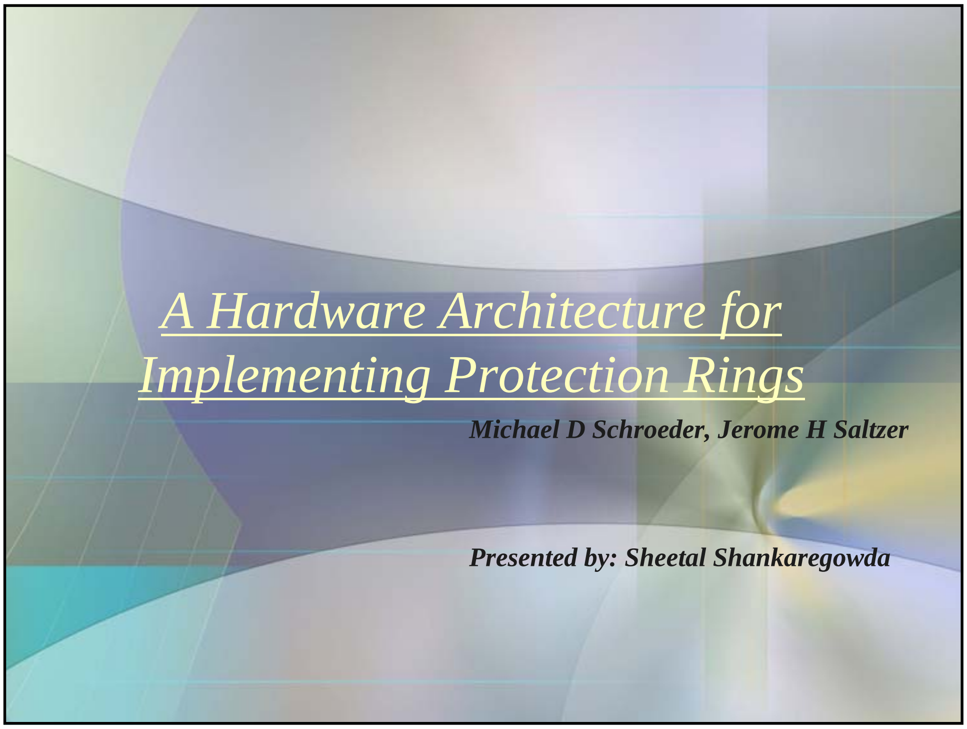
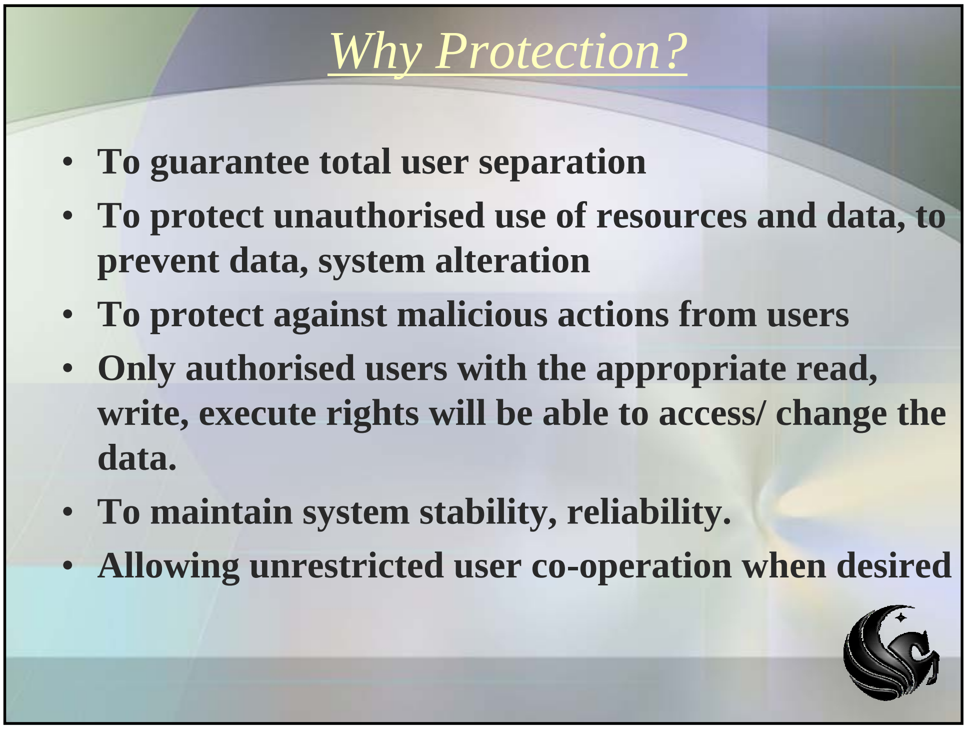


A Hardware Architecture for Implementing Protection Rings

Michael D Schroeder, Jerome H Saltzer

Presented by: Sheetal Shankaregowda



Why Protection?

- To guarantee total user separation
- To protect unauthorised use of resources and data, to prevent data, system alteration
- To protect against malicious actions from users
- Only authorised users with the appropriate read, write, execute rights will be able to access/ change the data.
- To maintain system stability, reliability.
- Allowing unrestricted user co-operation when desired

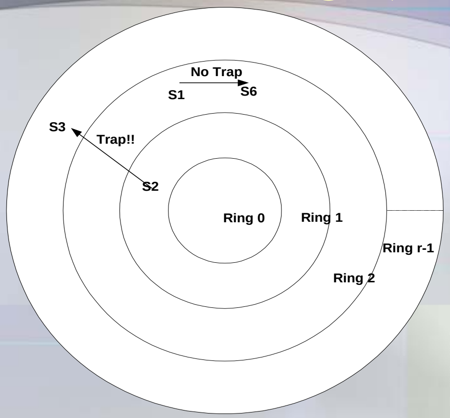
Protection Ring

- Fixed number of domains for each process.
- Rings numbered from 0 to r-1.
- Access capabilities of ring n is subset of m if n>m.
- Multi-layer protection.
- Calling segment in another ring → trap
- Multics has 7 rings/ domains.

R	W	E	BASE	LIMIT	RING NUMBER	GATE

Segment descriptor

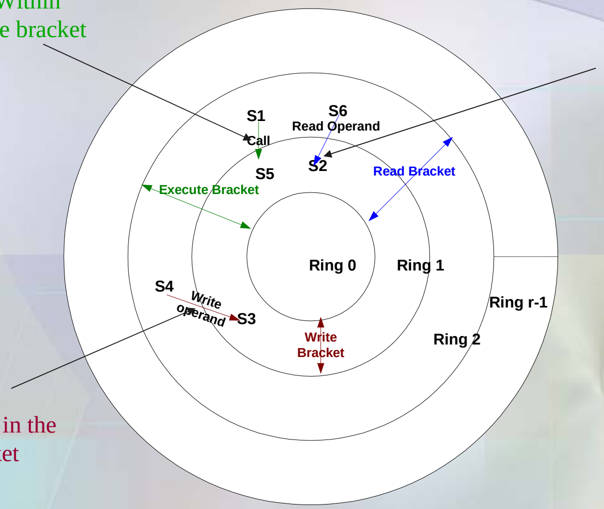
Protection rings (cont..)



- If an application calls segments which are in different rings, then it generates lot of traps.
- Need a way to minimise the traps and allow inter-ring access.

Access Brackets

No Trap. Within the execute bracket



No Trap, within the read bracket

Trap!! Not in the write bracket

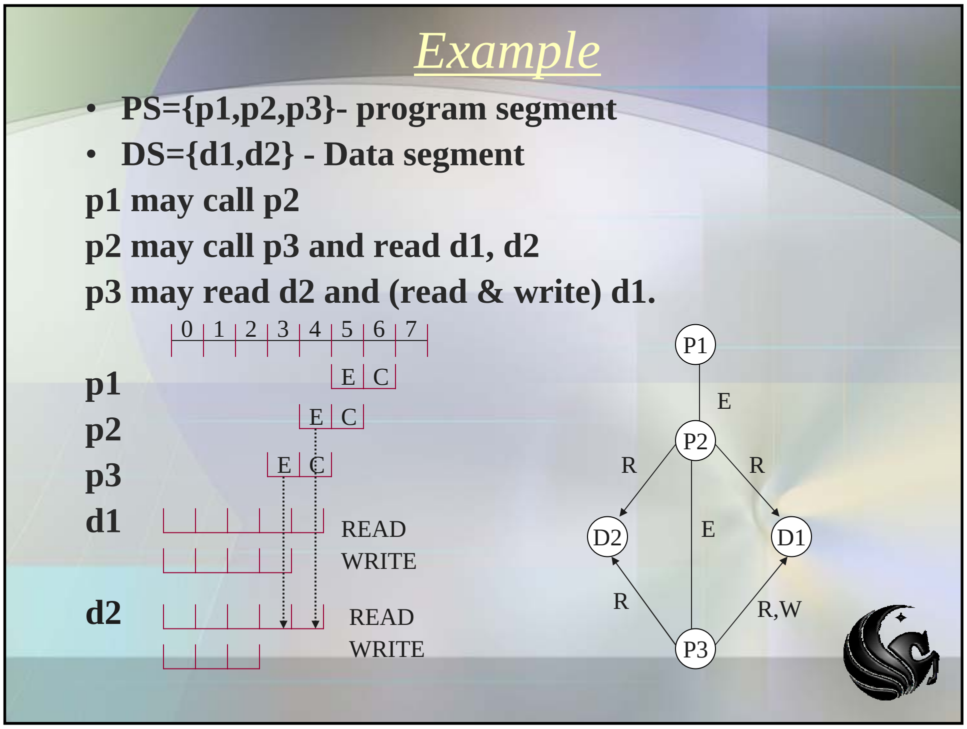
Access Brackets

- Allocate each segment to a set of consecutive rings.
- Any segment within access bracket ‘N’ will not generate trap.
- N1 Λ N2 : execution bracket

R	W	E	BASE	LIMIT	N1	N2	N3	GATE

Segment descriptor

- N2 Λ N3 : call bracket
- A call by a process executing in ring i to procedure in a segment with execution bracket N1, N2 does not generate trap if N1 ≤i ≤N2.
- Process executing in ring ‘i’ can access segment in ring ‘k’ with i ≤k, the access is governed by the access indicators of segment addressed.



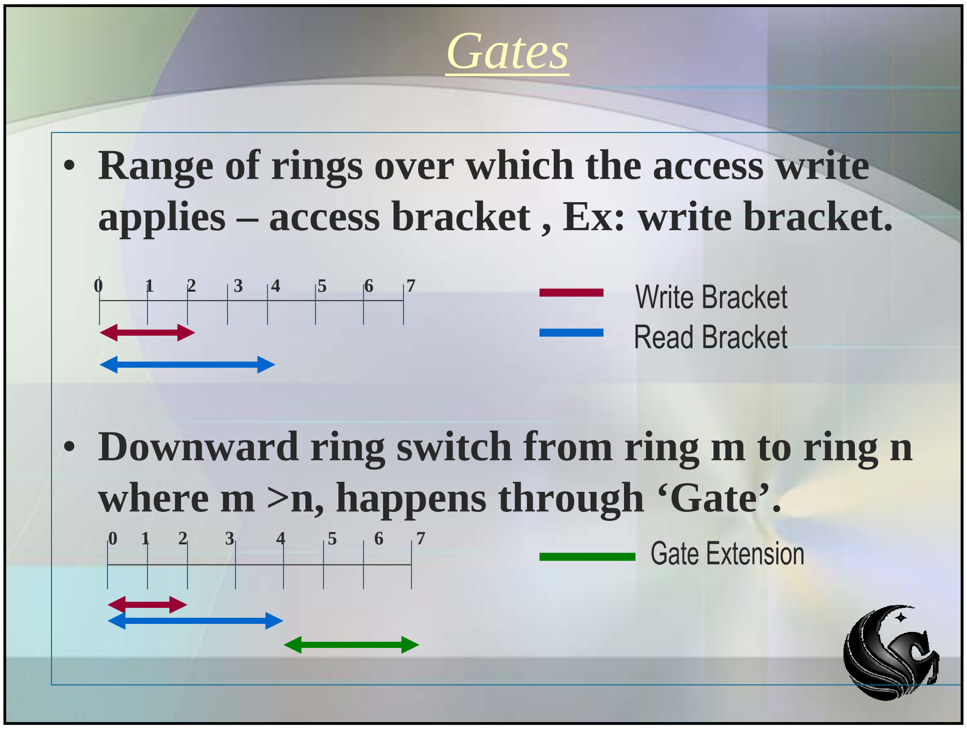
Example

- PS={p1,p2,p3}- program segment
- DS={d1,d2} - Data segment

p1 may call p2

p2 may call p3 and read d1, d2

p3 may read d2 and (read & write) d1.

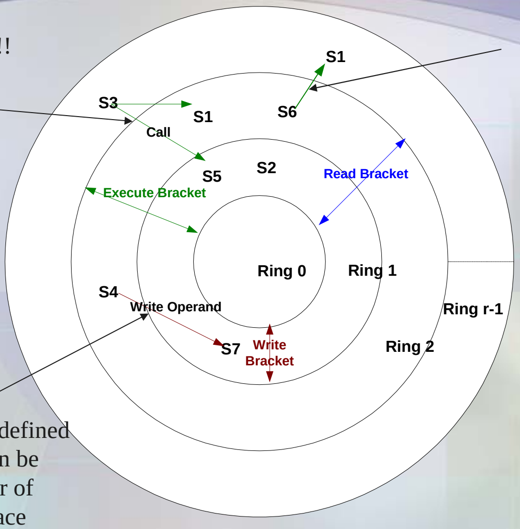


Gates

- Range of rings over which the access write applies – access bracket , Ex: write bracket.

- Downward ring switch from ring m to ring n where m >n, happens through 'Gate'.

Gates



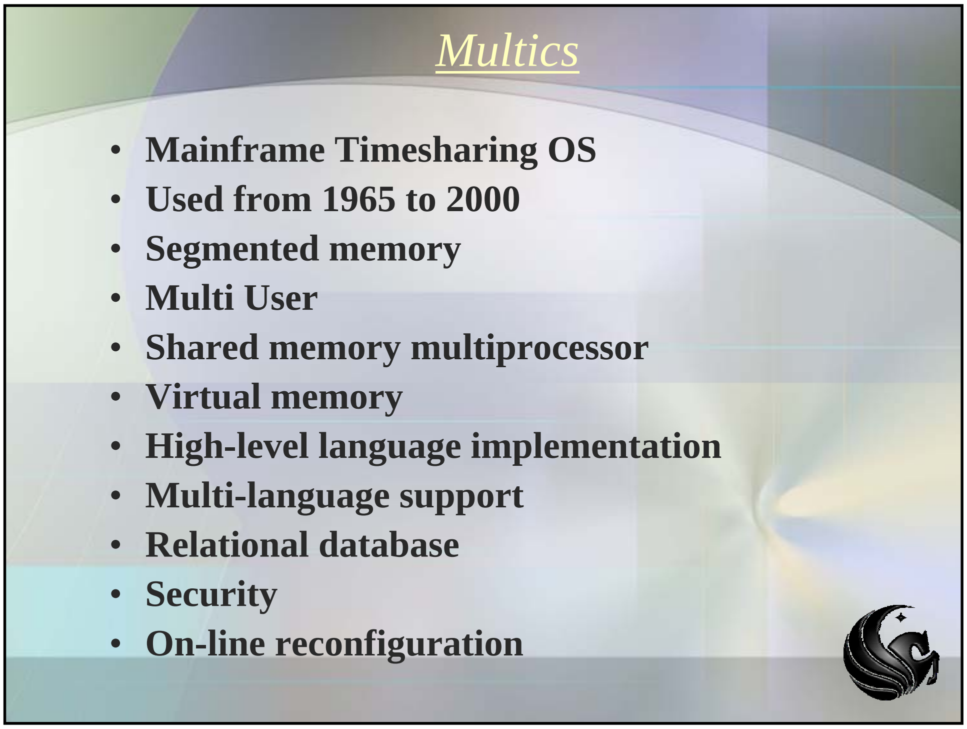
No access bracket!! Has to go through Gate. If gate location in SDW, transfer of control takes place, else access violation.

Access Violation!! Call to an upward ring. Cannot transfer control though gate.

If gate location defined then operand can be written. Transfer of control takes place through the gate, else results in trap.

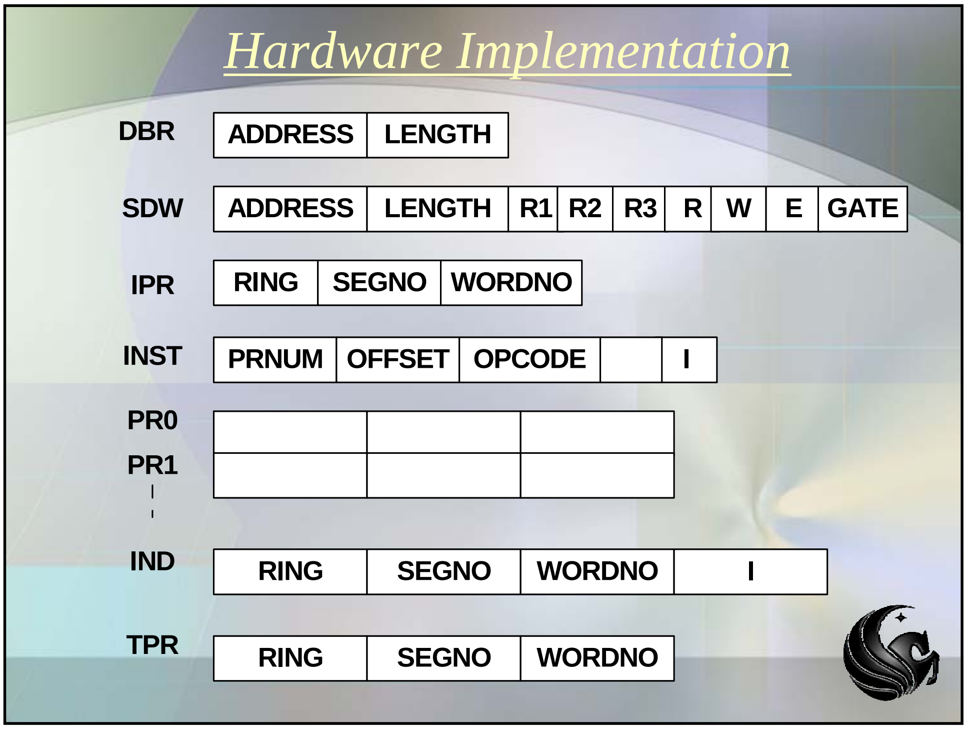
Gates (Cont…)

- **Downward ring switching capability must be coupled to transfer of control to a gate into the lower numbered ring**
- **Specified by associating list of gate locations with each segment in the virtual memory of a process**
- **If execution point of process is transferred to segment while ring of execution is above the top of execute bracket for the segment, then transfer must be directed to one of the gate locations in the segment**
- **Transfer not allowed if not directed to gate location**
- **If transfer is to a gate, then the ring of execution of the process will switch down to the top of bracket of segment as the transfer occurs**



Multics

- Mainframe Timesharing OS
- Used from 1965 to 2000
- Segmented memory
- Multi User
- Shared memory multiprocessor
- Virtual memory
- High-level language implementation
- Multi-language support
- Relational database
- Security
- On-line reconfiguration



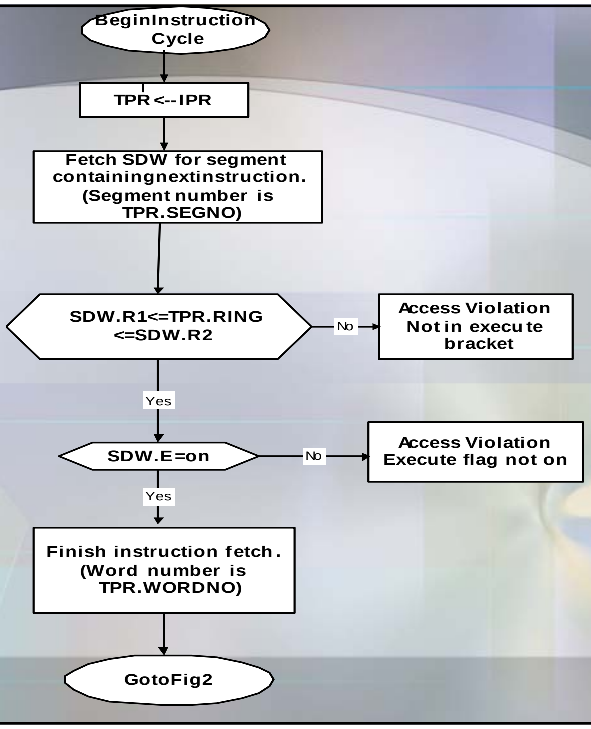
Hardware Implementation

DBR	ADDRESS	LENGTH							
SDW	ADDRESS	LENGTH	R1	R2	R3	R	W	E	GATE
IPR	RING	SEGNO	WORDNO						
INST	PRNUM	OFFSET	OPCODE		I				
PR0									
PR1									
IND	RING	SEGNO	WORDNO	I					
TPR	RING	SEGNO	WORDNO						

Hardware Implementation (cont..)

- Write bracket : ring 0 to SDW.R1.
- Read bracket : ring 0 to SDW.R2
- Execute bracket: SDW.R1 to SDW.R2
- Gate extension: SDW.R2+1 to SDW.R3.

  Provided SDW.R1≤SDW.R2 ≤SDW.R3.
- IPR- current ring of execution and two part address of next instruction.
- Program accessible pointers(PR0, PR1..) contains two part address and ring number.
- TPR : internal processor register not accessible by program.



BeginInstruction Cycle

TPR <--IPR

Fetch SDW for segment containingnextinstruction. (Segment number is TPR.SEGNO)

SDW.R1<=TPR.RING <=SDW.R2

No

Access Violation Not in execute bracket

Yes

SDW.E=on

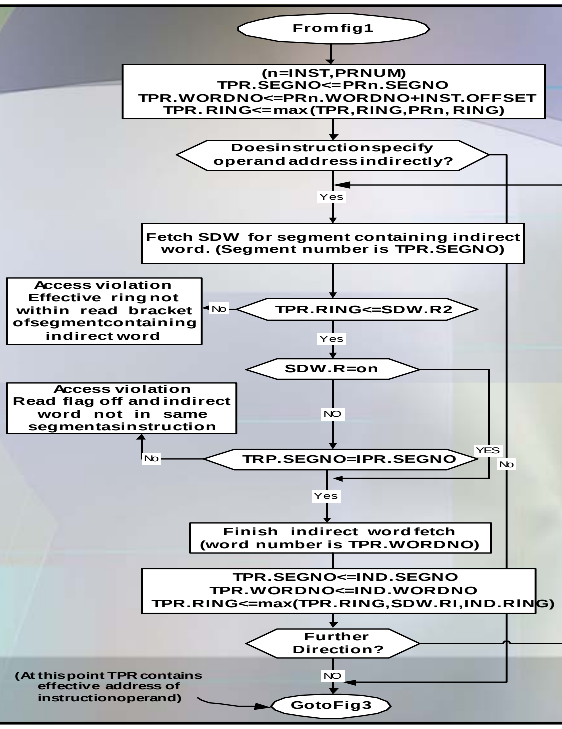
No

Access Violation Execute flag not on

Yes

Finish instruction fetch. (Word number is TPR.WORDNO)

GotoFig2



Fromfig1

(n=INST,PRNUM)
TPR.SEGNO<=PRn.SEGNO
TPR.WORDNO<=PRn.WORDNO+INST.OFFSET
TPR. RING<=max(TPR,RING,PRn, RING)

Doesinstructionspecify operandaddressindirectly?

Yes

Fetch SDW for segment containing indirect word. (Segment number is TPR.SEGNO)

TPR.RING<=SDW.R2

No

Access violation
Effective ring not within read bracket ofsegmentcontaining indirect word

Yes

SDW.R=on

NO

YES

TRP.SEGNO=IPR.SEGNO

No

Access violation
Read flag off and indirect word not in same segmentasinstruction

Yes

Finish indirect word fetch (word number is TPR.WORDNO)

TPR.SEGNO<=IND.SEGNO
TPR.WORDNO<=IND.WORDNO
TPR.RING<=max(TPR.RING,SDW.RI,IND.RING)

Further Direction?

YES

No

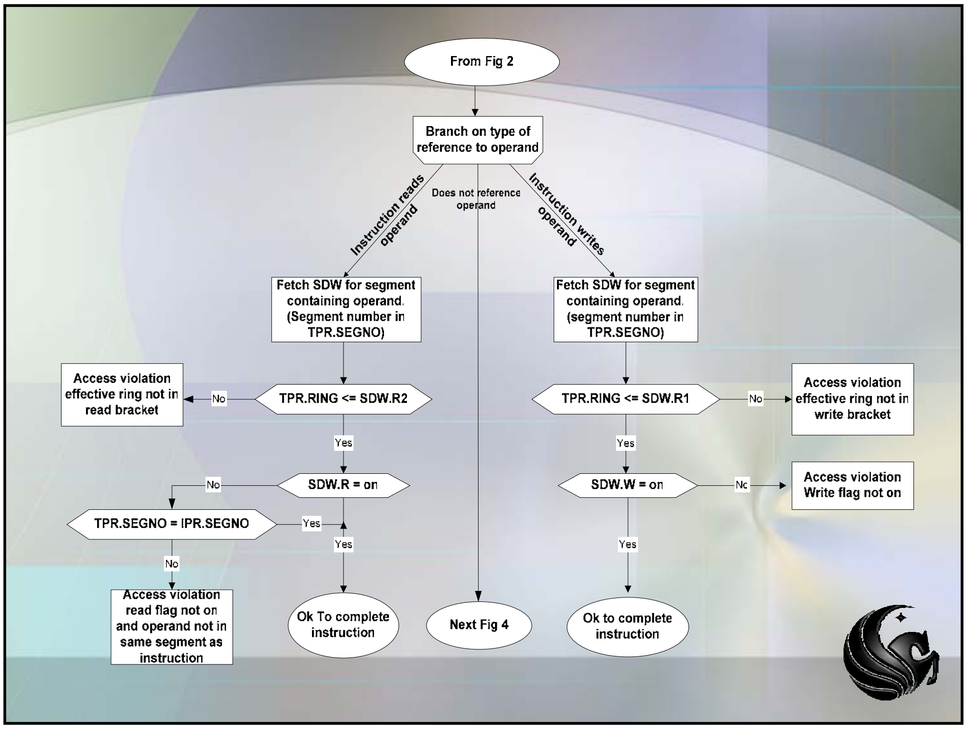
NO

(At thispoint TPR contains effective address of instructionoperand)

GotoFig3

Fig 1 & 2

- **Retrieve next instruction to execute**
- **TPR.RING matched against execute bracket defined in SDW**
- **If instruction can be executed in current ring, instruction fetch complete.**
- **Calculates TPR effective address of instruction operand**
- **Occurs if instruction operand in memory**
- **Change in TPR.RING – during effective address calculation, or during processing of indirect words involved in effective address caluclation**



From Fig 2

Branch on type of reference to operand

Instruction reads operand

Does not reference operand

Instruction writes operand

Fetch SDW for segment containing operand. (Segment number in TPR.SEGNO)

Fetch SDW for segment containing operand. (segment number in TPR.SEGNO)

Access violation effective ring not in read bracket

No

TPR.RING <= SDW.R2

Yes

SDW.R = on

No

TPR.SEGNO = IPR.SEGNO

Yes

Yes

No

Access violation read flag not on and operand not in same segment as instruction

Ok To complete instruction

Next Fig 4

TPR.RING <= SDW.R1

No

Access violation effective ring not in write bracket

Yes

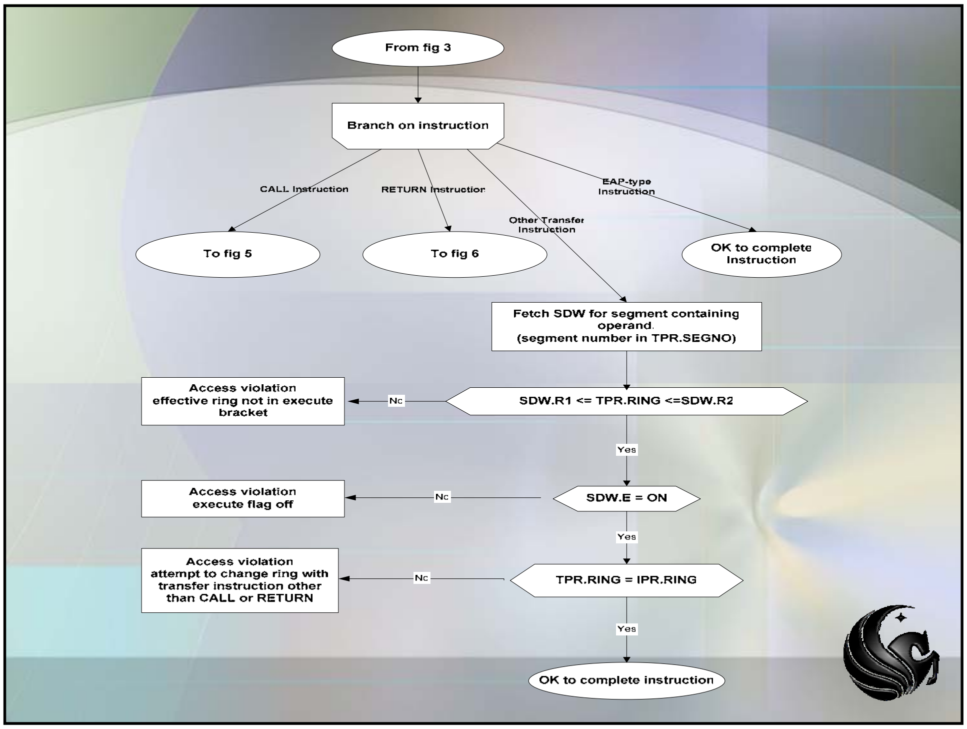
SDW.W = on

No

Access violation Write flag not on

Yes

Ok to complete instruction



From fig 3

Branch on instruction

CALL Instruction

RETURN Instruction

Other Transfer Instruction

EAP-type Instruction

To fig 5

To fig 6

OK to complete Instruction

Fetch SDW for segment containing operand. (segment number in TPR.SEGNO)

Access violation effective ring not in execute bracket

No

SDW.R1 <= TPR.RING <=SDW.R2

Yes

Access violation execute flag off

No

SDW.E = ON

Yes

Access violation attempt to change ring with transfer instruction other than CALL or RETURN

No

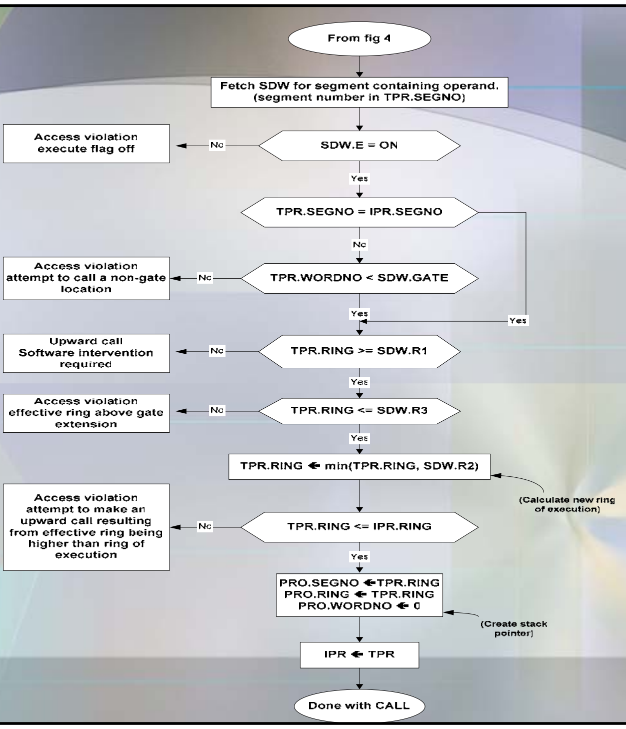
TPR.RING = IPR.RING

Yes

OK to complete instruction

Fig 3 & 4

- Perform the instruction
- Read and write instructions reference their operands
- EAP type instructions load the ring, segno and wordno of PRn to TPR
- No access validation required for EAP type
- For instructions which do not reference their operands, access violation is checked for reloading IPR from TPR
- Call and return can change the ring of execution
- Call can automatically switch to lower number and return to higher number, if other way then results in traps which requires software intervention



From fig 4

Fetch SDW for segment containing operand. (segment number in TPR.SEGNO)

SDW.E = ON

Access violation execute flag off

TPR.SEGNO = IPR.SEGNO

TPR.WORDNO < SDW.GATE

Access violation attempt to call a non-gate location

TPR.RING >= SDW.R1

Upward call Software intervention required

TPR.RING <= SDW.R3

Access violation effective ring above gate extension

TPR.RING ← min(TPR.RING, SDW.R2)

(Calculate new ring of execution)

TPR.RING <= IPR.RING

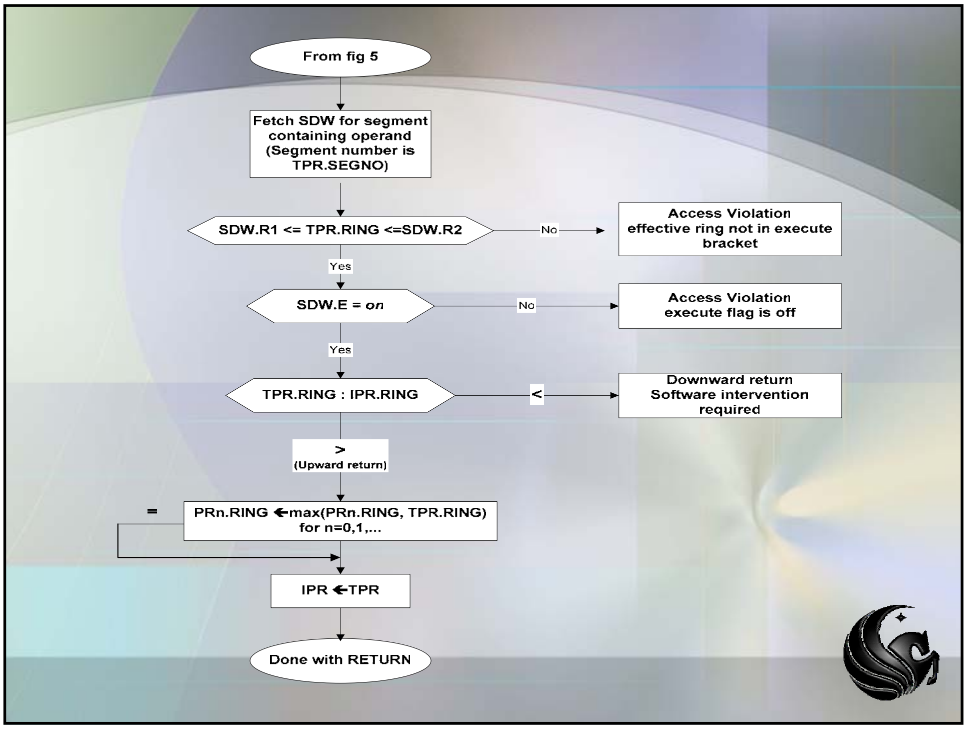
Access violation attempt to make an upward call resulting from effective ring being higher than ring of execution

PRO.SEGNO ←TPR.RING
PRO.RING ← TPR.RING
PRO.WORDNO ← 0

(Create stack pointer)

IPR ← TPR

Done with CALL



From fig 5

Fetch SDW for segment containing operand (Segment number is TPR.SEGNO)

SDW.R1 <= TPR.RING <=SDW.R2

No

Access Violation effective ring not in execute bracket

Yes

SDW.E = *on*

No

Access Violation execute flag is off

Yes

TPR.RING : IPR.RING

<

Downward return Software intervention required

>
(Upward return)

=

PRn.RING ← max(PRn.RING, TPR.RING) for n=0,1,...

IPR ← TPR

Done with RETURN

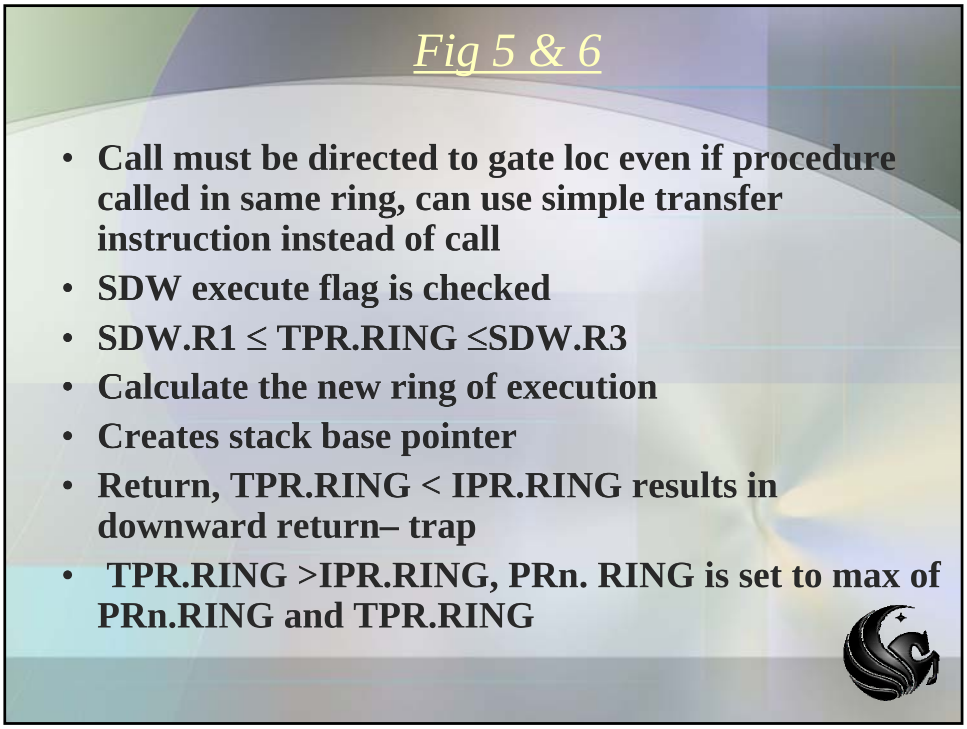
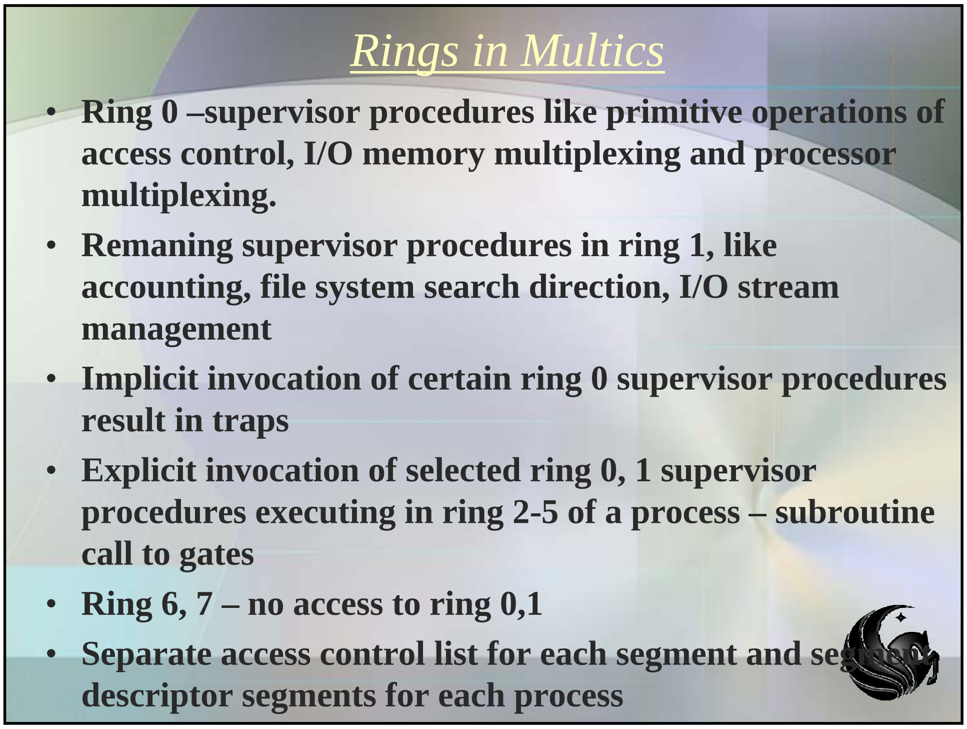


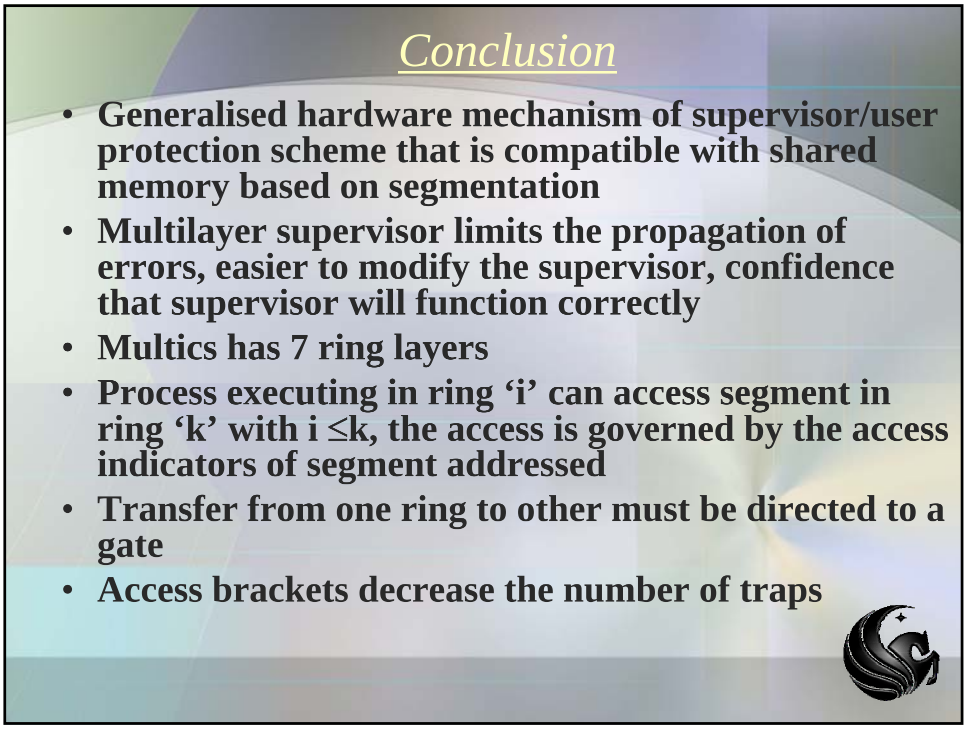
Fig 5 & 6

- **Call must be directed to gate loc even if procedure called in same ring, can use simple transfer instruction instead of call**
- **SDW execute flag is checked**
- **SDW.R1 ≤ TPR.RING ≤SDW.R3**
- **Calculate the new ring of execution**
- **Creates stack base pointer**
- **Return, TPR.RING < IPR.RING results in downward return– trap**
- **TPR.RING >IPR.RING, PRn. RING is set to max of PRn.RING and TPR.RING**



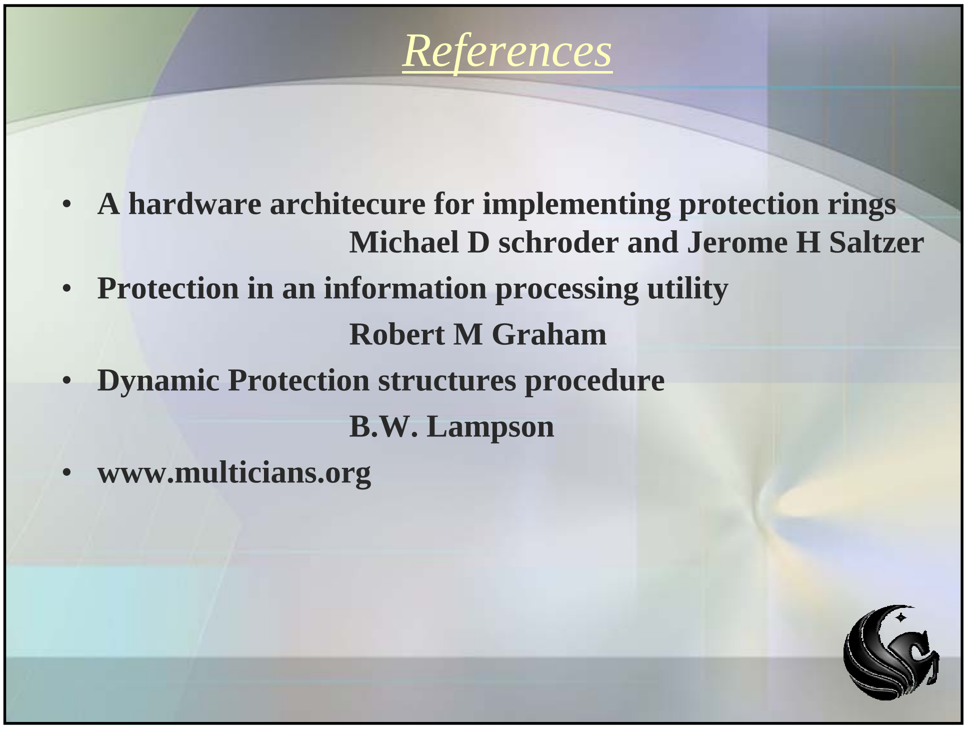
Rings in Multics

- Ring 0 –supervisor procedures like primitive operations of access control, I/O memory multiplexing and processor multiplexing.
- Remaning supervisor procedures in ring 1, like accounting, file system search direction, I/O stream management
- Implicit invocation of certain ring 0 supervisor procedures result in traps
- Explicit invocation of selected ring 0, 1 supervisor procedures executing in ring 2-5 of a process – subroutine call to gates
- Ring 6, 7 – no access to ring 0,1
- Separate access control list for each segment and segment descriptor segments for each process



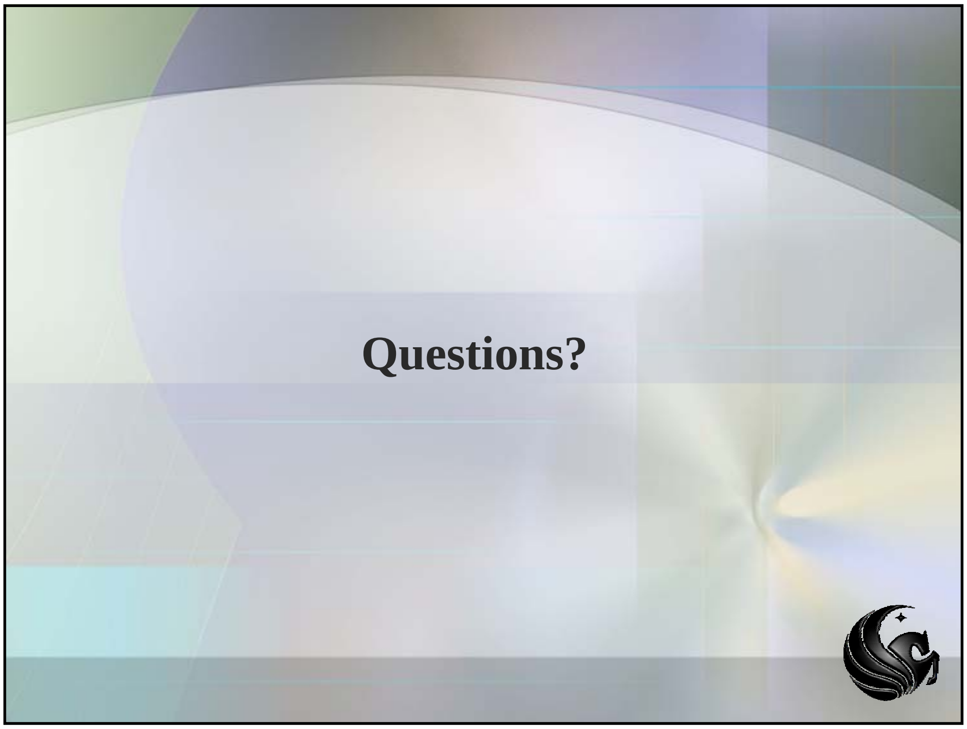
Conclusion

- Generalised hardware mechanism of supervisor/user protection scheme that is compatible with shared memory based on segmentation
- Multilayer supervisor limits the propagation of errors, easier to modify the supervisor, confidence that supervisor will function correctly
- Multics has 7 ring layers
- Process executing in ring ‘i’ can access segment in ring ‘k’ with i ≤k, the access is governed by the access indicators of segment addressed
- Transfer from one ring to other must be directed to a gate
- Access brackets decrease the number of traps



References

- A hardware architecure for implementing protection rings
  Michael D schroder and Jerome H Saltzer
- Protection in an information processing utility
  Robert M Graham
- Dynamic Protection structures procedure
  B.W. Lampson
- www.multicians.org



Questions?