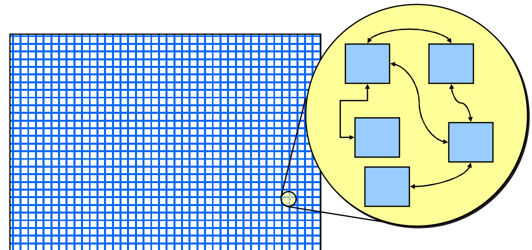
Microkernel Construction

IPC Implementation

THE UNIVERSITY OF NEW SOUTH WALES

IPC Importance



THE UNIVERSITY OF NEW SOUTH WALES

General IPC Algorithm

- Validate parameters
- Locate target thread
  - if unavailable, deal with it
- Transfer message
  - untyped - short IPC
  - typed message - long IPC
- Schedule target thread
  - switch address space as necessary
- Wait for IPC

THE UNIVERSITY OF NEW SOUTH WALES

IPC - Implementation

Short IPC

THE UNIVERSITY OF NEW SOUTH WALES

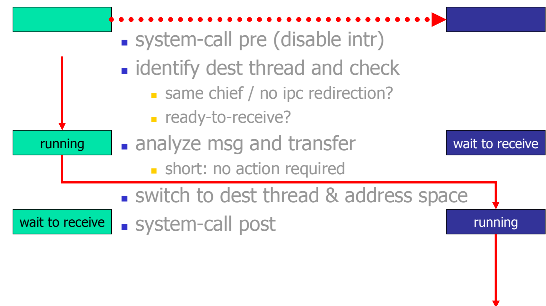
Short IPC (uniprocessor)

- system-call preamble (disable intr)
- identify dest thread and check
  - same chief / no ipc redirection?
  - ready-to-receive?
- analyze msg and transfer
  - short: no action required
- switch to dest thread & address space
- system-call postamble

The critical path

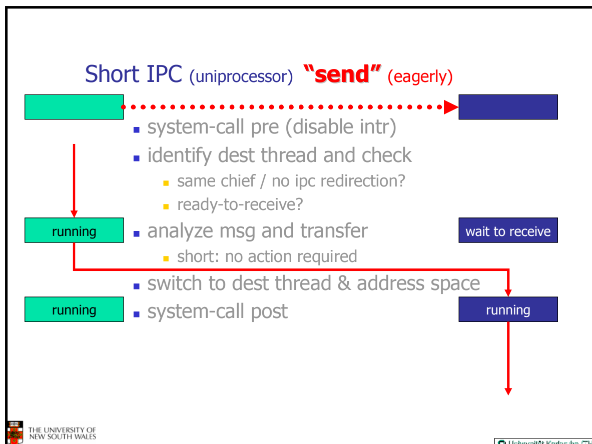
THE UNIVERSITY OF NEW SOUTH WALES

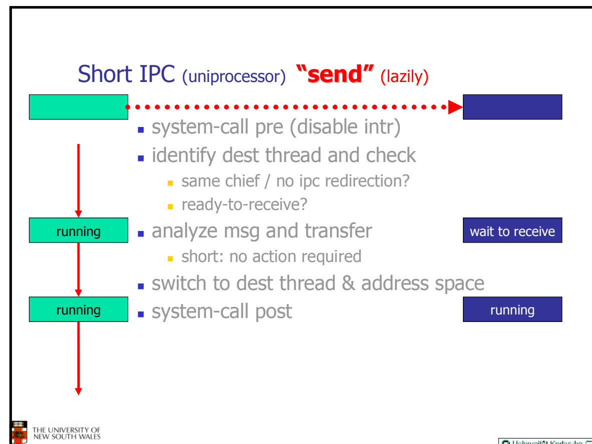
Short IPC (uniprocessor) **"call"**

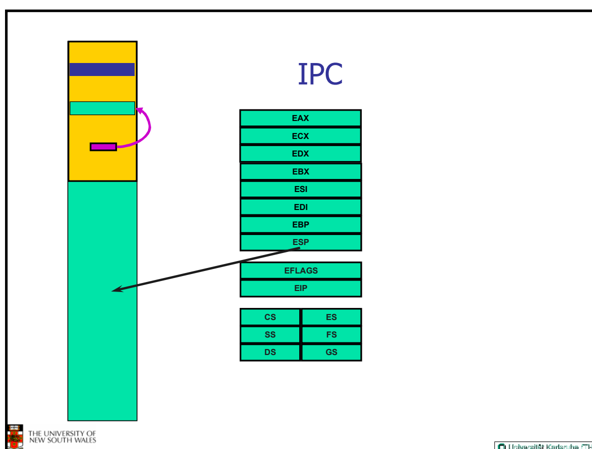


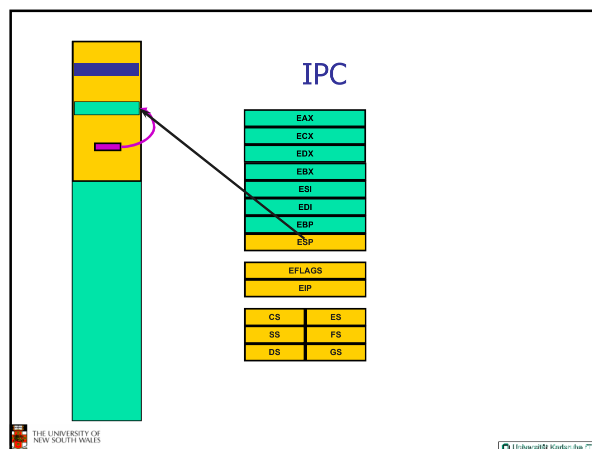
- system-call pre (disable intr)
- identify dest thread and check
  - same chief / no ipc redirection?
  - ready-to-receive?
- analyze msg and transfer
  - short: no action required
- switch to dest thread & address space
- system-call post

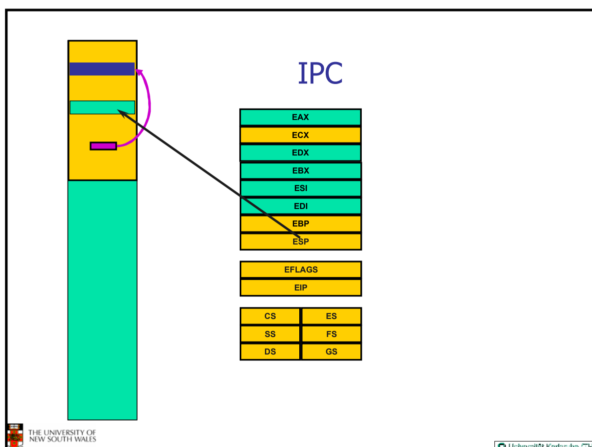
THE UNIVERSITY OF NEW SOUTH WALES

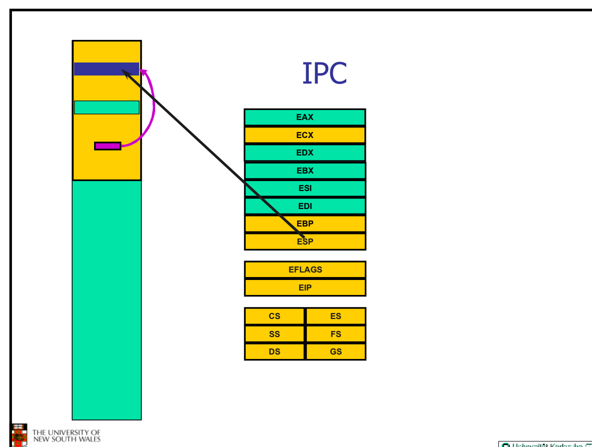

Short IPC (uniprocessor) "send" (eagerly)
system-call pre (disable intr)
identify dest thread and check
same chief / no ipc redirection?
ready-to-receive?
running
analyze msg and transfer
wait to receive
short: no action required
switch to dest thread & address space
running
system-call post
running
THE UNIVERSITY OF NEW SOUTH WALES

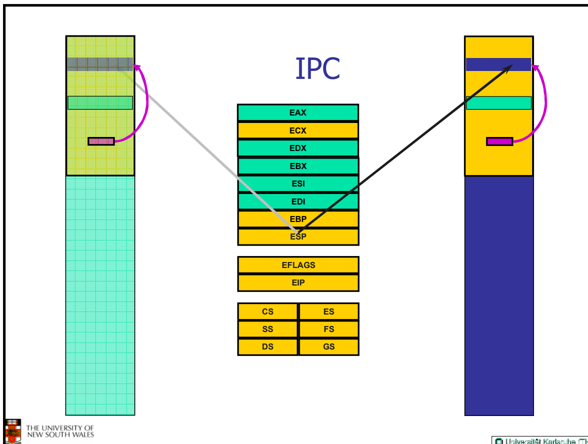

Short IPC (uniprocessor) "send" (lazily)
system-call pre (disable intr)
identify dest thread and check
same chief / no ipc redirection?
ready-to-receive?
running
analyze msg and transfer
wait to receive
short: no action required
switch to dest thread & address space
running
system-call post
running
THE UNIVERSITY OF NEW SOUTH WALES

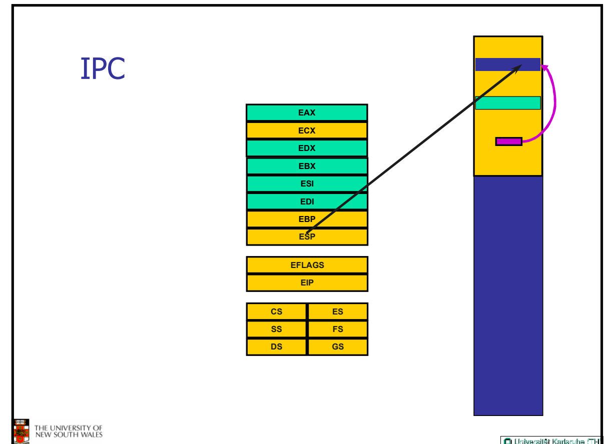

IPC
EAX
ECX
EDX
EBX
ESI
EDI
EBP
ESP
EFLAGS
EIP
CS
ES
SS
FS
DS
GS
THE UNIVERSITY OF NEW SOUTH WALES

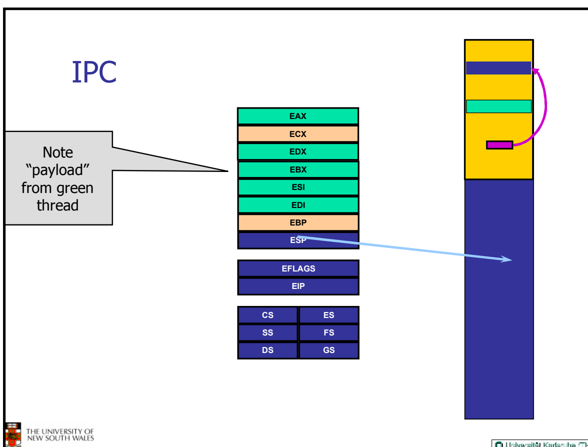

IPC
EAX
ECX
EDX
EBX
ESI
EDI
EBP
ESP
EFLAGS
EIP
CS
ES
SS
FS
DS
GS
THE UNIVERSITY OF NEW SOUTH WALES


IPC
EAX
ECX
EDX
EBX
ESI
EDI
EBP
ESP
EFLAGS
EIP
CS
ES
SS
FS
DS
GS
THE UNIVERSITY OF NEW SOUTH WALES


IPC
EAX
ECX
EDX
EBX
ESI
EDI
EBP
ESP
EFLAGS
EIP
CS
ES
SS
FS
DS
GS
THE UNIVERSITY OF NEW SOUTH WALES







Implementation Goal

- Most frequent kernel op: short IPC
  - thousands of invocations per second
- Performance is critical:
  - structure IPC for speed
  - **structure entire kernel to support fast IPC**
- What affects performance?
  - cache line misses
  - TLB misses
  - memory references
  - pipe stalls and flushes
  - instruction scheduling

THE UNIVERSITY OF NEW SOUTH WALES

Fast Path

- Optimize for common cases
  - write in assembler
  - non-critical paths written in C++
    - but still fast as possible
- Avoid high-level language overhead:
  - function call state preservation
  - poor code "optimizations"
- We want every cycle possible!

THE UNIVERSITY OF NEW SOUTH WALES

IPC Attributes for Fast Path

- untyped message
- single runnable thread after IPC
  - must be valid IPC call
  - switch threads, originator blocks
  - send phase:
    - the target is waiting
  - receive phase:
    - the sender is not ready to couple, causing us to block
- no receive timeout

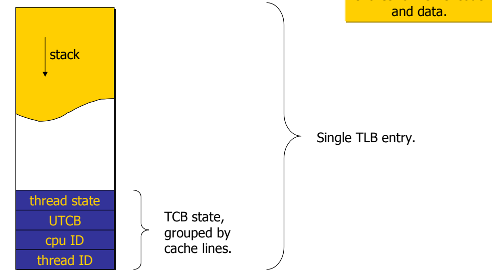
THE UNIVERSITY OF NEW SOUTH WALES

Avoid Memory References!!!

- Memory references are slow
  - avoid in IPC:
    - ex: use lazy scheduling
  - avoid in common case:
    - ex: timeouts
- Microkernel should minimize indirect costs
  - cache pollution
  - TLB pollution
  - memory bus

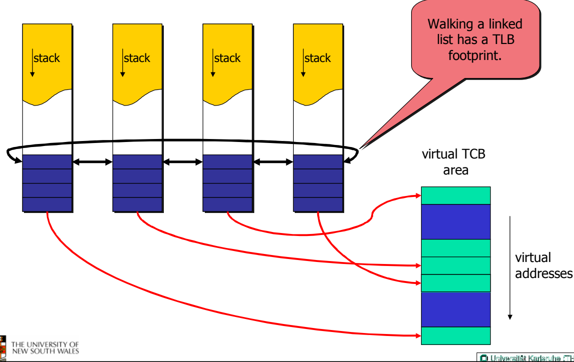
THE UNIVERSITY OF NEW SOUTH WALES

Optimized Memory



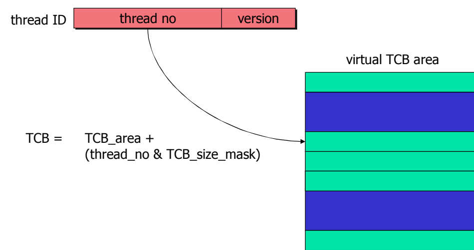
THE UNIVERSITY OF NEW SOUTH WALES

TLB Problem



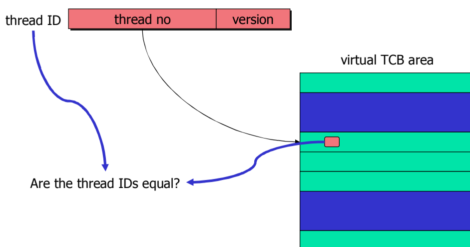
THE UNIVERSITY OF NEW SOUTH WALES

Avoid Table Lookups



THE UNIVERSITY OF NEW SOUTH WALES

Validate Thread ID



THE UNIVERSITY OF NEW SOUTH WALES

Branch Elimination

```
slow =    ~receiver->thread_state    +
          (timeouts & 0xffff)        +
          sender->resources          +
          receiver->resources;

if( slow )
          enter_slow_path()
```

Common case: -1

Common case: 0

- Reduces branch prediction foot print.
- Avoids mispredicts & stalls & flushes.
  - Increases latency for slow path

THE UNIVERSITY OF NEW SOUTH WALES

TCB Resources

- One bit per resource
- Fast path checks entire word
  - if not 0, jump to resource handlers

Resources bitfield

1 1

Copy area

Debug registers

Message Transfer

inter-address space IPC

cycles per IPC
instructions per IPC

message registers copied

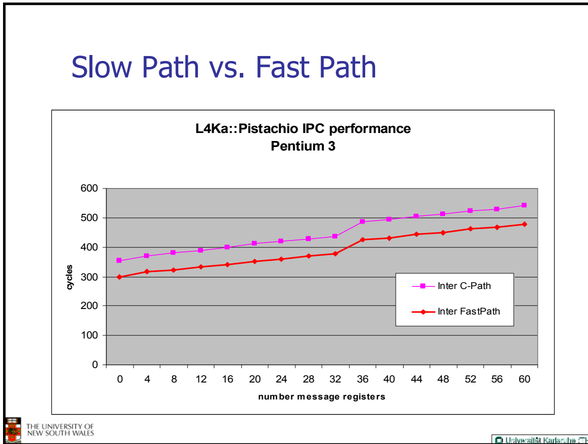
up to 10 physical registers

virtual register copy loop

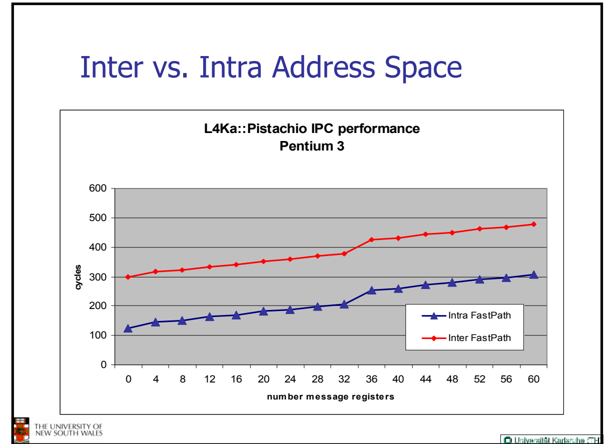
IBM PowerPC 750, 500 MHz, 32 registers

Many cycles wasted on pipe flushes for privileged instructions.

Slow Path vs. Fast Path



Inter vs. Intra Address Space



IPC - Implementation

Long IPC

Long IPC (uniprocessor)

- system-call preamble (disable intr)
- identify dest thread and check
  - same chief
  - ready-to-receive?
- analyze msg and transfer
  - long/map:
    - ***– transfer message –***
- switch to dest thread & address space
- system-call postamble

Preemptions possible!
(end of timeslice, device interrupt...)

Pagefaults possible!
(in source and dest address space)

Long IPC (uniprocessor)

- system-call pre (disable intr)
- identify dest thread and check
  - same chief
  - ready-to-receive?
- analyze msg and transfer
  - long/map:
    - lock both partners
    - *– transfer message –*
    - unlock both partners
- switch to dest thread & address space
- system-call post

Preemptions possible! (end of timeslice, device interrupt...)

Pagefaults possible! (in source and dest address space)

THE UNIVERSITY OF NEW SOUTH WALES

Long IPC (uniprocessor)

- system-call pre (disable intr)
- identify dest thread and check
  - same chief
  - ready-to-receive?
- analyze msg and transfer
  - long/map:
    - lock both partners
    - enable intr
    - *– transfer message –*
    - disable intr
    - unlock both partners
- switch to dest thread & address space
- system-call post

Preemptions possible! (end of timeslice, device interrupt...)

Pagefaults possible! (in source and dest address space)

THE UNIVERSITY OF NEW SOUTH WALES

Long IPC (uniprocessor)

- system-call pre (disable intr)
- identify dest thread and check
  - same chief
  - ready-to-receive?
- analyze msg and transfer
  - long/map:
    - lock both partners
    - enable intr
    - *– transfer message –*
    - disable intr
    - unlock both partners
- switch to dest thread & address space
- system-call post

running | wait
locked running | *locked* wait
wait to receive | running

THE UNIVERSITY OF NEW SOUTH WALES

IPC - mem copy

- Why is it needed? Why not share?
  - Security
    - Need own copy
  - Granularity
    - Object small than a page or not aligned

THE UNIVERSITY OF NEW SOUTH WALES

copy in - copy out

- copy into kernel buffer

THE UNIVERSITY OF NEW SOUTH WALES

copy in - copy out

- copy into kernel buffer
- switch spaces

THE UNIVERSITY OF NEW SOUTH WALES

copy in - copy out

- copy into kernel buffer
- switch spaces
- copy out of kernel buffer

- costs for n words
  - $2 \times 2n$ r/w operations
  - $3 \times n/8$ cache lines
    - $1 \times n/8$ overhead cache misses (small n)
    - $4 \times n/8$ cache misses (large n)

THE UNIVERSITY OF NEW SOUTH WALES

temporary mapping

THE UNIVERSITY OF NEW SOUTH WALES

temporary mapping

- select dest area (4+4 M)

THE UNIVERSITY OF NEW SOUTH WALES

temporary mapping

- select dest area (4+4 M)
- map into source AS (kernel)

THE UNIVERSITY OF NEW SOUTH WALES

temporary mapping

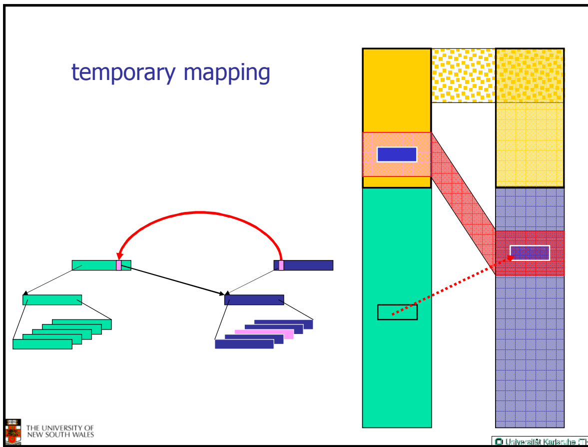
- select dest area (4+4 M)
- map into source AS (kernel)
- copy data

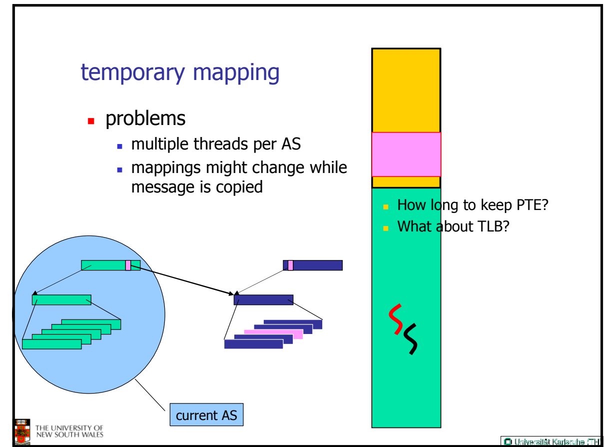
THE UNIVERSITY OF NEW SOUTH WALES

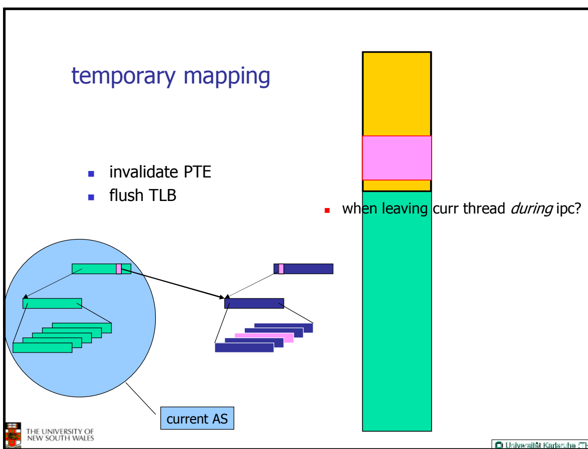
temporary mapping

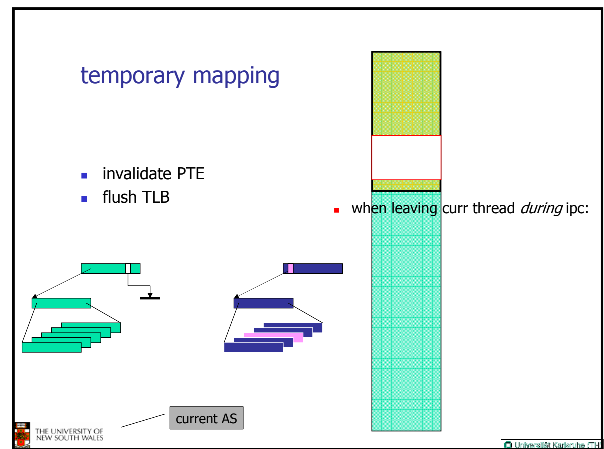
- select dest area (4+4 M)
- map into source AS (kernel)
- copy data
- switch to dest space

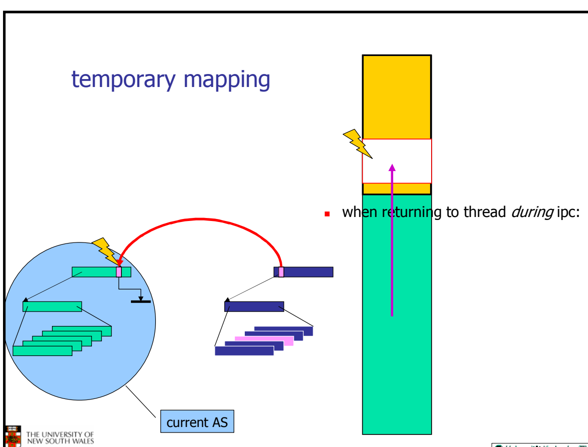
THE UNIVERSITY OF NEW SOUTH WALES

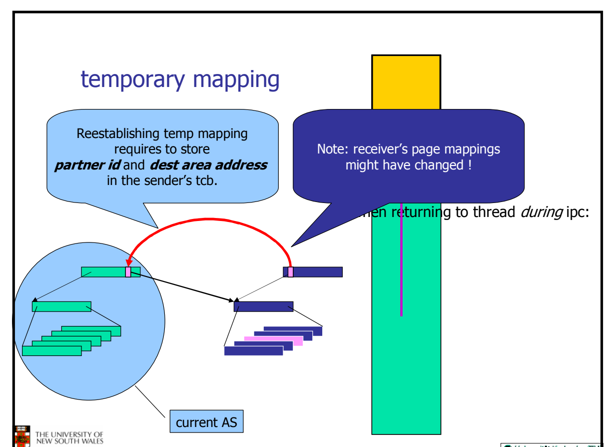

temporary mapping


temporary mapping
problems
multiple threads per AS
mappings might change while message is copied
How long to keep PTE?
What about TLB?
current AS


temporary mapping
invalidate PTE
flush TLB
when leaving curr thread during ipc?
current AS


temporary mapping
invalidate PTE
flush TLB
when leaving curr thread during ipc:
current AS


temporary mapping
when returning to thread during ipc:
current AS


temporary mapping
Reestablishing temp mapping requires to store partner id and dest area address in the sender's tcb.
Note: receiver's page mappings might have changed !
current AS

temporary mapping

Start temp mapping:
mytcb.partner := partner ;
mytcb.waddr := dest 8M area base ;
myPDE.TMarea := destPDE.destarea .

Leave thread:
if mytcb.waddr ≠ *nil* **then**
myPDE.TMarea := *nil* ;
if dest AS = my AS **then**
flush TLB
fi **fi**

why?

Close temp mapping:
mytcb.waddr := *nil* .

myPDE.TMarea := *nil* ??

optimization only:
avoids second TLB flush if subsequent thread switch would flush TLB anyhow

THE UNIVERSITY OF NEW SOUTH WALES

temporary mapping

- **Alternative method:**

Requires separation of TLB flush and load PT root !

Does therefore not work reasonably on x86.

Load PT root implicitly includes TLB flush on x86.

Leave thread:
if mytcb.waddr ≠ *nil* **then**
myPDE.TMarea := *nil* ;
flush TLB ;
TLB flushed := true
fi .

Thread switch :
...
if TLB just flushed
then TLB flushed := false
else flush TLB
fi ;
PT root := ...

current AS

THE UNIVERSITY OF NEW SOUTH WALES

temporary mapping

- **Page Fault Resolution:**

current AS

THE UNIVERSITY OF NEW SOUTH WALES

temporary mapping

- **Page Fault Resolution:**

current AS

THE UNIVERSITY OF NEW SOUTH WALES

temporary mapping

- **Page Fault Resolution:**

current AS

THE UNIVERSITY OF NEW SOUTH WALES

temporary mapping

- **Page Fault Resolution:**

TM area PF:
if myPDE.TMarea = destPDE.destarea **then**
tunnel to (partner) ;
access dest area ;
tunnel to (my)
fi ;
myPDE.TMarea := destPDE.destarea .

current AS

THE UNIVERSITY OF NEW SOUTH WALES

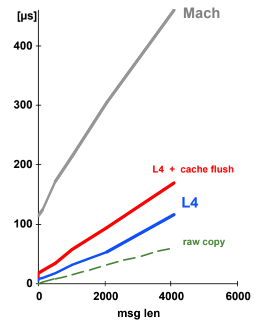
Cost estimates

	Copy in - copy out	Temporary mapping
R/W operations	2 × 2n	2n
Cache lines	3 × n/8	2 × n/8
Small n overhead cache misses	n/8	0
Large n cache misses	5 × n/8	3 × n/8
Overhead TLB misses	0	n / words per page
Startup instructions	0	50

THE UNIVERSITY OF NEW SOUTH WALES

486 IPC costs

- Mach: copy in/out
- L4: temp mapping



THE UNIVERSITY OF NEW SOUTH WALES

Dispatching

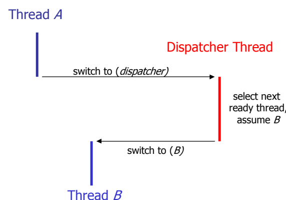
THE UNIVERSITY OF NEW SOUTH WALES

Dispatching topics:

- thread switch
  - (to a specific thread)
  - to next thread to be scheduled
    - (to nil)
    - implicitly, when ipc blocks
- priorities
- preemption
  - time slices
  - wakeups, interruptions
- timeouts and wake-ups
  - time

THE UNIVERSITY OF NEW SOUTH WALES

Switch to ():

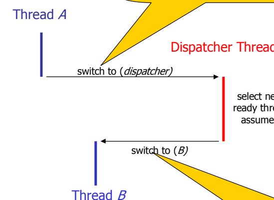


- Smaller stack per thread
- Dispatcher is preemptable
  - Improved interrupt latency if dispatching is time consuming

THE UNIVERSITY OF NEW SOUTH WALES

Switch to ():

tcb[A].sp := SP;
SP := disp thread bottom .

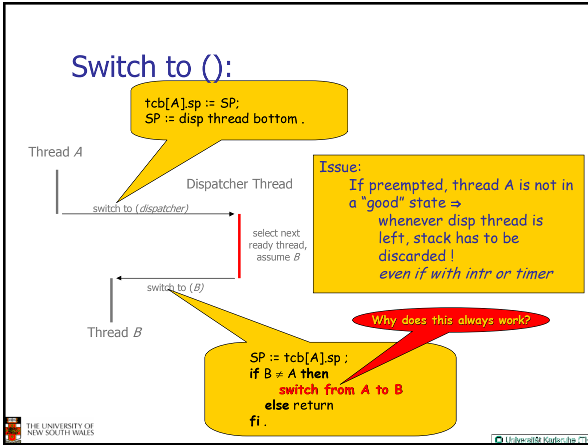


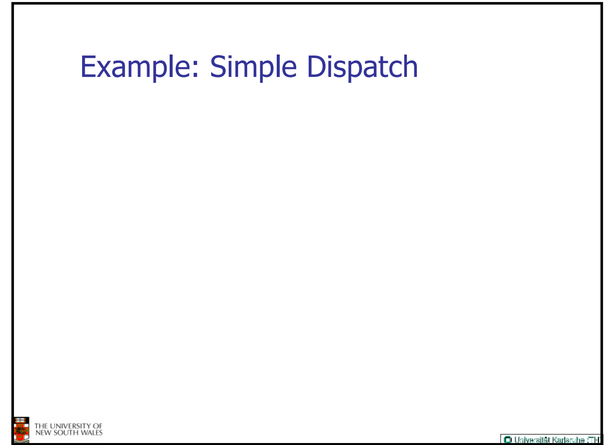
SP := tcb[A].sp ;
if B ≠ A **then**
switch from A to B
else return
fi .

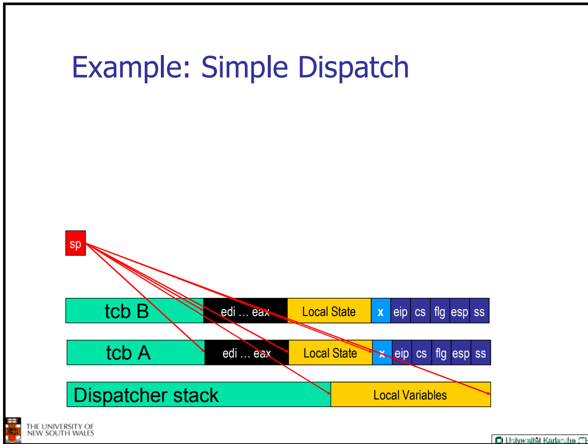
Why ??

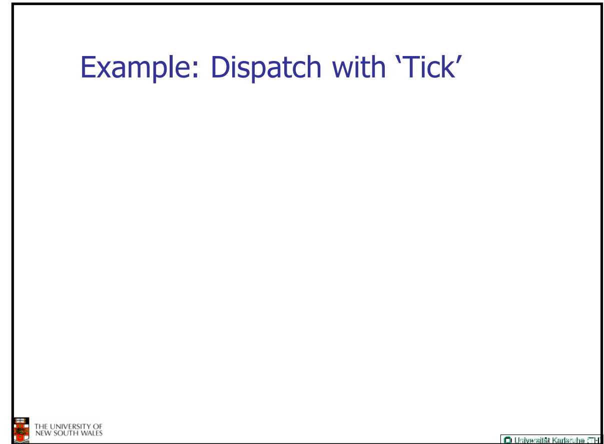
- Optimizations :
  - disp thread is special
    - no user mode,
      - no own AS required
        - Can avoid AS switch
      - no id required
        - Freedom from tcb layout conventions
    - almost stateless (see priorities)
      - No need to preserve internal state between invocations
      - External state must be consistent
  - *costs (A → B)*
  - ≈ *costs (A → disp → B)*
    - *costs (select next)*
  - *costs(A → disp → A) are low*

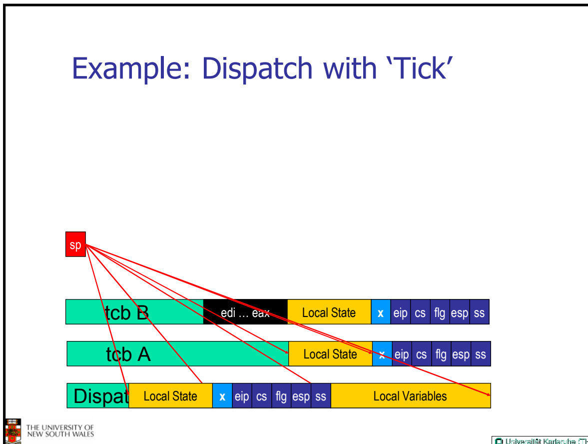
THE UNIVERSITY OF NEW SOUTH WALES

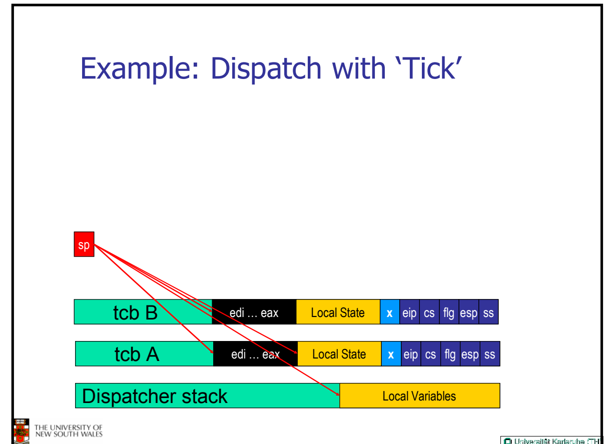

Switch to ():
tcb[A].sp := SP;
SP := disp thread bottom .
Thread A
Dispatcher Thread
switch to (dispatcher)
select next ready thread, assume B
switch to (B)
Thread B
Issue:
If preempted, thread A is not in a "good" state ⇒
whenever disp thread is left, stack has to be discarded !
even if with intr or timer
Why does this always work?
SP := tcb[A].sp ;
if B ≠ A then
switch from A to B
else return
fi .


Example: Simple Dispatch


Example: Simple Dispatch
sp
tcb B
edi … eax
Local State
x
eip
cs
flg
esp
ss
tcb A
Dispatcher stack
Local Variables

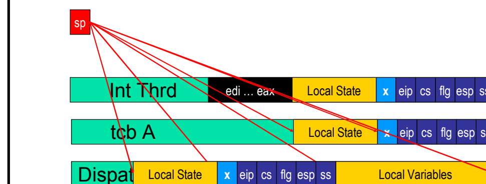

Example: Dispatch with 'Tick'


Example: Dispatch with 'Tick'
sp
tcb B
edi … eax
Local State
x
eip
cs
flg
esp
ss
tcb A
Dispat
Local Variables

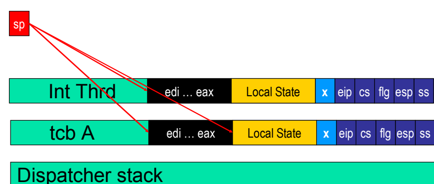

Example: Dispatch with 'Tick'
sp
tcb B
edi … eax
Local State
x
eip
cs
flg
esp
ss
tcb A
Dispatcher stack
Local Variables

Example: Dispatch with Interrupt

Example: Dispatch with Interrupt

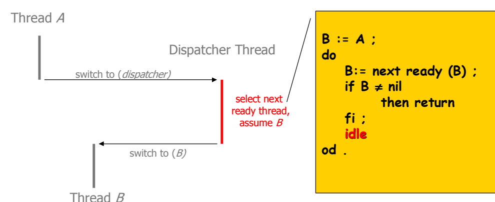


Example: Dispatch with Interrupt



Switch to ():

- dispatcher thread is also
  - ***idle thread***

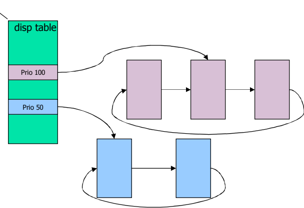



```
B := A ;
do
    B:= next ready (B) ;
    if B ≠ nil
        then return
    fi ;
    idle
od .
```

Priorities

- 0 (lowest) … 255
- hard priorities
- round robin per prio
- dynamically changeable
- ready tcb list per prio
- 'current tcb' per list

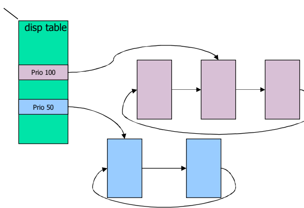
```
do
    p := 255;
    do
        if current[p] ≠ nil
            then B := current[p] ;
                return
        fi ;
        p -= 1
    until p < 0 od ;
    idle
od .
```



Priorities

- Optimization
  - keep highest active prio

```
do
    if current[highest active p] ≠ nil
        then B := current[highest active p];
            return
    elif highest active p > 0
        then highest active p -= 1
    else
        idle
    fi
od .
```



Priorities, Preemption

highest active p :=
max (new p, highest active p) .

```
do
   if current[highest active p] ≠ nil
      then B := current[highest active p];
           return
   elif highest active p > 0
      then highest active p -= 1
   else
      idle
   fi
od .
```

disp table
Prio 110
Prio 100
Prio 50
p= 110
intr/wakeup

THE UNIVERSITY OF NEW SOUTH WALES

Priorities, Preemption

- What happens when a prio falls empty ?

```
do
   if current[highest active p] ≠ nil
      then round robin if necessary;
           B := current[highest active p];
           return
   elif highest active p > 0
      then highest active p -= 1
   else
      idle
   fi
od .

round robin if necessary:
  if curr[hi act p] .rem ts = 0
    then curr[hi act p] := next ;
         current[hi act p].rem ts := new ts
  fi .
```

disp table
Prio 110
Prio 100
Prio 50
Remaining time slice > 0 ?

THE UNIVERSITY OF NEW SOUTH WALES

Priorities, Preemption

- What happens when a prio falls empty ?

```
do
   if current[highest active p] ≠ nil
      then round robin if necessary;
           B := current[highest active p];
           return
   elif highest active p > 0
      then highest active p -= 1
   else
      idle
   fi
od .

round robin if necessary:
  if curr[hi act p] .rem ts = 0
    then curr[hi act p] := next ;
         current[hi act p].rem ts := new ts
  fi .
```

disp table
Prio 110
Prio 100
Prio 50
Remaining time slice
> 0

THE UNIVERSITY OF NEW SOUTH WALES

Preemption

- Preemption, time slice exhausted

```
do
   if current[highest active p] ≠ nil
      then round robin if necessary;
           B := current[highest active p];
           return
   elif highest active p > 0
      then highest active p -= 1
   else
      idle
   fi
od .

round robin if necessary:
  if curr[hi act p] .rem ts = 0
    then curr[hi act p] := next ;
         current[hi act p].rem ts := new ts
  fi .
```

disp table
Prio 100
Prio 50
Remaining time slice
= 0

THE UNIVERSITY OF NEW SOUTH WALES

Preemption

- Preemption, time slice exhausted

```
do
   if current[highest active p] ≠ nil
      then round robin if necessary;
           B := current[highest active p];
           return
   elif highest active p > 0
      then highest active p -= 1
   else
      idle
   fi
od .

round robin if necessary:
  if current[hi act p] .rem ts = 0
    then current[hi act p].rem ts := new ts ;
         current[hi act p] := next
  fi .
```

disp table
Prio 100
Prio 50
Remaining time slice := new ts

THE UNIVERSITY OF NEW SOUTH WALES

Lazy Dispatching

Thread state toggles frequently (per ipc)

- *ready* ↔ *waiting*
  - delete/insert ready list is expensive
  - therefore: delete *lazily* from ready list

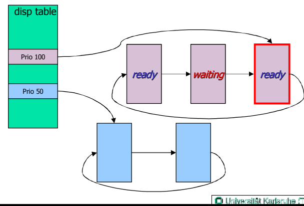
disp table
Prio 100
Prio 50
ready
ready
ready

THE UNIVERSITY OF NEW SOUTH WALES

Lazy Dispatching

Thread state toggles frequently (per ipc)

- *ready* ↔ *waiting*
  - delete/insert ready list is expensive
  - therefore: delete *lazily* from ready list

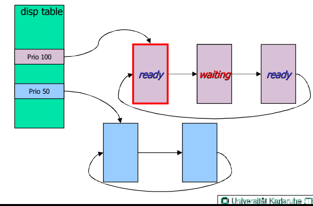


THE UNIVERSITY OF NEW SOUTH WALES

Lazy Dispatching

Thread state toggles frequently (per ipc)

- *ready* ↔ *waiting*
  - delete/insert ready list is expensive
  - therefore: delete *lazily* from ready list

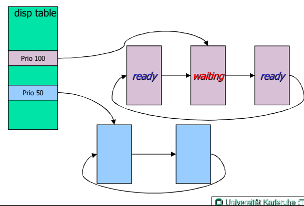


THE UNIVERSITY OF NEW SOUTH WALES

Lazy Dispatching

Thread state toggles frequently (per ipc)

- *ready* ↔ *waiting*
  - delete/insert ready list is expensive
  - therefore: delete *lazily* from ready list

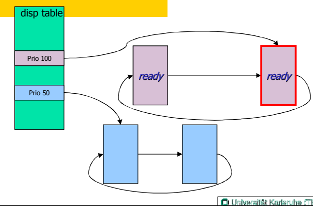


THE UNIVERSITY OF NEW SOUTH WALES

Lazy Dispatching

Thread state toggles frequently (per ipc)

- *ready* ↔ *waiting*
  - delete/insert ready list is expensive
  - therefore: delete *lazily* from ready list
  - Whenever reaching a non-ready thread,
    - delete it from list
    - proceed with next



THE UNIVERSITY OF NEW SOUTH WALES

Lazy Dispatching

```
do
    round robin if necessary;
    if current[highest active p] ≠ nil
        then B := current[highest active p]; return
    elif highest active p > 0
        then highest active p -= 1
    else
        idle
    fi
od .

round robin if necessary:
   while curr[hi act p] ≠ nil do
       if curr[hi act p].state ≠ ready
           then   delete from list (curr[hi act p])
       elif curr[hi act p].rem ts = 0
           then curr[hi act p].rem ts := new ts
       else  leave round robin if necessary
       fi ;
       curr[hi act p] := next ;
   od .
```