Arachne: Core Aware Thread Management

Henry Qin, Qian Li, Jacqueline Speiser,
Peter Kraft, John Ousterhout

Stanford University

PLATFORMLAB

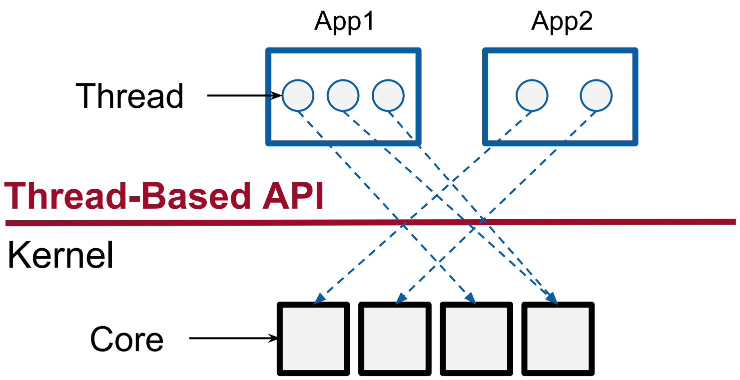
Latency Conflicts With Throughput

- **Task lifetimes getting shorter in the data center**
  - Memcached service time: 10 μs
  - RAMCloud service time: 2 μs

- **Low Latency → Poor Core Utilization → Low Throughput**

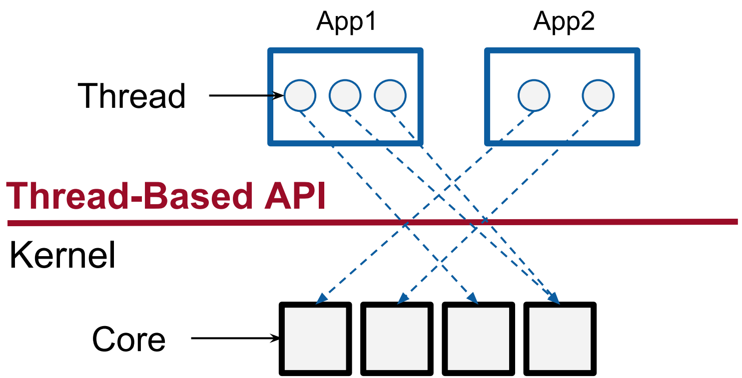
Arachne: Core Aware Thread Management

Today: Applications lack visibility and control over cores

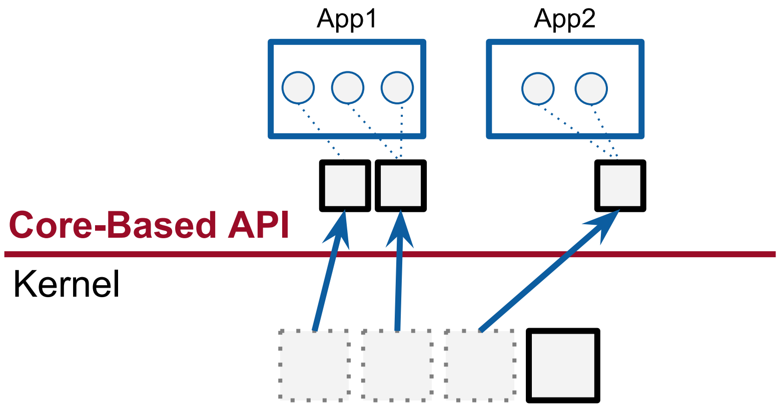


Arachne: Core Aware Thread Management

Today: Applications lack visibility and control over cores

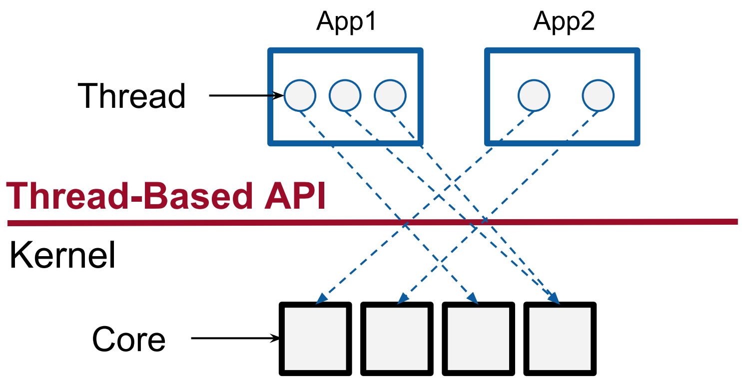


Arachne: Core Awareness for Applications

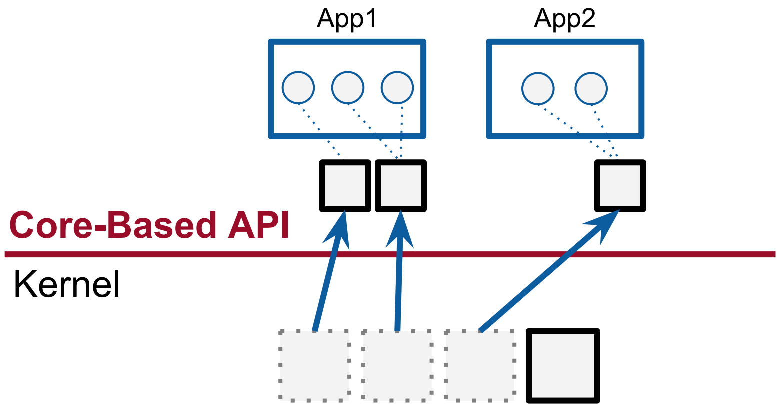


Arachne: Core Aware Thread Management

Today: Applications lack visibility and control over cores



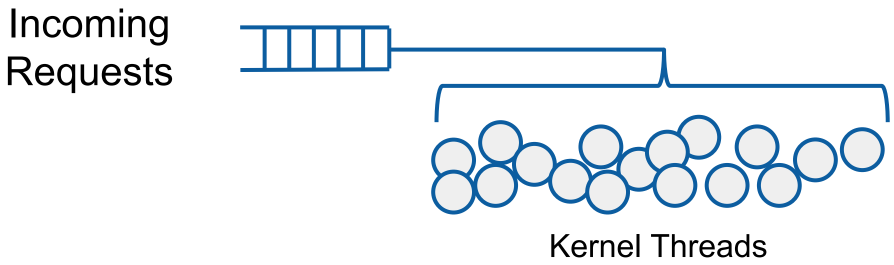
Arachne: Core Awareness for Applications



- **Better combination of latency and throughput**
  - Memcached: 4 – 43x reduction in tail latency
    37% higher throughput at 100 µs latency
  - RAMCloud: 2.5x higher throughput
- **Efficient threads implementation:** 100 - 300 ns thread primitives

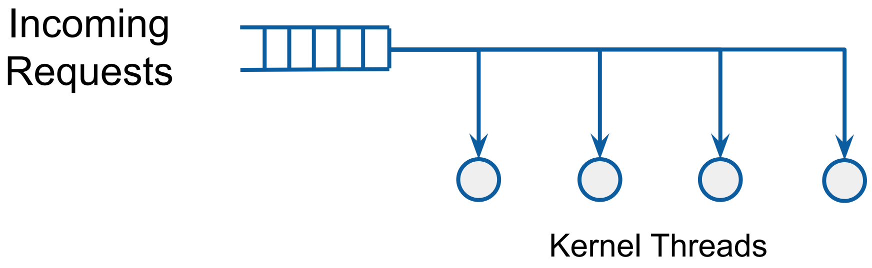
Problem: Kernel Threads Inefficient

One kernel thread per request? Too Slow!



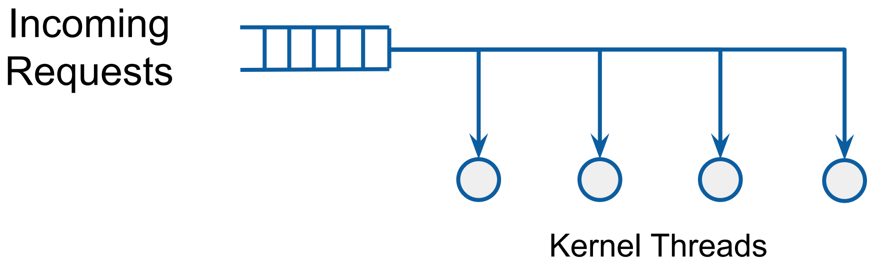
The Solution of Today's Applications

Multiplex requests across long-lived kernel threads.



Problem: Matching Parallelism to Resources

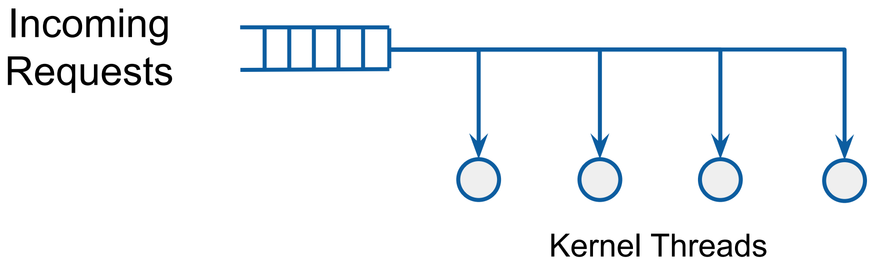
Multiplex requests across long-lived kernel threads.



How many threads?

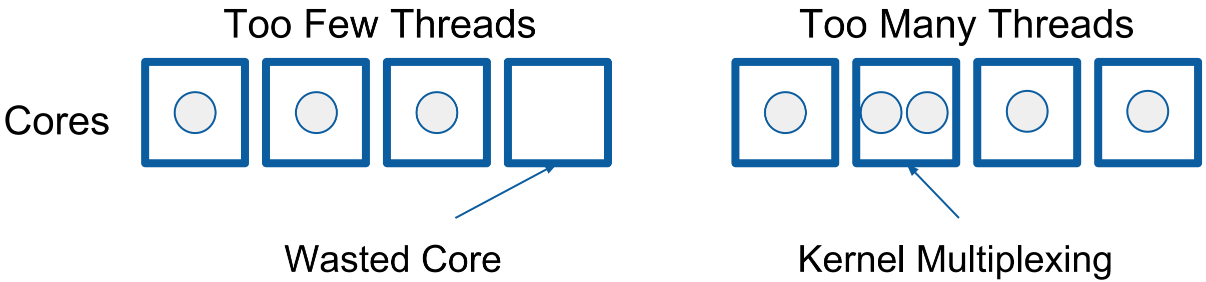
Problem: Matching Parallelism to Resources

Multiplex requests across long-lived kernel threads.



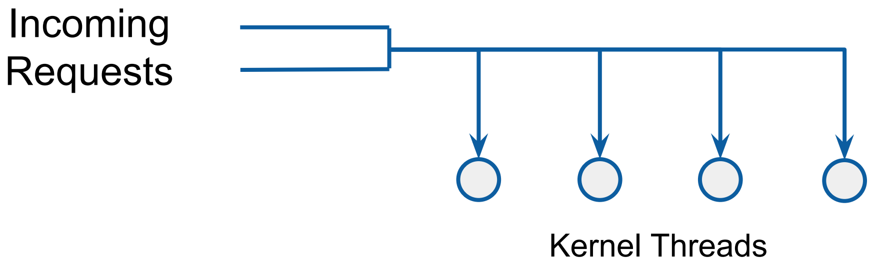
How many threads?

Goal: # of threads = # of cores, but don't know # of allocated cores

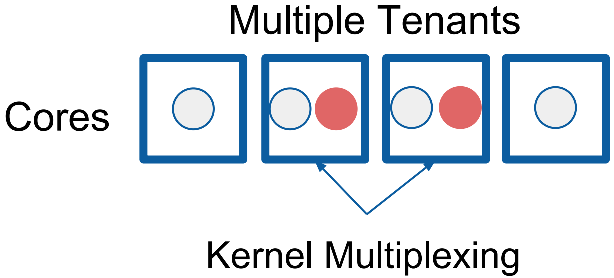


Problem: Must Choose Waste or Interference

Owning entire machine is wasteful.



Sharing causes competition for cores.

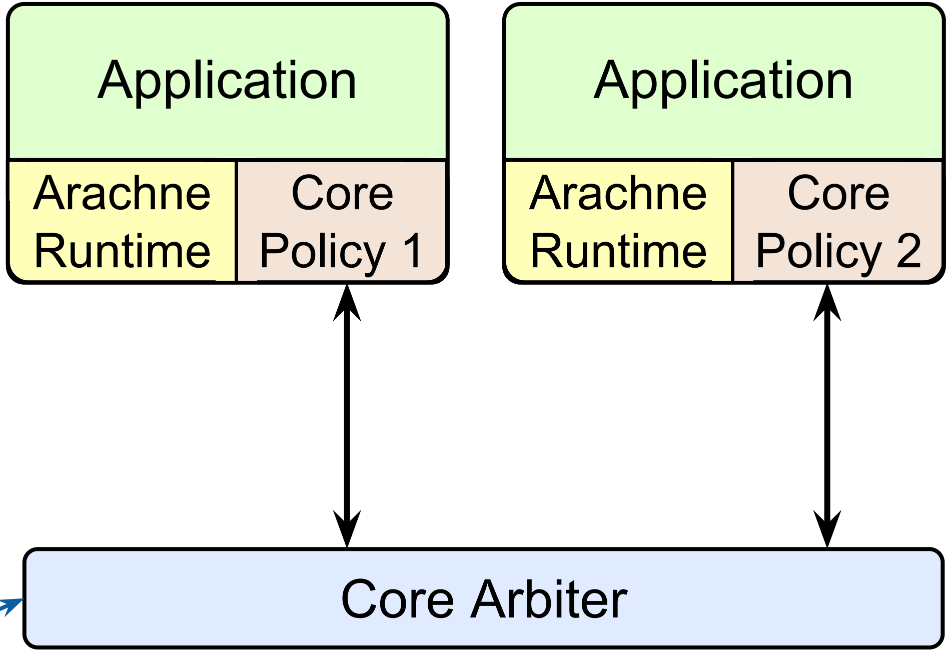


Arachne: Core-Aware Thread Management

- **Give applications more knowledge/control over cores**
  - Application requests cores, not threads
  - Application knows the exact cores it owns
  - Application has exclusive use of cores → eliminates interference

- **Move thread management to userspace**
  - Multiplex threads on allocated cores
  - Very fast threading primitives (100 - 300 ns)

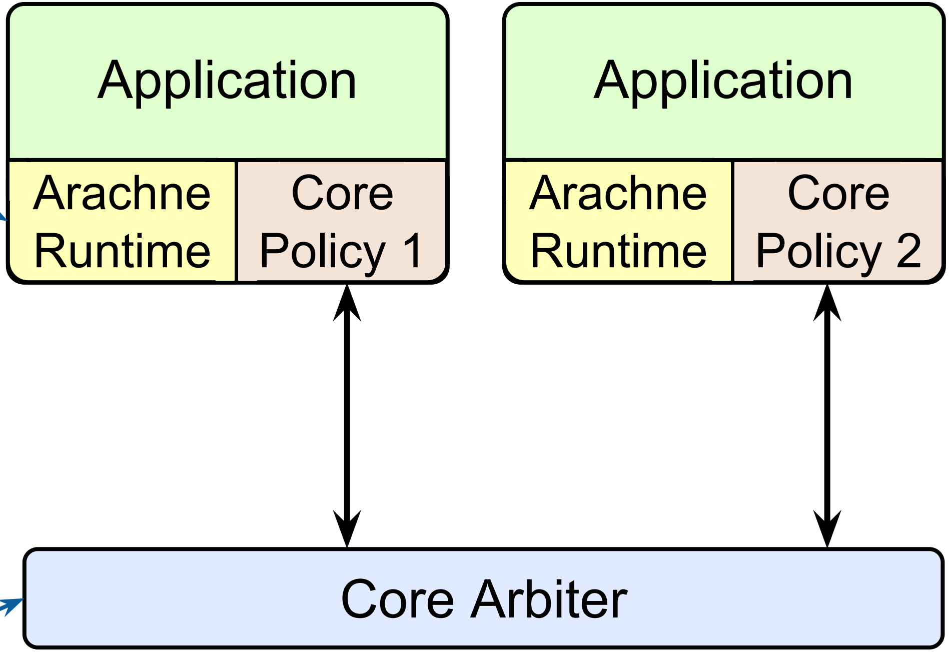
System Overview



- Allocates cores

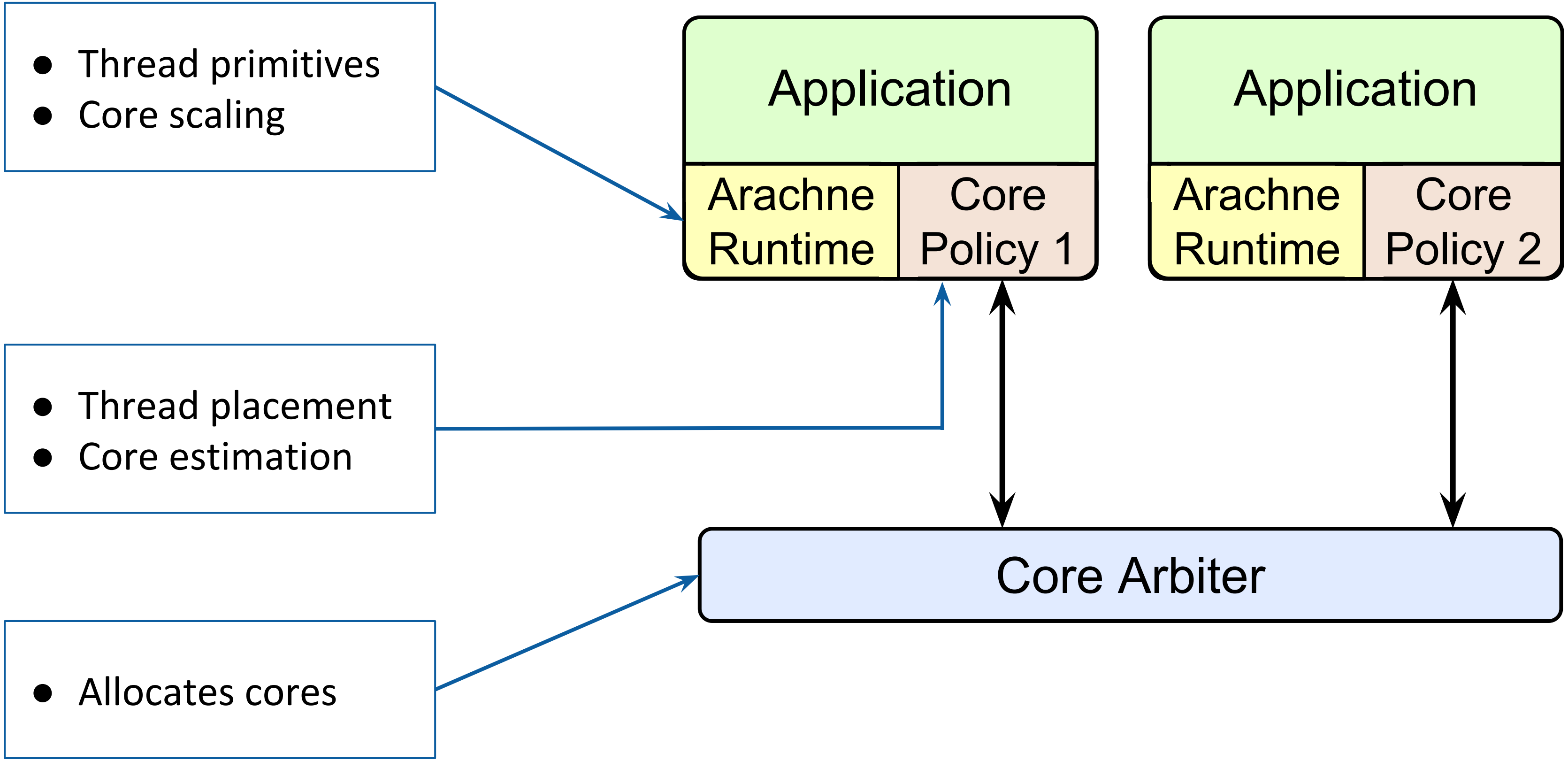
System Overview

- Thread primitives
- Core scaling



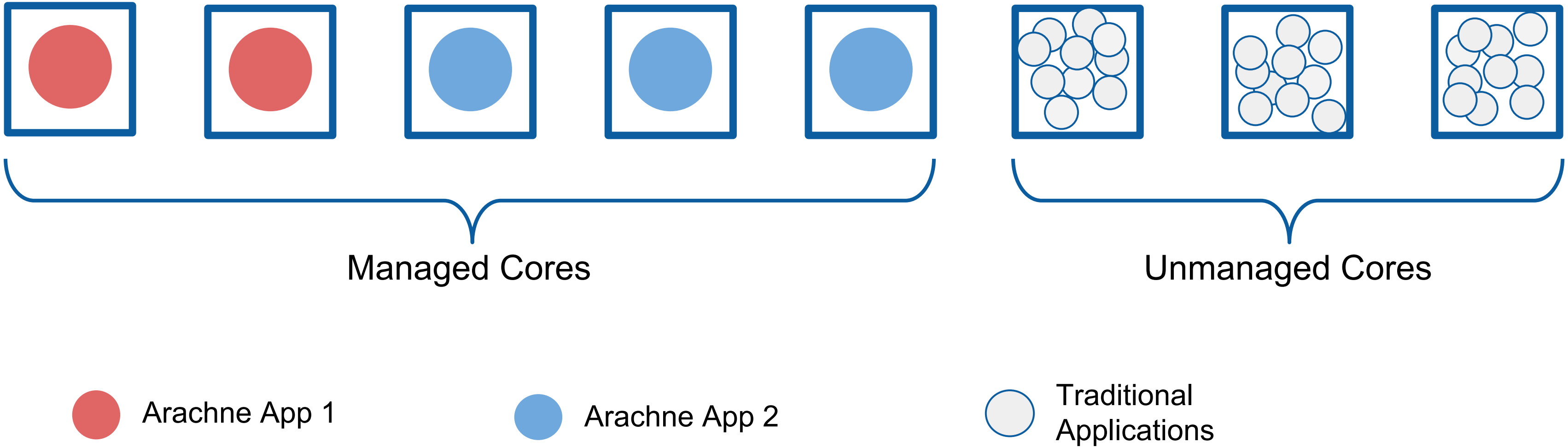
- Allocates cores

System Overview

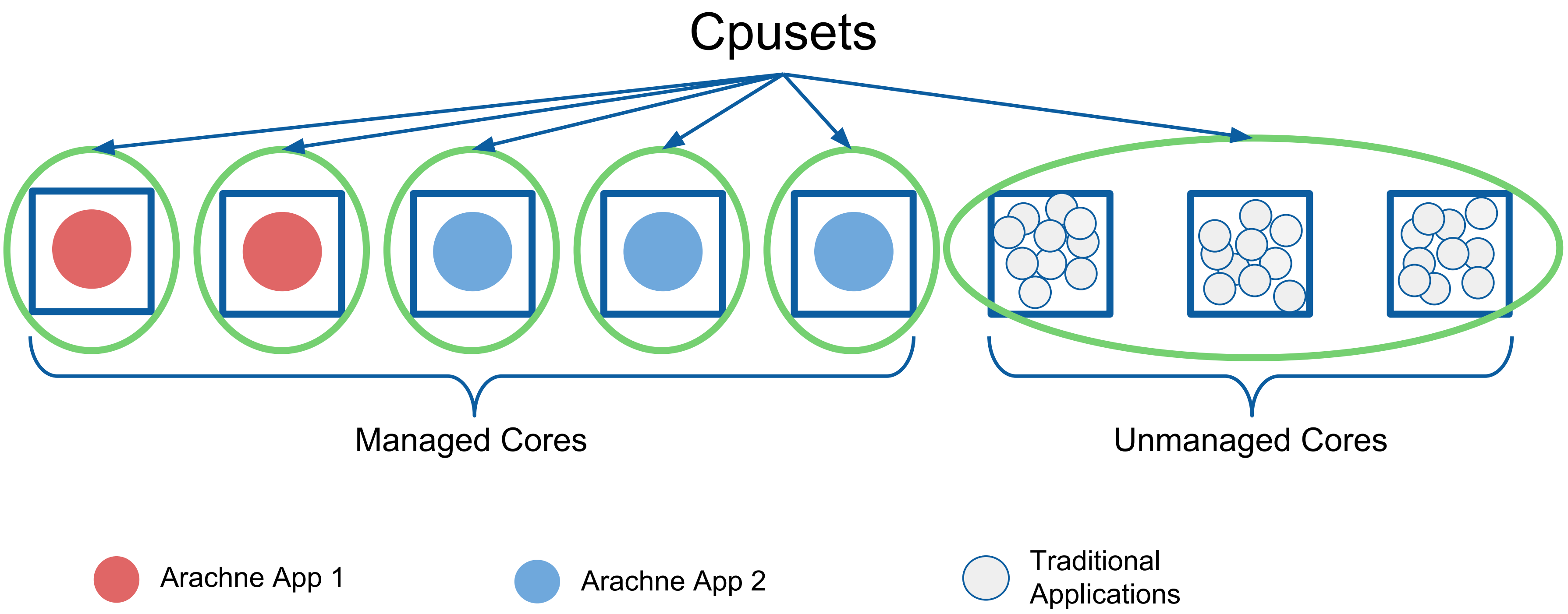


Core Allocation

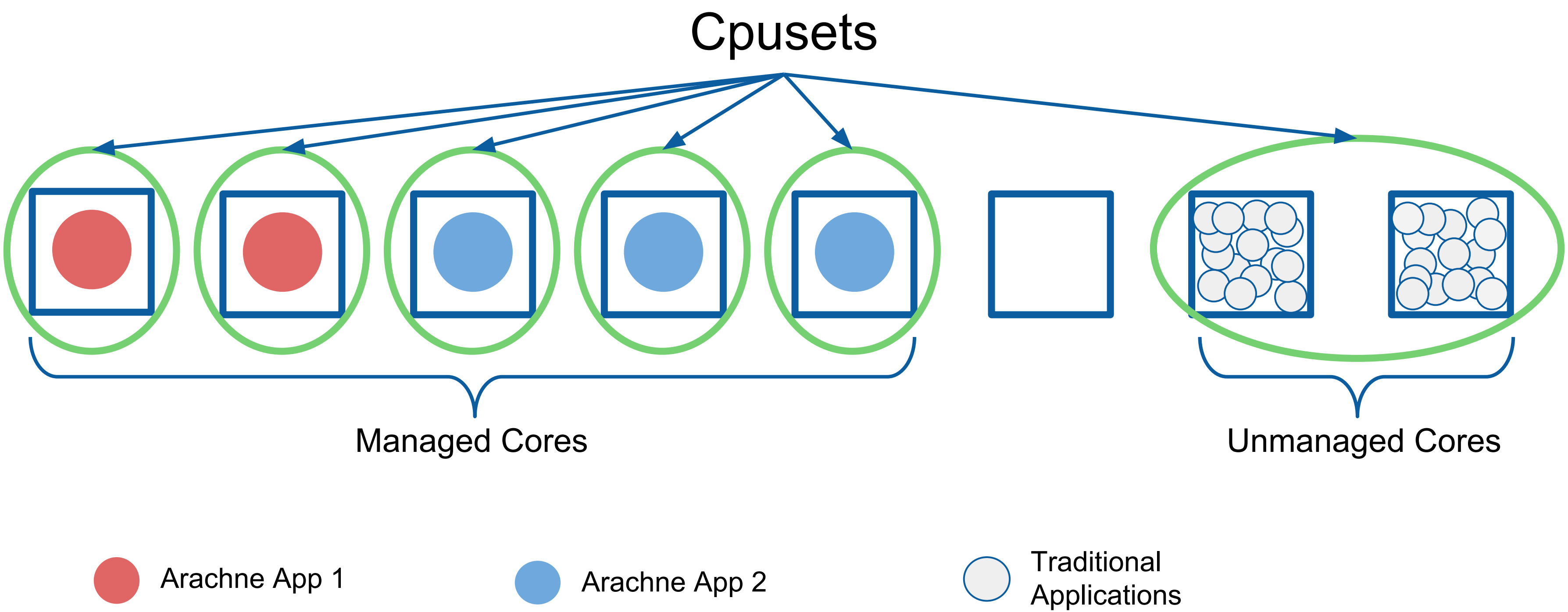
One Kernel Thread Per Managed Core



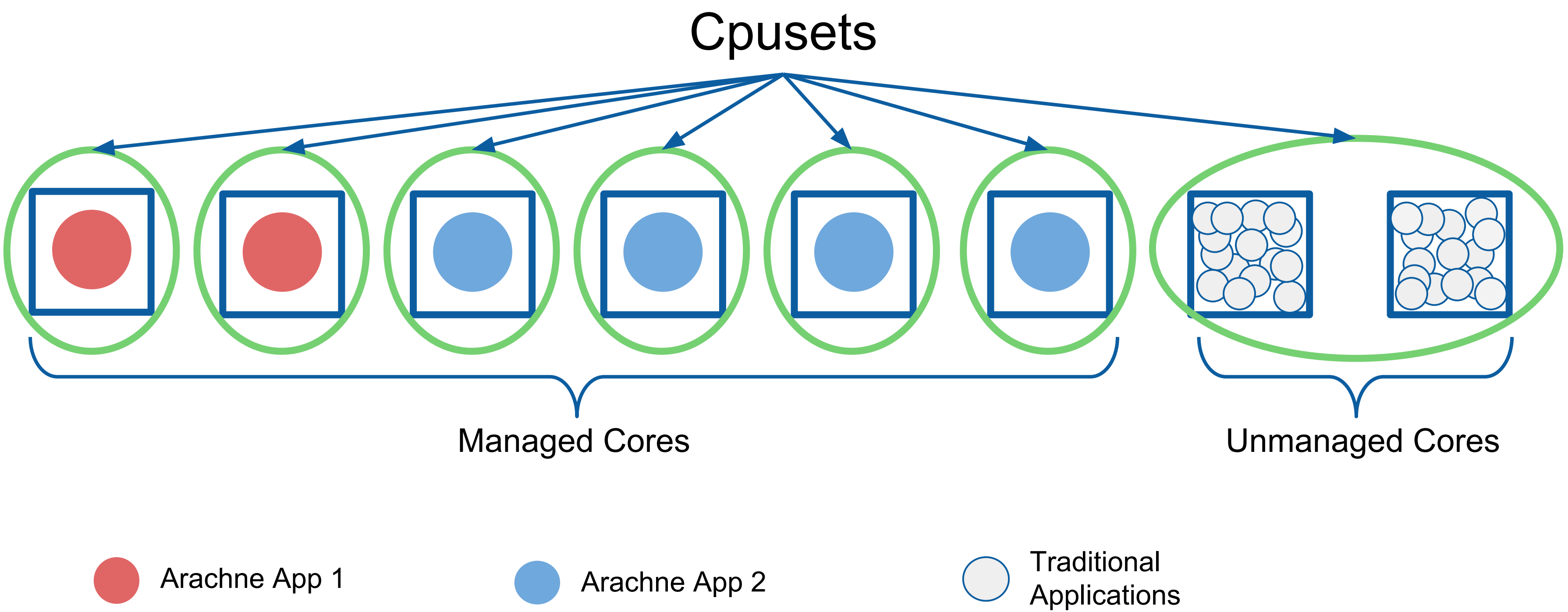
Leverage Linux cpusets



Granting a Core

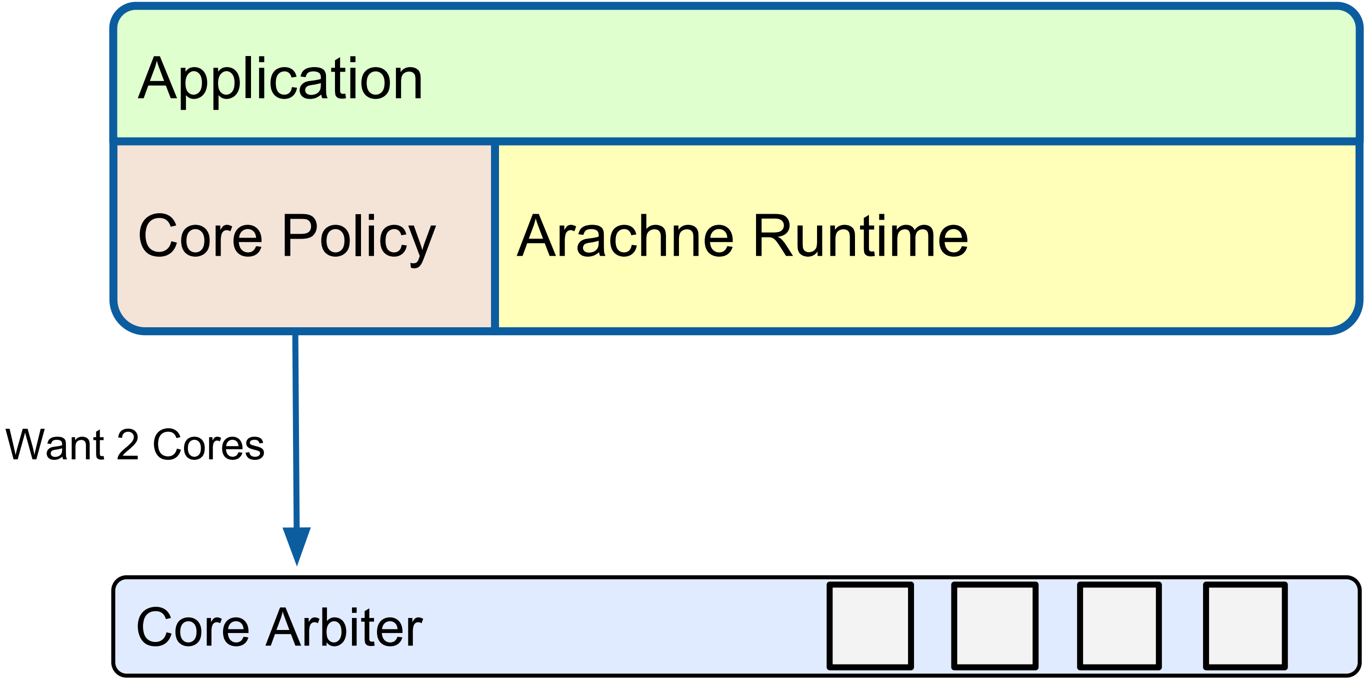


Granting a Core

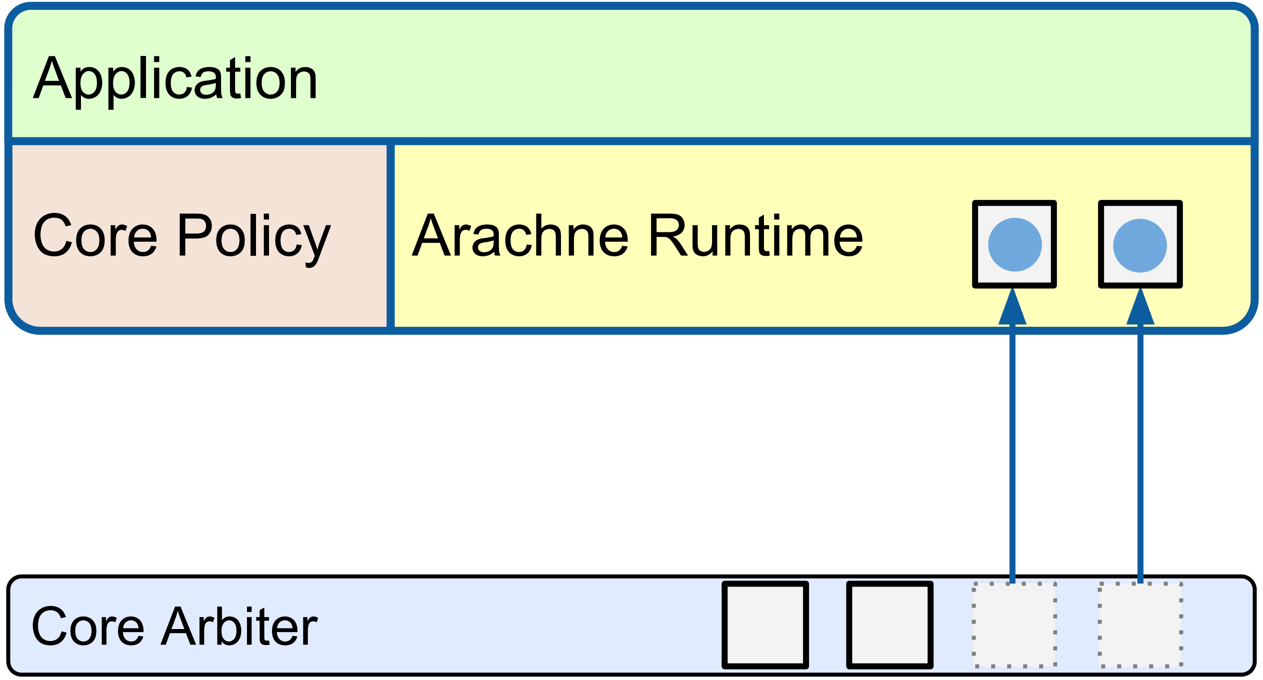


Life of an Arachne Application

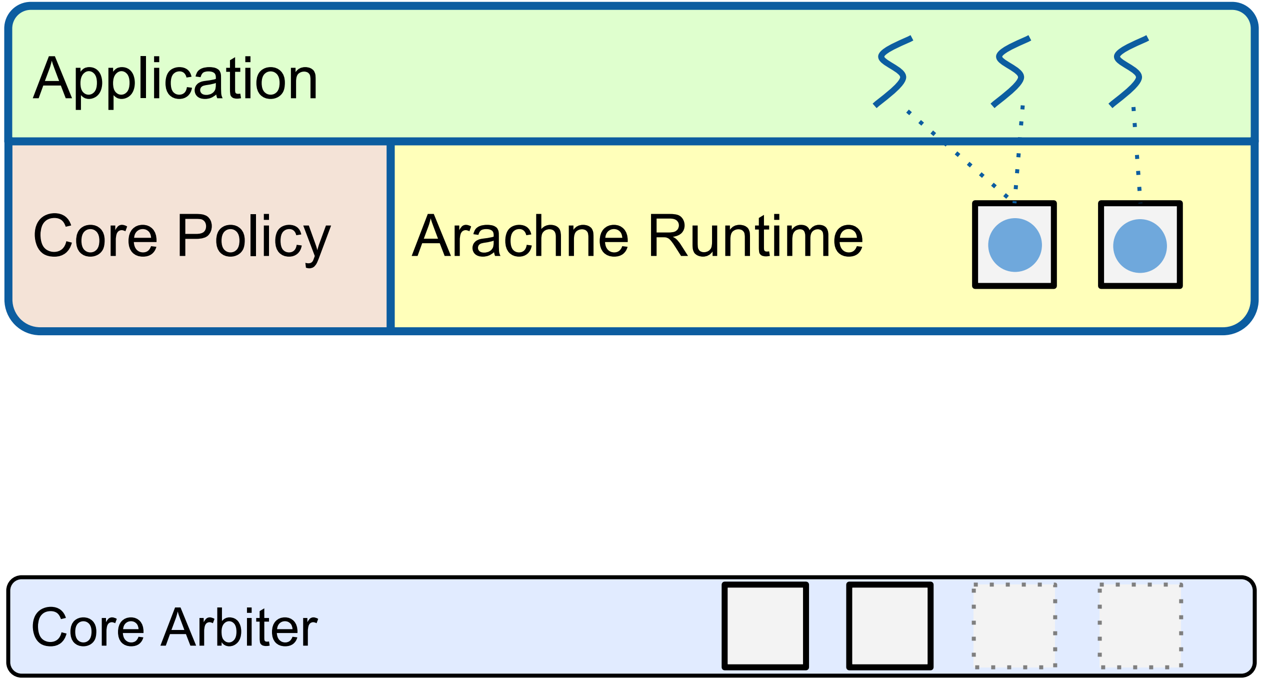
Application Startup



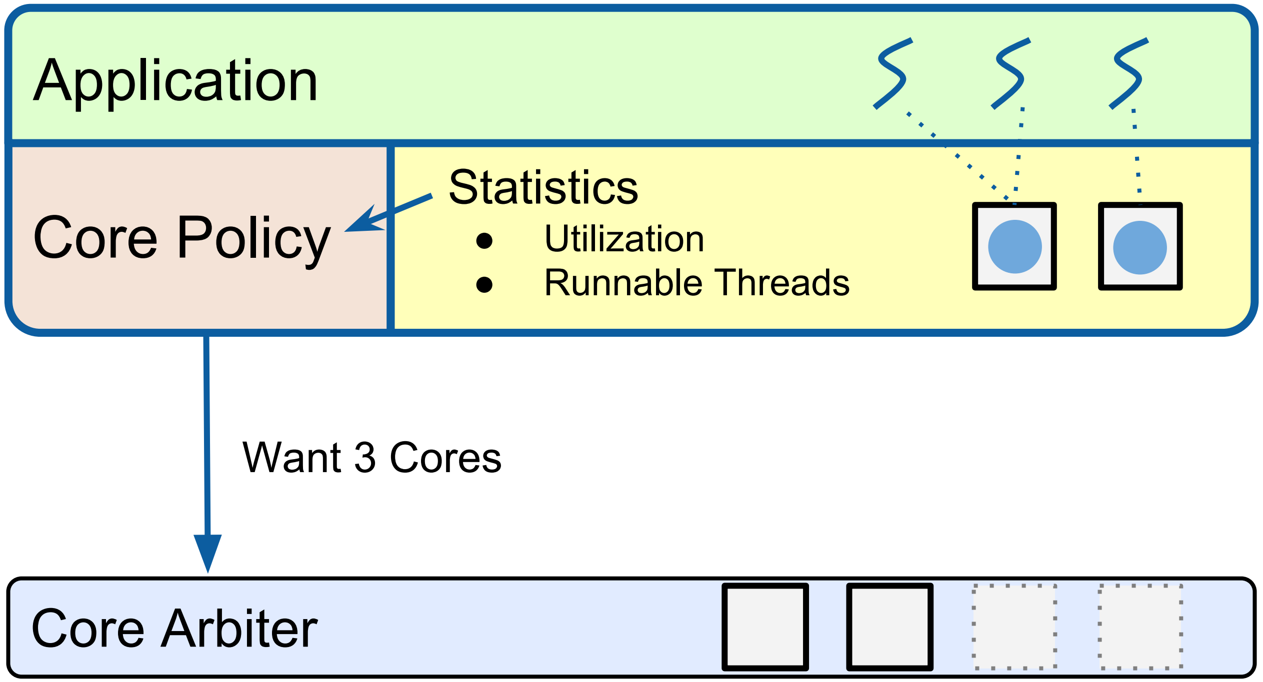
Application Startup



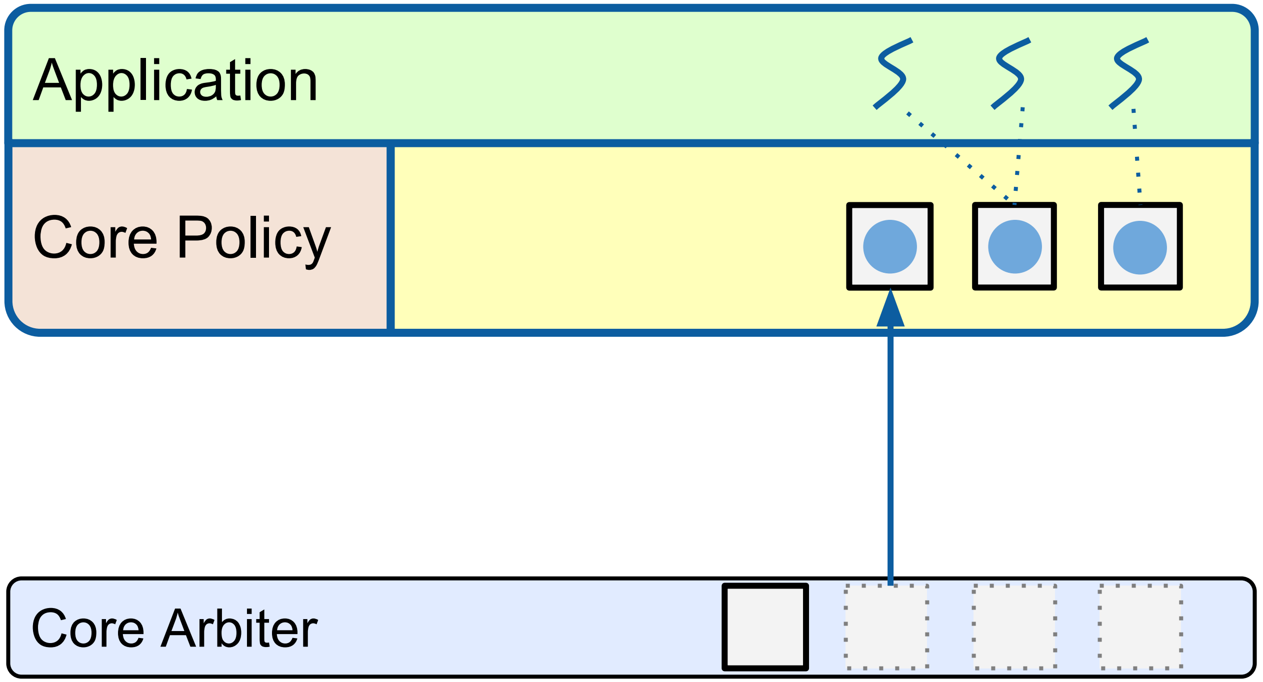
Multiplex User Threads



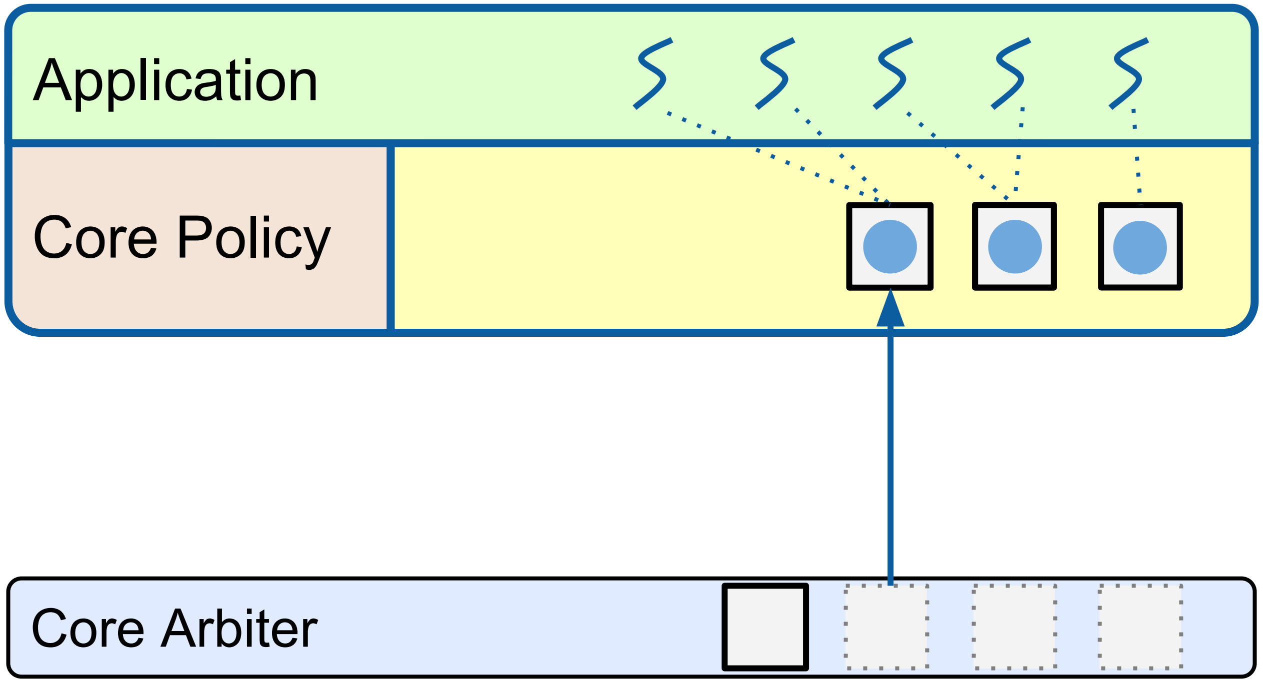
Core Estimation



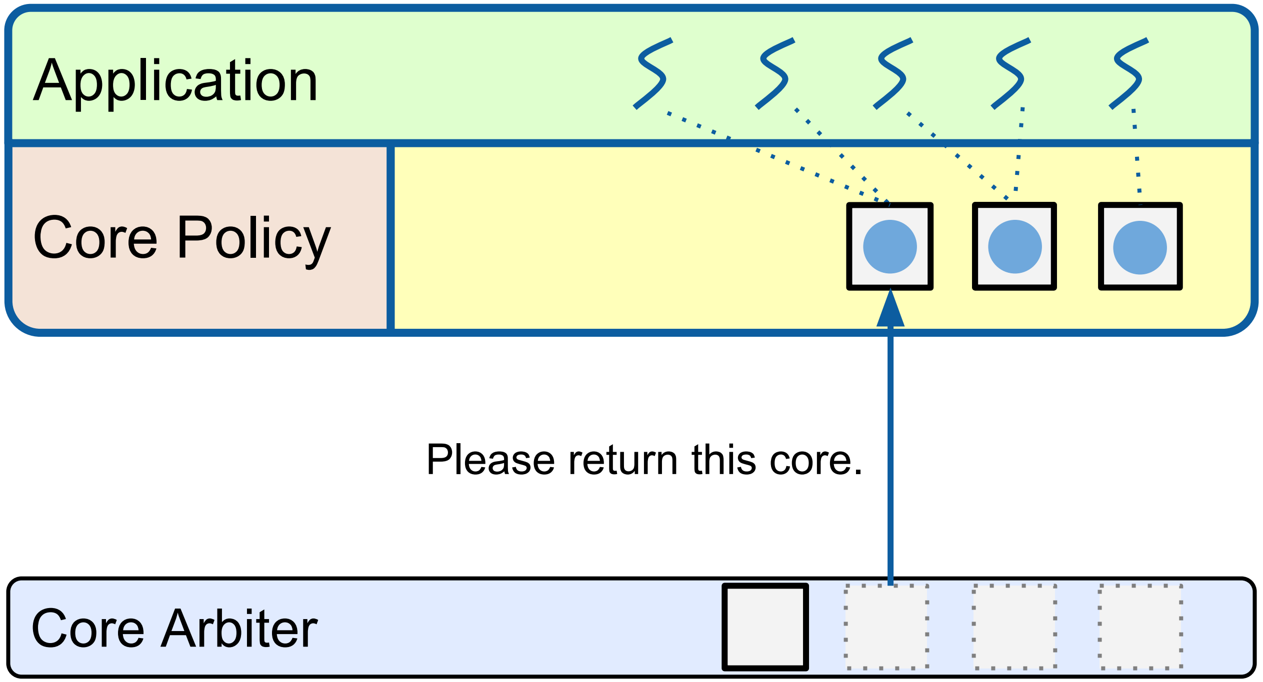
Core Grant



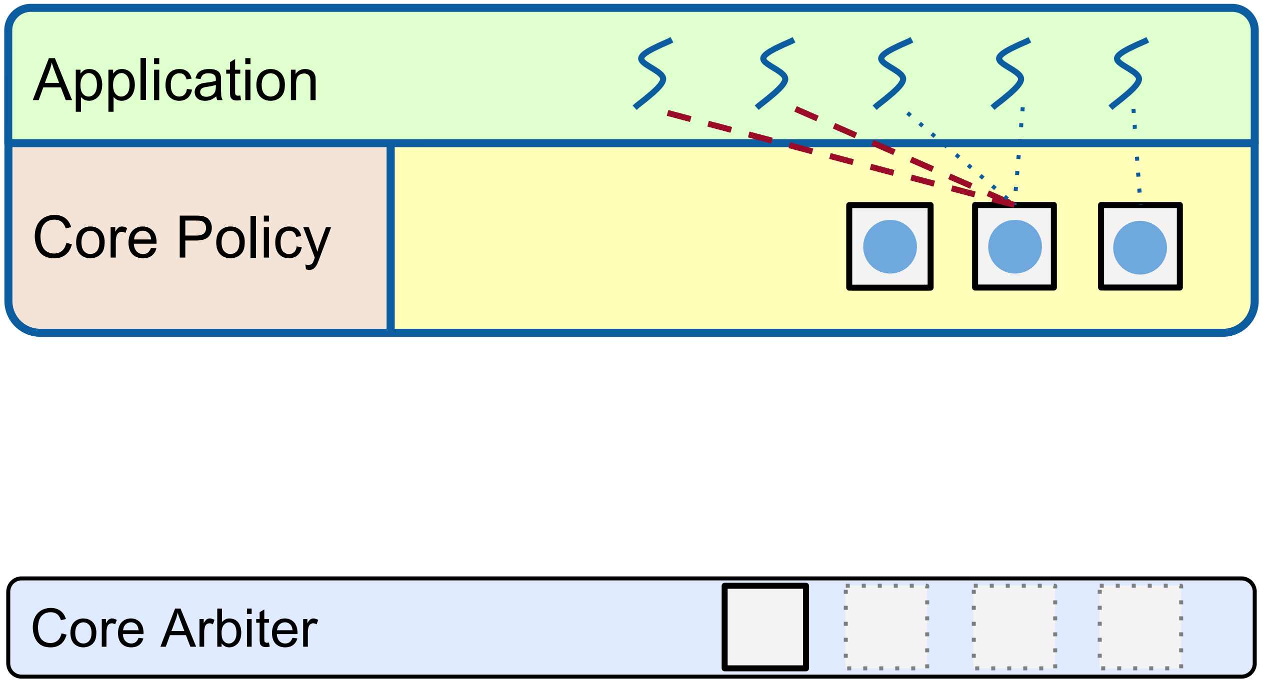
Core Grant



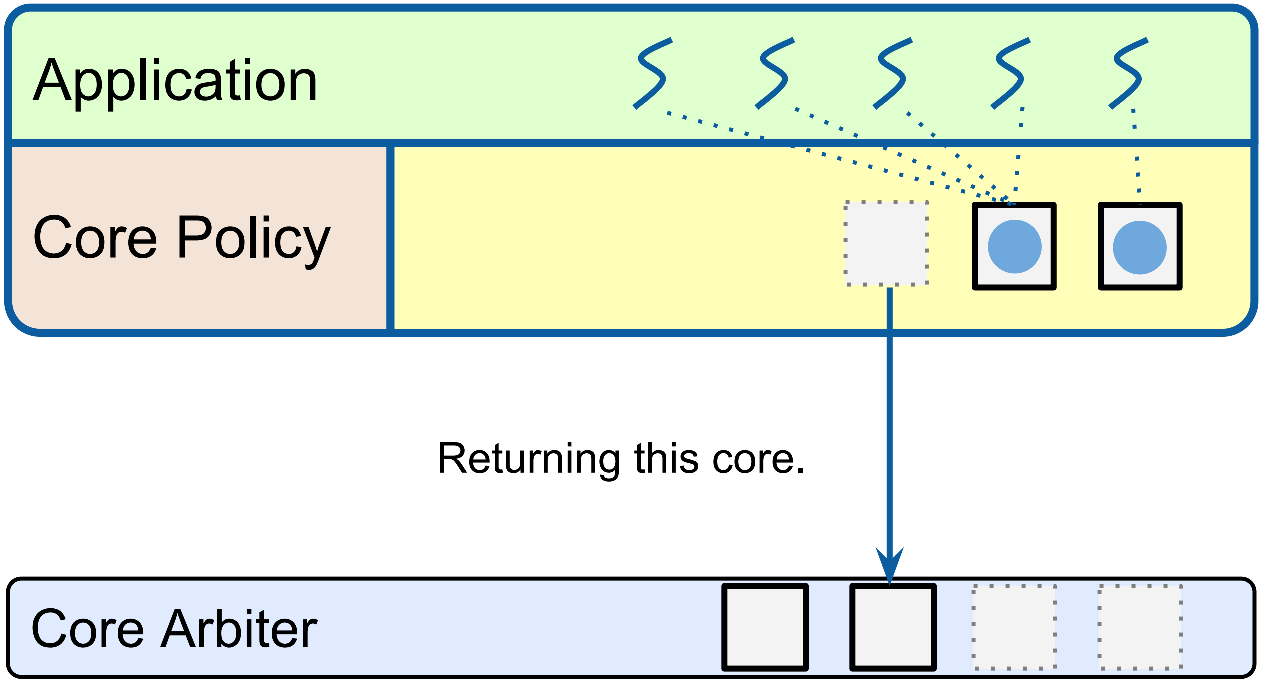
Core Preemption



User Thread Migration



Core Preemption Respected



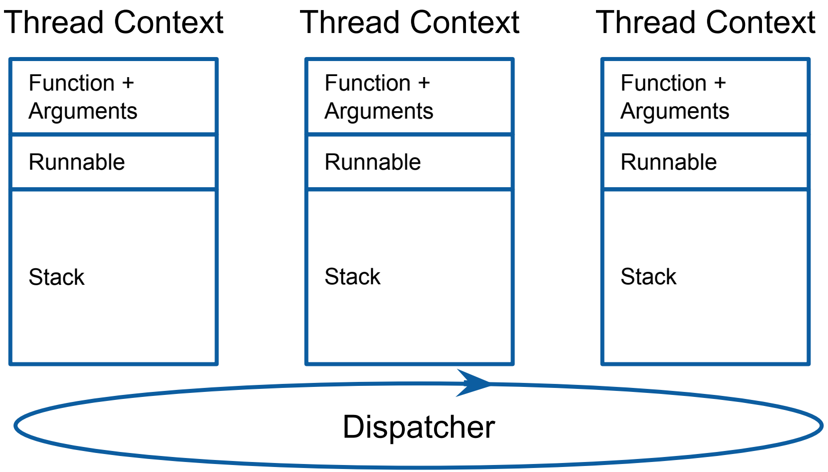
Arachne Runtime: Cache-Optimized

Cache-Optimized Design

- **Threading performance dominated by cache operations**
  - Basic operations not compute heavy
    - Context switch: only 14 instructions
  - Cost comes from cache coherency operations
    - Need to move data between caches
    - Cache miss: 100-200 cycles
- **Arachne runtime designed around cache as bottleneck**
  - Eliminate cache misses where possible
  - Overlap unavoidable cache misses

Cache-Optimized Design

- **Concurrent misses**
  - Read load information from multiple cores in parallel
- **No run queues; dispatcher scans context Runnable flags**



- **Total time to create a new thread, with load balancing: 4 cache misses**

Evaluation

Evaluation

- **Configuration (CloudLab m510)**
  - 8-Core (16 HT) Xeon D-1548 @ 2.0 Ghz
  - 64 GB DDR4-2133 @ 2400 Mhz
  - Dual-port Mellanox ConnectX-3 10 Gb
  - HPE Moonshot-45XGc

- **Experiments**
  - Threading primitives
  - Latency vs Throughput
  - Changing Load and Background Applications

What is cost of thread operations?

Operation	Arachne	Go	uThreads	std::thread
Thread Creation	320 ns	444 ns	6132 ns	13329 ns
Condition Variable Notify	272 ns	483 ns	4976 ns	4962 ns

What is cost of thread operations?

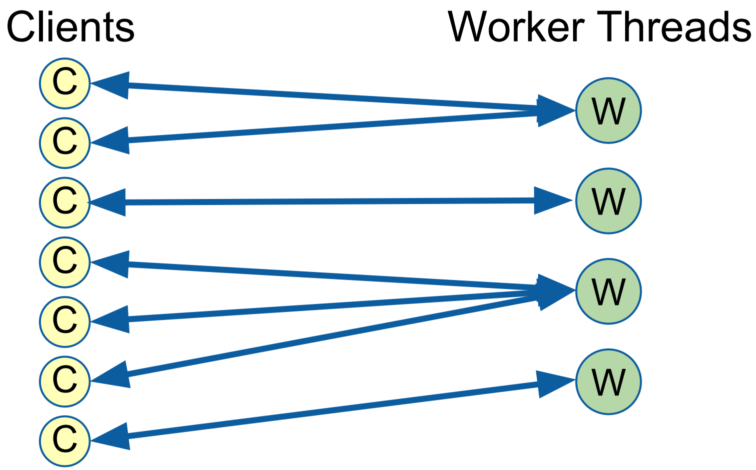
Child on different core, with load balancing

Child on same core

Operation	Arachne	Go	uThreads	std::thread
Thread Creation	320 ns	444 ns	6132 ns	13329 ns
Condition Variable Notify	272 ns	483 ns	4976 ns	4962 ns

Memcached Integration

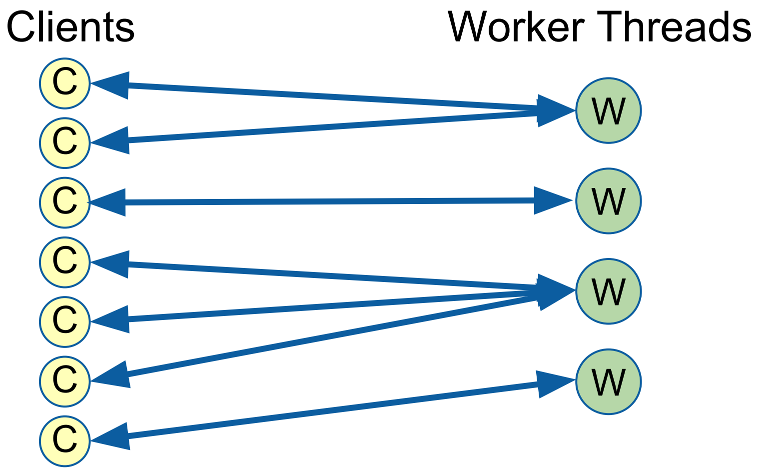
Before: Static Connection Assignment



Fixed pool of threads

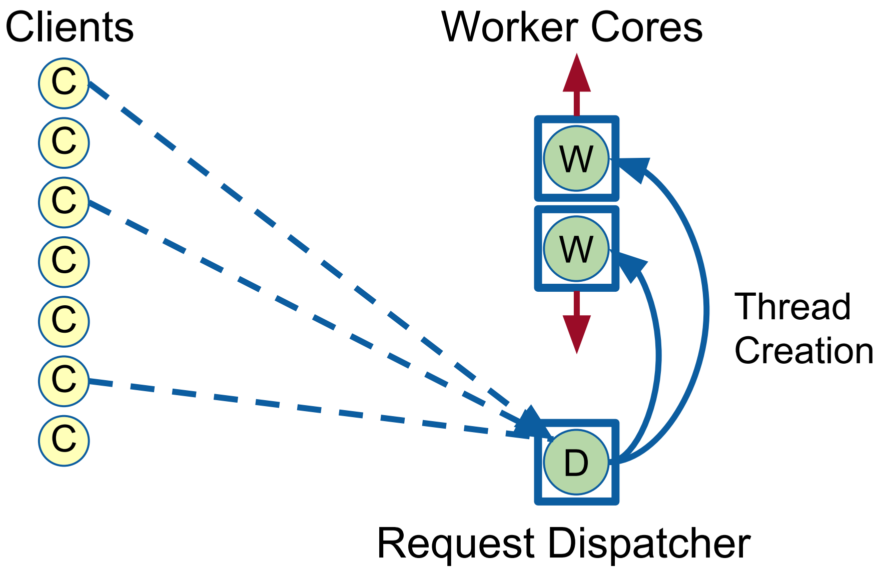
Memcached Integration

Before: Static Connection Assignment



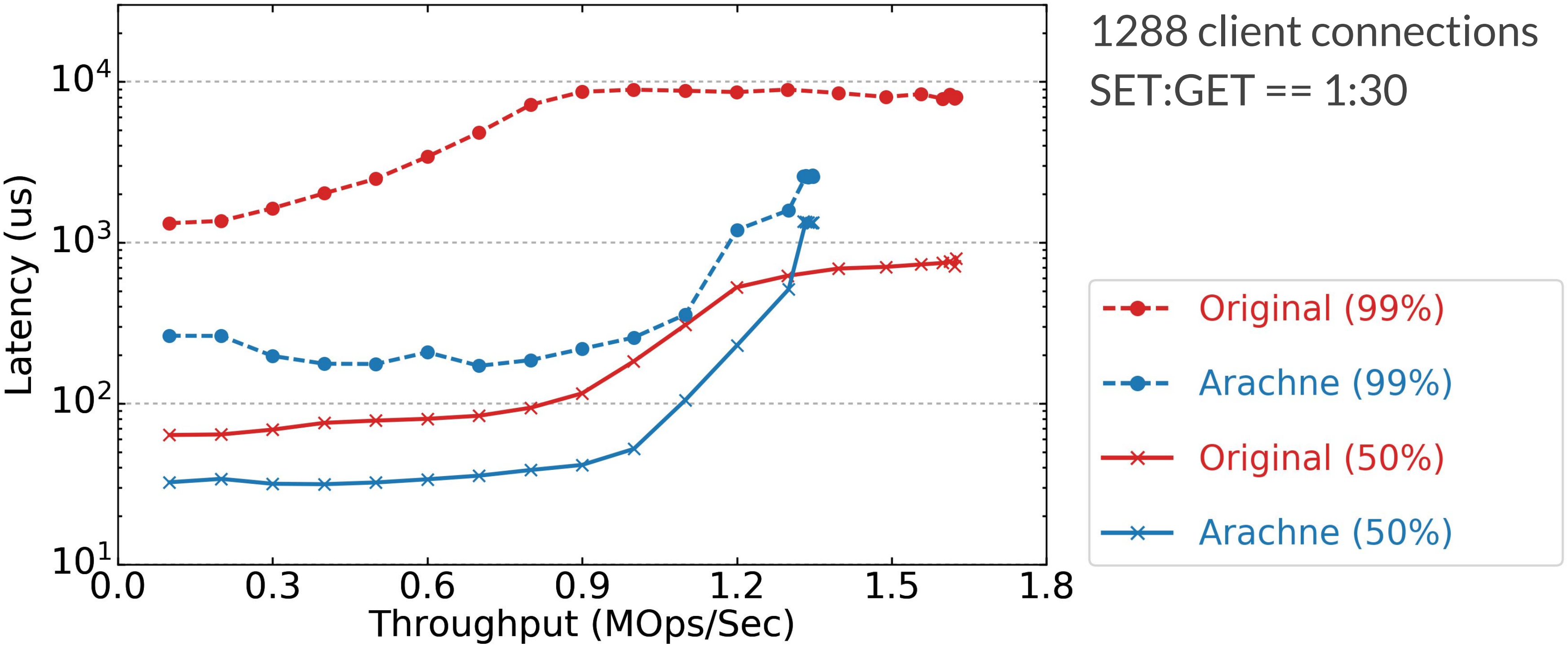
Fixed pool of threads

After: One Thread Per Request



Cores vary with load

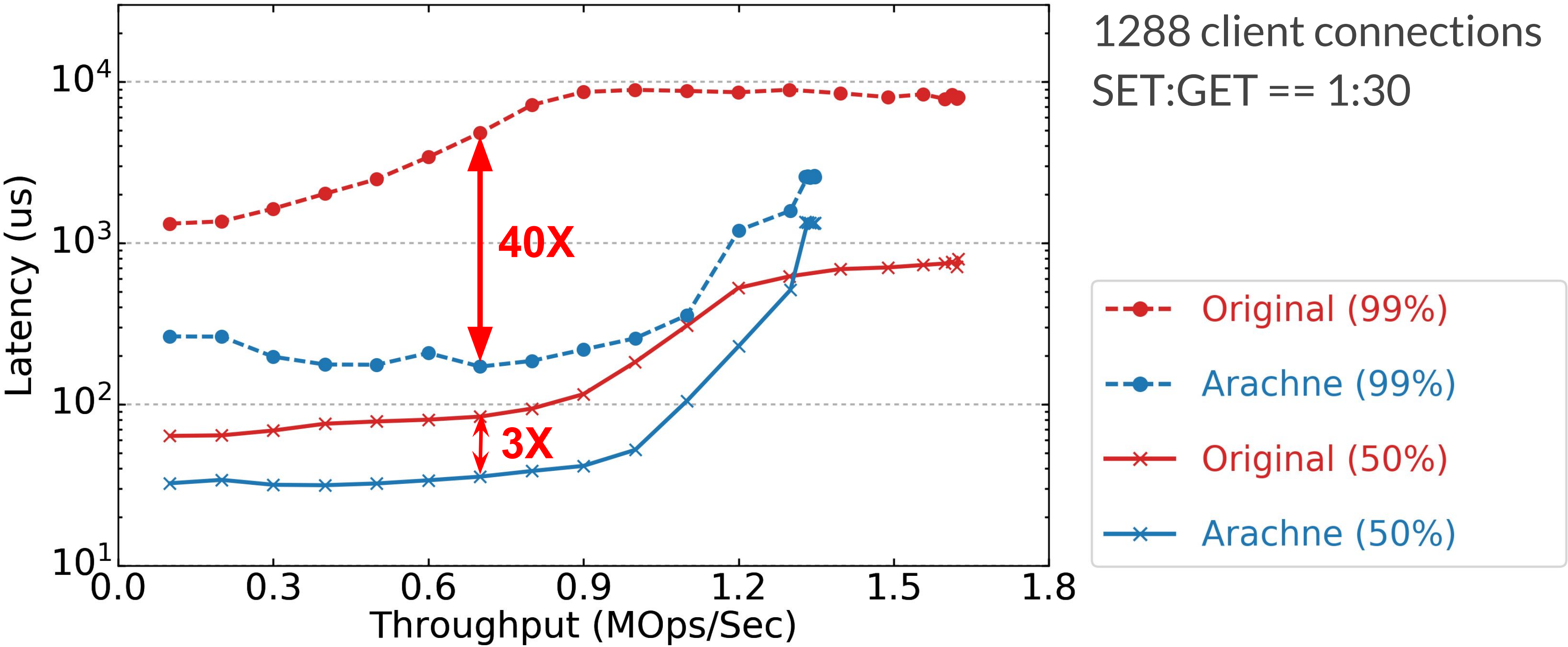
Memcached: Facebook ETC trace



1288 client connections

SET:GET == 1:30

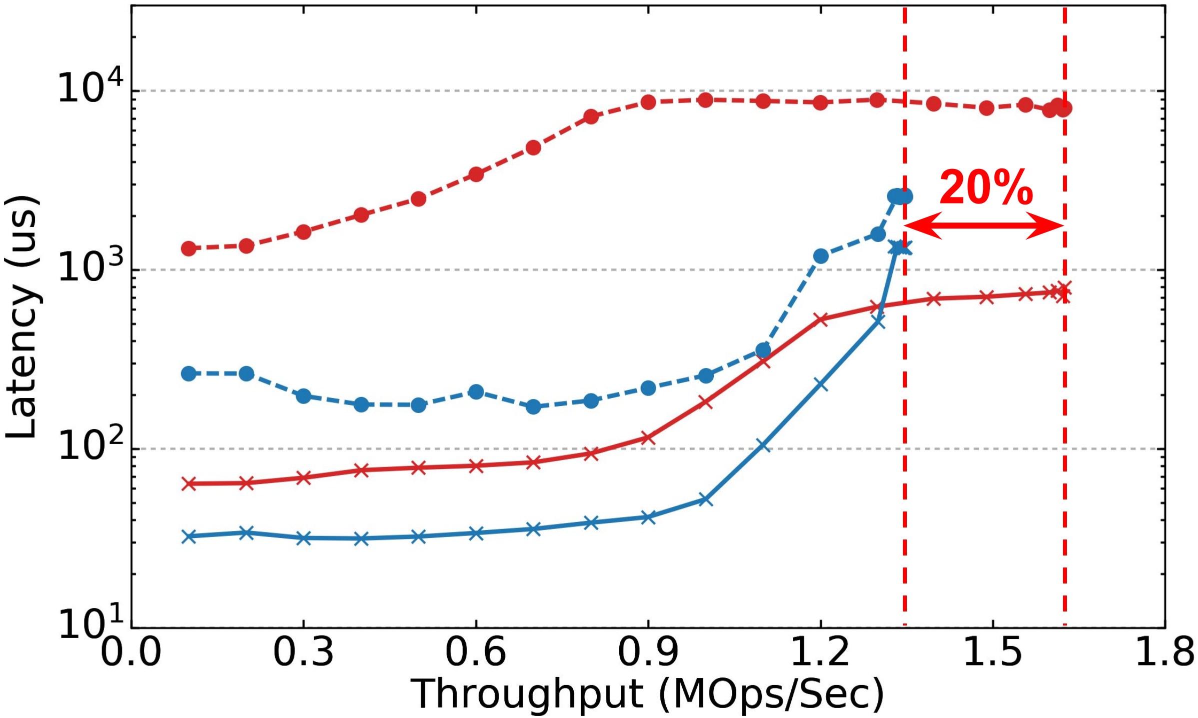
Memcached: Facebook ETC trace



1288 client connections

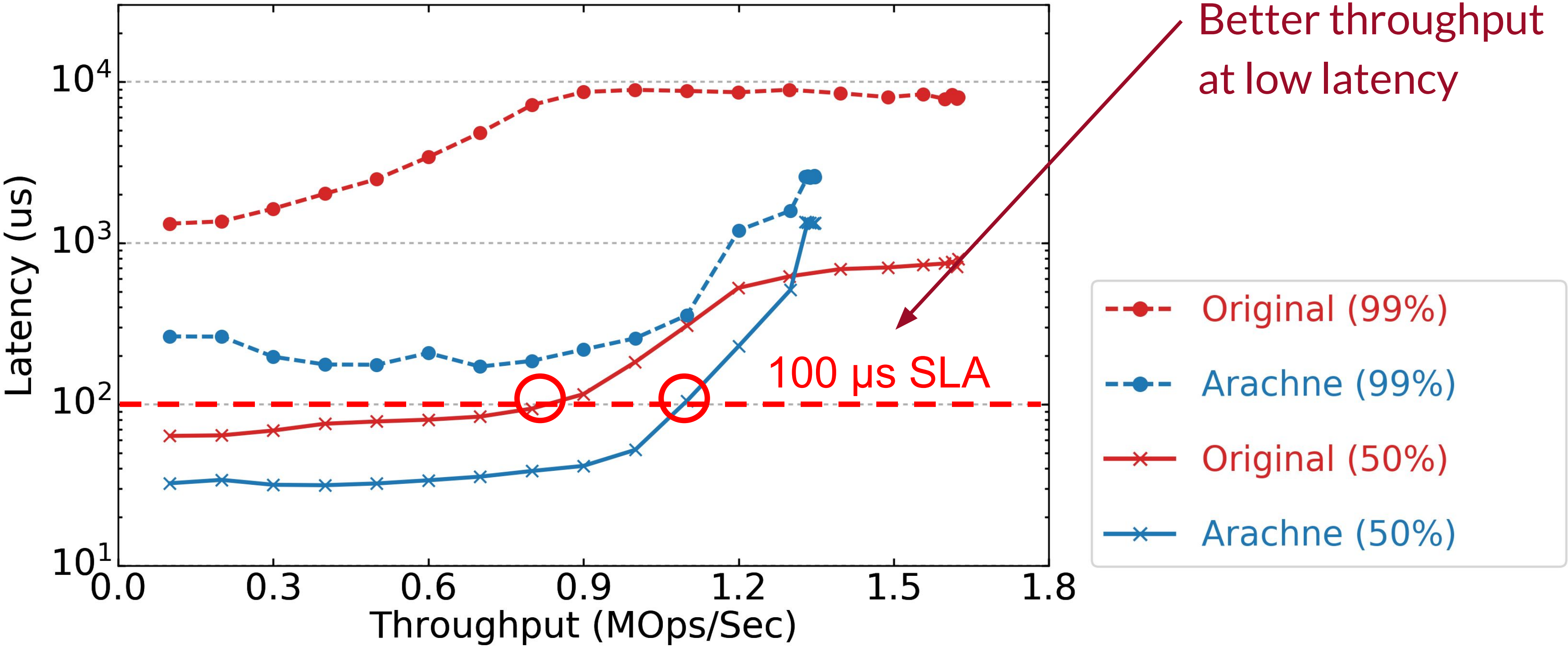
SET:GET == 1:30

Memcached: Facebook ETC trace



1288 client connections
SET:GET == 1:30

Memcached: Facebook ETC trace



Changing Load and Colocation

Does Arachne scale well with changing load?

Does Arachne enable high core utilization?

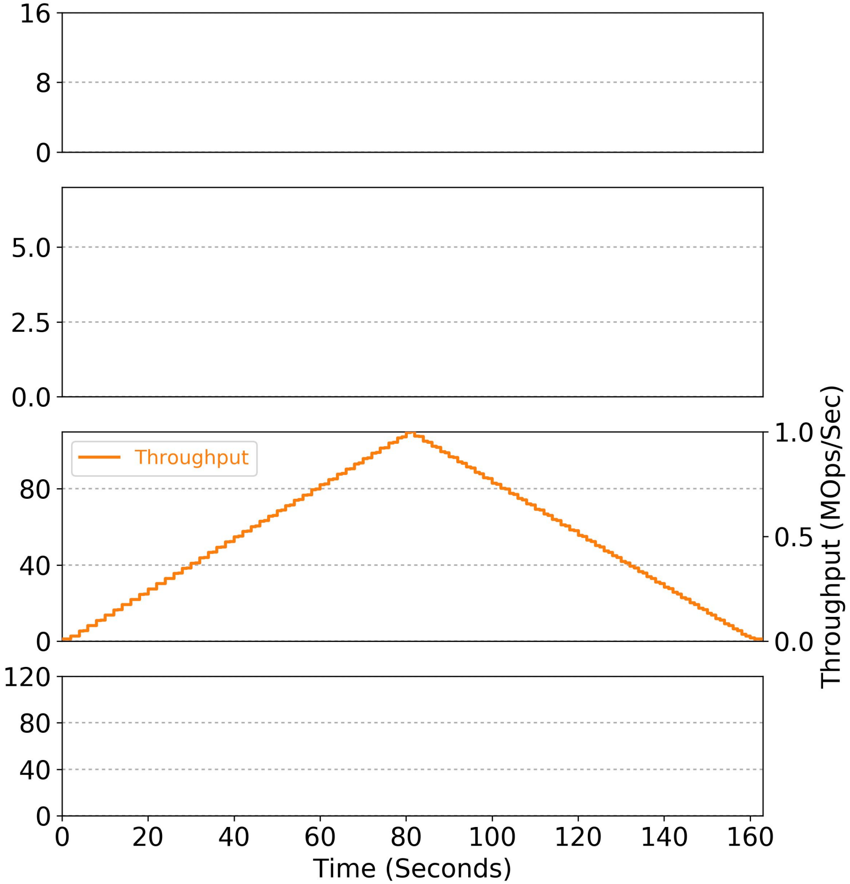
- Background app absorb unused resources
- Background app doesn’t interfere with memcached performance

Changing Load

Modified memtier

Poisson arrival rate

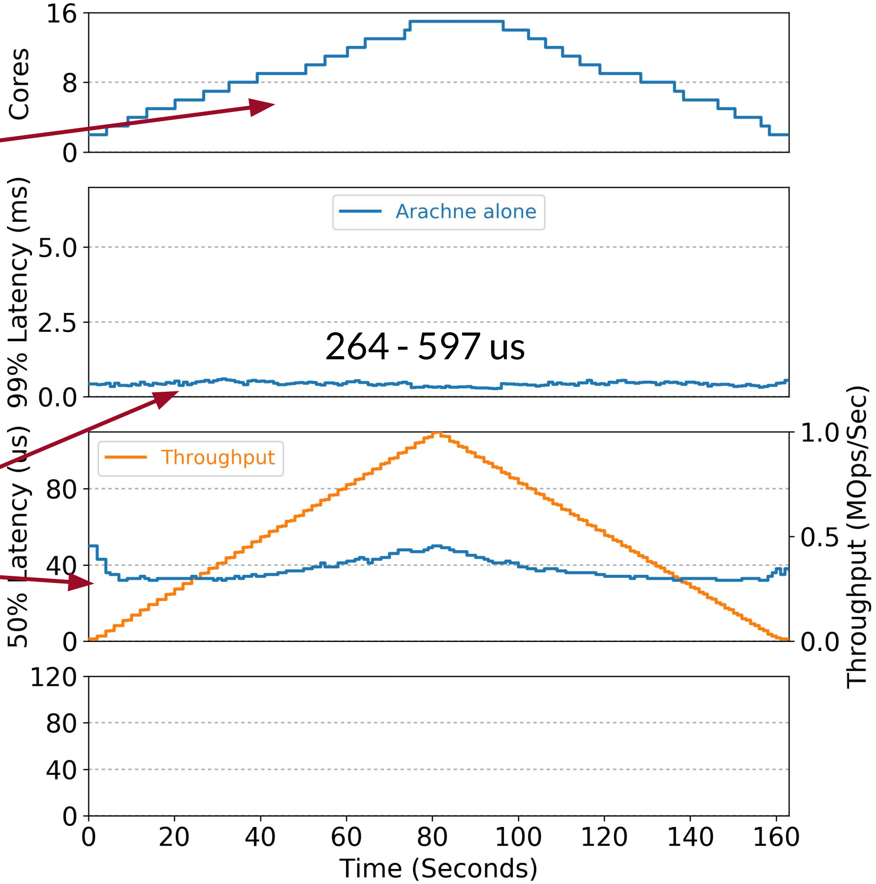
30B Keys, 200B values reads



Changing Load

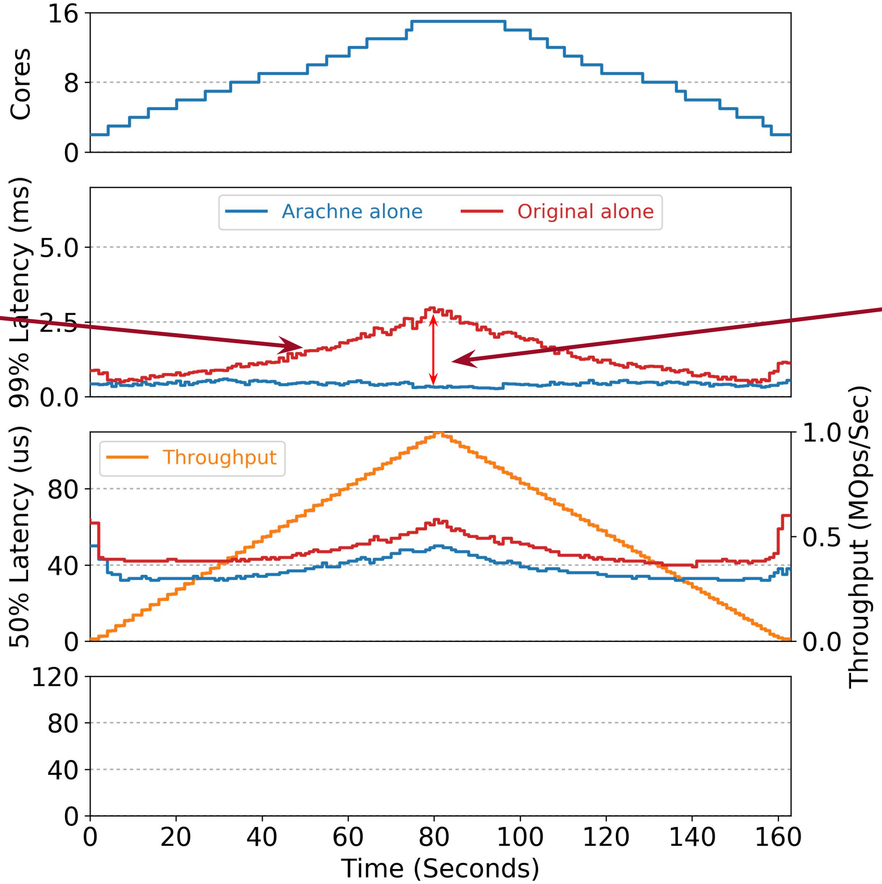
Cores scale with load

Nearly constant
median and tail latency



Changing Load

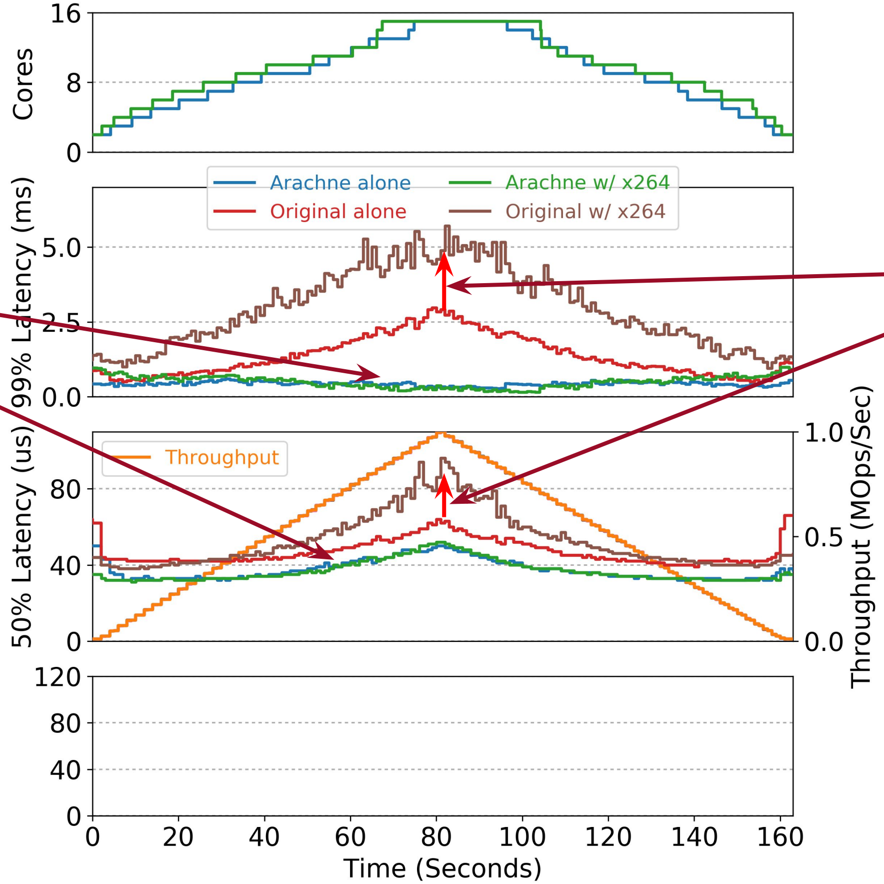
Tail latency increases with load



9x higher than Arachne at load

Colocated with x264 Video Encoder

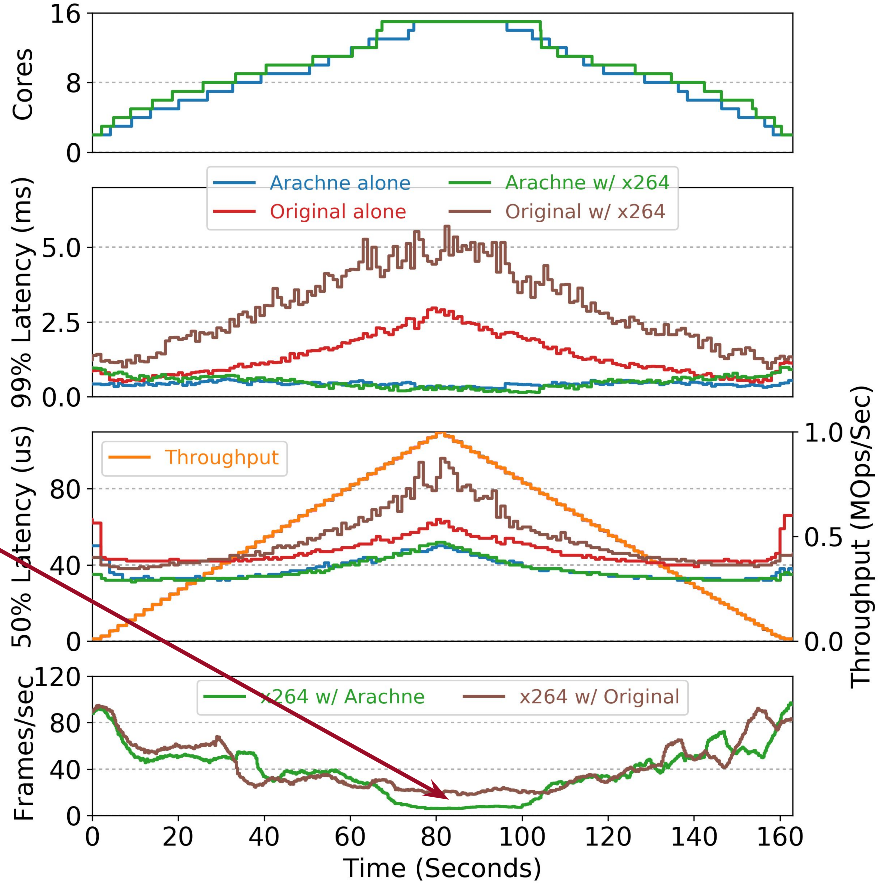
Arachne latency unchanged.



Memcached latency rises

Colocated with x264 Video Encoder

x264 throughput drops at high memcached load



Additional Experiments

- Memcached under a skewed workload
- RAMCloud write throughput
- RAMCloud under YCSB workload
- Thread creation scalability
- Comparison with a ready queue
- Arachne runtime without dedicated cores
- Cost of signaling a blocked thread
- Cost of allocating a core

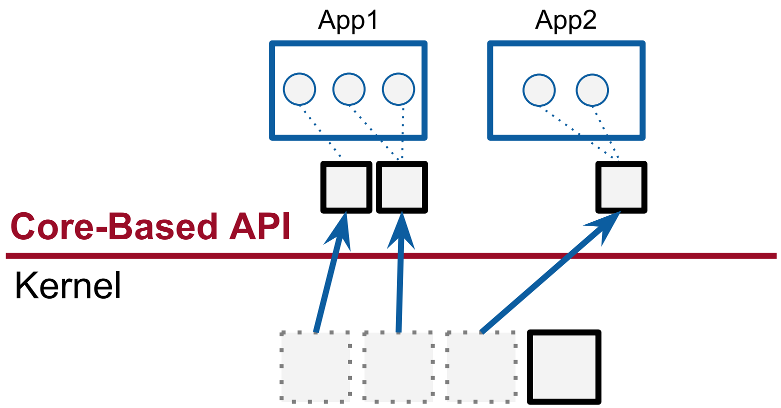
Conclusion

Arachne: core awareness for applications

- Applications request cores, not threads
- Application knows the exact cores it owns

Benefits

- Better combination of latency and throughput
- Efficient thread implementation



Questions?

github.com/PlatformLab/Arachne
github.com/PlatformLab/memcached-A

Poster #27