Conclusies over semaforen

- gebruik semaforen is subtiel

- signal & wait operaties,
  en access van shared data,
  op allerlei plekken in de code

Kan dit niet handiger?

Dwz: zijn er geen betere abstracties ?

Ja: **monitoren** of **message passing**

Monitor (Sectie 5.4)

Programmeertaalconstructie

Levert zelfde functionaliteit als semaforen maar is eenvoudiger in gebruik.

Bedacht door Tony Hoare in 1974

Monitoren

Idee: groepeer operaties op gedeelde data, met mutual exclusion voor deze operaties, en met synchronisatie-condities.

Bijvoorbeeld voor de buffer
operaties: *take(item i)* en *append(item i)*
synchronisatie-condities: *empty* en *full*

Voordeel 1: Overzichtelijker, betere abstractie

Producer/consumer probleem wordt triviaal.

Alle complexiteit verstopt in de monitor

Voordeel 2: Monitor is een soort object. Sluit mooi aan bij OO talen.

```
monitor boundedbuffer {
 int in, out, count;
 char buffer[N];
 condition_var notfull, notempty;

 void produce(char x) {
  while (count==N) { cwait(notfull);  }
  buffer[in]=x; in=(in+1)%N; count++;
  cnotify(notempty);
 }

 void consume(char x) {
  while (count==0) { cwait(notempty);  }
  x=buffer[out]; out=(out+1)%N; count--;
  cnotify(notfull);
 }
}
```

Message Passing (Sectie 5.5)

Operaties

send(destination, message)

receive(source, message)

om boodschappen te versturen/ontvangen

Synchronisatie

- blocking of nonblocking send
- blocking of nonblocking receive

Addressering

- directe addressing:
  bestemming/bron is een process
- indirecte addressing:
  bestemming/bron is een mailbox/port/buffer

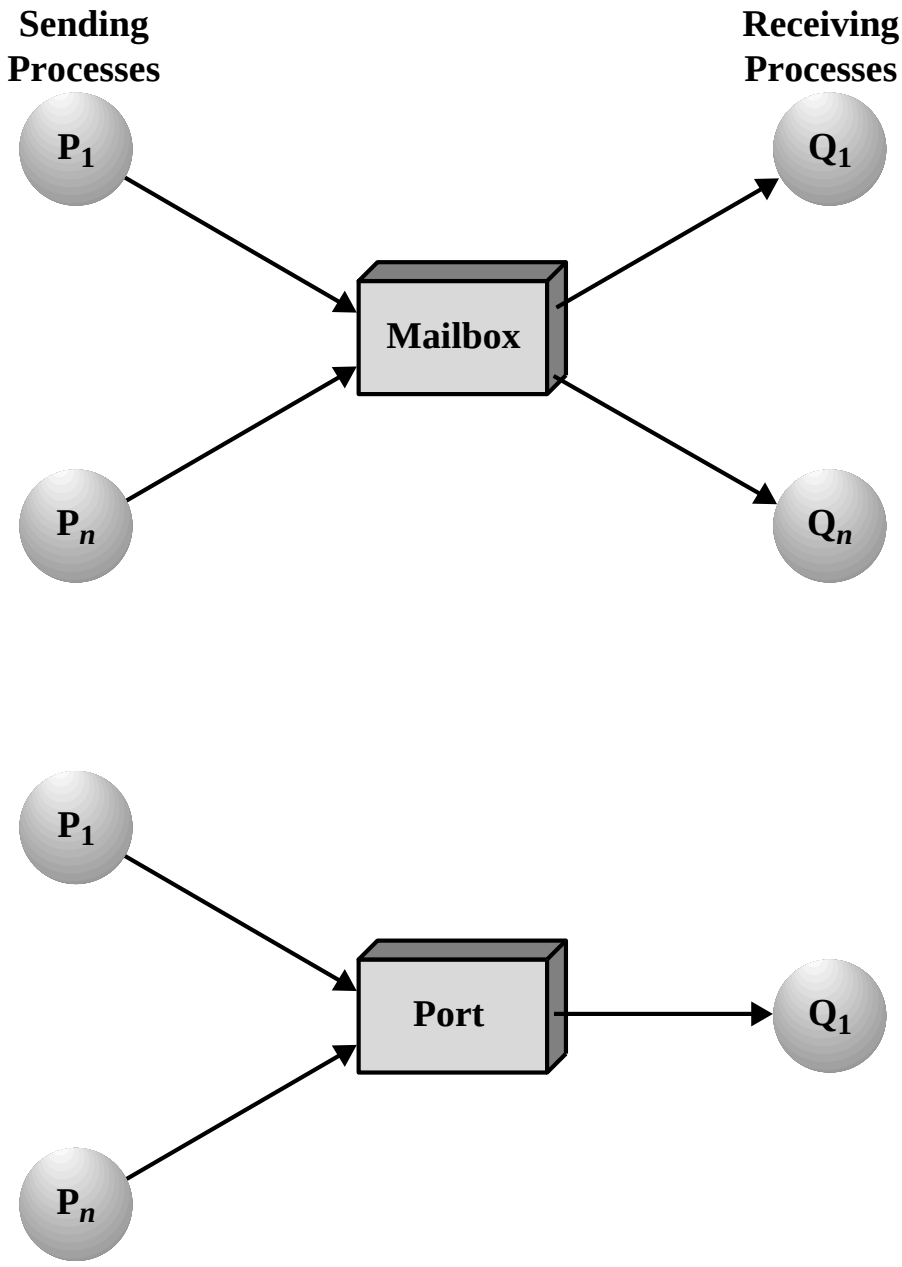

Figure 5.24 Indirect Process Communication

Mutual Exclusion via Messages

```
/* program mutualexclusion /*
const int n = /* number of processes */;
void P(int i)
{
 message msg;
 while (true)
 {
    receive(box,msg);
    /* critical section */;
    send(box,msg);
    /* remainder */;
  }
}
void main()
{
  create_mailbox(box);
  send(box,null);
  parbegin (P(1),P(2),...,P(n));
}
```

Readers/Writers Problem (Sectie 5.6)

Klassiek voorbeeld van synchronisatie.

Lees dit zelf!

H6: Deadlock & Starvation

Terminology:

- process *requests* resources
- OS *allocates* resources
- process *releases* resources

Dit kan leiden tot *deadlock* of *starvation*

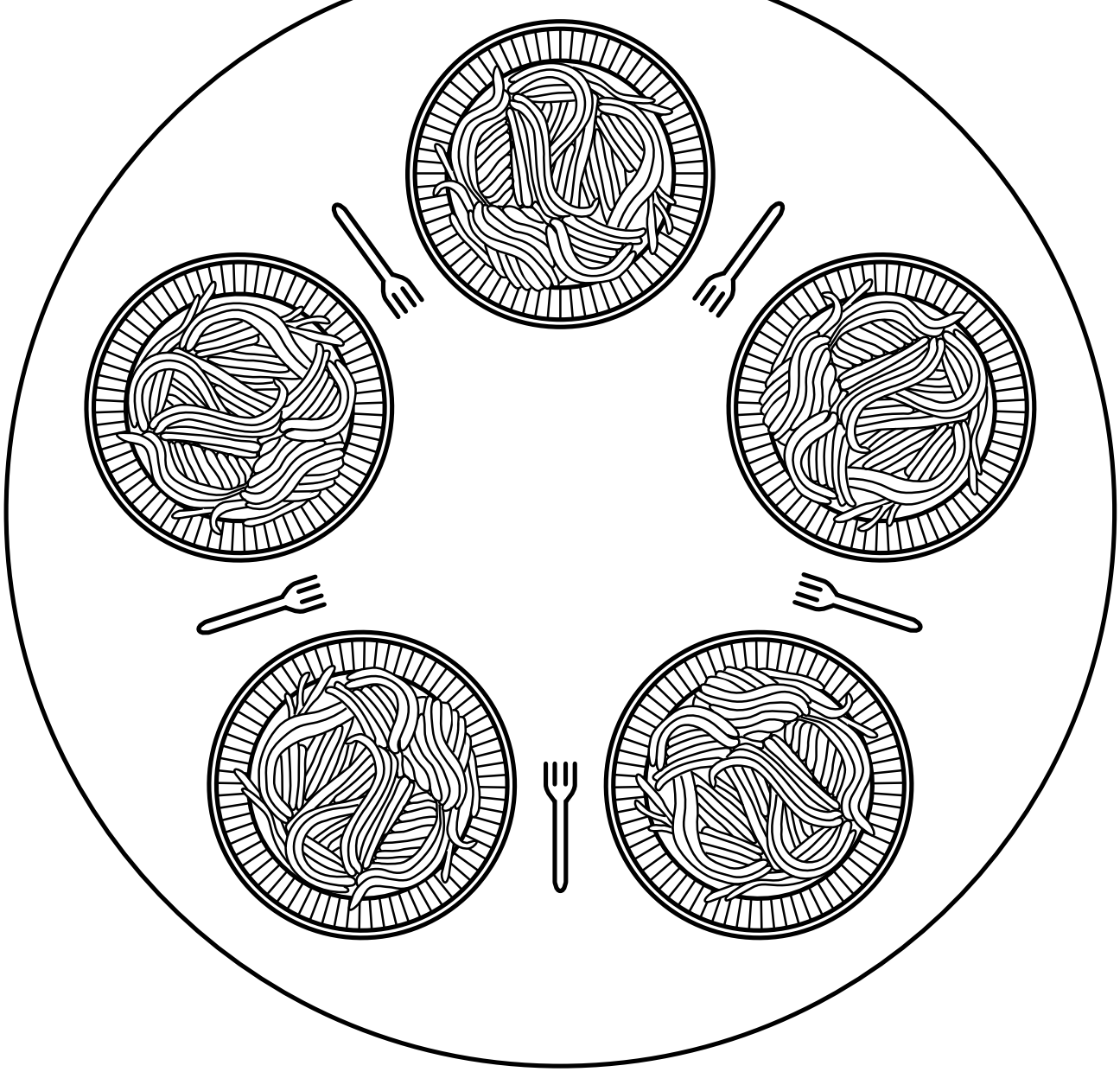

Fig. 2-31. Lunch time in the Philosophy Department.

Voorbeeld: Dining Philosophers

```
semaphore fork[4] (=1)

void philosopher(int i);
{ while (true)
    { think;
      wait(fork[i]);
      wait(fork[(i+1)mod 5];
      eat;
      signal(fork[i]);
      signal(fork[(i+1)mod 5];
    }
}
```

mogelijk deadlock !

Dining Philosophers

Een oplossing: maximaal 4 filosofen tegelijk aan tafel

```
semaphore fork[4] (=1)
semaphore room  (=4)

void philosopher(int i);
{ while (true)
    { think;
      wait(room);
      wait(fork[i]);
      wait(fork[(i+1)mod 5];
      eat;
      signal(fork[i]);
      signal(fork[(i+1)mod 5];
      signal(room);
    }
}
```

Resources kunnen zijn

- **reusable** : processoren, geheugen, devices, files ,. . .
- **consumable** : messages, information in buffers, signals aan semafoor

Allebei kunnen ze deadlock veroorzaken.

Condities voor Deadlock

1. **Mutual exclusion**

2. **Hold & Wait**

3. **No preemption of resources**

4. **Circular wait**

1,2 en 3 zijn *noodzakelijk* (*necessary*), maar niet *voldoende* (*sufficient*).

Als 1,2 en 3 gelden, dan is 4 *voldoende*.

Deadlocks: Wat doe je eraan ?

Er bestaat geen algemene, efficiente oplossing.
Je kunt deadlock proberen te

1. **voorkomen**
   maak deadlocks onmogelijk

2. **vermijden**
   garandeer steeds dat deadlock voorkomen kan worden

3. **detecteren** en probeer ze dan op te lossen

4. of **niks** doen

Voorkomen van Deadlocks?

Wat kunnen we aan oorzaken deadlocks doen?

- **Mutual exclusion**
  Meestal niks aan te doen.

- **No preemption of resources**
  Meestal niks aan te doen.

- **Hold & Wait**
  Vraag voor alle resources tegelijk.
  inefficient & gevaar op starvation

- **Circular wait**
  Orden resources en vraag om resources in die volgorde.
  inefficient

Voorkomen van Deadlocks?

Voorbeeld: deadlock door m.e. printer
Voorkomen door uitvoer te spoolen.

Voorbeeld: deadlock door gebrek aan geheugen.
Voorkomen door alle geheugen aan het begin te vragen.

Vermijden van Deadlock

Ken geen verzoeken toe die tot deadlock zouden kunnen leiden.

Bijv. Dijkstra's bankier's algoritme.

Probleem: vereist informatie over de toekomst en die heb je nooit . . .

Detecteren Deadlock

Hoe? Detecteer circular wait's.

En dan: wat kun je doen als deadlock gedetecteerd is ?