résumé de flux de données

CLEROT Fabrice

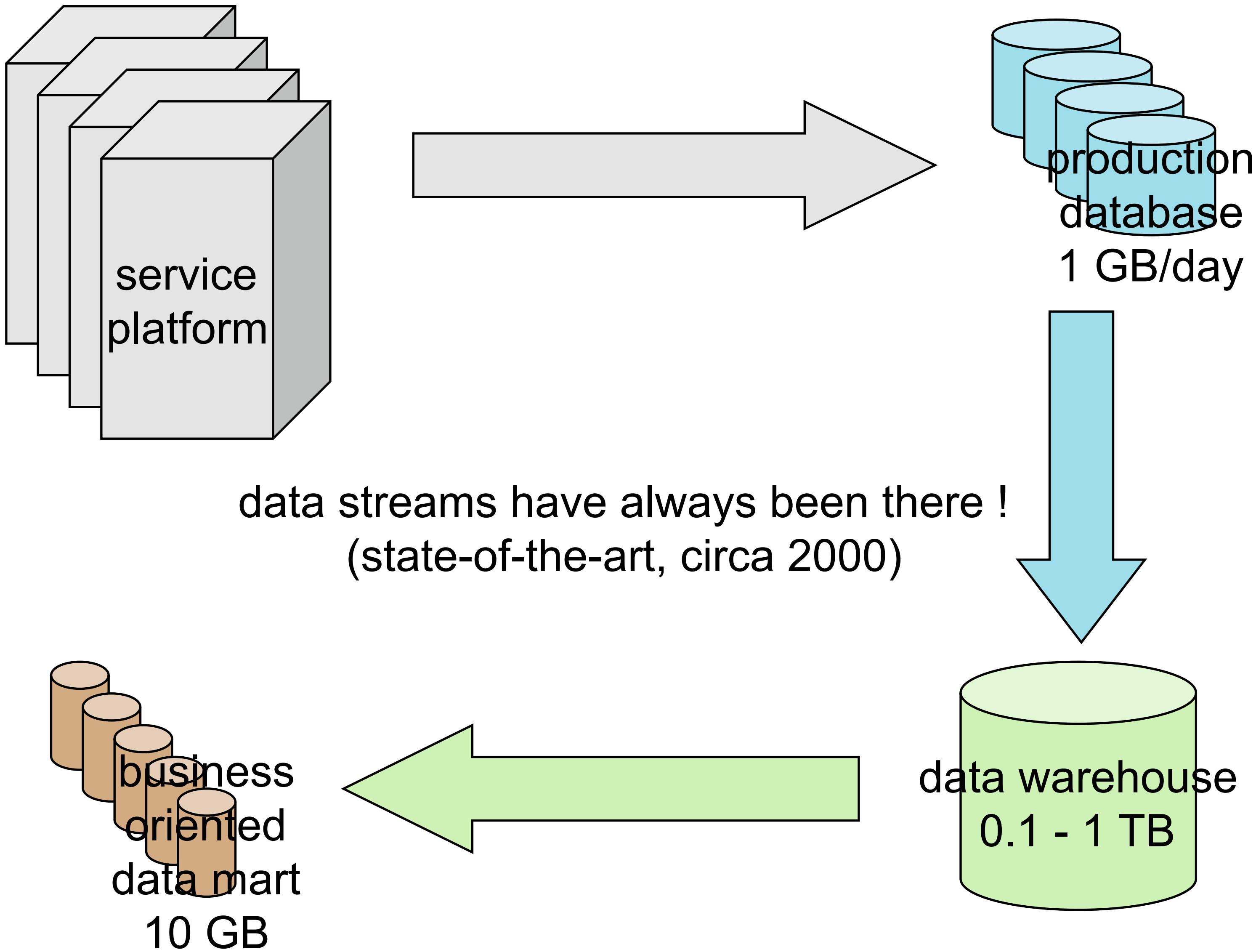
fabrice.clerot@orange-ftgroup.com

Orange Labs

data streams, why bother ?

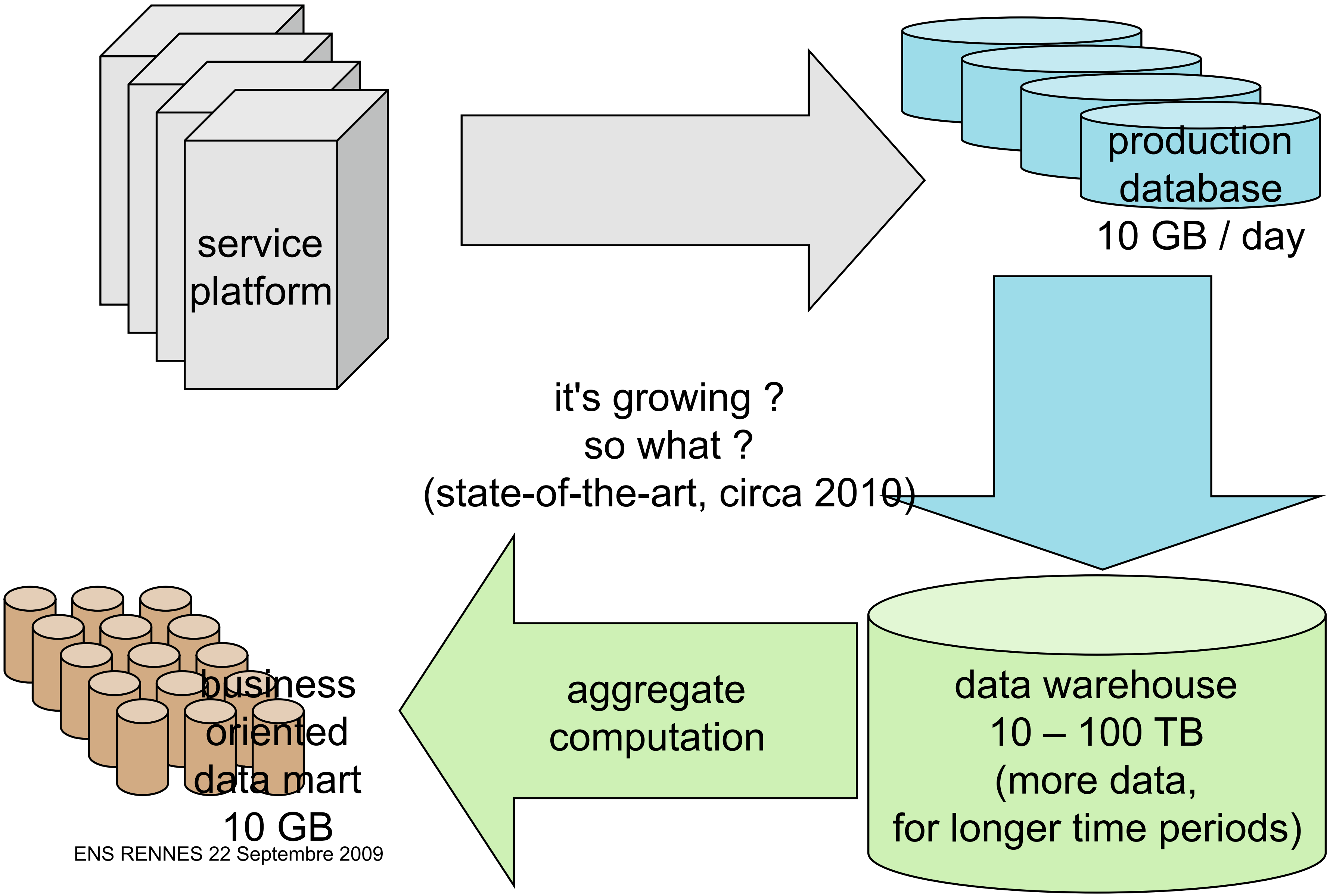
massive data is the talk of the town ...

data *streams*, why bother ?



data streams have always been there !
(state-of-the-art, circa 2000)

data *streams*, why bother ?



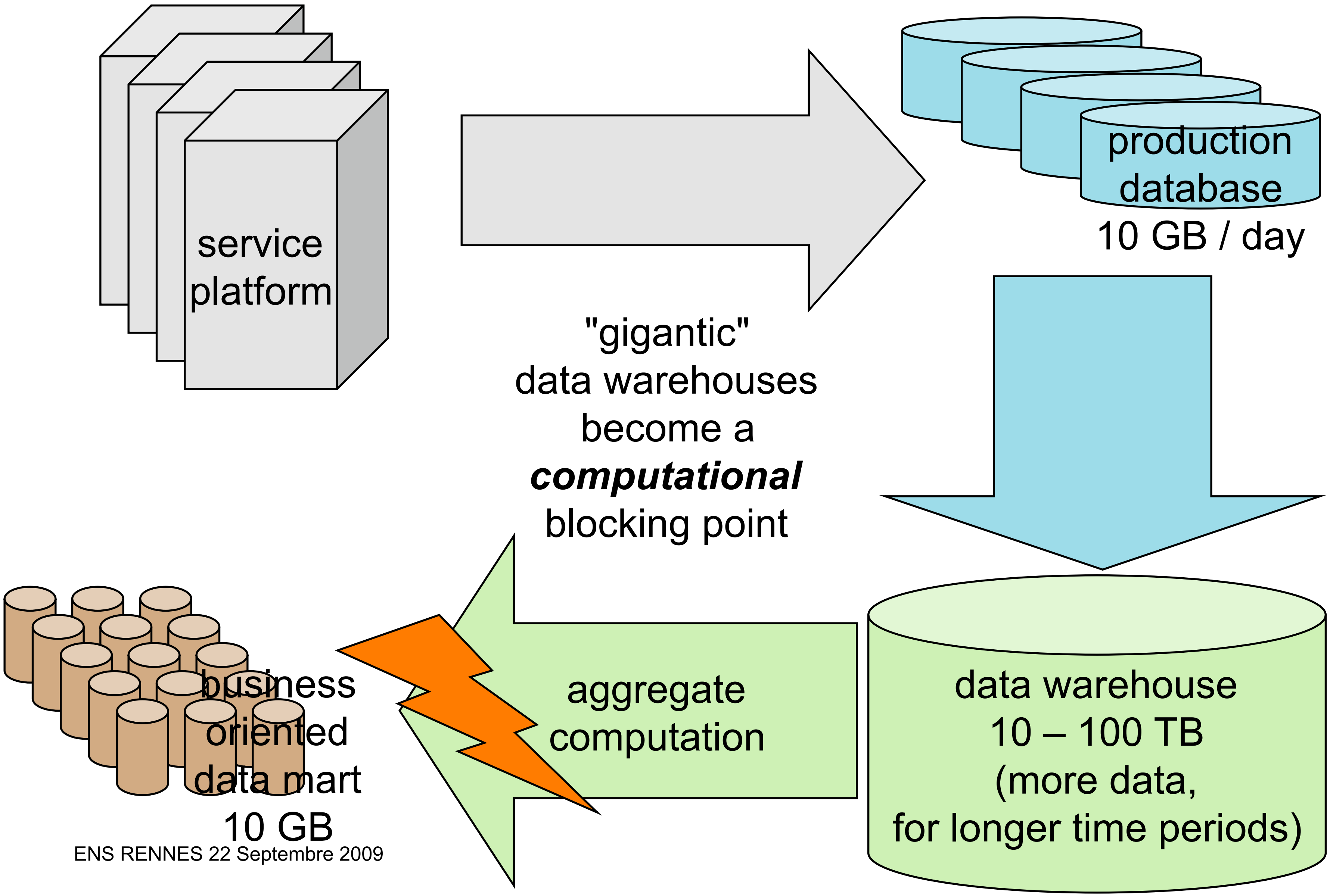
data streams, why bother ?

- moore's law (cpu) and kryder's law (storage) have roughly the same exponent
  - performance for unit cost doubles every 18 months
- so, indeed why bother ?

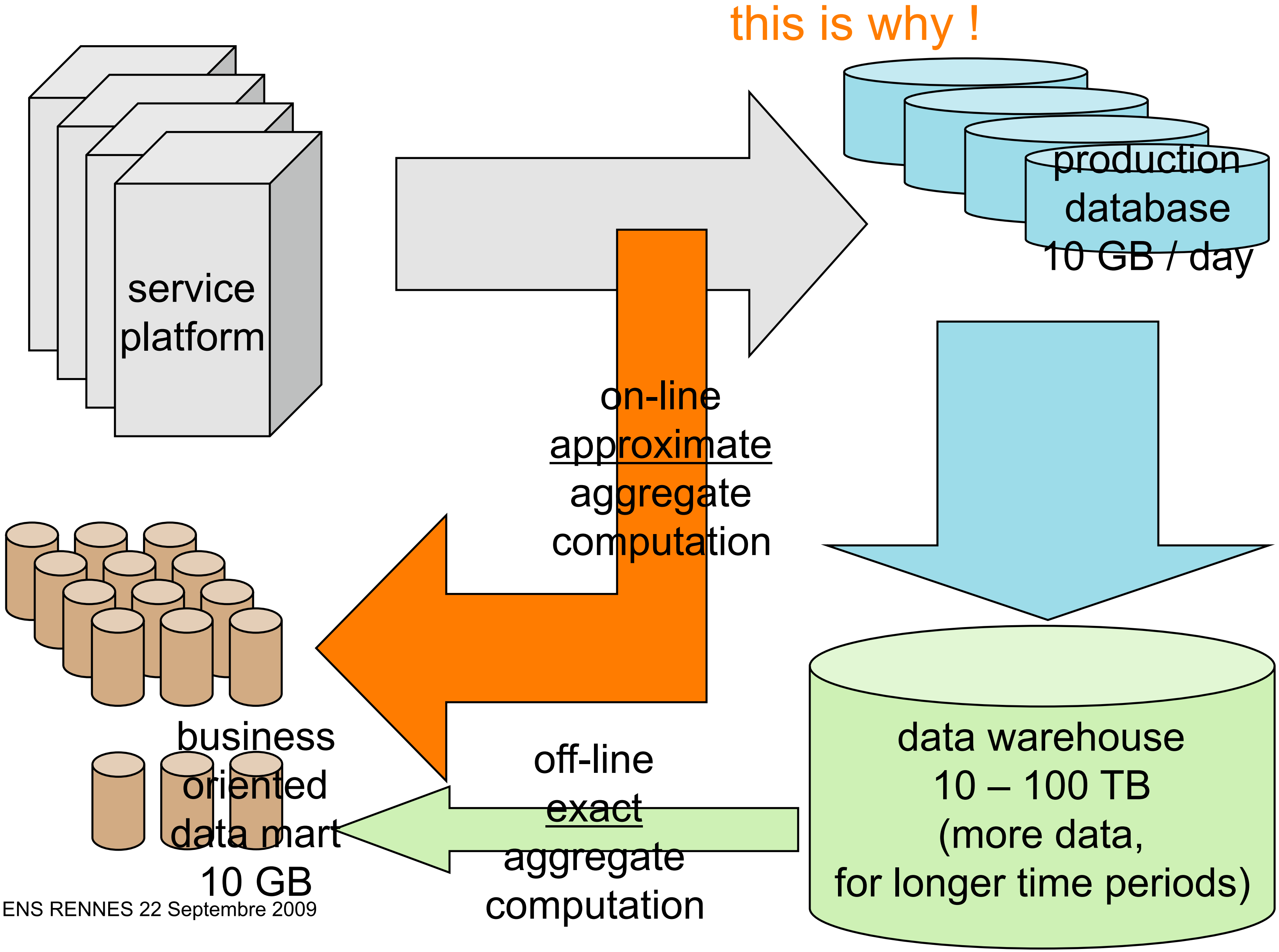
- <u>blindingly</u> obvious :
  - most *exact* data processing operations scale worse than O(n), often very much worse, as $O(n^2)$
    - even *sort* scales in O(nlog(n))

- petabytes irresistibly crawl their way to defeat teraflops !

data *streams*, why bother ?



data *streams*, why bother ?



... but small data also matters !

(almost) brainless chatters ...

- ... otherwise known as " (remote) sensors"

- potentially thousands of them in an ad hoc communication network, millions to come ("digital dust")
- very limited power autonomy : communication and processing drastically reduce the lifetime of the sensors

- processing data at the stream level is a way to reduce the cost of communication and processing
  - for instance, compute the mean of a measurement as data are sent to a base station *through* the ad hoc network instead of transmitting raw data to the base station

plan

- flux de données vs bases de données
- résumé vs "stream-mining"
- quelques résumés simples
- résumé de la jointure de deux flux de données

data stream vs data base

data base

mining

processing

management

data stream vs data base

data base

mining

volatile queries

on

persistent data

processing

management

data stream vs data base

data base

mining

volatile queries

on

persistent data

processing

centralized

management

data stream vs data base

data base

mining

volatile queries

on

persistent data

processing

centralized

management

load new data

for

existing queries

data stream vs data base

data base		***data stream***
	mining	
volatile queries on persistent data		persistent queries on volatile data
	processing	
centralized		distributed
	management	
load new data for existing queries		keep existing data for new queries

data stream vs data base

data base		***data stream***
	mining	
volatile queries on persistent data		persistent queries on volatile data
	processing	
centralized		distributed
	management	
load new data for existing queries		keep existing data for new queries

infinite streams
limited CPU
limited memory

data stream vs data base

data base		***data stream***	
	mining		
volatile queries on persistent data		persistent queries on volatile data	infinite streams limited CPU limited memory
	processing		
centralized		distributed	limited bw (cf Cormode)
	management		
load new data for existing queries		keep existing data for new queries	

data stream vs data base

data base		***data stream***
	mining	
volatile queries on persistent data		persistent queries on volatile data
	processing	
centralized		distributed
	management	
load new data for existing queries		keep existing data for new queries

infinite streams
limited CPU
limited memory

limited bw

(cf Cormode)

generic summary

"generic summary" vs "stream-mining"

- stream-mining
  - persistant queries valid for all the stream duration
  - the result of the query is a new data stream (in general)

  - if the application allows the <u>pre-definition</u> of <u>all</u> useful queries, this formalism answer all the applicative requirements

- what about a <u>new query on past data</u> ?
  - data are volatile and cannot be accessed anymore
  - no pre-defined query has been set, so no answer can be retrieved
  - a generic summary aims at allowing the (approximate) execution of such queries (with confidence bounds on the result)

"generic summary" vs "stream-mining"

- and what about a persistent query on two sliding windows
  - one on the immediate history
  - one on the distant past
    - warning … *persistent* query on *past* data …

what is the **present average invoice**
of the **10% best clients at the same period last year** ?

generic summary and density estimation

- a data-stream is a distribution on TxD
  - T description space for time
  - D description space of events
    - typically, (@, Value)
- a generic summary can be seen as a density estimation problem on TxD
  - under memory, cpu, bw constraints

generic summary and density estimation

- "ad hoc" approaches, however, with separate treatments of time and events
- CluStream (Aggarwal)
  - mixture of gaussians for the density estimation of event space
    - limited to numerical description of events
  - logarithmic "cliches" structure for time
- HClustream (Yang) or SCLOPE (Ong):
  - claim to extend Clustream to symbolic data by keeping the frequencies of the modalities per gaussian
  - clearly do not scale for large address space (for instance)

generic summary and sampling

- downgrade density estimation to sampling
  - generic summary $\sim$ keep a representative sample of the stream events
- constraints
  - limited memory : bound on the summary size
  - limited cpu : bound on the per event processing
  - distributed processing
    - limited bw : bound on the communication requirements
    - limited cpu : bound on the computational cost to get the global summary from the local summaries

some simple summaries

- uniform sampling of a stream
- weighted sampling of a stream
- uniform sampling from a sliding window on a stream
- weighted sampling from a sliding window on a stream

(over-)simplified steam model

- infinite sequence of events e
  - e.time is an integer
  - e.data is a description of the event
- perfect observation
- perfect time-ordering

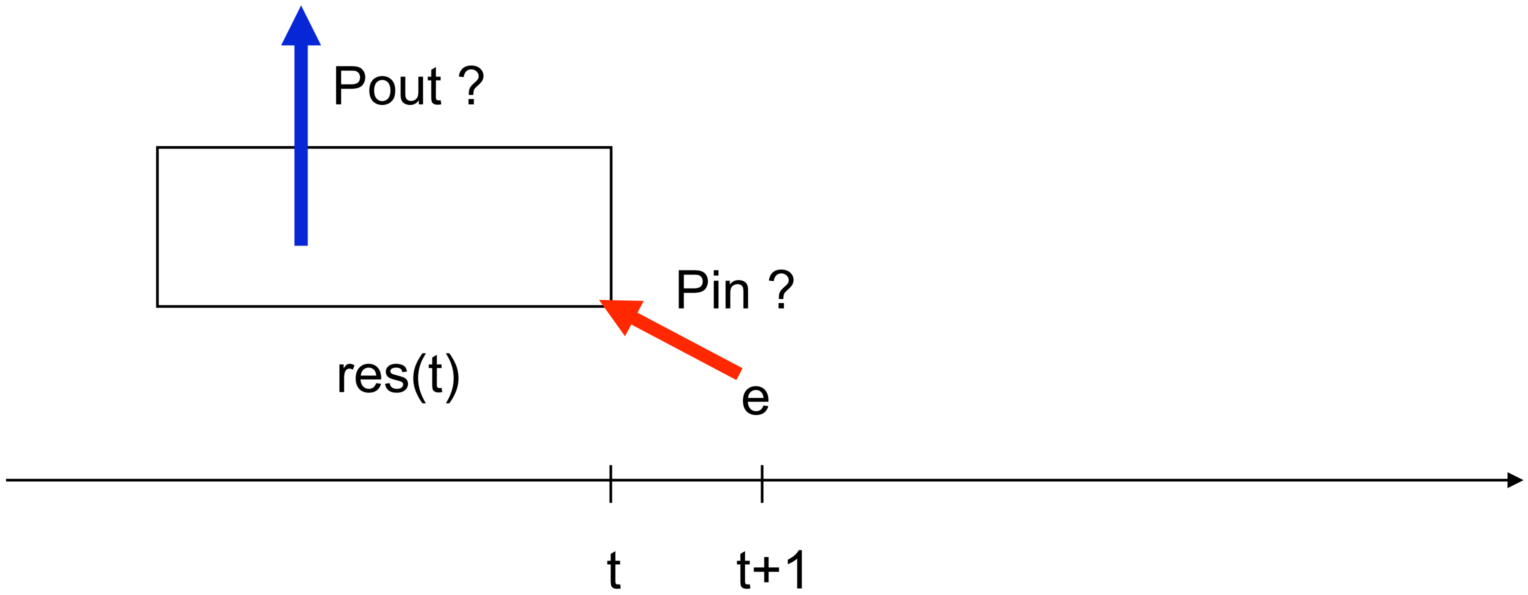
uniform sampling of a stream

- at any time, the probability for an event to be in the sample is uniform with respect to the complete history of the stream

- sample size r
  - at time t, the probability for event e (with e.time $\leqq$ t) to be in the sample is r/t

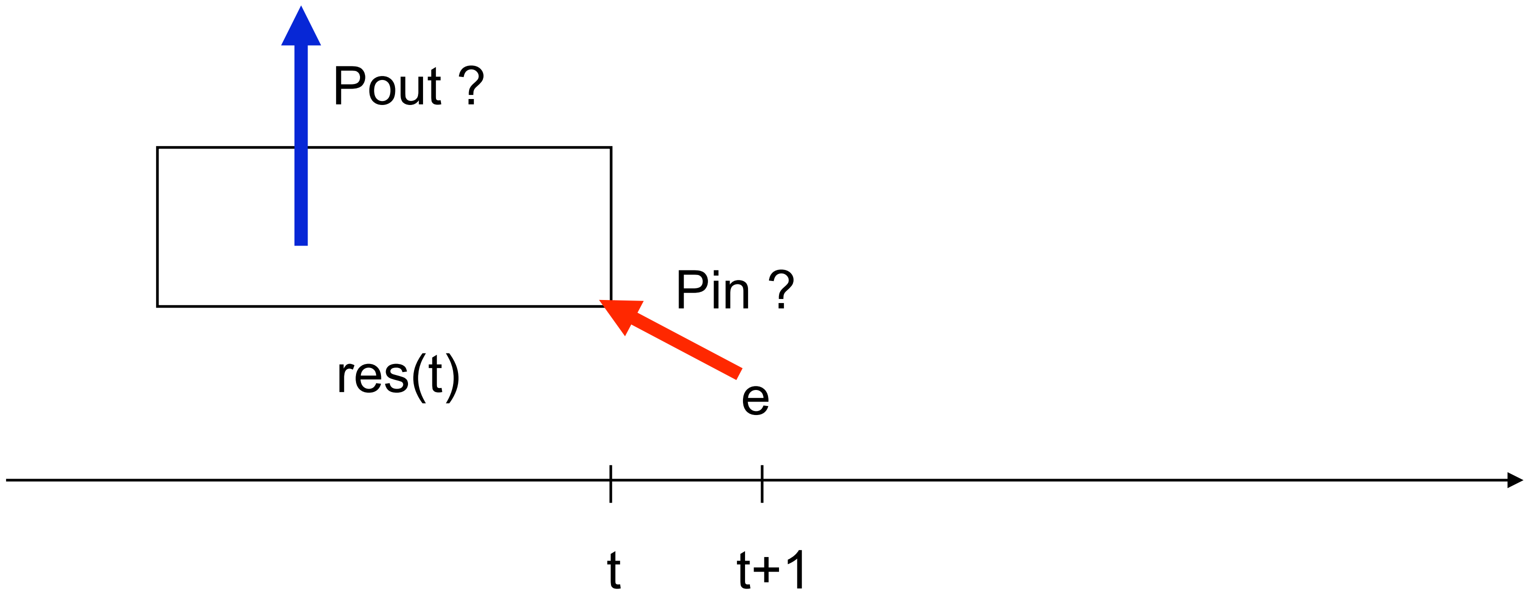
- time evolution of the sample : res(t)
  - condition for a new event to be sampled ?
  - condition for an event in the sample to be excluded ?

reservoir sampling



reservoir sampling

Pin = r/(t+1)

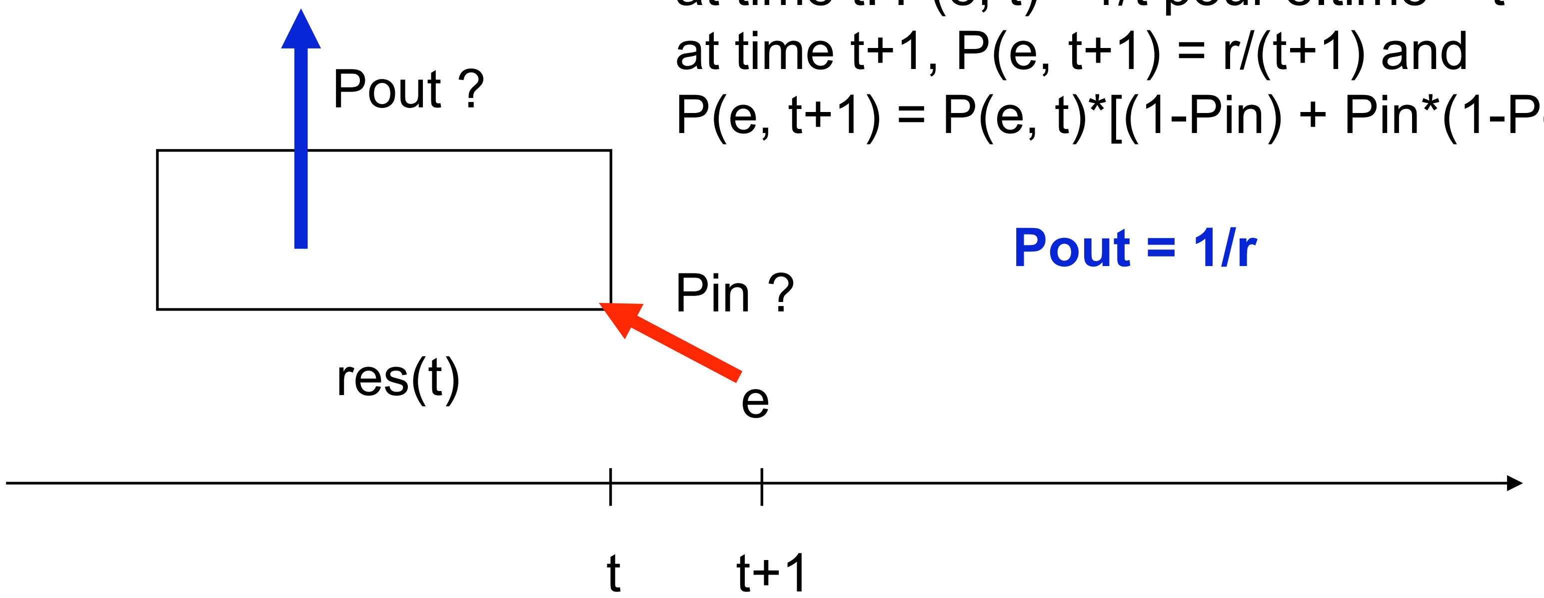


reservoir sampling

Pin = r/(t+1)

for all events in the reservoir:
at time t: P(e, t) = r/t pour e.time < t+1
at time t+1, P(e, t+1) = r/(t+1) and
P(e, t+1) = P(e, t)*[(1-Pin) + Pin*(1-Pout(e))]

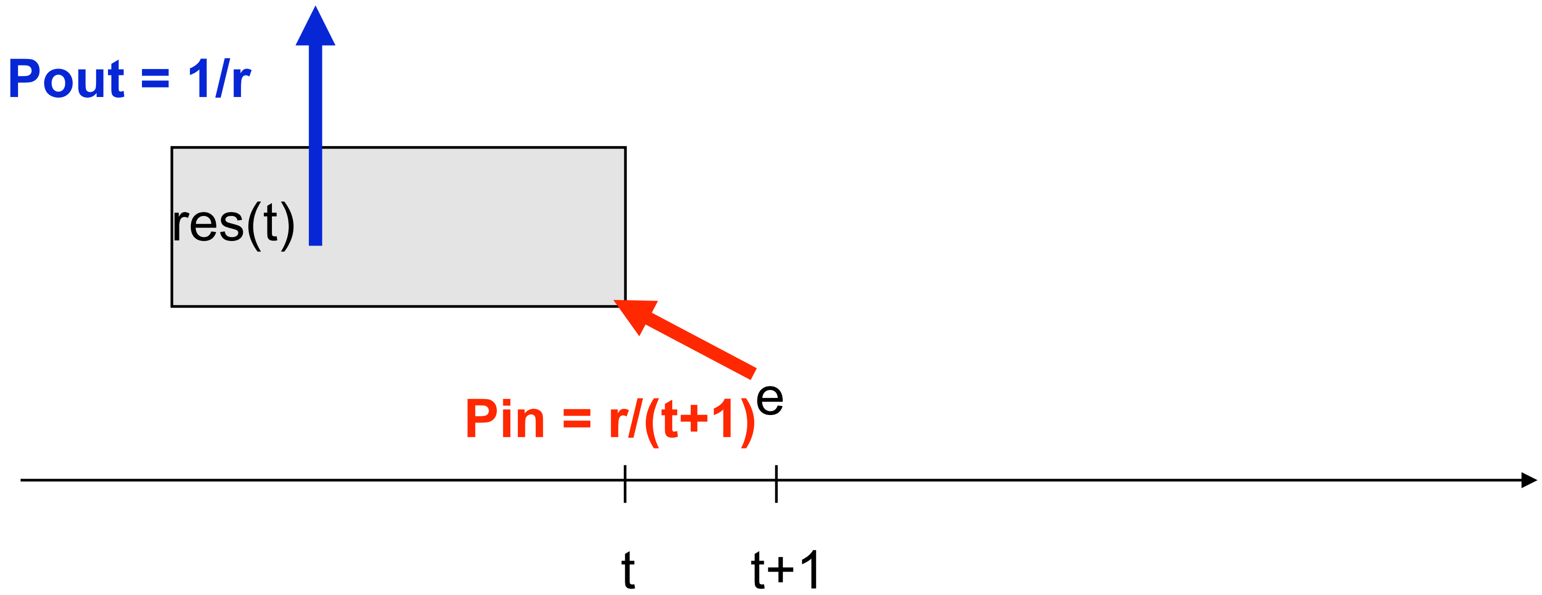
Pout = 1/r



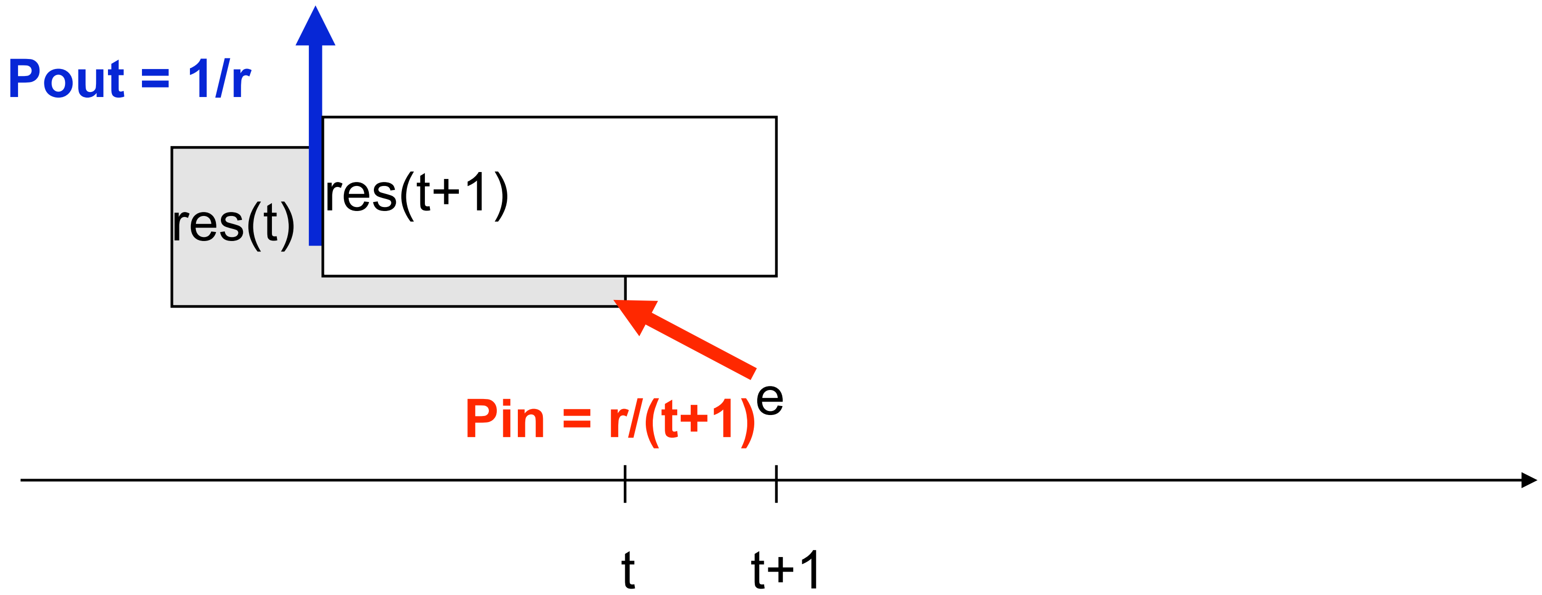
reservoir sampling (Vitter)

- for t < r+1, place all events into the reservoir
- for t > r
  - place a new event in the reservoir with probability r/t
  - if a new event is sampled, exclude one event from the reservoir randomly

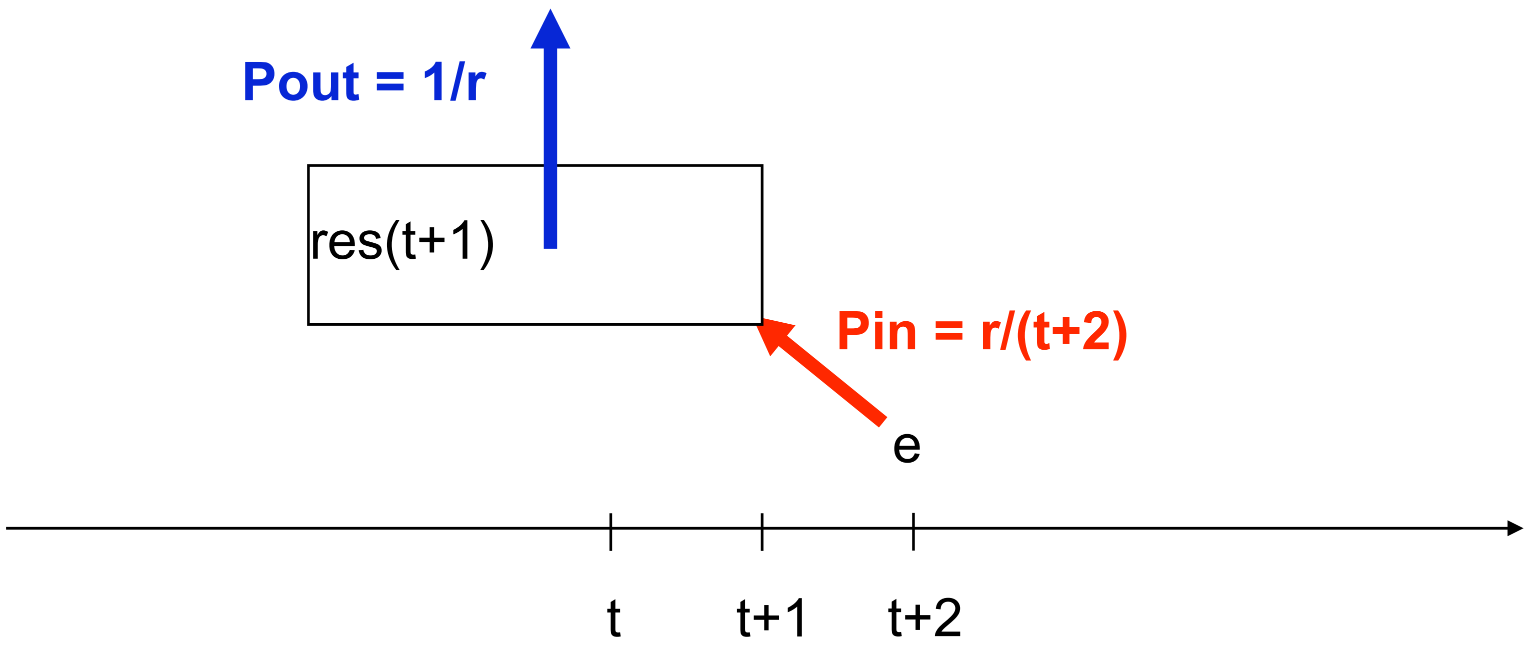
reservoir sampling



reservoir sampling



reservoir sampling



what about the constraints ?

- limited memory : size of the reservoir set a priori
- limited cpu :
  - naive implementation : one random draw per event plus another in case of success
  - in fact, only one random draw is enough ...
  - ... and much less : after an insertion, draw the time of the next insertion

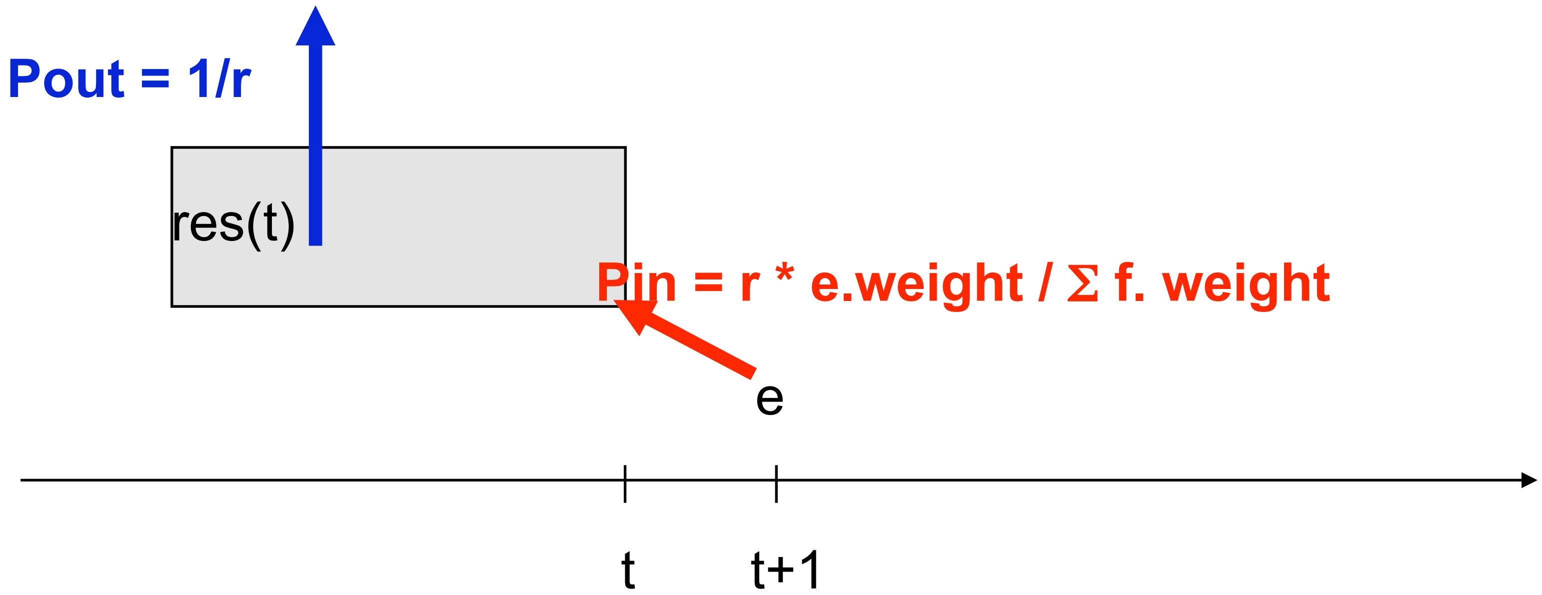
what about the constraints ?

- distributable: summary of a set of streams ϕ_i
  - local summaries of the same size
  - **res (MUX_i ϕ_i , t) = U_i res (ϕ_i , t)**
    - the summary of the multiplex is just the union of the summaries of the original streams,
  - either transmit the local summaries to a collector when required
  - or transmit the local updates to a collector
    - limited bw to the collector
    - no communication between streams

weighted sampling of a stream

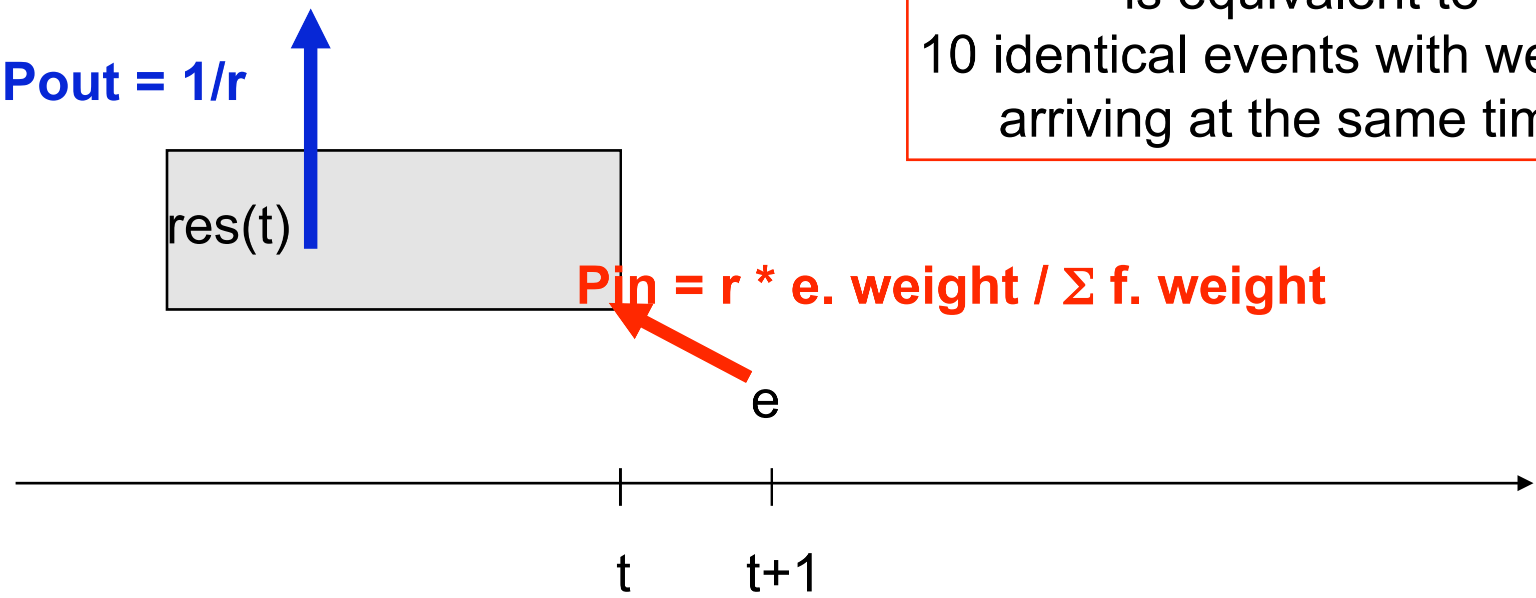
- a weight is associated to all events, e.weight
- at any time t, the probability of a past event …
  – e.time < t+1
- … to be in the sample is its relative weight with respect to the total weight of the past
  – Prob(e in res(t)) = e.weight / $\Sigma_{f.time<t+1}$f.weight
- time evolution of the sample : res(t)
  – condition for a new event to be sampled ?
  – condition for an event in the sample to be excluded ?

weighted reservoir sampling



$$W(t) = \sum_{f.time < t+1} f.weight \longrightarrow W(t+1) = W(t) + e.weight$$

weighted reservoir sampling



interpretation:
"1 event with weight 10
is equivalent to
10 identical events with weight 1
arriving at the same time"

$$\mathbf{W(t) = \sum_{f.time < t+1} f.\ weight \longrightarrow W(t+1) = W(t) + e.\ weight}$$

what about the constraints ?

- limited memory : size of the reservoir set a priori
- limited cpu :
  - naive implementation : one random draw per event plus another in case of success
  - in fact, only one random draw is enough ...
  - ... but cannot do less : future weights are unknown so you cannot draw the next insertion time

quid de nos contraintes ?

- distributable: summary of a set of streams ϕ_i
  - local summaries of the same size
  - the summary of the multiplex is the union of *resampled* summaries of the original streams according to the respective weights of the streams
    - n local reservoirs of size r will produce a global reservoir of size less than n*r
    - the size of the sample is bounded but unknown in advance
  - either transmit the local summaries to a collector when required
  - or transmit the local updates to a collector
    - limited bw to the collector
    - no communication between streams

uniform sampling from a sliding window on a stream

- at any time t, the probability for an event of [t-τ, t] to be in the sample is uniform on [t-τ, t]
  - zero probability for events in [0, t-τ-1]
- sample size r (< τ)
- time evolution of the sample : res(t)
  - condition for a new event to be sampled ?
  - condition for an event in the sample to be excluded ?
  - *how to deal with "expiring" events of the sample ?*
    - *at time t, e in res(t-1) and e.time = t-τ -1 "expires"*

why reservoir sampling doesn't work with sliding windows

- suppose an element in the reservoir expires
- need to replace it with a randomly-chosen element from the current window
- however, in the data stream model we have no access to past data
  - so we cannot sample from the "current" window, it's gone !
- we could store the entire window but this would require $O(\tau)$ memory

chain-sample (Babcock)

- initialisation: reservoir sampling on the first window
  - include each new element in the sample with probability 1/ min(t,τ)
  - as each element is added to the sample, choose the index of the element that will replace it when it expires
  - when the t-th element expires, the window will be (t+1…t+τ), so choose the index from this range
- once the element with that index arrives, store it and choose the index that will replace it in turn, building a “chain” of potential replacements
- when an element is chosen to be discarded from the sample, discard its “chain” as well

example: r = 1 et τ = 10

3 enters the sample : choose the *position* of its successor (*will* be 9)

3 5 1 4 6 2 8 5 2 3 5 4 2 2 5 0 9 8 4 6 7 3

9 enters the window : choose the *position* of its successor (*will* be 7)

3 5 1 4 6 2 8 5 2 3 5 4 2 2 5 0 9 8 4 6 7 3

3 expires : include its successor, 9, in the sample

3 5 1 4 6 2 8 5 2 3 5 4 2 2 5 0 9 8 4 6 7 3

3 5 1 4 6 2 8 5 2 3 5 4 2 2 5 0 9 8 4 6 7 3

what about the constraints ?

- limited memory :
  - size of the reservoir set a priori …
  - … and the number of pointers to successors can be bounded
    - mean length of a chain = e = 2.718

- limited cpu :
  - e = 2.718 random draws per event on average

what about the constraints ?

- distributable: summary of a set of streams ϕ_i
  - local summaries of the same size
  - **res (MUX$_i$ ϕ_i , t) = U$_i$ res (ϕ_i , t)**
    - the summary of the multiplex is just the union of the summaries of the original streams,
  - either transmit the local summaries to a collector when required
  - or transmit the local updates to a collector
    - limited bw to the collector
    - no communication between streams

uniform sampling from a sliding window on a stream

- a weight is associated to each event e : e.weight
- at any time t, the probability for an event of $[t-\tau, t]$ to be in the sample is its relative weight with respect to the total weight on $[t-\tau, t]$
  - zero probability for events in $[0, t-\tau-1]$
  - Prob(e in res(t)) = e.weight / $\Sigma_{t-\tau-1<f.time<t+1}$ f.weight
- sample size r ($< \tau$)
- time evolution of the sample : res(t)
  - condition for a new event to be sampled ?
  - condition for an event in the sample to be excluded ?
  - *how to deal with "expiring" events of the sample ?*
    - *at time t, e in res(t-1) and e.time = $t-\tau-1$ "expires"*

weighted chain sampling

- open problem ...
  - we do not know the future weights, so we cannot choose the position of the successors

another simple approach: oversample and select

uniform random sample from a sliding window

1. as each element arrives remember it with probability
   $p = c\, r/\tau \log \tau$; otherwise discard it
2. discard elements when they expire
3. when asked to produce a sample, choose r elements at random from the set in memory

- expected memory usage of $O(r \log \tau)$
- uses $O(r \log \tau)$ memory whp
- the algorithm can fail if less than r elements from a window are remembered; however whp this will not happen

adaptation to the ***weigthed case*** is obvious:

weighted random sampling in step 1 (if weights are known in advance) or in step 3 (otherwise: for instance, application-dependent weights)

bibliography

- Sketching Streams Through the Net: Distributed Approximate Query Tracking, G. Cormode, M. Garofalakis, VLDB, 2005
- A framework for clustering evolving data streams, C.C. Aggarwal, J. Han, J. Wang, and P.S. Yu, VLDB, 2003.
- Hclustream: a novel approach for clustering evolving heterogeneous data streams, Chunyu Yang and Jie Zhou, ICDMW, 2006
- Sclope: an algorithm for clustering data streams of categorical attributes, Kok leong Ong, Wenyuan Li, Wee keong Ng, and Ee peng Lim, Technical report, 2004.
- Random sampling with a reservoir. J. Vitter. ACM Trans. Math. Softw., 11(1) :37–57, 1985.
- Sampling from a moving window over streaming data. B. Babcock, M. Datar, and R. Motwani, ACM