

Piccolo

Fast, Distributed Programs with Partitioned Tables

Presenter: Wu, Weiyi
Yale University

Outline

- Background
- Intuition
- Design
- Evaluation
- Future Work

Outline

- **Background**
- Intuition
- Design
- Evaluation
- Future Work

MapReduce

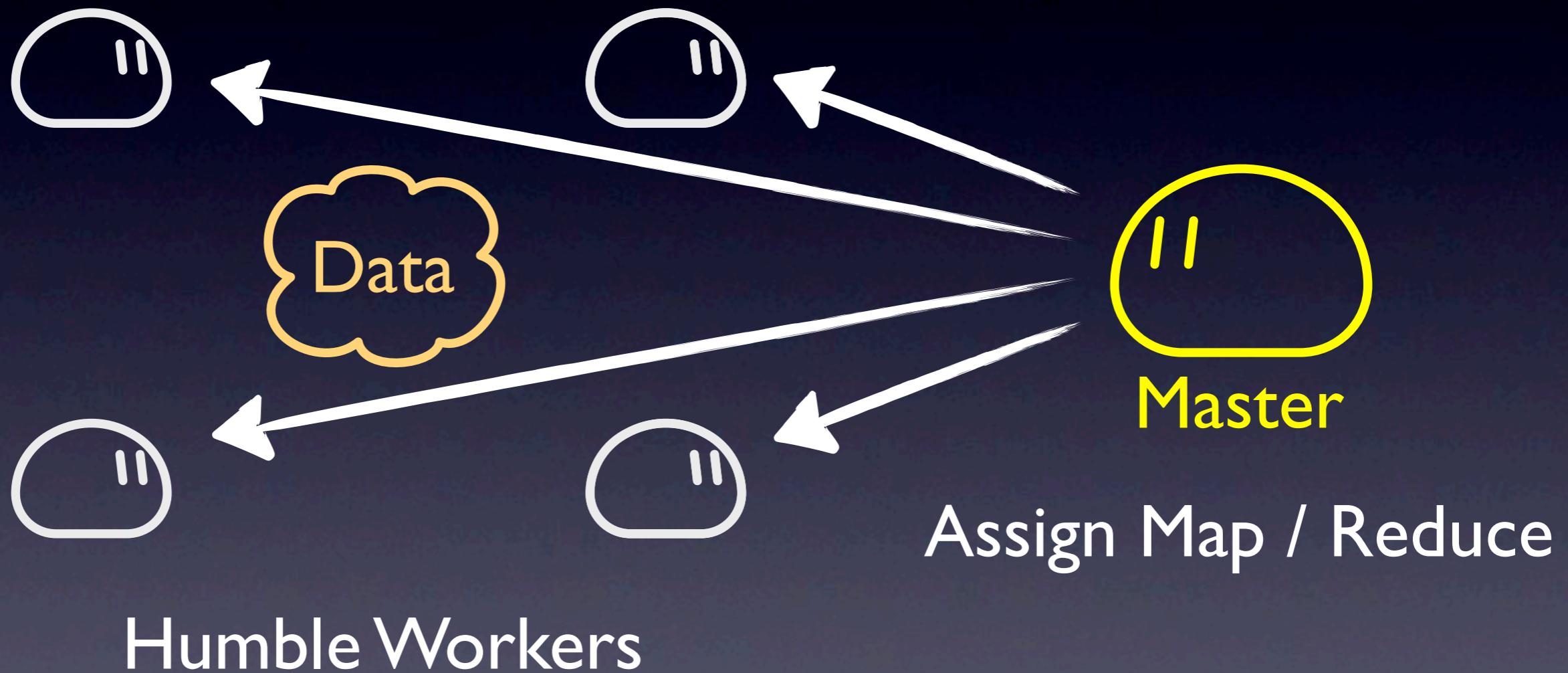


Master



Humble Workers

MapReduce



MapReduce



Master

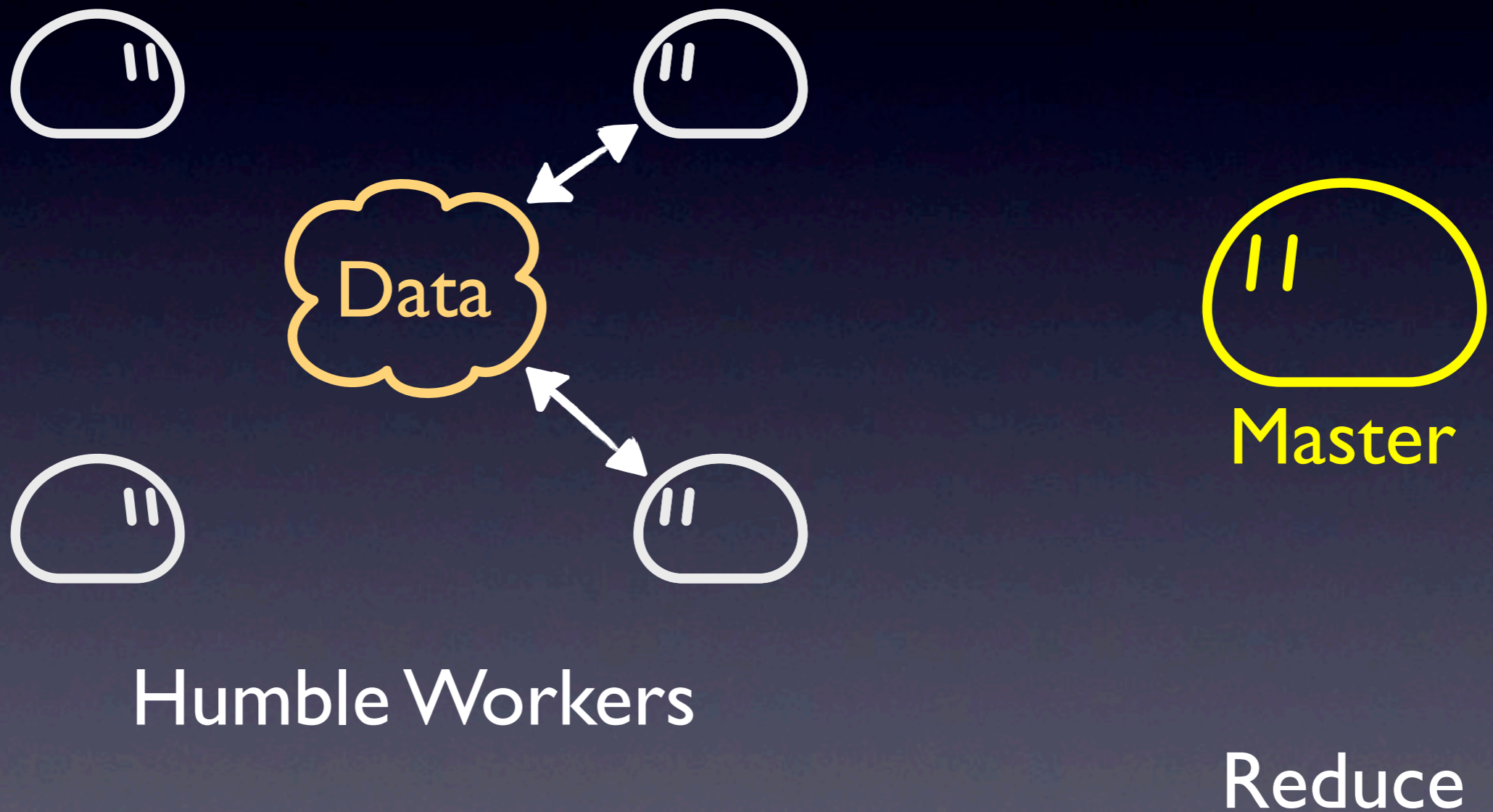


Humble Workers

MapReduce



MapReduce



MapReduce



Master



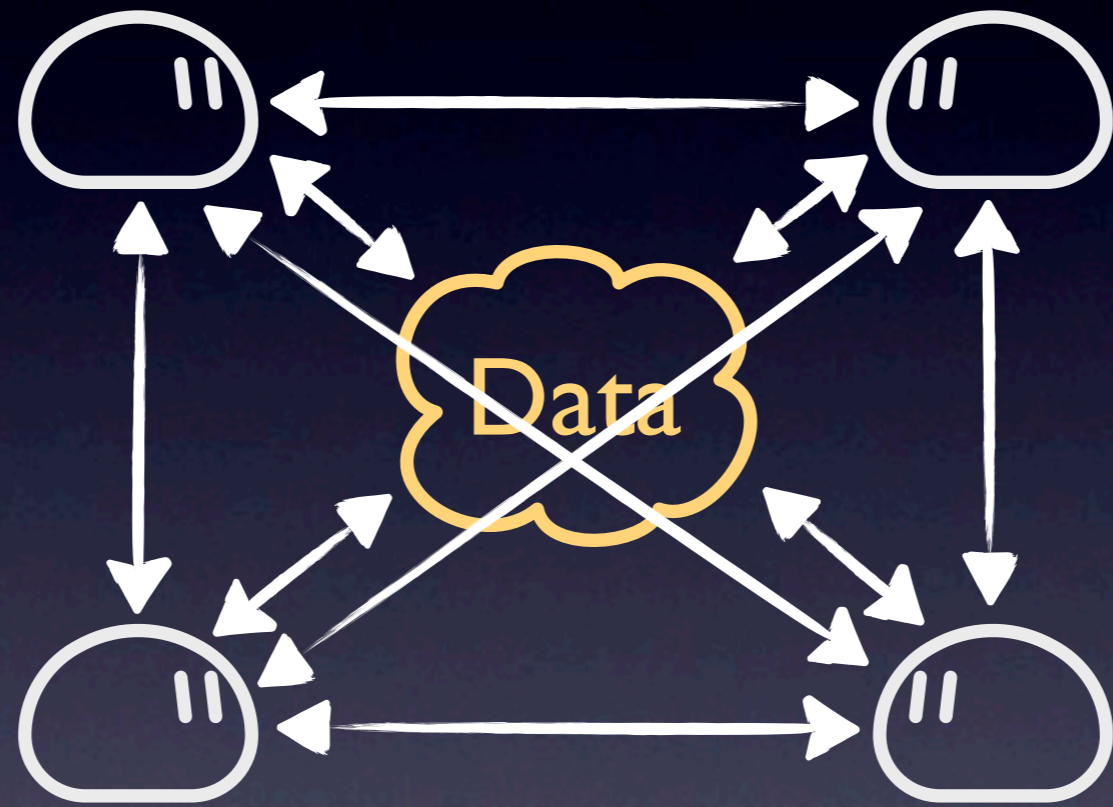
Humble Workers

MPI / RPC



Humble Workers

MPI / RPC



Messages
All Around

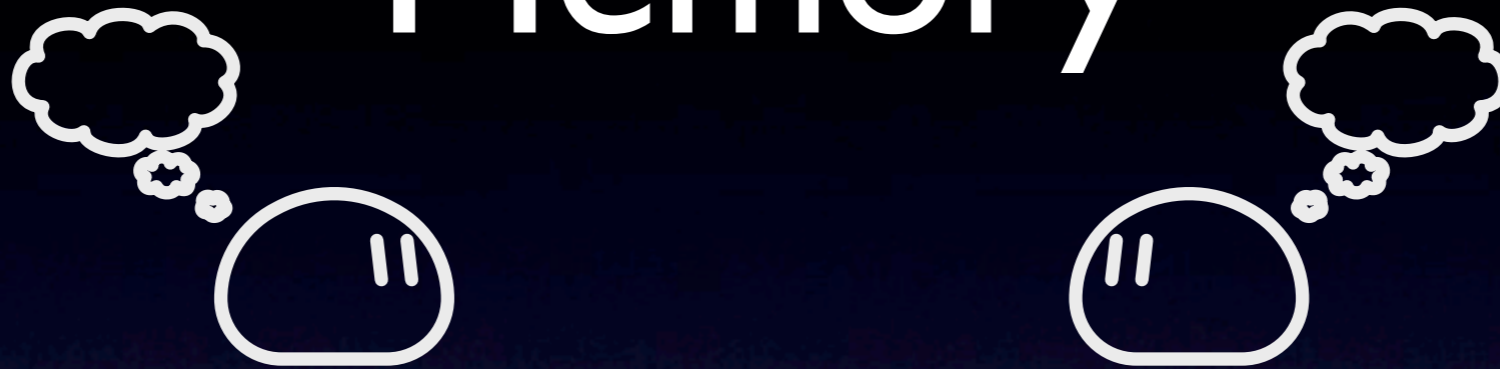
Humble Workers

MPI / RPC



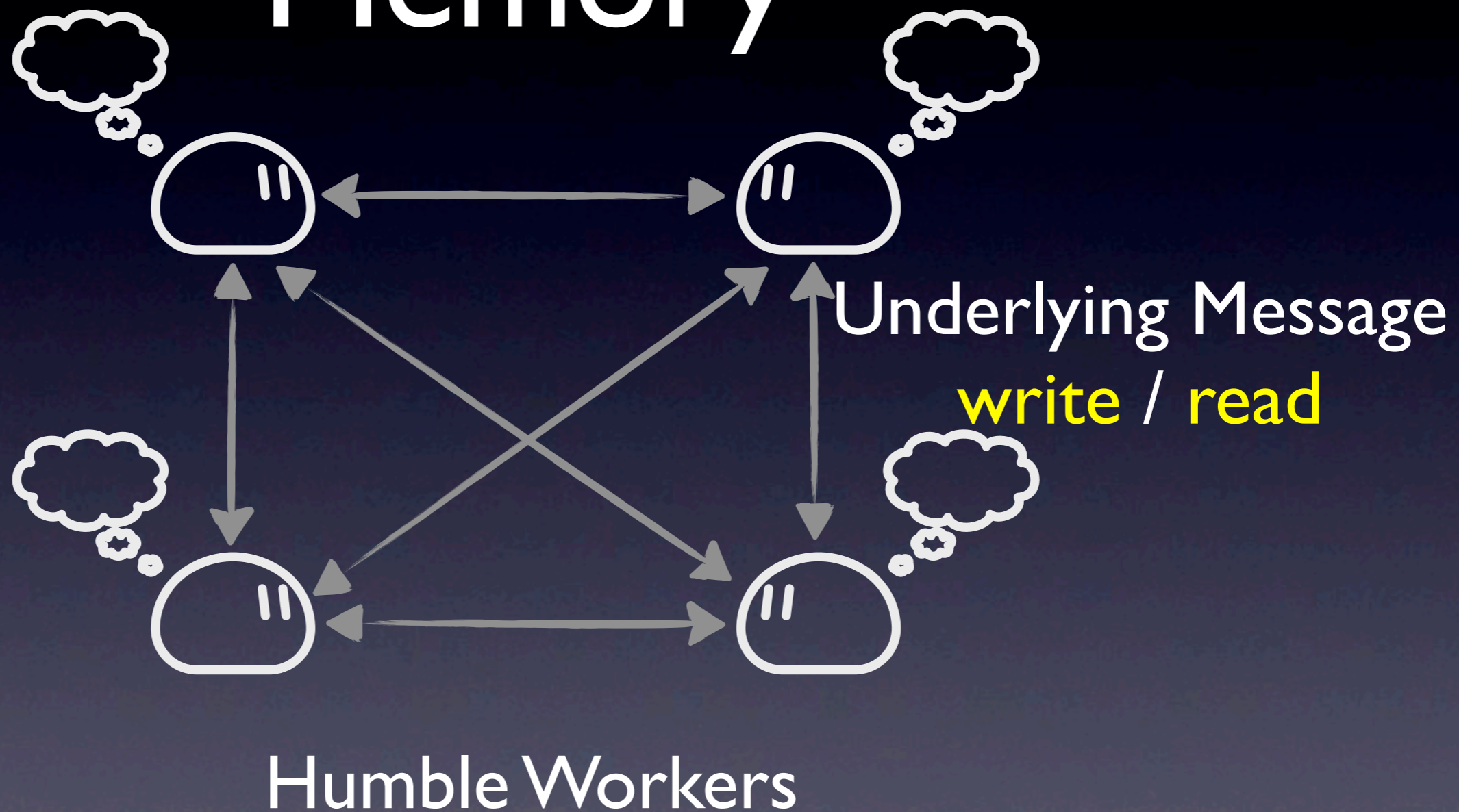
Humble Workers

Distributed Shared Memory

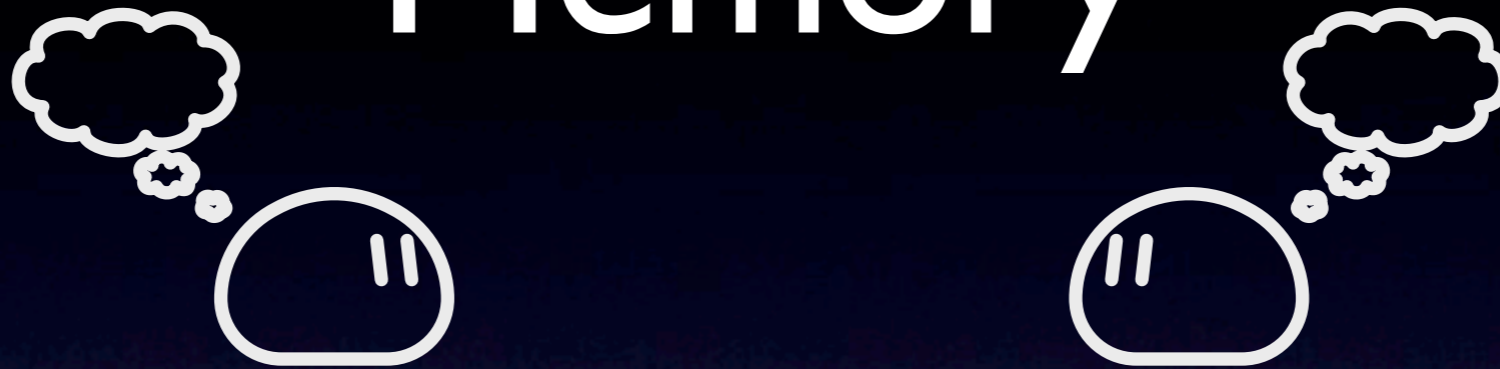


Humble Workers

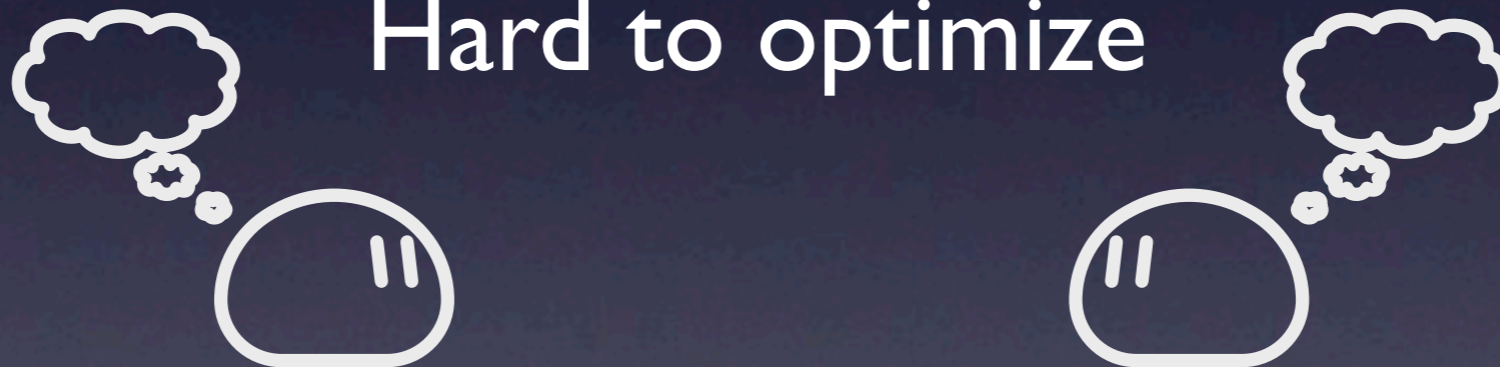
Distributed Shared Memory



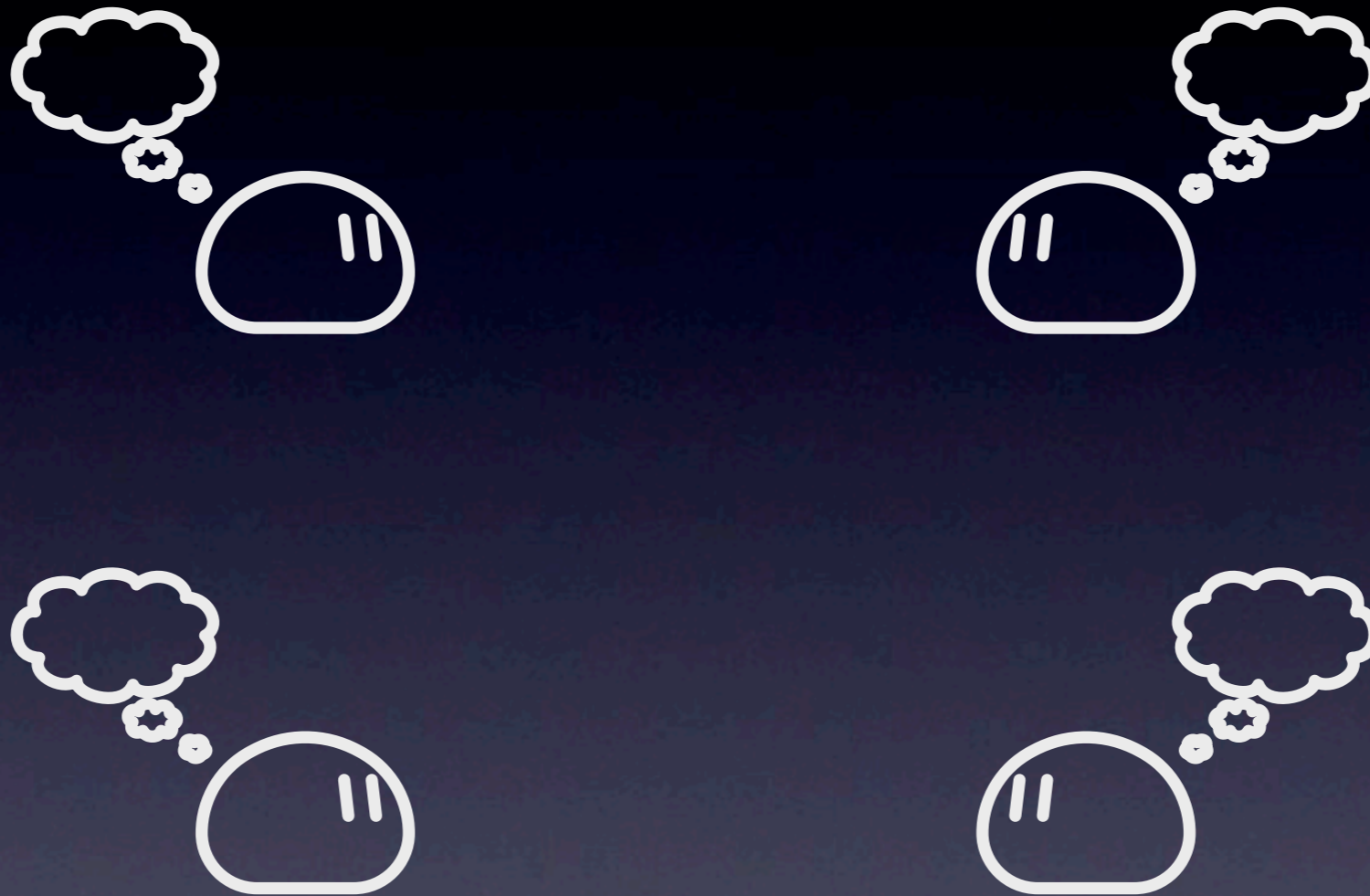
Distributed Shared Memory



Non-atomic
Hard to optimize

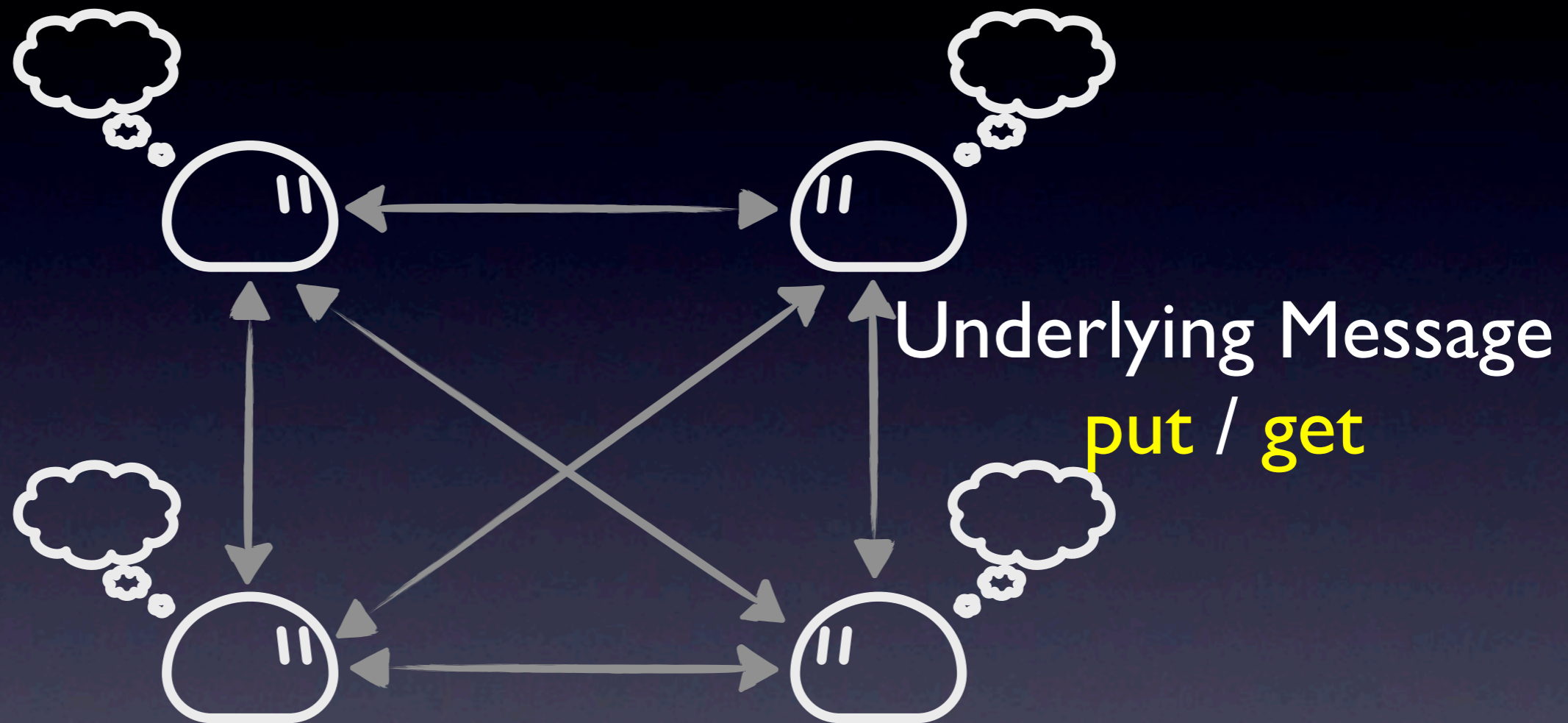


Key-Value Table



Humble Workers

Key-Value Table



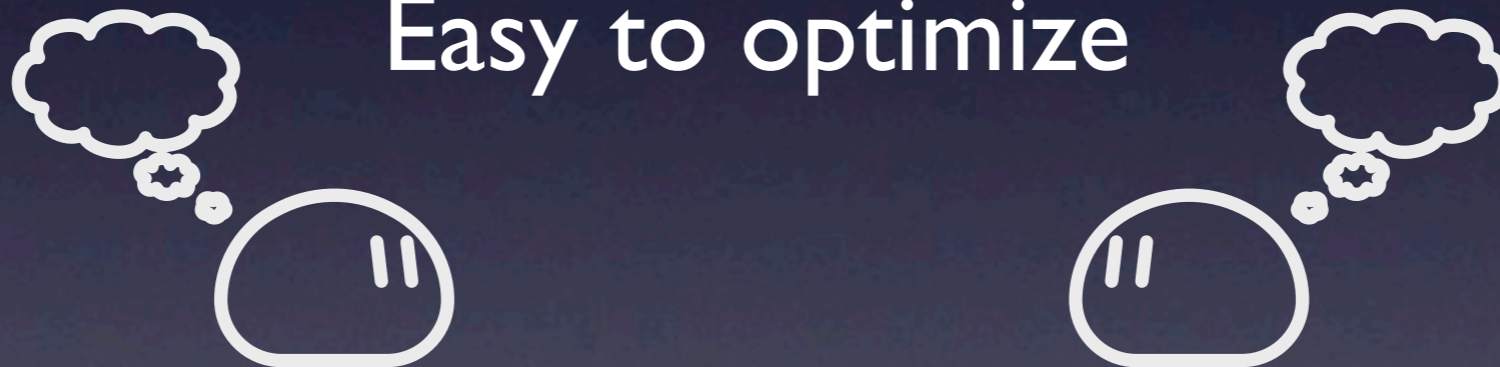
Humble Workers

Key-Value Table



Atomic

Easy to optimize



Outline

- Background
- **Intuition**
- Design
- Evaluation
- Future Work

What do we need?

MapReduce

MPI / RPC

DSM

k/v Table

What do we need?

- In-memory

MapReduce

MPI / RPC

DSM

k/v Table

What do we need?

- In-memory
- Data-centric

MapReduce

MPI / RPC

DSM

k/v Table

What do we need?

- In-memory
- Data-centric
- Exposing globally shared state

MapReduce

MPI / RPC

DSM

k/v Table

What do we need?

- In-memory
- Data-centric
- Exposing globally shared state

MPI / RPC

DSM

k/v Table

What do we need?

- In-memory
- Data-centric
- Exposing globally shared state
- No low-level messages

MPI / RPC

DSM

k/v Table

What do we need?

- In-memory
- Data-centric
- Exposing globally shared state
- No low-level messages

DSM

k/v Table

What do we need?

- In-memory
- Data-centric
- Exposing globally shared state
- No low-level messages
- Easy to use / optimize

DSM

k/v Table

What do we need?

- In-memory
- Data-centric
- Exposing globally shared state
- No low-level messages
- Easy to use / optimize

k/v Table

What do we need?

- In-memory
- Data-centric
- Exposing globally shared state
- No low-level messages
- Easy to use / optimize

Is k/vTable enough?

- Replace put-get pairs to atomic ops
- Improving locality
- Load Balancing
- Rapid and Reliable Checkpoint

Outline

- Background
- Intuition
- Design
- Evaluation
- Future Work

Overview

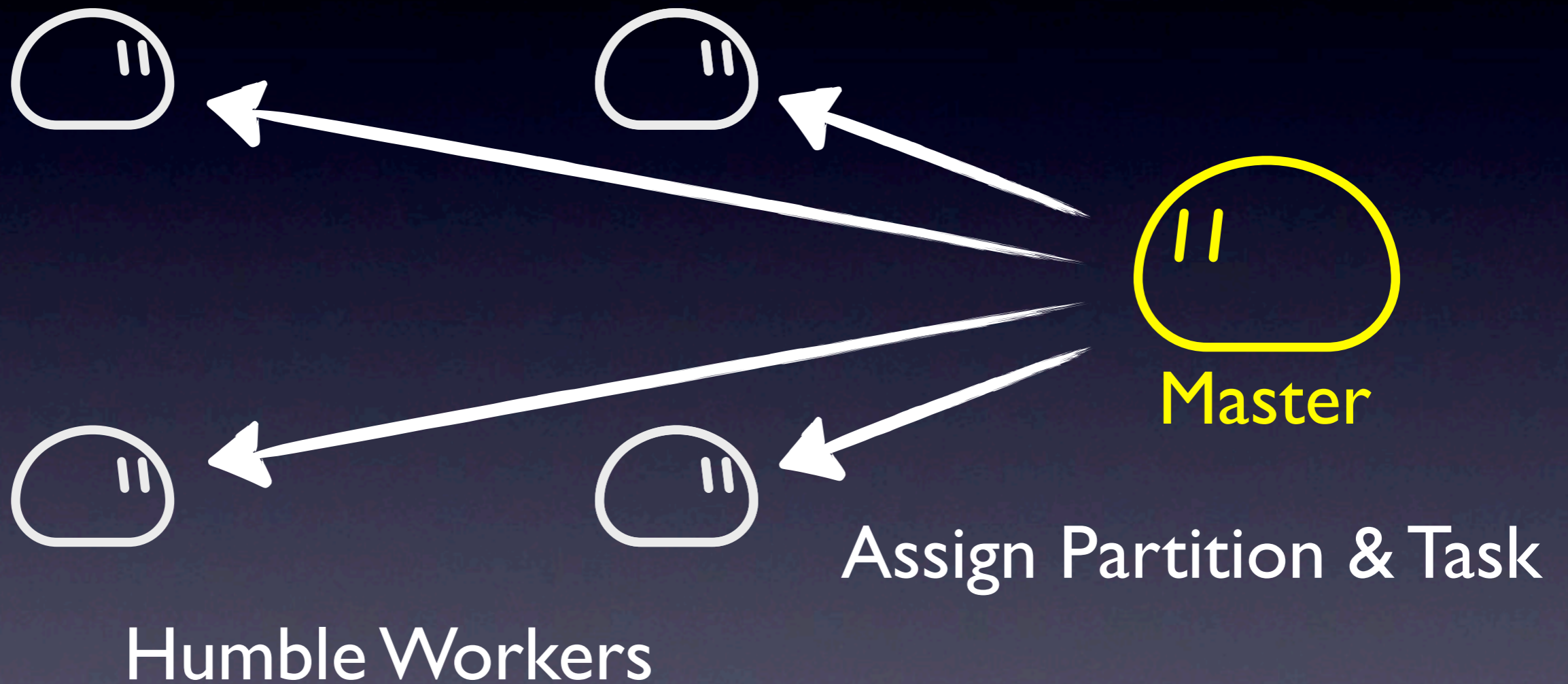


Master



Humble Workers

Overview



Overview



Overview



Overview



Humble Workers

Overview



Humble Workers

Kernel Finished

Overview



Humble Workers

Expressing Locality

- Reduce **remote read (get)**
- Co-locate a kernel execution with some table partitions
- Co-locate partitions of different tables (with same partition id)

User-defined accumulators

User-defined accumulators

- `a ← get(A)`
- `b ← get(B)`
- `res ← a + b`
- `put(B, res)`

User-defined accumulators

- `a ← get(A)`
 - `b ← get(B)`
 - `res ← a + b`
 - `put(B, res)`
- `a ← get(A)`

User-defined accumulators

- `a ← get(A)`
 - `b ← get(B)`
 - `res ← a + b`
 - `put(B, res)`
- `a ← get(A)`
 - `update(B, a)`

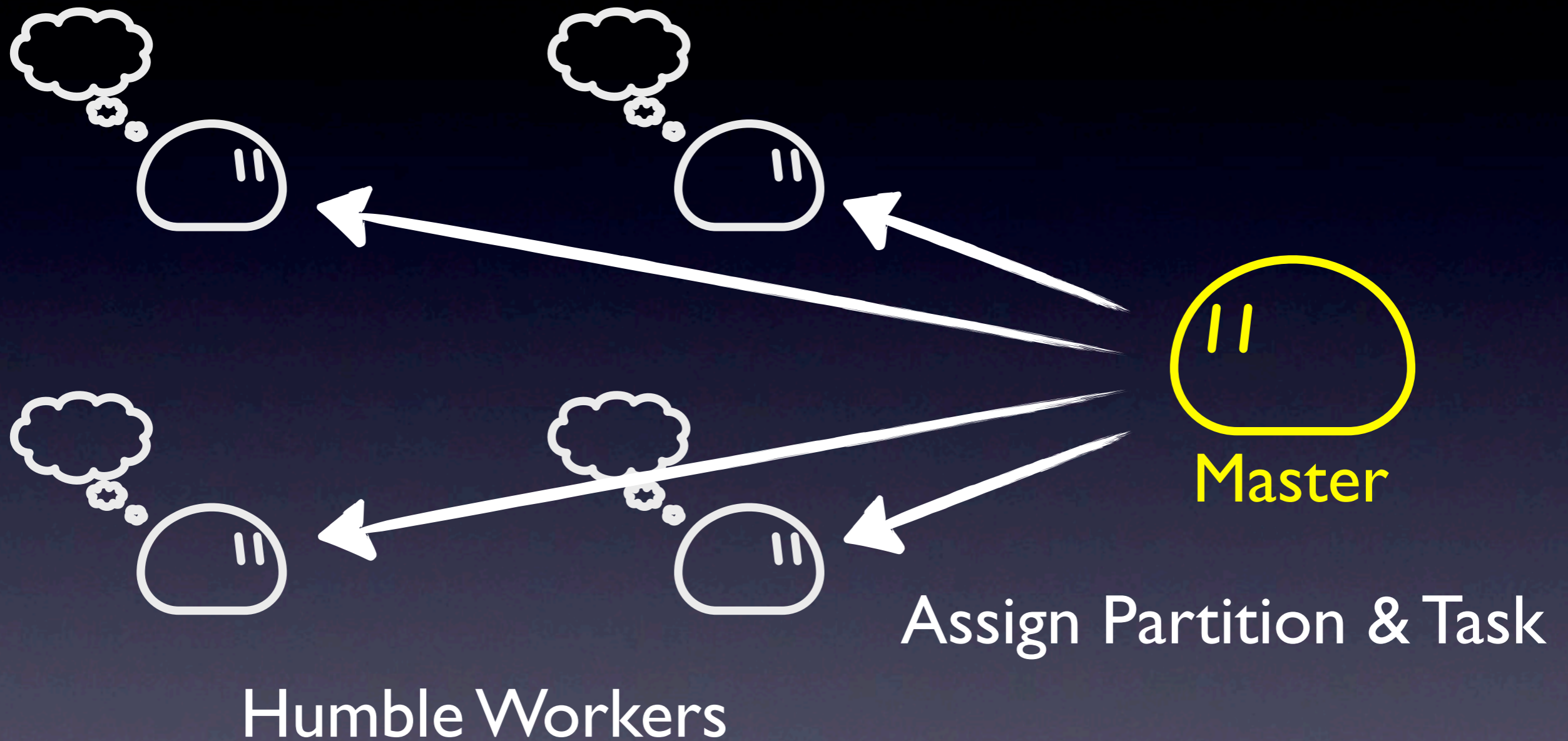
Load Balance



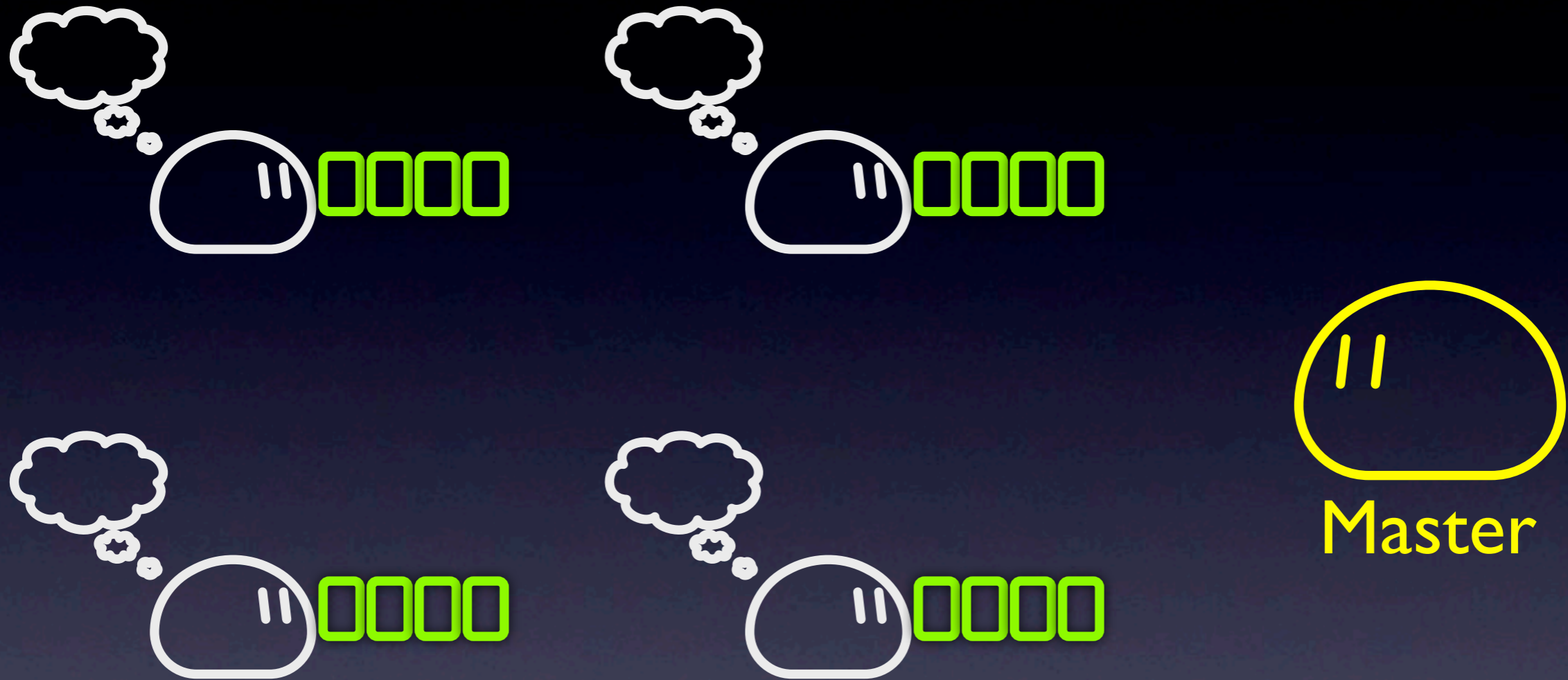
Master

Humble Workers

Load Balance

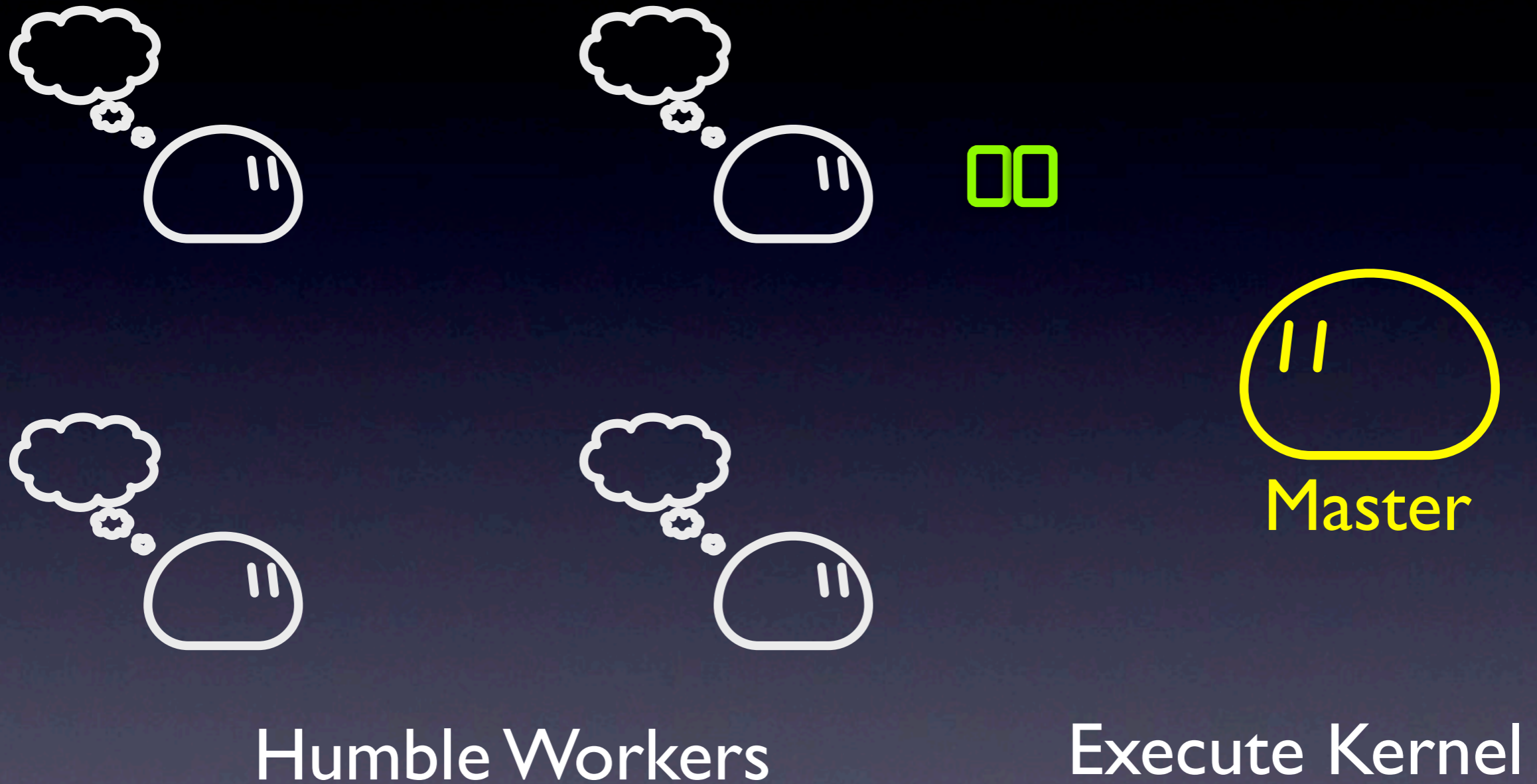


Load Balance

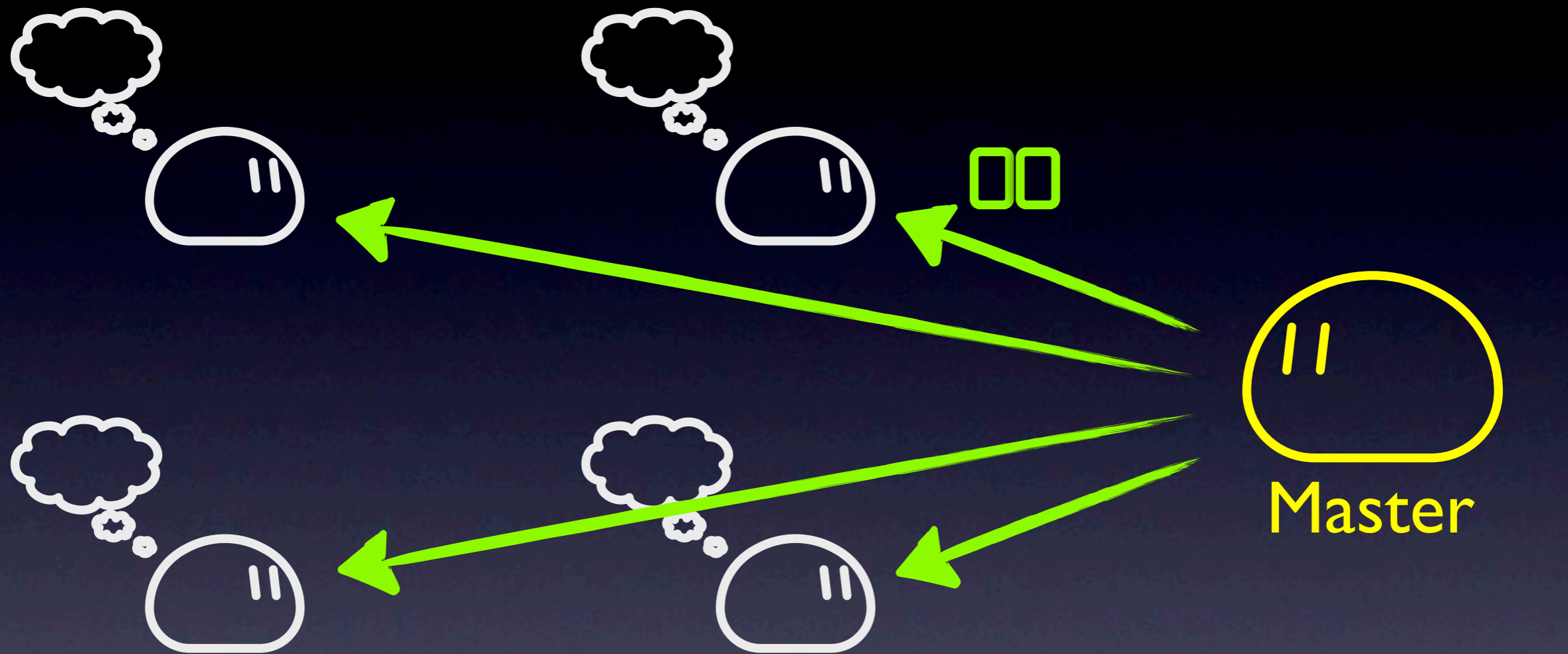


Humble Workers

Load Balance



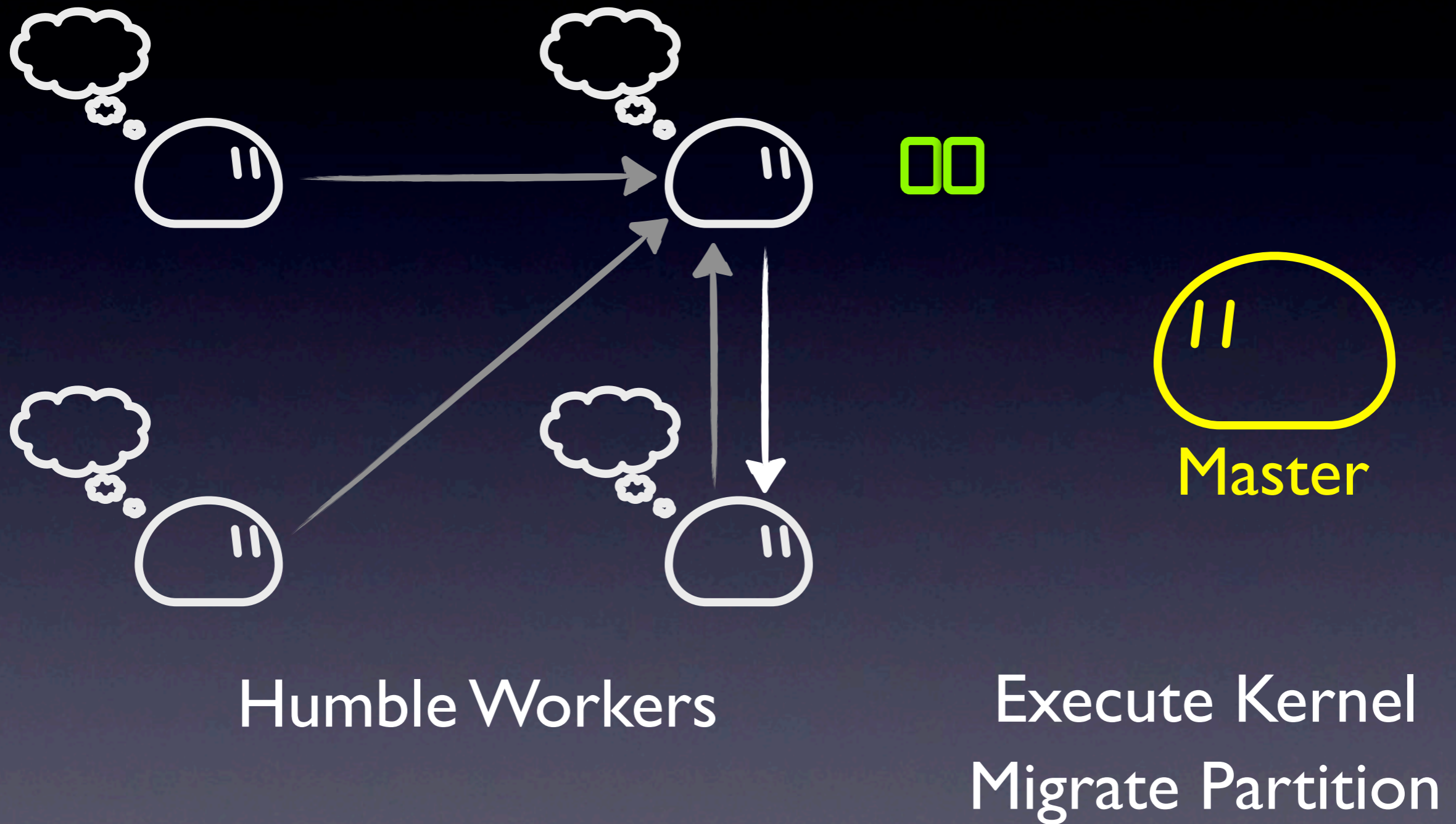
Load Balance



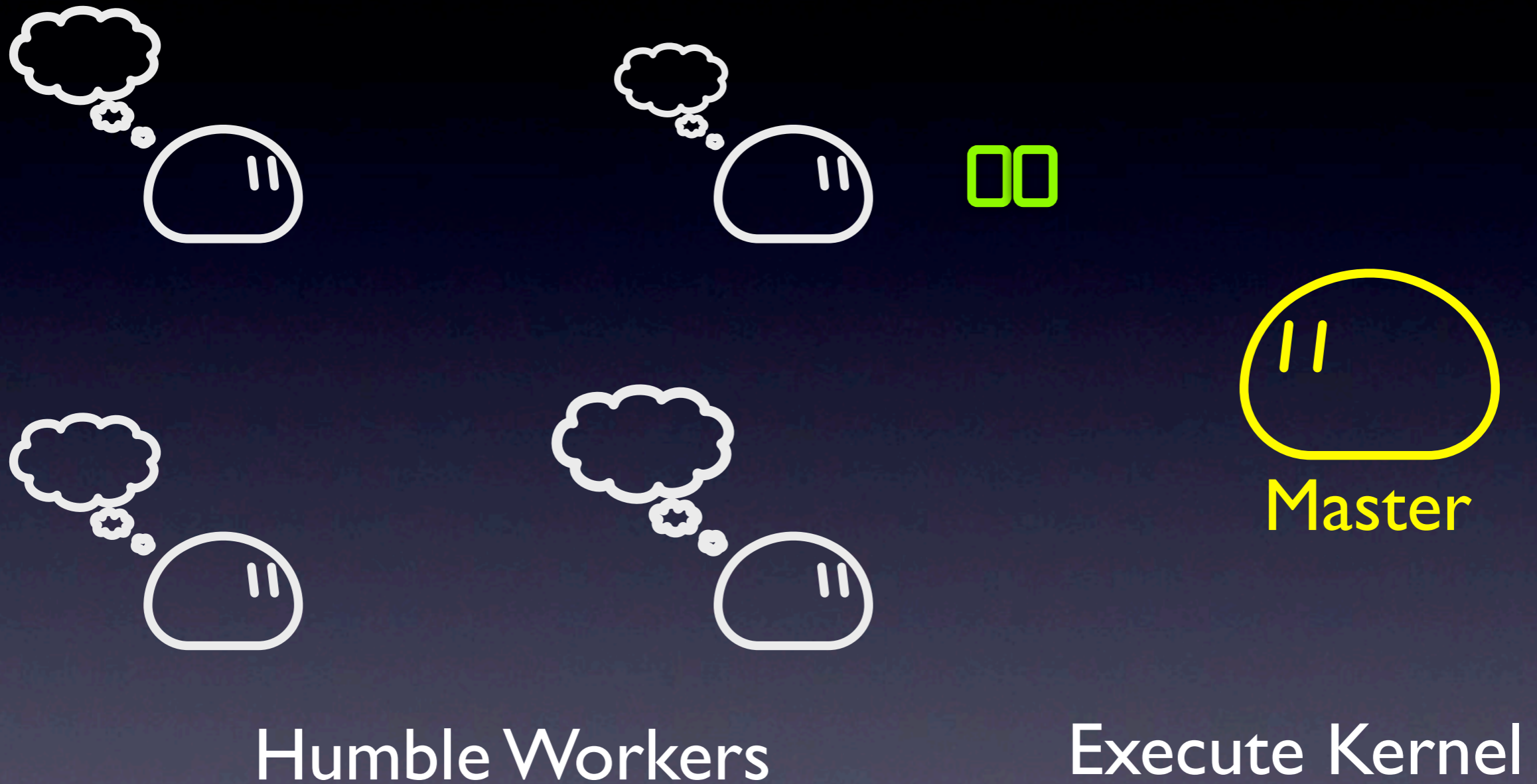
Humble Workers

Execute Kernel
Migrate Partition

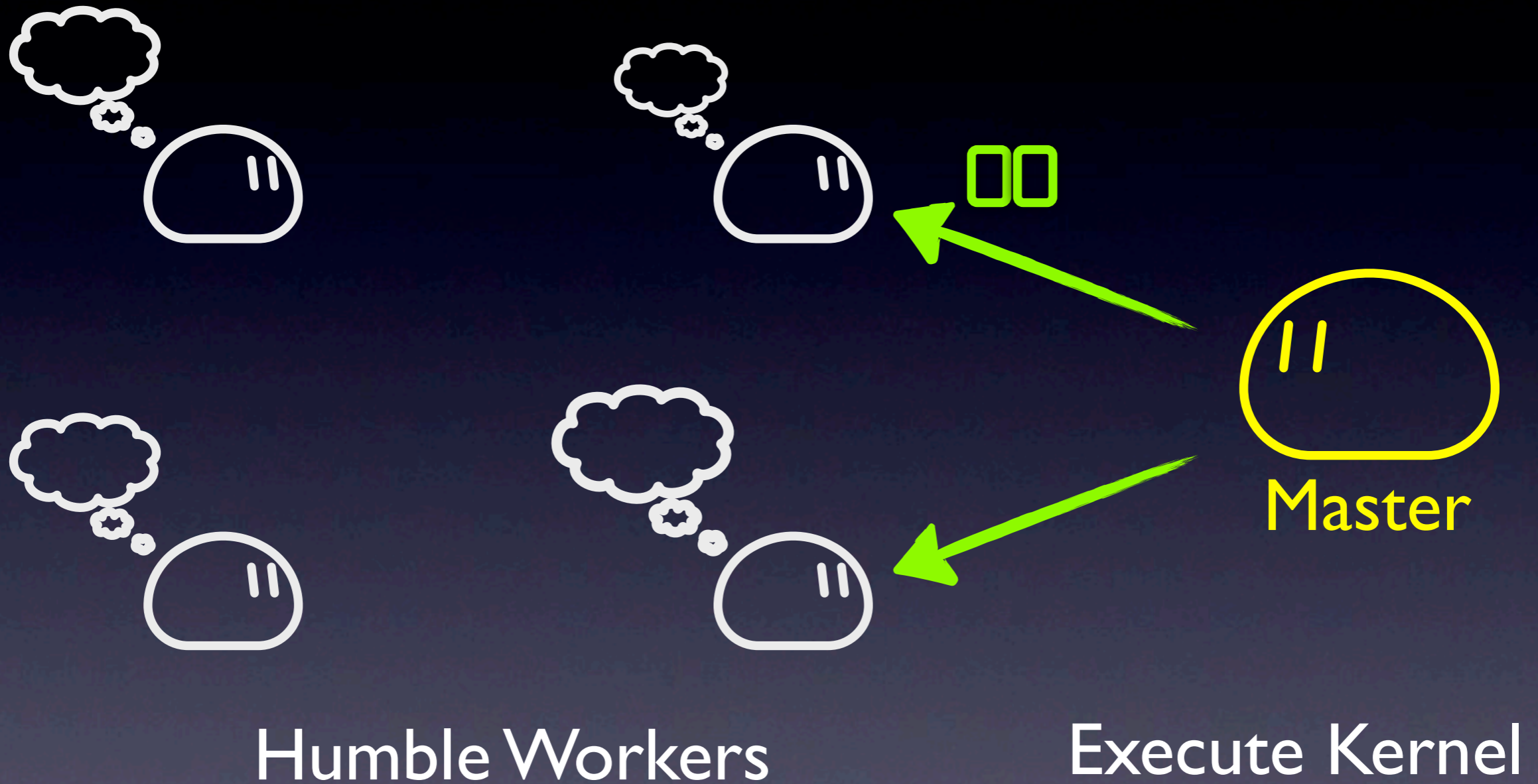
Load Balance



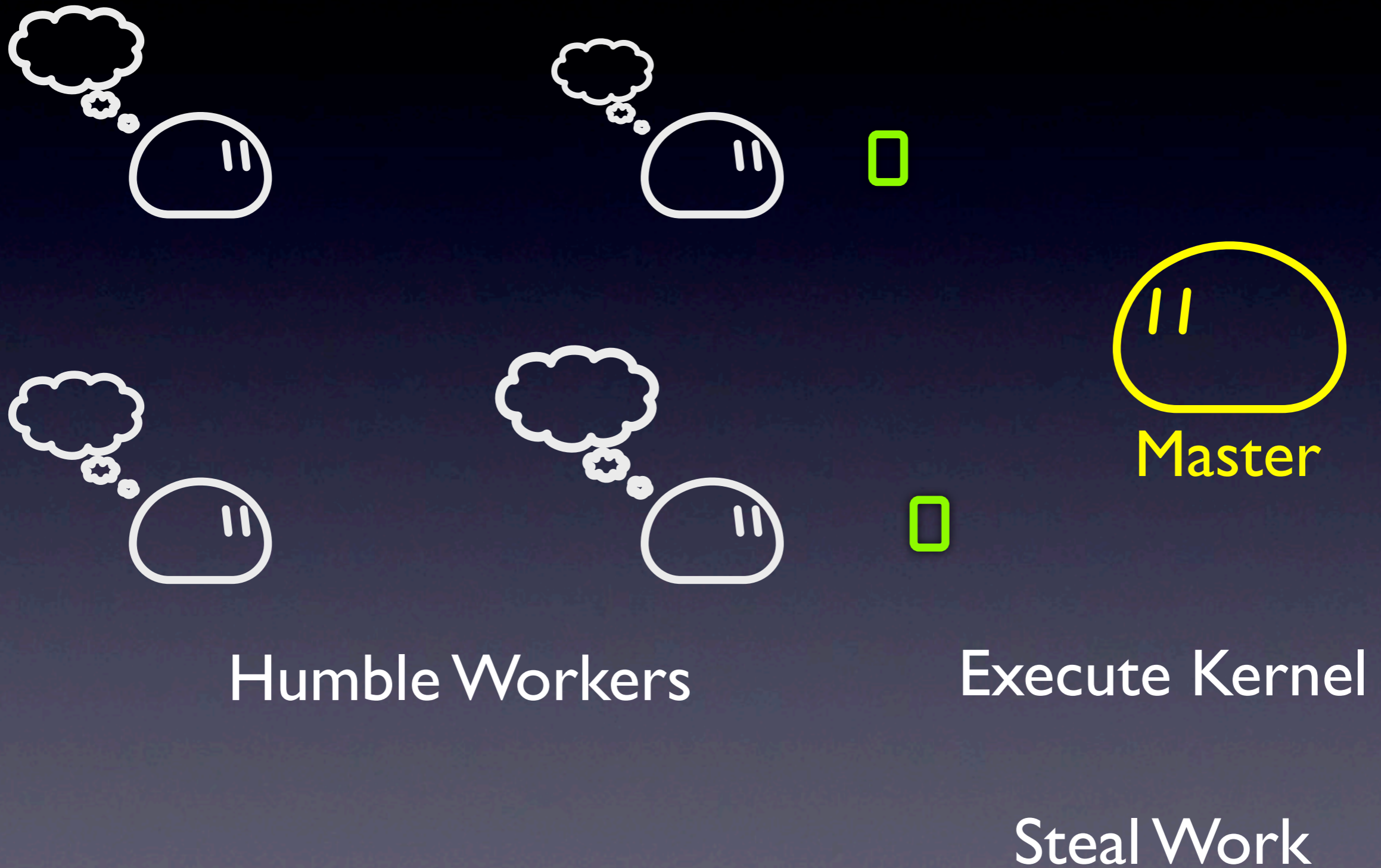
Load Balance



Load Balance



Load Balance



Load Balance



Master

Humble Workers

Execute Kernel

Load Balance



Humble Workers

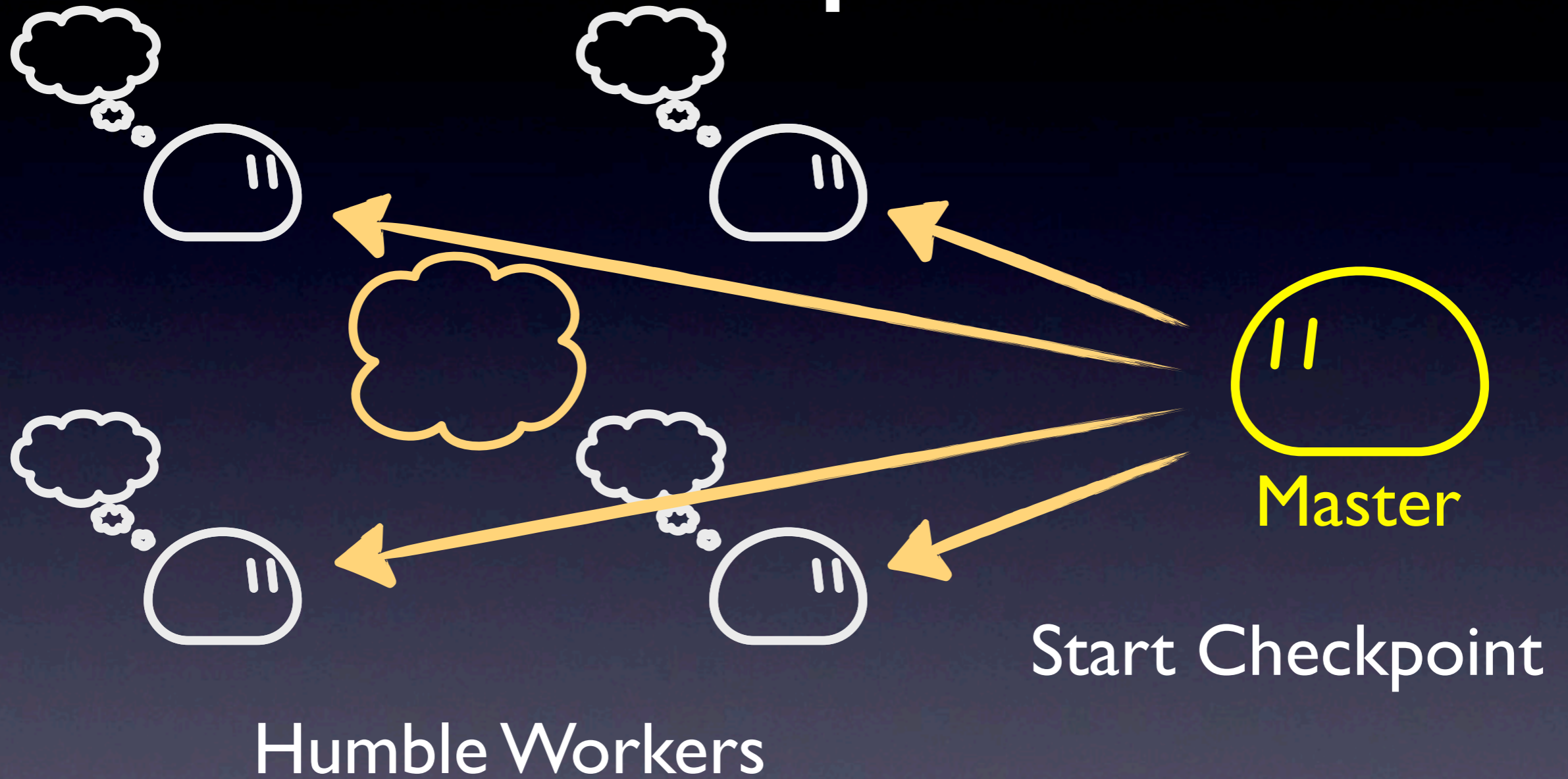
Checkpoint



Humble Workers

Master

Checkpoint



Checkpoint

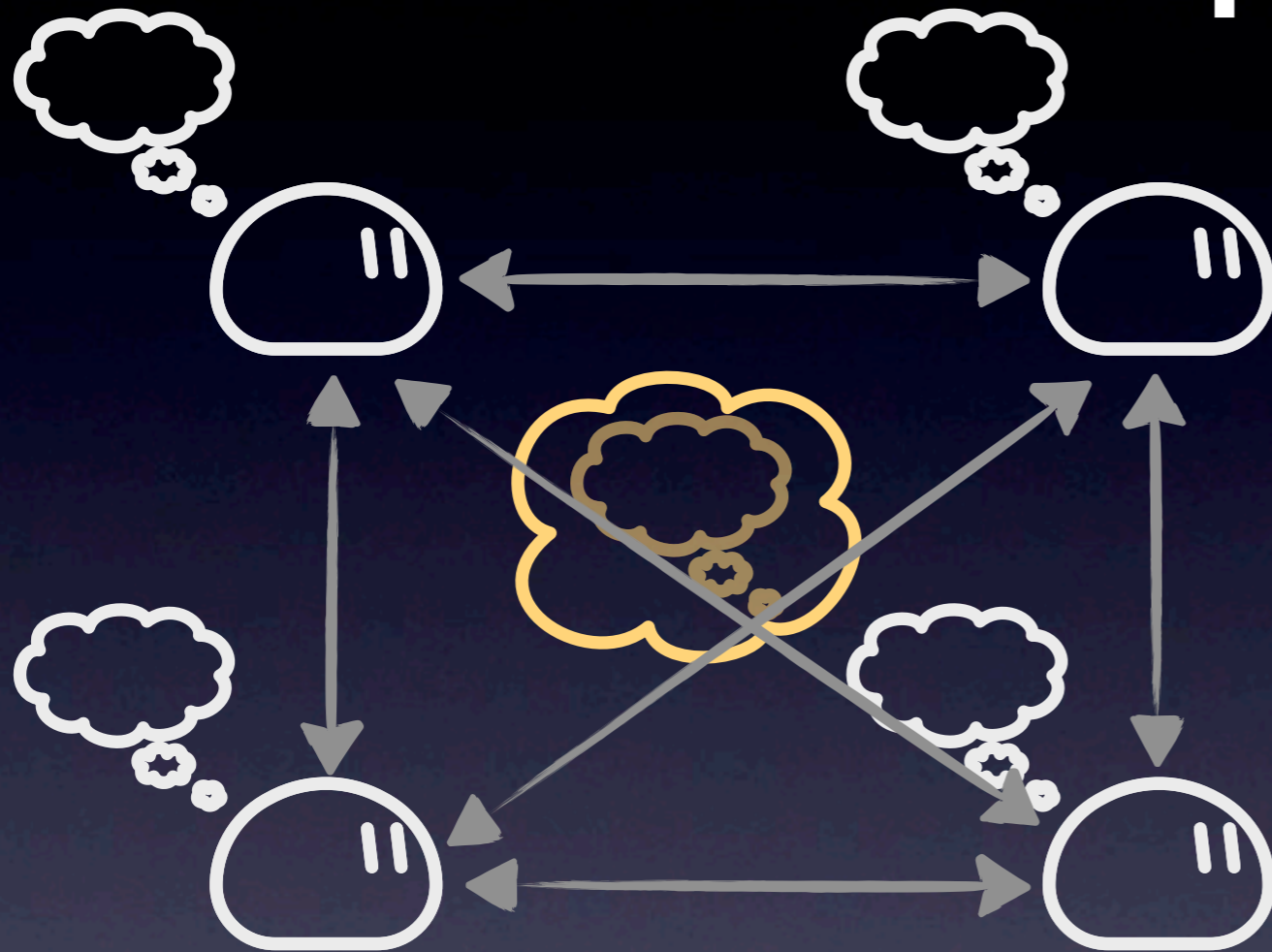


Humble Workers



Snapshot

Checkpoint



Humble Workers



Snapshot

Checkpoint



Humble Workers

Master

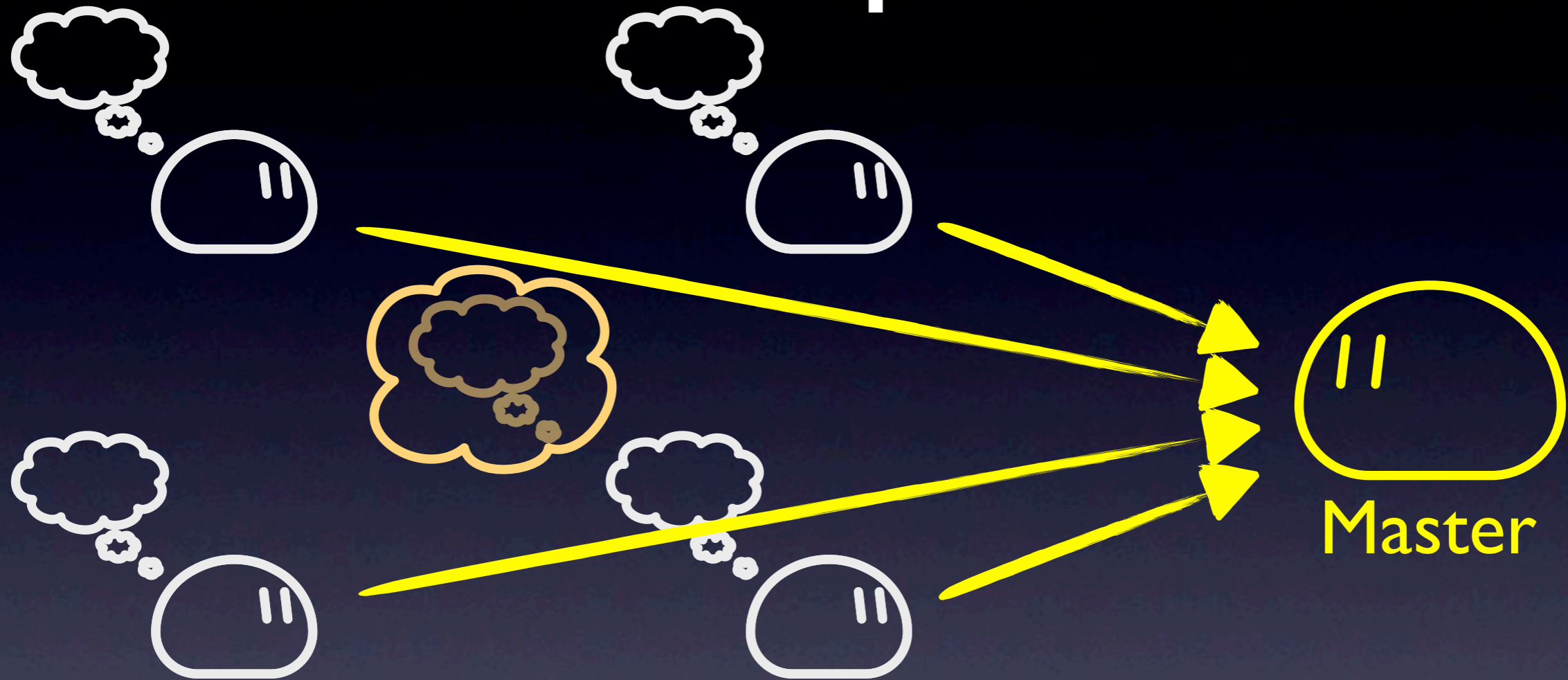
Log Ops

Checkpoint



Humble Workers

Checkpoint



Humble Workers

Master

Finish Checkpoint

Checkpoint



Humble Workers

Outline

- Background
- Intuition
- Design
- Evaluation
- Future Work

Scaling - Speedup

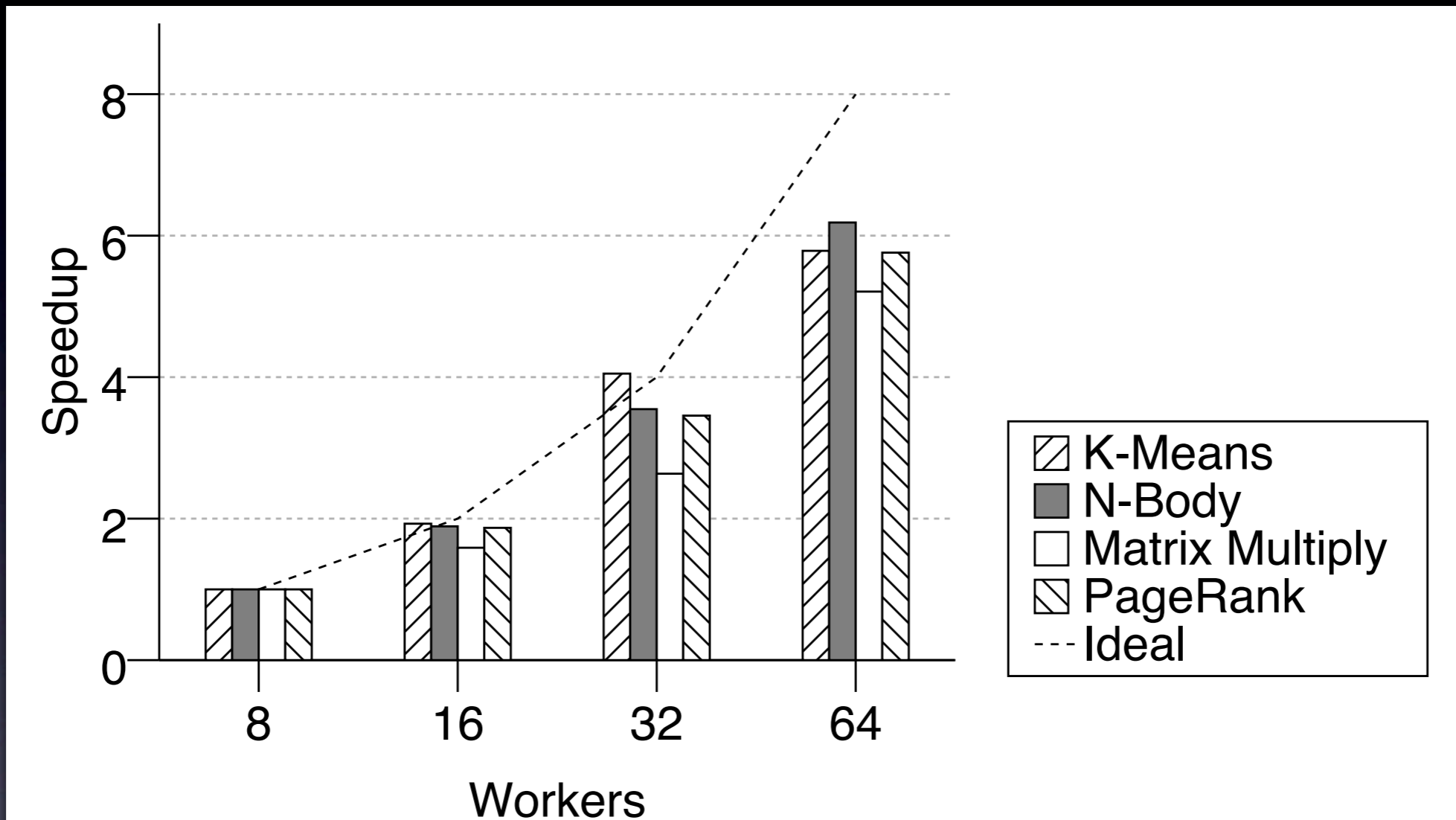


Figure 6: Scaling performance (fixed default input size)

Scaling - Input

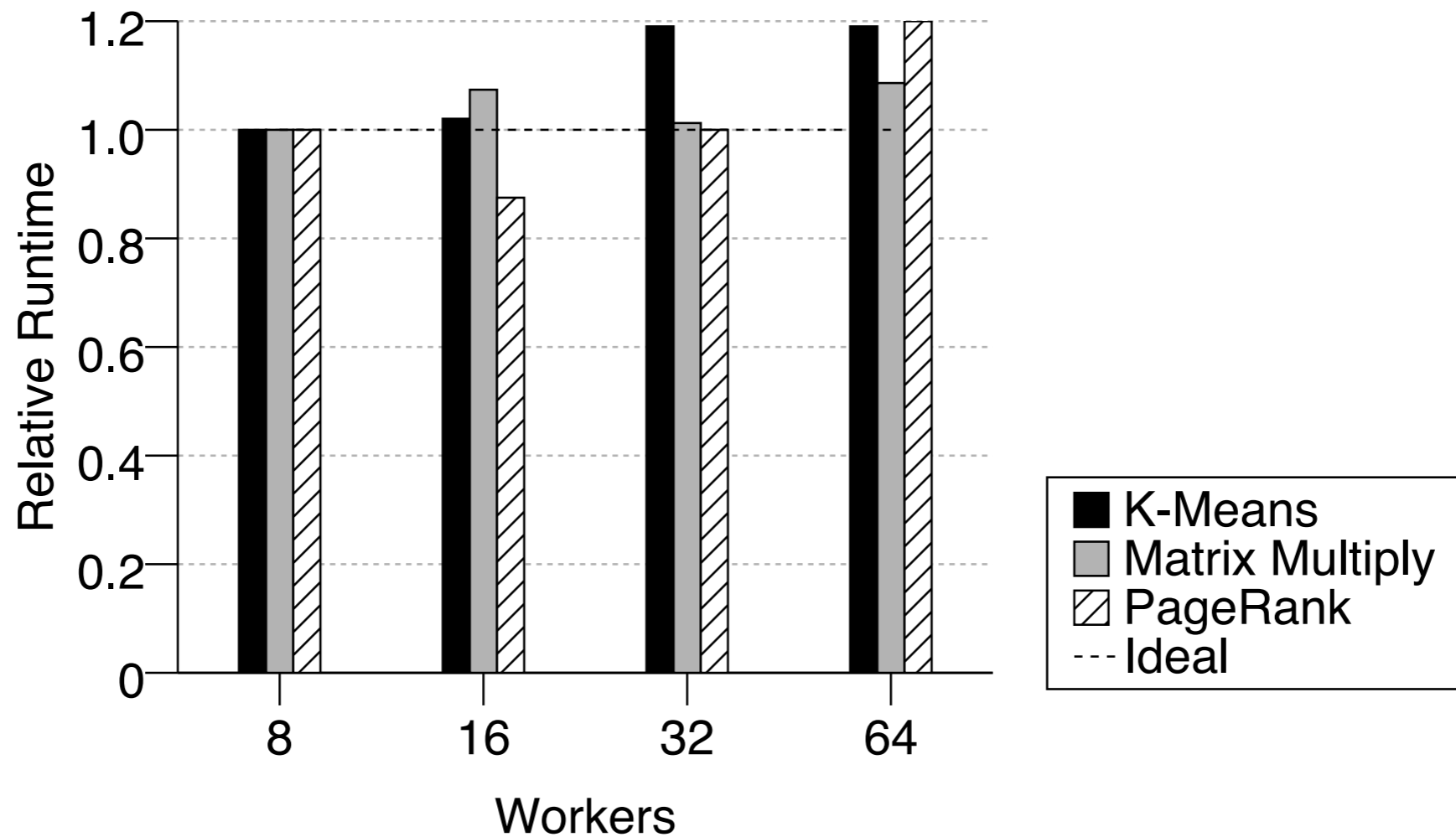


Figure 7: Scaling input size.

Scaling - Input (cont.)

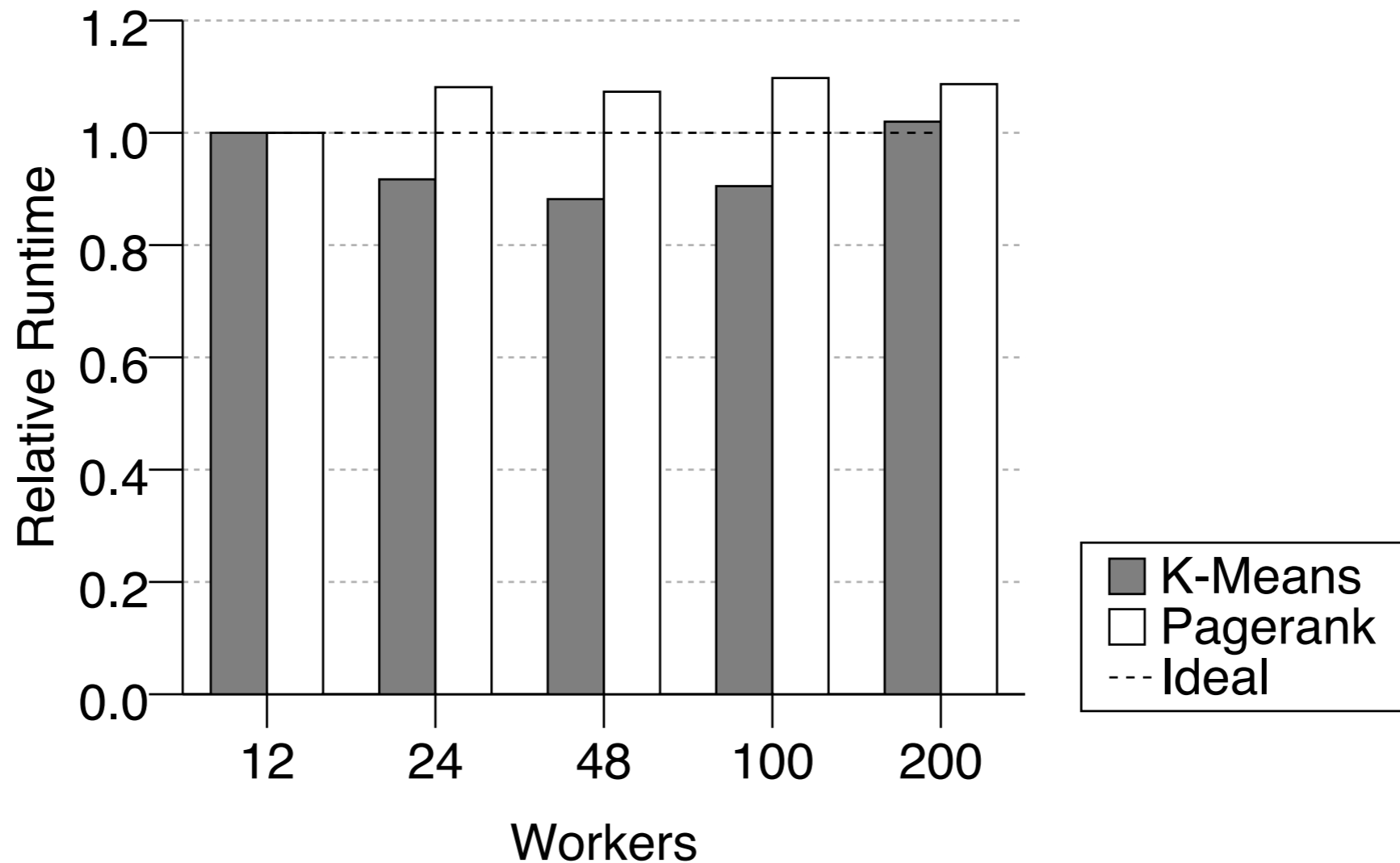


Figure 8: Scaling input size on EC2.

Comparison with MapReduce on Hadoop

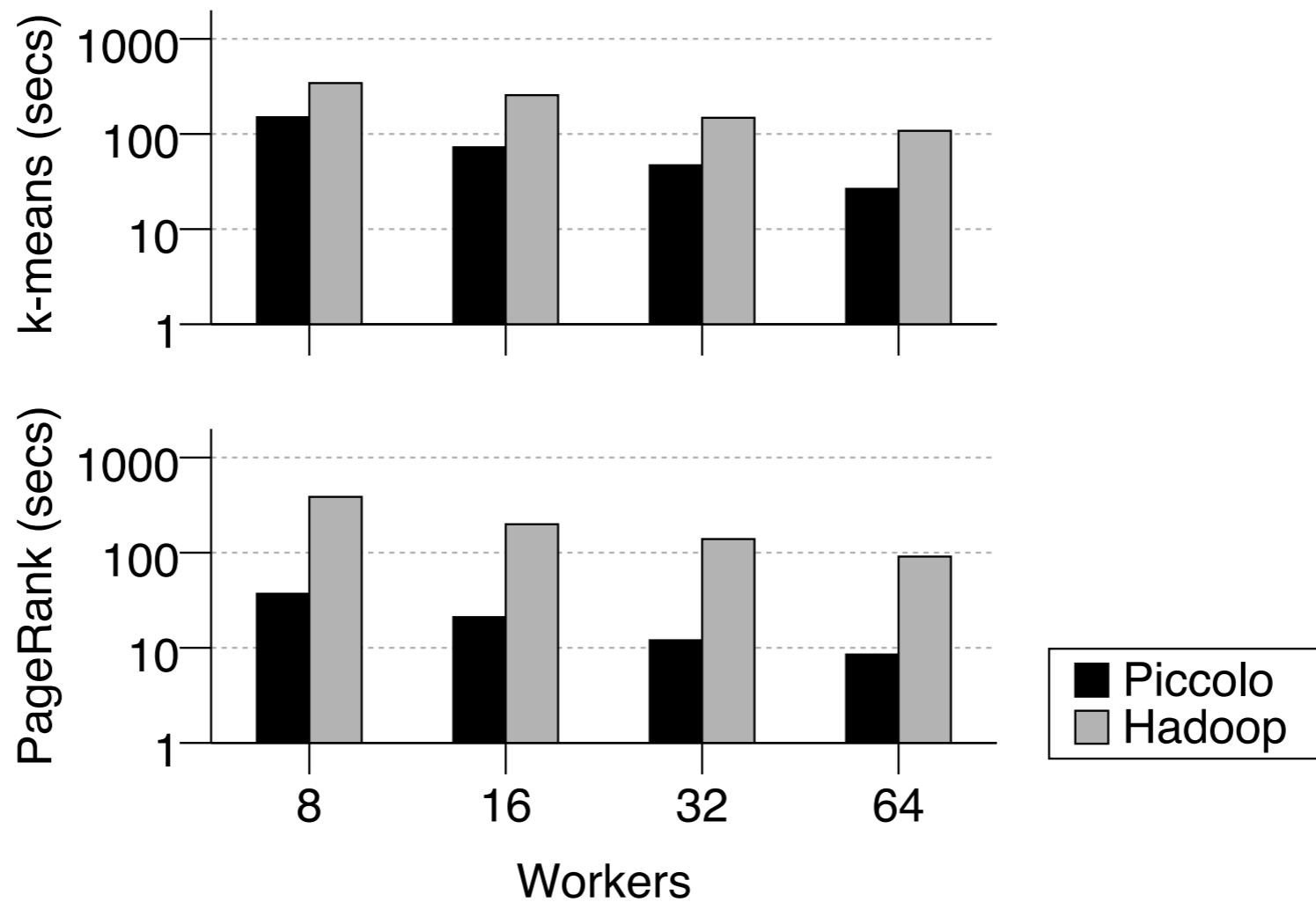


Figure 9: Per-iteration running time of PageRank and k -means in Hadoop and Piccolo (fixed default input size).

Comparison with MPI

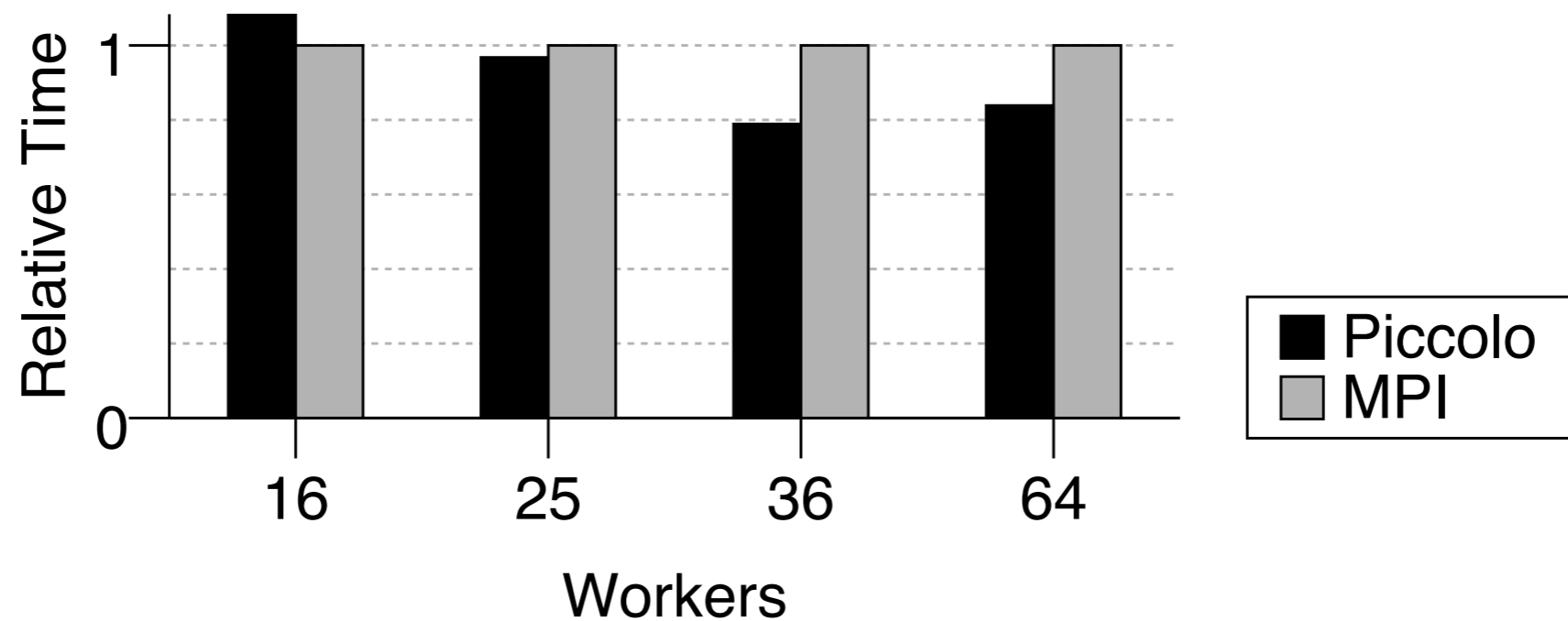


Figure 10: Runtime of matrix multiply, scaled relative to MPI.

Load Balance

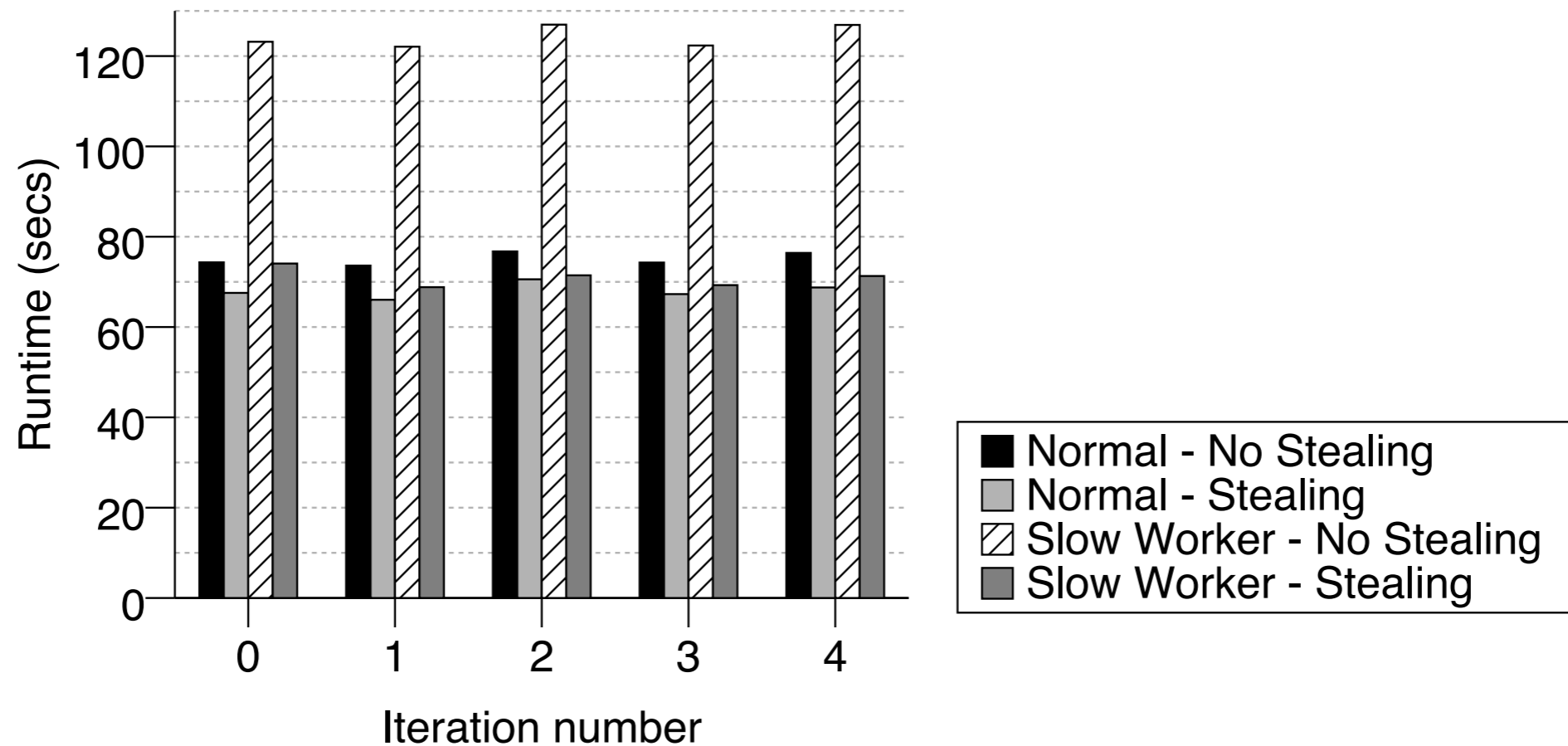


Figure 11: Effect of Work Stealing and Slow Workers

Outline

- Background
- Intuition
- Design
- Evaluation
- **Future Work**

Future Work

- Log-based scalable failure handling
- More user-defined accumulator per table
- **Distributed as Parallel**

Thanks ~.~