Pub/Sub on Stream: A Multi-Core Based Message Broker with QoS support

Zhaoran Wang[1], Yu Zhang[2], Xiaotao Chang[2], Xiang Mi[1], Yu Wang[1], Kun Wang[2], Huazhong Yang[1]

1. Dept. of Electronic Engineering, TNList, Tsinghua University

2. IBM Research - China

Motivation

- Publish/Subscribe (Pub/Sub) is becoming an increasingly popular message delivery technique in the Internet of Things (IoT) era.
  - Smart Grid
  - Transportation
  - Battlefield Monitoring

- However, classical Publish/Subscribe is not suitable for some emerging IoT applications due to its lack of QoS capability.
  - Smart Grid: late messages ➔ severe power grid failure

- **Goal**: Enable QoS support in a Publish/Subscribe message broker with a multi-core processor.

Contents

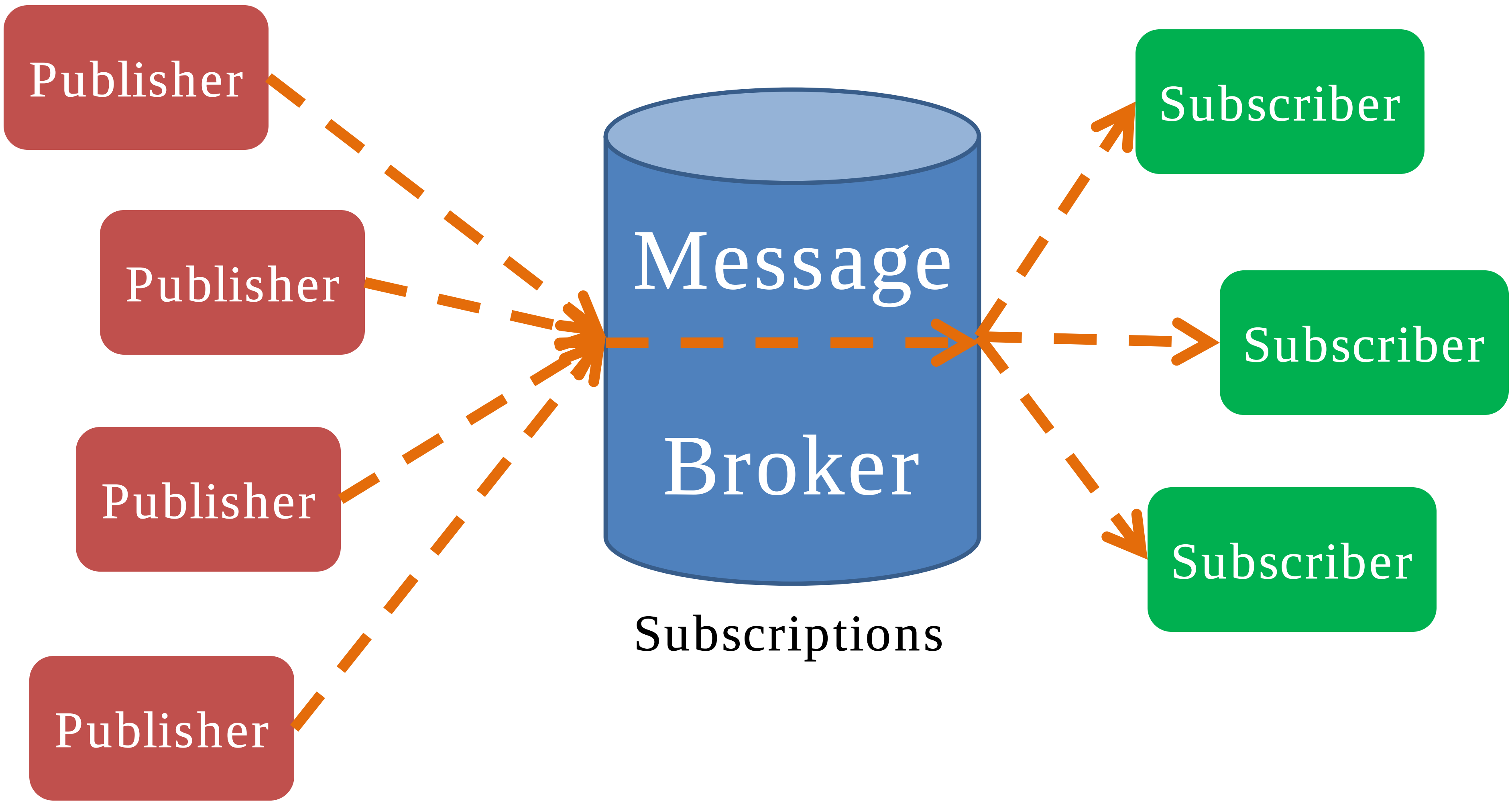
- Motivation
- QoS problems in Pub/Sub Systems for IoT applications
- Analysis and Partitioning of the Sequential Matching Algorithm
- Stream Matching Framework
  - Basic Framework
  - Deadline-aware Fine-Grained Scheduling(DFGS)
  - Smart Dispatch
- Evaluation

Contents

- Motivation and the Key Idea
- QoS problems in Pub/Sub Systems for IoT applications
- Analysis and Partitioning of the Sequential Matching Algorithm
- Stream Matching Framework
  - Basic Framework
  - Deadline-aware Fine-Grained Scheduling(DFGS)
  - Smart Dispatch
- Evaluation

QoS Problems in Pub/Sub Systems

- Publish/Subscribe (Pub/Sub) systems



QoS Problems in Pub/Sub Systems

- Applications such as the Smart Grid introduce various requirements on QoS.

[Dave Bakken, Proceeding of IEEE 2011]

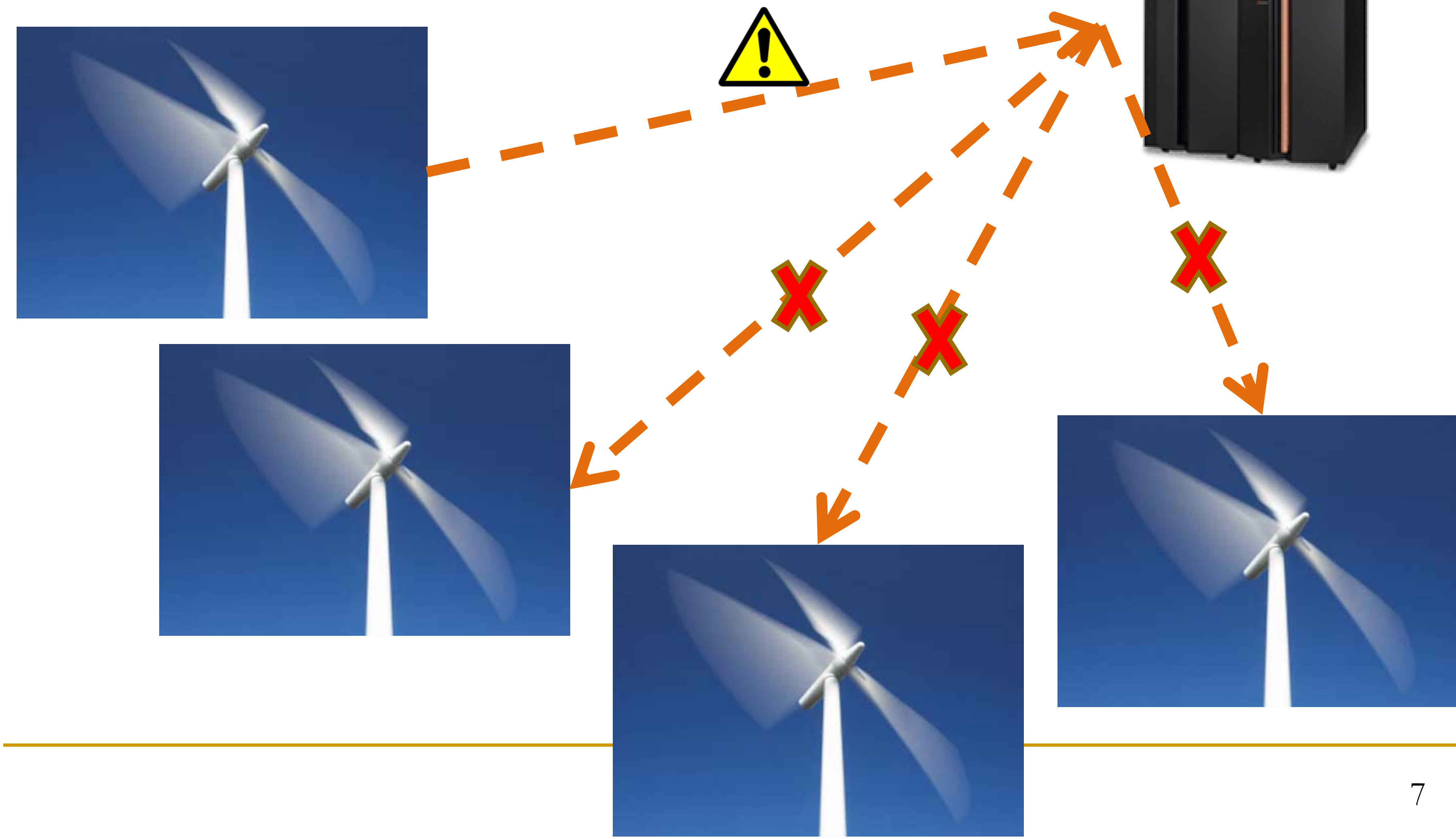
TABLE 1: NORMALIZED VALUES OF QOS+ PARAMETERS

Difficulty (5 hardest)	Latency (ms)	Rate (Hz)	Criticality	Quantity	Geography	Deadline (for bulk traffic)
5	5–20	240–720+	Ultra	Very High	Across grid or multiple ISOs	<5 seconds
4	20–50	120–240	High	High	Within an ISO/ RTO	1 minute
3	50–100	30–120	Medium	Medium	Between a few utilities	1 hour
2	100–1000	1–30	Low	Low	Within a single utility	1 day
1	>1000	<1	Very Low	Very Low (serial)	Within a substation	>1 day

- Tight latency requirements for critical messages.

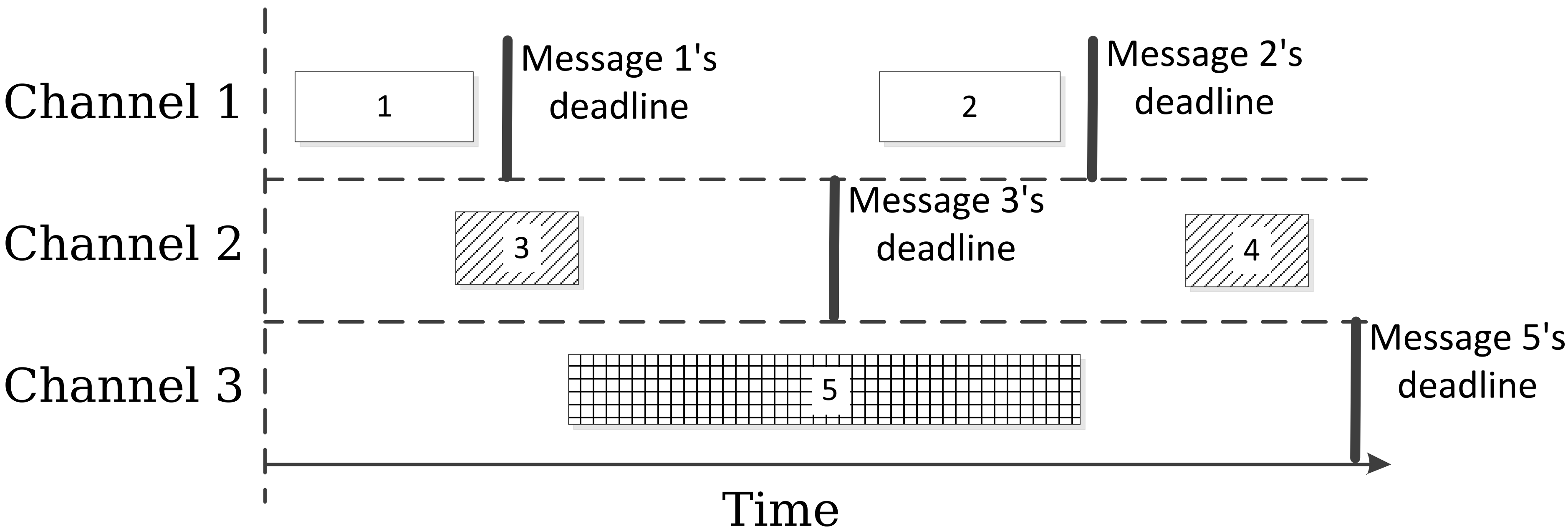
QoS Problems in Pub/Sub Systems

- Failure may lead to severe results.
  - Inefficient feedback control



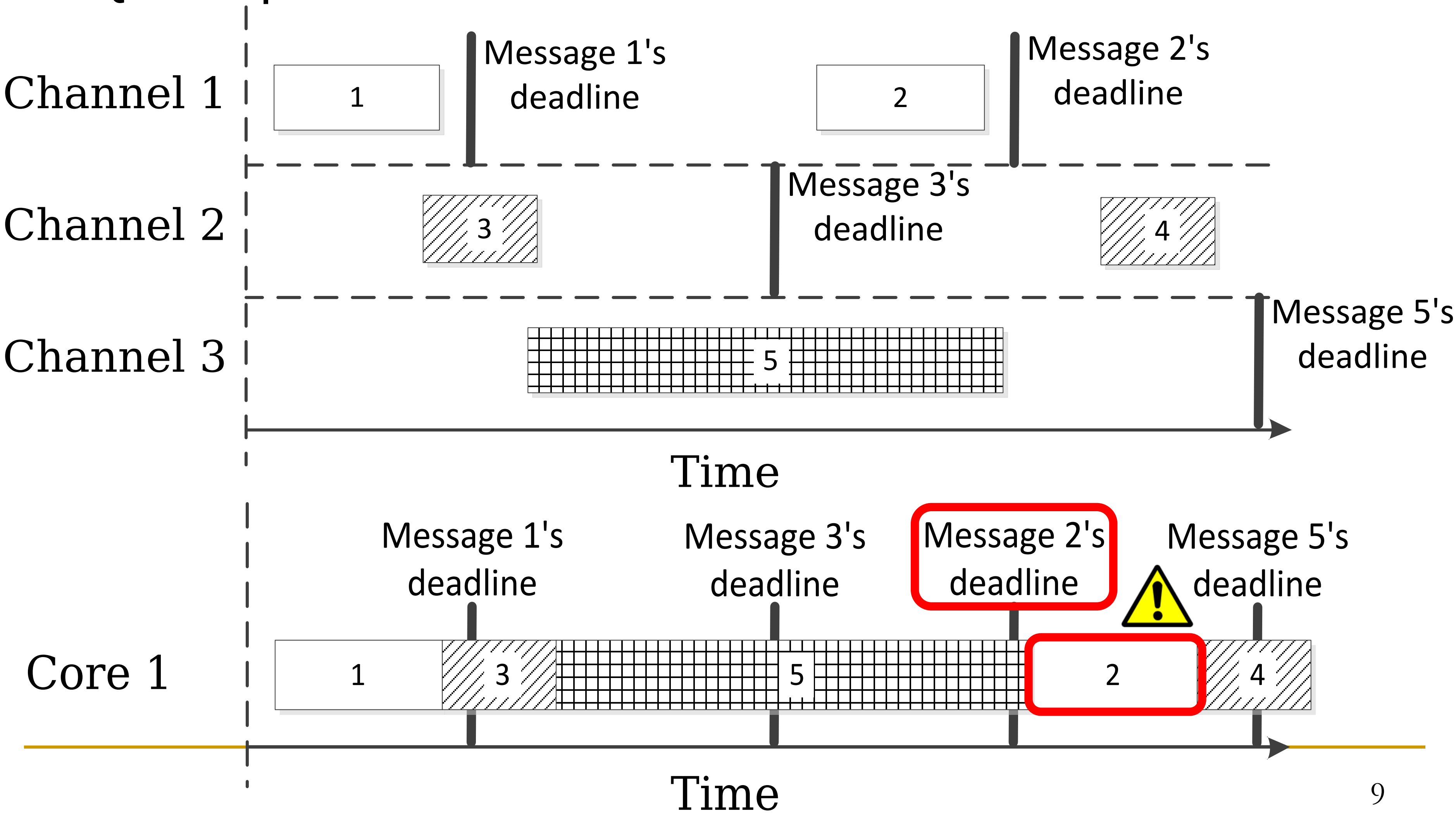
QoS Problems in Pub/Sub Systems

- Messages with the same QoS requirements➔Channel
- A message's QoS requirement, especially latency, is violated➔Failure



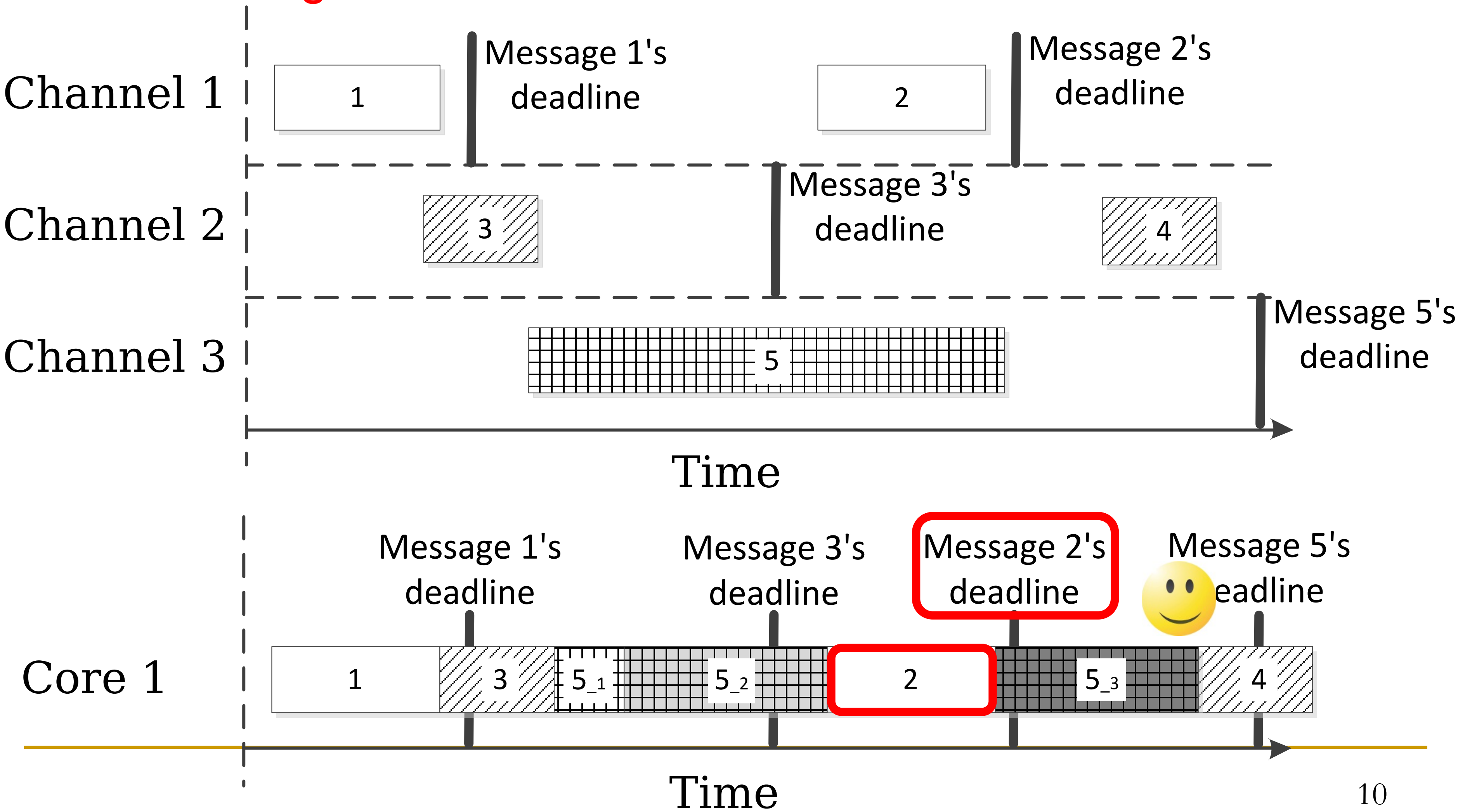
QoS Problems in Pub/Sub Systems

- Inappropriate allocation of resources will lead to violation of QoS requirements.



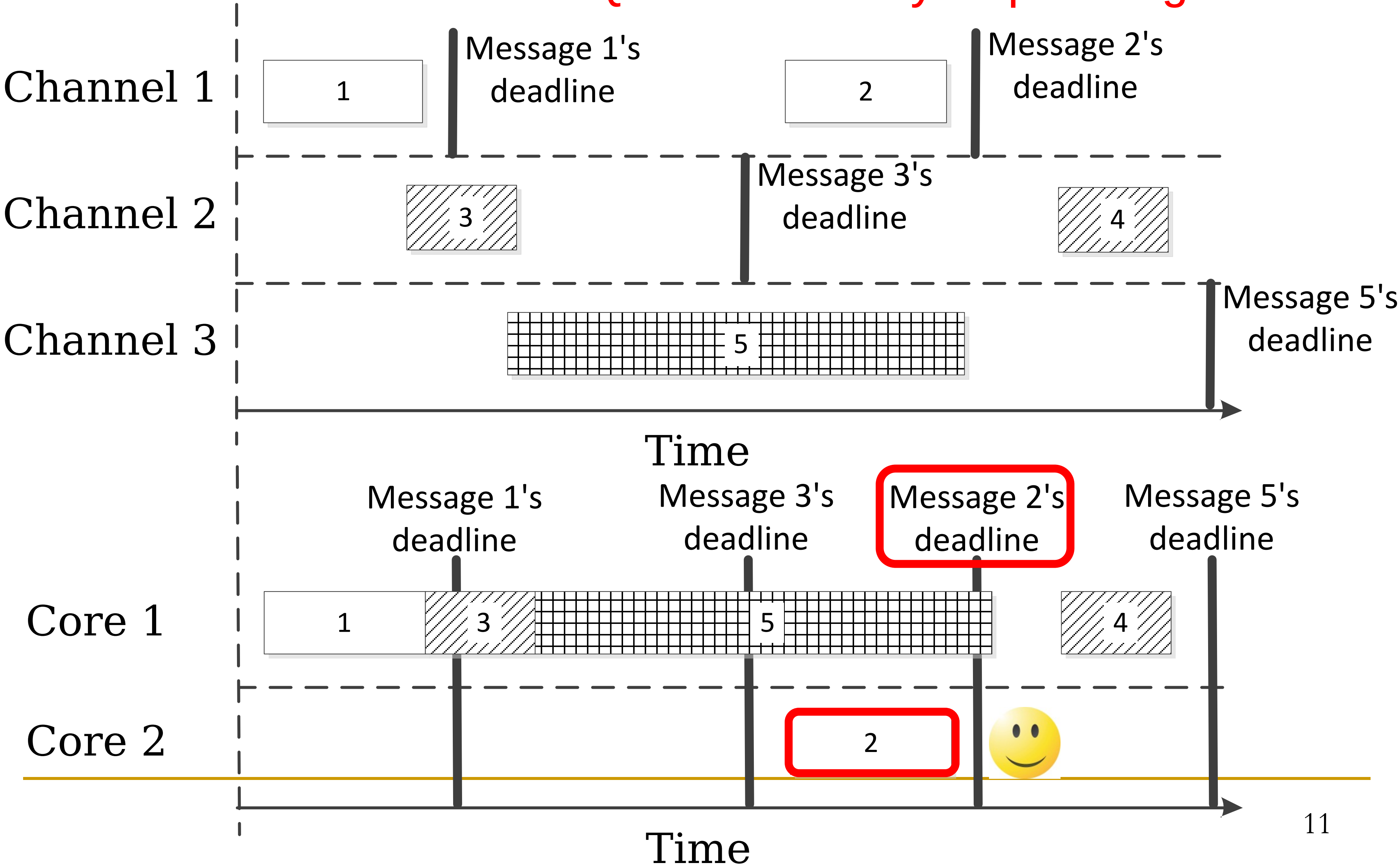
QoS Problems in Pub/Sub Systems

- Solution 1: Resolve the QoS violation by splitting and scheduling.



QoS Problems in Pub/Sub Systems

- Solution 2: Resolve the QoS violation by dispatching.



Key Idea

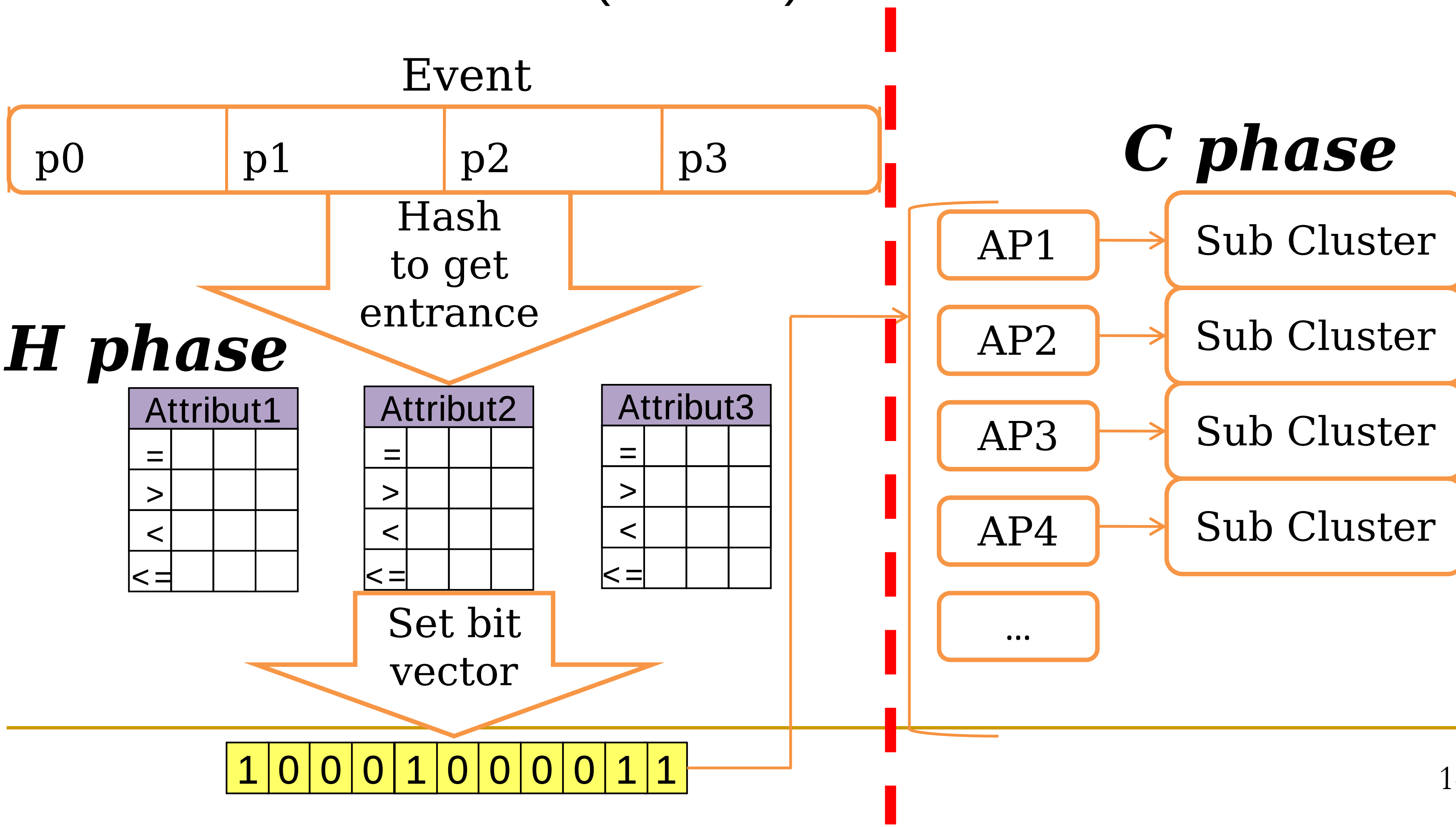
- **Key idea**: actively schedule computation resources to guarantee QoS requirements of messages.
  - Abstract the message matching algorithm into a task-based framework.
  - Two-level task scheduling mechanism to allocate the computation resources
    - Intra-core fine-grained task scheduling
      - Deadline-aware Fine-grained Scheduling (DFGS)
    - Inter-core message dispatching
      - Smart Dispatch

Contents

- Motivation and the Key Idea
- QoS problems in Pub/Sub Systems for IoT applications
- Analysis and Partitioning of the Sequential Matching Algorithm
- Stream Matching Framework
  - Basic Framework
  - Deadline-aware Fine-Grained Scheduling(DFGS)
  - Smart Dispatch
- Evaluation

Sequential Matching Algorithm

- Two-phase algorithm by [F. Fabret, SIGMOD 2001]
  - Hash Phase (H Phase)
  - Check Cluster Phase (C Phase)



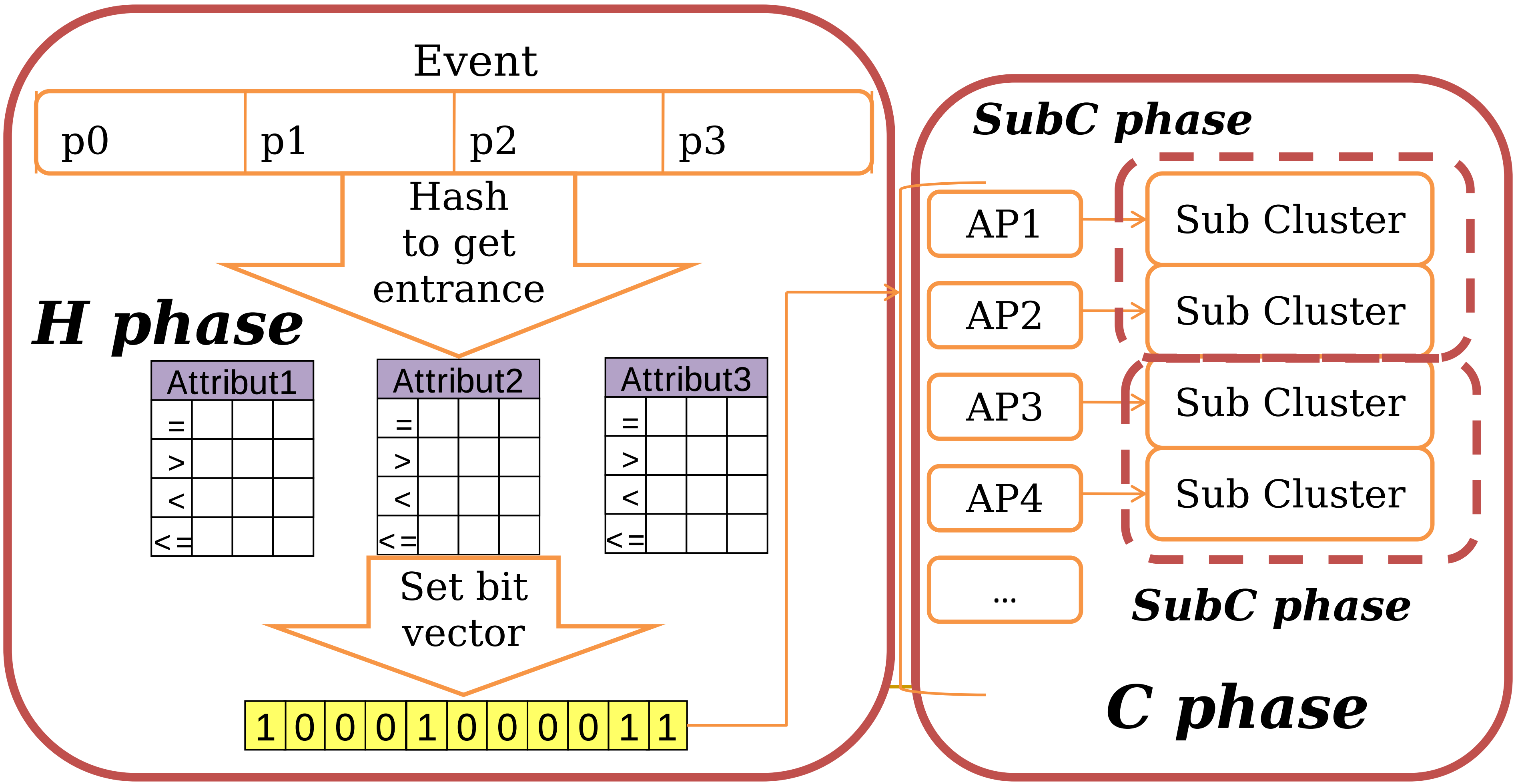
Sequential Matching Algorithm

- Two-phase algorithm by [F. Fabret, SIGMOD 2001]
  - Hash Phase (H Phase)
  - Check Cluster Phase (C Phase)

Number of Subscriptions	Processing time/ms	H Phase	C Phase
6000	0.034	65%	35%
60,000	0.15	14%	86%
600,000	0.81	3%	97%

Sequential Matching Algorithm

- Two-phase algorithm by [F. Fabret, SIGMOD 2001]
  - Hash Phase (H Phase)
  - Check Cluster Phase (C Phase)

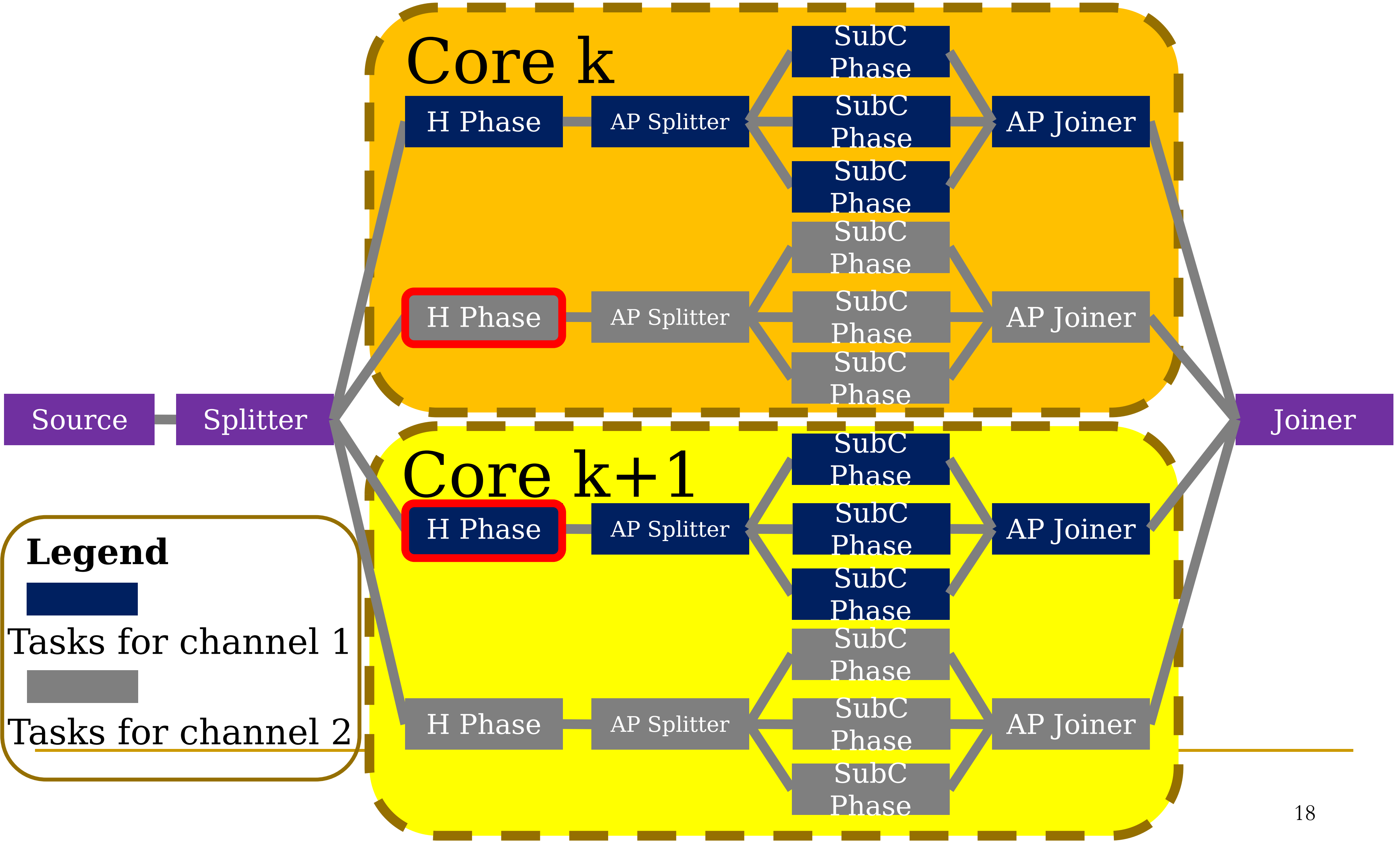


Contents

- Motivation and the Key Idea
- QoS problems in Pub/Sub Systems for IoT applications
- Analysis and Partitioning of the Sequential Matching Algorithm
- Stream Matching Framework
  - Basic Framework
  - Deadline-aware Fine-Grained Scheduling(DFGS)
  - Smart Dispatch
- Evaluation

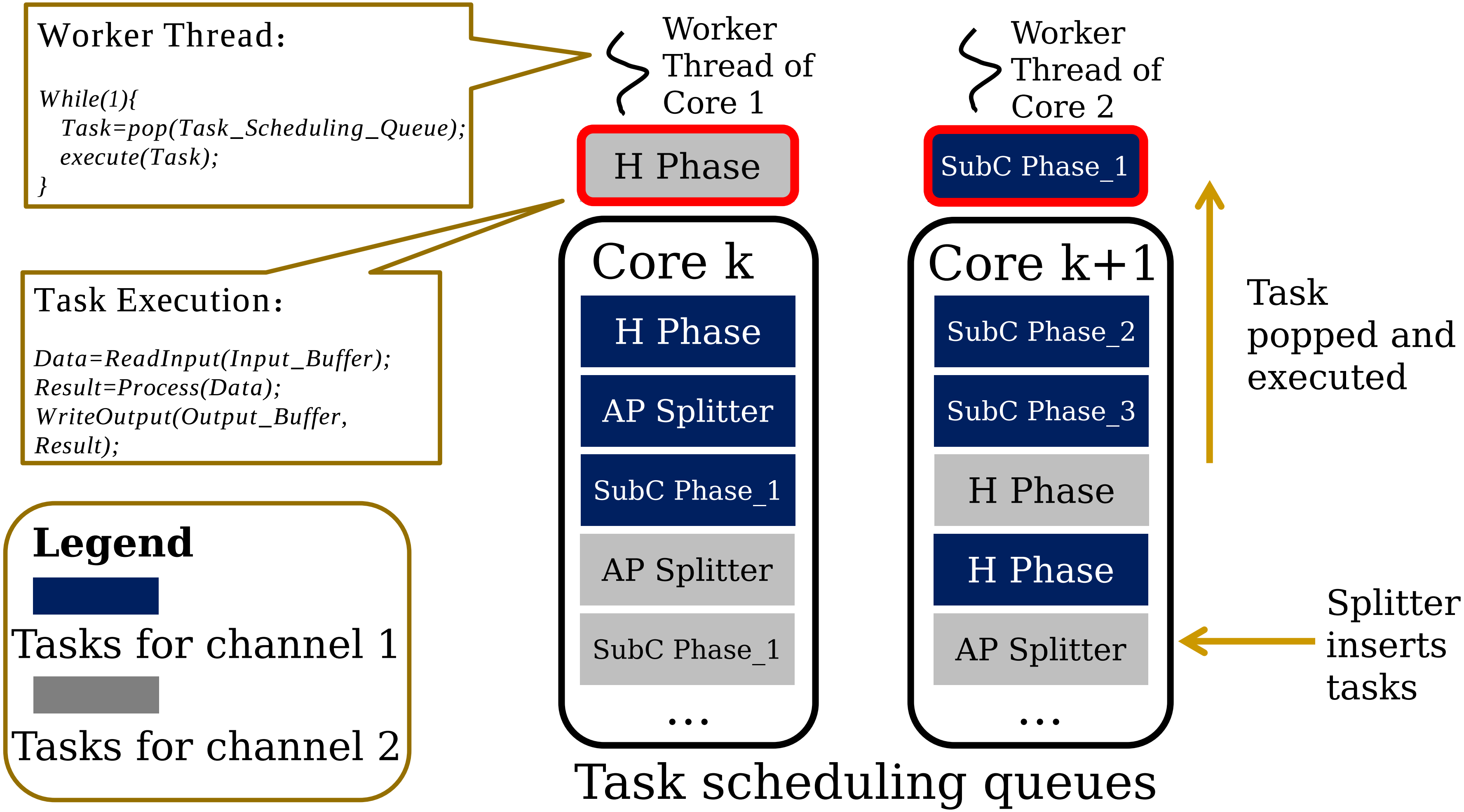
Stream Matching Framework

- The architecture of the task-based framework



Stream Matching Framework

- How tasks are executed on each core.



Contents

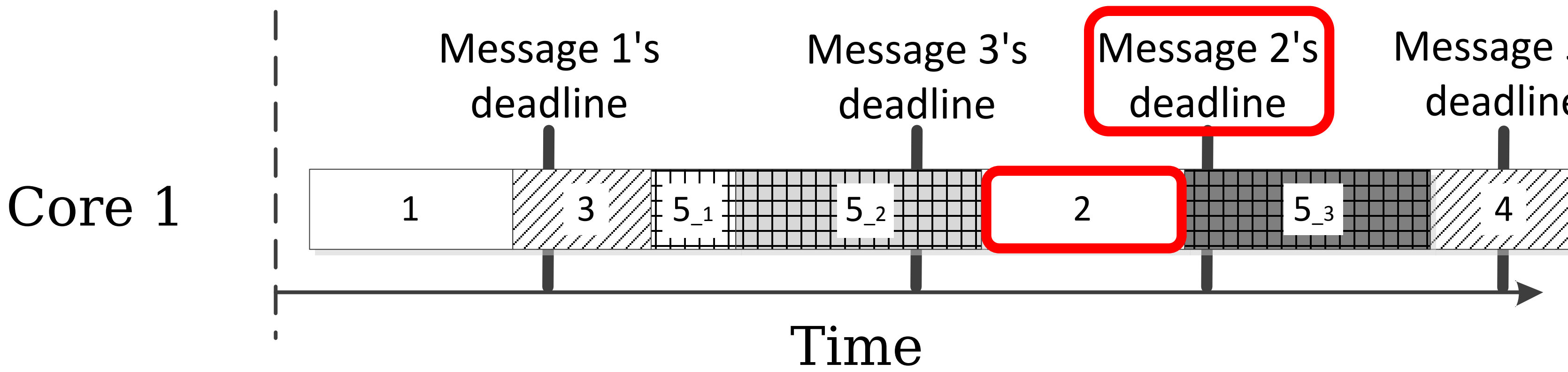
- Motivation and the Key Idea
- QoS problems in Pub/Sub Systems for IoT applications
- Analysis and Partitioning of the Sequential Matching Algorithm
- Stream Matching Framework
  - Basic Framework
  - Deadline-aware Fine-Grained Scheduling(DFGS)
  - Smart Dispatch
- Evaluation

Stream Matching Framework

- Deadline-aware Fine-Grained Scheduling(DFGS)
  - Splitter decides where in the task scheduling queue the tasks for a message should be inserted.
  - 3 criterions for inserting a task T:
    - T won’t cause any task after it to fail
    - Intra-channel order is preserved
    - T's deadline won’t to be violated
  - Linearly traverse the task scheduling queue to find the position satisfying the 3 criterions
    - Return false if there isn’t any.
    - Some optimizations in the paper.

Stream Matching Framework

- Pros
  - The tasks are NOT executed in the order the message comes.
  - Urgent message get processed first.
  - Urgent message can interrupt the processing of normal messages.

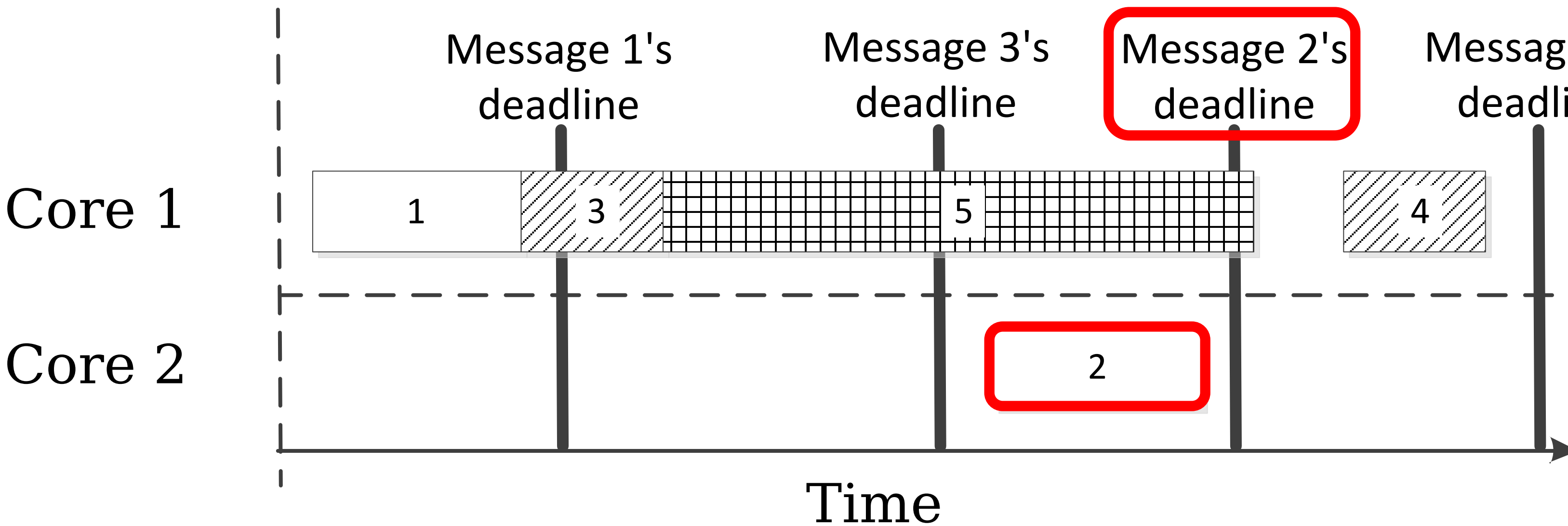


Stream Matching Framework

- Smart Dispatch
  - Splitter polls to decide which way to dispatch a message
    - Check resources on each core using DFGS
  - If All cores returns false, message is deserted and counted as a failure

Stream Matching Framework

- Pros
  - Explores the parallelism of multi-core machines by resource checking globally



Stream Matching Framework

- Cons of DFGS and Smart Dispatch
  - Scheduling overhead
    - Linear complexity for each core
    - The number of cores is limited

Contents

- Motivation and the Key Idea
- QoS problems in Pub/Sub Systems for IoT applications
- Analysis and Partitioning of the Sequential Matching Algorithm
- Stream Matching Framework
  - Basic Framework
  - Deadline-aware Fine-Grained Scheduling(DFGS)
  - Smart Dispatch
- Evaluation

Evaluation

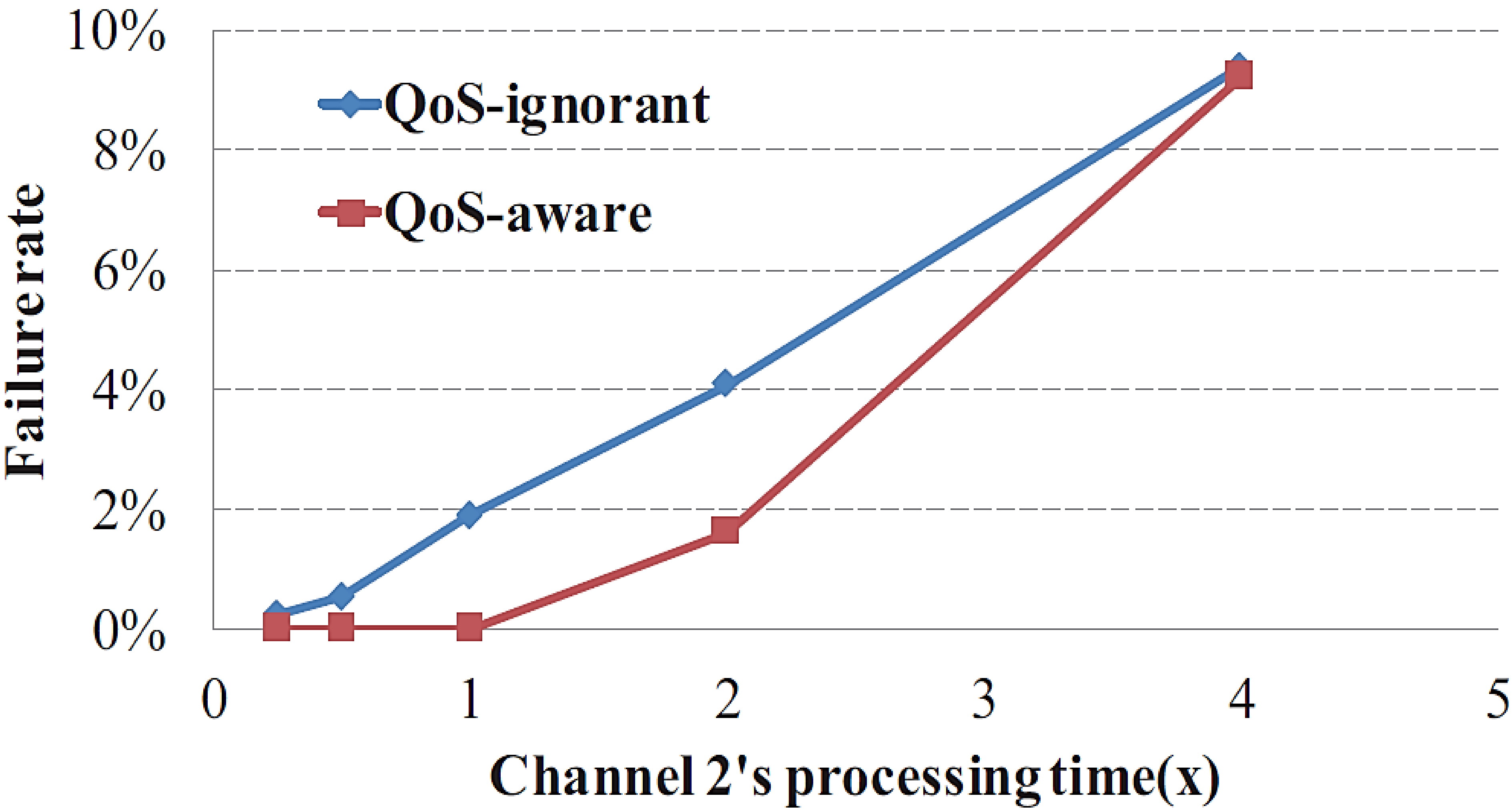
- Settings
  - Intel Xeon CPU E5507 at 2.27GHz
  - Fedora with kernel version 2.6.31.5
  - GCC 4.1.1 using -O3
  - Dataset
    - 1504 predicates; 238 attributes; Channel 1 with 60,000 subscriptions; Channel 2 with 600,000 subscriptions as in [F. Fabret, SIGMOD 2001]
    - Set the message processing deadline according to [Dave Bakken, Proceeding of IEEE 2011]

Evaluation

- How Parameters affects the failure rate?
  - QoS-ignorant system as the baseline
  - 2 channel for the convenience of discussion
  - Failure rate:
    - QoS-aware system, the message fails to be dispatched because there are no resources
    - QoS-aware or QoS-ignorant systems, the message's deadline violated
    - The percentage of total failed messages in all messages
- Scalability on multi-core machine?

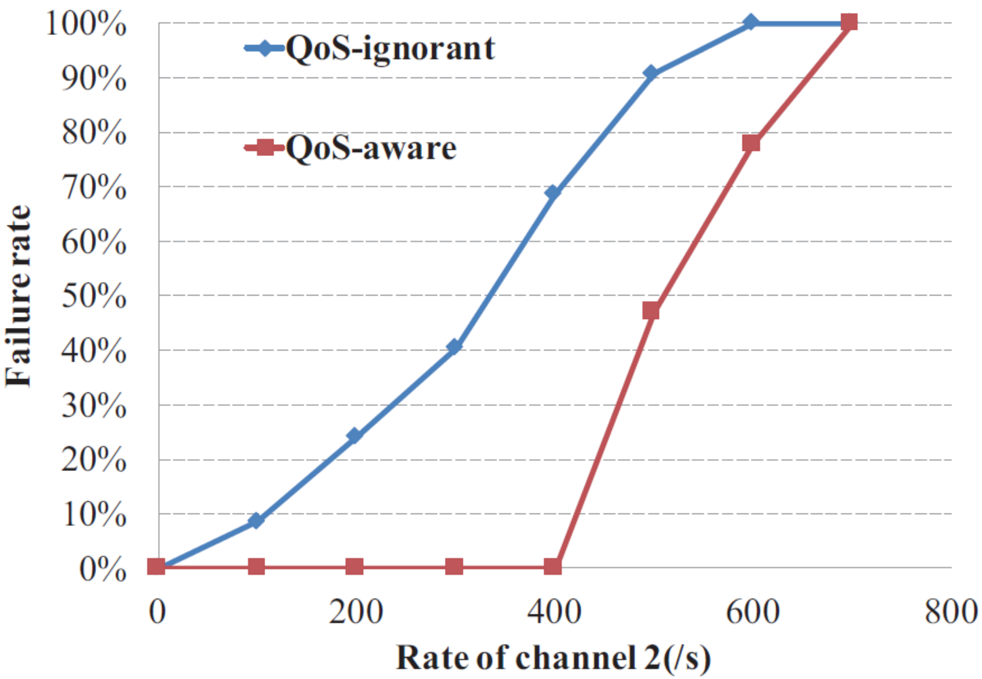
Evaluation

- The processing time' effects on the failure rate

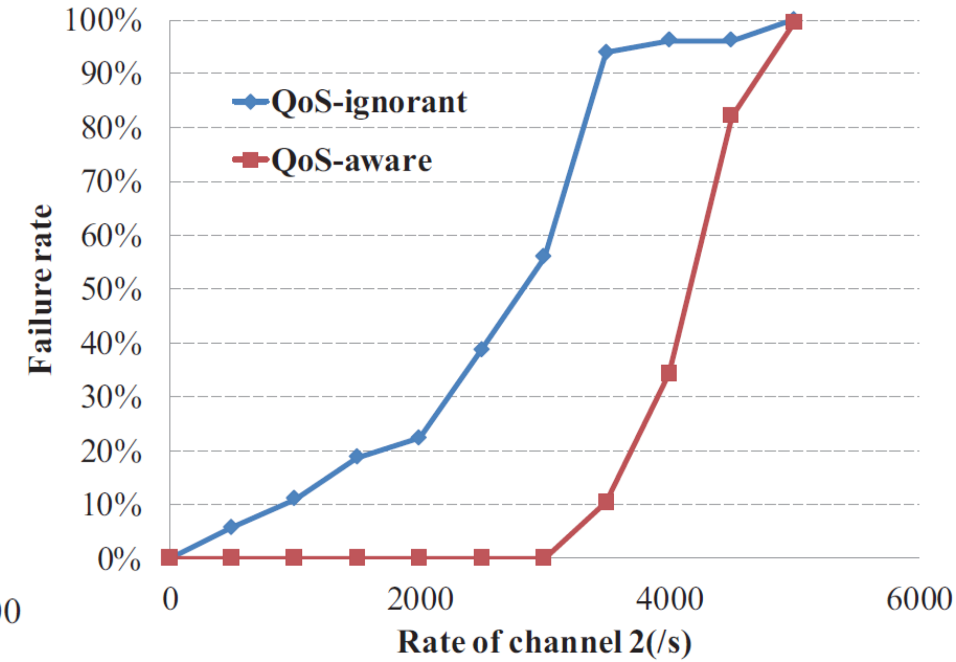


Evaluation

- The event's arriving period's effects on the failure rate



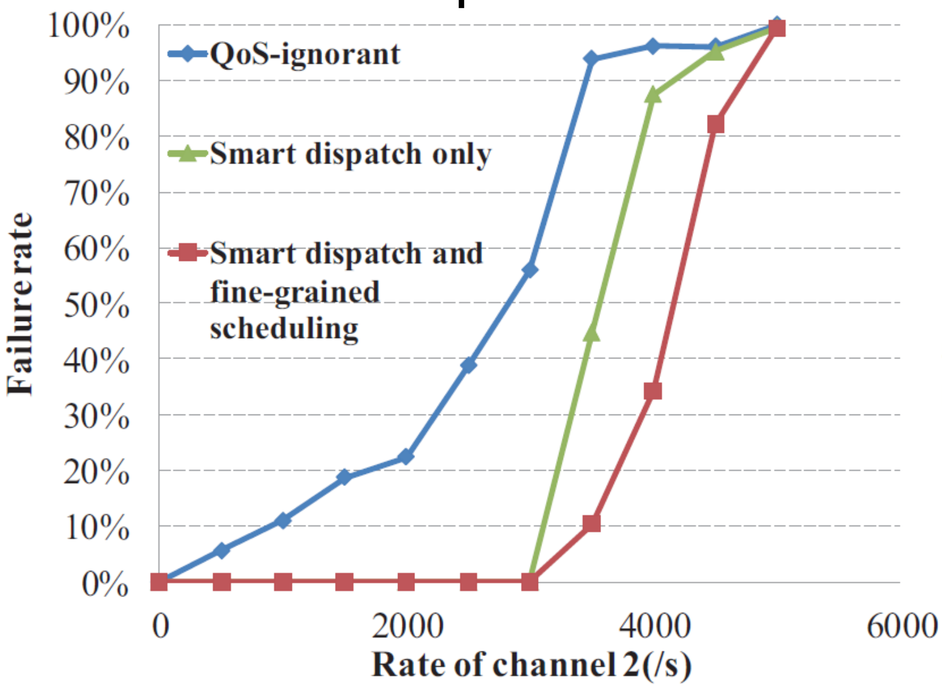
With 1 Core



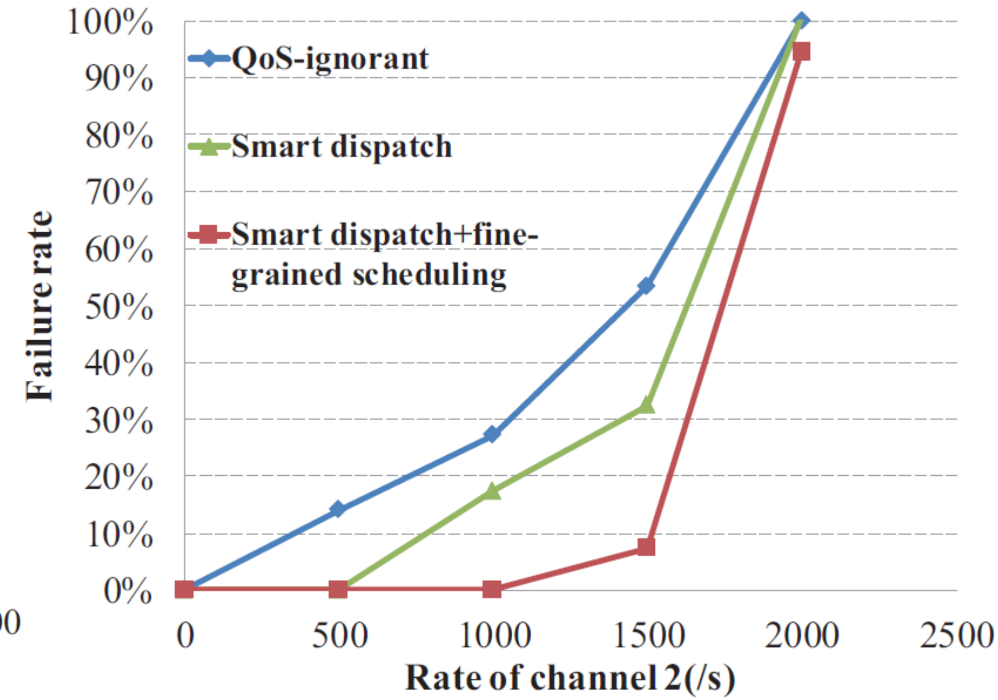
With 7 Core

Evaluation

- The event's arriving period's effects on the failure rate
  - Smart Dispatch or DFGS?



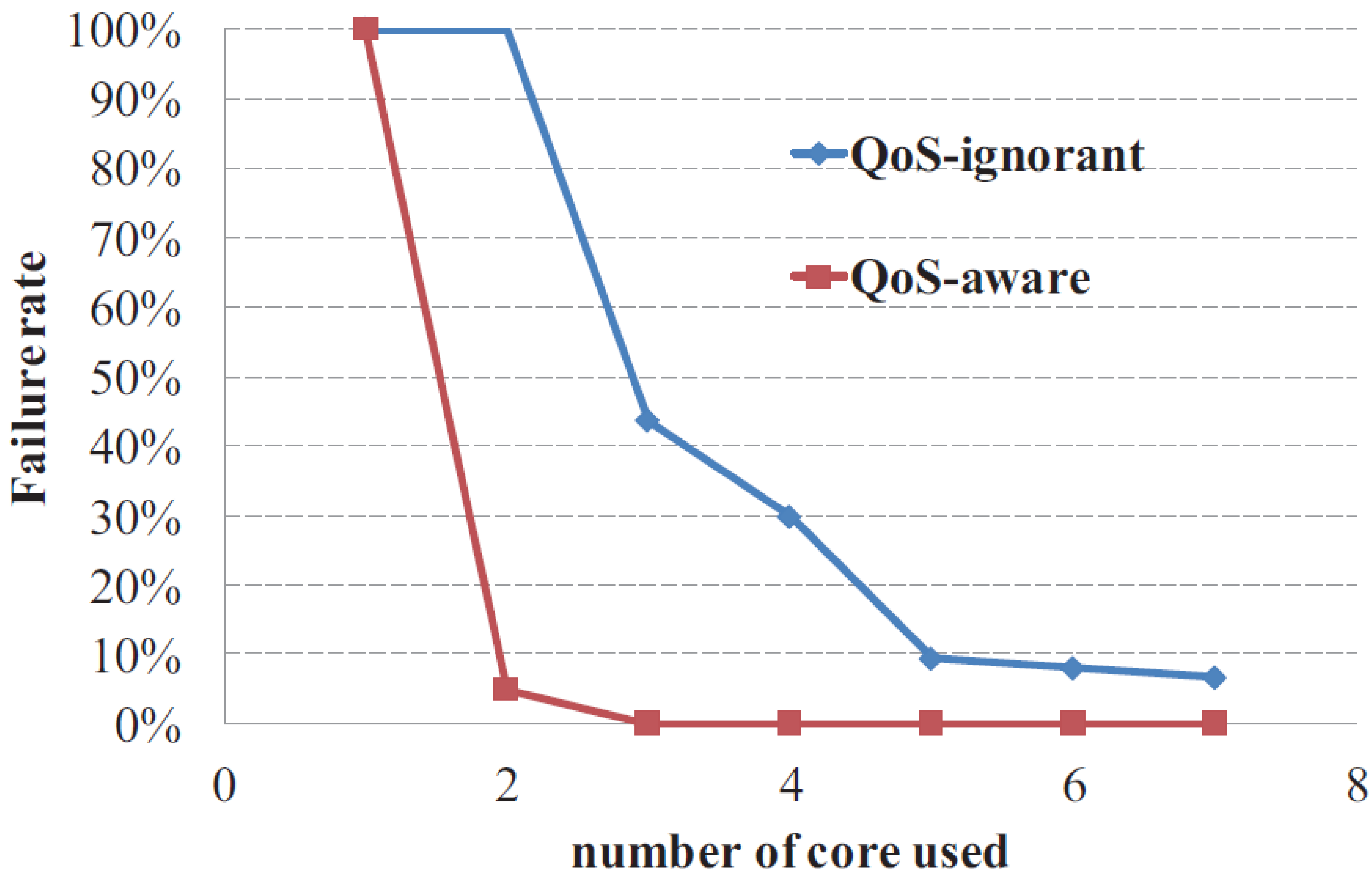
Channel 2 has shorter processing time



Channel 2 has longer processing time

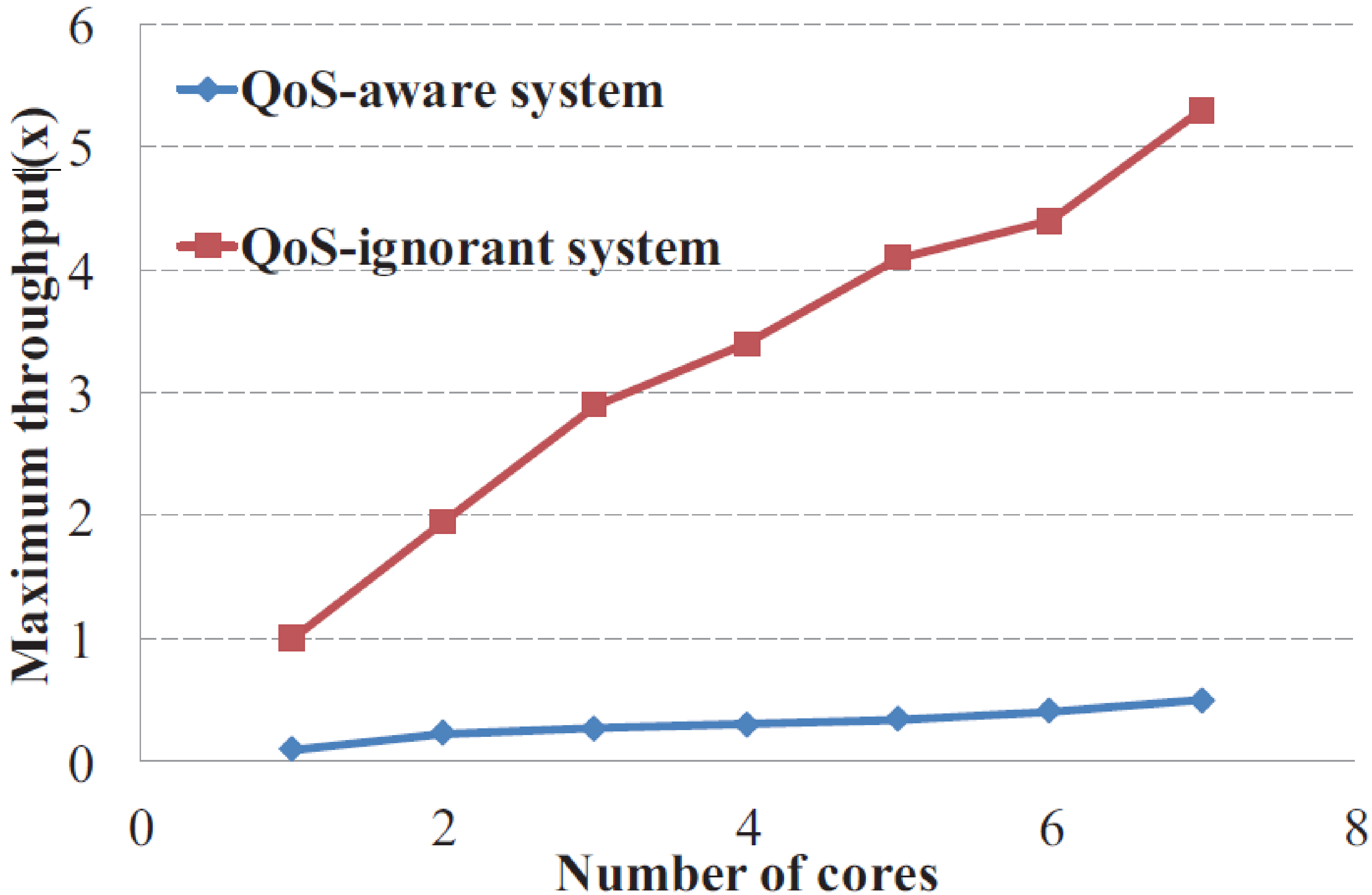
Evaluation

- Resources' effects on failure rate



Evaluation

- Maximum scalability



Conclusion

- Enable QoS support in a Publish/Subscribe message broker with a multi-core processor
  - Intra-core fine-grained scheduling
  - Inter-core message dispatching mechanism to provide
- Discuss about how parameters affect the message failure rate
  - Period
  - Processing time
  - The number of cores used
- Near-linear scalability/10x maximum throughput

THANK YOU☺