Operating System Mechanism for Continuation-based Fine-grained Threads on Dedicated & Commodity Processors

Shigeru Kusakabe[†], Satoshi Yamada[†], Mitsuhiro Aono[†], Masaaki Izumi[†], Satoshi Amamiya[†], Yoshinari Nomura[‡], Hideo Taniguchi[‡], and Makoto Amamiya[†]

† Kyushu University ‡ Okayama University

Outline

- Introduction
- Thread Model
- OS Issues on FUCE
- OS Issues on Commodity Processor
- Concluding remarks

Introduction

Multithreading: available on commodity platforms, derived from sequential model

Our approach

Model: dataflow

– natural to asynchronous/concurrent execution

Focus: architectures, languages, <u>operating systems</u>

Platform: dedicated & commodity processor

Introduction - on dedicated platform

Fuce: dedicated to fine-grained multithreading

Benchmarks were user applications,
How about operating systems?

System calls with I/O request
- Multithreading with continuation,
- Handling external events without "interrupt"
- Delivered without controller such as APIC

Introduction - on commodity platform

Dataflow concept useful on commodity platforms?

⇨ flexible scheduling to reduce overhead

Wrapped System Call

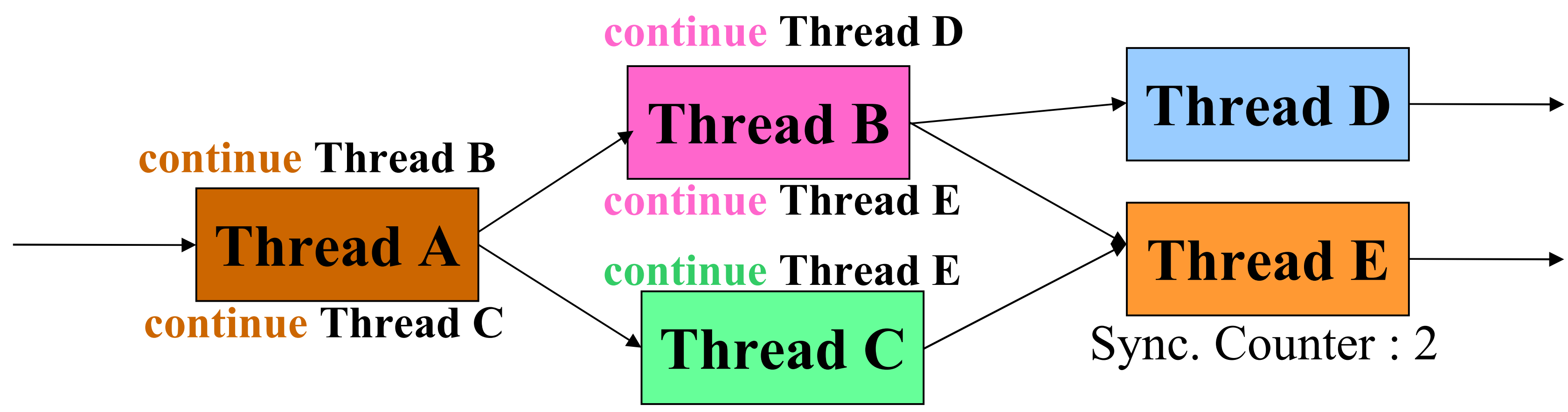
- buffer split-phase system call requests
- reduce context (mode) changes
- enhance locality of reference

Outline

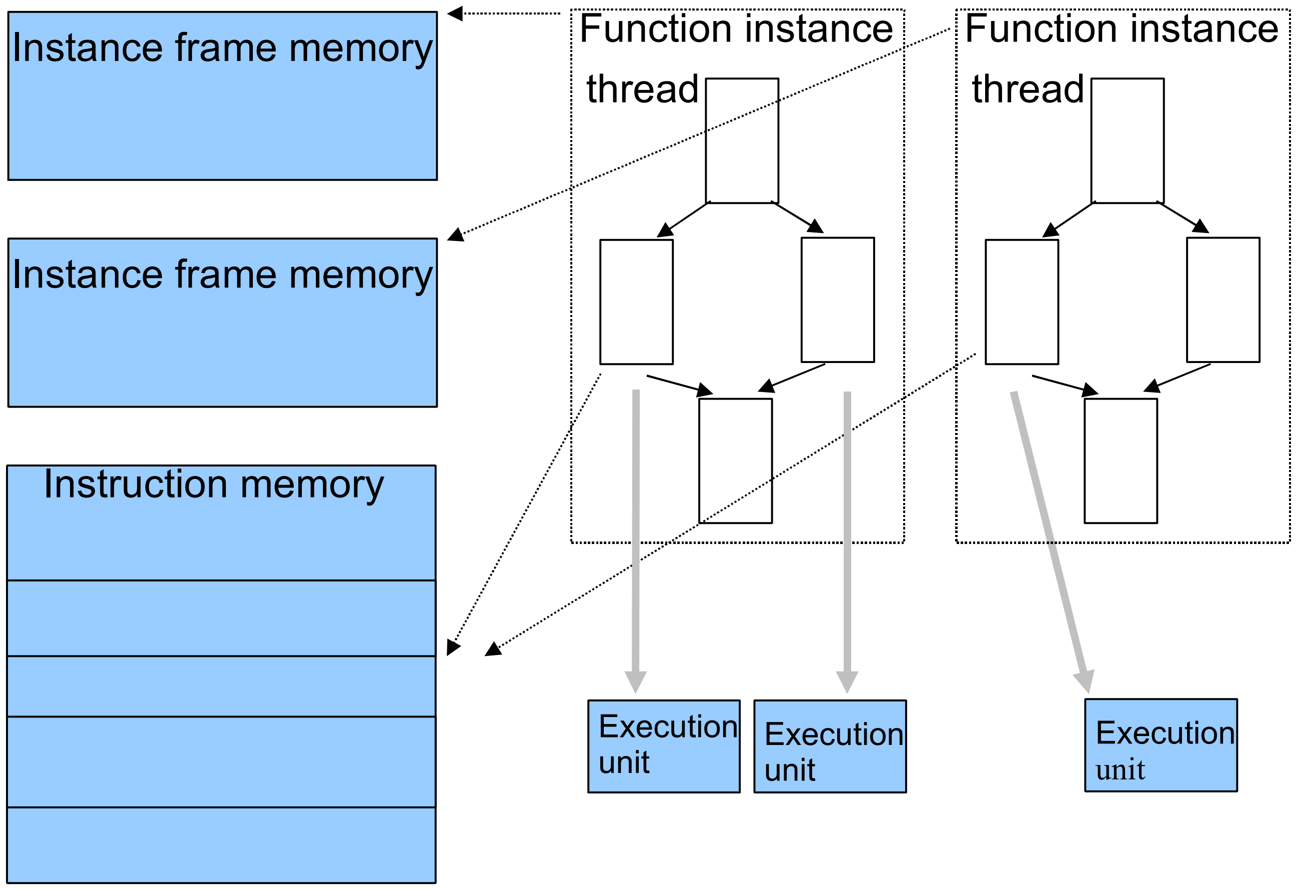
- Introduction
- Thread Model
- OS Issues on FUCE
- OS Issues on Commodity Processor
- Concluding Remarks

Zero-Wait Thread

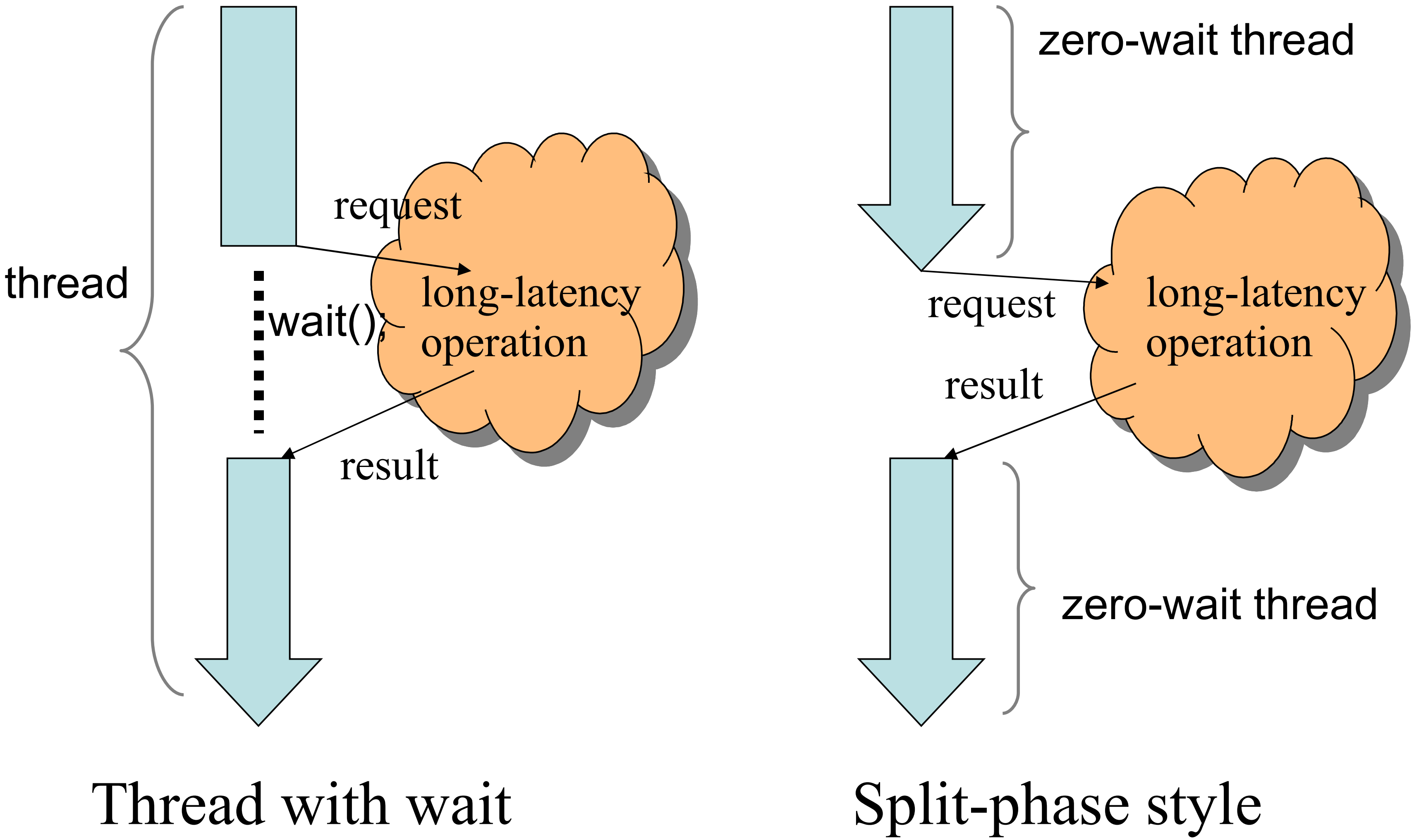
- Program graph: nodes / threads, edges / continuation relations.
- Thread: synch. counter & instruction sequence (incl. continuation)
- A continuation instruction specifies its succeeding thread code and context, and decrements the synchronization counter of the target.
- If the counter becomes to zero, the thread becomes ready to run, and runs to completion without suspension once started.



Thread and Instance



Split-phase



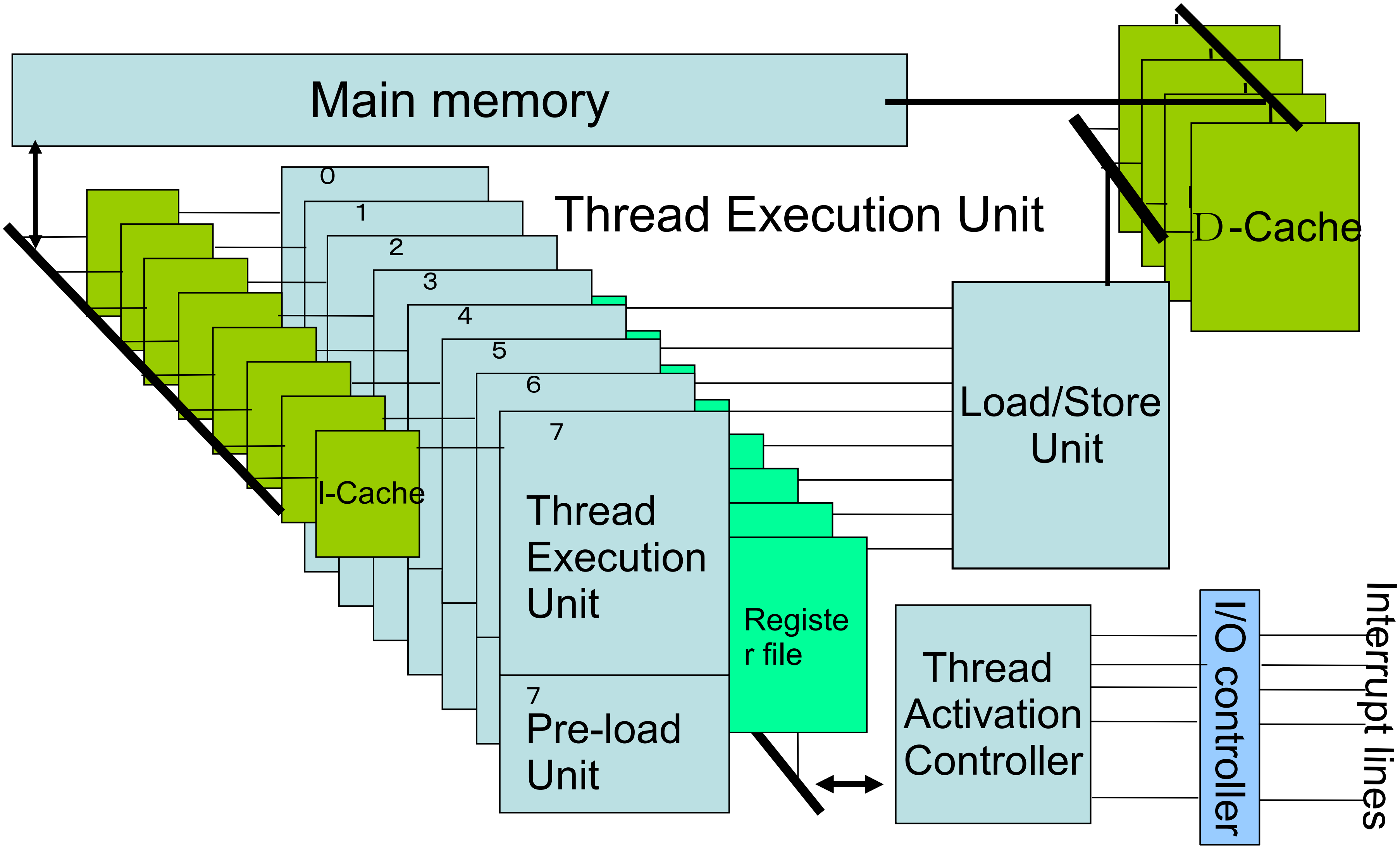
thread
request
wait();
long-latency operation
result
Thread with wait
zero-wait thread
request
long-latency operation
result
zero-wait thread
Split-phase style

Outline

- Introduction
- Thread Model
- OS Issues on FUCE
- OS Issues on Commodity Processor
- Concluding Remarks

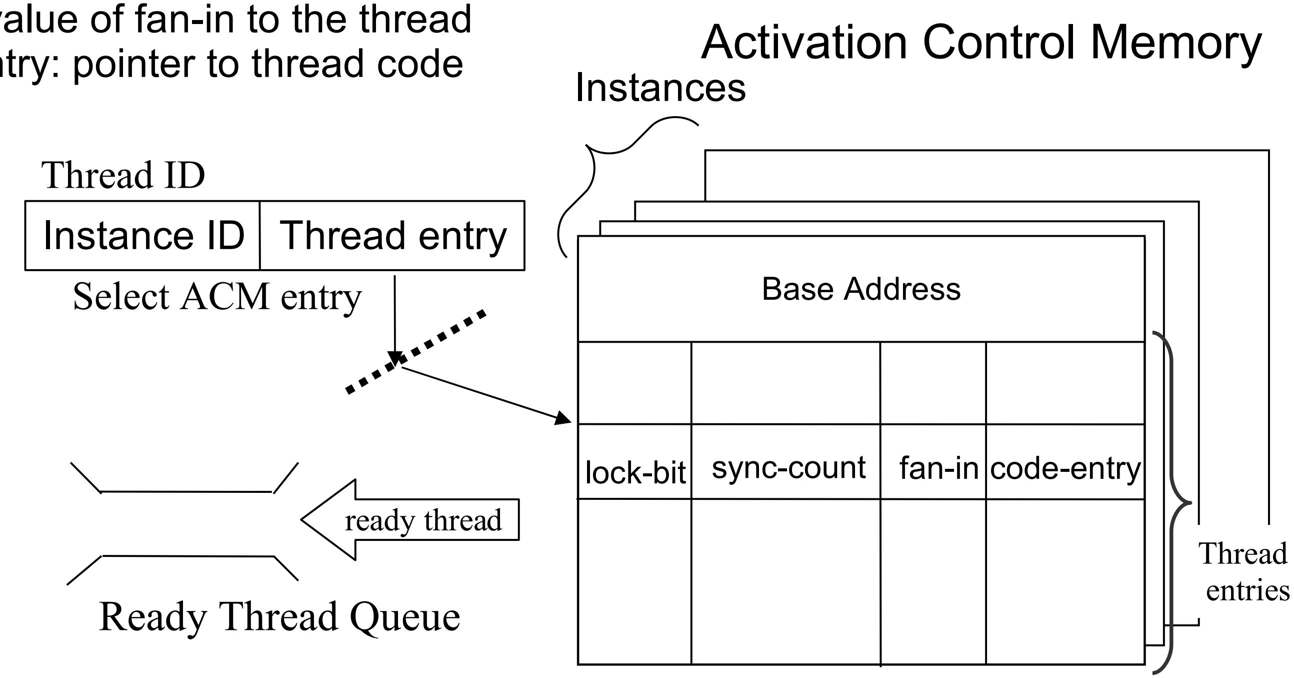
Fuce Processor

CMT (Chip multithreading) being developed at Kyushu University

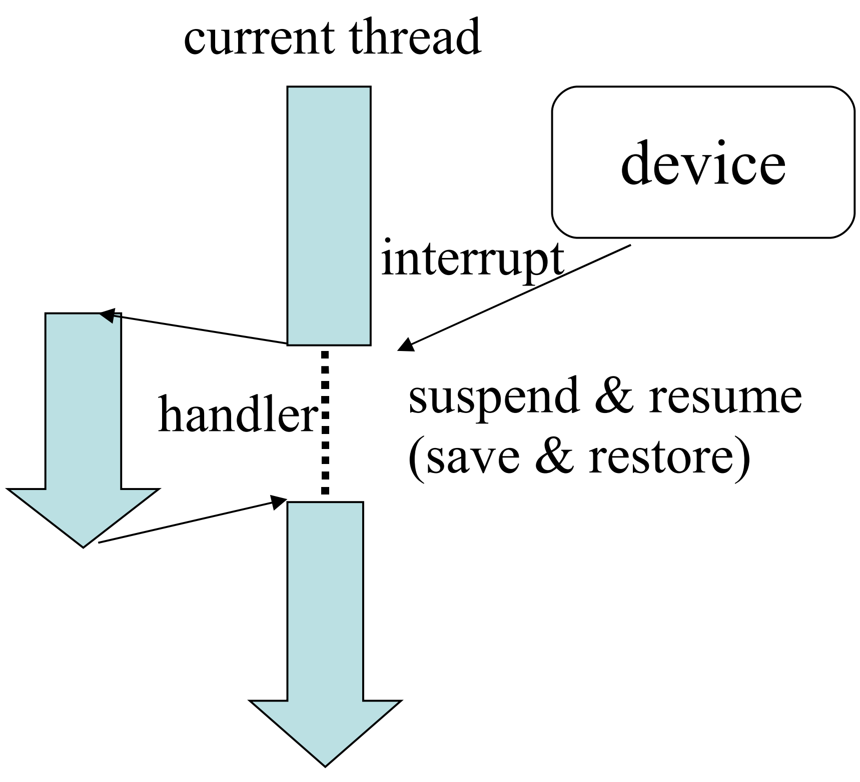


Thread Activation Controller

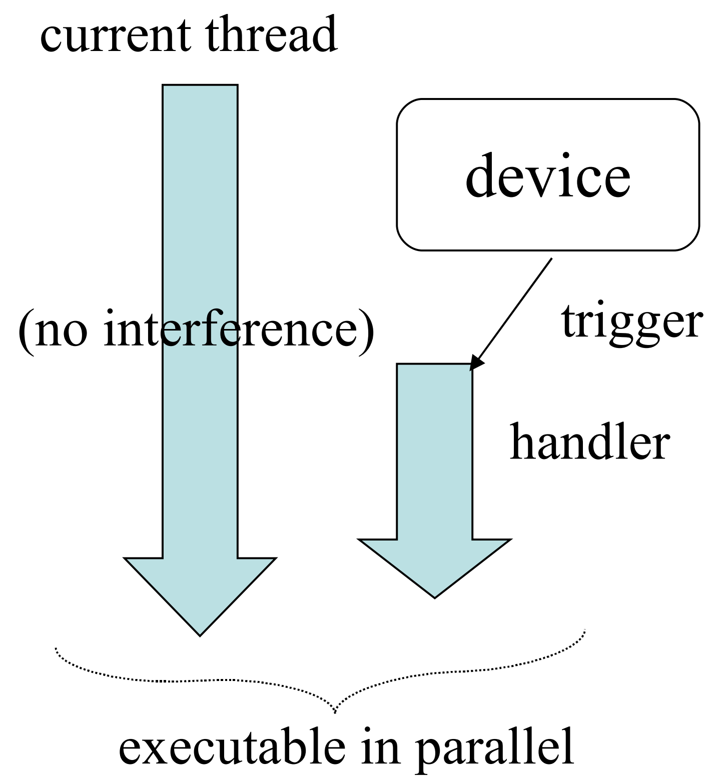
Base-address: pointer to data area
lock bit : semaphore
sync-count: # waiting continuations
fan-in: value of fan-in to the thread
code-entry: pointer to thread code



Handling External Event

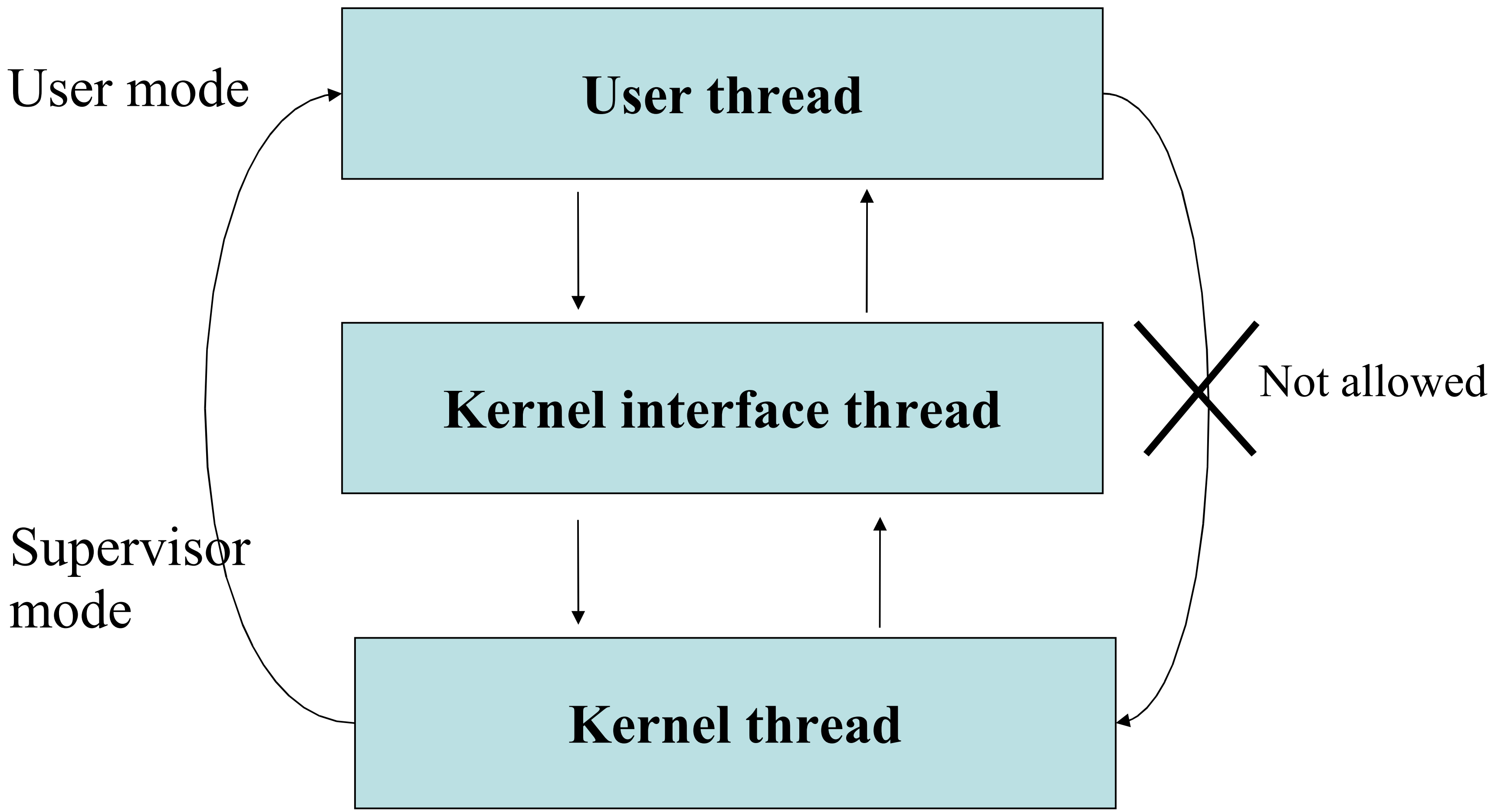


Interrupt-based
sequential approach

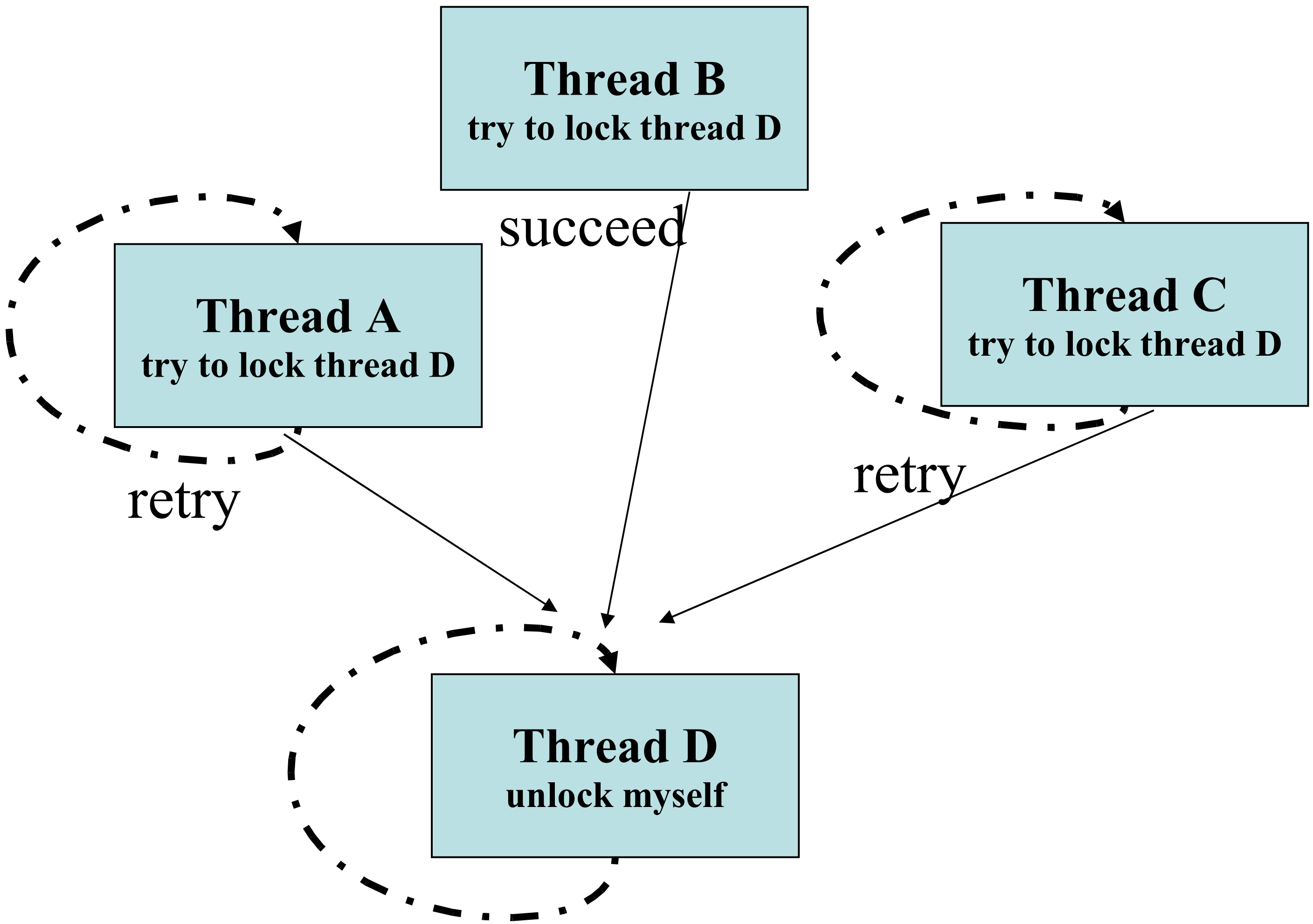


Continuation-based
zero-wait thread approach

Thread Mode


User mode
User thread
Kernel interface thread
Not allowed
Supervisor mode
Kernel thread

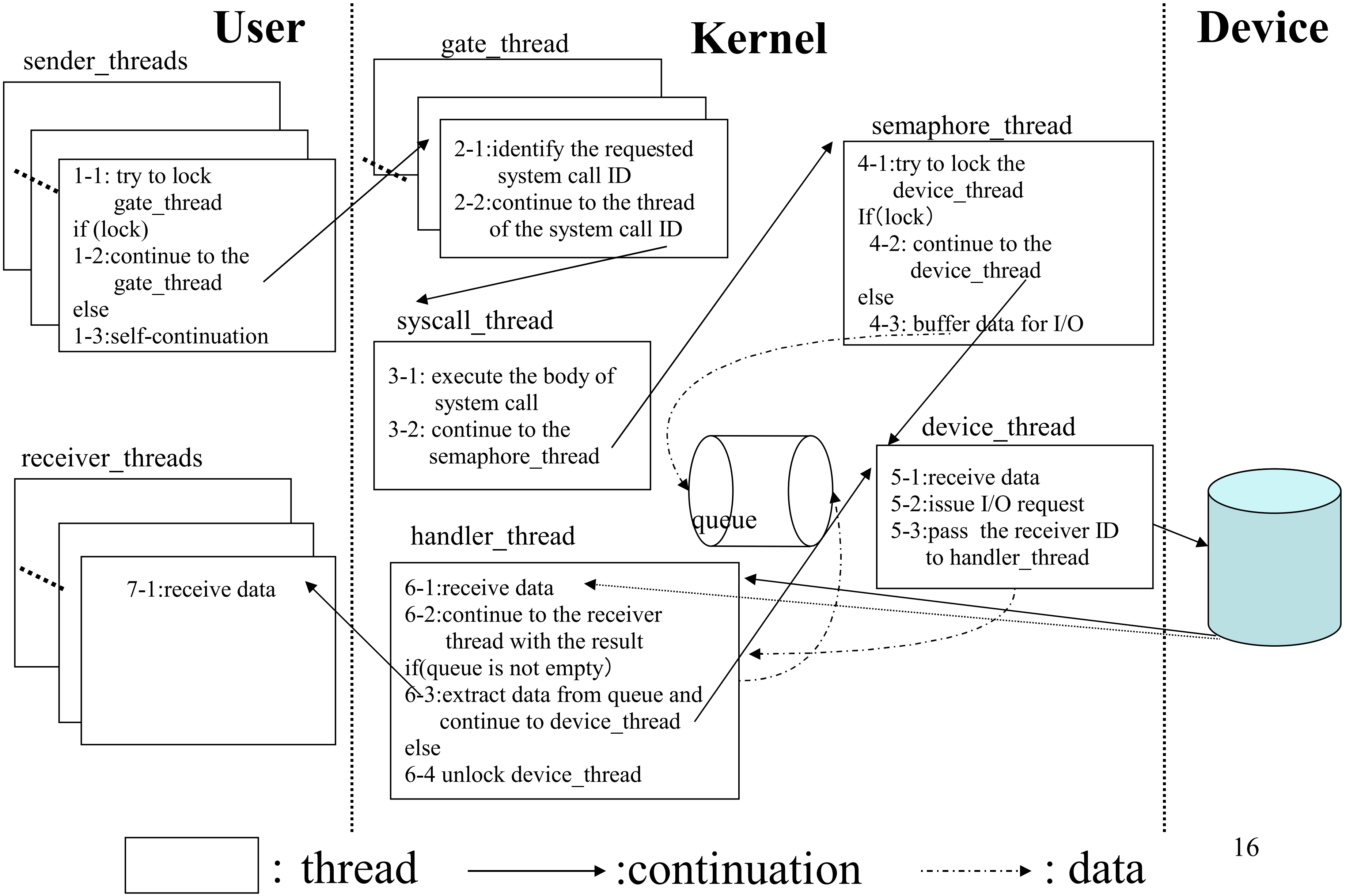
Critical Thread



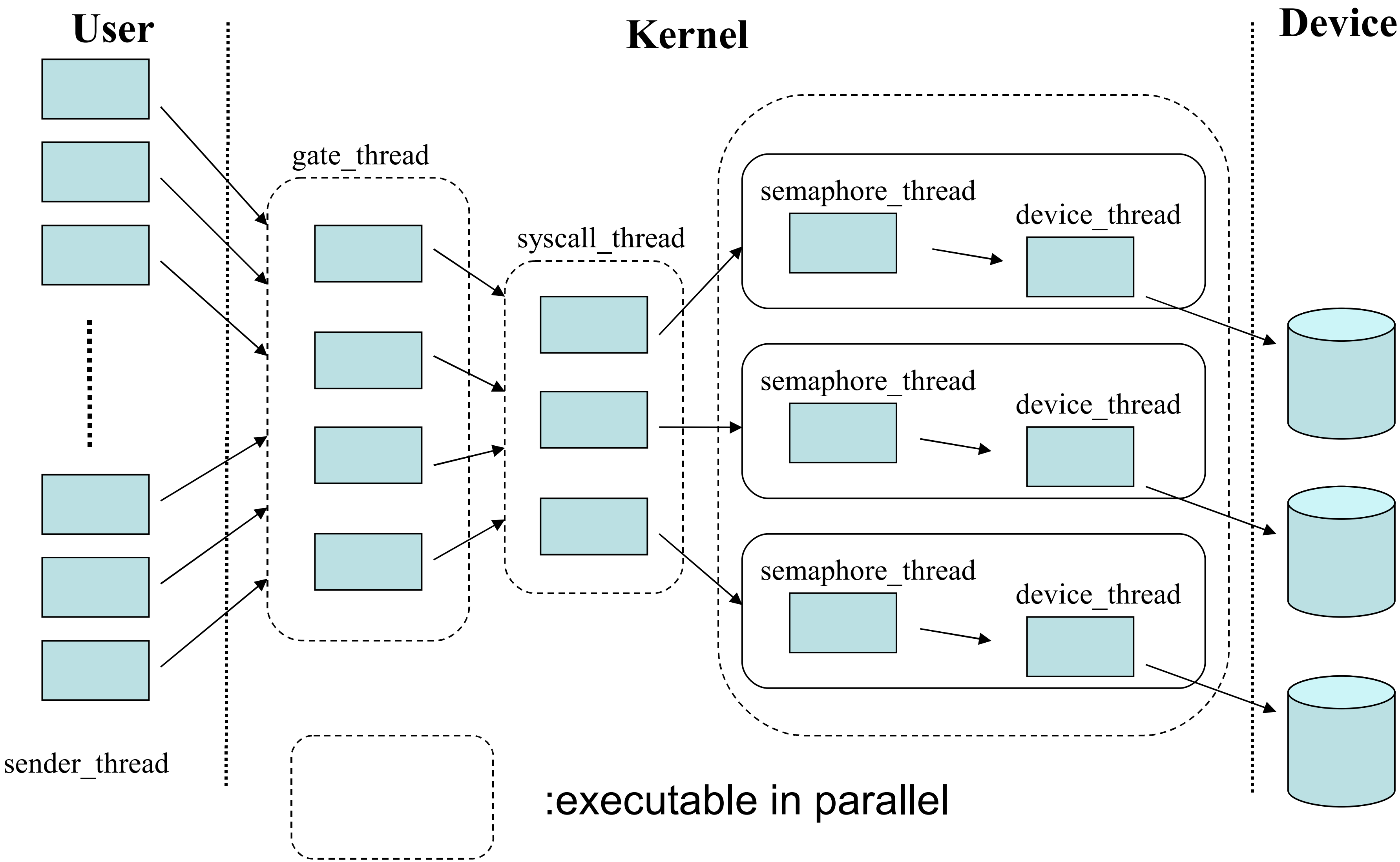
→ : continuation to another thread

- - · - - · → : continuation to itself

Handling System calls with I/O Request



Thread Activation



:executable in parallel

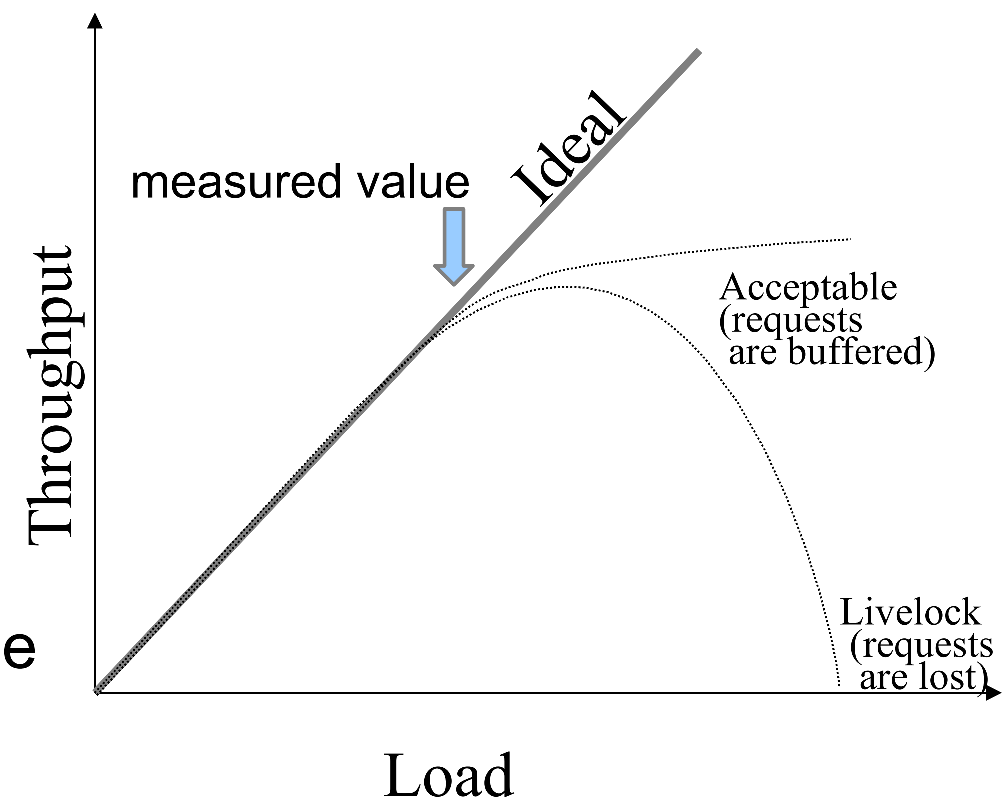
Measurement

Fuce in VHDL on ModelSim

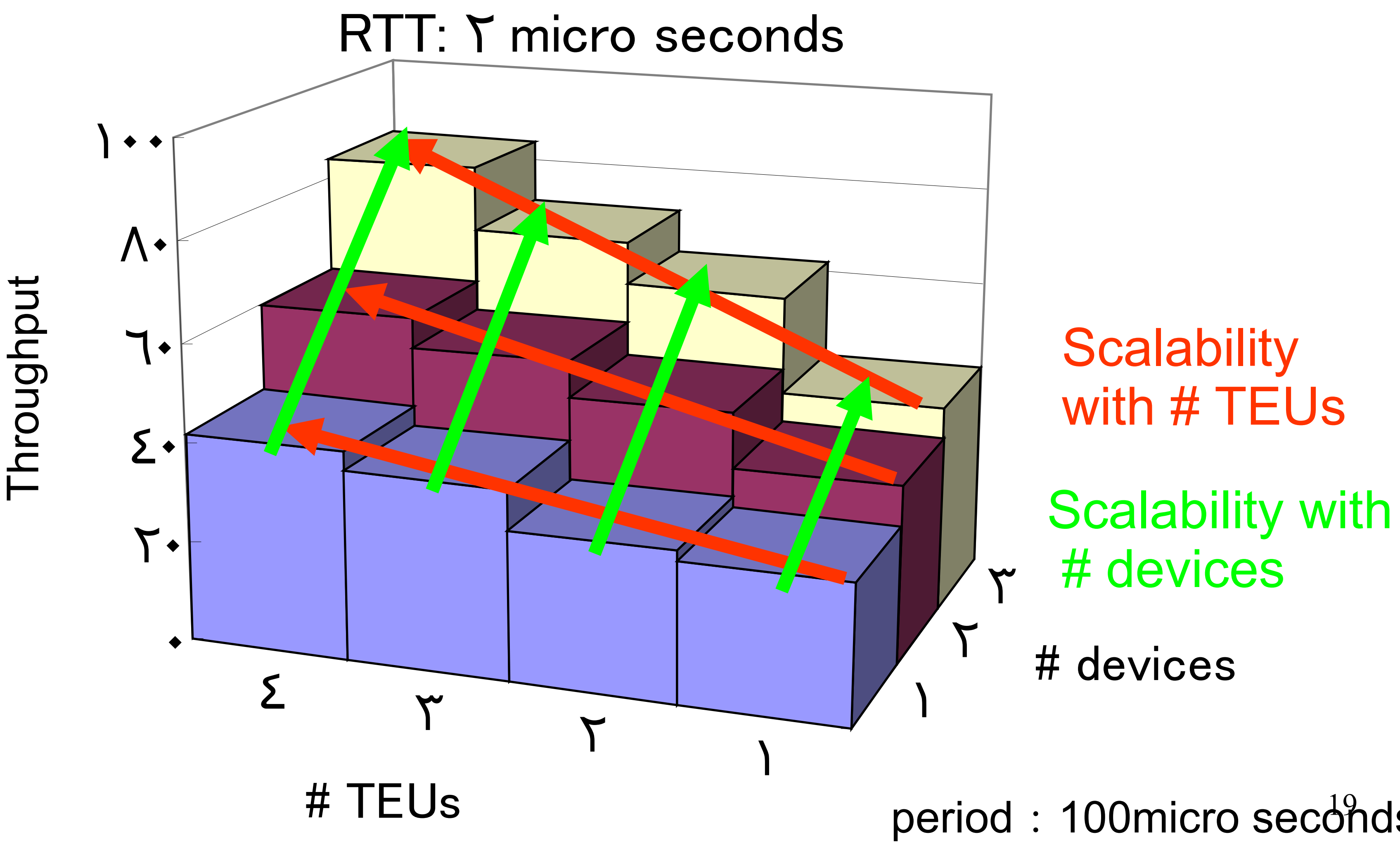
Measured the number of system calls with I/O request ideally completed within a fixed period.

The number of
TEUs:1..4, devices: 1..3

Expectation: scalability -- activation of hander thread due to continuation mechanism



Evaluation Result



period : 100micro seconds

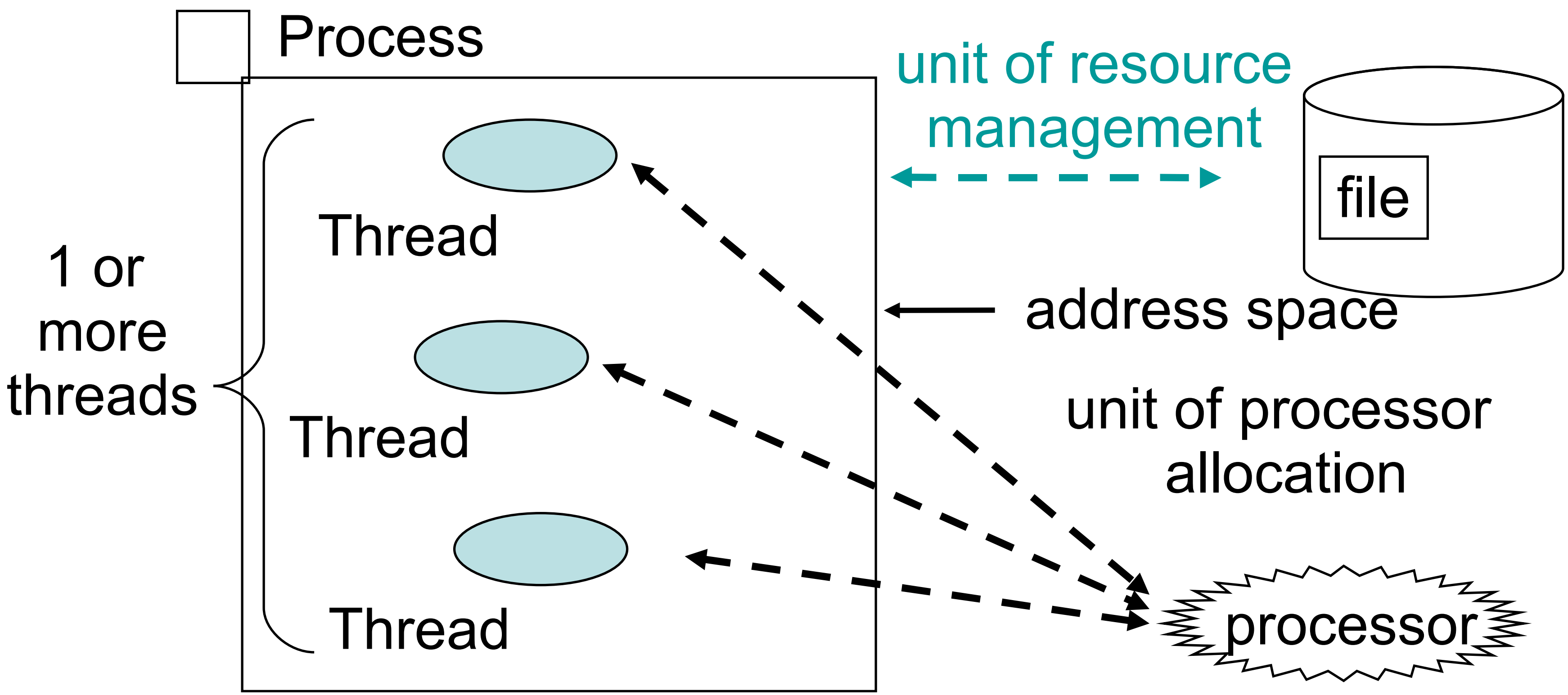
Outline

- Introduction
- Thread Model
- OS Issues on FUCE
- <u>OS Issues on Commodity Processor</u>
- Concluding Remarks

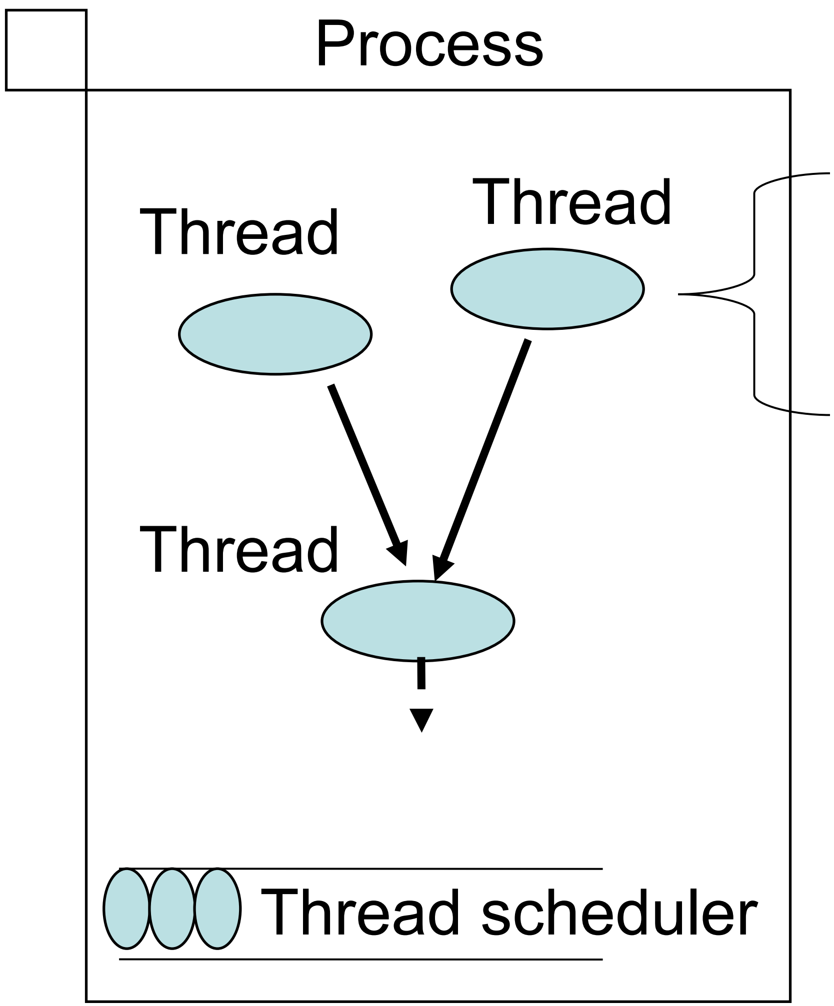
CEFOS: Process / Threads

Unix-like process, and dataflow-like thread

- dependence graph (partially ordered threads)
- process as thread context (color / tag)



CEFOS: Process / Threads



Controls between threads:
in a dataflow-like fashion

- synchronization counter
- serial code
- continuation

Dataflow concept useful?

Preliminary experiment

LMbench result for Linux 2.6.14 - Latency benchmark (in clocks)

processor	null call	2p/0K	2p/16K	L1	L2	M. Memory
pentium III	378	1576	5044	3	8	164
pentium4	1090	3298	5798	2	18	261
PowerPC G4	200	788	2167	4	10	127
PowerPC G5	306	13698	13734	3	11	113
Intel Core Duo	464	1327	2820	3	14	152

System Call Overhead

Process Switch

Memory Latency

CEFOS - wrapped system-call

Partially ordered fine-grain threads

split-phase style system calls

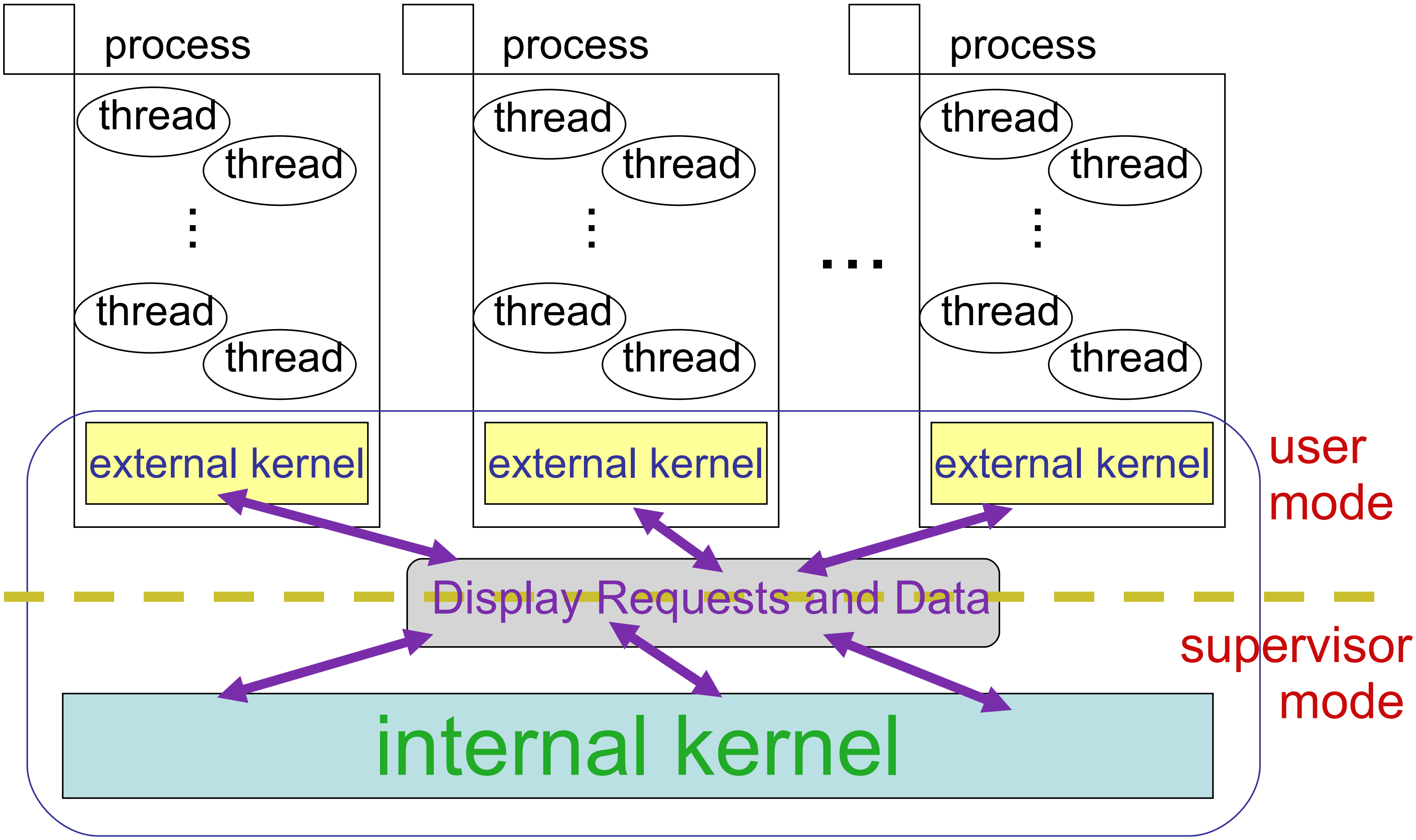
… various choices in scheduling threads/processes

Wrapped System-Call (WSC)

- aggregates multiple system-call requests
- handles them as a single system-call

to reduce overhead of system calls and enhance locality

CEFOS and WSC mechanism



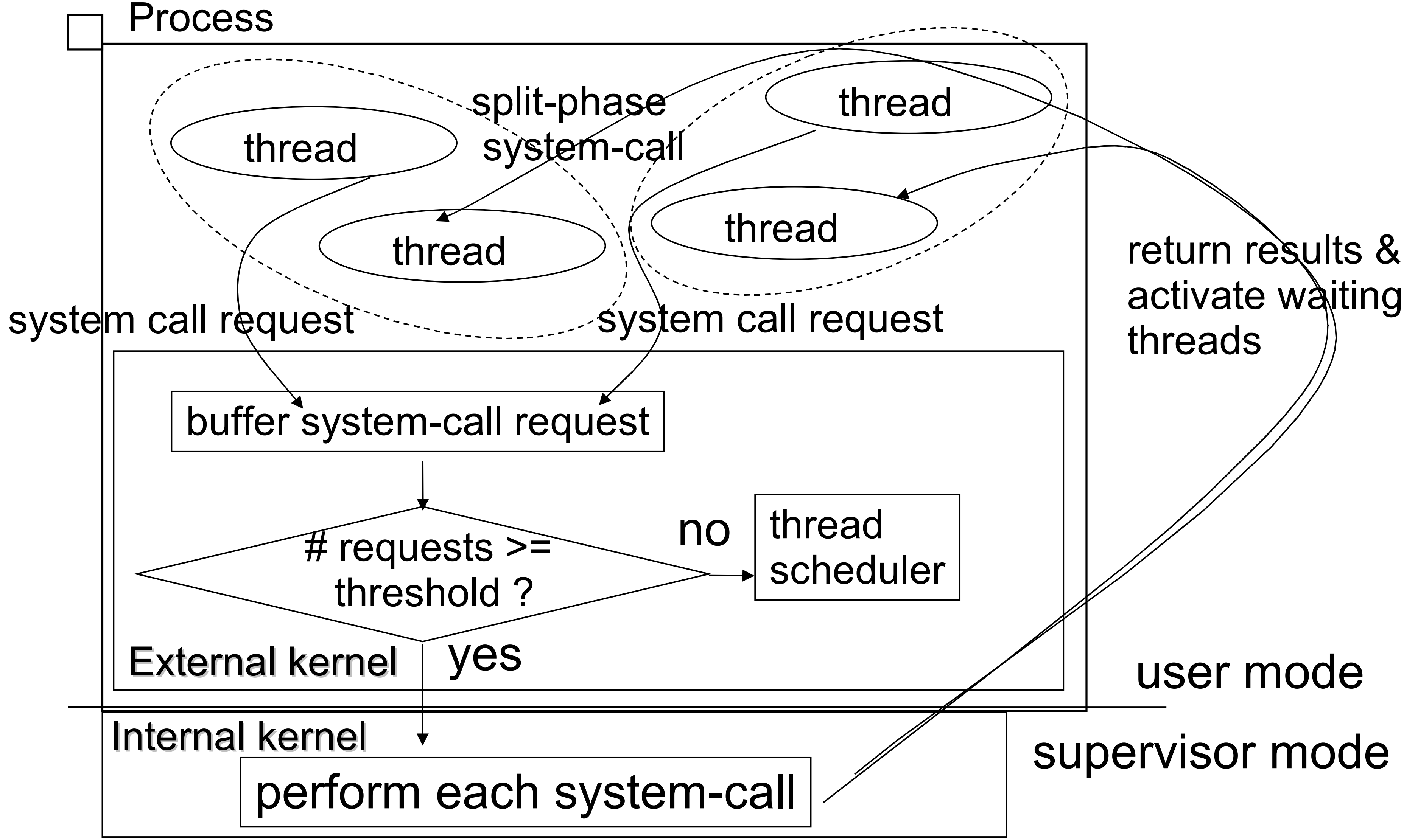
DRD

DRD (Display Request & Data)

Intermediate communications between Internal / External Kernel.

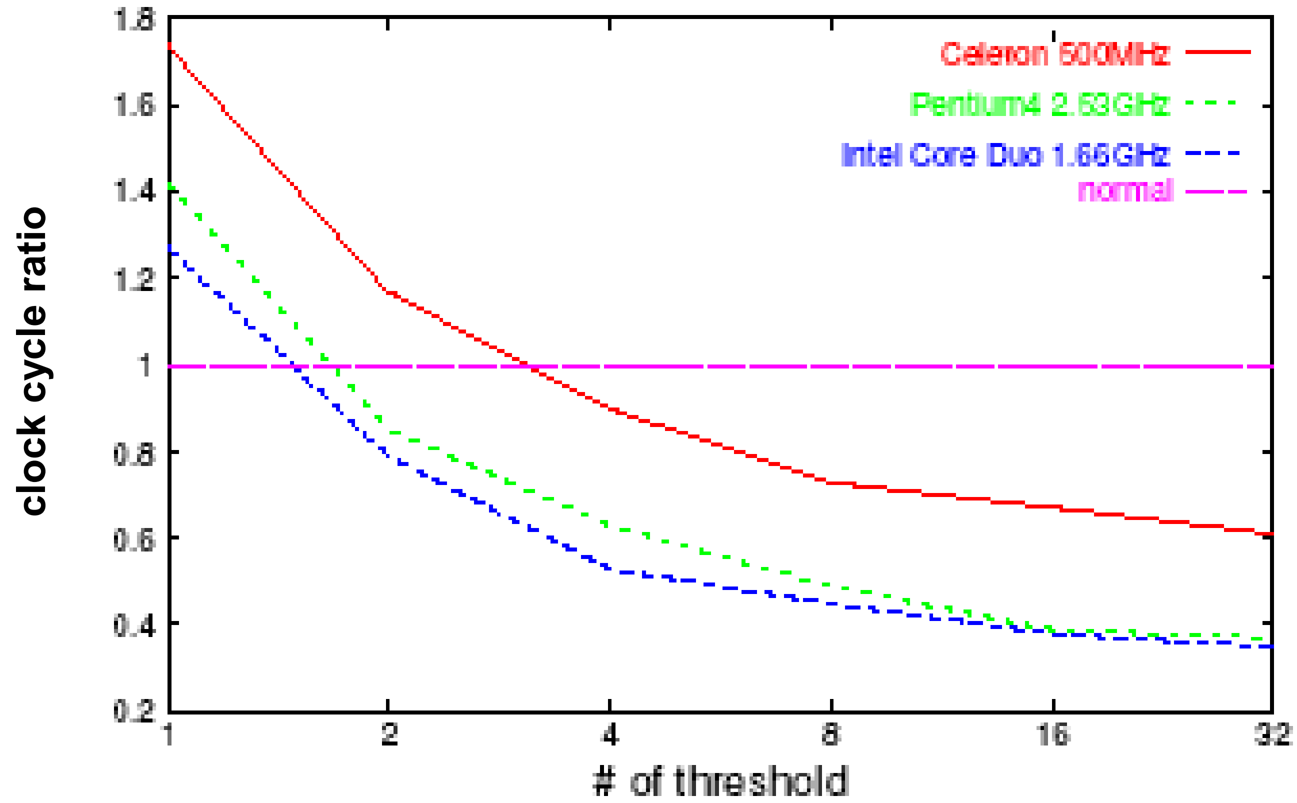
- Each process & kernel share common memory area (CA)
- Each process & kernel display requests and necessary information on CA
- At appropriate occasions, each process & kernel check requests and information displayed on CA, and change the control of execution if necessary.

Control flows in WSC



Impact on System Call Overhead

Implemented by modifying Linux.
Issuing system calls with thin body: getpid()



Locality of reference

- chatroom benchmark
  - simulate chat rooms (server and clients)
  - four threads per client (2 message handler (send /receive) in client & server)
  - parameters: number of clients = 20

Detailed memory events with performance monitoring counter - hardmeter (limited to focused part only)

	clocks	L2$ miss (%)	D-TLB miss (%)
normal	60217	1.01	2.78
WSC	48436	0.47	2.55
ratio: WSC/normal	0.80	0.47	0.92

Concluding remarks

Multithreading based on dataflow model

On Fuce

- event handling without “interrupt”

On commodity platforms

- Wrapped System-Call: aggregates split-phase style system call requests

Evaluation

  - scalability of throughput in handling I/O request
  - system call overhead and locality of reference