Nona: A Stochastic Congestion-Aware Job Scheduler for Real-Time Inference Queries

Benoit Pit-Claudel*, Derya Malak†, Alejandro Cohen‡, Muriel Médard*, Manya Ghobadi*

** MIT, † EURECOM, ‡ Technion.*

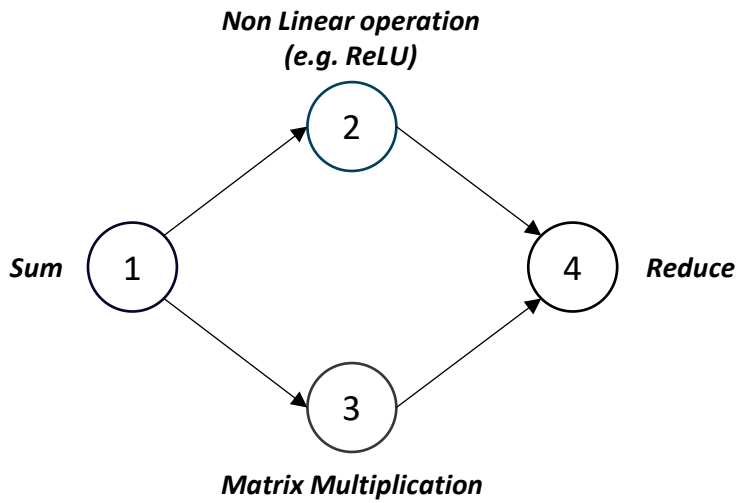
Nov 28th, 2024

Real-time Inference Requests are Flooding Datacenters

- ChatGPT serves in the order of 10 million inference queries per day
- 2024’s growth of AI unit at Microsoft is “all inference”
- Interactive workloads have strict latency requirements

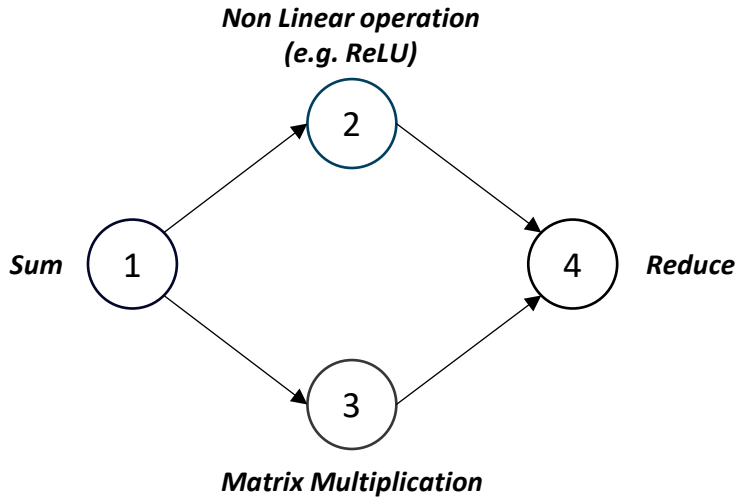
What is scheduling?

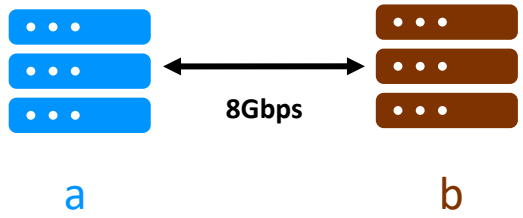
- Users submit *jobs*, or Directed Acyclic Graphs (DAG), of operations to the cluster.
- Along with the input data, this specifies what the cluster should do, and how to obtain the result.



What is scheduling?

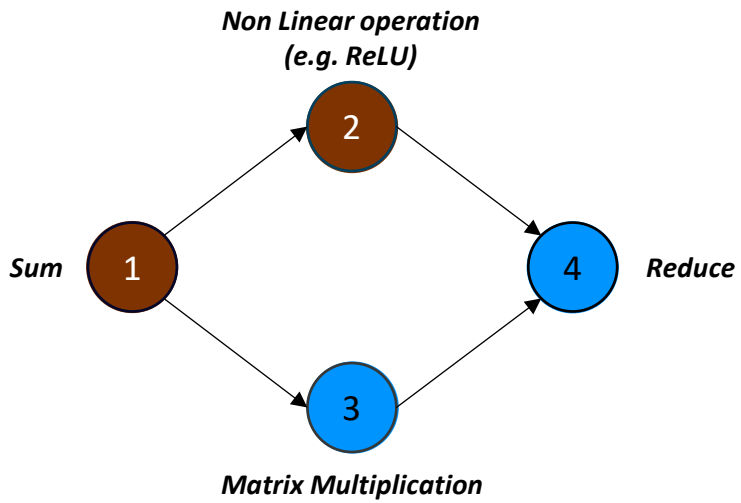
- Users submit *jobs*, or Directed Acyclic Graphs (DAG), of operations to the cluster.
- Along with the input data, this specifies what the cluster should do, and how to obtain the result.
- Scheduling is mapping these operations on compute resources.

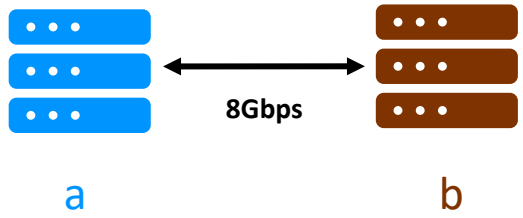




What is scheduling?

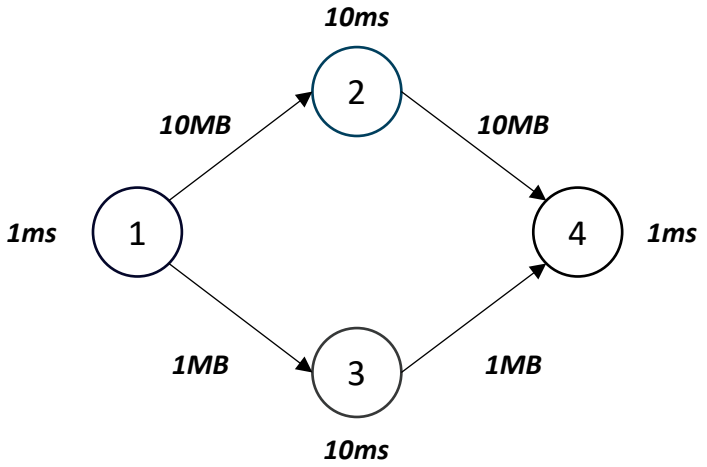
- Users submit *jobs*, or Directed Acyclic Graphs (DAG), of operations to the cluster.
- Along with the input data, this specifies what the cluster should do, and how to obtain the result.
- Scheduling is mapping these operations on compute resources.

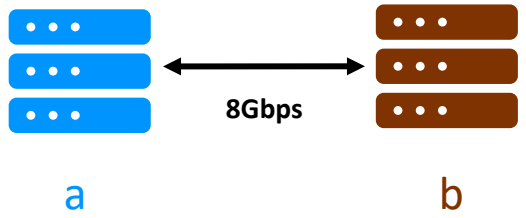




Scheduling matters - Example

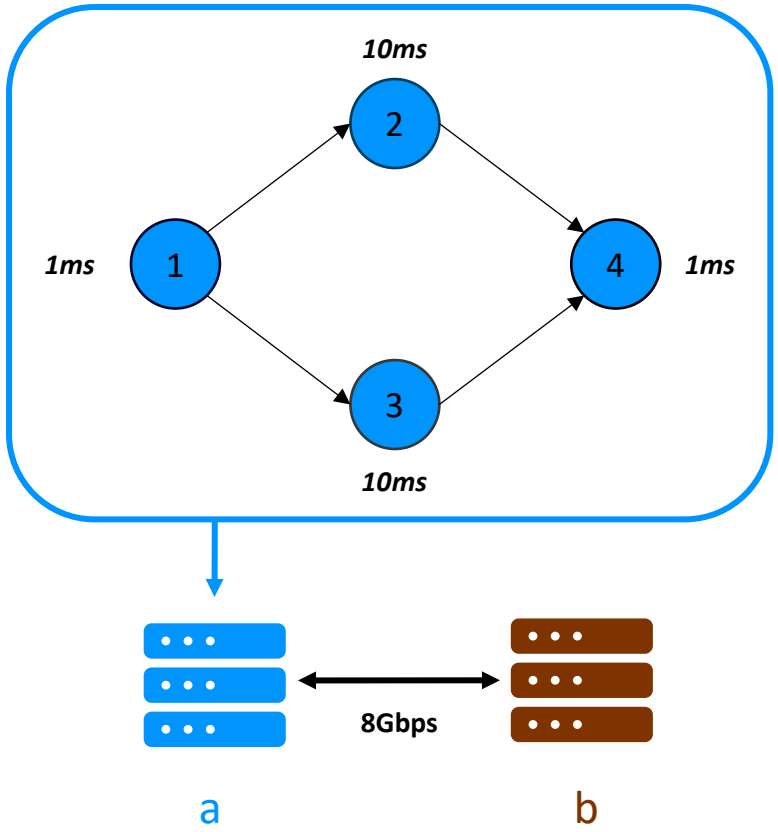
- Some operations can run in parallel. This saves time.
- Edges in DAGs denote data dependencies. If two consecutive nodes are on different servers, they need to exchange data. This transfers can get quite large.
- In this example:





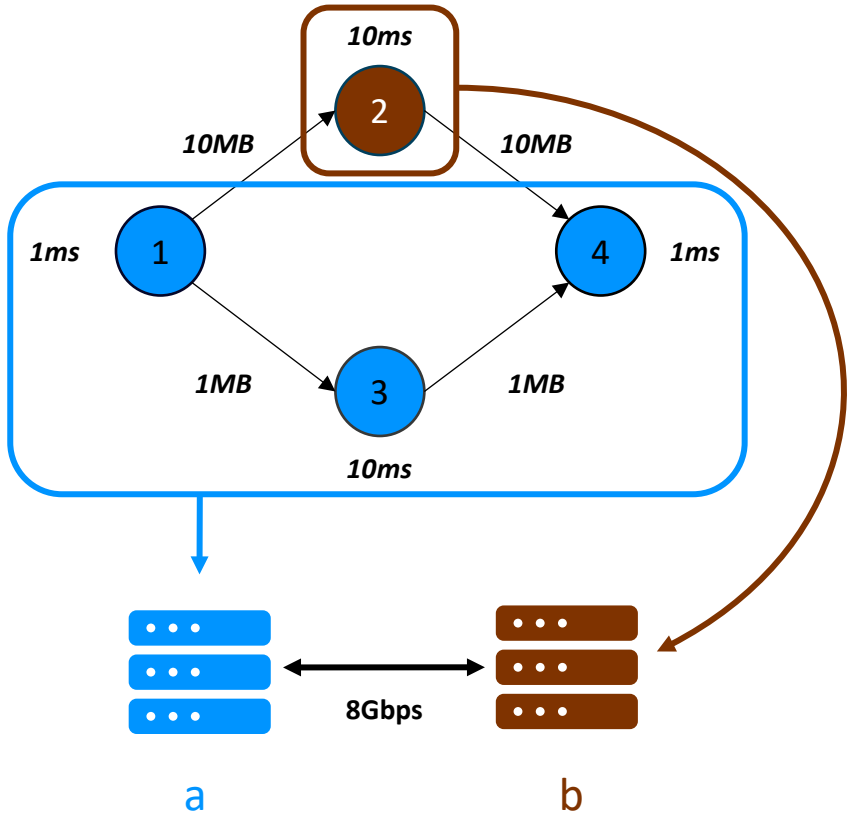
Scheduling matters - Example

- Some operations can run in parallel. This saves time.
- Edges in DAGs denote data dependencies. If two consecutive nodes are on different servers, they need to exchange data. This transfers can get quite large.
- In this example:
  - Place everything on the same server: 22 ms



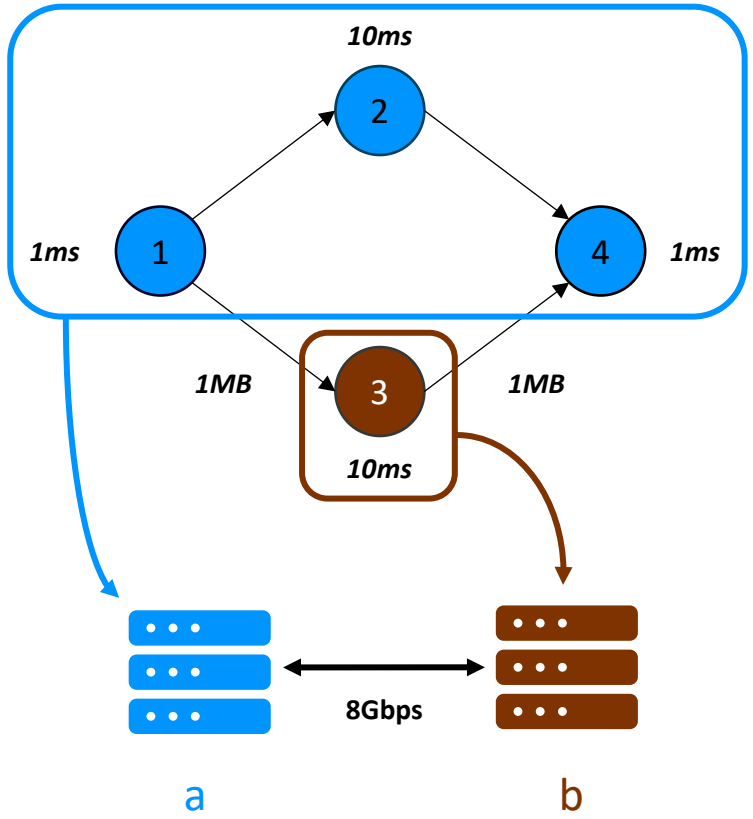
Scheduling matters - Example

- Some operations can run in parallel. This saves time.
- Edges in DAGs denote data dependencies. If two consecutive nodes are on different servers, they need to exchange data. This transfers can get quite large.
- In this example:
  - Place everything on the same server: 22 ms
  - Distribute (network-unaware): 32 ms



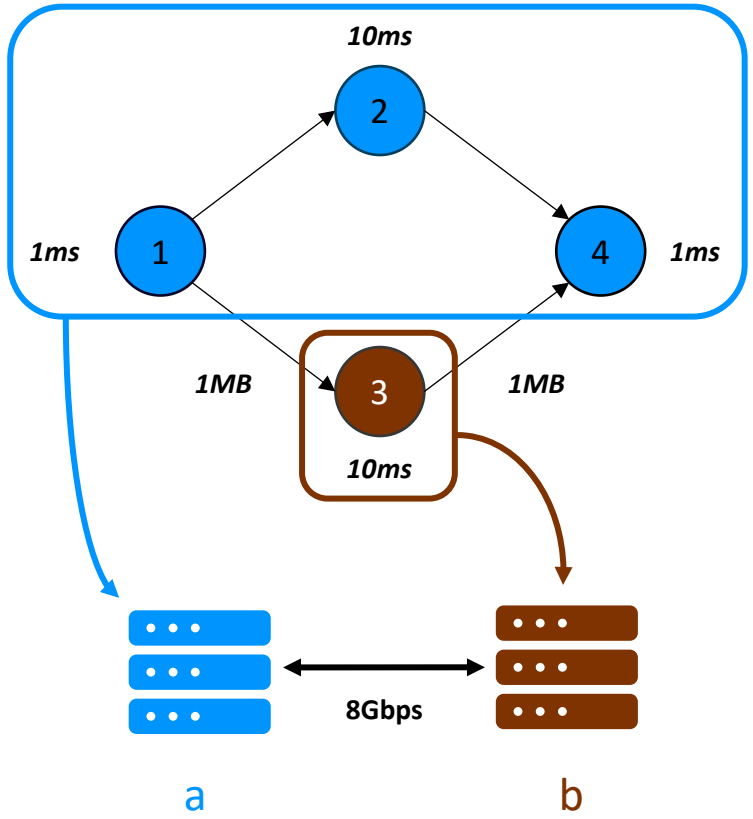
Scheduling matters - Example

- Some operations can run in parallel. This saves time.
- Edges in DAGs denote data dependencies. If two consecutive nodes are on different servers, they need to exchange data. This transfers can get quite large.
- In this example:
  - Place everything on the same server: 22 ms
  - Distribute (network-unaware): 32 ms
  - **Distribute (network-aware): 14 ms**



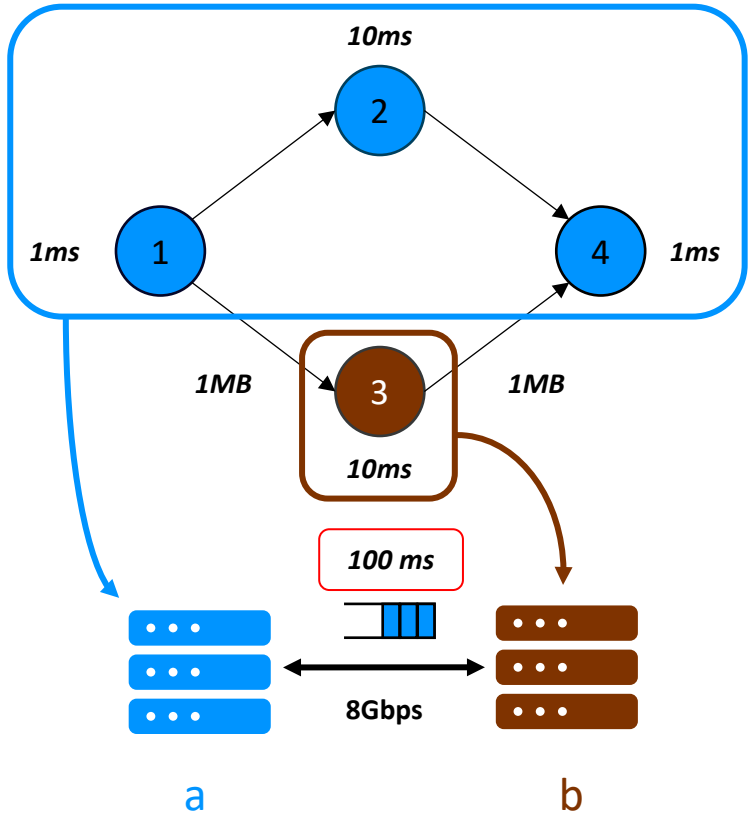
Scheduling matters - Example

- Datacenter is shared with other applications



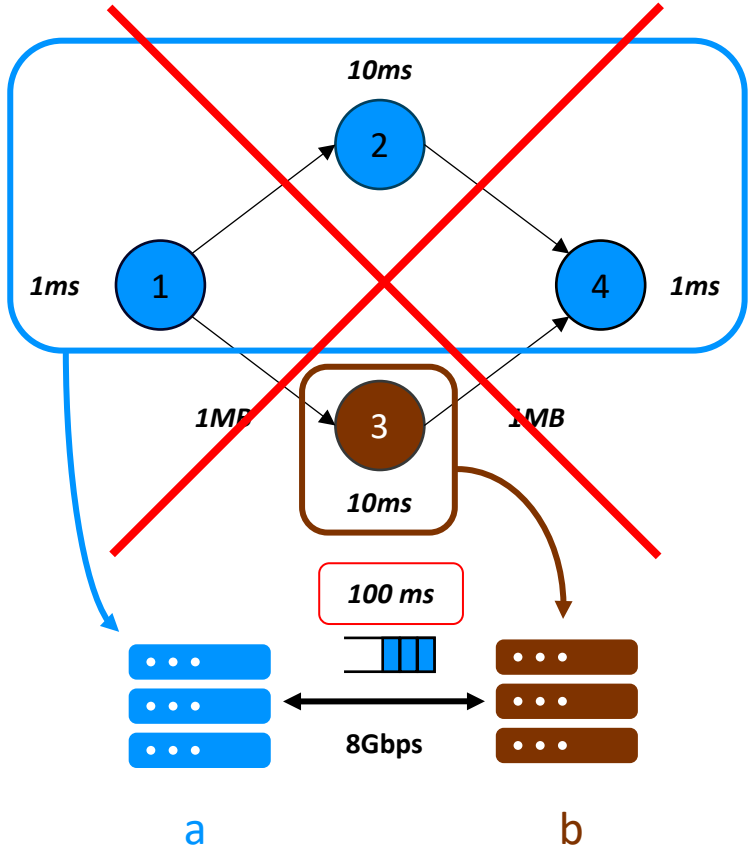
Scheduling matters - Example

- Datacenter is shared with other applications



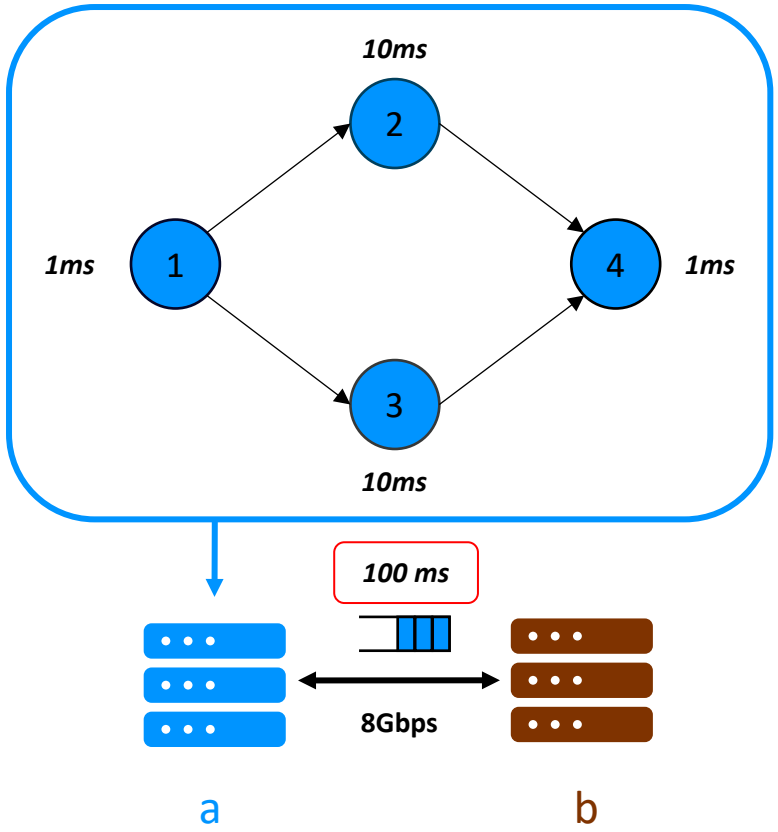
Scheduling matters - Example

- Datacenter is shared with other applications



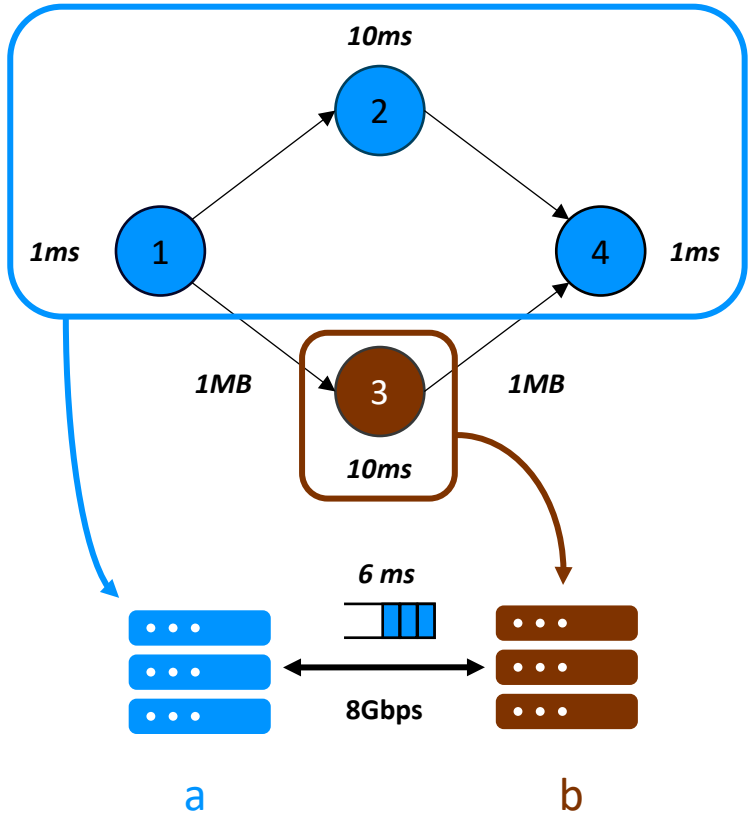
Scheduling matters - Example

- Datacenter is shared with other applications



Scheduling matters - Example

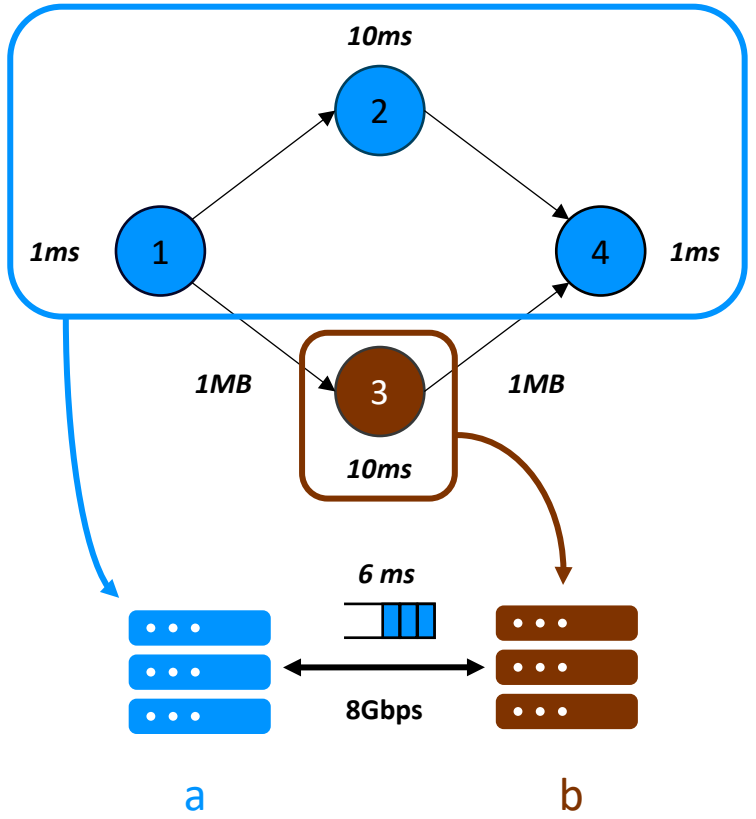
- Datacenter is shared with other applications
- When scheduling flows of jobs, other jobs in the flow contribute to queueing delay



Scheduling matters - Example

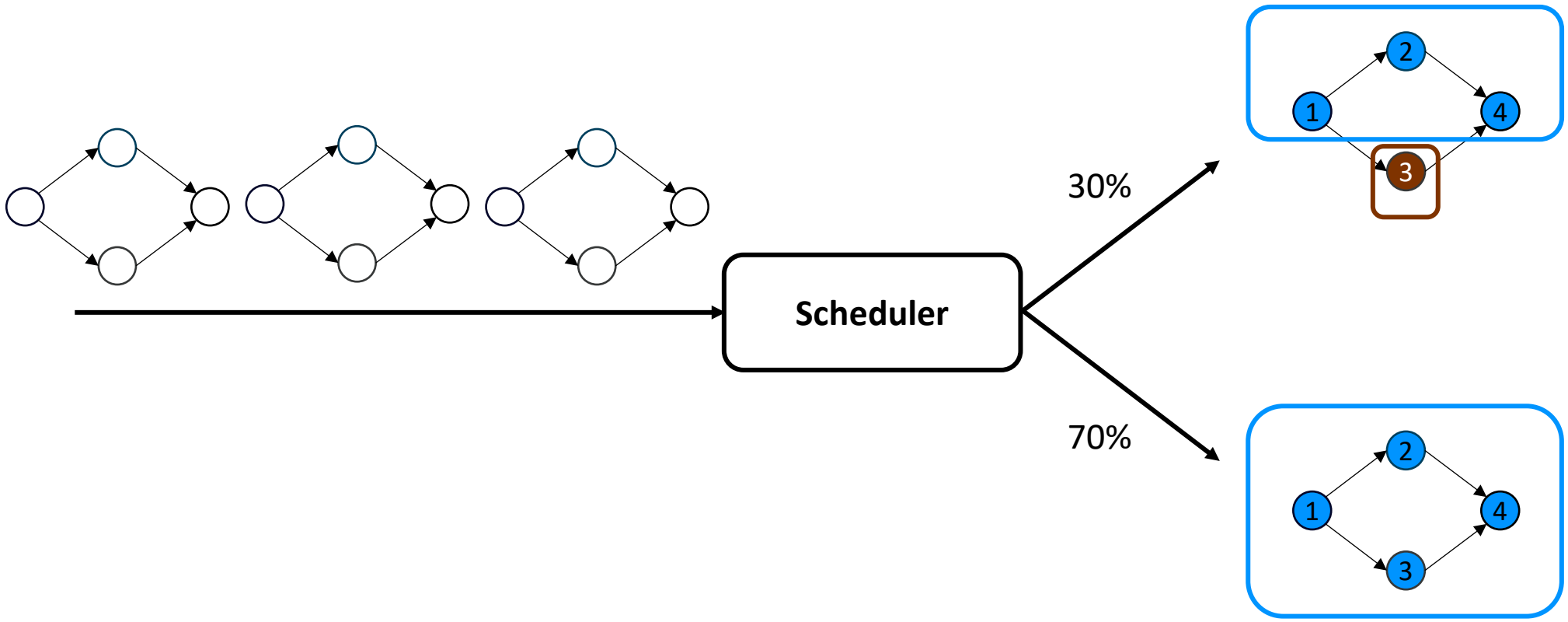
Queueing delay should be taken into account when making scheduling decisions

- Datacenter is shared with other applications
- When scheduling flows of jobs, other jobs in the flow contribute to queueing delay



Scheduling matters - Example

- The actual optimum is somewhere between different allocations



Existing schedulers - Network Congestion

Learning/History based

- Gandiva[OSDI'18]: learns isolated network costs
- Optimus[EuroSys'18]: assumes no congestion
- Decima[SIGCOMM'19]: does not include link sharing status in learning inputs
- Tiresias[NSDI'19]: assumes no congestion
- Themis[NSDI'20]: assumes no congestion
- Pollux[OSDI'21]: assumes no congestion

Model based

- Sparrow[SOSP'13]: assumes no congestion
- Tetris[SIGCOMM'14]: assumes no congestion
- FlexFlow[SysML'18]: assumes no congestion

Existing schedulers - Network Congestion

Learning/History based

- Gandiva[OSDI'18]: learns isolated network costs
- Optimus[EuroSys'18]: assumes no congestion
- Decima[SIGCOMM'19]: does not include link sharing status in learning inputs
- Tiresias[NSDI'19]: assumes no congestion
- Themis[NSDI'20]: assumes no congestion
- Pollux[OSDI'21]: assumes no congestion

Model based

- Sparrow[SOSP'13]: assumes no congestion
- Tetris[SIGCOMM'14]: assumes no congestion
- FlexFlow[SysML'18]: assumes no congestion

Taking the network into account is hard, predicting network queueing is even more challenging

One approach: Queueing models

Advantages:

- Pollaczek–Khinchine formula to predict queueing delays at different parts in the network

Challenges:

- Strong arrival process assumptions
- Incompatible with intricate heuristics

One approach: Queueing models

Advantages:

- Pollaczek–Khinchine formula to predict queueing delays at different parts in the network

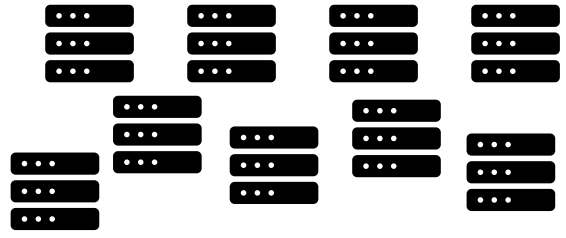
Challenges:

- Strong arrival process assumptions
- Incompatible with intricate heuristics

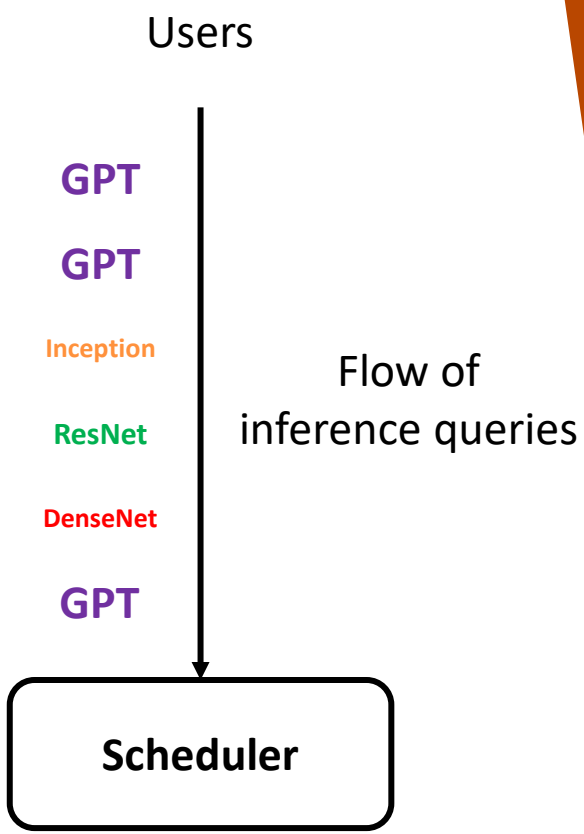
Solution: Stochastic scheduling!

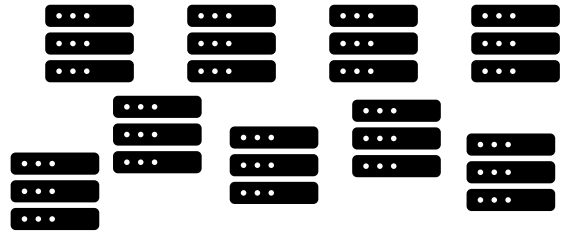
Users

Nona’s System Model

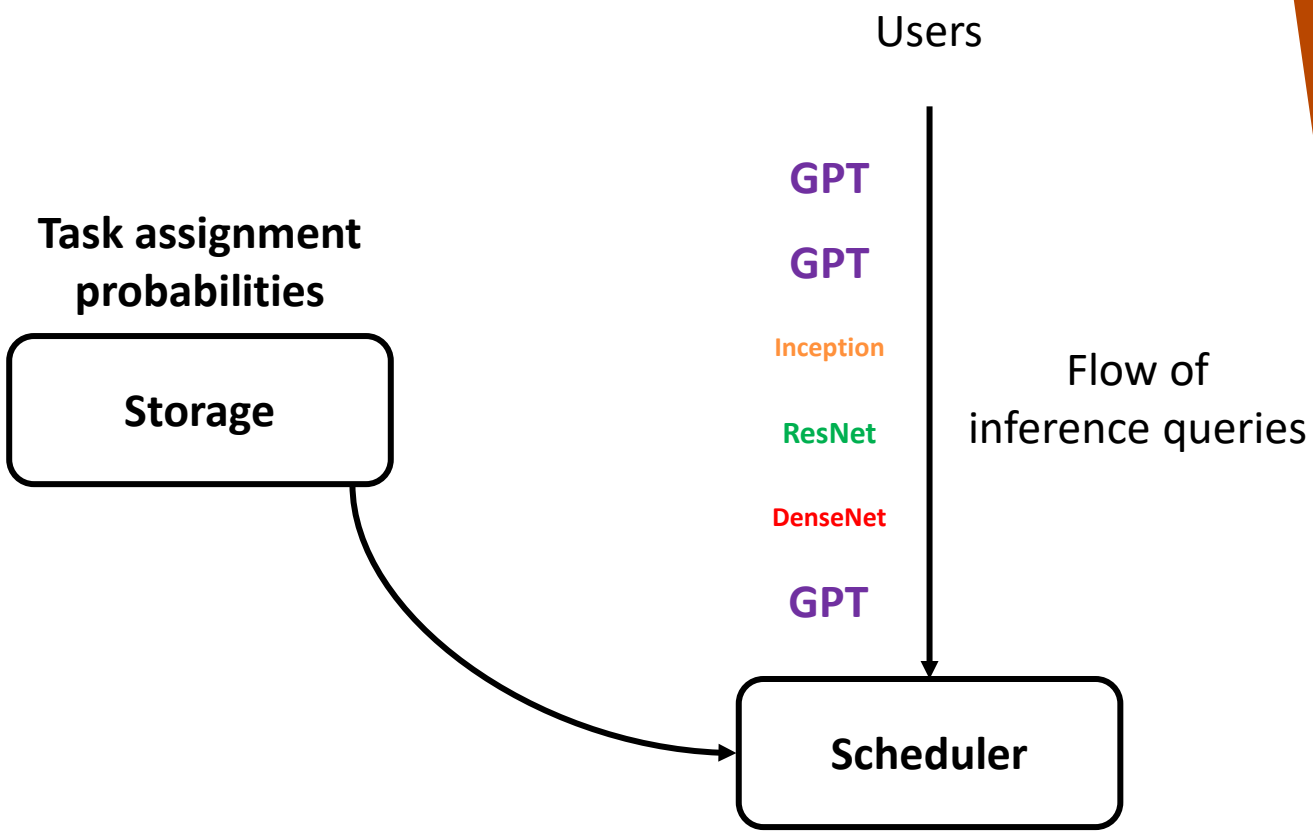


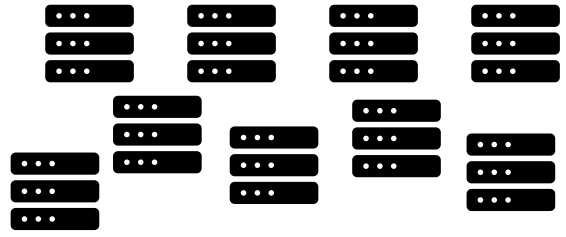
Nona's System Model





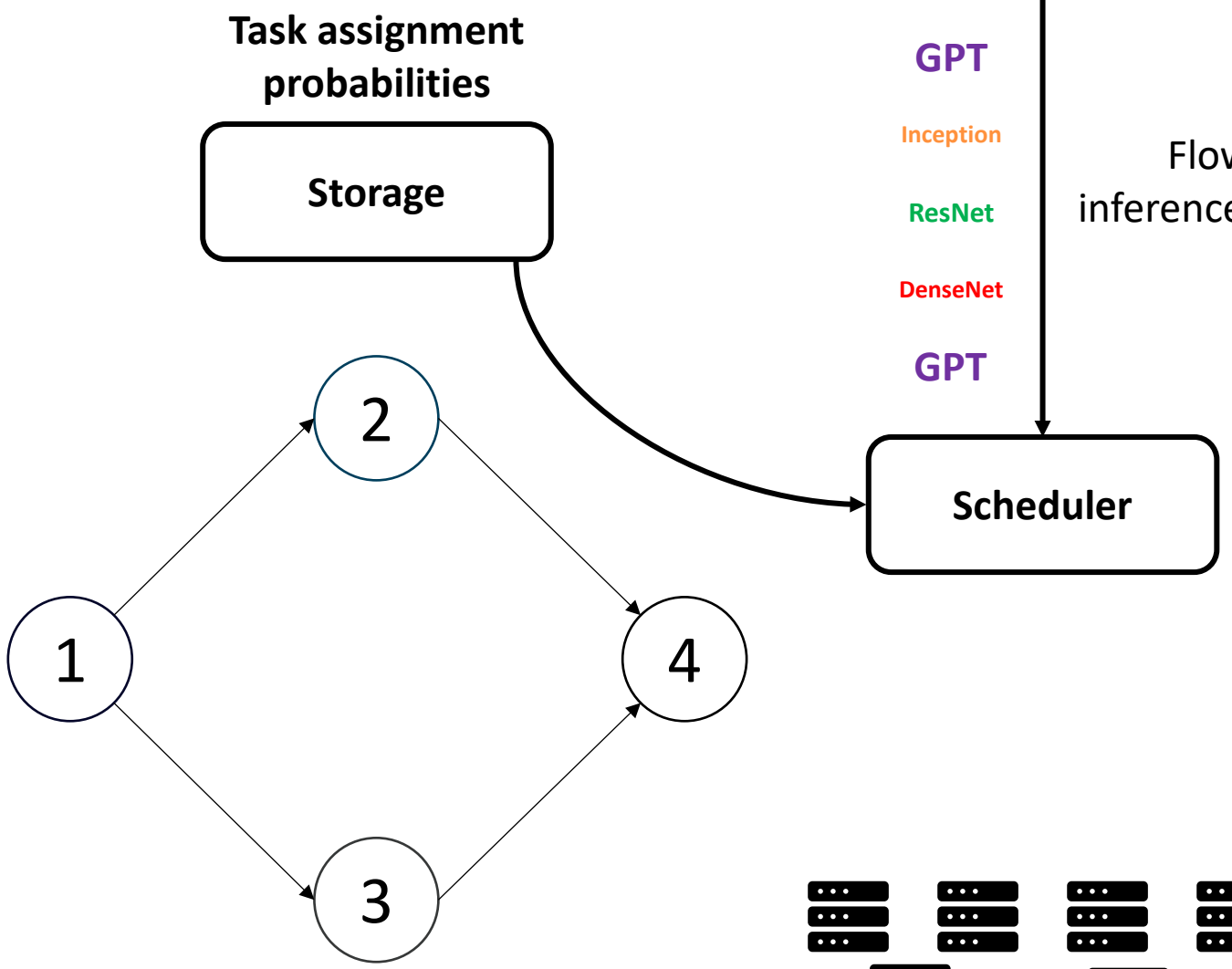
Nona's System Model



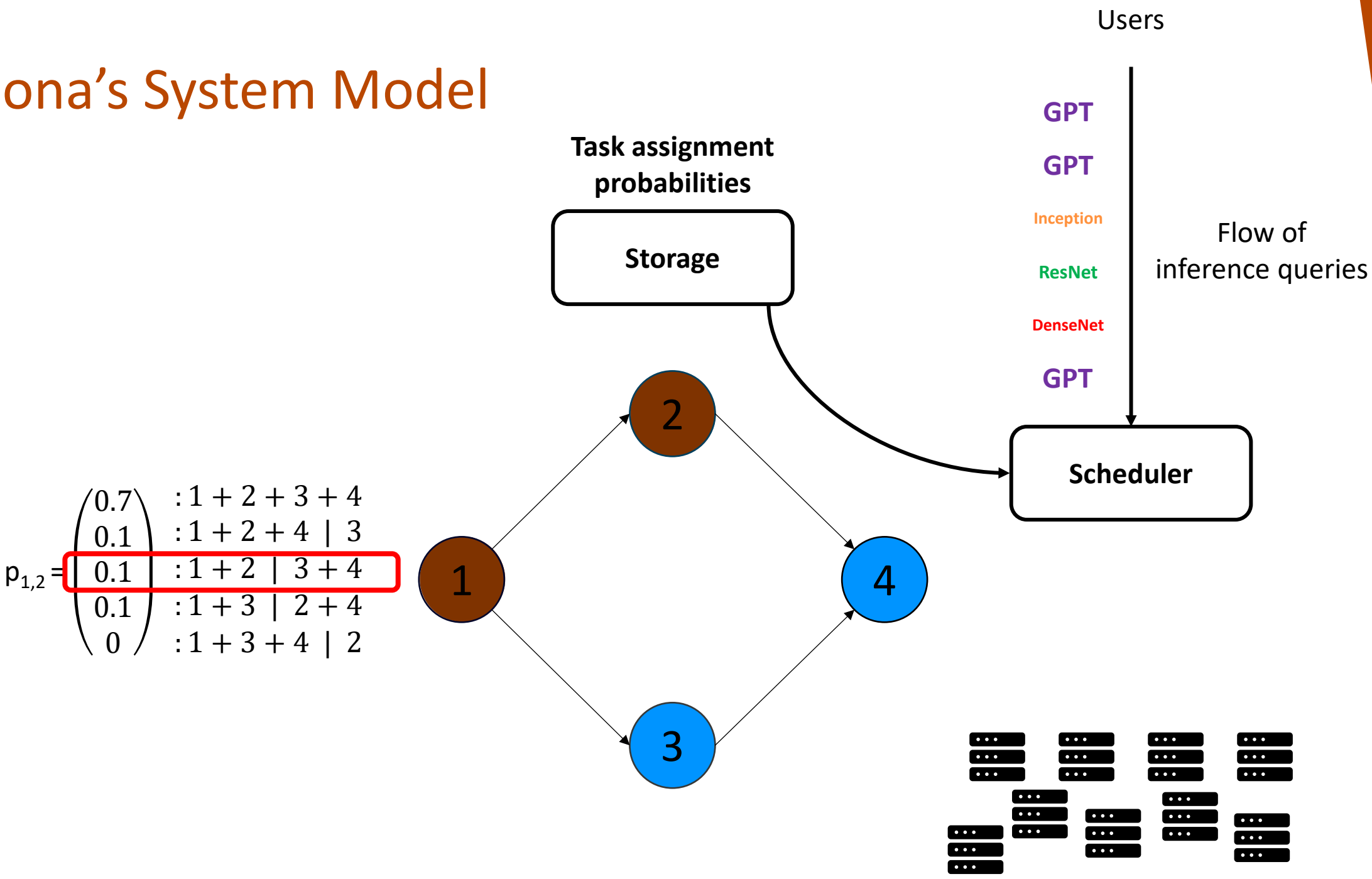


Nona's System Model

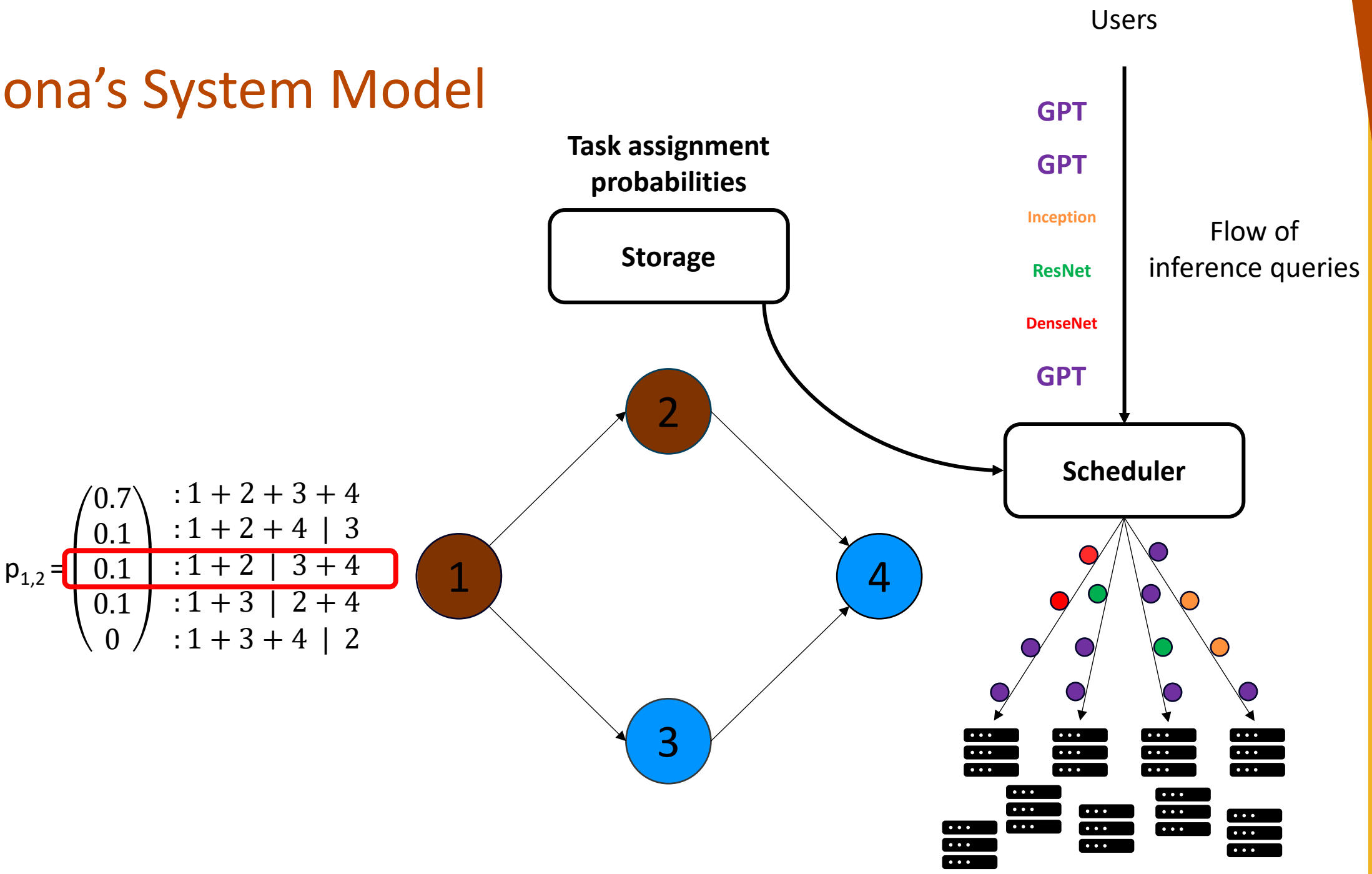
$$p_{1,2} = \begin{pmatrix} 0.7 \\ 0.1 \\ 0.1 \\ 0.1 \\ 0 \end{pmatrix} \begin{matrix} : 1+2+3+4 \\ : 1+2+4 \mid 3 \\ : 1+2 \mid 3+4 \\ : 1+3 \mid 2+4 \\ : 1+3+4 \mid 2 \end{matrix}$$



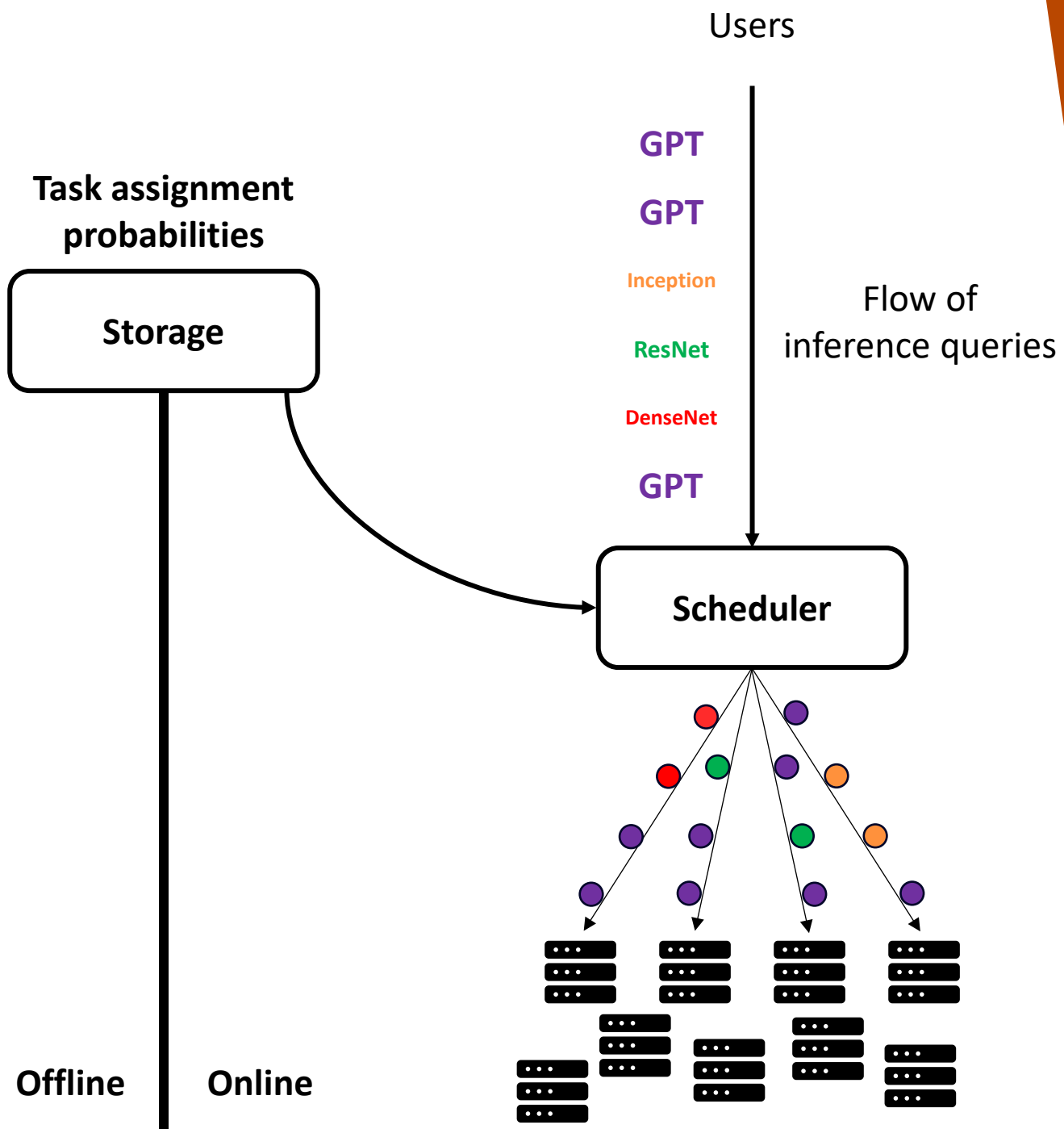
Nona's System Model



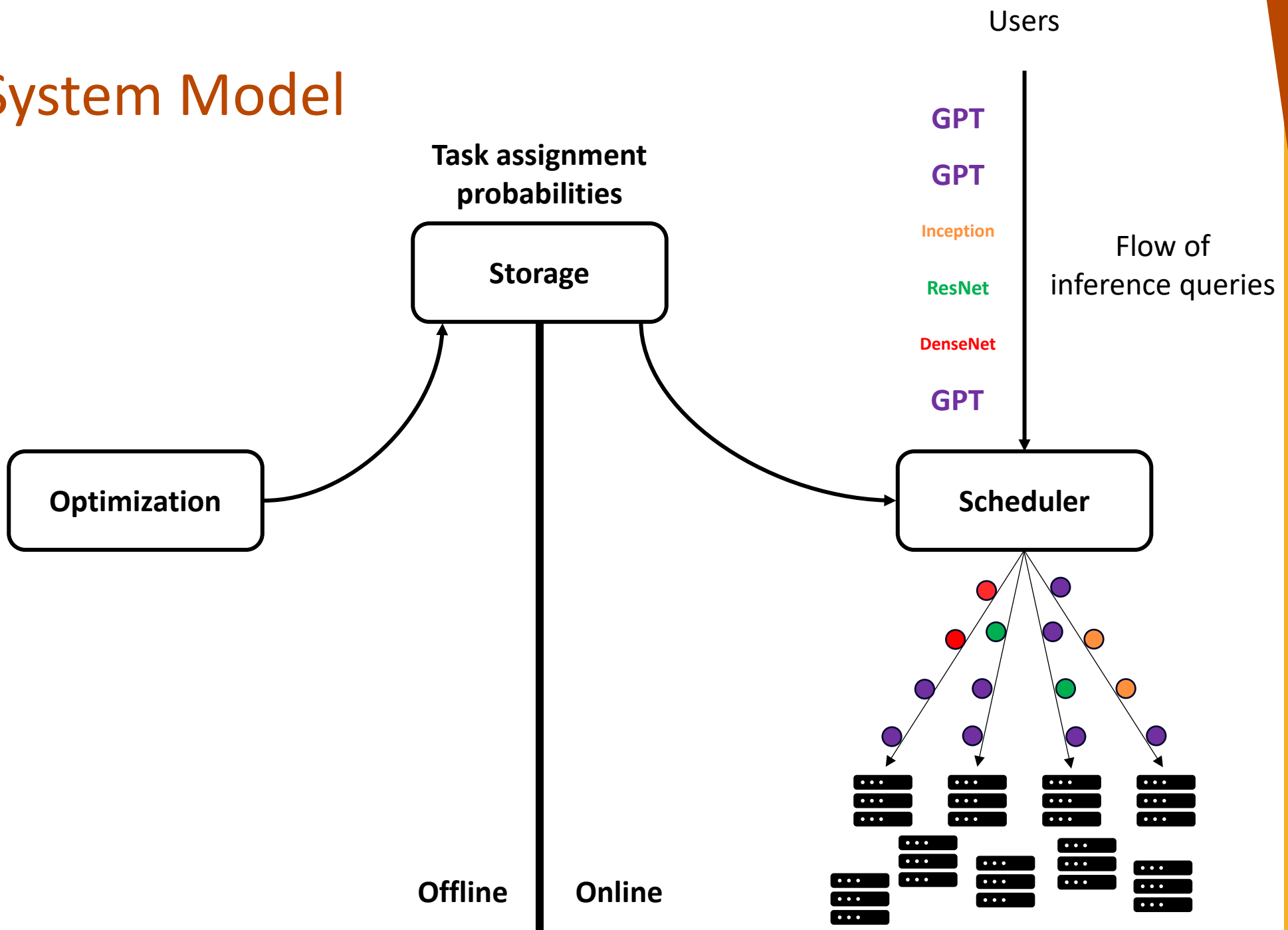
Nona's System Model



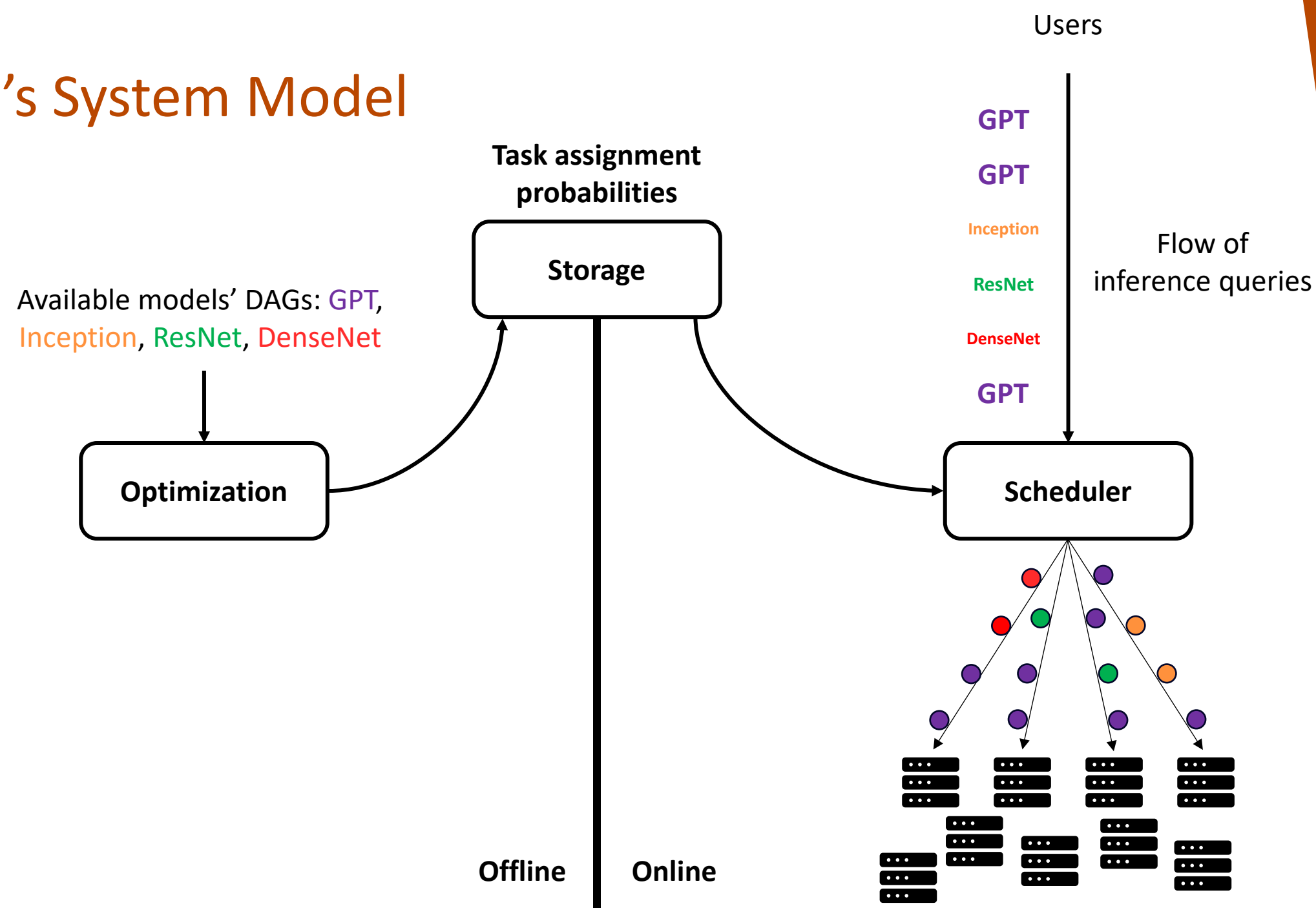
Nona's System Model



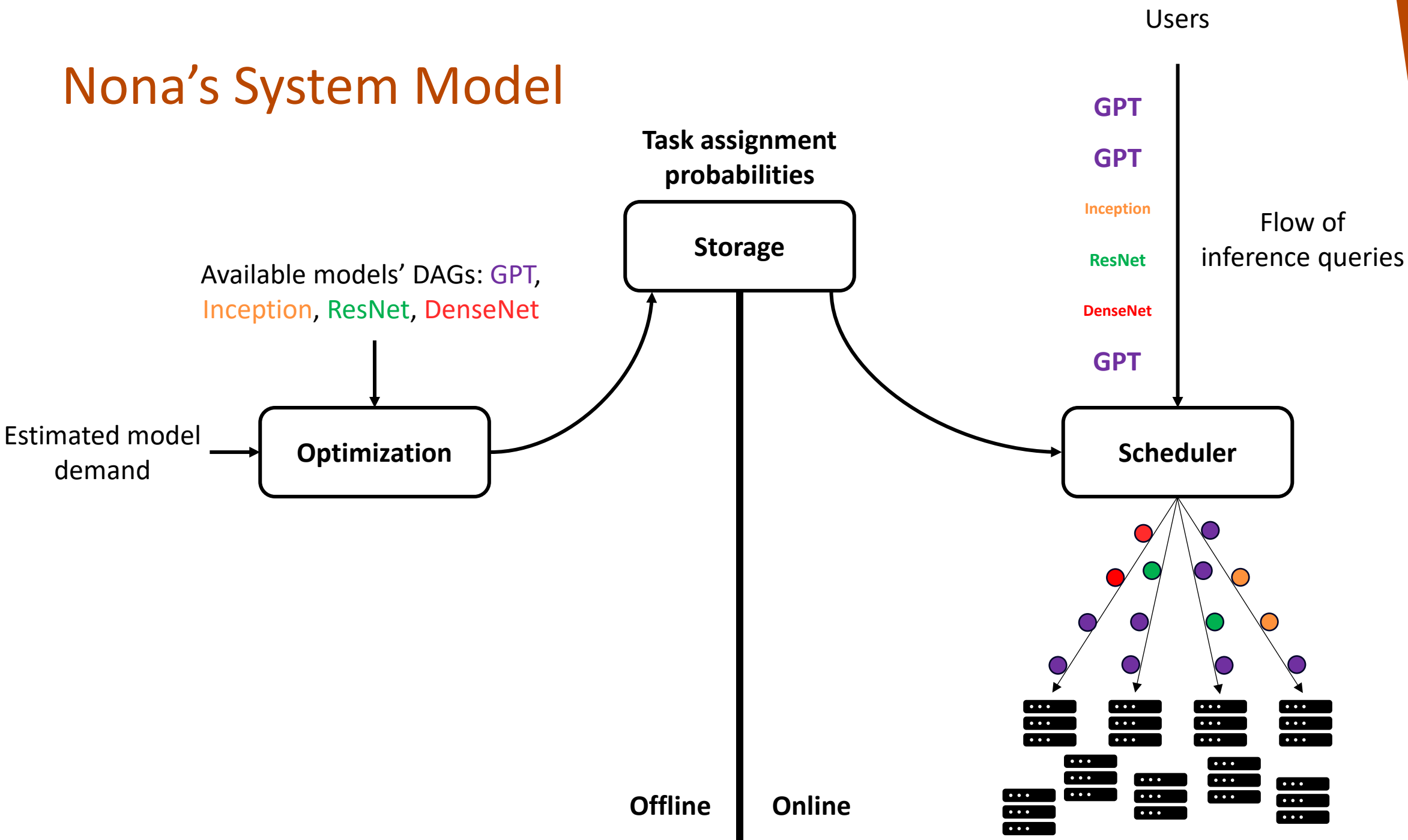
Nona's System Model



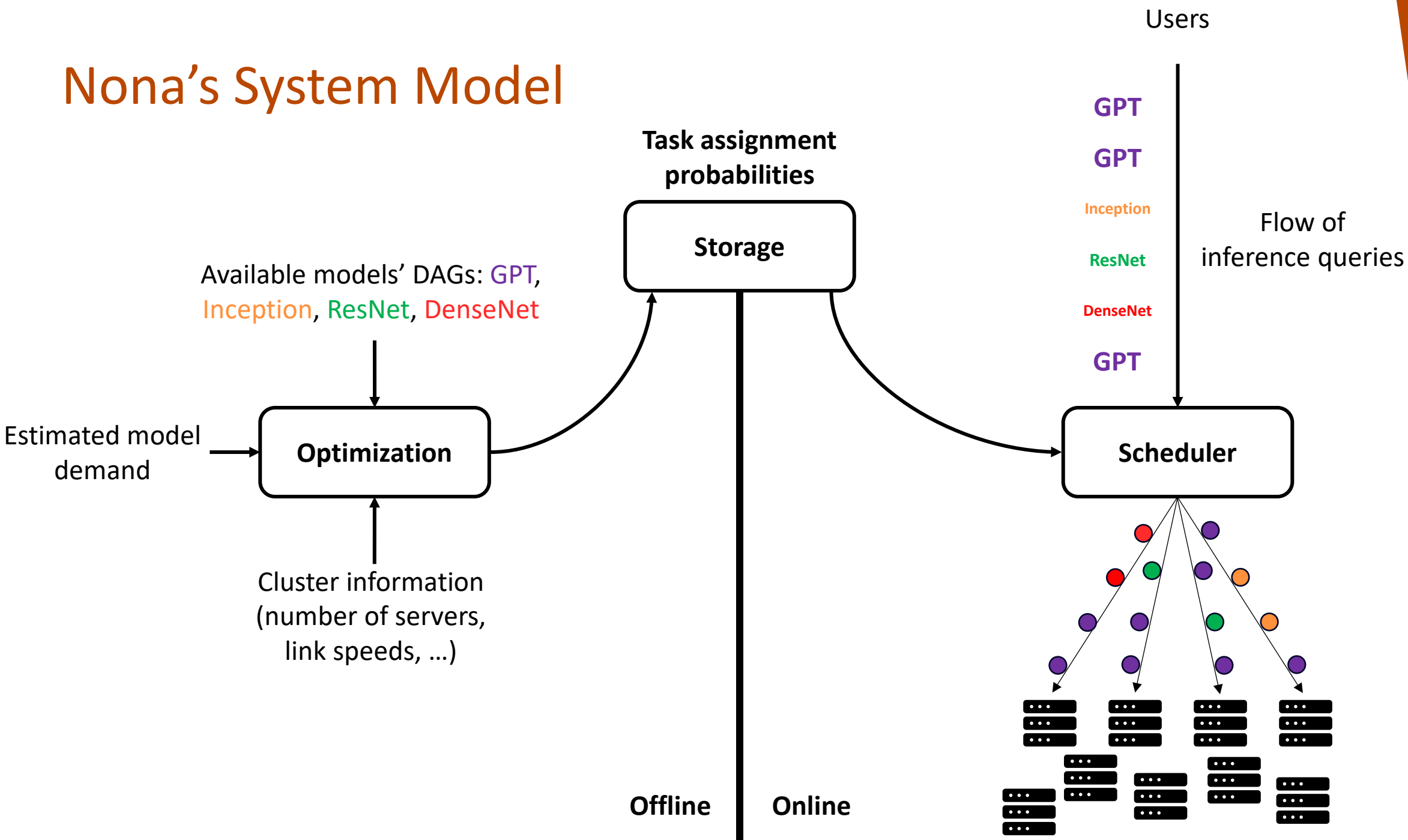
Nona's System Model



Nona's System Model



Nona's System Model



Nona’s Optimization formulation

- We want to determine the optimal assignment probability distributions for each task in each job:

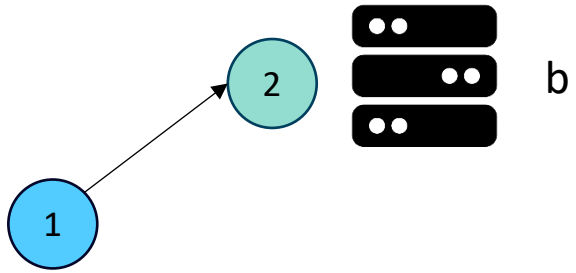
Minimize:	Job completion time
Subject to:	communications, computing, flow, and scheduling constraints, auxiliary variables

Nona's Optimization formulation

- We want to determine the optimal assignment probability distributions for each task in each job:

  Minimize: Job completion time
  Subject to: communications, computing, flow, and scheduling constraints, auxiliary variables

In practice, task 2 completes on server b when:



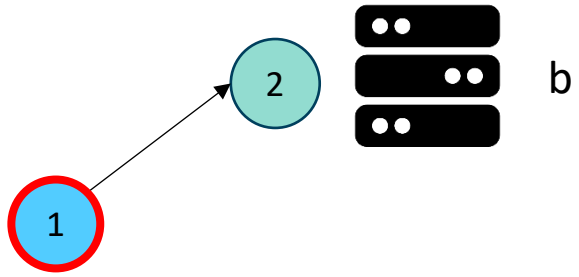
Nona's Optimization formulation

- We want to determine the optimal assignment probability distributions for each task in each job:

  Minimize: Job completion time
  Subject to: communications, computing, flow, and scheduling constraints, auxiliary variables

In practice, task 2 completes on server b when:

+Task 1 completed, then



Nona's Optimization formulation

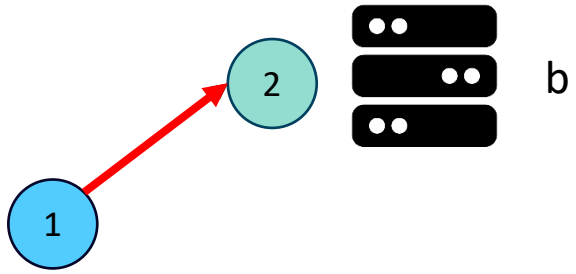
- We want to determine the optimal assignment probability distributions for each task in each job:

  Minimize: Job completion time
  Subject to: communications, computing, flow, and scheduling constraints, auxiliary variables

In practice, task 2 completes on server b when:

+Task 1 completed, then

+Data was transferred if needed, then



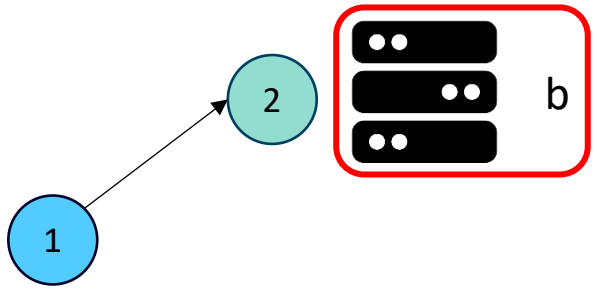
Nona's Optimization formulation

- We want to determine the optimal assignment probability distributions for each task in each job:

  Minimize: Job completion time
  Subject to: communications, computing, flow, and scheduling constraints, auxiliary variables

In practice, task 2 completes on server b when:

+Task 1 completed, then
+Data was transferred if needed, then
+Server b finished all previous jobs, then



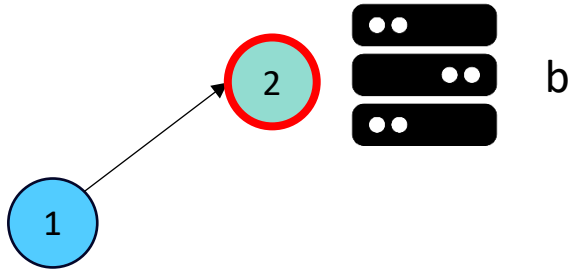
Nona's Optimization formulation

- We want to determine the optimal assignment probability distributions for each task in each job:

  Minimize: Job completion time
  Subject to: communications, computing, flow, and scheduling constraints, auxiliary variables

In practice, task 2 completes on server b when:

+Task 1 completed, then
+Data was transferred if needed, then
+Server b finished all previous jobs, then
+Task 2 finished running



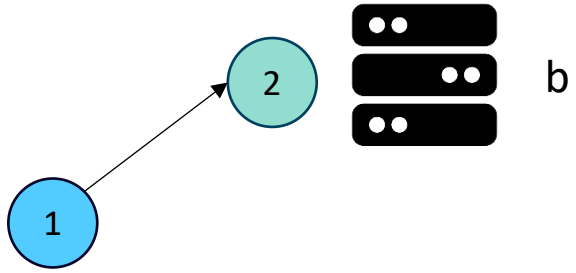
Nona's Optimization formulation

- We want to determine the optimal assignment probability distributions for each task in each job:

  Minimize: Job completion time
  Subject to: communications, computing, flow, and scheduling constraints, auxiliary variables

In practice, task 2 completes on server b when:

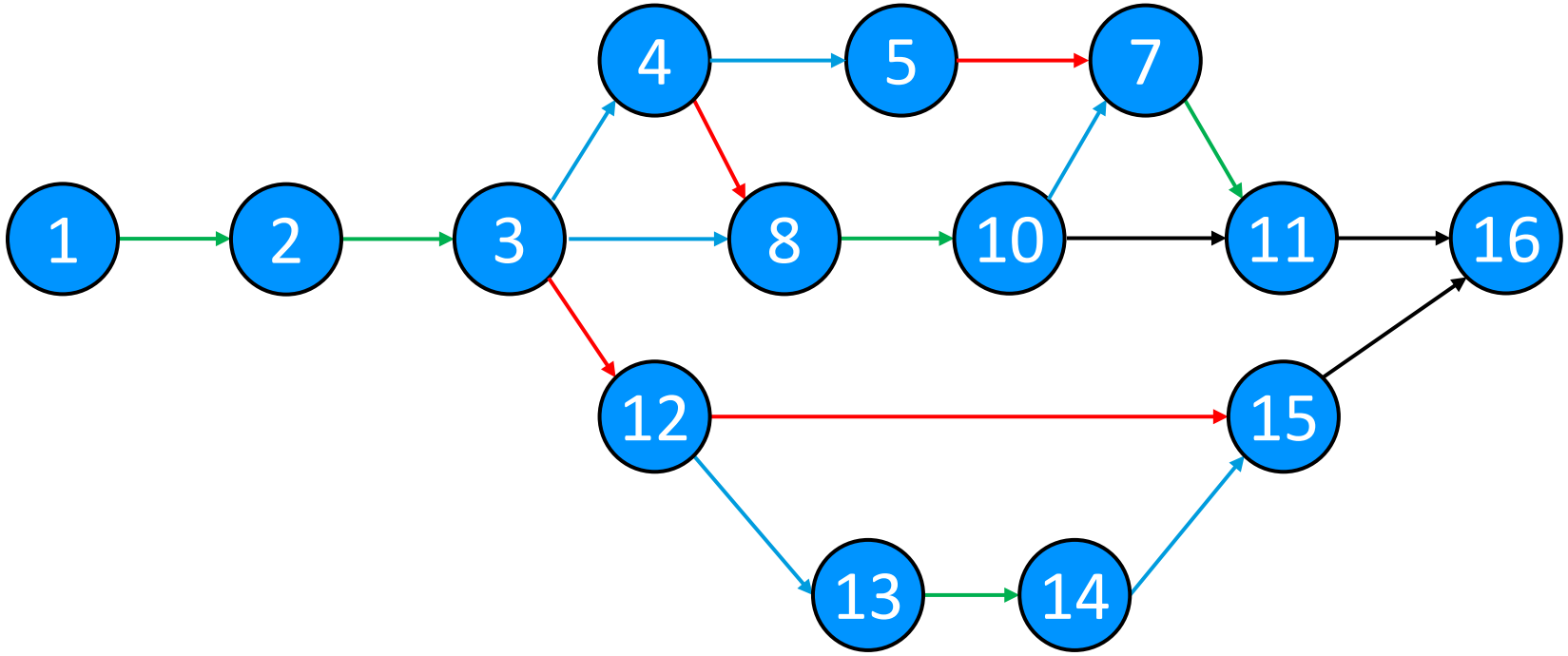
+Task 1 completed, then
+Data was transferred if needed, then
+Server b finished all previous jobs, then
+Task 2 finished running



See the paper or nona.csail.mit.edu for more details

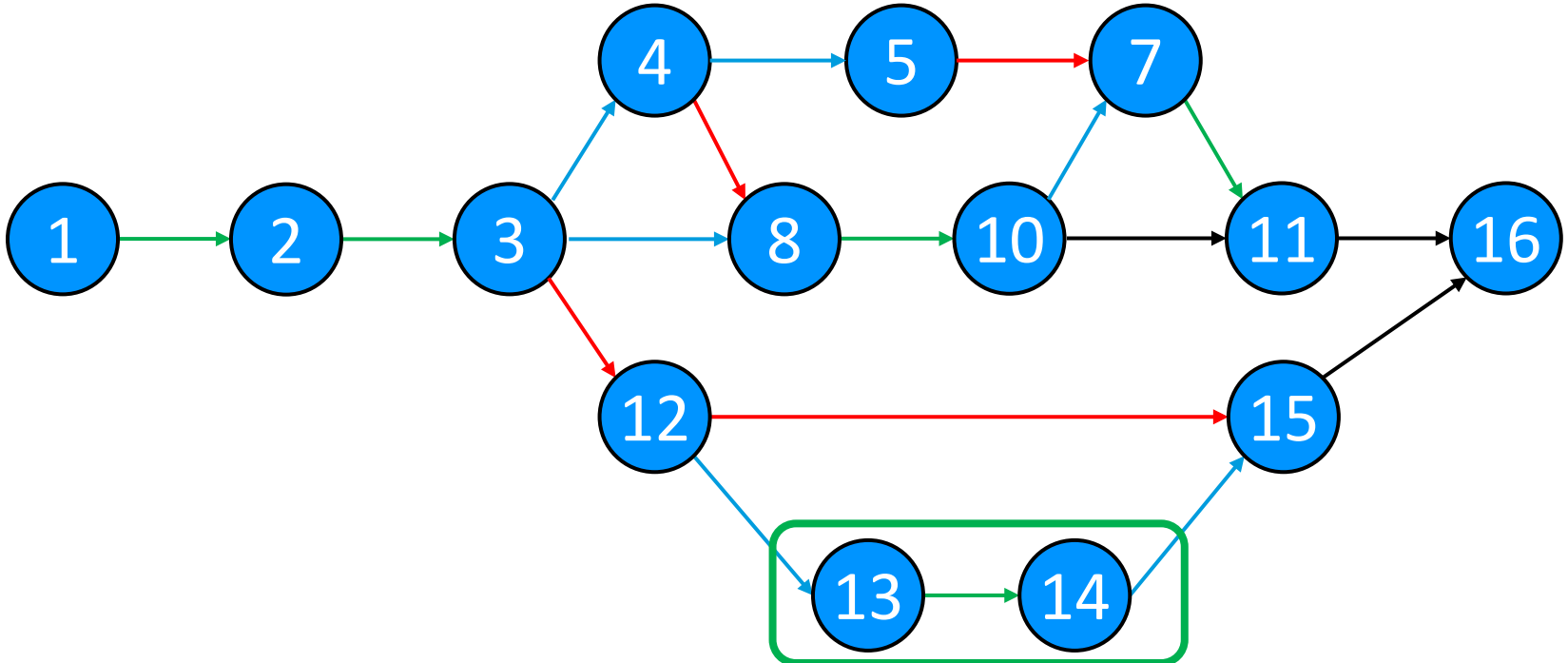
Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted, contract all edges a → b where
  - All input nodes of b are also input nodes of a, and
  - All output nodes of a are also output nodes of b.



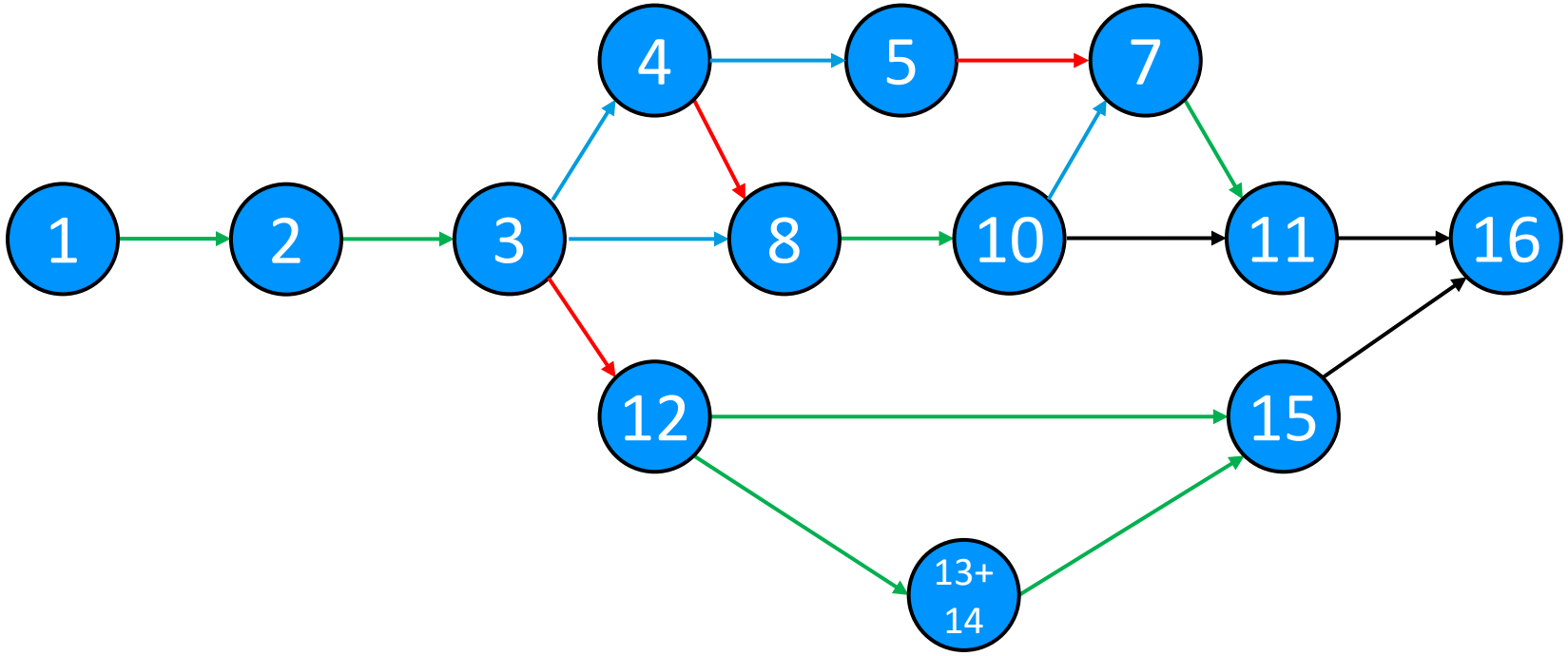
Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted, contract all edges a → b where
  - All input nodes of b are also input nodes of a, and
  - All output nodes of a are also output nodes of b.



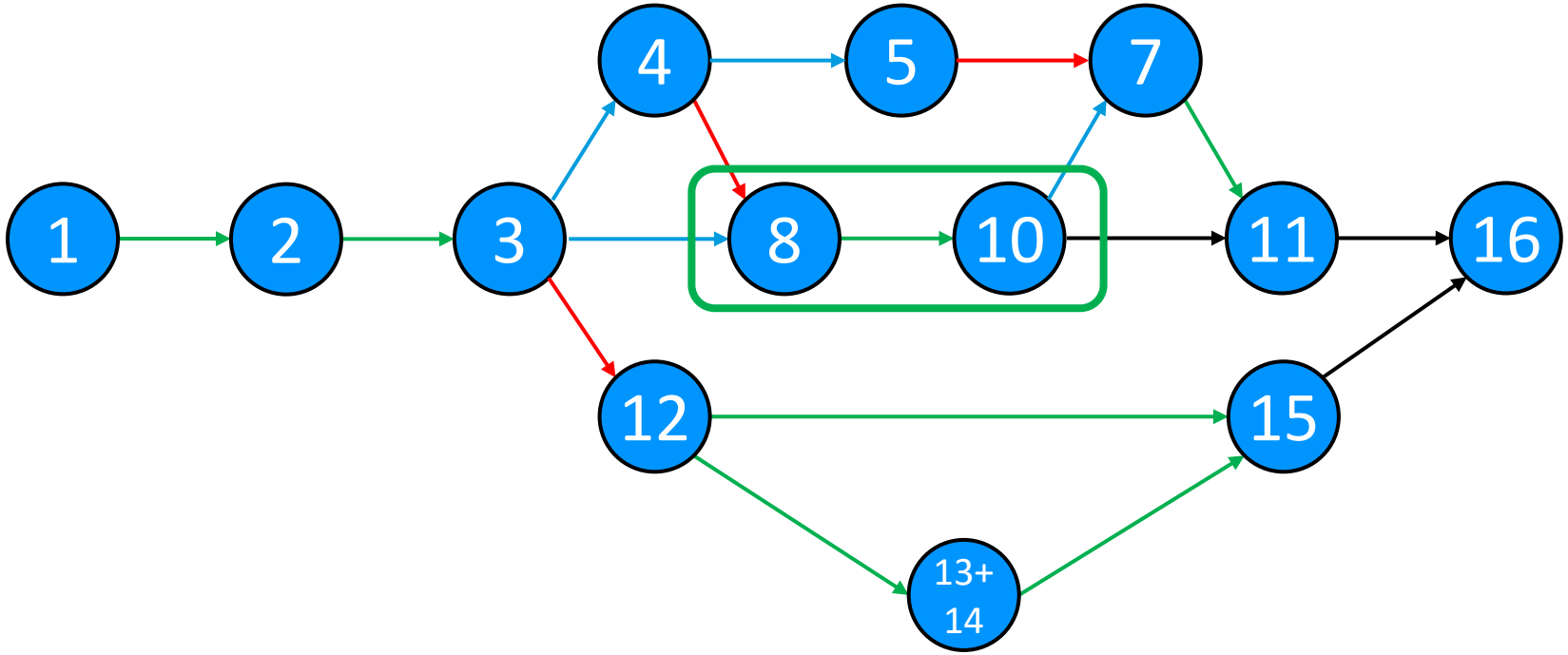
Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted, contract all edges a → b where
  - All input nodes of b are also input nodes of a, and
  - All output nodes of a are also output nodes of b.



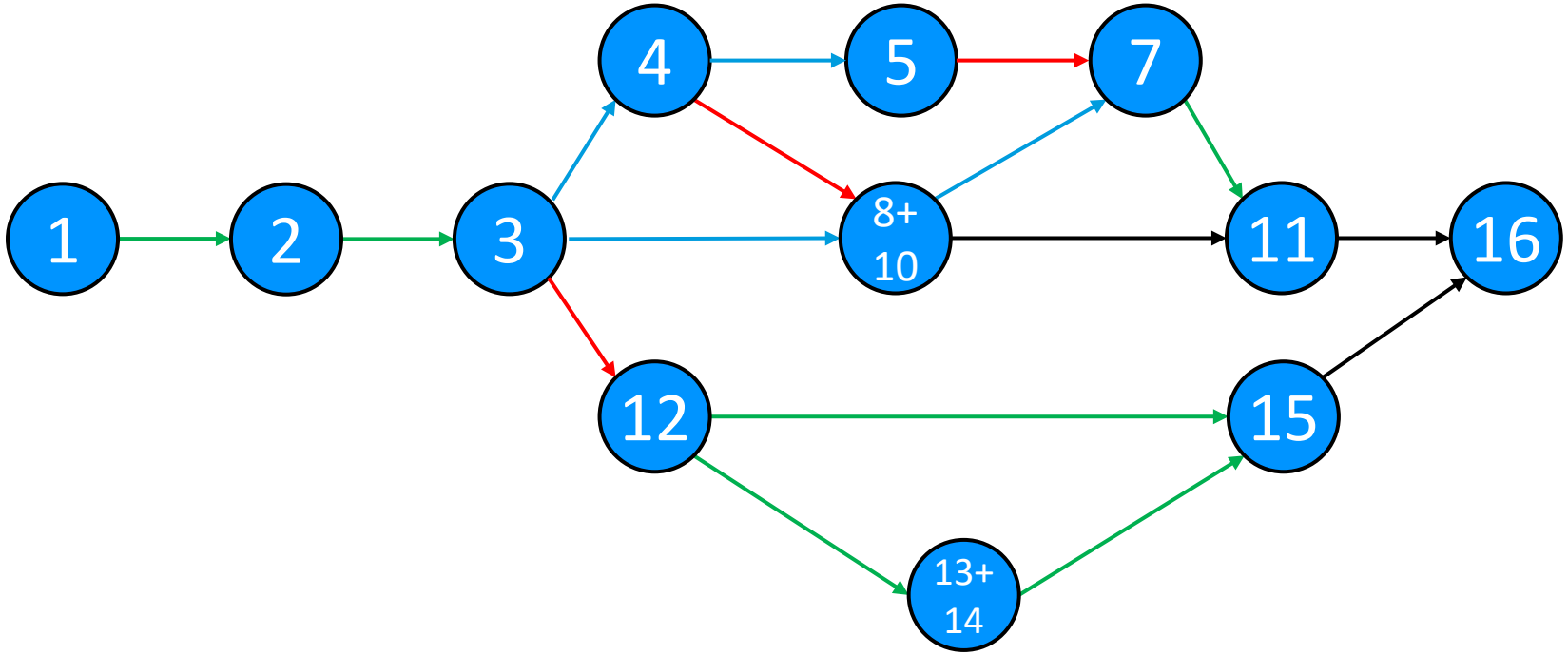
Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted, contract all edges a → b where
  - All input nodes of b are also input nodes of a, and
  - All output nodes of a are also output nodes of b.



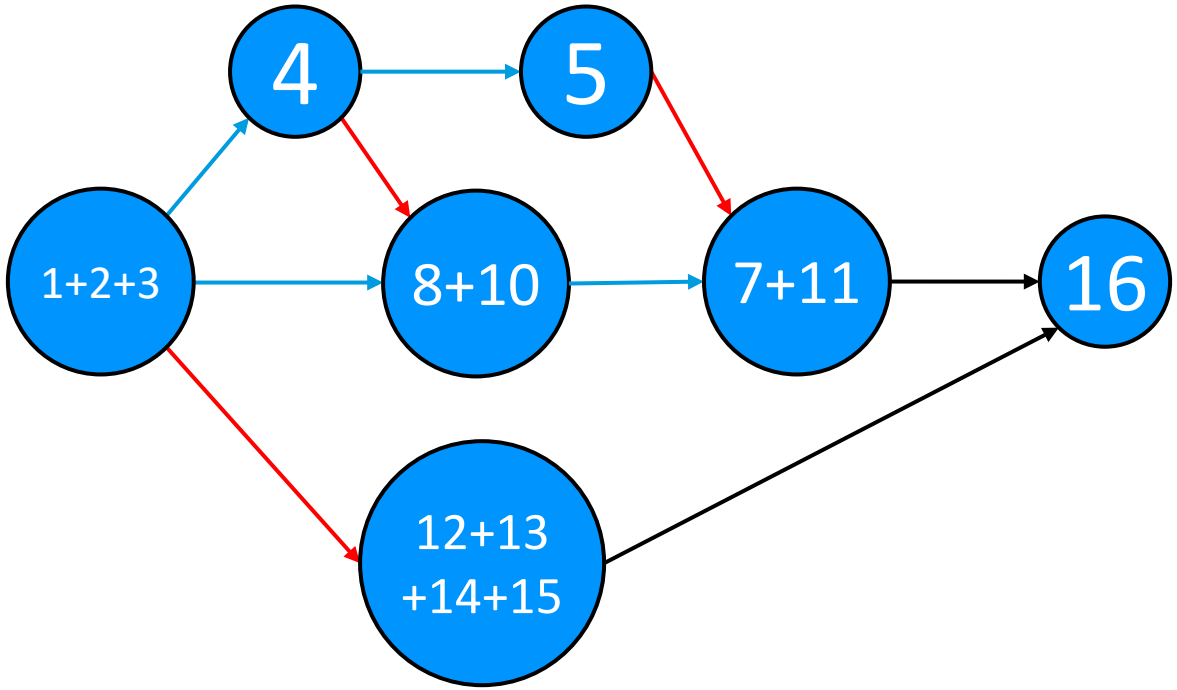
Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted, contract all edges a → b where
  - All input nodes of b are also input nodes of a, and
  - All output nodes of a are also output nodes of b.



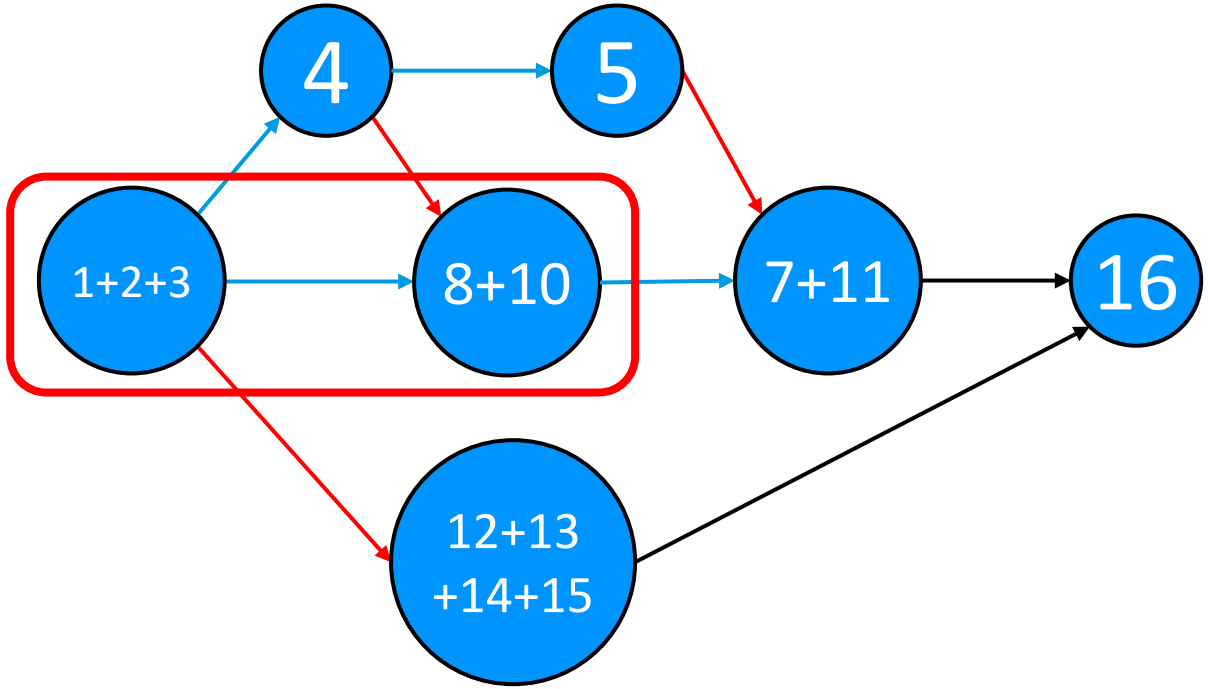
Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted, contract all edges a → b where
  - All input nodes of b are also input nodes of a, and
  - All output nodes of a are also output nodes of b.



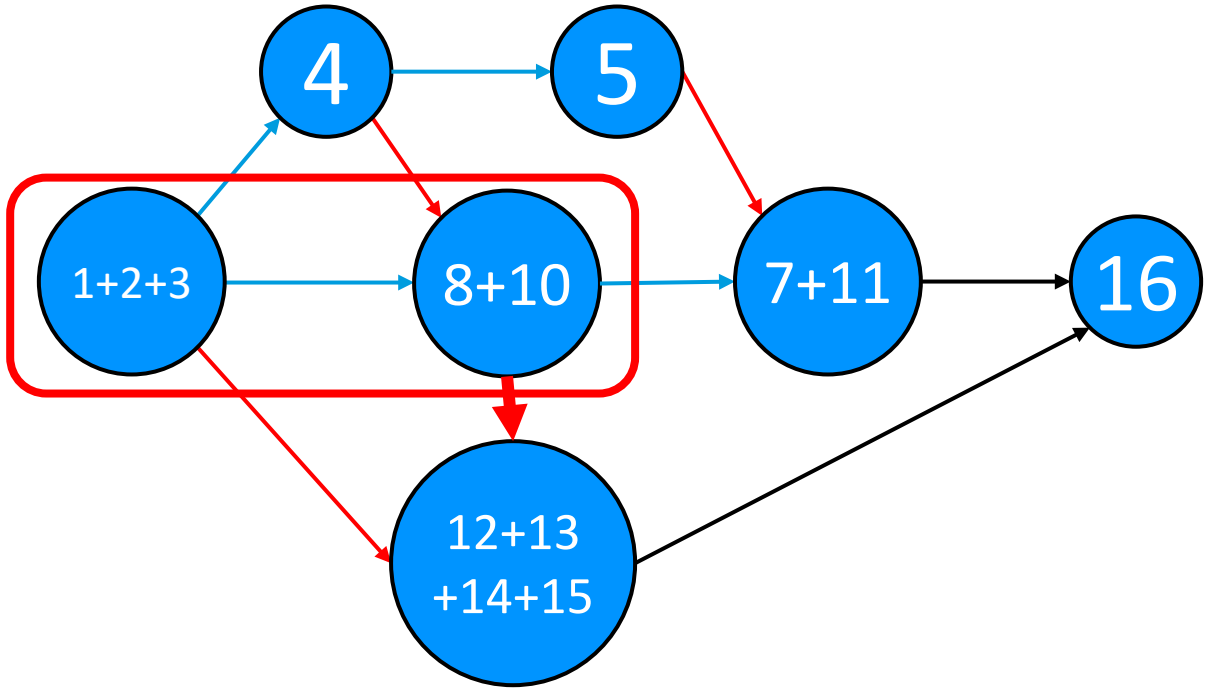
Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted, contract all edges a → b where
  - All input nodes of b are also input nodes of a, and
  - All output nodes of a are also output nodes of b.



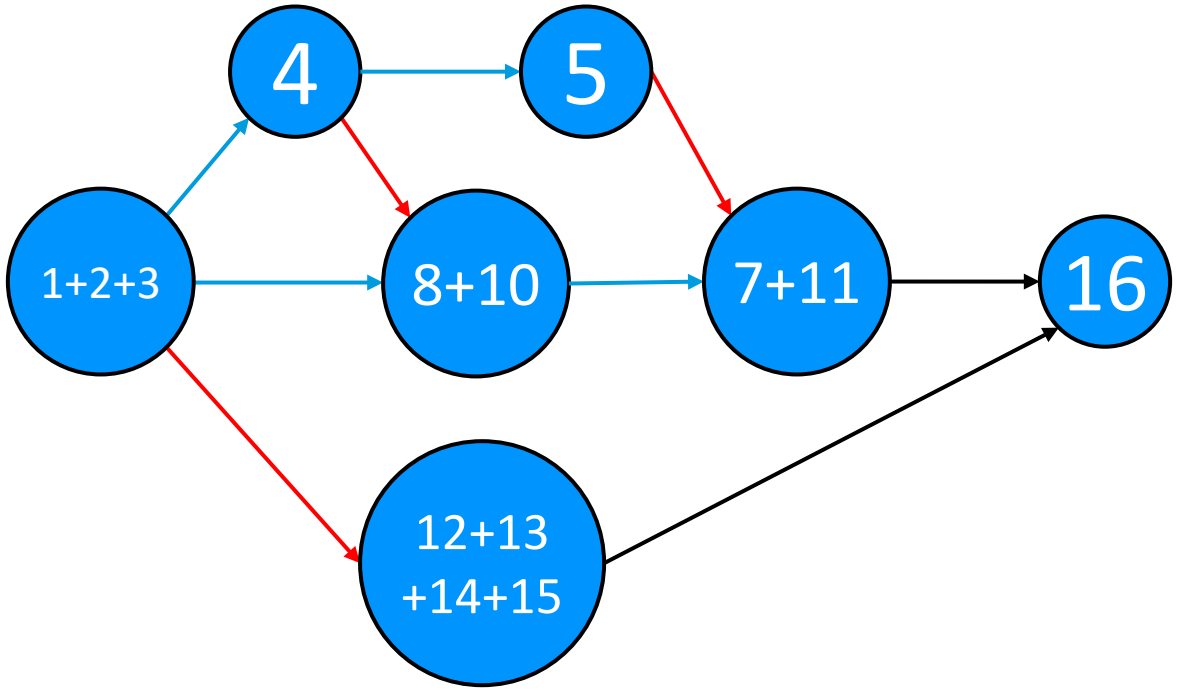
Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted, contract all edges a → b where
  - All input nodes of b are also input nodes of a, and
  - All output nodes of a are also output nodes of b.



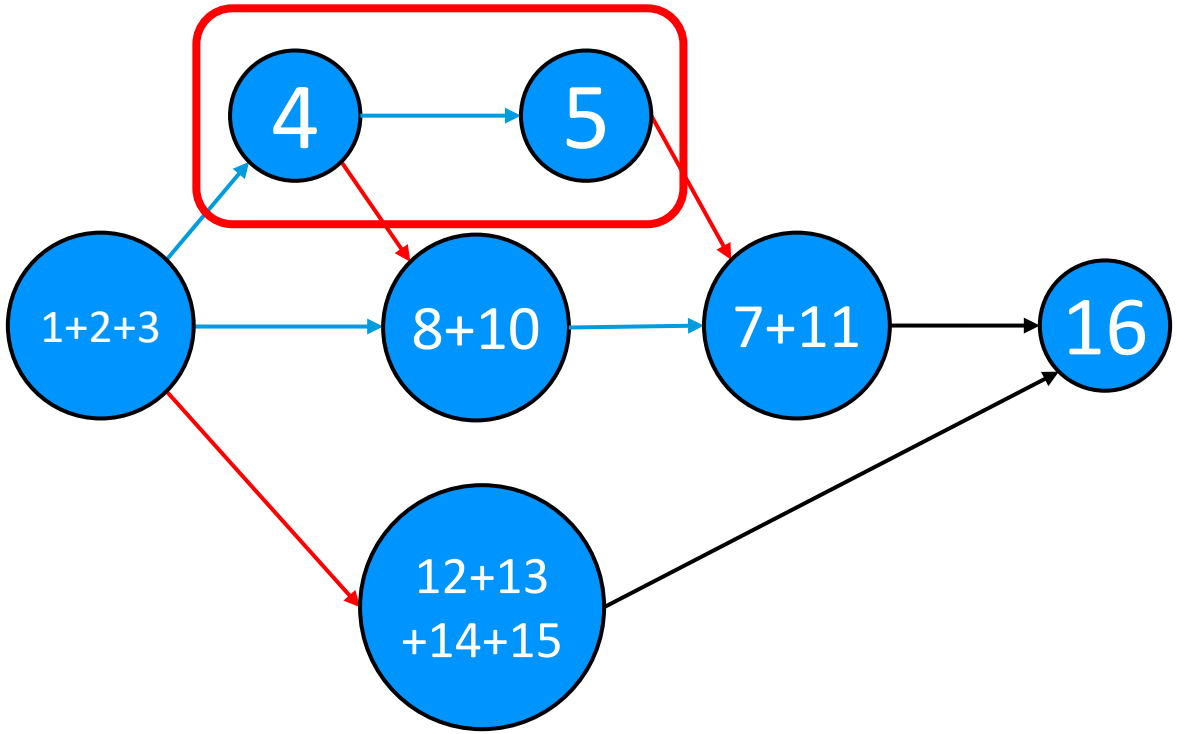
Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted, contract all edges a → b where
  - All input nodes of b are also input nodes of a, and
  - All output nodes of a are also output nodes of b.



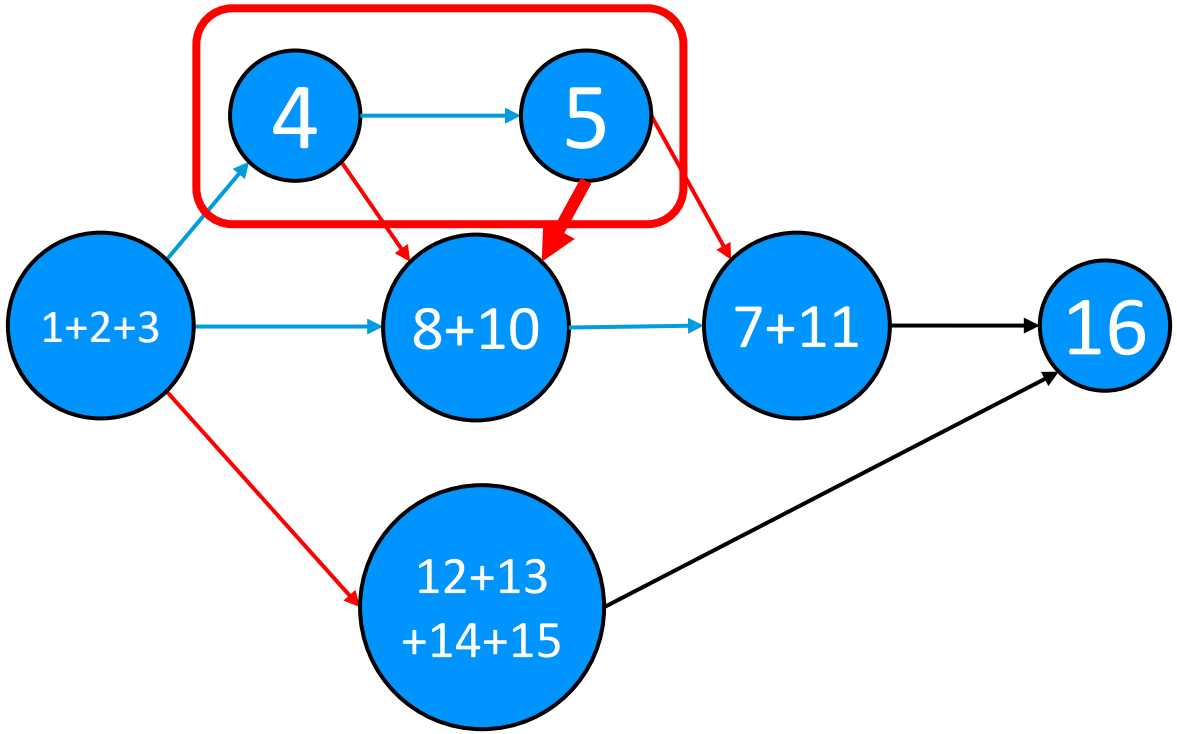
Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted, contract all edges a → b where
  - All input nodes of b are also input nodes of a, and
  - All output nodes of a are also output nodes of b.



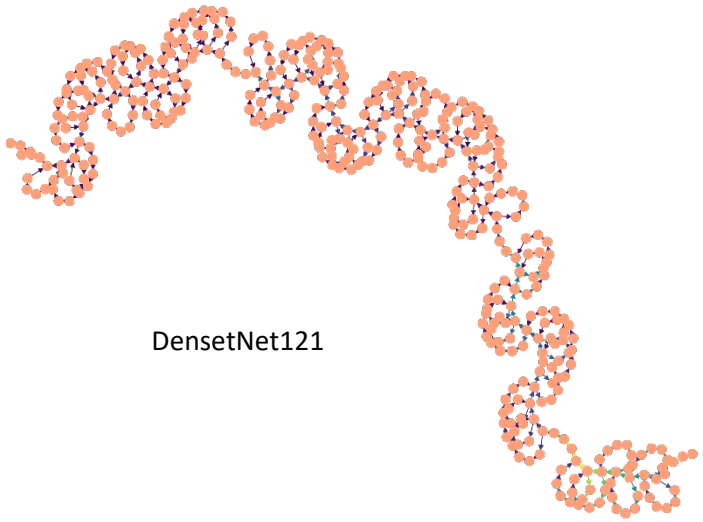
Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted, contract all edges a → b where
  - All input nodes of b are also input nodes of a, and
  - All output nodes of a are also output nodes of b.



Graph Contraction

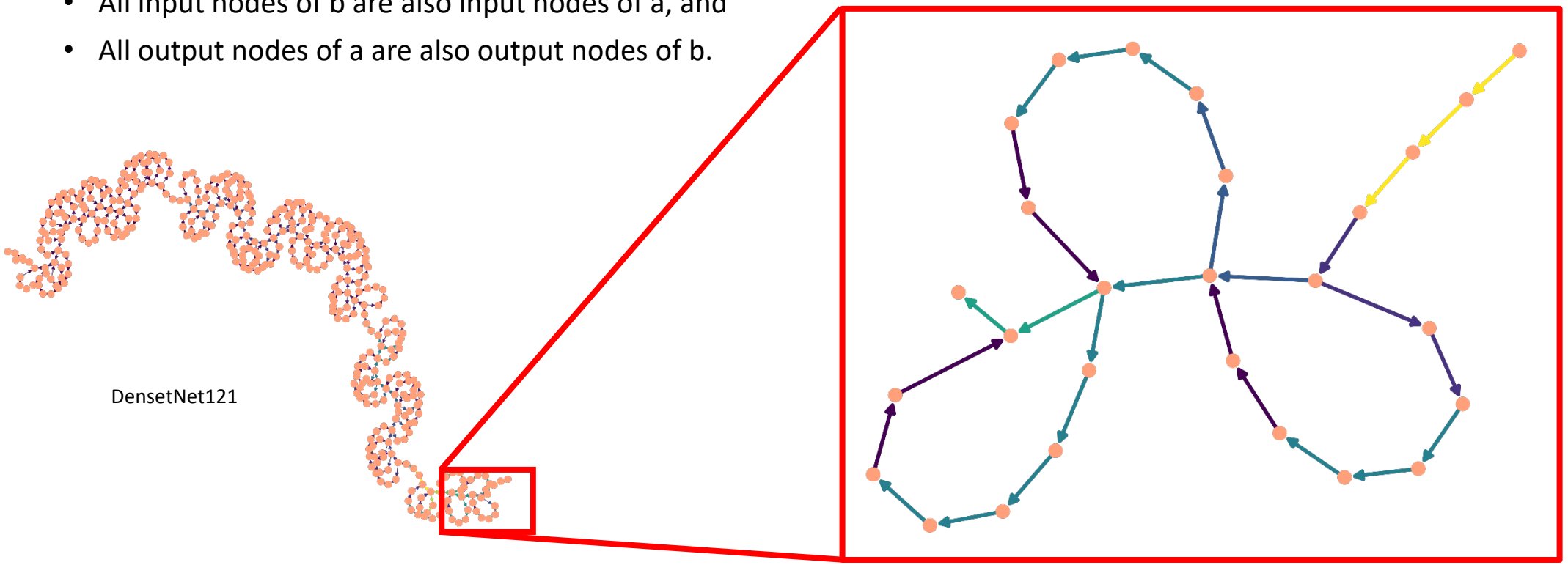
- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted:
  - Contract all edges a → b where
    - All input nodes of b are also input nodes of a, and
    - All output nodes of a are also output nodes of b.



DensetNet121

Graph Contraction

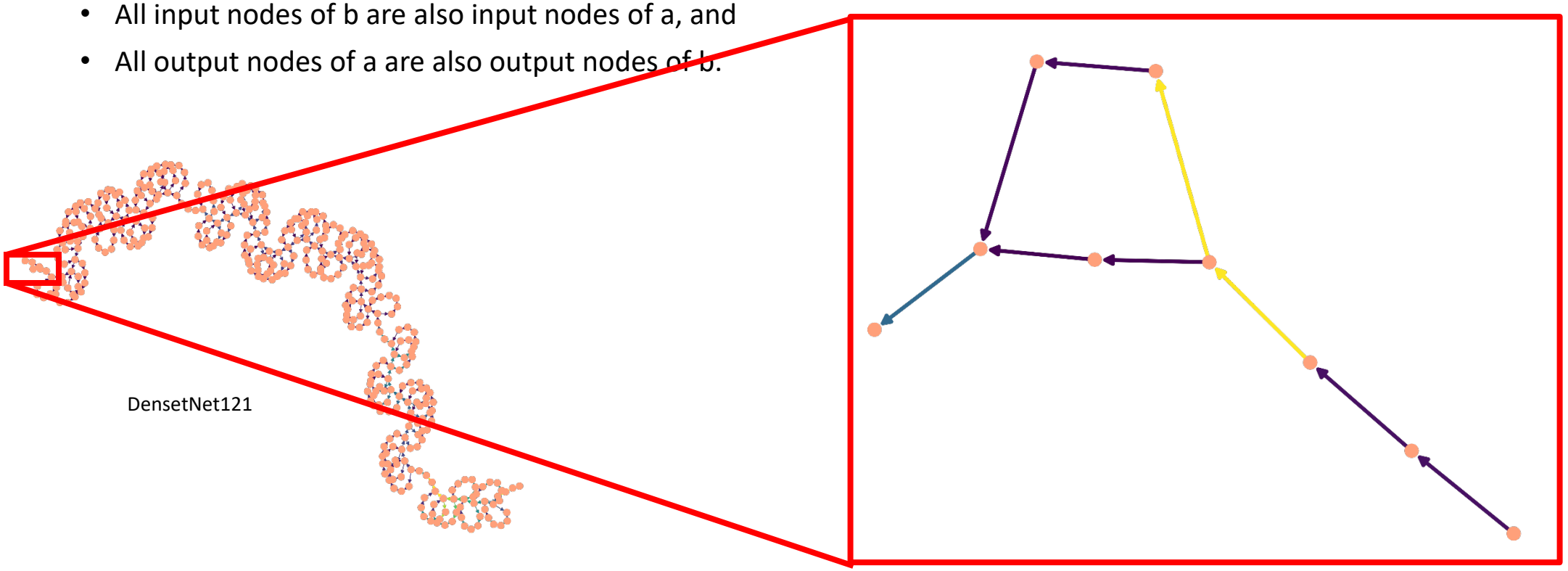
- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted:
  - Contract all edges a → b where
    - All input nodes of b are also input nodes of a, and
    - All output nodes of a are also output nodes of b.



DensetNet121

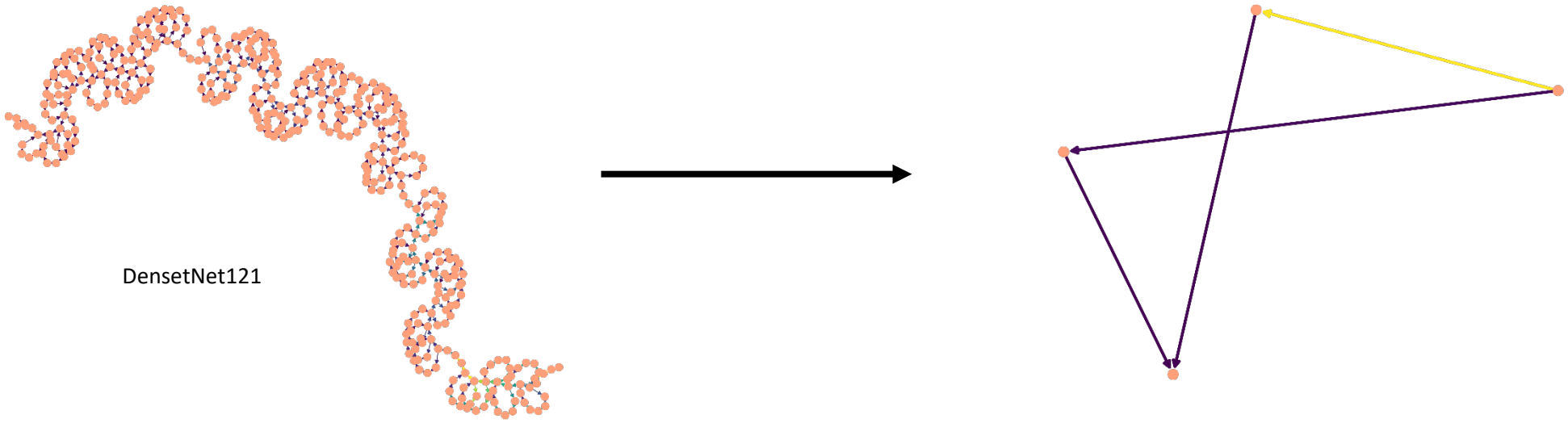
Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted:
  - Contract all edges a → b where
    - All input nodes of b are also input nodes of a, and
    - All output nodes of a are also output nodes of b.



Graph Contraction

- To reduce the number of tasks, we contract tasks that do not gain from being run in parallel.
- While the graph can be further contracted:
  - Contract all edges a → b where
    - All input nodes of b are also input nodes of a, and
    - All output nodes of a are also output nodes of b.

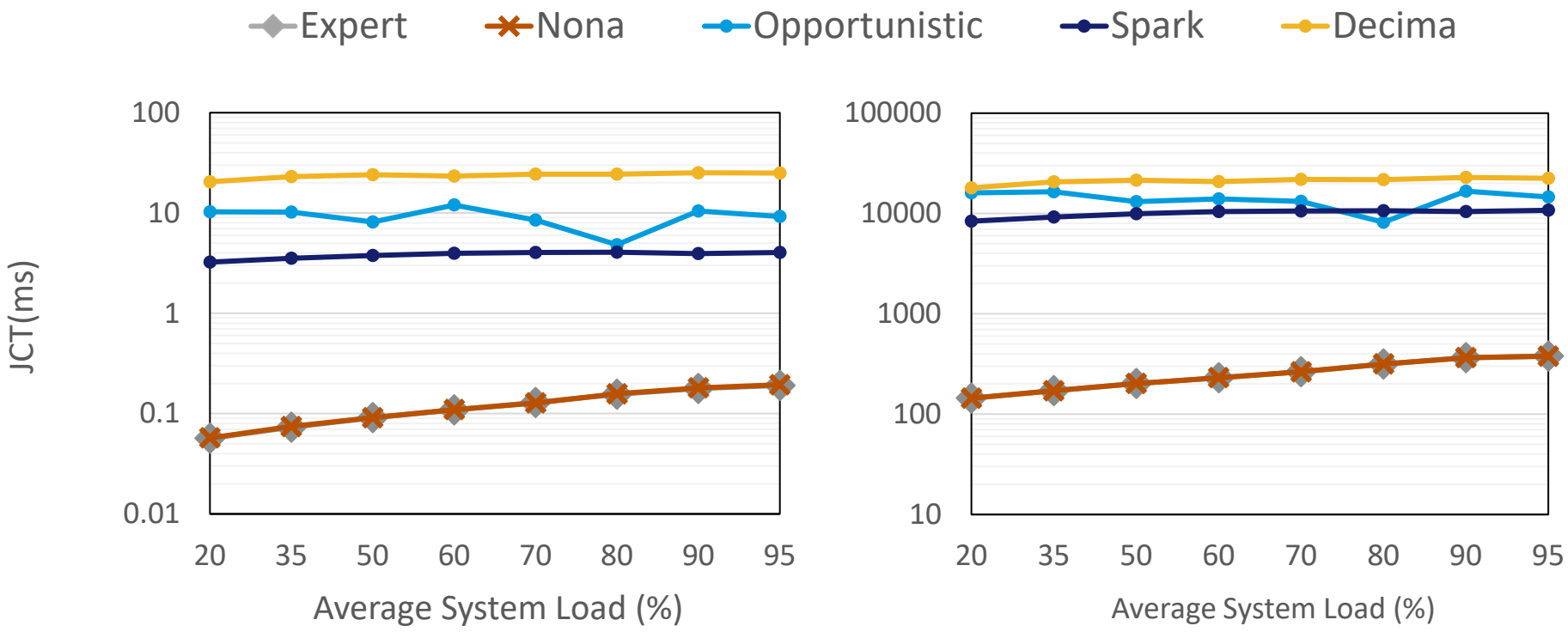


Evaluation

Setup

- Workload consists in contracted versions of AlexNet, ResNet18, VGG16, Densenet, and GPT2
  + Background traffic and compute tasks
- We use a simulator from Decima [SIGCOMM'19], a Reinforcement Learning (RL)-based scheduler, originally made for Spark
- We compare Nona with
  - Decima
  - Spark's Fair Scheduler
  - Opportunistic
  - Hand picked expert solutions
- 80-server cluster

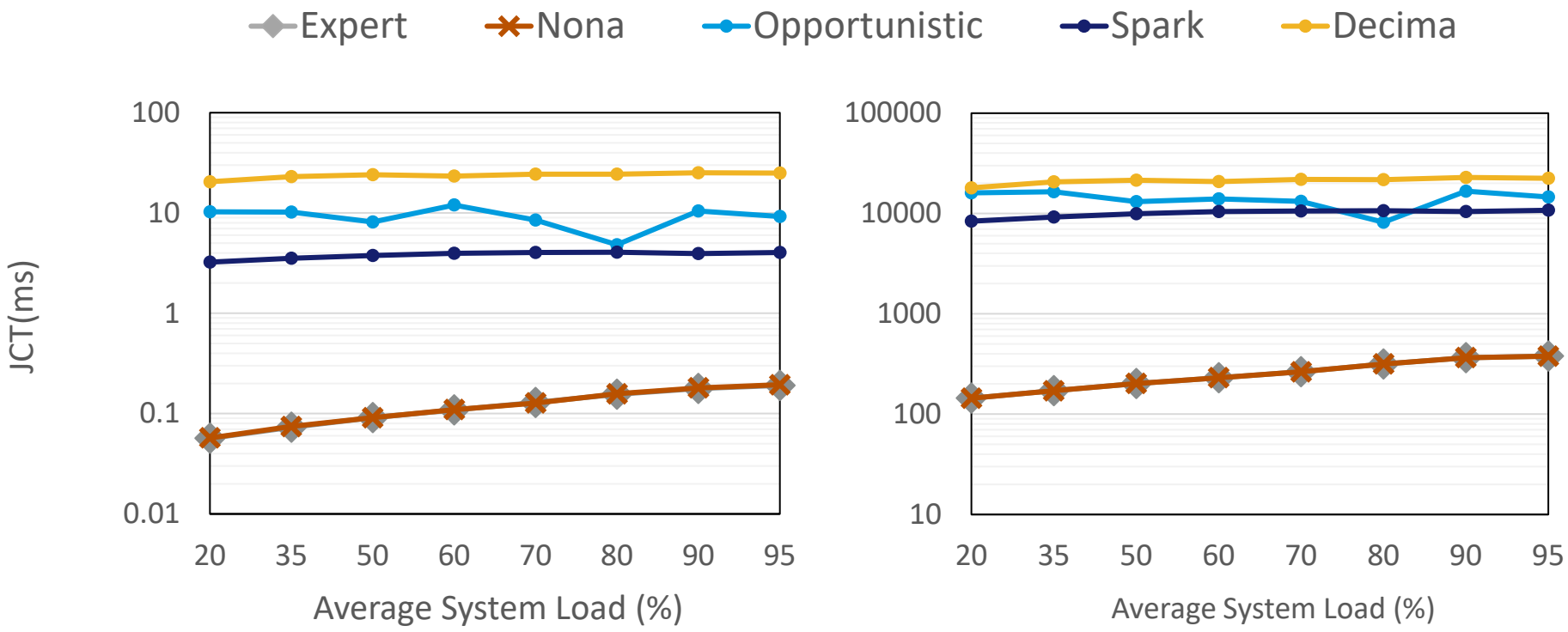
Impact of Job Arrival Rate



(a) Average over all jobs

(b) Average over foreground jobs

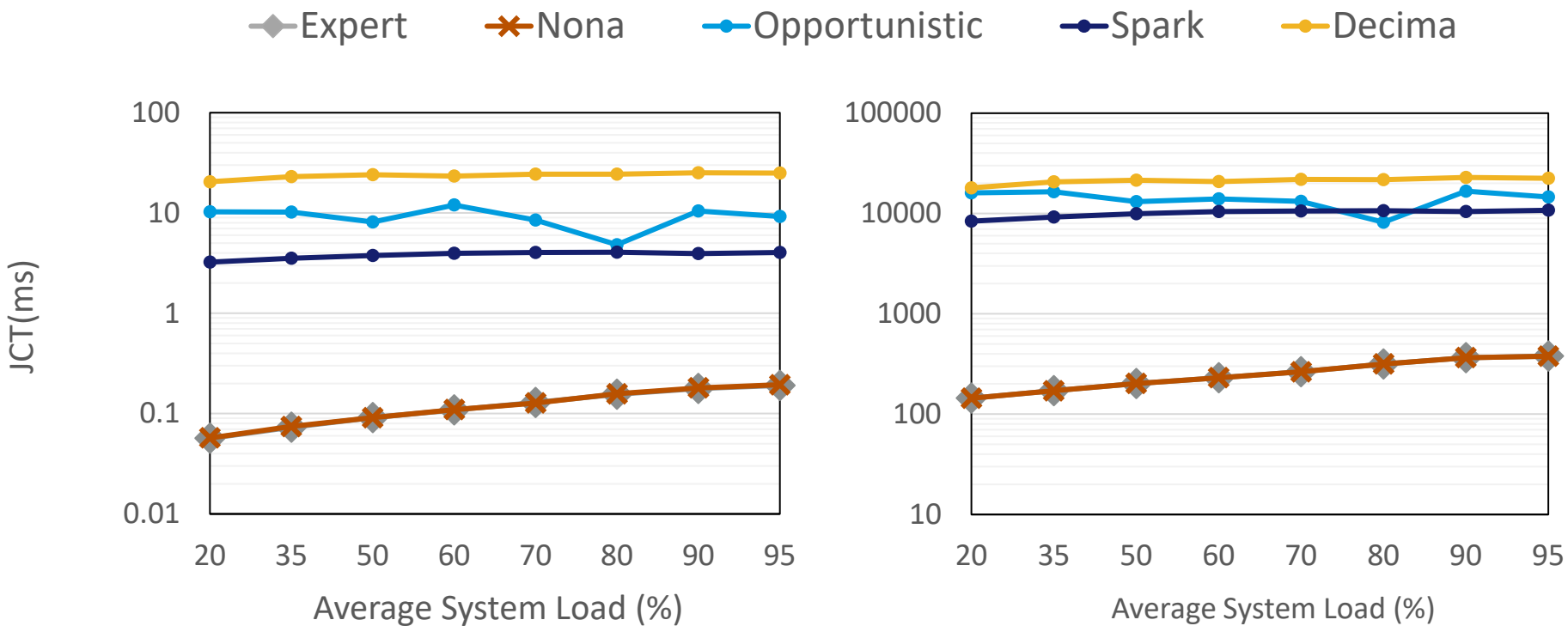
Impact of Job Arrival Rate



(a) Average over all jobs

(b) Average over foreground jobs

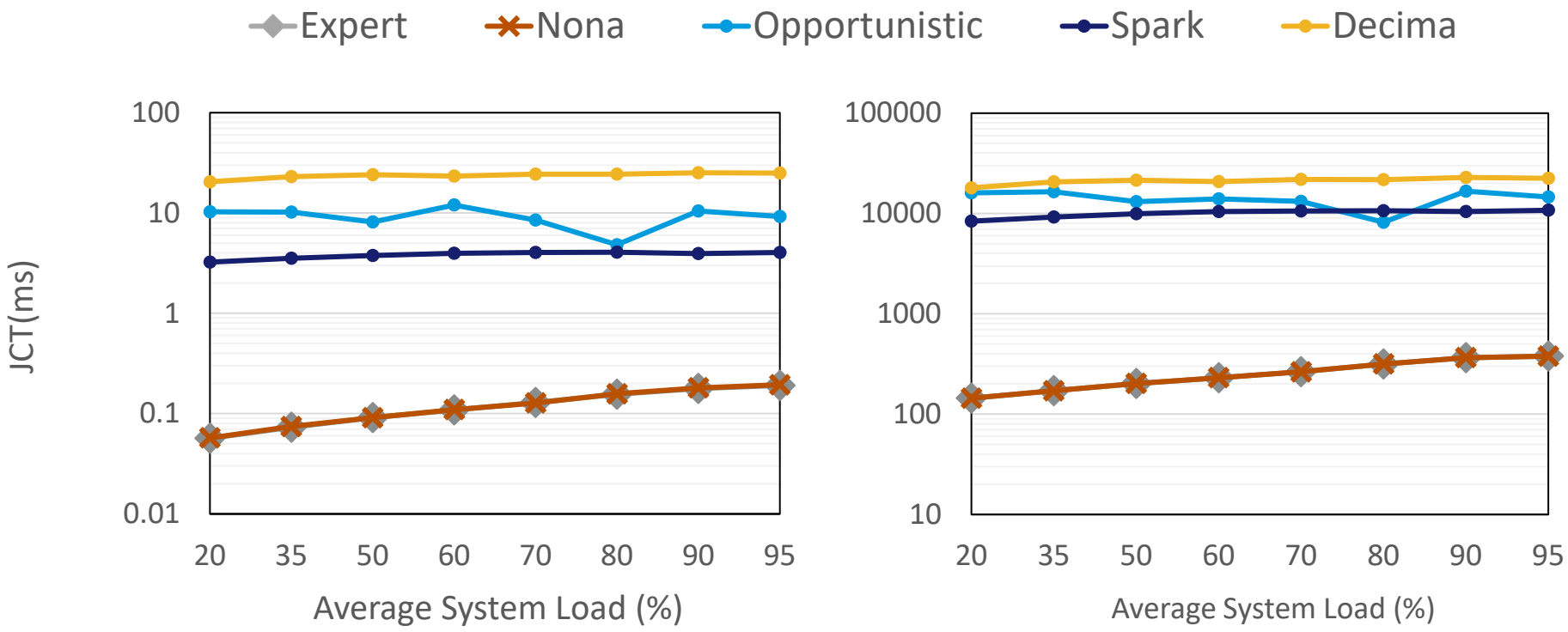
Impact of Job Arrival Rate



(a) Average over all jobs

(b) Average over foreground jobs

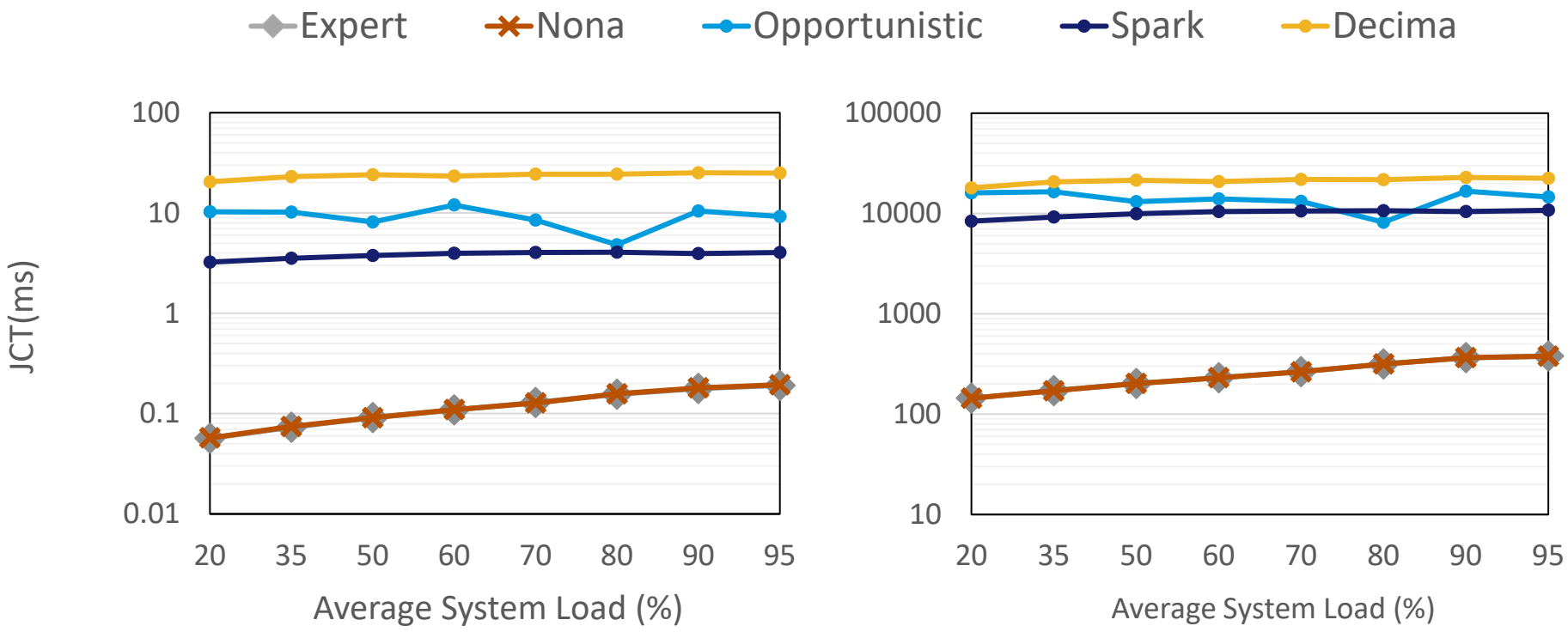
Impact of Job Arrival Rate



(a) Average over all jobs

(b) Average over foreground jobs

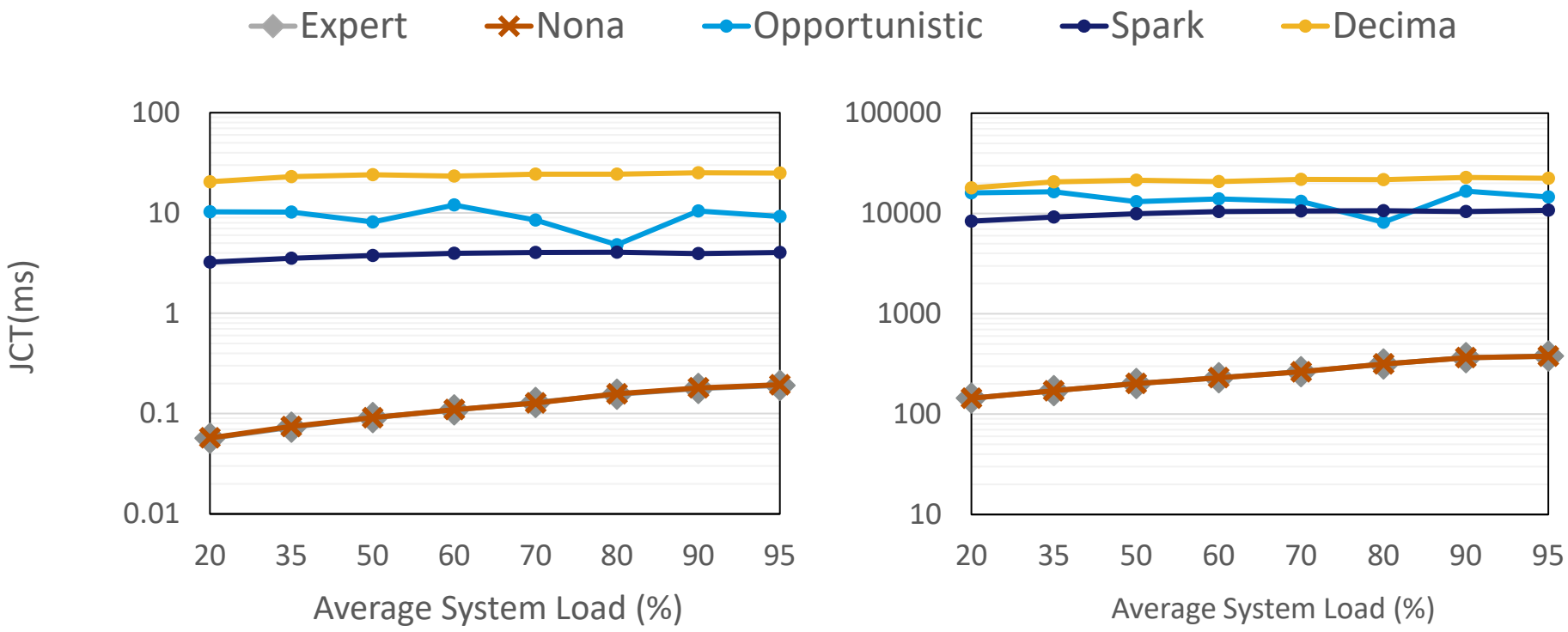
Impact of Job Arrival Rate



(a) Average over all jobs

(b) Average over foreground jobs

Impact of Job Arrival Rate



(a) Average over all jobs

(b) Average over foreground jobs

Conclusion

- Schedulers should consider network queueing and asymmetry in jobs' DAGs:
  - Nona achieves up to 350x better JCT than previous works by including network delays in the decision-making
- ML operation graphs can be largely compacted to simplify scheduling
- Stochastic scheduling allows moving the complexity of the problem offline for low latency scheduling