YORK UNIVERSITÉ UNIVERSITY

ASRL
Adaptive Systems Research Lab

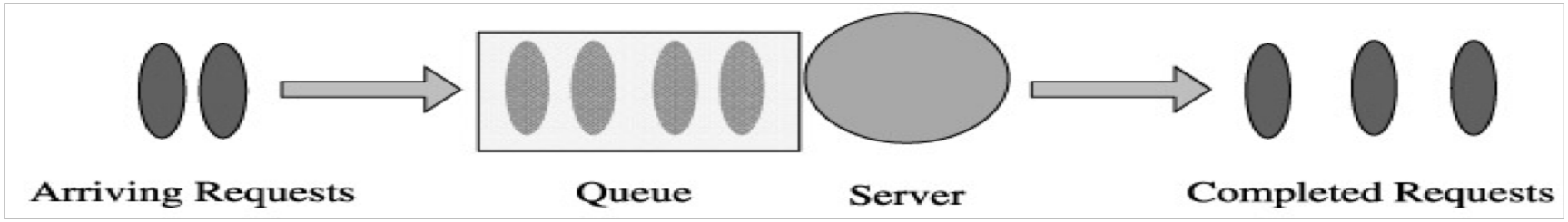
# Model Based Adaptive Systems

Marin Litoiu
Department of Electrical Engineering and Computer Science
York University
mlitoiu@yorku.ca
http://www.ceraslabs.com

# Models

- **Queuing Network Models**
- **Control Theoretic Models**
- **Machine Learning Models**
  - Regression models
  - Neural networks models

# Notations



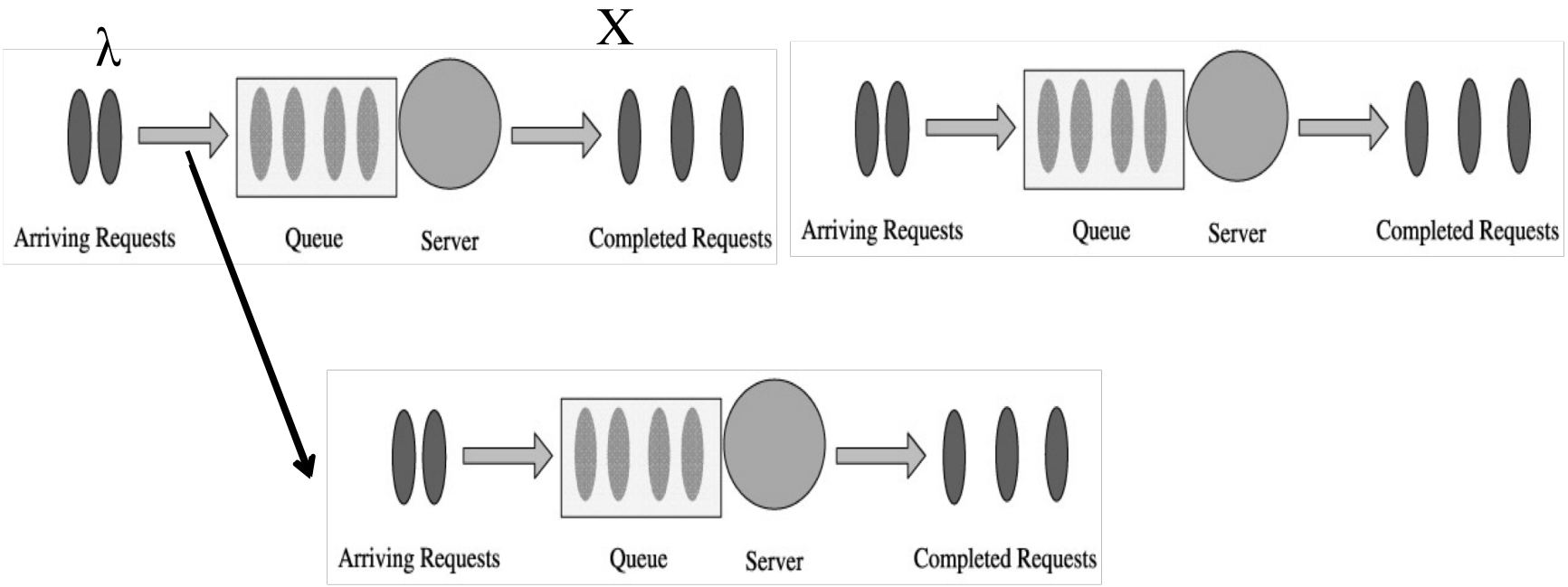

| Symbol | Semantics |
|---|---|
| *S* | Service time |
| *V* | Number of visits to the server |
| *D=V*S* | Service demand |
| *R* | Response time |
| *R'* | Residence time |
| *X* | Throughput |
| *λ* | Arrival rate |
| *U* | Utilization |
| *W* | Wait time |
| *N* | Total queue length (waiting and/or being serviced) |

Examples of Queues
-CPU
-DISK
-Network card
-a critical section
-a semaphore
-a threading pool

# Little's Law, Utilization Law

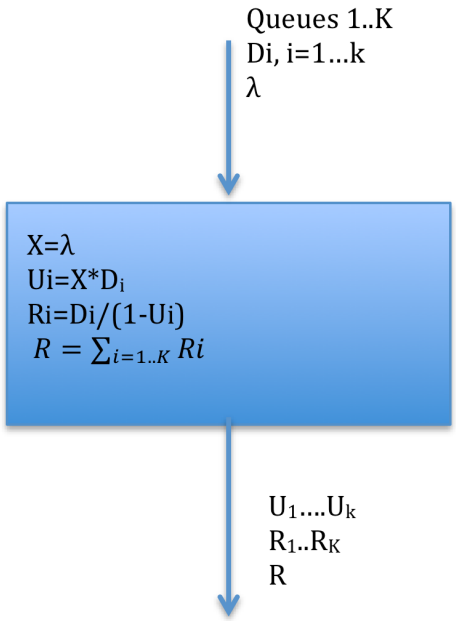
- **$N_i=X_i*R_i$; created by John Little (1961)**
  - where i is the queue; $R_i$ is the response time; $X_i$ is the throughput of queue;
- **$U_i=X_i*D_i$; where i is the queue; $D_i$ is the Demand(service time to complete a request at queue i); $X_i$ is the throughput of queue i;**

# Queues: there is a network of queues



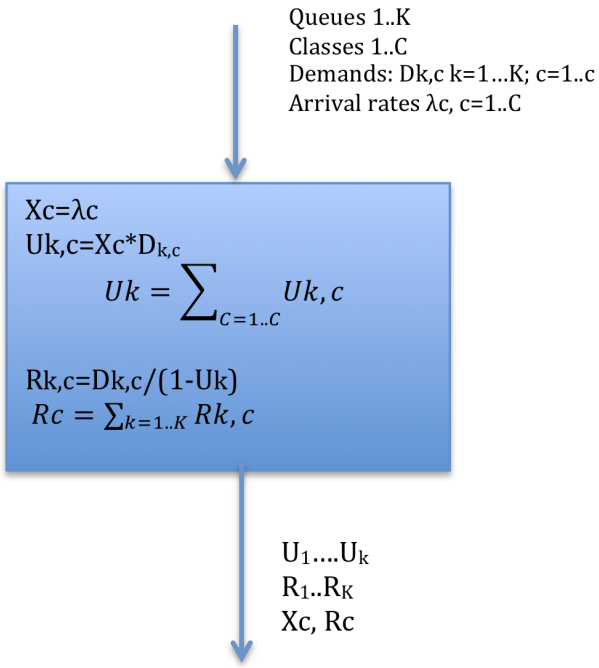
# Open Models

- Very simple and fast formulas
- Inputs: Queues and demands in queues; arrival rate ( req/sec)
- Outputs: overall response time; throughput and per server utilization
- Note that Xi=X ( forced flow law): the customers requests go through all queues and are not lost.
- Implement this in Java or Python



# Open Models for Multiple Classes

- **Very simple and fast formulas**
- **Inputs: Queues and demands in queues; arrival rate ( req/sec) for each class of request**
- **Outputs: overall response time per each class; throughput and per server utilization per each class of request**
- **You can implement this in Excel**
  - Keep in mind U<1, therefore, (1-Uk) should not be 0 or negative
- **Implement this in your prefered programming language**



# Data for a 3 tier app, 2 classes, 2 queuing centres (application server and database server)

| lambda_1 | lambda_2 | D11 | D12 | D21 | D22 | U11 | U12 | U21 | U22 | U1 | U2 | R11 | R12 | R21 | R22 | R1 | R2 |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0.15 | 0.1 | 0.1 | 0.15 | 0.2 | 0.25 | 0.015 | 0.015 | 0.03 | 0.025 | 0.03 | 0.055 | 0.103092784 | 0.154639175 | 0.211640212 | 0.264550265 | 0.314732995 | 0.41918944 |
| 0.3 | 0.2 | 0.1 | 0.15 | 0.2 | 0.25 | 0.03 | 0.03 | 0.06 | 0.05 | 0.06 | 0.11 | 0.106382979 | 0.159574468 | 0.224719101 | 0.280898876 | 0.33110208 | 0.440473344 |
| 0.45 | 0.3 | 0.1 | 0.15 | 0.2 | 0.25 | 0.045 | 0.045 | 0.09 | 0.075 | 0.09 | 0.165 | 0.10989011 | 0.164835165 | 0.239520958 | 0.299401198 | 0.349411068 | 0.464236362 |
| 0.6 | 0.4 | 0.1 | 0.15 | 0.2 | 0.25 | 0.06 | 0.06 | 0.12 | 0.1 | 0.12 | 0.22 | 0.113636364 | 0.170454545 | 0.256410256 | 0.320512821 | 0.37004662 | 0.490967366 |
| 0.75 | 0.5 | 0.1 | 0.15 | 0.2 | 0.25 | 0.075 | 0.075 | 0.15 | 0.125 | 0.15 | 0.275 | 0.117647059 | 0.176470588 | 0.275862069 | 0.344827586 | 0.393509128 | 0.521298174 |
| 0.9 | 0.6 | 0.1 | 0.15 | 0.2 | 0.25 | 0.09 | 0.09 | 0.18 | 0.15 | 0.18 | 0.33 | 0.12195122 | 0.182926829 | 0.298507463 | 0.373134328 | 0.420458682 | 0.556061158 |
| 1.05 | 0.7 | 0.1 | 0.15 | 0.2 | 0.25 | 0.105 | 0.105 | 0.21 | 0.175 | 0.21 | 0.385 | 0.126582278 | 0.189873418 | 0.325203252 | 0.406504065 | 0.451785531 | 0.596377483 |
| 1.2 | 0.8 | 0.1 | 0.15 | 0.2 | 0.25 | 0.12 | 0.12 | 0.24 | 0.2 | 0.24 | 0.44 | 0.131578947 | 0.197368421 | 0.357142857 | 0.446428571 | 0.488721805 | 0.643796992 |
| 1.35 | 0.9 | 0.1 | 0.15 | 0.2 | 0.25 | 0.135 | 0.135 | 0.27 | 0.225 | 0.27 | 0.495 | 0.136986301 | 0.205479452 | 0.396039604 | 0.495049505 | 0.533025905 | 0.700528957 |
| 1.5 | 1 | 0.1 | 0.15 | 0.2 | 0.25 | 0.15 | 0.15 | 0.3 | 0.25 | 0.3 | 0.55 | 0.142857143 | 0.214285714 | 0.444444444 | 0.555555556 | 0.587301587 | 0.76984127 |
| 1.65 | 1.1 | 0.1 | 0.15 | 0.2 | 0.25 | 0.165 | 0.165 | 0.33 | 0.275 | 0.33 | 0.605 | 0.149253731 | 0.223880597 | 0.506329114 | 0.632911392 | 0.655582845 | 0.856791989 |
| 1.8 | 1.2 | 0.1 | 0.15 | 0.2 | 0.25 | 0.18 | 0.18 | 0.36 | 0.3 | 0.36 | 0.66 | 0.15625 | 0.234375 | 0.588235294 | 0.735294118 | 0.744485294 | 0.969669118 |
| 1.95 | 1.3 | 0.1 | 0.15 | 0.2 | 0.25 | 0.195 | 0.195 | 0.39 | 0.325 | 0.39 | 0.715 | 0.163934426 | 0.245901639 | 0.701754386 | 0.877192982 | 0.865688812 | 1.123094622 |
| 2.1 | 1.4 | 0.1 | 0.15 | 0.2 | 0.25 | 0.21 | 0.21 | 0.42 | 0.35 | 0.42 | 0.77 | 0.172413793 | 0.25862069 | 0.869565217 | 1.086956522 | 1.04197901 | 1.345577211 |
| 2.25 | 1.5 | 0.1 | 0.15 | 0.2 | 0.25 | 0.225 | 0.225 | 0.45 | 0.375 | 0.45 | 0.825 | 0.181818182 | 0.272727273 | 1.142857143 | 1.428571429 | 1.324675325 | 1.701298701 |
| 2.4 | 1.6 | 0.1 | 0.15 | 0.2 | 0.25 | 0.24 | 0.24 | 0.48 | 0.4 | 0.48 | 0.88 | 0.192307692 | 0.288461538 | 1.666666667 | 2.083333333 | 1.858974359 | 2.371794872 |
| 2.55 | 1.7 | 0.1 | 0.15 | 0.2 | 0.25 | 0.255 | 0.255 | 0.51 | 0.425 | 0.51 | 0.935 | 0.204081633 | 0.306122449 | 3.076923077 | 3.846153846 | 3.28100471 | 4.152276295 |
| 2.7 | 1.8 | 0.1 | 0.15 | 0.2 | 0.25 | 0.27 | 0.27 | 0.54 | 0.45 | 0.54 | 0.99 | 0.217391304 | 0.326086957 | 20 | 25 | 20.2173913 | 25.32608696 |

Q: what device is the bottleneck? What is the max arrival rate you can accommodate above?

Exercise: Extend the model to 3 classes: use demands and arrival rates to be half of class 1. What do you notice with regard to Utilization and Response time?

# Your turn..

- **Get the spreadsheet of open queuing models from Moodle: it is for 2 classes, 3 tiers**
- **Understand the formulas**
- **What device is the bottleneck? What is the max arrival rate you can accommodate if you only have class 1(browse)?**
- **What is the bottleneck if you only have class 2(search)?**
- **Extend the model to 3 classes (payment): use demands and arrival rates to be half of class 1. What do you notice with regard to Utilization and Response time?**

# Closed Models ( for multiple classes of users c=1…C)

- More complex but more accurate
- Inputs: Queues and demands in queues; number of users and their think time
- Outputs: overall response time; throughput and per server utilization
- Note that Xi,c=Xc ( forced flow law): the customers requests go through all queues and are not lost.
- This is a bit more complicated to implement

Queues 1…K
Classes of users 1…C
Dk,c, k=1…K; c=1…C
Nc, c=1…C
Zc, c=1…C

The mean per class response time is:

$R_c(\mathbf{N}) = \sum_{i=1}^{K} D_{ic}[1 + Q_i(\mathbf{N} - \mathbf{1}_c)]$ where $Q_j(\mathbf{N}\text{-}\mathbf{1}_c)$ the queue length at device $i$ with one customer of class $c$ removed.

The *throughput* of class $c$, as the number of completions of class $c$ per time unit, is:
$X_c = N_c/(R_c + Z_c)$

where $Z_c$ is the think time of class $c$.

The *utilization law* is:

$U_{ic} = X_c D_{ic}$

$U_1 \ldots U_k$
$R_{1,c} \ldots R_{K,C}$
Rc
Xc

# Multiserver queues

- **How do you model**
  - Multiprocessors
  - Thread pools
  - Clusters of identical machines
    - Servers, VMS, containers

# Multiserver Queues, Erlang formula

Consider a queue with c servers ( a cluster with c nodes, a container cluster with c microservices..)

An M/M/c queue is a stochastic process where

- Arrivals occur at rate $\lambda$ req/time_unit and have Poisson process distribution. The arrivals move the process from state *i* to *i*+1, where I is the number of requests in the queue
- Service times have an exponential distribution with parameter $\mu$ in the M/M/c queue. *M= 1/D, where D is the demand*
- There are *c* servers
  - Servers serve from the front of the queue.
  - If there are less than *c* jobs, some of the servers will be idle. If there are more than *c* jobs, the jobs queue in a buffer.
- The buffer is of infinite size, so there is no limit on the number of requests it can contain.

Then, if $U=\frac{\lambda}{c\mu} < 1, then$

the probability for a request (customer) to find all servers busy is given by Erlang formula:

$$C(c,\lambda,\mu) = \frac{1}{1+(1-U)(\frac{c!}{(cU)^c})\sum_{k=0}^{c-1}\frac{(cU)^k}{k!}}$$

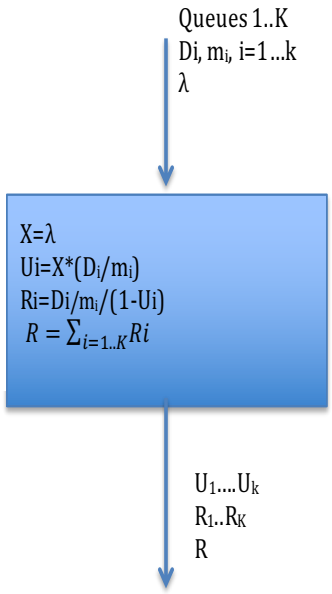
The response time of the queue (waiting + service)

$R=\frac{C(c,\lambda,\mu)}{c\mu-\lambda}+\frac{1}{\mu}$

1. At high load U-->1, how can we approximate R?

# Open Models with $m_i$ servers in clusters

- **Inputs: Queues and demands in queues for $m_i$=1; arrival rate ( req/sec);$m_i$**
- **Outputs: overall response time; throughput and per server utilization**
- **Note that Xi=X ( forced flow law): the customers requests go through all queues and are not lost.**

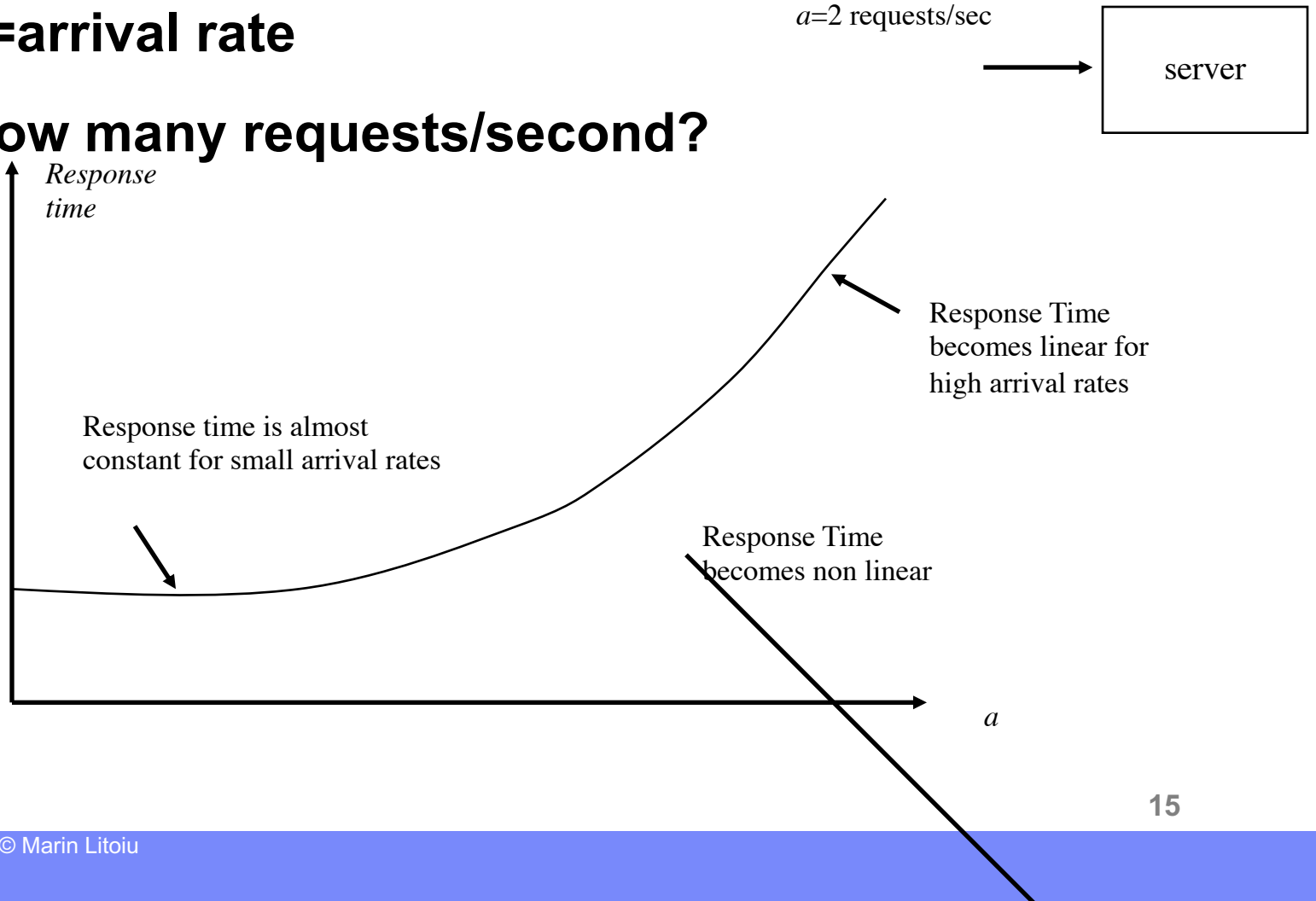


# Practical considerations

- **Typical behavior**
- **Scalability**

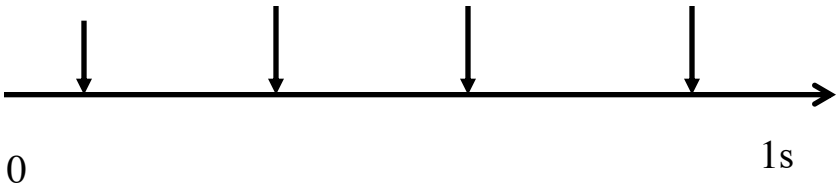
# Response time is non-linear

- ***a*=arrival rate**
- **How many requests/second?**



# Why is the response time non-linear?

- ***a* arrivals per seconds are distributed along the second**




*a*=4 arrivals/sec

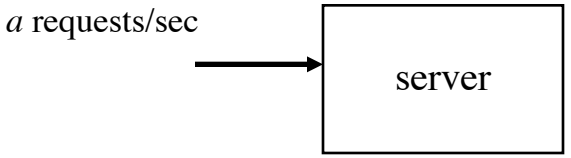
- The requests are processed by either disk, CPU, network or other devices,
- While one request is processed by CPU, another request uses the disk or network
- for a while, requests do not necessary wait for each other to finish
- as the number of requests increases, they start to wait for each other, at CPU, disk and network
- When there are two many requests, each request waits for the ones arrived before it to finish. In that case, the system is saturated.

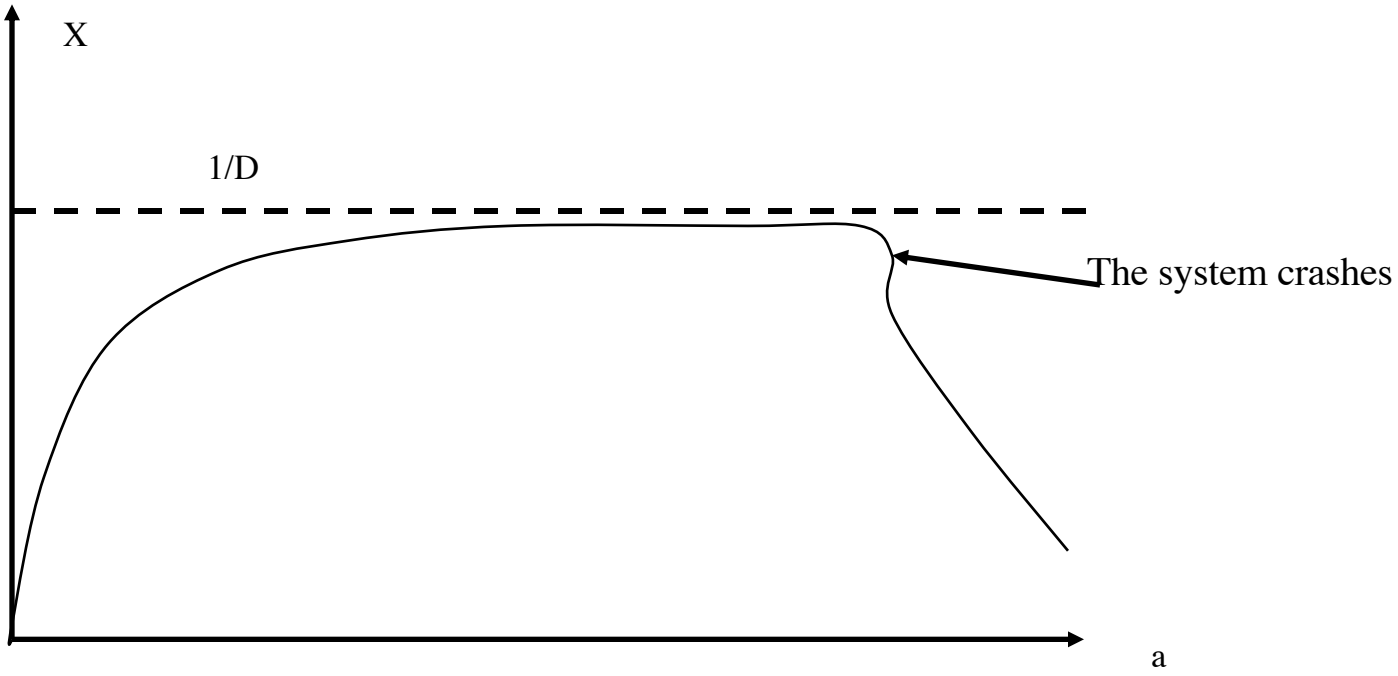
# Throughput

- **The number of transactions per time unit (per second, for example)**
- **In average, it should be equal with the arrival rate (up to a point)**
  - Otherwise the requests will queue in different places in the system,
  - The system will crash
- **Utilization law**
  - U is the utilization
  - X is the throughput
  - D is the time to process 1 request
  - U=X*D
  - Ex: D=0.03s; X=10req/s;-→U=0.3 or U=30%;
  - Q: When does the system saturate( when U=1?) ?
  - A: when X=U/D=33.3req/sec..

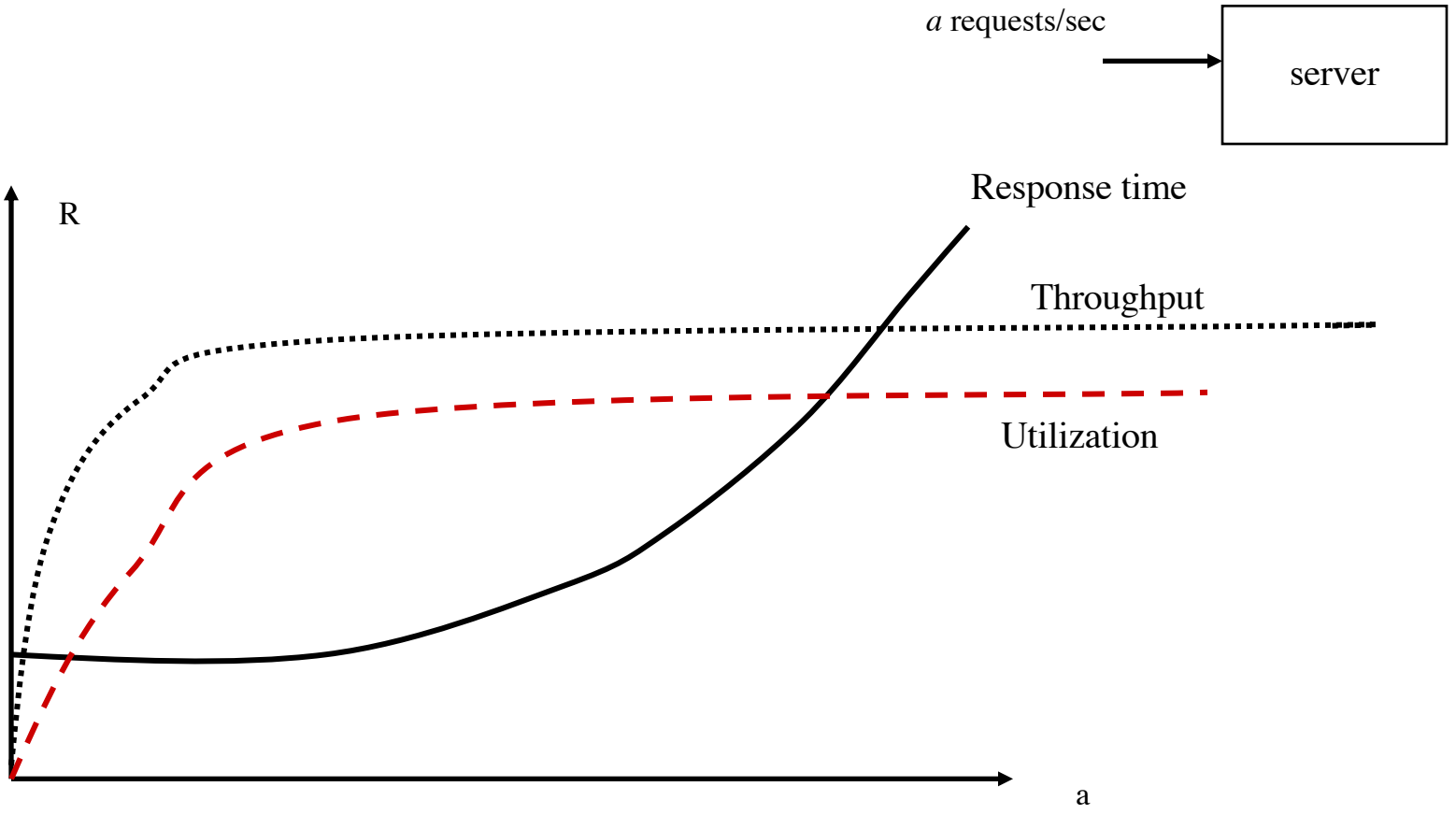
# Utilization Law

$U = X * D;\ U < 1 \rightarrow X < 1/D$

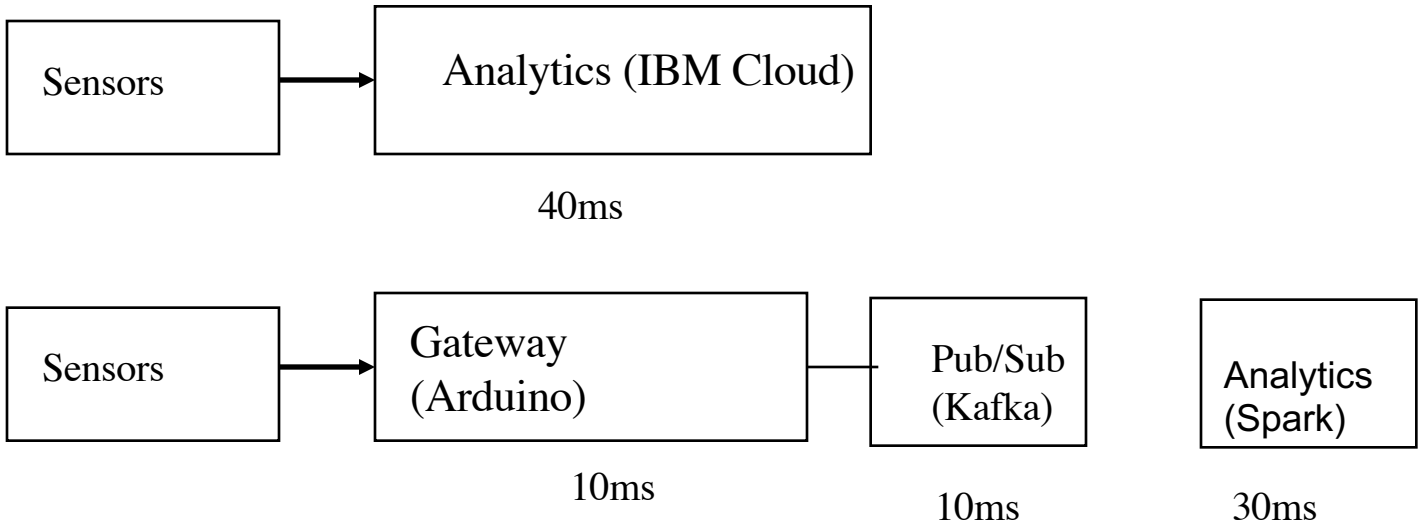


# Utilization, throughput, response time



# Choosing an IoT Architecture




2-tiers D= 40 ms

Moving some processing on other servers

-there is communication cost on both application and data servers

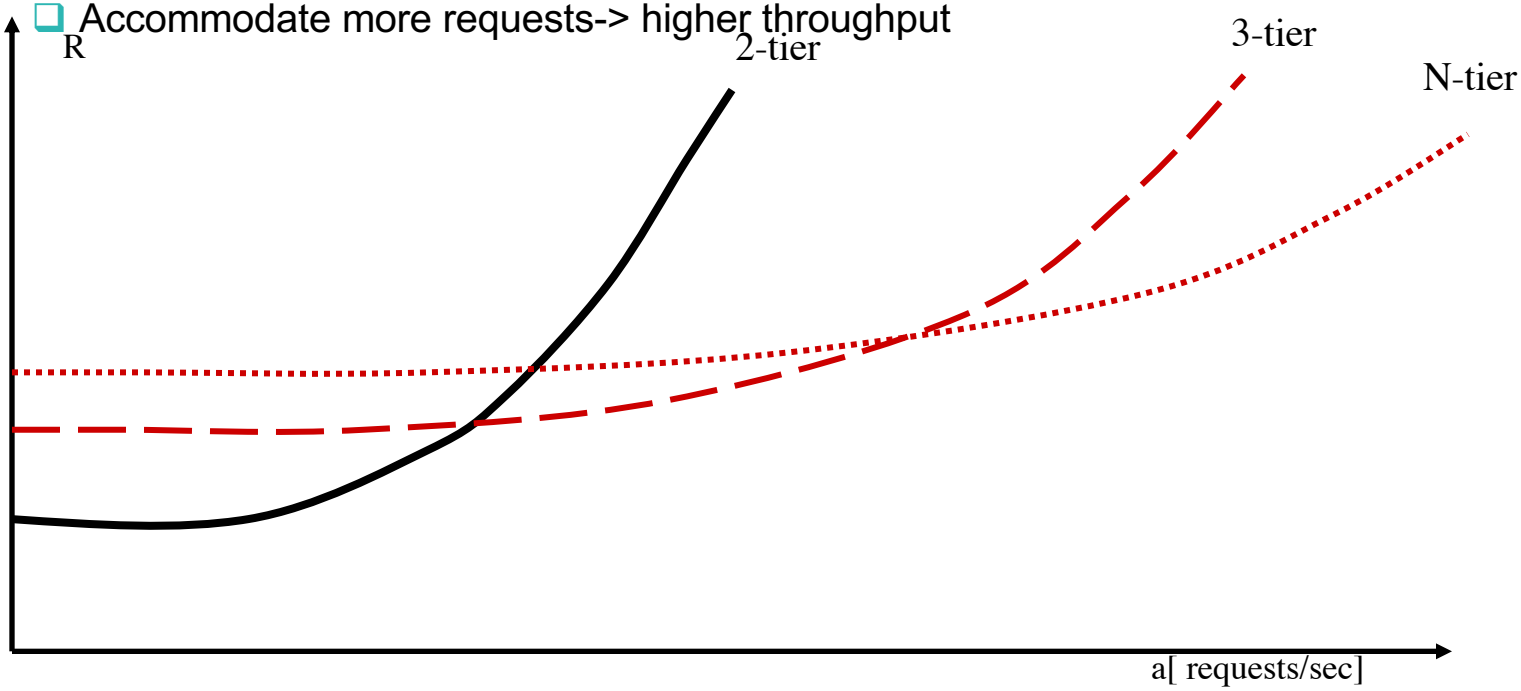
-there is network delay

3-tier D1=10ms; D2=10ms, D3=30ms

1. What is the response time for the two architectures for 1 request?
2. What is the response time when a= 4 requests/s? 100 req/sec?
3. How many servers in each tier do we need to have Ui<0.8?

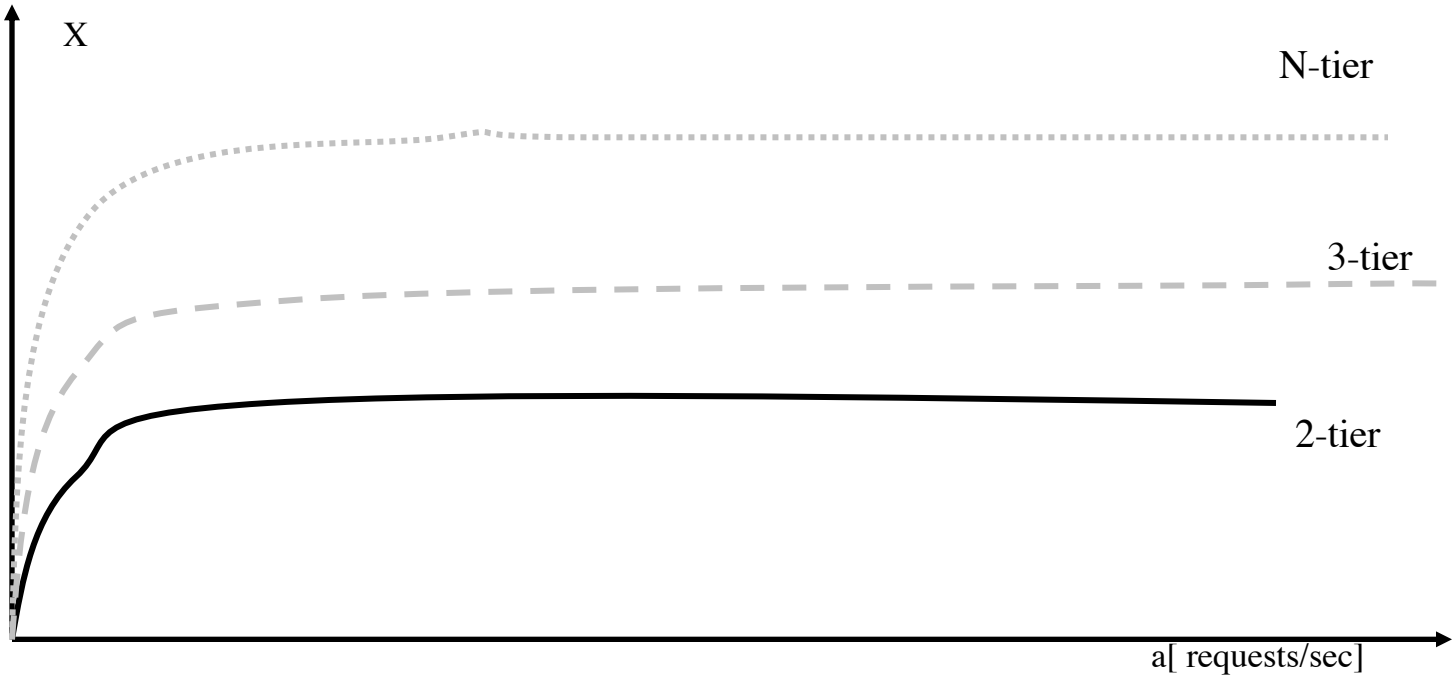
# More tiers better scalability

- **More tiers**
  - Higher response time at low arrival rates
  - Lower response time at higher arrival rates
  - Accommodate more requests-> higher throughput



# More tiers- better throughput

- **X< 1/D where D is the max among tiers**

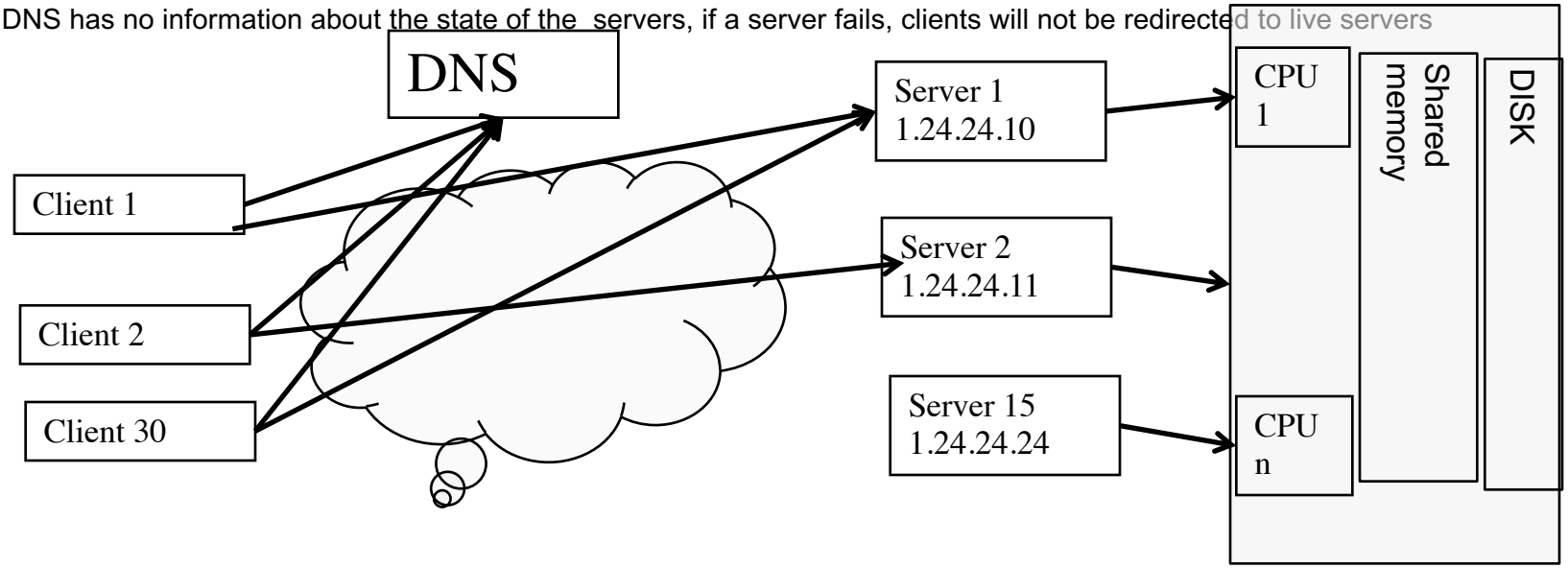


# Load balancing

- **Client servers applications have many clients ( tens, hundreds..millions)**
- **A small computer with 1CPU, can handle few requests(no more than 10, as per text book)**
  - Uses threads: one thread per request
- **We can use**
  - A power-full computer, with many CPUs ( this is expensive, x$100k) but one shared memory
    - Common for a data tier
  - Many small computers but arranged so they can process client requests in parallel
    - Common for the application and web tiers
- **Question: what prevents, in general, data tier to be deployed on many RDBMS small and identical servers?**
- **Question: for non-SQL databases, can we deploy the database on many servers?**

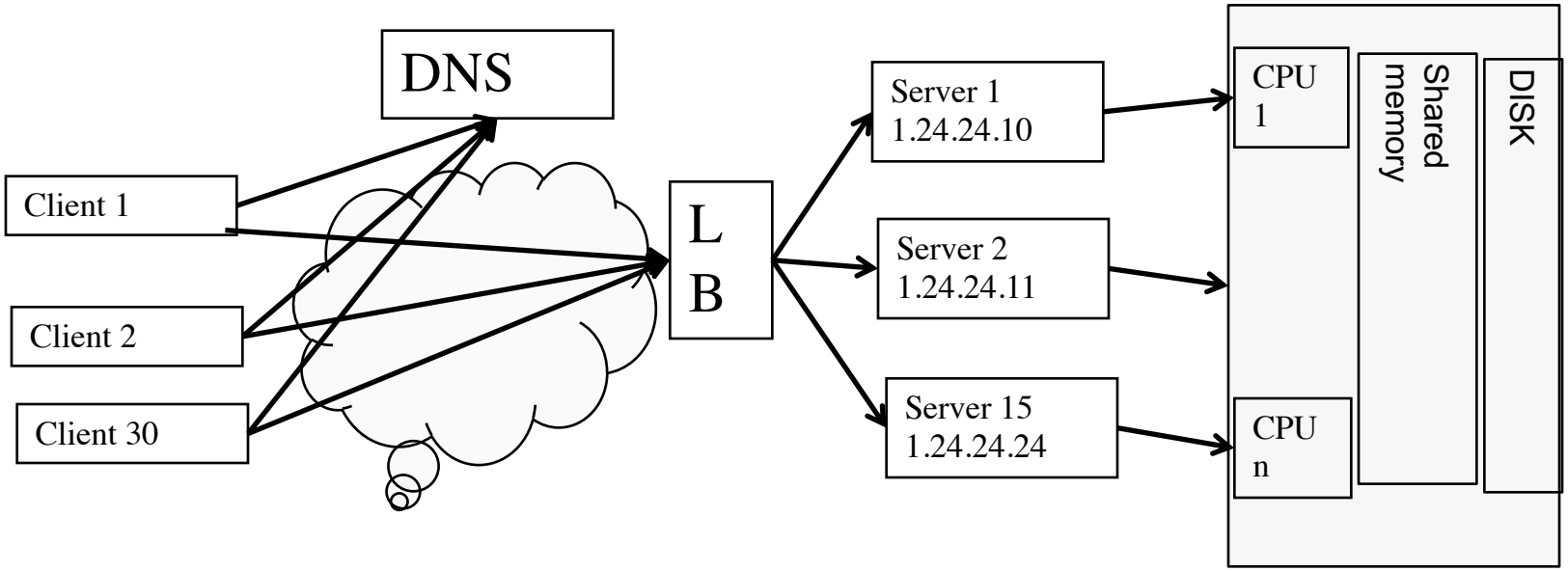
# Load balancing: DNS

- **Round Robin with Domain Name Server**
  - A server name is mapped to many IP addresses
    - Ex: cnn.com is mapped to 1.24.24.10....1.24.24.24...
  - When a client asks DNS for a server IP (cnn.com, for example ) the DNS returns IP addresses in a round robin manner
  - Clients interacts with the IP address they were given
  - Issues:
    - DNS has no information about the state of the servers, if a server fails, clients will not be redirected to live servers

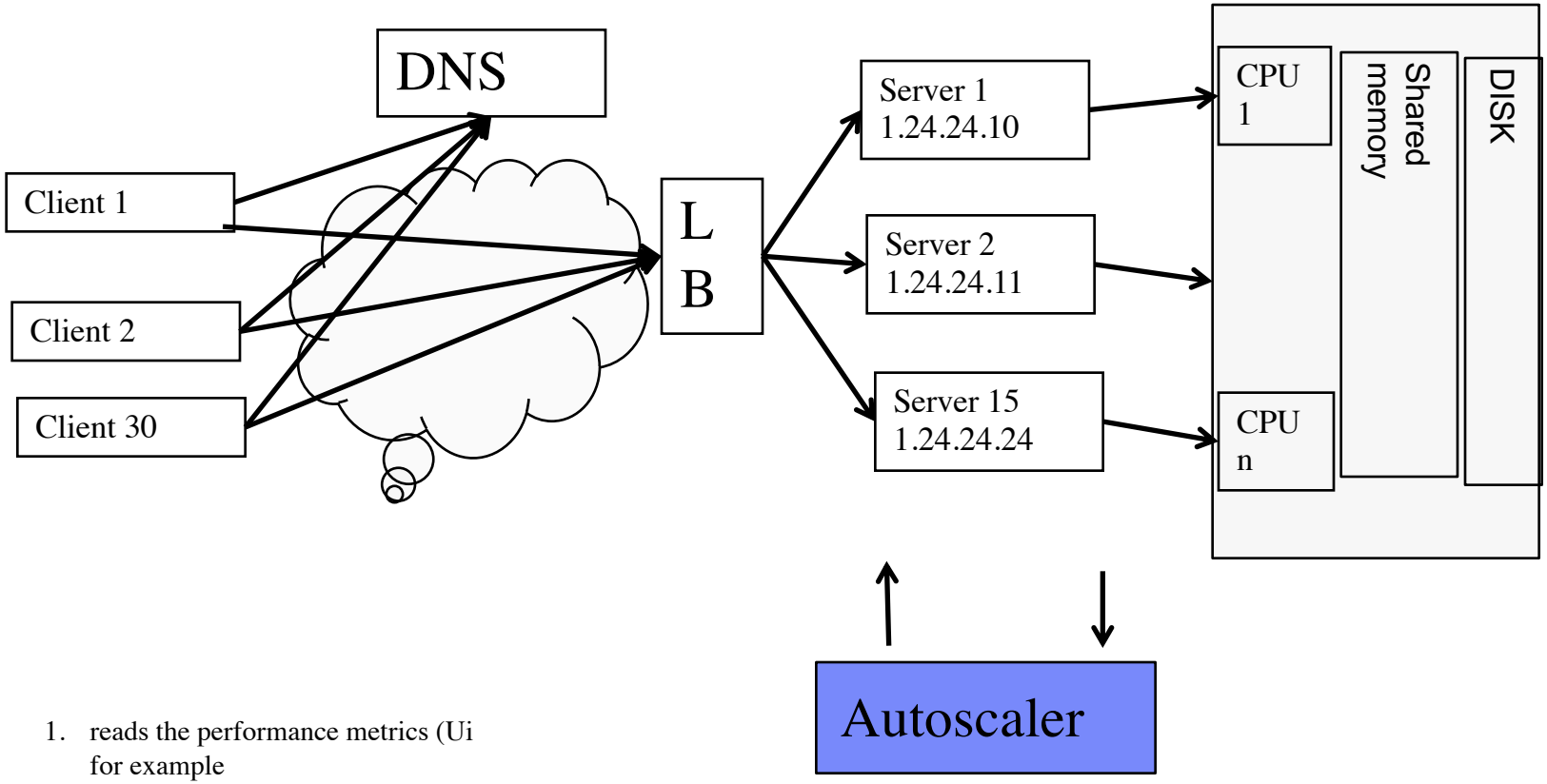


# Load balancing: Load balancer

- **A load balancer, with a well know address receives all client requests**
- **When clients ask DNS to resolve a name, they get beck the IP of the load balancer**
  - The load balancer forwards the users requests to a set of identical servers
  - LB knows how loaded  the servers are and if a server is down



# Auto Scaling in Cloud




1. reads the performance metrics (Ui for example
2. Runs "rules": if (Ui>50%) then "add 1 server"
3. Adds/remove servers dynamicaly

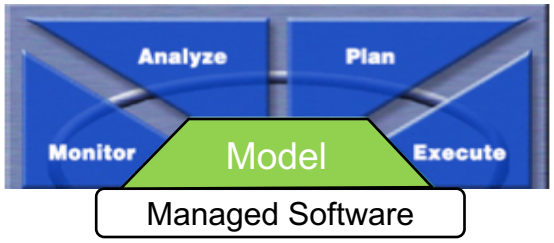
# Research Questions

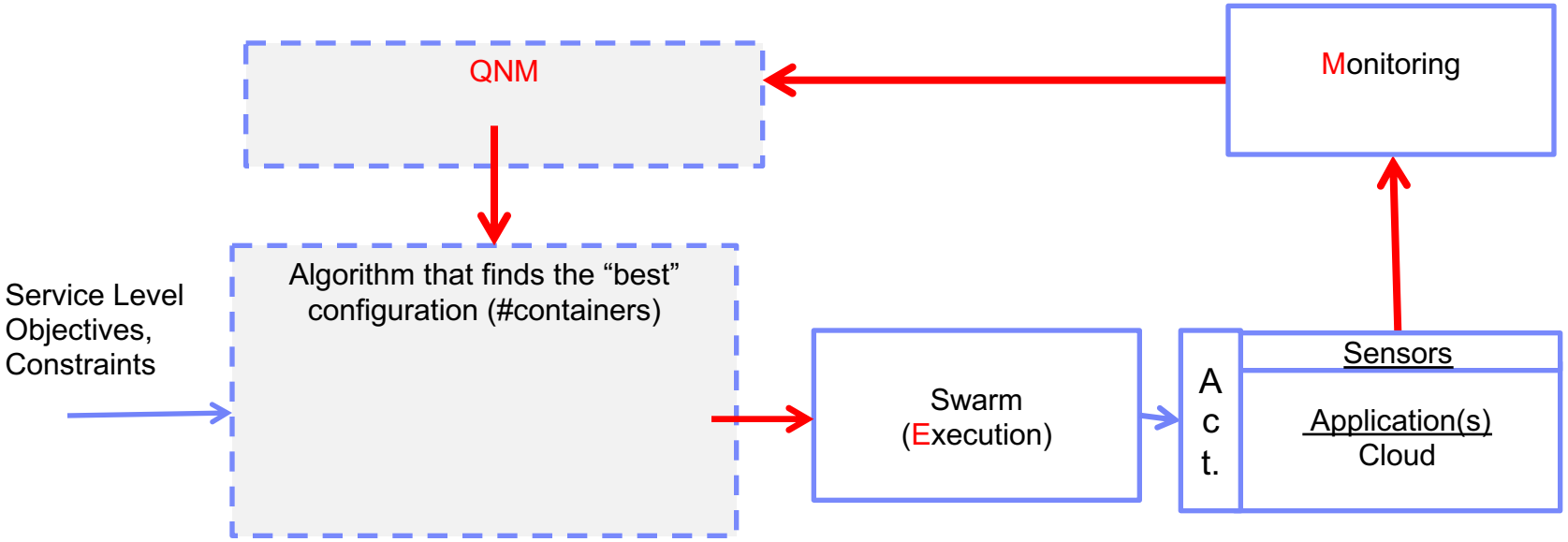
- **How do we build an accurate queuing model for a specific architecture**
  - How do we know all queues
  - How do we measure the demand Di?
    - It is easier to measure Ui and Xi
      - Di=Ui/Xi?
  - How do we learn the model at runtime
    - The model should be a good approximation of the system
  - How do we use the model at runtime for adaptive decisions?

# Your turn

- **You have your system described by an open or closed model**
- **How would you implement an autonomic manager using the model**
  - Y=?
  - U=?

# Model-based Adaptive Software



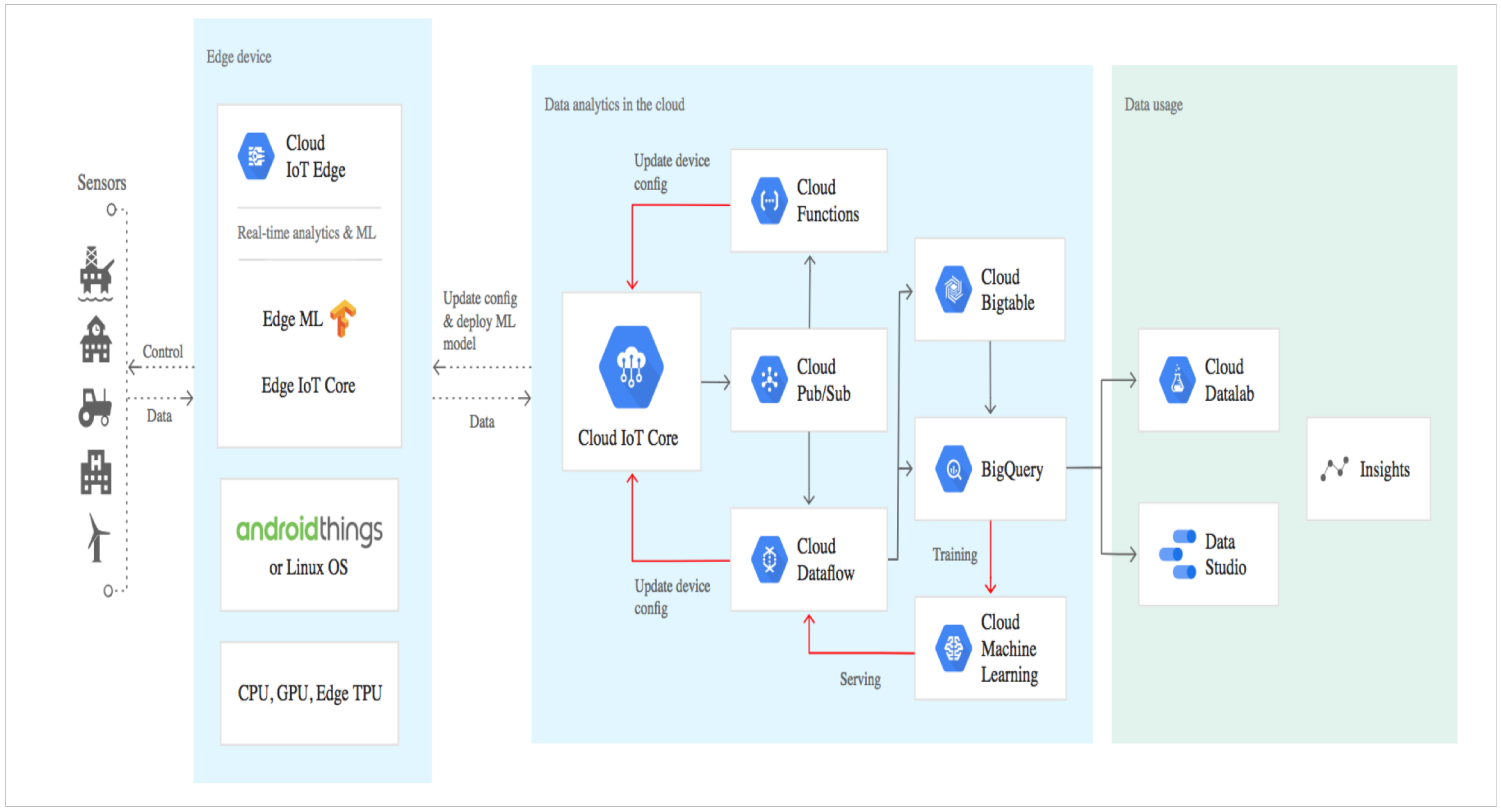



# Big Challenges

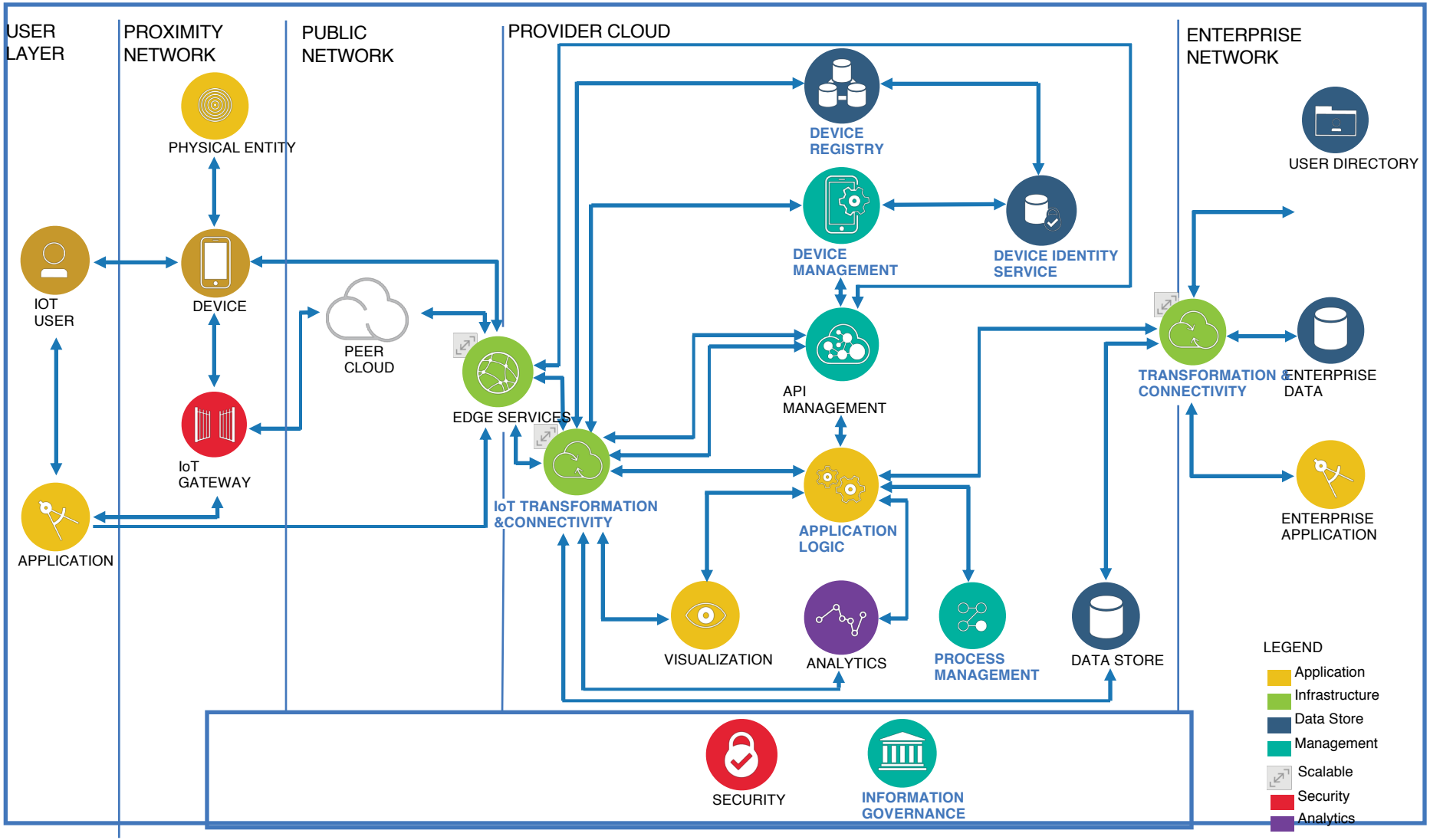
- **How do we model the IoT Platforms?**
  - See the next two slides
- **What are the big issues in IoT Platforms?**

# Google IoT Infrastructure

- **Google: https://cloud.google.com/solutions/iot/**



# IBM IoT Infrastructure



# Research ideas for the project

- **Use static QNMs ( models computed off-line)**
  - How accurate are models at runtime when we change the deployment in each tier ($m_i$)
  - Compare PID based adaptation versus QNM based adaptation
- **Can we "learn" QNMs at runtime?**
  - Can we use estimators to estimate parameters $D_i$
  - Can we use Di=Xi/Ui as a learning method at runtime?

# Summary

- **Performance metrics**
  - Utilization, response time, throughput
- **Scalability depends on how fast ONE transaction is, that is depends on D (demand); minimizing D makes an application more scalable;**
- **Fundamental laws**
  - Utilization law
  - Little's law
- **Queuing Network Models**
  - Open Models
  - Closed Models

# Summary

- **Multiple tiers distribute the work (D) across many computers, each computer doing a fraction of D ( of one user transaction)**
  - 3 tiers scale better than 2 tiers
  - However, if one fails in the chain, the whole system is down
- **Load balancing distributes the user requests across multiple identical machines**
  - QNMs can help with better decisions
  - Many open research questions