# APLICATII DISTRIBUITE IN CLOUD

Servicii AWS. SQS. Autoscalare.

# AWS: Infrastructure as a service

- O mare ferma de masini disponibile la cerere
- Platesti doar pentru ce folosesti
- Servicii peste ferma de masini:
  - Monitorizare
  - Auto scalare
  - Imagini preconfigurate pentru masini

# Servicii AWS

Retelistica si putere de calcul:

- EC2 – Masini pentru uz general
- VPC – Retea virtuala (Virtual private cloud)
- Route53 – Serviciu DNS
- Elastic MapReduce – Hadoop

# Servicii AWS

Stocare si livrare de continut

- CloudFront – Livrare de continut static ( CDN )
- S3 – Stocare nelimitata <cheie, valoare>
- Glacier – Stocare ieftina dar cu timp mare de acces

# Servicii AWS

Baze de date

- DynamoDB – NoSQL data store
- RDS – Relational database
- ElastiCache – In memory cache
- Redshift – Data warehouse ( petabyte scale )

# Servicii AWS

Gestionare

- CloudFormation – Infrastructure as code
- Elastic Beanstalk – Infrastructura pentru aplicatii web
- CloudWatch – Monitorizare
- IAM – Gestionare utilizatori, acces

# Servicii AWS

Servicii uzuale pentru aplicatii:

- SNS – Simple Notification Service
- SQS – Simple Queue Service
- SES – Simple Email Service
- Elastic Transcoder – Convertor video
- SWF – Serviciu de “Workflow”
- CloudSearch – Motor de cautare

# Argumente PRO Cloud

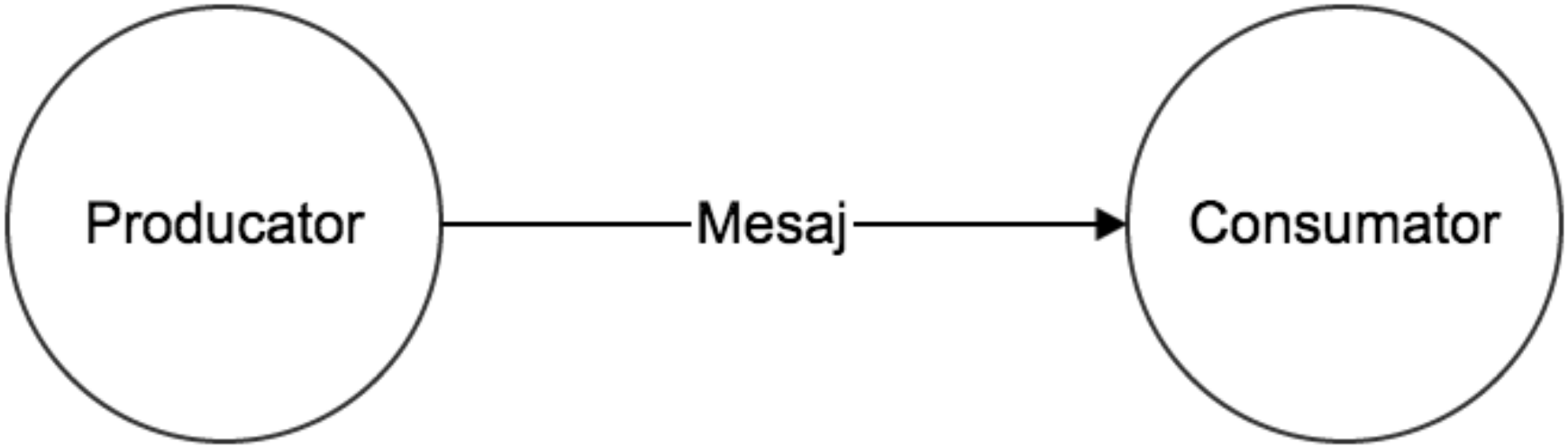
- Efortul depus este concentrat pe aplicatie, nu pe infrastructura
- Putere “nelimitata” de calcul
- Platesti pentru ce folosesti
- Infrastrucutra se auto ajusteaza pentru traficul din aplicatie

# Argumente CONTRA Cloud

Ai ales un provider, cu el ramai!

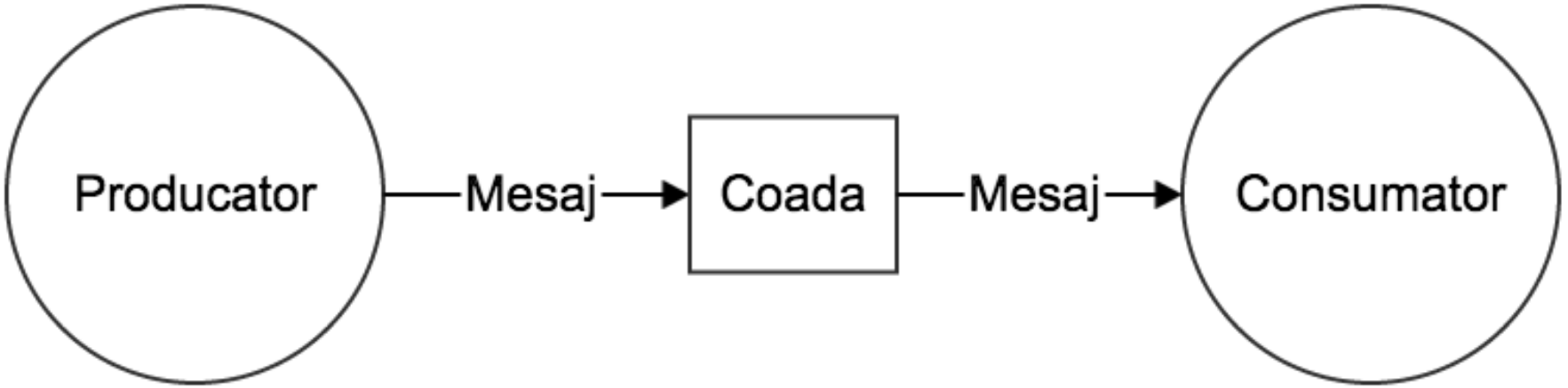
# Sisteme de cozi

# Producator consumator



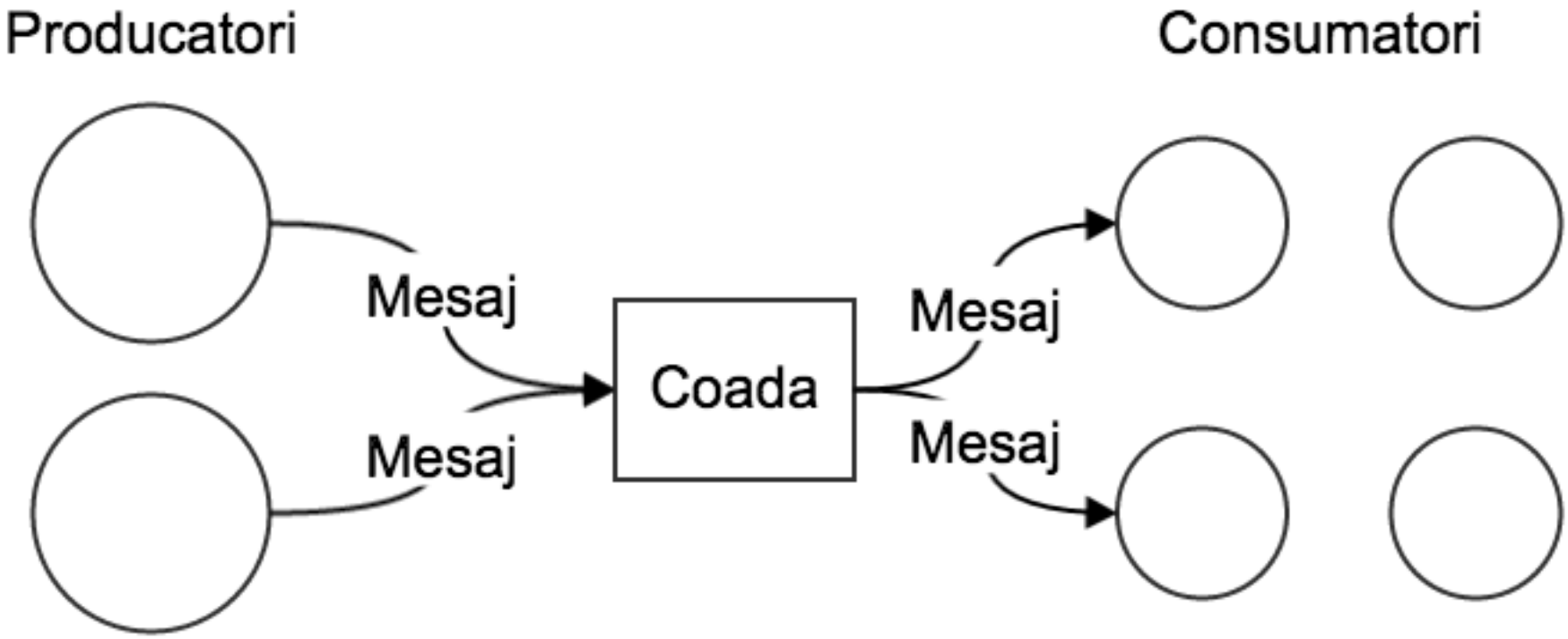

!Producatorul trimite mesaj direct consumatorului

# Producator consumator




!Producatorul pune mesajele in coada, iar consumatorul le extrage din coada

# Producator consumator

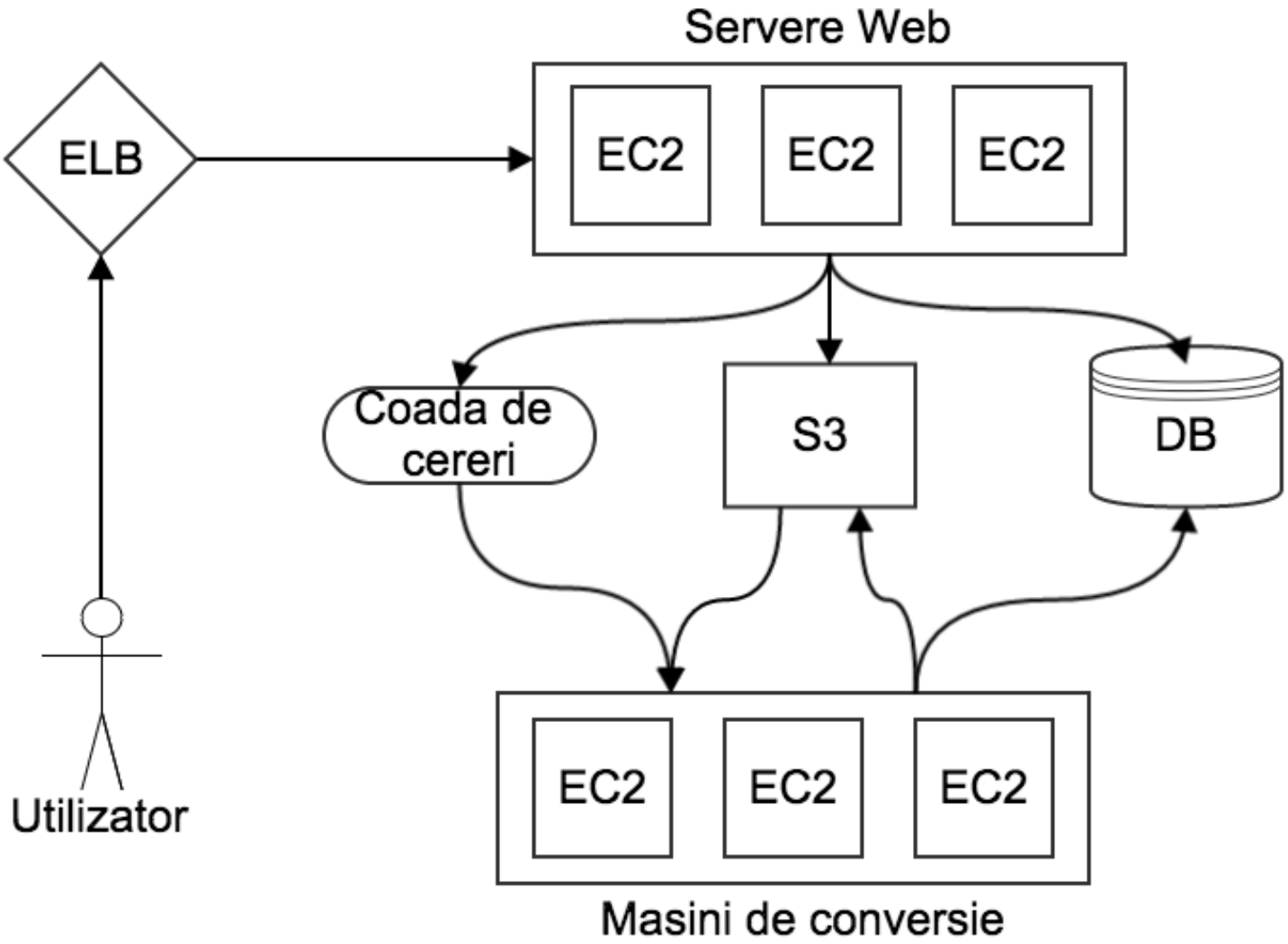



!Numar variabil de producatori si consumatori

# Observatii

- Producatorii sunt independenti de consumatori.
- In general, viteza cu care sunt produse mesajele difera de cea cu care sunt consumate.
- Coada joaca si un rol de tampon protejand consumatorii de prea multe mesaje ( flood )

# Exemplu: Serviciu de conversie video



# Exemplu: Serviciu de conversie video

- Utilizatorul trimite cererea de conversie si fisierul original catre masinile web prin ELB.
- Masinile web salveaza fisierul in S3, adauga o inregistrare in baza de date (starea conversiei) si pune un mesaj in coada ( contine id-ul inregistrarii din DB si calea catre fisierul din S3 ).
- Masinile de conversie extrag mesajul din coada si downloadeaza fisierul din S3. In timp ce fisierul original este convertit, progresul este actualizat in DB.

# Exemplu: Serviciu de conversie video

- Utilizatorul cere starea conversiei unei masini web, iar aceasta intergheaza DB-ul si apoi raspunde.
- Cand masina de conversie termina, ea salveaza fisierul nou inapoi in S3.
- Utilizatorul cere fisierul nou unei masini web, iar aceasta il serveste direct din S3.

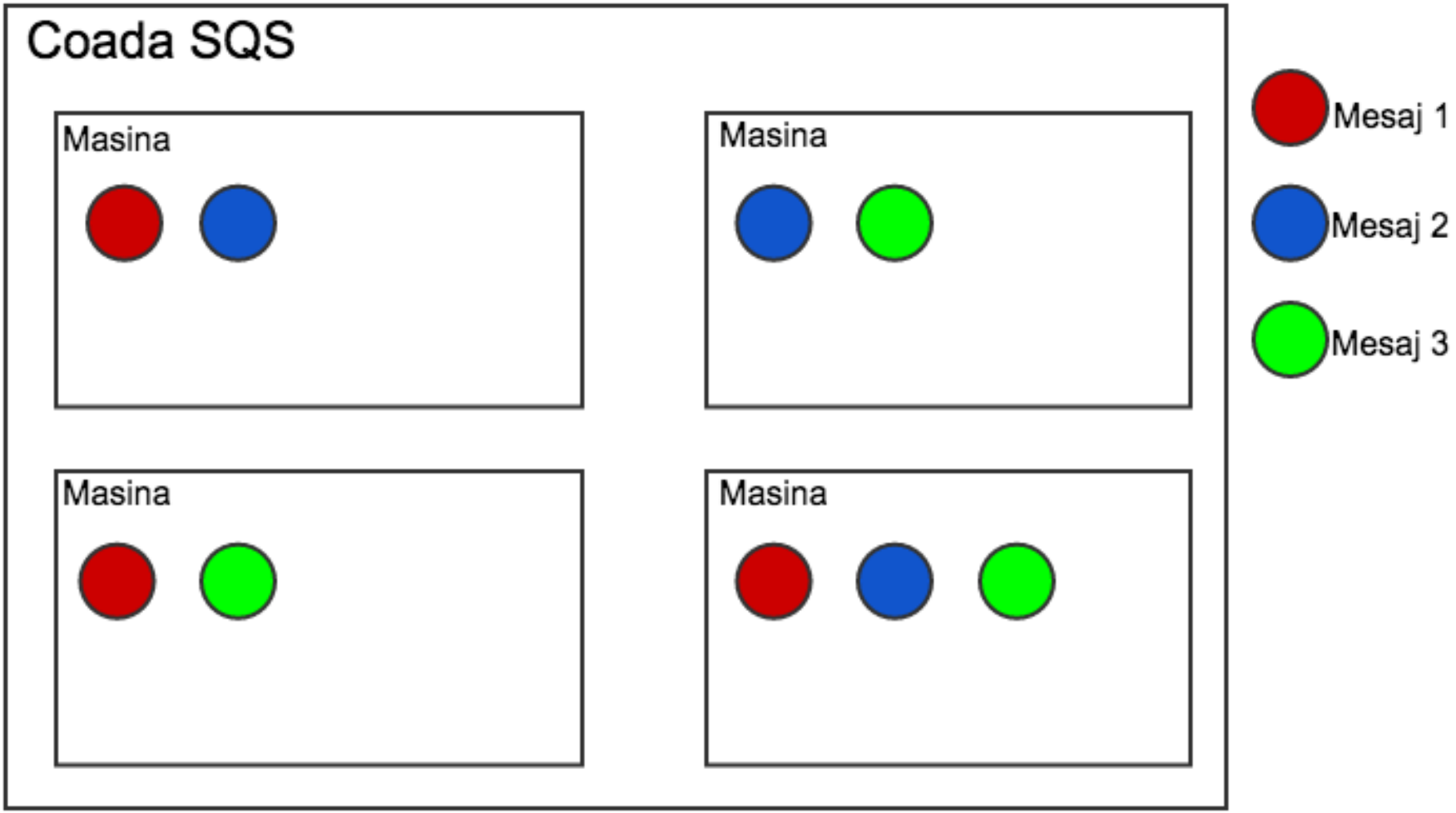
# Sisteme de cozi

- Un tabel intr-o baza de date SQL.
- Kestrel
- RabbitMQ
- Amazon SQS
- etc

# Amazon SQS

- Implementare proprie (closed source)
- O coada este distribuita pe mai multe masini
- Gestionata de amazon
- Dimensiune maxima a mesajului ( 256KB )
- Dupa citire, mesajul nu este sters ci este marcat ca invizibil pentru o perioada. Cand perioada expira mesajul devine din nou disponibil pentru consumator.
- Mesajul trebuie sters explicit.
- !Nu se garanteaza ca mesajul este citit o singura data!

# Amazon SQS: Multi redundanta



# Amazon SQS: API

- Creare, listare, stergere de cozi.
- Trimiterea unui mesaj, sau unui grup de mesaje.
- Citirea mesajelor se face in grup.
- Modificarea vizibilitatii unui mesaj, sau a unui grup de mesaje.
- Stergerea unui mesaj sau a unui grup de mesaje.
- Obtinerea si modificarea atributelor cozii ( ex. timpul de invizibilitate )

# Amazon SQS: Exemplu cod

```
AmazonSQSClient sqs = new AmazonSQSClient(credentials);
SendMessageRequest request = new
        SendMessageRequest("adresa-coada", "mesaj1");
SendMessageResult result = sqs.sendMessage(request);
```

# AutoScalare

# AutoScalare

- Un sistem care gestioneaza propria putere de calcul
- Monitorizeaza propriile statistici si reactioneaza la schimbari adaugand sau stergand resurse.

# AutoScalare: Statistici utilizabile

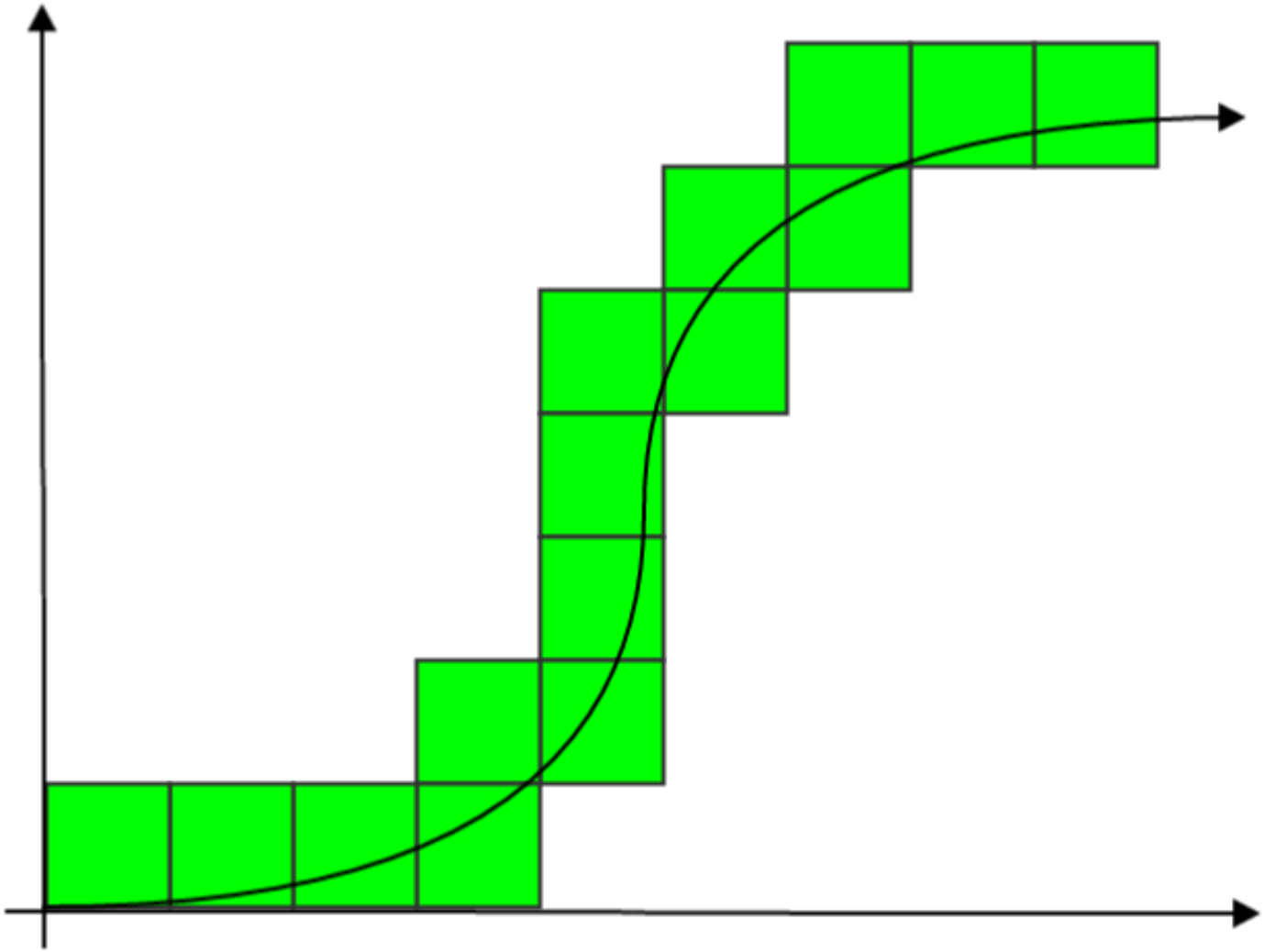
- Media de utilizare a procesorului
- Traficul de date pe o interfata de retea
- Numarul de mesaje dintr-o coada
- O anumita ora din zi
- etc

# AutoScalare: marimea unei masini

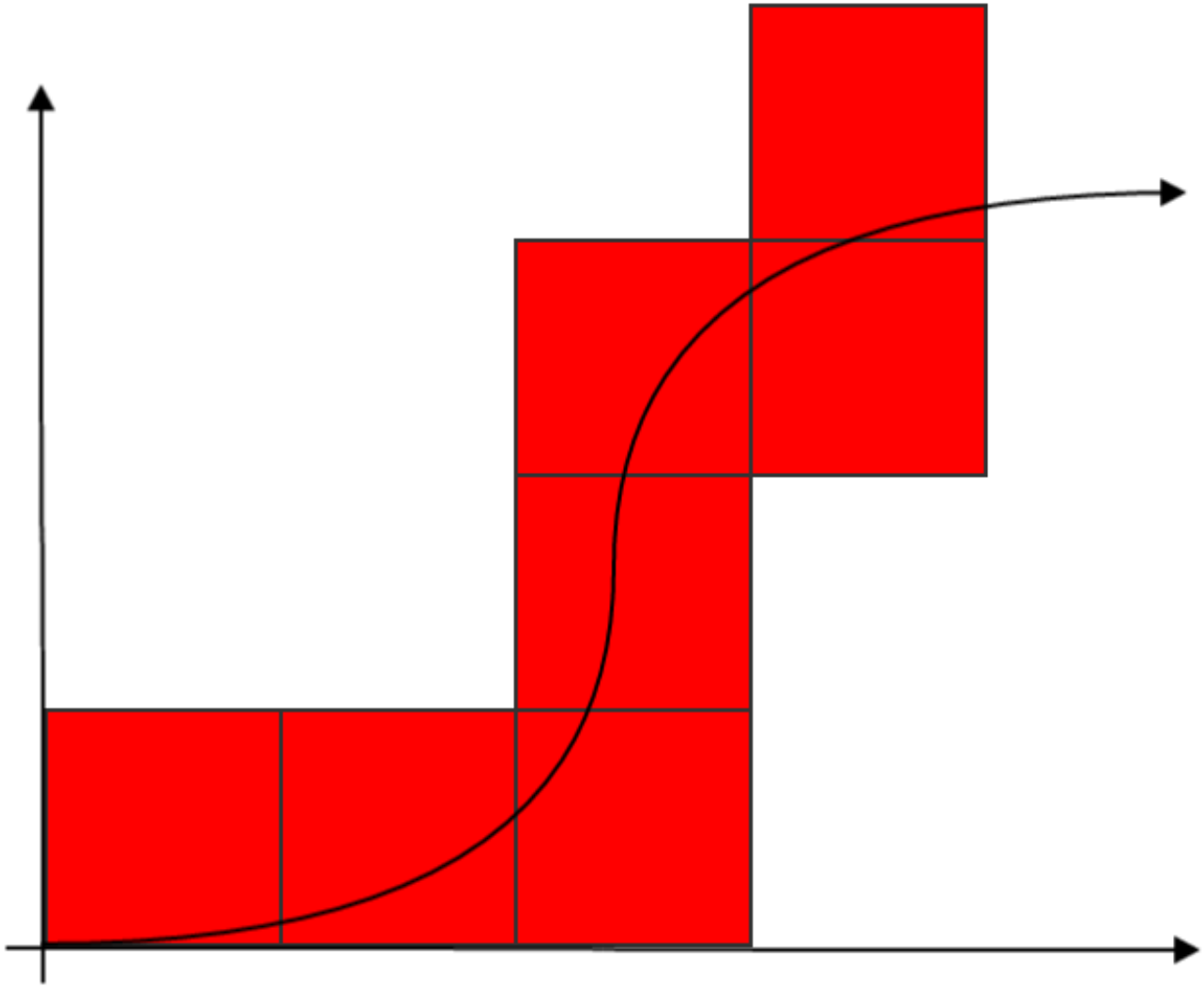
Ca sistemul sa fie functional, la orice moment de timp puterea de calcul a masinilor trebuie sa fie mai mare decat ce este nevoie pentru a servi toate cererile.

- Cu cat mai mare?
- Multe masini mici, sau mai putine dar mari?
- Cate masini adaugam/stergem pentru a mentine sistemul ruland?
- Raspuns: Depinde ce vrem sa minimizam!

# AutoScalare: Masini mici



# AutoScalare: Masini mari



# AutoScalare: Minimizarea costului

- Tot ce este deasupra graficului, sunt resurse pentru care platim si nu sunt utilizate.
- Cand folosim masini mici, alocam putine resurse peste ce este nevoie minimizand costul resurselor care sunt in asteptare.

# AutoScalare: Minimizarea timpului de raspuns

- Sistemul se ajusteaza pentru un trafic mai mare adaugand o masina la un moment dat.
- Dupa adaugarea unei masini, sistemul asteapta 10 minute sa se stabilizeze.

# AutoScalare: Minimizarea timpului de raspuns

Cand traficul creste exponential cu masini

- Mici: adauga 4 masini in timp de 40 de minute. Timp in care sistemul se misca mai greu.
- Mari: adauga doar 2 masini in 20 de minute. Dupa primele 10 minute poate servi mai multe cereri decat cel cu masini mici.

# AutoScalare: Observatii

- Pentru rezultate mai bune, scaleaza proactiv NU reactiv.
- Trebuie gasit un balans intre cost, timp de reactie, suportarea spike-urilor etc.

# Intrebari