Medusa
An Architecture for Distributed Multimedia

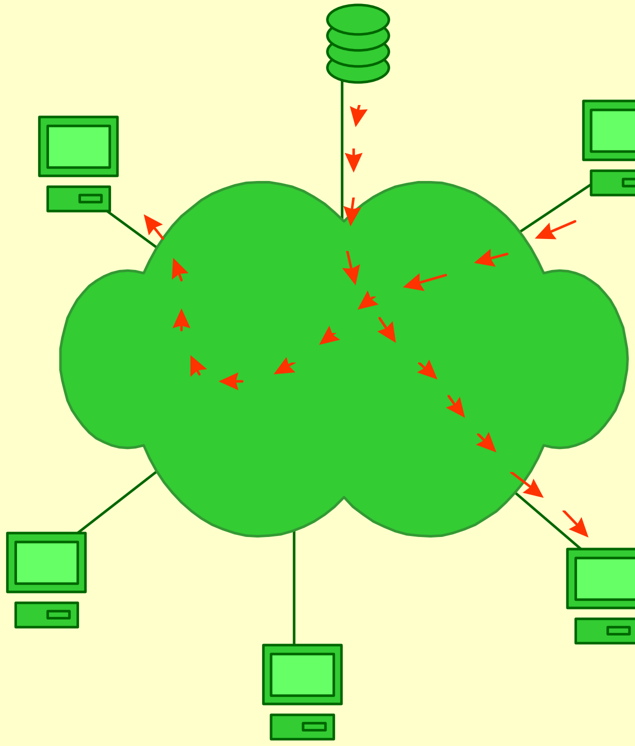
Frank Stajano

`fstajano@orl.co.uk`

THE OLIVETTI & ORACLE RESEARCH LABORATORY

http://www.orl.co.uk/

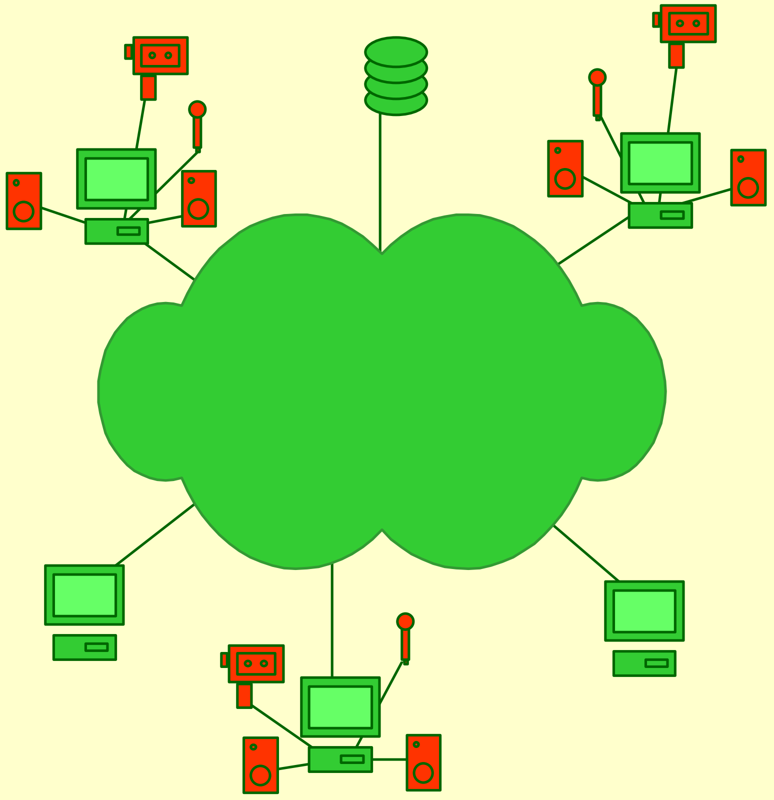
What is Medusa?



Medusa is a hardware and software architecture for distributed multimedia.

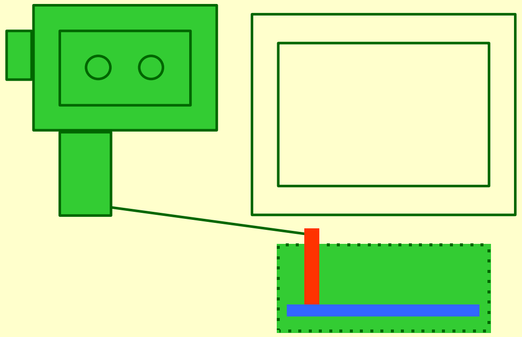
The scenario is one in which various devices, connected to a network, exchange streams of continuous media.

First-generation networked multimedia

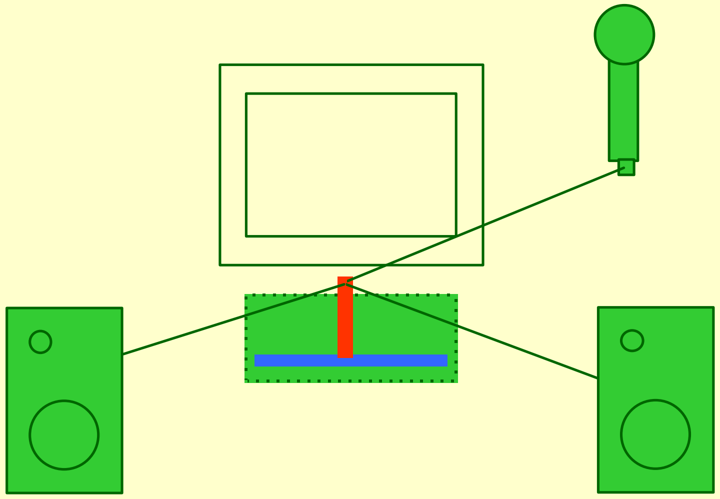


The traditional approach to this problem has been to deploy computers on the network and to attach multimedia peripherals to them.

Attaching multimedia peripherals to computers

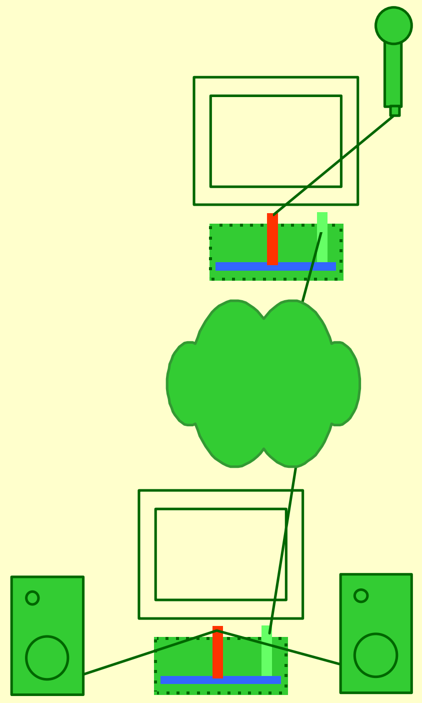


When a computer is equipped with a camera, the video grabber card which digitises the camera's analogue signal is plugged into the bus of the computer.



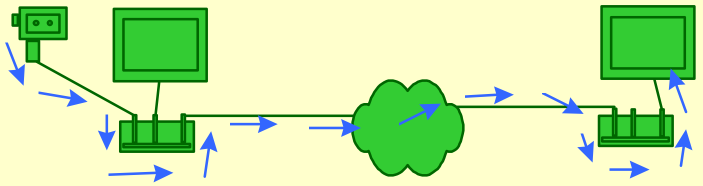
The microphone and speaker are traditionally dealt with in the same way: transducers are attached to an audio card which plugs into the computer's bus.

A convoluted path for multimedia streams



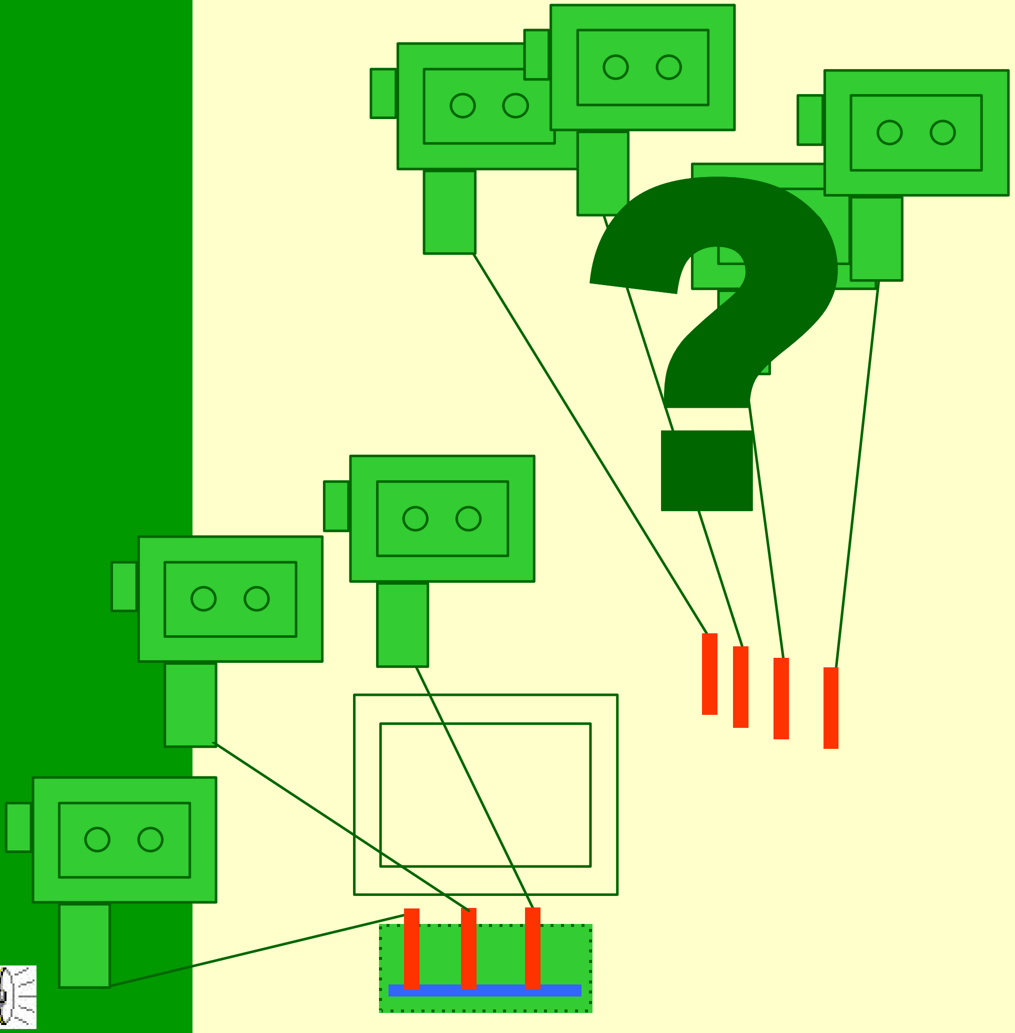
What happens when you establish an audio connection between two workstations?

The audio signal from the microphone is digitised by the audio card and routed through the bus to the network adapter of the first computer. Then it travels through the network, reaches the network card of the second computer, gets onto the bus and is picked up by the audio card, where it is transformed back to analogue format and sent to the speakers.



A video stream follows a similar route: from transducer to digitiser to bus to network to bus to display.

A strategy lacking scalability



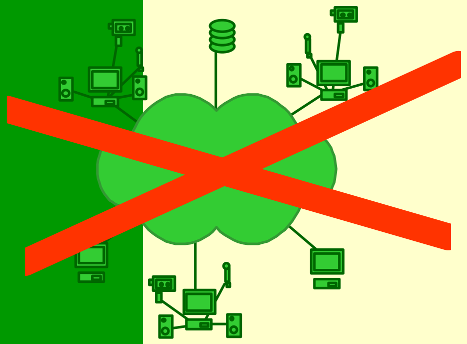
If we wanted to equip our workstation not with one but with four, eight or sixteen cameras, we would see that this system does not scale very well.

For a start, we would run out of slots to put video grabbers in.

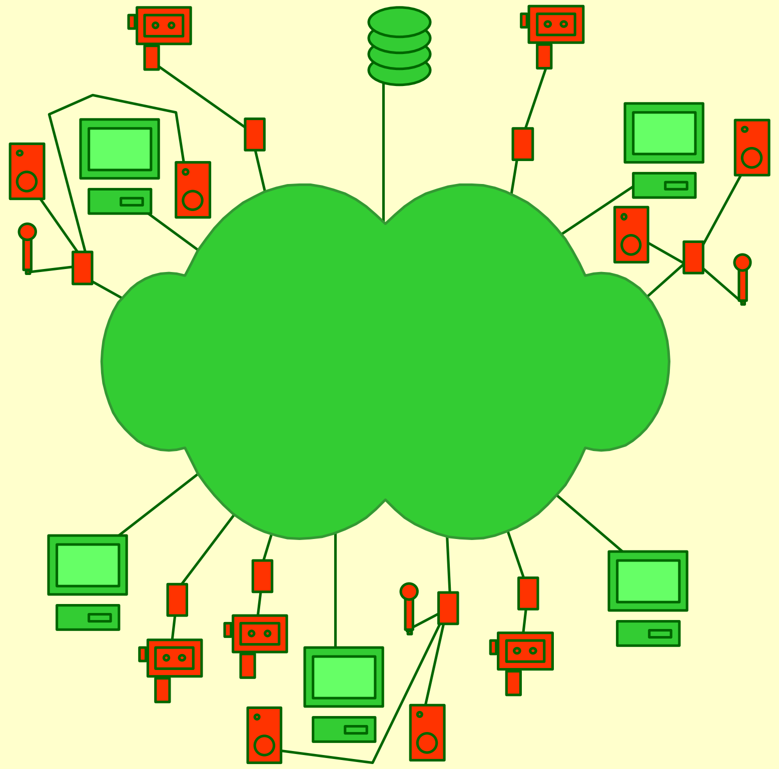
Secondly, assuming we had enough slots, we would run out of bandwidth on the computer's bus.

Even in a lightly loaded case with only one camera and one microphone, we may find the continuous media streams on the computer bus are a cause and a target of interference with the rest of the applications running on the machine.

Medusa's "network endpoints" philosophy

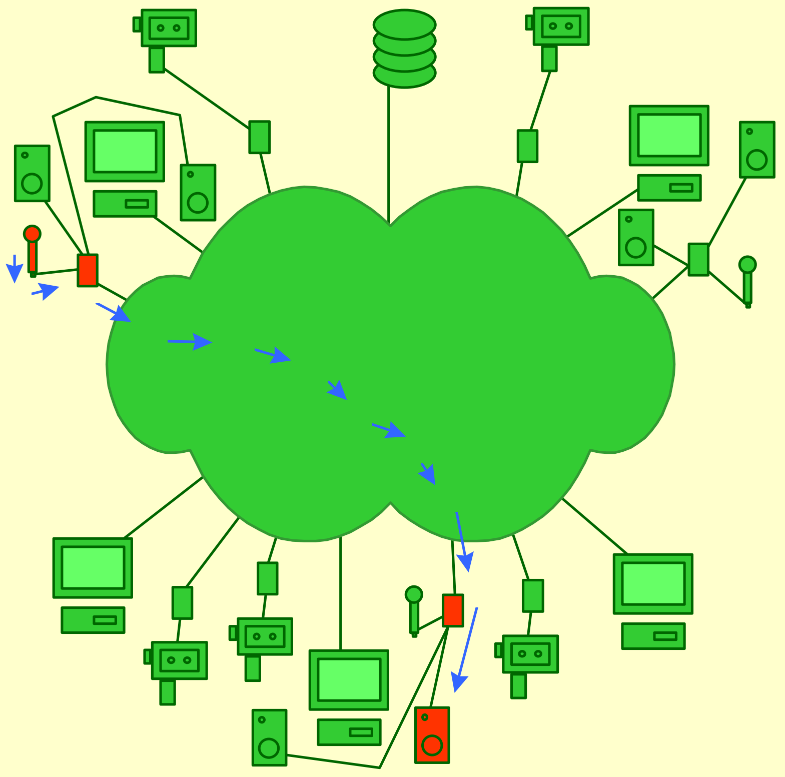


In Medusa we solve this problem with a new approach: the multimedia transducers such as cameras, microphones and speakers are no longer peripherals of the computer.



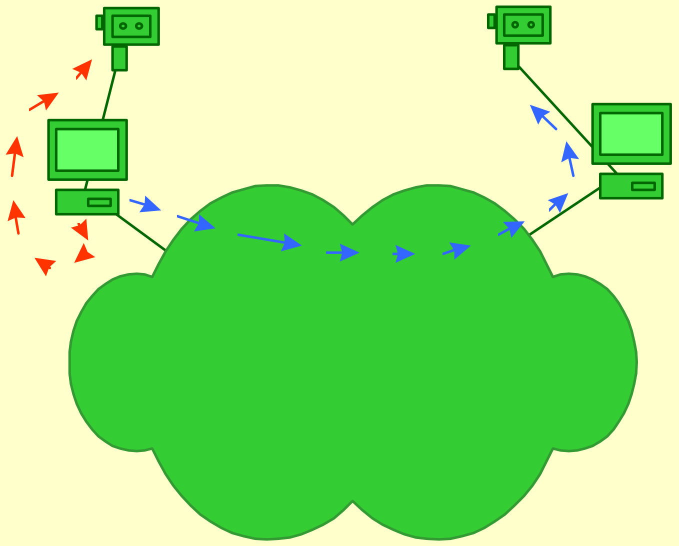
They become computers in their own right, with their own processor and network interface. We call them network endpoints. They no longer plug into the bus of a computer: instead, they connect to the network directly.

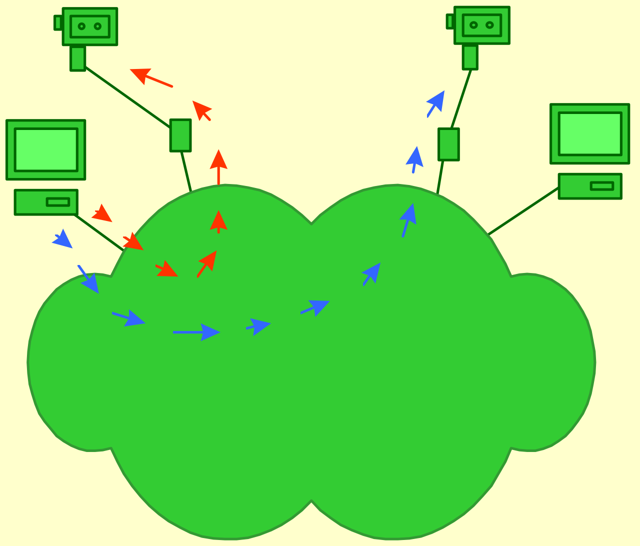
A simpler path



The path followed by the audio goes directly from a microphone on the source audio endpoint, via the network, to the speaker of the sink audio endpoint, without having to enter a conventional computer; so the audio doesn't slow down the applications and is not interrupted by them.

A more uniform software API

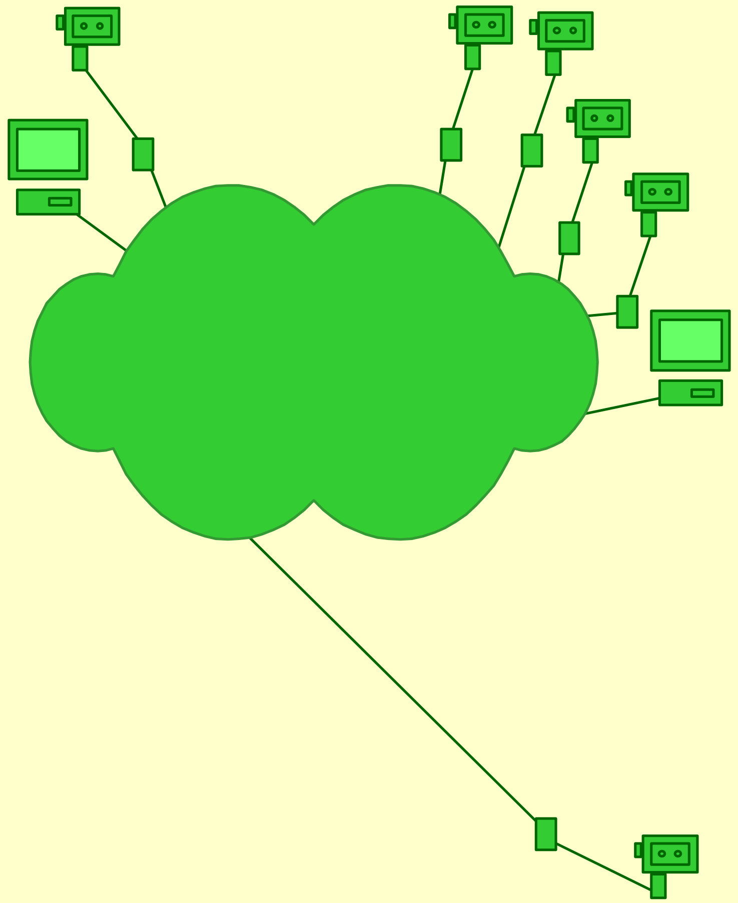




As an added advantage, local and remote peripherals all look alike to the software that drives them.

In contrast to the previous case, where the local peripheral had a privileged status and needed to be treated differently, everything here is on the network and is accessed in the same uniform way.

A flexible, scalable architecture

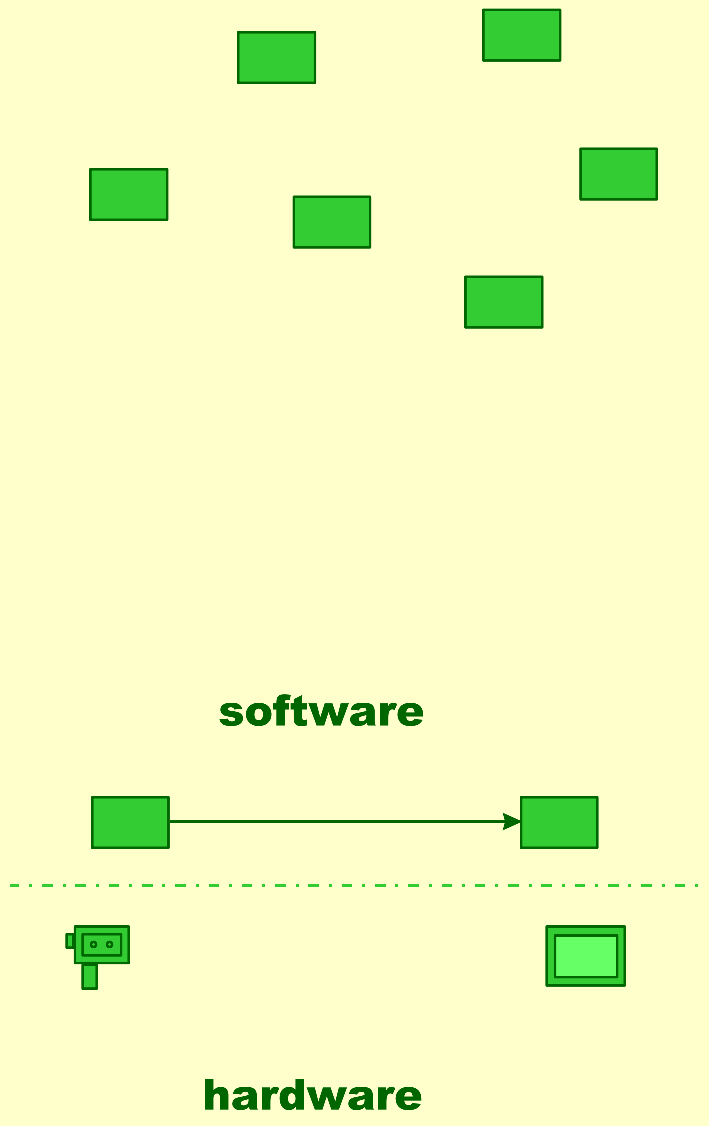


This architecture can be scaled very efficiently and flexibly.

A computer can have one camera, or four, or ten with the same ease. This is because these cameras are not peripherals of the computer, but just other stand-alone computers that happen to be located nearby.

You can also imagine locations where you have cameras without any computers at all, for example a smart doorbell.

Medusa software: a distributed, modular approach

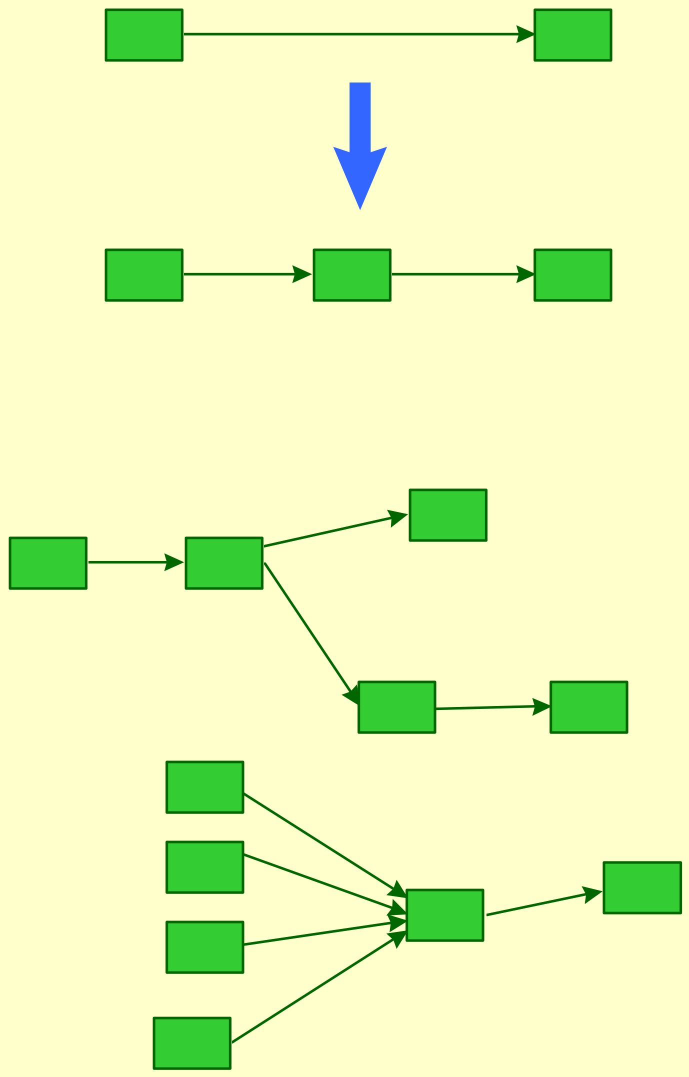


The software side of Medusa is based on software objects called modules.

Modules are the active objects that work as sources, processors and sinks of multimedia data, and everything of interest happens inside a module.

A video source module is a software object that produces frames of video. A video sink object accepts frames of video and it may display them on a screen. Between them, data flows on a demand-driven connection.

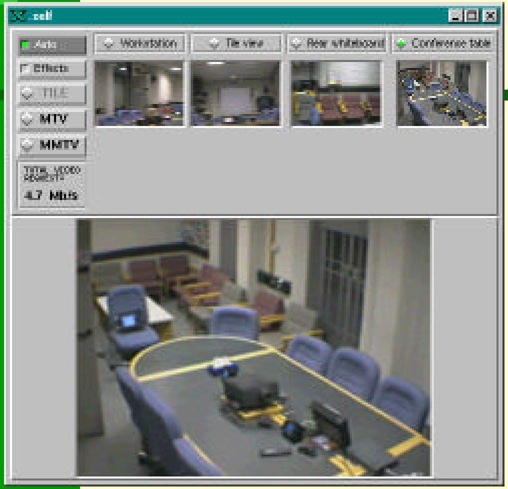
Replumbing modules with parts from a library

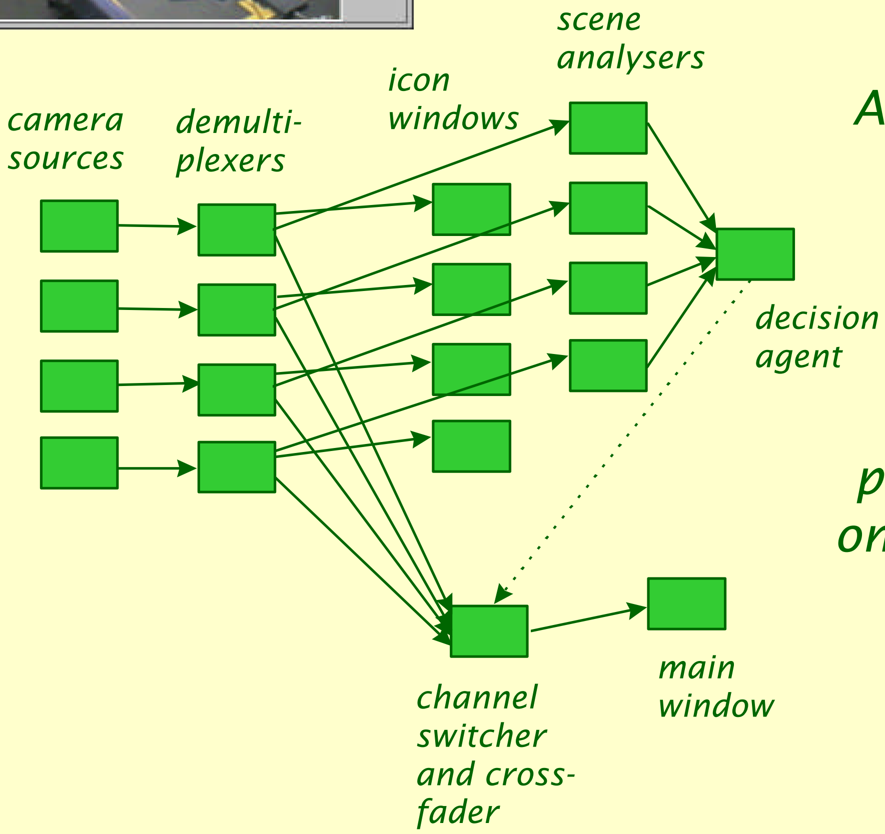


You can insert another module between these two, for example one that changes the brightness and contrast.

Other available modules detect the average brightness of the scene; split the video stream and send it to several sinks in parallel; switch the video stream and only send it to one output at a time; add subtitles; or track moving objects in the scene.

Complex networks of modules



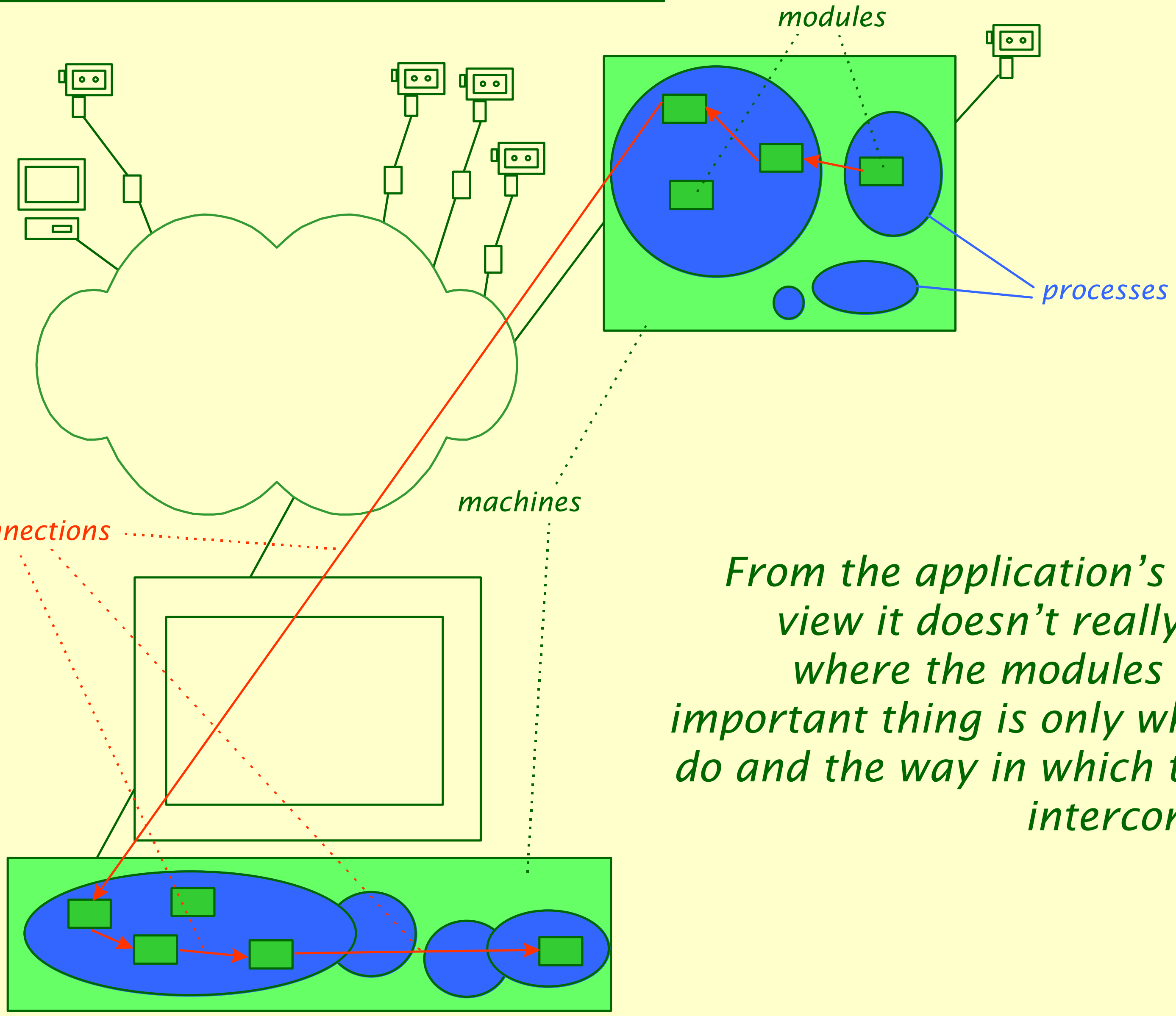


Of course you can combine these in a web of modules to build complex applications.

A typical Medusa application, like a multi-stream video phone, contains about a hundred modules.

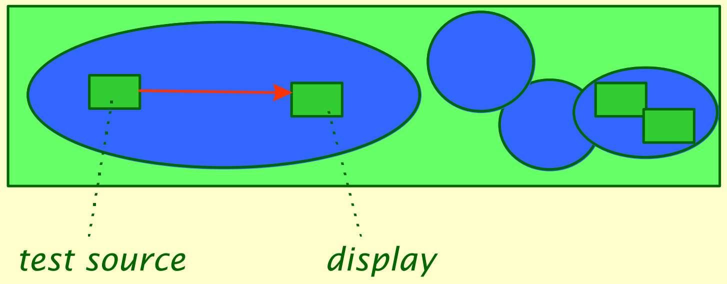
This is because there are many parallel video pipelines, including ones to scene analysers which can automatically cross-fade to the active camera.

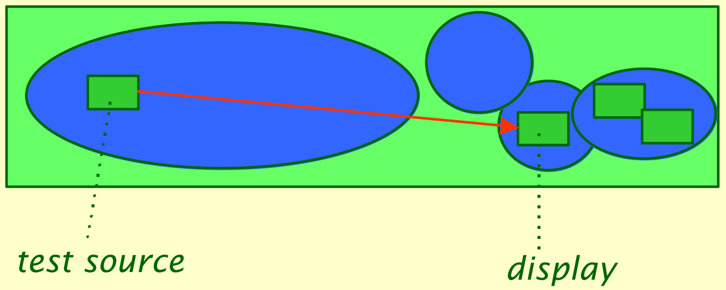
An entirely distributed approach



From the application's point of view it doesn't really matter where the modules are: the important thing is only what they do and the way in which they are interconnected.

Network-transparency of module connections





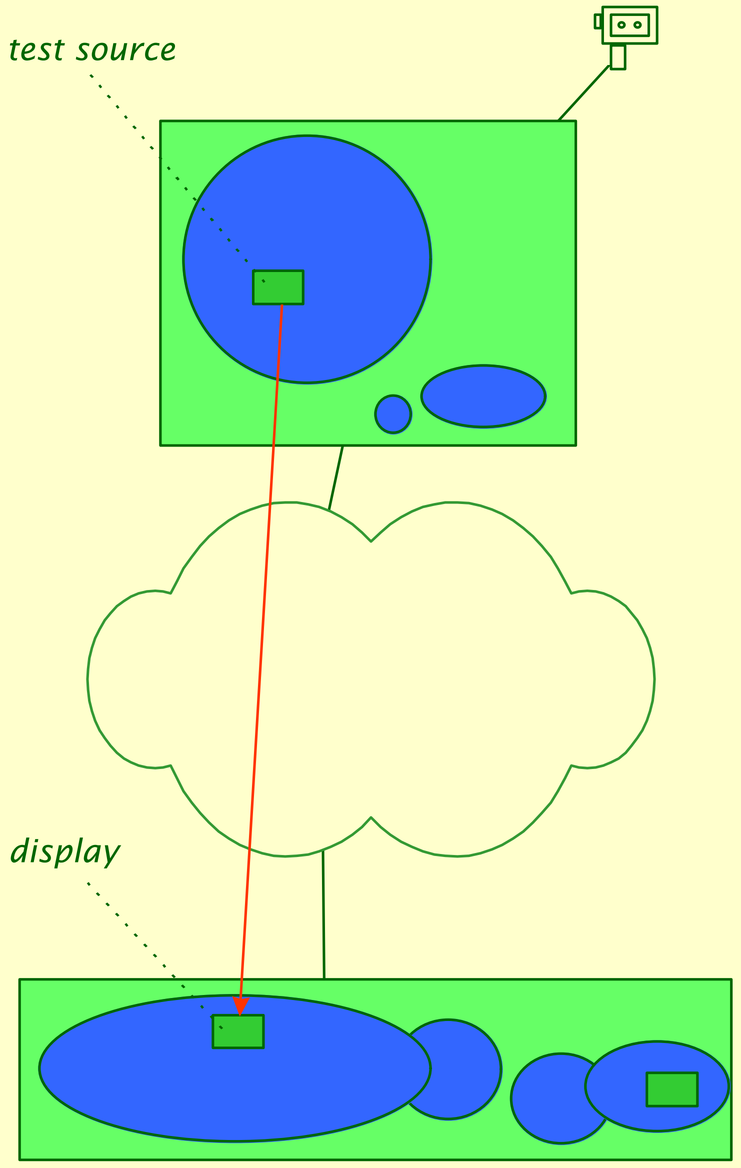
You can have a simple test program where a test source generates frames and sends them to a display module.

Both modules could live inside the same process on the same machine.

But it is also possible for the test source to be running inside a different process from the display module, though they would still be connected with a Medusa connection.

This connection would look exactly like the previous one to the application using the modules, but its implementation would be different and would involve inter-process communication.

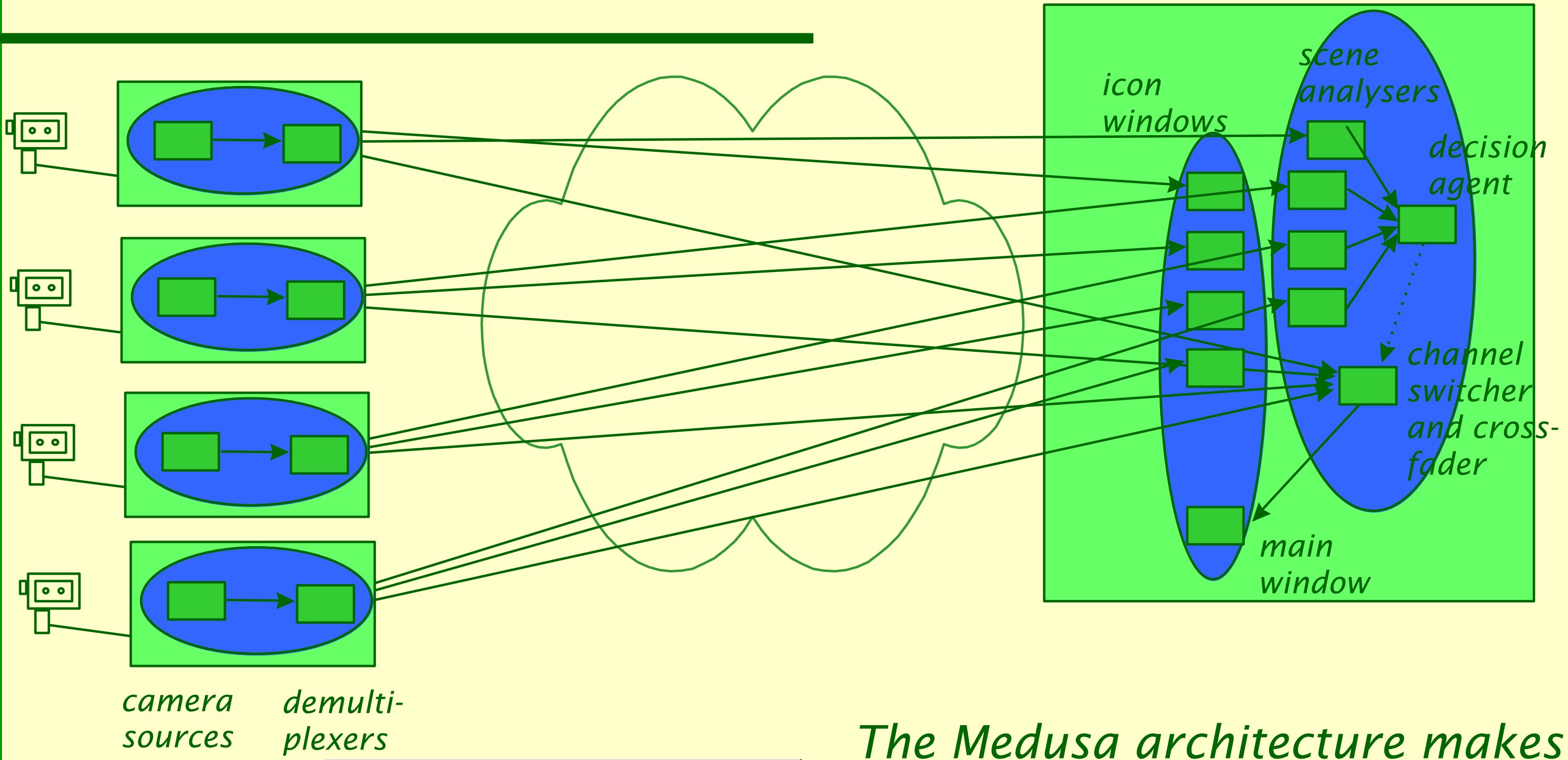
Connections spanning the network

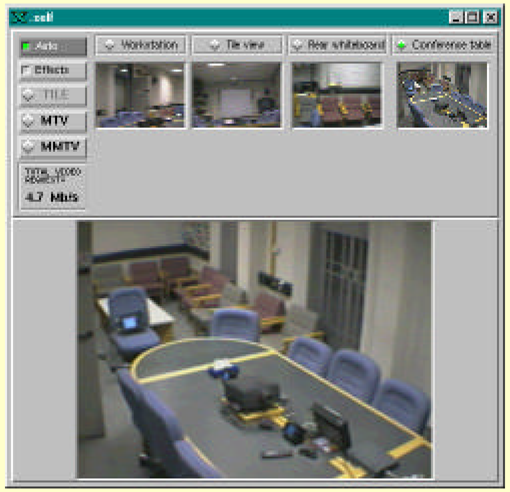


The source could be even running on a different machine on the network: in fact, the real camera module runs on the camera endpoint.

However, at the application level, the connection to the display module looks exactly the same.

An easier approach to distributed multimedia





The Medusa architecture makes distributed programming easy because modules are instantiated and connected to each other in exactly the same way, independent of whether jumps across the network are involved or not.