# Introduction to Batch Scheduling

## ICS632: Principles of High-Performance Computing

Henri Casanova (henric@hawaii.edu)

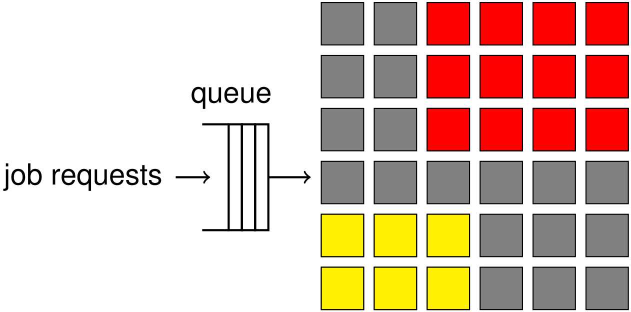
Fall 2015

# Foreword

- In these lecture notes we talk about batch scheduling
- This is something we are confronted to when using most HPC platforms
- We'll review basic principles, algorithms, and results
- We'll make small connections to theory

# Batch Scheduling

- Many users typically use the same cluster
- They shouldn't step of each others' toes
  - They should use *dedicated* subsets of the platform
- A Batch Scheduler is responsible for the sharing

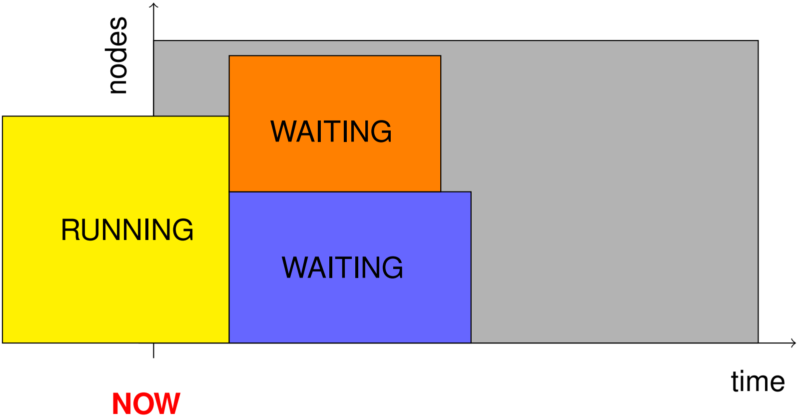


# Batch Scheduler

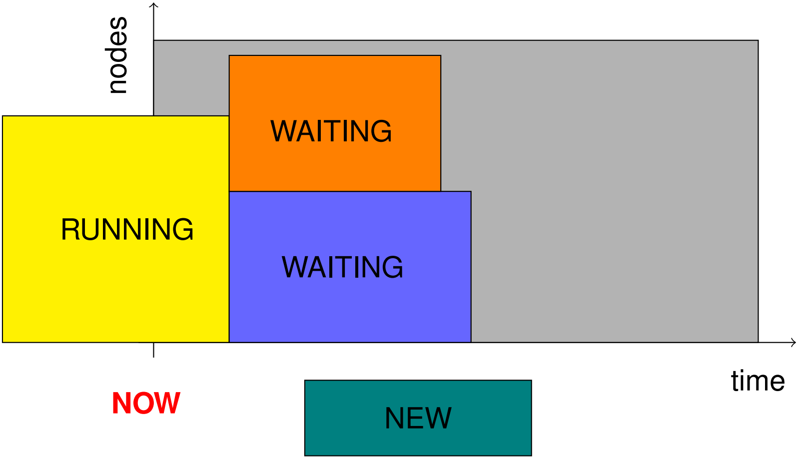
- The batch scheduler keeps track of:
  - Currently running jobs
  - Jobs in the queue
- A job is defined by:
  - Arrival time
  - Requested number of nodes and duration (“4 nodes for 2 hours”)
- Tons of bells and whistles
  - Multiple queues for various classes of jobs
  - Advance reservations
  - Charging policies
  - Heterogeneous nodes
  - ...
- Let’s keep it simple

# Graphical Representation of a Schedule

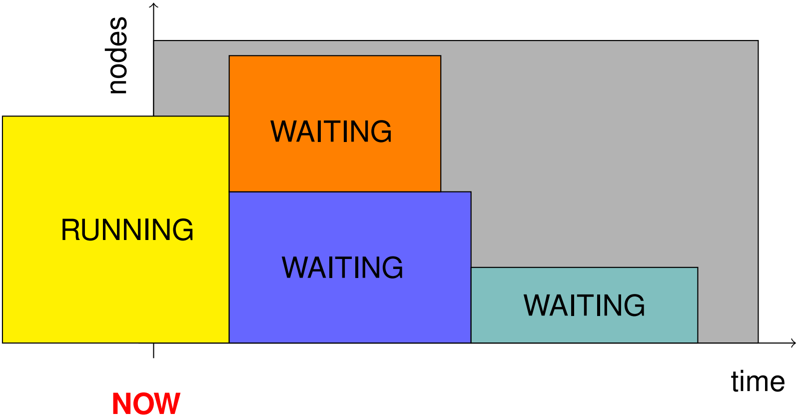
# Graphical Representation of a Schedule



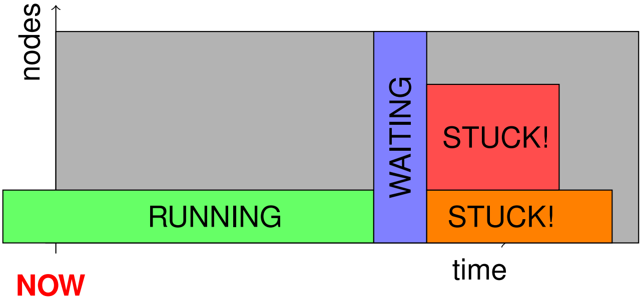
# Graphical Representation of a Schedule



# Graphical Representation of a Schedule

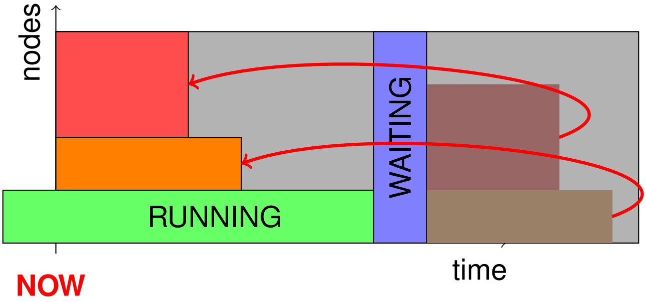


# First-Come-First-Serve (FCFS)




- Problem: fragmentation leads to idle time
  - Not good for users, not good for cluster owners

# Solution: Backfilling




- Allow jobs to “jump in line"
    - Sometimes happens at the supermarket checkout!

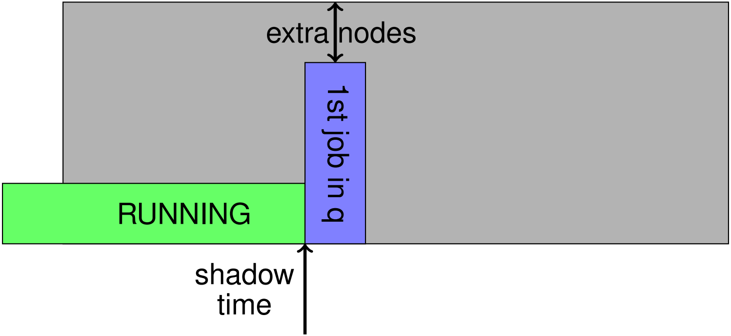
# Backfilling?

- Question: which jobs should be picked for promotion through the queue?
  - Clearly jobs that are short/small
  - Hence the motivation to provide not-too-conservative estimates through the `-t` option in SLURM
- Many heuristics are possible, but two have been studied in detail:
  - EASY
  - Conservative Back Filling (CBF)
- Most production batch schedulers are EASY-ish
  - A few systems use CBF (OAR)

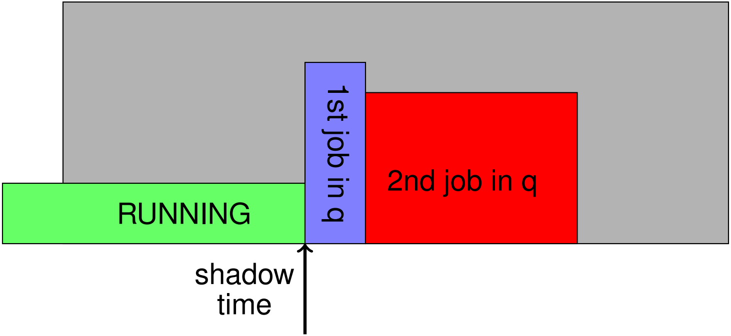
# EASY Backfilling

- Extensible Argonne Scheduling System
- Maintain only one "reservation," for the first job in the queue
- Definitions:
  - Shadow time: time at which the first job in the queue starts execution
  - Extra nodes: number of nodes idle when the first job in the queue starts execution
- Go through the queue in order starting with the 2nd job
- Backfill a job if:
  - It will terminate by the shadow time, or
  - it needs less than the extra nodes

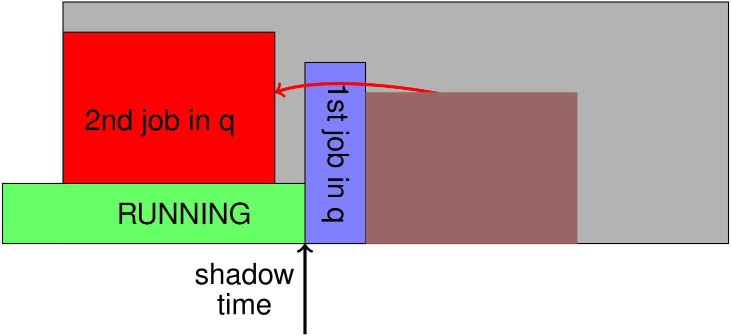
# EASY Example



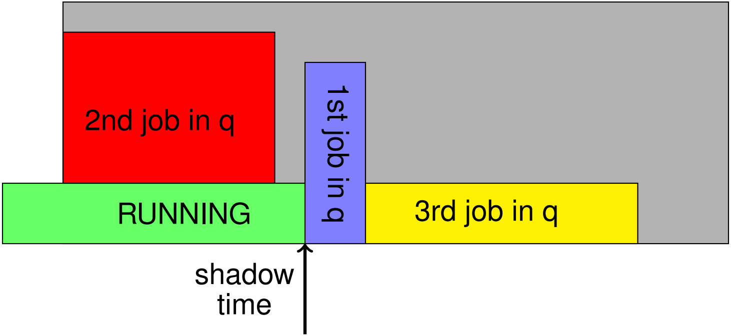
# EASY Example



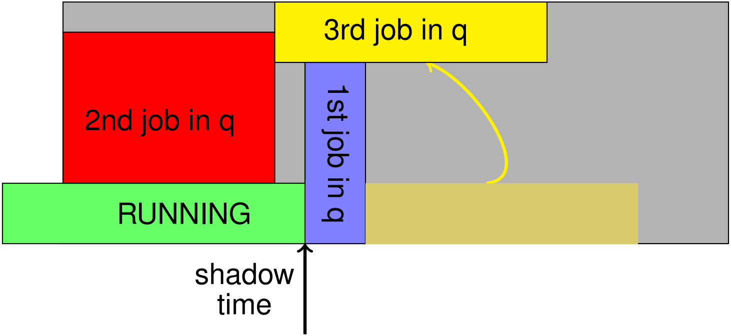
# EASY Example



# EASY Example



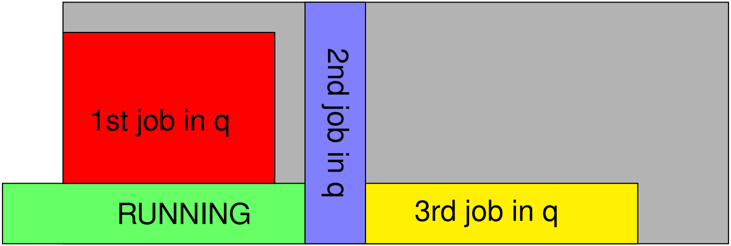
# EASY Example



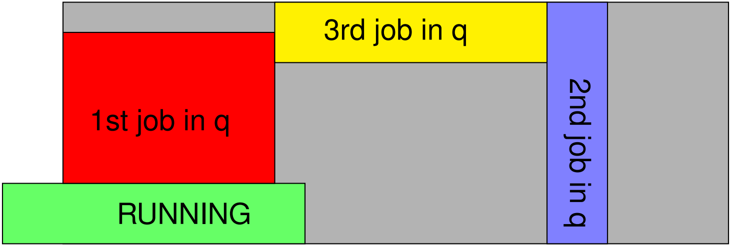
# EASY Properties

- Unbounded delay
  - The first job in the queue will never be delayed by backfilled jobs
  - BUT other jobs may be delayed infinitely!
- Let's see how that can happen...

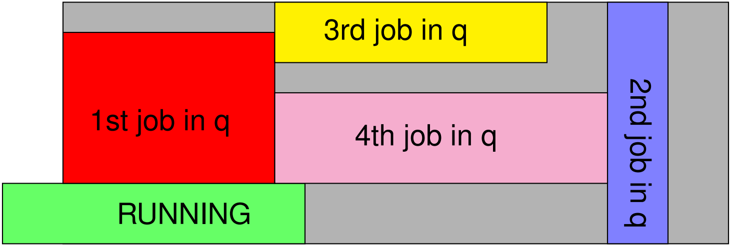
# EASY: Unbounded Delay



# EASY: Unbounded Delay



# EASY: Unbounded Delay



# EASY Properties

- Unbounded delay
  - The first job in the queue will never be delayed by backfilled jobs
  - BUT other jobs may be delayed infinitely!
- But there is no starvation:
  - Delay of first job is bounded by runtime of current jobs
  - When the first job runs, the second job becomes the first job in the queue
  - Once it is the first job, it cannot be delayed further

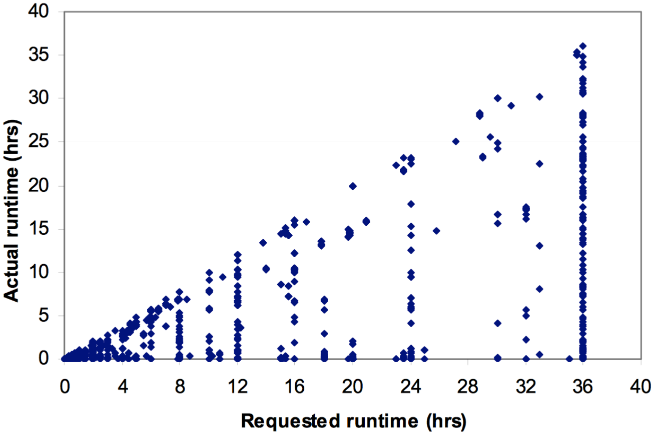
# Conservative Backfilling

- EVERY job has a "reservation"
- A job may be backfilled only if it does not delay any other job ahead of it in the queue
- Fixes the unbounded delay problem that EASY has
  - EASY favors small long jobs
  - EASY harms large short jobs
- More complicated to implement
  - The algorithm must find holes in the schedule

# Backfilling Decisions

- When does backfilling happen?
  - When a new job arrives
  - When the first job in the queue starts
  - When a job finishes (early!)
- All scheduling decisions are based on job durations
- Job durations are provided by users
- A job is killed if it goes over
- Trade-off:
  - Provide an aggressive estimate: you may be killed!
  - Provide a conservative estimate: you won't be backfilled easily
- Question: Are job duration estimates accurate?

# Estimates are NOT Accurate




from *Are user runtime estimates inherently inaccurate?*

Bailey Lee et al., JSSPP'04

# Batch Scheduling Research

- Most parallel platforms are managed by batch schedulers or things like batch schedulers
- There are many difficult questions
  - e.g., how do you schedule without knowing job durations?
- The people managing the platform often want to maximize resource utilization/profit
- The users want to minimize response time
- Many “tricks” can be played:
  - Picking the right “shape” so that you’ll be backfilled
  - Chop up your job into multiple pieces
  - Aggressively submit versions of the same job (different shapes), perhaps to multiple systems, and cancel when one begins
  - ...
- What do theoreticians think?

# Whats a Good Batch Schedule?

- We have an on-line scheduling problem
- The first step is to define a metric of goodness of a schedule
- Let's look at a few likely metrics
- **Wait time**: time spent in the queue
  - Wait time is annoying, so likely a good thing to minimize
  - Not a great idea:
    - Job #1 needs 100h on 1000 nodes and waits 1h
    - Job #2 needs 1s on 1 node and waits 1h
    - Clearly Job #1 is really happy, and Job #2 is not happy at all

# Whats a Good Batch Schedule?

- **Turn-around time**: Wait time + Execution time
  - Called *flow time* by theoreticians
  - Not a great idea:
    - Job #1 needs 1h of compute time and waits 1s
    - Job #2 needs 1s of compute time and waits 1h
    - Clearly Job #1 is really happy, and Job #2 is not happy at all

# Whats a Good Batch Schedule?

- What we want is a metric that represents “happiness” for small, large, short, long jobs
- **Slowdown**: (Wait time + Execution time) / Execution time
  - Called *stretch* by theoreticians
  - Quantifies loss of performance due to competition for the processors
  - Takes care of the short vs. long job problem
  - Doesn’t really say anything about job size...
- Two possible objectives:
  - minimize the *sum stretch* (make jobs happy on average)
  - minimize the *max stretch* (make the least happy job as happy as possible)

# On-line Max Stretch: Difficult

- The offline scheduling problem is NP-complete
- On 1 processor, with preemption allowed, there is a $O(\sqrt{X})$-competitive algorithm
  - $X$ is the ratio of largest to smallest job duration
  - Competitive ratio: ratio to the performance of an off-line algorithm that knows all jobs
- Without preemption, no approximation algorithm exists

# On-line Sum Stretch: “Easier”

- The offline scheduling problem is NP-complete
- The SRPT (Shortest Remaining Processing Time) with preemption is 2-competitive
  - At each instant, run the job that’s the closed to finishing
  - Preemption overhead is zero
- Theorems that show that an algorithm good at minimizing sum stretch can be arbitrarily bad at minimizing max stretch, and conversely

# Conclusion: A Massive Divide

- Practice: No preemption in batch scheduling, need for many scheduling configuration knobs
- Theory: Without preemption, we can't do anything guaranteed anyway

- The two remain very divorced
- The stretch is used as a metric to evaluate how good scheduling is in practice, but it often isn't the objective of the batch scheduler
  - That objective is complex, mysterious, and not necessarily theoretically-motivated