ELiSE

A tool to support algorithmic design for
HPC co-scheduling

JSSPP

IPDPS 2025 • Milano, Italy

Efstratios Karapanagiotis, Nikolaos Triantafyllis, Athanasios Tsoukleidis-Karydakis, Georgios Goumas, Nectarios Koziris

National Technical University of Athens
CSLab

Introduction

What is ELiSE?

- **E**fficient **L**ightweight **S**cheduling **E**stimator
- A framework for fast prototyping and evaluation of scheduling and co-scheduling algorithms for HPC systems

Why did we create it?

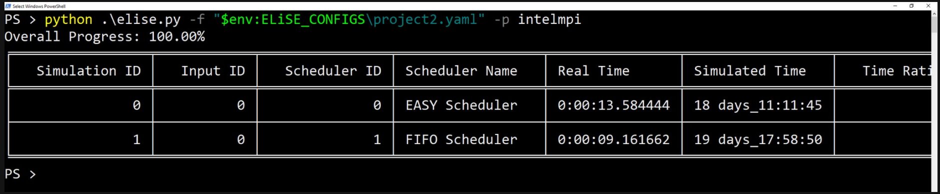
- Testing in a real HPC cluster is time-consuming and expensive
- Access to large scale resources is not straightforward
- Challenges in extending a real scheduler
- No administrative privileges for such changes

Why co-scheduling capability?

- Sharing node resources of memory-intensive jobs can boost performance
- Through it, potential gains in system throughput and reduced job wait times

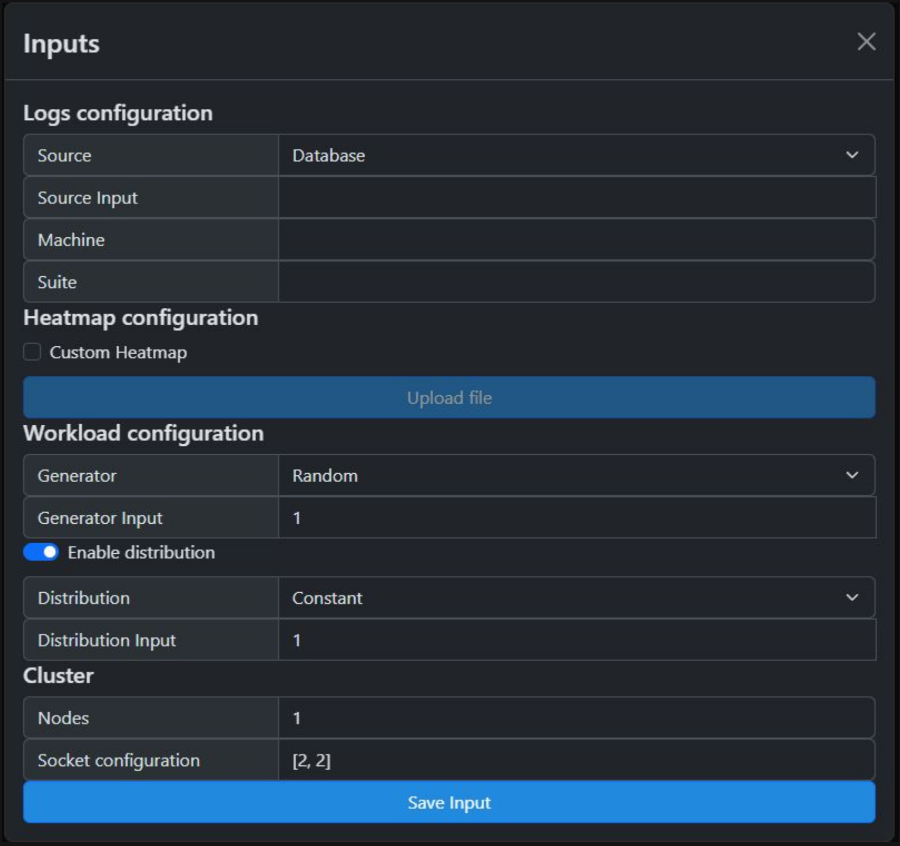
ELiSE: Features

- A graphical, web and command-line interface



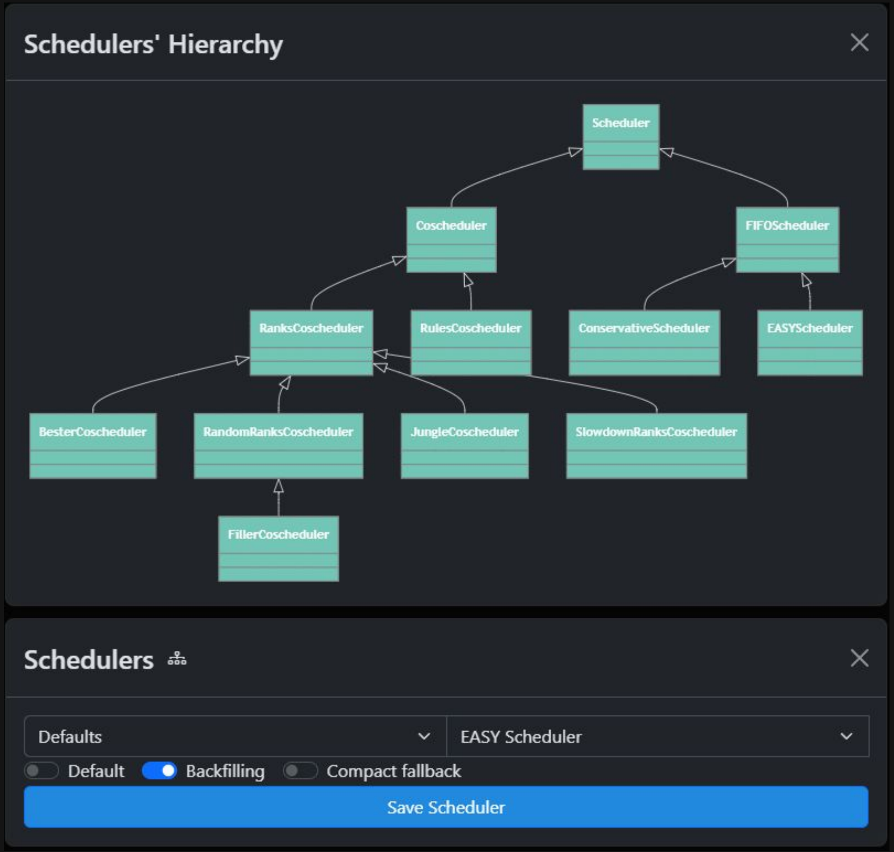
ELiSE: Features

- A graphical, web and command-line interface
- Workload generation with in-built or custom algorithms



ELiSE: Features

- A graphical, web and command-line interface
- Workload generation with in-built or custom algorithms
- Bundled schedulers or user-defined



ELiSE: Features

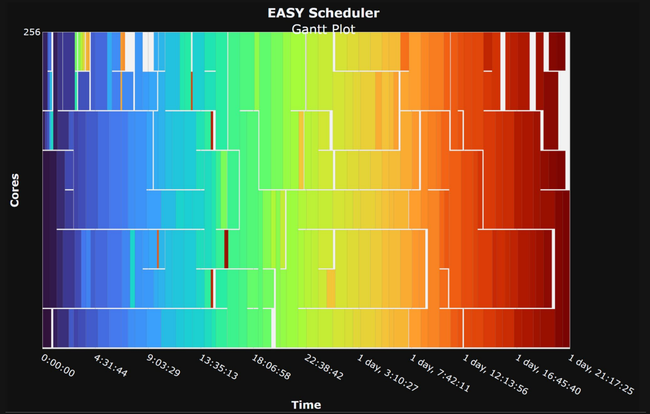
- A graphical, web and command-line interface
- Workload generation with in-built or custom algorithms
- Bundled schedulers or user-defined
- Reports and diagrams for scheduling evaluation:
  - General simulation, real time reports

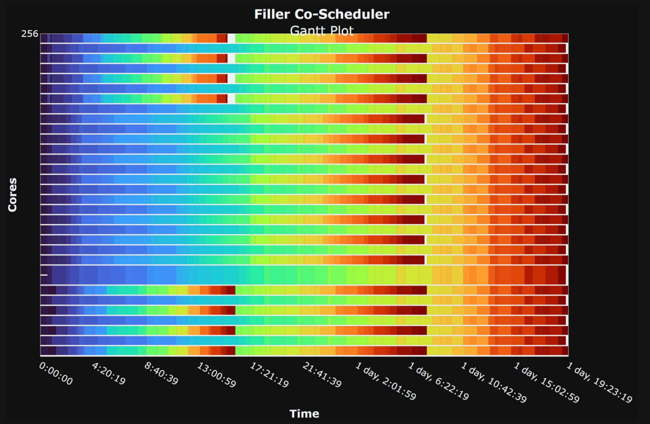
Simulation ID	Input ID	Scheduler ID	Scheduler Name	Real Time	Simulated Time	Time Ratio (Simulated Days / 1 real hour)
0	0	0	FCFS Scheduler	0:00:03.137	9 days 16:35:05	11121.375
1	0	1	EASY Scheduler	0:00:05.137	9 days 1:46:45	6359.114
2	0	2	Filler Co-Scheduler	0:00:04.582	8 days 12:41:51	6701.856

Job Number	Submit Time	Wait Time	Run Time	Number of Allocated Processors	Average CPU Time Used	Used Memory	Requested Number of Processors	Reque Ti
40	0.000	34845.000	4540.000	64			64	6356
41	0.000	34845.000	3470.000	32			32	4858
42	0.000	12165.000	2630.000	32			32	3682
43	0.000	37275.000	3580.000	128			121	5012
44	0.000	37275.000	3470.000	32			32	4858
45	0.000	38315.000	2630.000	32			32	3682

ELiSE: Features

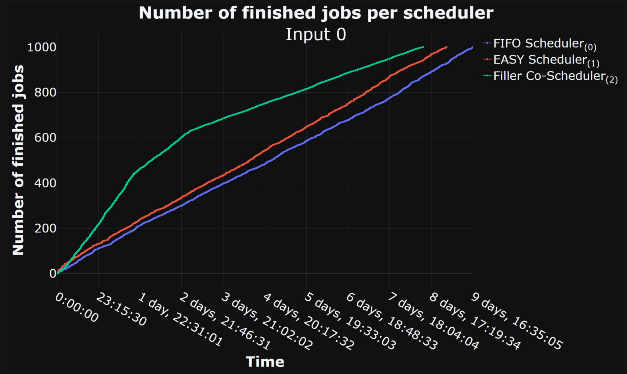
- A graphical, web and command-line interface
- Workload generation with in-built or custom algorithms
- Bundled schedulers or user-defined
- Reports and diagrams for scheduling evaluation:
    - General simulation, real time reports
    - Gantt diagrams

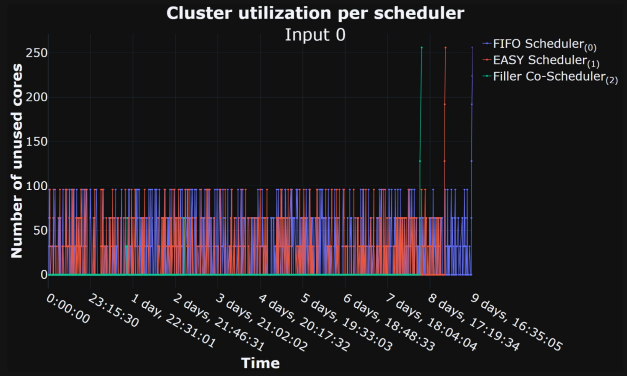




ELiSE: Features

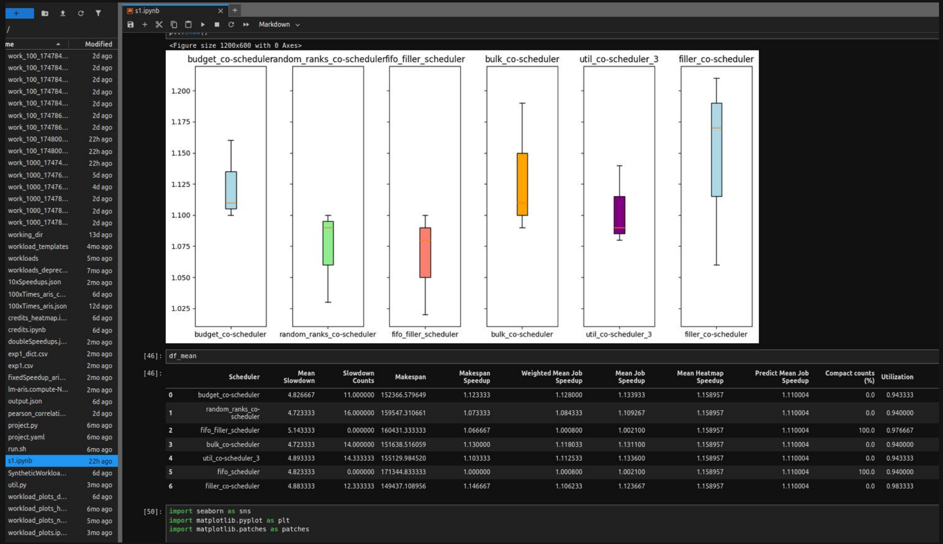
- A graphical, web and command-line interface
- Workload generation with in-built or custom algorithms
- Bundled schedulers or user-defined
- Reports and diagrams for scheduling evaluation:
  - General simulation, real time reports
  - Gantt diagrams
  - Throughput, Waiting Queue and System Utilization





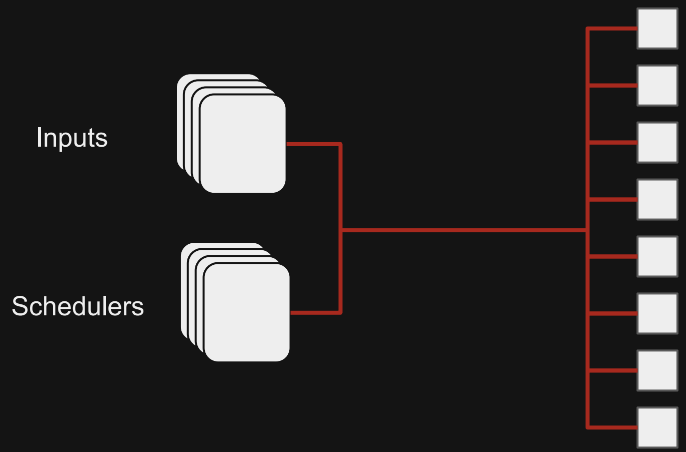
ELiSE: Features

- A graphical, web and command-line interface
- Workload generation with in-built or custom algorithms
- Bundled schedulers or user-defined
- Reports and diagrams for scheduling evaluation:
  - General simulation, real time reports
  - Gantt diagrams
  - Throughput, Waiting Queue and System Utilization
- Full experiment traceability for post-analysis



ELiSE: Features

- A graphical, web and command-line interface
- Workload generation with in-built or custom algorithms
- Bundled schedulers or user-defined
- Reports and diagrams for scheduling evaluation:
  - General simulation, real time reports
  - Gantt diagrams
  - Throughput, Waiting Queue and System Utilization
- Full experiment traceability for post-analysis
- Distribution of multiple simulations in parallel



Providers

- Python Multiprocessing
- Open MPI
- Intel MPI

What is co-location?

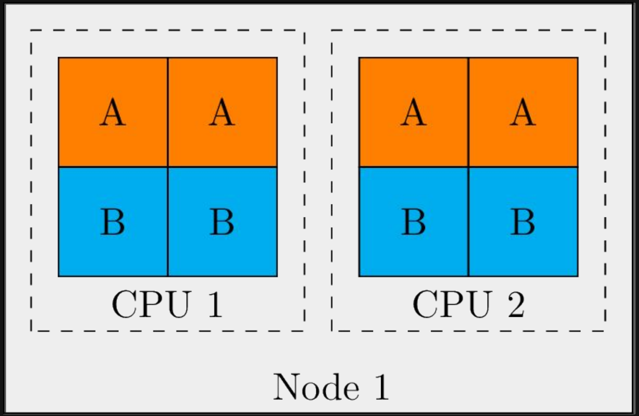
Node sharing between MPI applications

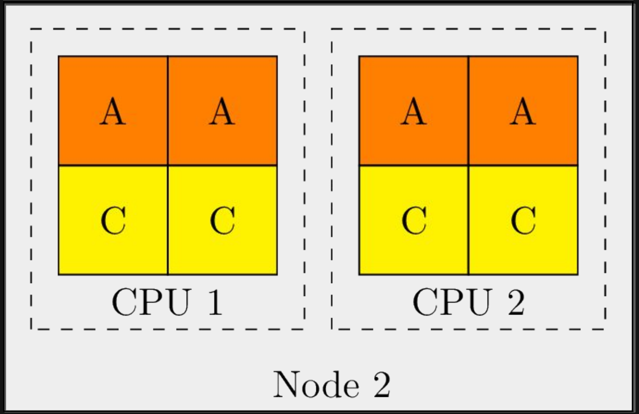
Current strategy:
Half Socket Allocation per app

Example of Co-location of 3 MPI apps:

- App A, 8 cores
- App B, 4 cores
- App C, 4 cores

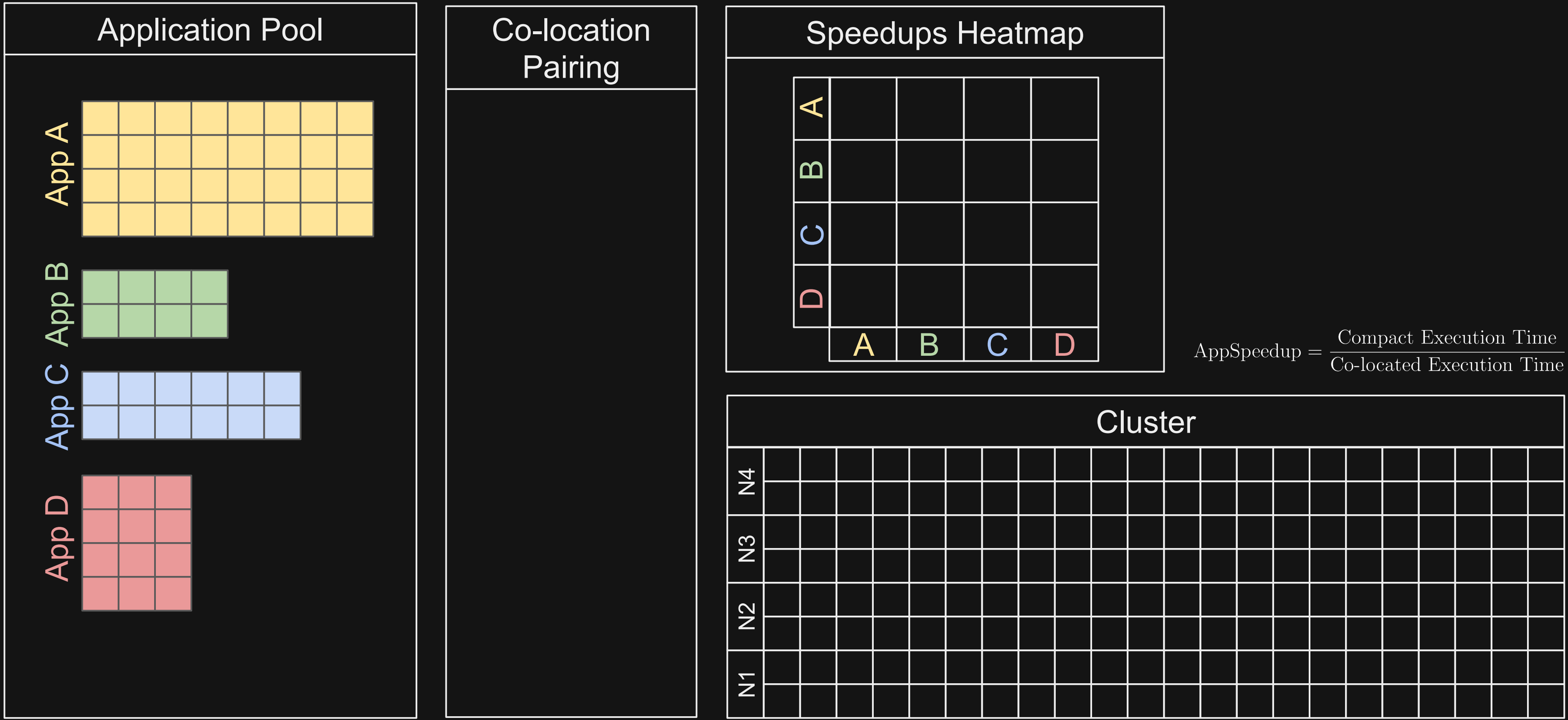
on 2 nodes with 2 CPUs (4 cores each)





What is a Heatmap of Speedups?

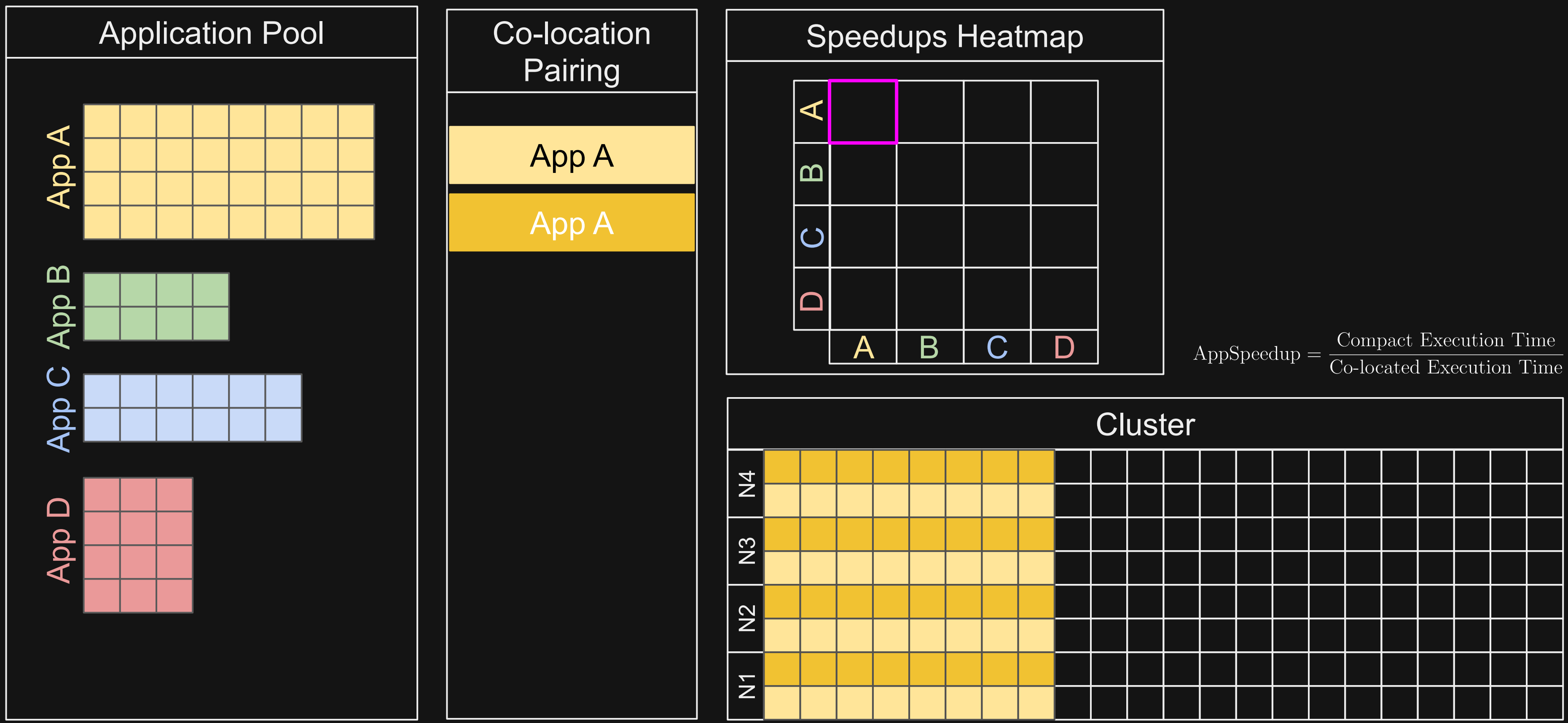
Computation Method



$$\text{AppSpeedup} = \frac{\text{Compact Execution Time}}{\text{Co-located Execution Time}}$$

What is a Heatmap of Speedups?

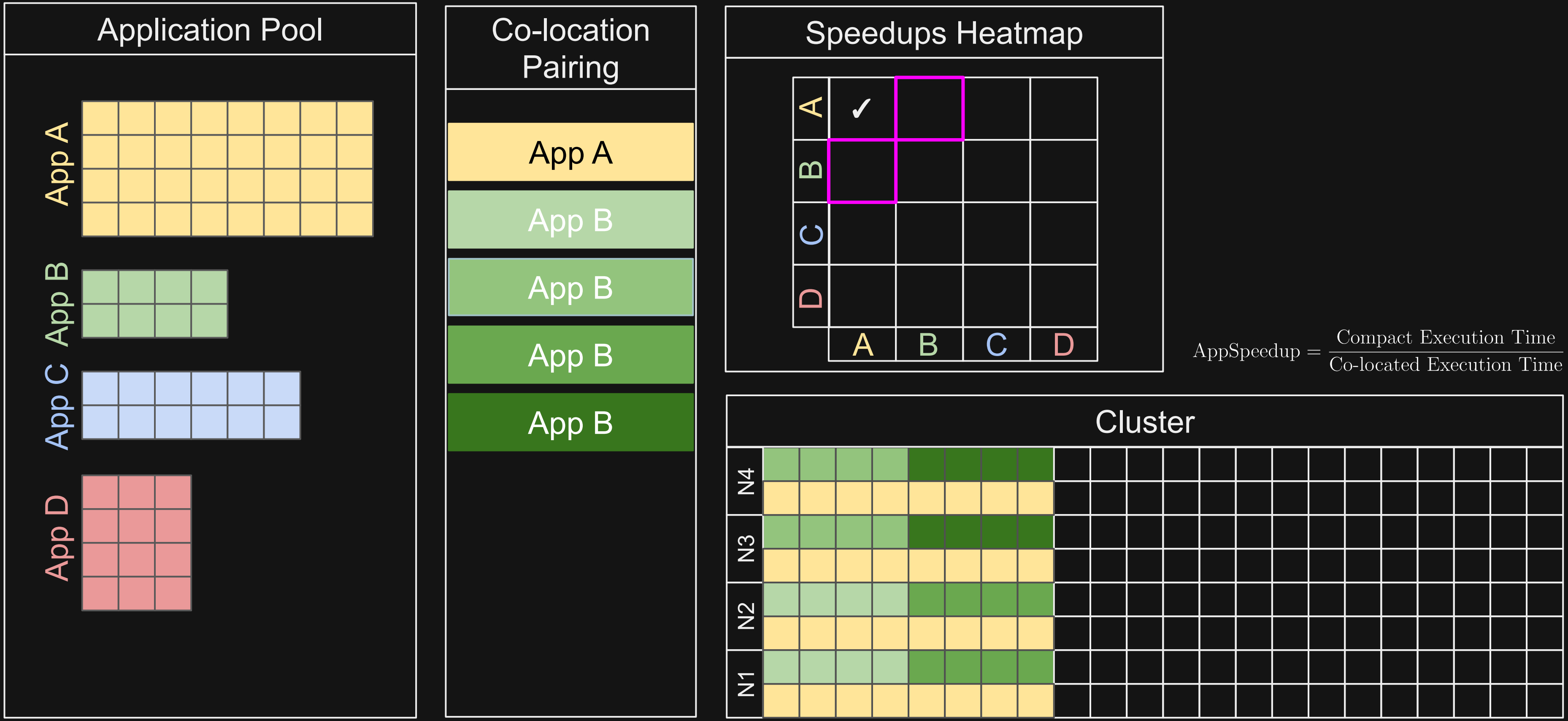
Computation Method



$$\text{AppSpeedup} = \frac{\text{Compact Execution Time}}{\text{Co-located Execution Time}}$$

What is a Heatmap of Speedups?

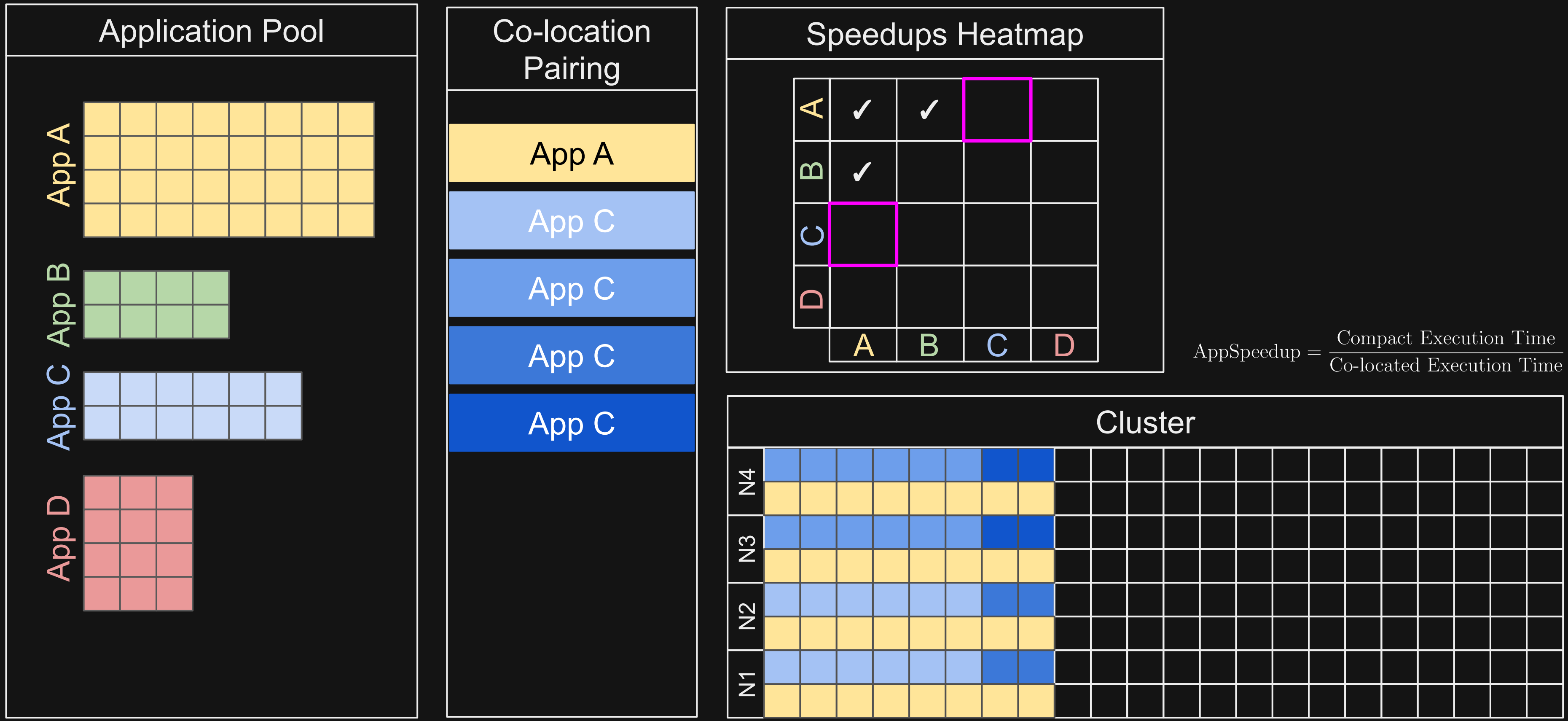
Computation Method



$$\text{AppSpeedup} = \frac{\text{Compact Execution Time}}{\text{Co-located Execution Time}}$$

What is a Heatmap of Speedups?

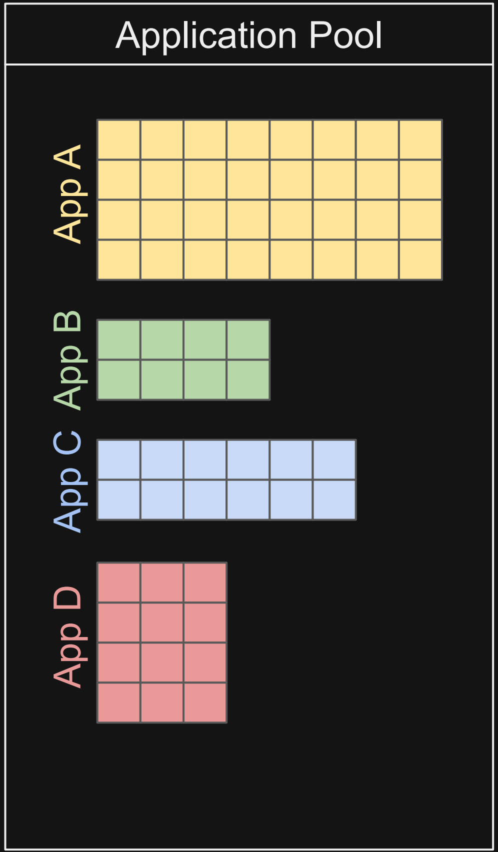
Computation Method

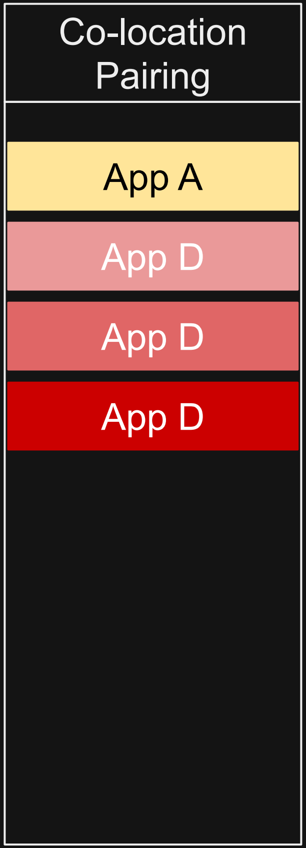


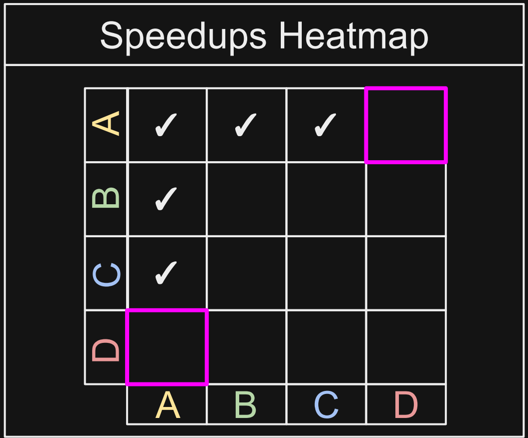
$$\text{AppSpeedup} = \frac{\text{Compact Execution Time}}{\text{Co-located Execution Time}}$$

What is a Heatmap of Speedups?

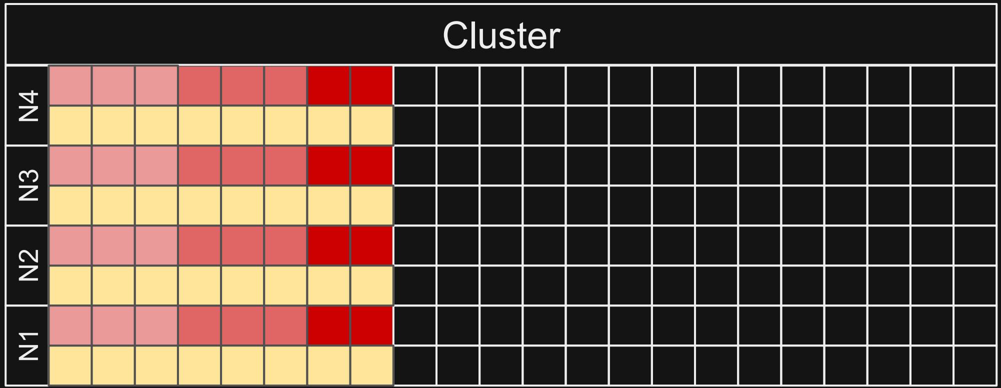
Computation Method





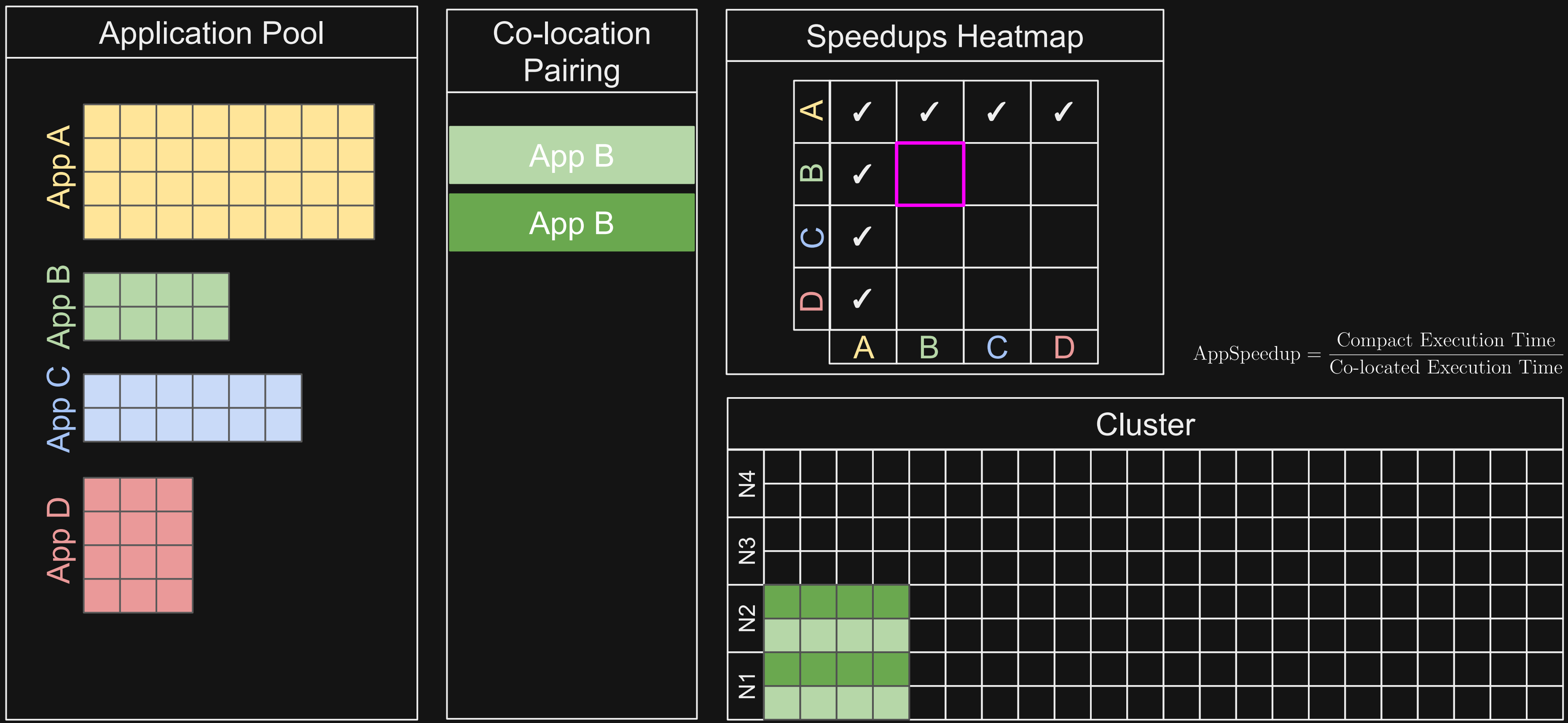


$$\text{AppSpeedup} = \frac{\text{Compact Execution Time}}{\text{Co-located Execution Time}}$$



What is a Heatmap of Speedups?

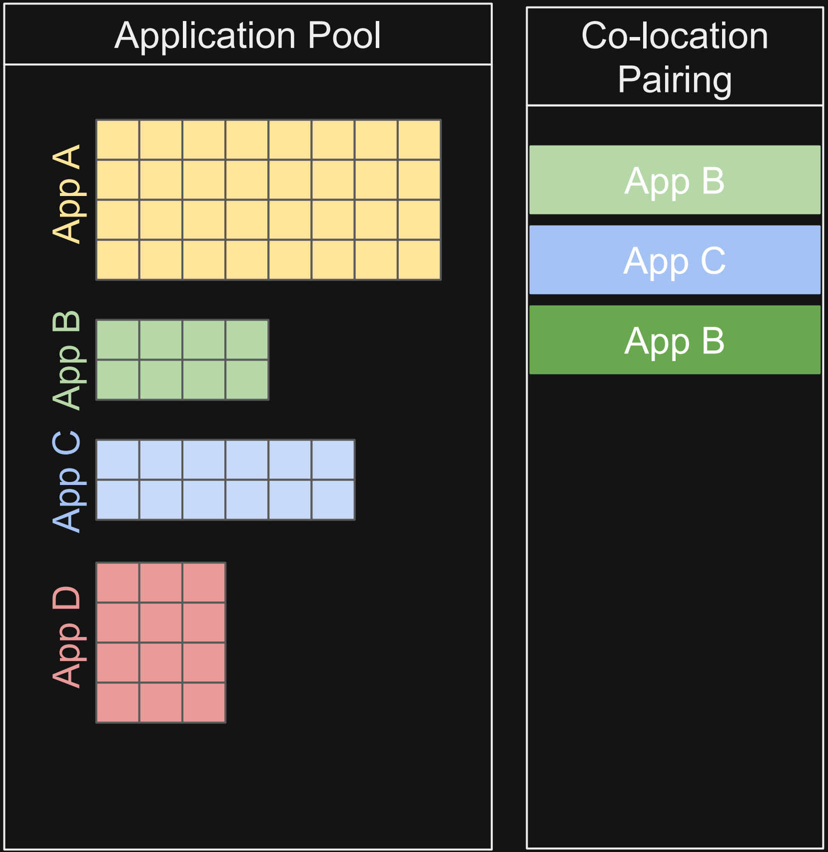
Computation Method

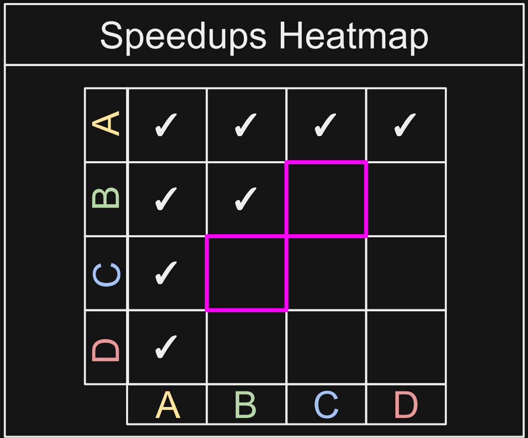


$$\text{AppSpeedup} = \frac{\text{Compact Execution Time}}{\text{Co-located Execution Time}}$$

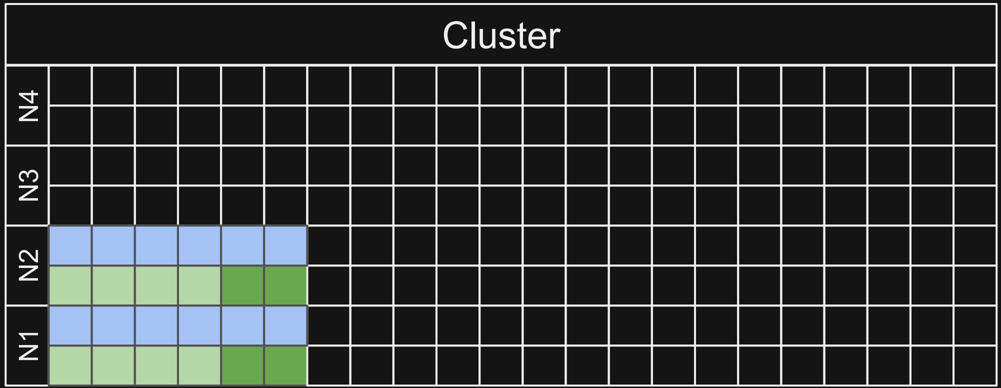
What is a Heatmap of Speedups?

Computation Method



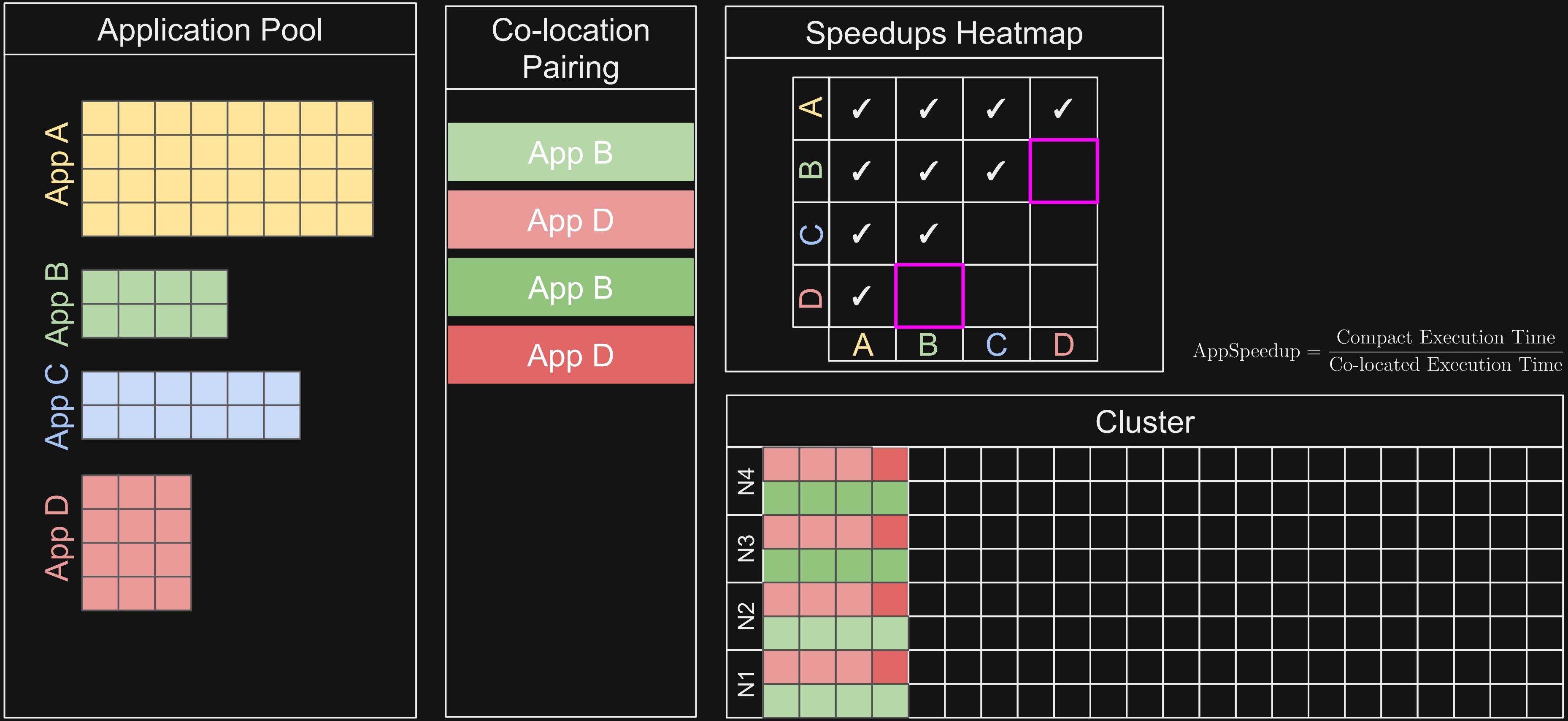


$$\text{AppSpeedup} = \frac{\text{Compact Execution Time}}{\text{Co-located Execution Time}}$$



What is a Heatmap of Speedups?

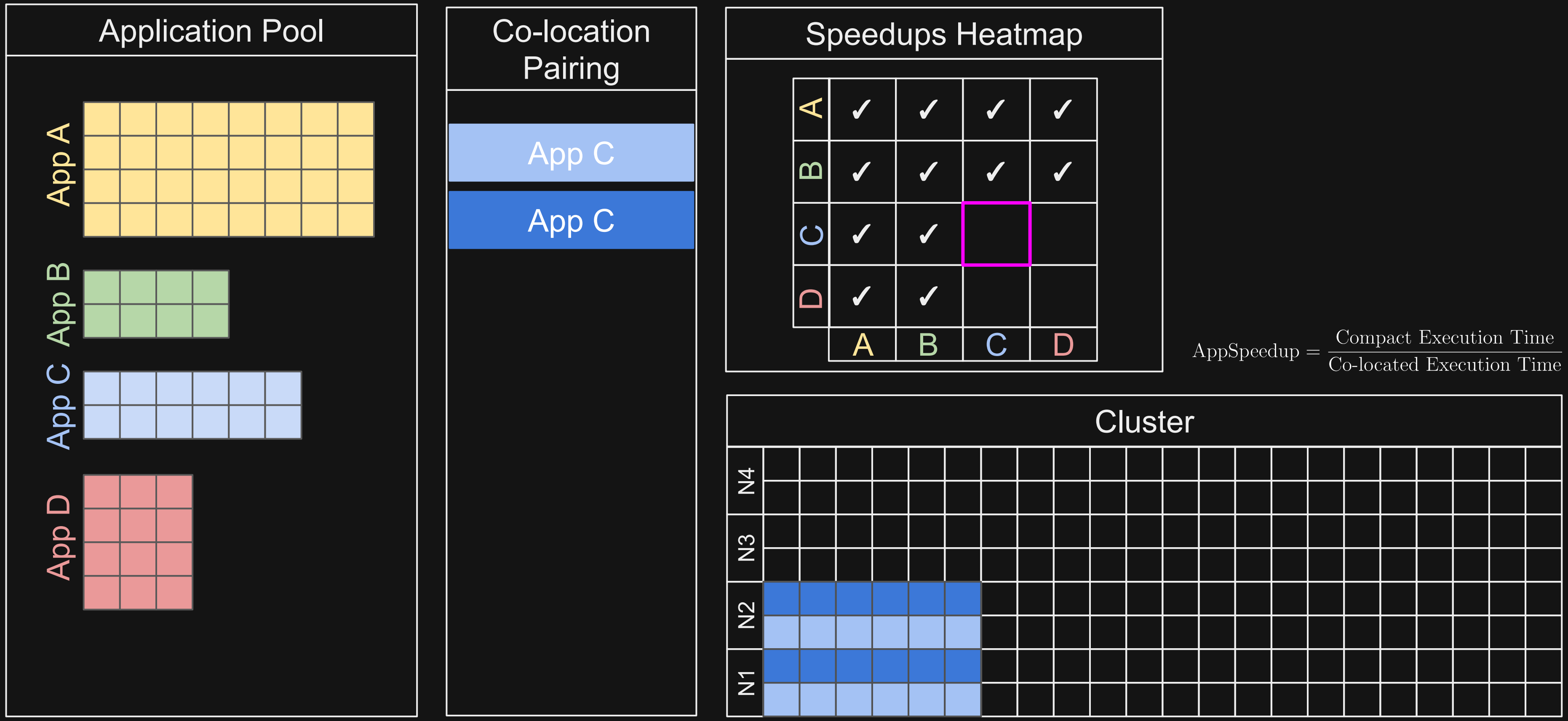
Computation Method



$$\text{AppSpeedup} = \frac{\text{Compact Execution Time}}{\text{Co-located Execution Time}}$$

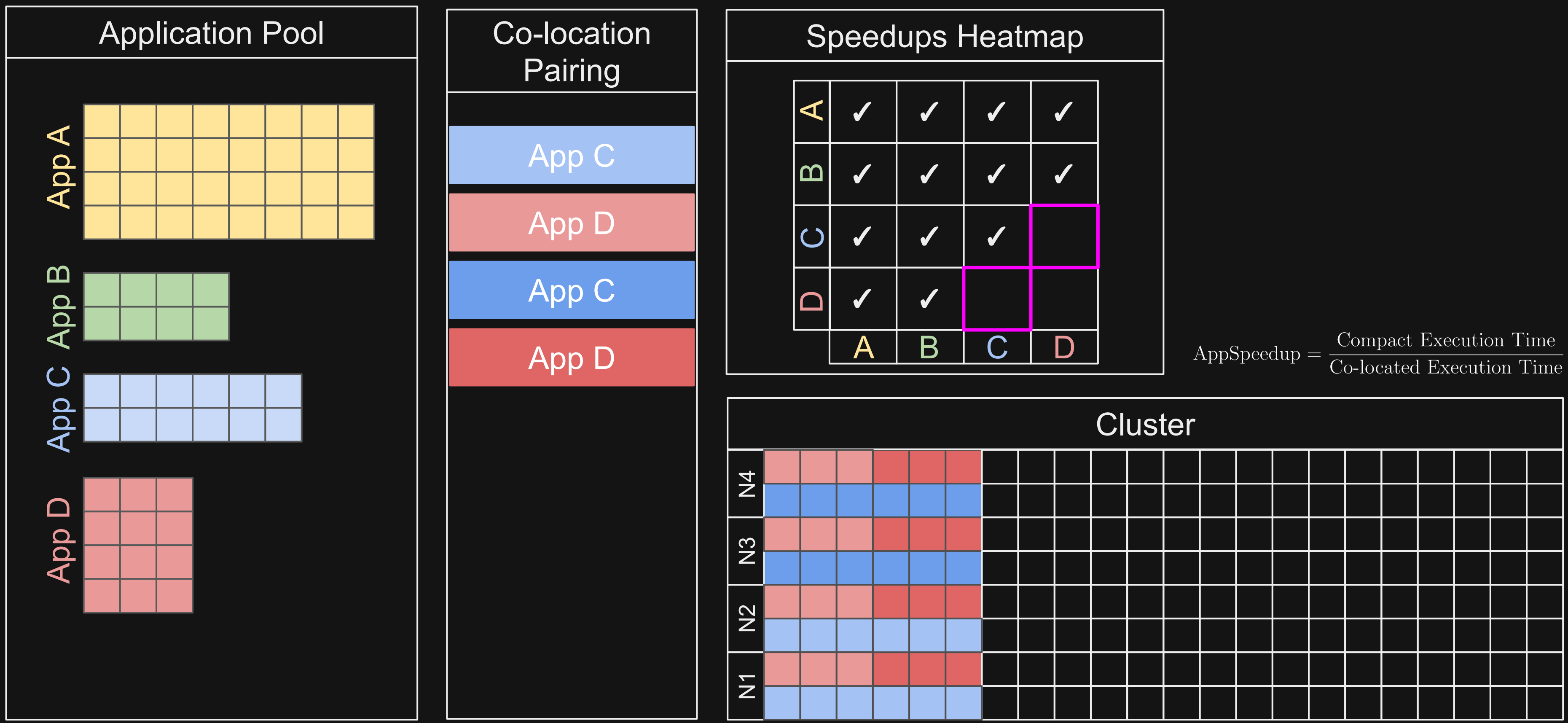
What is a Heatmap of Speedups?

Computation Method



$$\text{AppSpeedup} = \frac{\text{Compact Execution Time}}{\text{Co-located Execution Time}}$$

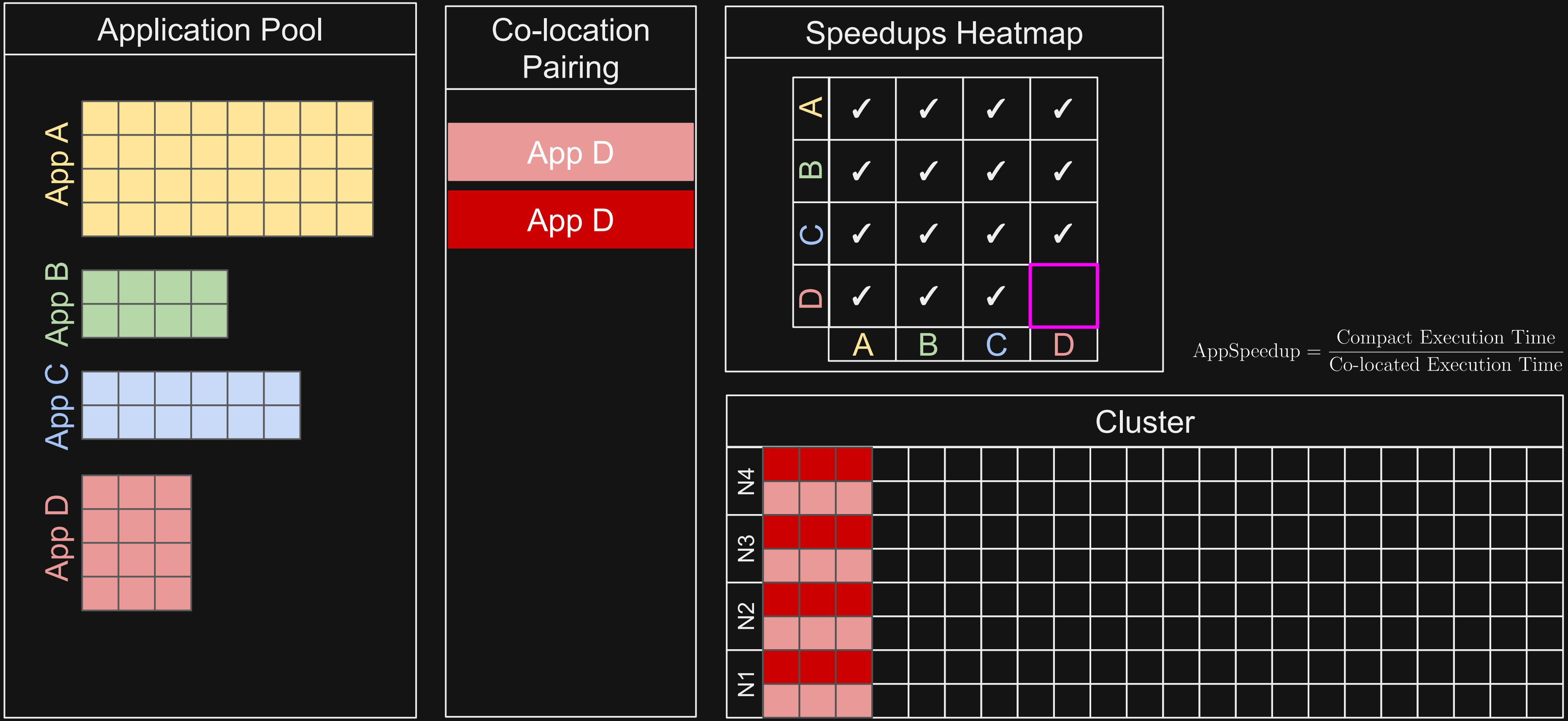
What is a Heatmap of Speedups?
Computation Method



$$\text{AppSpeedup} = \frac{\text{Compact Execution Time}}{\text{Co-located Execution Time}}$$

What is a Heatmap of Speedups?

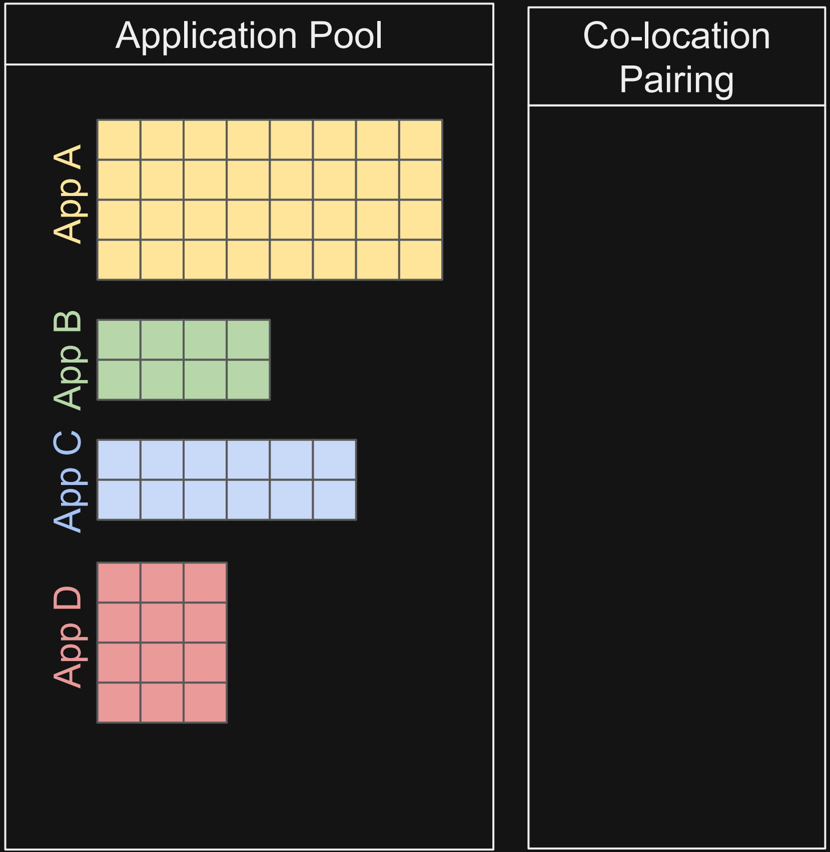
Computation Method

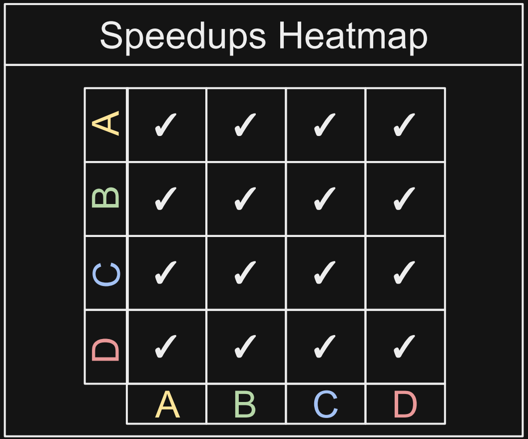


$$\text{AppSpeedup} = \frac{\text{Compact Execution Time}}{\text{Co-located Execution Time}}$$

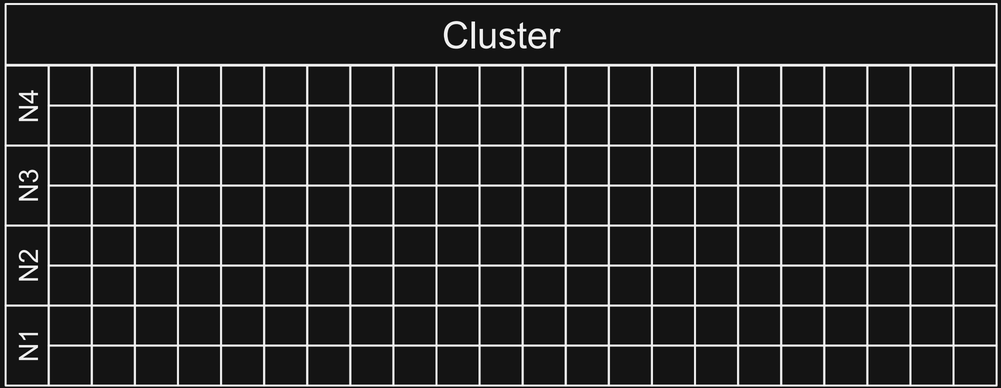
What is a Heatmap of Speedups?

Computation Method



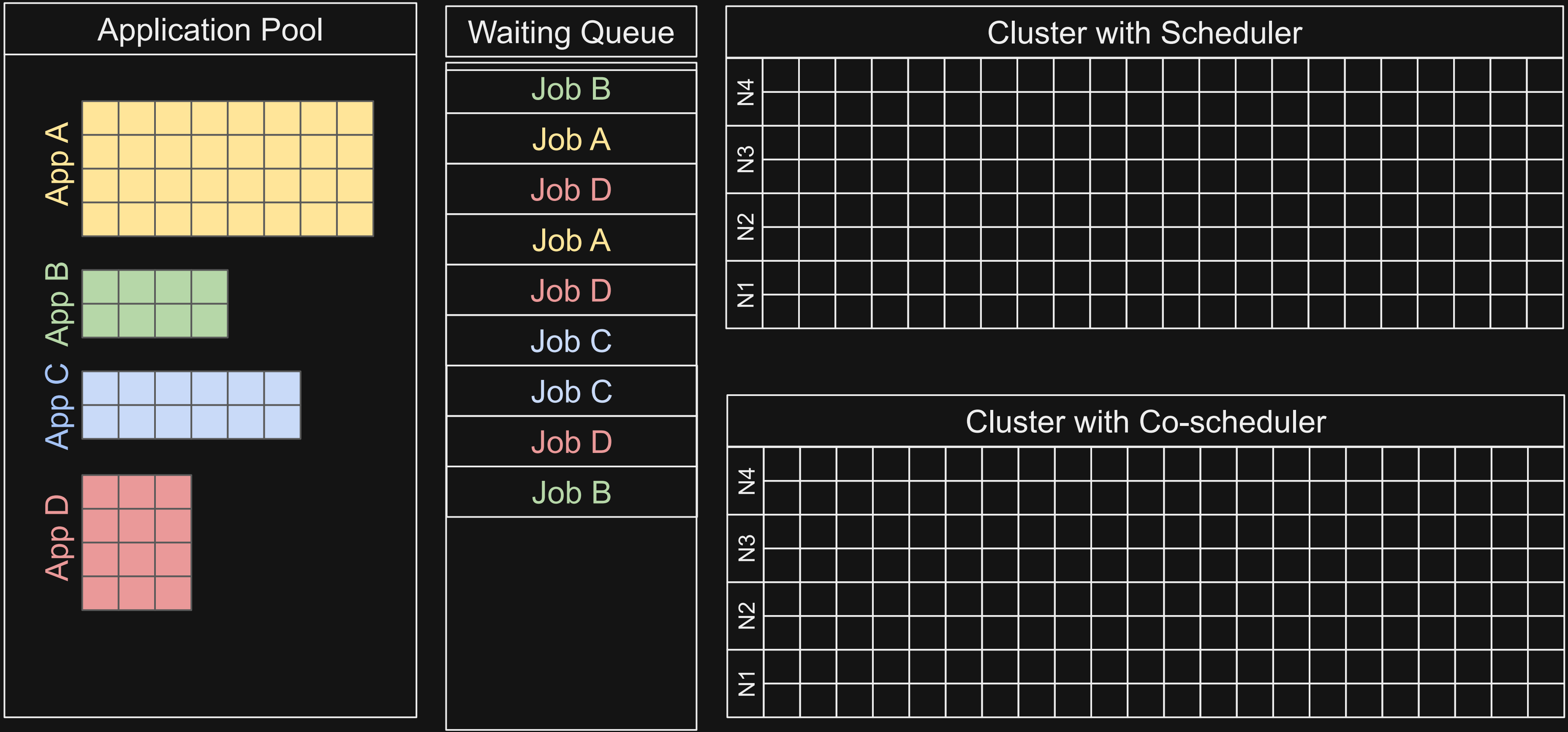


$$\text{AppSpeedup} = \frac{\text{Compact Execution Time}}{\text{Co-located Execution Time}}$$



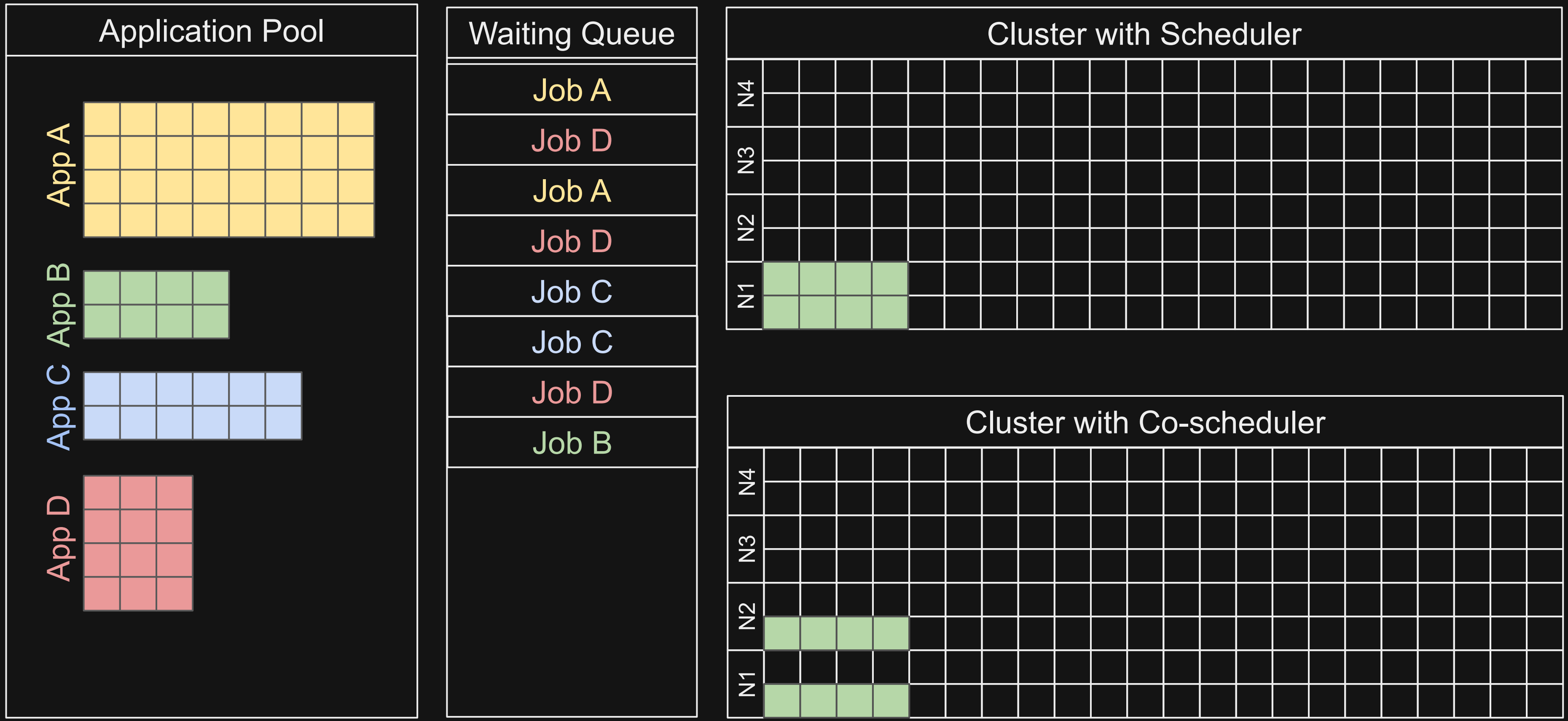
What is co-scheduling?

Co-location in the time domain - FCFS example



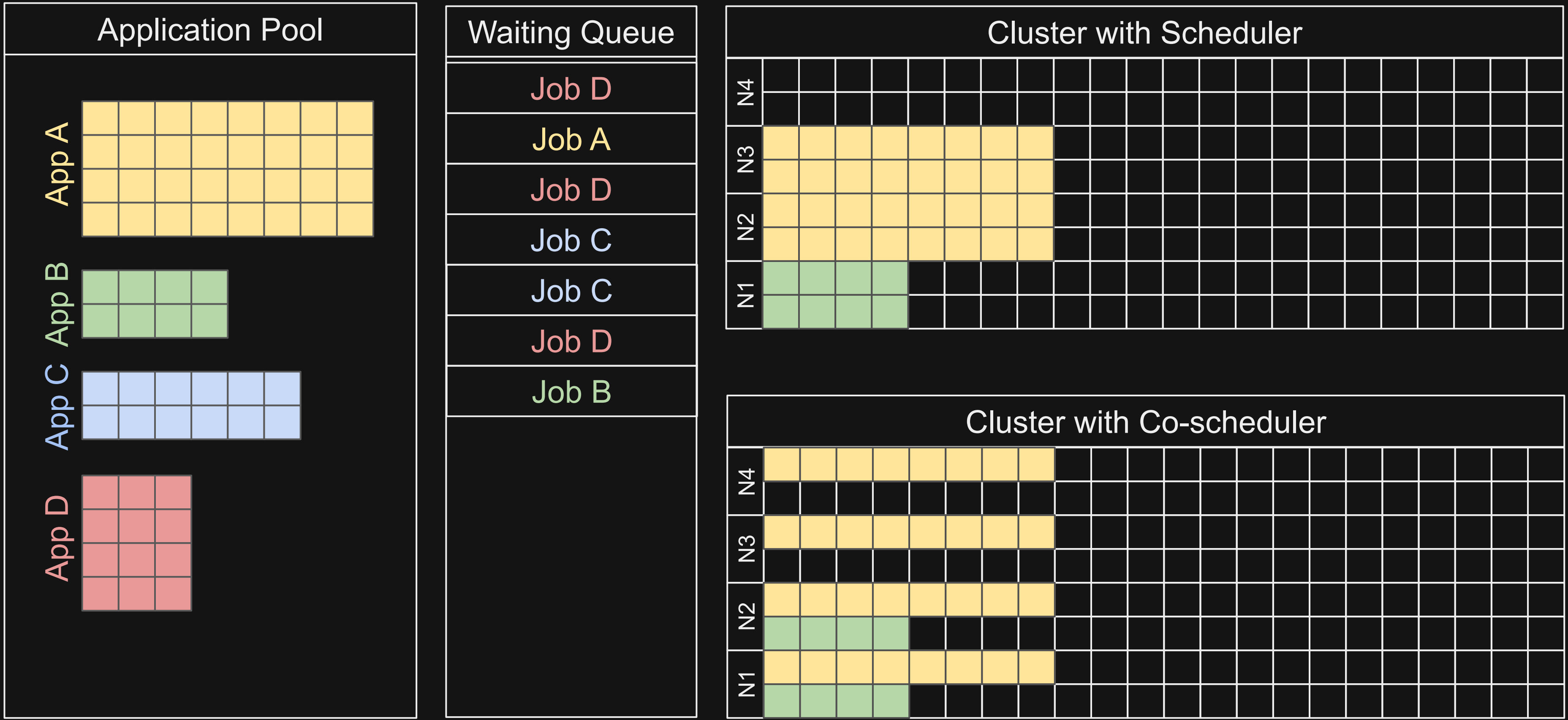
What is co-scheduling?

Co-location in the time domain - FCFS example



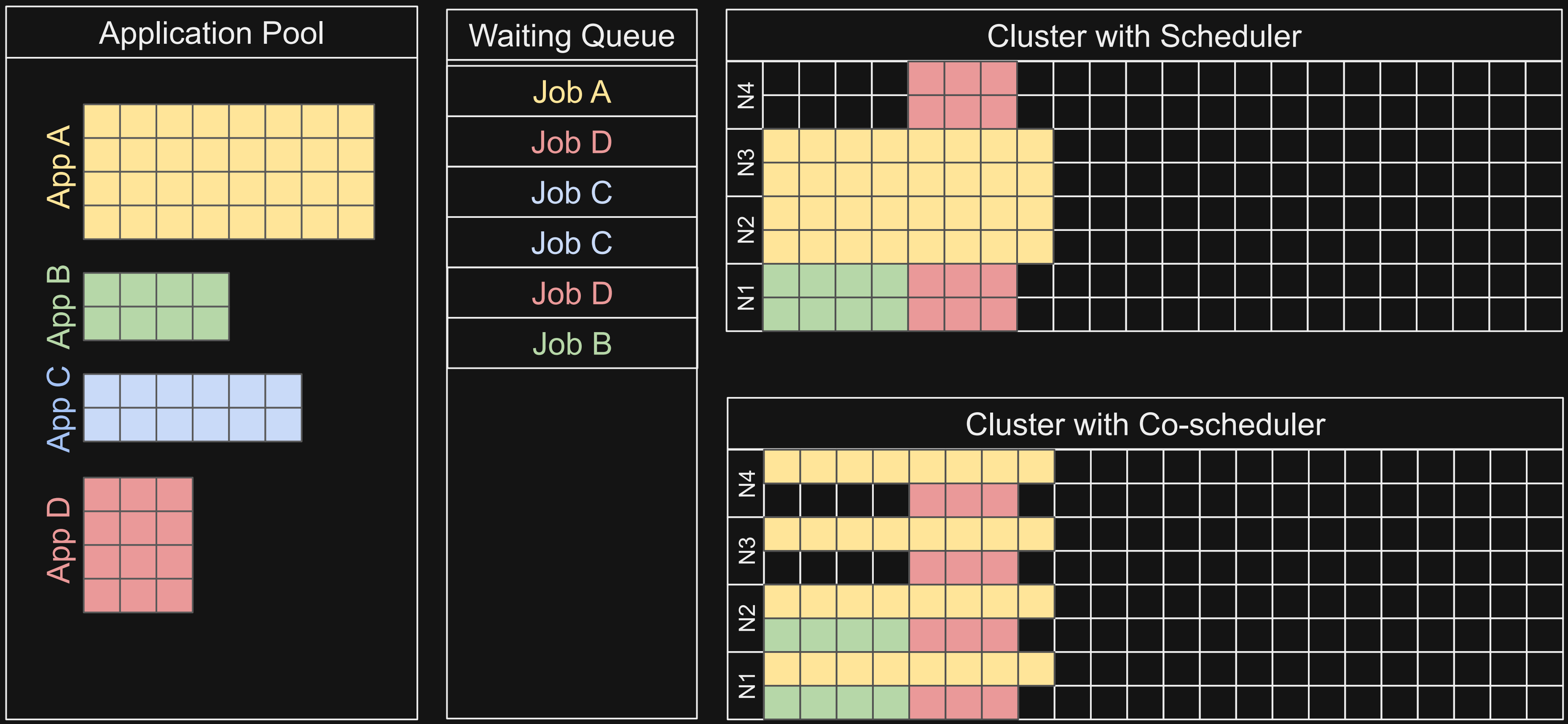
What is co-scheduling?

Co-location in the time domain - FCFS example



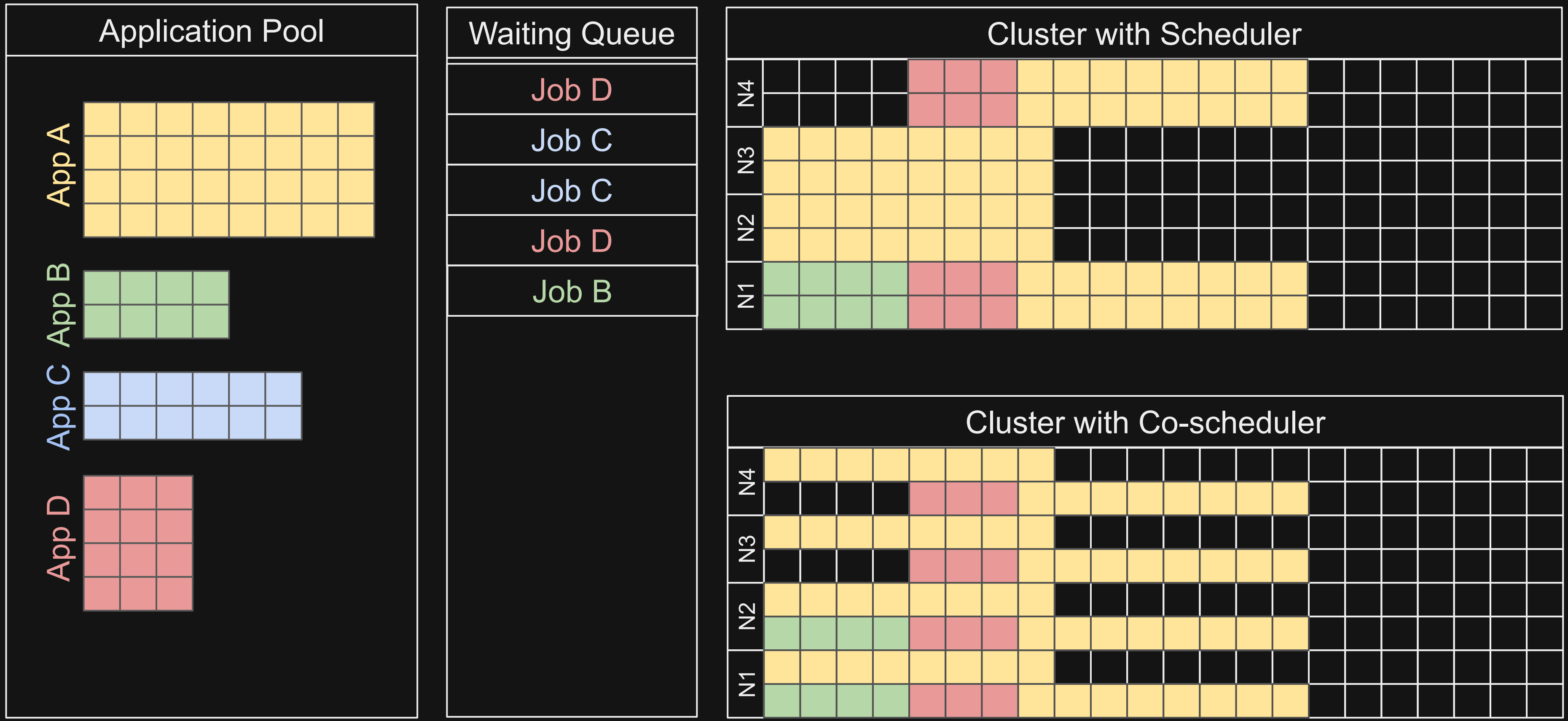
What is co-scheduling?

Co-location in the time domain - FCFS example



What is co-scheduling?

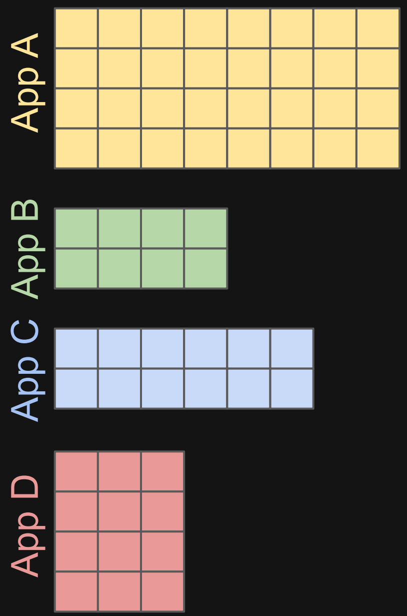
Co-location in the time domain - FCFS example



What is co-scheduling?

Co-location in the time domain - FCFS example

Application Pool



Waiting Queue

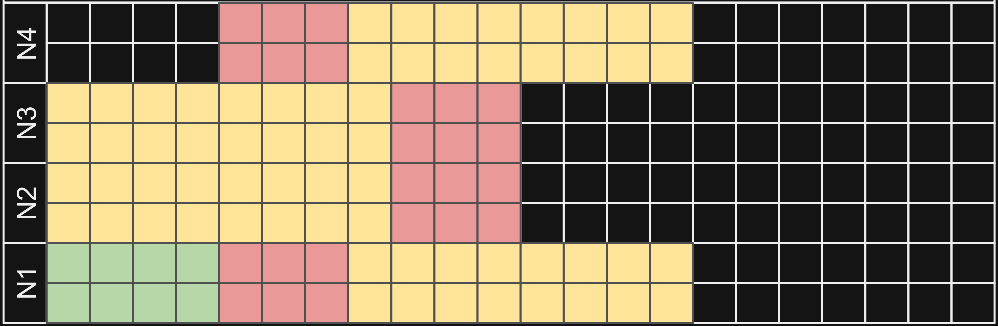
Job C

Job C

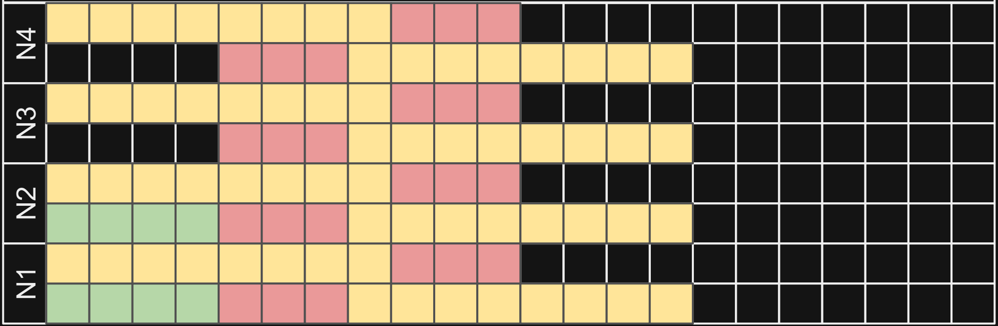
Job D

Job B

Cluster with Scheduler

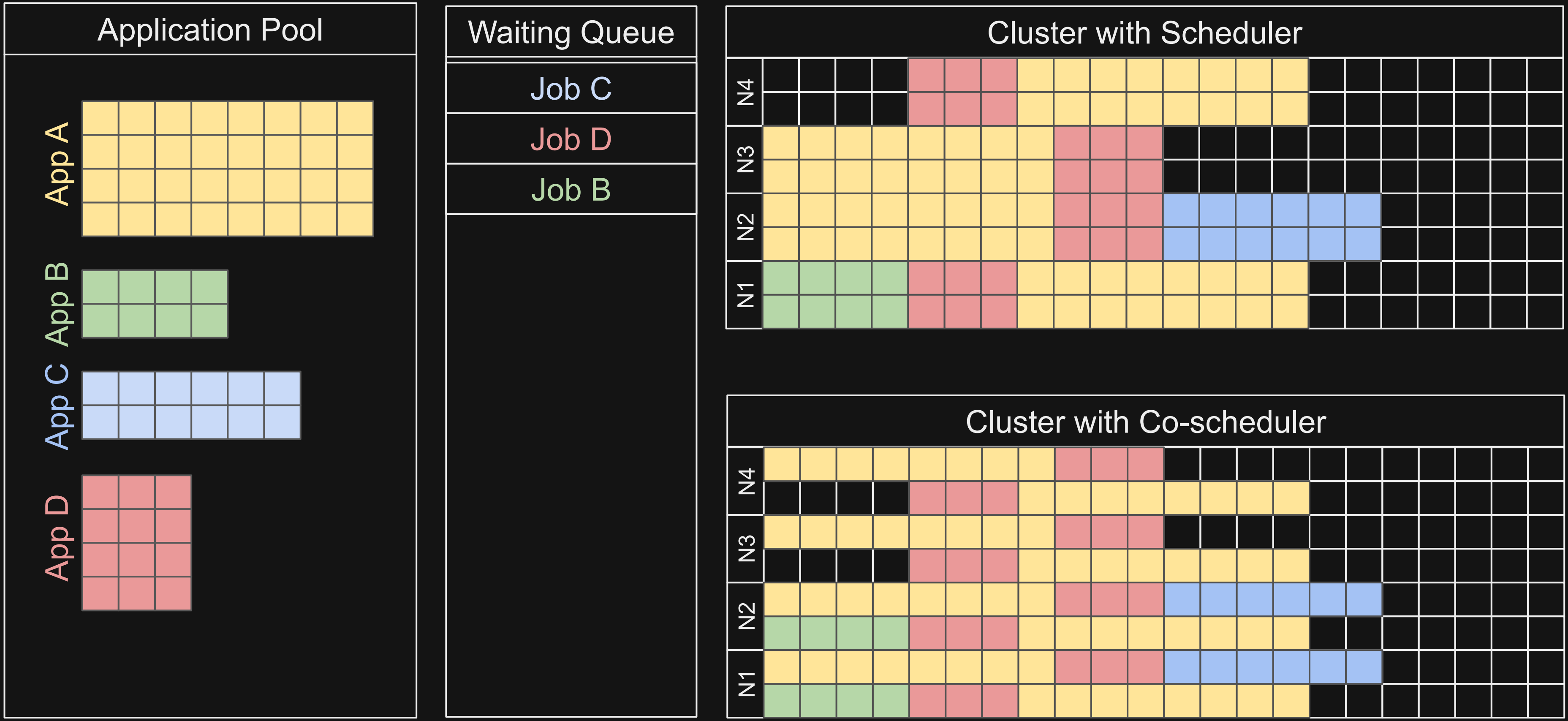


Cluster with Co-scheduler



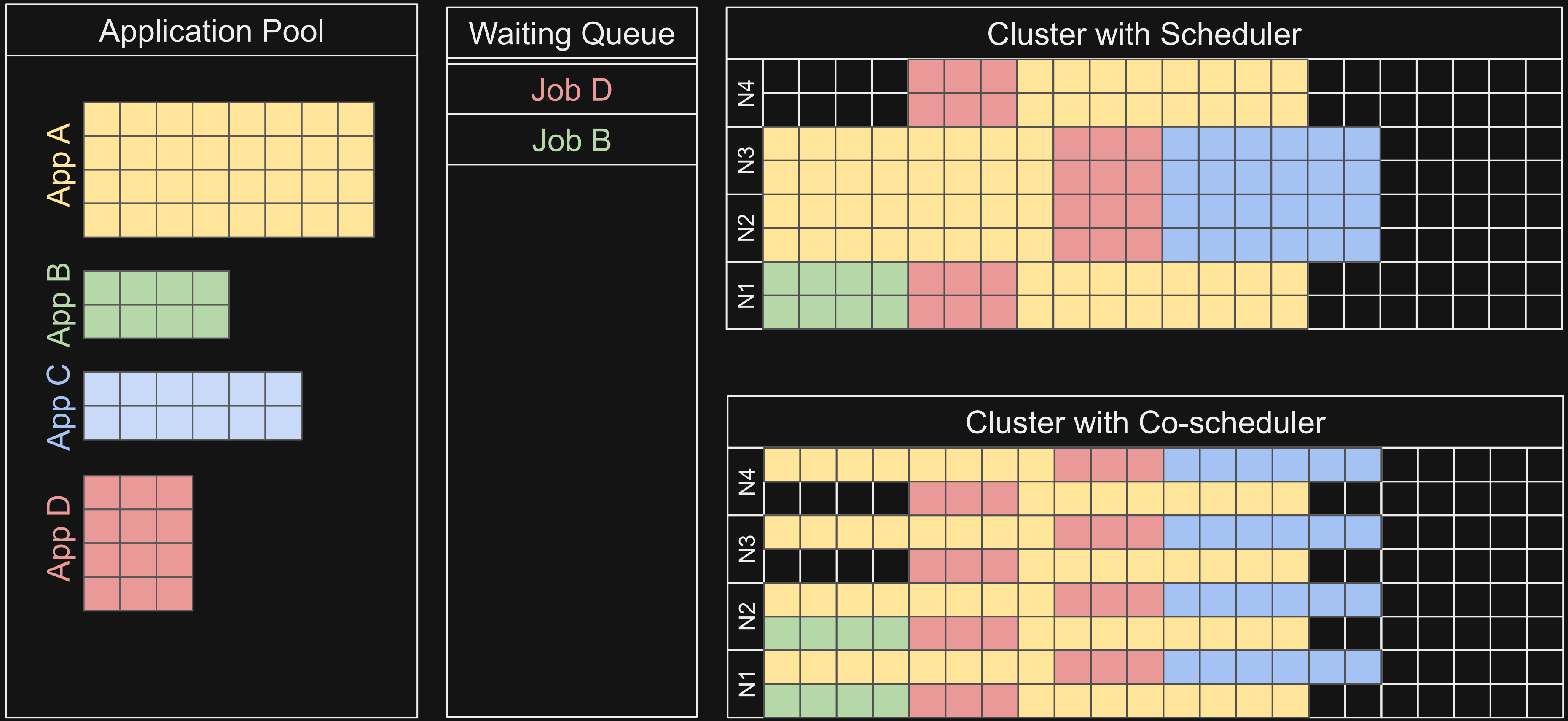
What is co-scheduling?

Co-location in the time domain - FCFS example



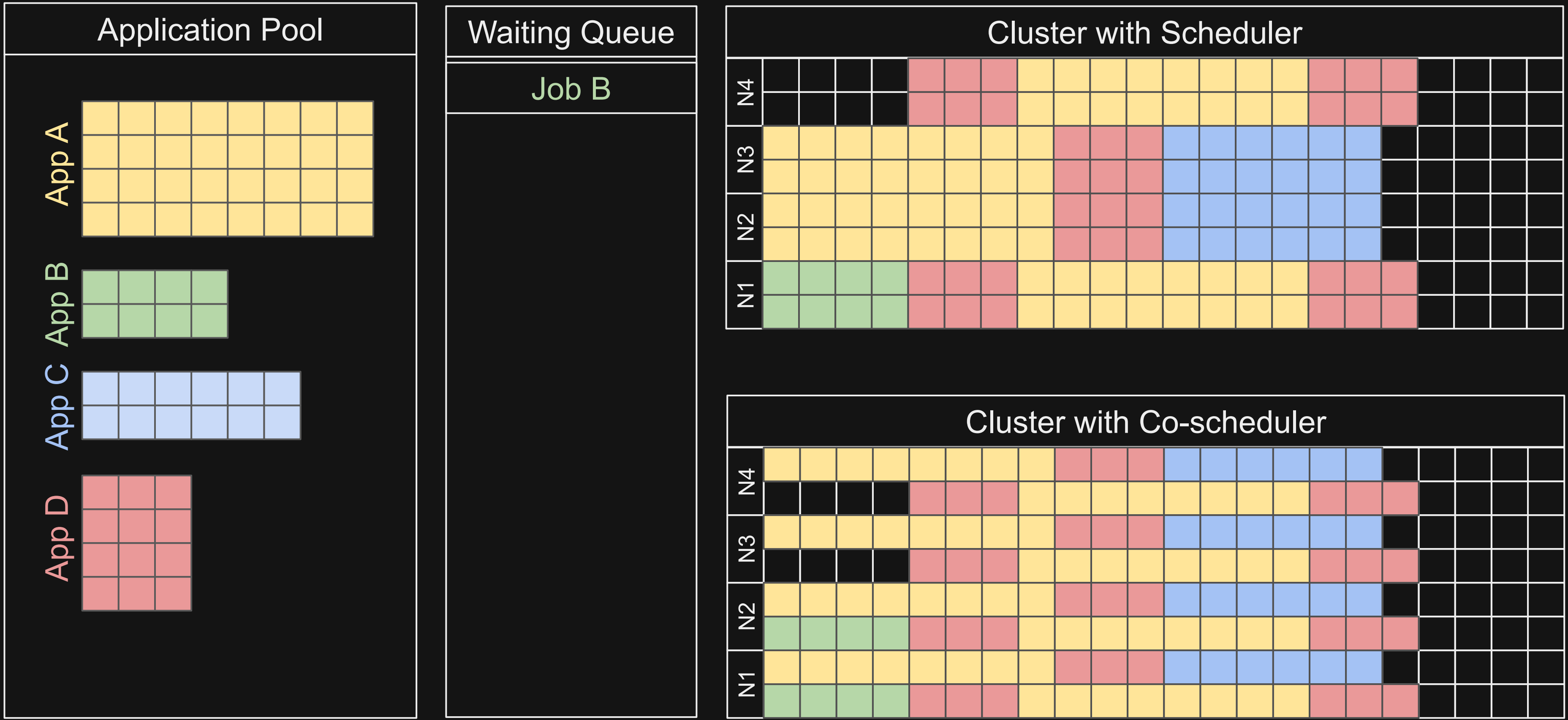
What is co-scheduling?

Co-location in the time domain - FCFS example



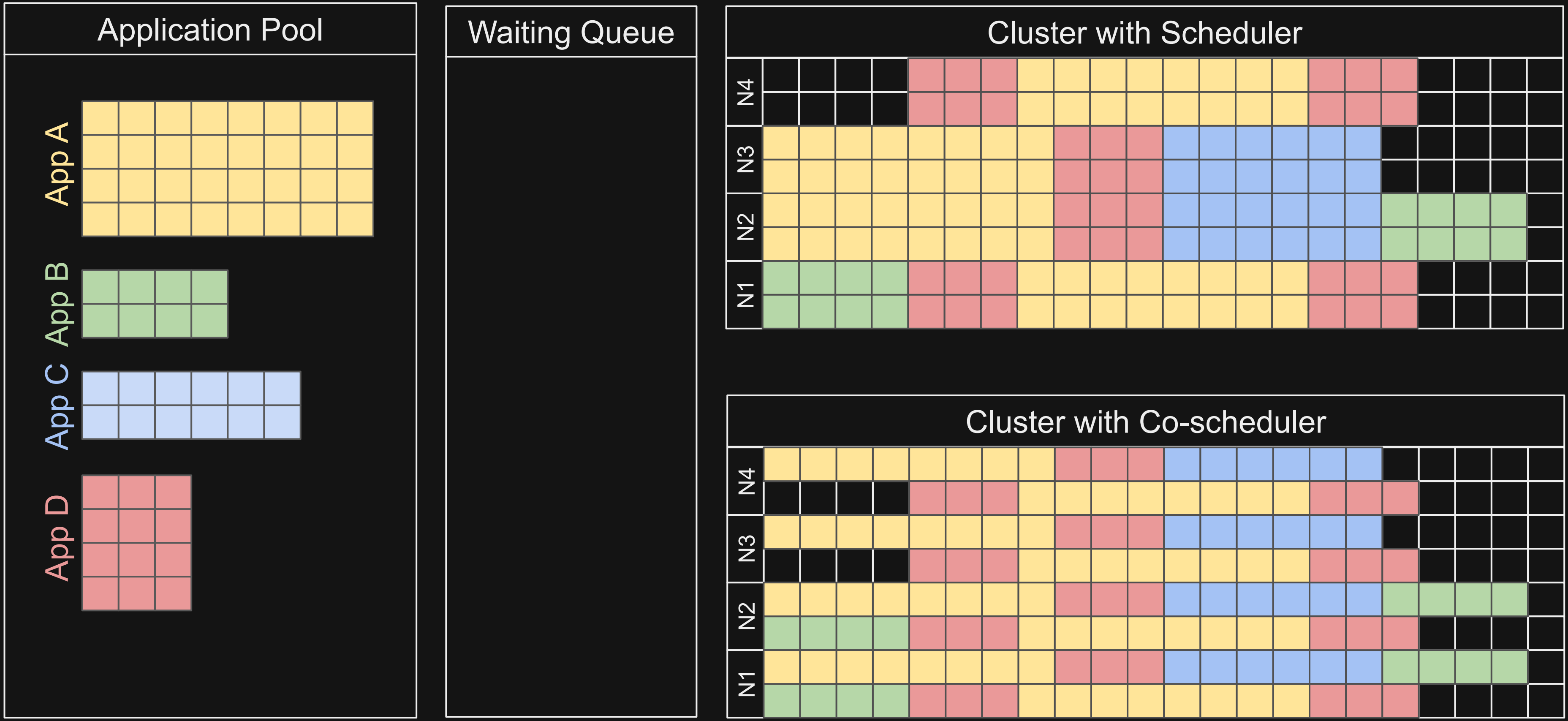
What is co-scheduling?

Co-location in the time domain - FCFS example



What is co-scheduling?

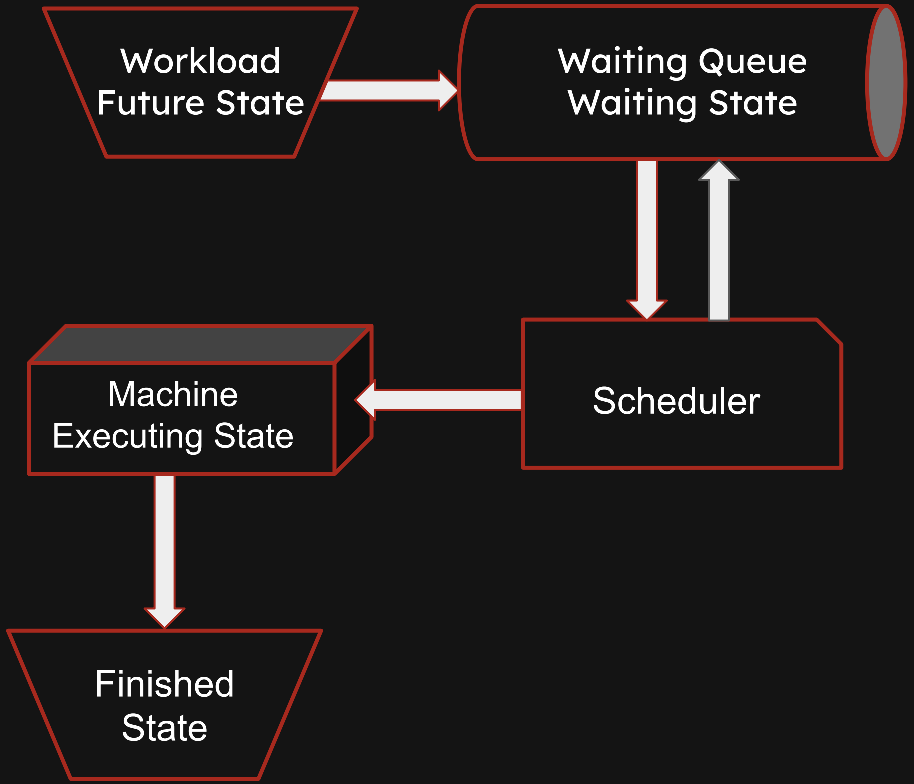
Co-location in the time domain - FCFS example



ELiSE: High-level logic

The simulation workflow

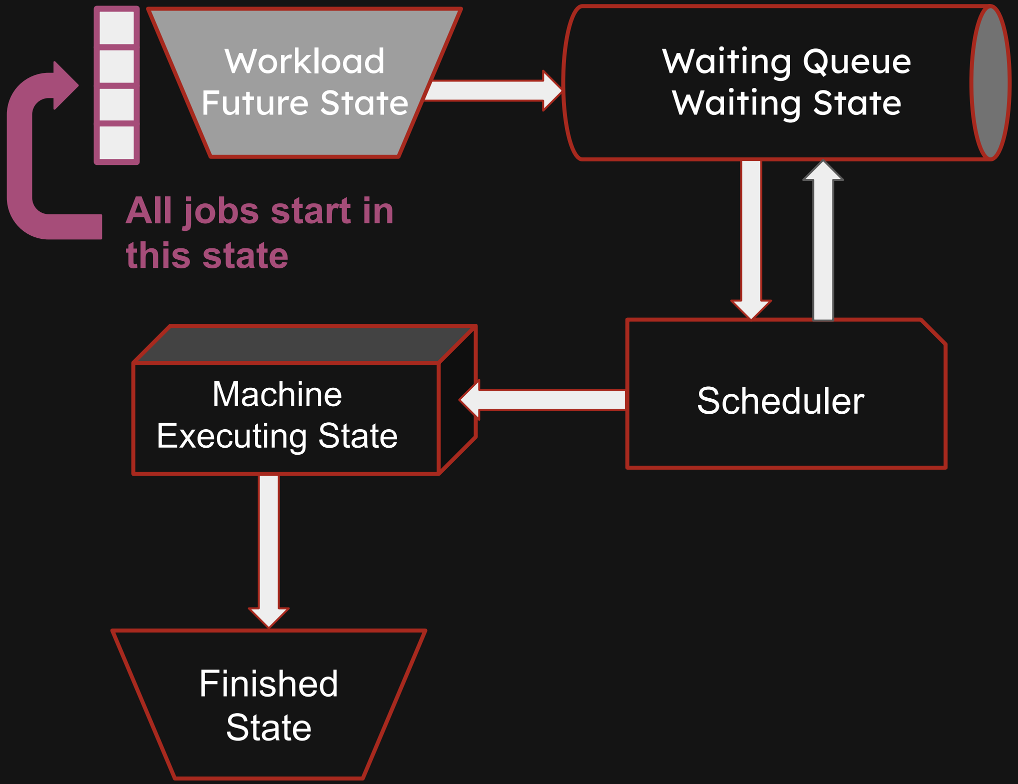
- ELiSE's main engine resembles a finite state machine, with jobs moving across four states



ELiSE: High-level logic

The simulation workflow

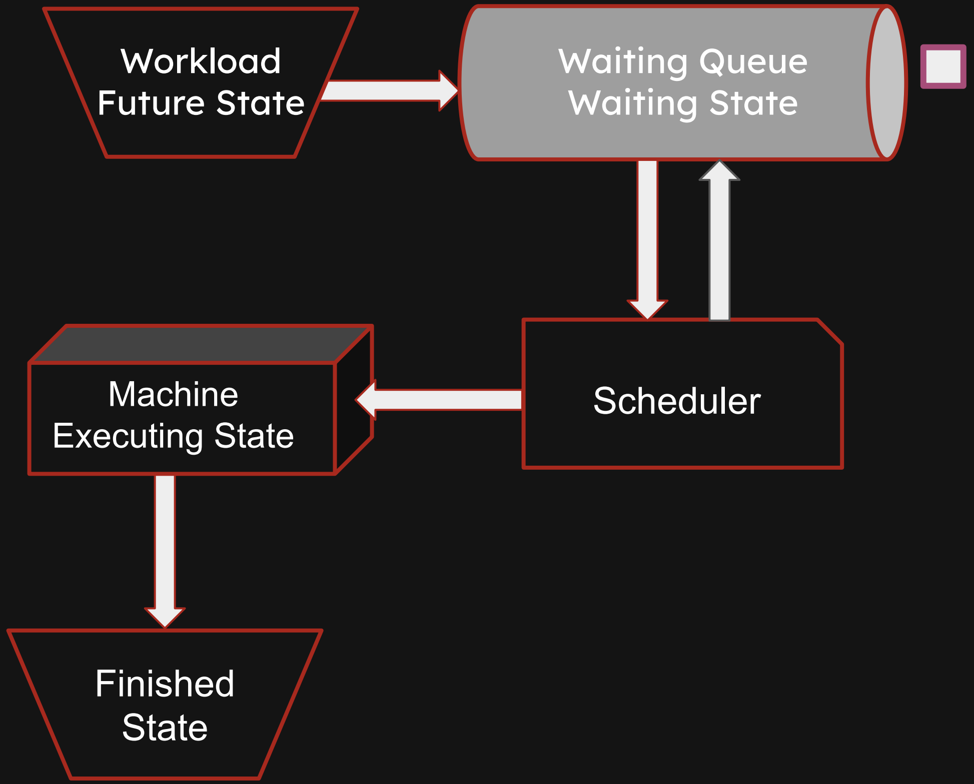
- ELiSE's main engine resembles a finite state machine, with jobs moving across four states
- Jobs in the Future State have been created, but the engine has not yet reached their scheduled arrival time



ELiSE: High-level logic

The simulation workflow

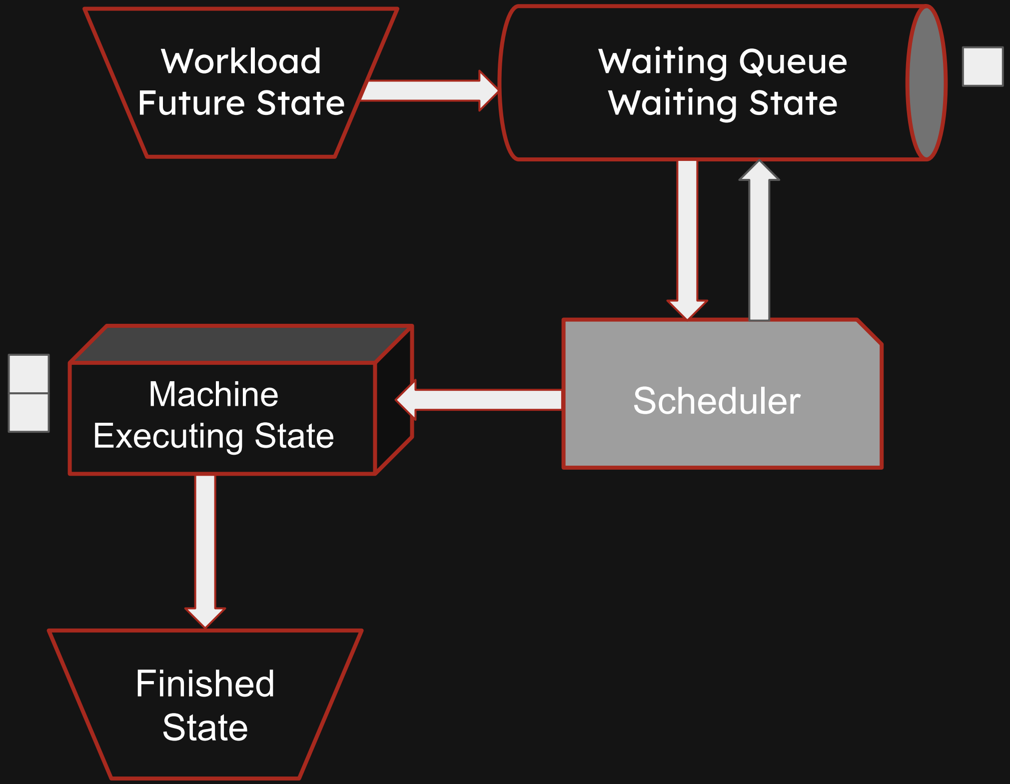
- ELiSE's main engine resembles a finite state machine, with jobs moving across four states
- Jobs in the Future State have been created, but the engine has not yet reached their scheduled arrival time
- Jobs in the Waiting State have arrived but aren't executed due to insufficient system resources



ELiSE: High-level logic

The simulation workflow

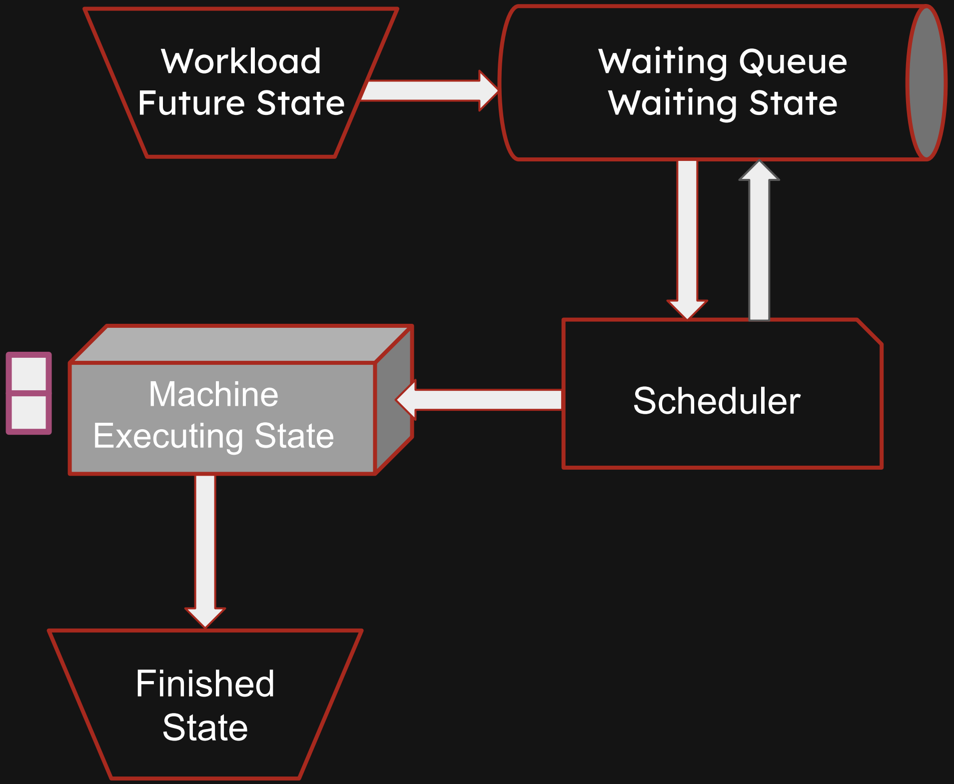
- The Scheduler determines which job to advance from the Waiting Queue to the Executing State based on its resource requests



ELiSE: High-level logic

The simulation workflow

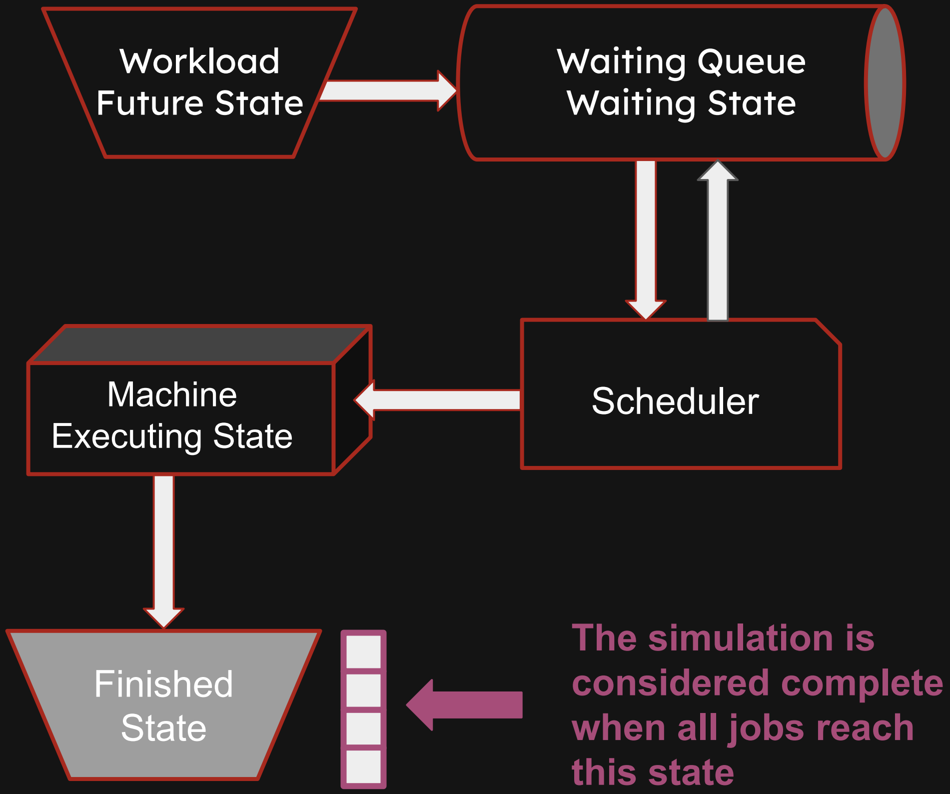
- The Scheduler determines which job to advance from the Waiting Queue to the Executing State based on its resource requests
- Jobs in the Executing State are those which are currently running in the system



ELiSE: High-level logic

The simulation workflow

- The Scheduler determines which job to advance from the Waiting Queue to the Executing State based on its resource requests
- Jobs in the Executing State are those which are currently running in the system
- Jobs in the Finished State are those that have finished their execution



ELiSE: Life-cycle of a simulation run

In detail

- Simulation run = multiple discrete steps = multiple events

ELiSE: Life-cycle of a simulation run

In detail

- Simulation run = multiple discrete steps = multiple events
- At each step, job state transitions based on current state and scheduling policy

Simulation run timespan

ELiSE: Life-cycle of a simulation run

In detail

- Simulation run = multiple discrete steps = multiple events
- At each step, job state transitions based on current state and scheduling policy
- For each step, the next event is calculated by selecting the **minimum of**:
  - the arrival time of each future job
  - the remaining time of each executing job

Simulation run timespan

ELiSE: Life-cycle of a simulation run

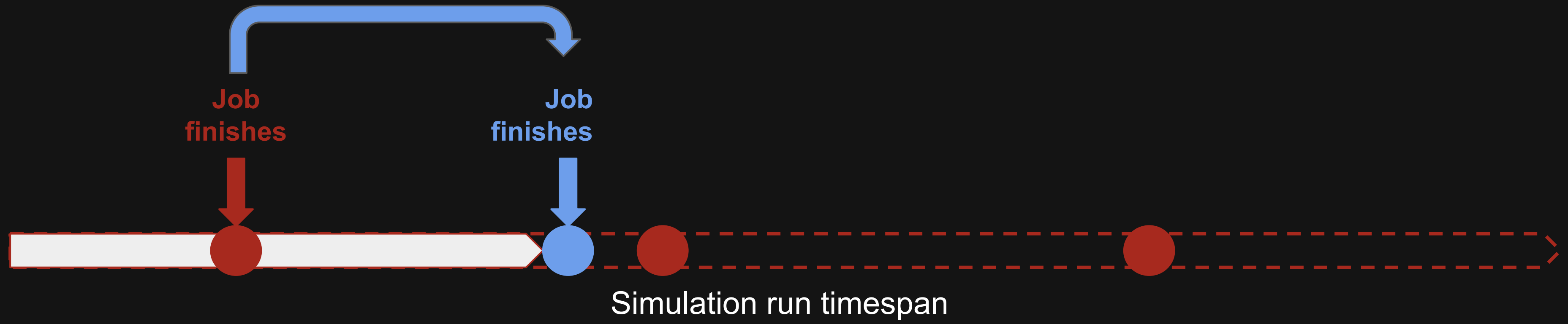
In detail

- Simulation run = multiple discrete steps = multiple events
- At each step, job state transitions based on current state and scheduling policy
- For each step, the next event is calculated by selecting the **minimum of**:
  - the arrival time of each future job
  - the remaining time of each executing job

ELiSE: Life-cycle of a simulation run

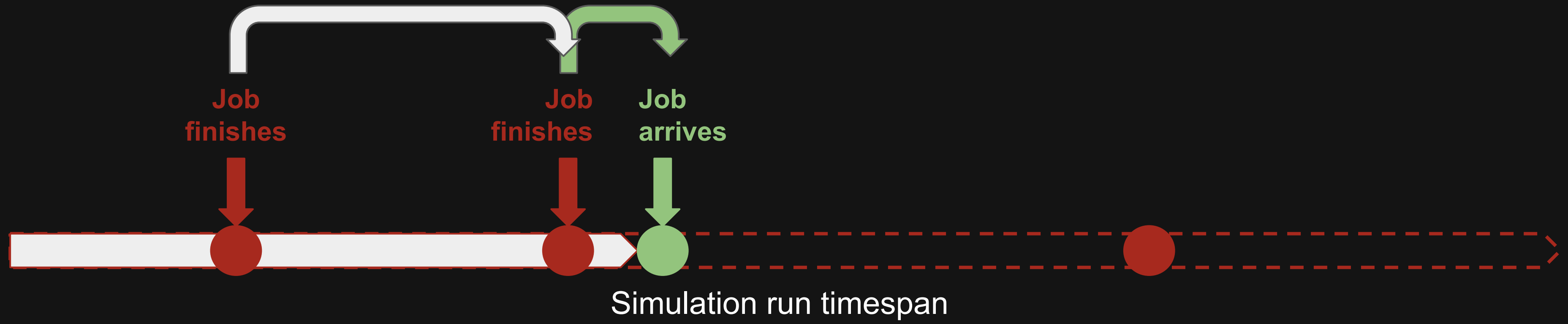
In detail

- Simulation run = multiple discrete steps = multiple events
- At each step, job state transitions based on current state and scheduling policy
- For each step, the next event is calculated by selecting the **minimum of**:
    - the arrival time of each future job
    - the remaining time of each executing job



ELiSE: Life-cycle of a simulation run
In detail

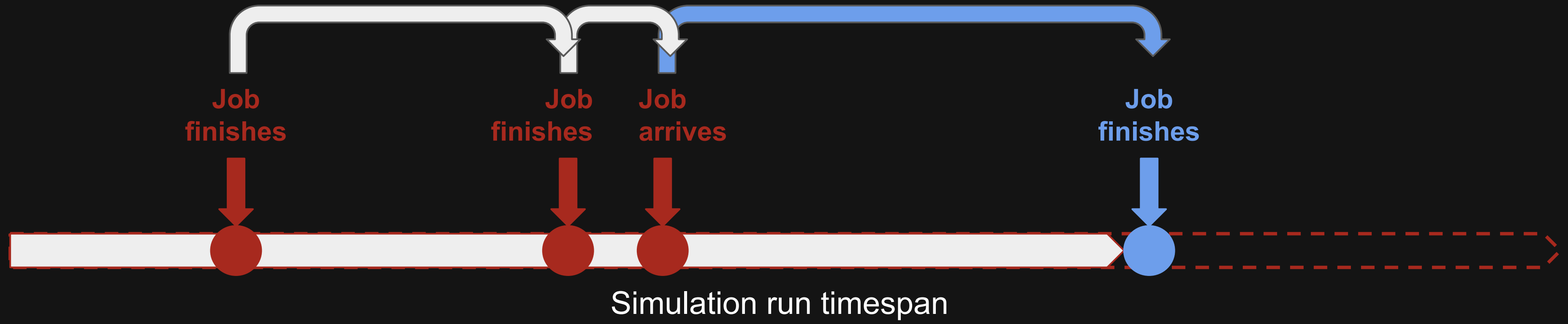
- Simulation run = multiple discrete steps = multiple events
- At each step, job state transitions based on current state and scheduling policy
- For each step, the next event is calculated by selecting the **minimum of**:
  - the arrival time of each future job
  - the remaining time of each executing job



ELiSE: Life-cycle of a simulation run

In detail

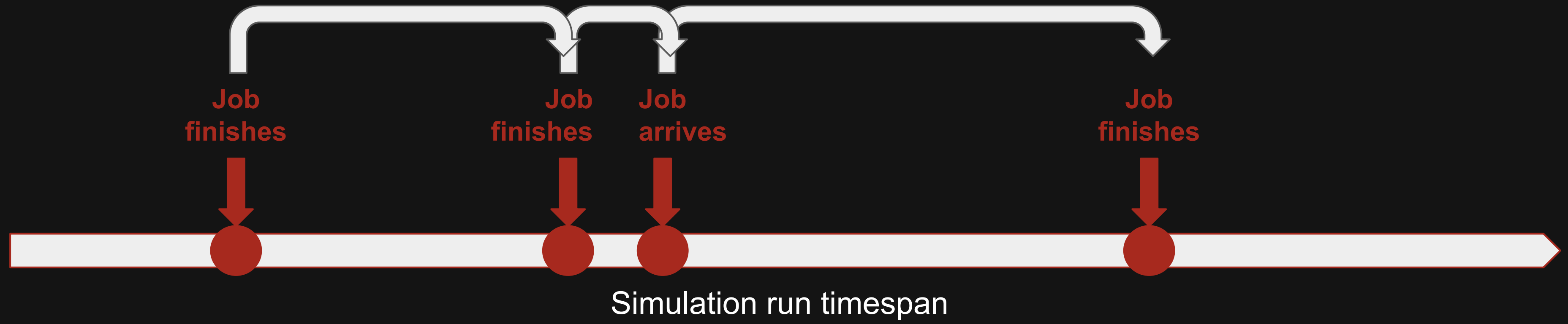
- Simulation run = multiple discrete steps = multiple events
- At each step, job state transitions based on current state and scheduling policy
- For each step, the next event is calculated by selecting the **minimum of**:
  - the arrival time of each future job
  - the remaining time of each executing job



ELiSE: Life-cycle of a simulation run

In detail

- Simulation run = multiple discrete steps = multiple events
- At each step, job state transitions based on current state and scheduling policy
- For each step, the next event is calculated by selecting the **minimum of**:
  - the arrival time of each future job
  - the remaining time of each executing job



ELiSE: Co-scheduling adaptation

In detail

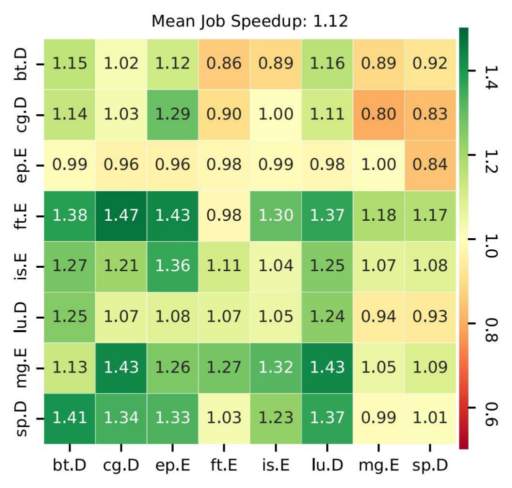
- We are using the heatmap of the workload to calculate the new remaining time of each co-scheduled job

- The formula which calculates the new remaining time is the following:

$$remExecTime = \frac{speedup_{old} \times remExecTime_{old}}{speedup_{new}}$$

- The new speedup of the job is calculated by taking the minimum speedup gain from the neighbors

$$speedup_{new} = \min_{\forall co-scheduled(job')} \{getSpeedupWith(job')\}$$



ELiSE: Co-scheduling adaptation

In detail

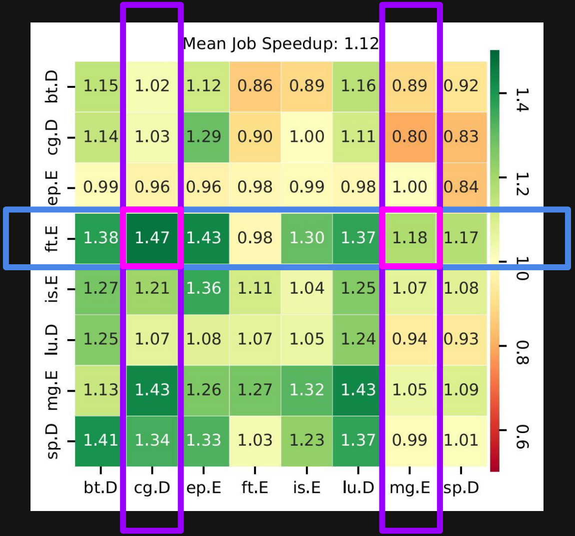
- We are using the heatmap of the workload to calculate the new remaining time of each co-scheduled job

- The formula which calculates the new remaining time is the following:

$$remExecTime = \frac{speedup_{old} \times remExecTime_{old}}{speedup_{new}}$$

- The new speedup of the job is calculated by taking the minimum speedup gain from the neighbors

$$speedup_{new} = \min_{\forall co-scheduled(job')} \{getSpeedupWith(job')\}$$

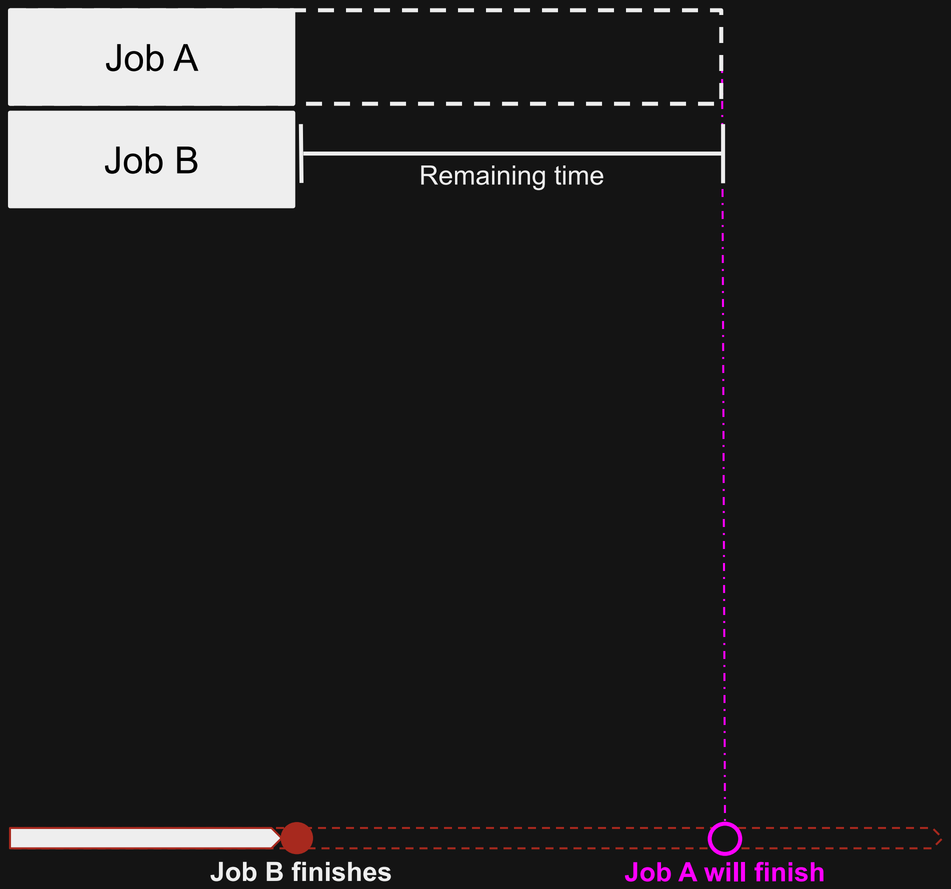


- For example if **ft.E** is co-scheduled with **cg.D** and **mg.E** then the new speedup will be 1.18

ELiSE: Co-scheduling adaptation

In detail

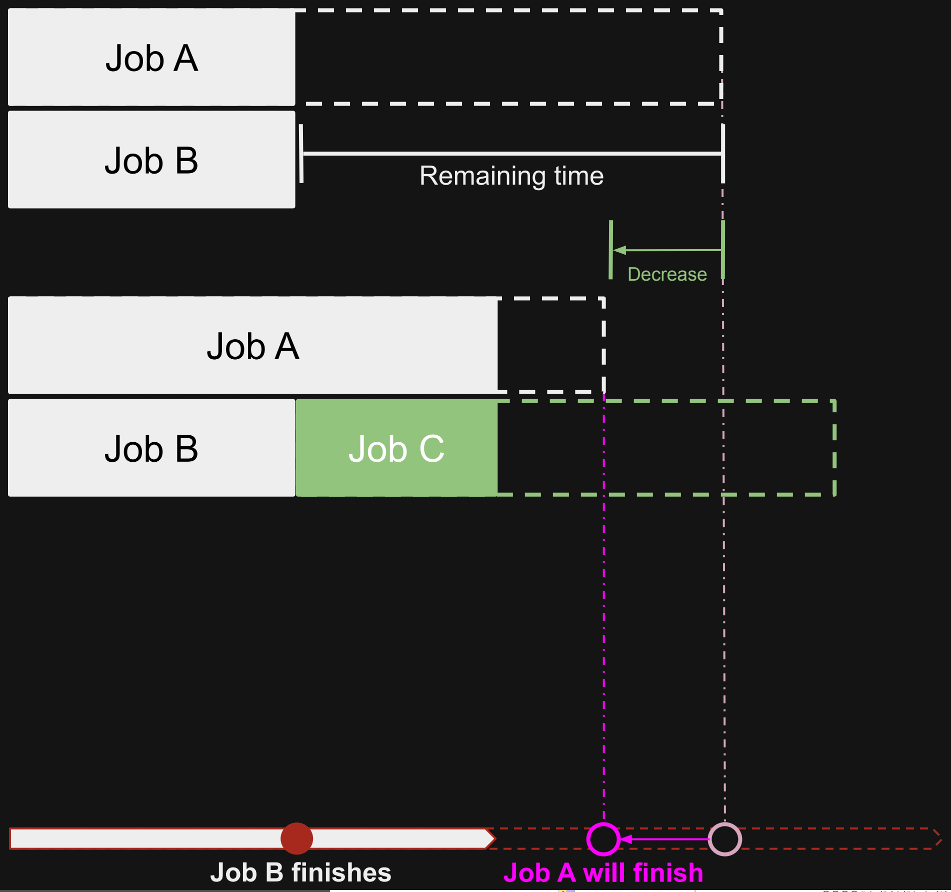
- At each step, we examine all running jobs to identify any that are co-scheduled. We adjust their remaining time according to the performance gains or losses from their neighbors



ELiSE: Co-scheduling adaptation

In detail

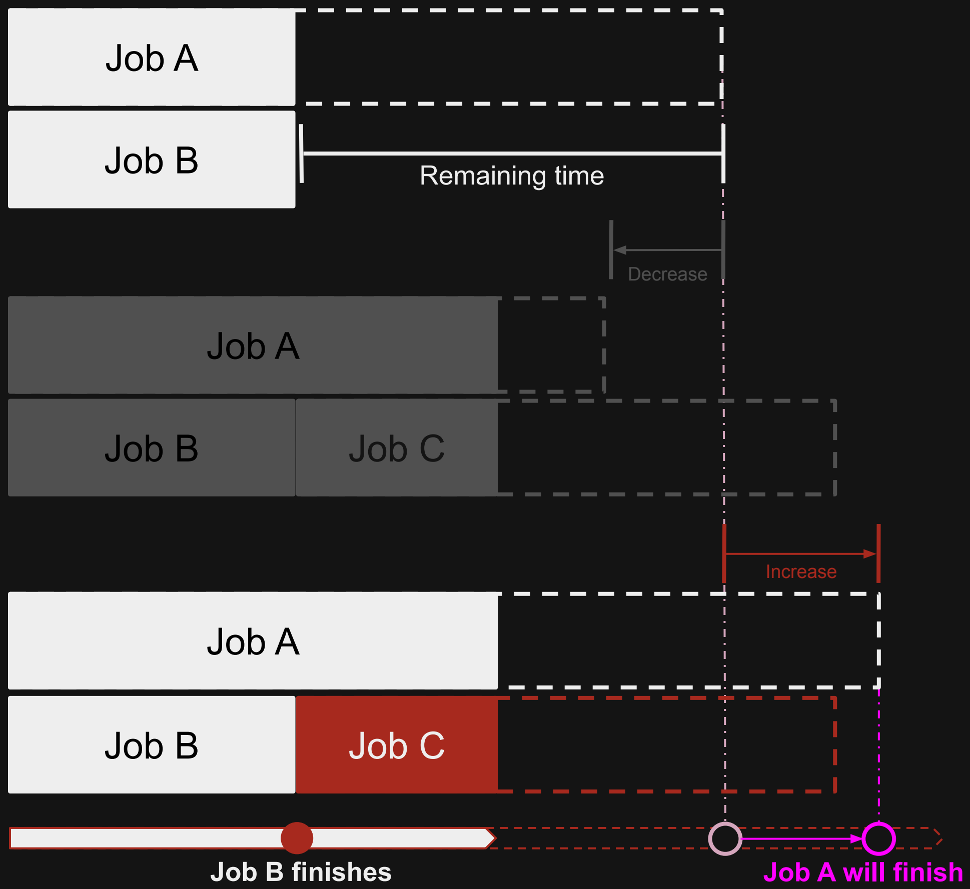
- At each step, we examine all running jobs to identify any that are co-scheduled. We adjust their remaining time according to the performance gains or losses from their neighbors
- If the co-scheduled job is a good neighbor then the remaining time of the job decreases



ELiSE: Co-scheduling adaptation

In detail

- At each step, we examine all running jobs to identify any that are co-scheduled. We adjust their remaining time according to the performance gains or losses from their neighbors
- If the co-scheduled job is a good neighbor then the remaining time of the job decreases
- If it is a bad neighbor then the remaining time increases



ELiSE: Develop a (co-)Scheduler

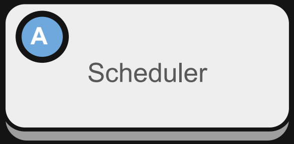
The Structure of the Schedulers

- In ELiSE all schedulers create a class hierarchy, making it easier to extend and combine algorithms into a new scheduler

ELiSE: Develop a (co-)Scheduler

The Structure of the Schedulers

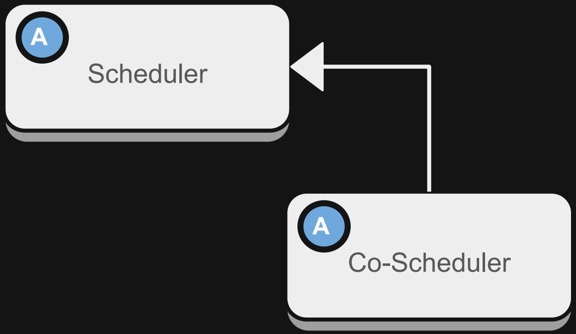
- In ELiSE all schedulers create a class hierarchy, making it easier to extend and combine algorithms into a new scheduler
- The root of the hierarchy is the Scheduler abstract class. It defines the specifications for the rest of the scheduling algorithms



ELiSE: Develop a (co-)Scheduler

The Structure of the Schedulers

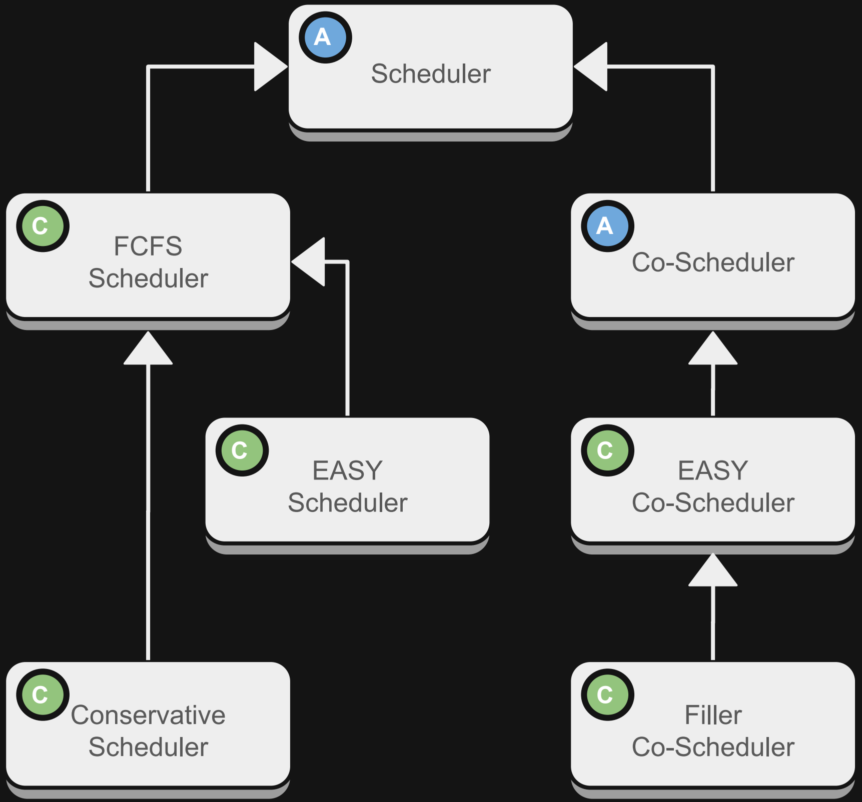
- In ELiSE all schedulers create a class hierarchy, making it easier to extend and combine algorithms into a new scheduler
- The root of the hierarchy is the Scheduler abstract class. It defines the specifications for the rest of the scheduling algorithms
- The Co-Scheduler abstract class extends Scheduler to include co-scheduling schemes



ELiSE: Develop a (co-)Scheduler

The Structure of the Schedulers

- In ELiSE all schedulers create a class hierarchy, making it easier to extend and combine algorithms into a new scheduler
- The root of the hierarchy is the Scheduler abstract class. It defines the specifications for the rest of the scheduling algorithms
- The Co-Scheduler abstract class extends Scheduler to include co-scheduling schemes
- All well known algorithms like FCFS, SJF, EASY Backfill, Conservative Backfill, etc. can be implemented



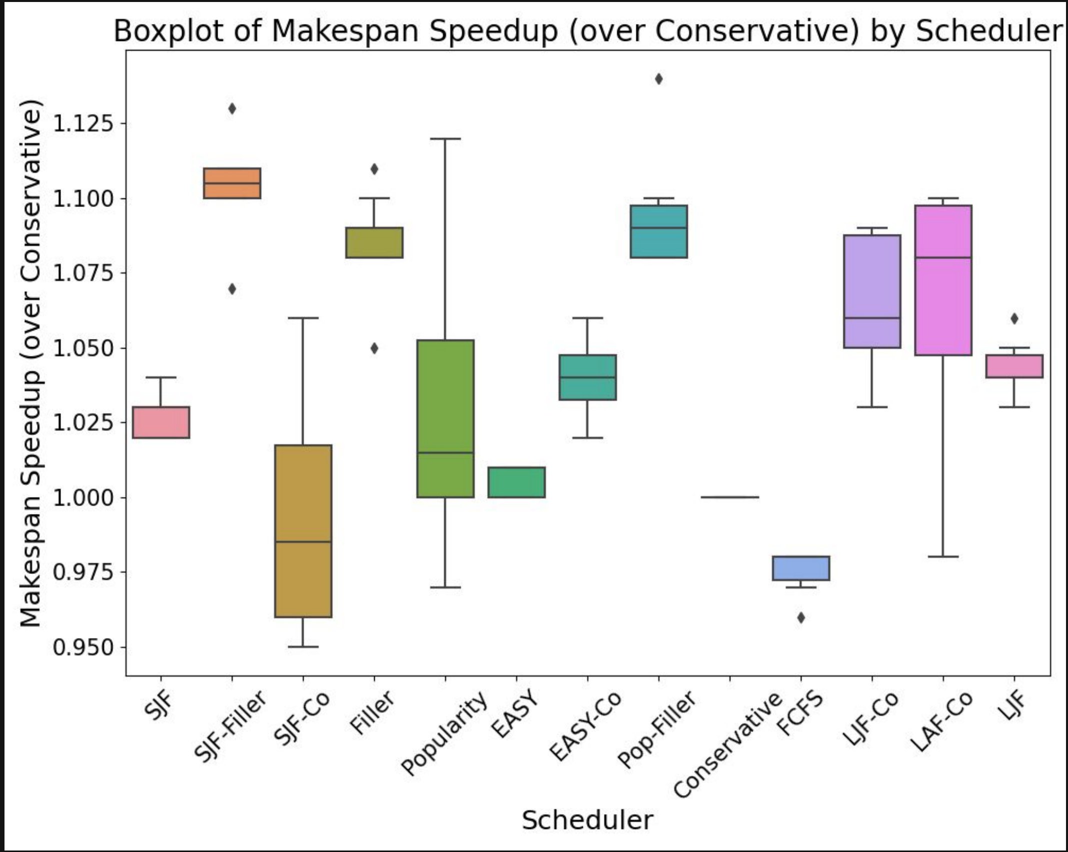
ELiSE: Develop a (co-)Scheduler

Examples of implemented (co-)Schedulers

Scheduler	Description
Conservative FCFS	Implements the SLURM's standard scheduling algorithm
EASY Co-scheduler	Deploys jobs in arrival order, co-allocating on half nodes, with EASY backfilling
Shortest / Longest / Largest Job First Co-scheduler	Prioritizes shortest / longest / largest jobs in the waiting queue
Popularity Co-scheduler	Prioritizes co-execution-friendly jobs
Filler Co-scheduler	Sacrifices user fairness by re-ordering the waiting queue in order to increase system utilization
Two Factors (SJF/Pop-Filler) Co-scheduler	Combination of Filler and SJF/Popularity co-schedulers

ELiSE: Develop a (co-)Scheduler

A (co-)Schedulers study based on the Makespan Speedup metric



Tests

- **Validation:** test ELiSE by comparing small scale results with a real system
- **Co-scheduling Results:** assess first co-scheduling outcomes
- **Performance:** compare real processing time against simulation time across workloads and clusters

ELiSE: Evaluation
Overview

Tests
- **Validation:** test ELiSE by comparing small scale results with a real system
- **Co-scheduling Results:** assess first co-scheduling outcomes
- **Performance:** compare real processing time against simulation time across workloads and clusters

Benchmarks
- NPB suite, D and E classes
- 64, 128, 512, 1024 process count
- Heatmap of Speedups from real systems

Tests

- **Validation:** test ELiSE by comparing small scale results with a real system
- **Co-scheduling Results:** assess first co-scheduling outcomes
- **Performance:** compare real processing time against simulation time across workloads and clusters

Benchmarks

- NPB suite, D and E classes
- 64, 128, 512, 1024 process count
- Heatmap of Speedups from real systems

Metrics

- **Makespan:** total time required to complete a set of jobs
- **Job turnaround time:** total time taken for a job from submission to completion
- **System utilization:** the percentage of the system that is in use until the waiting queue is empty
- **Simulated and Real Time Ratio:** simulated makespan / real time in simulated days per real hour

Experimental Setup

ARIS

CPU Type	Intel Xeon E5-2680v2 (Ivy Bridge), 2.8 GHz
CPUs per Node	2
Cores per CPU	10
Cores per Node	20
Hyperthreading	OFF
Memory per Node	64 GB
Network	Infiniband FDR, 56 Gb/s

Marconi

CPU Type	Intel Xeon 8160 (SkyLake), 2.10 GHz
CPUs per Node	2
Cores per CPU	24
Cores per Node	48
Hyperthreading	OFF
Memory per Node	196 GB
Network	Intel Omni-Path, 100 Gb/s

Grid5000 (Grvingt)

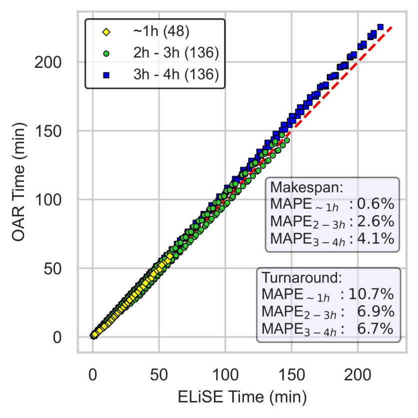
CPU Type	Intel Xeon Gold 6130, 2.10 GHz
CPUs per Node	2
Cores per CPU	16
Cores per Node	32
Hyperthreading	OFF
Memory per Node	192 GB
Network	Intel Omni-Path, 100 Gb/s

Used for validation

ELiSE: Evaluation

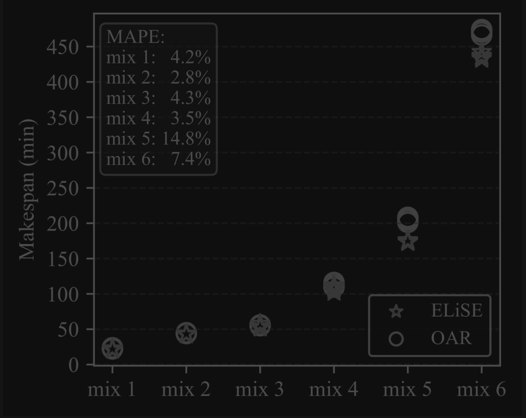
Validation

FCFS Validation



- 6 experiments: 2×(48 jobs, 1h), 2×(136 jobs, 2–3h), 2×(136 jobs, 3–4h)
- Point: turnaround time of one job
- Real vs. simulated times: low gap
- Differences due to:
  - slight per-job execution time variation
  - system overhead
  - scheduler overhead

Co-scheduling Validation

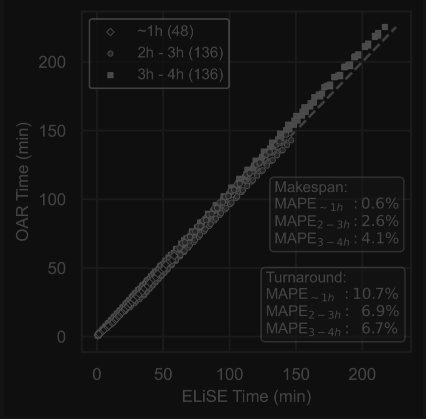


- Extended OAR to support co-location
- 30 experiments: 6 application mixes with increasing Mean Job Speedup; 5 job shuffles per mix
- Results vs. reality: close match
- Mean Absolute Percentage Error (MAPE) ≤15%

ELiSE: Evaluation

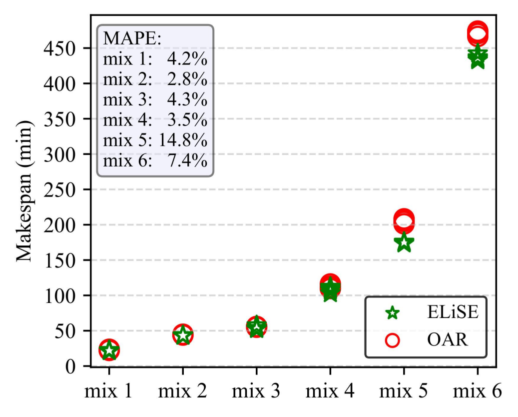
Validation

FCFS Validation



- 6 experiments: 2×(48 jobs, 1h), 2×(136 jobs, 2–3h), 2×(136 jobs, 3–4h)
- Point: turnaround time of one job
- Real vs. simulated times: low gap
- Differences due to:
  - slight per-job execution time variation
  - system overhead
  - scheduler overhead

Co-scheduling Validation



- Extended OAR to support co-location
- 30 experiments: 6 application mixes with increasing Mean Job Speedup; 5 job shuffles per mix
- Results vs. reality: close match
- Mean Absolute Percentage Error (MAPE) ≤15%

ELiSE: Evaluation

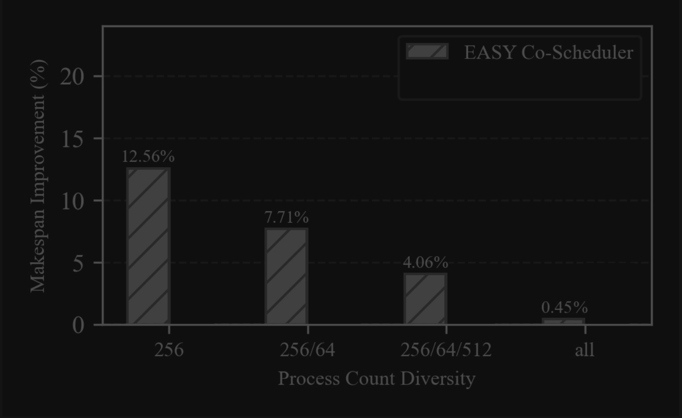
Co-scheduling Results

MJS & MS correlation

Mean Job Speedup	Makespan Improvement
1.07	24.42 %
1.10	25.29 %
1.12	25.87 %
1.15	30.63 %

- 4 workloads, 5 shuffles each; cluster: 100 nodes, 48 cores
- Each workload: 500 jobs from Marconi system applications; selection based on progressively increasing Mean Job Speedup
- Correlation between MJS and MS
- High Makespan improvement over EASY
- Process count: 256 across 4 workloads

Process Count Diversity



- 4 workloads, 500 jobs each from ARIS system; with increasing process count diversity
- Greater process count diversity → reduced benefits from simple co-scheduling
- Cause: resource fragmentation; effect: low System Utilization (from ELiSE diagrams)

ELiSE: Evaluation

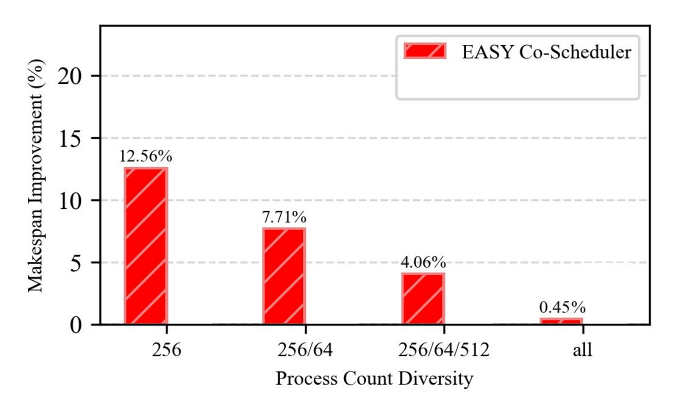
Co-scheduling Results

MJS & MS correlation

Mean Job Speedup	Makespan Improvement
1.07	24.42 %
1.10	25.29 %
1.12	25.87 %
1.15	30.63 %

- 4 workloads, 5 shuffles each; cluster: 100 nodes, 48 cores
- Each workload: 500 jobs from Marconi system applications; selection based on progressively increasing Mean Job Speedup
- Correlation between MJS and MS
- High Makespan improvement over EASY
- Process count: 256 across 4 workloads

Process Count Diversity



- 4 workloads, 500 jobs each from ARIS system; with increasing process count diversity
- Greater process count diversity → reduced benefits from simple co-scheduling
- Cause: resource fragmentation; effect: low System Utilization (from ELiSE diagrams)

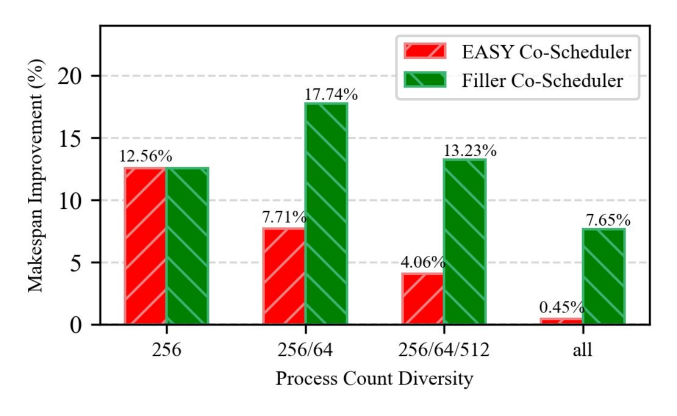
ELiSE: Evaluation
Co-scheduling Results

MJS & MS correlation

Mean Job Speedup	Makespan Improvement
1.07	24.42 %
1.10	25.29 %
1.12	25.87 %
1.15	30.63 %

- 4 workloads, 5 shuffles each; cluster: 100 nodes, 48 cores
- Each workload: 500 jobs from Marconi system applications; selection based on progressively increasing Mean Job Speedup
- Correlation between MJS and MS
- High Makespan improvement over EASY
- Process count: 256 across 4 workloads

Process Count Diversity



- Implemented Filler Co-scheduler; crude filling of unutilized resources with best-fit waiting jobs
- Co-scheduling benefits preserved **but** user fairness compromised — FCFS principle violations in waiting queue

ELiSE: Evaluation

Performance

Synthetic Workloads

FCFS Scheduler

Workload Size	Small Cluster 256 cores	Middle Cluster 25600 cores	Large Cluster 2560000 cores
100	2038.66	147.75	226.52
1000	838.74	14.31	12.56
10000	45.84	0.58	0.83

EASY Scheduler

Workload Size	Small Cluster 256 cores	Middle Cluster 25600 cores	Large Cluster 2560000 cores
100	2038.66	147.75	226.52
1000	838.74	14.31	12.56
10000	45.84	0.58	0.83

- Different cluster and workload sizes; 50 repetitions per experiment
- **Workload size:** 10-fold increase → 10-fold increase in simulation time
- **Cluster size:** 100-fold increase → 10-fold increase in simulation time

Real Workloads

Workload	FCFS Scheduler	EASY Scheduler
SDSC-SP2-1998 128 nodes, 64 ppn, 73496 jobs	116.87	117.77
KIT-FH2-2016 1152 nodes, 20 ppn, 114335 jobs	28.32	27.19

- 2 real workloads in ELiSE from Parallel Workload Archive
- **First workload:** 7 hours for 2-year trace for both FCFS and EASY schedulers (parallel execution)
- **Second workload:** maximum 20 hours for 1.5-year trace for both schedulers
- Fast simulation for small/medium cases, tolerable times for very large cases

ELiSE: Evaluation

Performance

Synthetic Workloads

FCFS Scheduler

Workload Size	Small Cluster 256 cores	Middle Cluster 25600 cores	Large Cluster 2560000 cores
100	2038.66	147.75	226.52
1000	838.74	14.31	12.56
10000	45.84	0.58	0.83

EASY Scheduler

Workload Size	Small Cluster 256 cores	Middle Cluster 25600 cores	Large Cluster 2560000 cores
100	2038.66	147.75	226.52
1000	838.74	14.31	12.56
10000	45.84	0.58	0.83

- Different cluster and workload sizes; 50 repetitions per experiment
- **Workload size:** 10-fold increase → 10-fold increase in simulation time
- **Cluster size:** 100-fold increase → 10-fold increase in simulation time

Real Workloads

Workload	FCFS Scheduler	EASY Scheduler
SDSC-SP2-1998 128 nodes, 64 ppn, 73496 jobs	116.87	117.77
KIT-FH2-2016 1152 nodes, 20 ppn, 114335 jobs	28.32	27.19

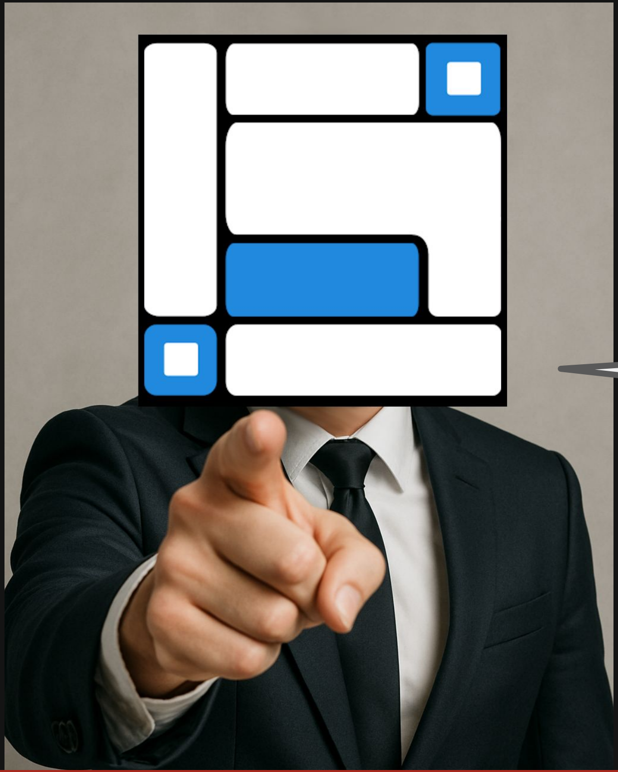
- 2 real workloads in ELiSE from Parallel Workload Archive
- **First workload:** 7 hours for 2-year trace for both FCFS and EASY schedulers (parallel execution)
- **Second workload:** maximum 20 hours for 1.5-year trace for both schedulers
- Fast simulation for small/medium cases, tolerable times for very large cases

ELiSE: Future Work

Focus on three main axes:

- **Modelling capabilities:**
  - Co-scheduling model extensions
  - Power consumption models, I/O bound applications, fault tolerance, etc
- **User experience:**
  - Additional visualization tools
  - Improved interfaces and API
  - Automated (co-)scheduling parameters optimization
- **Performance improvement:**
  - Faster large scale HPC simulation runs

ELiSE: More info



ELiSE: More info

Join us on GitHub (https://github.com/cslab-ntua/elise)

- Contribute with your (co-)scheduler implementation
- Contribute with your HPC cluster benchmark
- Report bugs
- Suggest features

Join us on DISCORD (https://discord.gg/cABwcWhBSx)

- Join our community!
- Get notifications on the latest commits, bug fixes and releases
- Get updates on the next features and events
- Q&A

THANK YOU FOR LISTENING