GUROBI
OPTIMIZATION

The Latest Advances in the Gurobi Optimizer

Recent Improvements

- Performance improvements
- New approaches for handling non-linear objective functions
- New remote computing capabilities
- New rich examples

New Rich Examples

Demo

- Rich example demo…

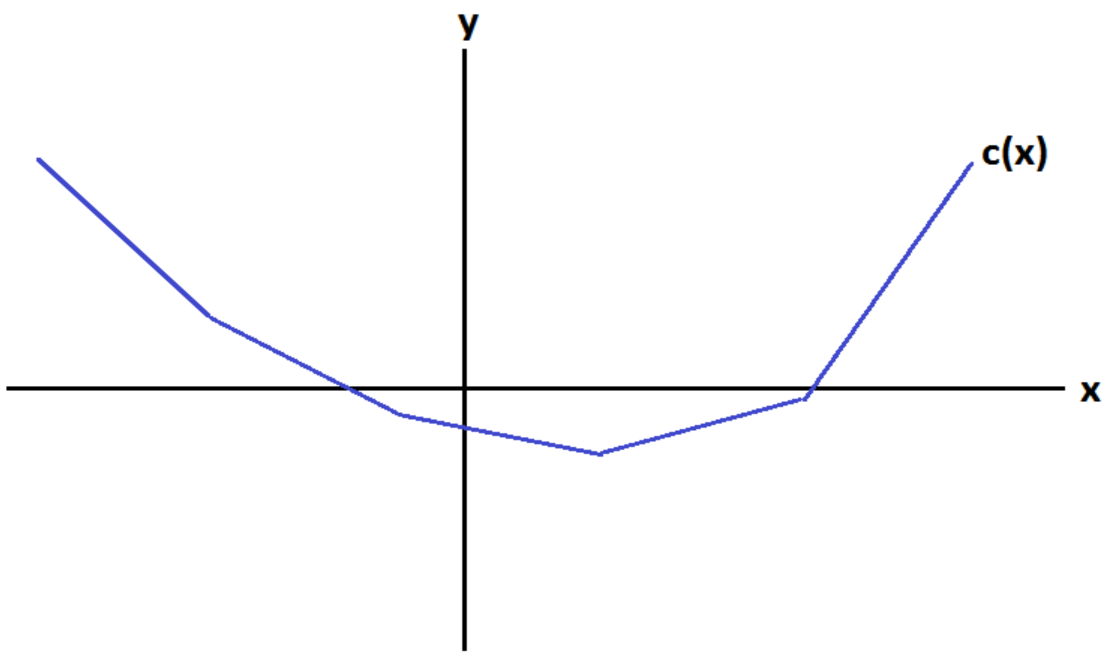
Piecewise-Linear Objective Functions

Motivation

- Non-linear objective functions – conventional wisdom:
  - Best way to model them is typically a piecewise-linear approximation
- Traditional PWL approach
  - One optimization variable for each piece
  - Increased accuracy -> more pieces -> rapid growth in problem size
- Objective of our "new" approach:
  - Extend simplex algorithm (primal and dual)
  - Handle PWL terms directly (no extra variables)
- New API to make it easy to express PWL functions

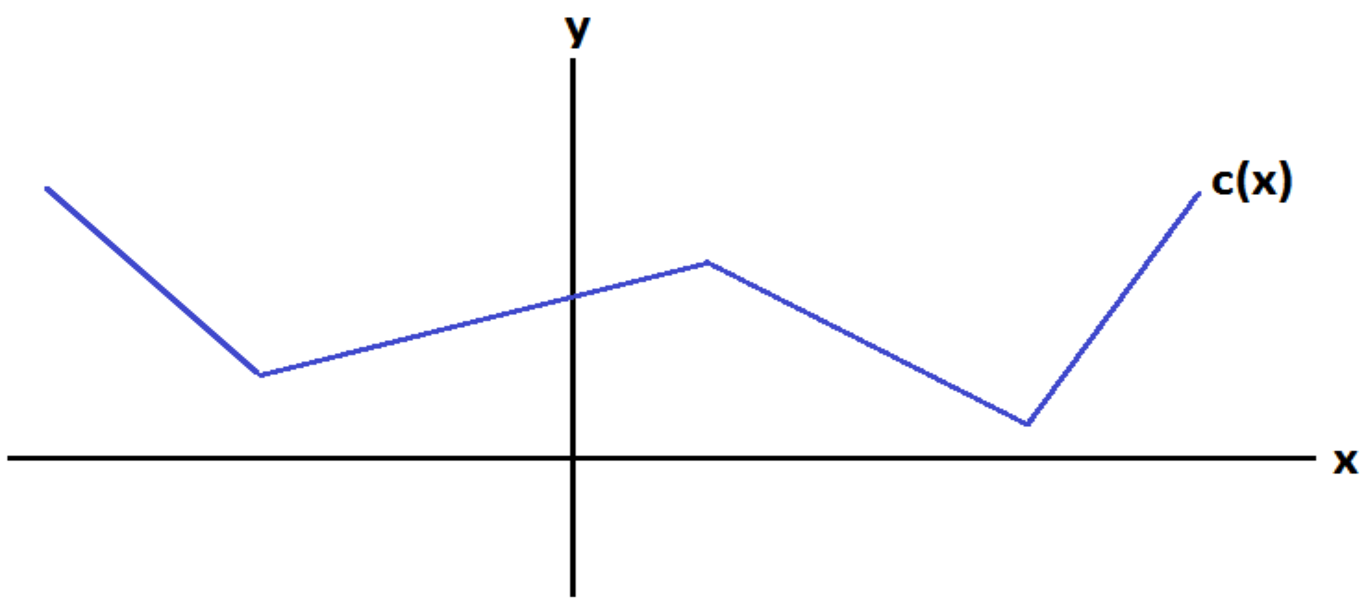
Convex PWL Objective Function

- Continuous
- Slopes are non-decreasing



Non-convex PWL Function

- Continuous non-convex function



PWL Simplex

- Extend both primal and dual simplex to support separable convex PWL objective
  - Primal PWL simplex, Fourer and Marsten, 1992
  - Dual PWL simplex, no reference

- Main difference between standard and PWL simplex
  - Ratio test
    - Depends on efficient implementation of the median algorithm

- Dual PWL simplex ~3X faster than primal PWL simplex

Computational Test, QP

- PWL approximation of QP objective
- Quality of approximation depends on number of pieces:
  - 100 pieces: 2 digits of accuracy
  - 1000 pieces: 4 digits of accuracy
- Compare PWL simplex against simplex on one-variable-per-piece model (for QP models that require >1s):

Pieces	Speedup
100	2.0X
1000	6.1X

- Compare PWL simplex (w/1000 PWL pieces) against QP simplex:
  - PWL only 2.3X slower than specialized QP simplex algorithm

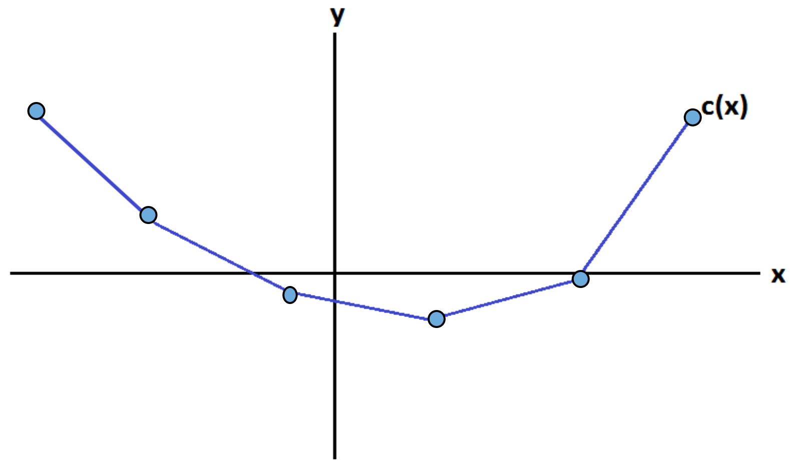
Gurobi PWL APIs

- Point representation
  - List points that define PWL function
    - Using (x,y) coordinates
  - C routines

```
GRBsetpwlobj(model, var,
             npoints, x, y)
```

  - OO APIs, e.g. C++

```
Model.setPWLObj
```



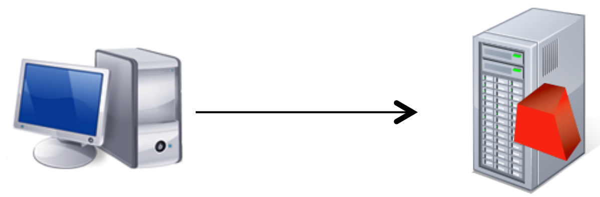
- Attributes
  - `NumPWLObjVars`
  - `PWLObjCvx`

Remote Computing – Compute Server, Distributed Optimization, and Gurobi Cloud

Gurobi Compute Server

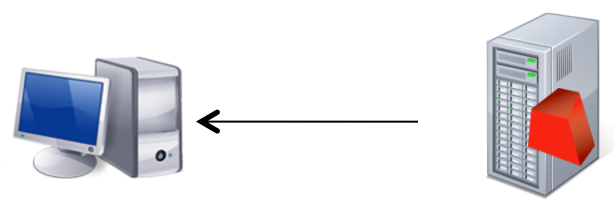
- Client computer uses Gurobi API to build model

- Client computer passes model data to server



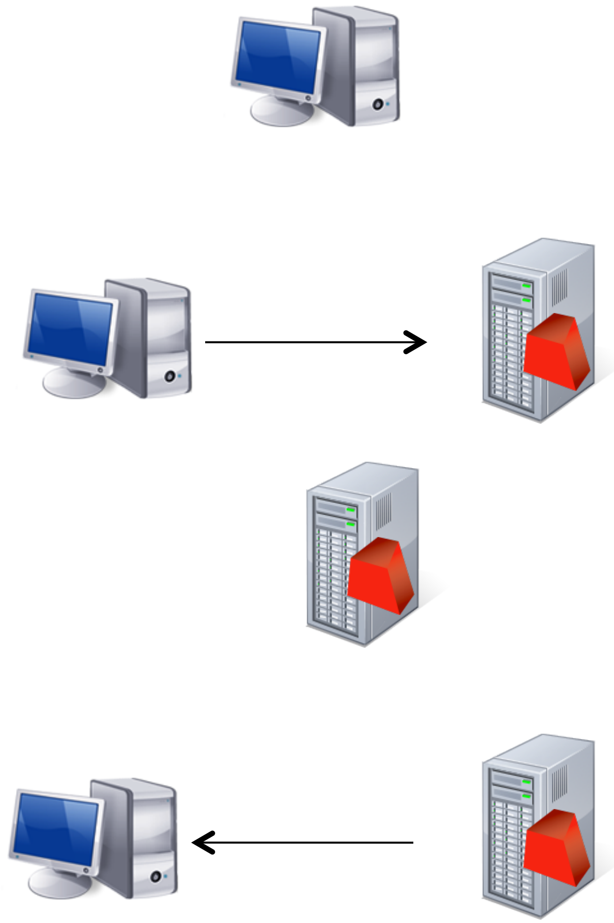
- Server solves the model

- Result values returned to client computer



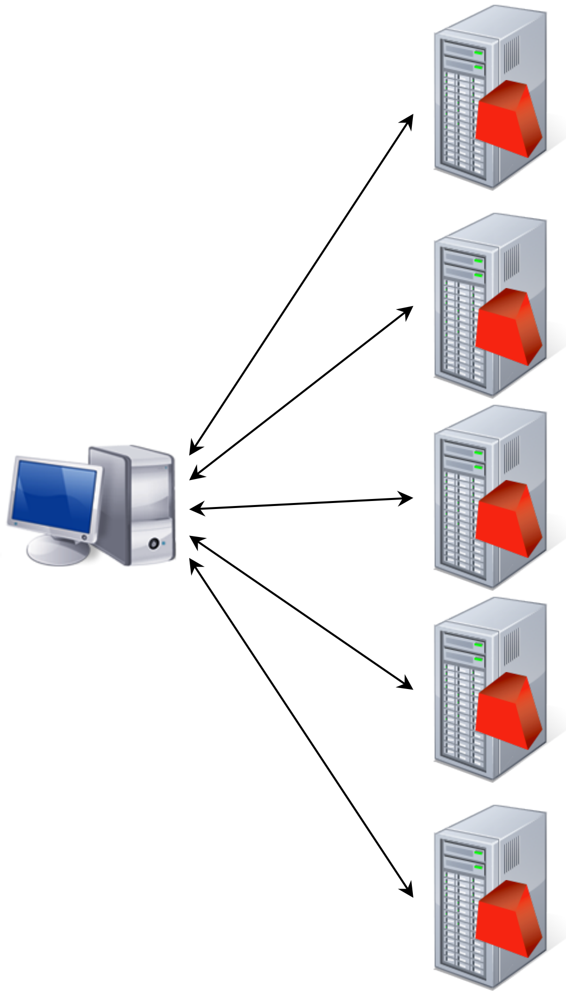
Use Case – Thin Client

- Gurobi Compute Server
  - Offload optimization from slow clients to fast servers
  - One customer example:
    - 4X improvement in MIP solution times
    - Two servers (and two Gurobi licenses) handle all optimization jobs for 50+ clients
    - No specialized implementation work required



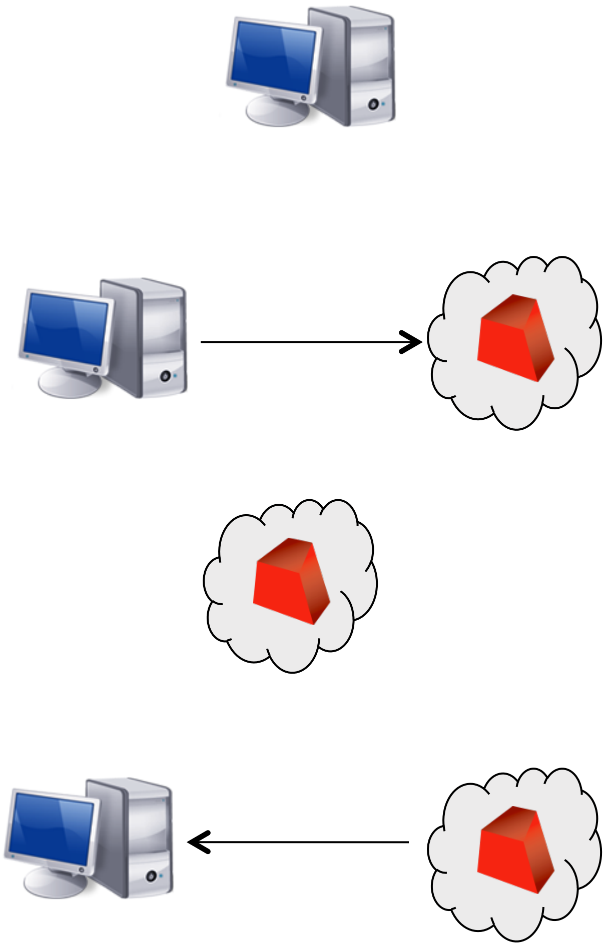
Use Case – Distributed Computing

- One manager offloads computation to multiple workers
- Multiple machines cooperate to solve a single MIP model



Use Case – Cloud Computing

- Gurobi Cloud hosted on Amazon EC2
- Launch an instance (or 100)
- Offload computation to it over the Internet
- Pay by the hour



Distributed MIP

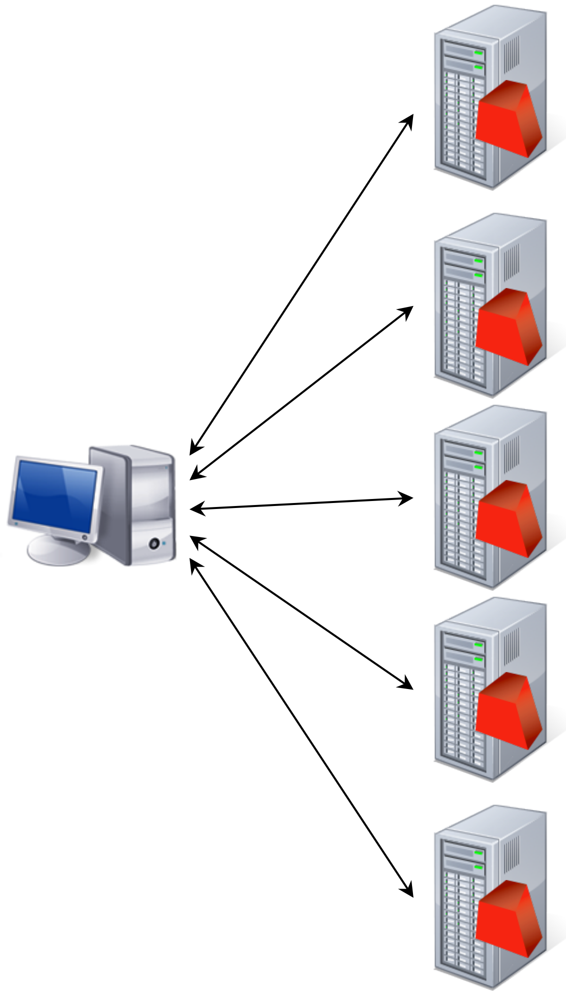
Distributed MIP Architecture

- Manager-worker paradigm

- Manager
  - Send model to all workers
  - Track dual bound and worker node counts
  - Rebalance search tree to put useful load on all workers
  - Distribute feasible solutions

- Workers
  - Solve MIP nodes
  - Report status and feasible solutions

- Deterministic synchronization



Distributed MIP Phases

- Racing ramp-up phase
  - Distributed concurrent MIP
    - Solve same problem individually on each worker, using different parameter settings
    - Stop when problem is solved or “enough” nodes are explored
    - Choose a “winner” – worker that made the most progress
- Main phase
  - Discard all worker trees except the winner's
  - Collect active nodes from winner, distribute them among now idle workers
  - Periodically synchronize to rebalance load

Distributed MIP Performance

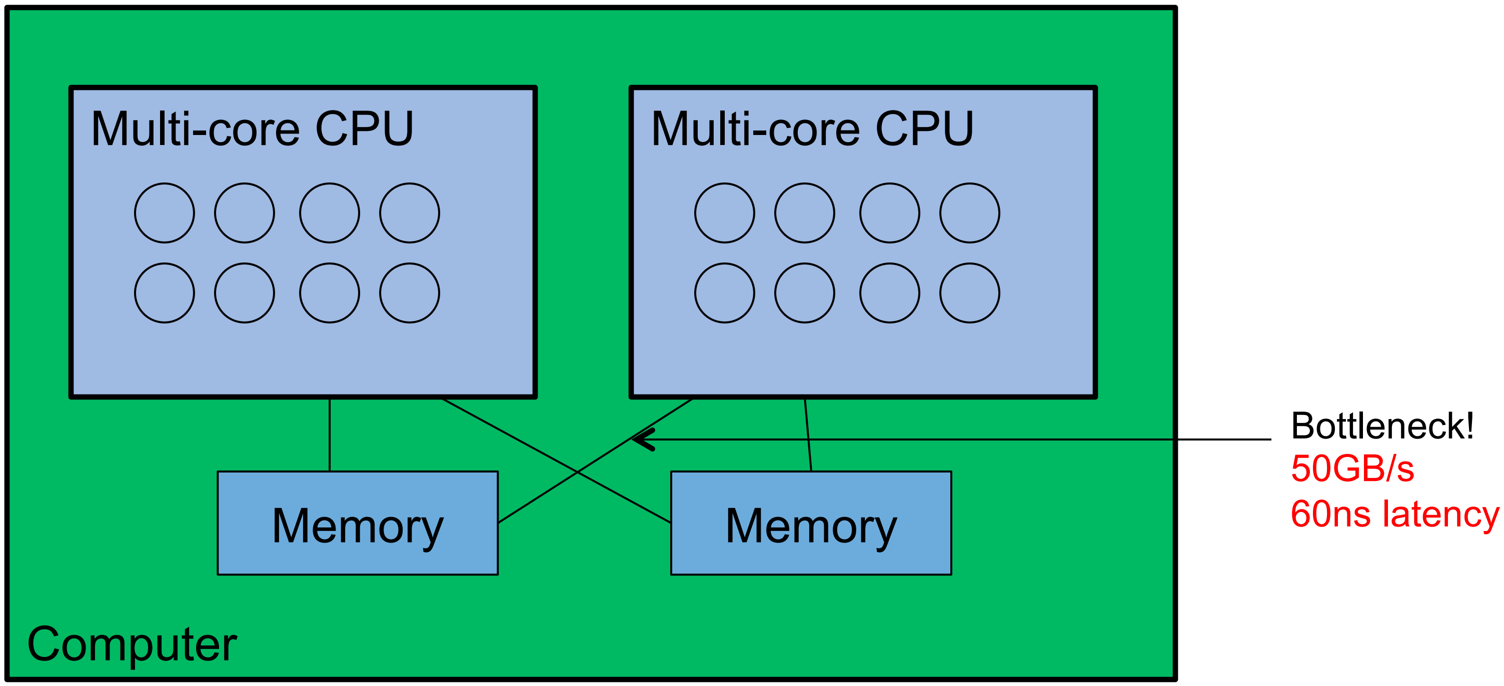
16 Cores vs. 16 Cores

- Consider two different tests using 16 cores:
  - On a 16-core machine:
    - Run the standard parallel code on all 16 cores
  - On four 4-way machines:
    - Run the distributed code

- Which gives better results?

Parallel MIP on 1 Machine

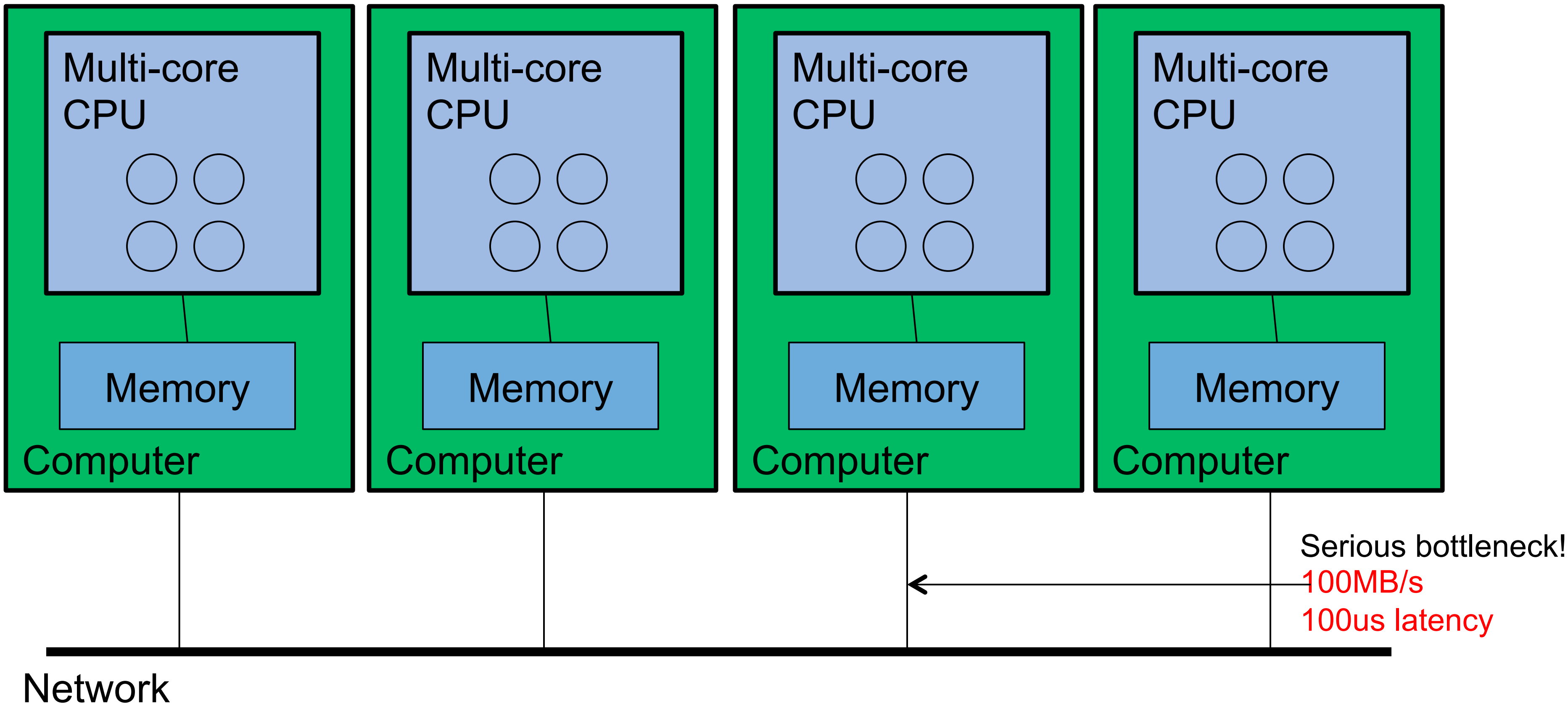
- Use one 16-core machine:



GUROBI OPTIMIZATION

Distributed MIP on 4 machines

- Use four 4-core machines



Performance Results

- Comparing one 16-core machine against four 4-core machines (MIPLIB 2010, baseline is single-machine, 4-core run)…

Config	>1s	>100s
One 16-core machine	1.57X	2.00X
Four 4-core machines	1.43X	2.09X

- Given a choice…
  - Comparable mean speedups
  - Other factors…
    - Cost: four 4-core machines are much cheaper
    - Admin: more work to admin 4 machines

Distributed Algorithms in 6.0

- MIPLIB 2010 benchmark set
  - Intel Xeon E3-1240v3 (4-core) CPU
  - Compare against 'standard' code on 1 machine

Machines	>1s			>100s		
	Wins	Losses	Speedup	Wins	Losses	Speedup
2	40	16	1.14X	20	7	1.27X
4	50	17	1.43X	25	2	2.09X
8	53	19	1.53X	25	2	2.87X
16	52	25	1.58X	25	3	3.15X

Some Big Wins

- Model *seymour*
  - Hard set covering model from MIPLIB 2010
  - 4944 constraints, 1372 (binary) variables, 33K non-zeroes

Machines	Nodes	Time (s)	Speedup
1	476,642	9,267s	–
16	1,314,062	1,015s	9.1X
32	1,321,048	633s	14.6X

Gurobi Distributed MIP

- Makes huge improvements in performance possible
- Mean performance improvements are significant but not huge
  - Much better than distributed concurrent
  - As effective as adding more cores to one box
- Effectively exploiting parallelism remains:
  - A difficult problem
  - A focus of ours

Gurobi Instant Cloud

Why Use The Cloud For Optimization?

- Pay just for what you use
  - Short-term projects
  - Occasional use
  - Meet a peak in demand

- No software or hardware to purchase or configure

- Get many fast computers quickly
  - Especially valuable for distributed optimization

- Increased robustness
  - Get computers in multiple locations
  - World-class management of computer center

Demo

- Gurobi Instant Cloud demo…

Pricing For Gurobi Instant Cloud

- Pay for your Cloud License plus the Machine Use

- Gurobi Cloud License

	Basic	Compute Server
Features	• Single-use license • No distributed algorithms • No job queuing	• Multiple concurrent uses • Distributed algorithms • Load balancing, job queuing
Cost	$10/hr or $1500/750 hrs/month	$20/hr or $3000/750 hrs/month
Minimum charge	30 minutes ($5)	30 minutes ($10)

  - Distributed worker: no cost for Gurobi license
  - No commitment – monthly price applies automatically at 150 hrs in calendar month

- Machine use
  - Typically ~$0.50/hr, rounded to next hour
  - Varies based on machine type and location

GUROBI OPTIMIZATION

Performance Improvements

Performance Improvements (6.0 vs 5.6)

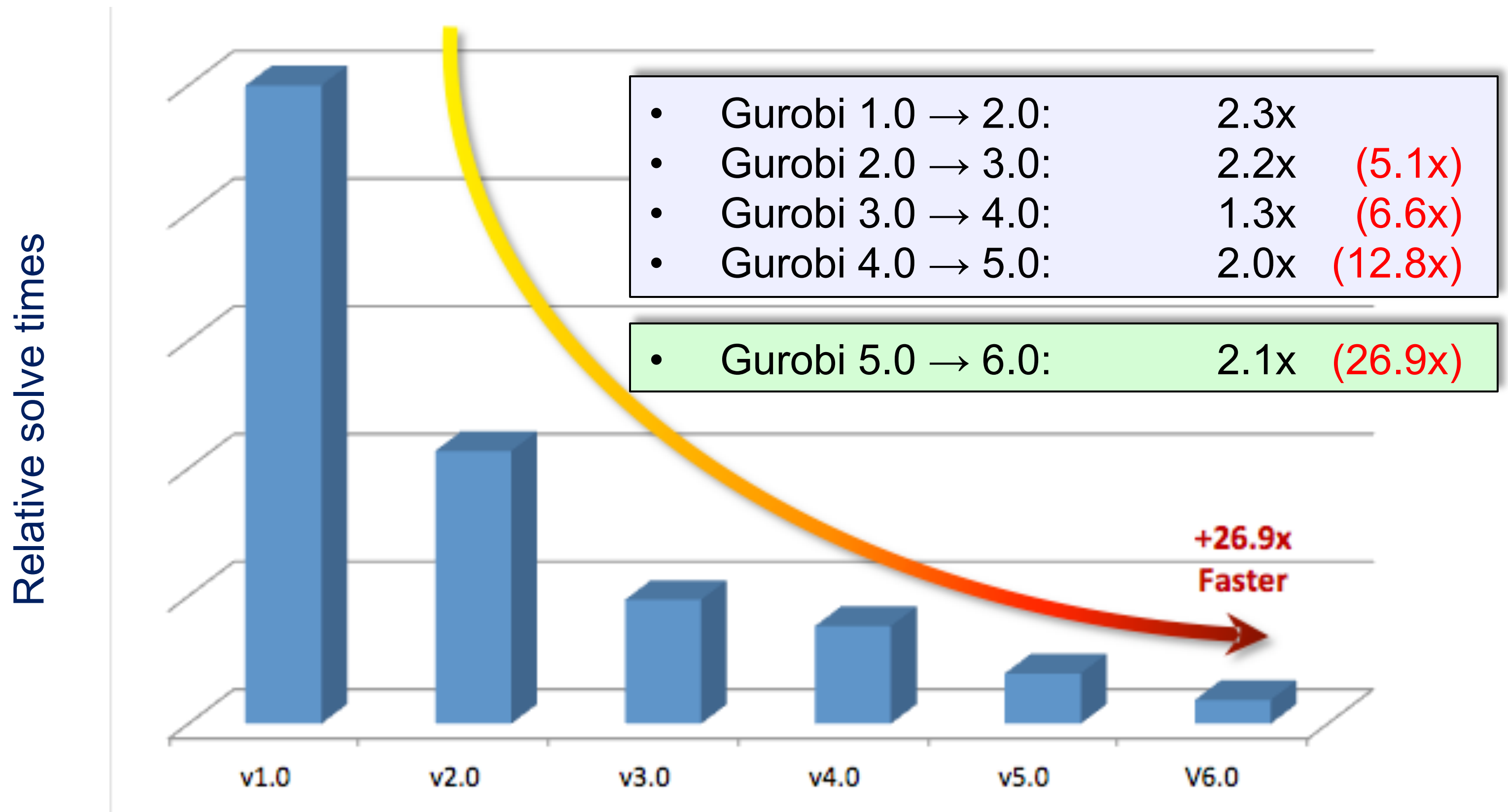
Problem Class	> 1 sec	> 100 sec
	Speed-up	Speed-up
INTERNAL		
LP	1.06x	1.07x
MIP	1.17x	1.27x
MIQP	1.18x	1.20x
MIQCP	1.28x	2.07x
EXTERNAL		
Mittelmann "Easy"	1.15x	1.23x
Mittelmann Optimality Benchmark*	1.12x	1.09x

Notes:

1. 4 threads, 10000s time limit
2. * Average over 5 random seeds

MIP Performance Improvements

- Version-to-version improvements:
  (Geometric mean speedup for models in our internal model set where at least one of the solvers takes more than 100s to solve)



GUROBI OPTIMIZATION

Gurobi 6.5 Coming Soon

- New release coming soon
- Significant improvements

Thank You