

Using Cohort Scheduling to Enhance Server Performance

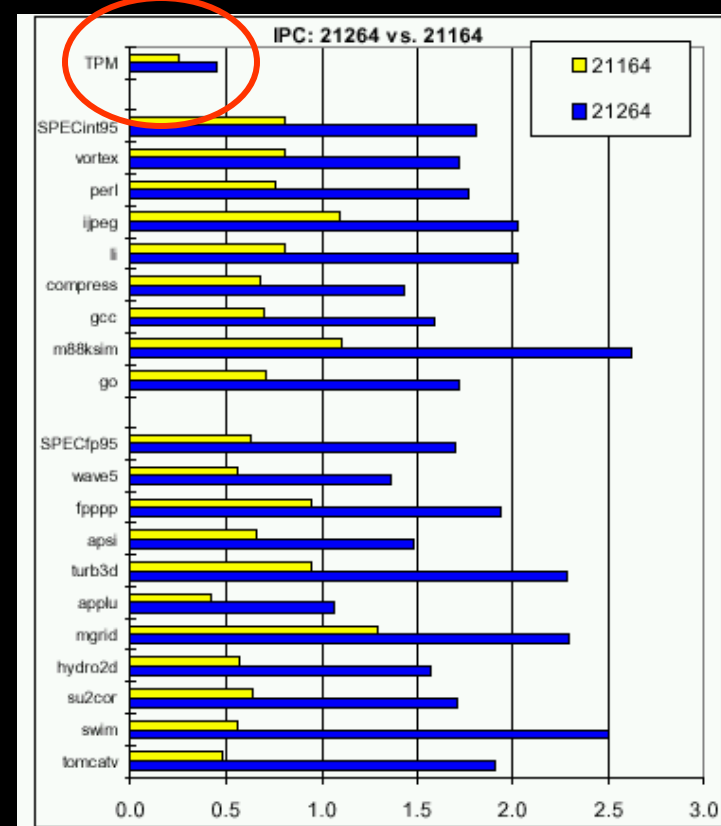
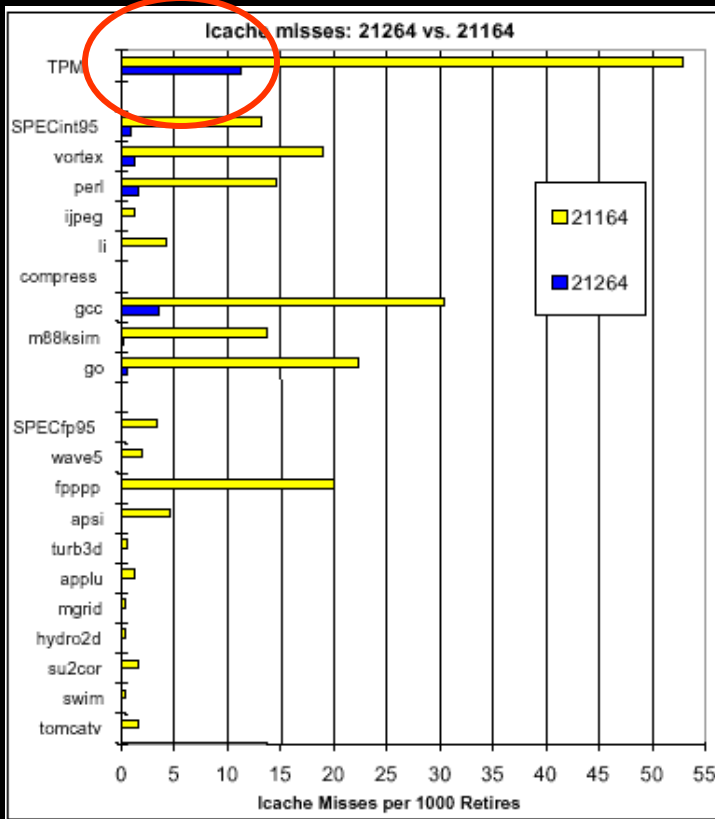
James Larus

[Joint work with Michael Parkes]

Microsoft Research

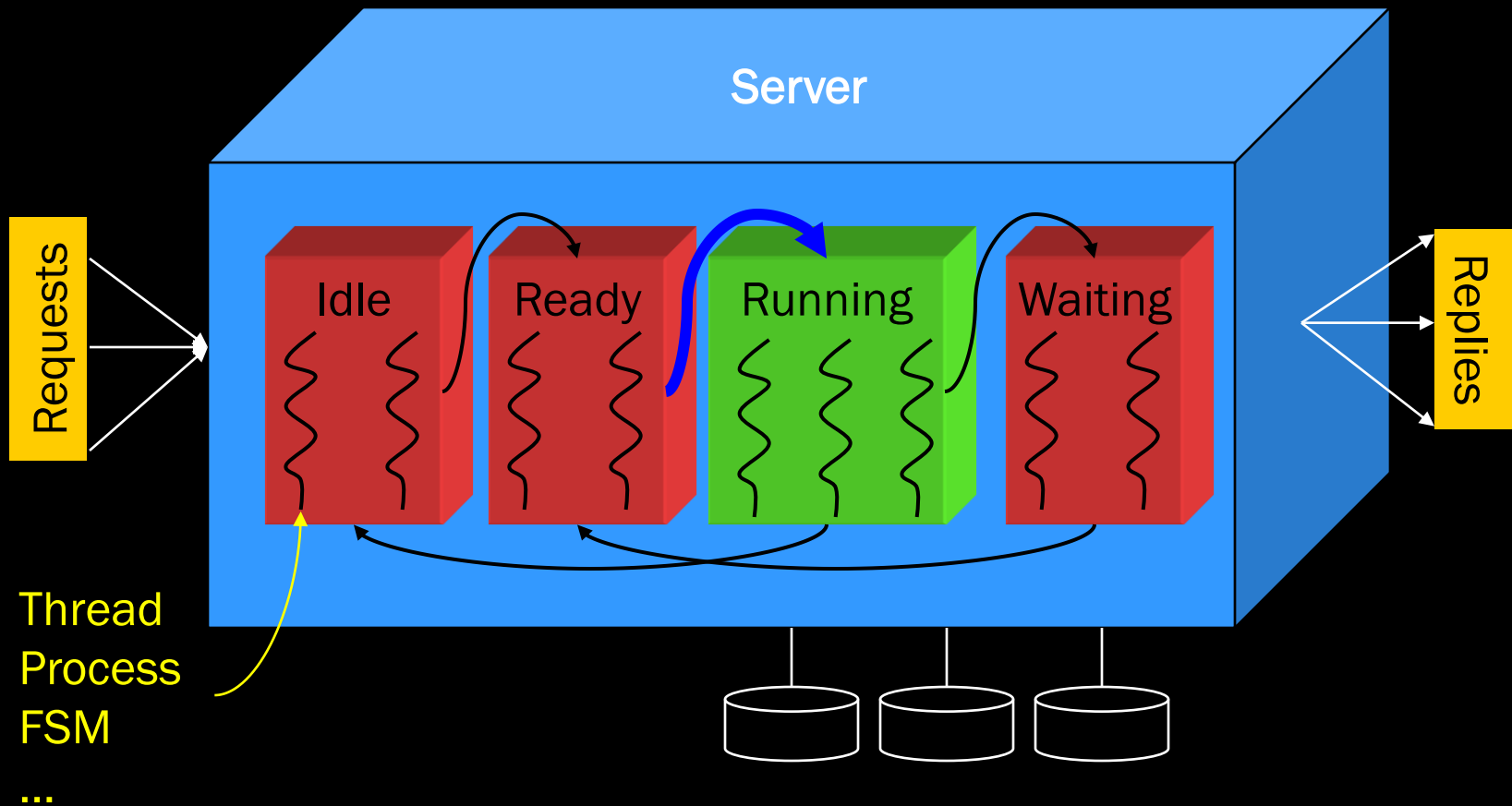
March 2001

ILP of Databases

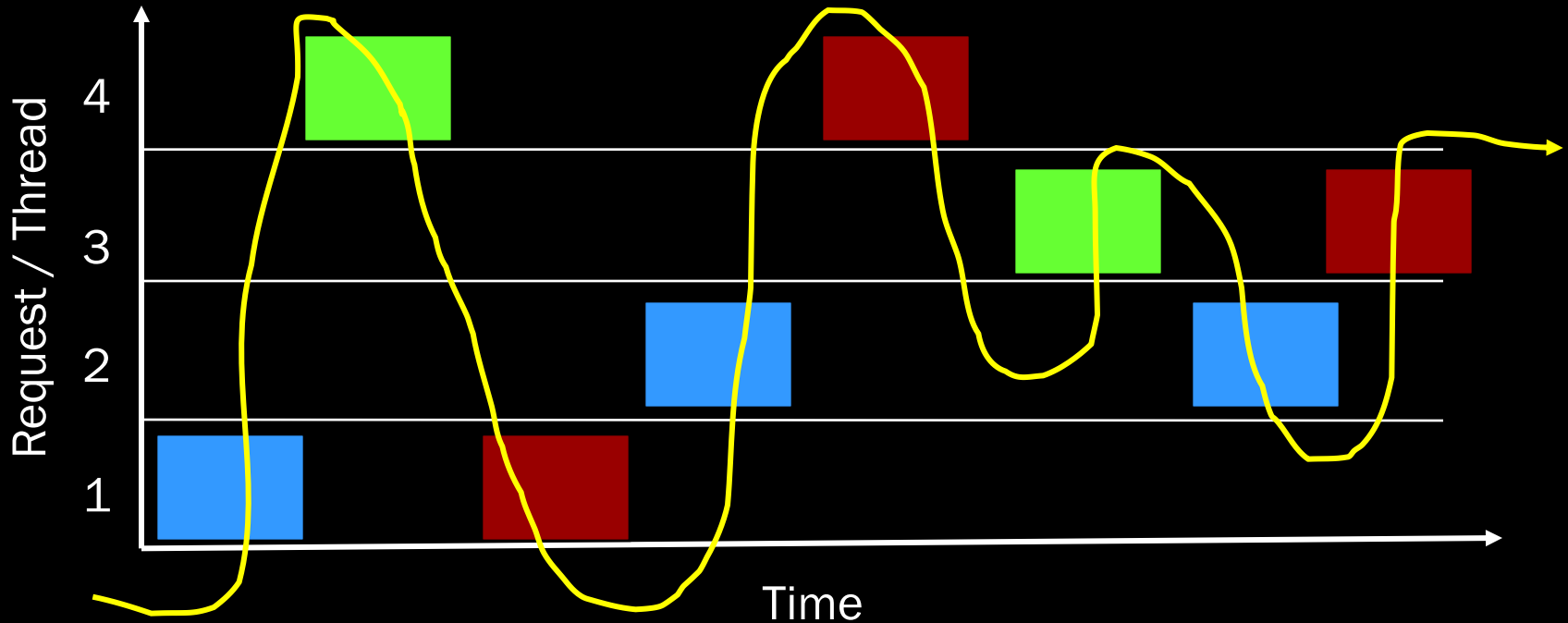


[Cvetanovic & Kessler, ISCA 2000]

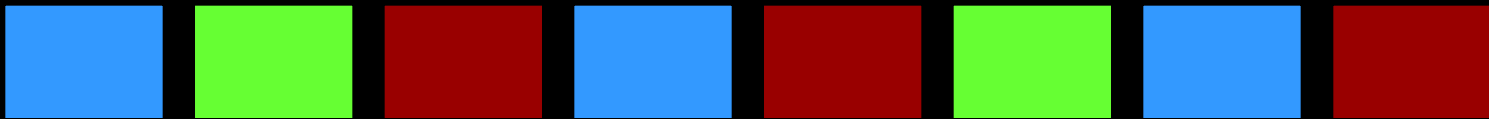
Server Software Architecture



Processor Scheduling



Processor



Threads and Locality

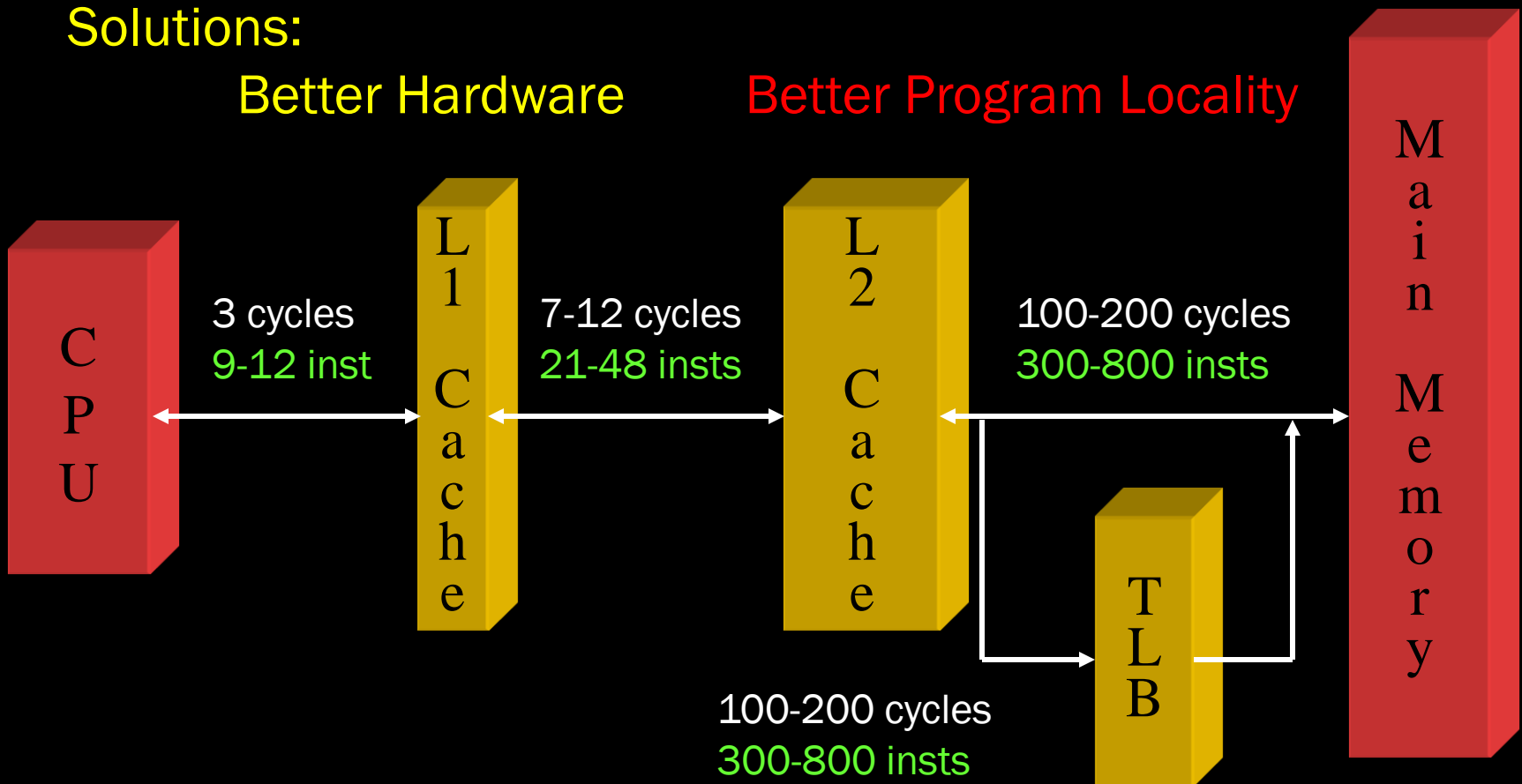
- Short intervals have poor locality [Barroso ISCA98]
 - 25K inst (TPC-B) → 7 CPI
 - 1.7M inst (TPC-D) → 1.6 CPI
- Costly context switches [Borg, ASPLOS91]
 - 400K inst shadow (process)
- Inter-processor cache conflicts and traffic
 - 5-20% loads hit dirty data in another L2 cache [Keeton, ISCA98]

Processor-Memory Hierarchy

Solutions:

Better Hardware

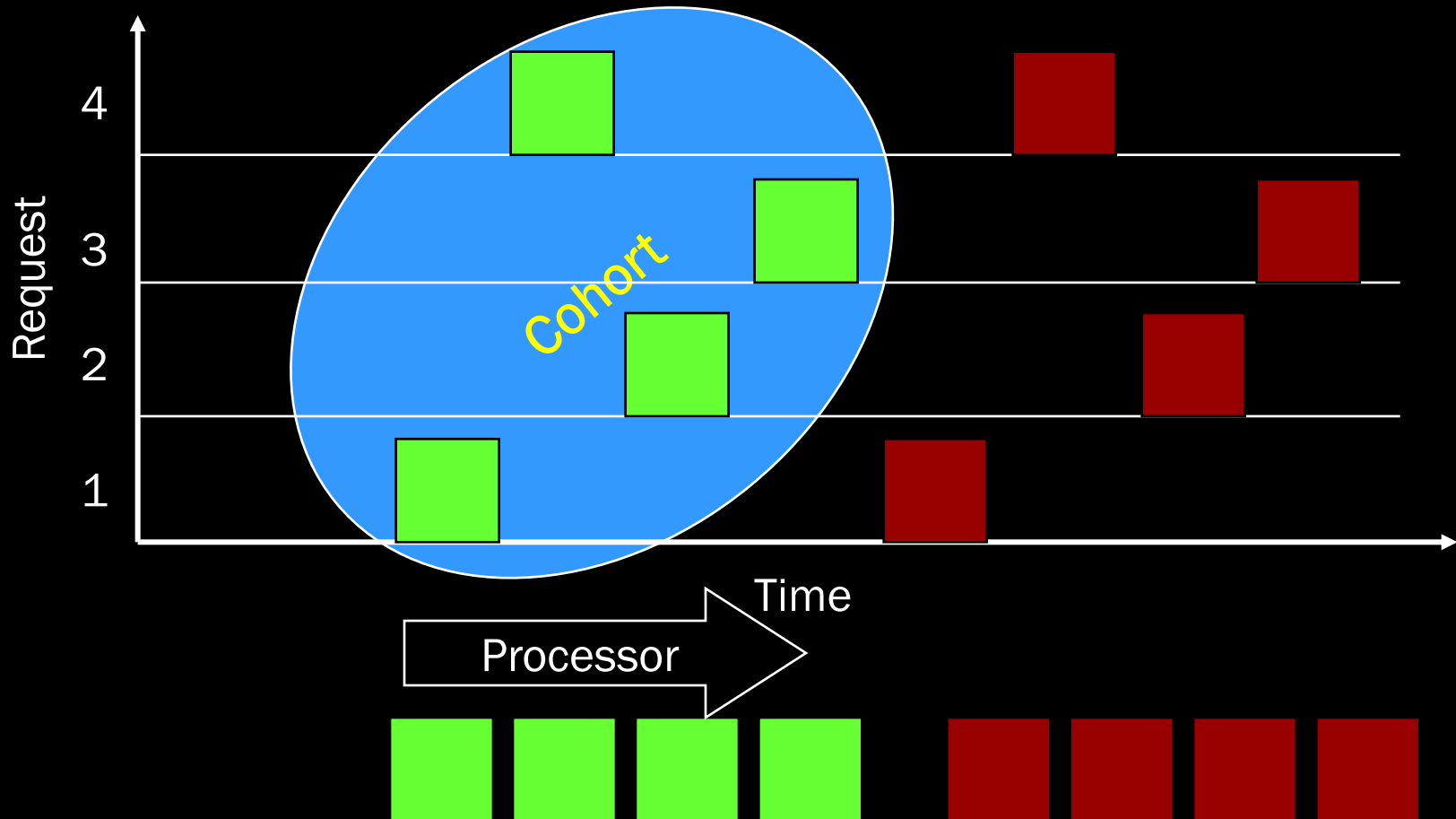
Better Program Locality



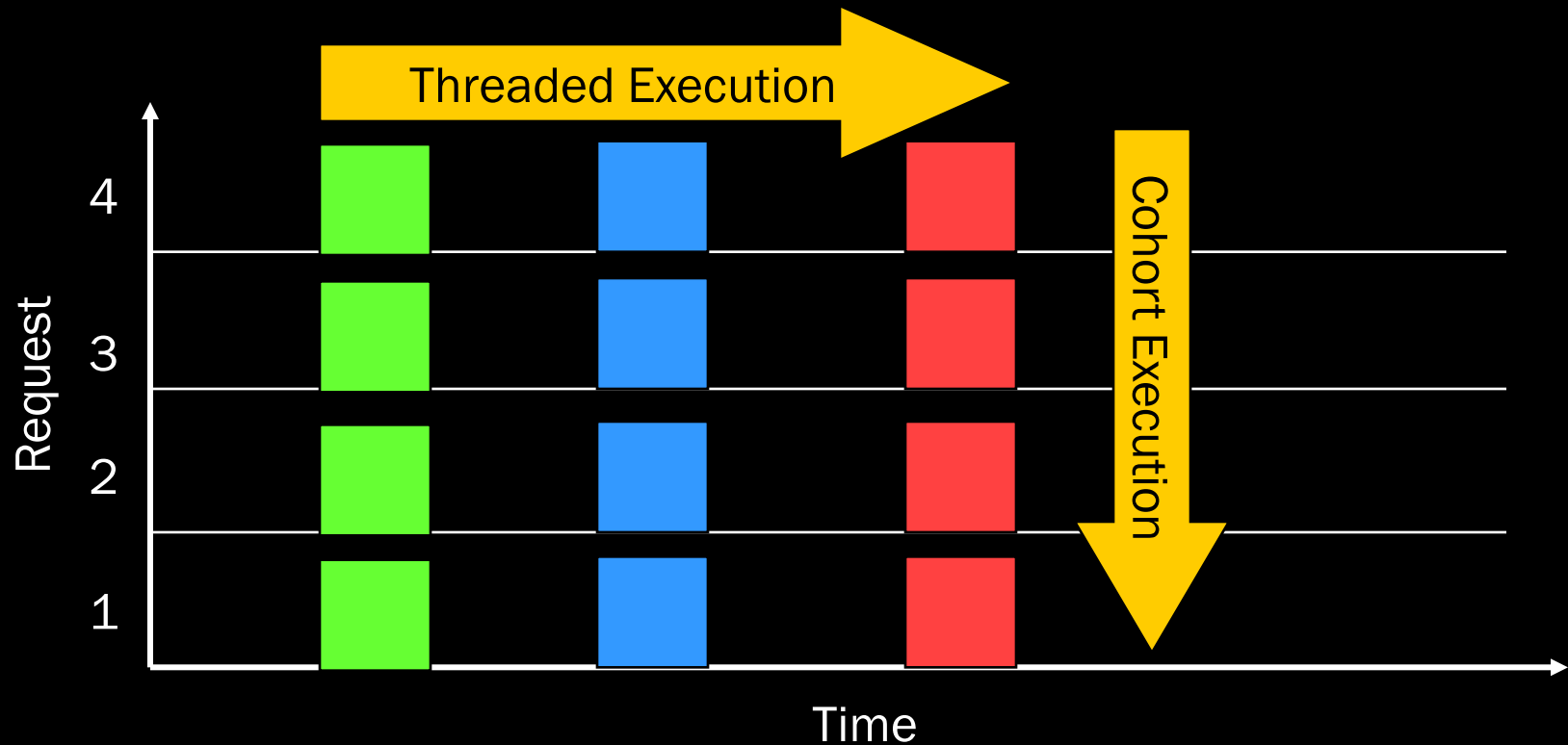
Talk Outline

- Cohort scheduling
- Staged computation
- StagedServer library
- Experiments

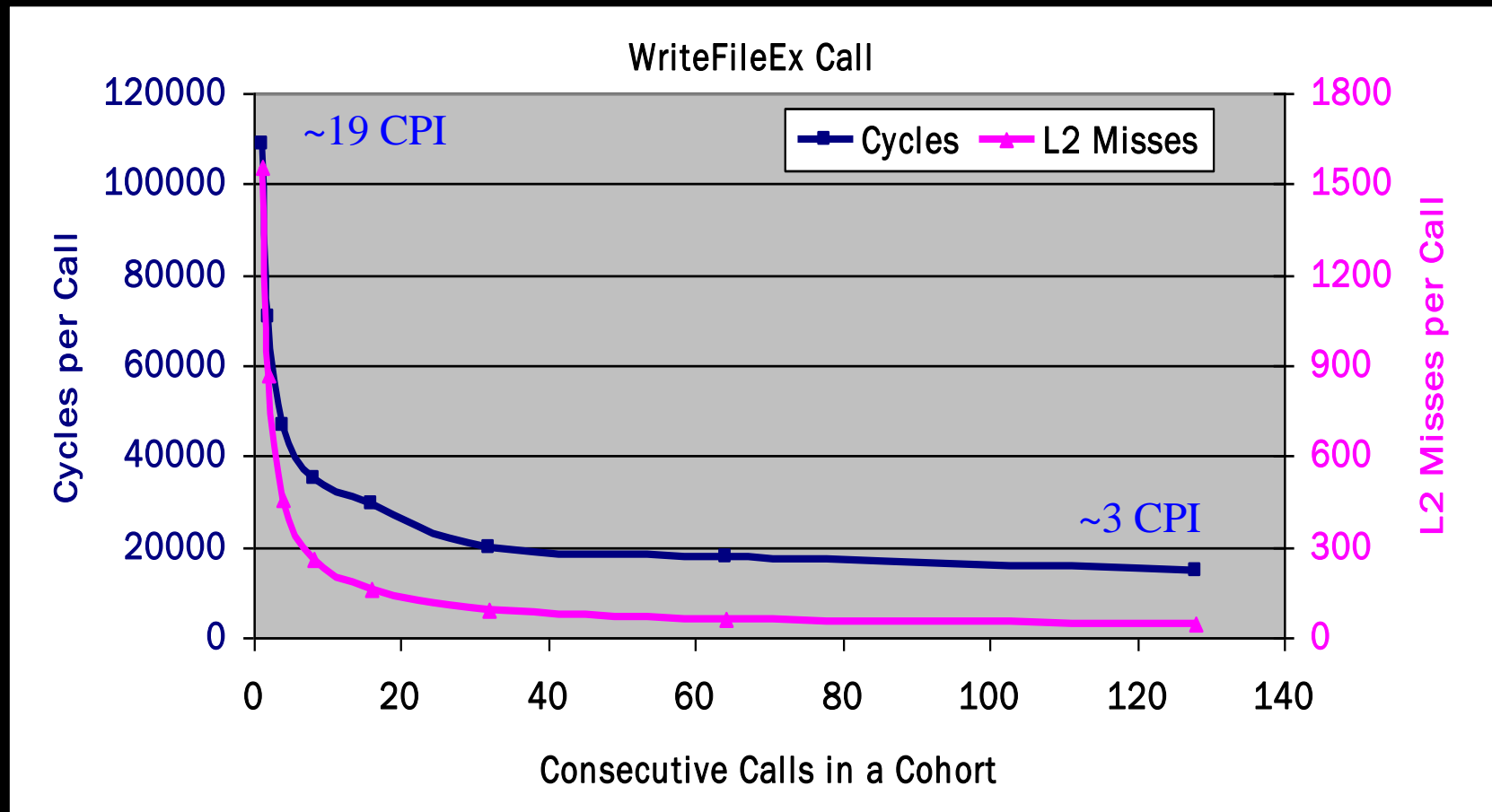
Cohort Scheduling



Another View of Cohorts



Cohort Scheduling Experiment



Aside: Cohort Schedule Thread?

- Cohort resumes execution at same PC
 - Schedule change
 - No programmer-visible changes
- Wrong boundaries
 - Cohort formed **after** system call
- Missed opportunities
 - Data structures accessed from many locations

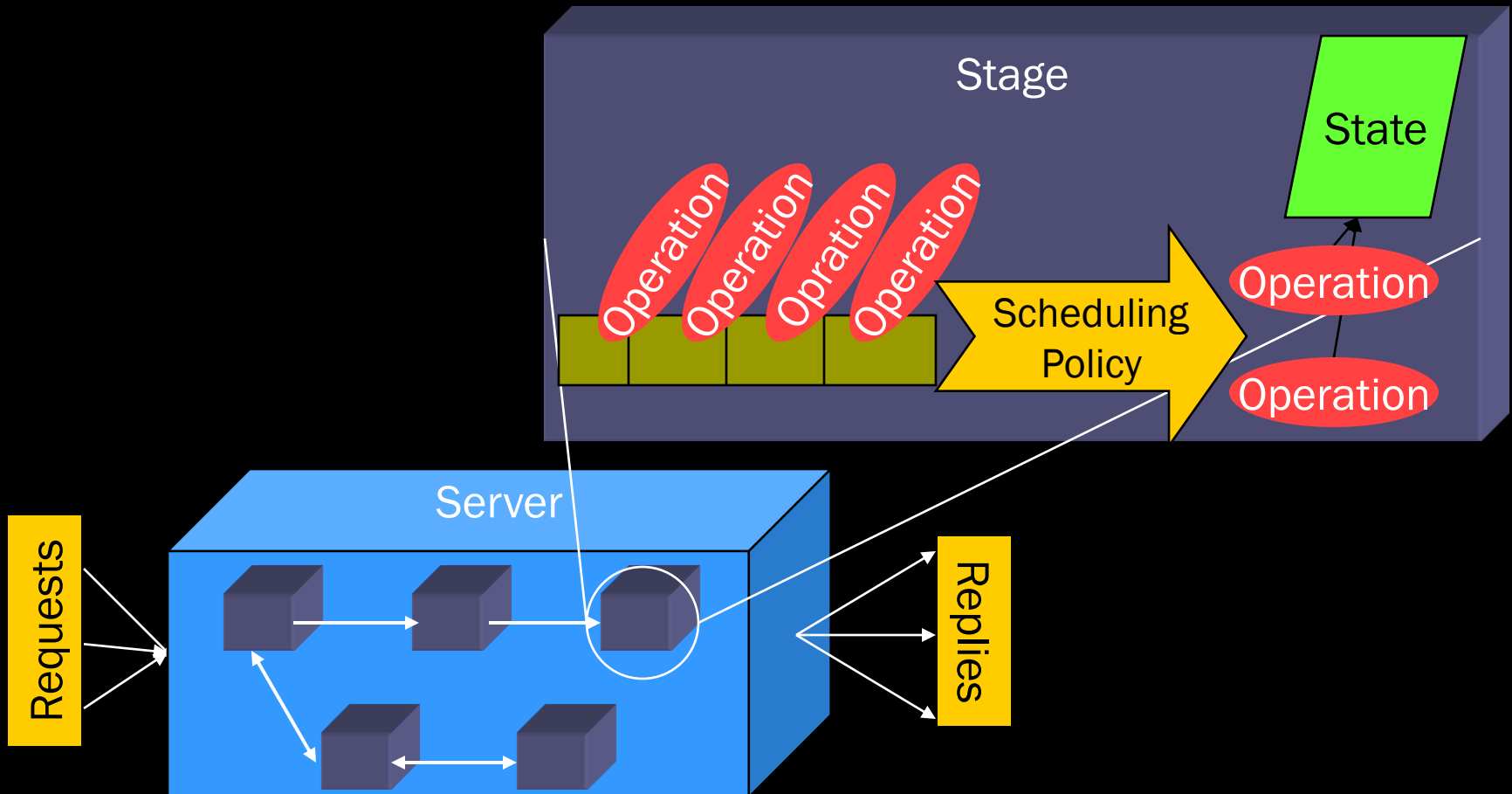
Talk Outline

- Cohort scheduling
- **Staged computation**
- StagedServer library
- Experiments

Motivation

- Programming model to support cohort scheduling
- Lack of structure in threads
- Expensive, error-prone synchronization

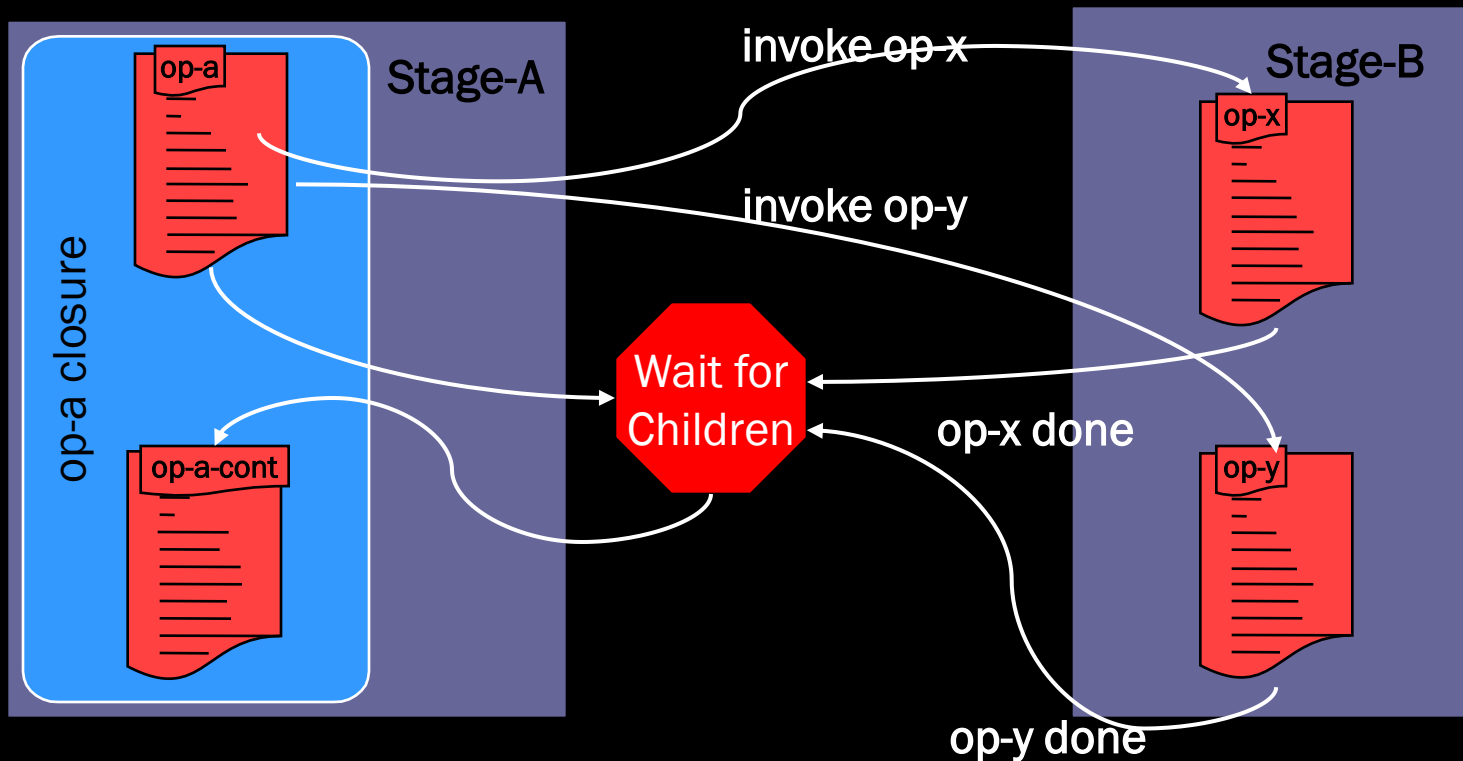
Staged Computation



Staged Programming Model

- Alternative to thread, processes, FSM
- Facilitate cohort scheduling
 - Natural abstraction for cohorts
 - Scheduling flexibility
- Reduce synchronization
- **StagedServer** library

Staged Computation Example



Stages

- Operations
 - Asynchronous, non-preemptible computations
- State
 - Private to stage
- Scheduling policy
 - When and how operations execute
 - Control concurrency within stage

Stages, cont'd

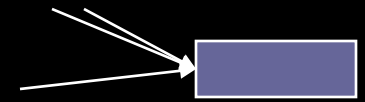
- Similar to object, but
 - Operations are asynchronous
 - Scheduling autonomy
- Natural cohort
 - Group logically related computation
 - Access share code and data

Scheduling

- Scheduling can supplant synchronization

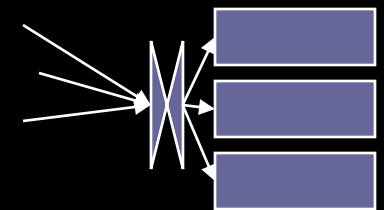
- **Exclusive stage**

- Execute one operation on one processor at a time
- Access local data without synchronization



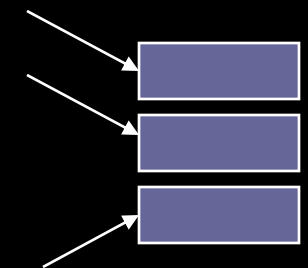
- **Partitioned stage**

- Send operations to processor based on key
- Processor can access local data w/o sync

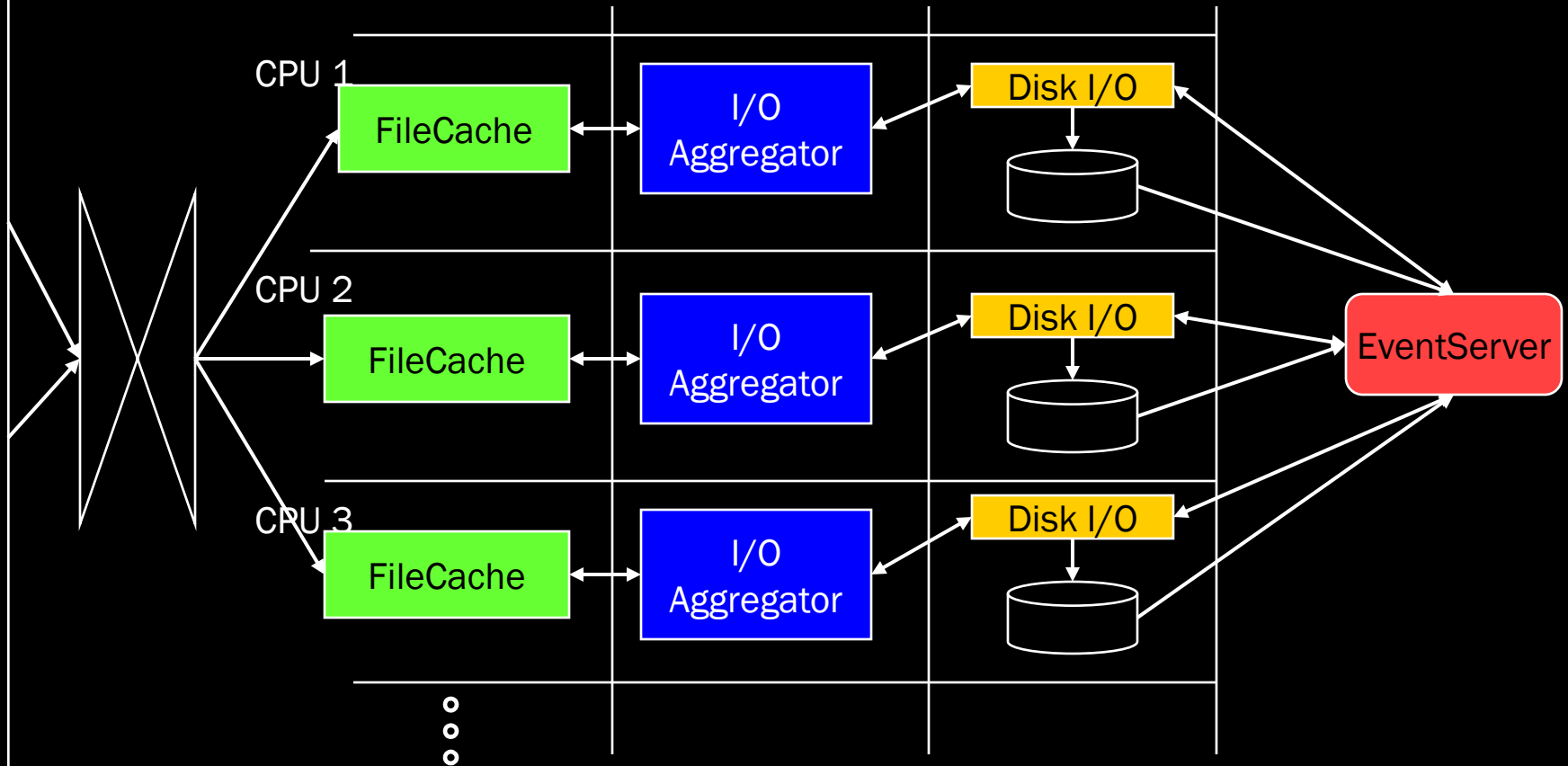


- **Shared stage**

- Operations run on all processors



Staged File Cache



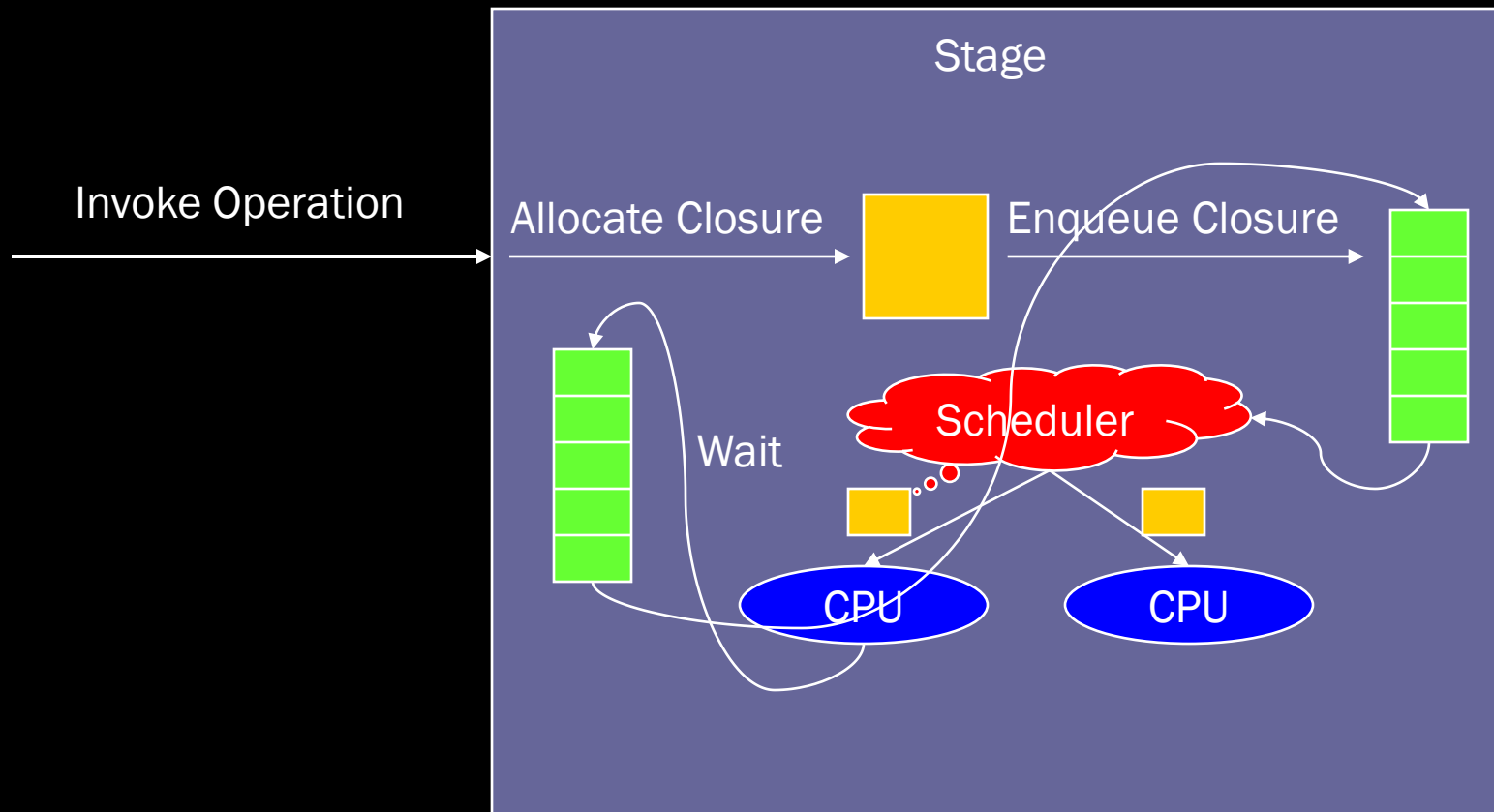
Talk Outline

- Cohort scheduling
- Staged computation
- **StagedServer library**
- Experiments

StagedServer

- C++ library
 - Uniprocessor or SMP
 - Mechanism for staged computation
 - Aggressive cohort scheduling
- Two parameterized classes
 - Stage
 - Closure

Stages and Closures



Stage Constructor

```
STAGE(const char *Name,  
      STAGE_TYPE Type,  
      bool BalanceLoad = false,  
      int CacheSize = 0,  
      int BatchThreshold = 0,  
      int BatchTimer = DefaultTimer,  
      bool MaintainOrder = false,  
      int MaxBatchSize = StageBatchSize)
```

Operation #1

```
ACTIONS WEB_CLOSURE::EstablishConnection()
{
    NetworkStage->CreateIncomingConnection(&NWCreateResult);

    return WaitForChildren(ReadRequest);
}
```

Operation #2

```
ACTIONS WEB_CLOSURE::ReadRequest()
{
  if (0 == NWCreateResult->LastError)
  {
    ConnectionNumber = NWCreateResult->ConnectionNumber;
    NetworkStage->ReadFromConnection(&NWReadWriteResult,
                                     ConnectionNumber,
                                     StrBuffer,
                                     sizeof(StrBuffer));

    return WaitForChildren(ParseRequest);
  }
  else
    return EstablishConnection();
}
```

Closure

```
NETWORK_STAGE::CreateIncomingConnection(RESULT<CR> *Result)
{
    static int roundRobin = 0;
    NETWORK_CLOSURE* x =
        new(NETWORK_CLOSURE::CreateIncomingConnection,
            this,
            Result,
            roundRobin++)
            NETWORK_CLOSURE( );

    x->Start( );
}
```

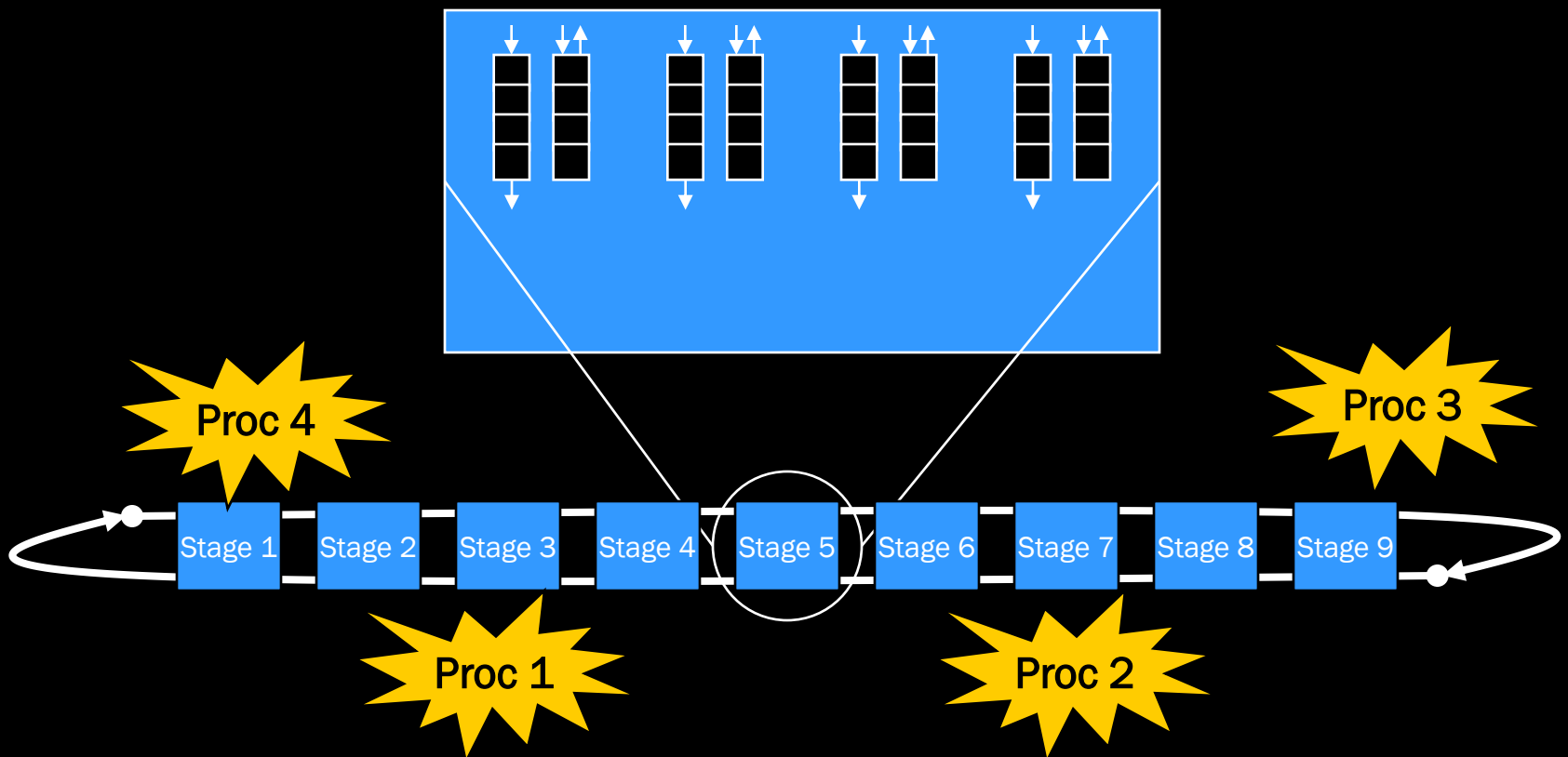
Aggressive Cohort Scheduling

- Processor affinity
 - Operation and children stay on processor
 - Ex: explicit placement, partitioning, load balancing
- Cohort scheduling
 - Per-processor, per-stage queue
 - Processor execute all operations in its queue
 - Ex: fixed cohort size

Processor Queues

- Pair of 'queues'
 - Stack for local operations
 - No synchronization
 - Queue for remote operations
 - Process stack LIFO then queue

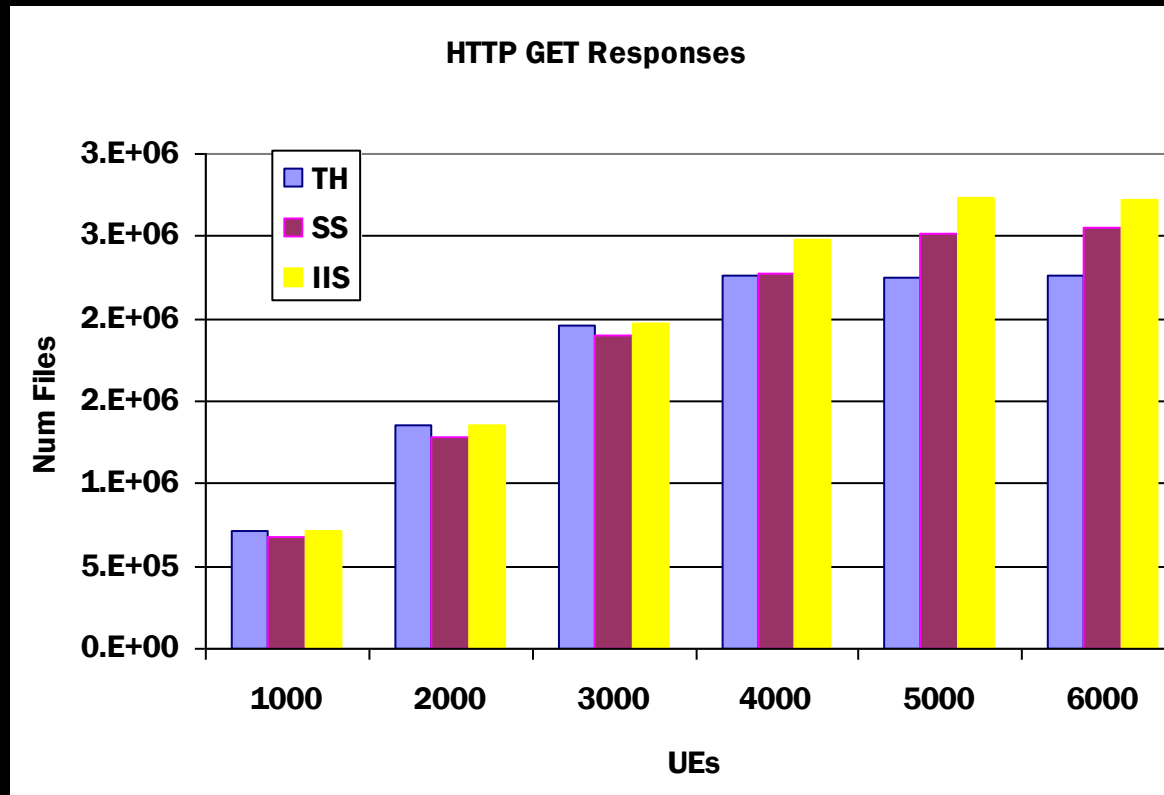
Wavefront Processor Scheduling



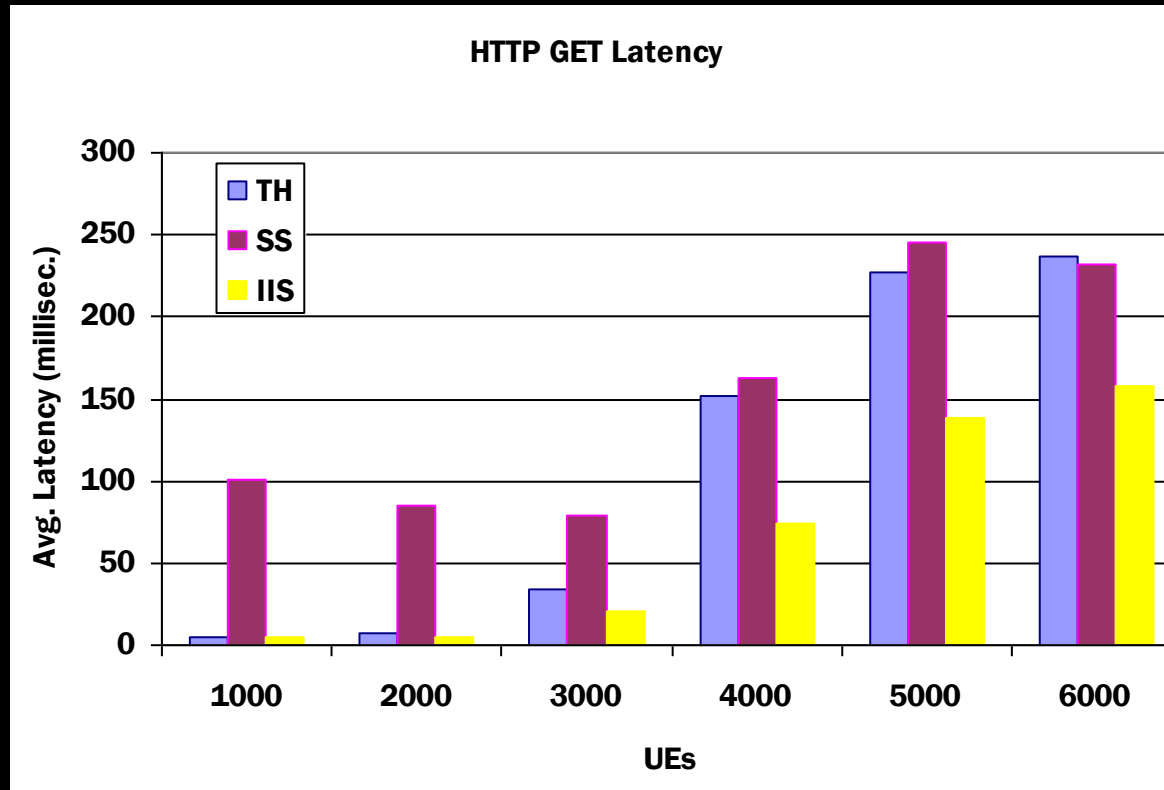
Talk Outline

- Cohort scheduling
- Staged computation
- StagedServer library
- Experiments

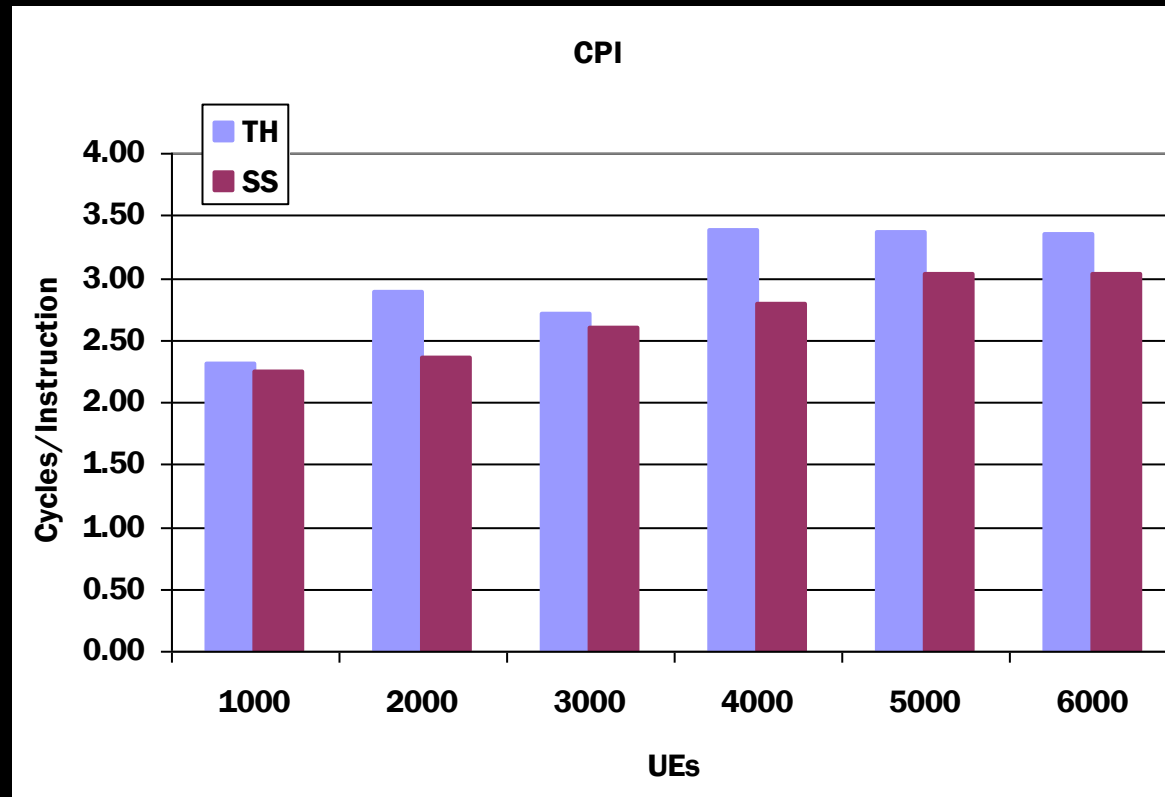
Web Server Bandwidth



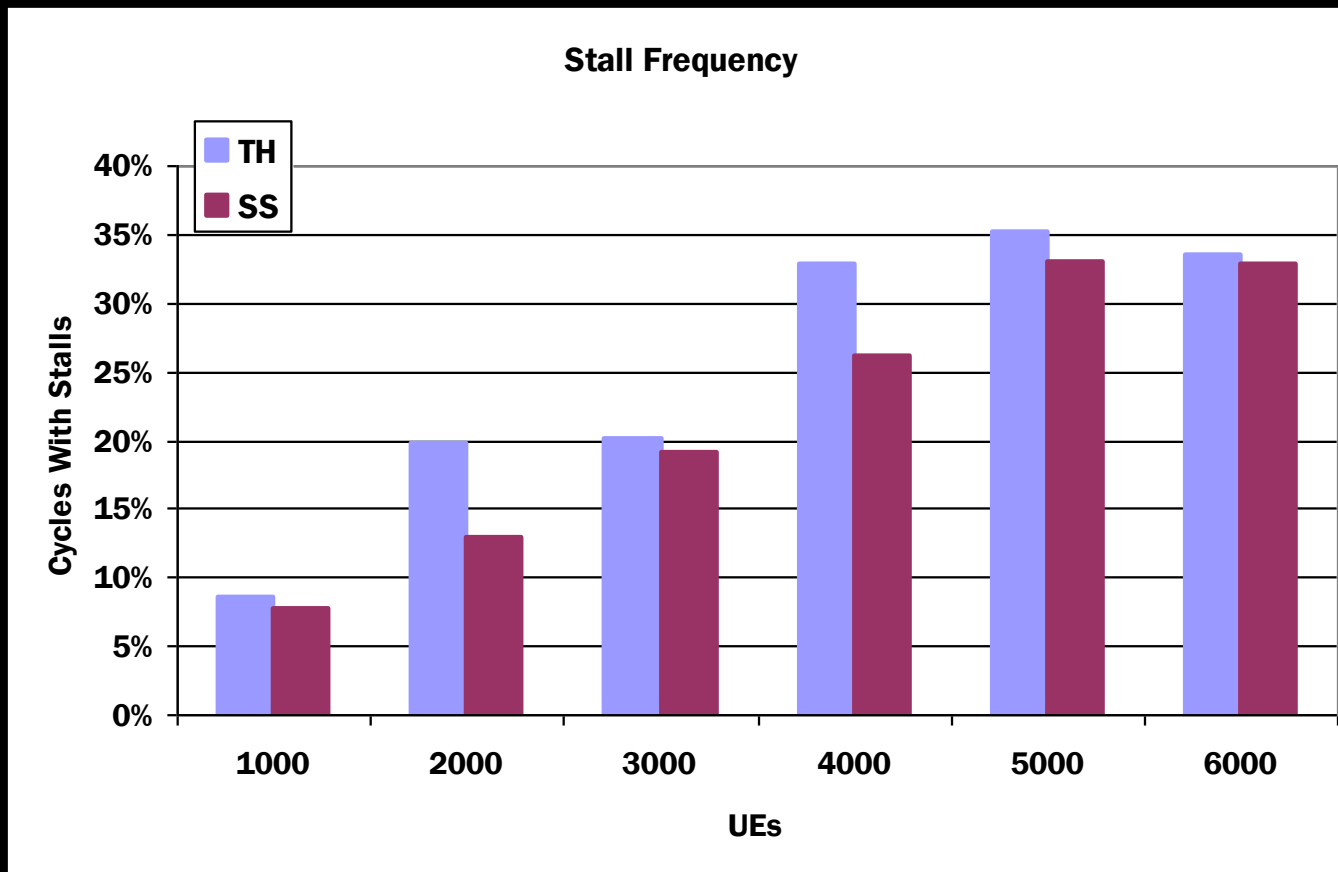
Web Server Latency



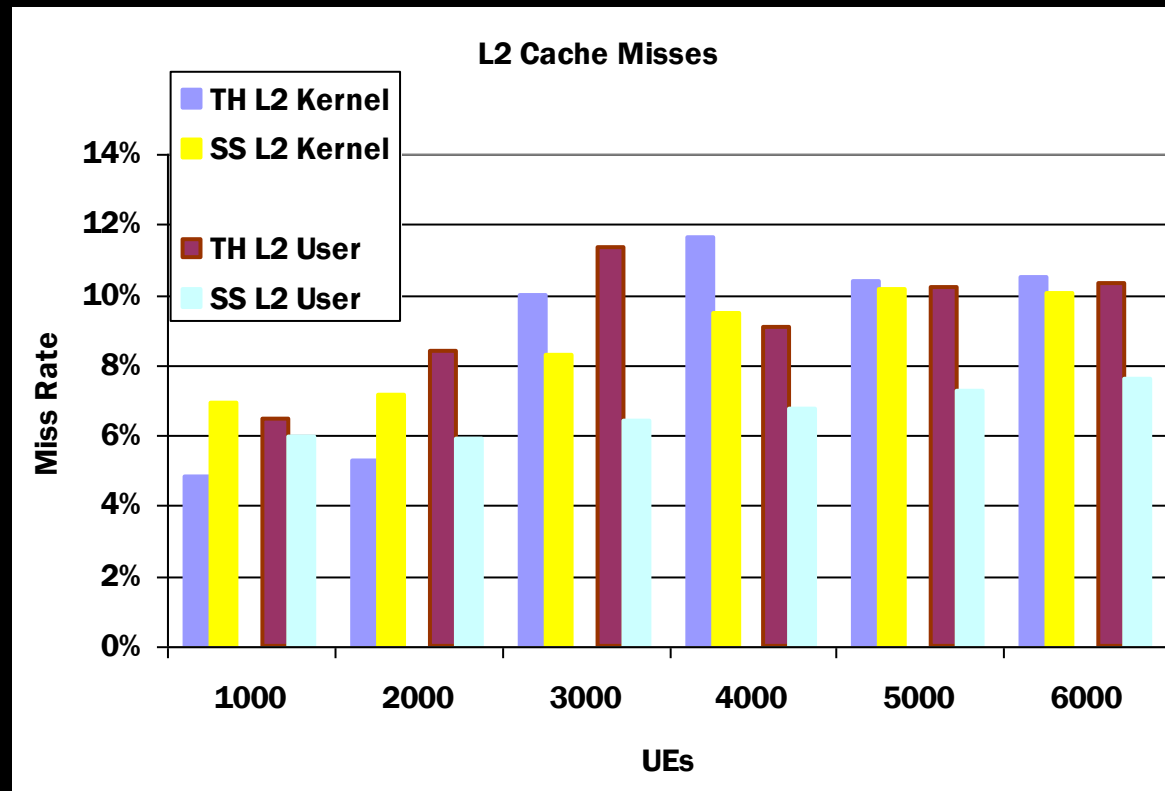
Server CPI



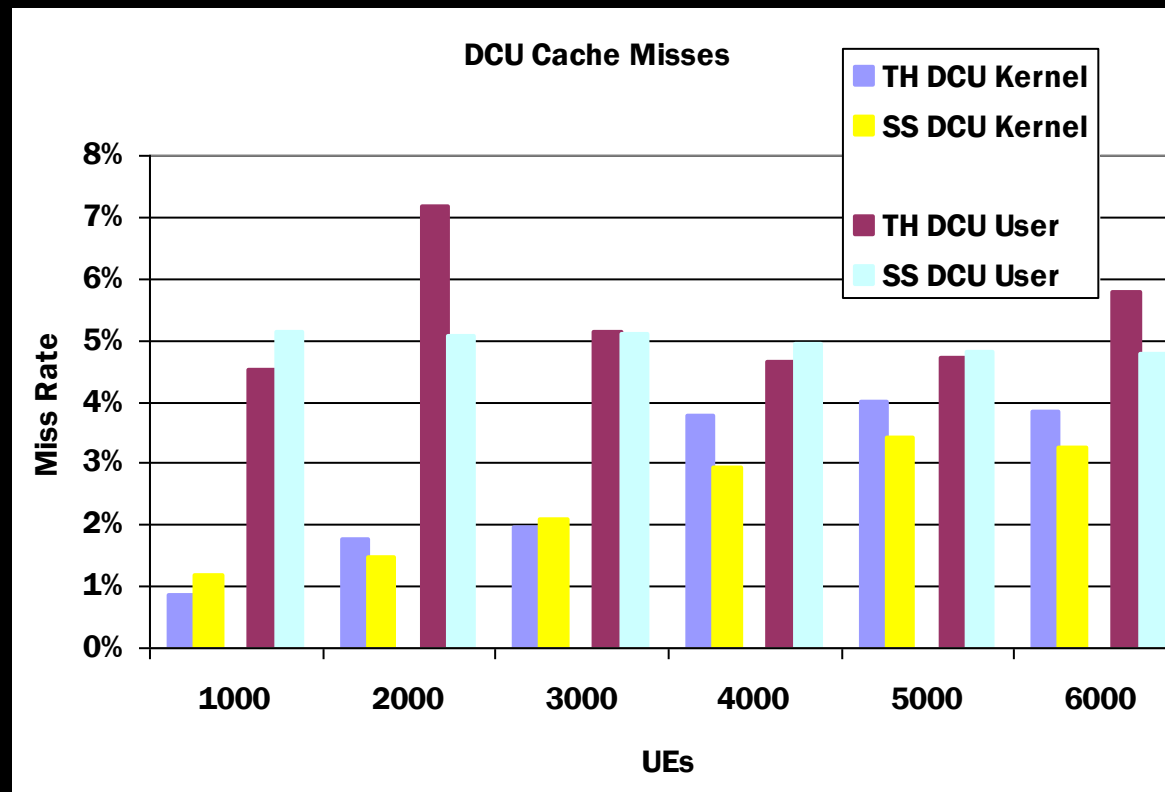
Cycles with Stalls



L2 Cache Misses



L1 Data Cache Misses



Future Work

- Error/fault handling
- System coordination language
 - Concise view of FSMs & communication
 - Verification of properties
 - Deadlock freedom, progress, don't lose work,...
- Extend to clusters
 - Same semantics shared/non-shared memory
 - Reconfigure without rewriting

Summary

- Good performance requires good software – not just hardware – architecture
- Threads are a weak foundation for locality

Cohort Scheduling

- Enhance locality by grouping similar operations
- Staged computation supports operation
 - Identifies cohorts
 - Supports cohort scheduling
 - Reduces synchronization