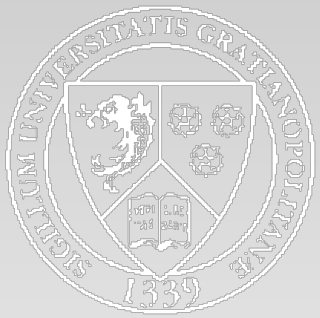
Équilibrage de charge dynamique sur plates-formes hiérarchiques

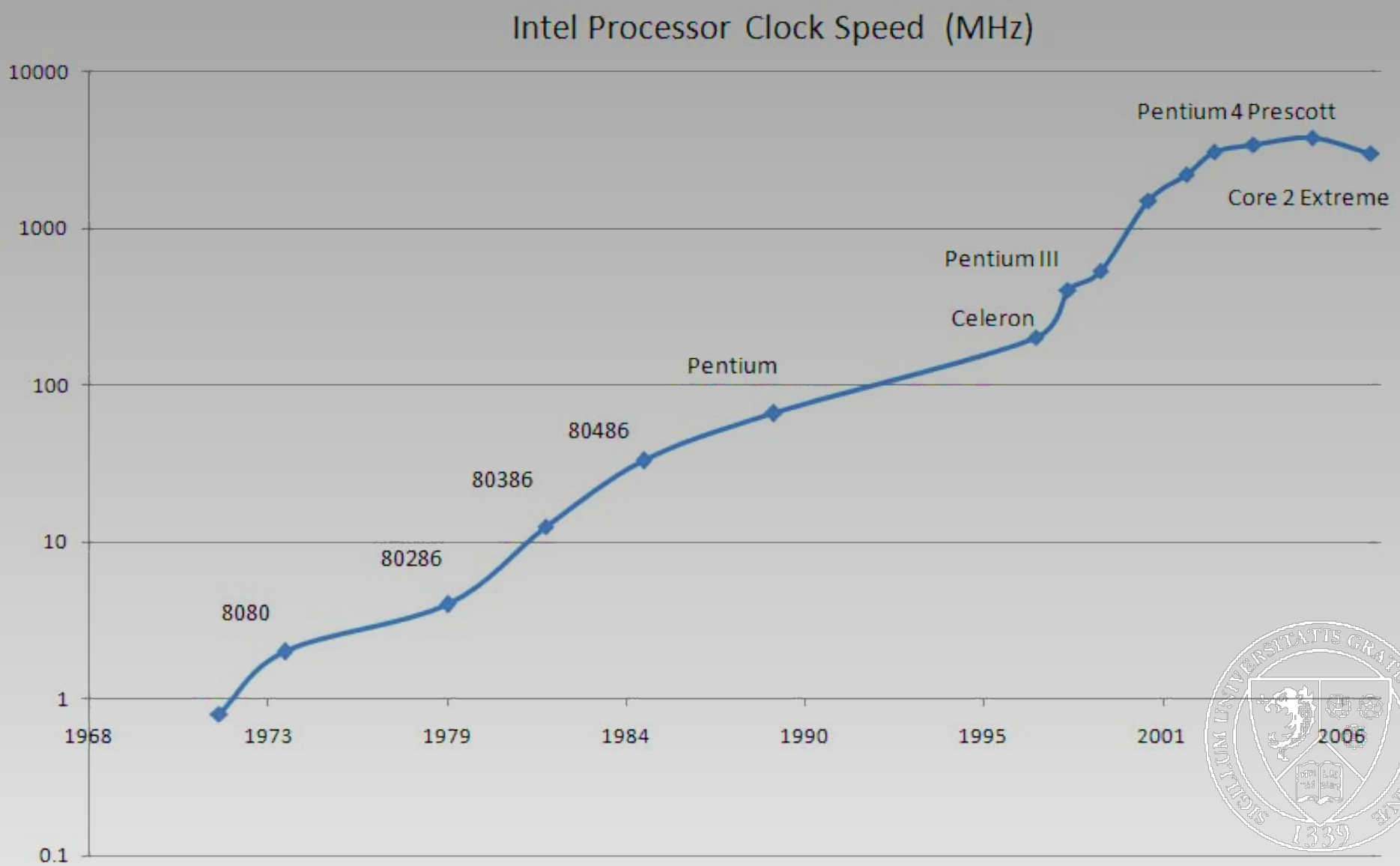
Quintin Jean-Noël

sous la direction de Denis Trystram et Frédéric Wagner
MOAIS, INRIA/LIG, Université de Grenoble

8 Decembre 2011

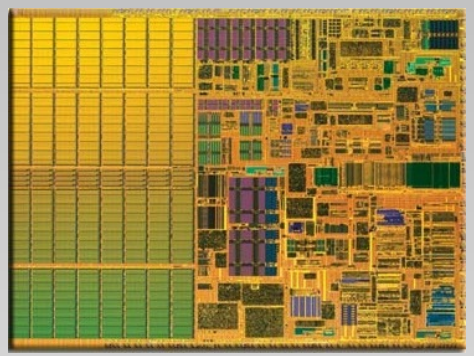


Clock Frequency

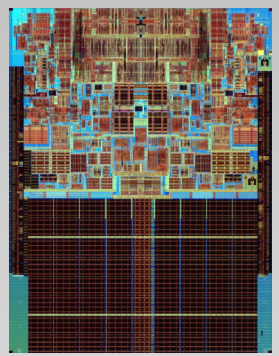


Processor Evolution

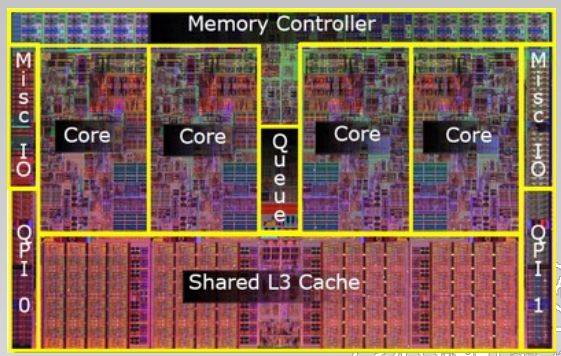
Pentium M
Mono-core 2003



Core 2 duo
Dual-core
2005

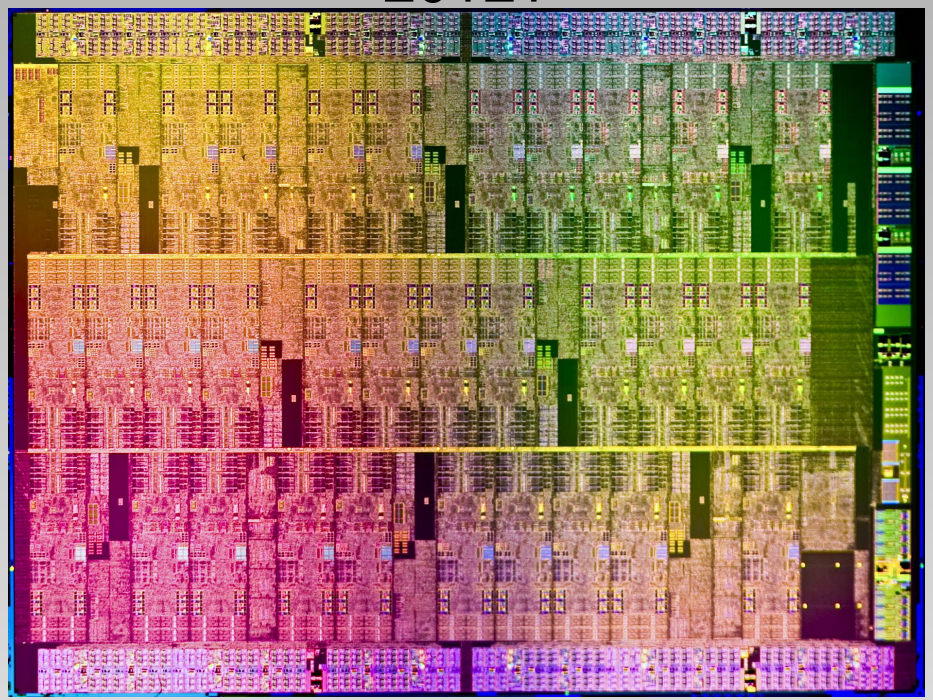


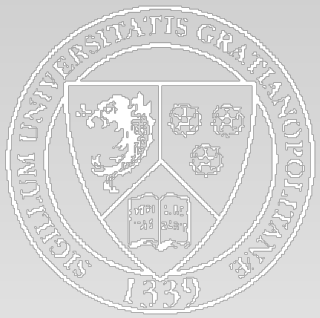
Core i7
Quad-core
2007



Next Generation: Network On Chip

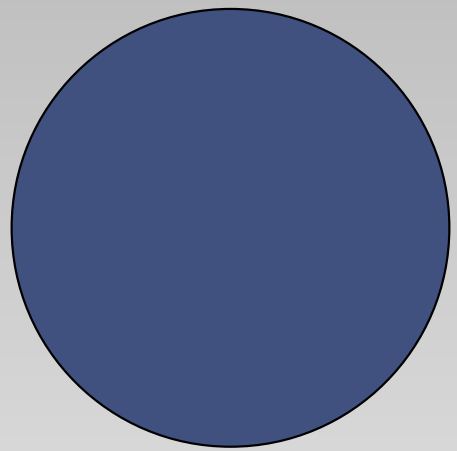
Knight Corner
50 cores
2012?

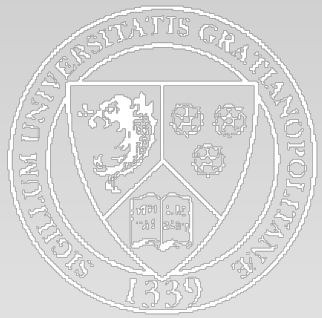




Parallel Programming

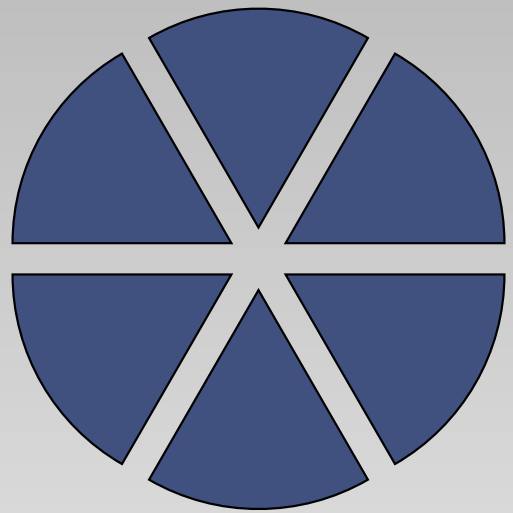
- Parallelization of an application:

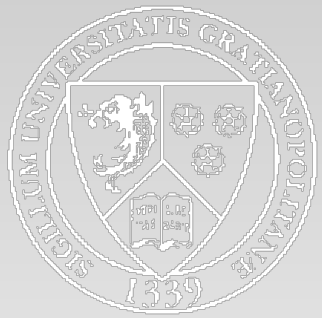




Parallel Programming

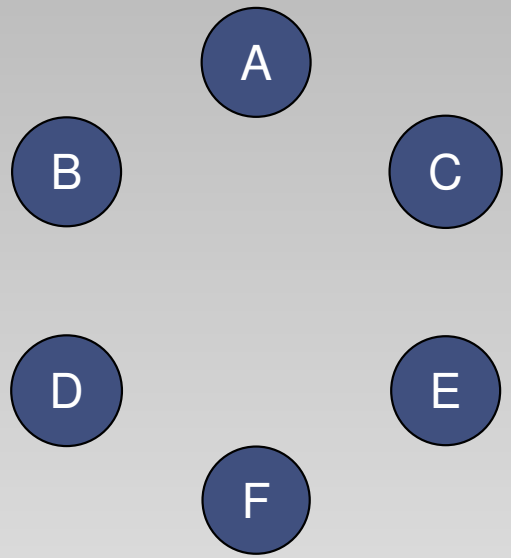
- Parallelization of an application:
  - Identify independent parts.

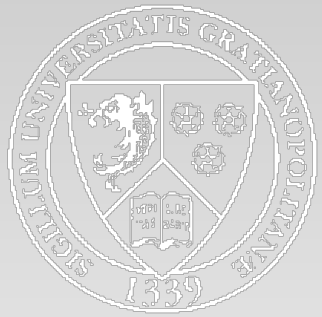




Parallel Programming

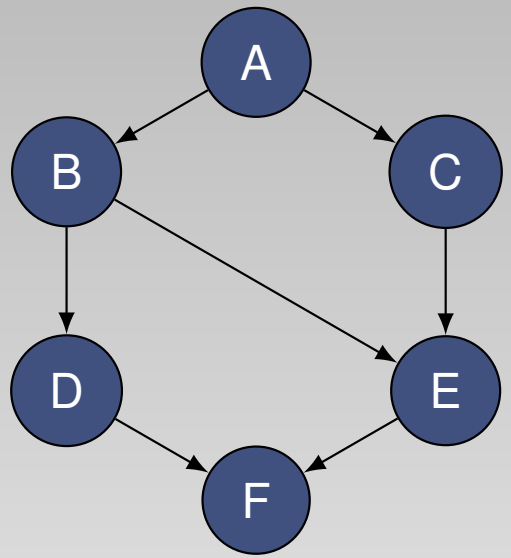
- Parallelization of an application:
  - Identify independent parts.

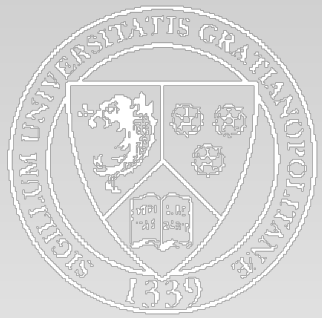




Parallel Programming

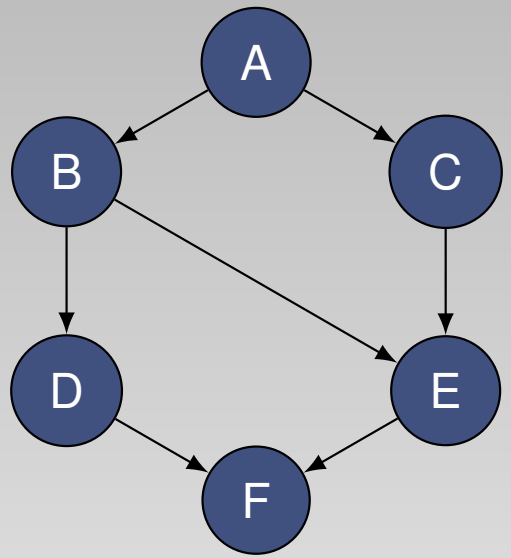
- Parallelization of an application:
  - Identify independent parts.
  - Describe tasks dependencies.

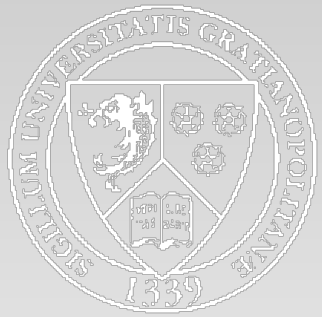




Parallel Programming

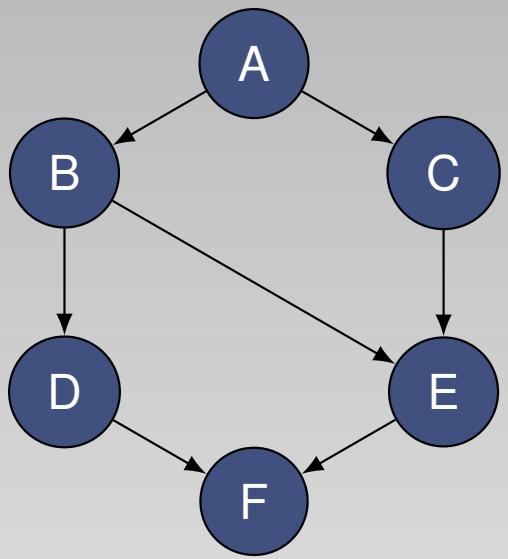
- Parallelization of an application:
  - Identify independent parts.
  - Describe tasks dependencies.
  - Schedule tasks on processors.

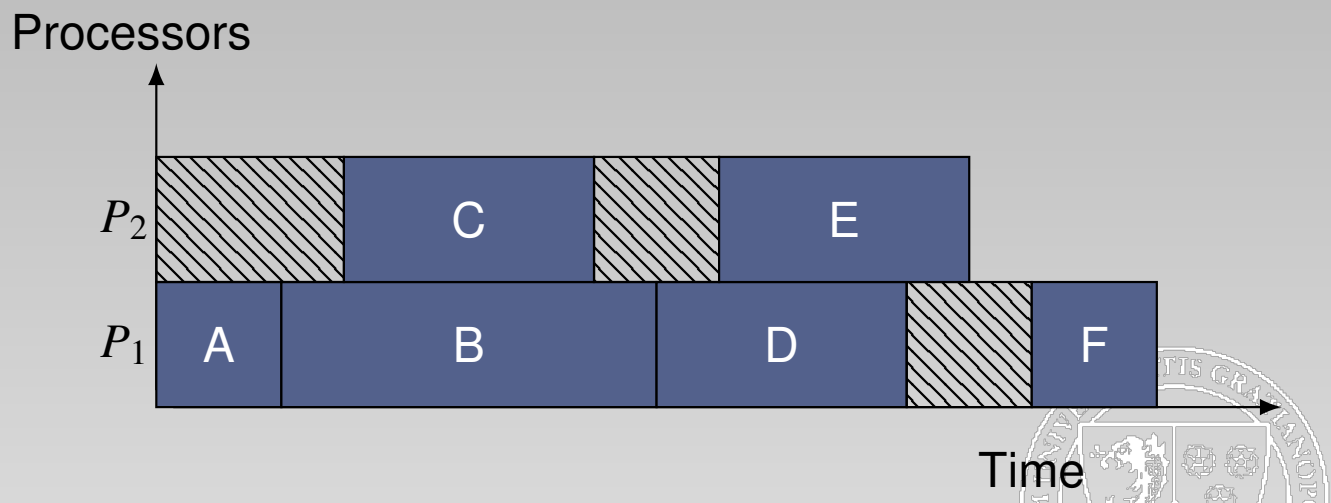




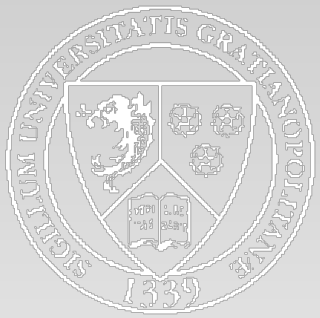
Scheduling

- Tasks scheduling:
  - Assigned a processor to each task.
  - Determined the starting time of tasks.



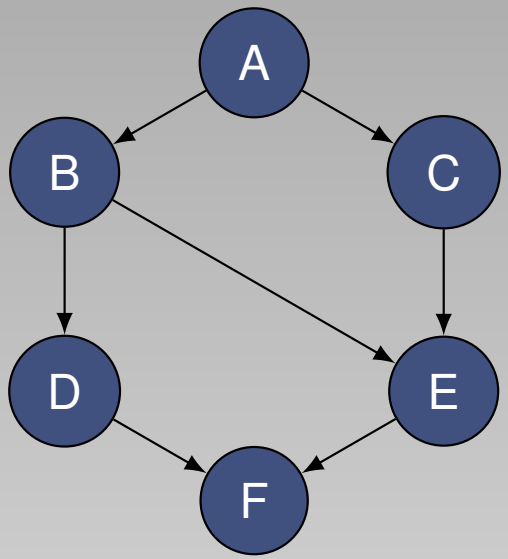


Outline

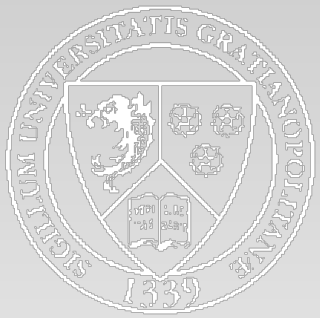


Online Scheduling

- List-scheduling [Graham-69].

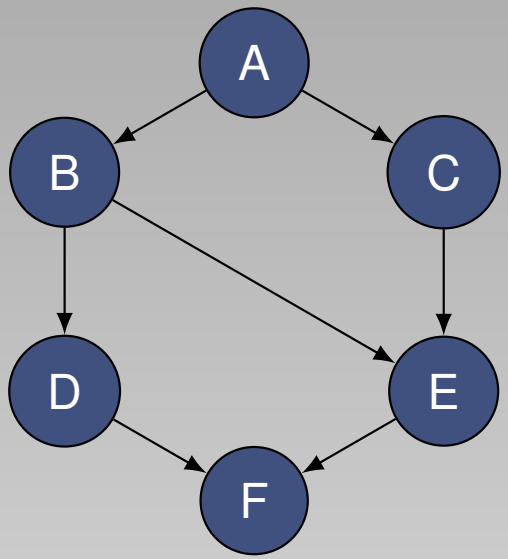


Ready tasks list

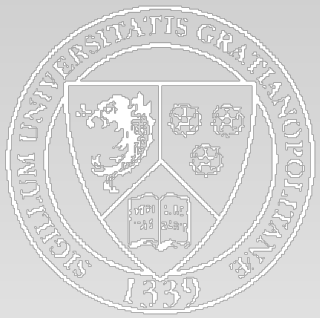


Online Scheduling

- List-scheduling [Graham-69].

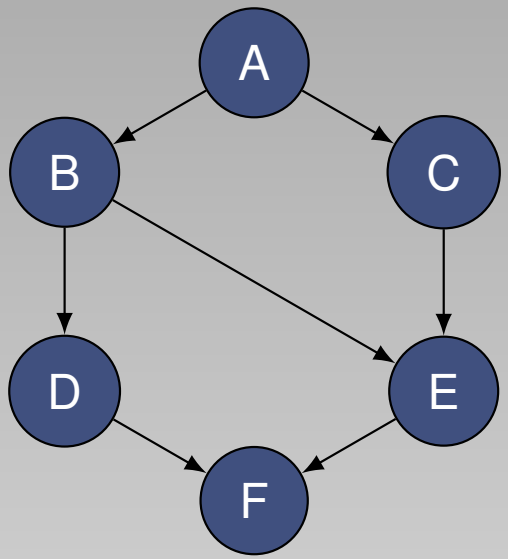


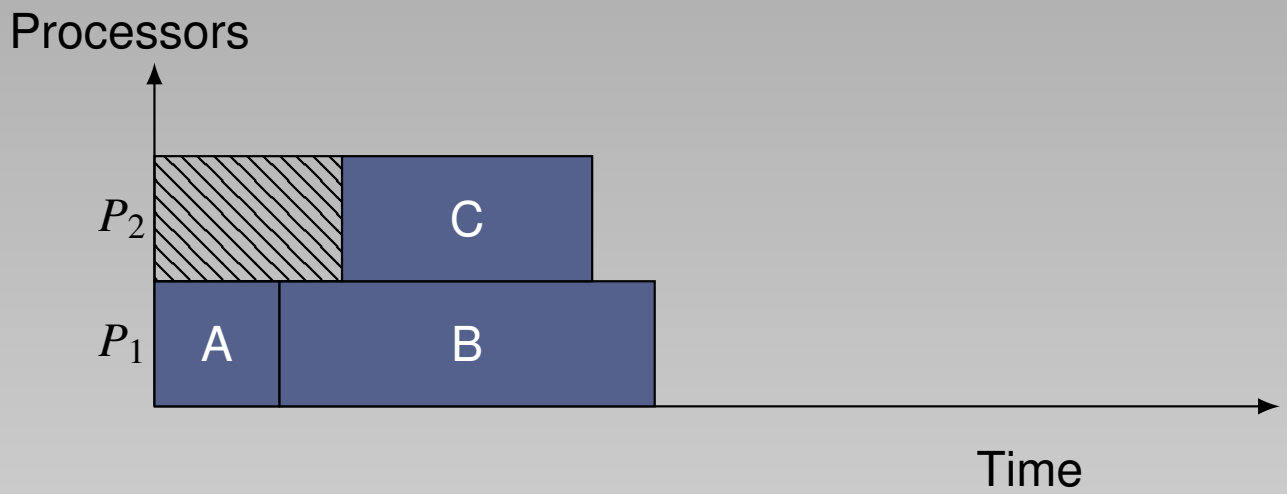
Ready tasks list



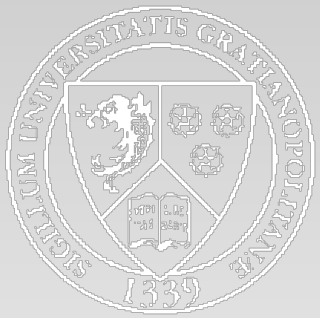
Online Scheduling

- List-scheduling [Graham-69].



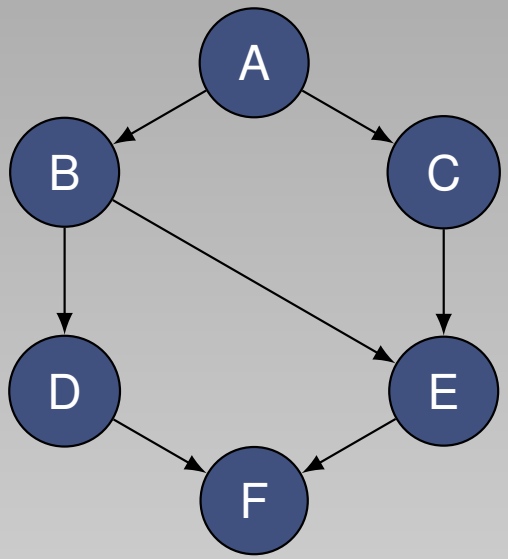


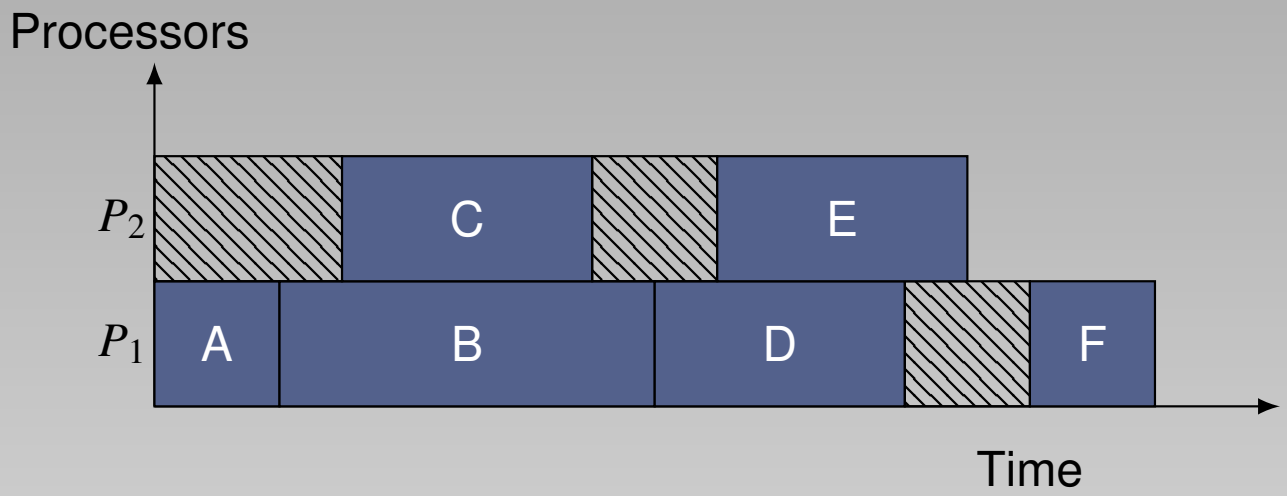
Ready tasks list



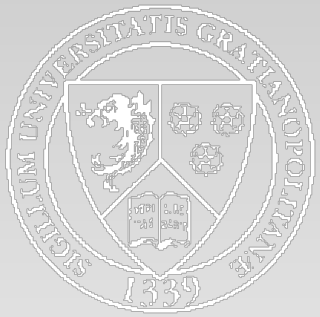
Online Scheduling

- List-scheduling [Graham-69].
  - Centralized



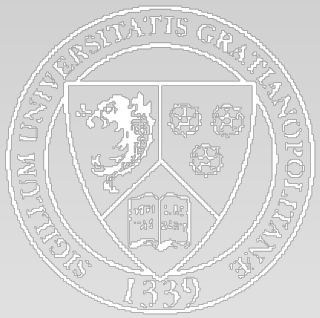


Ready tasks list



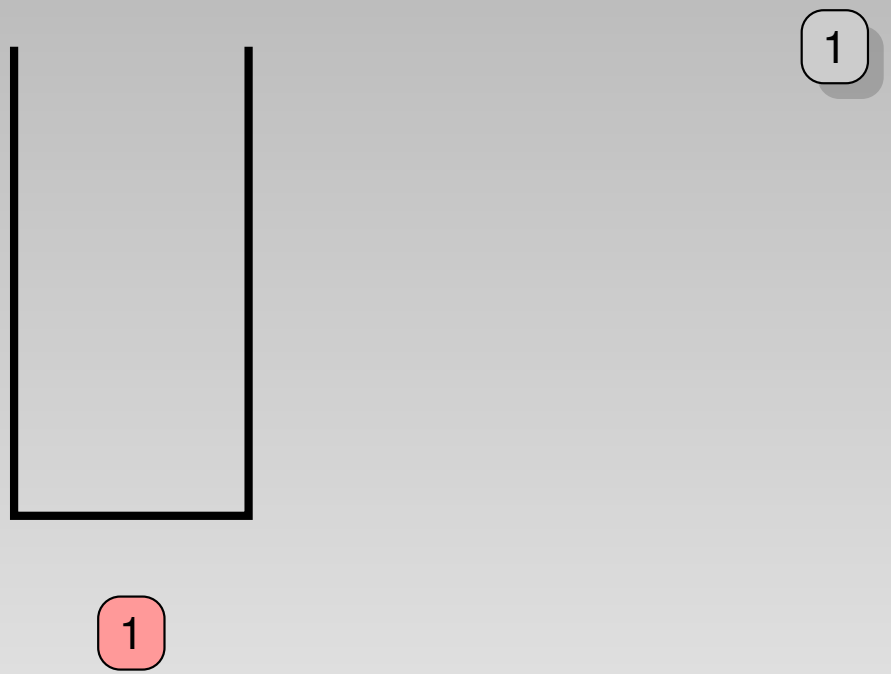
Work-Stealing [Blumofe-95]

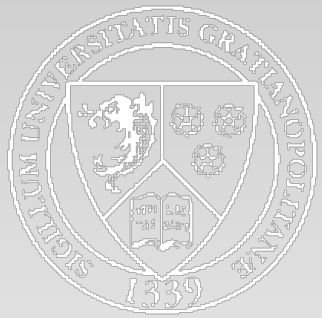
A distributed list scheduling



Work-Stealing [Blumofe-95]

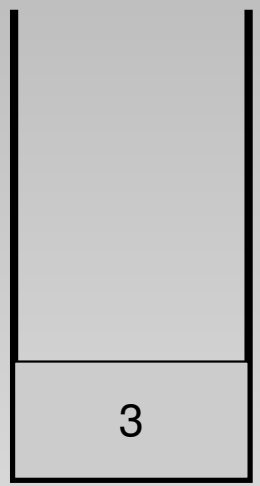
A distributed list scheduling

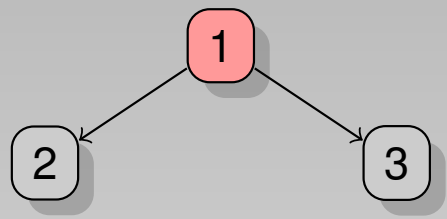


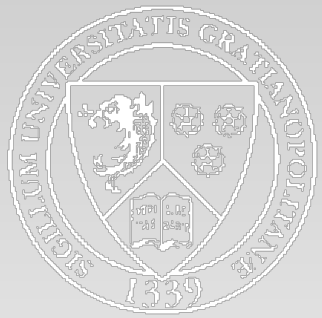


Work-Stealing [Blumofe-95]

A distributed list scheduling

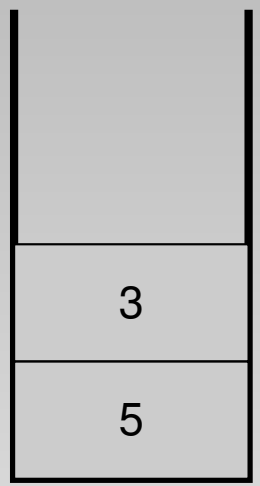


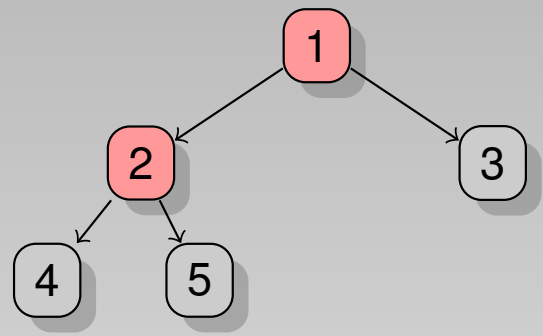


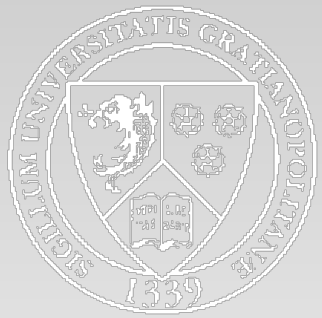


Work-Stealing [Blumofe-95]

A distributed list scheduling



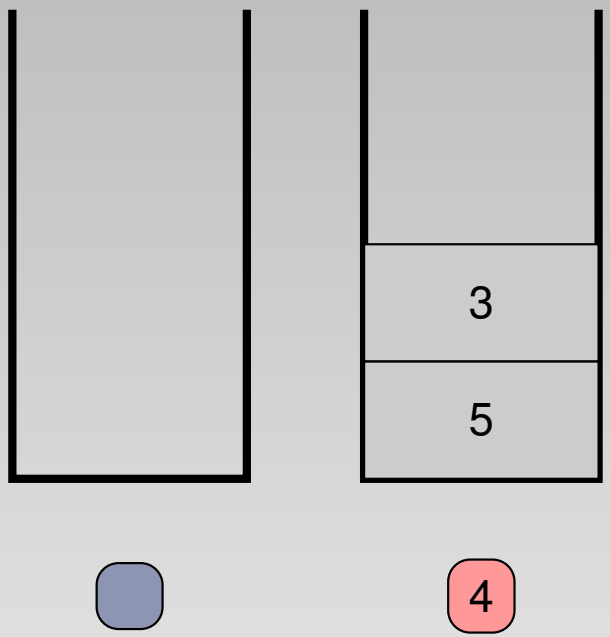


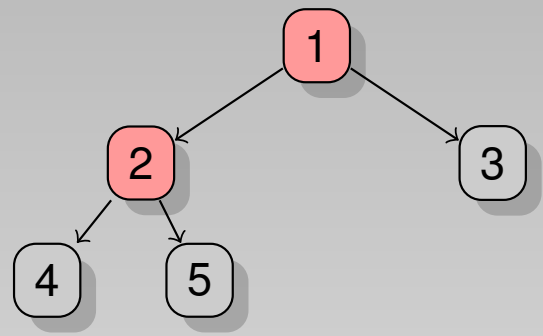


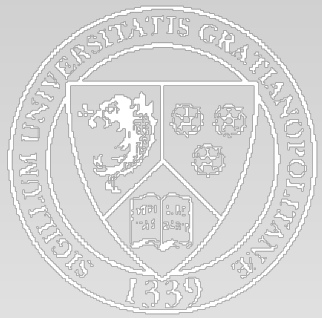
Work-Stealing [Blumofe-95]

A distributed list scheduling

- Choose the stolen processor.



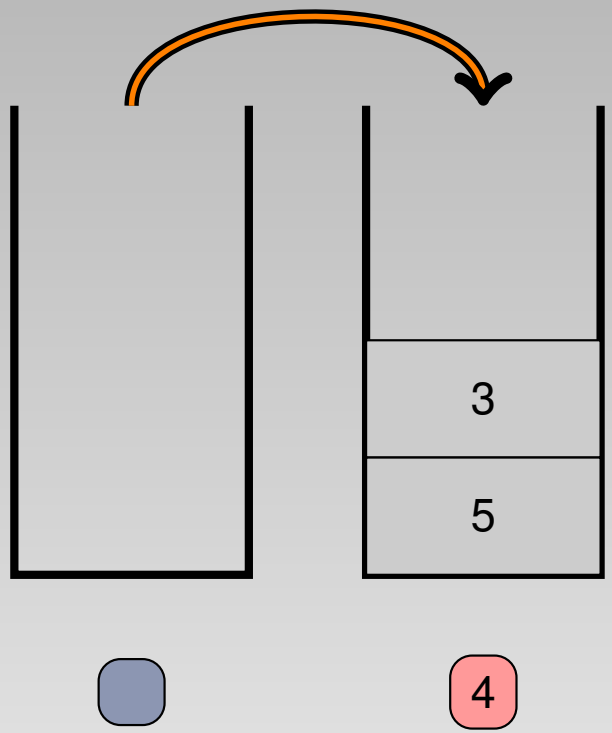


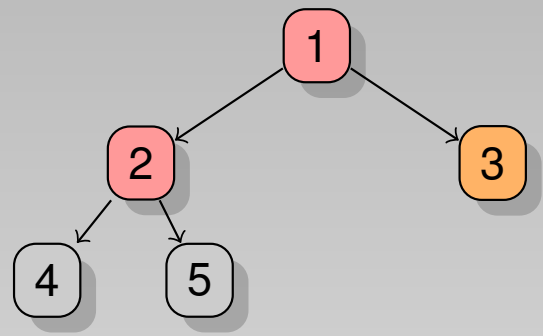


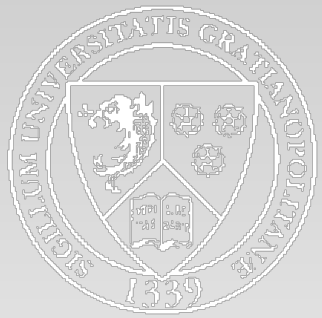
Work-Stealing [Blumofe-95]

A distributed list scheduling

- Choose the stolen processor.
- Choose the stolen task.



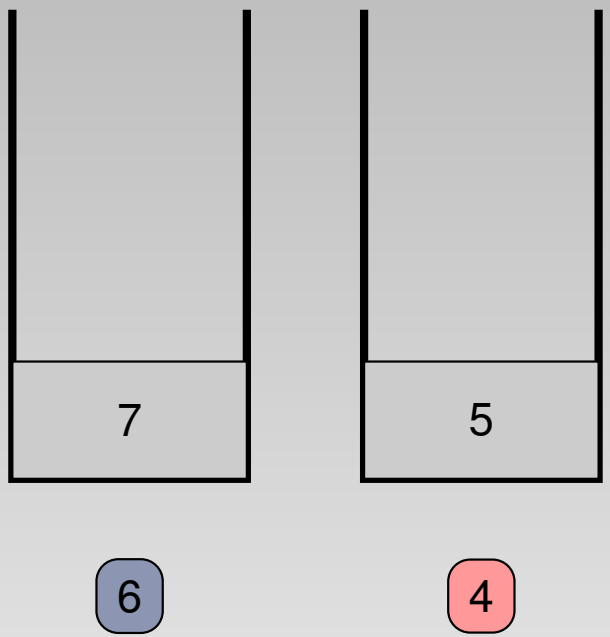


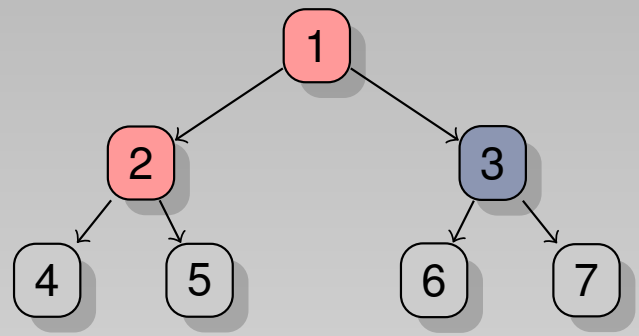


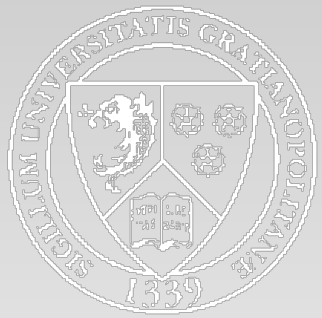
Work-Stealing [Blumofe-95]

A distributed list scheduling

- Choose the stolen processor.
- Choose the stolen task.



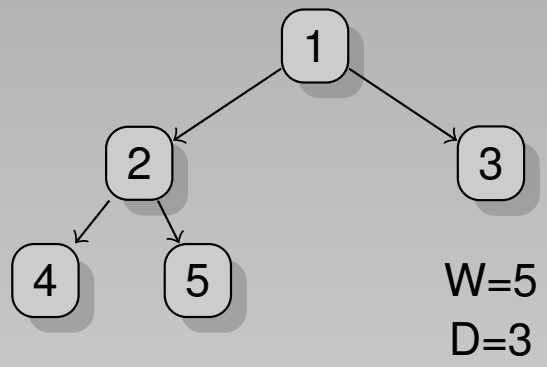


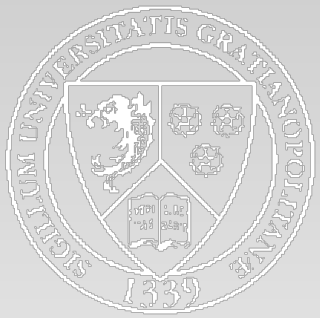


Work-Stealing

Performance analysis

- Assumptions:
  - Constant communication time and no data transfers.
  - DAG arity: 2, and unitary task.
  - Homogeneous processor (Bender & Rabin for heterogeneous).
- Bounds [Arora-01]:

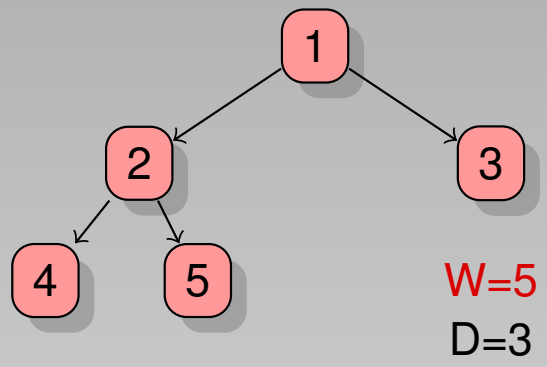


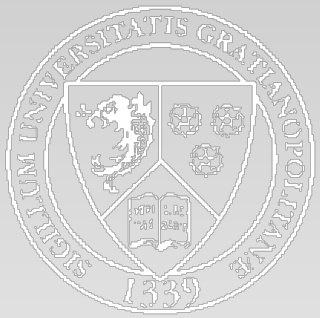


Work-Stealing

Performance analysis

- Assumptions:
  - Constant communication time and no data transfers.
  - DAG arity: 2, and unitary task.
  - Homogeneous processor (Bender & Rabin for heterogeneous).
- Bounds [Arora-01]:

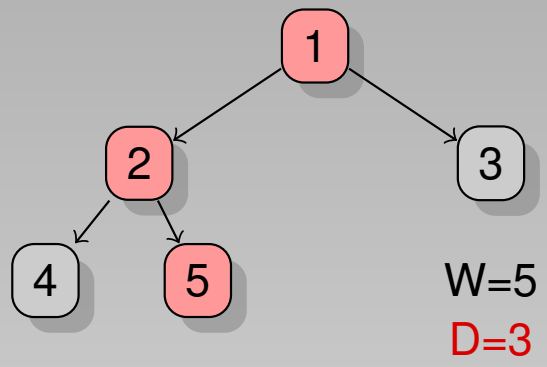


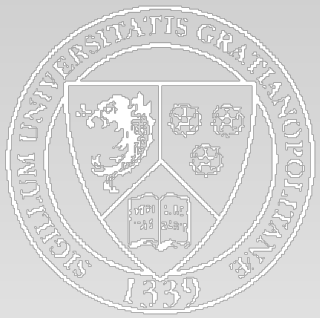


Work-Stealing

Performance analysis

- Assumptions:
  - Constant communication time and no data transfers.
  - DAG arity: 2, and unitary task.
  - Homogeneous processor (Bender & Rabin for heterogeneous).
- Bounds [Arora-01]:

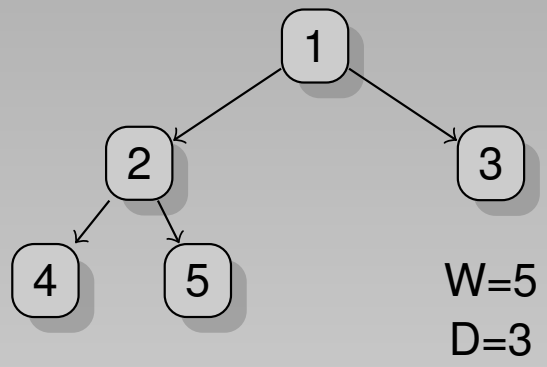




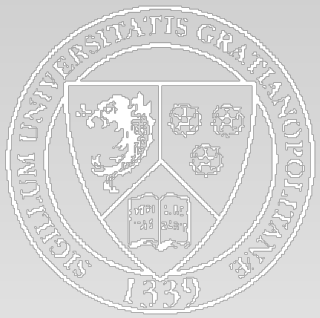
Work-Stealing

Performance analysis

- Assumptions:
  - Constant communication time and no data transfers.
  - DAG arity: 2, and unitary task.
  - Homogeneous processor (Bender & Rabin for heterogeneous).

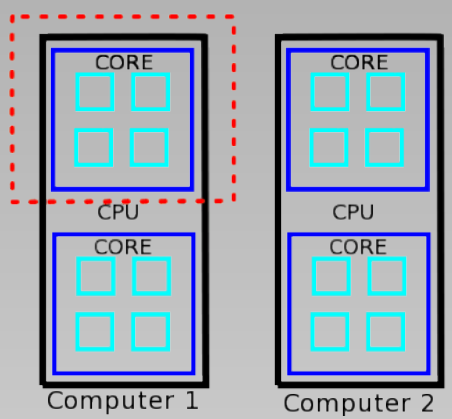


- Bounds [Arora-01]:
  - Execution time : $T_p \leq \frac{W}{p} + O(D)$.
  - Data transfers : bounded by the number of edges.
  - Steal requests : $\#S \leq O(p * D)$.

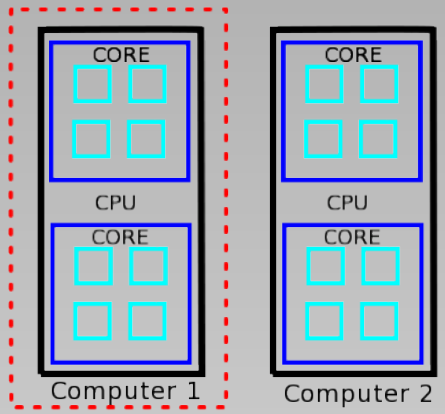


Work-Stealing and Communications

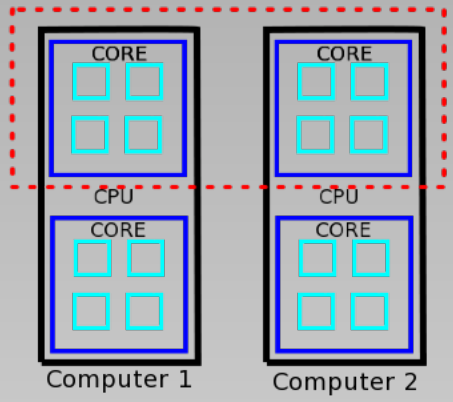
- Data intensive application.
- Distributed platform (cluster).



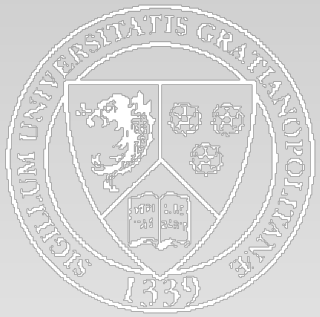
$\approx$ 55 seconds



$\approx$ 35 seconds



$\approx$ 58 seconds



Existing Algorithm

Taking into account platform topology

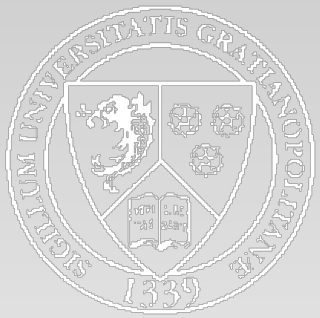
In Satin [Nieuwpoort-01]:

- CLS: Only one processor by cluster can send steal requests to processors inside other clusters.
- CHS: Each processor is a node of a binary tree. Steal requests are sent through the tree.
- CRS: Each processor can send two steal requests at the same time: one asynchronous and one synchronous.

In Kaapi [Gautier-07]:

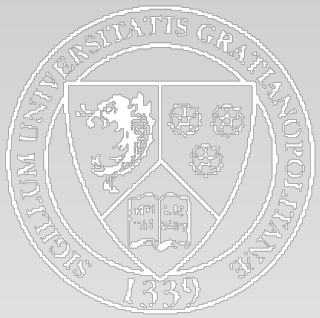
- KWS: Each processor sends a request to the processor on the same computer before stealing a processor on another computer.

Outline



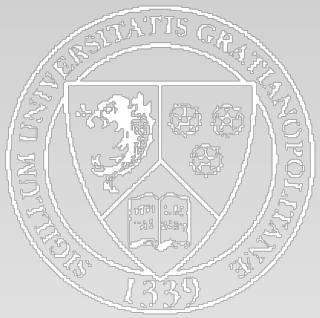
Context

- DSMake:
  - Makefiles executions on distributed platforms.
  - Structure unrestricted.
  - Static DAG: structure is known in advance.

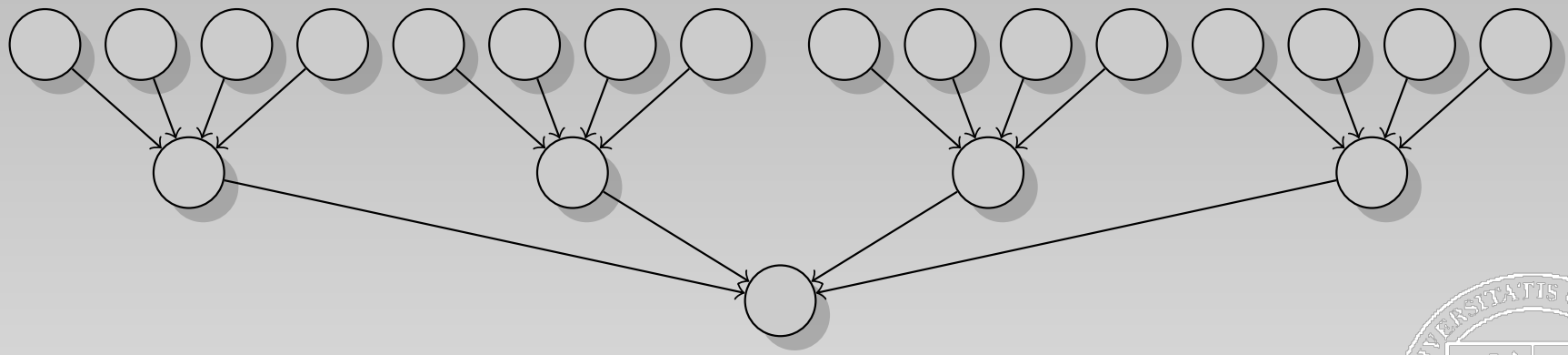


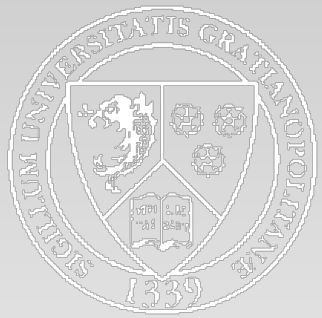
Context

- DSMake:
  - Makefiles executions on distributed platforms.
  - Structure unrestricted.
  - Static DAG: structure is known in advance.
- Our aim:
  - Take into account the DAG structure to minimize the number of transfers.

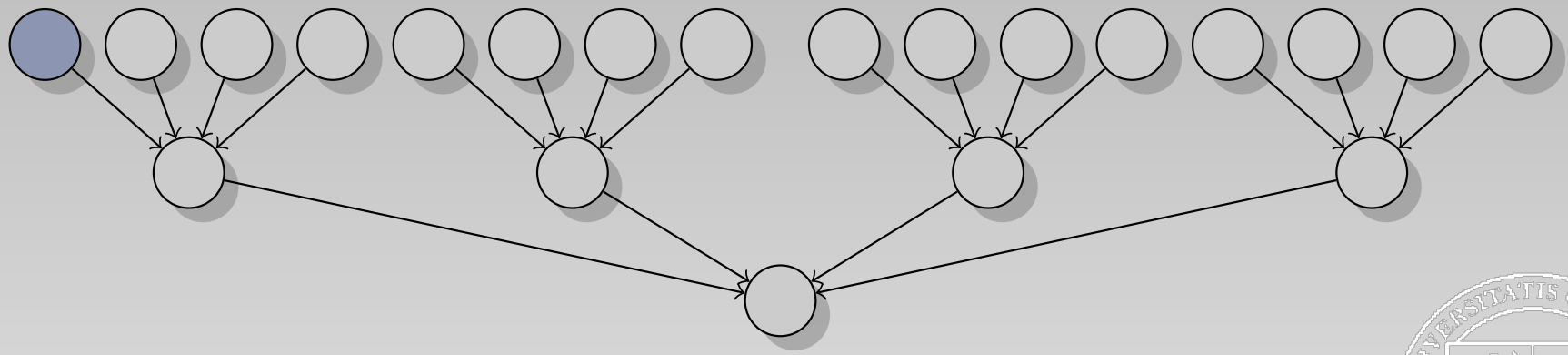


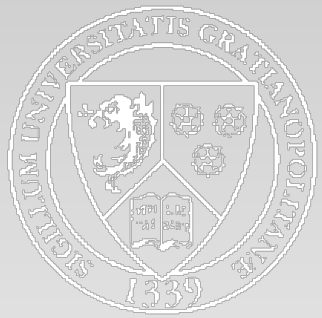
Simple Example



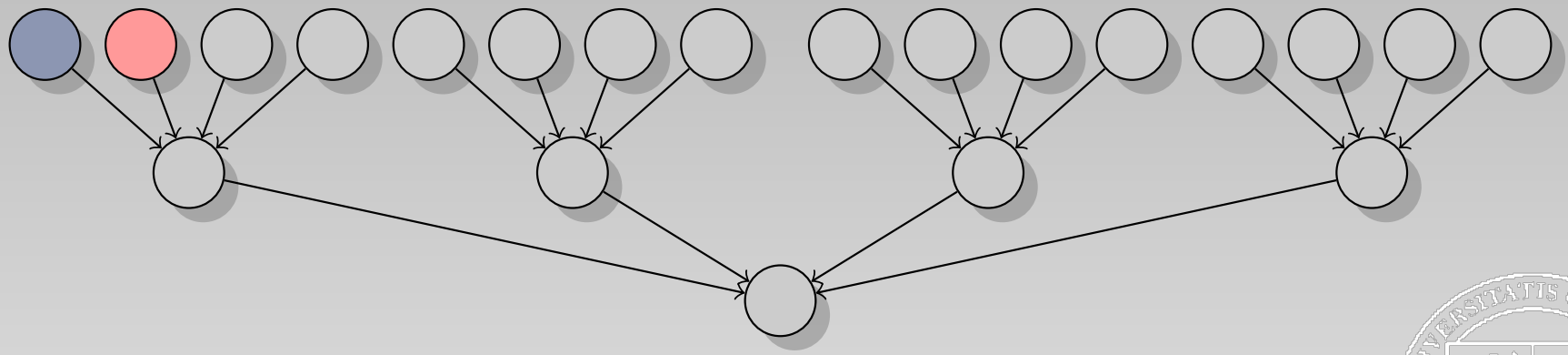


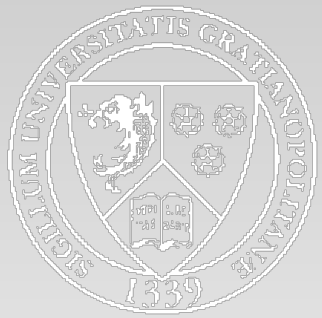
Simple Example



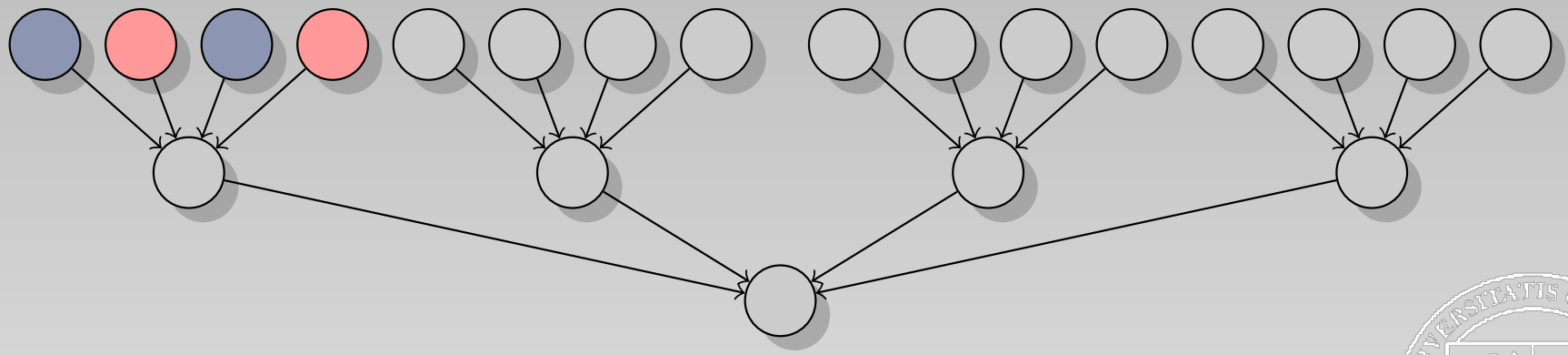


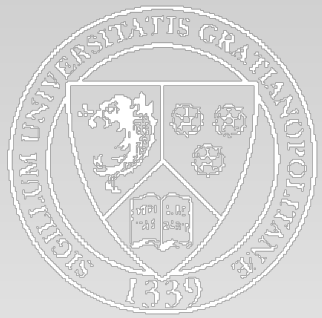
Simple Example



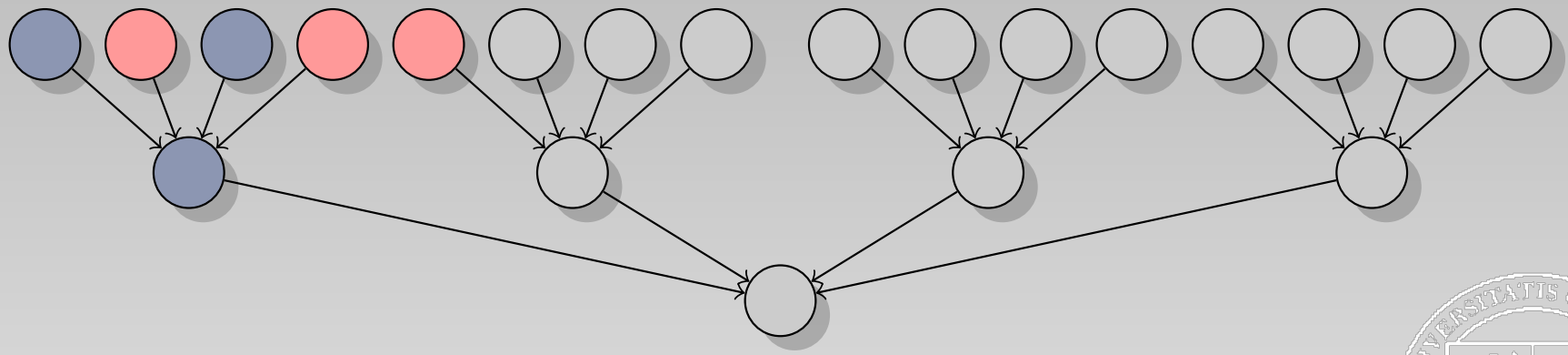


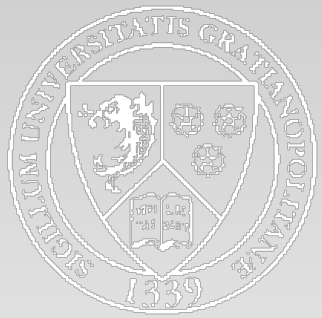
Simple Example



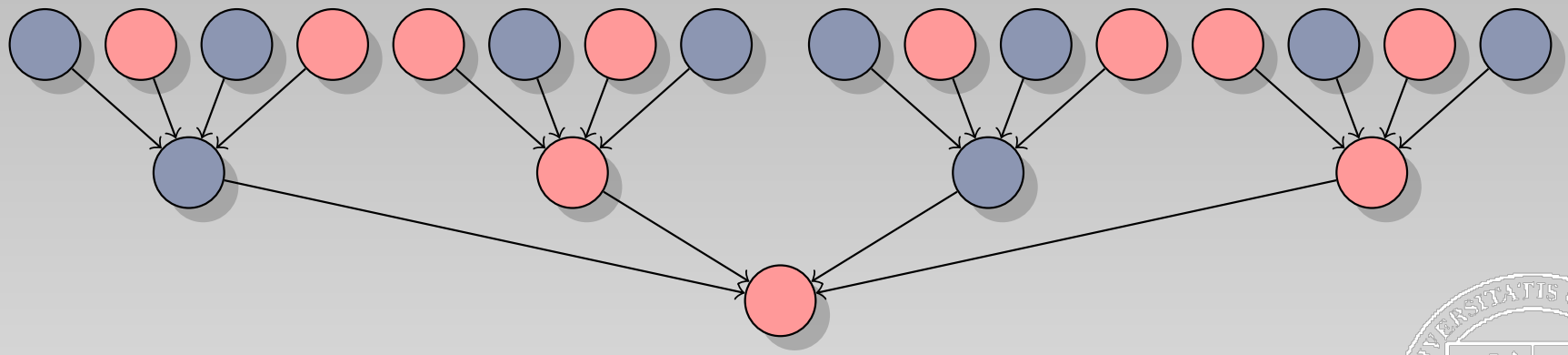


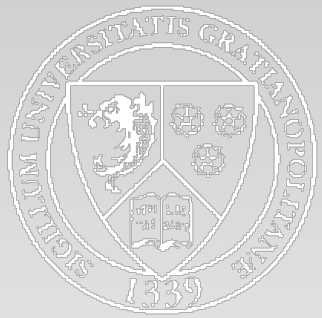
Simple Example



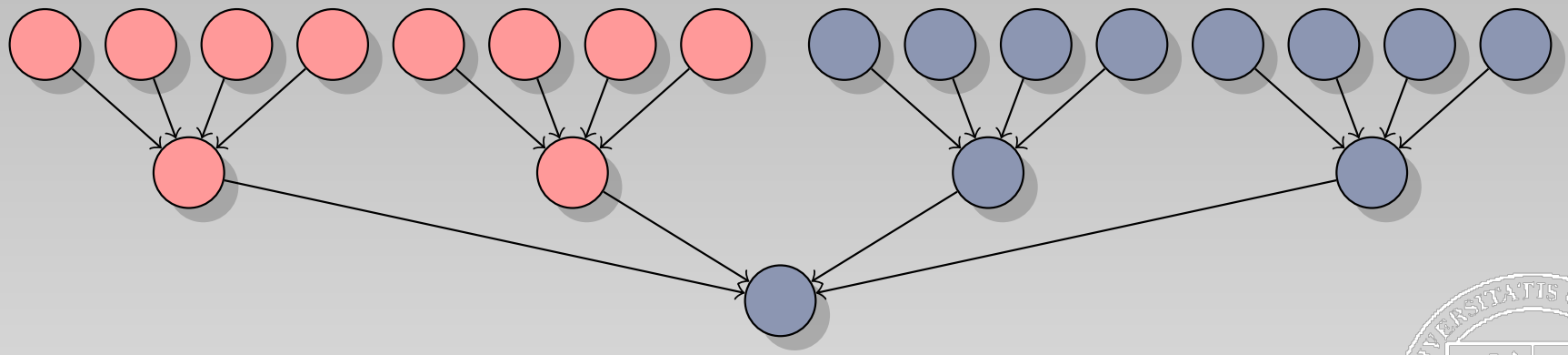


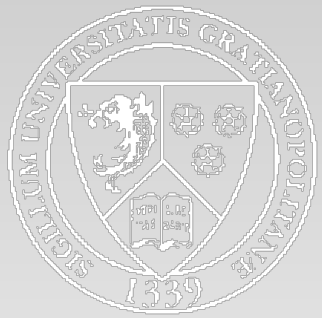
Simple Example





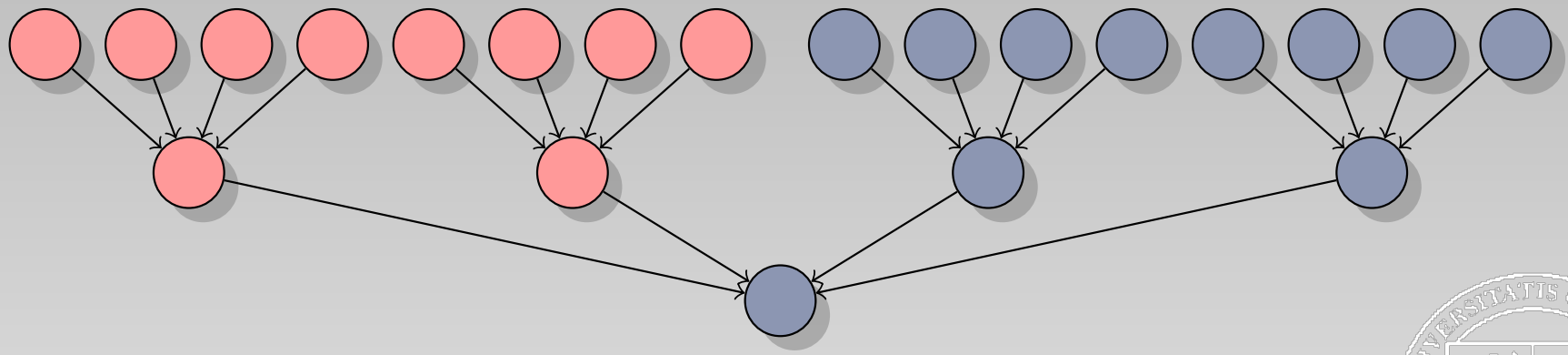
Simple Example

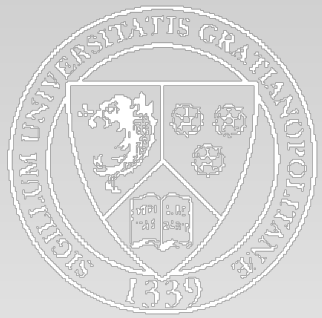




Simple Example

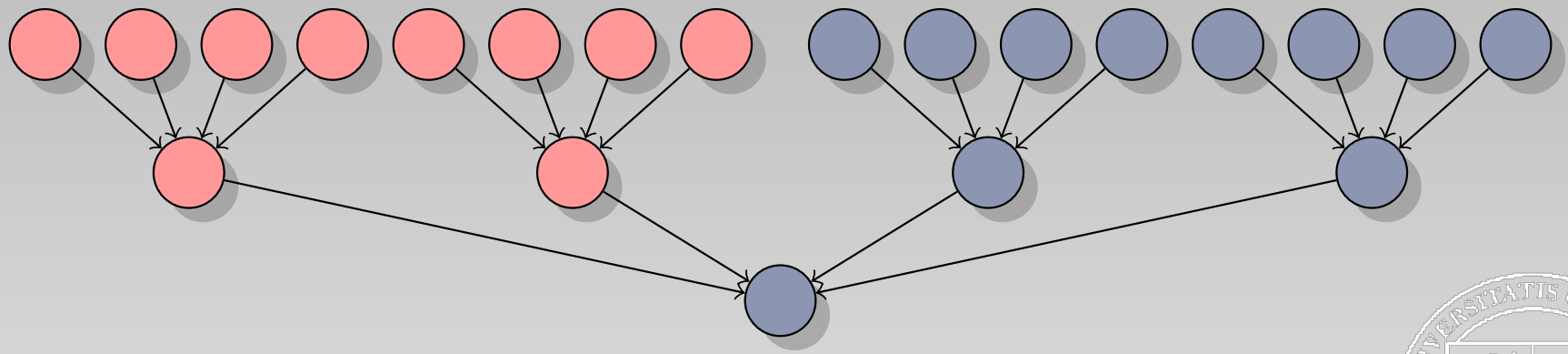
- The scheduling depends on tasks management.

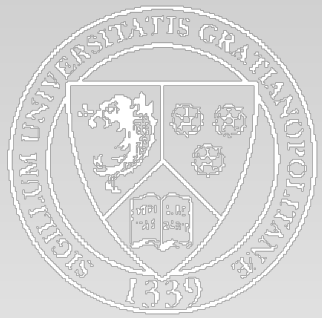




Simple Example

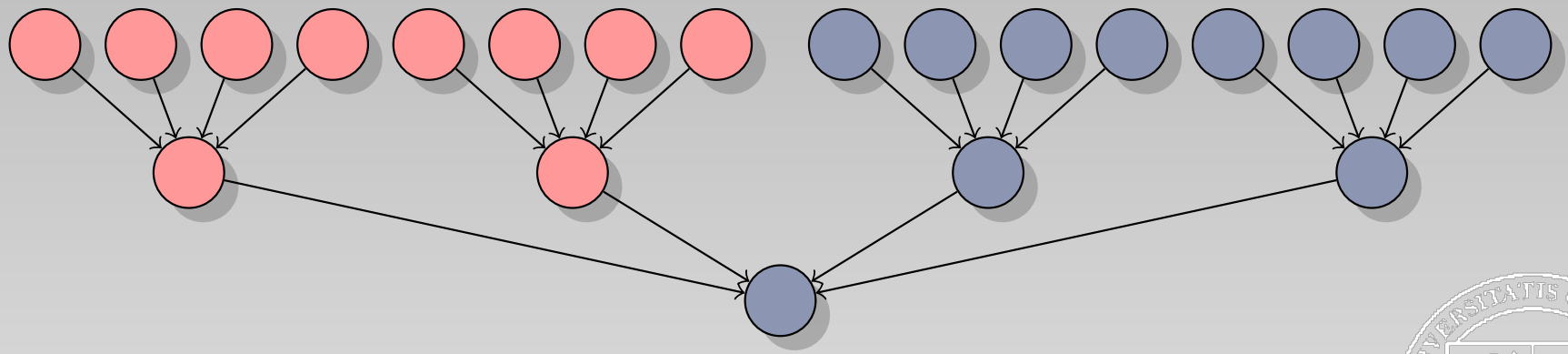
- The scheduling depends on tasks management.
- Add tasks which generate a tasks block.

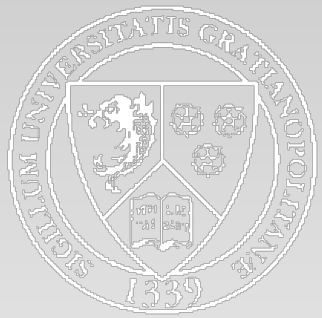




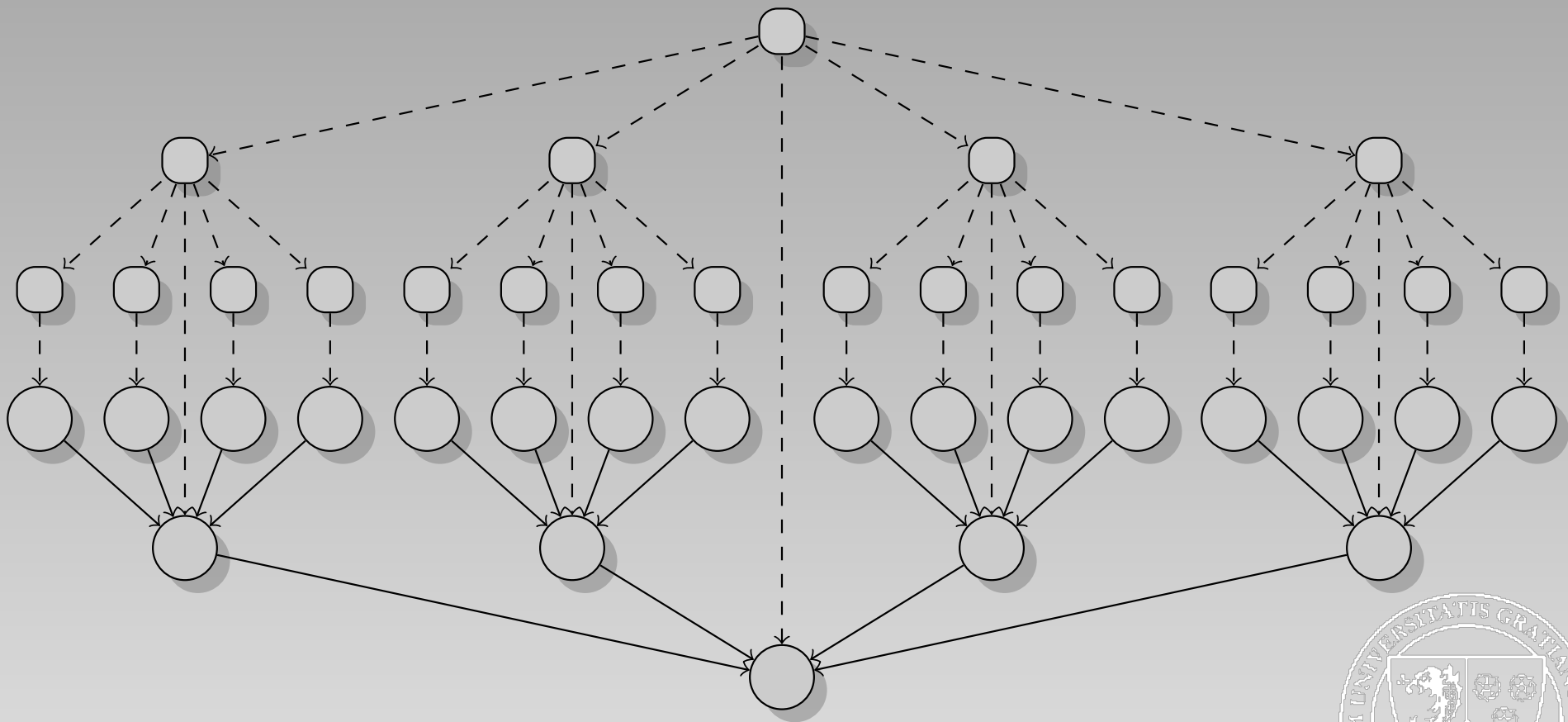
Simple Example

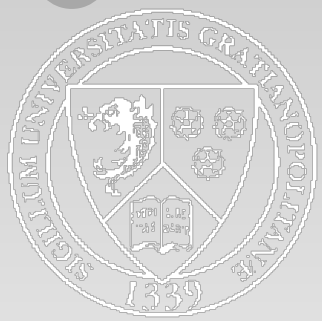
- The scheduling depends on tasks management.
- Add tasks which generate a tasks block.
  - Symmetry of the DAG.



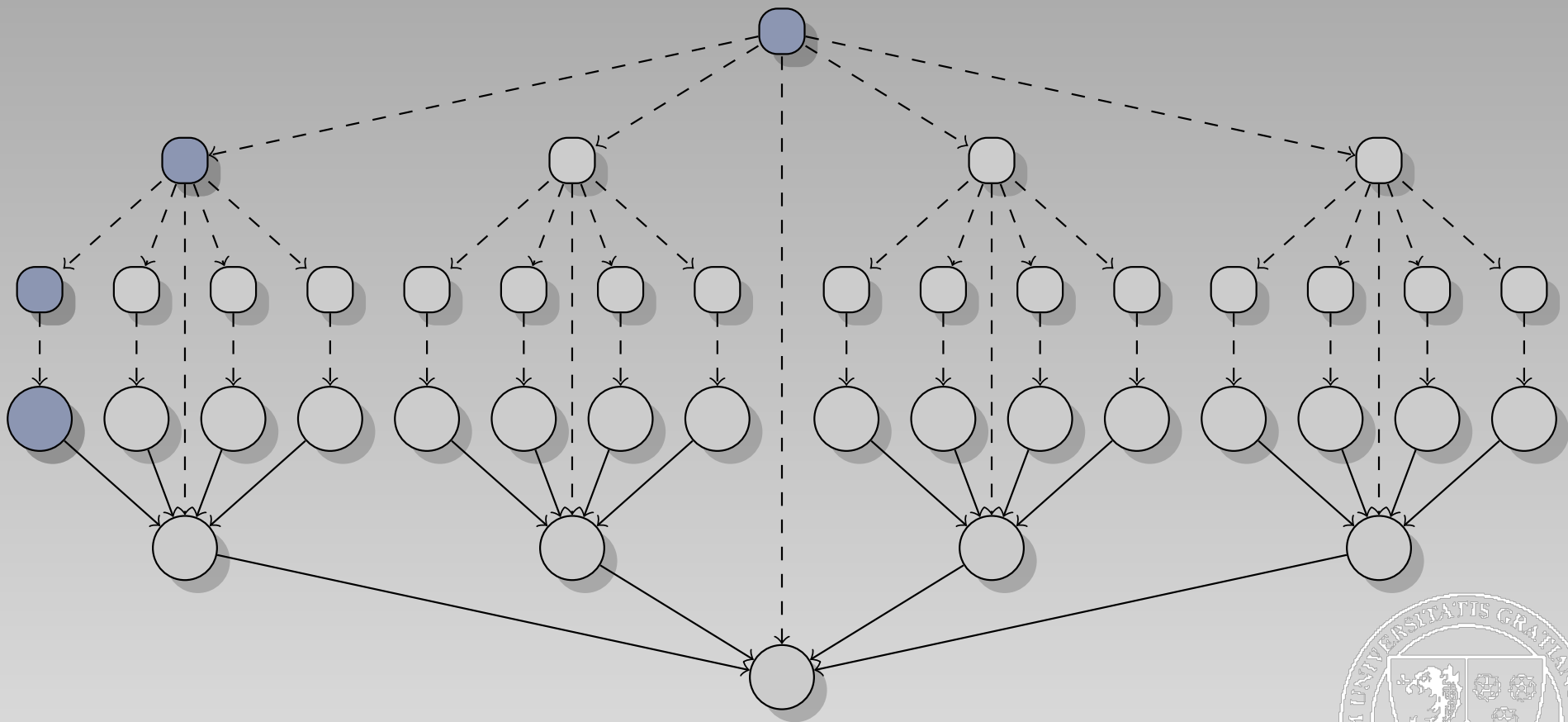


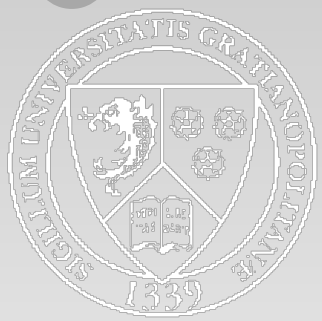
Symmetry of the DAG



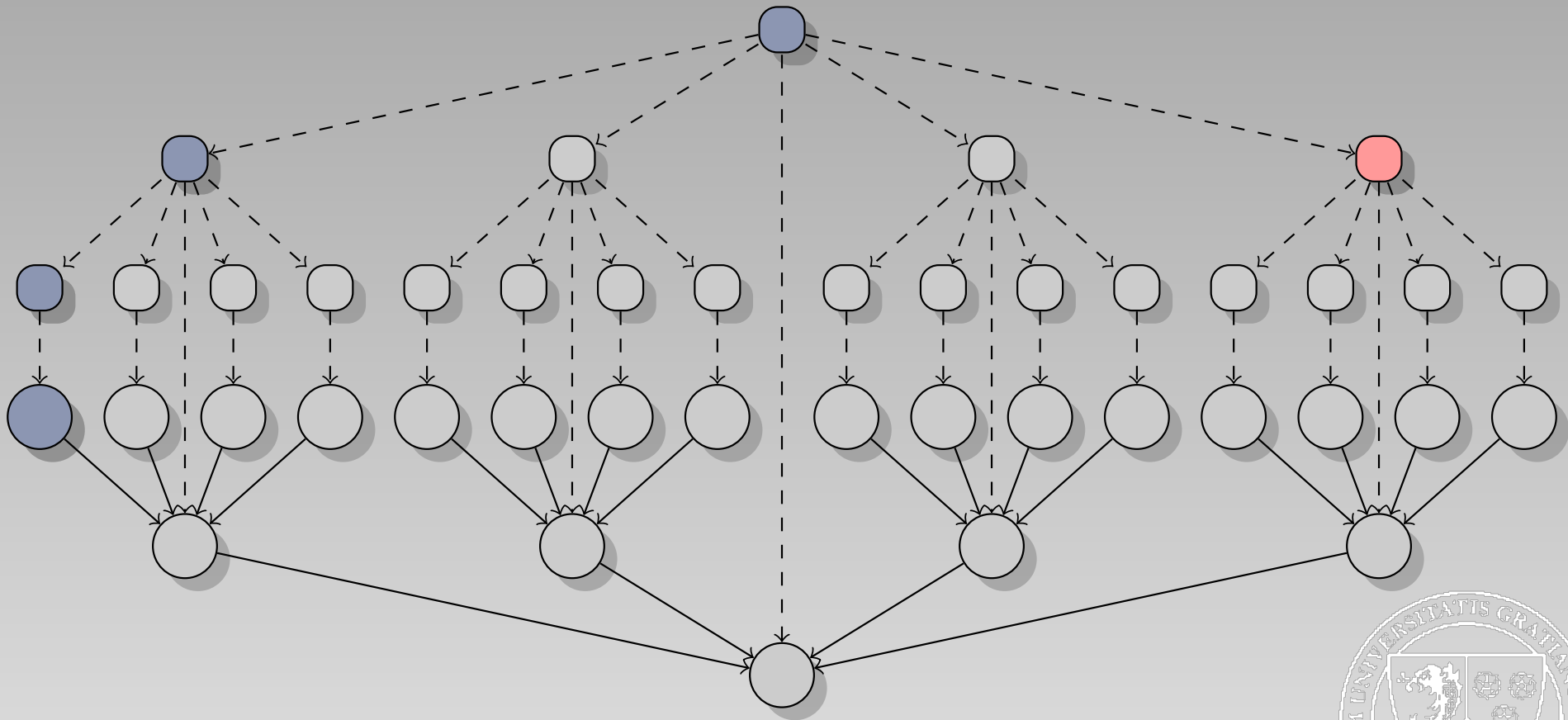


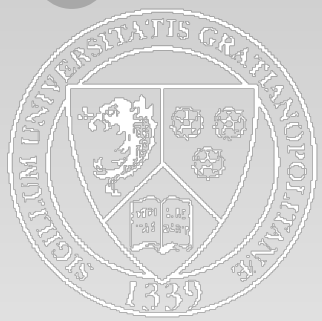
Symmetry of the DAG



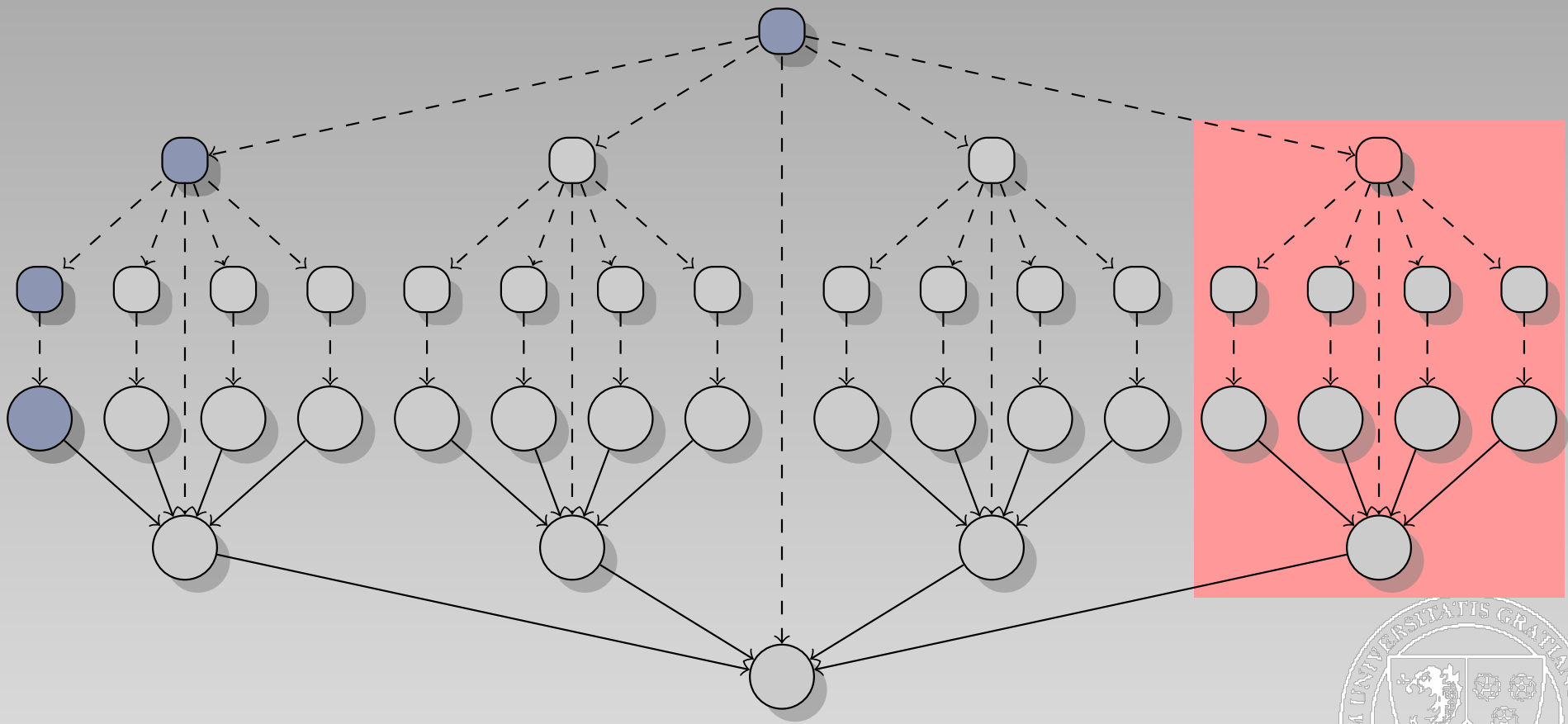


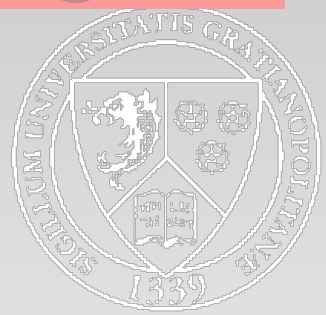
Symmetry of the DAG





Symmetry of the DAG

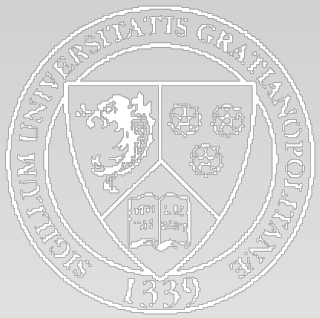




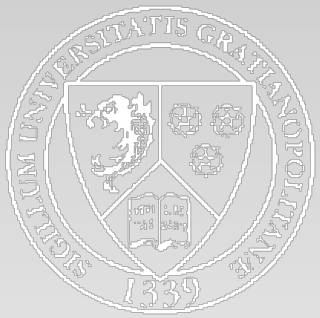
WSCOM

Data communications

1. Add some virtual task before the execution.
2. Execute the new DAG with a work-stealing algorithm.
   - Manage data transfers.
     - Send data the earliest (WSCOM$_{pf}$).
     - Send data the latest (WSCOM).

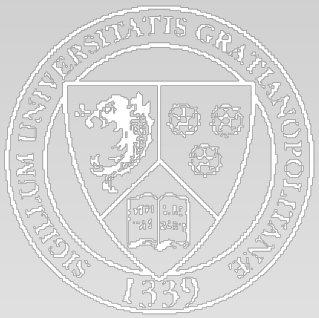


Outline



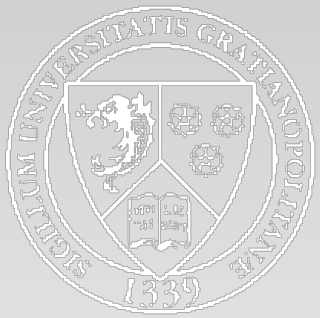
Practical Analysis

- Experiments vs simulations
- Simulations:
  - Many experiments.
  - Varying the platform characteristics (bandwidth).
  - Control of tasks execution time and communications time.
- Scheduling algorithms:
  - On-line heuristics:
    - Classical Work-Stealing.
  - Off-line heuristics:
    - List_min $\min(HEFT, CPOP, BIL, HBMCT, Sufferage, MinMin, MaxMin)$.
    - Known tasks execution time and data transfers time.
    - Not contention aware.



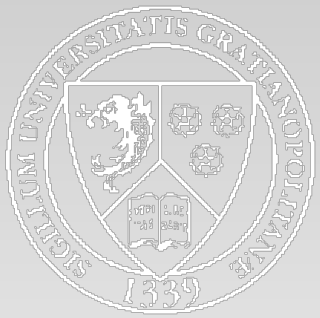
Inputs

- Platforms:
  - Clique without network contentions.
  - Cluster with network contentions.



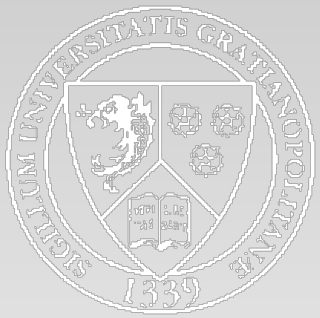
Inputs

- Platforms:
  - Clique without network contentions.
  - Cluster with network contentions.
- Application DAG:



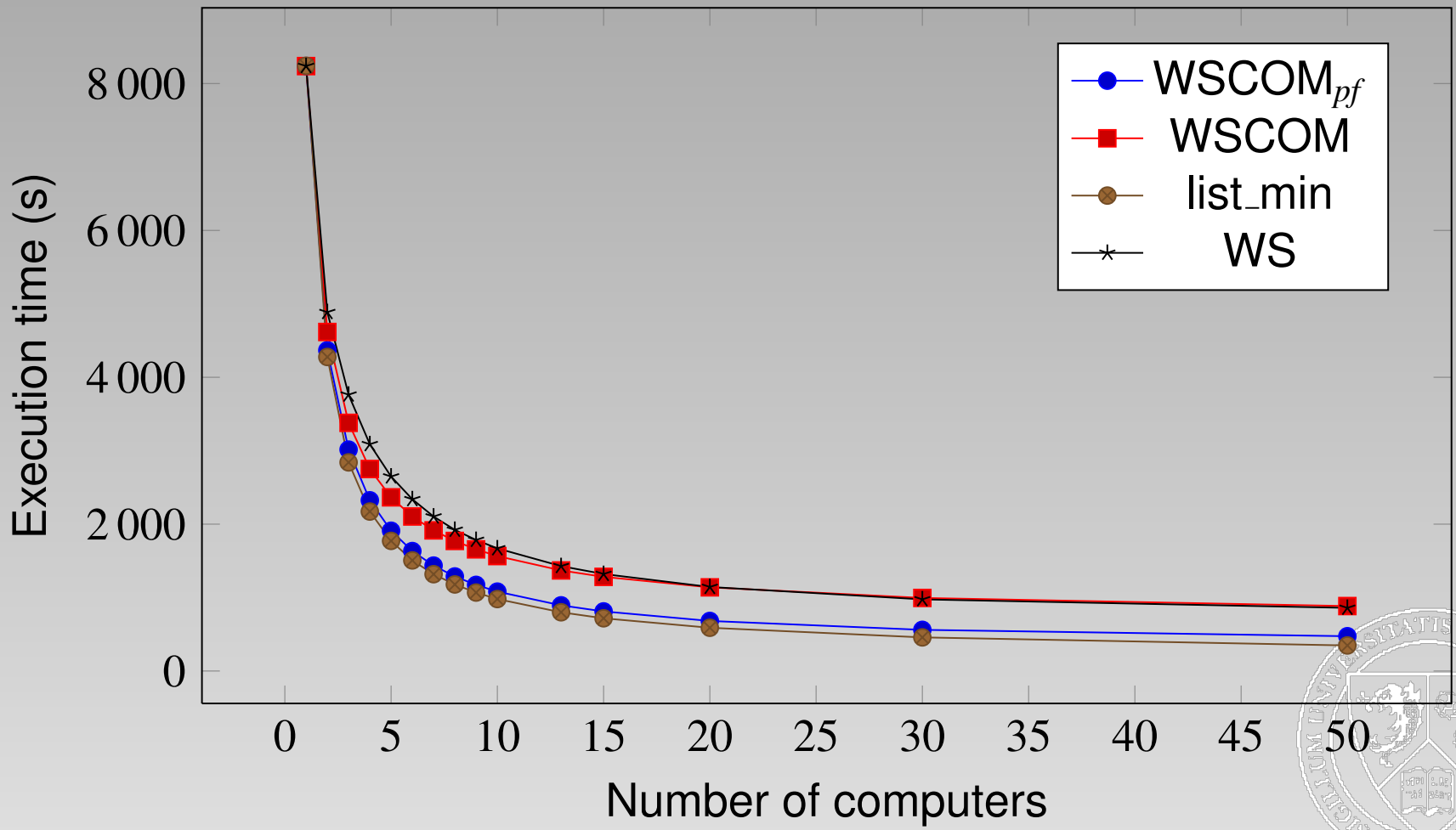
Inputs

- Platforms:
  - Clique without network contentions.
  - Cluster with network contentions.
- Application DAG:
  - Random DAG (TGFF [Dick-98], LBL [Tobita-02]).
  - DAG extracted from Makefile execution.



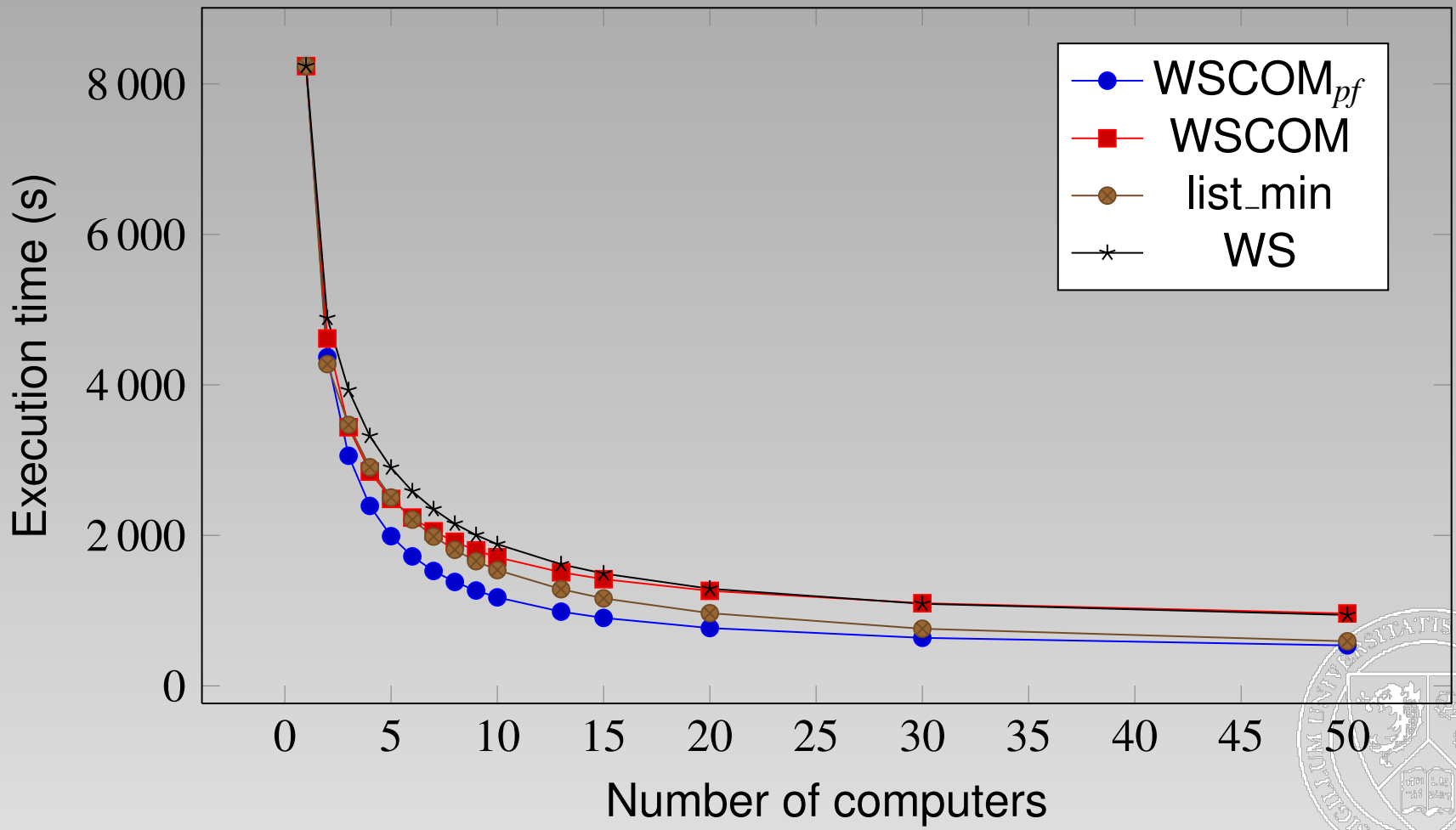
No Contention on Links

Clique platform, random DAG

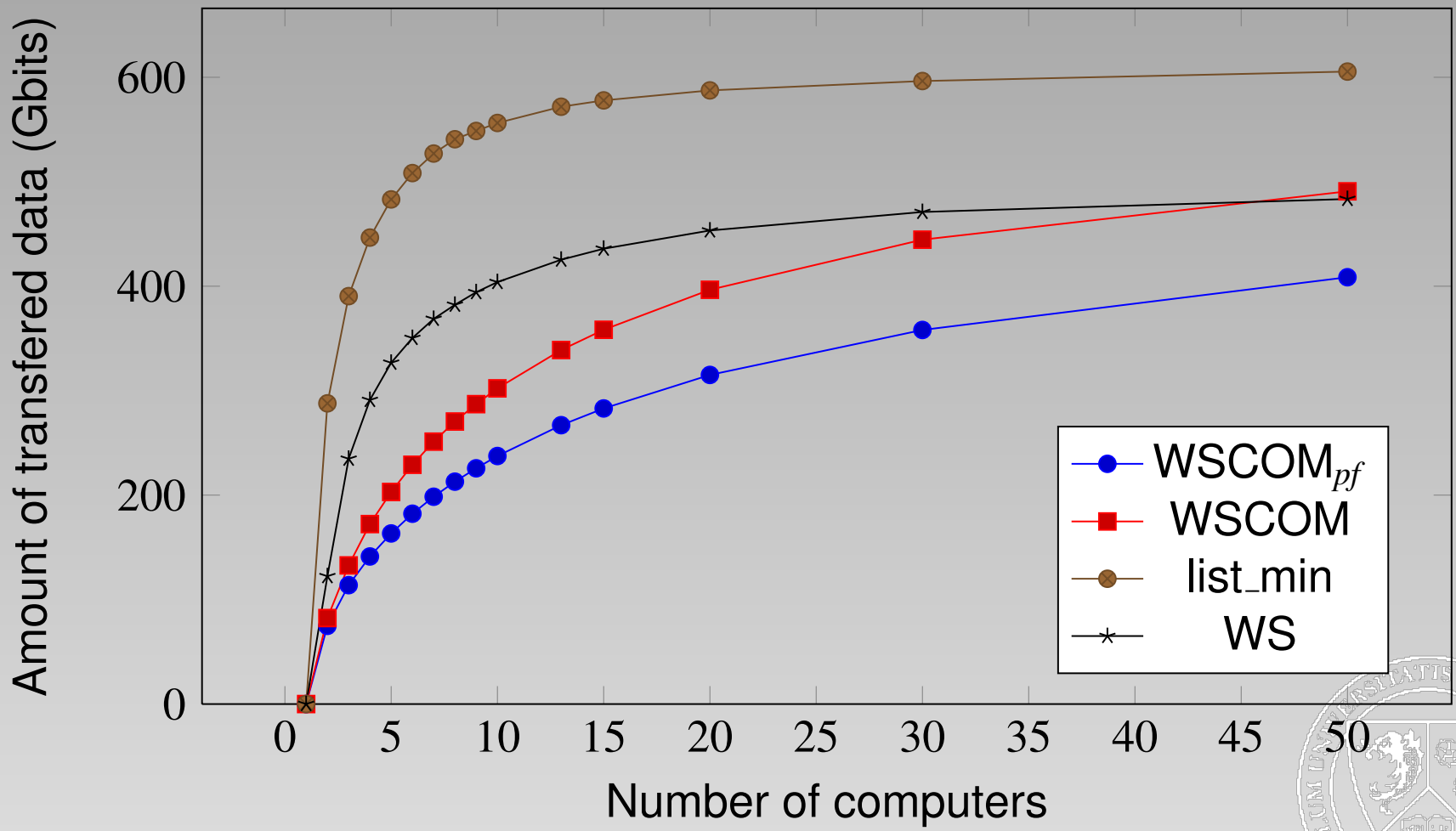


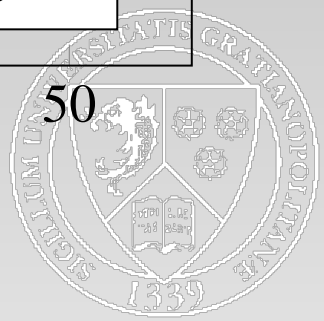
Contention on Links

Cluster Platform, Random DAG



Data Transfers

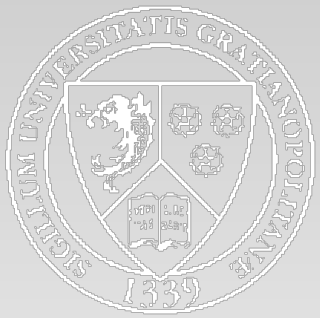




DAG from Makefile Executions

WSCOM vs WS

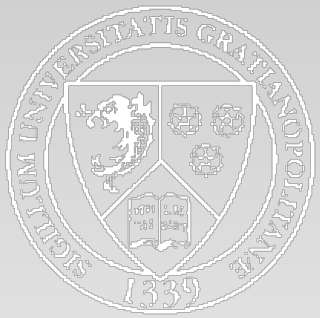
- DAG:
  - 500 different DAG.
  - Compilation of open-source softwares (MacPort [Rothman-08]).



DAG from Makefile Executions

WSCOM vs WS

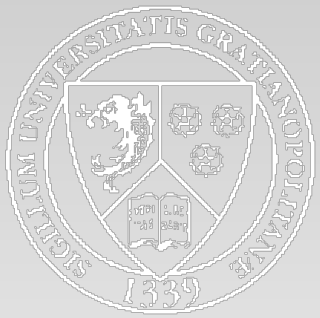
- DAG:
  - 500 different DAG.
  - Compilation of open-source softwares (MacPort [Rothman-08]).
- Two kind of experiments:
  - Experiments as previous on random DAG.
    - Sligthly different results
  - Highlight the ability to exploit different platforms:



DAG from Makefile Executions

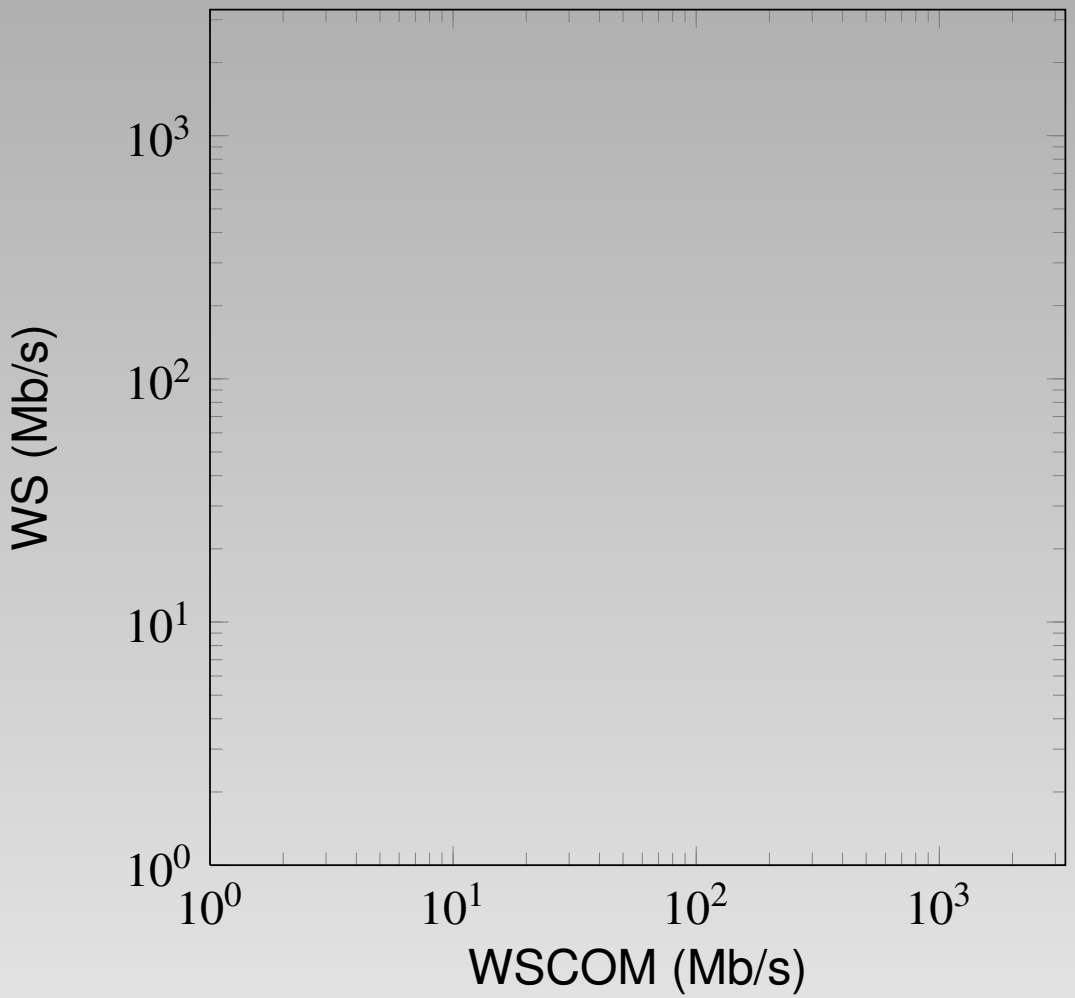
WSCOM vs WS

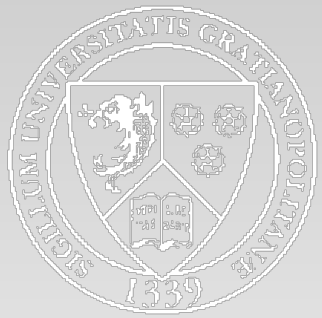
- DAG:
  - 500 different DAG.
  - Compilation of open-source softwares (MacPort [Rothman-08]).
- Two kind of experiments:
  - Experiments as previous on random DAG.
    - Sligthly different results
  - Highlight the ability to exploit different platforms:
    - Can WSCOM achieve a significant speed-up with a low bandwidth



DAG from Makefile Execution

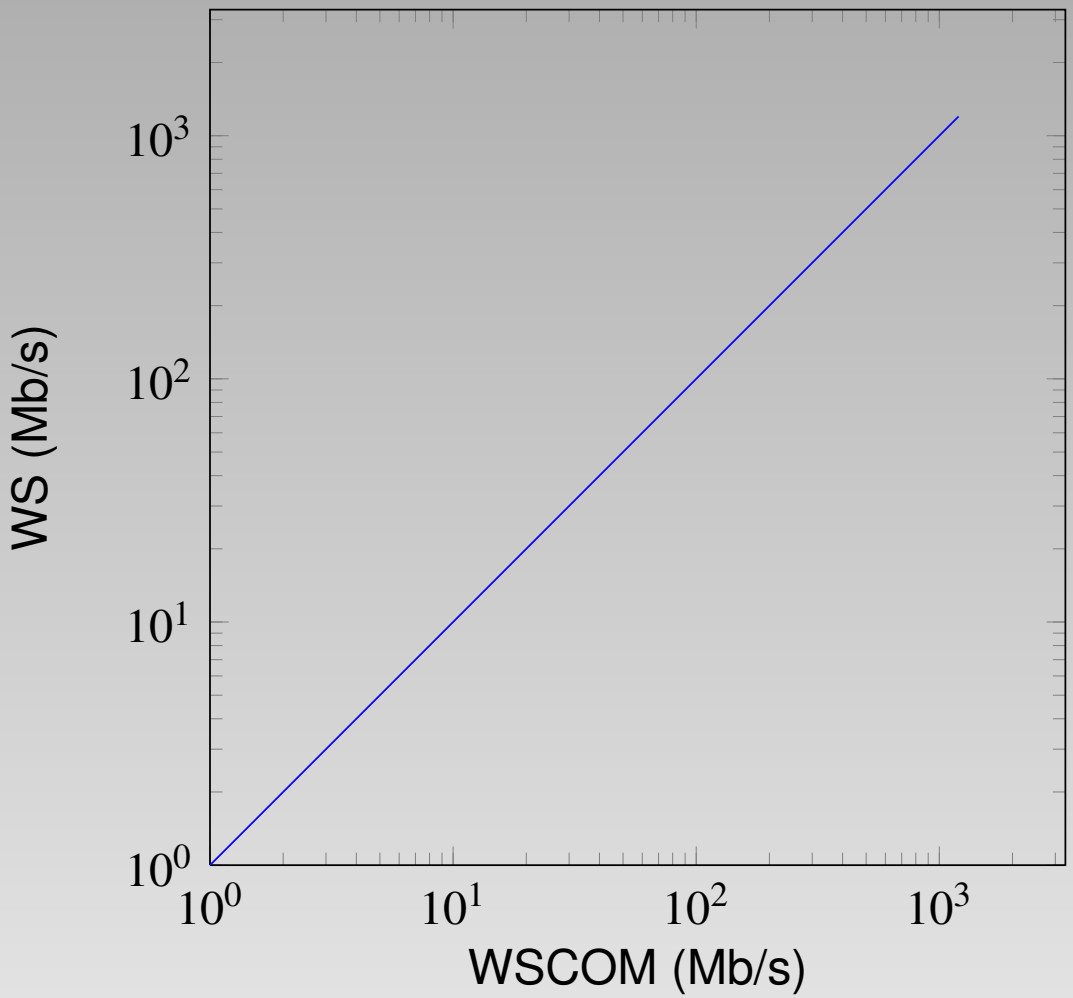
WSCOM vs WS

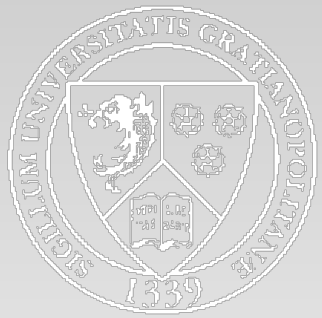




DAG from Makefile Execution

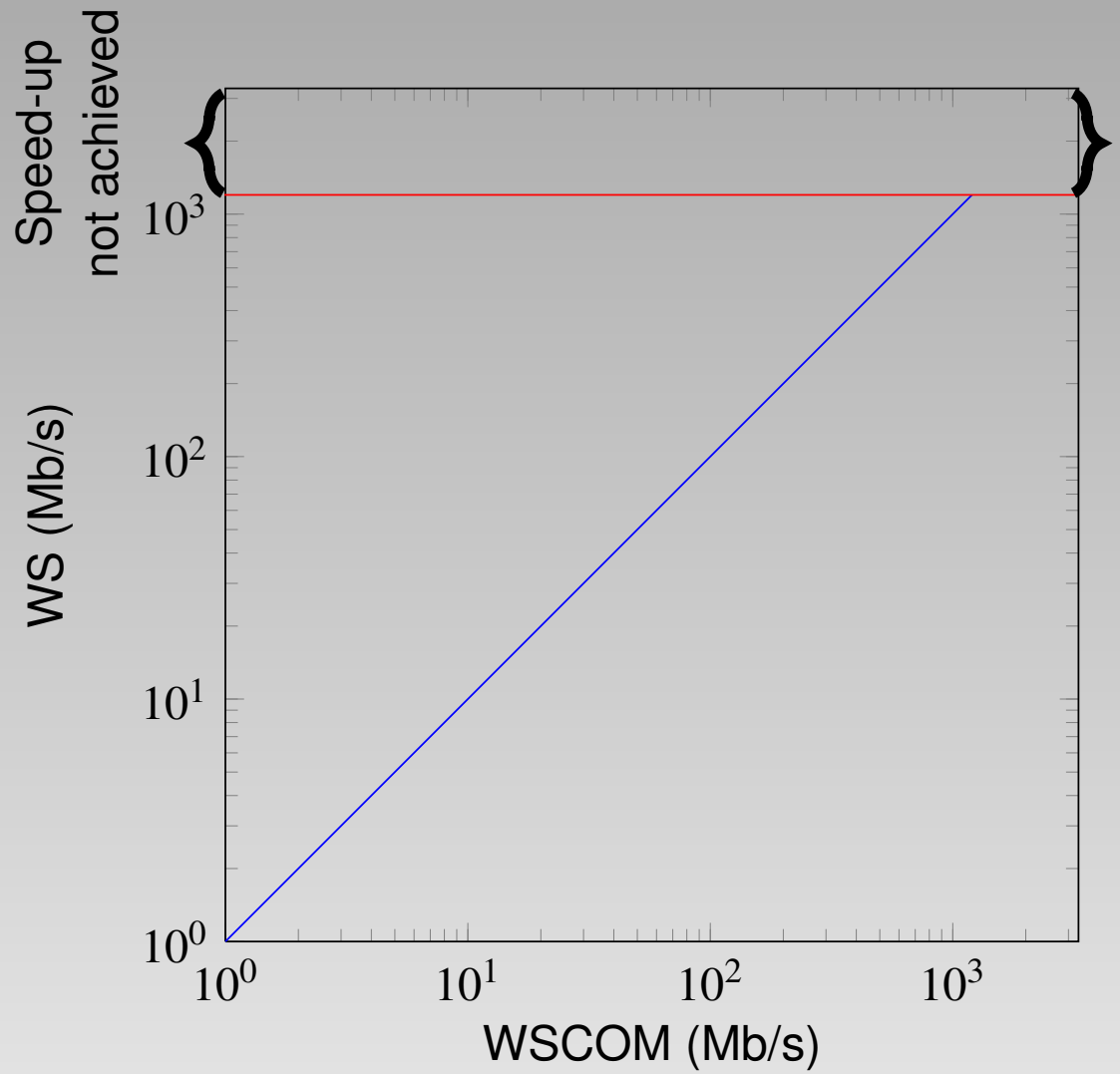
WSCOM vs WS

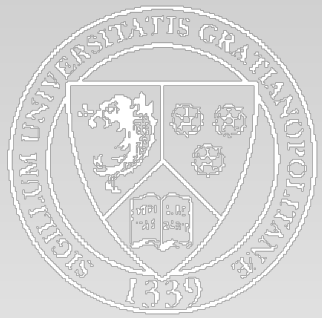




DAG from Makefile Execution

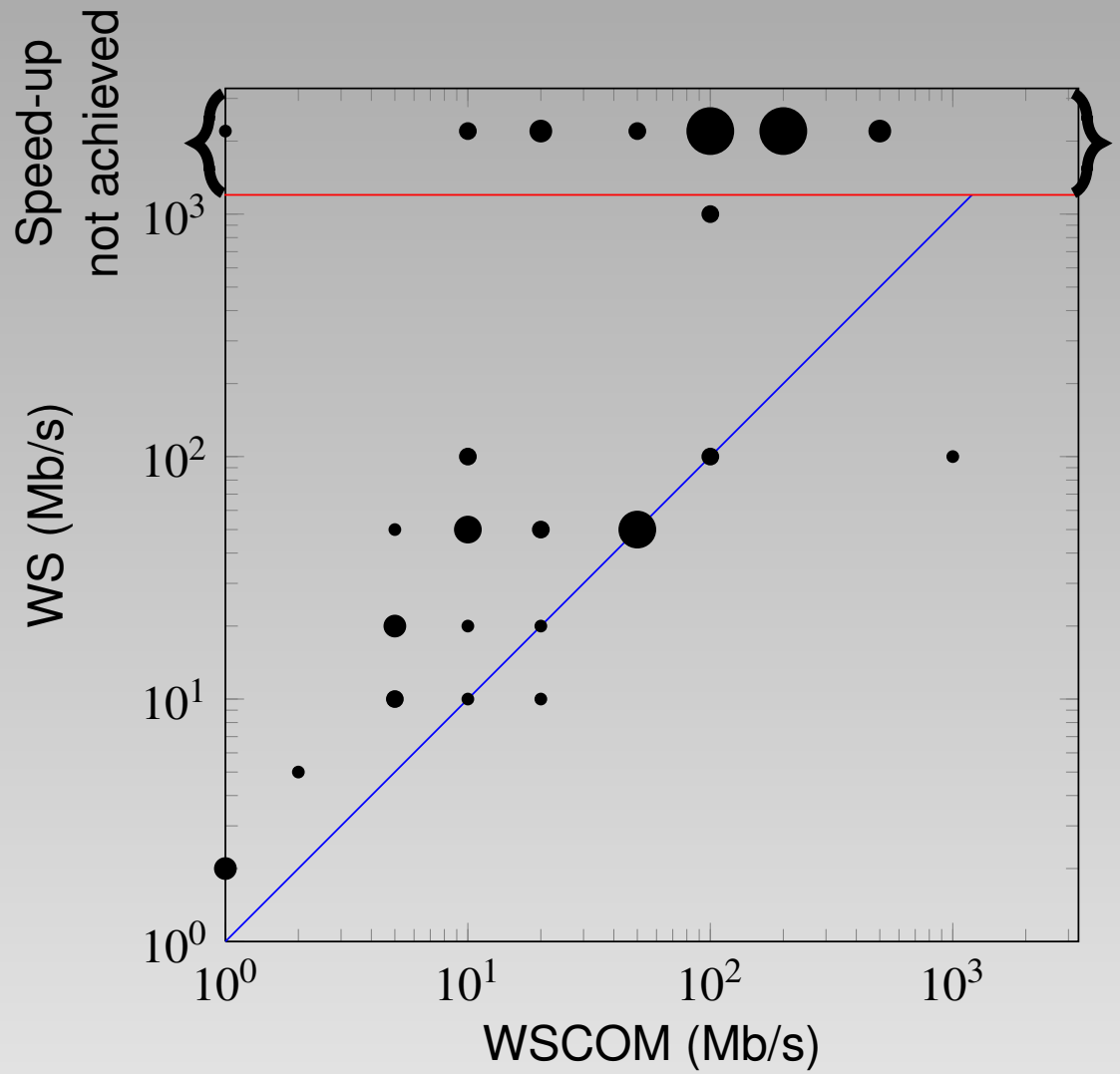
WSCOM vs WS

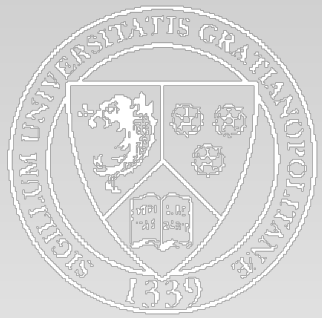




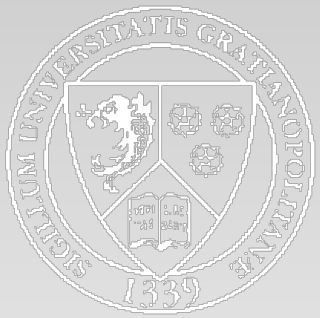
DAG from Makefile Execution

WSCOM vs WS

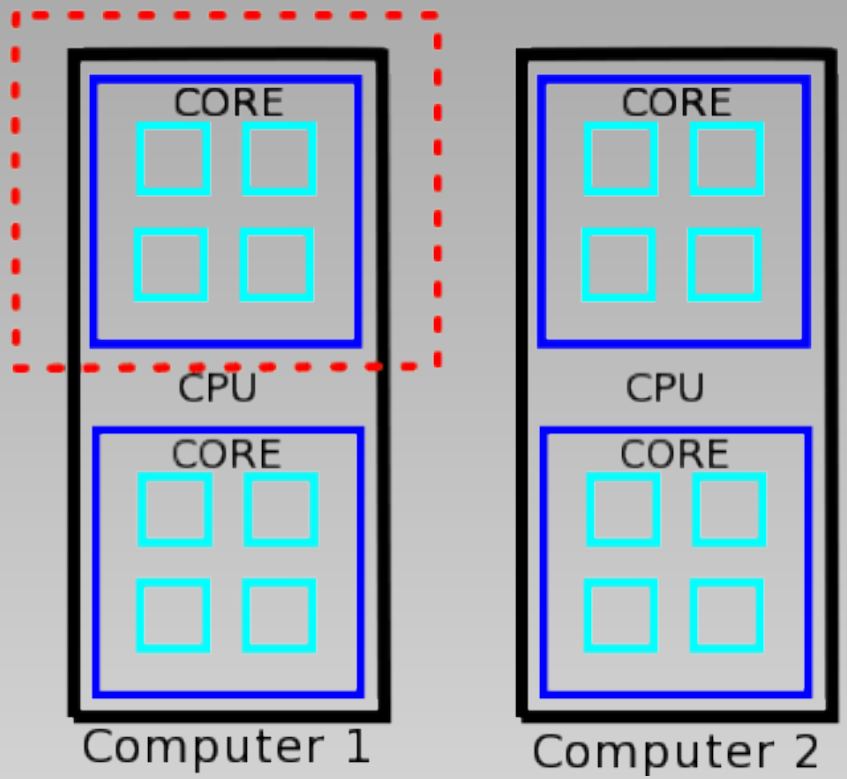


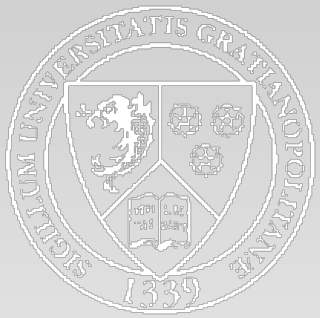


Outline



Modeling Hierarchical Platforms





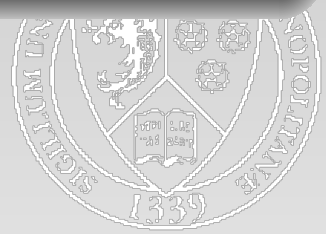
Request Steal Policies

PWS: Probabilistic Work-Stealing

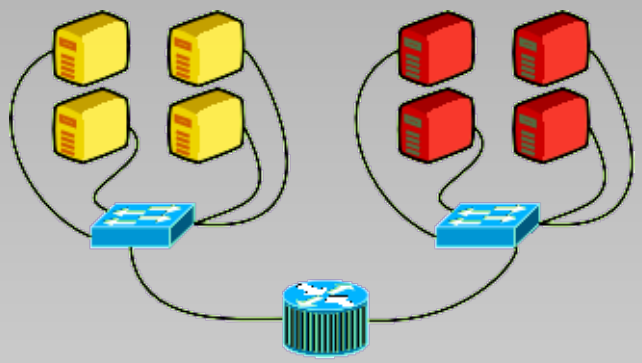
- Main idea: reduce the time to find some tasks.
- Steal probability depends on the distance between computers.
- No limit on hierarchy level.

HWS: Hierarchical Work-Stealing

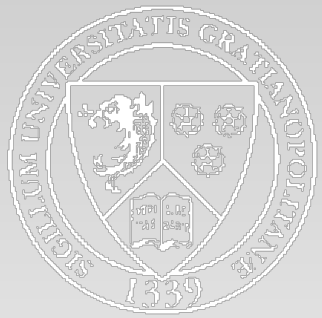
- Main idea: reduce the amount of data transfered between groups.



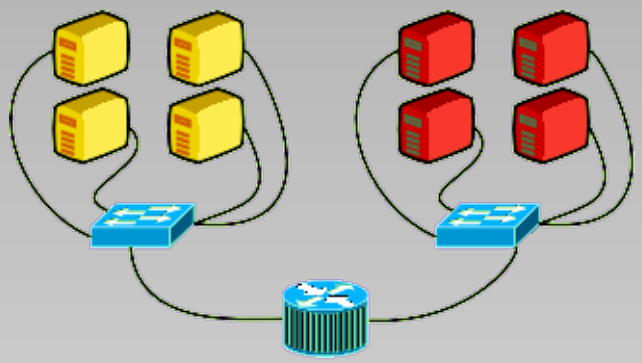
HWS: Reduce Expensive Steal Requests



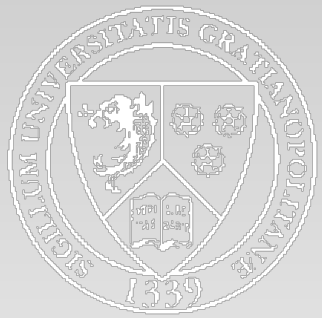
- Reduce the number of steal requests between clusters.
- Keep a fair load-balancing.



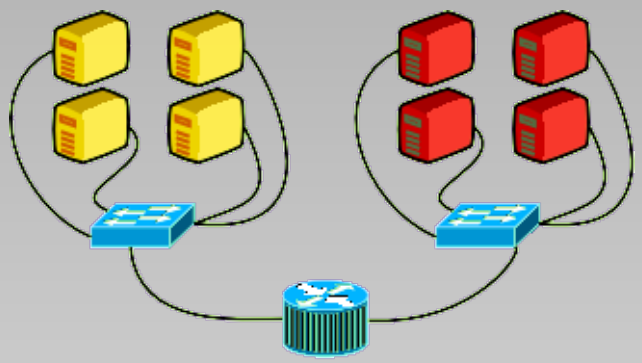
HWS: Reduce Expensive Steal Requests



- Reduce the number of steal requests between clusters.
- Keep a fair load-balancing.
  $\Rightarrow$ Steal greater amount of work.
  $\Rightarrow$ Avoid stealing tasks with few amount of work.

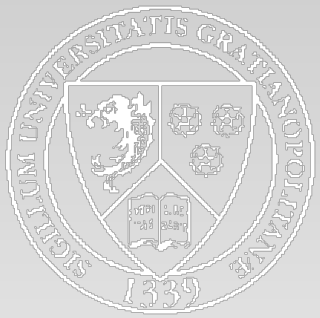


HWS: Reduce Expensive Steal Requests

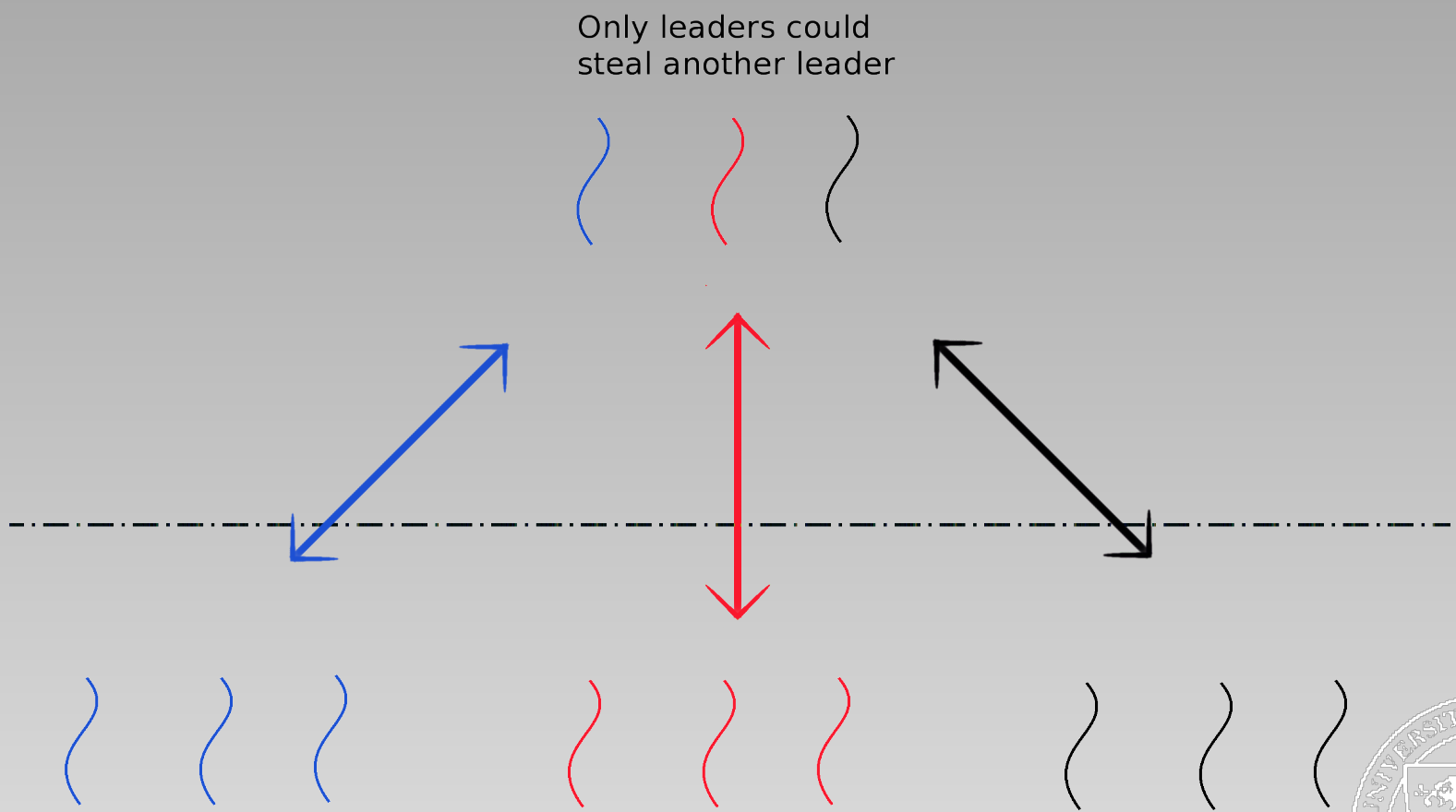


- Reduce the number of steal requests between clusters.
- Keep a fair load-balancing.
  $\Rightarrow$ Steal greater amount of work.
  $\Rightarrow$ Avoid stealing tasks with few amount of work.

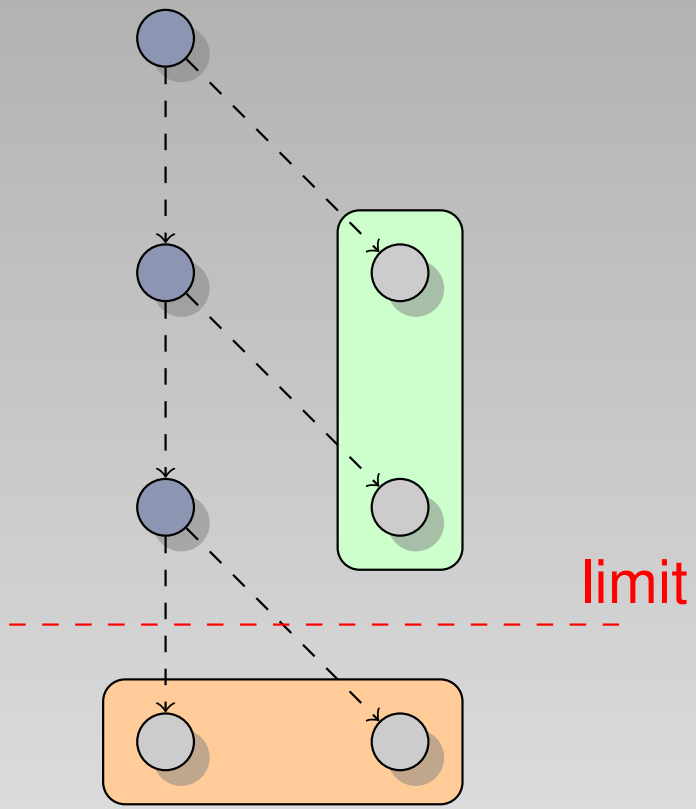
Only one computer steal other clusters.



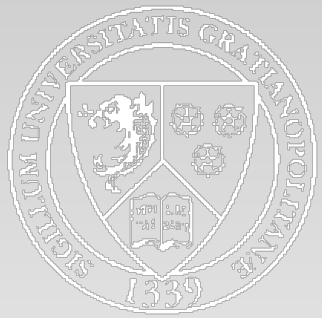
Hierarchical Work-Stealing



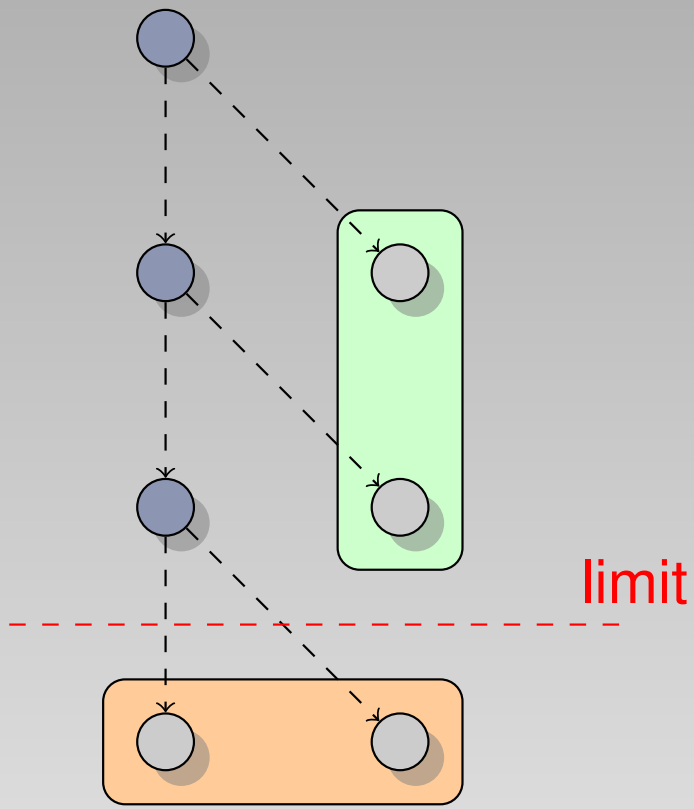
Hierarchical Work-Stealing



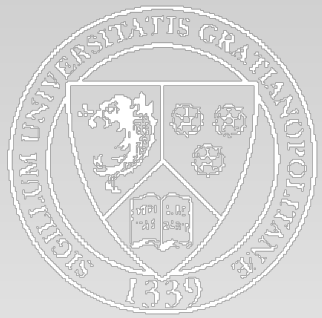
- Execute tasks in the same order as the classical work-stealing.



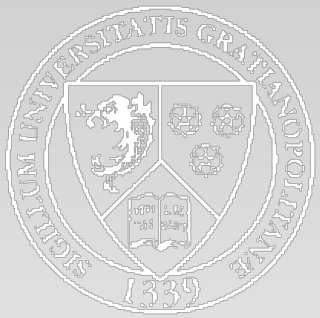
Hierarchical Work-Stealing



- Manage a second tasks stack restricted to its cluster.
- The Leader waits the end of block execution.

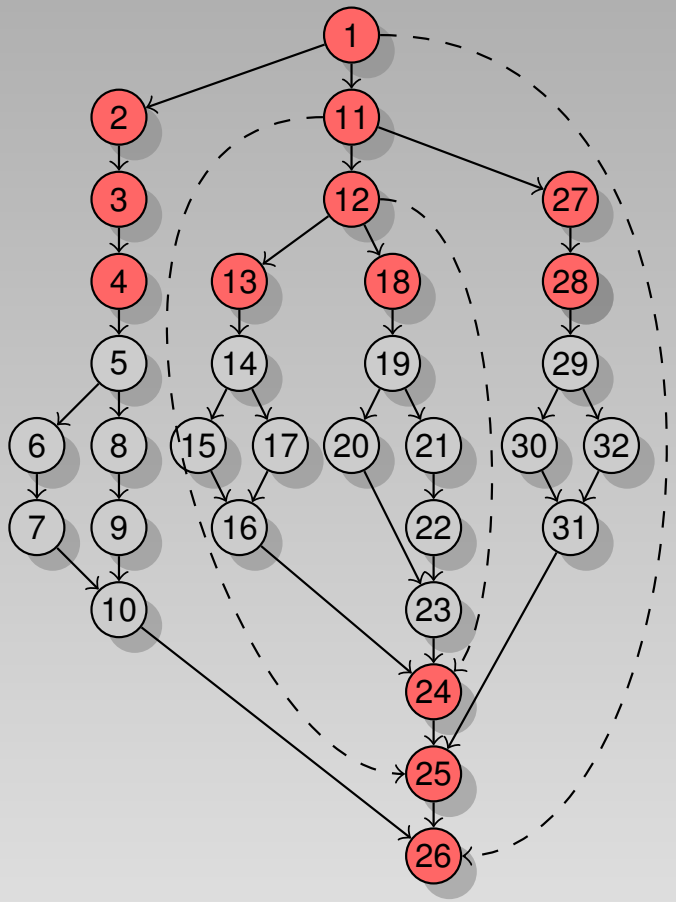


Outline

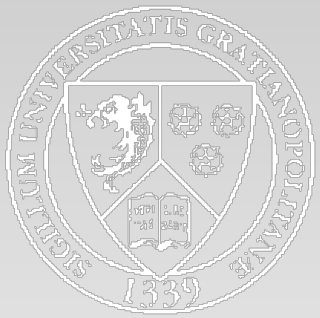


Execution Time Bound Analysis

$$\frac{W_u}{p_u} + \frac{W_d}{p} + O(D\frac{B}{p_u}) + O(D) + O(\max_i(\frac{W_i}{p_d}))$$

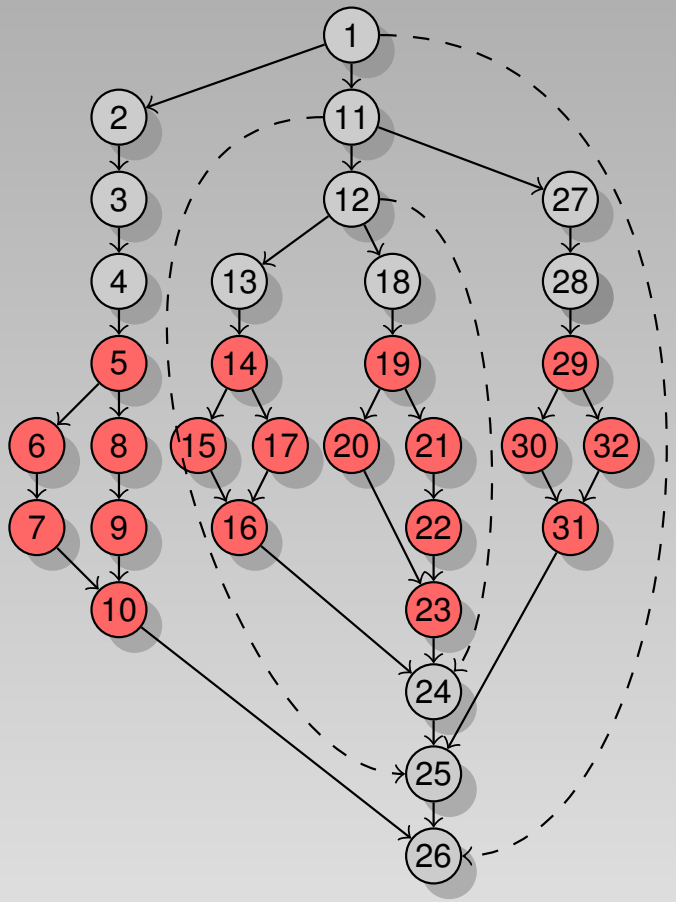


- Execution time depends on the work done by leaders.

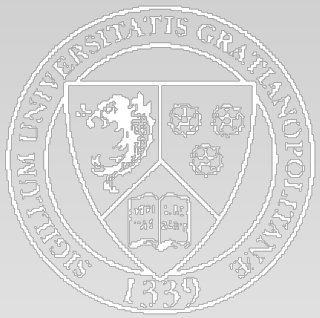


Execution Time Bound Analysis

$$\frac{W_u}{p_u} + \frac{W_d}{p} + O(D\frac{B}{p_u}) + O(D) + O(\max_i(\frac{W_i}{p_d}))$$

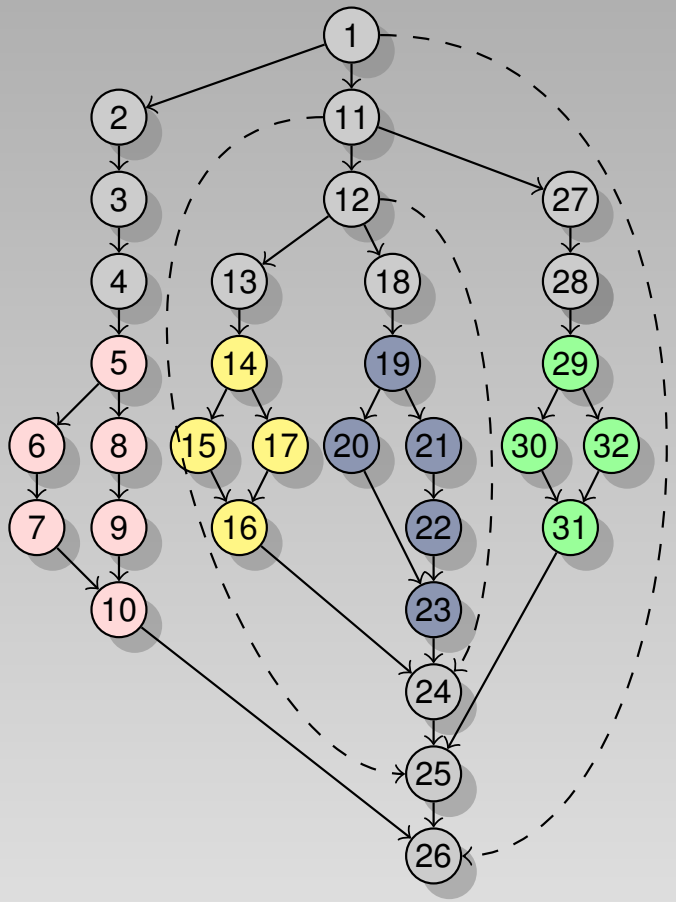


- The load is balanced between all processors.

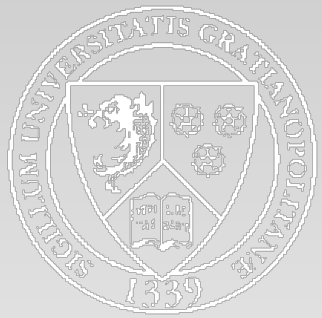


Execution Time Bound Analysis

$$\frac{W_u}{p_u} + \frac{W_d}{p} + O(D\frac{B}{p_u}) + O(D) + O(\max_i(\frac{W_i}{p_d}))$$

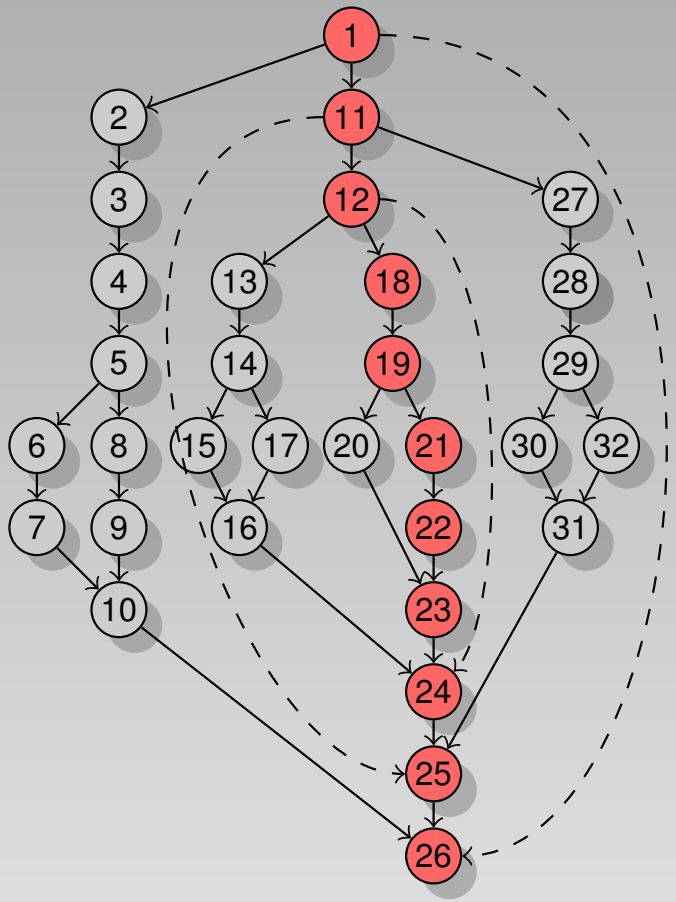


- Theoretical model limit.
- Only one block on a cluster at the same time.

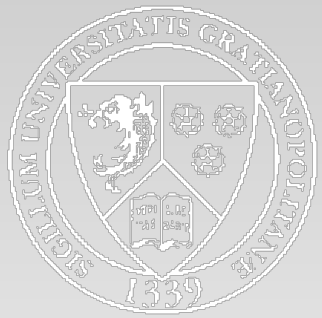


Execution Time Bound Analysis

$$\frac{W_u}{p_u} + \frac{W_d}{p} + O(D\frac{B}{p_u}) + O(D) + O(\max_i(\frac{W_i}{p_d}))$$

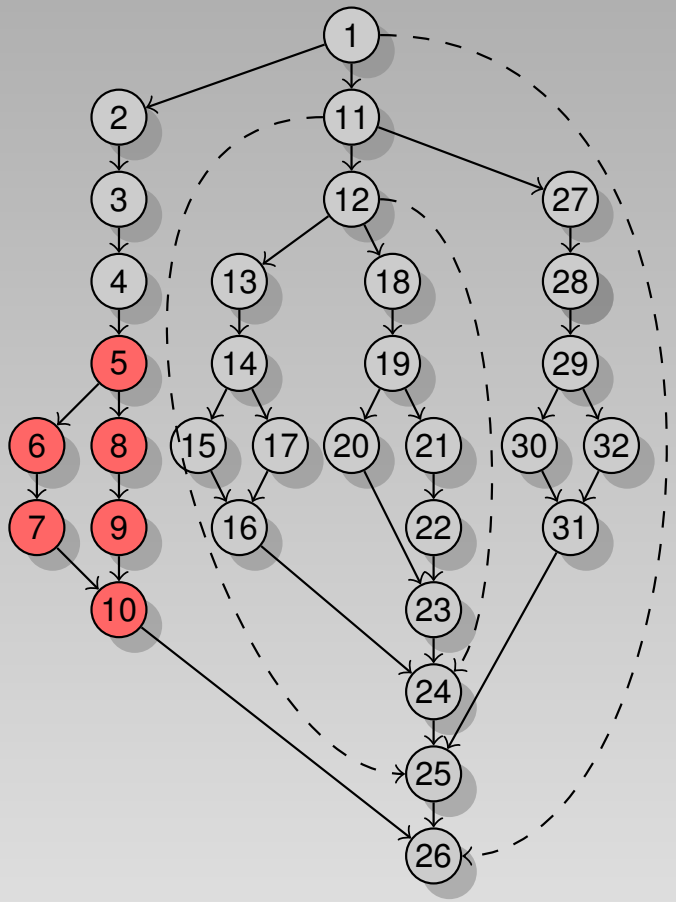


- Classical Work-Stealing.

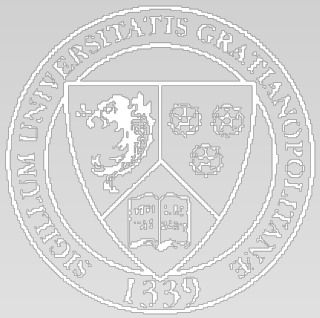


Execution Time Bound Analysis

$$\frac{W_u}{p_u} + \frac{W_d}{p} + O(D\frac{B}{p_u}) + O(D) + O(\max_i(\frac{W_i}{p_d}))$$

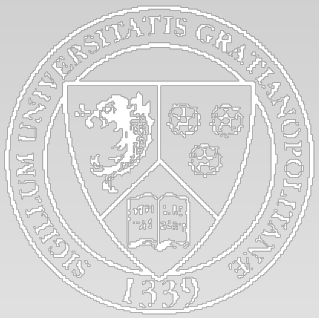


- The work inside a cluster cannot be stolen by another cluster.
- The largest block must be executed by one cluster.



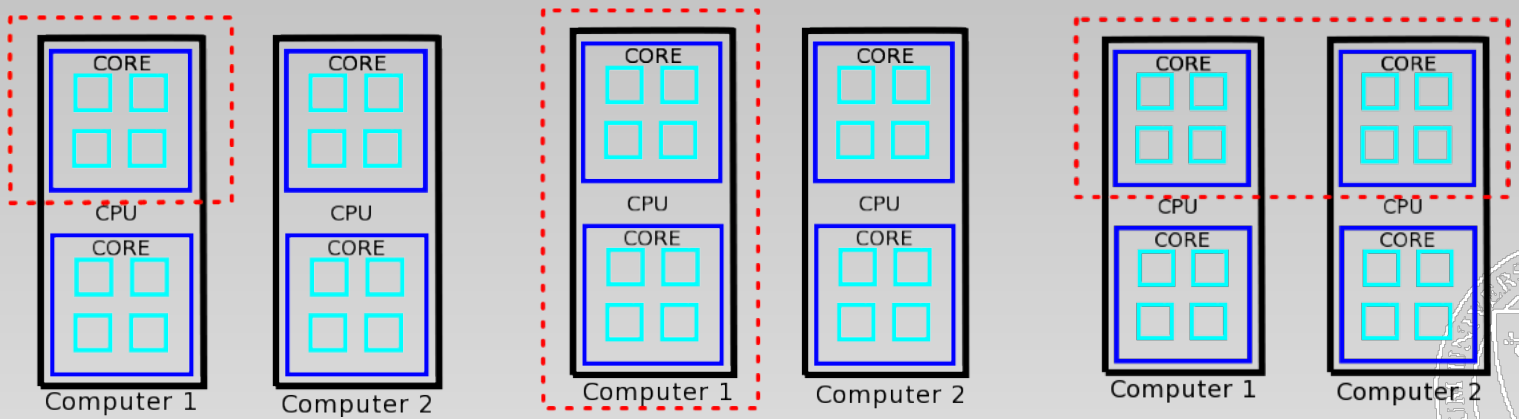
Conclusion of the Theoretical Analysis

- The load seems to be balanced on the whole platform.
- The number of steal request between clusters:
  - Classical Work-Stealing : $O((p - \min p_i)D)$.
  - HWS : $O(p_u \times (D + \frac{\max_i W_i}{p_d}))$.

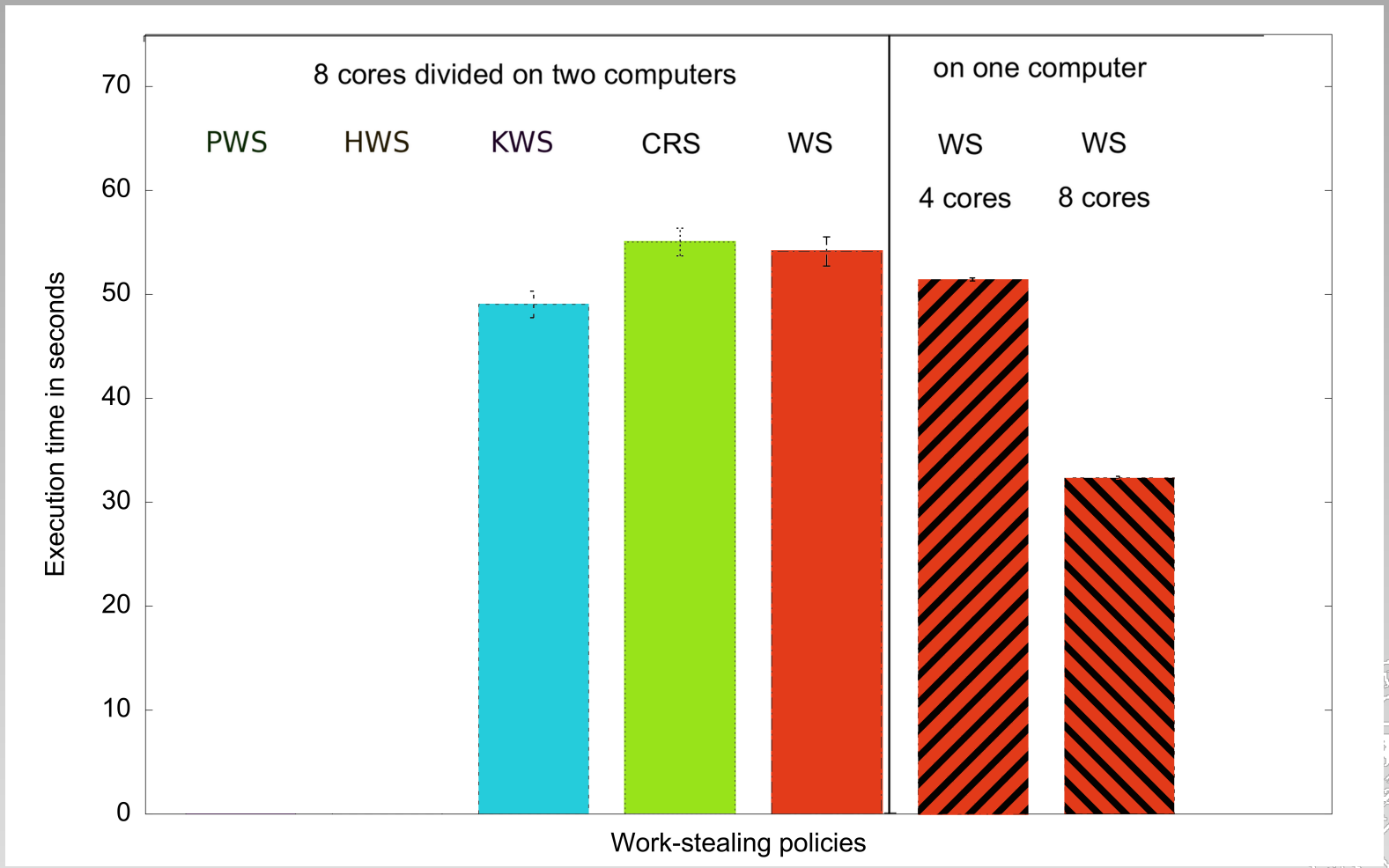


Executive Platform and Application

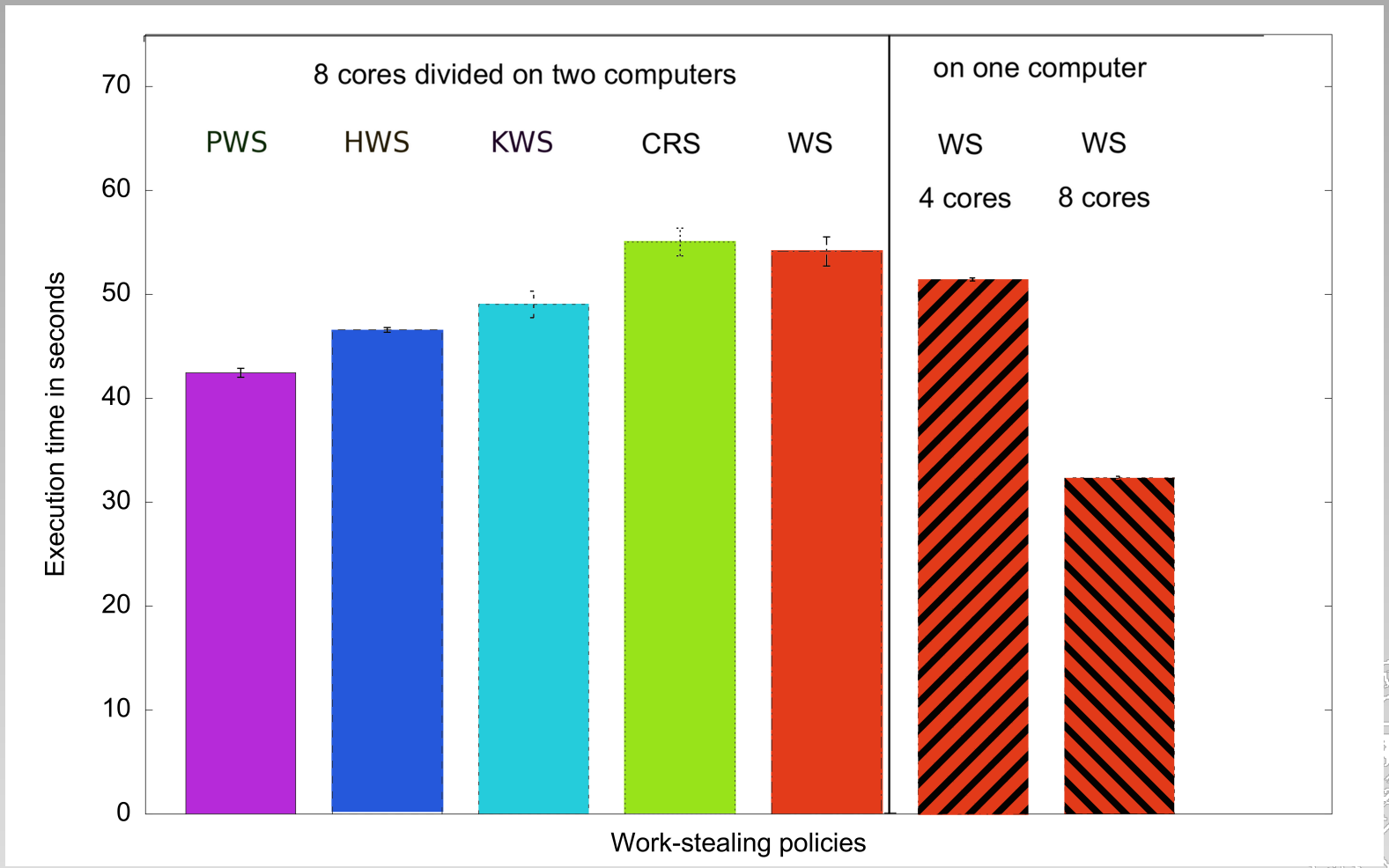
- Genepi cluster (Grid'5000).
  - Two quad-core processors
  - High performance network: Infiniband.
- Merge-Sort.
  - Array of doubles.
  - 4 GBytes of data.



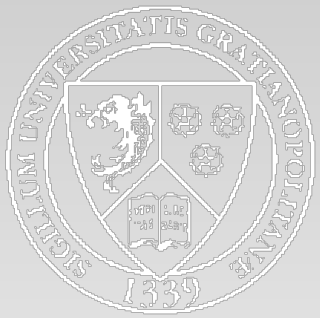
Merge-Sort with Hierarchical Heuristics



Merge-Sort with HWS and PWS



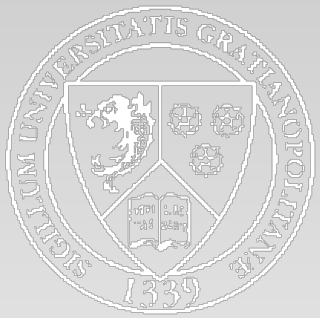
Outline



Conclusion

Extend the Work-Stealing Utilization:

- For specific applications.
- For hierarchical platforms.



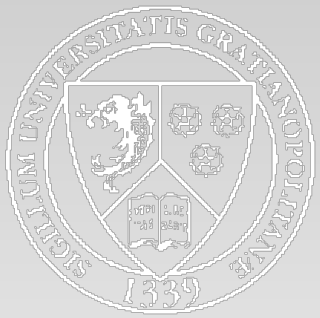
Conclusion

Extend the Work-Stealing Utilization:

- For specific applications.
- For hierarchical platforms.

Additional Information:

- DAG structure.
- Platform structure.



Conclusion

Extend the Work-Stealing Utilization:

- For specific applications.
- For hierarchical platforms.

Additional Information:

- DAG structure.
- Platform structure.

Algorithms:

- information vs performances
- practical problems vs theory

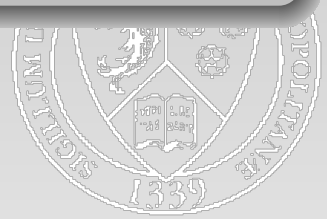
Perspectives

For WSCOM:

- Improve the pre-fetching (WSCOM_{pf}).
- Experiments WSCOM inside DSMake.
- Propose a new DAG generator.
- Parallelize the compilation of a Linux distribution.

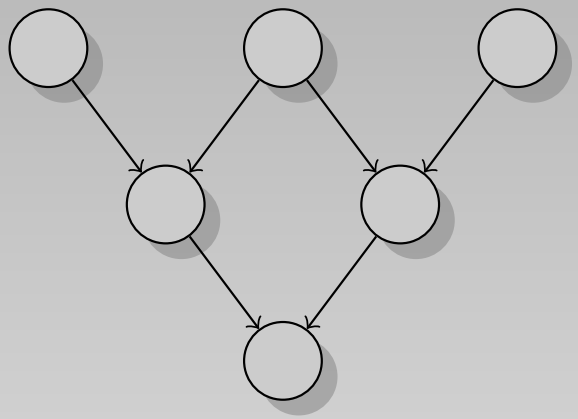
For hierarchical platforms:

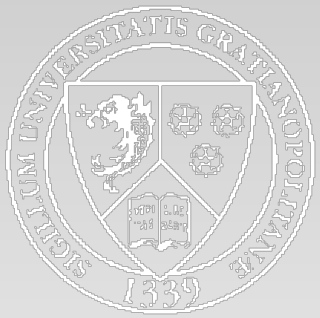
- Extend theoretical results
- Handle heterogeneous platforms.
- Automatically manage the limit inside HWS.



General DAG

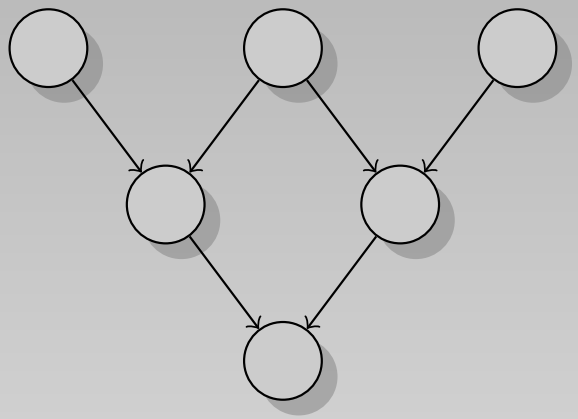
Problem

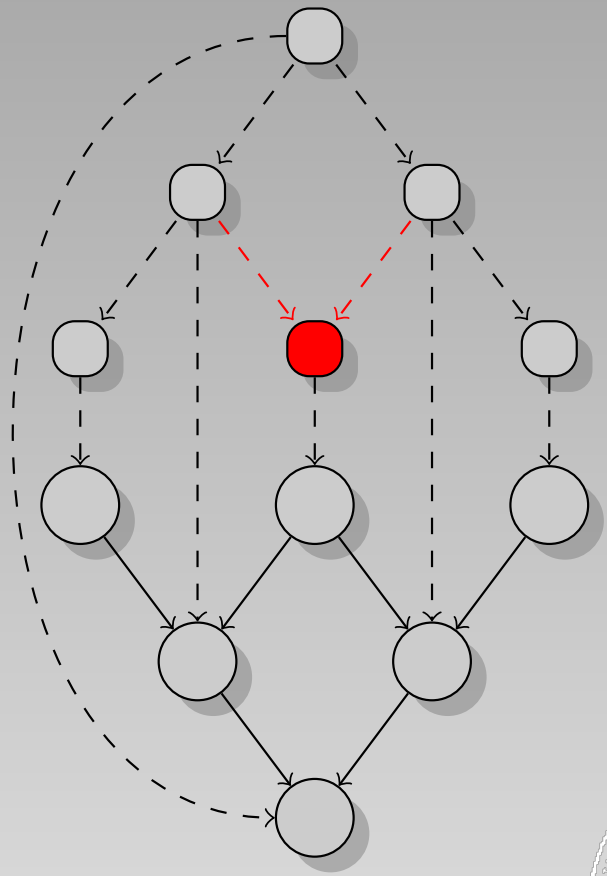


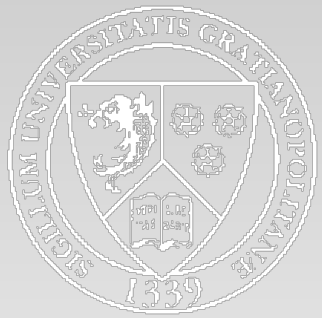


General DAG

Problem

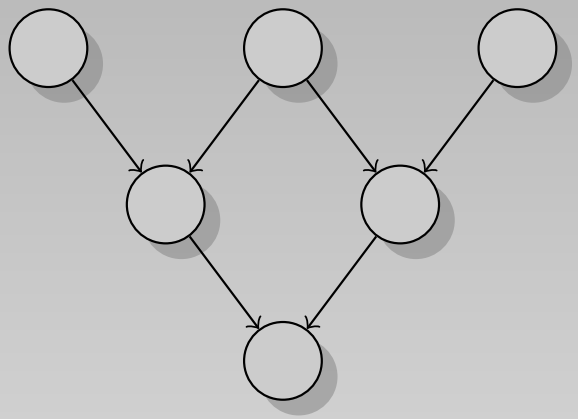


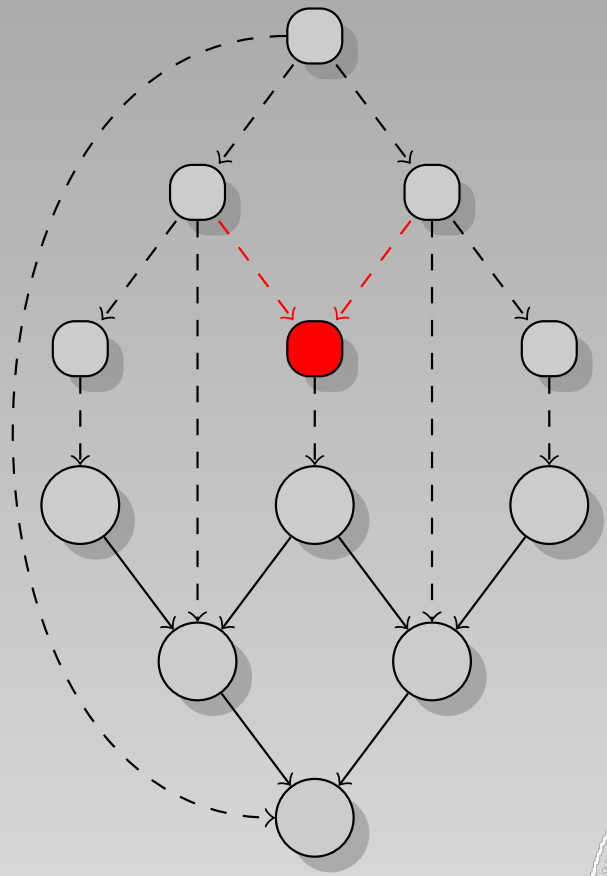




General DAG

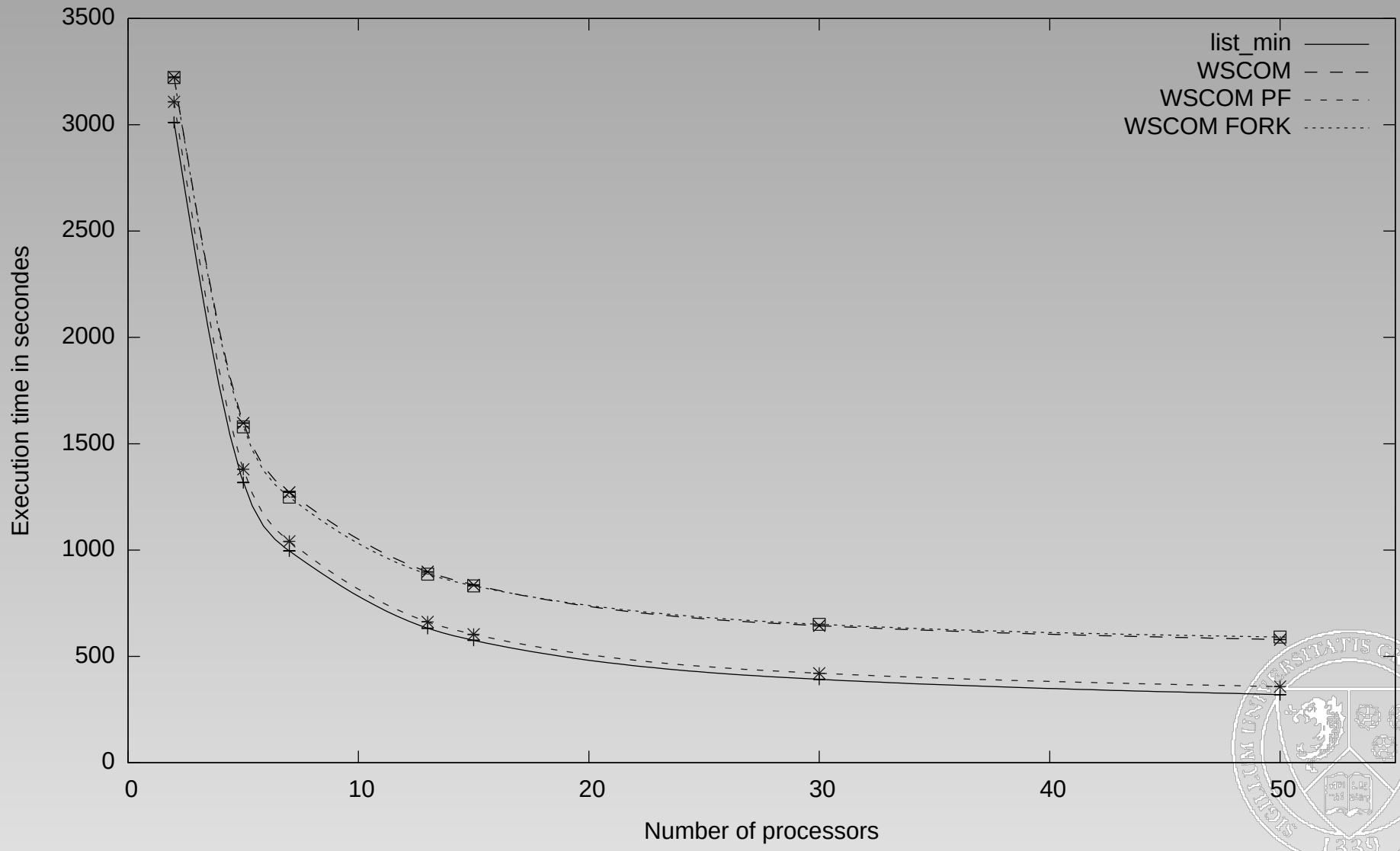
Problem





- Resolved before the execution: a spanning tree.
- Resolved during the execution: FIFO.

WSCOM$_{fork}$


3500
3000
2500
2000
1500
1000
500
0
0
10
20
30
40
50
Execution time in secondes
Number of processors
list_min
WSCOM
WSCOM PF
WSCOM FORK