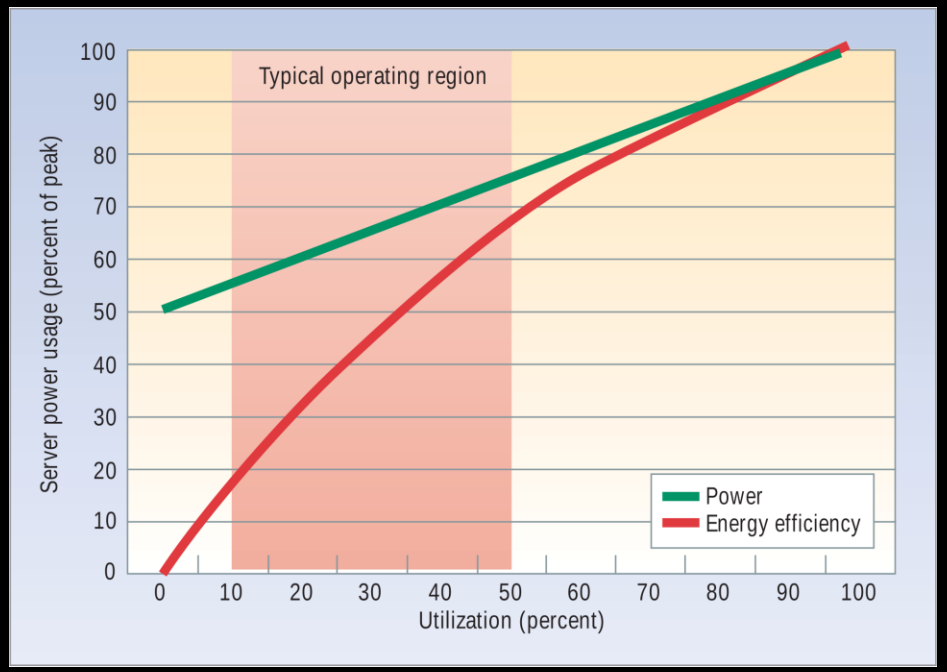
Give our Lambda's Connectivity, or else...

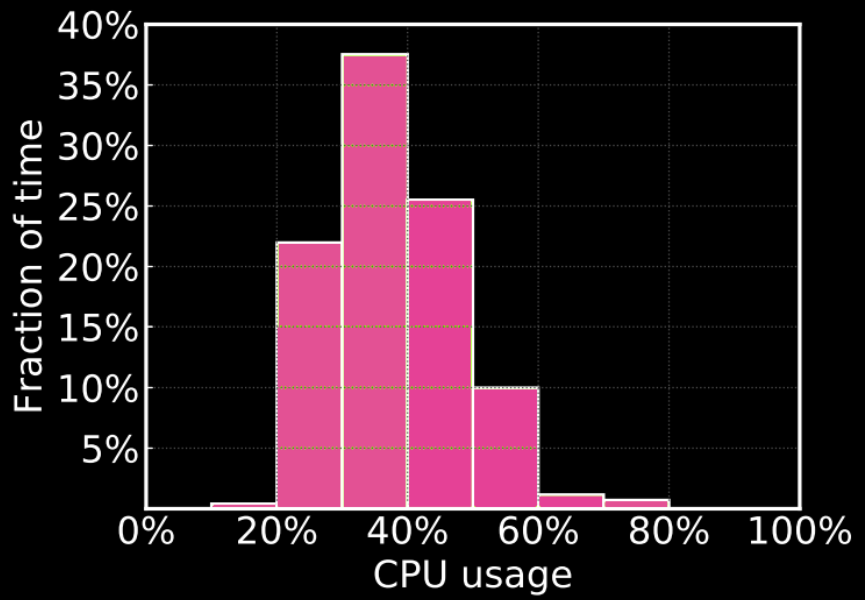
...we'll take it for ourselves

Finn Westenfelder | Baltasar Dinis | William Dale

Understanding the Problems Serverless wants to Solve



Machine **under-utilization** is **energy-inefficient**



Machines in datacenters are **very underutilized**

L. A. Barroso and U. Hölzle. The case for energy-proportional computing. Computer

Guo, Jing, et al. "Who limits the resource efficiency of my datacenter: An analysis of Alibaba datacenter traces." *Proceedings of the International Symposium on Quality of Service*. 2019

Understanding the Problems Serverless wants to Solve

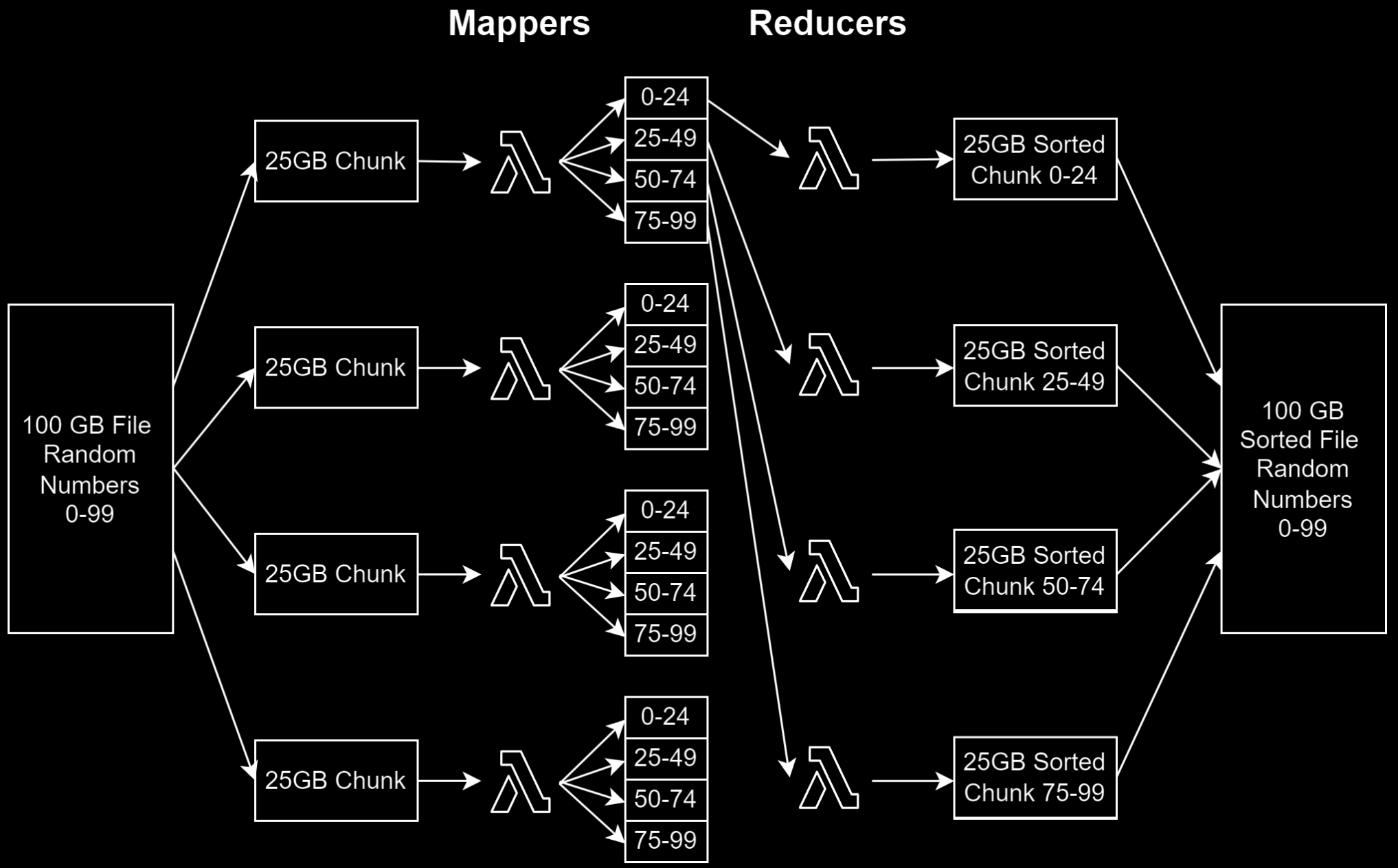
- Why are all these machines so underutilized?
  - Provisioning correctly is very hard
  - **Load changes** a lot over **time**
  - Need to **over-provision**
- If we allocate **resources as function of load**, we can achieve **better utilization**
- **Remove burden** on the developer to scale the deployment

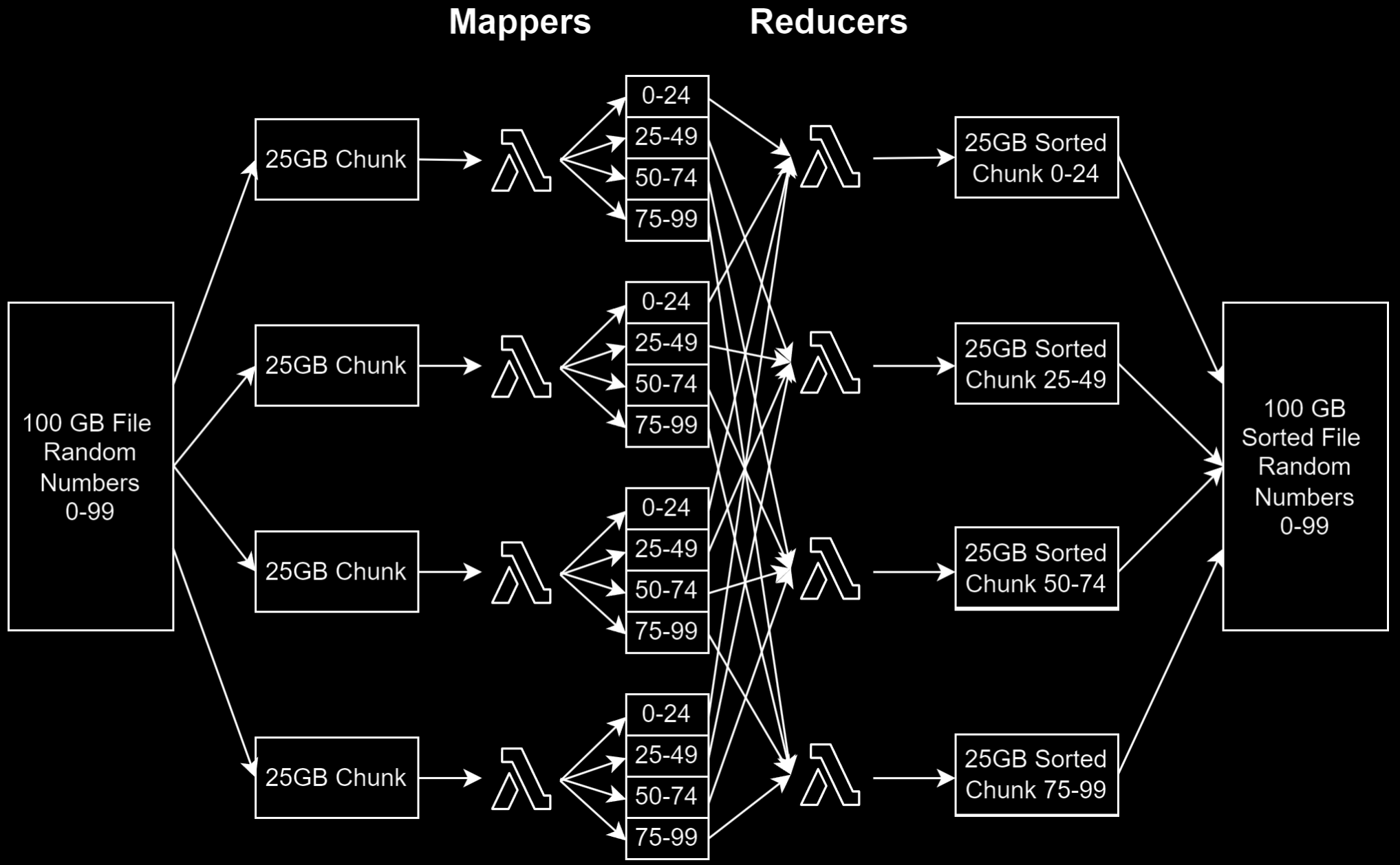
The core values of Serverless

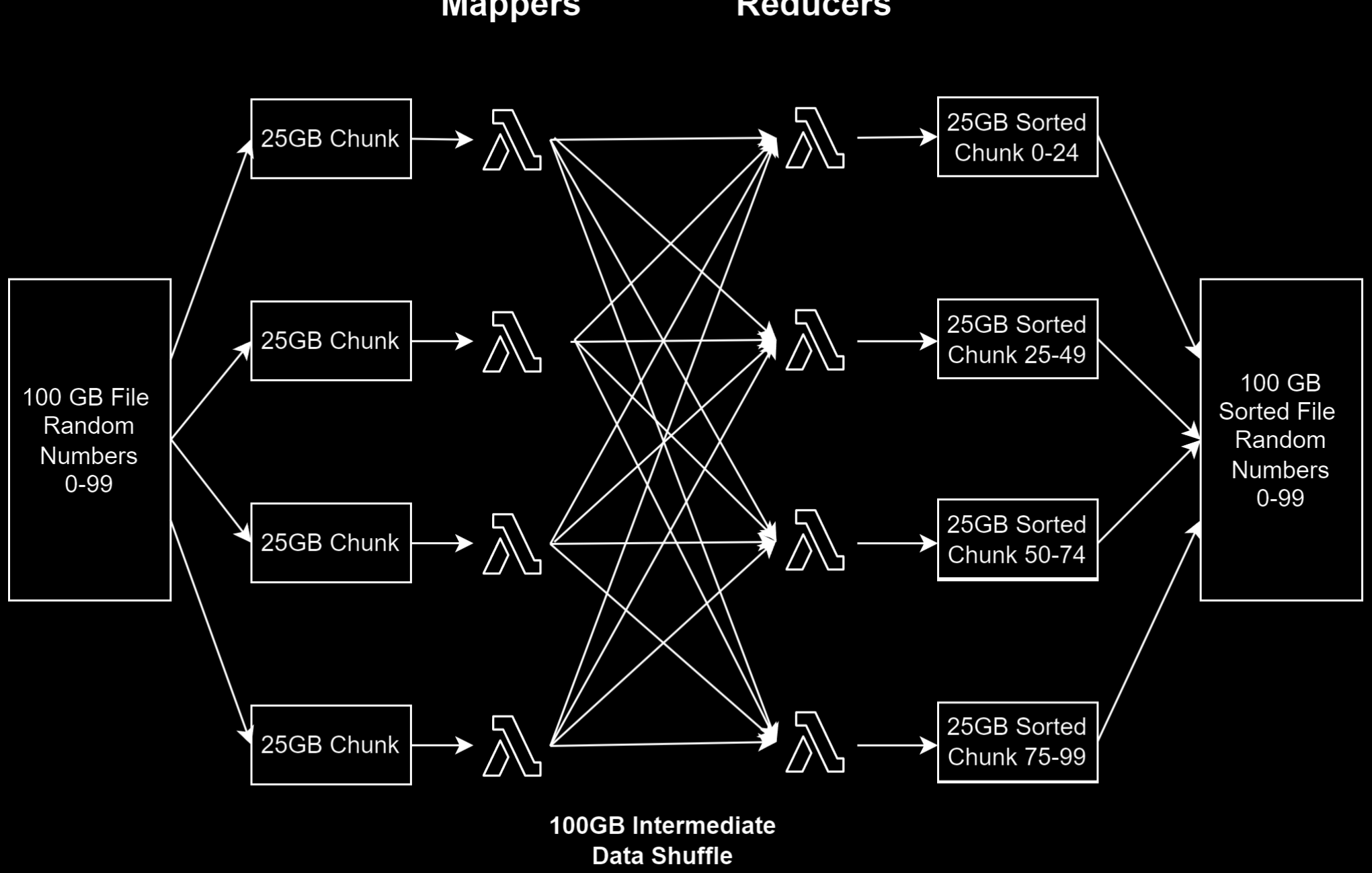
- The **client** expresses applications as **collections of functions**
- The **cloud provider** is responsible for:
  - Allocating resources
  - Load balancing
- The **cloud provider** also:
  - Retains the right to kill functions as needed for load balancing
  - Restricts the environment functions run on (e.g., network, local storage)
- The **client** must adapt to this environment

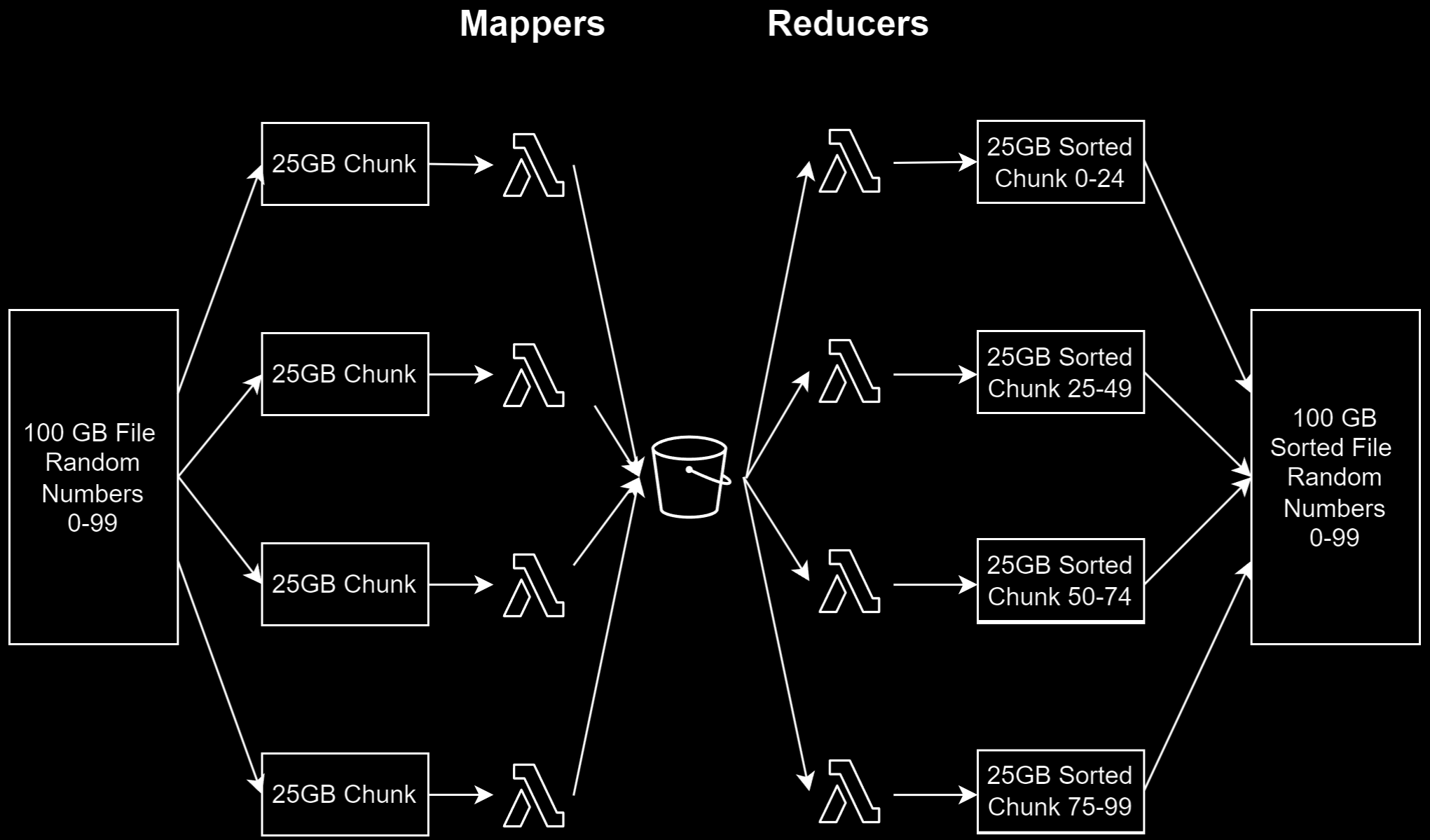
Why no networks for Lambdas

- AWS Lambdas cannot communicate directly (i.e., via IP) with each other
  - This would be useful to build **function pipelines**
- Why?
  - There is no guarantee that Lambda's would run **concurrently**
  - Lambdas can be killed by the administrator, which would **create a cascading failure**
- How is it done in practice?
  - Communicate asynchronously through a storage service (e.g., S3)
  - Expensive, slow and sub-optimal

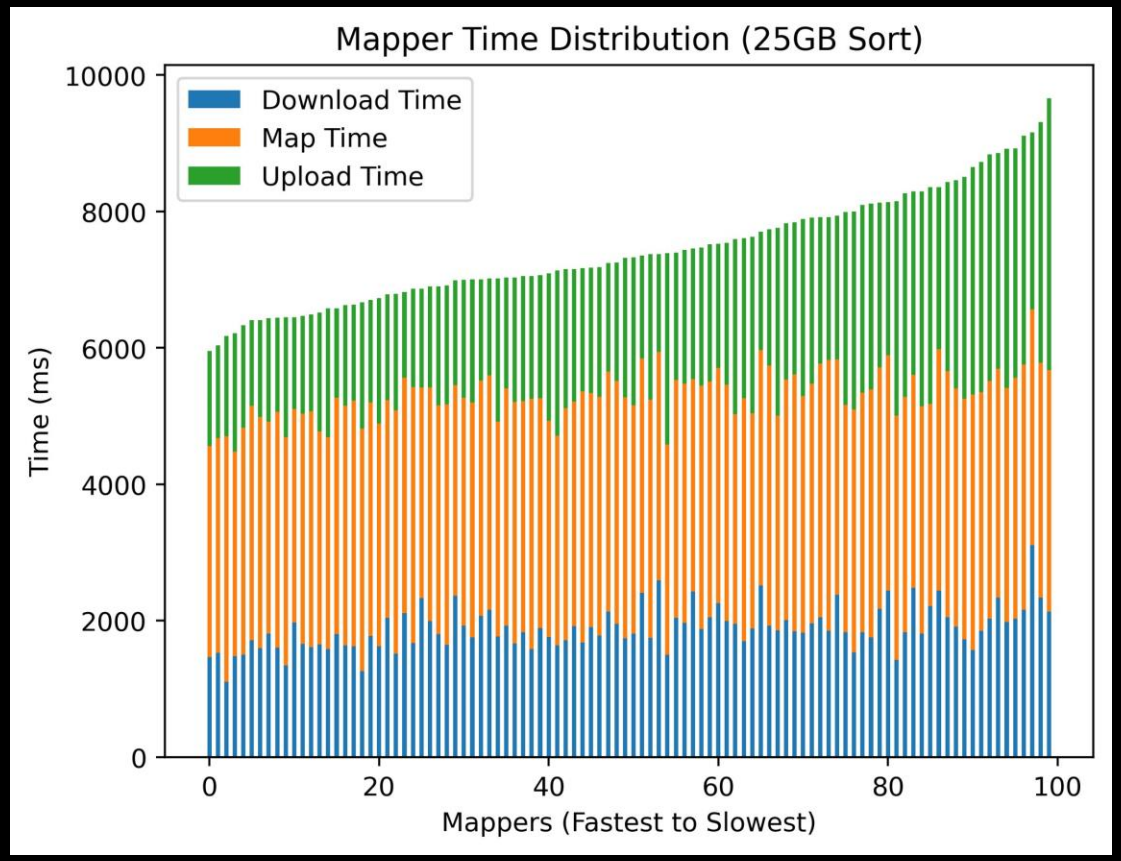

Mappers
Reducers
100 GB File Random Numbers 0-99
25GB Chunk
25GB Chunk
25GB Chunk
25GB Chunk
0-24
25-49
50-74
75-99
0-24
25-49
50-74
75-99
0-24
25-49
50-74
75-99
0-24
25-49
50-74
75-99
25GB Sorted Chunk 0-24
25GB Sorted Chunk 25-49
25GB Sorted Chunk 50-74
25GB Sorted Chunk 75-99
100 GB Sorted File Random Numbers 0-99

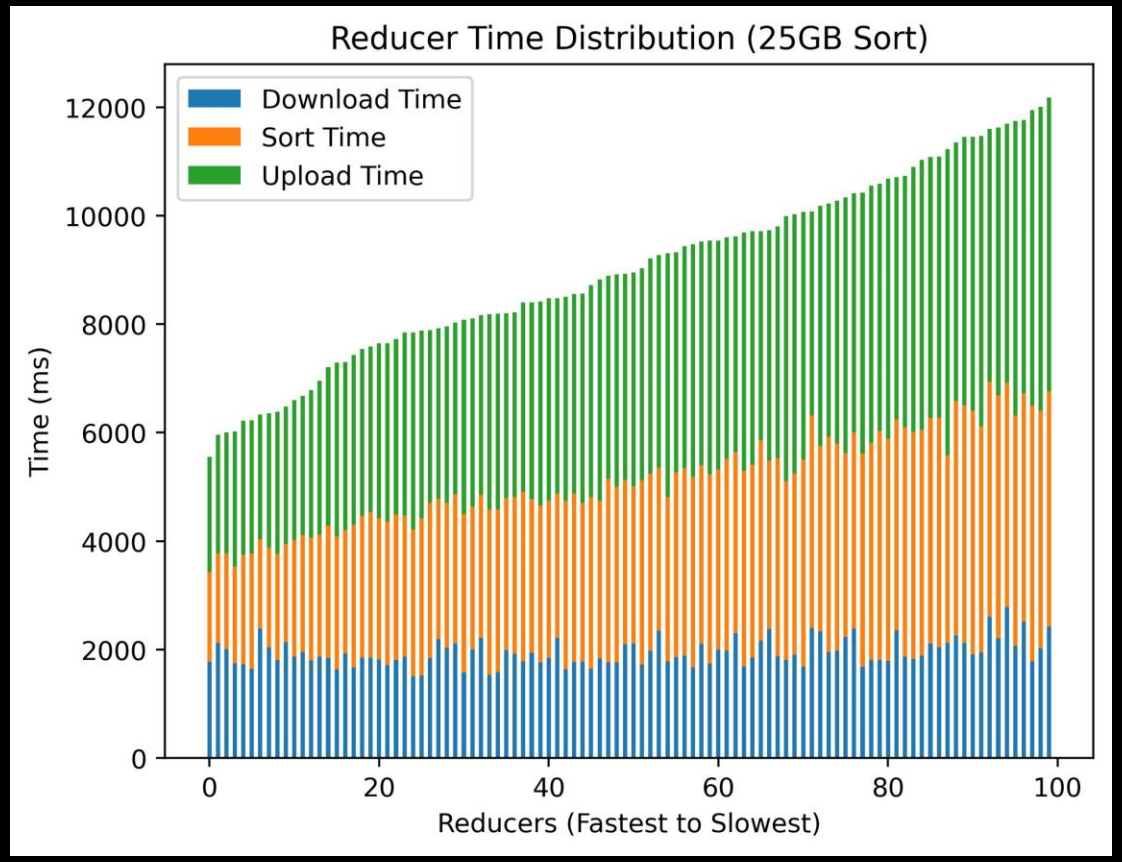

Mappers
Reducers
100 GB File Random Numbers 0-99
25GB Chunk
25GB Chunk
25GB Chunk
25GB Chunk
0-24
25-49
50-74
75-99
0-24
25-49
50-74
75-99
0-24
25-49
50-74
75-99
0-24
25-49
50-74
75-99
25GB Sorted Chunk 0-24
25GB Sorted Chunk 25-49
25GB Sorted Chunk 50-74
25GB Sorted Chunk 75-99
100 GB Sorted File Random Numbers 0-99


Mappers
Reducers
100 GB File Random Numbers 0-99
25GB Chunk
25GB Chunk
25GB Chunk
25GB Chunk
25GB Sorted Chunk 0-24
25GB Sorted Chunk 25-49
25GB Sorted Chunk 50-74
25GB Sorted Chunk 75-99
100 GB Sorted File Random Numbers 0-99
100GB Intermediate Data Shuffle


Mappers
Reducers
100 GB File Random Numbers 0-99
25GB Chunk
25GB Chunk
25GB Chunk
25GB Chunk
25GB Sorted Chunk 0-24
25GB Sorted Chunk 25-49
25GB Sorted Chunk 50-74
25GB Sorted Chunk 75-99
100 GB Sorted File Random Numbers 0-99
100GB Intermediate Data Shuffle

Communication is a bottleneck





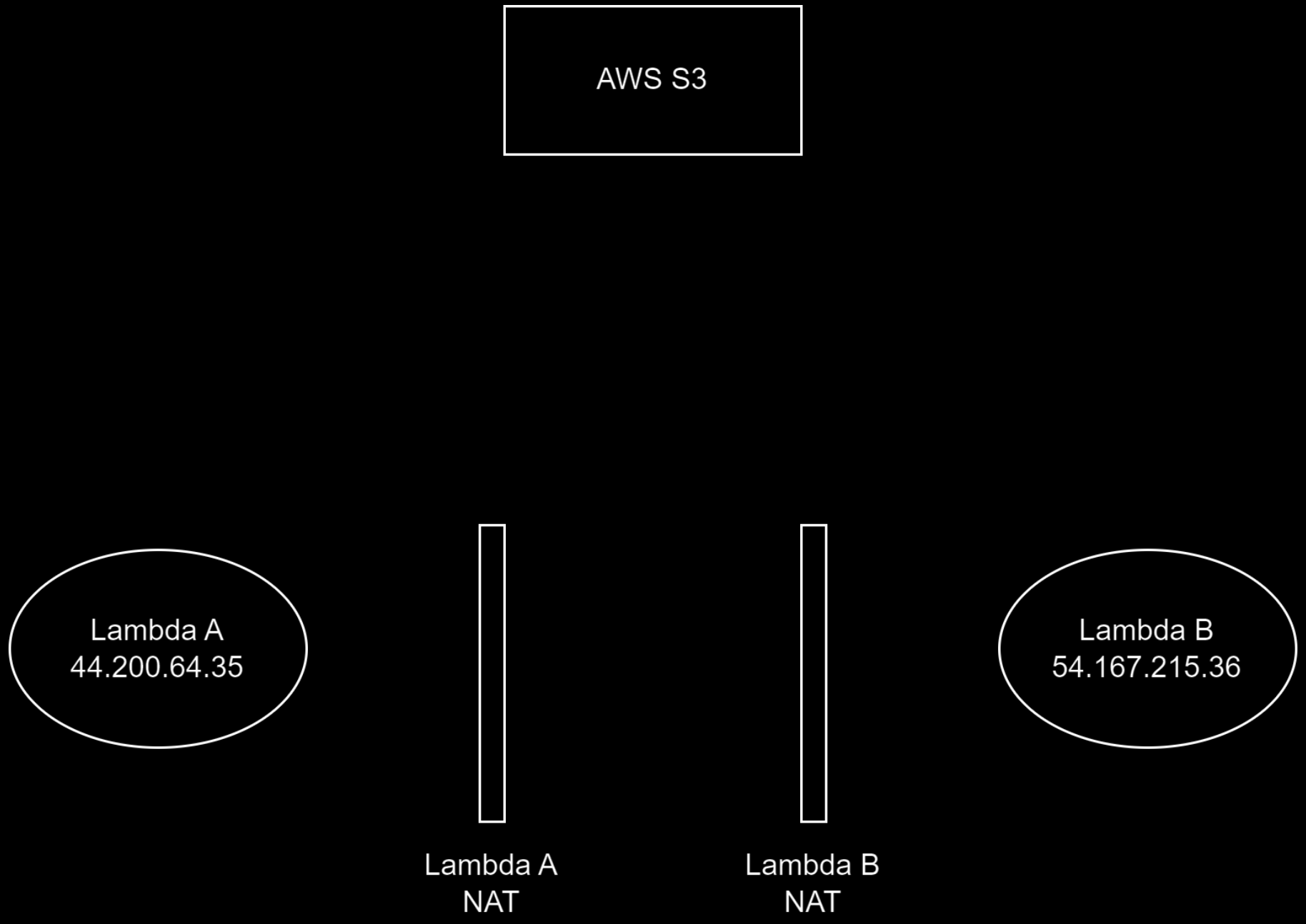
Latency breakdown of compute time and communication time (through S3) of a MapReduce-like job implemented using Lambdas communicating over S3

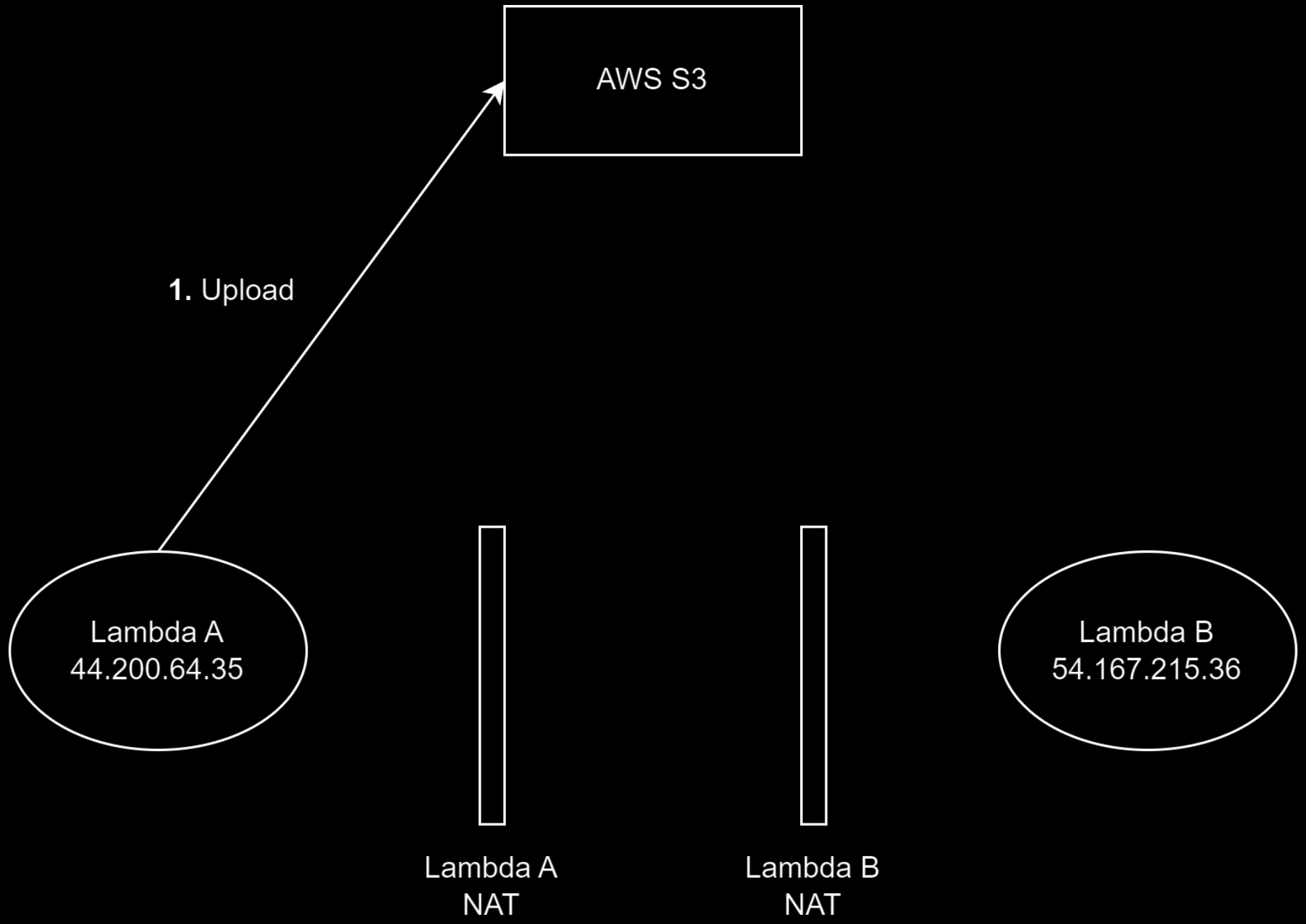
Roadmap

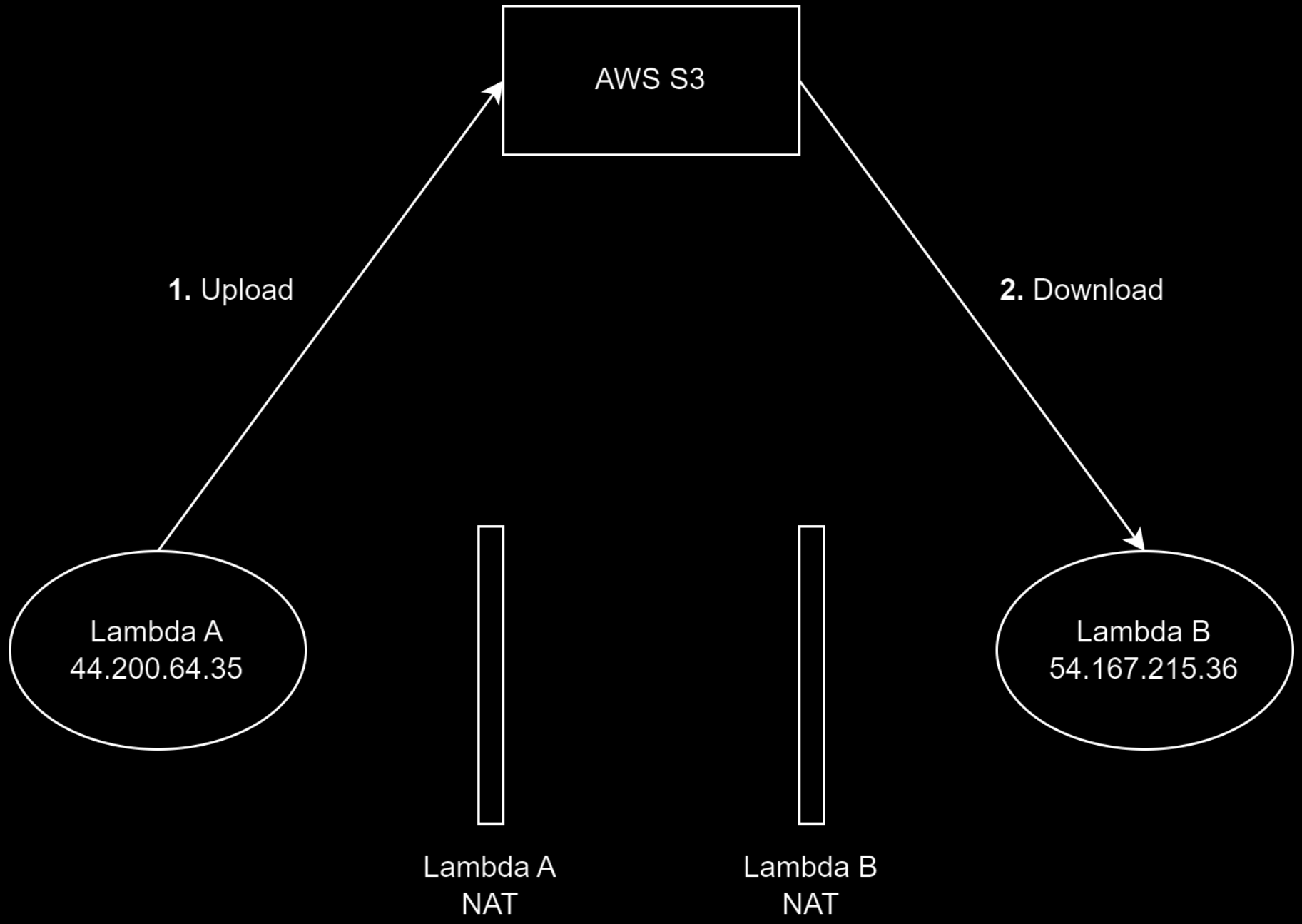
- Serverless primer
  - Why serverless
  - How serverless
  - Why no serverless + networks (so far)
- Application Example
  - MapReduce framework

- Punching holes in the Network for fun and profit
  - Technique
  - Battle Scars
  - Performance, or *the great S3 conspiracy*
- Comparison with AWS EMR
- Making it resilient
- Conclusion & Future Work

Punching Holes for Fun and Profit

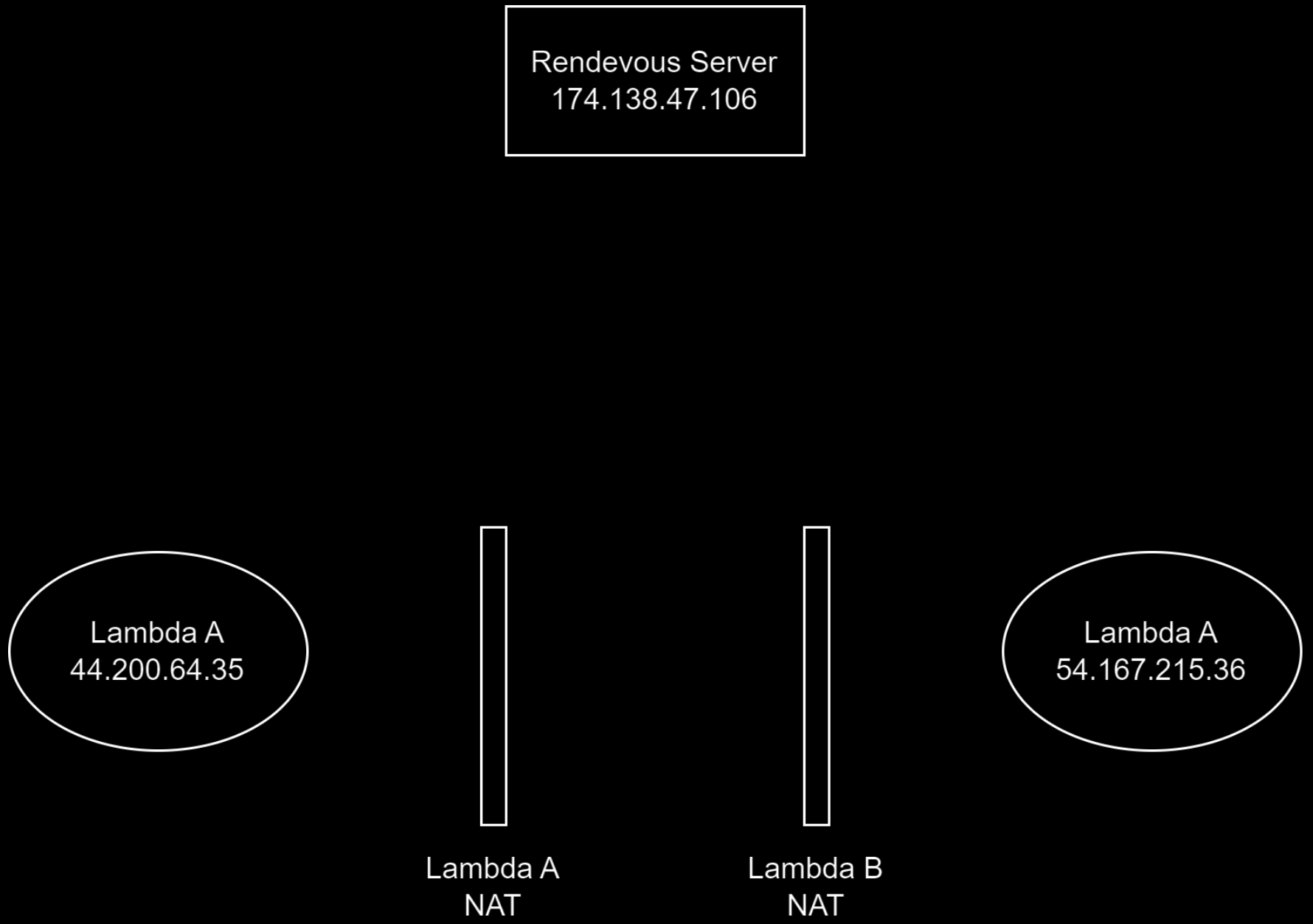

AWS S3
Lambda A
44.200.64.35
Lambda A
NAT
Lambda B
NAT
Lambda B
54.167.215.36

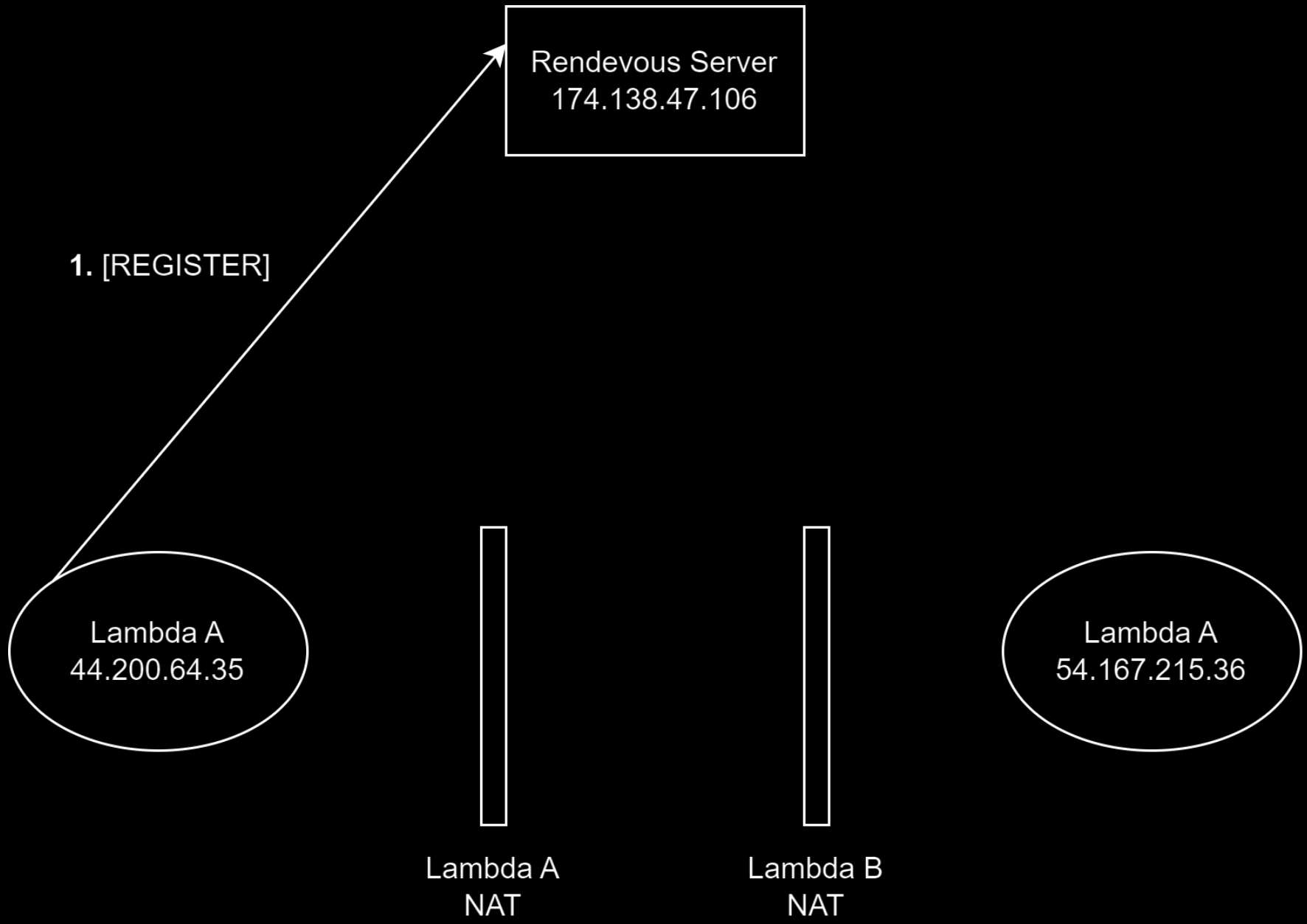

AWS S3
1. Upload
Lambda A
44.200.64.35
Lambda A
NAT
Lambda B
NAT
Lambda B
54.167.215.36

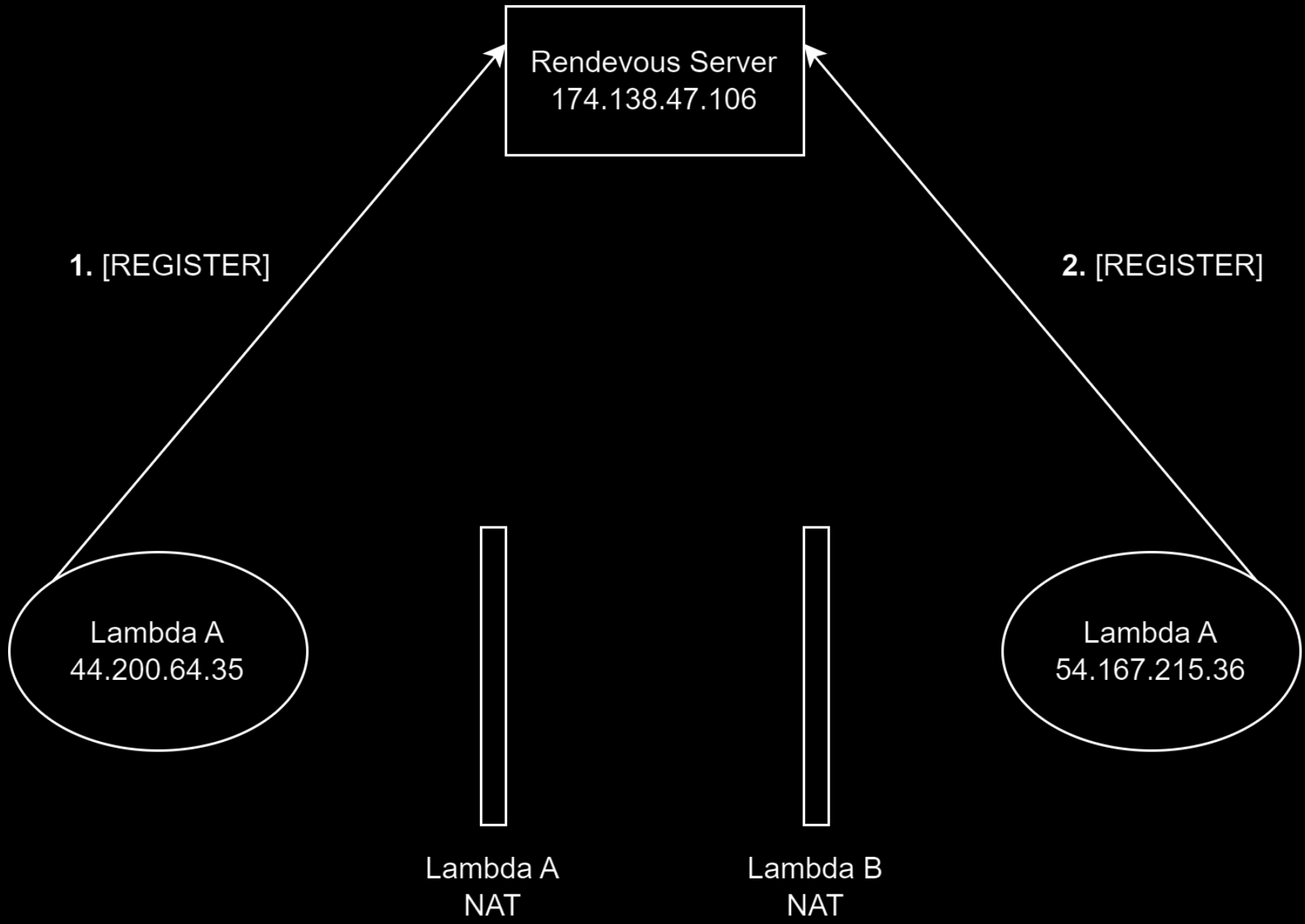

AWS S3
1. Upload
2. Download
Lambda A
44.200.64.35
Lambda B
54.167.215.36
Lambda A
NAT
Lambda B
NAT

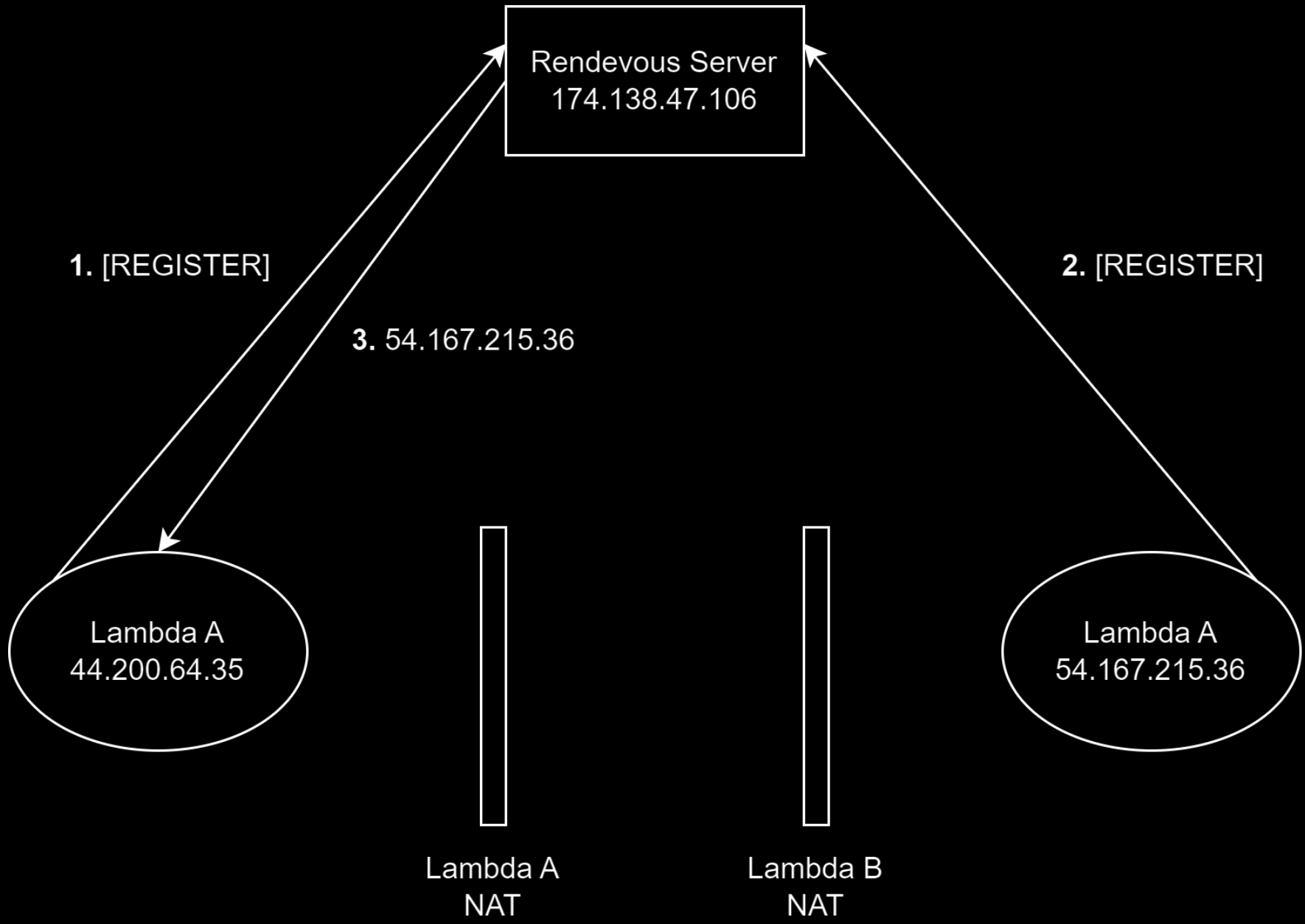
Hole Punching

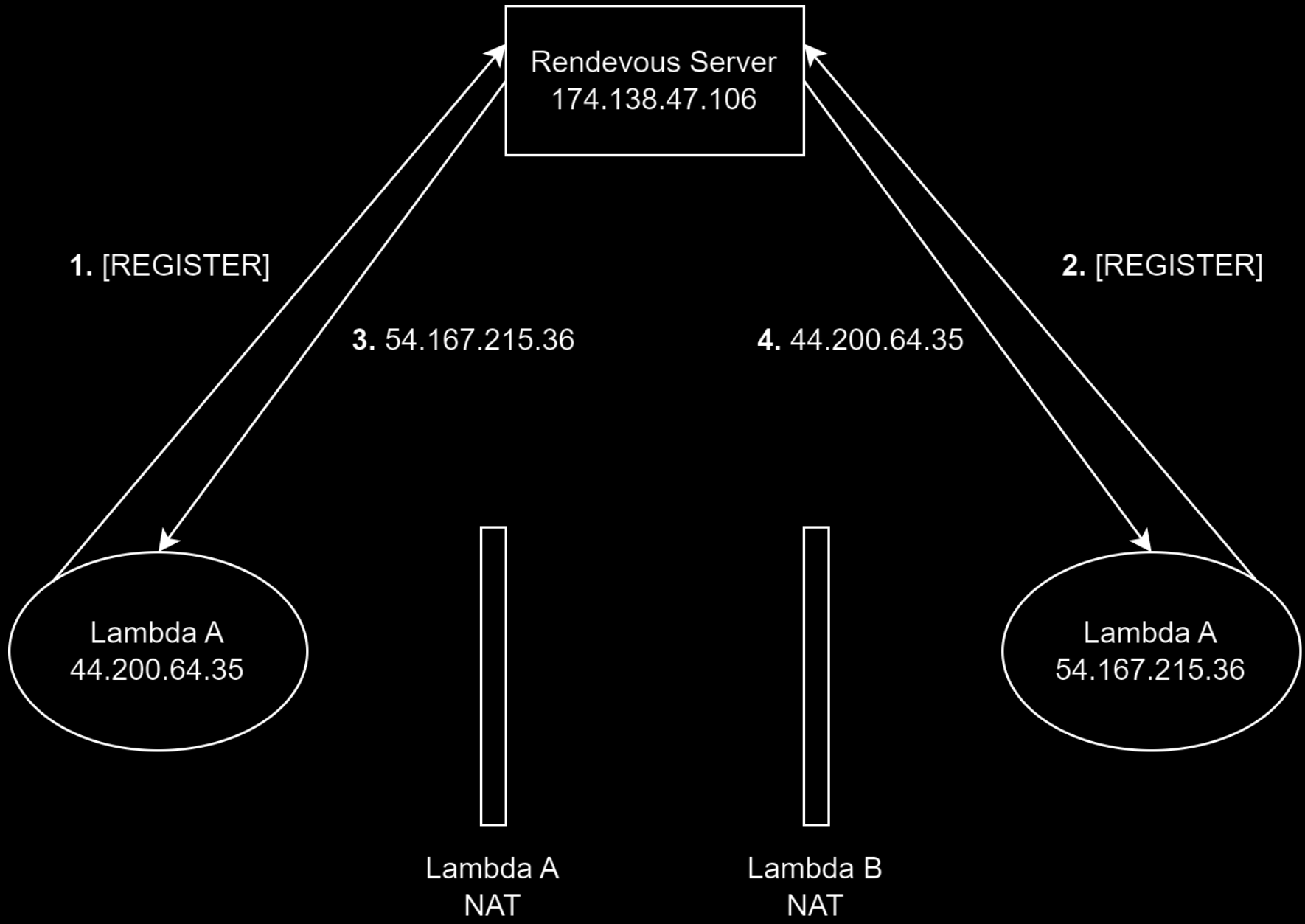
- Establishes a direct bi-directional link between NATed devices
- Popular technique in peer-to-peer applications
- Introduces **overhead** to establish a link
- Once established, a link is a standard UDP connection
- Maintaining a link requires keep-alive packets

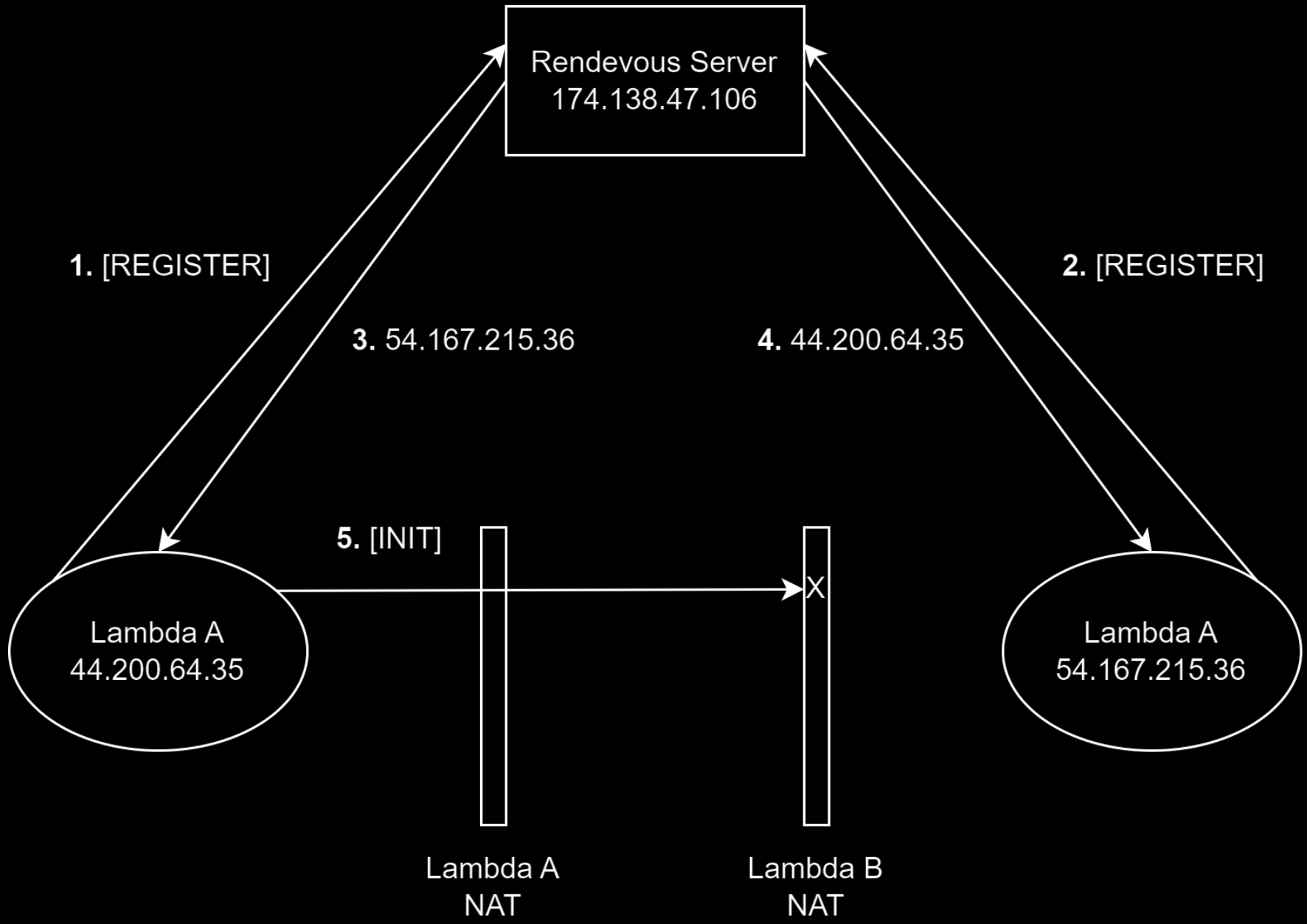

Rendevous Server
174.138.47.106
Lambda A
44.200.64.35
Lambda A
NAT
Lambda B
NAT
Lambda A
54.167.215.36

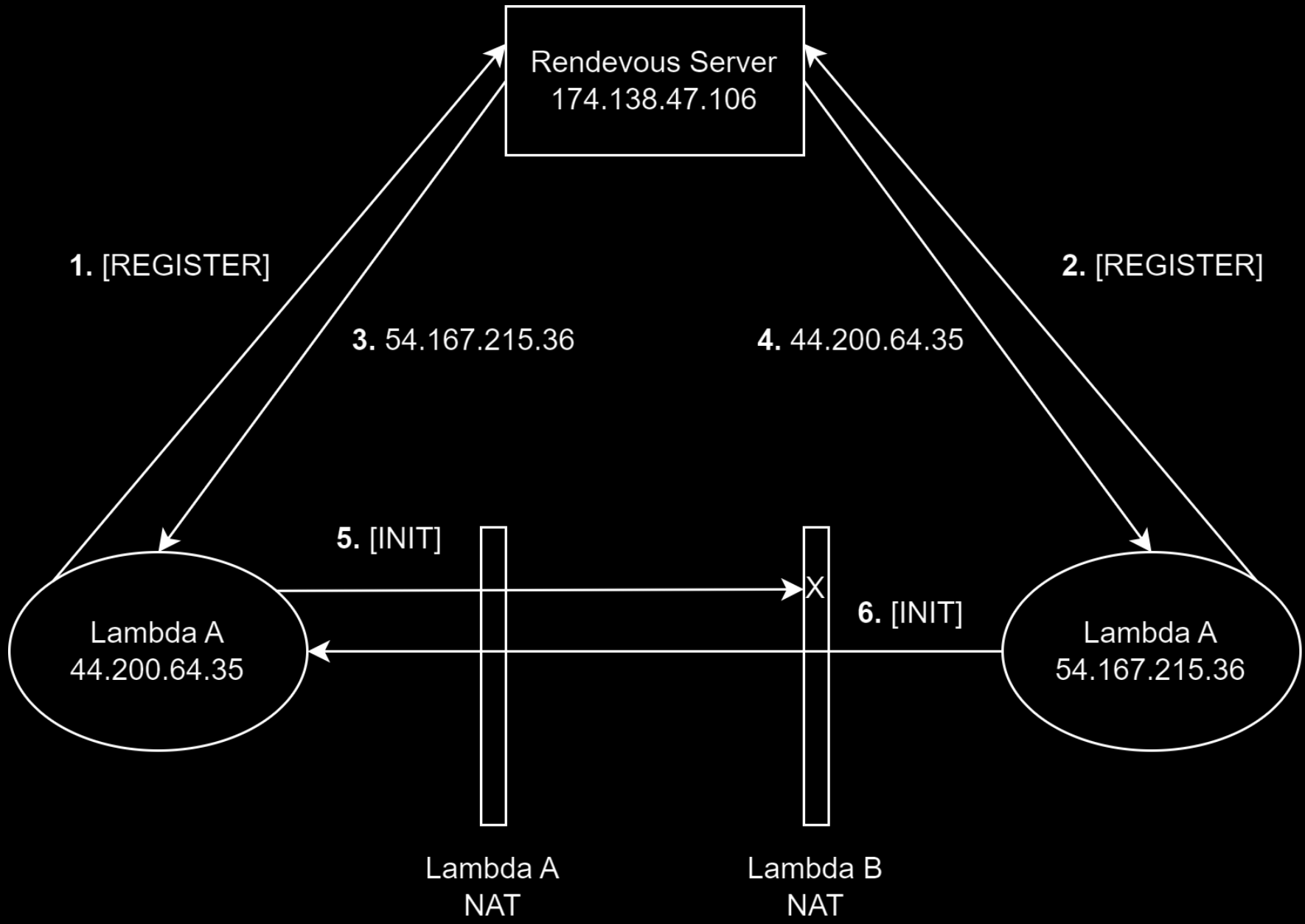

Rendevous Server
174.138.47.106
1. [REGISTER]
Lambda A
44.200.64.35
Lambda A
NAT
Lambda B
NAT
Lambda A
54.167.215.36

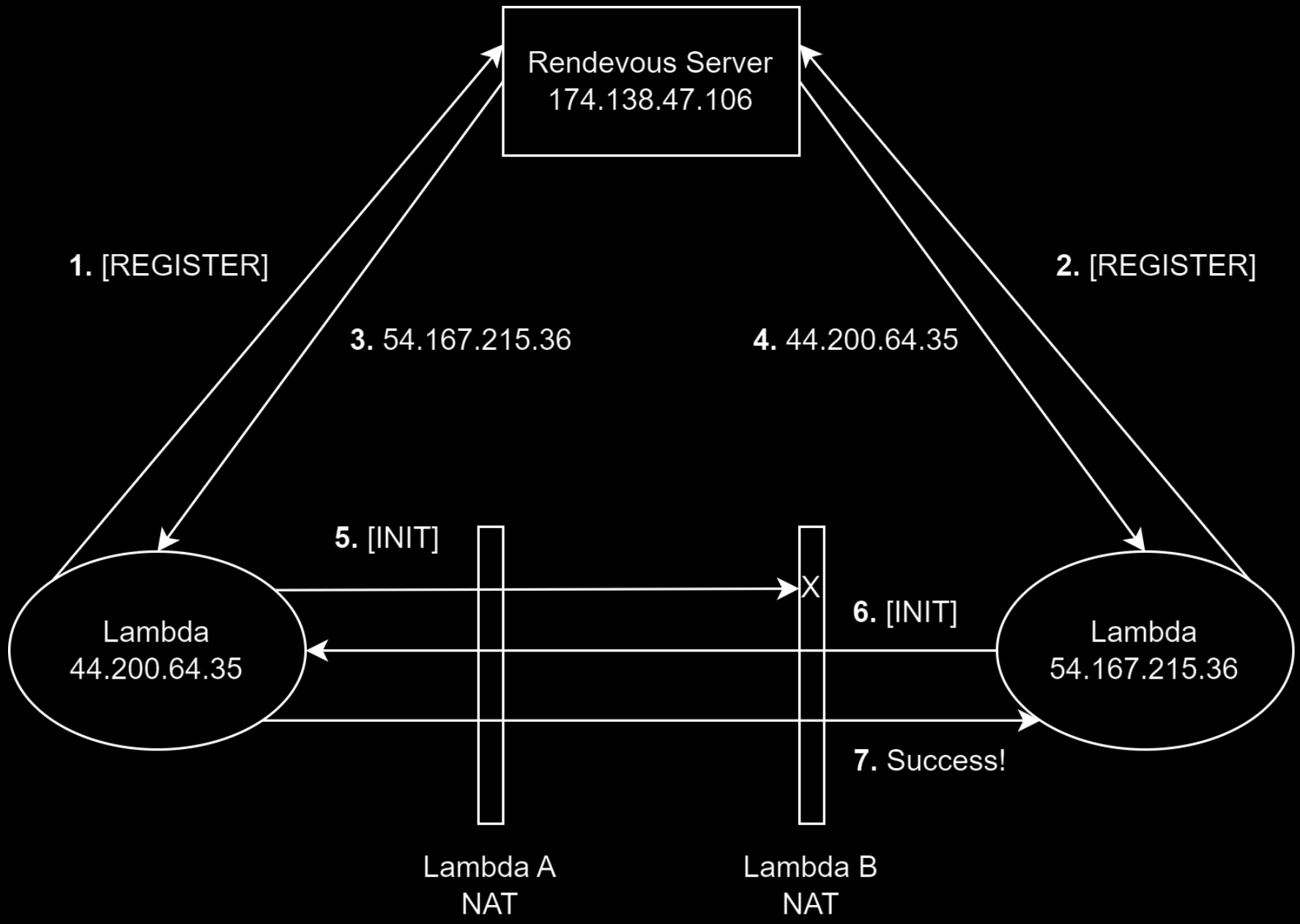

Rendevous Server
174.138.47.106
1. [REGISTER]
2. [REGISTER]
Lambda A
44.200.64.35
Lambda A
54.167.215.36
Lambda A
NAT
Lambda B
NAT

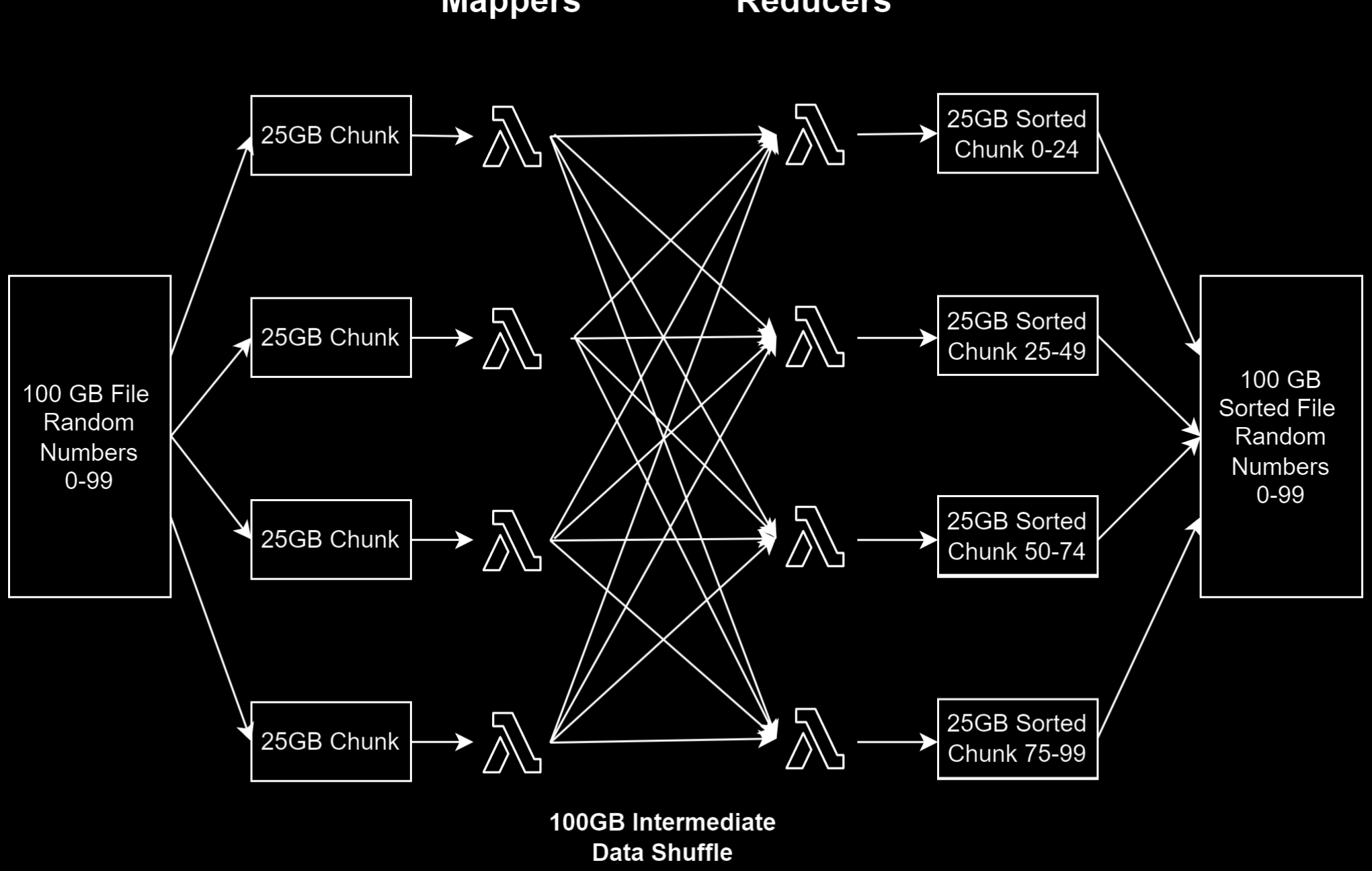

Rendevous Server
174.138.47.106
1. [REGISTER]
2. [REGISTER]
3. 54.167.215.36
Lambda A
44.200.64.35
Lambda A
54.167.215.36
Lambda A
NAT
Lambda B
NAT


Rendevous Server
174.138.47.106
1. [REGISTER]
2. [REGISTER]
3. 54.167.215.36
4. 44.200.64.35
Lambda A
44.200.64.35
Lambda A
54.167.215.36
Lambda A
NAT
Lambda B
NAT

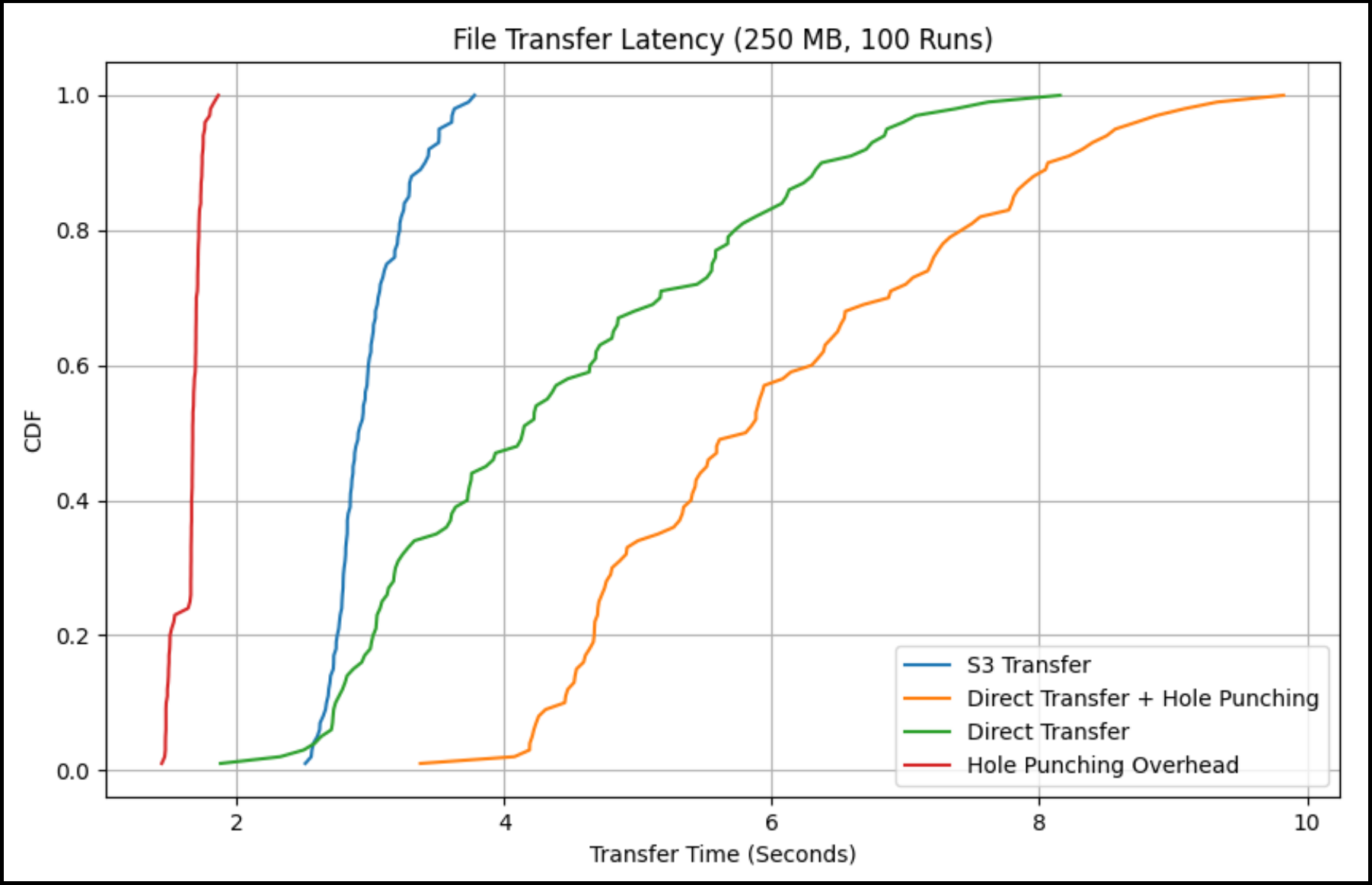

Rendevous Server
174.138.47.106
1. [REGISTER]
2. [REGISTER]
3. 54.167.215.36
4. 44.200.64.35
5. [INIT]
X
Lambda A
44.200.64.35
Lambda A
54.167.215.36
Lambda A
NAT
Lambda B
NAT


Rendevous Server
174.138.47.106
1. [REGISTER]
2. [REGISTER]
3. 54.167.215.36
4. 44.200.64.35
5. [INIT]
6. [INIT]
X
Lambda A
44.200.64.35
Lambda A
54.167.215.36
Lambda A
NAT
Lambda B
NAT

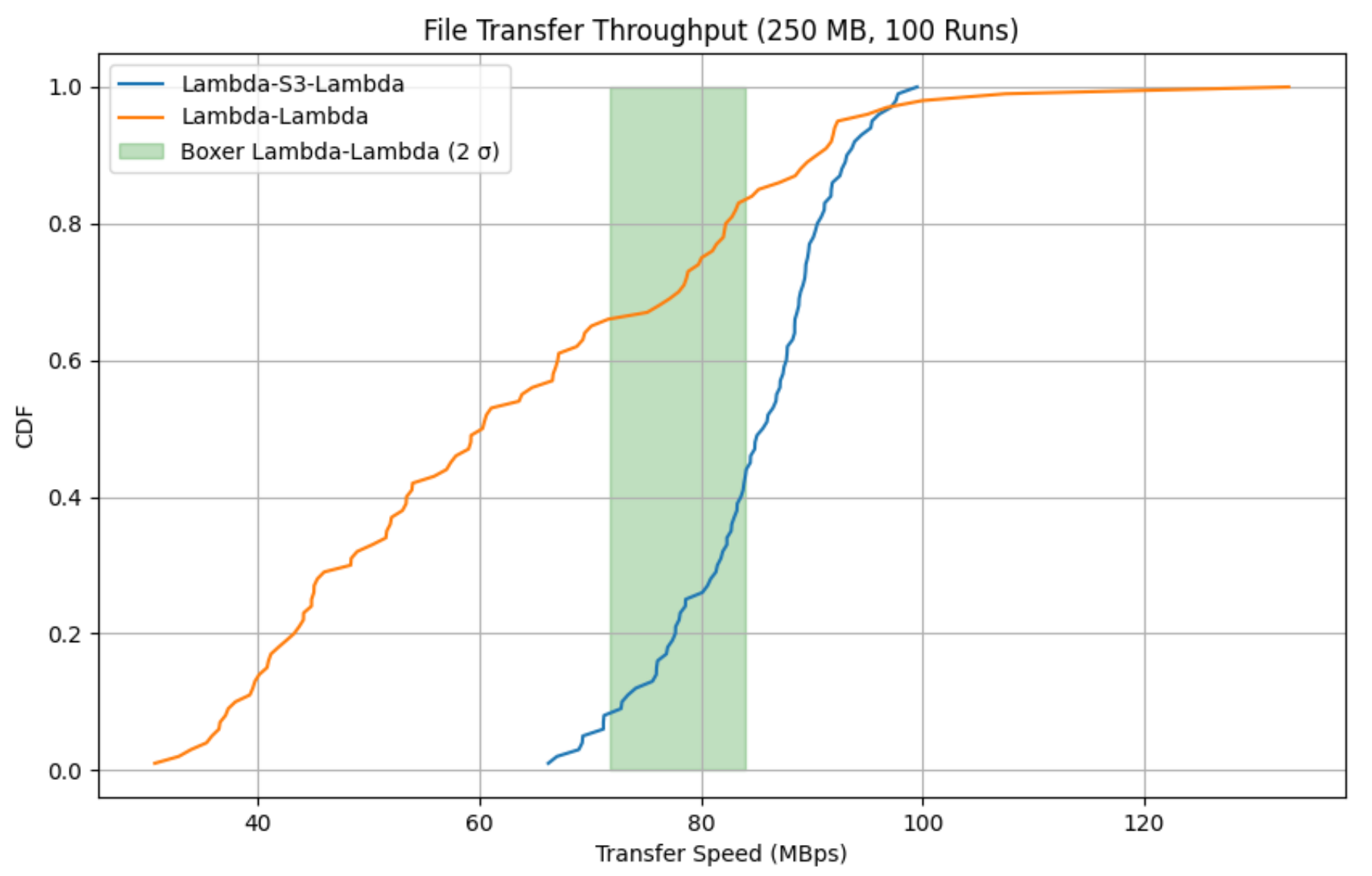

Rendevous Server
174.138.47.106
1. [REGISTER]
2. [REGISTER]
3. 54.167.215.36
4. 44.200.64.35
5. [INIT]
X
6. [INIT]
Lambda
44.200.64.35
Lambda
54.167.215.36
7. Success!
Lambda A
NAT
Lambda B
NAT


Mappers
Reducers
100 GB File Random Numbers 0-99
25GB Chunk
25GB Chunk
25GB Chunk
25GB Chunk
25GB Sorted Chunk 0-24
25GB Sorted Chunk 25-49
25GB Sorted Chunk 50-74
25GB Sorted Chunk 75-99
100 GB Sorted File Random Numbers 0-99
100GB Intermediate Data Shuffle

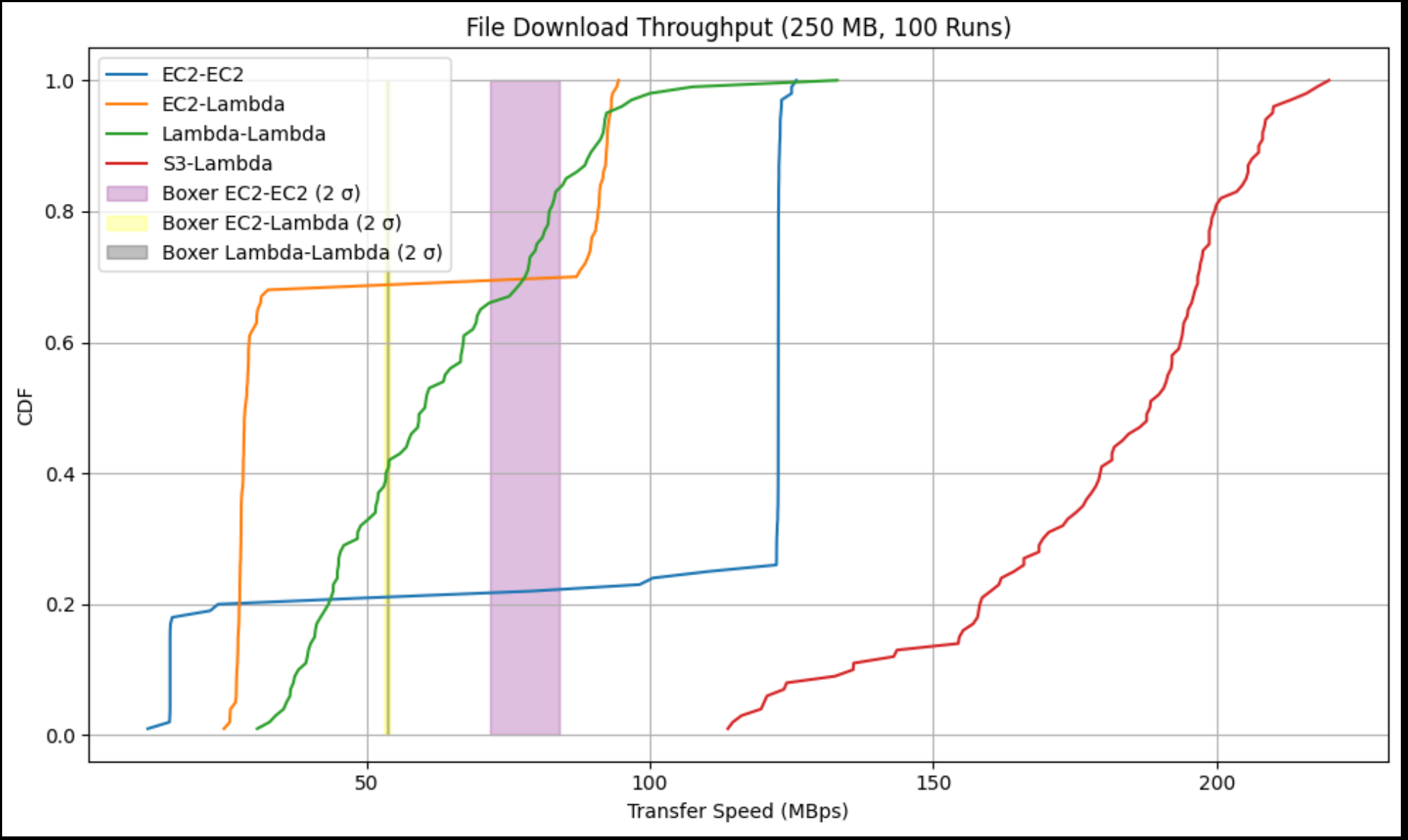
Performance Comparison



Performance Comparison



Performance Comparison



Battle Scars

- UDP hole punching is very sensitive to synchronization
  - All [INIT]s must occur within ~500ms of each other
  - Our lambdas **must** run in the same datacenter
- S3 performance improves with parallel uploads and downloads
  - However, rate limited to 5500 GET requests per second per file
- Reliable, in-order packet delivery over UDP is difficult
  - Don't try to implement your own protocol, use QUIC

Comparison with AWS EMR

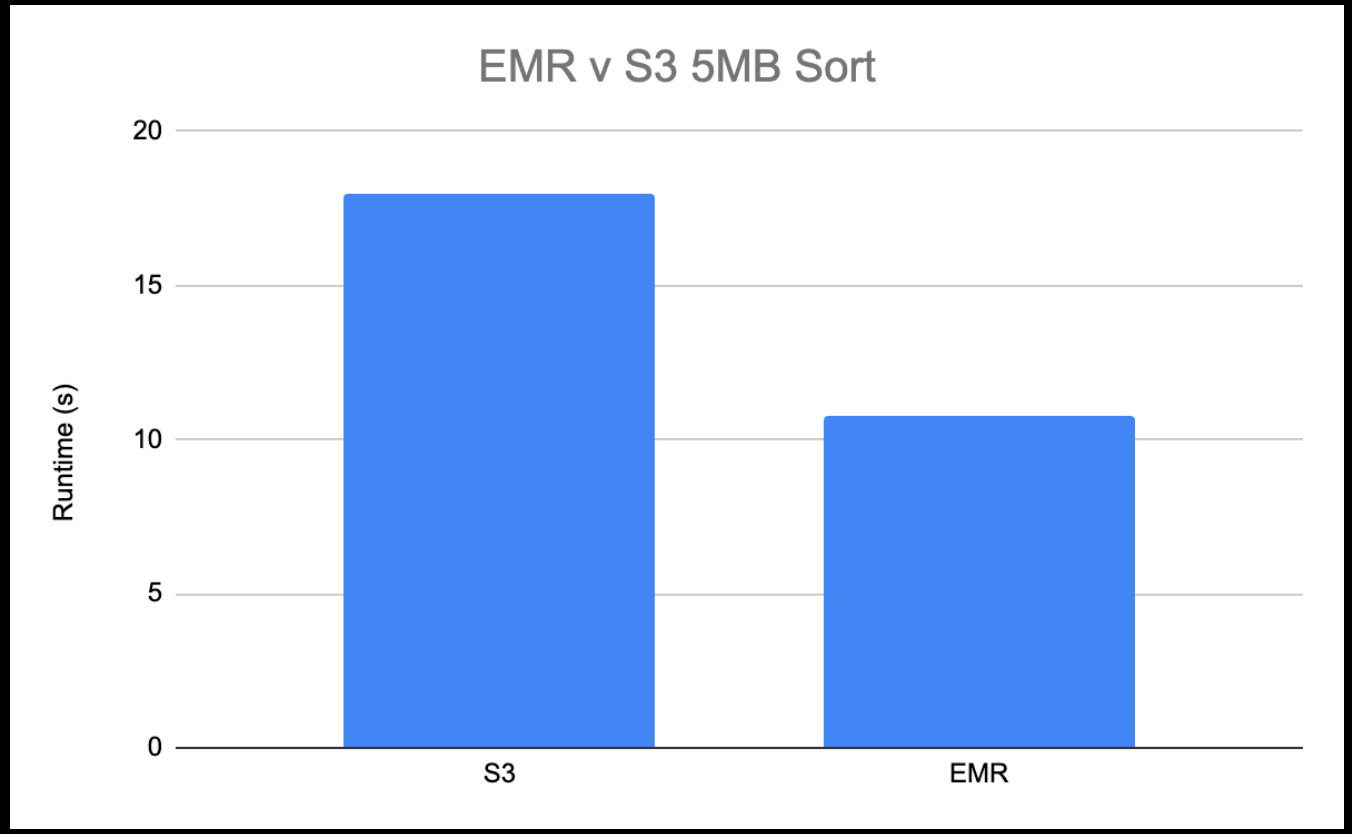
Amazon AWS EMR Serverless

- Elastic MapReduce (EMR)
  - Managed cloud service
  - No need to provision
- Run jobs without provisioning
  - Determines required resources, allocates resources and releases them on completion

EMR Serverless Communication

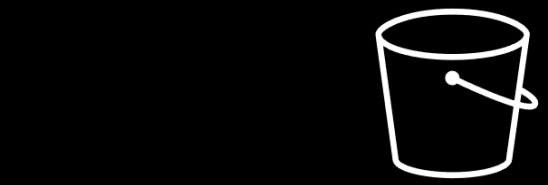
- Communication
  - S3 used to communicate
  - Limited peer-to-peer communication under the hood, not configurable
  - Uses RPCs (remote procedure calls)
- Implements TCP/IP communication
  - Using the default ElasticMapReduce-primary for allows the core and task nodes to communicate with each other over TCP

AWS EMR vs Lambda S3



Making it Resilient

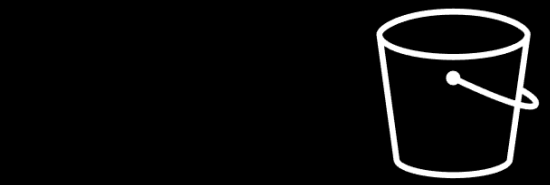
Fault Tolerance Example



Mapper

Reducer

Fault Tolerance Example

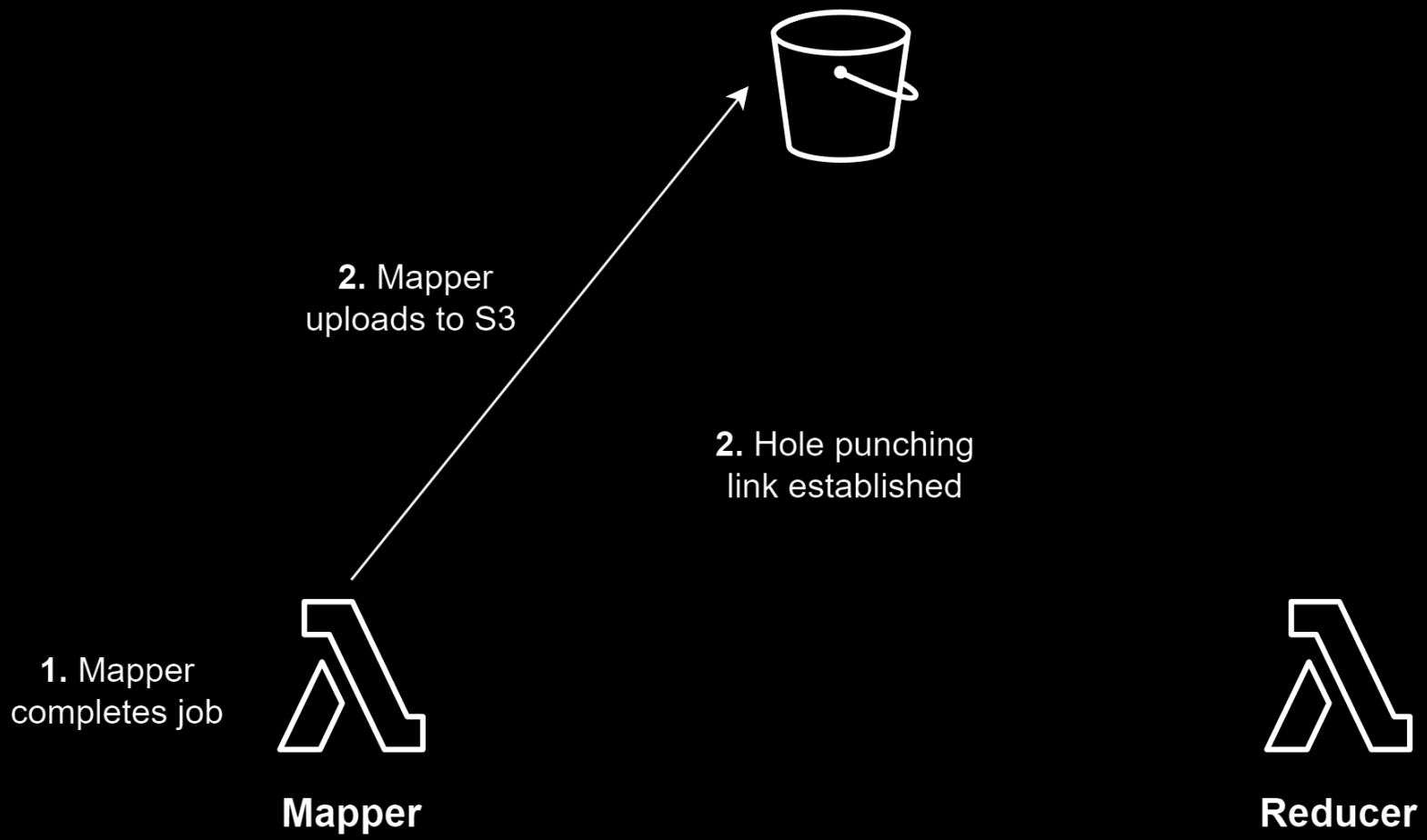


1. Mapper completes job

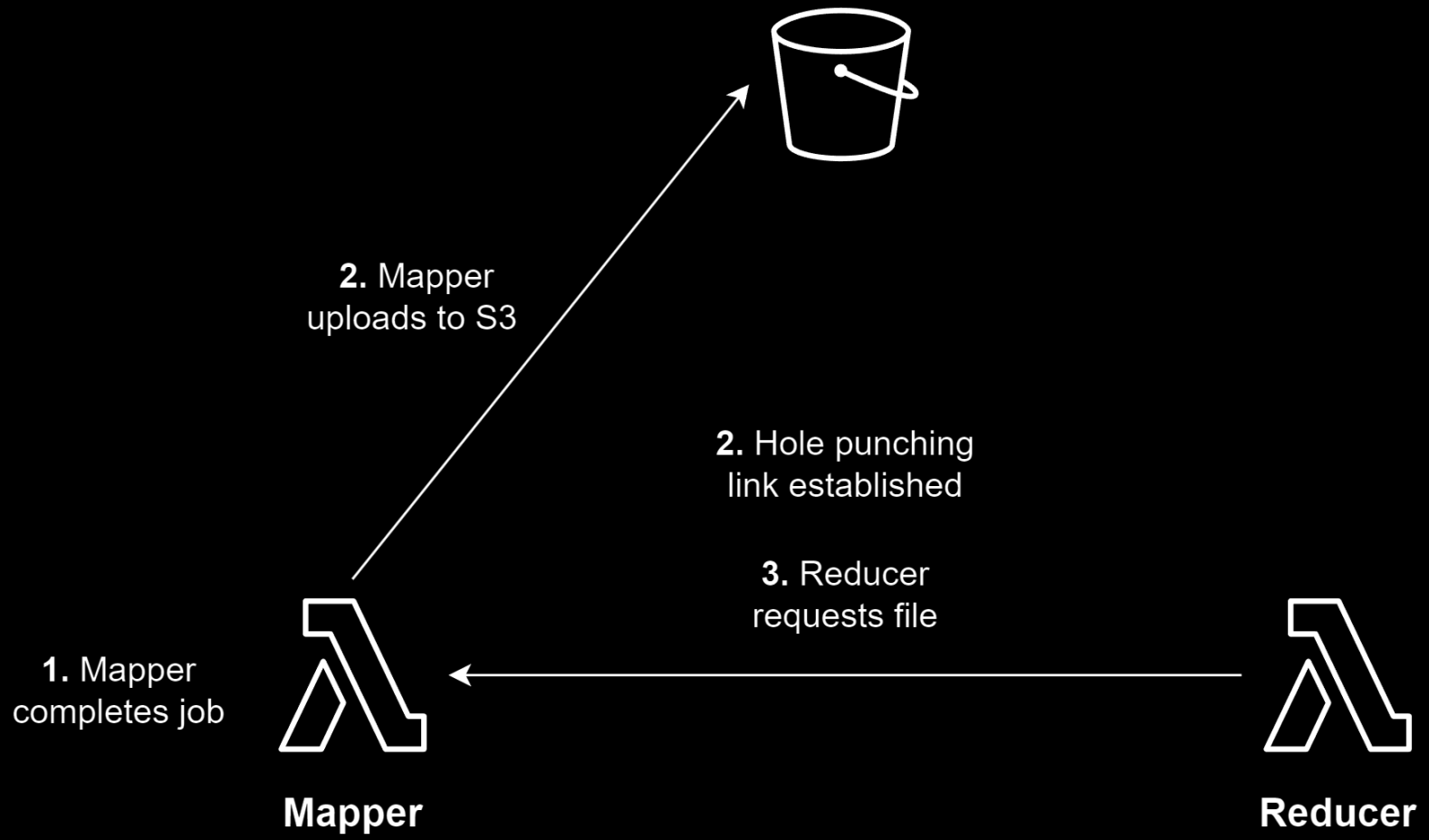
Mapper

Reducer

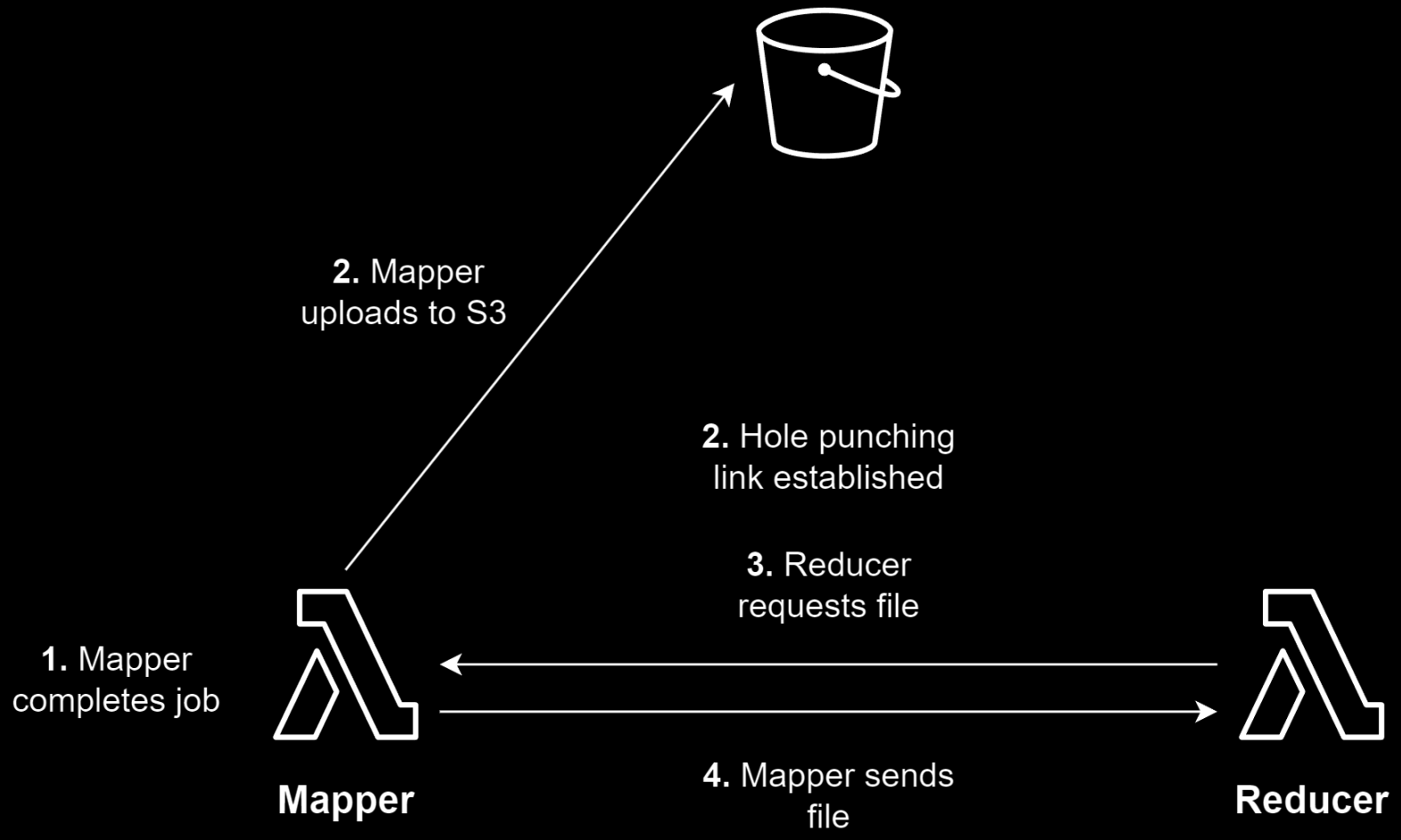
Fault Tolerance Example



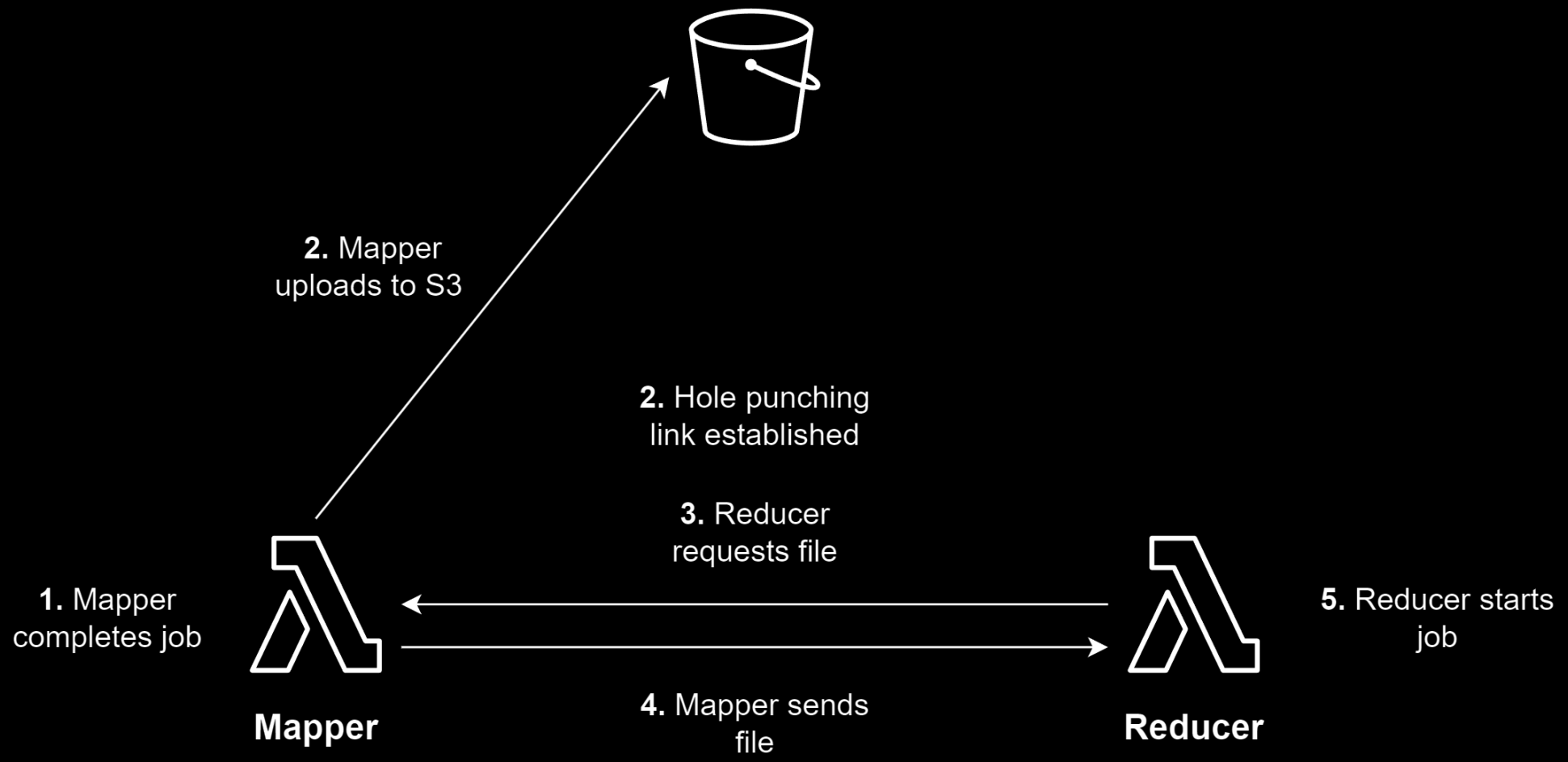
Fault Tolerance Example



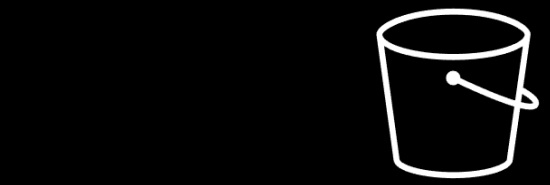
Fault Tolerance Example



Fault Tolerance Example



Fault Tolerance Example

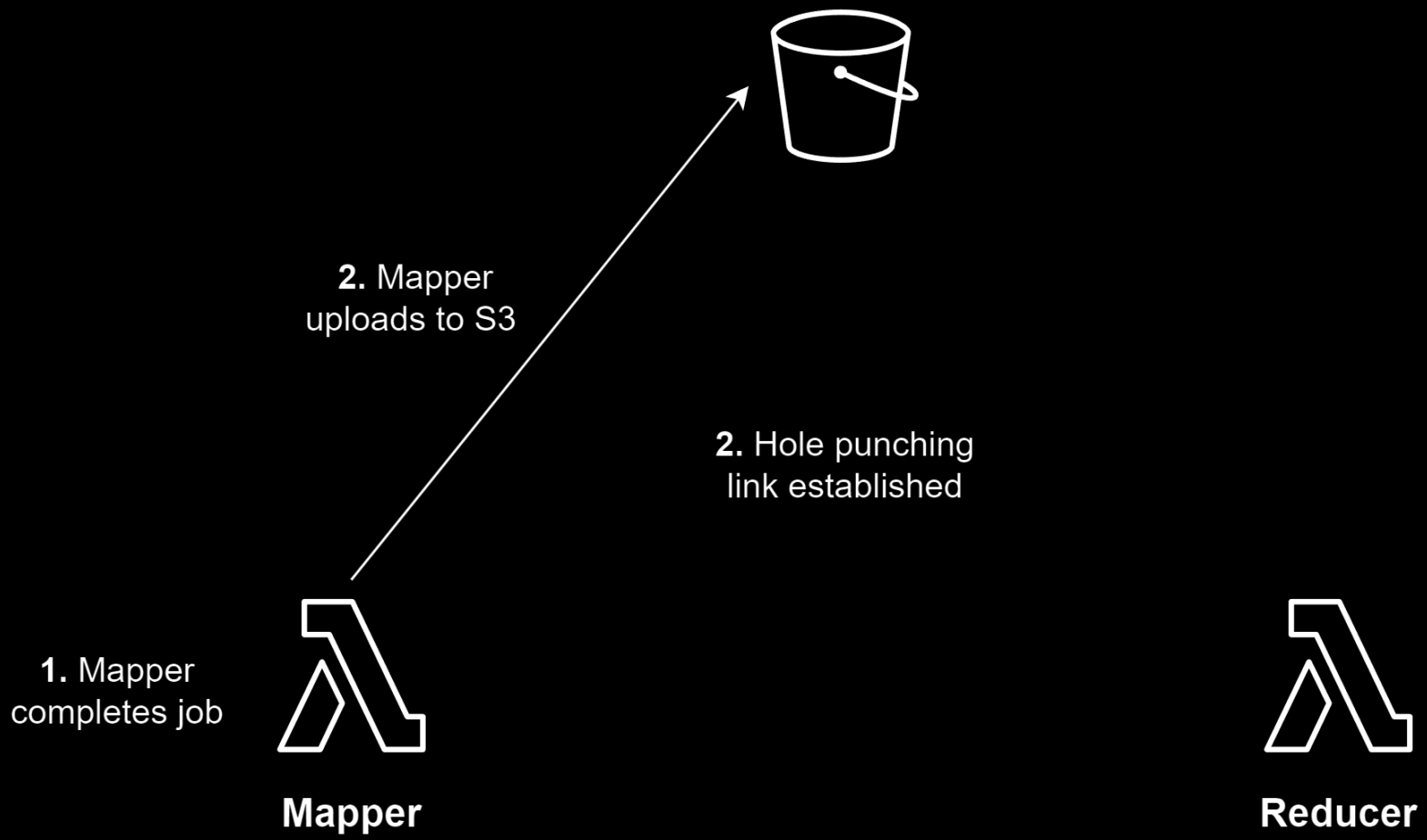


1. Mapper completes job

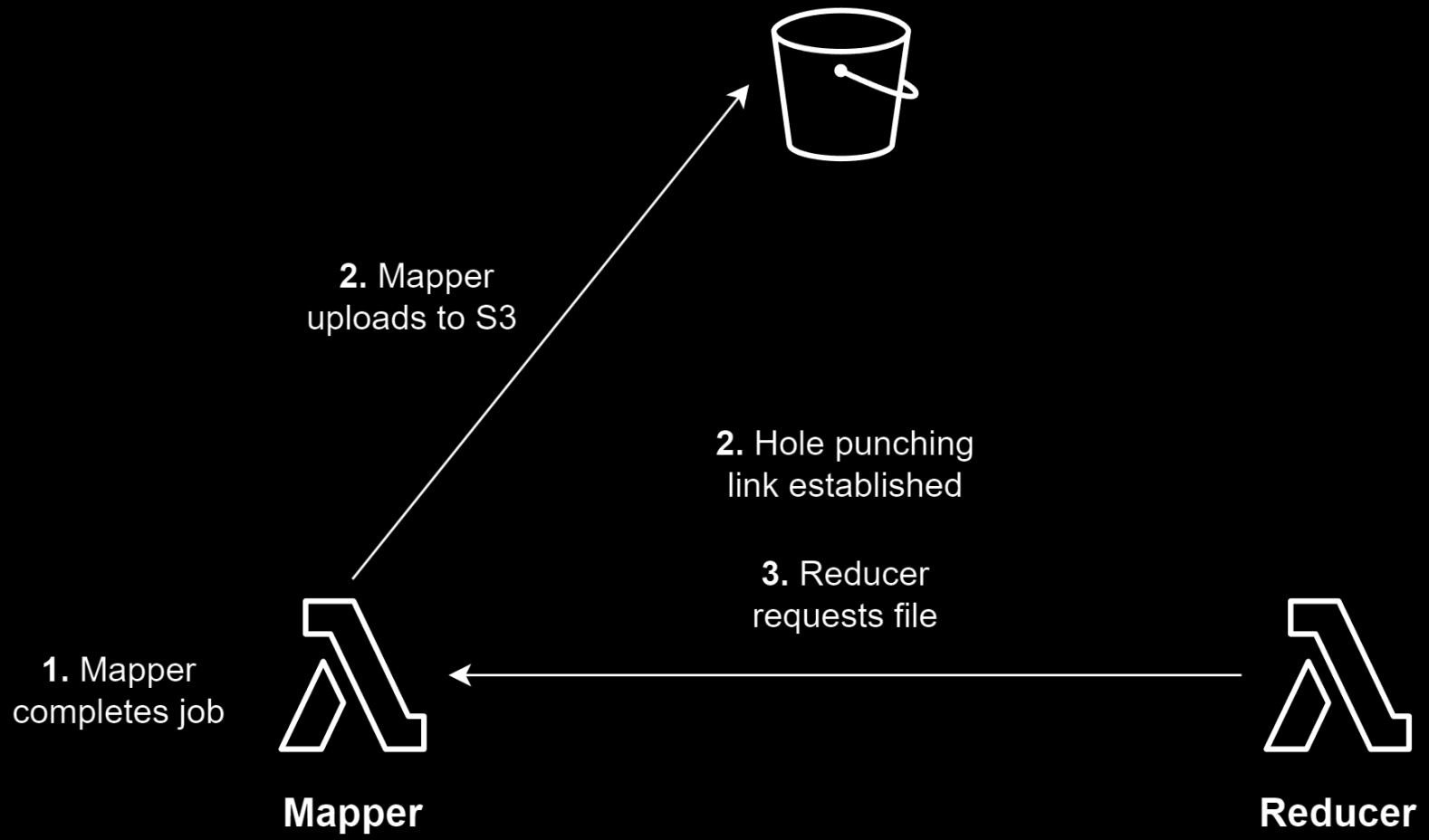
Mapper

Reducer

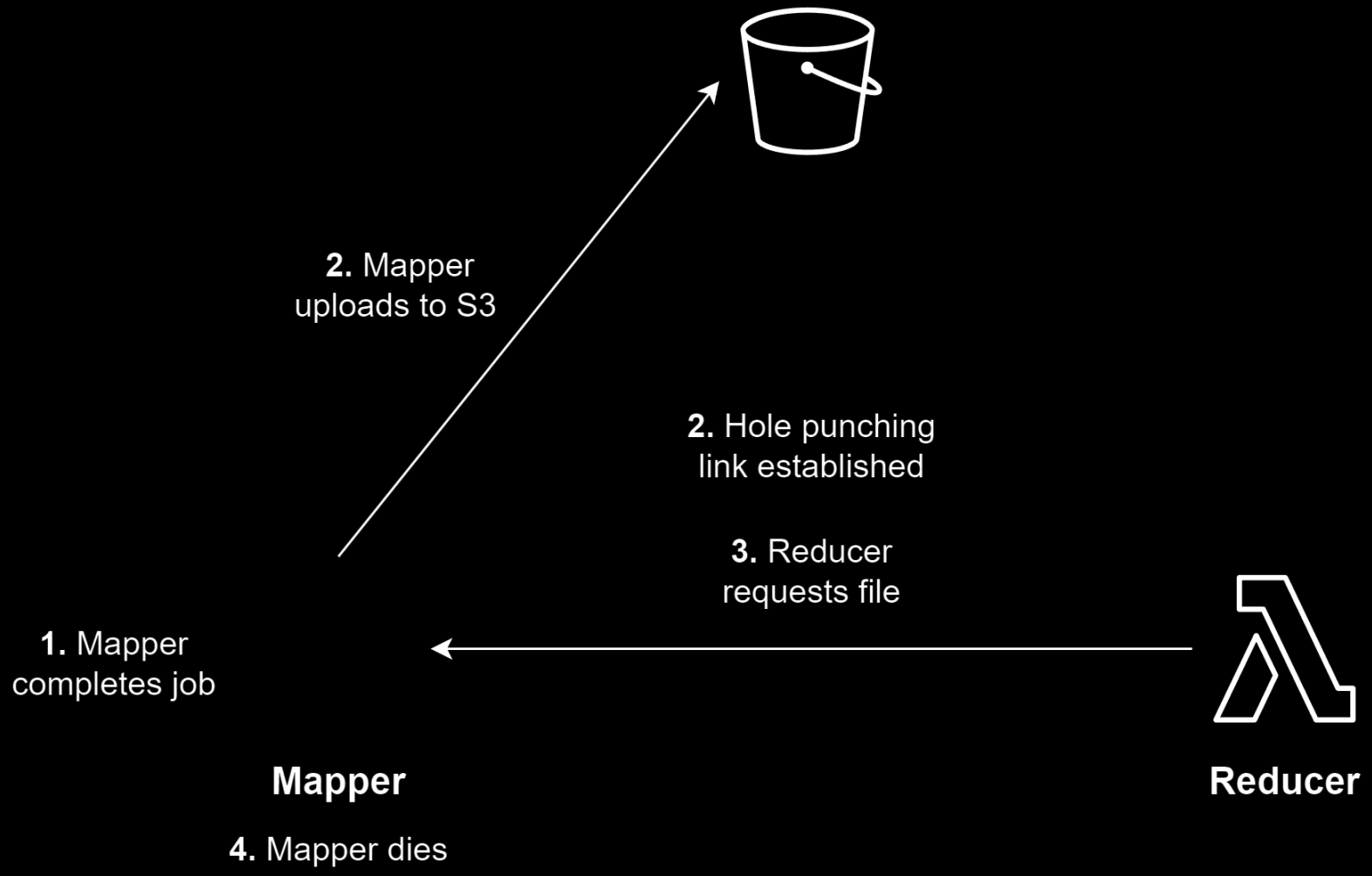
Fault Tolerance Example



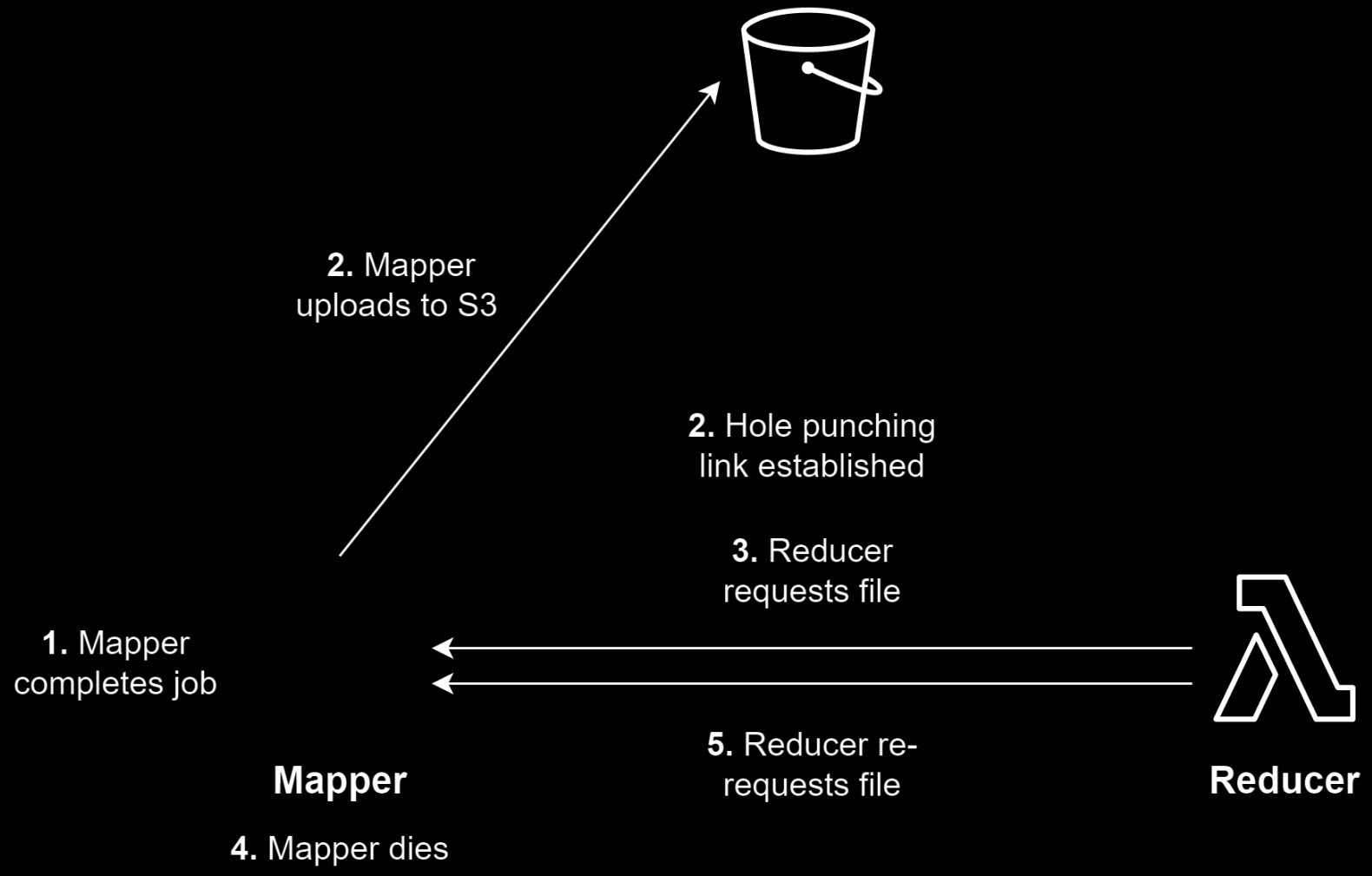
Fault Tolerance Example



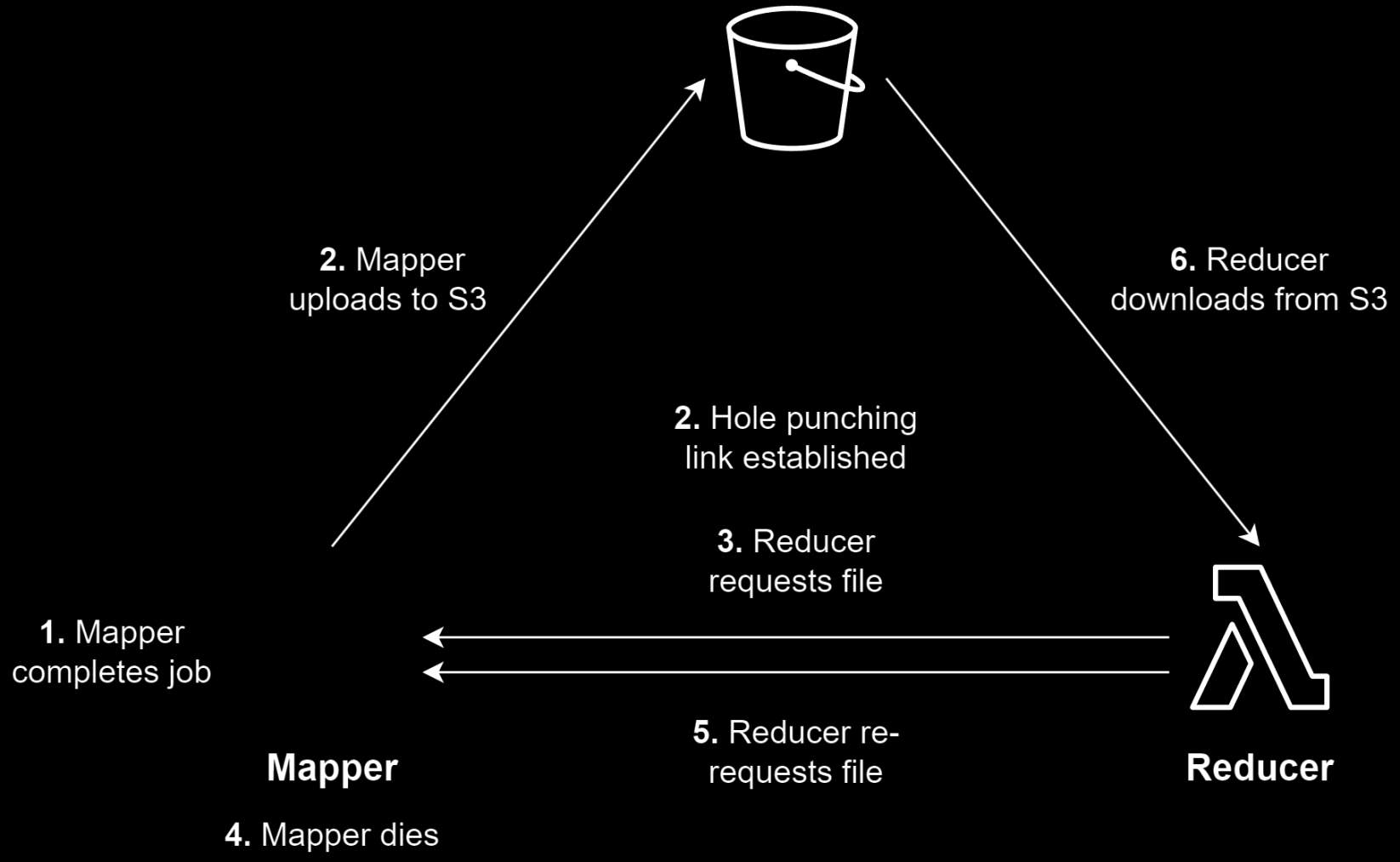
Fault Tolerance Example



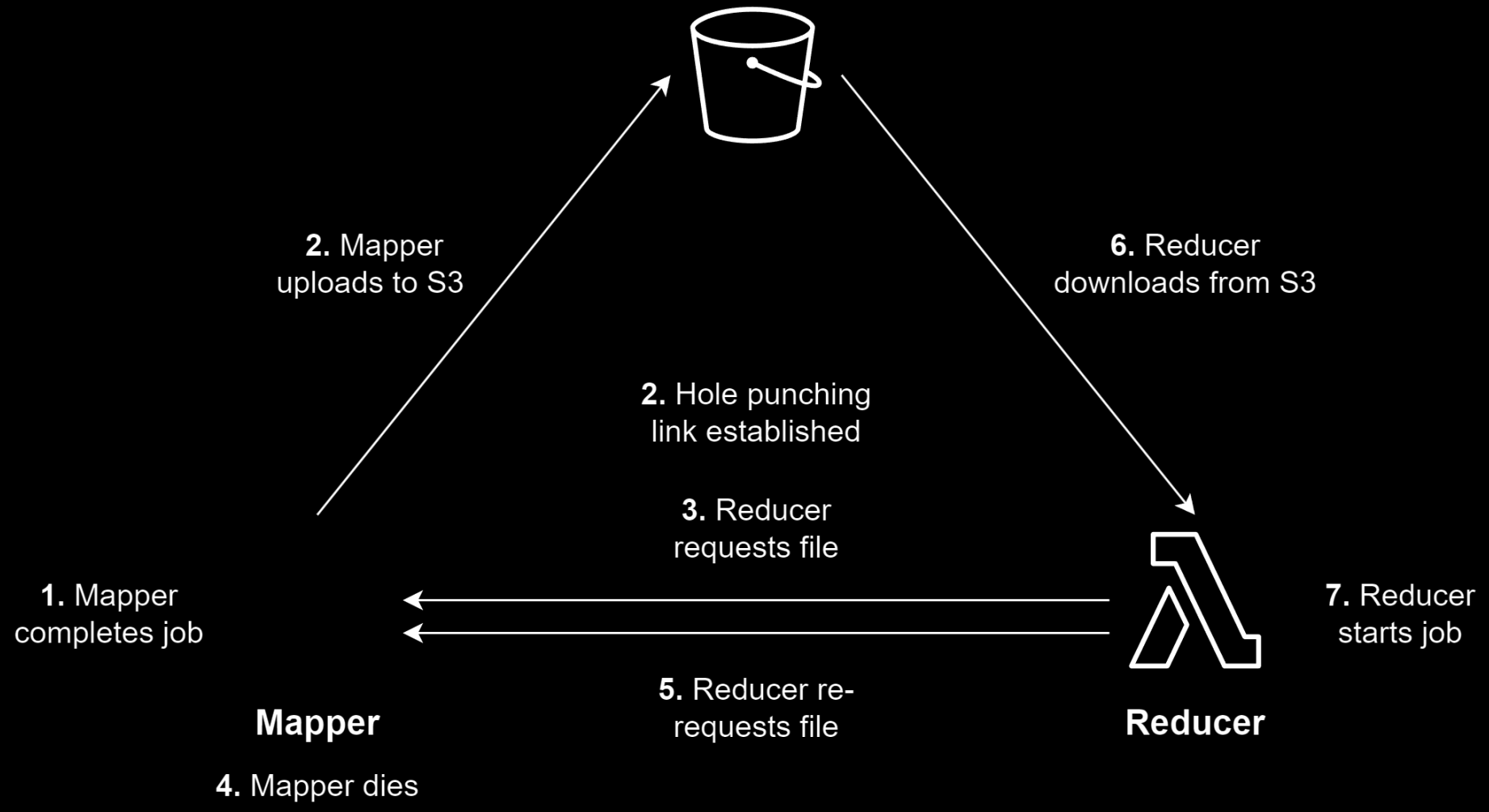
Fault Tolerance Example



Fault Tolerance Example



Fault Tolerance Example



Fault Tolerant Interfaces

Performance Centric

- Define a ***time to sync***
  - How long should the sender wait before writing to S3?
  - Navigates a durability-performance-cost tradefoff
- Define a ***time to live***
  - How long should the mapper stick around waiting for reducers to request data

Cost Centric

- Keep track of the **cost of compute** since the last checkpoint
- Sync once that value crosses a user defined **threshold**

Conclusion and Future Work

- Serverless is in need of **first-class network support**
- Through fault-tolerant communication, network support can be **aligned with the values of serverless**
- Providers like AWS **heavily optimize** their **paid services**
- We expect that the AWS paid tier would have better network performance
- Exploring different sized messages would be interesting
- Our implementation could be further optimized