# Simple Web Server: Bottlenecks

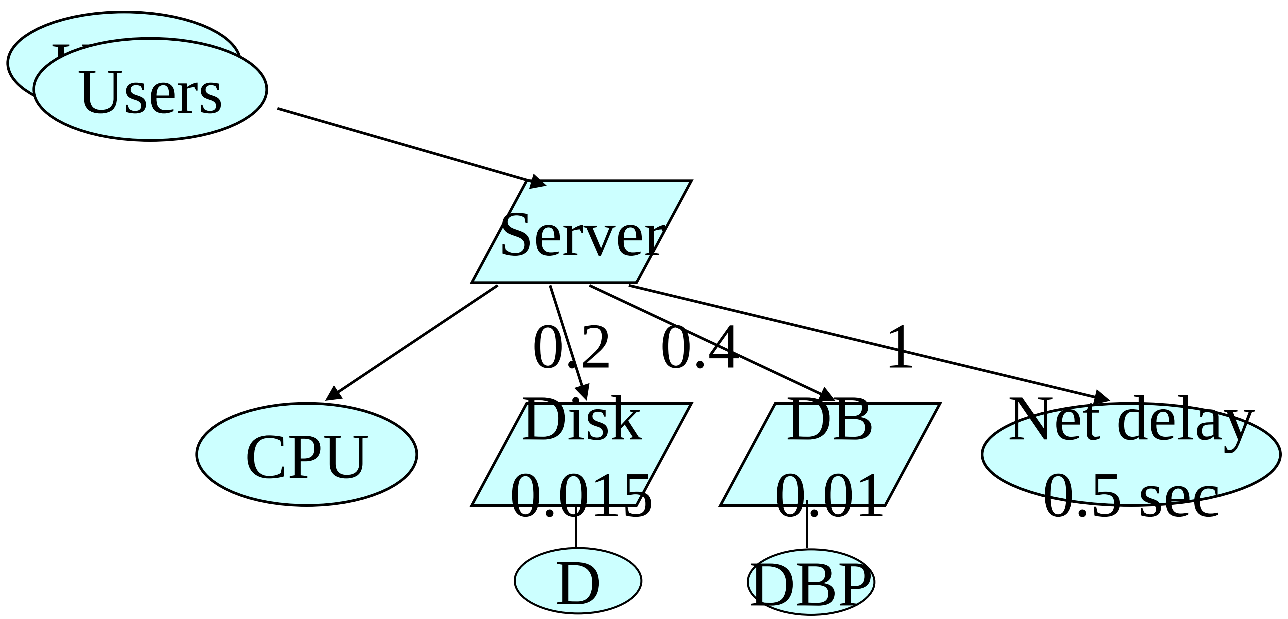
Murray Woodside
Department of Systems and Computer Engineering
Carleton University, Ottawa, Canada
cmw@sce.carleton.ca
www.sce.carleton.ca/faculty/woodside.html

Carleton UNIVERSITY

# LQN for a Web server

- Server has entry demand 0.005 sec
  - can be multithreaded
- Net delay represents total net delays that block a server thread in a response

**N Users with a thinking time of 5 sec.**

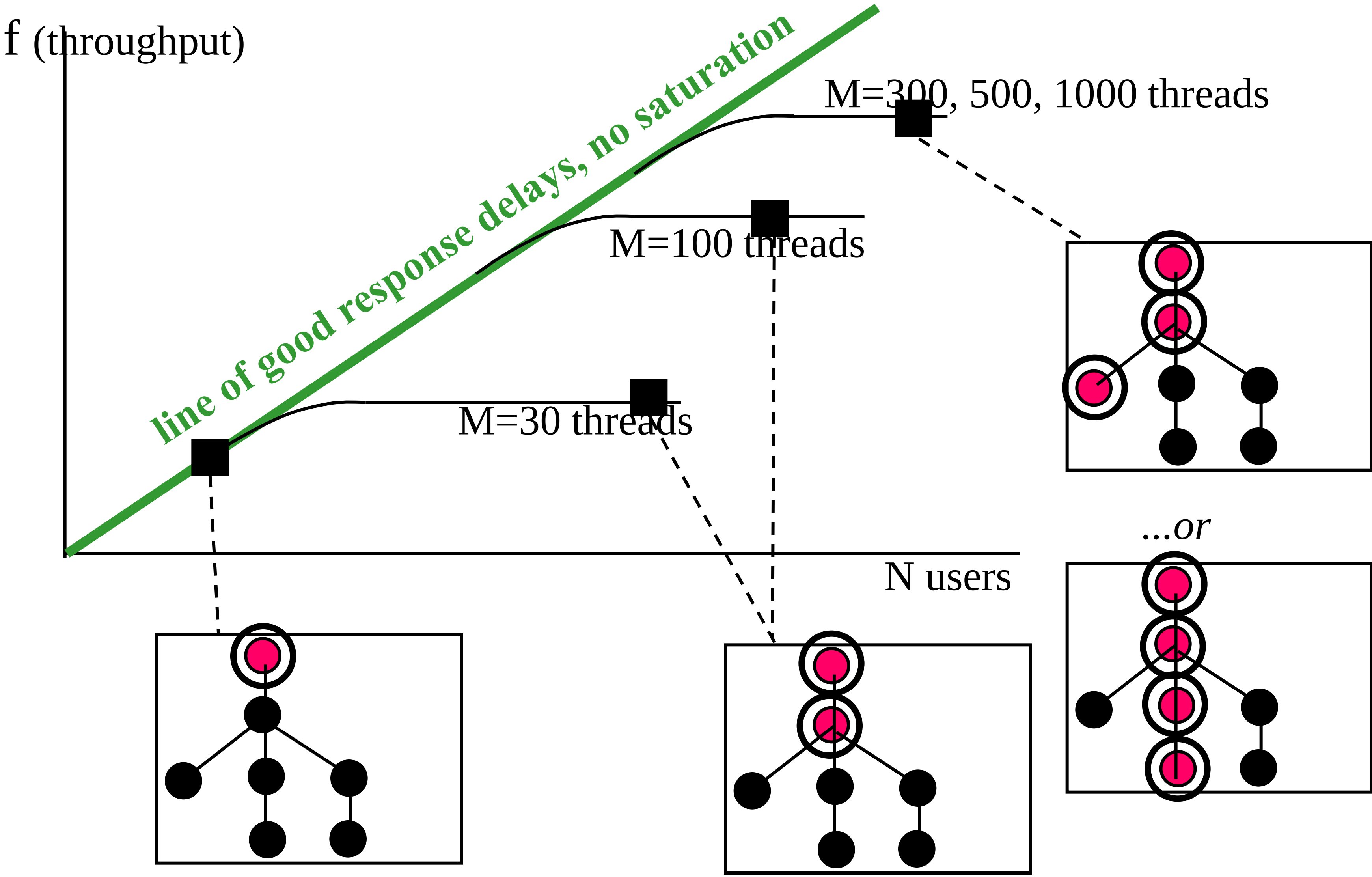


Carleton UNIVERSITY

# Bottleneck in the web server...

- is a saturation point that causes it to run slowly
  - a saturated resource that limits the throughput
- in a flat resource architecture one resource is saturated, the rest are underutilized at that throughput
- in a layered architecture several resources may be saturated
  - resources above the bottleneck have increased holding times due to pushback

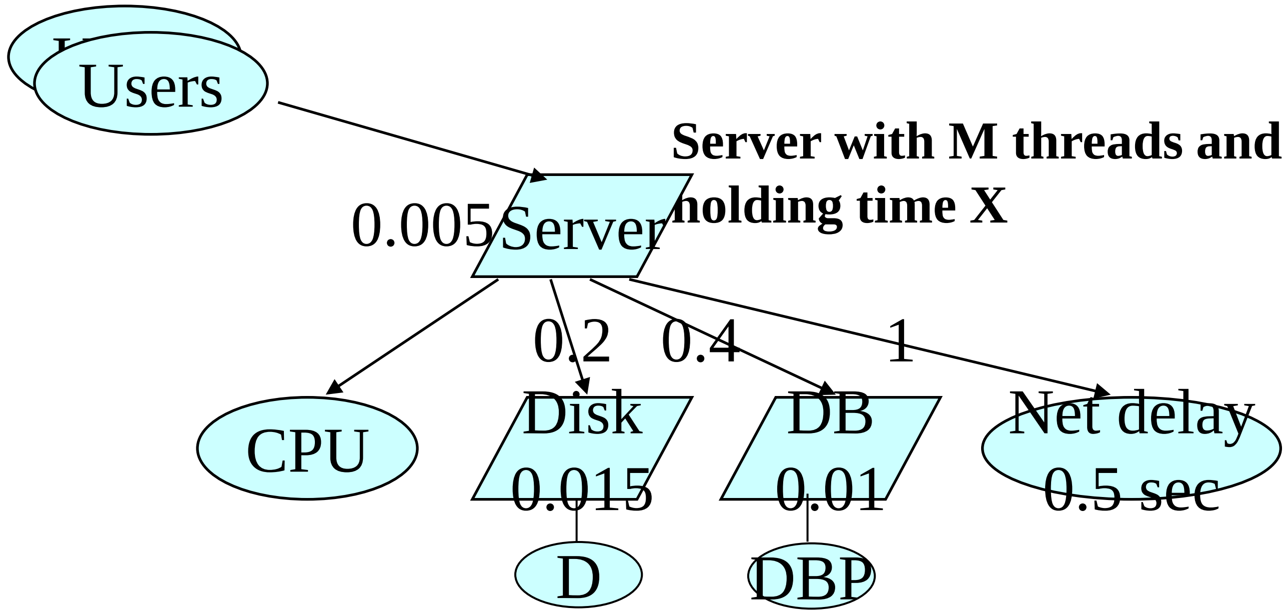
# Throughput saturation in the web server



# Bottleneck in a web server: use of threads

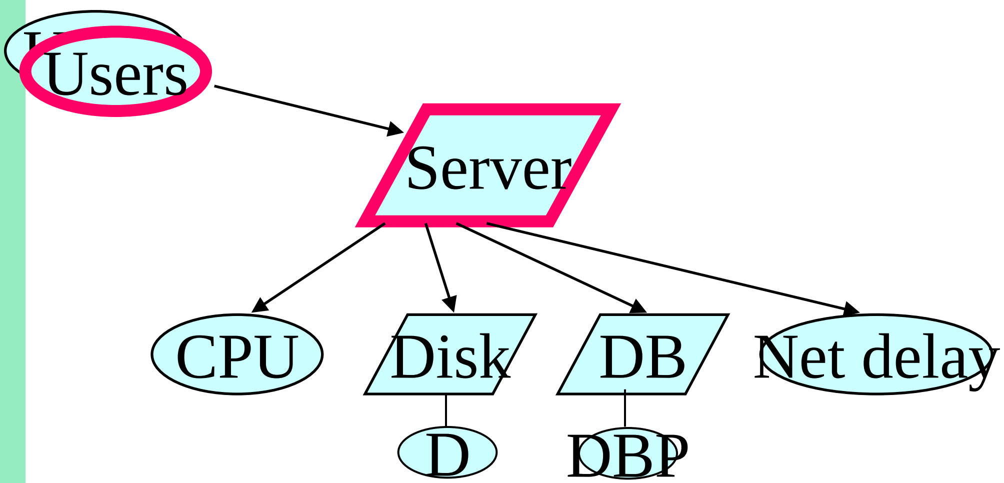
**N Users with a thinking time of 5 sec.**

**Server with M threads and holding time X**




| | | | | |
|---|---|---|---|---|
| N users | 500 | 500 | 500 | 500 |
| M threads | 10 | 30 | 100 | inf |
| X server | .512 | .52 | .52 | .52 |
| f thruput | 19.5 | 58.2 | 90.6 | 90.6 |
| W user wait | 20.6 | 3.6 | 0.51 | 0.5 |
| U server | 10 | 30 | 47 | 47 |
| U net | 9.7 | 29.1 | 45.3 | 45.3 |
| U CPU | .097 | .29 | .45 | .45 |

# Pattern around the bottleneck

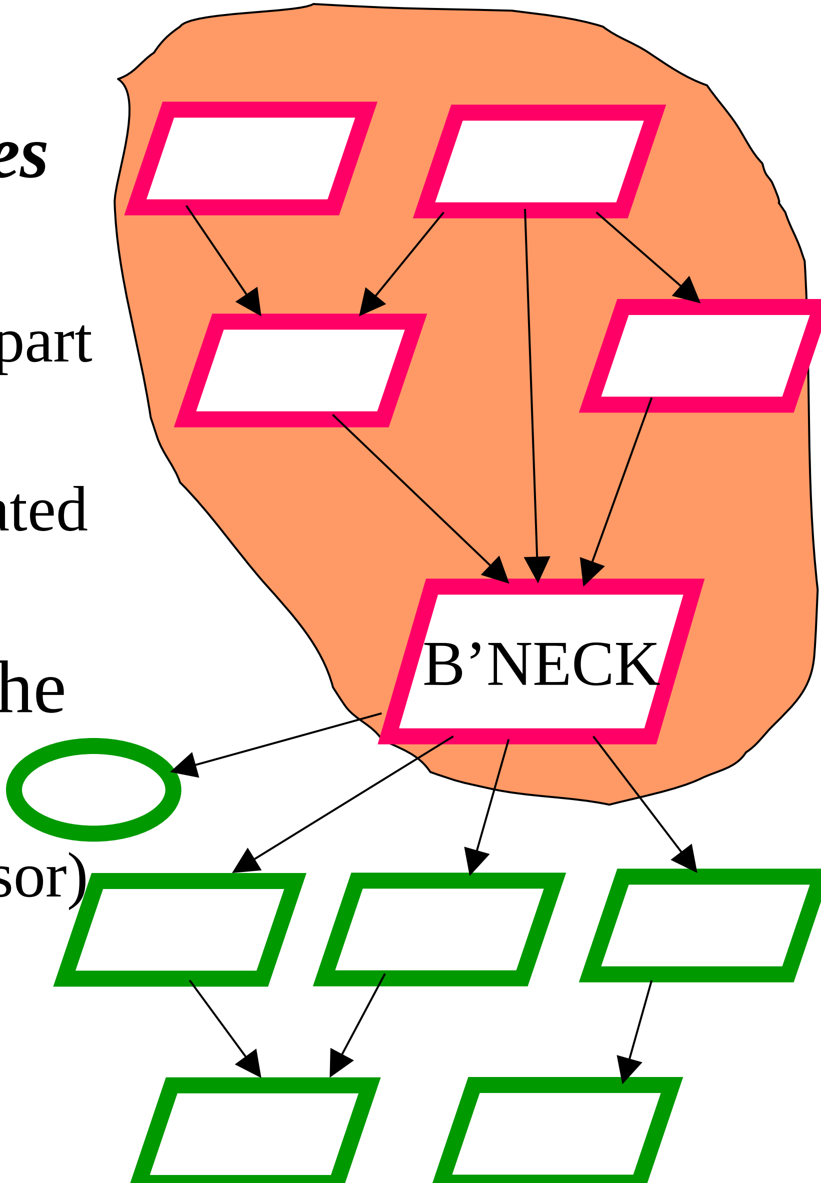



- users are always "busy" (waiting or "thinking")
  - saturated in a sense
- server is saturated
- devices and lower servers are unsaturated

....with sufficient server threads, the server is unsaturated and the devices too... this is the ideal

# Insight: Pattern for a "Software Bottleneck"

- a saturated server
- but.... a saturated server ***pushes back*** on its clients
  - the long waiting time becomes part of the client service time!!
  - result is often a cluster of saturated tasks above the bottleneck
- thus: the "real" bottleneck is the ***"lowest"*** saturated task
  - its servers (including its processor) are not saturated
  - some or all of its clients are saturated



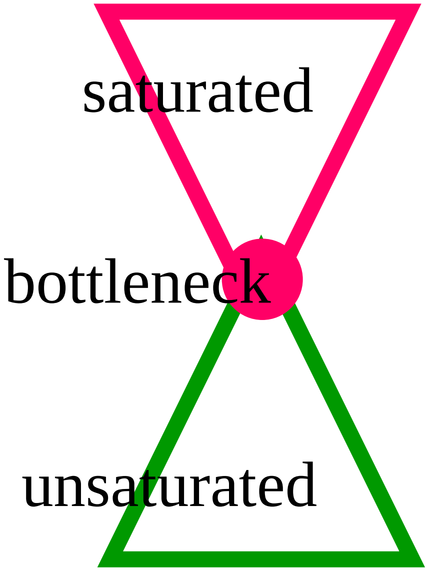
# Hourglass pattern shows saturation behaviour

***above:*** tasks above the bottleneck are saturated because of pushback delays

- there must be sufficient numbers to build a queue

***below:*** tasks below are unsaturated because the bottleneck throttles the load

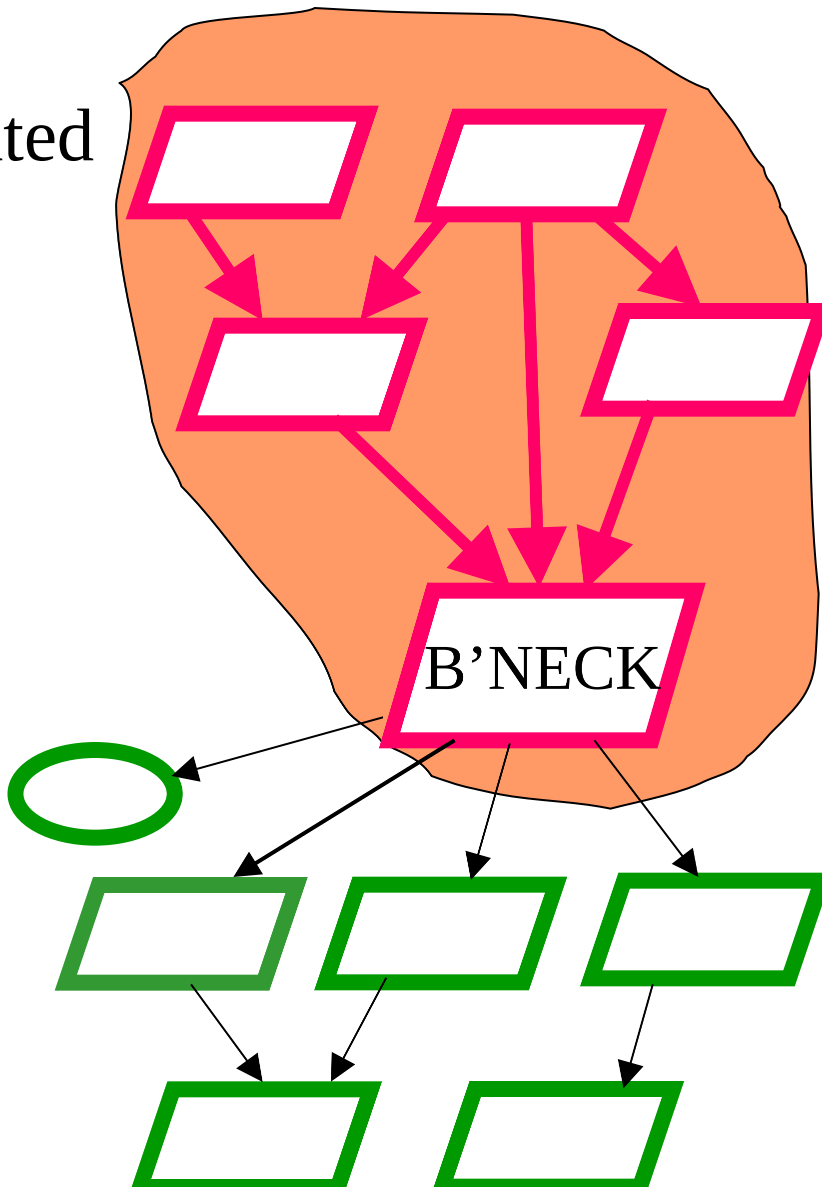
- typically their load is spread across several resources



Carleton UNIVERSITY

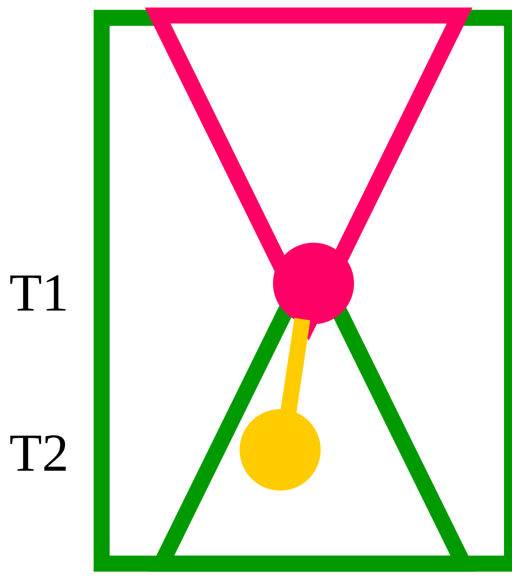
# Recognizing the "real" bottleneck

- a saturated task with unsaturated servers and host
- look at resource utilizations
- look for a step downwards in utilization, in descending the heirarchy:
  - sat
  - sat
  - ***sat: bottleneck***
  - unsat
  - unsat



# “Next bottleneck”

- if the capacity of bottleneck T1 can be increased
  - then lower task T2 with the max utilization $U_{T2}$ is the *next bottleneck*
  - strength measure is $U_{T1} / U_{T2}$
  - processor or server “support”
- the potential throughput increase
  - will raise $U_{T2}$ to unity and saturate T2
  - is bounded in ratio by the strength measure
- in practice the utilization of T2 may increase more rapidly with throughput, and T2 saturate at a lower throughput
- IEEE TSE paper 1995



# Mitigation of a bottleneck (Peter Tregunno)

(1) provide additional resources at the bottleneck

- for a software server, provide ***multiple threads***
  - some "asynchronous server" designs provide unlimited threads
- ***replicated*** servers can split the load and distribute it, but give them each a processor
- for a processor, a ***multiprocessor*** (or faster CPU)

(2) reduce its service time to make it ***faster***:

  - reduced host demand (tighter code)
  - reduced requests to its servers
  - parallelism, optimism
  - less blocking time (phase 1 time) at its servers

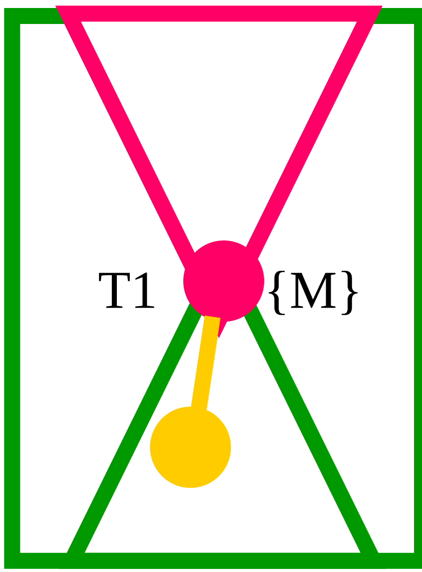
(3) divert load away from it

Carleton UNIVERSITY

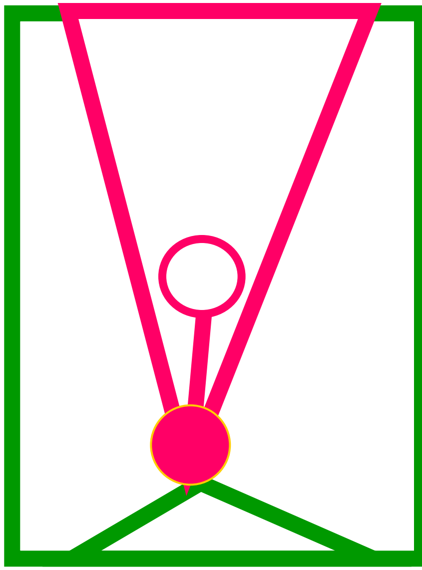
# Use additional resources...

- a resource may be given additional (M servers)
  - multiprocessor
  - multithreaded task
- a (rough) rule of thumb for M, based on potential needs for concurrency at a task T1:

M = min of { (1 + sum of resources of servers of T1),
(sum of clients of T1) }




- increase the capacity of the bottleneck resource
  - holding time drops, throughput increases
  - *lower* resources see more load and also more waiting
    - their utilization increases (bottleneck can move down to the “next bottleneck”)
- however, a *higher* resource may also remain saturated due to higher throughput
    - bottleneck can move up, to a destination difficult to predict.



Carleton University

# Comments on *additional resources*... e.g. increasing threading levels

- Useful with a *strong* software bottleneck
- Potential throughput at bottleneck <= $f_b * B_b$
  - $f$ = throughput
  - $B$ = ratio of utilizations (relative to saturation) at the bottleneck, to its highest utilized server.
  - $B > 1$ at a bottleneck
- Optimal threading level is usually found through experiment
  - first rule of thumb is to use the sum of threads or multiplicities of its servers
  - second rule, increase multiplicity by factor B (to provide the additional throughput)
- Cost is usually minimal (low overhead), unless software design is explicitly singlethreaded

Carleton UNIVERSITY

# Comments on *replication of task & processor*

- meaning, add more hardware…
  - Useful with a weak processor supported software bottleneck (threading helps strong bottlenecks)
  - Reduction in utilization of the bottleneck task proportional to $p/n$ (where $p$ is the percentage of total service time that a task spends blocked due to processor contention, and $n$ is the number of processors added)
  - Only effective when processor contention is high

- other ways to increase resource accessibility: more read access, less exclusive access

# Comments on *reducing processing demands*

- ... write faster code…
- Only applicable for processor supported software bottlenecks
- The utilization gain is only proportional to the reduction in total processing demands
- For a strong server supported software bottleneck, the underlying problem is blocking, not slow software at the bottleneck.

# Other ways to reduce holding time

- anticipation (prefetching)
- other optimistic operations
- parallelism in a server
- asynchronous operations

# Comments on *decreasing interactions*

- for example, batching multiple requests
  - if synchronous requests can be bundled together - server still has to be the same amount of work, but *n* times less waiting (waiting for rendezvous acceptance) required at the client
- effective when bottleneck is weak (long rendezvous delays are a product of high server utilizations, high server utilization = weak bottleneck)

Carleton UNIVERSITY

# Papers on the research


- Simeoni, Inverardi, DiMarco, Balsamo, “Model-based performance prediction in software development”, IEEE TSE May 2004 pp 295-310
- “The Layered Queueing Tutorial”, available at www.layeredqueues.org
- D. B. Petriu, M. Woodside, “A Metamodel for Generating Performance Models from UML Designs”, UML 2004, Lisbon, Oct. 2004.
- P. Maly, C.M. Woodside, "Layered Modeling of Hardware and Software, with Application to a LAN Extension Router", Proc. TOOLS 2000, pp 10-24
- J.E. Neilson, C.M. Woodside, D.C. Petriu and S. Majumdar, "Software Bottlenecking in Client-Server Systems and Rendezvous Networks", *IEEE TSE*, v. 21, pp. 776-782, Sept. 1995.
- D. C. Petriu and C. M. Woodside, "Performance Analysis with UML," in the volume "UML for Real", edited by B. Selic, L. Lavagno, and G. Martin, . Kluwer, 2003, pp. 221-240
- F. Sheikh and C.M. Woodside, "Layered Analytic Performance Modelling of a Distributed Database System", *Proc. 1997 International Conf. on Distributed Computing Systems*, May 1997, pp. 482-490.

Carleton UNIVERSITY

# Papers (2)


- M. Woodside, D.B. Petriu, K. H. Siddiqui, "Performance-related Completions for Software Specifications", Proc ICSE 2002.
- C.M. Woodside, "A Three-View Model for Performance Engineering of Concurrent Software", *IEEE TSE*, v. 21, No. 9, pp. 754-767, Sept. 1995.
- Pengfei Wu, Murray Woodside, and Chung-Horng Lung, "Compositional Layered Performance Modeling of Peer-to-Peer Routing Software," in Proc 23rd IPCCC, Phoenix, Ariz., April 2004
- Tao Zheng, Murray Woodside, "Heuristic Optimization of Scheduling and Allocation for Distributed Systems with Soft Deadlines", Proc. TOOLS 2003, Urbana, Sept 2003, pp 169-181, LNCS 2794.
- Jing Xu, Murray Woodside, Dorina Petriu "Performance Analysis of a Software Design using the UML Profile for Schedulability, Performance and Time", Proc. TOOLS 2003, Urbana, Sept 2003, pp 291 - 310, LNCS 2794.
- other papers on layered queueing by Perros, Kahkipuro, Menasce, and many others (see www.layeredqueues.org).