E2E Arguments & Project Suggestions (Lecture 4, cs262a)

Ion Stoica,
UC Berkeley
September 7, 2016

Software Modularity

Break system into modules:

Well-defined interfaces gives flexibility

- Change implementation of modules
- Extend functionality of system by adding new modules

Interfaces hide information

- Allows for flexibility
- But can hurt performance

Network Modularity

Like software modularity, but with a twist:

Implementation distributed across routers and hosts

Must decide:

- How to break system into modules
- Where modules are implemented

Layering

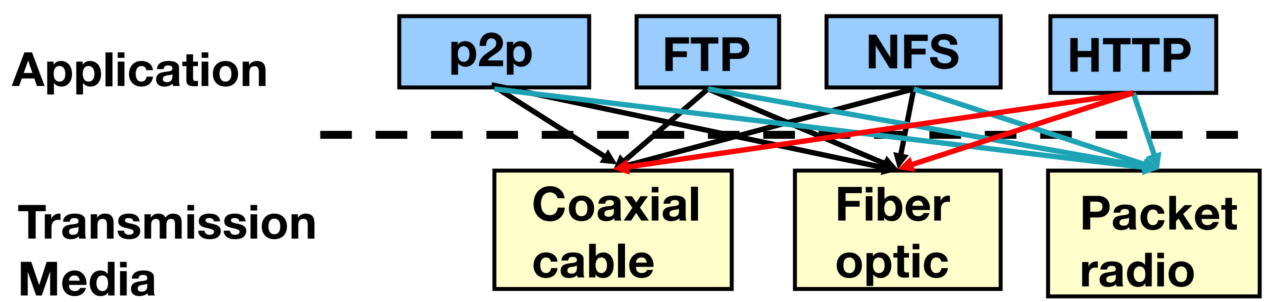
Layering is a particular form of modularization

System is broken into a vertical hierarchy of logically distinct entities (layers)

Service provided by one layer is based solely on the service provided by layer below

Rigid structure: easy reuse, performance suffers

The Problem



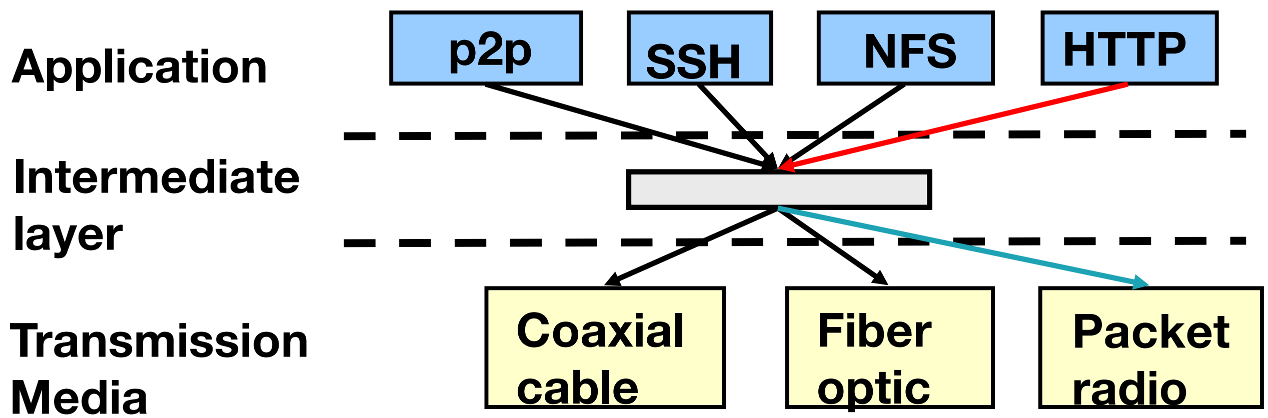
Re-implement every application for every technology?

No! But how does the Internet architecture avoid this?

Solution: Intermediate Layer

Introduce an intermediate layer that provides a single abstraction for various network technologies

- A new app/media implemented only once
- Variation on “add another level of indirection”



Placing Functionality

Most influential paper about placing functionality is “End-to-End Arguments in System Design” by Saltzer, Reed, and Clark

“Sacred Text” of the Internet

- Endless disputes about what it means
- Everyone cites it as supporting their position

Basic Observation

Some applications have end-to-end performance requirements

- Reliability, security, etc

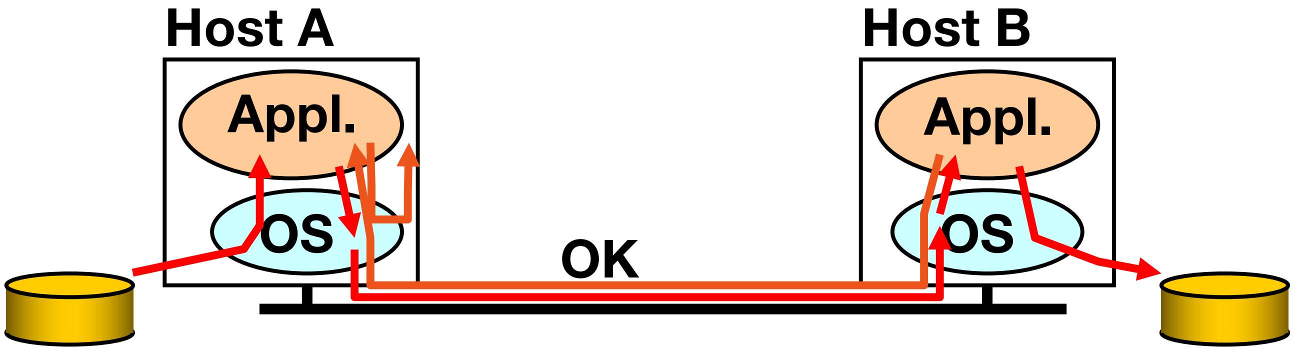
Implementing these in the network is very hard:

- Every step along the way must be fail-proof

Hosts:

- Can satisfy the requirement without the network
- Can't depend on the network

Example: Reliable File Transfer



Solution 1: make each step reliable, and then concatenate them

Solution 2: end-to-end check and retry

Discussion

Solution 1 not complete

- What happens if any network element misbehaves?
- Receiver has to do the check anyway!

Solution 2 is complete

- Full functionality can be entirely implemented at application layer with no need for reliability from lower layers

Is there any need to implement reliability at lower layers?

Take Away

Implementing this functionality in the network:

- Doesn't reduce host implementation complexity
- Does increase network complexity
- Probably imposes delay and overhead on all applications, even if they don't need functionality

However, implementing in network can enhance performance in some cases

- E.g., very lossy link

Conservative Interpretation

“Don’t implement a function at the lower levels of the system unless it can be completely implemented at this level”

Unless you can relieve the burden from hosts, then don’t bother

Radical Interpretation

Don't implement anything in the network that can be implemented correctly by the hosts

- E.g., multicast

Make network layer absolutely minimal

- Ignore performance issues

Moderate Interpretation

Think twice before implementing functionality in the network

If hosts can implement functionality correctly, implement it a lower layer only as a performance enhancement

But do so only if it does not impose burden on applications that do not require that functionality

Summary

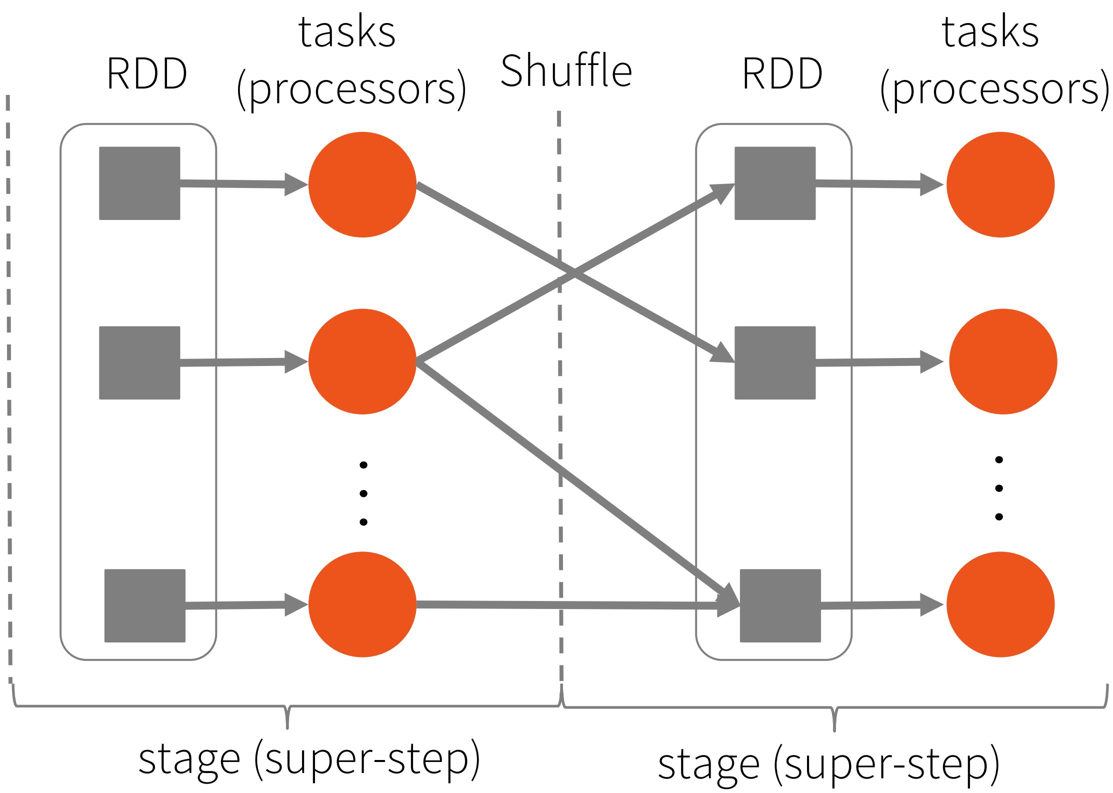
Layering is a good way to organize systems (e.g., networks)

Unified Internet layer decouples apps from networks

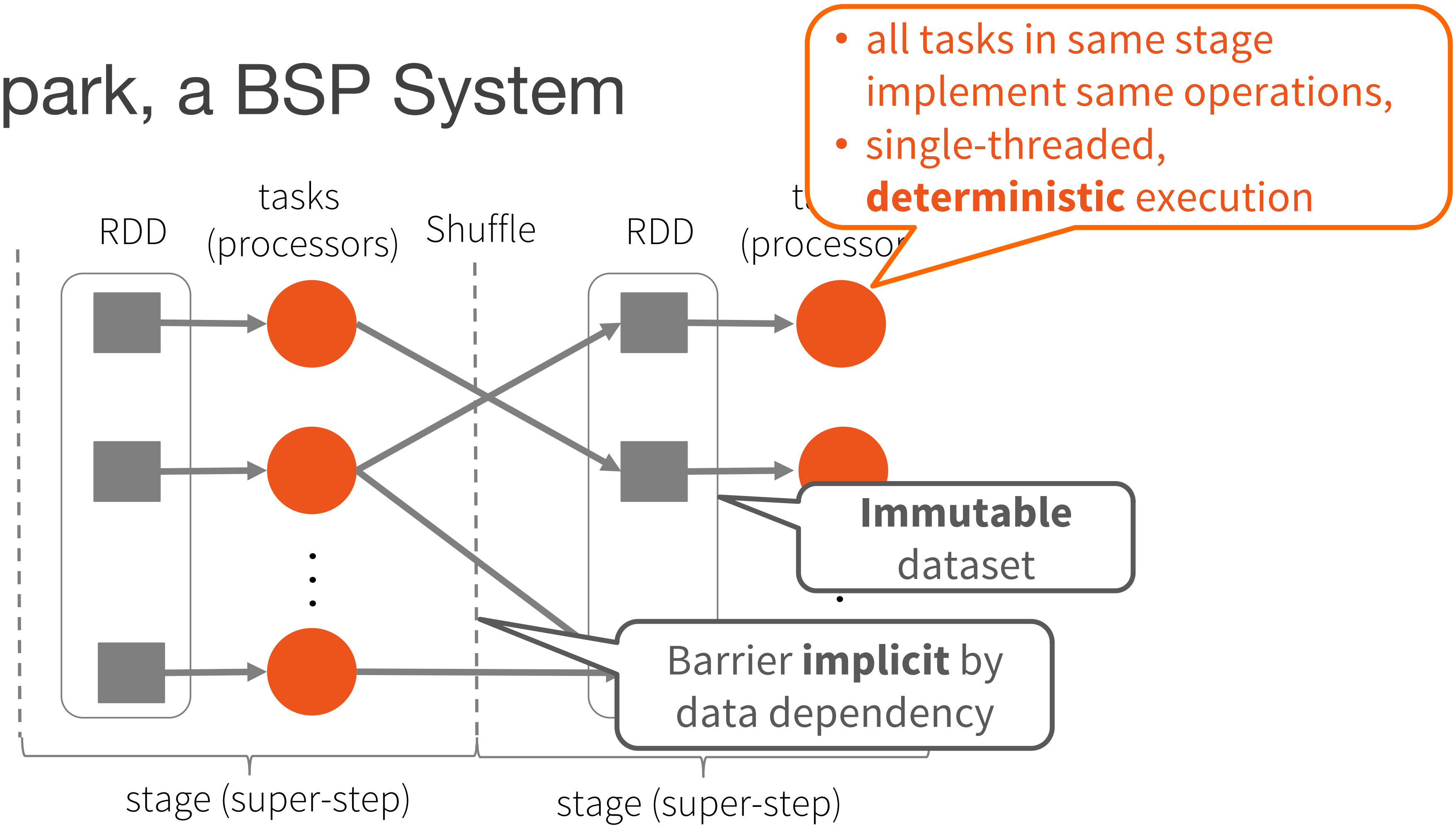
E2E argument encourages us to keep lower layers (e.g., IP) simple

Projects Suggestions

Spark, a BSP System



Spark, a BSP System



Scheduling for Heterogeneous Resources

Spark: assumes tasks are single-threaded

- One task per slot
- Typically, one slot per core

Challenge: a task my call a library that

- Is multithreaded
- Runs on other computation resources, GPUs

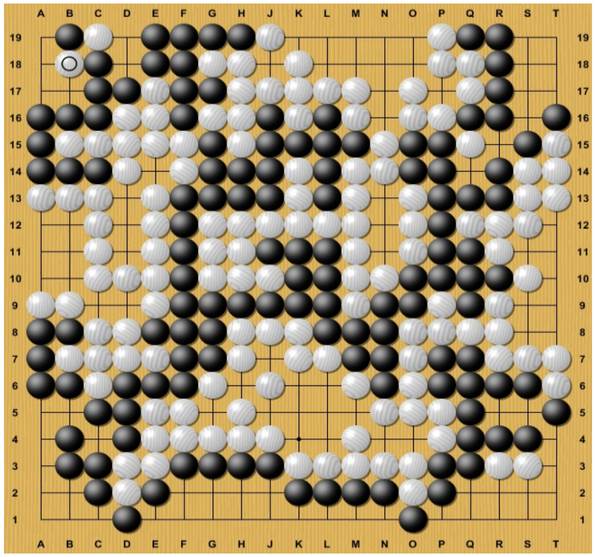
Generalize Spark's scheduling model

BSP Limitations

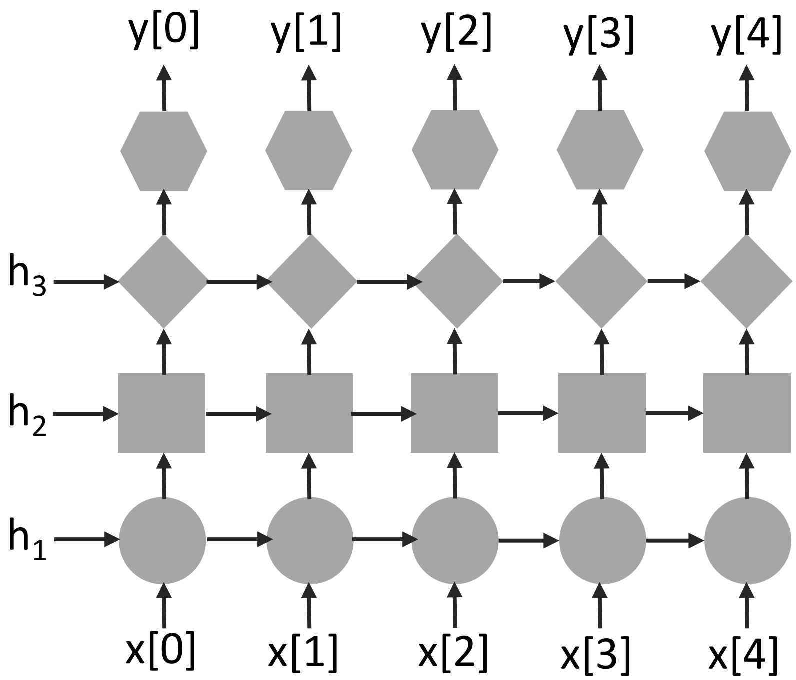
BSP, great for data parallel jobs

Not best fit for more complex computations

- Linear algebra algorithms (multiple inner loops)
- Some ML algorithms

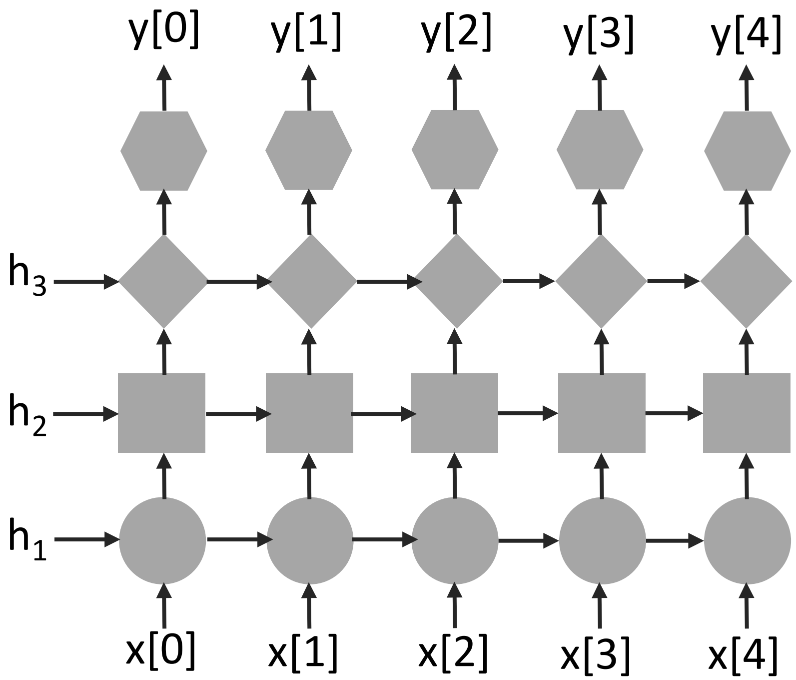


Example: Recurrent Neural Networks



- `x[t]`: input vector at time `t` (e.g., a frame in a video)
- `y[t]`: output at time `t` (e.g., a prediction about the activity in the video)
- h_l: initial hidden state for layer `l`

Example: Recurrent Neural Networks




```
for t in range(num_steps):
  h1 = rnn.first_layer(x[t], h1)
  h2 = rnn.second_layer(h1, h2)
  h3 = rnn.third_layer(h2, h3)
  y  = rnn.fourth_layer(h3)
```

Example: Recurrent Neural Networks

h_3

h_2

h_1

x[0] x[1] x[2]

```
for t in range(num_steps):
  h1 = rnn.first_layer(x[t], h1)
  h2 = rnn.second_layer(h1, h2)
  h3 = rnn.third_layer(h2, h3)
  y  = rnn.fourth_layer(h3)
```

```
t = 0
```

Example: Recurrent Neural Networks

h_3

h_2

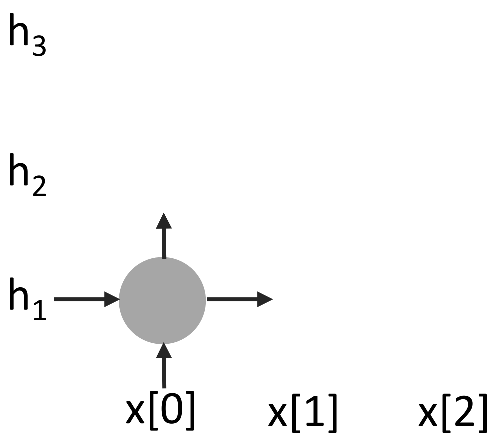
h_1

x[0] x[1] x[2]

```
for t in range(num_steps):
  h1 = rnn.first_layer(x[t], h1)
  h2 = rnn.second_layer(h1, h2)
  h3 = rnn.third_layer(h2, h3)
  y  = rnn.fourth_layer(h3)
```

```
t = 0
```

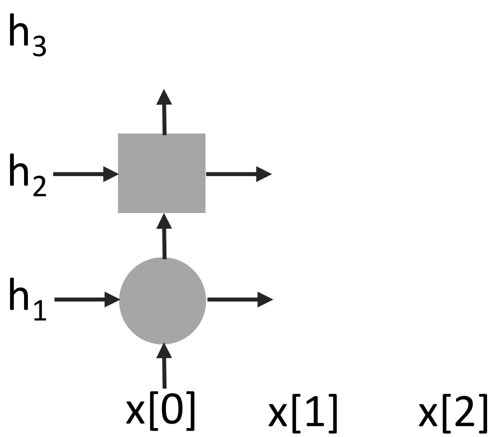
Example: Recurrent Neural Networks




```
for t in range(num_steps):
> h1 = rnn.first_layer(x[t], h1)
  h2 = rnn.second_layer(h1, h2)
  h3 = rnn.third_layer(h2, h3)
  y  = rnn.fourth_layer(h3)


  t = 0
```

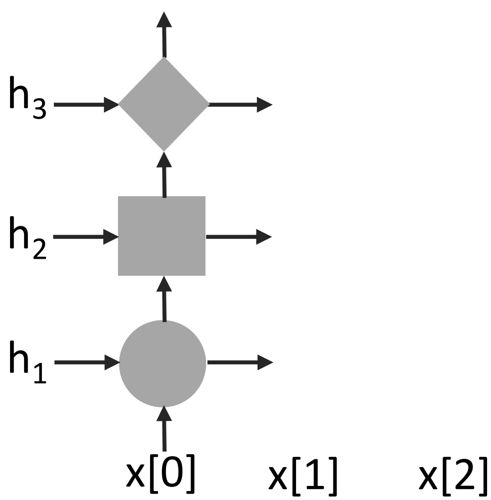
Example: Recurrent Neural Networks




```
for t in range(num_steps):
  h1 = rnn.first_layer(x[t], h1)
> h2 = rnn.second_layer(h1, h2)
  h3 = rnn.third_layer(h2, h3)
  y  = rnn.fourth_layer(h3)
```

t = 0

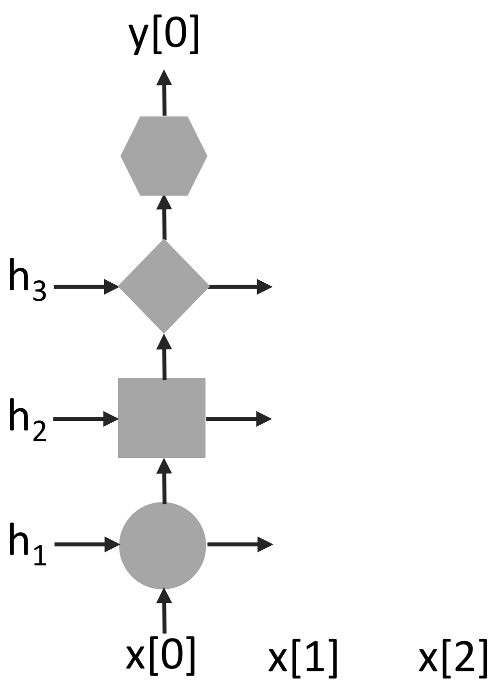
Example: Recurrent Neural Networks




```
for t in range(num_steps):
  h1 = rnn.first_layer(x[t], h1)
  h2 = rnn.second_layer(h1, h2)
> h3 = rnn.third_layer(h2, h3)
  y  = rnn.fourth_layer(h3)
```

```
t = 0
```

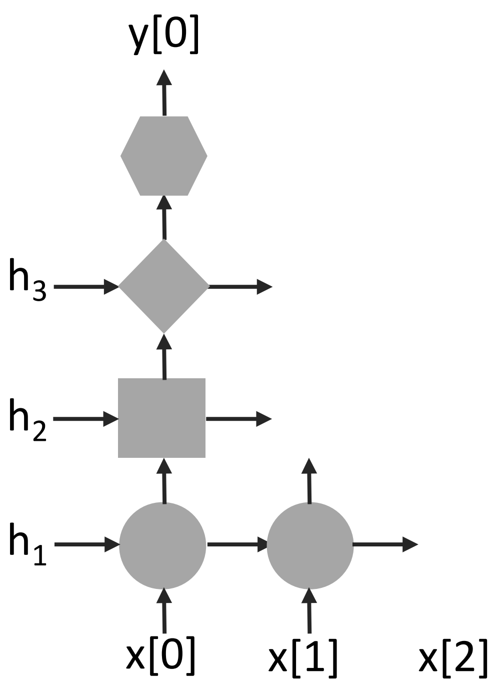
Example: Recurrent Neural Networks




```
for t in range(num_steps):
  h1 = rnn.first_layer(x[t], h1)
  h2 = rnn.second_layer(h1, h2)
  h3 = rnn.third_layer(h2, h3)
> y  = rnn.fourth_layer(h3)
```

```
  t = 0
```

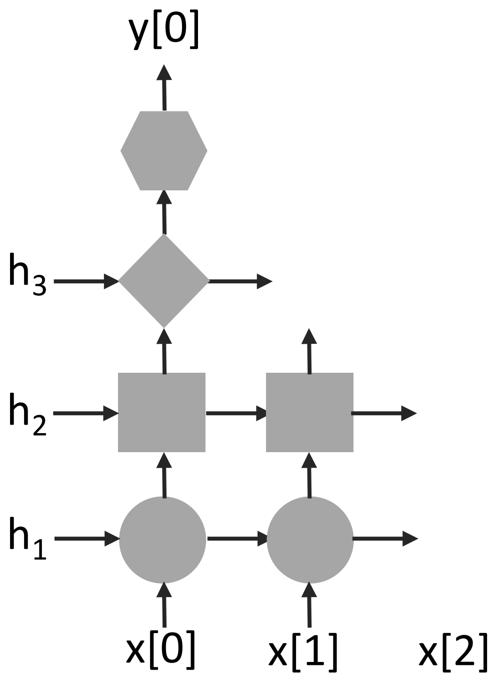
Example: Recurrent Neural Networks




```
for t in range(num_steps):
> h1 = rnn.first_layer(x[t], h1)
  h2 = rnn.second_layer(h1, h2)
  h3 = rnn.third_layer(h2, h3)
  y  = rnn.fourth_layer(h3)


  t = 1
```

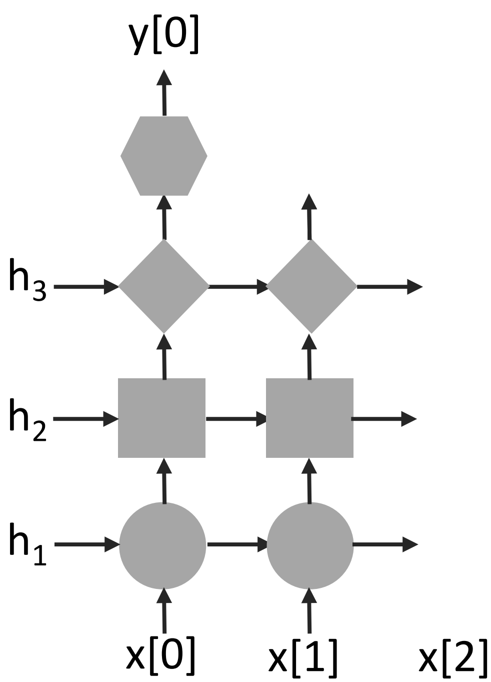
Example: Recurrent Neural Networks




```
for t in range(num_steps):
  h1 = rnn.first_layer(x[t], h1)
> h2 = rnn.second_layer(h1, h2)
  h3 = rnn.third_layer(h2, h3)
  y  = rnn.fourth_layer(h3)
```

t = 1

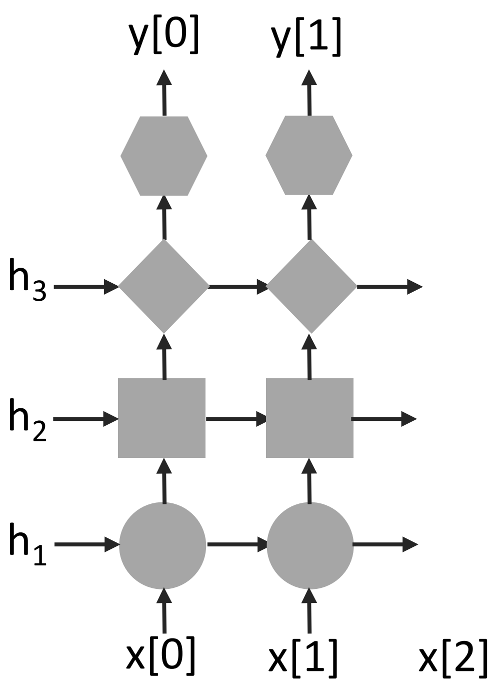
Example: Recurrent Neural Networks




```
for t in range(num_steps):
  h1 = rnn.first_layer(x[t], h1)
  h2 = rnn.second_layer(h1, h2)
> h3 = rnn.third_layer(h2, h3)
  y  = rnn.fourth_layer(h3)
```

t = 1

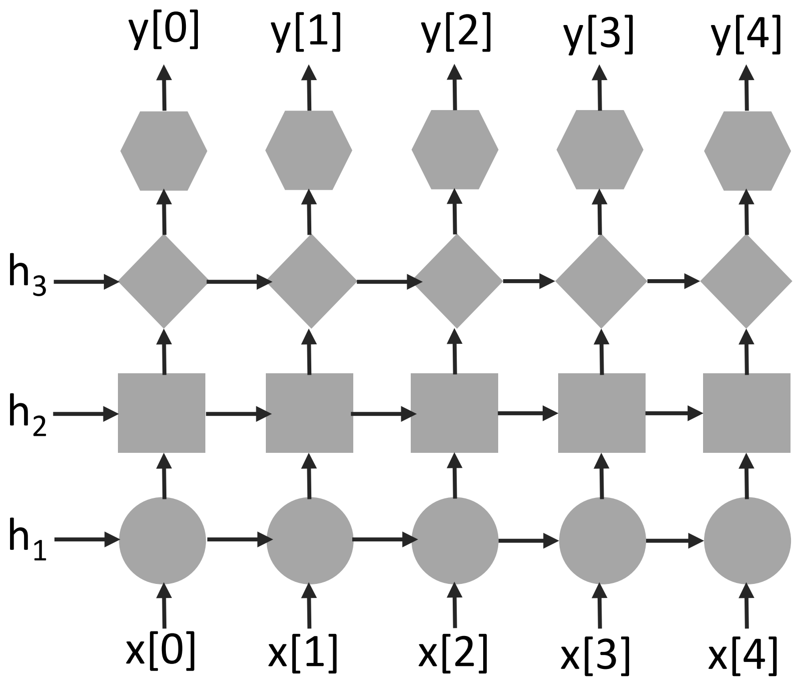
Example: Recurrent Neural Networks




```
for t in range(num_steps):
  h1 = rnn.first_layer(x[t], h1)
  h2 = rnn.second_layer(h1, h2)
  h3 = rnn.third_layer(h2, h3)
> y  = rnn.fourth_layer(h3)
```

t = 1

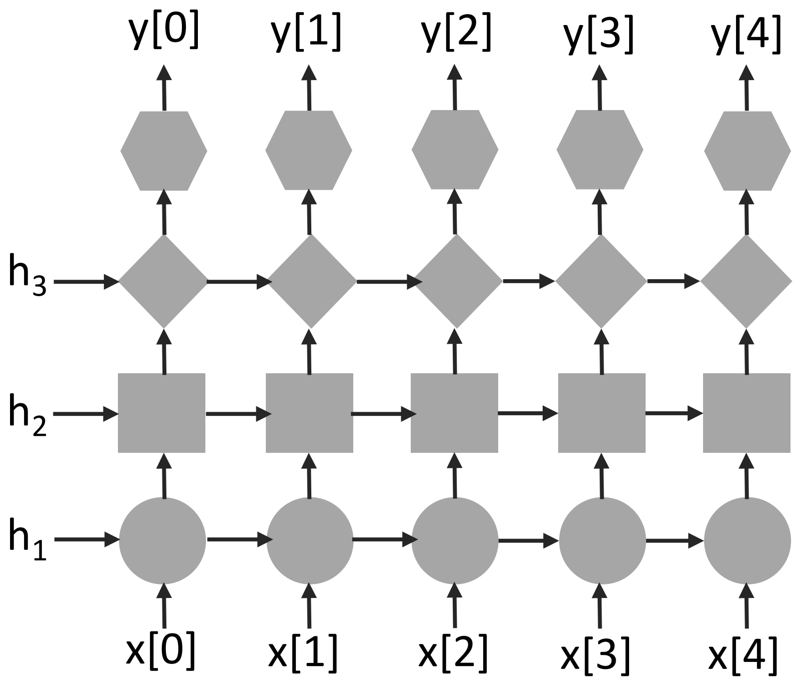
Example: Recurrent Neural Networks



- `x[t]`: input vector at time `t` (e.g., a frame in a video)
- `y[t]`: output output at time `t` (e.g., a prediction about the activity in the video)
- h_1: initial hidden state for layer `l`

blue - task completed
red - task running
- dependence ready
- dependence unready

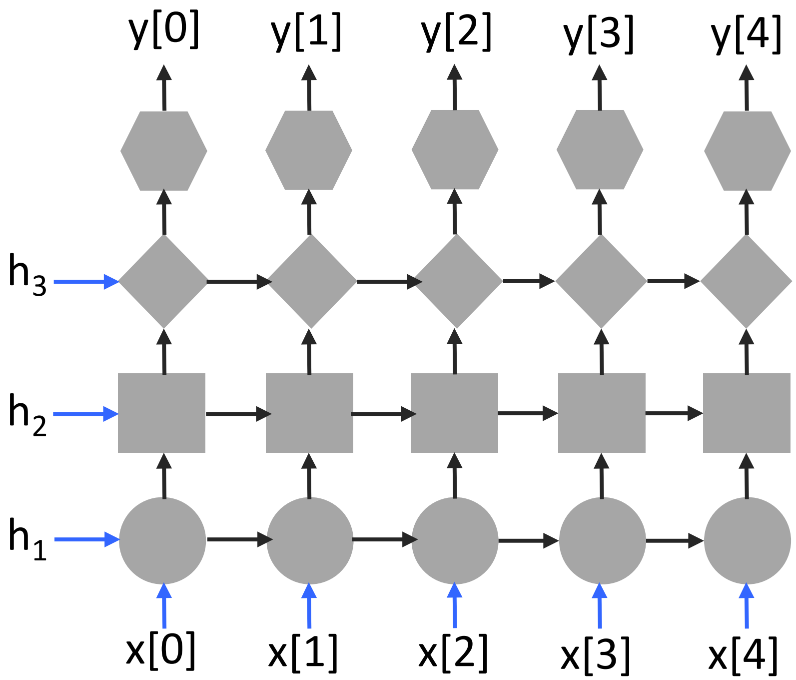
Example: Recurrent Neural Networks



- `x[t]`: input vector at time `t` (e.g., a frame in a video)
- `y[t]`: output at time `t` (e.g., a prediction about the activity in the video)
- h_1: initial hidden state for layer `1`

blue - task completed
red - task running
→ - dependence ready
→ - dependence unready

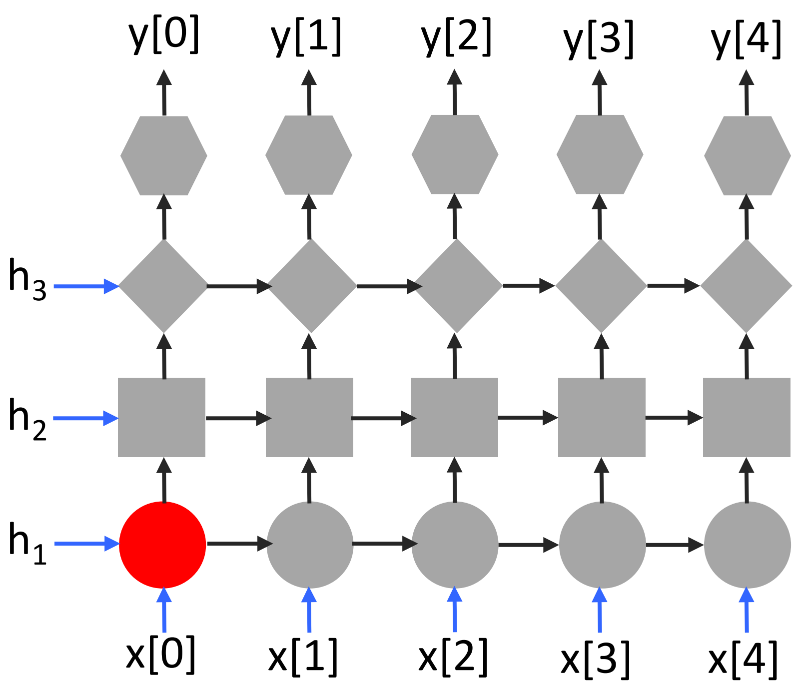
Example: Recurrent Neural Networks



- `x[t]`: input vector at time `t` (e.g., a frame in a video)
- `y[t]`: output at time `t` (e.g., a prediction about the activity in the video)
- h_1: initial hidden state for layer `1`

blue - task completed
red - task running
- dependence ready
- dependence unready

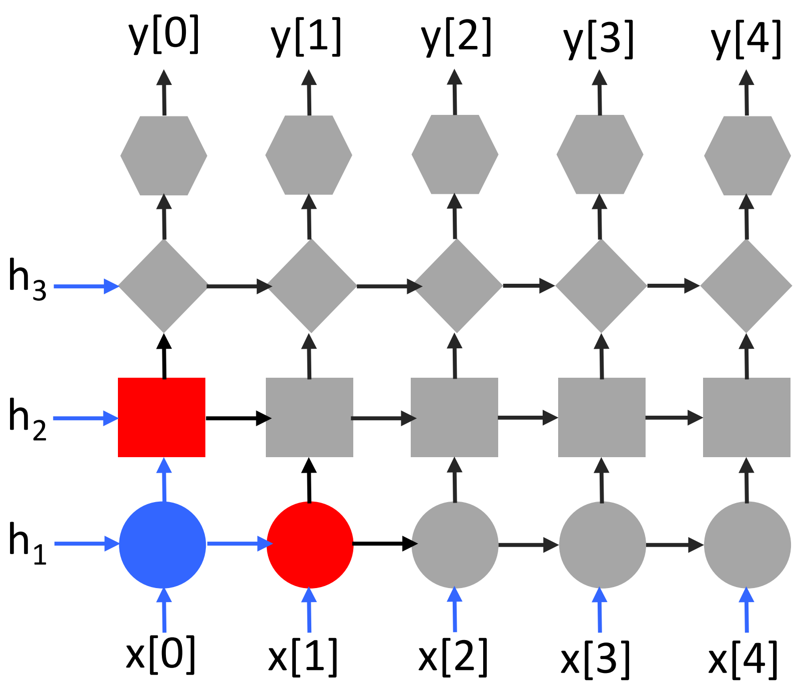
Example: Recurrent Neural Networks



- `x[t]`: input vector at time `t` (e.g., a frame in a video)
- `y[t]`: output at time `t` (e.g., a prediction about the activity in the video)
- `h1`: initial hidden state for layer `l`

blue - task completed
red - task running
- dependence ready
- dependence unready

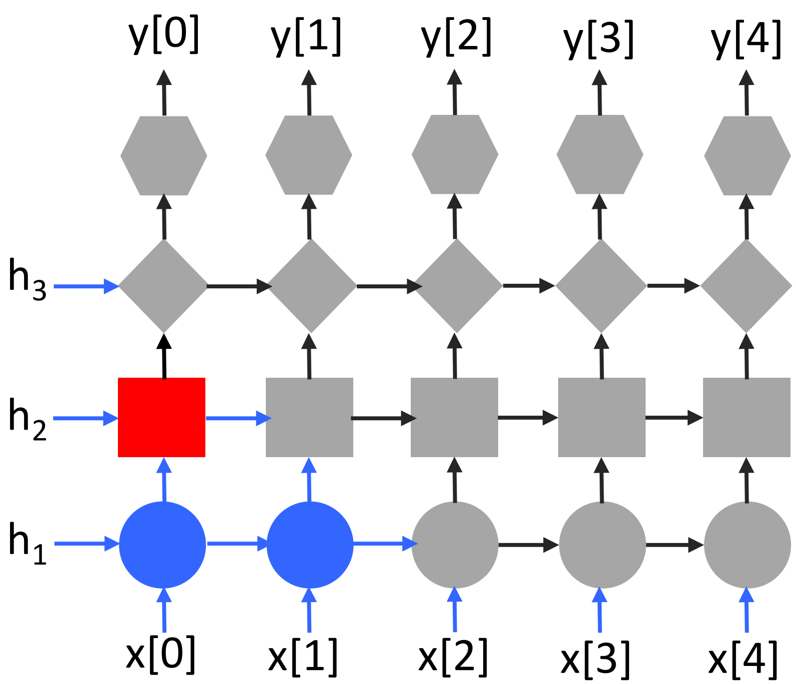
Example: Recurrent Neural Networks



- `x[t]`: input vector at time `t` (e.g., a frame in a video)
- `y[t]`: output at time `t` (e.g., a prediction about the activity in the video)
- h_1: initial hidden state for layer `1`

blue - task completed
red - task running
- dependence ready
- dependence unready

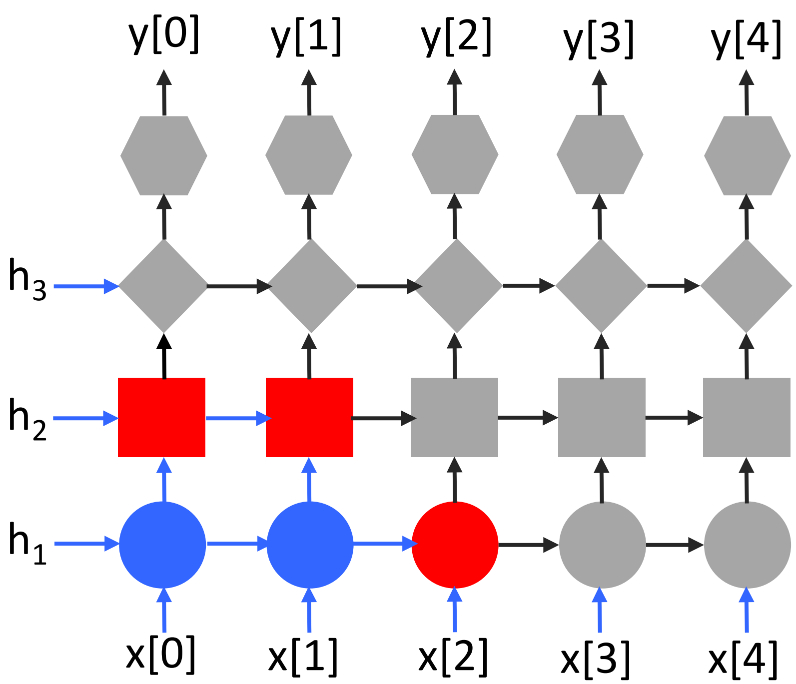
Example: Recurrent Neural Networks



- `x[t]`: input vector at time `t` (e.g., a frame in a video)
- `y[t]`: output at time `t` (e.g., a prediction about the activity in the video)
- $\texttt{h}_\texttt{l}$`:` initial hidden state for layer `l`

blue - task completed
red - task running
→ - dependence ready
→ - dependence unready

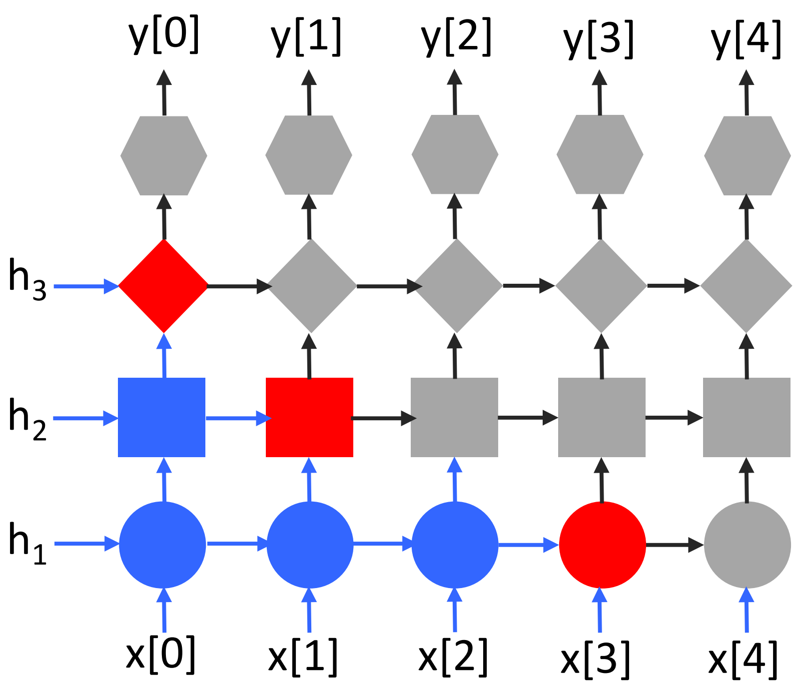
Example: Recurrent Neural Networks



- `x[t]`: input vector at time `t` (e.g., a frame in a video)
- `y[t]`: output at time `t` (e.g., a prediction about the activity in the video)
- h_1: initial hidden state for layer `1`

blue - task completed
red - task running
- dependence ready
- dependence unready

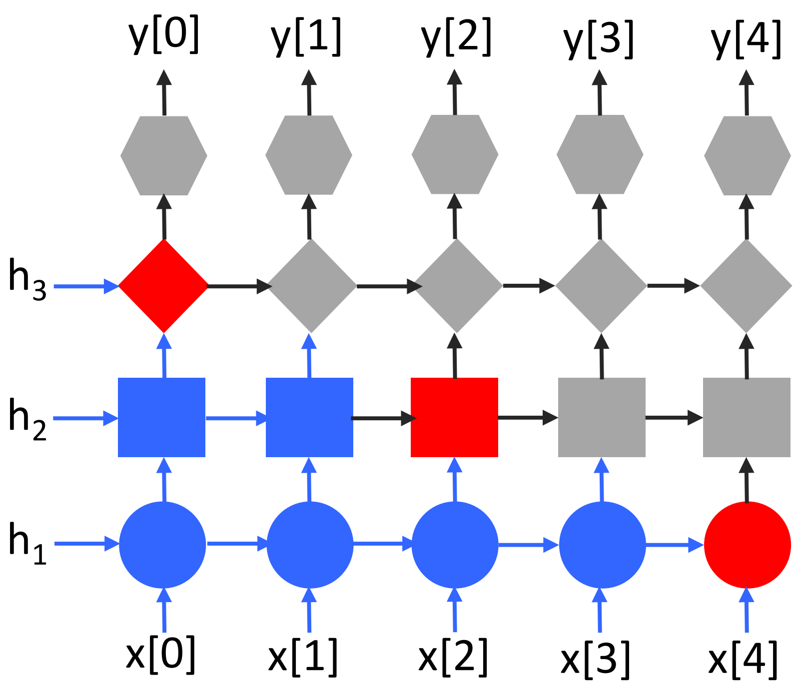
Example: Recurrent Neural Networks



- `x[t]`: input vector at time `t` (e.g., a frame in a video)
- `y[t]`: output at time `t` (e.g., a prediction about the activity in the video)
- h_1: initial hidden state for layer `l`

blue - task completed
red - task running
→ - dependence ready
→ - dependence unready

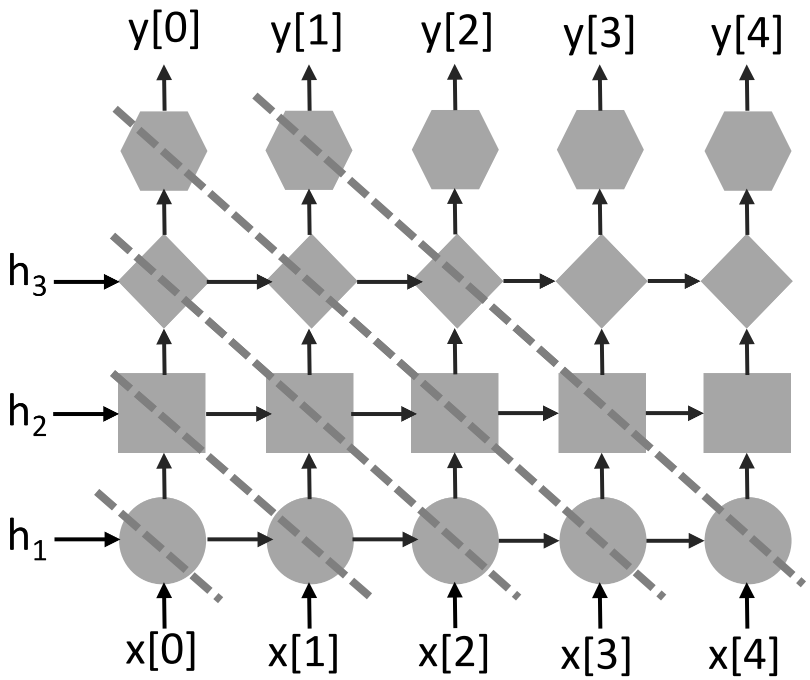
Example: Recurrent Neural Networks



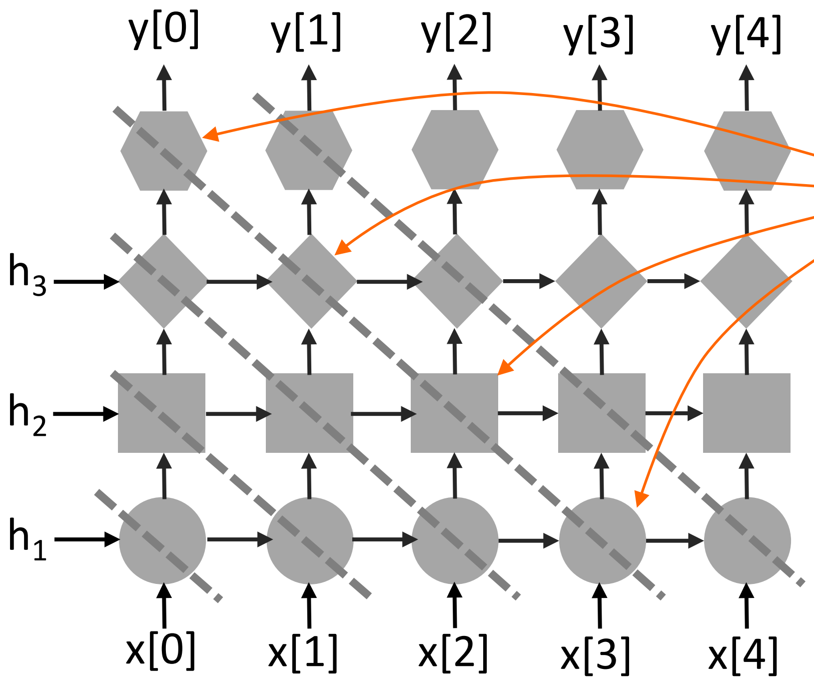
- `x[t]`: input vector at time `t` (e.g., a frame in a video)
- `y[t]`: output at time `t` (e.g., a prediction about the activity in the video)
- `h_l`: initial hidden state for layer `l`

blue - task completed
red - task running
→ - dependence ready
→ - dependence unready

How would BPS work?

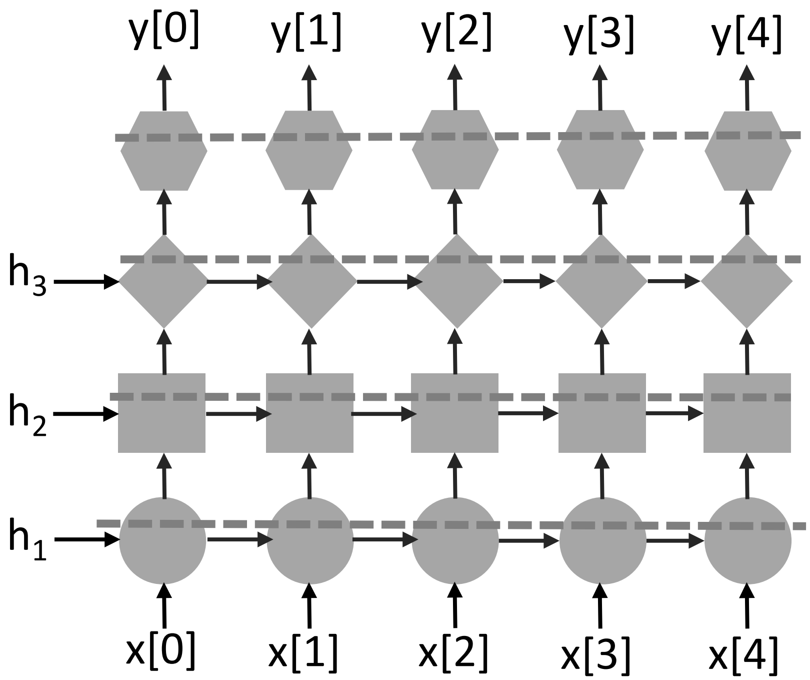


How would BPS work?

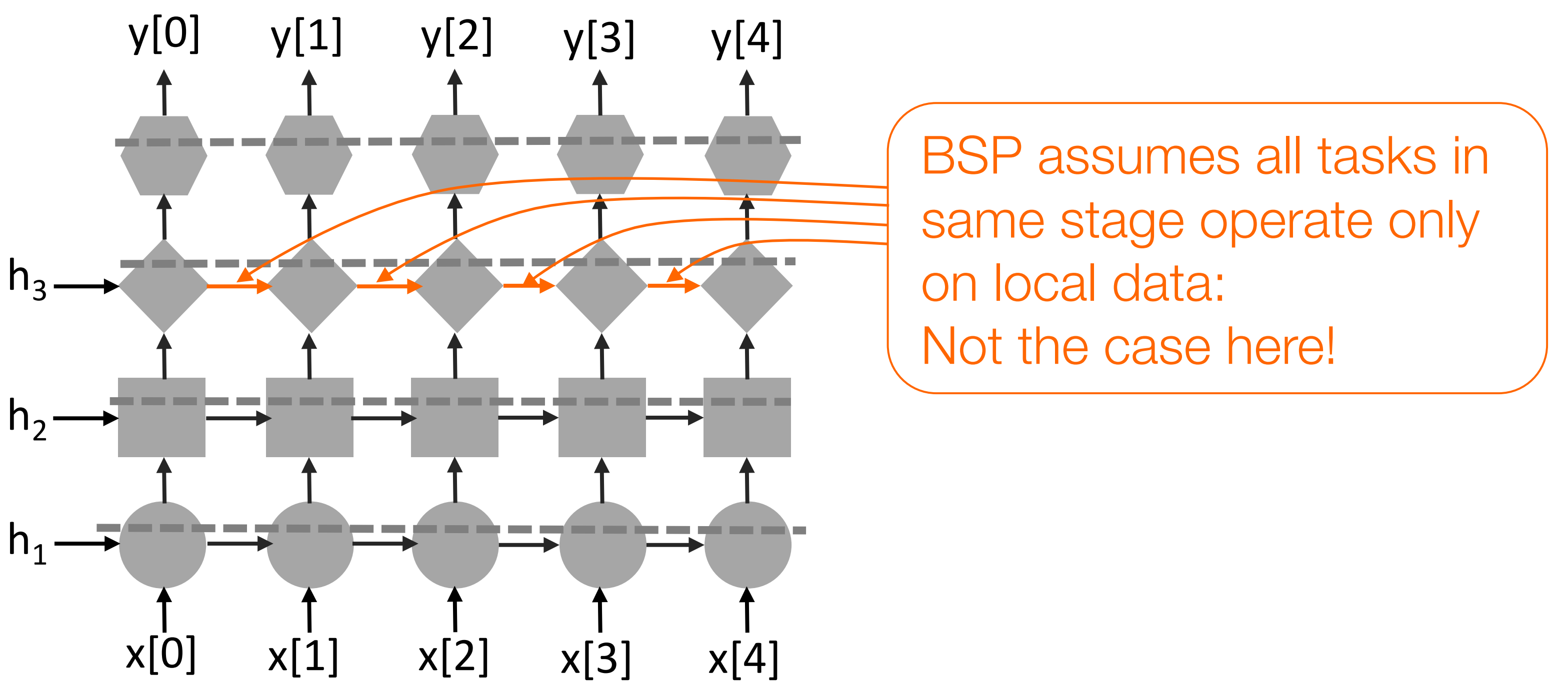


BSP assumes all tasks in same stage run same function: Not the case here!

How would BPS work?



How would BPS work?



Ray: Fine grained parallel execution engine

Goal: make it easier to parallelize Python programs, in particular ML algorithms

Python

```
add(a, b):
    return a + b

…
x = add(3, 4)
```

Ray

```
@ray.remote
add(a, b):
    return a + b

…
x_id = add.remote(3, 4)
x = ray.get(x_id)
```

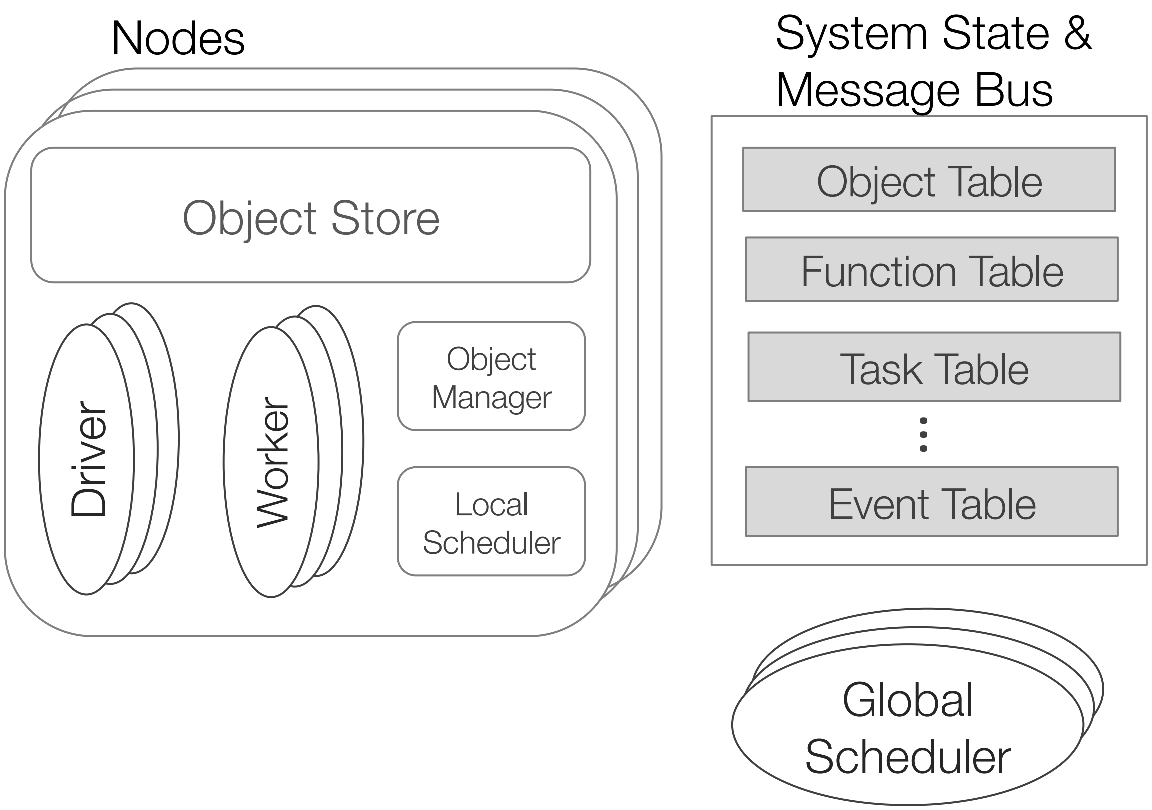
Another Example

```python
import ray

@ray.remote
def f(stepsize):
  # do computation…
  return result

# Run 4 experiments in parallel
results = [f.remote(stepsize) for stepsize in [0.001, 0.01, 0.1, 1.0]]
# Get the results
ray.get(results)
```

Ray Architecture



Driver: run a Ray program

Worker: execute Python functions (tasks)

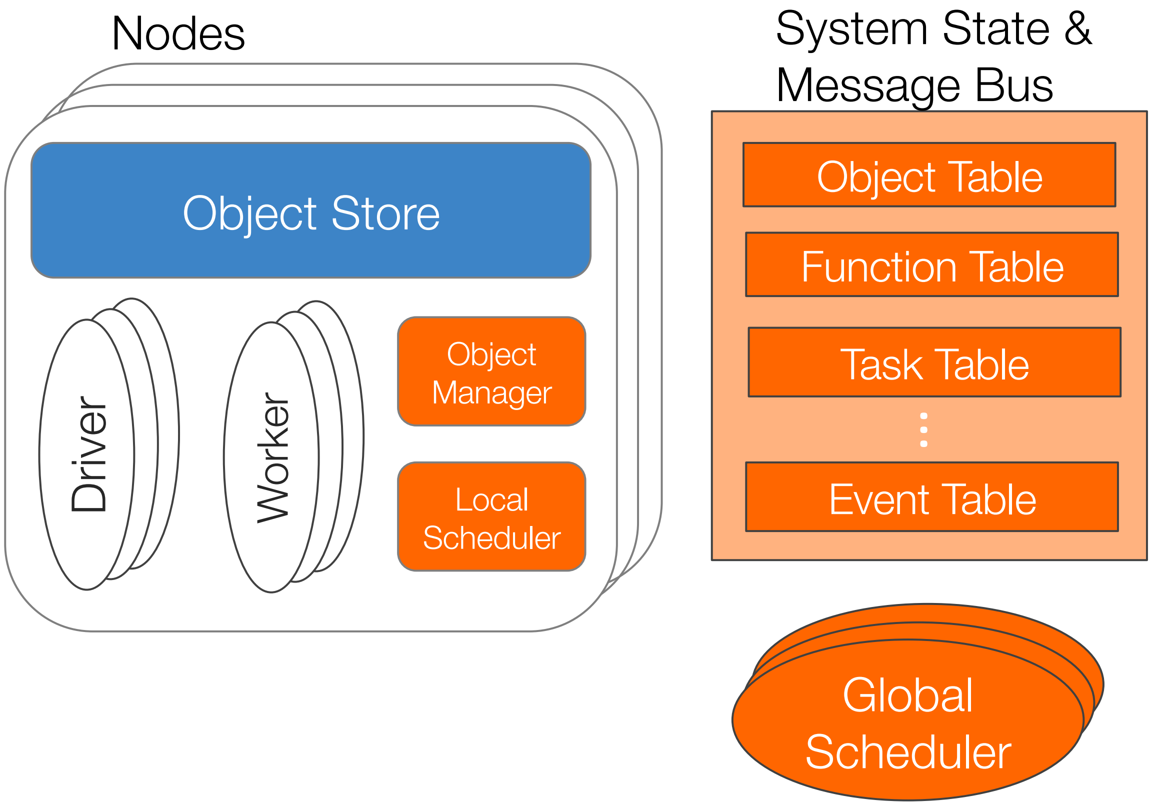
Object Store:
- Stores python objects
- Use shared memory on same node

Global scheduler: schedule tasks based on global state

Local scheduler: schedule tasks locally

System State & Msg Bus: store up-to-date state control state of entire system and relay events between components

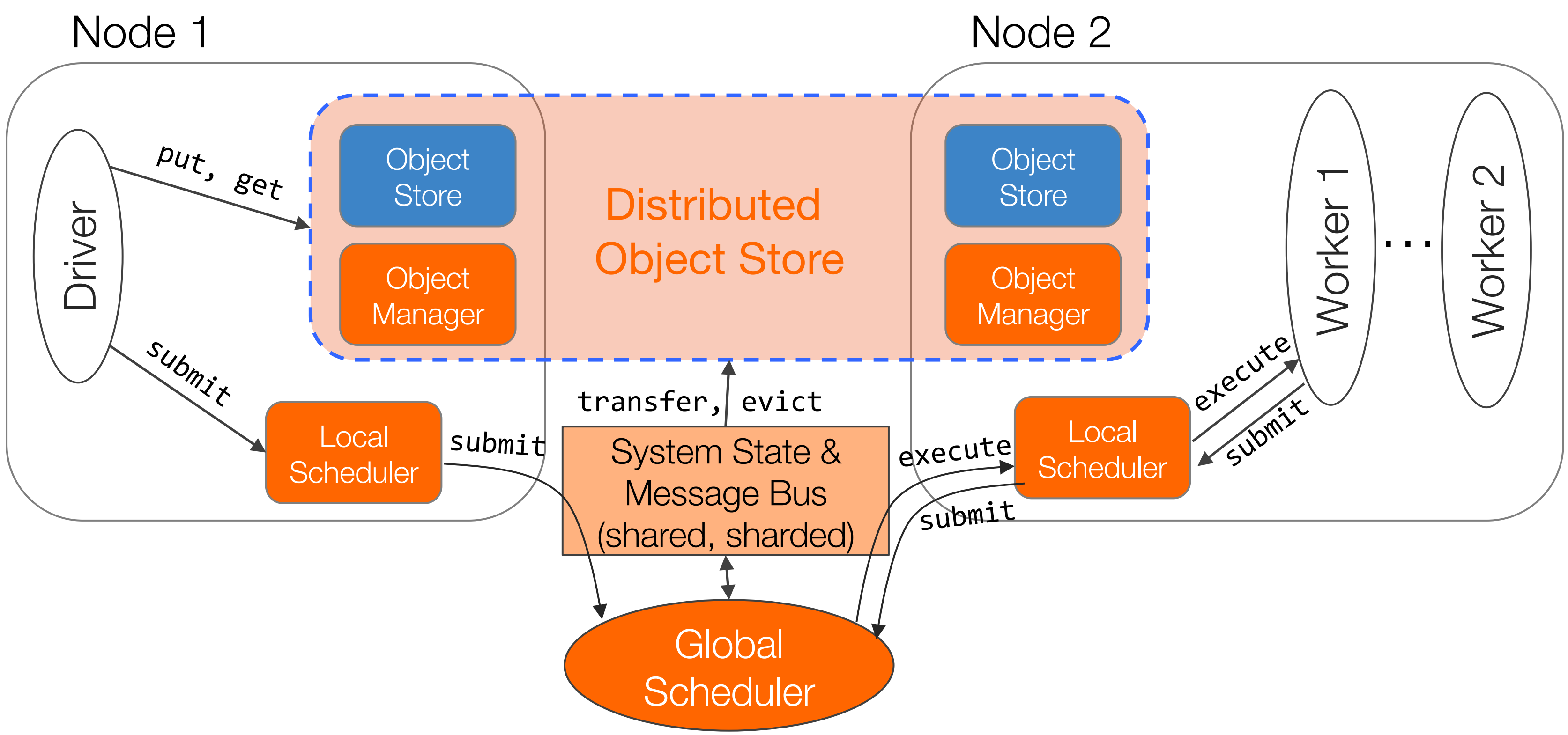
Ray Architecture



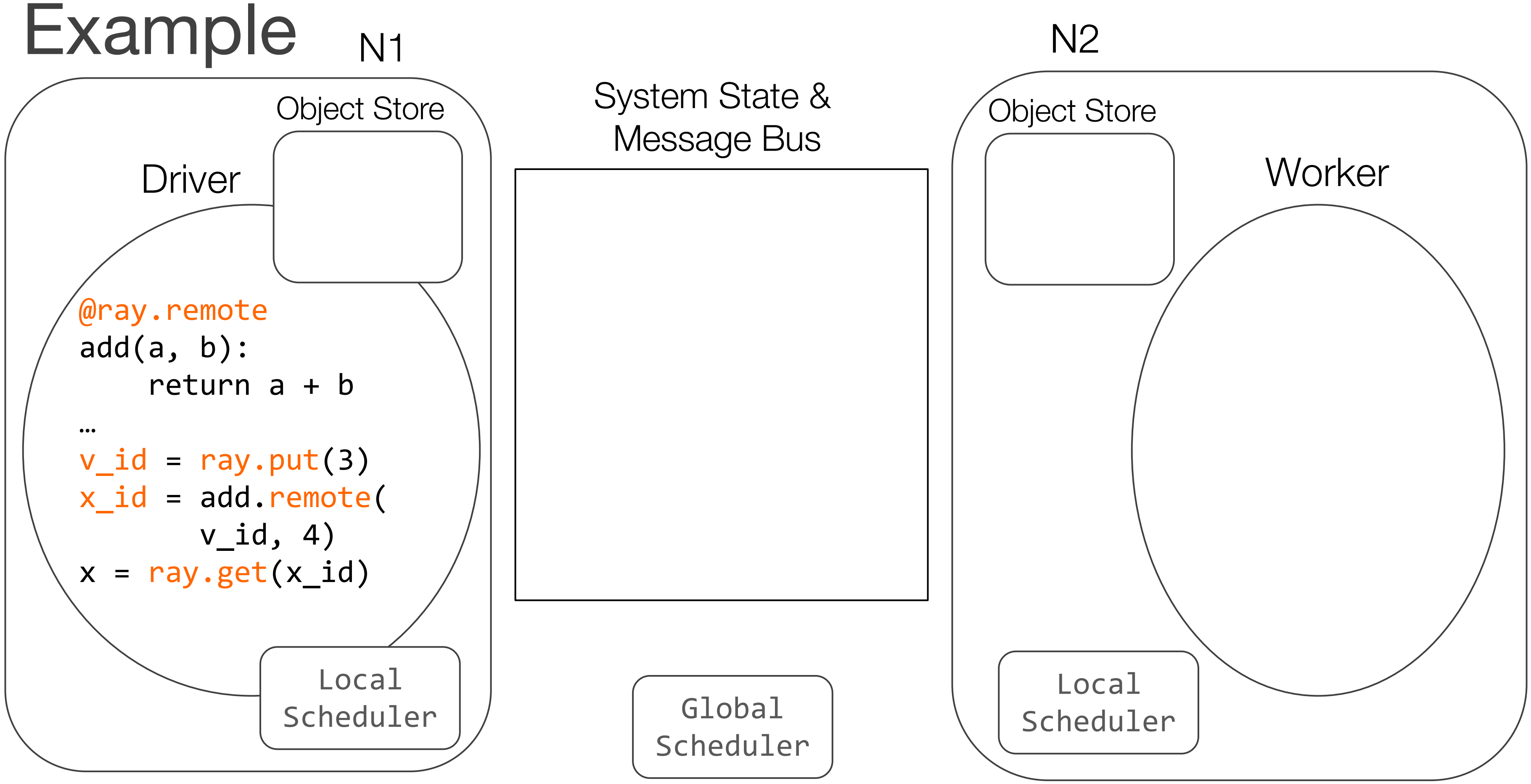
Object Store: could evolve into storage for Arrow

Backend: could evolve into RISE microkernel

Ray System Instantiation & Interaction

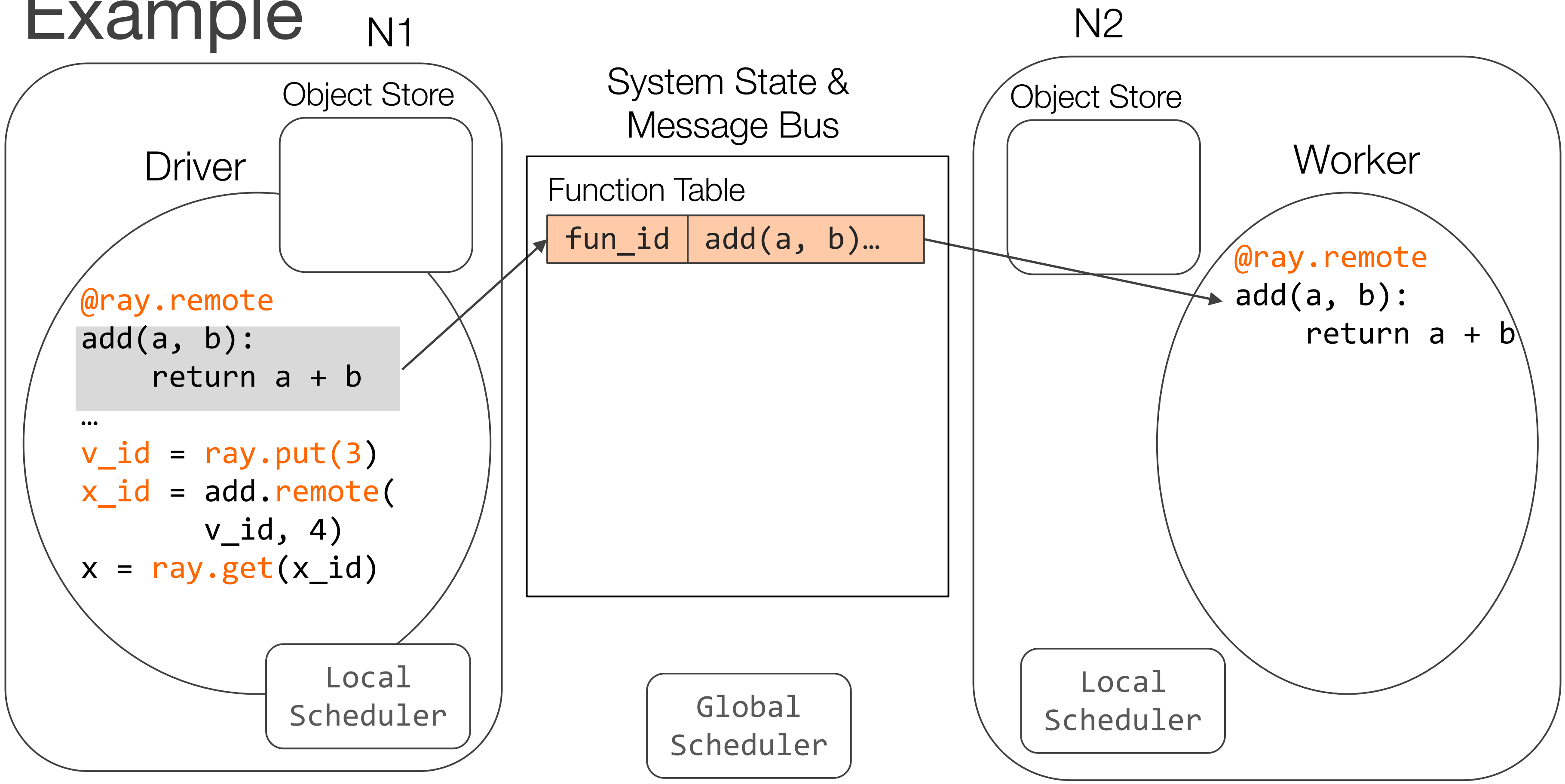


Example

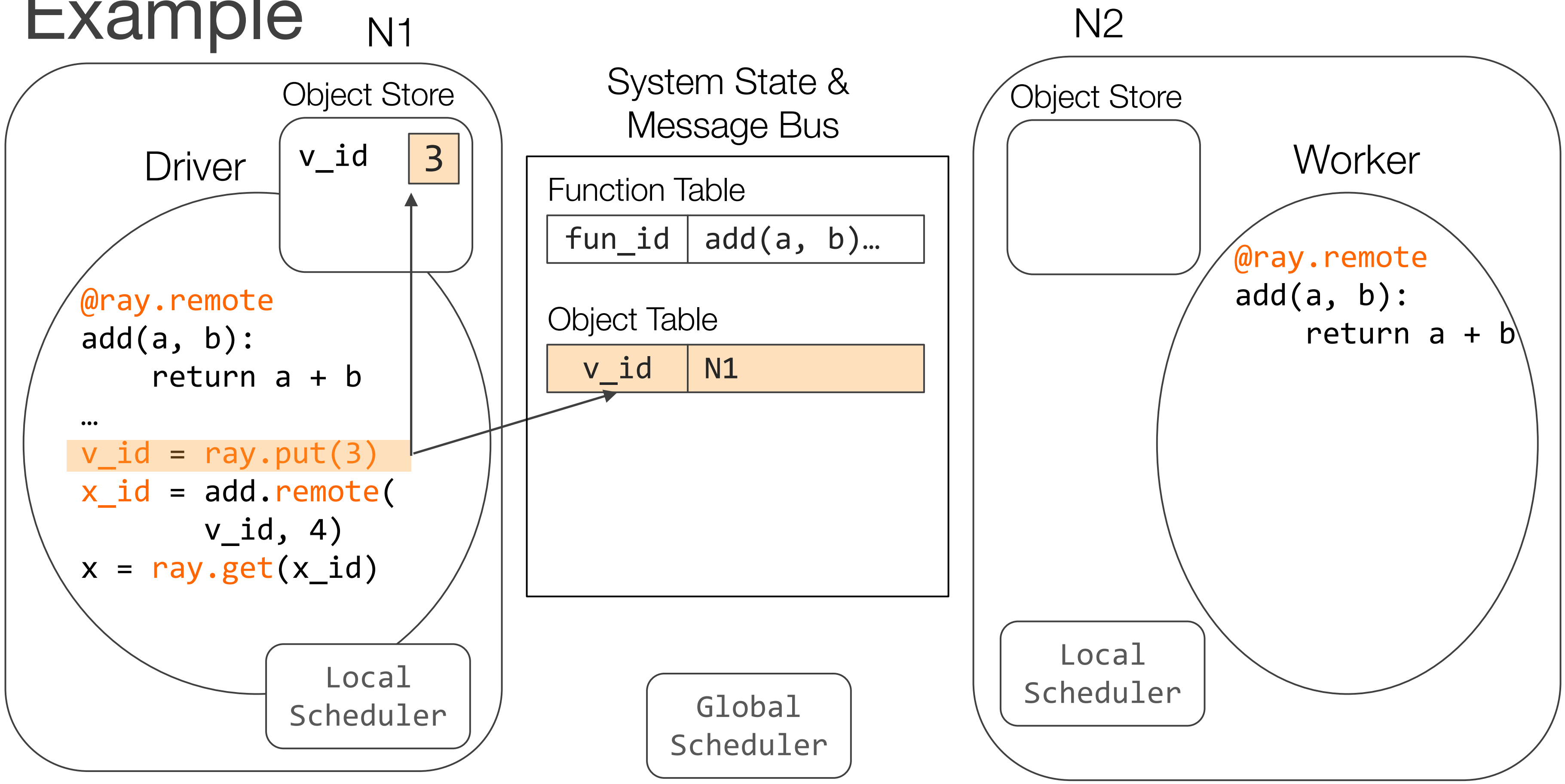


N1
Object Store
Driver
@ray.remote
add(a, b):
    return a + b
…
v_id = ray.put(3)
x_id = add.remote(
        v_id, 4)
x = ray.get(x_id)
Local Scheduler
System State & Message Bus
Global Scheduler
N2
Object Store
Worker
Local Scheduler

Example

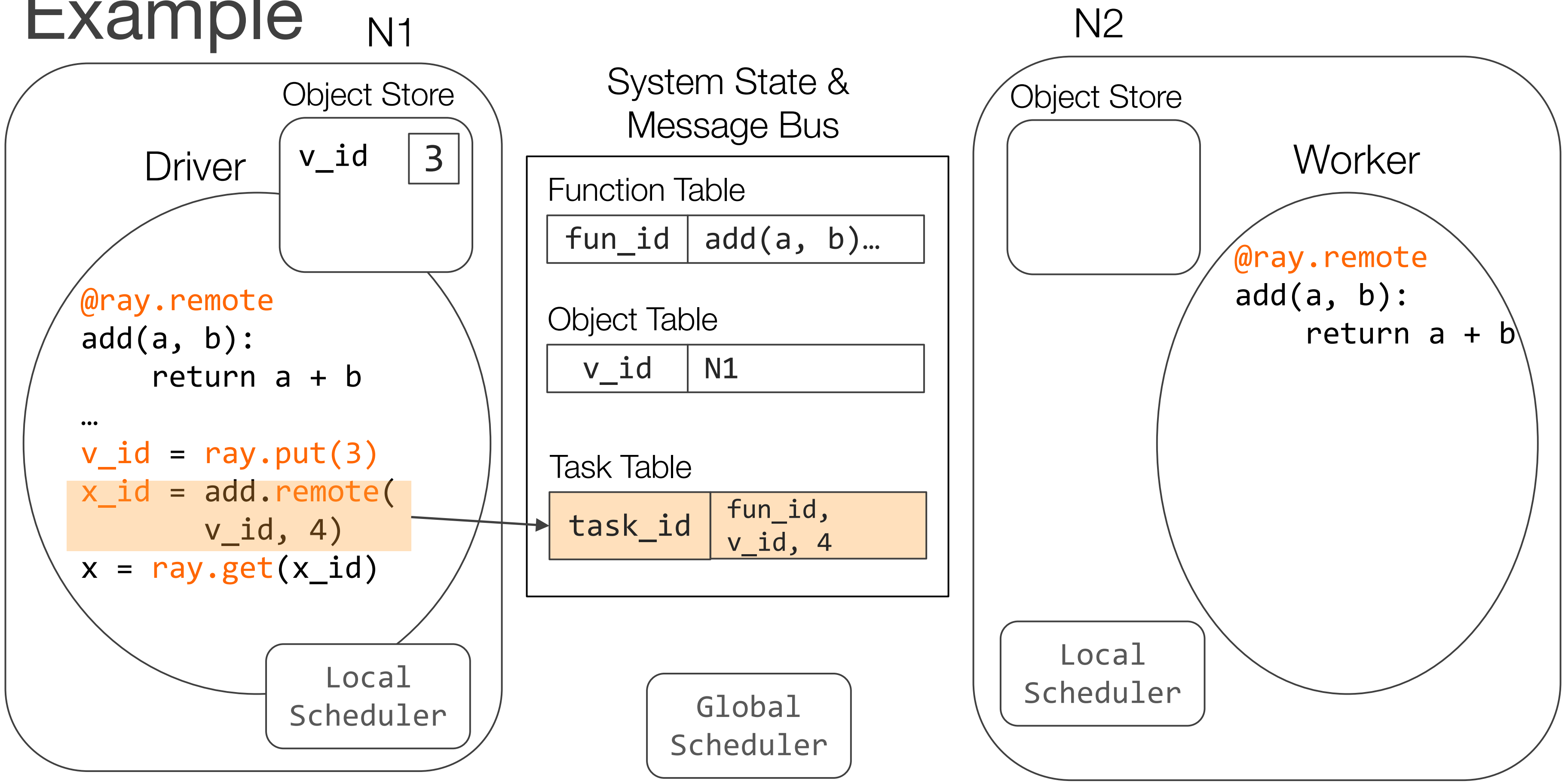

N1
Object Store
Driver
@ray.remote
add(a, b):
return a + b
…
v_id = ray.put(3)
x_id = add.remote(
v_id, 4)
x = ray.get(x_id)
Local
Scheduler
System State &
Message Bus
Function Table
fun_id
add(a, b)…
Global
Scheduler
N2
Object Store
Worker
@ray.remote
add(a, b):
return a + b
Local
Scheduler

Example

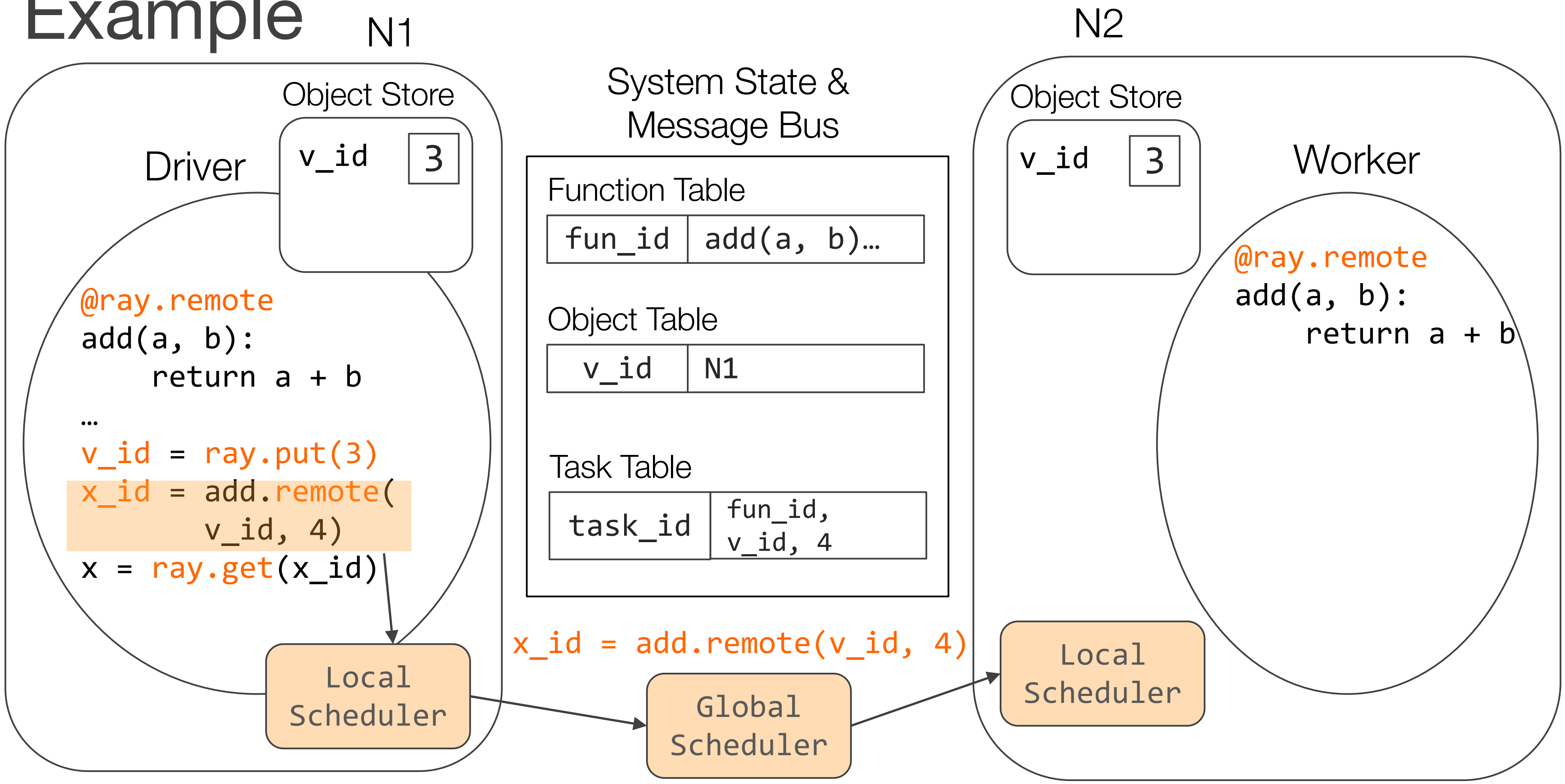


N1
Object Store
v_id
3
Driver
@ray.remote
add(a, b):
return a + b
…
v_id = ray.put(3)
x_id = add.remote(
v_id, 4)
x = ray.get(x_id)
Local Scheduler
System State & Message Bus
Function Table
fun_id
add(a, b)…
Object Table
v_id
N1
Global Scheduler
N2
Object Store
Worker
@ray.remote
add(a, b):
return a + b
Local Scheduler

Example

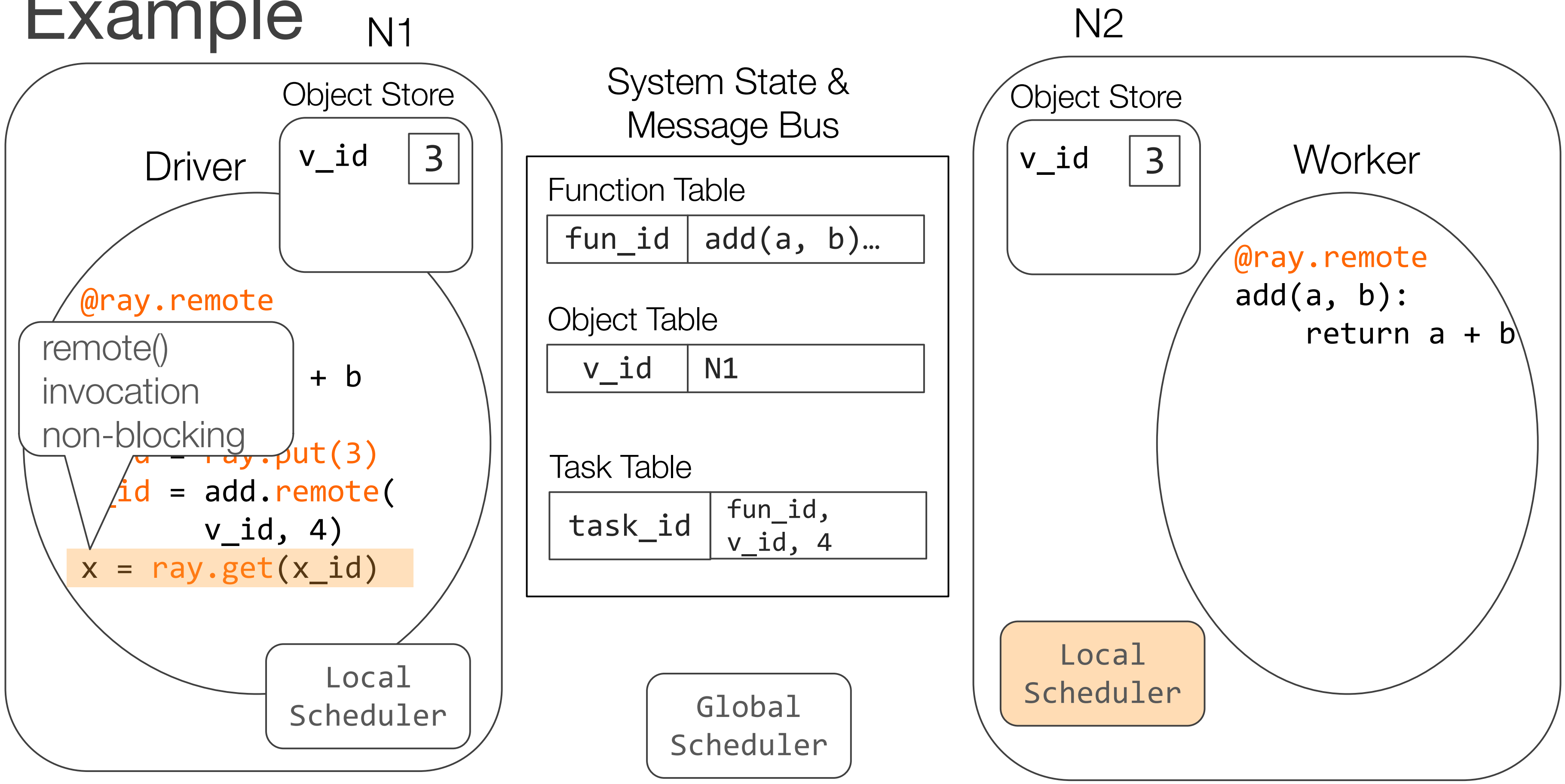

N1
Object Store
v_id
3
Driver
@ray.remote
add(a, b):
return a + b
…
v_id = ray.put(3)
x_id = add.remote(
v_id, 4)
x = ray.get(x_id)
Local Scheduler
System State & Message Bus
Function Table
fun_id
add(a, b)…
Object Table
v_id
N1
Task Table
task_id
fun_id, v_id, 4
Global Scheduler
N2
Object Store
Worker
@ray.remote
add(a, b):
return a + b
Local Scheduler

Example



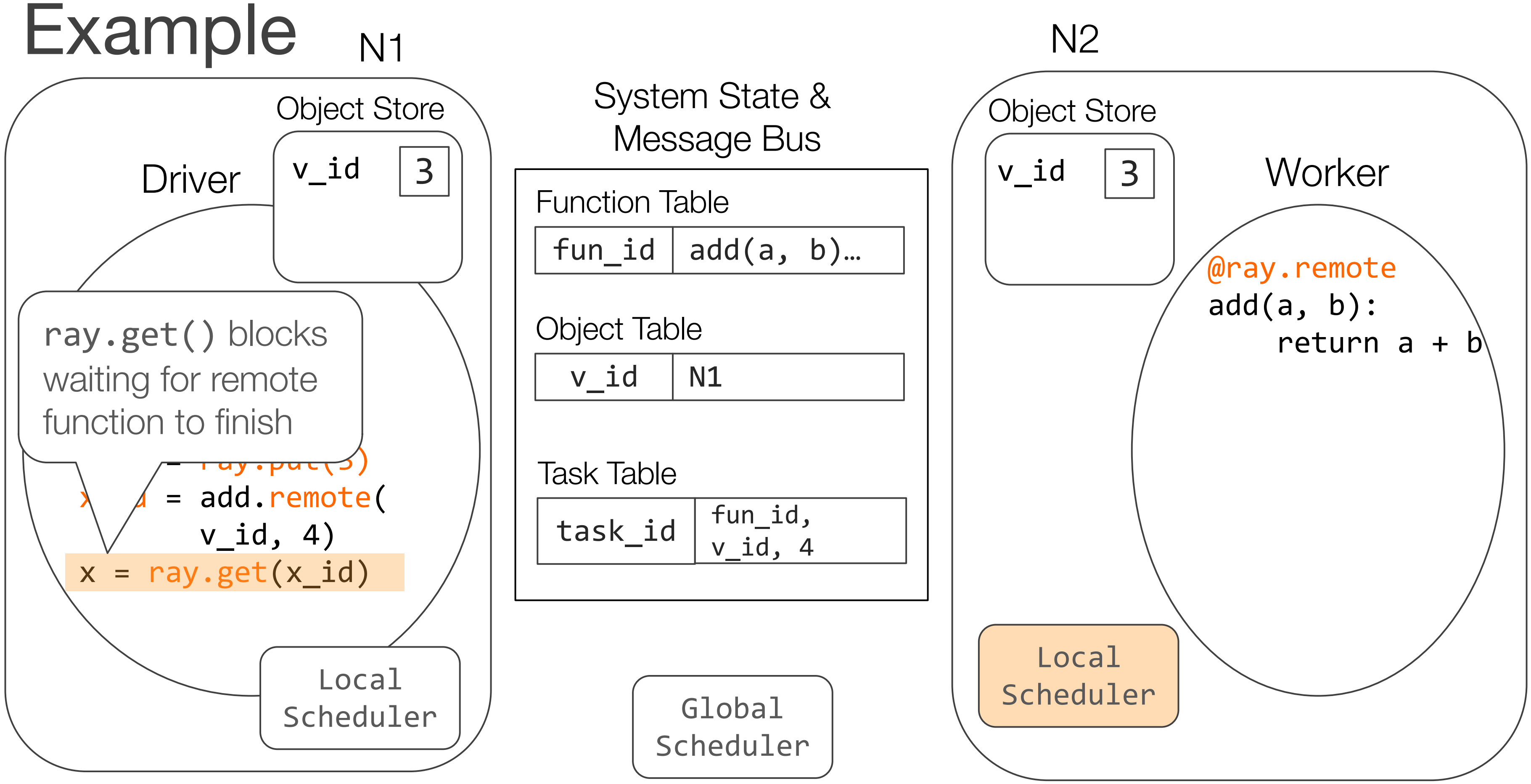
N1
Object Store
Driver
v_id
3
@ray.remote
add(a, b):
return a + b
…
v_id = ray.put(3)
x_id = add.remote(
v_id, 4)
x = ray.get(x_id)
Local Scheduler
System State &
Message Bus
Function Table
fun_id
add(a, b)…
Object Table
v_id
N1
Task Table
task_id
fun_id,
v_id, 4
x_id = add.remote(v_id, 4)
Global Scheduler
N2
Object Store
v_id
3
Worker
@ray.remote
add(a, b):
return a + b
Local Scheduler

Example

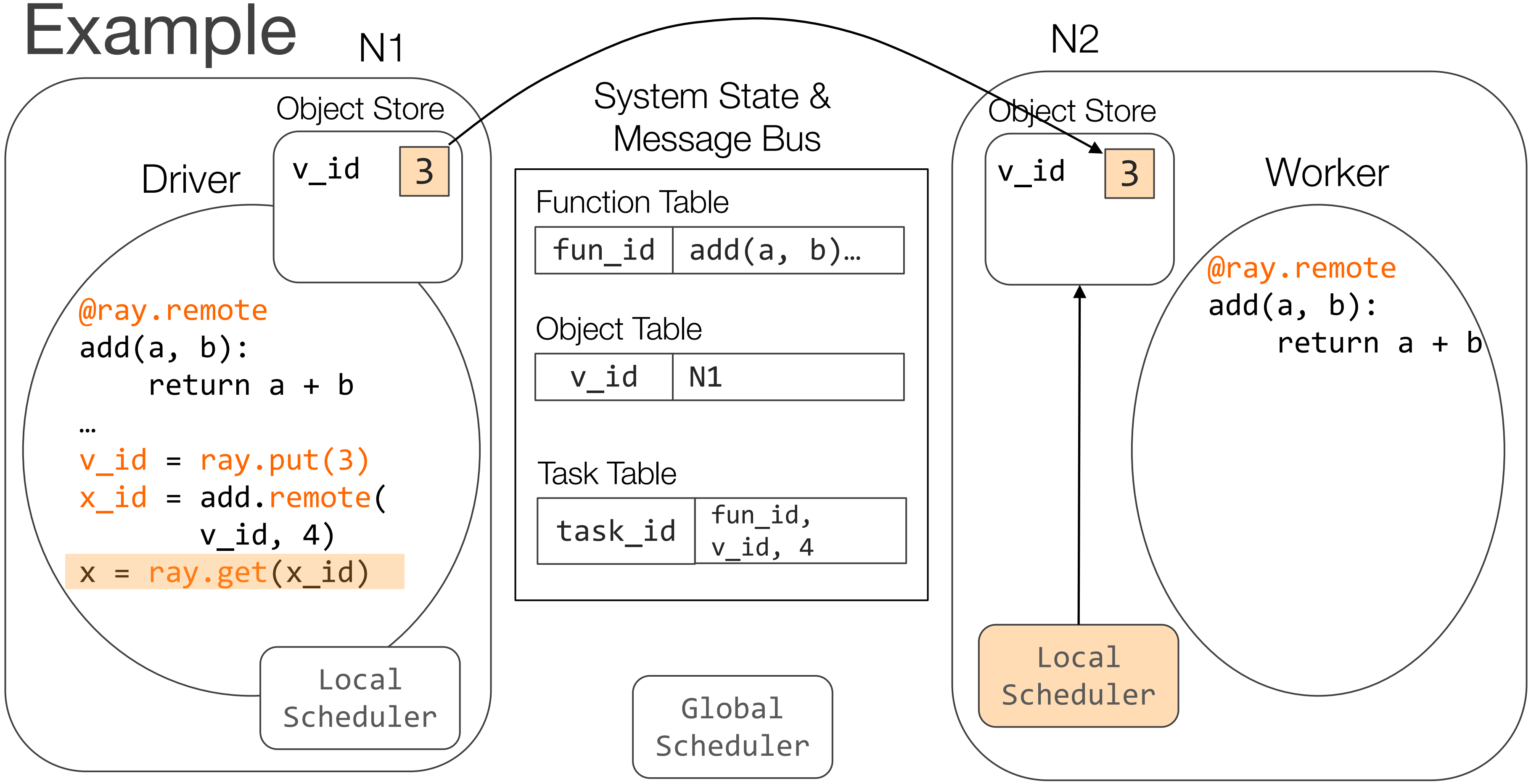


N1
Object Store
v_id
3
Driver
@ray.remote
remote()
invocation
non-blocking
+ b
id = add.remote(
v_id, 4)
x = ray.get(x_id)
Local
Scheduler
System State &
Message Bus
Function Table
fun_id
add(a, b)…
Object Table
v_id
N1
Task Table
task_id
fun_id,
v_id, 4
Global
Scheduler
N2
Object Store
v_id
3
Worker
@ray.remote
add(a, b):
return a + b
Local
Scheduler

Example

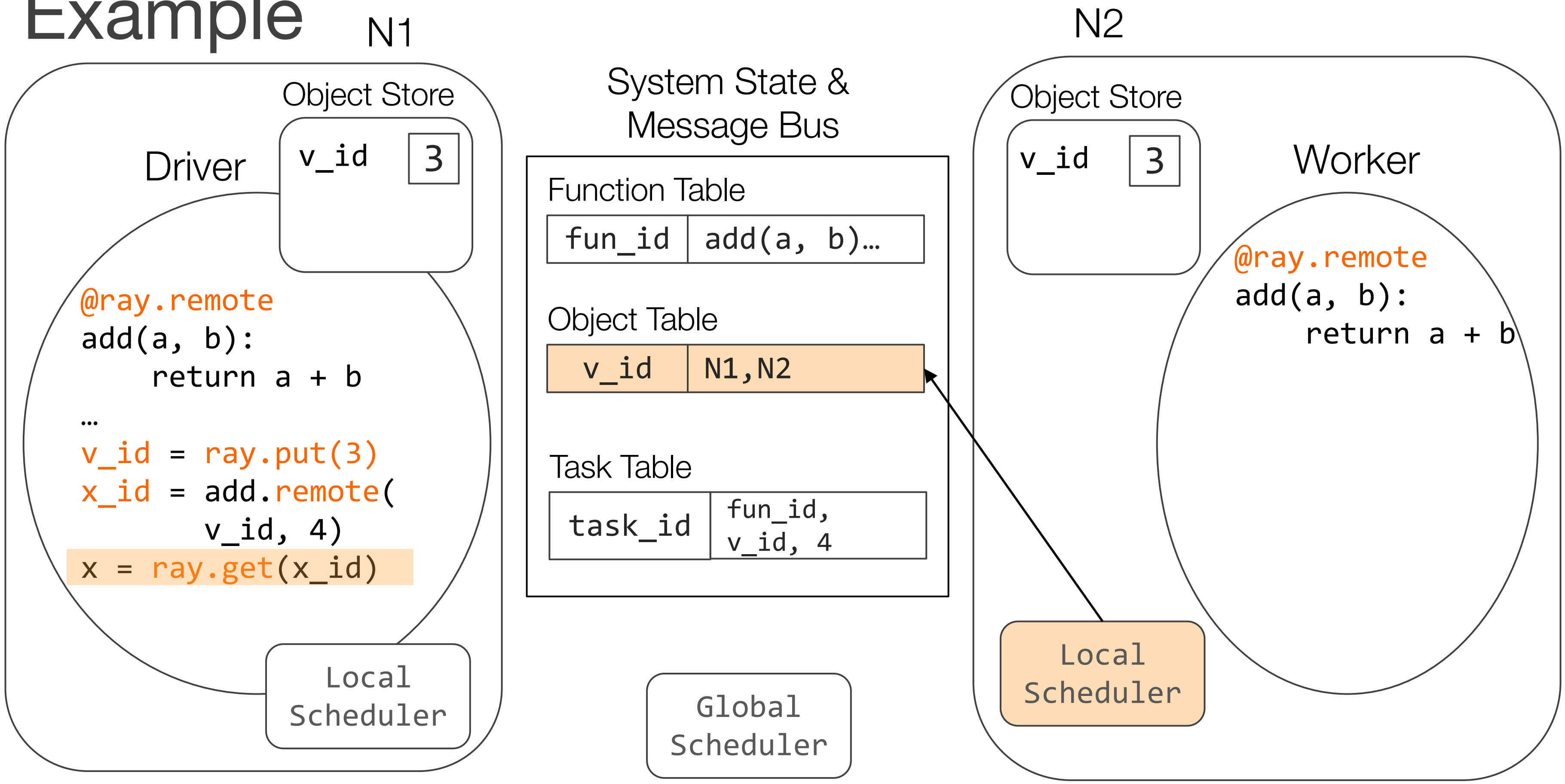

N1
Object Store
v_id
3
Driver
ray.get() blocks waiting for remote function to finish
= add.remote(
v_id, 4)
x = ray.get(x_id)
Local Scheduler
System State & Message Bus
Function Table
fun_id
add(a, b)…
Object Table
v_id
N1
Task Table
task_id
fun_id, v_id, 4
Global Scheduler
N2
Object Store
v_id
3
Worker
@ray.remote
add(a, b):
return a + b
Local Scheduler

Example



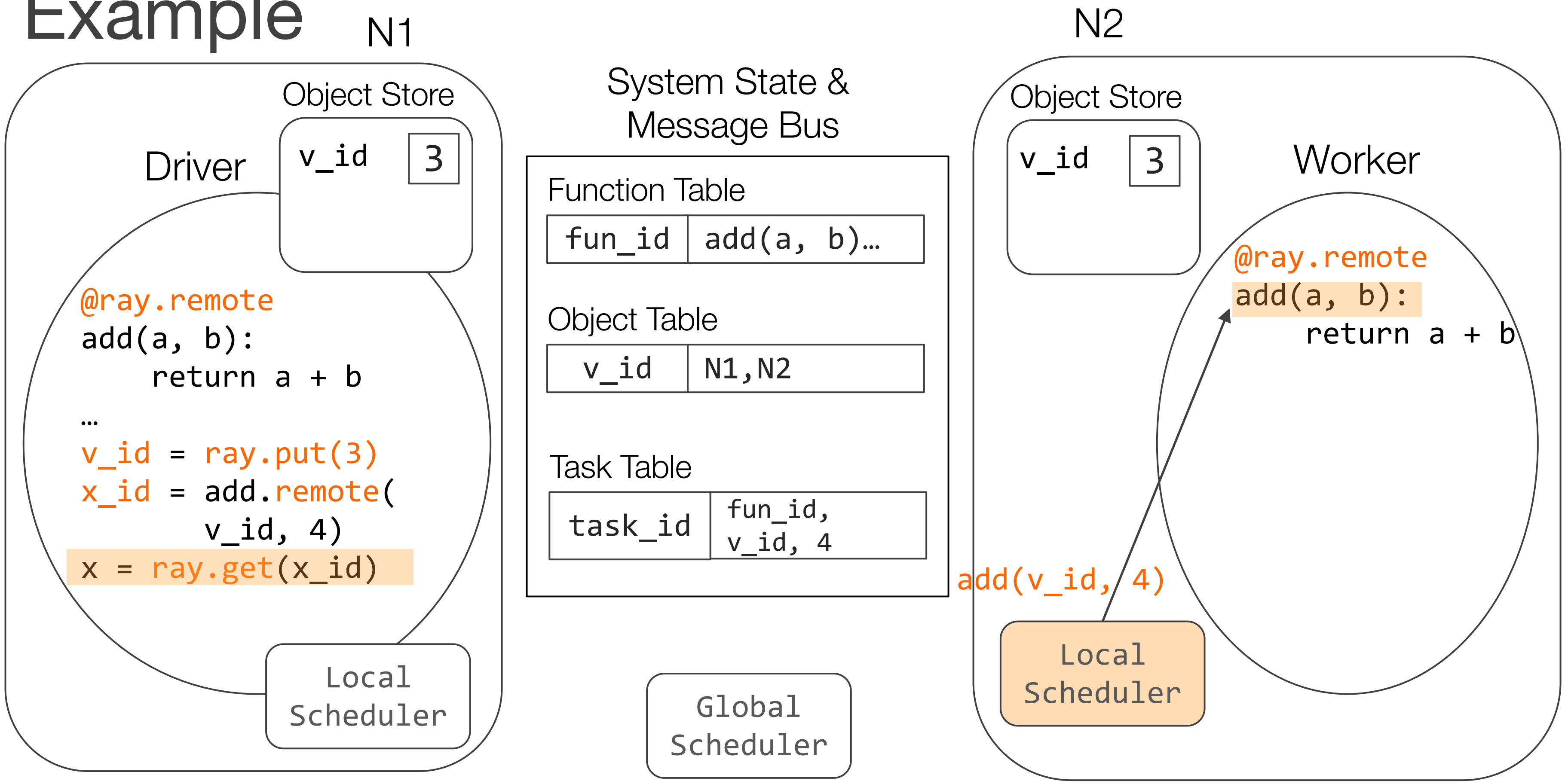
N1
Object Store
Driver
v_id
3
@ray.remote
add(a, b):
return a + b
…
v_id = ray.put(3)
x_id = add.remote(
v_id, 4)
x = ray.get(x_id)
Local Scheduler
System State & Message Bus
Function Table
fun_id
add(a, b)…
Object Table
v_id
N1
Task Table
task_id
fun_id, v_id, 4
Global Scheduler
N2
Object Store
v_id
3
Worker
@ray.remote
add(a, b):
return a + b
Local Scheduler

Example



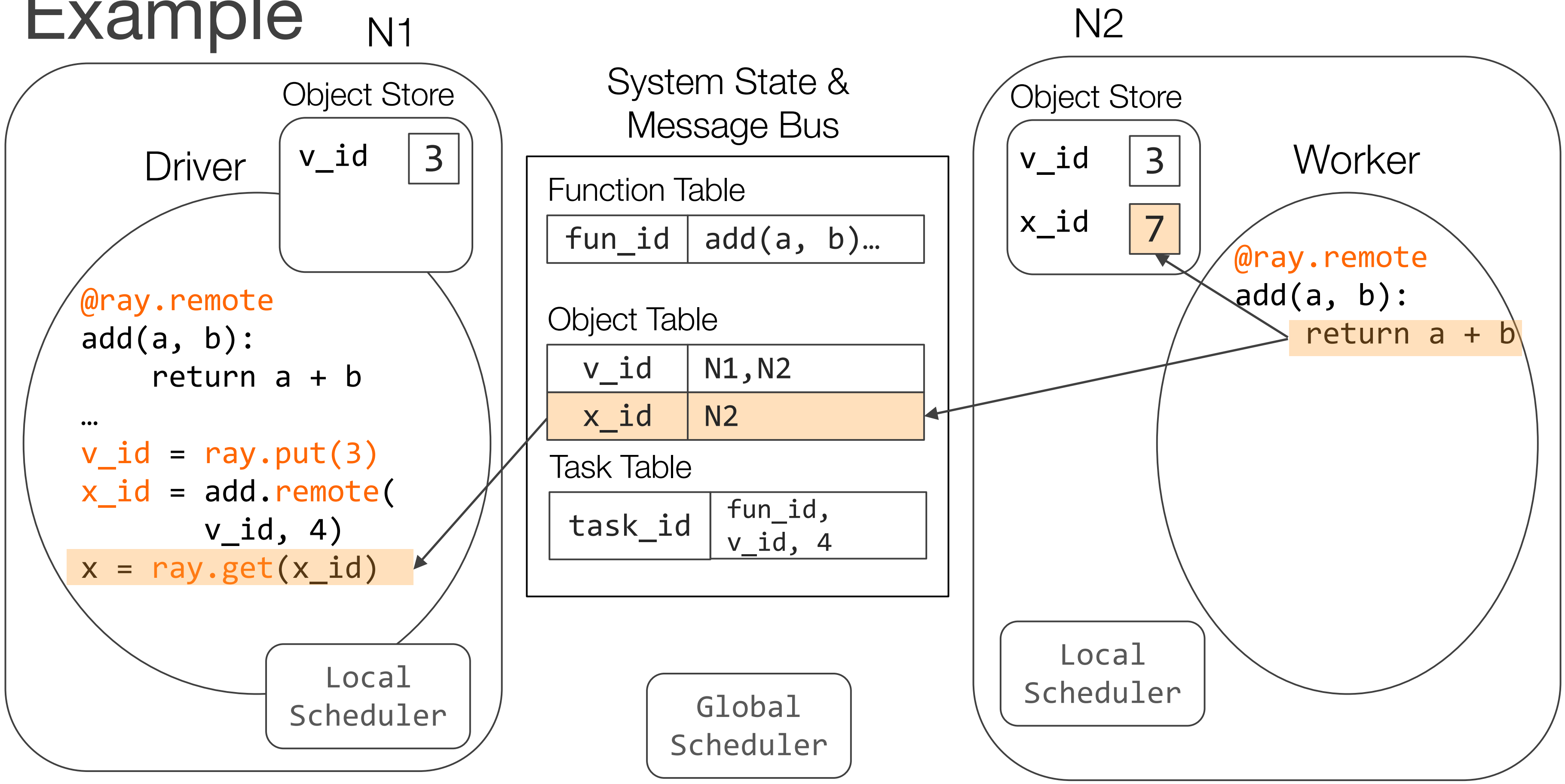
N1
Object Store
Driver
v_id
3
@ray.remote
add(a, b):
return a + b
…
v_id = ray.put(3)
x_id = add.remote(
v_id, 4)
x = ray.get(x_id)
Local Scheduler
System State &
Message Bus
Function Table
fun_id
add(a, b)…
Object Table
v_id
N1,N2
Task Table
task_id
fun_id, v_id, 4
Global Scheduler
N2
Object Store
v_id
3
Worker
@ray.remote
add(a, b):
return a + b
Local Scheduler

Example

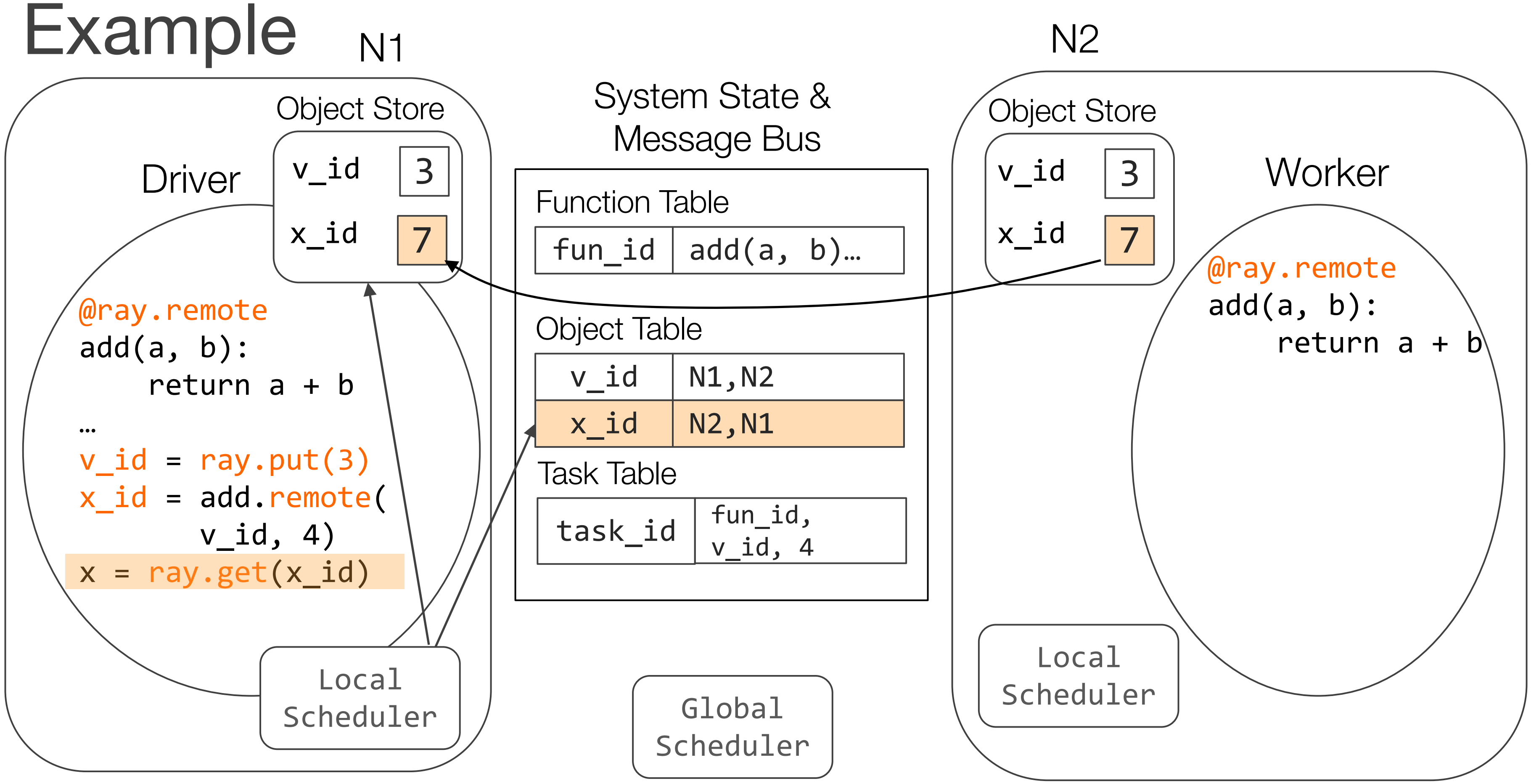


N1
Object Store
v_id
3
Driver
@ray.remote
add(a, b):
return a + b
…
v_id = ray.put(3)
x_id = add.remote(
v_id, 4)
x = ray.get(x_id)
Local Scheduler
System State & Message Bus
Function Table
fun_id
add(a, b)…
Object Table
v_id
N1,N2
Task Table
task_id
fun_id, v_id, 4
Global Scheduler
N2
Object Store
v_id
3
Worker
@ray.remote
add(a, b):
return a + b
add(v_id, 4)
Local Scheduler

Example

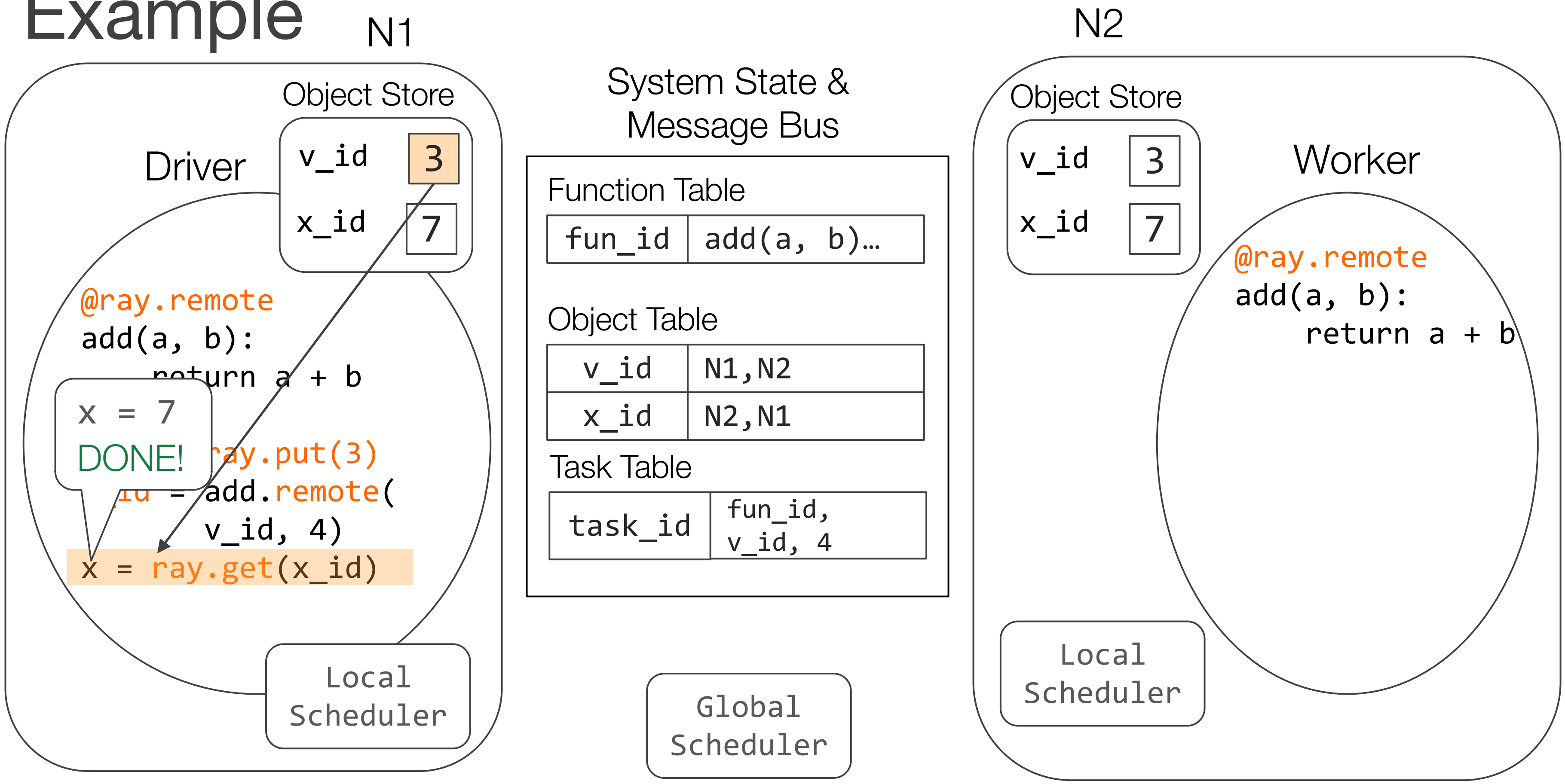

N1
N2
Object Store
Driver
v_id 3
@ray.remote
add(a, b):
return a + b
…
v_id = ray.put(3)
x_id = add.remote(
v_id, 4)
x = ray.get(x_id)
Local Scheduler
System State & Message Bus
Function Table
fun_id
add(a, b)…
Object Table
v_id
N1,N2
x_id
N2
Task Table
task_id
fun_id, v_id, 4
Global Scheduler
Object Store
v_id 3
x_id 7
Worker
@ray.remote
add(a, b):
return a + b
Local Scheduler

Example



N1
Object Store
Driver
v_id
3
x_id
7
@ray.remote
add(a, b):
return a + b
…
v_id = ray.put(3)
x_id = add.remote(
v_id, 4)
x = ray.get(x_id)
Local Scheduler
System State & Message Bus
Function Table
fun_id
add(a, b)…
Object Table
v_id
N1,N2
x_id
N2,N1
Task Table
task_id
fun_id, v_id, 4
Global Scheduler
N2
Object Store
v_id
3
x_id
7
Worker
@ray.remote
add(a, b):
return a + b
Local Scheduler

Example



N1
Object Store
v_id
3
x_id
7
Driver
@ray.remote
add(a, b):
return a + b
x = 7
DONE!
ay.put(3)
add.remote(
v_id, 4)
x = ray.get(x_id)
Local Scheduler
System State & Message Bus
Function Table
fun_id
add(a, b)…
Object Table
v_id
N1,N2
x_id
N2,N1
Task Table
task_id
fun_id, v_id, 4
Global Scheduler
N2
Object Store
v_id
3
x_id
7
Worker
@ray.remote
add(a, b):
return a + b
Local Scheduler

Project & Exam Dates

Wednesday, 9/7: google doc with project suggestions

- Include other topics, such as graph streaming

Monday, 9/19: pick a partner and send your project proposal

- I'll send a google form to fill in for your project proposals

Monday, 10/12: project progress review

- More details to follow

Wednesday, 10/5: Midterm exam