# Parallel Job Scheduling and Workloads

Dror Feitelson

Hebrew University

# Parallel Jobs

- A set of processes that cooperate to solve a problem
  - Example: weather forecast, industrial/military simulation, scientific discovery
- Processes run in parallel on distinct processors (or cores), communicate using high-speed (on-chip) network
- On MPPs: run to completion on dedicated processors to avoid memory problems
- On multicores: probably more dynamic

# MPP Parallel Job Scheduling

- Each job is a rectangle in processorsXtime space
- Given many jobs, we must schedule them to run on available processors
- This is like packing the rectangles
- Want to minimize space used, i.e. minimize used resources and fragmentation
- On-line problem: don’t know future arrivals or runtimes

# Workloads

- System performance depends on the workload
  - Analogy: algorithm performance depends on the input
- Evaluation workload should be representative of real workloads
- In our case, the workload is a sequence of jobs to run
- Can use data from system accounting logs

# Parallel Workloads Archive

- All large scale supercomputers maintain accounting logs
- Data includes job arrival, queue time, runtime, processors, user, and more
- Many are willing to share them

  (and shame on those who are not)
- Collection at www.cs.huji.ac.il/labs/parallel/workload/
- Uses standard format to ease use

# Example: NASA iPSC/860 trace

| *user* | *cmd* | *proc* | *runtm* | *date* | *time* |
|---|---|---|---|---|---|
| user8 | cmd33 | 1 | 31 | 10/19/93 | 18:06:10 |
| sysadmin | pwd | 1 | 16 | 10/19/93 | 18:06:57 |
| sysadmin | pwd | 1 | 5 | 10/19/93 | 18:08:27 |
| intel0 | cmd11 | 64 | 165 | 10/19/93 | 18:11:36 |
| user2 | cmd2 | 1 | 19 | 10/19/93 | 18:11:59 |
| user2 | cmd2 | 1 | 11 | 10/19/93 | 18:12:28 |
| user2 | nsh | 0 | 10 | 10/19/93 | 18:16:23 |
| user2 | cmd1 | 32 | 2482 | 10/19/93 | 18:16:37 |

# Using Traces

- In simulations, traces can be used directly to generate the input workload
  - Jobs arrive according to timestamps in the trace
  - Each job requires the number of processors and runtime specified in the trace
- Used to evaluate new schedulers
- Can also be used as data for workload models

# Outline

- Packing in gang scheduling
- Feedback and its effect
- Scheduling and workloads on CMPs
- Conclusions

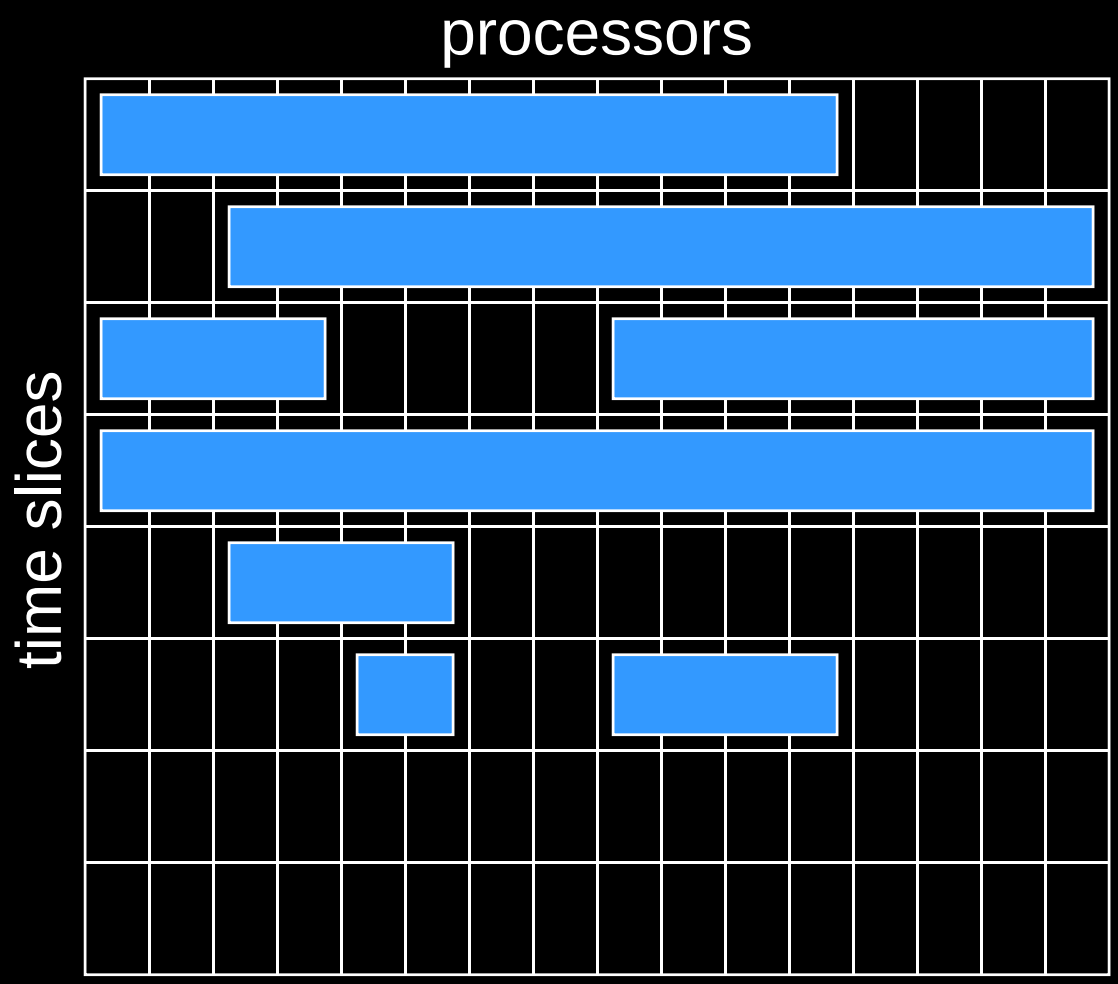
# Outline

- Packing in gang scheduling
- Feedback and its effect
- Scheduling and workloads on CMPs
- Conclusions

# Gang Scheduling

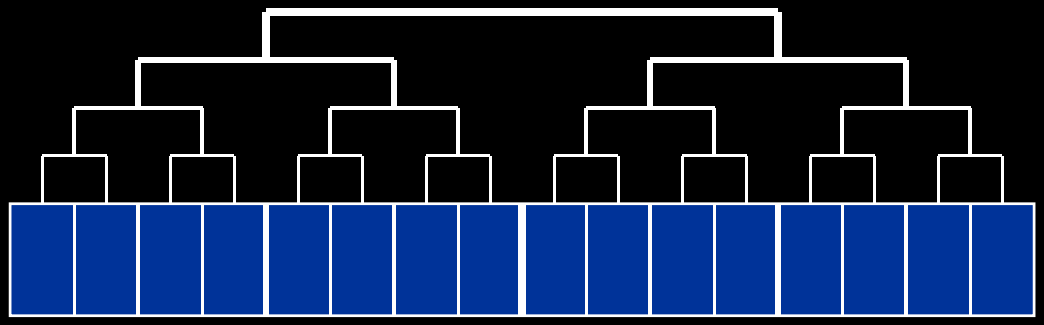
- Use coordinated time slicing across all nodes
  - Processes in same job run in parallel ⇨ enables fine-grain interactions
  - Short jobs are not delayed by long ones ⇨ better response time
- Only need to pack in one dimension (processors)
- Caveat: all jobs need to be memory-resident

# Ousterhout Matrix



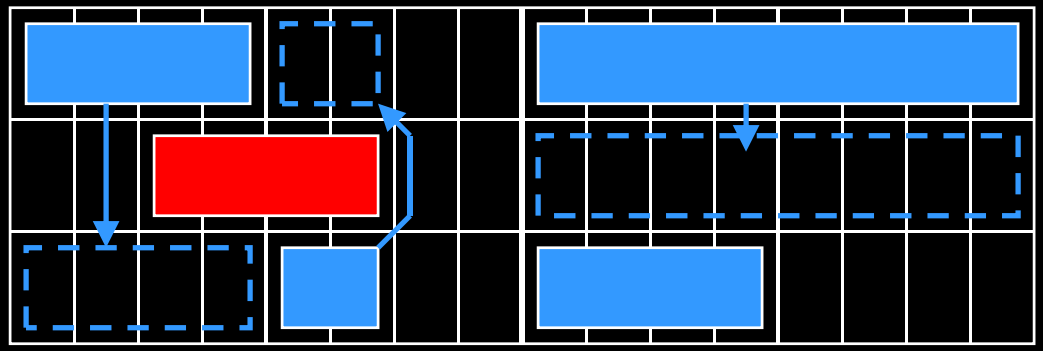
# Buddy-System Packing

- Allocate processors in predefined blocks of powers of two



# Buddy-System Packing

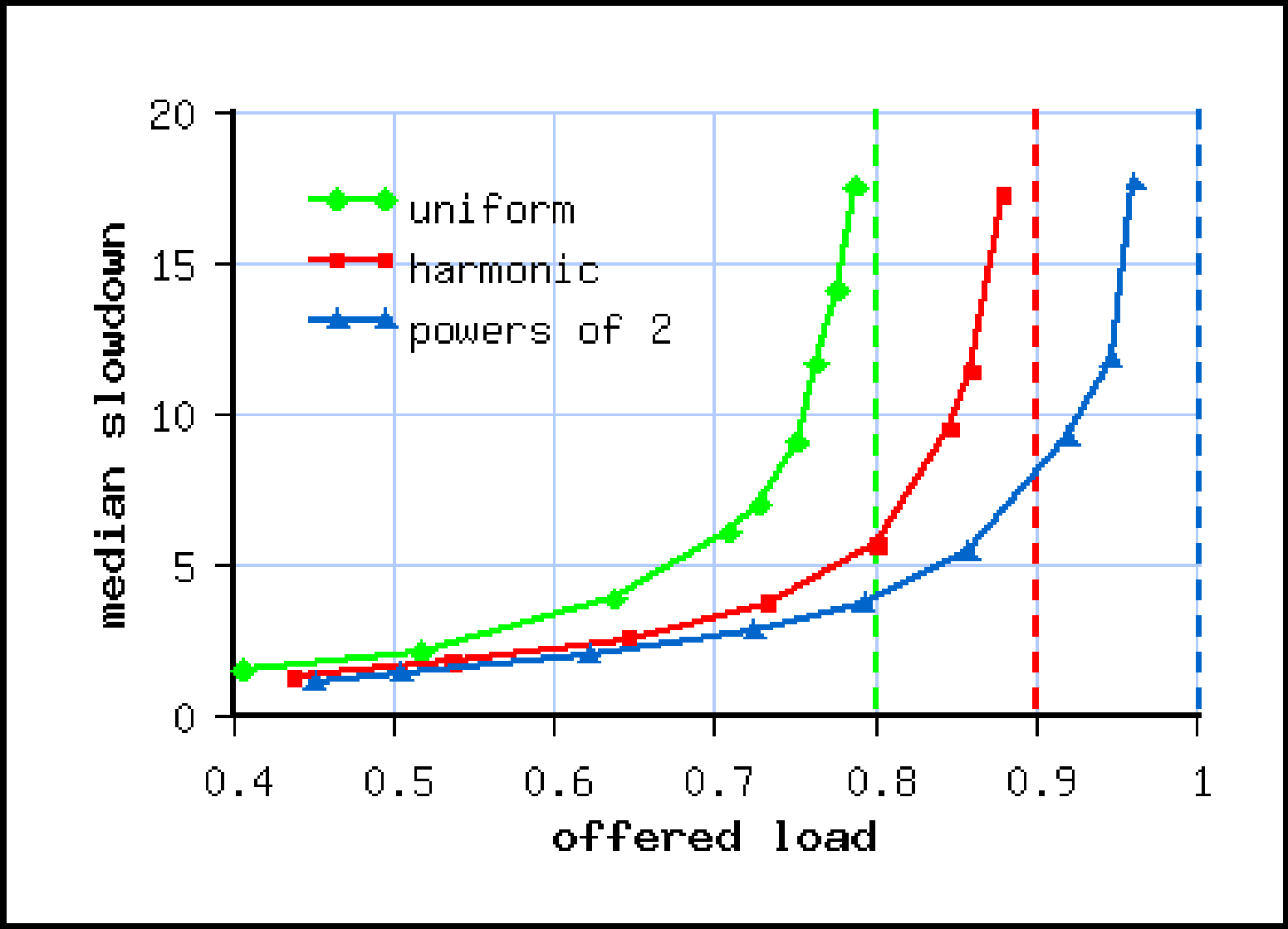
- Allocate processors in predefined blocks of powers of two
- Full block will either be allocated or left empty in different slots
- Jobs mapped to one slot may also run in another slot if block is free there



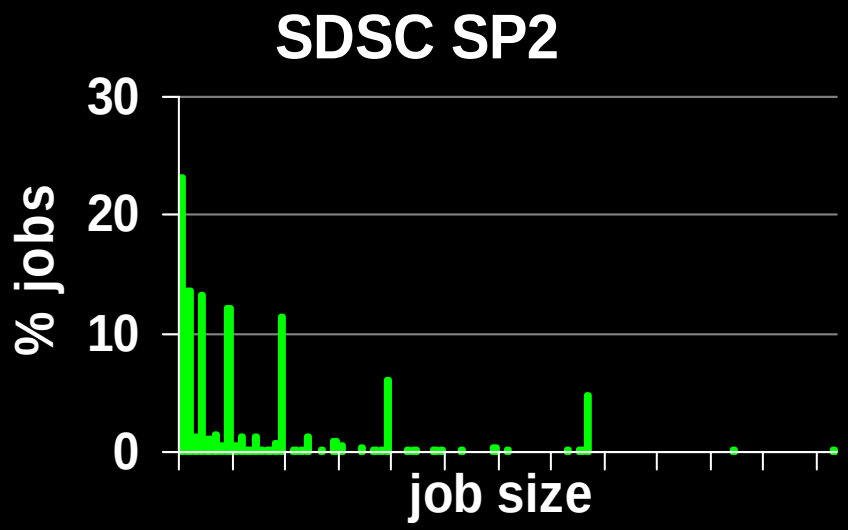
# Buddy-System Packing

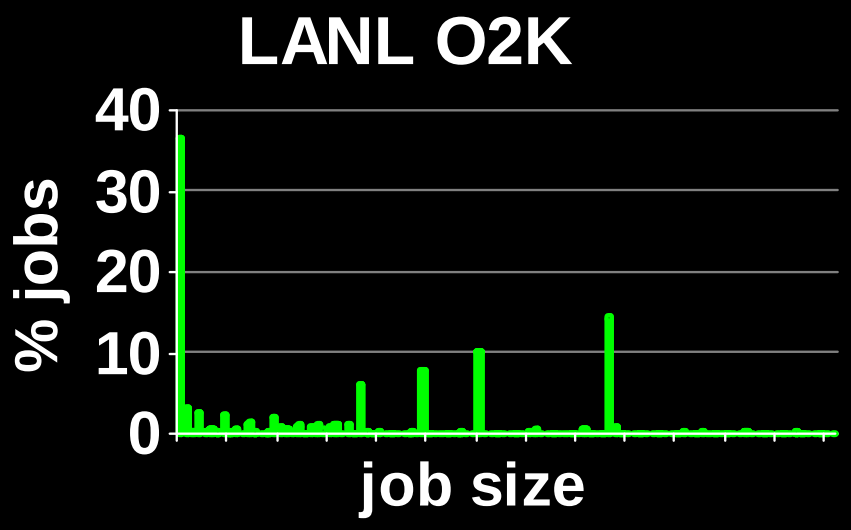
- Allocate processors in predefined blocks of powers of two
- Full block will either be allocated or left empty in different slots
- Jobs mapped to one slot may also run in another slot if block is free there
- Possible cost of additional fragmentation
- Depends on distribution of job sizes

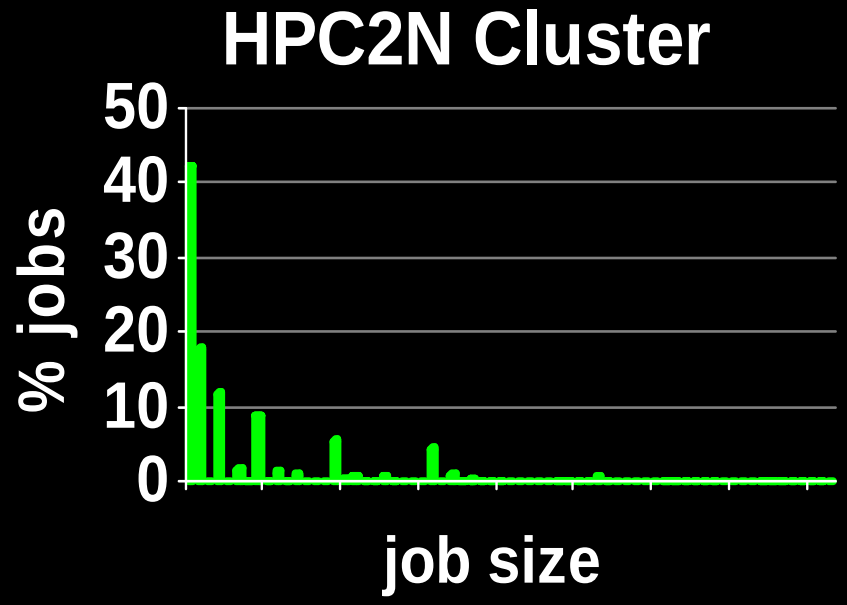
# Simulation Results

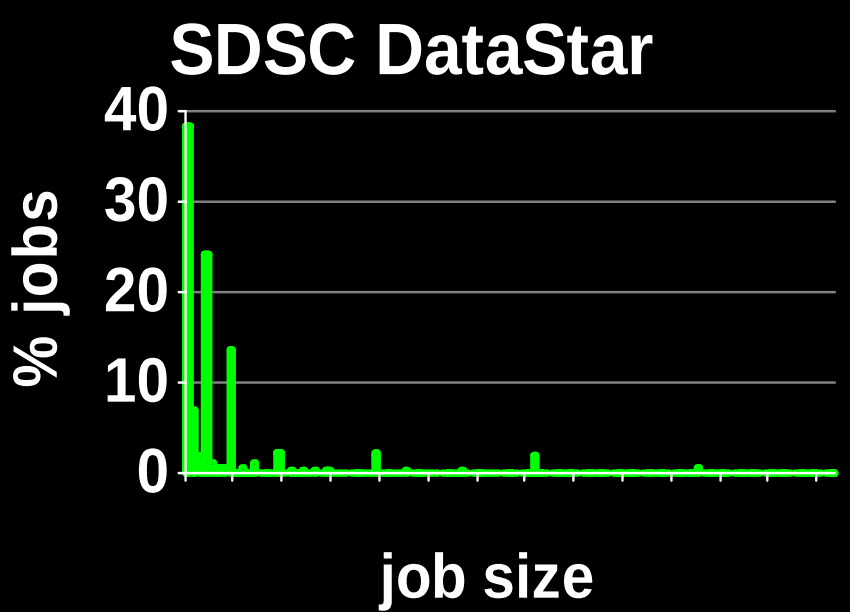


# Trace Data


SDSC SP2
30
20
10
0
% jobs
job size


LANL O2K
40
30
20
10
0
% jobs
job size


HPC2N Cluster
50
40
30
20
10
0
% jobs
job size


SDSC DataStar
40
30
20
10
0
% jobs
job size

# Trace Data

- Many small jobs
- Many sequential jobs
- Many power of two jobs
- Practically no jobs use full machine

Conclusion: buddy system should work well

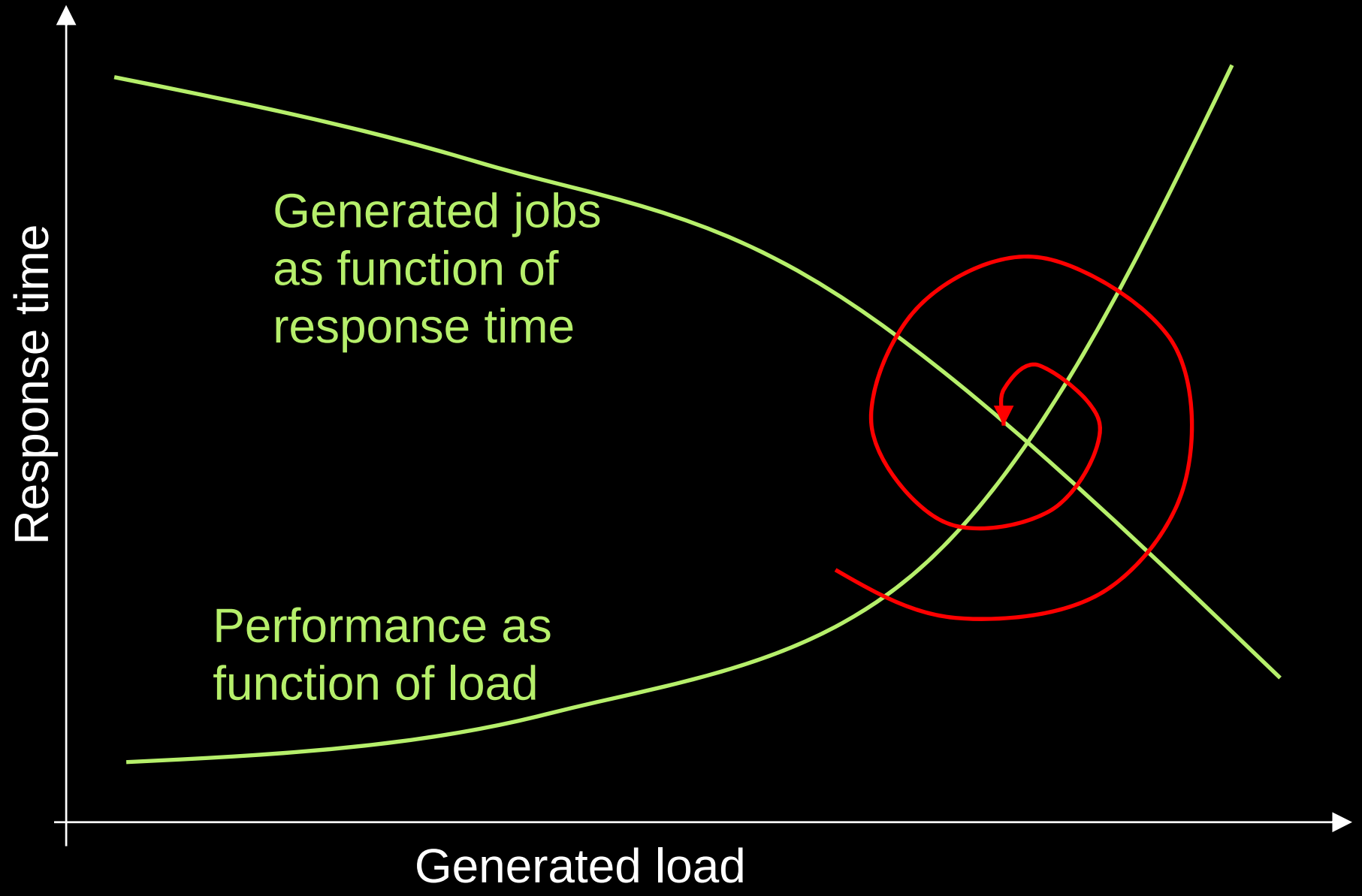
# Outline

- Packing in gang scheduling
- Feedback and its effect
- Scheduling and workloads on CMPs
- Conclusions

# Independence vs. Feedback

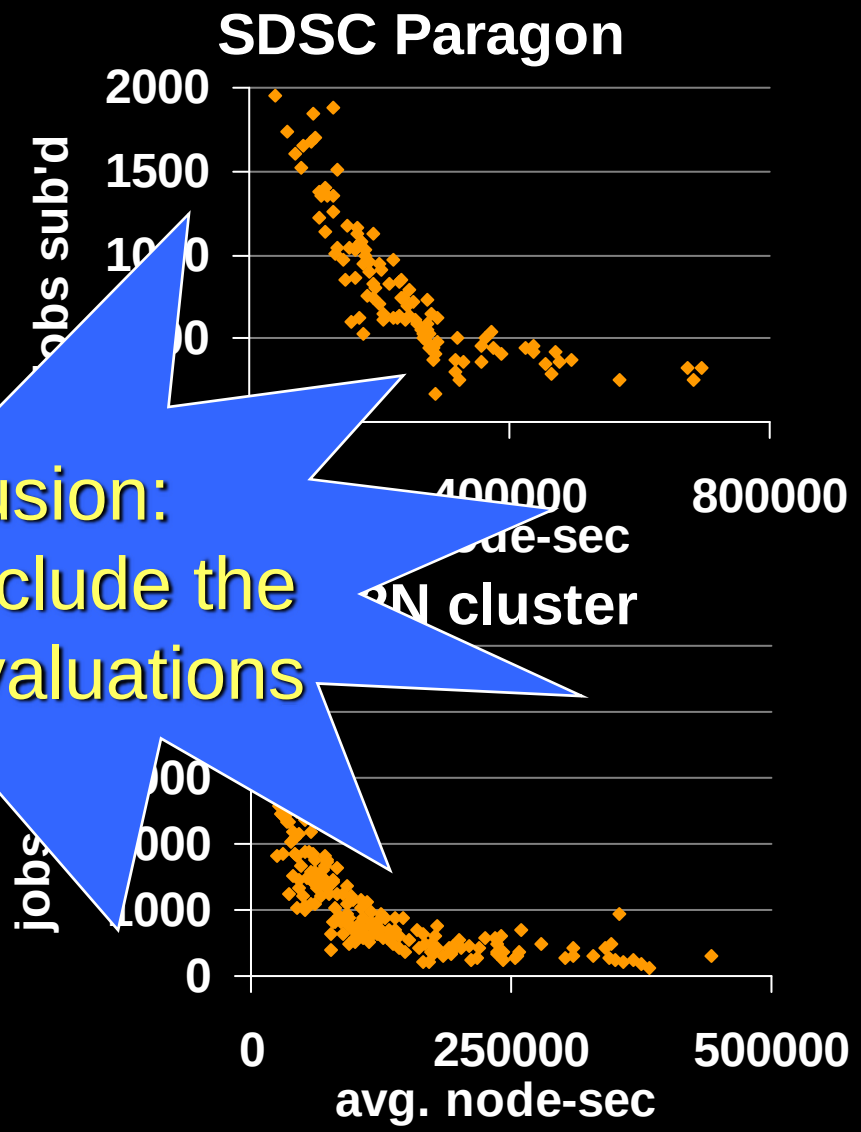
- Replaying a trace to recreate a workload assumes an open system model
  - Large user population insensitive to system performance
  - Jobs arrivals are independent of each other
- But real systems are often closed
  - Limited user population
  - New jobs submitted after previous ones terminate
- This leads to feedback from system performance to workload generation

# Feedback Determines Load



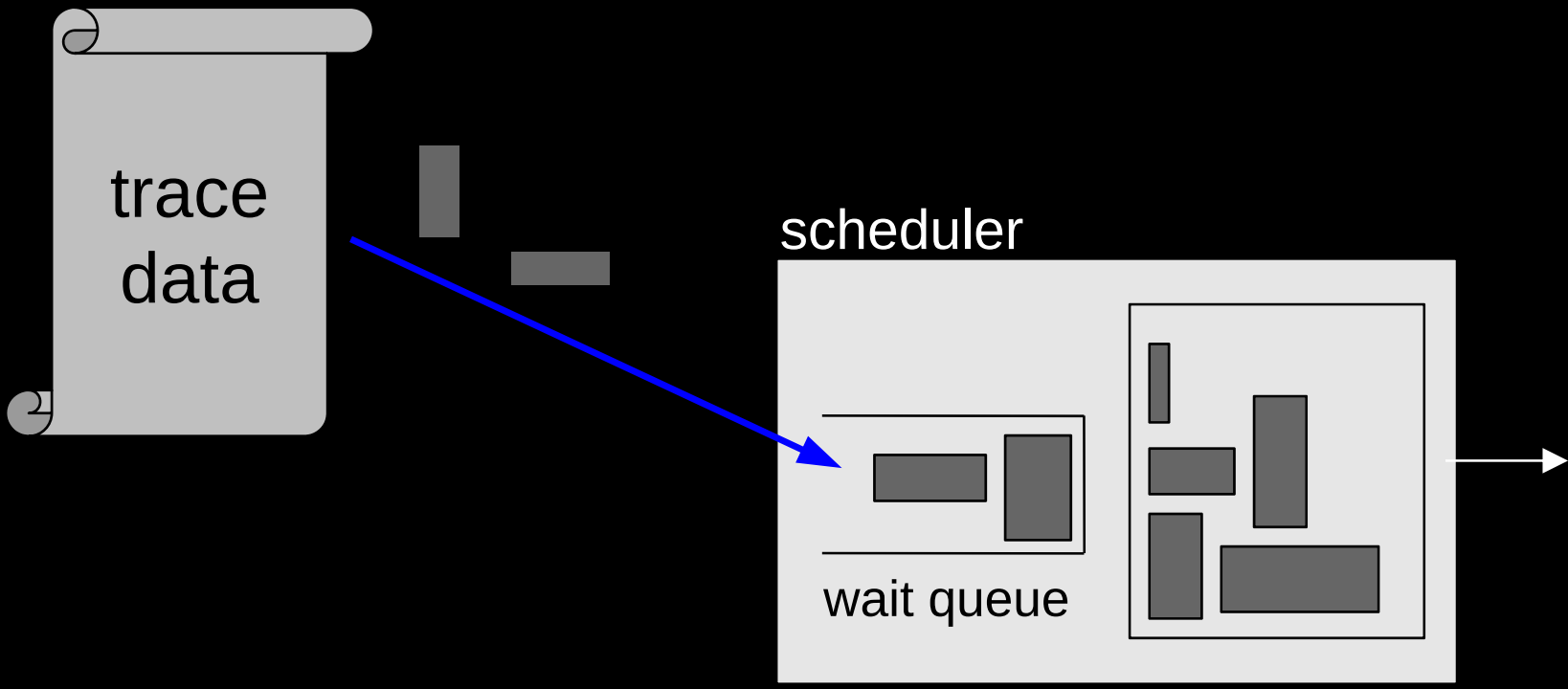
# Evidence for Feedback

- In weeks with many jobs, they are small
- In weeks with large jobs, there are few them
- Users reach conditions whe submitting new job
- Jobs are not independent

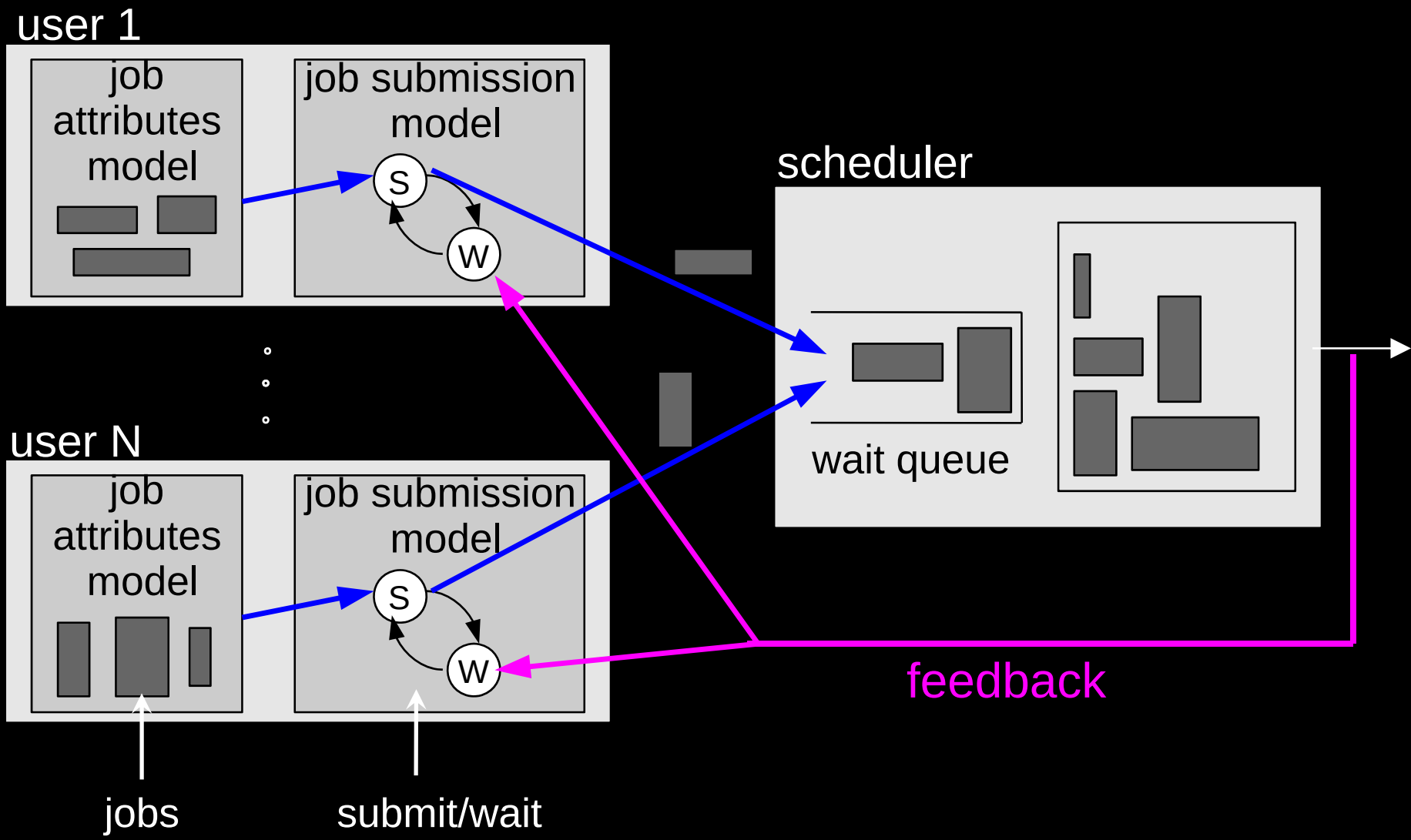



Conclusion:
Need to include the users in evaluations

# Old Model



# Site-Level Model



# The Problem with Traces

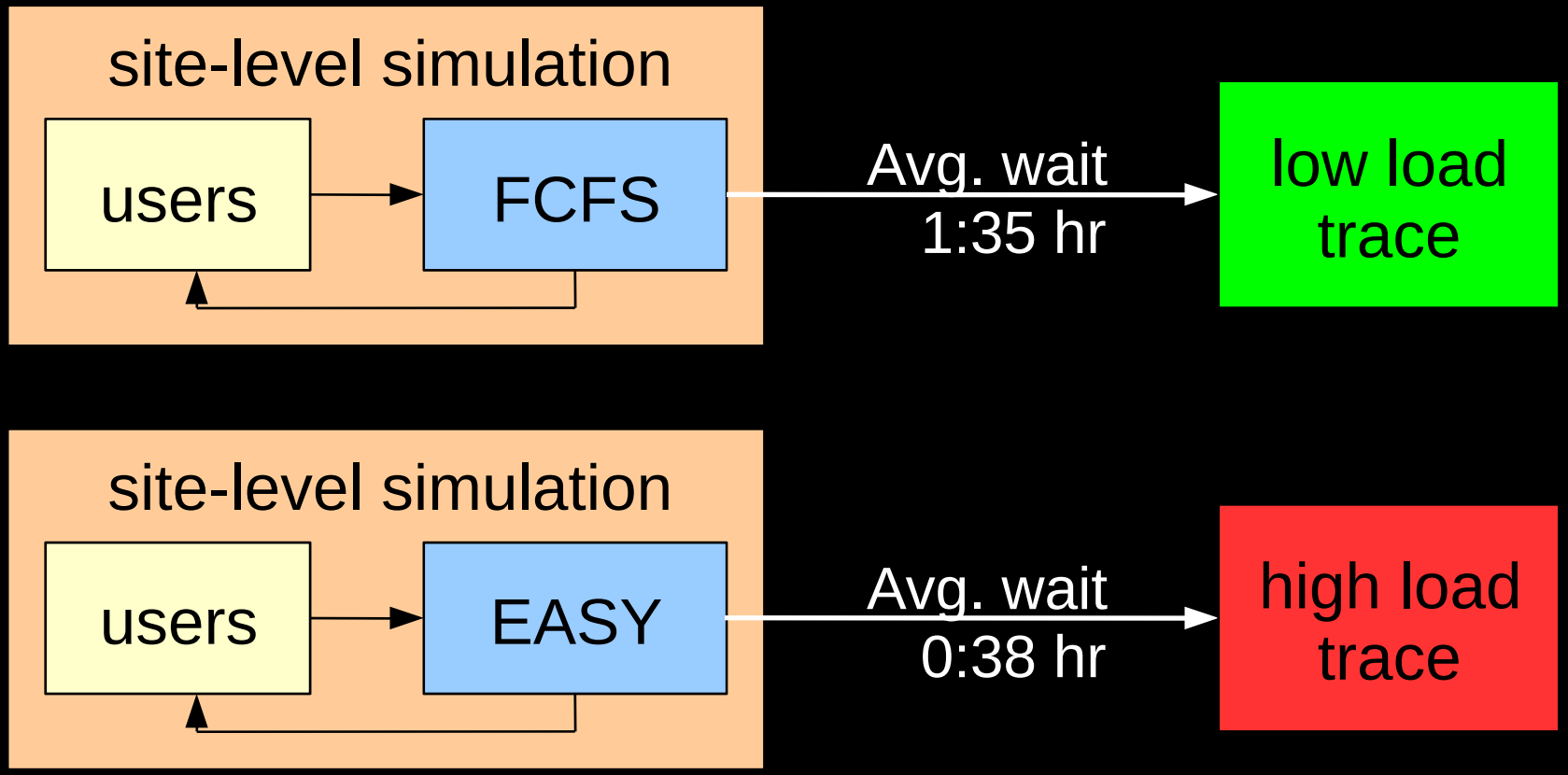
- A trace reflects the behavior of the scheduler on the traced system
- And its interaction with the users
- And the users' response to its performance

- So using it to evaluate another scheduler may lead to unreliable results!

# Example

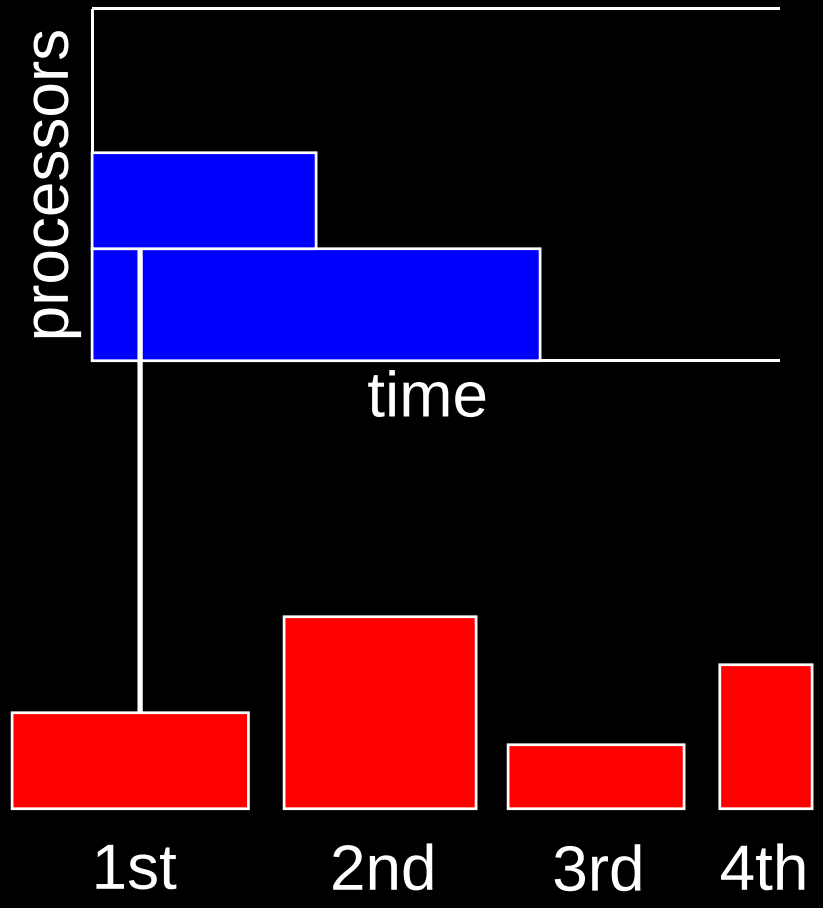
Step 1: generate traces

- FCFS – a simple scheduler that cannot support a high load
- EASY – a more efficient backfilling scheduler



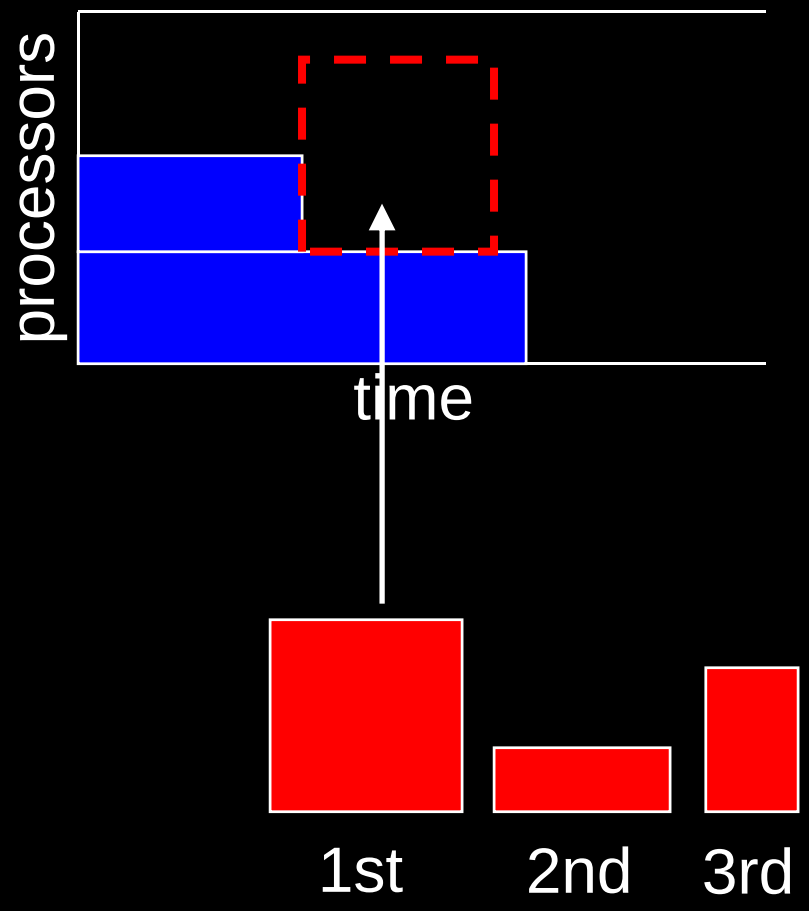
# EASY Operation

1. Schedule jobs on available processors in FCFS order
2. Make reservation for first job that cannot run
3. Schedule additional jobs provided they do not conflict with this reservation



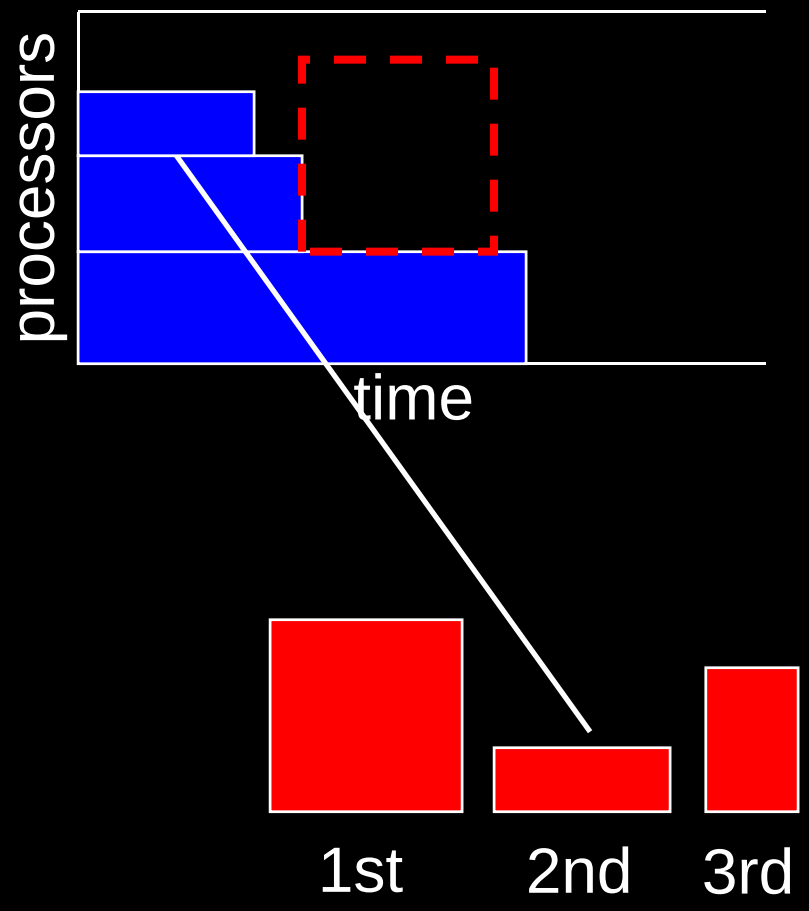
# EASY Operation

1. Schedule jobs on available processors in FCFS order
2. Make reservation for first job that cannot run
3. Schedule additional jobs provided they do not conflict with this reservation



# EASY Operation

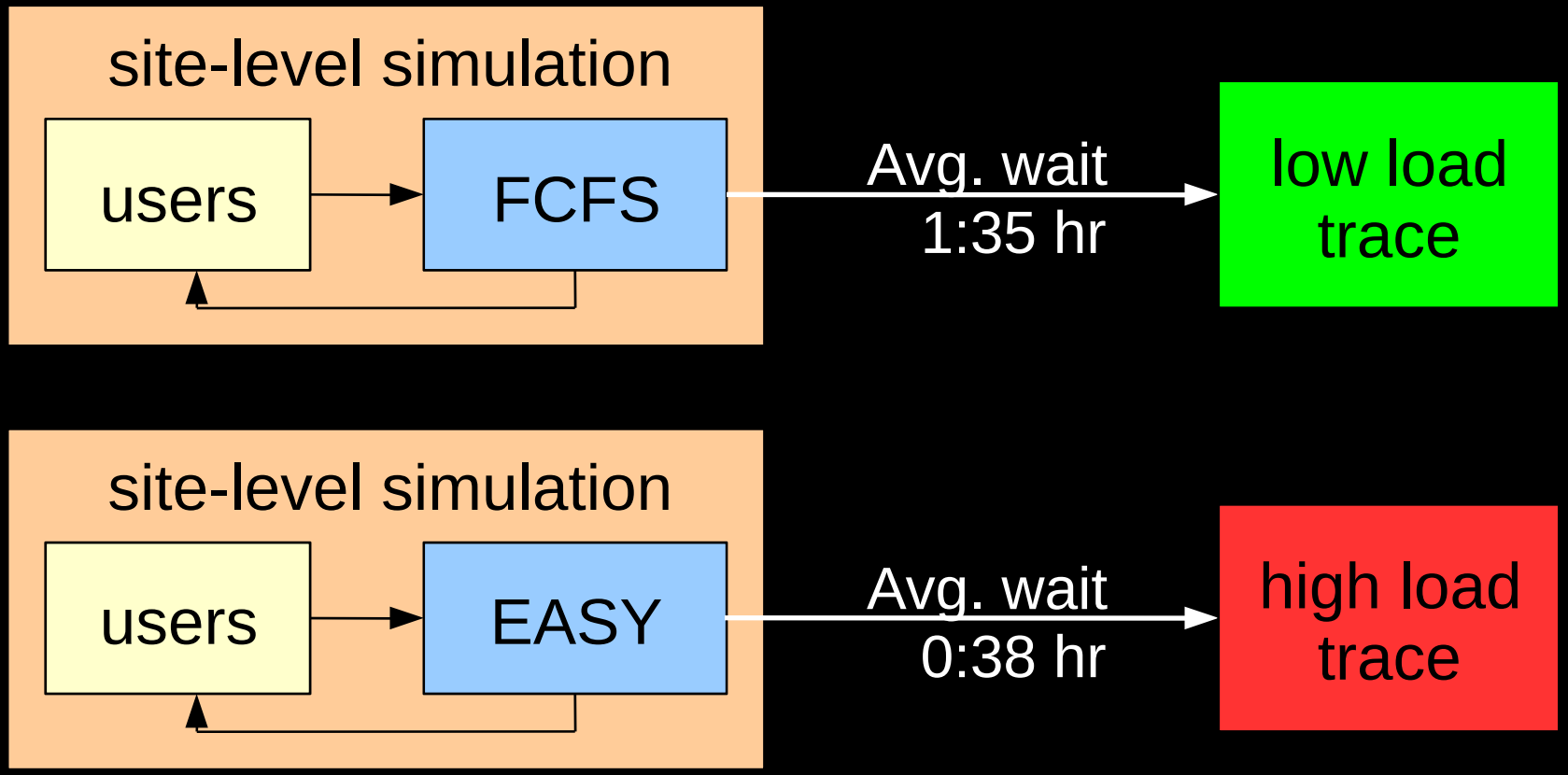
1. Schedule jobs on available processors in FCFS order
2. Make reservation for first job that cannot run
3. Schedule additional jobs provided they do not conflict with this reservation



# Example

Step 1: generate traces

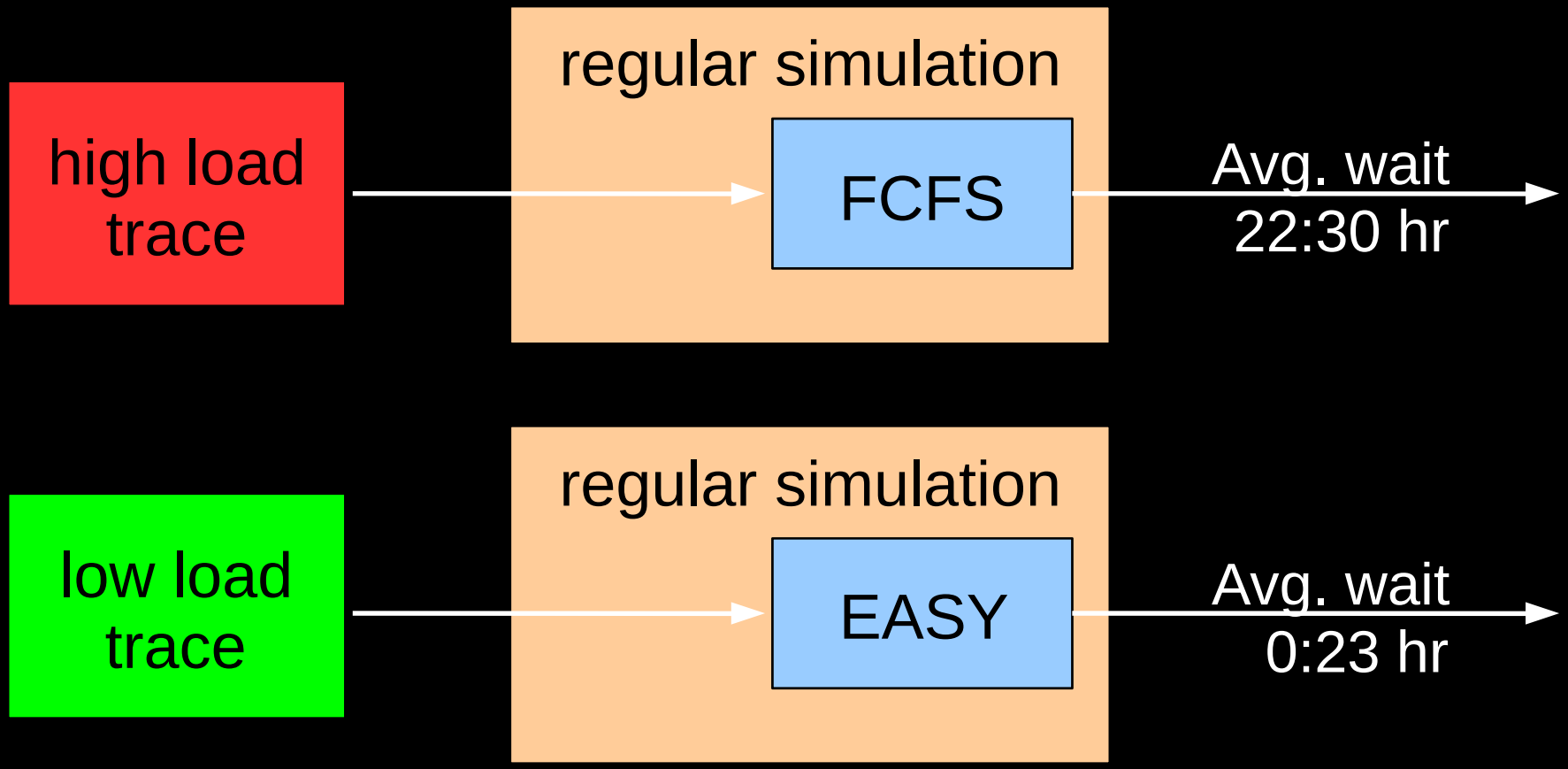
- FCFS – a simple scheduler that cannot support a high load
- EASY – a more efficient backfilling scheduler



# Example

Step 2: switch traces and use in regular simulation

- simulate FCFS with high-load trace generated from EASY
- simulate EASY with a low-load trace generated from FCFS



# Example

Comparison of wait-time results:

| Scheduler | Site-lev | Sim | difference |
|---|---|---|---|
| FCFS | | | +1345% |
| EASY | .38 | 0:23 | -38% |

Conclusion:
Need good user model

Using the wrong trace can have a huge effect!

# The Mechanics of Feedback

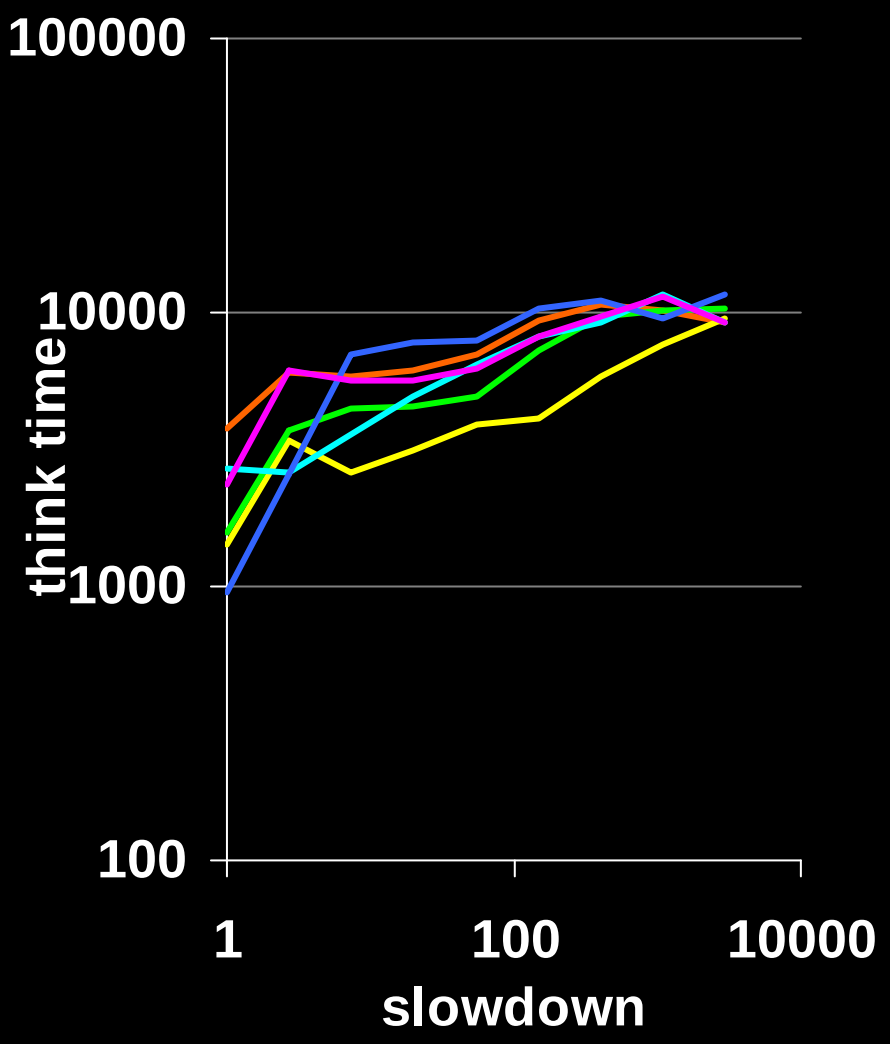
- If users perceive the system as loaded, they will submit less jobs
- But what exactly do users care about?
  - Response time: how long they wait for results
  - Slowdown: how much longer than expected
- Answer needed to create a user model that will react correctly to load conditions

# Data Mining

- Available data: system accounting log
- Need to assess user reaction to momentary condition
- The idea: associate the user's think time with the performance of the previous job
  - Good performance ⇨ satisfied user ⇨ continue work session ⇨ short think time
  - Bad performance ⇨ dissatisfied user ⇨ go home ⇨ long think time
- "performance" = response time or slowdown

# The Data

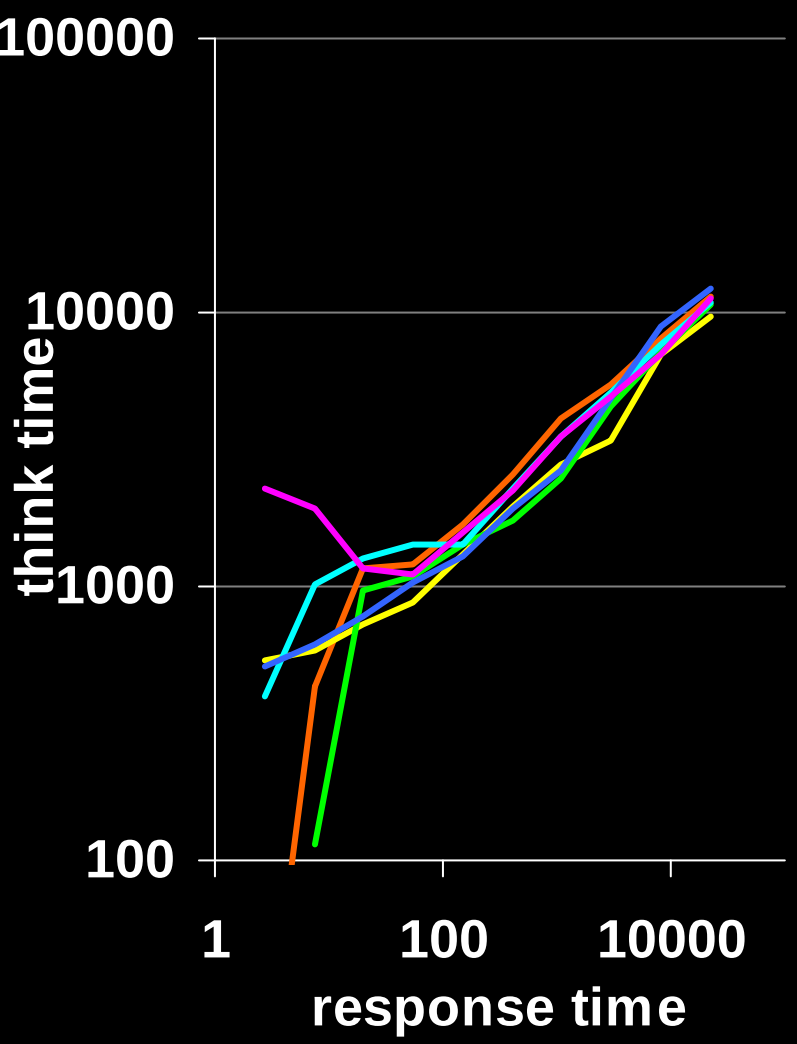
- Similar average think time for widely different slowdowns
- Differences between different logs
- Slowdown is not a good predictor of user behavior



# The Data

- Think time depends strongly on response time
- Very similar behavior in all logs
- Response time is a good predictor of user behavior

  (at least above 1 min.)



# User Behavior Model

- Users work in sessions
- If a job's response time is short, there is a high probability for a short think time and an additional job
- If a job's response time is long, there is a high probability for a long think time

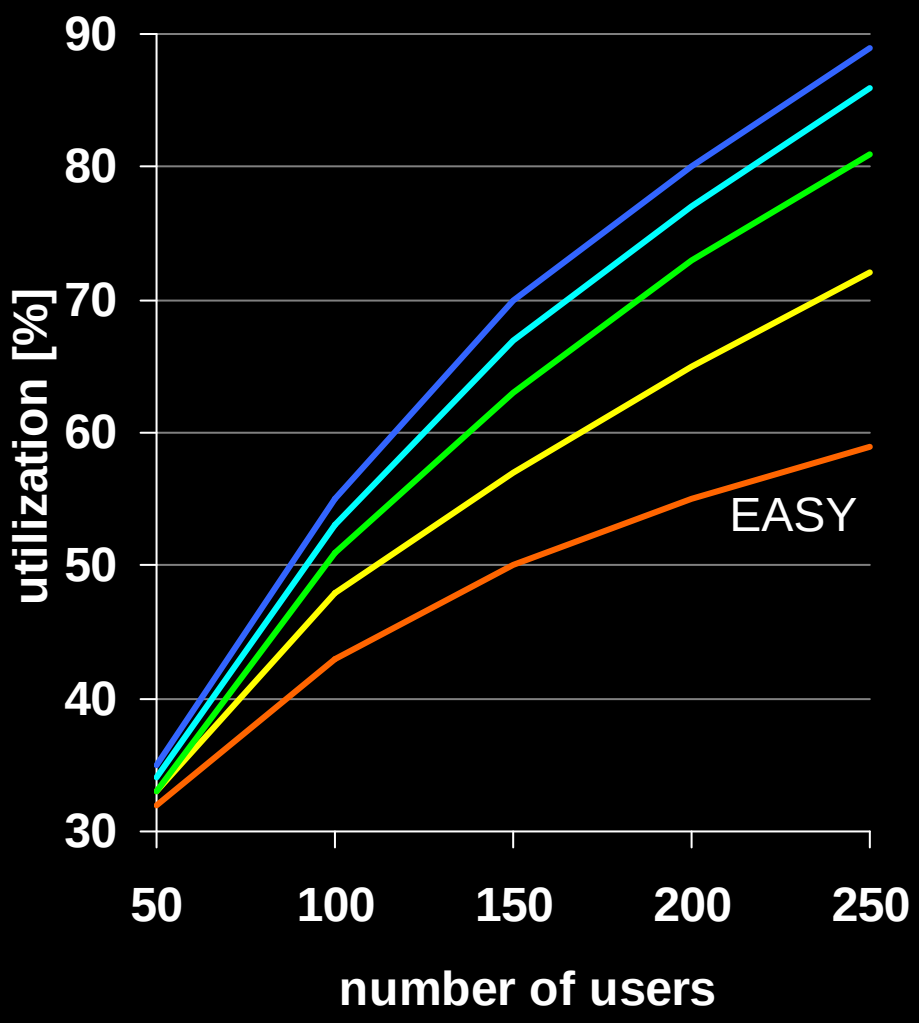
  i.e. the session will end
- A session may also end due to the daily cycle of activity

# User-Aware Scheduling

- Prioritize jobs with short expected response times
  - CREASY: CRiticality-based EASY tries to identify and prioritize critical jobs
  - An attempt to keep users satisfied and extend user sessions
- This also improves system utilization and throughput
- Balance with job seniority to prevent starvation

# Performance

- Load is expressed as the number of active users
- The higher the emphasis on responsiveness, the higher the achieved utilization
- No weight for responsiveness is the same as EASY



# Daily Cycle

- Improved throughput depends on
  - User behavior (extend session if short response time)
  - Scheduler design (prioritize based on expected short response time)
  - Daily cycle of activity
- During the day, accept higher load than can be sustained
- During the night drain the excess jobs, using resources that would otherwise remain idle

# Stationarity (not)

- Evaluations typically strive to characterize the system in its “steady state”
  - Simulation warmup
  - Analysis equations
- This implies a stationary workload model
- But a daily cycle is non-stationary by definition
- The system is always in a transient state
- Confidence intervals become meaningless

# Outline

- Packing in gang scheduling
- Feedback and its effect
- Scheduling and workloads on CMPs
- Conclusions

# CMPs

Defined by two quotations:

- “May you live in interesting times”

  (old Chinese curse)

- “If we knew what we are doing, it wouldn’t be called research”

# Interesting Times

- CMPs are a new thing
- ~30 years of research about parallel systems are largely irrelevant
  - Difference between HPC and mainstream desktop
- A flurry of new possibilities
  - Multicores
  - Manycores
  - GPGPUs
- An “undesirable” hardware development

# CMP Workloads

- We don’t know how CMPs will be used
- We don’t know what the workloads will be like
  - Degree of parallelism
  - Granularity of interactions
  - Required synchronism
  - Use of multitasking
- So we can’t design appropriate systems yet

# What, Me Worry?

- CMPs are a revolution
- Systems that incorporate them will evolve with time
  - Try out different hardware approaches
  - Different types of support from operating systems
  - Use for different applications
- Things will settle down in a few years
- Will probably be simpler than you think

# Action Items for NOW

- Include monitoring facilities in all new system designs
  - Hardware monitoring
  - System services
  - Workload characteristics
- Collect data about how real systems are being used
- Analyze such data for insights and new ideas

# Outline

- Packing in gang scheduling
- Feedback and its effect
- Scheduling and workloads on CMPs
- Conclusions

# Evaluations and Workloads

- The results of performance evaluations may depend on the workload used
- The devil is in the details
- It’s hard (or impossible) to anticipate which details are really important
- To keep in touch with reality, we need to

  look at data
- To look at data, we need to collect data and share it

# Scheduling and Workloads on CMPs

- We don’t yet have large production systems to monitor
- Everything is in a state of flux
- All options are on the table
- Workloads and schedulers will evolve together

# Thank You…

- My students
  - Edi Shmueli (PhD 2008)
  - Ahuva Mu'alem (MSc 1999)
  - David Talby (PhD 2006)
  - Dan Tsafrir (PhD 2006)
- Parallel Workloads Archive
  - CTC SP2 – Steven Hotovy and Dan Dwyer
  - SDSC Paragon – Reagan Moore and Allen Downey
  - SDSC SP2 and DataStar – Victor Hazelwood
  - SDSC Blue Horizon – Travis Earheart and Nancy Wilkins-Diehr
  - LANL CM5 – Curt Canada
  - LANL O2K – Fabrizio Petrini
  - HPC2N cluster – Ake Sandgren and Michael Jack
  - LLNL uBGL – Moe Jette