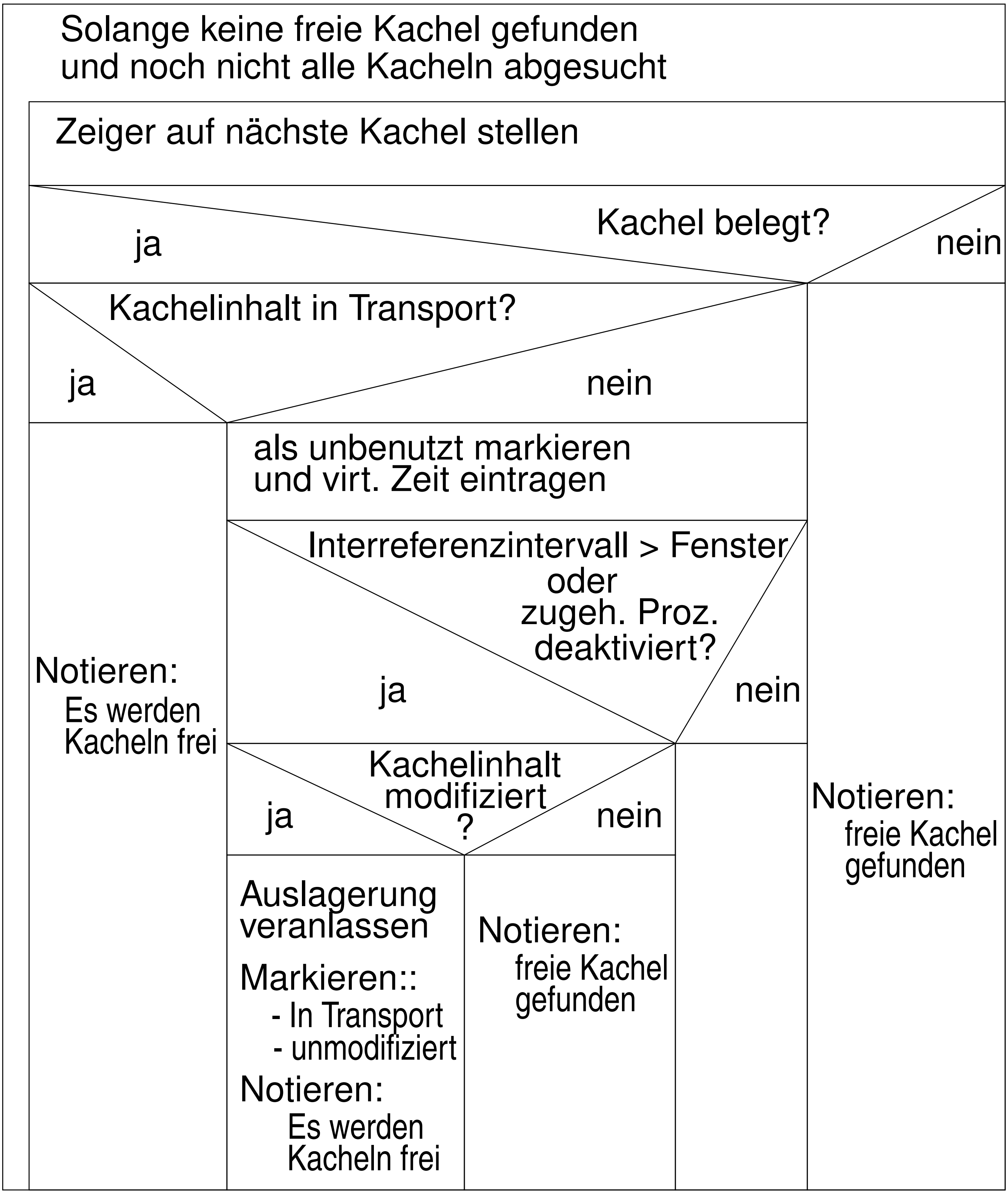
5.4 Der Algorithmus wsclock



sp1/96/folien/3: 18.12.1996

```
class pagedescriptor
  { int modified;
      // Modifikationsindikator
    int used;
      // Referenzindikator
    int present;
      // Praesenzindikator
    .
    .
    .
  public:
    pagedescriptor()
      { modified = FALSE;
        used = FALSE;
        present = FALSE;
      };
    void set_present()
      { present = TRUE; };
    void set_used()
      { used = TRUE; };
    void set_modified()
      { modified = TRUE; };
  friend class framedeskriptor;
  friend class frametable;
  friend class process;
  };
```

```
class process
  { int active;
      // Diese Komponente hat genau dann den Wert TRUE,
      // wenn sich der Prozess im Superzustand aktiv
      // befindet.
    int my_virtual_time;
      // Virtuelle Zeit, die angibt, wieviel CPU-Zeit
      // der Prozess bereits verbraucht hat.
    pagedescriptor pagetable[NUMBER_OF_PAGES];
  public:
    process()
      { active = TRUE;
        my_virtual_time = 0;
      };
    int is_active()
      { return active; };
    void set_active()
      { active = TRUE; };
    void set_inactive()
      { active = FALSE; };
    int is_present(int process_id)
      { return pagetable[process_id].present; };
    void set_present(int process_id)
      { pagetable[process_id].set_present(); };
    void set_used(int process_id)
      { pagetable[process_id].set_used(); };
    void set_modified(int process_id)
      { pagetable[process_id].set_modified(); };
    int get_virtual_time()
      { return my_virtual_time; };
    void increment_virtual_time()
      { my_virtual_time++; };
  friend class frametable;
  };
```

```
class framedeskriptor
  { int process_id;     // Nummer des Prozesses, der
                       // hier eine Seite gelagert hat.
    int pagenumber;     // Nummer der hier gelagerten
                       // Seite.
    int virtual_time;
    int in_transport;
    int in_use;
  public:
    void initialize(int f_process_id,
                    int f_pagenumber,
                    int f_vt)
      { process_id = f_process_id;
        pagenumber = f_pagenumber;
        virtual_time = f_vt;
        in_transport = FALSE;
        in_use = TRUE;
      };

  friend class frametable;
  };
```

```
class frametable
  { framedeskriptor frame[NUMBER_OF_FRAMES];
  public:
    void initialize(int i, framedeskriptor &newframe)
     { frame[i] = newframe;
     };
    int is_in_use(int i)
     { return frame[i].in_use; };
    void set_unused(int i)
     { processtable[frame[i].process_id].
         pagetable[frame[i].pagenumber].used = FALSE;
     };
    int frame_is_used(int i)
     { return processtable[frame[i].process_id].
                pagetable[frame[i].pagenumber].used; };
    int in_transport(int i)
     { return frame[i].in_transport; };
    void set_in_transport(int i)
     { frame[i].in_transport = TRUE; };
    void reset_in_transport(int i)
     { frame[i].in_transport = FALSE;
       frame[i].in_use = FALSE;
     };
    void update_virtual_time(int i)
     { frame[i].virtual_time
          = processtable
              [frame[i].process_id].my_virtual_time; };
    int unreferenced_virtual_interval(int i)
     { return processtable
                [frame[i].process_id].my_virtual_time
              - frame[i].virtual_time;
     };
    int active(int i)
     {return processtable
              [frame[i].process_id].active;
     };
```

sp1/96/folien/3: 18.12.1996


```
    int modified(int i)
     {return processtable[frame[i].process_id].
             pagetable[frame[i].pagenumber].modified;};
    void set_absent(int i)
     { processtable[frame[i].process_id].
              pagetable[frame[i].pagenumber].present
          = FALSE; };
    void set_empty(int i)
      { frame[i].in_use = FALSE; };
    void set_unmodified(int i)
      { processtable[frame[i].process_id].
              pagetable[frame[i].pagenumber].modified
           = FALSE; };
    int frametable::
          determine_free_frame(int *framenumber);
  } GFT;    // global_frametable
    // Die Kacheltabelle enthaelt zu jeder
    // der m Arbeitsspeicherkacheln
    // - einen Verweis pagenumber auf das Seiten-
    //   register, das auf diese Kachel verweist,
    //   bzw. NIL, wenn die Kachel nicht belegt ist;
    // - in process-id die Identifikation des
    //   Prozesses, fuer den eine Seite in dieser
    //   Kachel gespeichert ist, bzw. NIL, wenn die
    //   Kachel nicht belegt ist;
    // - in in_transport die Angabe, ob fuer diese
    //   Kachel ein Ein- oder Austransfer laeuft;
    //   diese Groesse wird durch das Betriebssystem
    //   nach Abschluss des Seitentransfers wieder
    //   auf FALSE gesetzt;
    // - in virtual_time den virtuellen Zeitpunkt,
    //   zu dem der zugehoerige Prozess die Ein-
    //   lagerung veranlasst hat bzw. zuletzt ueber-
    //   prueft wurde, ob der Kachelinhalt aus-
    //   gelagert werden kann.
    // - in in_use die Angabe, ob die Kachel in Benut-
    //   zung ist, d. h. eine Seite enthaelt oder ein
    //   die Kachel betreffender Transfer laeuft.
```

```
int searchpointer;
    // Suchzeiger, der festhaelt, welche Kachel
    // zuletzt darauf ueberprueft wurde, ob sie
    //  als Auslagerungskandidat in Frage kommt.

process processtable[NUMBER_OF_PROCESSES];

int number_of_active_processes;

void start_transfer_to_disk(int k)
{
  . . .
}
```



```
int frametable::determine_free_frame(int *framenumber)
{ int frame_available, frame_found;
  int number_of_inspected_frames;
  frame_available = FALSE;
  frame_found = FALSE;
  number_of_inspected_frames = 0;
  do {
    searchpointer = (searchpointer+1)%NUMBER_OF_FRAMES;
    number_of_inspected_frames++;
    if (!GFT.is_in_use(searchpointer)) {
      frame_found = TRUE;
    } else {
      if (!GFT.in_transport(searchpointer)) {
        if (GFT.frame_is_used(searchpointer)) {
          GFT.update_virtual_time(searchpointer);
          GFT.set_unused(searchpointer);
        }
        if (GFT.unreferenced_virtual_interval
                                        (searchpointer)
                > WINDOW
              || !GFT.active(searchpointer)) {
          if (!GFT.modified(searchpointer)) {
            frame_found = TRUE;
            GFT.set_absent(searchpointer);
            GFT.set_empty(searchpointer);
          } else {
            start_transfer_to_disk(searchpointer);
            frame_available = TRUE;
            GFT.set_in_transport(searchpointer);
            GFT.set_unmodified(searchpointer);
          }
        }
      } else {
        frame_available = TRUE;
      }
    }
  } while (!frame_found
              && number_of_inspected_frames
                                  != NUMBER_OF_FRAMES);
```

```
  if (frame_found) {
    *framenumber = searchpointer;
    return(1);
  } else {
    if (frame_available) {
      *framenumber = -1;
      return(0);
    } else {
      *framenumber = -1;
      return(-1);
    }
  }
}
```