Dynamic inter-core scheduling in Barrelfish

avoiding contention with malleable domains

Georgios Varisteas

October 2011

Outline

- Introduction
- Programming models
- Scheduling
- Malleability
- Future work

Overall Goals

- Allow for shared memory based parallel programming models
  - OpenMP, Wool, Cilk++
  - take advantage of the underlying hardware architecture.
- … while exploiting the message passing nature of Barrelfish
  - scalability
  - portability

Goals cont'd

- Increase throughput in Barrelfish by using dynamic inter-core scheduling

---

  - Inter-core
    - system wide scheduling
  - Dynamic
    - modifiable at any point of execution
  - In Barrelfish
    - maintain a scalable and portable design

Motivation

- Work-stealing scheduling can be wasteful
  - Threads can unnecessarily busy-wait
- Some real life applications are not that parallel
  - most expose fluctuating parallelism throughout their execution
- Current parallel programming models:
  - focus on running programs in isolation
  - have minimal operating system support

Shared memory programming models (OpenMP, Wool, Cilk++)

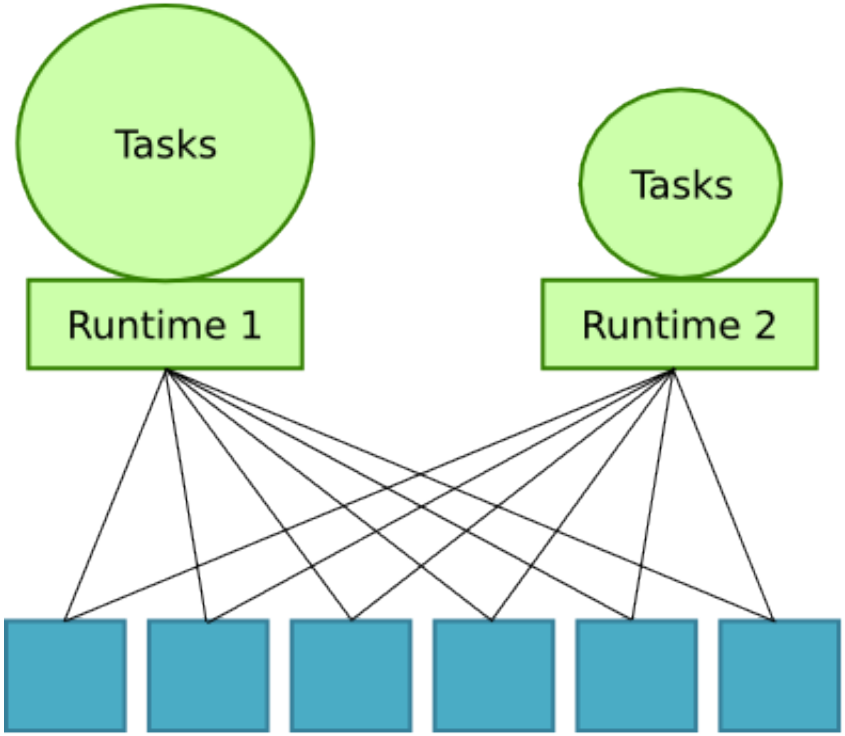
- Work-stealing / task-based models scale easily
- Wool already ported
  - very fast (low overhead) implementation of independent task parallelism

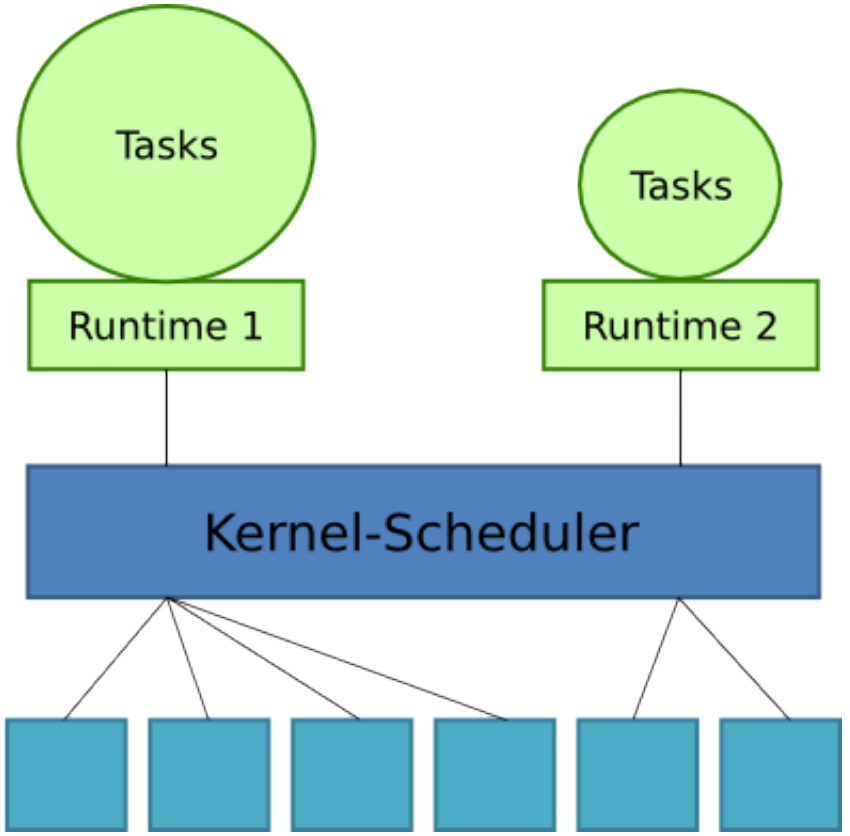
    application state in the stack
- Cilk++ requires a custom compiler (hake?)
  - important reason for needing the Cilk way of doing things:

    application state in the heap

Scheduling

- Split into two cooperating levels
- Kernel level,
  - system wide, mostly space-sharing, scheduler
  - aware of the global state and the availability of diverse resources
- User level,
  - application specific scheduler
  - aware of the parallelism in the application


Tasks
Tasks
Runtime 1
Runtime 2


Tasks
Tasks
Runtime 1
Runtime 2
Kernel-Scheduler

User level scheduler

- Integrated into the application run time
- Schedules a process' threads among the available cores (domain)
- Provides feedback on per core efficiency, to the Kernel level scheduler [1]
  - initial metric: *wasted cycles*

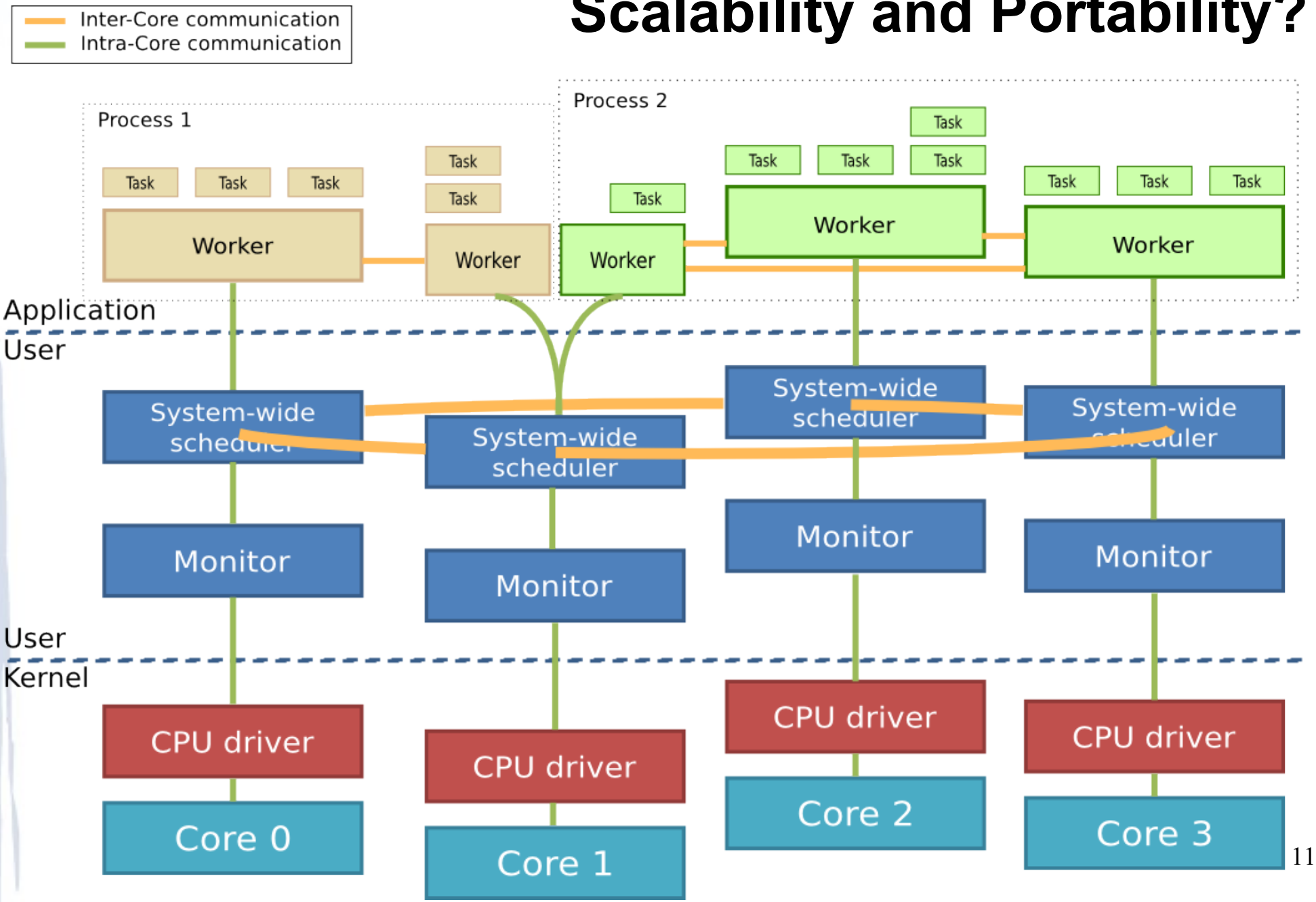
    “cycles spent while unsuccessfully trying to find work”

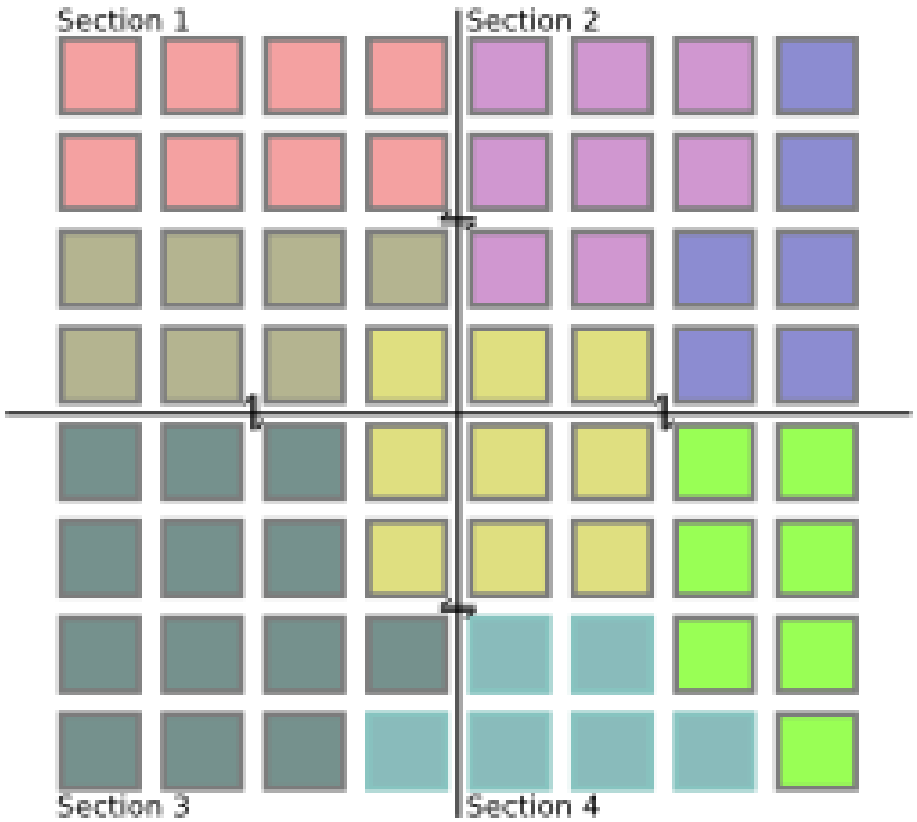
[1] Kunal Agrawal, Charles E. Leiserson, Yuxiong He, and Wen Jing Hsu. Adaptive work-stealing with parallelism feedback. ACM Transactions on Computer Systems, 26(3):1-32, September 2008.

Kernel-level scheduler

- Dynamically allots cores to processes
  - accepts feedback on process efficiency per core
  - modifies the domain of each process for maximum resource utilization
- Distributed service
  - multiple instances overlook distinct segments
  - domains extend over multiple segments
  - leader election decides primary instance per process

Scalability and Portability?





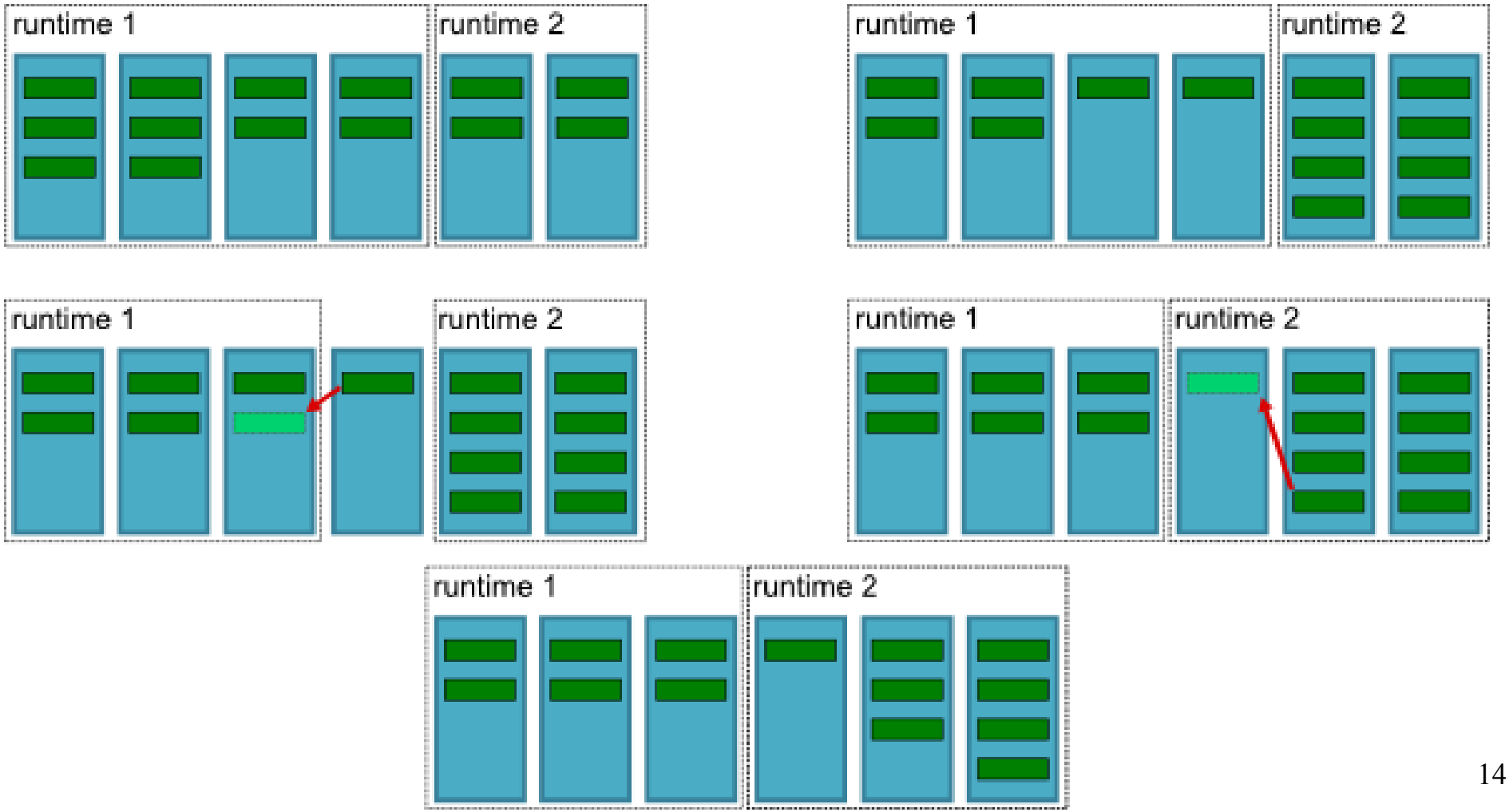
- 4 sections, 4 scheduler instances
- Yellow process extended over all sections
- Section 4 has primary control over the yellow process

Malleable domains

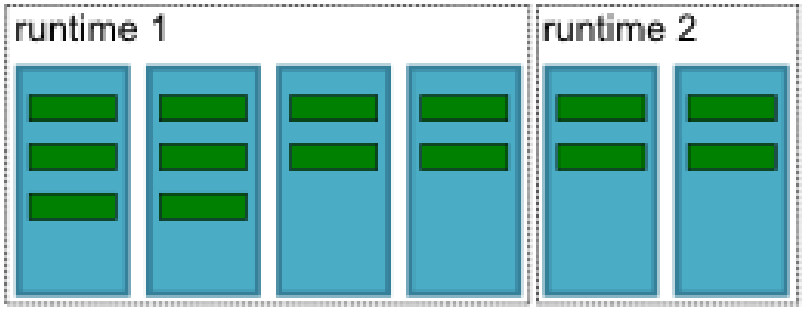
- Load balance the system by modifying the domain of each process
  - unwanted worker-threads are suspended
  - or new ones are added
- Worker-thread suspension tricky
  - **lazy-suspension**: threads are moved to a new core. scheduled until their subtree is synced
  - **immediate suspension**: application state in the heap[1], so some other worker can immediately take over

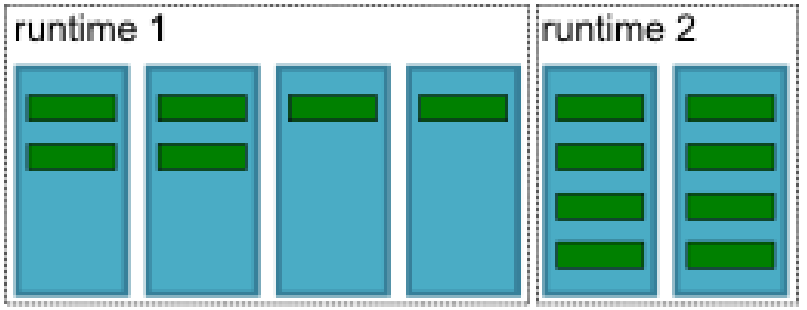
1) Continuation-passing-style: Shared memory is used instead of the CStack.

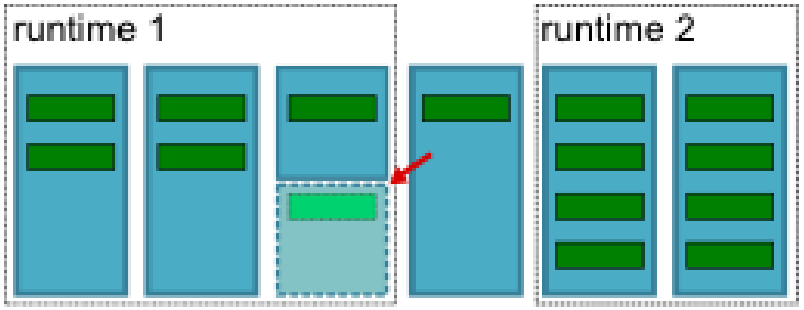
Immediate suspension

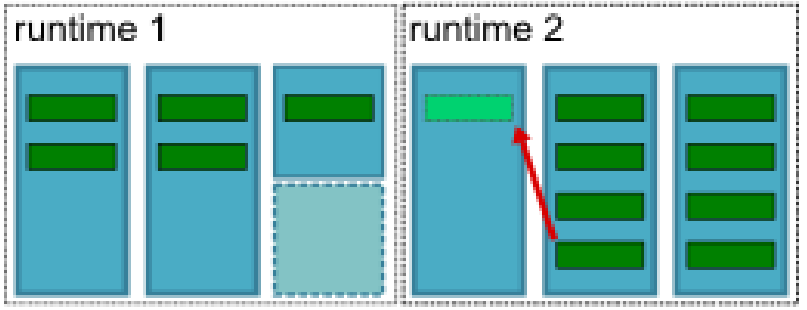


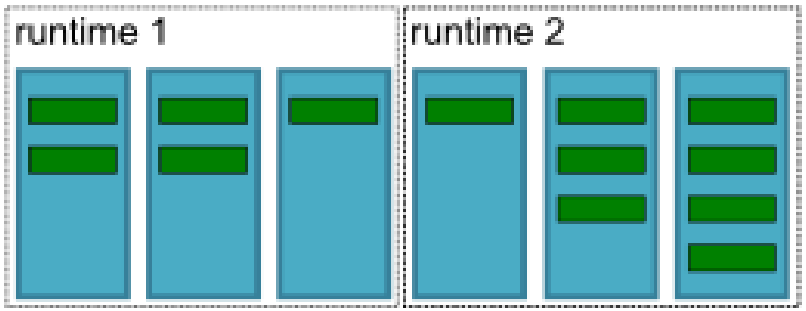
Lazy suspension











Time sharing not always avoided

- Phase-lock gang scheduling [1]
  - Efficient gang scheduling for barrelfish
- Joining a task requires simultaneous execution of the workers involved

[1] S. Peter et al., “Design principles for end-to-end multicore schedulers,” in Proceedings of the 2nd USENIX conference on Hot topics in parallelism, 2010, p. 10.

Future Work

- Locality aware allotment of cores
- Handling the absence of shared-memory support by the underlying architecture
- Explore heterogeneous architectures and take into account core properties

THANK YOU

Q & A