Small verse Large

The Performance Tester Paradox

The Paradox

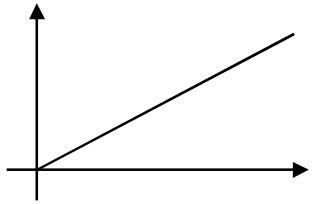
- Why do people want performance testing?
  - To stop performance problems in production
- How do we ensure this?
  - Performance test with
    - Realistic workload
    - Realistic test environment (production like/size)
- How much do they want to pay?
  - Not enough!

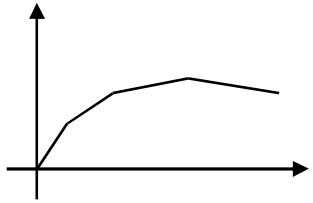
What do other People Say

- Recent message board thread asked
  - "How one should scale out the Performance results of Test env to that of Production?"
  - http://www.linkedin.com/groupAnswers?viewQuestionAndAnswers=&gid=104350&discussionID=1934185&sik=1240080932163&trk=ug_qa_q&goback=%2Eana_104350_1240080932163_3_1
- Some responses
  - "performance of the Test env should match that of the production. Otherwise there is no point"
  - "extrapolation is dangerous"
  - "Can we not extrapolate the results linearly "
  - "You need a model"
  - "It works by scaling load factors to the same percentage"

Terminology

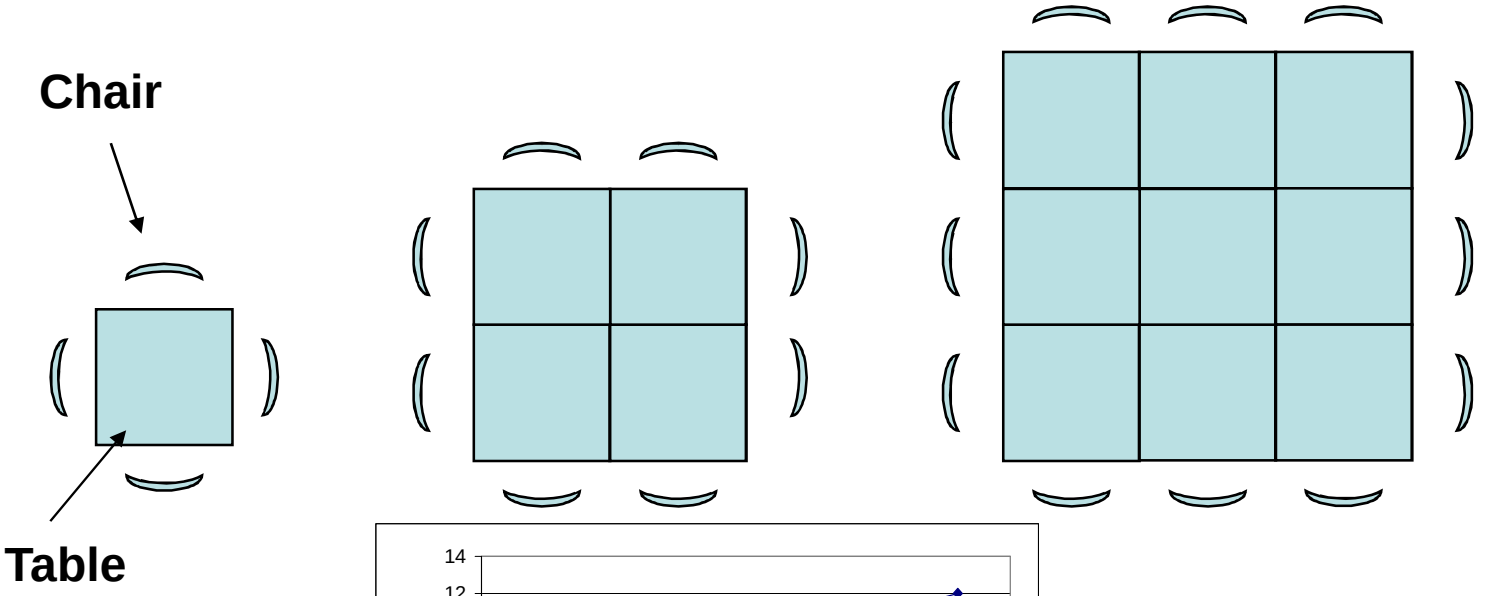
- Hardware Scalability
  - Horizontal
    - Adding additional boxes to increase throughput
  - Vertical
    - Increasing the size of a box to increase throughput
- Linear Scaling
  - Improvement is in proportional to the increase
- Nonlinear Scaling
  - Improvement is not in proportion to increase
    - Sub linear: diseconomies of scale
    - Super linear: economies of scale

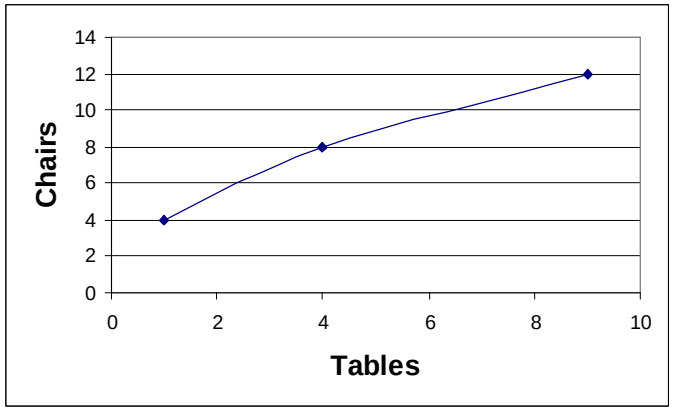




Dinner Party Chair Example

With a standard table size how do you increase the number of diners?





Terminology - 2

- Bottleneck
  - Area of a system that constrains the performance of the system
- Hard Bottleneck
  - Hardware resource that is fully utilized
    - CPU, Disk, etc
  - Typically bottleneck is the device operating “on the knee of the curve”
- Soft Bottleneck
  - Software constraint that inhibits the performance of the system
    - DB connection pool
    - Thread limit

Sizing/Extrapolation Techniques

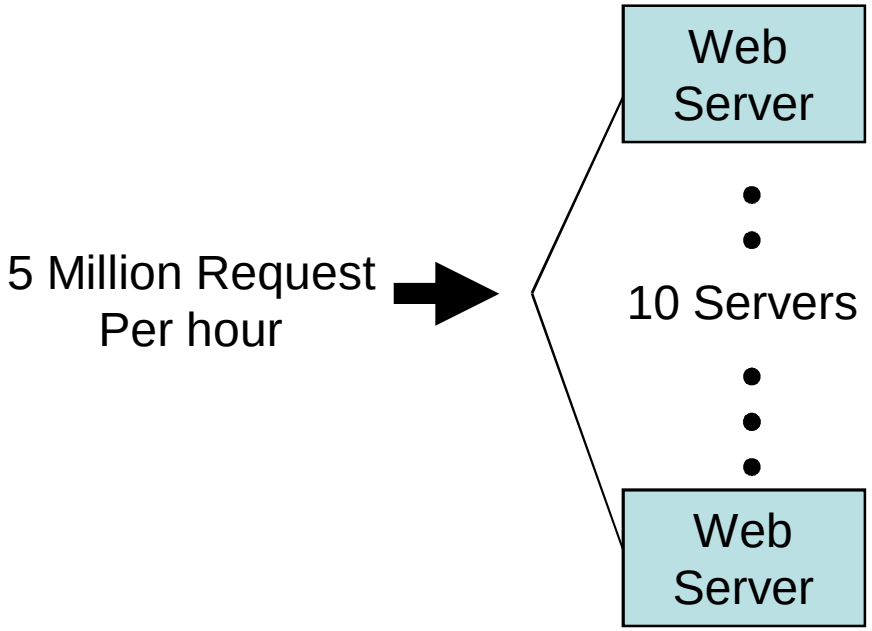
- Factoring
- Dimensioning
- Modelling
- Flipping
- Full Scale

Factoring

- What is it?
  - Test environment is a factor smaller than production
  - Test workload is reduced by same factor
  - Test Response times expected to be the same as production
- Two Examples
  - Web Farm
  - Oracle Database

Web Farm Example Production

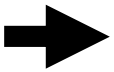
- A request can be processed by any single web server
- No data dependencies
- Each Web Server is identical hardware



Web Farm Example Test

- 1/10th Test Environment
- Test Web Server is same hardware as production
- Test Web Server has same software/data

1/2 Million Request
Per hour

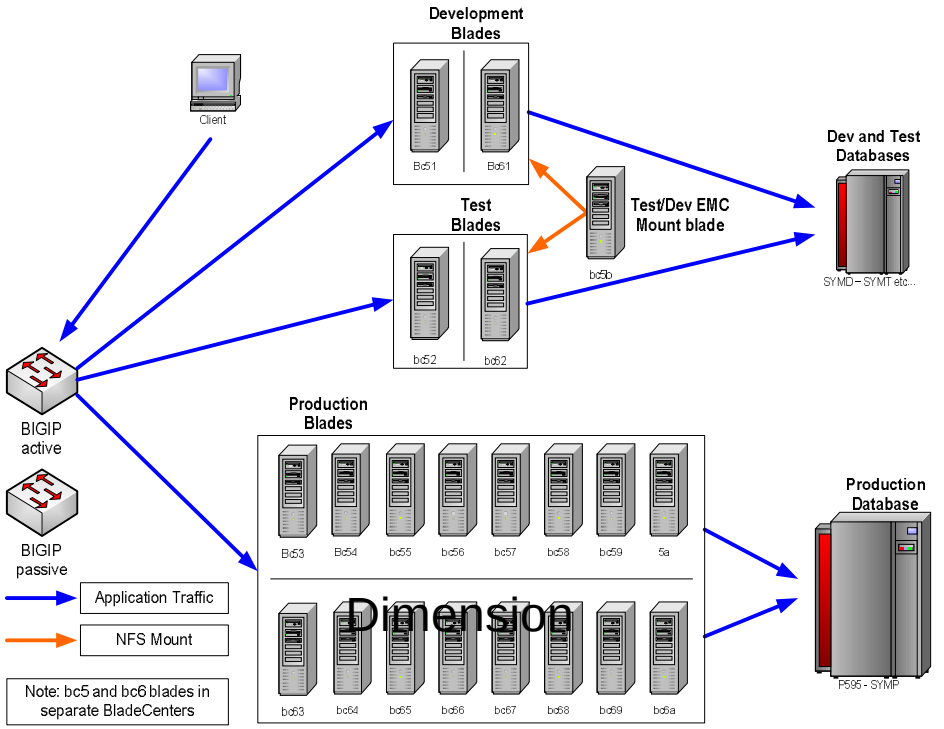


Web
Server

Oracle Database

AIX LPAR Allows DB Server CPU rating to be changed ("only thing changed*)

- Large
  - 16 App Servers
  - DB 8 CPU on P595, 99 G Memory
- Small
  - 16 App Servers
  - DB 2 CPU on P595, 25 G Memory
- Workload scaled by 1/4



***Oracle Parameters set to reflect new memory configuration**

Results

Metric	Small	Large
Name Search Average Response Time	0.64	0.30
Name Search 95% Percentile	2.43	1.03
Search per second	1.17	4.69
DB CPU Utilization	36.32	17.42

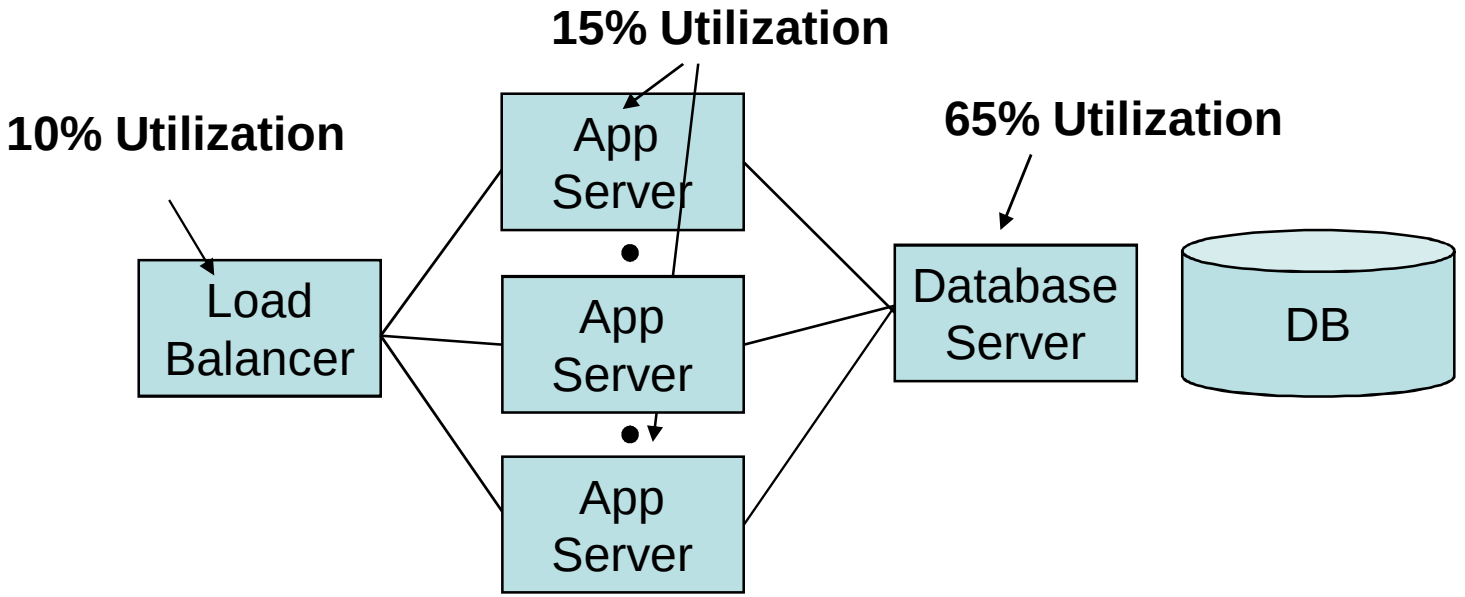
Factoring Summary

- Simplest to calculate
- Works well for
  - large horizontally scaled architectures
  - No shared data source
  - Same hardware specification in both environment
- Reality is that there are often shared components
  - Network
  - Load Balancer

Dimensioning

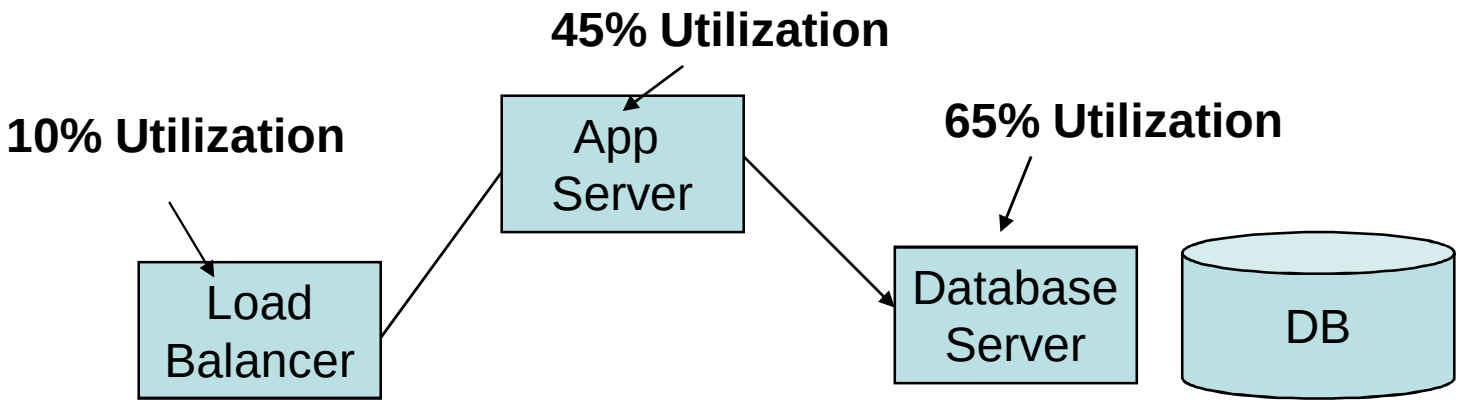
- All systems have bottleneck
  - Often a critical bottleneck device per system
- Need to identify critical bottleneck
  - Often central components shared by many others
  - Can change with workload
- Found by
  - Analysis
  - Performance tests
- Once identified
  - Replicate bottleneck component
  - Reduce less critical components

Example: Production Architecture



- **Central Database Server has highest utilization**
- **App Server performance test show good performance at higher load**
- **Load Balancer similar to those used in other configurations**

Example: 3 Tier Test Architecture

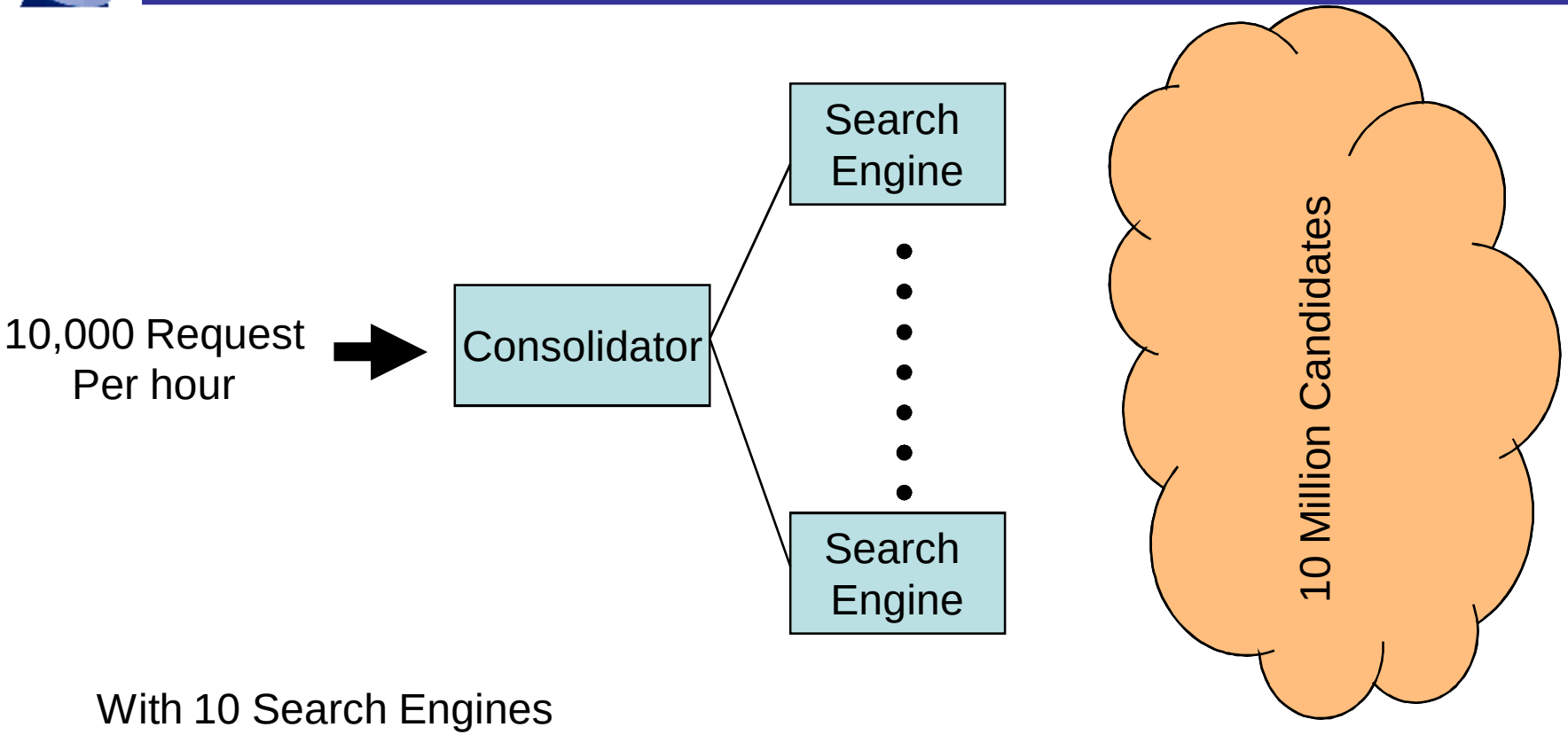


- **Database server kept the same but 1/3 number of application servers used**

Data Dimensioning

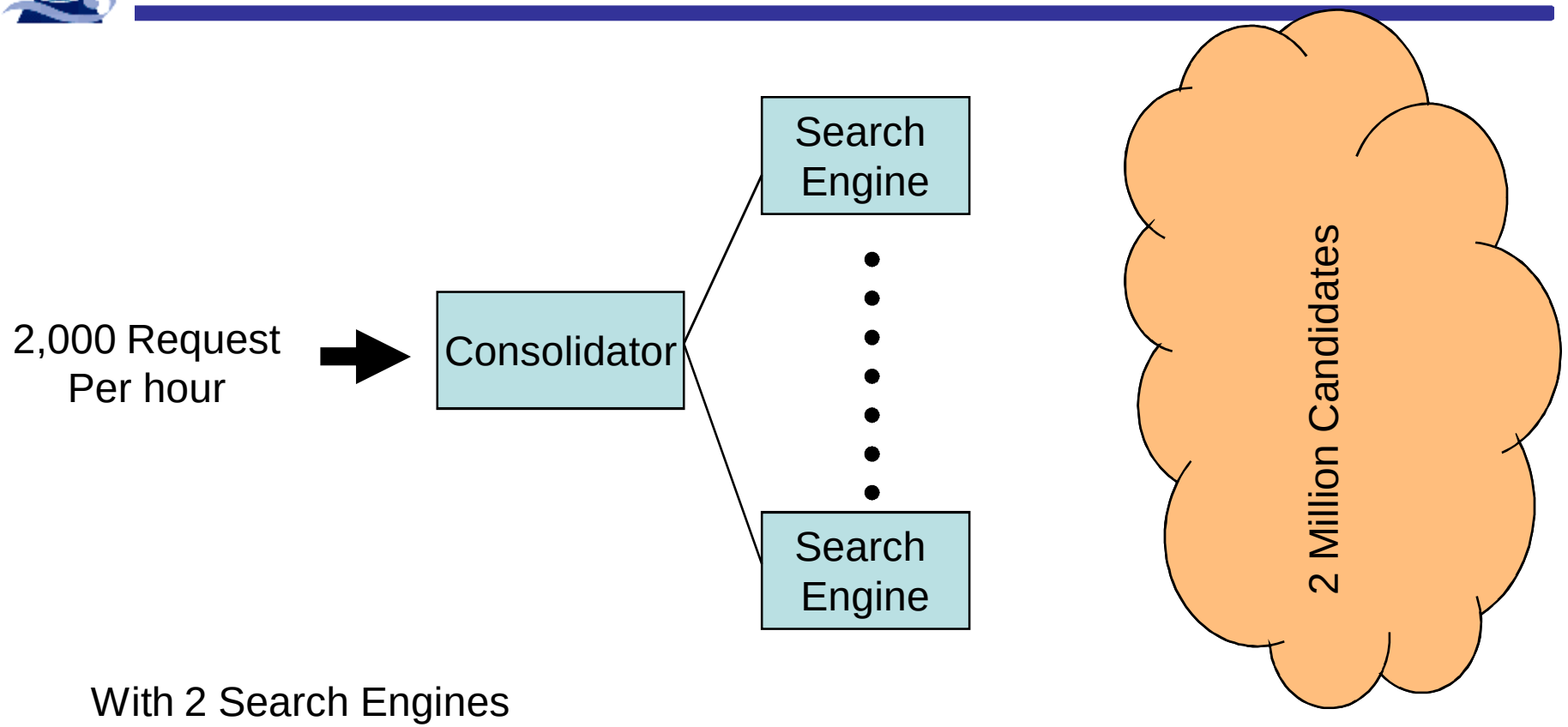
- Reducing the hardware components means that potentially computation is increased on components due to the system data being shared between less components.
- Consideration must be given to this effect.

Full Sized Architecture



With 10 Search Engines
Each Search Engine searches 1 Million Candidates
Each Search Engine processes 10K * 1M Search Computation per hour
Consolidator process 10,000 requests and consolidates 10 results per request

Test Architecture (1/5)



With 2 Search Engines
Each Search Engine searches 5 Million Candidates
Each Search Engine processes 2K * 1M Search Computation per hour
Consolidator process 2,000 requests and consolidates 2 results per request

Dimensioning Summary

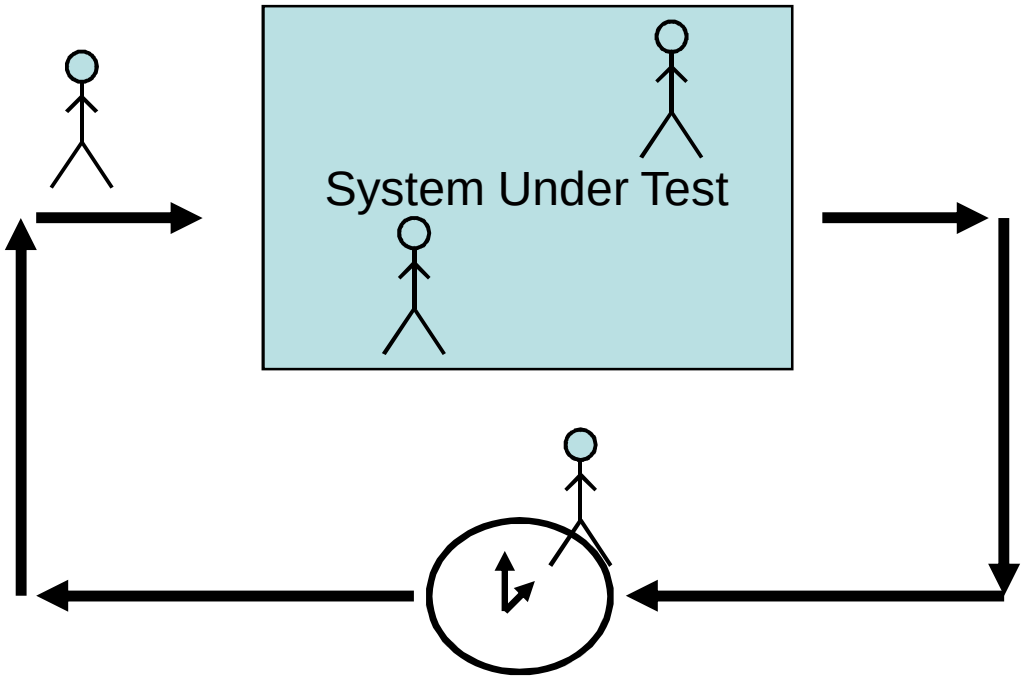
- Assumption and knowledge of system is needed
- An iterative approach is needed to boundary test
- Works well for
  - large understood architectures
  - Architectures with a single very dominate component.
  - No radical changes are to be tested.

Modelling

- What is a model?
  - An abstract representation of a system
  - Allows the performance of production to be predicted
- Modelling approach
  - Create a “model” of the test environment
  - Verify the model with test results
  - Change to production configuration and predict results
  - Refine model, Repeat
- What types of model?
  - Mathematical
    - Linear
    - Queue
  - Simulation

Closed System

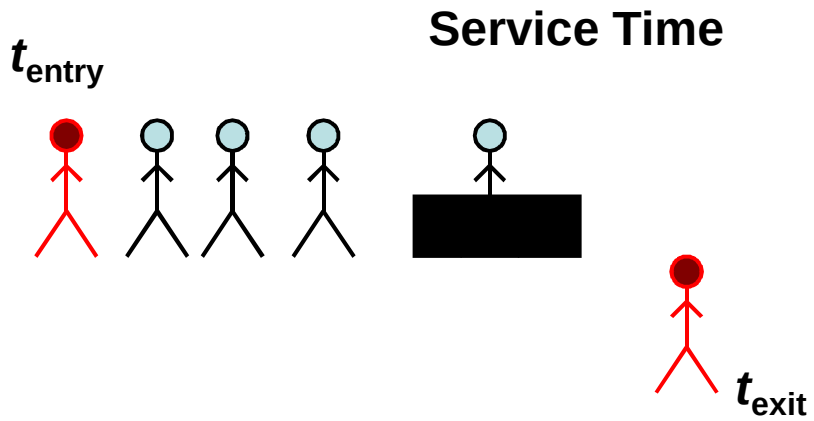
- A finite population loops through the system
- Similar to how a load testing tool generates workload



Closed Loop Modelling

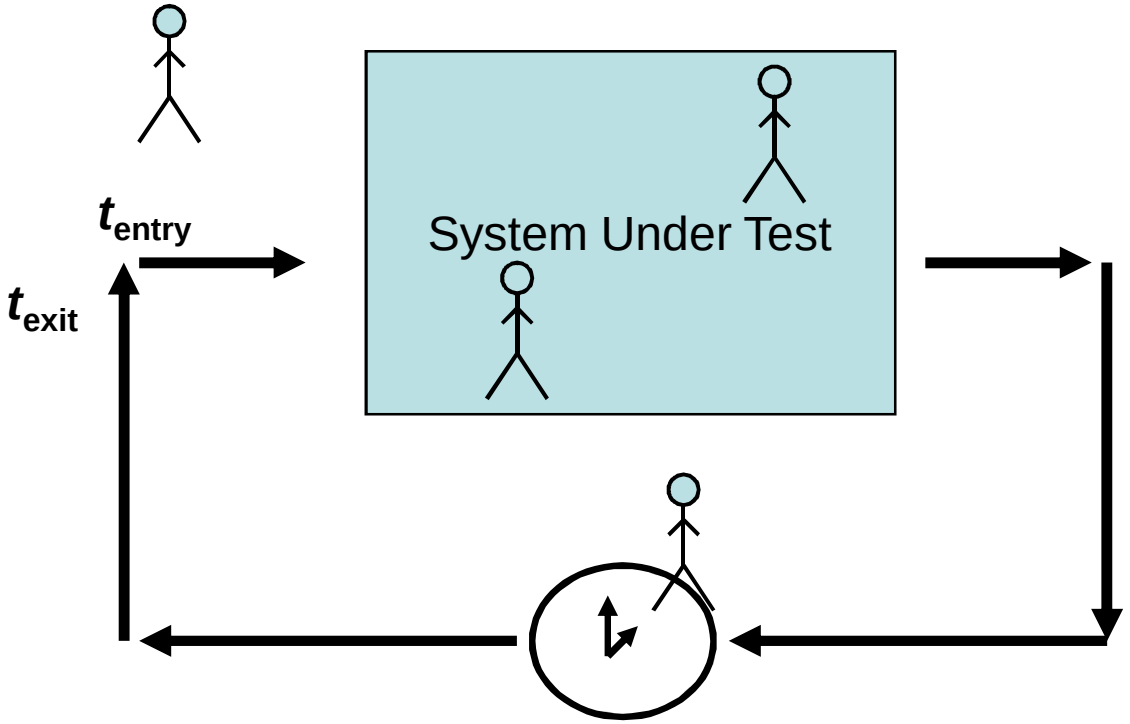
- Uses three equations..
  - Residence Time
    - Time spent queuing and being served
  - Throughput
    - Number of user processed per time unit
  - Queue length
    - Number of people in the queue
- Uses Mean Value Analysis
  - It is based on the arrival theorem, which states that when one customer in an *N*-customer closed system arrives at a service facility he/she observes the rest of the system to be in the equilibrium state for a system with $N - 1$ customers.

Residence Time



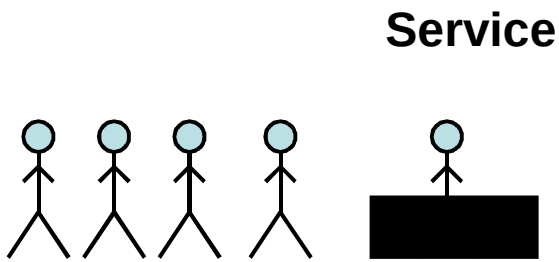
- Defined as
  - Time spent waiting and being served
  - t_{exit} - t_{entry}
- The time spent with the server is know as service time
- Residence Time = Queue Length * Service Time + Service Time
- Residence Time = Service Time (1 + Queue Length)

Throughput



Throughput = number of users / (t_{exit} – t_{entry})

Queue Length

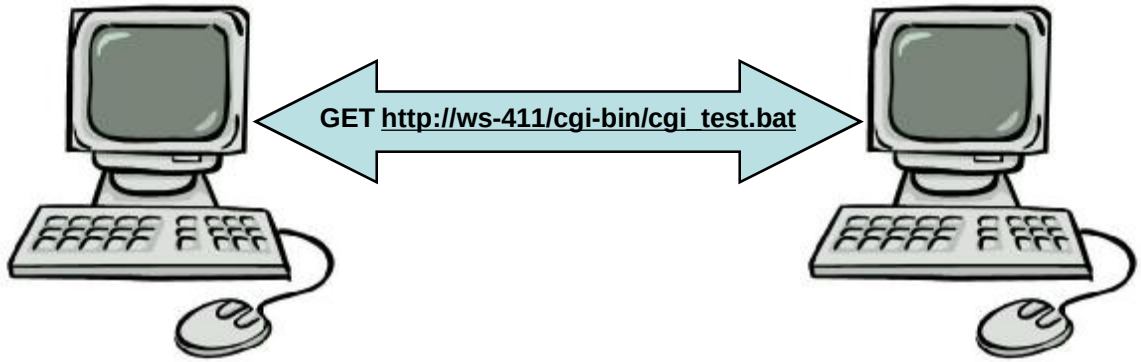


- Defined as
  - Throughput multiplied by Residence Time

Test Setup

Load Generator

Web Server



- Load Runner
- Simple GET request
- 2 second think time

- Jibble Webserver
- CGI Script

www.jibble.org

From test populate spreadsheet

Number of Users	Web Response Time	Delay	Throughput	Queue Length
1	0.018	2	0.495	0.009

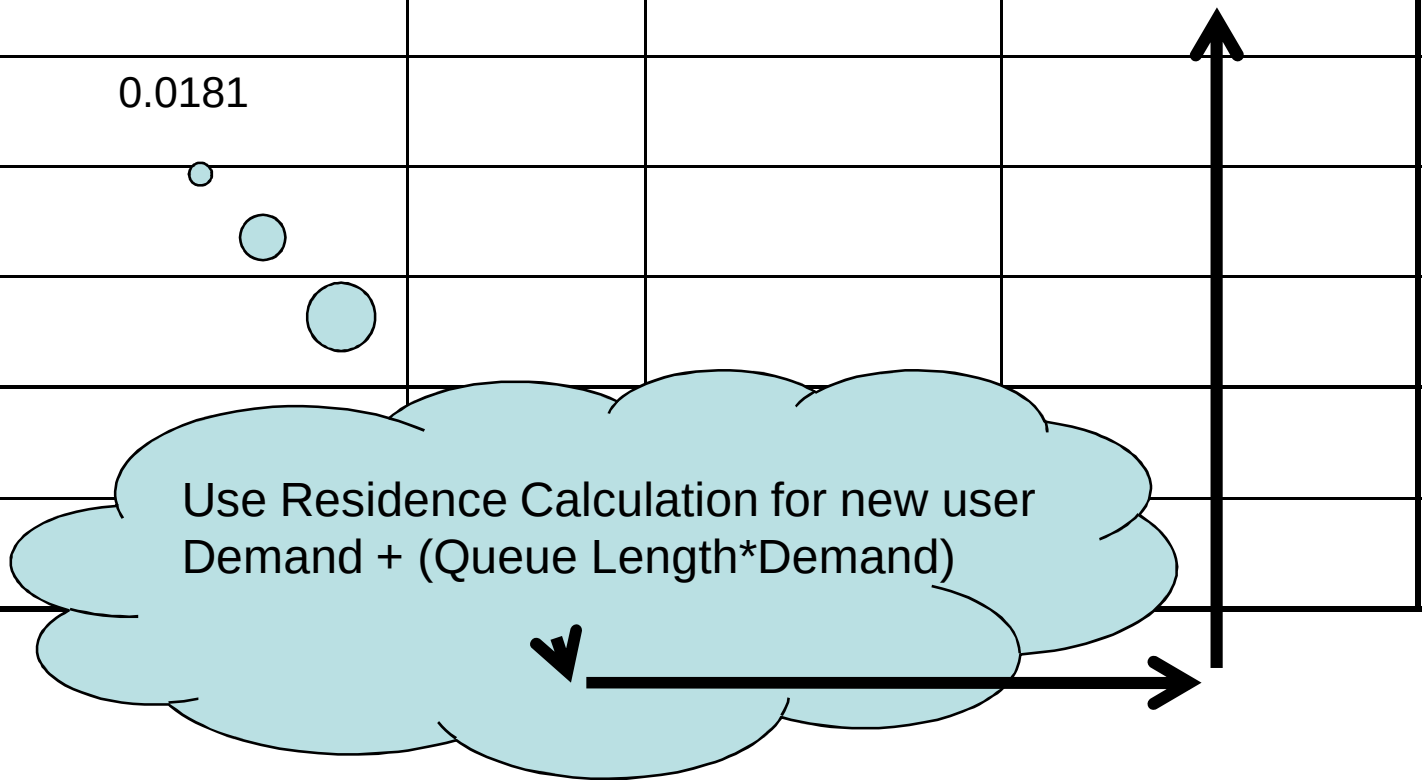
Measured Response Time

Transaction per Second
Number of Users /
(Web + Delay Resp Time)
1 / (0.018+2)

Web Response Time * Throughput

Add Next user Spreadsheet

Number of Users	Web Response Time	Delay	Throughput	Queue Length
1	0.018	2	0.495	0.009
2	0.0181			

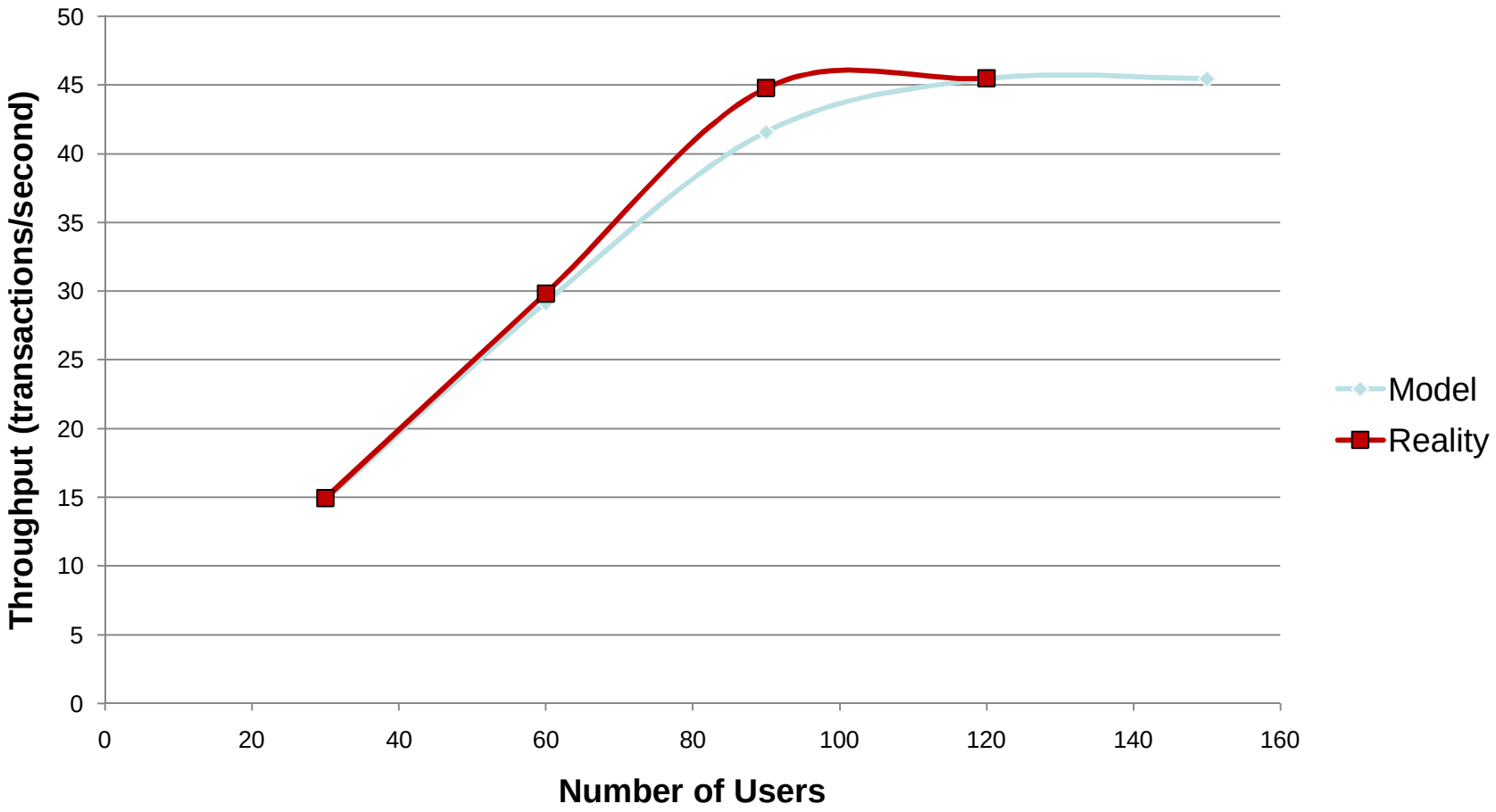


Add Next user Spreadsheet

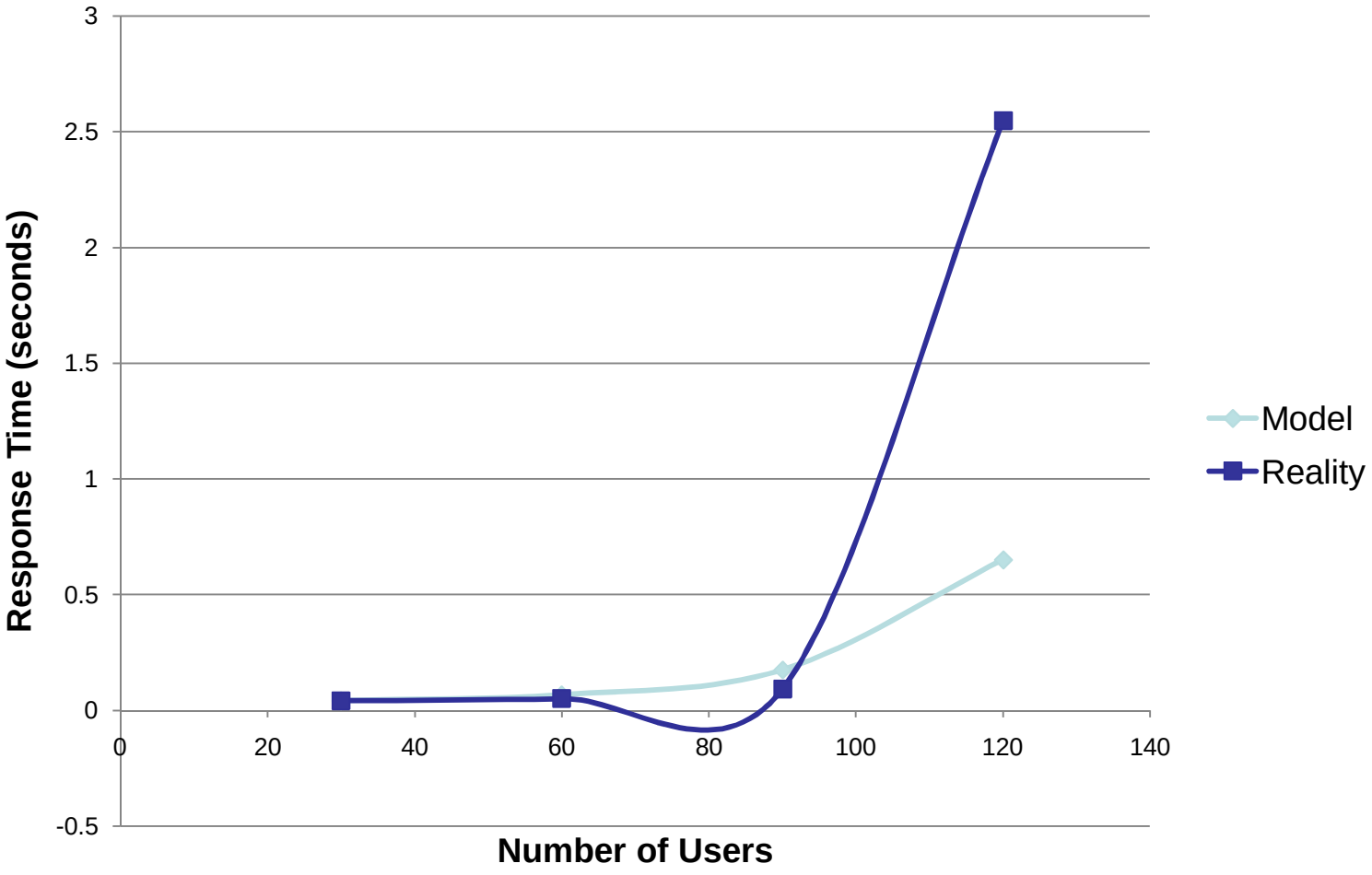
Number of Users	Web Response Time	Delay	Throughput	Queue Length
1	0.018	2	0.495	0.009
2	0.0181	2	0.98	0.017
3	0.0183	2	1.48	0.027
4	0.0185	2	1.98	0.037
5	0.0186	2	2.47	0.046

So how was the model predict reality

Throughput, Model vs Reality



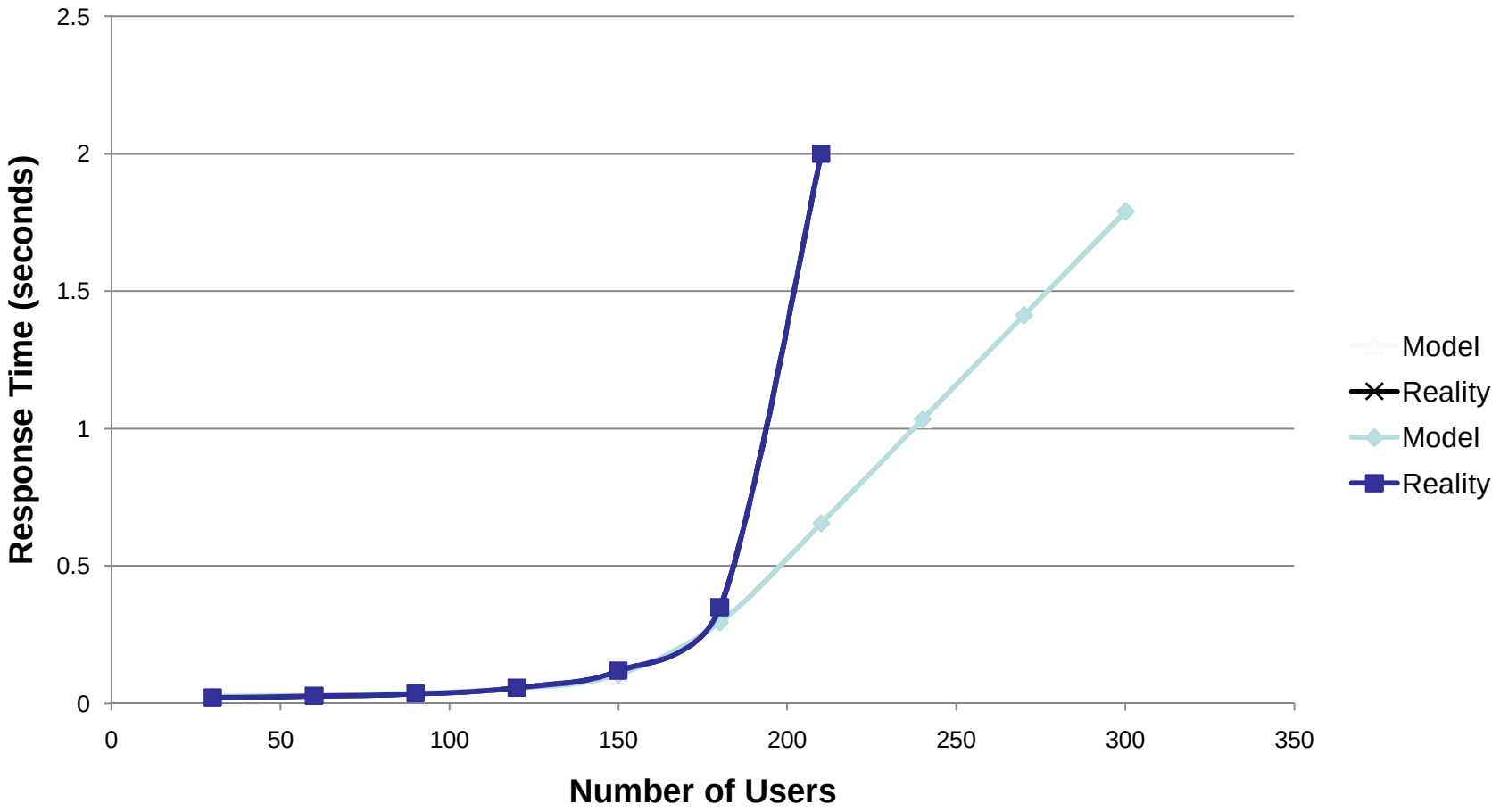
Response Time, Model vs Reality



Lets Extrapolate

- Installed Jibble on more powerful PC
  - 30% CPU increase

Results on Faster PC



Modelling Tools

- Model can grow quickly in complexity
  - Multiple queues
  - Multiple classes
  - Multi server
- Spreadsheet from Daniel A. Menasce books
  - http://www.cs.gmu.edu/~menasce/webservices/efiles.html
- Commercial Tools
  - Hyperformix
  - Teamquest Model
  - BMC Perform/predict

Model Summary

- Can be used to help extrapolate results in complex architectures
  - Good at getting utilization right
  - Response times are more difficult to estimate
- Models take time and skill to build
  - Don’t under estimate the data collection and validation steps

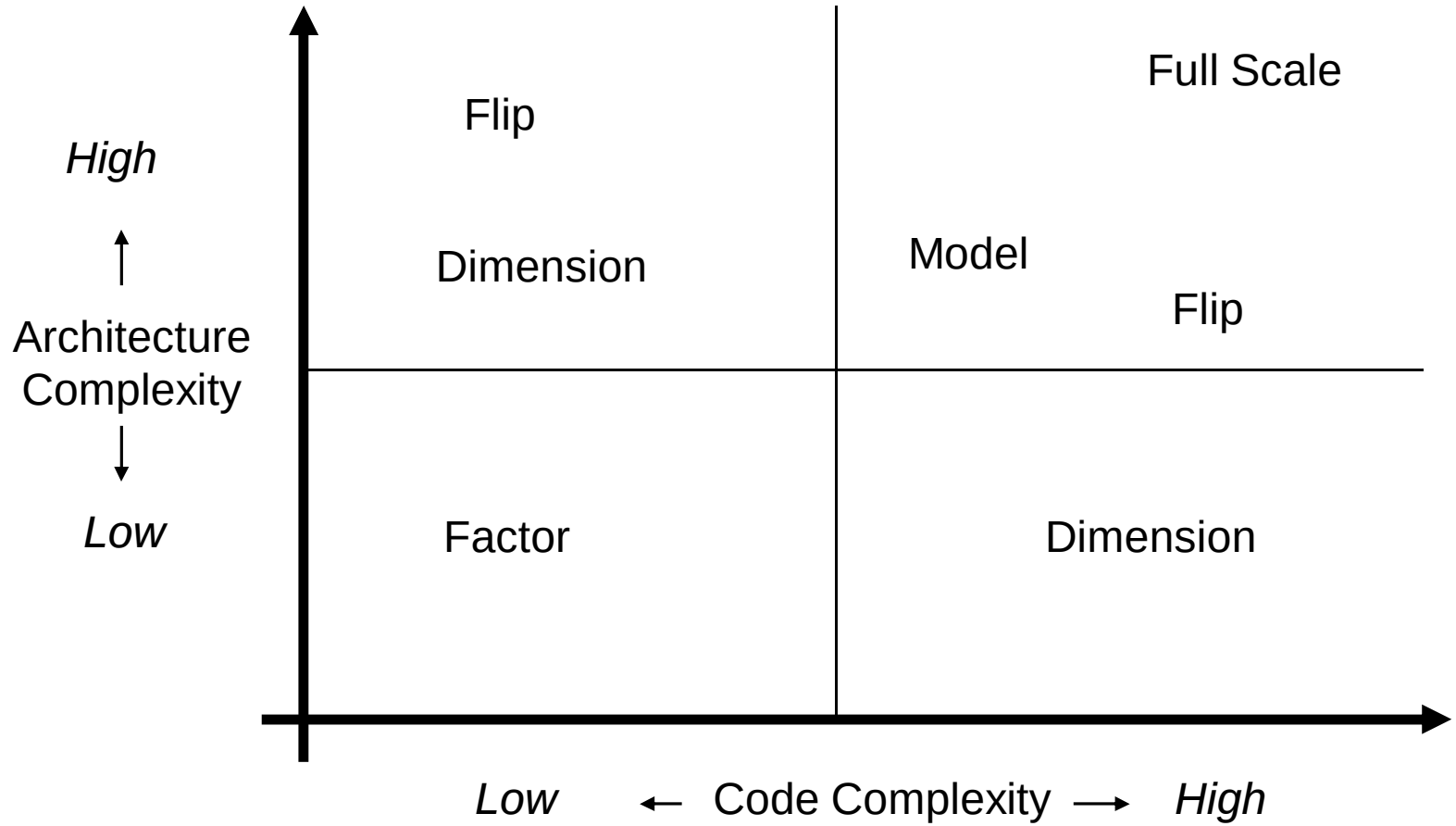
Flipping

- Flipping between large and small test environment
- Small Scale test environment
  - Permanent test environment
  - Used for routine performance tests (code benchmarking)
- Large Scale Test
  - Often temporary environment
  - Used for test application scalability
    - Useful for testing new technology
    - Significant software architecture changes

Considerations

- What is the aim of the performance test?
  - Performance regression test or radical change?
  - Are response time accuracy critical?
- What are the risk and consequence of failure?
- Is the architecture understood?
- Is the software understood?

Decision Matrix



Conclusion

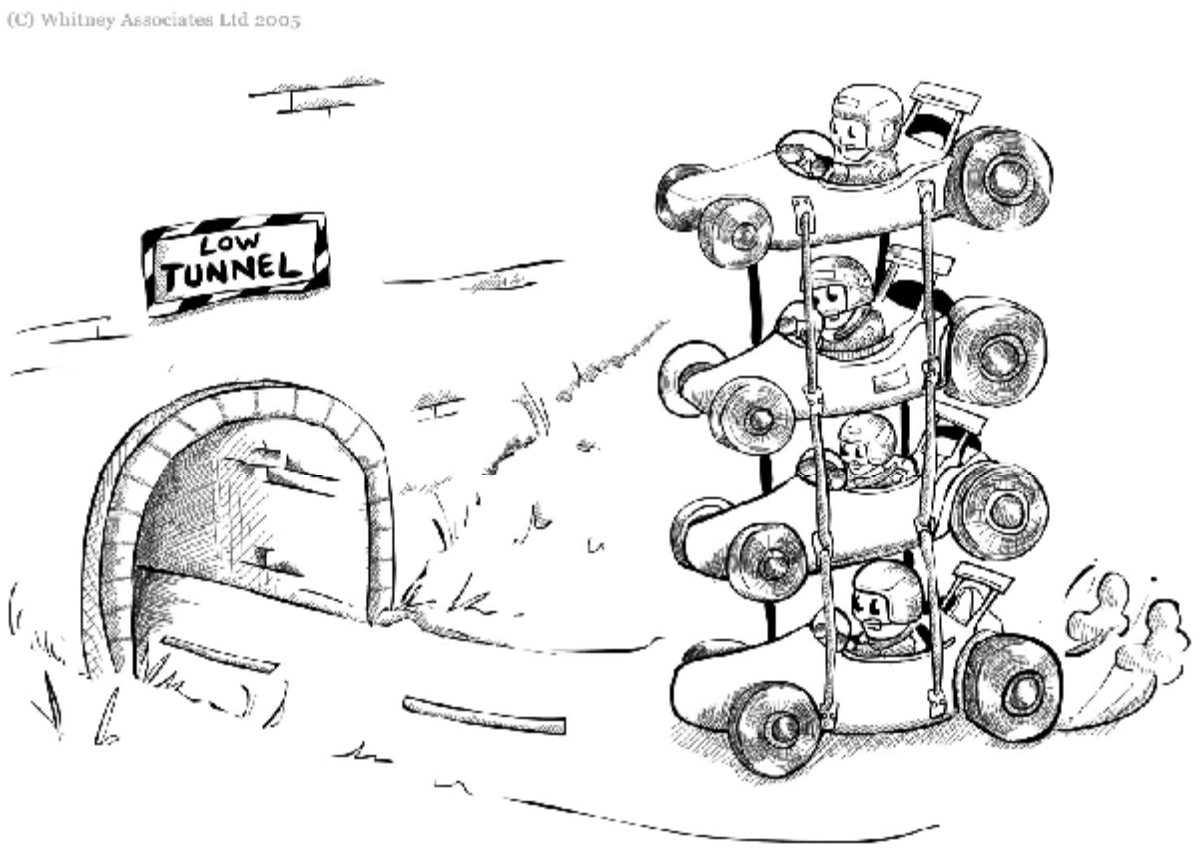
“There are known knowns. These are things we know that we know. There are known unknowns. That is to say, there are things that we know we don't know. But there are also unknown unknowns. There are things we don't know we don't know.”

Donald Rumsfeld

Conclusion

- None of these techniques guarantee success
  - Be realistic you may still have soft bottlenecks
- Aim for the largest test environment possible
- Use techniques to ensure you do the best job possible
  - Paradox lead to pragmatism
  - Don’t be afraid to walk away
- Remember your caveats will be lost in translation

Finally



Bob's optimisation strategy had a fatal flaw