

# New directions in OS design: the Barrelfish research operating system

Timothy Roscoe (Mothy)

ETH Zurich

Switzerland

# Acknowledgements



- ETH Zurich:
  - Zachary Anderson, Pierre-Evariste Dagand, Stefan Kästle, Dominik Menzi, Simon Peter, Kaveh Razavi, Jan Rellermeier, Timothy Roscoe, Bram Scheidegger, Raffaele Sandrini, Adrian Schüpbach, Pravin Shinde, Dario Simone, Akhilesh Singhanian, Animesh Trivedi
- Microsoft Research:
  - Paul Barham, Andrew Baumann, Richard Black, Tim Harris, Orion Hodson, Rebecca Isaacs, Ross McIlroy, Vijayan Prabhakaran
- And many other friends and collaborators...

# This talk...



- Context
  - Trends in hardware
- Rethinking OS design
  - The Multikernel
  - The System Knowledge Base
- Barrelfish design
  - Overview
  - Messaging
  - Distributed algorithms
  - Configuring hardware
- Ongoing work, goals, and Conclusion

# CONTEXT

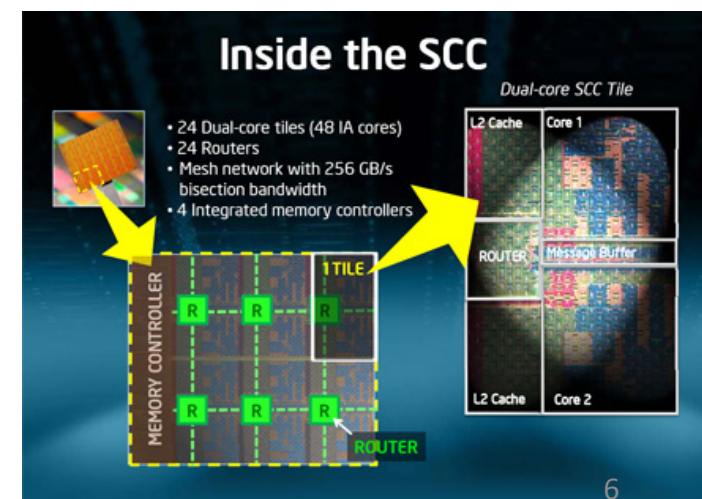
# A general purpose OS

- Mix of untrusted applications
- Soft real-time requirements for some
- Variety of hardware platforms



# Lots more cores per chip

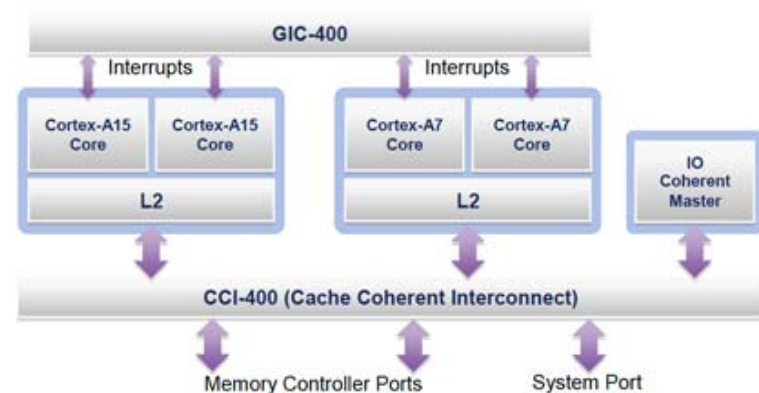
- Core counts now follow Moore's Law
- Cores will come and go
  - Energy!
- Diversity of system and processor configurations will grow
- Cache coherence may not scale to whole machine



# Cores will be heterogeneous

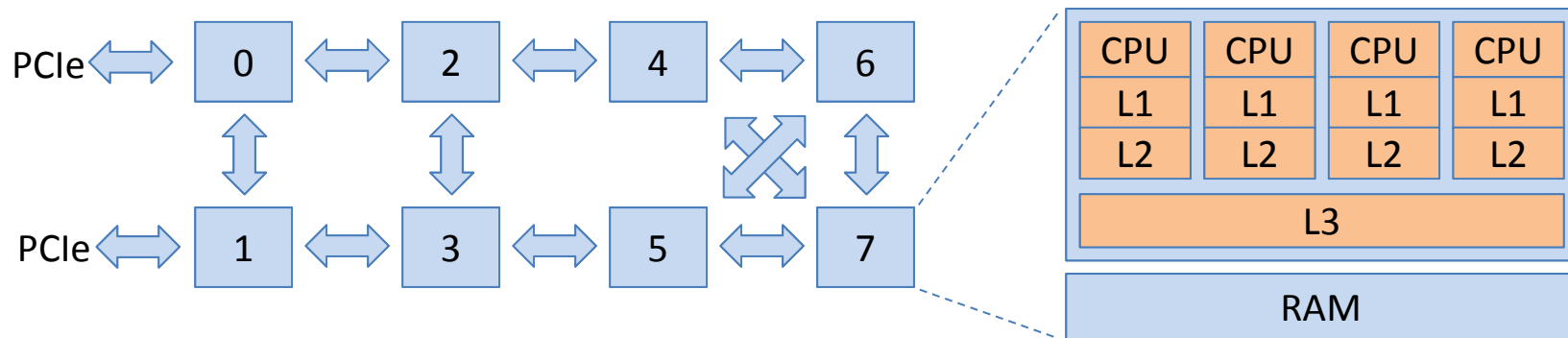


- NUMA is the norm today
- Heterogeneous cores for power reduction
- Integrated GPUs / Crypto / NPUs etc.
- Programmable peripherals



# 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**

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

# 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**

⇒ Forget about shared memory.

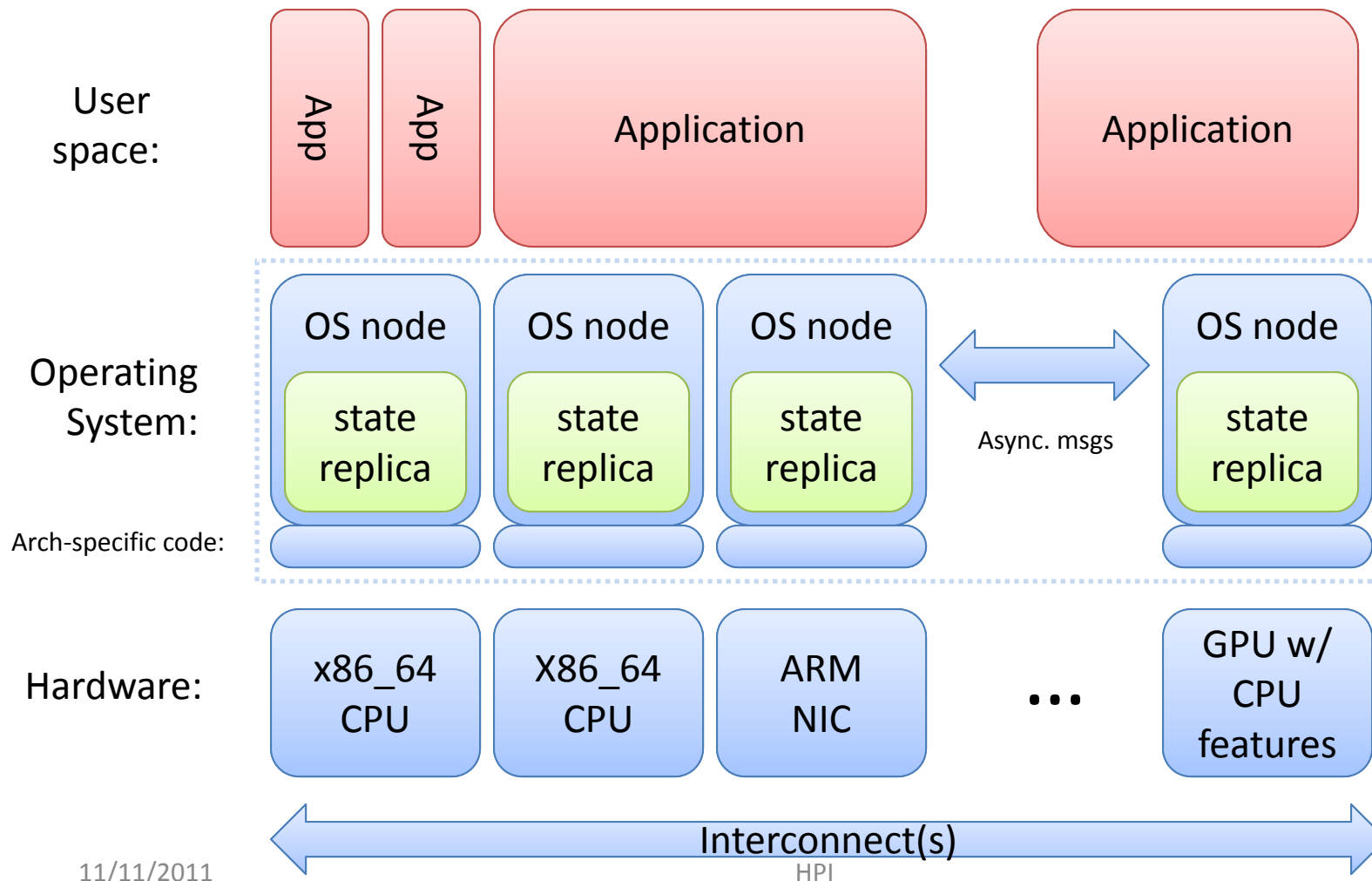
The OS is a distributed system based on **message passing**

# Multikernel principles

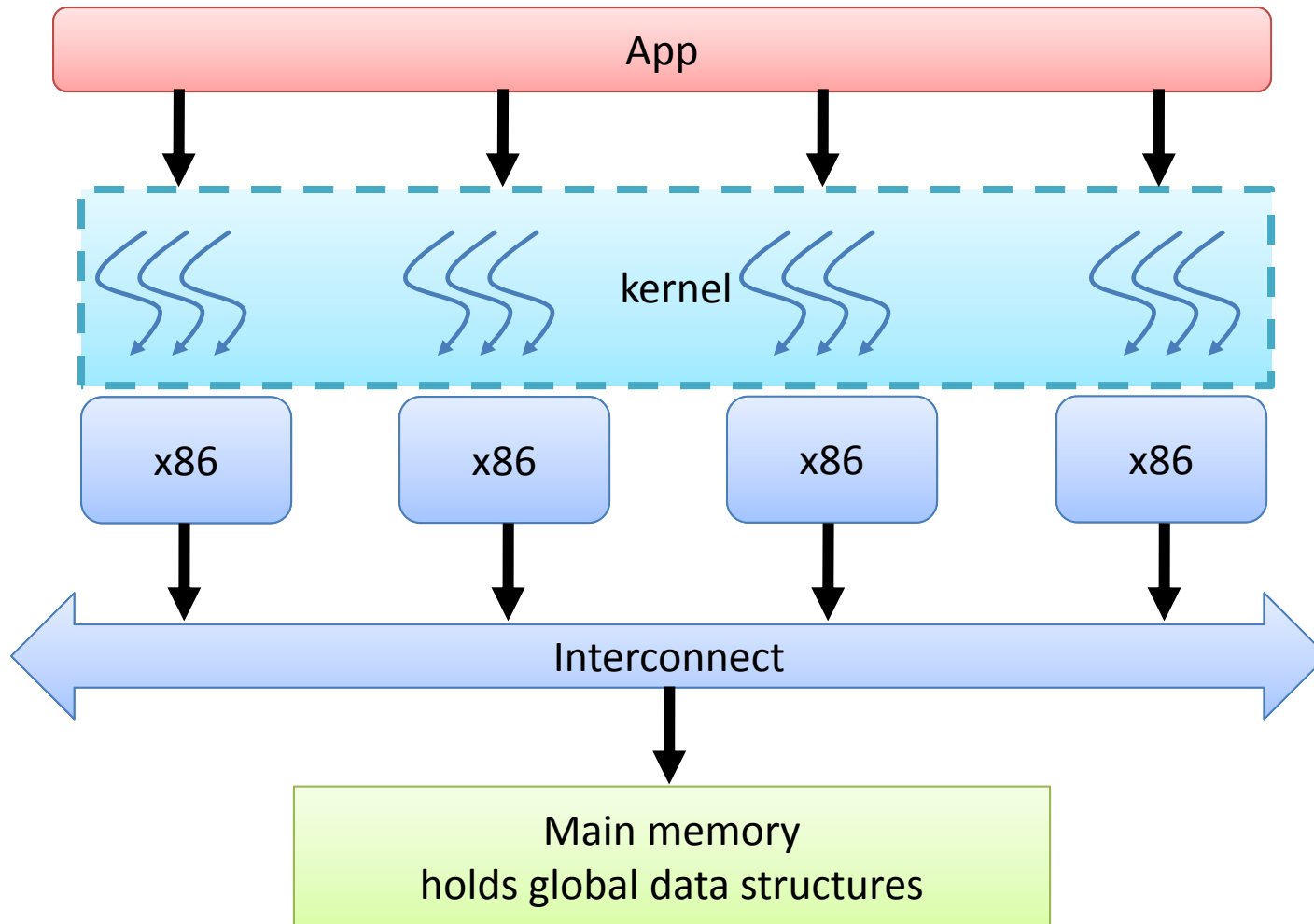


- Share **no data** between cores
  - All inter-core communication is via explicit messages
  - Each core can have its own implementation
- OS state **partitioned** if possible, **replicated** if not
  - State is accessed **as if** it were a local replica
- Invariants enforced by **distributed algorithms**,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

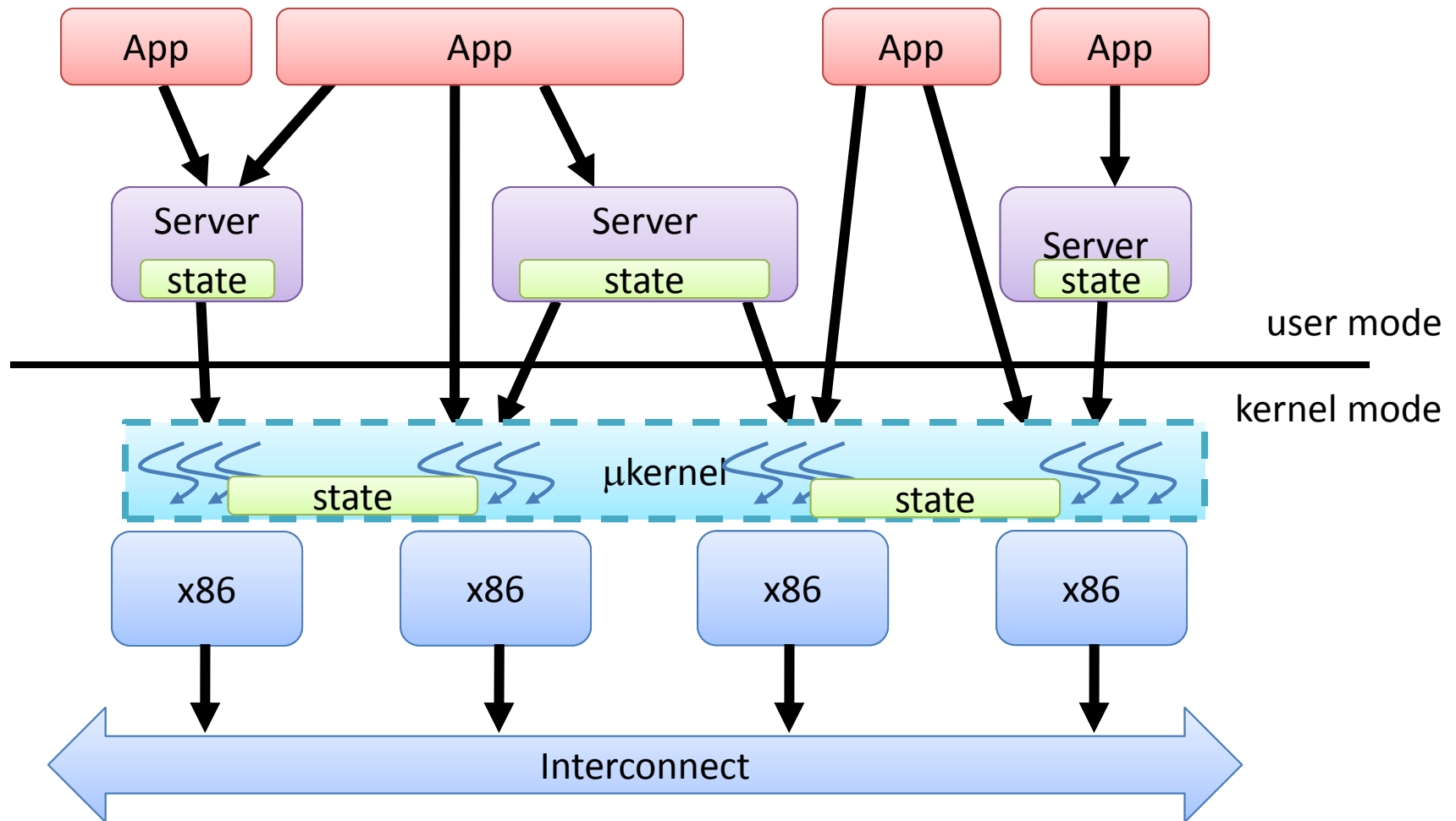
# The multikernel model



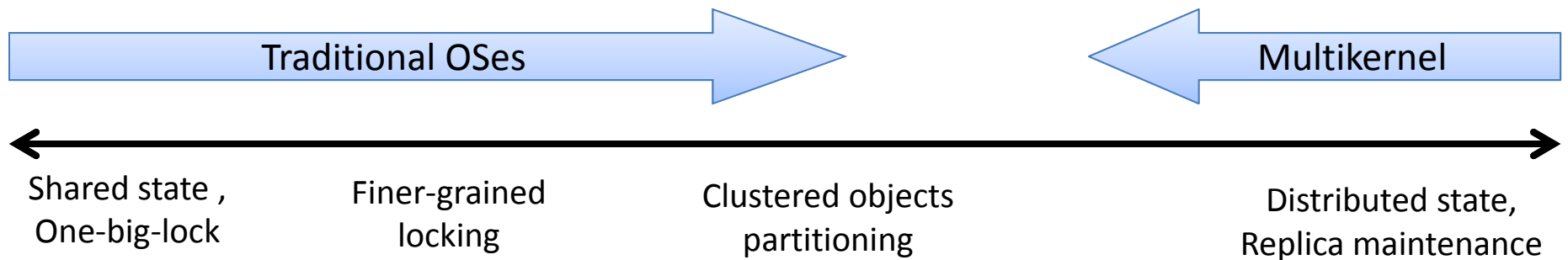
# ...vs a monolithic OS on multicore



# ...vs a $\mu$ kernel OS on multicore



# 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**

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

# 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 comes out?



OS and applications submit high-level queries:

- Relational-style queries
  - Logic programming
  - Satisfiability modulo theories
  - Linear, integer programming
  - Etc.
- SKB returns results for policies, optimization, etc.
    - Examples later

# BARRELFISH

# Barrelfish: our multikernel

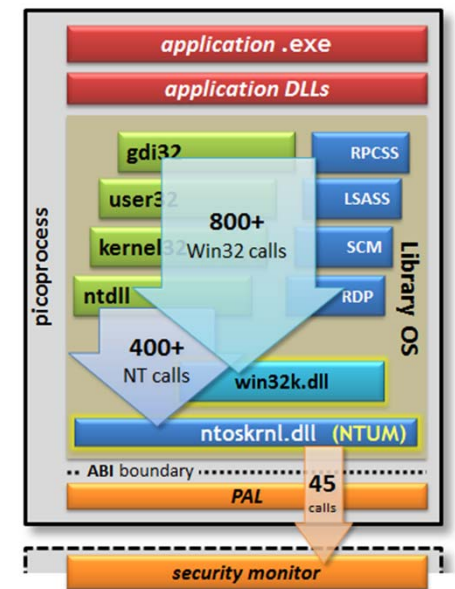


- ETH Zurich + Microsoft Research
- Currently supports:
  - 32- and 64-bit x86 AMD/Intel
  - Intel Single-chip Cloud Computer
  - Intel MIC (Knight's Ferry)
  - ARM (& Xscale)
  - Beehive (experimental softcore)
- Published 2009, available now
  - MIT open source licence, [www.barrelfish.org](http://www.barrelfish.org)

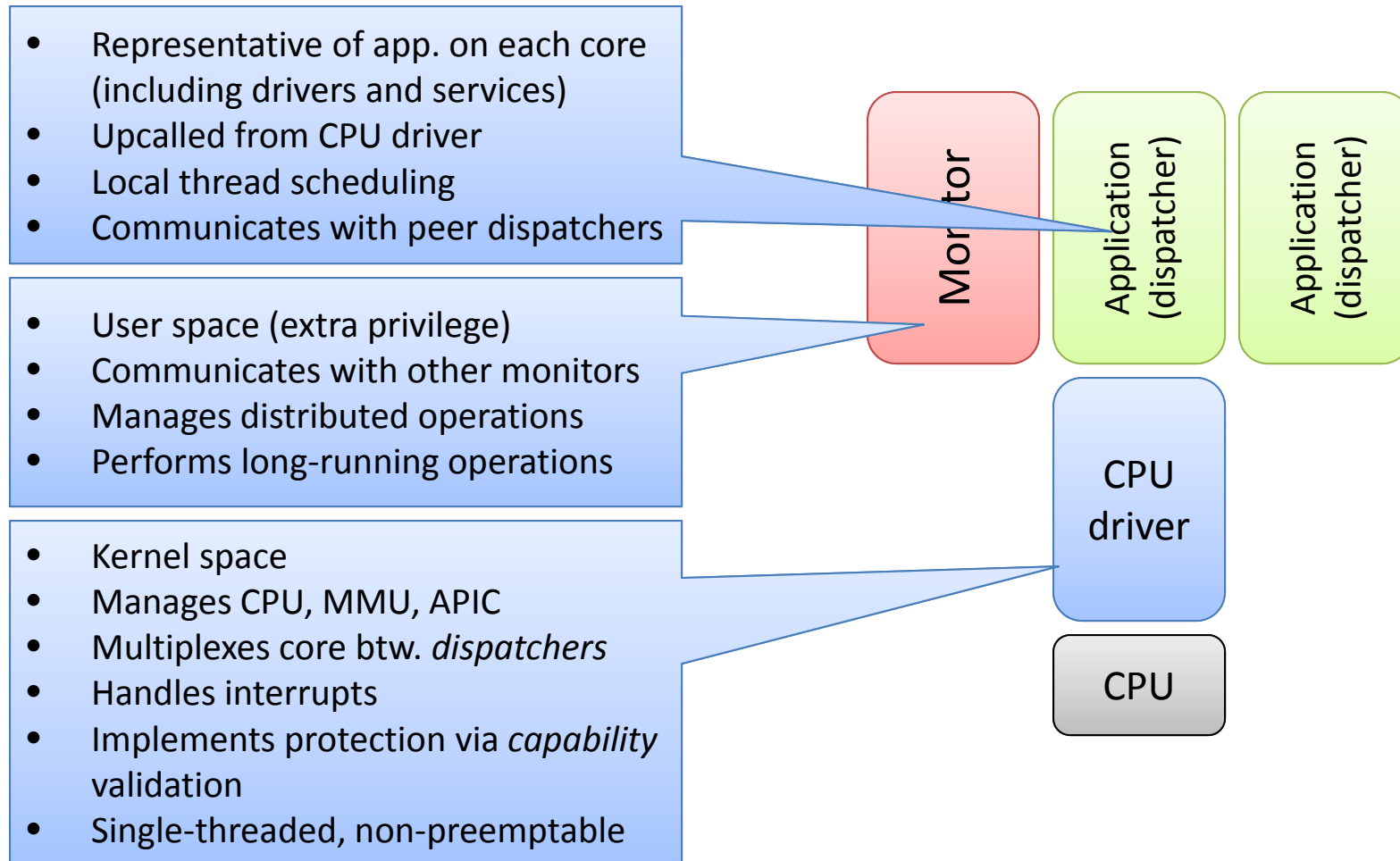


# What things run on it?

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



# Per-core architecture



# System Knowledge Base



- Port of popular ECLiPse CLP system
  - Constraint Logic Programming
    - not the IDE!
  - Very expressive (Prolog + constraints)
  - Quite slow – fairly old technology (vs., e.g. Z3)
  - But easy to hack for a prototype
- Starts very early in boot process
  - Initially runs from its own RAMdisk
  - Pulls more files from file system later
- Used for some surprising functions...



# 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 ( $\mu$ kernels, Xen)
- Specify device registers in a little language (Devil)

# EXAMPLE DESIGN CHALLENGES AND OPPORTUNITIES

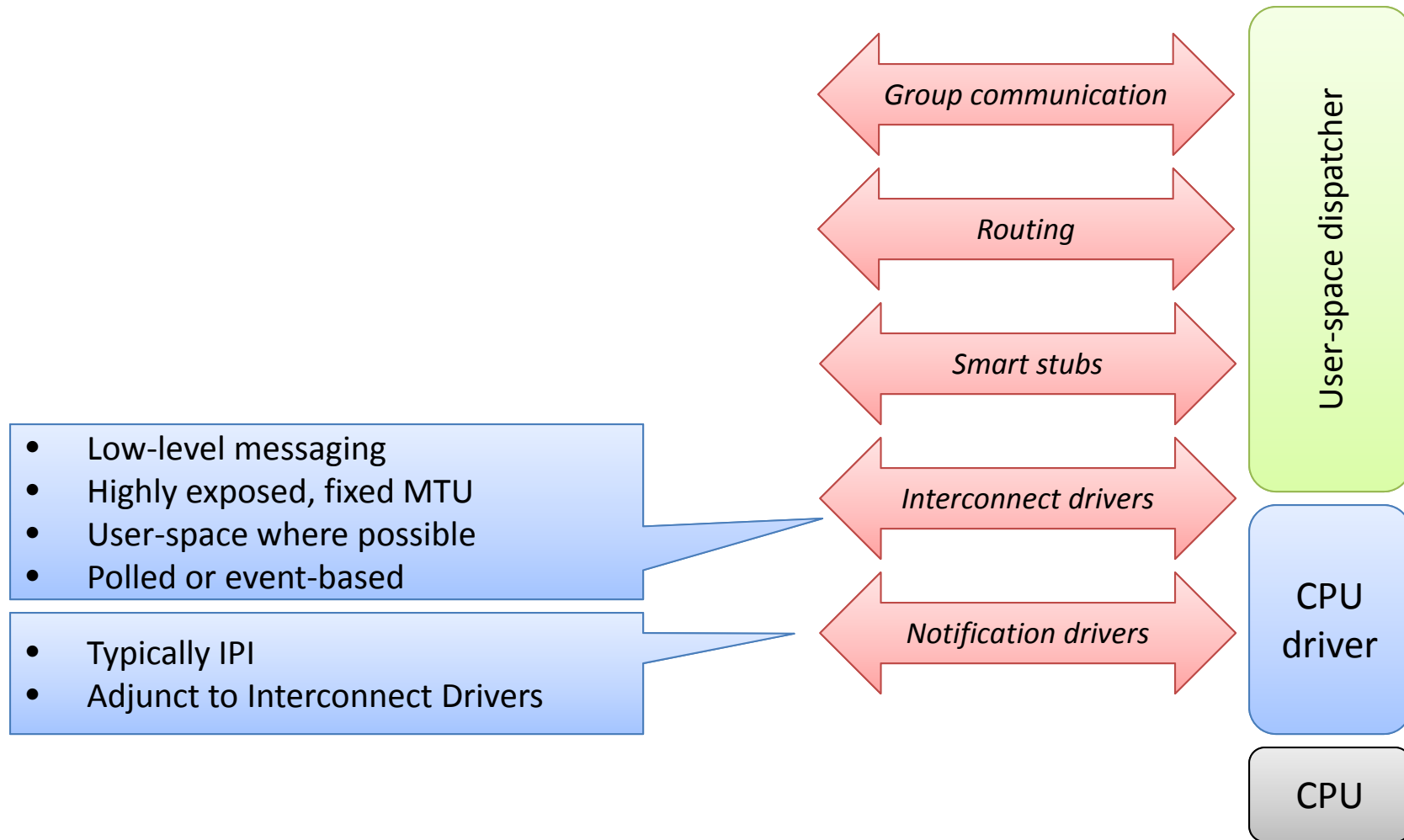
# Multikernel architecture + SKB

⇒ ??



- Messaging stack
- Fast messaging passing on diverse hardware
- Group communication in a machine
- Distributed algorithms
- Hardware configuration using CLP

# Communication stack



# Message-passing on x86

- LMP
  - Asynchronous
    - Deliver fixed-size message to the domain, and if necessary unblock it
    - Use shared-memory for more complex messages
  - Synchronous
    - Lightweight RPC (*Bershad et al, TOCS, 8(1), Feb 1990*)
- UMP
  - User-level RPC (*Bershad et al, TOCS, 9(2), May 1991*)
    - Use a region of memory as a channel
    - Transfer cache-line-sized messages (64 bytes)
  - Tailor to the cache-coherence protocol for good performance

# Other interconnect drivers



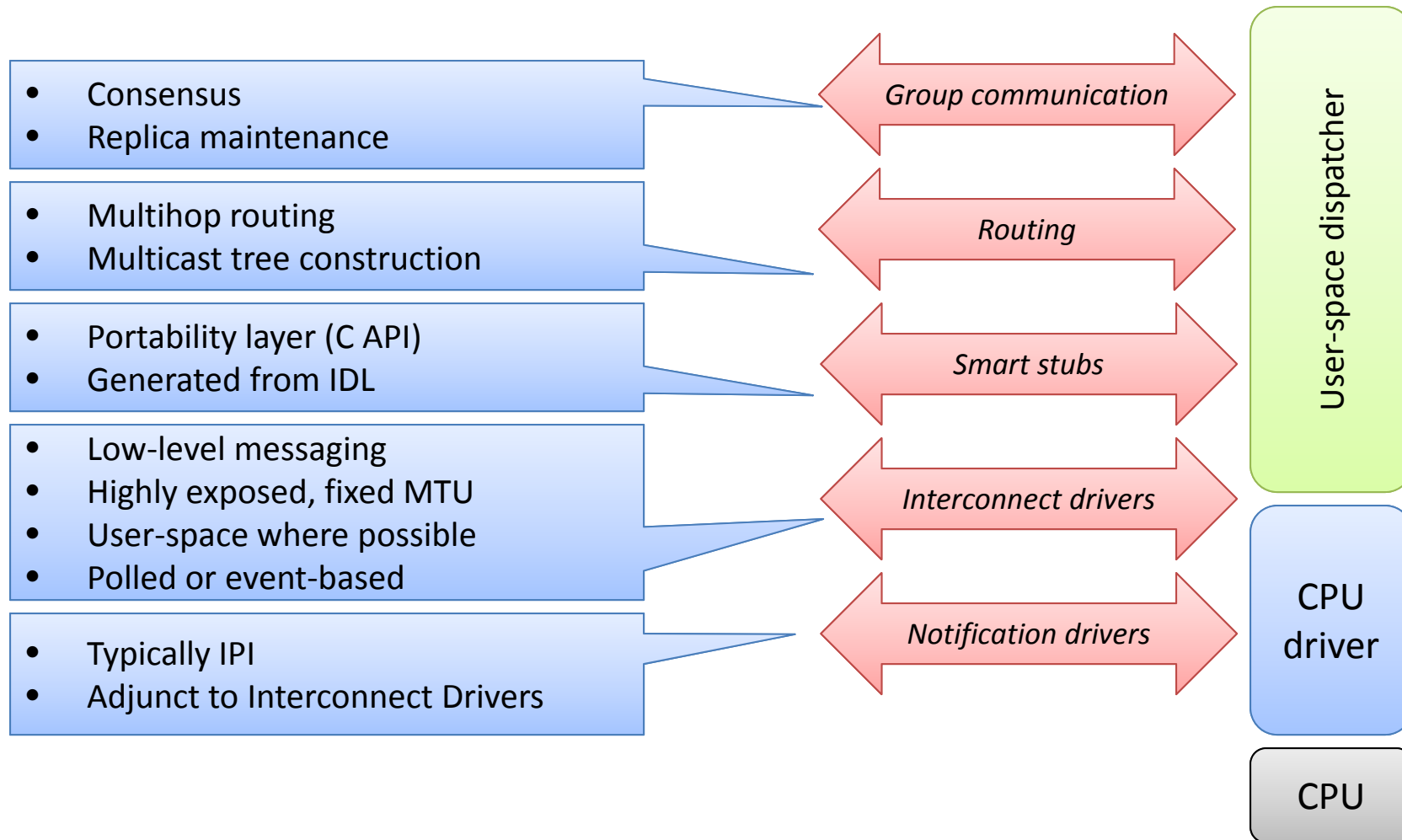
- LMP: L4-like single-core IPC
  - UMP: URPC-like shared-memory messaging
  - SCC: High performance tile memory
  - Beehive: per-core message FIFO
  - PCIe: x86  $\leftrightarrow$  SCC message passing
  - Tunnelling: multi-hop routed transport
  - Ethernet: cross-machine
- Etc.

# Message-passing



- Has to be very fast!
  - Specialize the implementation to the hardware
  - Select implementation at bind-time
- Must work between heterogeneous cores and across different machine architectures
  - Standardize the API
  - Generate code from an IDL (“Flounder”)

# Communication stack



# Examples of distributed algorithms in Barrelfish



- Keeping TLBs consistent
  - Each core holds a cache of virtual memory mappings, known as the TLB
  - Requires 1-phase commit
- Keep the capability database consistent
  - Virtual memory protection is enforced with capabilities
  - The capability database is replicated on every core
  - Requires 2-phase commit
- If cores can sleep (eg to save power), we may need a group membership protocol and more sophisticated consensus algorithms

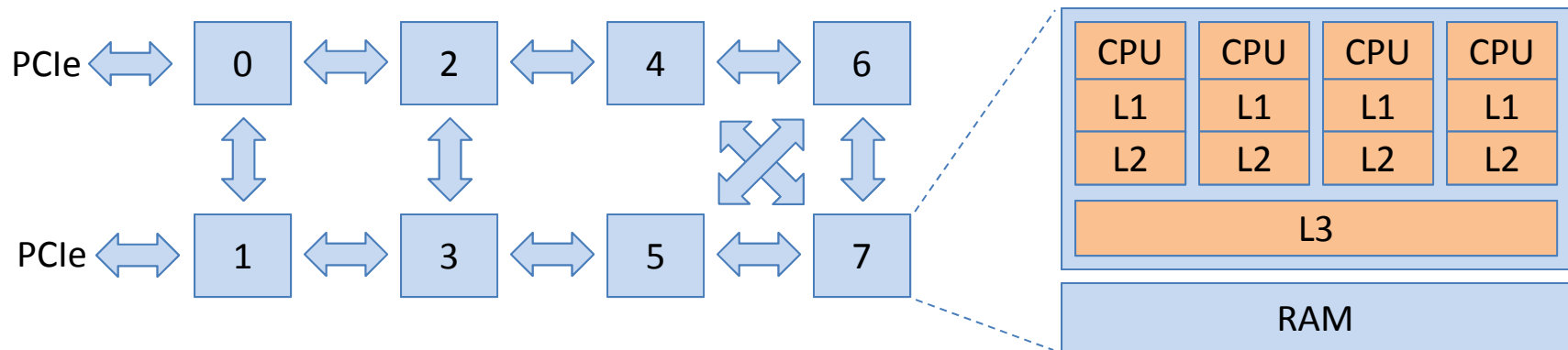
# TLB shutdown



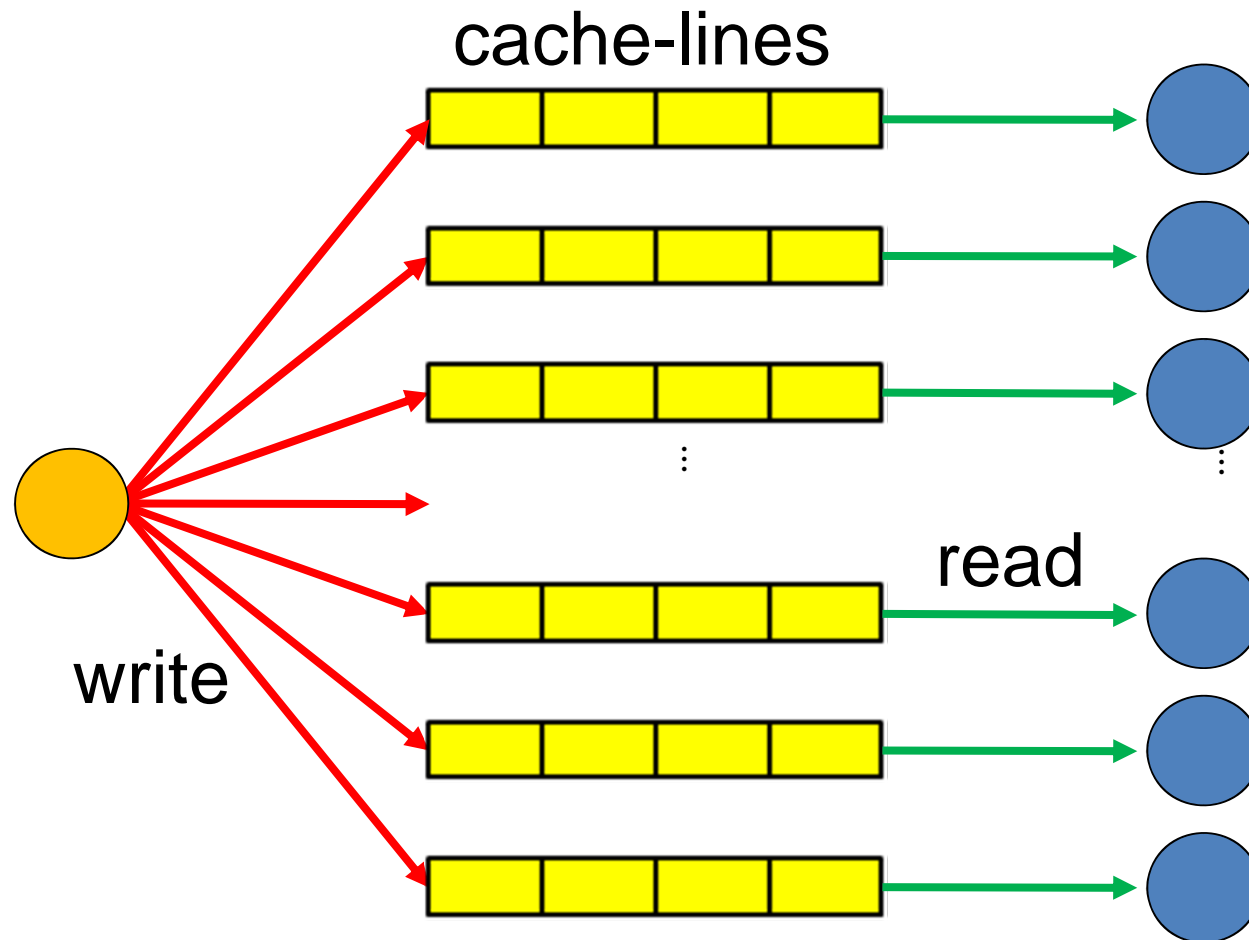
- When a mapping changes, the TLB must be flushed
  - On a multi-core machine, the TLB of every core that might contain the mapping must be flushed
- Requires global coordination (on every OS)
  - Send a message to every core with a mapping
  - Wait for acks (must be short!)
- Linux/Windows:
  - Send IPI (interprocessor interrupt)
  - Spin on shared ack count
- Barrelfish:
  - Monitor runs 1-phase commit protocol to remote cores
  - Can exploit knowledge of interconnect topology to improve performance

# Case study for TLB shutdown

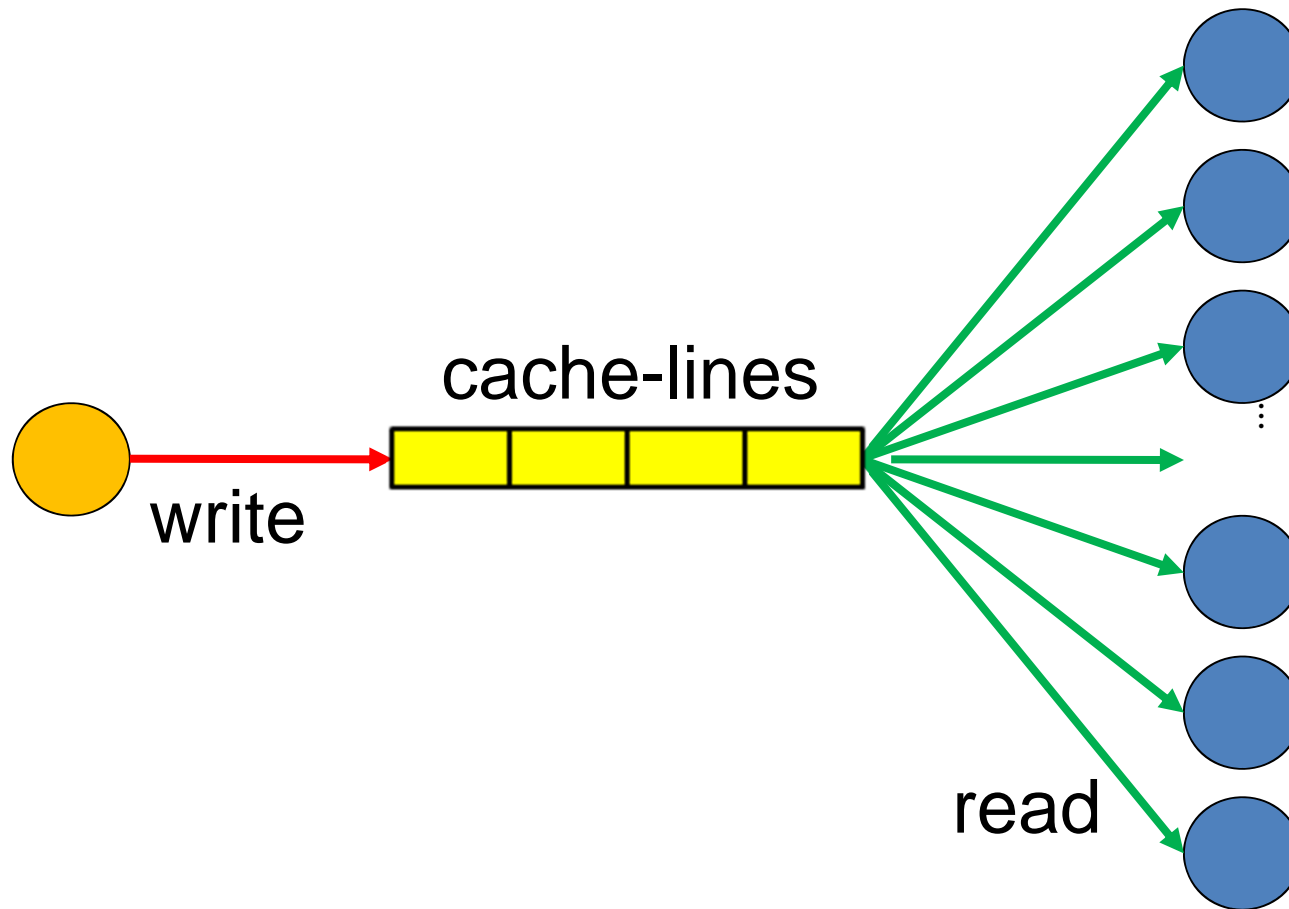
Hardware: 8 \* quad-core AMD Opteron



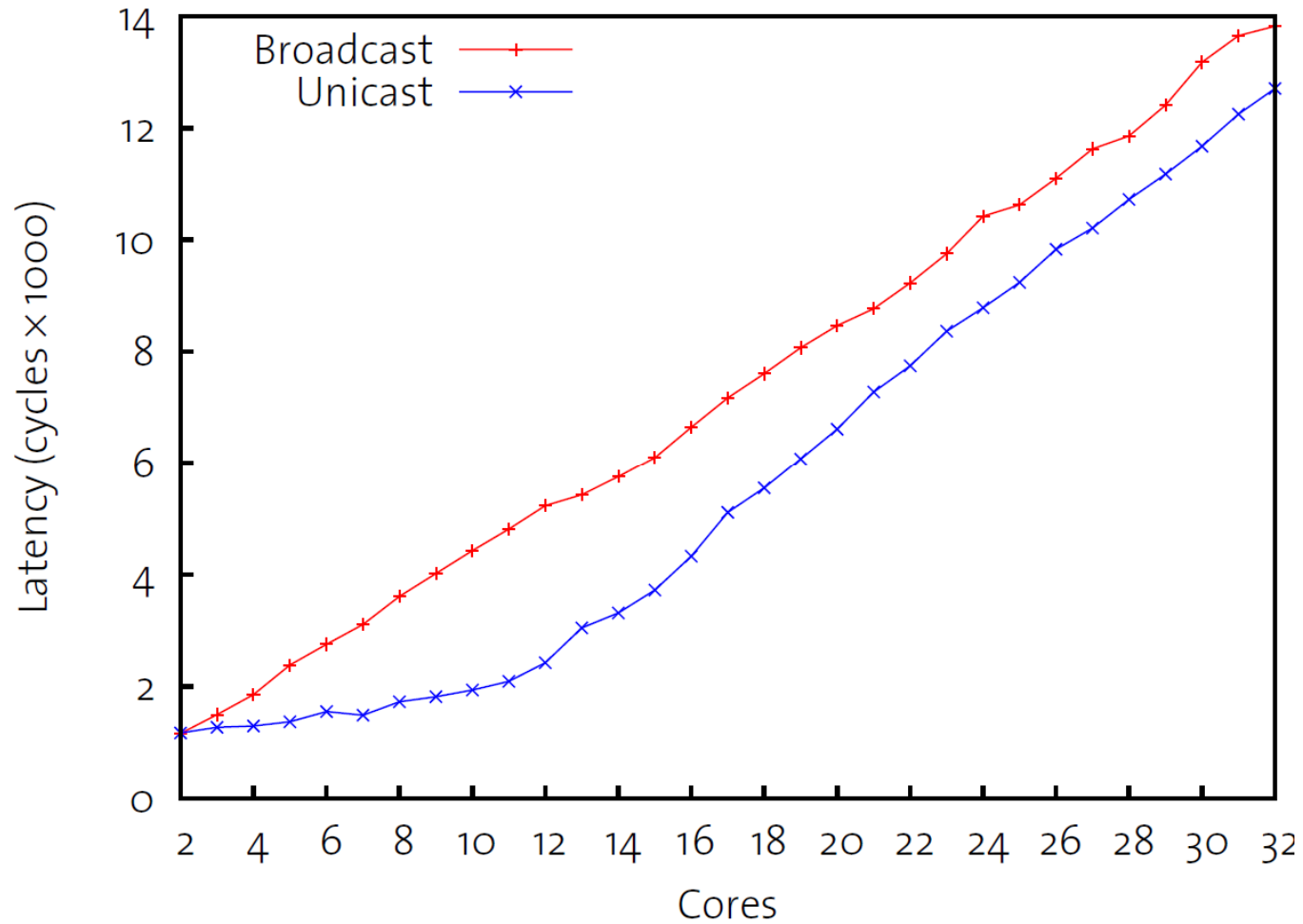
# TLB shutdown: $n * \text{unicast}$



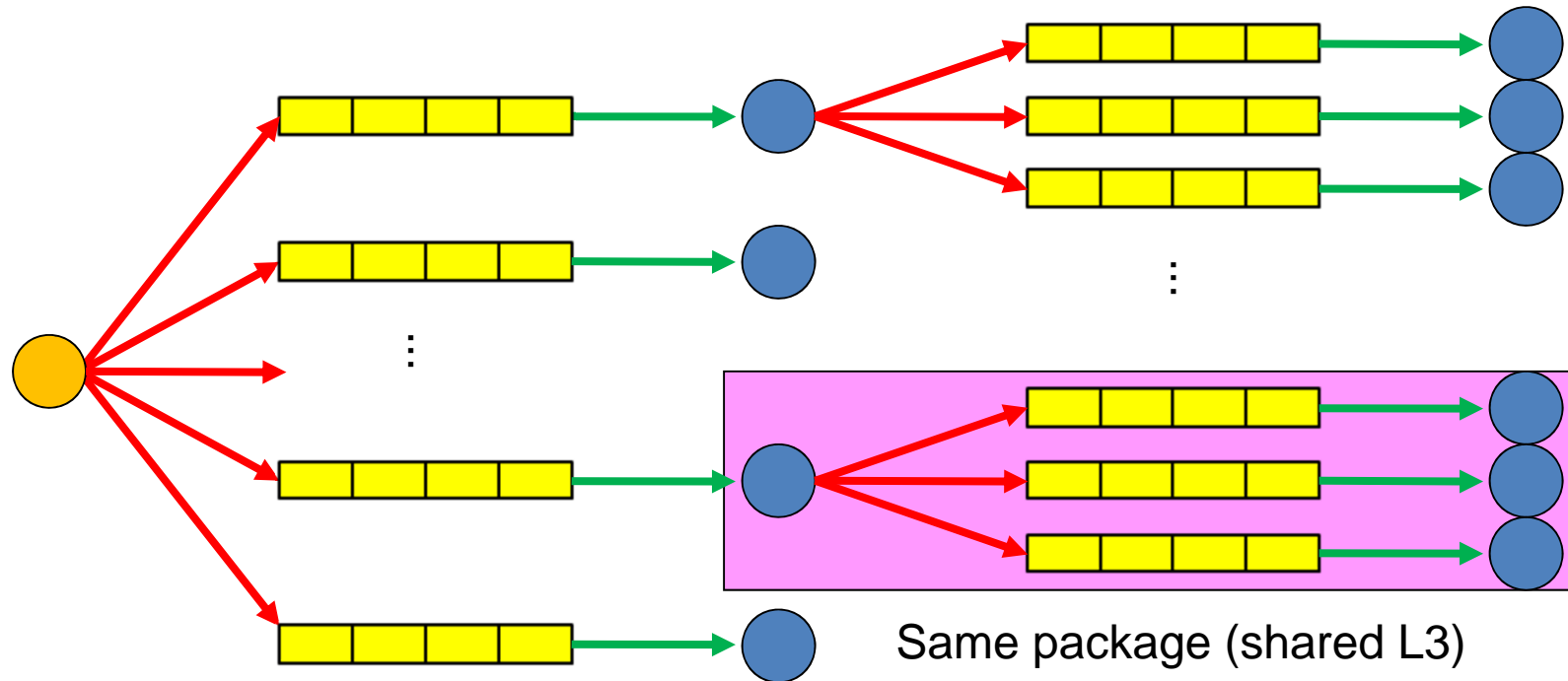
# TLB shutdown: 1\* broadcast



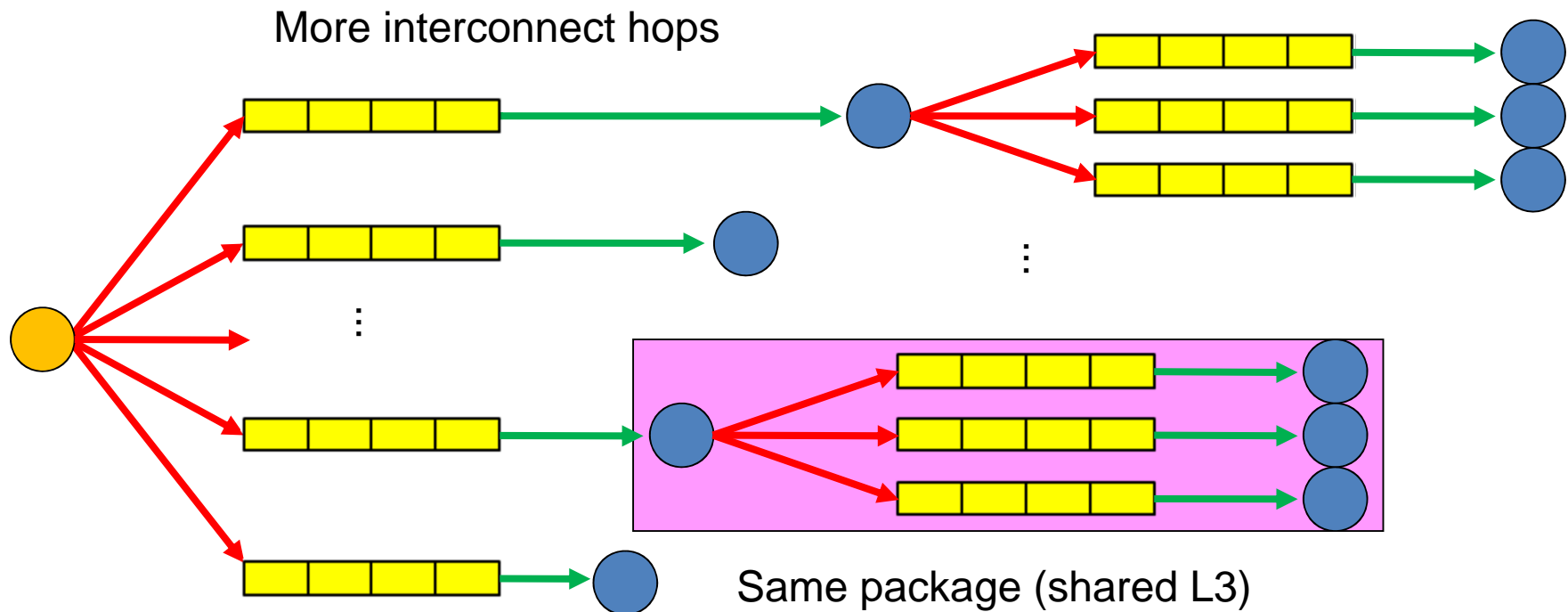
# Messaging costs



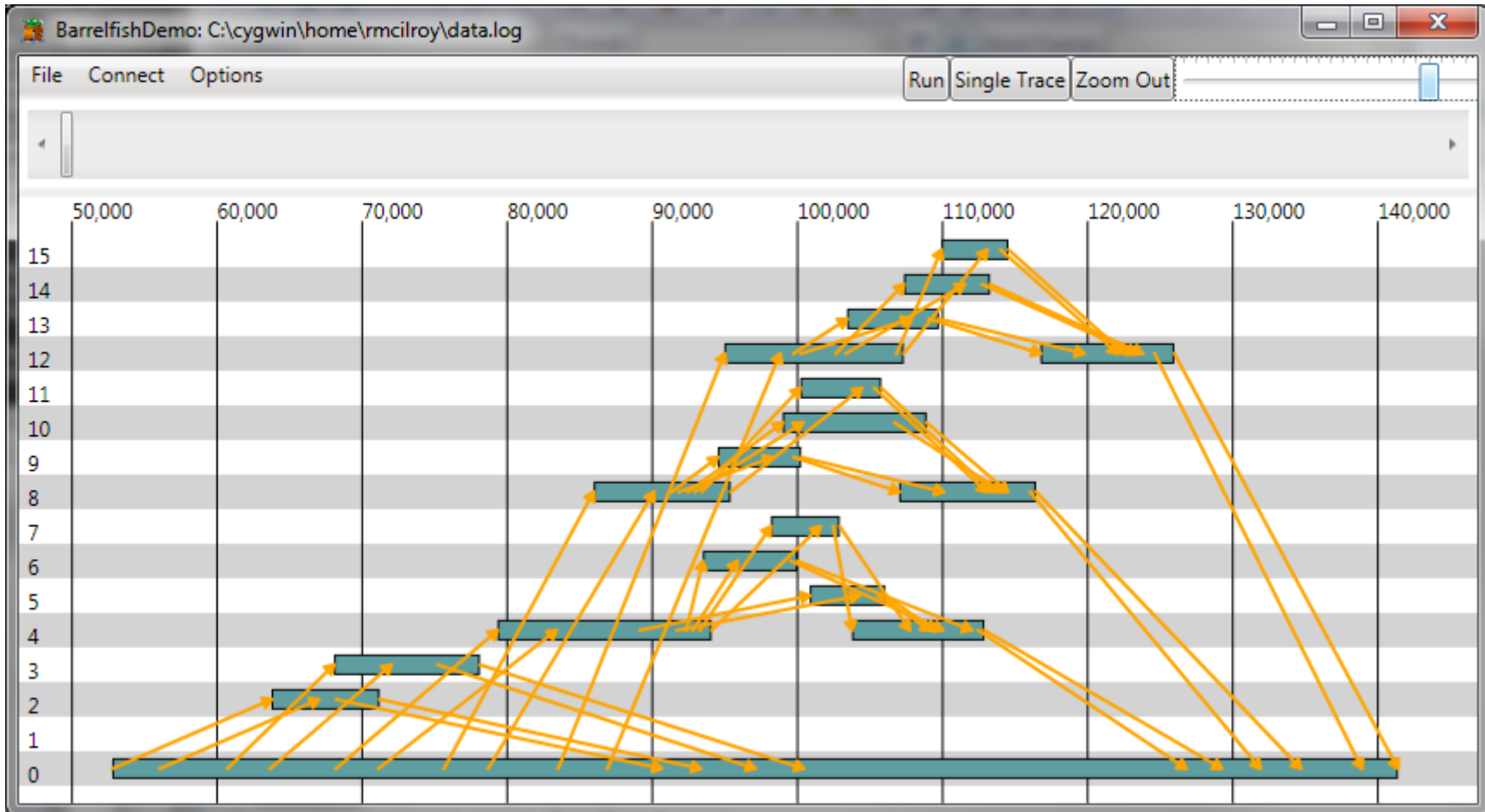
# TLB shutdown: multicast



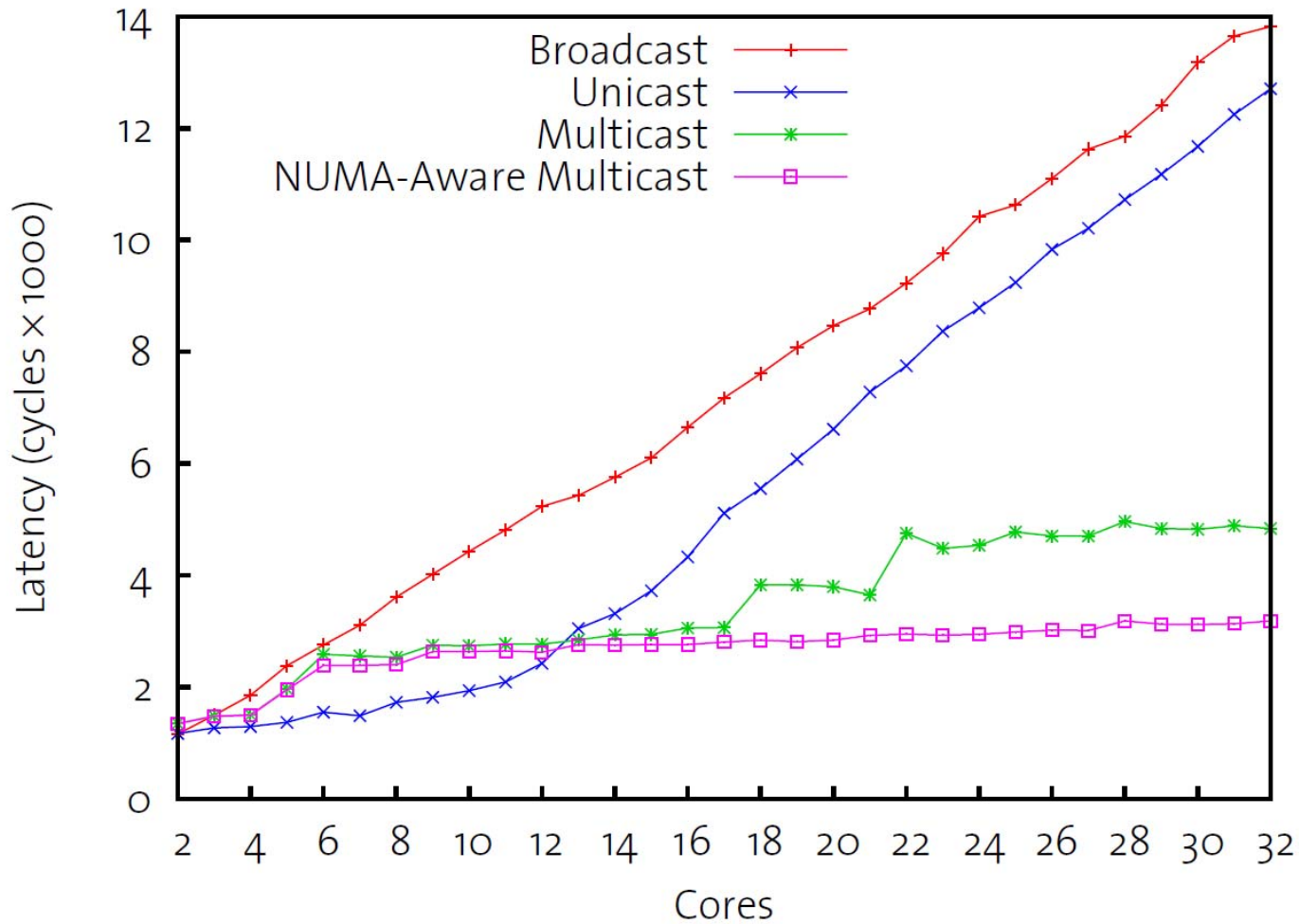
# TLB shutdown: NUMA-aware multicast



# Aggregation tree in action



# Messaging costs



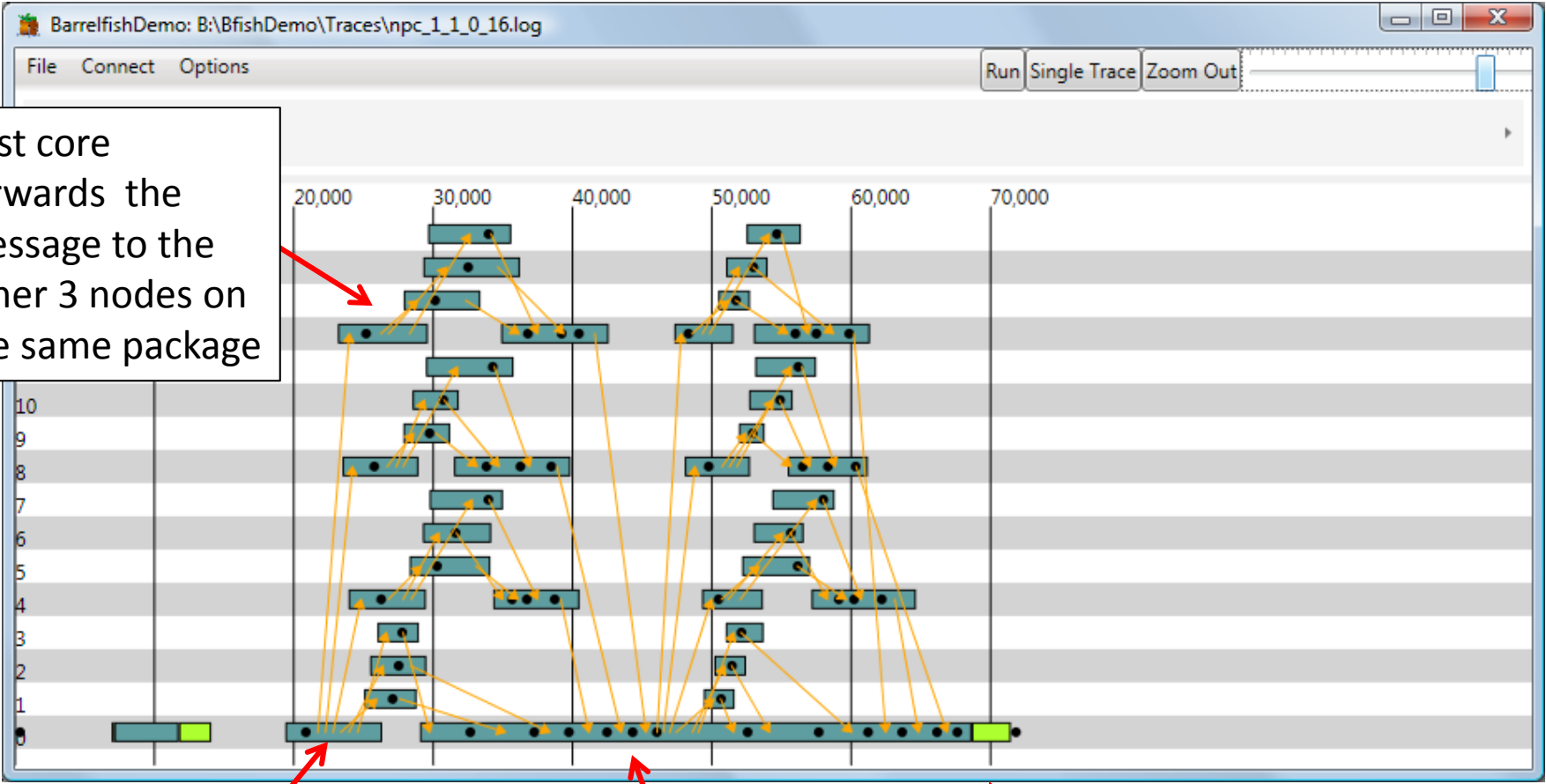
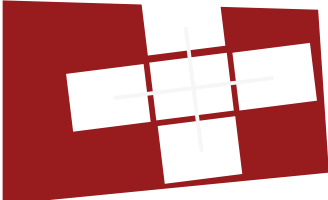
# SKB query to find the lowest-latency multicast tree



```
multicast_tree_cost(StartCore,[SendH|SendList], Cost) :-
    multicast_sanity_check,
    % determine package of start core
    cpu_thread(StartCore, StartPackage, _, _),
    % construct list of other packages
    findall(X, (cpu_thread(_,X,_,_), X =\= StartPackage), L),
    filter(L,PackageList),
    % compute possible links to those packages as SendList1
    sends(StartCore, PackageList, SendList1),
    % compute links from start core to its neighbours
    sendNeighbours(StartCore, Neighbours),
    append(SendList1, Neighbours, SendList2),
    % annotate with RTT of each link
    annotate_rtt(SendList2, SendList3),
    % sort by decreasing RTT
    sort(3, >=, SendList3, [SendH|SendList]),
    % determine cost as maximum single-link RTT
    % XXX: this is not quite right, we really care about maximum end-to-end RTT
    sendto(_,_,Cost) = SendH.

% goal to be called.
% Construct a list of sendto/3 goals, sort them by latency in decreasing order and
% minimize the value of the longest latency
multicast_tree(StartCore,SendList) :-
    minimize(multicast_tree_cost(StartCore, SendList, Cost), Cost).
```

# Trace: 2PC NUMA-aware multicast



First core forwards the message to the other 3 nodes on the same package

Core 0 sends a message to a core on each remote package

Gathers the replies

Total time ~70,000 cycles

# Applying distributed computing ideas to an OS



More subtle than it sounds...

- Algorithmic complexity typically measured in *rounds*
  - On a network, propagation time dominates
    - ⇒ maximise # messages in flight
- On a single machine, propagation time appears  $< 0!$ 
  - Rx and Tx overlap
  - Cost of Tx operation dominates
    - ⇒ minimize # back-to-back msg Tx/Rx operations
- Optimal algorithm may be different!
  - E.g. centralized agreement hits scaling wall much faster than on a LAN.

# And finally: PCIe programming: how hard can it be?

- Correct PCI bridge configuration turns out to be a nightmare!
    - Bridge/alignment constraints
    - Fixed devices
    - Devices that may or may not be PCI devices
    - Holes in physical address space
    - Quirks
    - Hardware bugs
    - Hotplugging of devices (including bridges)
- ⇒ almost no OS today does a proper job  
(amazing but true)



# Barrelfish: PCIe express configuration in Prolog!



- Barrelfish PCI programming:
  1. SKB boots before PCI
  2. PCI enumeration populates SKB with devices
  3. SKB solves PCI config as a constraint satisfaction problem
  4. PCI driver programs bridges and devices with BAR values
  5. Incremental algorithm can handle hotplug
- Handles all the weird corner cases
- Exceptions are expressed independently of the main algorithm
  - Most hardware bugs require a one-liner
- Can optimize for free space (e.g. for HotPlug)
- Details: See Adrian Schüpbach et.al., ASPLOS 2011, ACM TOCS 2012

# Wider insight



OS kernels use data structures for two purposes:

1. Traversal on fast path to route data and control correctly  
⇒ Must be small, efficient, fast, specialized to architecture, etc.
2. Traversal for policy, resource allocation, etc.  
⇒ Must be expressive, generic, self-describing, extensible, etc.

SKB  
each

*We argue something like this is essential for dealing with ever-more diverse and complex hardware*

zed to

# CONCLUSION

# Ongoing work (selection)



- Outside the box
  - Barrelfish over a rack
- Storage system design
  - Direct mapped PCM?
  - Support for databases and file systems
- Network stack
  - Model machine as distributed router
- Languages for message-based systems programming
  - THC: asynchronous extensions to C (Tim Harris)

# The Big Goal

- Key ideas:
  - Multikernel: the machine is a distributed system
  - System Knowledge Base: use decent reasoning online to guide the OS
- So far: great source of research directions!
- The Big Goal: a new platform for OS Research.
  - Flexible architecture
  - Less historical baggage
  - Well-matched to diverse future hardware
  - Plenty of research opportunities



# Many thanks for listening!

- Interested? Please join us!

<http://www.barrelfish.org/>

