TDDD07 – Real-time Systems

Lecture 4: Distributed Systems

Simin Nadjm-Tehrani

Real-time Systems Laboratory

Department of Computer and information Science

LINKÖPING UNIVERSITY

Overview: Next three lectures

From one CPU to networked CPUs:

- First, from one CPU to multiple CPUs
  - Allocating VMs on multiple CPUs: Cloud
- Next, fully distributed systems
  - fundamental issues with timing and order of events
- Next, hard real-time communication
  - Guaranteed message delivery within a deadline, bandwidth as a resource
- Finally: QoS guarantees instead of timing guarantees, focus on soft RT

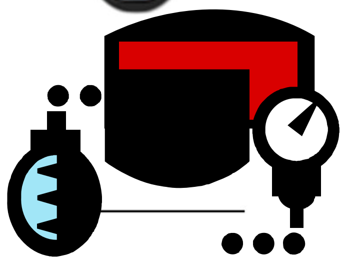
Reading material

- Note specially that from now on we have articles covering our topics
- These are posted for each lecture, linked from the course web page!
- For first part of this lecture, datacentre scheduling
  - Xiao et al. 2013

Overview

Distributed applications

- Banking and finance
- On-line access & electronic services
- Peer-to-Peer networks
- Distributed control
  - Cars, Airplanes, Smart factory
- Sensor networks
  - Buildings, Env. monitoring
- Mobile/Cloud computing



Common in all these?

Distributed model of computing:

- Multiple processes
- Disjoint address spaces
- Inter-process communication
- **Collective goal**

Reasons for distribution

- Locality
  - Engine control, brake system, gearbox control, airbag,…
  - Clients and servers
- Organisation of functions/code
  - An extension of modularisation, avoiding single points of failure
- Load sharing
  - Web services, search, parallelisation of heavy computations

Multicore scheduling: research topic - not part of the course!

This lecture

(1) Can we guarantee scheduling of tasks arriving from distributed nodes over a set of CPUs in the cloud?

(2) What are the fundamental time-related issues in distributed systems?

- Time, clocks, and ordering of events
- And why faults cannot be ignored…

Datacentre scheduling

Recall: Overloads with one CPU

- What happens if the task arrival rate is not predetermined?
- What if the load is not predictable?

Datacentre Scheduling

- Tasks are encapsulated in virtual machines (VM)

$\boxed{App_i}$

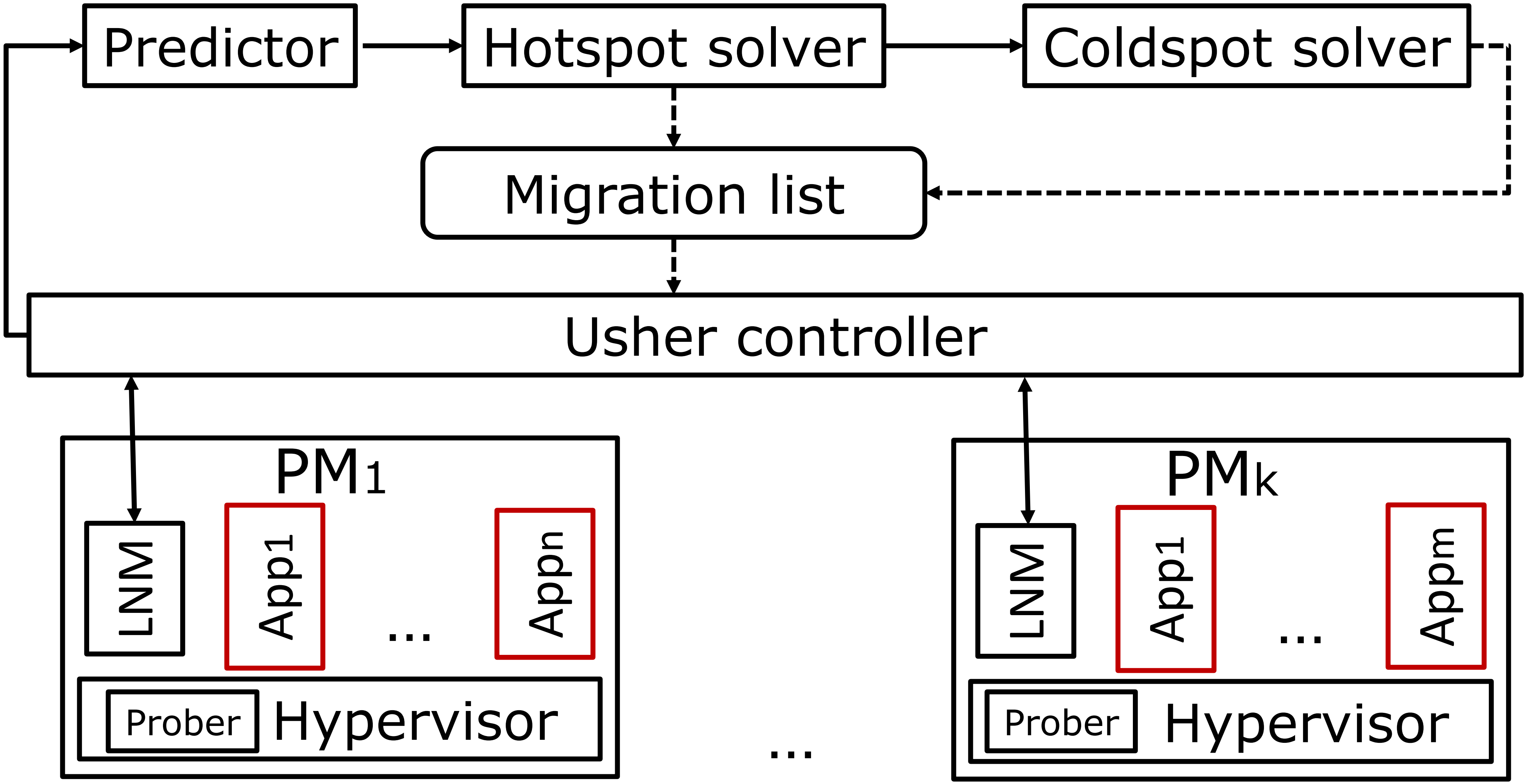
- Arrive from different nodes and their resource need varies over time
- Goals of scheduling: Allocate VMs to physical machines (PMs) such that
  - No PM is overloaded so that performance of tasks is not degraded
  - No PM is severely underloaded so that energy is not wasted
- As load changes, decide which VM to migrate!

[Xiao et al. 2013]

Used concepts

- Skewness: Notion that describes uneven utilisation of resources - minimise skewness over a set of PMs
- To deal with fluctuations it helps to predict the forthcoming load on each PM
- Identify the candidates for overload
  - Hotspot solver
- Identify any PM that runs at lower utilisation than an energy-efficient one
  - Coldspot solver

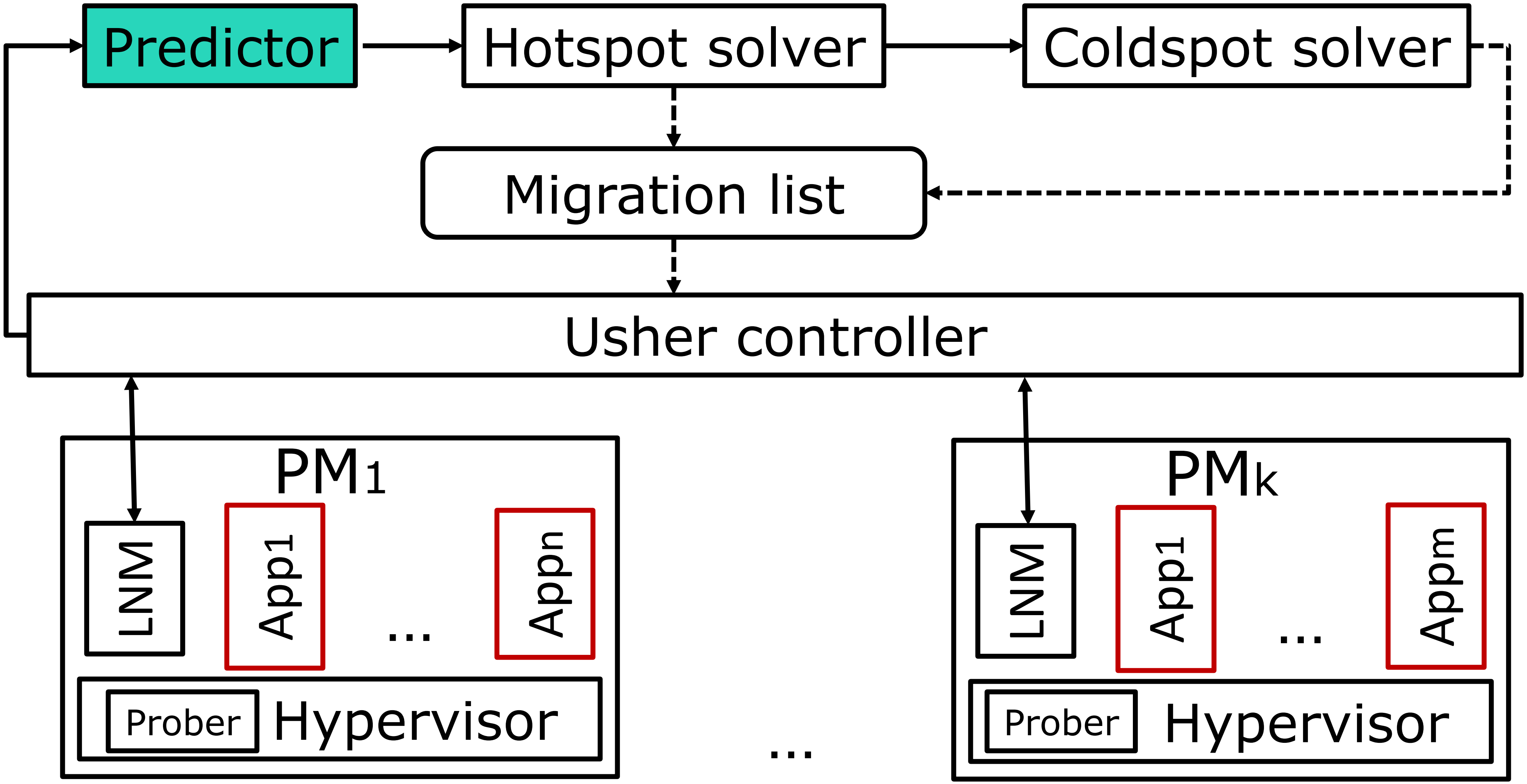
Adaptation architecture



Two mechanisms

- The hotspot solver detects if *any* PM's sum of resource usage is above the *hot threshold*
  - It will migrate away some VM from that PM
- The coldspot solver detects if the PMs are *on average* running at a utilisation below the *green computing (GC) threshold*
  - It will migrate away the VMs from some cold PM to prepare it for shut down

Adaptation architecture



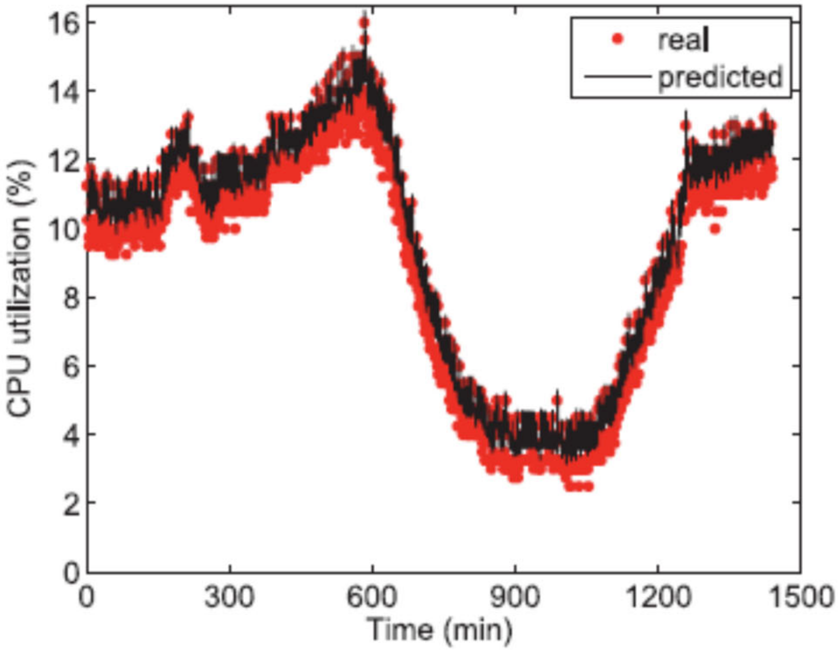
Estimating resource usage

- **Predictor:** Based on *observation O(t)* and previous *estimate E(t-1)* they suggest the Fast Up Slow Down (FUSD) estimation model:

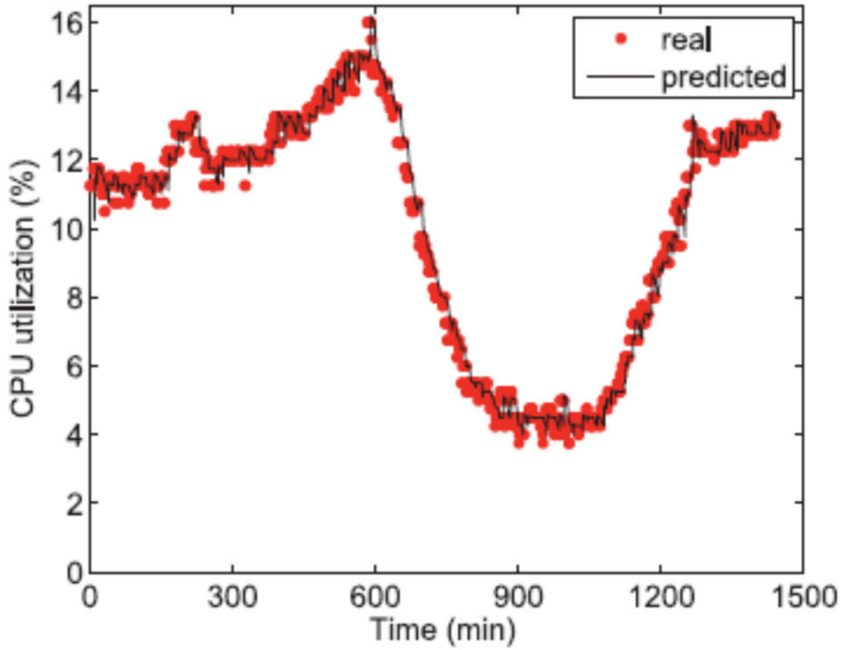
$$E(t) = \alpha \,.\, E(t-1) + (1 - \alpha).\, O(t)$$

where α is a parameter (experimentally) chosen as -0.2 when O(t) is rising and 0.7 when O(t) is falling

Window of observations



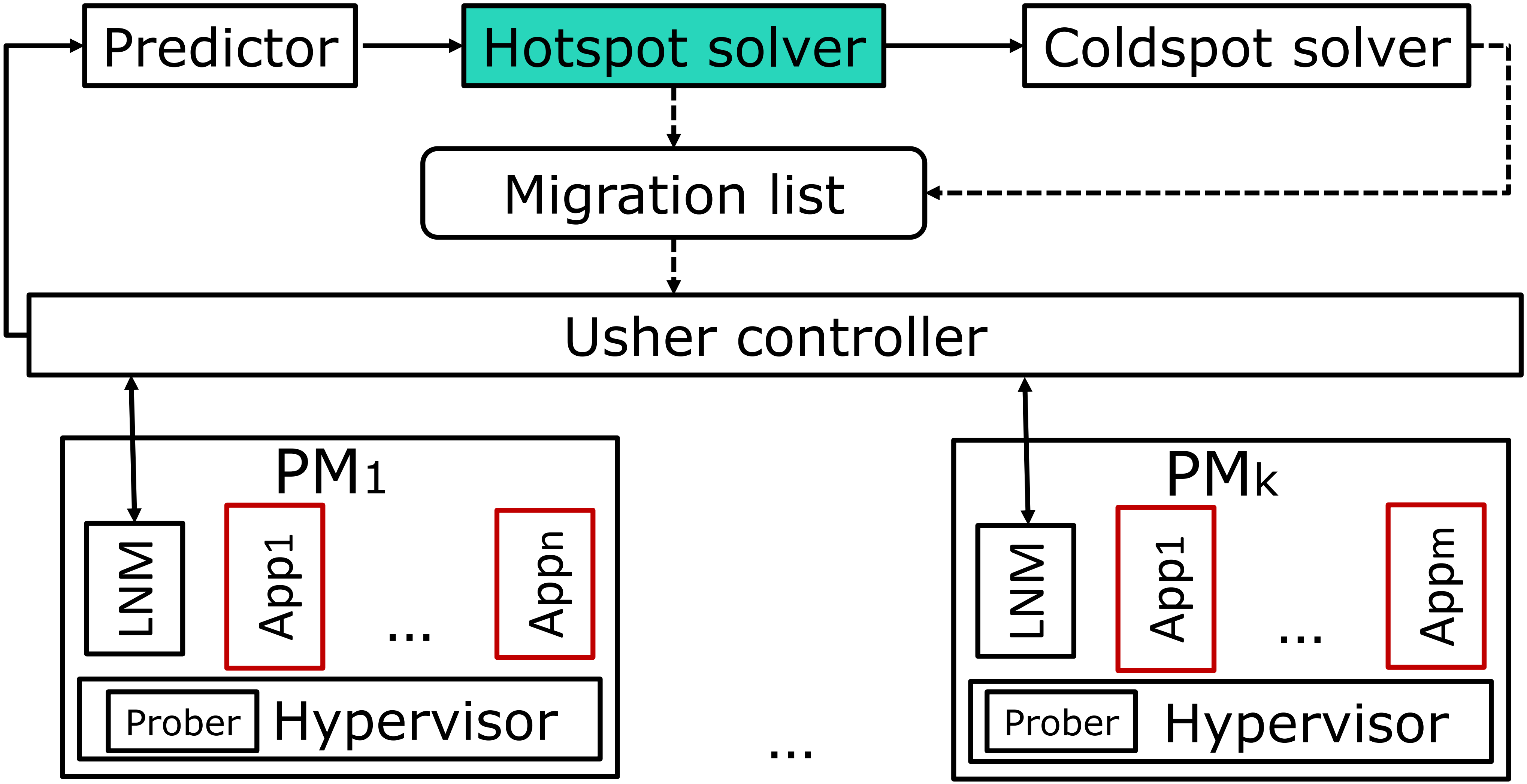
(b) FUSD: $\uparrow \alpha = -0.2, \downarrow \alpha = 0.7, W = 1$



(c) FUSD: $\uparrow \alpha = -0.2, \downarrow \alpha = 0.7, W = 8$

Left: uses only the current observation (W=1)
Right: uses the value below which 90% of the observed peak values in the past 8 observations fall

Adaptation architecture



Skewness and temperature

- The skewness for a server p (here a PM) as a function of the individual resources r_i used within it (here sum of VM utilisations for a resource r_i) and the average resource utilisation $\bar{r}$

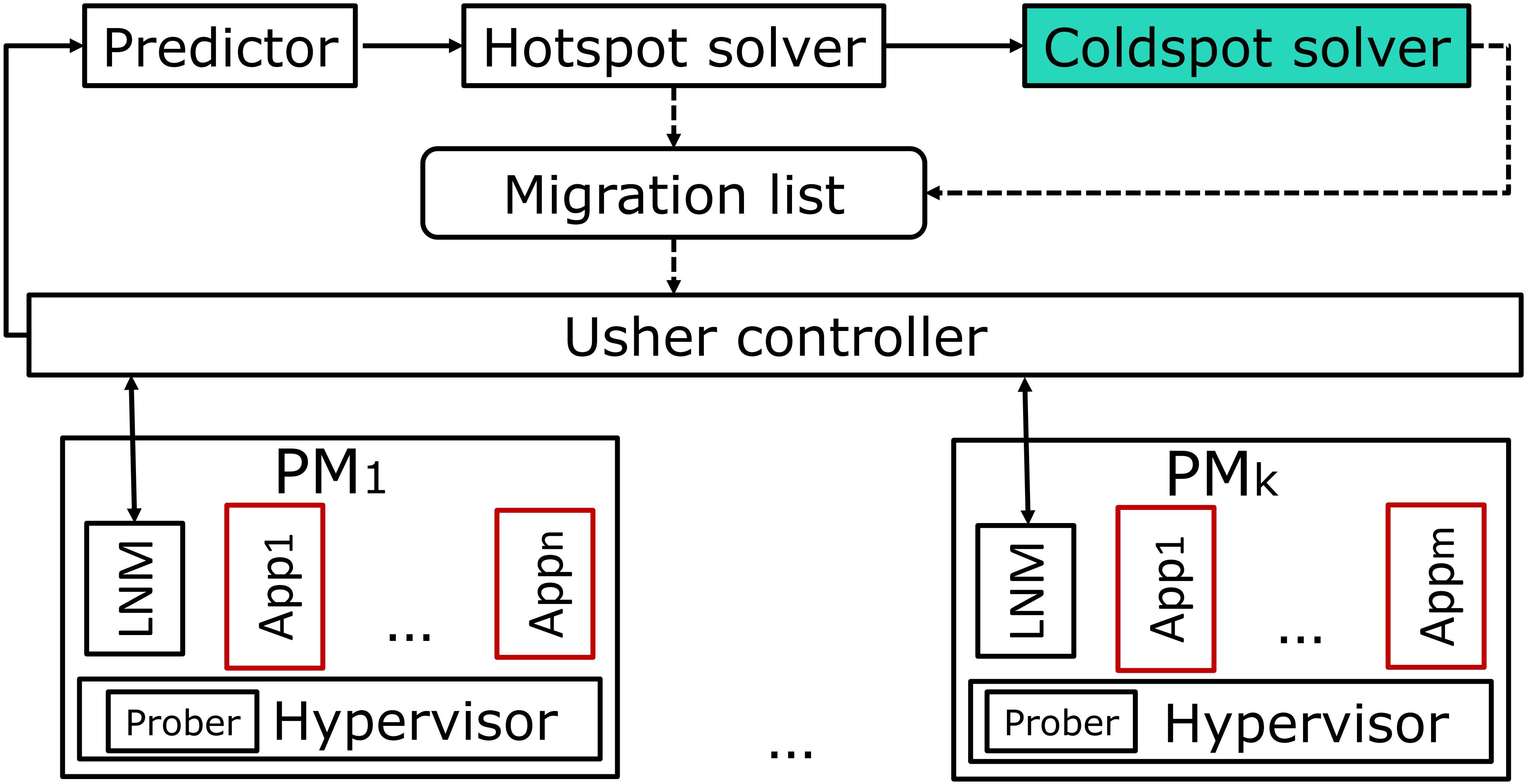
- A hotspot is a server that has high temperature

$$temperature(p) = \sum_{r \in R} (r - r_t)^2,$$

Load adaptation algorithm

- Order the PMs, choose a PM with highest temperature first
- Choose one VM on that PM that would reduce PM's temperature (if migrated away)
- If many such VMs, choose the one that increases skewness the least
- Find a PM that can accept this VM and not become a hot spot
- If many such PMs, choose the one that reduces its skewness most after accepting the VM
- If no such PM can be found proceed to the next VM on the hot PM (and eventually to next PM)

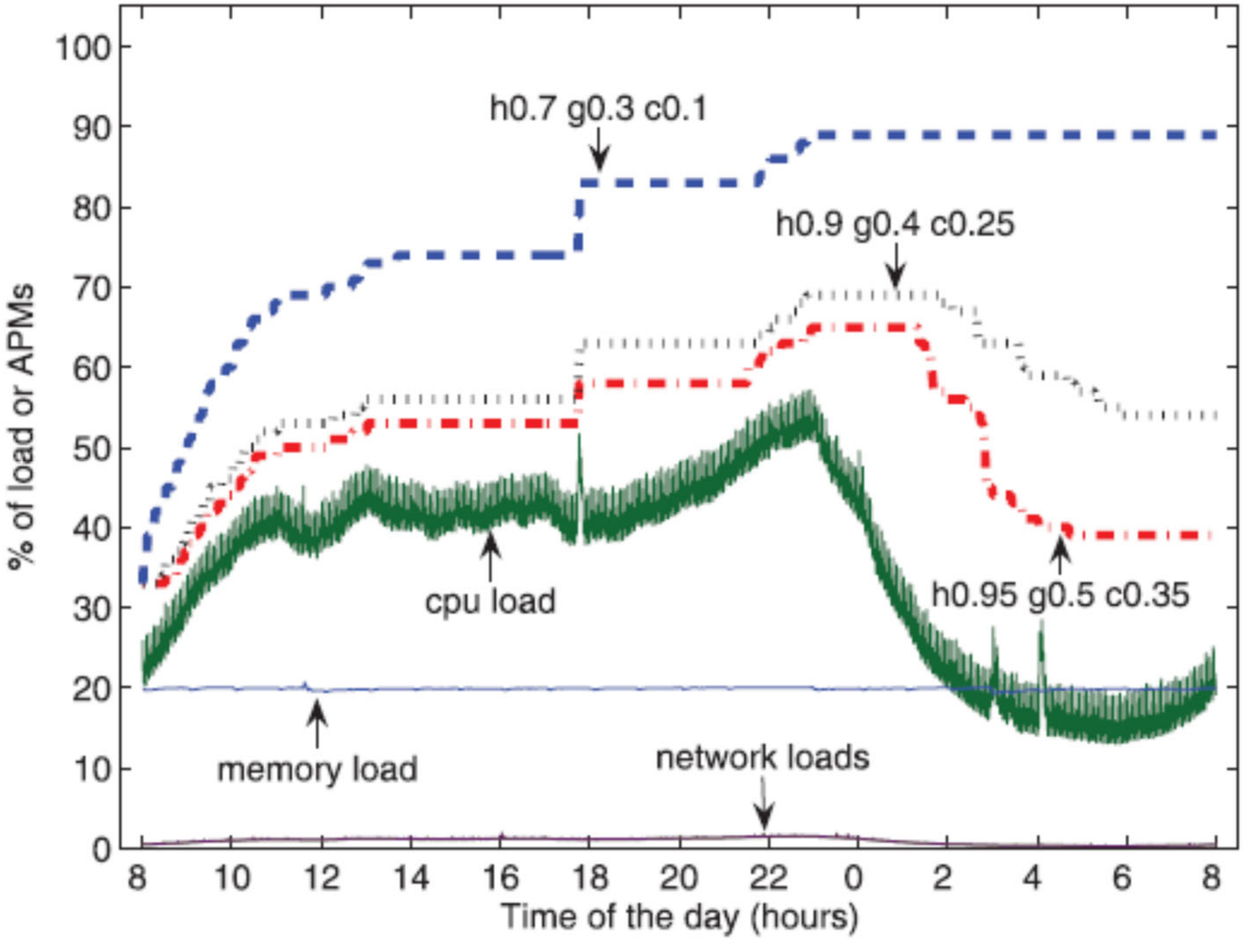
Adaptation architecture



Green computing adaptation

- The algorithm is invoked when average PM resource utilisation is below the GC threshold
- A PM is a cold spot if the utilisation of all of its different resources is below a given *cold threshold*
- Start with the PM that has the lowest utilisation for *memory* below the cold threshold
- Try to migrate all its VMs to another PM that will not become hot/cold after migration (will stay below a *warm* threshold) to avoid future hotspots

Does adaptation work?



Summary: Sharing the load

- A first attempt to study load adaptation vs. energy optimisation
- Clearly shows how scheduling is related to load control
- Note that there are no performance guarantees (live migration is expected to somewhat increase response time)

- Check the scalability arguments!

Time-related concepts in Distributed systems

LINKÖPING UNIVERSITY

Overview: Next three lectures

From one CPU to networked CPUs:

- First, from one CPU to multiple CPUs
  - Allocating VMs on multiple CPUs: Cloud
- Next, fully distributed systems
  - fundamental issues with timing and order of events
- Next, hard real-time communication
  - Guaranteed message delivery within a deadline, bandwidth as a resource
- Finally: QoS guarantees instead of timing guarantees, focus on soft RT

This lecture

(1) Can we guarantee scheduling of tasks arriving from distributed nodes over a set of CPUs in the cloud?

(2) What are the fundamental time-related issues in distributed systems?

– Time, clocks, and ordering of events
– And why faults cannot be ignored…

This lecture

(1) Can we guarantee scheduling of tasks arriving from distributed nodes over a set of CPUs in the cloud?

(2) What are the fundamental time-related issues in distributed systems?

- Time, clocks, and ordering of events
- And why faults cannot be ignored…

Reading Material

- Internal clock synchronisation
  – Slides are a summary of the article (up to section 2.2) in https://dl.acm.org/doi/pdf/10.1145/2455.2457

- Logical clocks
  Chapter 6 (specially sections 6.1 and 6.2) of the Attiya et al. book

Relevant questions

- Can we temporally order *all* events in a distributed system?
- Can we draw any conclusions if we do not have a global clock?
  - What about a set of local clocks?
  - What if no clocks at all?

Motivating examples

Timing of events

From Kopetz (1997):

- Consider a nuclear reactor equipped with many sensors that monitor different entities (e.g. Values of pressures, flows in various pipes). If a pipe ruptures, a number of entities will show values outside their normal operating ranges. When an entity enters its alarm region an alarm event is signalled to the operator.

Different views of *the same* system

The (global) system view:

- First, the pressure in the ruptured pipe changes abruptly
- Then the flow changes causing many other entities to react
- These in turn generate own alarms

The operator view:

- Operator sees a set of (correlated) alarms, called an “alarm shower”
- Operator wants to identify the primary event

Note on causality

- If event e occurs after event e’ then e cannot be the cause of e’
- If event e occurs before event e’ then e *can* be the cause of e’ (but need not be)
- Temporal order is necessary but not sufficient for causal order

Time vs. Events

Event detection

- The computer system must assist the operator to detect the primary event that triggers the alarm shower

- Knowledge of exact temporal order of the events is helpful in identifying the primary event

Other examples

- Detecting sequence of steps of an attack in a network forensics scenario

- Network time-stamps a major input

https://doi.org/10.1007/978-3-319-46298-1_16

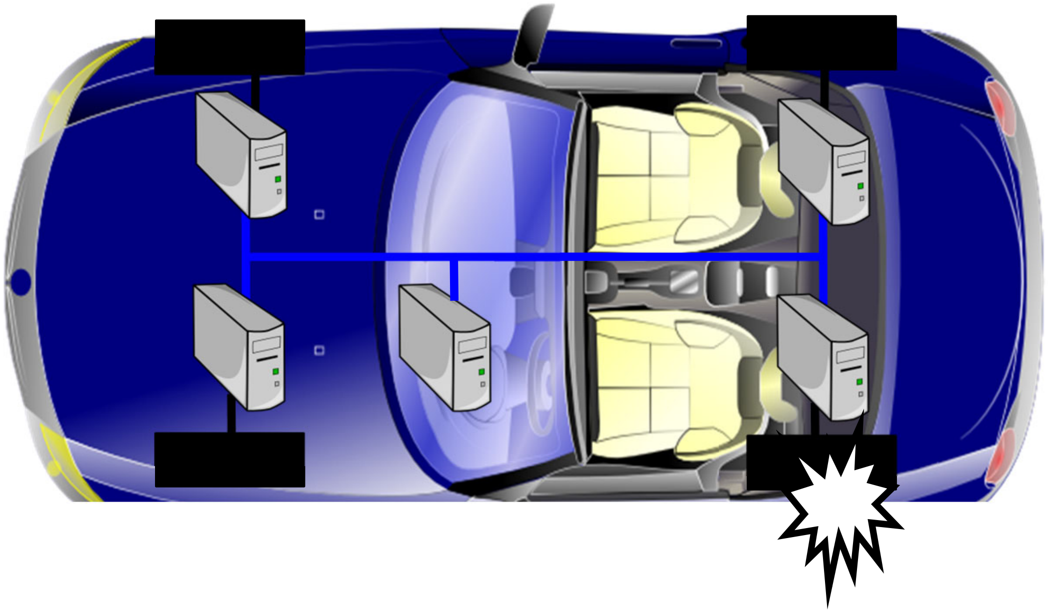
Security in industrial control

- Smart factories will become a reality
  - https://www.youtube.com/watch?v=CIAijpyN3_4
- It is not science fiction!
  - DOI: 10.1109/ICPHYS.2018.8390796
- Will we have security problems?
- Most ICS protocols have synchronous request-response patterns and detecting deviations helps to detect anomalies/malware
  - https://www.usenix.org/conference/raid2019/presentation/lin

Safety-critical systems

- Inaccurate local clocks can be a problem if the result of computations at different nodes depend on time
  - If the brake signal is issued separately in different wheels, will the car stop and when?

Brake-by-wire



We need to know when time is useful and when events will do

LINKÖPING UNIVERSITY

Relevant questions

- Can we temporally order *all* events in a distributed system?
  - Only if we can timestamp them with a value from a global (universal) clock

- Can we draw any conclusions if we do not have a global clock?
  - What about a set of local clocks?
  - What if no clocks at all?

Time in Distributed Systems

- Physical time vs. Logical time
- Example clock synchronisation algorithm
- Logical clocks
- Vector clocks

Local vs. global clock

- Most physical (local) clocks are not always accurate
- What is meant by accurate?
  - Agreement with UTC
  - Coordinated Universal Time (UTC) is in turn an *adjusted* time to account for the discrepancy between time measured based on rotation of earth, and the International Atomic Time (IAT)
- An atomic global clock accurately measures IAT
- If we rely on value of local clocks, they need to be *synchronised* regularly

GPS as a reliable source?

- January 2024: Manipulation of GPS signals have made reliance on current GPS questionable

  https://www.satmars.com/en/2024/01/24/massive-stoerung-des-gps-signals/

- September 2024: Flight safety needs to rely on alternative sources
  - https://ops.group/blog/gps-spoofing-final-report/

YOU CAN TELL WHEN SOMEONE'S BEEN A PROGRAMMER FOR A WHILE BECAUSE THEY DEVELOP A DEEP-SEATED FEAR OF TIME ZONE PROBLEMS.

Clock synchronisation

Clock synchronisation

Two types of algorithms:

- External synchronisation
  - Tries to keep the values of a set of clocks agree with an accurate clock, within a skew of δ
- Internal synchronisation
  - Tries to keep a set of clock values *close to each other* with a maximum skew of δ

Lamport/Melliar-Smith Algorithm

- Internal synchronisation of n clocks

- Each clock reads the value of all other clocks at regular intervals
  - If the value of some clock differs from value of own clock by more than δ, that clock value is replaced by own clock value
  - The average of all clocks is computed at each node
  - Own clock value is updated to the average value

Does it work?

- After each synchronisation interval the clocks get closer to each other

- If the skews are within δ, and the clocks are initially synchronised, then they are kept within δ from each other

- But what if clocks are faulty? What is considered a fault?

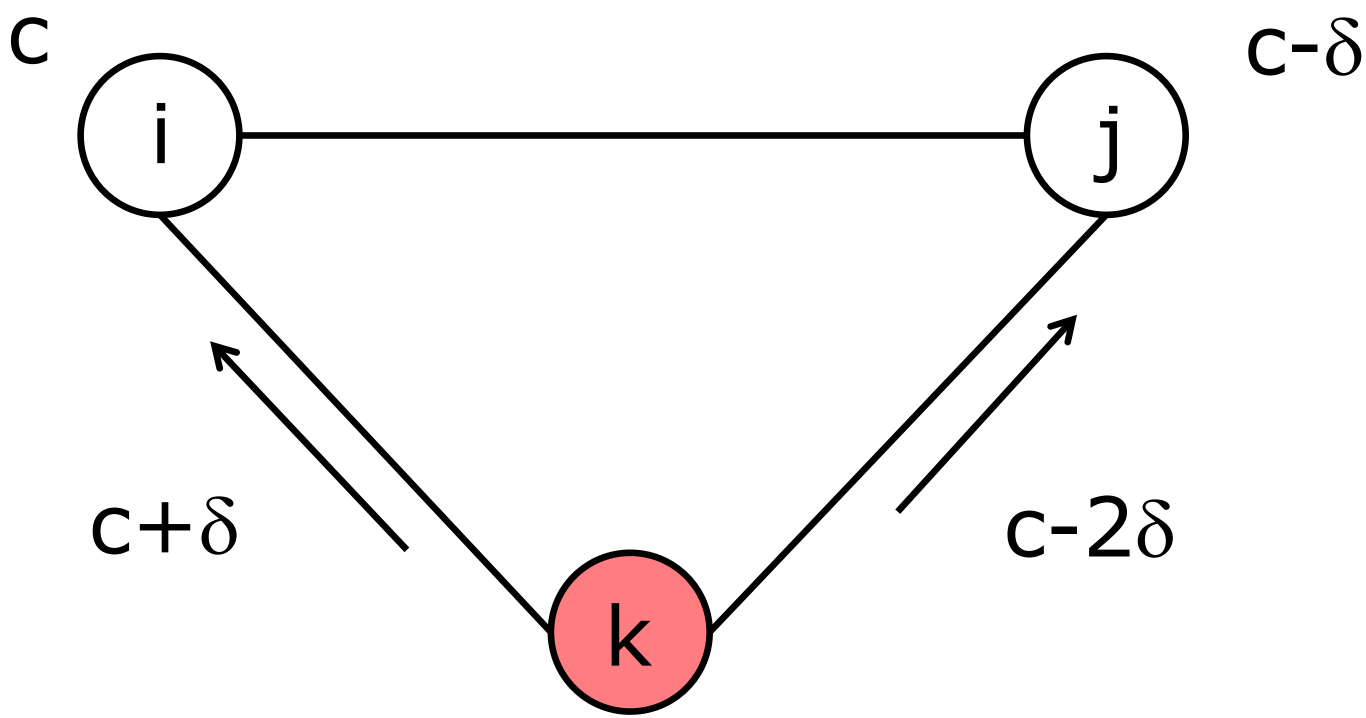
Faulty clocks

- If a clock skew exceeds δ then its value is eliminated – does not "harm" other clocks

- What if the skew is exactly δ?
  – check it as an exercise!

- What is the worst case?

A two-face faulty clock k



Will be considered as correct by i and j…

Bound on the faulty clocks

- To guarantee that the algorithm will keep all **non-faulty** clocks within δ we need an assumption on the number of faulty clocks

- For **f** faulty clocks the algorithm works if the number of clocks **n >3f**

Synchronisation example

"I also included a temperature compensated real time clock on Saboten to maintain an accurate alarm for periodic wake from sleep."

http://hackaday.com/2015/10/05/sensor-net-makes-life-easier-for-rice-farmers/

Ordering of events

Relevant questions

- Can we temporally order *all* events in a distributed system?
  - Only if we can timestamp them with a value from a global (universal) clock

- Can we draw any conclusions if we do not have a global clock?
  - What about a set of local clocks?
  - What if no clocks at all?

Time in Distributed Systems

- Physical time vs. Logical time

- Example clock synchronisation algorithm

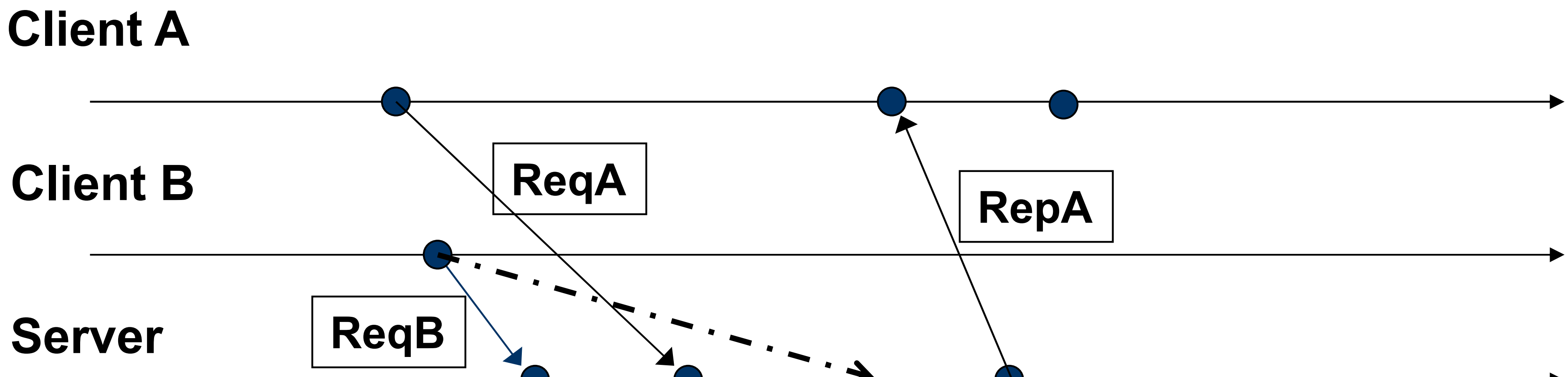
- Logical clocks

- Vector clocks

Are actually relating events...

Event ordering with no clocks

- In the absence of clock synchronisation, we may use order that is intrinsic in an application



Logical time

- Based on event counts at each node
- May reflect *causality*
- Sending a message always precedes receiving it
- Messages sent in a sequence by one node are (potentially) causally related to each other
  - I do not pay for an item if I do not first check the item's availability

Happened before $\prec$

- Assume each process has a monotonically increasing local clock
- Rule 1: if the time for event x is before the time for event y then $x \prec y$
- Rule 2: if x denotes sending a message and y denotes receiving the same message then $x \prec y$
- Rule 3: $\prec$ is transitive

A strict partial order...

Lamport’s Logical clocks

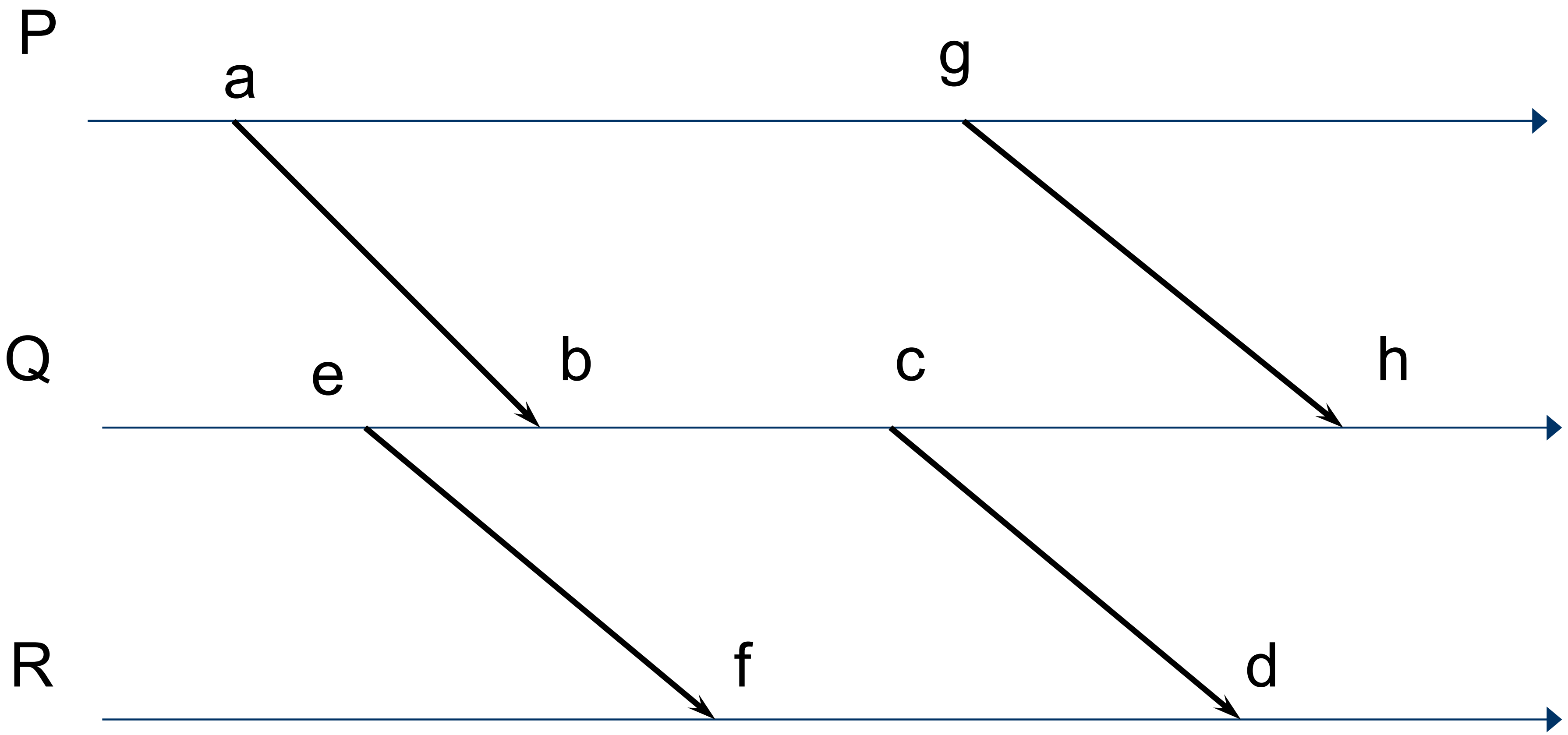
Seminal paper from 1978…

- Logical clock: An event counter that respects the “happened before” ordering

- Partial order: Hence, any events that are not in the “happened before” relation are treated as concurrent

Example (1)

What do we know here?



Implementing logical clocks

LC “time-stamps” each event

- Rule 1: Each time a local event takes place, increment LC by 1
- Rule 2: Each time a message m is sent the LC value at the sender is appended to the message (m_LC)
- Rule 3: Each time a message m is received set LC to max(LC, m_LC)+1

Exercise

- Calculate LC for all events in example (1)!

What does LC tell us?

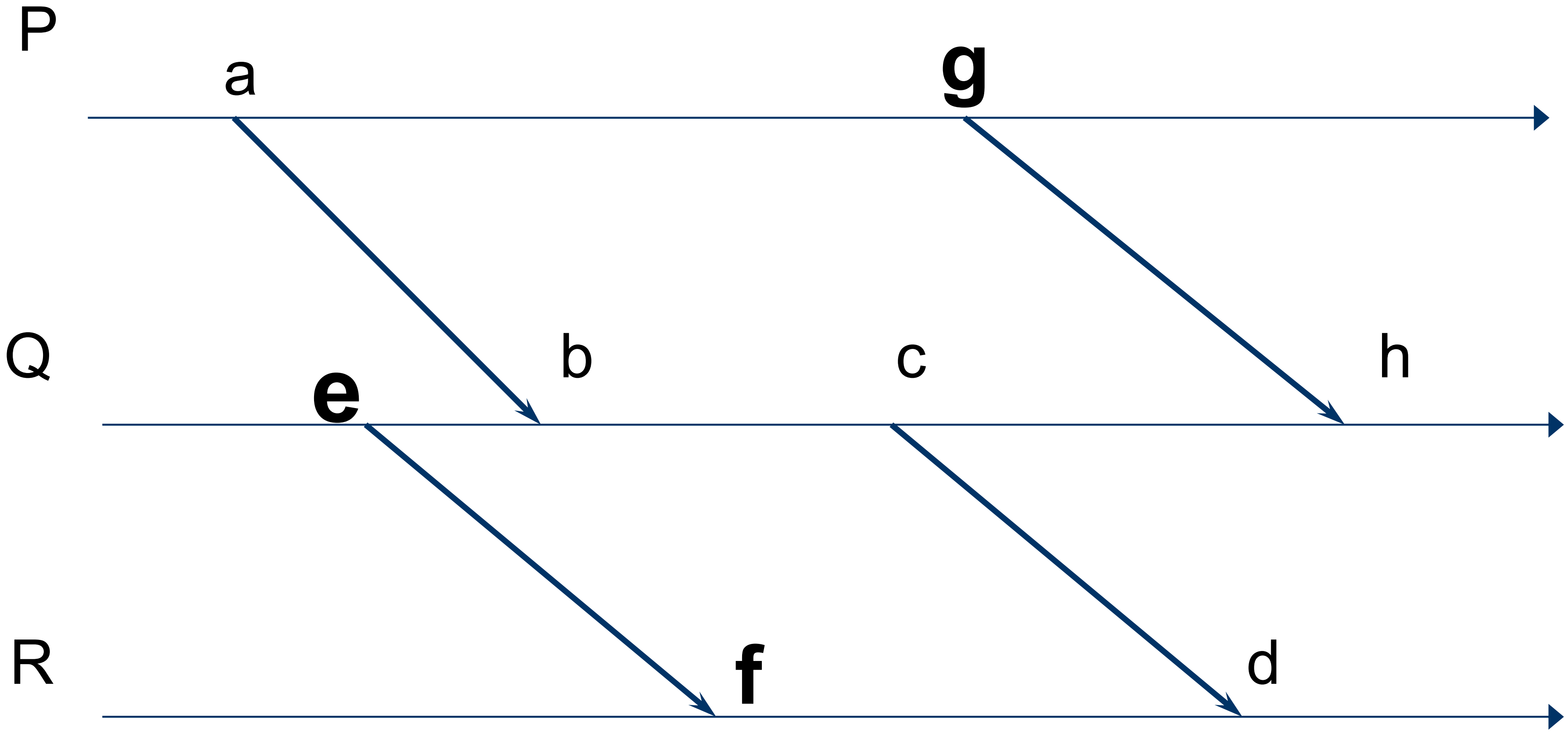
- $x \prec y \quad \rightarrow \quad LC(x) < LC(y)$

- Note that:

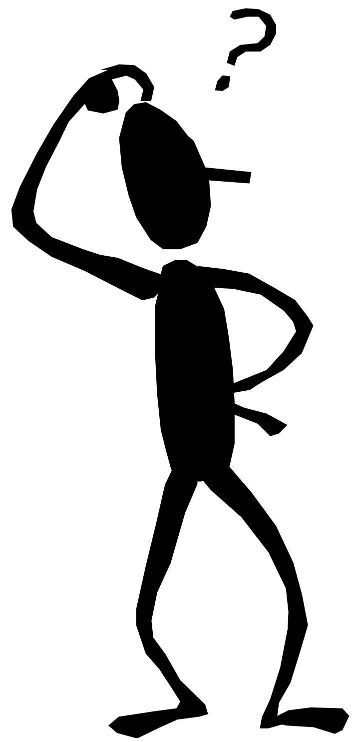
$LC(x) < LC(y)$ does not imply $x \prec y$

Example (1)

What did we capture by LC?



Example (1)



- e is concurrent with g
- g is concurrent with f
- but e is not concurrent with f!

- Comparing the LC values does not tell us if two events are concurrent in the sense of $\prec$
- Vector clocks do more…

Questions?

http://www.ida.liu.se/~TDDD07/

LINKÖPING
UNIVERSITY