Boomerang: Real-Time I/O Meets Legacy Systems

Ahmad Golchin, Soham Sinha, Richard West

Boston University

Introduction

- Key challenges:
  - How to **extend** real-time system with legacy functionality?
  - How to **extend** legacy system with real-time capabilities?
  - How to **isolate** tasks and services of different criticality levels?
  - How to **guarantee** I/O processing is real-time
    - Input → Processing → Output

- Contribution:
  - **Boomerang**: I/O system built using Quest-V partitioning hypervisor and Quest RTOS
  - **Composable tuned pipes**

Spatial and Temporal Isolation

- Spatial
  - Ensure one software component cannot alter another’s private code or data, or interfere with control of its devices
- Temporal
  - Ensure a software component cannot affect when another component accesses a resource

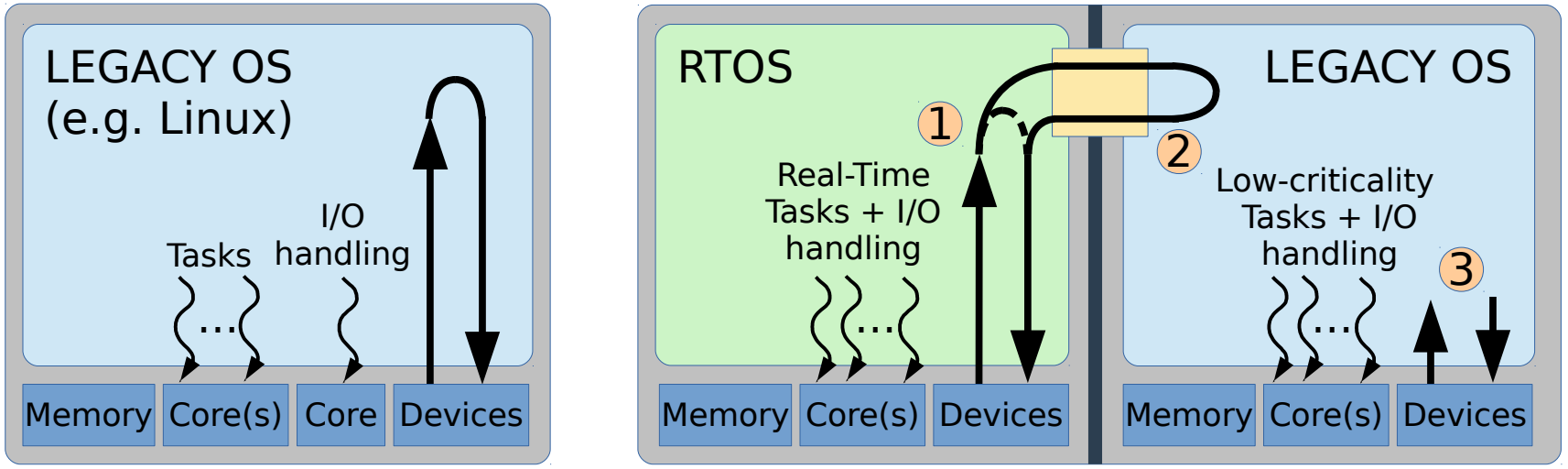
Spatial and Temporal Isolation

- Lack of isolation leads to timing and functional failures
  - Potentially catastrophic for high-criticality tasks
- Enforce isolation using separate hardware for each functional component
  - e.g., separate ECU in an automotive system
  - Not scalable or cost-effective

Spatial and Temporal Isolation

- Single hardware solution
  - Use machine virtualization for spatial isolation of tasks of different criticalities
  - Separate cores statically mapped to different virtual machines
  - Different cores used for timing-critical tasks
- Use a partitioning hypervisor to manage system
  - Quest-V

Conventional vs Boomerang I/O



(a) Conventional

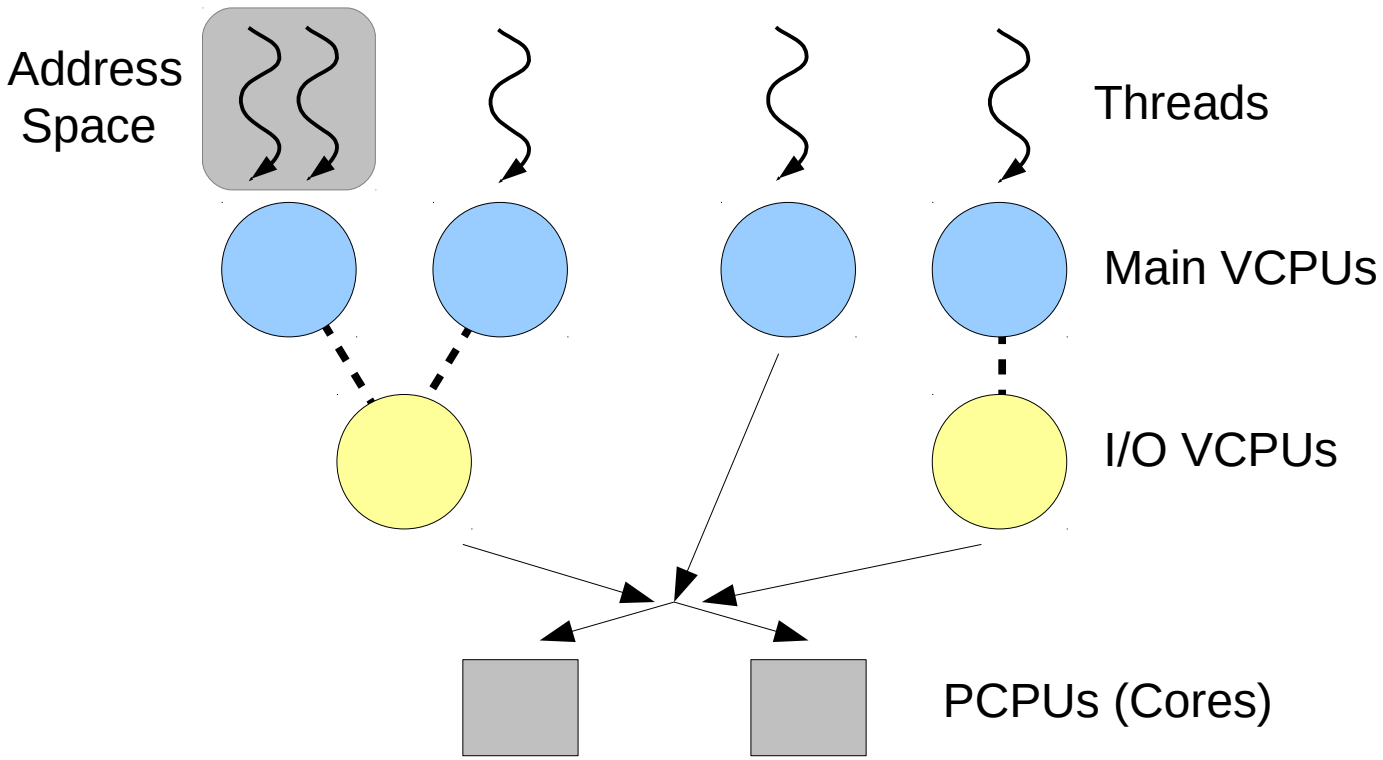
(b) Boomerang

- Boomerang uses Quest-V partitioning hypervisor to support tuned pipes between RTOS & legacy OS
  - Allows legacy OS to contribute to real-time I/O without interference from low-criticality device interrupts

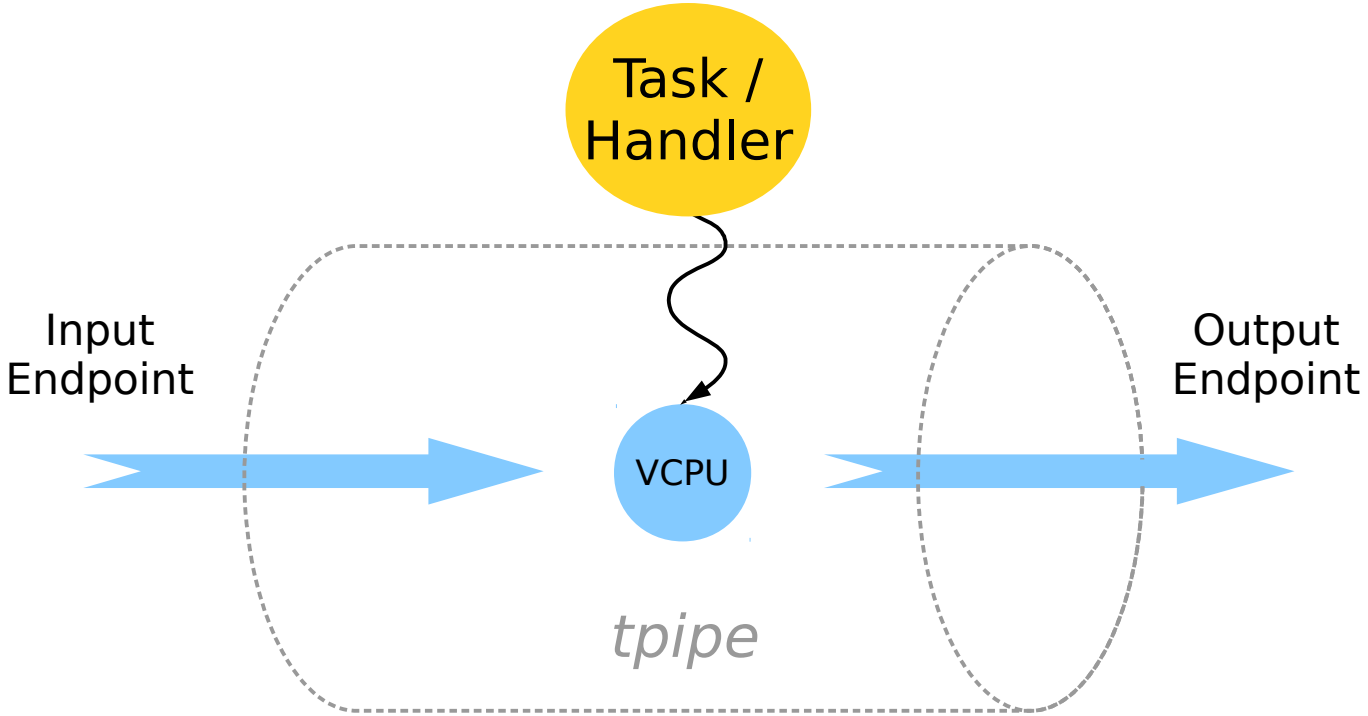
Tuned Pipes

- Like POSIX pipes but guarantee throughput and delay on communication
- Boomerang I/O subsystem supports real-time I/O (Tuned Pipes) across Quest RTOS and legacy OSes
  - Empowers legacy OSes (Linux, Android) with real-time capabilities
  - Leverages Quest RTOS VCPU scheduling of tasks and interrupts

VCPU Scheduling in Quest



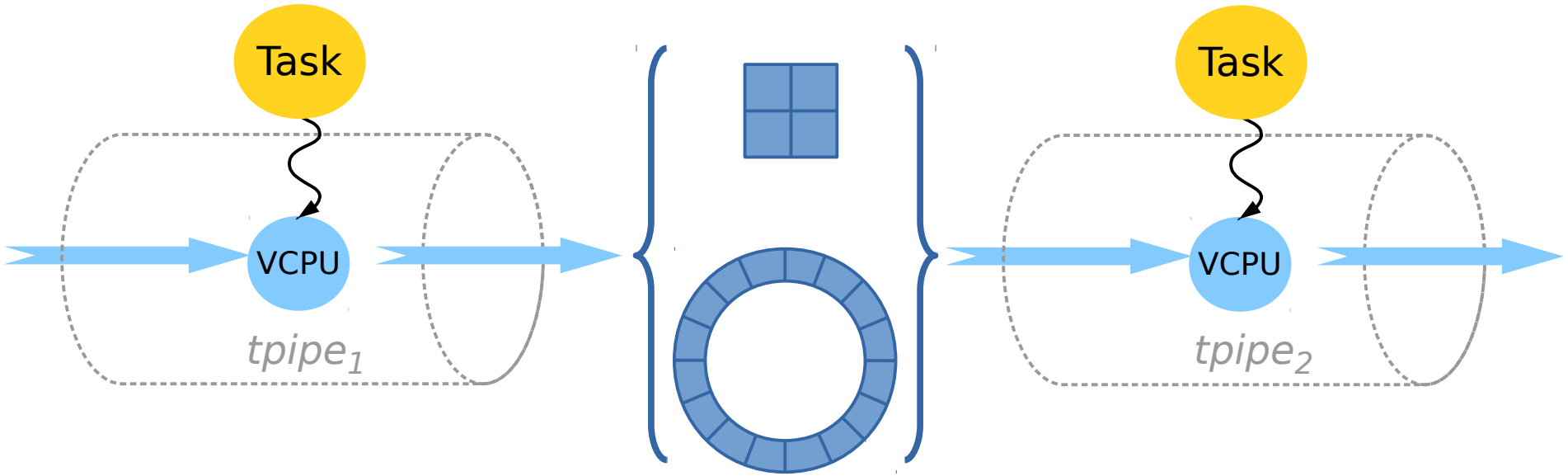
Boomerang: Tuned Pipes



tpipe_id_t tpipe (ep_t *inp[], int n_inp, ep_t *outp, qos_t spec, tpipe_task_t func, void* arg);

Boomerang automatically calculates VCPU parameters to meet QoS spec when all inputs and outputs are connected

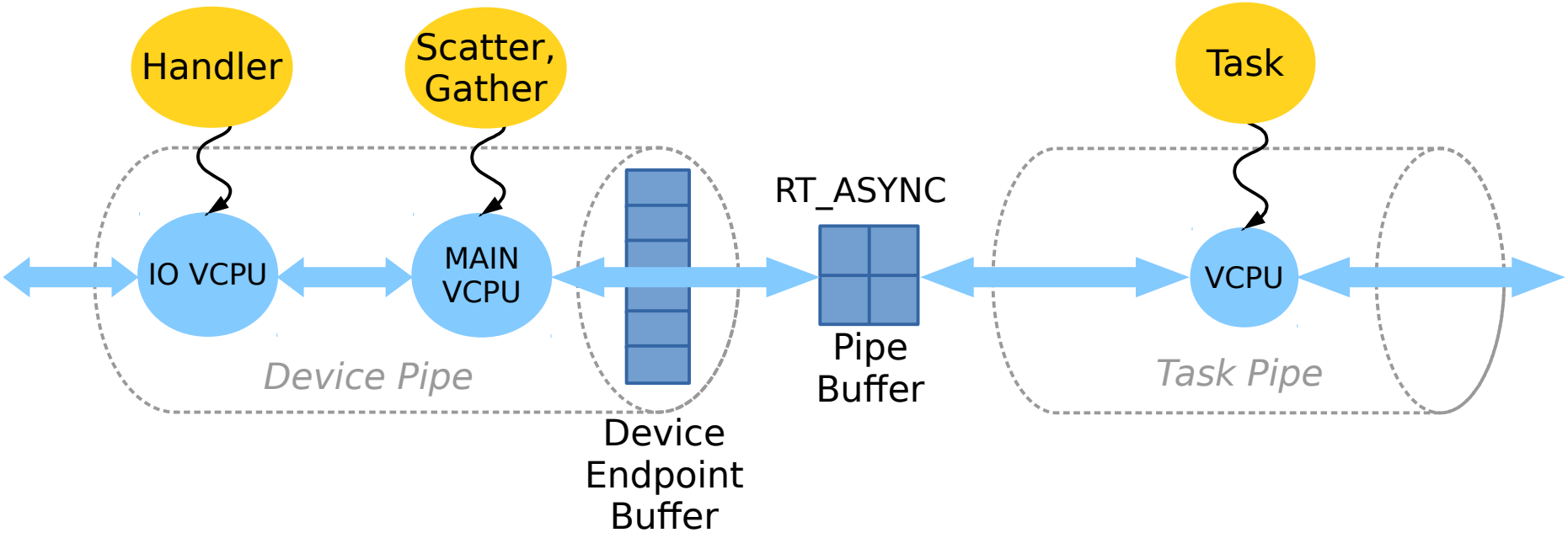
Tuned Pipes Buffering Semantics



- Pipeline composition of two or more tuned pipes

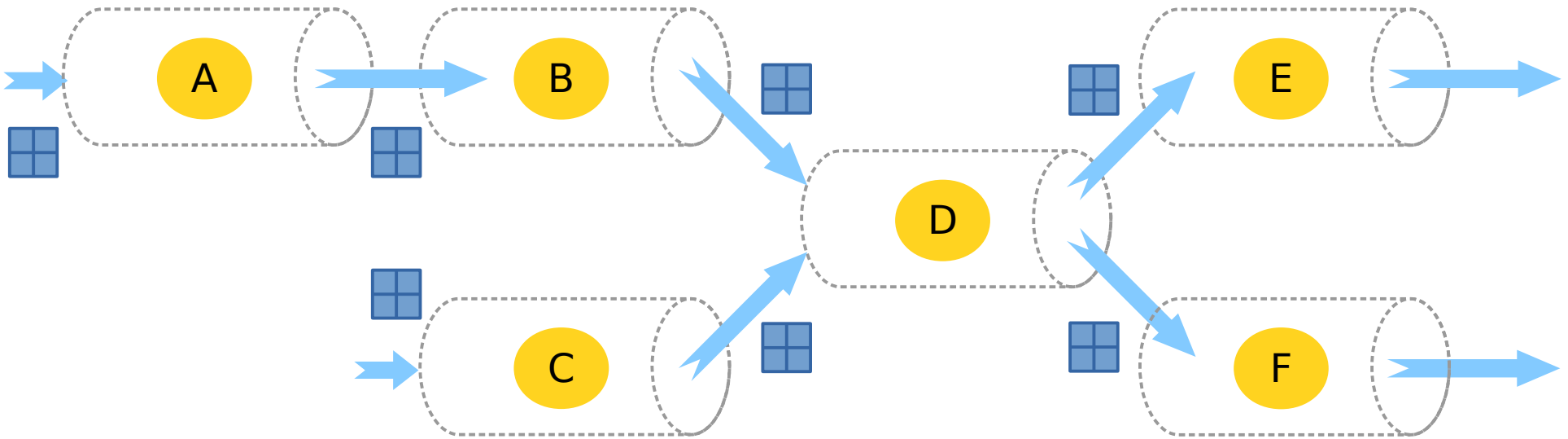
- RT_ASYNC – Simpson’s 4-slot asynchronous communication
- RT_FIFO – Ring-buffered semi-asynchronous communication

Tuned Pipes Buffering Semantics



- Pipeline composition of two or more tuned pipes
- Device Pipe – Interrupt handling (IO VCPU) + scatter/gather processing (Main VCPU)
- Task Pipe – Data processing (Main VCPU only)

Example Pipeline Composition



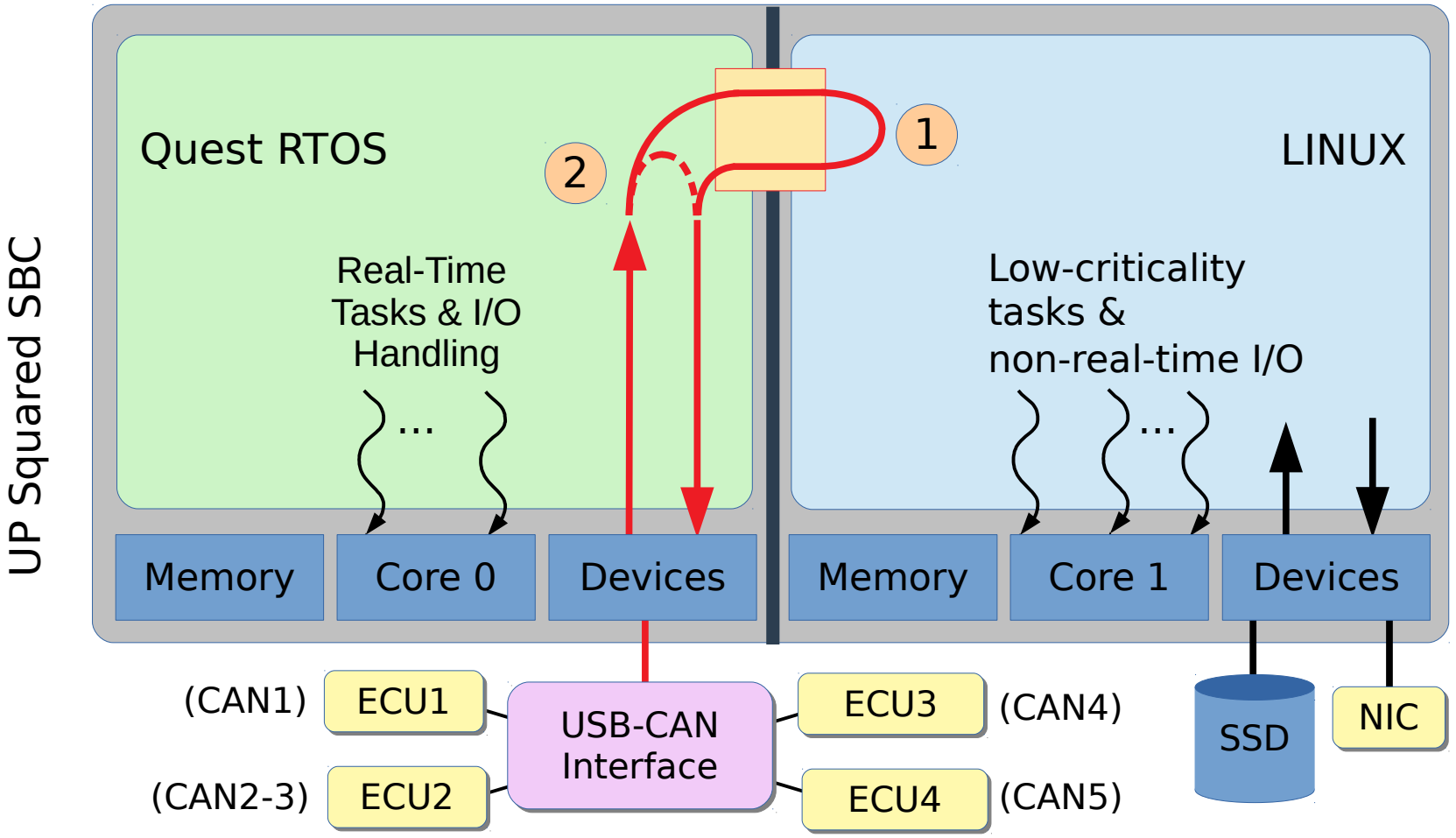
[∗](A | B), C | D | E, F [e2e_tput | loss_rate, e2e_delay]

- Parallel pipelines separated by commas
- Asterisk enforces ring-buffered lossless communication
- e2e_tput: min msgs/time exiting final pipe (for lossless comms)
- e2e_delay: applies to longest path

Boomerang: Pipeline Constraints

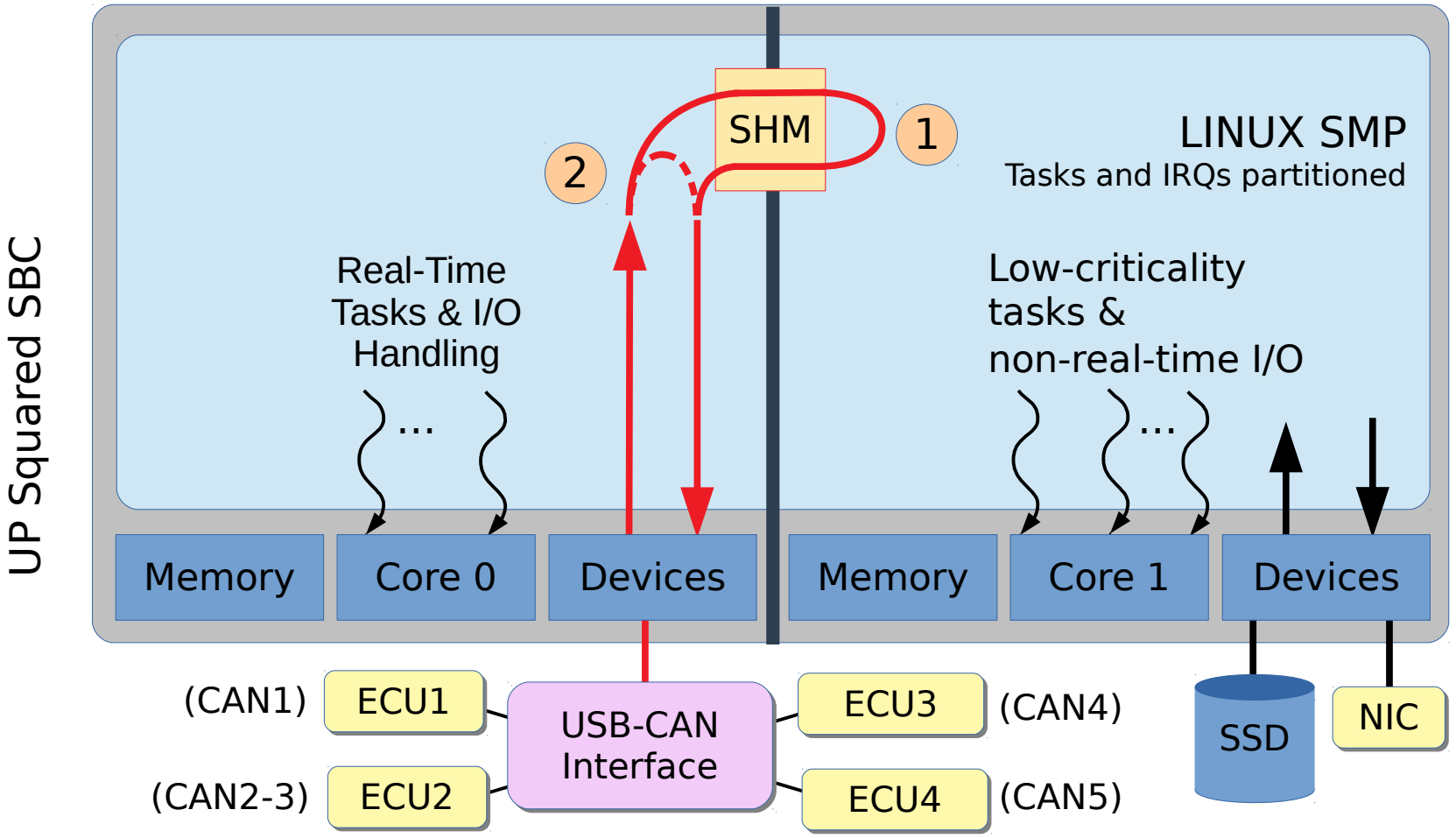
- Automatically establish VCPU (C,T) tuned pipe parameters satisfying:
  - For FIFO and 4-slot:
    1) $\sum_{i \in l} T_i \leq e2e\_delay \text{ for longest path } l$
    2) $\text{All task scheduling constraints are met}$
  - For FIFO only:
    3) $\min \forall i \{ \frac{m_i}{T_i} \} \geq e2e\_tput \text{, for } m_i \geq 1 \text{ messages transferred by } tpipe_i \text{ every } C_i$
    4) $\text{All FIFO buffers are sized to ensure no additional blocking delays}$
  - For 4-slot only:
    3) $\max \{ 1 - \frac{T_p}{T_c} \} \leq loss\_rate \text{, for all } T_p \leq T_c$

Boomerang: Experimental Setup



- Boomerang tuned pipe path (1) spans Quest + Linux + USB-CAN
- Boomerang tuned pipe path (2) spans Quest + USB-CAN

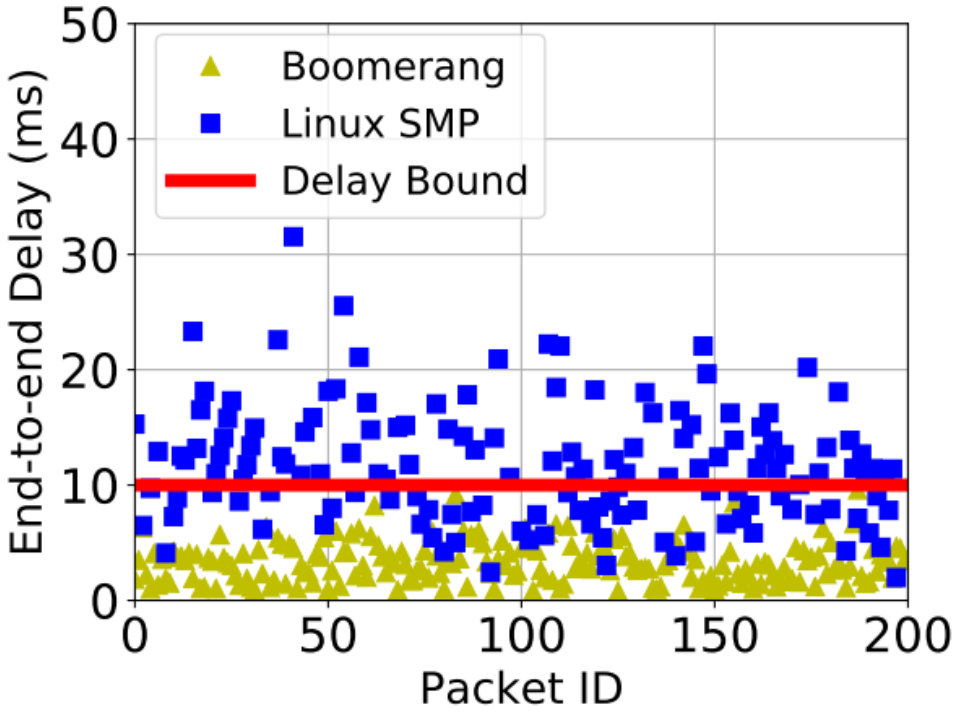
Boomerang: Experimental Setup



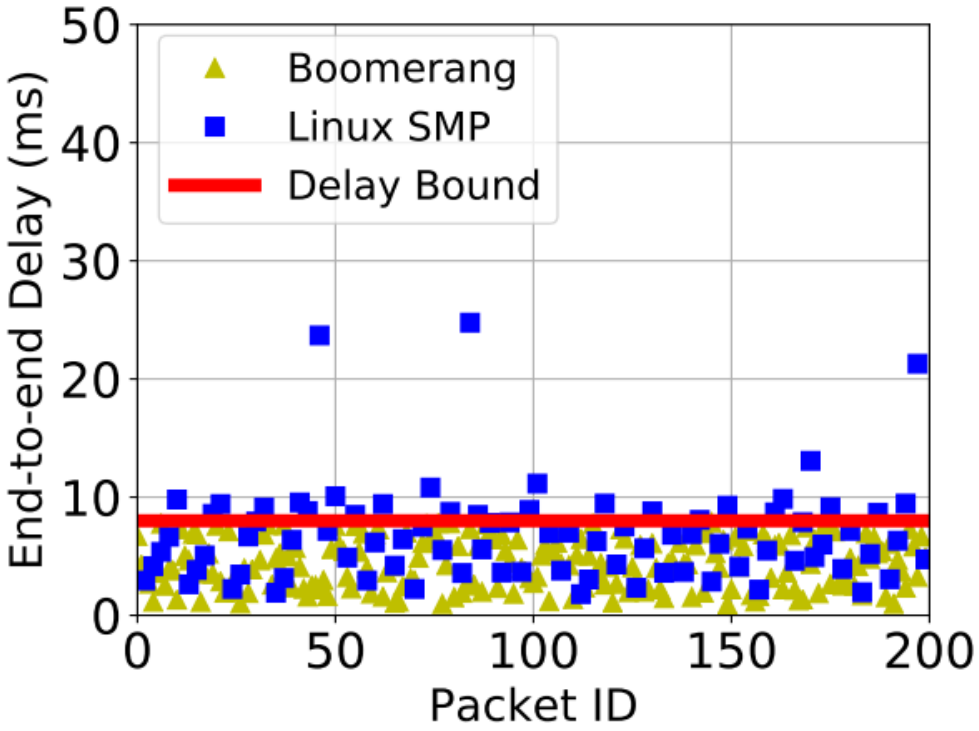
- Boomerang tuned pipe path (1) spans Quest + Linux + USB-CAN
- Boomerang tuned pipe path (2) spans Quest + USB-CAN

Boomerang: Asynchronous Results

Pipeline 1 (Asynchronous, no loss)



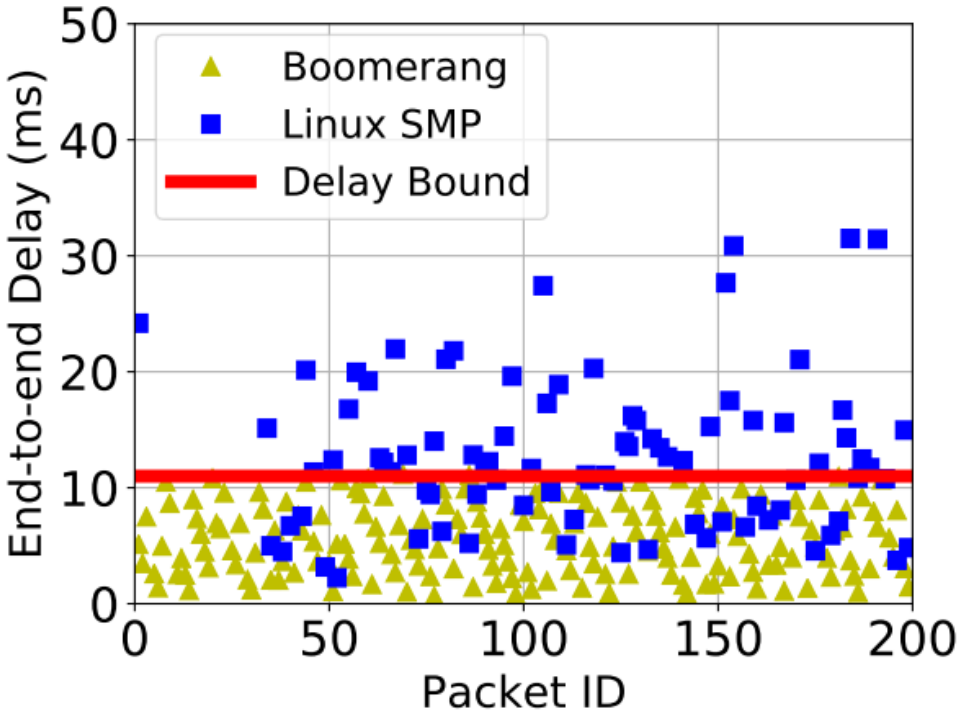
Pipeline 2 (Asynchronous, no loss)



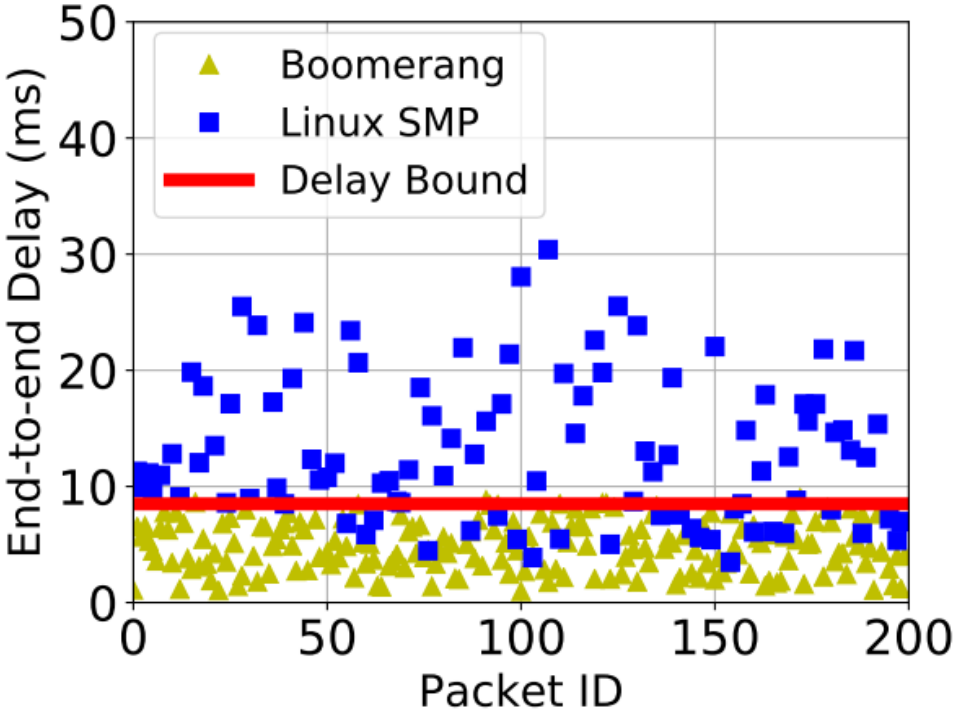
- Boomerang meets communication timing guarantees
- A Linux SMP (multicore) OS with real-time extensions cannot perform I/O predictably

Boomerang: Loss-tolerant Results

Pipeline 1 (Asynchronous, ≤20% loss)



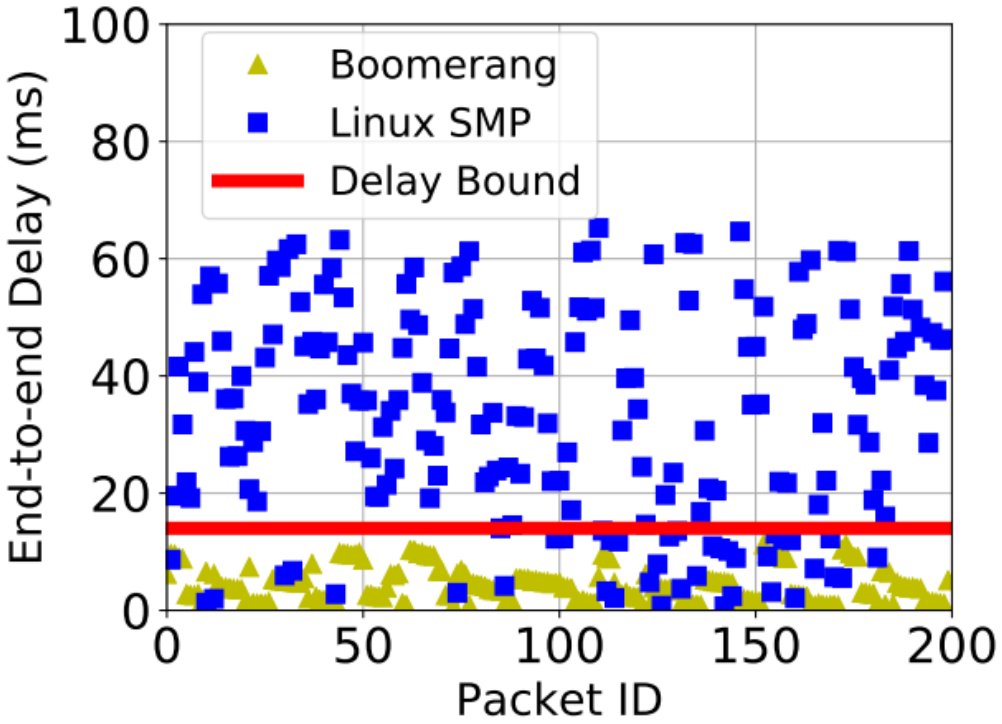
Pipeline 2 (Asynchronous, ≤20 % loss)



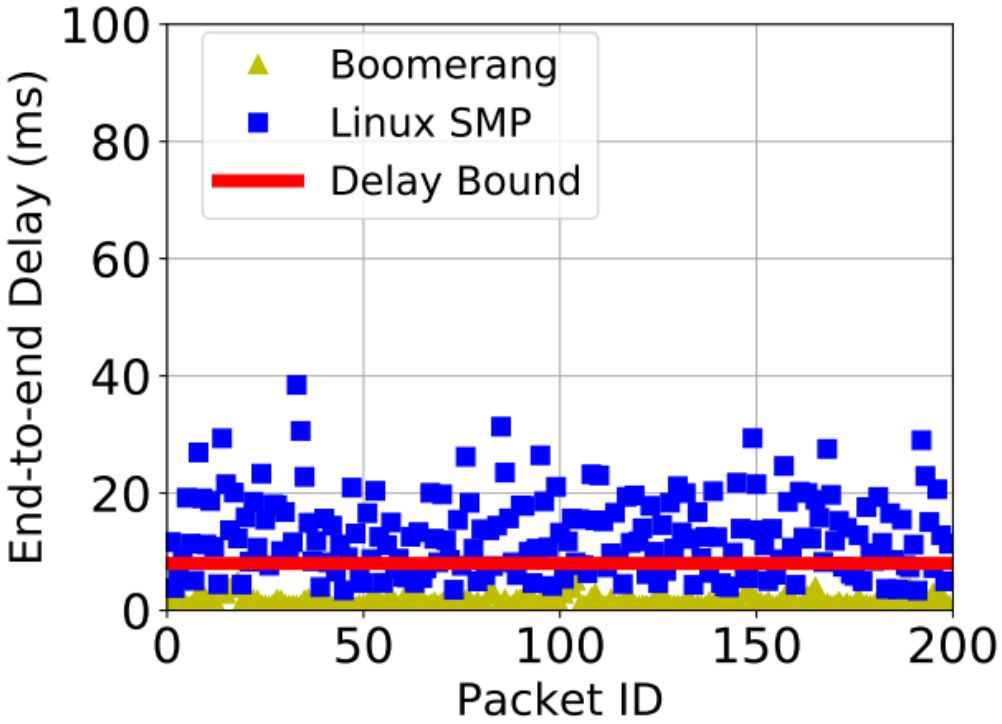
- Boomerang guarantees Pipeline 1 (3.5% loss) and Pipeline 2 (0% loss)
- Linux SMP fails Pipeline 1 (55% loss) and Pipeline 2 (50% loss)

Boomerang: Synchronous Results

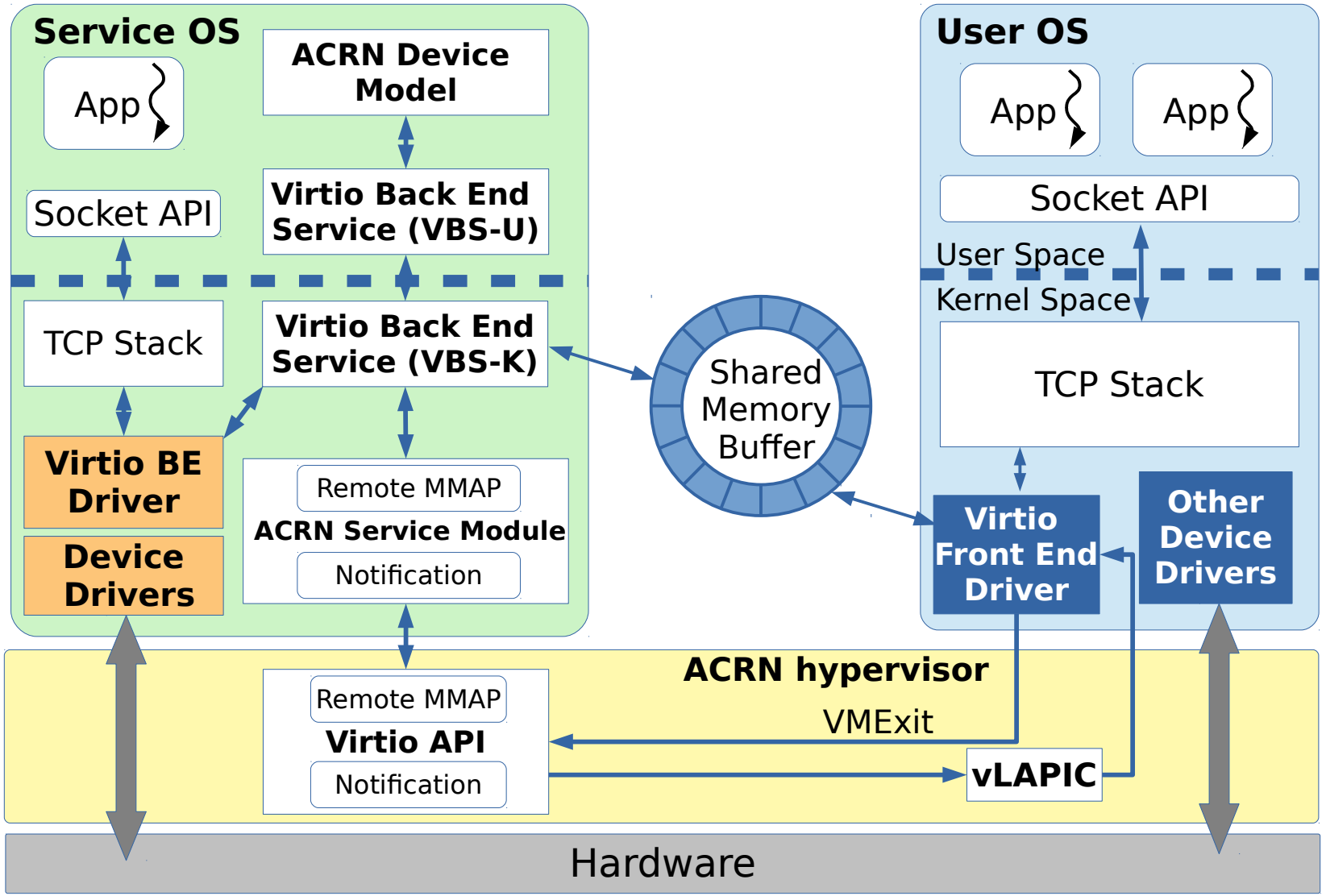
Pipeline 1 (FIFO buffering)



Pipeline 2 (FIFO buffering)

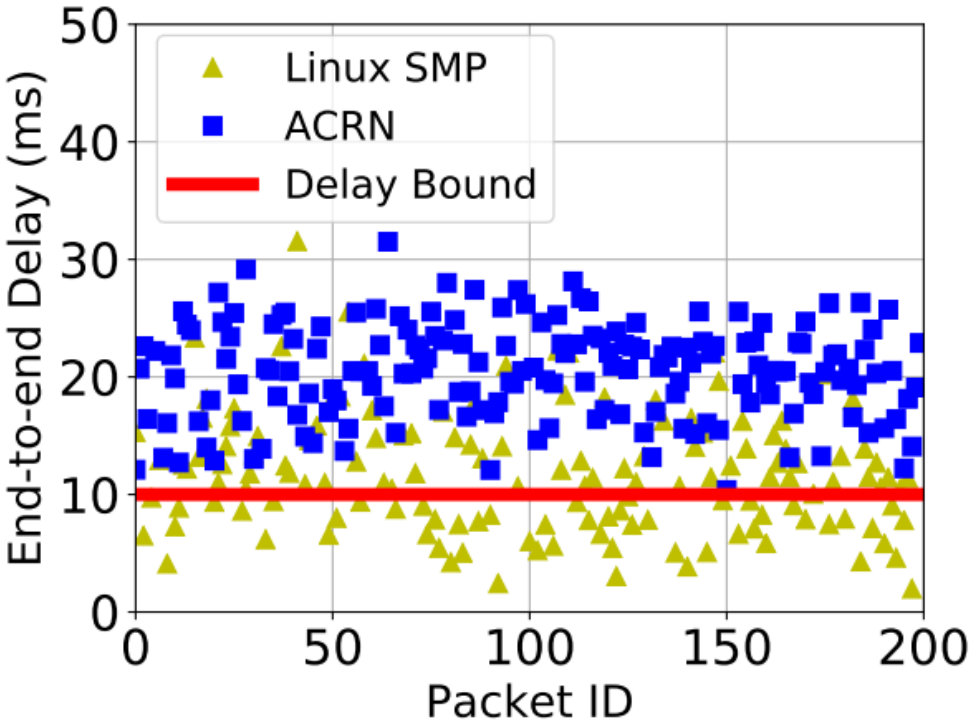


ACRN: Partitioning Hypervisor

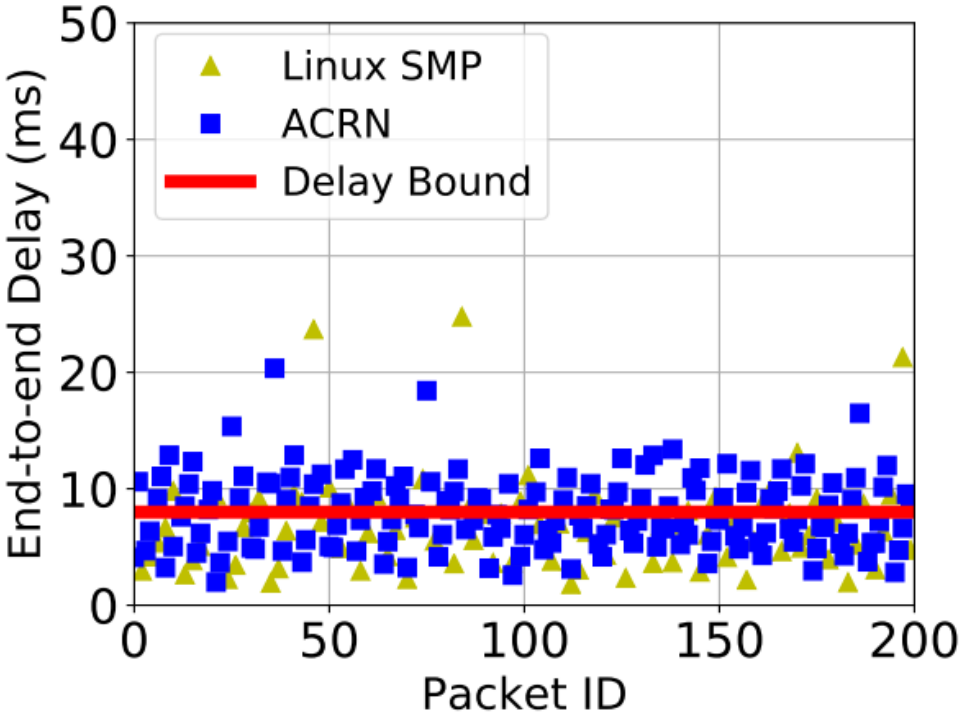


ACRN: Results

Pipeline 1 (Asynchronous, no loss)



Pipeline 2 (Asynchronous, no loss)



- ACRN generally worse than Linux SMP
- Neither as good as Boomerang (previously shown)

Conclusions

- Boomerang I/O system built for Quest-V partitioning hypervisor
  - Supports composable tuned pipes between guests
  - Empowers Non-RT OS with RT capabilities
  - Automatically tunes VCPU parameters
  - Guarantees E2E throughput, delay & loss
  - Outperforms Linux SMP (RT-PREEMPT + DEADLINE) & ACRN