ETH
Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

The multikernel
A new OS architecture for scalable multicore systems

Baumann, Barham et al.

Claudio Gargiulo
gargiulo.claudio@gmail.com

Monolithic OS

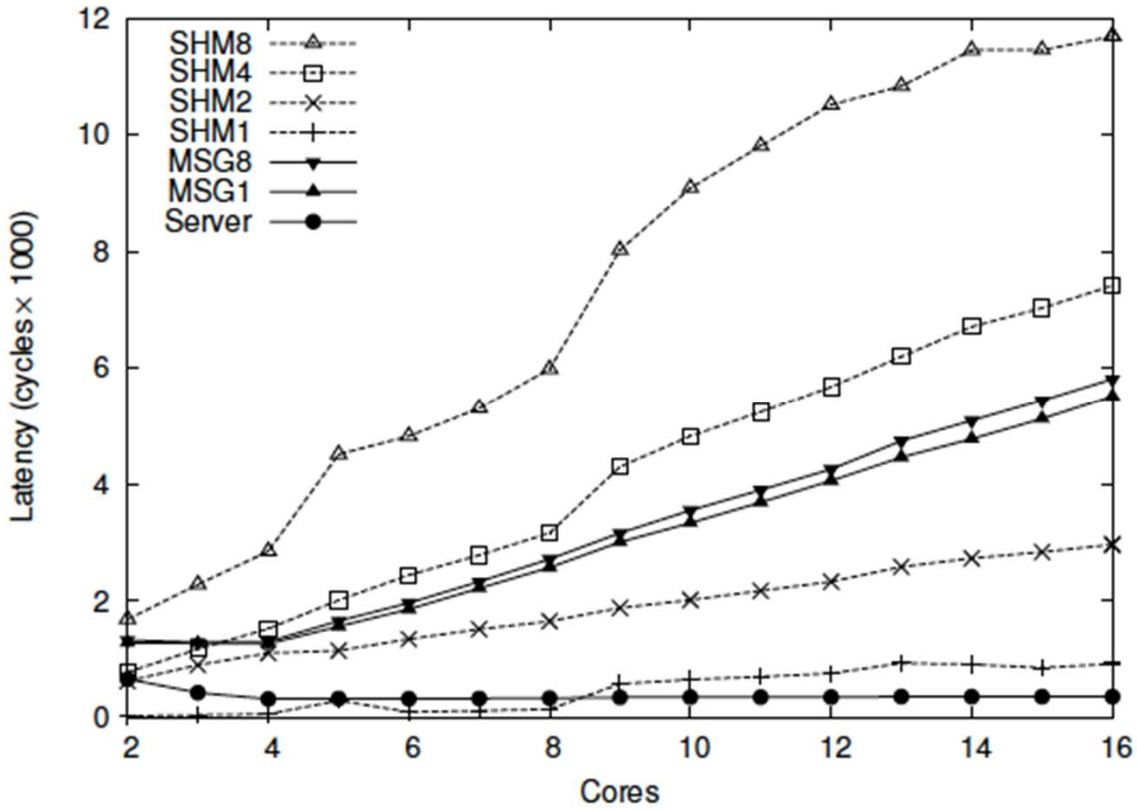
- Shared memory
- Supports different types of hardware
- Most of the OS layers run in kernel-mode
- OS state shared between cores
- Hardware-specific synchronization scheme

Monolithic OS: problems

- Hard to upgrade
- Not optimized for specific hardware
- Shared memory costs more than message passing

Shared memory vs. message passing

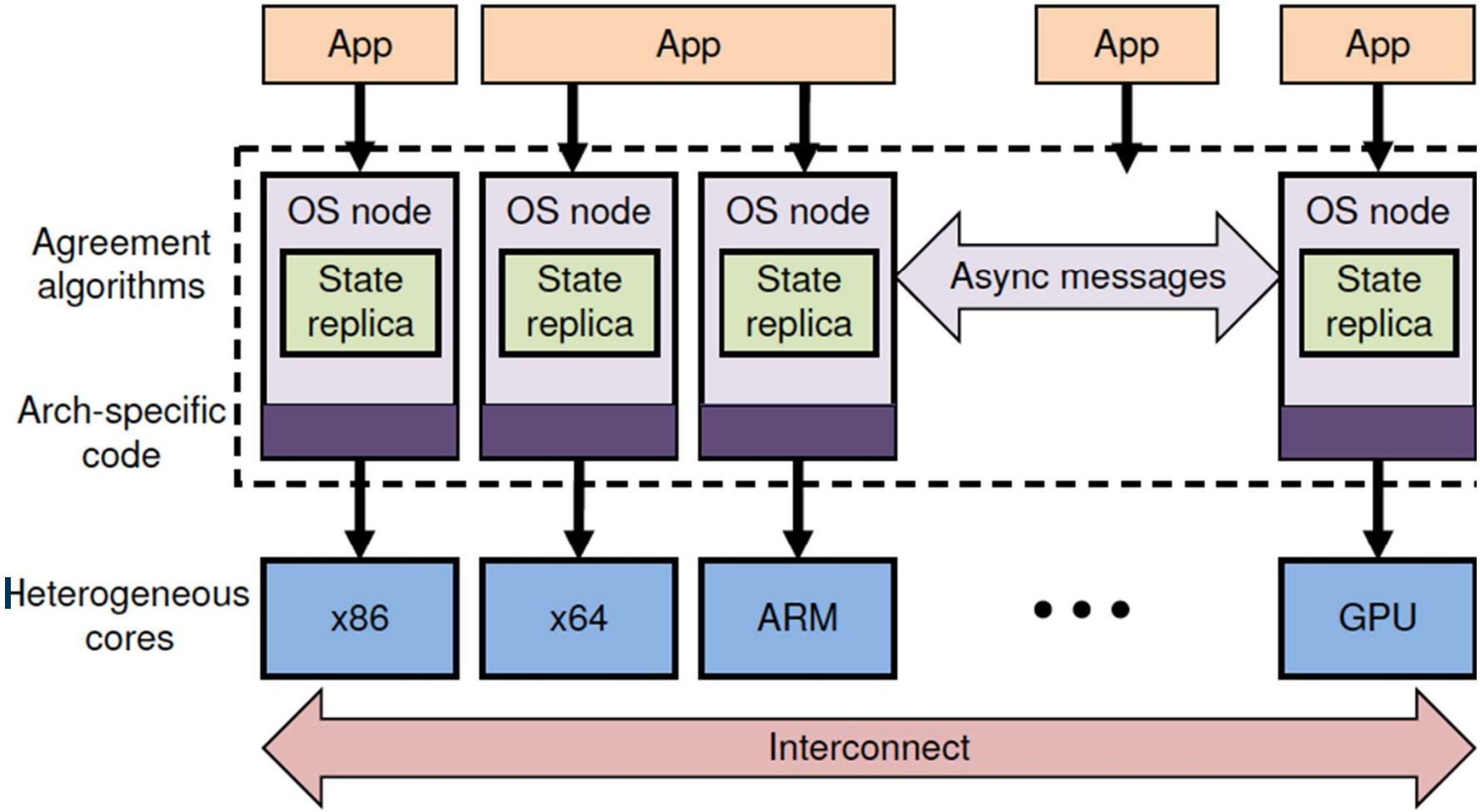
- Message cost for routing and bus congestion:
  - **Higher cost** for shared memory



Proposed approach: the multikernel model

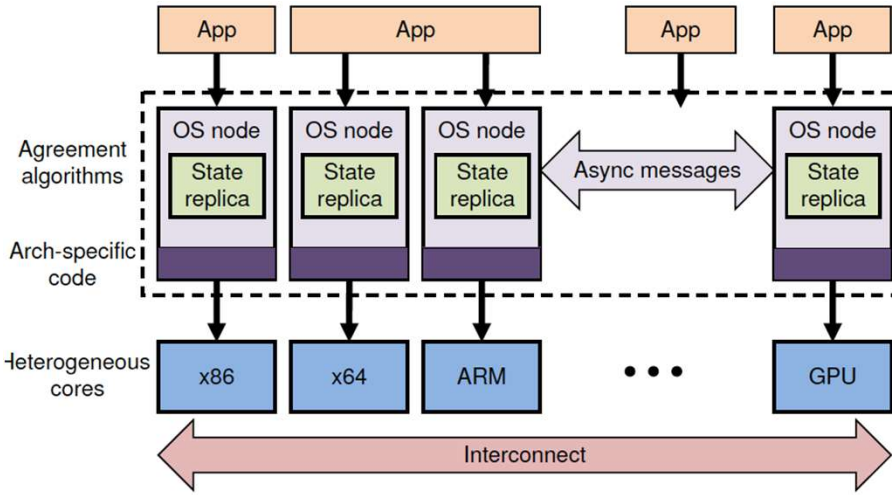
- Key features:
  - **Hardware neutrality**
  - Replicated OS state
  - Message passing for inter-core communication
  - Reuse of distributed systems optimizations
    - event-based communication

The multikernel: structure



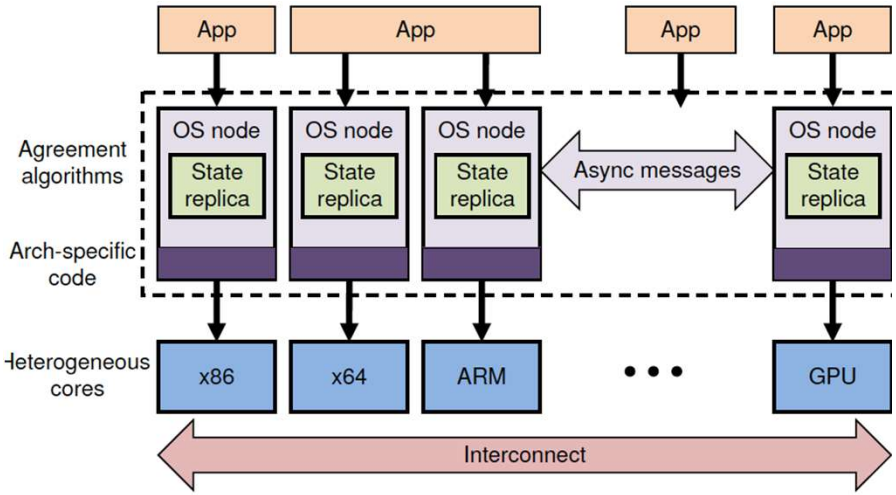
The multikernel: hardware neutrality

- Communication algorithms must be efficient
- Multikernel model:
  - Late binding of protocol implementation and message transport
  - Message transport optimized
  - Message-based algorithms hardware independent



The multikernel: replicated OS state

- Consistency maintained due to messages
- Bring data near the cores
- Improve scalability
- Ability to support hotplugging of cores

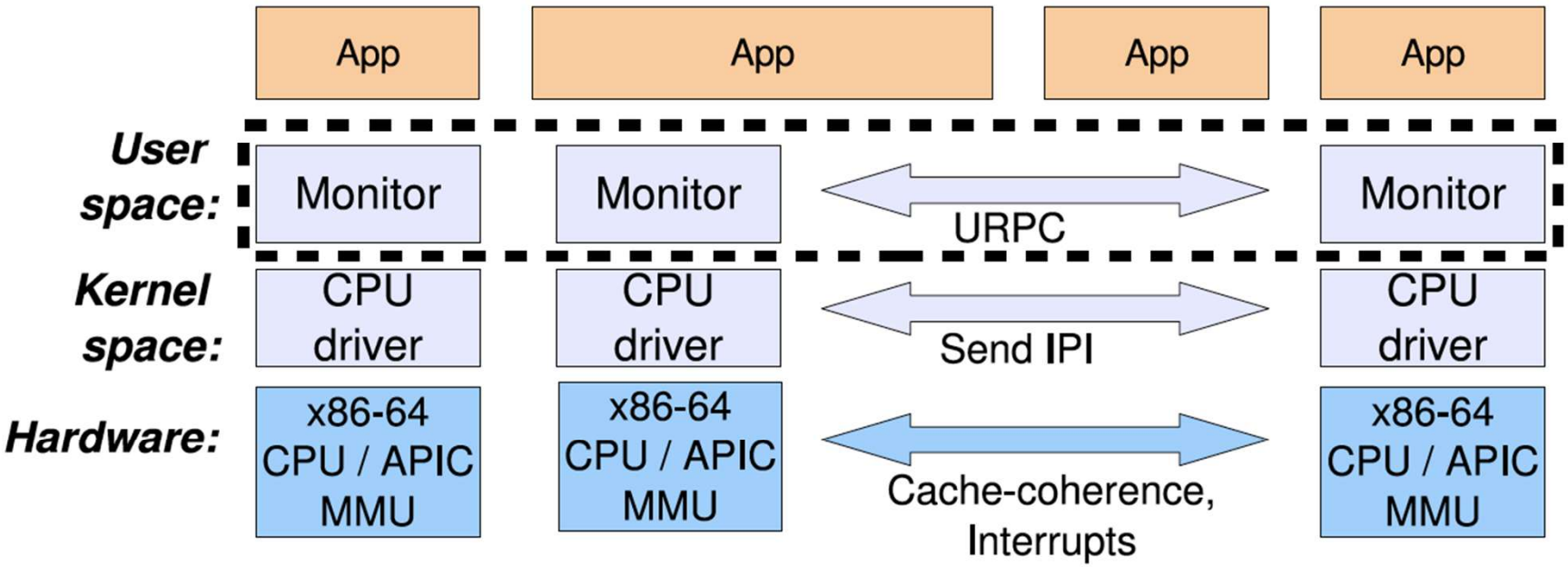


Barrelfish: a multikernel implementation

- The Barrelfish's multikernel model:
  - Idealist → platform-specific optimizations may be sacrified
  - Support for multiple agreement protocols for consistency

- Main goals:
  - Comparable performance of commodity OS
  - Support different hardware and different sharing mechanism
  - Good performance of message passing model
  - Develop a modular OS

Barrelfish: main structure



Barrelfish: CPU driver

- Enforces protection, authorization and mediation for accessing the core
- Performs dispatch and messaging within local processes
- Asynchronous and synchronous communication mechanisms

- No OS state shared with other cores
  - Single threaded
  - Event-driven
  - Non-preemptable
  - Easy to debug

Barrelfish: monitors

- Coordinate system-wide state
- Block and wake up local processes
- Work at user-space level
  - Schedulable!
  - Long-running remote operations
- Coordination by using agreement protocol

Barrelfish: process structure

- Every process as a group of dispatchers
  - One dispatcher per core
  - Communication between dispatchers
  - Dispatchers scheduled by the CPU driver

- Threads package similar to POSIX threads

Barrelfish: Inter-core communication

- Communication with cache-coherent memory
- Implementation tailored to minimize the number of interconnect messages
- Reception of URPC made by polling memory
- Optimized due to:
  - pipelining
  - prefetching instructions

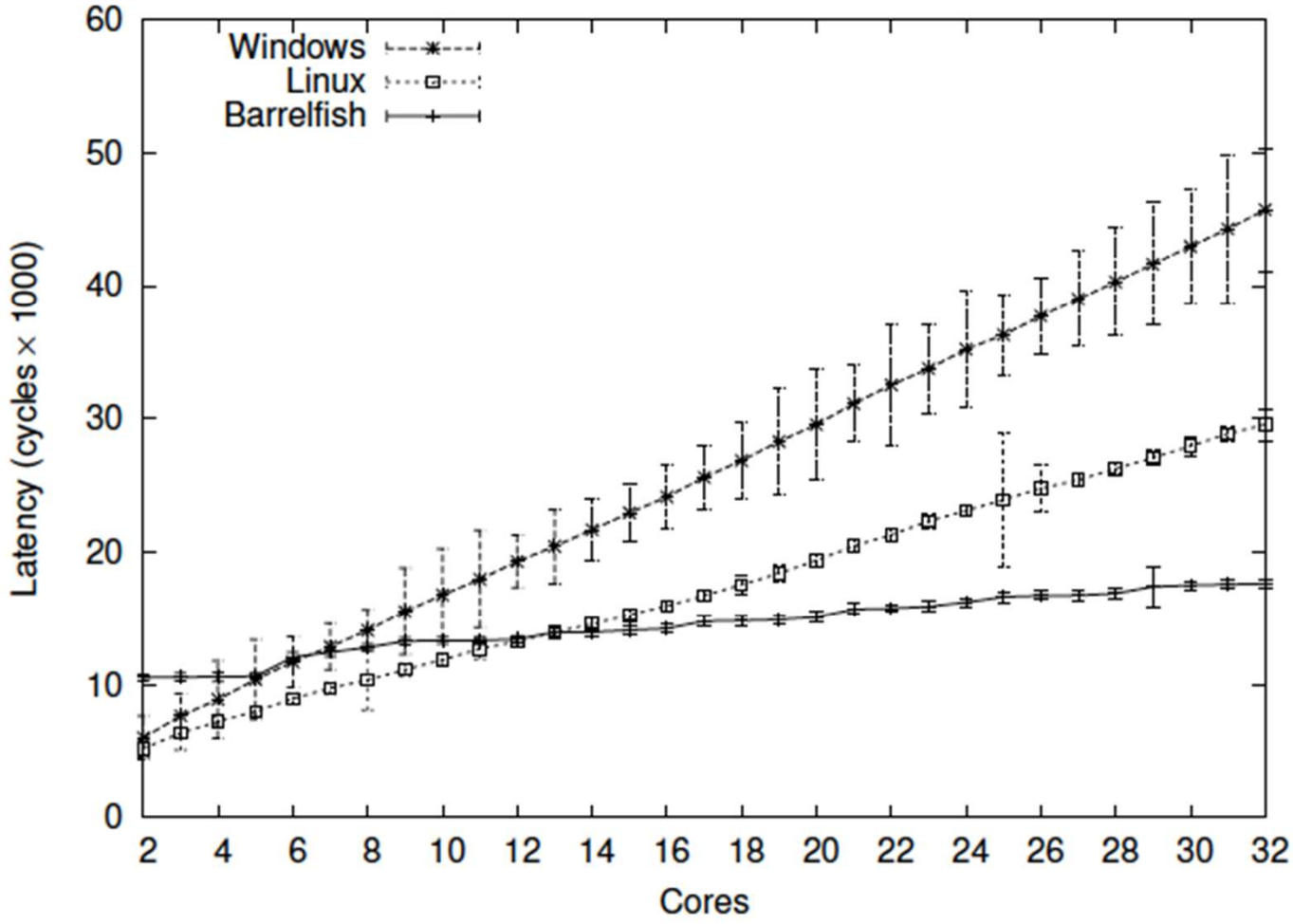
Barrelfish: memory management

- The allocation of the memory must be consistent
  - A user process can access an assigned memory region
- Tracking of ownership by using capabilities
  - Memory management performed through system calls
  - VM management made by user-level code
  - CPU driver only checks the capabilities
  - Decentralized memory management for achieving higher scalability

Barrelfish: shared address space

- can be achived by sharing a hardware page table among the dispatchers
  - Highly efficient
- or by replicating hardware page table
  - Reduce TLB invalidations
  - Support different page table formats

Barrelfish: Unmap latency test



Barrelfish: IP loopback performances

	Barrelfish	Linux
Throughput (Mbit/s)	2154	1823
Dcache misses per packet	21	77
source → sink HT traffic* per packet	467	657
sink → source HT traffic* per packet	188	550
source → sink HT link utilization	8%	11%
sink → source HT link utilization	3%	9%

* HyperTransport traffic is measured in 32-bit dwords.

Conclusions

- Higher scale of parallelism
- Each core managed independently
- SHM model not effective for large-scale multiprocessors
- Not a real heterogeneous environment is supported
- Model can be applied in one or between many machines

- Future works
  - A declarative language approach to device configuration
  - AC: Composable Asynchronous IO for Native Languages