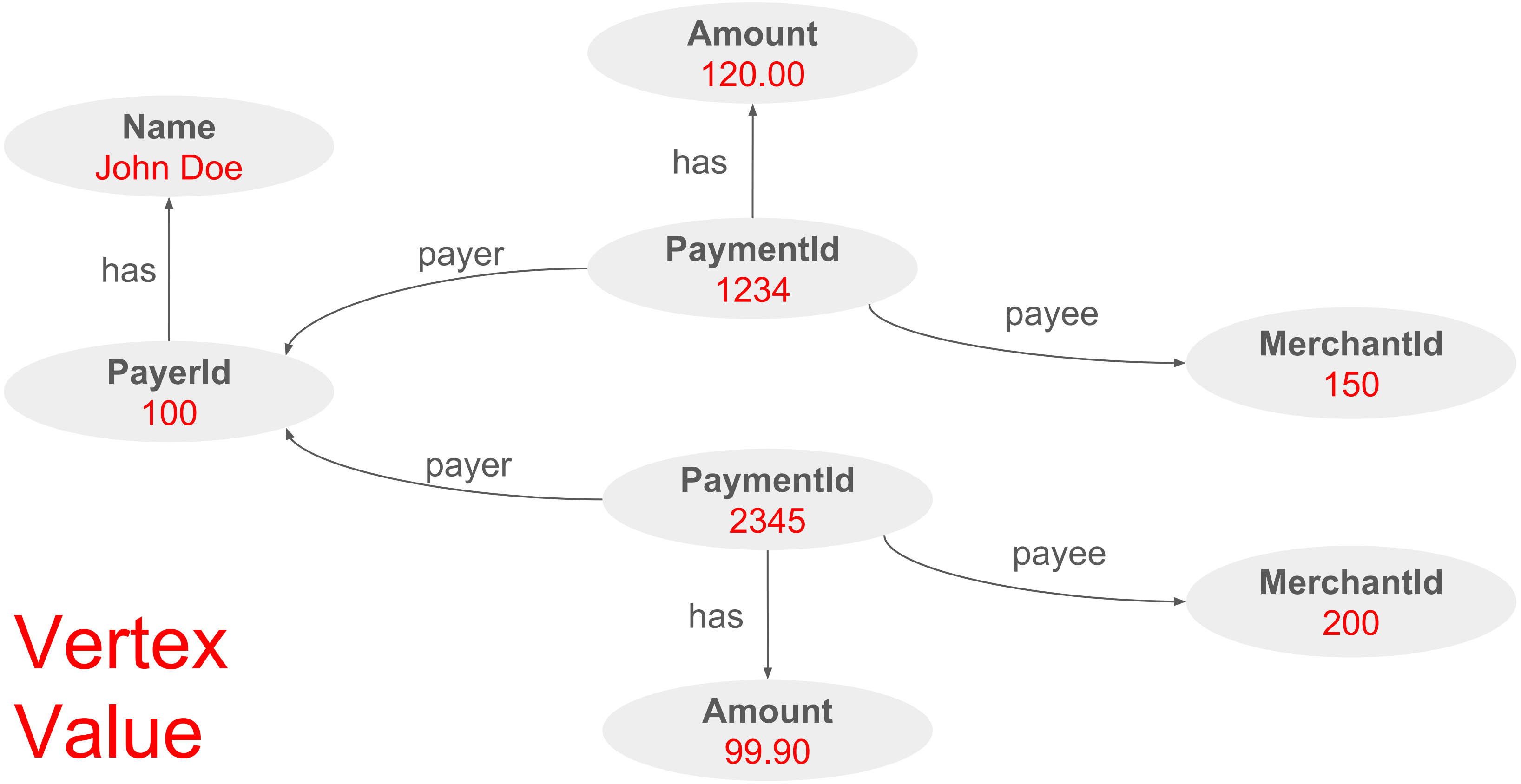
- (almost) in-memory database
    - Minimum JVM heap usage
    - Memory mapped file
    - Compact representation on file
        - Minimum record management overheads (minimum use of pointers, length fields, index structures, …)
- Distributed (partitioned)
- Updatable
    - Chronological segmentation
    - Segment merge


Amount
120.00
has
Name
John Doe
has
payer
PaymentId
1234
payee
MerchantId
150
PayerId
100
payer
PaymentId
2345
payee
MerchantId
200
has
Amount
99.90

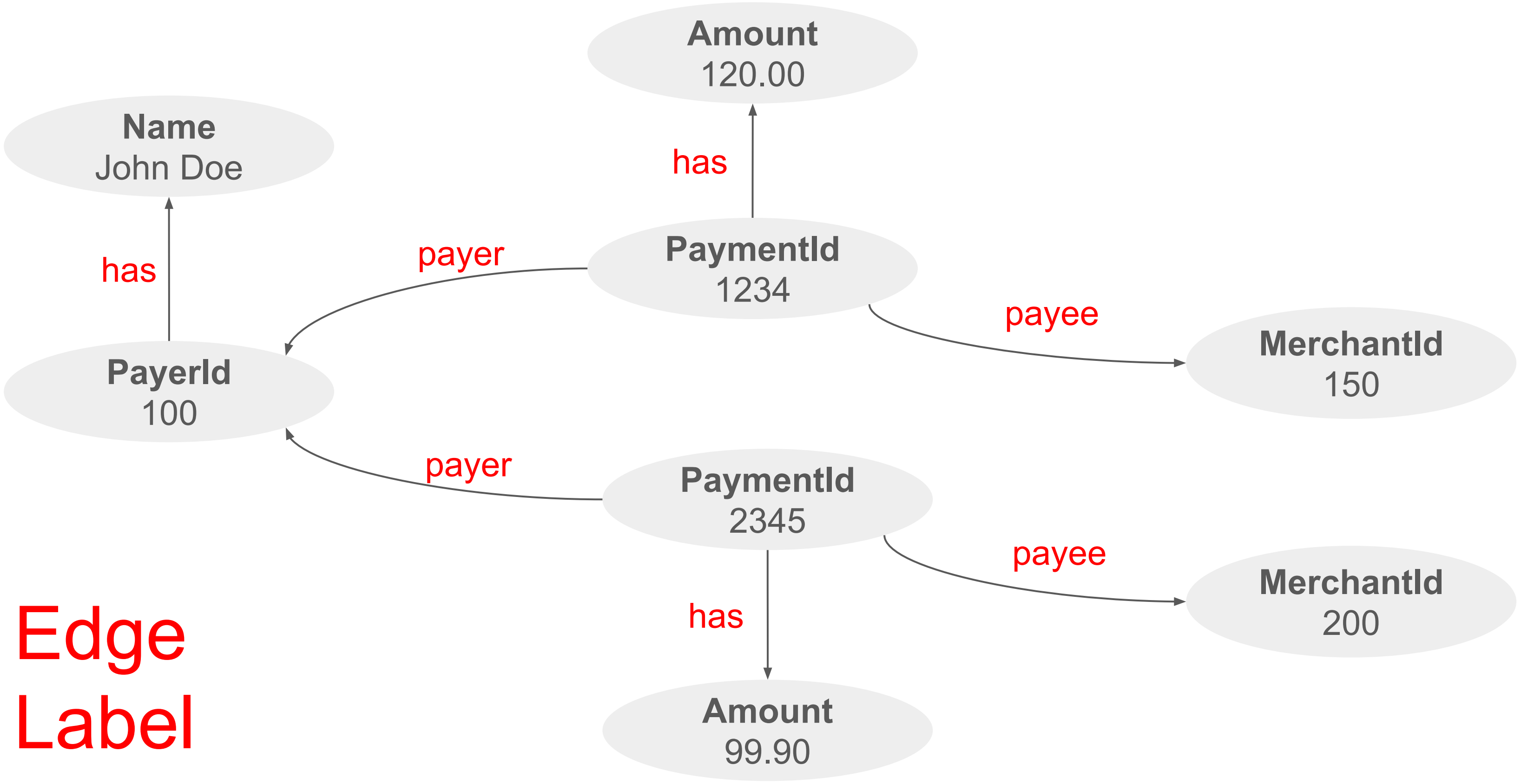
Graph


Amount
120.00
has
Name
John Doe
has
payer
PaymentId
1234
payee
MerchantId
150
PayerId
100
payer
PaymentId
2345
payee
MerchantId
200
has
Amount
99.90

Vertex
Type



Vertex
Value


Amount
120.00
has
Name
John Doe
has
payer
PaymentId
1234
payee
MerchantId
150
PayerId
100
payer
PaymentId
2345
payee
MerchantId
200
has
Amount
99.90

Edge Label

Data Model

Vertex

- Typed: Vertex Type is a subtype of Long, Double, or String
- No attribute

Edge

- Labeled (named)
    - A label is independent from vertex types of source/destination
- Directed
- No attribute

In the example graph...

Vertex Type

- PayerId : Long
- MerchantId: Long
- Amount : Double
- Name : String

Edge Label (Edge Type)

- payer, payee, has

Query Language (WIP)

```
SELECT PID, AMNT, M
FROM {
    P Payer,
    M Merchant,
    PID PaymentId,
    AMNT Amount

    P <-[payer]- PID -[payee]-> M,
    PID -[has]-> AMNT
}
WHERE P = 100 AND AMNT > 100.00;
```

```
SELECT PID, AMNT, M
FROM {
    P Payer,
    M Merchant,
    PID PaymentId,
    AMNT Amount

    P <-[payer]- PID -[payee]-> M,
    PID -[has]-> AMNT
}
WHERE P = 100 AND AMNT > 100.00;
```

Subgraph Specification

```
SELECT PID, AMNT, M
 FROM {
    P Payer,
    M Merchant,
    PID PaymentId,
    AMNT Amount

    P <-[payer]- PID -[payee]-> M,
    PID -[has]-> AMNT
}
WHERE P = 100 AND AMNT > 100.00;
```

Vertex Variable Declarations

```
SELECT PID, AMNT, M
 FROM {
    P Payer,
    M Merchant,
    PID PaymentId,
    AMNT Amount

    P <-[payer]- PID -[payee]-> M,
    PID -[has]-> AMNT
}
WHERE P = 100 AND AMNT > 100.00;
```

Vertex Variable Declarations

Vertex Variables

```
SELECT PID, AMNT, M
 FROM {
    P Payer,
    M Merchant,
    PID PaymentId,
    AMNT Amount

    P <-[payer]- PID -[payee]-> M,
    PID -[has]-> AMNT
}
WHERE P = 100 AND AMNT > 100.00;
```

Vertex Variable Declarations

Vertex Types

```
SELECT PID, AMNT, M
FROM {
    P Payer,
    M Merchant,
    PID PaymentId,
    AMNT Amount

    P <-[payer]- PID -[payee]-> M,
    PID -[has]-> AMNT
}
WHERE P = 100 AND AMNT > 100.00;
```

Subgraph Patterns

Edge Pattern

Single Edge Pattern

```
srcVertexVar -[ edgeLabel ]-> destVertexVar
```

or

```
destVertexVar <-[ edgeLabel ]- srcVertexVar
```

Edge patterns can be chained

```
v1 -[e1]-> v2 -[e2]-> v3 <-[e3]- v4
```

More Operators are planned

- GROUP BY
- HAVING
- ORDER BY
- Aggregate functions (COUNT, SUM, AVG, MIN, MAX)

Subgraph Specification ≈ Joined Table Expression

Edge Pattern ≈ Two-Column Table

```
V1 -[e]-> V2

≈ (V1, V2)
```

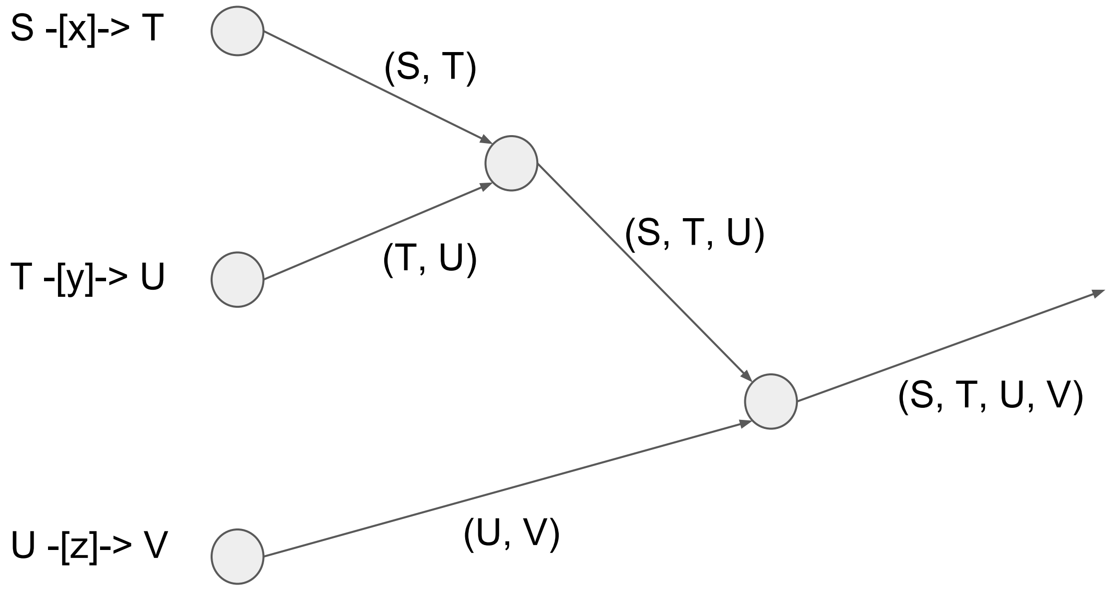
Edge Patterns are joined over shared Vertex Variables

```
V1 -[e]-> V2, V2 -[e]-> V3

≈ (V1, V2) join (V2, V3) over V2
```

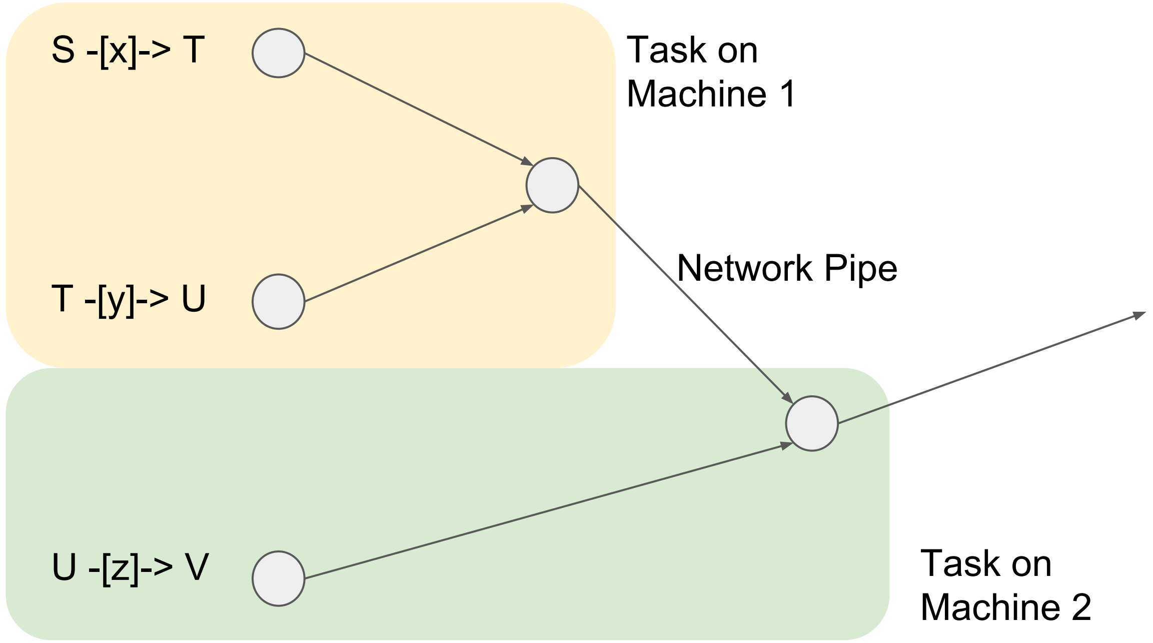
Subgraph Spec → Join Tree

S -[x]-> T -[y]-> U -[z]-> V

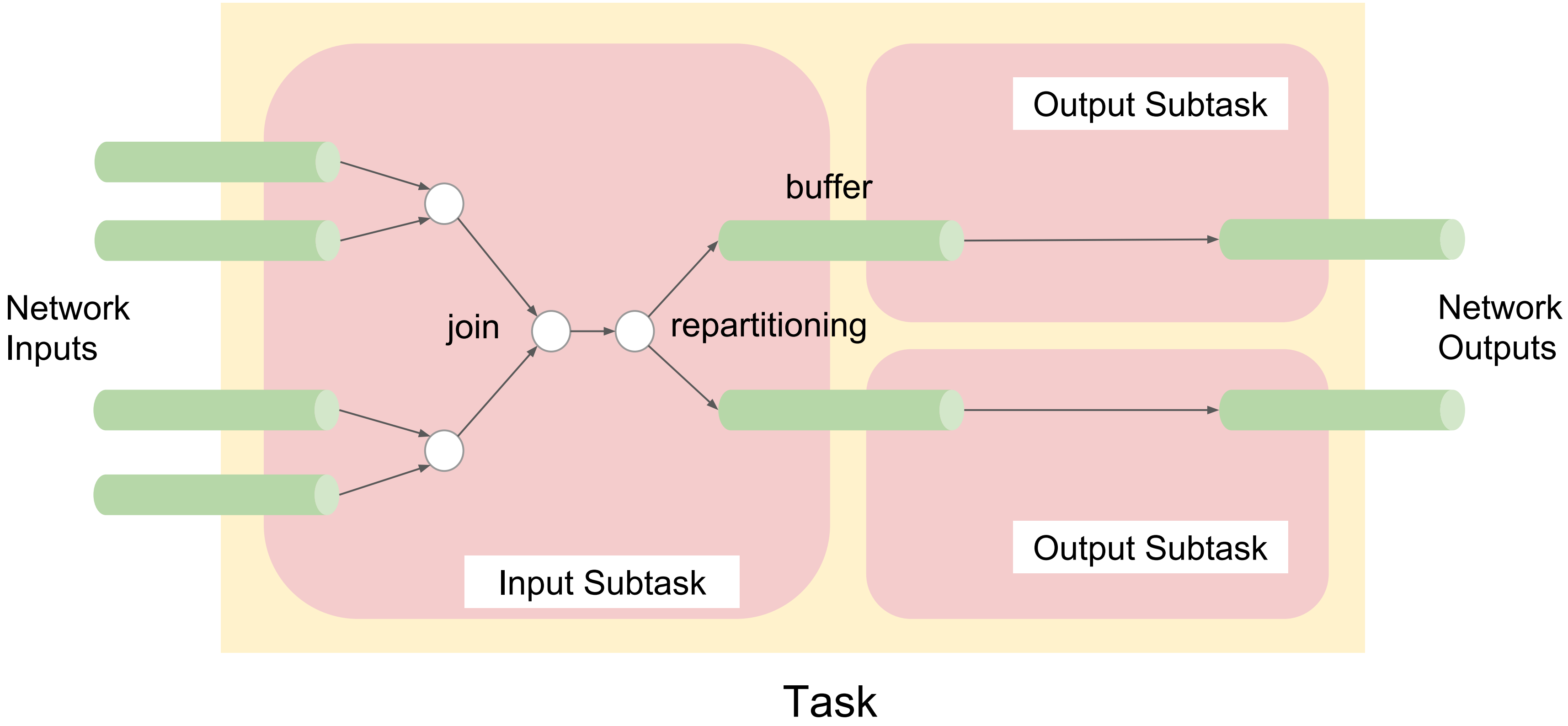


Distributed Processing

S -[x]-> T -[y]-> U -[z]-> V



Task Structure



Subtasks and Inputs/Outputs

a subtask ≈ a thread

- Subtasks run in a thread pool
- A subtask should never block.

Input/Output may be suspended when data are not ready

- If input/output is not ready, a subtask is suspend.
- The suspended subtask is resumed when input/output becomes ready.

(see LocalProcessor)

Query Optimization

- Predicate Push-down
- Cost Based
- Exhaustive Search

Query Optimization

- Predicate Push-down
- Cost Based → Ad hoc, Naive Cost Model
- Exhaustive Search → Slow

Task Generation and Distribution

- A query plan is generated at a server
- A row source tree is generated from the plan
- TaskDefs are created from the row source tree
- A task is a combination of a TaskDef and a partition
- Tasks are assigned to servers
- Send tasks and bind variables to assigned servers

Storage

Edge List (source, destination, edge type)

- Compressed by removing repeating source vertices
    - Vertex List (list of <source vertex>)
    - Out-Edge List (list of <destination vertex, edge type>)
    - A succinct bit vector connects a vertex list and an out-edge list.
- Vertex Records and Out-Edge records are fixed size
- Vertex Records are sorted
- Out-Edge Records are grouped by source vertex, and sorted within the group

Inverted Index

- Indexed by destination vertices

Storage (cont.)

Inverted Index

- Indexed by hash values of destination vertices

Reverse Edge

- Reverse Edge is automatically created
  - An edge, S -[e]-> D, creates a reverse edge D -[e^{-1}]-> S implicitly.
  - A reverse edge is use by the engine internally.

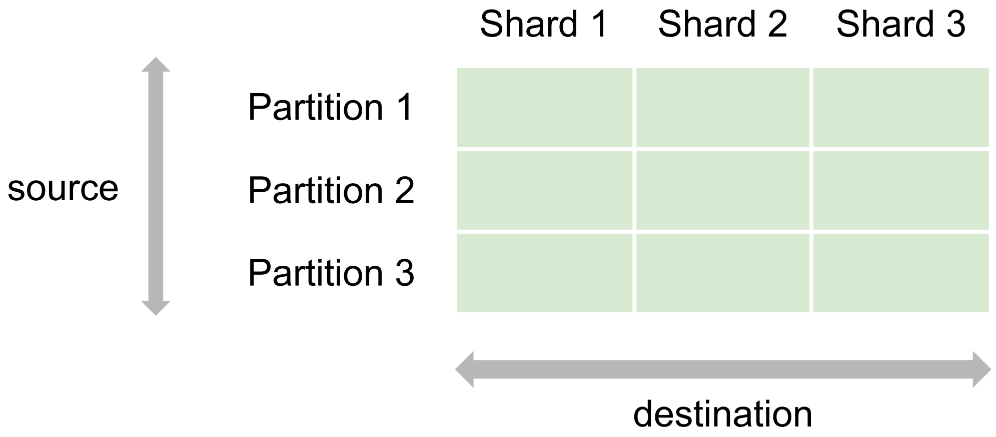
String Pool

String Vertex Value is a variable length data.

- Create a list of string values (sorted)
- Store pointer to a string in the pool in a vertex record.

Partitions and Shards

- Hash-partitioned by source vertices
- A partition is hash-partitioned into shards by destination vertices



Segments

A shard is chronologically segmented.

- Two in-memory segments
  - Writable in-memory segment (new data is written to it)
  - Read-only in-memory segment (it will be dumped to the disk)
- On-disk segments
  - A segment contains a Vertex list, an Out-edge list, a string pool, an inverted index
  - Segments are merged when there are too many.
  - We try to balance the merged segment sizes. (see MergePolicyImpl)

TODOs

- Language Parser [WIP]
- Language Features
  - Aggregate (group by, having, aggregate functions)
  - Arithmetic operators
  - Additional primitive types (Datetime?)
  - Edge label variable?
- Data Dictionary (schema managment)
- Client Library
  - Native networking vs gRPC
- Data Ingestion
  - Waltz integration?
- Cluster management (Config?, ZK integration?, Gossip Protocol?)

com.wepay.cadenza.common

- message package
    - Network messages
- network package
    - Networking layer (on Netty, copied from Waltz)
- pipe package
    - Network pipe (rows flow between remote tasks through this)
    - Local pipe (rows flow between local tasks through this)
- task package
    - Task, Subtask, TaskProcessor
- type package
    - Vertex, Accessor, Bind variable, Value, ...

com.wepay.cadenza.server.query

- engine package
  - RowDef (describes a row)
  - RowSources (data access, join, filter, sort) and RowInputs
  - RuntimeContextImpl (access to bind vars, routing table, storage, and network pipe)
  - Task implementation
- plan package
  - Query execution plan generation
  - Row source tree generation
- Query classes
  - Subgraph, Where, OrderBy, Select
- Query Processor class
  - LocalProcessor

com.wepay.cadenza.server.storage

- segment package
  - In-memory segment
  - On-disk segment
- util package
  - ImmutableBST (immutable randomized binary search tree)
  - SuccinctBitVector
- Input classes
  - VertexInput, OutEdgeInput, EdgeInput, …
- Partition class
- Shard class