Cloudy systems

— Taking the most out of the HPC Cloud

Workshop at UvA
1st February 2017

Ander Astudillo <ander.astudillo@surfsara.nl>
Markus van Dijk <markus.vandijk@surfsara.nl>

SURF SARA

Recap: defining cloud computing

NIST
National Institute of Standards and Technology
U.S. Department of Commerce

Essential characteristics:

- On-demand **self-service**
- Broad **network** access (ubiquitous + convenient + on-demand)
- Resource **pooling**
- Rapid **elasticity** ⟹
- **Measured** service

1. Scaling
2. API

Service models:

- Software as a Service (SaaS)
- Platform as a Service (PaaS)
- Infrastructure as a Service (IaaS)

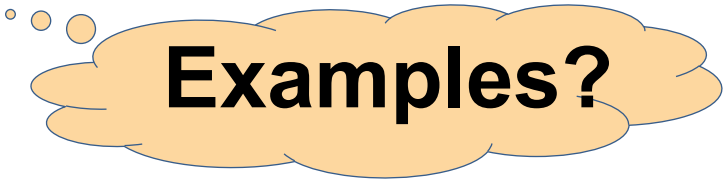
Examples?

SURF SARA

But why…?

…scaling

- Sequential run takes forever
- Not enough local resources (e.g.: memory)
- Analyse more data
- Achieve higher accuracy
- …



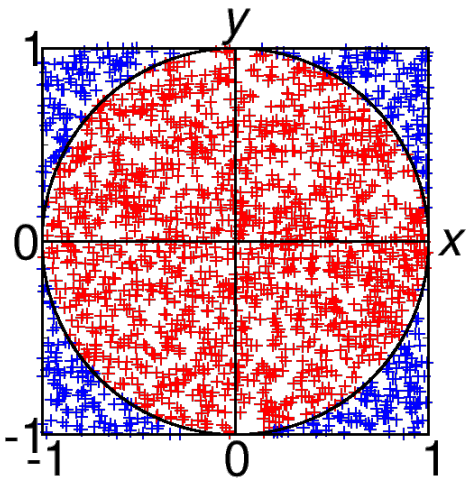
…elasticity

- Booking fixed resources in advance is:
  - A waste
  - Too expensive
  - Unpredictable
  - …

The naïve approach

Some programs are **already parallel**

- The end-user just needs to run them
- E.g.: Delft3D, XBeach, OpenFoam, Matlab…



Some problems are a matter of running the **same thing** (possibly) **with different parameters**

- You can simply run many of these runs independently at the same time on different computers
- E.g.: a Monte Carlo simulation

Embarrassingly parallel problems!

Agenda

1.- **Scaling** possibilities

2.- **API** overview

3.- Demo

SURF SARA

Scaling possibilities

1

SURF SARA

The concept (I)

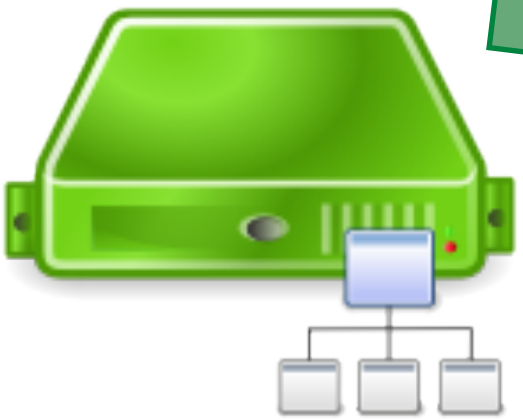
Your application

may need more…

Scale up

vs.

Scale out

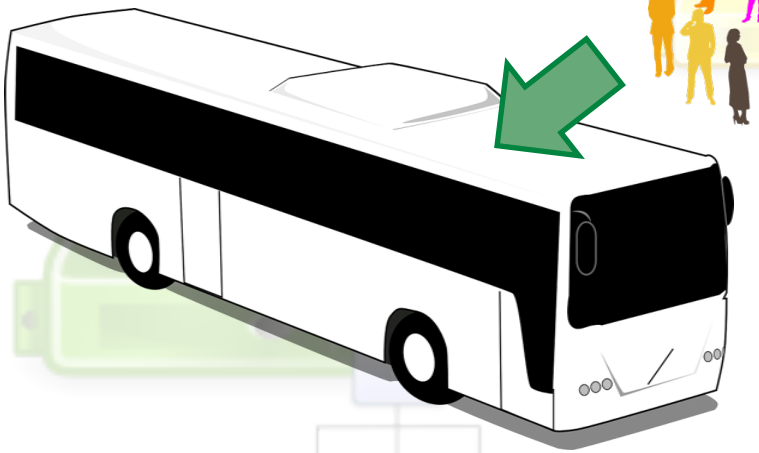


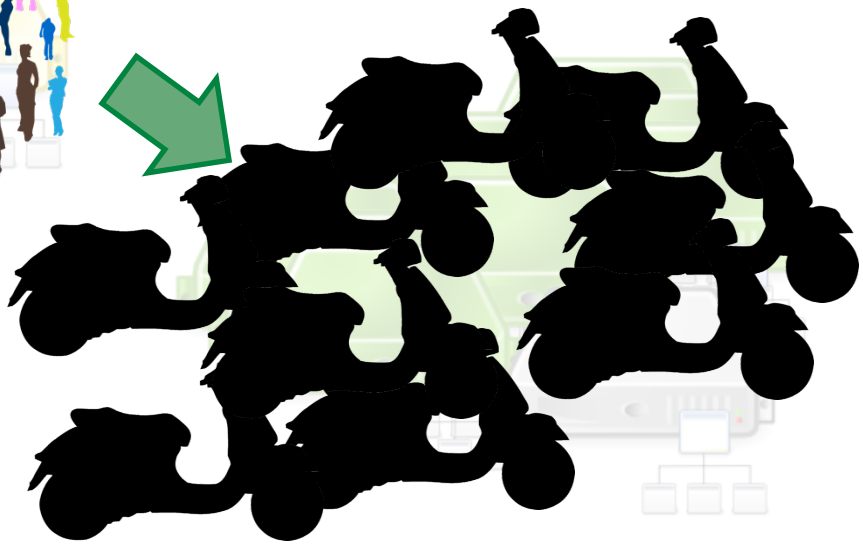
The concept (II)

e.g. transport people

may need more room…

Scale up vs. **Scale out**





The concept (and III)

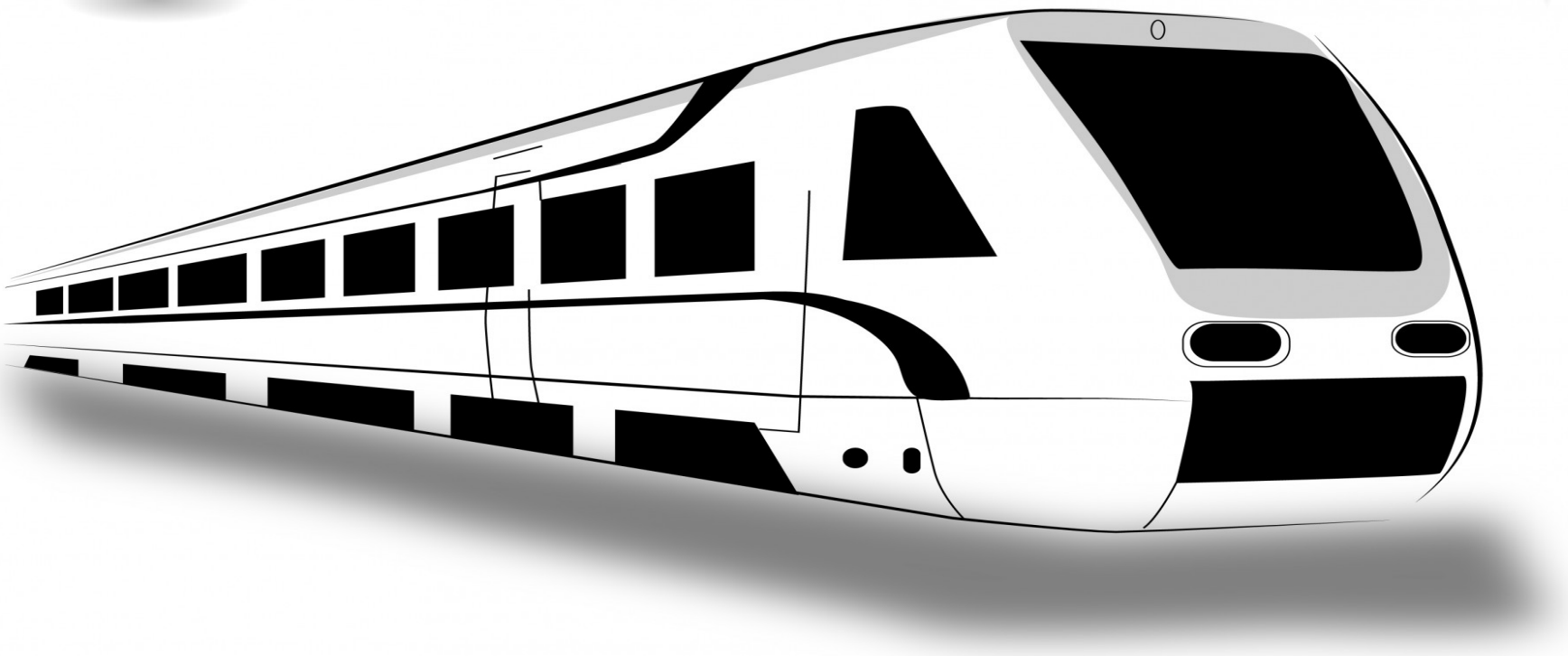
e.g. **transport people**

may need more room…

Scale up

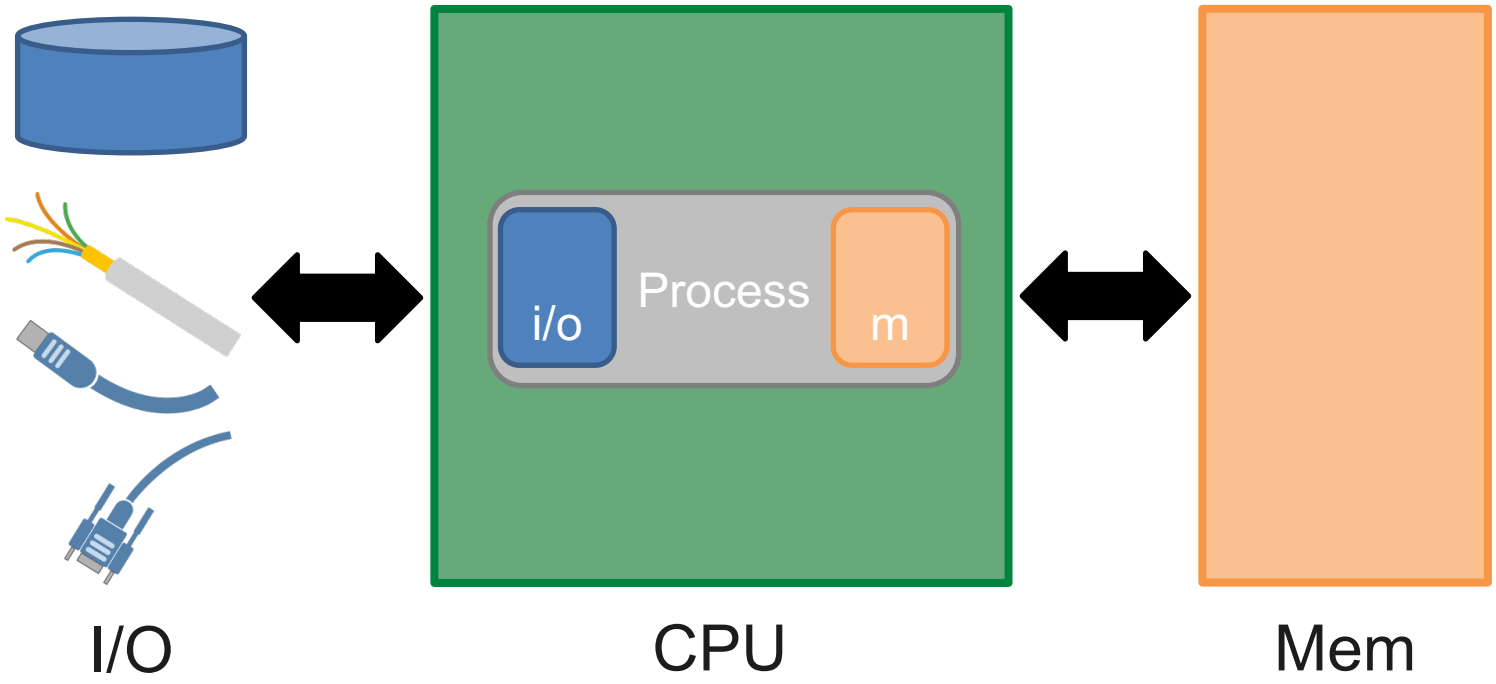
AND

Scale out



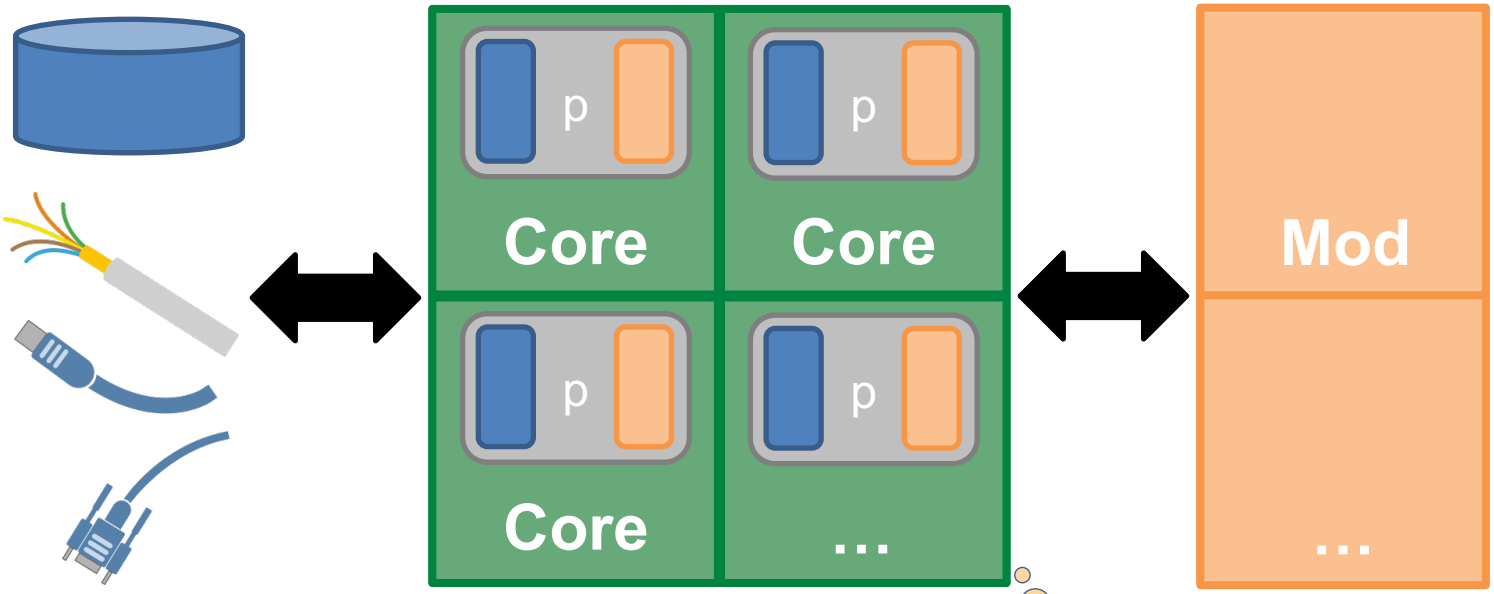
Some theory (I)

Meet: the CPU



Some theory (and II)

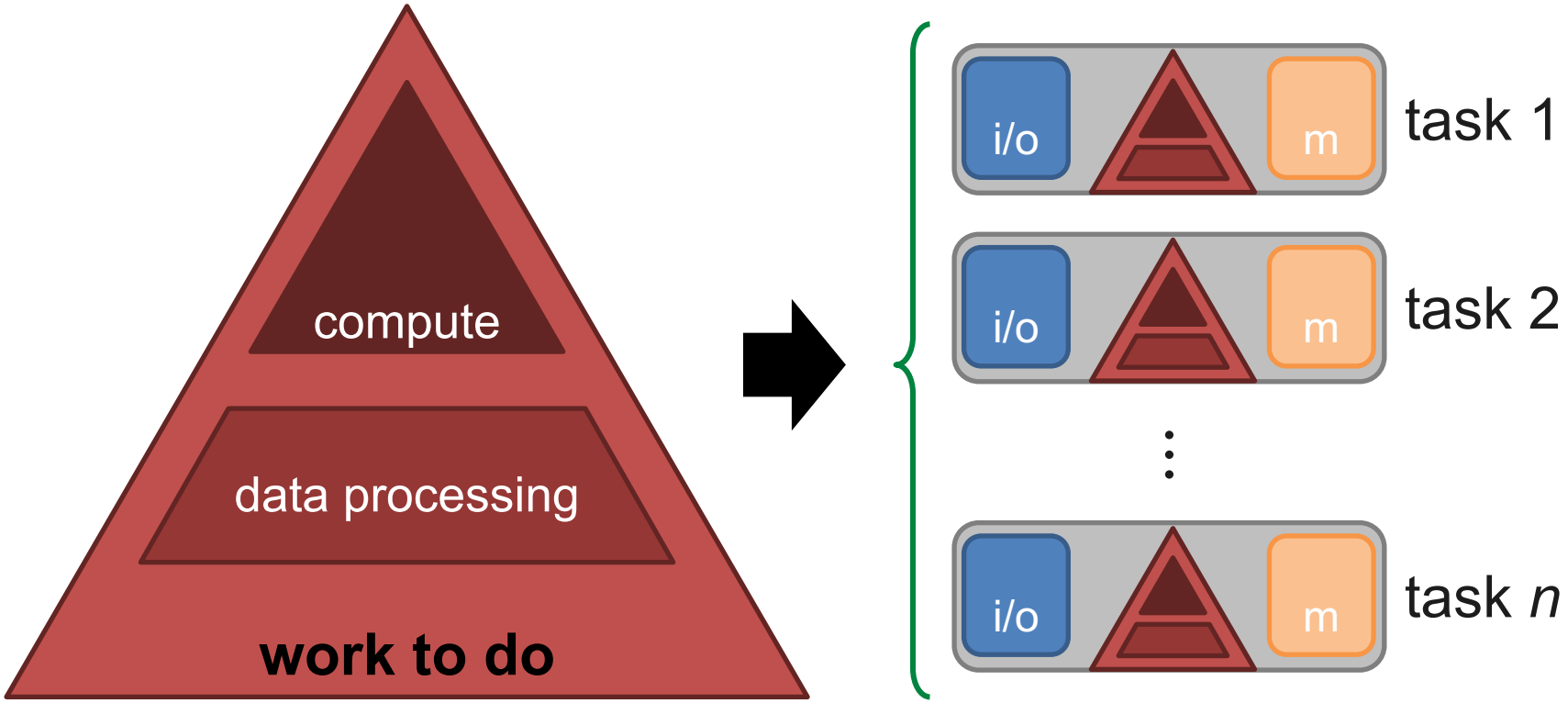
Meet: parallel processing



How does this **scale** anything?

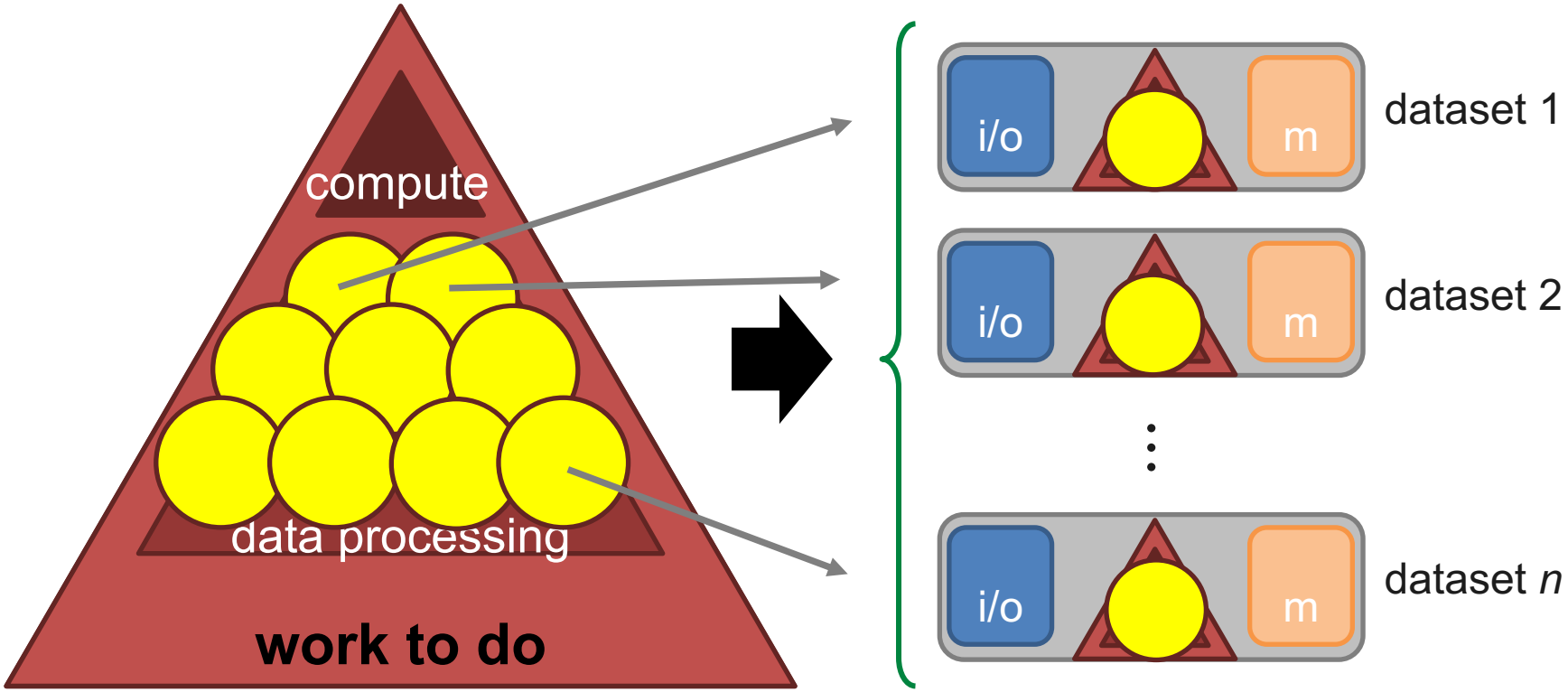
Dividing work (I)

Parallelism: task partitioning



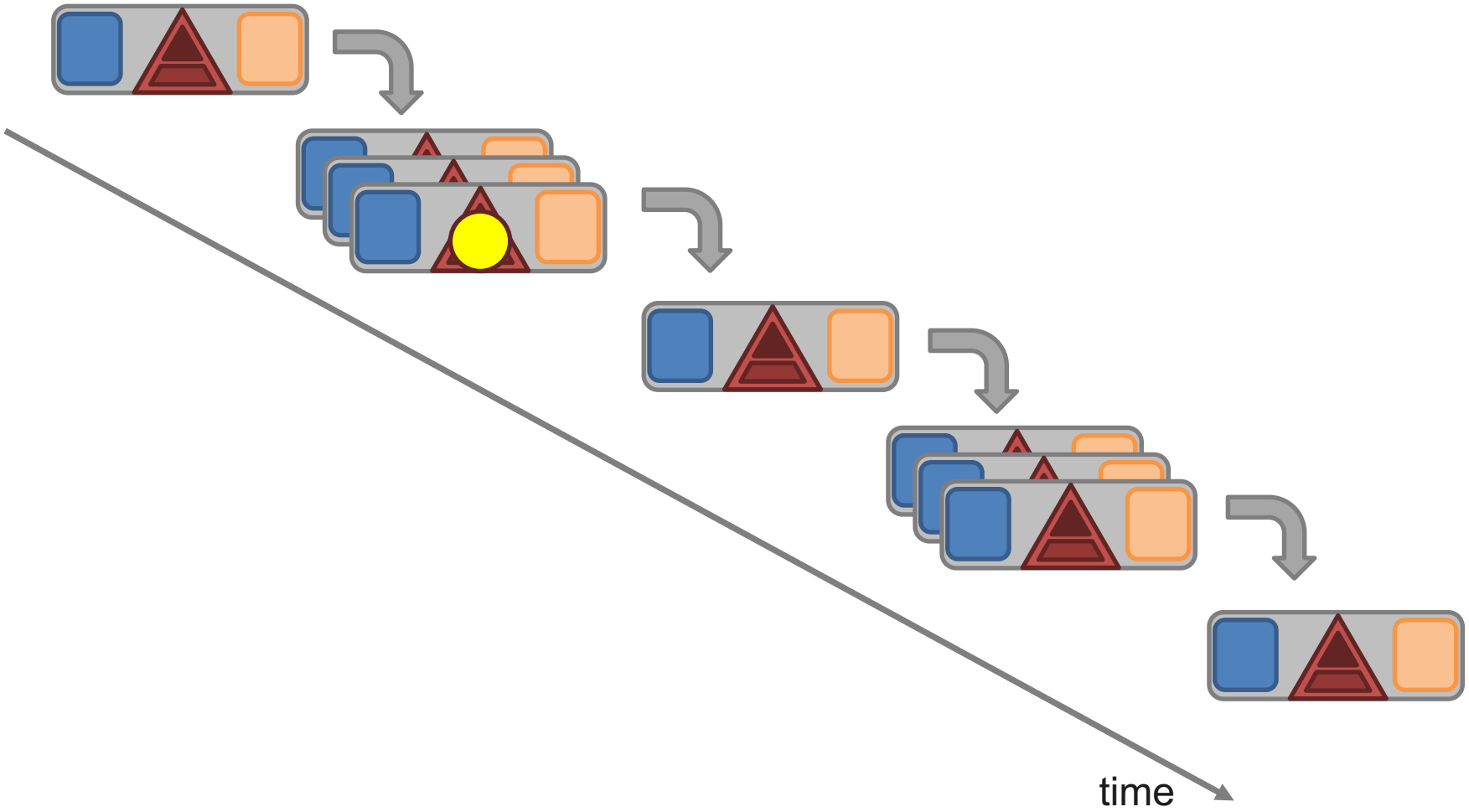
Dividing work (II)

Parallelism: data partitioning



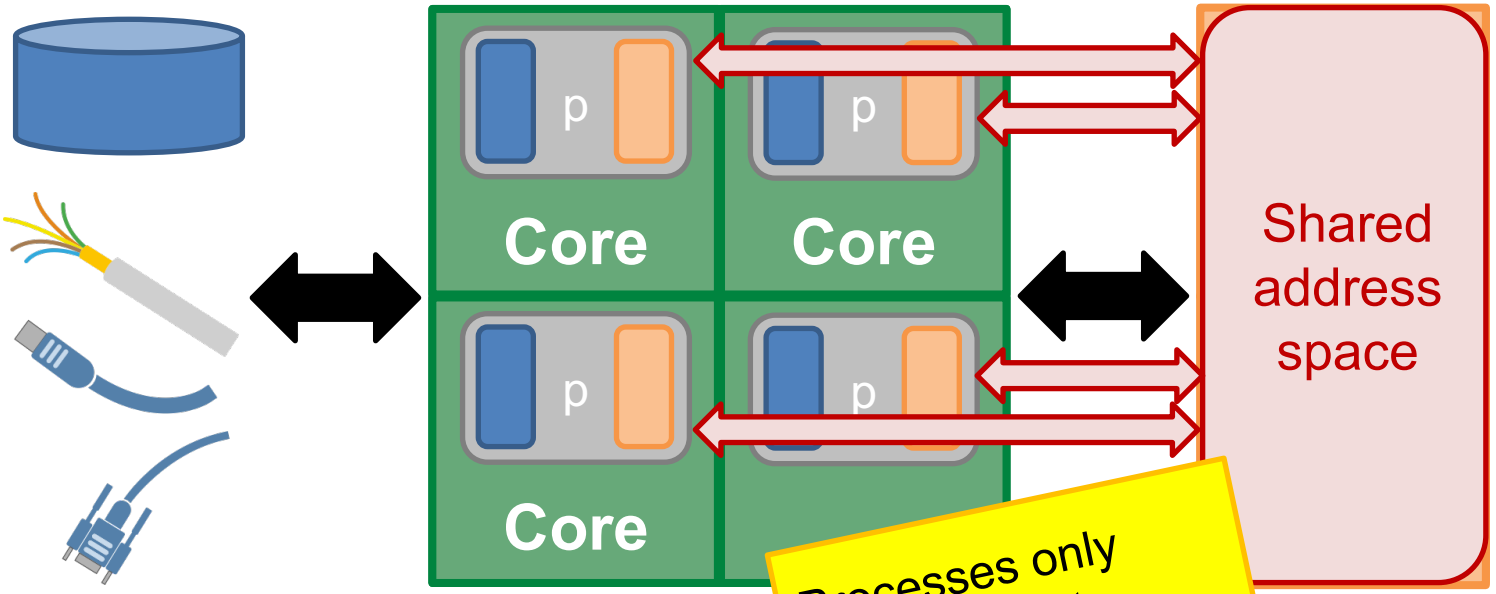
Dividing work (and III)

Example: a possible parallel program (or workflow)



Parallel programming (I)

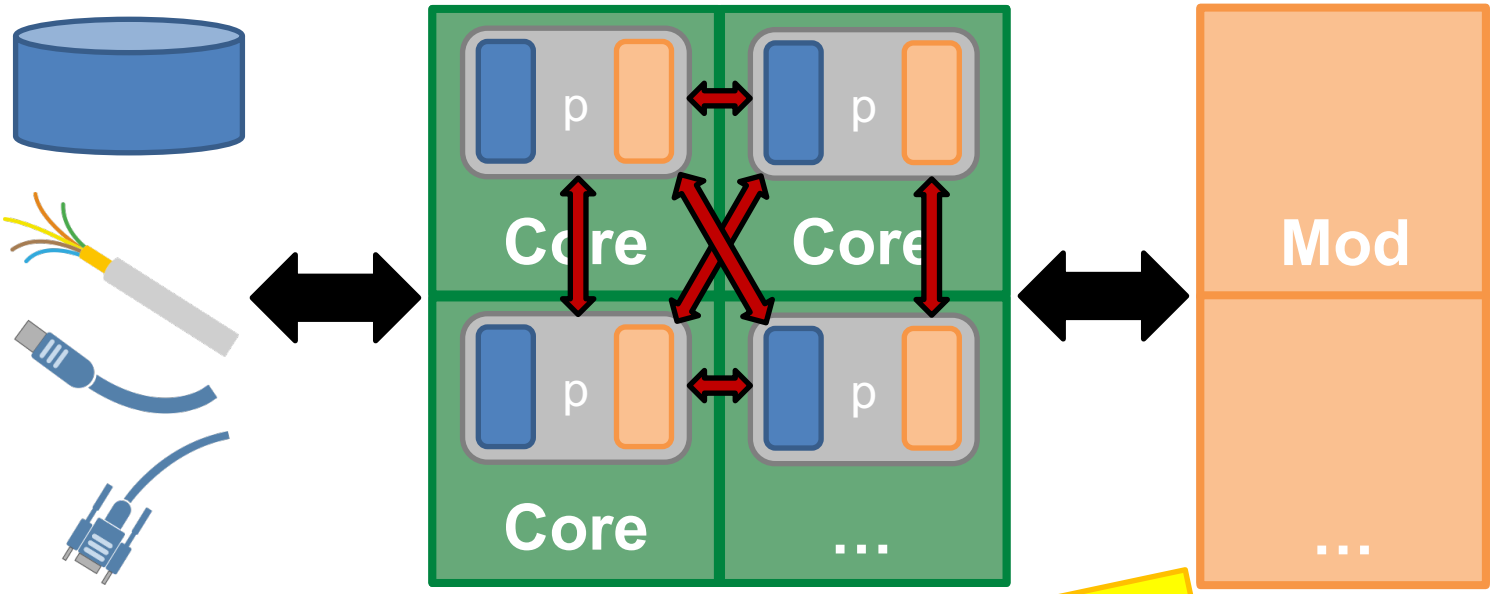
Technique: shared memory



e.g.: OpenMP

Parallel programming (II)

Technique: message-passing

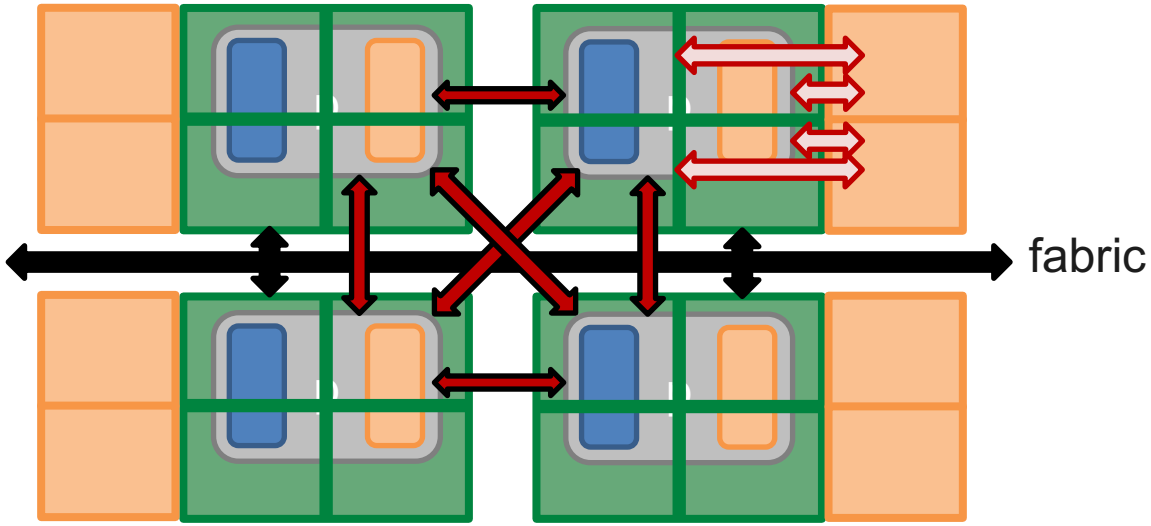


Processes can **communicate** directly

e.g.: MPI

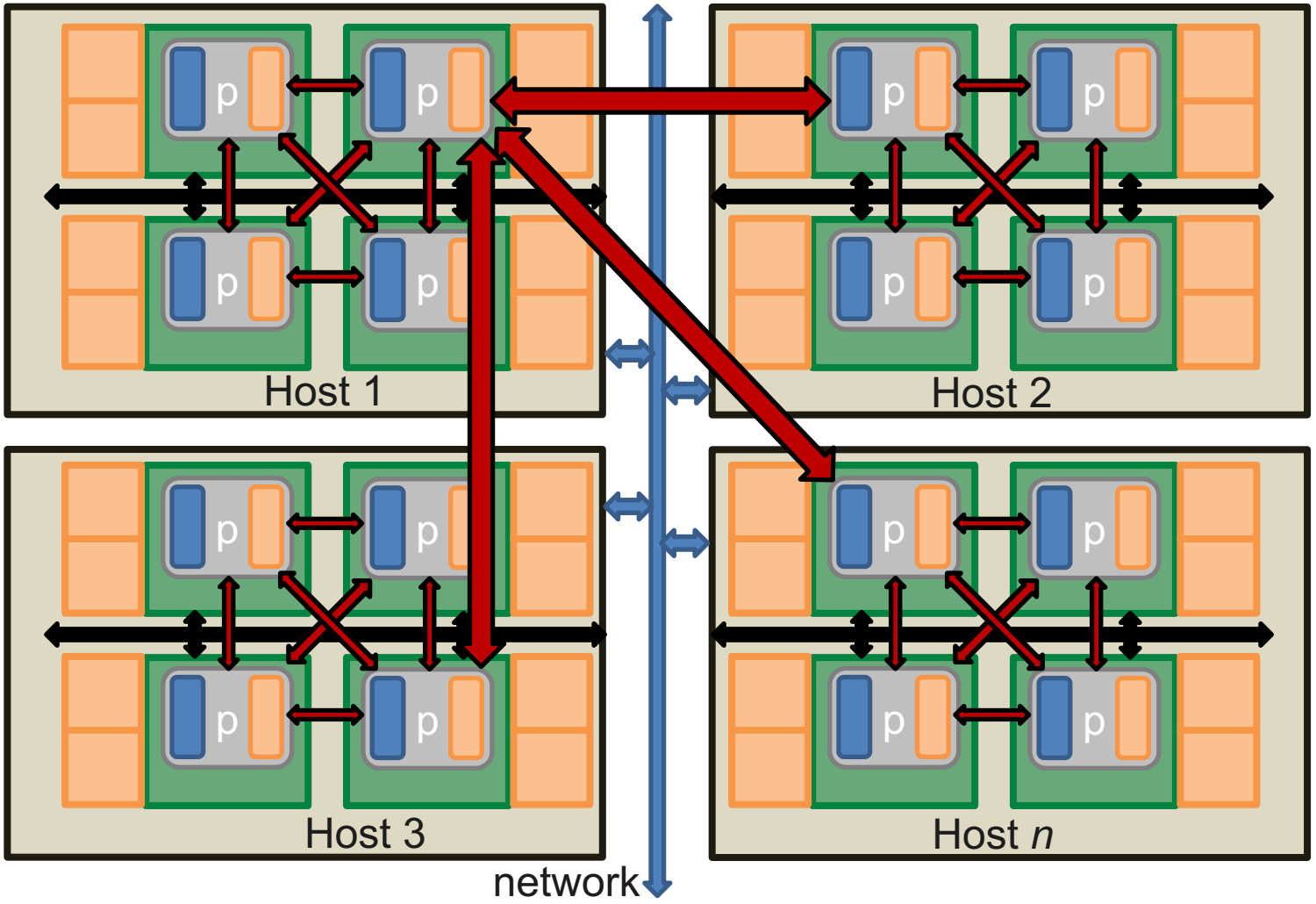
Parallel programming (III)

Combinations: shared memory and message passing

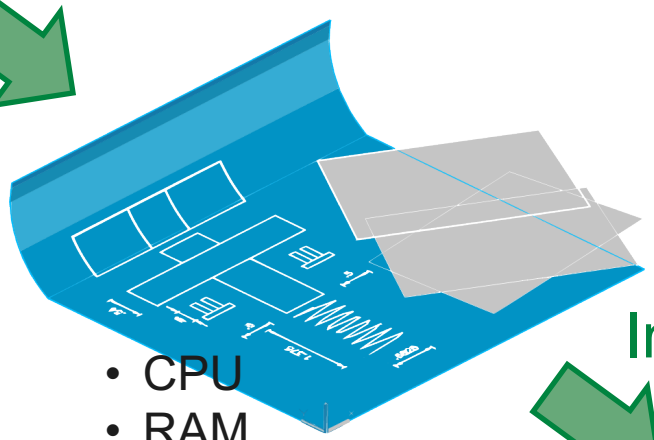


Parallel programming (and IV)

Combinations: multiple physical machines



IaaS: Your place to run VMs



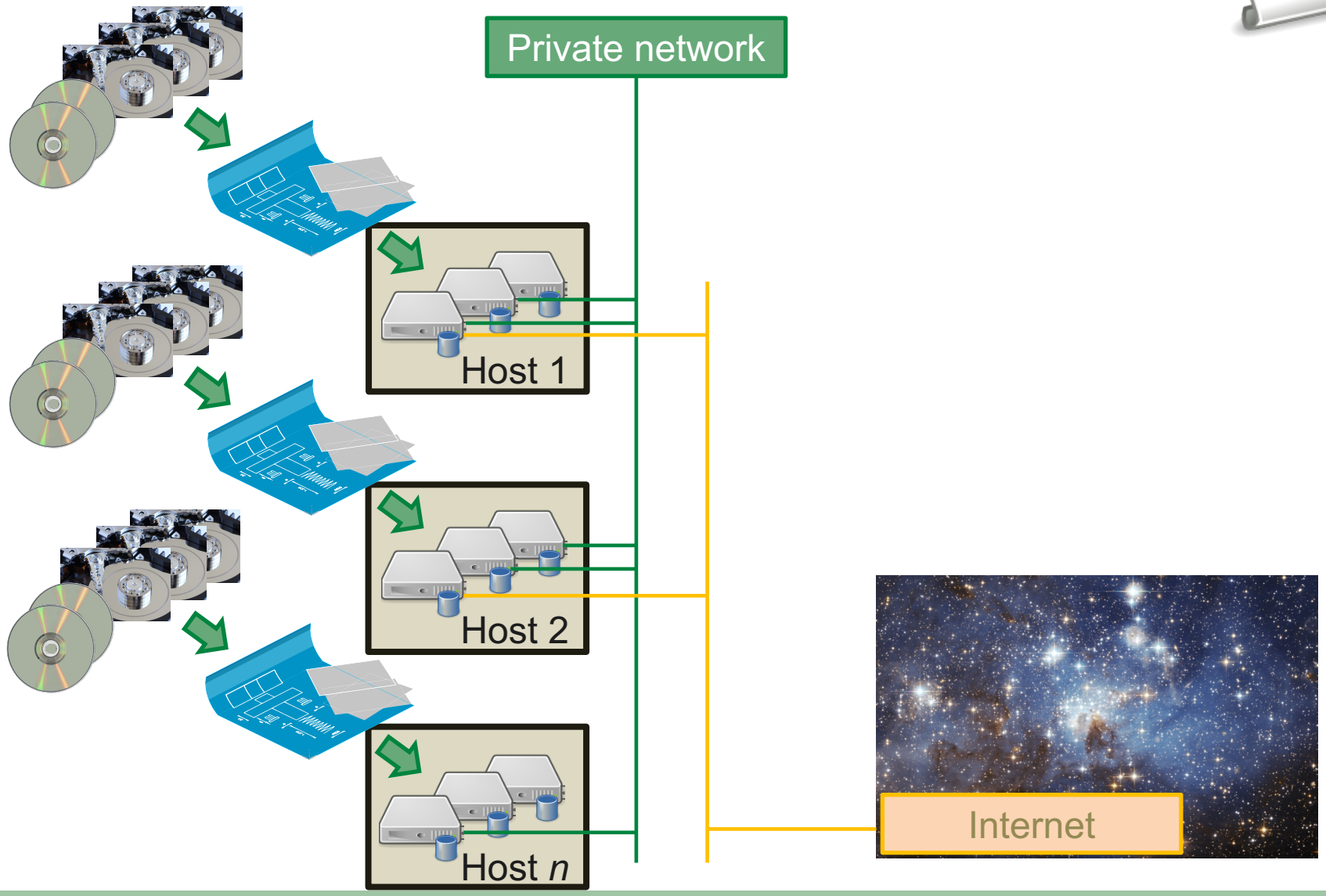
- CPU
- RAM
- I/O
  - Disks
  - Network
- …

Instantiate

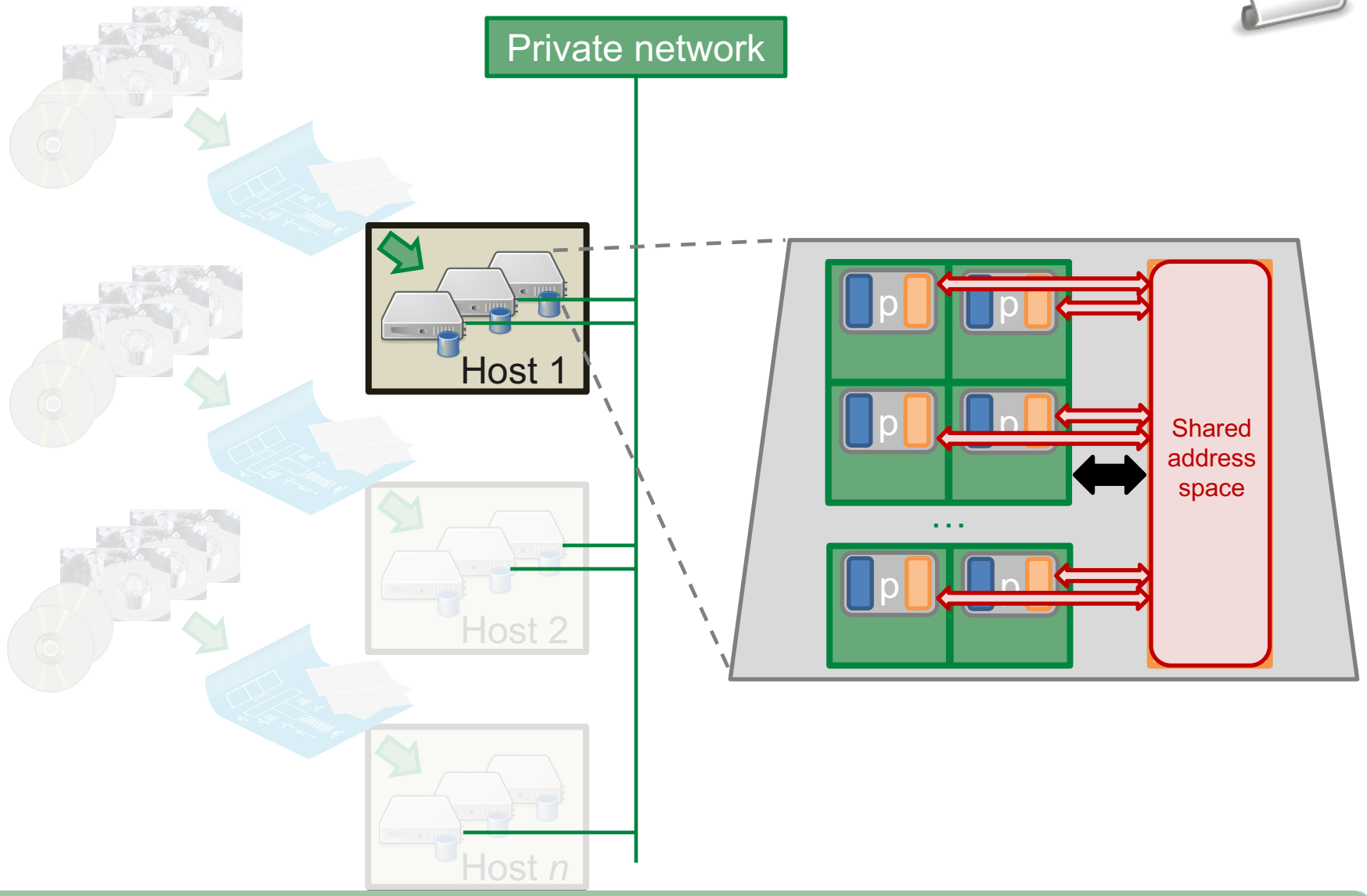
- Data store
- Persistency
- …

Images **Template** **VMs**

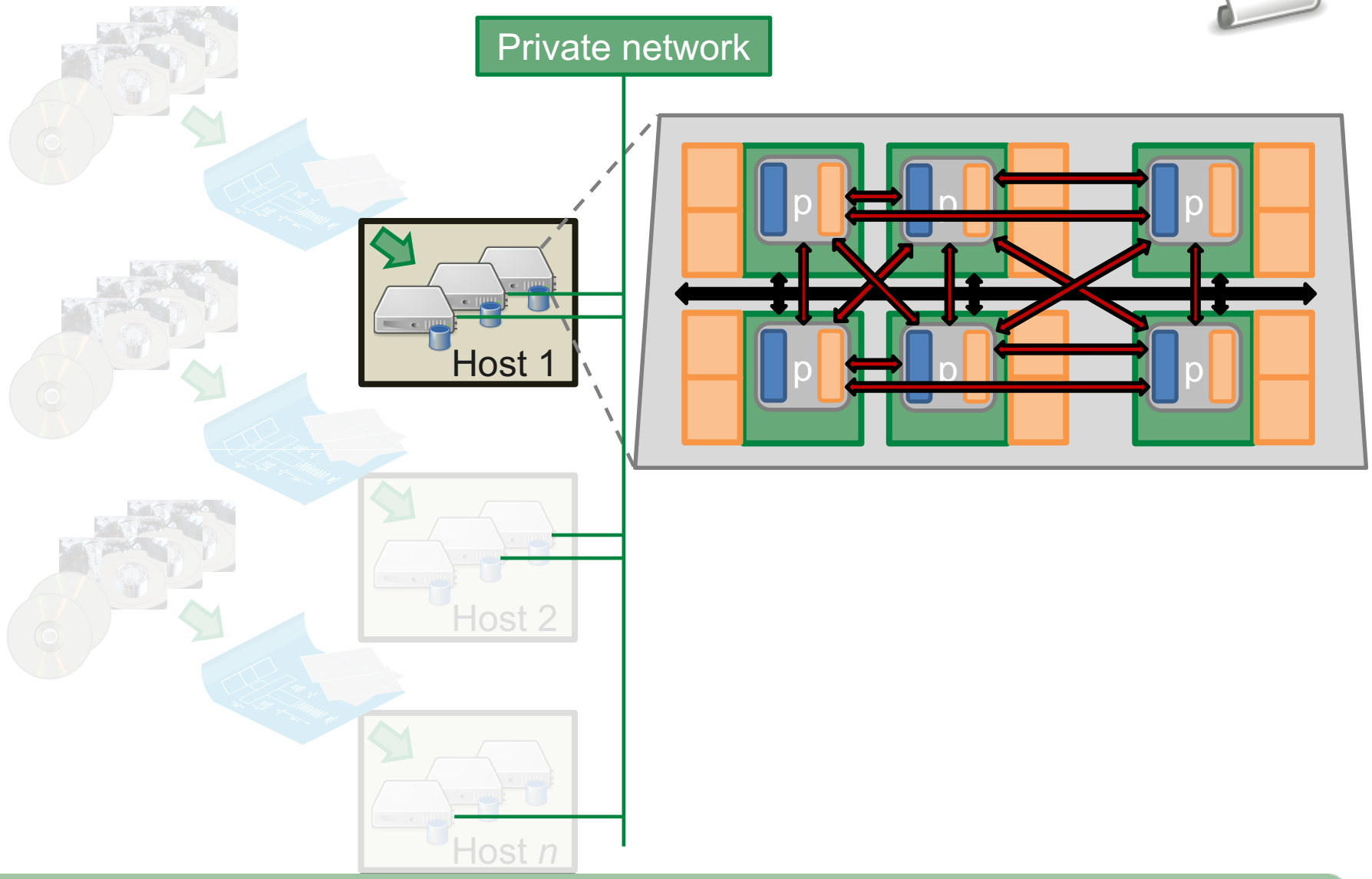
IaaS: your interconnected VMs



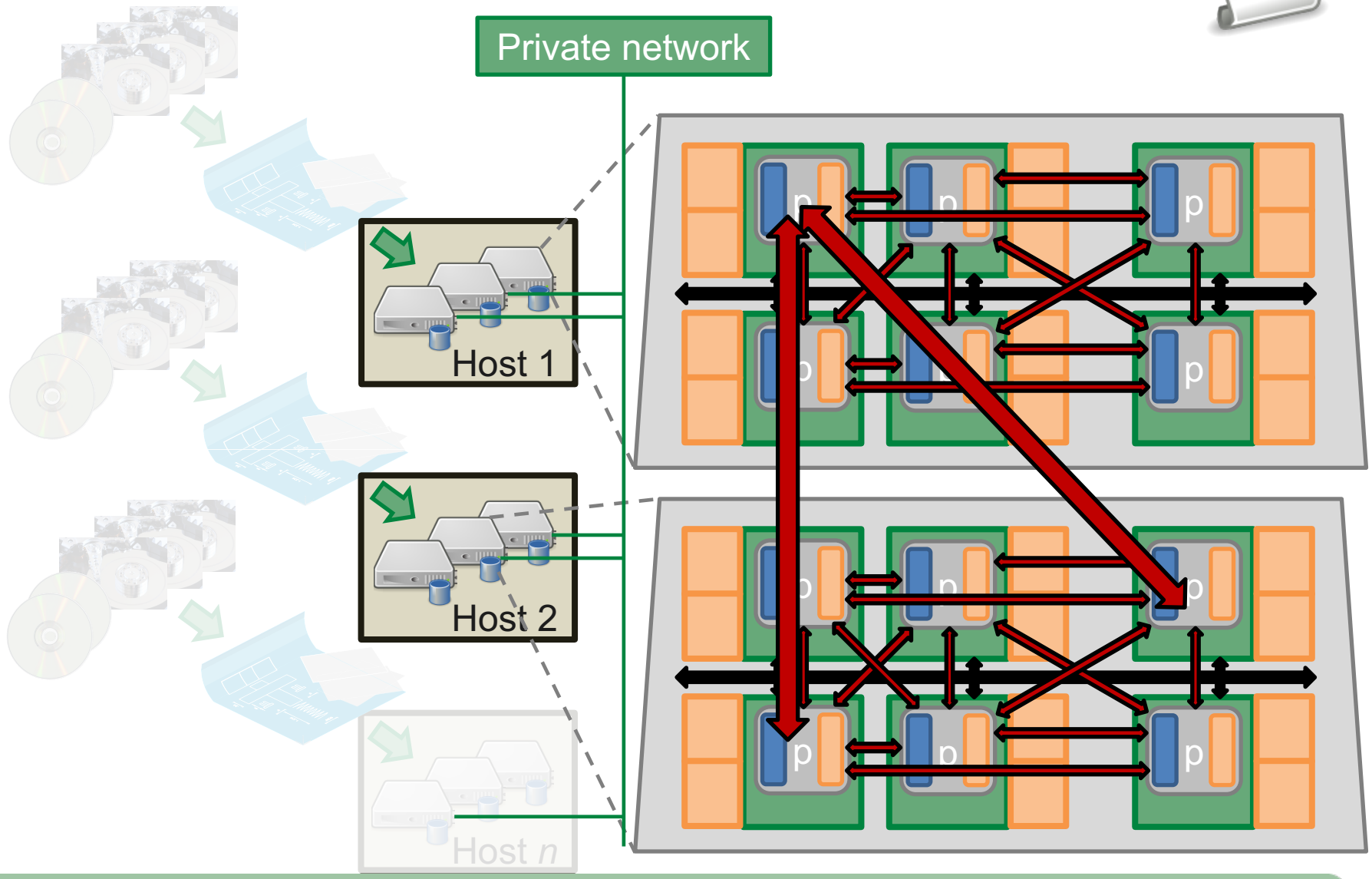
Example: OpenMP



Example: MPI



Example: MPI



Some thoughts

Parallel programming can be tricky:

- Need to know your **algorithm**
- Need to know your **data**
- Need to know your **architecture**

Try to **optimise:**

- Identify sequential **bottlenecks**
- Strive for **data** locality
- Identify **latencies**
- Minimise **communication**
- Be wary of **concurrency:**
  - **Deadlocks**
  - **Race conditions**
- Prepare for **failures**: machines, networks, timeouts…

So… you may as well be better off using a **naïve approach**! ☺

API overview

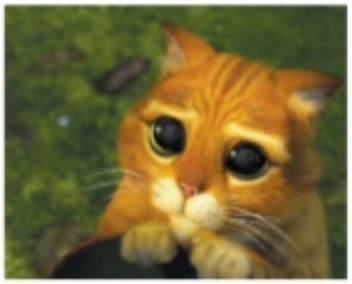
2

SURF SARA

Why automation?

pets vs. cattle

Borrowed from
@randybias at Cloudscaling
http://www.slideshare.net/randybias/the-cloud-revolution-cyber-press-forum-philippines



- Pets are given names like pussinboots.cern.ch
- They are unique, lovingly hand raised and cared for
- When they get ill, you nurse them back to health

- Cattle are given numbers like vm0042.cern.ch
- They are almost identical to other cattle
- When they get ill, you get another one

- Future application architectures should use Cattle but Pets with strong configuration management are viable and still needed

Gavin McCance, CERN 17

Description

OpenNebula

XML-RPC over http

- bindings for Java, Ruby (also Python, NodeJS…)

- **Methods** like `one.<object>.<action>`
  - e.g.: `one.vm.rename`
  - ***Pools***, like: `one.vmpool.info`

- **Parameters**, position-based

- **Output**, a 3-tuple (A, B, C) where:
  - A: correct or error response
  - B: returned info (if correct);
    error message (if error)
  - C: numeric error code

Operate/query on:

- Images
- Templates
- Virtual Machines
- Quotas
- …

Demo

3

SURF SARA

Example (I)

List my VMs

```
class VmList:
    """A simple list of my VMs"""
    ONE_ENDPOINT = 'http://ui.hpccloud.surfsara.nl:2633/RPC2'
    ONE_USER = 'username'  # replace this with yours
    ONE_PASS = 'pass'      # replace this with yours
    def __init__(self):
        self.client = oca.Client(
            self.ONE_USER + ':' + self.ONE_PASS, self.ONE_ENDPOINT)

    def fetch_vms(self):
        xml_string = self.client.call('vmpool.info', -3, -1, -1, -2)
        root = ET.fromstring(xml_string)
        return root

if __name__ == '__main__':
    xml = VmList()
        .fetch_vms()
    print(XmlUtil
        .prettify(xml))
```

one.vmpool.info

- Description: Retrieves information for all or part of the VMs in the pool.
- Parameters

Type	Data Type	Description
IN	String	The session string.
IN	Int	Filter flag - < = -3: Connected user's resources - -2: All resources - -1: Connected user's and his group's resources - > = 0: UID User's Resources
IN	Int	When the next parameter is >= -1 this is the Range start ID. Can be -1. For smaller values this is the offset used for pagination.
IN	Int	For values >= -1 this is the Range end ID. Can be -1 to get until the last ID. For values < -1 this is the page size used for pagination.
IN	Int	VM state to filter by.

Example (and II)

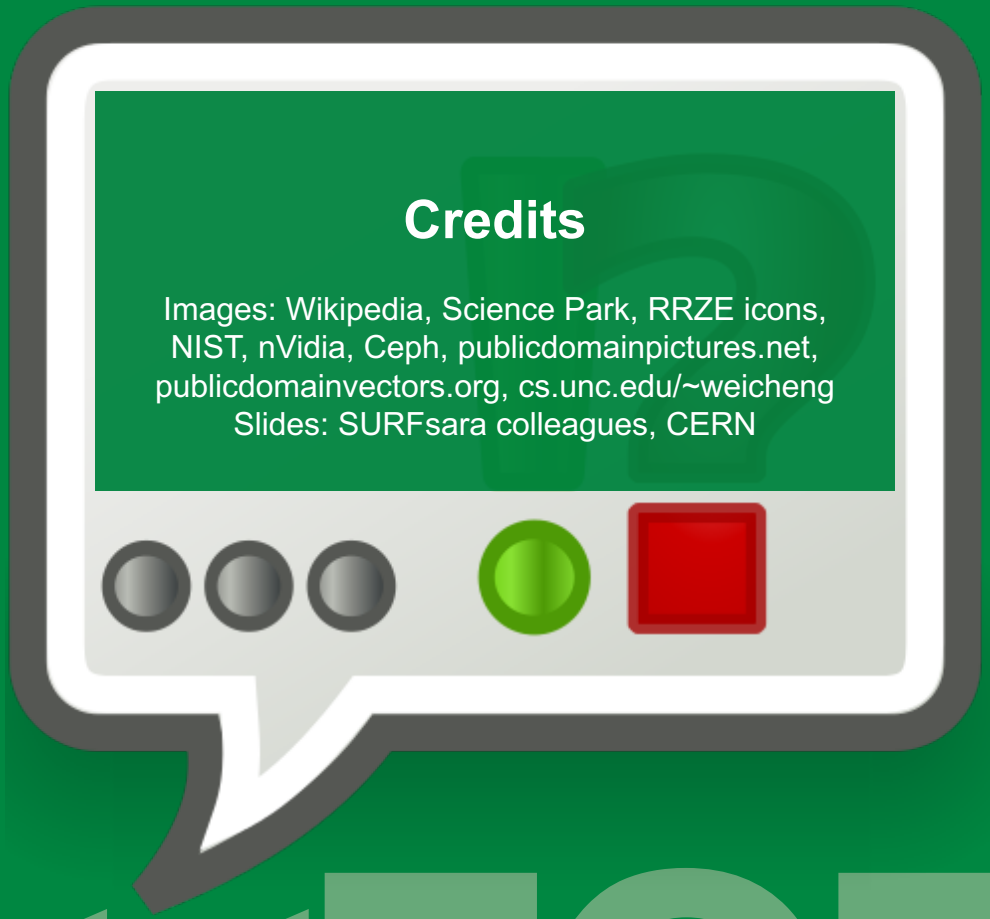
List my VMs (output)

```
<VM_POOL>
  <VM>
    <ID>164</ID>
    <UID>247</UID>
    <GID>108</GID>
    <UNAME>ander</UNAME>
    <GNAME>workshop</GNAME>
    <NAME>Ubuntu-15.04</NAME>
    …
    <LCM_STATE>3</LCM_STATE>
    <TEMPLATE>
      <CPU>…</CPU>
      ...
    </TEMPLATE>
  </VM>
  <VM>…</VM>
  ...
</VM_POOL>
```

Request: https://e-infra.surfsara.nl

UI: https://ui.hpccloud.surfsara.nl

Doc: https://doc.hpccloud.surfsara.nl



Ander Astudillo
<ander.astudillo@surfsara.nl>

Markus van Dijk
<markus.vandijk@surfsara.nl>

<<EOF

SURF SARA

Appendix

SURF SARA

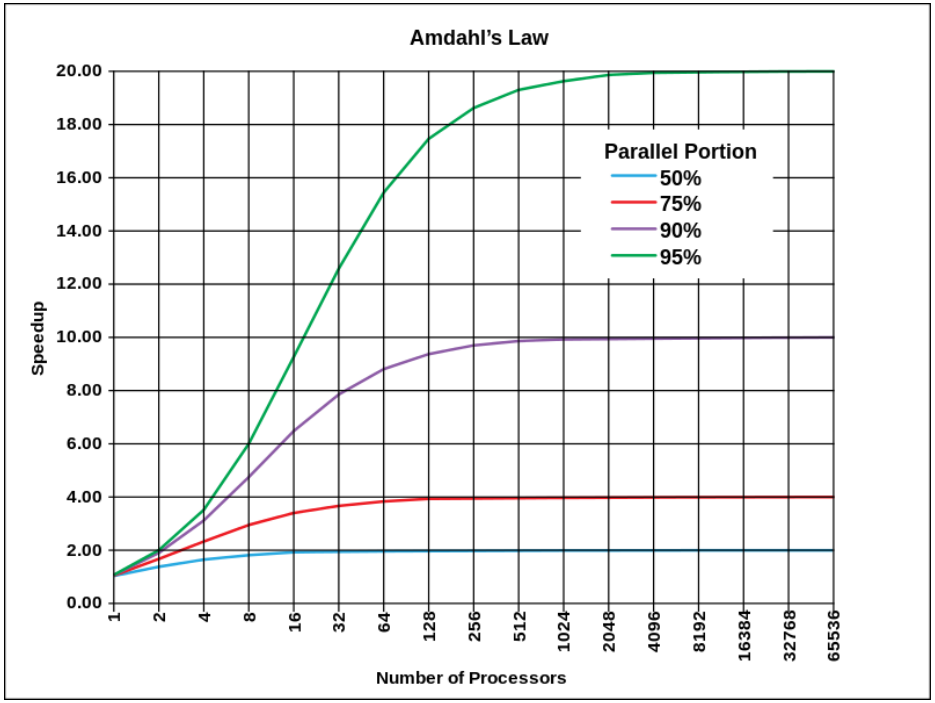
Amdahl's law

$$T(s) = (1-p)T + \frac{p}{s}T.$$

$$S_{\text{latency}}(s) = \frac{TW}{T(s)W} = \frac{T}{T(s)} = \frac{1}{1-p+\frac{p}{s}}.$$

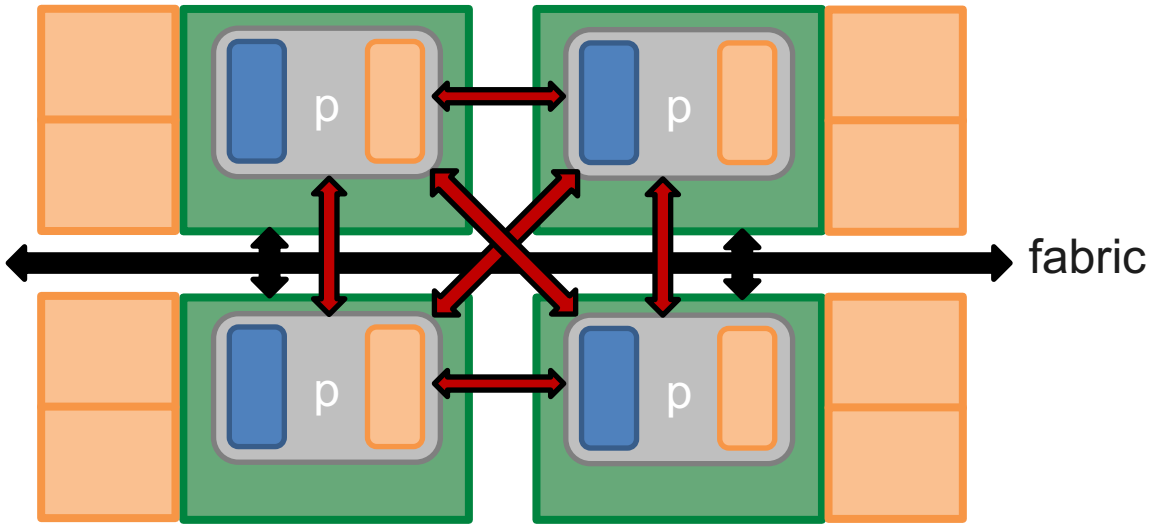
- *T(s)*: running time after an improvement of *s*
- *s*: speedup factor of parallel part
- *p*: % of the program that is parallel
- *T*: original running time
- *W*: fixed workload

It's mainly the **algorithm** that defines speedup; rather than the amount of processors



Speedup is limited by the serial part of the program. E.g., if 95% of the program can be parallelised, the theoretical maximum speedup using parallel computing would be 20 times.

NUMA



Processes **communicate** over the fabric