Supercomputing Operating Systems: A Naive View from Over the Fence

Timothy Roscoe (Mothy)

Systems Group, ETH Zurich

Disclaimer:
I am a stranger in a strange land

Thank you for inviting me!

- I’m assuming your field is “Supercomputing”
- Mine isn’t: I’m a “mainstream” OS researcher
  - Expect considerable naïveté on my part

Disclaimer:
I am a stranger in a strange land

Thank you for inviting me!

- I'm assuming your field is "Supercomputing"
- Mine isn't: I'm a "mainstream" OS researcher
  - Expect considerable naïveté on my part
- This talk is about the possible intersection and interaction of "Supercomputing" and "OS research"
- I will exaggerate for effect.
  - Please don't take it the wrong way.

Traditionally…

- Supercomputing people built and programmed their own machines
  - Wrote their own operating systems and/or complained about the existing ones

Traditionally…

- Supercomputing people built and programmed their own machines
  - Wrote their own operating systems and/or complained about the existing ones
- Mainstream OS people ignored them
  - Insignificant market, no real users
  - Weird, expensive hardware (too many cores)

Traditionally…

Systems@ETH zürich

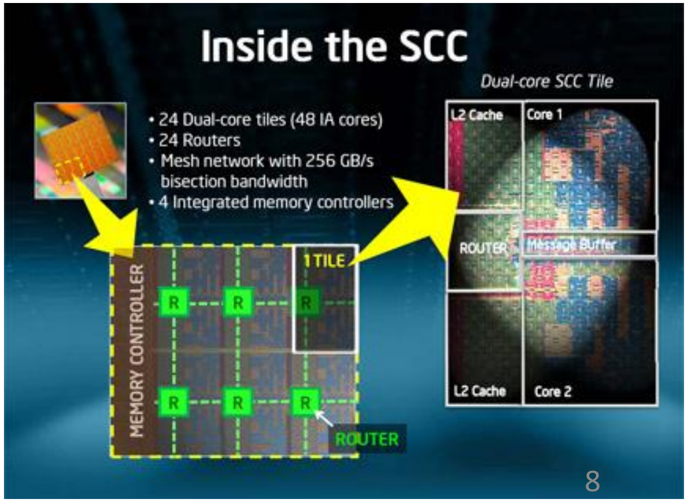
- Supercomputing people built and programmed their own machines
  - Wrote their own operating systems and/or complained about the existing ones
- Mainstream OS people ignored them
  - Insignificant market, no real users
  - Weird, expensive hardware (too many cores)

This is, of course, changing.

What's happening in general-purpose computing?

Lots more cores per chip

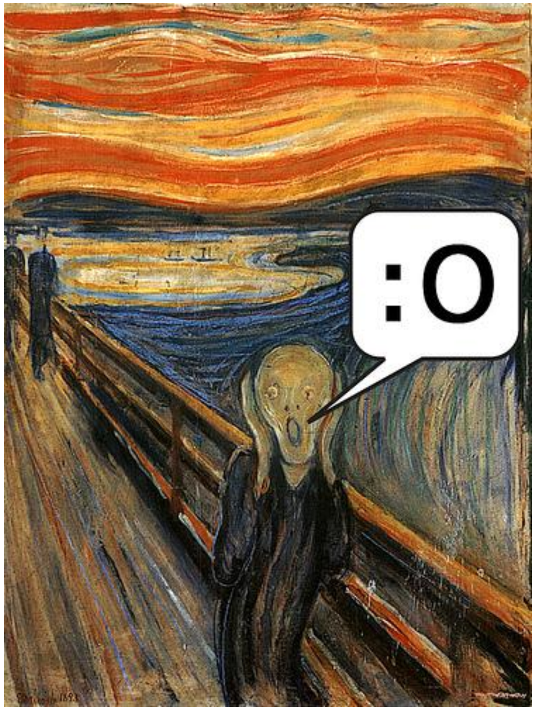
- Core counts now follow Moore's Law
- Cores will come and go
  - Energy!
- Diversity of system and processor configurations will grow
- Cache coherence may not scale to whole machine



Parallelism

- “End of the free lunch”:
  cores are not getting faster!

- Higher performance
  $\Rightarrow$ better parallelism
- New applications
  $\Rightarrow$ parallel applications
  - Mining
  - Recognition
  - Synthesis



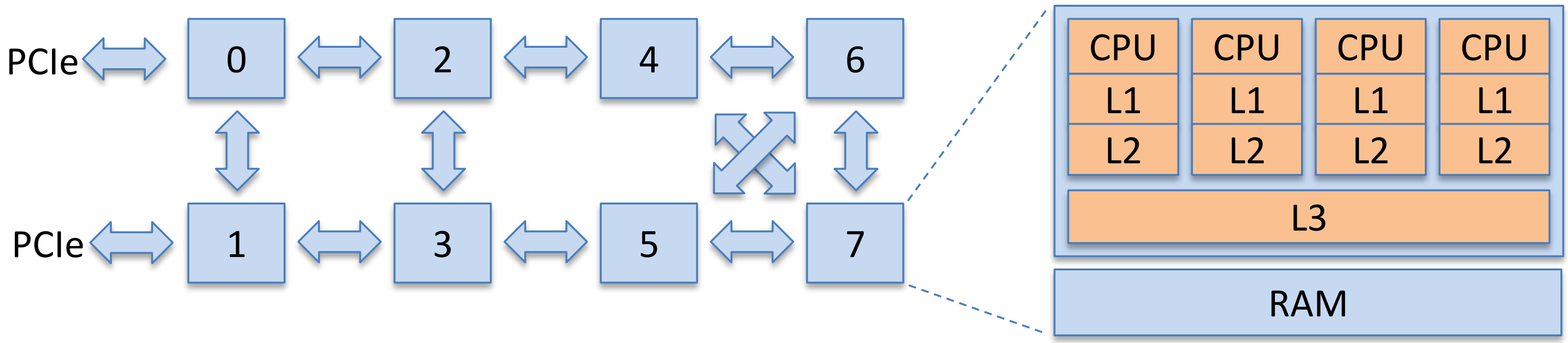
Cores will be heterogeneous

Systems@ETH zürich

- NUMA is the norm today
- Heterogeneous cores for power reduction
- Dark silicon, specialized cores
- Integrated GPUs / Crypto / NPUs etc.
- Programmable peripherals

Communication latency really matters

Example: 8 * quad-core AMD Opteron



Access	cycles	normalized to L1	per-hop cost
L1 cache	2	1	-
L2 cache	15	7.5	-
L3 cache	75	37.5	-
Other L1/L2	130	65	-
1-hop cache	190	95	60
2-hop cache	260	130	70

Implications

- Computers are systems of cores and other devices which:
  - Are connected by highly **complex** interconnects
  - Entail significant communication **latency** between nodes
  - Consist of **heterogeneous** cores
  - Show unpredictable **diversity** of system configurations
  - Have **dynamic** core set membership
  - Provide only **limited shared** memory or cache **coherence**

Implications

- Computers are systems of cores and other devices which:
  - Are connected by highly **complex** interconnects
  - Entail significant communication **latency** between nodes
  - Consist of **heterogeneous** cores
  - Show unpredictable **diversity** of system configurations
  - Have **dynamic** core set membership
  - Provide only **limited shared** memory or cache **coherence**

The OS model of cooperating processes over a shared-memory multithreaded kernel is dead.

What's really new?

- Actually, multiprocessors are nothing new in general purpose computing
- Neither are threads: people have been building systems with threads for a long time.
  - Word, databases, games, servers, browsers, etc.
- *Concurrency* is old. We understand it.
- *Parallelism* is new.

Parallels with Supercomputing

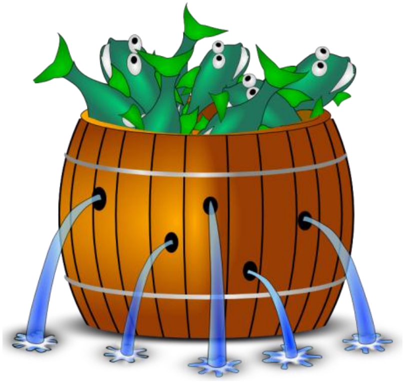
- Lots of cores
- Implies parallelism should be used!
- Message passing predominates
- Heterogeneous cores (GPUs, CellBE, etc.)
- Lots of algorithms highly tuned to complex interconnects, memory hierarchies, etc.

Surely we can use all the cool ideas in supercomputing for our new OS!

Barrelfish: our multikernel

Systems@ETH Zürich

- ETH Zurich + Microsoft Research
- Open source (MIT Licence)
- Published 2009
- Under active development
- External user community
- See www.barrelfish.org....



Non-original ideas in Barrelfish
Techniques we liked

- Capabilities for resource management (seL4)
- Minimize shared state (Tornado, K42)
- Upcall processor dispatch (Psyche, Sched. Activations)
- Push policy into user space domains (Exokernel, Nemesis)
- User-space RPC decoupled from IPIs (URPC)
- Lots of information (Infokernel)
- Single-threaded non-preemptive kernel per core (K42)
- Run drivers in their own domains (μkernels, Xen)
- Specify device registers in a little language (Devil)

What things does it run on?

Systems@ETH zürich

- PCs: 32-bit and 64-bit x86 architectures
  - Including mixture of the two!
- Intel SCC
- Intel MIC platform

Seamlessly with x86 host PCs!

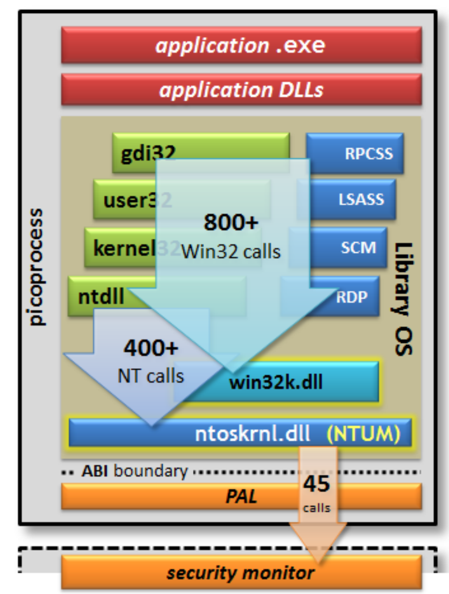
- Various ARM platforms
- Beehive
  - Experimental Microsoft Research softcore

What things run on it?

Systems@ETH Zürich

- Many microbenchmarks
- Webserver: **`http://www.barrelfish.org/`**
- Databases: SQLite, PostgreSQL, etc.
- Virtual machine monitor
  - Linux kernel binary
- Microsoft Office 2010!
  - via Drawbridge
- Parallel benchmarks:
  - Parsec, SPLASH-2, NAS

More on this later…



Rethinking OS Design #1: the Multikernel Architecture

The Multikernel Architecture

- Computers are systems of cores and other devices which:
  - Are connected by highly **complex** interconnects
  - Entail significant communication **latency** between nodes
  - Consist of **heterogeneous** cores
  - Show unpredictable **diversity** of system configurations
  - Have **dynamic** core set membership
  - Provide only **limited shared** memory or cache **coherence**

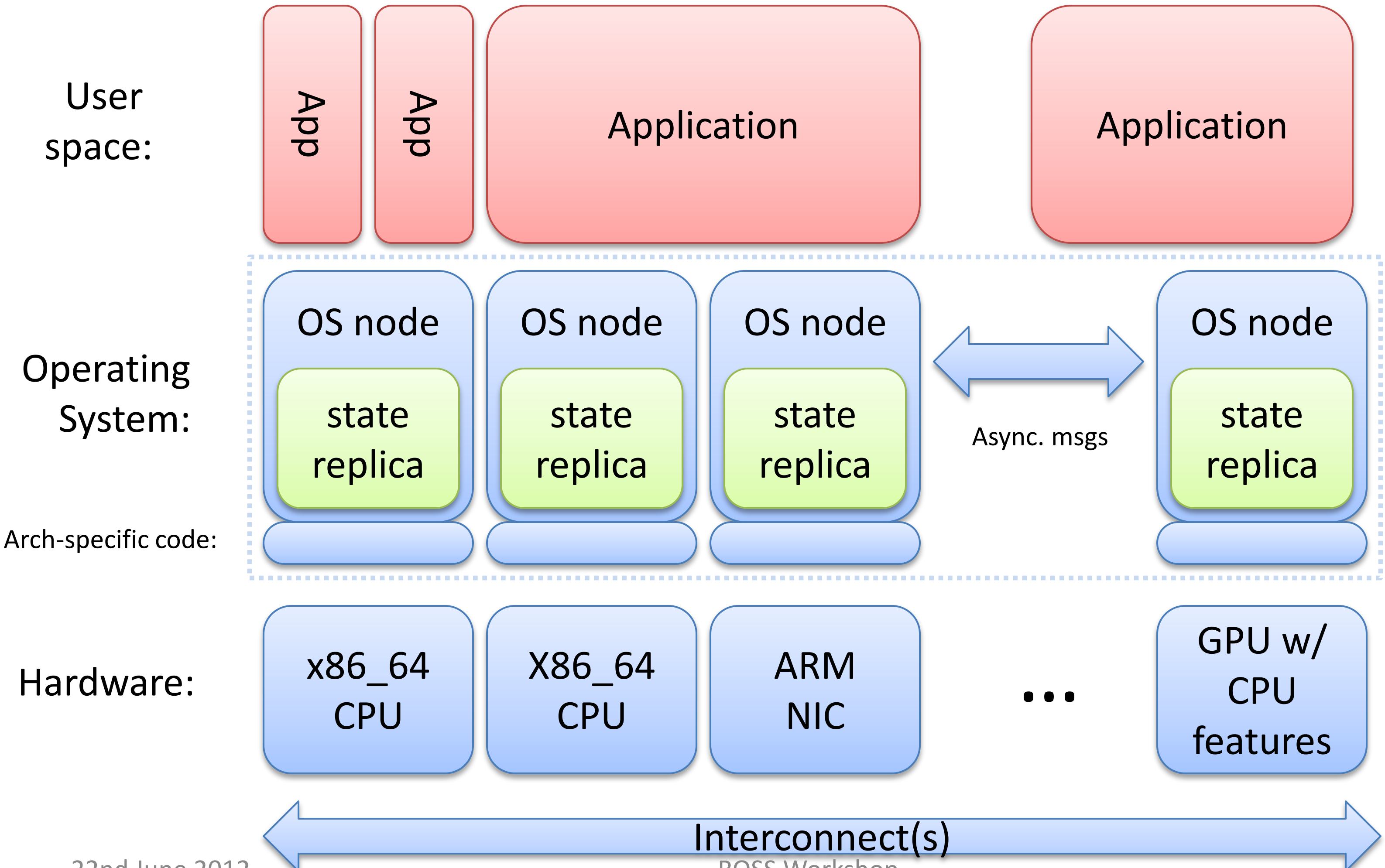
$\Rightarrow$ Forget about shared memory.

The OS is a distributed system based on **message passing**

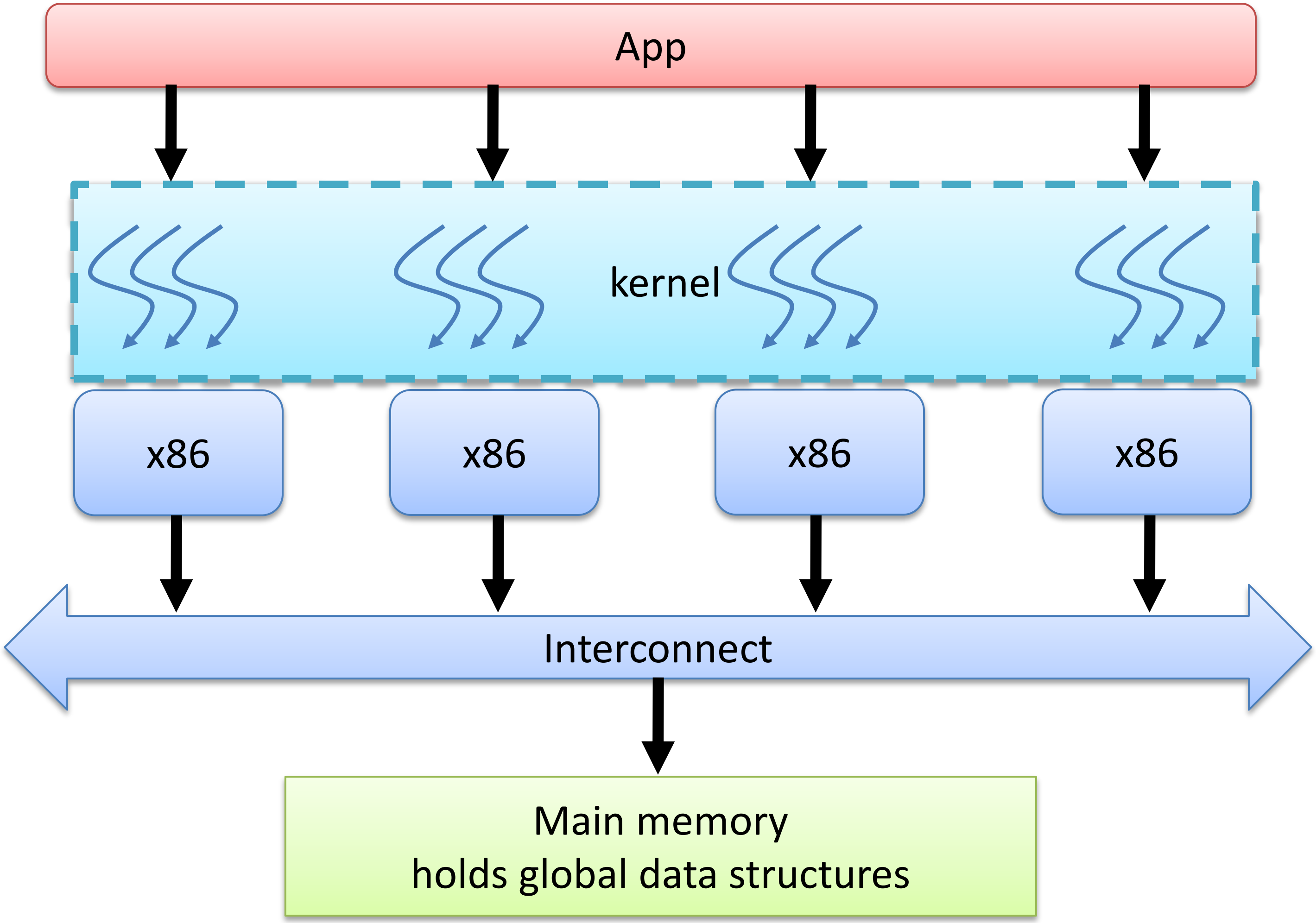
Multikernel principles

- Share ***no*** data between cores
  - All inter-core communication is via explicit messages
  - Each core can have its own implementation
- OS state partitioned if possible, replicated if not
  - State is accessed ***as if*** it were a local replica
- Invariants enforced by distributed algorithms, not locks
  - Many operations become split-phase and asynchronous

The multikernel model

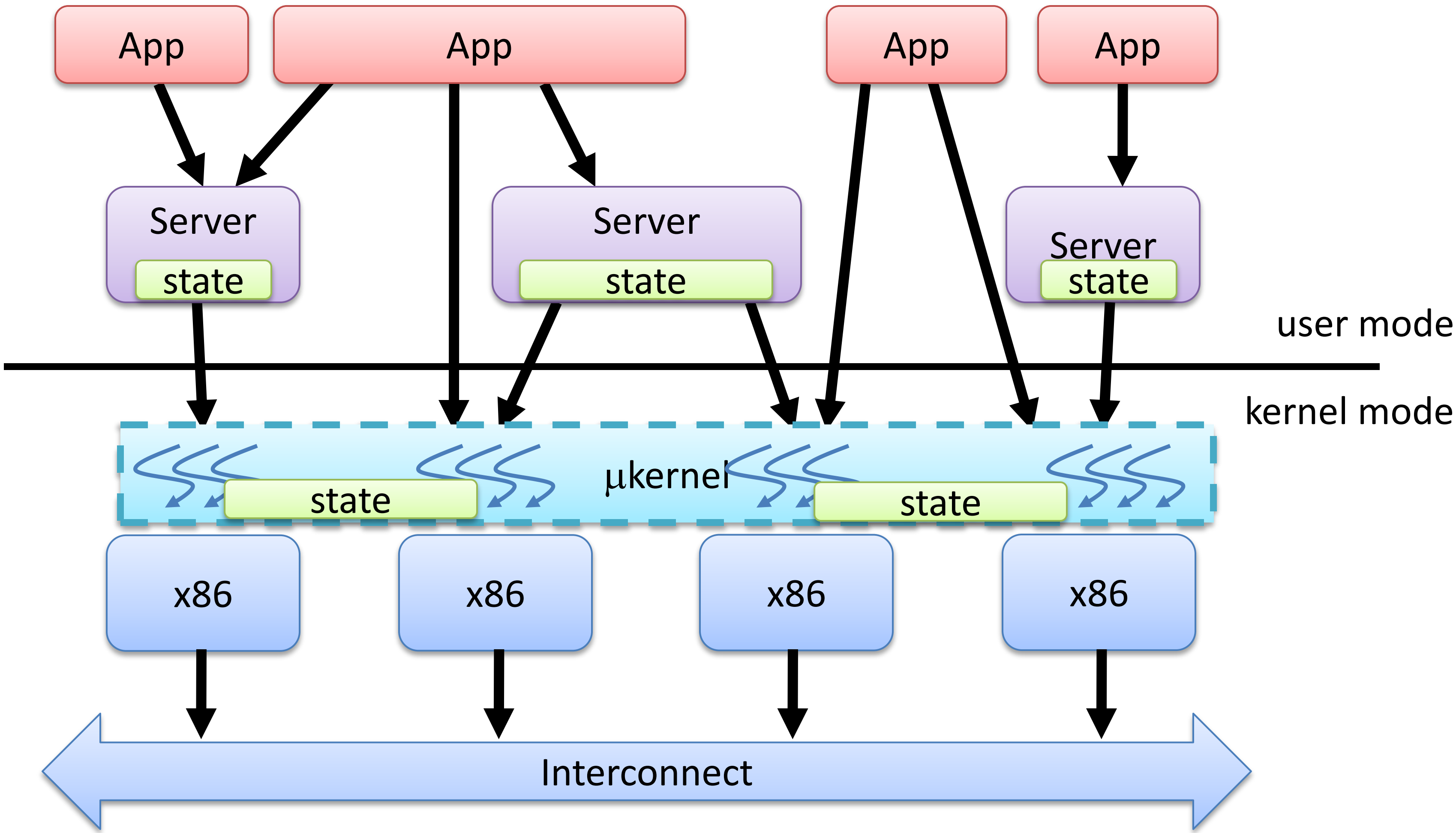


...vs a monolithic OS on multicore

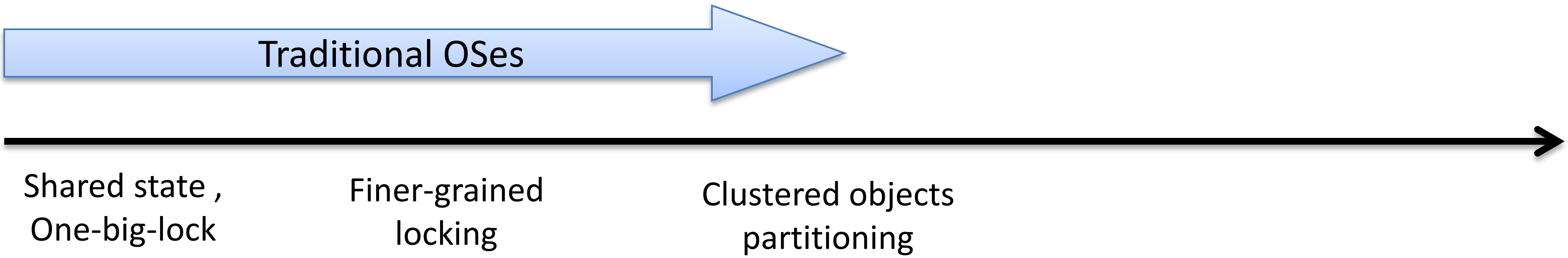


…vs a μkernel OS on multicore

Systems@ETH Zürich

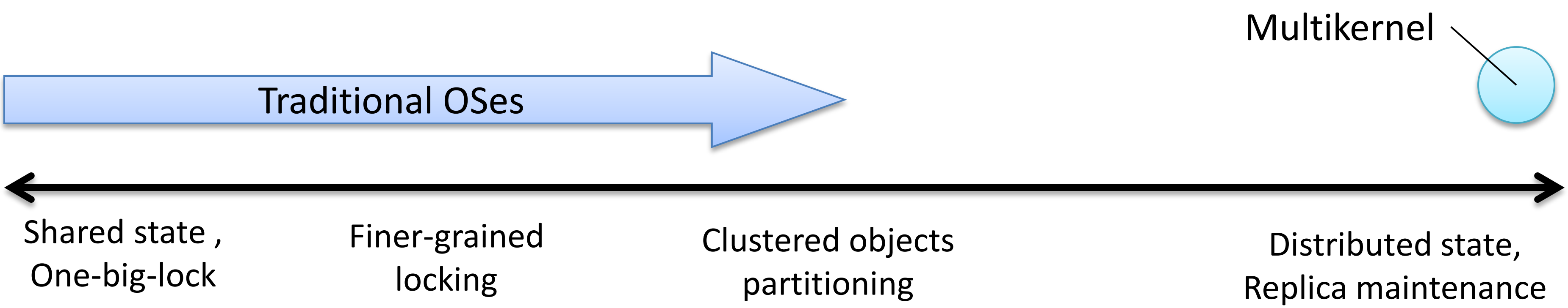


Replication vs sharing as the default



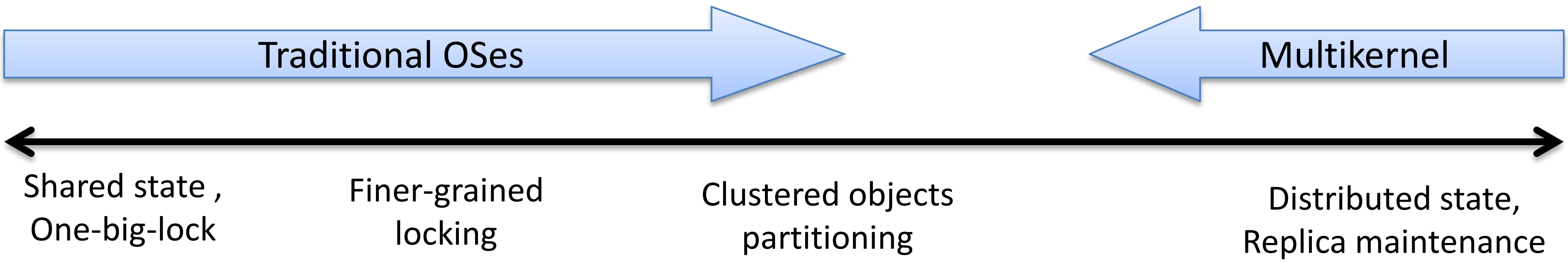
- Replicas used as an optimization in other systems

Replication vs sharing as the default



- Replicas used as an optimization in other systems

Replication vs sharing as the default



- Replicas used as an optimization in other systems
- In a multikernel, sharing is a local optimisation
  - Shared (locked) replica on closely-coupled cores
  - Only when faster, as *decided at runtime*
- Basic model remains split-phase messaging

Rethinking OS Design #2: the System Knowledge Base

System knowledge base

- Computers are systems of cores and other devices which:
  - Are connected by highly **complex** interconnects
  - Entail significant communication **latency** between nodes
  - Consist of **heterogeneous** cores
  - Show unpredictable **diversity** of system configurations
  - Have **dynamic** core set membership
  - Provide only **limited shared** memory or cache **coherence**

$\Rightarrow$ Give the OS advanced reasoning techniques to make sense of the hardware and workload at runtime.

System knowledge base

- Fundamental operating system service
- Knowledge-representation framework
  - Database
  - RDF
  - Logic Programming and inference
  - Description Logics
  - Satisfiability Modulo Theories
  - Constraint Satisfaction
  - Optimization

What goes in?

1. Resource discovery
   – E.g. PCI enumeration, ACPI, CPUID…
2. Online hardware profiling
   – Inter-core all-pairs latency, cache measurements…
3. Operating system state
   – Locks, process placement, etc.
4. “Things we just know”
   – Assertions from data sheets, etc.

What is it used for?

- Name service and registry
- Locking/coordination service
- Device management
- Hardware configuration
- Spatial scheduling and thread placement
- Optimization for hardware platform
- Intra-machine routing

etc.

So what happened?

What happened?

- Barrelfish achieved some of its goals
  - Showed scalability, adaptability, support for heterogeneous machines
  - More work in the pipeline
- HPC people contacted us because, apparently, they wanted a new OS
  - We couldn’t understand why.
- Much of what we borrowed from supercomputing turned out to be of limited use.
  - Why?

General-purpose computing

$\neq$

Supercomputing

Systems@ETH Zürich

The hardware is different.

These are supercomputers.

Artistic case design!

Plenty of custom hardware!

Supercomputers don’t just look cool

- Supercomputers have cool hardware!
  - Message passing networks
  - In-network collection and reduction primitives
  - Fault-tolerance & partial failure
  - Vector units
  - Etc.

This is not a supercomputer.

This is not a supercomputer.

Systems@ETH Zürich

This is Facebook.

Neither is this.

Neither is this.

This is actually a Microsoft 40-foot shipping container

Not very glamorous case design.

These aren't supercomputers either

The software is different

This is not a supercomputing application.

Computationally intensive, highly parallelizable

- Vision and depth-cam processing
- Skeletal body tracking
- Facial feature and gesture recognition
- Audio beamforming
- Speech and phoneme recognition
- 3D mesh construction

These are also not supercomputing applications.

- Facebook
- Google
- Bing
- Second Life
- World of Warcraft
- Twitter
- Youtube
- etc.

General-purpose software is…

- Parallel (increasingly)
  - But complex, dynamic structure!
- Continuous
  - Long-running services
- Soft real-time
  - Bounded response time, interactivity
- Imprecise
  - Sometimes it's better to be wrong than late
- Bursty, dynamic, interactive
  - No clear execution cycle, load changes unexpectedly

Overall workload is different.

Workload assumptions

- General purpose OS target:
  - Many concurrent tasks
  - Diverse performance requirements
  - Unpredictable mix
  - Goal: satisfy SLAs and then optimize power, throughput, responsiveness, etc.
- Supercomputing:
  - Serial jobs. Complete each one ASAP.

Example:
how long should a thread spin?

Example: how long should a thread spin?

- Operating Systems answer:
  1. It depends

     (on the workload)
  2. The time taken to context switch

     (If you know nothing about the workload)

Example: how long should a thread spin?

- Operating Systems answer:
  1. It depends

     (on the workload)
  2. The time taken to context switch

     (If you know nothing about the workload)
- HPC Answer:
  - As long as it takes for something to happen.
  - Intel OpenMP default spinwait time: **200ms**

Example:
how long should a thread spin?

Systems@ETH zürich

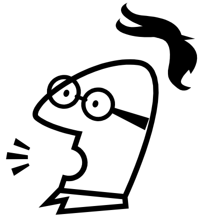
- Operating Systems answer:
  1. It depends

     (on the workload)
  2. The time taken to context switch

     (If you know nothing about the workload

600,000,000 cycles @ 3GHz!

- HPC Answer:
  - As long as it takes for something to happen.
  - Intel OpenMP default spinwait time: **200ms**



Consequences

1. Hardware optimization techniques not directly applicable

- Good performance $\Rightarrow$ careful use of hardware
  - Caches and memory hierarchy
  - Microarchitecture dependencies
  - Interconnect topology
- But:
  - Current hardware changes faster than software can
  - Commodity hardware already massively diverse
  - Dynamic sharing changes the problem

1. Hardware optimization techniques not directly applicable

Systems@ETH Zürich

- Good performance $\Rightarrow$ careful use of hardware
  - Caches and memory hierarchy
  - Microarchitecture dependencies
  - Interconnect topology
- But:
  - Current hardware changes faster than software can
  - Commodity hardware already massively diverse
  - Dynamic sharing changes the problem

$\Rightarrow$ Cannot tune OS or any other program to hardware at design time

1. Hardware optimization techniques not directly applicable

- Techniques *can* be used (and already are), but:
  - Can’t be baked into the software
  - Have to adapt dynamically to current hardware
    - We use SKB to optimize spatial placement, cache awareness, etc.
  - Must interact with the OS scheduler
    - Use Scheduler Activations, SKB state, user-level threads, etc.
    - Much ongoing research!

2. Benchmarks of limited use

- PARSEC-2, etc. are highly stylized
  - For good reason:
    highlight a range of execution patterns
  - Focus on performance of “simple” codes
  - Very little I/O
- Don’t stress OS (or even runtime)
- A general-purpose job mix would have:
  - Concurrent programs w/ diverse requirements
  - Multiple parallel tasks within a program
  - Copious I/O and asynchronicity

2. Benchmarks of limited use

- Still may be useful for
  - Characterizing *some* execution patterns
  - As synthetic load generators
  - Building blocks for larger workloads?

- Open question: how to benchmark general-purpose system software?
  - C.f. Avatar Kinect, etc.

3. Co-scheduling doesn't work (yet)

Systems@ETH zürich

- Almost nothing benefits from gang scheduling
  - Competitive spinning ⇒ backfilling makes more efficient use of the machine
  - If one app needs it ⇒ schedule with priority
  - More than one app ⇒ spatially partition
    or greedily schedule as best-effort
  - Only of benefit when compute phase ≈ context switch time
- Impact for turnaround time on one job is negligible.

3. Co-scheduling doesn't work (yet)

- Some kind of coordinated scheduling might be useful:
  - Multiple, parallel database joins
  - SMP virtual machines
- Needs to understand:
  - I/O operations
  - IPC
  - Etc.

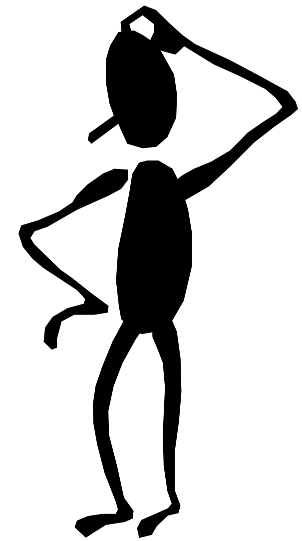
HPC folks were worried about OS "noise"

- Two problems:
  1. Message latency
  2. CPU "jitter"

- Message latency:
  - Custom MP hardware is rarely user-safe
  - Map device into user space (VIA, etc.)
  - More recent tricks: abuse SR-IOV!

CPU jitter

- CPU jitter is a spatial scheduling *non-problem*
  - At least in the OS research community
  - If you perform I/O, it's game over anyway
  - If you don't, your problem is caches and interrupts



- So, if you really want performance isolation:
  - Steer all your interrupts to different cores
  - Place applications to avoid cache crosstalk

Q. Why does no general-purpose OS do this?

- A. Nobody cares.
  - Plenty of tasks that you to run anyway
  - Applications aren’t sensitive to jitter
  - Most spend lots of time in the kernel
- However, Barrelfish can isolate applications…
  - Potentially useful for future applications
  - Investigate when Torsten Höfler arrives at ETHZ!

Q. Why does no general-purpose OS do this?

Q. Why does no general-purpose OS do this?

- A. Nobody cares.
  - Plenty of tasks that you to run anyway
  - Applications aren't sensitive to jitter
  - Most spend lots of time in the kernel

Q. Why does no general-purpose OS do this?

- A. Nobody cares.
  - Plenty of tasks that you to run anyway
  - Applications aren’t sensitive to jitter
  - Most spend lots of time in the kernel
- However, Barrelfish can isolate applications…
  - Potentially useful for future applications
  - Investigate when Torsten Höfler arrives at ETHZ!

4. Messaging hardware isn't useful (yet)

Systems@ETH zürich

- HPC-inspired proposals appearing for commodity hardware
  - E.g. Intel SCC message buffers
- Tailored to a single user
  - Can't be multiplexed efficiently
  - Requires kernel mediation for protection $\Rightarrow$ prohibitively expensive to use
- Tailored to a single application
  - Small, bounded buffers $\Rightarrow$ expensive flow control
  - Hard to context switch

4. Messaging hardware isn't useful (yet)

- Design of useful hardware support for general-purpose messages is an open research area
  - User-level multiplexing
  - Decoupling notification from delivery
  - Flow control and congestion avoidance
  - API design
- Many ideas from MPI, Blue Gene, etc. are highly relevant
  - But they require considerable changes!

Conclusion

- Supercomputing and OS research: Traditionally disjoint areas
  - Things are changing in both areas
  - Each side has ideas useful to the other
- Problems and assumptions remain very different
  - Cross-fertilization of fields is difficult (but interesting!)

Systems@ETH zürich

Open questions

- What ideas from supercomputing might be important to the design of general-purpose operating systems?

- Are there concepts and challenges from general-purpose operating systems which are becoming a concern in supercomputing?

Many thanks!