ProjectOZ

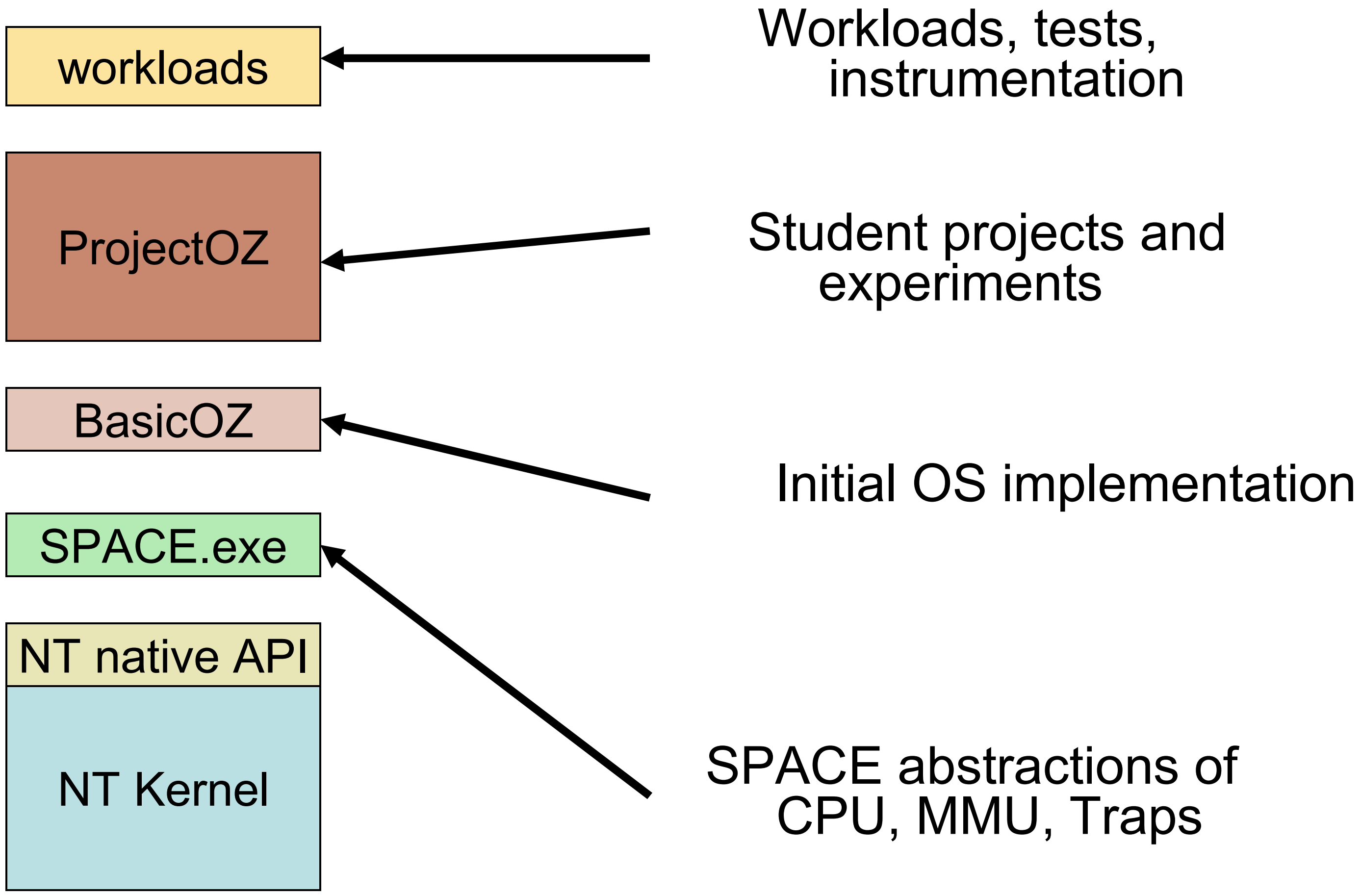
9 October 2006
Singapore

Dave Probert, Ph.D.
Architect, Windows Kernel Group
Windows Core Operating Systems Division
Microsoft Corporation

ProjectOZ Experimental Environment

- An OS project environment using the native NT API
  - Runs on Windows
  - Uses NT features designed for OS personality support
- Provides simple, user-mode abstractions of the
  - CPU, MMU, trap mechanism, and physical memory
- Experiments in OS principles, not computer organization
- Use real OS features rather than a ‘toy’ simulation
- Encourage ‘out-of-the-box’ thinking by students
- Based on SPACE project at UCSB (Probert & Bruno)

ProjectOZ



workloads

ProjectOZ

BasicOZ

SPACE.exe

NT native API

NT Kernel

Workloads, tests, instrumentation

Student projects and experiments

Initial OS implementation

SPACE abstractions of CPU, MMU, Traps

BasicOZ Functionality

Process/thread

- CreateProcess/Thread, Exit, Wait/Signal, Yield

Virtual Memory

- Allocate/Free virtual addresses
- Allocate backing memory

Files

- Get/Put file

Namespace

- Allocate/Free NS, Bind/Release names

Inter-process communication

- Send/receive

BasicOZ Device Model

Device emulators load in SPACE

- Implement access to device registers
- Call on SPACE to do DMA (background copies)
- Post interrupts at a specific IRQL

BasicOZ device access

- Access device registers
- Specify mapping of interrupts to handlers
- Control CPU IRQLs

ProjectOZ

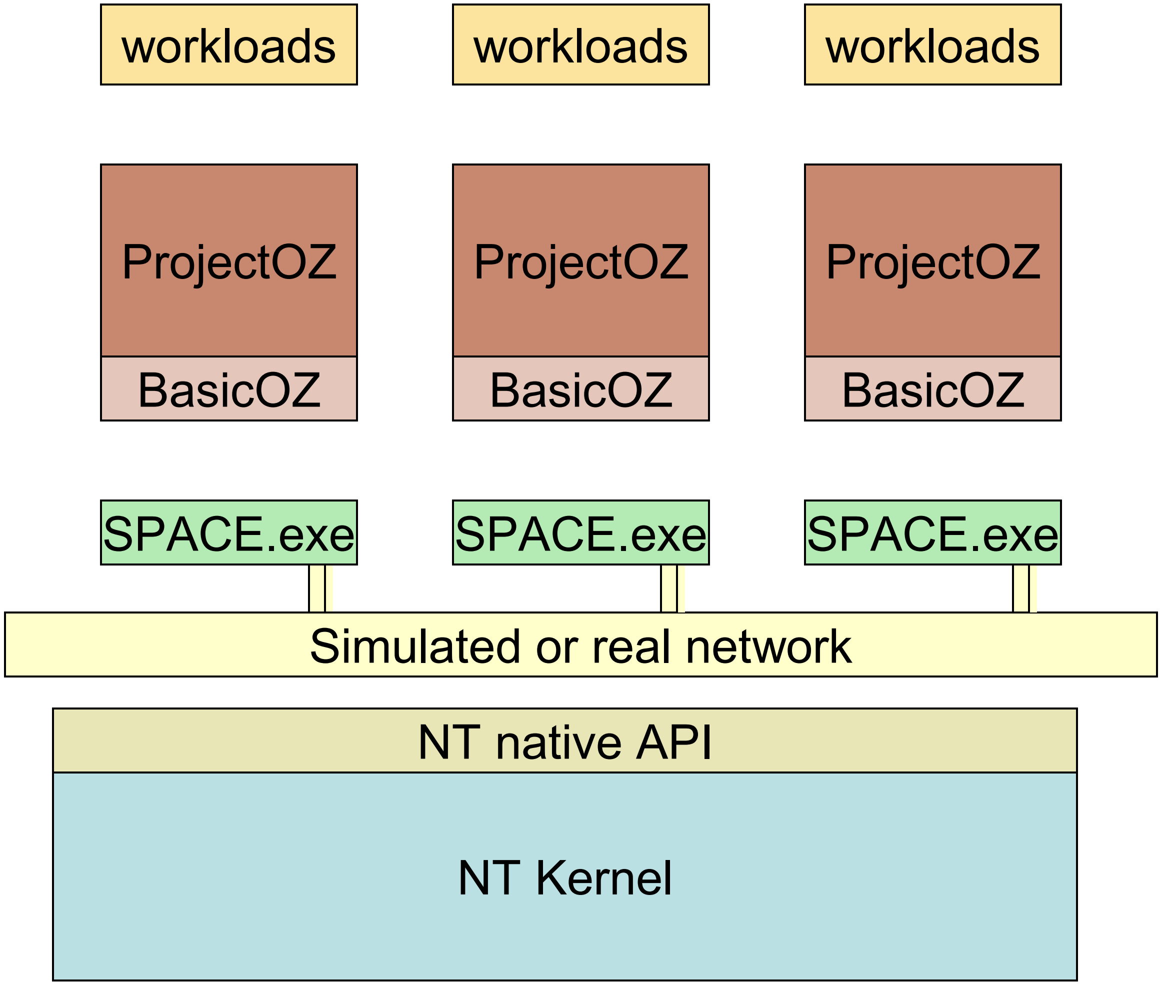
ProjectOZ refers to the projects students build

Projects take areas of BasicOZ with limited or missing functionality, poor algorithms, performance problems, etc, and extend the system

Examples

- Use timer to make threads preemptive
- Write a priority-based scheduler
- Implement open/read/write/code operations
- Add clock algorithm for pageout

ProjectOZ Multicomputer



Workloads, Instrumentation, Community

Workloads

Projects need a significant set of programs to exercise functionality, both for testing and evaluation

Instrumentation

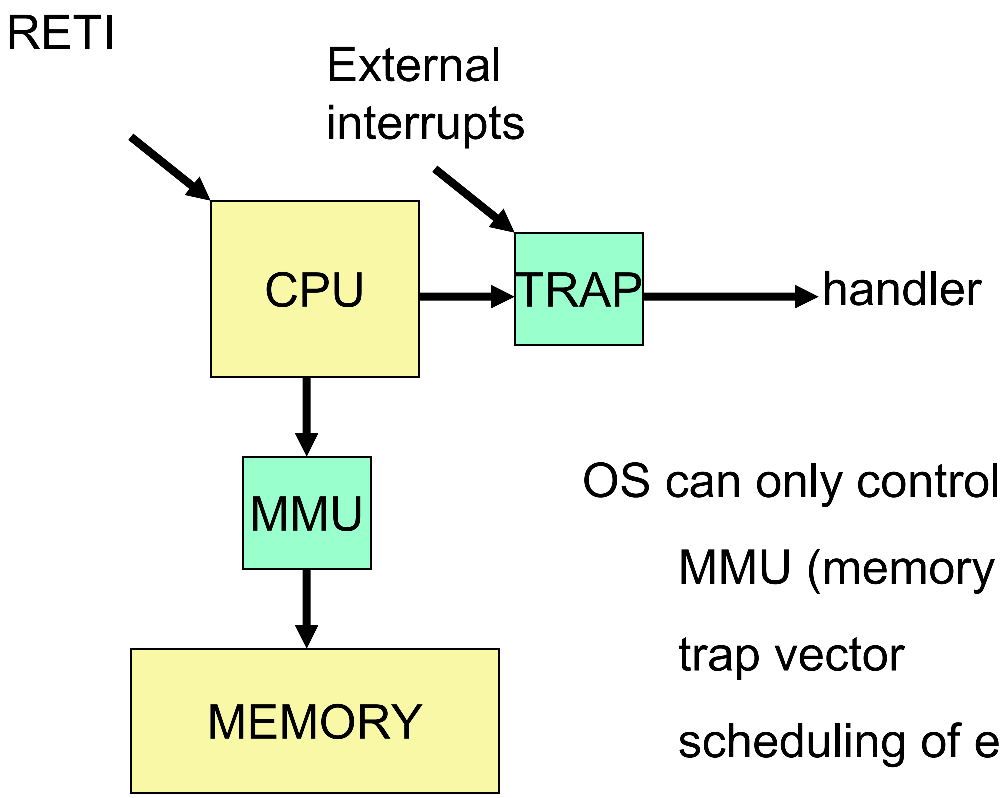
Still investigating how to appropriately instrument SPACE with measurements of CPU times and event counters to use for relative evaluation of projects

Community

ProjectOZ is successful if-and-only-if it proves valuable, in which case an academic community grows up around it – in which case Microsoft will assume a secondary role

SPACE

SPACE CPU Model



OS can only control:

- MMU (memory management unit)
- trap vector
- scheduling of external interrupts
- when it does an RETI (Return from Interrupt)

OS only regains control through trap/interrupt

SPACE Abstractions

CPU – sequences instructions until interrupted

- – traps, exceptions, interrupts, faults
- – CPU executes in a specific MMU context and CPU mode

MMU – maps virtual to physical addresses

- – invalid mapping/access causes a fault
- – each MMU context defines an (address) space
- – access for each mapping determined by CPU mode

PORTAL – specifies what to do when CPU is interrupted

- – portals specify new context, mode, and program counter
- – previous execution state preserved in a proc control blk
- – access to portals depends on mode

Primary SPACE Operations

Manage MMU and Trapvector

MapMemory(ctx, virtual, phys, modeaccess)

MapPortal(ctx, trap, pctx, pmode, ppc, modeaccess)

Portal Operations

Resume() – resumes at top entry on PCB chain

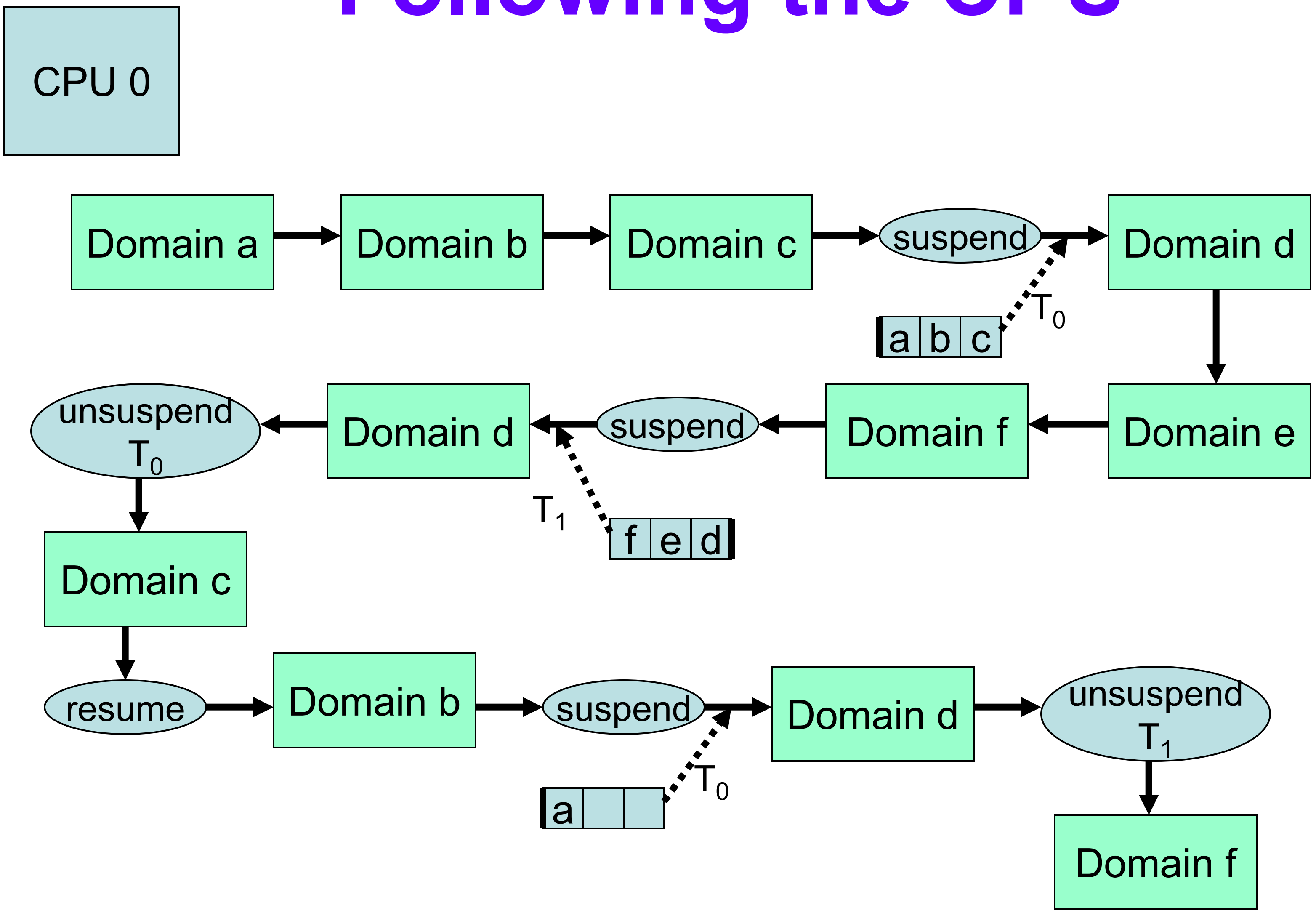
token = **Suspend**() – breaks current PCB chain, assigns token

Unsuspend(token) – like resume, but uses suspended chain

Portals generalize traps to multiple protection domains

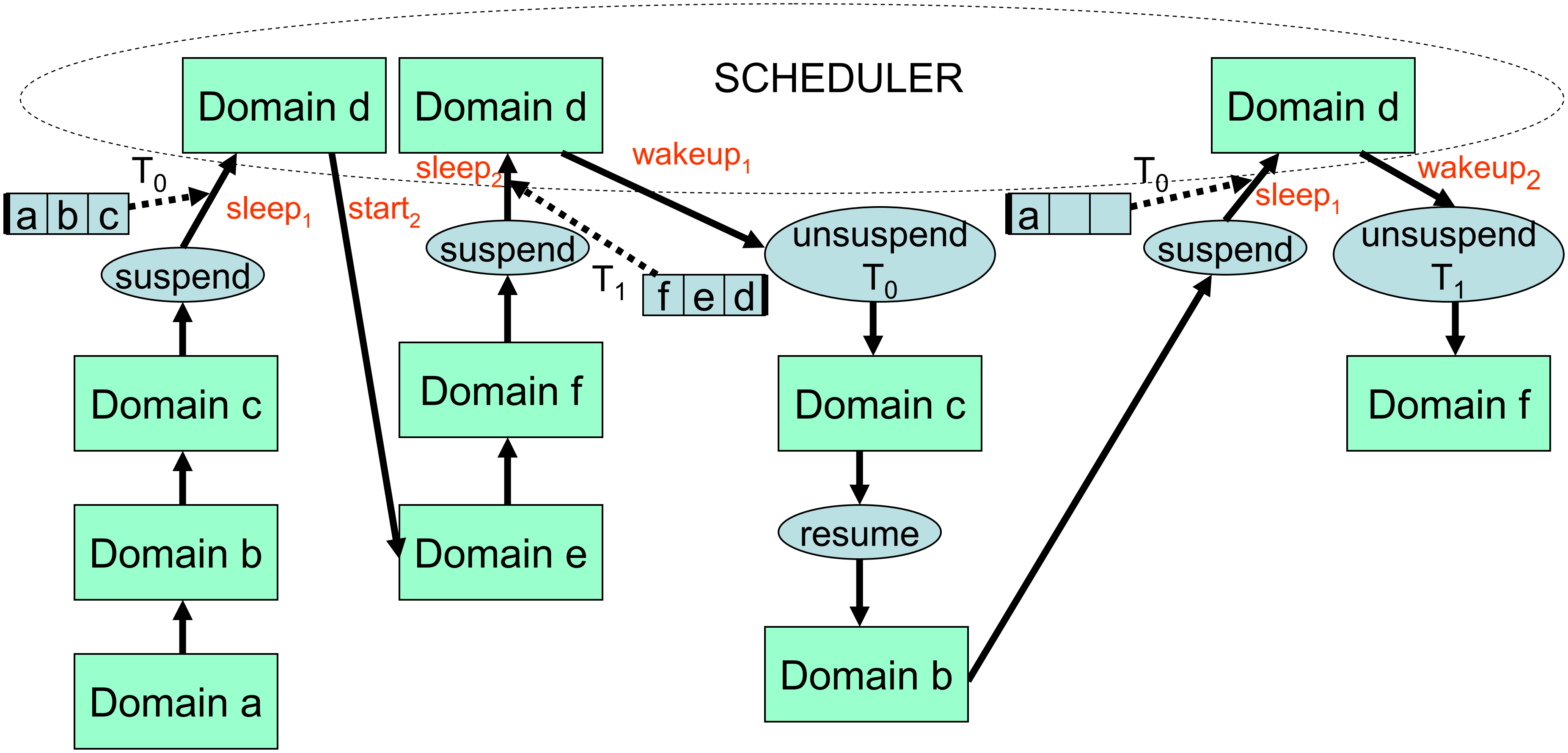
Processor context (PCB) implicitly managed

Following the CPU



CPU 0

Redrawing the picture



SCHEDULER

Domain d

Domain d

Domain d

a b c

T_0

sleep$_1$

start$_2$

sleep$_2$

wakeup$_1$

T_0

sleep$_1$

wakeup$_2$

a

suspend

suspend

suspend

T_1

f e d

unsuspend T_0

unsuspend T_1

Domain c

Domain b

Domain a

Domain f

Domain e

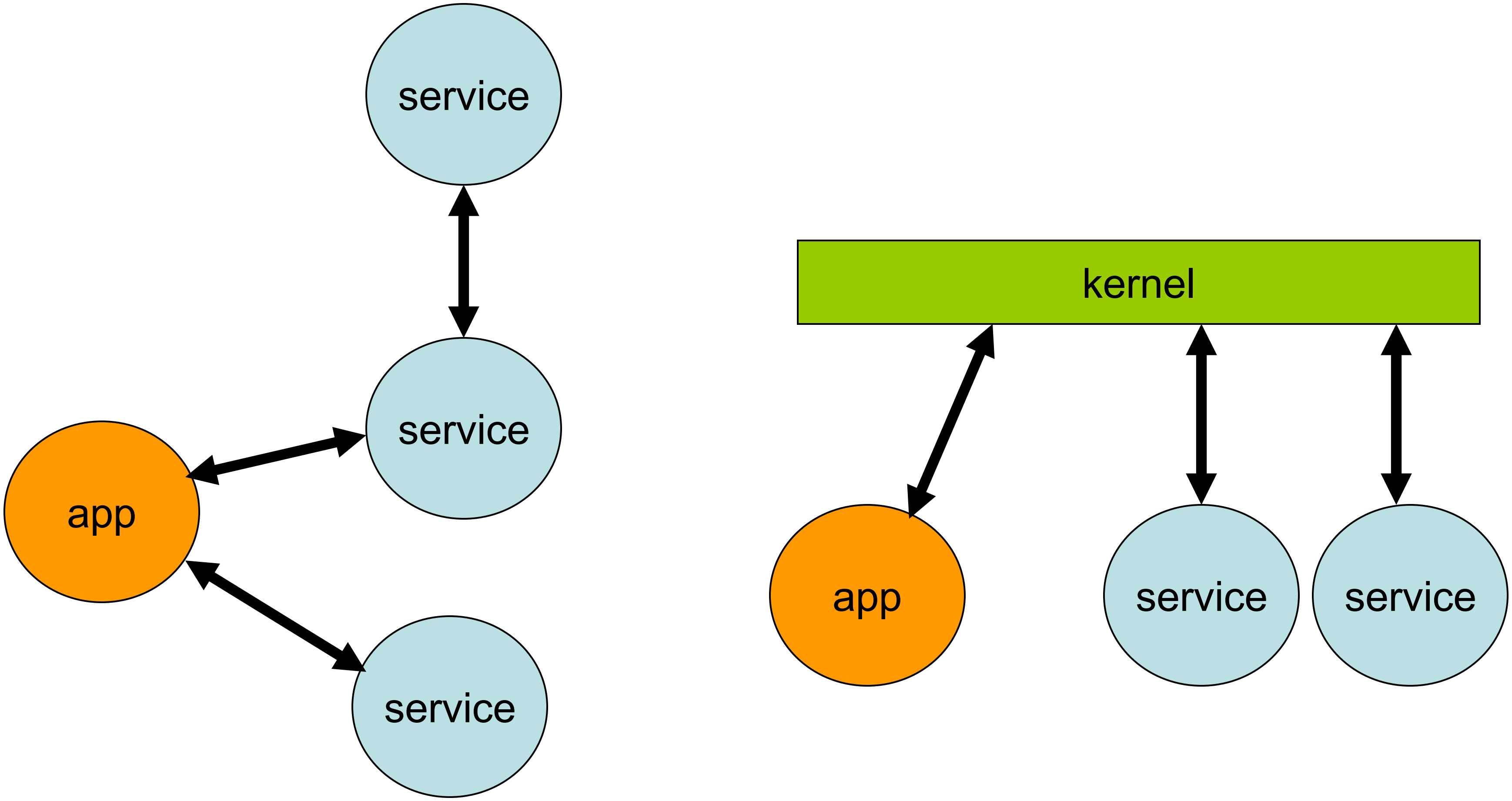
Domain c

resume

Domain b

Domain f

The General SPACE case vs kernel



SPACE using native NTAPI

NT Facilities used for SPACE

Objects

Threads – NT unit of CPU scheduling

Processes – NT virtual address space container

Sections – NT sharable memory objects

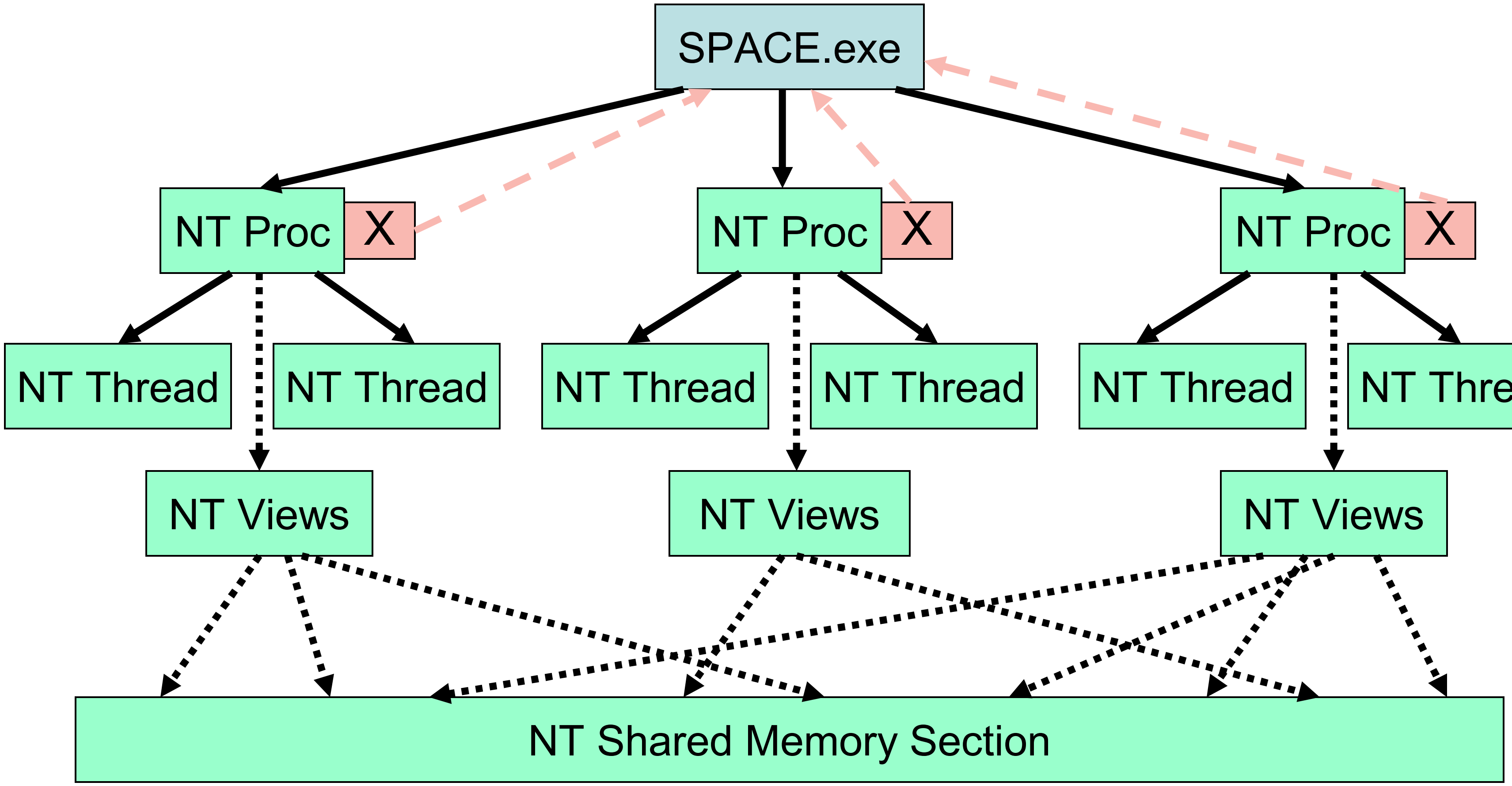
Exception port – NT mechanism for subsystem fault handling

Functions

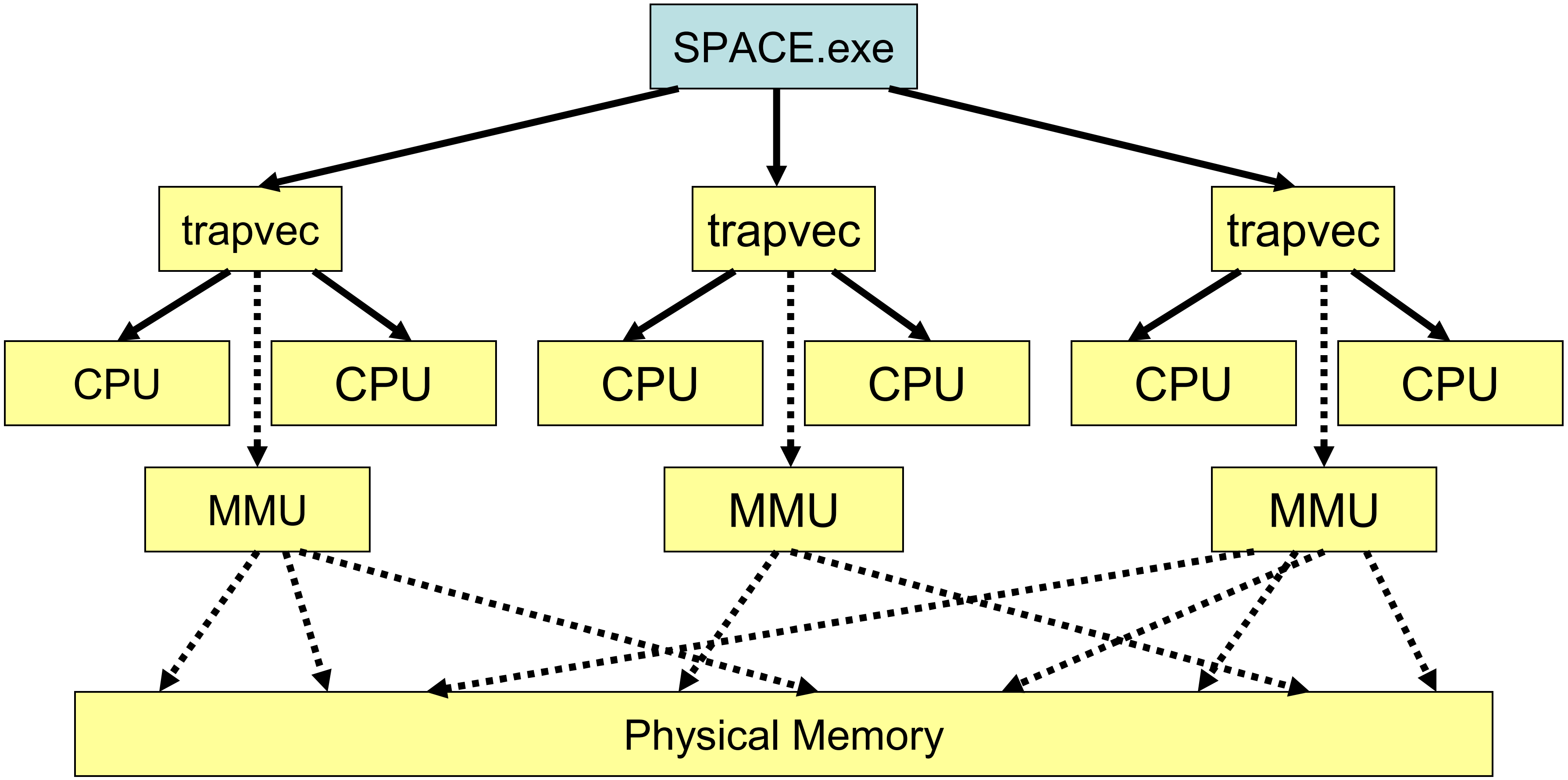
MapView – Map process addresses to section offsets

Wait/Reply port – Receive/Send message to port

ProjectOZ using NT



ProjectOZ using NT



SPACE.exe

trapvec

trapvec

trapvec

CPU

CPU

CPU

CPU

CPU

CPU

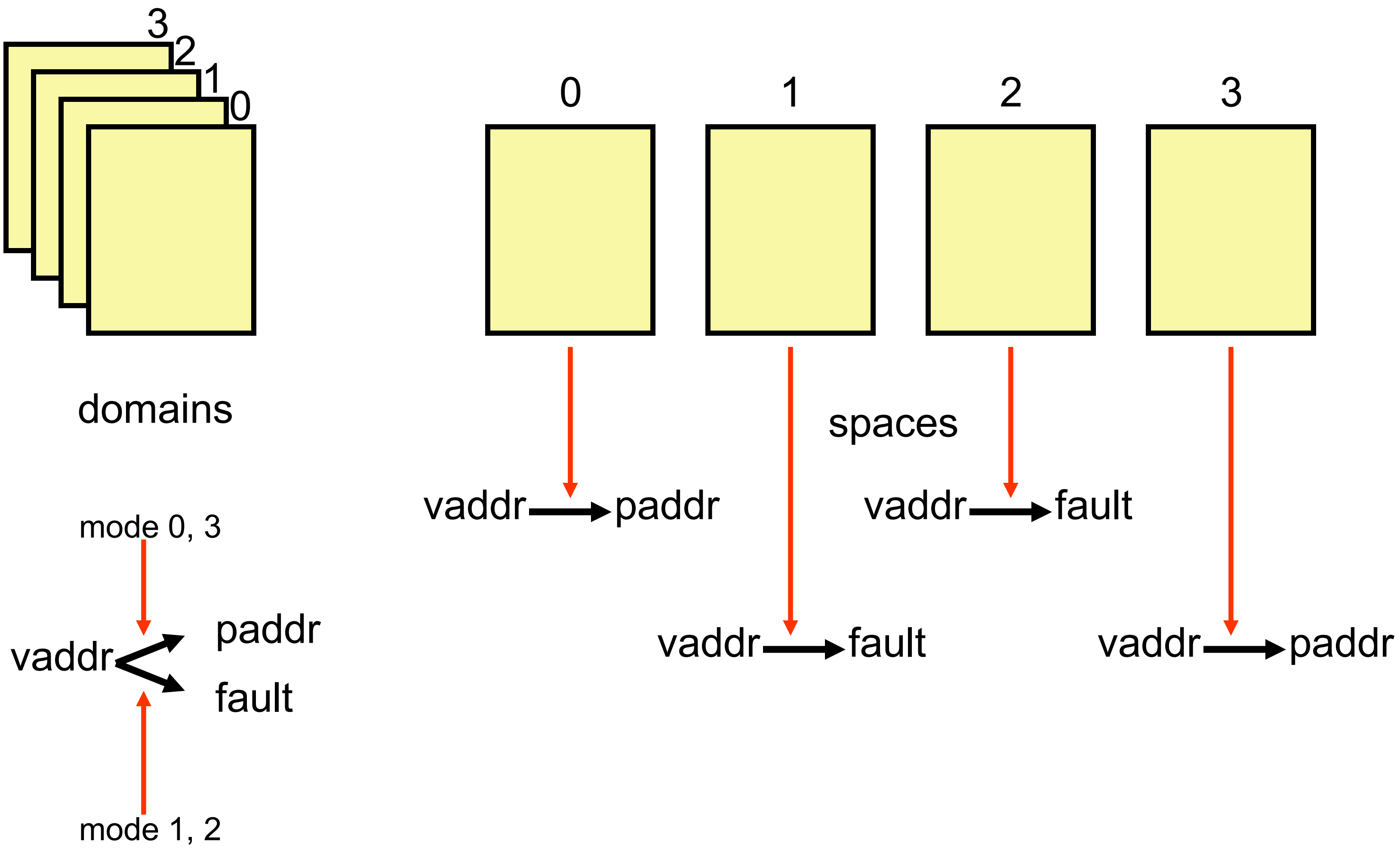
MMU

MMU

MMU

Physical Memory

domains == overlaid VA spaces



3
2
1
0

domains

0
1
2
3

spaces

vaddr → paddr

vaddr → fault

vaddr → fault

vaddr → paddr

mode 0, 3

vaddr

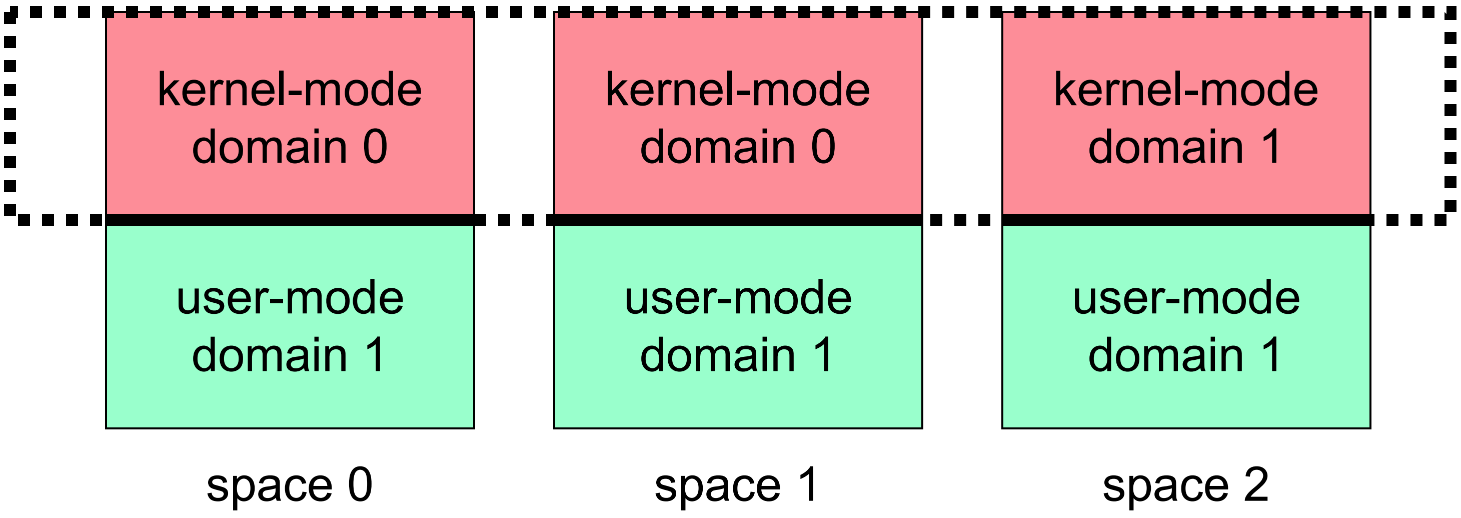
paddr

fault

mode 1, 2

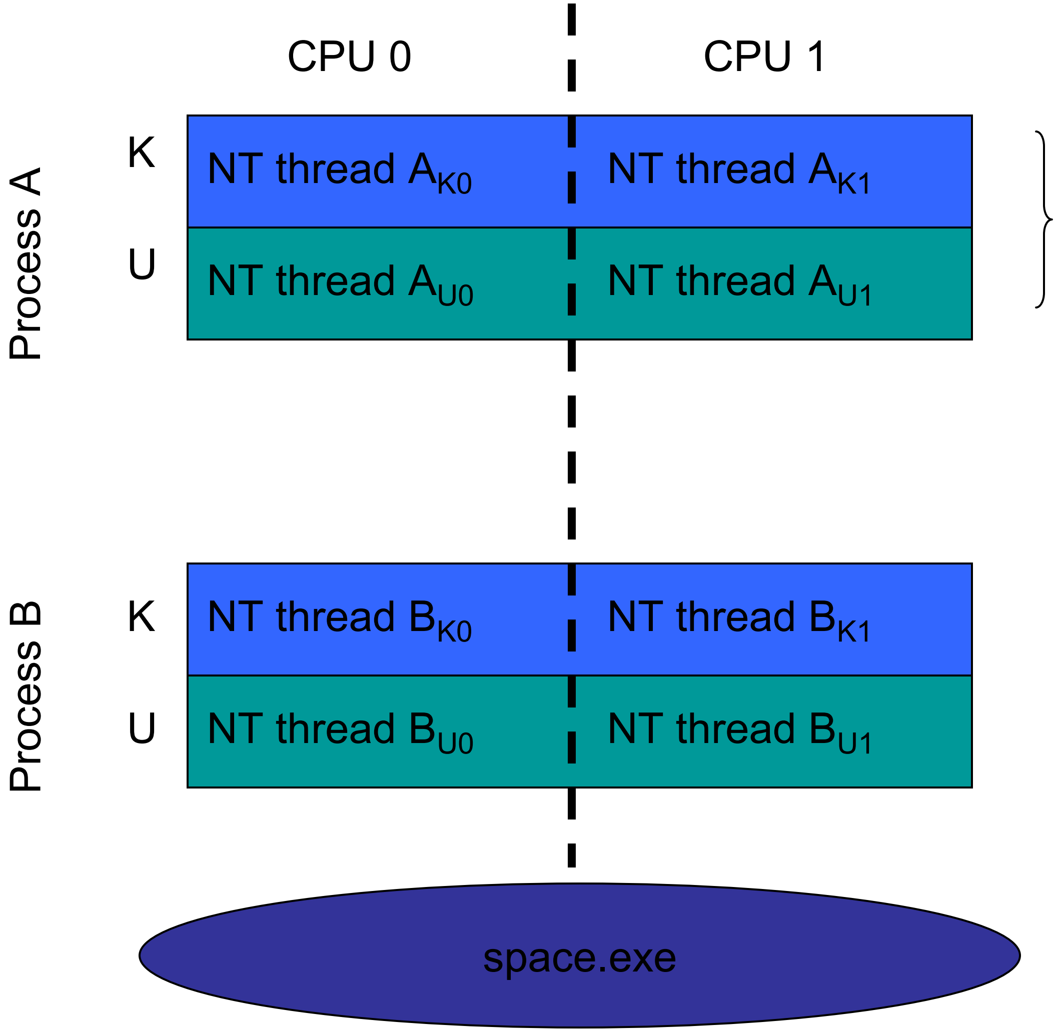
Kernels: special case of SPACE

Kernel-mode memory mappings (mostly) shared in all spaces



spaces used to build processes

Building CPUs out of NT threads



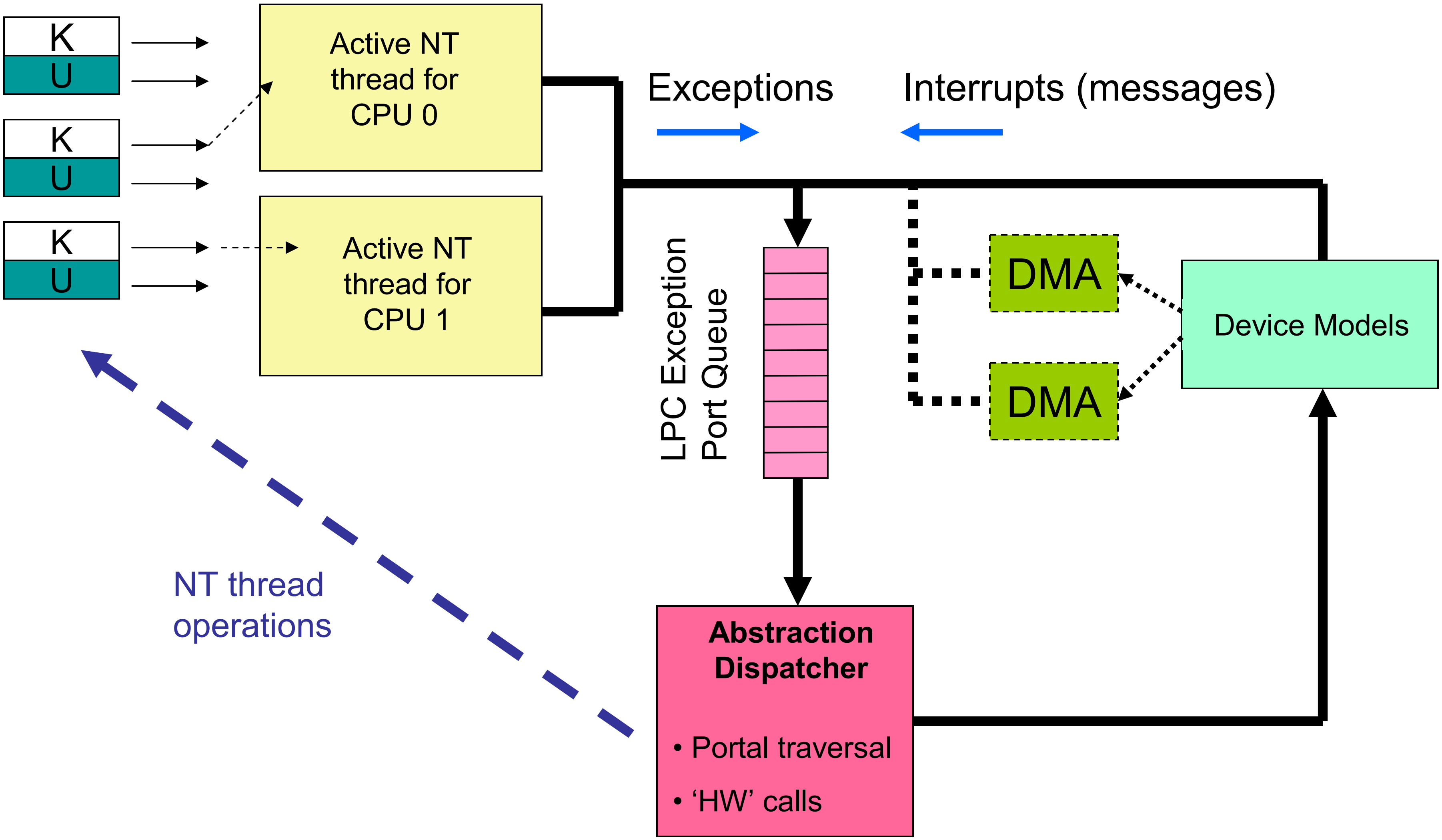
Two NT processes with same mappings, different protections

Space.exe uses *baton passing* so only one NT thread per CPU runs at a time

Limitations

- Some artifacts of NT still exist within spaces
  - Certain parts of address space have been claimed beyond our control
  - Ntdll – Mapped into every NT process
    - Unavoidable
    - Required for Exception port trampoline anyway
  - Lower 4MB of address space reserved
  - Additional space used by PEB/TEB
- Shared view granularity on a section is 64kb, restricting us to a 64kb page/frame size
- Unable to query dirty & reference bits
  - *NtWriteWatch* doesn't work on shared sections

Space.exe Control Flow



SPACE Device Model

Running inside SPACE.exe. Device models:

- Export function to emulate device access
- Call StartDMA function to emulate DMA between ‘physical memory’ and ‘device memory’
- SendInterrupt to a CPU
- Respect IOMMU and IRQL emulated for each processor
- ‘Software’ interrupts can be used to defer processing

Advanced devices

- Alternate interrupt schemes (mapping, preferred CPU, …) by modifying SPACE
- Add instrumentation and physical simulation (e.g. seek times, packet loss, errors)
- Can build ‘smart’ devices – it is all just software anyway
- Memory-mapping of device registers (fault handling)
- Per-device IOMMU, mask-based interrupts

BasicOZ

BasicOZ elements

- Kernel Object Management
- Name Space Management
- Address Space Management
- Paging
- Threading
- Processes
- Interrupts, Traps, System calls
- Driver model
- Booting & Initialization
- User-mode model

Kernel Object Management

- **Objects allocated from static pools**
- **Object states:**
  - **Free – available to be allocated**
  - **Allocated – assigned to thread, has refs**
  - **Activated – in-service**
  - **Shutting-down – no new access**

  Managed by references

  Separation of storage allocation from object use
- **Object instances have IDs**
  - **Lookup by thread or type**

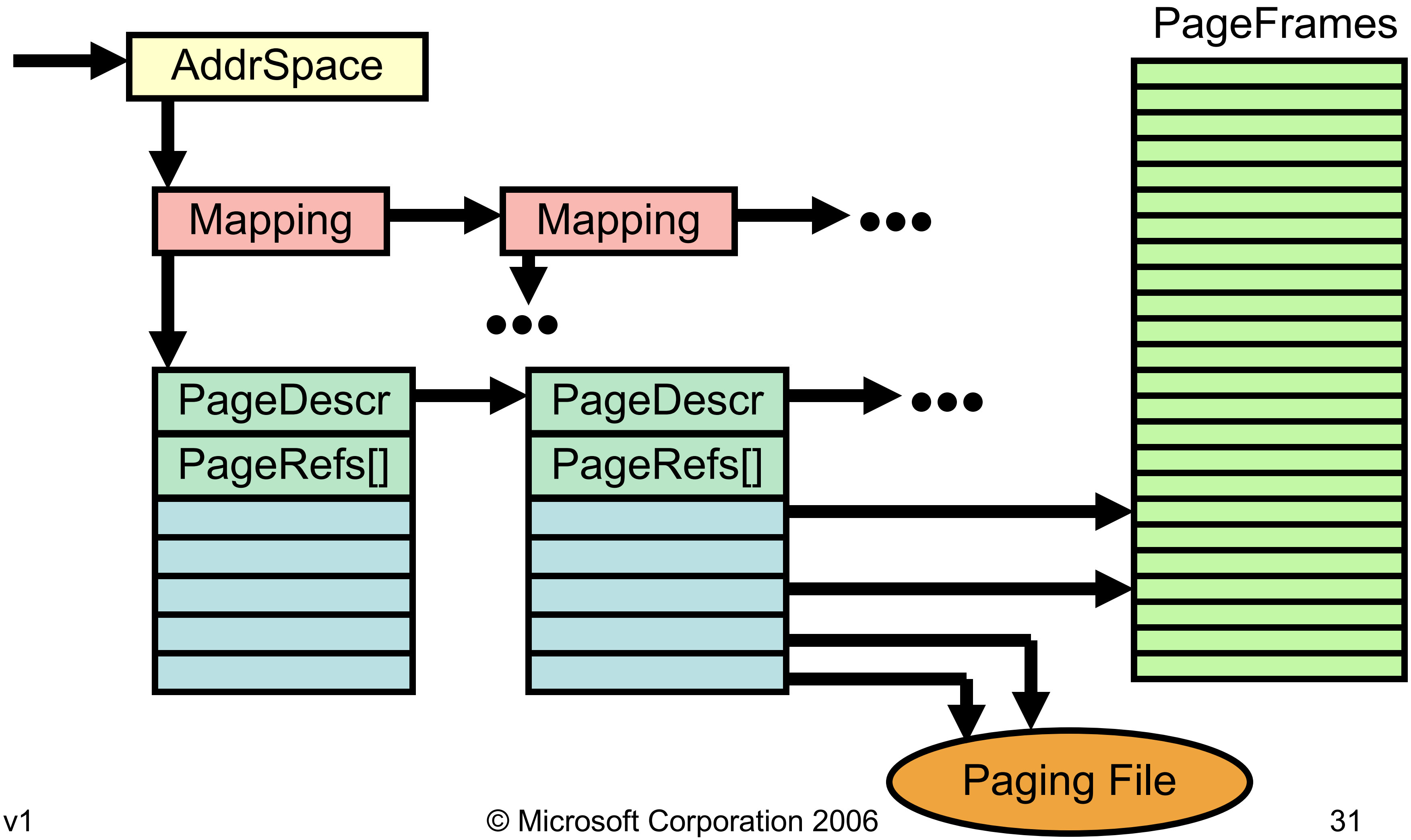
Name Space Management

- Name Spaces:
  - (ns, name) -> object
- Recursive: *objects can be Name Spaces*
- Lookup within a Name Space or recursively search reachable Name Spaces
- Name Space can be extended to persistent stores
- No central root
- Each process has starts with two Name Spaces
  - Shared – finds objects passed from parent
  - Private – not shared with parent
  - New Name Spaces can be readily created

Address Space Management

- Main data structures
  - AddrSpace, Mapping, PageDescr (with PageRefs)
- activateaddrspace(as, hwspaceid)
  - binds AS to a 'hardware' context
- activatemapping(map, npages, prot, PDlist)
  - binds map to PageDescr, sets protection
- linkmapping(as, map, vpage)
  - links map to as at vpage (no sharing)
- Main operations
  - findmap(as, vpage) and findpageref(map, vpage)
- Special operations for I/O mapping

Address Space Structures



Paging

- Allocate memory pageframes
- Allocate pageframes within pagefile
  - uses simple linked list of free pages
- Page-in, page-out, handle faults
  - Working-set based
  - Waits for pages in transition
  - No soft-faults
- Reference counts lock pages
  - E.g. for I/O operations

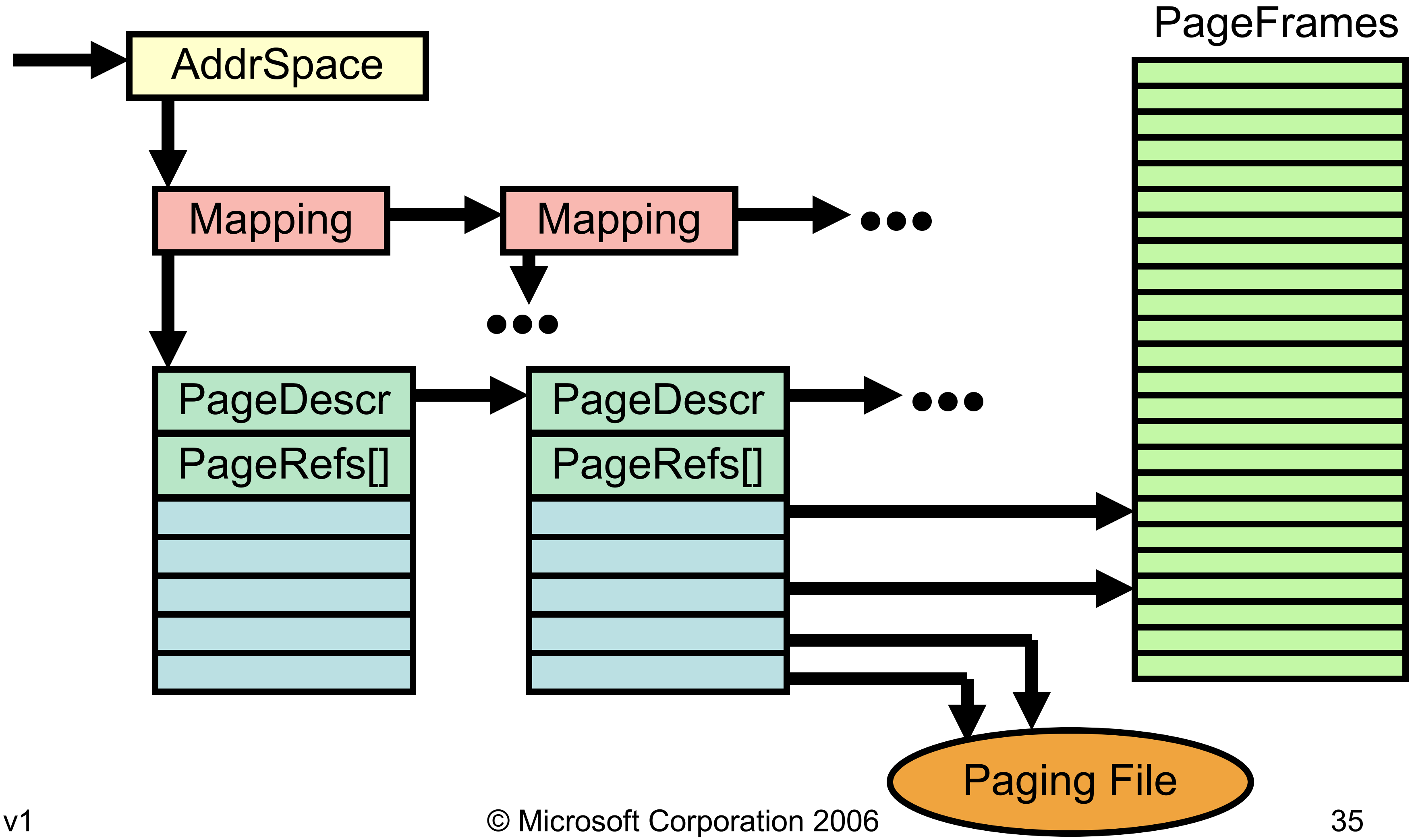
Threading

- activatethread(thread, process, waitvalue, startinfo)
  - Queues thread for run/wait
  - First time run starts at kernel routine
  - Kernel routine may enter user-mode through a portal
- Block by calling await(value)
  - Uses portal traversal to capture state
- signal(value) makes thread awaiting value runnable
- threads exit by returning (i.e. to scheduler)
- yield is await(0)
- preemption is involuntary yield()

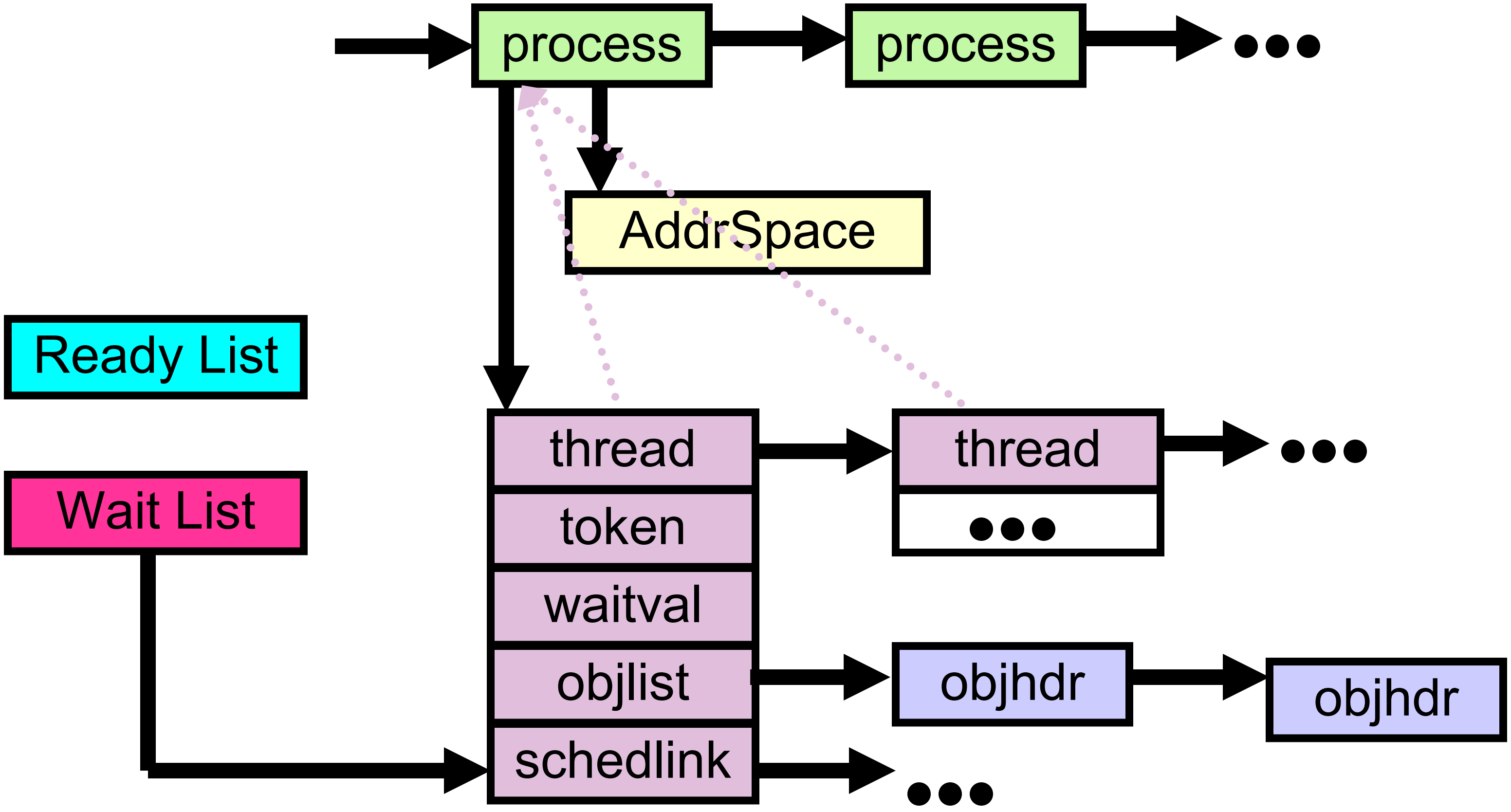
Processes

- Programs are NT executables
- BasicOZ allocates resources and loads
- Two initial Name Spaces
  - Shared: get parameters, arguments, files, and other objects from creating process
  - Private: private object directory
- Capability-based
  - Control of portal mappings controls access
    - SPACE_* hardware emulation
    - System Calls

Address Space Structures



Process & Thread Structures



Interrupts, Traps, et al

- **Implemented using SPACE portals**
  - Traps map to Portals
  - Portals specify (space,prot) [aka (ctx,mode)]
  - Stacks are dynamically allocated
  - Traps generalized
    - UD2 trap augmented with parameter (syscall number)
    - Different portals can map to different pagefaults
  - SPACE_* ‘instructions’ execute in SPACE.exe
    - Errors in SPACE_* => illegal instruction exceptions
    - Other traps, execeptions, interrupts => portal traversal
  - Glue code is Bootstrap.asm and Machine.asm

Driver Model

- **Device Models link with SPACE.exe**
  - Devices register SPACE by ‘device ID’
  - Device models implement device registers & memory
  - StartDMA transfers between device memory & physical memory through the IOMMU
  - Devices can interrupt a CPU at a specified IRQL
- **SPACE_MapIO()**
  - Supports IOMMU access from drivers in BasicOZ
- **SPACE_AccessDevice()**
  - Provides access to device registers from BasicOZ
- **Trap/Portals provide interrupt mechanism**

Booting & Initialization

- SPACE.exe %rundir%
  - Creates new domains via bootstrap.exe
  - Loads BasicOZ.boz and invokes boot()
  - SPACEOps.c and %arch%\Machine.asm invoke SPACE_* emulation instructions by executing illegal instructions
- SPACE uses native NT functionality
- BasicOZ uses only SPACE (& syslib)

Status

- **Code for SPACE.exe v1 available July 2006**
- **SPACE v2 and BasicOZ v1 available soon**
  - Watch community forums or MSDNAA
- **Work ahead**
  - Documentation
  - NCPUS > 1
  - Multicomputer support
  - x64 support
  - Instrumentation and Workloads
  - Projects, community involvement
  - WRK-enhancements, Rotor, C#, VisualStudio

Questions & Discussion