

# Scheduling Large Jobs by Abstraction Refinement

Vasu Singh

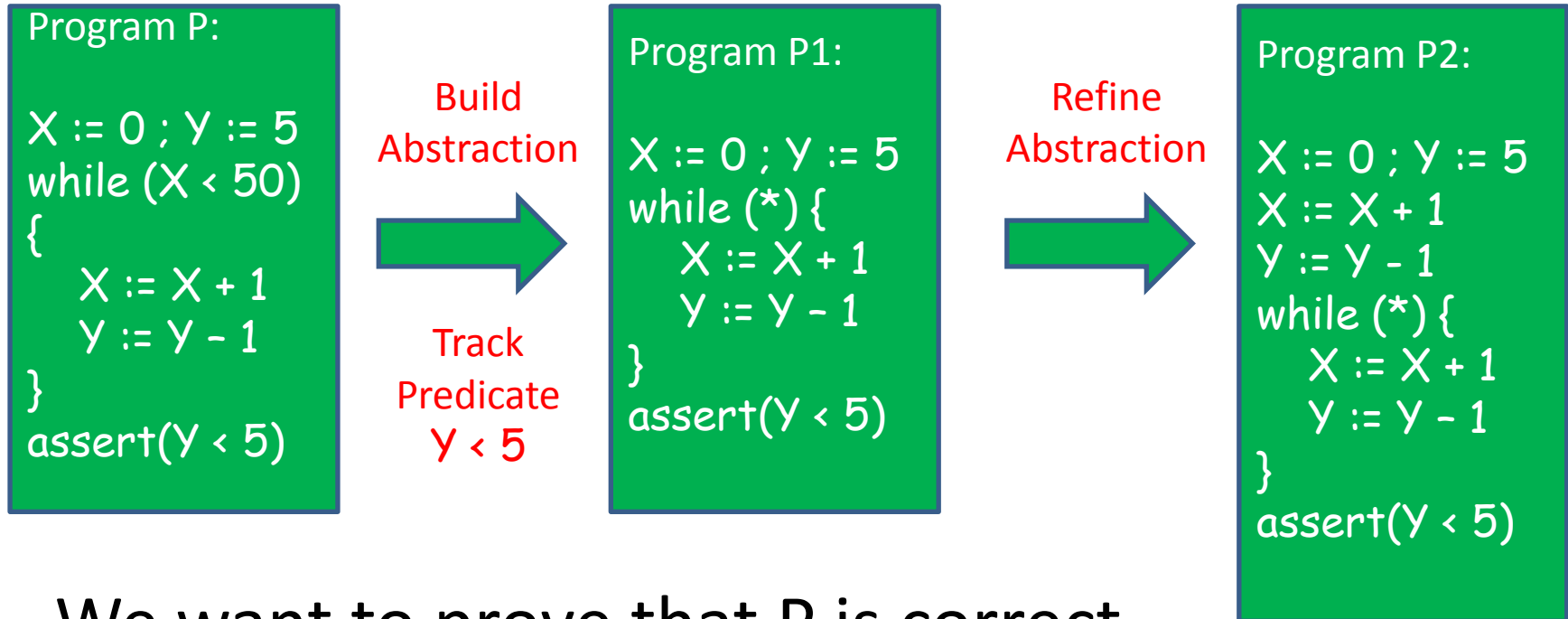
Joint work with Thomas Henzinger,  
Thomas Wies, Damien Zufferey

IST AUSTRIA

# A Word on Our Background

- Formal Verification Community
- My work: Formal Verification of Concurrent Programs and Distributed Systems
- In general, formal verification is **undecidable**. In many relevant cases, it is computationally hard. We develop techniques that make verification tractable.

# A View into Our World



We want to prove that P is correct.

First approach: Run the whole program **concretely**.

Second approach: Use **abstraction refinement**!

# In general

- What is an abstraction?
  - A **concise** representation of a system
  - Rely on **over-approximations** or **under-approximations** of the behavior of the system
  - A good abstraction loses a lot of irrelevant information and little relevant information
  - What is **relevance**? Depends on what property we are looking for!

# In general

- Why do we use abstractions?
  - They often allow fast efficient solutions where concrete solutions are tedious, or even infeasible.
  - If an abstraction is too coarse for some purpose, one always has a possibility to refine it closer to the real system

# We abstract all the time!

- The idea is not limited to formal verification community!

# In daily life

A: Mr. X has 3 fast cars !

[ABSTRACTION]

B: Which ones?

A: Aston Martin, Lamborghini, Ferrari!

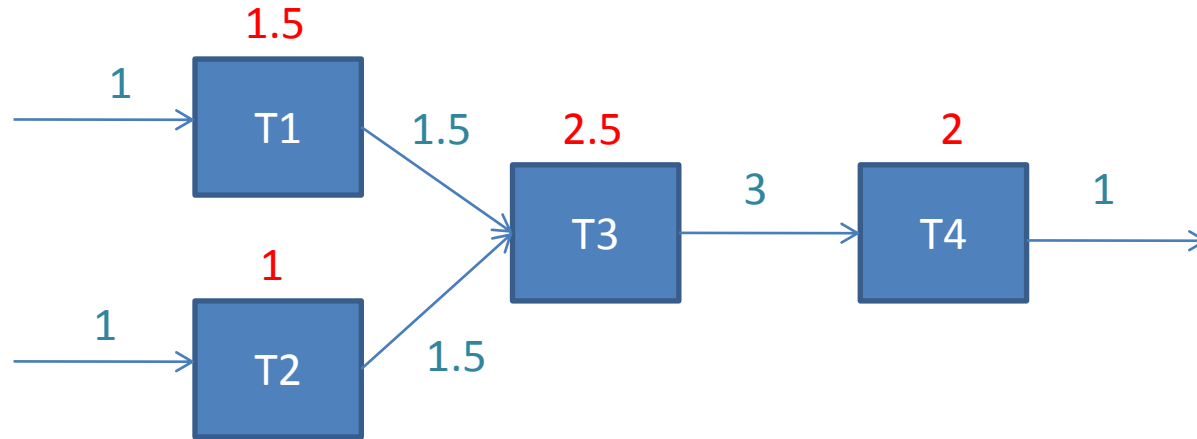
[REFINEMENT]

# In technology

- Image and video compression
- Program analysis
- Machine learning (classification)
- In general, whenever the concrete system is too big to handle!

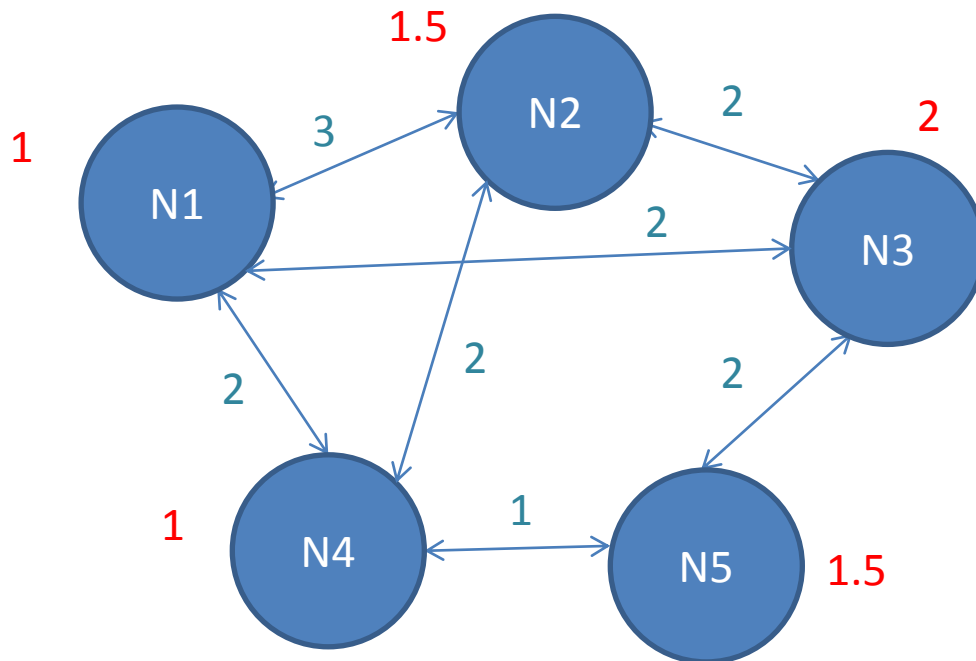
# **THE CLOUD SCHEDULING PROBLEM**

# Job



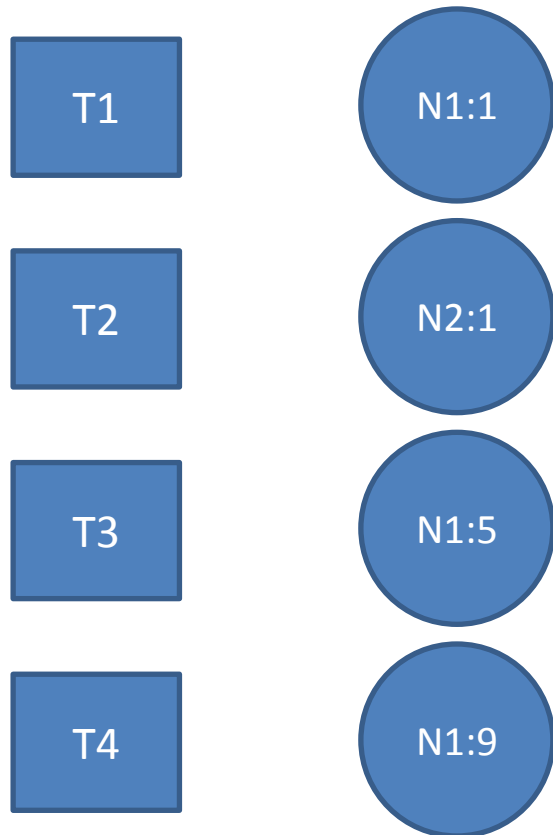
- A directed acyclic graph (DAG) of tasks
- Nodes marked with **worst case computing duration**
- Edges marked with **data transfer**
- These can be estimated for a large class of jobs in NLP, machine learning, image processing, bioinformatics (parametrized by input size)

# Cloud



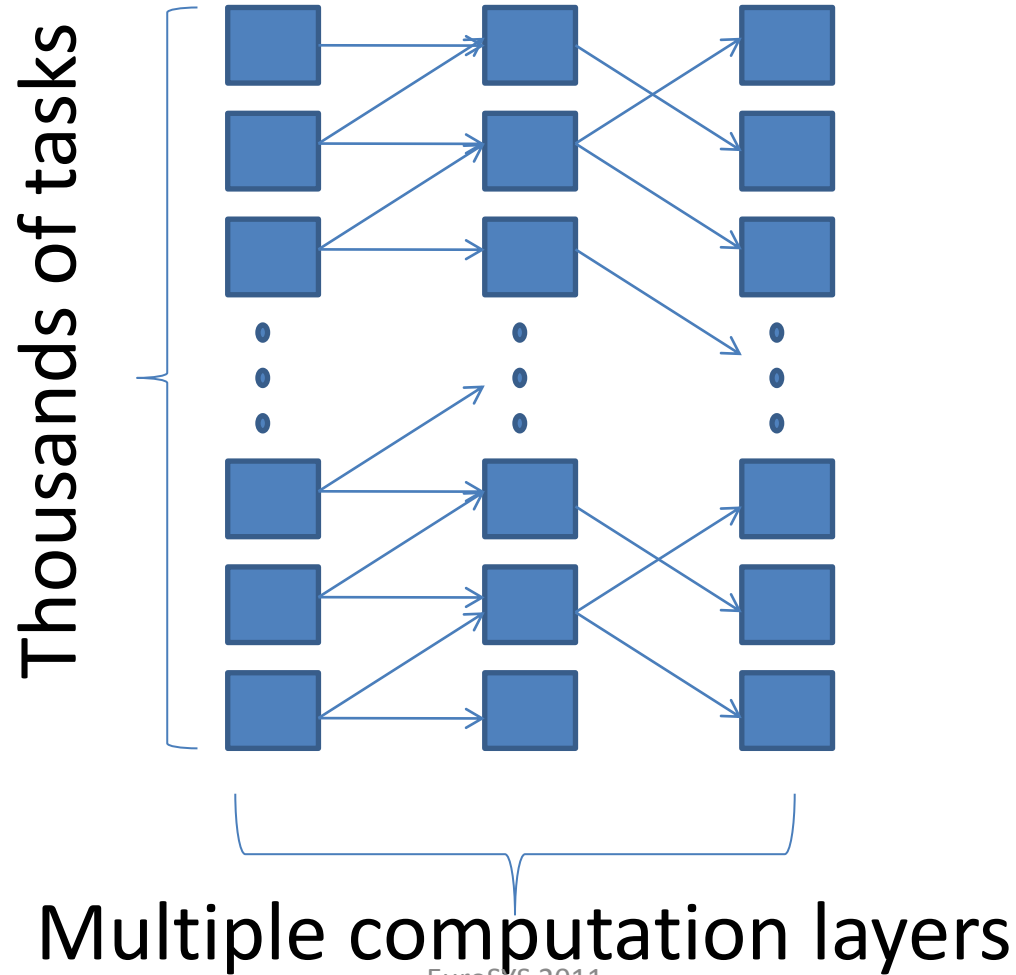
- A connected graph
- Nodes marked with **computation power**
- Edges marked with **link bandwidth**

# Schedule

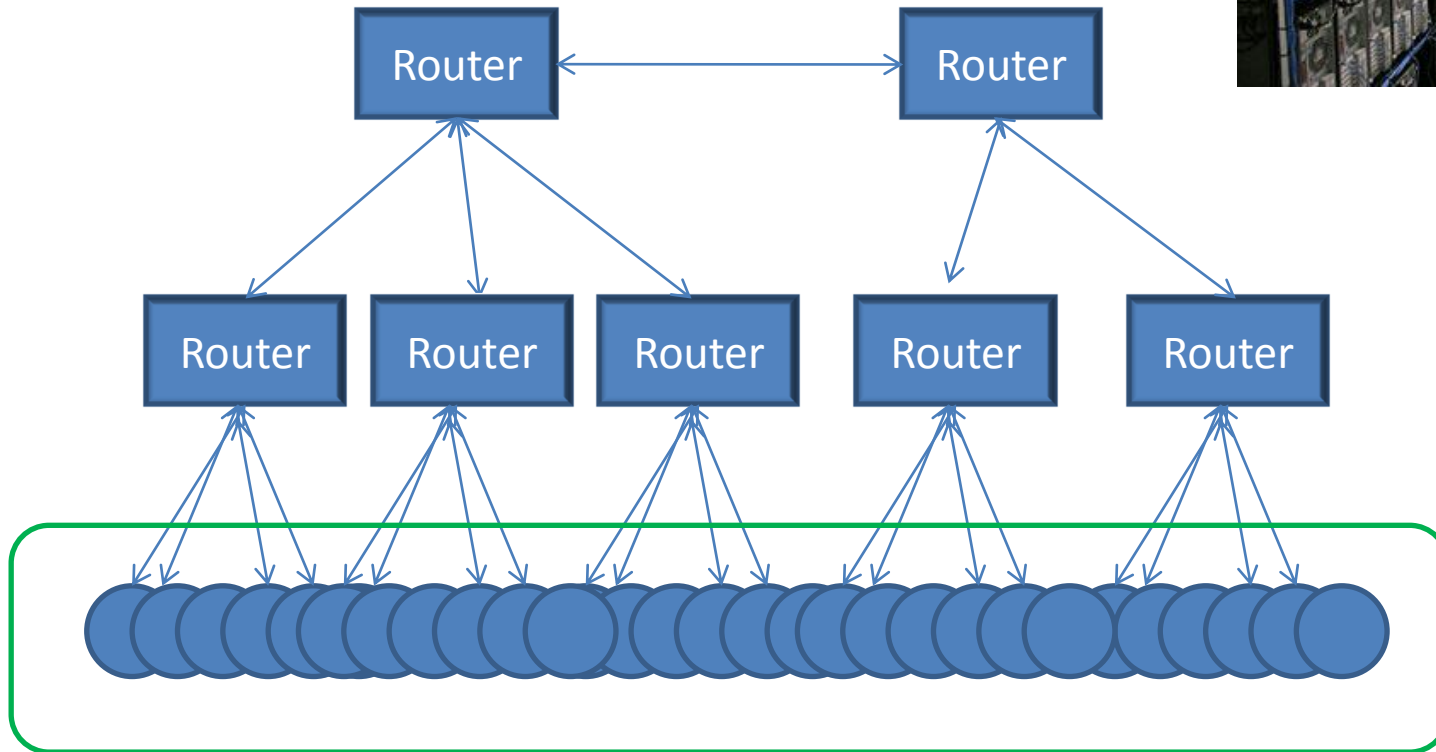


- A function from tasks to node-start time pairs

# A Large Job



# A Data Center

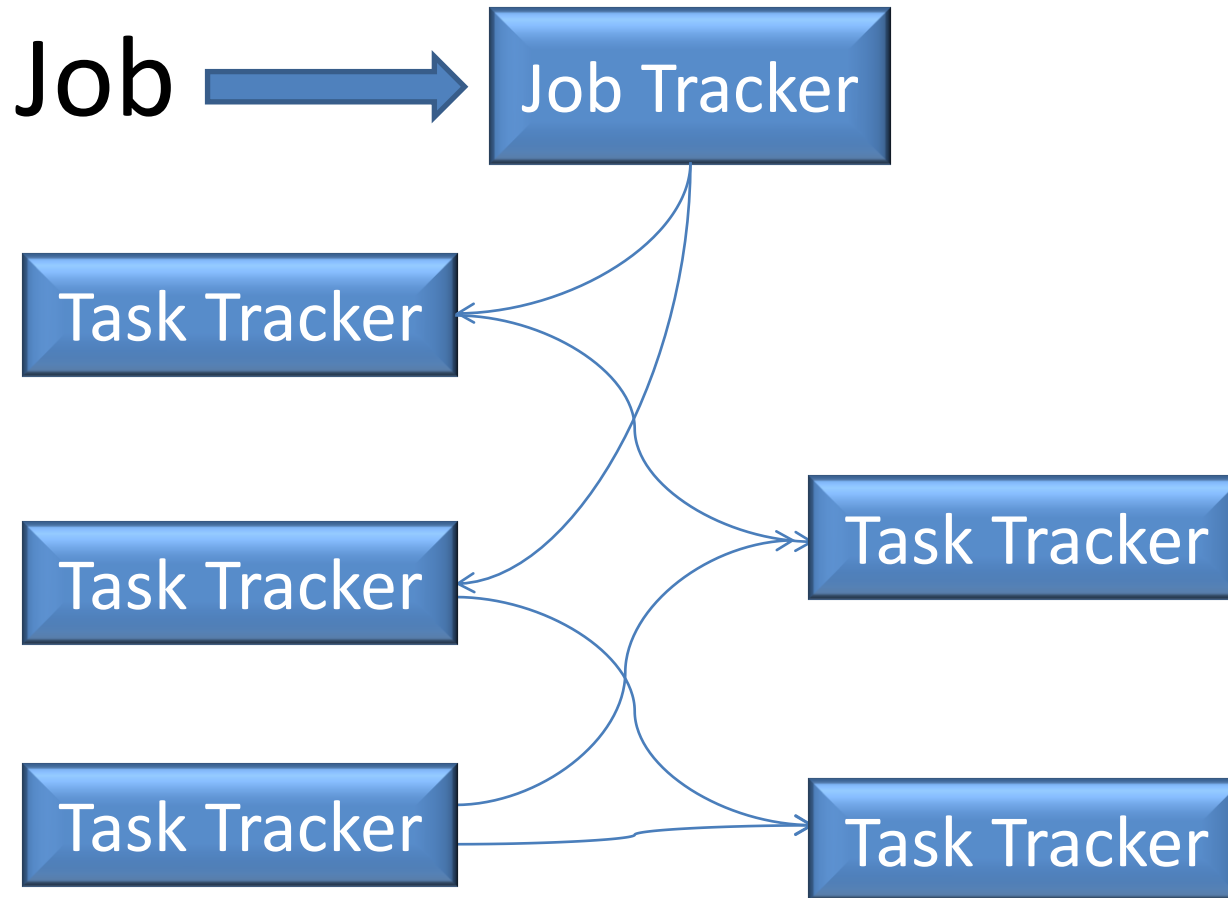


Thousands of Compute Nodes

Given the scale, conventional wisdom  
says:

**Use dynamic scheduling**

# Example: Hadoop



# Hadoop

Dynamic scheduling using Task Queues



Does not allow apriori knowledge of when a job finishes



A user cannot be promised  
a deadline



A cloud cannot plan ahead  
on future  
resource usage

Certainly, if task characteristics are not available, dynamic scheduling is  
the best option!

# Can we do better?

- We have talked a lot about managing data over the past few years
- As computation moves to the cloud, we might want to manage that too!
- Can we plan ahead our computation?

# Static Scheduling

- Static schedule: a schedule computed **before** executing the job
- Benefits:
  - The user can be promised a deadline
  - The resources can be planned
- Drawbacks:
  - Generally computationally expensive

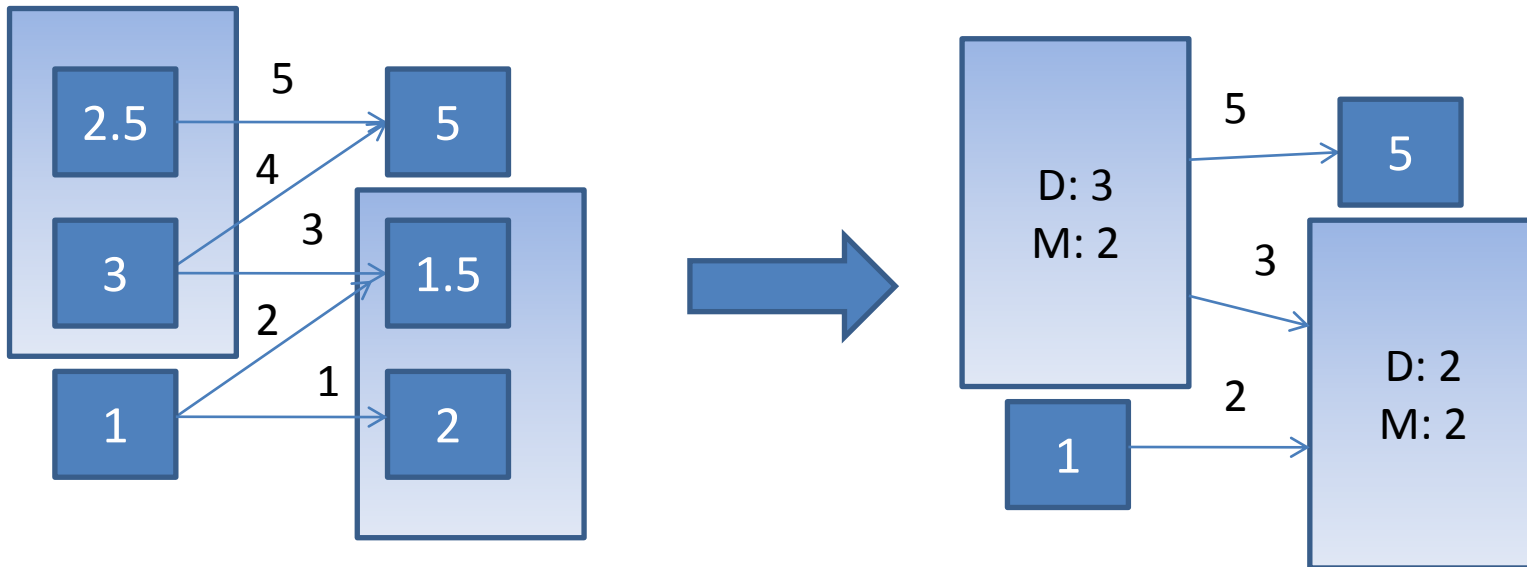
# Static Scheduling

- Computing optimal schedule: NP-hard
- Heuristics (Greedy, deadline division etc.):  
 $|J|.|C|$
- With 1000 tasks job and 200 nodes cloud, a greedy scheduler takes up to 5 minutes!

# The Core Idea

- Over-approximate the resource requirements of the job  $J$  to get  $Abs\ J$
- Under-approximate the computing power of the cloud  $C$  to get  $Abs\ C$
- Get a static schedule for  $(Abs\ J, Abs\ C)$ . Use it as a schedule for  $J, C$

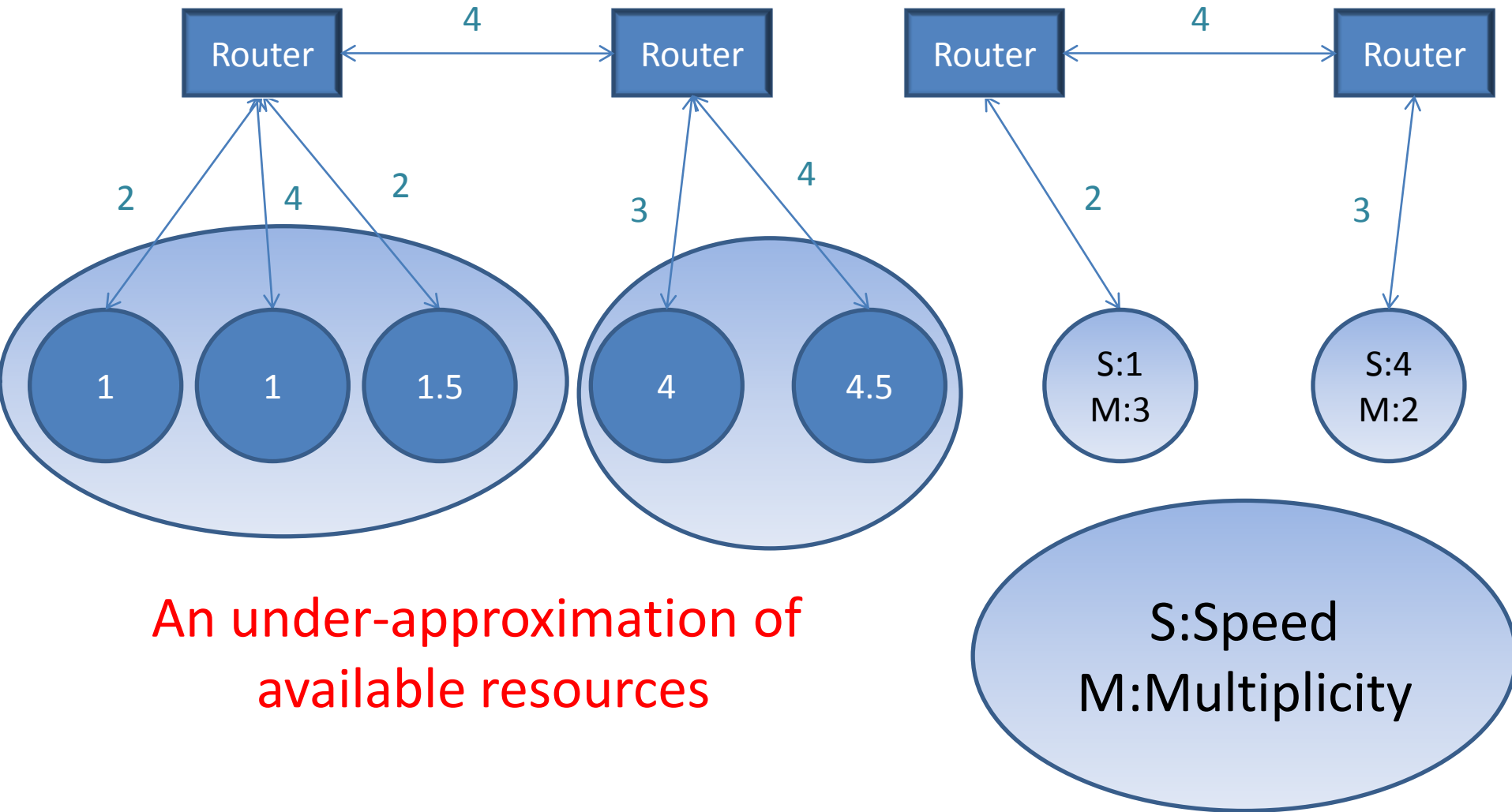
# Job Abstraction



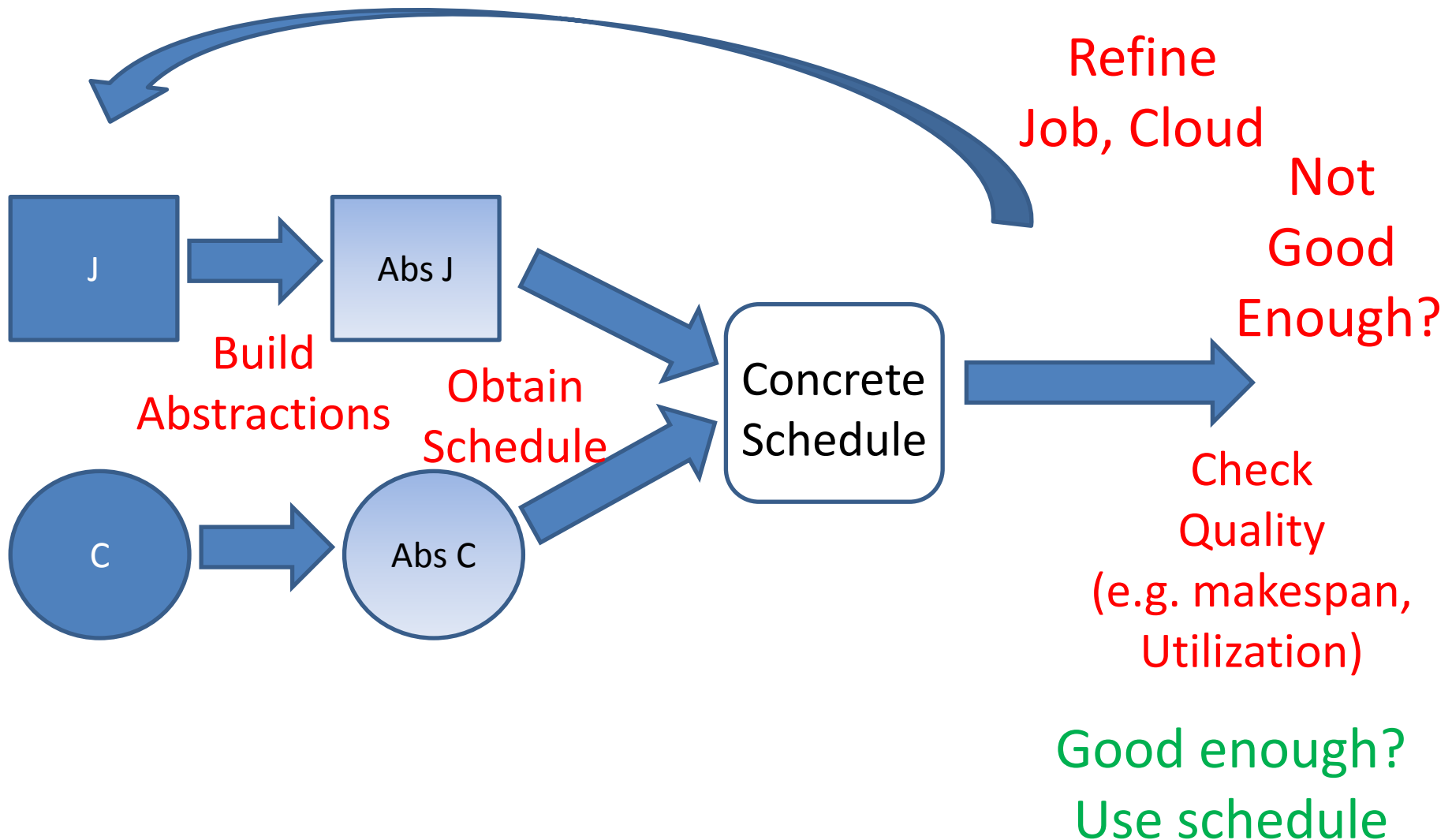
An over-approximation of  
resource requirements

D = Duration  
M = Multiplicity

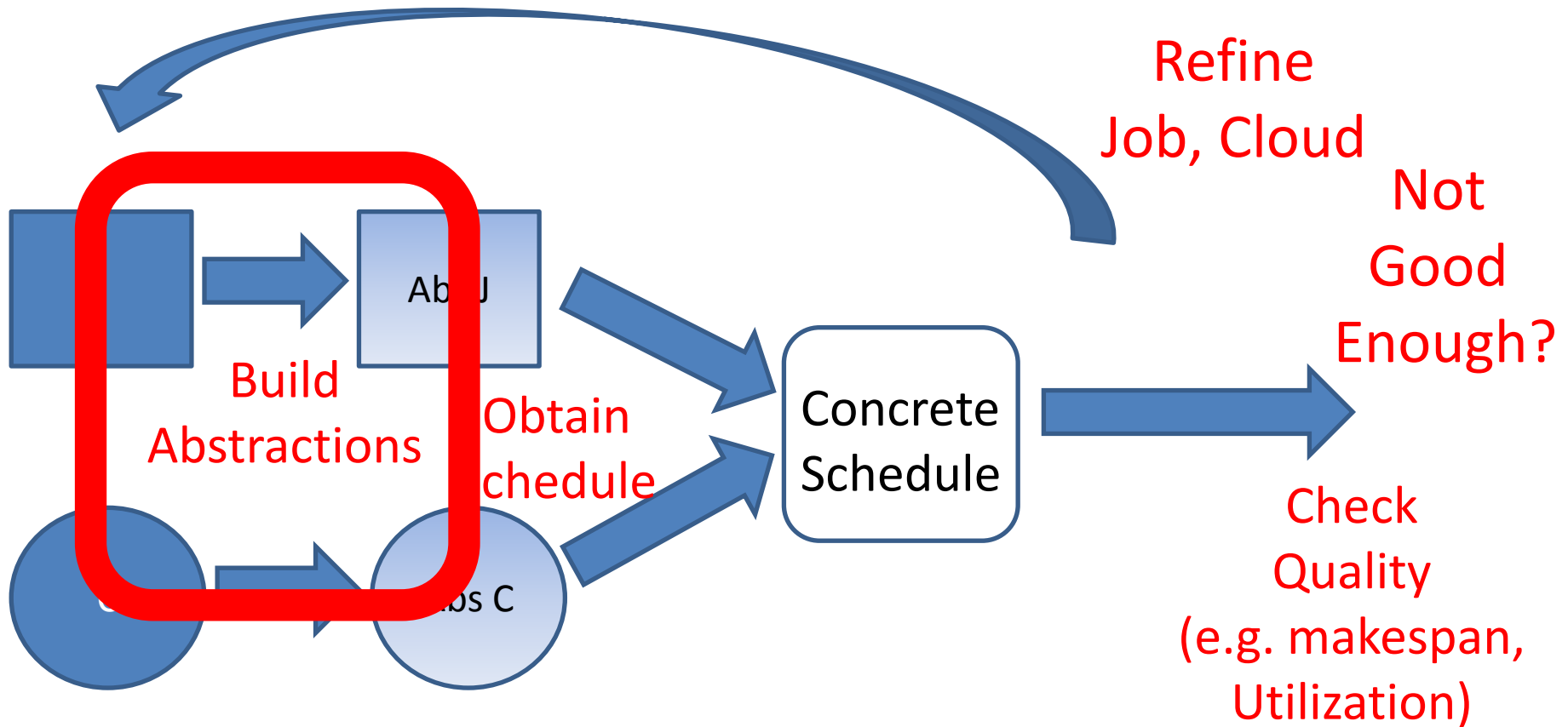
# Data Center Abstraction



# Generic AR Scheduler



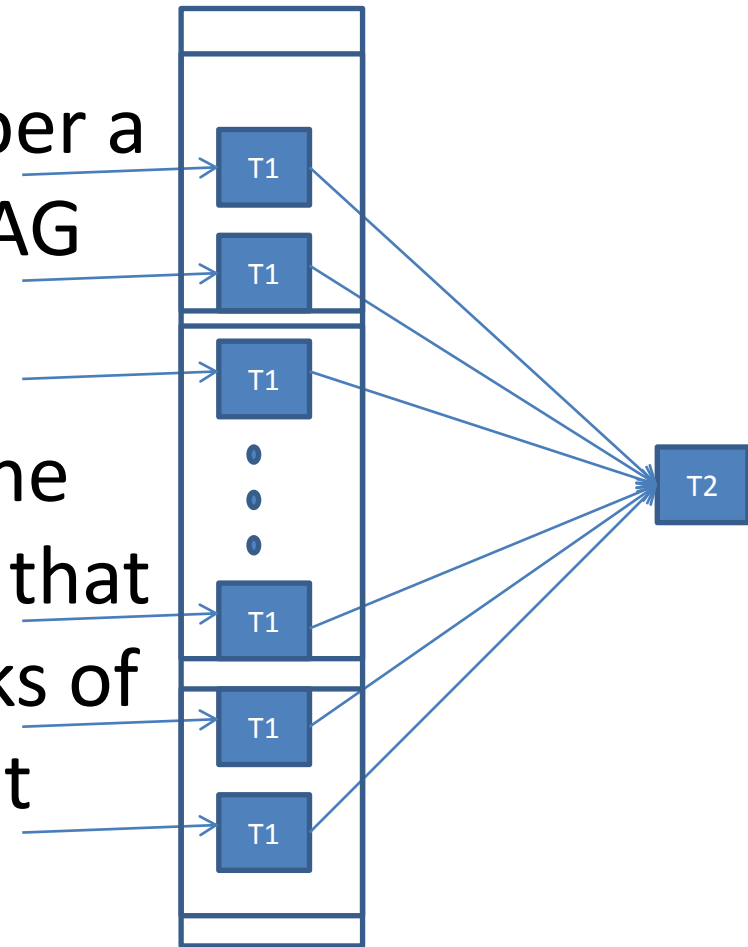
# Generic AR Scheduler



# Important Job Abstractions

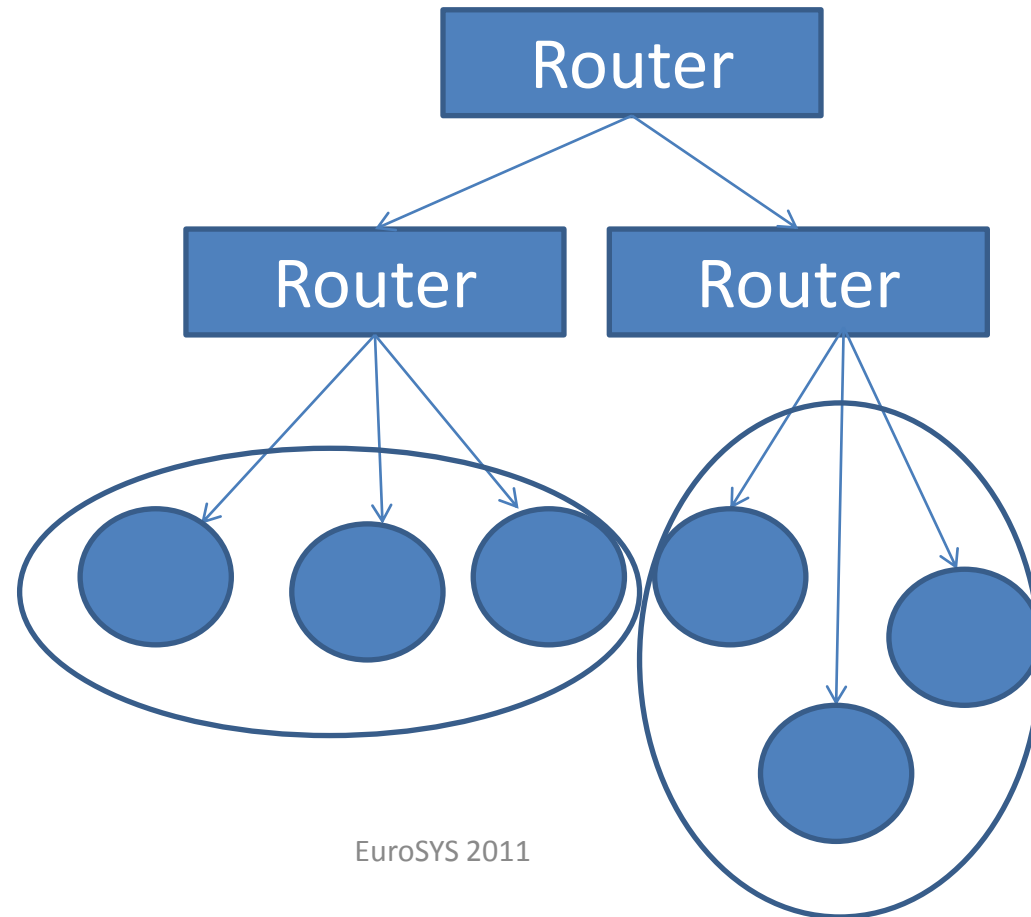
- Topological: Group tasks as per a topological sort of the job DAG
- X-similar: Partition tasks in the topological sort in a manner that no abstract task has two tasks of duration  $D1$  and  $D2$  such that

$$D2 > D1.X$$

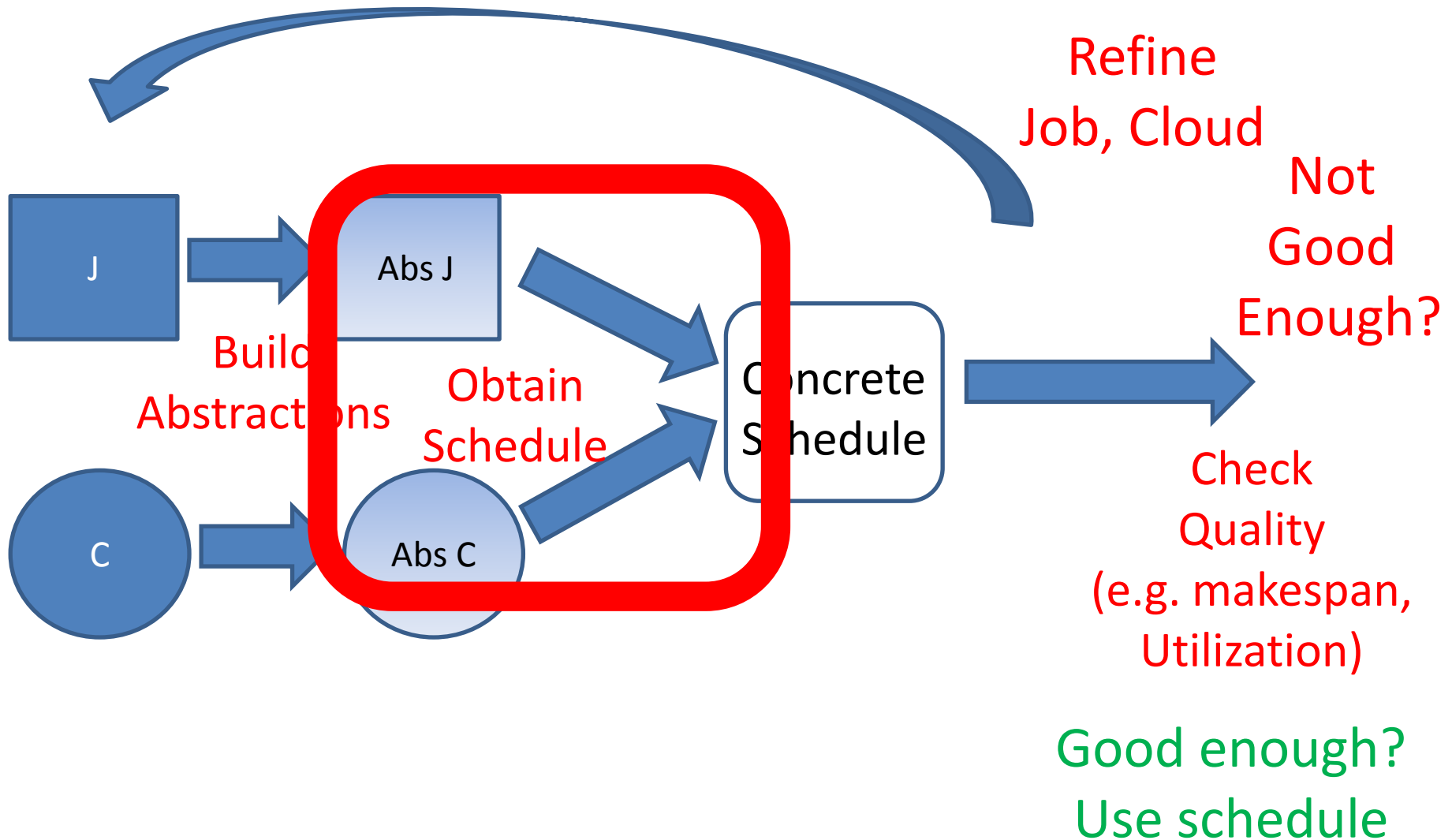


# Important Cloud Abstraction

- Rack abstraction: Create an abstract node for a group of nodes on a rack



# Generic AR Scheduler



# Two AR Schedulers

- FISCH – Free Intervals Scheduler
- BLIND – Buddy Lists IN Datacenters

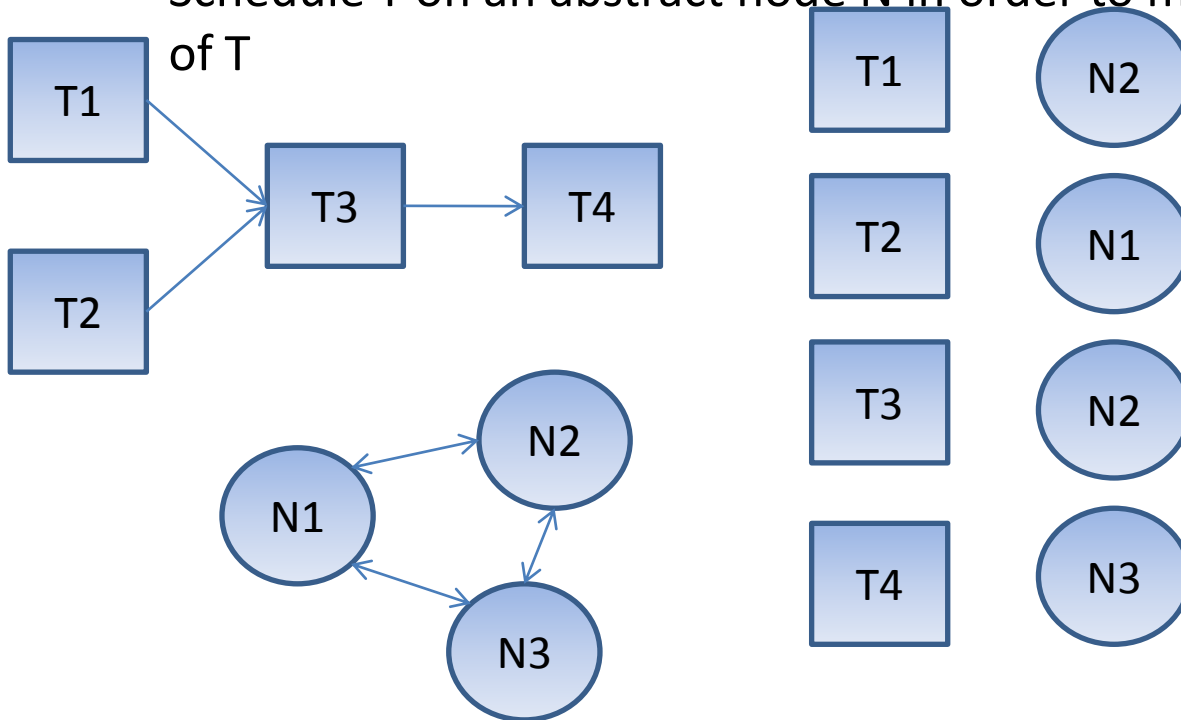
# THE FISCH SCHEDULER

# Abstractions for FISCH

- Starts with topological abstraction of job
- Keeps a constant rack abstraction of cloud

# FISCH: A Greedy Abstract Scheduler

- While an abstract tasks is yet to be scheduled :
  - Choose an abstract task T such that all predecessors of T have been scheduled
  - Schedule T on an abstract node N in order to minimize the finish time



A schedule for all concrete tasks in T1 has been computed

# Checking for An Abs Task on An Abs Node

N1: (2, 5), (8, 12), (20, 25)  
N2: (4, 9), (12, 35), (42, 45)  
N3: (10, 20), (25, 32), (35, 45)

- One possible data structure:
  - Every abstract node consists of the information as a sequence of (start\_time, end\_time) pairs when a concrete node is busy
- To schedule an abstract task of duration  $D$  and multiplicity  $M$ , we search for  **$M$   $D$ -sized gaps**
- Complexity:  **$O(\text{number of tasks scheduled})$**

# Search Engine 101

D1: Eurosys 2011  
was held in the  
Austrian city of  
Salzburg.

D2: Salzburg is a  
beautiful city. .

D3: How do I get to  
Salzburg?

- To search for Salzburg in these documents: we can go through each document one by one (Imagine Google doing that!)
- **OR** We can maintain a data structure as follows:
  - Salzburg: (D1, Line 4); (D2, Line 1), (D3, Line 2)
  - city: (D1, Line 3); (D2, Line 2)
- This data structure is known as the **inverted index**
- Benefit: Finite dictionary size leads to cost amortization

# Inverted Indices in FISCH

Node: (start, end) sequence  
N1: (2, 5), (8, 12), (20, 25)  
N2: (4, 9), (12, 35), (42, 45)  
N3: (10, 20), (25, 32), (35, 45)

Inverting  
Indices



Interval size: (Node, start) sequence  
3: (N1, 5), (N2, 9), (N3, 32)  
5: (N3, 20)  
7: (N2, 35)  
8: (N1, 12)  
\_: (N1, 25), (N2, 45), (N3, 45)

Benefit:  
To get  $M$  intervals of size  $D$ , we  
simply  
look at entries of  $N \geq D$ , and  
return  
first  $M$  intervals.

# FISCH Summary

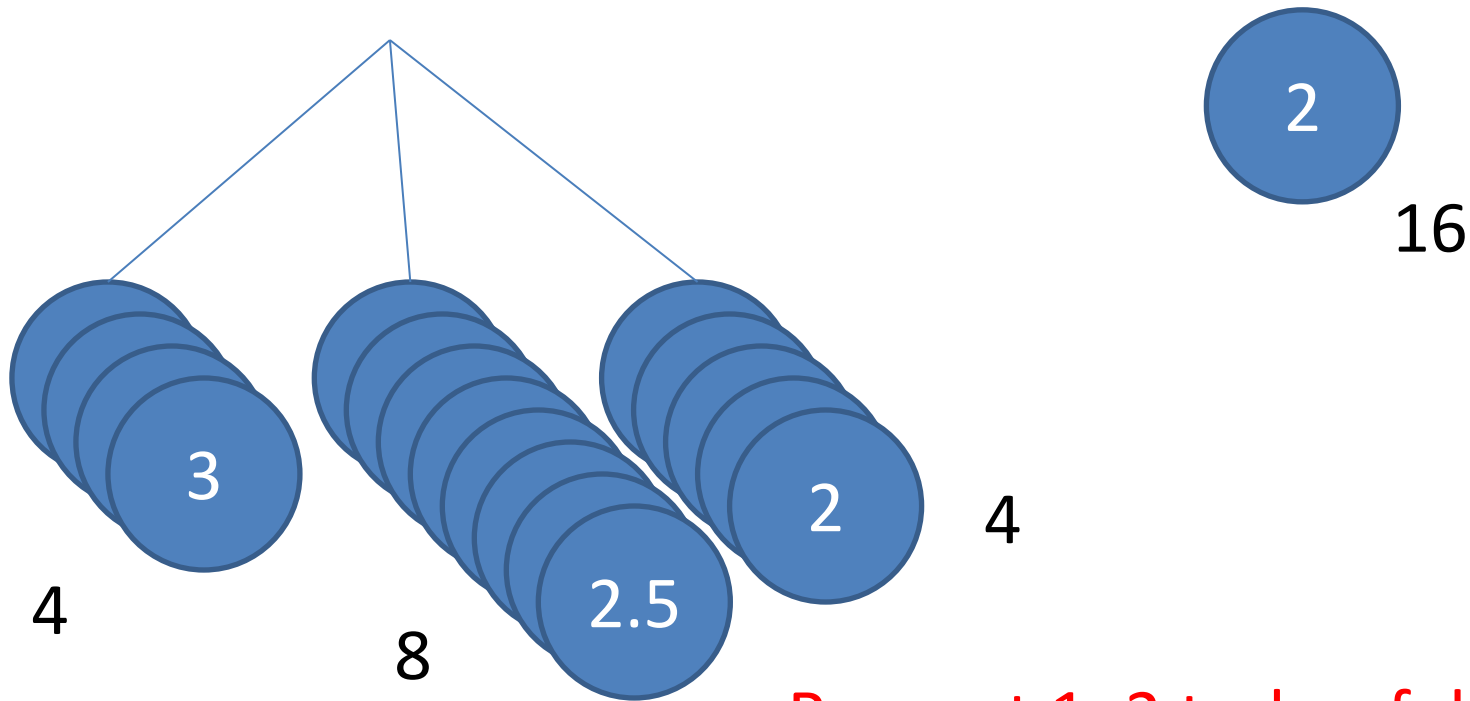
- Starts with a topological abstraction of the job, and a rack-abstraction of the cloud
- Uses inverted data structure to keep track when every concrete node is free or busy
- Greedy scheduler
- Refines the topological job abstraction when necessary
- Many details described in the paper!

# THE BLIND SCHEDULER

# Abstractions

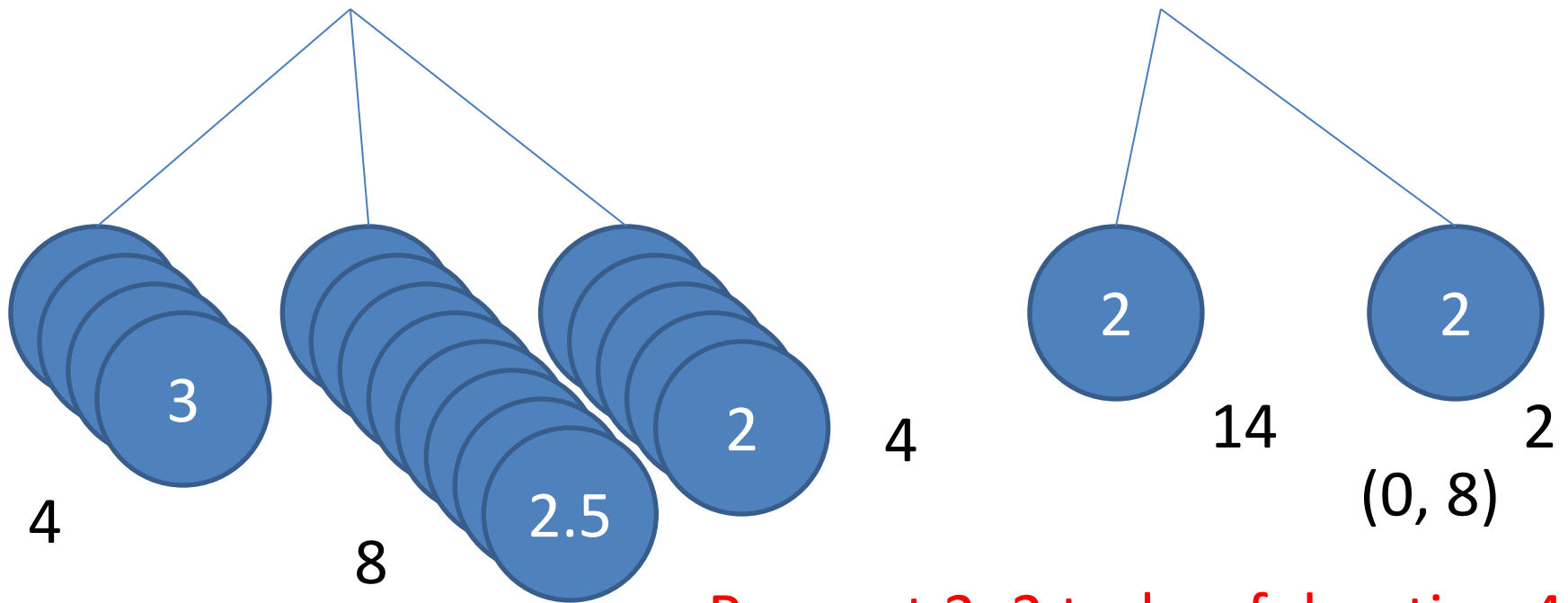
- Start with an X-similar topological job abstraction
- A single abstract node as cloud abstraction

# BLIND by example

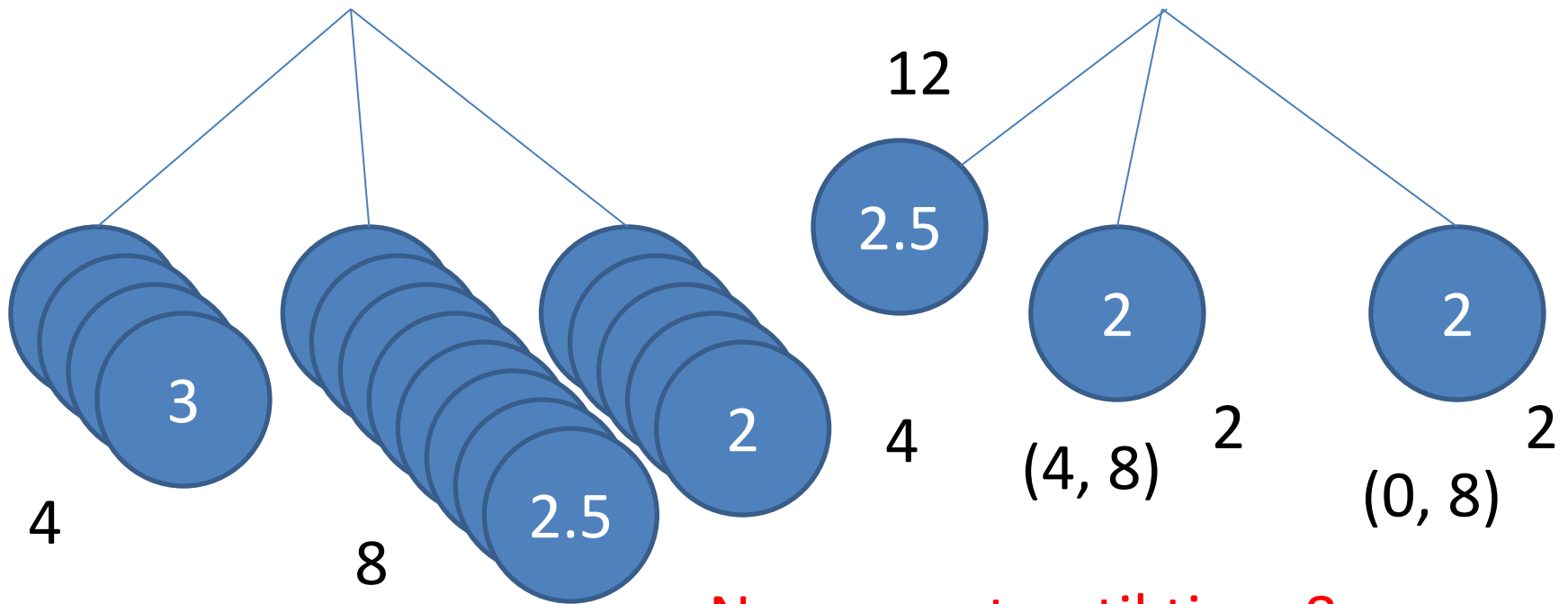


Request 1: 2 tasks of duration 8  
at time 0

# BLIND by example

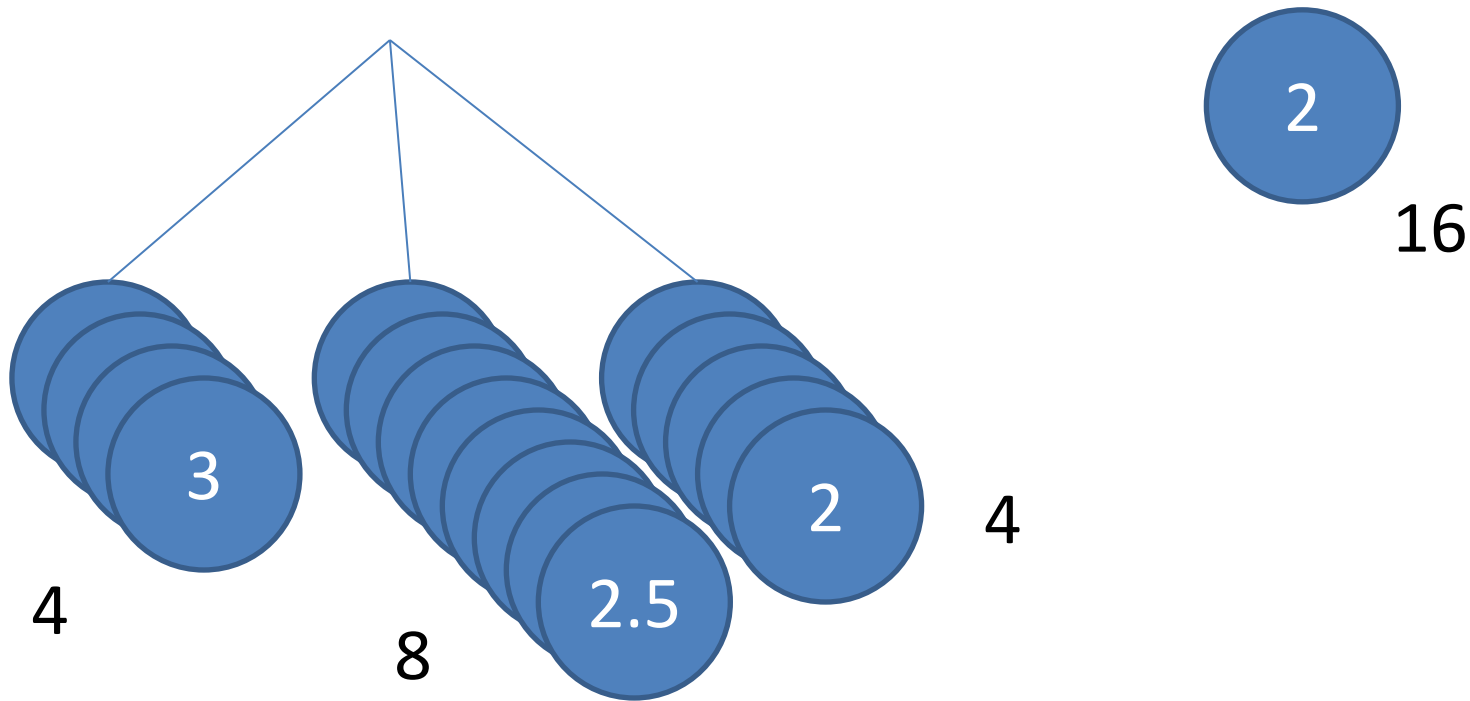


# BLIND by example



No request until time 8

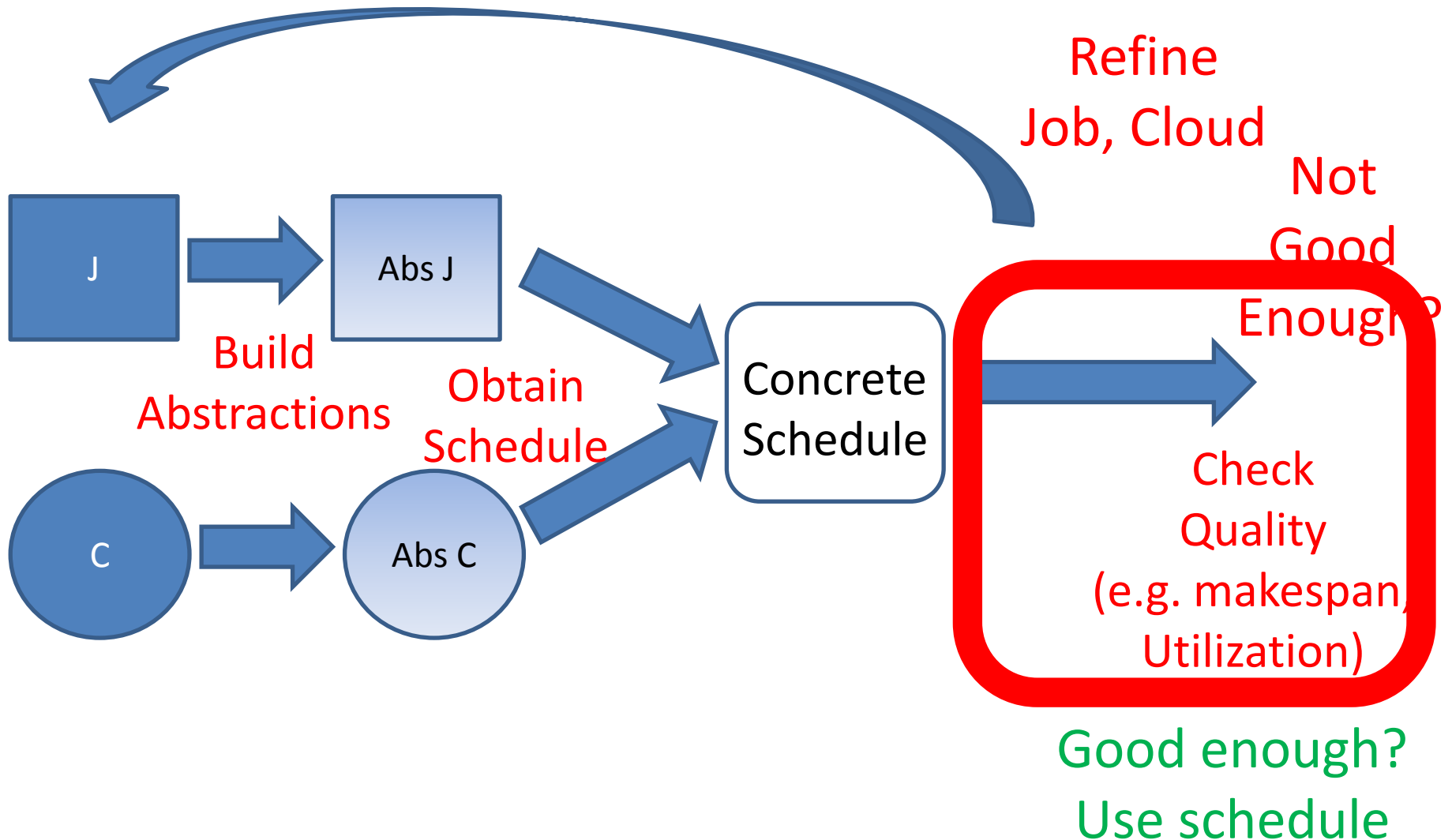
# BLIND by example



# BLIND Summary

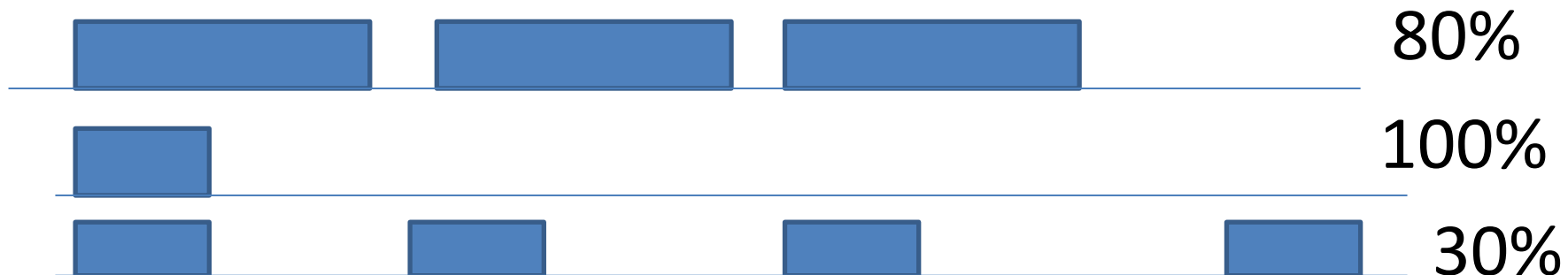
- Keep the abstraction of the cloud just as coarse as required
- On an incoming scheduling request, fragment the abstraction in a minimal fashion
- More details in the paper

# Generic AR Scheduler



# Check Quality: 2 Metrics

- **Cloud Utilization**: What is the proportion of used intervals to total scheduling duration across all nodes?

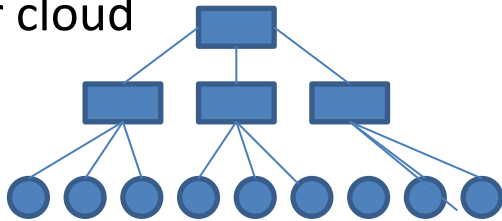


- **Schedule Makespan**: Compared to a sequential execution of the job, how much better is the duration of the schedule?

# Simulation Experiments

## Clouds

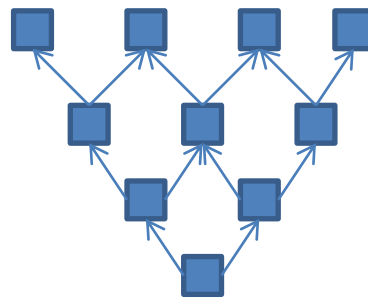
2-tier cloud



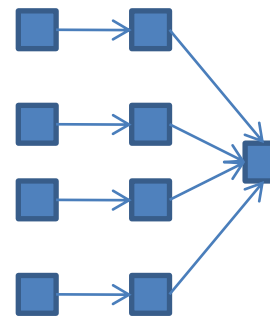
Half of the nodes:  
speed  $x$ ,  
Other half:  
speed  $1.5x$

## Jobs

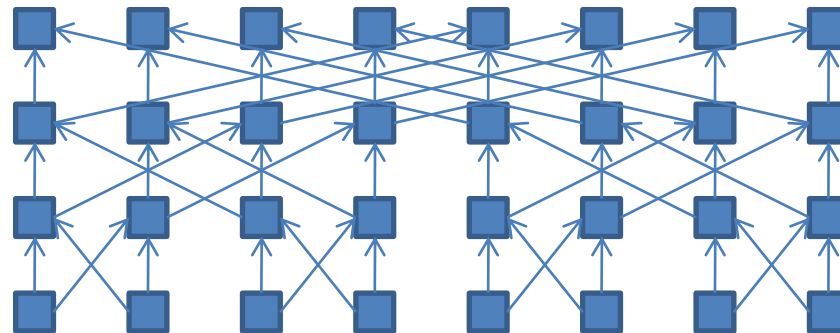
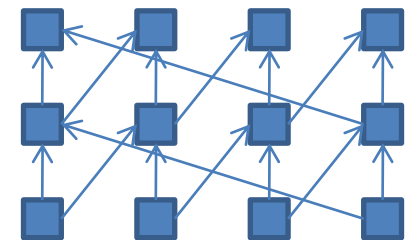
Wavefront (WF)



MapReduce(MR)



Matrix Mutl(MM)



FFT Transform  
(FFT)

# Simulation Results 1

- We create a sequence of 1000 jobs (each job with 1000 tasks with nonuniform data and compute requirements)
- We measure scheduling latency per task and cloud utilization on 2-tier cloud with 1600 nodes

---

| <b>Job</b> | <b>Latency<br/>(ms)</b> | <b>Util</b> | <b>Latency<br/>(ms)</b> | <b>Util</b> |
|------------|-------------------------|-------------|-------------------------|-------------|
| MR         | 0.34                    | 86%         | 0.32                    | 93%         |
| MM         | 1.34                    | 55%         | 1.95                    | 77%         |
| FFT        | 1.89                    | 68%         | 1.40                    | 78%         |
| WF         | 1.57                    | 49%         | 0.71                    | 62%         |

---

# Simulation Results 2

- We then compare FISCH and BLIND to a concrete greedy scheduler on a sequence of 100 jobs
- We measure scheduling latency per task and cloud utilization on 2-tier cloud with 210 nodes

| <b>Scheduler</b> | <b>Latency<br/>(ms)</b> | <b>Utilization</b> |
|------------------|-------------------------|--------------------|
| Baseline         | 293                     | 96%                |
| FISCH            | 0.27                    | 92%                |
| BLIND            | 0.16                    | 91%                |

# More Simulation Results

- ... in the paper

# COMPARISON WITH HADOOP

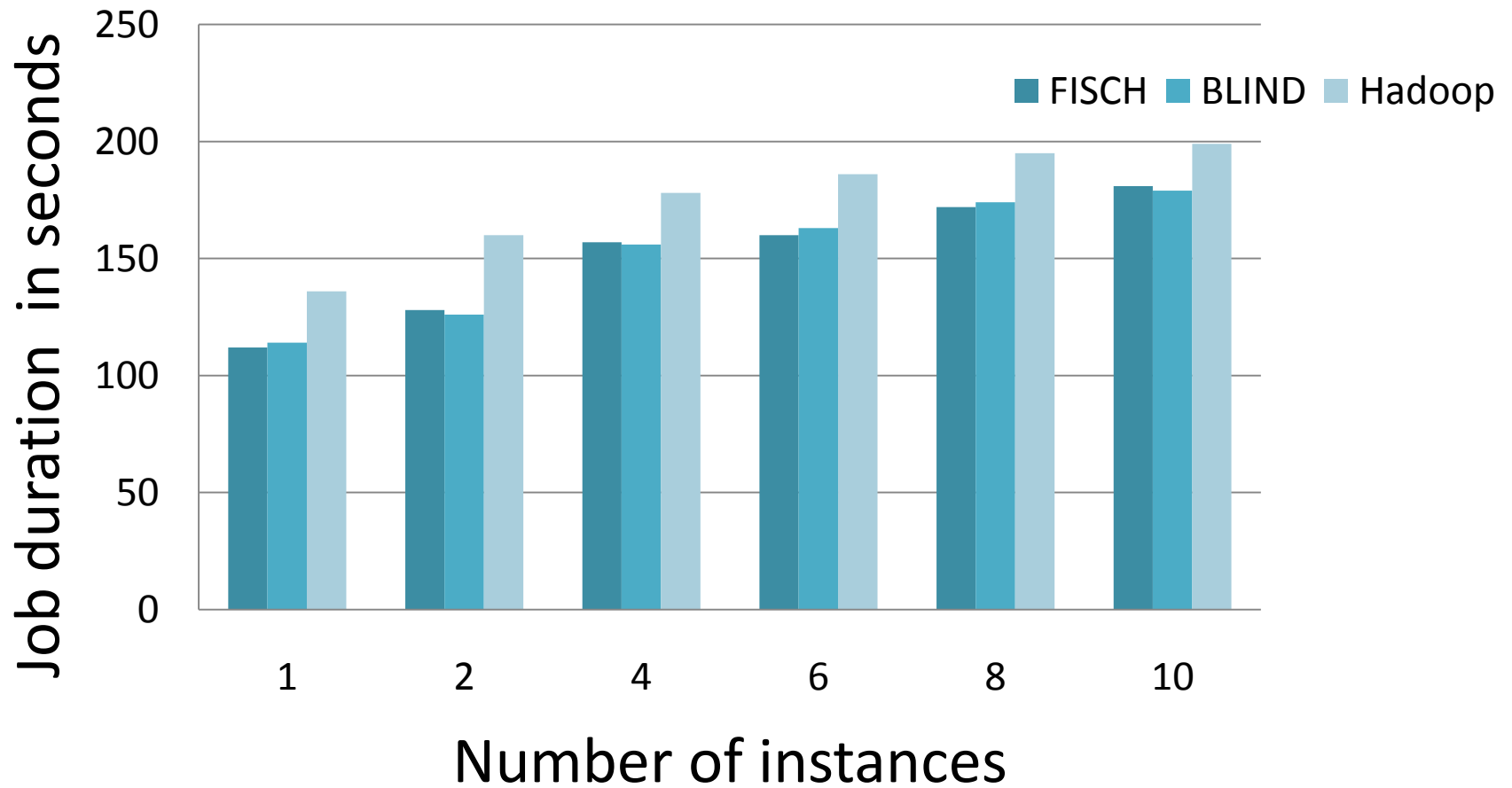
# A Word of Caution

- Static scheduling alone will not work in a data center due to
  - High variability in data center performance
  - Task durations are conservative estimates
- For comparison to Hadoop, we used static scheduling with backfilling (a dynamic scheduling technique)
- In this work (and the paper), we focus on static part, as there lies the foundation of this work

# Setup

- Job
  - A MapReduce Image Transformation Job
  - Size of each image: 4 MB
  - Mapper: An image transformation, requires 8.1 seconds on average, set the estimate to 40 seconds (use backfilling to use empty spaces)
  - Reducer: Identity operation
- Cloud
  - Amazon EC2 m1.xlarge instances (15GB RAM, 4 virtual cores, 64-bit)
  - Number of mappers = 50 \* number of instances
- Hadoop streaming version 0.19.0

# Results



# Observations

- The Hadoop framework requires large runtime overhead: results in slowdown of the job execution
- Offline scheduling allows to prefetch data in case of multiple computation stages, whereas dynamic scheduling does not

# Conclusion

- Proposed a new “offline” alternative for scheduling jobs on datacenters
- Goal: bring a deep theoretical concept from the formal methods community to build better systems
- We believe we have just scratched the surface of some appealing techniques for managing computation on the cloud
- Feel free to dive in!

# Questions?