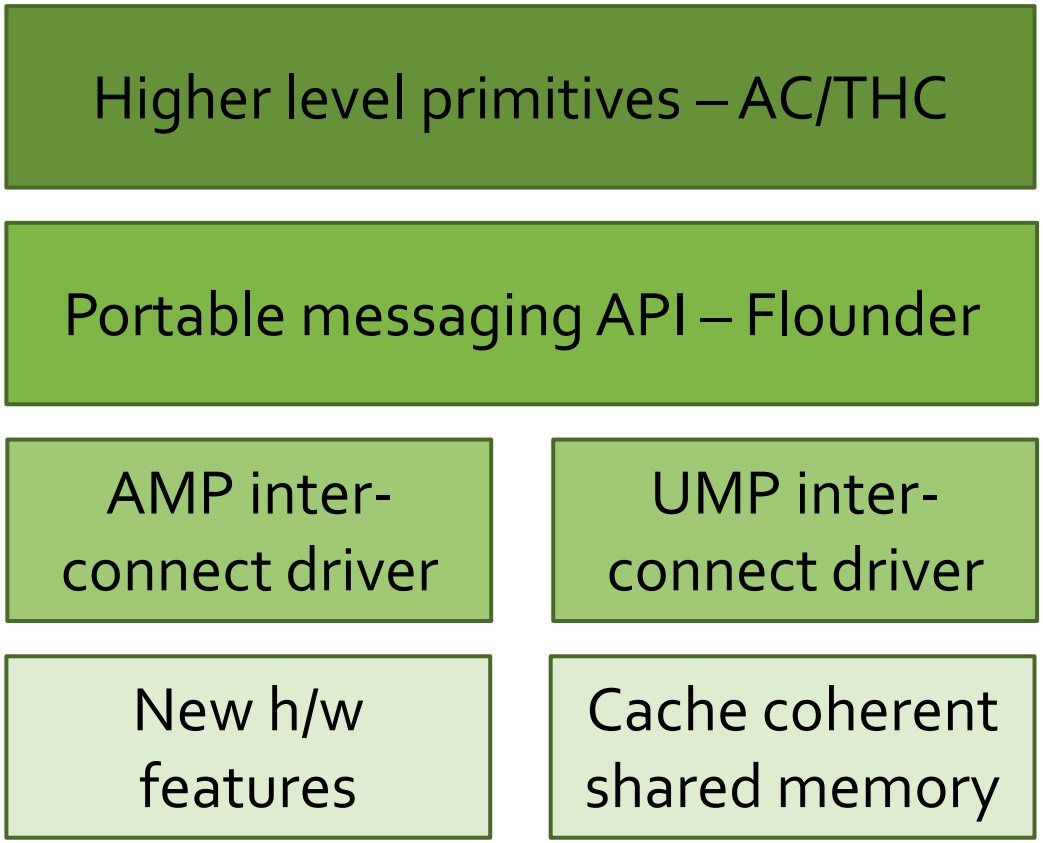
Microsoft Research

HARDWARE SUPPORT FOR MESSAGE PASSING IN THE BARRELFISH OPERATING SYSTEM

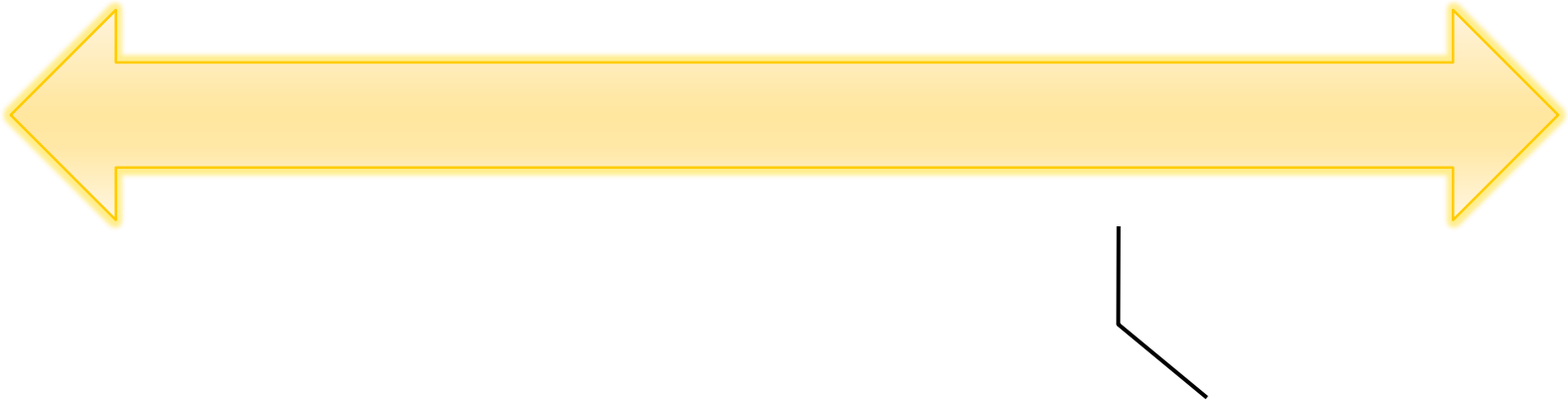
Richard Black, Vladimir Gajinov, Tim Harris, Ross McIlroy

Microsoft

Overview

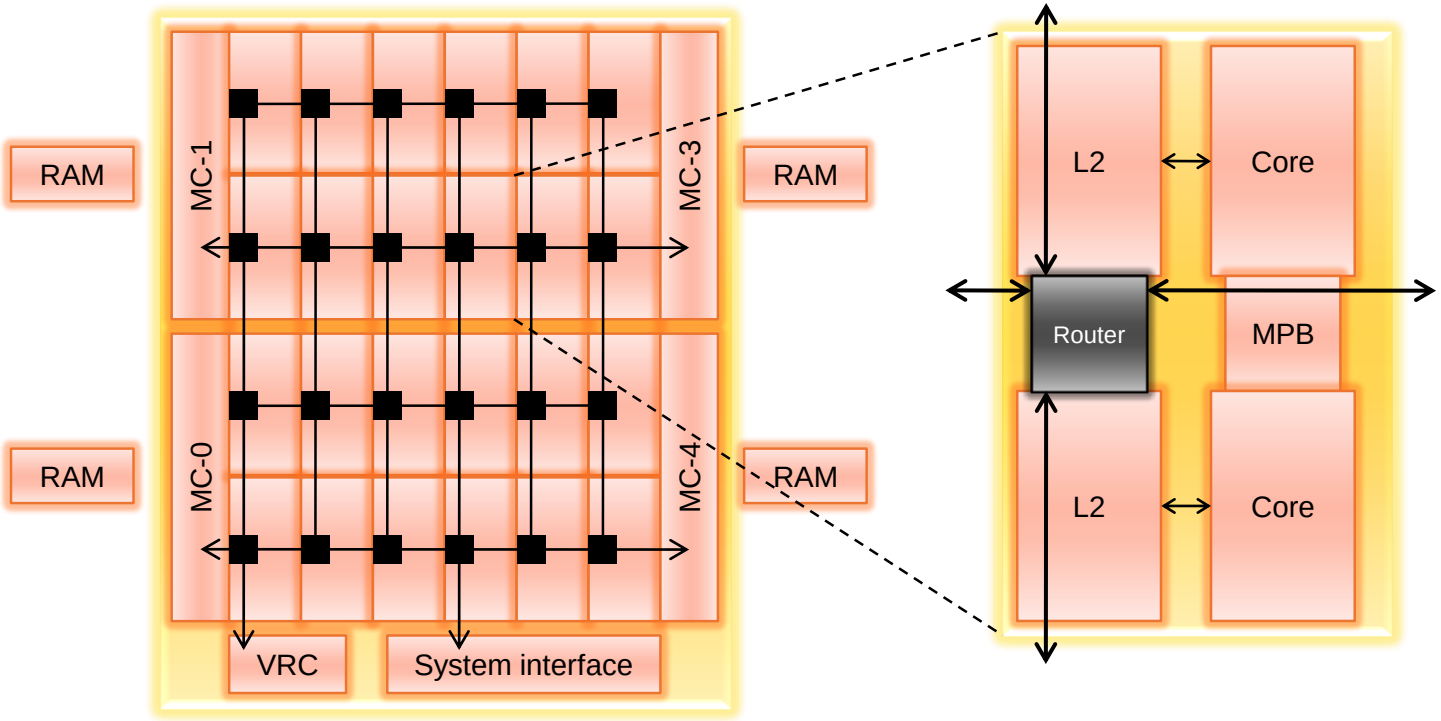

Higher level primitives – AC/THC
Portable messaging API – Flounder
AMP inter-connect driver
UMP inter-connect driver
New h/w features
Cache coherent shared memory

Prior work



HW approaches:
Generally – resources are per-core, not per process.
Communication within processes, not between them.

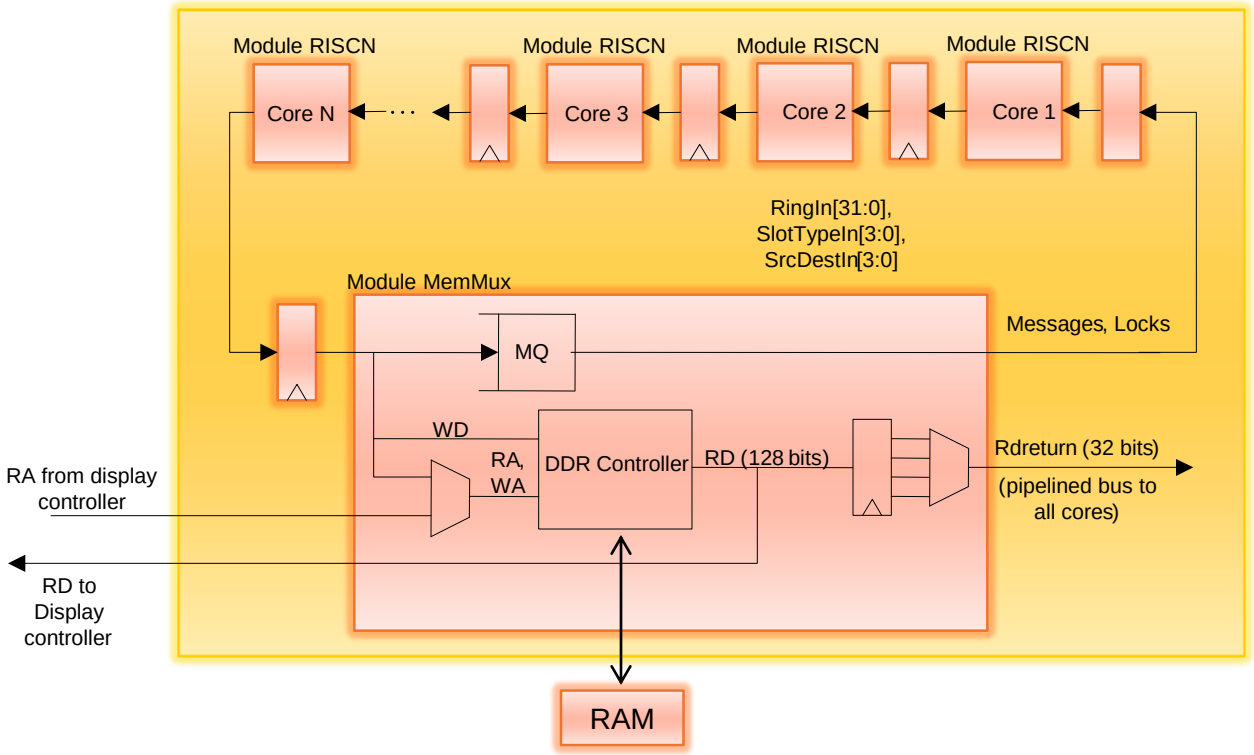
Prior work: SCC



24 * 2-core tiles
On-chip mesh n/w

Non-coherent caches
Message passing buffer

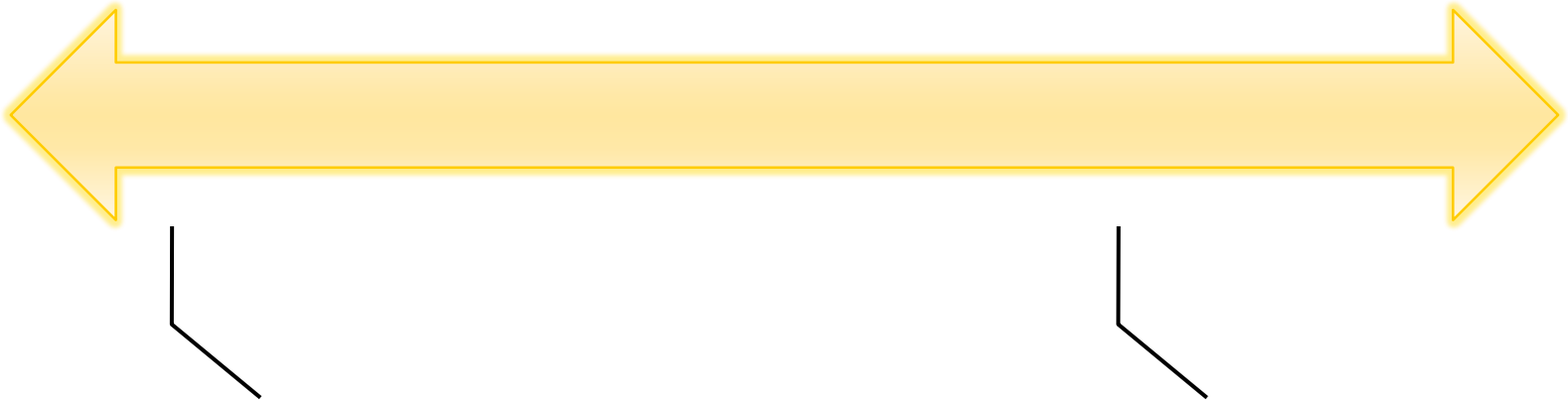
Prior work: MSR Beehive



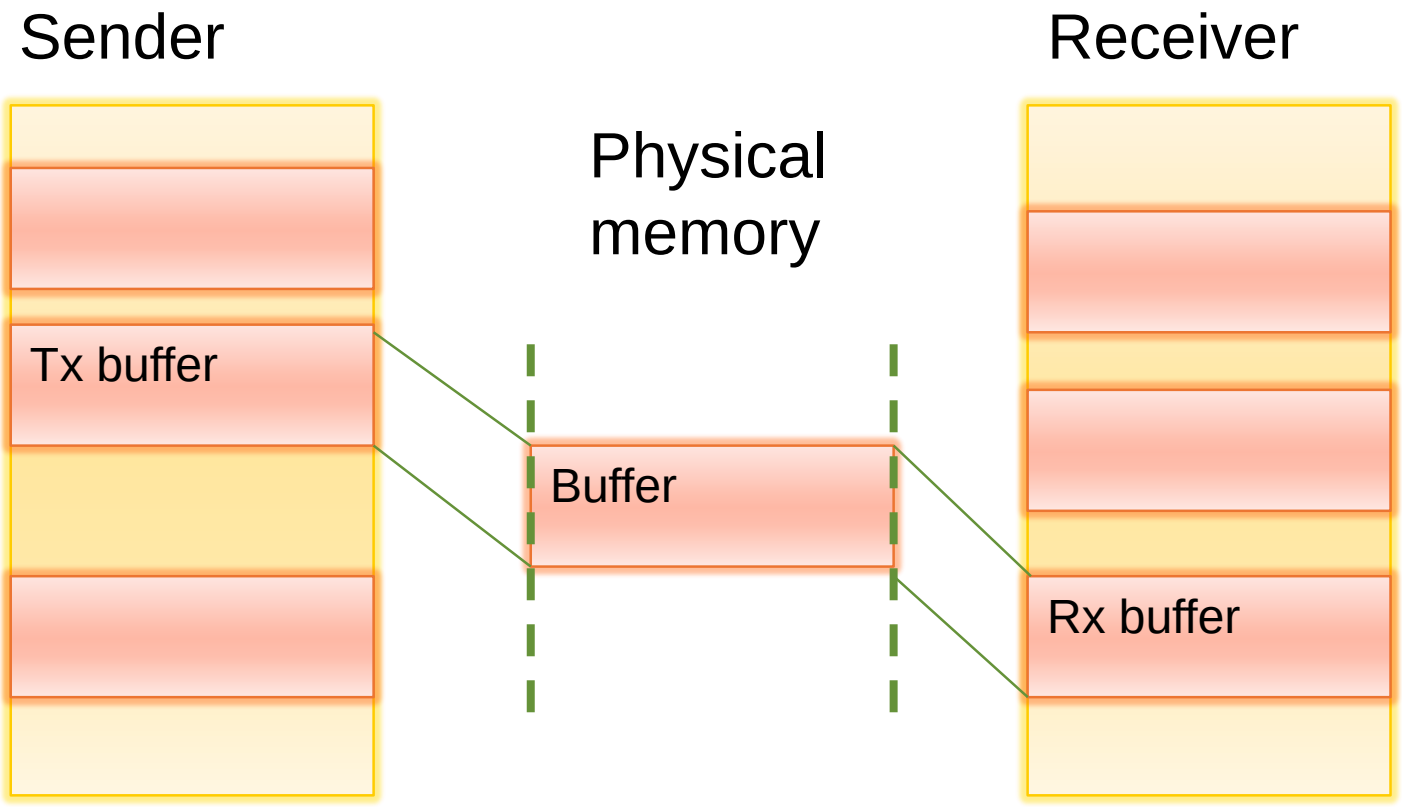
Ring interconnect
Message passing in h/w

No cache coherence
Split-phase memory access

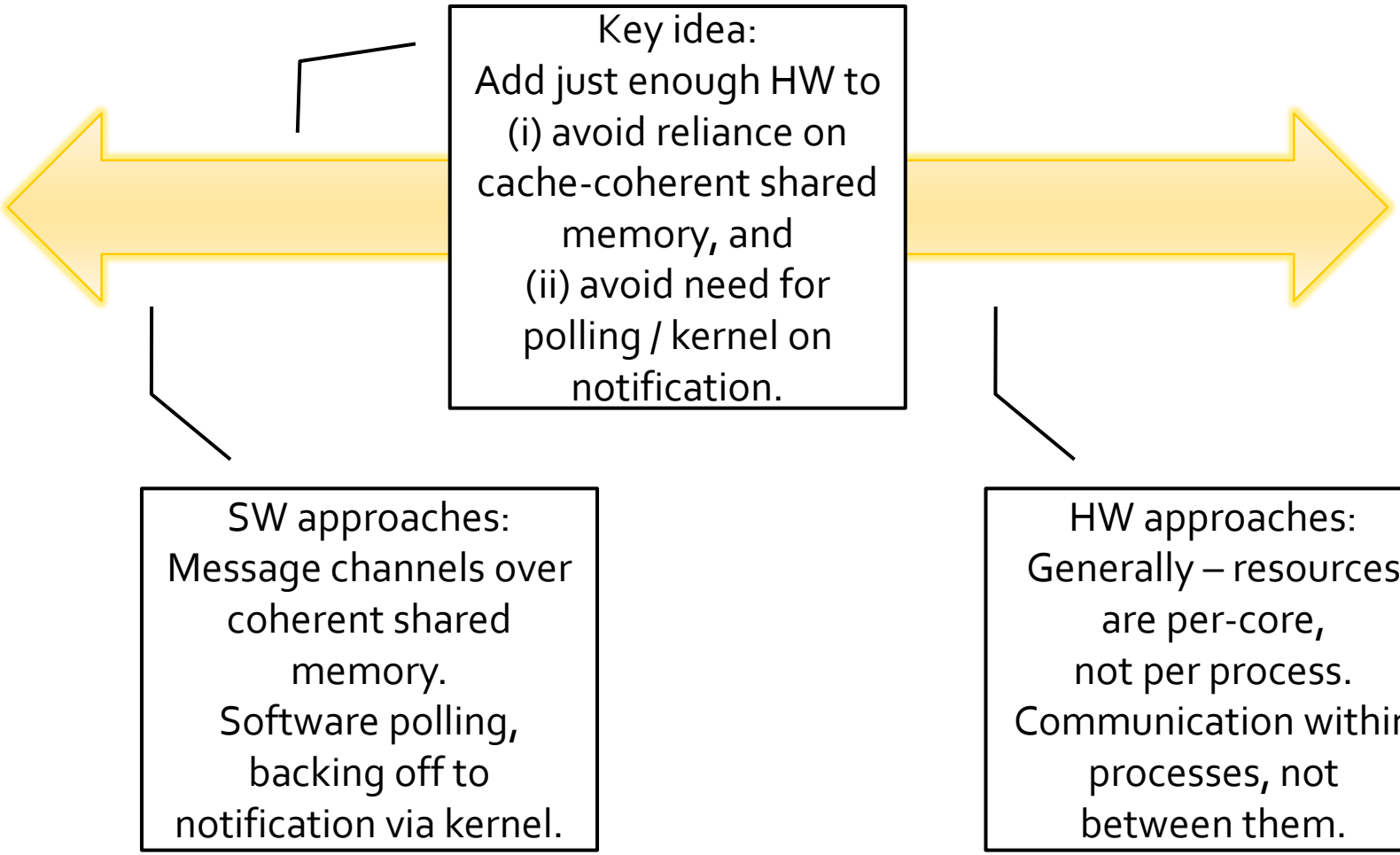
Prior work


SW approaches:
Message channels over coherent shared memory.
Software polling, backing off to notification via kernel.
HW approaches:
Generally – resources are per-core, not per process.
Communication within processes, not between them.

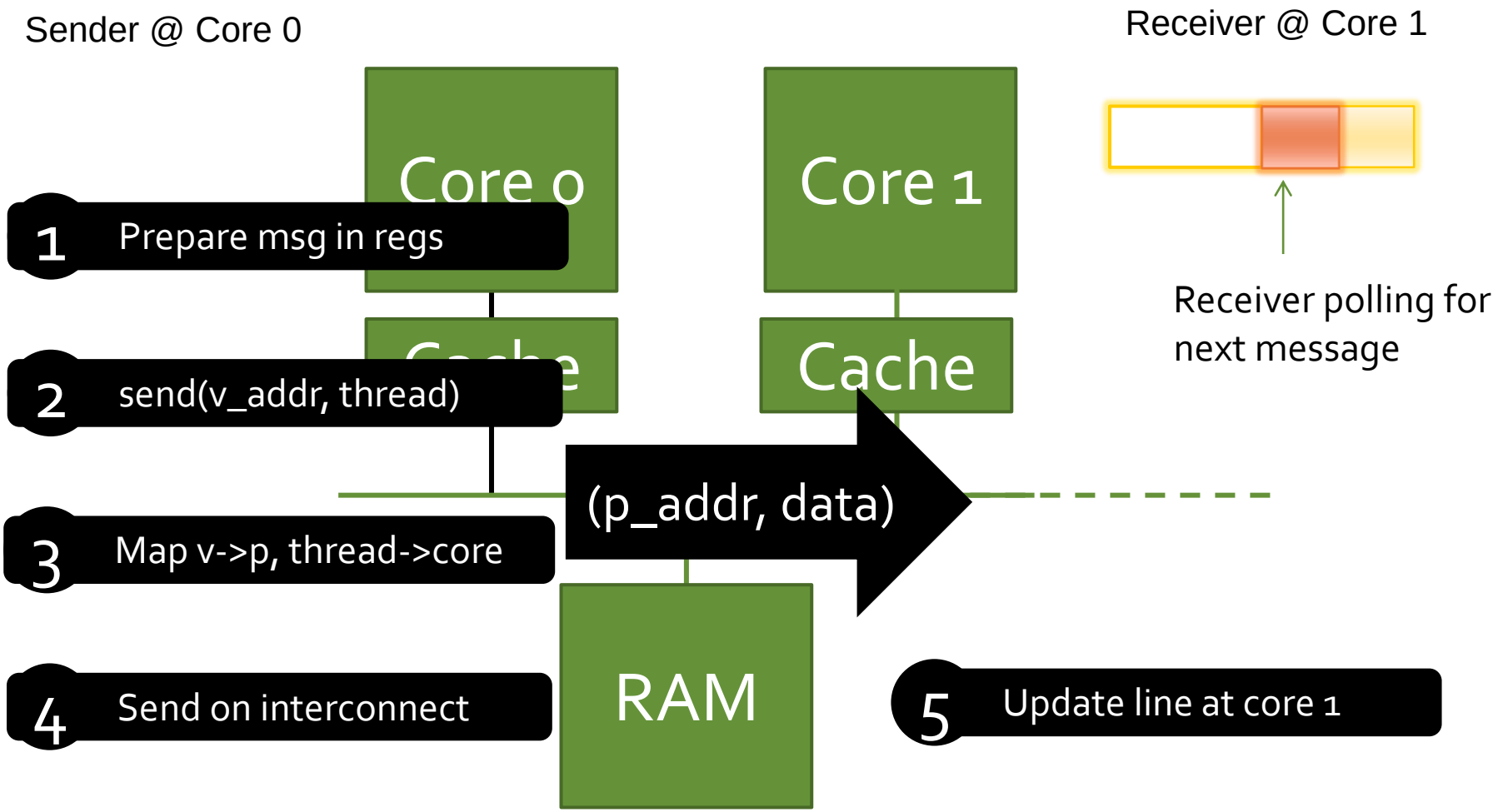
Address spaces



Prior work


Key idea:
Add just enough HW to (i) avoid reliance on cache-coherent shared memory, and (ii) avoid need for polling / kernel on notification.
SW approaches:
Message channels over coherent shared memory.
Software polling, backing off to notification via kernel.
HW approaches:
Generally – resources are per-core, not per process.
Communication within processes, not between them.

Message passing: fast path



Message passing: slow paths

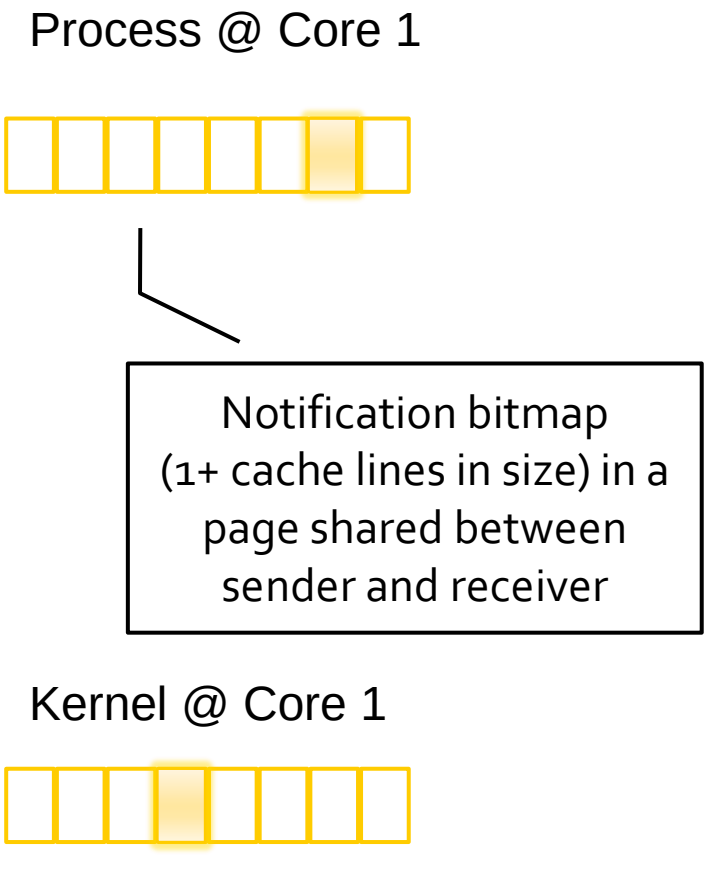
- Cache line not present at receiver?
  - Policy decision: speculatively allocate, or not
  - If not allocated: inject write from the sender's cache
- Receiver process not running…
  - …can always inject write, as above
- Receiver core unknown
  - Use thread translation buffer (TTB) to map s/w ID to target core
  - Handle consistency by TLB-shootdown-style approaches, or HW coherence between TTBs

Key ideas

- Protection and naming are done via existing virtual memory mechanisms
  - If I can write to a page you can read, then I can send a message to you
  - Multiple message channels can be used
  - Mutually-untrusting senders can send to the same receiver
- In the common case transfers are cache-to-cache
  - But storage is backed by ordinary memory…
  - …and in-flight messages can be reified in memory in tricky cases (paging to disk / hibernate / etc.)

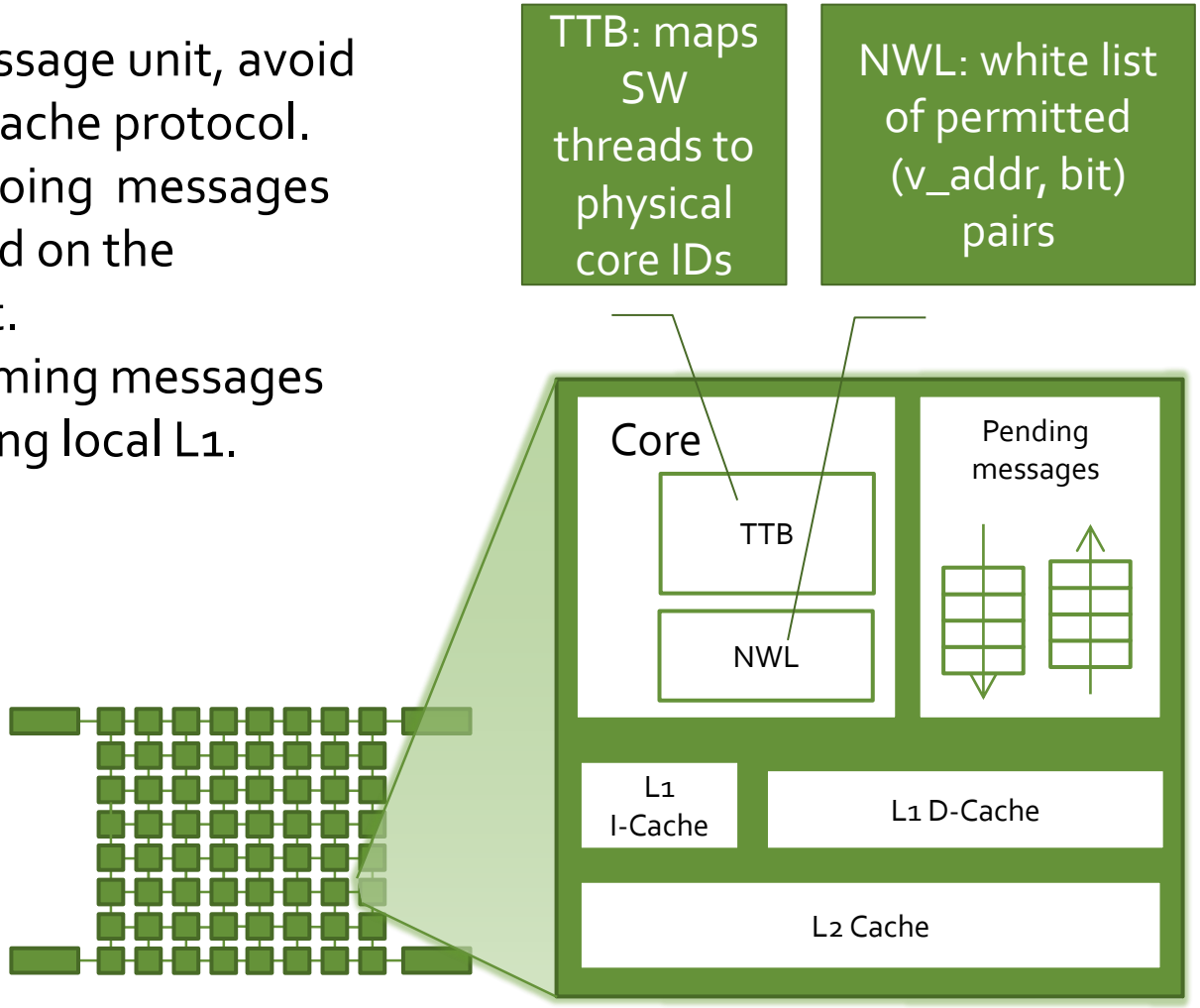
Notification

1. Send message as before
2. notify(user_addr, bit, thread)
3. notify(kernel_addr, bit, thread)
4. Rx core kernel watches bitmap



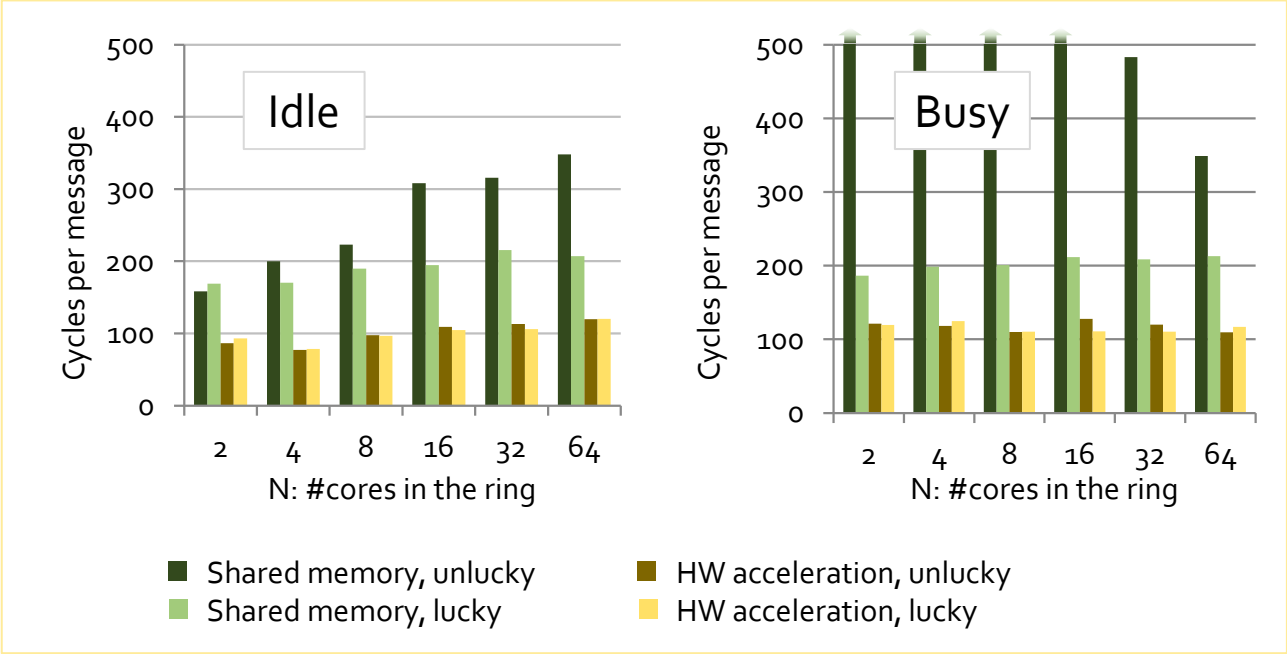
Simulated h/w

- Per-core message unit, avoid changes to cache protocol.
- Buffers outgoing messages while blocked on the interconnect.
- Buffers incoming messages while updating local L1.

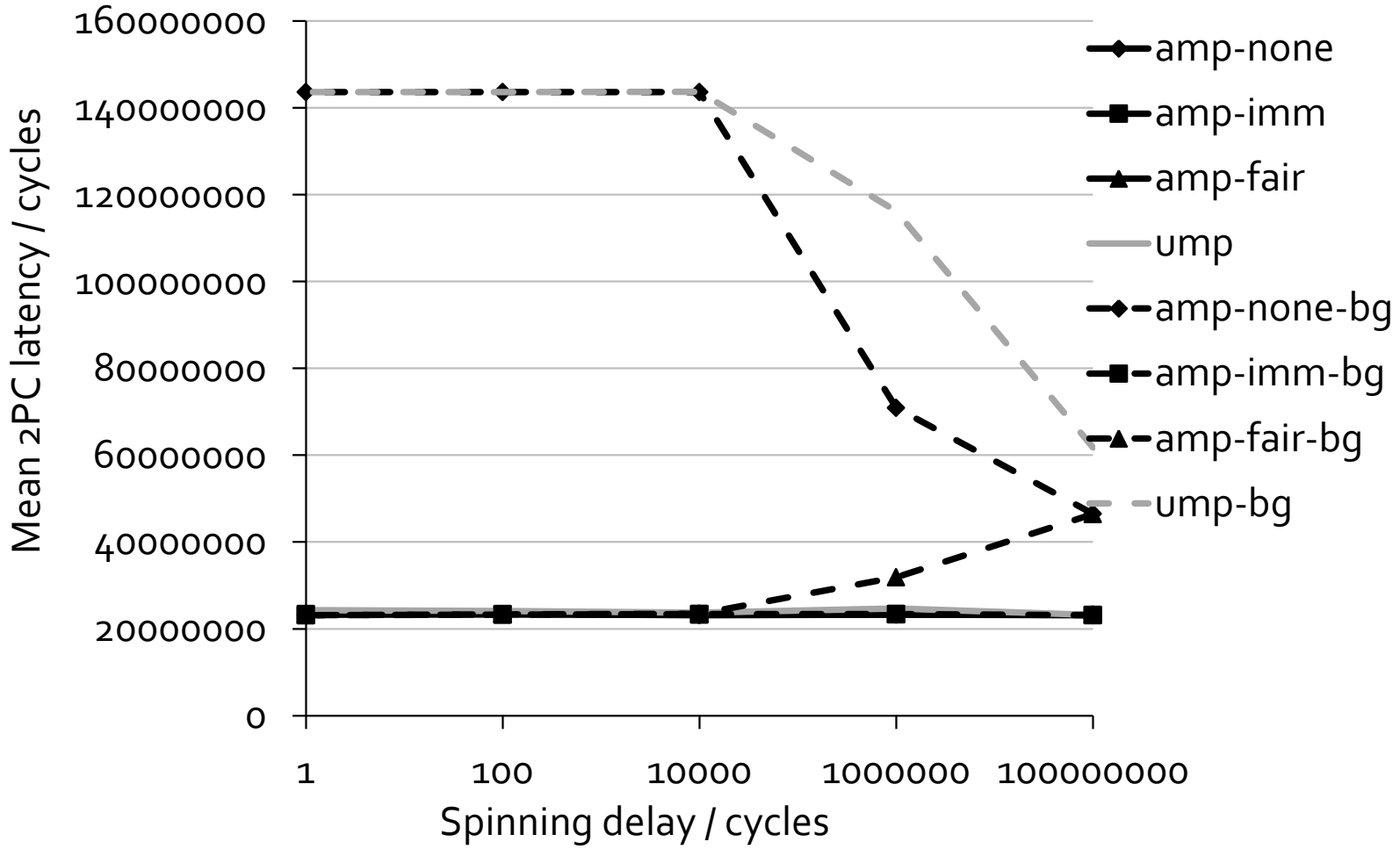


Microbenchmark

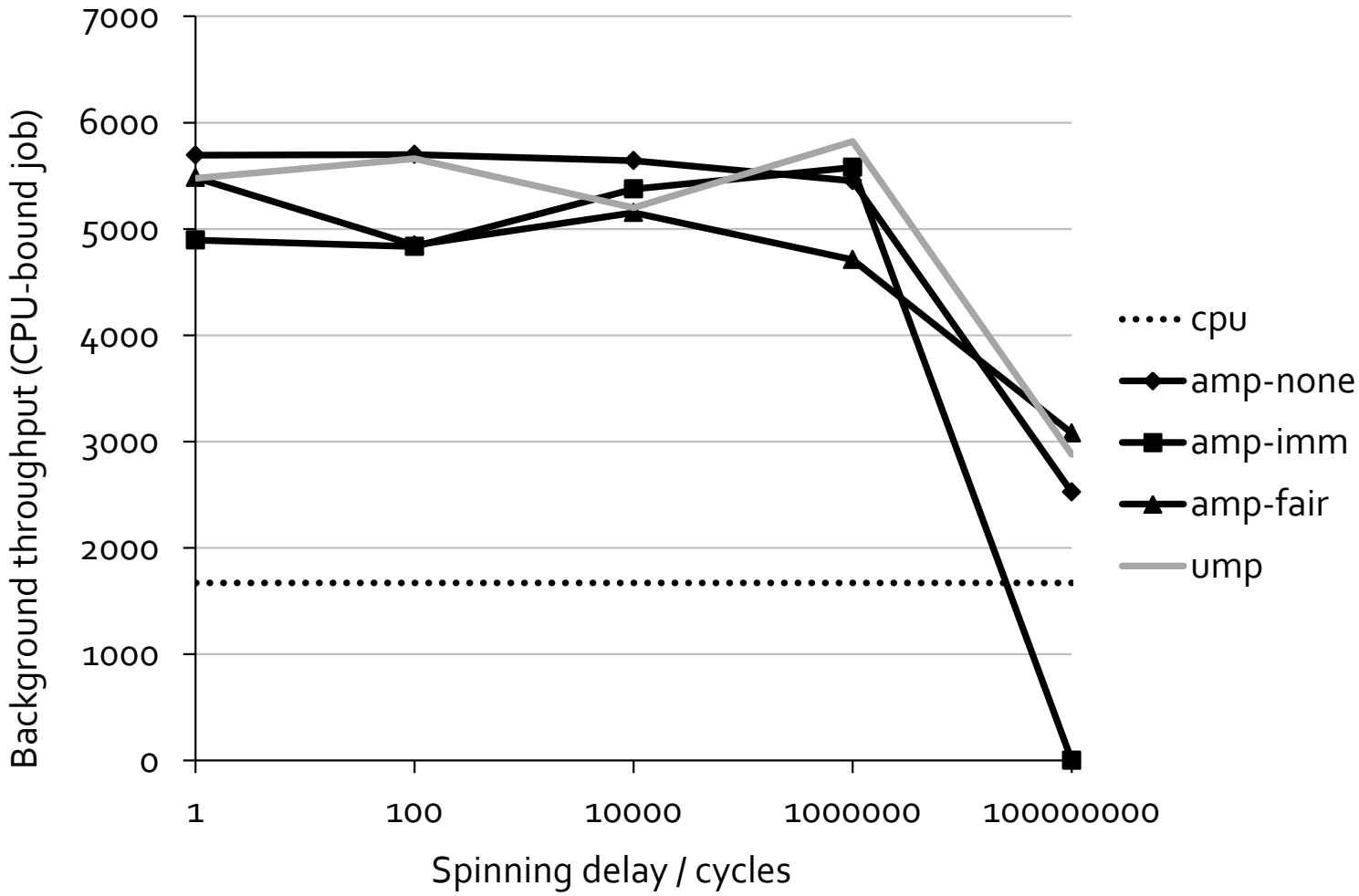
64-cores, per-core L1, plus one bank of a shared L2. A ring of N cores circulates a single message. We vary (*i*) N, the size of the ring, (*ii*) the placement of nodes and of buffers in L2, and (*iii*) cross-talk from other cores.



Preliminary results, 2-core 2PC



Preliminary results, 2-core 2PC



Current status

- Initial prototype for Beehive
  - Use in numerical message-based tests programs
  - Ping-pong, barrier, LU-factorization from SPLASH2
  - Message passing in Barrelfish
- Port of Barrelfish to Gem5 full-system simulator
  - Extensions to Gem5 to model non-coherent memory
- Implementation for real x86/x64 hardware
  - Ignores protection questions
  - Scaling is not quite right (modifications done sender-side)
  - Lets us look at OS integration
- What else can we use this for?

Microsoft Research

www.research.microsoft.com