Virtual Machines

Wolfgang Schröder-Preikschat

FRIEDRICH-ALEXANDER
UNIVERSITÄT
ERLANGEN-NÜRNBERG

TECHNISCHE FAKULTÄT

TRR 89

ESI Embedded Systems Institute

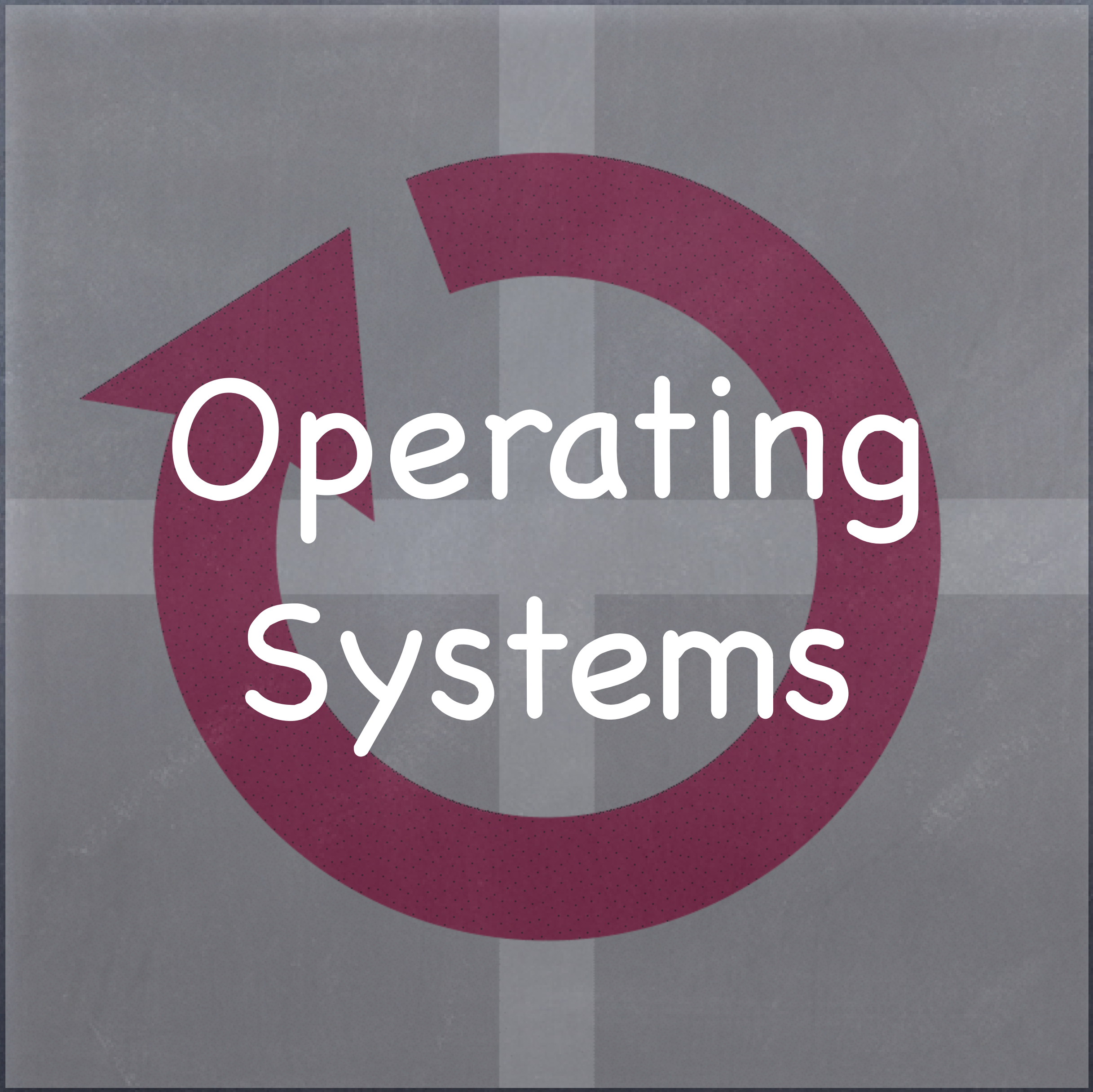
Background

multi/many-core systems

(since 2008)

uni-processor systems

(1981-1986)



Operating Systems

embedded & real-time systems

(since 1995)

multi-processor systems

(1986-1995)

Wat is´n dit?*

Portfolio lacks virtualization systems

*Berlin slang: „What is this?"

A question of semantics

Def.: virtual machine

- „an efficient, isolated duplicate of a real machine“ (Popek & Goldberg)
- a „hypothetical computer [...] whose machine language“ complies with the user‘s demands
  - „rather than thinking in terms of translation or interpretation“ (Tanenbaum)

Different views and concepts

bottom-up

(Popek & Goldberg)

top-down

(Tanenbaum)

Operating Systems

- implement virtual machines (AST)
- have 2/3 virtual machine properties (P&G)
  - efficiency property
  - resource control property
  - ~~equivalence property~~

Thus, an operating-system adept usually should know the ropes...

Outline

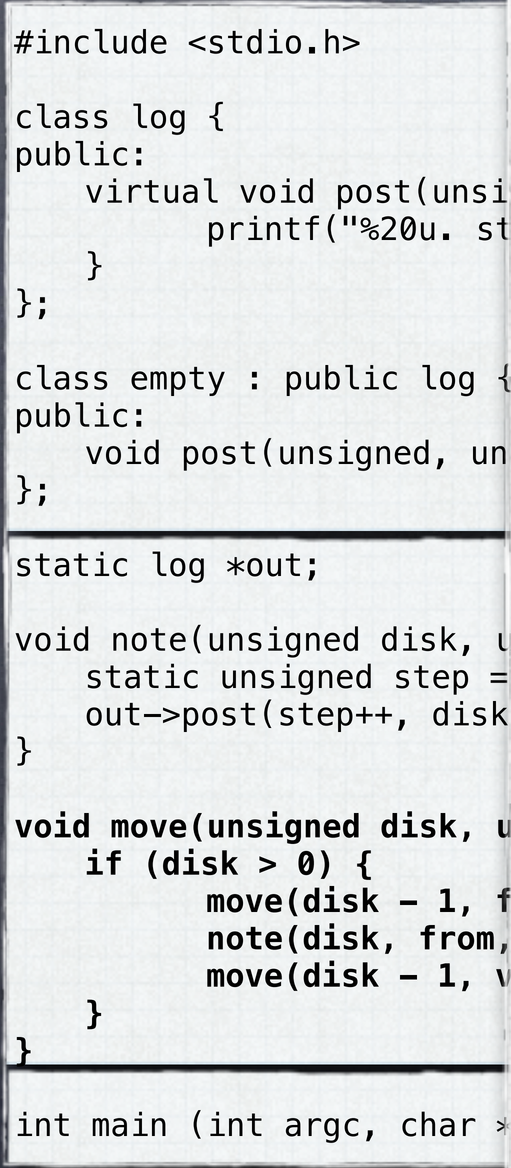
✔prologue

- historically distinct landscape
  - top-down view (AST)
  - bottom-up view (P&G)
- present-day problems
  - contemporary issues: ideas of one‘s own
- epilogue

„top-down“ de-/virtualization

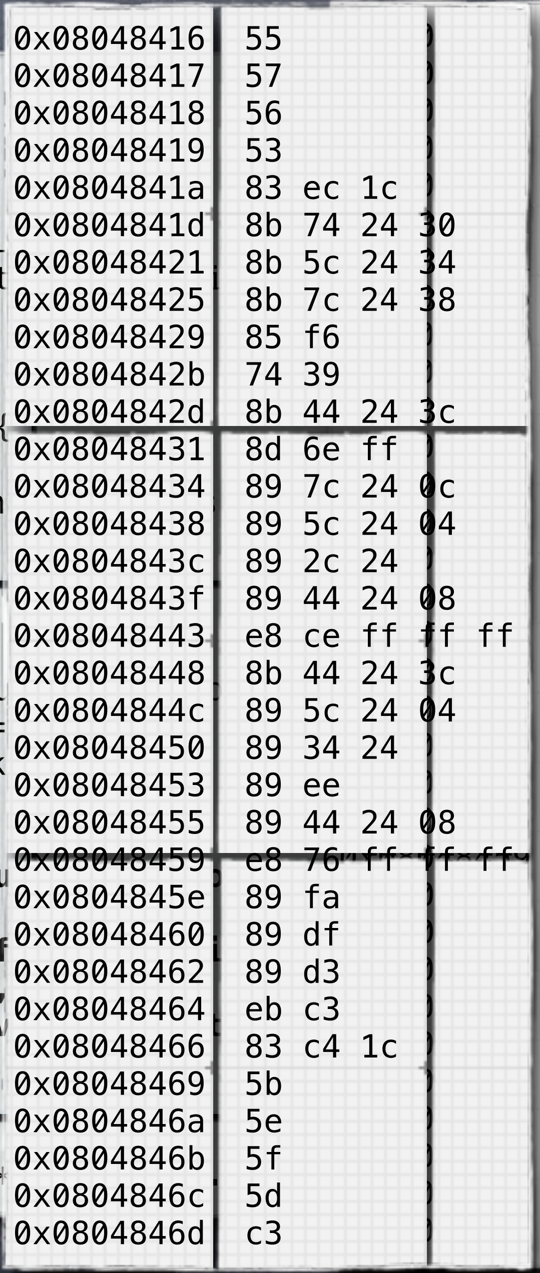
Devirtualization

problem-oriented language



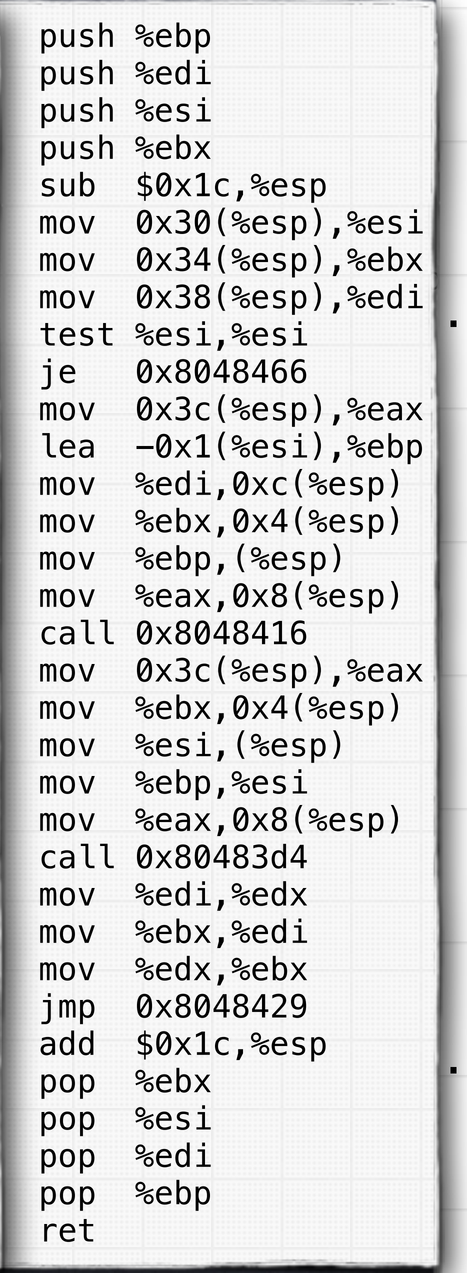
```
#include <stdio.h>

class log {
public:
    virtual void post(unsi
           printf("%20u. st
    }
};

class empty : public log {
public:
    void post(unsigned, un
};

static log *out;

void note(unsigned disk, u
    static unsigned step =
    out->post(step++, disk
}

void move(unsigned disk, u
    if (disk > 0) {
           move(disk - 1, f
           note(disk, from,
           move(disk - 1, v
    }
}

int main (int argc, char *
```

executable

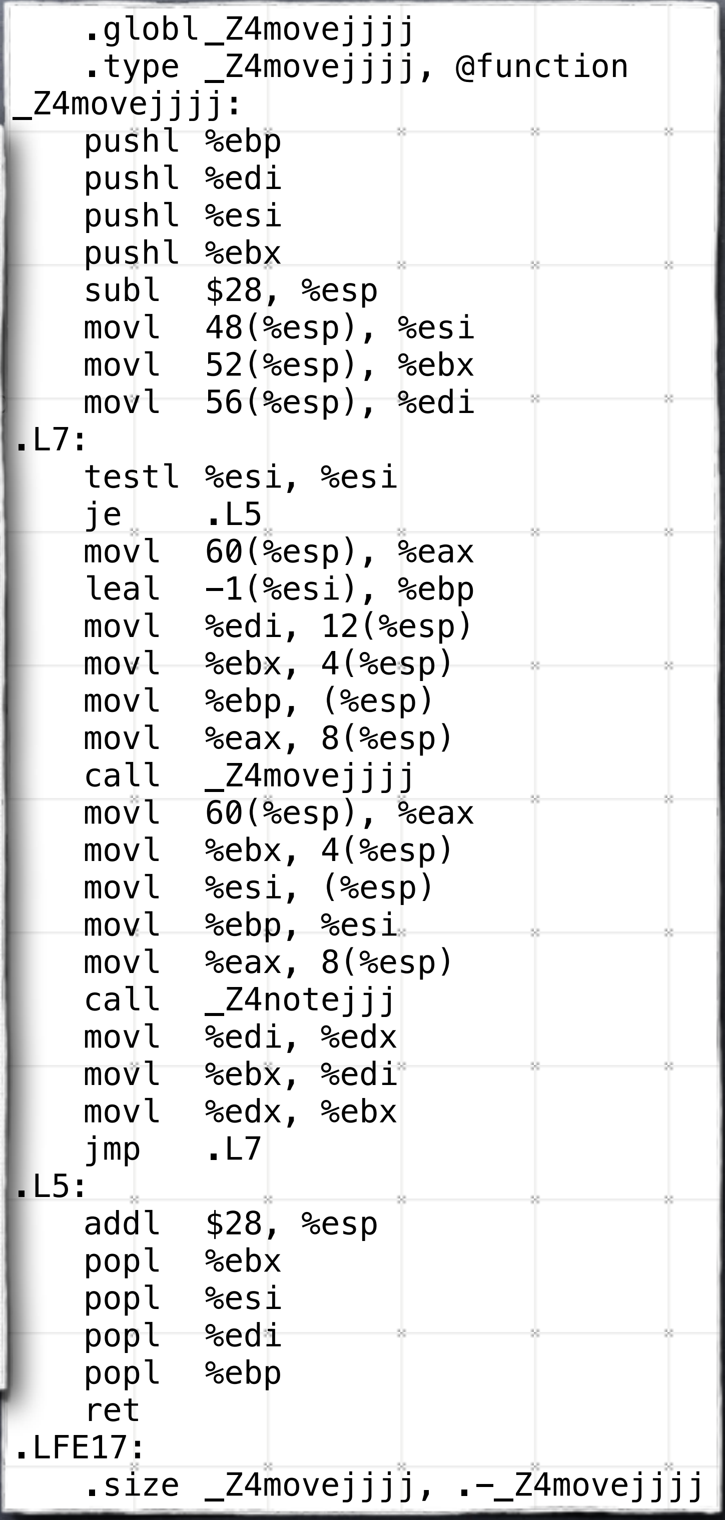


```
0x08048416  55
0x08048417  57
0x08048418  56
0x08048419  53
0x0804841a  83 ec 1c
0x0804841d  8b 74 24 30
0x08048421  8b 5c 24 34
0x08048425  8b 7c 24 38
0x08048429  85 f6
0x0804842b  74 39
0x0804842d  8b 44 24 3c
0x08048431  8d 6e ff
0x08048434  89 7c 24 0c
0x08048438  89 5c 24 04
0x0804843c  89 2c 24
0x0804843f  89 44 24 08
0x08048443  e8 ce ff ff ff
0x08048448  8b 44 24 3c
0x0804844c  89 5c 24 04
0x08048450  89 34 24
0x08048453  89 ee
0x08048455  89 44 24 08
0x08048459  e8 76 ff ff ff
0x0804845e  89 fa
0x08048460  89 df
0x08048462  89 d3
0x08048464  eb c3
0x08048466  83 c4 1c
0x08048469  5b
0x0804846a  5e
0x0804846b  5f
0x0804846c  5d
0x0804846d  c3
```

symbolic
machine code



```
push %ebp
push %edi
push %esi
push %ebx
sub  $0x1c,%esp
mov  0x30(%esp),%esi
mov  0x34(%esp),%ebx
mov  0x38(%esp),%edi
test %esi,%esi
je   0x8048466
mov  0x3c(%esp),%eax
lea  -0x1(%esi),%ebp
mov  %edi,0xc(%esp)
mov  %ebx,0x4(%esp)
mov  %ebp,(%esp)
mov  %eax,0x8(%esp)
call 0x8048416
mov  0x3c(%esp),%eax
mov  %ebx,0x4(%esp)
mov  %esi,(%esp)
mov  %ebp,%esi
mov  %eax,0x8(%esp)
call 0x80483d4
mov  %edi,%edx
mov  %ebx,%edi
mov  %edx,%ebx
jmp  0x8048429
add  $0x1c,%esp
pop  %ebx
pop  %esi
pop  %edi
pop  %ebp
ret
```



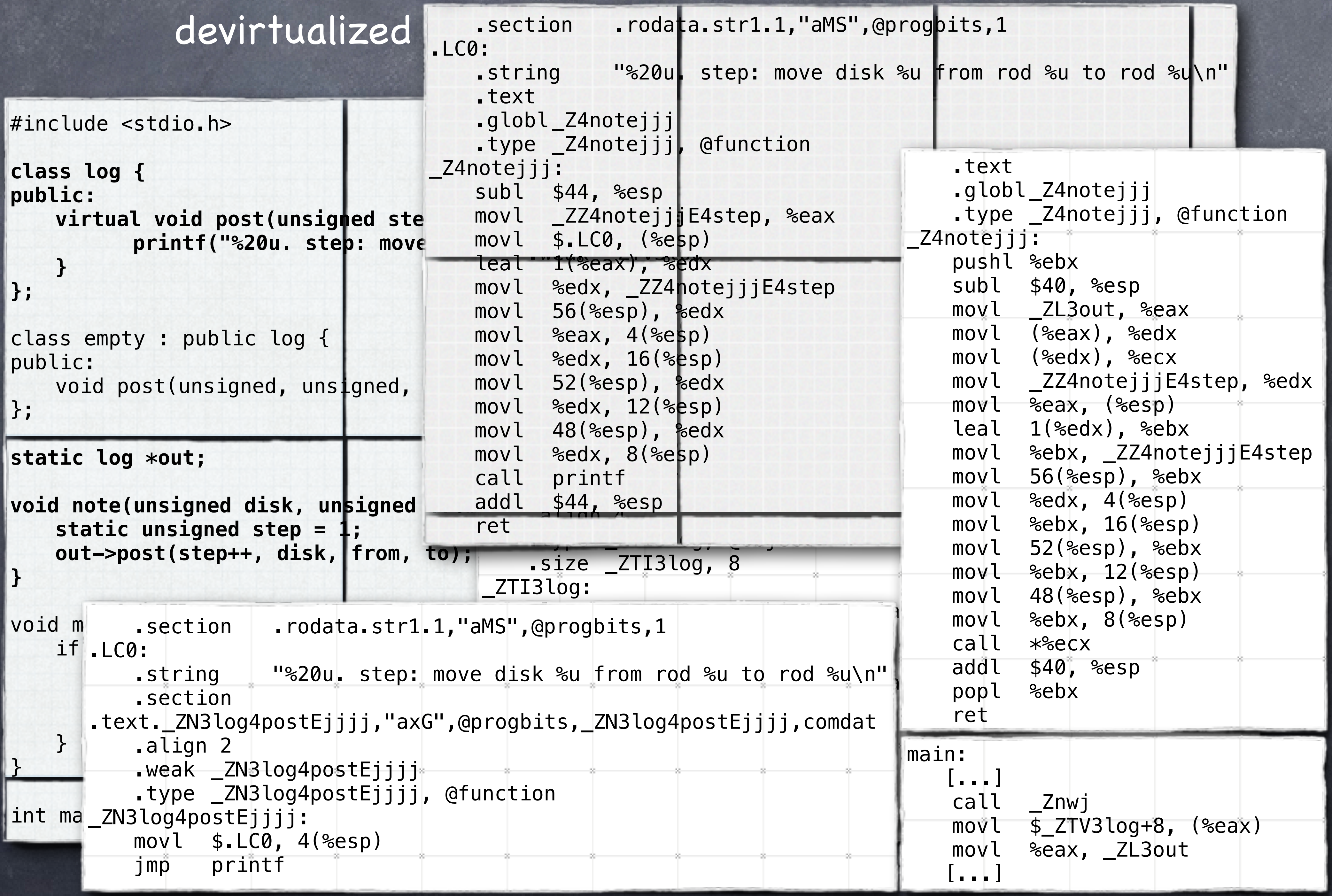
```
    .globl_Z4movejjjj
    .type _Z4movejjjj, @function
_Z4movejjjj:
    pushl %ebp
    pushl %edi
    pushl %esi
    pushl %ebx
    subl  $28, %esp
    movl  48(%esp), %esi
    movl  52(%esp), %ebx
    movl  56(%esp), %edi
.L7:
    testl %esi, %esi
    je    .L5
    movl  60(%esp), %eax
    leal  -1(%esi), %ebp
    movl  %edi, 12(%esp)
    movl  %ebx, 4(%esp)
    movl  %ebp, (%esp)
    movl  %eax, 8(%esp)
    call  _Z4movejjjj
    movl  60(%esp), %eax
    movl  %ebx, 4(%esp)
    movl  %esi, (%esp)
    movl  %ebp, %esi
    movl  %eax, 8(%esp)
    call  _Z4notejjj
    movl  %edi, %edx
    movl  %ebx, %edi
    movl  %edx, %ebx
    jmp   .L7
.L5:
    addl  $28, %esp
    popl  %ebx
    popl  %esi
    popl  %edi
    popl  %ebp
    ret
.LFE17:
    .size _Z4movejjjj, .-_Z4movejjjj
```

assembly language

Devirtualization

`g++ -Os -m32 -static -fomit-frame-pointer -Dvirtual="" -S`

devirtualized



```
#include <stdio.h>

class log {
public:
    virtual void post(unsigned ste
          printf("%20u. step: move
    }
};

class empty : public log {
public:
    void post(unsigned, unsigned,
};

static log *out;

void note(unsigned disk, unsigned
    static unsigned step = 1;
    out->post(step++, disk, from, to);
}

void m
    if

    }
}

int ma
```

```
    .section    .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string     "%20u. step: move disk %u from rod %u to rod %u\n"
    .text
    .globl _Z4notejjj
    .type _Z4notejjj, @function
_Z4notejjj:
    subl   $44, %esp
    movl   _ZZ4notejjjE4step, %eax
    movl   $.LC0, (%esp)
    leal   1(%eax), %edx
    movl   %edx, _ZZ4notejjjE4step
    movl   56(%esp), %edx
    movl   %eax, 4(%esp)
    movl   %edx, 16(%esp)
    movl   52(%esp), %edx
    movl   %edx, 12(%esp)
    movl   48(%esp), %edx
    movl   %edx, 8(%esp)
    call   printf
    addl   $44, %esp
    ret
```

```
        .size  _ZTI3log, 8
_ZTI3log:
```

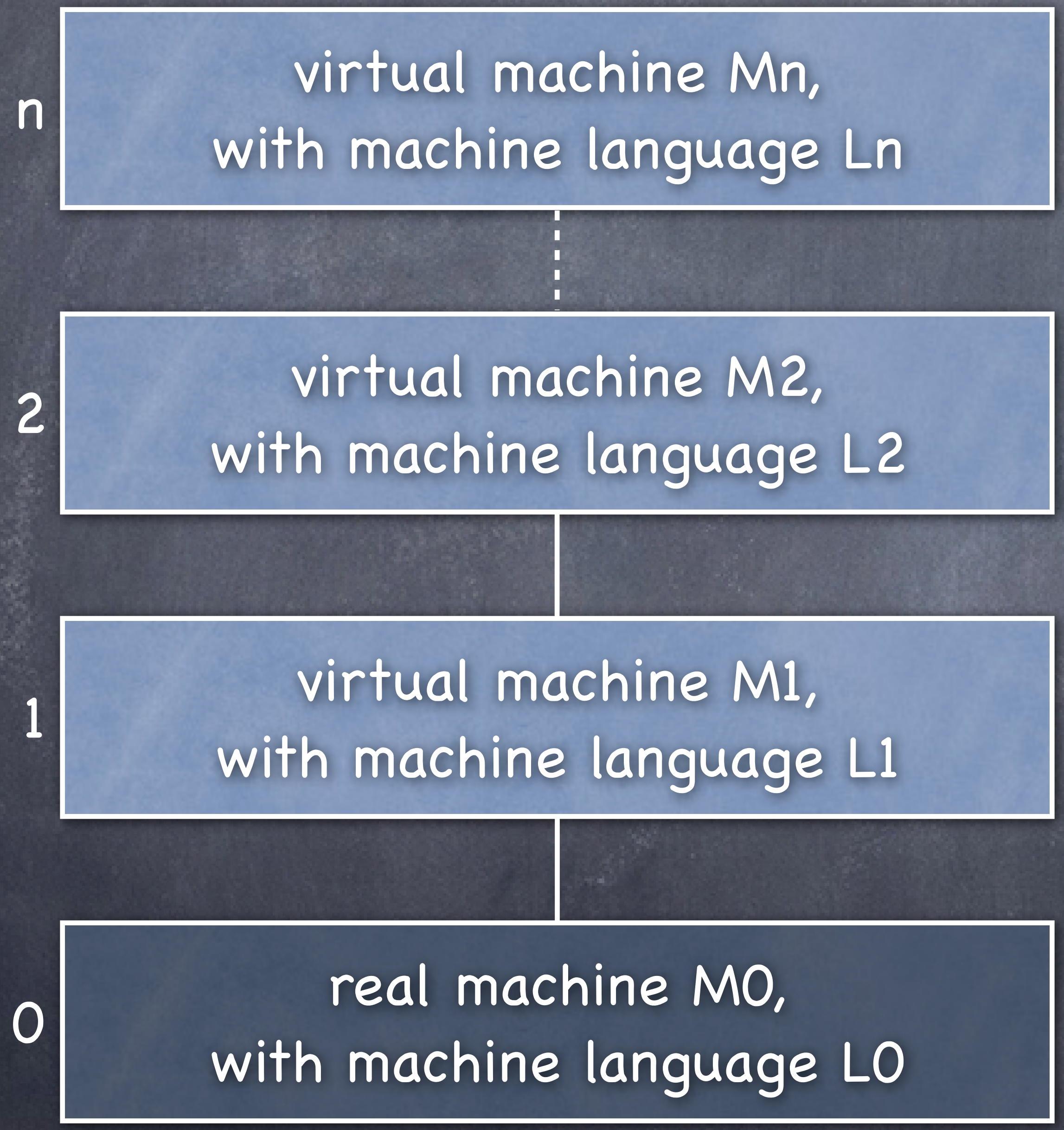
```
    .section    .rodata.str1.1,"aMS",@progbits,1
.LC0:
    .string     "%20u. step: move disk %u from rod %u to rod %u\n"
    .section
.text._ZN3log4postEjjjj,"axG",@progbits,_ZN3log4postEjjjj,comdat
    .align 2
    .weak _ZN3log4postEjjjj
    .type _ZN3log4postEjjjj, @function
_ZN3log4postEjjjj:
    movl   $.LC0, 4(%esp)
    jmp    printf
```

```
    .text
    .globl _Z4notejjj
    .type _Z4notejjj, @function
_Z4notejjj:
    pushl %ebx
    subl  $40, %esp
    movl  _ZL3out, %eax
    movl  (%eax), %edx
    movl  (%edx), %ecx
    movl  _ZZ4notejjjE4step, %edx
    movl  %eax, (%esp)
    leal  1(%edx), %ebx
    movl  %ebx, _ZZ4notejjjE4step
    movl  56(%esp), %ebx
    movl  %edx, 4(%esp)
    movl  %ebx, 16(%esp)
    movl  52(%esp), %ebx
    movl  %ebx, 12(%esp)
    movl  48(%esp), %ebx
    movl  %ebx, 8(%esp)
    call  *%ecx
    addl  $40, %esp
    popl  %ebx
    ret
```

```
main:
   [...]
    call  _Znwj
    movl  $_ZTV3log+8, (%eax)
    movl  %eax, _ZL3out
   [...]
```

virtualized

Languages, levels and virtual machines



n virtual machine Mn, with machine language Ln

programs in Ln are either interpreted by a lower machine or translated to a language of a lower machine

2 virtual machine M2, with machine language L2

programs in L2 are either interpreted by M1 or M0 or translated to L1 or L0

1 virtual machine M1, with machine language L1

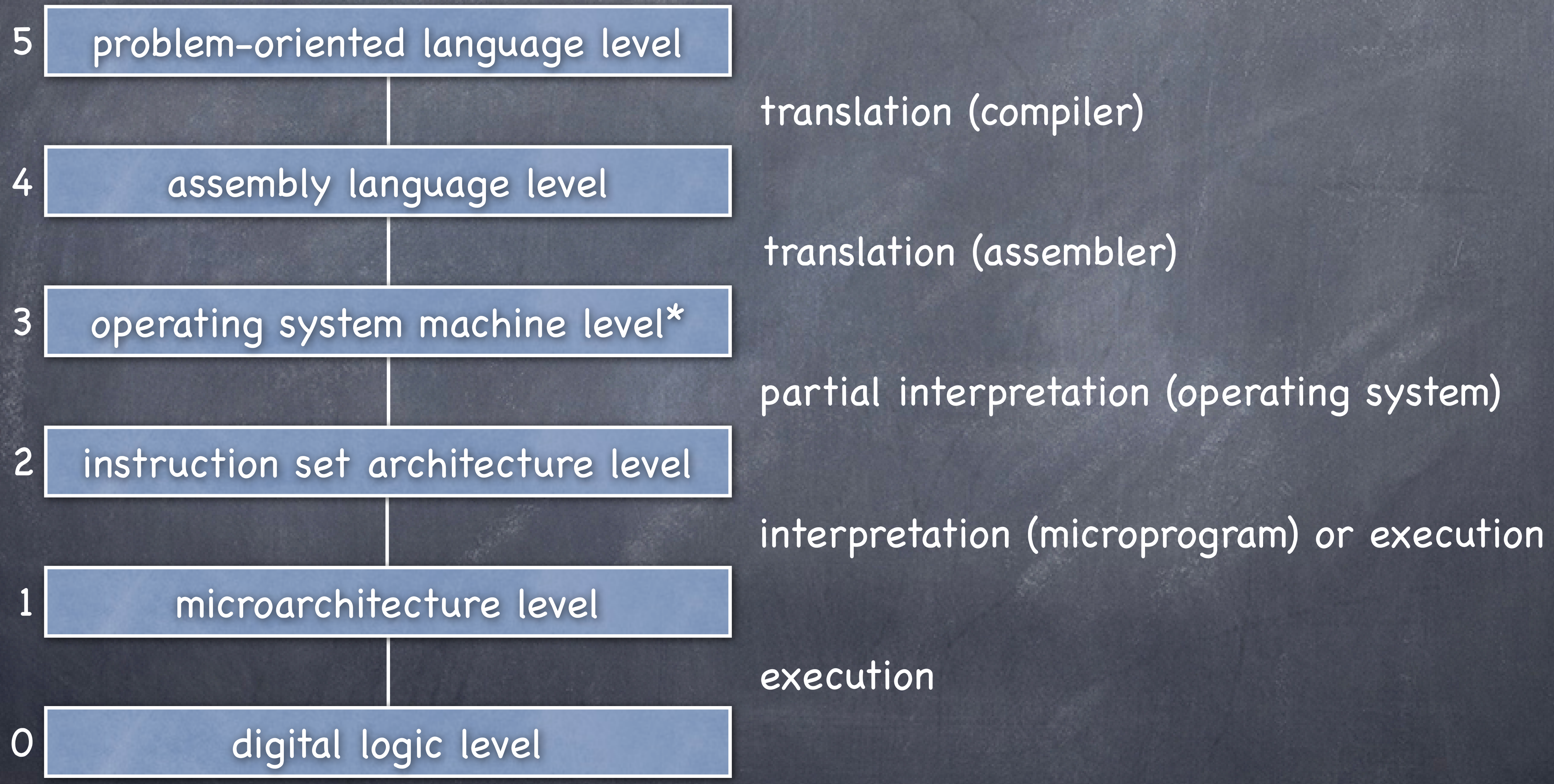
programs in L1 are either interpreted by M0 or translated to L0

0 real machine M0, with machine language L0

programs in L0 can be executed by M0 (electronic circuits)

Tanenbaum, Structured computer organization

Contemporary multilevel machine



5 problem-oriented language level

translation (compiler)

4 assembly language level

translation (assembler)

3 operating system machine level*

partial interpretation (operating system)

2 instruction set architecture level

interpretation (microprogram) or execution

1 microarchitecture level

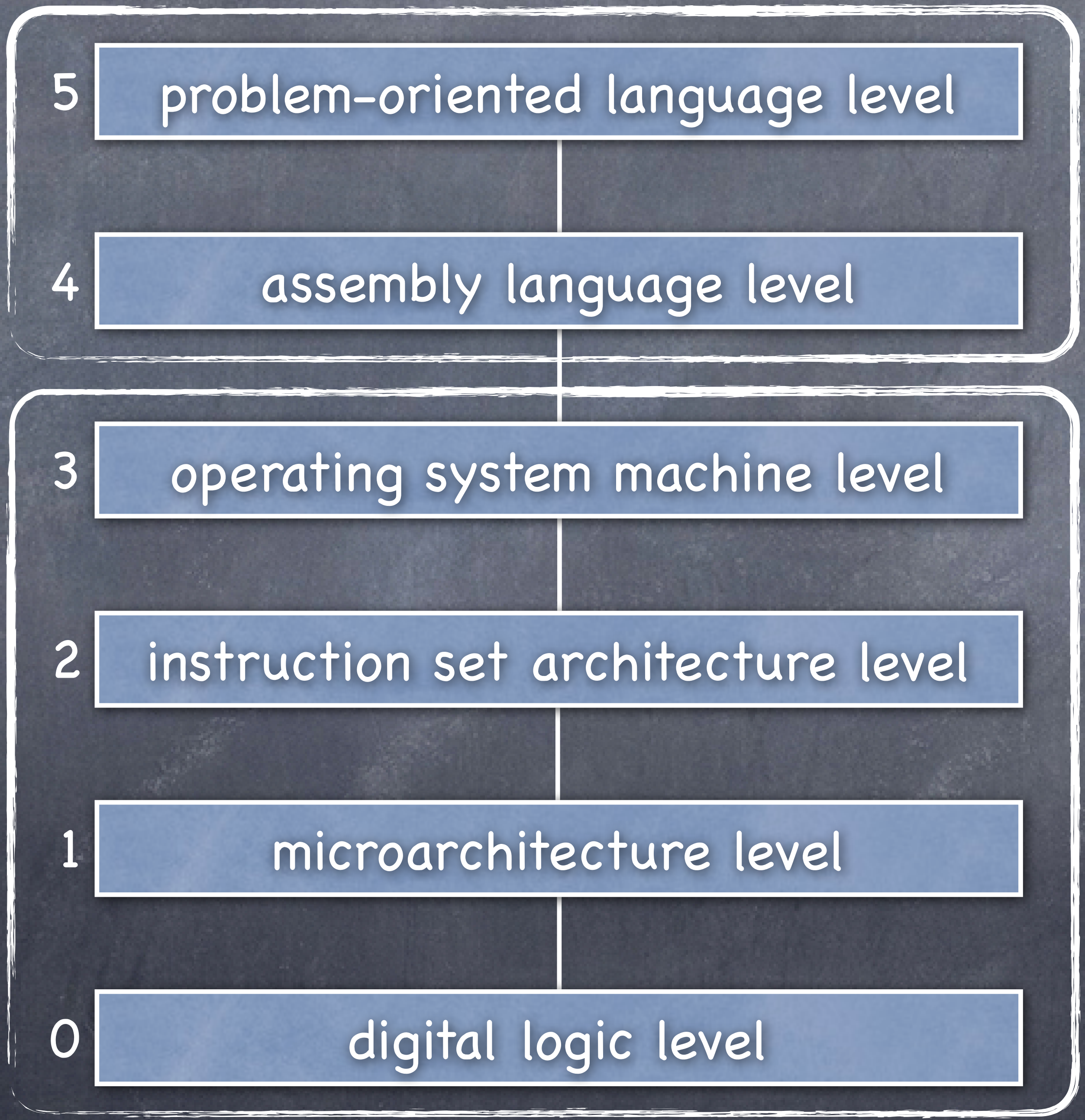
execution

0 digital logic level

*machine program level

Tanenbaum, Structured computer organization

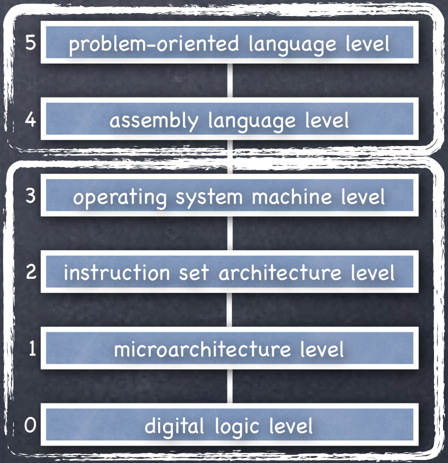
Line of demarcation


5
problem-oriented language level
4
assembly language level
3
operating system machine level
2
instruction set architecture level
1
microarchitecture level
0
digital logic level

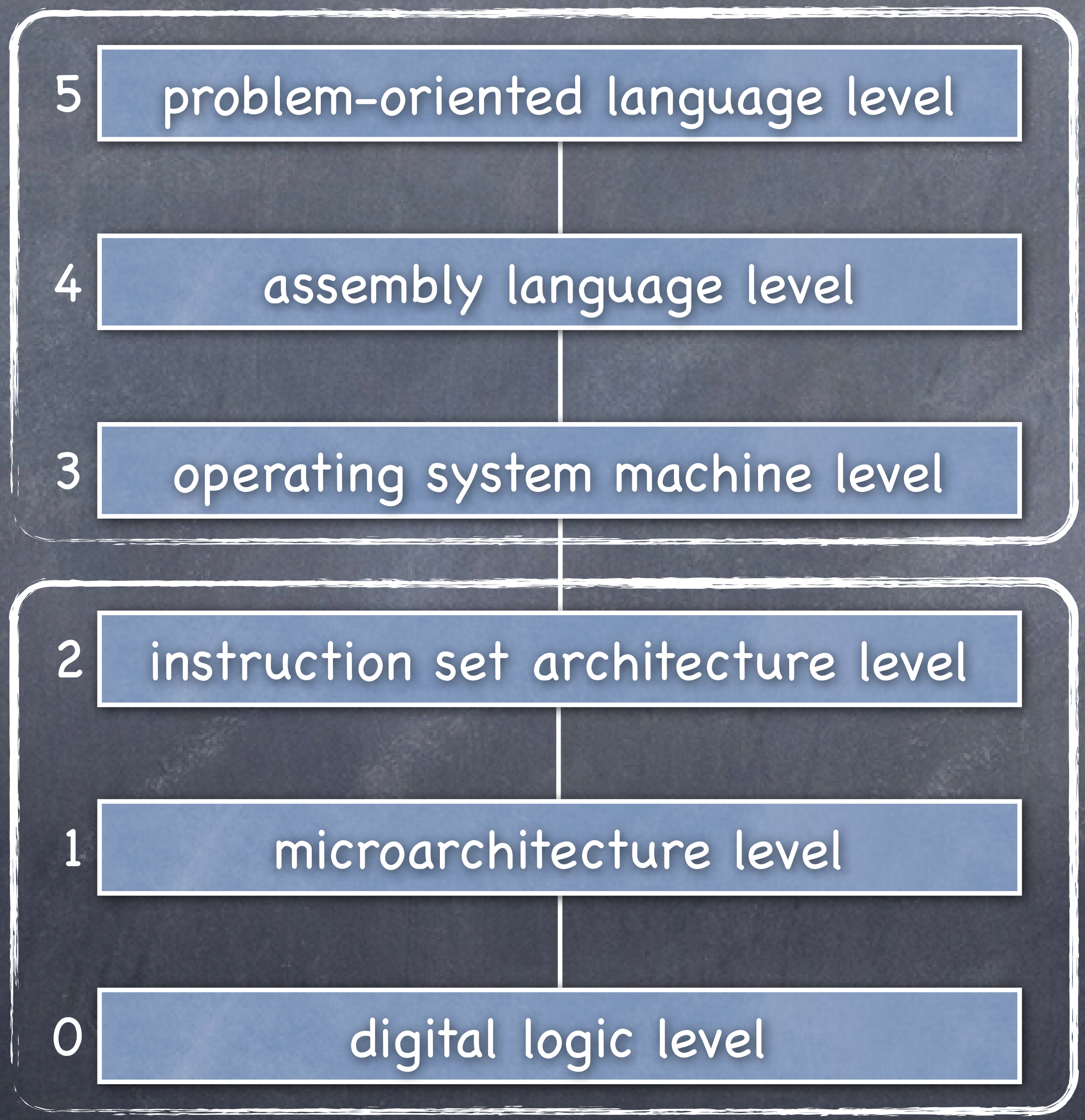
Line of demarcation

Fundamental break between levels 3 and 4:

- art of computer programming
  - system (≤3) versus application (≥4)
- method by which higher levels are supported
  - interpretation (≤3) versus translation (≥4)
- nature of language provided
  - numeric (≤3) versus symbolic (≥4)


5 problem-oriented language level
4 assembly language level
3 operating system machine level
2 instruction set architecture level
1 microarchitecture level
0 digital logic level

Partial interpretation



5 problem-oriented language level

4 assembly language level

3 operating system machine level

2 instruction set architecture level

1 microarchitecture level

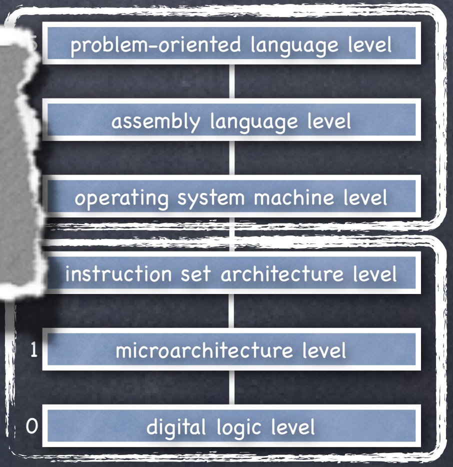
0 digital logic level

Partial interpretation

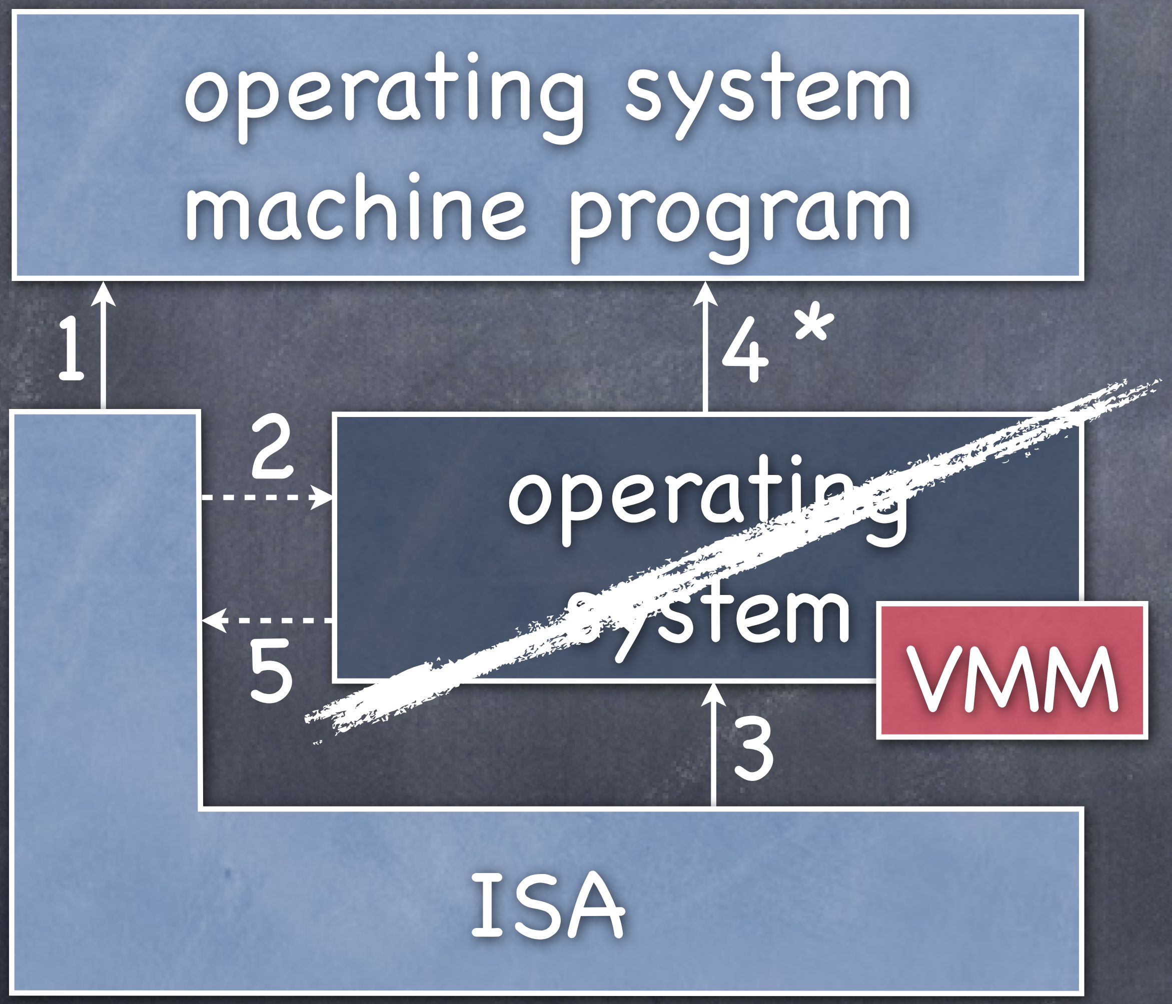
- real machine „central processing unit" (CPU)
  - interpretes CPU instructions
  - detects and releases exceptions

- abstract processor „operating system" (OS)
  - accepts and handles exceptions
  - interpretes system calls

- virtual machine monitor (VMM) like OS
  - interpretes „sensible instructions"


problem-oriented language level
assembly language level
operating system machine level
instruction set architecture level
1
microarchitecture level
0
digital logic level

Vertical cooperation



*partial interpretation

while powered on:

1. fetch-execute (CPU)
2. exception
3. fetch-execute (CPU)
4. fetch-execute (OS)
5. return from exception

exceptional case

Operating system machine program

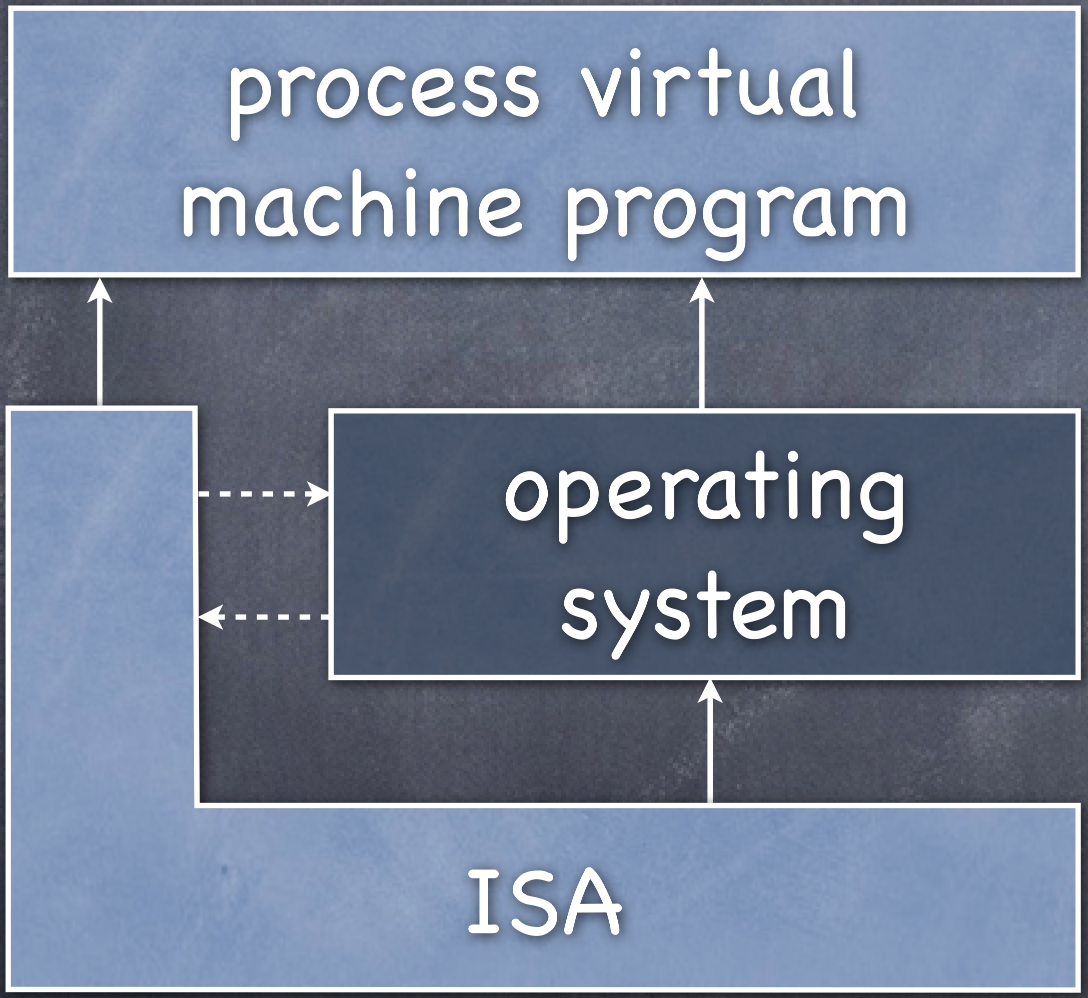
a.k.a. machine program

- executable (e.g. a.out or .exe)
- to be precise: any executable that logically uses* at least one operating-system function
- in particular:
  - a process virtual machine program

* Parnas, Some hypothesis about the „uses" hierarchy for operating systems

Process virtual machine

JNode Pure Java OS HAIKU JamaicaVM ...



process virtual machine program

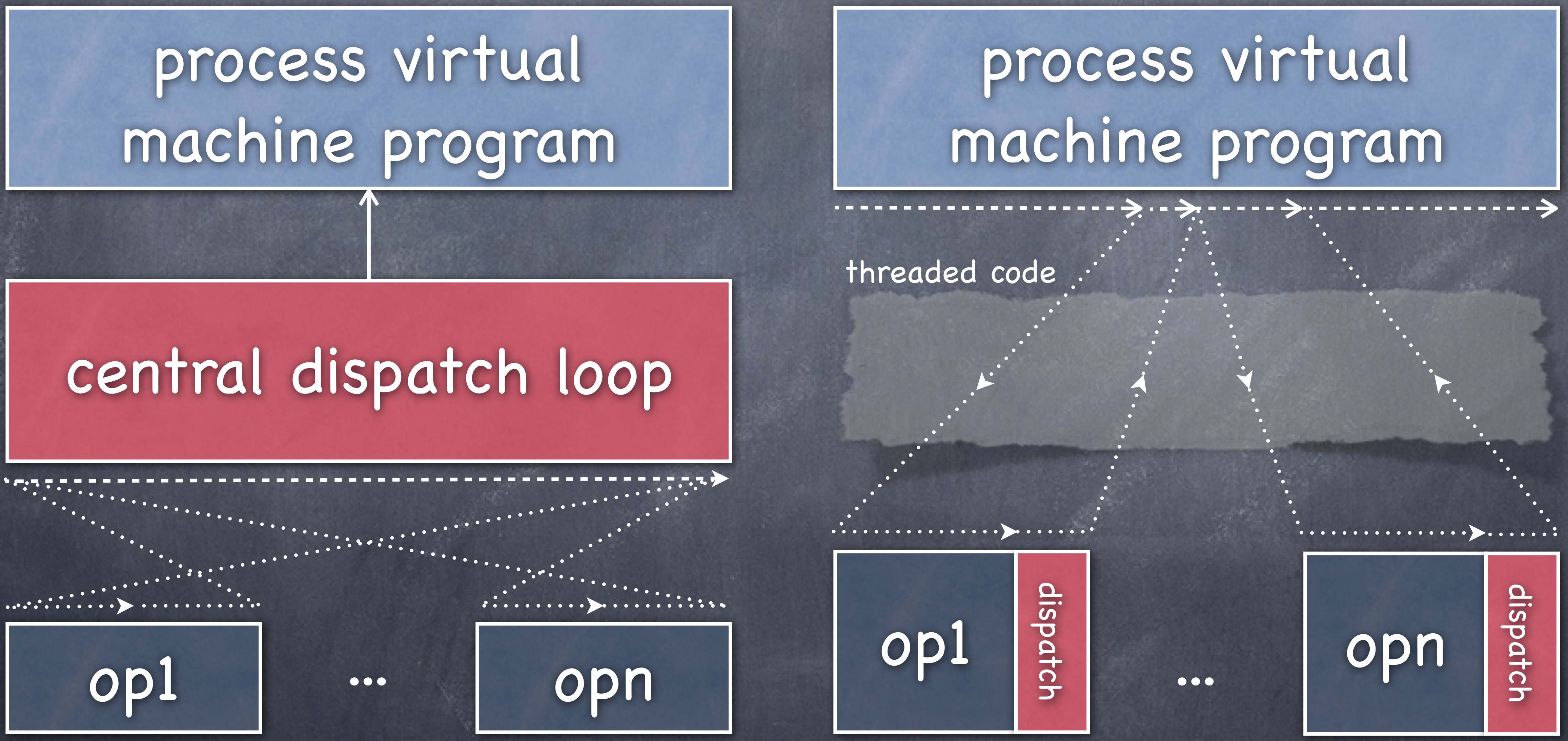
operating system

ISA

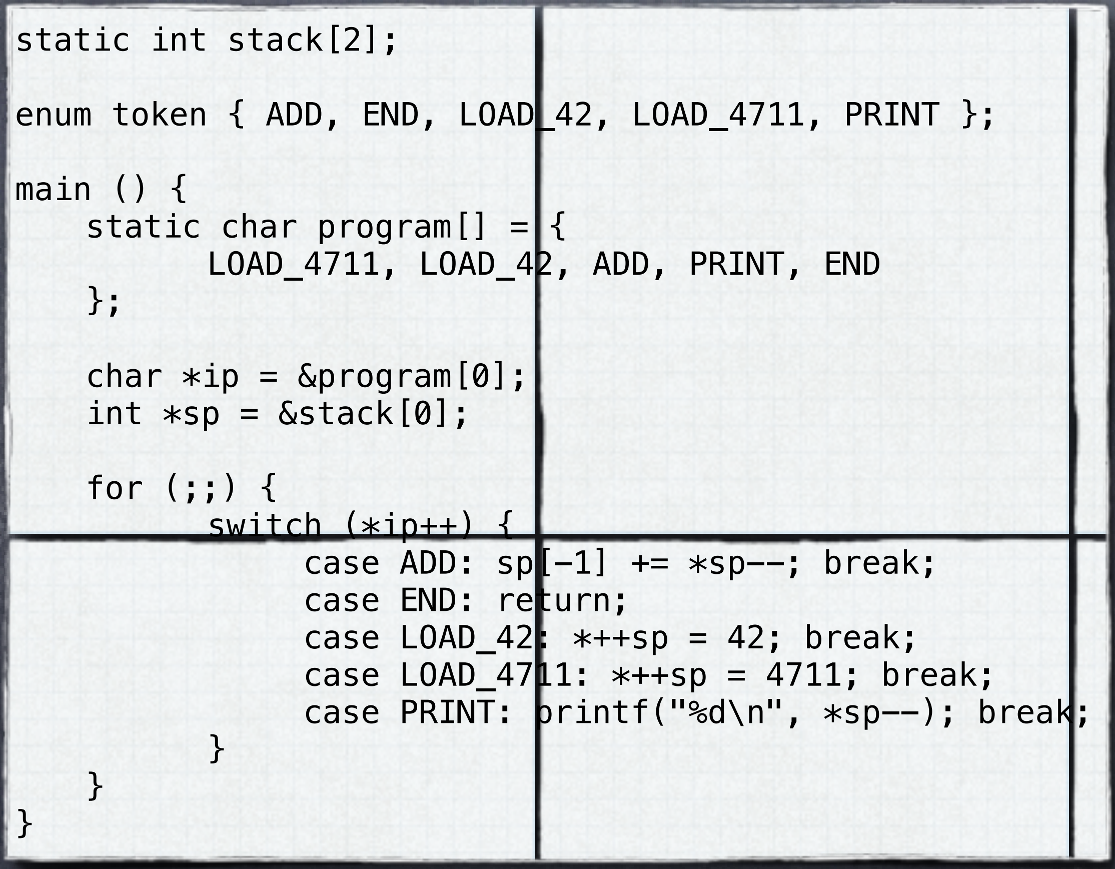
- multiprogramming
- interpretation (CSIM)
- binary translation
- high-level language VM

managed code

Complete software interpreter machine

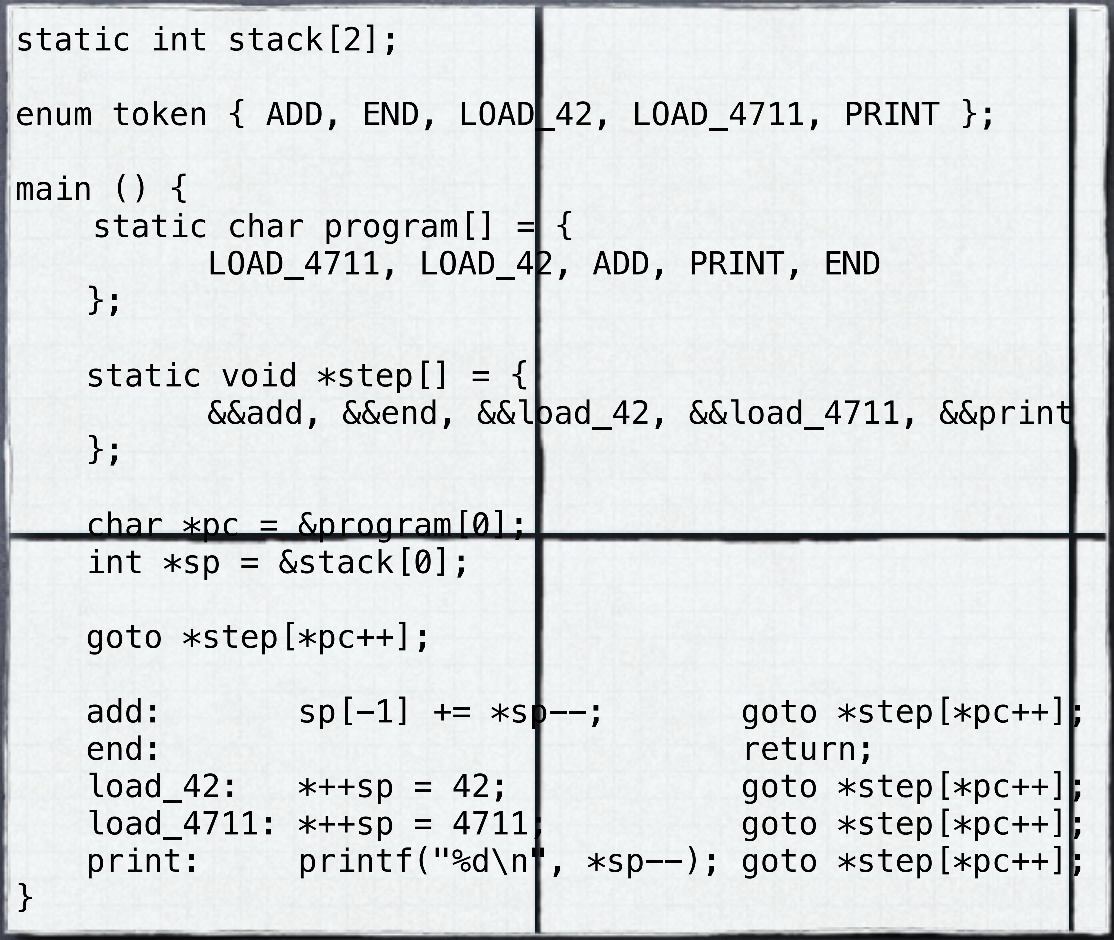


Byte code interpreter



```
static int stack[2];

enum token { ADD, END, LOAD_42, LOAD_4711, PRINT };

main () {
    static char program[] = {
          LOAD_4711, LOAD_42, ADD, PRINT, END
    };

    char *ip = &program[0];
    int *sp = &stack[0];

    for (;;) {
          switch (*ip++) {
               case ADD: sp[-1] += *sp--; break;
               case END: return;
               case LOAD_42: *++sp = 42; break;
               case LOAD_4711: *++sp = 4711; break;
               case PRINT: printf("%d\n", *sp--); break;
          }
    }
}
```

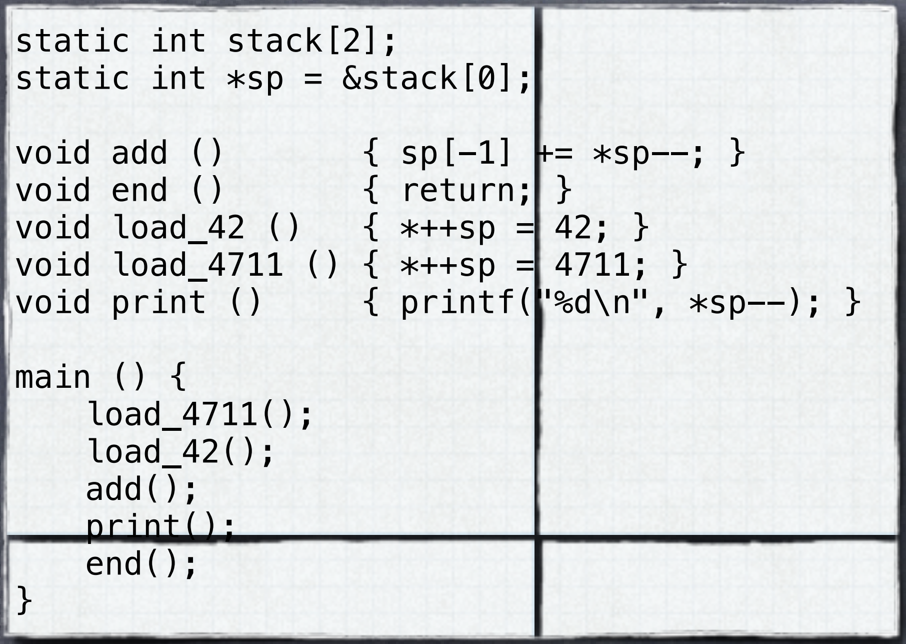
„central dispatch loop“
switch-threaded code (STC)



```
static int stack[2];

enum token { ADD, END, LOAD_42, LOAD_4711, PRINT };

main () {
    static char program[] = {
          LOAD_4711, LOAD_42, ADD, PRINT, END
    };

    static void *step[] = {
          &&add, &&end, &&load_42, &&load_4711, &&print
    };

    char *pc = &program[0];
    int *sp = &stack[0];

    goto *step[*pc++];

    add:       sp[-1] += *sp--;       goto *step[*pc++];
    end:                              return;
    load_42:   *++sp = 42;            goto *step[*pc++];
    load_4711: *++sp = 4711;          goto *step[*pc++];
    print:     printf("%d\n", *sp--); goto *step[*pc++];
}
```

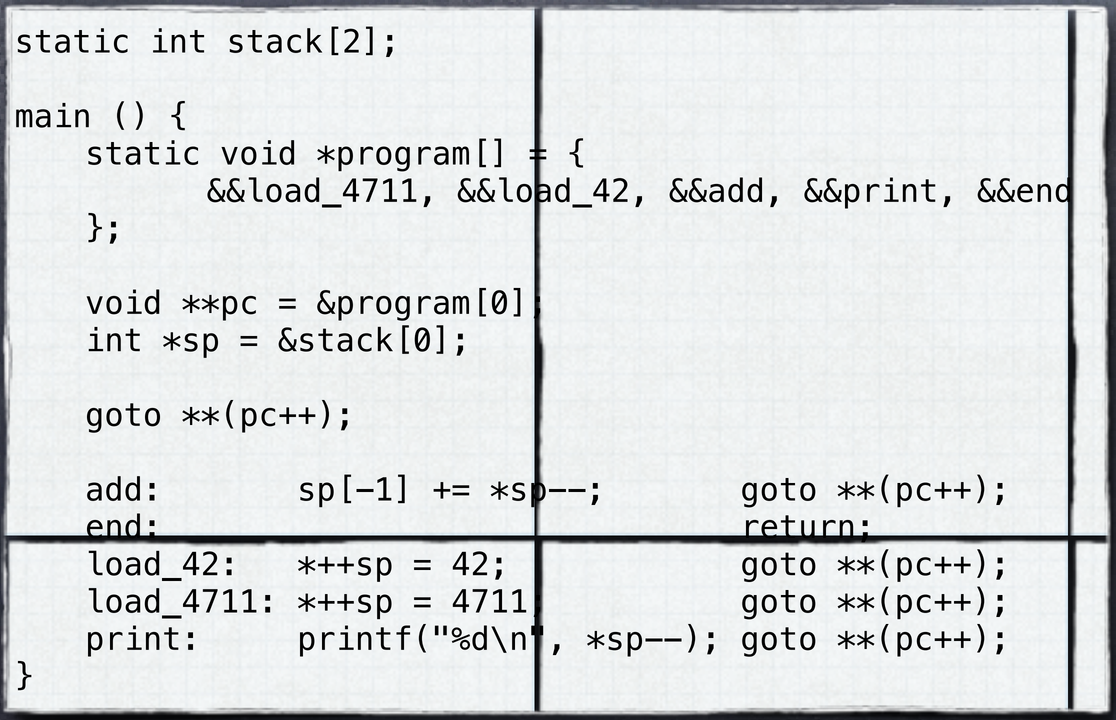
token-threaded code (TTC)

Threaded code (cont.)



```
static int stack[2];
static int *sp = &stack[0];

void add ()       { sp[-1] += *sp--; }
void end ()       { return; }
void load_42 ()   { *++sp = 42; }
void load_4711 () { *++sp = 4711; }
void print ()     { printf("%d\n", *sp--); }

main () {
    load_4711();
    load_42();
    add();
    print();
    end();
}
```

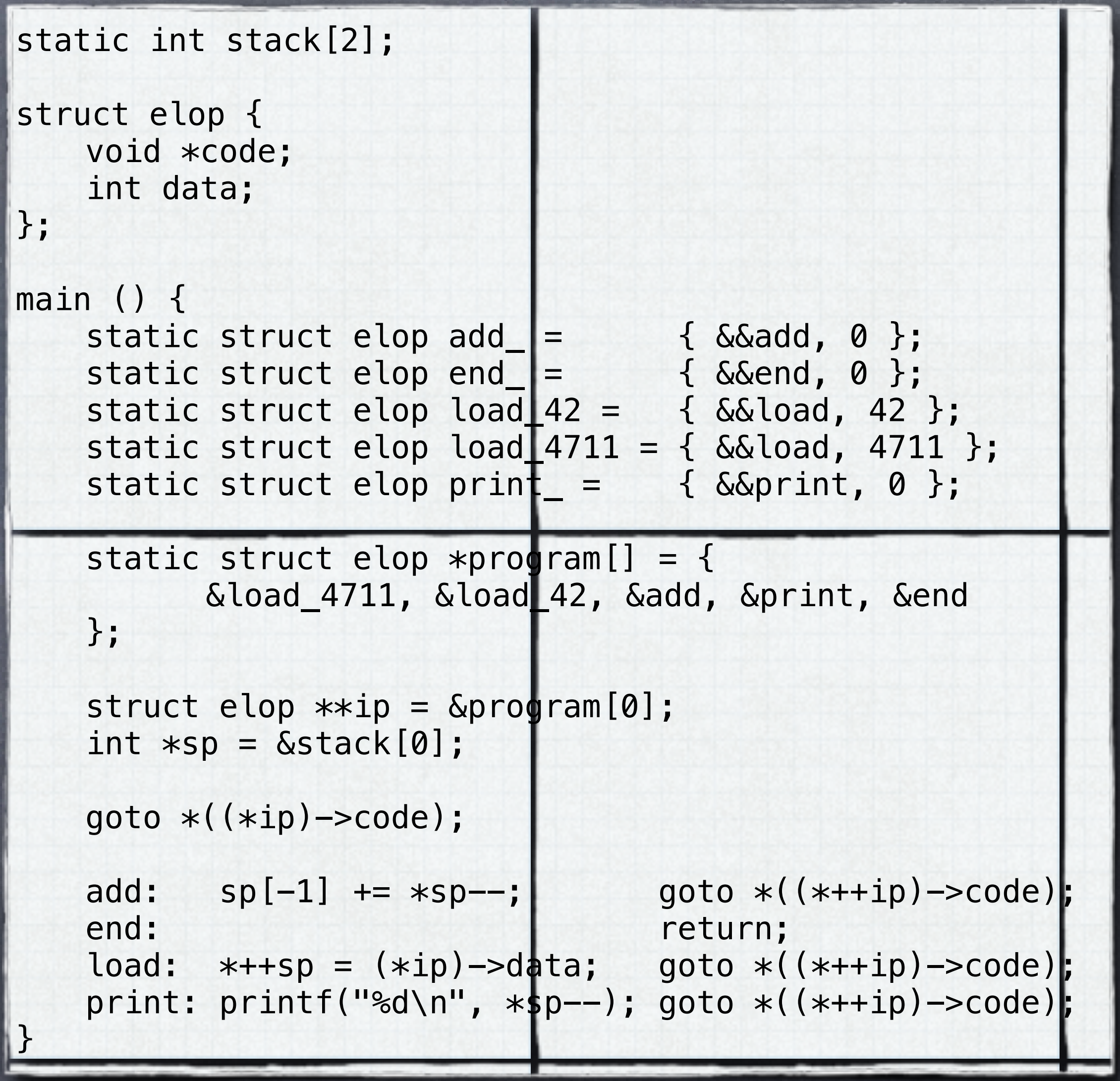
procedure threaded code (PTC)



```
static int stack[2];

main () {
    static void *program[] = {
          &&load_4711, &&load_42, &&add, &&print, &&end
    };

    void **pc = &program[0];
    int *sp = &stack[0];

    goto **(pc++);

    add:       sp[-1] += *sp--;       goto **(pc++);
    end:                              return;
    load_42:   *++sp = 42;            goto **(pc++);
    load_4711: *++sp = 4711;          goto **(pc++);
    print:     printf("%d\n", *sp--); goto **(pc++);
}
```

direct threaded code (DTC)

indirect threaded code (ITC)



```
static int stack[2];

struct elop {
    void *code;
    int data;
};

main () {
    static struct elop add_ =      { &&add, 0 };
    static struct elop end_ =      { &&end, 0 };
    static struct elop load_42 =   { &&load, 42 };
    static struct elop load_4711 = { &&load, 4711 };
    static struct elop print_ =    { &&print, 0 };

    static struct elop *program[] = {
          &load_4711, &load_42, &add, &print, &end
    };

    struct elop **ip = &program[0];
    int *sp = &stack[0];

    goto *((*ip)->code);

    add:   sp[-1] += *sp--;       goto *((*++ip)->code);
    end:                          return;
    load:  *++sp = (*ip)->data;   goto *((*++ip)->code);
    print: printf("%d\n", *sp--); goto *((*++ip)->code);
}
```

Comparison of CSIM variants

PTC

```
ret
call <elop>
```

DTC

```
movl (%esi),%eax
addl $4,%esi
jmp  *%eax
```

ITC

```
addl $4,%esi
movl (%esi),%eax
movl (%eax),%eax
jmp  *%eax
```

TTC

```
  movsbl (%esi),%eax
  incl   %esi
  movl   step(,%eax,4),%eax
  jmp    *%eax
step:
  <vector table>
```

STC

```
.L16:
  movsbl (%ebx),%eax
  incl   %ebx
  cmpb   $4,%al
  ja     .L16
  movl   %al,%eax
  jmp    *.L8(,%eax,4)
.L8:
  <vector table>
.L<token>:
  jmp    .L16
```

switching overhead
in # of x86 instructions

„bottom-up“ virtualization

Goldberg

A Virtual Computer System is a hardware-software duplicate of a real existing computer system in which a statistically dominant subset of the virtual processor‘s instructions execute directly on the host processor in native mode.

Species of virtualization

- self-virtualization
  - the VM is identical to the host
- family-virtualization
  - the VM is a member of the same computer family as the host
- para-virtualization
  - the VM is similar — but not identical — to the host

Goldberg, Architectural principles for virtual computer systems

Virtual machine monitor

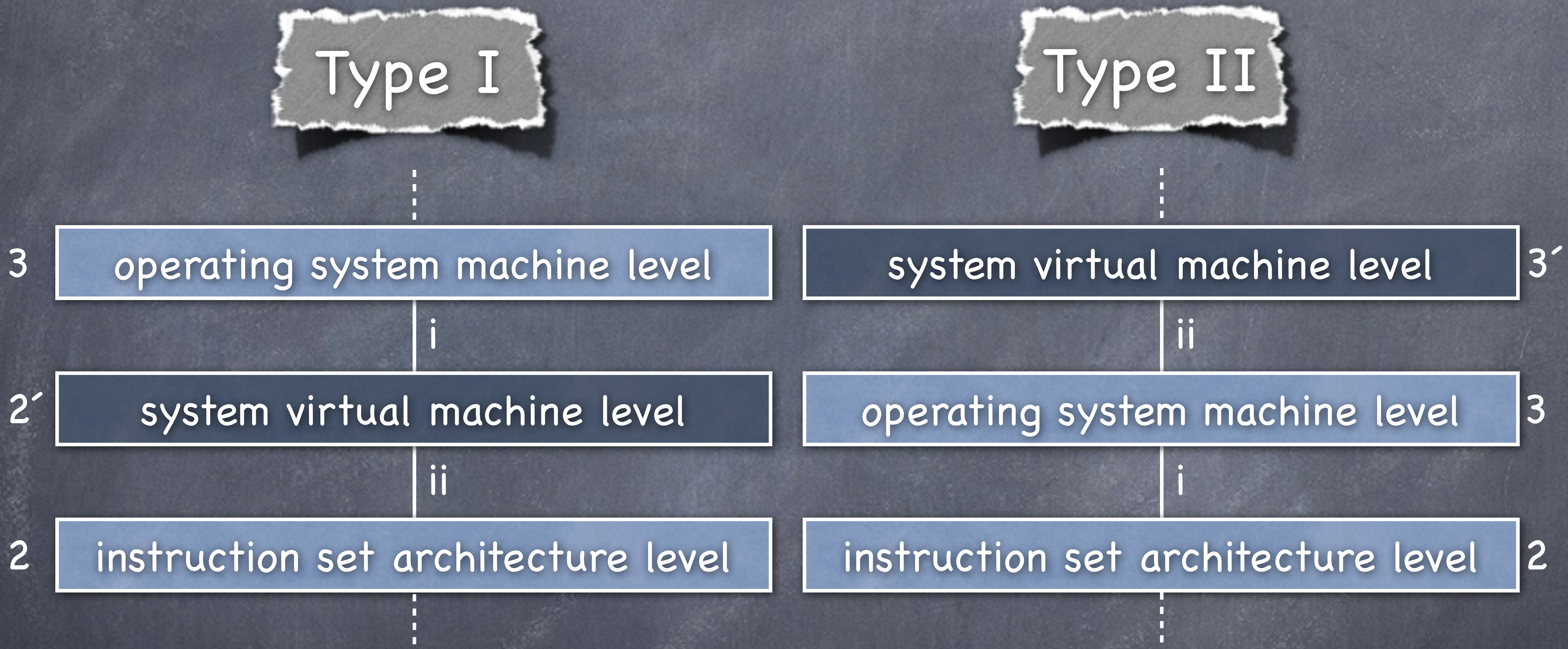
„The program executing on the host machine that creates the VCS environment“

control program (IBM)

- Type I
  - „the VMM runs on a bare machine“
- Type II
  - „the VMM runs on an extended host“
    - „under the host operating system“

Goldberg, Architectural principles for virtual computer systems

System virtual machine



Type I

3 operating system machine level

i

2´ system virtual machine level

ii

2 instruction set architecture level

Type II

system virtual machine level 3´

ii

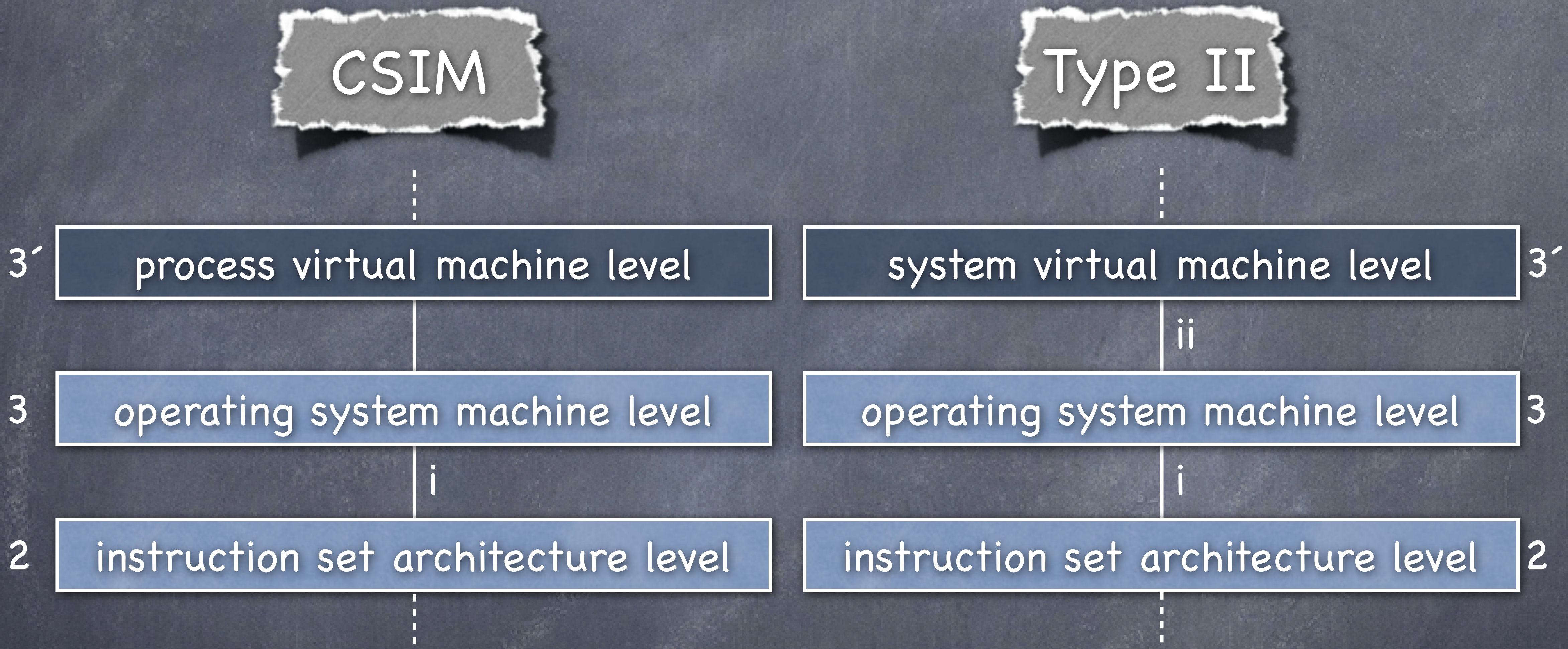
operating system machine level 3

i

instruction set architecture level 2

- partial interpretation:
  (i) operating system, (ii) virtual machine monitor

Process vs. system VM



CSIM

3´ process virtual machine level

3 operating system machine level

i

2 instruction set architecture level

Type II

system virtual machine level 3´

ii

operating system machine level 3

i

instruction set architecture level 2

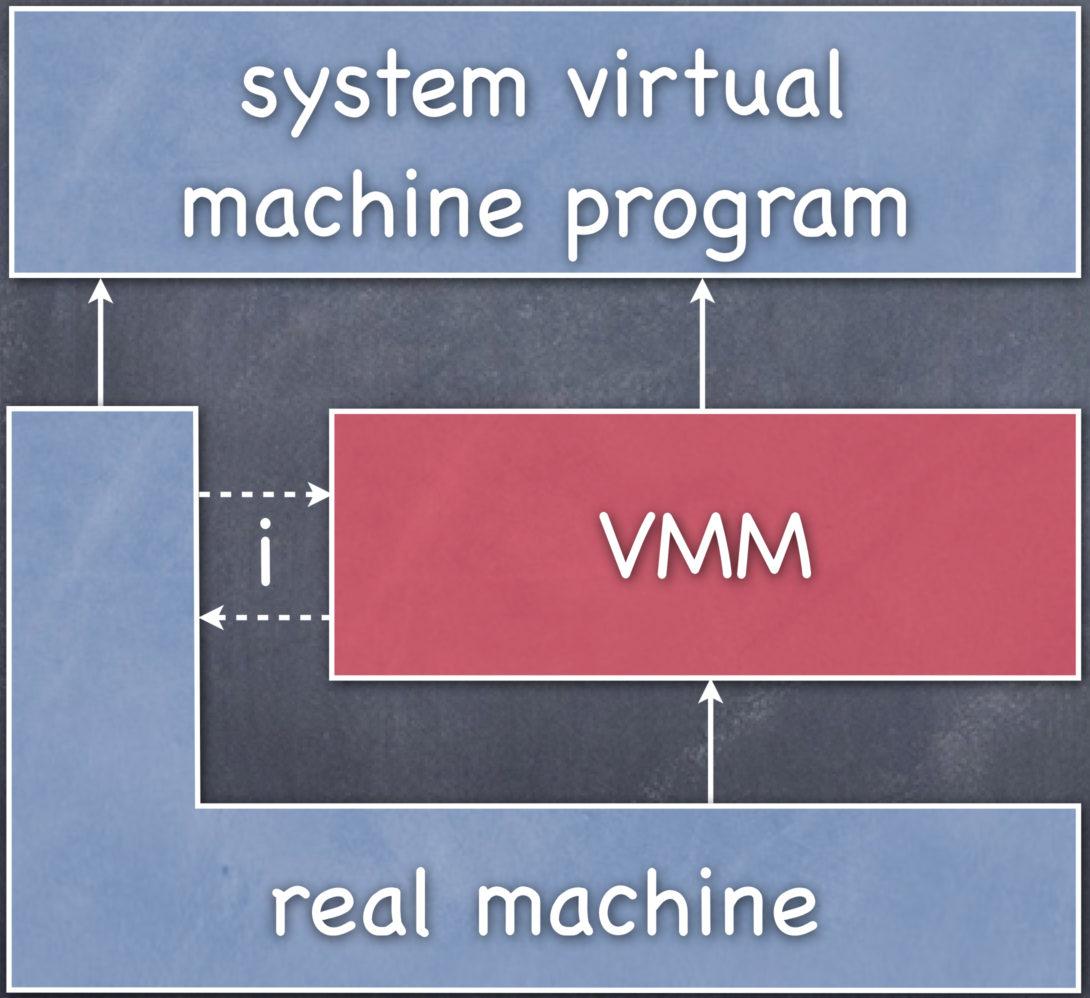
- partial interpretation:
  (i) operating system, (ii) virtual machine monitor

Separation of concerns

- Type I VMM based VCS, „weak“ first
  - system scheduling
  - (real) resource management

  } specific to OS
  - functions unspecific to provide a VCS
- Type II VMM based VCS, „strong“ first
  - system/resource scheduling done by OS
  - OS gives extended machine environment to the VMM program(s)

Self-virtualization



- real machine
  - Type I ☛ ISAL
  - Type II ☛ OSML
- needs virtualizable ISA level
- recursion property

i. exception (cf. p. 15)

Virtualizable ISA level

1. roughly equivalent execution of non-privileged instructions in supervisor and user mode
2. protection of supervisor mode programs
3. interception of sensitive instructions
   - 3.a) alter/query system state
   - 3.b) alter/query state of reserved entities
   - 3.c) reference to protection system
   - 3.d) I/O

Goldberg, Architectural principles for virtual computer systems

Sensitive instruction

„Any instruction whose direct execution [by the virtual machine] cannot be tolerated "

- easygoing with privileged instructions when being executed in user mode ☛ trap
- quite the contrary: unprivileged instructions
  - headliner Intel Pentium*

    (3.b) SGDT, SIDT, SLDT; [SMSW;] PUSHF, POPF

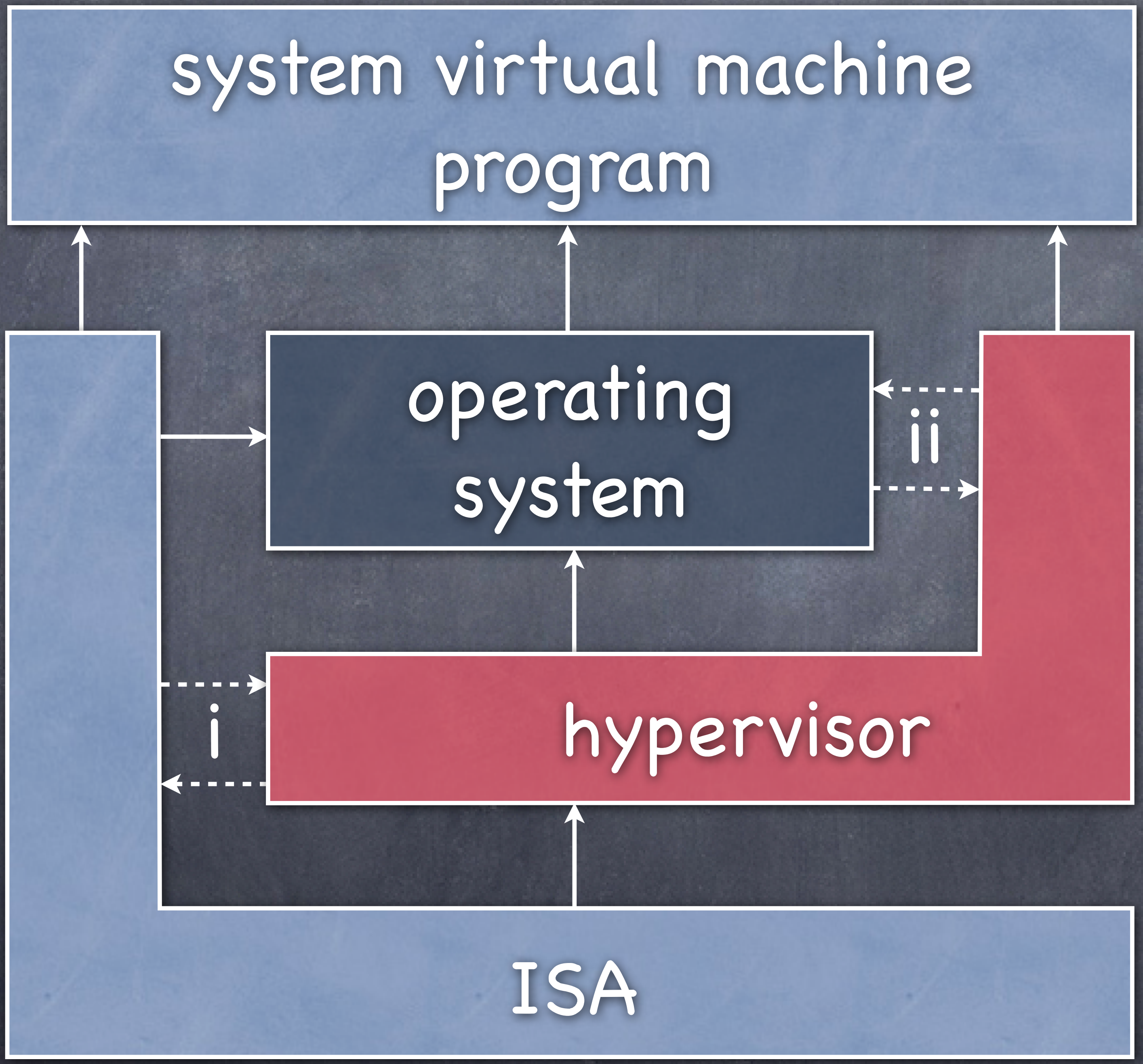
    (3.c) LAR, LSL, VERR, VERW; POP, PUSH; CALL, INT n, JMP, RET; STR, MOVE

*Robin & Scott, Analysis of the Intel Pentium's ability to support a secure virtual machine monitor

Handling unprivileged sensitve instructions

- partial virtualization (CTSS)
  - multiprogramming, only ☛ is not an issue
- full virtualization (VMware)
  - binary translation on guest code
    - explicit „VMM call“, e.g.: popf ☛ int $99
    - VMM emulates „patched“ instructions
  - sort of hybrid virtual machine (HVM)
- para-virtualization (VM/370, Denali, Xen)

Para-virtualization



- OS ☛ hypercalls
  - VM/370: DIAG
  - Xen: int $130

i. real exception (cf. p. 15)

ii. virtual exception

Hypervisor

Legend has it that the term was coined during the development of VM/370.

- colourful term:
  - a synonym for VMM, Type I or II
  - a thin software layer between OS and HW
  - a software layer that implements a VM
    - imperceptible by the SW run by the VM
  - an entity that monitors guest systems

Contemporary issues

Ongoing research

- large-scale IT infrastructure abstraction
- prevention of virtualization sprawl
- [server] consolidation
- de-duplication of virtual machine state
- performance: 50-80% effectiveness*

*VMware

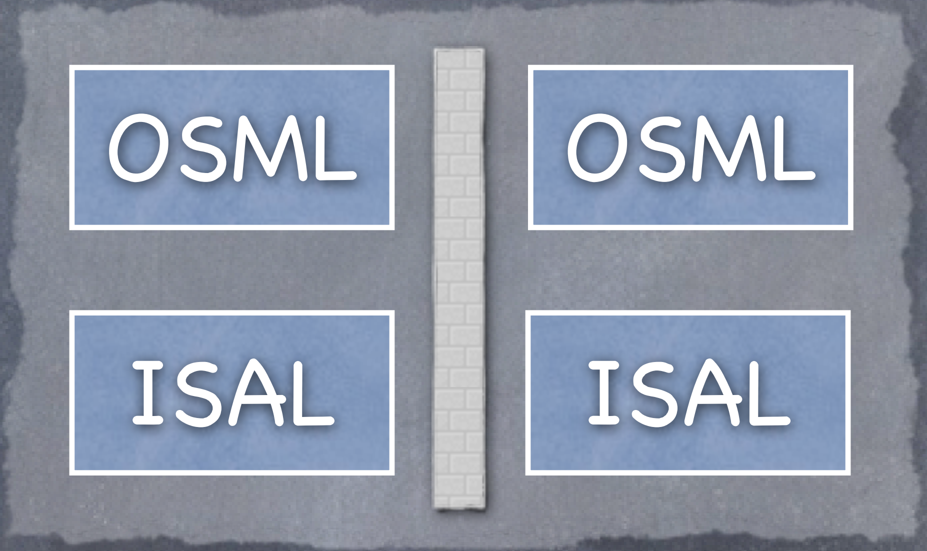
Consolidation

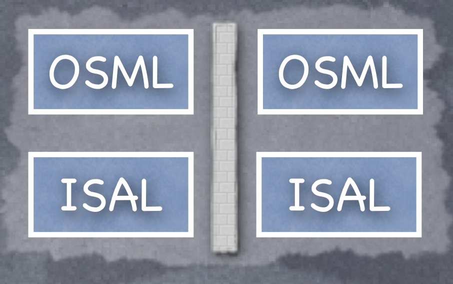
- logical
  - simplified operations, common processes
- physical
  - co-location of multiple platforms, fewer sites
- workload
  - more users, same application, fewer platforms
- application
  - combine mixed workloads, fewer platforms

rationalized

x

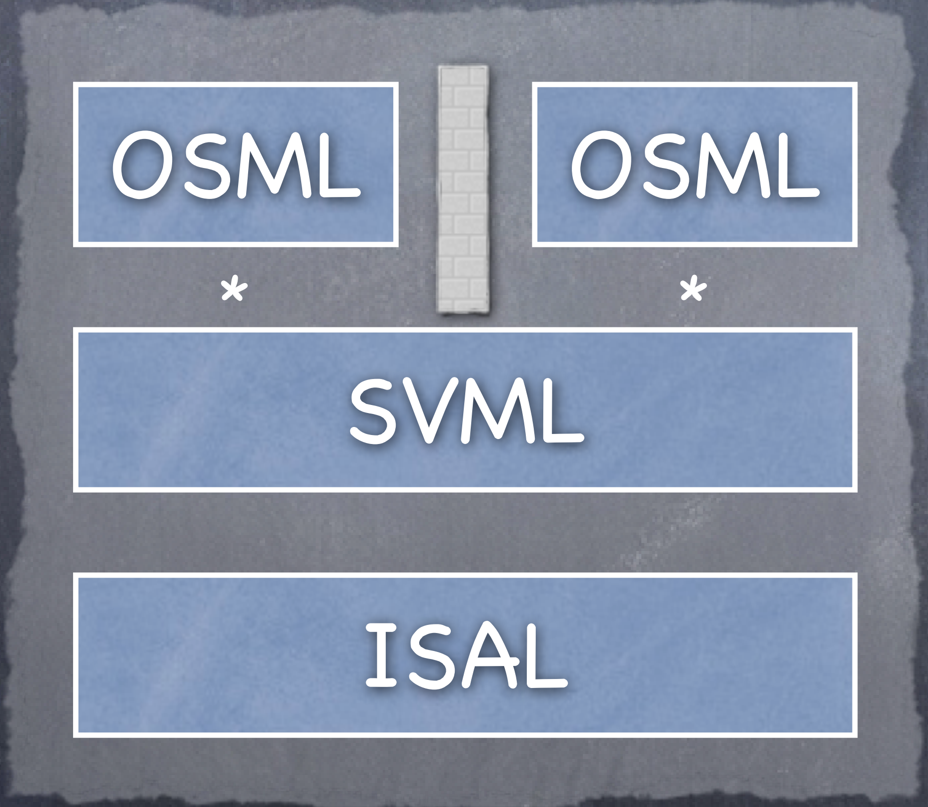
Consolidation



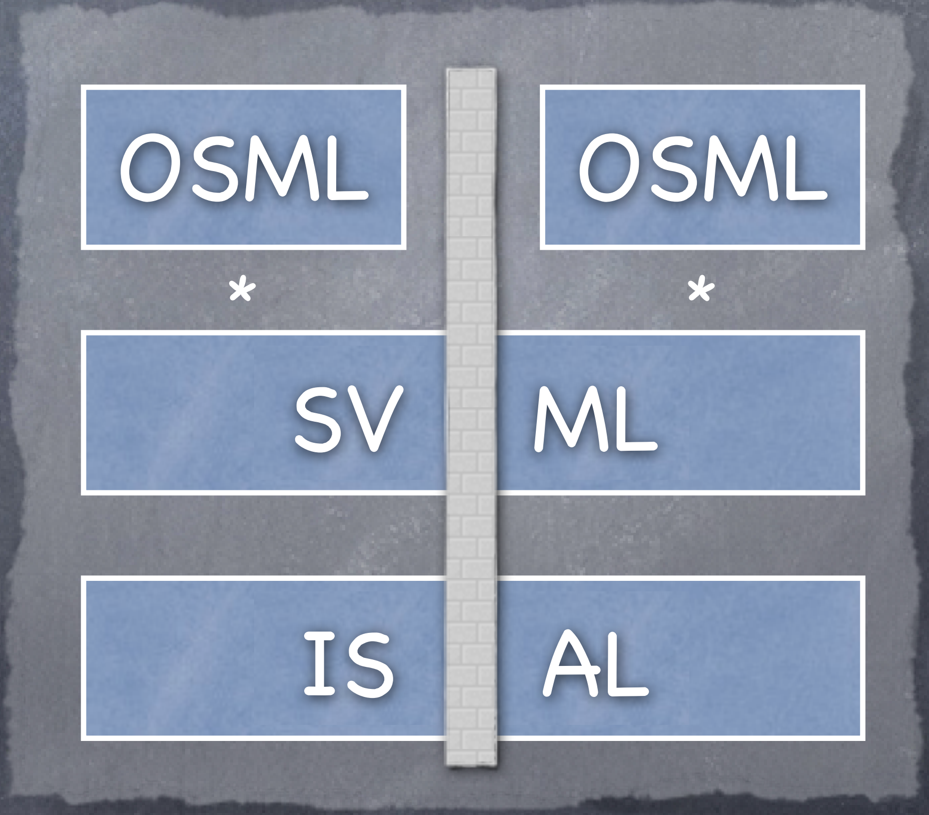


Consolidation

partitioning in time



partitioning in space



*interference with (guest) operating system

Partitioning techniques

- with HW support
  - physical
  - logical
    - microprogramm
    - hypervisor

efficiency

- without HW support
  - SVM-based
    - homogeneous
    - heterogeneous
  - OS-based

flexibility

Performance handicaps

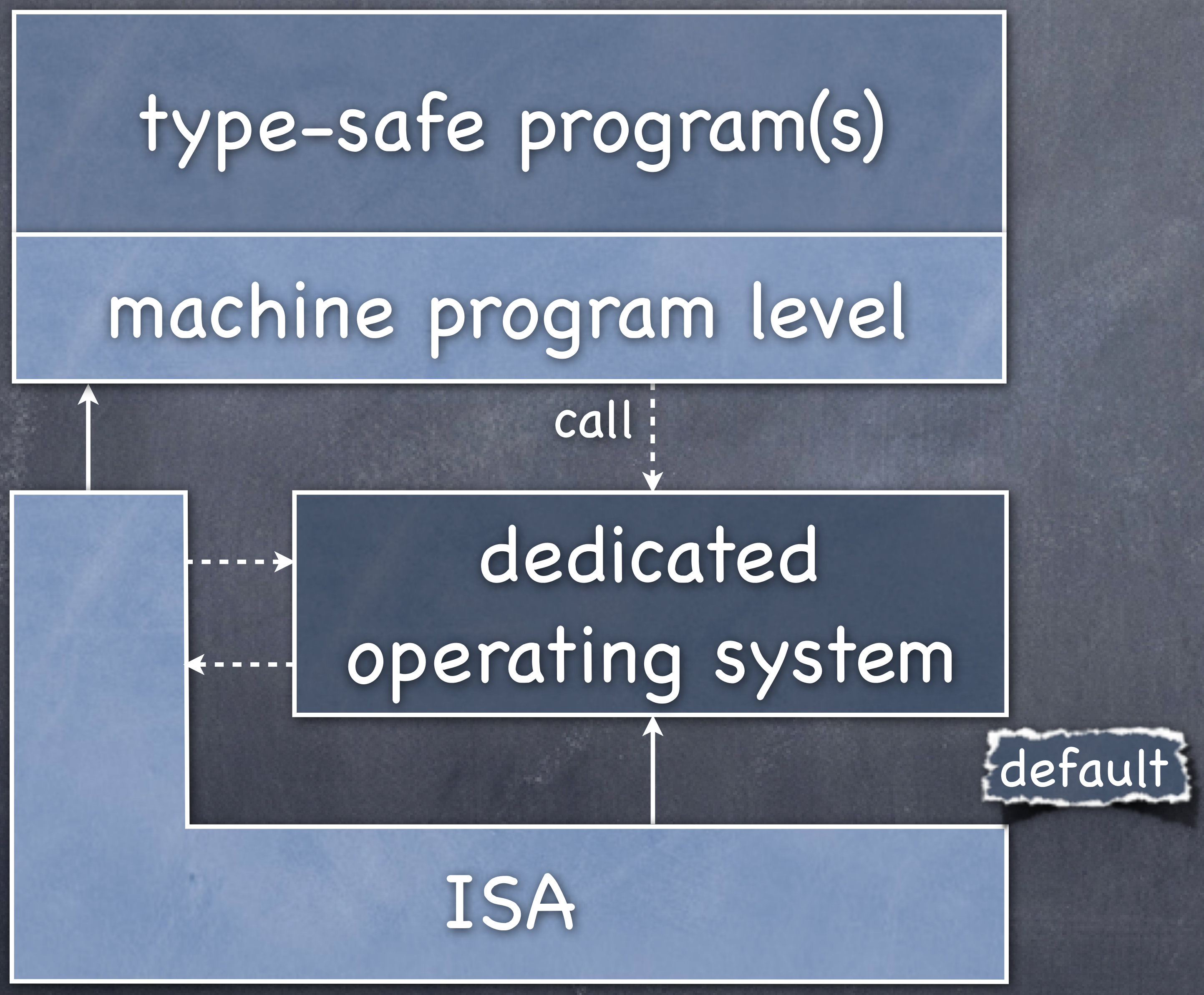
- partial interpretation of system requests
  - traps, interrupts
- maintenance of real-machine state
  - processor state, shadow page tables, …
- interference with guest operating system
  - scheduling, synchronization
- interference with guest system(s) in general
  - cache-aware (machine) programs

Temporary virtualization: anticipatory, on demand

- transient multiprogramming
  - partial virtualization
- transient virtual machine monitor
  - self-virtualization
  - para-virtualization

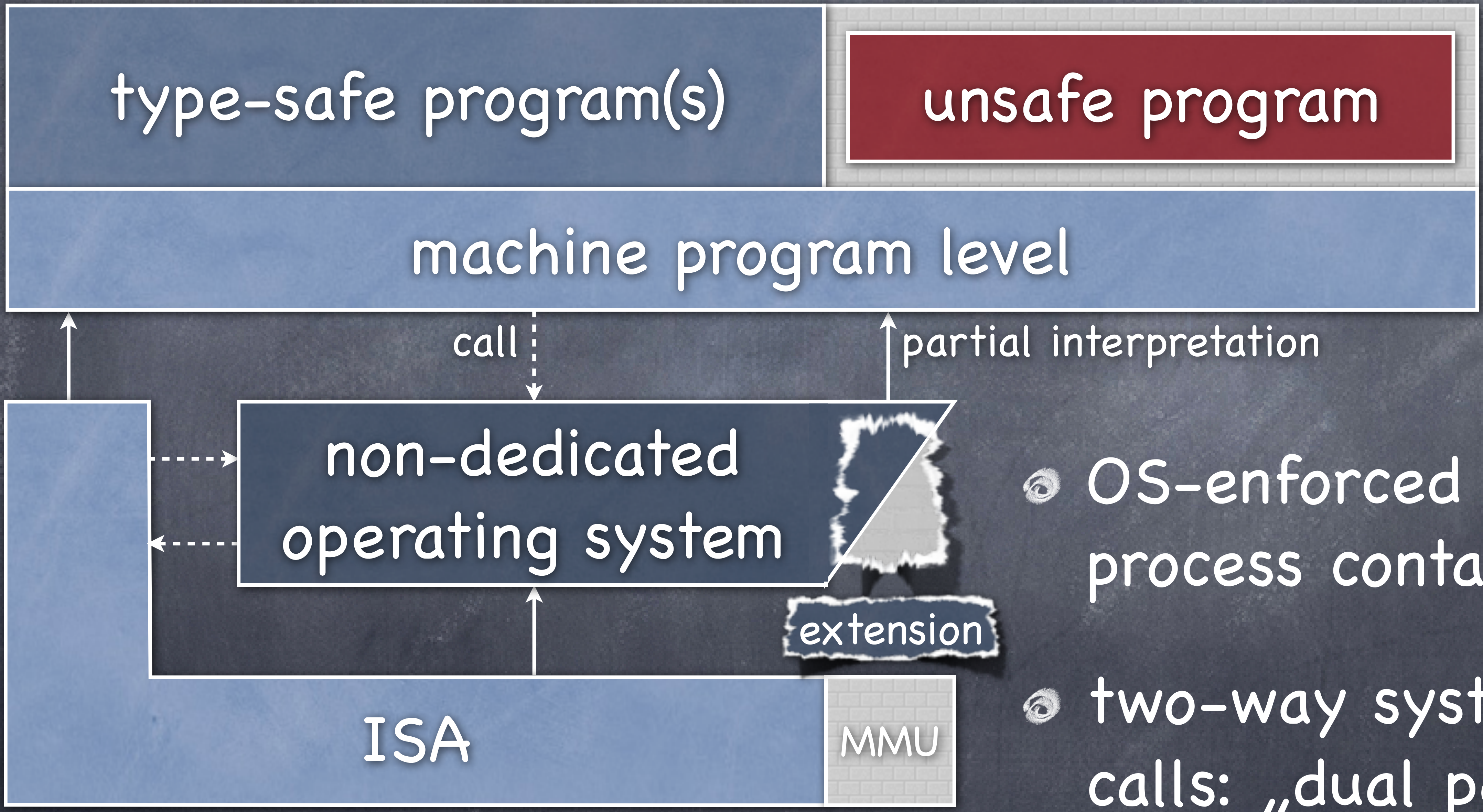
dynamically alterable operating system

Transient multiprogramming



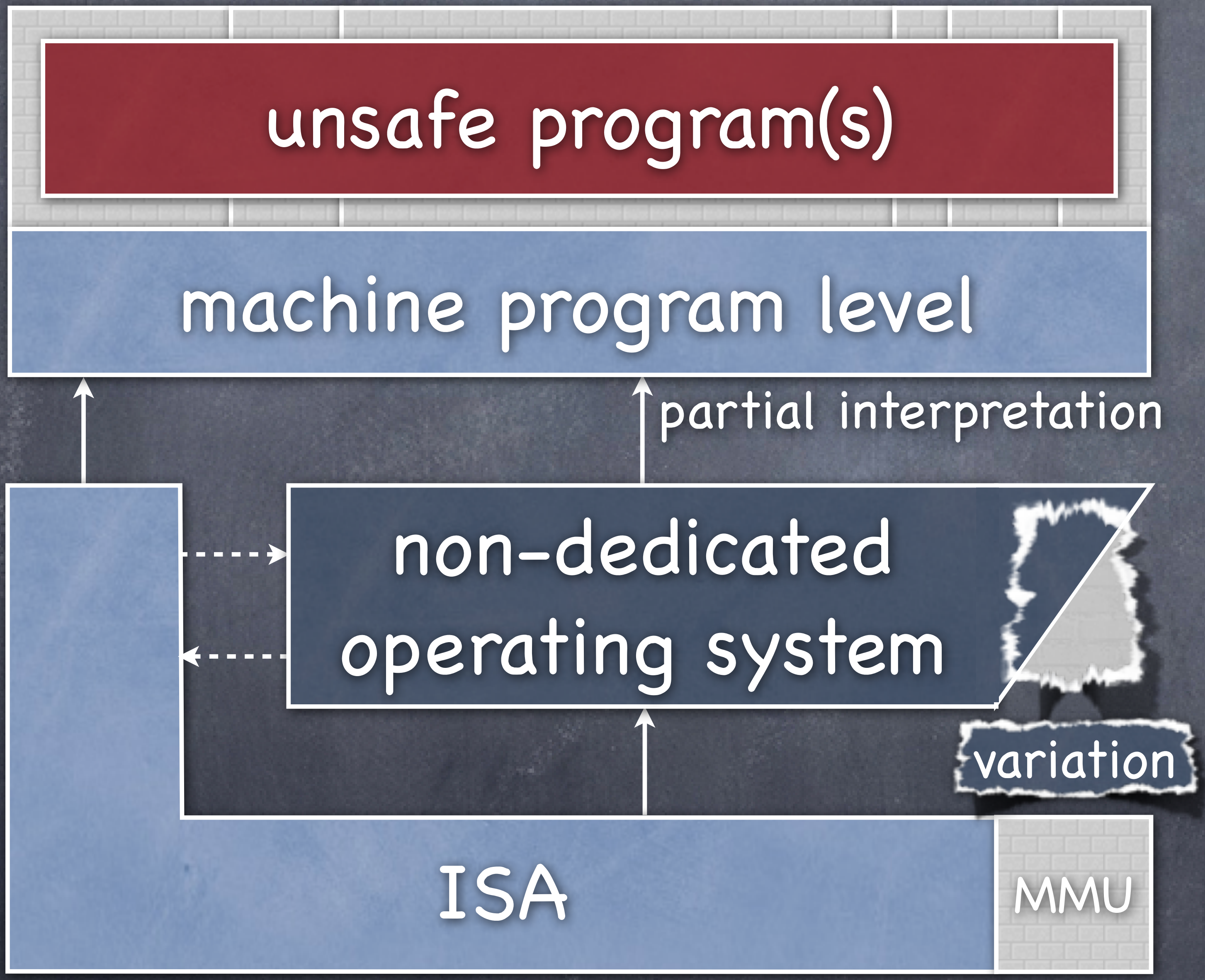
- type-safe language (e.g. X10)
- compiler-enforced protection domains
- globally integrative address space
- library-like operating system

Transient multiprogramming



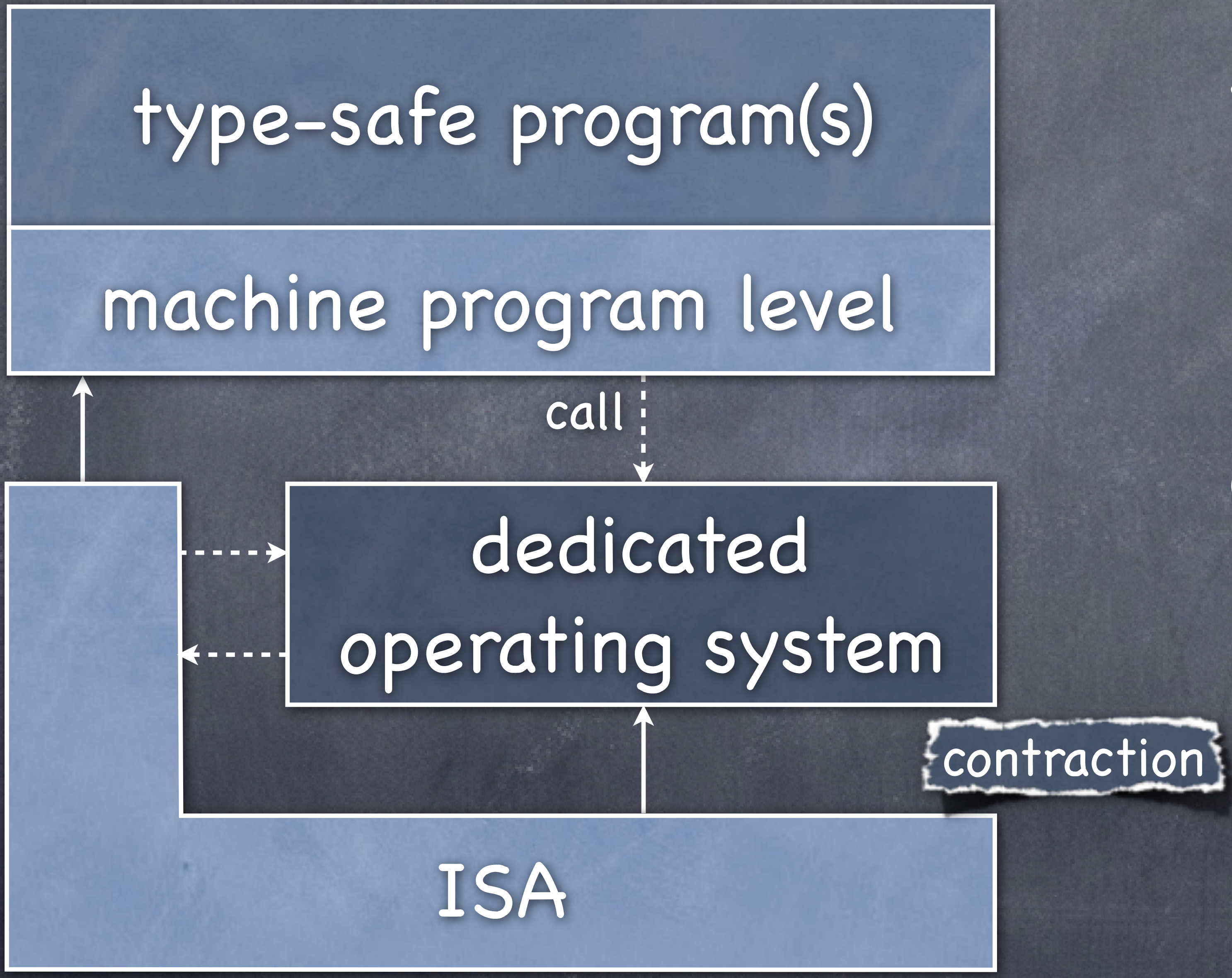
- OS-enforced process containment
- two-way system calls: „dual ported“

Transient multiprogramming



- type-unsafe languages (e.g. C)
- OS-enforced protection domains
- private address spaces
- one-way system calls: as usual

Transient multiprogramming



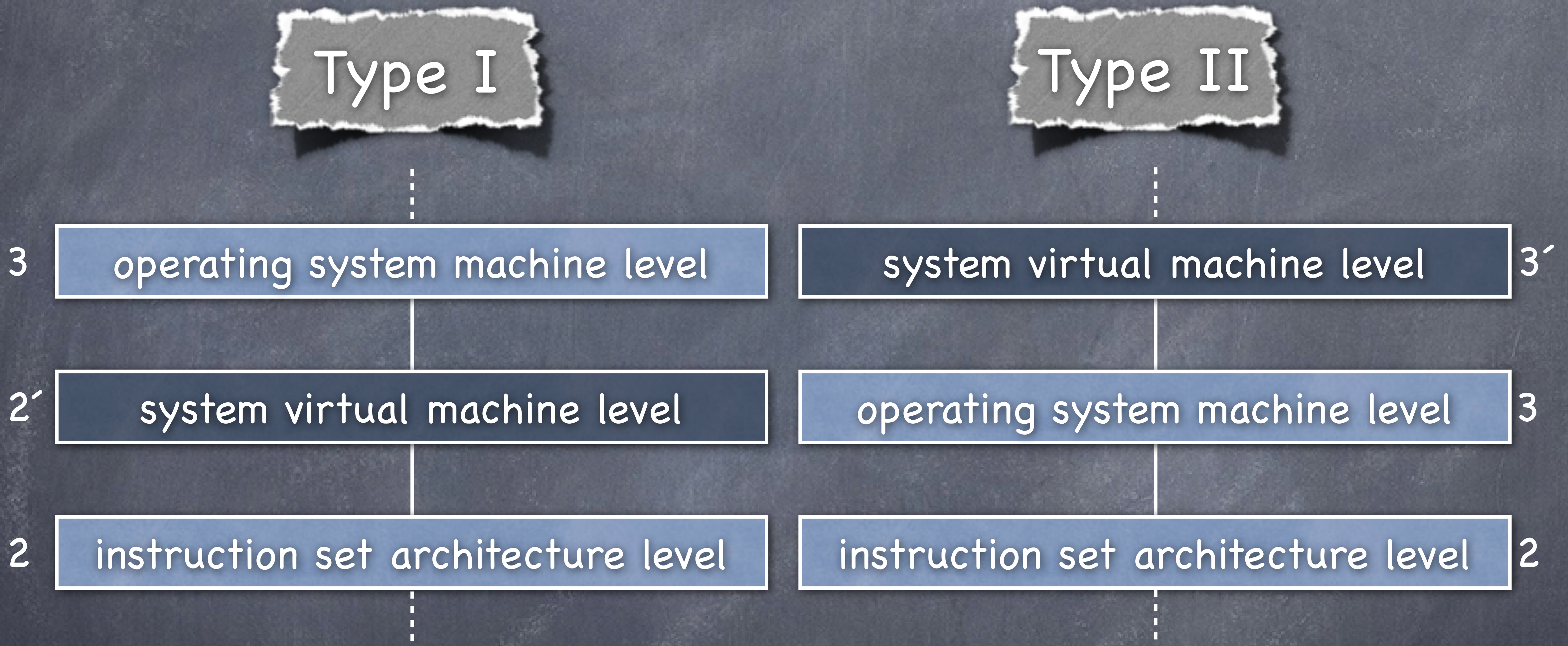
re-establishment:

- minimal subset of system functions

deconstruction:

- minimal system...
  - extensions
  - variations

Transient VMM



Type I

3 operating system machine level

2´ system virtual machine level

2 instruction set architecture level

Type II

system virtual machine level 3´

operating system machine level 3

instruction set architecture level 2

transparent to OS

intransparent to OS

Epilogue

Emperor‘s old/new clothes?

- CTSS (1961), CP-40 (1964), VM/370 (1972)
- Moore‘s law leveraged system virtualization
- multifaceted: process/system virtual machine
- a means to an end:
  - consolidation, customization, maintenance
  - compatibility, reliability, security
- but not the last conclusion of wisdom…

Parnas

Designing software for the ease of extension and contraction.

Some users may require only a subset of the services of features that other users need. These „less demanding“ users may demand that they are not be forced to pay for the resources consumed by the unneeded features.

Thanks for your attention!

Bibliography

[1] Goldberg: Architectural principles for virtual computer systems

[2] Popek, Goldberg: Formal requirements for virtualizable third generation architectures

[3] Robin, Irvine: Analysis of the Intel Pentium‘s ability to support a secure virtual machine monitor

[4] Tanenbaum: Structured computer organization

[5] Whitacker, Shaw, Gribble: Denali: Lightweight virtual machines for distributed and networked applications