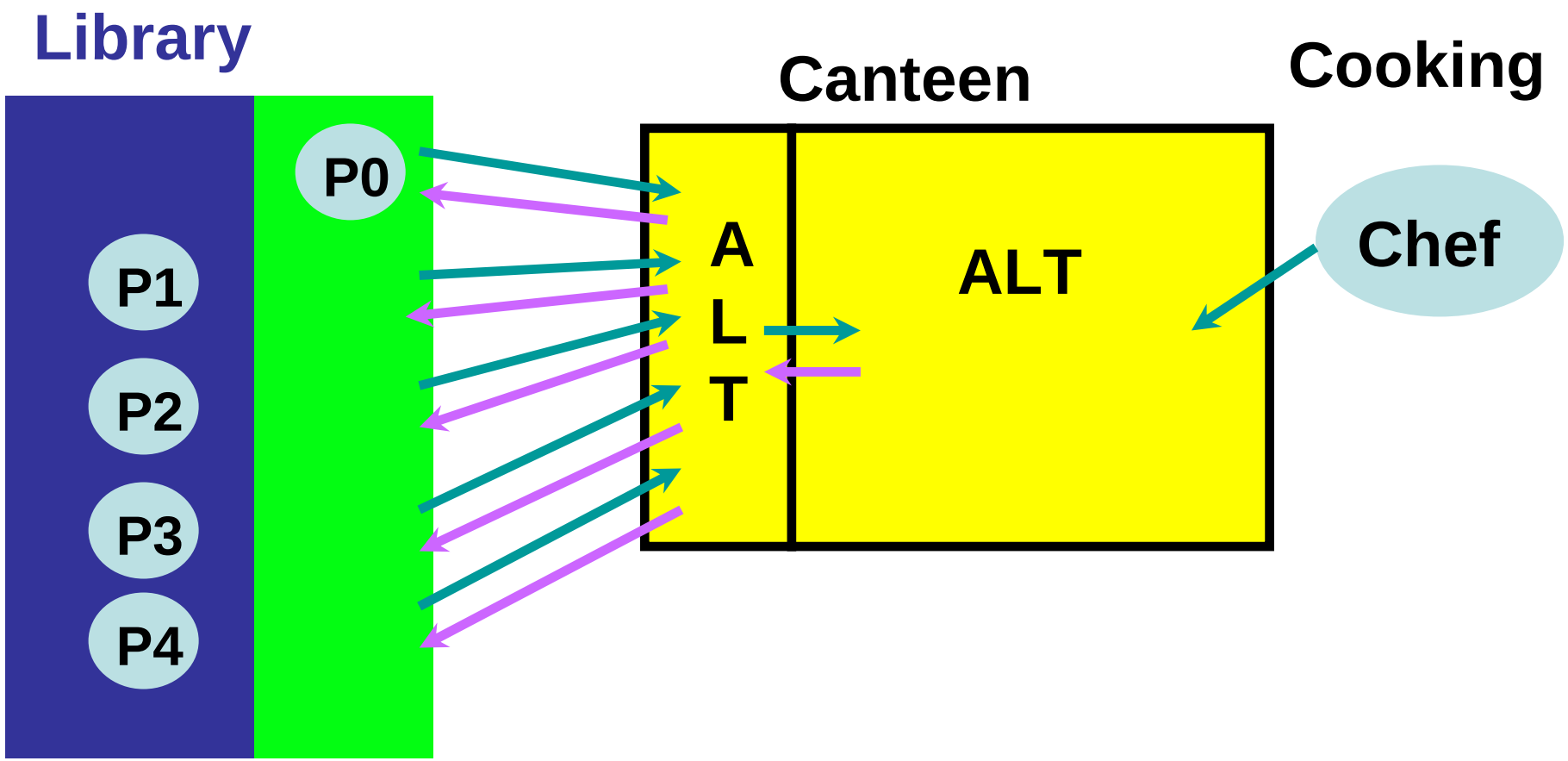
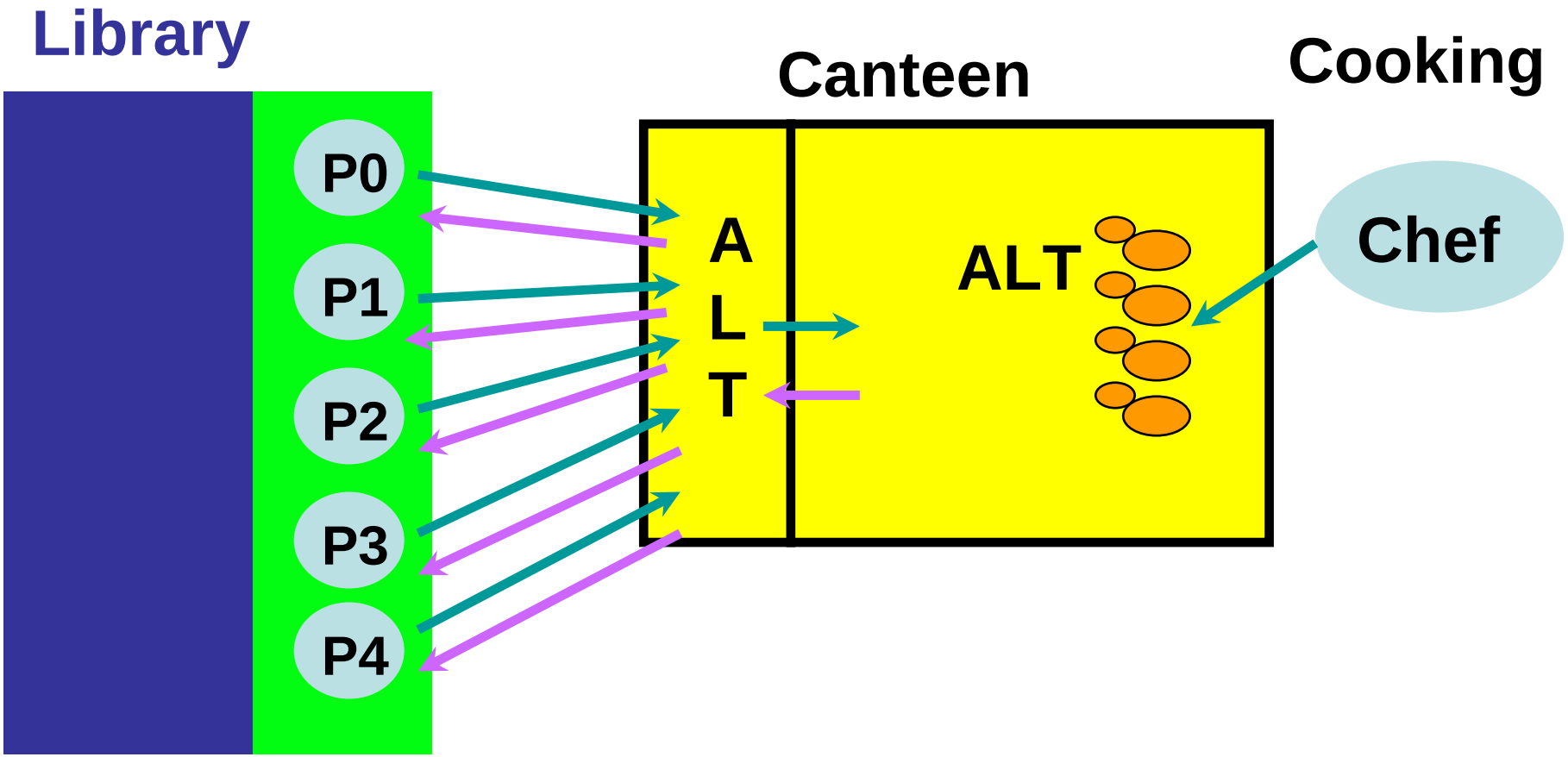
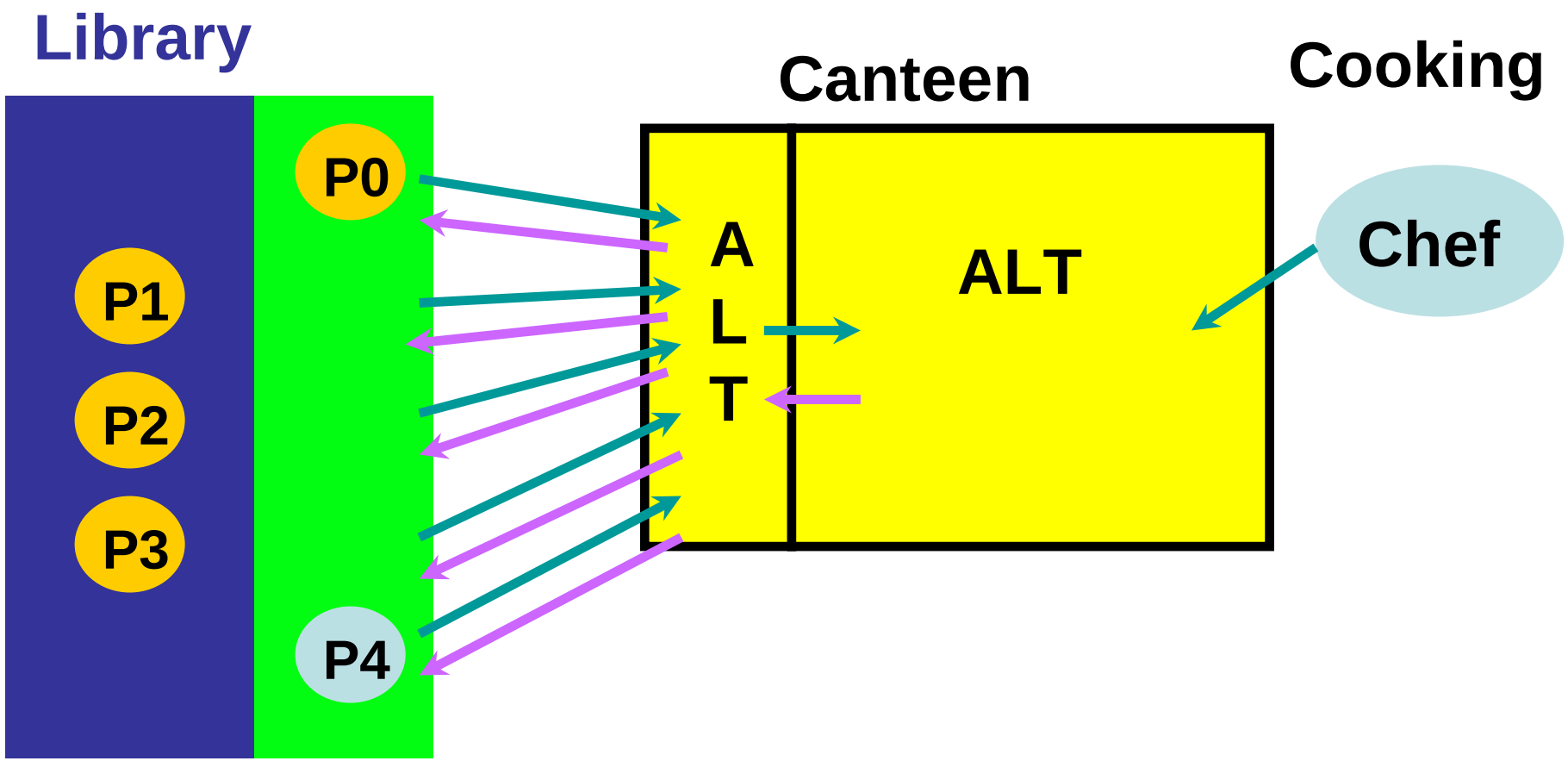
“Wot, No Chickens!”



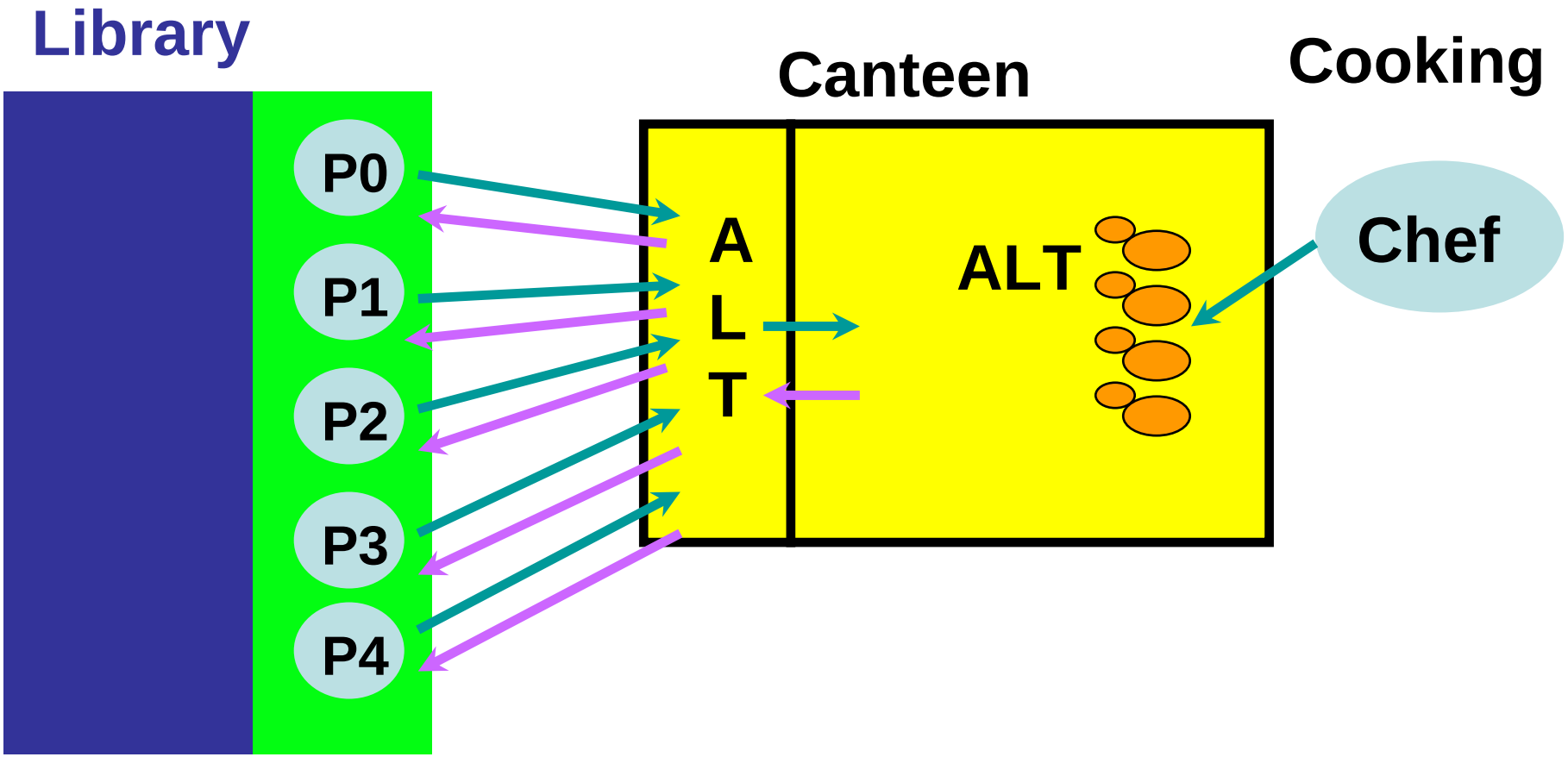
“Wot, No Chickens!”



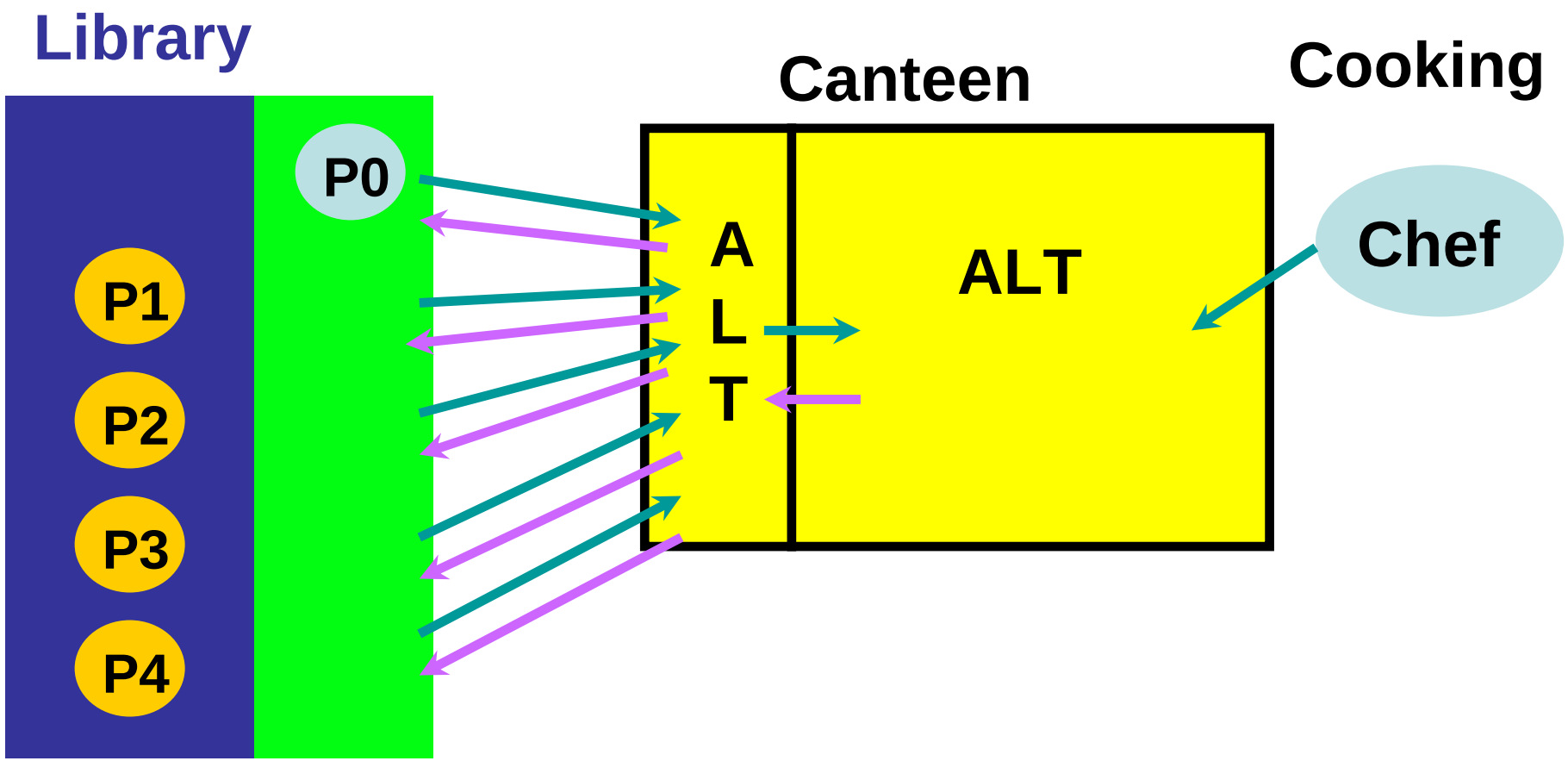
“Wot, No Chickens!”



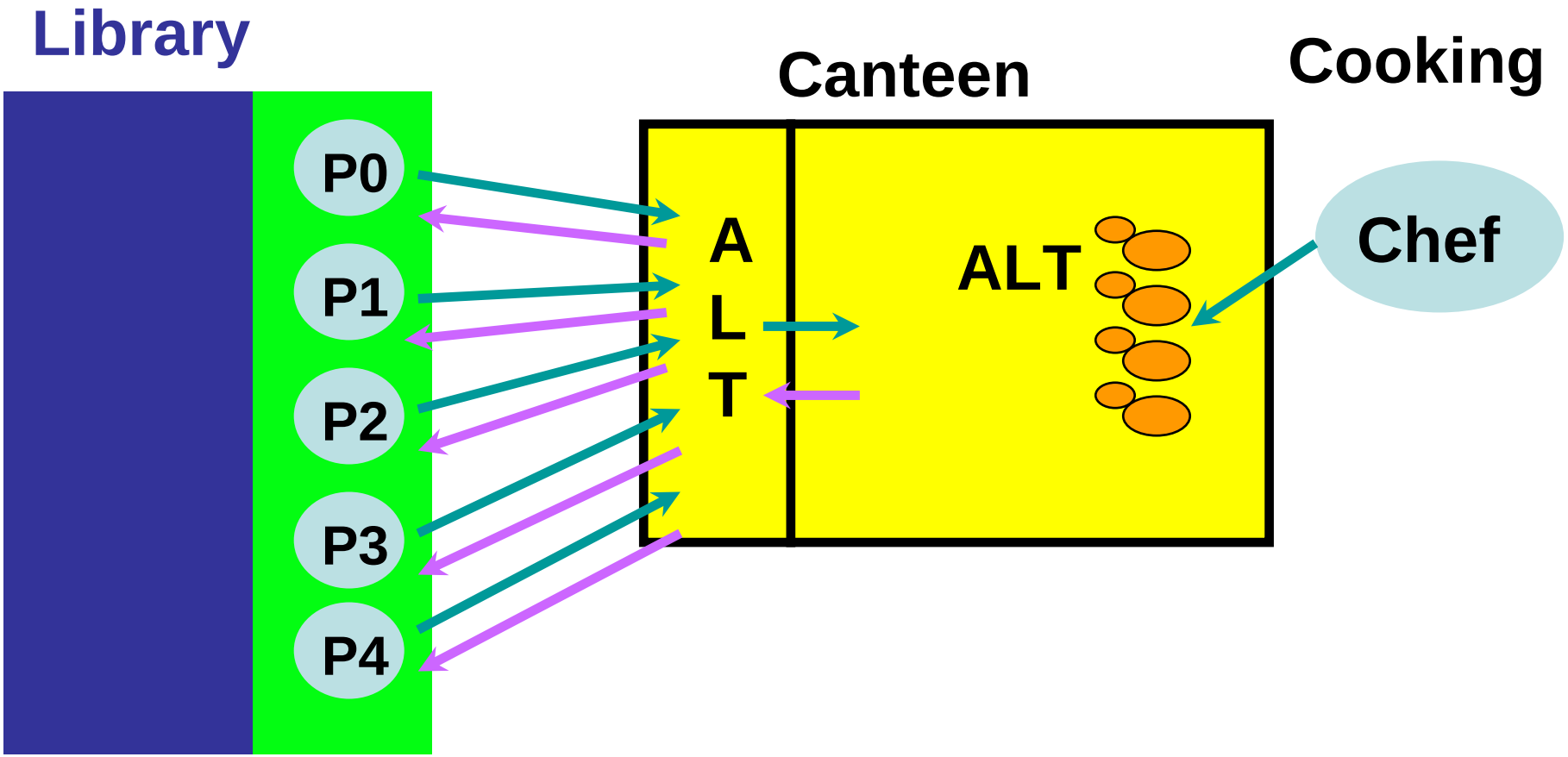
“Wot, No Chickens!”



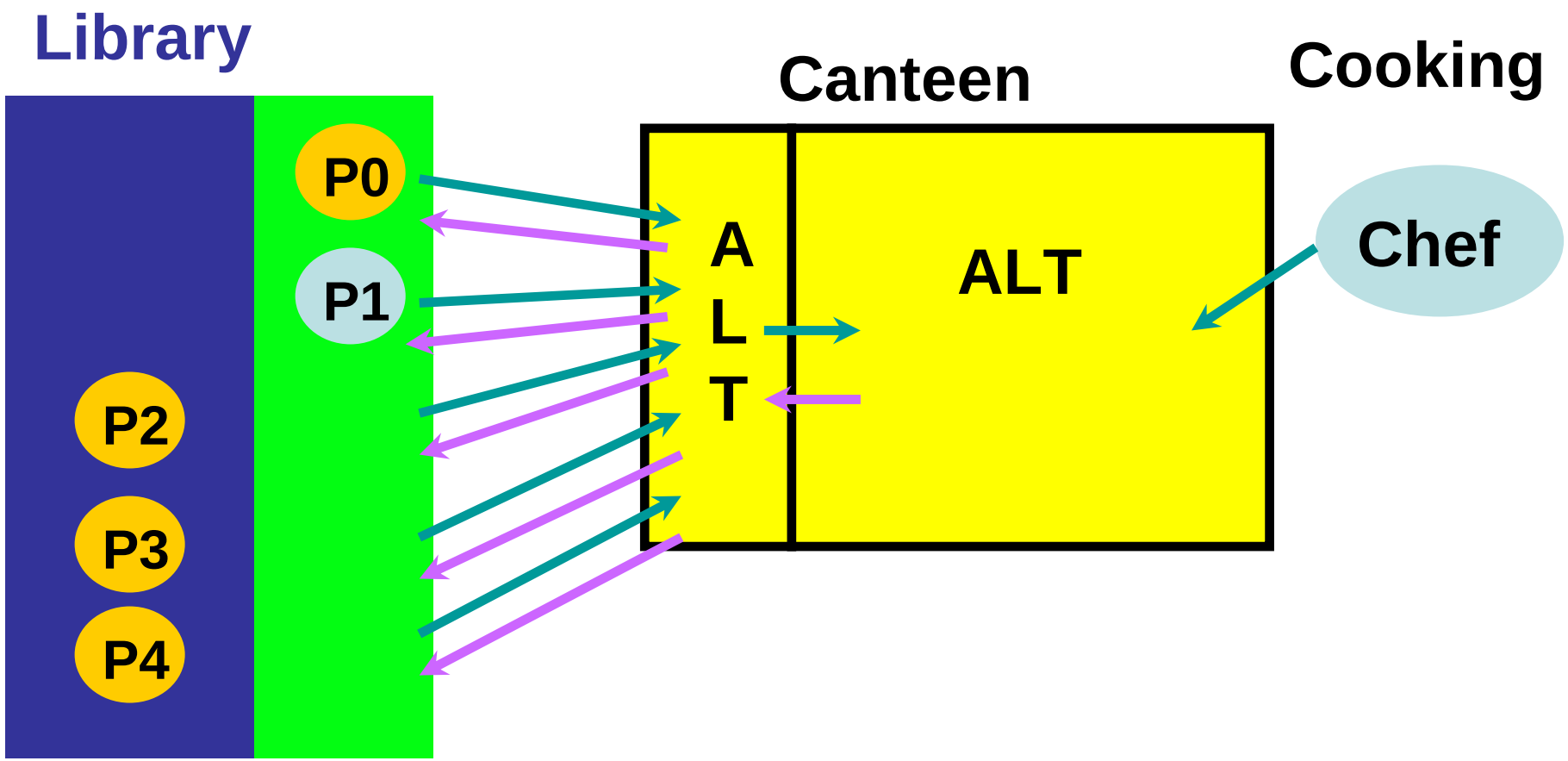
“Wot, No Chickens!”



“Wot, No Chickens!”



“Wot, No Chickens!”



```
class Server implements CSProcess{
  private AltingChannelInputInt  in[]  = null;
  private ChannelOutputInt      out[]  = null;
  private ChannelOutputInt toService   = null;
  private ChannelInputInt  fromService = null;

  public Server(AltingChannelInputInt  in[],
                ChannelOutputInt      out[],
                ChannelOutputInt toService,
                ChannelInputInt  fromService){
    this.in  = in;
    this.out = out;
    this.toService   = toService;
    this.fromService = fromService;
  }
```

```
 public void run(){  // Server
    Alternative alt = new Alternative(in);
    int index;

    while(true){
      index = alt.fairSelect();
        in[index].read();
        toService.write(0);
        fromService.read();
        out[index].write(1);
    }
  }
}
```

```
class Service extends MyObject
              implements CSProcess{
  private AltingChannelInputInt  in[]  =
                  new AltingChannelInputInt[2];
  private ChannelOutputInt toServer;

  public Service(AltingChannelInputInt fromChef,
                 AltingChannelInputInt fromServer,
                 ChannelOutputInt      toServer){
    this.in[0]    = fromChef;
    this.in[1]    = fromServer;
    this.toServer = toServer;
  }
```

```
  public void run(){  // Service
    int numChicks = 0;
    int index;
    boolean ready[] = new boolean[2];
    Alternative alt = new Alternative(in);
    ready[0] = true;

    while(true){
      ready[1] = (numChicks > 0);
      index = alt.select(ready);
      if(index == 0){
        numChicks = in[0].read();
        nap(3000);
        in[0].read();
      }
      else{
        in[1].read();
        numChicks--;
        toServer.write(1);
} } } }
```

```
class Canteen extends MyObject
              implements CSProcess{
 private AltingChannelInputInt  fromPhil[] = null;
 private ChannelOutputInt       toPhil[]   = null;
 private AltingChannelInputInt  fromChef   = null;

 public Canteen(AltingChannelInputInt fromPhil[],
                ChannelOutputInt      toPhil[],
                AltingChannelInputInt fromChef){
   this.fromPhil = fromPhil;
   this.toPhil   = toPhil;
   this.fromChef = fromChef;
 }
```

```
public void run(){ // Canteen
  One2OneChannelInt ServertoService =
                              new One2OneChannelInt();
  One2OneChannelInt ServicetoServer =
                              new One2OneChannelInt();
  new Parallel( new CSProcess[]{
    new Server(fromPhil, toPhil,
               ServertoService, ServicetoServer),
    new Service(fromChef,
                ServertoService, ServicetoServer)
   }).run();
}
```

```
class Chef extends MyObject implements CSProcess{
  private ChannelOutputInt toCanteen   = null;

  public Chef(ChannelOutputInt toCanteen){
    this.toCanteen   = toCanteen;
  }

  public void run(){
    int numChicks = 4;
    while(true){
      nap(2000);                    // Cooking
      toCanteen.write(numChicks);// Delivering
      toCanteen.write(0);           // 4 chickens up
    }
  }
}
```

```
class Phil extends MyObject implements CSProcess{

  private int id;
  private ChannelOutputInt toCanteen   = null;
  private ChannelInputInt  fromCanteen = null;

  public Phil (int id,
               ChannelOutputInt toCanteen,
               ChannelInputInt fromCanteen) {
    this.id = id;
    this.toCanteen   = toCanteen;
    this.fromCanteen = fromCanteen;
  }
```

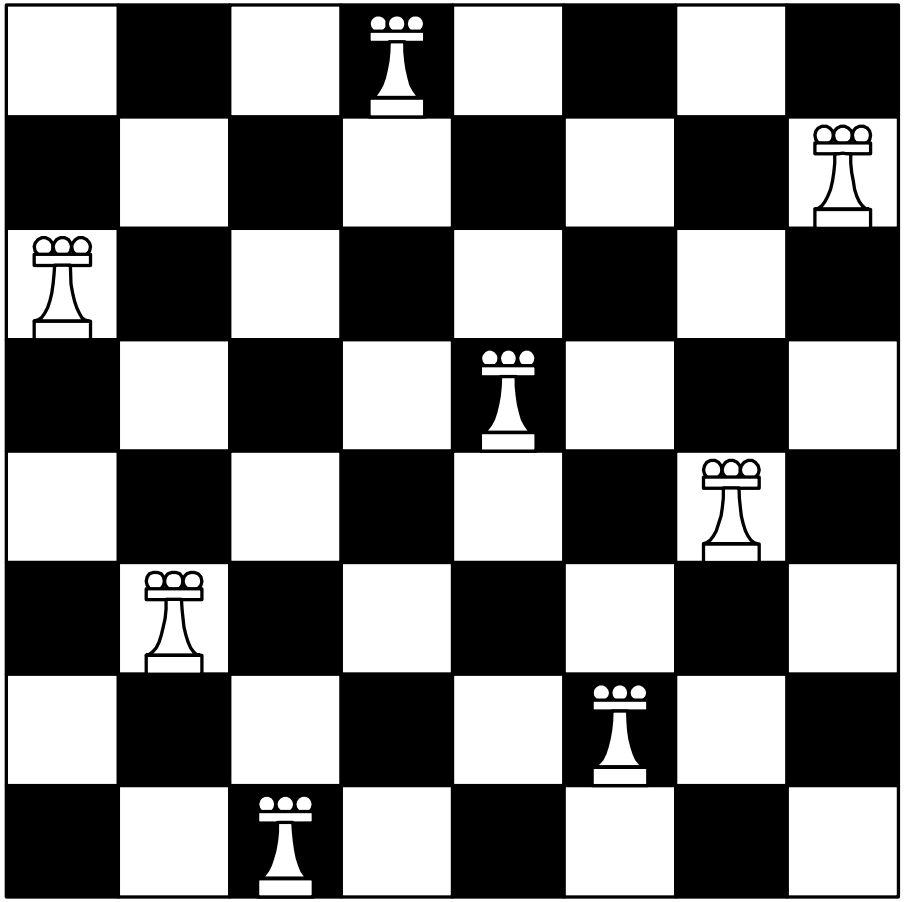
```
public void run () {  // Phil
  int chicken;

  while (true) {
    if (id > 0) {
      nap(3000);                    // Thinking
    }
    toCanteen.write(id);            // Gotta Eat
    chicken = fromCanteen.read();// Mmmm...
   }                                // that's good
 }
}
```

```
class CollegeCircuit implements CSProcess{

  public void run(){
    One2OneChannelInt PhiltoCanteen[] = new One2OneChannelInt[5];
    One2OneChannelInt CanteentoPhil[] = new One2OneChannelInt[5];
    One2OneChannelInt CheftoCanteen = new ONe2OneChannelInt;

    Phil phillist[] = new Phil[5];

    for(int k = 0; k < 5; k++){
      PhiltoCanteen[k] = new One2OneChannelInt();
      CanteentoPhil[k] = new One2OneChannelInt();
    }

    for(int k = 0; k < 5; k++)
      phillist[k] = new Phil(k, PhiltoCanteen[k],
                                CanteentoPhil[k]);

    new Parallel( new CSProcess[]{
      new Parallel(phillist),
      new Chef(CheftoCanteen),
      new Canteen(PhiltoCanteen, CanteentoPhil, CheftoCanteen)
    }).run();
} }
```

Bag-of-tasks programming

- When a problem can easily be split into independent sub-tasks
- Concurrent versions are very easy
  - But you can run into load-imbalance if your task size is too big

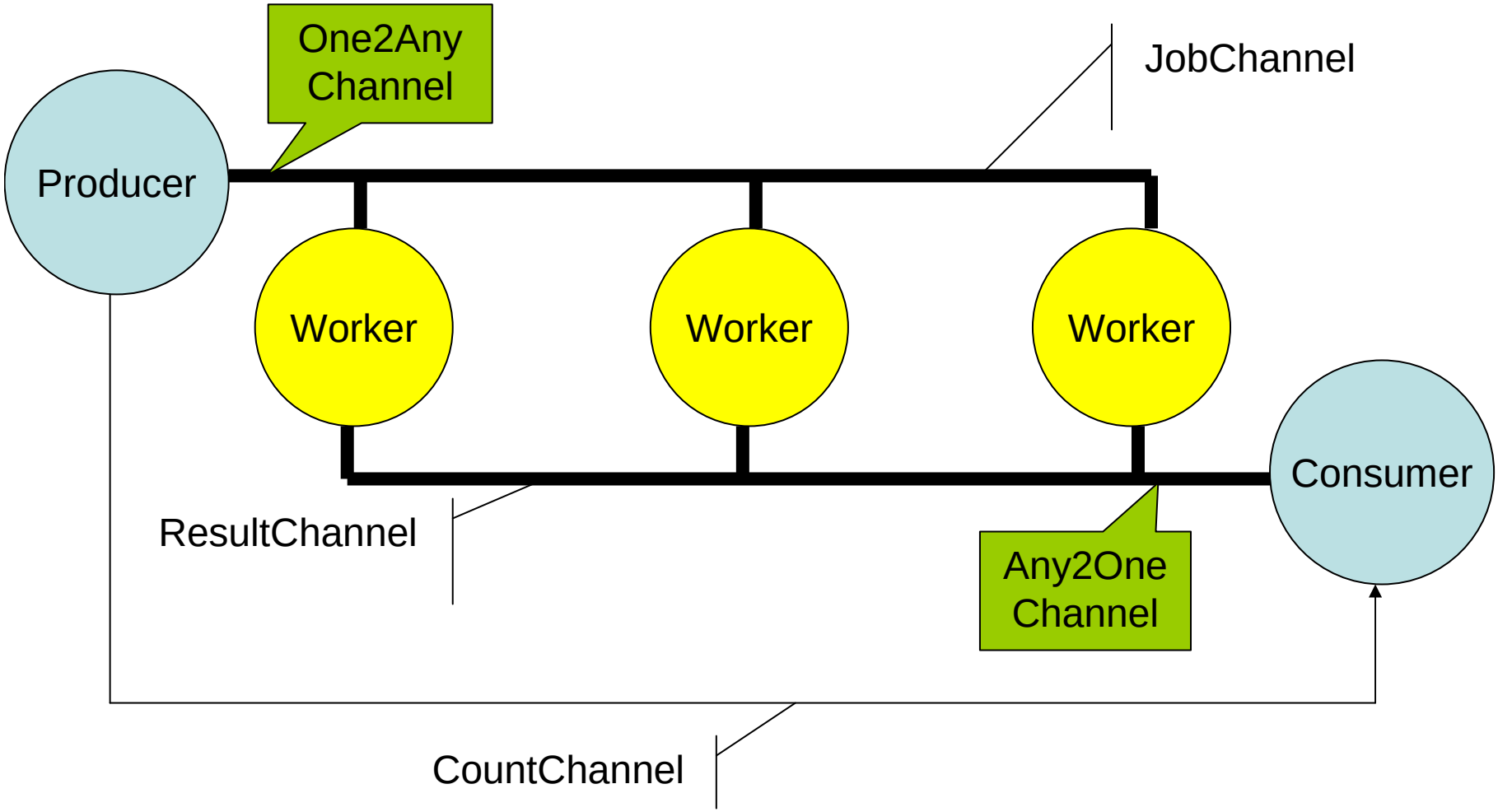
N-Queens Problem



Approach

- Generate a set of subtrees
- Put each subtree in a bag
  - Or simply make a list – it does not take long
- Have workers take tasks from the bag
  - And place results in another bag
- Sum the results
  - Or just let the bag do the summing

N-Queens example



Implementation

- **Master**

```
pool=[]
for i in range(n):
  for j in range(n):
    if(board(i,j)):
      pool.append([i,j])
CountChannel.Write(len(pool))
for x in pool:
  JobChannel.write(x)
```

Implementation

- **Worker**

```
while(true):
  coor=JobChannel.read()
  valid=checkboard(i,j)
  ResultChannel.write(valid)
```

Implementation

- **Consumer**

```
n = CountChannel.read()
sum = 0
for i in range(n):
  sum = sum + ResultChannel.read()
print sum
```

Two concurrent webservers

- Two basic designs for a concurrent webserver
  - High throughput webserver
  - High performance webserver

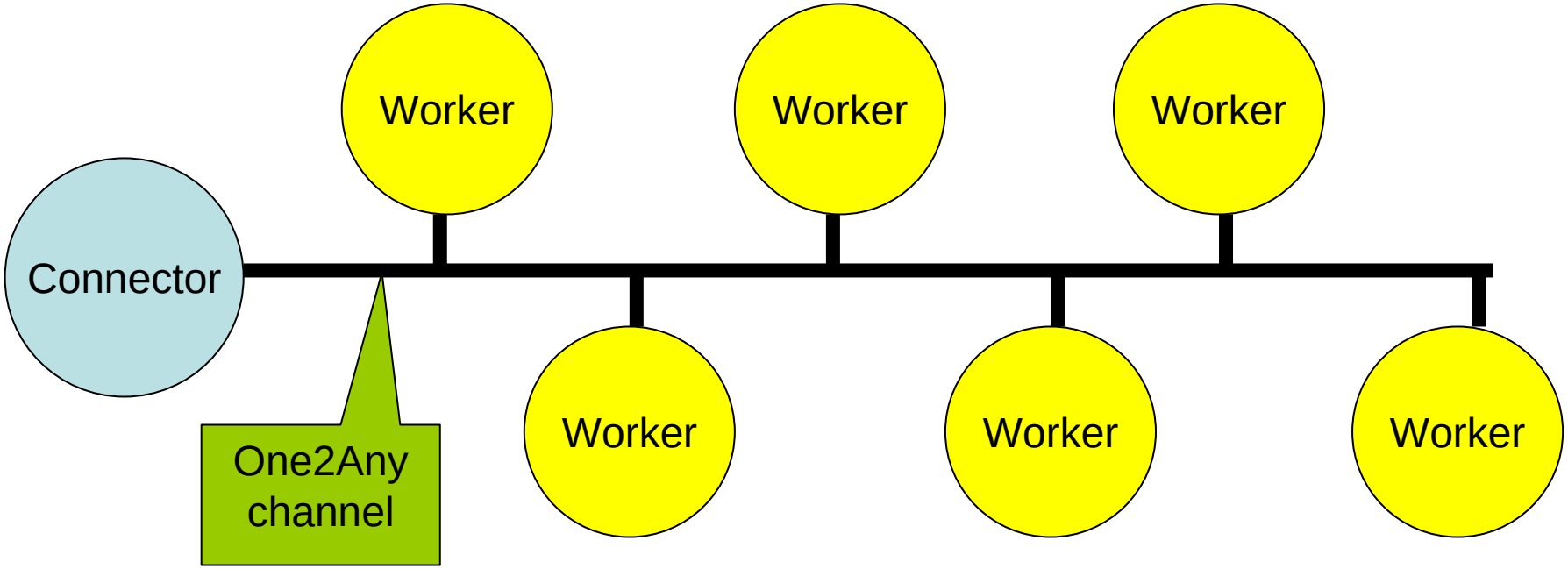
High throughput webserver

- We desire a webserver that can handle many conconcurrent sessions
- Each request is an operation in its own right
  - So each request can be implemented as a process in it’s own right
  - This is the same approach that is used in common web-servers

High throughput webserver

- Two basic approaches to loadbalancing
  - Use a semaphore to protect the listen/accept calls
  - Have one thread do all listen/accept calls and the pass the new socket to a worker thread
- The latter is far preferable

High throughput webserver



Psudo Code

Master:

```
x = new socket()
bind(x, 80)
while(true):
  listen(x,1)
  y=accept(x)
  chan ! y
```

Worker:

```
while(true):
  chan ? y
  service(y)
```

High performance webserver

- We desire a webserver that can service a single webpage very quickly
- http 1.1 has support for ’keep-alive’ and pipelined requests
  - So that when parsing a html page a client can send mutible requests towards the same page
    - Mutible images i.e.

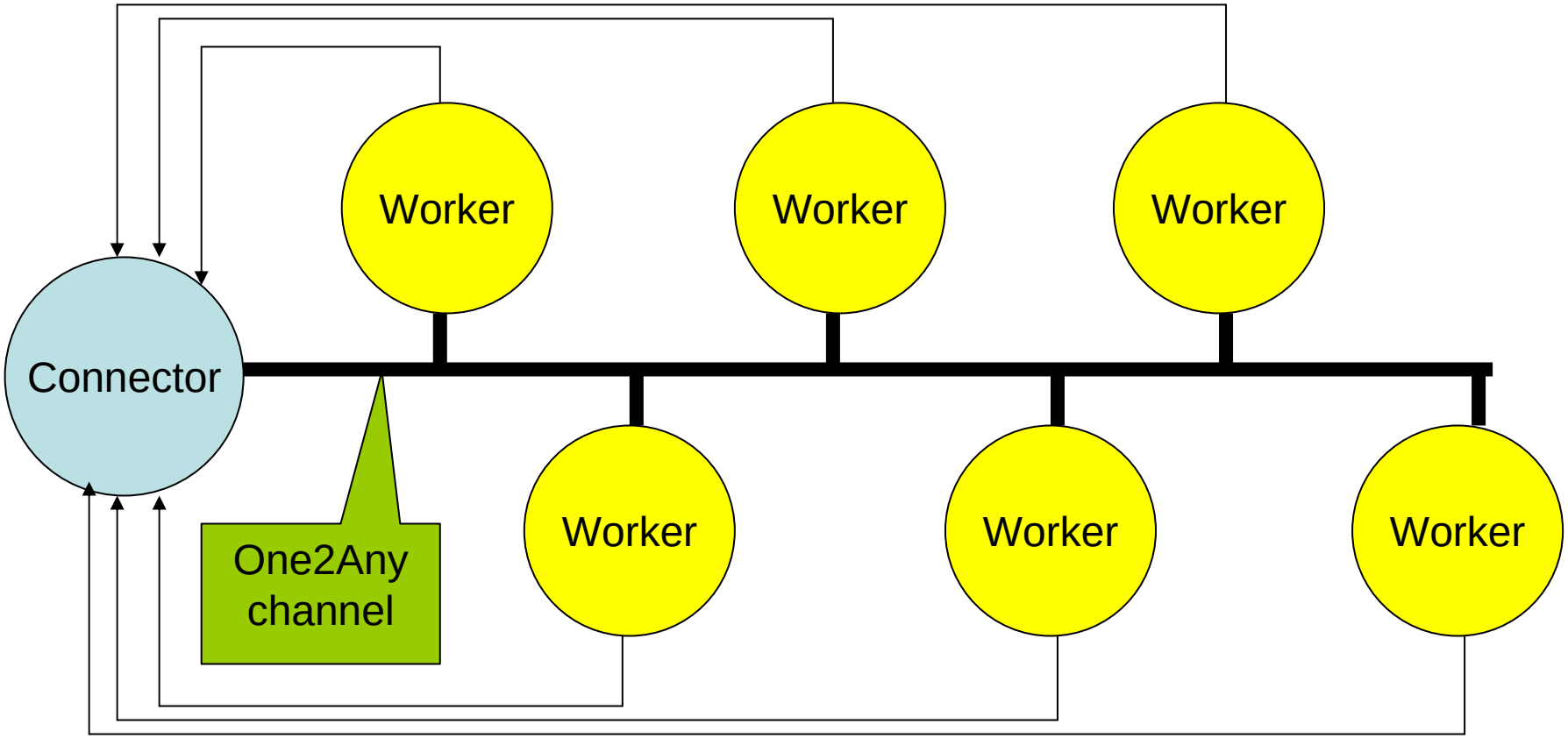
Approach

- Similar to the high throughput
  - But instead of passing sockets to the workers we pass requests
  - But the workers cannot write the answer directly to the socket any longer
- Thus the replys will have to be sent through the master

Approach

- We can use the same basic approach to distribute the requests
- But then we need each process to pass its results to the master who can then send it back

High performance webserver



Psudo Code

Master:

```
x = new socket()
bind(x, 80)
while(true):
  listen(x,1)
  y=accept(x)
  PRIALT:
    y      ? req; chan ! req
    w[0] ? ans; y ! ans
    ...
    w[n] ? ans; y ! ans
```

Worker:

```
while(true):
  chan?y
  chan ! service(y)
```